PHP Cookbook

第二版
涵盖PHP 5

PHP 经典实例

David Sklar & Adam Trachtenberg 著
李松峰 秦绪文 李丽 译

O'REILLY®
中国电力出版社

PHP经典实例

第二版

PHP经典实例

David Sklar and Adam Trachtenberg 著
李松峰 秦绪文 李丽 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc.授权中国电力出版社出版

中国电力出版社

www.TopSage.com

图书在版编目（CIP）数据

PHP经典实例（第二版）/（美）斯克拉（Sklar, D.）；（美）切贝特伯格（Tracbtenberg, A.）著；李松峰、秦绪文、李丽译. －北京：中国电力出版社，2009
书名原文：PHP Cookbook, second edition
ISBN 978-7-5083-8608-9

I. P... II. ①斯... ②切... ③李... ④秦... ⑤李... III. PHP语言－程序设计 IV. TP312
中国版本图书馆CIP数据核字（2009）第037430号

北京市版权局著作权合同登记
图字：01-2006-7453号

书　　名/ PHP经典实例（第二版）
书　　号/ ISBN 978-7-5083-8608-9
责任编辑/ 马首鳌
封面设计/ Karen Montgomery，张健
出版发行/ 中国电力出版社（www.infopower.com.cn）
地　　址/ 北京三里河路6号（邮政编码100044）
经　　销/ 全国新华书店
印　　刷/ 汇鑫印务有限公司
开　　本/ 787毫米×980毫米　16开本　52.5印张　999千字
版　　次/ 2009年10月第一版　2009年10月第一次印刷
印　　数/ 0001－3000册
定　　价/ 98.00元（册）

O'Reilly Media, Inc.介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构O'Reilly Media, Inc.授权中国电力出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc.是世界上在 Unix、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时也是联机出版的先锋。

从最畅销的*The Whole Internet User's Guide & Catalog*（被纽约公共图书馆评为20世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc.一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc.是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc.具有深厚的计算机专业背景，这使得O'Reilly Media, Inc.形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc.所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc.还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc.依靠他们及时地推出图书。因为O'Reilly Media, Inc.紧密地与计算机业界联系着，所以O'Reilly Media, Inc.知道市场上真正需要什么图书。

扫一扫加关注，获取更多学习资源，

目录

计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

Java 一览无余：Java 视频教程 | Java SE | Java EE

.Net 技术精品资料下载汇总：ASP.NET 篇

.Net 技术精品资料下载汇总：C#语言篇

.Net 技术精品资料下载汇总：VB.NET 篇

撼世出击：C/C++编程语言学习资料尽收眼底 电子书+视频教程

Visual C++(VC/MFC)学习电子书及开发工具下载

Perl/CGI 脚本语言编程学习资源下载地址大全

Python 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载

数据库管理系统(DBMS)精品学习资源汇总：MySQL 篇 | SQL Server 篇 | Oracle 篇

平面设计优秀资源学习下载 | Flash 优秀资源学习下载 | 3D 动画优秀资源学习下载

最强 HTML/xHTML、CSS 精品学习资料下载汇总

最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总

UML 学习电子资下载汇总 软件设计与开发人员必备

经典 LinuxCBT 视频教程系列 Linux 快速学习视频教程一帖通

天罗地网：精品 Linux 学习资料大收集(电子书+视频教程) Linux 参考资源大系

Linux 系统管理员必备参考资料下载汇总

Linux shell、内核及系统编程精品资料下载汇总

UNIX 操作系统精品学习资料<电子书+视频>分类总汇

FreeBSD/OpenBSD/NetBSD 精品学习资源索引 含书籍+视频

Solaris/OpenSolaris 电子书、视频等精华资料下载索引

前言

PHP是数百万动态Web应用程序背后的引擎。其博大宽泛的特性集合、简捷友好的语法结构以及对不同操作系统和Web服务器的支持，已经使它成为实现敏捷Web开发和有条理地构建复杂系统的一门理想语言。

作为一种Web脚本语言，PHP之所以能够取得今天的成功，主要原因在于它最初就是作为一种新的、处理HTML表单和创建网页的工具而设计的。因此PHP与Web开发有着天然的血缘关系。而PHP丰富的外部应用程序和扩展库，又使它具有浓重的混血特征。PHP可以与多种不同的数据库对话，也能理解大量Internet协议。PHP也使得解析表单数据和进行HTTP请求更加简单。在本书中有专门以Web为主题的许多技巧和例子。

本书集合了PHP中各种常见任务的解决方案。我们也努力让书中包含从新手到专家都会感兴趣的内容。如果我们的努力没有白费的话，那么你将从本书中学到一些（或许可能是很多）东西。书中不仅为专门的PHP程序员，也为那些有着其他语言编写经验但想掌握PHP的人提供了很多有益的提示。

PHP的源代码以及二进制文件，都可以从*http://www.php.net*上免费下载。该网站中还包含安装PHP的说明、全面的文档和在线资源的链接、用户组、邮件列表以及其他与PHP相关的资源。

本书的读者对象

本书是为想要解决PHP中问题的程序员们而写。如果你还不了解PHP，最好把本书作为学习PHP的第二本书。第一本应该是同样由O'Reilly出版的《Learning PHP 5》。

如果你对PHP已经很熟悉了，那么本书会帮助你解决一些具体的问题，并且会从此伴你左右（至少是在你编程的时候）。本书还会向你展示如何在PHP中完成一些特定的任

务，比如说发送电子邮件或者编写SOAP服务器系统等你可能已经知道用其他语言如何实现的功能。而试图把应用程序从其他语言转换到PHP的程序员们也将发现，本书是他们值得信赖的朋友。

本书内容简介

我们不指望你能坐下来从头到尾看完本书（尽管如果你这样做我们会很开心！）。PHP程序员经常要面对基于宽泛主题之上的各种各样的挑战。当你需要解决的问题时，就可以翻开本书。书中的每个技巧都有完备的解释，可以为你解决问题提供超前的建议。当技巧中引用了本身范围之外的主题时，会包含指向相关技巧和其他在线及离线资源的指示。

如果你决定一次读完一整章，那简直太棒了。因为每章中的技巧通常都是按照难易程度由低到高组织的，而且许多章的结尾都会有一个示例程序将这些技巧综合运用。每章的概述部分都简要介绍了该章中所涵盖的所有内容，包括相关的背景材料，并着重指出了一些特别有意思的技巧。

本书开始的四章是有关基本数据类型的。第1章中详细介绍了像操作子字符串、处理大小写、字符串分割以及解析逗号分隔的数据等内容。第2章的内容涉及操作浮点型数、随机数、数字在不同基数之间转换以及数字格式化等方面的问题。第3章展示了如何操纵日期和时间，格式化日期和时间，处理时区和夏时制，以及在微秒级的精度上确定时间等内容。第4章中包括对数组的相关操作，如迭代、合并、反转、排序以及提取特定元素等。

接下来的三章的主题是构造程序块。第5章中介绍了PHP在变量处理方面值得注意的特性，像设置默认值、静态变量和生成对复杂数据类型的字符串表示等。第6章中的技巧是有关在PHP中如何使用函数的。包括处理参数，传递和返回变量的引用，在运行时创建函数以及变量的作用域等内容。第7章展示了PHP面向对象编程方面的能力，其中的技巧既涉及到OOP的基础，也涵盖了PHP 5中的新特性，如魔术方法、解构器、访问控制和反射等。

在数据类型和构造程序块这两个主题之后的六章专注于以Web为中心的主题。其中，第8章涵盖了cookie、头部信息、身份认证、使用查询字符串以及其他Web应用程序的基本内容。第9章讨论了处理和验证表单输入、处理多页表单、显示带有错误信息的表单，以及预防跨站点脚本和同一表单多次提交等方面的问题。第10章解释了DBM和SQL数据库之间的区别，使用PHP的PDO数据库访问抽象层等问题，还展示了如果连接数据库、

指定唯一ID值、检索数据行、更新数据、转义引号以及记录调试信息等内容。第11章介绍了PHP内置的会话模块，利用该模块可以对在你网站中浏览跳转的用户的信息保持维护。该章也重点讲解了与会话有关的一些安全问题。第12章专题讨论了XML，包括PHP 5的简单XML扩展和改进的DOM函数、使用XPath和XSLT、读写RSS与Atom新闻推送等。第13章探讨了与整合外部网站的PHP应用程序有关的主题和客户端JavaScript，如获取远程URL、清理HTML以及响应Ajax请求等内容。

下面的三章是有关网络交互的。第14章详细讨论了在代码中通过使用外部REST，SOAP或XML-RPC等服务消费 Web 服务的方方面面。第15章中则涉及到处理Web 服务方程式另一边的内容 —— 相互提供REST、SOAP或XML-RPC请求。以上两章也都讨论到了WSDL、身份认证、头部信息以及错误处理等主题。第16章介绍了网络服务其他方面的内容，例如发送电子邮件、使用LDAP和实现DNS查询等。

本书接下来的章节讨论的是有关PHP特性以及扩展等方面的内容。这些特性和扩展可以提高PHP应用程序的健壮度、安全性、亲和力以及运行效率。第17章展示的是如何创建图形，其中的技巧包括了绘制文本、直线、多边形以及曲线。第18章集中讨论了安全这一主题，比如防止会话攻击和跨站点脚本，正确地使用密码以及数据加密等。第19章中内容有助于使应用程序适应全球化的趋势，该章包括了本地化文本、日期和时间、货币、图像，也包括了使用不同字符编码的文本（如UTF-8）等例子。第20章深入讨论了错误处理、调试技术以及为代码编写测试用例等内容。第21章解释了如何比较两个函数的性能，并给出了使你的程序高效运行的建议。第22章介绍了正则表达式，包括在HTML标签中查寻文本、在正则表达式内部调用一个PHP函数，以及使用贪婪和非贪婪匹配等细节。

第23章和第24章讨论了文件系统。第23章主要讨论的是文件，包括打开及关闭文件、使用临时文件、锁定文件、发送压缩的文件以及处理文件的内容等。第24章讨论的是处理文件目录和文件元数据，该章的技巧涉及如何修改文件的权限以及所有权，移动或删除文件和处理同一个文件夹中全部文件。

最后两章的主题是关于PHP还能做什么的扩展内容。第25章包括在Web程序设计之外使用PHP的内容。其中的技巧主要是有关在命令行中使用PHP的，比如解析程序参数和读取密码等。第26章讨论的是PEAR（the PHP Extension and Application Repository）和PECL（the PHP Extension Community Library）这两个扩展的。PEAR是一个能够提供函数和PHP扩展的PHP代码的集合。PECL是一个小型的集合，它只包括以C编写的PHP扩展。我们在本书中经常会用到PEAR和PECL这两个扩展模块，而第26章则介绍了如何安装和更新它们。

其他资源

网站

PHP有着数量庞大的在线参考资料。这些资料包罗万象，从解释性的PHP手册到定期更新文章和教程的网站，应有尽有。可以说，快捷的互联网络连接着一个PHP有用文档的大型书库。下面介绍一些主要的网站：

解释性的PHP在线手册：http://www.php.net/manual

支持17种语言，该网站不仅囊括了官方的函数文档和语言特性，而且还包括很多来自用户贡献的注释。

PHP邮件列表：http://www.php.net/mailing-lists.php

众多的邮件列表涵盖了安装、编程、扩展PHP以及其他多个主题。此外，还有一个PHP邮件列表的只读Web界面*http://news.php.net/*。

PHP演示文档：http://talks.php.net

在各种会议中配发的PHP演示文档的集合。

PEAR：http://pear.php.net

PEAR自称“一个可重用PHP组件的框架和发布系统”。可以在其中找到许多有用的PHP类和示例代码。在本书第26章中有关于PEAR的更详细介绍。

PECL：http://pecl.php.net

PECL称自己为“一个PHP扩展的资料库，提供一份包含所有已知扩展和存储设施的目录，以便下载和开发PHP扩展”。可以在第26章中了解更多有关PECL的内容。

PHP.net：A Tourist's Guide: http://www.php.net/sites.php

这是一个对php.net域名之下的众多网站提供导航的页面。

PHP知识库：http://php.faqts.com

包括来自PHP社区的很多问题和答案，也包括指向其他资源的链接。

PHP DevCenter：http://www.onlamp.com/php

汇集了大量PHP文章和教程，集介绍性内容和高级主题于一身。

Planet PHP：http://www.planet-php.net

一个聚集PHP开发人员及相关人士发表的博客文章的网站。

Zend Developer Zone：http://devzone.zend.com

包含定期更新的文章、教程和代码示例。

SitePoint Blogs on PHP: http://www.sitepoint.com/blogs/category/php

探索PHP相关内容的一个不错的地方。

图书

本节所列出的图书都是对于用PHP构建应用程序非常有帮助的参考和教程。其中，多数都是具体讨论Web相关程序开发的；因此如必要，需要参考MySQL，HTML，XML和HTTP等方面的图书。

在本节的最后，我们也提供了一些对使用任何语言的程序员都非常有用的书。这些书可以助你在编程的境界中更上一层楼，因为它们可以教会你如何在一个解决问题的大框架下来思考编程的问题：

- *Learning PHP* 5 by David Sklar著（O'Reilly）。
- *Upgrading to PHP* 5 by Adam Trachtenberg 著（O'Reilly）。
- *Programming PHP* by Rasmus Lerdorf, Kevin Tatroe, and Peter MacIntyre (O'Reilly)
- *Essential PHP Tools* by David Sklar (Apress)
- *Advanced PHP Programming* by George Schlossnagle (Sams)
- *Extending and Embedding PHP* by Sara Golemon (Sams)
- *HTML and XHTML: The Definitive Guide* by Chuck Musciano and Bill Kennedy (O'Reilly)
- *Dynamic HTML: The Definitive Guide* by Danny Goodman (O'Reilly)
- *Mastering Regular Expressions* by Jeffrey E. F. Friedl (O'Reilly)
- *XML in a Nutshell* by Elliotte Rusty Harold and W. Scott Means (O'Reilly)
- *MySQL Reference Manual*, by Michael "Monty" Widenius, David Axmark, and MySQL AB (O'Reilly); also available at *http://www.mysql.com/documentation/*
- *MySQL*, by Paul DuBois (New Riders)
- *Web Security, Privacy, and Commerce* by Simson Garfinkel and Gene Spafford (O'Reilly)
- *HTTP Pocket Reference*, by Clinton Wong (O'Reilly)
- *The Practice of Programming*, by Brian W. Kernighan and Rob Pike (Addison-Wesley)
- *Programming Pearls* by Jon Louis Bentley (Addison-Wesley)

- *The Mythical Man-Month*, by Frederick P. Brooks (Addison-Wesley)

使用本书的约定

编程约定

本书中的例子是在PHP 5.1.4版本之上编写并测试通过的。示例代码可以同时运行在Unix和Windows平台 —— 文中注明的除外。在有赖于PHP 4.3.0或5.0.0版本之后才增加的特性时，我们一般都会在文中进行注释。

我们也会调用一些在尚未发布的版本（包括PHP 6）中才可能包含的特性。在这种情况下，务请认真阅读我们的代码，因为事情可能会在一个开发周期内就发生变化。

版式约定

本书使用了以下有关排版印刷的约定：

斜体（*Italic*）
: 用于文件和文件夹名、电子邮件地址和URL，也用于文中定义的术语。

等宽字体（`Constant width`）
: 用于程序代码清单，以及关键字、变量、函数、命令选项、参数、类名和出现在文中的HTML标记。

等宽粗体（**`Constant width bold`**）
: 用于标记由程序代码清单中输出的结果数据行，以及由用户键入的命令行文本。

等宽斜体（*`Constant width italic`*）
: 用于标记应该在你的程序中用实际的值来替换的项目的常规占位符。

联系方式

关于本书的批评建议和相关问题请使用如下地址与出版社联系：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）

奥莱利技术咨询（北京）有限公司

我们为本书提供了一个网页，在此页面中我们列出了勘误表、示例以及所有其他信息。读者可以登录以下网址访问该页面：

http://www.oreilly.com/catalog/9780596513986（英文版）
http://www.oreilly.com/catalog/978-7-5083-8608-9（中文版）

要对本书发表评论或咨询相关技术问题，请发电子邮件至：

bookquestions@oreilly.com

要了解我们的书籍、会议、资源中心以及O'Reilly Network的详细信息，请访问以下网址：

http://www.oreilly.com
http://www.oreilly.com.cn

书籍链接的更新，请参考本书前言提到的相关链接。

致谢

最重要的，是要感谢那些奉献出自己的时间、创意和技能使PHP能取得今天这样成就的所有人。无数志愿者们付出了难以想象的努力，得到的不仅仅是几百几千行源代码，还有完备的文档、问答式的基础组织、许许多多的扩展应用程序和库，以及遍布全球、蓬勃发展的用户社区。能把本书添加到PHP的世界中，我们感到无比激动和自豪。

同样要感谢我们的审稿人：Wez Furlong、James Nash和Mark Oglia.

感谢 Chris Shiflett 和 Clay Loveless 以及他们的重要贡献。如果不是 Chris，第18章的讨论可能不会如此充分。而如果没有 Clay，可能就不会有本书的第11章、第20章、第21章以及第26章。还要特别感谢我们不辞辛劳的编辑Tatiana Apandi。她那解决难题的缜密逻辑性和外柔内刚的强人本色，为统筹协调保证本书的顺利完成提供了不可或缺的粘合剂。如果没有 Tatiana的工作，本书大概只是一本要到2012年才能付梓的只有27页的小册子。

David Sklar

再次感谢Adam。我们在一起共事（以这样或那样的方式）已经有11年了，而共同从事与PHP有关的工作也有10年了。如果不是和他一起，我想我也不会写出这本书（除非，说心里话，是Ben Franklin，如果他还在世）。

感谢Ning的人们，为我（在其他事情之外）提供了一个通过PHP实现很多有意思事情的机会。

我要对我的父母和姐姐说：感谢你们对我坚定不移的支持和关爱，也为了当我需要通过试验来寻找技术上的结论时所做的反反复复的测试，希望你们不会把我想象成一个怪人。

感谢给予我耐心、灵感的小家伙 —— Susannah，她总是时不时地令我大吃一惊。

Adam Trachtenberg

我都不敢相信自己使用PHP已经有10个年头了。至今我还记得第一次使用这门语言时的情景，那个时候这门语言叫PHP/FI。我发现用PHP来编写Web应用程序比我之前使用的其他语言容易多了，于是我立即转变方向。而促使我下定决心的则是在把文本写入到错误日志中时，不需要再通过一些涉及到文件处理的复杂步骤，而只要把文本字符串直截了当地传递给一个名为error_log()的函数就可以了。真是天才的杰作。

我想大声地对David说，如果没有你我不会（也不可能）写出这本书。我和这本书都欠你一笔大大的人情债。

提起Ben Franklin是很无奈的。无论如何，请了解一点，我也像PHP的官方动物一样忍受着不可名状的孤独感。

感谢eBay的每一个人，感谢他们为我提供了与搭建整个eBay社区的那么多形形色色的人在一起工作的良机。

感谢我的父母、家人以及朋友们给予我的支持和鼓励。

感谢 Elizabeth Hondl。我是那么的爱你。仍然在为我的下一本书《Maritime Disaster Cookbook》而精心安排。

Clay Loveless

我想要感谢 Adam Trachtenberg、David Sklar、Tatiana Apandi和O'Reilly的全体人员，是他们使这本书的面世成为可能，也感谢他们在这一过程中把我也包括了进来。特别感谢我的妻子 Kendra和我的儿子 Wade，感谢她们给了我时间参与。

Chris Shiflett

感谢 Adam 和 David 写了这么好的一本书。也感谢他们给了我这个撰稿的机会。

计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

Java 一览无余：Java 视频教程 | Java SE | Java EE

.Net 技术精品资料下载汇总：ASP.NET 篇

.Net 技术精品资料下载汇总：C#语言篇

.Net 技术精品资料下载汇总：VB.NET 篇

撼世出击：C/C++编程语言学习资料尽收眼底 电子书+视频教程

Visual C++(VC/MFC)学习电子书及开发工具下载

Perl/CGI 脚本语言编程学习资源下载地址大全

Python 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载

数据库管理系统(DBMS)精品学习资源汇总：MySQL 篇 | SQL Server 篇 | Oracle 篇

平面设计优秀资源学习下载 | Flash 优秀资源学习下载 | 3D 动画优秀资源学习下载

最强 HTML/xHTML、CSS 精品学习资料下载汇总

最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总

UML 学习电子资下载汇总 软件设计与开发人员必备

经典 LinuxCBT 视频教程系列 Linux 快速学习视频教程一帖通

天罗地网：精品 Linux 学习资料大收集(电子书+视频教程) Linux 参考资源大系

Linux 系统管理员必备参考资料下载汇总

Linux shell、内核及系统编程精品资料下载汇总

UNIX 操作系统精品学习资料<电子书+视频>分类总汇

FreeBSD/OpenBSD/NetBSD 精品学习资源索引 含书籍+视频

Solaris/OpenSolaris 电子书、视频等精华资料下载索引

第1章

字符串

1.0 概述

PHP中的字符串指的是字符的序列，例如："We hold these truths to be self-evident"，或"Once upon a time"，甚至"111211211"。当我们从某个文件中读取数据或者把数据输出到Web浏览器的时候，其中的数据就表现为字符串。

PHP字符串是二进制安全的（例如，字符串中可以包含空字节）而且可以随意加长或者缩短。对字符串大小的唯一限制就是PHP可用的内存数量。

警告：通常情况下，PHP字符串是ASCII字符串。对于像UTF-8或者其他多字节的字符编码这样一些非ASCII数据，则必须做一些额外的工作，参见第19章。

与Perl及Unix命令解释程序（Unix shell）中的字符串在形式和行为方面类似，PHP字符串可以通过三种方式来初始化：单引号、双引号和使用"here document（heredoc）"形式。在使用单引号字符串时，字符串中需要转义的特殊字符只有反斜杠和单引号本身。例1-1中是四个单引号字符串。

例1-1：单引号字符串

```
print 'I have gone to the store.';
print 'I\'ve gone to the store.';
print 'Would you pay $1.75 for 8 ounces of tap water?';
print 'In double-quoted strings, newline is represented by \n';
```

例1-1的输出结果：

```
I have gone to the store.
I've gone to the store.
Would you pay $1.75 for 8 ounces of tap water?
In double-quoted strings, newline is represented by \n
```

因为PHP不会检查单引号字符串中的插入变量及任何转义序列，所以用这种方式定义字符串不仅直观而且速度快。

双引号字符串虽然不能识别转义的单引号，但是却能够识别插入的变量和表1-1中的转义序列。

表1-1：双引号字符串转义序列

转义序列	字符
\n	换行符（ASCII码10）
\r	回车符（ASCII码13）
\t	制表符（ASCII码9）
\\	反斜杠
\$	美元符号
\"	双引号
\0至\777	八进制数值
\x0至\xFF	十六进制数值

例1-2：双引号字符串

```
print "I've gone to the store.";
print "The sauce cost \$10.25.";
$cost = '$10.25';
print "The sauce cost $cost.";
print "The sauce cost \$\061\060.\x32\x35.";
```

例1-2的输出结果：

```
I've gone to the store.
The sauce cost $10.25.
The sauce cost $10.25.
The sauce cost $10.25.
```

例1-2中最后一行输出的酱油价格是正确的，因为字符1在ASCII字符集中用十进制数49和八进制数061表示；字符0在ASCII字符集中用十进制数48和八进制数060表示；而2对应的是ASCII字符集中的十进制数50和十六进制数32；5对应的是ASCII字符集中的十进制数53和十六进制数35。

由heredoc定义的字符串可以识别所有的插入变量以及双引号字符串能够识别的转义序列，却不要求对双引号进行转义。Heredoc以符号<<<加一个记号（不能使用空行或者带有空格后缀）来定义字符串的开始，并以该记号后跟一个分号（如必要的话）标识字符串的结尾，以结束heredoc定义。例1-3显示了如何用heredoc定义字符串。

例1-3：用heredoc定义多行字符串

```
print <<< END
It's funny when signs say things like:
   Original "Root" Beer
   "Free" Gift
   Shoes cleaned while "you" wait
or have other misquoted words.
END;
```

例1-3的输出结果：

```
It's funny when signs say things like:
   Original "Root" Beer
   "Free" Gift
   Shoes cleaned while "you" wait
or have other misquoted words.
```

在heredoc定义的字符串中，保留了所有换行符、空格符以及引号。按照约定，字符串结束标识符通常全部大写，而且是区分大小写的。例1-4显示了另外两个有效的heredoc定义。

例1-4：更多heredoc定义的例子

```
print <<< PARSLEY
It's easy to grow fresh:
Parsley
Chives
on your windowsill
PARSLEY;

print <<< DOGS
If you like pets, yell out:
DOGS AND CATS ARE GREAT!
DOGS;
```

在输出带有插入变量的HTML时，heredoc方式特别有用，因为不需要转义HTML元素中的双引号。例1-5使用了一个heredoc来输出HTML。

例1-5：用heredoc定义的方式来输出HTML

```
if ($remaining_cards > 0) {
    $url = '/deal.php';
    $text = 'Deal More Cards';
} else {
    $url = '/new-game.php';
    $text = 'Start a New Game';
}
print <<< HTML
There are <b>$remaining_cards</b> left.
<p>
<a href="$url">$text</a>
HTML;
```

在例1-5中，字符串结尾标识符后面需要跟一个分号来告诉PHP语句已经结束。不过在某些情况下，则不必在这里使用分号。例1-6所示就是一种情况，其中用到了一个带有串连接操作符的heredoc定义。

例1-6：连接一个heredoc定义的字符串

```
$html = <<< END
<div class="$divClass">
<ul class="$ulClass">
<li>
END
. $listItem . '</li></div>';

print $html;
```

如果为变量$divClass、$ulClass和$listItem假定一些合理的值，例1-6的输出结果可能是：

```
<div class="class1">
<ul class="class2">
<li> The List Item </li></div>
```

在例1-6中，表达式需要延续到下一行，所以不必使用分号。同时，还要注意的是为了让PHP识别出字符串结尾的标识符，后面的字符串连接操作符，需要与字符串结尾标识符分别放在两行中。

字符串中的个别位置上的字符可以通过方括号来引用。字符串中第一个字符的索引值为0。例1-7从一个字符串中取出一个字符。

例1-7：取得字符串中个别字符

```
$neighbor = 'Hilda';
print $neighbor [3];
```

例1-7的输出结果：

```
d
```

也可以通过大括号来取得一个字符串中个别位置上的字符。也就是说，$neighbor{3}与$neighbor[3]的结果是相同的。使用大括号的语法是不久前刚加入到PHP中的。这样就能直观地分辨出字符串索引和数组索引。

1.1 访问子字符串

问题

你想知道一个字符串中是否包含着一个特殊的子字符串。例如，你想找出包含@的电子邮件地址。

方案

使用strpos()，如例1-8所示。

例1-8：用strpos()来查找子字符串

```
<?php

if (strpos($_POST['email'], '@') === false) {
    print 'There was no @ in the e-mail address!';
 }

?>
```

讨论

由strops()返回的值，是在这个字符串（“大海”）中找到的子字符串（要捞的“针”）的起始位置。如果在这个字符串中没有找到相应的子字符串，strpos()就返回false。如果子字符串位于这个字符串的开始处，strpos()返回0，因为位置0表示这个字符串的开始。为了区分返回的0和false值，必须使用等同操作符（===）或者不等同操作符（!==），而不是使用常规的相等操作符（==）或者不等操作符（!=）。例1-8就使用了===来比较strpos()的返回值与false是否等同。这一测试只有当strpos()返回false而不是0或其他值时才是成功的。

参见

strops()函数的文档（*http://www.php.net/strpos*）。

1.2 提取子字符串

问题

你想从字符串的一个特定的位置开始，提取字符串的一部分。例如，你想取得表单中用户名的前八个字符。

方案

使用substr()提取子字符串，如例1-9所示。

例1-9：用substr()来提取一个子字符串

```
<?php
$substring = substr($string,$start,$length);
$username = substr($_GET['username'],0,8);
?>
```

讨论

如果$start和$length是正值，substr()返回字符串中的$length个字符，字符串中第一个字符的位置为0。例1-10中$start和$length都是正值。

例1-10：$start和$length都是正值。

```
print substr('watch out for that tree',6,5);
```

例1-10的输出结果：

```
out f
```

如果略去$length，substr()会返回原始字符串中从位置$start开始到字符串结尾处的子字符串，如例1-11所示。

例1-11：$start为正值，但未指定长度值

```
print substr('watch out for that tree',17);
```

例1-11的输出结果：

```
t tree
```

如果$start的值大于字符串的长度，substr()返回false。

如果$start加$length超过了字符串的结尾，substr()返回从位置$start开始至字符串结尾的所有字符，如例1-12所示。

例1-12：指定的长度超过了字符串的结尾

```
print substr('watch out for that tree',20,5);
```

例1-12的输出结果：

```
ree
```

如果$start是负值，substr()会从这个字符串的结尾处开始反向推算，来确定要返回的子字符串的开始位置，如例1-13所示。

例1-13：使用负的起始位置

```
print substr('watch out for that tree',-6);
print substr('watch out for that tree',-17,5);
```

例1-13的输出结果：

```
t tree
out f
```

当一个负的$start值超过了这个字符串的开始位置时（例如，如果对于长度为20的字符串设置的$start是-27），substr()将$start的值视为0。

如果$length是负值，substr()会从这个字符串的结尾处反向推算，来确定要返回的子字符串的结尾位置，如例1-14所示。

例1-14：使用负的长度值

```
print substr('watch out for that tree',15,-2);
print substr('watch out for that tree',-4,-1);
```

例1-14的输出结果：

```
hat tr
tre
```

参见

substr()函数的文档（*http://www.php.net/substr*）。

1.3 替换子字符串

问题

你想用另一个字符串来替换一个子字符串。例如，你想在打印之前，对一个信用卡号码的后四位之前的部分进行模糊处理。

方案

使用substr_replace()，如例1-15所示。

例1-15：用substr_replace()替换子字符串。

```
// 把从位置$start开始到$old_string结尾处的所有字符
// 替换成$new_substring
$new_string = substr_replace($old_string,$new_substring,$start);

// 把从$start位置开始的$length个字符替换成$new_substring
$new_string = substr_replace($old_string,$new_substring,$start,$length);
```

讨论

在没有指定参数$length的情况下，substr_replace()会替换从位置$start开始到这个字符串结尾处的所有字符。如果指定了$length，只替换指定长度的字符：

```
print substr_replace('My pet is a blue dog.','fish.',12);
print substr_replace('My pet is a blue dog.','green',12,4);
$credit_card = '4111 1111 1111 1111';
print substr_replace($credit_card,'xxxx ',0,strlen($credit_card)-4);

My pet is a fish.
My pet is a green dog.
xxxx 1111
```

如果$start是负值，则新子字符串开始替换的位置通过从$old_string的结尾处反向推算来确定，而不是以开始处为起点计算：

```
print substr_replace('My pet is a blue dog.','fish.',-9);
print substr_replace('My pet is a blue dog.','green',-9,4);

My pet is a fish.
My pet is a green dog.
```

如果$start和$length全都是0，新子字符串则会被插入到$old_string的开始位置：

```
print substr_replace('My pet is a blue dog.','Title: ',0,0);

Title: My pet is a blue dog.
```

如果你有一个无法一次显示的长文本，并想要将其大部分内容通过链接来显示时，substr_replace()函数就能派上用场了。例1-16显示了一条信息的前25个字符，然后通过带有省略号的链接指向显示更多文本的页面。

例1-16：用一个带省略号的链接显示更多文本内容

```
$r = mysql_query("SELECT id,message FROM messages WHERE id = $id") or die();
$ob = mysql_fetch_object($r);
printf('<a href="more-text.php?id=%d">%s</a>',
       $ob->id, substr_replace($ob->message,' ...',25));
```

例1-16中引用的*more-text.php*页面，可以根据在查询字符串中传递过来的信息ID取得全部文本内容并显示出来。

参见

substr_replace()函数的文档（*http://www.php.net/substr-replace*）。

1.4 逐字节处理字符串

问题

你需要对字符串中的每一个字节分别进行处理。

方案

利用for循环遍历字符串的每一个字节。例1-17用于计算一个字符串中包含的元音字母的数目。

例1-17：处理字符串中每一字节

```
<?php
$string = "This weekend, I'm going shopping for a pet chicken.";
$vowels = 0;
for ($i = 0, $j = strlen($string); $i < $j; $i++) {
    if (strstr('aeiouAEIOU',$string[$i])) {
        $vowels++;
    }
}
?>
```

讨论

通过每次处理字符串中的一个字符，很容易计算出“Look and Say”序列（译注1），如例1-18所示。

例1-18：“Look and Say” 序列

```
<?php
function lookandsay($s) {
    // 将保存返回值的变量初始化为空字符串
    $r = '';
    // $m 用于保存我们要查找的字符，
```

译注1： J. H. Conway发明的一个著名的整数序列。

```
    // 同时将其初始化为字符串中的第一个字符
    $m = $s[0];
    // $n 用于保存我们找到的$m的数目，将其初始化为1
    $n = 1;
    for ($i = 1, $j = strlen($s); $i < $j; $i++) {
        // 如果这个字符与上一个字符相同
        if ($s[$i] == $m) {
            // 这个字符的数目加1
            $n++;
        } else {
            // 否则，把数目和这个字符追加到返回值
            $r .= $n.$m;
            // 把要找的字符设置为当前的字符
            $m = $s[$i];
            // 并把数目重置为1
            $n = 1;
        }
    }
    // 返回构建好的字符串以及最终的数目和字符
    return $r.$n.$m;
}

for ($i = 0, $s = 1; $i < 10; $i++) {
    $s = lookandsay($s);
    print "$s <br/>\n";
}
```

例1-18的输出结果：

```
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
13211311123113112211
```

之所以把其称为“Look and Say”序列，是因为其中的每一个元素都可以通过看前一个元素，并根据前一个元素说出来这个元素的构成而得出来。比如说，看到第一个元素——1，你会说“一个1”。所以第二个元素就是“11”。它是两个1，所以第三个元素是“21”。类似地，它是一个2和一个1，所以第四个元素就是“1211”，之后依次类推。

参见

for语句的文档（*http://www.php.net/for*）。关于“Look and Say”序列的详细介绍可以参考*http://mathworld.wolfram.com/LookandSaySequence.html*。

1.5 按字或按字节来反转字符串

问题

你想要反转字符串的字或字节的顺序。

方案

使用strrev()来按字节反转字符的顺序，如例1-19所示。

例1-19：按字节来反转字符串

```
<?php
print strrev('This is not a palindrome.');
?>
```

例1-19的输出结果：

```
.emordnilap a ton si sihT
```

如果想按字反转，可以先根据字之间的空格符把整个字符串分解为独立的字，然后反转这些字的顺序，最后把反转后的字重新组合起来，如例1-20所示。

例1-20：按字反转字符串

```
<?php
$s = "Once upon a time there was a turtle.";
// 将字符串分解为独立的字
$words = explode(' ',$s);
// 反转这个字数组
$words = array_reverse($words);
// 重建反转后的字符串
$s = implode(' ',$words);
print $s;
?>
```

例1-20 的输出结果：

```
turtle. a was there time a upon Once
```

讨论

按字反转字符串也可以用例1-21中所示的一行代码来实现。

例1-21：简化后的按字反转字符串的代码。

```
<?php
```

```
$reversed_s = implode(' ',array_reverse(explode(' ',$s)));
?>
```

参见

技巧23.7中讨论了使用非空格字符作为字边界分解文件的内容。strrev()函数的文档（*http://www.php.net/strrev*）和array_reverse()函数的文档（*http://www.php.net/array-reverse*）。

1.6 扩展和压缩制表符

问题

你想把字符串中的空格符变成制表符（或相反），直到文本与制表位对齐。例如，你想以标准的方式向用户显示格式化以后的文本内容。

方案

使用str_replace()把空格符替换成制表符，或者把制表符替换成空格符，如例1-22所示。

例1-22：制表符与空格符相互转换

```
<?php
$r = mysql_query("SELECT message FROM messages WHERE id = 1") or die();
$ob = mysql_fetch_object($r);
$tabbed = str_replace(' ',"\t",$ob->message);
$spaced = str_replace("\t",' ',$ob->message);

print "With Tabs: <pre>$tabbed</pre>";
print "With Spaces: <pre>$spaced</pre>";
?>
```

但是，使用str_replace()进行转换没有考虑到制表位的问题。如果你想要每八个字符设置一个制表位，那么对于以一个五个字母的单词和一个制表符开头的一行文本，这个制表符就要用三个空格符来替换，而不是一个。使用例1-23中的pc_tab_expand()函数，可以在考虑制表位的前提下把制表符转换为空格符。

例1-23：pc_tab_expand()函数

```
<?php
function pc_tab_expand($text) {
    while (strstr($text,"\t")) {
        $text = preg_replace_callback('/^([^\t\n]*)(\t+)/m','pc_tab_expand_helper',
```

```
$text);
    }
    return $text;
}

function pc_tab_expand_helper($matches) {
    $tab_stop = 8;

    return $matches[1] .
    str_repeat(' ',strlen($matches[2]) *
                    $tab_stop - (strlen($matches[1]) % $tab_stop));
}


$spaced = pc_tab_expand($ob->message);
?>
```

可以使用例1-24中的pc_tab_unexpand()函数把空格符转换回制表符。

例1-24：pc_tab_unexpand()函数

```
<?php
function pc_tab_unexpand($text) {
    $tab_stop = 8;
    $lines = explode("\n",$text);
    foreach ($lines as $i => $line) {
        // 把所有制表符扩展为空格符
        $line = pc_tab_expand($line);
        $chunks = str_split($line, $tab_stop);
        $chunkCount = count($chunks);
        // 扫描除最后一个字符段之外的所有字符段
        for ($j = 0; $j < $chunkCount - 1; $j++) {
            $chunks[$j] = preg_replace('/ {2,}$/',"\t",$chunks[$j]);
        }
        // 如果最后一个字符段是相当于一个制表位的空格符
        // 将其转换为制表符，否则，不作任何处理
        if ($chunks[$chunkCount-1] == str_repeat(' ', $tab_stop)) {
            $chunks[$chunkCount-1] = "\t";
        }
        // 重组所有字符段
        $lines[$i] = implode('',$chunks);
    }
    // 重组所有行
    return implode("\n",$lines);
}

$tabbed = pc_tab_unexpand($ob->message);
?>
```

这两个函数都以一个字符串作为参数，并返回经过修改后的结果字符串。

讨论

这两个函数都假定每8个字符设置一个制表位，不过这可以通过更改变量$tab_stops的设置来重新设定。

函数pc_tab_expand()中的正则表达式既可以匹配一组制表符，也可匹配一行中位于这组制表符之前的所有文本。之所以需要匹配这组制表符之前的文本，是因为那些文本的长度会影响到这组制表符应该用多少个空格来替换，才能使后面的文本与下一个制表位对齐。这个函数不仅仅只是将每个制表符都替换为8个空格符，它还要调整制表符后面的文本与制表位对齐。

同样地，pc_tab_unexpand()函数也不仅仅是寻找8个连续的空格符，然后用一个制表符将其替换掉那么简单。它会把每一行都分割成8个字符一组的字符段，然后把这些字符段末尾处的空白（至少两个空格）替换成制表符。这样，不仅可以保持文本与制表符对齐，而且还可以保留字符串中的空格。

参见

str_replace()函数的文档（*http://www.php.net/str-replace*），preg_replace_callback()函数的文档（*http://www.php.net/preg_replace_callback*），str_split()函数的文档（*http://www.php.net/str_split*）；技巧22.10中有关preg_replace_callback()函数的更多信息。

1.7 控制大小写

问题

你需要将一个字符串中的字母转换成首字母大写、小写或其他大小写形式。例如，你想让名字的首字母大写，而其他字母小写。

方案

使用ucfirst()或者ucwords()将一个或多个单词的首字母转换为大写形式，如例1-25所示。

例1-25：将字母转换成大写

```
<?php
print ucfirst("how do you do today?");
```

```
print ucwords("the prince of wales");
?>
```

例1-25的输出结果：

```
How do you do today?
The Prince Of Wales
```

使用strtolower()或者strtoupper()修改整个字符串的大小写，如例1-26所示。

例1-26：改变字符串的大小写形式

```
print strtoupper("i'm not yelling!");
// 要符合XHTML的标准，标记必须小写s
print strtolower('<A HREF="one.php">one</A>');
```

例1-26的输出结果：

```
I'M NOT YELLING!
<a href="one.php">one</a>
```

讨论

使用ucfirst()可以将一个字符串中的第一个字母转换成大写形式：

```
<?php
print ucfirst('monkey face');
print ucfirst('1 monkey face');
?>
```

其输出结果：

```
Monkey face
1 monkey face
```

注意，第二个字符串没有变成“1 Monkey face”。

使用ucwords()将一个字符串中每个单词的首字母转换成大写形式：

```
<?php
print ucwords('1 monkey face');
print ucwords("don't play zone defense against the philadelphia 76-ers");
?>
```

其输出结果：

```
1 Monkey Face
Don't Play Zone Defense Against The Philadelphia 76-ers
```

同我们所希望的一样，ucwords()没有将“don’t”中“t”变成大写。但是，它同样也不会把“76-ers”中“e”转换为大写。对于ucwords()而言，一个词指的是任何非空白符序列后跟一个或多个空白字符。因为“'”和“-”均不是空白字符，所以ucwords()就不会认为“don’t”中的“t”或者“76-ers”中的“e”是一个单词的首字母。

ucfirst()和ucwords()函数都不会改变非首字符的大小写形式：

```
<?php
print ucfirst('macWorld says I should get an iBook');
print ucwords('eTunaFish.com might buy itunaFish.Com!');
?>
```

其输出结果：

```
MacWorld says I should get an iBook
ETunaFish.com Might Buy ItunaFish.Com!
```

函数strtolower()和strtoupper()都是对整个字符串起作用的，而不仅仅是针对个别的字符。strtolower()会把所有字母字符转换为小写字母，而strtoupper()则会把所有字母字符转换为大写字母：

```
<?php
print strtolower("I programmed the WOPR and the TRS-80.");
print strtoupper('"since feeling is first" is a poem by e. e. cummings.');
?>
```

其输出结果：

```
i programmed the wopr and the trs-80.
"SINCE FEELING IS FiRST" IS A POEM BY E. E. CUMMINGS.
```

当确定大写或小写的时候，这些函数会考虑到你的本地化设置。

参见

有关本地化设置的更多内容，详见第19章；ucfirst()函数的文档（*http://www.php.net/ucfirst*）；ucwords()函数的文档（*http://www.php.net/ucwords*）；strtolower()函数的文档（*http://www.php.net/strtolower*）和strtoupper()函数的文档（*http://www.php.net/strtoupper*）。

1.8 在字符串中插入函数和表达式

问题

你想要在一个字符串中包含执行一个函数或表达式后的结果。

方案

当你想包含的值不能直接出现在字符串中时，就使用字符串连接操作符（.），如例1-27所示。

例1-27：字符串连接

```
<?php
print 'You have '.($_REQUEST['boys'] + $_REQUEST['girls']).' children.';
print "The word '$word' is ".strlen($word).' characters long.';
print 'You owe '.$amounts['payment'].' immediately';
print "My circle's diameter is ".$circle->getDiameter().' inches.';
?>
```

讨论

你可以把变量、对象的属性及数组元素（在数组元素的引用变量不加引号的情况下）直接放在双引号字符串中。

```
<?php
print "I have $children children.";
print "You owe $amounts[payment] immediately.";
print "My circle's diameter is $circle->diameter inches.";
?>
```

在双引号字符串中插入值会受到其自身插入语法的一些限制。在上例中，`$amounts['payment']`必须写成`$amounts[payment]`，才能够正确地插入。使用大括号可以将更复杂的表达式插入到一个字符串中。例如：

```
<?php
print "I have less than {$children} children.";
print "You owe {$amounts['payment']} immediately.";
print "My circle's diameter is {$circle->getDiameter()} inches.";
?>
```

直接插值或者使用字符串连接同样也适用于以heredoc形式定义的字符串。以字符串连接的形式在heredoc定义的字符串中插值，看起来可能会有一点不习惯，因为结束heredoc的定界符与字符串连接操作符必须写在不同的行中：

```
<?php
print <<< END
Right now, the time is
END
. strftime('%c') . <<< END
 but tomorrow it will be
END
. strftime('%c',time() + 86400);
?>
```

同样，如果使用heredoc进行插值，要保证包含适当的空格，以便整个字符串能够自然地显示。在前面的例子中，“Right now, the time is”后面必须包含一个尾随的空格，而“but tomorrow it will be”则必须同时包含前置和尾随的空格。

参见

有关插入可变变量（诸如${"amount_$i"}）的语法，参见技巧5.4；有关字符串连接操作符的文档（*http://www.php.net/language.operators.string*）。

1.9 删除字符串两端的空白符

问题

你想删除一个字符串开始或者结尾处的空白符。例如，你想在验证用户的输入内容之前，先对输入内容进行清理。

方案

使用ltrim()、rtrim()或trim()。其中，ltrim()用于删除字符串开始处的空白符，rtrim()用于删除字符串结尾处的空白符，而trim()则能够同时删除字符串开始和结尾处的空白符：

```
<?php
$zipcode = trim($_REQUEST['zipcode']);
$no_linefeed = rtrim($_REQUEST['text']);
$name = ltrim($_REQUEST['name']);
?>
```

讨论

对于这些函数而言，所谓的空白符是指下列字符：换行符、回车符、空格符、水平和垂直制表符以及空值（null）。

删除字符串两端的空白符，不仅能够节省存储空间，而且还能使<pre>标记中的数据或文本内容显示得更精确。例如，如果你想比较用户的输入内容，就应该首先对数据进行清理，这样如果某人错将其所在地的邮政编码写成了“98052”，也不会被强迫修改为一个的确不是邮政编码的错误数字。在进行实际的文本比较之前进行清理也能够确保，比如说“salami\n”等于“salami”。而且，在将字符串数据存储到数据库中以前，通过删除空白符对其进行规范化处理也是一个好习惯。

trim()函数还能够从字符串中删除用户指定的字符。需要将要删除的字符作为第二个参数传入这个函数中。可以指定要删除字符的范围，在用于指定范围的首字符和末字符之间插入两个点来表示字符的范围：

```
<?php
// 从一行的开始处删除数字和空格
print ltrim('10 PRINT A$',' 0..9');
// 从一行的结尾处删除分号
print rtrim('SELECT * FROM turtles;',';');
?>
```

其输出结果：

```
PRINT A$
SELECT * FROM turtles
```

PHP还为rtrim()提供了一个别名函数chop()。然而，最好还是使用rtrim()，因为PHP中chop()函数的行为与Perl语言中的chop()（此chop()饱受非议，而chomp()更讨人喜欢）并不相同，而且使用了它也会使看到你代码的其他人感到迷惑。

参见

trim()函数的文档（*http://www.php.net/trim*）；ltrim()函数的文档（*http://www.php.net/ltrim*）和rtrim()函数的文档（*http://www.php.net/rtrim*）。

1.10 生成逗号分隔的数据

问题

你想把数据格式化为逗号分隔的值（Comma-separated values，CSV），以便导入到电子表格或者数据库中。

方案

使用fputcsv()函数由数据数组生成一行CSV格式的数据。例1-28把$sales中的数据写入到了一个文件中。

例1-28：生成逗号分隔的数据

```
<?php

$sales = array( array('Northeast','2005-01-01','2005-02-01',12.54),
                array('Northwest','2005-01-01','2005-02-01',546.33),
                array('Southeast','2005-01-01','2005-02-01',93.26),
                array('Southwest','2005-01-01','2005-02-01',945.21),
                array('All Regions','--','--',1597.34) );

$fh = fopen('sales.csv','w') or die("Can't open sales.csv");
foreach ($sales as $sales_line) {
    if (fputcsv($fh, $sales_line) === false) {
        die("Can't write CSV line");
    }
}
fclose($fh) or die("Can't close sales.csv");

?>
```

讨论

如果想输出CSV格式的数据而不是将其写入到一个文件中，可以像例1-29所示的那样使用特殊的输出流——php://output。

例1-29：显示逗号分隔的数据

```
<?php

$sales = array( array('Northeast','2005-01-01','2005-02-01',12.54),
                array('Northwest','2005-01-01','2005-02-01',546.33),
                array('Southeast','2005-01-01','2005-02-01',93.26),
                array('Southwest','2005-01-01','2005-02-01',945.21),
                array('All Regions','--','--',1597.34) );

$fh = fopen('php://output','w');
foreach ($sales as $sales_line) {
    if (fputcsv($fh, $sales_line) === false) {
        die("Can't write CSV line");
    }
}
fclose($fh);
?>
```

如果想把CSV格式的数据放到一个字符串中，而不是输出或者写入到文件，可以综合运用例1-29中的技术和输出缓冲，如例1-30所示。

例1-30：把CSV格式的数据放到一个字符串中。

```
<?php

$sales = array( array('Northeast','2005-01-01','2005-02-01',12.54),
                array('Northwest','2005-01-01','2005-02-01',546.33),
                array('Southeast','2005-01-01','2005-02-01',93.26),
                array('Southwest','2005-01-01','2005-02-01',945.21),
                array('All Regions','--','--',1597.34) );

ob_start();
$fh = fopen('php://output','w') or die("Can't open php://output");
foreach ($sales as $sales_line) {
    if (fputcsv($fh, $sales_line) === false) {
        die("Can't write CSV line");
    }
}
fclose($fh) or die("Can't close php://output");
$output = ob_get_contents();
ob_end_clean();
?>
```

参见

fputcsv()函数的文档（*http://www.php.net/fputcsv*）；技巧8.12中有关输出缓冲的更多信息。

1.11 解析逗号分隔的数据

问题

你有一些逗号分隔格式的数据（CSV）值 —— 例如，一个由Excel或者数据库导出的文件 —— 想把这些记录和字段抽取成一种可以在PHP中处理的数据格式。

方案

如果CSV数据包含在一个文件（包括可以通过一个URL读取的文件）中，先用fopen()打开这个文件，然后用fgetcsv()函数读取其中的数据。例1-31把CSV数据输出到了一个HTML表格中。

例1-31：从文件中读取CSV数据

```
<?php
$fp = fopen('sample2.csv','r') or die("can't open file");
print "<table>\n";
while($csv_line = fgetcsv($fp)) {
    print '<tr>';
    for ($i = 0, $j = count($csv_line); $i < $j; $i++) {
```

```
        print '<td>'.htmlentities($csv_line[$i]).'</td>';
    }
    print "</tr>\n";
}
print '</table>\n';
fclose($fp) or die("can't close file");
?>
```

讨论

在PHP 4中，必须为fgetcsv()提供第二个参数，这个参数是一个大于CSV文件中最长一行长度的值（别忘记把行尾的空白符计算在内）。在PHP 5中，这个表示行长度的参数变成了可选的。如果不指定这个参数，fgetcsv()函数会读取一整行数据（在PHP 5.0.4及以后的版本中，将此参数设置为0也会达到同样的效果）。当平均的行长度超过8192字节时，如果你指定了一个明确的行长度，而不是让PHP自己去计算的话，那么程序运行速度会加快。

也可以给fgetcsv()函数传递可选的第三个参数，这个参数代替逗号作为数据的分隔符。但是，使用其他的分隔符会令CSV格式作为表列数据交换的一种简便途径的效果稍微打一些折扣。

不要试图绕过fgetcsv()函数，只想读取一行然后使用explode()按照逗号进行解析。CSV的实际情况要比这种方式能够处理的格式更复杂，比如说某些字段值中包含逗号直接量时，不应该将字段中包含的逗号当成是字段的分隔符。使用fgetcsv()可以保证你和你的代码避免这些不明显的错误。

参见

fgetcsv()函数的文档（*http://www.php.net/fgetcsv*）。

1.12 生成字段宽度固定的数据记录

问题

你需要格式化数据记录，使其中的每个字段都包含特定数量的字符。

方案

使用pack()函数，向其传递一个表示空格填充模式的字符串序列作为参数。例1-32把一个数据的数组转换成了一个固定宽度的记录。

例1-32：生成固定宽度字段的数据记录

```
<?php

$books = array( array('Elmer Gantry', 'Sinclair Lewis', 1927),
                array('The Scarlatti Inheritance','Robert Ludlum',1971),
                array('The Parsifal Mosaic','William Styron',1979) );

foreach ($books as $book) {
    print pack('A25A15A4', $book[0], $book[1], $book[2]) . "\n";
}

?>
```

讨论

表示格式的字符串A25A15 A4告诉pack()把其后的参数分别转换成一个25个字符长的以空格填充的字符串、一个14个字符长的以空格填充的字符串和一个4个字符长的以空格填充的字符串。Pack()函数为在固定宽度的记录中生成以空格填充的字段，提供了一个简洁的解决方案。

如果想用空格以外的其他字符来填充字段，就要使用substr()来确保每个字段的值不会过长，并且还要使用str_pad()来保证每个字段的值不会过短。例1-33把一个记录数组转换成了一个以 . 来填充的固定宽度记录。

例1-33：不使用pack()函数生成固定宽度字段的数据记录

```
<?php

$books = array( array('Elmer Gantry', 'Sinclair Lewis', 1927),
                array('The Scarlatti Inheritance','Robert Ludlum',1971),
                array('The Parsifal Mosaic','William Styron',1979) );

foreach ($books as $book) {
    $title  = str_pad(substr($book[0], 0, 25), 25, '.');
    $author = str_pad(substr($book[1], 0, 15), 15, '.');
    $year   = str_pad(substr($book[2], 0, 4), 4, '.');
    print "$title$author$year\n";
}

?>
```

参见

pack()函数的文档（*http://www.php.net/pack*）和str_pad()函数的文档（*http://www.php.net/str_pad*）；技巧1.16中有关使用pack()格式化字符串的更详细讨论。

1.13 解析字段宽度固定的数据记录

问题

你需要把固定宽度的记录分解成字符串。

方案

如例1-34所示使用substr()函数。

例1-34：用substr()函数解析固定宽度的记录

```
<?php
$fp = fopen('fixed-width-records.txt','r') or die ("can't open file");
while ($s = fgets($fp,1024)) {
    $fields[1] = substr($s,0,10);  // 第一个字段：此行中的前10个字符
    $fields[2] = substr($s,10,5);  // 第二个字段：此行中的下5个字符
    $fields[3] = substr($s,15,12); // 第三个字段：此行中的下12个字符
    // 调用对这个数组进行处理的函数
    process_fields($fields);
}
fclose($fp) or die("can't close file");
?>
```

或者使用unpack()，如例1-35所示。

例1-35：用unpack()函数解析固定宽度的记录

```
<?php
$fp = fopen('fixed-width-records.txt','r') or die ("can't open file");
while ($s = fgets($fp,1024)) {
    // 一个键分别为"title"、"author" 和 "publication_year"的关联数组
    $fields = unpack('A25title/A14author/A4publication_year',$s);
    // 调用处理这个数组的函数
    process_fields($fields);
}
fclose($fp) or die("can't close file");
?>
```

讨论

这些每行中的字段都被分配了固定字符数的数据类似下面的图书目录，每一行中都包含标题、作者和出版日期：

```
<?php
$booklist=<<<END
Elmer Gantry            Sinclair Lewis1927
The Scarlatti InheritanceRobert Ludlum 1971
```

```
The Parsifal Mosaic      Robert Ludlum 1982
Sophie's Choice          William Styron1979
END;
?>
```

在每一行中，标题占据了前25个字符的位置，作者的名字占据了下14个字符的位置，而出版日期则占据了最后4个字符的位置。知道了这些字段的宽度后，就很容易使用substr()函数把这些字段解析为一个数组：

```
<?php
$books = explode("\n",$booklist);

for($i = 0, $j = count($books); $i < $j; $i++) {
  $book_array[$i]['title'] = substr($books[$i],0,25);
  $book_array[$i]['author'] = substr($books[$i],25,14);
  $book_array[$i]['publication_year'] = substr($books[$i],39,4);
}
?>
```

将$booklist分解成一个以每行数据作为元素的数组后，相应的循环代码同操纵一个字符串或者操纵从文件中读取的一系列行的代码就没有什么分别了。

如果把字段名称和宽度分别以独立数组的形式作为参数传递给一个解析函数，会使得以上循环变得更加灵活，如例1-36中所示的pc_fixed_width_substr()函数。

例1-36：pc_fixed_width_substr()函数

```
<?php
function pc_fixed_width_substr($fields,$data) {
  $r = array();
  for ($i = 0, $j = count($data); $i < $j; $i++) {
    $line_pos = 0;
    foreach($fields as $field_name => $field_length) {
      $r[$i][$field_name] = rtrim(substr($data[$i],$line_pos,$field_length));
      $line_pos += $field_length;
    }
  }
  return $r;
}

$book_fields = array('title' => 25,
                     'author' => 14,
                     'publication_year' => 4);

$book_array = pc_fixed_width_substr($book_fields,$books);
?>
```

变量$line_pos中保存着每一个字段的开始位置，而且当代码处理每一行时会不断地将前一个字段的宽度加到这个变量上。同时，要用rtrim()把每个字段末尾的空白符删除。

可以使用unpack()代替substr()来提取字段。此时不必使用包含字段名和宽度的关联数组，而是要为unpack()创建一个格式化字符串作为参数。调用unpack()函数的固定宽度字段解析程序，与例1-37中的pc_fixed_width_unpack()函数相似。

例1-37：pc_fixed_width_unpack()函数

```
<?php

function pc_fixed_width_unpack($format_string,$data) {
  $r = array();
  for ($i = 0, $j = count($data); $i < $j; $i++) {
    $r[$i] = unpack($format_string,$data[$i]);
  }
  return $r;
}

$book_array = pc_fixed_width_unpack('A25title/A14author/A4publication_year',
                                    $books);

?>
```

因为A格式对于unpack()函数意味着“空格填充的字符串”，所以这里不需要用rtrim()删除每个字段末尾的空格。

只要调用上面两个函数中的任何一个把字段数据解析到$book_array中，就能以HTML表格的形式显示出来，例如：

```
<?php
$book_array = pc_fixed_width_unpack('A25title/A14author/A4publication_year',
                                    $books);
print "<table>\n";
// 输出表头
print '<tr><td>';
print join('</td><td>',array_keys($book_array[0]));
print "</td></tr>\n";
// 输出每一行
foreach ($book_array as $row) {
    print '<tr><td>';
    print join('</td><td>',array_values($row));
    print "</td></tr>\n";
}
print '</table>\n';
?>
```

用</td><td>来连接数据，会导致缺少两头的<td>和</td>表格行标记。为此，我们在连接数据之前先输出<tr><td>，在连接数据之后再输出</td></tr>就能解决这个问题。

当要解析的固定宽度的字段都是字符串时，函数substr()和unpack()具有相同的效用。但当字段中的元素不仅仅是字符串时，使用unpack()函数则是更好的解决方案。

如果所有字段的长度都是相等的，那么str_split()函数就是一个分割输入数据的便捷方法。可以在PHP 5中使用这个函数，这个函数会返回由字符串的各个部分构成的一个数组。例1-38就使用了str_split()函数把一个字符串分割成32个字节一组的片段。

例1-38：用str_split()分割字符串

```
<?php
$fields = str_split($line_of_data,32);
// $fields[0] 中保存着第 0~31字节
// $fields[1] 中保存着第 32~63字节
// 依次类推
?>
```

参见

技巧1.16和文档（*http://www.php.net/unpack*）中有关unpack()的更多信息；str_split()函数的文档（*http://www.php.net/str_split*）；技巧4.8中讨论的join()函数的相关内容。

1.14 分离字符串

问题

你需要把一个字符串分割成一些片段。例如，你想要访问用户在<textarea>表单字段中键入的每一行文本。

方案

如果可以用一个固定的字符串作为分隔符来分割这些片段，就使用explode()函数：

```
<?php
$words = explode(' ','My sentence is not very complicated');
?>
```

如果需要使用POSIX或者符合Perl语言实现的正则表达式来描述这个分隔符，则需使用split()或preg_split()：

```
<?php
$words = split(' +','This sentence  has  some extra whitespace  in it.');
$words = preg_split('/\d\. /','my day: 1. get up 2. get dressed 3. eat toast');
$lines = preg_split('/[\n\r]+/',$_REQUEST['textarea']);
?>
```

使用spliti()或者在preg_split()中使用/i标志，表示匹配的分隔符不区分大小写：

```
<?php
$words = spliti(' x ','31 inches x 22 inches X 9 inches');
$words = preg_split('/ x /i','31 inches x 22 inches X 9 inches');
?>
```

讨论

以上函数中最简单的方案就是explode()。只需给它传递三个简单的参数：一个作为分隔符的字符串、要分割的字符串和一个可选的用于限定返回的数组中元素个数的数值：

```
<?php
$dwarves = 'dopey,sleepy,happy,grumpy,sneezy,bashful,doc';
$dwarf_array = explode(',',$dwarves);
?>
```

以上代码会将一个包含7个元素的数组保存到$dwarf_array中，所以print_r($dwarf_array)的输出结果：

```
Array
(
    [0] => dopey
    [1] => sleepy
    [2] => happy
    [3] => grumpy
    [4] => sneezy
    [5] => bashful
    [6] => doc
)
```

如果指定的限定值小于可能字符段数，那么最后的字符段中将包含剩余的所有字符串：

```
<?php
$dwarf_array = explode(',',$dwarves,5);
print_r($dwarf_array);
?>
```

其输出结果：

```
Array
(
    [0] => dopey
    [1] => sleepy
    [2] => happy
    [3] => grumpy
    [4] => sneezy,bashful,doc
)
```

explode()会将作为分隔符的字符串看成是直接量。所以，如果你指定将一个逗号和一个

空格作为一个分隔符，这个函数会只在逗号后面跟一个空格的地方分割字符串，而不会在一个逗号或者一个空格处分割字符串。

使用split()函数会获得更大的灵活性。这个函数不使用字符串直接量作为分隔符，而是使用一个POSIX正则表达式：

```
<?php
$more_dwarves = 'cheeky,fatso, wonder boy, chunky,growly, groggy, winky';
$more_dwarf_array = split(', ?',$more_dwarves);
?>
```

以上代码中的正则表达式表示将一个逗号后跟一个可选的空格作为分隔符，这样就可以适当地区分所有侏儒的名字。主要是它不会分割带有空格的侏儒名字，而是以“, ”或者“, ”来分隔每一个名字。print_r($more_dwarf_array)的输出结果为：

```
Array
(
    [0] => cheeky
    [1] => fatso
    [2] => wonder boy
    [3] => chunky
    [4] => growly
    [5] => groggy
    [6] => winky
)
```

preg_split()与split()类似，它使用的是一个符合Perl语言实现的正则表达式引擎，而不是一个POSIX正则表达式引擎。通过preg_split()，既可以利用多种Perl风格的正则表达式扩展，还能在返回的字符串数组中包含作为分隔符的文本：

```
<?php
$math = "3 + 2 / 7 - 9";
$stack = preg_split('/ *([+\-\/*]) */',$math,-1,PREG_SPLIT_DELIM_CAPTURE);
print_r($stack);
?>
```

其输出结果：

```
Array
(
    [0] => 3
    [1] => +
    [2] => 2
    [3] => /
    [4] => 7
    [5] => -
    [6] => 9
)
```

以上代码中用到的正则表达式会寻找四个数学运算符（+、-、*、/），这些运算符可能会被可选的前置或后置空格所包围。PREG_SPLIT_DELIM_CAPTURE标志的作用是告诉preg_split()，在返回的字符串数组中也要包含与括号中的正则表达式匹配的作为分隔符的字符串。因为括号中只包含数学运算符类的字符，所以作为返回数组元素的这些数学运算符不会包含任何空格。

参见

第22章中有关正则表达式的更详细介绍；explode()函数的文档（*http://www.php.net/explode*），split ()函数的文档（*http://www.php.net/split*）和preg_split()函数的文档（*http://www.php.net/preg-split*）。

1.15 使文本在特定长度处自动换行

问题

你需要对一个字符串进行换行处理。例如，你想显示<pre>/</pre>标记中的文本，但希望这些文本能在正常大小的浏览器窗口中自动地换行。

方案

使用wordwrap()函数：

```
<?php
$s = "Four score and seven years ago our fathers brought forth ↵
   on this continent a new nation, conceived in liberty and ↵
   dedicated to the proposition that all men are created equal.";

print  "<pre>\n".wordwrap($s)."\n</pre>";
?>
```

其输出结果：

```
<pre>
Four score and seven years ago our fathers brought forth on this continent
a new nation, conceived in liberty and dedicated to the proposition that
all men are created equal.
</pre>
```

讨论

在默认情况下，wordwrap()会按照每行75个字符来自动将文本换行。使用可选的第二个参数来指定不同的行长度：

```
<?php
print wordwrap($s,50);
?>
```

这样，其输出结果就变成了：

```
Four score and seven years ago our fathers brought
forth on this continent a new nation, conceived in
liberty and dedicated to the proposition that all
men are created equal.
```

除了\n之外的其他字符也可以作为换行符（译注2）。要想输出两个空行，可以用“\n\n”：

```
<?php
print wordwrap($s,50,"\n\n");
?>
```

其输出结果：

```
Four score and seven years ago our fathers brought

forth on this continent a new nation, conceived in

liberty and dedicated to the proposition that all

men are created equal.
```

wordwrap()还有可选的第四个参数，用于控制对那些比指定的行长度值更长的单词的处理方式。如果这个参数值为1，超出指定长度的这些单词会换行；否则，这些单词就会保留原来的行长度：

```
<?php
print wordwrap('jabberwocky',5);
print wordwrap('jabberwocky',5,"\n",1);
?>
```

其输出结果：

```
jabberwocky
```

译注2：在第三个参数中指定。

```
jabbe
rwock
y
```

参见

wordwrap()函数的文档（*http://www.php.net/wordwrap*）。

1.16 在字符串中存储二进制数据

问题

你想解析一个包含着二进制编码值的字符串，或者想把值编码为字符串。例如，你想以二进制表示法保存数值，而不以是ACSII字符的序列的形式。

方案

使用pack()把二进制数据保存到一个字符串中：

```
<?php
$packed = pack('S4',1974,106,28225,32725);
?>
```

使用unpack()从一个字符串中抽取二进制数据：

```
<?php
$nums = unpack('S4',$packed);
?>
```

讨论

传递给pack()函数的第一个参数是一个用于描述如何对其他参数进行编码的格式化字符串。其中格式化字符串S4告诉pack()，按照计算机字节的序列由输入的数据生成四位无符号的短16位数。对于给定的1974，106，28225和32725作为基于little-endian序列的计算机输入数据，会生成8组数字：182，7，106，0，65，110，213和127。其中，每两个字节对应着一个输入的数据值，即7 × 256 + 182是1974，0 × 256 + 106是106，110 × 256 + 65 = 28225，127 × 256 + 213 = 32725。

传递给unpack()函数的第一个参数也是一个格式化字符串，而第二个参数是要解码的数据。将S4作为格式化字符串，可以把由pack()函数生成的8组数字返回一个包含四个元素的数组，每个元素中保存着一个原始的数值。print_r($nums)的输出结果为：

```
Array
(
    [1] => 1974
    [2] => 106
    [3] => 28225
    [4] => 32725
)
```

在unpack()函数内部，格式字符以及其数目的后面还可以加一个字符串，这个字符串将作为一个数组的键。例如：

```
<?php
$nums = unpack('S4num',$packed);
print_r($nums);
?>
```

其输出结果：

```
Array
(
    [num1] => 1974
    [num2] => 106
    [num3] => 28225
    [num4] => 32725
)
```

如果要在unpack()函数中使用多个格式化字符，则必须使用/来分隔：

```
<?php
$nums = unpack('S1a/S1b/S1c/S1d',$packed);
print_r($nums);
?>
```

其输出结果：

```
Array
(
    [a] => 1974
    [b] => 106
    [c] => 28225
    [d] => 32725
)
```

可以用在pack()和unpack()函数中的格式化字符见表1-2。

表1-2：用于pack()和unpack()函数的格式化字符

格式化字符	数据类型
a	无填充的字符串
A	空格填充的字符串

表1-2：用于pack()和unpack()函数的格式化字符（续）

格式化字符	数据类型
h	16位字符串，以低四位字节开始
H	16位字符串，以高四位字节开始
c	带符号的字符
C	无符号的字符
s	带符号的短整型数（16位，计算机字节序列）
S	无符号的短整型数（16位，计算机字节序列）
n	无符号的短整型数（16位，big endian字节序列）
v	无符号的短整型数（16位，little endian字节序列）
i	带符号的整数（大小与字节序列同计算机相关）
I	无符号的整数（大小与字节序列同计算机相关）
l	带符号的长整型数（32位，计算机字节序列）
L	无符号的长整型数（32位，计算机字节序列）
N	无符号的长整型数（32位，big endian字节序列）
V	无符号的长整型数（32位，little endian字节序列）
f	浮点型数（大小和表示法同计算机相关）
d	双精度型数（大小和表示法同计算机相关）
x	空字节
X	倒退一个字节
@	绝对位置以空值填充

对于a、A、h和H，将一个数字放在它们后面来表示这个字符串有多长。例如，A25表示一个25个字符的、以空格填充的字符串。对于其他格式化字符，后置的数字意味着相应的类型在字符串中出现多少次。使用*表示其余的现有数据。

```
<?php
$s = 'platypus';
$ascii = unpack('c*',$s);
print_r($ascii);
?>
```

其输出结果：

```
Array
(
    [1] => 112
    [2] => 108
```

```
    [3] => 97
    [4] => 116
    [5] => 121
    [6] => 112
    [7] => 117
    [8] => 115
)
```

参见

pack()函数的文档（*http://www.php.net/pack*）和unpack()函数的文档（*http://www.php.net/unpack*）。

1.17 编程：可下载的CSV文件

通过结合使用header()函数来改变在PHP程序中以fputcsv()函数输出的数据格式的内容类型（contetn type），可以实现将CSV文件发送给浏览器的功能。浏览器接收到CSV文件后会自动调用电子表格软件（或者其他与特定的客户端系统相关联的CSV文件编辑程序）对其进行处理。例1-39将SQL SELECT查询的结果格式化为CSV数据，并提供了正确的响应头部，保证了发送给浏览器的数据文件能够得到适当的处理。

例1-39：可下载的CSV文件

```
<?php

require_once 'DB.php';
// 连接到数据库
$db = DB::connect('mysql://david:haxOr@localhost/phpcookbook');

// 从数据库中获取数据
$sales_data = $db->getAll('SELECT region, start, end, amount FROM sales');
// 为fputcsv()函数打开文件句柄
$output = fopen('php://output','w') or die("Can't open php://output");
$total = 0;

// 告诉浏览器发送的是一个CSV文件
header('Content-Type: application/csv');
header('Content-Disposition: attachment; filename="sales.csv"');

// 输出表头
fputcsv($output,array('Region','Start Date','End Date','Amount'));
// 输出每一行数据，并递增$total
foreach ($sales_data as $sales_line) {
    fputcsv($output, $sales_line);
    $total += $sales_line[3];
}
```

```
// 输出全部数据行，并关闭文件句柄
fputcsv($output,array('All Regions','--','--',$total));
fclose($output) or die("Can't close php://output");

?>
```

例1-39为了保证浏览器能够正确地处理CSV输出设置了两个响应头部信息。第一个，Content-Type，用于告诉浏览器所输出的不是HTML格式，而是CSV格式。第二个为Content-Disposition，告诉浏览器不要直接显示所输出的CSV格式的数据，而是要尝试载入一个外部程序来处理这些数据。其中的filename属性，为浏览器下载的文件提供了一个默认的文件名。

如果想为同一数据提供不同的查看方式，可以在一个页面中组合使用格式化代码，并通过一个查询字符串变量来决定要生成何种格式。在例1-40中，格式化查询字符串变量用于控制SQL SELECT查询的结果是作为一个HTML表格还是一个CSV文件返回。

例1-40：动态生成CSV或HTML

```
<?php

$db = new PDO('sqlite:/usr/local/data/sales.db');

$query = $db->query('SELECT region, start, end, amount FROM sales', PDO::FETCH_
NUM);
$sales_data = $db->fetchAll();
$total = 0;
$column_headers = array('Region','Start Date','End Date','Amount');
// 确定使用何种格式
$format = $_GET['format'] == 'csv' ? 'csv' : 'html';

// 输出格式对应的开始部分
if ($format == 'csv') {
    $output = fopen('php://output','w') or die("Can't open php://output");
    header('Content-Type: application/csv');
    header('Content-Disposition: attachment; filename="sales.csv"');
    fputcsv($output,$column_headers);
 } else {
    echo '<table><tr><th>';
    echo implode('</th><th>', $column_headers);
    echo '</th></tr>';
 }

foreach ($sales_data as $sales_line) {
    // 输出格式对应的内容
    if ($format == 'csv') {
        fputcsv($output, $sales_line);
    } else {
        echo '<tr><td>' . implode('</td><td>', $sales_line) . '</td></tr>';
   }
```

```
    $total += $sales_line[3];
}
$total_line = array('All Regions','--','--',$total);

// 输出格式对应的结尾部分
if ($format == 'csv') {
    fputcsv($output,$total_line);
    fclose($output) or die("Can't close php://output");
 } else {
    echo '<tr><td>' . implode('</td><td>', $total_line) . '</td></tr>';
    echo '</table>';
 }
?>
```

如果在查询字符串中以format=csv来访问例1-40中的程序，会得到CSV格式的输出结果。而查询字符串中的任何其他格式的值都会得到HTML格式的输出结果。其中将$format设置为CSV或者HTML的逻辑很容易扩展到其他输出格式，如XML等。如果你在很多地方都需要对相同数据提供多种格式的下载，可以将例1-40中的代码打包成一个函数，使其接受两个参数 —— 第一个是要输出的数据数组，第二个是一个格式说明符，据此二者即可显示正确的结果。

第2章

数字

2.0 概述

在日常生活中，数字可以说无处不在。在表示当前时间的时候，它们可能是下午3:00，在表示一品脱牛奶的价格时，它们可能是1.29美元。它们也可能是指一个圆周相对于直径的比率——π。数字可以相当大，比如亚佛加德罗数（Avogadro's number）就有6×10^{23}那么大。在PHP中，数字可以表示以上全部含义。

但是，PHP不会把上述数字都看成没有区别的“数字”。相反，它把数字分成两组——整型数字和浮点型数字。整型数是指整数，比如-4、0、5和1，975。浮点型数指的是带小数的数，比如-1.23、0.0、3.14159和9.9999999999。

为了方便，在PHP中多数情况下你都不用考虑这两类数的差别，PHP会自动地把整型数转换为浮点型数，或者把浮点型数转换为整型数。这样你根本不用关心底层的细节。但这样同样也意味着`3/2`是`1.5`，而不像它在其他编程语言中那样是`1`。PHP还能够自动地把字符串转换成数字，反之亦然。例如：`1+“1”`会得到`2`。

然而，有时候这种有福气的省心也会带来麻烦。首先，数字不可以是无穷大或无穷小，比如一个最小的数字可能是2.2e-308，而一个最大的数字可能是1.8e308（注1）。如果你需要用到更大或者更小的数字，就必须使用BCMath库或者GMP库，这两个库我们会在技巧2.14中介绍。

其次，浮点型数计算仅能保证一定大小的数的加减运算的准确性，但不能保证全部计算结果都严格地正确。虽然能保证的范围很小，但除了在某些个别的情况下会碰到问题，大多数时候都是可以满足需要的。例如，人类会认为6.9999999999……是7，而PHP认为

注1：实际上，这些数字是与平台有关的，之所以这些值很常用是因为它们都是由64位的IEEE754标准所规定的。

它是6后面跟着一连串的9。因此，如果对于这个数你想取整，PHP会返回6而不是7。由于类似的原因，如果位于第200个小数位上数字对你而言非常重要，就不要使用浮点型数，而必须使用BCMath库和GMP库。但大多数情况下，在PHP中操纵数字值都会如你所愿的。

2.1 检查变量中是否包含有效的数字

问题

你想要确保一个变量中包含一个数字，即使其类型是一个字符串。同时，你还想要检查一个变量是否不仅是一个数字，而且也是某种特定的数据类型。

方案

使用is_numeric()来判断一个变量中是否包含着数字值：

```
<?php
if (is_numeric(5))            { /* true  */ }
if (is_numeric('5'))          { /* true  */ }
if (is_numeric("05"))         { /* true  */ }
if (is_numeric('five'))       { /* false */ }

if (is_numeric(0xDECAFBAD))   { /* true  */ }
if (is_numeric("10e200"))     { /* true  */ }
?>
```

讨论

数字可能以任何形式和大小出现。所以，不可能简单地因为某事物只包含0～9这几个字符就认为它是一个数字。如果是这样，那么小数点呢，还有负号呢？而且，也不能简单地把它们混到一起，因为负号必须位于数字的前面，并且一个数字中也只能出现一个小数点。再者，还有十六进制数和科学记数法要考虑。

与其编写自己的函数，不如使用is_numeric()来检查一个变量中是不是保存着一个实际的数字（以整型数或者浮点型数的形式出现），还是包含着一个可以转换成数字的字符串。

事实上，这两种情况是有区别的。从技术角度上说，整数5和字符串5在PHP中并不是同一类型。然而，多数时候你都不必担心这种差别，这就是is_numeric()函数的用处所在。

虽然is_numeric()函数能正确地解析带小数点的数字，比如5.1；但是，对于带有千分位分隔符的数字，比如5，000，这个函数却返回false。

所以，必须在调用is_numeric()函数之前先用str_replace()函数替换掉其中的千位分隔符：

```
<?php
is_numeric(str_replace($number, ',', ''));
?>
```

要检查数字是否是一个特殊的类型，可以使用很多种带有自解释名称的相关函数。比如，is_float()（或者is_double()、is_real()，它们都一样）和is_int()（或者is_integer()或is_long()）。

要验证输入的数据，可以使用技巧9.3中介绍的技术而非is_numeric()函数。该技巧中介绍了如何检查正的或负的整型数、小数以及许多其他格式的数。

参见

技巧9.3中有关验证用户输入数字的内容；is_numeric()函数的文档（*http://www.php.net/is-numeric*）和str_replace()函数的文档（*http://www.php.net/str-replace*）。

2.2 比较浮点型数字

问题

你想要检查两个浮点型数字是否相等。

方案

使用一个小增量值，并且检查两个数的差是否比这个增量值小：

```
<?php
$delta = 0.00001;

$a = 1.00000001;
$b = 1.00000000;

if (abs($a - $b) < $delta) { /* $a和$b相等*/ }
?>
```

讨论

浮点型数在计算机中以二进制形式表示时，只用有限的位数保存尾数和指数。当超出既定的位数时，就会发生溢出。因此，有的时候PHP（就像其他语言一样）不会认为两个相等的数就是真正的相等，因为也许在最后的某几位上存在着不一致的地方。

为了避免这个问题出现，不是要检查是否`$a==$b`，而是要确保第一个数位于第二个数的一个非常小的浮动范围（`$delta`）之内。这个浮动范围（增量）应该是你认为两个数之间应该存在的最小的差值数。然后，用`abs()`对这个差值取绝对值。

参见

技巧2-3中关于浮点型数取整的内容；PHP中有关浮点型数的文档（*http://www.php.net/language.types.float*）。

2.3 对浮点型数取整

问题

你想要对浮点型数进行取整处理，结果或者是一个整型数，或者是一个带有若干小数位的数。

方案

要想把一个数字取整为一个最接近的整型数，使用`round()`：

```
$number = round(2.4);   // $number = 2
```

要向上取整，使用`ceil()`：

```
$number = ceil(2.4);    // $number = 3
```

要向下取整，使用`floor()`：

```
$number = floor(2.4);   // $number = 2
```

讨论

如果一个数位于两个整数之间，PHP会向远离0的方向取整：

```
$number = round(2.5);   // 3

$number = round(-2.5);  // -3
```

我们在技巧2.2中提到过，由于存储它们的计算机不同，浮点型数并不总是能够计算出准确的值。这就导致了混乱的出现。一个你认为小数部分是“0.5”的值，可能会被“0.499999…9（后面全都是数字9）”或者是“.500000…1（后面除了最后是1全都是0）”所代替。

PHP会自动地在取整运算中加入一个小的“模糊因子（fuzz factor）”，所以你不必为此而担心。

为了在小数点后面保留一定的小数位，round()函数接受一个可选的表示精度的参数。例如，可能你需要计算用户的购物车中所有选择项目的总价格：

```
<?php
$cart = 54.23;
$tax = $cart * .05;
$total = $cart + $tax;        // $total = 56.9415

$final = round($total, 2);    // $final = 56.94
?>
```

若想向下取整，则须使用floor()函数：

```
$number = floor( 2.1);  // 2

$number = floor( 2.9);  // 2

$number = floor(-2.1);  // -3

$number = floor(-2.9);  // -3
```

若要向上取整，使用ceil()函数即可：

```
$number = ceil( 2.1);  // 3

$number = ceil( 2.9);  // 3

$number = ceil(-2.1);  // -2

$number = ceil(-2.9);  // -2
```

这两个函数之所如此命名，是因为当你向下取整时，你是在“向地板的方向”舍入，而当你向下取整时，你是在“向天花板的方向”舍入。

参见

技巧2.2中有关比较浮点型数的内容；ceil()函数的文档（*http://www.php.net/ceil*），floor()函数的文档（*ttp://www.php.net/floor*）以及round()函数的文档（*http://www.php.net/round*）。

2.4 操纵一系列连续的整数

问题

你想把一段代码应用到某一范围中的整数上。

方案

使用一个for循环：

```
<?php
for ($i = $start; $i <= $end; $i++) {
    plot_point($i);
}
?>
```

也可以让每次的增量不等于1。例如：

```
<?php
for ($i = $start; $i <= $end; $i += $increment) {
    plot_point($i);
}
?>
```

如果想保证使用的数字不只通过枚举产生，可以使用range()方法：

```
<?php
$range = range($start, $end);
?>
```

讨论

类似这种循环很常见。例如，你在绘制一幅函数曲线图的时候，会对图中连续多个点的位置进行计算；或者，你作为一名学生，会对离放学还有多少秒钟进行倒计时。

for循环的方法使用了一个整数作为计数器，而且能让你对整个循环的过程进行极好的控制，因为你可以让$i这个计数器变量自由地递增或者递减。而且，你还能在循环的内部修改$i。

在本方案的最后一个例子中，range()函数返回的是一个包含从$start到$end的数值的数组。使用range()函数的好处是它非常简短，但这一技术也有一些缺陷。举例来说，如果数组很大，那么就会浪费很多内存。而且，你每次只能对数组元素序列递增一个数，所以就不可能做到诸如循环遍历一个偶数序列。

$start可以比$end大。在这种情况下，由range()函数返回的数组元素中保存的是相应数字的降序序列。而且，你还可以使用这个函数来获得字符序列：

```
<?php
print_r(range('l', 'p'));
?>

Array
(
    [0] => l
    [1] => m
    [2] => n
    [3] => o
    [4] => p
)
```

参见

技巧4.3中有关按照指定的整数范围初始化一个数组的细节；range()函数的文档（*http://www.php.net/range*）。

2.5 在一个范围内生成随机数

问题

你想在给定的数字范围内生成一个随机数。

方案

使用mt_rand()：

```
// 生成一个大于等于$lower而小于等于$upper的随机数
$random_number = mt_rand($lower, $upper);
```

讨论

当你想要在网页中随机显示一幅图像时，随机地设置游戏的开始位置时，从数据库中随

机地选择一条记录时，或者生成一个随机的session标识符时，生成随机的数字都会非常有用。

要生成介于两个数之间的一个随机数，需要向mt_rand()函数传递两个参数：

```
$random_number = mt_rand(1, 100);
```

如果在调用mt_rand()方法时不带任何参数，则会返回一个介于0和最大的随机数之间的数，这个最大的随机数由mt_getrandmax()函数返回。

要生成真正的随机数，对于计算机而言并不是件容易的事。计算机擅长按照系统的指令运行，但却没有自发性。如果你想让计算机返回随机数，就需要给它设定一组可以重复执行的命令；事实上，就是让计算机不断地破坏可能出现的随机结果。

PHP中有两个不同的随机数生成程序，一个经典的函数叫rand()，而另一个更出色的函数是mt_rand()。MT代表的是Mersenne Twister，源于法国的一位修道士兼数学家Marin Mersenne和他的素数类型。该算法就是基与这些素数。因为mt_rand()更难以预测而且速度比rand()还要快，所以我更喜欢它而不是rand()。

如果你运行的是PHP 4.2之前的版本，那么在脚本中第一次使用mt_rand()（或rand()）函数之前，需要通过调用mt_srand()（或srand()）来产生“随机种子”。这个“随机种子”是随机函数用于生成返回的随机数的基础，也是对前面提到的重复性和随机性两难局面的一种解决方案。使用由microtime() —— 一个高精度的时间函数 —— 返回的值，就可以得到一个变化迅速而且不可能重复的种子（即高品质的种子）。初始化这个种子后，就不用再对随机数生成器重复播种了。PHP 4.2及以后的版本会自动地生成“随机种子”，但是如果你在第一次调用mt_rand()函数之前手动提供了一个种子，PHP不会用自己的新种子替换你的种子。

如果你想从数据库中随机地选择一条记录，一种简单的方式就是如例2-1所示的那样，先找出数据表中所有记录的总数，并在该总数的范围内选择一个随机数，然后再从数据库中请求选中的随机数对应的那一行记录。

例2-1：从数据库中随机选择一行记录

```
<?php
$sth = $dbh->query('SELECT COUNT(*) AS count FROM quotes');
if ($row = $sth->fetchRow()) {
    $count = $row[0];
} else {
    die ($row->getMessage());
}

$random = mt_rand(0, $count - 1);
```

```
$sth = $dbh->query("SELECT quote FROM quotes LIMIT $random,1");
while ($row = $sth->fetchRow()) {
    print $row[0] . "\n";
}
?>
```

以上代码片断先找到数据表中记录的总数，并在该范围内计算出一个随机数，然后使用LIMIT $random 1来到数据表中从$random位置开始查询（SELECT）一行记录。

还有另一种可能的解决方案，如果你使用的是MySQL 3.23或以上的版本，可以这样来做：

```
$sth = $dbh->query('SELECT quote FROM quotes ORDER BY RAND() LIMIT 1');
while ($row = $sth->fetchRow()) {
    print $row[0] . "\n";
}
```

这种情况下，MySQL会随机化记录行的排列顺序，然后返回第一行记录。

参见

技巧2.6中关于如何生成有偏随机数；mt_rand()函数的文档（*http://www.php.net/mt-rand*），rand()函数的文档（*http://www.php.net/rand*）；MySQL手册中有关rand()函数的内容部分（*http://www.mysql.com/doc/M/a/Mathematical_functions.html*）。

2.6 生成有偏随机数

问题

你想生成随机数字，但又希望这些数字具有某种倾向性，也就是说这些数字出现在某个范围内的概率会比出现在其他范围内的概率更高。例如，你想要根据每个广告活动未展示的剩余数目成比例地推出一系列网页横幅广告。

方案

使用如例2-2中所示的pc_rand_weighted()函数。

例2-2：pc_rand_weighted()函数

```
<?php
// 返回加权随机选择的键
function pc_rand_weighted($numbers) {
    $total = 0;
    foreach ($numbers as $number => $weight) {
```

```
            $total += $weight;
            $distribution[$number] = $total;
        }
        $rand = mt_rand(0, $total - 1);
        foreach ($distribution as $number => $weights) {
            if ($rand < $weights) { return $number; }
        }
    }
    ?>
```

讨论

想象一下，如果不使用一个其值为剩余展示数目的数组，而是用一个广告数组，其中每个广告的实际展示次数都与该广告的剩余展示数目一样多。你可以简单地在数组中选取一个未加权的随机位置，并从该位置开始展示。

如果广告的剩余展示数目达到百万级的话，使用这种技术则会消耗大量的内存。相反，你可以计算相应数组的大小（通过对剩余展示数目求合计值），从这个伪（make-believe）数组的元素总数中选取一个随机的数字，然后遍历该数组并找出与所选数字对应的广告。例如：

```
$ads = array('ford' => 12234, // 广告客户，剩余展示数目
             'att'  => 33424,
             'ibm'  => 16823);

$ad = pc_rand_weighted($ads);
```

参见

技巧2.5中有关在某一范围内生成随机数的内容。

2.7 取对数

问题

你想取一个数的对数。

方案

对于以*e*为底的对数（自然log），使用log()函数：

```
$log = log(10);                    // 2.30258092994
```

对于以10为底的对数，使用log10()函数：

```
$log10 = log10(10);          // 1
```

对于底为其他数的对数，则将该底作为第二个参数传递到log()函数中：

```
$log2  = log(10, 2);         // 3.3219280948874
```

讨论

log()和log10()函数都只是针对大于零的数而设计的。如果你在调用它们时传递的参数是一个等于或者小于零的数，它们会返回NAN，意即“Not A Number”。

参见

log()函数的文档（*http://www.php.net/log*）和log10()函数的文档（*http://www.php.net/log10*）。

2.8 计算指数

问题

你想计算一个数的幂。

方案

计算某数的*e*次幂，使用exp()：

```
$exp = exp(2);                       // 7.3890560989307
```

计算某数的任何次幂，使用pow()：

```
$exp = pow( 2, M_E);         // 6.5808859910179

$pow = pow( 2, 10);          // 1024
$pow = pow( 2, -2);          // 0.25
$pow = pow( 2, 2.5);         // 5.6568542494924

$pow = pow(-2, 10);   /      / 1024
$pow = pow( 2, -2);          // 0.25
$pow = pow(-2, -2.5);        // NAN (错误：不是一个数字Not a Number)
```

讨论

内置的常量M_E是一个与e近似的值。它等于2.7182818284590452354。所以exp($n)和pow(M_E,$n)是相同的。

要想通过exp()和pow()创建一个大数是很容易的；如果你记不清PHP允许的最大数（大概是1.8e308），可以参考技巧2.14中关于如何使用任意精度函数的介绍。对于exp()和pow()函数，结果超出了允许的最大数，PHP会返回INF（无穷大）；如果结果错误，则PHP会返回NAN（不是一个数字）。

参见

pow()函数的文档（*http://www.php.net/pow*）、exp()函数的文档（*http://www.php.net/exp*）以及预定义的数学常数信息（*http://www.php.net/math*）。

2.9 格式化数字

问题

你有一个数，需要以带有千分位和小数点的格式输出。例如，你想要显示浏览过某一网页的人数，或者在某次民意测验中投过票的人数。

方案

使用number_format()函数来格式化为一个整数：

```
$number = 1234.56;
print number_format($number);          // 1,235 数字已经向上舍入了
```

指定一个表示小数点位置的数字，来格式化为一个小数：

```
print number_format($number, 2);               // 1,234.56
```

讨论

number_format()函数是通过在你的数字中正确的位置插入小数点和千位分隔符来完成格式化的。如果你想手工指定这些值，可以将它们作为第三和第四个参数传入：

```
$number = 1234.56;
print number_format($number, 2, '@', '#'); // 1#234@56
```

第三个参数表示小数点，第四个参数表示千位分隔符。如果使用这一选项，就必须同时指定两个参数。

在默认的情况下，number_format()会将数字舍入到最接近的整数。如果你想保留全部数字，但事先又不知道数字的小数点后面有几位数，那么可以参照下面这样做：

```
$number = 1234.56;                  // 你的数字
list($int, $dec) = explode('.', $number);
print number_format($number, strlen($dec));
```

localeconv()函数可以提供本地化数据，也包含数字格式化信息。例如：

```
setlocale(LC_ALL, 'zh_CN');

print_r(localeconv());


Array
(
    [decimal_point] => .
    [thousands_sep] => ,
    [int_curr_symbol] => CNY
    [currency_symbol] => ¥
    [mon_decimal_point] => .
    [mon_thousands_sep] => ,
    [positive_sign] =>
    [negative_sign] => -
    [int_frac_digits] => 0
    [frac_digits] => 0
    [p_cs_precedes] => 1
    [p_sep_by_space] => 0
    [n_cs_precedes] => 1
    [n_sep_by_space] => 0
    [p_sign_posn] => 1
    [n_sign_posn] => 4
    [grouping] => Array
        (
            [0] => 3
            [1] => 3
        )

    [mon_grouping] => Array
        (
            [0] => 3
            [1] => 3
        )

)
```

可以用decimal_point、thounsands_sep和其他设置项查看该函数是如何进行本地格式化的。

参见

第19章中有关国际化和本地化的内容；localeconv()函数的文档（*http://www.php.net/localeconv*）和number_format()函数的文档（*http://www.php.net/number-format*）。

2.10 格式化货币值

问题

你有一个数，需要以带有千分位和小数点的格式输出。例如，你想显示购物车中每件商品的价格。

方案

使用带有%n格式化选项的money_format()函数，得到一个本国货币格式：

```
$number = 1234.56;
setlocale(LC_MONETARY, 'en_US');
print money_format('%n', $number);              // $1,234.56
```

要得到国际化的货币格式，需要传递%i：

```
print money_format('%i', $number);              // USD 1,234.56
```

讨论

money_format()函数通过在你的数字中插入正确的货币符号、小数点以及千位分隔符来完成格式化。它接受一个格式化字符串和要进行格式化的数字作为参数。

对于简单的格式化任务，使用%n和%i标识符分别作为国内和国际标准货币显示的标识符。

要得到正确的国家货币格式，可以像例2-3一样改变地区设置。

例2-3：用标准的格式来显示货币

```
<?php
$number = 1234.56;
setlocale(LC_MONETARY, 'en_US');
print money_format('%n', $number);     // $1,234.56
print money_format('%i', $number);     // USD 1,234.56

setlocale(LC_MONETARY, 'fr_FR');
print money_format('%n', $number);     // 1 234,56 Eu
```

```
print money_format('%i', $number);     // 1 234,56 EUR

setlocale(LC_MONETARY, 'it_IT');
print money_format('%n', $number);     // Eu 1.235
print money_format('%i', $number);     // EUR  1.235
?>
```

如果没有设置地区，这个函数会返回你所提供的同一字符串。要了解本地化设置的更多信息，请参考第19章。

也可以使用printf-like格式化选项，包括（—— 用于将负数包含在括号当中，和! —— 用于消除货币符号，如例2-4所示。

例2-4：用自定义的格式显示货币

```
<?php
$number = -1234.56;
setlocale(LC_MONETARY, 'en_US');
print money_format('%n', $number);    // -$1,234.56

print money_format('%(n', $number);   // ($1,234.56)

print money_format('%!n', $number);   // -1,234.56
?>
```

可以在*http://www.php.net/money-format*查到一个包含左、右精度，填充字符和禁用分组的完整选项列表。

这个函数使用了Unix底层的strfmon()系统函数，所以对于Windows系统来说，是无效的。

要了解更多有关货币格式的内容，包括一个针对Windows系统的替代算法，请参见技巧19.6。

参见

技巧19.6；money_format()函数的文档（*http://www.php.net/money-format*）。

2.11 正确地打印复数

问题

你想要基于一个变量的值决定单词是否显示为复数的形式。例如，基于在一个搜索中找到的匹配项目的数量返回相应的单数或复数文本。

方案

使用一个条件表达式：

```
$number = 4;
print "Your search returned $number " . ($number == 1 ? 'hit' : 'hits') . '.';

Your search returned 4 hits.
```

讨论

如果写成下面一行这样，会稍微短一些：

```
print "Your search returned $number hit" . ($number == 1 ? '' : 's') . '.';
```

但是，对于一些不固定的复数形式，例如“person”相对于“people”，还是替换整个词比替换单个字符更清楚一些。

另一种选择是使用一个函数来处理所有复数化操作，如例2-5中所示的pc_may_pluralize()函数。

例2-5：pc_may_pluralize()函数

```
<?php
function pc_may_pluralize($singular_word, $amount_of) {

    // 特殊复数的数组
    $plurals = array(
        'fish' => 'fish',
        'person' => 'people',
    );

    // 只有一个
    if (1 == $amount_of) {
        return $singular_word;
    }

    // 超过一个，而且是特殊的复数形式
    if (isset($plurals[$singular_word])) {
        return $plurals[$singular_word];
    }

    // 超过一个，而且是标准的复数：在单词后面加“s”
    return $singular_word . 's';
}
?>
```

下面是几个例子：

```
$number_of_fish = 1;
print "I ate $number_of_fish " . pc_may_pluralize('fish', $number_of_fish) . '.';
```

```
$number_of_people = 4;
print 'Soylent Green is ' . pc_may_pluralize('person', $number_of_people) . '!';

I ate 1 fish.
Soylent Green is people!
```

如果你在代码中设计了多处以复数形式表示的单词，那么使用类似pc_may_pluralize()这样的函数就可以增加代码的可读性。使用这个函数时，将单词的单数形式作为其第一个参数，将数目作为第二个参数即可。在这个函数内部，有一个很大的数组——$plurals，它保存着所有特殊单词的复数形式。如果$amount是1，这个函数就返回最初的单词。如果大于1，而且单词的复数是特殊形式，那么就返回对应的复数形式。但默认情况下，只在单词的末尾加“s”来表示复数形式。

2.12 计算三角函数

问题

你想使用三角函数，例如正弦、余弦和正切。

方案

PHP支持许多三角函数，如sin()、cos()和tan()等：

```
$cos = cos(2.1232);
```

也可以使用它们的反函数asin()、acos()和atan()：

```
$atan = atan(1.2);
```

讨论

这些函数假设所有的角度都以弧度而不是度数的形式表示（如果有问题，请参考技巧2.13中的相关介绍）。

函数atan2()接受两个变量$x和$y，并计算atan($x/$y)。然而之所以它总是能返回正确的符号，是因为它在查找结果所在的象限时能够同时使用两个参数。

若要计算正割、余割和余切，则需要手工来计算sin()、cos()和tan()的倒数：

```
$n = .707;
```

```
$secant    = 1 / sin($n);
$cosecant  = 1 / cos($n);
$cotangent = 1 / tan($n)
```

还可以使用双曲线函数，如sinh()、cosh()和tanh()，当然也有相应的asinh()、cosh()和tanh()函数。然而，Window系统不支持这些逆函数。

参见

技巧2.13中有关如何完成度数而不是弧度的三角操作；sin()函数的文档（*http://www.php.net/sin*），cos()函数的文档（*http://www.php.net/cos*），tan()函数的文档（*http://www.php.net/tan*），asin()函数的文档（*http://www.php.net/asin*），acos()函数的文档（*http://www.php.net/acos*），atan()函数的文档（*http://www.php.net/atan*）和atan2()函数的文档（*http://www.php.net/atan2*）。

2.13 用度数而不是弧度来度量三角

问题

你想在调用三角函数时使用度数。

方案

在输入和输出时使用deg2rad()和rad2deg()函数：

```
$cosine = cos(deg2rad($degree));
```

讨论

根据定义，360°等于2π弧度，所以通过手工方式也很容易将两种单位相互转换。但是，这些函数使用了PHP内部的π值，因而可以保证得到高精度的结果。如果因为其他计算需要访问这些数值，可以使用常数M_PI，其值为3.14159265358979323846。

需要说明的一点是，PHP对度数没有提供内置的支持，是出于特性的考虑，而非程序的缺陷。

参见

技巧2.12中有关三角函数的基本概念；deg2rad()函数的文档（*http://www.php.net/deg2rad*）和rad2deg()函数的文档（*http://www.php.net/rad2deg*）。

2.14 处理极大数或极小数

问题

你需要使用PHP内置的极大（或极小）的浮点型数。

方案

使用BCMath或者GMP库。

使用BCMath：

```
$sum = bcadd('1234567812345678', '8765432187654321');

// $sum 现在是字符串 '9999999999999999'
print $sum;
```

使用GMP：

```
$sum = gmp_add('1234567812345678', '8765432187654321');

// $sum 现在是一个GMP 资源，而不是一个字符串，需要用gmp_strval()函数转换
print gmp_strval($sum);
```

讨论

BCMath库使用很方便。只需将数字作为字符串传入，函数就能返回两个数的和（或者差、积等）。然而，使用BCMath库进行数字操作的范围仅限于基本的算术运算。

另外一个可用的库是GMP库。多数GMP函数库的成员都接受整数和字符串作为参数，但最好是传递数字作为资源 —— 其本质是指向数字的指示器。因此，不同于BCMath库的函数（它们返回字符串），GMP函数只返回资源。然后，可以把这个资源传递给任何GMP函数，让这个资源来充当数字。

唯一的不足是，当你想要查看或者通过一个非GMP函数使用这个资源时，你还需要明确地使用gmp_strval()或者gmp_intval()函数对其进行转换。

GMP函数对可接受的参数类型很开放。例如，我们来看一下例2-6。

例2-6：通过GMP数值做加法。

```
<?php
$four = gmp_add(2, 2);                          // 可以传递整数
$eight = gmp_add('4', '4');                     // 或者字符串
$twelve = gmp_add($four, $eight);               // 或者GMP 资源
print gmp_strval($twelve);                      // 输出 12
?>
```

而且，不仅能通过GMP数值来做加法，还能通过其实现对一个数的升幂操作、快速地计算大阶乘、找出最大公因数（GCD）等其他常见的数学问题的计算，见例2-7。

例2-7：通过GMP库计算常见的数学问题

```
<?php
// 对一个数进行升幂操作
$pow = gmp_pow(2, 10);              // 1024

// 非常快地计算大阶乘
$factorial = gmp_fact(20);          // 2432902008176640000

// 找到 GCD
$gcd = gmp_gcd (123, 456);          // 3

// 其他常见的数学问题
$legdendre = gmp_legendre(1, 7);    // 1
?>
```

BCMath和GMP库并不是所有PHP必须启用的配置。BCM库是同PHP绑定的，所以它可能是有效的。然而，GMP库并没有与PHP绑定，所以你还需要下载并安装这个库，然后还要通过配置告诉PHP可以使用它。通过检查functioin_defined('bcadd')和function_defined('gmp_init')的返回值，可以知道BCMath和GMP库是否有效。如果你使用的是Windows系统，则必须在PHP 5.1或以上版本中才能使用GMP库。

另一个高精度数学运算的选择是PECL的big_int库，见例2-8。

例2-8：使用big_int库实现加法

```
<?php
$two  = bi_from_str('2');
$four = bi_add($two, $two);
print bi_to_str($four)              // Prints 4

// 非常快地计算大阶乘
$factorial = bi_fact(20);           // 2432902008176640000
?>
```

这个库比BCMath要快，而且几乎与GMP库一样强大。不过，GMP库基于LGPL许可，而big_int库则是基于BSD许可的。

参见

BCMath库的文档（*http://www.php.net/bc*），big_int库的文档（*http://pecl.php.net/big_int*）以及GMP库的文档（*http://www.php.net/gmp*）。

2.15 在不同进制间转换

问题

你想将一个数在不同的进制之间进行转换。

方案

使用base_convert()函数：

```
$hex = 'a1';                                    // 十六进制数 (base 16)
// 将十六进制转换为十进制
$decimal = base_convert($hex, 16, 10);          // $decimal 现在是十进制的161
```

讨论

base_convert()函数用于将一个基于某种进制的字符串转换成基于另外一种进制的字符串。它可以转换的进制范围是二~三十六（包括 2 和 36），用字符a~z作为附加的符号表示十以上的进制。其接受的第一个参数是要进行转换的数，第二个和第三个参数分别是该数当前的进制和要转换成的进制。

也有一些专门针对十进制与其他进制数相互转换的函数，其中最常用的进制是二，八和十六进制。对应的函数分别是bindec()和decbin()、octdec()和decoct()，以及hexdec()和dechex()：

```
// 转换成10进制数
print bindec(11011);          // 27
print octdec(33);             // 27
print hexdec('1b');           // 27

// 将十进制数转换成其他进制数
print decbin(27);             // 11011
print decoct(27);             // 33
print dechex(27);             // 1b
```

另外一个可供选择的函数是printf()，这个函数可以将十进制数转换成二进制、八进制

和十六进制数，其特点是可以接受多种格式的参数，比如，带前导零的数和在十六进制数中使用大写或小写字母。

例如，假设你想输出HTML颜色值：

```
printf('#%02X%02X%02X', 0, 102, 204); // #0066CC
```

参见

base_convert()函数的文档（*http://www.php.net/base-convert*）和sprintf()函数可用的格式选项（*http://www.php.net/sprintf*）。

2.16 非十进制数的计算

问题

你想进行八进制或者十六进制等非十进制数之间的运算。例如，你想要以十六进制数的形式计算Web安全色。

方案

给相应的数字加上前导字符，以便使PHP知道它不是一个十进制的数。下面这几个数都是相等的：

```
0144  // base 8
 100  // base 10
0x64  // base 16
```

下面示范了如何用十六进制符号计算十进制的1 ~ 15：

```
for ($i = 0x1; $i < 0x10; $i++) { print "$i\n"; }
```

讨论

即使是在for循环中使用十六进制数，在默认的情况下，所有数值仍然是以十进制输出的。换句话说，以上方案中的代码不会输出“..., 8, 9, a, b, ...”。如果想以十六进制的形式输出，就需要使用技巧2.15中提到的函数。请看下面的例子：

```
for ($i = 0x1; $i < 0x10; $i++) { print dechex($i) . "\n"; }
```

对于多数计算问题，还是使用十进制比较简单。但是，有时候必须转换成其他进制形式才更符合逻辑。比如在使用216个Web安全色的时候，每个Web安全色值都是RRGGBB形式的，其中，RR代表红色，GG代表绿色，而BB代表蓝色。实际上，其中每种颜色都是用两位从0～F的十六进制数来表示的。

Web安全色的特别之处在于，RR、GG和BB都必须是下列六组数字之一：00，33，66，99，CC和FF（对应着十进制的0，51，102，153，204和255）。所以003366是Web安全色，而112233则不是。Web安全色在256色显示中不用进行抖动处理。

当需要创建这些数的列表时，就要在例2-9所示的三重循环中使用十六进制符号，来加强列表的十六进制特征。

例2-9：输出全部十六进制的Web安全色值

```
<?php
for ($rr = 0; $rr <= 0xFF; $rr += 0x33)
    for ($gg = 0; $gg <= 0xFF; $gg += 0x33)
        for ($bb = 0; $bb <= 0xFF; $bb += 0x33)
            printf("%02X%02X%02X\n", $rr, $gg, $bb);
?>
```

这个例子中的循环计算了所有可能的Web安全色。但是，这里没有用十进制数来表示步长，而是使用了十六进制符号，因为这样做可以增强数值与十六进制的颜色值之间的联系。然后，分别将这些值用printf()函数格式化为大写的、至少两位长的十六进制数输出。如果是一个数的话，则在输出时前置一个零作为前导符。

参见

技巧2.15中有关在不同进制间相互转换的详细介绍；由Jennifer Niederst Robbins编著、O'Reilly出版的《Web Design in a Nutshell》中的第3章“Web Design Principles for Print Designers”。

2.17 计算球面坐标系中两点间的距离

问题

你想要计算地球表面坐标系中两个点之间的距离。

方案

使用例2-10所示的pc_sphere_distance函数。

例2-10：计算两点之间的距离。

```
<?php
function pc_sphere_distance($lat1, $lon1, $lat2, $lon2, $radius = 6378.135) {
  $rad = doubleval(M_PI/180.0);

  $lat1 = doubleval($lat1) * $rad;
  $lon1 = doubleval($lon1) * $rad;
  $lat2 = doubleval($lat2) * $rad;
  $lon2 = doubleval($lon2) * $rad;

  $theta = $lon2 - $lon1;
  $dist = acos(sin($lat1) * sin($lat2) + cos($lat1) * cos($lat2) * cos($theta));
  if ($dist < 0) { $dist += M_PI; }

    return $dist = $dist * $radius; // Default is Earth equatorial radius in kilometers
}

// NY（纽约），NY（纽约） (10040)
$lat1 = 40.858704;
$lon1 = -73.928532;

// SF（旧金山），CA（加拿大）(94144)
$lat2 = 37.758434;
$lon2 = -122.435126;

$dist = pc_sphere_distance($lat1, $lon1, $lat2, $lon2);
printf("%.2f\n", $dist * 0.621); // 格式化并转换成英里
?>

2570.18
```

讨论

由于地球表面并不是平坦的，所以不可能使用标准的毕达哥拉斯距离公式计算出地球上面两个位置之间的距离，而是必须使用大圆（Great Circle）算法，比如在函数pc_sphere_distance()中所用到的算法。

将两个位置各自的经度和纬度作为该函数的前四个参数。首先是起始地的经度和纬度，然后是目的地的经度和纬度。返回的值是两个位置之间距离的公里数：

```
// NY（纽约），NY（纽约） (10040)
$lat1 = 40.858704;
$lon1 = -73.928532;

// SF（旧金山），CA（加拿大）(94144)
$lat2 = 37.758434;
$lon2 = -122.435126;

$dist = pc_sphere_distance($lat1, $lon1, $lat2, $lon2);
printf("%.2f\n", $dist * 0.621); // 格式化并转换成英里
```

这段代码用于计算纽约到旧金山之间的距离，然后将距离转换成英里，并将其格式化成带两位小数的数值，最后输出结果。

因为地球并不是严格意义上的球体，所以这样计算出来的结果会稍有误差，误差上限为0.5%。

pc_sphere_distance()函数还接受一个可选的球面半径作为第五个参数。这样，你就可以利用它来计算火星上两点之间的距离了：

```
$martian_radius = 3397;
$dist = pc_sphere_distance($lat1, $lon1, $lat2, $lon2, $martian_radius);
printf("%.2f\n", $dist * 0.621); //格式化并转换成英里
```

参见

技巧2.12中三角函数的基本介绍；维基麦（*http://en.wikipedia.org/wiki/Earth_radius*；）中有关地球半径的条目，以及本书合著者之一David Sklar的文章“Trip Mapping with PHP”（*http://www.onlamp.com/pub/a/php/2002/11/07/php_map.html*）。

第3章

日期和时间

3.0 概述

显示、操纵日期和时间一开始看起来简单，但随着用户复杂性和多样性的增加，这些处理任务也会变得困难起来。你的用户是不是来自不同的时区？很有可能。除非你是在建设一个局域网中的应用或者是一个专门针对某一特定地理区域内的读者的网站。你的读者是不是曾经被像“2002-07-20 14:56:34 EDT”这样的时间戳吓跑过？他们是不是应该能够平静地面对像“Saturday July 20, 2000 (2:56 P.M.)”这样的表现形式？要计算从今天上午10点到晚上7点有几个小时相当简单。但要是计算从今天凌晨3点到下个月第一天的中午共多少个小时呢？技巧3.5节和3.6节讨论了如何计算日期之间差的问题。

如果再考虑日光节约（或夏令）时间（daylight saving time，DST），这些计算和处理任务就更复杂了。因为DST，有的时间可能会不存在（在美国大部分地区，春天中每天的凌晨2点到3点），而有的时间可能会出现两次（在美国大部分地区，秋天中每天的凌晨1点到2点）。你的一部分用户可能生活在实行DST的地区，但另一部分则不是。技巧3.11节和3.12节提供了处理时区与DST问题的方法。

对于计划性的时间处理，由于有了两个约定变得更简单了。第一个约定，是将时间内在地看成等同于国际标准时间（小型UTC也称为GMT，即格林威治标准时间），这个时区家族中的元老并不涉及DST或者叫夏令时的惯例。它是位于0纬度处的时区，而其他所有时区都是以相对于它的偏移距离（包括正偏移和负偏移）表示的。第二个约定，是不将时间看成一组表示不同的月、日、年、分、秒等的值，而是看成自Unix纪元——1970年1月1日午夜（当然是UTC时间）——开始经过的秒数。这就使计算时间间隔更容易了，而且PHP中也有许多函数可以帮你把纪元时间戳转换成人类可读取的时间表现形式。

函数mktime()会根据一段给定的时间生成纪元时间戳，而date()函数会根据给定的纪元时间戳，返回一个格式化后的时间字符串。例3-1中使用这两个函数来计算出了1986年新年那一天是星期几。

例3-1：使用mktime()和date()函数

```
<?php
$stamp = mktime(0,0,0,1,1,1986);
print date('l',$stamp);
?>
```

例3-1的输出结果：

```
Wednesday
```

在例3-1中，mktime()函数返回1986年1月1日午夜的纪元时间戳。date()函数中l格式化字符的含义是让函数返回与这个给定的时间戳对应的那一天的完整的星期名称。技巧3.4节详细介绍了可以在date()函数中使用的格式化字符。

本书中，术语“纪元时间戳”指的是自Unix纪元开始经过的秒数。“时间部件”（或“日期部件”、或者“时间和日期部件”）的含义是指一组诸如日、月、年、时、分、秒这样的时间或日期部件。“格式化的时间字符串”（或“格式化的日期字符串”等）意味着一个包含着一些由特定的时间和日期部件组成的字符串。例如：“2002-03-12”，“Wednesday，11:23 A.M.”或“February 25”。

如果你把纪元时间戳作为你的内部时间表示法，就能够避免Y2K问题，因为946702799（1999-12-31 23:59:59 UTC）和946702800（2000-01-01 00:00:00 UTC）的差与其他任何两个时间戳之间的差没有任何区别。但是，你可能会碰到Y2038问题。2038年1月19日凌晨3:14:07（UTC）从1970年1月1日午夜开始算是2147483647秒。2147483647有什么特别的地方吗？它是2^{31}-1，是32位系统表示带符号整数的情况下能够表示的最大的整数（第32位表示符号，即符号位）。

有解决方案吗？就是在2038年1月19日之前的某一天，确保花钱升级你所用的硬件，比如说用64位来存储时间的系统。这样就等于你购买了另外2920亿年（只用39位就足以让你维持大约10680年，足以渡过可以与地球的冷聚变因子和超光速空间站相提并论的Y10K问题的冲击）。2038年看起来似乎离现在还很遥远，但正是同样的短视导致了COBOL程序员在1950年代和1960年代制造了2000年问题。绝不能在同一个地方跌倒两次！

3.1 查出当前的日期和时间

问题

你想要知道当前的时间和日期。

方案

使用strftime()或date()函数来得到一个格式化的字符串，像例3-2那样。

例3-2：查出当前的日期和时间

```
<?php
print strftime('%c');
print "\n";
print date('r');
?>
```

例3-2的输出结果：

```
Wed May 10 18:29:59 2006
Wed, 10 May 2006 18:29:59 -0400
```

如果想要的是时间部件，那么使用getdate()或localtime()函数。例3-3显示了如果使用这两个函数。

例3-3：查出时间部件。

```
<?php
$now_1 = getdate();
$now_2 = localtime();
print "{$now_1['hours']}:{$now_1['minutes']}:{$now_1['seconds']}\n";
print "$now_2[2]:$now_2[1]:$now_2[0]";
```

例3-3的输出结果：

```
18:23:45
18:23:45
```

讨论

函数strftime()和date()可以生成多种格式的时间和日期字符串。这两个函数会在技巧3.4节中详细讨论。而localtime()和getdate()函数，则返回一个数组，这个数组中的元素分别保存着指定的日期和时间的特定部分。

getdate()函数返回的关联数组中的“键/值对”如表3-1所示。

表3-1：由getdate()函数返回数组的构成

键	值
seconds	秒数
minutes	分钟数

表3-1：由getdate()函数返回数组的构成（续）

键	值
hours	小时数
mday	一月中的第几天
wday	一周中的第几天，数字值（周日是0，周六是6）
mon	月份
year	年份，数字（4位）
yday	一年中的第几天，数字（例如：299）
weekday	一周中的第几天，文本（例如：“Friday”）
month	月份，文本，全称（例如：“January”）
0	自纪元起的秒数（即time()函数的返回值）

例3-4：用getdate()函数输出月、日和年

```
<?php
$a = getdate();
printf('%s %d, %d',$a['month'],$a['mday'],$a['year']);
?>
```

例3-4 的输出结果：

```
May 5, 2007
```

给getdate()函数传递一个纪元时间戳作为参数，可以使返回的数组中包含符合这个时间戳的本地时间值。例3-5显示了位于时间戳163727100的月、日和年数。

例3-5：以特定的时间戳调用getdate()函数

```
<?php
$a = getdate(163727100);
printf('%s %d, %d',$a['month'],$a['mday'],$a['year']);
?>
```

例3-5 的输出结果：

```
March 10, 1975
```

函数localtime()返回一个包含时间和日期部件的数组。它同样以一个纪元时间戳作为可选的第一个参数，也将一个布尔值作为可选的第二个参数。如果这第二个参数是true，localtime()会返回一个关联数组而不是一个数字索引的数组。这个关联数组的键与C函数localtime()返回的tm_struct结构的成员相同，如表3-2所示。

表3-2：由函数localtime()返回数组的构成

数字位置	键	值
0	tm_sec	秒
1	tm_min	分
2	tm_hour	时
3	tm_mday	一月中的第几天
4	tm_mon	一年中的月份（0表示1月）
5	tm_year	自1900年起经过的年数
6	tm_wday	一周中的第几天（表示星期六）
7	tm_day	一年中的第几天
8	tm_isdst	夏令时是否有效？

例3-6显示了如何使用localtime()函数以“月/日/年”的格式输出今天的日期。

例3-6：使用localtime()函数

```
<?php
$a = localtime();
$a[4] += 1;
$a[5] += 1900;
print "$a[4]/$a[3]/$a[5]";
```

例3-6 的输出结果：

```
6/23/2006
```

由于localtime()从0开始算一月份，而我们希望如实地显示当前的月份，所以在月份数输出之前要加1。同样地，年份要加上1900是因为localtime()函数用0表示1900年。

参见

strftime()函数的文档（*http://www.php.net/strftime*），date()函数的文档（*http://www.php.net/date*），getdate()函数的文档（*http://www.php.net/getdate*）和localtime()函数的文档（*http://www.php.net/localtime*）。

3.2 将时间和日期部件转换为纪元时间戳

问题

你想知道与一组时间和日期部件对应的纪元时间戳。

方案

如果这些时间和日期部件都来自本地时区，可以像例3-7中所示的那样使用mktime()函数。

例3-7：获得特定的纪元时间戳

```
<?php
// 本地时间1975年3月10日晚上7:45:03
$then = mktime(19,45,3,3,10,1975);
?>
```

如果这些时间和日期部件是GMT时间，则要像例3-8所示的那样使用gmmktime()。

例3-8：获得基于GMT的特定纪元时间戳

```
<?php
// GMT时间1975年3月10日晚上7:45:03s
$then = gmmktime(19,45,3,3,10,1975);
?>
```

讨论

mktime()和gmmktime()这两个函数都是接受一个日期和时间部件（时、分、秒、月、日、年）作为参数，并返回相应的Unix纪元时间戳。mktime()把部件视为本地时间，gmmktime()则视部件为以UTC形式表示的日期和时间。只有当时间不超出纪元之外时，这两个函数才会返回切合实际的结果。大多数系统都以32位的有符号整数来存储纪元时间戳，所以“不超出纪元”意味着处于UTC时间1901年12月13日晚上8:45:51和UTC时间2038年1月19日凌晨3:14:07之间。

在例3-9中，$stamp_future设为了2012年6月4日下午3:25分的纪元时间戳。这个纪元时间戳可以通过strftime()函数回馈生成一个格式化时间字符串。

例3-9：使用纪元时间戳

```
<?php
$stamp_future = mktime(15,25,0,6,4,2012);

print $stamp_future;
print strftime('%c',$stamp_future);
?>
```

例3-9的输出结果：

```
1338837900
Mon Jun 4 15:25:00 2012
```

由于例3-9是在一个设置为EDT（比GMT晚4个小时）的计算机中调用mktime()函数，所以如果使用gmmktime()函数来生成时间戳就会比这个时间戳少14400秒（4小时），如例3-10所示。

例3-10：纪元时间戳和gmmktime()函数

```
<?php
$stamp_future = gmmktime(15,25,0,6,4,2012);

print $stamp_future;
print strftime('%c',$stamp_future);
?>
```

例3-10的输出结果：

```
1338823500
Mon Jun 4 11:25:00 2012
```

将gmmktime()生成的纪元时间戳回馈给strftime()函数所生成的格式化的时间字符串同样也少4个小时。

在PHP 5.1.0以前，mktime()和gmmktime()函数可以接受一个可选的布尔值作为第七个参数，用于表示DST标记（如果要考虑夏令时间，设为1，否则设为0）。在PHP 5.1.0之后，是否考虑夏令时则由当前默认的有效时区来控制，这个时区通过date_default_timezone_set()函数来设定。

参见

技巧3.3中如何将纪元时间戳转换回时间和日期部件的内容。mktime()函数的文档（*http://www.php.net/mktime*）和gmmktime()函数的文档（*http://www.php.net/gmmktime*），以及date_default_timezone_set()函数的文档（*http://www.php.net/date_default_timezone_set*）。

3.3 将纪元时间戳转换为时间和日期部件

问题

你想得到与特定的纪元时间戳对应的一组时间和日期部件。

方案

把一个纪元时间戳传递给函数getdate()——$time_parts=getdate(163727110)。

讨论

由getdate()函数返回的时间部件的细节见表3-1。这些时间部件都是本地时间。如果你想得到与一个特定的纪元时间戳对应的另一个时区的时间部件，请参考技巧3.11。

参见

技巧3.2中将时间和日期部件转换回一个纪元时间戳的方法。技巧3.11中关于处理不同时区的内容。getdate()函数的文档（*http://www.php.net/getdate*）。

3.4 以特定的格式打印日期和时间

问题

你想以特定的方式输出格式化的时期和时间。

方案

使用date()或strftime()函数，如例3-11所示。

例3-11：使用date()和strftime()函数

```
<?php
print strftime('%c');
print date('m/d/Y');
?>
```

例3-11会输出类似下面的结果：

```
Mon Dec  3 11:31:08 2007
12/03/2007
```

讨论

date()和strftime()函数都具有很强的通用性，它们都可以接受多种部件作为参数从而生成格式化的时间字符串。针对这两个函数的格式化字符如表3-3所示。其中Windows一栏中所列出的是在Winodows系统中使用strftime()函数时是否支持相应的字符。

表3-3：适用于date()和strftime()函数的格式化字符

类型	strftime()	date()	描述	范围	Windows
小时	%H	H	小时，数字，24小时制	00 ~ 23	是
小时	%I	h	小时，数字，12小时制	01 ~ 12	是
小时	%k		小时，数字，24小时制。一位数字前会加上一个空格	0 ~ 23	否
小时	%l		小时，数字，12小时制。一位数字前会加上一个空格	1 ~ 12	否
小时	%p	A	当前时区的AM或PM标志——大写		是
小时	%P	a	当前时区的am 或pm标志——小写		否
小时		G	小时，数字，24小时制。无前导零	0 ~ 23	否
小时		g	小时，数字，12小时制。无前导零	0 ~ 1	否
分钟	%M	i	分钟，数字	00 ~ 59	是
秒	%S	s	秒，数字	00 ~ 61[a]	是
日	%d	d	每月中的第几天，数字	01 ~ 31	是
日	%e		每月中的第几天，数字，一位数字前会加上一个空格	1 ~ 31	否
日	%j	z	一年中的第几天，数字	对strftime()是001 ~ 366 对for date()是0 ~ 365	是
日	%u	N	一周中的第几天，数字（周日是1）	1 ~ 7	否
日	%w	w	一周中的第几天，数字（周日是0）	0 ~ 6	是
日		j	一月中的第几天，数字。无前导零	1 ~ 31	否
日		S	表示一月中第几天的英语顺序后缀	"st," "th," "nd," "rd"	否
周	%a	D	简写的星期名，当前地区的文本形式		是

表3-3：适用于date()和strftime()函数的格式化字符（续）

类型	strftime()	date()	描述	范围	Windows
周	%A	l	完整的星期名，当前地区的文本形式		是
周	%U		一年中的第几周，数字。周日是第一周的第一天	00 ~ 53	是
周	%V	W	ISO 8601:1988标准规定的一年中的第几周，数字。周1表示一年中的第一周，该周至少有四天位于当年中。周一是该周的第一天	01 ~ 53	否
周	%W		一年中的第几周，数字。周一是第一周的第一天	00 ~ 53	是
月	%B	F	完整的月份名，当前地区的文本形式		是
月	%b	M	简写的月份名。当前地区的文本形式		是
月	%h		与%b相同		否
月	%m	m	月份，数字	01 ~ 12	是
月		n	月份，数字。无前导零	1 ~ 12	否
月		t	一月中的天数，数字	28, 29, 30, 31	否
年	%C		世纪，数字	00 ~ 99	否
年	%g		类似%G，但不带世纪	00 ~ 99	否
年	%G	o	ISO 8601标准中规定的带世纪的年份，数字。这个四位数的年份对应着ISO的周数，除了属于上一年或下一年的 ISO 周数由相应年份占有外，其他与%y相同		否
年	%y	y	不带世纪的年份，数字	00 ~ 99	是
年	%Y	Y	年份，数字。包括世纪		是
年		L	闰年标志 （1表示是）	0, 1	否
时区	%z	O	基于GMT的小时偏移，±HHMM （例如：-0400，+0230）	-1200 ~ +1200	是，但类%Z

表3-3：适用于date()和strftime()函数的格式化字符（续）

类型	strftime()	date()	描述	范围	Windows
时区		P	时区偏移，包含冒号（例如：-04:00，+02:30）	-12:00 ~ +12:00	
时区	%Z	T	时区，名称或简写，文本		是
时区		e	时区标识符，例如：America/New_York		
时区		I	夏时制标志（1表示是）	0, 1	否
时区		Z	基于 GMT 的秒数偏移。GMT 以西是负值，GMT 以东是正值	-43200 ~ 43200	否
复合	%c		当前地区的标准日期和时间格式		是
复合		c	ISO 8601 标准格式的日期和时间		是
复合	%D		与%m/%d/%y相同		否
复合	%F		与%Y-%m-%相同		否
复合	%r		当前地区时间的A.M.或P.M.表示法		否
复合	%R		当前地区时间的24小时制表示法		否
复合	%T		24小时制表示法的时间（与 %H:%M:%S 相同）		否
复合	%x		当前地区的标准日期格式（没有时间）		是
复合	%X		当前地区的标准时间格式（没有日期）		是
复合		r	RFC 822 格式的日期（例如："Thu, 22 Aug 2002 16:01:07 +0200"）		否
其他	%s	U	从纪元时间开始的秒数		否
其他		B	样本互联网时间		否
格式化	%%		百分号直接量字符（%）		是
格式化	%n		换行符		否
格式化	%t		制表符		否

a. *秒的范围扩展到61是为了解决闰秒的问题*

PHP 5.0.0在date()函数中增加了格式化字符c。PHP 5.1.0在date()函数中增加了字符N、o和e。到PHP 5.1.3时又增加了字符P。

这两个函数的第一个参数都是格式字符，而第二个参数则是一个纪元时间戳。如果省略了第二个参数，这两个函数都默认采用当前的日期和时间。date()和strftime()函数都是在本地时间基础上使用的，但它们也都分别有一个基于UTC时间的版本（gmdate()和gmstrftime()）。

在PHP 5.1.1及以后版本中，增加了一些表示格式字符串的方便的常量，可以传递到date()函数中，以返回一些常见的日期格式。这些常量都列在了表3-4中。

表3-4：date()函数可以使用的常量

常量	值	例子	用法
DATE_ATOM	Y-m-d\TH:i:sO	2010-12-03 T06:23:39-0500	Atom 源聚合3.3节中的格式（*http://www.atomenabled.org/developers/syndication/atom-format-spec.php#date.constructs*）
DATE_COOKIE	D, d M Y H:i:s T	Fri, 03 Dec 2010 06:23:39 EST	HTTP Cookies（如*http://wp.netscape.com/newsref/std/cookie_spec.html* 中所定义相同的）
DATE_ISO8601	Y-m-d\TH:i:sO	2010-12-03 T06:23:39-0500	ISO 8601（如*http://www.w3.org/TR/NOTE-datetime*所讨论的）
DATE_RFC822	D, d M Y H:i:s T	Fri, 03 Dec 2010 06:23:39 EST	电子邮件（如*http://www.faqs.org/rfcs/rfc822.html*中所定义的）
DATE_RFC850	l, d-M-y H:i:s T	Friday, 03-Dec-10 06:23:39 EST	新闻组消息（如*http://www.faqs.org/rfcs/rfc850.html*中所定义的）
DATE_RFC1036	l, d-M-y H:i:s T	Friday, 03-Dec-10 06:23:39 EST	新闻组消息（如*http://www.faqs.org/rfcs/rfc1036.html*）
DATE_RFC1123	D, d M Y H:i:s T	Fri, 03 Dec 2010 06:23:39 EST	As defined in *http://www.faqs.org/rfcs/rfc1123.html*中所定义的）
DATE_RFC2822	D, d M Y H:i:s O	Fri, 03 Dec 2010 06:23:39 -0500	电子邮件（如*http:// www.faqs.org/rfcs/rfc2822.html*中所定义的）
DATE_RSS	D, d M Y H:i:s T	Fri, 03 Dec 2010 06:23:39 EST	RSS 源（如 *http://blogs.law.harvard.edu/tech/rss* 中所定义的）
DATE_W3C	Y-m-d\TH:i:sO	2010-12-03 T06:23:39-0500	与 DATE_ISO8601相同

这些date()可以使用的格式化字符是PHP专有的，而strftime()函数使用的则是C库strftime()函数。这可能会使得从其他编程语言中转到PHP上来的人更容易理解这个函数，但也会使这个函数的行为在多种平台下使用时暴露出一些轻微的差异。Windows就不支持大多数Unix系统所支持的strftime()函数的格式化命令。同样地，strftime()函数需要其每一个格式化字符都前置一个%（想想printf()函数），所以它很容易生成内部带有插值的时间和日期值的字符串。

例如，对于“12:49 P.M. on July 15，2002”，代码需要输出：

```
It's after 12 pm on July 15, 2002。
```

通过strftime()可以这样实现：

```
print strftime("It's after %I %P on %B %d, %Y");
```

通过date()则需要这样来实现：

```
print "It's after ".date('h a').' on '.date('F d, Y');
```

格式字符串中与日期无关的字符对于strftime()函数非常合适，因为这个函数是通过寻找%来决定在哪里插入适当的时间信息的。然而，date()函数中却没有这个定界符，所以唯一能塞进格式化字符串中的只有空格符和标点符号。如果你像这样把strftime()函数的格式化字符串传递给date()函数：

```
print date("It's after %I %P on %B%d, %Y");
```

你肯定不会对下面的结果感觉满意：

```
131'44 pmf31eMon, 15 Jul 2002 12:49:44 -0400 %1 %P o7 %742%15, %2002
```

用date()函数生成的时间部件可以很容易用于插值，可以先将date()函数生成的时间和日期部件组织成一个字符串，之后以一个date()函数不能解析并且也不属于你的子字符串的定界符来分离其生成的部件。然后，用这个定界符来调用explode()函数，把从date()函数中返回的各个部分保存到一个数组中，这个数组便可以方便地用于你生成输出字符串了。例3-12演示了这一过程，其中使用了|符号作为定界符。

例3-12：联合使用explode()和date()

```
<?php
$ar = explode('|',date("h a|F d, Y"));
print "It's after $ar[0] on $ar[1]";
?>
```

参见

date()函数的文档（*http://www.php.net/date*）和strftime()函数的文档（*http://www.php.net/strftime*）。在基于Unix的系统中，可以使用的系统专有的strftime()函数的选项。在Windows系统中，可以在*http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_strftime.2c_.wcsftime.asp*中查到有关strftime()函数的详细资料。

3.5 计算两个日期间的时间差

问题

你想知道两个日期间相差多长时间。例如，你想告诉用户他最后登录你的网站至今已经过了多长时间了。

方案

先把两个日期转换成纪元时间戳，并计算两个时间戳的差值。例3-13中将这个差值分解成了周、天、小时、分钟和秒。

例3-13：计算两个日期间的时间差

```
<?php
// 1965年5月10日晚上7:32:56
$epoch_1 = mktime(19,32,56,5,10,1965);
// 1962年11月20日凌晨4:29:11
$epoch_2 = mktime(4,29,11,11,20,1962);

$diff_seconds  = $epoch_1 - $epoch_2;
$diff_weeks    = floor($diff_seconds/604800);
$diff_seconds -= $diff_weeks   * 604800;
$diff_days     = floor($diff_seconds/86400);
$diff_seconds -= $diff_days    * 86400;
$diff_hours    = floor($diff_seconds/3600);
$diff_seconds -= $diff_hours   * 3600;
$diff_minutes  = floor($diff_seconds/60);
$diff_seconds -= $diff_minutes * 60;

print "The two dates have $diff_weeks weeks, $diff_days days, ";
print "$diff_hours hours, $diff_minutes minutes, and $diff_seconds ";
print "seconds elapsed between them.";
?>
```

例3-13的输出结果：

```
The two dates have 128 weeks, 6 days, 14 hours, 3 minutes,
and 45 seconds elapsed between them.
```

注意，上例中并没有把时间差值分解成比周更大的时间单位（例如：月或者年），这是因为更大的时间单位具有可变的长度，所以不可能准确地表达对时差计算的结果。

讨论

这里有几个你应该知道的奇怪的情况。首先，1962年和1965年位于 Unix 纪元开始之前。幸运的是，mktime()函数在遇到这种情况时会优雅的降级（一种能够向前兼容的能力。译者注），为每个时间生成相应的负纪元时间戳。这没有问题，因为需要的并不是这两个时间戳的绝对时间值，而只是它们的差值而已。只要日期的纪元时间戳位于有符号的整数的范围之内，就能够正确地计算它们的差值。

第二，挂钟（或日历）会反映出之两个日期之间时差的轻微不同，因为这两个日期分别位于夏令时切换点的两侧。虽然时间戳相减得到的是正确的时差值，但人类感觉到的时间推移却要多1小时。比如说，在夏令时4月份的星期天凌晨，1:30 ~ 4:30之间的时差是多少？好像应该是3小时，但是这两个时间的纪元时间戳之间却只相关7200秒，即2小时。当一个本地时钟在春天向前调快1小时（或者在10月份调慢1小时）的时候，连续的时间流逝并不会引起人们的注意。所以虽然我们的时钟显示的好像是3小时，而实际上，只过了两个小时。

如果你想要测度实际经过的时间（通常都是这样），这种方法非常有效。如果你更关注这两个时间点上时钟的报时，那么可以使用儒略日（Julian days）来计算时间间隔，这种方法在3.6节中讨论。

要想告诉用户自他上次登录以来经过了多长时间，必须要知道当前登录时间和他上次最后登录时间的时差，如例3-14所示。

例3-14：计算自上次登录以来经过的时间

```
<?php
$db = new PDO('mysql:host=db.example.com', $user, $password);
$epoch_1 = time();
$st = $db->prepare("SELECT UNIX_TIMESTAMP(last_login) AS login " .
                   "FROM user WHERE id = ?");
$st->execute(array($id));
$row = $st->fetch();
$epoch_2 = $row['login'];

$diff_seconds  = $epoch_1 - $epoch_2;
$diff_weeks    = floor($diff_seconds/604800);
$diff_seconds -= $diff_weeks   * 604800;
$diff_days     = floor($diff_seconds/86400);
$diff_seconds -= $diff_days    * 86400;
$diff_hours    = floor($diff_seconds/3600);
$diff_seconds -= $diff_hours   * 3600;
```

```
$diff_minutes  = floor($diff_seconds/60);
$diff_seconds -= $diff_minutes * 60;

print "You last logged in $diff_weeks weeks, $diff_days days, ";
print "$diff_hours hours, $diff_minutes minutes, and $diff_seconds ago.";
```

参见

技巧3.6中用儒略日方式计算两个日期间时差的讨论。技巧3.10中在一个日期上增加以及减少时间的方法。MySQL的UNIX_TIMESTAMP()函数的文档（*http://www.mysql.com/doc/D/a/Date_and_time_functions.html*）。

3.6 用儒略日计算两个日期间的时间差

问题

你想要以时钟所指示的时间为准来计算两个日期间的时间差，而不是计算这两个日期间实际的时间差。

方案

使用gregoriantojd()函数来取得两个日期部件的儒略日，之后从一个儒略日中减去另一个儒略日，以得到日期差值；然后再将两个时间部件转换为秒，并计算返回结果的差，得到时间的差值。如果时间的差值小于0，则减少1天日期差值，并调整时间差值以适应前一天。例3-15显示了整个过程。

例3-15：用儒略日计算两个日期间的时间差

```
<?php
$diff_date = gregoriantojd($date_1_mo, $date_1_dy, $date_1_yr) -
             gregoriantojd($date_2_mo, $date_2_dy, $date_2_yr);
$diff_time = $date_1_hr * 3600 + $date_1_mn * 60 + $date_1_sc -
             $date_2_hr * 3600 - $date_2_mn * 60 - $date_2_sc;
if ($diff_time < 0) {
   $diff_date--;
   $diff_time = 86400 - $diff_time;
}
?>
```

讨论

用儒略日计算两个日期间的时间差可以使你在纪元秒的值域之外实现运算，同时也解决了DST时差的问题。

例3-16：以保存在数组中的日期部件来实现相同的运算

```
<?php
// 1965年5月10日晚上7:32:56
list($date_1_yr, $date_1_mo, $date_1_dy, $date_1_hr, $date_1_mn, $date_1_sc)=
    array(1965, 5, 10, 19, 32, 56);
// 1962年11月20日凌晨4:29:11
list($date_2_yr, $date_2_mo, $date_2_dy, $date_2_hr, $date_2_mn, $date_2_sc)=
    array(1962, 11, 20, 4, 29, 11);

$diff_date = gregoriantojd($date_1_mo, $date_1_dy, $date_1_yr) -
             gregoriantojd($date_2_mo, $date_2_dy, $date_2_yr);
$diff_time = $date_1_hr * 3600 + $date_1_mn * 60 + $date_1_sc -
             $date_2_hr * 3600 - $date_2_mn * 60 - $date_2_sc;
if ($diff_time < 0) {
   $diff_date--;
   $diff_time = 86400 - $diff_time;
}
$diff_weeks = floor($diff_date/7); $diff_date -= $diff_weeks * 7;
$diff_hours = floor($diff_time/3600); $diff_time -= $diff_hours * 3600;
$diff_minutes = floor($diff_time/60); $diff_time -= $diff_minutes * 60;

print "The two dates have $diff_weeks weeks, $diff_date days, ";
print "$diff_hours hours, $diff_minutes minutes, and $diff_time ";
print "seconds between them.";
?>
```

例3-16的输出结果：

```
The two dates have 128 weeks, 6 days, 15 hours, 3 minutes,
and 45 seconds between them.
```

这一方法会生成一个基于时钟指示的时间差，所以上例的结果比技巧3.5中的结果要多一个小时。5月10日正是实行夏令时的时候，而11月20日使用的则是标准时间。

gregoriantojd()包含在PHP的日历（calendar）扩展中，所以只有加载了该扩展才能使用该函数。

参见

技巧3.5中计算两个日期经过的时间差的讨论。技巧3.10中有关增加和减少时间的内容。gregoriantojd()函数的文档（*http://www.php.net/gregoriantojd*）以及有关儒略日系统的简要介绍。

3.7 找到周、月或者年中的某一天

问题

你想找到一年中的某一天，或者某个星期，或者一周、一个月中的某一天。例如，你想要在每个星期一或者每个月的第一天输出特别的信息。

方案

在date()或strftime()函数中使用适当的参数，如例3-17所示。

例3-17：找到周、月及年中的某一天

```
<?php
print strftime("Today is day %d of the month and %j of the year.");
print 'Today is day '.date('d').' of the month and '.date('z').' of the year.';
?>
```

讨论

函数date()和strftime()的运算机制并不相同。对于date()来说，一年的第一天用0表示，而strftime()则用1来表示一年中的第一天。表3-5包含了date()和strftime()函数能够理解的所有日和周的数值格式字符。

表3-5：日和周的数字格式字符

Type	strftime()	date()	Description	Range	Windows
Day	%d	d	一月中的第几天，数字	01 ~ 31	是
Day	%e		一月中的第几天，数字。一位数字前会加上一个空格	1 ~ 31	否
Day	%j	z	一年中的第几天，数字	001~366 for strftime(); 0~365 for date()	是
Day	%u	N	一周中的第几天，数字（1表示周一）	1 ~ 7	否
Day	%w	w	一周中的第几天，（0表示周日）	0 ~ 6	是
Day		j	一月中的第几天，数字。无前导零	1 ~ 31	否

表3-5：日和周的数字格式字符（续）

Type	strftime()	date()	Description	Range	Windows
Day		S	一月中第几天的英文后缀，文本	"st," "th," "nd," "rd"	否
Week	%a	D	简写的周名，当前地区的文本形式		是
Week	%A	l	完整的周名，当前地区的文本形式		是
Week	%U		一年中的第几周，数字。第一周的周日是该周第一天	00 ~ 53	是
Week	%V	W	ISO 8601:1988 标准规定的一年中第几周，数字。周1表示一年中的第一周，该周至少有4天位于当年中。周一是该周的第一天	01 ~ 53	否
Week	%W		一年中的第几周，数字。周一是第一周的第一天	00 ~ 53	是

要想只在周一的时候输出一些信息，可以在date()函数中使用格式化字符w或者在strftime()函数中使用%w字符串，如例3-18所示。

例3-18：检测一周中的某一天

```
<?php
if (1 == date('w')) {
    print "Welcome to the beginning of your work week.";
}

if (1 == strftime('%w')) {
    print "Only 4 more days until the weekend!";
}
```

可以用不同的方式来计算周的数值和一周中某一天的数值，所以在选择适当的方法时必须谨慎一些。ISO标准（ISO8601）中规定，每周以周一作为第一天，而且每周中的7天用数字1（周一）~7（周日）来表示。数字1所表示的那一周是指一年中第一个带有周二的那一周。也就是说，一年中的第一周必须有多数天属于这一年。而表示一年中周的范围是从01 ~ 53。

另一种表示周的标准范围是从00 ~ 53，在这种标准下，一年中的第53周潜在地会与下一年的第00周中的某些天重合。

只要你在自己的程序中保持前后使用的方法和标准一致，就不会有问题，但是当你的程序需要同其他PHP程序或者数据库相互操作时，则必须当心一些。例如，MySQL的DAYOFWEEK()函数将周日作为一周中的第一天，但表示周几的数值也是1～7，而且这也是ODBC的标准。然而，其WEEKDAY()函数，却将周一作为每周的第一天，而周几对应的数字值却是0～6。其WEEK()函数可以让你来选择一周的第一天是周日还是周一，但这个函数却与ISO标准不兼容。

参见

date()函数的文档（*http://www.php.net/date*）和strftime()函数的文档（*http://www.php.net/strftime*）。MySQL的DAYOFWEEK()函数、WEEKDAY()函数以及WEEK()函数的文档（*http://www.mysql.com/doc/D/a/Date_and_time_functions.html*）。

3.8 验证日期

问题

你想要检查一个日期是否有效。例如，你想要确保用户不会提交一个类似“1962年2月30日”这样的生日。

方案

使用checkdate()函数：

```
$valid = checkdate($month,$day,$year);
```

讨论

当$month位于1～12之间，$year位于1～32767之间，$day位于1和$year及$month所代表的相应年、月份中最大的天数之间时，函数checkdate()会返回true。checkdate()函数能够正确地处理闰年，而且日期使用公历（阳历）来表示。

由于checkdate()的有效年份太过宽泛，所以假如你要得到一个有效的出生日期的话，就必须要对年份进行额外的验证。已经证实的人类的最长寿命是122岁。要验证出生日期所表示的年龄处在18～122岁之间，可以使用例3-19中所示的pc_checkbirthdate()函数。

例3-19：pc_checkbirthdate()函数

```
<?php
function pc_checkbirthdate($month,$day,$year) {
    $min_age = 18;
    $max_age = 122;

    if (! checkdate($month,$day,$year)) {
        return false;
    }

    list($this_year,$this_month,$this_day) = explode(',',date('Y,m,d'));

    $min_year = $this_year - $max_age;
    $max_year = $this_year - $min_age;

    print "$min_year,$max_year,$month,$day,$year\n";

    if (($year > $min_year) && ($year < $max_year)) {
        return true;
    } elseif (($year == $max_year) &&
              (($month < $this_month) ||
               (($month == $this_month) && ($day <= $this_day)))) {
        return true;
    } elseif (($year == $min_year) &&
              (($month > $this_month) ||
               (($month == $this_month && ($day > $this_day))))) {
        return true;
    } else {
        return false;
    }
}

// 检查1974年12月3日
if (pc_checkbirthdate(12,3,1974)) {
    print "You may use this web site.";
} else {
    print "You are too young to proceed.";
    exit();
}
?>
```

这个函数首先使用checkdate()来确保$month、$day和$year表示的都是有效日期。通过多次比较来保证所提供的日期处在由$min_age和$max_age所设置的范围之内。

如果$year未包含在$min_year和$max_year之间，表示日期处在合理的范围之内，所以函数返回true。如果$year没有包含在设定的范围之内，则还需进行额外的验证。如果$year等于$max_year（例如，2002年，$year是1984），$month必须是在当前的月份之前。如果$month等于当前月份，$day就必须在当天或者是之前的某一天。如果$year等于$min_year（例如，在2002年，$year是1880），$month必须是在当前月份之后。如果

$month等于当前月份，$day就必须在当天以后。如果不符合以上任何一个条件，则表明所提供的日期位于可以接受的范围之外，函数就会返回false。

如果所提供的日期按当前日期算正好是$min_age年，函数会返回true；但如果提供的日期按当前日期算正好过了$max_age年，函数会返回false。也就是说，如果当前日期正好是用户18岁的生日，则验证通过，但如果是123岁则验证不会通过。

参见

checkdate()函数的文档（*http://www.php.net/checkdate*）。有关Jeanne Calment —— 经证实是最长寿的人的信息（*http://en.wikipedia.org/wiki/Jeanne_Calment*）。

3.9 从字符串中解析日期和时间

问题

你想要从某种格式的字符串中得到一个可以用于计算的日期或者时间值。例如，你想把诸如“last Thursday”之类的日期表达式转换成一个纪元时间戳。

方案

要从某个特定格式的字符串中解析出一个日期或时间的最简单的方式就是使用strtotime()函数，这个函数会把许多符合人类阅读习惯的日期和时间字符串转换成纪元时间戳。其使用方法如例3-20所示。

例3-20：用strtotime()函数来解析字符串

```
<?php
$a = strtotime('march 10');   // 默认为当前年份
$b = strtotime('last thursday');
$c = strtotime('now + 3 months');
```

讨论

strtotime()所使用的语法错综复杂。它使用了GNU数据输入格式规范，可以在下面地址中找到该规范中的有关信息 —— *http://www.gnu.org/software/coreutils/manual/html_chapter/coreutils_27.html*。

strtotime()函数能够理解表示当前时间的词：

```
<?php
$a = strtotime('now');
print strftime('%c',$a);
$a = strtotime('today');
print strftime('%c',$a);
?>

Mon Aug 12 20:35:10 2002
Mon Aug 12 20:35:10 2002
```

能理解以不同方式表示的时间和日期：

```
<?php
$a = strtotime('5/12/1994');
print strftime('%c',$a);
$a = strtotime('12 may 1994');
print strftime('%c',$a);
?>

Thu May 12 00:00:00 1994
Thu May 12 00:00:00 1994
```

能理解相对的时间和日期：

```
<?php
$a = strtotime('last thursday');   // On August 12, 2002
print strftime('%c',$a);
$a = strtotime('2001-07-12 2pm + 1 month');
print strftime('%c',$a);
?>

Thu Aug  8 00:00:00 2002
Mon Aug 12 14:00:00 2002
```

还能理解时区。如果下列代码运行在一台EDT计算机中，会输出相同的时间：

```
<?php
$a = strtotime('2002-07-12 2pm edt + 1 month');
print strftime('%c',$a);
?>

Mon Aug 12 14:00:00 2002
```

然而，如果下面的代码运行在一台EDT计算机中，当时间是下午2点的时候，它会以MDT（比EDT时间早两个小时）的形式输出这一时间：

```
<?php
$a = strtotime('2002-07-12 2pm mdt + 1 month');
print strftime('%c',$a);
?>

Mon Aug 12 16:00:00 2002
```

如果你要从字符串中解析出来的日期和时间是你预先知道的格式，不要使用strtotime()函数，你可以通过正则表达式来取得想要的、不同的日期和时间部件。例3-21显示了如何解析“YYYY-MM-DD HH:MM:SS”日期，这个格式可能来自于MySQL中的DATATIME字段。

例3-21：用正则表达式来解析日期

```
<?php
$date = '1974-12-03 05:12:56';
preg_match('/(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})/',$date,$date_parts);
?>
```

以上代码将年、月、日、时、分、秒放到了$date_parts[1]到$date_parts[6]中（preg_match()函数将把与整个表达式匹配的结果放到$date_parts[0]中）。

你可以用正则表达式从一个包含着其他信息的大型字符串（用户的输入或者一个文件）中提取日期和时间信息。但是，如果你知道日期信息在这个要解析的字符串中的位置，那么使用substr()函数可以加快提取的速度，如例3-22所示。

例3-22：用substr()函数解析日期。

```
<?php
$date_parts[0] = substr($date,0,4);
$date_parts[1] = substr($date,5,2);
$date_parts[2] = substr($date,8,2);
$date_parts[3] = substr($date,11,2);
$date_parts[4] = substr($date,14,2);
$date_parts[5] = substr($date,17,2);
?>
```

也可以使用preg_split()函数，如例3-23所示。

例2-23：用preg_split()函数来解析日期

```
<?php $ar = preg_split('/[- :]/',$date);
var_dump($ar);
?>
```

例3-23的输出结果：

```
array(6) {
  [0]=>
  string(4) "1974"
  [1]=>
  string(2) "12"
  [2]=>
  string(2) "03"
  [3]=>
  string(2) "05"
  [4]=>
```

```
  string(2) "12"
  [5]=>
  string(2) "56"
}
```

当心：PHP在进行数字和字符串间的转换时不会给出任何提示，但对于以0开头的数字，PHP会将其看成是一个八进制（基为8）数。所以03和05是3和5，但08和09却不是8和9。

在PHP 5.1及之后的版本中，在解析类似“YYYY-MM-DD HH:MM:SS”这样的日期格式时，preg_match()函数比strtotime()函数的速度更快。而在之前的版本中，strtotime()的速度稍快一些。如果你所需要的是日期字符串中个别的部分，preg_match()函数更方便，但strtotime()函数显然是更有弹性的选择。

参见

strtotime函数的文档（*http://www.php.net/strtotime*）。strtotime()函数能够解析的字符串的规则（*http://www.gnu.org/software/coreutils/manual/html_chapter/coreutils_27.html*）。

3.10 对日期进行加、减运算

问题

你需要在一个日期上加或减一定的时间间隔。

方案

根据日期和时间间隔的表现形式，可以使用strtotime()或者一些简单的算术方法。

如果日期和时间间隔的格式适当，那么最简单的方法就是使用strtotime()，如例3-24所示。

例3-24：用strtotime()来计算日期的时间间隔

```
<?php
$birthday = 'March 10, 1975';
$whoopee_made = strtotime("$birthday - 9 months ago");
?>
```

如果日期是纪元时间戳的形式，而且时间间隔也可以用秒来表示，那么就简单地从时间戳中减去时间间隔，如例3-25所示。

例3-25：基于纪元时间戳来计算日期的时间间隔

```
<?php
$birthday = 163727100;
$gestation = 36 * 7 * 86400; // 36 周
$whoopee_made = $birthday - $gestation;
?>
```

讨论

strtotime()适合于计算诸如月份等可变长度的时间间隔。如果无法使用strtotime()，可以尝试把日期转换为纪元时间戳，然后再加或减相应的、以秒表示的时间间隔。这种方式对于计算固定长度的时间间隔最有效，比如几天或者几周。例3-26中把相当于7天时间的秒数加到了一个时间戳上。

例3-26：基于纪元时间戳的另一种日期时间间隔

```
<?php
$now = time();
$next_week = $now + 7 * 86400;
?>
```

但是，在使用这种方法时，如果时间间隔的两端分别落在了DST时制切换点的两侧，那么麻烦就来了。这种情况下，本来是长度固定的一天时间不再是86400秒，而变成可能是82800秒或者是90000秒，最终取决于处在哪个季节。

参见

技巧3.5中计算两个日期经过的时间差的内容。技巧3.6中计算两个日期儒略日时间差的内容。strtotime()函数的文档（*http://www.php.net/strtotime*）。

3.11 根据时区计算时间

问题

你需要计算不同时区中的时间。例如，你想让用户看到他们的本地时间，而不是你服务器的时间。

方案

对于简单的计算问题，可以直接加或减两个时区间的偏移量，如例3-27所示。

例3-27：简单的时区计算。

```
<?php
// 如果本地时间是美国东部时间（EST）
$time_parts = localtime();
// 加利福尼亚（太平洋标准时间，PST）要早三个小时
$california_time_parts = localtime(time() - 3 * 3600);
?>
```

在PHP 5.1.0及以后的版本中，可以通过date_default_timezone_set()函数来调整PHP使用的时区。例3-28两次输出本地时间，一次是纽约的本地时间，第二次是巴黎的本地时间。

例3-28：用date_default_timezone_set()函数来改变时区。

```
<?php
$now = time();
date_default_timezone_set('America/New York');
print date('c', $now);
date_default_timezone_set('Europe/Paris');
print date('c', $now);
?>
```

在基于Unix系统中使用PHP早期版本的情况下，如果你不知道时区间的偏移量，只要把TZ环境变量设置为你的目标时区即可，如例3-29所示。

例3-29：用环境变量改变时区

```
<?php
putenv('TZ=PST8PDT');
$california_time_parts = localtime();
?>
```

讨论

在深入讨论有关时区的各种细节问题之前，我们打算先介绍一些被否决的美国海军天文台的情况（相关信息的网络地址为*http://tycho.usno.navy.mil/tzones.html*）。这种官方的全球时区信息系统有点脆弱——“因为国家拥有独立的主权，可以自主选择认为适合自己国家的计时系统”。所以，要记住我们时刻处在变幻莫测的国际关系的支配之下，下面介绍应对地球上众多时区的一些方法。

PHP 5.1.0对时间和日期功能进行了彻底的重建，其中最突出的部分是极大地改进了对时区的处理。新增的date_default_timezone_get()函数和date_default_timezone_set()函数使得根据时区得到适当的格式化输出结果成为一件轻而易举的小事。而且，还增加了一个新的配置命令——date.timezone，在没有调用date_default_timezone_set()函数时可以用它来设置要使用的默认时区。

有了这些功能后，在用date()或strftime()生成一个格式化的时间或日期之前，你所要做的就是确保把你想使用的时区设置当前的默认时区（使用date.timezone或者date_default_timezone_set()）。如果你的应用程序要面向多个时区的用户，有一种方法很方便，即把默认的时区设置为GMT，然后再在创建日期或时间字符串之前明确地设置适当的时区（也可能是基于用户的偏好）。这样就使得生成针对特定时区的时间值的代码变得很清晰。

PHP能够识别的时区在PHP手册（*http://www.php.net/timezones*）的附录H中。其中诸如“America/New_York，Europe/Paris，Africa/Dar_es_Salaam”之类的时区名称映像自流行的“zoneinfo”数据库。

如果你使用的是PHP的一个早期版本，你必须要自己来完成时区的匹配。对于相对简单的时区偏移量的问题，可以在程序中使用一个数组，用它来保存各个时区基于UTC的偏移量。当你确定了用户在某个时区内时，只需将对应的偏移量加上适当的UTC时间，之后用输出UTC时间的函数（例如：gmdate()，gmstrftime()）就能够输出经过调整后的正确的时间。例3-30把时间从UTC调整为PST。

例3-30：把时间从UTC调整到另一个时区

```
<?php
// 取得当前时间
$now = time();

// 加利福尼亚晚于UTC时间8小时
$now += $pc_timezones['PST'];

// 使用gmdate()或gmstrftime()来输出加利福尼亚对应的时间
print gmstrftime('%c',$now);
?>
```

例3-30中使用的数组$pc_timezones在例3-31中定义，这个数组中包含了各个时区基于UTC的偏移量。

例3-31：各个时区基于UTC的偏移量

```
// 引自Perl中的Time::Timezone
$pc_timezones = array(
  'GMT'  =>   0,              // Greenwich Mean
  'UTC'  =>   0,              // Universal (Coordinated)
  'WET'  =>   0,              // Western European
  'WAT'  =>  -1*3600,         // West Africa
  'AT'   =>  -2*3600,         // Azores
  'NFT'  =>  -3*3600-1800,    // Newfoundland
  'AST'  =>  -4*3600,         // Atlantic Standard
  'EST'  =>  -5*3600,         // Eastern Standard
  'CST'  =>  -6*3600,         // Central Standard
  'MST'  =>  -7*3600,         // Mountain Standard
```

```
    'PST'  =>  -8*3600,          // Pacific Standard
    'YST'  =>  -9*3600,          // Yukon Standard
    'HST'  => -10*3600,          // Hawaii Standard
    'CAT'  => -10*3600,          // Central Alaska
    'AHST' => -10*3600,          // Alaska-Hawaii Standard
    'NT'   => -11*3600,          // Nome
    'IDLW' => -12*3600,          // International Date Line West
    'CET'  =>  +1*3600,          // Central European
    'MET'  =>  +1*3600,          // Middle European
    'MEWT' =>  +1*3600,          // Middle European Winter
    'SWT'  =>  +1*3600,          // Swedish Winter
    'FWT'  =>  +1*3600,          // French Winter
    'EET'  =>  +2*3600,          // Eastern Europe, USSR Zone 1
    'BT'   =>  +3*3600,          // Baghdad, USSR Zone 2
    'IT'   =>  +3*3600+1800,     // Iran
    'ZP4'  =>  +4*3600,          // USSR Zone 3
    'ZP5'  =>  +5*3600,          // USSR Zone 4
    'IST'  =>  +5*3600+1800,     // Indian Standard
    'ZP6'  =>  +6*3600,          // USSR Zone 5
    'SST'  =>  +7*3600,          // South Sumatra, USSR Zone 6
    'WAST' =>  +7*3600,          // West Australian Standard
    'JT'   =>  +7*3600+1800,     // Java
    'CCT'  =>  +8*3600,          // China Coast, USSR Zone 7
    'JST'  =>  +9*3600,          // Japan Standard, USSR Zone 8
    'CAST' =>  +9*3600+1800,     // Central Australian Standard
    'EAST' => +10*3600,          // Eastern Australian Standard
    'GST'  => +10*3600,          // Guam Standard, USSR Zone 9
    'NZT'  => +12*3600,          // New Zealand
    'NZST' => +12*3600,          // New Zealand Standard
    'IDLE' => +12*3600           // International Date Line East
);
```

在Unix系统中，可以使用*zoneinfo*库来完成这种转换。这样会使代码更简洁，并且也会更加明晰地处理DST问题，具体内容请参见技巧3.12。

要在PHP中利用*zoneinfo*库，就要使所有内部日期都与纪元时间戳相匹配。可以用例3-32所示的pc_mktime()函数来基于时间部件生成这些匹配的日期。

例3-32：pc_mktime()函数

```
<?php
function pc_mktime($tz,$hr,$min,$sec,$mon,$day,$yr) {
    putenv("TZ=$tz");
    $a = mktime($hr,$min,$sec,$mon,$day,$yr);
    putenv('TZ=EST5EDT');   // 把 EST5EDT 改为你的服务器的时区!
    return $a;
}
?>
```

在调用mktime()函数之前调用putenv()函数，是为了欺骗系统函数mktime()，使其认为它们位于不同的时区。在调用完mktime()后，必须再恢复正确的时区。在美国东海岸，是“EST5EDT”。你在使用这个函数时要把这个值修改为你的计算机所在的时区（参

见表3-6）。但是，在多线程的环境中利用环境变量有可能会导致问题。如果你在一个多线程的Web服务器中运行PHP，最好是把PHP更新到5.1.0版以上，以便能够直接使用date_default_timezone_set()函数。

pc_mktime()函数用于把时间部件转换成纪元时间戳。与其相对应的函数是pc_strftime()，它把纪元时间戳转换成格式化的时间字符串以及时间部件。pc_strftime()函数在例3-33中定义。

例3-33：pc_strftime()函数

```
<?php
function pc_strftime($tz,$format,$timestamp) {
    putenv("TZ=$tz");
    $a = strftime($format,$timestamp);
    putenv('TZ=EST5EDT');      // 把 EST5EDT 改为你的服务器的时区!
    return $a;
}
?>
```

例3-33使用了与pc_mktime()函数同样的系统欺骗函数，以便从strftime()中得到正确的结果。

这两个函数的高明之处在于，不管是否处于夏令时，或者有其他任何基于时区差别的不规则的情况，你都不用再担心不同时区基于UTC的偏移量，而只需设置适当的时区，其余的事就都由你的系统库去完成了。

要注意的是，这两个函数中的变量$tz并非时区名称，而是*zoneinfo*库中的时区。*zoneinfo*库中的时区比时区名称更详尽，因为它考虑到了特殊的地区。表3-6中包含了针对UTC偏移量的*zoneinfo*库中的时区列表。表中最后一栏表示相应时区是否实行夏令时。

表3-6：zoneinfo库中的时区

UTC 偏移（小时）	UTC 偏移（秒）	时区信息 时区 zone	DST?
-12	-43,200	Etc/GMT+12	否
-11	-39,600	Pacific/Midway	否
-10	-36,000	US/Aleutian	是
-10	-36,000	Pacific/Honolulu	否
-9	-32,400	America/Anchorage	是
-9	-32,400	Etc/GMT+9	否
-8	-28,800	PST8PDT	是

表3-6：zoneinfo库中的时区（续）

UTC 偏移（小时）	UTC 偏移（秒）	时区信息 时区 zone	DST?
-8	-28,800	America/Dawson_Creek	否
-7	-25,200	MST7MDT	是
-7	-25,200	MST	否
-6	-21,600	CST6CDT	是
-6	-21,600	Canada/Saskatchewan	否
-5	-18,000	EST5EDT	是
-5	-18,000	EST	否
-4	-14,400	America/Halifax	是
-4	-14,400	America/Puerto_Rico	否
-3.5	-12,600	America/St_Johns	是
-3	-10,800	America/Buenos_Aires	否
0	0	Europe/London	是
0	0	GMT	否
1	3,600	CET	是
1	3,600	GMT-1	否
2	7,200	EET	否
2	7,200	GMT-2	否
3	10,800	Asia/Baghdad	是
3	10,800	GMT-3	否
3.5	12,600	Asia/Tehran	是
4	14,400	Asia/Dubai	否
4	14,400	Asia/Baku	是
4.5	16,200	Asia/Kabul	否
5	18,000	Asia/Tashkent	否
5.5	19,800	Asia/Calcutta	否
5.75	20,700	Asia/Katmandu	否
6	21,600	Asia/Novosibirsk	是
6	21,600	Etc/GMT-6	否
6.5	23,400	Asia/Rangoon	否
7	25,200	Asia/Jakarta	否
8	28,800	Hongkong	否

表3-6：zoneinfo库中的时区（续）

UTC 偏移（小时）	UTC 偏移（秒）	时区信息 时区 zone	DST?
9	32,400	Japan	否
9.5	34,200	Australia/Darwin	否
10	36,000	Australia/Sydney	是
10	36,000	Pacific/Guam	否
12	43,200	Etc/GMT-13	否
12	43,200	Pacific/Auckland	是

世界各国不会在同一天或者同一时间开始（结束）夏令时。如果想要计算一个合乎国际性DST惯例的地区时间，那么就需要尽可能选择与你期望的地区更匹配的*zoneinfo*库中的时区。

参见

技巧3.12中处理DST的内容。date_default_timezone_set()函数的文档（*http://www.php.net/date_default_timezone_set*），date_default_timezone_get()函数的文档（*http://www.php.net/date_default_timezone_get*），putenv()函数的文档（*http://www.php.net/putenv*），localtime()函数的文档（*http://www.php.net/localtime*），gmdate()函数的文档（*http://www.php.net/gmdate*）以及gmstrftime()函数的文档（*http://www.php.net/gmstrftime*）。PHP能够识别的时区列表信息（*http://www.php.net/timezones*）。*zoneinfo*库中的时区名称以及全世界数百个地区的经纬度坐标信息（*ftp://elsie.nci.nih.gov/pub/*）。要查找名称以*tzdata*开头最新文件、众多有关时区历史和技术信息的链接资源，以及有关*zoneinfo*数据库的信息，请访问下面的地址：*http://www.twinsun.com/tz/tz-link.htm*。

3.12 处理夏令时

问题

你需要确保在计算时间的过程中适当地考虑了夏令时（DST）。

方案

如果你使用的是PHP 5.1.0及以后的版本，可以通过date_default_timezone_set()来设置

适当的时区。这些时区都能识别夏令时。例3-34使用了date_default_timezone_set()来输出适当的DST格式的时间字符串。

例3-34：用date_default_timezone_set()来处理DST

```
<?php
// 科罗拉多州，丹佛遵守DST
date_default_timezone_set('America/Denver');
// 2008年7月4日是夏天
$summer = mktime(12,0,0,7,4,2008);
print date('c', $summer) . "\n";
// 亚利桑那州，凤凰城不实行DST
date_default_timezone_set('America/Phoenix');
print date('c', $summer) . "\n";
?>
```

例3-34的输出结果：

```
2008-07-04T12:00:00-06:00
2008-07-04T11:00:00-07:00
```

对于PHP早期的版本，就必须用另一种方法了。*zoneinfo*库能够满足计算DST的需要。如果你使用的是基于Unix的系统，可以通过putenv()来利用*zoneinfo*库，如例3-35所示。

例3-35：用zoneinfo库来处理DST

```
<?php
// 科罗拉多州，丹佛遵守DST
putenv('TZ=America/Denver');
// 2008年7月4日是夏天
$summer = mktime(12,0,0,7,4,2008);
print date('c', $summer) . "\n";
// 亚利桑那州，凤凰城不实行DST
putenv('TZ=America/Phoenix');
print date('c', $summer) . "\n";
?>
```

如果你不能使用*zoneinfo*库，那么可以基于本地时区是否遵守DST来修改硬编码的时区偏移量。使用localtime()函数可以确定当前时区遵守DST的状态，如例3-36所示。

例3-36：通过明确的偏移量来处理DST

```
<?php
// 取得当前的UTC时间
$now = time();

// 加利福尼亚晚于UTC时间8个小时
$now -= 8 * 3600;

// 遵守DST吗?
$ar = localtime($now,true);
if ($ar['tm_isdst']) { $now += 3600; }
```

```
// 使用gmdate()或gmstrftime()输出加利福尼亚的适当时间
print gmstrftime('%c',$now);
?>
```

讨论

用基于UTC时间的时区偏移量来代替纪元时间戳，之后用gmdate()或gmstrftime()函数来输出针对时区的函数具有一定的灵活性——它能计算任何时区的时间——但DST的计算则会有一点不正确。对于简短的时间间隔，当服务器的DST状态不同于目标时区时，所得的结果就不正确。例如，美国东部（EDT）时间四月份第一个周日的凌晨3:30（在DST生效时间之后），对于美国西部（PDT）来说仍然还处在DST生效时间（晚上11:30）以前。以东部时间为准的服务器使用此方法计算得到的加利福尼亚州的时间，会晚于UTC时间7个小时，但实际上却是8个小时。EDT时间早晨6:00（PDT凌晨3:00），美国东西部都进入夏令时，而此时的计算又正确了（加利福尼亚州的时间比UTC时间晚7个小时）。

参见

技巧3.11中处理与时区问题的有关内容。date_default_timezone_set()函数的文档（*http://www.php.net/date_default_timezone_set*），putenv()函数的文档（*http://www.php.net/putenv*），localtime()函数的文档（*http://www.php.net/localtime*），gmdate()函数的文档（*http://www.php.net/gmdate*）以及gmstrftime()函数的文档（*http://www.php.net/gmstrftime*）。关于夏令时（DST）的详细介绍（*http://webexhibits.org/daylightsaving/*）。

3.13 生成高精度的时间

问题

你需要使用高于秒的计量精度的时间——例如，要生成一个唯一的ID或者一个函数调用的基准。

方案

使用microtime(true)来以秒和微秒计量当前的时间。例3-37使用了microtime(true)来计算要完成1000个正则表达式匹配需要多长时间。

例3-37：用microtime()来计时。

```
<?php
$start = microtime(true);
for ($i = 0; $i < 1000; $i++) {
    preg_match('/age=\d+/',$_SERVER['QUERY_STRING']);
}
$end = microtime(true);
$elapsed = $end - $start;
?>
```

讨论

PHP 5.0.0中增加了对在microtime()中使用可选参数的支持。如果不带参数，或者带有不能转换成true的参数，或者在PHP的早期版本中，microtime()会返回一个自Unix纪元起所经过时间的微秒部分，一个空格和自纪元起所经过的秒数。例如，返回值“0.41644100 1026683258”意味着自纪元起经过了“1026683258.41644100秒”时间。

包含微秒的时间值对于生成唯一的ID非常有用。只要一个进程不会在一微秒内同时生成一个以上的ID，就能保证用包含微秒的时间值与当前进程的ID组合成一个唯一的ID值。例3-38使用microtime()（返回的字符串格式）生成了这样一个ID。

例3-38：用microtime()生成一个ID

```
<?php
[list($microseconds,$seconds) = explode(' ',microtime());
$id = $seconds.$microseconds.getmypid();
?>
```

注意一下，例3-38在多线程的系统中并不十分可靠，因为存在一个非零（但极小）的概率，同一个进程的两个线程会在同一微秒内调用microtime()。

参见

microtime()函数的文档（*http://www.php.net/microtime*）。用于生成有效唯一ID的uniqid()函数。

3.14 生成时间范围

问题

你需要获得一周或一月内的所有天。例如，你想要打印一周内所有约会的列表。

方案

用time()和strftime()来标识你的开始日期。如果你的时间间隔长度固定，可以通过循环来操作其中每一天。如果时间间隔长度不固定，则需要测试后来的每一天是否属于你所期望的时间范围。

例如，一周有7天，所以你就可以用一个固定的循环来枚举出这一周之内的每一天，如例3-39所示。

例3-39：枚举一周内的7天

```
<?php
// 为本周产生一个时间范围
$now = time();

// 如果在凌晨3点之前，$now递增，所以当返回本周的开始时，我们不会碰到DST(?)
if (3 < strftime('%H', $now)) { $now += 7200; }

// 今天是本周的哪一天?
$today = strftime('%w', $now);

// 几天之前是本周的第一天?
$start_day = $now - (86400 * $today);

// 输出本周的每一天
for ($i = 0; $i < 7; $i++) {
  print strftime('%c',$start_day + 86400 * $i);
}
?>
```

讨论

一个特殊的月份或者年份中可能会包含不同的天数，所以你必须基于特殊的月份或者年份来计算时间范围的终点。要循环一月中的每一天，计算出该月第一天的时间戳，以及下个月第一天的时间戳。在例3-40中，循环变量$day每次递增一天（86400秒），直到它到了下个月的开始而不再小于纪元时间戳为止。

例3-40：枚举一个月中的每一天。

```
<?php
// 为本月生成一个时间范围
$now = time();

//如果在凌晨3点之前，$now递增，所以当返回月初时，我们不会碰到DST(?)
// when moving back to the beginning of the month
if (3 < strftime('%H', $now)) { $now += 7200; }

// 本月是几月?
$this_month = strftime('%m',$now);
```

```
// 本月第一天午夜的纪元时间戳
$day = mktime(0,0,0,$this_month,1);
// 下月第一天午夜的纪元时间戳
$month_end = mktime(0,0,0,$this_month+1,1);

while ($day < $month_end) {
  print strftime('%c',$day);
  $day += 86400;
}
?>
```

参见

time()函数的文档（*http://www.php.net/time*）和strftime()函数的文档（*http://www.php.net/strftime*）。

3.15 使用非公历纪年

问题

你想使用一种非公历纪年法，例如，儒略（Julian）历、犹太（Jewish）历或者法国共和（French Republican）历。

方案

PHP的日历扩展为使用儒略历、法国共和历和犹太历提供了转换函数。如果想使用这些转换函数，必须加载相应的日历扩展。

这些函数使用儒略日（不同于儒略历）作为它们之间交换信息的中间格式。cal_to_jd()会将月、日和年转换为儒略日的计数值；而cal_from_jd()则能够将儒略日计数值转换成特定历法下的月、日和年。例3-41实现了在儒略日和我们熟悉的公历纪年之间的转换。

例3-41：实现儒略日与公历之间的转换。

```
<?php
// 1876年3月9日
$jd = gregoriantojd(3,9,1876);
// $jd = 2406323

$gregorian = cal_from_jd($jd, CAL_GREGORIAN);
/* $gregorian 是一个数组:
array(9) {
  ["date"]=>
  string(8) "3/9/1876"
  ["month"]=>
```

```
  int(3)
  ["day"]=>
  int(9)
  ["year"]=>
  int(1876)
  ["dow"]=>
  int(4)
  ["abbrevdayname"]=>
  string(3) "Thu"
  ["dayname"]=>
  string(8) "Thursday"
  ["abbrevmonth"]=>
  string(3) "Mar"
  ["monthname"]=>
  string(5) "March"
}
*/
?>
```

公历的有效范围是公元前4714年～公元9999年。

讨论

要实现儒略日与儒略历之间的转换，需要像例3-42所示的那样，使用CAL_JULIAN常量。

例3-42：使用儒略历

```
<?php
// 1900年2月29日（不是一个公历闰年）
$jd = cal_to_jd(CAL_JULIAN, 2, 29, 1900);
// $jd = 2415092

$julian = cal_from_jd($jd, CAL_JULIAN);
/* $julian 是一个数组：
array(9) {
  ["date"]=>
  string(9) "2/29/1900"
  ["month"]=>
  int(2)
  ["day"]=>
  int(29)
  ["year"]=>
  int(1900)
  ["dow"]=>
  int(2)
  ["abbrevdayname"]=>
  string(3) "Tue"
  ["dayname"]=>
  string(7) "Tuesday"
  ["abbrevmonth"]=>
  string(3) "Feb"
  ["monthname"]=>
  string(8) "February"
```

```
}
*/

$gregorian = cal_from_jd($jd, CAL_GREGORIAN);
/* $gregorian 是一个数组
array(9) {
  ["date"]=>
  string(9) "3/13/1900"
  ["month"]=>
  int(3)
  ["day"]=>
  int(13)
  ["year"]=>
  int(1900)
  ["dow"]=>
  int(2)
  ["abbrevdayname"]=>
  string(3) "Tue"
  ["dayname"]=>
  string(7) "Tuesday"
  ["abbrevmonth"]=>
  string(3) "Mar"
  ["monthname"]=>
  string(5) "March"
}
*/
?>
```

儒略历的有效范围是公元前4713年～公元9999年，但由于该历法创建于公元前46年，所以如果你想用它来表示早于公元前46年的日期，可能会有被视为儒略日纯粹论者的风险。

要实现儒略日与法国共和历之间的转换，需要使用CAL_FRENCH常量，如例3-43所示。

例3-43：使用法国共和历

```
<?php
// 法国共和历11年热（8）月13日
$jd = cal_to_jd(CAL_FRENCH, 8, 13, 11);
// $jd = 2379714

$french = cal_from_jd($jd, CAL_FRENCH);
/* $french 是一个数组：
array(9) {
  ["date"]=>
  string(7) "8/13/11"
  ["month"]=>
  int(8)
  ["day"]=>
  int(13)
  ["year"]=>
  int(11)
  ["dow"]=>
```

```
  int(2)
  ["abbrevdayname"]=>
  string(3) "Tue"
  ["dayname"]=>
  string(7) "Tuesday"
  ["abbrevmonth"]=>
  string(7) "Floreal"
  ["monthname"]=>
  string(7) "Floreal"
}
*/

// 1803年5月3日——把路易斯安那州卖给美国
$gregorian = cal_from_jd($jd, CAL_GREGORIAN);
/* $gregorian 是一个数组：
array(9) {
  ["date"]=>
  string(8) "5/3/1803"
  ["month"]=>
  int(5)
  ["day"]=>
  int(3)
  ["year"]=>
  int(1803)
  ["dow"]=>
  int(2)
  ["abbrevdayname"]=>
  string(3) "Tue"
  ["dayname"]=>
  string(7) "Tuesday"
  ["abbrevmonth"]=>
  string(3) "May"
  ["monthname"]=>
  string(3) "May"
}
*/
?>
```

法国共和历的有效范围是公历1972年9月 ~ 1806年9月，虽然这一时间区间很短，但因为这一历法只用于自1973年10月 ~ 1806年1月之间，所以也足够用了。注意，cal_from_jd()函数返回的月份名称不包含撇号 —— 即，返回的是“Floreal”而非“Floréal”。

要实现儒略日与犹太历之间的转换，需要使用CAL_JEWISH常量，如例3-44所示。

例3-44：使用犹太历。

```
<?php
// 犹太历5761年亚达（6）月14日
$jd = cal_to_jd(CAL_JEWISH, 6, 14, 5761);
// $jd = 2451978

$jewish = cal_from_jd($jd, CAL_JEWISH);
/* $jewish is an array:
```

```
array(9) {
  ["date"]=>
  string(9) "6/14/5761"
  ["month"]=>
  int(6)
  ["day"]=>
  int(14)
  ["year"]=>
  int(5761)
  ["dow"]=>
  int(5)
  ["abbrevdayname"]=>
  string(3) "Fri"
  ["dayname"]=>
  string(6) "Friday"
  ["abbrevmonth"]=>
  string(5) "AdarI"
  ["monthname"]=>
  string(5) "AdarI"
}
*/

$gregorian = cal_from_jd($jd, CAL_GREGORIAN);
/* $gregorian is an array:
array(9) {
  ["date"]=>
  string(8) "3/9/2001"
  ["month"]=>
  int(3)
  ["day"]=>
  int(9)
  ["year"]=>
  int(2001)
  ["dow"]=>
  int(5)
  ["abbrevdayname"]=>
  string(3) "Fri"
  ["dayname"]=>
  string(6) "Friday"
  ["abbrevmonth"]=>
  string(3) "Mar"
  ["monthname"]=>
  string(5) "March"
}
*/
?>
```

犹太历的有效范围始自公元前3761年（犹太历1年）。注意，无论是否落在闰年中，亚达月总是返回为“AdarI”。在闰年中，AdarII返回为“AdarII”。

参见

历法函数的文档（*http://www.php.net/calendar*）。有关公历历史的资料（*http://scienceworld.wolfram.com/astronomy/GregorianCalendar.html*）。

3.16 使用纪元时间戳范围之外的日期

问题

你想要使用位于32位的纪元时间戳能够处理范围之外的日期 —— 概言之，即1901之前或者2038之后的日期。

方案

使用PEAR的Date_calc类，该类可以处理自公元1年1月1日～公元9999年12月31日之间的日期。例3-45输出了公元18世纪中两天的格式化日期。

例3-45：使用Date_calc。

```
<?php
require_once 'Date/Calc.php';

// 1790年4月17日
$date = Date_Calc::dateFormat( 17, 4, 1790, '%A %B %e, %Y');

print "Benjamin Franklin died on $date.";
?>
```

例3-45的输出结果：

```
Benjamin Franklin died on Saturday April 17, 1790.
```

讨论

因为Date_calc使用其自身内部的日期表示法，所以它不受存储32位整型纪元时间戳的限制。其dateFormat()方法同strftime()函数类似 —— 同样也是将一个格式字符串返回为一个格式化的日期和时间字符串。表3-7中列出了dateFormat()函数能够接受的格式化字符。

表3-7：Date_Calc::dateFormat()函数可用的格式化字符

字符	说明
%d	一月中的第几天，带前导0
%e	一月中的第几天，无前导0
%w	一周中的第几天，无前导0，周日是0
%j	一年中的第几天，带前导0
%E	根据内部的 Date_Calc 纪元时间戳计算机的天数
%a	简写的周名
%A	完整的周名
%U	当年的周数
%m	月份数，无前导0，一月是1
%b	简写的月份名
%B	完整的月份名
%y	年份，带前导0的二位数字
%Y	年份，带前导0的4位数字
%n	换行符
%t	制表符
%%	百分号（%）

Date_Calc虽然能使处理大跨度的公历日期变得更容易，但它却没有考虑随着时间的推移，由于宗教、政治和文化因素所导致的历法变更。

参见

PEAR 的 Date包（*http://pear.php.net/package/Date*）。有关历法随时间怪人改变的研究资料（*http://en.wikipedia.org/wiki/Old_Style_and_New_Style_dates*）。

3.17 编程：日历

例3-47中的pc_calendar()函数会输出一个月历，这个函数与Unix的*cal*程序类似。例3-46是关于如何使用这个函数的，其中也包含月历布局的默认样式。

例3-46：使用pc_calendar()函数。

```
<style type="text/css">
.prev { text-align: left; }
.next { text-align: right; }
```

```
.day, .month, .weekday { text-align: center; }
.today { background: yellow; }
.blank { }
</style>
<?php
// 如果年或月不在查询字符串中，则输出当前月份的日历
$month = isset($_GET['month']) ? intval($_GET['month']) : date('m');
$year = isset($_GET['year']) ? intval($_GET['year']) : date('y');
?>
```

pc_calendar()函数会输出一个包含月历的表格。其中还提供了显示上一个月和下一个月的链接，同时突出显示了当前日期，如例3-47所示。

例3-47：pc_calendar()函数。

```
<?php
function pc_calendar($month,$year,$opts = '') {
    // set default options
    if (! is_array($opts)) { $opts = array(); }
    if (! isset($opts['id'])) { $opts['id'] = 'calendar'; }
    if (! isset($opts['month_link'])) {
        $opts['month_link'] =
            '<a href="'.$_SERVER['PHP_SELF'].'?month=%d&amp;year=%d">%s</a>';
    }
    $classes = array();
    foreach (array('prev','month','next','weekday','blank','day','today') as $class) {
        if (isset($opts[$class.'_class'])) {
            $classes[$class] = htmlentities($opts[$class.'_class']);
        } else {
            $classes[$class] = $class;
        }
    }

    list($this_month,$this_year,$this_day) = split(',',strftime('%m,%Y,%d'));
    $day_highlight = (($this_month == $month) && ($this_year == $year));

    list($prev_month,$prev_year) =
        split(',',strftime('%m,%Y',mktime(0,0,0,$month-1,1,$year)));
    $prev_month_link = sprintf($opts['month_link'],$prev_month,$prev_year,'&laquo;');

    list($next_month,$next_year) =
        split(',',strftime('%m,%Y',mktime(0,0,0,$month+1,1,$year)));
    $next_month_link = sprintf($opts['month_link'],$next_month,$next_year,'&raquo;');

?>
<table id="<?php echo htmlentities($opts['id']) ?>">
        <tr>
                <td class="<?php echo $classes['prev'] ?>">
                        <?php print $prev_month_link ?>
                </td>
                <td class="<?php echo $classes['month'] ?>" colspan="5">
                <?php print strftime('%B %Y',mktime(0,0,0,$month,1,$year)); ?>
                </td>
                <td class="<?php echo $classes['next'] ?>">
```

```
                    <?php print $next_month_link ?>
                </td>
            </tr>
<?php
    $totaldays = date('t',mktime(0,0,0,$month,1,$year));

    // print out days of the week
    print '<tr>';
    $weekdays = array('Su','Mo','Tu','We','Th','Fr','Sa');
    while (list($k,$v) = each($weekdays)) {
        print '<td class="'.$classes['weekday'].'">'.$v.'</td>';
    }
    print '</tr><tr>';
    // align the first day of the month with the right week day
    $day_offset = date("w",mktime(0, 0, 0, $month, 1, $year));
    if ($day_offset > 0) {
        for ($i = 0; $i < $day_offset; $i++) {
            print '<td class="'.$classes['blank'].'"> </td>';
        }
    }
    $yesterday = time() - 86400;

    // print out the days
    for ($day = 1; $day <= $totaldays; $day++) {
        $day_secs = mktime(0,0,0,$month,$day,$year);
        if ($day_secs >= $yesterday) {
            if ($day_highlight && ($day == $this_day)) {
                print '<td class="' . $classes['today'] .'">' . $day . '</td>';
            } else {
                print '<td class="' . $classes['day'] .'">' . $day . '</td>';
            }
        } else {
            print '<td class="' . $classes['day'] .'">' . $day .'</td>';
        }
        $day_offset++;

        // start a new row each week //
        if ($day_offset == 7) {
            $day_offset = 0;
            if ($day < $totaldays) { print "</tr>\n<tr>"; }
        }
    }
    // fill in the last week with blanks //
    if ($day_offset > 0) { $day_offset = 7 - $day_offset; }
    if ($day_offset > 0) {
        for ($i = 0; $i < $day_offset; $i++) {
            print '<td class="'.$classes['blank'].'"> </td>';
        }
    }
    print '</tr></table>';
}
?>
```

pc_calendar()函数首先会检查传递给它的参数$opts。你可以在$opts['month_link']中传递一个printf()式的格式字符串，用于修改表示上个月和下个月链接的输出样式，也可以为表格设置一个id属性。如果没有指定，其id值会使用默认的“calendar”。

此外，还能为布局中的多个元素传递不同的类名。这些类的可选名称包括prev_class、month_class、 next_class、weekday_class、blank_class、day_class和today_class。默认的值是prev、month、next、weekday、blank、day和today。例3-46中包含了符合人们视觉习惯的基本布局样式，并用黄色背景突出显示了当前日期。

接着，如果日历的月份和年份与当前的月份和年份匹配，函数会将$day_highlight设置为true。指向上一个月和下一个月的链接会使用%opts['month_link']中的格式字符串并保存在$prev_month_link和$next_month_link变量中。

之后，这个函数开始输出包含月历的HTML表格的顶部以及一个包含星期缩写的表格行。根据strftime('%w')返回的表示周几的值，输出空白的表格单元，使每个月的第一天与相应的周几对齐。例如，如果该月的第一天是星期二，那么就会在表格第一行的星期天和星期一下方输出两个空白表格单元以补空。

在输出以上预备信息之后，pc_calendar()函数会循环枚举该月中的每一天。它会按照最多的天数输出一个普通的表格，只是对于当前日期所在的单元会使用一个不同地背景颜色。当$day_offset等于7时，即表示已经输出完一周，接着就会输出另一行表格单元。

当该月中的每一天都输出完毕后，最后一行剩余的表格单元会以空白填充。例如，如果该月的最后一天是星期四，那么在星期五和星期六对应的表格单元处就会输出两个空白的表格单元进行补空。最后，输出表格的闭标记，完整的月历输出完毕。

第4章

数组

4.0 概述

数组就是列表 —— 人的列表、尺寸的列表、书的列表。要在一个变量中保存一组相关的项目，就要用到数组。就和写在一张纸上的列表一样，数组中的元素也有先后顺序。通常，每个新项目都添加到数组中最后一个项目的后面，但正如你能够在纸上列表的两行间插入一个新条目一样，在PHP中也能在数组元素间插入新的项目。

许多编程语言都只有一种类型的数组 —— 数字数组（或者数组）。在数字数组中，如果你想要找一个项目，首先需要知道该项目在数组中的位置 —— 也就是该项目的索引值。项目在数组中的位置是以数字来表示的，而且从0开始，向上逐项加1。

在某些编程语言中，还有另外一种类型的数组 —— 关联数组，也称为散列（哈希）表。在关联数组中，索引值不以整数表示，而是用字符串来表示。所以，在一个保存着美国总统名单的数字数组中，“Abraham Lincoln”的索引值可能是16，而在该数组的关联数组版中，该项目的索引可能会是“Honest”。虽然在数字数组中囿于索引的限制，项目之间都有严格的先后顺序，但在关联数组中，项目间通常没有先后顺序可言。虽然向数组中添加新元素是依照某种顺序完成的，但之后却没办法确定这些元素的顺序。

在少数编程语言中，会同时拥有数字数组和关联数组。但是，一般来说数字数组$presidents和关联数组$presidents应该是截然不同的。每种类型的数组都有其特殊的行为，因此需要分别采取相应的操作方式。PHP就同时拥有数字数组和关联数组，但PHP中的这两种类型的数组却不是相互独立的。

在PHP中，数字数组也是关联数组，而关联数组也是数字数组。那么，到底是什么类型的数组呢？两者都是也都不是。这两种类型间的界限经常会变得很模糊。起先，这可能会引起你的迷惑不解，特别是当你使用一种严格的行为时尤其如此。但用不了多久，你会发现这种灵活性的妙处。

要一次将多个值赋予数组，可以使用array()：

```
$fruits = array('Apples', 'Bananas', 'Cantaloupes', 'Dates');
```

现在，$fruits[2]的值就是'Cantaloupes'。

在根据已知值的一个简短列表创建数组时，array()是很方便的。同样的数组也可以用以下方式来生成：

```
$fruits[0] = 'Apples';
$fruits[1] = 'Bananas';
$fruits[2] = 'Cantaloupes';
$fruits[3] = 'Dates';
```

或者

```
$fruits[] = 'Apples';
$fruits[] = 'Bananas';
$fruits[] = 'Cantaloupes';
$fruits[] = 'Dates';
```

将一个值赋给用空下标（索引）表示的数组，是向数组结尾添加新元素的简写方式。PHP会在添加新元素之前，先确定$fruits的长度值，然后将这个值作为新元素的索引。当然，这是假设没有把$fruits变量设置为一个标量值，比如3或者一个对象。PHP会抗议把一个非数组当成一个数组来操作。但是，如果是第一次使用一个变量，PHP则会自动地把它转换成一个数组，并将起始索引设为0。

具有同样功能的是函数array_push()，它会把一个新值推送到数组栈的顶部。而$foo[]标识符是PHP中更为传统的风格，而且速度也更快。然而，有的时候用array_push()会更精确地传达你想要的栈的特性，特别是在与array_pop()函数组合使用时。array_pop()用于删除数组中的最后一个元素并返回这个元素。

到现在为止，我们只涉及到了数组中的整数和字符串。实际上，PHP允许给数组元素指定任何数据类型的值，像布尔值、整数、浮点数、字符串、对象、NULL、甚至其他的数组。因此，你可以直接把从数据库中取出的数组或者对象保存的一个数组中：

```
while ($row = mysql_fetch_row($r)) {
    $fruits[] = $row;
}

while ($obj = mysql_fetch_object($s)) {
    $vegetables[] = $obj;
}
```

第一个while语句创建了一个数组的数组；第二个while语句创建了一个对象的数组。有关以一个键保存多个元素的内容请参见技巧4.2。

如果想使用字符串键，而不是整数键来定义数组，也可以使用array()，只不过需要用=>操作符来指定“键/值”：

```
$fruits = array('red' => 'Apples', 'yellow' => 'Bananas',
                'beige' => 'Cantaloupes', 'brown' => 'Dates');
```

现在，$fruits['beige']的值是'Cantaloupes'。这是对以下定义语句的简写：

```
$fruits['red'] = 'Apples';
$fruits['yellow'] = 'Bananas';
$fruits['beige'] = 'Cantaloupes';
$fruits['brown'] = 'Dates';
```

每个数组中的一个键只能保存一个唯一的值。如果添加：

```
$fruits['red'] = 'Strawberry';
```

那么，就会覆盖'Apples'值。不过，可以在定义数组之后再增加新的键：

```
$fruits['orange'] = 'Orange';
```

使用PHP编程的时间越久，你就越能感觉到自己更倾向于使用关联数组，而不是数字数组。不是创建一个保存着字符串值的数字数组，而是创建一个关联数组，把要保存的值作为数组中的键。如果你愿意，还可以把更多的信息保存为这个元素的值。这样做不会带来任何的速度损失，而且PHP也会自动保存键与额外信息的次序。此外，由于你事先知道相应的键，所以查询和修改保存的值也很简单。

循环遍历数组以及对其中全部或部分元素进行操作的最简单方式就是使用foreach语句：

```
$fruits = array('red' => 'Apples', 'yellow' => 'Bananas',
                'beige' => 'Cantaloupes', 'brown' => 'Dates');

foreach ($fruits as $color => $fruit) {
    print "$fruit are $color.\n";
}
Apples are red.
Bananas are yellow.
Cantaloupes are beige.
Dates are brown.
```

在执行每次循环时，PHP会把下一个键指定给$color，把与该键对应的值指定给$fruit。当循环到数组中最后一个元素后，循环退出。

用list()可以把一个数组分离到独立的变量中：

```
$fruits = array('Apples', 'Bananas', 'Cantaloupes', 'Dates');
list($red, $yellow, $beige, $brown) = $fruits;
```

4.1 定义一个起始元素不为零的数组

问题

你想要将多个元素一次指定给数组，但不希望第一个元素的索引值为0。

方案

用=>语法命令array()使用不同的索引：

```
$presidents = array(1 => 'Washington', 'Adams', 'Jefferson', 'Madison');
```

讨论

PHP中的数组 —— 与多数（不是全部）计算机语言类似 —— 从索引值0开始保存第一个项目。然而，有时候如果数组中保存的项目能够从索引值1开始会更有意义（这里，我们并不是非议Pascal程序员。译者注）。

在此方案中，George Washington是第一位总统，而不是第0个，所以如果你想打印出一份总统名单，显然这样更简单：

```
foreach ($presidents as $number => $president) {
    print "$number: $president\n";
}
```

而这样则麻烦一些：

```
foreach ($presidents as $number => $president) {
    $number++;
    print "$number: $president\n";
}
```

可以指定索引值不从0开始，并不意味只能指定从1开始，而是可以指定从任意整数开始：

```
$reconstruction_presidents = array(16 => 'Lincoln', 'Johnson', 'Grant');
```

同样地，也可以在一次调用中多次使用=>操作符：

```
$whig_presidents = array(9 => 'Harrison', 'Tyler' (注1) ,* 12 => 'Taylor', 'Fill more');
```

PHP甚至还允许你在调用数组时使用负索引值（实际上，这种方式对于非整数键也有效）。而从技术角度上讲，此时你所得到的是一个关联数组。我们曾经说过，在PHP中数字数组和关联数组之间的界限经常是模糊的，而这里也只是模糊的一种表现而已：

```
$us_leaders = array(-1 => 'George II', 'George III', 'Washington');
```

如果Washington是美国的第一位领袖，George III是第0位，而它的爷爷George II就是第−1位。

当然，你还可以在一个array()函数中混合使用数字和字符串键，但这种数组很容易混淆，所以较少使用：

```
$presidents = array(1 => 'Washington', 'Adams', 'Honest' => 'Lincoln', 'Jefferson');
```

这等价于：

```
$presidents[1]          = 'Washington';         // 键是 1
$presidents[]           = 'Adams';              // 键是 1 + 1 => 2
$presidents['Honest']   = 'Lincoln';            // 键是 'Honest'
$presidents[]           = 'Jefferson';          // 键是 2 + 1 => 3
```

参见

array()函数的文档（*http://www.php.net/array*）。

4.2 用数组中的一个键保存多个元素

问题

你想用一个键关联多个元素。

方案

在一个数组中保存多个元素：

注1： John Tyler是辉格党人Harrison的副总统，但在Harrison逝世后继任总统的任期内被驱逐。

```
$fruits = array('red' => array('strawberry','apple'),
                'yellow' => array('banana'));
```


或者使用一个对象：

```
while ($obj = mysql_fetch_object($r)) {
    $fruits[] = $obj;
}
```

讨论

在PHP中，每个数组中的键都是唯一的，所以如果不覆盖键的前一个值，就不可能将这个键与其他的多个项目关联。但是，可以把多个值保存在一个匿名数组中：

```
$fruits['red'][] = 'strawberry';
$fruits['red'][] = 'apple';
$fruits['yellow'][] = 'banana';
```

或者，如果你通过一个循环来处理多个项目，可以这样：

```
while (list($color,$fruit) = mysql_fetch_array($r)) {
    $fruits[$color][] = $fruit;
}
```

如果想输出这些项目，那么可以循环遍历这个数组：

```
foreach ($fruits as $color=>$color_fruit) {
    // $color_fruit是一个数组
    foreach ($color_fruit as $fruit) {
        print "$fruit is colored $color.<br>";
    }
}
```

或者，使用在技巧4.9中定义的pc_array_to_comma_string()函数。

```
foreach ($fruits as $color=>$color_fruit) {
    print "$color colored fruits include " .
        pc_array_to_comma_string($color_fruit) . "<br>";
}
```

在PHP 5.0.0及以上版本中，不需要使用pc_array.rang()方法，只要将增量作为第三个参数传递给rang()方法即可。

```
$odd = range(1, 52, 2);
$even = range(2, 52, 2);
```

参见

技巧4.9中有关输出带逗号的数组项目的介绍。

4.3 用一个整数范围来初始化数组

问题

你想用一个数组保存一系列连续的整数值。

方案

使用range($start, $stop)：

```
$cards = range(1, 52);
```

讨论

如果步长大于1，可以用以下自定义函数：

```
function pc_array_range($start, $stop, $step) {
    $array = array();
    for ($i = $start; $i <= $stop; $i += $step) {
        $array[] = $i;
    }
    return $array;
}
```

所以，对于连续的奇数，可用：

```
$odd = pc_array_range(1, 52, 2);
```

而对于连续的偶数，可用：

```
$even = pc_array_range(2, 52, 2);
```

在PHP 5.0.0及以上版本中，不需要使用pc_array_range()函数 —— 只需要将一个步长值作为第三个参数传递给range()函数即可：

```
$odd = range(1, 52, 2);
$even = range(2, 52, 2);
```

参见

技巧2.4中有关操作一系列整数的内容。range()函数的文档（*http://www.php.net/range*）。

4.4 遍历数组

问题

你想要循环遍历一个数组，并对其中的全部或部分元素进行操作。

方案

使用foreach语句：

```
foreach ($array as $value) {
    // 对$value进行操作
}
```

或者同时取得数组的键和值：

```
foreach ($array as $key => $value) {
    // 第二种
}
```

另一种技术是使用for语句：

```
for ($key = 0, $size = count($array); $key < $size; $key++) {
   // 第三种
}
```

最后，还可以在list()函数与while语句的组合中使用each()函数：

```
reset($array) // 将内部指针复位于数组的第一个元素处
while (list($key, $value) = each ($array)) {
    // 最后一种
}
```

讨论

在迭代数组元素时使用foreach语句更简洁一些：

```
// foreach 与值
foreach ($items as $cost) {
    ...
}
```

```
// foreach 与键和值
foreach($items as $item => $cost) {
    ...
}
```

在使用foreach语句时，PHP迭代的是相应数组的一个副本，而非数组本身。相反地，在使用each()函数和for语句时，PHP迭代的则是原始的数组。所以，如果你要在循环内部修改数组，你就应该选择（或不选择）某种方式。

如果你想修改数组，可以直接引用它：

```
reset($items);
while (list($item, $cost) = each($items)) {
    if (! in_stock($item)) {
        unset($items[$item]);           // 直接对其进行操作
    }
}
```

each()返回的变量不是数组中的原始值，而是原始值的副本，所以如果你修改这个变量，不会影响原始的数组。这正是要修改$items[$item]而不是$item的原因所在。

在使用each()函数时，PHP会跟踪循环的位置。当完成第一次循环后，再重新调用reset()函数将指针向前移动到数组中的前一个元素。如果指针越过了数组的末端，each()返回false。

for循环只能用于使用连续整数作为键的数组。除非你在同时修改数组的大小，否则如果每次循环都要重新计算$times的大小会影响效率，所以应该用一个$size变量来保存数组的大小值：

```
for ($item = 0, $size = count($items); $item < $size; $item++) {
    ...
}
```

如果你更喜欢用一个变量来计数，可以使用倒计法：

```
for ($item = count($items) - 1; $item >= 0; $item--) {
    ...
}
```

对关联数组使用for循环的示例如下：

```
for (reset($array); $key = key($array); next($array) ) {
    ...
}
```

如果数组中有一个值的计算结果是false，就会导致这种方式失效。所以一个像0这样完全正常的值都会导致循环提前结束。

最后，使用array_map()函数把数组中的每个元素发送到一个处理函数：

```
// 把数组中的所有单词转换成小写
$lc = array_map('strtolower', $words);
```

array_map()的第一个参数是用于修改单个数组元素的函数，第二个参数是要迭代的数组。

通常，我们感觉这种方法不如前一种方法更灵活，但它却非常适合处理并合并多个数组。

在不能确定要处理的数据是一个标量还是一个数组的情况下，需要避免对一个非数组变量使用foreach语句。一种方法就是使用is_array()函数：

```
if (is_array($items)) {
    // 针对数组的foreach 循环
} else {
    // 针对标题的代码
}
```

另一种方法就是使用settype()函数将所有变量都强制转换成数组形式：

```
settype($items, 'array');
 // 针对数组的循环代码
```

这种方法会把标量值转换成一个单元素数组，虽然增加了一些系统开销，但能使代码更清晰。

参见

for语句的文档（*http://www.php.net/for*），foreach语句的文档（*http://www.php.net/foreach*），while语句的文档（*http://www.php.net/while*），each()函数的文档（*http://www.php.net/each*），reset()函数的文档（*http://www.php.net/reset*）和array_map()函数的文档（*http://www.php.net/array-map*）。

4.5 从数组中删除元素

问题

你想从数组中删除一个或多个元素。

方案

要删除一个元素，用unset()：

```
unset($array[3]);
unset($array['foo']);
```

要删除多个不连续的元素，也用unset()：

```
unset($array[3], $array[5]);
unset($array['foo'], $array['bar']);
```

要删除多个连续的元素，用array_splice()：

```
array_splice($array, $offset, $length);
```

讨论

用这些函数可以从PHP中删除对这些元素的引用。如果你想在数组中保留某个键，并将其值设为空值，可以把空字符串指定给这个元素：

```
$array[3] = $array['foo'] = '';
```

除了语法外，用unset()和为元素指定空字符串还有一个逻辑上的区别。前者意味着“这个元素不存在了”，而后者意味着“该元素仍然存在，只是值为空”。

如果处理的是数字，指定0值会更好。所以，如果公司停产了XL1000型号链轮齿，那么应该像这样来更新商品目录：

```
unset($products['XL1000']);
```

然而，如果公司只是暂时停止生产XL1000型的链轮齿，并计划在本周的最后几天再接受一个来自工厂的订货，那么这样比较好：

```
$products['XL1000'] = 0;
```

如果你unset()一个元素，PHP会调整数组以便循环仍然可以正常完成。但不会因此压缩数组以填补缺少的元素位置。这也正是我们之所以说所有数组都是关联数组的原因所在，即使数组是以数字数组的形式出现也是如此。请看下面的例子：

```
// 创建一个数字数组
$animals = array('ant', 'bee', 'cat', 'dog', 'elk', 'fox');
print $animals[1];    // 输出'bee'
print $animals[2];    // 输出'cat'
count($animals);      // 返回6
```

```
// unset()
unset($animals[1]);   // 删除元素$animals[1] = 'bee'
print $animals[1];    // 输出 '' 并抛出一个 E_NOTICE 错误
print $animals[2];    // 仍然输出 'cat'
count($animals);      // 返回5，即使$array[5]是'fox'

// 添加新元素
$animals[] = 'gnu';   // 添加新元素(非Unix系统)
print $animals[1];    // 输出''，即仍然是空值
print $animals[6];    // 输出'gnu'，该位置是'gnu'的位置
count($animals);      // 返回6

// 指定''
$animals[2] = '';     // 清零
print $animals[2];    // 输出''
count($animals);      // 返回6,数组中元素数并未减少
```

如果想把数组压缩成一个稠密填充的数字数组，可以用array_values()：

```
$animals = array_values($animals);
```

可替代的方案是用array_splice()，它可以自动地重新索引数组以清除空缺的元素位置：

```
// 创建一个数字数组
$animals = array('ant', 'bee', 'cat', 'dog', 'elk', 'fox');
array_splice($animals, 2, 2);
print_r($animals);
Array
(
    [0] => ant
    [1] => bee
    [2] => elk
    [3] => fox
)
```

如果你把数组当成一个队列看待，并希望从队列中删除一个项目后，仍然能以随机方式访问其中的项目，这种方法非常合适。如果想可靠地删除数组中的第一个或最后一个元素，就要分别用到array_shift()和array_pop()这两个函数。

如果经常发现自己被数组中的空位所困扰，多半是因为你没有“以PHP的方式进行思考”。那么请看一看技巧4.4中不用for循环来实现迭代数组的讨论。

参见

技巧4.4中有关迭代数组技术的讨论。unset()函数的文档（*http://www.php.net/unset*），array_splice()函数的文档（*http://www.php.net/array-splice*）和array_values()函数的文档（*http://www.php.net/array-values*）。

4.6 改变数组大小

问题

你想修改数组的大小，即增大或减小当前数组的大小。

方案

使用array_pad()使用数组增大：

```
// 最初大小为3
$array = array('apple', 'banana', 'coconut');

// 增大到5
$array = array_pad($array, 5, '');
```

现在，count($array)的值是5，而最后两个元素，$array[3]和$array[4]中包含的是空字符串。

要减小数组大小，可以使用array_splice()：

```
// 未对$array赋值
array_splice($array, 2);
```

这样就把除了前两个元素之外的所有元素从$array中删除了。

讨论

PHP中的数组并没有预定义的大小，所以可以随时修改其大小。

要扩充一个数组，使用array_pad()。其第一个参数是要扩充的数组，第二个参数表示想要扩充的大小和方向。如果想向右扩充，那么使用正整数；想向左扩充，就使用负整数。第三个参数是要赋给新创建元素的值。这个函数会返回修改后的数组，但不会修改原来的数组。

下面请看几个例子：

```
// 向右扩充生成一个包含四个元素的数组，第四个元素的值为'dates'
$array = array('apple', 'banana', 'coconut');
$array = array_pad($array, 4, 'dates');
print_r($array);
Array
(
    [0] => apple
```

```
    [1] => banana
    [2] => coconut
    [3] => dates
)
// 向左扩充生成一个包含六个元素的数组，最左端两个元素的值为'zucchini'
$array = array_pad($array, -6, 'zucchini');
print_r($array);
Array
(
    [0] => zucchini
    [1] => zucchini
    [2] => apple
    [3] => banana
    [4] => coconut
    [5] => dates
)
```

请注意：array_pad($array, 4, 'dates')可以保证生成一个至少包含四个元素的数组，但它不是添加四个新元素。此时，如果$array中已经包含了四个或更多个元素，array_pad()则会返回原来的数组。

同样地，如果为第四个元素$array[4]声明了一个值：

```
$array = array('apple', 'banana', 'coconut');
$array[4] = 'dates';
```

那么所得到数组的索引依次是0,1,2,和4：

```
Array
(
    [0] => apple
    [1] => banana
    [2] => coconut
    [4] => dates
)
```

实际上，PHP在这种情况下把数组转换成了一个以整数作为键的关联数组。

与array_pad()函数不同，array_splice()有修改原先数组的副作用。它返回的是经过拼合后的数组。然而，与array_pad()相似的是，也可以从左、右两端对数组进行拼合。所以如果在调用array_splice()时使用-2作为参数，那么就会将数组中倒数后两个元素删掉。

```
// 生成一个包含四个元素的数组
$array = array('apple', 'banana', 'coconut', 'dates');

// 缩减为三个元素
array_splice($array, 3);
```

```
// 删除最后一个元素，等价于使用array_pop()
array_splice($array, -1);

// 只剩下水果中的apple和banana
print_r($array);
Array
(
    [0] => apple
    [1] => banana
)
```

参见

array_pad函数的文档（*http://www.php.net/array-pad*）和array_splice()函数的文档（*http://www.php.net/array-splice*）。

4.7 将一个数组追加到另一个数组

问题

你想要把两个数组合并成一个数组。

方案

使用array_merge()：

```
$garden = array_merge($fruits, $vegetables);
```

讨论

array_merge()函数使用的数组，可以是预定义的数组，也可以是在调用时用array()定义的数组：

```
$p_languages = array('Perl', 'PHP');
$p_languages = array_merge($p_languages, array('Python'));
print_r($p_languages);
Array
(
    [0] => PHP
    [1] => Perl
    [2] => Python
)
```

也就是说，参与合并的数组可以是已经存在的数组，如$p_languages，也可是匿名数组，如array('Python')。

由于PHP不能自动地把数组展开为一系列互不相关的变量，所以不能在合并数组时使用array_push()，否则所得到的将是嵌套的数组。像这样：

```
array_push($p_languages, array('Python'));
print_r($p_languages);
Array
(
    [0] => PHP
    [1] => Perl
    [2] => Array
        (
            [0] => Python
        )

)
```

在合并数组时，如果使用数字键，那么索引会重新编号，以保证值不会丢失。如果使用字符串键，则会导致第二个数组中的键覆盖第一个数组中的同名键。如果是带有两种键的数组，那么合并时也会表现出上述两种行为特征。例如：

```
$lc = array('a', 'b' => 'b'); // 值为小写字母
$uc = array('A', 'b' => 'B'); // 值为大写字母
$ac = array_merge($lc, $uc);  // 混合大小写?
print_r($ac);
Array
(
    [0] => a
    [b] => B
    [1] => A
)
```

为了避免冲突，大写字母A的索引经过重新编号，由0变成了1，并被添加到了新数组的末尾。同时，在新数组中大写字母B则覆盖了小写字母b并取代了其在数组中的原始位置。

使用+操作符也可以实现合并数组的功能。此时，右边的数组会覆盖左边的数组中所有重名的键。而且，也不会为避免冲突而进行索引重排。还举上面的例子：

```
print_r($uc + $lc);
print_r($lc + $uc);
Array
(
    [0] => a
    [b] => b
)
```

```
Array
(
    [0] => A
    [b] => B
)
```

因为a和A的键都是数字0，而b和B的键也都是b，所以合并后的数组仍然只包含两个元素。

在第一种情况下，$a + $b的结果是剩下$b，而第二种情况下，$b + $a的结果是剩下$a。

不过，如果合并数组中没有重复的键，这个问题也就不存在了，而合并后的新数组将是两个数组的联合。

参见

array_merge()函数的文档（*http://www.php.net/array-merge*）。

4.8 把数组转换成字符串

问题

你想把一个数组转换成一种格式规范的字符串形式。

方案

使用join()：

```
// 生成以逗号分隔的列表
$string = join(',', $array);
```

或者手工定义循环：

```
$string = '';

foreach ($array as $key => $value) {
    $string .= ",$value";
}

$string = substr($string, 1);          // 删除第一个","
```

讨论

如果能够使用join()函数，没问题 —— 而且它还会比基于PHP的循环速度更快。但是，join()函数却并不是很灵活。首先，它只能在元素之间添加分隔符，而不是在元素周围。如果想把元素包围在一对HTML标记中，并以逗号对其进行分隔，就要这样：

```
$left  = '<b>';
$right = '</b>';

$html = $left . join("$right,$left", $html) . $right;
```

其次，join()函数不能对值区别对待。如果你只想要包含某些元素的值，则需要自己动手写循环代码：

```
$string = '';

foreach ($fields as $key => $value) {
    // 不包含密码
    if ('password' != $key) {
        $string .= ",<b>$value</b>";
    }
}

$string = substr($string, 1);          // 删除第一个","
```

注意，分隔符始终是先添加到每个值之前，然后再在循环外面去掉第一个。虽然先加再减的方式有点烦琐，但也远比试图在循环内部嵌入额外的逻辑更清晰、更有效率（多数情况下）。如果多花些心思的话，又可以写成这样：

```
$string = '';
foreach ($fields as $key => $value) {
    // 不包含密码
    if ('password' != $key) {
        if (!empty($string)) { $string .= ','; }
        $string .= "<b>$value</b>";
    }
}
```

这样，你就必须在每次向$string追加值时都检查一下$string。显然就不如只调用一次substr()函数。出于同样的考虑，前置分隔符（此处的逗号），而不是后置分隔符，是因为从前面缩短字符串，要快于从后面缩短字符串。

参见

技巧4.9中有关使用逗号输出数组的内容。join()函数的文档（*http://www.php.net/join*）和substr()函数的文档（*http://www.php.net/substr*）。

4.9 使用逗号来打印数组

问题

你想要以逗号作为分隔符来输出数组，而且当数组中有超过两个以上的元素时，还要在最后一个元素之前打印“and”。

方案

使用例4-1中定义的pc_array_to_comma_string()函数，该函数可以返回恰当的字符串。

例4-1：pc_array_to_comma_string()函数

```
function pc_array_to_comma_string($array) {

    switch (count($array)) {
    case 0:
        return '';

    case 1:
        return reset($array);

    case 2:
        return join(' and ', $array);

    default:
        $last = array_pop($array);
        return join(', ', $array) . ", and $last";
    }
}
```

讨论

如果你有一个准备打印的项目列表，那么以一种符合规范的正确格式打印出来是很有用的。如果是像这样来显示文本，就有点太不方便使用了：

```
$thundercats = array('Lion-O', 'Panthro', 'Tygra', 'Cheetara', 'Snarf');
print 'ThunderCat good guys include ' . join(', ', $thundercats) . '.';
ThunderCat good guys include Lion-O, Panthro, Tygra, Cheetara, Snarf.
```

之所以这个函数的实现不是那么简明，是因为我们希望pc_array_to_comma_string()函数不仅能适合数字数组使用，而是要适合所有类型的数组使用。如果只局限于数字数组，那么对于只包含一个元素的数组，可以返回$array[0]。但要是数组索引不从0开始，$array[0]返回的就是空值。不过，你可以利用reset()函数能够复位数组的内部指针并返回数组中第一个元素值的事实。

出于类似的原因，我们调用array_pop()函数取出最后一个元素，而不是通过其位置$array[count($array)-1]来取得该元素。因为，之后我们可以对数组直接使用join()。

也要注意，case 2中的代码对于case 1也适用。而default部分的代码对于case 2也同样适用（虽然效率低）。不过，这种推理不能延伸，也就是说不能把default部分的代码用于case 1。

参见

技巧4.8中将数组转换成字符串的内容。join()函数的文档（*http://www.php.net/join*），array_pop()函数的文档（*http://www.php.net/array-pop*）和reset()函数的文档（*http://www.php.net/reset*）。

4.10 检查数组中是否存在某个键

问题

你想知道数组中是否包含某个特定的键。

方案

使用array_key_exists()来检查数组元素的键（不考虑对应的值）。

```
if (array_key_exists('key', $array)) {
    /* $array['key']存在 */
}
```

使用isset()来检查是否存在一个值不为null的键：

```
if (isset($array['key'])) { /* 在 $array 中键为 'key' 的元素值非空*/ }
```

讨论

array_key_exists()函数完全忽略数组中的值 —— 它只负责报告数组中是否存在具有特定键的元素。而isset()函数则会以对待其他变量的方式来检查数组中的键。如果该键对应的值为null，isset()函数返回false。请参考第5章中更多关于变量真值的内容。

参见

isset()函数的文档（*http://www.php.net/isset*）以及array_key_exists()函数的文档（*http://www.php.net/array_key_exists*）。

4.11 检查数组中是否包含某个元素

问题

你想知道数组中是否包含一个特定的值。

方案

使用in_array()：

```
if (in_array($value, $array)) {
    // 在数组 $array 中有一个值为$value 的元素
}
```

讨论

用in_array()来检查数组中是否有一个元素包含着特定的值：

```
$book_collection = array('Emma', 'Pride and Prejudice', 'Northhanger Abbey');
$book = 'Sense and Sensibility';

if (in_array($book, $book_collection) {
    echo 'Own it.';
} else {
    echo 'Need it.';
}
```

in_array()函数在默认情况下使用==操作符来比较两个项目是否相等。如果使用严格相等===作为标准，则须将true作为第三个参数传递给in_array()：

```
$array = array(1, '2', 'three');

in_array(0, $array);                    // true!
in_array(0, $array, true);              // false
in_array(1, $array);                    // true
in_array(1, $array, true);              // true
in_array(2, $array);                    // true
in_array(2, $array, true);              // false
```

第一次检查，in_array(0, $array)，计算的值是true，是因为将数字0与字符串three比

较的结果。PHP会把three转换成一个整数，因为three不是一个数字字符串（比如2），所以就变成了0。因此，in_array()就认为存在一个匹配的值。

所以，在把数字与包含字符串的数据比较时，最保险的方式就是使用严格型比较。

如果你预感到会在同一个数组上多次调用in_array()函数，可能使用关联数组更好一些，这个新的关联数组是以原始的数组元素作为键的。用in_array()查寻项目时，所用时间与数组项目成正比，而对于关联数组，时间则是恒定不变的。

如果不能直接创建关联数组，而是需要从传统的整数键数组进行转换，可以用array_flip()来包装这个数组中的键和值：

```
$book_collection = array('Emma',
                         'Pride and Prejudice',
                         'Northhanger Abbey');

// 把数字数组转换为关联数组
$book_collection = array_flip($book_collection);
$book = 'Sense and Sensibility';

if (isset($book_collection[$book])) {
    echo 'Own it.';
} else {
    echo 'Need it.';
}
```

注意，这样一来，在交换后的数组中具有相同值的多个键只会保留为一个元素。

参见

技巧4.12中确定值在数组中位置的内容。in_array()函数的文档（*http://www.php.net/in-array*）和array_flip()函数的文档（*http://www.php.net/array-flip*）。

4.12 确定值在数组中的位置

问题

你想知道数组中是否包含一个值，如果包含这个值，那么要知道它的键是什么。

方案

使用array_search()。该函数返回匹配值的键。如果未找到匹配的值，则返回false。

```
$position = array_search($value, $array);
if ($position !== false) {
    // 在数组$array中，位置$position处的值$value就是该元素在数组中的值
}
```

讨论

用in_array()可以查寻是否数组中包含一个值，用array_search()可以找到该值所在的位置。但是，由于array_search()能够适当地处理找不到相应值的情况，所以使用array_search()比使用in_array()更好。两者的速度之差很微小，但潜在的信息却格外有用：

```
$favorite_foods = array(1 => 'artichokes', 'bread', 'cauliflower', 'deviled eggs');
$food = 'cauliflower';
$position = array_search($food, $favorite_foods);

if ($position !== false) {
    echo "My #$position favorite food is $food";
} else {
    echo "Blech! I hate $food!";
}
```

之所以用!==来检查false值，是因为如果字符串在数组中的位置0处，那么if会将0计算为逻辑false，而事实并非如此，也不是我们所希望的。

如果数组中包含多个要搜索的值，array_search()只能保证返回其中的一个技巧，但不一定是第一个技巧。

参见

技巧4.11中有关检查数组中是否存在特定元素的方法。array_search()函数的文档（*http://www.php.net/array-search*）。第22章以及*http://www.php.net/preg-replace*中使用正则表达式进行更高级的数组搜索的内容。

4.13 确定通过某种测试的元素

问题

你想查找数组中符合某种条件的项目。

方案

使用foreach循环：

```
$movies = array(...);

foreach ($movies as $movie) {
    if ($movie['box_office_gross'] < 5000000) { $flops[] = $movie; }
}
```

或者使用array_filter()：

```
$movies = array(...);

function flops($movie) {
    return ($movie['box_office_gross'] < 5000000) ? 1 : 0;
}

$flops = array_filter($movies, 'flops');
```

讨论

foreach循环很简单：迭代数组，并将满足条件的元素追加到返回的数组中。

如果你只想取得第一个满足条件的元素，可以使用break退出循环：

```
foreach ($movies as $movie) {
    if ($movie['box_office_gross'] > 200000000) { $blockbuster = $movie; break; }
}
```

当然，也可以直接从函数中返回：

```
function blockbuster($movies) {
    foreach ($movies as $movie) {
        if ($movie['box_office_gross'] > 200000000) { return $movie; }
    }
}
```

在使用array_filter()时，首先需要创建一个回调函数，该函数会对满足条件的值返回true，而对不满足条件的值返回false。创建完成回调函数后，你就可以像在foreach语句中那样来命令PHP通过array_filter()来处理相应的数组了。

由于使用array_filter()函数不可能提前退出，所以foreach语句显得更具灵活性，而且更容易理解。同时，这也是PHP的内置函数不如用户级代码更清晰的少数情况之一。

参见

array_filter()函数的文档（*http://www.php.net/array-filter*）。

4.14 确定数组中经计算后的最大或最小元素

问题

你有一个元素数组，并想找出其中经计算后的最大或最小元素。例如，在创建一个直方图的时候，你想计算出适当的比例。

方案

要找到最大的元素，使用max()：

```
$largest = max($array);
```

要找到最小的元素，使用min()：

```
$smallest = min($array);
```

讨论

通常，max()会返回两个元素中较大的一个，但如果你给它传递一个数组，它就会改为搜索整个数组。遗憾的是，用max()函数无法找到最大元素的索引。如果想找到最大元素的索引，则必须对数组以颠倒的顺序进行重新排序，把最大的元素放在位置0：

```
arsort($array);
```

现在，数组中最大元素的值就是$array[0]。

如果你不想打乱原先数组中元素的次序，那么可以先复制一个副本，改为对副本进行排序：

```
$copy = $array;
arsort($copy);
```

以上讨论的内容也同样适用于min()，只不过在排序时要使用asort()而不是arsort()。

参见

技巧4.16中有关对数组排序的讨论。max()函数的文档（*http://www.php.net/max*），

min()函数的文档（*http://www.php.net/min*），arsort()函数的文档（*http://www.php.net/arsort*）以及asort()函数的文档（*http://www.php.net/asort*）。

4.15 反转数组

问题

你想反转数组中元素的次序。

方案

使用array_reverse()：

```
$array = array('Zero', 'One', 'Two');
$reversed = array_reverse($array);
```

讨论

array_reverse()函数会将数组中元素的次序反转过来。然而，通常情况下要避免这种操作。如果你希望把刚经过排序的数组反转过来，就要修改次序来实现反转。如果你想要反转一个打算要遍历及进行处理的列表，只需反转循环即可。即不是用：

```
for ($i = 0, $size = count($array); $i < $size; $i++) {
    ...
}
```

而是用：

```
for ($i = count($array) - 1; $i >=0 ; $i--) {
    ...
}
```

然而，如前所述，只能在紧密压缩的数组上使用for循环。

另一个备选方案就是，如果可能的话，可以事先反转放到数组中的元素。例如，如果你要从数据库中返回的一系列记录组织成一个数组，你应该可以在查询语句中加上ORDER DESC —— 在你所使用的数据库手册中查询实际的语法。

参见

array_reverse()函数的文档（*http://www.php.net/array-reverse*）。

4.16 数组排序

问题

你想要以一种特殊的方式对数组进行排序。

方案

要按惯例对数组进行排序，使用sort()：

```
$states = array('Delaware', 'Pennsylvania', 'New Jersey');
sort($states);
```

要以数字为标准排序，把SORT_NUMERIC作为第二个参数传递给sort()：

```
$scores = array(1, 10, 2, 20);
sort($scores, SORT_NUMERIC);
```

这样会按数字升序，而不是按词典中的顺序对数组进行重排。

讨论

sort()函数不会保留元素间的键/值关联，相反，经过排序后所有的索引都会从0开始以升序进行重新排定（此规则有一个例外，就是对于单元素数组，不会将单个元素的索引重置为0。但这个问题在PHP 4.2.3版之后得到了修正）。

要想保留键/值关联，可以使用asort()函数。asort()函数通常用于关联数组，但当索引具有某种特定的含义时也可以使用该函数：

```
$states = array(1 => 'Delaware', 'Pennsylvania', 'New Jersey');
asort($states);

while (list($rank, $state) = each($states)) {
    print "$state was the #$rank state to join the United States\n";

}
```

使用natsort()可以对数组按自然的排序算法进行排序。在自然排序的基础之上，即使数组元素中混合了字符串和数字也能得到正确的排序结果：

```
$tests = array('test1.php', 'test10.php', 'test11.php', 'test2.php');
natsort($tests);
```

现在，数组中元素的顺序为：'test1.php'，'test2.php'，'test10.php'和'test11.php'。在使用自然排序时，数字10会排序到数字2的后面，而在使用传统的排序算法时，结果则相反。若要进行区分大小写的自然排序，可以使用natcasesort()。

要想相反的算法对数组中的元素排序（或反转排序后的结果），使用rsort()或arsort()（与rsort()相似，只是保留了键/值关系）。没有natrsort()或natcasersort()函数。也可以把SORT_NUMERIC传递给这两个函数。

参见

技巧4.17中关于根据自定义的比较函数、技巧4.18中对多个数组进行排序的内容。sort()函数的文档（*http://www.php.net/sort*），asort()函数的文档（*http://www.php.net/asort*），natsort()函数的文档（*http://www.php.net/natsort*），natcasesort()函数的文档（*http://www.php.net/natcasesort*），rsort()函数的文档（*http://www.php.net/rsort*）和arsort()函数的文档（*http://www.php.net/arsort*）。

4.17 根据可计算的字段对数组进行排序

问题

你想要定义自己的排序程序。

方案

使用usort()来按自定义的比较程序对数组进行排序。

```
// 按反自然顺序排序
function natrsort($a, $b) {
    return strnatcmp($b, $a);
}

$tests = array('test1.php', 'test10.php', 'test11.php', 'test2.php');
usort($tests, 'natrsort');
```

讨论

所用的排序函数必须在$a > $b时返回大于0的值；在$a == $b时返回0；而在$a < $b时返回小于0的值。如果要以相反的规则进行排序，则需要返回相反的值。本方案中的排序函数——strnatcmp()——就遵循了这些规则。

如果想反转排序规则，不用给strnatcmp($a, $b)返回的值乘上-1，只需将strnatcmp($b, $a)的两个参数交换位置即可。

不需要为一个存在的次序而包装排序函数。例如，例4-2中的pc_date_sort()函数，显示了如何对日期进行排序。

例4-2：pc_date_sort()函数

```
// 预定的日期格式为"MM/DD/YYYY"
function pc_date_sort($a, $b) {
    list($a_month, $a_day, $a_year) = explode('/', $a);
    list($b_month, $b_day, $b_year) = explode('/', $b);

    if ($a_year  > $b_year ) return  1;
    if ($a_year  < $b_year ) return -1;

    if ($a_month > $b_month) return  1;
    if ($a_month < $b_month) return -1;

    if ($a_day   > $b_day  ) return  1;
    if ($a_day   < $b_day  ) return -1;

    return 0;
}

$dates = array('12/14/2000', '08/10/2001', '08/07/1999');
usort($dates, 'pc_date_sort');
```

在排序过程中，每当usort()函数需要比较两个元素大小时，都要计算排序函数的返回值，这就降低了排序的速度。为了减少不必要的计算，可以用例4-3所示的pc_array_sort()函数来缓存要进行比较的值。

例4-3：pc_array_sort()函数

```
function pc_array_sort($array, $map_func, $sort_func = '') {
    $mapped = array_map($map_func, $array);     // 缓存 $map_func() 的值

    if ('' == $sort_func) {
        asort($mapped);                         // asort()比usort()速度快
    } else {
        uasort($mapped, $sort_func);            // 需要保留键/值关系
    }

    while (list($key) = each($mapped)) {
        $sorted[] = $array[$key];               // 使用排序后的键
    }

    return $sorted;
}
```

为了减少不必要的计算，pc_array_sort()使用了一个临时数组$mapped来缓存返回的值。然后，使用默认的排序规则或者用户指定的排序程序对$mapped进行排序。关键在于，该函数进行的是保留键/值关系的排序。在默认情况下，使用asort()是因为其速度比uasort()更快（使用uasort()是pc_array_sort()变慢的根本原因）。最后，它又创建了一个经过排序的数组$sorted，其中使用在$mapped中经过排序后的键来索引原始数组中的值。

对于小型数组或者简单的排序函数，usort()更快一些，但随着计算量的增加，pc_array_sort()的速度就会超过usort()。下面根据元素中字符串的长度进行排序的例子，就是一个自定义排序函数更快的例子：

```
function pc_u_length($a, $b) {
    $a = strlen($a);
    $b = strlen($b);

    if ($a == $b) return  0;
    if ($a  > $b) return  1;
                  return -1;
}

function pc_map_length($a) {
    return strlen($a);
}

$tests = array('one', 'two', 'three', 'four', 'five',
               'six', 'seven', 'eight', 'nine', 'ten');

// 少于5个元素时，使用 pc_u_length() 速度快
usort($tests, 'pc_u_length');

// 大于或等于5个元素时，使用 pc_map_length() 的速度更快
$tests = pc_array_sort($tests, 'pc_map_length');
```

在这个例子中，一旦数组中的元素达到5个（或以上），使用pc_array_sort()就会比使用usort()的速度更快。

参见

技巧4.16中有关基本排序的讨论和技巧4.18中对多个数组进行排序的讨论。usort()函数的文档（*http://www.php.net/usort*），asort()函数的文档（*http://www.php.net/asort*）和array_map()函数的文档（*http://www.php.net/array-map*）。

4.18 对多个数组进行排序

问题

你想要对多个数组、或者一个多维数组进行排序。

方案

使用array_multisort()：

要同时对多个数组进行排序，只需将多个要排序的数组传递给array_multisort()：

```
$colors = array('Red', 'White', 'Blue');
$cities = array('Boston', 'New York', 'Chicago');

array_multisort($colors, $cities);
print_r($colors);
print_r($cities);
Array
(
    [0] => Blue
    [1] => Red
    [2] => White
)
Array
(
    [0] => Chicago
    [1] => Boston
    [2] => New York
)
```

要对一个多维数组进行排序，则需要传递相关的数组元素：

```
$stuff = array('colors' => array('Red', 'White', 'Blue'),
               'cities' => array('Boston', 'New York', 'Chicago'));

array_multisort($stuff['colors'], $stuff['cities']);
print_r($stuff);
Array
(
    [colors] => Array
        (
            [0] => Blue
            [1] => Red
            [2] => White
        )

    [cities] => Array
        (
```

```
            [0] => Chicago
            [1] => Boston
            [2] => New York
        )

)
```

要像在sort()函数中那样修改排序的规则，可以在数组的后面传递常量SORT_REGULAR、SORT_NUMERIC或SORT_STRING。如果想修改排序的次序（与sort()函数中不同），则要在数组后面传递常量SORT_ASC或SORT_DESC。当然，也可以在数组后面同时传递表示排序规则和排序次序的两个常量。

讨论

array_multisort()函数可以一次完成对多个数组，或者一个包含不同大小的多维数组的排序。这些数组被看成是一个表格中的列，该函数则按照行进行排序。其中，第一个数组是主要排序数组，其他数组中的排序规则都参照第一个数组中的排序规则。如果第一个数组中各个元素比较的结果相等，那么排序规则就由第二个数组决定，依次类推。

其默认的排序常量为SORT_REGULAR和SORT_ASC，而且在对每个数组排序后都会重置这两个常量，所以除非要修改规则，否则不需要明确地传递这两个常量：

```
$numbers = array(0, 1, 2, 3);
$letters = array('a', 'b', 'c', 'd');
array_multisort($numbers, SORT_NUMERIC, SORT_DESC,
                $letters, SORT_STRING , SORT_DESC);
```

这个例子将数组中元素的次序进行了反转。

参见

技巧4.16中简单排序的内容和技巧4.17中用自定义函数对数组进行排序的内容。array_multisort()函数的文档（*http://www.php.net/array-multisort*）。

4.19 使用方法而不是函数来对数组进行排序

问题

你想用一个自定义的程序对数组进行排序。但是，不是使用函数，而是要使用一个对象的方法。

方案

传递一个包含类名和方法名的数组来代替函数名：

```
usort($access_times, array('dates', 'compare'));
```

讨论

如同使用自定义的排序函数一样，对象的方法也要接受两个输入的参数，并根据第一个参数是大于、等于还是小于第二个参数，返回1，0或者－1：

```
class pc_sort {
    // 反序字符串比较
    function strrcmp($a, $b) {
        return strcmp($b, $a);
    }
}

usort($words, array('pc_sort', 'strrcmp'));
```

参见

第7章中更多有关类和对象的内容。技巧4.17中更多自定义排序数组的内容。

4.20 对数组进行随机化处理

问题

你想要随机地对数组中的元素进行排序。

方案

使用shuffle()：

```
shuffle($array);
```

讨论

要想完全打乱数组中元素的次序，必须慎重行事。事实上，直至PHP 4.3，PHP的shuffle()例程都没有真正实现对数组的随机化处理。它只是将元素混合到一起，但不同的混合结果之间仍然很相近。

因此，需要在有必要时再使用PHP的shuffle()函数。

参见

shuffle()函数的文档（*http://www.php.net/shuffle*）。

4.21 删除数组中重复的元素

问题

你想要从数组中删除重复的元素。

方案

如果是创建完成的数组，使用array_unique()，该函数会返回一个只包含唯一元素的新数组：

```
$unique = array_unique($array);
```

如果是在程序运行时根据结果来创建数组，那么对于创建数字数组可以使用以下技术：

```
foreach ($_REQUEST['fruits'] as $fruit) {
    if (!in_array($array, $fruit)) { $array[] = $fruit; }
}
```

对于创建关联数组可以使用以下技术：

```
foreach ($_REQUEST['fruits'] as $fruit) {
    $array[$fruit] = $fruit;
}
```

讨论

如果是对于创建完成的数组，要删除重复元素的最好方法就是使用array_unique()函数。但是，如果是在一个循环当中，则需要通过检查数组中是否已经存在相同的元素来决定是否排除一个元素。

比使用in_array()更快的方法是创建一个每个元素的键和值都相同的混合数组。这样不仅可以消除in_array()方法的线性查寻，而且仍然能够利用对数组值而不是键进行操作的函数。

实际上，如果先使用关联数组的方法，然后再对结果调用array_values()函数（此时还可以使用array_key()函数，但array_values()稍快一些），会比直接使用in_array()来创建数字数组更快。

参见

array_unique()函数的文档（*http://www.php.net/array-unique*）。

4.22 对数组中的每个元素都应用一个函数

问题

你想要对数组中的每个元素都应用一个函数或者方法，以便一次性地转换所有的输入数据。

方案

使用array_walk()：

```
function escape_data(&$value, $key) {
    $value = htmlentities($value, ENT_QUOTES);
}

$names = array('firstname' => "Baba",
               'lastname'  => "O'Riley");

array_walk($names, 'escape_data');

foreach ($names as $name) {
    print "$name\n";
}

Baba
O&#039;Riley
```

对于嵌套的数据，使用array_walk_recursive()：

```
function escape_data(&$value, $key) {
    $value = htmlentities($value, ENT_QUOTES);
}

$names = array('firstnames' => array("Baba", "Bill"),
               'lastnames'  => array("O'Riley", "O'Reilly"));

array_walk_recursive($names, 'escape_data');
```

```
foreach ($names as $nametypes) {
    foreach ($nametypes as $name) {
        print "$name\n";
    }
}

Baba
Bill
O&#039;Riley
O&#039;Reilly
```

讨论

对数组中的元素进行循环遍历是很常用的。为此，可以使用foreach语句。但，有一个可替代的选择就是array_walk()函数。

这个函数接受一个数组和一个回调函数 —— 用于处理数组中元素的函数。而回调函数也接受两个参数，一个是值，一个是键。回调函数也可以接受第三个可选的参数，这个参数表示你希望在回调函数中暴露的额外数据。

下面是一个保证对数组$name中的所有数据进行适当的HTML编码的例子。其中的回调函数escape_data()，接收数组中元素的值，并将其传递给htmlentities()函数，完成对关键HTML实体的编码，最后把编码后的结果赋给$value：

```
function escape_data(&$value, $key) {
    $value = htmlentities($value, ENT_QUOTES);
}

$names = array('firstname' => "Baba",
               'lastname'  => "O'Riley");

array_walk($names, 'escape_data');

foreach ($names as $name) {
    print "$name\n";
}

Baba
O&#039;Riley
```

由于array_walk函数进行的是内部的操作而不是返回修改后数组的副本，所以当你想要修改这些元素时，必须以引用的方式传递这些值。如果是那样的话，也就是说和本例一样，要在参数名的前面加上一个&。然而，只有当你想要修改这个数组时才有必要这样。

如果要操作一系列嵌套的数组，那么可以使用array_walk_recursive()函数：

```
function escape_data(&$value, $key) {
    $value = htmlentities($value, ENT_QUOTES);
}

$names = array('firstnames' => array("Baba", "Bill"),
               'lastnames'  => array("O'Riley", "O'Reilly"));

array_walk_recursive($names, 'escape_data');

foreach ($names as $nametypes) {
    foreach ($nametypes as $name) {
        print "$name\n";
    }
}

Baba
Bill
O&#039;Riley
O&#039;Reilly
```

array_walk_recursive()函数只把非数组元素传递给回调函数，所以如果是从array_walk()函数改为使用这个函数，没有必要修改回调函数。

参见

array_walk()函数的文档（http://*www.php.net/array-walk*），array_walk_recursive()函数的文档（*http://www.php.net/array_walk_recursive*）和htmlentities()函数的文档（*http://www.php.net/htmlentities*）。

4.23 计算两个数组的并集、交集和差集

问题

你有两个数组，想要计算它们的并集（包含两个数组中的所有元素）、交集（包含两个数组中共有的元素）和差集（包含两个数组中独有的元素）。

方案

计算并集：

```
$union = array_unique(array_merge($a, $b));
```

计算交集：

```
$intersection = array_intersect($a, $b);
```

计算简单的差集：

```
$difference = array_diff($a, $b);
```

计算对称差集：

```
$difference = array_merge(array_diff($a, $b), array_diff($b, $a));
```

讨论

PHP已经内置了完成类似计算的必要组件，问题只是如何以适当的序列来组合运用这些组件。

在计算并集时，首先是将两个数组合并成一个同时包含这两个数组中所有元素的大数组。但array_merge()允许在合并两个数字数组时存在重复的元素，所以还要再调用array_unique()来过滤掉重复的元素。由于array_unique()函数不压缩过滤后的数组，所以结果可能会导致数组中存在间隙（空元素位）。但这不成问题，因为foreach和each()在处理稀疏填充的数组时都不会有障碍。

计算交集的函数就叫array_intersection()，而且不用你参与任何处理。

array_diff()函数会返回一个包含在$old中，但不包含在$new中所有唯一元素的数组。这种计算的结果称为简单差集：

```
$old = array('To', 'be', 'or', 'not', 'to', 'be');
$new = array('To', 'be', 'or', 'whatever');
$difference = array_diff($old, $new);

$old = array('To', 'be', 'or', 'not', 'to', 'be');
$new = array('To', 'be', 'or', 'whatever');
$difference = array_diff($old, $new);
Array
(
    [3] => not
    [4] => to
)
```

在结果数组$difference中，包含'not'和'to'，是因为array_diff()是大小写敏感的；不包含'whatever'，是因为它不属于$old。

如果想得到一个相反的差集，换句话说，就是要得到包含在$new中，不包含在$old中的元素，只须交换这两个参数即可：

```
$old = array('To', 'be', 'or', 'not', 'to', 'be');
$new = array('To', 'be', 'or', 'whatever');
$reverse_diff = array_diff($new, $old);

$old = array('To', 'be', 'or', 'not', 'to', 'be');
$new = array('To', 'be', 'or', 'whatever');
$reverse_diff = array_diff($new, $old);
Array
(
    [3] => whatever
)
```

此时的$reverse_diff数组中只包含'whatever'。

如果想要在array_diff()的结果中应用一个函数或者进行其他过滤处理，就需要自己设计相应的差集计算算法：

```
// 实现不区分大小写的差集计算——即diff -i

$seen = array();
foreach ($new as $n) {
    $seen[strtolower($n)]++;
}

foreach ($old as $o) {
    $o = strtolower($o);
    if (!$seen[$o]) { $diff[$o] = $o; }
}
```

第一个foreach语句建立了一个关联数组查寻表格。然后，循环遍历$old，如果在查询表格中没有发现相应元素，就把该元素添加到$diff中。

如果将array_diff()和array_map()组合起来使用，速度会稍微快一点：

```
$diff = array_diff(array_map('strtolower', $old), array_map('strtolower', $new));
```

所谓对称差集是指同时满足“包含在$a但不包含在$b中”和“包含在$b但不包含在$a中”这两个条件的结果集：

```
$difference = array_merge(array_diff($a, $b), array_diff($b, $a));
```

有了定义之后，这里的算法变得非常直观。即先调用两次array_diff()得到两个相应的简单差集，然后再将这两个差集合并为一个数组。之所以不用调用array_unique()，是因为该算法已经潜在地把两个简单差集构造得毫无相同之处的了。

参见

array_unique()函数的文档（*http://www.php.net/array-unique*），array_intersect()函数的文档（*http://www.php.net/array-intersect*），array_diff()函数的文档（*http://www.php.net/array-diff*），array_merge()函数的文档（*http://www.php.net/array-merge*）和array_map函数的文档（*http://www.php.net/array-map*）。

4.24 创建一个类数组对象

问题

你有一个对象，但你希望能将其看作一个数组。这样就既可以利用面向对象设计的益处，又可以享有熟悉的数组接口的便利。

方案

实现SPL的ArrayAccess接口：

```
class FakeArray implements ArrayAccess {

    private $elements;

    public function __construct() {
        $this->elements = array();
    }

    public function offsetExists($offset) {
        return isset($this->elements[$offset]);
    }

    public function offsetGet($offset) {
        return $this->elements[$offset];
    }

    public function offsetSet($offset, $value) {
        return $this->elements[$offset] = $value;
    }

    public function offsetUnset($offset) {
        unset($this->elements[$offset]);
    }
}

$array = new FakeArray;

// What's Opera, Doc?
$array['animal'] = 'wabbit';
```

```
// 保持安静，我正在搜寻wabbits
if (isset($array['animal']) &&
    // Wabbit 现身了!!!
    $array['animal'] == 'wabbit') {

    // 干掉 wabbit，干掉 wabbit，干掉 wabbit
    unset($array['animal']);
    // Yo ho to oh! Yo ho to oh! Yo ho...
}

// 我干什么了?? 我打死了 wabbit....
// 可怜的小兔子，可怜的小 wabbit...
if (!isset($array['animal'])) {
    print "Well, what did you expect in an opera? A happy ending?\n";
}
Well, what did you expect in an opera? A happy ending?
```

讨论

ArrayAccess接口可以让你在操纵对象中的数据时，使用与操纵数组中的数据相同的一套约定。这样，我们就既能够充分利用面向对象设计的好处，例如使用基于类的继承，或者在对象中实现额外的方法；而且，还能继续使用熟悉的接口与对象进行交互。作为一种选择，我们可以创建一个把数据保存到外部位置（比如说共享内存或者数据库中）的“数组”。

要实现ArrayAccess接口，必须实现四个方法：offsetExists()，用于标识一个元素是否已定义；offsetGet()，用于返回一个元素的值；offsetSet()，用于为一个元素设置新值；offsetUnset()，用于删除一个元素和相应的值。

这个例子把数据保存到了本地的对象属性中：

```
class FakeArray implements ArrayAccess {

    private $elements;

    public function __construct() {
        $this->elements = array();
    }

    public function offsetExists($offset) {
        return isset($this->elements[$offset]);
    }

    public function offsetGet($offset) {
        return $this->elements[$offset];
    }

    public function offsetSet($offset, $value) {
        return $this->elements[$offset] = $value;
```

```
    }

    public function offsetUnset($offset) {
        unset($this->elements[$offset]);
    }
}
```

其中的对象构造函数把$elements属性初始化为一个新数组。这样就为保存数组的键和值提供了本地空间。该属性定义为私有，所以人们只能通过属于该接口的访问器方法来访问这个属性。

接下来的四个方法实现了我们操纵数组必须的动作。offsetExists()检查是否已经定义了一个数组，该方法会返回isset($this->elements[$offset])的值。

而offsetGet()和offsetSet()方法则实现了与$elements属性的交互，就和我们通常对数组进行的操作一样。

最后，offsetUnset()方法则在数组元素上简单地调用了unset()。与其他方法不同，这个方法在操作后不返回任何值。这是因为unset()是一条语句，而非一个函数，因此不会返回值。

现在，就可以技巧化一个FakeArray对象，并像操纵数组一样来操纵该技巧：

```
$array = new FakeArray;

// What's Opera, Doc?
$array['animal'] = 'wabbit';

// 保持安静，我正在搜寻wabbits
if (isset($array['animal']) &&
    // Wabbit 现身了!!!
    $array['animal'] == 'wabbit') {

    // 干掉 wabbit, 干掉 wabbit, 干掉 wabbit
    unset($array['animal']);
    // Yo ho to oh! Yo ho to oh! Yo ho...
}

// 我干什么了?? 我打死了 wabbit....
// 可怜的小兔子，可怜的小 wabbit...
if (!isset($array['animal'])) {
    print "Well, what did you expect in an opera? A happy ending?\n";
}
Well, what did you expect in an opera? A happy ending?
```

其中的每一步操作都会调用一个相应的方法：对$array['animal']赋值会触发offset-Set(),检查isset($array['animal'])会调用offsetExists()，而offsetGet()则在你比较

$array['animal'] == 'wabbit'时自动运行，最后当执行unset($array['animal'])时会调用offsetUnset()方法。

结果，我们都已经看到了，那只wabbit的确“死了”。

参见

第7章中有关对象的更多内容。ArrayAccess接口的更多参考资料（*http://www.php.net/~helly/php/ext/spl/interfaceArrayAccess.html*）和"What's Opera, Doc?"的维基条目（*http://en.wikipedia.org/wiki/What%27s_Opera%2C_Doc*）。

4.25 编程：输出水平居中的HTML表格

把一个数组转换成一个水平居中的、每行设置为固定列数的表格。第一组内容放到包含表格开标记的行中，第二组内容放到下一行中，依次类推。最终，到了最后一行时，可能需要使用若干空表元来填充该行。

例4-4中的函数pc_grid_horizontal()可以接受我们指定的数组和列数。同时，它假设生成的表格宽度为100%，不过可以通过修改变量$table_width来改变表格的宽度。

例4-4：pc_grid_horizontal()函数

```
function pc_grid_horizontal($array, $size) {

    // 计算 <td> 的宽度 (%)
    $table_width = 100;
    $width = intval($table_width / $size);

    // 定义输出的<tr>和<td>标签
    // sprintf()需要使用%%得到直接量%
    $tr = '<tr align="center">';
    $td = "<td width=\"$width%%\">%s</td>";

    // 表格的开标签
    $grid = "<table width=\"$table_width%%\">$tr";

    // 循环遍历数组中的元素，并将元素加入到 $size 列中
    // $i 用于跟踪是否需要开始新行
    $i = 0;
    foreach ($array as $e) {
        $grid .= sprintf($td, $e);
        $i++;

        // 一行结束
        // 输出闭标签，并开始新行
        if (!($i % $size)) {
            $grid .= "</tr>$tr";
```

```
        }
    }

    // 用空白符填充剩余的表元
    while ($i % $size) {
        $grid .= sprintf($td, ' ');
        $i++;
    }

    // 如果需要，加上</tr>
    $end_tr_len = strlen($tr) * -1;
    if (substr($grid, $end_tr_len) != $tr) {
        $grid .= '</tr>';
    } else {
        $grid = substr($grid, 0, $end_tr_len);
    }

    // 输出表格闭标签
    $grid .= '</table>';

    return $grid;
}
```

这个函数首先从计算每个<td>元素占整个表格宽度的百分之几开始，这取决于列数和整个表格的宽度。所有<td>元素的宽度之和可能不正好等于<table>元素的宽度，但这不会影响到我们以鲜明的风格来输出HTML表格。接着，使用printf式的格式化符号来定义<td>和<tr>标签。其中，要在定义<td>的百分比宽度时输出必要的直接量%，需要使用两个百分号 —— %%。

这个函数的精华部分是通过foreach来循环遍历数组，在循环过程中，我们把每个<td>元素追加到$grid中。当被处理过的元素的总数是一行中元素个数的整数倍时，说明到达了一行的结尾，那么关闭该行，并输出一个新行标签。

当把所有元素都添加完毕时，就需要用空白符或者说空的<td>元素来填充最后一行。这时候要在表元中放入一个不间断的空白符，而不是简单地留空，这样才能在浏览器中得到适当的解析结果。现在，要保证的是在HTML标记的末尾不会有多余的<tr>标签，这种情况当元素数目正好是表格列数的整数倍时（即，在不用空白表元填充最后一行的情况下）会出现。最后，剩下的就是输出关闭表格的标签。

例如，我们可以用一个六列的表格来输出美国50个州的名字：

```
// 建立到数据库的连接
$dsn = 'mysql://user:password@localhost/table';
$dbh = DB::connect($dsn);
if (DB::isError($dbh)) { die ($dbh->getMessage()); }
```

```
// 在数据库中查询50个州
$sql = "SELECT state FROM states";
$sth = $dbh->query($sql);

// 从数据库中把数据载入到一个数组中
while ($row = $sth->fetchRow(DB_FETCHMODE_ASSOC)) {
  $states[] = $row['state'];
}

// 生成HTML表格标记
$grid = pc_grid_horizontal($states, 6);

// 并输出
print $grid;
```

当浏览器解析完成这些HTML表格标记后，效果如图4-1所示。

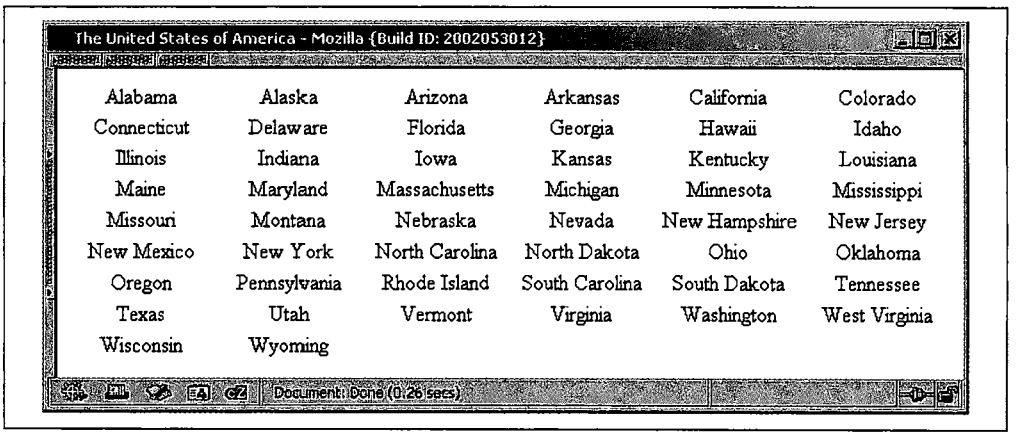

图4-1：美利坚合众国的50个州

因为50不能被6整除，所以在表格的最后一行有四个额外的填充表元。

第5章

变量

5.0 概述

伴随着条件逻辑而生的变量，是计算机程序之所以强大和灵活的核心。如果把变量比喻成一个有自己名字并保存着数值的桶，那么在PHP中，就有用于装无格式数据的旧桶、装其他桶的名字的桶、装数值和字符串的桶、装保存着其他桶的数组的桶、装满对象的桶等。对于其他各种变量，也都可以用这种比喻来形容。

变量要么是经过设置的，要么是未经过设置的。如果把一个值赋给变量，不论这个值是true还是false，是空值还是非空值，这个变量就是经过设置的。当传递给函数isset()的变量是经过设置的，那该函数会返回true。要把变量从设置的转换成未设置的，需要对该变量调用unset()函数，或者给这个变量赋null值。标量、数组和对象都可以传递给unset() 函数。也可以给unset()传递多个变量，一起销毁它们。

```
unset($vegetables);
unset($vegetables[12]);
unset($earth, $moon, $stars);
```

如果变量出现在一个URL的查询字符串中，即使它没有被指定值，也是经过设置的。因此以下URL：

```
http://www.example.com/set.php?chimps=&monkeys=12
```

会将$_GET['monkeys']设置为12，而将$_GET['chimps']设置成了空字符串。

所有未经设置的变量也是空的。设置的变量可能是空的，也可能是非空的。空的变量，是指其值可以转换成布尔值false的变量，包括：整型数0，双精度型数0.0，空字符串，字符串"0"，布尔值false，没有元素的数组，没有属性的对象（在PHP 5之前的版本中）和NULL。其他的变量都是非空的。其中包括字符串"00"和字符串" "——只包含一个空白符。

变量的值要么可以转换成true，要么可以转换成false。前面罗列出来的可以转换成false布尔值的值就是PHP中所有可以用false表示的值。其他的值就是可以用true来表示的值。空值和false之间的区别在于，只有对于变量来说才能有空值。

常量以及由函数返回的值可以是false，但不能是空值。例如，例5-1中是关于empty()函数的一个有效用法，因为$first_name是一个变量。

例5-1：正确地检查一个变量是否为空

```
if (empty($first_name)) { .. }
```

另一方面，在例5-2的代码中，会返回解析错误，这是因为0（一个常量）和get_first_name()函数的返回值不能为空。

例5-2：不正确地检查一个常量和函数是否为空

```
if (empty(0)) { .. }
if (empty(get_first_name())) { .. }
```

5.1 消除 == 和 = 的困扰

问题

你不想在比较一个变量和一个常量时意外地进行赋值。

方案

使用：

```
if (12 == $dwarves) { ... }
```

而不是

```
if ($dwarves == 12) { ... }
```

把常量放在赋值操作符的左侧会触发一个解析错误。换句话说，PHP会拒绝如下写法：

```
if (12 = $dwarves) { ... }
```

但对于：

```
if ($dwarves = 12) { ... }
```

则会默默地执行，即把12赋值给变量$dwarves，然后再执行块中的代码（$dwarves = 12的计算值是12，可以转换为true）。

讨论

把常量放在比较操作符的左侧会强制按常量的类型进行比较。当你将一个整型数和一个可能是整型数或者是一个字符串的变量进行比较时，会导致一些问题。当$dwarves为0时，0 == $dwarves返回true，但是当$dwarves是字符串sleepy时，该比较表达式仍然返回true。因为一个整型数（0）位于比较表达式的左侧时，PHP会在进行比较之前把右侧的操作数（字符串sleepy）先转换为一个整型数。为了避免此类问题发生，可以使用等同操作符，即 0 === $dwarves。

参见

=操作符的文档（*http://www.php.net/language.operators.assignment.php*）及==和===操作符的文档（*http://www.php.net/manual/language.operators.comparison.php*）

5.2 为变量设定默认值

问题

你想为一个没有值的变量指定一个默认的值。在编程中，经常会遇到为变量指定的硬编码默认值被来自表单输入字段的值或者被一个环境变量的值覆盖的情况。

方案

通过isset()对可能包含值的变量进行检测，判断是否需要为其指定默认值：

```
if (! isset($cars)) { $cars = $default_cars; }
```

使用三元（a ? b : c）操作符为一个新变量指定一个（可能是默认的）值：

```
$cars = isset($_REQUEST['cars']) ? $_REQUEST['cars'] : $default_cars;
```

讨论

在为变量指定默认值时，使用isset()是必需的。如果不使用这个函数，那么变量的非默认值就不能是0或者其他能转换为false的值。考虑以下赋值语句：

```
$cars = $_REQUEST['cars'] ? $_REQUEST['cars'] : $default_cars;
```

如果$_REQUEST['cars']是0,那么即使0对$car而言也是一个有效的值，$cars也会被指定为$default_cars。

在检测数组是否经过设置时，需要使用另外一个函数array_key_exists()：

```
$cars = array_key_exists('cars', $_REQUEST) ? $_REQUEST['cars'] : $default_cars;
```

isset()和array_key_exists()之间的一个区别是，当数组中存在一个键但对应的值为null时，array_key_exists()返回true，isset() 则返回false：

```
$vehicles = array('cars' => null);
array_key_exists('cars', $vehicles);          // true
isset($vehicles['cars']);                     // false
```

很容易使用一个默认的数组来为多个变量设置默认值。这个默认数组中的键是变量名，而数组中的值则是要为每个变量设置的默认值：

```
$defaults = array('emperors'  => array('Rudolf II','Caligula'),
                  'vegetable' => 'celery',
                  'acres'     => 15);

foreach ($defaults as $k => $v) {
    if (! isset($GLOBALS[$k])) { $GLOBALS[$k] = $v; }
}
```

由于变量是在全局名称空间中设置的，所以不能用前面的代码来设置私有函数的默认值。如果想设置私有函数的默认值，可以通过可变变量来实现：

```
foreach ($defaults as $k => $v) {
    if (! isset($$k)) { $$k = $v; }
}
```

在这个例子中，执行第一次循环时，$k是emperors，所以，$$k就是$emperors。

参见

isset()函数的文档（*http://www.php.net/isset*）。

5.3 不使用临时变量而实现变量值的交换

问题

你想在不使用额外变量的情况下，交换现有两个变量的值。

方案

要交换$a和$b：

```
list($a,$b) = array($b,$a);
```

讨论

PHP的list()语言结构可以把数组中的值分别指定给单独的变量。与其相对的、位于表达式右侧的array()，则可以把单独的变量值构造成一个数组。所以，把由array()返回的数组（元素）指定给list()中的变量，就可以实现两个值次序的交换。当然，这样对于多值的交换也是有效的：

```
list($yesterday,$today,$tomorrow) = array($today,$tomorrow,$yesterday);
```

这种方法并不如使用临时变量的速度更快，所以需要明确使用这种方法是为了前后关系清楚，而不是为了速度。

参见

list()结构的文档（*http://www.php.net/list*）和array()的文档（*http://www.php.net/array*）。

5.4 动态创建变量名

问题

你想要动态地创建一个变量名。例如，你想要使用与数据库查询得到的字段名称一致的变量名。

方案

使用PHP中“可变变量”的语法，即在一个其值为你想作为变量名称的变量前面加一个$：

```
$animal = 'turtles';
$turtles = 103;
print $$animal;

103
```

讨论

如果在一个变量名称前面放两个美元（$）符号，PHP就会废弃右面的变量，而取其值，然后，将该值作为“真正”的变量名称。例如：

```
$animal = 'turtles';
$turtles = 103;
print $$animal;

103
```

这段代码会输出103。因为$animal = 'turtles',$$animal 就是$turtles,而$turtles等于103。

如果使用大括号，还能够构造出用于表示变量名的更复杂的表达式：

```
$stooges = array('Moe','Larry','Curly');
$stooge_moe = 'Moses Horwitz';
$stooge_larry = 'Louis Feinberg';
$stooge_curly = 'Jerome Horwitz';

foreach ($stooges as $s) {
  print "$s's real name was ${'stooge_'.strtolower($s)}.\n";
}
Moe's real name was Moses Horwitz.
Larry's real name was Louis Feinberg.
Curly's real name was Jerome Horwitz.
```

PHP会求得位于大括号之间的表达式的值，并将这个值作为一个变量名。大括号中的表达式甚至还可以调用函数，比如上面代码中的strtolower()。

在迭代类似的变量名称时，可变变量也很有用。假如你想要查询一个数据库表，其字段名分别是title_1，title_2等。如果你想要检查一个标题（$title）是否与其中一个字段的值匹配，那么最简单的方式就是像下面这样进行循环遍历：

```
for ($i = 1; $i <= $n; $i++) {
    $t = "title_$i";
    if ($title == $$t) { /* 匹配 */ }
}
```

当然，如果把这些值保存到一个数组中可能会更直观，不过，要是你在维护一段使用了这种技术的老代码（即不能修改），可变变量就很有用了。

大括号语法在厘清有关数组元素的歧义时是必要的。可变变量$$donkeys[12]可能有两种含义。第一种含义是“拿$donkeys数组中第12元素的值作为一个变量名”，要想明确表示这种含义，应该写成：${$donkeys[12]}。第二种含义是“用标量$donkeys的值作为

一个数组名称，并找到该数组中的第12元素”，若想明确表示这种含义，应该写成：${$donkeys}[12]。

可变变量语法并不局限于两个美元符号，可以使用三个甚至更多个。然而，在实践中很少出现有必要使用超过两个间隔层次的情况。

参见

可变变量的文档（*http://www.php.net/language.variables.variable*）。

5.5 使用静态变量

问题

你想让一个本地变量在两个函数的调用之间保留自己的值。

方案

把这个本地变量声明为静态变量：

```
function track_times_called() {
    static $i = 0;
    $i++;
    return $i;
}
```

讨论

把一个变量声明为静态变量会使得函数记住这个变量的值。所以，如果在后面调用该函数，那么就可以访问到这个被保存的变量的值。例5-3中的pc_check_the_count()函数，用静态变量保持了对一个棒球击球手所击的好球数（strikes）和坏球数（balls）的跟踪。

例5-3：pc_check_the_count()函数

```
<?php
function pc_check_the_count($pitch) {
    static $strikes = 0;
    static $balls   = 0;

    switch ($pitch) {
    case 'foul':
        if (2 == $strikes) break; // nothing happens if 2 strikes
```

```
        // 否则，则视同好球
    case 'strike':
        $strikes++;
        break;
    case 'ball':
        $balls++;
        break;
    }

    if (3 == $strikes) {
        $strikes = $balls = 0;
        return 'strike out';
    }
    if (4 == $balls) {
        $strikes = $balls = 0;
        return 'walk';
    }
    return 'at bat';
}

$what_happened = pc_check_the_count($pitch);
?>
```

在pc_check_the_count()函数中，击球手根据投球次数发挥的逻辑在函数中以switch语句来表示。当然，也可以返回好球数和坏球数，但这需要在代码中的多处位置上分别检查垒中的出局（striking out）、走步（walking）和停留（staying）。

虽然静态变量会在函数调用之间保留它们的值，但也只是在一个脚本程序中才有效。在一次请求中访问的页面中的静态变量，不会将值保留到对同一页面的下一次请求。

参见

静态变量的文档（*http://www.php.net/language.variables.scope*）。

5.6 在进程间共享变量

问题

你需要一种在进程间共享信息的途径，并保证能快速地访问这些共享数据。

方案

使用shmop或System V这两种共享内存扩展中的一种。

若使用shmop，可以像例5-4那样创建一个内存块，从中读取并向其中写入信息。

例 5-4：使用shmop共享内存功能

```
<?php
// 创建键
$shmop_key = ftok(__FILE__, 'p');
// 创建16384 字节的共享内存块
$shmop_id = shmop_open($shmop_key, "c", 0600, 16384);
// 取得全部共享内存片段中数据
$population = shmop_read($shmop_id, 0, 0);
// 生成数据
$population += ($births + $immigrants - $deaths - $emigrants);
// 将生成的数据回写入共享内存片段
$shmop_bytes_written = shmop_write($shmop_id, $population, 0);
// 检查回写长度是否符合
if ($shmop_bytes_written != strlen($population)) {
   echo "Can't write the all of: $population\n";
}
// 关闭句柄
shmop_close($shmop_id);
?>
```

对于System V共享内存，也是把数据存储在共享内存片段中，并通过信号机制（semaphore）来保证对共享内存片段的排他性访问，具体过程见例5-5。

例5-5：使用System V共享内存功能

```
<?php
$semaphore_id = 100;
$segment_id   = 200;
// 获取一个与我们想取得的共享内存关联的信号的句柄
$sem = sem_get($semaphore_id,1,0600);
// 确保对信号的独占访问
sem_acquire($sem) or die("Can't acquire semaphore");
// 获取共享内存片段的句柄
$shm = shm_attach($segment_id,16384,0600);
// 从共享内存片段中取回值
$population = shm_get_var($shm,'population');
// 生成值
$population += ($births + $immigrants - $deaths - $emigrants);
// 把生成的值存回写到共享内存片段中
shm_put_var($shm,'population',$population);
// 释放对共享内存的句柄
shm_detach($shm);
// 释放信号，以便其他进程可以捕获它
sem_release($sem);
?>
```

讨论

所谓共享内存片段，就是指你电脑的内存中可以供不同进程（例如处理请求的多个Web服务器进程）访问的一段内存。这两个扩展都能以快速而有效的方式解决在不同的进程

间保存信息的问题，但由于这两种手段的实现机制稍有不同，因而其接口也存在一定的差异。

其中，shmop功能具有与我们熟悉的文件操作类似的接口。可以打开一个内存片段，从中读取数据，向其中写入数据，并关闭该内存片段。与文件操作相同的是，其中没有内置的数据分段，全部内容只是一系列连续的字符而已。

在例5-4中，我们首先创建了一个共享内存块。这里不同于对文件的操作，必须预先声明该内存块的最大容量。在该例子中，即16384字节：

```
// 创建键
$shmop_key = ftok(__FILE__, 'p');
// 创建16384 字节的共享内存块
$shmop_id = shmop_open($shmop_key, "c", 0600, 16384);
```

同我们使用文件名来区分文件一样，shmop内存片段以键来区分。但与文件使用字符串命名不同，这些键是用整型数来表示的，所以不太容易记忆。因此，最好是用`ftok()`函数来把与人类友好的名称 —— 这里使用的是类似文件名形式的`__FILE__` —— 转换成`shmop_open()`能够接受的适当格式。这个`ftok()`函数也可以接受一个one-fscharacter参数，即“项目标识符（project identifier）”（对于PHP而言，是这里的p），用于在万一用到相同的字符串时避免发生冲突。

当创建了一个键后，就可以把这个键连同需要的“标志（flag）”、操作权限（八进制）以及块的大小传递给`shmop_open()`。可以在表5-1中查到适当的标志列表。

其中的操作权限类似于对文件的操作权限，而0600的含义就是创建这个内存块的用户可以从中读取或向其中写入数据。在这个环境中，所谓用户并不仅意味着创建该信号的进程，也包含具有相同用户ID的其他进程。也就是说，0600权限可以供很多以同一个用户身份运行的Web服务器进程使用。

表5-1：shmop_open()标志

标志	描述
a	以只读权限打开
c	创建新片段，如果已经存在，则以可读写权限打开
w	以可读写权限打开
n	创建新片段，如果已经存在则失败。对于避免竞态条件有用

当我们取得了一个句柄之后，就可以使用`shmop_read()`从相应的内存片段中读取数据，并对读出的数据进行操作了：

```
// 取得全部共享内存片段中数据
$population = shmop_read($shmop_id, 0, 0);
// 生成数据
$population += ($births + $immigrants - $deaths - $emigrants);
```

上面的代码读取的是整个内存片段。如果想读取更少的部分，可以调整第二和第三个参数。第二个参数表示开始读取的位置，第三个参数表示要读取数据的长度。也可以使用简写方式，即把长度设置为0，表示读到该内存的结尾处。

在对数据重新调整后，就可以通过shmop_write()把这些数据再存储回共享内存中，然后调用shmop_close()来释放句柄：

```
// 把生成的值存回写到共享内存片段中
$shmop_bytes_written = shmop_write($shmop_id, $population, 0);
// 检查回写长度是否符合
if ($shmop_bytes_written != strlen($population)) {
   echo "Can't write the all of: $population\n";
}
// 关闭句柄
shmop_close($shmop_id);
```

因为共享内存片段的大小是固定的，如果不小心很可能会尝试向其中写入超出容量限制的数据。可以通过比较shmop_write()的返回值与写入数据的字符串长度，来检测出是否存在上述问题。如果shmop_wirte()返回的值更小一些，就表示共享内存中的空间都已用完 —— 只能存储那么多字节的数据了。

与shmop相对应，System V 共享内存功能的行为类似一个数组。我们通过指定一个键来访问内存片段中的数据，例如population，并且可以直接操纵这些数据。如果你存储的内容适当，这种直接地访问可能会很方便。

然而，这种接口因此也会更加复杂一些。而且， System V 共享内存还需要你自己以信号的方式来控制对内存访问的锁定。

通过一个信号就可以确保不同的进程在访问相同的共享内存片段时不会相互干扰。在一个进程能够使用这个信号之前，它首先需要取得对该信号的控制权。而当这个进程使用完该信号后，它应该释放该信号以便其他进程使用。

要获得对一个信号的控制权，需要使用sem_get()来取得相应信号的ID。sem_get()接受的第一个参数是一个以整型数表示的信号键。可以把这个键设定为任何你认为合适的整型数，只要所有需要访问这一特定信号的程序都使用该键就没有问题。如果不存在指定的信号键，那么就会创建该信号。进程可以访问的这个信号的最大数值由sem_get()的第二个参数（这里的1）决定，而对这个信号的权限则通过sem_get()的第三个参数设置。例如：

```
$semaphore_id = 100;
$segment_id   = 200;
// 获取一个与我们想取得的共享内存关联的信号的句柄
$sem = sem_get($semaphore_id,1,0600);
// 确保对信号的独占访问
sem_acquire($sem) or die("Can't acquire semaphore");
```

sem_get()会返回一个指向潜在的系统信号的标识符。通过这个ID可以使用sem_acquire()函数来得到对相应信号的控制权。这个函数等到信号可以取得时（可能是等到其他进程释放这个信号）会取得信号并返回true。如果发生了错误则返回false。可能的错误包括无效的权限或没有足够的内存创建信号。一旦得到了信号，就可以从相应的共享内存片段中读取数据了：

```
// 获取共享内存片段的句柄
$shm = shm_attach($segment_id,16384,0600);
// 从共享内存片段中取回值
$population = shm_get_var($shm,'population');
// 生成值
$population += ($births + $immigrants - $deaths - $emigrants);
```

首先，用shm_attach()来建立一个指向特定共享内存片段的连接。与sem_get()相同，shm_attach()的第一个参数也是以整型数表示的信号键。然而，这时该信号键表示的是目标内存片段，而不再是信号。如果指定的键所表示的内存片段并不存在，那么由另一参数来创建它。第二个参数（16384）是以字节表示的内存片段的容量，而最后一个参数（0600）表示对这个片段的访问权限。shm_attach(200，16384，0600)会创建一个16K的共享内存片段，但只有创建该片段的用户才能对其进行读写操作。这个函数会返回一个表示可以进行读写的共享内存片段的标识符。

当加了内存片段之后，就可以通过shm_get_var($shm，'population')来取得相应变量的值。这个函数搜索以$shm标识的共享内存片段，并取得名为population变量的值。可以在共享内存中存储任何类型的变量。当取得了该变量后，就可以和操纵其他变量一样来操纵这个变量。接着，shm_put_var($shm,'population',$population)把$population的值作为一个名为population的变量的值写入到共享内存片段中。

至此，对共享内存的操作已经结束了。因此，需要用shm_detach()来断开与内存片段的连接，并用sem_release()来释放相应的信号，以便其他进程能够使用这个信号：

```
// 释放对共享内存的句柄
shm_detach($shm);
// 释放信号，以便其他进程可以捕获它
sem_release($sem);
```

共享内存的最大好处是它的速度非常快。但是，由于它是在RAM中存储，所以不可能保

存太多数据。而且，当计算机重新启动时保存的数据就会丢失（除非在关机之前采取特殊的措施把共享内存中的数据写入到硬盘，并在启动后再将这些数据载入到内存中）。

在Windows平台上虽然不能使用 System V 共享内存，但可以正常使用shmop功能。除了这两种扩展以外，还有一种可能是使用APC扩展，该扩展除了支持PHP缓存和优化的功能外，也提供了一种存储数据的方式。

参见

APC的有关资料（*http://pecl.php.net/apc*）。shmop的文档（*http://www.php.net/shmop*）和System V共享内存以及semaphore特性的资料（*http://www.php.net/sem*）。

5.7 把复杂的数据类型压缩到一个字符串中

问题

你想用一个字符串来表示一个数组或者对象，以便能将其保存到数据库中。这个字符串应该很容易恢复为原始的数组或者对象。

方案

使用serialize()把变量和它们的值编码成文本形式：

```
$pantry = array('sugar' => '2 lbs.','butter' => '3 sticks');
$fp = fopen('/tmp/pantry','w') or die ("Can't open pantry");
fputs($fp,serialize($pantry));
fclose($fp);
```

要想恢复这些变量，使用unserialize()：

```
$new_pantry = unserialize(file_get_contents('/tmp/pantry'));
```

讨论

经过序列化之后的可恢复字符串保存到$pantry中，其值为：

```
a:2:{s:5:"sugar";s:6:"2 lbs.";s:6:"butter";s:8:"3 sticks";}
```

这些字符保存了将来恢复数组中所有值的足够信息，不过在这个序列化的字符串中不包括变量的名称。

当把这些序列化的数据放在URL中在页面之间传递时，需要对这些数据调用urlencode()，以确保对其中的URL元字符进行处理：

```
$shopping_cart = array('Poppy Seed Bagel' => 2,
                       'Plain Bagel' => 1,
                       'Lox' => 4);
print '<a href="next.php?cart='.urlencode(serialize($shopping_cart)).'">Next</a>';
```

magic_quotes_gpc和magic_quotes_runtime配置项的设置会影响传递到unserialize()中的数据。如果magic_quotes_gpc项是启用的，那么在URL、POST变量以及cookies中传递的数据在反序列化之前必须用stripslashes()进行处理：

```
$new_cart = unserialize(stripslashes($cart));  // 如果magic_quotes_gpc开启
$new_cart = unserialize($cart);                // 如果magic_quotes_gpc关闭
```

如果magic_quotes_runtime是启用的，那么在向文件中写入序列化的数据之前必须用addslashes()进行处理，而在读取它们之前则必须用stripslashes()进行处理：

```
$fp = fopen('/tmp/cart','w');
fputs($fp,addslashes(serialize($a)));
fclose($fp);

// 如果magic_quotes_runtime开启
$new_cart = unserialize(stripslashes(file_get_contents('/tmp/cart')));
// 如果magic_quotes_runtime关闭
$new_cart = unserialize(file_get_contents('/tmp/cart'));
```

在启用了magic_quotes_runtime的情况下，从数据库中读取序列化的数据也必须经过stripslashes()的处理：

```
mysql_query(
    "INSERT INTO cart (id,data) VALUES (1,'".addslashes(serialize($cart))."')");

$r = mysql_query('SELECT data FROM cart WHERE id = 1');
$ob = mysql_fetch_object($r);
// 如果magic_quotes_runtime开启
$new_cart = unserialize(stripslashes($ob->data));
// 如果magic_quotes_runtime关闭
$new_cart = unserialize($ob->data);
```

保存到数据库中的序列化数据必须要经过addslashes()的处理（或者通过适合数据库的转义方法进行处理会更好），以便能够适当地存储。

当对一个对象进行反序列化操作时，PHP会自动地调用其__wakeUp()方法。这样就使得对象能够重新建立起序列化时未能保留的各种状态，例如：数据库连接等。因为可能会影响到整个代码执行环境，所以在进行反序列化时必须保持足够的清醒。有关此问题的更详细内容，请参考技巧7.18。

参见

技巧10.9中为数据库存储而转义数据的内容。

5.8 将变量内容转存为字符串

问题

你想检查保存在一个变量中的值。这个变量可能是一个复杂的嵌套数组或者对象，所以不可能直接输出，或者不能对其进行循环遍历。

方案

使用print_r()或var_dump()：

```
$array = array("name" => "frank", 12, array(3, 4));

print_r($array);
Array
(
    [name] => frank
    [0] => 12
    [1] => Array
        (
            [0] => 3
            [1] => 4
        )
)
var_dump($array);
array(3) {
  ["name"]=>
  string(5) "frank"
  [0]=>
  int(12)
  [1]=>
  array(2) {
    [0]=>
    int(3)
    [1]=>
    int(4)
  }
}
```

讨论

print_r()的输出更简洁也适宜阅读。而var_dump()的输出中则额外给出了变量的数据类型以及长度信息。

因为这两个函数都是以递归的方式循环变量，所以如果在一个变量中包含了指向该变量自身的引用，那么就可能导致死循环。但这两个函数都能够自动终止无休止地输出变量信息的行为。如果print_r()再次遇到同一个变量，它会输出*RECURSION* —— 而不是再次输出该变量的信息，然后再继续迭代输出它还没有输出的其他信息。var_dump()则会在输出两次重复变量的信息后再输出*RECURSION*，然后跳过该变量。考虑下面的两个数组$user_1和$user_2，它们通过各自的friend元素相互引用：

```
$user_1 = array('name' => 'Max Bialystock',
                'username' => 'max');

$user_2 = array('name' => 'Leo Bloom',
                'username' => 'leo');

// Max和Leo是朋友
$user_2['friend'] = &$user_1;
$user_1['friend'] = &$user_2;

// Max和Leo都有工作
$user_1['job'] = 'Swindler';
$user_2['job'] = 'Accountant';
print_r($user_2)的输出结果为：
Array
(
    [name] => Leo Bloom
    [username] => leo
    [friend] => Array
        (
            [name] => Max Bialystock
            [username] => max
            [friend] => Array
                (
                    [name] => Leo Bloom
                    [username] => leo
                    [friend] => Array
 *RECURSION*
                    [job] => Accountant
                )

            [job] => Swindler
        )

    [job] => Accountant
)
```

当print_r()第二次碰到对$user_1的引用时，它没有再次深入到该数组中去，而是输出*RECURSION*。之后，它又继续输出了$user_1和$user_2中其余变量的信息。

在面临递归时，var_dump()采取了不同的处理方式：

```
array(4) {
```

```
    ["name"]=>
    string(9) "Leo Bloom"
    ["username"]=>
    string(3) "leo"
    ["friend"]=>
    &array(4) {
      ["name"]=>
      string(14) "Max Bialystock"
      ["username"]=>
      string(3) "max"
      ["friend"]=>
      &array(4) {
        ["name"]=>
        string(9) "Leo Bloom"
        ["username"]=>
        string(3) "leo"
        ["friend"]=>
        &array(4) {
          ["name"]=>
          string(14) "Max Bialystock"
          ["username"]=>
          string(3) "max"
          ["friend"]=>
          &array(4) {
            ["name"]=>
            string(9) "Leo Bloom"
            ["username"]=>
            string(3) "leo"
            ["friend"]=>
            *RECURSION*
            ["job"]=>
            string(10) "Accountant"
          }
          ["job"]=>
          string(8) "Swindler"
        }
        ["job"]=>
        string(10) "Accountant"
      }
      ["job"]=>
      string(8) "Swindler"
    }
    ["job"]=>
    string(10) "Accountant"
  }
```

var_dump()是在第三次碰到对$user_1的引用时才终止递归。

尽管print_r()和var_dump()会输出变量的结果，而不是返回这些变量，但仍然可以通过相应的方式来捕获到它们输出的数据。

第一种，可以将true作为第二个参数传递给print_r()：

```
$output = print_r($user, true);
```

这种方式对var_dump()无效。不过，可以使用输出缓冲（output buffering）的方式：

```
ob_start();
var_dump($user);
$dump = ob_get_contents();
ob_end_clean();
```

这样就可以把var_dump($user)的结果保存到$dump中了。

参见

技巧8.12中有关输出缓冲的讨论。print_r()的文档（*http://www.php.net/print-r*）和var_dump()的文档（*http://www.php.net/var-dump*）。

第6章

函数

6.0 概述

函数可以帮助我们创建有组织的、可重用的代码。通过函数对细节的抽象，可以使代码更具有灵活性，也更容易看懂。如果不使用函数，就不可能写出容易维护的程序，因为我们需要不断对分布于多个文件中多处不同位置的相同代码块进行更新。

可以向函数中传递一些参数，并从函数得到返回值：

```
// 两个数相加
function add($a, $b) {
    return $a + $b;
}

$total = add(2, 2);    // 4
```

声明一个函数时要使用function关键字，后跟函数名以及用括号括住的参数。调用一个函数时，只需写出函数名，并为这个函数的所有参数指定相应的值。如果函数有返回值，可以如前面例子所示的那样，把函数的返回值指定给一个变量。

被调用的函数不一定要在调用位置之前声明，因为PHP是在解析完整个文件后才开始执行，所以声明函数和调用函数的位置没有先后之分。不过，在PHP中不能重复定义同一个函数。如果PHP解析到一个与已经存在的函数同名的函数，它就会抛出一个致命错误并终止运行。

有时候，传递固定个数的参数并返回一个值这样的标准过程并不一定适合编码的需要，因为事先可能无法知晓函数实际上需要接受几个参数。也可能事先知道参数的个数，但这些参数几乎每次都使用相同的值，所以不断地重复地传递这些参数会令人生厌。还有可能我们希望函数能够返回多个值。

本章可以帮助你用PHP来解决类似的问题。我们会在一开始详细介绍向函数传递参数的几种不同方式。技巧6.1～6.5中包括了传递变量的值、引用以及将变量作为命名参数传递、指定默认的参数值和创建可接受不定参数个数的函数等内容。

在技巧6.5之后的四个技巧介绍的都是与函数返回值有关的内容。其中，技巧6.6阐释了返回引用的方法，技巧6.7介绍的是返回多个值的方法，技巧6.8演示了跳过选定的返回值的过程，而技巧6.9则讨论了检测并返回函数失效的最佳方式。最后三个技巧分别介绍了如何调用可变函数、处理变量的作用域问题和动态创建函数的内容。如果想使一个变量在两个函数调用期间维持它的值，请参考技巧5.5中的相关内容。

6.1 访问函数的参数

问题

你想要访问传递给函数的参数的值。

方案

使用函数原型中的参数名：

```
function commercial_sponsorship($letter, $number) {
    print "This episode of Sesame Street is brought to you by ";
    print "the letter $letter and number $number.\n";
}

commercial_sponsorship('G', 3);
commercial_sponsorship($another_letter, $another_number);
```

讨论

在函数内部，无论传入的值是字符串、数字、数组，还是其他类型的变量，都可以一视同仁地使用原型中定义的参数来引用它们的值。

除非另有指定，否则所有被传入函数或由函数返回的非对象变量所传递的都是变量的值，而不是对变量的引用（在默认情况下，传递对象是传递引用）。这意味着，PHP会复制相应的值，并提供对该副本的访问和操作。因此，对这个副本的任何改动都不会影响原先变量中保存的值。请看下面的例子：

```
function add_one($number) {
    $number++;
}
```

```
$number = 1;
add_one($number);
print "$number\n";
1
```


如果传递的是变量的引用，那么$number的值就会是2了。

在多数语言当中，传递变量引用都能够获到比传递变量值明显速度更快的额外好处。虽然在PHP中传递引用的速度也会更快一些，但与传递值相比也所差无几。为此，我们建议只在确实必要的情况下传递变量的引用，并且任何时候也不要将其作为提升性能的一种诀窍来运用。

参见

技巧6.3中传递变量引用和技巧6.6中返回引用值的有关内容。

6.2 为函数的参数设定默认值

问题

你希望当函数调用者没有为函数传递必要的参数时，为该参数指定默认的值。例如，一个绘制表格的函数可能需要通过一个参数来指定边框的宽度，而如果未传递该参数则把其默认值设置为1。

方案

在函数原型内部为参数指定默认值：

```
function wrap_html_tag($string, $tag = 'b') {
    return "<$tag>$string</$tag>";
}
```

讨论

本方案中把标签的默认值设置成了b，可以实现加粗的效果。例如：

```
$string = 'I am some HTML';
wrap_html_tag($string);
```

会返回：

```
<b>I am some HTML</b>
```

而下面的例子：

```
wrap_html_tag($string, 'i');
```

则会返回：

```
<i>I am some HTML</i>
```

在指定默认值时有两件事值得特别关注。第一，所有指定了默认值的参数必须出现在未指定默认值的参数后面。否则，PHP无法判定省略的哪个参数应该取得默认值，以及哪个变量需要覆盖默认值。所以，wrap_html_tag()不能定义成：

```
function wrap_html_tag($tag = 'i', $string)
```

如果采取以上形式定义并给wrap_html_tag()只传递一个参数，PHP会将这个值指定给$tag，并发出缺少第二个参数的警告抗议。

第二，指定的值必须是常量，例如字符串或者数字，而不能是变量。同样地，以wrap_html_tag()为例，不能像下面这样：

```
$my_favorite_html_tag = 'i';
function wrap_html_tag($string, $tag = $my_favorite_html_tag) {
    ...
}
```

如果想以空值作为默认值，一种选择就是为参数指定一个空字符串：

```
function wrap_html_tag($string, $tag = '') {
    if (empty($tag)) return $string;
    return "<$tag>$string</$tag>";
}
```

该函数会在未传递$tab参数时返回原始的字符串。或者，当传递了$tab参数（非空）时，会返回以相应标记包围的字符串。

取决于具体的情况，也可以用0或者NULL作为$tag的默认值。虽然在wrap_html_tag()函数的例子中，我们并不想以一个空值作为标记。然而，在其他一些情况下，空字符串也可能是一个适当的选择。例如，为了把文件内容保存到一个字符串中，我们在调用file()函数后，经常会以一个空字符串来调用join()函数。同样地，正如下面代码所显示的，如果除了传递空字符串作为空信息之外，就可以认为没有提供参数而使用默认的信息：

```
function pc_log_db_error($message = NULL) {
    if (is_null($message)) {
        $message = 'Couldn't connect to DB';
    }
```

```
    error_log("[DB] [$message]");
}
```

参见

技巧6.5中创建可以接受个数可变的参数的函数的讨论。

6.3 传递引用

问题

你想把一个变量传递给一个函数，并希望保留在函数内部对该变量值的修改。

方案

指令函数接受一个传递引用而不是传递值的变量作为参数，即在函数原型中该参数的前面加上一个&：

```
function wrap_html_tag(&$string, $tag = 'b') {
    $string = "<$tag>$string</$tag>";
}
```

这样，也没有必要返回字符串了，因为在函数中修改的就是其原始的值。

讨论

通过给函数传递变量的引用，省去了返回变量值并指定给原始变量的步骤。当需要一个函数返回true或false的布尔值并仍然希望通过函数来修改变量值时，就要使用传递引用的方式。

传递变量的引用和传递变量的值是不能混用的，要么传引用，要么传值，非此即彼。换句话说，谁也没有办法让PHP随意地将一个变量一会儿看成是引用，而一会儿又看成是值。

同样，如果将参数声明为接受变量的引用，就不能再给这个参数传递一个常量字符串（或数字）值。否则，PHP会因为致命的错误而停止执行。

参见

技巧6.6中从函数中返回引用的内容。

6.4 使用命名的参数

问题

你想通过名字来为一个函数指定参数，而不是受限于调用函数时参数的位置。

方案

让函数只接受一个参数，但该参数是一个关联数组：

```
function image($img) {
    $tag  = '<img src="' . $img['src'] . '" ';
    $tag .= 'alt="' . ($img['alt'] ? $img['alt'] : '') .'">';
    return $tag;
}

$image = image(array('src' => 'cow.png', 'alt' => 'cows say moo'));
$image = image(array('src' => 'pig.jpeg'));
```

讨论

虽然使用命名的参数会使函数中的代码变得更复杂，但却能够使调用代码更简单易读。因为函数只需定义一次，但却会被多次调用，所以这样做有助于让代码更容易看懂。

在使用这种技术时，PHP不会因为你偶然拼错了一个参数的名称而抗议，所以必须小心谨慎，否则解析器根本捕获不到这种错误。不过，这样也就不能利用PHP的特性来为某些参数指定默认值了。幸运的是，可以在函数的顶部添加一些简单的代码来弥补这一不足：

```
function image($img) {
    if (! isset($img['src']))    { $img['src']    = 'cow.png';      }
    if (! isset($img['alt']))    { $img['alt']    = 'milk factory'; }
    if (! isset($img['height'])) { $img['height'] = 100;            }
    if (! isset($img['width']))  { $img['width']  = 50;             }
    ...
}
```

代码使用isset()函数来检查是否为每个参数都设置了值，如果没有，则为该参数指定默认值。

作为一种替换方案，还可以用更少的代码来实现这一点：

```
function pc_assign_defaults($array, $defaults) {
    $a = array();
```

```
    foreach ($defaults as $d => $v) {
        $a[$d] = isset($array[$d]) ? $array[$d] : $v;
    }

    return $a;
}
```

这个函数循环遍历一个默认数组中所有的键，并检查给定的数组$array中是否为对应的键设置了值。如果没有，函数就使用$defaults数组中相应键的值作为默认值。将这个函数用于前面的代码片断后，可以取代函数顶部的那些代码行：

```
function image($img) {
    $defaults = array('src'    => 'cow.png',
                      'alt'    => 'milk factory',
                      'height' => 100,
                      'width'  => 50
                     );
    $img = pc_assign_defaults($img, $defaults);
    ...
}
```

之所以这样更好，是因为它给代码带来了更大的灵活性。如果想要修改指定默认值的方式，那么只要修改pc_assign_defaults()函数就可以了，而不用再逐一地修改多个函数中的几百行代码。同时，使用名/值对形式的数组和一行代码来指定默认值，比在一连串内容几乎相同的代码行中混杂两种意图的方式更清楚。

参见

技巧6.5中创建可以接受个数可变的参数的函数的讨论。

6.5 创建可以接受个数可变的参数的函数

问题

你想要定义一个可以接受个数可变的参数的函数。

方案

向函数传递一个数组，并把要传递的参数变量作为数组的元素：

```
// 计算一组数字的"平均数"
function mean($numbers) {
    // 初始化以消除警告
    $sum = 0;
```

```
    // 数组中元素的数量
    $size = count($numbers);

    // 迭代遍历数组并加总每个元素的值
    for ($i = 0; $i < $size; $i++) {
        $sum += $numbers[$i];
    }

    // 用元素数除总和
    $average = $sum / $size;

    // 返回平均值
    return $average;
}

$mean = mean(array(96, 93, 97));
```

讨论

取决于你的编码风格和偏好，这个问题可能有两种不错的解决方案。更传统一些的PHP方法就是上面方案中所说明的。我们之所以提出这个方法，是因为在PHP中数组的使用非常频繁，因此，所有程序设计人员都很熟悉数组及其特性。

所以，尽管使用数组会有一些额外的开销，但对于绑定变量来说却是小事一桩。数组也在技巧6.4中用于创建命名的参数，在技巧6.7中用于保存从一个函数中返回的多个值。同样，在本技巧的函数中，用于访问和操纵数组的语法也只是涉及到了诸如`$array[$i]`和`count($array)`之类的基本指令。

但是，这种方法看起来有点缺少新意，所以PHP还提供了一个替代方案，并且允许你直接访问参数列表，请看例6-1。

例6-1：不通过参数列表而访问函数的参数

```
// 计算一组数字的"平均数"
function mean() {
    // 初始化以消除警告
    $sum = 0;

    // 传递到函数的参数个数
    $size = func_num_args();

    // 迭代遍历所有参数并加总它们的值
    for ($i = 0; $i < $size; $i++) {
        $sum += func_get_arg($i);
    }

    // 用个数来除总和
    $average = $sum / $size;
```

```
    // 返回平均值
    return $average;
}

$mean = mean(96, 93, 97);
```

这个例子使用了一组根据在调用时传递的参数个数而返回数据的函数。首先，func_num_args() 返回了一个表示传递到调用函数中（此处的mean()）参数个数的整数；此后，就可以调用func_get_arg()来取得每个特定位置上参数的值。

如果调用语句是mean(96, 93, 97)，func_num_args()返回3。其中，第一个参数的位置是0，所以我们迭代从0到2，而不是从1到3。也就是说，在for循环中$i从0开始自增，除非不再小于$size。我们也看到了，这和例6-1传递数组参数时的逻辑关系是相同的。如果你担心在循环中使用func_get_arg()会存在潜在的额外开销的话，还是不必了 —— 因为这个方法实际上比传递数组的方法速度更快。

这个函数还有第三个版本，即使用func_get_args()返回一个包含所有传递给函数的参数的数组。在例6-2中所示的这个版本，看起来其实是混合了前面介绍的两个方法。

例6-2：不通过参数列表而访问函数的参数

```
// 计算一组数字的"平均数"
function mean() {
    // 初始化以消除警告
    $sum = 0;

    // 把参数载入到 $numbers中
    $numbers = func_get_args();

    // 数组中元素的数量
    $size = count($numbers);

    // 迭代遍历数组并加总每个元素的值
    for ($i = 0; $i < $size; $i++) {
        $sum += $numbers[$i];
    }

    // 用元素数除总和
    $average = $sum / $size;

    // 返回平均值
    return $average;
}

$mean = mean(96, 93, 97);
```

使用这个方法，既不用在把数值传递给函数之前先放到一个临时数组中，也能够在函数中使用操纵数组的方法。遗憾的是，这个方法的速度与前两个方法相比要慢一些。

参见

技巧6.7中从一个函数中返回多个值的内容；func_num_args()函数的文档（*http://www.php.net/func-num-args*），func_get_arg()函数的文档（*http://www.php.net/func-get-arg*）和func_get_args()函数的文档（*http://www.php.net/func-get-args*）。

6.6 返回变量的引用

问题

你想返回变量的引用，而非变量的值。这样，就可以减少一个变量的副本。

方案

返回变量引用的语法与传递变量引用的语法都使用&。不过，返回变量引用不是把&放在参数前面，而是把它放在函数名的前面：

```
function &pc_array_find_value($needle, &$haystack) {
    foreach ($haystack as $key => $value) {
        if ($needle == $value) {
            return $haystack[$key];
        }
    }
}
```

同样地，当调用这个函数时，也必须使用=&赋值操作符，而不是纯=操作符：

```
$html =& pc_array_find_value('The Doors', $artists);
```

讨论

从函数返回变量的引用后，在操作该引用时就等于是在操作该变量的原始值。

举例来说，用例6-3就可以在一个数组中搜索并返回与特定值匹配的第一个元素的引用值。比如说，你要在明尼苏达州的名人录中搜索prince，以便更新他的名字。

例6-3：由函数返回一个数组值的引用

```
function &pc_array_find_value($needle, &$haystack) {
    foreach ($haystack as $key => $value) {
        if ($needle == $value) {
            return $haystack[$key];
        }
```

```
    }
}

$minnesota = array('Bob Dylan', 'F. Scott Fitzgerald', 'Prince', 'Charles Schultz');

$prince =& pc_array_find_value('Prince', $minnesota);

$prince = 'O(+>'; // 该巨头名字中不易发音符号的ASCII码

print_r($minnesota);
Array
(
    [0] => Bob Dylan
    [1] => F. Scott Fitzgerald
    [2] => O(+>
    [3] => Charles Schultz
)
```

如果不是返回对数组元素的引用，那么就只能是先返回该数组的键，然后再重新引用原始数组：

```
function pc_array_find_value($needle, &$haystack) {
    foreach ($haystack as $key => $value) {
        if ($needle == $value) {
            return $key;
        }
    }
}

$minnesota = array('Bob Dylan', 'F. Scott Fitzgerald', 'Prince', 'Charles Schultz');

$prince =& pc_array_find_value('Prince', $minnesota);

$minnesota[$prince] = 'O(+>';                    // 该巨头名字中不易发音的符号的ASCII码
```

当函数返回引用时，必须返回对变量的引用，而不能返回一个字符串。例如，下面的函数是不合法的：

```
function &pc_array_find_value($needle, &$haystack) {
    foreach ($haystack as $key => $value) {
        if ($needle == $value) {
            $match = $haystack[$key];
        }
    }

    return "$match is found in position $key";
}
```

这是因为"$match is found in position $key" 是一个字符串，而把引用返回给一个非变量不具有任何逻辑上的意义。如果是在PHP 5中，调用这个函数会触发一个警告信息。

与向函数中传递参数时要么是传递变量的值，要么是传递变量的引用不同，我们也可以选择不返回变量引用，而是返回变量的值。如果用=代替=&，那么PHP就会返回变量的值，而非返回引用。

参见

技巧6.3中传递变量引用的内容。

6.7 返回多个值

问题

你想让一个函数返回多个值。

方案

返回一个数组，之后再用list()来分离其中的元素：

```
function averages($stats) {
    ...
    return array($median, $mean, $mode);
}

list($median, $mean, $mode) = averages($stats);
```

讨论

如果从性能的角度看，这个方案并不十分理想。因为在PHP先创建数组，然后又破坏这个数组的过程中，会增加一些额外的开销。所以，也就有了以下的例子：

```
function time_parts($time) {
    return explode(':', $time);
}

list($hour, $minute, $second) = time_parts('12:34:56');
```

按照数字时钟的时间格式向这个函数传递一个时间字符串参数，该函数会调用explode()来将其解析成数组元素。当time_parts()返回后，再使用list()取得每个元素，并分别保存在一个标量中。尽管这种方式的效率也有点差，但也比其他可能的方法更好，因为那些方法中的代码可能会令人迷惑。

另一个可选的方案就是传递变量的引用。不过，这个方案稍显笨拙，而且并不直观。因为把必要的变量的引用传递到函数中，并不总是合乎逻辑。例如：

```
function time_parts($time, &$hour, &$minute, &$second) {
    list($hour, $minute, $second) = explode(':', $time);
}

time_parts('12:34:56', $hour, $minute, $second);
```

如果不了解函数的原型，谁也没有办法在看到这个调用语句后就知道，$hour、$minute和 $second其实就是time_parts()的返回值。

虽然也可以用全局变量，但是这会导致全局名称空间的混乱，并且也会使人难以分辨函数到底修改的是哪个变量。例如：

```
function time_parts($time) {
    global $hour, $minute, $second;
    list($hour, $minute, $second) = explode(':', $time);
}

time_parts('12:34:56');
```

再者，这里虽然能分辨清楚，是因为函数原型和调用语句放在一起的缘故，如果这个函数被包含在另一个文件当中，或者由另一个人编写，那它就会更具有隐蔽性，因而容易导致错误发生。

我们的建议是，如果要在函数中修改一个变量的值，那么就返回修改后的值，并将其赋给一个变量 —— 除非是有充分的理由，比如说存在突出的性能问题等。否则，这将更清晰、也更容易理解和维护。

参见

技巧6.3中有关传递变量引用的讨论和技巧6.11中有关变量作用域的内容。

6.8 跳跃选择返回的值

问题

一个函数返回多个值，而你只关心其中的某几个值。

方案

在list()中忽略不关心的变量：

```
// 只关心 minutes
function time_parts($time) {
    return explode(':', $time);
}

list(, $minute,) = time_parts('12:34:56');
```

讨论

尽管本方案看起来好像是存在错误，但其中的代码仍然是有效的PHP代码。这种情况在编程人员使用each()迭代遍历一个数组，但只想取得数组元素的值时最常见：

```
while (list(,$value) = each($array)) {
    process($value);
}
```

不过，还是使用foreach语句来编写显得更清楚一些：

```
foreach ($array as $value) {
    process($value);
}
```

若想减少混淆，可以不使用这一特性；但如果一个函数返回的值太多，而你又只想要其中的一两个值时，这种方法是很方便的。举一个这方面的例子，比如说你用fgetcsv()读取字段中的值，而这个函数会返回一个包含一行文本中所有字段值的数组。在这种情况下，可以运用以下方法：

```
while ($fields = fgetcsv($fh, 4096)) {
    print $fields[2] . "\n";  // 输出第三个元素的值
}
```

如果这个函数是一个组织内部编写的函数，而非PHP的内置函数，还可以让函数返回的数组中使用字符串键，因为不太容易确定数组元素2就是与'rank'关联的：

```
while ($fields = read_fields($filename)) {
    $rank = $fields['rank']; // 现在将第三个字段命名为'rank'
    print "$rank\n";
}
```

不过，下面还要再介绍一个最有效的方法：

```
while (list(,,$rank,,) = fgetcsv($fh, 4096)) {
```

```
    print "$rank\n";          //直接指定$rank
}
```

注意，不要数错了逗号的个数，否则最终可能会导致bug。

参见

技巧1.11中有使用fgetcsv()读取文件的更多内容。

6.9 返回失败信息

问题

你想指出函数失败的信息。

方案

返回false：

```
function lookup($name) {
    if (empty($name)) { return false; }
    ...
}

if (false !== lookup($name)) { /* 遵照查询结果执行 */ } else { /* 加入错误日志 */ }
```

讨论

在PHP中，非true值并不是标准的，而且容易造成错误。因此，函数应该返回定义的false关键字，因为当检测逻辑值时它是最有效的。

其他可能的返回值还有''和0。但是，当我们将这三个能转换成非true的值放到if语句中时，就会发现它们其实是不一样的。并且，有时候虽然返回0值是有意义的，但你仍然想要令其返回false。

例如，strpos()返回子字符串在原始字符串第一次出现的位置。如果没有找到子字符串，strpos()返回false。如果找到了，就会返回一个表示子字符串出现位置的整数。因此，如果要查找一个子字符串的位置，你可能会这样来写：

```
if (strpos($string, $substring)) { /* 找到了! */ }
```

然而，如果$substring实际上是在$string的开始位置找到的，那么函数的返回值是0。遗憾的是，在if语句中0会被转换成false，所以该条件语句就不会执行。下面我来告诉你正确处理strpos()的返回值的方法：

```
if (false !== strpos($string, $substring)) { /* 找到了! */ }
```

而且，false在当前乃至以后的PHP各个版本中都会保证始终是false。但其他值则不一定能保证这一点。例如，在PHP3中，empty('0')返回true，不过在PHP4中已经修改为返回false了。

参见

第5章中有关变量真值的更多介绍；strpos()函数的文档（*http://www.php.net/strpos*）和empty()函数的文档（*http://www.php.net/empty*）。

6.10 调用可变函数

问题

你想要依据变量的值来调用不同的函数。

方案

使用call_user_func()：

```
function get_file($filename) { return file_get_contents($filename); }

$function = 'get_file';
$filename = 'graphic.png';

// 调用get_file('graphic.png')
call_user_func($function, $filename);
```

如果函数能够接受不同个数的参数，可以用call_user_func_array()：

```
function get_file($filename) { return file_get_contents($filename); }
function put_file($filename, $data) { return file_put_contents($filename, $data); }

if ($action == 'get') {
    $function = 'get_file';
    $args = array('graphic.png');
} elseif ($action == 'put') {
    $function = 'put_file';
```

```
    $args = array('graphic.png', $graphic);
}

// 调用 get_file('graphic.png')
// 调用 put_file('graphic.png', $graphic)
call_user_func_array($function, $args);
```

讨论

函数call_user_func()和call_user_func_array()与标准的PHP函数略有不同。它们的第一个参数都不是一个要输出的字符串或一个要添加的数，而是要执行的函数名。在一门语言中，把传递的要调用的函数名称为回调，或者回调函数。

当需要在一个能接受很多参数的函数中调用回调函数时，就能感觉到call_user_func_array()函数的原型很方便了。因为这时候，不用在函数内部嵌入逻辑代码，而是通过func_get_args()来直接处理这些参数：

```
// 记录接受printf式的格式的函数
// 它输出一个时间戳、字符串和一个换行符
function logf() {
    $date = date(DATE_RSS);
    $args = func_get_args();

    return print "$date: " . call_user_func_array('sprintf', $args) . "\n";
}

logf('<a href="%s">%s</a>','http://developer.ebay.com','eBay Developer Program');
```

```
Sat, 23 Sep 2006 18:32:51 PDT: <a href="http://developer.ebay.com">
eBay Developer Program</a>
```

logf()函数具有与printf系列函数相同的风格：第一个参数是一个格式化分类说明符，而其余的参数则是需要插入到基于格式化代码生成的字符串中的数据。由于在格式化代码之后可能会有任意数量的参数，所以这里不能使用call_user_func()。

而是通过func_get_args()函数把取得的所有参数都放到一个数组中，然后再使用call_user_func_array()把这个数组传递到sprintf回调函数中。

在这个特定的例子中，也可以使用vsprintf()，它是sprintf()的另一个版本，而且与call_user_func_array()类似，它也能接受一个参数数组：

```
// 记录接受printf式的格式的函数
// 它输出一个时间戳、字符串和一个换行符
function logf() {
    $date = date(DATE_RSS);
```

```
    $args = func_get_args();
    $format = array_shift($args);

    return print "$date: " . vsprintf($format, $args) . "\n";
}
```

如果需要调用两个以上可能的回调函数，就要使用一个包含这些函数名的关联数组：

```
$dispatch = array(
    'add'      => 'do_add',
    'commit'   => 'do_commit',
    'checkout' => 'do_checkout',
    'update'   => 'do_update'
);

$cmd = (isset($_REQUEST['command']) ? $_REQUEST['command'] : '');

if (array_key_exists($cmd, $dispatch)) {
    $function = $dispatch[$cmd];
    call_user_func($function); // 调用函数
} else {
    error_log("Unknown command $cmd");
}
```

上面的代码从一个请求中获得命令名，并根据命令名执行相应的函数。注意，其中检查了获得的命令是否存在于可接受的命令列表中。这样就防止了从请求中传递的命令随意调用其他函数，诸如`phpinfo()`等。同时，这样也使代码更安全，并且能够易于记录错误。

还有一个好处就是，可以将多个命令映射到同一个函数，这样就可以用长、短命令名来调用函数了：

```
$dispatch = array(
    'add'      => 'do_add',
    'commit'   => 'do_commit',   'ci' => 'do_commit',
    'checkout' => 'do_checkout', 'co' => 'do_checkout',
    'update'   => 'do_update',   'up' => 'do_update'
);
```

参见

`array_key_exists()`函数的文档（*http://www.php.net/array-key-exists*），`call_user_func()`函数的文档（*http://www.php.net/call-user-func*），`call_user_func_array()`函数的文档（*http://www.php.net/call-user-func-array*）和`isset()`函数的文档（*http://www.php.net/isset*）。

6.11 在函数内部访问全局变量

问题

你需要在函数的内部访问一个全局变量。

方案

把这个全局变量放到函数的局部作用域中，并加上global关键字：

```
function eat_fruit($fruit) {
   global $chew_count;

   for ($i = $chew_count; $i > 0; $i--) {
       ...
   }
}
```

或者，直接在$GLOBALS全局数组中引用这个变量：

```
function eat_fruit($fruit) {
   for ($i = $GLOBALS['chew_count']; $i > 0; $i--) {
       ...
   }
}
```

讨论

如果在一个函数中用到许多全局变量，使用global关键字会使得函数的语法更容易理解，特别是这些全局变量被插入到字符串中时。

也可以通过global关键字在局部作用域中声明多个全局变量，每个变量之间用逗号分隔：

```
global $age,$gender,shoe_size;
```

也可以用一个可变的变量来指定全局变量的名称：

```
$which_var = 'age';
global $$which_var; // 引用全局变量 $age
```

不过，如果对用global关键字在局部作用域中声明的全局变量调用unset()，则该变量只会在函数中被清除。要想在全局作用域中清除这个变量，则必须在$GLOBALS数组中的相应元素上调用unset()：

```
$food =  'pizza';
$drink = 'beer';

function party() {
    global $food, $drink;

    unset($food);               // 吃比萨
    unset($GLOBALS['drink']);   // 喝啤酒
}

print "$food: $drink\n";
party();
print "$food: $drink\n";

pizza: beer
pizza:
```

由上面可以看出，$food没有变化，而$drink则被清除了。在函数中声明一个全局变量，就如同把这个全局变量的引用指定给一个本地变量一样：

```
$food = &GLOBALS['food'];
```

参见

有关变量作用域的文档（*http://www.php.net/variables.scope*）和变量引用的文档（*http://www.php.net/language.references*）。

6.12 创建动态函数

问题

你想要在程序运行中创建并定义一个函数。

方案

使用create_function()：

```
$add = create_function('$i,$j', 'return $i+$j;');

$add(1, 1); // 返回 2
```

讨论

函数create_function()的第一个参数是一个字符串，其中包含着新创建函数的参数，而

第二个参数则是新创建函数的函数体。使用create_function()动态创建函数的速度是比较慢的，所以如果能够预定义函数，最好还是预先定义好。

在实践当中，最频繁用到create_function()的地方是为usort()或array_walk()创建自定义的排序函数：

```
// 以颠倒的次序对文件进行排序
usort($files, create_function('$a, $b', 'return strnatcmp($b, $a);'));
```

参见

技巧4.17中有关usort()函数的介绍；create_function()函数的文档（*http://www.php.net/create-function*）和unsort()函数的文档（*http://www.php.net/usort*）。

第7章

类和对象

7.0 概述

PHP 5在面向对象编程（object-oriented programming，OOP）方面有了显著的改进。这一主要变化也是你把代码由PHP 4更新到PHP 5的一个关键原因。如果你非常钟情于OOP，那么一定会因为PHP 5所提供的功能而兴奋不已。

PHP的早期版本只限于能够编程 —— 只能定义函数，而不能定义对象。PHP 3开始引入了一个非常基本的对象形式，被称为午夜hack。在1997年的时候，没有人认为PHP的程序员会暴增，或者说能用PHP开发一个大型的程序。因此，这些限制在当时并没有被当成一个问题。

几年后，PHP增加了更多面向对象的特性，不过，其开发团队仍然没有对核心的OO代码进行重新设计，所以对对象和类的处理仍然不够好。为此，尽管PHP 4的整体性能提高了，但要想用它进行复杂的OO编程还是非常困难的 —— 虽然并非不可能。

PHP 5通过使用Zend引擎2（Zend Engine5，ZE2）解决了这一问题。ZE2不仅使PHP得以包含更高级的面向对象的特性，而且实现了对数百万计既有PHP脚本程序的高度向后兼容。

如果在PHP之外你还没有面向对象编程的经验，可能会感觉到一些惊讶。尽管某些新特性可以使事情简化，但很多特性则根本不允许你实现一些新的想法，它们会限制你的想象力。

即使这看起来好像有悖常理，但实际上正是这些限制保证了你能够快速地写出安全的代码，因为它们促进了代码重用和数据封装。这些关键的OO编程技术都会在本章中涉及到。不过，首先我们先来简单地介绍一下面向对象编程，包括相关的术语和概念。

一个类是一个程序包，它包含两样东西：数据和访问及修改数据的方法。数据部分由变量组成 —— 它们被称为属性。类的另一部分是可以使用其属性的一组函数 —— 被称为方法。

定义一个类，并不是定义一个能够访问及操作的对象。相反，类是生成对象的模板。以此为蓝本，我们可以通过一个称之为技巧化的过程，来创建一个容易控制的对象。程序中可以有基于同一个类的多个对象，就如同一个人可以拥有多本书或者多项成果一样。

类本身也处在定义的层次之中。每个类都是其上一级类的具体化。这个更具体的类称为子类，被它们修改的类称为父类。例如，可以把一个父类比作建筑物。而建筑物可以进一步分成民用建筑和商用建筑。民用建筑又可以进一步细分为住宅和公寓等。最顶级的父类也称为基类。

不论是住宅还是公寓都具有民用建筑共同属性，就像民用建筑同商用建筑具有某些共性一样。当用类来表示这些父子关系时，就可以说子类继承了父类中定义的属性和方法。继承可以实现重用父类的代码，并且只需要修改那些不适合子类特殊环境的代码。类的继承性是它超出函数的主要优势之一。这种定义源于父类的一个子类的过程称之为子类化或者扩展。

对象在PHP中除了扮演其传统的OO角色之外，还有另一个用途。由于PHP不能使用多个名称空间，所以通过类把多个属性打包到一个独立的对象中是极其重要的功能。这样就能够清晰地划分出变量与变量之间的界限。

在PHP中定义和创建类很简单：

```
class guest_book {
 public $comments;
 public $last_visitor;

 function update($comment, $visitor) {
 ...
 }

}
```

就像用function关键字定义函数一样，定义类要使用class关键字。而声明类的属性要使用public关键字，声明方法则与定义函数的过程相同。

new关键字用于技巧化一个对象：

```
$gb = new guest_book;
```

有关对象技巧化的话题在技巧7.1中有更详细的介绍。

在一个类的内部，用public关键字声明属性是可选的。也就是说，并不是必须使用public，不过这也是保证类中所有的变量对外部可见的一种方式。因为PHP并不强制要求预先声明所有的变量，所以就可以在PHP不抛出错误的情况下在类的内部创建一个变量，或者采取可以为人所知的其他方式。但是，这样会使在类的上部定义的变量列表容易引起人的误解，因为实际上它们与类中的变量列表并不相同。

在PHP 4中，声明属性使用的是var而不是public。也可以继续使用var，而将public当作一个同义词。事实上，PHP 5中存在三种不同类型的属性 —— public、protected和private属性。其中，public与PHP 4中的属性相同，而另外两种类型的属性则分别具有不同的特性。这些问题将在技巧7.4中详细讨论。

除了声明属性外，也可以为属性赋值：

```
public $last_visitor = 'Donnan';
```

而且，只能以这种形式给属性指定常量值：

```
public $last_visitor = 'Donnan'; // 正确
public $last_visitor = 9; //正确
public $last_visitor = array('Jesse'); // 正确
public $last_visitor = pick_visitor(); // 错误
public $last_visitor = 'Chris' . '9';  // 错误
```

如果试图为属性指定其他类型的值，PHP会因解析错误而停止执行。

若要为变量指定一个非常量值，那么就要通过类中的方法来实现：

```
class guest_book {
  public $last_visitor;

  public function update($comment, $visitor) {
    if (!empty($comment)) {
      array_unshift($this->comments, $comment);
      $this->last_visitor = $visitor;
    }
  }
}
```

如果访客留下了评论，就将其添加到评论数组的开始处，并将该访客设置为留言簿的最后一个访问者。其中，变量$this是一个特殊的变量，它表示的是当前对象。所以，要在这个对象的内部访问其$last_visitor属性，则以$this->last_visitor表示。

如果想在技巧化的过程中为变量指定非常量值，则可以在类的构造器中进行赋值。类的

构造器指的是在创建新对象时自动调用的一个方法，而其名字为_ _construct()，其用法见例7-1。

例7-1：在类构造器中为属性赋值

```
class guest_book {
  public $comments;
  public $last_visitor;

  public function _ _construct($user) {
    $dbh = mysqli_connect('localhost', 'username', 'password', 'sites');
    $user = mysqli_real_escape_string($dbh, $user);
    $sql = "SELECT comments, last_visitor FROM guest_books WHERE user='$user'";
    $r = mysqli_query($dbh, $sql);

    if ($obj = mysqli_fetch_object($dbh, $r)) {
      $this->comments = $obj->comments;
      $this->last_visitor = $obj->last_visitor;
    }
  }
}

$gb = new guest_book('stewart');
```

构造器的具体内容包含在技巧7.2中。注意，在PHP 4中，构造器是指与类同名的方法。如果在本例中，也就是guest_book。

小心点，不要错误地写成了$this->$size，这虽然是合法的，但其含义却与$this->size完全不同。它所表示的是访问对象中名称为$size变量中值的属性。而$size很可能没有定义，所以$this->$size就是空值。要深入了解有关可变属性名的内容，请参考技巧5.4。

除了使用->来访问方法或者成员属性外，也可以使用::。这种语法用于访问类中的静态方法。静态方法在类的所有技巧中都是一致的，因为它们不能依赖于技巧所在的特定数据环境。在静态方法中，不能使用$this变量。例如：

```
class convert {
 // 将摄氏度转换成华氏度
 public static function c2f($degrees) {
   return (1.8 * $degrees) + 32;
 }
}

$f = convert::c2f(100); // 212
```

如果要通过扩展一个现存的类来实现继承，就要用到extends关键字：

```
class xhtml extends xml {
  // ...
}
```

子类可以继承父类的方法，并且能够随意选择实现子类自己特定的版本。下面以例7-2来说明。

例7-2：*覆盖父类方法*

```
class DB {
 public $result;

 function getResult() {
  return $this->result;
 }

 function query($sql) {
  error_log("query() must be overridden by a database-specific child");
  return false;
 }
}

class MySQL extends DB {
 function query($sql) {
  $this->result = mysql_query($sql);
 }
}
```

上面的MySQL类从父类DB继承了相同的getReult()方法，但重新实现了自己特定的MySQL方法query()。

在方法名前面前置parent::用以明确地调用一个父类的方法，如例7-3所示。

例7-3：*明确地调用父类的方法*

```
function escape($sql) {
 $safe_sql = mysql_real_escape_string($sql); // escape special characters
 $safe_sql = parent::escape($safe_sql); // parent method adds '' around $sql
 return $safe_sql;
}
```

技巧7.14中介绍了如何访问被覆盖的方法。

7.1 技巧化对象

问题

你想要创建一个新的对象技巧。

方案

定义类，然后使用new来创建这个类的技巧：

```
class user {
    function load_info($username) {
       //从数据库中载入简介
    }
}

$user = new user;
$user->load_info($_GET['username']);
```

讨论

可以基于同一个类创建多个技巧：

```
$adam = new user;
$adam->load_info('adam');

$dave = new user;
$dave->load_info('adam');
```

这两个相互独立的技巧具有相同的信息。它们就像是双胞胎 —— 一母所生，但却过着各自不同的生活。

参见

技巧7.10中拷贝和克隆了对象的更多内容。类和对象的文档见*http://www.php.net/oop*。

7.2 定义对象构造器

问题

你想要定义一个在对象技巧化时调用的方法。例如，你想要在创建对象时自动从数据库向对象中载入信息。

方案

定义一个名为_ _construct()的方法：

```
class user {
    function _ _ construct($username, $password) {
        ...
    }
}
```

讨论

这个方法名为_ _ construct()（即在construct前面加两个下划线符号），以其作为对象的构造器，其用法如例7-4所示。

例7-4：定义一个对象构造器

```
class user {
  public $username;

  function _ _ construct($username, $password) {
     if ($this->validate_user($username, $password)) {
       $this->username = $username;
     }
  }
}

$user = new user('Grif', 'Mistoffelees'); // 使用内置的构造器
```

在PHP 4中，构造器具有与类相同的名字，如例7-5所示。

例7-5：在PHP 4中定义对象构造器

```
class user {
  function user($username, $password) {
  ...
  }
}
```

为了向后兼容，如果PHP 5没有发现名为_ _ construc()的方法，但是发现了一个与类同名的方法（符合PHP 4的构造器方法命名约定），它会将这个方法作为类的构造器。

像PHP 5这样，令所有构造器都有一个标准的方法名，可以使得调用父类的构造器更容易（因为这样就不需要知道父类的名称了），而且当对类进行重命名时，也不用修改构造器。

参见

技巧7.14中有关调用父类构造器的更多内容，对象构造器的文档见*http://www.php.net/oop.constructor*。

7.3 定义对象解构器

问题

你想定义一个当对象被销毁时调用的方法。例如，你想要在对象被删除之前自动地把对象中的信息保存到数据库中。

方案

当脚本停止执行时对象会自动被销毁。要强制销毁一个对象，可以使用unset()函数，如例7-6所示。

例7-6：删除一个对象

```
$car = new car; // 买一辆新车
...
unset($car);    // 将车报废
```

要想让PHP在对象被删除之前调用一个方法，就要像例7-7所示的那样定义一个名为__destruct()的方法。

例7-7：定义一个对象解构器

```
class car {
    function __destruct() {
        // 汽车经销商首脑
    }
}
```

讨论

虽然正常情况下没有必要手工清除对象，但是，如果使用一个大型循环，那么unset()的确有助于在几近失控的状况下保证内存的正常占用。

PHP 5支持对象解构器。解构器与构造器类似，只不过是在对象被删除时调用。即使不用unset()删除对象，当PHP检测到一个对象不再有用时，它仍然会调用解构器。这一动作可能发生在脚本终止执行的时候，但会提前一些。

用解构器可以在创建对象后删除对象。例如，数据库解构器会断开与数据库的连接，以释放连接占用的资源。与构造器不同，你不能向解构器传递参数，因为你无法确定它会在何时被调用。

因此，如果解构器需要任何与技巧相关的信息，可以把这些信息保存为一个属性，如例7-8所示。

例7-8：在解构器中访问与技巧相关的数据

```
// 解构器
class Database {
    function _ _ destruct() {
        db_close($this->handle);              // 关闭数据库连接
    }
}
```

解构器会在PHP终止请求之前被调用并完成数据发送。因此，可以在其中输出信息、保存数据库内容，甚至向远程服务器发送ping命令。

然而，你不能假设PHP会按照某种特定的次序来逐一销毁对象。所以，不能在解构器中引用另外一个对象 —— 因为PHP可能已经把那个对象删除了。虽然这不会导致程序崩溃，但却会令你的代码以无法预料（并且充满bug）的方式运行。

对于解构器而言，不会存在向后兼容的问题，因为在PHP 4中没有解构器。不过，这并不意味着人们不会利用其他的语言特性自造解构器。如果你曾经模仿过解构器的话，最好转移你的代码，因为PHP 5的解构器更有效率也更容易使用。

参见

unset()函数的文档见*http://www.php.net/unset*。

7.4 实现访问控制

问题

你想要为方法和属性指定可见性，以使得只有在与对象具有特殊关系的类中才能够访问它们。

方案

使用public、protected和private关键字，如例7-9所示。

例7-9：在类中使用访问控制

```
class Person {
    public $name;                    //可在任何地方访问
    protected $age;                  //在类和子类中可以访问
```

```
    private $salary;                    //只在特定类中可以访问

    public function _ _ construct() {
    // ...
    }

    protected function set_age() {
    // ...
    }

    private function set_salary() {
    // ...
    }
}
```

讨论

PHP可以允许你强制设定方法和属性的可见性。这里有三个层次的可见性：

- public（公共）。
- protected（受保护）。
- private（私有）。

用public关键字声明一个方法或属性，意味着任何人都可以调用和修改它。这与PHP 5之前所有PHP版本中的方法和属性具有的行为是相同的。

也可以给方法和属性加上protected修饰符，这样就能够限制只有在当前类及扩展了当前类的子类中才可以访问它们。

最后一种可见性是private，这是最严格的一种限制。以private声明的属性和方法只能在特定的类中才能访问。

如果你还不熟悉这些概念，可能会觉得访问控制有点多余。不过，事实上只有你真正使用了访问控制，才能创建出更加健壮的代码。因为访问控制中蕴含着一个OO编程的重要原则 —— 数据封装。

当我们在实际编码时，不可避免地会在某些阶段遇到一些问题，如存储数据的方式，函数应该接受什么参数，如何组织数据库等。可能是速度慢，操作不方便，亦或是无法添加新特性。所以，需要进一步优化。

清理代码是一件非常有益的事情，除非在优化代码的过程中意外地破坏了系统的其他部分。当程序是按照高度封装的原则进行设计的时候，底层的数据结构和数据库表是不能直接访问的。相反，你要定义一组借以实现所有操作的函数和例程。

例如，一个数据库表中保存着姓名和电子邮件地址。一个封装不好的程序会在需要的时候直接从表中读取某个人的电子邮件地址，如例7-10所示。

例7-10：选取一个电子邮件地址

```
$name   = 'Rasmus Lerdorf';
$db     = mysqli_connect();
$result   = mysqli_query($db, "SELECT email FROM users
                          WHERE name  LIKE '$name'");
$row    = mysqli_fetch_assoc($db, $r);
$email  = $row['email'];
```

另一个封装得较好的程序则会通过一个函数来实现同样的目标，如例7-11所示。

例7-11：通过函数来选取一个电子邮件地址

```
function getEmail($name) {
    $db = mysqli_connect();
    $result = mysqli_query($db, "SELECT email FROM users
                              WHERE name  LIKE '$name'");
    $row    = mysqli_fetch_assoc($db, $r);
    $email  = $row['email'];
    return $email
}

$email = getEmail('Rasmus Lerdorf');
```

使用getEmail()函数有很多好处，包括减少取得电子邮件地址的代码编写量等。然而，更重要的是它可以保证确凿地修改数据库模型，因为只需要修改getEmail()函数中的一条查询语句即可，而用不着搜索所有文件中的每一行代码，并查看SELECT语句选择的是用户表中的哪个字段。

仅通过函数很难实现良好封装的程序，因为告诉人们“这个不能碰”的方式只能是通过注释或者编码规则。

而通过对象则可以把内部实现与外部访问隔离开来。这样就可以避免人们依赖可能会改变的代码的情形，而强迫他人只能使用你的函数来访问数据。像这种类型的函数就称为访问器，因为可以通过它们来访问其他受保护的信息。当代码重构时，如果仍然能保证新访问器实现与以前相同的功能，那么就不会导致代码崩溃。

通过给某些方法和属性标记上protected或private，以标识它们可能会在将来被改变，那么人们就不应该访问它们或者知道它们可能被封装。

这样比一个宽泛的约定要好。实际上，PHP预防的正是人们在类的外部调用私有方法或者读取私有属性。因此，从外部的角度上看，这些方法和属性就相当于不存在——因为没有办法访问它们。

在面向对象编程中，在类的设计者和用户之间还存在着一个隐含的约定。用户同意不关心实现的细节。而设计者则承诺一个人只需使用公共方法就能完成所有工作——即使是在设计者重构这些类的条件下。

因为protected和private都提供了对来自类的外部访问的保护，所以在设计时就需要在两者之间做出选择。而在这两种的可见性类型之间做出选择的过程可以归结为一个必要性判断 —— 即你是否认为有人有必要在子类中调用相应的方法？

由于很难事先就拉出一个完整的列表来讨论，我们的建议倾向于使用protected而不是private。要使用private，除非你有110%的把握认为private是正确的选择，或者说的确没有任何迹象表明某人会在将来需要用到相应的方法。

7.5 防止修改类和方法

问题

你想要防止其他的开发者重新定义一个子类中的特殊方法，或者在子类化过程重新定义整个类。

方案

将这个特殊的类或者方法标记为final：

```
final public function connect($server, $username, $password) {
    // 方法在这里定义
}
```

及：

```
final class MySQL {
    // 类在这里定义
}
```

讨论

虽然正常的继承很有必要，但只有在受限制的情况下才有意义。

用final声明一个方法的最重要原因在于，如果有人会覆盖这个方法，那么就会面临着现实的风险。例如，可能会导致数据损坏、竞争条件、潜在的系统崩溃、由于忘记加锁（或忘记释放）或因修改信号而造成的死锁等。

用final声明一个方法的另一个常见的原因是这个方法很“完美”。如果你认为一个方法已经无法再改进了，那么也可以用final关键字来声明这个方法。通过final来声明方法，可以防止因子类用其他不好的方式重新实现这个方法而导致其质量下降。

然而，在这种情况要选择final关键字，一定要深思熟虑。但若想将某个人可能需要覆盖一个方法的理由全部考虑到几乎是不可能的。当你在分发一个第三方库（比如PEAR包）时，如果错误地将一个方法标记为final，则会导致令人头疼的事情发生。

要将一个方法标记为final，也就是在这个方法声明的前面加上final，如例7-12所示。

例7-12：定义一个最终方法

```
final public function connect($server, $username, $password) {
    // 方法在这里定义
}
```

这样就可以防止有人在子类化过程中创建一个不同的connect()方法。

要想防止子类化一个完整的类，不用将每个方法都标记为final。相反，可以像例7-13那样声明一个最终的类。

例7-13：定义一个最终类

```
final class MySQL {
    // 类在这里定义
}
```

一个最终类是不能被子类化的。这与一个所有方法都是最终方法的类不同。对于后者来说，它是可以被扩展的，并且虽然不能修改任何一个已经存在的方法，但能够添加额外方法。

7.6 定义字符串化的对象

问题

你想要控制PHP在输出时以何种形式显示一个对象。

方案

如例7-14所示，实现一个__toString()方法。

例7-14：为一个类定义字符串化方法

```
class Person {
    // 类的其他内容

    public function __toString() {
        return "$this->name <$this->email>";
    }
}
```

讨论

PHP为对象提供了一种控制它们如何转换为字符串的方法。这样就可以使我们不用写很多代码，就能够以一种友好的方式输出一个对象。

如果用echo或print语句输出一个对象，PHP就会调用这个对象的_ _ toString()方法，如例7-15所示。

例7-15：为一个类定义字符串化方法

```
class Person {
    protected $name;
    protected $email;

    public function setName($name) {
        $this->name = $name;
    }

    public function setEmail($email) {
        $this->email = $email;
    }

    public function _ _ toString() {
        return "$this->name <$this->email>";
    }
}
```

输出一个对象技巧：

```
$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setEmail('rasmus@php.net');
print $rasmus;
Rasmus Lerdorf <rasmus@php.net>
```

这样PHP会在后台调用_ _ toString()方法，并返回对象的字符串化内容。

该方法必须返回一个字符串，否则PHP会报错。虽然这似乎是很显而易见的，但也说不定什么时候PHP的自动转换（auto-casting）功能会让你吃一次亏，我们在这里不介绍这一功能。

例如，把字符串'9'和一个整数9当成一回事是很自然的，因为PHP通常会根据具体的环境自动地在两者之间转换，并且几乎总是能够得到正确的结果。

不过，在本技巧中的这种情况下，__toString()方法不会返回整数值。如果你感觉到自己可能会通过这个方法返回一个非字符串值的话，那么可以考虑明确地对返回值进行转换，如例7-16所示。

例7-16：转换返回的值

```
class TextInput {
    // 类的其他内容

    public function __toString() {
        return (string) $this->label;
    }
}
```

通过把$this->label转换为一个字符串，就不用再担心有人使用数字去标记这个文件输入框了。

在PHP 5.2之前，使用__toString()方法也存在一些限制。例如，这个方法不支持插入值或字符串连接操作（见例7-17）。

例7-17：调用__toString()

```
print  "PHP was created by $rasmus";
print  'PHP was created by  '. $rasmus;
printf('PHP was created by %s', $rasmus);
```

PHP中一个容易混淆的例外是使用echo和一个逗号（,）而不是句号（.）来组合项目，如例7-18所示。

例7-18：调用字符串化的对象并使用连接

```
echo   'PHP was created by ', $rasmus;
PHP was created by Rasmus Lerdorf <rasmus@php.net>
```

当把对象传递给一个接受字符串参数的函数时，PHP 5的早期版本也会把对象转换成字符串。这种情况下，需要调用__toString()（见例7-19）。

例7-19：直接调用__toString()

```
print htmlentities($rasmus);                          //错误
print htmlentities($rasmus->__toString());            //正确
```

这种方法也适用于以下情况：

- 将对象置于双引号或一个heredoc结构中。

- 用点（.）连接对象。
- 用(string)或strval()将对象转换为字符串。
- 将printf()中的对象看作是要以%s来进行格式化的一个字符串。

因此，如果需要在代码中频繁地使用_ _ toString()方法，最好还是使用PHP 5.2或者更高版本。

7.7 定义接口

问题

你想要保证一个类按照特定的名称、可见性和原型实现一个或多个方法。

方案

定义一个接口，并声明一个实现该接口的类：

```
interface Nameable {
    public function getName();
    public function setName($name);
}

class Book implements Nameable {
    private $name;

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        return $this->name = $name;
    }
}
```

其中，Nameable接口定义了命名一个对象所必需的两个方法。由于书是可以命名的，而Book类实现了Nameable接口，并在类主体中定义了这两个方法。

讨论

在面向对象的编程中，对象必须一起工作。因此，为了在系统中顺利地交互，应该能够要求一个类（或多个类）实现必要的方法。

比如说，在一个电子商务应用程序中需要知道有关全部待售商品的一组信息。这些商品可能会以不同的类表示，如Book、CD、DVD等。而有关这些商品先不论其型号是多少，最起码需要知道目录中每一件商品的名称（你可能还想让它们有一个价格、甚至一个ID号）。

这种强制地让类支持同样方法集合的机制称为接口。定义接口与定义类相似（见例7-20）。

例7-20：定义一个接口

```
interface Nameable {
    public function getName();
    public function setName($name);
}
```

定义类要用关键字class，而定义接口则要用关键字interface。在接口内部，只需定义方法的原型，而不用实现这些方法。

上面的代码创建了一个名为Nameable的接口。任何可命名的类都必须实现这个接口中定义的方法 —— getName()和setName()。

如果一个类实现了一个接口中的所有方法，就可以说它实现这个接口。如果你承诺在类中实现一个接口，可以像例7-21所示那样。

例7-21：实现一个接口

```
class Book implements Nameable {
    private $name;

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        return $this->name = $name;
    }
}
```

如果未能实现接口中定义的所有方法，或者以不同的原型实现了这些方法，会导致PHP报出致命错误。

一个类可以实现任意多个接口。例如，你可以有一个Listenable接口用于定义如何检索一件商品的音频柱。在这种情况下，CD和DVD类就再可以实现这个Listenable接口，而Book类则没有必要（当然，除非是音频书）。

在使用接口时，最重要的是必须在技巧化一个对象之前声明实现该接口的类。否则，在PHP 5中如果声明一个类实现接口可能会导致混淆。如果想避免破坏现有的应用程序，也可以不强制遵循这一做法，但最好还是不要依赖这种行为。

要检查一个类是否实现了特殊的接口，可以使用class_implements()，如例7-22所示。

例7-22：检查一个类是否实现了特定的接口

```
class Book implements Nameable {
    // .. 这里是代码
}

$interfaces = class_implements('Book');
if (isset($interfaces['Nameable'])) {
    // Book 实现了 Nameable
}
```

也可以像例7-23所示的那样使用反射类（Reflection）。

例7-23：用反射类来检查一个类是否实现了一个接口

```
class Book implements Nameable {
// .. 这里是代码
}
$rc = new ReflectionClass('Book');
if ($rc->implementsInterface('Nameable')) {
  print "Book implements Nameable\n";
}
```

参见

技巧7-19中有关反射类的更多内容。class_implements()函数的文档（*http://www.php.net/class_implements*）和interfaces的文档（*http://www.php.net/interfaces*）。

7.8 创建抽象的基类

问题

你想要创建一个“抽象”的类，或者，换句话说，一个不能直接技巧化，但可以作为子类公共基准的类。

方案

将类标记为abstract：

```
abstract class Database {
  // ...
}
```

在类的定义前添加abstract关键字就可以定义一个抽象类。

同时，抽象类中也必须要定义至少一个抽象方法。在方法定义的前面添加abstract关键字就可以定义抽象方法：

```
abstract class Database {
    abstract public function connect();
    abstract public function query();
    abstract public function fetch();
    abstract public function close();
}
```

讨论

抽象类最适宜用于一系列涉及“是什么”关系的对象。这样就可以使得它们都从一个公共的父类衍生下来具有了逻辑上的意义。区别只是在于子类是实际的类，而父类是抽象的类。

举例来说，比如一个Database类。数据库是真实的对象，所以有一个Database类是有意义的。然而，虽然有 Oracle、MySQL、Postgres、MSSQL数以百计的数据库，但我们不可能下载并安装一个通用的数据库，而必须选择某种特定的数据库。

PHP为创建一个不能技巧化的类提供了解决方案。这种不能技巧化的类就称为抽象类。例如，请看例7-24中的Database类。

例7-24：定义一个抽象类

```
abstract class Database {
    abstract public function connect();
    abstract public function query();
    abstract public function fetch();
    abstract public function close();
}
```

将一个类标记为抽象类，就是在类的定义之前加上一个abstract关键字。

抽象类必须包含至少一个被标记为abstract的方法 —— 这个（些）方法称为抽象方法。上例中的Database类包含了四个抽象方法 —— connect()、query()、fetch()和close()。这四个方法是使用数据库所必需的基本功能集合。

如果一个类中包含了一个抽象方法，那么这个类也必须声明为抽象类。然而，抽象类则可以包含非抽象方法（尽管这个例子中Database类中没有常规方法）。

抽象方法 —— 如同接口中的方法一样 —— 不在抽象类中实现，而是在扩展这个抽象父类的子类中实现。例如，你可以使用一个MySQL类，如例7-25所示。

例7-25：基于一个抽象类实现一个类

```
class MySQL extends Database {
    protected $dbh;
    protected $query;

    public function connect($server, $username, $password, $database) {
        $this->dbh = mysqli_connect($server, $username,
                                    $password, $database);
    }

    public function query($sql) {
        $this->query = mysqli_query($this->dbh, $sql);
    }

    public function fetch() {
        return mysqli_fetch_row($this->dbh, $this->query);
    }

    public function close() {
        mysqli_close($this->dbh);
    }
}
```

如果子类没有实现父类中的所有抽象方法，那么这个类仍然只能是抽象的，必须还由另一个类来进一步扩展这个类。例如，当需要创建两个MySQL类（一个将取得的信息保存为对象，而另一个则返回数组）时，可能就要经过这样的过程。

抽象方法有两个必要条件：

- 抽象方法不能定义为私有方法，因为它们需要被继承。
- 抽象方法不能定义为最终方法，因为它们需要被覆盖。

抽象类与接口是两个相似的概念，但并不相同。比如说，可以实现多个接口，但只能扩展一个抽象类。此外，在一个接口中只能定义方法原型 —— 并且不能实现它们。而一个抽象类，相对于接口而言，不仅只需要一个抽象方法，而且还能带有多个非抽象方法，甚至可以带有属性。

也可以在适用于“是什么”规则的时候使用抽象类。例如，可以说“MySQL”是一个数据库，那么将Database作为抽象类是有意义的。相反，不能说“书是一个可命名的”或者“书是一个名字”，所以Nameable应该是一个接口。

7.9 传递对象引用

问题

你想连接两个对象，以便在更新一个对象时，同时更新另一个。

方案

用“=”把一个对象的引用赋给一个变量：

```
$adam = new user;
$dave = $adam;
```

讨论

在使用“=”进行对象赋值时，并没有创建该对象的一个副本，而是传递了一个对该对象的引用。所以，修改一个也就同时修改了另一个。

这与PHP 5处理其他类型变量的方式不同，对其他类型的变量传递的则是值的副本。也不同于PHP 4，在PHP 4中所有变量都是传递值的副本。

所以，在PHP 4中需要使用“=&”来使两个对象建立连接，而现在只需要使用“=”就可以了：

```
$adam = new user;
$adam->load_info('adam');

$dave = $adam;
```

现在，$dave和$adam实际上就是同一个对象的两个名称。

参见

技巧7.10中有关克隆对象的更多内容。有关引用的文档（*http://www.php.net/references*）。

7.10 克隆对象

问题

你想要拷贝一个对象。

方案

用“=”实现通过引用来拷贝对象：

```
$rasmus = $zeev;
```

要拷贝对象的值，要使用clone：

```
$rasmus = clone $zeev;
```

讨论

PHP 5拷贝的是对象的引用而不是值。当把一个现存的对象指定给一个新变量时，新变量中保存的只是这个现存对象的引用。而不管是通过新变量还是旧变量来访问这个对象，都会得到相同的结果。

要想基于相同的内容创建包含独立值的技巧，可以使用另一种称为拷贝值的方式，即使用clone关键字。否则，第二个变量中保存的将只是对第一个对象的引用。

克隆的过程中会把第一个对象中所有的属性都拷贝到第二个对象中。其中也包括属性中保存的对象，所以克隆的对象不会与原始对象共享同一个引用。

然而，这通常不是人们所期望的行为。例如，我们考虑一下例7-26中保存着Address对象的聚合版本Person。

例7-26：使用一个聚合类

```
class Address {
  protected $city;
  protected $country;

  public function setCity($city) { $this->city = $city; }
  public function getCity() { return $this->city; }
  public function setCountry($country) { $this->country = $country; }
  public function getCountry() { return $this-> country;}
}

class Person {
 protected $name;
 protected $address;

 public function _ _ construct() { $this->address = new Address; }
 public function setName($name) { $this->name = $name; }
 public function getName() { return $this->name; }
 public function __call($method, $arguments) {
  if (method_exists($this->address, $method)) {
   return call_user_func_array( array($this->address, $method), $arguments);
  }
```

```
    }
}
```

所谓聚合类，是指在类中以某种方式嵌入了其他的类，使得不论访问原始的类，还是嵌入的类都很方便。其中需要记住的关键在于，$address属性保存的是一个Address对象。

通过这个类，例7-27中显示了在克隆相应对象时会出现什么问题。

例7-27：克隆一个聚合类

```
$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setCity('Sunnyvale');

$zeev = clone $rasmus;
$zeev->setName('Zeev Suraski');
$zeev->setCity('Tel Aviv');

print $rasmus->getName() . ' lives in ' . $rasmus->getCity() . '.';
print $zeev->getName() . ' lives in ' . $zeev->getCity() . '.';

Rasmus Lerdorf lives in Tel Aviv.
Zeev Suraski lives in Tel Aviv.
```

有意思吧。调用setName()时没有问题是因为$name属性是一个字符串，也就是说它在传递时拷贝的是值。但是，因为$address是一个对象，而对象拷贝的是引用，所以getCity()的结果出现了问题——让Rasmus搬到了Tel Aviv。

这种形式的对象克隆被称为浅克隆，或者浅拷贝。相对地，“深克隆”指的是克隆所有有关对象的克隆。这正是PHP 4中的克隆方法。

要在PHP 5中控制如何克隆对象，是通过在相应的类中实现一个_ _ clone()方法来达成的。如果这个方法存在，PHP会允许_ _ clone()覆盖默认的行为，如例7-28所示。

例7-28：在聚合类中适当地实现克隆

```
class Person {
 // ... 之前的所有代码
 public function _ _ clone() {
  $this->address = clone $this->address;
 }
}
```

在_ _ clone()方法内部，会有一个保存在$this变量中的浅拷贝来表示PHP在没有_ _clone()方法时所提供的对象。

因为PHP已经拷贝了所有属性，所以只需要覆盖不需要的属性即可。在这个例子中，$name是没问题的，只有$address需要明确地克隆。

现在，如例7-29所示，克隆行为被纠正了。

例7-29：克隆一个聚合类。

```
$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setCity('Sunnyvale');

$zeev = clone $rasmus;
$zeev->setName('Zeev Suraski');
$zeev->setCity('Tel Aviv');

print $rasmus->getName() . ' lives in ' . $rasmus->getCity() . '.';
print $zeev->getName() . ' lives in ' . $zeev->getCity() . '.';

Rasmus Lerdorf lives in Sunnyvale.
Zeev Suraski lives in Tel Aviv.
```

使用克隆操作符克隆保存在属性中的对象，会导致PHP检测这些对象中是否包含__clone()方法。如果包含一个，PHP就会调用它。而且，即使对于嵌套在更深层中的对象也会重复这样地检测。

这一过程保证了正确地克隆完整的对象，并且也证明了它为什么被称为深度拷贝。

参见

技巧7.9中有关传递对象引用的介绍。

7.11 重要的属性访问

问题

你想在读、写对象的属性时运行处理器函数。这样就可以用通用的代码来处理对类中属性的访问。

方案

使用魔术方法——__get()和__set()来拦截对属性的请求。

为了提高分离的程度，还要实现__isset()和__unset()方法，以便当我们用isset()来检测属性或者用unset()来删除属性时，能够保证类的行为正确。

讨论

属性的重载可以让我们对用户自然隐藏对象属性的位置和用于保存这些属性的数据结构。

例如，例7-30中的Person类以一个数组$__data来存储变量。

例7-30：实现魔术访问器方法

```
class Person {
    private $_ _ data = array();

    public function _ _ get($property) {
        if (isset($this->_ _ data[$property])) {
            return $this->_ _ data[$property];
        } else {
            return false;
        }
    }

    public function _ _ set($property, $value) {
        $this->_ _ data[$property] = $value;
    }
}
```

例7-31显示了如何使用Person类。

例7-31：使用魔术访问器方法

```
$johnwood = new Person;
$johnwood->email = 'jonathan@wopr.mil';// 设置 $user->_ _data['email']
print $johnwood->email;                // 读取 $user->_ _data['email']
jonathan@wopr.mil
```

当设置数据时，_ _set()会重写$_ _data中的同名元素。同样，使用_ _get()可以捕获调用并返回正确的数组元素。

使用这些方法和一个数组作为变量存储的替代方案，可以实现更严谨的对象封装。只要使用_ _get()和_ _set()就可以省去为每个类属性都编写一对访问器方法的麻烦。

通过_ _get()和_ _set()，我们可以像使用公共属性——如$johnwood->name一样，而不会妨碍封装。这是因为程序设计人员不会直接对属性进行读取和写入，而是按照已经确定的访问器方法的路径进行。

_ _ get()方法接受一个属性名作为其唯一的参数。在该方法内部，先要检查在$_ _ data中是否存在一个相应的属性值。如果有，就返回那个值；否则，返回false。

注意： 当你读取$johnwood->name时，实际上你是在调用_ _ get('name') ，并得到返回$_ _ data['name'] —— 除了你想进行其他无关的操作。

_ _ set()方法接受两个参数 —— 属性名和新的值。除此之外，方法中的逻辑与_ _ get()类似。

除了可以减少类中的方法外，这些魔术方法也使得实现一套集中式的输入和输出验证功能更加便利。

而且，例7-32中也显示了对于一个给定的类而言，我们还可以精确地限制哪个属性是合法或者是不合法的。

例7-32：使用魔术访问器方法限制对属性访问

```
class Person {
    // 列出了 person 和 email 是有效的属性
    protected $_ _data = array('person', 'email');

    public function _ _get($property) {
        if (isset($this->_ _data[$property])) {
            return $this->_ _data[$property];
        } else {
            return false;
        }
    }

    // 实施只有预定义的属性才能设置的限制
    public function _ _set($property, $value) {
        if (isset($this->_ _data[$property])) {
            return $this->_ _data[$property] = $value;
        } else {
            return false;
        }
    }
}
```

在上面更新后的代码中，我们通过定义$_ _data的属性明确地列出了对象中有效的属性名。然后，在_ _set()内部，又使用isset()来确认所有写入的属性都是可以接受的。

之所以没有把$_ _data属性的可见性设置为public，是为了防止故意捣乱的人读写这个类中的属性。要不然，可能会有人干这样的事：

```
$person = new Person;
$person->_ _data['fake_property'] = 'fake_data';
```

因为魔术访问器不会捕捉已有的属性。

注意一下这个关键的实现细节。在特定的情况下，如果你想让别人来扩展这个类，别人

则有可能加入一个属性，而这个属性会与你想要通过_ _get()和_ _set()来处理的属性发生冲突。为此，例7-32中在调用$_ _data时在前面添加了两个下划线。

所以，可以在使用魔术访问器时考虑给类中所有“真实的”属性添加前缀，以防止用常规方法处理的属性与那些通过确定的_ _get()和_ _set()来处理的属性发生冲突。

使用_ _get()和_ _set()有三个不足。第一，这两个方法只会捕捉缺少的属性。如果你为你的类定义了一个属性，那么当访问这个属性时PHP不会调用_ _get()和_ _set()。

而且，即使你试图访问的属性在当前的作用域中并不可见（例如，访问类中存在但因为声明为private而对你不可见的属性）也同样如此。这样做会导致PHP发出一个致命错误：

```
PHP Fatal error:  Cannot access private property...
```

第二，这两个方法会完全破坏任何属性继承的想法。如果父对象中有一个_ _get()方法，而你在子类中又实现了自己的_ _get()方法，那么你的对象不会正确地运行，因为父类的_ _get()方法永远不会被调用。

当然，这个问题可以通过调用parent:: _ _get()来解决，但这仍然需要明确地声明，而没有OO设计理念所提倡的那样“自然取得”。

这些问题并不是全部，因为还没有考虑到isset()和unset()方法。比如说，当你想要检测一个重载的属性是否isset()时，可能会得不到准确的结果，因为PHP不会调用_ _get()方法。

要解决上述问题，可以在你自己的类中实现_ _isset()和_ _unset()方法，如例7-33所示。

例7-33：为isset()和unset()实现魔术方法

```
class Person {
    // 列出了 person 和 email 是有效的属性
    protected $data = array('person', 'email');

    public function _ _get($property) {
        if (isset($this->data[$property])) {
            return $this->data[$property];
        } else {
            return false;
        }
    }

    // 实施只有预定义的属性才能设置的限制
    public function _ _ set($property, $value) {
        if (isset($this->data[$property])) {
            return $this->data[$property] = $value;
```

```
        } else {
            return false;
        }
    }

    public function _ _isset($property) {
        if (isset($this->data[$property])) {
            return true;
        } else {
            return false;
    }

    public function _ _unset($property) {
        if (isset($this->data[$property])) {
            return unset($this->data[$property]);
        } else {
            return false;
        }
    }
}
```

_ _isset()方法用于检查$data属性的内部，并根据要检验的属性是否存在返回true或者false。

同样地，_ _unset()将参数值传递给unset()以应用到“真实的”属性上，或者如果该属性没有设置就返回false。

虽然在使用_ _get()和_ _set()时不用同时实现_ _isset()和_ _unset()方法，但最好还是实现它们，因为很难保证不会用到它们。如果没有实现它们，那么当别人（或许是你自己）不知道（或者忘记）类中使用了魔术访问器方法时，很可能会导致混乱。

但是，_ _isset()和_ _unset()方法只针对PHP 5.1之后的版才有效。

不考虑使用魔术访问器方法的其他原因还有：

- 速度相对较慢。不仅比直接访问属性的速度慢，而且也比明确地为所有属性编写访问器方法的速度慢。
- 使用魔术访问器方法就不可能再使用反射类，以及诸如phpDocumentor这样的工具将代码自动文档化。
- 不能将其用于静态属性。

参见

魔术方法的文档（*http://www.php.net/manual/en/language.oop5.magic.php*）。

7.12 调用由另一个方法返回对象的方法

问题

你想要调用由另一个方法返回的对象的方法。

方案

直接在第一个方法后面调用第二个方法：

```
$orange = $fruit->get('citrus')->peel();
```

讨论

PHP自然会知道首先调用$fruit->get('citrus')，然后在返回的对象上再调用peel()方法。

这是在PHP 4基础上的一个改进措施。在PHP 4中，要实现同样的功能还需要使用一个临时变量：

```
$orange = $fruit->get('citrus');
$orange->peel();
```

这是PHP 5的又一个成功之处！

7.13 聚合对象

问题

你想把两个或更多个对象组合在一起，使结果就像是一个对象。

方案

聚合对象并用_ _call()魔术方法截获对方法的调用，然后为这些调用确定相应的路线：

```
class Address {
    protected $city;

    public function setCity($city) {
        $this->city = $city;
    }
```

```
    public function getCity() {
        return $this->city;
    }
}

class Person {
    protected $name;
    protected $address;

    public function _ _construct() {
        $this->address = new Address;
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function _ _call($method, $arguments) {
        if (method_exists($this->address, $method)) {
            return call_user_func_array(
                array($this->address, $method), $arguments);
        }
    }
}

$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setCity('Sunnyvale');

print $rasmus->getName() . ' lives in ' . $rasmus->getCity() . '.';
Rasmus Lerdorf lives in Sunnyvale.
```

当构造每一个Person对象时，都会创建一个Address对象的技巧。当所调用的方法在Person中没有定义时，_ _call()就会捕获它们，在符合条件时，再用call_user_func_array()将其分派给相应的对象来处理。

讨论

在这个技巧中，我们不能说Person“是一个”Address，反之亦然。因此，用其中一个类去扩展另一个类是没有任何意义的。

不过，保持这两个类的独立却是有意义的。这样，不仅为我们最大限度地提供了灵活性和可重用性，同时也减少了重复的代码。因此，我们可以考虑另一个规则——“有一个”规则是否适用。因为Person“有一个”Address，所以将这两个类聚合起来是有意义的。

通过聚合，可以将一个对象作为其他一个或多个对象的容器。这也是解决多重继承问题的另一种方案，因为我们很容易就可以把一些较小的组件拼合成一个对象。

例如，一个Person类可以包含一个Address对象。显然，人可以有地址这么一个属性。然而，并非只有人才有地址属性 —— 商店或者其他机构也可以有地址。因此，与其在Person类中硬编码一个地址属性，不如创建一个独立的Address类使其能被更多的类使用才更有意义。

例7-34展示了按照以上分析编写的代码。

例7-34：聚合一个地址对象

```
class Address {
    protected $city;

    public function setCity($city) {
        $this->city = $city;
    }

    public function getCity() {
        return $this->city;
    }
}

class Person {
    protected $name;
    protected $address;

    public function _ _construct() {
        $this->address = new Address;
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function _ _call($method, $arguments) {
        if (method_exists($this->address, $method)) {
            return call_user_func_array(
                array($this->address, $method), $arguments);
        }
    }
}
```

其中，Address类用于存储城市信息，并带有两个用于数据读写的访问器方法 —— setCity()和getCity()。

Person类与Address类相似，有一个setName()和一个getName()方法，除此之外，它还有另外两个方法 —— _ _construct()和_ _call()。

它的构造器初始化一个Address对象，并将该对象保存在一个受保护的$address属性中。这样就可以从Person类内部访问$address属性，但不能从外部直接访问它。

在理想的情况下，当我们调用Address中的一个方法时，PHP就会自动运行那个方法。但这种情况不会发生，因为Person并没有扩展Address。所以，我们必须自己动手编写代码将调用与适当的方法关联起来。

包装方法是一种选择。例如：

```
public function setCity($city) {
        $this->address->setCity($city);
}
```

上面的setCity()方法将数据传递到保存在$address中的setCity()方法中。虽然这样很容易，但也很枯燥，因为我们必须要为每一个方法都写个包装方法。

而使用_ _call()则可以把这些方法集中起来自动地完成这一过程，如例7-35所示。

例7-35：用_ _call()集中方法调用

```
public function _ _call($method, $arguments) {
        if (method_exists($this->address, $method)) {
            return call_user_func_array(
                array($this->address, $method), $arguments);
        }
}
```

_ _call()方法会捕捉到任何对未在类中定义的方法的调用。在调用这个方法时，需要传递两个参数：一个是方法名，另一个是保存着要传递给方法的参数的数组。通过第一个参数我们可以知道要调用什么方法，所以可以确定是否适合将其分派给$address。

在这里，如果我们检测到这个方法是Address类的一个有效的方法，就会传递它。检测是通过使用method_exists()方法并为其提供两个参数 —— 第一个参数是对象，第二个参数是方法名来进行的。

如果检测结果返回true，也就是说调用的方法是有效的，那么我们就可以调用它。不幸的是，我们接下来要面对的是打开$arguments数组参数的难题。这可是件苦差事。

然而，一个名叫call_user_func_array()的不常用而且很古怪的函数可以用来解决这个难题。这个函数可以让我们调用一个用户函数，并且在数组中传递参数。传递给它的第一个参数是你的函数名，第二个参数是参数数组。

但是，在这里我们想调用的是对象的一个方法而非一个函数。有一种特殊的语法可以用于这种情况，即不传递函数名，而是代之以传递一个包含两个元素的数组。数组中的第一个元素是对象，另一个元素是要调用的方法名。

这样，就可以通过call_user_func_array()调用这个对象的方法了。之后，必须将call_user_func_array()的结果返回给原始的调用程序，要不然返回的值就会被悄无声息地废弃掉。

下面就是一个Person调用分别在Person自身和Address中定义的方法的例子：

```
$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setCity('Sunnyvale');

print $rasmus->getName() . ' lives in ' . $rasmus->getCity() . '.';
Rasmus Lerdorf lives in Sunnyvale.
```

虽然setCity()和getCity()不是Person的方法，但我们已经将它们聚合到这个类中了。

可以将一个对象聚合到一个类中，也可以进一步选择把哪个方法暴露给外部用户。这样就必须基于方法名进行一些必要的条件过滤。

参见

魔术方法的文档（*http://www.php.net/manual/en/language.oop5.magic.php*）。

7.14 访问被覆盖的方法

问题

你想要一个在子类中访问已经被覆盖的父类中的方法。

方案

给方法名加上parent::前缀：

```
class shape {
    function draw() {
        // 输出到屏幕
    }
}
```

```
class circle extends shape {
   function draw($origin, $radius) {
      // 检验数据
      if ($radius > 0) {
          parent::draw();
          return true;
      }

      return false;
   }
}
```

讨论

当子类中的方法覆盖了父类中的方法时，除非明确地引用，否则不会自动调用父类的方法。

在这个解决方案中，由于我们希望能够接受各种圆形的参数并对数据进行检验，所以在子类circle中覆盖了draw()方法。然而，在覆盖了该方法后，我们仍然想要执行原始的shape::draw()动作，以便完成实际的绘制工作，所以在方法中，如果$radius大于0，我们就会调用parent::draw()。

只有在类的内部代码中才可以使用parent::前缀。如果在类的外部调用parent::draw()则会发生解析错误。例如，如果在circle::draw()检验完半径之后，还想要调用shape::draw()，那么下面的代码是不起作用的：

```
$circle = new circle;
if ($circle->draw($origin, $radius)) {
    $circle->parent::draw();
}
```

这种情况对于调用构造器也同样适用，所以像下面的代码是很常见的：

```
class circle {
    function _ _construct($x, $y, $r) {
        // 首先调用shape的构造器
     parent:: _ _construct();
        // 再针对circle完成具体操作
    }
}
```

这种对调用父类构造器的简化，得益于PHP 5对构造器实行的一致的命名方案。而在PHP 4中如果想以一个不算脆弱的方式来实现这样的调用，是需要绕过诸多限制的。

参见

技巧7.2中有关对象构造器的更多内容。有关父类的文档（*http://www.php.net/keyword.parent*）和get_parent_class()方法的文档（*http://www.php.net/get-parent-class*）。

7.15 使用方法的多态性

问题

你想要根据传递给方法的参数数量和类型来决定执行不同的代码。

方案

PHP没有支持方法多态性的内在特性。但是，可以模仿可变类型检测函数。像下面的combine()函数就使用了is_numeric()、is_string()、is_array()和is_bool()：

```
// combine() 用于数字相加、连接字符串、合并数组
// 以及将布尔参数进行按位与运算
function combine($a, $b) {
    if (is_int($a) && is_int($b))     {
        return $a + $b;
    }

    if (is_float($a) && is_float($b))   {
        return $a + $b;
    }

    if (is_string($a) && is_string($b))  {
        return "$a$b";
    }

    if (is_array($a) && is_array($b))   {
        return array_merge($a, $b);
    }

    if (is_bool($a) && is_bool($b))    {
        return $a & $b;
    }

    return false;
}
```

讨论

因为PHP不允许我们在方法原型中声明变量的类型，因此就不能像Java和C++那样基于

方法的定义来有条件地调用不同的方法。不过，我们可以声明一个函数并使用switch语句来手工地重建这一特性。

例如，PHP可以让我们通过GD来编辑图像。如果能向一个图像类中传递图像的位置（远程或者本地）或者PHP已经指定给当前图像流的句柄将是很方便的。例7-36显示的pc_Image类，就可以实现这一点。

例7-36：pc_Image类

```
class pc_Image {

    protected $handle;

    function ImageCreate($image) {
        if (is_string($image)) {
            // 简单地推测文件类型

            // 取得文件后缀
            $info = pathinfo($image);
            $extension = strtolower($info['extension']);
            switch ($extension) {
            case 'jpg':
            case 'jpeg':
                $this->handle = ImageCreateFromJPEG($image);
                break;
            case 'png':
                $this->handle = ImageCreateFromPNG($image);
                break;
            default:
                die('Images must be JPEGs or PNGs.');
            }
        } elseif (is_resource($image)) {
            $this->handle = $image;
        } else {
            die('Variables must be strings or resources.');
        }
    }
}
```

在上面的类中，会把任何传递进来的字符串当作文件的位置，进而再使用pathinfo()取得文件的扩展名。在取得了扩展名之后，就可以调用不同的ImageCreateFrom()函数精确地打开图像并创建一个句柄。

如果传递进来的参数不是字符串，就直接作为一个GD流来处理，即将其视为一种资源。因为没有转换的必要，所以就直接将该流指定给$handle。当然，如果是在真正的开发环境中使用这个类，必须针对错误处理编写更健壮的代码。

方法的多态性也包括带有不同参数个数的方法。在方法中检测参数个数的代码与使用func_num_args()来处理可变参数的代码是一样的。有关这一点在技巧6.5中讨论过。

参见

技巧6.5中有关可变参数函数的讨论。is_string()函数的文档（*http://www.php.net/is-string*），is_resource()函数的文档（*http://www.php.net/is-resource*）和pathinfo()函数的文档（*http://www.php.net/pathinfo*）。

7.16 定义类常量

问题

你想要在每个类的基础上，而不是全局的基础上定义常量。

方案

如同定义类属性，只不过在该属性前加上const标签：

```
class Math {
    const pi = 3.14159; //通用的
    const  e = 2.71828; // 常量
}

$area = math::pi * $radius * $radius;
```

讨论

PHP在类中重用了其全局常量的概念。从本质上来说，这些常量其实是final属性。

用const标签来声明类常量：

```
class Math {
    const pi = 3.14159; // 通用的数
    const  e = 2.71828; // 常数
}

$area = math::pi * $radius * $radius;
```

与访问静态属性类似，在访问类常量时也不用首先技巧化一个类的新技巧，而且要使用双冒号（::）。在同一个类内部访问该常量时需要在常量名前面加上self::前缀。

与属性不同的是，常量的前面没有美元符号（$）：

```
class Circle {
    const pi = 3.14159;
    protected $radius;
```

```
    public function _ _construct($radius) {
        $this->radius = $radius;
    }

    public function circumference() {
        return 2 * self::pi * $this->radius;
    }
}

$c = new circle(1);
print $c->circumference();
6.28318
```

这个例子以1为半径创建了一个圆，之后又调用了circumference()方法来计算圆的周长。要使用类的pi常量，就需要在circumference方法中引用它；否则，PHP会尝试使用全局pi常量计算周长的值：

```
define('pi', 10); // 全局 pi 常量

class Circle {
    const pi = 3.14159; // 类中的 pi 常量
    protected $radius;

    public function _ _construct($radius) {
        $this->radius = $radius;
    }

    public function circumference() {
        return 2 * pi * $this->radius;
    }

}

$c = new circle(1);
print $c->circumference();
20
```

噢！ PHP使用的是10而不是3.14159，所以新答案是20而不是6.28318。

虽然意外地重新定义PI的可能性不太大（因为可以使用内置的M_PI常量），但毕竟还是有可能的。

不能把一个表达式的值赋给常量，也不能使用传递到脚本中的信息：

```
// 无效
class permissions {
    const    read = 1 << 2;
    const   write = 1 << 1;
    const execute = 1 << 0;
}
```

```
// 无效而且不安全
class database {
    const debug = $_REQUEST['debug'];
}
```

不论是permissions中的常量还是debug中的数据库常量都是不允许的，因为它们不是固定不变的。即使是在第一个例子 —— 1<<2中，虽然PHP不需要读取外部的数据，仍然是不允许的。

因为在访问常量时需要使用明确无误的名称，所以不管是self::，还是类名，都不能在程序运行过程中动态计算生成。也就是说，必须提前声明。例如：

```
class Constants {
    const pi = 3.14159;

    // 类的其余代码
}

$class = 'Constants';

print $class::pi;
```

即使以上代码中的语法结构对于非常量表达式而言是合法的（如$class->pi），但对于常量仍然会产生解析错误。

参见

类常量的文档（*http://www.php.net/manual/en/language.oop5.constants.php*）。

7.17 定义静态属性和方法

问题

你想要定义一个类方法，并且使其在不技巧化的情况下就能访问。

方案

将方法声明为static：

```
class Format {
    public static function number($number, $decimals = 2,
                                  $decimal = ',', $thousands = '.') {
        return number_format($number, $decimals, $decimal, $thousands);
```

```
    }
}

print Format::number(1234.567);
1,234.57
```

讨论

偶尔，你也许会想定义一些对象的方法，但希望能在不技巧化对象的情况下就可以访问这些方法。在PHP 5中，把一个方法声明为静态方法，就可以直接调用了：

```
class Format {
    public static function number($number, $decimals = 2,
                                  $decimal = ',', $thousands = '.') {
        return number_format($number, $decimals, $decimal, $thousands);
    }
}

print Format::number(1234.567);
1,234.57
```

因为静态方法不需要通过一个对象技巧来调用，而是通过类名来调用的。但是，不要在类名前面加美元符号（$）。

静态方法不是用箭头（->）引用，而是用双冒号（::）引用（双冒号会告诉PHP该方法是一个静态方法）。所以在上面的例子中，Format类的number()方法就是通过Format::number()来调用的。

格式化数字不需要依赖于其他对象的属性和方法。因此，把这个方法声明为静态方法是有意义的。比如说，在一个购物车程序中，通过静态方法能够仅用一行代码就把商品的价格格式化为适当的形式，并且使用的还是对象而不是全局函数。

静态方法不会由定义它的类的一个特殊技巧来操作。PHP不会为你在这个方法的内部“构造”一个临时的对象。也就是说，你不能在一个静态方法内部引用$this变量，因为不存在调用该方法的$this。调用静态方法与调用一个常规的函数没有什么区别。

PHP 5中还有一个称作静态属性的特性。一个类的所有技巧都会共用这些属性。因此，静态属性就像是一个类名称空间中的全局变量一样。

使用静态属性的一种可能是在多个Database对象之间共享一个数据库连接。出于效率的考虑，我们不应该在每次技巧化Database类时都创建一个新的连接。相反，而是要像例7-37所示的那样只创建一次，然后在其余的每个技巧中重用这个连接。

例7-37：跨技巧共享一个静态属性

```
class Database {
    private static $dbh = NULL;

    public function _ _construct($server, $username, $password) {
        if (self::$dbh == NULL) {
            self::$dbh = db_connect($server, $username, $password);
        } else {
            // 重用已创建的连接
        }
    }
}

$db  = new Database('db.example.com', 'web', 'jsd6w@2d');
// 完成一系列的查询

$db2 = new Database('db.example.com', 'web', 'jsd6w@2d');
// 再完成另一些查询
```

静态属性和静态方法类似，都使用双冒号来引用。如果要在一个类中引用静态属性，也需要使用一个特殊的self前缀。self之与静态属性和静态方法，就如同$this之与技巧属性和技巧方法一样。

上例中的构造器使用了self::$dbh访问静态连接属性。当技巧化$db时，如果dbh仍然设置为NULL，构造器就会调用db_connect()与数据库建立一个新的连接。

但是当技巧化$db2时就不会重复这一过程了，因为dbh已经被设置为数据库的句柄了。

参见

static关键字的文档（*http://www.php.net/manual/en/language.oop5.static.php*）。

7.18 控制对象的序列化

问题

你想要在对对象执行serialize()和unserialize()时控制对象的行为。当你需要建立和关闭与远程资源（如数据库、文件和Web 服务等）的连接时，这是非常有用的。

方案

如例7-38所示的那样定义魔术方法_ _sleep()和wakeUp()。

例7-38：用_ _sleep()和wakeUp()控制序列化

```
<?php
class LogFile {
    protected $filename;
    protected $handle;

    public function _ _construct($filename) {
        $this->filename = $filename;
        $this->open();
    }

    private function open() {
        $this->handle = fopen($this->filename, 'a');
    }

    public function _ _ destruct($filename) {
        fclose($this->handle);
    }

    // 当对象序列化时调用
    // 返回一个可序列化的对象属性的数组
    public function_ _ sleep() {
        return array('filename');
    }

    // 当对象反序列化时调用
    public function_ _ wakeUp() {
        $this->open();
    }
}
?>
```

讨论

在PHP中序列化一个对象时，会保存对象的所有属性。但其中包含指向外部资源（如数据库、文件和Web 服务等）的连接或者句柄。

这些连接或者句柄必须在对象反序列化时重新建立，否则对象的运行就会出现问题。可以通过编写代码来实现这一点，但更好的办法则是将这个过程抽象出来，让PHP在后台进行处理。

为此，就要使用_ _sleep()和_ _wakeUp()这两个魔术方法。PHP会在对象调用serialize()进行序列化时调用_ _sleep()方法，在对象调用unserialize()进行反序列化时调用_ _wakeUp()方法。

例7-38中的LogFile类有五个简单的方法。其中构造器方法接受一个文件名并保存起来以备将来使用。open()方法会打开文件并保存句柄，该句柄会在对象解构时关闭。

_ _sleep()方法在对象序列化过程中返回一个保存着对象属性的数组。由于文件的句柄在序列化中不会被保留，所以该方法只返回array('filename')，这也是所有要保存的信息。

这也就是为什么在对象被序列化以后，我们还要重新打开文件的原因。而重新打开文件的操作是在_ _wakeUp()方法中完成的，其中调用了与构造器所用相同的open()方法。因为我们不能给_ _wakeUp()传递参数，所以还需要从一个地方获得文件名。幸运的是，这个方法可以访问对象的属性，这也是为什么把文件名保存在对象属性中的原因所在。

这里关键是要知道，在一个单独的请求中可以对同一个对象技巧进行多次序列化，甚至在将对象序列化后仍然可能继续使用该对象。因此，不要在_ _sleep()方法中做任何妨碍序列化或反序列化动作的事。_ _sleep()方法只能用于排除那些由于会占用过多磁盘空间而不应该被序列化、或者要基于其他数据计算并且在对象反序化过程中应该被重新计算或者被更新的属性。

这也是为什么把fclose()放在结构方法中，而没有放在_ _sleep()中的原因。

参见

魔术方法的文档（*http://www.php.net/manual/en/language.oop5.magic.php*）。unserialize()函数的文档（*http://www.php.net/unserialize*）和serialize()函数的文档（*http://www.php.net/serialize*）。

7.19 分析对象

问题

你想要对一个对象进行检查，看一下它有什么方法和属性，以便编写可以使用任何普通对象的代码，而不管其类型。

方案

使用反射（Reflection）类来查明对象的信息。

要想获得一个类的快速预览，可以调用Reflection::export()：

```
// 了解汽车
Reflection::export(new ReflectionClass('car'));
```

或者检测特定的数据：

```
$car = new ReflectionClass('car');
if ($car->hasMethod('retractTop')) {
    // 汽车是可以伸缩的
}
```

讨论

很少能在不实际查看内部代码的情况下就可以知道类的内部细节。尽管如此，通过反射类，我们既可以大概地提取出面向对象的特性，如类、方法和属性；也可以提取出非OO特性，如函数和扩展等。

这种能力对于在项目中应用了不同类的情况很有帮助。比如说，创建自动化的类文档、通用的对象调试器以及声明节约装置，像serialize()等。

为了演示Reflection类如何工作，例7-39包含了一个示范性的Person类，其中使用了很多PHP 5的OO特性。

例7-39：Person类

```
class Person {
    public $name;
    protected $spouse;
    private $password;

    public function __construct($name) {
        $this->name = $name
    }

    public function getName() {
        return $name;
    }

    protected function setSpouse(Person $spouse) {
        if (!isset($this->spouse)) {
            $this->spouse = $spouse;
        }
    }

    private function setPassword($password) {
        $this->password = $password;
    }
}
```

为获得对这个类的快速预览，调用Reflection::export()：

```
Reflection::export(new ReflectionClass('Person'));
Class [ <user> class Person ] {
```

```
@@ /www/reflection.php 3-25

- Constants [0] {
}

- Static properties [0] {
}

- Static methods [0] {
}

- Properties [3] {
  Property [ <default> public $name ]
  Property [ <default> protected $spouse ]
  Property [ <default> private $password ]
}

- Methods [4] {
  Method [ <user> <ctor> public method _ _construct ] {
    @@ /www/reflection.php 8 - 10

    - Parameters [1] {
      Parameter #0 [ $name ]
    }
  }

  Method [ <user> public method getName ] {
    @@ /www/reflection.php 12 - 14
  }

  Method [ <user> protected method setSpouse ] {
    @@ /www/reflection.php 16 - 20

    - Parameters [1] {
      Parameter #0 [ Person or NULL $spouse ]
    }
  }

  Method [ <user> private method setPassword ] {
    @@ /www/reflection.php 22 - 24

    - Parameters [1] {
      Parameter #0 [ $password ]
    }
  }
 }
}
```

静态方法Reflection::export()将一个ReflectionClass类的技巧作为参数，返回了丰富的信息。我们已经看到了，返回的结果中详细地包含了类中常量、静态属性、静态方法、属性和方法的数量。每个项目都是相对独立的信息部件。例如，所有包含可见性标识符（private、protected或public）的条目和基于各自定义的带参数列表的方法。

Reflection::export()不仅报告了类中所有元素的定义，甚至还给出了相应的行数！如果要从文件中摘录代码，并放到文档中，就使用它会非常方便。

例7-40显示了一个用于搜索文件名和方法或函数起始行位置的简短的命令行脚本。

例7-40：使用反射查寻函数和方法的定义

```
<?php
if ($argc < 2) {
    print "$argv[0]: function/method, classes1.php [, ... classesN.php]\n";
    exit;
}

// 得到函数名
$function = $argv[1];

// 包含文件
foreach (array_slice($argv, 2) as $filename) {
    include_once $filename;
}

try {
    if (strpos($function, '::')) {
        // 是方法
        list ($class, $method) = explode('::', $function);
        $reflect = new ReflectionMethod($class, $method);
    } else {
        // 是函数
        $reflect = new ReflectionFunction($function);
    }

    $file = $reflect->getFileName();
    $line = $reflect->getStartLine();

    printf ("%s | %s | %d\n", "$function()", $file, $line);
} catch (ReflectionException $e) {
    printf ("%s not found.\n", "$function()");
}

?>
```

将函数或方法名作为第一个参数，而将包含文件作为后续的参数。这些文件都将被包含，所以要保证它们不会输出任何信息。

下一步是确定第一个参数是方法还是函数。因为方法是以class::method的形式出现的，所以可以通过strpos()来进行辨别。

如果是一个方法，就使用explode()将方法从类中分离出来，并将二者都传递给ReflectionMethod。如果是一个函数，那么就直接技巧化一个ReflectionFunction。

因为ReflectionMethod扩展了ReflectionFunction，所以可以调用任何一个类的getFileName()和getStartLine()方法。这两个方法会搜索需要输出的信息，并通过printf()完成输出。

如果试图用一个未定义的方法名来技巧化ReflectionMethod或ReflectionFunction，它们会抛出一个ReflectionException异常。之后，会捕获该异常并显示错误信息。

技巧7.23中列举了一个能够输出全部用户定义的方法和函数的同类信息的更复杂脚本。

如果你只是需要粗略地了解一个对象技巧，并不想使用Reflection类，那么，可以选用var_dump()、var_export()或print_r()打印对象的值。每个函数输出的信息都不尽相同，而var_export()还可以选择返回信息而不是显示信息。

参见

技巧5.8中有关输出变量的更多内容。Reflection的文档（*http://www.php.net/manual/en/language.oop5.reflection.php*），var_dump()函数的文档（*http://www.php.net/var-dump*），var_export()函数的文档（*http://www.php.net/var-export*）和print_r()函数的文档（*http://www.php.net/print-r*）。

7.20 检查某对象是不是一个特定类的技巧

问题

你想要检查一个对象是不是一个特定类的技巧。

方案

将一个特定类的技巧作为参数传递给函数，并在函数原型中指定类名：

```
public function add(Person $person) {
        // 把 $person 添加到地址簿
    }
}
```

在其他情况下，可以使用instanceof操作符：

```
<?php
$media = get_something_from_catalog();
if ($media instanceof Book) {
  // 看书学习
```

```
} else if ($media instanceof DVD) {
  // 看电影
}
?>
```

讨论

在对象中进行强制类型控制的一种方式是使用类型提示。类型提示可以起到告知PHP传递到函数或方法中的对象必须是某个类技巧的作用。

为此，需要在函数和方法原型中指定一个类名。从PHP 5.1开始，也可以通过使用关键字array来要求该参数必须是一个数组。不过，这种方式只对类和数组有效，不适用于其他类型的变量。例如，我们不能将参数限定为字符串或者整数。

如果想要求AddressBook类中add()方法的第一个参数是一个Person类的技巧，可以这样：

```
class AddressBook {

    public function add(Person $person) {
        // 把 $person 添加到地址簿
    }
}
```

之后，如果你在调用add()时传递的参数是一个字符串，你会看到一个严重的错误：

```
$book = new AddressBook;

$person = 'Rasmus Lerdorf';

$book->add($person);
PHP Fatal error:  Argument 1 must be an object of class Person in...
```

把一个类型提示放到函数声明的第一个参数的位置上相当于在函数中添加了以下PHP代码：

```
public function add($person) {
        if (!($person instanceof Person)) {
                die("Argument 1 must be an instance of Person");
        }
}
```

其中的instanceof操作符用于检查一个对象是否是一个特定类的技巧。这些代码保证了$person是Person类的技巧。

PHP 4中没有instanceof操作符，要完成相同的检查必须使用is_a()函数 —— 这个函数在PHP 5中已经不建议使用了。

当一个对象是要比较的类的子类的技巧时，instanceof操作符也会返回true。例如：

```
class Person { /* ... */ }

class Kid extends Person { /* ... */ }

$kid = new Kid;

if ($kid instanceof Person) {
    print "Kids are people, too.\n";
}
```

```
Kids are people, too.
```

最后，还可以用instanceof来检测一个类是否实现了一个特定的接口：

```
interface Nameable {
    public function getName();
    public function setName($name);
}

class Book implements Nameable {
    private $name;

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        return $this->name = $name;
    }
}

$book = new Book;
if ($book instanceof Book) {
    print "You can name a Book.\n";
}
```

```
You can name a Book
```

类型提示具有类似于把API文档直接整合到类中的作用。如果你在一个类的构造器中发现它需要一个Event类型的参数，实际上就已经知道这个方法的用途了。而且，你也知道其中代码和“文档”必须始终处于同步状态，因为它们是直接被“烙”在了类的定义里的。

在接口的定义中同样也可以使用类型提示，借此可以对接口加诸更详细的说明。

但是，类型提示的确也需要以降低灵活性为代价。因为没有办法允许一个参数接受超过两种类型的对象，所以这就相当于给设计对象的层次施加了限制。

同样地，对于违反类型提示的处罚也是相当严厉的 —— 脚本会因为一个严重的错误而异常中断。在web环境下，可能你会希望能对此类错误拥有更多的控制权，并且以更妥善的方式来处理这些错误。

还有一点与其他语言的区别，就是不能在返回值上使用类型提示，因此也就不能限制一个函数必须返回一个对象或者其他特定类型的值。

参见

类型提示的文档（*http://www.php.net/manual/language.oop5.typehinting.php*）和instanceof()函数的文档（*http://www.php.net/manual/language.operators.type.php*）。

7.21 在对象技巧化期间自动地加载类文件

问题

你不想在每个页面中都包含全部的类定义。而是想动态地加载那些与相应页面有关的部分。

方案

使用_ _autoload()魔术方法：

```
function_ _autoload($class_name) {
    include "$class_name.php";
}

$person = new Person;
```

讨论

在正常情况下，当我们要技巧化一个没有定义的类时，PHP会因为一个严重错误而停止运行，因为它找不到要技巧化的类。因此，我们通常会在一个页面中加载所有可能用到的类，而不管这些类是否全会都被调用。

这样，由于PHP要解析所有的类 —— 即使用不到，因此就产生了增加处理时间的负面后果。有一个解决方案就是使用_ _autoload()方法动态加载缺少的代码，该方法会在技巧化未定义的类时被调用。

下面是如何在脚本中包含所有需要的类的示例：

```
function _ _autoload($class_name) {
    include "$class_name.php";
}

$person = new Person;
```

其中，_ _autoload()函数接受一个唯一的类名作为参数。在这个例子中，将*.php*扩展名添加到了$class_name后面，并尝试包含一个基于$class_name的文件。这样，当我们技巧化一个new Person时，它就会在包含路径中查找并加载*Person.php*文件。

如果_ _autoload()不能成功地加载要技巧化的类的定义，PHP也会和没有自动加载的情况下，它找不到相应的类定义一样，因一个严重错误而中断。

如果你采用的是以下划线分隔单词的、PEAR风格的命名约定来反映文件的层次结构，可以使用例7-41中的代码。

例7-41：使用PEAR命名约定自动加载类

```
Function _ _ autoload($package_name) {
    // 按下划线分割
    $folders = split('_', $package_name);
    // 基于目录结构重新组合
    // 使用 DIRECTORY_SEPARATOR 常量以适应所有平台
    $path    =  join(DIRECTORY_SEPARATOR, $folders);
    // 添加扩展名
    $path   .= '.php';

    include $path;
}
```

有了例7-41中代码，就可以像下面这样来使用了：

```
$person = new Animals_Person;
```

如果上面的类没有定义，Animals_Person就会被传递到_ _autoload()方法中。然后，该方法会按照下划线（_）来分割这个类名，并使用DIRECTORY_SEPARATOR再结合起来。结果在Unix平台上会返回字符串Animals/Person（在Windows平台上则返回Animals\Person）。

接着，再加上*.php*扩展名，最后将文件*Animals/Person.php*包含进来以借调用。

虽然使用_ _autoload()会稍微增加一些加载类的时间，但对每个类它只加载一次。也就是说，多次技巧化同一个类不会导致多次调用_ _autoload()方法。

在使用_ _autoload()之前，一定要确保打开、读取以及关闭多个必需的文件，确实不比额外解析不用的类更费时间。

在特定的情况下，如果你使用了一种opcode缓存技术，比如APC或Zend加速器，使用_ _autoload()和include_once可能会导致性能损失。因此，为了保证最好的效果，应该在脚本的一开始就包含所有文件，并确保不会重复包含同一个文件。

参见

自动加载的文档（*http://www.php.net/manual/language.oop5.autoload.php*）。

7.22 动态技巧化一个对象

问题

你想技巧化一个对象技巧，但只能在代码执行中才能知道类名。例如，你想要通过创建一个属于某种特殊语种的技巧来本地化一个网站。但是，在页面被请求之前，你无法知晓要使用什么语种。

方案

使用一个变量作为类名：

```
$language = $_REQUEST['language'];
$valid_langs = array('en_US' => 'US English',
                     'en_UK' => 'British English',
                     'es_US' => 'US Spanish',
                     'fr_CA' => 'Canadian French');

if (isset($valid_langs[$language]) && class_exists($language)) {
    $lang = new $language;
}
```

讨论

有时候你可能在代码运行时也不知道想要技巧化的类的完整名称，但却知道其名称的一部分。比如说，要体现类层次所表示的伪名称空间，可以在所有类名前面加上一串前导字符作为前缀。这就是为什么本书中经常使用pc_来表示*PHP Cookbook*及PEAR将Net_放在所有与网络相关的类之前的原因。

然而，虽然下面是合法的PHP代码：

```
$class_name = 'Net_Ping';
$class = new $class_name;                // new Net_Ping
```

但下面的代码却不合法：

```
$partial_class_name = 'Ping';
$class = new "Net_$partial_class_name"; // new Net_Ping
```

不过，下面这些则又是合法的：

```
$partial_class_name = 'Ping';
$class_prefix = 'Net_';

$class_name = "$class_prefix$partial_class_name";
$class = new $class_name;                // new Net_Ping
```

也就是说，如果类名是通过变量连接的方式构成的，那么不能在定义类名的同时技巧化该类。不过，为了使用简单的变量名，合法的解决方案应该是先行连接、定义类名，再技巧化。

参见

Class_exists()函数的文档（*http://www.php.net/class-exists*）。

7.23 编程：whereis

虽然类似phpDocumentor这样的工具能够提供有关一系列类的相当详尽的信息，但是，如果能将在一组文件中定义的全部函数和方法快速地罗列出来，形成一个简单的列表相信一定也是非常有用的。

例7-42中的程序通过循环遍历一组文件，逐一地包含它们并使用Reflection类来收集与每个类有关的信息。当主要列表编译完成后，所有函数和方法就已经按照字母顺序保存好并输出了。

例7-42：whereis

```
<?php
if ($argc < 2) {
    print "$argv[0]: classes1.php [, ...]\n";
    exit;
}

// 包含这些文件
```

```
foreach (array_slice($argv, 1) as $filename) {
    include_once $filename;
}

// 从类开始收集方法和函数的信息
$methods = array();
foreach (get_declared_classes() as $class) {
    $r = new ReflectionClass($class);
    // 排除内置类
    if ($r->isUserDefined()) {
        foreach ($r->getMethods() as $method) {
            // 排除继承的方法
            if ($method->getDeclaringClass()->getName() == $class) {
                $signature = "$class::" . $method->getName();
                $methods[$signature] = $method;
            }
        }
    }
}

// 接下来添加函数
$functions = array();
$defined_functions = get_defined_functions();
foreach ($defined_functions['user'] as $function) {
    $functions[$function] = new ReflectionFunction($function);
}

// 按类的字母顺序对方法进行排序
function sort_methods($a, $b) {
    list ($a_class, $a_method) = explode('::', $a);
    list ($b_class, $b_method) = explode('::', $b);

    if ($cmp = strcasecmp($a_class, $b_class)) {
        return $cmp;
    }

    return strcasecmp($a_method, $b_method);
}
uksort($methods, 'sort_methods');

// 按字母顺序对函数进行排序
// 虽然不复杂，但也不要忘记从列表中删除方法排序函数
unset($functions['sort_methods']);
// 排序
ksort($functions);

// 输出信息
foreach (array_merge($functions, $methods) as $name => $reflect) {
    $file = $reflect->getFileName();
    $line = $reflect->getStartLine();

    printf ("%-25s | %-40s | %6d\n", "$name()", $file, $line);
}
?>
```

以上代码同时使用了Reflection类和两个PHP函数 —— get_declared_classes()和get_declared_functions() —— 这两个函数与Reflection类无关，但却能帮助分析类。

注意，过滤掉所有内置的PHP类和函数非常重要。否则，最终的报告里你自己代码中的信息会减少，而你PHP安装包中的信息将充斥其中。这是通过两种不同的方式来处理的。由于get_declared_classes()不会区分用户定义的类和内部类，所以代码调用了ReflectionClass::isUserDefined()方法来检查。另一方面，调用get_defined_function()实际上就是为了计算这一点，并把信息放到用户数组元素中。

为了比较容易查看排序后列表的输出，脚本对方法和函数数组进行了排序。由于不同的类可能会定义相同的方法，所以我们需要使用一个用户定义的排序方法 —— sort_methods()，它会先比较两个方法的类名，然后再比较方法名。

当对数据排完序后，剩下的遍历合并的数组、收集文件名和起始行号，以及输出报告等等步骤就相对容易多了。

下面是通过脚本来运行这个PEAR HTTP类的结果：

```
HTTP::Date()                | /usr/lib/php/HTTP.php                |    38
HTTP::head()                | /usr/lib/php/HTTP.php                |   144
HTTP::negotiateLanguage()   | /usr/lib/php/HTTP.php                |    77
HTTP::redirect()            | /usr/lib/php/HTTP.php                |   186
```

第8章

Web基础

8.0 概述

学习基于Web的编程可能正是你读本书的原因。事实上，Web编程也是编写PHP第一版的初衷和PHP能有今天这么流行的原因。通过PHP，使得编写能做任何事情的动态Web程序易如反掌。在其他几章中讨论的PHP所具有的各种能力，如图像处理、正则表达式、访问数据库及文件I/O等，都属于Web编程的范畴。而本章则集中讨论了一些能够保证Web程序设计更加健壮的、与Web相关的概念和组织方面的话题。

技巧8.1、8.2和8.3展示的是如何设置、读取和删除cookie。所谓cookie，就是浏览器按照服务器的指令随同其发起的请求共同发送的一小串文本字符。在默认的情况下，HTTP请求并非“有状态”的 —— 即每一个请求之间都是独立无关的。不过，cookie可以将同一个用户不同的请求连接起来。这样，就能实现诸如购物车或者跟踪用户的搜索记录之类的功能了。

技巧8.4展示了如何把用户的请求重新定向到一个不同的网页上面去。而检测用户浏览器特性的部分则放在了技巧8.5中介绍。技巧8.6详细说明了如何构造URL，使其包含get查询字符串，包含对特殊字符适当的编码及HTML实体的处理等。同样地，技巧8.7中介绍了如何从一个提交的post请求的主体中读取数据的细节。技巧8.8讨论了一个常见Web格式化的需求 —— 区别显示HTML表格行，如表中各行交替采取不同的颜色或者样式。

接下来的两个技巧示范了如何运用身份认证，来通过密码等手段保护网页。PHP所特有的处理HTTP基本身份认证方面的功能在技巧8.9中讨论。而有的时候，像技巧8.10中所展示的那样使用自己的身份认证方式可能也不错。

接下来三个技巧涉及到输出控制。技巧8.11介绍了如何强制把输出发送到浏览器。技巧8.12则解释了输出的缓冲功能。输出缓冲可以让我们捕获输出并控制其输出的方式，或者将输出延迟到对整个页面处理完成之后。有关输出自动压缩的内容放在第8.13中介绍。

再往后的两个技巧展示的是如何同外部变量 —— 环境变量和PHP配置设置进行交互。技巧8.14和技巧8.15围绕环境变量进行讨论。如果你使用Apache作为Web服务器，那么你还可以运用技巧8.16中的技术实现PHP程序与不同的Apache模块之间的通信。

本章还包括了两个编程技巧，用来示范前面提及的一些编程思想。其中，技巧8.17中的程序会为每个新用户发送一封带有自定义链接的电子邮件，以便激活用户的账户。如果用户在收到邮件一周内没有访问相应的链接，就会删除该账户。技巧8.18是一个小型维基（Wiki）系统的例子。该系统可以使网站中的每个页面都能在用户的浏览器中进行编辑。

8.1 设置 Cookie

问题

你想要设置一个cookie，以便让你的网站可以识别出来自同一个浏览器的后续请求。

方案

调用setcookie()，并传递一对cookie的名和值，如例8-1所示。

例8-1：设置cookie

```
<?php
setcookie('flavor','chocolate chip');
?>
```

讨论

Cookie是随着HTTP头部一起发送的，所以如果没有使用输出缓冲，那么必须在生成任何输出之前调用setcookie()。

还可以给setcookie()传递额外的参数以控制cookie的行为。其中，setcookie()的第三个参数表示过期时间，使用时间戳形式。例如，例8-2中设置的过期时间为GMT时间2004年12月3日中午。

例8-2：为cookie设置过期时间

```
<?php
setcookie('flavor','chocolate chip',1259841600);
?>
```

如果setcookie()的第三个参数丢失了（或留空），那么cookie就会在浏览器关闭的时候过期。并且，许多系统都不能处理过期时间大于2147483647的cookie，因为那是32位系统所能处理的最大纪元时间戳，相关主题在第3章的内容提要部分已经讨论过了。

setcookie()的第四个参数是一个路径。只有在被请求的页面路径以指定的字符串开头的情况下，cookie才会被发送回服务器。例如，在例8-3中设置的cookie，只有当页面的路径以*/products/*开头时才会发回。

例8-3：在设置cookie时附加路径限制

```
<?php
setcookie('flavor','chocolate chip','','/products/');
?>
```

在例8-3中，虽然设置cookie的这个页面的URL不一定是以*/products/*开头，但一定是在请求以*/products/*开头的页面时才会发回cookie。

setcookie()的第五个参数是一个域名。只有当被请求页面的主机名以指定的域名结尾时，cookie才会被发送回服务器。例如，在例8-4中设置的第一个cookie，会被发回到*example.com*域名下所有的主机上，而第二个cookie则只有在请求*jeannie.example.com*主机时才会被发回。

例8-4：在设置cookie时附加域名限制

```
<?php
setcookie('flavor','chocolate chip','','','.example.com');
setcookie('flavor','chocolate chip','','','jeannie.example.com');
?>
```

如果上面设置的第一个cookie中只使用了*example.com*而不是*.example.com*，那么这个cookie只能发回到唯一的主机*example.com*（而不是*www.example.com*或*jeannie.example.com*）。如果在调用setcookie()时没有指定域名，那么浏览器则只有在请求的主机名与设置cookie的主机名相同时，才会发回cookie。

setcookie()的最后一个可选参数是一个安全标记，如果将其设为true，表示浏览器只能以SSL连接的方式发送cookie。如果cookie中包含敏感的信息，使用这个选项会有一些用处，但是不要忘了cookie中的数据可能仍然是以未加密的纯文本形式保存在用户的计算机上。

此外，不同的浏览器会以稍有差别的方式处理 cookie，特别是在面临考虑如何严格地匹配路径和域名字符串，以及如何决定同名cookie的优先级次序等问题时。setcookie()函数的在线手册中已经对面临这些问题时浏览器的不同处理方式进行了说明。

参见

技巧8.2中关于读取cookie值的介绍。技巧8.3中关于如何删除cookie的介绍。技巧8.12中介绍的有关输出缓冲的内容。setcookie()函数的文档（*http://www.php.net/setcookie*）。一个扩展的cookie规范 —— RFC2965的详细内容（*http://www.faqs.org/rfcs/rfc2965.html.*）。

8.2 读取Cookie的值

问题

你想读取先前设置的cookie的值。

方案

在自动全局数组$_COOKIE中查询，具体步骤如例8-5所示。

例8-5：读取cookie的值。

```
<?php
if (isset($_COOKIE['flavor'])) {
    print "You ate a {$_COOKIE['flavor']} cookie.";
}
?>
```

讨论

在通过一个请求设置cookie的时候，该cookie的值在$_COOKIE中是查询不到的。换句话说，setcookie()函数不会修改$_COOKIE的值。不过，在后续的请求中，每个被发回到服务器的cookie都会保存在$_COOKIE中。如果register_globals的值是on，那么cookie的值也会同时赋给全局变量。

当浏览器把一个cookie发送回服务器的时候，所发送的只是cookie的值。我们无法通过$_COOKIE访问那个cookie的域名、路径、过期时间或者安全状态等信息，因为浏览器不会把这些信息发送回服务器。

如果想输出某个特定请求状态下所有cookie的名字和值，可以像例8-6所示的那样循环遍历$_COOKIE数组。

例8-6：读取所有cookie的值

```
<?php
foreach ($_COOKIE as $cookie_name => $cookie_value) {
    print "$cookie_name = $cookie_value <br/>";
}
?>
```

参见

技巧8.1中有关设置cookie的内容。技巧8.3中有关删除cookie的内容。技巧8.12中有关输出缓冲的介绍。技巧9.15中有关register_globals的信息。

8.3 删除Cookie

问题

你想要删除一个cookie，这样浏览器就不会再把它发送回服务器了。例如，你正在使用cookie跟踪一个用户是否已经登录到你的网站上，而此时有一个用户选择了注销操作。

方案

在调用setcookie()时在第二个参数的位置使用空值，并传递一个已经过期的时间参数，如例8-7所示。

例8-7：删除一个cookie

```
<?php
setcookie('flavor','',1);
?>
```

讨论

在你的服务器和用户的计算机时钟不同步的情况下，最好设置一个已经过期很长的过期时间。例如，如果你的服务器时钟是下午3:06，用户的计算机时钟是下午3:02，而cookie的过期时间是下午3:05。那么，即使对于服务器而言该cookie已经过期了，但实际上这个cookie也不会被用户的浏览器删除。

为删除cookie调用setcookie()函数所使用的参数，必须与在设置那个cookie时调用setcookie()函数所使用的参数（除了值和过期时间之外）相同，即如果必要的话还要包含相同的路径、域名和安全标记。

参见

技巧8.1中有关设置cookie的内容。技巧8.2中有关读取cookie值的内容。技巧8.12中关于输出缓冲的介绍。setcookie()函数的文档（*http://www.php.net/setcookie*）。

8.4 重定向到一个不同的位置

问题

你想要自动把一个用户的请求转发到一个新URL。例如，在成功地保存了表单数据之后，你想要把用户重新定位到一个确认数据已经保存的页面上。

方案

在输出被打印到屏幕之前，使用header()发送一个包含新URL的Location头部信息，然后再调用exit()函数，以防止输出。例8-8演示了这些步骤。

例8-8：重定向到一个不同的位置

```
<?php
header('Location: http://www.example.com/confirm.html');
exit();
?>
```

讨论

如果你想向新页面中传递变量，可以将要传递的变量包含在URL的查询字符串中，如例8-9所示。

例8-9：带有查询字符串的重定向

```
<?php
header('Location: http://www.example.com/?monkey=turtle');
exit();
?>
```

重定向URL应该包含协议和主机名，而不应该只用路径名。例8-10中展示了一个推荐的和一个不推荐的Loaction头部信息。

例8-10：好的和不好的Location头部信息

```
<?php
// 好的重定向
header('Location: http://www.example.com/catalog/food/pemmican.php');
```

```
// 不好的重定向
header('Location: /catalog/food/pemmican.php');
?>
```


将用户重定向到的目标URL可以通过get实现。不能通过post把用户重定向到取得的URL上。但是，如果使用JavaScript，则可以通过生成一个自动提交的表单（通过post）来模仿通过post重定向。当一个（启用JavaScript的）浏览器加载完成例8-11中的页面时，它会立即以post方法提交其中的表单。

例8-11：通过post表单实现重定向

```
<html>
  <body onload="document.getElementById('redirectForm').submit()">
    <form id='redirectForm' method='POST' action='/done.html'>
      <input type='hidden' name='status' value='complete'/>
      <input type='hidden' name='id' value='Ou812'/>
      <input type='submit' value='Please Click Here To Continue'/>
    </form>
  </body>
</html>
```

例8-11中的表单的id值为redirectForm，因此<body/>元素的onload属性中的代码会提交这个表单。如果浏览器没有启用对JavaScript的支持则onload动作不会被触发。在那种情况下，用户将会看到一个显示为“Please Click Here To Continue”的按钮。

参见

header()函数的文档（*http://www.php.net/header*）。

8.5 检测不同的浏览器

问题

你想基于用户所用浏览器的能力生成相应的内容。

方案

根据get_browser()返回的对象来判断浏览器的能力，如例8-12所示。

例8-12：获取浏览器信息

```
<?php
$browser = get_browser();
if ($browser->frames) {
    // 输出基于帧的布局
```

```
} elseif ($browser->tables) {
    // 输出基于表格的布局
} else {
    // 输出其他布局
}
?>
```

讨论

get_browser()函数会检查环境变量（由web服务器设定）并将其与外部的浏览器能力文件进行比较。由于许可问题，PHP没有自带浏览器能力文件。在PHP FAQ的“获得（Obtaining） PHP”（*http://www.php.net/faq.obtaining*）一节中，给出了下载浏览器能力文件的地址：*http://www.garykeith.com/browsers/downloads.asp*。可以从该网站中下载到*php_browscap.ini*文件。

当下载到浏览器能力文件之后，可以通过把browscap配置指令设置为该文件的路径来告诉PHP到哪里找到这个文件。如果你把PHP当作一个CGI，那么可以像例8-13那样在*php.ini*中设置这个指令。

例8-13：在php.ini中设置browscap指令

```
browscap=/usr/local/lib/php_browscap.ini
```

表8-1中是get_browser()可以发现的一些能力。对于诸如javascript或cookie之类的用户可以配置的能力，get_browser()只能告诉你浏览器是否支持那些功能，而不会告诉你用户是否关闭了该功能。即，如果在支持JavaScript的浏览器中关闭了JavaScript，或者当浏览器提示要写入cookie时，用户拒绝接受cookie，get_browser()仍然会告诉你该浏览器支持这些功能。

表8-1：浏览器能力对象的属性

属性	描述
platform	浏览器运行其上的操作系统（例如，Windows、Macintosh、Unix、Win32、Linux、MacPPC）
version	浏览器的完整版本号（例如，5.0、3.5、6.0b2）。
majorver	浏览器的主版本号（例如，5、3、6）。
minorver	浏览器的次要版本号（例如，0、5、02）。
frames	如果浏览器支持框架，值为1
tables	如果浏览器支持表格，值为1
cookies	如果浏览器支持 cookie，值为1

表8-1：浏览器能力对象的属性（续）

属性	描述
backgroundsounds	如果浏览器支持以<embed>或<bgsound>添加的背景声音，值为1
vbscript	如果浏览器支持VBScript，值为1
javascript	如果浏览器支持JavaScript，值为1
javaapplets	如果浏览器支持Java applets，值为1
activexcontrols	如果浏览器支持ActiveX 控件，值为1

参见

get_browser()函数的文档（*http://www.php.net/get-browser*）。

8.6 建立查询字符串

问题

你想要构造一个在其查询字符串中包含名/值对的链接。

方案

使用http_build_query()函数，如例8-14所示。

例8-14：建立一个查询字符串

```
<?php
$vars = array('name' => 'Oscar the Grouch',
              'color' => 'green',
              'favorite_punctuation' => '#');
$query_string = http_build_query($vars);
$url = '/muppet/select.php?' . $query_string;
?>
```

讨论

例8-14中建立的URL是：

```
/muppet/select.php?name=Oscar+the+Grouch&color=green&favorite_punctuation=%23
```

在其查询字符串中，空格被编码为“+”，像“#”这样的特殊字符也按十六进制编码为“%23”（#的ASCII值是35，而35是十六进制的23）。

虽然http_build_query()通过自动编码的确能够防止变量名或值中的特殊字符破坏构建的URL，但如果变量名是以HTML实体作为开头，那么问题仍然不可避免。考虑下面这个为取得一种立体声音响系统信息的URL片断：

```
/stereo.php?speakers=12&cdplayer=52&amp=10
```

&符号的HTML实体是&，所以浏览器可能会把这段URL解释为：

```
/stereo.php?speakers=12&cdplayer=52&=10
```

为了避免嵌入的实体破坏URL，我们有三种选择。第一种是选择不会与实体混淆的变量名，如_amp而不是amp。第二种是在输出URL之前把带有HTML实体等价物的字符转换成相应的实体。使用htmlentities()：

```
$url = '/muppet/select.php?' . htmlentities($query_string);
```

结果URL是：

```
/muppet/select.php?name=Oscar+the+Grouch&amp;color=green&amp;favorite_punctuation=%23
```

第三种选择就是通过将arg_separator.input配置指令设置为&来把分隔参数的符号由&改为&。设置该指令以后，http_build_query()函数就会用&来组合不同的name=value形式的参数了：

```
/muppet/select.php?name=Oscar+the+Grouch&amp;color=green&amp;favorite_punctuation=%23
```

参见

http_build_query()函数的文档（*http://www.php.net/http_build_query*）和htmlentities()函数的文档（*http://www.php.net/htmlentities*）。

8.7 读取Post请求的主体

问题

你想直接访问一个post请求的主体，而不仅是使用PHP解析之后放在$_POST中的数据。例如，你想要处理一个作为Web 服务请求发送过来的XML文档。

方案

如例8-15所示的那样，从php://input流中读取。

例8-15：读取post请求的主体。

```
<?php
$body = file_get_contents('php://input');
?>
```

讨论

如果你只需要访问提交的表单变量，那么使用自动全局数组$_POST会更合适，但它无法满足你对原始数据的需要，更不可能通过它来访问整个请求主体。这就是为什么会有php://input流的原因。可以通过file_get_contents()来读取php://input流中的内容，而如果你想得到的是一个更大的请求主体，那么可以通过fread()以块为单位来进行读取。

如果配置指令always_populate_raw_post_data设置为on，那么原始的post数据也会被保存到全局变量$HTTP_RAW_POST_DATA中。但是，从编写最大化可移植代码的角度来考虑，应该使用php://input流 —— 因为它即使在always_populate_raw_post_data指令设置为off时仍然是有效的。

参见

php://input流的文档（*http://www.php.net/wrappers*）和always_populate_raw_post_data指令的文档（*http://www.php.net/ini.core#ini.always-populate-raw-post-data*）。

8.8 生成具有交替样式的HTML表格

问题

你想通过一个表格来显示信息，而且想让表格的行交替具有不同的外观。例如，让表格中的偶数行显示白色背景，而让奇数行显示灰色背景。

方案

在生成HTML表格时，交替使用两种不同的CSS样式。例8-16对取自数据库的数据采用了这一技术。

例8-16：生成具有交替行样式的HTML表格。

```
<style tye="text/css">
.even-row{
    background:white;
```

```
}
.odd-row{
     background:gray;
}
</style>
<table>
<tr><th>Quantity</th><th>Ingredient</th></tr>
<?php
$styles=array('even-row','odd-row');
$db=new PDO('sqlite:altrow.db');
foreach($db->query('SELECT quantity,ingredient FROM ingredients')as $i=>$row){?>
<tr class="<?php echo $styles[$i%2];?>">
     <td><?php echo htmlentities($row['quantity'])?></td>
     <td><?php echo htmlentities($row['ingredient'])?></td></tr>
<?php}?>
</table>
```

讨论

例8-16中的代码之所以如此简洁，关键在于使用了CSS之类名数组$styles和PHP的“余数”操作符%。取余操作符会返回两个数相除之后的余数。而在用2除某个数时（在本例中，是保存在$i结果集中的行数），余数非0即1。这样，就为交替地访问$styles数组中的第一和第二个元素提供了方便。

参见

PHP中算术操作符的文档：*http://www.php.net/language.operators.writhmetic*。

8.9 使用HTTP的基本或摘要认证

问题

你想使用PHP并通过密码来保护网站中的某一部分内容。而不是把密码保存到一个外部文件中，让Web服务器处理认证。你希望把验证密码的逻辑放在PHP程序中。

方案

全局变量$_SERVER['PHP_AUTH_USER']和$_SERVER['PHP_AUTH_PW']包含着用户提供的用户名和密码（如果有）。如果想要拒绝对一个页面的访问，可以如例8-17所示发送一个WWW-Authenticate头部信息将认证区域标识为状态码为401的响应。

例8-17：实施基本认证。

```
<?php
```

```
header('WWW-Authenticate: Basic realm="My Website"');
header('HTTP/1.0 401 Unauthorized');
echo "You need to enter a valid username and password.";
exit();
?>
```

讨论

当浏览器收到401头部信息后，会弹出一个要求输入用户名和密码的对话框。那些认证证书（即用户名和密码），如果被服务器接受，是与WWW-Authenticate头部的区域关联的。检查认证证书的代码必须在把任何输出发送到浏览器之前执行，因为服务器可以先发送头部信息。例如，你可以使用一个类似例8-18中pc_validate()的函数。

例8-18：pc_validate()函数。

```
<?php
function pc_validate($user,$pass) {
    /* 以适当的用户名和密码检查来替换，
       例如，检查一个数据库 */
    $users = array('david' => 'fadj&32',
                   'adam'  => '8HEj838');

    if (isset($users[$user]) && ($users[$user] == $pass)) {
        return true;
    } else {
        return false;
    }
}
?>
```

例8-19中显示了如何使用pc_validate()。

例8-19：使用一个验证函数。

```
<?php
if (! pc_validate($_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW'])) {
    header('WWW-Authenticate: Basic realm="My Website" ');
    header('HTTP/1.0 401 Unauthorized');
    echo "You need to enter a valid username and password.";
    exit;
}
?>
```

用适当的逻辑来代替pc_validate()函数中的内容，以判断用户是否键入了正确的密码。也可以修改区域字符串“My Website”及当用户在浏览器中点击认证对话框的“取消”按钮后，打印到屏幕上面的信息“You need to enter a valid username and password.”。

PHP 5.1.0及更高版除了支持基本（Basic）认证，还支持摘要（Digest）认证。使用基本认证时，用户名和密码在网络中基本上都是以明码发送的，只是进行了最低限度的Base64编

码。而使用摘要认证时，则永远不会将密码本身从浏览器发送到服务器。相反，发送的只是密码的散列码及其他一些值。这样就减少了网络通信被黑客程序捕获并重发的可能性。而对于摘要认证而言，安全性增加同时也意味着用于实现认证的代码要比仅仅比较一下密码复杂得多。例8-20中提供了一个按照RFC 2617计算摘要认证的函数。

例8-20：使用摘要（Digest）认证

```
<?php

    /* 以适当的用户名和密码检查来替换，
       例如，检查一个数据库 */
$users = array('david' => 'fadj&32',
               'adam'  => '8HEj838');
$realm = 'My website';

$username = pc_validate_digest($realm, $users);

// 如果提供的认证数据无效，则不会执行到此处
print "Hello, " . htmlentities($username);

function pc_validate_digest($realm, $users) {
    // 如果客户端未提供摘要信息，则认证失败
    if (! isset($_SERVER['PHP_AUTH_DIGEST'])) {
        pc_send_digest($realm);
    }
    // 如果无法解析摘要信息，则认证失败
    $username = pc_parse_digest($_SERVER['PHP_AUTH_DIGEST'], $realm, $users);
    if ($username === false) {
        pc_send_digest($realm);
    }
    // 返回摘要中指定的有效用户名
    return $username;
}

function pc_send_digest($realm) {
    header('HTTP/1.0 401 Unauthorized');
    $nonce = md5(uniqid());
    $opaque = md5($realm);
    header("WWW-Authenticate: Digest realm=\"$realm\" qop=\"auth\" ".
           "nonce=\"$nonce\" opaque=\"$opaque\"");
    echo "You need to enter a valid username and password.";
    exit;
}

function pc_parse_digest($digest, $realm, $users) {
    // 我们要在摘要头部信息中查找下列值：
    // username, uri, qop, cnonce, nc和response
    $digest_info = array();
    foreach (array('username','uri','nonce','cnonce','response') as $part) {
        // 分隔符可以是 ' 或 " 或无 (对于 qop 和nc而言)
        if (preg_match('/'.$part.'=([\? "]?)(.*?)\1/', $digest, $match)) {
            // 找到这一部分了，将其保存起来备用
            $digest_info[$part] = $match[2];
        } else {
```

```
            // 如果没有这一部分，说明摘要信息无法通过验证
            return false;
        }
    }
    // 确保提供了正确的qop
    if (preg_match('/qop=auth(,|$)/', $digest)) {
        $digest_info['qop'] = 'auth';
    } else {
        return false;
    }
    // 确保提供了有效的 nonce count
    if (preg_match('/nc=([0-9a-f]{8})(,|$)/', $digest, $match)) {
        $digest_info['nc'] = $match[1];
    } else {
        return false;
    }

    // 既然已经验证完了摘要头部信息中所有必要的值
    // 那么就继续执行必需的算法指令以保证提供信息的正确性
    //
    // 这些计算在RFC 2617的第3.2.2、3.2.2.1和3.2.2.2中有说明
    // 采用MD5算法
     $A1 = $digest_info['username'] . ':' . $realm . ':' . $users[$digest_info['username']];
    // qop 等于 'auth'
    $A2 = $_SERVER['REQUEST_METHOD'] . ':' . $digest_info['uri'];
    $request_digest = md5(implode(':', array(md5($A1), $digest_info['nonce'],$digest_info['nc'],
    $digest_info['cnonce'], $digest_info['qop'], md5($A2))));

    // 发送的信息与我们计算的值是否一致?
    if ($request_digest != $digest_info['response']) {
        return false;
    }

    // 万事大吉，返回用户名
    return $digest_info['username'];
}
?>
```

如果你没有使用PHP 5.1.0或更高版本，而是将PHP作为Apache的一个模块，那么你可以通过像HTTPDigest类这样的代码来实现摘要认证。HTTPDigest类的作者是Paul James，相关资料的网址是*http://www.peej.co.uk/projects/phphttpdigest.html*。

如果是将PHP作为一个CGI程序，那么HTTP基本认证和摘要认证都将无法使用。如果你不能将PHP作为一个服务器模块，那么可以采用cookie认证的方式，有关这一点在技巧8.10中讨论。

HTTP认证的另一个问题是它没有提供让用户简单注销的方式 —— 除了关闭浏览器。在PHP的在线手册中给出了一些有关注销方法的建议，其中按照不同的服务器与浏览器组合划分出很多不同的级别，详见*http://www.php.net/features.http-auth*。

不过，还有一个直截了当的方法，即在一定的时间间隔后强制注销——在区域字符串中包含时间的计算。每当浏览器检查同一个区域中的证书时，它都会使用相同的用户名和密码组合。通过修改区域的名称，浏览器就会强制用户提供新的证书。例8-21使用了基本认证方式，并在每天午夜强制注销。

例8-21：基本认证搭配强制注销

```
<?php
if (! pc_validate($_SERVER['PHP_AUTH_USER'],$_SERVER['PHP_AUTH_PW'])) {
    $realm = 'My Website for '.date('Y-m-d');
    header('WWW-Authenticate: Basic realm="'.$realm.'"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You need to enter a valid username and password.";
    exit;
}
?>
```

还可以通过保存用户登录时间或者访问一个受保护的页面来根据用户指定超时时间，而不用修改区域的名称。例8-22中的pc_validate2()函数将登录时间保存到数据库中，并在用户最后一次请求受保护的页面15分钟后强制注销。

例8-22：pc_validate2()函数

```
<?php
function pc_validate2($user,$pass) {
    $safe_user = strtr(addslashes($user),array('_' => '\_', '%' => '\%'));
    $r = mysql_query("SELECT password,last_access
                      FROM users WHERE user LIKE '$safe_user' ");

    if (mysql_numrows($r) == 1) {
        $ob = mysql_fetch_object($r);
        if ($ob->password == $pass) {
            $now = time();
            if (($now - $ob->last_access) > (15 * 60)) {
                return false;
            } else {
                // 更新最后访问时间
                mysql_query("UPDATE users SET last_access = NOW()
                             WHERE user LIKE '$safe_user' ");
               return true;
            }
        }
    } else {
        return false;
    }
}
```

参见

技巧8.10。PHP在线手册中HTTP认证部分（*http://www.php.net/features.http-auth*）。

8.10 使用Cookie认证

问题

你想要更多地控制用户登录的过程，比如给出自己的登录表单。

方案

把认证状态保存到cookie中或者作为某个session的一部分。当用户登录成功时，把他的用户名保存到一个cookie中，同时也包含该用户名的散列码和一个加密字，使用户不能仅凭其中的用户名就可以通过认证，具体过程见例8-23。

例8-23：使用cookie认证

```
<?php
$secret_word = 'if i ate spinach';
if (pc_validate($_POST['username'],$_POST['password'])) {
    setcookie('login',
              $_POST['username'].','.md5($_POST['username'].$secret_word));
}
?>
```

讨论

在使用cookie认证时，必须使用你自己的登录表单，例8-24中就提供了一个这样的表单。

例8-24：使用cookie认证的表单示例

```
<form method="POST" action="login.php">
Username: <input type="text" name="username"> <br>
Password: <input type="password" name="password"> <br>
<input type="submit" value="Log In">
</form>
```

可以使用例8-18中那个pc_validate()函数来验证用户名和密码。唯一的区别在于所传递的认证证书分别是$_POST['username']和$_POST['password']，而不是$_SERVER['PHP_AUTH_USER']和$_SERVER['PHP_AUTH_PW']。如果密码检验通过，就发回一个包含用户名和该用户名散列码及一个加密字的cookie。其中的散列码用以防止用户通过发送一个只包含用户名的cookie来骗过登录验证。

当用户登录之后，只需要用一个页面来验证发回的有效登录cookie，并根据它来决定为登录用户提供个性化的内容。例8-25提供了一种实现方法。

例8-25：验证登录cookie

```
<?php
unset($username);
if ($_COOKIE['login']) {
    list($c_username,$cookie_hash) = split(',',$_COOKIE['login']);
    if (md5($c_username.$secret_word) == $cookie_hash) {
        $username = $c_username;
    } else {
        print "You have sent a bad cookie.";
    }
}

if ($username) {
    print "Welcome, $username.";
} else {
    print "Welcome, anonymous user.";
}
?>
```

如果启用了内置的session支持，那么也可以把用户名及散列码添加到session中，而不用单独发送cookie。在有人登录时，就可以在session中附加一个额外的变量，而不用再发送cookie，如例8-26所示。

例8-26：在session中保存登录信息

```
<?php
if (pc_validate($_POST['username'],$_POST['password'])) {
    $_SESSION['login'] =
        $_POST['username'].','.md5($_POST['username'].$secret_word));
}
?>
```

例8-27中的验证代码几乎是相同的 —— 只不过是用$_SESSION代替了$_COOKIE。

例8-27：验证session信息

```
<?php
unset($username);
if (isset($_SESSION['login'])) {
    list($c_username,$cookie_hash) = explode(',',$_SESSION['login']);
    if (md5($c_username.$secret_word) == $cookie_hash) {
        $username = $c_username;
    } else {
        print "You have tampered with your session.";
    }
}
?>
```

使用 cookie或session认证而不是HTTP基本认证，可以使注销用户更容易 —— 只需要删除他们的登录cookie或者从他们的session中去掉相应的登录变量即可。把认证信息保存到session中的另一个好处是，可以在用户登录之前或者注销以后连接到用户浏览器，记

录他们浏览器的活动。而使用HTTP基本认证，则无法通过一个用户名要求同一个用户在提供用户名之前发送一个请求。从同一个IP地址寻找更多的请求是一种错误倾向，特别是当用户隐藏在防火墙或者代理服务器后面的情况下。如果使用的是session，那么你就可以通过修改登录程序，通过如例8-28所示的代码来记录session ID与用户名之间的连接。

例8-28：连接注销和登录的使用

```
<?php
if (pc_validate($_POST['username'],$_POST['password'])) {
    $_SESSION['login'] =
        $_POST['username'].','.md5($_POST['username'].$secret_word));
    error_log('Session id '.session_id().' log in as '.$_REQUEST['username']);
}
?>
```

例8-28会向错误日志中写入一条信息，但它只是尽可能简单地在数据库中记录这些信息，你可以将这些信息用于网站可用性及流量分析。

使用session ID的一个危险在于session可能被劫持。如果Alice猜到了Bob的session ID，她就可以化装成Bob混入web服务器中。在session模块中，有两个可选的配置指令能够使session ID更难猜测。session.entropy_file指令包含着指向一个驱动器或者一个文件的路径，这些路径都是随机生成的，例如，*/dev/random*或*/dev/urandom*。而session.entropy_length指令保存着在创建session ID时从entropy文件中读取字节数。

无论session ID怎样难猜，只要它在服务器和用户的浏览器之间采取明码文本形式传送，就有可能被盗取。HTTP基本认证也存在这个问题。就此问题可以参考技巧18.13中介绍的如何使用SSL预防网络测错的讨论。

参见

技巧8.9，技巧20.9中讨论的记录错误的内容。技巧18.9中介绍的通过散列码验证数据的内容。setcookie()函数的文档（*http://www.php.net/setcookie*）和MD5函数的文档（*http://www.php.net/md5*）。

8.11 把输出冲刷（Flushing）到浏览器

问题

你想要把输出强制发送到浏览器。例如，在执行一个缓慢的数据库查询之前，你想给用户一个正在更新的提示。

方案

使用flush()，如例8-29所示。

例8-29：把输出冲刷（Flushing）到浏览器

```
<?php
print 'Finding identical snowflakes...';
flush();
$sth = $dbh->query(
    'SELECT shape,COUNT(*) AS c FROM snowflakes GROUP BY shape HAVING c > 1');
?>
```

讨论

flush()函数会把PHP内部缓冲的所有输出发送到web服务器，而web服务会在数据到达浏览器之前提供它自己的内部延迟缓冲。此外，有些浏览器在接收数据后并不会马上显示出来，比如某些版本的Internet Explorer在接收的数据达不到256字节的情况下，不会显示页面。如果想强制让IE立即显示内容，可以像例8-30那样在输出页面的开始处打印一些空格符。

例8-30：强制IE立即显示内容

```
<?php
print str_repeat(' ',300);
print 'Finding identical snowflakes...';
flush();
$sth = $dbh->query(
    'SELECT shape,COUNT(*) AS c FROM snowflakes GROUP BY shape HAVING c > 1');
?>
```

参见

技巧23.13。flush()函数的文档（*http://www.php.net/flush*）。

8.12 缓冲到浏览器的输出

问题

你想在发送完头部或cookie信息之后再开始生成输出。

方案

在页面的顶部调用ob_start()，而在页面的底部调用ob_end_flush()。然后，就可混合使

用生成输出和发送头部信息的命令了。但此时，在调用ob_end_flush()之前是不会发送输出的。这个过程可以用例8-31来说明。

例8-31：缓冲输出

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

I haven't decided if I want to send a cookie yet.

<?php setcookie('heron','great blue'); ?>

Yes, sending that cookie was the right decision.

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

讨论

可以给ob_start()传递一个回调函数的名称，以便用该函数来处理输出缓冲。这样可以实现对页面中所有内容的再加工，比如，向搜索地址的机器人程序隐藏电子邮件地址。例8-32中就展示了这样一个回调函数。

例8-32：在调用ob_start()时使用回调

```
<?php
function mangle_email($s) {
    return preg_replace('/([^@\s]+)@([-a-z0-9]+\.)+[a-z]{2,}/is',
                        '<$1@...>',
                        $s);
}

ob_start('mangle_email');
?>

I would not like spam sent to ronald@example.com!

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

其中，mangle_email()函数会把输出转换为：

```
I would not like spam sent to <ronald@...>!
```

output_buffering配置指令会为所有页面打开输出缓冲功能。

```
output_buffering = On
```

同样地，ouput_handler可以设置对所有页面的输出缓冲进行再处理的回调函数：

```
output_handler=mangle_email
```

如果设置了output_handler，那么output_buffering也会自动地设置为on。

参见

ob_start()函数的文档（*http://www.php.net/ob-start*），ob_end_flush()函数的文档（*http://www.php.net/ob-end-flush*）和output_buffering函数的文档（*http://www.php.net/outcontrol*）。

8.13 压缩Web输出

问题

你想给支持自动解压缩的浏览器发送压缩后的内容。

方案

在*php.ini*文件中增加如下设置：

```
zlib.output_compression=1
```

讨论

浏览器是通过Accept-Encoding头部信息告诉服务器它是否能够接受压缩的响应数据的。如果某浏览器发送了Accept-Encoding:gzip或Accept-Encoding:deflate，而PHP也建立于zlib扩展之上，那么zlib.output_compression配置指令会告诉PHP在将输出发送到浏览器之前，先以适当的算法对其进行压缩处理。而浏览器则会对接收到的数据先解压缩再显示。

可以通过zlib.output_compression_level配置指令调整压缩的级别：

```
; minimal compression
zlib.output_compression_level=1

; maximal compression
zlib.output_compression_level=9
```

压缩级别越高，意味着服务器发送给浏览器的数据量越少，但同时也需要服务器CPU更多的时间进行数据压缩。

参见

*zlib*扩展的文档（*http://www.php.net/zlib*）。

8.14 读取环境变量

问题

你想取得一个环境变量的值。

方案

从$_ENV自动全局数组中读取相应的值，如例8-33所示。

例8-33：读取环境变量

```
<?php
$name = $_ENV['USER'];
?>
```

讨论

环境变量是与一个进程相关联的命名值。例如，在Unix系统中，可以检查$_ENV['HOME']的值得知用户主文件夹的位置，如例8-34所示。

例8-34：读取另一个环境变量

```
<?php
print $_ENV['HOME']; //用户的主文件夹
?>
```

在PHP的早期版本中，默认地会为所有环境变量创建相应的PHP变量。出于速度的考虑，从4.1.0版本开始，*php.ini-recommended*文件默认关闭了这一选项，而*php.ini-dist*文件为了向后兼容则仍然启用了%_ENV。

只有在variables_order配置指令中包含字母E的时候，才会创建$_ENV数组。如果$_ENV数组无效，则需要像例8-35所示的那样使用getenv()来取得环境变量的值。

例8-35：使用getenv()函数

```
<?php
$path = getenv('PATH');
?>
```

如果是把PHP作为一个ISAPI模块来使用，那么getenv()函数也是无效的。

参见

技巧8.15中有关设置环境变量的内容。getenv()函数的文档（*http://www.php.*

net/getenv）。PHP中有关环境变量的资料（*http://www.php.net/reserved.variables.php#reserved.variables.environment*）。

8.15 设置环境变量

问题

你想在脚本中或者服务器配置项中设置一个环境变量。按照“host-by-host”原则在服务器配置项中设置环境变量，可以配置不同的虚拟主机。

方案

要在脚本中设置环境变量，可以像例8-36那样使用putenv()函数。

例8-36：设置一个环境变量

```
<?php
putenv('ORACLE_SID=ORACLE'); // 配置oci扩展
?>
```

要在Apache的*httpd.conf*文件中设置一个环境变量，可以使用例8-37中用到的SetEnv语句。注意，以这种方式设置的变量会出现在PHP自动全局数组$_SERVER中，但不会出现在$_ENV中。

例8-37：在Apache配置文件中设置一个环境变量

```
<?php
SetEnv DATABASE_PASSWORD password
?>
```

讨论

在*httpd.conf*中设置环境变量的一个优势是，可以在其中设置比在PHP脚本中更多的限制性读取权限。由于PHP文件对web服务进程而言必须是可读的，而这通常也意味着系统中的其他用户可以看到这些文件。通过把密码保存到*httpd.conf*文件中，就可以避免把密码放到公共文件中的问题。同样，如果有多个主机名都映射到同一个文档目录上，也可以基于各自的主机名配置不同行为的脚本。

例如，有*members.example.com*和*guests.example.com*两个主机名。其中，members版必须通过认证才能开放更多的访问权限；而guests版则提供受限的选项列表，但不需要进行认证。例8-38演示了如何实现这一设置。

例8-38：基于环境变量调整行为

```
<?php
$version = $_SERVER['SITE_VERSION'];

// 如果用户未能正确地登录，则重定向到http://guest.example.com
if ('members' == $version) {
    if (!authenticate_user($_POST['username'], $_POST['password'])) {
        header('Location: http://guest.example.com/');
        exit;
    }
}
include_once "${version}_header"; // 加载自定义的页眉
```

参见

技巧8.14中有关取得环境变量值的讨论。putenv()函数的文档（*http://www.php.net/putenv*）以及在Apache服务器中设置环境变量的相关信息（*http://httpd.apache.org/docs/mod/mod_env.html*）。

8.16 在Apache服务器内部通信

问题

你想要实现PHP与Apache请求进程的其他部分之间的通信。其中包括在*access_log*中设置变量。

方案

如例8-39所示，使用apache_note()。

例8-39：实现与Apache内部的通信

```
<?php
// 取值
$session = apache_note('session');

// 设置值
apache_note('session', $session);
?>
```

讨论

Apache在处理来自客户端的请求时，会经过一系列步骤，而PHP只是整个链条中的一环而已。

Apaceh也能够实现重新映射URL、认证用户身份、记录请求等功能。而在处理请求时，每个句柄都需要访问一组称为记录表的键/值对。通过apache_note()函数可以访问由该请求中的先行句柄在记录表中设置的信息，并为后来的句柄留下信息。

例如，如果你使用session模块对用户进行跟踪而且实现了跨请求保存变量，就可以将这些功能与日志文件分析结合起来，最后能够得到每个用户的平均页面浏览量（page views）。通过apache_note()与日志模块的结合，可以把每个请求的session ID直接写入到*access_log*中。首先，用例8-40中的代码将session ID添加到记录表。

例8-40：把session ID添加到记录表。

```
<?php
// 取得session ID，并将其添加到Apache的记录表中
apache_note('session_id', session_id());
?>
```

然后，修改*httpd.conf*文件，把字符串%{session_id}n添加到LogFormat指令中。该字符串后面字母n的含义是告诉Apache通过另一个模块来使用保存在其记录表中的变量。

如果PHP中启用了--enable-memory-limit配置选项，就会把每个请求的峰值内存使用率保存到名为mod_php_memory_usage的记录中。用%{mod_php_memory_usage}n就可以把相应的内存使用率信息添加到LogFormat中。

参见

apache_note()函数的文档（*http://www.php.net/apache-note*）和Apache中有关日志的信息（*http://httpd.apache.org/docs/mod/mod_log_config.html*）。

8.17 编程：网站账户（反）激活

当用户在你的网站中注册之后，知道用户所提供的电子邮件地址很重要。要验证这个电子邮件地址，可以给他们在注册时提供的地址发一封邮件。如果他们在几天内没有访问包含在邮件中的特定URL，那么就解除它们注册的账户。

整个系统分三个步骤。第一步是通过*notify-use.php*程序给新用户发送一封电子邮件，请用户访问一个确认URL，如例8-42所示。第二步，如例8-43所示，是*verify-user.php*页面程序，用于验证URL并为有效用户添加标记。第三步是*delete-user.php*程序，该程序当用户在一定的时间内没有访问验证URL的情况下，取消相应的账户。第三步程序在例8-44中。

例8-41中包含创建一个用户信息表的SQL语句。

例8-41：创建用户验证表的SQL语句。

```
CREATE TABLE users (
 email VARCHAR(255) NOT NULL,
 created_on DATETIME NOT NULL,
 verify_string VARCHAR(16) NOT NULL,
 verified TINYINT UNSIGNED
);
```

用例8-41中的SQL语句所创建的表，保存的是用户验证所需要的最少信息（也许你还想保存更多有关用户的信息）。在创建了一个用户账户之后，把相应的信息保存到用户表中，然后再给用户发送一封电子邮件告诉他如何激活账户。例8-42中的代码假设用户的电子邮件地址被保存在$email变量中。

例8-42：notify-user.php。

```
<?php
// 连接到数据库
$db = new PDO('sqlite:users.db');

$email = 'david';

// 生成验证字符串
$verify_string = '';
for ($i = 0; $i < 16; $i++) {
    $verify_string .= chr(mt_rand(32,126));
}

// 把用户信息插入到数据库
// 其中使用的是SQLite的datetime()函数
$sth = $db->prepare("INSERT INTO users " .
                    "(email, created_on, verify_string, verified) " .
                    "VALUES (?, datetime('now'), ?, 0)");
$sth->execute(array($email, $verify_string));

$verify_string = urlencode($verify_string);
$safe_email = urlencode($email);

$verify_url = "http://www.example.com/verify-user.php";

$mail_body=<<<_MAIL_
To $email:

Please click on the following link to verify your account creation:

$verify_url?email=$safe_email&verify_string=$verify_string

If you do not verify your account in the next seven days, it will be
deleted.
_MAIL_;
```

```
// mail($email,"User Verification",$mail_body);
print "$email, $mail_body";
```

电子邮件中包含的链接所指向的验证页面，会在验证了适当的信息后更新用户表，如例8-43所示。

例8-43：verify-user.php

```
<?php
// 连接到数据库
$db = new PDO('sqlite:users.db');

$sth = $db->prepare('UPDATE users SET verified = 1 WHERE email = ? '.
                    ' AND verify_string = ? AND verified = 0');

$res = $sth->execute(array($_GET['email'], $_GET['verify_string']));
var_dump($res, $sth->rowCount());
if (! $res) {
    print "Please try again later due to a database error.";
} else {
    if ($sth->rowCount() == 1) {
        print "Thank you, your account is verified.";
    } else {
        print "Sorry, you could not be verified.";
    }
}
?>
```

只有当提供的电子邮件地址和验证字符串与数据库中某一行未经验证的记录匹配时，才会更新用户的验证状态。最后一步，是删除在一定的时间间隔之后仍未验证的用户账户的简短代码，如例8-44所示。

例8-44：delete-user.php

```
<?php
// 连接到数据库
$db = new PDO('sqlite:users.db');

$window = '-7 days';

$sth = $db->prepare("DELETE FROM users WHERE verified = 0 AND ".
                    "created_on < datetime('now',?)");
$res = $sth->execute(array($window));

if ($res) {
    print "Deactivated " . $sth->rowCount() . " users.\n";
} else {
    print "Can't delete users.\n";
}
?>
```

例8-44中的程序每天会运行一次，以便清理用户表中到期仍未验证的用户。如果想要改

变用户验证账户的时效，需要调用变量$window的值，并更新发送给用户的电子邮件内容，以反映新的验证有效期。

8.18 编程：小型Wiki

例8-45中的程序综合运用了本章中所讨论的许多概念，并实现了一个完整的Wiki系统（用户可以编辑其中每一个页面的网站）。它遵循了各种类型的简单PHP程序中一种常见的结构。程序中的代码首先定义了不同的配置选项。然后使用了很多if/else语句来根据表单提交的值或者URL变量决定所要做的事（显示一个页面、保存或者编辑页面等）。程序的其他部分则是供前面if/else语句调用的函数 —— 输出页眉和页脚的函数、加载保存的页面内容的函数以及显示编辑页面表单的函数等。

这个小型Wiki系统需要一个扩展库 —— PHP Markdown（由Michel Fortin开发） —— 的支持，以便处理方便、简洁的Markdow语法与HTML之间的转换。可以从*http://www.michelf.com/projects/php-markdown/*下载PHP Markdown扩展库。

例8-45：小型Wiki

```
<?php

// 使用 Markdown函数生成类-Wiki的标签，详
// 见http://www.michelf.com/projects/php-markdown/
// for Wiki-like text markup
require_once 'markdown.php';

// 保存Wiki页面的文件夹
// 确保web服务器可以写入
define('PAGEDIR',dirname(__FILE__) . '/pages');

// 取得页面名称，或者使用默认页面
$page = isset($_GET['page']) ? $_GET['page'] : 'Home';

// 决定做什么：显示一个编辑表单、保存编辑表单
// 或者显示一个页面

// 显示请求编辑的表单
if (isset($_GET['edit'])) {
    pageHeader($page);
    edit($page);
    pageFooter($page, false);
}
// 保存编辑表单提交的内容
else if (isset($_POST['edit'])) {
    file_put_contents(pageToFile($_POST['page']), $_POST['contents']);
    // 重定向到编辑过页面的常规视图
    header('Location: http://'.$_SERVER['HTTP_HOST'] . $_SERVER['SCRIPT_NAME'].
           '?page='.urlencode($_POST['page']));
    exit();
```

```
}
// 显示一个页面
else {
    pageHeader($page);
    // 如果页面存在，显示该页面，并在页脚中显示一个“Edit”链接
    if (is_readable(pageToFile($page))) {
        // 从保存的文件中取得页面的内容
        $text = file_get_contents(pageToFile($page));
        // 转换 Markdown 语法 (使用markdown.php中Markdown()函数)
        $text = Markdown($text);
        // 生成指向其他wiki页面的空链接
        $text = wikiLinks($text);
        // 显示这个页面
        echo $text;
        // 显示页脚
        pageFooter($page, true);
    }
    // 如果页面不存在，显示一个编辑表单
    // 以及不带“Edit”链接的页脚
    else {
        edit($page, true);
        pageFooter($page, false);
    }
}

// 页面的页眉 —— 相当简单，只包含标题和通常的
// pleasantries
function pageheader($page) { ?>
<html>
<head>
<title>Wiki: <?php echo htmlentities($page) ?></title>
</head>
<body>
<h1><?php echo htmlentities($page) ?></h1>
<hr/>
<?php
}

// 页面的页脚 —— 包含一个“last modified”时间戳，一个可选的“Edit”链接
// 和一个返回Wiki首页的链接
function pageFooter($page, $displayEditLink) {
    $timestamp = @filemtime(pageToFile($page));
    if ($timestamp) {
        $lastModified = strftime('%c', $timestamp);
    } else {
        $lastModified = 'Never';
    }
    if ($displayEditLink) {
        $editLink = ' - <a href="?page='.urlencode($page).'&edit=true">Edit</a>';
    } else {
        $editLink = '';
    }
?>
<hr/>
```

```
<em>Last Modified: <?php echo $lastModified ?></em>
<?php echo $editLink ?> - <a href="<?php echo $_SERVER['SCRIPT_NAME'] ?>">Home<
/a>
</body>
</html>
<?php
}

// 显示一个编辑表单。如果页面存在，在表单中
// 包含其内容
function edit($page, $isNew = false) {
    if ($isNew) {
        $contents = '';
?>
<p><b>This page doesn't exist yet.</b> To create it, enter its contents below
and click the <b>Save</b> button.</p>
    <?php } else {
        $contents = file_get_contents(pageToFile($page));
    }
?>
<form method='post' action='<?php echo htmlentities($_SERVER['SCRIPT_NAME']) ?>'>
<input type='hidden' name='edit' value='true'/>
<input type='hidden' name='page' value='<?php echo htmlentities($page) ?>'/>
<textarea name='contents' rows='20' cols='60'>
<?php echo htmlentities($contents) ?></textarea>
<br/>
<input type='submit' value='Save'/>
</form>
<?php
}

// 将提交的页面转换为一个文件名。使用md5()函数避免$page中淘气的字符
// 可能导致的安全问题
function pageToFile($page) {
    return PAGEDIR.'/'.md5($page);
}

// 把页面中诸如 [something] 之类的文本转换成
// Wiki页面中 "something" 这样的HTML链接
function wikiLinks($page) {
    if (preg_match_all('/\[([^\]]+?)\]/', $page, $matches, PREG_SET_ORDER)) {
        foreach ($matches as $match) {
            $page = str_replace($match[0], '<a href="'.$_SERVER['SCRIPT_NAME'].
'?page='.urlencode($match[1]).'">'.htmlentities($match[1]).'</a>', $page);
        }
    }
    return $page;
}
?>
```

第9章

表单

9.0 概述

能将表单变量无缝地整合到程序当中是PHP天才一面的体现。这使得Web编程更加平滑而简单，加速了从Web表单到PHP代码再到HTML输出的整个开发周期。

然而，在带来方便的同时，对于这么轻易就跑到程序中去的那些用户提供的信息，也产生确保其中包含适当内容的责任。由于不能信任外部输入，所以总是要强制性地验证所有输入数据。技巧9.2～9.9分别讨论了如何验证常见的各种信息，同时也对可能需要的任意表单验证给出了一般的指导方针。技巧9.10介绍了通过转义HTML实体来安全地显示用户键入的数据。技巧9.14讲解了如何处理用户上传的文件。

HTTP是一种“无状态”的协议 —— 即它没有内在的机制可以使一个页面中的信息能够保存到另一个页面中被访问。技巧9.11、9.12和9.13讨论的都是“如何知道是哪个用户发出的哪一个对Web服务器的请求”这样一个基本的问题。

不论什么时候，PHP在处理一个页面时，它都会检查URL和表单变量、上传的文件、可用的cookie及Web服务器和环境变量。之后，这些信息就可以通过下面这些数组直接访问了：`$_GET`、`$_POST`、`$_FILES`、`$_COOKIE`、`$_SERVER`和`$_ENV`。也就是说，PHP会分别保存在查询字符串中设置的变量、一个post请求的主体内容、上传的文件、cookie、Web服务器及Web服务器运行其上的环境等这些信息。当然，也有一个`$_REQUEST`变量，它是包含着前面六个数组中全部值的一个大数组。

当在`$_REQUEST`数组中放置元素时，如果两个数组都有一个同名的键，PHP会根据`variables_order`配置指令决定如何切断它们之间的联系。在默认情况下，`variables_order`的值是EGPCS（如果你使用的是*php.ini-recommended*配置文件，则是GPCS）。也就是说，PHP首先会把环境变量添加到`$_REQUEST`中，然后再依次添加查询

字符串、post、cookie和Web服务器变量。在这种情况下，因为默认是C在P的后面，所以一个名为username的cookie会覆盖一个同样名为username的post变量。注意，*php.ini-recommended*文件中的GPCS值表明$_ENV数组中的环境变量不会被添加到$_REQUEST数组中。

虽然$_REQUEST很方便，但一般来说还是直接在更具体的数组中查找相应的变量更好一些。因为这样，你会清楚地知道自己在找什么，并且不用关心variables_order指令的改变是否会影响你的程序。

以上所有数组都是自动全局数组。这就意味着无论是在函数还是在类中，它们都有定义。

在PHP 4.1之前，这些自动全局变量是不存在的。当时，它们还只是一些常规的数组，名字分别为 $HTTP_COOKIE_VARS、$HTTP_ENV_VARS、$HTTP_GET_VARS、$HTTP_POST_VARS、$HTTP_POST_FiLES和$HTTP_SERVER_VARS。出于继承的原因，这些数组仍然是有效的，只不过新增加的数组更容易使用。这些老数组只有在track_vars配置指令为on时才会被赋值，但从PHP 4.0.3开始，这个功能就成了默认开启的了。

最后，如果register_globals配置指令的值是on，那么以上所有的变量作为全局名称空间中的变量也都将是有效的。所以$_GET['password']也可以改写为$password。这样虽然方便了，但却引入了一个较大的安全问题。不怀好意的人也会因此很容易从外部设置一个变量从而覆盖受信任的内部变量。自PHP 4.2起，register_globals的默认值是off。

例9-1是一个基本的表单。这个表单要求用户填写他的第一个名字。当表单被提交时，信息就会发送到*hello.php*页面。

例9-1：基本的HTML表单

```
<form action="hello.php" method="post">
What is your first name?
<input type="text" name="first_name" />
<input type="submit" value="Say Hello" />
</form>
```

表单中文本输入元素的名称是first_name。并且，表单的提交方法是post。也就是说，当提交表单时，$_POST['first_name']会保存用户输入的任何字符串（当然，如果用户一个字也没有打，那就可能是空的）。

例9-2显示了*hello.php*的内容，它可以显示来自表单的信息。

例9-2：基本的表单处理

```
<?php
echo 'Hello, ' . $_POST['first_name'] . '!';
?>
```

如果你在例9-1的表单中输入了Twinkle，那么例9-2就会输出：

```
Hello, Twinkle!
```

例9-2实在是太简单了，其中忽略了所有PHP表单处理应用程序中很重要的两个步骤：数据验证（保证输入的信息对程序而言是可以接受的）和输出转义（保证恶意用户不会利用你的网站攻击其他的网站）。技巧9.2～9.9围绕着数据验证的主题进行讨论，而技巧9.10则讨论了输出转义的话题。

9.1 处理表单的输入

问题

你想利用同一个HTML页面来发出表单，然后处理在其中键入的数据。换句话说，你在努力避免将一宗事务分割成几步分别处理而导致页面激增的现象。

方案

使用$_SERVER['REQUEST_METHOD']变量来确定提交的请求使用的是get方法还是post方法。如果使用的是get方法，则输出表单。如果使用的是post方法，则对表单进行处理。例9-3将例9-1的表单和例9-2中的代码组合到了一个程序当中，并根据$_SERVER['REQUEST_METHOD']来判断该做什么。

例9-3：根据请求的方法决定做什么

```
<?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
<form action="<?php echo $_SERVER['SCRIPT_NAME'] ?>" method="post">
What is your first name?
<input type="text" name="first_name" />
<input type="submit" value="Say Hello" />
</form>
<?php } else {
    echo 'Hello, ' . $_POST['first_name'] . '!';
}
?>
```

讨论

回首过去的岁月，还是在Web流行的早期，当我们的前人草创表单的时候，通常都会用

到两个文件：一个包含表单的静态HTML页面和一个用于处理那个表单并给用户返回动态生成的响应的脚本文件。这种机制是不够灵活的，因为*form.html*引导到*form.cgi*，而当你改变某个页面时，必须要记得去编辑另一个文件，否则脚本就可能中断。

通常，把所有部分都放到同一个文件和前后环境当中，并规定哪一部分用于显示的表单会更容易维护。get方法（当你只是在地址栏中输入URL再按回车或者点击链接之后浏览器使用的方法）的意思是“嗨，服务器，把你拿到的内容给我”。而post方法（当你提交一个方法属性的值为post的表单时浏览器所使用的方法）的意思则是“嗨，服务器，给你一些要更新的数据”。所以，对get请求的典型响应就是HTML表单，而对post请求的响应则是对那个表单处理的结果。在例9-3中，所谓的“处理”极其简单 —— 只是输出一句问候语。在许多典型的应用程序当中，这个处理过程会更复杂 —— 如把信息保存到数据库或者发送一封电子邮件等。

注意，虽然XHTML规范要求<form/>元素的method属性必须小写（get或post），而HTTP规范则要求Web浏览器在向服务器发送请求方法类型时必须使用全部大写形式（GET或POST）。而$_SERVER['REQUESTS_METHOD']中的值是与浏览器一致的，所以在实践中这两种方法总是采用大写形式。

还有一种可以令页面容易维护的技术 —— 不要直接在表单的action属性中使用目标页面的硬编码路径。因为采用硬编码形式之后，除非编辑该属性，否则不可能对该页面重命名或者重新设定目标页面。这种技术就是在表单的action属性中使用$_SERVER['SCRIPT_NAME']变量。这个变量是PHP在每个请求的基础之上建立的，其中包含着当前脚本的文件名（相对于文档根目录）。

参见

技巧9.11中处理多个表单的内容。

9.2 验证表单输入：必填字段

问题

你想要确保从一个表单元素中得到一个值。例如，你想确保一个文本框不会被留空。

方案

使用strlen()来测试$_GET或$_POST中的元素值，如例9-4所示。

例9-4：测试一个必填字段

```
<?php
if (! strlen($_POST['flavor'])) {
   print 'You must enter your favorite ice cream flavor.';
}
?>
```

讨论

不同类型的表单元素留空会导致$_GET和$_POST中的元素值不尽相同。空文本框、空文本区域及空的文件上传字段的值是零长度字符串。而未选中的复选框和单选按钮，不会在$_GET或$_POST中生成任何值。浏览器通常会强迫在单选下拉列表选中一个项目，而对于多选下拉列表，如果没有选中其中的任何项目，那么结果和复选框是一样的，即不会在$_GET或$_POST中生成任何值。

不仅如此，就连请求本身也并非只能来自Web浏览器。也就是说，PHP程序可能会收到来自另一个程序请求、一个好奇的电脑黑客通过手工构造的请求或者一个恶意攻击者为了发现你系统中的漏洞而建立的请求。要想让程序尽可能地健壮，就必须在对一个元素应用其他验证策略以前，首先检查这个元素是否存在于$_GET或$_POST中。此外，如果验证策略假设元素包含在一个数组中（如例9-5那样），可以使用is_array()来验证是否存在一个数组。

例9-5使用了isset()、strlen()和is_array()，以便完成最严格的表单验证。

例9-5：严格的表单验证

```
<?php
// 在检查长度之前，先检查$_POST['flavor']是否存在
if (! (isset($_POST['flavor']) && strlen($_POST['flavor']))) {
   print 'You must enter your favorite ice cream flavor.';
}

// $_POST['color']是可选的，但如果没有留空
// 就必须大于5个字符
if (isset($_POST['color']) && (strlen($_POST['color']) <=5 )) {
   print 'Color must be more than 5 characters.';
}

// 确保 $_POST['choices'] 存在并且是一个数组
if (! (isset($_POST['choices']) && is_array($_POST['choices']))) {
   print 'You must select some choices.';
}
?>
```

出于某种偏好，你可能会尝试使用empty()代替strlen()来测试一个文本框中是否填写了值。但根据PHP的布尔值计算规则，字符0可以转换为false，因此这种偏好可能会导致

问题。也就是说，如果有人在children文本框中填入了0，那么$_POST['children']中的值就是0，而empty($_POST['children'])测试的结果是true——显然，从表单验证的角度来看这是错误的。

参见

技巧9.5中有关验证下拉菜单的内容，技巧9.6中有关验证单选按钮的内容和技巧9.7中有关验证复选框的内容。

9.3 验证表单输入：数字

问题

你想确保一个表单输入框中输入的是数字。例如，你不想有人在告诉你他的年龄时说“一把年纪了”或者“还很稚嫩”，而实际上他们的年龄一个13岁一个56岁。

方案

如果要确定值是否为大于或等于零的整数，可以像例9-6那样使用ctype_digit()。

例9-6：用ctype_digit()来验证数字

```
<?php
if (! ctype_digit($_POST['age'])) {
    print 'Your age must be a number bigger than or equal to zero.';
}
?>
```

如果要确定值是否为正整数或者负整数，可以将这个提交的值与该值在转换成整数之后返回的字符串进行比较，如例9-7所示。

例9-7：用类型转换法验证整数

```
<?php
if ($_POST['rating'] != strval(intval($_POST['rating']))) {
   print 'Your rating must be an integer.';
}
?>
```

如果要确定值是否为正或负的小数，可以将这个提交的值与该值在转换为浮点数之后返回的字串进行比较，如例9-8所示。

例9-8：用类型转换法验证小数

```
<?php
```

```
if ($_POST['temperature'] != strval(floatval($_POST['temperature']))) {
   print 'Your temperature must be a number.';
}
?>
```

讨论

数字验证在PHP中属于那种看似简单，实则不然的一类操作。一种常见的冲动可能是使用内置的is_number()函数来验证数字。遗憾的是，is_number()所认为的数字更符合计算机的特点而不是人的思维。比如说，十六进制的数字字符串0xCAFE和带指数符号的数字字符串10e40对于is_number()来说都是数字。

在验证数字（及所有表单输入）时必须牢记的事项是：$_GET和$_POST中的值始终是字符串。也就是说，如果某人在一个名为zip_code的文本框中填入了06520并提交了表单，那么$_POST['zip_code']的值会是五位的字符串06520，而不是整数6520。

因此，如果需要验证“仅由数字组成的值”时，ctype_digit()就是正确的选择。ctype_digit()与其他ctype函数类似，要求传递给它的参数必须是一个字符串，但只是在验证表单输入时如此，因为$_GET和$_POST中所有的值都是字符串。

在PHP 5.1版之前，如果向ctype_digit()传递一个空字符串，不会得到满意的结果（ctype_digit('') 返回 true），所以要保证在将值传递给ctype_digit()之前，要先照技巧9.2中讨论的那样验证一下输入是否为空。同样，ctype_digit()也有一个缺点（存在于PHP的所有版中），即它不是很灵活 —— 只能用于验证数字。如果你想接受负数或者小数时，它就无能为力了。

在想验证值是否为负数或者小数时，可以使用PHP的两个类型转换函数：intval() —— 返回“完整的整数”字符串和floatval() —— 返回“完整的浮点数”字符串。当给这两个函数传递一个字符串时，它们都会尽力从那个字符串中返回一个数值。例如，intval('06520')返回整数6520，intval('-2853')返回整数-2853，floatval('3.1415')返回浮点数3.1415，而floatval('-473.20')则返回浮点数-473.2。

这两个函数之所以能在验证表单输入时派上用场的关键之处，就在于它如何看待不是有效数字的字符串。它们都会返回能够从字符串中找到的最大的数字，从第一个字符开始并忽略起首的空白字符。也就是说，intval('-6 weeks')返回-6，intval('30x bigger')返回30，intval('3.1415')返回3，而intval('21+up')返回21。floatval()的行为类似，但可以接受小数点。例如，floatval('127.128.129.130')返回127.128。如果给这两函数传递一个不包含任何有效数字字符的字符串，它们都返回0。

这就意味着，不论是把用户输入传递给intval()还是floatval()，它们的作用就像是一个过滤器 —— 除了将无效的值转换为有数字意义的值之外，只留下未经修改的有效值。如果通过过滤后没有修改其中任何字符，那么与原始输入的比较结果就会成功 —— 换句话说，如果原始输入是一个有效的整数或者小数时比较就会成功。

为了保证PHP能够进行适宜的比较，有必要把intval()或floatval生成的结果再通过strval()转换成一个字符串。当PHP比较的是两个字符串时，比较的结果才是有保证的（如果字符串相同结果是true，否则就是false）。但是，当PHP比较一个字符串和一个数值（比如intval()或floatval()返回的结果）时，它会尝试把字符串转换为数值（用上面提到的方法）。而这样就会抵消intval()或floatval()的“过滤”效果，所以我们应该防止这种情况出现。而要确保通过比较两个字符串来实现验证。

如果这一次次转换令你觉得不够稳妥，而你又是一个正则表达式的爱好者，当然也可使用正则表达式来实现验证。例9-9中给出了一个用于验证一个整数和一个小数的正则表达式。

例9-9：用正则表达式验证数字

```
<?php
// 这个模式匹配一个可选的-号
// 后跟至少一位数字
if (! preg_match('/^-?\d+$/'$_POST['rating'])) {
    print 'Your rating must be an integer.';
}

// 这个模式匹配一个可选的-号
// 后跟位于一个小数点之前的可选数字
// 一个可选的小数点
// 和至少一位小数
if (! preg_match('/^-?\d*\.?\d+$/',$_POST['temperature'])) {
   print 'Your temperature must be a number.';
}
?>
```

对于性能调试纯粹论者来说，这是一个老生常谈的问题，他们认为应该避免使用正则表达式，因为使用正则表达式的速度会比较慢。然而，在当前的情况下，使用那么简单的正则表达式的效率与使用类型转换函数是没有什么差别的。如果你对正则表达式很熟悉，或者同时也在其他环境当中使用这些正则表达式，那么使用正则表达式就是一个便利的选择。而且，正则表达式也可以处理PHP无法在不损失精度的前提下保存的有效浮点数，如782364.238723123。如果你打算把经度和纬度的数据保存为字符串时，正则表达式就会派上用场了。最后要说的是，ctype_digit()函数不论比类型转换还是比正则表达式的速度都要快得多，所以如果该函数能满足你的需要，它还是首选。

参见

技巧9.2中有关验证必填字段的内容。ctype_digit()函数的文档（*http://www.php.net/ctype_digit*）。

9.4 验证表单输入：电子邮件地址

问题

你想知道用户所提供的一个电子邮件地址是否有效。

方案

使用例9-10中的is_valid_email_address()函数。这个函数会依据RFC 822标准中的有关规则判断一个电子邮件地址是否有效。

例9-10：验证一个电子邮件地址

```
function is_valid_email_address($email){
        $qtext = '[^\\x0d\\x22\\x5c\\x80-\\xff]';
        $dtext = '[^\\x0d\\x5b-\\x5d\\x80-\\xff]';
        $atom = '[^\\x00-\\x20\\x22\\x28\\x29\\x2c\\x2e\\x3a-\\x3c'.
            '\\x3e\\x40\\x5b-\\x5d\\x7f-\\xff]+';
        $quoted_pair = '\\x5c[\\x00-\\x7f]';
        $domain_literal = "\\x5b($dtext|$quoted_pair)*\\x5d";
        $quoted_string = "\\x22($qtext|$quoted_pair)*\\x22";
        $domain_ref = $atom;
        $sub_domain = "($domain_ref|$domain_literal)";
        $word = "($atom|$quoted_string)";
        $domain = "$sub_domain(\\x2e$sub_domain)*";
        $local_part = "$word(\\x2e$word)*";
        $addr_spec = "$local_part\\x40$domain";
        return preg_match("!^$addr_spec$!", $email) ? 1 : 0;
}

if (is_valid_email_address('cal@example.com')) {
   print 'cal@example.com is a valid e-mail address';
} else {
   print 'cal@example.com is not a valid e-mail address';
}
```

讨论

RFC 822中定义了有效电子邮件的标准。例9-10中函数的作者是Cal Henderson，该函数使用了RFC 822中自带的语法建立了一个正则表达式。如果你想了解这个函数的建立过程，可以访问 *http://www.iamcal.com/publish/articles/php/parsing_email*。Cal同时还写了

一个根据更复杂的RFC 2822标准中有关规则验证电子邮件地址的函数，该函数可以在*http://code.iamcal.com/php/rfc822/ rfc822.phps*下载到。

例9-10中的函数只是检查了一个特定的电子邮件地址是否符合构成规则。如果一个用户突然告诉你他的电子邮件地址是`bingolover2261@example`而不是`bingolover2261@example.com`，这个函数立即就可以分辨出来。但是，该函数无法告诉你的是，那个电子邮件地址是否能收到你发送的邮件。而且，更不能告诉你提供那个电子邮件地址的人是否有权控制该地址。要了解这种信息，你必须向那个地址发一封确认邮件。其中的确认信息可以让该用户执行一些确定的任务（回复该邮件，或点击一个链接），以便确认他与在表单中填写该地址的人是同一个人。或者，也可以考虑在确认信息中告诉用户如果他与在表单中填写该地址的人并非同一个人，他该怎么做（回复该邮件，或点击一个链接）。技巧8.17中演示了一个系统，该系统会发送一封包含链接的电子邮件，而收件人必须通过点击该链接来确认所提供的电子邮件地址。

参见

RFC 822标准（*http://www.faqs.org/rfcs/rfc2822.html*），Cal Henderson的文章“Parsing Email Addresses in PHP”（*http://www.iamcal.com/publish/articles/php/parsing_email*）。可以在*http:// code.iamcal.com/php/rfc822/*下载到本技巧中验证电子邮件地址的函数。

9.5 验证表单输入：下拉菜单

问题

你想确定由HTML`<select/>`元素生成的下接菜单中是否有被选中的项目。

方案

使用一个包含值的数组生成相应的下拉列表。然后，在验证输入时检查提交的值是否在该数组中。例9-11使用`in_array()`函数实现了这一验证。

例9-11：用in_array()来验证一个下拉菜单

```
<?php
// 生成下拉菜单
$choices = array('Eggs','Toast','Coffee');
echo "<select name='food'>\n";
foreach ($choices as $choice) {
   echo "<option>$choice</option>\n";
}
echo "</select>";
```

```
// 之后，再验证菜单项目
if (! in_array($_POST['food'], $choices)) {
    echo "You must select a valid choice.";
}
?>
```

例9-11所生成的下拉菜单如下：

```
<select name='food'>
<option>Eggs</option>
<option>Toast</option>
<option>Coffee</option>
</select>
```

如果想验证在每个<option/>元素中设置的value属性的值，可以像例9-12那样使用array_key_exist()函数来对输入进行验证。

例9-12：用array_key_exist()来验证一个下拉菜单

```
<?php
// 生成下拉菜单
$choices = array('eggs' => 'Eggs Benedict',
                 'toast' => 'Buttered Toast with Jam',
                 'coffee' => 'Piping Hot Coffee');
echo "<select name='food'>\n";
foreach ($choices as $key => $choice) {
   echo "<option value='$key'>$choice</option>\n";
}
echo "</select>";

// 之后，再验证菜单项目的值
if (! array_key_exists($_POST['food'], $choices)) {
    echo "You must select a valid choice.";
}
?>
```

例9-12所生成的下拉菜单如下：

```
<select name='food'>
<option value='eggs'>Eggs Benedict</option>
<option value='toast'>Buttered Toast with Jam</option>
<option value='coffee'>Piping Hot Coffee</option>
</select>
```

讨论

例9-11和例9-12中的两种方法因生成的菜单不同而不同。例9-11用带有自动数字编号的$choices数组来生成并输出<option/>元素。例9-12则是用带有字符串键的$choices数组来生成并输出<option/>元素。

不论是哪种情况，验证的策略都是相同的，即确保表单元素提交的值是一个存在的数组元素。对于行为规范的浏览器所提交的请求，这里的验证规则永远不会失败 —— 因为Web浏览器不会让你自己在下拉菜单中生成选项。虽然如此，也要记住发送到你的PHP程序的请求不一定来自一个行为规范的浏览器。那个浏览器可能是一个充满bug的浏览器，也可能是一个11岁大的无聊少年，一手拿着HTTP规范，而另一只手在操作着telnet客户端向你的程序发送请求。因为你始终都需要留意那些恶意的、手工的HTTP请求，所以即使在多数用户不会出现差错的环境中，也需要验证输入。

参见

in_array()函数的文档（*http://www.php.net/in_array*）和array_key_exists()函数的文档（*http:// www.php.net/array_key_exists*）。

9.6 验证表单输入：单选按钮

问题

你想确定在一组单选按钮中有一个是被选中的。

方案

用数组的值生成单选按钮。然后通过检查提交的值是否存在于数组中来验证输入。例9-13使用了array_key_exists()来完成验证。

例9-13：*验证单选按钮*

```
<?php
// 生成单选按钮
$choices = array('eggs' => 'Eggs Benedict',
                 'toast' => 'Buttered Toast with Jam',
                 'coffee' => 'Piping Hot Coffee');
foreach ($choices as $key => $choice) {
   echo "<input type='radio' name='food' value='$key'/> $choice \n";
}


// 之后，验证提交的单选按钮
if (! array_key_exists($_POST['food'], $choices)) {
    echo "You must select a valid choice.";
}
?>
```

讨论

例9-13中的单选按钮验证与例9-12中的下拉菜单验证非常类似。它们都遵循同样的模式——定义描述选项的数据，生成适当的HTML，然后再用定义的数据来验证提交值的有效性。它们的区别在于生成的HTML不同。

下拉菜单和单选按钮的另一个区别是处理默认值的方式不同。对于下拉菜单，如果HTML中没有明确地为它指定默认的选项，则默认选中第一个选项。然而，如果在HTML中没有明确地为一组单选按钮设定默认值，那么就不会有默认的选项被选中。

在一个行为规范的浏览器中，要想保证一组单选按钮中有一个是被选中的，就要在默认的选项中加上checked="checked"属性。另外，为了预防手工恶意请求中遗漏的值，还需要像技巧9.2中介绍的那样使用isset()来确保输入中包含来自单选按钮的值。

参见

技巧9.2中有关验证必填字段的内容。array_key_exists()函数的文档（*http://www.php.net/array_key_exists*）。

9.7 验证表单输入：复选框

问题

你想确定只有有效的复选框被选中。

方案

如果只有一个复选框，要保证若提供了值，则该值应该是正确的。如果没有提供值，则说明那个复选框没有被选中。例9-14中对一个复选框是否被选中、未被选中或者提供了无效的值进行了验证。

例9-14：验证一个复选框

```
<?php
// 生成复选框
$value = 'yes';
echo "<input type='checkbox' name='subscribe' value='yes'/> Subscribe?";

// 之后，验证复选框
if (isset($_POST['subscribe'])) {
    // 所提供的值是正确的
    if ($_POST['subscribe'] == $value) {
```

```
        $subscribed = true;
    } else {
        // 所提供的值不正确
        $subscribed = false;
        print 'Invalid checkbox value submitted.';
    }
} else {
    // 没有提供值
    $subscribed = false;
}

if ($subscribed) {
    print 'You are subscribed.';
} else {
    print 'You are not subscribed';
}
?>
```

对于一组复选框，使用数组的值生成复选框。然后，再用array_intersect()来保证所提供的一组值包含在可接受值的范围之内，如例9-15所示。

例9-15：验证一组复选框

```
<?php
// 生成复选框
$choices = array('eggs' => 'Eggs Benedict',
                 'toast' => 'Buttered Toast with Jam',
                 'coffee' => 'Piping Hot Coffee');
foreach ($choices as $key => $choice) {
   echo "<input type='checkbox' name='food[]' value='$key'/> $choice \n";
}
?>

// 之后，验证复选框的值
if (array_intersect($_POST['food'], array_keys($choices)) != $_POST['food']) {
    echo "You must select only valid choices.";
}
?>
```

讨论

为了让PHP更稳妥地处理多个复选框的值，复选框的名字必须以[ ]结尾，技巧9.17中讨论了这一点。在$_POST中所提供的多个值会被格式化为一个数组。因为例9-15中复选框的名字是food[]，所以$_POST['food']就保存着所选中的复选框的值组成的数组。

array_intersect()函数用于筛选出$_POST['food']和array_keys($choices)共有的全部元素。也就是说，该函数会对提交的选项（$_POST['food']）进行过滤，只有是$choices数组中某个键的值才会留下。如果$_POST['food']中所有的值都被留下了，则array_intersect($_POST['food']、array_keys ($choices))的结果就是$_POST['food']

的一个未更改过的副本。所以，如果返回的结果数组不等于$_POST['food']，那么就说明所提交的输入中包含无效的值。

复选框也和单选按钮一样存在默认值的问题。所以，就和处理单选按钮一样，在进一步验证之前仍然要用技巧9.2中介绍的规则来确定输入中包含复选框提交的值。

参见

技巧9.2中有关验证必填字段的内容。array_intersect()函数的文档（*http://www.php.net/array_intersect*）。

9.8 验证表单输入：日期和时间

问题

你想确定用户填写的日期或时间是否有效。例如，你想确保用户不会在日程表中安排8月份的第45天开会，或者提供已经过期的信用卡。

方案

如果你在表单中分别提供了月、日、年字段，可以把它们的值组合起来传递给checkdate()函数进行验证，如例9-16所示。该函数可以报告不论月、日还是年有问题的情况。

例9-16：检查特定的日期

```
<?php
if (! checkdate($_POST['month'], $_POST['day'], $_POST['year'])) {
   print "The date you entered doesn't exist!";
}
?>
```

如果要检查一个日期值是在一个特定的时间之前还是之后，可以将用户提供的值转换为时间戳，并计算有效日期的时间戳，最后比较这两个值。例9-17用于检查所提供信用卡的有效年月值是否在当前月份之后。

例9-17：检查信用卡是否有效

```
<?php
// 信用卡失效月份的初始时刻
$expires = mktime(0, 0, 0, $_POST['month'], 1, $_POST['year']);
// 下个月的初始时刻
// 如果 date('n') + 1 == 13, mktime() 正常执行并使用
```

```
// 下一年的一月份
$nextMonth = mktime(0, 0, 0, date('n') + 1, 1);
if ($expires < $nextMonth) {
   print "Sorry, that credit card expires too soon.";
}
?>
```


讨论

因为checkdate()知道哪一年是闰年以及每月都有多少天，所以使用它可以免除我们逐一比较每个日期元件的烦恼。至于有效范围的验证（确定一个日期或时间在其他日期或时间之前、之后还是之间），使用时间戳是最佳方式。

参见

第3章中有关处理日期和时间的深入讨论。

9.9 验证表单输入：信用卡

问题

你想确定用户没有填写假信用卡号码。

方案

例9-18中的is_valid_credit_card()函数可以告诉你所提供的信用卡号从构成规则上判断是否有效。

例9-18：验证信用卡号

```
<?php
function is_valid_credit_card($s) {
    // 删除非数组并反序
    $s = strrev(preg_replace('/[^\d]/','',$s));
    // 计算检查次数
    $sum = 0;
    for ($i = 0, $j = strlen($s); $i < $j; $i++) {
        // 偶数原封不动
        if (($i % 2) == 0) {
            $val = $s[$i];
        } else {
            // 奇数乘以2再减9是否大于9
            $val = $s[$i] * 2;
            if ($val > 9) { $val -= 9; }
        }
```

```
        $sum += $val;
    }
    // 如果和是10的倍数则号码有效
    return (($sum % 10) == 0);
}

if (! is_valid_credit_card($_POST['credit_card'])) {
    print 'Sorry, that card number is invalid.';
}

?>
```

讨论

为了避免意外差错，信用卡使用了Luhn算法。该算法，也就是例9-18的`is_valid_credit_card()`函数中使用的算法，分别对信用卡号码的每一位数字进行处理，然后会得到卡号是否有效的结论。

验证信用卡号多少有点类似验证电子邮件地址。验证构成规则——保证提供的值是符合标准规定的字符序列——还是相对简单的。如果是语义层面的验证，就需要更多的技巧了。比如，像“`4111 1111 1111 1111`”这样的信用卡号虽然能够轻易通过例9-18中函数的验证，但它却是无效的。这个号码是一个众所周知的类似Visa卡号的测试号码（而且，同样地，如果书中需要以它为例时也可以用）。

就如同严格的电子邮件地址验证必须通过外部确认（通常是给相应的地址发送一封带有确认链接的电子邮件）一样，完整的信用卡号验证也少不了外部验证这一环，也就是说，还需要将卡号和相关的账户信息（卡持有者的姓名和地址）提交给付款机，并确保通过认可。

显然，构成规则验证能够很好地防止由于疏漏而导致的填写错误，但却不是验证信用卡号的全部。

参见

技巧9.4中有关验证电子邮件地址的内容。有关Luhn算法的信息（*http://en.wikipedia.org/wiki/Luhn*）。

9.10 预防跨站点脚本

问题

你想在一个HTML页面上安全地显示用户输入的数据。例如，你想允许用户给博客文章添加评论，但又希望评论中的HTML或JavaScript不会导致问题。

方案

如例9-19所示的那样，在显示用户输入的数据之前先用htmlentities()转换该数据。

例9-19：转义HTML

```
<?php
print 'The comment was: ';
print htmlentities($_POST['comment']);
?>
```

讨论

PHP有两个用于转义HTML实体的函数。最基本的是能够转义<、>、"和&这四个特殊字符的htmlspecialchars()函数，根据一个可选的参数，该函数也可以转换'而不转换"，或者同时转换'和"。而对于更复杂的编码需求，就要使用htmlentities()了。htmlentities()扩展了htmlspecialchars()，并实现了对任何HTML实体字符的转换。例9-20中使用了htmlspecialchars()函数。

例9-20：转义HTML实体

```
<?php
$html = "<a href='fletch.html'>Stew's favorite movie.</a>\n";
print htmlspecialchars($html);                  // 双引号
print htmlspecialchars($html, ENT_QUOTES);      // 单引号和双引号
print htmlspecialchars($html, ENT_NOQUOTES);    // 两者都不转换
?>
```

例9-20的输出结果：

```
&lt;a href=&quot;fletch.html&quot;&gt;Stew's favorite movie.&lt;/a&gt;
&lt;a href=&quot;fletch.html&quot;&gt;Stew&#039;s favorite movie.&lt;/a&gt;
&lt;a href="fletch.html"&gt;Stew's favorite movie.&lt;/a&gt;
```

在默认情况下，htmlentities()和htmlspecialchars()使用的都是ISO-8859-1字符集。如果要使用其他字符集，可以将要使用的字符集作为第三个参数。例如，要使用UTF-8，就可以调用htmlentities($string, ENT_QUOTES, 'UTF-8')。

参见

技巧18.4和19.13。htmlentities()函数的文档（*http://www.php.net/htmlentities*）和htmlspecialchars()函数的文档（*http://www.php.net/htmlspecialchars*）。

9.11 处理多页表单

问题

你想使用一个由多个页面组成的表单，同时在各个页面间保有数据。例如，一个调查表单由于问题太多不可能全都放到一个页面中。

方案

使用session跟踪法保存每个页面中的表单信息，同时使用一个变量保持对当前所显示页面的跟踪。例9-21显示了一个由两个页面组成的表单，以及收集的提交结果。

例9-21：多页表单

```
<?php
// 开启session
session_start();

// 确定当前的页面
if (($_SERVER['REQUEST_METHOD'] == 'GET') || (! isset($_POST['stage']))) {
    $stage = 1;
} else {
    $stage = (int) $_POST['stage'];
}

// 保存所有提交的数据
if ($stage > 1) {
    foreach ($_POST as $key => $value) {
        $_SESSION[$key] = $value;
    }
}

if ($stage == 1) { ?>

<form action='<?php echo $_SERVER['SCRIPT_NAME'] ?>' method='post'>

Name: <input type='text' name='name'/> <br/>
Age:  <input type='text' name='age'/> <br/>

<input type='hidden' name='stage' value='<?php echo $stage + 1 ?>'/>
<input type='submit' value='Next'/>
</form>
```

```
<?php } else if ($stage == 2) { ?>

<form action='<?php echo $_SERVER['SCRIPT_NAME'] ?>' method='post'>

Favorite Color: <input type='text' name='color'/> <br/>
Favorite Food:  <input type='text' name='food'/> <br/>


<input type='hidden' name='stage' value='<?php echo $stage + 1 ?>'/>
<input type='submit' value='Done'/>
</form>

<?php } else if ($stage == 3) { ?>

    Hello <?php echo $_SESSION['name'] ?>.
    You are <?php echo $_SESSION['age'] ?> years old.
    Your favorite color is <?php echo $_SESSION['color'] ?>
    and your favorite food is <?php echo $_SESSION['food'] ?>.

<?php } ?>
```

讨论

在例9-21中的每个页面开始，会把所有提交的表单数据拷贝到$_SESSION中。这样，上一页所提交的表单数据就可以在下一次请求中得以保留，包括第三个页面中显示保存数据的代码都可以使用这些提交的数据。

PHP的session对于完成这种任务再合适不过了，因为session中的所有数据都会保存到服务器中。多页表单可以使每一次请求最小化，即不需要重新提交上一页中已经填写过的资料，而且，减小了验证的开销——每一次都只须验证一部分提交的数据。

参见

技巧11.1中有关session处理的内容。

9.12 重新显示带有内置错误提示的表单

问题

当填写的表单内容有问题时，你想在有问题的表单字段旁显示出相应的错误信息，而不是在表单顶部生成一个通用的错误提示。同时，你还想保留已经在表单中填写的数据，以便他们不用再重新填写那些数据。

方案

在验证表单时，用一个键名为表单元素名的数组保持对表单错误信息的跟踪。当需要显示表单时，在每个问题元素旁边显示错误信息。要保留用户输入的数据，可以使用适当的HTML习惯用法：对于多数<input/>元素使用其value属性（经过实体编码），对于单选按钮和复选框使用checked='checked'属性，对于下拉列表中的<option/>元素使用selected='selected'属性。例9-22用于显示并验证一个带有文本框、复选框和下拉菜单的表单。

例9-22：重新显示带有错误提示并保留输入的表单

```
<?php
// 为下拉菜单建立一些选项
$flavors = array('Vanilla','Chocolate','Rhinoceros');

if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    // 如果请求方法是GET，则只显示表单
    display_form(array());
} else {
    // 如果请求方法是POST，则验证表单
    $errors = validate_form();
    if (count($errors)) {
        // 如果有错误，重新显示带有错误提示的表单
        display_form($errors);
    } else {
        // 如果表单数据全部有效，则向用户发送祝贺信息
        print 'The form is submitted!';
    }
}

function display_form($errors) {
    global $flavors;

    // 建立默认值
    $defaults['name'] = isset($_POST['name']) ? htmlentities($_POST['name']) :
'';
    $defaults['age'] = isset($_POST['age']) ? "checked='checked'" : '';
    foreach ($flavors as $flavor) {
        if (isset($_POST['flavor']) && ($_POST['flavor'] == $flavor)) {
            $defaults['flavor'][$flavor] = "selected='selected'";
        } else {
            $defaults['flavor'][$flavor] = '';
        }
    }
    ?>

<form action='<?php echo $_SERVER['SCRIPT_NAME'] ?>' method='post'>
<dl>
<dt>Your Name:</dt>
<?php print_error('name', $errors) ?>
<dd><input type='text' name='name' value='<?php echo $defaults['name'] ?>'/></d
d>
```

```
<dt>Are you over 18 years old?</dt>
<?php print_error('age', $errors) ?>
<dd><input type='checkbox' name='age' value='1' <?php echo $defaults['age'] ?>/
> Yes</dd>
<dt>Your favorite ice cream flavor:</dt>
<?php print_error('flavor', $errors) ?>
<dd><select name='flavor'>
<?php foreach ($flavors as $flavor) {
    echo "<option {$defaults['flavor'][$flavor]}>$flavor</option>";
} ?>
</select></dd>
</dl>
<input type='submit' value='Send Info'/>
</form>
<?php }

// 为更容易生成HTML错误提示而编写的辅助函数
function print_error($key, $errors) {
    if (isset($errors[$key])) {
        print "<dd class='error'>{$errors[$key]}</dd>";
    }
}

function validate_form() {
    global $flavors;

    // 从没有错误开始
    $errors = array();

    // name是必填的，而且必须至少3个字符
    if (! (isset($_POST['name']) && (strlen($_POST['name']) > 3))) {
        $errors['name'] = 'Enter a name of at least 3 letters';
    }
    if (isset($_POST['age']) && ($_POST['age'] != '1')) {
        $errors['age'] = 'Invalid age checkbox value.';
    }
    // flavor是选填的，但如果提交则必须在$flavors中
    if (isset($_POST['flavor']) && (! in_array($_POST['flavor'], $flavors))) {
        $errors['flavor'] = 'Choose a valid flavor.';
    }

    return $errors;
}
?>
```

讨论

当提交的表单中包含无效数据时，如果能够重新显示在相应字段位置处带有错误提示的表单，而不是在表单的顶部生成一个通用的“这个表单无效”之类的信息，对用户而言会更容易接受。例9-22中的那个validate_form()函数建立了一个包含错误信息的数组，而display_form()则用该错误数组在适当的位置上输出相应的错误提示。

要扩展例9-22需要从两方面着手：一方面需要扩展validate_form()函数中的检查范围，以处理扩展后所增加的验证需求；另一方面，就是要在display_form()函数中增加正确生成HTML元素的代码，以使表单中包含你所需要的输入元素。

参见

技巧9.2～9.9中有关各种表单验证策略的讨论。

9.13 防止多次提交同一表单

问题

你想防止用户多次提交同一个表单。

方案

在表单中添加一个带有唯一值的隐藏字段。在验证表单时，先检查带有该唯一值的表单是否已经提交过了。如果是，拒绝再次提交；如果不是，则处理表单并记录相应的值以备后用。另外，当表单提交后，通过JavaScript来禁用表单的提交按钮。

例9-23使用uniqid()和md5函数在表单中插入了一个值为唯一ID的字段。同时，也在表单的onsubmit处理器属性中设置了一小段JavaScript代码，用于在表单提交后禁用提交按钮。

例9-23：在表单中插入唯一ID字段

```
<form method="post" action="<?php echo $_SERVER['SCRIPT_NAME'] ?>"
      onsubmit="document.getElementById('submit-button').disabled = true;">
<!-- 插入所需的任何正常表单元素-->
<input type='hidden' name='token' value='<?php echo md5(uniqid()) ?>'/>
<input type='submit' value='Save Data' id='submit-button'/>
</form>
```

例9-24根据已保存到SQLite数据库中的数据来检查提交记号，以判断该表单是否已经提交过了。

例9-24：检查是否重复提交了表单

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $db = new PDO('sqlite:/tmp/formjs.db');
    $db->beginTransaction();
    $sth = $db->prepare('SELECT * FROM forms WHERE token = ?');
    $sth->execute(array($_POST['token']));
```

```
    if (count($sth->fetchAll())) {
        print "This form has already been submitted!";
        $db->rollBack();
    } else {
        /* 对其他表单元素进行验证的代码放在此处
         * 在插入记号之前验证全部表单数据 */
        $sth = $db->prepare('INSERT INTO forms (token) VALUES (?)');
        $sth->execute(array($_POST['token']));
        $db->commit();
        print "The form is submitted successfully.";
    }
}
?>
```

讨论

由于种种原因，用户经常会重复提交表单。通常这只是鼠标的误操作，如双击了提交按钮。也可能是为了编辑或者再次核对填写过的信息，点击了浏览器的后退按钮，然后又再次点击了提交按钮而不是浏览器的前进按钮。当然，也可能是故意的——比如，在某项在线调查或者博彩活动中重复投票。我们的解决方案可以防止非恶意的攻击，并能使恶意用户暂时不知所措。然而，它却不能排除所有欺骗性的动机，对此类情况还需要更复杂的工作。

这个解决方案的确可以防止向数据库中插入多条相同记录的副本，而把数据库搞得一团糟。通过在表单中生成一个记号，我们可以识别出表单的每个具体技巧，即使是在禁用cookie的情况下。`uniqid()`函数用于生成一个合格的一次性记号。`md5()`函数不是为了给记号增加随机性，而是为了限制可以使用的字符。`uniqid()`的结果可能是一个由不同的字母和其他字符混合在一起的字符串。而`md5()`的结果则完全由数字和字母`abcdef`构成。至少对于英语国家的用户来说，这样可以保证生成的记号中不会包含漫无边际的词句。

另一种极具诱惑力的方法是不生成随机记号，而是使用一个比已经插入到数据库表中的记录数大1的数字。而这种方法（至少）存在两个问题：首先，它会导致静态条件。如果第二个人在第一个人提交的表单数据尚未处理完成时提交了他的表单会怎么样呢？第二个表单此时就会获得与第一个表单相同的记号，冲突随之出现。这种情况可以通过在请求表单时向数据库中插入一条空白记录的方式来解决。但是，如果用户最终没有提交表单，就会导致数据库中出现空记录。

不采用这种方法的第二原因是，当需要编辑另一条记录而通过手工将其ID值调整为其他数字时，这一过程毫无价值可言。如果你的安全设置有问题，一个假冒的`get`或`post`提交就可能毫不费力地修改你的数据。毕竟一个随机的记号，是不能仅仅通过换成另一个整数就可以猜中的。

参见

技巧18.9中有关通过散列表验证数据的更多内容。uniqid()函数的文档（*http://www.php.net/uniqid*）和md5()函数的文档（*http://www.php.net/md5*）。

9.14 处理上传文件

问题

你想处理一个由用户上传的文件。例如，你正在建设一个共享照片的网站，所以需要存储用户提供的照片。

方案

使用$_FILES数组取得与上传文件相关的信息。例9-25中显示了如何把一个上传的文件保存到Web服务器的*/tmp*文件夹中。

例9-25：上传文件

```
<?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
<form method="post" action="<?php echo $_SERVER['SCRIPT_NAME'] ?>"
      enctype="multipart/form-data">
<input type="file" name="document"/>
<input type="submit" value="Send File"/>
</form>
<?php } else {
    if (isset($_FILES['document']) &&
    ($_FILES['document']['error'] == UPLOAD_ERR_OK)) {
        $newPath = '/tmp/' . basename($_FILES['document']['name']);
        if (move_uploaded_file($_FILES['document']['tmp_name'], $newPath)) {
            print "File saved in $newPath";
        } else {
            print "Couldn't move file to $newPath";
        }
    } else {
        print "No valid file uploaded.";
    }
}
?>
```

讨论

从PHP 4.1开始，所有上传的文件都被保存到自动全局数组$_FILES中。对于表单中的每一个文件元素，都会在$_FILES中建立一个其键为该文件元素名称的数组。例如，在例

9-25的表单中有一个文件元素名为document，所以$_FILES['document']中就包含了上传的文件信息。这种按文件元素建立的数组中都包含五个元素：

name
: 上传的文件名。这个元素的值由浏览器提供，所以可能是完整的路径名，也可能只是一个文件名。

type
: 文件的MIME类型，同样由浏览器提供。

size
: 以字节计算的文件大小，以服务器的计算结果为准。

tmp_name
: 文件上传后保存到服务器中临时位置。

error
: 用于描述在文件上传过程中出现问题（不论什么问题）的错误代码（这个元素只在PHP 4.2.0及更高版本中有效）。

如果你使用的是PHP 4.1之前的版本。那么，上述这些信息会保存在$HTTP_POST_FILES而不是$_FILES中。

其中，error元素可能的值包括：

UPLOAD_ERR_OK (0)
: 上传成功（未发生错误）。

UPLOAD_ERR_INI_SIZE (1)
: 上传文件的大小超过了upload_max_filesize配置指令中限定的值。

UPLOAD_ERR_FORM_SIZE (2)
: 上传文件的大小超过了表单中名为MAX_FILE_SIZE（的input隐藏）元素限定的值。

UPLOAD_ERR_PARTIAL (3)
: 只上传了部分文件。

UPLOAD_ERR_NO_FILE (4)
: 没有发现上传文件。

UPLOAD_ERR_NO_TMP_DIR (6)
: 由于没有临时文件夹保存上传的文件而导致上传失败（在PHP 4.3.10、5.0.3及更高版本中有效）。

UPLOAD_ERR_CANT_WRITE (7)

PHP没有向磁盘写入文件的权限（在PHP 5.1.0及更高版本中有效）。

对于全部错误代码值来说，以上列出的常量在PHP 4.3.0及更高版本中才有效。而在该版本之前的PHP版本中，使用的则是常量后面括号中的数值。

is_uploaded_file()函数用于确认要处理的用户上传文件是一个有效文件。一定要在像处理其他文件一样处理保存在临时位置的文件之前，检查一下tmp_name的值。这样才能保证恶意用户无法通过欺骗代码将一个系统文件上传到服务器。

就和在例9-25中一样，我们可以通过move_uploaded_file()把文件移动到一个固定的位置。该函数也会通过检查来确保移动的的确是上传的文件。注意，tmp_name中的值保存着临时文件的完整路径，而不是基本的文件名。所以在必要的时候可以通过basename()函数去掉该路径前面的目录名。

另外，还要确保检查PHP对保存临时文件的文件夹（通过upload_tmp_dir配置指令设置）和要将文件拷入其中的目标文件夹都拥有读写权限。PHP通常是在一个特殊的用户名之下运行（比如nobody或apache），而不是在你个人的用户名之下运行。为此，如果你同时开启了safe_mode配置指令，那么把一个文件拷贝到新位置后你可能就不能再访问它了。

由于并非所有浏览器提交的信息都完全一致，所以处理文件也是一项很微妙的任务。确保正确无误地完成这项任务非常重要，否则你的麻烦可能会接踵而至。毕竟，开放上传功能就相当于允许陌生人将他们挑选的任何文件上传到你的机器中，而不怀好意的人就可能将其当作进入甚至破坏机器的一个好机会。

有鉴于此，PHP中有许多特性可以让你对上传文件加以限制，直至完全关闭文件上传功能。而且，如果你缺乏处理上传文件的经验，可以检查一个貌似危险分子但又没有被拒绝上传的文件。

要进行这种检查，首先要确定将配置文件中的file_uploads设置为On。接下来，确定你的文件大小没有超过upload_max_filesize的限制 —— 该值默认为2MB（这是为了防止有人试图通过将大型文件填满硬盘而破坏机器）。另外，还有一个post_max_size指令，这个指令用于控制在一次请求中总共能够发送（POST）的最大数据量 —— 初始值为8MB。

从浏览器差异及用户错误的角度来看，如果你不会留意$_FiLES中是否包含预期的信息，那么就要确定在表单的开标签中添加enctype="multipart/form-data"属性。因为PHP需要这一信息才能对文件信息进行适当的处理。

而且，如果在文件元素中并没有选中要上传的文件，在4.1版之前的PHP中会将tmp_name设置为none；更高的版本则将其设置为空字符串。PHP 4.2.1还允许上传大小为0的文件。所以，要想保证上传了文件而且不是空文件（在某种环境下，也可能你需要空文件），就需要确保tmp_name已被设置并且文件尺寸大于0。最后，并非所有浏览器在发送文件时都必须使用相同的MIME类型 —— 它们会根据自己对不同文件类型的判断来发送相应的内容。

参见

有关处理上传文件的文档（*http://www.php.net/features.file-upload*）和basename()函数的文档（*http://www.php.net/basename*）。

9.15 防止全局变量注射

问题

你想访问表单输入变量，但不想让恶意用户随意设置程序中的全局变量。

方案

禁用register_globals配置指令，然后只从数组$_GET、$_POST和$_COOKIE中访问有关变量，以确保只有自己知道要访问的变量保存在哪里。

为此，必须让register_globals = off出现在*php.ini*文件中。

从PHP 4.2开始，这也是默认的设置。

讨论

当register_globals配置指令开启时，外部变量，包括那些来自表单和cookie的变量，都会直接被引入到全局名称空间中。这的确能带来极大的方便，但是如果你不能做到非常细心并坚持不懈地检查变量及它们的来源，那么就可能暴露出许多安全漏洞。为什么这么说呢？因为可能有一个在内部使用的变量，该变量不允许从外部访问，但是这个变量可能在你不知道的情况下被重新设置。

例9-26中包含一个简单的例子 —— 想象一下你有一个需要用户填写用户名和密码的页面。如果被验证通过，你会返回用户的标识代号，并用该数字标识代号查询并输出其个人信息。

例9-26：不安全的 register_globals 代码

```
<?php
// 假设magic_quotes_gpc设置为 off
$username = $dbh->quote($_GET['username']);
$password = $dbh->quote($_GET['password']);

$sth = $dbh->query("SELECT id FROM users WHERE username = $username AND
                    password = $password");

if (1 == $sth->numRows()) {
    $row = $sth->fetchRow(DB_FETCHMODE_OBJECT);
    $id = $row->id;
} else {
    "Print bad username and password";
}

if (!empty($id)) {
    $sth = $dbh->query("SELECT * FROM profile WHERE id = $id");
}
?>
```

正常情况下，变量$id只能由程序设置，并且它是在通过验证之后执行数据库查询的结果。然而，如果有人通过修改查询字符串，给$id传递一个值的话，那么问题就出现了。也就是说，在开启register_globals的情况下，即使用户名和密码查询失败，你的脚本仍然可以执行第二个数据库查询语句并返回结果。而如果禁用register_globals，此时的$id会保持其未设置的状态。因为只有$_REQUEST['id']和$_GET['id']可以被设置。

当然，即使是在开启register_globals的情况下也有办法解决这一问题。你可以重新组织代码来堵住这个漏洞。其中一种方式如例9-27所示。

例9-27：消除register_globals 带来的问题

```
<?php
$sth = $dbh->query("SELECT id FROM users WHERE username = $username AND
                    password = $password");

if (1 == $sth->numRows()) {
    $row = $sth->fetchRow(DB_FETCHMODE_OBJECT);
    $id = $row->id;
    if (!empty($id)) {
        $sth = $dbh->query("SELECT * FROM profile WHERE id = $id");
    }
} else {
    "Print bad username and password";
}
?>
```

在例9-27中，$id只有当它明确地被一个对数据库的调用设置时才会得到赋值。但是，有时候由于程序整体布局的原因，很难做到这一点。另外一种可能的解决方案是在脚本

的开始处手动unset()或初始化所有变量。这样就能将干扰程序的不良$id值拒之门外。但是，由于PHP不需要进行变量的初始化，所以你可能在某个地方会忘记这样做 —— 于是，在没有PHP警告的情况下一个bug就悄悄地出现了。

参见

register_globals配置指令的文档（*http://www.php.net/security.registerglobals.php*）。

9.16 处理名字中带句点的远程变量

问题

你想处理一个名字中有句点的变量，但是当提交表单时，你在$_GET或$_POST中又找不到这些变量。

方案

把变量名字中的句点替换成下划线。例如，如果有一个表单输入元素名字为hot.dog，那么在PHP中就需要以$_GET['hot_dog']或$_POST['hot_dog']来访问它。

讨论

在早期的PHP版本中，register_globals指令在默认情况下是启用的，而且名为hot.dog的表单变量不可能变成$hot.dog —— 因为句点是不允许出现在变量名称当中的。为了解决这个问题，.就被转换成_。虽然$_GET['hot.dog']和$_POST['hot.dog']不存在这个问题，但考虑到传统和一致性，不论register_globals如何设置，仍然会执行这一转换。

通常当表单通过一个image类型的元素来提交时，就会进行这种转换。例如，当点击像<input type="image" name="locations" src="locations.gif">这样的表单元素时，会提交表单。同时，点击位置的x和y坐标也会以location.x和location.y的形式被提交。所以，在PHP中如果想知道用户点击了哪里，需要看的应该是$_POST['locations_x']和$_POST['locations_y']。

参见

有关PHP外部变量的文档（*http://www.php.net/language.variables.external*）。

9.17 使用带有多个选项的表单元素

问题

你有一个可以让用户选择多个选项的表单元素，例如，下拉菜单或一组复选框，但是PHP看起来只取其中一个提交的值。

方案

用一对方括号作为该表单元素名字的结尾。例9-28中是一组命名适当的复选框。

例9-28：命名一组复选框

```
<input type="checkbox" name="boroughs[]" value="bronx"> The Bronx
<input type="checkbox" name="boroughs[]" value="brooklyn"> Brooklyn
<input type="checkbox" name="boroughs[]" value="manhattan"> Manhattan
<input type="checkbox" name="boroughs[]" value="queens"> Queens
<input type="checkbox" name="boroughs[]" value="statenisland"> Staten Island
```

之后，就可以像9-29中那样，将提交的数据作为保存在$_GET或$_POST中的数组来对待。

例9-29：处理一组提交的复选框

```
<?php
print 'I love ' . join(' and ', $_POST['boroughs']) . '!';
?>
```

讨论

把[]放在表单元素名字的末尾，就是要告诉PHP把输入的数据保存到一个数组而不是一个标量中。当PHP发现有多个指定给那个变量的值时，就会将这些值全部保存到一个数组中。如果例9-28中的前三个复选框被选中并提交，那么结果就相当于你在程序的开始位置编写了例9-30中的那些代码一样。

例9-30：与提交的多值表单元素等价的代码

```
<?php
$_POST['boroughs'][] = "bronx";
$_POST['boroughs'][] = "brooklyn";
$_POST['boroughs'][] = "manhattan";
?>
```

类似的语法也可以建立多维数组。例如，表单中有一个复选框<input type="checkbox" name="population[NY][NYC]" value="8008278">。如果该复选框被选中，那么最终这个表单元素会把$_POST['population']['NY']['NYC']的值设置成 8008278。

参见

第4章中有关数组的更详细介绍。

9.18 基于当前日期创建下拉菜单

问题

你想基于当前日期自动地创建一系列下拉菜单项。

方案

使用date()来取得Web服务器所在时区的当前时间，并在循环语句中使用mktime()函数。

例9-31用今天及之后六天的时间值生成了<option/>选项。在这个例子中“今天”指的是2008年12月3日。

例9-31：生成基于日期的下拉菜单项目

```
<?php
list($hour, $minute, $second, $month, $day, $year) =
                                    split(':', date('h:i:s:m:d:Y'));
// 输出对应于一个星期中每一天的选项
for ($i = 0; $i < 7; ++$i) {
    $timestamp = mktime($hour, $minute, $second, $month, $day + $i, $year);
    $date = date("D, F j, Y", $timestamp);
    print "<option value='$timestamp'>$date</option>\n";
}
?>
```

如果以上代码是在2008年12月3日运行的话，那么例9-31的输出结果将是：

```
<option value='1228305600'>Wed, December 3, 2008</option>
<option value='1228392000'>Thu, December 4, 2008</option>
<option value='1228478400'>Fri, December 5, 2008</option>
<option value='1228564800'>Sat, December 6, 2008</option>
<option value='1228651200'>Sun, December 7, 2008</option>
<option value='1228737600'>Mon, December 8, 2008</option>
<option value='1228824000'>Tue, December 9, 2008</option>
```

讨论

在例9-31中，我们之所以把每个日期项目的value设置为Unix时间戳的形式，是因为我们

感觉这样在程序中处理起来会更容易。当然，你可以使用自己认为好用或者合适的任何格式。

不要试图排除对mktime()的调用，因为日期和时间并不能和所你希望的始终保持一致。取决于你想做什么，也许你就得不到自己想要的结果。例9-32采用一种简化的方式——每次只是给时间戳增加一天中的秒数（60秒/分钟×60分钟/小时×24小时/天＝86400秒）。

例9-32：不正确地生成基于日期的下拉菜单选项

```
<?php
$timestamp = mktime(0, 0, 0, 10, 30, 2008); // 2008年10月30日
$one_day = 60 * 60 * 24; // 一天中的分数

// 输出对应于一个星期中每一天的选项
for ($i = 0; $i < 7; ++$i) {
    $date = date("D, F j, Y", $timestamp);
    print "<option value='$timestamp'>$date</option>\n";
    $timestamp += $one_day;
}
?>
```

例9-32的输出结果为：

```
<option value='1225339200'>Thu, October 30, 2008</option>
<option value='1225425600'>Fri, October 31, 2008</option>
<option value='1225512000'>Sat, November 1, 2008</option>
<option value='1225598400'>Sun, November 2, 2008</option>
<option value='1225684800'>Sun, November 2, 2008</option>
<option value='1225771200'>Mon, November 3, 2008</option>
<option value='1225857600'>Tue, November 4, 2008</option>
```

例9-32会输出自2008年10月30日起七天中每一天的月、日和年份。但是，结果却并非如我们所愿。

为什么在菜单项中会出现两个Sun, November 2, 2008？答案就是：夏令时。也就是说，一天中的秒数并不是恒定不变的。实事上，这个数字几乎肯定是会变动的。最糟糕的是，如果你当时并没有处在转换日期附近，很容易在测试中遗漏这个bug。

参见

第3章，特别是技巧3.12，当然还有技巧3.2、3.3、3.4、3.5、3.6、3.11和3.14中的有关内容。date()函数的文档（*http://www.php.net/date*）和mktime()函数的文档（*http://www.php.net/mktime*）。

第10章

访问数据库

10.0 概述

对许多Web应用程序而言，数据库都是其核心所在。数据库几乎可以用来存储你想检索和更新的任何信息，比如用户列表、产品目录或者最新要闻等。PHP之所以能够成为一门伟大的web编程语言的原因之一，就是它对数据库的广泛支持。最新统计表明，PHP可以与20多种数据库结合使用，这些数据库有的是关系型的，有的是非关系型的。PHP可以与之对话的关系型数据库有Apache Derby、DB++、FrontBase、 IBM Cloudscape、IBM DB2、 Informix、 Interbase、Ingres II，Microsoft SQL Server、 mSQL、MySQL、MySQL MaxDB、Oracle、Ovrimos SQL Server、PostgreSQL、SQLite和Sybase等。可以与之对话的非关系型数据库有dBase、filePro、HyperWave、Paradox和flat-file（译注1）类型的DBM数据库系列。而且，PHP还支持ODBC（Open Database Connectivity，开放式数据库连接），所以即使上面没有列出你偏爱的数据库，但只要它支持ODBC，你就可以通过PHP来使用它。

技巧10.1中所讨论的DBM数据库，虽然是一种简单、健壮而且高效的平面文件（flat-file），但它却把数据结构局限为名/值对的形式。如果可以把数据组织成一种名称与值的对应关系，DBM数据库就是一个理想的选择。

PHP真正能显示出强大威力的时候，是在与SQL数据库结合使用的时候。本章中多数技巧采用的都是这种组合。SQL数据库也许是复杂了一些，但它们却极其强大。如果要把PHP与某种特定的SQL数据库结合使用，就必须在编译时明确地告诉PHP包含对相应数据库的支持。如果PHP构建为支持动态模块加载，则对数据库的支持也可以作为动态模块载入。

译注1： flat-file是指没有特殊格式的非二进制的文件，如XML文件等。

本章中有关SQL数据库的例子使用的都是PHP 5的PDO（PHP Data Object，PHP数据源）数据库访问层。通过PDO，就可以在同任何数据库引擎对话的情况下都使用相同的PHP函数。尽管不同数据库的SQL语法可能会不同，但PHP代码总是类似的。在这方面，PDO提供的是对数据访问的抽象，而不是对整个数据库的抽象。其他的PHP库，像PEAR DB（*http://pear.php.net/package/db*）、ADODB（*http://adodb.sourceforge.net/*）及MDB2（*http://pear.php.net/package/MDB2*），则试图解决对整个数据库进行抽象的问题——隐藏不同数据库的实现细节，比如代码层后面的数据处理和列类型等。这种抽象可以在你使用多种不同类型的数据库时为你节省一些工作量，但同时也可能会导致其他问题。如果是针对一种特定类型的数据库编写SQL语句，就可以利用那种数据库的特性获得最好的性能。

PHP 5默认支持的SQLite也是一种强大的数据库，而且不需要单独的服务器。假如你的应用程序通信量适中，而且你也不希望被运行数据库服务器的问题所困扰，SQLite就是一个非常不错的选择。技巧10.2中讨论了一些有关SQLite的细节问题。在PHP4中，可以通过PECL SQLite扩展（*http://pecl.php.net/package/SQLite*）来使用SQLite。

本章中很多SQL的技巧都用到了一个有关黄道十二宫信息的表。表的结构如例10-1所示，而表中的数据则用例10-2表示。

例10-1：*例表的结构*

```
CREATE TABLE zodiac (
  id INT UNSIGNED NOT NULL,
  sign CHAR(11),
  symbol CHAR(13),
  planet CHAR(7),
  element CHAR(5),
  start_month TINYINT,
  start_day TINYINT,
  end_month TINYINT,
  end_day TINYINT,
  PRIMARY KEY(id)
);
```

例10-2：*例表的数据*

```
INSERT INTO zodiac VALUES (1,'Aries','Ram','Mars','fire',3,21,4,19);
INSERT INTO zodiac VALUES (2,'Taurus','Bull','Venus','earth',4,20,5,20);
INSERT INTO zodiac VALUES (3,'Gemini','Twins','Mercury','air',5,21,6,21);
INSERT INTO zodiac VALUES (4,'Cancer','Crab','Moon','water',6,22,7,22);
INSERT INTO zodiac VALUES (5,'Leo','Lion','Sun','fire',7,23,8,22);
INSERT INTO zodiac VALUES (6,'Virgo','Virgin','Mercury','earth',8,23,9,22);
INSERT INTO zodiac VALUES (7,'Libra','Scales','Venus','air',9,23,10,23);
INSERT INTO zodiac VALUES (8,'Scorpio','Scorpion','Mars','water',10,24,11,21);
INSERT INTO zodiac VALUES (9,'Sagittarius','Archer','Jupiter','fire',11,22,12,21);
```

```
INSERT INTO zodiac VALUES (10,'Capricorn','Goat','Saturn','earth',12,22,1,19);
INSERT INTO zodiac VALUES (11,'Aquarius','Water Carrier','Uranus','air',1,20,2,18);
INSERT INTO zodiac VALUES (12,'Pisces','Fishes','Neptune','water',2,19,3,20);
```


技巧10.3～10.8涵盖了连接到数据库服务器、发送请求及取回结果的基本知识，也使用了修改数据库中数据的查询语句。

典型的PHP程序都是通过HTML表单的字段来获得信息，并把信息保存到数据库中的。这些信息中的某些字符，例如，单引号和反斜线，在SQL中都具有特殊的含义，所以如果表单的数据中包含这类字符，就要格外小心。PHP有一个称之为“魔术引号”的功能，可以帮我们处理这一问题。如果在配置文件中将magic_quotes_gpc设置为on，那么来自get请求、post请求及cookie中带单引号、双引号、反斜线及值为null的数据都会通过一个反斜线来转义。也可以把magic_quotes_runtime项设置为on，达到自动将外部数据源（例如，数据库查询或文本文件）中的引号、反斜线及null值转义的目的。例如，如果magic_quotes_runtime设置为on，而且我们使用了file()把一个文件读到一个数组中，那么最终数组中的那些特殊字符都会用反斜线转义。

令人遗憾的是，“魔术引号”所带来的问题使它更像是“麻烦引号”。如果想在其他非SQL查询的环境中使用提交的表单数据，比如在页面中显示这些数据，必须要撤销转义，才能保证页面正常显示。你要是碰到一个PHP网站的文本输入框中，出现了单引号前面带反斜线的情形，那几乎可以肯定就是魔术引号搞的怪。技巧10.7中介绍了PDO对绑定参数（bound parameters）的支持，那是一种确保把用户输入的特殊字符组合成SQL查询语句的更好方式。技巧10.9会更详细地讨论查询语句中的特殊字符转义问题。而用于处理数据库查询结果中错误的常规调试技术会在技巧10.10中介绍。

本章中其他的技巧涉及到了更复杂的数据库操作，而不仅仅只是简单查询。技巧10.11展示了如何自动地生成可以作为记录标识符的唯一ID值。技巧10.12则涉及在程序运行过程中由字段列表组合成查询。这个技巧可以使涉及到多个列的INSERT和UPDATE查询更容易处理。技巧10.13示范了如何显示让页面能够遍历一个结果集的链接，以及在每一页上都显示一些记录。要想改善对数据库访问的速度，可以像技巧10.14所说明的那样缓存查询及其结果。

技巧10.15展示的是在一个大型程序中管理在不同环境下访问一个单独数据库连接的相关技术。技巧10.16把本章所讨论的一些技术综合到了一个完整的程序中，这个程序用于把主题留言板中的信息保存到一个数据库中。

10.1 使用DBM数据库

问题

你有一些很适合用键/值对形式来表示的数据，你想把这些数据安全地存储起来，并且希望能够基于那些键来快速地找到相关信息。

方案

使用DBA抽象层访问一个DBM式的数据库，如例10-3所示。

例10-3：使用DBM数据库

```
<?php
$dbh = dba_open('fish.db','c','gdbm') or die($php_errormsg);

// 取得并修改值
if (dba_exists('flounder',$dbh)) {
  $flounder_count = dba_fetch('flounder',$dbh);
  $flounder_count++;
  dba_replace('flounder',$flounder_count, $dbh);
  print "Updated the flounder count.";
} else {
  dba_insert('flounder',1, $dbh);
  print "Started the flounder count.";
}

// 没有罗非鱼了
dba_delete('tilapia',$dbh);

// 还有什么鱼?
for ($key = dba_firstkey($dbh);  $key !== false; $key = dba_nextkey($dbh)) {
   $value = dba_fetch($key, $dbh);
   print "$key: $value\n";
}

dba_close($dbh);
?>
```

讨论

PHP可以支持少数不同种类的DBM后台：GDBM、NDBM、DB2、DB3、DBM和CDB。而DBA抽象层可以让我们在任何DBM后台之上都使用相同的函数。所有这些后台都是以键/值对的形式存储数据。可以迭代遍历数据库中所有的键，并取得与特定键关联的值，并且还能检测某个特定的键是否存在。不论键还是值，都是字符串。

例10-4中的程序维护的是一个存储在DBM数据库中的用户名和密码的列表。用户名作

为第一个命令行参数，密码则是第二个参数。如果这个数据库中已经存在了给定的用户名，就把密码修改为给定的密码；否则，就把用户名与密码的组合添加到数据库中。

例10-4：用DBM数据库实现对用户名和密码的跟踪

```
<?php
$user = $_SERVER['argv'][1];
$password = $_SERVER['argv'][2];

$data_file = '/tmp/users.db';

$dbh = dba_open($data_file,'c','gdbm') or die("Can't open db $data_file");

if (dba_exists($user,$dbh)) {
    print "User $user exists. Changing password.";
} else {
    print "Adding user $user.";
}

dba_replace($user,$password,$dbh) or die("Can't write to database $data_file");

dba_close($dbh);
?>
```

dba_open()函数返回一个对DBM文件的引用句柄（或在出错时返回false）。它带有三个参数。第一个是DBM文件的文件名。第二个是打开文件的模式，其中，r表示以只读的方式打开存在的数据库；w表示以读/写方式打开存在的数据库；c表示以读/写方式打开一个数据库，并且如果这个数据库不存在就创建它；最后，n与c的含义相同，只不过如果数据库存在就会清空数据库。dba_open()的第三个参数表示要使用哪一个DBM处理程序；例子中使用的是“gdbm”。要想知道在你的PHP安装程序中都编译了哪些DBM处理程序，可以调用phpinfo()方法，查看其输出结果中的“DBA”部分。在“支持的处理程序”中会列出所有的选项。

要知道DBM数据库中是否存在一个键，用dba_exists()。这个函数带两个参数：一个键名字符串和一个DBM文件句柄。它会查找DBM数据库文件中是否存在这个键，如果存在就返回true（否则返回false）。dba_replace()函数带有三个参数：一个键名字符串、一个字符串值和一个DBM文件句柄。这个函数会把键/值对发送到数据库中。如果现有条目中已经存在给定的键名了，它就用新的值覆盖那个条目。

要关闭数据库时，调用dba_close()。通过dba_open()打开的DBM文件会在一个请求结束后自动关闭，但是，我们仍然需要明确地调用dba_close()来关闭由dba_open()建立的持续性连接。

可以使用dba_firstkey()和dba_nextkey()来迭代遍历一个DBM文件中的所有键，使用

dba_fetch()取得每个键对应的值。例10-5中的程序用于计算一个DBM文件中所有密码的长度之和。

例10-5：计算DBM文件中密码的长度之和

```
<?php
$data_file = '/tmp/users.db';
$total_length = 0;
if (! ($dbh = dba_open($data_file,'r','gdbm'))) {
    die("Can't open database $data_file");
}

$k = dba_firstkey($dbh);
while ($k) {
    $total_length += strlen(dba_fetch($k,$dbh));
    $k = dba_nextkey($dbh);
}

print "Total length of all passwords is $total_length characters.";

dba_close($dbh);
```

dba_firstkey()函数把$k初始化为DBM文件的第一个键。通过while循环，每次都由dba_fetch()取得键$k对应的值，并把这个值的长度（用stren()计算的）加到$total_length上面。然后，再通过dba_nextkey()把$k设为文件中的下一个键。

要在DBM数据库中存储复杂数据的一种方式是使用serialize()函数。例10-6在把用户信息存储到DBM数据库之前，先对数据结构进行序列化，然后再将结构化的用户信息保存到数据库文件中。在要获取这些信息时，再进行反序列化操作。

例10-6：在一个DBM数据库中存储结构化的数据

```
<?php
$dbh = dba_open('users.db','c','gdbm') or die($php_errormsg);

// 读取并反序列化数据
if ($exists = dba_exists($_POST['username'], $dbh)) {
    $serialized_data = dba_fetch($_POST['username'], $dbh) or die($php_errormsg);
    $data = unserialize($serialized_data);
} else {
    $data = array();
}

// 更新键值
if ($_POST['new_password']) {
    $data['password'] = $_POST['new_password'];
}
$data['last_access'] = time();

// 把数据写回文件
if ($exists) {
```

```
    dba_replace($_POST['username'],serialize($data), $dbh);
} else {
    dba_insert($_POST['username'],serialize($data), $dbh);
}

dba_close($dbh);
?>
```

例10-6虽然能在同一个文件中保存多个用户数据，但你却不可能在不循环迭代文件中每个键的情况下搜索到一个用户的最后访问时间。要是想完成此类搜索，就必须把数据存储在SQL数据库中。

每个DBM处理程序在某些情况下可能具有不同的行为。例如，GDBM提供内部锁定。如果一个进程已经以读/写模式打开了一个GDBM文件，那么其他调用dba_open()函数试图以读/写模式打开同一个文件的行为都会失败。另外一个DBM处理程序，当在向dba_open()传递的模式参数中加上一个l时，它会用一个单独的*.lck*文件锁定数据库，而加上一个d时则会锁定数据库本身。还有两个DBA函数 —— dba_optimize()和dba_sync() —— 也是与数据库相关的。其中，dba_optimize()函数调用一个与处理程序相关的DBM文件优化函数。目前，这个功能仅在GDBM中实现了，其中调用的是它的gdbm_reorganize()函数。而dba_sync()函数会调用一个与处理程序相关的DBM文件同步函数。对于DB 2和DB3，调用的是各自的sync()函数。对于GDBM，调用的是gdbm_sync()函数。对于其他的DBM处理程序，则不会有什么函数被调用。

使用DBM数据库是对使用纯文本文件的一种提升，但却缺少SQL数据库具有的大多数功能。主要是数据结构局限于键/值对形式和锁定功能不够健壮，而且非常依赖于DBM处理程序。即便如此，DBM处理程序对于频繁访问只读数据的应用来说仍然是一个不错的选择。

参见

技巧5.7中讨论的序列化数据主题；DBA函数的文档（*http://www.php.net/dba.*）；要了解关于DB2和DB3处理程序的更多内容，请访问*http://www.sleepycat.com/products/bdb.html*（注意这些处理程序对商业应用通常不是免费的）；要了解GDBM，可以访问*http://www.gnu.org/directory/gdbm.html*或*http://www.mit.edu:8001/afs/athena.mit.edu/project/gnu/doc/html/gdbm_toc.html*。

10.2 使用SQLite数据库

问题

你想使用一个不涉及独立的服务器进程的关系型数据库。

方案

使用SQLite —— 这个与PHP 5捆绑发布的健壮、强大，又不需要运行独立服务器的数据库。一个SQLite数据库就是一个文件。例10-7创建了一个SQLite数据库，在数据库中没有相应表的情况下创建表，并向表中插入一些数据。

10-7：创建一个SQLite数据库

```
<?php
$db = new PDO('sqlite:/usr/local/zodiac');

// 创建表并自动插入数据
$db->beginTransaction();
// 试着查找名称为zodiac的表
$q = $db->query("SELECT name FROM sqlite_master WHERE type = 'table'" .
                " AND name = 'zodiac'");
// 如果查询中没有返回结果行，就创建这个表并插入数据
if ($q->fetch() === false) {
    $db->exec(<<<_SQL_
CREATE TABLE zodiac (
  id INT UNSIGNED NOT NULL,
  sign CHAR(11),
  symbol CHAR(13),
  planet CHAR(7),
  element CHAR(5),
  start_month TINYINT,
  start_day TINYINT,
  end_month TINYINT,
  end_day TINYINT,
  PRIMARY KEY(id)
)
_SQL_
);

    // 独立的SQL语句
    $sql=<<<_SQL_
INSERT INTO zodiac VALUES (1,'Aries','Ram','Mars','fire',3,21,4,19);
INSERT INTO zodiac VALUES (2,'Taurus','Bull','Venus','earth',4,20,5,20);
INSERT INTO zodiac VALUES (3,'Gemini','Twins','Mercury','air',5,21,6,21);
INSERT INTO zodiac VALUES (4,'Cancer','Crab','Moon','water',6,22,7,22);
INSERT INTO zodiac VALUES (5,'Leo','Lion','Sun','fire',7,23,8,22);
INSERT INTO zodiac VALUES (6,'Virgo','Virgin','Mercury','earth',8,23,9,22);
INSERT INTO zodiac VALUES (7,'Libra','Scales','Venus','air',9,23,10,23);
INSERT INTO zodiac VALUES (8,'Scorpio','Scorpion','Mars','water',10,24,11,21);
```

```
INSERT INTO zodiac VALUES (9,'Sagittarius','Archer','Jupiter','fire',11,22,12,21);
INSERT INTO zodiac VALUES (10,'Capricorn','Goat','Saturn','earth',12,22,1,19);
INSERT INTO zodiac VALUES (11,'Aquarius','Water Carrier','Uranus','air',1,20,2,18);
INSERT INTO zodiac VALUES (12,'Pisces','Fishes','Neptune','water',2,19,3,20);
_SQL_;

// 将SQL语句按行分割并逐一执行
foreach (explode("\n",trim($sql)) as $q) {
        $db->exec(trim($q));
    }
    $db->commit();
} else {
    // 什么也不做，结束事务
    $db->rollback();
}
?>
```

讨论

由于SQLite数据库只是常规的文件，所以在PHP中所有应用到文件的预防措施和转向（gotcha）对SQLite数据库也同样适用。PHP进程的用户同样必须在得到许可后才能从SQLite数据库所在的位置读取或向该位置写入信息。而把SQLite数据库存放在web服务器文档根目录之外的某个地方,是一个非常好的主意。如果可以通过Web服务器直接读取数据库文件，那么能推测出其位置的用户就可以获取整个文件，从而能够绕开你在PHP程序中内置于查询里的所有限制。

在PHP中，`sqlite`扩展为SQLite 2提供了常规的SQLite访问功能及PDO驱动程序。而`pdo_sqlite`扩展则为SQLite 3提供了PDO驱动程序。如果你是从头开始，那就使用针对SQLite 3提供的PDO驱动程序，因为它不仅速度快而且还提供更多的功能。如果你已经有了一个SQLite 2数据库，可以考虑使用PDO驱动程序将其移植到SQLite 3。

例10-7所引用的`sqlite_master`表是一个特殊的系统表，这个表中保存着关于其他表的信息，所以它对于确定某个表是否存在相当有用。其他的数据库也都有它们自己的方式提供这种系统元数据。

参见

SQLite文档（*http://www.sqlite.org/docs.html*）和`sqlite_master`文档（*http://www.sqlite.org/faq.html#q9*）。

10.3 连接到SQL数据库

问题

你想连接到一个SQL数据库以便存储或抽取信息。没有数据库，动态网站也不会特别名副其实。

方案

用适当的连接字符串创建一个新的PDO对象。例10-8显示了为一些不同类型的数据库创建PDO对象的细节。

例10-8：使用PDO连接

```
<?php
// MySQL 希望在字符串中嵌入参数
$mysql = new PDO('mysql:host=db.example.com', $user, $password);
// 用；分隔多个参数
$mysql = new PDO('mysql:host=db.example.com;port=31075', $user, $password)
$mysql = new PDO('mysql:host=db.example.com;port=31075;dbname=food', $user,$password)
// 连接到一个本地 MySQL 服务器
$mysql = new PDO('mysql:unix_socket=/tmp/mysql.sock', $user, $password)

// PostgreSQL 也希望在字符串中嵌入参数
$pgsql = new PDO('pgsql:host=db.example.com', $user, $password);
// 但是它用"来分隔多个参数
$pgsql = new PDO('pgsql:host=db.example.com port=31075', $user, $password)
$pgsql = new PDO('pgsql:host=db.example.com port=31075 dbname=food', $user,$password)
// 如果愿意，可以把用户名和密码放在DSN中
$pgsql = new PDO("pgsql:host=db.example.com port=31075 dbname=food user=$user password
=$password");

// Oracle
// 如果数据库名是在tnsnames.ora中定义的，只要把它放在DSN中即可
$oci = new PDO('oci:food', $user, $password)
// 否则，就要指定一个瞬时客户端URI
$oci = new PDO('oci:dbname=//db.example.com:1521/food', $user, $password)

// Sybase (如果PDO使用的是FreeTDS)
$sybase = new PDO('sybase:host=db.example.com;dbname=food', $user, $password)
// Microsoft SQL Server (如果 PDO使用的是MS SQL Server库)
$mssql = new PDO('mssql:host=db.example.com;dbname=food', $user, $password);
// DBLib (针对DB-lib的其他版本)
$dblib = new PDO('dblib:host=db.example.com;dbname=food', $user, $password);

// ODBC —— 一个预定义的连接
$odbc = new PDO('odbc:DSN=food');
// ODBC —— 一个特定的连接，当底层驱动程序需要时提供
$odbc = new PDO('odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=
C:\\data\\food.mdb;Uid=Chef');
```

```
// SQLite 只需要一个文件名 —— 不需要用户名和密码
$sqlite = new PDO('sqlite:/usr/local/zodiac.db');
$sqlite = new PDO('sqlite:c:/data/zodiac.db');
// SQLite 也可以处理内存、临时数据库
$sqlite = new PDO('sqlite::memory:');
// SQLite v2 的DSN与v3的DSN看起来很相似
$sqlite2 = new PDO('sqlite2:/usr/local/old-zodiac.db');
?>
```

讨论

如果一切顺利，PDO构造函数会返回一个用于查询相应数据库的新对象。如果出现什么问题，则会抛出一个PDOException异常。

我们在例10-8中已经看到了，DSN的格式高度依赖于你所要连接的数据库类型。但一般来说，PDO构造函数中的第一个参数都是一个描述所要连接数据库位置和名称的字符串，而第二和第三个参数则是要连接数据库的用户名及密码。要注意的是，如果使用的是一个特殊的PDO后台，PHP则必须要建立对那个后台的支持。根据phpinfo()函数的输出结果，可以确定你所安装的PHP支持哪些后台。

参见

技巧10.4中有关查询一个SQL数据库的内容；技巧10-6中有关修改SQL数据库的内容；PDO的文档（*http://www.php.net/PDO*）。

10.4 查询一个SQL数据库

问题

你想从数据库中抽取一些数据。

方案

使用PDO::query()将SQL查询发送到数据库，然后再用foreach循环抽取结果集中的每一行记录，如例10-9所示。

例10-9：向数据库发送一个查询

```
<?php
$st = $db->query('SELECT symbol,planet FROM zodiac');
foreach ($st->fetchAll() as $row) {
    print "{$row['symbol']} goes with {$row['planet']} <br/>\n";
```

```
}
?>
```

讨论

其中的query()方法返回一个PDOStatement对象。该对象的fetchAll()方法提供了对一个查询所返回的每一行记录进行操作的简洁方式。

而fetch()方法每次只返回一行记录，如例10-10所示。

例10-10：获取个别的行

```
<?php
$rows = $db->query('SELECT symbol,planet FROM zodiac');
$firstRow = $rows->fetch();
print "The first results are that {$row['symbol']} goes with {$row['planet']}";
?>
```

每次调用fetch()都返回结果集中的下一条记录。如果下一条记录不存在，fetch()返回false。

在默认情况下，fetch()会返回两次包含着记录中每一列信息的数组 —— 第一次的数组以列名为索引，第二次的数组则以数字为索引。也就是说，例10-10中的变量$firstRow有四个元素：$firstRow[0]是Ram，$firstRow[1]是Mars，$firstRow['symbol']是Ram，而 $firstRow['planet']是Mars。

要想让fetch()返回的记录具有不同的格式，需要给query()传递PDO::FETCH_*常量作为其第二个参数。或者，也可以将某个常量传递给fetch()作为其第一个参数。有效的常量及它会令fetch()返回什么样的记录如表10-1所示。

表10-1：PDO::FETCH_* 常量

常量	记录格式
PDO::FETCH_BOTH	数组，同时包含数字和字符串（列名）键。默认格式
PDO::FETCH_NUM	带数字键的数组
PDO::FETCH_ASSOC	带字符串（列名）键的数组
PDO::FETCH_OBJ	以列名作为属性名的stdClass类的对象
PDO::FETCH_LAZY	以列名作为属性名的PDORow类的对象。其属性只有在访问时才会被赋值，所以如果记录行中包含的列很多，这种格式倒是个不错的选择。注意，当你在保存了返回的对象之后又取得另一行记录时，保存的对象会被新记录的值更新

除了表10-1中的这些可选项之外，还可以通过其他方式构造返回的记录行。不过，这些方式都需要向query()或fetch()中传递不止一个常量。

在与bindColumn()组合使用的时候，PDO::FETCH_BOUND取出模式可以让你设置变量，这些变量的值会在每次调用fetch()时自动更新。例10-11演示了这个过程。

例10-11：绑定结果记录中的列

```
<?php
$row = $db->query('SELECT symbol,planet FROM zodiac',PDO::FETCH_BOUND);
// 将 'symbol' 列的值放到 $symbol变量中
$row->bindColumn('symbol', $symbol);
// 将第二列('planet')的值放到 $planet变量中
$row->bindColumn(2, $planet);
while ($row->fetch()) {
    print "$symbol goes with $planet. <br/>\n";
}
?>
```

在例10-11中，每当调用fetch()时，$symbol和$planet都会被赋予新值。注意，在bindColumn()中既可以使用列名也可以使用列数。列数从1开始算起。

当与query()共同使用时，PDO::FETCH_INTO和PDO::FETCH_CLASS常量会将结果行保存到特定类的一个专门对象中。要使用这两种模式，首先要创建一个扩展了内置PDOStatement类的类。

例10-12中创建一个扩展了PDOStatement类的类，该类带有一个报告所有列值平均长度的方法。然后又通过查询来使用了这个类。

例10-12：扩展PDOStatement类

```
<?php
class AvgStatement extends PDOStatement {
    public function avg() {
        $sum = 0;
        $vars = get_object_vars($this);
        // 删除 PDOStatement 内置的 'queryString' 变量
        unset($vars['queryString']);
        foreach ($vars as $var => $value) {
            $sum += strlen($value);
        }
        return $sum / count($vars);
    }
}
$row = new AvgStatement;
$results = $db->query('SELECT symbol,planet FROM zodiac',PDO::FETCH_INTO, $row)
;
// 每次调用fetch()时，对象 $row都会被注入新值
while ($results->fetch()) {
    print "$row->symbol belongs to $row->planet (Average: {$row->avg()}) <br/>\n";
```

```
}
?>
```

在例10-12中，query()的第二个和第三个参数告诉PDO，“每当你取得一条新记录时，就用该记录中的值来更新$row变量的属性”。而在while()循环内部，$row的属性都是可用的，同时也包括刚刚定义的avg()方法。

如果你想将数据都保存到同一个对象中（比如，可能你想在对fetch()的所有调用过程中显示奇数或者偶数行记录），可以使用PDO::FETCH_INTO。但是，如果你想将每一条记录都用一个新对象来保存，那么就要使用PDO::FETCH_CLASS。将它像PDO::FETCH_INTO一样传递到query()中，但要保证传递给query()的第三个参数是一个类名，而非一个对象技巧。而且，与PDO::FETCH_CLASS一并提供的类必须扩展了PDOStatement类。

参见

技巧10.5中讨论的抽取数据的其他方式。技巧10.6中有关修改SQL数据库的内容;技巧10.7中有关重复查询效率问题的讨论;PDO的文档（*http://www.php.net/PDO*）。

10.5 不通过循环抽取记录

问题

你想通过一种简练的方式来执行一个查询，并抽取返回的数据。

方案

使用fetchAll()一次性取得查询返回的所有结果记录，如例10-13所示。

例10-13：一次得到所有结果

```
<?php
$st = $db->query('SELECT planet, element FROM zodiac');
$results = $st->fetchAll();
foreach ($results as $i => $result) {
   print "Planet $i is {$result['planet']} <br/>\n";
}
?>
```

讨论

当需要根据一个查询返回的所有记录来决定做什么时，fetchAll()方法是很有用的，诸如计算总共有多少条记录或者以次序颠倒的方式对记录进行处理等。同fetch()相似的

是，fetchAll()在默认情况下也会用同时包含数字键和字符串键的数组来表示每一条记录，并且也可以接受多种PDO::FETCH_*常量以改变默认的返回结果。

此外，fetchAll()还能接受其他一些影响其返回结果的常量。比如，如果只想从结果中抽取一列数据的值，那么就可以传递PDO::FETCH_COLUMN及另一个参数，即所需列的索引。但是列数是从0开始，而不是从1开始。

参见

技巧10.4中有关查询SQL数据库及更多获取模式的信息；10.6中有关修改SQL数据库的介绍；10.7中有关重复查询效率问题的讨论；PDO的文档（*http://www.php.net/PDO*）。

10.6 修改SQL数据库中的数据

问题

你想添加、删除或更改SQL数据库中的数据。

方案

使用PDO::exec()来发送一个INSERT、DELETE或UPDATE命令，如例10-14所示。

例10-14：使用PDO::exec()

```
<?php
$db->exec("INSERT INTO family (id,name) VALUES (1,'Vito')");

$db->exec("DELETE FROM family WHERE name LIKE 'Fredo'");

$db->exec("UPDATE family SET is_naive = 1 WHERE name LIKE 'Kay'");
?>
```

还可以使用PDO::prepare()来准备一条查询语句，并通过PDOStatement::execute()来执行该查询，如例10-15所示。

例10-15：准备并执行一个查询

```
<?php
$st = $db->prepare('INSERT INTO family (id,name) VALUES (?,?)');
$st->execute(array(1,'Vito'));

$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));
```

```
$st = $db->prepare('UPDATE family SET is_naive = ? WHERE name LIKE ?');
$st->execute(array(1,'Kay');
?>
```

讨论

exec()方法会把传递给它的任何内容都发送给数据库。对于INSERT，UPDATE和DELETE查询，该方法返回的是被相应查询影响的行数。

而prepare()和execute()方法则对于想要多次执行的查询非常有用。一旦准备好查询语句，就可以用新值重复使用该查询而不用重新再准备。例10-16中先后三次使用了准备好的同一个查询语句。

例10-16：重用准备好的语句

```
<?php
$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));
$st->execute(array('Sonny'));
$st->execute(array('Luca Brasi'));
?>
```

参见

10.7中有关重复查询的信息；PDO::exec()方法的文档（*http://www.php.net/PDO::exec*）；PDO::prepare()方法的文档（*http://www.php.net/PDO::prepare*）和PDOStatement::execute()方法的文档（*http://www.php.net/PDOStatement::execute*）。

10.7 有效地重复查询

问题

你想要多次运行同一条查询语句，但每次都要代入不同的值。

方案

用PDO::prepare()来建立查询，然后对由prepare()返回的、准备好的语句调用execute()来执行。如例10-17所示，execute()会以不同的数据替换传递到prepare()中查询语句里面的占位符。

例10-17：运行准备好的语句

```
<?php
// 准备
$st = $db->prepare("SELECT sign FROM zodiac WHERE element LIKE ?");
// 执行一次
$st->execute(array('fire'));
while ($row = $st->fetch()) {
    print $row[0] . "<br/>\n";
}
// 再执行一次
$st->execute(array('water'));
while ($row = $st->fetch()) {
    print $row[0] . "<br/>\n";
}
?>
```

讨论

这个传递到execute()中的值称为绑定参数 —— 每个值都会与查询语句中的占位符关联（或“绑定”）。绑定参数的两大优点是安全和快速。使用绑定参数，就不用再担心SQL注射攻击了。PDO可以适当地对每个参数添加引号或者转义其中的特殊字符，因此特殊的字符都会得到控制。而且，由于建立在prepare()之上，许多数据库后台都会对查询进行解析和优化，所以每次调用execute()时的速度要快于以自己建好的字符串形式的完整查询语句来调用exec()或者query()的速度。

在例10-17中，execute()第一次执行了“SELECT sign FROM zodiac WHERE element LIKE 'fire'”，第二次执行的是“SELECT sign FROM zodiac WHERE element LIKE 'water'”。

每一次，execute()都会用其参数中的值替换?占位符。如果有多个占位符，就需要按照它们在查询语句中的先后顺序将用于替换的参数值放到数组中。例10-18显示了在有两个占位符时的prepare()和execute()的使用情况。

例10-18：多个占位符

```
<?php
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element LIKE ? OR planet LIKE ?");

// SELECT sign FROM zodiac WHERE element LIKE 'earth' OR planet LIKE 'Mars'
$st->execute(array('earth','Mars'));
?>
```

除了?占位符，PDO也支持命名的占位符。如果需要在查询语句中放入很多占位符的话，使用命名的占位符会更容易理解。放在查询语句中的命名占位符必须以一个冒号开

头，而在传递给execute()的数组中使用的键则是不带冒号的占位符名。例10-19显示了如何使用命名的占位符。

例10-19：使用命名的占位符

```
<?php
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element LIKE :element OR planet LIKE :planet");
// SELECT sign FROM zodiac WHERE element LIKE 'earth' OR planet LIKE 'Mars'
$st->execute(array('planet' => 'Mars', 'element' => 'earth'));
$row = $st->fetch();
```

使用命名占位符不仅能使查询语句更容易理解，而且提供给execute()的值也可以按照任何顺序出现。但是，必须要注意的是每个占位符名只能在一条查询语句中出现一次。如果在一条查询语句中需要多次使用同一个命名占位符，也仍然需要使用两个不同的占位符名称，而后在传递给execute()的数组中对应地添加两个元素。

除了?和命名占位符以外，prepare()还提供了在查询语句中插入值的第三种方式：bindParam()。该方法可以自动地将变量中的值关联到特定的占位符。例10-20显示了bindParam()的用法。

例10-20：使用binkParam()

```
<?php
$pairs = array('Mars' => 'water',
               'Moon' => 'water',
               'Sun' => 'fire');
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element LIKE :element AND planet LIKE :planet");
$st->bindParam(':element', $element);
$st->bindparam(':planet', $planet);
foreach ($pairs as $planet => $element) {
    // 不需要给execute()传递参数——
    // 相应的值来自$element和$planet
    $st->execute();
    var_dump($st->fetch());
}
?>
```

在例10-20中，不需要给execute()传递任何值。两次对binkParam()的调用就相当于告诉PDO："不管什么时候执行$st，都使用变量$element中的值替换:element占位符，使用变量$planet中的值替换:planet占位符"。在调用bindParam()时，这两个变量中保存着什么值并不重要 —— 只有在调用execute()时这两个变量中的值才会用到。因为foreach语句把数组的键放在了变量$planet中，把数组的值放在了变量$element中，所以数组$pairs中的键和值就分别被代换到了查询语句中。

如果在prepare()中使用的是?占位符，那么bindParam()的第一个参数应该是一个表示占位符位置的数字，而不是一个占位符名。占位符位置起始于1，而不是0。

bindParam()会基于所提供值的PHP类型来决定如何处理这些值。通过给bindParam()传递一个类型常量作为第三个参数，可以强制bindParam()将相应的值作特定的类型来处理。bindParam()可以理解的类型常量如表10-2所示。

表10-2：PDO::PARAM_* 常量

常量	类型
PDO::PARAM_NULL	NULL
PDO::PARAM_BOOL	布尔
PDO::PARAM_INT	整数
PDO::PARAM_STR	字符串
PDO::PARAM_LOB	“大型对象”

其中，PDO::PARAM_LOB类型尤其方便，因为该常量会强制bindParam()将变量的值作为流来处理。这样就为将文件内容（或者其他能够表现为流内容的对象，例如，远程URL等）填充到数据库表中提供了一种有效的方法。例10-21中使用glob()将一个字典中所有文件的内容转移到了一个数据库表中。

例10-21：通过PDO::PARAM_LOB将文件内容保存到数据库中

```
<?php
$st = $db->prepare('INSERT INTO files (path,contents) VALUES (:path,:contents)');
$st->bindParam(':path',$path);
$st->bindParam(':contents',$fp,PDO::PARAM_LOB);
foreach (glob('c:/documents/*.*') as $path) {
    // 取得PDO::PARAM_LOB所表示的文件句柄
    $fp = fopen($path,'r');
    $st->execute();
}
?>
```

是否能够有效地利用PDO::PARAM_LOB取决于你的底层数据库。例如，对于Oracle数据库而言，查询语句必须创建一个空的LOB句柄，而且必须是在一个事务当中。具体语法可以参考PDO文档页面（*http://www.php.net/PDO*）中给出的例子“Inserting an image into a database:Oracle”。

参见

PDO::prepare()方法的文档（*http://www.php.net/PDO::prepare*），PDOStatement::execute()方法的文档（*http://www.php.net/PDOStatement::execute*），以及PDO::

bindParam()方法的文档（*http://www.php.net/PDO::bindParam*）和PDO::PARAM_LOB常量的文档（*http://www.php.net/PDO*中“Large Objects”部分）。

10.8 确定查询返回的行数

问题

你想知道一个SELECT查询返回了多少行数据，或者想知道一个INSERT、UPDATE或DELETE查询改变了多少行数据。

方案

如果你是使用PDO::exec()来发送INSERT、UPDATE或 DELETE，那么exec()方法的返回值就是被更新的行数。

如果你是使用PDO:prepare()和PDOStatement::execute()来发送INSERT、UPDATE或DELETE，那么可以像例10-22中所示的那样使用PDOStatement::rowCount()来取得被更新的行数。

例10-22：用rowCount()方法来取得行数

```
<?php
$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));
print "Deleted rows: " . $st->rowCount();
$st->execute(array('Sonny'));
print "Deleted rows: " . $st->rowCount();
$st->execute(array('Luca Brasi'));
print "Deleted rows: " . $st->rowCount();
?>
```

如果你发送的是一个SELECT语句，那么确定返回了多少行的唯一可靠方式是用fetchAll()来抽取返回的全部数据，之后再计算其中包含多少行记录。计算过程如例10-23所示。

例10-23：计算由SELECT语句返回的行数

```
<?php
$st = $db->query('SELECT symbol,planet FROM zodiac');
$all= $st->fetchAll(PDO::FETCH_COLUMN, 1);
print "Retrieved ". count($all) . " rows";
?>
```

讨论

尽管有的数据库后台可以给PDO提供有关SELECT查询返回行数的信息，以使得rowCount()可以在那些数据库的环境下使用，但并非所有数据库都会提供这一信息。所以，依赖于这一特性是不明智的。

然而，抽取一个包含一切的大结果集的效率是很低的。另一种解决方案就是使用COUNT(*)函数来让数据库计算返回结果集的大小。即仍然可以使用与其他情况下相同的WHERE子句，只不过让SELECT返回COUNT(*)的结果而不是字段列表。

参见

PDO::rowCount()方法的文档（PDO::rowCount at *http://www.php.net/PDO::rowCount*）和PDO::exec()方法的文档（*http://www.php.net/exec*）。

10.9 转义引号

问题

你需要在查询中保证文本或者二进制数据的安全。

方案

使用占位符方法来组织所有的查询语句，这样prepare()和execute()就可以替你转义字符串了。技巧10.7中详细地讨论了使用占位符的各种方式。

如果你需要自己处理转义，那么可以使用PDO::quote()方法。事实上，只有很少的情况下才需要你自己来做这件事，比如说你需要对来自用户输入的SQL通配符进行转义，如例10-24所示。

例10-24：手工转义引号

```
<?php
$safe = $db->quote($_GET['searchTerm']);
$safe = strtr($safe,array('_' => '\_', '%' => '\%'));
$st = $db->query("SELECT * FROM zodiac WHERE planet LIKE $safe");
?>
```

讨论

PDO::quote()方法能够确保文本或者二进制数据中的引号被转义，但是你可能需要自

已转义SQL通配符%和_以保证使用LIKE操作符的SELECT语句能够返回正确的结果。如果$_GET['searchTerm']被设置为Melm%并且例10-24没有调用strtr()，则该查询会返回planet为 Melmac、Melmacko、Melmacedonia或其他以Melm开头的单词。

%是SQL中含义为“匹配任何数字和字符”（类似shell globbing中的*）的通配符，而_是SQL中含义为“匹配一个字符”（类似shell globbing中的?）的通配符，这两个通配符也都需要进行转义处理。

必须在PDO::quote()方法之后调用strtr()。此外，PDO::quote()会用反斜杠转义strtr()添加的反斜杠。首先，通过调用PDO::quote()，Melm_被转换成Melm_，这会被数据库解释为“字符串Melm后面跟一个下划线直接量字符”。通过调用PDO::quote()之后的strtr()，Melm_又被转换成Melm_，这会被数据库解释为“字符串Melm后面跟一个反斜杠直接量字符和一个下划线通配符”。这与我们直接转义SQL中的通配符并将结果值用作绑定参数的效果相同。

即使在magic_quotes_gpc或magic_quotes_runtime转为on时，占位符值的引号问题依然存在。类似地，如果在魔术引号有效的情况下调用PDO::quote()，那么无论如何值都会加上引号。为了保证最大的可移植性，就需要在使用带占位符的查询或调用PDO::quote()之前删除魔术引号提供的反斜线。

例10-25：检查魔术引号

```
<?php
// magic_quotes_sybase 的行为也是一个影响因素
if (get_magic_quotes_gpc() && (! ini_get('magic_quotes_sybase'))) {
    $fruit = stripslashes($_GET['fruit']);
} else {
    $fruit = $_GET['fruit'];
}
$st = $db->prepare('UPDATE orchard SET trees = trees - 1 WHERE fruit = ?');
$st->execute(array($fruit));
?>
```

如果你能够控制服务器，那么可以通过禁用魔术引用功能来简化自己的工作。但是，如果你试图编写的是一个需要在你所不能控制的环境中运行的、具有最大可移植性的代码，那么你就必须留意这个问题。

参见

PDO::quote()方法的文档（*http://www.php.net/PDO::quote*）和有关魔术引号的信息（*http://www.php.net/manual/en/ref.info.php#ini.magic-quotes-gpc*）。

10.10 记录调试信息和错误

问题

你想访问能帮你调试数据库问题的信息。例如，当某个查询失败时，你想知道数据库返回了哪些错误信息。

方案

在执行某次操作失败后使用PDO::errorCode()或PDOStatement::errorCode方法获得相应的错误代码。对应的errorInfo()方法则会返回有关该错误的更多信息。例10-26显示了如何处理试图访问一个不存在的数据库时产生的错误信息。

例10-26：输出错误信息

```
<?php
$st = $db->prepare('SELECT * FROM imaginary_table');
if (! $st) {
    $error = $db->errorInfo();
    print "Problem ({$error[2]})";
}
?>
```

讨论

errorCode()方法会返回一个5个字符的错误代码。PDO使用SQL 92 SQLSTATE规定的错误代码。在该标准中，00000的含义是“没有错误”，因此当调用errorCode()返回00000时，表示成功。

errorInfo()方法返回的是一个包含三个元素的数组。其中第一个元素包含5个字符的SQLSTATE代码（与errorCode()返回的代码相同）。第二个元素是具体的数据库后台专有的错误代码。第三个元素是具体的数据库后台专有的错误信息。

必须保证在调用方法出错的对象上调用errorCode()或errorInfo()方法。在例10-26中，prepare()方法是在PDO对象上被调用的，所以errorInfo()也被在PDO对象上调用。如果你想检查一下在PDOStatement对象上调用fetch()是否成功，可以在相应的PDOStatement对象上调用errorCode()或errorInfo()方法。

这个规则的一个特例发生在创建一个新的PDO对象时。如果创建失败，PDO会抛出一个异常。之所以如此，是因为除此之外不存在一个可以调用errorCode()或errorInfo()方法的对象。而异常中的信息则包含了连接为什么失败的具体原因。

如果想让PDO每次遇到错误时都抛出一个异常，可以在创建该PDO对象后在其上调用 setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION) 。这样，你就可以采取统一的方式来处理数据库错误，而不用再在代码中一遍遍地重复调用errorCode()或errorInfo()了。例10-27演示了如何执行包装在try/catch块中的一系列数据库操作。

例10-27：捕捉数据库异常

```
<?php
try {
    $db = new PDO('sqlite:/usr/local/zodiac.db');
    // 让所有数据库错误都抛出异常
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $st = $db->prepare('SELECT * FROM zodiac');
    $st->execute();
    while ($row = $st->fetch(PDO::FETCH_NUM)) {
        print implode(',',$row). "<br/>\n";
    }
} catch (Exception $e) {
    print "Database Problem: " . $e->getMessage();
}
?>
```

同样，将PDO错误以异常的方式来处理在事务内部也是很有用的。比如说，如果在事务启动时查询出现了问题，那么只需在处理异常时回滚该事务即可。

类似的异常错误处理模式还有“警告”错误模式。通过调用 setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING)，可以告诉PDO在遇到数据库错误时放出一个警告信息。要是你刚好喜欢使用常规的PHP错误处理方式而不是异常模式，那么这种错误模式正好适合你。另外，通过set_error_handler()建立一个自定义的错误处理器来处理E_WARNING级别的事件，你就可以通过这个错误处理器处理你的数据库错误了。

无论哪种错误模式，如果初始的PDO对象创建失败，PDO都会抛出异常。在使用PDO时，最好是使用set_exception_handler()来建立一个默认的异常处理器。如果没有定义异常处理器，那么当出现未能捕捉到的异常而且display_error值为on的情况下，就会导致完全的栈跟踪显示，而该栈跟踪中有可能包含一些敏感的信息，比如数据库连接证书信息。

参见

PDO::errorCode()方法的文档（*http://www.php.net/PDO::errorCode*），PDO::errorInfo()方法的文档（*http://www.php.net/PDO::errorInfo*），PDO::errorCode()方法的文档（*http://www.php.net/PDOStatement::errorCode*），PDOStatement::errorInfo()方法的文档（*http://www.php.net/PDOStatement::errorInfo*），set_exception_handler()

函数的文档（*http://www.php.net/set_exception_handler*）和set_error_handler()函数的文档（*http://www.php.net/set-error-handler*）。PDO可以识别的SQL 92 SQLSTATE所规定的错误代码列表（*http://cvs.php.net/viewcvs.cgi/php-src/ext/pdo/pdo_sqlstate.c?view=markup*），但是某些数据库后台也可能会发生该列表之外的错误。

10.11 创建唯一的标识符

问题

你想在把用户、文章或者其他对象添加到数据库中的时候，为其指定一个唯一的标识符。

方案

使用PHP的uniqid()函数生成一个标识符。为了限制标识符中的字符范围，将标识符传递到md5()函数中，返回一个只包含由数字和字母a～f组成的字符串。例10-28就使用了这两种技术来创建标识符。

例10-28：创建唯一标识符

```
<?php
$st = $db->prepare('INSERT INTO users (id, name) VALUES (?,?)');
$st->execute(array(uniqid(), 'Jacob'));
$st->execute(array(md5(uniqid()), 'Ruby'));
?>
```

也可以使用数据库专有的方法得到数据库生成的ID值。例如，SQLite 3和MySQL都支持AUTOINCREMENT列，可以在插入记录行时自动地在该列中生成递增的整数值。

讨论

uniqid()使用当前的时间（以毫秒计）和随机数来生成一个极难猜测的字符串。md5()则将传递给它的字符串转换成相应的散列，它不会给标识符中添加任何随机性，只是限制标识符中出现的字符范围。由于经过md5()计算的结果中不会包含任何标识符号，所以不用担心转义的问题。而且，字母表中的前6个字母也不可能拼写出任何不规范的单词来（至少在英语中是这样）。

如果你愿意把生成唯一标识符的事情交给数据库来做，那么就需要在创建相应的数据库表时使用适当的语法。例10-29显示了如何在SQLite中创建一个表，其中有一列会在每次插入新记录行时自动得到一个递增的整数ID值。

例10-29：在SQLite中创建一个自动递增的列

```
<?php
// 类型声明INTEGER PRIMARY KEY AUTOINCREMENT 是在告诉SQLite 分配递增的ID值
$db->exec(<<<_SQL_
  CREATE TABLE users (
    id  INTEGER PRIMARY KEY AUTOINCREMENT,
    name VARCHAR(255)
  )
_SQL_
);

// 不需要为“id”插入值 —— SQLite会自动为它赋值
$st = $db->prepare('INSERT INTO users (name) VALUES (?)');

// 插入的行都会给“id”列赋值
foreach (array('Jacob','Ruby') as $name) {
    $st->execute(array($name));
}
?>
```

例10-30显示了如何在MySQL中做到这一点。

例10-30：在MySQL中创建一个自动递增的列

```
<?php
// AUTO_INCREMENT告诉MySQL分配递增的ID值
// 而该列必须是主键（PRIMARY KEY）
$db->exec(<<<_SQL_
  CREATE TABLE users (
    id  INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(255),
    PRIMARY KEY(id)
  )
_SQL_
);

// 不需要为“id”插入值 —— MySQL会自动为它赋值
$st = $db->prepare('INSERT INTO users (name) VALUES (?)');

//  插入的行都会给“id”列赋值
foreach (array('Jacob','Ruby') as $name) {
    $st->execute(array($name));
}
?>
```

如果是通过数据库自动创建的ID值，那么就可以用PDO::lastInsertId()方法来取得相应的ID值。也就是说，在相应的PDO对象上调用lastInsertId()可以取得最后插入的那一行记录所自动生成的ID值。有一些数据库后台也支持给lastInsertId()传递一个顺序名作为参数，进而返回该顺序中的最后一个值。

参见

uniqid()函数的文档（*http://www.php.net/uniqid*），md5()函数的文档（*http://www.php.net/md5*），PDO::lastInsertId()方法的文档（*http://www.php.net/PDO::lastInsertId*），SQLite和AUTOINCREMENT的有关信息（*http://www.sqlite.org/faq.html#q1*）和MySQL及AUTO_INCREMENT的相关资料（*http://dev.mysql.com/doc/refman/5.0/en/example-auto-increment.html*）。

10.12 以程序化的方式建立查询

问题

你想根据包含字段名的数组来建立一个INSERT或UPDATE查询。例如，你想把一个新用户插入到数据库中。对该用户的信息（比如username、emailaddress、postaladdress、birthdate等等）不是采取硬编码每个字段的方式，而是将这些字段名放到一个数组中，并根据该数组来构建相应的查询。这样可以使维护更容易，特别是当在需要有条件地INSERT或UPDATE同一组字段的时候。

方案

如果要构建一个UPDATE查询，需要先建立一个包含字段/值对的数组，然后通过implode()函数使该数组中的每一个元素结合在一起构成更新语句，具体内容如例10-31所示。

例10-31：构建一个UPDATE查询

```
<?php
// 一个字段名列表
$fields = array('symbol','planet','element');

$update_fields = array();
$update_values = array();
foreach ($fields as $field) {
    $update_fields[] = "$field = ?";
    // 假定数据来自表单
    $update_values[] = $_POST[$field];
}

$st = $db->prepare("UPDATE zodiac SET " .
                   implode(',', $update_fields) .
                   'WHERE sign = ?');

// 把'sign'的值添加到值数组中
$update_values[] = $_GET['sign'];
```

```
// 执行查询
$st->execute($update_values);
?>
```


对于INSERT查询，要做同样的查询，尽管SQL语法略有差别，其余逻辑过程都是一样的，如例10-32所示。

例10-32：构建一个INSERT查询

```
<?php
// 一个字段名列表
$fields = array('symbol','planet','element');
$placeholders = array();
$values = array();
foreach ($fields as $field) {
    // 每个字段一个占位符
    $placeholders[] = '?';
    // 假定数据来自表单
    $values[] = $_POST[$field];
}

$st = $db->prepare('INSERT INTO zodiac (' .
                   implode(',',$fields) .
                   ') VALUES (' .
                   implode(',', $placeholders) .
                   ')');
// 执行查询
$st->execute($values);
?>
```

讨论

使用占位符可以使这种技术显得轻而易举。而且由于它们会处理（转义）提供的数据，所以的确很容易就可以把用户提交的数据填入到程序生成的查询语句当中。

如果你使用顺序生成的整数作为主键，那么就可以将这两种查询构造技术组合到一个函数中。这个函数会判断一条记录是否存在，然后生成正确的查询，包括新的ID值。例10-33中就提供了这样一个函数pc_build_query()。

例10-33：pc_build_query()函数

```
<?php
function pc_build_query($db,$key_field,$fields,$table) {
    $values = array();
    if (! empty($_POST[$key_field])) {
        $update_fields = array();
        foreach ($fields as $field) {
            $update_fields[] = "$field = ?";
            // 假定数据来自表单
            $values[] = $_POST[$field];
        }
```

```
        // 把键的字段的值添加到$Values数组
        $values[] = $_POST[$key_field];
        $st = $db->prepare("UPDATE $table SET " .
                  implode(',', $update_fields) .
                  "WHERE $key_field = ?");
    } else {
        // 默认使用唯一的ID值
        // 如果数据库设置为自动生成该值，则使用NULL来代替
        $values[] = md5(uniqid());
        $placeholders = array('?');
        foreach ($fields as $field) {
            // 每个字段一个占位符
            $placeholders[] = '?';
            // 假定数据来自表单
            $values[] = $_POST[$field];
        }
        $st = $db->prepare("INSERT INTO $table ($key_field," .
                           implode(',',$fields) . ') VALUES ('.
                           implode(',',$placeholders) .')');
    }
    $st->execute($values);
    return $st;
}
?>
```

通过使用这个函数，可以实现以简单的页面来编辑zodiac表中的所有信息，如例10-34所示。

例10-34：一个简单的添加/编辑记录的页面

```
<?php
$db = new PDO('sqlite:/usr/local/data/zodiac.db');
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$fields = array('sign','symbol','planet','element',
                'start_month','start_day','end_month','end_day');

$cmd = isset($_REQUEST['cmd']) ? $_REQUEST['cmd'] : 'show';

switch ($cmd) {
 case 'edit':
 try {
    $st = $db->prepare('SELECT ' . implode(',',$fields) .
                       ' FROM zodiac WHERE id = ?');
    $st->execute(array($_REQUEST['id']));
    $row = $st->fetch(PDO::FETCH_ASSOC);
 } catch (Exception $e) {
     $row = array();
 }
 case 'add':
     print '<form method="post" action="' .
           htmlentities($_SERVER['PHP_SELF']) . '">';
     print '<input type="hidden" name="cmd" value="save">';
     print '<table>';
```

```
        if ('edit' == $_REQUEST['cmd']) {
            printf('<input type="hidden" name="id" value="%d">',
                   $_REQUEST['id']);
        }
        foreach ($fields as $field) {
            if ('edit' == $_REQUEST['cmd']) {
                $value = htmlentities($row[$field]);
            } else {
                $value = '';
            }
            printf('<tr><td>%s: </td><td><input type="text" name="%s" value="%s">',
                   $field,$field,$value);
            printf('</td></tr>');
        }
        print '<tr><td></td><td><input type="submit" value="Save"></td></tr>';
        print '</table></form>';
        break;
    case 'save':
        try {
          $st = pc_build_query($db,'id',$fields,'zodiac');
          print 'Added info.';
        } catch (Exception $e) {
          print "Couldn't add info: " . htmlentities($e->getMessage());
        }
        print '<hr>';
    case 'show':
    default:
        $self = htmlentities($_SERVER['PHP_SELF']);
        print '<ul>';
        foreach ($db->query('SELECT id,sign FROM zodiac') as $row) {
            printf('<li> <a href="%s?cmd=edit&id=%s">%s</a>',
                   $self,$row['id'],$row['sign']);
        }
        print '<hr><li> <a href="'.$self.'?cmd=add">Add New</a>';
        print '</ul>';
        break;
}?>
```

其中的switch语句根据$_REQUEST['cmd']的值控制着程序所要执行的动作。如果$_REQUEST['cmd']的值是add，则程序会显示一个表单，表单中包含着与$fields数组中的每个字段对应的文本输入框，如图10-1所示。如果$_REQUEST['cmd']的值是edit，那么程序会从数据库中加载与提供的$id对应的记录行数据，并将各个字段的值显示为文本输入框中的默认值。如果$_REQUEST['cmd']的值是save，程序会通过调用pc_build_query()函数生成相应的查询，向数据库中INSERT或UPDATE数据。在保存数据（或者如果没有指定$_REQUEST['cmd']）之后，程序则会显示所有黄道十二宫列表，如图10-2所示。

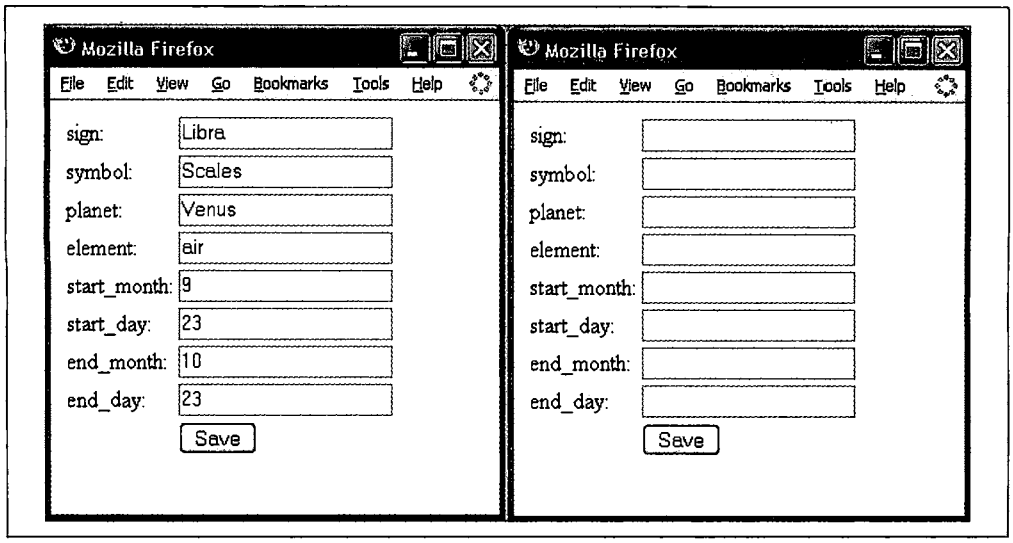

图10-1：添加和编辑记录

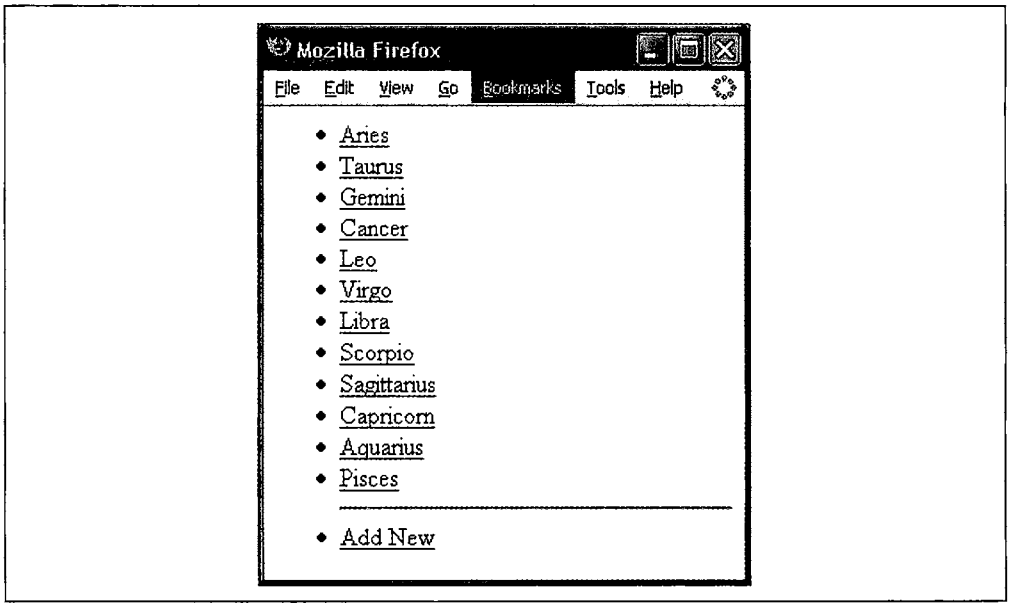

图10-2：记录列表

至于pc_build_query()函数是构建NSERT还是构建UPDATE语句，取决于请求变量$_REQUEST['id']是否存在（因为id传入$key_field中）。如果$_REQUEST['id']非空，该函数会构建一个UPDATE查询以更新带有那个ID值的记录行。如果$_REQUEST['id']是空的（或者根本没有设置），则该函数会生成一个新ID值，并在INSERT查询中使用这个新ID

值向表中插入一条新记录。如果想让pc_build_query()考虑数据库中的AUTOINCREMENT设置，可以将$value变量的默认值改为NULL，而不是md5(uniqid())。

参见

10.7中有关占位符、prepare()和execute()方法的用法；PDO::prepare()方法的文档（*http://www.php.net/PDO::prepare*）和PDOStatement::execute()方法的文档（*http://www.php.net/PDOStatement::execute*）。

10.13 为连续的记录生成分页链接

问题

你想在一个页面中每次显示一个大数据集，并提供遍览该数据集的链接。

方案

使用数据库专有的语法来取得与你的查询匹配的一部分记录。例10-35显示的是在SQLite中的特殊用法。

例10-35：通过SQLite实现分页

```
<?php
// 从第3行开始，选择5行
foreach ($db->query('SELECT * FROM zodiac ' .
                    'ORDER BY sign LIMIT 5 ' .
                    'OFFSET 3') as $row) {
    // 对每一条记录进行处理
}
?>
```

本技巧中的pc_indexed_links()和pc_print_link()函数用于辅助输出分页信息。例10-36中演示了如何进行分页处理。

例10-36：显示分页结果

```
<?php
$offset = isset($_GET['offset']) ? intval($_GET['offset']) : 1;
if (! $offset) { $offset = 1; }
$per_page = 5;
$total = $db->query('SELECT COUNT(*) FROM zodiac')->fetchColumn(0);

$limitedSQL = 'SELECT * FROM zodiac ORDER BY id ' .
              "LIMIT $per_page OFFSET " . ($offset-1);
$lastRowNumber = $offset - 1;
```

```
foreach ($db->query($limitedSQL) as $row) {
    $lastRowNumber++;
    print "{$row['sign']}, {$row['symbol']} ({$row['id']}) <br/>\n";
}

pc_indexed_links($total,$offset,$per_page);
print "<br/>";
print "(Displaying $offset - $lastRowNumber of $total)";
?>
```

讨论

其中pc_print_link()如例10-37所示，pc_indexed_links()如例10-38所示。

例10-37：pc_print_link()函数

```
<?php
function pc_print_link($inactive,$text,$offset='') {
    if ($inactive) {
        print "<span class='inactive'>$text</span>";
    } else {
        print "<span class='active'>".
              "<a href='" . htmlentities($_SERVER['PHP_SELF']) .
              "?offset=$offset'>$text</a></span>";
    }
}
?>
```

例10-38：pc_indexed_links()函数

```
<?php
function pc_indexed_links($total,$offset,$per_page) {
    $separator = ' | ';

    // 输出 "<<Prev" 链接
    pc_print_link($offset == 1, '<< Prev', $offset - $per_page);

    // 输出除最后一个之外的全部分组
    for ($start = 1, $end = $per_page;
         $end < $total;
         $start += $per_page, $end += $per_page) {
            print $separator;
            pc_print_link($offset == $start, "$start-$end", $start);
    }

    /* 输出最后一组——
     * 这里，$start指向最后一组中开始处的元素
     */

    /* 如果在最后一页包含多于1个元素，那么text中应只包含一个表示范围的字符串
     * 例如，11个元素，每页5个的情况下，最后一组只应该是“11”而非“11-11”
     */
    $end = ($total > $start) ? "-$total" : '';
```

```
    print $separator;
    pc_print_link($offset == $start, "$start$end", $start);

    // 输出 "Next>>" 链接
    print $separator;
    pc_print_link($offset == $start, 'Next >>',$offset + $per_page);
}
?>
```

通过使用这两个函数，可以抽取所用数据集中正确的子集并输出。而调用pc_indexed_links()则可以显示带索引的链接。

在连接到数据库之后，需要确保的是$offset中要有一个适当的值。$offset表示要显示的结果记录集中的起始记录。如果想从记录集的起始位置开始显示，$offset应该设置为1。变量$per_page用于定义每页显示几条记录，而$total则是整个结果集中的记录总数。对于本例而言，要显示所有的黄道十二宫记录，所以$total就设置为整个表中的总行数。

而用于抽取信息的SQL查询的正确顺序应该是：

```
<?php
$limitedSQL = 'SELECT * FROM zodiac ORDER BY id ' .
"LIMIT $per_page OFFSET " . ($offset-1);
?>
```

其中，LIMIT和OFFSET关键字用于告诉SQLite要返回的只是所有匹配记录行的一个子集。

相应的记录行是通过$db->query($limitedSQL)取得的，之后显示了每一行记录中的信息。在这些信息的后面，pc_indexed_links()提供了导航链接。$offset没有设置（或设置为1）时的输出结果如图10-3所示。

在图10-3中“6-10”、“11-12”和“Next>>”是带有调整后的$offset参数的、指向同一个页面的链接，而“<< Prev”和“1-5”则显示为灰色，因为它们所指向的都是当前显示的页面。

参见

有关Solar框架中分页问题的讨论（*http://paul-m-jones.com/blog/?p=185*）和不同数据库中分页语法的信息（*http://troels.arvin.dk/db/rdbms/#select-limit-offset*）。

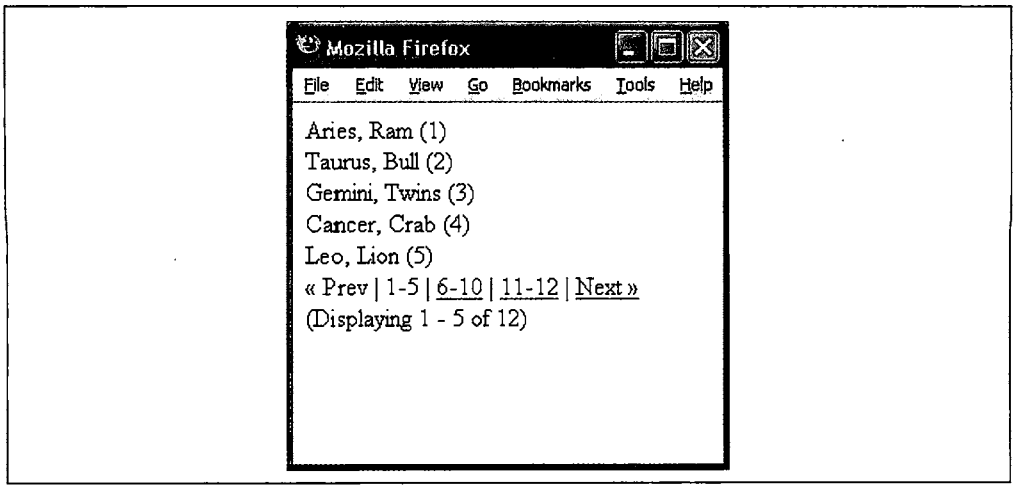

图10-3：通过pc_indexed_links()函数实现的分页结果

10.14 缓存查询和结果

问题

你不想在结果未改变的情况下重新运行存在潜在开销的数据库查询。

方案

使用PERA的Cache_Lite包。这个包使得缓存任何数据都变得很简单。这里，我们缓存的是一个SELECT查询的结果，并把查询的文本作为缓存键。例10-39显示了如何使用Cache_Lite来缓存查询的结果。

例10-39：缓存查询的结果

```
<?php
require_once 'Cache/Lite.php';

$opts = array(
   // 把缓存的数据放在何处
   'cacheDir' => 'c:/tmp',
   // 我们在缓存中保存数组
  'automaticSerialization' => true,
  // 缓存中的信息保存多长时间
  'lifeTime' => 600 /* 10分钟 */);

// 创建缓存
$cache = new Cache_Lite($opts);
```

```
// 连接到数据库
$db = new PDO('sqlite:c:/data/zodiac.db');

// 定义查询及其参数
$sql = 'SELECT * FROM zodiac WHERE planet = ?';
$params = array($_GET['planet']);

// 取得唯一的缓存键
$key = cache_key($sql, $params);

// 尝试从缓存中取得结果
$results = $cache->get($key);

if ($results === false) {
    // 没找到结果，那么执行查询并将结果放入缓存中
    $st = $db->prepare($sql);
    $st->execute($params);
    $results = $st->fetchAll();
    $cache->save($results);
}

// 无论是否取自缓存，$results中都保存我们的数据
foreach ($results as $result) {
    print "$result[id]: $result[planet], $result[sign] <br/>\n";
}

function cache_key($sql, $params) {
    return md5($sql .
               implode('|',array_keys($params)) .
               implode('|',$params));
}
?>
```

讨论

Cache_Lite是缓存任意信息的一种普通的、轻量级的机制。它把缓存的信息保存在文件中。Cache_Lite构造器接受一个用于控制其行为的选项数组作为参数。例10-39的核心在于自动序列化（automaticSerialization）处理，这使得在缓存和cacheDir（其中定义的是缓存文件的存储位置）中存储数组更加容易。最后，要确保cacheDir以一个“/”（斜杠）结尾。

缓存中保存的只是键与值的映射。用于唯一标识要缓存数据的缓存键 —— 在本例中，是SQL查询及其绑定的参数 —— 是由我们来提供的。cache_key函数负责计算出一个合适的键值。此后，例10-39只是检查了一下缓存中是否有保存的结果。如果没有，则会执行对数据库的查询操作，并将返回的结果保存到缓存中以备后用。

注意，我们不能把PDO或者PDOStatement对象放到缓存文件中 —— 而只能是取得结果，把结果放到缓存中。

在默认情况下，缓存内容保存的时间为一小时。也可以在创建新的Cache_Lite对象时通过为lifeTime选项指定不同的时间值（以秒表示）来调整缓存时间。如果不想让保存的数据自动过期，可以传递NULL。

缓存中保存的数据不会因为我们通过INSERT、UPDATE或DELETE查询更新了数据库而改变。如果缓存的SELECT语句所引用的数据在数据库中已经不在了，则需要明确地调用Cache_Lite::clean()方法来删除缓存中的数据。也可以通过给Cache_Lite::remove()方法传递一个键作为参数，而删除缓存中个别的元素。

例10-39中的cache_key()函数是大小写敏感的。这意味着，如果缓存中保存的是SELECT * FROM zodiac，而你运行的查询是SELECT * from zodiac，那么就会因为没有发现缓存中保存的结果而重新执行该查询。所以，如果我们在构造SQL查询结果时能够保持一致的大小写、空格及字段顺序，那么就会提高缓存应用的效率。

参见

Cache_Lite包的文档（*http://pear.php.net/manual/en/package.caching.cache-lite.php*）。

10.15 在程序中任何地方都能访问数据库连接

问题

你的程序中包含了许多函数和类，而你想单独保存一个数据库连接，以便在程序中的任何地方都能容易使用该连接。

方案

使用一个静态类方法，该方法在连接不存在时会创建并返回连接（参见例10-40）。

例10-40：用一个静态类方法创建一个数据库连接

```
<?php
class DBCxn {
    // 要连接到哪个DSN?
    public static $dsn = 'sqlite:c:/data/zodiac.db';
    public static $user = null;
    public static $pass = null;
    public static $driverOpts = null;

    // 保存连接的内部变量
    private static $db;
    // 不允许克隆或技巧化
```

```
    final private function _ _construct() { }
    final private function _ _clone() { }

    public static function get() {
        // 如果不存在连接，则进行连接
        if (is_null(self::$db)) {
            self::$db = new PDO(self::$dsn, self::$user, self::$pass,
                                self::$driverOpts);
        }
        // 返回连接
        return self::$db;
    }
}
?>
```

讨论

例10-40中定义的DBCxn::get()方法完成了两件事：你可以在程序中的任何地方访问它，而不用担心变量作用域的问题，而且它可以防止你的程序创建第二个连接。

如果想改变DBCxn::get()所提供的连接类型，只需修改这个类的$dsn、$user、$pass和$driverOpts属性即可。如果在执行同一个脚本的过程中，需要管理多个不同的数据库连接，可以把$dsn和$db改为数组并让get()接受一个标明要使用连接的参数。例10-41中显示了通过DBCxn类来提供对三个不同数据库进行访问的用法。

例10-41：处理对多个数据库的连接

```
<?php
class DBCxn {
    // 要连接到哪个DSN?
    public static $dsn =
       array('zodiac' => 'sqlite:c:/data/zodiac.db',
             'users' => array('mysql:host=db.example.com','monty','7f2iuh'),
             'stats' => array('oci:statistics', 'statsuser','statspass'));


    // 保存连接的内部变量
    private static $db = array();
    // 不允许克隆或技巧化
    final private function _ _construct() { }
    final private function _ _clone() { }

    public static function get($key) {
        if (! isset(self::$dsn[$key])) {
            throw new Exception("Unknown DSN: $key");
        }
        // 如果不存在连接，则进行连接
        if (! isset(self::$db[$key])) {
            if (is_array(self::$dsn[$key])) {
                // 下面两行代码只在PHP 5.1.3及以上版本中有效
                $c = new ReflectionClass('PDO');
```

```
                self::$db[$key] = $c->newInstanceArgs(self::$dsn[$key]);
            } else {
                self::$db[$key] = new PDO(self::$dsn[$key]);
            }
        }
        // 返回连接
        return self::$db[$key];
    }
}
?>
```

在例10-41中，必须给DBCxn::get()方法传递一个在$dsn中标明要使用项目的键。同时，get()方法内部的代码也更复杂一些，因为此时必须要处理与PDO构造器相关的多个可变参数。某些数据库，例如SQLite，只需要一个参数。而有的数据库则需要两个，甚至四个参数。例10-41使用了ReflectionClass::newInstanceArgs()方法（该方法在PHP 5.1.3中引入），以便用简洁的方式调用一个构造器并通过数组来传递参数。如果你使用的是PHP早期的版本，那么需要将这里的代码new ReflectionClass('PDO')和newInstanceArgs()替换成例10-42中的代码。

例10-42：在PHP的旧版本中调用PDO构造器

```
<?php
$args = self::$dsn[$key];
$argCount = count($args);
if ($argCount == 1) {
    self::$db[$key] = new PDO($args[0]);
} else if ($argCount == 2) {
    self::$db[$key] = new PDO($args[0],$args[1]);
} else if ($argCount == 3) {
    self::$db[$key] = new PDO($args[0],$args[1],$args[2]);
} else if ($argCount == 4) {
    self::$db[$key] = new PDO($args[0],$args[1],$args[2],$args[3]);
}
?>
```

例10-42检查了提供给PDO构造器的每一个可能的参数数目，进而调用相应的构造器。

参见

PDO::_ _construct()方法的文档（*http://www.php.net/PDO::__construct*）和ReflectionClass::newInstanceArgs()方法的文档（*http://www.php.net/language.oop5.reflection*）。

10.16 编程：存储链式（Threaded）留言板

要存储链式留言（messages）必需要特别仔细，才能保证以正确的顺序来显示线索。而

找到每条留言的子留言并构建相应的留言关系树状结构，则可以很容易地通过一个递归的Web查询实现。用户在阅读一个留言列表时，通常会打开某条留言并地逐个阅读其中包含的子留言，最后才发表自己的留言。通过在向数据库中保存新留言时做一些额外的处理，就可以使取得要显示留言列表的查询变得相对简单而且更富效率。

用于保存留言的表结构如下：

```
CREATE TABLE pc_message (
  id INT UNSIGNED NOT NULL,
  posted_on DATETIME NOT NULL,
  author CHAR(255),
  subject CHAR(255),
  body MEDIUMTEXT,
  thread_id INT UNSIGNED NOT NULL,
  parent_id INT UNSIGNED NOT NULL,
  level INT UNSIGNED NOT NULL,
  thread_pos INT UNSIGNED NOT NULL,
  PRIMARY KEY(id)
);
```

表的主键id是一个用于标识特定留言的唯一整数。发表留言的时间和日期保存在posted_on中，而author、subject和body顾名思义分别是留言的作者、主题和内容。其余的四个字段则用于跟踪留言之间的链式关系。整数类型的thread_id用于标识每一个线索。某个线索中的所有留言都具有相同的thread_id。如果一条留言回复了另一条留言，则其parent_id就是它所回复的留言的id。level中保存的是一条线索有多少级回复留言。一个线索中的第一条留言的级别（level）是0，而对该级别留言的回复级别则是1，相应地，1级留言的回复级别是2。而一条线索中的多条留言可能拥有同样的级别和相同的parent_id。例如，如果某人发表了一条有关“BeOS胜过CP/M之优点”的留言线索，那么来自CP/M拥护者阵营对该条留言的愤怒回复的级别都是1，而其parent_id也都等于所回复的原始留言的id。

最后一个字段thread_pos，控制着所有留言的显示，即一条线索中的所有留言都按照相应的thread_pos的值来进行排序。

以下是计算thread_pos值的规则：

- 线索中的第一条留言包含thread_pos=0。
- 对于新留言N，如果在该线索中没有留言与N拥有相同的父留言，则N的thread_pos值就是其父留言的thread_pos值加1。
- 对于新留言N，如果在该线索中有其他留言与N拥有相同的父留言，则N的thread_pos值就比其他留言中最大的thread_pos值加1。

- 当新留言N的thread_pos值确定之后，同一线索中的所有thread_pos值大于或等于N的thread_pos值的留言，其thread_pos值增1（给N留出空间）。

例10-43中的留言板程序（*message.php*）可以保存留言，并恰当地计算thread_pos值。图10-4显示了该程序输出的样例。

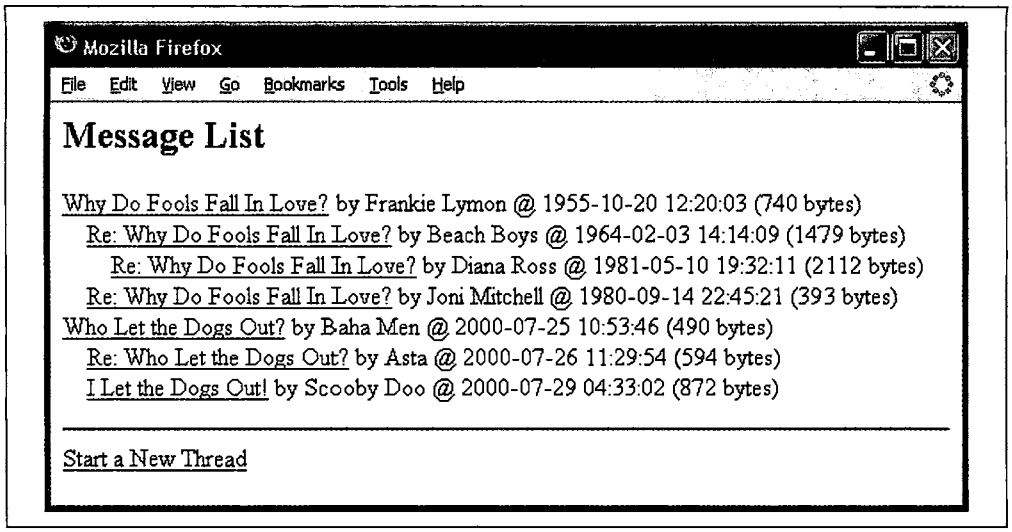

图10-4：链式留言板

例10-43：message.php

```
<?php

$board = new MessageBoard();
$board->go();

class MessageBoard {
    protected $db;
    protected $form_errors = array();
    protected $inTransaction = false;

    public function __construct() {
        set_exception_handler(array($this,'logAndDie'));
        $this->db = new PDO('sqlite:/usr/local/data/message.db');
        $this->db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
    }

    public function go() {
        //  $_REQUEST['cmd'] 值告诉我们要做什么
        $cmd = isset($_REQUEST['cmd']) ? $_REQUEST['cmd'] : 'show';
        switch ($cmd) {
            case 'read':          // 读取个别留言
              $this->read();
              break;
```

```
        case 'post':          // 显示发表留言的表单
          $this->post();
          break;
        case 'save':          // 保存发表的留言
          if ($this->valid()) { // 如果留言有效，
              $this->save();    // 则将其保存
              $this->show();    // 并显示留言列表
          } else {
              $this->post();    // 否则，再次显示发表过的表单
          }
          break;
        case 'show':          // 默认情况下显示留言列表
        default:
          $this->show();
          break;
    }
}

// save() 把留言保存到数据库中
protected function save() {

    $parent_id = isset($_REQUEST['parent_id']) ?
                 intval($_REQUEST['parent_id']) : 0;

    // 确保我们在使用pc_message时，它不会改变
    $this->db->beginTransaction();
    $this->inTransaction = true;

    // 这个留言是回复吗?
    if ($parent_id) {
        // 取得父留言的线索、级别和thread_pos值
        $st = $this->db->prepare("SELECT thread_id,level,thread_pos
                                  FROM pc_message WHERE id = ?");
        $st->execute(array($parent_id));
        $parent = $st->fetch();

        // 回复的级别是其父留言级别加1
        $level = $parent['level'] + 1;

        /* 该线索中具有相同父留言的留言里面，
        最大的thread_pos值是多少? */
        $st = $this->db->prepare('SELECT MAX(thread_pos) FROM pc_message
                WHERE thread_id = ? AND parent_id = ?');
        $st->execute(array($parent['thread_id'], $parent_id));
        $thread_pos = $st->fetchColumn(0);

        // 存在对这个父留言的回复吗?
        if ($thread_pos) {
            // thread_pos值加1
            $thread_pos++;
        } else {
            // 这是第一条回复，所以排在父留言之后
            $thread_pos = $parent['thread_pos'] + 1;
        }
```

```
            /* 给线索中在此条留言之后的所有留言的thread_pos
            都加上1*/
            $st = $this->db->prepare('UPDATE pc_message SET thread_pos = thread_pos + 1
                        WHERE thread_id = ? AND thread_pos >= ?');
            $st->execute(array($parent['thread_id'], $thread_pos));

            // 新留言中应该保存父留言的thread_id
            $thread_id = $parent['thread_id'];
        } else {
            // 该留言不是回复，所以建立一条新线索
            $thread_id = $this->db->query('SELECT MAX(thread_id) + 1 FROM pc_message')
                          ->fetchColumn(0);
            $level = 0;
            $thread_pos = 0;
        }

        /* 把留言插入到数据库中。使用prepare()和execute()
        保证所有字段都能被适当地引用转义 */
        $st = $this->db->prepare("INSERT INTO pc_message (id,thread_id,parent_id,
                    thread_pos,posted_on,level,author,subject,body)
                    VALUES (?,?,?,?,?,?,?,?,?)");

        $st->execute(array(null,$thread_id,$parent_id,$thread_pos,
                           date('c'),$level,$_REQUEST['author'],
                           $_REQUEST['subject'],$_REQUEST['body']));

        // 提交所有操作
        $this->db->commit();
        $this->inTransaction = false;
    }

    // show() 用于显示所有留言的列表
    protected function show() {
        print '<h2>Message List</h2><p>';

        /* 根据线索（thread_id）及各自在线索中的位置（thread_pos）
        对留言进行排序 */
        $st = $this->db->query("SELECT id,author,subject,LENGTH(body) AS body_length,
                    posted_on,level FROM pc_message
                    ORDER BY thread_id,thread_pos");
        while ($row = $st->fetch()) {
            // 对于级别大于0的留言缩进显示
            print str_repeat(' ',4 * $row['level']);
            // 输出可以读取该留言信息的链接
            print "<a href='" . htmlentities($_SERVER['PHP_SELF']) .
            "?cmd=read&amp;id={$row['id']}'>" .
            htmlentities($row['subject']) . '</a> by ' .
            htmlentities($row['author']) . ' @ ' .
            htmlentities($row['posted_on']) .
            " ({$row['body_length']} bytes) <br/>";
        }

        // 提供一种发表非回复留言的途径
        print "<hr/><a href='" .
              htmlentities($_SERVER['PHP_SELF']) .
```

```
            "?cmd=post'>Start a New Thread</a>";
    }

    // read() 用于显示个别的留言
    public function read() {

        /* 确保我们传递的留言id是一个整数，并且
        的确代表一条留言 */
        if (! isset($_REQUEST['id'])) {
            throw new Exception('No message ID supplied');
        }
        $id = intval($_REQUEST['id']);
        $st = $this->db->prepare("SELECT author,subject,body,posted_on
                                  FROM pc_message WHERE id = ?");
        $st->execute(array($id));
        $msg = $st->fetch();
        if (! $msg) {
            throw new Exception('Bad message ID');
        }

        /* 不显示用户键入的HTML代码，但是显示在HTML换行时
        显示换行符 */
        $body = nl2br(htmlentities($msg['body']));

        // 显示留言、回复链接及返回留言列表的链接
        $self = htmlentities($_SERVER['PHP_SELF']);
        $subject = htmlentities($msg['subject']);
        $author = htmlentities($msg['author']);
        print<<<_HTML_
<h2>$subject</h2>
<h3>by $author</h3>
<p>$body</p>
<hr/>
<a href="$self?cmd=post&parent_id=$id">Reply</a>
<br/>
<a href="$self?cmd=list">List Messages</a>
_HTML_;
    }

    // post() 显示发表留言的表单
    public function post() {
        $safe =  array();
        foreach (array('author','subject','body') as $field) {
            // 对默认字段值中的字符进行转义
            if (isset($_POST[$field])) {
                $safe[$field] = htmlentities($_POST[$field]);
            } else {
                $safe[$field] = '';
            }
            // 用红色显示错误信息
            if (isset($this->form_errors[$field])) {
                $this->form_errors[$field] = '<span style="color: red">' .
                   $this->form_errors[$field] . '</span><br/>';
            } else {
                $this->form_errors[$field] = '';
```

```
            }
        }

        // 该留言是回复吗
        if (isset($_REQUEST['parent_id']) &&
        $parent_id = intval($_REQUEST['parent_id'])) {

        // 在表单提交时一起发送parent_id
        $parent_field =
            sprintf('<input type="hidden" name="parent_id" value="%d" />',
                    $parent_id);

        // 如果没有传递进来主题，则使用父留言的主题
        if (! strlen($safe['subject'])) {
             $st = $this->db->prepare('SELECT subject FROM pc_message WHERE id= ?');
             $st->execute(array($parent_id));
             $parent_subject = $st->fetchColumn(0);

            /* 如果父主题存在且尚未添加'Re:'前缀，
               则添加 'Re: '  */
            $safe['subject'] = htmlentities($parent_subject);
            if ($parent_subject && (! preg_match('/^re:/i',$parent_subject))) {
                $safe['subject'] = "Re: {$safe['subject']}";
            }
          }
        } else {
            $parent_field = '';
        }


    // 显示带有错误提示和默认值的发表表单
    $self = htmlentities($_SERVER['PHP_SELF']);
    print<<<_HTML_
<form method="post" action="$self">
<table>
<tr>
 <td>Your Name:</td>
 <td>{$this->form_errors['author']}
     <input type="text" name="author" value="{$safe['author']}" />
</td>
<tr>
 <td>Subject:</td>
 <td>{$this->form_errors['subject']}
     <input type="text" name="subject" value="{$safe['subject']}" />
</td>
<tr>
 <td>Message:</td>
 <td>{$this->form_errors['body']}
     <textarea rows="4" cols="30" wrap="physical"
               name="body">{$safe['body']}</textarea>
</td>
<tr><td colspan="2"><input type="submit" value="Post Message" /></td></tr>
</table>
$parent_field
<input type="hidden" name="cmd" value="save" />
```

```
</form>
_HTML_;
    }

    // validate() 保证每个字段中都填写了内容
    public function valid() {
        $this->form_errors = array();
        if (! (isset($_POST['author']) && strlen(trim($_POST['author'])))) {
            $this->form_errors['author'] = 'Please enter your name.';
        }
        if (! (isset($_POST['subject']) && strlen(trim($_POST['subject'])))) {
            $this->form_errors['subject'] = 'Please enter a message subject.';
        }
        if (! (isset($_POST['body']) && strlen(trim($_POST['body'])))) {
            $this->form_errors['body'] = 'Please enter a message body.';
        }

        return (count($this->form_errors) == 0);
    }

    public function logAndDie(Exception $e) {
        print 'ERROR: ' . htmlentities($e->getMessage());
        if ($this->db && $this->db->inTransaction) {
            $this->db->rollback();
        }
        exit();
    }
}
?>
```

要严格处理并发访问，save()需要从开始计算新留言的thread_pos值到实际把新留言插入到数据库中的这段时间内保持对msg表的独占访问。为了实现这一点，我们使用的是PDO的beginTransaction()和commit()方法。注意，logAndDie()是一个异常处理器，它会在适当的时候（比如事务中出错时）回滚事务。尽管PDO在事务启动后遇到脚本停止执行时也会调用rollback()，但在logAndDie()中明确包含对该方法的调用，可以使阅读这一代码的人更清楚其作用。

当显示留言时需要限制从数据库中取出的数据时，可以使用level字段。如果讨论线索的回复层次非常多，该字段可以帮我们防止页面过分膨胀。例10-44显示了如何只显示每条线索的第一条留言，以及针对第一条留言的所有回复。

例10-44：限制线索深度

```
<?php
$st = $this->db->query(
    "SELECT * FROM pc_message WHERE level <= 1 ORDER BY thread_id,thread_pos");
while ($row = $st->fetch()) {
    // 显示每一条留言
}
?>
```

如果你乐于在自己的网站中建立一个论坛，可能会考虑使用某个现成的PHP留言板包。FUDForum（*http://fudforum.org/forum/*）就是一个流行的留言板系统，而类似的程序还有很多，可以参考*http://www.zend.com/apps.php?CID=261*中列出的一些程序。

第11章

Session和数据保持

11.0 概述

当Web应用程序成熟后，对于保持会话状态的需求就成为了一个必须的条件。具有保持会话状态的Web应用程序，即可以实时掌握在网站中浏览的特定访问者信息的应用程序，现在已经常见到了无所不在的程度。

跟踪用户相关信息的Web应用程序（例如购物车、在线银行、个人网站门户和公共网络社区等）如此盛行，很难想象我们每天都在使用的Internet，如果没有了保持会话状态的应用程序会变成什么样子。

HTTP —— Web服务器与客户端相互对话的协议，在设计时就是一种无状态的协议。然而，从PHP 4.0开始，用PHP开发应用程序的开发者们就有了一套方便的管理session的函数，这些函数使得实现保持会话状态的任务更加容易。本章集中介绍了一些在开发保持会话状态的应用程序时应该谨记的优秀实践。

Session用于维护请求之间的特定用户状态。某些应用程序在服务器端层次上同样也需要一种轻量级的、能够在一段时间内保存非特定用户状态的等价方案。这个方案就称为数据保持。

技巧11.1解释了PHP的session模块，这个模块可以使你很容易就把进入你网站的用户与要保持的数据结合起来。技巧11.2和11.3分析了session劫持和session定置攻击，以及如何避免这些问题。

在默认情况下，Session数据被保存在服务器端的*/tmp*文件夹下的普通文件中。技巧11.4和11.5介绍了如何在其他可替代的位置（例如，数据库或共享内存）存储session数据，并且讨论了这些不同手段的优势及缺点。

技巧11.6演示了如何为存储session数据之外的目的而共享内存，而技巧11.7则展示了长期存储来自日志文件中摘要信息的技术。

11.1 使用Session跟踪

问题

你想要维护在你的网站中浏览的用户信息。

方案

使用session模块。其中session_start()函数初始化一个session，并访问自动全局数组$_SESSION中的一个元素，通知PHP程序保持对相应变量的跟踪：

```
<?php
session_start();
$_SESSION['visits']++;
print 'You have visited here '.$_SESSION['visits'].' times.';
?>
```

讨论

这个session函数通过发布带有随机生成的sessionID的cookie来掌握用户信息。

默认情况下，PHP把session数据保存在服务器端的*/tmp*文件夹下。每个session都保存为其自己的文件。要想改变保存文件的路径，可以把*php.ini*文件中的session.save_path配置指令设置为新的路径，也可以通过ini_set()函数来设置新路径。还可以用新的路径作为参数调用session_save_path()改变保存文件的路径，但必须要在启用session或者访问任何session变量之前调用这个函数。

若要每次请求都自动启动一个session，可以在*php.ini*文件中把session.auto_start设置为1。设置以后，就不需要再调用session_start()函数了。

当session.use_trans_sid配置指令设置为启用时，如果PHP检测到一个用户不接受会话ID cookie，它会自动地把sessionID添加到URL和表单中（注1），以下面打印一个URL的代码为例：

注1：　在PHP 4.2.0之前，必须明确启用PHP中的enable-trans-sid配置设置才能做到。

```
<?php
print '<a href="train.php">Take the A Train</a>';
?>
```

如果启用了session，但用户不接受cookie，就会有类似下面的这些信息发送到浏览器：

```
<?php
<a href="train.php?PHPSESSID=2eb89f3344520d11969a79aea6bd2fdd">Take the A Train</a>
?>
```

本例中，Session的名称是PHPSESSID，值是2eb89f3344520d11969a79aea6bd2fdd。PHP把这些信息附加到URL上，使这些信息能被传递到后续访问的页面。而表单中则会额外包含一个隐藏的元素，用这个元素来保存并传递sessionID。

由于在URL中附加sessionID的行为可能会导致许多安全隐患，所以这种行为在默认情况下是被禁止的。要想在URL中启用透明sessionID，则需要在*php.ini*文件中把`session.use_trans_sid`设置为启用，或者是session启动之前，在你的脚本中通过`ini_set('session.use_trans_sid', true)`来设置。

设置`session.use_trans_sid`虽然很方便，但却会造成令人头痛的安全性问题。由于URL中包含着sessionID，任何接收到这个URL的人都有可能冒充该sessionID所代表的用户。如果有人在没有防备的情况下，把这样的URL从它的web浏览器复制粘贴到电子邮件当中，并发给了他的朋友，就等于他的朋友（以及其他收到这封转发邮件的人）可以冒充他的身份访问你的网站了。

更糟的是，当一个用户在你的网站上点击了指向外部网站的链接时，用户的浏览器就会把包含sessionID的URL作为引用URL传递到那个外部网站。即使运营那个外部网站的人不会有意识地抓取这类引用URL，但引用日志也经常会在不经意间把这些信息透露给搜索引擎。可以在你常用的搜索引擎中搜一下`PHPSESSID reffer`，你可能会找到一些包含着PHPsessionID的引用日志。

此外，用`Location`头部重定向也不会自动地修改这些ID信息，所以你必须还要自己用SID常量把sessionID添加到其中：

```
$redirect_url = 'http://www.example.com/airplane.php';
if (defined('SID') && (!isset($_COOKIE[session_name()]))) {
    $redirect_url .= '?' . SID;
}

header("Location: $redirect_url");
```

在上面的代码中，`session_name()`函数返回保存sessioniID的cookie名称，所以上面

的代码会在SID常量有定义且没有设置session cookie的情况下，把SID常量追加到$redirect_url的后面。

参见

Sessioin_start()的文档（*http://www.php.net/session-start*）和session_save_path()的文档（*http://www.php.net/session-save-path*）。在session模块中有许多配置指令，可以通过它们设置诸如session的保存时间和session的缓存方式之类的有用功能。其中各种选项的详细介绍可以参考其在线手册（*http://www.php.net/session.*）中的“Session”一节。

11.2 预防Session劫持

问题

你要确保攻击者无法访问到其他用户的sessioin。

方案

只允许通过cookie来传递sessionID，同时生成一个由URL传递的额外session记号（token）。只有当请求包含有效的sessionID和有效的session记号时，才可以访问该session：

```
<?php
ini_set('session.use_only_cookies', true);
session_start();

$salt     = 'YourSpecialValueHere';
$tokenstr = (str) date('W') . $salt;
$token    = md5($tokenstr);

if (!isset($_REQUEST['token']) || $_REQUEST['token'] != $token) {
    // 提示登录
    exit;
}

$_SESSION['token'] = $token;
output_add_rewrite_var('token', $token);
?>
```

如果你使用的是PHP 4.3.0之前的版本，则output_add_rewrite_var()是无效的。因此，要改为使用例11-1中的代码。

例11-1：为链接添加session记号

```
<?php
ini_set('session.use_only_cookies', true);
session_start();

$salt     = 'YourSpecialValueHere';
$tokenstr = (str) date('W') . $salt;
$token    = md5($tokenstr);

if (!isset($_REQUEST['token']) || $_REQUEST['token'] != $token) {
    // 提示登录
    exit;
}

$_SESSION['token'] = $token;

ob_start('inject_session_token');

function inject_session_token($buffer)
{
    $hyperlink_pattern = "/<a[^>]+href=\"([^\"]+)/i";
    preg_match_all($hyperlink_pattern, $buffer, $matches);

    foreach ($matches[1] as $link) {
        if (strpos($link, '?') === false) {
            $newlink = $link . '?token=' . $_SESSION['token'];
        } else {
            $newlink = $link .= '&token=' . $_SESSION['token'];
        }
        $buffer = str_replace($link, $newlink, $buffer);
    }

    return $buffer;
}
```

inject_session_token()函数中为了匹配超链接的正则表达式并不是无懈可击的，它不会匹配href属性使用单引号的超链接。

讨论

本例中，通过把代表当前星期的数字与你自己选择的一句俏皮话连接起来，创建了一个经过自动移位转换后的记号。通过这一技术，记号可以在一个适当的时间段内使用而不必修改。

当我们检查到请求中没有包含这个记号时，我们会提示用户登录。

如果有这个记号，就把它附加到生成的链接上。用output_add_rewrite_var()很容易做到这一点。如果不能使用output_add_rewrite_var()，那我们就要继续生成这个页面，并

且声明一个输出缓冲回调函数，其作用是保证在页面被显示以前，页面中所有的超链接都会包含当前的记号。

注意本例中的inject_session_token()函数，它不会追踪图像分区链接、表单提交信息或者Ajax调用。所以，如有必要，你得自己调整页面功能才能包含已经生成并保存在session中的session记号。

参见

技巧18.1中包含重新生成ID以防止session定置的更多内容。

11.3 预防Session定置

问题

你想确保你的应用程序在面对session定置攻击时不会轻易就范。

方案

要求使用不会把session标识符附加到URL上的session cookie，并且频繁地生成新的sessionID：

```
ini_set('session.use_only_cookies', true);
session_start();
if (!isset($_SESSION['generated'])
    || $_SESSION['generated'] < (time() - 30)) {
    session_regenerate_id();
    $_SESSION['generated'] = time();
}
```

讨论

本例中，我们一开始就把PHP的session行为设置为只能使用cookie。这样就覆盖了PHP的默认行为，即不会在用户浏览器未启用cookie时，自动把用户的sessionID值（以?PHPSESSID=12345678的形式）附加到页面中所有URL上。

当session启动后，我们设置了一个值，用于记录上次生成sessionID的时间。通过请求一个定期（本例中是每30秒）生成的新ID，使得攻击者获取有效sessionID的机会大大降低。

这两种手段的组合可以在实践中消除session定置的风险。一方面，由于sessionID频繁改变，使攻击者难有时机获取有效的sessionID；另一方面，因为sessionID只能在cookie中传递，所以基于URL攻击的可能性为零。最后，由于我们启用了session.use_only_cookie设置，因此也不会有session cookie留在浏览器的历史记录或者服务器的引用日志里面。

参见

“在基于Web的应用程序中预防Session定置攻击”（*http://www.acros.si/papers/session_fixation.pdf*）"；技巧18.1中基于增加特许权限重新生成sessionID的相关内容。

11.4 在数据库中保存Session

问题

你想把session数据保存到数据库而不是保存到文件中。如果多个Web服务器都必须访问同一个数据库，那么就可以在所有Web服务器之间镜像session数据。

方案

使用一个类或一组函数，连同session_set_save_handler()函数来为管理session定义可以操作数据库的例程。例如，使用PEAR的HTTP_Session包就可以方便地实现在数据库中存储session：

```
<?php
require_once 'HTTP/Session/Container/DB.php';

$s = new HTTP_Session_Container_DB('mysql://user:password@localhost/db');
ini_get('session.auto_start') or session_start();
?>
```

讨论

session模块中最强大的一点体现在对如何保存session数据的抽象上。session_get_sav_handler()函数可以告诉PHP对于诸如保存和读取session这样的操作都需要调用什么函数。

PEAR的HTTP_Session包提供了利用其DB、MDB和MDB2数据库抽象包把session数据保存到数据库中的一些类。如果数据库由多个Web服务器共享，那么用户的session信息就可以

实现跨Web服务器的移植。这样，即使你在负载均衡器中绑定了多个Web服务器，也用不着再使用什么奇特的诀窍，就可以确保任何Web服务器发送的用户session数据都能够准确无误了。

而要使用HTTP_Session_Container_DB，需要在技巧化该类时，向其传递一个数据源名（Data Source Name，DSN）。保存session数据的表sessiondate的结构如下：

```
CREATE TABLE sessiondata
(
  id CHAR(32) NOT NULL,
  data MEDIUMBLOB,
  expiry INT UNSIGNED NOT NULL,
  PRIMARY KEY (id)
);
```

如果你想把表名sessiondate改成其他的名字，可以在技巧化HTTP_Session_Container_DB类时，通过一个选项数组来设置新的表名：

```
<?php
require_once 'HTTP/Session/Container/DB.php';

$options = array(
  'table'  => 'php_session',
  'dsn'    => 'mysql://user:password@localhost/db'
);
$s = new HTTP_Session_Container_DB($options);
ini_get('session.auto_start') or session_start();
?>
```

若要定制由HTTP_Session提供的容器类如何操作session数据，可以通过扩展其中的一个容器类来改变相应的行为。这样比重复编写一个新的session处理器类更好。

参见

session_set_save_handler()函数的文档（*http://www.php.net/session-set-save-handler*），有关安装PEAR包（如HTTP_Session）的内容在技巧26.4中介绍。

11.5 在共享内存中保存Session

问题

为了获得最佳的性能，你想把session数据保存到共享内存中。

方案

使用例11-3中的pc_Shm_Session类。例如：

```
<?php
$s = new pc_Shm_Session();
ini_get('session.auto_start') or session_start();
?>
```

讨论

我们在技巧11.4中讨论过，session模块允许用户定义自己的session处理方法。虽然这一灵活性最常见的体现是把session保存到数据库中，但你会发现由于数据库连接和并发查询的开销会带来性能损失。如果所关心的问题并不是在多个Web服务器之间共享session数据，那么就可以通过把相应的数据保存到共享内存中来提升session处理的性能。

在决定用共享内存来存储session之前，要确保能够分出足够的内存空间用于处理访问流量和保存session数据。如果你网站的session消耗了你系统的所有可用内存，那么在共享内存中存储session所带来的性能提升就得不偿失了！

要在共享内存中保存session数据，需要在建立PHP环境时明确地开启--enable-shmop，以保证能够使用操纵共享内存的函数。同时，还需要例11-2中的pc_Shm类及例11-3中的pc_Shm_Session类。

例11-2：pc_Shm类

```
class pc_Shm {

  var $tmp;
  var $size;
  var $shm;
  var $keyfile;

  function pc_Shm($tmp = '') {
    if (!function_exists('shmop_open')) {
      trigger_error('pc_Shm: shmop extension is required.', E_USER_ERROR);
      return;
    }

    if ($tmp != '' && is_dir($tmp) && is_writable($tmp)) {
      $this->tmp = $tmp;
    } else {
      $this->tmp = '/tmp';
    }

    // 默认为 16k
    $this->size = 16384;
```

```
    return true;
  }

  function _ _construct($tmp = '') {
    return $this->pc_Shm($tmp);
  }

  function setSize($size) {
    if (ctype_digit($size)) {
      $this->size = $size;
    }
  }

  function open($id) {
    $key = $this->_getKey($id);
    $shm = shmop_open($key, 'c', 0644, $this->size);
    if (!$shm) {
      trigger_error('pc_Shm: could not create shared memory segment', E_USER_ERROR);
      return false;
    }
    $this->shm = $shm;
    return true;
  }

  function write($data) {
    $written = shmop_write($this->shm, $data, 0);
    if ($written != strlen($data)) {
      trigger_error('pc_Shm: could not write entire length of data', E_USER_ERROR);
      return false;
    }
    return true;
  }

  function read() {
    $data = shmop_read($this->shm, 0, $this->size);
    if (!$data) {
      trigger_error('pc_Shm: could not read from shared memory block', E_USER_ERROR);
      return false;
    }
    return $data;
  }

  function delete() {
    if (shmop_delete($this->shm)) {
      if (file_exists($this->tmp . DIRECTORY_SEPARATOR . $this->keyfile)) {
        unlink($this->tmp . DIRECTORY_SEPARATOR . $this->keyfile);
      }
    }
    return true;
  }

  function close() {
    return shmop_close($this->shm);
  }
```

```
    function fetch($id) {
      $this->open($id);
      $data = $this->read();
      $this->close();
      return $data;
    }

    function save($id, $data) {
      $this->open($id);
      $result = $this->write($data);
      if (! (bool) $result) {
        return false;
      } else {
        $this->close();
        return $result;
      }
    }

    function _getKey($id) {
      $this->keyfile = 'pcshm_' . $id;
      if (!file_exists($this->tmp . DIRECTORY_SEPARATOR . $this->keyfile)) {
        touch($this->tmp . DIRECTORY_SEPARATOR . $this->keyfile);
      }
      return ftok($this->tmp . DIRECTORY_SEPARATOR . $this->keyfile, 'R');
    }
  }
```

这个pc_Shm类对PHP的shmop函数提供了面向对象的包装器。其中的pc_Shm::_getKey()方法提供了一种明确地计算内存地址的便捷方式，而这往往是人们熟悉shmop函数的最大障碍。通过对内存地址的抽象，无论对共享内存是读还是写，都变得如同操作一个关联数组中的值一样容易。

在默认的情况下，pc_Shm会创建16kB的内存块。如果想调整该内存块的大小，只需给pc_Shm::setSize()方法传递一个以字节计量的值即可。

在定义了pc_Shm之后，pc_Shm_Session类又定义了它所需要的，为session_set_save_handler()方便地提供自定义方法的功能。例11-3中是pc_Shm_Session类的定义。

例11.3：pc_Shm_Session类

```
class pc_Shm_Session {

  var $shm;

  function pc_Shm_Session($tmp = '') {
    if (!function_exists('shmop_open')) {
      trigger_error("pc_Shm_Session: shmop extension is required.",E_USER_ERROR);
      return;
    }
```

```
    if (! session_set_save_handler(array(&$this, '_open'),
                    array(&$this, '_close'),
                    array(&$this, '_read'),
                    array(&$this, '_write'),
                    array(&$this, '_destroy'),
                    array(&$this, '_gc'))) {
      trigger_error('pc_Shm_Session: session_set_save_handler() failed', E_USER_ERROR);
      return;
    }

    $this->shm = new pc_Shm();

    return true;
  }

  function _ _construct() {
    return $this->pc_Shm_Session();
  }

  function setSize($size) {
    if (ctype_digit($size)) {
      $this->shm->setSize($size);
    }
  }

  function _open() {
    return true;
  }

  function _close() {
    return true;
  }

  function _read($id) {
    $this->shm->open($id);
    $data = $this->shm->read();
    $this->shm->close();
    return $data;
  }

  function _write($id, $data) {
    $this->shm->open($id);
    $this->shm->write($data);
    $this->shm->close();
    return true;

  }

  function _destroy($id) {
    $this->shm->open($id);
    $this->shm->delete();
    $this->shm->close();
  }
```

```
    function _gc($maxlifetime) {
      $d = dir($this->tmp);
      while (false !== ($entry = $d->read())) {
        if (substr($entry, 0, 6) == 'pcshm_') {
          $tmpfile = $this->tmp . DIRECTORY_SEPARATOR . $entry;
          $id = substr($entry, 6);
          $fmtime = filemtime($tmpfile);
          $age = now() - $fmtime;
          if ($age >= $maxlifetime) {
            $this->shm->open($id);
            $this->shm->delete();
            $this->shm->close();
          }
        }
      }
      $d->close();
      return true;
    }

}
```

微软Windows 2000之前的版本中没有提供对共享内存的支持。并且，如果是在Windows服务器环境中使用PHP，也只有当PHP是作为一个Web服务器模块（类似Apache或IIS所提供的功能）的情况下，才能使用shmop函数。CLI与CGI的PHP的接口在Windows平台下不支持shmop函数。

也可能你根本不会用到这里定义的类。如果你的Web服务器可以配置为支持随机磁盘分区，例如/dev/shm，那么通过共享内存来保存sessions可能就会变得像下面这样简单：

```
<?php
ini_set('session.save_path', '/dev/shm');
ini_get('session.auto_start') or session_start();
?>
```

参见

session_set_save_handler()函数的文档（*http://www.php.net/session-set-save-handler*）。shmop函数的文档（*http://www.php.net/shmop*）。有关在基于Linux的系统中配置随机磁盘的信息（*http://www.linuxhq.com/kernel/file/Documentation/ramdisk.txt*）。

11.6 在共享内存中保存独立数据

问题

你想在共享内存中来保存一段所有Web服务器进程都可以访问的数据。

方案

使用例11-3中定义的pc_Shm类。例如，要在共享内存中保存一个字符串，可以使用pc_Shm::save()方法，该方法接受一个键/值对参数：

```
<?php
$shm = new pc_Shm();
$secret_code = 'land shark';
$shm->save('mysecret', $secret_code);
?>
```

然后，其他的进程可以通过pc_Shm::fetch()方法来访问保存于共享内存中的这个数据：

```
<?php
$shm = new pc_Shm();
print $shm->fetch('mysecret');
?>
```

讨论

有时候，为了快速检索的需要，你可能会在共享内存中缓存一个或一组值。如果你的Web服务器的磁盘I/O很繁忙，那么利用shmop函数在高速缓存中实现高性能的信息存储和检索就有意义了。

pc_Shm类有两个便捷的方法，pc_Shm::fetch()和pc_Shm::save()。这两个方法抽象了设置内存地址和明确地打开并关闭共享内存片段的功能。

一个必须要记住的重要的问题是，与在常规的PHP数组中设置键/值对不同，shmop函数需要为存储数据预期所需的耗用分配特定的空间。在默认的情况下，pc_Shm类为每个值分配了16kB空间。如果要保存的数据大于16kB，则必须增大shmop函数应该保留的内存空间。例如：

```
<?php
$shm = new pc_Shm();
$shm->setSize(24576); // 24k
$shm->save('longstring', 'Lorem ipsum pri eu simul nominati...');
?>
```

参见

技巧11.5和技巧5.6，PHP在线手册的Memcache部分（*http://www.php.net/memcache*）。Memcache是替代shmop函数的一种快速而且有效的方案。有关Memcache的更多信息可

以访问*http://www.danga.com/memcached/*。此外，PECL的apc模块（*http://pecl.php.net/apc*）也提供了在共享内存中保存数据的功能。

11.7 在摘要表中缓存计算结果

问题

你想在那些因为过于庞大而无法有效地进行实时查询的日志表中收集统计信息。

方案

创建一个表来保存完整日志表的摘要信息，查询该摘要表来生成接近实时的报告。

讨论

假设你记录了网站访问者使用Google或Yahoo之类的搜索引擎搜索你网站的记录，同时在MySQL中跟踪那些查询。你的搜索项目跟踪日志表具有以下结构：

```
CREATE TABLE searches
(
  searchterm    VARCHAR(255) NOT NULL,  # search term determined from HTTP_REFERER
                                        parsing
  dt            DATETIME NOT NULL,      # request date
  source        VARCHAR(15) NOT NULL    # site where search was performed
);
```

如果够幸运的话，你每小时可能会记录到数千甚至上万条来自主要搜索引擎的查询记录，而几个月后你的searchs表也会迅速增大到难以处理的程度。

你可能希望生成能够说明在过去一段时间里，来自主要搜索引擎的查询中所包含的搜索项目倾向性的报告，以便决定购买哪个搜索引擎的广告。

创建一个反应报告所需要信息的摘要表，然后每小时查询一次完整的数据集，并把结果保存到摘要表中以加快报告生成期间的检索速度。这个摘要表的结构可能是这样的：

```
CREATE TABLE searchsummary
(
  searchterm    VARCHAR(255) NOT NULL,  # search term
  source        VARCHAR(15) NOT NULL,   # site where search was performed
  sdate         DATE NOT NULL,          # date search performed
  searches      INT UNSIGNED NOT NULL,  # number of searches
  PRIMARY KEY (searchterm, source, sdate)
);
```

之后，生成报告的脚本可以使用PDO来查询这个searchsummary表，如果结果不可用，则从searches表中搜集相关信息并将结果缓存到searchsummary表中：

```
$st = $db->prepare('SELECT COUNT(*)
                    FROM
                       searchsummary
                    WHERE
                       sdate = ?');
$st->execute(array(date('Y-m-d', strtotime('yesterday'))));

$row = $st->fetch();

// 缓存中没有匹配的信息
if ($row[0] == 0) {
    $st2 = $db->prepare('SELECT
                           searchterm,
                           source,
                           FROM_DAYS(TO_DAYS(dt)) AS sdate,
                           COUNT(*) as searches
                         WHERE
                           TO_DAYS(dt) = ?');
   $st2->execute(array(date('Y-m-d', strtotime('yesterday'))));

   $stInsert = $db->prepare('INSERT INTO searchsummary
                             (searchterm,source,sdate,searches)
                             VALUES (?,?,?,?)');
   while ($row->fetch(PDO::FETCH_NUM)) {
       $stInsert->execute($row);
   }
}
?>
```

通过使用这一技术，你的脚本只引起了一次查询整个日志表的开销，而后续的请求都只是针对每个搜索项目检索一行摘要数据。

参见

10.7中有关PDO::prepare()和PDOStatement::execute()方法的信息。

第12章

XML

12.0 概述

XML作为一种数据交换和信息传递的格式已经得到了普及。随着Web 服务日益广泛的应用，XML在开发人员的日常工作中也扮演了愈来愈重要的角色。PHP借助于一些扩展可以让我们在各种应用中实现对XML的操纵。

XML为开发人员提供了一种用事先规划好的标记，按照树形层次来标记数据的结构化方式。一种看法认为XML是一种与CSV相近的“类固醇（steroid）”。它可以用来保存分为一系列字段的记录。但并不是像CSV那样只能用逗号来分隔每个字段，而是可以同时包含数据的字段名、类型和属性信息。

还有一种观点认为XML是一种用于文档表现的语言。举例来说，假如本书就是使用XML来编写的。那么，这本书分成了很多章，每一章又分成了几个技巧，而每个技巧中又分为问题、方案和讨论几个部分。在每个部分中，我们还可以进一步把文本细分为段落、表格、插图和例子。而一篇网页中的文章也可以类似地分成页面标题、文章标题、文章作者、文章内容及一些选项栏、相关链接和其他内容。

XML文档从内容上看与HTML类似。两者都使用“<”和“>”括起来的标签来标记文本。但是，XML却比HTML更严格而且也更宽松。说它更严格是因为所有容器标签都必须适当地关闭。即不允许一个元素有开标签但没有对应的闭标签。说它更宽松的意思是XML不会强迫你只能使用一组给定的标签，比如<a>、<img>和<h1>。相反，你可以自主地选择一组能够描述你的数据的最合适的标记名。

XML同HTML之间另外一些关键的差别还包括大小写敏感性、属性是否带引号和空格的处理。在HTML中，<B>和<b>都是同样的粗体标签；而在XML中，它们则是不同的标签。在HTML中，通常可以省略属性值两边的引号；但在XML中则必须要给属性值加引号。也就是说，必须写成下面这样：

```
<element attribute="value">
```

另外，HTML解析器通常会忽略空格，所以连续的20空格也被看成是一个空格。XML解析器则会保留空格，除非明确地指令以其他方式来处理。因为所有元素必须关闭，所以空元素也必须以/>结尾。例如，在HTML中，换行标签是
，而在XHTML（经XML规则验证有效的HTML）中则必须写成
（注1）。

XML文档还有一种限制。当把XML文档解析为一个元素树的情况下，其最外层的元素被称为根元素。就和树只有一根主干一样，XML文档中也只能有一个根元素。在前面书的例子中，就意味着所有各章必须位于一个book标签中。如果要在一个文档中保存多本书，就需要将这些书打包放到一个bookcase或者其他什么容器标签中。这个限制只适用于文档的根元素。同样地，树干可以有多个分支，而一个书架（bookcase）中保存多本书也是合理的。

本章并不是专门讲述XML的，如果要了解XML可以参考Erik T. Ray的书《Learning XML》（O'Reilly）。还有一本详尽地介绍了XML方方面面的书《XML in a Nutshell》，作者是Elliotte Rusty Harold 和 W. Scott Means（O'Reilly）。

既然我们已经明确了规则，下面就来看一个例子：假如你是一名图书管理员，想把卡片目录转换为XML文件，那么就可以从以下基本的XML标签开始做起：

```
<book>
    <title>PHP Cookbook</title>
    <author>Sklar, David and Trachtenberg, Adam</author>
    <subject>PHP</subject>
</book>
```

以此为基础，你还可以添加新的元素或者修改现有的元素。例如，<author>可以分成姓和名，或者也可以添加另外一个相同的元素以便两名作者不必放在一个字段中。

PHP 5中有一组主要应对PHP 4的XML扩展中存在问题的全新XML扩展。PHP 4虽然允许你操纵XML，但其XML工具之间只具有简单的关联。虽然每个工具都涵盖了XML应用的一部分，但却不能在一起协同工作，而且PHP 4对于更多高级XML特性的支持经常存在着不一致的问题。但在PHP 5中已经不再是这种情况了。PHP 5中新的XML扩展具有以下特点：

- 能够像一个整体一样协调工作。
- 是一个标准化的XML库：libxml2。

注1：　这就是为什么nl2br()会输出
的原因；其输出是XML兼容的。

- 完全遵照W3C规范。
- 更有效地处理数据。
- 是你工作中合适的XML工具。

此外，遵循PHP“创建Web应用程序应该很容易”的原则，还有一个新的XML扩展用于简化对XML文档的读取和修改。而这个名副其实的扩展就是SimpleXML，它可以让你在对XML文档中的信息操作时，就如同在操纵数组和对象中的数据一样简单，可以使用foreach循环进行迭代，而且仅仅通过把新值赋给一个变量就能修改其中的数据。

本章的前两个技巧是关于生成XML的。技巧12.1中介绍了如何在不使用额外工具的情况下生成XML。如果想通过DOM扩展来以标准化的风格生成XML文件，可以参考12.2。

在讨论完生成XML之后要讨论的是解析XML。这正是接下来三个技巧的主题。这几个主题是按照要解析的XML文档的复杂程度和大小进行划分的。技巧12.3中包括了如何解析基本的XML文档的问题。如果你需要更加完善的XML解析工具，可以把书翻到技巧12.4。如果你的XML文档非常大并且内存紧张，可以在技巧12.5中找到解决方案。如果这是你第一次使用XML，并且还不确定哪一个技巧适合你，那最好是按照顺序来阅读各个技巧，因为随着你需求的增长技巧中的代码也会相应更加复杂。

XPath是技巧12.6中的主题。它是一个从XML文档中提取特定信息的W3C标准。我们喜欢把它理解成针对XML的正则表达式。XPath是XML规范家族中最有用但却没有被广泛使用的一个标准。如果你想根据规范的原则来处理XML，就应该熟悉XPath。

通过XSLT，你可以用一个XSL样式表把XML转换为可见的输出。在内容与表现分离的基础上，你可以为Web浏览器编写一个样式表，为PDA编写另一个样式表，还可以为手机再编写第三个样式表，满足所有这些显示需求都不用改变XML文档的内容。这些都是技巧12.7中要讨论的主题。

在介绍完XSLT之后，随后的两个技巧是有关如何在PHP和XSLT之间传递信息的。12.8中讨论了如何把数据从PHP发送到XLST样式表，技巧12.9则展示了如何从一个XLST样式表内部将信息传递到PHP。

只要你的XML文档遵守XML结构化规则，就可以称之为格式良好。但与HTML不同的是，HTML文档必须有一组特殊的元素和属性放在某个地方，而XML则没有这个限制。

然而，在有些情况下，比如XHTML（HTML的XML版）中，确保你的XML文档遵守相应的规范是很重要的。这样就可以使得一些工具，比如Web浏览器、RSS阅读器或者你

自己编写的脚本，都能够很容易地处理输入的内容。当一个XML文档遵循了规范所规定的全部规则时，就可以说它是有效的。12.10中包含了如何验证XML文档有效性的介绍。

PHP 5的一个主要局限性表现在它对字符集和文档编码的处理上。PHP中的字符串并不与特定的编码关联，但是所有XML扩展都要求以UTF-8格式输入和输出。因此，如果你使用的字符集不能与UTF-8兼容，就必须在发送数据到XML扩展前和从XML扩展取得数据后相应地进行两次手工转换。技巧12.11中探讨了处理这一过程的最佳途径。

本章最后的几个技巧涉及到读写一些常见的XML类型文档，特别是RSS和Atom。这是两个最流行的数据整合格式，它们对于博客文章(blog）、播客条目（podcast）甚至地图（mapping）信息之间的数据格式交换具有重要意义。

PHP经典技巧也会涵盖所有流行的Web 服务类型：REST、XML-RPC和SOAP。因为这个话题实在是太重要了，所以我们专门给它们设置了两章。第14章介绍了如何消费Web服务，而第15章讨论了如何实现你自己的Web 服务。

12.1 生成XML为字符串

问题

你想要生成XML文档。例如，你想为满足另一个程序的解析需求向其提供一根据你的数据生成的XML文档。

方案

循环遍历你的数据并在输出时加上正确的XML标签：

```
<?php
header('Content-Type: text/xml');
print '<?xml version="1.0"?>' . "\n";
print "<shows>\n";

$shows = array(array('name'     => 'Simpsons',
                     'channel'  => 'FOX',
                     'start'    => '8:00 PM',
                     'duration' => '30'),

               array('name'     => 'Law & Order',
                     'channel'  => 'NBC',
                     'start'    => '8:00 PM',
                     'duration' => '60'));
```

```
foreach ($shows as $show) {
    print "    <show>\n";
    foreach($show as $tag => $data) {
        print "        <$tag>" . htmlspecialchars($data) . "</$tag>\n";
    }
    print "    </show>\n";
}

print "</shows>\n";
?>
```

讨论

手工输出XML主要涉及到在枚举数组中的数据时用到的许多foreach循环。不过，也有一些技巧性的细节需要关注。首先，需要调用header()为该文档设置正确的Content-Type。因为你要发送的是XML而不是HTML，所以应该用text/xml。

其次，取决于你的short_open_tag配置指令的设置，尝试输出XML声明的时候可能会意外地开启PHP进程。因为<?xml versioni="1.0"?>中的<?是PHP开标签的简写方式。如果要正确地将声明输出到浏览器，要么禁用该指令，要么就要在PHP内部输出该行。在这里的解决方案中我们采取后者。

最后，必须对实体进行转义。例如，“Law & Order”中的&符号必须转义为&。在这里就是要调用htmlspecialchars()来对数据进行转义处理。

这个解决方案中的例子输出的结果如例12-1所示。

例12-1：今晚的电视节目

```
<?xml version="1.0"?>
<shows>
    <show>
        <name>Simpsons</name>
        <channel>FOX</channel>
        <start>8:00 PM</start>
        <duration>30</duration>
    </show>
    <show>
        <name>Law &amp; Order</name>
        <channel>NBC</channel>
        <start>8:00 PM</start>
        <duration>60</duration>
    </show>
</shows>
```

参见

技巧12.2中有关使用DOM扩展生成XML文档的介绍。htmlspecialchars()函数的文档（*http://www.php.net/htmlspecialchars*）。

12.2 通过DOM生成XML

问题

你想生成XML文档，但想通过一种有序的方式而不是输出命令和循环语句。

方案

通过DOM扩展来创建一个DOMDocument对象。在建立了文档之后，调用DOMDocument::save()或DOMDocument::saveXML()方法生成一个格式良好的XML文档：

```
<?php
// 创建一个新文档
$dom = new DOMDocument('1.0');

// 创建根元素，<book>，并将其添加到文档
$book = $dom->appendChild($dom->createElement('book'));

// 创建一个title元素并将其添加到$book中
$title = $book->appendChild($dom->createElement('title'));

// 设置title元素中的文本及cover属性的值
$title->appendChild($dom->createTextNode('PHP Cookbook'));
$title->setAttribute('cover', 'soft');

// 创建并将author元素添加到$book中
$sklar = $book->appendChild($dom->createElement('author'));
// 为每个元素创建并添加文本
$sklar->appendChild($dom->createTextNode('Sklar'));

$trachtenberg = $book->appendChild($dom->createElement('author'));
$trachtenberg->appendChild($dom->createTextNode('Trachtenberg'));

// 将一个完美格式化的DOM文档输出为XML
$dom->formatOutput = true;
echo $dom->saveXML();
?>
<?xml version="1.0"?>
<book>
  <title cover="soft">PHP Cookbook</title>
  <author>Sklar</author>
  <author>Trachtenberg</author>
</book>
```

讨论

这里的DOM方法遵循了一种模式：创建一个元素或者文本节点对象，添加或设置想要的属性，然后把它们添加到所属的位置上。

在创建元素之前，先创建一个新文档对象，并传递一个XML版本号作为唯一的参数：

```
$dom = new DOMDocument('1.0');
```

现在创建一个属于该文档的新元素。无论是否与特定的文档有关联，节点在被添加之前都不会加入文档树中：

```
$book_element = $dom->createElement('book');
$book = $dom->appendChild($book_element);
```

这里创建了一个新的book元素并将其指定给$book_element对象。要创建文档的根元素，必须将$book_element作为$dom文档对象的子元素添加进来。结果是，$book引用了一个特殊的元素并且其位置位于DOM对象之中。

所有节点都是通过调用$dom对象的方法来创建的。在创建节点后，可以把该节点添加到树中的任何元素上。我们以之调用appendChild()方法的那个元素决定了节点在树中所处的位置。在前面的例子中，$book_element被添加到了$dom。而添加到$dom的元素是顶级节点，或者说是根节点。

也可以把一个新元素添加到$book中。因为$book是$dom的一个子元素，所以这个新元素，更进一步讲，就是$dom的孙子元素：

```
$title_element = $dom->createElement('title');
$title = $book->appendChild($title_element);
```

通过调用$book->appendChild()，该代码就把$title_element元素放到了$book元素的内部。

要在<title></title>内部添加文本内容，需要先用createTextNode()创建一个文本节点，然后再把这个节点添加到$title：

```
$text_node = $dom->createTextNode('PHP Cookbook');
$title->appendChild($text_node);
```

因为$title已经添加到了文档中，所以不需要再重复把它添加$book。

也就是说，将子元素添加节点的顺序并不重要。像下面这四行代码，首先是把文本节点添加到$title_element，然后再添加到$book，这与前面代码的效果是相同的：

```
$title_element = $dom->createElement('title');
$text_node = $dom->createTextNode('PHP Cookbook');

$title_element->appendChild($text_node);
$book->appendChild($title_element);
```

要添加属性，就在一个节点上调用setAttribute()方法，并传递属性的名和值作为其参数：

```
$title->setAttribute('cover', 'soft');
```

如果你现在输出title元素，结果会是这样的：

```
<title cover="soft">PHP Cookbook</title>
```

在完成之后，可以把这个文档作为字符串输出，也可以输出为一个文件：

```
// 把XML文档的字符串表现形式放在$books中
$books = $dom->saveXML();

// 把XML文档输出为books.xml文件
$dom->save('books.xml');
```

在默认情况下，这两个方法生成的XML输出都会放在长长的一行中，并且不包含任何空格、缩进和换行。要解决这个问题，需要把DOMDocument对象的formatOutput属性设置为true：

```
// 把DOM文档输出为一个格式良好的XML文档
$dom->formatOutput = true;
```

这样，就会使DOM扩展生成的XML文档像下面这样了：

```
<?xml version="1.0"?>
<book>
  <title cover="soft">PHP Cookbook</title>
</book>
```

参见

技巧12.1中有关不通过DOM扩展生成XML文档的内容。技巧12.4中有关通过DOM扩展解析XML文档的讨论。DOMDocument对象的文档（*http://www.php.net/function.dom-domdocument-construct.php*）和常用的DOM函数（*http://www.php.net/dom*）以及有关基础的libxml2 C库的信息（*http://xmlsoft.org/*）。

12.3 解析基本的XML文档

问题

你想解析一个遵循已知模式（Schema）的基本XML文档，同时也不需要访问更多的XML特性，比如处理指令等。

方案

使用SimpleXML扩展。下面演示了如何从一个文档中读取XML：

```
<?php
$sx = simplexml_load_file('address-book.xml');

foreach ($sx->person as $person) {
    $firstname_text_value = $person->firstname;
    $lastname_text_value = $person->lastname;

    print "$firstname_text_value $lastname_text_value\n";
}
?>
David Sklar
Adam Trachtenberg
```

讨论

SimpleXML被描述为“有史以来最佳的方案”。虽然要与这么高的赞誉相称也很难，但SimpleXML确实在简化与XML的交互方面 —— 我们敢说 —— 实现了令人瞩目的进步。当我们在读取XML文件中的配置信息、解析RSS源或者处理REST请求的结果时，SimpleXML都是非常得力的助手。但是，它对于更复杂的XML相关的任务则不一定能胜任，比如读取一个事先并不知道格式的文档，需要访问处理指令或注释的情况等。

SimpleXML把元素转换成对象属性。位于标签之间的文本被指定给属性。如果同一个位置上有多个同名元素（比如多个<people>），那么这些元素会被放在一个列表中。

元素的属性会转换为数组元素，其中数组的键是属性名，键的值就是属性的值。

要访问一个单独的值，可以直接使用对象方法来引用那个值。我们用下面的这个XML片断作例子：

```
<firstname>David</firstname>
```

如果将这行代码放在一个SimpleXML对象中，就是$firstname，而要访问David所需的只有：

```
$firstname
```

SimpleXML假定当一个节点只包含文本时，你所感兴趣的是其中的文本。因此，输出$firstname的确能如你所愿 —— 这就是David。

而迭代方法，像foreach，是一种循环遍历多个元素的最佳选择。后面的例子中显示了使用迭代方法的代码。

属性被保存为数组元素。例如，下面的代码会输出第一个person元素的id属性：

```
<?php
$ab = simplexml_load_file('address-book.xml');

// 每个person元素的id属性
print $ab->person ['id']  . "\n";
?>
```

它会给你的结果是：

```
1
```

例12-2中包含一个基于XML简单地址簿的更复杂一些的例子。它被用在了接下来的例子代码中。

例12-2：XML格式的简单地址簿

```
<?xml version="1.0"?>
<address-book>
    <person id="1">
        <!--David Sklar-->
        <firstname>David</firstname>
        <lastname>Sklar</lastname>
        <city>New York</city>
        <state>NY</state>
        <email>sklar@php.net</email>
    </person>

    <person id="2">
        <!--Adam Trachtenberg-->
        <firstname>Adam</firstname>
        <lastname>Trachtenberg</lastname>
        <city>San Francisco</city>
        <state>CA</state>
        <email>amt@php.net</email>
    </person>
</address-book>
```

例12-3显示了如何使用SimpleXML取出名和姓。

例12-3：使用SimpleXML析取数据

```
$sx = simplexml_load_file('address-book.xml');

foreach ($sx->person as $person) {
    $firstname_text_value = $person->firstname;
    $lastname_text_value = $person->lastname;

    print "$firstname_text_value $lastname_text_value\n";
}
```

```
David Sklar
Adam Trachtenberg
```

在使用SimpleXML的时候，可以直接使用foreach语句来迭代每个元素。这里，迭代发生在$sx->person处，其中保存着所有person节点。

也可以像例12-4那样，直接输出SimpleXML对象。

例12-4：输出SimpleXML对象

```
<?php
foreach ($sx->person as $person) {
    print "$person->firstname $person->lastname\n";
}
?>
```

```
David Sklar
Adam Trachtenberg
```

PHP会把SimpleXML对象插入引用的字符串中，并取得保存于其中的文本。

参见

技巧12.4中有关解析复杂XML文档的内容。技巧12.5中解析大型XML文档的讨论。SimpleXML的文档（*http://www.php.net/simplexml*）和基础的libxml2 C库的信息（*http://xmlsoft.org/*）。

12.4 解析复杂的XML文档

问题

你有一份复杂的XML文档，比如需要分析之后才能确定其所遵循的模式（Schema），或者是你需要使用更高级的XML特性，如处理指令或注释。

方案

使用DOM扩展。该扩展为XML规范的各个方面提供了完整的接口。

```
<?php
$dom = new DOMDocument;
$dom->load('address-book.xml');

foreach ($dom->getElementsByTagname('person') as $person) {
    $firstname = $person->getElementsByTagname('firstname');
    $firstname_text_value = $firstname->item(0)->firstChild->nodeValue;

    $lastname = $person->getElementsByTagname('lastname');
    $lastname_text_value = $lastname->item(0)->firstChild->nodeValue;

    print "$firstname_text_value $lastname_text_value\n";
}
?>
David Sklar
Adam Trachtenberg
```

讨论

W3C的DOM规范提供了一种指定文档结构和内容的、平台和语言中立的方法。通过DOM，我们可以把XML文档创建为一个节点树，之后可以在这个树状结构中随意穿行，寻找一个特殊的元素或者那些匹配我们标准的元素。这个过程叫做基于树状结构的解析。

另外，你可以通过创建、编辑及删除节点的方式来修改文档的结构。实际上，你甚至可以使用DOM函数直接创建一个新的XML文档（见技巧12.2）。

DOM的一个主要的优势在于，由于遵循W3C的规范，许多语言都以差不多相同的方式实现了DOM函数。因此，从一个应用程序向另一个应用程序转换的逻辑和指令过程都相当简单。PHP 5中提供了一系列新的DOM方法，这些方法比前一个版本中的方法都更严格地遵守了DOM标准。

DOM规范既庞大也复杂。如果要了解有关DOM的更多信息，可以到*http://www.w3.org/DOM/*中查看或者在《XML in a Nutshell》中查看该规范的一份副本。

DOM函数在PHP中是面向对象的。要想从一个节点移动到另一个节点，可以通过`$node->childNodes`或`$node->parentNode`来访问节点对象的属性，前者是一个包含子节点对象的数组，后者则是该节点的父节点对象。因此，处理节点就是要检查节点的类型并调用相应的方法，如例12-5所示。

例12-5：解析一个DOM对象

```
<?php
// $node 是DOM解析后的节点 <book cover="soft">PHP Cookbook</book>
$type = $node->nodeType;

switch($type) {
case XML_ELEMENT_NODE:
    // 我是一个标签。我有一个tagname属性
    print $node->tagName;  // 输出标签名属性: "book"
    break;
case XML_ATTRIBUTE_NODE:
    // 我是一个属性。我有名和值属性
    print $node->name;  // 输出名称属性: "cover"
    print $node->value; // 输出值属性: "soft"
    break;
case XML_TEXT_NODE:
    // 我是元素中的一段文本
    // 我有名称和内容属性
    print $node->nodeName;  // 输出名称属性: "#text"
    print $node->nodeValue; // 输出文本内容: "PHP Cookbook"
    break;
default:
    // 其他类型
    break;
}
?>
```

要自动地搜索一遍DOM树以寻找特定的元素，使用getElementsByTagname()方法。例12-6显示了如何用该方法处理多本图书记录。

例12-6：XML格式的卡片目录

```
<books>
    <book>
        <title>PHP Cookbook</title>
        <author>Sklar</author>
        <author>Trachtenberg</author>
        <subject>PHP</subject>
    </book>
    <book>
        <title>Perl Cookbook</title>
        <author>Christiansen</author>
        <author>Torkington</author>
        <subject>Perl</subject>
    </book>
</books>
```

例12-7显示如何找到所有作者。

例12-7：用DOM方法查找输出所有作者

```
// 找到所有作者并输出
$authors = $dom->getElementsByTagname('author');
```

```
// 遍历所有author元素
foreach ($authors as $author) {
    // 子节点中保存关author元素的值
    $text_nodes = $author->childNodes;

    foreach ($text_nodes as $text) {
        print $text->nodeValue . "\n";
    }
}
```

```
Sklar
Trachtenberg
Christiansen
Torkington
```

getElementsByTagname()方法返回一个包含元素节点对象的数组。通过遍历每个元素的子元素，可以得到与每个元素关联的文本节点。然后可以在这些文本节点中取出节点的值，在这个例子中也就是图书作者的名字，如Sklar和Trachtenberg。

参见

技巧12.3中有关解析简单XML文档的内容。技巧12.5中有关解析大型XML文档的讨论。DOM在PHP中实现的文档（*http://www.php.net/dom*）和基础的libxml2 C库的信息（*http://xmlsoft.org/*）。

12.5 解析大型XML文档

问题

你想要解析一个大型的XML文档。由于这个大型文档无法全部载入内存中，所以不可能使用SimpleXML或DOM扩展。所以，每次只能加载文档的一部分。

方案

使用XMLReader扩展：

```
<?php
$reader = new XMLReader();
$reader->open('card-catalog.xml');

/* 循环遍历文档 */
while ($reader->read()) {
    /* 如果当前是名为'author'的元素 */
    if ($reader->nodeType == XMLREADER::ELEMENT && $reader->localName == 'author') {
```

```
        /* 找到其中的文本节点并将其值输出 */
        $reader->read();
        print $reader->value . "\n";
    }
}
?>
```


讨论

有两种主要的XML解析器：一种需要把整个文档都加载到内存中才能工作；而另一种则在任何给定的时间内只需把文档的一小部分加载到内存中就可以工作。

第一种解析器称为基于树的解析器，因为它需要把整个文档存储为一种称为树的数据结构中。技巧12.3和12.4中介绍SimpleXML和DOM扩展都属性这种基于树的解析器。使用基于树的解析器对我们而言虽然容易，但它要求PHP占用更多的内存。对于多数XML文档来说，这不是问题。然而，如果你的XML文档相当大，那么这种处理方式可能就会导致很大的性能问题。

另一种解析器称为基于流的解析器。所谓基于流是指解析器不会一次就把整个文档都加载到内存中，而是每次分别读取其中的一个节点并允许你实时地与之交互。而当移向下一个节点时，上一个节点就会被抛弃 —— 除非你为了将来使用而明确地将其保存起来。这种机制使得基于流的解析器的速度更快同时消耗的内存也更少，但是可能必须要编写更多的代码来完成对文档的处理。

使用基于流的解析器来处理XML数据最简单的方式，就是使用XMLReader扩展。这个扩展是基于C# XMLTextReader API的。如果你熟悉PHP4的SAX（Simple API for XML）接口，那么在PHP 5中该接口仍然是有效的，但XMLReader扩展则更加直观、功能丰富并且速度也更快。

XMLReader在PHP 5.1中默认是启用的。如果你使用的是PHP 5.0.x，可以自己从PECL（*http://pecl.php.net/package/xmlReader*）上面下载并安装这个扩展。

使用XMLReader扩展，首选是要创建一个XMLReader类的新技巧，并指定要处理的XML数据的位置：

```
<?php
// 创建一个新的 XMLReader 对象
$reader = new XMLReader();

// 从一个文档或URL中加载XML数据
$reader->open('document.xml');
```

```
// 或者，也可以从一个PHP变量中加载

$reader->XML($document);
?>
```

多数情况下，我们都要使用XMLReader::open()方法从一个外部的源中抽取数据，但是也可以通过XMLReader::XML()来从一个现有的PHP变量中加载数据。

当把对象配置完成后，就可以着手处理其中的数据了。一开始，我们的位置就在文档的顶部。这时可以通过组合运用XMLReader提供的两个导航方法XMLReader::read()和XMLReader::next()在文档中随意地移动位置。第一个方法用来读取直接位于当前位置之后的XML数据片段。第二个方法则可以将当前位置移动到下一个相邻同辈的子元素上。

例如，我们来看一看例12-8中的XML文档。

例12-8：XML格式的卡片目录

```
<books>
    <book isbn="1565926811">
        <title>PHP Cookbook</title>
        <author>Sklar</author>
        <author>Trachtenberg</author>
        <subject>PHP</subject>
    </book>
    <book isbn="0596003137">
        <title>Perl Cookbook</title>
        <author>Christiansen</author>
        <author>Torkington</author>
        <subject>Perl</subject>
    </book>
</books>
```

在对象的当前位置处于第一个<book>元素上面时，read()方法会把我们的目标移动到<book>下面紧邻的元素上（从技术角度来看，是根据<book>和<title>之间的空白符来确定位置的）。相对地，next()方法则会把我们的目标移动到下一个<book>元素上，也就是跳过整个*PHP Cookbook*这个子树。

当这两个方法能够成功地移动到另一个节点上面时，会返回true；如果移动失败，则返回false。所以，下面代码中的while循环是使用这两种方法的典型方式：

```
/* 循环遍历整个文档 */
while ($reader->read()) {
    /* 处理XML */
}
```

这样，就会使$reader对象在整个XML文档中每次读取一段数据。在while循环内部，我们可对$reader进行检查，并执行相应的处理。

常见的检查对象就是节点类型。通过检查节点类型，可以确定当前位置是不是一个元素（然后，可以再检查元素的名称），还是一个结束的元素、属性、一段文本、一些空白符或者其他XML文档的构成要素。检查节点类型是通过引用nodeType属性来进行的：

```
/* 循环遍历整个文档 */
while ($reader->read()) {
    /* 如果当前是名为'author'的元素 */
    if ($reader->nodeType == XMLREADER::ELEMENT && $reader->localName == 'author') {
        /* 处理author元素 */
    }
}
```

这里的代码检查了当前的节点是否是一个元素，如果是，其名字是否为author。要了解nodeType中可能包含的所有值的列表，请看表12-1。

表12-1：XMLReader节点类型值

节点类型	说明
XMLReader::NONE	非节点类型
XMLReader::ELEMENT	开始元素
XMLReader::ATTRIBUTE	属性节点
XMLReader::TEXT	文本节点
XMLReader::CDATA	CDATA节点
XMLReader::ENTITY_REF	实体引用节点
XMLReader::ENTITY	实体声明节点
XMLReader::PI	处理指令节点
XMLReader::COMMENT	注释节点
XMLReader::DOC	文档节点
XMLReader::DOC_TYPE	文档类型节点
XMLReader::DOC_FRAGMENT	文档片断节点
XMLReader::NOTATION	符号节点
XMLReader::WHITESPACE	空白节点
XMLReader::SIGNIFiCANT_WHITESPACE	显著空白节点
XMLReader::END_ELEMENT	结束元素
XMLReader::END_ENTITY	结束实体
XMLReader::XML_DECLARATION	XML声明节点

据此，我们就可以确定如何处理相应的元素以及元素所包含的数据了。例如，输出卡片目录中所有的作者名：

```
$reader = new XMLReader();
$reader->open('card-catalog.xml');

/* 循环遍历整个文档 */
while ($reader->read()) {
    /* 如果当前是名为'author'的元素 */
    if ($reader->nodeType == XMLREADER::ELEMENT && $reader->localName == 'author') {
        /* 移动到文本节点并输出文本 */
        $reader->read();
        print $reader->value . "\n";
    }
}
```

```
Sklar
Trachtenberg
Christiansen
Torkington
```

当我们到了<author>元素后，再调用$reader->read()就可以前进到其中的文本点上了。而到了文本节点，就可以通过$reader->value来找到其中的作者名字。

XMLReader::value属性为我们提供了访问节点值的方法。而这个方法只适用于节点是有意义的概念的情况下，例如，文本节点或者CDATA节点。对于所有其他的情况，如元素节点，这个属性值都会被设为空字符串。

表12-2中包含了XMLReader对象属性的完整列表，也包括值。

表12-2：XMLReader节点的类型值

名称	类型	说明
attributeCount	int	节点的属性数
baseURI	string	节点的Base URI
depth	int	节点的树结构深度，从0开始计算
hasAttributes	bool	节点是否包含属性
hasValue	bool	节点是否包含文本值
isDefault	bool	属性值是否来自默认的 DTD
isEmptyElement	bool	节点是否是一个空的元素标签
localName	string	节点的本地（Local）名
name	string	节点的资格（Qualified）名
namespaceURI	string	与节点相关的名称空间的URI
nodeType	int	节点的节点类型
prefix	string	与节点相关的名称空间前缀

表12-2：XMLReader节点的类型值（续）

名称	类型	说明
value	string	节点的文本值
xmlLang	string	节点的xml:lang作用域

XMLReader的功能里面还剩下一项主要的特性：属性。XMLReader拥有一组访问所在元素节点的属性数据的特殊方法，包括：moveToAttribute()、moveToFirstAttribute()、和moveToNextAttribute()。

其中，moveToAttribute()可以让我们来指定要访问的属性名。例如，下面的代码输出卡片目录XML文档中的ISBN号码：

```
<?php
$reader = new XMLReader();
$reader->XML($catalog);

/* 循环遍历整个文档 */
while ($reader->read()) {
    /* 如果当前是名为'author'的元素 */
    if ($reader->nodeType == XMLREADER::ELEMENT && $reader->localName == 'book') {
        $reader->moveToAttribute('isbn');
        print $reader->value . "\n";
    }
}
?>
```

当找到<book>元素时，调用 moveToAttribute('isbn')可以前进到其isbn属性上，进而就可以读取属性的值并输出。

在本技巧的例子当中，我们输出了所有图书的相关信息。不过，要修改上面的例子只取得某本特殊图书的数据也很简单。例如，下面的代码就组合了上面例子中的代码段，并以一种有效的方式输出了有关*Perl Cookbook*的所有数据：

```
<?php
$reader = new XMLReader();
$reader->XML($catalog);

// Perl Cookbook的ISBN是0596003137
// 使用数组来简化添加额外ISBN号码的工作
$isbns = array('0596003137' => true);

/* 循环遍历文档找到第一个<book> */
while ($reader->read()) {
    /* 如果当前是名为'book'的元素 */
    if ($reader->nodeType == XMLREADER::ELEMENT &&
        $reader->localName == 'book') {
        break;
```

```
    }
}

/* 循环遍历<book>查找正确的ISBN */
do {
    if ($reader->moveToAttribute('isbn') &&
        isset($isbns [$reader->value] )) {
        while ($reader->read()) {
            switch ($reader->nodeType) {
            case XMLREADER::ELEMENT:
                print $reader->localName . ": ";
                break;
            case XMLREADER::TEXT:
                print $reader->value . "\n";
                break;
            case XMLREADER::END_ELEMENT;
                if ($reader->localName == 'book') {
                    break 2;
                }
            }
        }
    }
} while ($reader->next());
?>
```

```
title: Perl Cookbook
author: Christiansen
author: Torkington
subject: Perl
```

第一个while()顺序地迭代每个元素直到找到第一个<book>元素为止。

在找到了正确的对象后，就可以中止第一个循环并继续检查ISBN号码。这个过程是通过do... while()循环来处理的。第二个循环中使用了$reader->next()在<book>列表中向下移动。在此，我们不能使用常规的while()语句，否则就会漏掉第一个<book>元素。同样地，这个例子也是说明应该在什么时候使用$reader->next()而不是$reader->read()的最好例子。

如果ISBN匹配了$isbns中的一个值，我们希望进一步处理当前<book>中的数据。这仍然是通过一个while语句和一个switch()语句来实现的。

switch()需要处理三种不同的情况：打开的元素、元素文本和关闭的元素。如果到了一个打开的元素处，那么输出该元素的名和一个冒号。如果到了文本元素处，则输出原本的文本数据。如果到了关闭的元素处，需要检查一下关闭的是否是<book>元素。如果是，则说明已经到了与那本特殊的图书相关数据的结尾处了，此时就需要返回do... while()循环中。这是通过break 2语句来实现的 —— 即退出两级循环，而不是通常的一级。

参见

技巧12.3中有关解析简单XML文档的内容。技巧12.4中有关解析复杂XML文档的讨论。XMLReader的文档（*http://www.php.net/xmlreader*）。基础的libxml2 C库中的XMLReader函数的更多信息（*http://xmlsoft.org/xmlreader.html*）。

12.6 用XPath来提取信息

问题

你想对XML数据进行非常老到的查询，而不是一个节点一个节点地解析文档。

方案

使用XPath。

可以在SimpleXML中使用XPath：

```
<?php
$s = simplexml_load_file('address-book.xml');
$emails = $s->xpath('/address-book/person/email');

foreach ($emails as $email) {
    // 对$email进行某些操作
}
?>
```

也可以在DOM扩展中使用XPath：

```
<?php
$dom = new DOMDocument;
$dom->load('address-book.xml');
$xpath = new DOMXPath($dom);
$email = $xpath->query('/address-book/person/email');

foreach ($emails as $email) {
    // 对$email进行某些操作
}
?>
```

讨论

有时候，除了最简单的文档，很少能够轻易地访问到一个元素中的数据。一方面因为你的XML文件会日益复杂，而另一方面你的解析需求也会不断增长，而此时使用XPath就会比在foreach内部过滤数据更为简单。

PHP中有一个其构造器接受DOM对象作为参数的XPath类。通过该类可以搜索对象并取得相应的DOM节点。SimpleXML也支持XPath，而且由于整合到SimpleXML中，所以使用起来也更容易。

DOM支持XPath查询，但是并不是直接在DOM对象自身来执行查询。而是需要创建一个OMXpath对象，如例12-9所示。

例12-9：使用XPath和DOM

```
$dom = new DOMDocument;
$dom->load('address-book.xml');
$xpath = new DOMXPath($dom);
$email = $xpath->query('/address-book/person/email');
```

技巧化DOMXPath时需要给构造函数传递一个DOMDocument对象。要执行XPath查询，则需要在调用query()方法时向其传递一个查询文本作为参数。该方法会返回一个包含匹配节点的、可枚举的DOM节点列表（见例12-10）。

例12-10：通过DOM使用XPath的基本操作

```
$dom = new DOMDocument;
$dom->load('address-book.xml');
$xpath = new DOMXPath($dom);
$emails = $xpath->query('/address-book/person/email');

foreach ($emails as $e) {
    $email = $e->firstChild->nodeValue;
    // 对$email进行某些操作
}
```

在创建了新的DOMXPath对象之后，使用DOMXPath::query()方法来查询该对象，而传递的第一个参数则是XPath查询字符串（即上面例子中的/people/person/email）。这个方法返回的是一个匹配的DOM节点的列表。

在默认情况下，DOMXPath::query()会对整个XML文档进行操作。如果要对DOM树的一部分进行搜索，就需要将该部分作为最后一个参数传递给query()。例如，要搜索地址簿中所有人的名和姓，就需要取得所有people节点并单独地查询每一个节点，如例12-11所示。

例12-11：通过DOM来使用XPath的一个更复杂的例子

```
$dom = new DOMDocument;
$dom->load('address-book.xml');
$xpath = new DOMXPath($dom);
$person = $xpath->query('/address-book/person');

foreach ($person as $p) {
    $fn = $xpath->query('firstname', $p);
    $firstname = $fn->item(0)->firstChild->nodeValue;
```

```
    $ln = $xpath->query('lastname', $p);
    $lastname = $ln->item(0)->firstChild->nodeValue;

    print "$firstname $lastname\n";
}

David Sklar
Adam Trachtenberg
```

在foreach内部，调用DOMXPath::query()取得firstname和lastname节点。此时，除了XPath查询字符串，我们还给该方法传递了$p作为参数。这样就可以在该局部节点中进行搜索了。

与DOM相对应，所有SimpleXML对象都有一个整合的xpath()方法。而调用该方法可以实现通过XPath对当前对象的查询并返回一个包含匹配节点的SimpleXML对象。也就是说，不用通过另外技巧化一个对象就可以使用XPath。这个方法唯一的参数就是XPath查询字符串 。

例12-12在示例的地址簿中搜索所有匹配的电子邮件地址。

例12-12：使用XPath和SimpleXML的一个简单例子

```
$s = simplexml_load_file('address-book.xml');
$emails = $s->xpath('/address-book/person/email');

foreach ($emails as $email) {
    // 对$email进行某些操作
}
```

由于不需要查找firstNode和取得nodeValue的操作，所以代码更短。

SimpleXML也能处理更复杂的情况。因为xpath()返回的是SimpleXML对象，所以可以直接进行查询，如例12-13所示。

例12-13：使用XPath和SimpleXML的一个更复杂的例子

```
$s = simplexml_load_file('address-book.xml');
$people = $s->xpath('/address-book/person');

foreach($people as $p) {
    list($firstname) = $p->xpath('firstname');
    list($lastname) = $p->xpath('lastname');

    print "$firstname $lastname\n";
}

David Sklar
Adam Trachtenberg
```

因为foreach内部的XPath查询只返回包含一个元素的数组，所以用list取出那个元素。

参见

DOM XPath的文档（*http://www.php.net/function.dom-domxpath-construct.php*）。官方的XPath规范（*http://www.w3.org/TR/xpath*）以及XML in a Nutshell一书中介绍XPath的一章（*http://www.oreilly.com/catalog/xmlnut/chapter/ch09.html*）。

12.7 通过XSLT转换XML

问题

你有一份XML文档和一个XSL样式表。你想要使用XSLT转换这份文档并取得结果。这样你就可以通过把样式表应用数据上面并为不同的媒体创建不同的内容版本。

方案

使用XSLT扩展。

```
// 加载 XSL 模板
$xsl = newDOMDocument;
$xsl->load('stylesheet.xsl');

// 创建一个新 XSLTProcessor对象
$xslt = new XSLTProcessor();
// 加载样式表
$xslt->importStylesheet($xsl);

// 加载XML文档
$xml = new DOMDocument;
$xml->load('data.xml');

// 转换成字符串
$results = $xslt->transformToXML($xml);

// 转换为一个文件
$results = $xslt->transformToURI($xml, 'results.txt');

// 转换成DOM对象
$results = $xslt->transformToDoc($xml);
```

转换后的文本保存在`$results`中。

讨论

XML文档描述的是数据的内容，但是其中不会包含任何与应该怎样显示有关的信息。但

是，当XML文档与有XSL（eXtensible Stylesheet Language，可扩展样式表语言）描述的样式表联系到一块时，其中的内容就会按照特定的视觉规则来显示。

用于联系XML和XSL的是XSLT（eXtensible Stylesheet Language Transformations，可扩展样式表语言转换）。这些转换把样式表中列举的一系列规则应用到XML数据上。这样，就和PHP解析脚本代码并将其与用户的输入组合起来创建动态页面一样，XSLT程序会使用XSL和XML来输出一个包含更多XML,HTML或其他可描述格式内容的新页面。

在几个可用的XSLT程序中，每个都有不同的特性和局限。PHP 5只支持libxslt处理机。这个处理机与PHP 4中所用的处理机是不同的。

在PHP 5中使用XSLT涉及两个主要的步骤：准备XSLT对象和触发对每个XML文件的实际转换。

一开始，先用一个DOM对象加载样式表文件。然后技巧化一个新的`XSLTProcessor`对象，再通过把新创建的DOM对象传递给`importStylesheet()`方法来导入XSLT文档，如例12-14所示。

例12-14：配置XSLT处理器

```
// 加载 XSL 模板
$xsl = newDOMDocument;
$xsl->load('stylesheet.xsl');

// 创建一个新 XSLTProcessor对象
$xslt = new XSLTProcessor();
// 加载样式表
$xslt->importStylesheet($xsl);
```

现在，转换器已经创建好并开始运行了。你可以把任何DOM对象转换为以下三种形式：字符串、文件或另一个DOM对象，如例12-15所示。

例12-15：转换XML数据

```
// 加载XML文档
$xml = new DOMDocument;
$xml->load('data.xml');

// 转换成字符串
$results = $xslt->transformToXML($xml);

// 转换为一个文件
$results = $xslt->transformToURI($xml, 'results.txt');

// 转换成DOM对象
$results = $xslt->transformToDoc($xml);
```

在调用transformToXML()或transformToDoc()方法时，该扩展会返回结果字符串或者结果对象。不同的是，调用transformToURI()返回的是写入到文件中的字节数，而不是实际的文档。

当这些方法调用失败时都会返回false，所以若要正确地检查调用失败，需要这样：

```
if (false === ($results = $xslt->transformToXML($xml))) {
    // 发生了错误
}
```

使用===可以防止返回的0值与实际的错误混淆。

参见

XSL函数功能的文档（*http://www.php.net/xsl*）。Doug Tidwell的书《XSLT》（O'Reilly）

12.8 在PHP中设置XSLT参数

问题

你想在PHP中设置XSLT样式表的参数。

方案

使用XSLTProcessor::setParameter()方法：

```
// 也可能来自 $_GET ['city'] ;
$city = 'San Francisco';

$dom  = new DOMDocument
$dom->load('address-book.xml');
$xsl  = new DOMDocument
$xsl->load('stylesheet.xsl');

$xslt = new XSLTProcessor();
$xslt->importStylesheet($xsl);
$xslt->setParameter(NULL, 'city', $city);
print $xslt->transformToXML($dom);
```

以上代码把PHP中变量$city的值设置为XSLT中city的参数。

讨论

我们可以通过setParmameter()方法把PHP中的数据传递给XSLT样式表。这样我们就可以实现诸如基于用户输入来筛选样式表中数据的功能。

例如，例12-16中的程序可以根据所在城市来找到人。

例12-16：在PHP中设置XSLT的参数

```
// 也可能来自 $_GET ['city'] ;
$city = 'San Francisco';

$dom  = new DOMDocument
$dom->load('address-book.xml');
$xsl  = new DOMDocument
$xsl->load('stylesheet.xsl');

$xslt = new XSLTProcessor();
$xslt->importStylesheet($xsl);
$xslt->setParameter(NULL, 'city', $city);
print $xslt->transformToXML($dom);
```

以上程序使用了下面的样式表：

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="/address-book/person">
  <xsl:if test="city=$city">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

以上程序和样式表组合后会得到以下结果：

```
<?xml version="1.0"?>
<address-book>
    <person id="2">
        <!--Adam Trachtenberg-->
        <firstname>Adam</firstname>
        <lastname>Trachtenberg</lastname>
        <city>San Francisco</city>
```

```
        <state>CA</state>
        <email>amt@php.net</email>
    </person>
</address-book>
```

除了调用$xslt->setParameter(NULL,'city',$city)之外，PHP脚本所做的是一个标准的XSLT转换。在调用setParameter()方法时所传递的第一个参数是参数的名称空间，第二个是参数的名字，而第三个是参数的值。

在这个例子中，保存在PHP变量$city中的值 —— 即San Francisco —— 被赋予XSLT的参数city，而该参数不处在任何名称空间中。这就相当于在XSLT文件中放置了下面这行代码：

```
<xsl:param name="city">San Francisco</xsl:param>
```

我们通常会在一个样式表中像访问PHP的变量一样访问一个参数（在变量名前添加一个美元符号$），这个样式表的例子创建了一个匹配/address-book/person节点的模板。

在这个模板中，我们测试是否city=$city，换句话说，当前节点的city子节点是否等于city参数的值？如果有一个匹配的值，其子节点就会被拷走；否则，该记录就被删除。

在这个例子中，city被设置为San Francisco，所以David的记录被删掉而Adam的记录被保留下来。

参见

XSLTProcessor::setParameter方法的文档（*http://www.php.net/manual/function.xsl-xsltprocessor-set-parameter.php*）。Doug Tidwell的书《XSLT》（O Reilly）

12.9 在XSLT样式表中调用PHP函数

问题

你想在一个XSLT样式表内部调用PHP函数。

方案

调用XSLTProcessor::registerPHPFunctions()方法来实现这个功能：

```
$xslt = new XSLTProcessor();
$xslt->registerPHPFunctions();
```

然后在样式文件中使用function()或functionString()函数：

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:php="http://php.net/xsl"
    xsl:extension-element-prefixes="php">

<xsl:template match="/">
    <xsl:value-of select="php:function('strftime', '%c')" />
</xsl:template>

</xsl:stylesheet>
```

讨论

当需要PHP向XSLT传递数据时使用XSLT参数非常有效。然而，如果你的需求正好相反它就不那么有用了。我们不能在转换过程中通过参数来从样式表中提取信息。在理想的情况下，可以在样式表中调用PHP函数并把信息传递给PHP。

幸运的是，有一个方法实现了这个功能：registerPHPFunctions()。可以像下面这样启用这一功能：

```
$xslt = new XSLTProcessor();
$xslt->registerPHPFunctions();
```

这样，你就可以在样式表文件中调用任何PHP函数了。之所以在默认情况下不能使用这个功能，是为了防止在你的样式表由他人使用时出现安全隐患。

无论是内置的还是用户定义的函数都可以调用。在样式表中，你必须定义一个名称空间并调用function()或functionString()方法，如例12-17所示。

例12-17：从XSL样式表中调用PHP函数

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:php="http://php.net/xsl"
    xsl:extension-element-prefixes="php">

<xsl:template match="/">
    <xsl:value-of select="php:function('strftime', '%c')" />
</xsl:template>

</xsl:stylesheet>
```

在这个样式表的顶部，为PHP定义了名称空间：http://php.net/xsl。该例中把名称空间前缀设置为php。同样地，把extension-element-prefixes的值设置为php也就是让XSLT知道这些是函数。

在调用PHP函数时，要使用php:function()这样的格式。其中的第一个参数是函数名，另一个参数是函数的参数。在这里，函数名是strftime，而一个参数是%c。这会让strftime返回当前的日期和时间。

例12-18使用了这个被保存为*stylesheet.xsl*的样式表，来处理只包含一个元素的XML文档。

例12-18：通过XSLT和PHP函数来转换XML

```
$dom  = new DOMDocument;
$dom->loadXML('<blank/>');
$xsl  = new DOMDocument
$xsl->load('stylesheet.xsl');

$xslt = new XSLTProcessor();
$xslt->importStylesheet($xsl);
$xslt->registerPHPFunctions();
print $xslt->transformToXML($dom);
```

```
Mon Jul 22 19:10:21 2004
```

整个过程与标准的XSLT处理过程类似，但其中包含了为激活对PHP函数的支持而对registerPHPFunctions()的调用。

也可以返回DOM对象。例12-19处理了一个XML地址簿文档，把所有的电子邮件地址都打乱并将主机名部分转换为三个点，而对文档的其他部分未进行任何处理。

例12-19：保护电子邮件地址

```
function mangle_email($nodes) {
    return preg_replace('/( [^@\s] +)@( [-a-z0-9] +\.)+ [a-z] {2,}/is',
                        '$1@...',
                        $nodes [0] ->nodeValue);
}

$dom  = new DOMDocument;
$dom->load('address-book.xml');
$xsl  = new DOMDocument
$xsl->load('stylesheet.xsl');

$xslt = new XSLTProcessor();
$xslt->importStylesheet($xsl);
$xslt->registerPhpFunctions();
print $xslt->transformToXML($dom);
```

在你的样式表中，像例12-20那样为/address-book/person/email元素创建一个特殊的模板。

例12-20：针对防垃圾电子邮件地址的XSL样式表

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:php="http://php.net/xsl"
  xsl:extension-element-prefixes="php">

<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>


<xsl:template match="/address-book/person/email">
  <xsl:copy>
    <xsl:value-of select="php:function('mangle_email', node())" />
  </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

第一个模板确保元素没有被修改。而第二个模板则把当前的节点传递给PHP以便打乱电子邮件地址。在第二个模板中向mangle_email()函数传递了当前的节点，该节点用XPath中的node()来表示，而不是用字符串。注意不要把该节点放在引号中，否则所传递的将是文本直接量node()。

在PHP中该节点会变成DOM对象，而且是保存在数组中。在这里的例子中，mangle_email()函数知道只有一个对象，而且是一个DOMText对象，所以电子邮件地址会包含在$nodes0->nodeValue中。

如果你知道只会用到节点的文本部分，可以使用functionString()函数。该函数把节点转换为PHP字符串，这样就可省掉对数组访问和对nodeValue废弃的环节：

```
function mangle_email($email) {
    return preg_replace('/( [^@\s] +)@( [-a-z0-9] +\.)+ [a-z] {2,}/is',
                        '$1@...',
                        $email);
}

// 其他代码同上
```

而为/address-book/person/email元素创建的新样式表模板变成了：

```
<xsl:template match="/address-book/person/email">
  <xsl:copy>
```

```
      <xsl:value-of
        select="php:functionString('mangle_email', node())" />
    </xsl:copy>
  </xsl:template>
```

现在，因为模板调用的是functionString()函数，所以mangle_email()要处理的是$email而不是$nodes0->nodeValue。

虽然function()和functionString()这两个方法难以置信地好用，但使用它们却破坏了XSL作为一门语言中立的转换引擎的基础。当你在XSLT中调用PHP函数时，就不能轻易地将该样式表重用于使用Java、Perl及其他语言开发的项目上，因为这些语言都无法调用PHP。因此，在使用这一功能之前必须权衡便利与可移植性的问题。

参见

XSLTProcessor::registerPHPFunctions()方法的文档（*http://www.php.net/manual/function.xsl- xsltprocessor-register-php-functions.php*）。Doug Tidwell的书《XSLT》（O'Reilly）。

12.10 验证XML文档

问题

你想确保你的XML文档遵守了一个模式（Schema），如XML Schema，RelaxNG或DTD。

方案

使用DOM扩展。

用已有的DOM对象调用DOMDocument::schemaValidate() 或DOMDocument::relaxNGValidate()：

```
$file = 'address-book.xml';
$schema = 'address-book.xsd';
$ab = new DOMDocument
$ab->load($file);

if ($ab->schemaValidate($schema)) {
    print "$file is valid.\n";
} else {
```

```
    print "$file is invalid.\n";
}
```

如果你在XML文档的顶部指定了一个DTD，那么可以调用DOMDocument::validate() 来按照该DTD来进行验证。

对于以字符串形式表示的XML，调用DOMDocument::schemaValidateSource() 或DOMDocument::relaxNGValidateSource()：

```
$xml = '<person><firstname>Adam</firstname></person>';
$schema = 'address-book.xsd';
$ab = new DOMDocument
$ab->&gt;load($file);

if ($ab->&gt;schemaValidateSource($schema)) {
    print "XML is valid.\n";
} else {
    print "XML is invalid.\n";
}
```

讨论

模式（Schema）是一种定义XML文档规范的方式。虽然目标是相同的，但不同的模式会使用不同的语法来描述XML文档的规范。

流行的XML定义模式包括DTD（Document Type Definitions，文档类型定义），XML Schema和RelaxNG。DTD已经使用了很长时间了，但它不是用XML语言编写的，而且还存在其他的问题，所以属于较难使用的一种。XML Schema和RelaxNG是两种更新的模式，它们都尝试着解决DTD存在的一些问题。

PHP 5通过libxml2库来提供对验证的支持。因此，它可以让你依照所有三种模式来验证文件。而在使用XML Schema和RelaxNG模式时它最能体现出灵活性，但它对XML Schema的支持仍然是不够完善的。虽然对于多数基于XML Schema的文档都不会出现问题，但是你可能也会发现libxml2不能处理某些复杂的模式或者说使用了更高级特性的模式。

在PHP中，DOM扩展支持基于DTD，XML Schema和RelaxNG的验证。而SimpleXML则只提供了XML Schema验证。

通过DOM来验证任何文件的过程都是类似的，不论其遵循何各种模式。在验证时，都是在一个DOM对象上调用验证方法（见例12-21）。如果文件验证通过，则返回true；如

果存在错误，返回false并将错误信息写入错误日志中。目前还没有办法“捕获”错误信息。

例12-21：验证XML文档。

```
$file = 'address-book.xml';
$schema = 'address-book.xsd';
$ab = new DOMDocument
$ab->load($file);

if ($ab->schemaValidate($schema)) {
    print "$file is valid.\n";
} else {
    print "$file is invalid.\n";
}
```

如果模式保存于一个字符串中，就需要使用DOMDocument::schemaValidateSource()而不是schemaValidate()。

表12-3中列出全部验证方法。

表12-3：DOM模式验证方法

方法名	模式类型	数据位置
schemaValidate	XML Schema	File
schemaValidateSource	XML Schema	String
relaxNGValidate	RelaxNG	File
relaxNGValidateSource	RelaxNG	String
validate	DTD	N/A

所有验证方法的使用方式都是类似的，所以我们只须根据不同的验证模式来改变前面例子中的方法名就可以了。

XML Schema和RelaxNG都支持文件和字符串形式的文档验证。而对于DOM对象则只能依据XML文档顶部指定的DTD来进行验证。

参见

XML Schema规范（*http://www.w3.org/XML/Schema*）。RelaxNG规范（*http://www.relaxng.org/*）。

12.11 处理内容编码

问题

PHP的XML扩展使用的是UTF-8编码，而你的数据内容使用的则是另外一种编码。

方案

在将其传递给XML扩展之前使用iconv库进行转换：

```
$utf_8 = iconv('ISO-8859-1', 'UTF-8', $iso_8859_1);
```

在处理完成后再将其转换回原来的编码：

```
$iso_8859_1 = iconv('UTF-8', 'ISO-8859-1', $utf_8);
```

讨论

字符编码是PHP 5的一个主要弱点。幸运的是，对Unicode的支持正是PHP 6背后的主要驱动力。因为PHP 6仍在开发之中，在此期间，如果你想通过XML扩展来处理一些特定编码的数据就有可能遇到问题。

为了简单起见，XML扩展全部使用UTF-8字符编码。也就是说，这些扩展只接受和输出UTF-8编码的数据。如果你的数据使用的是ASCII编码，不用担心，因为UTF-8是ASCII的父集。然而，如果你使用的是其他编码，那么迟早会遇到问题。

为了解决这一问题，需要使用iconv扩展来手工地在你的编码格式和UTF-8编码之间进行编码转换。例如，要从ISO-8859-1转换为UTF-8，需要这样：

```
$utf_8 = iconv('ISO-8859-1', 'UTF-8', $iso_8859_1);
```

iconv函数支持两种目标编码的修改参数：//TRANSLIT和//IGNORE。第一个参数告诉iconv当它不能把一个字符复制到目标编码时，应该尝试用一系列近似的字符来代替该字符。第二个参数则会使iconv自动忽略任何不能转换的字符。

例如，$geb中保存着字符串文本Gödel、Escher、Bach。如果直接将其转换为ASCII就会发生错误：

```
echo iconv('UTF-8', 'ASCII', $geb);
PHP Notice:  iconv(): Detected an illegal character in input string...
```

而加上//IGNORE参数就可以完成转换：

```
echo iconv('UTF-8', 'ASCII//IGNORE', $geb);
```

然而，输出的结果并不完整，因为ö不见了：

```
Gdel, Escher, Bach
```

最佳的解决方案是使用//TRANSLIT：

```
echo iconv('UTF-8', 'ASCII//TRANSLIT', $geb);
```

这样就会生成一个更容易接受的字符串：

```
G"odel, Escher, Bach
```

但是，在使用//TRANSLIT参数一定要仔细一些，因为它会增加结果中的字符数。例如，前面一个字符ö变成了两个字符："和o。

参见

技巧19.13中有关使用UTF-8编码文本的更多信息。iconv函数的文档（*http://www.php.net/iconv*）。GNU libiconv网站中的相关内容，其主页是*http://www.gnu.org/software/libiconv/*。

12.12 读取RSS和Atom源

问题

你想取得RSS和Atom源并查看其中的条目。这样你就可以把多个网站中的新闻资源整合到你的应用程序当中。

方案

使用MagpieRSS解析器。下面是一个读取*php.announce*邮件列表的RSS源的例子：

```
<?php
require 'rss_fetch.inc';

$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';

$rss = fetch_rss( $feed );
```

```
print "<ul>\n";
foreach ($rss->items as $item) {
    print '<li><a href="' . $item ['link']  . '">' . $item ['title']  . "</a></li>\
n";
}
print "</ul>\n";
?>
```

讨论

RSS（RDF Site Summary）是一种易用的标题或文章整合的XML格式（注2）。有很多新闻网站，像纽约时报和华盛顿邮报，每当发表新的消息时都会提供即时更新的RSS源。Weblogs也接受了RSS并且将RSS源作为博客的标准部件。PHP网站也为多数PHP邮件列表发布了相应的RSS源。

Atom是一种类似的XML整合格式。它在RSS的基础上扩展了很多概念，包括读写Atom数据的方式。而且它也尝试为整合提供更多比RSS定义明确的语法，因为RSS规范始终未能清晰地列举出它是什么或者什么不能在源中出现。

使用MagpieRSS来取得并解析RSS和Atom源很简单：

```
<?php
$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';

$rss = fetch_rss($feed);
?>
```

这个例子读取了*php.announce*邮件列表中的RSS源。该源随后被`fetch_rss()`解析并保存在`$rss`内部。

虽然这个源是RSS 0.93，但也不需要指定给MagpieRSS。其`fetch_rss()`函数会检测整合的格式，包括Atom，并根据检测的结果对文档进行格式化。

每一条RSS项目都会以`items`属性的形式被保存到一个关联数组中：

```
<?php
print "<ul>\n";

foreach ($rss->items as $item) {
    print '<li><a href="' . $item ['link']  . '">' . $item ['title']  . "</a></li>\
n";
}
```

注2：　RDF指的是Resource Definition Framework，RSS的另外一个含义是Rich Site Summary。

```
print "</ul>\n";
?>
```

这个foreach循环创建了一个包含条目标题的无序列表，每个标题都可以链接到与完整的文章关联的URL，如图12-1所示。除了必需的title和link字段，每个条目也可以包含一个可选的description字段，该字段中包含条目的介绍。

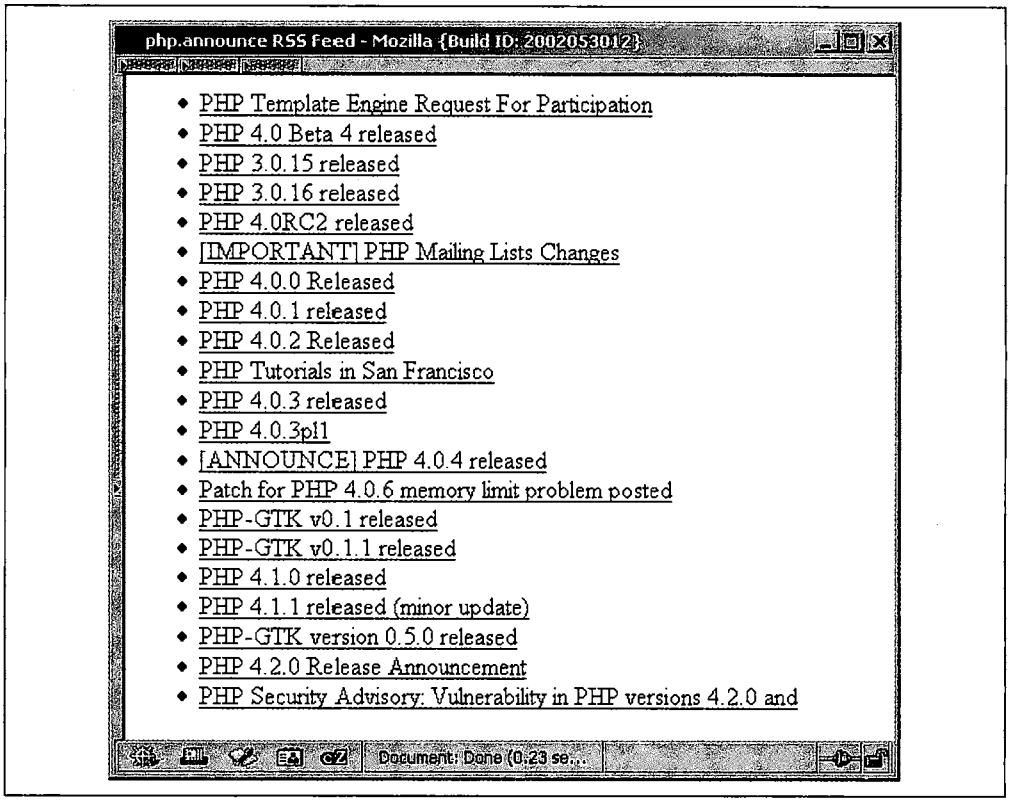

图12-1：php.announce RSS 源

每个channel元素中也会包含一个有关源的条目，如图12-2所示。要取得这些数据，可以访问channel属性：

```
<?php
$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';
$rss = fetch_rss($feed);


print "<ul>\n";

foreach ($rss->channel as $key => $value) {
    print "<li>$key: $value</li>\n";
}
```

```
print "</ul>\n";
?>
```

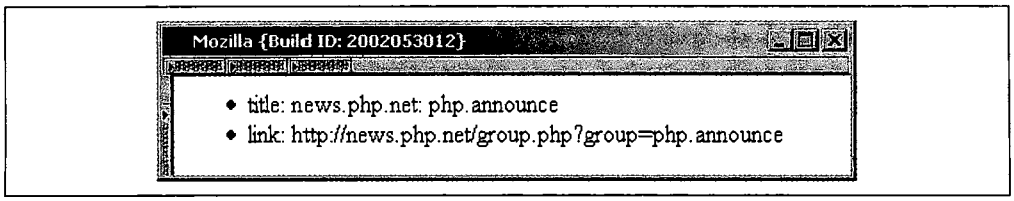

图12-2：php.announce RSS channel 信息

参见

MagpieRSS的相关信息，主页位于*http://magpierss.sourceforge.net/*。RSS的更多资料*http://en.wikipedia.org/wiki/RSS_(protocol)*。

12.13 生成RSS源

问题

你想根据自己的数据生成RSS源以便对你的内容进行整合。

方案

使用下面这个类：

```
<?php
class rss2 extends DOMDocument {
    private $channel;

    public function _ _construct($title, $link, $description) {
        parent::_ _construct();
        $this->formatOutput = true;

        $root = $this->appendChild($this->createElement('rss'));
        $root->setAttribute('version', '2.0');

        $channel= $root->appendChild($this->createElement('channel'));

        $channel->appendChild($this->createElement('title', $title));
        $channel->appendChild($this->createElement('link', $link));
        $channel->appendChild($this->createElement('description', $description));

        $this->channel = $channel;
    }
```

```
    public function addItem($title, $link, $description) {
        $item = $this->createElement('item');
        $item->appendChild($this->createElement('title', $title));
        $item->appendChild($this->createElement('link', $link));
        $item->appendChild($this->createElement('description', $description));

        $this->channel->appendChild($item);
    }
}

$rss = new rss2('Channel Title', 'http://www.example.org',
                'Channel Description');

$rss->addItem('Item 1', 'http://www.example.org/item1',
              'Item 1 Description');
$rss->addItem('Item 2', 'http://www.example.org/item2',
              'Item 2 Description');

print $rss->saveXML();
?>
```

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Channel Title</title>
    <link>http://www.example.org</link>
    <description>Channel Description</description>
    <item>
      <title>Item 1</title>
      <link>http://www.example.org/item1</link>
      <description>Item 1 Description</description>
    </item>
    <item>
      <title>Item 2</title>
      <link>http://www.example.org/item2</link>
      <description>Item 2 Description</description>
    </item>
  </channel>
</rss>
```

讨论

RSS也是XML文件，所以我们可以利用DOM扩展的所有生成特性。本方案中的代码扩展了DOMDocument类，以便通过创建元素并将元素添加到适当的结构中建立一个DOM树。

这个类的构造器中创建了<rss>和<channel>元素。它接受三个参数：channel的title、link和description：

```
    public function __construct($title, $link, $description) {
        parent::__construct();
        $this->formatOutput = true;
```

```
        $root = $this->appendChild($this->createElement('rss'));
        $root->setAttribute('version', '2.0');

        $channel= $root->appendChild($this->createElement('channel'));

        $channel->appendChild($this->createElement('title', $title));
        $channel->appendChild($this->createElement('link', $link));
        $channel->appendChild($this->createElement('description', $description));

        $this->channel = $channel;
    }
```

在这个方法内部，我们使用了parent::_ _construct()以调用真正的DOMDocument::_ _construct()方法。然后，就可以开始构建文档了。

首先，把formatOutput属性设置为true。这样就会在输出中加上缩进和回车以使其容易阅读。

然后，创建文档的根元素 —— rss，并将其version属性设置为2.0,因为这是一个RSS 2.0源。

所有实际的数据都位于rss节点下面的channel元素内部，所以下一步就是生成这个元素，同时也包括其titl、link和description子元素。

所需的数据来自传递给构造器的参数。其中使用了方便的createElement()方法，该方法可以让我们在一次调用中就能用数据来指定元素的名字和文本节点。这是PHP 5对DOM规范的一个扩展。

最后，将创建好的channel元素保存起来以便稍后访问。

对于主要内容的定义，就是使用addItem方法添加新条目：

```
    public function addItem($title, $link, $description) {
        $item = $this->createElement('item');
        $item->appendChild($this->createElement('title', $title));
        $item->appendChild($this->createElement('link', $link));
        $item->appendChild($this->createElement('description', $description));

        $this->channel->appendChild($item);
    }
```

因为item元素中包含的数据与channel相同，所以这个方法中的代码几乎和构造器中是一样的。

虽然titl、link和description是channel的必要元素，但在item中，其实它们都是可选的。一个条目中唯一必须的是包含一个title或一个description元素，仅此而已。

为了简单起见，这里的代码同时要求提供三个元素。同样地，也没有提供追加其他channel或item元素的方法，比如项目中发布的数据或者用于唯一标识条目的GUID等。

仅仅40余行代码后，这个生成基本的RSS 2.0源的类就算大功告成了。其用法如下：

```
$rss = new rss2('Channel Title', 'http://www.example.org',
                'Channel Description');

$rss->addItem('Item 1', 'http://www.example.org/item1',
              'Item 1 Description');
$rss->addItem('Item 2', 'http://www.example.org/item2',
              'Item 2 Description');

print $rss->saveXML();
```

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Channel Title</title>
    <link>http://www.example.org</link>
    <description>Channel Description</description>
    <item>
      <title>Item 1</title>
      <link>http://www.example.org/item1</link>
      <description>Item 1 Description</description>
    </item>
    <item>
      <title>Item 2</title>
      <link>http://www.example.org/item2</link>
      <description>Item 2 Description</description>
    </item>
  </channel>
</rss>
```

创建rss2类的一个新技巧，并向其传递channel的有关数据。然后，再调用其addItem()方法分别向该channel元素中添加新条目。当添加完所有条目后，就可以通过父类的DOMDocument::saveXML()方法将其转换成XML文件了。

12.14 生成Atom源

问题

你想根据自己的数据生成Atom源以便可以对自己的内容进行整合。

方案

使用下面这个类：

```
class atom1 extends DOMDocument {
    private $ns;

    public function _ _construct($title, $href, $name, $id) {
        parent::_ _construct();
        $this->formatOutput = true;

        $this->ns = 'http://www.w3.org/2005/Atom';

        $root = $this->appendChild($this->createElementNS($this->ns, 'feed'));

        $root->appendChild($this->createElementNS($this->ns, 'title', $title));
        $link = $root->appendChild($this->createElementNS($this->ns, 'link'));
        $link->setAttribute('href', $href);
        $root->appendChild($this->createElementNS($this->ns, 'updated',
            date('Y-m-d\\TH:i:sP')));
        $author = $root->appendChild($this->createElementNS($this->ns, 'author'));
        $author->appendChild($this->createElementNS($this->ns, 'name', $name));
        $root->appendChild($this->createElementNS($this->ns, 'id', $id));
    }

    public function addEntry($title, $link, $summary) {
        $entry = $this->createElementNS($this->ns, 'entry');
        $entry->appendChild($this->createElementNS($this->ns, 'title', $title));
        $entry->appendChild($this->createElementNS($this->ns, 'link', $link));

        $id = uniqid('http://example.org/atom/entry/ids/');
        $entry->appendChild($this->createElementNS($this->ns, 'id', $id));

        $entry->appendChild($this->createElementNS($this->ns, 'updated',
            date(DATE_ATOM)));
        $entry->appendChild($this->createElementNS($this->ns, 'summary',
    $summary));

        $this->documentElement->appendChild($entry);
    }
}

$atom = new atom1('Channel Title', 'http://www.example.org',
                'John Quincy Atom', 'http://example.org/atom/feed/ids/1');

$atom->addEntry('Item 1', 'http://www.example.org/item1',
              'Item 1 Description', 'http://example.org/atom/entry/ids/1');

$atom->addEntry('Item 2', 'http://www.example.org/item2',
              'Item 2 Description', 'http://example.org/atom/entry/ids/2');

print $atom->saveXML();

<?xml version="1.0"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Channel Title</title>
  <link href="http://www.example.org"/>
  <updated>2006-10-23T22:33:59-07:00</updated>
  <author>
```

```
    <name>John Quincy Atom</name>
  </author>
  <id>http://example.org/atom/feed/ids/1</id>
  <entry>
    <title>Item 1</title>
    <link>http://www.example.org/item1</link>
    <id>http://example.org/atom/entry/ids/1</id>
    <updated>2006-10-23T20:23:32-07:00</updated>
    <summary>Item 1 Description</summary>
  </entry>
  <entry>
    <title>Item 2</title>
    <link>http://www.example.org/item2</link>
    <id>http://example.org/atom/entry/ids/2</id>
    <updated>2006-10-23T21:53:44-07:00</updated>
    <summary>Item 2 Description</summary>
  </entry>
</feed>
```

讨论

这个atom1类与技巧12.13中的rss2类的结构相似。可以参考该技巧的讨论部分中关于完整的代码结构和DOM扩展行为的介绍。本技巧中包含了RSS和Atom之间的不同及atom1类如何修正以处理这些差异的内容。

Atom规范比RSS更复杂一些。它要求把元素放在一个名称空间中，并强制为源和每个独立的条目生成唯一的标识符，并要求包含每个条目的最后更新时间。

而且，虽然它生成的结果与RSS类似，但使用的却是另外一套概念。它的根元素变成了feed，而条目则包含在entry元素中。它不需要提供源的描述（description），但却要求提供作者（author）的信息。而且在条目的内部，description也被summary取代了。

最后，Atom也没有channel的概念。所有源数据和条目都直接位于文档根元素之下。

下面是经过修正后的构造器：

```
public function _ _construct($title, $href, $name, $id) {
    parent::_ _construct();
    $this->formatOutput = true;

    $this->ns = 'http://www.w3.org/2005/Atom';

    $root = $this->appendChild($this->createElementNS($this->ns, 'feed'));

    $root->appendChild(
        $this->createElementNS($this->ns, 'title', $title));
    $link = $root->appendChild(
        $this->createElementNS($this->ns, 'link'));
    $link->setAttribute('href', $href);
```

```
        $root->appendChild($this->createElementNS(
            $this->ns, 'updated', date(DATE_ATOM)));
        $author = $root->appendChild(
            $this->createElementNS($this->ns, 'author'));
        $author->appendChild(
            $this->createElementNS($this->ns, 'name', $name));
        $root->appendChild(
            $this->createElementNS($this->ns, 'id', $id'));
    }
```

所有Atom元素都寄身于http://www.w3.org/2005/Atom XML名称空间之中。因此，所有atom1的方法都使用了DOMDocument::createElementNS()方法，该方法是DOMDocument::createEle-ment()的名称空间版。Atom的名称空间被保存在atom1::ns中，访问起来很方便。

当前的构造器需要四个参数：标题（title）、链接（link）、作者姓名（author）和源（feed）ID。其中title和id的定义与在RSS中是类似的。但是，link在这里则成为设置link元素的href属性了，而且name是作为author元素的子元素而存在的。

另外，还有一个新增的元素，这个元素被设置为最后更新时间。在这个例子当中，使用了当前时间并用PHP内置的DATE_ATOM常量格式规范进行了格式化。该常量只在PHP 5.1.1之后才是有效的，如果你使用的PHP的早期版本，则需要换成字符串Y-m-d\\TH:i:sP。

rss2类中的addItem()方法被重命名为addEntry()以便同Atom规范保持一致：

```
    public function addEntry($title, $link, $summary, $id) {
        $entry = $this->createElementNS($this->ns, 'entry');
        $entry->appendChild(
            $this->createElementNS($this->ns, 'title', $title));
        $entry->appendChild(
            $this->createElementNS($this->ns, 'link', $link));
        $entry->appendChild(
            $this->createElementNS($this->ns, 'id', $id));
        $entry->appendChild(
            $this->createElementNS($this->ns, 'updated', date(DATE_ATOM)));
        $entry->appendChild(
            $this->createElementNS($this->ns, 'summary', $summary));

        $this->documentElement->appendChild($entry);
    }
```

但这个方法中除了额外多了几个id和updated之类的新元素外其行为与addItem()非常相似。

把所有修改后的代码付诸实践的过程如下：

```
$atom = new atom1('Channel Title', 'http://www.example.org',
                'John Quincy Atom', 'http://example.org/atom/feed/ids/1');

$atom->addEntry('Item 1', 'http://www.example.org/item1',
              'Item 1 Description', 'http://example.org/atom/entry/ids/1');

$atom->addEntry('Item 2', 'http://www.example.org/item2',
              'Item 2 Description', 'http://example.org/atom/entry/ids/2');

print $atom->saveXML();
```

```
<?xml version="1.0"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Channel Title</title>
  <link href="http://www.example.org"/>
  <updated>2006-10-23T22:33:59-07:00</updated>
  <author>
    <name>John Quincy Atom</name>
  </author>
  <id>http://example.org/atom/feed/ids/1</id>
  <entry>
    <title>Item 1</title>
    <link>http://www.example.org/item1</link>
    <id>http://example.org/atom/entry/ids/1</id>
    <updated>2006-10-23T20:23:32-07:00</updated>
    <summary>Item 1 Description</summary>
  </entry>
  <entry>
    <title>Item 2</title>
    <link>http://www.example.org/item2</link>
    <id>http://example.org/atom/entry/ids/2</id>
    <updated>2006-10-23T21:53:44-07:00</updated>
    <summary>Item 2 Description</summary>
  </entry>
</feed>
```

同rss2类一样，atom1也只是实现了完整规范的一个小子集。虽然也足以生成有效的源了，但是如果你的需求更多的话，那么就必须亲自再扩展这个类。

参见

Atom的相关信息，主页位于*http://www.atomenabled.org/*。维基（Wiki）中的Atom条目*http://www.intertwingly.net/wiki/pie/*。有关Atom的更多资料，要访问*http://en.wikipedia.org/wiki/Atom_(standard)*。

第13章

Web自动化

13.0 概述

大多数时候，PHP都是作为Web服务器的一部分，向浏览器发送内容。即使是在命令行中运行PHP，它都能执行相应的任务并输出一些内容。不过，PHP也可以作为Web客户端来使用，通过它可以检索URL并对相应的内容进行操作。本章中的大部分技巧都涉及到检索URL并处理其结果，还有少数几个与清除URL及JavaScript有关的技巧。

在PHP中有多种方式可以取得URL。选择哪一种方式取决于你是需要简单、控制还是可移植性。本章中所讨论的三个方法包括标准的文件函数、cURL扩展和PEAR中的HTTP_Request类。一般而言，这三个方法完全可以满足我们的需要，而且无论你的服务器是如何配置的或者安装了哪些自定义的扩展，都至少有一个方法是可用的。另外一种取得远程URL的方法包含在pecl_http扩展（*http://www.pecl.php.net/package/pecl_http*）中，虽然这种方法仍在开发之中，但却提供了一些令人期待的特性，通过使用fsockopen()函数打开socket，让你能够在此之上发送一个自己逐段构造的HTTP请求。

使用标准的文件函数如file_get_contents()既简单也很方便。它能够自动跟踪重定向，也就是说如果你用这个函数取得了目录*http://www.example.com/people*而服务器又将其重定向到*http://www.example.com/people/*，你所得到的将是目录索引页面的内容而不是一个告诉你URL已经改变的信息。标准文件函数既可以用于HTTP也可以用于FTP。这种方法的缺点是它需要开启allow_url_fopen配置指令。

cURL扩展是一个强大的request请求“万事通”。它依靠流行的libcurl（*http://curl.haxx.se/*）提供快速的、可配置的机制，来处理各种各样的网络请求。如果这个扩展在你的服务器上是可用的，我们推荐你使用它。

如果allow_url_fopen配置指令关闭而且cURL也不可用，那么PEAR的HTTP_Request模块可以担当重任。与PEAR的其他模块一样，它也是普通的PHP，所以如果你能够在服务器中保存一个PHP文件，就可以使用它。HTTP_Request支持几乎所有请求远程URL的操作，包括修改请求信息的头部和主体，使用任意方法以及取得响应信息的头部。

技巧13.1～13.7中解释了如何实现各种HTTP请求，处理头部、方法、主体和计时。技巧13.8中带你到HTTP请求的幕后查看请求和响应中的头部信息。如果在某个程序中执行的请求没有带给你期望的结果，查看一下头部能够获得错误提示。

当把一个网页中的内容读取到程序中后，可以使用技巧13.9～技巧13.14中介绍的方法来操纵这些页面内容。技巧13.9中演示了如何在页面中以颜色块来标识某个词，而这一技术可以用于突出显示搜索关键字。技巧13.11中提供了一个查找页面中所有链接的函数，这一功能正是Web蜘蛛或链接检测程序的基本组成部分。有关在纯文本和HTML之间进行转换的内容放在了技巧13.12和技巧13.13中来讨论。技巧13.14中则展示了如何从一个网页中删除所有HTML和PHP标签。

技巧13.15和技巧13.16中讨论的是PHP与JavaScript如何协同工作。技巧13.15中演示了用PHP来响应由JavaScript生成的请求，其中必须要留意的是缓存和使用可替代的内容类型。技巧13.16中则提供了使用流行而强大的Dojo工具包将PHP与JavaScript整合到一起的一个完整例子。

最后两个编程技巧使用了技巧13.11中讨论过的链接析取器。技巧13.17中对页面中的链接进行扫描并报告哪个链接仍然有效，哪个链接已经被删除以及哪个链接不再可用。而13.18节中的程序则报告了哪些是最新的链接，它会告诉你某个链接的目标页面的最后更新时间以及该页面是否已经被删除。

13.1 通过Get方法定位URL

问题

你想取得一个URL的内容。例如，你想把一个网页中的一部分内容包含到另一个网页中。

方案

如例13-1所示，把URL传递给file_get_contents()函数。

例13-1：用file_get_contents()取得URL的内容

```
<?php
$page = file_get_contents('http://www.example.com/robots.txt');
?>
```

也可以使用cURL扩展，如例13-2所示。

例13-2：用cURL取得URL的内容

```
<?php
$c = curl_init('http://www.example.com/robots.txt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
?>
```

还可以使用PEAR的HTTP_Request类，如例13-3所示。

例13-3：用HTTP_Request取得URL的内容

```
<?php
require_once 'HTTP/Request.php';
$r = new HTTP_Request('http://www.example.com/robots.txt');
$r->sendRequest();
$page = $r->getResponseBody();
?>
```

讨论

与PHP所有的文件处理函数一样，file_get_contents()函数也是建立在PHP对流进行操作的基础之上的。这意味着该函数不仅能处理本地文件，也可以处理多种网络资源，包括基于HTTP协议的URL。不过，采用此方法存在一个难以两全的问题，即必须要启用allow_fopen_configuration配置指令（一般都会要求如此）。

这样会使得检索远程文档变得极为容易。如例13-4所示，可以运用同样的技术来取得一个远程XML文档。

例13-4：取得远程XML文档

```
<?php
$url = 'http://rss.news.yahoo.com/rss/oddlyenough';
$rss = simplexml_load_file($url);
print '<ul>';
foreach ($rss->channel->item as $item) {
   print '<li><a href="' .
         htmlentities($item->link) .
         '">' .
         htmlentities($item->title) .
         '</a></li>';
}
```

```
print '</ul>';
?>
```


要想取得一个包含查询字符串变量的页面，则需要通过http_build_query()创建一个查询字符串。该函数接受一个键/值对构成的数组作为参数并返回一个经过适当转义处理的字符串。但我们仍然要自己设置URL中触发查询字符串的“?”。例13-5示范了http_build_query()的用法。

例13-5：通过http_build_query()来建立一个查询字符串

```
<?php
$vars = array('page' => 4, 'search' => 'this & that');
$qs = http_build_query($vars);
$url = 'http://www.example.com/search.php?' . $qs;
$page = file_get_contents($url);
?>
```

在要取得一个受保护的页面时，需要把用户名和密码放到URL当中。在例13-6中，用户名是david，而密码是haxOr。

例13-6：取得一个受保护的页面

```
<?php
$url = 'http://david:haxOr@www.example.com/secrets.php';
$page = file_get_contents($url);
?>
```

例13-7显示了如何通过cURL来取得受保护的页面。

例13-7：通过cURL来取得一个受保护的页面

```
<?php
$c = curl_init('http://www.example.com/secrets.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_USERPWD, 'david:haxOr');
$page = curl_exec($c);
curl_close($c);
?>
```

例13-8显示了通过HTTP_Request来取得一个受保护页面的过程。

例13-8：通过HTTP_Request来取得受保护的页面

```
<?php
$r = new HTTP_Request('http://www.example.com/secrets.php');
$r->setBasicAuth('david','haxOr');
$r->sendRequest();
$page = $r->getResponseBody();
?>
```

PHP的流包装（stream wrapper）能够自动跟踪重定向。从PHP 5.0.0开始，

file_get_contents()和fopen()都支持一个设置流环境的参数，该参数可以允许我们指定如何取得流的选项。在PHP 5.1.0及以后版本中，其中的选项之一就是max_redirects——即跟踪重定向的最大次数。例13-9将max_redirects设置为1，即禁用了重定向跟踪。

例13-9：不跟踪重定向

```
<?php
$url = 'http://www.example.com/redirector.php';
// 定义选项
$options = array('max_redirects' => 1 );
// 用选项为http流创建一个执行环境
$context = stream_context_create(array('http' => $options));
// 将选项传递给file_get_contents。其中第二个参数表示是否使用包含路径，在这里不需要使用
print file_get_contents($url, false, $context);
```

这个max_redirects流包装选项真正表示的不是应该跟踪多少次重定向，而是表示在跟踪重定向链接时最多应该生成多少次请求。换句话说，1表示让PHP最多只生成一次请求——不跟踪重定向，而2则表示最多生成两次请求——只跟踪一次重定向（0和1的作用是一样的，都是让PHP只生成一次请求）。

如果重定向链接需要PHP生成的请求数大于max_redirects的值，PHP就给出一个警告。

cURL只有在设置CURLOPT_FOLLOWLOCATION选项时才会跟踪重定向，如例13-10所示。

例13-10：通过cURL来跟踪重定向

```
<?php
$c = curl_init('http://www.example.com/redirector.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_FOLLOWLOCATION, true);
$page = curl_exec($c);
curl_close($c);
?>
```

要设置cURL应该跟踪重定向的最大次数，可以在把CURLOPT_FOLLOWLOCATION设置为true的基础上，再把CURLOPT_MAXREDIRS设置为所允许的最大值。

HTTP_Request不能用于跟踪重定向，但另一个PEAR模块——HTTP_Client则可以。HTTP_Client包装了HTTP_Request并提供了更多的功能。例13-11显示如何通过HTTP_Client来重定向。

例13-11：通过HTTP_Client进行重定向

```
<?php
require_once 'HTTP/Client.php';

// 创建一个客户端
$client = new HTTP_Client();
```

```
// 发出一个GET请求
$client->get($url);
// 取得响应
$response = $client->currentResponse();
// $response是一个包含三个元素的数组：
// code, headers和body
print $response ['body'] ;
?>
```

通过cURL可以对它取得的页面进行其他一些处理。我们在前面的例子中已经看到了，如果设置了CURLOPT_RETURNTRANSFER，则curl_exec()会返回被请求页面的主体内容（body）。如果没有设置URLOPT_RETURNTRANSFER，curl_exec()则会输出响应中的主体内容。

若要把取得的页面写到一个文件中，可以先用fopen()来打开一个文件句柄，然后为该文件句柄设置CURLOPT_FiLETF选项。例13-12使用cURL将一个远程网页复制到了一个本地文件当中。

例13-12：通过cURL把响应中的主体内容写入一个文件

```
<?php
$fh = fopen('local-copy-of-files.html','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/files.html');
curl_setopt($c, CURLOPT_FiLE, $fh);
curl_exec($c);
curl_close($c);
?>
```

如果要把cURL资源以及取得的页面的内容传递给一个函数，可以在那个函数中将CURLOPT_WRI-TEFUNCTION选项设置为回调（可以是一个字符串形式的函数名，也可以是一个包含对象或技巧及其方法的数组）。其中的“写入函数”必须返回所传入的字节数。注意，对于一个比较大的响应来说，由于cURL是按块来处理响应的，所以可能会不止一次地调用该写入函数。例13-13使用一个cURL写入函数把内容保存到了数据库中。

例13-13：通过cURL把页面保存到数据库表中

```
<?php
class PageSaver {
    protected $db;
    protected $page ='';

    public function __construct() {
        $this->db = new PDO('sqlite:./pages.db');
    }

    public function write($curl, $data) {
        $this->page .= $data;
```

```
        return strlen($data);
    }

    public function save($curl) {
        $info = curl_getinfo($curl);
        $st = $this->db->prepare('INSERT INTO pages '.
                                 '(url,page) VALUES (?,?)');
        $st->execute(array($info ['url'] , $this->page));
    }
}

// 创建PageSaver的技巧
$pageSaver = new PageSaver();
// 创建cURL 资源
$c = curl_init('http://www.sklar.com/');
// 设置写入函数
curl_setopt($c, CURLOPT_WRITEFUNCTION, array($pageSaver,'write'));
// 执行请求
curl_exec($c);
// 保存累积的数据
$pageSaver->save($c);
```

参见

技巧13.2中有关使用POST方法取得URL的内容。file_get_contents()函数的文档（*http://www.php.net/file_get_contents*），simplexml_load_file()函数的文档（*http://www.php.net/simplexml_load_file*），stream_context_create()函数的文档（*http://www.php.net/stream_context_create*），curl_init()函数的文档（*http://www.php.net/curl-init*），curl_setopt()函数的文档（*http://www.php.net/curl-setopt*），curl_exec()函数的文档（*http://www.php.net/curl-exec*），curl_getinfo()函数的文档（*http://www.php.net/curl_getinfo*），curl_close()函数的文档（*http://www.php.net/curl-close*）。PEAR的HTTP_Request类的文档（*http://pear.php.net/package/HTTP_Request*）和PEAR的HTTP_Client类的文档（*http://pear.php.net/package/HTTP_Client.*）。

13.2 通过Post方法定位URL

问题

你想用post方法取得一个URL，而不是默认的get方法。比如说，你想提交一个表单。

方案

在使用http流的时候，设置相应的方法和内容流环境选项，如例13-14所示。

例13-14：通过POST方法使用http流

```
<?php
$url = 'http://www.example.com/submit.php';
// 将提交的数据编码为查询字符串
// 名值对
$body = 'monkey=uncle&rhino=aunt';
$options = array('method' => 'POST', 'content' => $body);
// 创建流环境
$context = stream_context_create(array('http' => $options));
// 把环境传递给 file_get_contents()
print file_get_contents($url, false, $context);
?>
```

如果使用cURL扩展，则需要如例13-15所示设置CURLOPT_POST和CURLOPT_POSTFiELDS选项

例13-15：在使用cURL时使用POST方法。

```
<?php
$url = 'http://www.example.com/submit.php';
// 将提交的数据编码为查询字符串
// 名值对
$body = 'monkey=uncle&rhino=aunt';
$c = curl_init($url);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFiELDS, $body);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
?>
```

例13-16显示了在HTTP_Request中设置post方法的过程：把HTTP_REQUEST_METHOD_POST传递给构造器，并对提交数据中的每个名值对调用一次addPostData()方法。

例13-16：在HTTP_Request中使用POST方法

```
<?php
require 'HTTP/Request.php';
$url = 'http://www.example.com/submit.php';
$r = new HTTP_Request($url);
$r->setMethod(HTTP_REQUEST_METHOD_POST);
$r->addPostData('monkey','uncle');
$r->addPostData('rhino','aunt');
$r->sendRequest();
$page = $r->getResponseBody();
?>
```

讨论

用post方法发送一个请求需要专门处理所有的参数。在get请求中，这些参数都包含在

查询字符串中，但在post请求中，它包含在请求体当中。而且，请求中还需要包含一个Content-Length头部信息用于通知服务器请求体中的内容有多大。

虽然本技巧解决方案中的每个例子都有各不相同的指定请求方法和主体内容的机制，但每个例子中使用的方法都会自动添加适当的Content-Length头部信息。

如果你使用流环境来发送post请求，不要忘了把方法选项设置为post —— 而且要注意用大写。

在查询字符串非常长（比如超过了200个字符）的情况下，通过post而不是get来取得URL特别有用。RFC 2616中的HTTP 1.1规范并未规定URL的最大长度是多少，所以在不同的Web服务器和代理服务器中可能会有所差别。一般来说，当你通过get方法取得URL时出现了意外结果或者产生了状态码414（“Reqeuest-URI过长”）错误，就应该转而使用post请求了。

参见

技巧13.1中通过get方法取得URL的内容。curl_setopt()函数的文档（*http://www.php.net/curl-setopt*）和有关流选项的文档（*http://www.php.net/wrappers.http*）。PEAR中的HTTP_Request类的文档（*http://pear.php.net/package/HTTP_Request*）以及RFC 2616规范（*http://www.w3.org/Protocols/rfc2616/rfc2616.html*）。

13.3 通过Cookie定位URL

问题

你想要取得一个必须在请求中同时发送一个cookie才能取得的页面。

方案

使用cURL中的CURLOPT_COOKIE选项，如例13-17所示。

例13-17：通过cURL发送cookie

```
<?php
$c = curl_init('http://www.example.com/needs-cookies.php');
curl_setopt($c, CURLOPT_COOKIE, 'user=ellen; activity=swimming');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
?>
```

对于HTTP_Request，可以使用addHeader()方法在请求中添加Cookie头部信息，如例13-18所示。

例13-18：通过HTTP_Request发送cookie

```
<?php
require 'HTTP/Request.php';
$r = new HTTP_Request('http://www.example.com/needs-cookies.php');
$r->addHeader('Cookie','user=ellen; activity=swimming');
$r->sendRequest();
$page = $r->getResponseBody();
?>
```

讨论

cookie是通过Cookie请求头部信息发送到服务器的。在cURL扩展中有一个专门针对cookie的选项，而通过HTTP_Request则必须和添加其他请求头部信息一样添加Cookie头部信息。要发送多个cookie值，值与值之间需要使用分号隔开。该解决方案中的例子中发送了两个cookie：一个名为user值为ellen，另一个名为activity值为swimming。

要请求一个设置cookie的页面，然后生成包含那些新设置cookie的请求，可以使用cURL的“cookie jar”功能。在最初的请求中，把CURLOPT_COOKIEJAR设置为要保存cookie的文件名。在后续的请求中，把CURLOPT_COOKIEFILE设置为同一个文件名，cURL就会从该文件中读取cookie的值并将其附加到请求中发送出去。这一功能经常用于第一次请求登录到一个站点，设置了session或身份认证cookie，而后来的请求需要包含那些cookie才能保证有效的情况。

例13-19：通过cURL的cookie jar来实现cookie跟踪

```
<?php
// 保存cookie的临时文件
$cookie_jar = tempnam('/tmp','cookie');

// 登录
$c = curl_init('https://bank.example.com/login.php?user=donald&password=b1gmoney$');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_COOKIEJAR, $cookie_jar);
$page = curl_exec($c);
curl_close($c);

// 取得账户余额
$c = curl_init('http://bank.example.com/balance.php?account=checking');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_COOKIEFILE, $cookie_jar);
$page = curl_exec($c);
curl_close($c);

// 进行累计
```

```
$c = curl_init('http://bank.example.com/deposit.php');
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFIELDS, 'account=checking&amount=122.44');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_COOKIEFILE, $cookie_jar);
$page = curl_exec($c);
curl_close($c);

// 删除 cookie jar
unlink($cookie_jar) or die("Can't unlink $cookie_jar");
?>
```

在保存cookie jar时要格外注意。因为需要把它保存到web服务器有写入权限的地方，如果其他用户能够读取这个文件，则可能会窃取保存在cookie中的认证证书信息。

HTTP_Client也提供了一个类似的cookie跟踪功能。只需启用该功能，就可以在由同一个HTTP_Client对象生成的多个请求之间自动地保留cookie信息。

参见

curl_setopt()函数的文档（*http://www.php.net/curl-setopt*），PEAR的HTTP_Request类的文档（*http://pear.php.net/package/HTTP_Request*），PEAR的HTTP_Client类的文档（*http://pear.php.net/package/HTTP_Client*）。文章“Persistent Client State - HTTP Cookies”（*http://wp.netscape.com/newsref/std/cookie_spec.html*）和“HTTP Cookies: Standards, Privacy, and Politics”（*http://arxiv.org/abs/cs.SE/0105018*），作者 David M. Kristol。

13.4 通过任意头部信息定位URL

问题

你想要取得一个必须在请求中同时发送一个特定的头部信息才能取得的页面。

方案

如果使用http流，则可以像例13-20中所示的那样设置头部流环境选项。头部信息的值必须是字符串。多个头部信息要用回车加换行（双引号中的\r\n）符来分开。

例13-20：通过http流来发送头部信息

```
<?php
$url = 'http://www.example.com/special-header.php';
```

```
$header = "X-Factor: 12\r\nMy-Header: Bob";
$options = array('header' => $header);
// 创建流环境
$context = stream_context_create(array('http' => $options));
// 把环境传递给file_get_contents()
print file_get_contents($url, false, $context);
?>
```

如果使用cURL，则需要把CURLOPT_HTTPHEADER选项设置为一个包含要发送的头部信息的数组，如例13-21所示。

例13-21：通过cURL来发送头部信息

```
<?php
$c = curl_init('http://www.example.com/special-header.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_HTTPHEADER, array('X-Factor: 12', 'My-Header: Bob'));
$page = curl_exec($c);
curl_close($c);
?>
```

如果使用HTTP_Request，可以像例13-22那样使用addHeader()方法。

例13-22：通过HTTP_Request发送头部信息

```
<?php
require 'HTTP/Request.php';

$r = new HTTP_Request('http://www.example.com/special-header.php');
$r->addHeader('X-Factor',12);
$r->addHeader('My-Header','Bob');
$r->sendRequest();
$page = $r->getResponseBody();
?>
```

讨论

在cURL中有专门针对Referer和User-Agent请求头部信息的选项 —— CURLOPT_REFERER和CURLOPT_USERAGENT。例13-23中同时用到了这两个选项。

例13-23：通过cURL来设置Referer和User-Agent头部信息

```
<?php
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_REFERER, 'http://www.example.com/form.php');
curl_setopt($c, CURLOPT_USERAGENT, 'cURL via PHP');
$page = curl_exec($c);
curl_close($c);
?>
```

参见

http流包装的文档（*http://www.php.net/wrappers.http*），curl_setopt()函数的文档（*http://www.php.net/curl-setopt*）和PEAR的HTTP_Request类的文档（*http://pear.php.net/package/HTTP_Request*）。有关“Referer”中只有一个“r”这个现象背后的远大而具有颠覆性目标的说明，请参见邮件列表中的信息*http:///lists.w3.org/Archives/Public/ietf-http-wg-old/1996MayAug/0734.html*。

13.5 通过任意方法定位URL

问题

你想通过更多get或post以外的方法，如put或delete来取得URL。

方案

同使用post方法一样，在使用http流时要设置方法和内容流环境选项，如例13-24所示。

例13-24：通过http流使用put方法

```
<?php
$url = 'http://www.example.com/put.php';
// The request body, in arbitrary format
$body = '<menu>
 <dish type="appetizer">Chicken Soup</dish>
 <dish type="main course">Fried Monkey Brains</dish>
</menu>';
$options = array('method' => 'PUT', 'content' => $body);
// 创建流环境
$context = stream_context_create(array('http' => $options));
// 把环境传递给 file_get_contents()
print file_get_contents($url, false, $context);
?>
```

如果使用cURL，则需要把CURLOPT_CUSTOMREQUEST选项设置为相应的方法名。如果要包含一个请求体，还需要把CURLOPT_POSTFiELDS选项设置为表示请求体的变量，如例13-25所示。

例13-25：通过cURL来使用put方法

```
<?php
// 请求体，任意格式
$body = '<menu>
 <dish type="appetizer">Chicken Soup</dish>
 <dish type="main course">Fried Monkey Brains</dish>
</menu>';
```

```
$c = curl_init($url);
curl_setopt($c, CURLOPT_CUSTOMREQUEST, 'PUT');
curl_setopt($c, CURLOPT_POSTFiELDS, $body);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
?>
```


例13-26中显示了如何通过HTTP_Request来使用put的方法。即把HTTP_REQUEST_METHOD_PUT传递给构造器，并以请求体的内容作为参数调用setBody()方法。

例13-26：通过HTTP_Request来使用put方法

```
<?php
require 'HTTP/Request.php';
$url = 'http://www.example.com/put.php';
$body = '<menu>
 <dish type="appetizer">Chicken Soup</dish>
 <dish type="main course">Fried Monkey Brains</dish>
</menu>';
$r = new HTTP_Request($url);
$r->setMethod(HTTP_REQUEST_METHOD_PUT);
$r->setBody($body);
$page = $r->getResponseBody();
?>
```

讨论

随着REST式的Web服务API日益流行，生成HTTP请求也开始使用请求方法群体中轻量级的成员，比如put和delete方法。

其中put方法通常用于上传某个特定文件的内容。cURL中有三个专门的选项用以实现这一点：CURLOPT_PUT，CURLOPT_INFiLE和CURLOPT_INFiLESIZE。如果想通过cURL和put方法上传文件，可以把CURLOPT_PUT选项设置为true，把CURLOPT_INFiLE设置为一个打开的上传文件的句柄，把CURLOPT_INFiLESIZE设置为该文件的大小。例13-27示范了如何使用put方法上传文件。

例13-27：通过cURL和put来上传文件

```
<?php
$url = 'http://www.example.com/upload.php';
$filename = '/usr/local/data/pictures/piggy.jpg';
$fp = fopen($filename,'r');
$c = curl_init($url);
curl_setopt($c, CURLOPT_PUT, true);
curl_setopt($c, CURLOPT_INFiLE, $fp);
curl_setopt($c, CURLOPT_INFiLESIZE, filesize($filename));
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
```

```
print $page;
curl_close($c);
?>
```

参见

curl_setopt()函数的文档（*http://www.php.net/curl-setopt*）和流选项的文档（*http://www.php.net/wrappers.http*）。PEAR的HTTP_Request类的文档（*http://pear.php.net/package/HTTP_Request*）。RFC 2616规范第5.1.1节所讨论的请求方法（*http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5.1.1*）。

13.6 通过超时定位URL

问题

你想取得一个远程URL，但又不想因为远程服务器繁忙或者速度慢而等待太长时间。

方案

在使用http流时，启用default_socket_timeout配置指令。例13-28中等待与远程服务器建立连接的时间不会超过15秒。

例13-28：在使用http流时设置超时

```
<?php
// 超时时间设为15 秒
ini_set('default_socket_timeout', 15);
$page = file_get_contents('http://slow.example.com/');
```

注意，改变default_socket_timeout后会影响所有新的套接字或者通过脚本执行创建的远程连接。

在使用cURL时，需要像例13-29所示的那样设置CURLOPT_CONNECTTIMEOUT选项。

例13-29：在使用cURL时设置超时

```
<?php
$c = curl_init('http://slow.example.com/');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_CONNECTTIMEOUT, 15);
$page = curl_exec($c);
curl_close($c);
?>
```

使用HTTP_Request时，在传递给HTTP_Request构造器的参数数组中设置超时选项，如例13-30所示。

例13-30：在使用HTTP_Request时设置超时

```
<?php
require_once 'HTTP/Request.php';
$opts = array('timeout' => 15);
$req = new HTTP_Request('http://slow.example.com/', $opts);
$req->sendRequest();
?>
```

讨论

远程服务器就像是反复无常的动物。即使是最最健壮的、企业级的中枢服务都可能会遭遇中断。作为备用方案，可以启用一个值得依赖的远程服务，但又可能因为本地服务器与远程服务器之间的网络问题而导致你的请求得不到处理。此时，如果要使用的远程数据源是构造你的页面过程中的一个环节，那么限制PHP在与远程服务器建立连接时的等待时间是个不错的主意。

本方案中介绍的所有技术都是用于限制PHP等待与远程服务器连接的超时时间的。当连接建立以后，所有碰运气的做法都会根据响应时间而取消。如果你确实需要快速响应，还可以另外设置PHP在从已经连接的套接字获取数据时需要等待的最长时间。对于基于流的连接而言，可以使用stream_set_timeout()函数。该函数接受一个流资源，因而必须要通过fopen —— 不是file_get_contents() —— 来打开一个流。例13-31把读取数据的超时时间设置为20秒。

例13-31：在使用http流时设置读取超时时间

```
<?php
$url = 'http://slow.example.com';
$stream = fopen($url, 'r');
stream_set_timeout($stream, 20);
$response_body = stream_get_contents($stream);
?>
```

在使用cURL时，需要把CURLOPT_TIMEOUT设置为curl_exec()应该执行的最长时间。其中既包括连接超时也包括读取整个响应主体中数据的超时时间。

如果使用的是HTTP_Request，那么需要在传递给构造器的参数数组中添加readTimeout元素及相应的值。该元素的值必须是一个包含秒数和微秒数两个元素的数组。例13-32把读取数据的超时时间设置为20秒。

例13-32：通过HTTP_Request来设置读取数据的超时时间

```
<?php
require_once 'HTTP/Request.php';
$opts = array('readTimeout' => array(20,0));
$req = new HTTP_Request('http://slow.example.com/', $opts);
$req->sendRequest();
?>
```

虽然设置连接和读取超时时间可以提升性能，但也有可能会导致不完整的响应。也就是说，你的脚本在到达超时时间之前可能只读取了部分响应。因此，如果设置了超时时间，一定要对取得的数据进行完整性验证。一种替代性的解决方案是，在快速生成页面的情况下，可以用一个单独的进程来取得外部数据并将数据写入本地缓存当中。这样就可以从缓存中读取数据来生成页面，而无须担心超时或者部分响应的问题。

参见

curl_setopt()函数的文档（http://*www.php.net/curl-setopt*），stream_set_timeout()函数的文档（*http://www.php.net/stream_set_timeout*），default_socket_timeout()函数的文档（*http://www.php.net/filesystem*）以及PEAR的HTTP_Request类的文档（*http://pear.php.net/package/HTTP_Request*）。

13.7 定位HTTPS URL

问题

你想取得一个安全的URL。

方案

使用技巧13.1与技巧13.2中介绍的任何一种技术，并提供以https开头的URL。

讨论

只要PHP中启用了SSL库，如OpenSSL，那么所有能够取得常规URL的函数也都可以取得安全的URL。可以通过查看phpinfo()输出中的“openssl”部分来知道你所安装的PHP是否支持SSL。

参见

技巧13.1和技巧13.2中定位常规URL的内容，以及OpenSSL项目的资料（Project at *http://www.openssl.org/*）。

13.8 调试原始HTTP数据交换

问题

你想对浏览器向你的服务器发出的HTTP请求以及对应的HTTP响应进行分析。例如，你的服务器并没有对某个特定的请求提供所要求的响应，而你想搞清楚所请求的组件到底是什么。

方案

对于简单的请求，可以通过Telnet连接到服务器并输入请求的头部信息。例13-33是一个信息交换的示例。

例13-33：通过Telnet发送请求

```
% telnet www.example.com 80
Trying 10.3.75.31...
Connected to www.example.com (10.3.75.31).
Escape character is '^] '.
GET / HTTP/1.0
Host: www.example.com

HTTP/1.1 200 OK
Date: Sun, 03 Dec 2006 02:54:01 GMT
Server: Apache/2.2.2 (Unix)
Last-Modified: Fri, 20 Oct 2006 20:16:24 GMT
ETag: "1348010-2c-4c23b600"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html

 [ the body of the response ]
```

讨论

当你输入请求的头部信息时，Web服务器并不知道它是什么，可能也仅仅是你在输入一行字而不是浏览器提交的请求。不过，某些浏览器会设置它们等待请求的超时时间，所以重新输入这个请求然后再粘贴到Telnet中去还是有用的。请求中的第一行包含了请

求的方法（get），一个空格和你想取得文件的路径（/），之后又是一空格和你所使用的协议（HTTP/1.0）。第二行，是主机头信息，它告诉服务器如果许多Web应用程序都共享同一个IP地址，那么应该使用哪一个虚拟主机。接下来的空行告诉服务器请求结束——紧接着下一行就是返回的响应信息：首先是一些头部信息，然后是一个空行，接着是响应的主体内容。要完成这类任务，也可以使用Netcat程序（*http://netcat.sourceforge.net/*）。

往Telnet中粘贴文本可能会很乏味，而且用这种方式在发送post请求时会更麻烦。如果你是通过HTTP_Request来发送请求，那么可以通过getResponseHeader()和getResponseBody()方法来取得响应的头部和主体信息，如例13-34所示。

例13-34：通过HTTP_Request来取得响应信息

```
<?php
require 'HTTP/Request.php';
$r = new HTTP_Request('http://www.example.com/submit.php');
$r->setMethod(HTTP_REQUEST_METHOD_POST);
$r->addPostData('monkey','uncle');
$r->sendRequest();

$response_headers = $r->getResponseHeader();
$response_body    = $r->getResponseBody();
?>
```

如果要取得特殊的响应头部信息，可以把相应的头部名称传递给getResponseHeader()。头部名称必须是小写形式。在没有参数的情况下，getResponseHeader()会返回一个包含所有头部信息的数组。HTTP_Request不会把输出请求保存到变量中，但是可以通过调用_buildRequest()方法来重新建立该请求，如例13-35所示。

例13-35：通过HTTP_Request来取得请求头部信息

```
<?php
require 'HTTP/Request.php';

$r = new HTTP_Request('http://www.example.com/submit.php');
$r->setMethod(HTTP_REQUEST_METHOD_POST);
$r->addPostData('monkey','uncle');

print $r->_buildRequest();
?>
```

例13-35中的请求包含如下内容：

```
POST /submit.php HTTP/1.1
User-Agent: PEAR HTTP_Request class ( http://pear.php.net/ )
Content-Type: application/x-www-form-urlencoded
Connection: close
```

```
Host: www.example.com
Content-Length: 12

monkey=uncle
```

通过http流来访问响应头部信息是可行的，但必须使用像fopen()一样能返回流资源的函数。在请求发送完毕后把相应的流资源传递给stream_get_meta_data()时，所得到的一组元数据就是响应头部信息的集合。例13-36中示范了如何通过流资源来访问响应头部信息。

例13-36：通过http流来获得响应头部信息

```
<?php
$url = 'http://www.example.com/submit.php';
$stream = fopen($url, 'r');
$metadata = stream_get_meta_data($stream);
// 头部信息保存在 'wrapper_data'中
foreach ($metadata ['wrapper_data']  as $header) {
    print $header . "\n";
}
// 响应主体信息可以通过
// stream_get_contents()取得
$response_body = stream_get_contents($stream);
?>
```

stream_get_meta_data()返回一个包含有关流信息的数组。而该数组中的wrapper_data元素包含的是有关包装器的专门数据。对于http包装器来说，其中每一个子数组元素就是一个响应头部信息。例13-36的输出结果如下：

```
HTTP/1.1 200 OK
Date: Sun, 07 May 2006 18:24:37 GMT
Server: Apache/2.2.2 (Unix)
Last-Modified: Sun, 07 May 2006 01:58:12 GMT
ETag: "1348011-7-16167500"
Accept-Ranges: bytes
Content-Length: 7
Connection: close
Content-Type: text/plain
```

fopen()函数可以接受一个可选的流环境作为参数。如果你想使用一个这样的参数，可以将其作为第四个参数传递给fopen()（第二个参数是模式；第三个参数是可选的，表示在查找文件时是否使用include_path的标志）。

对于cURL，可以通过设置CURLOPT_HEADER选项使curl_exec()的输出中包含响应头部信息，如例13-37所示。

例13-37：通过cURL来获取响应头部信息

```
<?php
```

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_HEADER, true);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFiELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$response_headers_and_page = curl_exec($c);
curl_close($c);
?>
```

要想直接把响应头部信息写入一个文件中，可以通过fopen()打开一个文件句柄并将CURLOPT_ WRITEHEADER选项设置为该句柄，如例13-38所示。

例13-38：通过cURL把响应头部信息写入一个文件中

```
<?php
$fh = fopen('/tmp/curl-response-headers.txt','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFiELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_WRITEHEADER, $fh);
$page = curl_exec($c);
curl_close($c);
fclose($fh) or die($php_errormsg);
?>
```

cURL的CURLOPT_VERBOSE选项会使curl_exec()和curl_close()把调试信息输出为标准的错误信息，包括请求的内容，如例13-39所示。

例13-39：通过cURL输出详尽的信息

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_VERBOSE, true);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFiELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
```

例13-39的输出结果如下：

```
* Connected to www.example.com (10.1.1.1)
> POST /submit.php HTTP/1.1
Host: www.example.com
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Content-Length: 23
Content-Type: application/x-www-form-urlencoded

monkey=uncle&rhino=aunt* Connection #0 left intact
* Closing connection #0
```

因为cURL将调试信息输出为标准的错误信息而非标准的输出，所以使用输出缓冲无法

捕获这些信息。不过，你可以打开一个准备写入的文件句柄并把CURLOUT_STDERR选项设置为该文件句柄，以便将调试信息转存到文件中。这个过程用例13-40来演示。

例13-40：将cURL详尽的输出写入文件

```
<?php
$fh = fopen('/tmp/curl.out','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_VERBOSE, true);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFiELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_STDERR, $fh);
$page = curl_exec($c);
curl_close($c);
fclose($fh) or die($php_errormsg);
?>
```

通过cURL访问头部信息的另一种方式是编写一个“头部函数”。这个函数与cURL的“写入函数”相似，只不过调用它是为了处理响应头部而不是处理响应主体。例13-41定义了一个HeaderSaver类，其header()方法可以用作头部函数来累积响应头部信息。

例13-41：使用cURL头部函数

```
<?php

class HeaderSaver {
    public $headers = array();
    public $code = null;

    public function header($curl, $data){
        if (is_null($this->code) &&
            preg_match('@^HTTP/\d\.\d (\d+) @',$data,$matches)) {
            $this->code = $matches [1] ;
        } else {
            // 删除后面的换行
            $trimmed = rtrim($data);
            if (strlen($trimmed)) {
                // 如果该行以空格或制表符开头，它是前一个头部信息的延续
                if (($trimmed [0]  == ' ') || ($trimmed [0]  == "\t")) {
                    // 把前导空白符折叠为一个空格
                    $trimmed = preg_replace('@^ [ \t] +@',' ', $trimmed);
                    $this->headers [count($this->headers)-1]  .= $trimmed;
                }
                // 否则，是一个新的头部
                else {
                    $this->headers []  = $trimmed;
                }
            }
        }
        return strlen($data);
    }
```

```
}

$h = new HeaderSaver();
$c = curl_init('http://www.example.com/plankton.php');
// 注意头部函数
curl_setopt($c, CURLOPT_HEADERFUNCTION, array($h,'header'));
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
// 在 $h 中保存数据
print 'The response code was: ' . $h->code . "\n";
print "The response headers were: \n";
foreach ($h->headers as $header) {
    print "  $header\n";
}
```

HTTP 1.1标准规定头部信息可以跨行，但必须在其他头部信息行的开始处放置至少一个空格或者制表符。但是，由stream_get_meta_data()和HTTP_Request::getResponseHeader()返回的头部信息数组却不能适当地处理多行头部信息。多行头部信息的其他行会被看成单独的头部信息。不过，例13-41能够正确地组合多行头部信息的其他附加行。

参见

curl_setopt()函数的文档（*http://www.php.net/curl-setopt*），stream_get_meta_data()函数的文档（*http://www.php.net/stream_get_meta_data*），fopen()函数的文档（*http://www.php.net/fopen*）和PEAR中HTTP_Request类的文档（*http://pear.php.net/package/HTTP_Request*）。RFC 2616规范中定义的HTTP请求的语法规则（*http://www.w3.org/Protocols/rfc2616/rfc2616.html*）。该规范中的第4.2节中有关多行信息头的内容（*http://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html#sec4.2*）。此外，可以从GNU Netcat项目的网站（*http://netcat.sourceforge.net/*）中下载netcat程序。

13.9 标记网页

问题

你想在显示网页时，比如在显示一个搜索结果页面时突出显示某些关键词。

方案

建立一个用于替换每个需要想突出显示的词的数组。然后，将页面分割成“HTML元素”和“HTML元素之间的文本”，并且只对HTML元素之间的文本进行替换操作。例13-42将$body中与$words中匹配的内容进行了突出显示。

例13-42：为网页加标记

```
$body = '
<p>I like pickles and herring.</p>

<a href="pickle.php"><img src="pickle.jpg"/>A pickle picture</a>

I have a herringbone-patterned toaster cozy.

<herring>Herring is not a real HTML element!</herring>
';

$words = array('pickle','herring');
$replacements = array();
foreach ($words as $i => $word) {
    $replacements []  = "<span class='word-$i'>$word</span>";
}

// 用看起来与HTML元素类似的合理标记符把页面分割成块
$parts = preg_split("{(<(?:\" [^\"] *\"|' [^'] *'| [^'\">] )*>)}",
                    $body,
                    -1,  // 不限制块的大小
                    PREG_SPLIT_DELIM_CAPTURE);
foreach ($parts as $i => $part) {
    // 如果是HTML元素则跳过
    if (isset($part [0] ) && ($part [0]  == '<')) { continue; }
    // 用<span/>标签包围单词
    $parts [$i]  = str_replace($words, $replacements, $part);
}

// 重建$body
$body = implode('',$parts);

print $body;
?>
```

讨论

例13-42的输出结果为：

```
<p>I like <span class='word-0'>pickle</span>s and <span class='word-1'>herring</
span>.
</p>

<a href="pickle.php"><img src="pickle.jpg"/>A <span class='word-0'>pickle</span>
picture</a>

I have a <span class='word-1'>herring</span>bone-patterned toaster cozy.

<herring>Herring is not a real HTML element!</herring>
```

$words中包含的每个单词（pickle和herring）都被一个带有特殊类属性的标签

给包围了起来。进而可以通过CSS样式表为这些类添加特殊的显示属性，例如亮黄色的背景或者边框。

例13-42中的正则表达式通过HTML元素把$body分割为一系列文本块。这样我们就可以只对HTML元素之间的文本执行替换操作，而跳过HTML元素或者其值中可能包含搜索项目的属性。虽然正则表达式在这里能够完美地实现其搜索匹配的目标，但如果你的网页中存在不符合标准的标签，比如搭配不当或者没有对引号进行转义，那么最终得到的结果可能会令你困惑。

由于str_replace()区分大小写，所以只有与$words中的单词完全匹配的字符串才会被替换。例13-42中最后一个Herring没有被替换就是因为它有一个字母是大写的。如果要实现不区分大小写的匹配，那么就需要在正则表达式上作文章而不是依靠str_replace()（因为必须要保持所匹配内容的大小写不变，所以不能用str_ireplace()）。例13-43显示了通过正则表达式来实现这一点的修正后的代码。

例13-43：通过正则表达式实现在网页中加标记

```
<?php
$body = '
<p>I like pickles and herring.</p>

<a href="pickle.php"><img src="pickle.jpg"/>A pickle picture</a>

I have a herringbone-patterned toaster cozy.

<herring>Herring is not a real HTML element!</herring>
';

$words = array('pickle','herring');
$patterns = array();
$replacements = array();
foreach ($words as $i => $word) {
    $patterns []  = '/' . preg_quote($word) .'/i';
    $replacements []  = "<span class='word-$i'>\\0</span>";
}

// 用看起来与HTML元素类似的合理标记符把页面分割成块
$parts = preg_split("{(<(?:\" [^\"] *\"|' [^'] *'| [^'\">] )*>)}",
                    $body,
                    -1,  // 不限制块的大小
                    PREG_SPLIT_DELIM_CAPTURE);
foreach ($parts as $i => $part) {
    // 如果是HTML元素则跳过
    if (isset($part [0] ) && ($part [0]  == '<')) { continue; }
    // 用<span/>标签包围单词
    $parts [$i]  = preg_replace($patterns, $replacements, $part);
}
```

```
// 重建$body
$body = implode('',$parts);

print $body;
?>
```

以上两者的差别是例13-43在循环中构建了一个$patterns数组，而且它调用的是preg_replace()（其中使用了$patterns数组）而不是str_replace()。$patterns中每个元素结尾处的i表示的就是不区分大小写。而$replacements中的\\0则表示在替换过程中保持所匹配内容的大小写形式。

运用正则表达式也能轻易做到避免子字符串匹配。如在例13-42和例13-43中，herringbone中的herring也被突出显示出来了。要避免这种情况发生，可以把例13-43中的$patterns[]= '/' . preg_quote($word) .'/i'改成$patterns[]= '/\b' . preg_quote($word) .'\b/i'。该模式中后加的\b选项的作用是告诉preg_replace()只匹配独立存在的单词。

参见

str_replace()函数的文档（*http://www.php.net/str_replace*），str_ireplace()函数的文档（*http://www.php.net/str_ireplace*），preg_replace()函数的文档（*http://www.php.net/preg_replace*）和preg_split()函数的文档（*http://www.php.net/preg_split*）。

13.10 清理不完整或非标准的HTML

问题

你手上有一些未遵循语法规范编写的HTML代码，你想对其进行清理。这样可以使HTML更容易解析并且能保证所生成的页面与标准兼容。

方案

使用PHP的Tidy扩展。通过流行而且强大的HTML Tidy库可以把乱糟糟的标签代码转换成格式良好并与标准兼容的HTML或XHTML。例13-44显示了如何修补HTML文件。

例13-44：通过Tidy库修补HTML文件

```
<?php
$fixed = tidy_repair_file('bad.html');
file_put_contents('good.html', $fixed);
?>
```

讨论

这里使用了HTML Tidy库中随着时间推移而建立起来的数量庞大的规则和功能，能够创造性地处理各种HTML代码问题。幸运的是，我们不用关心Tidy库中都包含哪些具体的规则，而只要坐享其成就可以了。向tidy_repair_file()传递一个文件名就能得到一个清理后的版本。例如，如果*bad.html*中包含以下代码：

```
<img src="monkey.jpg">

<b>I <em>love</b> monkeys</em>.
```

那么例13-44就会把下面的代码写入到*good.html*中：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title></title>
</head>
<body>
<img src="monkey.jpg"> <b>I <em>love</em> monkeys</b>.
</body>
</html>
```

Tidy库中有复杂的配置选项可以影响其生成的输出内容。有关这些选项可以在*http://tidy.sourceforge.net/docs/quickref.html*上面找到。要设置相关的配置选项，可以给tidy_repair_file()传递一个包含配置选项及设定值的数组作为其第二个参数。例13-45中使用了output-xhtml选项，含义是让Tidy生成有效的XHTML文档。

例13-45：通过Tidy生成XHTML

```
<?php
$config = array('output-xhtml' => true);
$fixed = tidy_repair_file('bad.html', $config);
file_put_contents('good.xhtml', $fixed);
?>
```

例13-45会把以下内容写入到*good.html*中：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
</head>
<body>
<img src="monkey.jpg" /> <b>I <em>love</em> monkeys</b>.
</body>
</html>
```

如果你的HTML代码是字符串形式并且没有保存为文件，可以使用tidy_repair_string()函数。该函数要求第一个参数是一个包含HTML代码的字符串而不是一个文件名。

参见

tidy_repair_file()函数的文档（*http://www.php.net/tidy_repair_file*），tidy_repair_string()函数的文档（*http://www.php.net/tidy_repair_string*）和Tidy配置选项的详细资料（*http://tidy.sourceforge.net/docs/quickref.html*）。

13.11 从HTML文件中提取链接

问题

你想提取在HTML文档中指定的URL。

方案

使用Tidy将该文档转换成XHTML，然后使用XPath查找所有链接，如例13-46所示。

例13-46：通过Tidy和XPath提取链接

```
<?php
$doc = new DOMDocument();
$opts = array('output-xml' => true,
              // 防止DOMDocument在实体处理上出现混淆
              'numeric-entities' => true);
$doc->loadXML(tidy_repair_file('linklist.html',$opts));
$xpath = new DOMXPath($doc);
// 告诉 $xpath 有关XHTML名称空间的信息
$xpath->registerNamespace('xhtml','http://www.w3.org/1999/xhtml');
foreach ($xpath->query('//xhtml:a/@href') as $node) {
    $link = $node->nodeValue;
    print $link . "\n";
}
```

如果Tidy不可用，可以通过例13-47中定义的pc_link_extractor()函数来提取链接。

例13-47：在没有Tidy的情况下提取链接

```
<?php
$html = file_get_contents('linklist.html');
$links = pc_link_extractor($html);
foreach ($links as $link) {
    print $link [0]  . "\n";
}
```

```
function pc_link_extractor($html) {
    $links = array();
    preg_match_all('/<a\s+.*?href= [\"\'] ?( [^\"\' >] *) [\"\'] ? [^>] *>(.*?)<\/a>/i',
                   $html,$matches,PREG_SET_ORDER);
    foreach($matches as $match) {
        $links []  = array($match [1] ,$match [2] );
    }
    return $links;
}
```

讨论

如果在使用Tidy生成XHTML文档时设置了output-xhtml选项，那么生成的文档中可能会包含实体字符而不是由基本的XML规范定义的四个实体值（< > & "）。而设置numeric-entities选项可以避免这些实体出现在生成的XHTML文档中。这些实体的存在会导致DOMDocument出现“未定义实体”的错误。另一种解决方案是不设置numeric-entities选项而将$doc->resolveExternals设置为true。这是告诉DOMDocument取得所加载文件中引用的文档类型定义（Document Type Definition）并据以处理实体。Tidy也能生成带有适当DTD的XML文档。以这种方式生成XML的一个不足是，DTD中URL指向的是一个外部Web服务器上的资源，因而每当你的程序运行时都要下载该资源。

XHTML也是XML应用程序 —— 为表示HTML而定义的XML词汇表。因此，其包含的所有元素（我们熟悉的<a/>,</h1>等等）都处于同一个名称空间中。该名称空间的URI是*http://www.w3.org/1999/xhtml*。为了保证XPath能有效地工作，这个名称空间必须要加上一个前缀（这就是registerNamespace()方法所做的事），然后再进行XPath查询。

在Tidy库不可用的情况下，pc_link_extractor()函数是一个有效的替代方案。不过，其中的正则表达式并不能匹配所有的链接，例如那些通过十六进制转义字符建立的链接。但对于多数格式良好的HTML来说这个函数都是适用的。该函数返回的是一个数组，其中的每个元素本身也是一个包含两个元素的数组。第一个元素是链接的目标，第二个元素是链接的锚 —— 即链接中的文本。

例13-46中的XPath表达式只用于取得链接，而不能取得锚。例13-48中提供了一个既能提取链接也能提取锚的替代方案。

例13-48：通过Tidy和XPath提取链接和锚

```
<?php
$doc = new DOMDocument();
$opts = array('output-xhtml'=>true,
              // 防止DOMDocument在实体处理上出现混淆
              'numeric-entities' => true);
```

```
$doc->loadXML(tidy_repair_file('linklist.html',$opts));
$xpath = new DOMXPath($doc);
//  告诉 $xpath 有关XHTML名称空间的信息
$xpath->registerNamespace('xhtml','http://www.w3.org/1999/xhtml');
foreach ($xpath->query('//xhtml:a') as $node) {
    $anchor = trim($node->textContent);
    $link = $node->getAttribute('href');
    print "$anchor -> $link \n";
}
```

在例13-48中，XPath查询会查找所有<a/>元素的节点。而该节点的textContent属性保存了锚文本和作为href属性值的链接。

参见

DOMDocument的文档（*http://www.php.net/DOM*），DOMXPath::query()方法的文档（*http://www.php.net/DOM_DOMXPath::query*），DOMXPath::registerNamespace()方法的文档（*http://www.php.net/DOM_DOMXPath::registerNamespace*），tidy_repair_file()函数的文档（*http://www.php.net/tidy_repair_file*）以及preg_match_all()函数的文档（*http://www.php.net/preg_match_all*）。技巧13.10中有关Tidy库的更多信息。关于XPath的更多资料（*http://www.w3.org/TR/xpath*）以及XHTML的详细信息（*http://www.w3.org/TR/xhtml1*）。

13.12 将纯文本转换为HTML

问题

你想将纯文本内容转换成经过适当格式化的HTML代码。

方案

首选，通过htmlentities()来对其中的实体进行编码。然后，把该文本转换为各种对应的HTML结构。例13-49中的pc_text2html()函数可以对链接和段落进行基本的转换。

例13-49：pc_text2html()函数

```
<?php
function pc_text2html($s) {
  $s = htmlentities($s);
  $grafs = split("\n\n",$s);
  for ($i = 0, $j = count($grafs); $i < $j; $i++) {
    // 链接为http或ftp的URL
    $grafs [$i]  = preg_replace('/((ht|f)tp:\/\/ [^\s&] +)/',
                                '<a href="$1">$1</a>',$grafs [$i] );
```

```
    // 电子邮件地址的链接
    $grafs [$i]  = preg_replace('/ [^@\s] +@( [-a-z0-9] +\.)+ [a-z] {2,}/i',
        '<a href="mailto:$1">$1</a>',$grafs [$i] );

    // 建立新段落
    $grafs [$i]  = '<p>'.$grafs [$i] .'</p>';
  }
  return implode("\n\n",$grafs);
}
?>
```

讨论

你对纯文本内容适合以何种结构表现考虑得越到位，那么所生成的HTML文件就会越好。例如，如果纯文本中是以*asterisks*或/slashes/来表示强调，那么可以添加转换这种表现形式的规则，如例13-50所示。

例13-50：文本到HTML转换的更多规则

```
<?php
$grafs [$i]  = preg_replace('/(\A|\s)\*( [^*] +)\*(\s|\z)/',
                            '$1<b>$2</b>$3',$grafs [$i] );
$grafs [$i]  = preg_replace('{(\A|\s)/( [^/] +)/(\s|\z)}',
                            '$1<i>$2</i>$3',$grafs [$i] );
?>
```

参见

preg_replace()函数的文档（*http://www.php.net/preg_replace*）。

13.13 将HTML转换为文本

问题

你需要把HTML转换成具有可读性的、格式化的纯文本。

方案

使用html2text类（从*http://www.chuggnutt.com/html2text.php*中能下载到）。例13-51示范了该类的用法。

例13-51：将HTML转换成纯文本

```
<?php
require_once 'class.html2text.inc';
$html = file_get_contents('http://www.example.com/article.html');
```

```
$converter = new html2text($html);
$plain_text = $converter->get_text();
?>
```

讨论

html2text类中内置了大量的格式化规则，因而也会为生成的纯文本中的标题、段落等提供很多种可视的布局效果。而且，在生成的纯文本的底部还会包含HTML文件中所有链接的一个列表。

参见

*http://www.chuggnutt.com/html2text.php*中有关html2text类的更多信息和下载该类的链接。

13.14 删除HTML和PHP标签

问题

你想删除一个字符串或一个文件中的HTML和PHP标签。例如，你想在输出一段字符串之前保证其中不会包含HTML标签，或者把该字符串传递给eval()函数之前确保其中不会包含PHP标签。

方案

用strip_tags()来删除字符串中包含的HTML和PHP标签，如例13-52所示。

例13-52：删除HTML和PHP标签

```
<?php

$html = '<a href="http://www.oreilly.com">I <b>love computer books.</b></a>
         <?php echo "Hello!" ?>';
print strip_tags($html);
?>
```

例13-52的输出结果为：

```
I love computer books.
```

如果想在读取流资源的同时剥离其中的标签，可以使用string.strip_tags流过滤器，如例13-53所示。

例13-53：从流资源中删除HTML和PHP标签

```
<?php
$stream = fopen('elephant.html','r');
stream_filter_append($stream, 'string.strip_tags');
print stream_get_contents($stream);
?>
```

讨论

无论strip_tags()还是string.strip_tags过滤器都可以根据我们的设置保留（不删除）某些标签。例如，可以为strip_tags()传递一个包含允许保留标签的字符串作为第二个参数。字符串中标签的书写规范是大小写敏感的，而且对于成对出现的标签，只须指定开标签。例如，如果想从$html中删除之外的标签，可以调用strip_tags($html,'')。

使用string.strip_tags过滤器时，可以给stream_filter_append()传递一个类似的字符串作为其第四个参数。stream_filter_append()的第三个参数用于控制过滤器是否应用于读取（STREAM_FILTER_READ）、写入（STREAM_FILTER_WRITE）还是二者全都适用（STREAM_FILTER_ALL）。例13-54中除了允许保留标签之外，其他与例13-53相同。

例13-54：从流资源中删除部分HTML和PHP标签

```
<?php
$stream = fopen('elephant.html','r');
stream_filter_append($stream, 'string.strip_tags',STREAM_FILTER_READ,'<b><i>');
print stream_get_contents($stream);
?>
```

stream_filter_append()除了能够接受字符串，也可以接受包含标签名的数组：用array('b','i')可以代替''。

无论是使用strip_tags()还是流过滤器，都不会删除允许保留的标签中的属性。这意味着能够改变显示外观（如style）或执行JavaScript（任何事件处理器）的属性会得到保留。如果你想在web浏览器中把任意来源的内容经“剥离后”显示给用户，这些被保留的属性就有可能导致跨站点的脚本攻击问题。

参见

strip_tags()函数的文档（*http://www.php.net/strip-tags*），stream_filter_append()函数的文档（*http://www.php.net/stream_filter_append*）和流过滤器的相关资料（*http://www.php.net/filters*）。PEAR中的HTML_Safe包致力于删除HTML中的不安全内容，有

关该包的资料在*http://pear.php.net/package/HTML_Safe*中可以找到。另外，技巧18.4中介绍了更多有关跨站点脚本的内容。

13.15 响应Ajax请求

问题

你想用JavaScript通过XMLHTTPRequest生成页面内的请求，同时需要发送数据以答复这些请求。

方案

设置Content-Type头部信息并发送经过适当格式化的数据。例13-55发送了一个小型XML文档作为响应。

例13-55：发送XML响应

```
<?php header('Content-Type: text/xml'); ?>
<menu>
 <dish type="appetizer">Chicken Soup</dish>
 <dish type="main course">Fried Monkey Brains</dish>
</menu>
```

例13-56使用了PEAR Services_JSON包发送了一个JSON响应。

例13-56：发送一个JSON响应

```
<?php
require_once 'Services/JSON.php';
$menu = array();
$menu [] = array('type' => 'appetizer',
                 'dish' => 'Chicken Soup');
$menu [] = array('type' => 'main course',
                 'dish' => 'Fried Monkey Brains');
header('Content-Type: application/json');
$json = new Services_JSON();
print $json->encode($menu);
?>
```

例13-57使用了PECL的json扩展（在PHP 5.2及之后版本中包含了该扩展）发送了一个JSON响应。

例13-57：通过PECL的json扩展发送JSON响应

```
<?php
$menu = array();
$menu [] = array('type' => 'appetizer',
                 'dish' => 'Chicken Soup');
```

```
$menu[]  = array('type' => 'main course,'
                 'dish' => 'Fried Monkey Brains');
header('Content-Type: application/json');
print json_encode($menu);
?>
```

讨论

单纯从PHP的角度来看，发送一个基于XMLHTTPRequest请求的响应与其他响应没有什么不同，都是发送某种必要的头部信息并返回一些文本。要说有不同，那就是通常这些头部信息都会与所发送的文本内容保持一致。

对这类响应来说，JSON是一种特别有用的格式。因为在JavaScript中处理JSON格式的数据很容易。例13-56中的输出结果如下：

```
 [{"type":"appetizer","dish":"Chicken Soup"},
{"type":"main course","dish":"Fried Monkey Brains"}]
```

这其实就是把数据编码成JavaScript的关联数组。PEAR的Services_JSON模块是把PHP的数据结构（标量数组和对象）转换成JSON字符串（或者反之）的一种便捷的方法。由于它是一个PEAR模块，所以即使从未修改过*php.ini*文件或者在无法安装二进制扩展的情况下仍然可以使用它。如果你能自己安装扩展，可以考虑PECL的json扩展来获得极大的速度提升。它的json_encode()和json_decode()函数可以把PHP的数据结构转换成JSON字符串，并进行反向转换。

对于这种类型的响应，注意缓存也是相当重要的。对于来自JavaScript内部的请求，不同浏览器都具有自己创造性的缓存策略。如果你在响应中发送的是动态数据（一般都是），那么你可能不希望这些数据被缓存。此时可用的两种防止缓存的技术就是设置头部信息和URL异化。例13-58显示了为避免浏览器缓存响应数据而在PHP中使用的各种反缓存头部信息。

例13-58：设置反缓存头部信息

```
<?php
header("Expires: 0");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");
// 添加一些IE专有的选项
header("Cache-Control: post-check=0, pre-check=0", false);
// 针对 HTTP/1.0
header("Pragma: no-cache");
?>
```

另一种反缓存手段是URL异化，它需要与生成请求的JavaScript协作。这种方法就是在

每个请求的查询字符串后面都添加由任意值生成的名/值对。这样就会使请求每次生成的URL都不相同，这样就能对预防任何不加分析就缓存的行为。在生成任意值时，可以适当地使用JavaScrip的Math.random()函数。

参见

header()函数的文档（*http://www.php.net/header*）。有关XMLHTTPRequest的更多阅读资料（*http://en.wikipedia.org/wiki/XMLHttpRequest*），JSON的信息（*http://www.json.org*），Services_JSON的资料（*http://pear.php.net/pepr/pepr-proposal-show.php?id=198*）和PECL中json扩展的信息（*http://pecl.php.net/package/json*）。Michael Radwin撰写的一篇有关HTTP缓存入门的好文章“HTTP Caching and Cache-Busting for Content Publishers”（*http://public.yahoo.com/~radwin/talks/http-caching-apachecon2005.htm*）。RFC 2616规范第13节中有关HTTP缓存的详尽内容（*http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13*）。

13.16 与JavaScript应用集成

问题

你想在不重新加载整个页面的前提下，通过服务器端的数据来更新页面局部的内容。例如，你想根据搜索结果生成一个列表。

方案

使用JavaScript工具包，比如Dojo，来承担客户端的工作。这样，某个用户的特定动作（如点击按钮）就会触发一个发送到服务器的请求。同时用PHP编写适当的代码来生成包含正确数据的响应。然后，再使用该JavaScript工具包来把结果正确地放到页面中。

例13-59显示了一个简单HTML文档，其中加载了Dojo和在例13-60中定义的其他代码文件。例13-60中的代码起到了JavaScript与服务器端之间粘合剂的作用，它会在搜索按钮被点击时向服务器发送一个请求，并保证返回的结果能被更新到页面中正确的位置上。例13-61中是执行搜索任务并发回JSON格式响应的PHP代码。

例13-59：用于整合JavaScript应用的基本HTML代码

```
<!—加载Dojo -->
<script type="text/javascript" src="/dojo.js"></script>
<!—加载我们编写的 JavaScript -->
<script type="text/javascript" src="/search.js"></script>
```

```
<!-- 某些输入元素 -->
<input type="text" id="q" />
<input type="button" id="go" value="Search"/>
<hr/>
<!-- 输入结果放在这里 -->
<div id="output"></div>
```

例13-60：JavaScript整合应用的粘合剂

```
// 当页面加载完成后，运行下面的代码
dojo.addOnLoad(function() {
    // 当“go”按钮被点击时调用search()函数
    dojo.event.connect(dojo.byId('go'), 'onclick', 'search');
});

function search() {
    // 取得文本框中的值
    var q = dojo.byId('q').value;
    // 向服务器发送请求
    // url的值应该是处理搜索的页面所保存的位置
    dojo.io.bind({ 'url': '/search.php',
                   'content': { 'q': q },
                   // 响应的类型
                   'mimetype': 'text/json',
                   // 当响应数据接收完成后调用的函数
                   'load': showResults });
}

// 处理响应结果
function showResults(type, results, evt) {
    var html = '';
    // 如果得到一些结果……
    if (results.length > 0) {
        html = '<ul>';
        //建立一个列表
        for (var i in results) {
            html += '<li>' + dojo.string.escapeXml(results [i] ) + '</li>';
        }
        html += '</ul>';
    } else {
        html = 'No results.';
    }
    // 把生成的HTML代码放到页面中
    dojo.byId('output').innerHTML = html;
}
```

例13-61：为JavaScript生成响应的PHP代码

```
<?php
// 初始化 JSON
require_once 'Services/JSON.php';
$json = new Services_JSON();

$results = array();
$q = isset($_GET ['q'] ) ? $_GET ['q']  : '';
```

```
// 连接到第10章中的例子数据库
$db = new PDO('sqlite:/usr/local/data/zodiac.db');

// 执行查询
$st = $db->prepare('SELECT symbol FROM zodiac WHERE planet LIKE ? ');
$st->execute(array($q.'%'));

// 建立结果数组
while ($row = $st->fetch()) {
    $results []  = $row ['symbol'] ;
}

// 输出所有反缓存设置
header("Expires: 0");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");
// 添加一些IE专有的选项
header("Cache-Control: post-check=0, pre-check=0", false);
// 针对 HTTP/1.0
header("Pragma: no-cache");

// 响应类型是 JSON
header('Content-Type: application/json');

// 输出JSON数据
print $json->encode($results);
?>
```

讨论

例13-59中的HTML代码是有意设计成最少的。其全部内容都是一些必要的元素和调用外部脚本文件的代码。将JavaScript代码与HTML代码分离是一种良好的开发习惯 —— 就如同在服务器端分离表现逻辑和业务逻辑一样。例13-59中的第一个<script/>标签应该指向Dojo库所在的位置。第二个则应该指向保存例13-60中代码文件的位置。该例中的那一小段JavaScript函数为例13-59中的HMTL元素和例13-61中的服务器端代码提供了一座连通彼此的桥梁。该例开始处调用dojo.addOnLoad()是告诉web浏览器“当页面加载完成后，运行下面的JavaScript代码”，而那行代码则告诉浏览器“在go按钮被点击时，运行search()函数”。

很多JavaScript程序都是基于事件的 —— 一每行代码中设置的规则都在类似地告诉web浏览器“当某某事件发生时，运行这个函数”。而充斥页面中的JavaScript代码也并没有“从始至终”这样一个严格的执行顺序。这反而为用户提供了很多可能性 —— 点击按钮，填写文本框，点击链接等等。你的JavaScript代码中通常也会设置许多事件处理器 —— 响应点击、输入以及其他事件时运行的函数。

在例13-60中，search()函数使用了Dojo的dojo.io.bind()函数来把请求发送到服务器，并传递了文本框中的值作为q查询字符串的参数。dojo.io.bind()的另一个参数表示需要一个JSON类型的响应，而当请求的响应返回时，结果JSON数据应该被传递到showResults()函数中进行进一步处理。

接着，showResults()函数接受到响应的结果并据此建立HTML列表。最后，在列表建立完成时，将id值为output的 元素的内容设置为该HTML列表。

例13-61很像是三方小组中的一方。它与任何“根据用户输入从数据库中搜索资料”的PHP脚本一样，区别仅在于返回结果的方式不同。该脚本没有输出HTML，而是使用了技巧13.15中介绍的技术发回了一个不能缓存的JSON响应。

编写依赖于客户端JavaScript的应用程序需要采用同典型的PHP应用程序不一样的编程模式。例如，不是考虑如何生成完整的动态页面，而是要考虑如何生成客户端逻辑能够方便地显示或操纵的少量动态数据。诸如Dojo之类的工具包为构建这样的应用程序提供了健壮的平台。这些工具包抽象了许多繁琐的JavaScript编程实践 —— 如跨浏览器兼容性、异步I/O调用以及其他内部事务的处理过程。

也有一些围绕PHP开发的JavaScritp工具包，例如PEAR的HTML_Ajax和xajax。它们的目标是让你能够在JavaScript中方便地调用PHP函数和方法，以及简化把特定的JavaScript函数映射为特定的PHP函数的枯燥过程。虽然这些工具包能够提供基于PHP的便利，但却是以牺牲基于JavaScript的健壮性为代价的。尽管不论HTML_Ajax还是xajax在把某些服务器端PHP代码快速地捆扎成客户端逻辑时，可能会有效而且也很方便。但是如此构建的应用程序仍然无法与完全基于客户端而设计的应用程序相提并论。

同时这也说明，为了解决PHP与JavaScript之间轻松交互这一问题，很多人都在积极地寻求解决方案。相信在你看到这些文字时，更加便捷的方法一定已经出现了。

参见

技巧13.15中发送JSON响应的详细内容。Dojo的网站*http://www.dojotoolkit.org/*，xajax的网站*http://www.xajaxproject.org/*以及HTML_Ajax的文档（*http://pear.php.net/package/HTML_Ajax*）。“Getting Rich with PHP（*http://talks.php.net/show/tek06*）”中有关响应过多基于JavaScript的请求导致性能问题的讨论。其他的JavaScript工具包还包括script.aculo.us（*http://script.aculo.us/*），Prototype（*http://prototype.conio.net/*）和Yahoo!UI（用户界面，User Interface）库（*http://developer.yahoo.com/yui/index.html*）等。

13.17 编程：查找失效的链接

例13-62中的*stale-links.php*程序会生成一个包含链接及其状态信息的列表。它能告诉你链接是否正常，是否被转移到了其他地址或者是否已经无效了。在运行该程序时，需要传递一个要扫描其中链接的URL作为参数：

```
% php stale-links.php http://www.oreilly.com
http://www.oreilly.com/: OK
http://oreillynet.com/: OK
http://www.oreilly.com/store/: OK
http://safari.oreilly.com: OK
http://conferences.oreillynet.com/: OK
http://www.oreillylearning.com: OK
http://academic.oreilly.com: MOVED: http://academic.oreilly.com/index.csp
http://www.oreilly.com/about/: OK
...
```

这个*stale-links.php*程序使用了cURL扩展来找到网页（见例13-62）。首先，它会寻找命令行中指定的URL。当找到相应的页面后，该程序会通过技巧13.11中介绍的XPath技术取得那个页面中所有链接的列表。然后，在有必要时还会给每个链接前面添加基本的URL信息，以便找到这些链接。因为我们需要的只是来自响应中的头部信息，所以通过设置CURLOPT_NOBODY来使用HEAD方法而不是GET方法。而设置CURLOPT_HEADER选项是要告诉curl_exec()在它返回的字符串中包含响应头部。根据响应代码，会输出链接的状态，如果该链接已经转移则还会输出相应的新地址。

例13-62：stale-links.php程序

```
<?php

if (! isset($_SERVER ['argv'] [1] )) {
    die("No URL provided.\n");
}

$url = $_SERVER ['argv'] [1] ;

// 加载页面
list($page,$pageInfo) = load_with_curl($url);

if (! strlen($page)) {
    die("No page retrieved from $url");
}

// 为了便于解析转换成XML
$opts = array('output-xhtml' => true,
              'numeric-entities' => true);
$xml = tidy_repair_string($page, $opts);
$doc = new DOMDocument();
$doc->loadXML($xml);
```

```
$xpath = new DOMXPath($doc);
$xpath->registerNamespace('xhtml','http://www.w3.org/1999/xhtml');

// 对相对链接计算基本URL
$baseURL = '';
// 检查页面中是否包含 <base href=""/>
$nodeList = $xpath->query('//xhtml:base/@href');
if ($nodeList->length == 1) {
    $baseURL = $nodeList->item(0)->nodeValue;
}
// 如果没有 <base href=""/>, 那么在$url的基础上建立基本URL
else {
    $URLParts = parse_url($pageInfo ['url'] );
    if (! (isset($URLParts ['path'] ) && strlen($URLParts ['path'] ))) {
        $basePath = '';
    } else {
        $basePath = preg_replace('#/ [^/] *$#','',$URLParts ['path'] );
    }
    if (isset($URLParts ['username'] ) || isset($URLParts ['password'] )) {
        $auth = isset($URLParts ['username'] ) ? $URLParts ['username']  : '';
        $auth .= ':';
        $auth .= isset($URLParts ['password'] ) ? $URLParts ['password']  : '';
        $auth .= '@';
    } else {
        $auth = '';
    }
    $baseURL = $URLParts ['scheme']  . '://' .
               $auth . $URLParts ['host']  .
               $basePath;
}

// 跟踪我们访问过的链接以保证不会重复访问同一个链接
$seenLinks = array();

// 取得所有链接
$links = $xpath->query('//xhtml:a/@href');

foreach ($links as $node) {
    $link = $node->nodeValue;
    // 解决相对链接问题
    if (! preg_match('#^(http|https|mailto):#', $link)) {
        if (((strlen($link) == 0)) || ($link [0]  != '/')) {
            $link = '/' . $link;
        }
        $link = $baseURL . $link;
    }
    // 如果是我们已经访问过的链接，则跳过
    if (isset($seenLinks [$link] )) {
        continue;
    }
    // 对找到的链接给予标记
    $seenLinks [$link]  = true;
    // 输出访问过的链接
    print $link.': ';
    flush();
```

```
    list($linkHeaders, $linkInfo) = load_with_curl($link, 'HEAD');
    // 根据响应代码来判断该做什么
    // 2xx 的响应代码表示页面正常
    if (($linkInfo ['http_code']  >= 200) && ($linkInfo ['http_code']  < 300)) {
        $status = 'OK';
    }
    // 3xx 的响应代码表示页面重定向
    else if (($linkInfo ['http_code']  >= 300) && ($linkInfo ['http_code']  < 400))
{
        $status = 'MOVED';
        if (preg_match('/^Location: (.*)$/m',$linkHeaders,$match)) {
                $status .= ': ' . trim($match [1] );
        }
    }
    // 其他响应代码意味着错误
    else {
        $status = "ERROR: {$linkInfo ['http_code'] }";
    }
    // 输出与链接有关的已知信息
    print "$status\n";
}

function load_with_curl($url, $method = 'GET') {
    $c = curl_init($url);
    curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
    if ($method == 'GET') {
        curl_setopt($c,CURLOPT_FOLLOWLOCATION, true);
    }
    else if ($method == 'HEAD') {
        curl_setopt($c, CURLOPT_NOBODY, true);
        curl_setopt($c, CURLOPT_HEADER, true);
    }
    $response = curl_exec($c);
    return array($response, curl_getinfo($c));
}
?>
```

13.18 编程：查找新链接

例13-63中的程序为了生成链接及其最后更新时间的列表，在例13-62的基础上进行了修改。如果URL指向页面所在的服务器没有提供最后更新的时间信息，这个程序会把请求该URL的时间作为它的最后更新时间。如果程序找不到URL，则会输出在寻找URL时得到的状态码。在运行该程序时，需要传递一个要扫描其中链接的URL作为参数：

```
% php fresh-links.php http://www.oreilly.com
https://epoch.oreilly.com/account/default.orm: MOVED: https://epoch.oreilly.com/
lib/p_sso.orm?d=account
https://epoch.oreilly.com/shop/cart.orm: OK
http://www.oreilly.com/: OK; Last Modified: Mon, 08 May 2006 22:11:04 GMT
http://oreillynet.com/: OK
http://www.oreilly.com/store/: OK
```

```
http://safari.oreilly.com: OK
http://conferences.oreillynet.com/: OK
http://www.oreillylearning.com: OK
http://academic.oreilly.com: MOVED: http://academic.oreilly.com/index.csp
...
```

这里输出的是在GMT时间2006年5月8日晚上11时48分运行程序时得到的结果。其中多数链接都没有带最后更新时间，这表明服务器没有提供该信息，即页面有可能是动态的。链接*http://www.oreilly.com/*显示相应的页面是在90分钟之前更新的。而链接*http://academic.oreilly.com*显示了相应的页面已经被转移到其他地方，就和技巧13.17中*stale-links.php*程序的输出报告内容一样。

查找新链接的程序和查找失效链接的程序从原理上说几乎是一样的 —— 两者都使用了同样的技术从页面中提取链接，但查找新链接的程序使用了HTTP_Request类而没有使用cURL。而且把取得命令行中指定的基本URL的代码放在循环内部，可以保证跟踪到所提供的任何重定向信息，并且能够轻易地返回重定向链接中的最终URL地址。

当找到了对应的页面后，每个链接的URL都会通过head方法找到。对于转移的链接，这个程序不只是输出新的地址，而是在Last-Modified头部信息有效的情况下还会输出经过格式化的最后更新时间。

例13-63：fresh-links.php程序

```
<?php
error_reporting(E_ALL);
require_once 'HTTP/Request.php';

if (! isset($_SERVER ['argv']  [1] )) {
    die("No URL provided.\n");
}

$url = $_SERVER ['argv']  [1] ;

// 加载页面
$r = load_with_http_request($url);

if (! strlen($r->getResponseBody())) {
    die("No page retrieved from $url");
}

// 为了便于解析转换成XML
$opts = array('output-xhtml' => true,
              'numeric-entities' => true);
$xml = tidy_repair_string($r->getResponseBody(), $opts);
$doc = new DOMDocument();
$doc->loadXML($xml);
$xpath = new DOMXPath($doc);
$xpath->registerNamespace('xhtml','http://www.w3.org/1999/xhtml');
```

```
// 对相对链接计算基本URL
$baseURL = '';
// Check if there is a <base href=""/> in the page
$nodeList = $xpath->query('//xhtml:base/@href');
if ($nodeList->length == 1) {
    $baseURL = $nodeList->item(0)->nodeValue;
}
// 如果没有 <base href=""/>, 那么在$url的基础上建立基本URL
else {
    $URLParts = parse_url($r->_url->getURL());
    if (! (isset($URLParts ['path'] ) && strlen($URLParts ['path'] ))) {
        $basePath = '';
    } else {
        $basePath = preg_replace('#/ [^/] *$#','',$URLParts ['path'] );
    }
    if (isset($URLParts ['username'] ) || isset($URLParts ['password'] )) {
        $auth = isset($URLParts ['username'] ) ? $URLParts ['username']  : '';
        $auth .= ':';
        $auth .= isset($URLParts ['password'] ) ? $URLParts ['password']  : '';
        $auth .= '@';
    } else {
        $auth = '';
    }
    $baseURL = $URLParts ['scheme']  . '://' .
               $auth . $URLParts ['host']  .
               $basePath;
}

// 跟踪我们访问过的链接以保证不会重复访问同一个链接
$seenLinks = array();

// 取得所有链接
$links = $xpath->query('//xhtml:a/@href');

foreach ($links as $node) {
    $link = $node->nodeValue;
    // 解决相对链接问题
    if (! preg_match('#^(http|https|mailto):#', $link)) {
        if (((strlen($link) == 0)) || ($link [0]  != '/')) {
            $link = '/' . $link;
        }
        $link = $baseURL . $link;
    }
    // 如果是我们已经访问过的链接，则跳过
    if (isset($seenLinks [$link] )) {
        continue;
    }
    // 对找到的链接给予标记
    $seenLinks [$link]  = true;
    // 输出访问过的链接
    print $link.': ';
    flush();
```

```
    $r = load_with_http_request($link, 'HEAD');
    // 根据响应代码来判断该做什么
    // 2xx 的响应代码表示页面正常

    if (($r->getResponseCode() >= 200) && ($r->getResponseCode() < 300)) {
        $status = 'OK';
    }
    // 3xx 的响应代码表示页面重定向
    else if (($r->getResponseCode() >= 300) && ($r->getResponseCode() < 400)) {
        $status = 'MOVED';
        if (strlen($location = $r->getResponseHeader('location'))) {
            $status .= ": $location";
        }
    }
    // 其他响应代码意味着错误
    else {
        $status = "ERROR: {$r->getResponseCode()}";
    }
    if (strlen($lastModified = $r->getResponseHeader('last-modified'))) {
        $status .= "; Last Modified: $lastModified";
    }
    // 输出与链接有关的已知信息
    print "$status\n";
}

function load_with_http_request($url, $method = 'GET') {
    if ($method == 'GET') {
        $done = false; $max_redirects = 10;
        while ((! $done) && ($max_redirects > 0)) {
            $r = new HTTP_Request($url);
            $r->sendRequest();
            $responseCode = $r->getResponseCode();
            if (($responseCode >= 300) && ($responseCode < 400) &&
                strlen($location = $r->getResponseHeader('location'))) {
                    $url = $location;
                    $max_redirects--;
            } else {
                $done = true;
            }
        }
    } else {
        $r = new HTTP_Request($url);
        $r->setMethod(HTTP_REQUEST_METHOD_HEAD);
        $r->sendRequest();
    }
    return $r;
}
?>
```

第14章

消费Web服务

14.0 概述

Web 服务可以让你在HTTP协议的基础上通过XML来交换信息。如果你想知道纽约市的天气预报、IBM现在的股票报价，或者eBay上一台平板电视的价格，你可以编写一段简短的脚本，以某种你容易操作的格式来搜集这些信息。从一名开发人员角度看，这就如同你调用一个本地函数并返回一个值。

Web 服务背后的关键在于与平台无关的通讯。你运行在Linux上的PHP脚本可以与其他人运行在Window框架中IIS Server上的ASP脚本无障碍地对话。同样地，也可以通过同一组工具和接口，与运行Solaris，Apache和JSP的系统对话。

有两种主要的Web 服务 —— REST和SOAP。相对而言一个REST请求是很直观的，因为这涉及到一个针对服务器的HTTP请求和处理作为响应返回的XML文档的操作。由于多数开发人员都熟悉HTTP和XML，所以学习REST是一个不费时间的简单过程。

REST不足的一面是它超出HTTP与XML之外的部分没有太多可参照的标准，像数据应该如何传输或返回等。每个网站都可以随意使用自己认为最好的方法。虽然对于小型服务来说这不成问题，但如果设计不当，则可能会在服务规模增大时导致棘手的情况发生。

尽管如此，REST仍然是一种非常受欢迎的形式，其成功的关键因素就是简单。技巧14.1中介绍了如何发送REST请求的内容。

另一个流行的Web 服务形式是SOAP，它是W3C在跨网络信息传递和远程计算机函数调用方面的一个标准。SOAP为开发者提供了很多种选择，但是，这些选择也是以其复杂性为代价的。SOAP非常复杂，其完整的规范篇幅很长，而且内容仍然在增加。

在理想的情况下，SOAP可以使事情简单化。服务之间的通讯由客户端和服务器端自动

处理，可以把数据类型从一种语言序列化为另一种语言。因此，你可以反复地传递和取回复杂的数据结构而不必担心互操作性的问题。

如果真是这样，那就万幸了。然而，由于SOAP把抽象层建立在通讯的基础上，所以一旦碰到问题就很不容易排除。因为你不熟悉底层的XML，所以可能需要在诊断问题过程中投入不少的时间。

PHP 5中捆绑了一个SOAP扩展 —— ext/soap。目前，这个扩展实现了SOAP1.2的大部分功能，但并没有实现所有的功能。从整体上来看，这个实现不亚于甚至要好于PHP中其他的SOAP工具包，但一两处煞风景的地方还是有的。

从PHP 5.1开始，SOAP默认是启用的，对于早期的版本，可以通过在PHP配置行中添加--enable-soap来启用SOAP。你所需要的唯一一个外部的库是libxml2，它与任何PHP 5的XML扩展一样，也是一个必备的组件。

技巧14.2～14.11中介绍的都是与SOAP相关的内容。

有关SOAP的完整资料可以在W3C的网站中（*http://www.w3.org/2000/xp/Group/*）和James Snell, Doug Tidwell 以及Pavel Kulchenko合著的《Programming Web Services with SOAP》（O Reilly）中查到。

除了REST和SOAP之外，还有一种Web 服务形式也相当常见，即XML-RPC。XML-RPC类似于SOAP的精简版，因为它也会把本地数据转换为一种中性语言的格式，而你可以把转换后的数据传递到函数中并得到返回的结果。不过XML-RPC远不如SOAP那样复杂。

如果你需要的全部功能都可以由XML-RPC的特性组件来提供，可真是一件值得庆幸的事。但是，如果你试图摆脱其中某些功能的限制，问题就会出现。由于XML-RPC规范不能持续地更新，所以一旦出现的问题无法解决，你仍然需要切换回REST或SOAP。

技巧14.12和技巧14.13中是有关XML-RPC内容的。

14.1 调用REST方法

问题

你想要发送一个REST请求。

方案

使用file_get_contents()：

```
<?php
$base = 'http://music.example.org/search.php';
$params = array('composer' => 'beethoven',
                'instrument' => 'cello');

$url = $base . '?' . http_build_query($params);

$response = file_get_contents($url);
?>
```

或者使用cURL：

```
<?php
$base = 'http://music.example.org/search.php';
$params = array('composer' => 'beethoven',
                'instrument' => 'cello');

$url = $base . '?' . http_build_query($params);

$c = curl_init($url);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($c);
curl_close($c);
?>
```

讨论

REST是一种可以使用HTTP的get和post方法发送请求的Web 服务，而方法类型用于告知服务器要采取什么样的处理方式。例如，get告诉服务你想要获取现有的数据，而post则意味着你想更新现有的数据。而服务器会将响应结果以XML文档的形式发送给你进一步处理。

REST的优点在于它的简单以及使用了现有的标准。PHP支持发送HTTP请求和对XML文档的处理已经很多年了，所以发送并处理REST请求不过是穿新鞋走老路而已。

在PHP中，有多种方式可以执行HTTP请求，包括file_get_contents()方式、cURL扩展的方式以及PEAR包的方式。相关细节在第13章的开头部分都已经介绍过了。

当你获得了XML文档之后，可以使用任何PHP的XML扩展来处理它。基于REST文档的特性，加之你通常比较熟悉响应的模式，最佳的选择可能是SimpleXML扩展。技巧12.3介绍了这个扩展。然而，有时候你可能会想用其他的扩展，比如DOM、XMLReader甚至XSLT等等。有关这些扩展的介绍都包含在了第12章当中。

参见

第13章详细介绍了检索远程URL的过程；技巧15.1中包含了更多REST请求的例子。

14.2 通过WSDL调用SOAP的方法

问题

你想要发送一个SOAP请求。创建一个SOAP客户端可以让你从SOAP服务器端获得信息，而不用考虑是什么操作系统以及中间软件。

方案

使用ext/soap扩展。下面的代码是要查询当前的股票报价：

```
<?php
$wsdl_url =
    'http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl';

$client = new SOAPClient($wsdl_url);

$quote = $client->getQuote('EBAY'); // eBay, Inc.
print $quote;
?>
31.49
```

讨论

基于PHP的SOAP实现有很多。如果你使用的是PHP 5，我们推荐你使用它捆绑的ext/soap扩展。虽然这个扩展与PHP 4不兼容，但它具有许多超出PEAR::SOAP和NuSOAP（这两个重要的PHP SOAP扩展与PHP4兼容）的优势。ext/soap特别是：

- 用C编写的，而不是用PHP，所以它执行速度快、效率高；
- 从PHP 5开始与PHP捆绑，并从PHP 5.1开始是默认启用的；
- 与SOAP规范的许多部分保持一致；
- 利用了PHP 5的新特性，包含异常处理。

要发送SOAP请求，必须先技巧化一个新的SOAPClient对象，并向构造函数传递相应Web服务的WSDL地址：

```
$client = new SOAPClient('http://api.example.com/service.wsdl');
```

WSDL（Web Services Description Language，Web 服务描述语言）是一个XML词汇表，它可以让实现者创建一个定义他的Web服务都支持哪些方法和变量的文件。这个文件会放在Web服务器上供其他程序调用。

WSDL与人类并不十分友好，但它与计算机却能配合默契。当你让SOAP扩展指向一个WSDL文件时，这个扩展会自动地为相应的Web 服务创建一个对象，而你可以像操作PHP的类一样来操作这个对象。

这个对象甚至能够知道每个方法都带什么参数，以及每个参数的类型。之所以这一点很重要，是因为SOAP与PHP不同，它是严格类型的。因此，你不能在SOAP期望一个整数1时为它传递一个字符串1。WSDL允许SOAP扩展把PHP变量强制转换成适当的类型，而不用人为干预。

为此，只要可能，你都应该知道服务器端WSDL文件的位置。因为这可以使进行SOAP请求更容易。例14-1显示如何通过WSDL来完成一个查询。

例14-1：使用WSDL的SOAP客户端

```
<?php
$wsdl_url = 'http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl';

$client = new SOAPClient($wsdl_url);

$quote = $client->getQuote('EBAY'); // eBay公司.
print $quote;
?>
31.49
```

从XMethods的网站上，你知道了提供这种服务的WSDL文件位置是：*http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl*。

现在，你把`$wsdl_url`（WSDL文件的位置）传递到构造函数，以技巧化一个新的SOAPClient对象。这个构造函数返回一个客户端对象 —— `$client`，你可以用它来实现SOAP请求。

构造函数创建了一个SOAP客户端，但仍然需要你自己来定义查询，即调用`getQuote()`方法。这个方法需要一个参数 —— 股票代码。直接把要查询的股票代码（本例中的EBAY）传递到这个方法中。

当你调用的是`$client->getQuote(10001)`时，SOAP扩展会把这个PHP字符串EBAY转换成XML格式的SOAP信息，并向XMethodes服务器发送一个HTTP请求。在XMethods收到并处理完你的查询以后，它会以自己的格式返回一个SOAP信息。SOAP扩展会监听到

服务器的响应并把XML信息解析为一个PHP对象，这个对象由调用的方法返回并保存在$quote中。

现在$quote变量保存着EBAY股票当前的市价。也就是当前每股31.49美元的成交价。

参见

技巧14.3介绍了不通过WSDL实现SOAP请求的例子；技巧15.2介绍了更多的SOAP服务器；ext/soap文档（*http://www.php.net/soap*）；James Snell, Doug Tidwell 以及Pavel Kulchenko合著的《Programming Web Services with SOAP》（O Reilly）。

14.3 不通过WSDL调用SOAP的方法

问题

你想要向一个没有提供WSDL文件的服务发送一个SOAP请求，所以你必须自己指定必要的信息。

方案

向WSDL文件参数的位置传递一个null值，而主要的服务设置（如位置及名称空间URI）则以一个选项数组提供：

```
<?php
$opts = array('location' => 'http://64.124.140.30:9090/soap',
              'uri'      => 'urn:xmethods-delayed-quotes',
$client = new SOAPClient(null, $opts);
?>
```

通过__soapCall()方法来实现请求，把方法名作为第一个参数，把方法变量数组作为第二个参数：

```
$quote = $client->__soapCall('getQuote', array('EBAY')); // eBay公司。

print $quote;
31.49
```

讨论

由于你没有使用WSDL，所以要向SOAPClient传递一个null作为第一个参数，这样就相当于告诉SOAP扩展，你会在第二个选项参数中传递有关Web服务的详细设置信息。

这个设置必须保存为一个数组。最低限度，你也必须提供两个元素 —— 表示SOAP服务器所在位置的URL和标识服务的名称空间URI。例如：

```
<?php
$opts = array('location' => 'http://64.124.140.30:9090/soap',
              'uri'      => 'urn:xmethods-delayed-quotes',
$client = new SOAPClient(null, $opts);
?>
```

服务器的URL是loaction元素，即例子中的*http://64.124.140.30:9090/soap*。而服务器的名称空间用uri元素来设置，即urn:xmethods-delayed-quotes。

现在你同样获得了一个客户端，不过是一个不基于WSDL的客户端，所以你不能用$client方法直接调用SOAP方法。相反，你要引用__soapCall()方法，把方法名作为它的第一个参数，把变量数组作为它的第二个参数：

```
<?php
$quote = $client->__soapCall('getQuote', array('EBAY')); // eBay公司.

print $quote;
?>
31.49
```

由于这里的SOAP客户端无法知道要传递多少个参数，所以你必须把自己的参数绑定为一个数组传递到__soapCall()中。因此，这里的股票报价以array('EBAY')的形式传递，而不是以'EBAY'的形式。

这里的代码比使用WSDL的方案更复杂一些，而且它还利用了SOAPClient假定的一些SOAP的默认设置。调用SOAP方法的接口也不够优雅。

但是，这却是传递或读取额外信息（如SOAP头部信息）的唯一方法。

参见

技巧14.2介绍了如何通过WSDL实现SOAP请求；技巧15.2介绍了更多的SOAP服务器；ext/soap文档（*http://www.php.net/soap*）；James Snell, Doug Tidwell 以及Pavel Kulchenko合著的《Programming Web Services with SOAP》（O Reilly）。

14.4 调试SOAP请求

问题

你的SOAP请求没有正常工作，而你想找到原因所在。

方案

在创建SOAPCient对象时启用跟踪选项：

```
<?php
$opts = array('trace' => true);
$client = new SOAPClient($wsdl_url, $opts);
?>
```

之后，你就可以访问通过报文（wire）发送的数据了：

```
<?php
$response = $client->getQuote('EBAY');

// 发送
print $client->__getRequestHeaders() . "\n";
print $client->__getRequest() . "\n";

// 返回
print $client->__getReponseHeaders() . "\n";
print $client->__getRequest() . "\n";
?>
```

讨论

调试SOAP请求是很困难的。在没有其他办法的情况下，就必须实实在在地对往往返返传递的实际HTTP和XML数据进行检查。

当数据是通过SSL保护或者你无权控制服务器的时候，这就需要一些技巧了。在这种情况下，最简单的办法就是求助于ext/soap扩展对发送和接受的所有数据给你一份完整的报告清单。

首先，必须启用跟踪选项，这样就可以让该扩展保存相应的信息以备后需：

```
<?php
$opts = array('trace' => true);
$client = new SOAPClient($wsdl_url, $opts);
?>
```

然后，无论你什么时候发送请求，最近的请求数据都能通过下列四个函数访问到 —— 其中两个处理PHP外发的请求，另外两个处理来自服务器的响应。

```
<?php
$response = $client->getQuote('EBAY');

// 发送
print $client->__getRequestHeaders() . "\n";
print $client->__getRequest() . "\n";
```

```
// 返回
print $client->__getReponseHeaders() . "\n";
print $client->__getRequest() . "\n";
?>
```


现在，你就可以查看数据找一找出现问题的原因何在了。通常，SOAP封套中会包含错误的XML信息。这些信息格式良好，而且不仅仅包含了SOAP服务器想要的信息。

在这个层次上，如果能有一些你知道有效的SOAP请求和应答的例子会对查错非常有帮助。在这种情况下，你就可以尝试一步一步地通过ext/soap扩展来重建这些请求，以便找到问题所在。

在这个层次上进行调试，可能会需要对HTTP、XML、XML名称空间、XML模式（Schema）、SOAP以及WSDL都有一定程度的理解。特别是，最后你可能会得到两个XML文档，这两个文档具有相同的语义模式，但由于XML名称空间、前缀和默认名称空间不同会导致外观形式不一样。如果你能够完全肯定这两个文档类似，但是其中一个却不能正常使用，可能就需要通过cURL把这两个不同的XML文件明确地发送到服务器上面了。

14.5 使用复杂的SOAP类型

问题

你需要传递比字符串、整数以及其他简单类型更复杂的数据结构。比如，要传递数组和对象。

方案

将数据作为关联数组传递：

```
$args = array('ticker' => array('EBAY', 'YHOO', 'GOOG'));

$client->getQuotes($args);
```

讨论

要想领会到如何把PHP数据结构映射为WSDL中所描述的XML文件并不是一件容易做到的事，特别是在SOAP作为SOAP服务器的一个普通消费者而不会关心底层的XML的前提下。

虽然ext/soap扩展在适当地转换数据方面做得非常不错，但有时你仍需要放下身段、染指其中，自己亲自查看一番WSDL文件或者看看服务所发布的一些表现为XML的SOAP例子是否能够满足需要。

经验表明，如果你想在同一个层次中传递的数据是同名多值的情况时，比如：

```
<ticker>EBAY</ticker>
<ticker>YHOO</ticker>
<ticker>GOOG</ticker>
```

你应该定义一个数组，其键为ticker而值则是另一个数组 —— 这个数组应该包含被<ticker>标签包围的那些数据。

参见

技巧14.6中有关设置SOAP类型的讨论。

14.6 设置SOAP类型

问题

你想要明确地设置XML模式类型，但是通过常规的PHP数据结构无法告知ext/soap如何设置该类型值。

方案

创建一个SOAP对象并给其构造器传递要设置的类型值和名称空间：

```
$ns = 'https://adwords.google.com/api/adwords/v2';
$job = new SOAPVar($data, SOAP_ENC_OBJECT, 'CustomReportJob', $ns);
$response = $client->scheduleReportJob(array('job' => $job));
```

这样所创建的XML将会如下面这样：

```
<ns1:job xsi:type="ns1:CustomReportJob">
...
</ns1:job>
```

其中ns1名称空间的的前缀是https://adwords.google.com/api/adwords/v2。

讨论

SOAP扩展中包含了许多有助于我们创建向SOAP客户端传递数据的类。这些类在使用WSDL文件时几乎是用不到的。但在某些时候则又是必须使用的。

一种情况就是当你必须要设置XML模式类型属性的时候。SOAP将XML模式作为其后台对XML数据进行编码的方式。而XML模式的一个作用就是定义结构类型的能力。在正常情况下，这些类型都是内置的XML模式类型中的一个，例如字符串、整数或对象。不过，你也能通过扩展XML模式来创建自定义的类型。

当某个服务需要一个自定义类型时，必须使用SOAPVar类来设置这个类型。SOAPVar类接受六个参数。这里最重要的是前四个，它们分别是要发送的数据、XML模式类型的一般类、指定给xsi:type属性值的名称和该值所处的名称空间。

例如，下面的这段代码针对Google Adwords报告Web 服务创建了一个自定义的CustormReportJob类型：

```
$ns = 'https://adwords.google.com/api/adwords/v2';
$job = new SOAPVar($data, SOAP_ENC_OBJECT, 'CustomReportJob', $ns);
$response = $client->scheduleReportJob(array('job' => $job));
```

你不用关心到底向Web服务器中发送了什么信息（保存在$data中），或者什么对象（作为第二个参数的SOAP_ENC_OBJECT）。虽然这也很重要，但与XML模式类型是无关的。

有关系的是SOAPVar构造器的第三和第四个参数：'CustomReportJob'和$ns。

在这个例子当中，你所创建的是位于XML名称空间https://adwords.google.com/api/adwords/v2（$ns的值）之下的CustomReportJob类型的工作。

这样，当ext/soap将数据序列化为XML时，就会像下面所示的这样添加上必要的属性：

```
<ns1:job xsi:type="ns1:CustomReportJob">
...
</ns1:job>
```

根据代码中使用的XML名称空间的数量，可能会也可能不会有相同的ns1前缀。这个特殊的前缀字符串不是问题，关键是这个前缀必须在SOAP-Envelope元素中映射到https://adwords.google.com/api/adwords/v2上面，就像下面这样：

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="https://adwords.google.com/api/adwords/v2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

参见

技巧14.5中使用复杂的SOAP类型的讨论。

14.7 使用SOAP头部

问题

你需要创建一个SOAP头部信息并将其放在请求中一起发送。这种情况通常发生在某个服务要求验证认证证书或其他与请求无直接关系的信息的时候。

方案

使用SOAPHeader类创建头部信息。

要在发送服务的所有请求中都使用相同的头部信息，调用__setSoapHeaders()：

```
$client = new SOAPClient('http://www.example.com/service.wsdl');

$username = new SOAPHeader('urn:service-namespace', 'Username', 'elvis');
$password = new SOAPHeader('urn:service-namespace', 'Password', 'the-king');

$headers = array($username, $password);

$client->__setSoapHeaders($headers);
```

也可以根据每次调用原则将头部信息作为第四个参数传递给__soapCall()：

```
$client = new SOAPClient('http://www.example.com/service.wsdl');

$username = new SOAPHeader('urn:service-namespace', 'Username', 'elvis');
$password = new SOAPHeader('urn:service-namespace', 'Password', 'the-king');

$headers = array($username, $password);

$client->__soapCall($function, $args, $options, $headers);
```

这样会生成下面的XML：

```
<SOAP-ENV:Header>
    <ns2:Username>elvis</ns2:Username>
    <ns2:Password>the-king</ns2:Password>
</SOAP-ENV:Header>
```

名称空间前缀可以改变，但也会被映射到urn:service-namespace名称空间URI上。

讨论

SOAP封套元素被分成两部分，一部分是SOAP头部信息，另一部分是SOAP主体信息。这种划分与HTTP中的头部与主体的划分类似。多数情况下，我们只需要访问主体信息，而在某些情况下也会用到头部信息。

ext/soap扩展在简化传入部分SOAP主体信息方面是卓有成效的。然而，在需要用它来支持你创建SOAP头部信息时，问题就没有那么简单了。

取决于SOAP头部信息的设计，创建所需头部信息的难点也表现各异。如果你所需要的只是被元素包装的数据，那么可以创建SOAPHeader对象并将其打包到一个数组中：

```
$client = new SOAPClient('http://www.example.com/service.wsdl');

$username = new SOAPHeader('urn:service-namespace', 'Username', 'elivs');
$password = new SOAPHeader('urn:service-namespace', 'Password', 'the-king');

$headers = array($username, $password);
```

这些头部信息随后会被添加到源自一个特定的SOAP客户端技巧的所有请求中，或者按照每次调用的原则：

```
$client = new SOAPClient('http://www.example.com/service.wsdl');

// 使用 __setSoapHeaders()把头部信息添加到 *所有* 请求中
$client->__setSoapHeaders($headers);

// 或者根据每次调用原则行事
$client->__soapCall($function, $args, $options, $headers);
```

这样做的结果是在请求中添加了下列XML：

```
<SOAP-ENV:Header>
    <ns2:Username>elivs</ns2:Username>
    <ns2:Password>the-king</ns2:Password>
</SOAP-ENV:Header>
```

从你的角度说，其中的名称空间前缀可以不用ns2，但无论用什么都会被映射到urn:service-namespace名称空间URI上面。

参见

技巧15.6中有关处理SOAP头部信息的讨论和技巧15.7中有关从SOAP服务器发送SOAP头部信息的内容。

14.8 通过SOAP实现认证

问题

你想使用HTTP基本认证方式来对SOAP请求进行认证。

方案

通过设置login和password键以用选项数组来传递用户名和密码：

```
$options = array('login' => 'elvis',
                 'password' => 'the-king');

$client = new SOAPClient('http://www.example.com/service.wsdl',
                         $options);
```

讨论

有很多种在SOAP请求中处理认证的方式。其中一些流行的方式是HTTP基本认证，即把数据放到SOAP头部，再通过WS-*规范进行验证，也包含WS-Security。

若要使用ext/soap在每个请求中都添加HTTP基本认证证书信息，需要在创建SOAP客户端对象时通过选项数组来传递用户名和密码：

```
<?php
$options = array('login' => 'elvis',
                 'password' => 'the-king');

$client = new SOAPClient('http://www.example.com/service.wsdl',
                         $options);
?>
```

现在每个请求都将包含一个额外的HTTP Authorization头部信息。

如果你的服务需要SOAP请求头部的数据集，可以参考技巧26.1中有关处理这一问题的详细介绍。

在写作本书时，还不能针对ext/soap使用任何XML安全规范。不过，Rob Richards正在致力解决这个问题。要了解更详细的信息，可以访问*http://www.cdatazone.org/index.php?/archives/9-WSSE-and-extsoap.html*。

参见

技巧15.8中有关在SOAP服务器端处理SOAP认证的信息。

14.9 重新定义终点

问题

WSDL文件定义了服务的终点，但你需要将它改为另一个URL。这种情况发生在服务中包含测试和产品站点或者要求你在URL中传递额外查询参数的时候。

方案

如果你想为所有请求都设置相同的终点，可以在选项数组中为构造器指定一个新位置：

```
<?php
$options = array('location' => 'http://www.example.com/testing-endpoint');

$client = new SOAPClient('http://www.example.com/service.wsdl',
                         $options);
?>
```

如果每个请求的终点都不一样，那么可以使用__soapCall()方法分别传递每个请求的位置：

```
<?php
$client = new SOAPClient('http://www.example.com/service.wsdl');

$method = 'getTemp';
$args = array('94114');
$options = array('location' => 'http://www.example.com/endpoint?method=getTemp'
);

$request = $client->__soapCall($method, $args, $options);
?>
```

讨论

在多数情况下，是不用修改WSDL文件中设置的终点的。多数服务中都具有单一的固定终点，在将其放到WSDL文件中后你只管使用就可以了。

然而，有些站点可能会根据一些条件要求你修改终点的位置。例如，服务中可能既包含测试站点也包含产品站点。这对于只读的服务而言不是问题，但是如果当你对该站点同时拥有读写权限时，用一个沙箱环境来测试代码就很重要了。

为了简单起见，有的站点可能只发布了一个单独的WSDL文件，而在默认的情况下只指向产品服务器，但是要求你在开发期间将该文件转换为指向测试服务器。在这种情况下，最简单的方法就是将新位置作为一个一次性配置选项指定给SOAPClient构造器：

```
<?php
$options = array('location' => 'http://www.example.com/testing-endpoint');

$client = new SOAPClient('http://www.example.com/service.wsdl',
                         $options);
?>
```

SOAPClient构造器接受一个选项数组作为其第二个参数。当你在该数组中设置了location元素时，就会覆盖WSDL中的位置并将相应的URL用作所有请求的目标位置。

这个方法对于为所有请求都指定相同的URL的情况非常有效。不过，有时候可能需要为不同的请求设置不同的终点。例如，你可能需要在URL中放入方法名或者其他信息。

由于把数据放到URL中可以使服务不用解析XML文档就能够对请求进行分发，因而能够提高发送请求的效率。比如说，一个大型的Web服务可能使用一个计算机池处理搜索，使用另一个计算机池来处理更新。如果它能够通过简单地检测URL就可以直接把SOAP请求分发到适当的池上面去，那么就可以提高处理请求的速度。

当然，这就需要你按照一次性原则来修改每一个请求。因此，不能再使用SOAPClient的方法进行抽象地覆盖了。相反，需要直接使用__soapCall()方法传递方法名、参数和选项：

```
<?php
$client = new SOAPClient('http://www.example.com/service.wsdl');

$method = 'getTemp';
$args = array('94114');
$options = array('location' => 'http://www.example.com/endpoint?method=getTemp'
);

$request = $client->__soapCall($method, $args, $options);
?>
```

这些代码等价于：

```
<?php
$client = new SOAPClient('http://www.example.com/service.wsdl');

$request = $client->getTemp('94114');
?>
```

但是，这也会把终点URL改为指向*http://www.example.com/endpoint?method=getTemp*，而非WSDL中默认的设定值。

为了保持简单调用的惯例，可以针对你的服务子类化SOAPClient并提供一个自定义的__call()方法：

```
<?php
class TemperatureService extends SOAPClient {
    public function __call($method, $args) {

        // 修改终点使其包含方法名
        // 采取一致的命名约定
        $location = "http://www.example.com/endpoint?method={$method}";
        $options = array('location' => $location);

        return $this->__soapCall($function, $args, array('location' => $location));
    }
}

$client = new TemperatureService('http://www.example.com/service.wsdl');

$request = $client->getTemp('94114');
?>
```

14.10 捕捉SOAP故障

问题

你想要处理SOAP服务器以SOAP故障的形式返回的一个错误。这样就可以在请求或者服务存在问题时适当地进行舍弃处理。

方案

将代码块包装在try/catch结构中，检查SOAPFault：

```
<?php
try {
    $wsdl_url ='http://www.example.com/TemperatureService.wsdl';

    $client = new SOAPClient($wsdl_url);

    $temp = $client->getTemp('New York'); // 这应该是一个邮政编码
    print $temp;
} catch (SOAPFault $exception) {
    print $exception;
}
?>
```

或者将SOAPClient配置为不使用异常，并检查is_soap_fault()的返回值：

```
<?php
$wsdl_url ='http://www.example.com/TemperatureService.wsdl';

// 禁用异常
$opts = array('exceptions' => 0);
$client = new SOAPClient($wsdl_url, $opts);

$temp = $client->getTemp('New York'); // 这应该是一个邮政编码
if (is_soap_fault($temp)) {
    print $exception;
} else {
    print $temp;
}
?>
```

讨论

当SOAP服务器产生错误时，会返回一个SOAP故障。其原因可能在你这一边，例如调用一个不存在的方法或者传递了不正确的参数数值（或类型）。也有可能是服务器端的错误，例如，服务中缺少与特殊的邮政编码对应的气温信息，因而不能满足SOAP请求的需要。

SOAP扩展会将SOAP故障转换为PHP异常，如例14-2所示。

例14-2：通过异常处理机制检测SOAP故障

```
<?php
try {
    $wsdl_url ='http://www.example.com/TemperatureService.wsdl';

    $client = new SOAPClient($wsdl_url);

    $temp = $client->getTemp('New York'); // 这应该是一个邮政编码
    print $temp;
} catch (SOAPFault $exception) {
    print $exception;
}
?>
SOAPFault exception: [SOAP-ENV:Server] Zip Code New York is unknown.

  in /www/www.example.com/soap.php:8
Stack trace:
#0 /www/www.example.com/soap.php(8): SOAPClient->getTemp('getTemp', Array)
#1 {main}
```

由于服务器需要的是邮政编码而例14-2中传递的是New York，所以服务器返回了一个SOAP故障。对应的异常信息“Zip Code New York is unknown”会与其他调试信息一同被输出。

如果你不喜欢异常，还可以通过把异常配置设置为0来让SOAP根据返回的代码来处理故障。这个过程以例14-3示范。

例14-3：不能过异常机制检测SOAP故障

```
<?php
$wsdl_url ='http://www.example.com/TemperatureService.wsdl';

// 禁用异常
$opts = array('exceptions' => 0);
$client = new SOAPClient($wsdl_url, $opts);

$temp = $client->&gt;getTemp('New York');// 这应该是一个邮政编码
if (is_soap_fault($temp)) {
    print $exception;
} else {
    print $temp;
}
?>
SOAPFault exception: [SOAP-ENV:Server] Zip Code New York is unknown.
  in /www/www.example.com/soap.php:8
#0 {main}
```

要修改SOAPClient对象的默认设置，可以将一个数组作为第二个参数传递给其构造器。这个数组与用来指定非WSDL服务器信息的数组是同一个数组。

在禁用异常后，$temp中不是包含了有效的响应就是包含着一个SOAP故障。通过检查is_soap_fault()可以发现其中包含的是不是错误。

参见

技巧15.5中有关从SOAP服务器中抛出SOAP故障的内容。

14.11 将XML模式数据类型映射为PHP类

问题

你想自动地把一个SOAP对象转换成一个PHP对象。

方案

定义一个PHP类并使用classmap选项告知SOAPClient把一个SOAP对象映射到这个类上面：

```
class PHPStockType {}; // 占位类
```

```
$opts = array('classmap' => array('StockType' => 'PHPStockType'));
$client = new SOAPClient($wsdl_url, $opts);
```

现在，任何StockType结果都会被转换为PHPStockType对象。

讨论

使用类映射对于简化SOAP对象的使用非常有帮助。尤其是，可以在你认为合适的地方定义__toString()方法以便控制对象的输出显示。而且对于实现IteratorAggregate和ArrayAccess接口同样也很有用。

例如，一个股票报价对象有可能返回大量有关股票的数据：当前价格、52周的最高和高低价、股权信息等等。但是，其中关键的数据是当前价格，所以你可能想要实现如下代码：

```
<?php
class PHPStockType {
    public function __toString() {
        return (string) $this->currentPrice;
    }
}
?>
```

其他的数据当然还是有效的，只不过输出对象时，你选择的是其中最重要的数据。

14.12 调用XML-RPC方法

问题

你想为一个XML-RPC客户端生成对服务器的请求。XML-PRC可以使PHP对服务器进行函数调用，即使该服务器并没有使用PHP。所得到的数据会自动转换成你的应用程序中可以使用的PHP变量。

方案

使用PHP的XML-RPC扩展以及一些辅助函数。PHP中捆绑了*xmlrpc-epi*扩展，遗憾的是，xmlrpc-epi中没有任何本地C函数可以接受XML-RPC格式的字符串并生成请求。但是，xmlrpc-epi背后的开发人员开发了一系列以PHP形式编写的辅助函数，可以从*http://xmlrpc-epi.sourceforge.net/*下载到。这里要使用的唯一一个文件就是*utils.php*，

其位置是*sapmle/utils*。要安装这些函数，只需把相应的文件拷贝到PHP能够找得到的include_path所指向的位置即可。

下面的客户端代码调用了一个XML-RPC服务器端的函数并返回了州名：

```
<?php
// 这是包中默认的文件名
// 保持不变是为了避免混淆
require 'utils.php';

// 服务器端设置
$host = 'betty.userland.com';
$port = 80 ;
$uri = '/RPC2';

// 请求设置
// 传递一个1到50的数，得到按字母顺序排列的第n个州名
// 1 是 Alabama, 50 是Wyoming
$method = 'examples.getStateName';
$args = array(32); // 传递的数据

// 根据这些变量拼成关联数组
$request = compact('host', 'port', 'uri', 'method', 'args');

// 生成XML-PRC请求的函数
$result = xu_rpc_http_concise($request);

print "I love $result!\n";
?>
```

讨论

XML-PRC是一种由Userland软件公司创立的格式，可以允许我们通过HTTP生成一个对Web服务器的请求，请求自身是一个特殊格式的XML文档。作为客户端，需要建立并发送一个符合XML-PRC规范的XML请求。在发送请求之后，服务器也会以XML文档进行响应。最后，可以通过解析响应的XML得到需要的结果。在本解决方案中，XML-RPC服务器返回了一个州名，因而代码输出的结果为：

```
I love New York!
```

与XML-RPC早期的实现（那是用PHP编写的）不同，当前捆绑的版本是用C编写的，所以在处理速度上有显著的提升。要在配置PHP时启用这一扩展，可以添加`--with-xmlrpc`。

服务器端的设置是告诉PHP所接触的哪一个站点生成请求。`$host`是机器的主机名，`$port`是Web服务器使用的端口号，`$uri`是你希望连接的XML-RPC服务器的路径名。这

个请求等价于*http://betty.userland.com:80/RPC2*。如果没有给定端口号，该函数会默认使用80端口，而默认的URI是Web服务器的根目录——/。

请求设置了要调用的函数和要传递给这个函数的数据。方法example.getStateName接受从1到50的整数并按字母顺序返回美国州名的字符串。在XML-RPC中，方法名可以带句点，而在PHP中这是不合法的。在可能的情况下，PHP等价地传递了32作为发送到XML-RPC的请求参数，而调用examples.getStateName就是调用名为examples.getStateName()的函数：

```
examples.getStateName(32);
```

在XML-RPC中，就像下面这样：

```
<?xml version='1.0' encoding="iso-8859-1" ?>
<methodCall>
<methodName>examples.getStateName</methodName>
<params><param><value>
   <int>32</int>
  </value>
 </param>
</params>
</methodCall>
```

服务器端的设置和请求信息被放到了一个单独的关联数组中，并被传递给xu_rpc_http_concise()。为了简化操作，调用compact()的过程就相当于使用了以下代码：

```
$request = array('host'   => $host,
                 'port'   => $port,
                 'uri'     => $uri,
                 'method' => $method,
                 'args'   => $args);
```

xu_rpc_http_concise()函数执行XML-RPC调用并返回结果。因为返回的值是字符串，所以可以直接输出$results。如果XML-RPC调用返回了多个值，xu_rpc_http_concise()就会返回一个数组。

可以给xu_rpc_http_concise()传递10个参数，但只有一个host是必须的。这10个参数如表14-1所示。

表14-1：xu_rpc_http_concise()可用的参数

名称	描述
host	服务器主机名。
uri	服务器URI（默认为/）。

表14-1：xu_rpc_http_concise()可用的参数（续）

名称	描述
port	服务器端口（默认为80）。
method	调用的方法名。
args	传递给方法的参数。
debug	调试级别（0到2。0为none，2为多）。
timeout	请求超时的秒数。值为0意味着永不超时。
user	Basic HTTP认证的用户名——必要情况下。
pass	Basic HTTP认证的密码——必要情况下。
secure	使用SSL进行加密传输。要求PHP必须建立对SSL的支持（传递任何可以转换为ture的值）。

参见

技巧15.9中有关于XML-RPC服务器的更多信息。为使用*xmlrpc-epi*扩展所提供的PHP辅助函数（*http://xmlrpc-epi.sourceforge.net/*）。Simon St.Laurent, Joe Johnston和Edd Dumbill合著的“Programming Web Services with XML-RPC”（O'Reilly）。有关XML-RPC的更多信息（*http://www.xml-rpc.com*）。

14.13 通过XML-RPC实现验证

问题

你需要随同你的XML-RPC请求一起发送用户名和密码以进行验证。

方案

设置user和pass选项并调用xu_rpc_http_concise()方法：

```
<?php
require 'utils.php';

// ... 在这里设置其他请求参数
$user = 'elvis';
$pass = 'the-king';

// 根据这些变量拼成关联数组
$request = compact('host', 'port', 'uri', 'method', 'args', 'user', 'pass');
```

```
// 生成XML-PRC请求的函数
$result = xu_rpc_http_concise($request);
?>
```


讨论

令人意外的是，XML-RPC库不支持HTTP基本认证。但是，技巧14.12中提到的*utils.php*文件能在你将user和pass传递到xu_rpc_http_concise()时，为你创建正确的HTTP头部信息。例如：

```
<?php
require 'utils.php';

// ... 在这里设置其他请求参数
$user = 'elvis';
$pass = 'the-king';

// 根据这些变量拼成关联数组
$request = compact('host', 'port', 'uri', 'method', 'args', 'user', 'pass');

// 生成XML-PRC请求的函数
$result = xu_rpc_http_concise($request);
?>
```

在上面的代码中，user是elvis而pass是the-king。这些变量会跟其他必须的请求数据一同被转换成一个关联数组$request（为了明确起见省略了其他数据）。

当把这一信息传递给xu_rpc_http_concise()时，辅助函数将会对其进行Base64编码并为你构造相应的头部信息。

假设你的认证证书是正确的，那么其余的处理步骤事实上应该与非认证请求下的操作相同。

第15章

建立Web服务

15.0 概述

本章介绍构建Web服务的内容。如果你还不熟悉Web服务的基本概念，包括REST、SOAP和XML-PRC，请查看第14章，其中提供了本章所介绍的Web服务服务器端的构造块。

技巧15.1介绍了建立REST方法的过程。通过REST服务器，我们可以接受一个HTTP请求，处理接收到的数据并返回结果（通常是XML形式）。

接着，本章介绍了SOAP。技巧15.2和15.3展示了如何在带有和不带输入参数的条件下提供SOAP方法。

技巧15.4一分为二地解释了PHP为什么不能通过PHP类自动地生成WSDL文件，而技巧15.5讨论了如何抛出SOAP故障。

SOAP头部信息是接下来两个技巧的主题。首先，在技巧15.6中可以学习到如何处理SOAP头部信息。然后，技巧15.7中又介绍了如何生成SOAP头部信息。

有关SOAP主题的介绍在技巧15.8中讨论完如何通过SOAP进行身份认证后结束。

本章最后一个技巧是15.9，介绍了提供XML-RPC请求的方法。

15.1 以REST方法提供服务

问题

你想通过REST暴露一个服务。该服务可以接受其他人发送的HTTP请求并以XML进行响应。

方案

最基本的REST服务器是一个能够接收查询参数并返回XML的页面：

```
<?php
// 数据
$music_database = <<<_MUSIC_
<?xml version="1.0" encoding="utf-8" ?>
<music>
    <album id="1">
        <name>Revolver</name>
        <artist>The Beatles</artist>
    </album>
    <!-- 这里是其余941个唱片 -->
    <album id="943">
        <name>Miles And Coltrane</name>
        <artist>Miles Davis</artist>
        <artist>John Coltrane</artist>
    </album>
</music>
_MUSIC_;

// 加载数据
$s = simplexml_load_string($music_database);

// 查询数据
$artist = addslashes($_GET ['artist'] );
$query = "/music/album [artist = '$artist'] ";
$albums = $s->xpath($query);

// 以XML显示查询结果
print "<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n";
print "<music>\n\t";
foreach ($albums as $a) {
    print $a->asXML();
}
print "\n</music>";
?>
```

如果把这个页面保存在*http://api.example.org/music*，那么一个HTTP GET请求*http://api.example.org/music?artist=The+Beatles*将会返回：

```
<?xml version="1.0" encoding="utf-8" ?>
<music>
    <album id="1">
        <name>Revolver</name>
        <artist>The Beatles</artist>
    </album>
</music>
```

讨论

在最基本的层次上，提供一个REST请求同处理一个HTML表单没有什么不同，关键的差别是以XML而不是HTML来应签。

输入参数作为查询参数到来，所以PHP要将其解析入$_GET中。然后，再根据$_GET中的值来确定针对数据的正确查询，通过该查询可以抽取适当的记录以便返回。

例如，例15-1使用XPath从XML文档查询出了由变量artist get传递进来的艺术家所出版的全部唱片。

例15-1：实现REST查询服务

```
<?php
// 数据
$music_database = <<<_MUSIC_
<?xml version="1.0" encoding="utf-8" ?>
<music>
    <album id="1">
        <name>Revolver</name>
        <artist>The Beatles</artist>
    </album>
    <!-- 941 more albums here -->
    <album id="943">
        <name>Miles And Coltrane</name>
        <artist>Miles Davis</artist>
        <artist>John Coltrane</artist>
    </album>
</music>
_MUSIC_;

// 加载数据
$s = simplexml_load_string($music_database);

// 查询数据
$artist = addslashes($_GET ['artist'] );
$query = "/music/album [artist = '$artist'] ";
$albums = $s->xpath($query);

// 以XML显示查询结果
print "<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n";
print "<music>\n\t";
foreach ($albums as $a) {
    print $a->asXML();
}
print "\n</music>";
?>
```

为了简单起见，例15-1中使用XML作为数据源，并使用XPath作为查询语言。这样就消除了把结果转换为XML的需求。或许你将会在此使用SQL来查询数据库，没问题！从REST的宗旨来看，特定的后台系统是无关的。

关键的地方在于将结果以XML形式输出。在这个例子中，因为我们操纵的数据本身就是XML，因而可以将其包装在根元素中且不经过任何转换直接返回：

```
<?php
// 以XML显示查询结果
print "<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n";
print "<music>\n\t";
foreach ($albums as $a) {
    print $a->asXML();
}
print "\n</music>";
?>
```

你所得到的结果将是：

```
<?xml version="1.0" encoding="utf-8" ?>
<music>
    <album id="1">
        <name>Revolver</name>
        <artist>The Beatles</artist>
    </album>
</music>
```

现在你的工作完成了，剩下的事情就是REST客户端使用它所选择的XML处理工具来处理返回的XML。在PHP 5中，处理XML的工具通常会使用SimpleXML。

公布REST响应的数据模式也是很有用的。这样就可以让人们知道你所返回的数据结构，并让他们对数据进行验证以确保格式良好。XML Schema和RelaxNG就是两种非常好的模式。

REST不会限于只读的操作，例如搜索。REST还支持读写数据，包括添加、更新和删除记录。

有两种流行的方式可以暴露这些特性的完整集合：

1. 在查询字符串中接受一个额外的参数。
2. 使用HTTP动词，如post和put。

两种方式实现起来都相当简单。第一种更简单一些，不论是你还是REST客户端，只不过这种方式会限制你能够接收的数据量并且具有潜在的负面效应。

如果你到处使用get方法，那么别人要想建立一个请求就是一件非常容易的事，因为那不过就是一个标准的URL加上查询字符串。这是人们非常熟悉的操作，并且可以从浏览器地址栏中复制请求来测试他们的代码。

不过，许多Web服务器都在它们能够处理的URL的大小方面施加了限制。而如果想添加一条新记录，通常需要传递大量的数据。对于以post方法发送的数据就没有这种限制。因此，对于添加或更新记录的请求，get方法不是一个好的选择。

此外，根据HTTP规范，get请求被认为不应该修改后台数据。如果你设计了一个站点，那么当一个人发送了两个一样的get请求，他就应该得到两个相同的应答结果。

当你允许人们通过get方法来添加、更新或者删除记录时，其实就是在违反HTTP的原则。虽然这在正常情况下不成问题，但当你不注意时很可能会吓人一跳。例如，对于自动化的脚本，如Google Spider程序，会设法索引你的页面。如果你在HTML锚标签的href属性中暴露了能实施破坏性操作的URL，那么Spider程序就可能会打开该链接，并从你的数据库中删除信息。

而且，Google Web Accelerator的最初版本在使用短查询字符串的URL执行删除操作的网站上会导致很多问题。有关这一问题的讨论可以参考*http://radar.oreilly.com/archives/2005/05/google_web_acce_1.html*。

尽管如此，添加另一个get参数仍然是很直观的，而且只需要最少的编码，如例15-2所示。

例15-2：实现能够完成多种操作的REST服务器

```
<?php
// 在switch()中添加更多具体的动作特定逻辑
switch ($_GET ['action'] ) {
case 'search':
    $action = 'search';
    break;
case 'add':
    $action = 'add';
    break;
case 'update':
    $action = 'update';
    break;
case 'delete':
    $action = 'delete';
    break;
default:
    // 无效动作
    exit();
}

// 音乐数据库XML文档转移到文件中
$s = simplexml_load_string('music_database.xml');

if ($action == 'search') {
    $artist = $_GET ['artist'] ;
```

```
    $query = "/music/album [artist = '$artist'] ";
    $albums = $s->xpath($query);

    // 在这里显示结果
} elseif ($action == 'add') {
    $artist = $_GET ['artist'] ;
    $album = $_GET ['album'] ;

    // 用输入数据插入新节点
}

// ... 在此执行其他动作
?>
```

在页面的顶部，通过检查$_GET['action']来发现一组有效的动作，如果找到一个则将其赋给变量$action。

然后，加载数据源（这里是不合适使用XML文件的地方，因为你不能像对数据库一样给它加锁）。

现在就可以执行相应的操作了。如果是搜索，那么就像例15-1一样查询数据并输出结果。

对于添加操作，应该更新存储的数据，然后用一段简短的信息回复说一切顺利。例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<response code="200">Album added</response>
```

否则，如果操作失败，就发送一个出错信息：

```
<?xml version="1.0" encoding="UTF-8"?>
<response code="400">Invalid request</response>
```

虽然很多人都使用这种方法来检测查询参数以决定采取什么动作，但你还有另外一种选择，就是使用HTTP动词，如get，post，put和delete。这种方式是更“纯粹”的REST形式，它不仅可以让你能够轻松地处理大型的请求，而且也更加安全。因为它意外删除数据的概率非常之小。

表15-1显示了SQL命令和HTTP动词之间的一般对应关系

表15-1：SQL命令、HTTP动词和REST动作

SQL	REST
CREATE	POST
SELECT	GET

表15-1：SQL命令、HTTP动词和REST动作（续）

SQL	REST
UPDATE	PUT
DELETE	DELETE

在使用HTTP动词的时候，要检查的$_SERVER['REQUEST_METHOD']而不是$_GET['action']中的值，如例15-3所示。

例15-3：使用HTTP动词实现的REST服务器

```
<?php
// 在switch()中添加更多具体的动作特定逻辑

// 转换为大写
$request_method = strtoupper($_SERVER ['REQUEST_METHOD'] );

switch ($request_method) {
case 'GET':
    $action = 'search';
    break;
case 'POST':
    $action = 'add';
    break;
case 'PUT':
    $action = 'update';
    break;
case 'DELETE':
    $action = 'delete';
    break;
default:
    // 无效动作
    exit();
}

// ... 在此执行其他动作
?>
```

在例15-3中，除了在页面顶部改为使用REQUEST_METHOD之外，还必须更新代码来检查HTTP动词名，包括get，post，put和delete。而且现在必须在动词不是get的情况下使用$_POST 而不是$_GET。

记住，$_SERVER['REQUEST_METHOD']的安全性与$_GET['action']是相同的，也就是说根本没有安全性。因为这两个值都很容易被设置，所以如果你想暴露敏感的数据或者允许可能破坏数据的操作，一定要保证发送请求的人得到相应的授权。

参见

技巧14.1中有关如何调用REST方法的介绍；技巧9.1中更多检查REQUEST_METHOD值的讨论。

15.2 以SOAP方法提供服务

问题

你想创建一个SOAP服务器来响应SOAP请求。

方案

使用ext/soap的SOAPServer类。下面就是一个返回当前日期和时间的服务器：

```
<?php
class pc_SOAP_return_time {
    public function return_time() {
        return date('Ymd\THis');
    }
}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');
$server->handle();
?>
```

讨论

通过ext/soap的SOAPServer类建立一个SOAP服务器分三个步骤：

1. 创建一个处理SOAP方法的类。
2. 创建一个SOAP服务器的技巧，并以该技巧关联这个处理类。
3. 命令SOAP服务器处理请求并对SOAP客户端作出应答。

ext/soap的SOAPServer类可以使用函数或者类来处理SOAP请求。例15-4中使用了pc_SOAP_return_time类，该类有一个方法 —— return_time()。

例15-4：pc_SOAP_return_time类

```
<?php
class pc_SOAP_return_time {
    public function return_time() {
```

```
        return date('Ymd\THis');
    }
}
?>
```

一旦定义好类之后，就要技巧化SOAPServer对象。如果你的服务有一个WSDL文件，将该文件作为第一个参数；否则，就像本例中一样，传递null。第二个参数包含你的配置选项。这里只有一个uri用来指定SOAP服务器的命名空间。在例15-5中即urn:pc_SOAP_return_time。

例15-5：技巧化SOAPServer

```
<?php
$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
?>
```

当ext/soap在处理SOAP请求时，它不会注意你的PHP类的名字，诸如pc_SOAP_return_time。真正要紧的是XML命名空间，即刚才设置的urn:pc_SOAP_return_time。

接下来，以类名为参数调用SOAPServer::setClass()方法。当SOAP服务器收到对一个方法的请求时，它就会尝试以相同的名称调用一个类方法：

```
<?php
$server->setClass('pc_SOAP_return_time');
$server->handle();
?>
```

最后，通过调用SOAPServer::handle()来通知服务器响应请求。SOAPServer会自动处理$GLOBALS['HTTP_RAW_POST_DATA']变量，其中保存了PHP的 POST数据。

如果SOAP请求来自另一个源，假设来自电子邮件信息，可以将其中的数据传递给SOAP-Server::handle()：

```
<?php
$server->handle($soap_message_from_someplace_else);
?>
```

在这两种情况下，SOAPServer会仔细解析SOAP数据，并进行相应的例行处理。

要使用ext/soap客户端来调用这一程序，可以使用例15-6中的代码。

例15-6：使用SOAP取得时间

```
<?php
$opts = array('location' => 'http://api.example.org/getTime',
              'uri' => 'urn:pc_SOAP_return_time');

$client = new SOAPClient(null, $opts);
```

```
$result = $client->__soapCall('return_time', array());

print "The local time is $result.\n";
?>
```

这样会输出：

```
The local time is 20060816T083225.
```

也可以不绑定一个类，而是关联一个单独的函数：

```
<?php
function return_time() {
    return date('Ymd\THis');
}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->addfunction('return_time');
$server->handle();
?>
```

还能以一个包含函数名的数组作为参数来调用SOAPServer::addFunction()，以绑定多于一个的函数：

```
<?php
$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));

// 要暴露的函数数组
$functions = array('return_time', 'return_date');
$server->addfunction($functions);
?>
```

另一种选择是绑定全部函数：

```
<?php
$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));

// 添加 *all* 函数
$server->addfunction(SOAP_FUNCTIONS_ALL);
?>
```

但我们强烈反对这种用法，因为存在巨大的安全隐患。当你偶尔包含带有秘密信息的函数时，该信息会由于你使用了SOAP_FUNCTIONS_ALL而被暴露出来。所以，你应该始终按照选择参加的原则明确地列举出希望暴露的函数。

如果方法没有标记为public或者根本就没有定义，例如set_time，则服务器会以一个SOAP故障作为应答，应答中将包含字符串"Function 'set_time' doesn't exist"和SOAP-ENV:Server故障代码。

要想改变这一行为，可以通过__call()方法来绑定一个类：

```
<?php
class pc_SOAP_Process_All_Methods {

    // 在这里处理任何未定义的方法
    public function __call($name, $args) {
        // ...
    }
}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_Process_All_Methods'));
$server->setClass('pc_SOAP_Process_All_Methods');
$server->handle();
?>
```

如果你使用zlib扩展来建立PHP程序，SOAPServer将会自动支持以gzip和compress方式压缩的请求。在处理这些数据之前它会先进行解压缩。

参见

技巧14.2中有关调用SOAP方法的介绍；技巧15.3中在SOAP方法中接收参数的讨论；技巧15.5中有关抛出SOAP故障的信息；ext/soap扩展的文档（*http://www.php.net/soap*）。

15.3 在SOAP方法中接受参数

问题

你想让SOAP方法接受参数。

方案

更新该方法的原型，使其包含参数：

```
<?php
class pc_SOAP_return_time {
    public function return_time($tz = '') {
        // 根据输入来设置时区
        if ($tz) { $my_tz = date_default_timezone_set($tz); }
        // 取得新的时间戳
        $date = date('Ymd\THis');
        // 把时区重置为默认值
        if ($tz) { date_default_timezone_set(ini_get('date.timezone')); }
        // 返回时间戳
        return $date;
```

```
    }
}

$server = new SOAPServer(null,array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');
$server->handle();
?>
```

讨论

服务于SOAP请求的基本知识在技巧15.2中已经介绍了。本技巧扩展了上一个技巧中的示例，以示范如何让方法接受参数。

可以通过修改方法的原型使其包含参数来实现可读入参数。随后再修改客户端请求，使其包含传递给新增参数的数据。例15-7修改了SOAP程序以使其接受一个可选的时区参数

例15-7：处理带参数的SOAP方法

```
<?php
class pc_SOAP_return_time {
    public function return_time($tz = '') {
        // 根据输入来设置时区
        if ($tz) { $my_tz = date_default_timezone_set($tz); }
        // 取得新的时间戳
        $date = date('Ymd\THis');
        // 把时区重置为默认值
        if ($tz) { date_default_timezone_set(ini_get('date.timezone')); }
        // 返回时间戳
        return $date;
    }
}

$server = new SOAPServer(null,array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');
$server->handle();
?>
```

现在，SOAP客户端就可以传入tz选项了。在本例中该参数为Europe/Oslo：

```
<?php
$opts = array('location' => 'http://api.example.org/getTime',
              'uri' => 'urn:pc_SOAP_return_time');

$client = new SOAPClient(null, $opts);

$result = $client->__soapCall('return_time', array('tz' => 'Europe/Oslo'));

print "The local time is $result.\n";
?>
```

根据这个新的设置，服务器返回了比前一个时间早9个小时的时间：

```
The local time is 20060816T173225.
```

可以给SOAP方法传递字符串、数字、数组和对象参数。ext/soap扩展可以将它们由XML转换成本地的PHP数据类型。

参见

技巧15.2中有关处理不带参数的SOAP请求的介绍。

15.4 自动生成WSDL文件

问题

你想通过SOAP的Web服务暴露一组方法。你想自动生成一个描述这一服务的WSDL文件。

方案

ext/soap扩展不支持WSDL生成功能。不过，可以使用其他一些PHP脚本。

讨论

从SOAP的自身特点来看，为客户提供一个WSDL文件以便他们能够根据你的服务器自己配置请求是极其重要的。遗憾的是，ext/soap却不支持生成WSDL的功能。

因此，我们要么必须从零开始手工生成WSDL文件或者修改一个支持类似操作集合的文档，要么使用非官方的脚本，比如：

WSDL_Gen, 开发者 George Schlossnagle

http://www.schlossnagle.org/~george/blog/index.php?/archives/234-WSDLGeneration.html

wsdl-writer, 开发者 Katy Coe based on code by David Griffin

http://www.djkaty.com/drupal/php-wsdl

Web service helper, 开发者David Kingma

http://jool.nl/new/

以上任何一个脚本都没有完全支持SOAP和WSDL规则，而且每个脚本都采取了彼此稍有不同的语法来实现各自的功能。所以，你需要对它们仔细地分析一番，以便确定哪一个适合你的需要且符合你的编程风格。

参见

WSDL规范（*http://www.w3.org/TR/wsdl*）。

15.5 抛出SOAP故障

问题

你想要生成一个SOAP故障，它是SOAP用于指示错误的机制。

方案

调用SOAPServer::fault()方法：

```
<?php
class pc_SOAP_return_time {
    public function return_time() {
        $date = date('Ymd\THis');
        if ($date === false) {
            $GLOBALS ['server'] ->fault(1, 'Bad dates.');
        }
        return $date;
    }
}

$server = new SOAPServer(null,array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');

?>
```

或者抛出一个SOAPFault：

```
<?php
class pc_SOAP_return_time {
    public function return_time() {
        $date = date('Ymd\THis');
        if ($date === false) {
            throw new SOAPFault(1, 'Bad dates.');
        }
        return $date;
    }
}
```

```
$server = new SOAPServer(null,array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');

?>
```


也可以return一个SOAPFault而不是抛出它。

讨论

SOAP规范中有一个指示错误的标准方式 —— SOAP故障。SOAP故障与面向对象概念中的异常非常相似。实际上，ext/soap允许你以PHP处理异常非常相近的方式处理SOAP故障，不论是在SOAP服务器端还是在SOAP客户端。

虽然可以使用几种不同的方式指示SOAP故障，但最简单的方式是throw一个SOAPFault类的技巧，并给构造器传递一个错误代码和一个错误字符串，如例15-8所示。

例15-8：抛出一个SOAP故障

```
<?php
class pc_SOAP_return_time {
    public function return_time() {
        $date = date('Ymd\THis');
        if ($date === false) {
            throw new SOAPFault(1, 'Bad dates.');
        }
        return $date;
    }
}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');

$server->handle();
}
?>
```

在例15-8中，当date()返回false时会抛出一个SOAPFault。其错误代码为1，而按照印第安那州Jones的突发奇想，错误消息是Bad dates。

这两个值会分别被映射到SOAP 1.1规范中的faultcode和faultstring元素。在本书写作之时，还没有对SOAP 1.2式的SOAP故障的支持。

在正常情况下，错误代码用于允许知程序处理错误，而错误消息则用于让人类理解发生了什么 —— 一般是通过日志文件或者输出。

与HTTP和状态代码不同，对于SOAP的错误代码并没有约定。例如，500区段并没有为服务器错误而保留。你可以随意设置自己想要的代码集合。

但是，当你放出一个SOAP故障时，SOAP扩展会自动地把HTTP状态代码设置为500。这是SOAP规范所要求的。也就是说，你不能把非500的代码用作HTTP状态代码。

除了抛出SOAPFault之外，你也可以通过自己的方法return一个SOAP故障，或者调用SOAPServer::fault()方法。这些方式生成的SOAP故障数据都是一致的，所以这是一个个人偏好或者视代码情况而定的问题。

除了直接使用SOAPFault，还可以将其子类化并使用该子类。这样你就可以使用一个综合的日志系统了，例如：

```
<?php
class pc_My_SOAPFault extends SOAPFault {
    public function __construct($code, $string) {
        parent::__construct($code, $string);
        error_log($this);
    }
}
?>
```

当代码在执行过程中出现什么错误的时候，例如调用一个未定义的函数时，就会自动生成SOAP故障。

参见

技巧14.10中有关在SOAP客户端捕捉SOAP故障的讨论。

15.6 处理SOAP头部信息

问题

你想要能够读取来自SOAP服务器中的SOAP头部信息。

方案

绑定一个与SOAP头部元素同名的函数或者方法：

```
class pc_SOAP_return_time {
    public function set_timezone($tz) {
        date_default_timezone_set($tz);
    }

    public function return_time() {
        return date('Ymd\THis');
```

```
    }
}
```

当ext/soap得到了一个名为set_timezone的SOAP头部信息时，它就会调用set_timezone()方法。set_timezone元素中的数据则作为参数传递。

SOAP头部信息先于SOAP主体信息被处理。

讨论

与HTTP相似，SOAP可以让你定义SOAP头部和SOAP主体元素的内容。但SOAP主体是必须定义的，而SOAP头部则是可选的。SOAP头部中通常会包含诸如身份认证证书或其他适合发送到服务的数据，它与你调用某种方法并不特别相关。

当ext/soap发现一个带有SOAP头部的客户端请求时，它首先会尝试调用一个具有相同名字的函数。调用完成后，会继续调用在SOAP主体中指定的函数。这样就可以使你基于SOAP头部数据来执行任何预请求的事务。例如，如果在头部中包含了身份认证证书，你可以验证该用户，或者对未经授权的用户抛出一个SOAP故障。

例15-9中显示了一个来自例15-7中定义的SOAP服务器return_time的示例，它可以让你通过设置SOAP头部而不是将其作为一个可选的参数传递来设置时区。

例15-9：处理SOAP头部信息

```
<?php
class pc_SOAP_return_time {
    public function set_timezone($tz) {
        date_default_timezone_set($tz);
    }

    public function return_time() {
        return date('Ymd\THis');
    }
}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');

$server->handle();
?>
```

例15-9支持两个方法：set_timezone()和return_time()。在实践中，可以支持通过SOAP头部来调用第一个方法，而对第二个方法通过SOAP主体来调用。但ext/soap不会以编程的方式对这两者进行真正的区分。

但是，当ext/soap发现一个SOAP头部时，它会在处理主体之前尝试调用与该头部元素同名的方法。因此，现在你就可以传递一个set_timezone的SOAP头部信息，把Web服务器默认时区设置为其他时区。而这个头部信息中应该包含时区名作为信息内容。

在设置完时区之后，SOAP服务器会继续检查SOAP主体。当它发现了通常的return_time请求时，就会返回该数据。不过，修改后的时区值仍旧会保持，所以返回的也是相应移位后的数据。

例15-10显示如何在PHP中实现这一点。

例15-10：通过SOAP取得时间并使用SOAP头部设置时区

```
<?php
$opts = array('location' => 'http://api.example.org/getTime',
              'uri' => 'urn:pc_SOAP_return_time');

$client = new SOAPClient(null, $opts);

$set_timezone = new SOAPVar('Europe/Oslo', XSD_STRING);
$tz = new SOAPHeader('urn:pc_SOAP_return_time', 'set_timezone', $set_timezone);

$result = $client->__soapCall('return_time', array(), array(), array($tz));

print "The local time is $result.\n";
?>
```

在为服务器创建了一个SOAPClient后，又创建了SOAPHeader。这个SOAPHeader元素处于urn:pc_SOAP_return_time XML命名空间中，名为set_timezone。实际上，ext/soap扩展并不使用这个XML命名空间的值（这可能与其他SOAP服务器不同），但是头部名称非常重要，因为它控制着SOAPServer调用的方法。

SOAPHeader构造器的第三个参数$set_timezone就是包含在SOAP头部信息中的数据。在例15-10中，就是SOAPVar。

SOAPVar类是一个用于创建SOAP变量的低级类，例如可以用它创建字符串和数组。当你在SOAP主体中发送数据时，很少会用到这个类。然而，由于ext/soap的限制，这个类在处理SOAP头部时就可以派上用场了。

虽然通过SOAPVar来构造SOAP请求组件有点麻烦，但它却能使你完全控制要发送的内容。上例中的代码创建了一个值为Europe/Oslo的字符串。XSD_STRING常量是许多XML模式中的一种，而SOAP常量则是由ext/soap扩展注册的。要了解完整的常量列表，可以访问PHP手册中的SOAP页面，网址为*http://www.php.net/soap*。

例15-10中的请求会序列化为：

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="urn:pc_SOAP_return_time"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <SOAP-ENV:Header>
        <ns1:set_timezone>Europe/Oslo</ns1:set_timezone>
    </SOAP-ENV:Header>

    <SOAP-ENV:Body>
        <ns1:return_time/>
    </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

如果SOAP客户端指定了一个头部，但没有相应的方法来处理它，ext/soap会跳过该方法直接转到主体中。但是，如果头部中的mustUnderstand属性标记为true，那么SOAPServer就会放出一个包含SOAP-ENV:MustUnderstand故障代码和Header not understood故障字符串的SOAP故障。

参见

技巧15.7中有关生成SOAP头部信息的内容；技巧14.7中有关使用SOAP头部的信息；SOAPHeader的文档（*http://www.php.net/manual/function.soap-soapheader-construct.php*）。

15.7 生成SOAP头部信息

问题

你想从你的SOAP服务器发送一个SOAP头部信息。

方案

调用 addSoapHeader()方法：

```
<?php
class pc_SOAP_return_time {
```

```
    public function return_time() {
        $tz = date_default_timezone_get();
        $header = new SoapHeader('urn:pc_SOAP_return_time', 'get_timezone', $tz)
        $GLOBALS ['server'] ->addSoapHeader($header);

        return date('Ymd\THis');
    }
}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');

$server->handle();
?>
```

这会把下面的XML添加到SOAP响应中：

```
<SOAP-ENV:Header><ns1:get_timezone>America/Los Angeles</ns1:get_timezone>
</SOAP-ENV:Header>
```

讨论

处理由SOAP客户端发送的SOAP头部信息是一种典型的模式。相应的主题在技巧15.6中已经讨论过了。但是，你也可以在你的SOAP服务器内部创建一个新的SOAP头部信息，并将其发回到客户端。

这个过程可以分成两步进行：

1. 创建一个SOAPHeader的新技巧。
2. 使用SOAPServer::addHeader()方法将该头部添加到应答中。

例15-11中显示了PHP中的实现过程。

例15-11：从SOAP服务器端发送一个SOAP头部信息

```
<?php
class pc_SOAP_return_time {
    public function return_time() {
        $tz = date_default_timezone_get();
        $header = new SoapHeader('urn:pc_SOAP_return_time', 'get_timezone', $tz
)
        $GLOBALS ['server'] ->addSoapHeader($header);

        return date('Ymd\THis');
    }
}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');
```

```
$server->handle();
?>
```

在例15-11中，pc_SOAP_return_time()方法创建了一个包含默认时区的、名为get_timezone的新SOAP头部信息。

随后，通过调用$GLOBALS['server']->addSoapHeader($header);把这个头部添加到应答当中。由于在方法的作用域中不能轻易地访问到$server对象，所以就直接通过$GLOBALS数组来访问该对象。

现在，当ext/soap响应时，其应答信息中就会包含这个SOAP头部信息。例如：

```
<SOAP-ENV:Header><ns1:get_timezone>America/Los Angeles</ns1:get_timezone>
</SOAP-ENV:Header>
```

至于如何处理该头部信息则完全由SOAP客户端决定了。可以通过向__soapCall()中传递第五个参数来使用SOAPClient访问这个头部信息：

```
$result = $client->__soapCall('return_time', array(), array(), array(), $output);
print_r($output);

Array
(
    [timezone]  => America/Los_Angeles
)
```

参见

技巧15.6中有关处理SOAP头部信息的讨论；技巧14.7中有关使用SOAP头部信息的讨论；SOAPHeader的文档（*http://www.php.net/manual/function.soap-soapheader-construct.php*）。

15.8 通过SOAP实现验证

问题

你想要验证SOAP请求。这样可以只对受信任的客户提供服务。

方案

使用HTTP Basic验证方式进行验证：

```
<?php
// 验证逻辑
// 可能需要与你的SOAP服务器进行解耦
function pc_authenticate_user($username, password) {
    // 验证用户
    $is_valid = true; //在这里实现查找过程

    if ($is_valid) {
        return true;
    } else {
        return false;
    }
}


class pc_SOAP_return_time {
    public function __construct() {
        // 对于无效的用户名和密码组合抛出SOAP故障
        if (! pc_authenticate_user($_SERVER['PHP_AUTH_USER'],
                                   $_SERVER['PHP_AUTH_PW'])) {

            throw new SOAPFault("Incorrect username and password combination.", 401);
        }
    }

    // 这里是其他的SOAP服务器方法……
}

$server = new SOAPServer(null,array('uri'=>"urn:pc_SOAP_return_time"));
$server->setClass("pc_SOAP_return_time");

$server->handle();
?>
```

或者，使用SOAP头部：

```
<?php
// 验证逻辑
// 可能需要与你的SOAP服务器进行解耦
function pc_authenticate_user($username, password) {
    // 验证用户
    $is_valid = true; // 在这里实现查找过程

    if ($is_valid) {
        return true;
    } else {
        return false;
    }
}

class pc_SOAP_return_time {
    public function authenticate_user($args) {
        // 对于无效的用户名和密码组合抛出SOAP故障
        if (! pc_authenticate_user($args->username,
                                   $args->password)) {
```

```
            throw new SOAPFault("Incorrect username and password combination.", 401);
        }
    }

    // 这里是其他的SOAP服务器方法……
}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');

$server->handle();
?>
```

讨论

相较于应用程序其他部分中的标准验证，SOAP认证并不十分复杂。将其整合到你的SOAP服务器中的途径有两个：使用HTTP Basic验证或者处理自定义的SOAP头部。

如果你对这些概念不熟悉，应该先看一看技巧8.9中有关HTTP的Basic和Digest验证的介绍和技巧15.6中有关SOAP头部信息及如何处理这些头部信息的讨论。

例15-12中显示了如何使用HTTP Basic验证方式。

例15-12：通过HTTP Basic验证和SOAP服务器实现认证

```
<?php
// 验证逻辑
// 可能需要与你的SOAP服务器进行解耦
function pc_authenticate_user($username, password) {
    // 验证用户
    $is_valid = true; //在这里实现查找过程

    if ($is_valid) {
        return true;
    } else {
        return false;
    }
}


class pc_SOAP_return_time {
    public function __construct() {
        // 对于无效的用户名和密码组合抛出SOAP故障
        if (! pc_authenticate_user($_SERVER['PHP_AUTH_USER'],
                                   $_SERVER['PHP_AUTH_PW'])) {

            throw new SOAPFault("Incorrect username and password combination.", 401);
        }
    }

    // 这里是其他的SOAP服务器方法……
}
```

```
$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');

$server->handle();
?>
```

例15-12中定义了pc_authenticate_user()。这个函数并非SOAP专有的，这是你用来处理用户验证的标准代码。在这个例子中，它以独立的函数出现以强调其解耦的特点。但是，如果你想通过它来保存状态或者访问其他对象属性，也可以在pc_SOAP_return_time中将它定义为一个私有方法。

通过为pc_SOAP_return_time定义构造器，就可以强制SOAP服务器在处理任何SOAP头部或主体内容之前先执行其中的代码。在__construct()内部，调用pc_authenticate_user，并传递PHP存储HTTPBasic验证证书的变量，$_SERVER['PHP_AUTH_USER']和$_SERVER['PHP_AUTH_PW']。

如果验证失败，会抛出一个SOAP故障。记住，你必须在SOAP故障上发送一个值为500的HTTP状态码，才能使PHP不返回401错误。

可以像下面这样来传递HTTP Basic验证证书：

```
<?php
$opts = array('location' => 'http://api.example.org/getTime',
              'uri' => 'urn:pc_SOAP_return_time',
              'login' => 'elvis',
              'password' => 'the-king',

$client = new SOAPClient(null, $opts);

$result = $client->__soapCall('return_time');
?>
```

SOAPClient可以接受login和password选项。这两个选项应该被设置为适当的用户名和密码，而ext/soap会处理以HTTP Basic验证方式来发送它们的其他工作。

另一种途径是不通过HTTP Basic验证，而是通过一个自定义的SOAP头部信息来传递用户名和密码。这样使你可以更多地控制能够得到的信息，并进一步让你使用非HTTP协议。

这种方式的缺点是它不像HTTP Basic验证那样是一种熟悉的概念，对此人们还需要学习如何构造自定义的头部信息。

例15-13大致地展示了这种途径的基本步骤。

例15-13：通过SOAP头部和SOAP服务器实现验证

```
<?php
// 验证逻辑
// 可能需要与你的SOAP服务器进行解耦
function pc_authenticate_user($username, password) {
    // 验证用户
    $is_valid = true; // 在这里实现查找过程

    if ($is_valid) {
        return true;
    } else {
        return false;
    }
}


class pc_SOAP_return_time {
    private $authenticated;

    public function __construct() {
        $this->authenticated = false;
    }

    public function authenticate_user($args) {
        // 对于无效的用户名和密码组合抛出SOAP故障
        if (! pc_authenticate_user($args->username,
                                   $args->password)) {

            throw new SOAPFault("Incorrect username and password combination.", 401);
        }

        $this->authenticated = true;
    }

    // 这里是其他的SOAP服务器方法……
    public function soap_method() {
        if ($this->authenticated) {
            // 这里是方法主体……
        } else {
            throw new SOAPFault("Must pass authenticate_user Header.", 401);
        }
    }

}

$server = new SOAPServer(null, array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');

$server->handle();
?>
```

其中的pc_authenticate_user()函数是相同的，因为没有必要仅仅修改了传递用户名和密码的接口就改变底层系统。

但是，这次没有把验证检查放在构造器中，而是将其放在了验证头部信息后的一个命名方法中。因为没有办法强制ext/soap请求SOAP头部信息，就使用这个方法来把authenticated属性设置为true。

然后，要确保每个SOAP主体中的方法被包含在一个if($this->authenticated)检查为“真”的代码中。如果SOAP客户端没有通过验证，则抛出一个SOAP故障。

下面示范了如何使用SOAPClient传递证书：

```
<?php
$opts = array('location' => 'http://api.example.org/getTime',
              'uri' => 'urn:pc_SOAP_return_time');

$client = new SOAPClient(null, $opts);

class SOAPAuth {
    public $username;
    public $password;

    public function __construct($username, $password) {
        $this->username = $username;
        $this->password = $password;
    }
}

$auth = new SOAPAuth('elvis', 'the-king');
$header = new SOAPHeader('urn:example.org/auth', 'authenticate_user', $auth);
$result = $client->__soapCall('return_time', array(), array(), array($header));
?>
```

创建同时带有username和password变量的SOAP头部的最简单方式就是创建一个简单的类，该类的属性中带有相应的用户名，然后将该类的一个技巧传递给SOAPHeaher。

之后，整个SOAP头部信息就会在__soapCall()中作为第四个参数被发送到服务器端。

参见

技巧14.8中在SOAP客户端中使用验证的介绍；技巧8.9中有关HTTP Basic验证和Digest验证的常识性信息。

15.9 以XML-RPC方法提供服务

问题

你想创建一个XML-RPC服务器并对XML-RPC请求作出响应。这样就可以使任何启用了XML-RPC的客户端能够与你的服务器进行对话并得到应答数据。

方案

使用PHP的XML-PRC扩展。下面是Userland的XML-RPC演示程序的PHP版，该程序返回了带有当前日期和时间的ISO 8601字符串：

```
// 这是暴露为"get_time()"的函数
function return_time($method, $args) {
   return date('Ymd\THis');
}

$server = xmlrpc_server_create() or die("Can't create server");
xmlrpc_server_register_method($server, 'return_time', 'get_time')
    or die("Can't register method.");

$request = $GLOBALS['HTTP_RAW_POST_DATA'];
$options = array('output_type' => 'xml', 'version' => 'xmlrpc');

print xmlrpc_server_call_method($server, $request, NULL, $options)
    or die("Can't call method");

xmlrpc_server_destroy($server);
```

讨论

由于绑定的XML-RPC扩展xmlrpc-epi是以C编写的，所以它会以迅速而高效的方式来处理XML-RPC请求。在编译期间通过在配置字符串中添加--with-xmlrpc就可以启用这个扩展。要了解XML-RPC的更多内容，请参见技巧14.12。

这个方案在开始定义了一个与XML-RPC方法关联的PHP函数。该函数的名字为return_time()。后面将会通过XML-RPC的get_time()方法与其建立关联：

```
function return_time($method, $args) {
   return date('Ymd\THis');
}
```

这个函数会返回一个包含当前日期和时间的ISO 8601格式的字符串。之所以在调用date()时对T进行转义，是因为规范中要求用一个T直接量字符来分隔日期部件和时间部件。对于2002年8月21日下午3:03:51，该函数的返回值为20020821T150351。

这个函数会自动地通过两个参数被调用：服务器响应的XML-RPC方法名和由XML-RPC客户端传递到服务器端的参数数组。在这个例子中，服务器忽略了这两个变量。

接着，创建XML-RPC服务器并注册get_time()方法：

```
$server = xmlrpc_server_create() or die("Can't create server");
xmlrpc_server_register_method($server, 'return_time', 'get_time');
```

创建一个新的服务器并将其指定给$server，然后以三个参数来调用xmlrpc_server_register_method()。第一个是新创建的服务器，第二个是要注册的方法名，第三个是处理请求的PHP函数名。

现在，配置工作已经就绪，需要告诉XML-RPC服务器为了处理和向客户端输出结果而分派方法：

```
$request = $GLOBALS ['HTTP_RAW_POST_DATA'] ;
$options = array('output_type' => 'xml', 'version' => 'xmlrpc');

print xmlrpc_server_call_method($server, $request, NULL, $options);
```

客户端的请求来自于POST数据。PHP会将HTTP的POST数据转换为变量，但这是一个XML-RPC数据，所以服务器需要访问的是未解析的数据，该数据保存在$GLOBALS['HTTP_RAW_POST_DATA']中。在这个例子中，请求XML是下面这样的：

```
<?xml version="1.0" encoding="iso-8859-1"?>
<methodCall>
<methodName>get_time</methodName>
<params/></methodCall>
```

因而，服务器对get_time()方法作出响应，并且不接受参数。

你还必须对以XML输出结果的响应选项进行配置，并将其作为XML-RPC请求进行解释。这两个变量随后连同XML-RPC服务器$server被传递给xmlrpc_server_call_method()。传递给这个函数的第三个参数是你希望提供的任意用户数据 —— 这里是空值，所以传递NULL。

xmlrpc-server_call_method()函数对变量进行解码，调用正确的函数来处理这个方法，然后将响应编码为XML-RPC。为了应答客户端，你要做的就是输出xmlrpc_server_call_method()的返回结果。

最后，通过调用xmlrpc_server_destroy()完成清理：

```
xmlrpc_server_destroy($server);
```

通过技巧14.12中的XML-RPC客户端代码，你可以发送一个请求并查寻时间，像下面这样：

```
require 'utils.php';

$output = array('output_type' => 'xml', 'version' => 'xmlrpc');
$result = xu_rpc_http_concise(array(
                              'method'  => 'get_time',
                              'host'    => 'clock.example.com',
                              'port'    => 80,
                              'uri'     => '/time-xmlrpc.php',
                              'output'  => $output));

print "The local time is $result.\n";

The local time is 20020821T162615.
```

通过一个单独的XML-RPC服务器关联多个方法是合法的。一个PHP函数也可以关联多个方法，例如，可以创建一个服务器同时应答两个方法：get_gmtime()和get_time()。第一个方法，get_gmtime()与get_time()类似，但是返回的是当前的GMT时间。要做到这一点，可以扩展get_time()，使其接收一个可选的参数，该参数是在需要计算当前时间时所用的时区名。

下面示范了如何修改return_time()函数，使其能够同时处理两个方法：

```
function return_time($method, $args) {
    if ('get_gmtime' == $method) {
        $tz = 'GMT';
    } elseif (!empty($args [0] )) {
        $tz = $args [0] ;
    } else {
        // 使用本地时区
        $tz = '';
    }

    if ($tz) { putenv("TZ=$tz"); }
    $date = date('Ymd\THis');
    if ($tz) { putenv('TZ=EST5EDT'); } // change EST5EDT to your server's zone

    return $date;
}
```

这个函数同时使用了$method和$args参数。在函数的顶部，我们检查是否请求了get_gmtime。如果是，将时区设置为GMT。如果不是，则通过检查$args[0]来确定是否通过参数指定了一个替代时区。如果以上两项检查都返回true，则保持当前时区不变。

在服务器端进行配置以便支持这个新方法，需要新添加仅一行代码：

```
xmlrpc_server_register_method($server, 'return_time', 'get_gmtime');
```

这样就把 get_gmtime()映射到了return_time()。

下面是一个实际的客户端的例子。第一个请求是请求不带参数的get_time()，第二个请求以PST8PDT调用了get_time()，该时区位于服务器的时区之后三个小时。最后一个请求所请求的是新的get_gmtime()方法，它位于服务器的时区之前四个小时：

```
require 'utils.php';

$output = array('output_type' => 'xml', 'version' => 'xmlrpc');

// get_time()
$result = xu_rpc_http_concise(array(
                              'method'  => 'get_time',
                              'host'    => 'clock.example.com',
                              'port'    => 80,
                              'uri'     => '/time.php',
                              'output'  => $output));

print "The local time is $result.\n";

// get_time('PST8PDT')
$result = xu_rpc_http_concise(array(
                              'method'  => 'get_time',
                              'args'    => array('PST8PDT'),
                              'host'    => 'clock.example.com',
                              'port'    => 80,
                              'uri'     => '/time.php',
                              'output'  => $output));

print "The time in PST8PDT is $result.\n";

// get_gmtime()
$result = xu_rpc_http_concise(array(
                              'method'  => 'get_gmtime',
                              'host'    => 'clock.example.com',
                              'port'    => 80,
                              'uri'     => '/time.php',
                              'output'  => $output));

print "The time in GMT is $result.\n";

The local time is 20020821T162615.
The time in PST8PDT is 20020821T132615.
The time in GMT is 20020821T202615.
```

参见

技巧14.12中有关XML-RPC客户端的更多信息；xmlrpc_server_create()函数的文档（*http://www.php.net/xmlrpc-server-create*），xmlrpc_server_register_method()函数的文档（*http://www.php.net/xmlrpc-server-register-method*），xmlrpc_server_

call_method()函数的文档（*http://www.php.net/xmlrpc-server-call-method*）和xmlrpc_server_destroy()函数的文档（*http://www.php.net/xmlrpc-server-destroy*）。Simon St.Laurent, Joe Johnston和Edd Dumbill 合著的《Programming Web Services with XML-RPC》（O'Reilly）；有关XML-RPC的更多内容（*http://www.xml-rpc.com*）；初始当前时间XML-RPC服务器的资料（*http://www.xmlrpc.com/currentTime*）。

第16章

互联网服务

16.0 概述

在HTTP出现之前，还有FTP，NNTP，IMAP，POP3等一系列其他的协议。由于浏览器提供了一个整合的程序，它能够让人们查收电子邮件、阅读新闻组、传输文件和查看文件而不用担心基于潜在通讯手段的各种细节。PHP提供了内置的和通过PEAR提供的函数，以使用这些其他的协议。通过这些函数，你可以使用PHP来创建各种基于网络的Web前后台应用程序，比如查找域名或者发送基于Web的电子邮件等。虽然PHP能够简化这些工作，但理解每个协议的长处和局限也是非常重要的。

技巧16.1到16.3介绍了这些协议支持的最流行的功能 —— 电子邮件。16.1介绍了如何发送基本的电子邮件信息。技巧16.2则说明了MIME编码的电子邮件，这种电子邮件能够让你发送纯文本和HTML格式的信息。有关读取收件箱的IMAP和POP3协议在技巧16.3中讨论。

接下来的两个技巧讨论了如何通过NNTP读取新闻组信息。新闻组与邮件列表的用途类似，但它不是让列表中的每个人都通过接收电子邮件进行沟通，而是通过让人们访问新闻服务器并查看各自感兴趣的信息。新闻组也可以发表带线索（thread）的信息，所以通过它能够很方便地跟踪某个话题的讨论。技巧16.4讨论了发表信息，而16.5则介绍了检索信息。

技巧16.6包含了如何使用FTP（文件传输协议，File Transfer Protocol）交换文件的内容，FTP是一种跨互联网发送和接收文件的方法。FTP服务器可以要求用户使用密码登录或者匿名使用。

搜索LDAP服务器是技巧16.7的主题，而技巧16.8则讨论了如何针对一个LDAP服务器来验证用户信息。LDAP服务器可以用于地址簿或者用于集中地存储用户信息。这两种应

用都要求对信息检索进行优化，并且能够做到可配置地复制其中的信息以确保高度的可靠性和快速的响应时间。

本章最后几个技巧与网络有关。技巧16.9包含了DNS查询，既包括从域名到IP的查询也包含相反的查询。技巧16.10讲述了如何判断某个主机是否处于工作状态并且可以通过PEAR的ping模块进行访问。

本书的其他部分也都涉及了一些网络协议。HTTP的详细信息包含在了第13章中。其中的技巧讨论了如何以各种不同的方式定位URL。组合了HTTP和XML的协议在第14和15章中进行了讨论。这两章讨论了消费和提供Web服务的方法，涉及到REST，SOAP和XML-RPC协议。

16.1 发送电子邮件

问题

你想要发送一封电子邮件。可能是直接响应某个用户的操作，例如在你的网站上注册，或者是基于设定时间的循环事件，例如发送每周一次的时事通讯。

方案

使用PEAR的Mail类：

```
require 'Mail.php';

$to = 'adam@example.com';

$headers['From'] = 'webmaster@example.com';
$headers['Subject'] = 'New Version of PHP Released!';

$body = 'Go to http://www.php.net and download it today!';

$message =& Mail::factory('mail');
$message->send($to, $headers, $body);

   If you can't use PEAR's Mail class, use PHP's built-in mail()
   function:
$to = 'adam@example.com';
$subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';

mail($to, $subject, $body);
```

讨论

PEAR的Mail类可以让你通过三种方式发送电子邮件。你需要在使用Mail::factory()技巧化一个mail对象时指明一种方法。

- 要通过*sendmail*或*qmail*之类的外部程序来发送邮件，传递sendmail。
- 要使用SMTP服务器，传递smtp。
- 要使用内置的mail()函数，传递mail。这是告诉Mail把设置应用到你的*php.ini*文件中。

在使用sendmail或smtp方式时，你必须传递第二个参数来说明你的设置。若使用sendmail，需要指定sendmail_path和sendmail_args：

```
$params['sendmail_path'] = '/usr/sbin/sendmail';
$params['sendmail_args'] = '-oi -t';

$message =& Mail::factory('sendmail', $params);
```

对sendmail来说一个不错的值是*/usr/lib/sendmail*。然而遗憾的是，*sendmail*可能会在各个系统之中位于不的位置，所以要捕捉到很不容易。如果你找不到它，可以试试*/usr/sbin/sendmail*或者去问一下系统管理员。

传递给*sendmail*的两个标记-oi和-t都是很有用的。其中，-oi标记是告诉*sendmail*不将一行中的一个单独的点（.）当成邮件信息的结尾。而-t标记则可以让*sendmail*为“收件人”解析文件和其他标题行。

如果更喜欢*qmail*，可以试试*/var/qmail/bin/qmail-inject*或*/var/qmail/bin/sendmail*。

如果你使用Windows操作系统，可能希望使用SMTP服务器，因为大多数Windows机器可能都不会安装sendmail。为些，需要传递smtp：

```
$params['host'] = 'smtp.example.com';

$message =& Mail::factory('smtp', $params);
```

在smtp模式下，可以传递五个可选的参数：host表示SMTP服务器主机名，默认是localhost；port指的是连接端口，默认为25；要启用SMTP身份验证，需要将auth设置为true；如果想让服务器验证你，则需要设置username和password。SMTP功能并不局限于Windows系统中，在Unix服务器中同样可以使用。

如果你没有PEAR的Mail类，可以使用内置的mail()函数。mail()用于发送邮件的程序在

*php.ini*文件的sendmail_path配置变量中指定。如果你使用Windows，可以将SMTP变量设置为你的SMTP服务器的主机名。而“发件人”地址则来自于sendmail_from变量。

下面是使用mail()发送邮件的一个例子：

```
$to = 'adam@example.com';
$subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';

mail($to, $subject, $body);
```

其中第一个参数是接收者的电子邮件地址，第二个是信息的主题，而最后一个是信息的主体内容。也可以通过可选的第四个参数再添加额外的头部信息。例如，下面示范了如何添加Reply-To和Organization头部信息：

```
$to = 'adam@example.com';
$subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';
$header = "Reply-To: webmaster@example.com\r\n"
        ."Organization: The PHP Group";

mail($to, $subject, $body, $header);
```

其中使用了\r\n来分割每个头部元素，但是不要在最后一个头部元素后面加上\r\n。

无论你选择了什么方法，编写一个辅助你发送电子邮件的包装函数都是一个不错的主意。强制所有的邮件都通过这个函数来发送可以更容易地添加日志，以及对发送的每一封邮件进行检查等其他功能：

```
function pc_mail($to, $headers, $body) {
    $message =& Mail::factory('mail');

    $message->send($to, $headers, $body);
    error_log("[MAIL][TO: $to]");
}
```

这里，可以将一个信息写入错误日志，以记录所发送的每封邮件的收件人。它还为你跟踪某人通过你的服务器发送垃圾邮件提供了一个时间戳。另一种功能是创建一个“不能发送”的电子邮件地址清单，通过该清单来防止有人不断地从你的服务器接收到邮件。也可以对所有收件人的地址进行验证，这样可以减少退信的数量。

参考

技巧9.4中有关通过正则表达式来验证电子邮件地址的介绍。技巧16.2中有关发送MIME电子邮件的讨论。技巧16.3中更多关于检索邮件的内容。mail()函数的文档

（*http://www.php.net/mail*），PEAR的Mail类的文档（*http://pear.php.net/package-info.php?package=Mail*），RFC 822规范（*http://www.faqs.org/rfcs/rfc822.html*），O'Reilly出版的两本*sendmail*的书：ryan Costales 同 Eric Allman合著的《sandmail》和《sendmail Desktop Reference》

16.2 发送MIME邮件

问题

你想要发送一封MIME电子邮件。例如，你想发送既包含纯文本也包含HTML内容的多部分（multipart）的邮件，而且能够识别MIME的邮件阅读器会自动正确地显示出各部分信息。

方案

使用PEAR中的Mail_mime类：

```
require 'Mail.php';
require 'Mail/mime.php';

$to = 'adam@example.com, sklar@example.com';

$headers['From'] = 'webmaster@example.com';
$headers['Subject'] = 'New Version of PHP Released!';

// 创建MIME对象
$mime = new Mail_mime;

// 添加邮件主体内容
$text = 'Text version of email';
$mime->setTXTBody($text);

$html = '<html><body>HTML version of email</body></html>';
$mime->setHTMLBody($html);

$file = '/path/to/file.png';
$mime->addAttachment($file, 'image/png');

// 取得MIME格式化后的邮件信息头部和主体
$body = $mime->get();
$headers = $mime->headers($headers);

$message =& Mail::factory('mail');
$message->send($to, $headers, $body);
```

讨论

PEAR的Mail_mime类为创建包含文本和HTML邮件的所有幕后细节提供了一个面向对象的接口。这个类与PEAR的Mail类相似，但是它没有把邮件主体定义为字符串，而是通过创建一个Mail_mime对象并调用其方法的方式把邮件的各个部分添加到邮件主体中：

```
// 创建MIME对象
$mime = new Mail_mime;

// 添加邮件主体内容
$text = 'Text version of email';
$mime->setTXTBody($text);

$html = '<html><body>HTML version of email</body></html>';
$mime->setHTMLBody($html);

$file = '/path/to/file.txt';
$mime->addAttachment($file, 'text/plain');

// 取得MIME格式化后的邮件信息头部和主体
$body = $mime->get();
$headers = $mime->headers($headers);
```

其中的Mail_mime::setTXTBody()和Mail_mime::setHTMLBody()方法分别用于向邮件主体中添加纯文本内容和HTML内容。在这里，我们传递的是变量，你也可以传递一个文件名，让Mail_mime自行读取其中的内容。要传递文件名，还需要同时传递true作为第二个参数：

```
$text = '/path/to/email.txt';
$mime->setTXTBody($text, true);
```

如果想给邮件添加附件，如图片或文档，需要调用Mail_mime::addAttachment()：

```
$file = '/path/to/file.png';
$mime->addAttachment($file,'image/png');
```

给这个函数传递文件所在的位置以及MIME类型。

在邮件完成后，就是最后的准备并将其发出：

```
// 取得MIME格式化后的邮件信息头部和主体
$body = $mime->get();
$headers = $mime->headers($headers);

$message =& Mail::factory('mail');
$message->send($to, $headers, $body);
```

首先，取得Mail_mime对象提供的经过适当格式化的头部和主体信息。然后，使用父类Mail来创建邮件并通过Mail::send()将邮件发出。

参见

技巧16.1中发送常规电子邮件的内容。技巧16.3中有关接收邮件的更多介绍。PEAR的Mail_Mime类的文档（*http://pear.php.net/package-info.php?package=Mail_Mime*）。

16.3 通过IMAP或POP3读取邮件

问题

你想使用IMAP或POP3来读取邮件，可以通过这种方式创建一个基于web的客户端。

方案

使用PHP的IMP扩展，该扩展能够同时与IMAP和POP3进行对话：

```
// 打开IMAP连接
$mail = imap_open('{mail.server.com:143}',      'username', 'password');
// 或者，打开POP3连接
$mail = imap_open('{mail.server.com:110/pop3}', 'username', 'password');

// 取得所有邮件头部信息的列表
$headers = imap_headers($mail);

// 取得邮箱中最后一封邮件的头部对象
$last = imap_num_msg($mail);
$header = imap_header($mail, $last);

// 取得邮件的主体
$body = imap_body($mail, $last);

// 关闭连接
imap_close($mail);
```

讨论

PHP用于支持IMAP和POP3的后台库为你自行编写一个完整的邮件客户端提供了几乎无数种功能。虽然功能强大的了，但复杂性也相应地提高。事实上，当前PHP的IMAP扩展中有多达63种不同的函数，并且这还没有算上那些能实现POP3与NNTP之间对话的函数。

然而，与邮件服务器通讯的基础部分是很直观明了的。同PHP中很多其他的功能类似，首先也是打开连接并取得一个句柄：

```
$mail = imap_open('{mail.server.com:143}', 'username', 'password');
```

这样就打开了一个到名为*mail.server.com*的邮件服务器143端口的IMAP连接。同时也传递了一个用户名和一个密码作为第二和第三个参数。

而在打开POP3连接时，则需要在服务器和端口号后面加上/pop3。因为POP3通常占用110端口，所以在服务器名后面加上了:110：

```
$mail = imap_open('{mail.server.com:110/pop3}', 'username', 'password');
```

如果想通过SSL来对连接进行加密，可以同加/pop3一样，再在末尾加上/ssl。而且，你还必须确保你的PHP安装程序中除配置了对--with-imap的支持外，也配置了--with-imap-ssl选项。同样，也需要你自己通过SSL支持来构建系统IMAP库自身。如果你使用的是自标记（self-signed）的认证而希望不进行验证，还要添加/novalidate-cert。最后，多数SSL连接都会使用993或995端口。以上所有选项可以以任意顺序出现，所以下面的的写法完全合法：

```
$mail = imap_open('{mail.server.com:993/novalidate-cert/pop3/ssl}',
                  'username', 'password');
```

像{$var}这样，用大括号在一对双引号中括住一个变量，是告诉PHP实际要插入的是什么变量的一种方式。因此，在将插值变量作为imap_open()的第一个参数时，需要用表示转义开始的{符号：

```
$server = 'mail.server.com';
$port = 993;

$mail = imap_open("\{$server:$port}", 'username', 'password');
```

在打开连接之后，你就可以向邮件服务器提出各种请求了。要得到邮箱中所有邮件的列表，使用imap_headers()：

```
$headers = imap_headers($mail);
```

这时会返回一个数组，数组中的每个元素都是一个与邮件对应的格式化字符串：

```
A   189) 5-Aug-2007 Beth Hondl          an invitation (1992 chars)
```

另外，如果想取得具体的邮件信息，使用imap_header()和imap_body()取得头部对象和主体内容字符串：

```
$header = imap_header($message_number);
$body   = imap_body($message_number);
```

其中，imap_header()函数返回一个包含很多字段的对象。有用的字段包含subject，fromaddress和update等。技巧16.5中的表16-2中列出了其中的全部字段。

主体元素只是一个字符串，但如果邮件是一封多部分邮件，如一封既包含HTML也包含文本内容的邮件，那么$body中则包含了各个组成部分以及用于描述它们的MIME信息：

```
------=_Part_1046_3914492.1008372096119
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Plain-Text Message

------=_Part_1046_3914492.1008372096119
Content-Type: text/html
Content-Transfer-Encoding: 7bit

<html>HTML Message</html>
------=_Part_1046_3914492.1008372096119--
```

为了避免出现这种问题，可以组合使用imap_fetchstructure()和imap_fetchbody()来发现主体内容是如何被格式化的，并解析出你想要的各个部分：

```
// 从邮件信息$n中取得纯文本内容
$st = imap_fetchstructure($mail, $n);
if (!empty($st->parts)) {
    for ($i = 0, $j = count($st->parts); $i < $j; $i++) {
        $part = $st->parts[$i];
        if ($part->subtype == 'PLAIN') {
            $body = imap_fetchbody($mail, $n, $i+1);
        }
     }
} else {
    $body = imap_body($mail, $n));
}
```

如果一封邮件中包含多个部分，$st->parts中会保存一个描述它们的对象数组。而$part中保存着一个描述主体内容MIME类型的整数。表16-1中列出了数字与MIME类型的对应关系。而subtype属性中保存的是MIME的子类型，并可以根据它来判断该部分内容是纯文本、HTML、PNG或者是其他类型，如八进制数据流。

表16-1：IMAP中的MIME类型值

数字	MIME 类型	PHP 常量	描述	例子
0	text	TYPETEXT	未格式化的文本	纯文本、HTML、XML

表16-1：IMAP中的MIME类型值（续）

数字	MIME 类型	PHP 常量	描述	例子
1	multipart	TYPEMULTIPART	多部分邮件	混合的、带符号的表单数据
2	message	TYPEMESSAGE	压缩邮件	新闻，HTTP
3	application	TYPEAPPLICATION	应用程序数据	八位字节流、PDF、Zip
4	audio	TYPEAUDIO	音乐文件	MP3、RealAudio
5	image	TYPEIMAGE	图像文件	GIF、JPEG、PNG
6	video	TYPEVIDEO	视频剪辑	MPEG, Quicktime
7	other	TYPEOTHER	其他	VRML models

参见

技巧16.1中发送常规电子邮件的内容。技巧16.2中有关发送邮件的更多介绍。imap_open()函数的文档（*http://www.php.net/imap_open*），imap_header()函数的文档（*http://www.php.net/imap_header*），imap_body()函数的文档（*http://www.php.net/imap_body*）和IMAP的相关资料（*http://www.php.net/imap*）。

16.4 将邮件发表到Usenet新闻组

问题

你想在Usenet新闻组，如*comp.lang.php*中发表一条消息。

方案

使用imap_mail_compose()来对消息进行格式化，然后使用套接字将这则消息写入服务器：

```
$headers['from'] = 'adam@example.com';
$headers['subject'] = 'New Version of PHP Released!';
$headers['custom_headers'][] = 'Newsgroups: comp.lang.php';

$body[0]['type'] = TYPETEXT;
$body[0]['subtype'] = 'plain';
$body[0]['contents.data'] = 'Go to http://www.php.net and download it today!';

$post = imap_mail_compose($headers, $body);

$server = 'nntp.example.com';
$port = 119;
```

```
$sh = fsockopen($server, $port) or die ("Can't connect to $server.");
fputs($sh, "POST\r\n");
fputs($sh, $post);
fputs($sh, ".\r\n");
fclose($sh);
```

讨论

PHP中没有内置函数可以将消息发表到新闻组中。因此，必须打开一个到新闻服务器的直接套接字连接并发送命令，才能发表该消息。不过，你可以使用imap_mail_compose()来格式化一条消息并为其创建头部和主体内容。每条消息必须带有三个头部信息：发件人地址、消息的主题和新闻组的名称：

```
$headers['from'] = 'adam@example.com';
$headers['subject'] = 'New Version of PHP Released!';
$headers['custom_headers'][] = 'Newsgroups: comp.lang.php';
```

我们创建了一个数组$headers，用于保存消息的头部。可以直接指定发件人地址from和消息的主题subject的值，但对于Nesgroups:头部元素则不能直接指定。因为imap_mail_compose()多数情况下都是用于创建电子邮件信息，而Nesgroups:头部并不是一个预定义的头部元素。为了解决这个问题，你必须将其添加到custom_header数组元素中。

对于custom_headers这里使用了一种不同的语法。不是将小写的头部元素名称作为数组元素的名称，也不是将头部元素的值作为数组元素的值，而是将整个头部作为一个数组的值。在头部元素名称与值之间，添加了一个冒号和一个空格，还要确保正确地拼写了Nesgroups: ——N大写，最后加s。

这条消息的主体可以由多个部分组成。为此，传递给imap_mail_compose()的body元素是一个数组的数组。在本方案中，只有一个部分，所以我们直接把值赋给了$body[0]：

```
$body[0]['type'] = TYPETEXT;
$body[0]['subtype'] = 'plain';
$body[0]['contents.data'] = 'Go to http://www.php.net and download it today!';
```

消息的每个部分都需要一个MIME类型和一个子类型。这条消息的编码为ASCII，所以类型为TYPETEXT，而子类型为plain。要了解IMAP的MIME类型常量及其含义，可以参考技巧16.3中的表16-1。这里用到的contents.data字段保存了消息的主体内容。

为了把这些数组转换成为一个格式化的字符串，我们调用了imap_mail_compose($body, $headers)。然后，会返回一条如下所示的消息：

```
From: adam@example.com
Subject: New Version of PHP Released!
MIME-Version: 1.0
Content-Type: TEXT/plain; CHARSET=US-ASCII
Newsgroups: comp.lang.php

Go to http://www.php.net and download it today!
```

组成了一则新闻服务器能够接收的消息后，接着调用fsockopen()打开一个连接：

```
$server = 'nntp.example.com';
$port = 119;

$sh = fsockopen($server, $port) or die ("Can't connect to $server.");
```

fsockopen()的第一个参数是服务器的主机名，第二个参数是使用的端口号。如果你不知道新闻服务器的主机名，可以在相应的域名前面加上*news*，*nntp*或*news-server*之类的主机名试一试。例如：*news.example.com*, *nntp.example.com*或*news-server.example.com*。如果这些都不成功，可以问一下系统管理员。按照惯例，所有的新闻服务器都会使用119端口。

连接成功后，就可以发送消息了：

```
fputs($sh, "POST\r\n");
fputs($sh, imap_mail_compose($headers, $body));
fputs($sh, ".\r\n");
```

上面第一行代码是告诉新闻服务器你想发表消息，第二行则是消息本身。为了标识出消息的结尾，在单独的一行中使用了一个句点。其中每一行代码中都必须同时包含一个回车符和一个换行符。最后调用fclose($sh)来关闭连接。

服务器上的每一条消息都有一个唯一的名称，称为消息ID。如果你想要回复一条消息，需要得到被回复消息的消息ID，并将其指定为References头部元素的值：

```
// 在读取原消息时取得
$message_id = '<20030410020818.33915.php@news.example.com>';

$headers['custom_headers'][] = "References: $message_id";
```

参见

技巧16.5中有关新闻组的更多内容。imap_mail_compose()函数的文档（*http://www.php.net/imap-mail-compose*），fsockopen()函数的文档（*http://www.php.net/fsockopen*），fputs()函数的文档（*http://www.php.net/fputs*）和fclose()函数的文档（*http://www.php.net/fclose*）。RFC 977规范（*http://www.faqs.org/rfcs/rfc977.html*）。

16.5 读取Usenet新闻消息

问题

你想通过NNTP与服务器通讯来读取Usenet新闻消息。

方案

使用PHP的IMAP扩展。该扩展也支持NNTP：

```
// 打开一个到nntp服务器的连接
$server = '{news.php.net/nntp:119}';
$group = 'php.general'; // 主PHP邮件列表
$nntp = imap_open("$server$group", '', '', OP_ANONYMOUS);

// 取得头部信息
$header = imap_header($nntp, $msg);

// 找出相关字段
$subj  = $header->subject;
$from  = $header->from;
$email = $from[0]->mailbox."@".$from[0]->host;
$name  = $from[0]->personal;
$date  = date('m/d/Y h:i A', $header->udate);

// 取得主体内容
$body  = nl2br(htmlspecialchars(imap_fetchbody($nntp,$msg,1)));

// 关闭连接
imap_close($nntp);
```

讨论

在从新闻服务器中读取新闻消息时，要求你必须连接到服务器并指定一个你有兴趣阅读的组：

```
// 打开一个对nntp服务器的连接
$server = "{news.php.net/nntp:119}";
$group = "php.general";
$nntp = imap_open("$server$group",'','',OP_ANONYMOUS);
```

imap_open()函数带有四个参数。第一个指定的是所用的新闻服务器和要读取的新闻组。这里的服务器是*news.php.net*，这个新闻服务器镜像了所有的PHP邮件列表。添加/nntp可以让IMAP扩展知道你要读取的是新闻而不是邮件，后面的119是端口号 —— 通常都是为NTTP（网络新闻传输协议，Network News Transport Protocol）保留的，用于与新闻

服务器之间进行通讯（同HTTP与web服务器通讯一样）。新闻组是*php.general*——PHP社区中的主邮件列表。

imap_open()的中间两个参数是用户名和密码，以提供身份验证信息。因为*news.php.net*是对所有读者开放的，所以这里为空。最后，传递的是标志OP_ANONYMOUS，这个标志告诉IMAP你是一个匿名读者——这样它就不会在特殊的*.newsrc*文件中保存你的记录。

在连接成功之后，通常你既想取得一份当前消息的列表，还想取得某条消息的具体内容。下面是用于显示当前消息的代码：

```
// 读取并显示消息索引
$last = imap_num_msg($nntp);
$n = 10; // 显示最新的10条消息

// 表头
print <<<EOH
<table>
<tr>
    <th align="left">Subject</th>
    <th align="left">Sender</th>
    <th align="left">Date</th>
</tr>
EOH;

// 消息
for ($i = $last-$n+1; $i <= $last; $i++) {
    $header = imap_header($nntp, $i);

    if (! $header->Size) { continue; }

    $subj  = $header->subject;
    $from  = $header->from;
    $email = $from[0]->mailbox."@".$from[0]->host;
    $name  = $from[0]->personal ? $from[0]->personal : $email;
    $date  = date('m/d/Y h:i A', $header->udate);

print <<<EOM
<tr>
    <td><a href="$_SERVER[PHP_SELF]"?msg=$i\">$subj</a></td>
    <td><a href="mailto:$email">$name</a></td>
    <td>$date</td>
</tr>
EOM;
     }

// 表结尾
echo "</table>\n";
```

在浏览消息列表时，你需要通过数字来指定想查看的个别消息。一个组中的第一条消息

总是以数字1来表示，而表示最新消息的数字是imap_num_msg()的返回值。所以，要想得到最新的$n条消息，需要在$last-$n+1到$last之间进行循环。

在这个循环内部，调用imap_header()取得与消息有关的头部信息。这个头部信息中包含了所有的元信息，但不会包含实际的消息文本 —— 它们保存在主体中。因为头部通常比主体要小一些，所以你不用花太多时间就能检索很多消息的数据。

现在给imap_header()传递两个参数 —— 服务器连接句柄和消息序号。然后，这个函数会返回一个包含很多属性的对象，这些属性如表16-2所示。

表16-2：imap_header()从NNTP服务器中取得的字段

名称	说明	类型	例子
date或Date	RFC 822格式的数据：date('r')	String	Fri, 16 Aug 2002 01:52:24 -0400
subject或Subject	邮件主题	String	Re: PHP Cookbook Revisions
message_id	标识邮件的唯一ID	String	<20030410020818.33915.<php@news.example.com>>
newsgroups	邮件发送给的组名称	String	php.general
toaddress	邮件发送的地址	String	<php-general@lists.php.net>
to	toaddress字段的解析版	Object	mailbox: "php-general", host: "lists-php.net"
fromaddress	发送邮件的地址	String	Ralph Josephs <<ralph@example.net>>
from	fromaddress字段的解析版	Object	personal: "Ralph Josephs", mailbox: "ralph", host: "example.net"
reply_toaddress	回复地址 —— 如果你想联络作者	String	<rjosephs@example.net>
reply_to	reply_toaddress字段的解析版	Object	Mailbox: "rjosephs", host: "example.net"
senderaddress	发送邮件的人 —— 几乎总是与from字段相同。但如果from字段无法唯一标识是谁发送的邮件，则使用这个字段	String	Ralph Josephs <<ralph@example.net>>
sender	senderaddress字段的解析版	Object	Personal: "Ralph Josephs", mailbox: "ralph", host: "example.net"
Recent	自用户上次查收邮件以来，这封邮件是否是新邮件？	String	Y或N
Unseen	这封邮件是否未见过？	String	Y或" "

表16-2：imap_header()从NNTP服务器中取得的字段（续）

名称	说明	类型	例子
Flagged	这封邮件是否已标记？	String	Y或" "
Answered	是否已回复了这封邮件？	String	Y或" "
Deleted	这封邮件是否已经删除？	String	Y 或 " "
Draft	这封邮件草稿吗？	String	Y 或 " "
Size	这封邮件的字节大小	String	1345
udate	邮件日期的Unix时间戳	Int	1013480645
Mesgno	组中的邮件数	String	34943

其中比较有用的几个字段是：size，subject，from和udate。size字段是消息的字节大小，如果值为0，说明该消息不是被删除了就是被移到别处去了。subject字段是消息的主题。而from列表稍微复杂一些，它是一个数组对象，其中每个元素都保存了一个带有三个属性（personal，mailbox和host）的对象。personal属性是发表人的名字（Homer Simpson）。mailbox属性是电子邮件地址中@符号之前的部分（homer）。而host是电子邮件地址中@符号后面的部分（thesimpsons.com）。一般而言，在这个from列表数组中只有一个元素（对象），因为一条消息通常只有一个发表人。

之所以把$header->from对象放到$from变量中，是因为由于存在中间数组，导致PHP不能直接访问$header->from[0]->personal。然后再将$from[0]->mailbox和$from[0]->host组合构成发表人的电子邮件地址。如果personal属性的值存在，则通过三元操作符将其指定为发表人的名字，否则使用电子邮件地址代替。

udate字段是以Unix时间戳表示的发表时间。这里使用date()来把它由秒转换为合适的格式。

你也可以像下面这样来查询一条特殊的消息：

```
// 读取并显示一条单独的消息
$header = imap_header($nntp, $msg);

$subj  = $header->subject;
$from  = $header->from;
$email = $from[0]->mailbox."@".$from[0]->host;
$name  = $from[0]->personal;
$date  = date('m/d/Y h:i A', $header->udate);
$body  = nl2br(htmlspecialchars(imap_fetchbody($nntp,$msg,1)));

print <<<EOM
<table>
<tr>
```

```
    <th align=left>From:</th>
    <td>$name &lt;<a href="mailto:$email">$email</a>&gt;</td>
</tr>
<tr>
    <th align=left>Subject:</th>
    <td>$subj</td>
</tr>
<tr>
    <th align=left>Date:</th>
    <td>$date</td>
</tr>
<tr>
    <td colspan="2">$body</td>
</tr>
</table>
EOM;
```

上面取得一条单独消息的代码与取得一系列消息头部的代码类似。主要的区别在于：这里你将`$body`变量定义为由三个函数嵌套作用的返回结果。在最内层，调用了`imap_fetchbody()`返回消息主体，调用时使用了与调用`imag_header()`时相同的参数。然后，将返回的结果传递给`htmlspecialchars()`以转义可能妨碍你的HTML。最后再把转义后的结果传递给`nl2bc()`，该函数将所有的回车转换成XHTML的`<br/>`标签 —— 这样才能保证消息能够正确地显示在网页中。

在断开与IMAP服务器的连接并关闭流时，需要把IMAP连接句柄传递给`imap_close()`：

```
// 完成任务时关闭连接
imap_close($nntp);
```

参见

技巧16.4中向在新闻组发表消息的内容。`imap_open()`函数的文档（*http://www.php.net/imap-open*），`imap_header()`函数的文档（*http://www.php.net/imap-header*），`imap_body()`函数的文档（*http://www.php.net/imap-body*）和IMAP的基本资料（*http://www.php.net/imap*）。在不使用IMAP的情况下通过PHP读取新闻组信息的代码（*http://cvs.php.net/cvs.php/php-news-web*）。RFC 977规范（*http://www.faqs.org/rfcs/rfc977.html*）。

16.6 通过FTP取得及放置文件

问题

你想使用FTP来传输文件。

方案

使用PHP内置的FTP函数：

```
$c = ftp_connect('ftp.example.com')     or die("Can't connect");
ftp_login($c, $username, $password)     or die("Can't login");
ftp_put($c, $remote, $local, FTP_ASCII) or die("Can't transfer");
ftp_close($c);                          or die("Can't close");
```

也可以使用cURL扩展：

```
$c = curl_init("ftp://$username:$password@ftp.example.com/$remote");
// $local 是保存在本地计算机中的文件位置
$fh = fopen($local, 'w') or die($php_errormsg);
curl_setopt($c, CURLOPT_FILE, $fh);
curl_exec($c);
curl_close($c);
```

讨论

FTP是在计算机间交换文件的一种方法。与HTTP服务器不同，设置一台既能发送又能接收文件的FTP服务器相当容易。

使用内置的FTP函数虽然不需要额外的库，但你必须明确通过`--enable-ftp`来启用它们。因为这些函数专门为FTP设计，所以通过它们来传输文件很简单。

所有的FTP传输一开始都是建立一个从你的计算机 —— 本地客户端，到另一台计算机 —— 远程服务器端的连接：

```
$c = ftp_connect('ftp.example.com')     or die("Can't connect");
```

在连接成功之后，需要发送你的用户和密码以让远程服务器进行验证并允许你登录：

```
ftp_login($c, $username, $password)     or die("Can't login");
```

有的FTP服务器支持一种叫做匿名FTP的特性。在匿名FTP之上，用户可以在没有远程系统账户的情况下登录。在使用匿名FTP时，你的用户名是anonymous，而你的密码则是你的电子邮件地址。

下面两行代码显示了如何通过`ftp_put()`和`ftp_get()`传输文件：

```
ftp_put($c, $remote, $local,  FTP_ASCII) or die("Can't transfer");
ftp_get($c, $local,  $remote, FTP_ASCII) or die("Can't transfer");
```

`ftp_put()`函数会从你的计算机中取得一个文件并将其拷贝到远程服务器，而`ftp_get()`

则会从远程服务器中拷贝一个文件到你的计算机中。在前面的代码中，$remote是远程文件的路径名，而$local指向了你计算机中的文件。

这两个函数的最后一个参数可以传递两个值。这里使用的是FTP_ASCII参数，会将文件作为ASCII文本进行传输，而结尾的换行符会根据不同的操作系统自动转换。另一个参数是FTP_BINARY，这个选项是针对非纯文本文件传输的，所以不会进行换行符的转换。

使用ftp_fget()和ftp_fput()可以将文件下载或者上传到一个打开的文件指针（用fopen()打开）处，而不是传输到文件系统中的某个位置上。例如，下面示范了如何对一个存在的文件指针$fp实现检索和写入的操作：

```
$fp = fopen($file, 'w');
ftp_fget($c, $fp, $remote, FTP_ASCII)   or die("Can't transfer");
```

最后，为了断开与远程主机的连接，调用ftp_close()注销：

```
ftp_close($c);                          or die("Can't close");
```

如果想调用连接超时的秒数，可以使用ftp_set_option()：

```
// 将超时时间提高到两分钟
set_time_limit(120)
$c = ftp_connect('ftp.example.com');
ftp_set_option($c, FTP_TIMEOUT_SEC, 120);
```

超时时间的默认值为90秒，但是PHP脚本默认的max_executiong_time则是30秒。所以如果发现你的连接过早地超时，要确保同时检查一下这两个值。

在使用cURL扩展时，必须先从*http://curl.haxx.se*下载cURL并在安装PHP时设置--with-curl配置选项。开始时使用curl_init()创建一个cURL句柄，然后通过curl_setopt()指定你要做什么。

curl_setopt()函数接受三个参数：一个cURL资源，一个要修改的cURL常量和要指定给第二个参数的值。在本方案中，使用了CURLOPT_FiLE常量：

```
$c = curl_init("ftp://$username:$password@ftp.example.com/$remote");
// $local是保存在本地客户端中的文件位置
$fh = fopen($local, 'w') or die($php_errormsg);
curl_setopt($c, CURLOPT_FiLE, $fh);
curl_exec($c);
curl_close($c);
```

在调用curl_init()时我们传递了一个URL地址。因为该URL开始于ftp://，所以cURL会使用FTP协议。这里不是分开传递用户名和密码，而是把用户名及密码直接嵌入到了

URL之中，之后又设置了在服务器端保存该文本的位置。接着，以写入模式打开了名为$local的文件，并将其文件句柄传递给curl_setopt()中作为CURLOPT_FILE的值。在通过cURL传输文件时，它会自动地写入到文件句柄。当一切配置就绪后，就可以调用curl_exec()来启动传输并在之后通过curl_close()关闭连接。

参见

FTP扩展的文档（*http://www.php.net/ftp*）和cURL的文档（*http://www.php.net/curl*）。RFC 959规范（*http://www.faqs.org/rfcs/rfc969.html*）。

16.7 通过LDAP找地址

问题

你想查询一个LDAP服务器的地址信息。

方案

使用PHP的LDAP扩展：

```
$ds = ldap_connect('ldap.example.com')                 or die($php_errormsg);
ldap_bind($ds)                                         or die($php_errormsg);
$sr = ldap_search($ds, 'o=Example Inc., c=US', 'sn=*') or die($php_errormsg);
$e  = ldap_get_entries($ds, $sr)                       or die($php_errormsg);

for ($i=0; $i < $e['count']; $i++) {
    echo $info[$i]['cn'][0] . ' (' . $info[$i]['mail'][0] . ')<br>';
}

ldap_close($ds)                                        or die($php_errormsg)
```

讨论

（LDAP）服务器中保存着目录信息，例如名称和地址，并允许你通过查询得到结果。从很多方面看，它都类似于一个数据库，只不过它是保存着有关人的信息并进行了最优化处理。

此外，LDAP服务器也不像数据库那样提供的是一种平面的数据结果，它所提供的是一种便于组织人员信息的分层结构形式。例如，员工可以分为市场人员、技术人员和操作人员，或者按地域分成北美、欧洲和亚洲。这样就可以很方便地找到某公司一个特定子单位中所有员工的信息。

在使用LDAP时，地址库被称为数据源。该库中的每一个条目都拥有一个全局唯一的标识符，称为识别名称。识别名称中既包含个人的姓名，也包含公司的信息。例如，供职于一家美国公司Example Inc.的John Q.Smith，在LDAP中拥有一个识别名称cn=John Q. Smith, o=Example Inc., c=US，其中，cn是common name的简写，o代表organization，c表示country。

此外，还必须通过设置--with-ldap来启用PHP对LDAP的支持。可以从*http://www.openldap.org*下载LDAP服务器。本技巧涉及了LDAP中基本的知识，要了解更多的相关信息，可以读一读O'Reilly网络中的一些文章（*http://www.onlamp.com/topics/apache/ldap*）。

与LDAP服务器通讯需要四个步骤：连接、认证、搜索记录及注销。除了搜索之外，同样可以添加、修改和删除记录。

开门第一件事就是连接到一个特定的LDAP服务器并通过一个称为绑定的过程来认证你自己：

```
$ds = ldap_connect('ldap.example.com')                    or die($php_errormsg);
ldap_bind($ds)                                            or die($php_errormsg);
```

如果只把连接句柄$ds传递给ldap_bind()，则实现的是匿名绑定。如果想通过一个特殊的用户名和密码进行绑定，可以将用户名及密码作为第二和第三个参数，像下面这样：

```
ldap_bind($ds, $username, $password)                      or die($php_errormsg);
```

在登录成功之后，就可以请求信息了。由于信息是以分层结构组织的，所以需要将基本的识别名称作为第二个参数指出来。最后，需要传递的是搜索标准。例如，下面示范了如何查询供职于美国Example Inc.的姓Jones的人的信息：

```
$sr = ldap_search($ds, 'o=Example Inc., c=US', 'sn=Jones') or die($php_errormsg);
$e  = ldap_get_entries($ds, $sr)                           or die($php_errormsg);
```

在ldap_search()返回结果后，使用ldap_get_entries()来取得特定的数据记录。然后对返回的条目数组$e进行迭代操作：

```
for ($i=0; $i < $e['count']; $i++) {
    echo $e[$i]['cn'][0] . ' (' . $e[$i]['mail'][0] . ')<br>';
}
```

其中没有使用count($e)，而是使用了$e['count']中预先计算好的记录数目值。在循环内部，分别输出每条记录的第一个common name和电子邮件地址。例如：

```
David Sklar (sklar@example.com)
Adam Trachtenberg (adam@example.com)
```

ldap_search()函数会在等于和小于基本识别名称的范围内搜索整个树状数据结构。为了限制结果来自特定的层次，可以使用ldap_list()。因为搜索只发生在一个较小的记录集当中，所以ldap_list()比ldap_search()的速度明显更快。

参见

技巧16.8中有关通过LDAP验证用户身份的介绍。LDAP的文档（*http://www.php.net/ldap*）。RFC 2251规范（*http://www.faqs.org/rfcs/rfc2251.html*）。

16.8 通过LDAP进行用户身份验证

问题

你想将部分站点内容限制为只能由通过身份认证的用户访问。但不是基于数据库中的信息或者使用HTTP的Basic方式认证，而是想使用LDAP服务器来验证。将所有用户信息保存在LDAP服务器中可以使对用户集中管理更加容易。

方案

使用PEAR的Auth类，该类支持LDAP认证：

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');

$auth = new Auth('LDAP', $options);

// 开始验证
// 对匿名用户输出登录界面
$auth->start();

if ($auth->getAuth()) {
    // 对通过验证的用户的内容
} else {
    // 对匿名用户的内容
}

// 注销用户
$auth->logout();
```

讨论

LDAP服务器是专为存储、查询及检索地址信息而设计的，所以比使用MySQL或Oracle之类的标准数据库的效果更好。LDAP服务器的速度非常快，而且通过对不同的用户组授予不同的权限，很容易实现访问控制，此外还支持多种程序查询该服务器。例如，多数电子邮件客户端可以将LDAP服务器作为地址簿使用，如果你请求“John Smith”的信息，服务器会返回John的电子邮件地址*jsmith@example.com*。

PEAR的Auth类可以让你基于文件、数据库和LDAP服务器来验证用户的身份。创建该类技巧时的第一个参数是要使用的认证类型，第二个参数是一个有关如何验证用户信息的数组。例如：

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');

$auth = new Auth('LDAP', $options);
```

这样就创建了一个新的Auth对象，通过它可以根据位于*ldap.example.com*的LDAP服务器对用户身份进行验证，而通讯使用的端口是389。其中，基本目录名是o=Example Inc., c=US，而用户名则根据uid属性来检查。uid是用户标识符（user identifier）的简写。它通常是一个网站的用户名或者一个普通账户的登录名。如果你的服务器没有为每个用户保存uid属性，那么可以用cn属性来代替。这个普通名称（common name）字段中保存的是用户的全名，例如“John Q. Smith”。

Auth::auth()方法还可以接受一个可选的第三个参数 —— 用于显示登录表单的函数名。这个表单可以按照你自己的想法来设计，唯一必须遵守的是该表单的输出字段必须叫做username和password。而且，表单数据的提交方法必须使用POST:

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');

function pc_auth_ldap_signin() {
    print<<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
Name: <input name="username" type="text"><br />
Password: <input name="password" type="password"><br />
<input type="submit" value="Sign In">
</form>
_HTML_;
}
```

```
$auth = new Auth('LDAP', $options, 'pc_auth_ldap_signin');
```

在Auth对象技巧化完成后，可以调用Auth::start()方法对用户进行验证：

```
$auth->start();
```

如果该用户已经登录，什么也不会发生。如果该用户是匿名用户，则会出现登录表单。在对用户进行验证时，Auth::start()会连接到LDAP服务器，完成匿名绑定并搜索地址记录，按照在构造器中指定的用户属性来查找是否存在与表单提交的用户名匹配的记录：

```
$options['userattr'] = $_POST['username']
```

如果Auth::start()发现了一个人符合这个标准，它会取得这个用户的认定名，并尝试使用来自表单的认定名和密码作为登录凭证进行认证绑定。然后，LDAP服务器就会将这个密码与提供的认定名对应的userPassword属性进行比较。如果两者匹配，则用户验证通过。

可以调用Auth::getAuth()，根据返回的布尔值来确定用户的状态：

```
if ($auth->getAuth()) {
    print 'Welcome member! Nice to see you again.';
} else {
    print 'Welcome guest. First time visiting?';
}
```

Auth类使用的是内置的session模块进行用户跟踪，所以当验证通过后，用户的有效期可持续到session过期为止。也可以明确地用以下代码注销该用户：

```
$auth->logout();
```

参见

技巧16.7中有关搜索LDAP服务器的内容。PEAR的Auth类的文档（*http://pear.php.net/package-info.php?package=Auth*）。

16.9 执行DNS查找

问题

你想查询一个域名或者相应的IP地址。

方案

使用gethostbyname()和gethostbyaddr()：

```
$ip   = gethostbyname('www.example.com'); // 192.0.34.72
$host = gethostbyaddr('192.0.34.72'); // www.example.com
```

讨论

你不能完全相信由gethostbyaddr()返回的主机名。一台授权给特定IP地址的DNS服务器可能返回任何主机名。一般来说，管理员在配置DNS服务器时，都要返回正确的主机名，但存心不良的人可能会将它的DNS服务器配置为返回不正确的主机名。揭开这种骗局的一种方法是以gethostbyaddr返回的主机名调用gethostbyname()，看该主机名返回的IP地址是否与原始的IP地址相同。

如果函数都不能成功地查到IP地址和域名，而将返回传递给它的参数，而不会返回false。所以要检查失败时，主要像下面这样：

```
if ($host == ($ip = gethostbyname($host))) {
    // 失败
}
```

这里把gethostbyname()返回的值指定给$ip，并比较$ip是否不等于原始的$host。

有时候一个主机名可能会映射到多个IP地址。如果想找到所有的主机，可以使用gethostbynamel()：

```
$hosts = gethostbynamel('www.yahoo.com');
print_r($hosts);

$hosts = gethostbynamel('www.yahoo.com');
print_r($hosts);
Array
(
    [0] => 64.58.76.176
    [1] => 64.58.76.224
    [2] => 64.58.76.177
    [3] => 64.58.76.227
    [4] => 64.58.76.179
    [5] => 64.58.76.225
    [6] => 64.58.76.178
    [7] => 64.58.76.229
    [8] => 64.58.76.223
)
```

与gethostbyname()和gethostbyaddr()不同，gethostbynamel()返回的是一个数组，而不是一个字符串。

还可以实现更复杂的DNS相关的任务。例如，可以通过getmxrr()得到MX记录：

```
getmxrr('yahoo.com', $hosts, $weight);
for ($i = 0; $i < count($hosts); $i++) {
    echo "$weight[$i] $hosts[$i]\n";
}

getmxrr('yahoo.com', $hosts, $weight);
for ($i = 0; $i < count($hosts); $i++) {
    echo "$weight[$i] $hosts[$i]\n";
}
5 mx4.mail.yahoo.com
1 mx2.mail.yahoo.com
1 mx1.mail.yahoo.com
```

如果想实现时区转换、动态DNS更新以及更多的功能，可以参考PEAR的Net_NDS包。

参见

gethostbyname()函数的文档（*http://www.php.net/gethostbyname*），gethostbyaddr()函数的文档（*http://www.php.net/gethostbyaddr*），gethostbynamel()函数的文档（*http://www.php.net/gethostbynamel*）和getmxrr()函数的文档（*http://www.php.net/getmxrr*）。PEAR的Net_DNS包的文档（*http://pear.php.net/package-info.php?package=Net_DNS*）。Paul Albitz 和 Cricket Liu合著的《DNS and BIND》（O'Reilly）。

16.10 检查主机是否处于活动状态

问题

你想查验一个主机是否仍然正常工作，并且从当前的位置能够访问到它。

方案

使用PEAR的Net_Ping包：

```
require 'Net/Ping.php';

$ping = new Net_Ping;
if ($ping->checkhost('www.oreilly.com')) {
    print 'Reachable';
} else {
    print 'Unreachable';
}

$data = $ping->ping('www.oreilly.com');
```

讨论

这个查验程序会设法从你的机器向另一个台机器发送一些信息。如果一切顺利，你会得到一系列有关该活动的统计记录。而如果返回了错误信息，则说明查验程序出于某些原因无法到达目的主机。

在发生错误时，Net_Ping::checkhost()返回false，而Net_Ping::pinig()则返回常量PING_HOST_NOT_FOUND。如果是因为运行查验程序出现了问题（因为Net_Ping实际上只是这个程序的包装程序），则返回PING_FAILED。

如果没有出现问题，那么你会得到类似下面的数组：

```
$results = $ping->ping('www.oreilly.com');

foreach($results as $result) { print "$result\n"; }

$results = $ping->ping('www.oreilly.com');

foreach($results as $result) { print "$result\n"; }
PING www.oreilly.com (209.204.146.22) from 192.168.123.101 :
    32(60) bytes of data.
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=0 ttl=239
    time=96.704 msec
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=1 ttl=239
    time=86.567 msec
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=2 ttl=239
    time=86.563 msec
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=3 ttl=239
    time=136.565 msec
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=4 ttl=239
    time=86.627 msec

 -- - www.oreilly.com ping statistics  -- -
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/mdev = 86.563/98.605/136.565/19.381 ms
```

Net_Ping不会将这些信息分拆，也不会作任何分析，比如丢包率或者平均往返时间等。不过，你可以自己来解析这些数据：

```
$results = $ping->ping('www.oreilly.com');

// 取得数组的最后一行——等同于非破坏性的array_pop()
// 或者 $results[count($results) - 1]
$round_trip = end($results);
preg_match_all('#[ /]([.\d]+)#', $round_trip, $times);

// 取出数据
list($min,$avg,$max,$mdev) = $times[1];
// 或将其输出
foreach($times[1] as $time) { print "$time\n"; }
```

```
$results = $ping->ping('www.oreilly.com');

// 取得数组的最后一行——等同于非破坏性的array_pop()
// 或者 $results[count($results) - 1]
$round_trip = end($results);
preg_match_all('#[ /]([.\d]+)#', $round_trip, $times);

// 取出数据
list($min,$avg,$max,$mdev) = $times[1];
// 或将其输出
foreach($times[1] as $time) { print "$time\n"; }
83.229
91.230
103.223
7.485
```

这里的正则表达式用于搜索一个空格或一个斜杠，后跟一或多个数字及一个小数点，为了避免转义/，我们用非标准字符#作为分隔符。

参见

PEAR的Net_Ping包的文档（*http://pear.php.net/package-info.php?package=Net_Ping*）。

16.11 获取与域名相关的信息

问题

你想查询与某个域名有关的联系信息或者其他资料。

方案

使用PEAR的Net_Whois类：

```
require 'Net/Whois.php';
$server = 'whois.networksolutions.com';
$query  = 'example.org';
$data = Net_Whois::query($server, $query);
```

讨论

Net_Whois::query()方法会返回一个很大的文本字符串，其内容事实上增加了确凿能够解析出不同Whois结果的难度：

```
Registrant:
Internet Assigned Numbers Authority (EXAMPLE2-DOM)
   4676 Admiralty Way, Suite 330
   Marina del Rey, CA 90292
   US

   Domain Name: EXAMPLE.ORG

   Administrative Contact, Technical Contact, Billing Contact:
      Internet Assigned Numbers Authority  (IANA)  iana@IANA.ORG
      4676 Admiralty Way, Suite 330
      Marina del Rey, CA 90292
      US
      310-823-9358
      Fax- 310-823-8649

   Record last updated on 07-Jan-2002.
   Record expires on 01-Sep-2009.
   Record created on 31-Aug-1995.
   Database last updated on 6-Apr-2002 02:56:00 EST.

   Domain servers in listed order:

   A.IANA-SERVERS.NET                192.0.34.43
   B.IANA-SERVERS.NET                193.0.0.236
```

例如，如果你想解析出域名服务器的名称及IP地址，可以使用下面的方法：

```
preg_match_all('/^\s*([\S]+)\s+([\d.]+)\s*$/m', $data, $dns,
               PREG_SET_ORDER);

foreach ($dns as $server) {
    print "$server[1] : $server[2]\n";
}
```

为保证取得与域名有关的信息，必须将$server设置为正确的Whois服务器。如果不知道使用哪个服务器，可以查询*whois.internic.net*：

```
require 'Net/Whois.php';

print Net_Whois::query('whois.internic.net','example.org');

require 'Net/Whois.php';

print Net_Whois::query('whois.internic.net','example.org');
[whois.internic.net]

Whois Server Version 1.3

Domain names in the .com, .net, and .org domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.
```

```
   Domain Name: EXAMPLE.ORG
   Registrar: NETWORK SOLUTIONS, INC.
   Whois Server: whois.networksolutions.com
   Referral URL: http://www.networksolutions.com
   Name Server: A.IANA-SERVERS.NET
   Name Server: B.IANA-SERVERS.NET
   Updated Date: 19-aug-2002

>>> Last update of whois database: Wed, 21 Aug 2002 04:56:56 EDT <<<

The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains and
Registrars.
```

其中Whois Server:一行告诉我们，要查询有关*example.org*信息的正确服务器是whois.*networksolutions.com*。

参见

PEAR的Net_Whois类的文档（*http://pear.php.net/package-info.php?package=Net_Whois*）。

第17章

图形

17.0 概述

在GD库的协助下，你可以使用PHP创建动态图像来显示股票报价、展示投票结果、监控系统性能，甚至制作游戏。然而，这与使用Photoshop或GIMP（能够通过拖动鼠标来划一条线）是不同的，使用PHP你必须精确地指定形状的类型、大小和位置。

GD有现成的API，因而PHP也尽力遵循它的语法和函数命名约定。所以，如果你通过其他语言，如C或Perl熟悉GD，那么你就能通过PHP来熟练地使用GD。如果GD对你还是个新概念，那么也只需花很少的时间就能够学会使用它，很快你就能像毕加索一样挥笔涂鸦了。

GD的功能多寡会因你使用的GD版本以及在配置期间启用了哪些选项而有很大的不同。GD可以支持GIF、JPEG、PNG和WBMP格式。GD在读取PNG和JPEG格式的图像时几乎不会有任何质量损失。而且，GD还支持PNG的alpha通道特性，这样就可以为每个像素指定一个透明度级别。

除了能够支持多种文件格式，你还能使用GD绘制各种颜色的像素、直线、矩形、多边形、弧形、椭圆形和圆形。技巧17.1中包含了绘制直线图形的内容。技巧17.2中介绍了绘制曲线图形。如果想知道如何用图案填充图形而不只是使用纯色，见技巧17.3。

还可以使用各种字体绘制文本，包括内置的字体、TrueType和PostScript Type 1字体。技巧17.4中讨论了这三种字体绘制函数的各种细节，而技巧17.5则展示了如何在画布上居中绘制文本。这两技巧又构成了技巧17.6的基础，在技巧17.6中我们将一个图像模板与实时的数据结合起来动态地生成图像，也可以使用GD来生成透明的GIF或PNG图像。而设置透明颜色和在图案中使用透明度的问题我们放在技巧17.7中讨论。

要想从数字照片或其他以EXIF标准保存信息的图像中提取元素数据，请看技巧17.8。

技巧17.9从GD的话题转移到如何通过限制用户访问来安全地提供图像的话题。最后，我们给出了一个例子应用程序，该程序根据投票结果动态生成条形图以显示用户对每个答案的投票比例。

以上所有功能都包含在与PHP 5.1绑定的GD版本中。如果你的版本比较早，也不会有问题。不过，如果某个技巧需要GD的一个特殊版，会在文中给出提示。

GD可以从其官方网站*http://www.boutell.com/gd/*下载。在PHP在线手册的GD部分中也列出了对JPEG和Type 1字体提供支持的其他库的下载位置。不过，PHP开发组最近接手了GD库的开发工作，因此我们希望将来使用PHP与GD会比现在更容易。

如果有必要的话，可以通过两种方式来查看你的服务器中安装了GD的哪个版本，以及它是如何配置的。第一种方式是调用`phpinfo()`。你应该可以在页面顶部的“Configure Command”条目中看到`--with-gd`，页面再往下还有一个标题为“gd”的部分，其中包含了有关安装的是GD的哪个版本以及启用了什么功能的更详细信息。另一种方式是检查`function_exists('imagecreate')`函数的返回值。如果该函数调用返回`true`，说明已经安装了GD。函数`imagetypes()`会返回一些能够指出哪些图形格式有效的字段。要了解更多如何使用这个函数的内容，请参考*http://www.php.net/imagetypes*。如果你想使用的功能没有启用，那么就需要重新配置PHP或者让你的ISP为你提供帮助。

基本的图像生成过程可以分成三步：创建图像、在画布上添加图形和文本、显示或者保存图像。请看下面的例子：

```
$image = ImageCreate(200, 50);
$background_color = ImageColorAllocate($image, 255, 255, 255); // white
$gray             = ImageColorAllocate($image, 204, 204, 204); // gray

ImageFilledRectangle($image, 50, 10, 150, 40, $gray);

header('Content-type: image/png');
ImagePNG($image);
```

以上代码会输出一个位于白色背景上面的灰色矩形，如图17-1所示。

图17-1：位于白色背景上面的灰色矩形

一开始，我们要创建的是一幅图像画布。`ImageCreate()`函数并不返回实际的图像。它只是提供了指向图像对象的句柄 —— 在我们明确地告诉PHP写出图像之前它都不能算是真正的图像。通过`ImageCreate()`可以同时生成多幅图像。

传递给ImageCreate()的参数分别是以像素表示的图形的宽度和高度。在本例中，是200×50像素。如果还想创建新图像，也可以打开一幅已有的图像来编辑。打开图像需要调用ImageCreateFromPNG()或者以类似方式命名的其他函数。文件名就是该函数的唯一参数，而文件可以是本地文件，也可以是位于远程服务器中的文件：

```
// 打开一个本地PNG文件
$graph = ImageCreateFromPNG('/path/to/graph.png');

// 打开一个远程服务器中的JPEG文件
$icon  = ImageCreateFromJPEG('http://www.example.com/images/icon.jpeg');
```

在创建了可编辑的画布对象之后，就可以进一步通过调用ImageColorAllocate()绘制颜色了：

```
$background_color = ImageColorAllocate($image, 255, 255, 255); // 白色
$gray             = ImageColorAllocate($image, 204, 204, 204); // 灰色
```

ImageColorAllocate()函数给一个图像的句柄分配由三个整数表示的颜色。这三个整数的取值范围都是0到255，分别对应着红、绿、蓝三种基本颜色。这种用法同HTML中设置字体或者背景颜色时使用的RGB颜色是一样的。所以，白色就是（255, 255, 255），黑色就是（0,0,0），而其他的颜色都以位于这两者之间的某个三位整数表示。

第一次调用ImageColorAllocate()设置了背景颜色。另外的调用则是为绘制的直线、形状或文本分配颜色。因此，这里我们是将背景颜色设置成（255, 255, 255），然后又通过“ImageColorAllocate($image, 204, 204, 204);”拿起了一支灰色的画笔。你可能会觉得依靠调用ImageColorAllocate()的次序而不是通过单独的函数来决定背景颜色有点古怪，但这正是GD的工作方式，所以PHP也尊重了这一惯例。

调用ImageFilledRectangle()会在画布上放置一个矩形盒子。ImageFilledRectangle()接受了很多参数：要绘制的画布、矩形左上角的x和y坐标、矩形右下角的x和y坐标以及用于绘制图形的颜色（画笔对象）。要让ImageFilledRectangle()在$image上用灰色画一个起点为（50,10），终点为（150,40）的矩形，就需要像下面这样调用：

```
ImageFilledRectangle($image, 50, 10, 150, 40, $gray);
```

与笛卡尔图形不同的是，（0,0）并不是指左下角，而是指左上角。因此，距一个50像素高的画布顶边10像素的点的垂直坐标是10，因为该点位于画布上沿以下10像素。之所以不是40，是因为度量的方向是从上往下，不是从下往上。而且，这个坐标也不是-10，因为向下的距离是正值，而非负值。

至此，图像的绘制已经就绪，可以让它现身。首先，我们发送了一个Content-Type头部

信息，以便让浏览器知道所发送的图像是什么类型。在本例中，是PNG格式。接着，通过PHP使用ImagePNG将PNG图像输出。图像发送完后，任务也完成了：

```
header('Content-Type: image/png');
ImagePNG($image);
```

如果不想把图像发送到浏览器，而是想写入到磁盘中，可以给ImagePNG()提供第二个参数，用于指定保存文件的路径：

```
ImagePng($image, '/path/to/your/new/image.png');
```

因为这次文件不会被发送到浏览器，所以就不需要调用header()。但是要确保指定了路径和相应的图像名，并确保PHP对相应的位置具有写入的权限。

PHP会在脚本执行完毕后清理图像，但是如果你希望手工释放被图像占用的内存，可以调用ImageDestroy($image)来强制让PHP立即释放内存空间。

17.1 绘制直线、矩形和多边形

问题

你想要绘制直线、矩形或多边形，而且想控制矩形或多边形是空心的还是实心的。例如，你希望能够绘制条形图表或股票报价的曲线图。

方案

要绘制直线，使用ImageLine()：

```
ImageLine($image, $x1, $y1, $x2, $y2, $color);
```

要绘制空心矩形，使用ImageRectangle()：

```
ImageRectangle($image, $x1, $y1, $x2, $y2, $color);
```

要绘制实心矩形，使用ImageFilledRectangle()：

```
ImageFilledRectangle($image, $x1, $y1, $x2, $y2, $color);
```

要绘制空心多边形，使用ImagePolygon()：

```
$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImagePolygon($image, $points, count($points)/2, $color);
```

要绘制实心多边形，使用ImageFilledPolygon()：

```
$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImageFilledPolygon($image, $points, count($points)/2, $color);
```

讨论

本方案中涉及到的五个函数的原型都十分类似。它们的第一个参数是要绘制的画布，接下来的一组参数是用于指定GD应该在何处绘制图形的x和y坐标。在ImageLine()中，四个坐标值标出了直线的两个端点，而在ImageRectangle()中，这四个坐标值表示的是矩形的两个对角的位置。例如，ImageLine($image, 0, 0, 100, 100, $color)，会生成一条倾斜的直线。而将相同的坐标值传递给ImageRectangle()则会绘制出四个角分别位于(0,0), (100,0),(0,100) 和 (100,100)的矩形。这两个图形如图17-2所示。

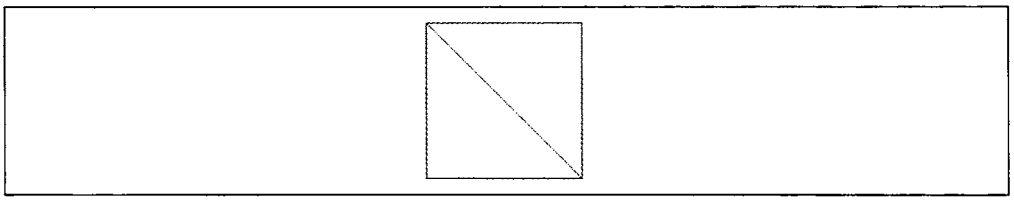

图17-2：斜线和方形

ImagePolygon()函数稍有不同，因为它要接受多个不同的顶点值。因此，第二个参数是一个包含x和y坐标的数组。这个函数会从第一点开始，依次在各个顶点间绘制直线，最终再连接回原来的出发点上完成轮廓的绘制。在绘制多边形时，最少必须指定三个顶点坐标（以总共六个数组元素表示）。第三个参数表示的是该多边形的顶点数——该值是表示顶点坐标的数组元素个数的一半，而动态计算这个值的方式就是count($points)/2，这样你就可以在不修改ImagePolygon()参数的前提下任意添加或减少数组元素，而同时修改顶点个数。

最后，这些函数也都接受一个表示绘制颜色的参数。通常这个值是由ImageColorAllocate()返回的，但是也可以使用常量IMG_COLOR_STYLED或IMG_COLOR_STYLEDBRUSHED。如果你想要绘制虚线，请看技巧17.3。

以上所述的函数绘制的都是空心图形，如果想用指定的颜色填充所绘的形状，以绘制空心图形时使用的同一组参数分别调用mageFilledRectangle()和ImageFilledPolygon()即可。

参见

技巧17.2中有关绘制其他形状的介绍；技巧17.3中通过样式和画笔绘制图形的讨论。

ImageLine()函数的文档（*http://www.php.net/imageline*），ImageRctangle()函数的文档（*http://www.php.net/imagerectangle*），ImagePolygon()函数的文档（*http://www.php.net/imagepolygon*）和ImageColorAllocate()函数的文档（*http://www.php.net/imagecolorallocate*）。

17.2 绘制弧形、椭圆形和圆形

问题

你想绘制空心或实心的曲线形状。例如，你想绘制一个显示用户投票结果的饼形图。

方案

要绘制弧线，使用ImageArc()：

```
ImageArc($image, $x, $y, $width, $height, $start, $end, $color);
```

要绘制椭圆形，使用ImageArc()，并将$start设置为0，$end设置为360：

```
ImageArc($image, $x, $y, $width, $height, 0, 360, $color);
```

要绘制圆形，使用ImageArc()，将$start设置为0，$end设置为360，同时给$width和$height设置相等的值：

```
ImageArc($image, $x, $y, $diameter, $diameter, 0, 360, $color);
```

讨论

由于ImageArc()函数具有高度的灵活性，因而通过传递恰当的值可以很容易地用它来绘制出常见的曲线图形，如椭圆形或圆形。与许多GD函数一样，该函数的第一个参数也是一个画布对象，接下来的两个参数是弧形的圆点坐标，再后面是弧形的宽度和高度值。因为圆形是一种宽度和高度相等的特殊弧形，所以在绘制圆形时，把宽度和高度都设置成想绘制的圆形直径即可。

第六和第七个参数是以度数表示的起点和终点的角度值。0度表示三点钟时针位置，而弧形的绘制方向是顺时针。所以90度也就是六点钟，180度就是九点钟，而270度则表示时针指向正上方（注意：在GD的所有函数中这种行为并不一致。例如，你旋转文本时，旋转的方向就会变成逆时针）。因为弧的圆点位于($x,$y)，如果你要绘制一个从0度到180度的半圆形，弧线的起点并不是($x,$y)而是($x+($diameter/2),$y)。

最后一个参数控制弧形的颜色。

例如，下面绘制的是一个在画布中居中、直径为100像素的黑色空心的圆形，如图17-3左半部分所示：

```
$image = ImageCreate(100,100);
$bg = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageArc($image, 50, 50, 100, 100, 0, 360, $black);
```

要生成实心的椭圆形或圆形，调用ImageFillToBorder()：

```
ImageArc($image, $x, $y, $diameter, $diameter, 0, 360, $color);
ImageFillToBorder($image, $x, $y, $color, $color);
```

ImageFillToBorder()函数会从($x,$y)点开始按照最后一个参数指定的颜色进行填充直到画布边缘或者与第三个参数指定的颜色相同的线为止。

将以上两处代码合并到一起就是：

```
$image = ImageCreate(100,100);
$bg = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageArc($image, 50, 50, 100, 100, 0, 360, $black);
ImageFillToBorder($image, 50, 50, $black, $black);
```

以上代码的输出结果如图17-3右半部分所示。

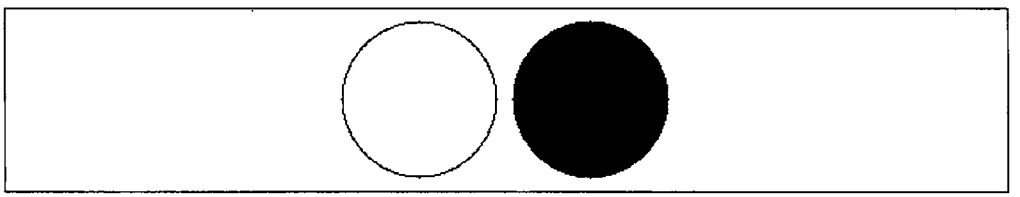

图17-3：黑色的空心圆形和一个以黑色填充的实心圆形

如果使用GD 2.x，可以调用ImageFilledArc()并传递一个表示填充样式的最终参数。GD 2.x还支持专门的ImageEllipse()和ImageFilledEllipse()函数。

参见

技巧17.3中通过样式和画笔绘制图形的讨论；ImageArc()函数的文档（*http://www.php.net/imagearc*），ImageFilledArc()函数的文档（*http://www.php.net/imagefilledarc*）和ImageFillToBorder()函数的文档（*http://www.php.net/imagefilltoborder*）。

17.3 用图案化的线条绘制图形

问题

你想使用各种样式的线条绘制图形，而不是使用默认的实线。

方案

要用图案化的线条绘制图形，可以使用ImageSetStyle()并传递IMG_COLOR_STYLED作为图像的颜色：

```
$black = ImageColorAllocate($image,   0,   0,   0);
$white = ImageColorAllocate($image, 255, 255, 255);

// 生成两个像素宽的黑白相间的虚线
$style = array($black, $black, $white, $white);
ImageSetStyle($image, $style);

ImageLine($image, 0, 0, 50, 50, IMG_COLOR_STYLED);
ImageFilledRectangle($image, 50, 50, 100, 100, IMG_COLOR_STYLED);
```

讨论

线条的图案由颜色数组定义。数组中的每个元素都相当于笔刷中的一个像素。通过让连续的数组元素重复相同的颜色可以使图案中的斑纹尺寸增大。

例如，下面是以交替出现的黑白像素绘制正方形，效果如图17-4所示的左边的正方形：

```
$style = array($white, $black);
ImageSetStyle($image, $style);
ImageFilledRectangle($image, 0, 0, 49, 49, IMG_COLOR_STYLED);
```

下面绘制的是相同大小的正方形，不过是以五个白色像素加五个黑色像素的线条来绘制而成的，效果如图17-4的中间图案所示：

```
$style = array($white, $white, $white, $white, $white,
               $black, $black, $black, $black, $black);
ImageSetStyle($image, $style);
ImageFilledRectangle($image, 0, 0, 49, 49, IMG_COLOR_STYLED);
```

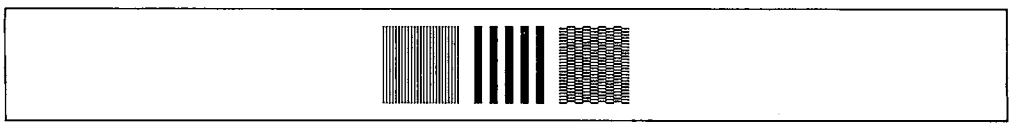

图17-4：以交替的黑白像素线条绘制的三个不同的正方形

即使只使用了白色和黑色两种像素，这些图案看起来也完全不同。

如果笔刷在图形中不能正好绘制整数次，就会导致回绕现象。在前面的例子中，正方形的宽度是50像素。由于第一个笔刷的宽度为2像素，所以正好可以绘制25次；而第二个笔刷的宽度为10像素，正好绘制5次。但是，如果在使用第二个笔刷绘制45×45像素的正方形时，就不会得到像前面一样的直线效果了，而是会得到如图17-4中右边图形的效果：

```
ImageFilledRectangle($image, 0, 0, 44, 44, IMG_COLOR_STYLED);
```

参见

技巧17.1和技巧17.2中有关绘制图形的更多内容；ImageSetStyle()函数的文档（*http://www.php.net/imagesetstyle*）。

17.4 绘制文本

问题

你想把文本绘制成图形。这样可以用来生成动态按钮或者点击计数器。

方案

对于内置的GD字体，使用ImageString()：

```
ImageString($image, 1, $x, $y, 'I love PHP Cookbook', $text_color);
```

对于TrurType字体，使用ImageTTFText()：

```
ImageTTFText($image, $size, 0, $x, $y, $text_color, '/path/to/font.ttf',
             'I love PHP Cookbook');
```

对于PostScript Type 1字体，使用ImagePSLoadFont()和ImagePSText()：

```
$font = ImagePSLoadFont('/path/to/font.pfb');
ImagePSText($image, 'I love PHP Cookbook', $font, $size,
            $text_color, $background_color, $x, $y);
```

讨论

调用ImageString()可以把文本内容放到画布中。与GD的其他绘图函数一样，ImageString()也需要输入很多参数，包括画布、字号、第一个字符右上角位置的x和y坐标、要显示的文本字符串，最后是用于绘制文本的颜色。

在使用ImageString()时，有五种字体尺寸可以选择，从1到5。1是最小号字体，5是最大号字体，效果如图17-5所示。任何比这个5个数字大或者小的值都会被转换成其最接近的合法值。

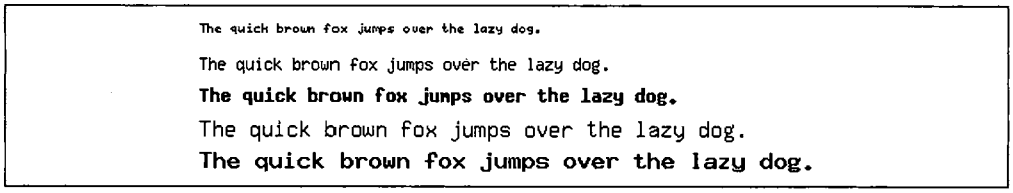

图17-5：内置的GD字体大小

要想垂直而不是水平的绘制文本，可以使用函数 ImageStringUp()。图17-6显示了下面的代码的输出结果：

```
ImageStringUp($image, 1, $x, $y, 'I love PHP Cookbook', $text_color);
```

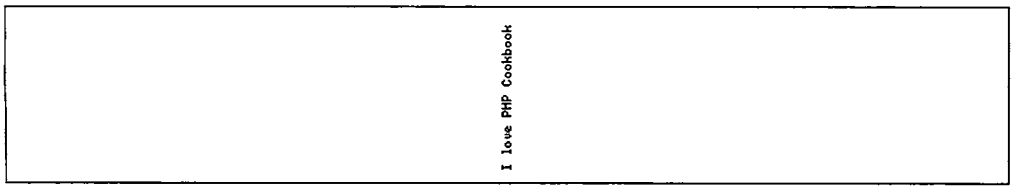

图17-6：垂直文本

如果想使用TrueType字体，必须同时安装FreeType库并在安装PHP时配置好使用FreeType。FreeType库的主要网站是*http://www.freetype.org*。要启用对FreeType 1.x的支持，使用--with-ttf，而对于FreeType 2.x，则需要使用--with-freetype-dir=DIR。

同ImageString()类似，ImageTTFText()也能把字符串绘制到画布中，但它使用的参数略有不同，而且先后次序也不一样：

```
ImageTTFText($image, $size, $angle, $x, $y, $text_color, '/path/to/font.ttf',
             $text);
```

其中，$size参数是以像素表示的字体大小，$angle是旋转的角度值，以度数表示，逆时针为正值。而*/path/to/font.ttf*表示TrueType字体文本的路径。与ImageString()不同的

是，($x,$y)表示的是第一个字符基线左下角位置的坐标（基线，是指多数字符排列的标准线。字符“g”和“j”会向下超出基线，而“a”和“z”则正好位于基线之上）。

使用PostScript Type 1字体需要安装*t1lib*库，该库文件可以从*ftp://sunsite.unc.edu/pub/Linux/libs/graphics/*下载，并使用--with-t1lib在PHP中启用。

这次的情况也一样，即打印文本的语法虽然类似，但仍然不完全相同：

```
$font = ImagePSLoadFont('/path/to/font.pfb');
ImagePSText($image, $text, $font, $size, $text_color, $background_color, $x, $y);
ImagePSFreeFont($font);
```

首先，PostScript字体名不能直接传递到ImagePSText()中，而是必须使用ImagePSLoadFont()加载。在加载成功后，ImagePSLoadFont()会返回一个可供ImagePSText()使用的字体资源。此外，除了指定文本的颜色外，还必须传递背景颜色用于反锯齿处理。这里($x,$y)点的含义与在TrueType库中类似。最后，当使用完一种字体后，可以通过调用ImagePSFreeFont()将其从内存中释放掉。

除了以上所列举的强制性参数外，ImagePSText()还可以接受四个可选的参数，按照先后顺序分别是：间隔、紧缩、角度和平滑幅度。这四个可选的参数要么同时提供，要么一个也不要提供（也就是说，不能只传递一个、两个或者三个可选参数）。第一个参数控制着实际间距的大小（即填充几个空格），第二个参数控制字符间紧缩程度，第三个参数是旋转角度，以度数表示，逆时针为正值，最后一个参数是控制平滑程度的值，该值必须是4或者16。为了达到更好的视觉效果，选择16的效果好于选择4，但可能会牺牲一些图形计算的开销。

在默认情况下，间隔、紧缩和角度的值都是0。这些值如果取大于零的正值，会增大词和字母的间距，或者会使图形逆时针方向旋转。而小于零的负值对词间距、字母间距以及图形旋转的作用则正好相反。下面例子的输出结果显示在图17-7中：

```
// 正常的图像
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y,
            0, 0, 0, 4);

// 单词间距加大
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y + 30,
            100, 0, 0, 4);

// 字母间距加大
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y + 60,
            0, 100, 0, 4);
```

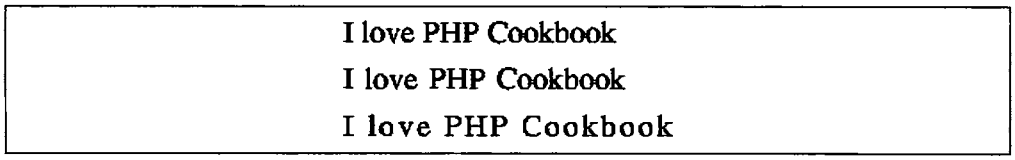

图17-7：加大间距和紧缩距离的文本

参见

技巧17.5中有关绘制居中文本的内容；ImageString()函数的文档（*http://www.php.net/imagestring*），ImageStringUp()函数的文档（*http://www.php.net/imagestringup*），ImageTTFText()函数的文档（*http://www.php.net/imagettftext*），ImagePSText()函数的文档（*http://www.php.net/imagepstext*）和ImagePSLoadFont()函数的文档（*http://www.php.net/imagepsloadfont*）。

17.5 绘制居中的文本

问题

你想在图像的中间位置绘制文本。

方案

先计算出图像的大小和文本盒子的边界位置，再使用这些坐标计算出绘制文本的正确位置。

对于GD的内置字体，可以使用例17-1中的函数pc_ImageStringCenter()。

例17-1：pc_ImageStringCenter()

```
function pc_ImageStringCenter($image, $text, $font) {

    // 字体大小
    $width  = array(1 => 5, 6, 7, 8, 9);
    $height = array(1 => 6, 8, 13, 15, 15);

    // 计算图像大小
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // 计算文本大小
    $xr = $width[$font] * strlen($text);
    $yr = $height[$font];
```

```
    // 计算中心点位置
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi - $yr) / 2);

    return array($x, $y);
}
```

例如：

```
list($x, $y) = pc_ImageStringCenter($image, $text, $font);
ImageString($image, $font, $x, $y, $text, $fore);
```

对于PostScript字体，使用例17-2中的函数pc_ImagePSCenter()。

例17-2：pc_ImagePSCenter()

```
function pc_ImagePSCenter($image, $text, $font, $size, $space = 0,
                          $tightness = 0, $angle = 0) {

    // 计算图像的大小
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // 计算文本的大小
    list($xl, $yl, $xr, $yr) = ImagePSBBox($text, $font, $size,
                                $space, $tightness, $angle);

    // 计算中心点位置
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi + $yr) / 2);

    return array($x, $y);
}
```

例.如：

```
list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
ImagePSText($image, $text, $font, $size, $fore, $back, $x, $y);
```

对于TrueType字体，使用例17-3中的函数pc_ImageTTFCenter()。

例17-3：pc_ImageTTFCenter()

```
function pc_ImageTTFCenter($image, $text, $font, $size) {

    // 计算图像的大小
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // 计算文本的大小
    $box = ImageTTFBBox($size, $angle, $font, $text);

    $xr = abs(max($box[2], $box[4]));
    $yr = abs(max($box[5], $box[7]));
```

```
        // 计算中心点位置
        $x = intval(($xi - $xr) / 2);
        $y = intval(($yi + $yr) / 2);

        return array($x, $y);
    }
```

例如：

```
list($x, $y) = pc_ImageTTFCenter($image, $text, $font, $size);
ImageTTFText($image, $size, $angle, $x, $y, $fore, $font, $text);
```

讨论

以上三个方案中的函数都返回要绘制图像的x和y坐标。当然，根据字体、大小和设置的不同，用于计算相应坐标的方法也各异。

对于PostScript Type 1字体，需要给pc_ImagePSCenter()传递一个由ImageCreate()指定的图像(或者它的朋友之一)以及说明如何绘制文本的几个参数。有三个参数是必须的：要绘制的文本、字体以及字体大小。后面三个参数则是可选的：间距、紧缩和旋转角度。

在这个函数内部，使用ImageSX()和ImageSY()找到画布的尺寸，这两个函数分别返回图形的宽度和高度。然后，调用ImagePSBBox()，这个函数返回了四个整数值：分别是文本最左下角位置的x、y坐标和文本最右上角位置的x、y坐标。因为坐标值是相对于文本基线的，所以这些值通常不会为0。例如，小写的字母“g”会垂直伸到其他字母的下方，所以在这种情况下，左下角的y坐标值就是负值。

在取得了以上六个值后，就可以计算中心点的正确位置了。因为画布的左上角坐标是(0,0)，但ImagePSText()是以左下角为参照的，因此计算$x和$y的公式并不相同。对于$x，画布宽度与文本宽度的差就是文本外面空白区域的宽度，而这个宽度除以2就是文本距画布左边的距离。同样的道理，对于$y，只是$yi和$yr相加的关系，通过相加可以得到边框远边的坐标，由于GD中计算y坐标的反向方式决定了这里必须这样计算。

在以上计算中，我们有意地忽略了左下角位置的坐标。这是因为大部分文本都位于基线以上，如果在计算中心点的算法中加入了降低文本位置的像素，实际上会损害代码的有效性。因为从视觉上来看，最终文本会出现在中心点以下。

为了计算文本的居中位置，我们把以上分析综合到一起就是：

```
function pc_ImagePSCenter($image, $text, $font, $size, $space = 0,
                          $tightness = 0, $angle = 0) {
```

```
    // 计算图像的大小
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // 计算文本的大小
    list($xl, $yl, $xr, $yr) = ImagePSBBox($text, $font, $size,
                                    $space, $tightness, $angle);

    // 计算中心点位置
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi + $yr) / 2);

    return array($x, $y);
}

$image = ImageCreate(500,500);
$text = 'PHP Cookbook Rules!';
$font = ImagePSLoadFont('/path/to/font.pfb');
$size = 20;
$black = ImageColorAllocate($image, 0, 0, 0);
$white = ImageColorAllocate($image, 255, 255, 255);

list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
ImagePSText($image, $text, $font, $size, $white, $black, $x, $y);
ImagePSFreeFont($font);

header('Content-type: image/png');
ImagePng($image);

ImageDestroy($image);
```

遗憾的是，这个例子对于GD内置的字体及TrueType字体都不适用，没有能够返回使用内置字体的文本字符串大小的函数，而ImageTTFBBox()返回的也是八个值而非四个。不过，经过一番调整，还是能够解决这些问题的。

因为内置字体的宽度是固定不变的，所以很容易编写一个函数，它根据对每个字符大小的度量最终基于文本的长度返回文本的大小。表17-1虽然不是100%精确，但实际返回结果的误差应该在1到2个像素之间，而这对于大多数情况已经足够了。

表17-1：内置的GD字体字符大小

字号	宽度	高度
1	5	6
2	6	8
3	7	13
4	8	15
5	9	15

在pc_ImageStringCenter()函数内部，字符串的长度是每个字符宽度乘以字符数，而高度等于一个字符的高度。注意，ImageString()将字符串的y坐标值看成是文本以上部分空白区域的高度，所以应该在计算$y的值时把加号换成减号。

以下是使用全部五种内置字体水平居中绘制文本的例子：

```
$text = 'The quick brown fox jumps over the lazy dog.';
for ($font = 1, $y = 5; $font <= 5; $font++, $y += 20) {
    list($x, $y) = pc_ImageStringCenter($image, $text, $font);
    ImageString($image, $font, $x, $y, $text, $color);
}
```

输出结果如图17-8所示。

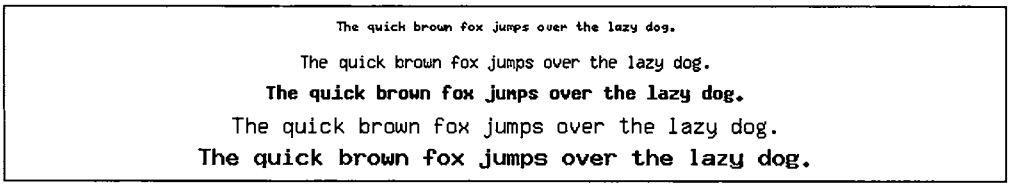

图17-8：居中GD内置的字体

对TrueType字体，需要使用ImageTTFBBox()或者更新的ImageFtBBox()（函数名中带TTF的适用于FreeType 1.x版，而带Ft的则适用于FreeType 2.x版）。该函数返回八个数值：分别是文本四个角的（x,y）坐标，顺序是从左下角逆时针方向旋转，即第二个角是右下角，依此类推。

在编写pc_ImageTTFCenter()时，以pc_ImagePSCenter()为基础，代码如下：

```
// 计算文本的大小
list($xl, $yl, $xr, $yr) = ImagePSBBox($text, $font, $size,
                                       $space, $tightness, $angle);
```

换成：

```
// 计算文本的大小
$box = ImageTTFBBox($size, $angle, $font, $text);

$xr = abs(max($box[2], $box[4]));
$yr = abs(max($box[5], $box[7]));
```

下面是一个使用pc_ImageTTFCenter()的例子：

```
list($x, $y) = pc_ImageTTFCenter($image, $text, $font, $size);
ImageTTFText($image, $size, $angle, $x, $y, $white, $black,
             '/path/to/font.ttf', $text);
```

参见

技巧17.4中更多与绘制文本有关的内容和技巧17.5中更多与居中文本有关的内容；ImageSX()函数的文档（*http://www.php.net/imagesx*），ImageSY()函数的文档（*http://www.php.net/imagesy*），ImagePSBBox()函数的文档（*http://www.php.net/imagepsbbox*），ImageTTFBBox()函数的文档（*http://www.php.net/imagettfbbox*）和ImageFtBBox()函数的文档（*http://www.php.net/imageftbbox*）。

17.6 生成动态图像

问题

你想基于现有的图像模板和动态数据（通常是文本）创建一幅图像。例如，你想生成一个点击计数器。

方案

载入模板图像，找到正确位置以便适当地居中文本，把文本添加到画布中，最后将图像发送到浏览器：

```
// 配置设置
$image    = ImageCreateFromPNG('button.png');
$text     = $_GET['text'];
$font     = ImagePSLoadFont('Times');
$size     = 24;
$color    = ImageColorAllocate($image,   0,   0,   0); // 白色
$bg_color = ImageColorAllocate($image, 255, 255, 255); // 黑色

// 输出居中的文本
list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
ImagePSText($image, $text, $font, $size, $color, $bg_color, $x, $y);

// 发送图像
header('Content-type: image/png');
ImagePNG($image);

// 清理内存
ImagePSFreeFont($font);
ImageDestroy($image);
```

讨论

通过GD来生成动态图像是很容易的，你要做的就是把我们介绍过的一些技巧组合到一起。在本方案代码中的开始处，首先从保存的模板按钮中载入一幅图像，这幅图像将作

为文本的背景。我们将来自查询字符串的内容直接作为动态生成的文本，当然也可以使用从数据库（在这个例子中是访问计数器表）或者远程服务器（股票报价或者天气预报信息）中提取的字符串。

然后，继续设置其他选项：载入字体并设置其大小、颜色以及背景颜色。在输出文本之前，必须计算它的位置 —— 技巧17.5中的pc_ImagePSCenter()函数可以非常完美地解决这个问题。最后，绘制图像并释放字体和图像占用的内存空间。

例如，下面的代码生成了一个HTML页面和基于动态按钮的图像标签，结果如图17-9所示：

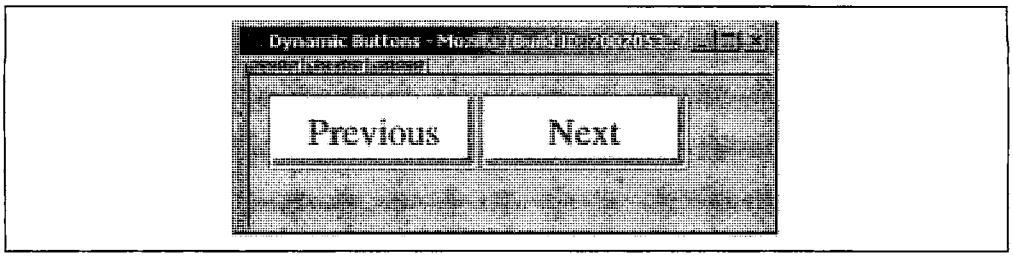

图17-9：按钮示例页面

```
<?php
if (isset($_GET['button'])) {

    // 配置设置
    $image    = ImageCreateFromPNG('button.png');
    $text     = $_GET['button'];      // dynamically generated text
    $font     = ImagePSLoadFont('Times');
    $size     = 24;
    $color    = ImageColorAllocate($image,   0,   0,   0); // 黑色
    $bg_color = ImageColorAllocate($image, 255, 255, 255); // 白色

    // 输出居中的文本
    list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
    ImagePSText($image, $text, $font, $size, $color, $bg_color, $x, $y);

    // 发送图像
    header('Content-type: image/png');
    ImagePNG($image);

    // 清理内存
    ImagePSFreeFont($font);
    ImageDestroy($image);

    $url = htmlentities($_SERVER['PHP_SELF']);

} else {
?>
<html>
```

```
<head>
    <title>Sample Button Page</title>
</head>
<body>
    <img src="<?php echo $url; ?>?button=Previous"
         alt="Previous" width="132" height="46">
    <img src="<?php echo $url; ?>?button=Next"
         alt="Next"     width="132" height="46">
</body>
</html>
<?php
}
?>
```

在这个脚本当中，如果向$_GET['button']中传递了一个值，就能据以生成一个按钮并作为PNG图像发送到浏览器中。如果$_GET['button']没有设置，则会输出基本的HTML页面，其中包含对生成按钮图像（一个Previous按钮，一个Next按钮）脚本的调用。常见的方案是编写一个单独的*button.php*页面，这个页面只负责返回图像内容并将图像来源指向该页面。

参见

技巧17.4中有关绘制文本的更多内容和技巧17.5中有关居中文本的讨论。Rasmus Lerdorf, Kevin Tatroe 和 Peter MacIntyre合著的《Programming PHP》（O'Reilly）的第9章“图形”。

17.7 取得并设置透明颜色

问题

你想将一幅图像中的某种颜色设置为透明的。当图像覆盖在背景上时，背景会透过图像的透明区域显示出来。

方案

使用ImageColorTransparent()：

```
$color = ImageColorAllocate($image, $red, $green, $blue);
ImageColorTransparent($image, $color);
```

讨论

GIF和PNG格式都支持透明颜色，但是JPEG格式不支持透明颜色。在GD中指定透明颜

色，需要使用常量IMG_COLOR_TRANSPARENT。例如，下面显示了如何绘制黑色和透明相间的虚线：

```
// 生成两像素宽的黑白相间的虚线
$style = array($black, $black, IMG_COLOR_TRANSPARENT, IMG_COLOR_TRANSPARENT);
ImageSetStyle($image, $style);
```

要找到当前的透明设置，先取得ImageColorTransparent()的返回值并将其传递给ImageColorsForIndex()：

```
$transparent = ImageColorsForIndex($image, ImageColorTransparent($image));
print_r($transparent);

$transparent = ImageColorsForIndex($image, ImageColorTransparent($image));
print_r($transparent);
Array
(
    [red] => 255
    [green] => 255
    [blue] => 255
)
```

ImageColorsForIndex()函数返回了包含红、绿、蓝三种颜色值的数组。在上面的例子中，透明颜色是白色的。

参见

ImageColorTransparent()函数的文档（*http://www.php.net/imagecolortransparent*）和ImageColorsForIndex()函数的文档（*http://www.php.net/imagecolorsforindex*）。

17.8 读取EXIF数据

问题

你想从一幅图像文件中提取出元数据。根据这些元数据就能知道该照片的拍摄时间、图像大小以及MIME类型等信息。

方案

使用exif_read_data()函数：

```
$exif = exif_read_data('/beth-and-seth.jpeg');
```

```
print_r($exif);
Array
(
    [FileName] => beth-and-seth.jpg
    [FileDateTime] => 1096055414
    [FileSize] => 182080
    [FileType] => 2
    [MimeType] => image/jpeg
    [SectionsFound] => APP12
    [COMPUTED] => Array
        (
            [html] => width="642" height="855"
            [Height] => 855
            [Width] => 642
            [IsColor] => 1
        )

    [Company] => Ducky
    [Info] =>
)
```

讨论

可交换图像文件格式（EXIF，Exchangeable Image File Format）是一种在照片中嵌入元数据的标准。大多数数码相机都使用EXIF，因此这种格式也日益成为一种在照片共享网站，如Flickr中提供丰富数据的方式。

PHP有许多EXIF函数。虽然不需要扩展库，但必须通过传递`--enable-exif`配置标记来启用这些函数。

提取数据的最简单方法就是使用`exif_read_data()`。该函数返回一个元数据数组，包括照片的创建日期、MIME类型（有助于向其他程序提供图像）以及图像分辨率等信息：

```
$exif = exif_read_data('/beth-and-seth.jpeg');

print_r($exif);
Array
(
    [FileName] => beth-and-seth.jpg
    [FileDateTime] => 1096055414
    [FileSize] => 182080
    [FileType] => 2
    [MimeType] => image/jpeg
    [SectionsFound] => APP12
    [COMPUTED] => Array
        (
            [html] => width="642" height="855"
            [Height] => 855
            [Width] => 642
```

```
            [IsColor] => 1
        )

    [Company] => Ducky
    [Info] =>
)
```

使用html值可以直接将信息嵌入到<img>源标签中。

也可以使用这些EXIF函数取得与相应照片关联的缩略图。为此，调用exif_thumbnail()：

```
$thumb = exif_thumbnail('beth-and-seth.jpeg', $width, $height, $type);
```

exif_thumbnail()函数接受四个参数。第一个是文件名，后面三个是将要保存的宽度、高度和图像类型。该函数会以二进制字符串形式返回缩略图，如果失败则返回false。

如果想直接将图像显示出来，需要使用image_type_to_mime_type()来取得正确的MIME类型。将该MIME类型作为HTTP头部传递，并显示该图像：

```
$thumb = exif_thumbnail('beth-and-seth.jpeg', $width, $height, $type);

if ($thumb != false) {
    $mime = image_type_to_mime_type($type);
    header("Content-type: $mime");
    print $image;
}
```

或者，也可以创建一个<img>标签显示图像：

```
$file = 'beth-and-seth.jpeg';
$thumb = exif_thumbnail($file, $width, $height, $type);

if ($thumb != false) {
    $img = "<img src=\"$file\" alt=\"Beth and Seth\"
                width=\"$width\" height=\"$height\" />
    print $img;
}
```

参见

exif_read_data()函数的文档（*http://www.php.net/exif-read-data*）和exif_thumbnail()函数的文档（*http://www.php.net/exif-thumbnail*）。

17.9 安全地提供图像

问题

你想要对谁能够查看哪些图像进行控制。

方案

不要把图像保存在你的文档根目录中，把它们保存到其他地方。在交付某个图像文件时，再手工将其打开并传递给浏览器：

```
header('Content-Type: image/png');
readfile('/path/to/graphic.png');
```

讨论

本方案中的第一行向浏览器发送了Content-Type头部信息，因而浏览器会知道接收到的是什么类型的对象进而可以正确地显示。第二行代码从其他磁盘（或者从远程URL）中读取文件，并将读取的内容直接发送给浏览器，然后关闭文件。

提供图像的典型方式是使用<img>标签，将其src属性指向你网站中的某个文件。如果你想保护这些文件，可能有必要使用某种形式的密码验证。有关HTTP的Basic和Digest认证方式，在技巧8.9中曾经讨论过。

然而，典型的方式不一定是最佳方式。首先，如果你既想限制人们能够访问的图像文件，又不想通过用户名和密码验证这种复杂的办法，那你该怎么办？一种选择是只对相应的文件提供链接，如果用户无法点击链接就看不到这些文件。然而，他们有可能会用书签标记旧文件，或者也有可能根据你的命名规律推测其他的文件名，并在浏览器中键入URL尝试访问你不想提供的那些文件。

如果你的内容受限时，那么你肯定不希望人们能够猜到命名规律并由此看到你不想提供的图像。当信息受限时，会有选定的一组人，通常是记者，得到预览版，以便这些人能够撰写相关主题的内容并准备好在限制撤销时发表。为解决这一问题，应该做到只把合法的内容放在文档根目录中，但这样会导致在目录之间频繁地来回复制删除文件。不过，也可以把所有文件都放在一个固定的地方，而只对通过你代码检验的人交付相应的文件。

例如，我们假设你签了一份关于出版公司在你的网站上发布它们喜剧作品的合同。但是，它们不想让你创建真实的文档，而你同意让用户只能看到最后两周剪辑。对于其他

内容，用户需要到官方网站上才能看到。同样地，你可能会在喜剧作品发表前得到相关的内容，但你不想让人们可以随意看完所有内容，而是想让读者每天都光顾你的网站。

下面给出解决方案。文件按收到的日期命名，以便根据日期就能知道文件的先后关系。现在锁定超过14天以上的内容，并使用下面的代码：

```
// 如果喜剧内容没有超过14天并且也不是将来要显示的内容则显示

// 计算当前日期
list($now_m,$now_d,$now_y) = explode(',',date('m,d,Y'));
$now = mktime(0,0,0,$now_m,$now_d,$now_y);

// 为保证任何一边解决夏令时问题，加上两个小时
$min_ok = $now - 14*86400 - 7200; // 14 天以前
$max_ok = $now + 7200;            // 今天

$mo = (int) $_GET['mo'];
$dy = (int) $_GET['dy'];
$yr = (int) $_GET['yr'];

// 取得请求的喜剧内容的时间戳
$asked_for = mktime(0,0,0,$mo,$dy,$yr);

// 比较日期
if (($min_ok > $asked_for) || ($max_ok < $asked_for)) {
    echo 'You are not allowed to view the comic for that day.';
} else {
    header('Content-type: image/png');
    readfile("/www/comics/{$mo}{$dy}{$yr}.png");
}
```

参见

技巧23.5中更多有关读取文件的内容。

17.10 编程：根据投票结果生成条形图

使用彩色的条形图表显示投票结果会比单纯的文字更有冲击力。例17-4中的函数使用GD创建了根据测验问题的累计响应数显示的柱状图。

例17-4：条形图

```
function pc_bar_chart($question, $answers) {

    // 定义绘制条形的颜色
    $colors = array(array(255,102,0), array(0,153,0),
                    array(51,51,204), array(255,0,51),
                    array(255,255,0), array(102,255,255),
                    array(153,0,204));
```

```
$total = array_sum($answers['votes']);

// 定义一些间隔和其他相关数值（包括魔术数值$scale）
$padding = 5;
$line_width = 20;
$scale = $line_width * 7.5;
$bar_height = 10;

$x = $y = $padding;

// 分配较大的区域绘制图形，因为无法提前知道图形的长度
$image = ImageCreate(150, 500);
$bg_color = ImageColorAllocate($image, 224, 224, 224);
$black = ImageColorAllocate($image, 0, 0, 0);

// 输出问题
$wrapped = explode("\n", wordwrap($question, $line_width));
foreach ($wrapped as $line) {
    ImageString($image, 3, $x, $y , $line, $black);
    $y += 12;
}

$y += $padding;

// 输出答案
for ($i = 0; $i < count($answers['answer']); $i++) {

    // 格式化百分比
    $percent = sprintf('%1.1f', 100*$answers['votes'][$i]/$total);
    $bar = sprintf('%d', $scale*$answers['votes'][$i]/$total);

    // 得到颜色
    $c = $i % count($colors); // 处理条形比colors更多的情况
    $text_color = ImageColorAllocate($image, $colors[$c][0],
                                     $colors[$c][1], $colors[$c][2]);

    // 绘制条形图及百分比数值
    ImageFilledRectangle($image, $x, $y, $x + $bar,
                         $y + $bar_height, $text_color);
    ImageString($image, 3, $x + $bar + $padding, $y,
                "$percent%", $black);

     $y += 12;

     // 输出答案
     $wrapped = explode("\n", wordwrap($answers['answer'][$i], $line_width));
     foreach ($wrapped as $line) {
         ImageString($image, 2, $x, $y, $line, $black);
         $y += 12;
     }

     $y += 7;
 }
```

```
    // 通过拷贝获取图像
    $chart = ImageCreate(150, $y);
    ImageCopy($chart, $image, 0, 0, 0, 0, 150, $y);

    // 发送图像
    header ('Content-type: image/png');
    ImagePNG($chart);

    // 清理内存空间
    ImageDestroy($image);
    ImageDestroy($chart);
}
```

在调用这个程序时，需要创建一个包含两个平行数组的数组：$answers['answer']和$answer['votes']。每个数组的$i元素保存着答案的文本内容和对答案$i的投票总数。图17-10显示了这个例子的输出结果：

```
// Act II. Scene II.
$question = 'What a piece of work is man?';

$answers['answer'][] = 'Noble in reason';
$answers['votes'][]  = 29;

$answers['answer'][] = 'Infinite in faculty';
$answers['votes'][]  = 22;

$answers['answer'][] = 'In form, in moving, how express and admirable';
$answers['votes'][]  = 59;

$answers['answer'][] = 'In action how like an angel';
$answers['votes'][]  = 45;

pc_bar_chart($question, $answers);
```

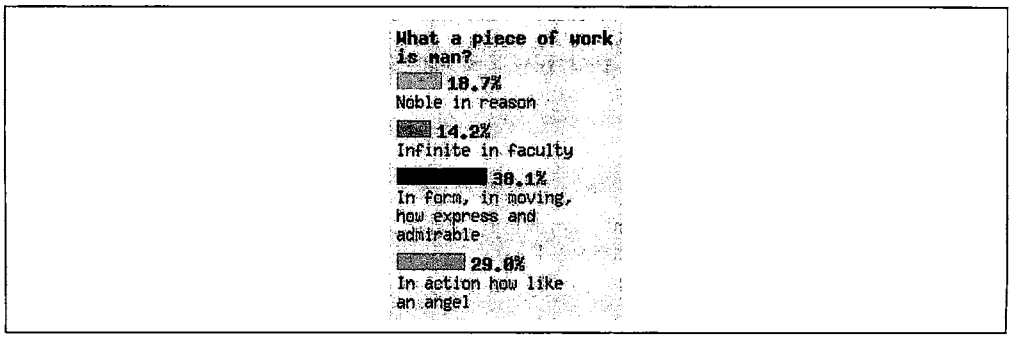

图17-10：投票结果的条形图

在这个例子当中，投票的答案是手工填写的，而对于真实的投票来说，这些数据应该来自数据库中。

这个程序是一个良好的开端，不过因为使用的是内置GD字体，所以程序中包含了一些与字体高度和宽度对应的魔术数值（magic number）。同样地，每个答案之间的间隔采用的也是硬编码方式。如果为了使用更高级的字体，如PostScript或TrueType，需要对这些地方进行修改，必须更新控制那些数值的算法。

在这个函数的顶部，定义了一大堆RGB值的组合，这些颜色都是用于绘制条形图的。而且还定义好了几个常数备用，如`$line_width`，用于限制每行中字符的最大数目。`$bar_height`变量决定条形图的高度，而`$scale`是根据最长行的长度来计算的条形图的长度。`$padding`则用于把结果和画布之间分开5像素大小的间隔。

然后，我们创建一个非常大的画布以便绘制图表。后面，我们会对画布进行适当地缩小处理，但不可能提前知道整个图像的大小是多少。条形图默认的背景颜色为（224, 224, 224），是一种亮灰色。

为了将图表的宽度限制在合理的范围内，我们使用了`wordwrap()`把`$question`分成几段并通过`explode()`加上换行符`\n`。结果是一个尺寸适当的多行数组，这样就可以通过循环每次打印其中一行了。

在输出完问题之后，下一个要处理的就是答案。首先，我们使用`sprintf()`对结果数值进行格式化处理。为了把针对某个答案投票的总百分比值格式化为带小数点的浮点型数，使用`%1.1f`作为参数。在计算与该数值对应的条形的长度时，计算的是一个近似值，这里没有乘以100，而是乘以一个魔术数值`$scale`，并返回一个整数值。

文本的颜色取自包含RGB三个值的`$color`数组。随后调用`ImageFilledRectangle()`绘制条形，调用`ImageString()`在相应的条形右侧绘制百分比文本。在添加了一些内边距后，再使用与打印问题相同的规则输出答案。

在打印完答案后，条形图的整体尺寸保存在了`$y`中。现在就可以对整个图形的大小进行适当裁剪了，但是没有`ImageCrop()`函数。为了解决这个问题，我们根据前面的恰当尺寸重新创建了一个新画布并使用`ImageCopy()`把位于原来画布之上的图形拷贝出来放在其中。然后，使用`ImagePNG()`将尺寸正确的图像保存为PNG图像，并调用两次`ImageDestroy()`完成内存释放。

我们在本节一开始提到过，这只是输出条形图的一个蜻蜓点水式的函数。它能够解决一些问题，如折行，但仍然不够完美。例如，它的自定义功能不够丰富。许多设置都直接写在了代码中。不过，这个函数还是演示了如何通过把多种GD功能综合到一起来创建有用的图形应用程序的过程。

第18章

安全和加密

18.0 概述

无论是创建web应用程序的开发者还是企图利用web应用程序的攻击者，对于web程序安全这个话题都给予了越来越多的关注。作为一名PHP程序的开发者，一定要知道你的应用程序也会成为众多攻击的目标，并提前有所准备。

很多web应用程序中的安全问题都是由于轻信了第三方提供的数据所造成的。比如对于输入数据，在对其进行验证之前都应该将其视为嫌疑数据。如果是把嫌疑数据发送给用户浏览器，就可能导致跨站点脚本（XSS）问题。如果把嫌疑数据用于SQL查询，那么有可能会造成SQL注射问题。技巧18.5中讨论了如何避免这些问题。

在使用第三方提供的数据，包括你的用户提供的数据时，首先检验其合法性非常重要。这个过程叫做过滤。技巧18.3介绍了如何保证对所有输入的数据进行过滤处理。

过滤输入和转义输出并不能解决所有的安全问题。像技巧18.1中讨论的session定置攻击，会导致受骗者使用攻击者选定的session标识符实施某些行为。而技巧18.1中讨论的另一种攻击行为，伪造跨站点请求，则会致使受骗者发送攻击者指定的请求。

与安全密切相关的，能够增强你的应用程序安全性的强大手段是加密。不过，与其他任何工具一样，加密手段也必须运用得当。

加密的本质就是扰乱数据。某些不可恢复的数据扰乱，称为单向加密或者散列算法。另一种双向加密方式既能对数据加密，而且也能对加密后的数据进行解密。

PHP中提供了通过加密来保障数据安全的很多工具。其中一些工具，如`md5()`函数，属于PHP的基本函数。而其他一些扩展工具（例如：*mcrypt*，*mhash*和cURL）则需要在PHP编译时明确包含进来。

技巧18.7讨论了如何使用md5()函数。该函数最常用于对密码进行加密。

*mcrypt*是一种功能更全的加密库，它提供了多种不同的算法和加密模式。由于它支持多种不同的加密方式，所以特别适合于与其他系统或者非PHP程序交换加密数据。技巧18.10对*mcrypt*进行了详细地介绍。

虽然PHP为我们提供了对数据进行有效加密的各种工具，但加密只不过是纷繁复杂的安全蓝图中一个环节而已。加密的数据可以通过密钥（key）进行解密，所以保护密钥非常重要。如果非授权用户能够访问到你的密钥（比如，该密钥保存在了web服务器能够访问的文件或者其他用户能够访问的共享主机环境中），那么不论你选择的加密算法有多可靠，你的数据同样会面临安全问题的威胁。

对于敏感的数据，不仅需要在服务器端提供保护，而且当在服务器与用户之间传送这些数据时也需要进行保护。通过常规的HTTP发送数据对于处在你的服务器和用户之间网络中任何一个节点的任何人来说都是可见的。技巧18.13讨论了如何通过SSL（安全套接字协议层）避免网络窃取程序注意到你所传送的数据。要全面了解有关PHP程序安全问题的内容，请阅读 Chris Shiflett 著的《PHP applications》（O'Reilly）。

18.1 预防Session定置

问题

你想确保用户的session标识符不会由第三方提供，例如劫持了用户session的攻击者。

方案

只要用户的授权范围改变，如成功登录后，就通过session_regenerate_id()来重新生成session标识符：

```
<?php

session_regenerate_id();
$_SESSION['logged_in'] = true;

?>
```

讨论

可以通过session实现在不同请求之间的会话状态持续。为了保证session有效，用户的每一次请求都必须包含一个能够唯一标识一次会话的session标识符。

在默认情况下，PHP可以接受来自cookie或URL中的session标识符。而攻击者可能会欺骗受害人点击一个包含session标识符并指向你应用程序的链接：

```
<a href="http://example.org/login.php?PHPSESSID=1234">Click Here!</a>
```

点击了该链接的用户，其session标识符会被重置为1234。因此，攻击者在知道了这个用户的session标识符后，就可以通过使用相同的session标识符来尝试劫持用户的会话。

如果这个用户没有登录或者没有实施与你应用程序中其他用户不同的操作，那么攻击者将一无所获。因此，通过确保只要改变用户的授权范围就重新生成session标识符，可以有效地消除session定置攻击。由于PHP会自动更新存储的session数据并传送新的session标识符，所以必须在适当的时候调用这个函数。

参见

技巧11.2中有助于防止劫持和定置的更多session相关的信息。技巧11.3中讨论的基于时间重新生成session ID的方案。

18.2 防止表单提交骗术

问题

你想确保表单的提交是合法的也是有意识的。

方案

向表单中添加一次性记号，并将该记号保存在用户的session中：

```
<?php

session_start();

$_SESSION['token'] = md5(uniqid(mt_rand(), true));

?>

<form action="buy.php" method="POST">
<input type="hidden" name="token" value="<?php echo $_SESSION['token']; ?>" />
<p>Stock Symbol: <input type="text" name="symbol" /></p>
<p>Quantity: <input type="text" name="quantity" /></p>
<p><input type="submit" value="Buy Stocks" /></p>
</form>
```

当你得到提交表单的请求时，检查该记号并确保匹配：

```
<?php

session_start();

if ($_POST['token'] != $_SESSION['token'] ||
    !isset($_SESSION['token'])) {
    /* 提示用户输入密码 */
} else {
    /* 继续 */
}

?>
```

讨论

这种技术会防止一组被称为伪造跨站点请求（CSRF，Cross-site request forgeries）的攻击。这些攻击都是在受骗者不知情的情况下，让受骗者向某个目标站点发送请求来达到攻击目的的。通常，受骗者都拥有对目标站点一定程度的授权，所以这些攻击能够实现攻击者以其他方式无法实施的动作。

这种在表单中添加记号的方式不会阻止用户自己的请求，但这也是我们不能、也不该阻止的。如果你像技巧18.3中所介绍的对输入采取了过滤手段，就可以强制请求遵守你的规则。本技巧中介绍的这种技术只是确保请求是用户有意做出的。

18.3 确保过滤输入

问题

你需要确保在使用所有输入数据之前先进行过滤。

方案

初始化一个空数组用于保存过滤后的数据。在验证输入有效后，将输入保存到这个数组中：

```
<?php

/* 初始化一个用于保存过滤后数据的数组 */
$clean = array();

/* 允许名字中包含字母 */
if (ctype_alpha($_POST['name'])) {
```

```
    $clean['name'] = $_POST['name'];
} else {
    /* 错误 */
}

?>
```

讨论

通过使用严格的命名约定，可以更容易地保持过滤后数据的合法性。而始终都使用初始化的空数组$clean，确保了数据无法被注射到数组中 —— 因为只有你才能明确地添加数组元素。

如果你打算采取像$clean空数组这样的技术，那么关键的一点就是要保证在你的业务逻辑中只使用该数组中的数据。

参见

技巧9.2～9.9中有关对不同类型的表单输入数据进行验证的详细介绍。

18.4 避免跨站点脚本

问题

你需要安全地消除跨站点脚本（XSS）对你的PHP应用程序的攻击。

方案

通过htmlentities()对所有HTML输出进行转义，并保证指明正确的字符编码：

```
<?php

/* 注意使用正确的字符编码 */
header('Content-Type: text/html; charset=UTF-8');

/* 为保存转义后的数据初始化一个数组 */
$html = array();

/* 转义过滤后的数据 */
$html['username'] = htmlentities($clean['username'], ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}.</p>";

?>
```

讨论

htmlentities()函数会用HTML实体替换内容中可能存在的每一个字符。例如，“>”会被替换成“>”。虽然直接的效果是修改了数据，但转义的目的是为了在不同的环境中保持数据的内容不变。无论何时，当浏览器解析作为HTML代码的“>”时，屏幕上显示的都是“>”。

XSS攻击试图利用由第三方提供的包含HTML代码的数据不会被适当转义的漏洞。聪明的攻击者可能会利用这一漏洞向你的用户提供一些代码，当浏览器执行这些代码时则会导致严重问题。通过使用htmlentities()，就可以确保包括第三方数据在内的输出都能得到适当地显示，并且不会被解释执行。

参见

技巧9.10中在提交表单数据的情况下如何避免跨站点脚本的讨论。

18.5 避免SQL注入

问题

你需要在自己的PHP应用程序中消除SQL注入的漏洞。

方案

使用诸如PDO之类的数据库来针对你的数据库进行适当的转义：

```
<?php

$db = new PDO('mysql:host=localhost;dbname=users',
              $_SERVER['DB_USER'],
              $_SERVER['DB_PASSWORD']);

$statement = $db->prepare("INSERT
                           INTO   users (username, password)
                           VALUES (:username, :password)");

$statement->bindParam(':username', $clean['username']);
$statement->bindParam(':password', $clean['password']);

$statement->execute();

$db = NULL;

?>
```

讨论

使用绑定参数能够保证除了原始数据之外的其他数据都不会被接收，因此没什么值可能修改SQL查询语句的格式。

如果不能使用PDO，那么可以使用以PHP编写的其他数据库，如PEAR::DB，这个库也提供了相似的特性：

```
<?php

$st = $db->query('INSERT
            INTO   users (username, password)
            VALUES (?, ?)',
            array($clean['username'], $clean['password']));

?>
```

虽然这种方法仍然将数据与SQL查询混合在一起，但PEAR::DB能够保证对数据进行适当地引用和转义，因此事实上消除了SQL注入的问题。

参见

第10章中有关PDO的更多信息，特别是技巧10.6和技巧10.7。PDO的文档（*http://www.php.net/pdo*），PEAR::DB的文档（*http://pear.php.net/manual/en/package.database.db.php*）。

18.6 将密码置于站点文件外部

问题

你需要使用密码连接到一个数据库，但不想把这个密码放在你使用的站点中的PHP文件里，以防止由于那些文件暴露而丢失密码。

方案

将密码保存到web服务器启动时加载的某个文件中的环境变量中。然后，只需在代码中引用那个环境变量即可：

```
<?php

mysql_connect('localhost', $_SERVER['DB_USER'], $_SERVER['DB_PASSWORD']);

?>
```

讨论

虽然这一技术从你页面的代码中移走了密码，但却把密码放到了其他需要保护的地方。因此最重要的是，要保证能被公众查看的页面不会调用phpinfo()函数。因为phpinfo()会显示所有的环境变量，因而也就会暴露保存在其中的密码。同样地，也要保证不以其他方式暴露$_SERVER，比如通过print_f()函数。

还有，特别是当你在使用共享主机时，要保证环境变量设置只对你的虚拟主机才有效，而不是对所有用户都有效。在使用Apache服务器时，可以通过在与主配置文件分离的文件中设置相应的变量来做到这一点：

```
SetEnv  DB_USER     "susannah"
SetEnv  DB_PASSWORD "y23a!t@ce8"
```

在主配置文件（httpd.conf）中针对你站点的<VirtualHost>指令，像下面这样包含上面那个单独的文件：

```
Include "/usr/local/apache/database-passwords"
```

要保证这个包含密码的独立文件（例如*/usr/local/apache/database-passwords*）不会被除了管理虚拟主机的人之外的任何用户读到。当Apache启用并读取配置文件时，一般都会从根目录开始，因此就能够读取包含的文件。而处理请求的子进程通常都是作为未被授予特殊权限的用户运行的，所以恶意的脚本也无法读到这些受保护的文件。

参见

Apache包含指令的文档（*http://httpd.apache.org/docs/mod/core.html#include*）。

18.7 存储密码

问题

你需要对用户的密码进行跟踪，以便他们能够登录到你的网站。

方案

当用户登记或者注册时，用md5()对所选择的密码进行加密处理，并把加密后的密码保存到用户数据库中。为了取得最佳效果，使用一个salt：

```
<?php

/* 初始化一个用于保存过滤后数据的数组 */
$clean = array();

/* 定义一个 salt */
define('SALT', 'flyingturtle');

/* 对密码进行加密 */
$encrypted_password = md5(SALT . $_POST['password']);

/* 允许字母和数字用户名 */
if (ctype_alnum($_POST['username'])) {
    $clean['username'] = $_POST['username'];
} else {
    /* 错误 */
}

/* 把用户保存到数据库中 */
$st = $db->prepare('INSERT
            INTO   users (username, password)
            VALUES (?, ?)');
$st->execute(array($clean['username'], $encrypted_password));

?>
```

然后，在用户登录你的网站时，再通过md5()对所提供的密码进行加密，并与保存的加密密码进行比较。如果两个加密值相同，说明用户提供的密码正确：

```
<?php

/* 初始化一个用于保存过滤后数据的数组 */
$clean = array();

/* 定义一个 salt */
define('SALT', 'flyingturtle');

/* 允许字母和数字用户名 */
if (ctype_alnum($_POST['username'])) {
    $clean['username'] = $_POST['username'];
} else {
    /* 错误 */
}

$encrypted_password = $db->getOne('SELECT password
                                   FROM   users
                                   WHERE  username = ?',
                                   array($clean['username']));

if (md5(SALT . $_POST['password']) == $encrypted_password) {
    /* 登录成功 */
} else {
```

```
    /* 登录失败 */
}

?>
```

讨论

存储加密后的密码可以防止未经许可的人偷看数据库中的用户名和密码而泄密（尽管这种未经许可的人能够偷看数据库可能也表示存在其他安全问题）。

使用前面示范的salt可以保护rainbow表的有效性。Rainbow表是与字符串的加密码版伴生的字符串集合。例如，*http://md5.rednoize.com/*就是一个针对MD5的rainbow表查询工具。如果你键入像`6b34fe24ac2ff8103f6fce1f0da2ef57`（`chris`的MD5）这样的查询，就能看到纯粹的MD5是多么容易被破坏。但是通过使用salt，这种工具的有效性会显著降低。

因为MD5是单向算法，所以用它来保存密码会更加安全一些。这也同样意味着你将无从知晓用户密码的原始文本是什么，即使你想知道也无能为力。例如，如果某个用户忘记了自己的密码，你就没办法告诉他密码是什么。而此时最好的方式就是重置这个用户的密码，并告诉用户使用新密码。有关处理丢失密码的问题在技巧18.8中讨论。

参见

技巧18.11中有关存储加密数据的介绍。`md5()`函数的文档（*http://php.net/md5*）。

18.8 处理遗忘的密码

问题

你想为忘记密码的用户提供一个密码。

方案

生成一个新密码并将其发送到用户的电子邮件地址（你的存档文件中应该有）：

```
<?php

/* 生成新密码 */
$new_password = '';
```

```
for ($i = 0; $i < 8; $i++) {
    $new_password .= chr(mt_rand(33, 126));
}

/* 定义 salt. */
define('SALT', 'flyingturtle');

/* 对新密码加密 */
$encrypted_password = md5(SALT . $new_password);

/* 把新密码保存到数据库中 */
$st = $db->prepare('UPDATE users
            SET    password = ?
            WHERE  username = ?');

$st->execute(array($encrypted_password, $clean['username']));

/* 将新密码明文通过电子邮件发送给用户 */
mail($clean['email'], 'New Password', "Your new password is: $new_password");

?>
```

讨论

如果用户忘记了密码，而你使用了技巧18.7中推荐的方式保存了加密后的密码，那么将无法为用户提供遗忘的密码明文。这是由于md5()函数的单向加密特性所决定的。

对于这种情况，必须生成一个新密码并发送到该用户的电子邮箱中。如果接收新密码的电子邮件地址并不是你记录在案的该用户的地址，那么你就无法验证这个地址是不是那个用户本人的。而很可能是一个攻击者在冒充某个用户。

由于包含新密码的邮件没有加密，所以本方案中的代码没有同时发送用户名信息，以降低攻击者通过窃听手段盗取完整账户信息的可能性。如果想从根本上杜绝密码泄露，可以让用户通过回答几个问题（相应的答案也是预先保存的）的方式——而不是直接提供密码，来自我验证。这些问题可以是“你第一个宠物的名字叫什么？”或者“你妈妈娘家姓什么？”等等恶意攻击者不可能知道的问题。在用户正确回答了问题后，那么就可以让他自己设定新密码。

还有一种兼顾安全性和可读性的方式，是以在单词之间插入某些数字的方式生成新密码：

```
<?php

$words = array('mother', 'basset', 'detain', 'sudden', 'fellow', 'logged',
               'remove', 'snails', 'direct', 'serves', 'daring', 'chirps',
               'reward', 'snakes', 'uphold', 'wiring', 'nurses', 'regent',
```

```
                'ornate', 'dogmas', 'mended', 'hinges', 'verbal', 'grimes',
                'ritual', 'drying', 'chests', 'newark', 'winged', 'hobbit');

$word_count = count($words);

$password = sprintf('%s%02d%s',
                    $words[mt_rand(0,$word_count - 1)],
                    mt_rand(0,99),
                    $words[mt_rand(0,$word_count - 1)]);

echo $password;

?>
```

这些代码会生成由两个六位字母的单词和位于这两个单词之间的两个数字组成的密码，像mother43hobbit或verbal68nurses。这样生成的密码比较长，但由于中间用数字隔开也比较容易记忆。

参见

技巧18.7中有关存储加密密码的内容。

18.9 使用散列码验证数据

问题

你想要保证用户不会修改你通过cookie发给他们或者放在表单元素中的数据。

方案

在发送或设置这些数据的同时，也发送并设置使用salt对这些数据进行MD5处理后的散列码。当你接收到返回的数据时，再以相同的salt计算收到数据的MD5散列码。如果两者不匹配，则说明用户修改了数据。

下面示范了如何在隐藏表单字段中生成MD5散列码：

```
<?php

/* 定义salt */
define('SALT', 'flyingturtle');

$id = 1337;
$idcheck = md5(SALT . $id);

?>
```

```
<input type="hidden" name="id" value="<?php echo $id; ?>" />
<input type="hidden" name="idcheck" value="<?php echo $idcheck; ?>" />
```

下面示范了当用户提交表单后如何验证隐藏字段中的数据：

```
<?php

/* 初始化一个保存过滤后数据的数组 */
$clean = array();

/* 定义salt */
define('SALT', 'flyingturtle');

if (md5(SALT . $_POST['id']) == $_POST['idcheck']) {
    $clean['id'] = $_POST['id'];
} else {
    /* 错误 */
}

?>
```

讨论

在处理提交的表单数据时，以相同的salt来计算$_POST['id']值的MD5散列码。如果计算结果与$_POST['idcheck']匹配，则说明$_POST['id']没有被用户修改。如果不匹配，则说明$_POST['id']的值并不是你最初设置的值。

在为cookie添加MD5散列码时，可以使用implode()函数：

```
<?php

/* 定义salt */
define('SALT', 'flyingturtle');

$name = 'Ellen';

$namecheck = md5(SALT . $name);

setcookie('name', implode('|',array($name, $namecheck)));

?>
```

在解析cookie值中的散列码时使用explode()：

```
<?php

/* 定义salt */
define('SALT', 'flyingturtle');

list($cookie_value, $cookie_check) = explode('|', $_COOKIE['name'], 2);
```

```
if (md5(SALT . $cookie_value) == $cookie_hash) {
    $clean['name'] = $cookie_value;
} else {
    /* 错误 */
}

?>
```

在表单或cookie中使用散列码对数据进行验证，明显依赖于在计算散列码时的salt。如果有恶意用户发现了你的salt，那么散列码也就失去了保护作用。而除了注意保护salt外，频繁地更换salt也是一种不错的方案。为了额外增加一层保护，可以使用不同的salt，即基于某些`$id`属性值(例如，通过`$id%10`选择十个不同单词)选择一个特殊的salt用于计算散列码。多了一层保护以后，在某个单词泄露的情况下能够在一定程度上降低危害。

如果你安装了*mhash*模式，也可以不局限于使用MD5散列码。*mhash*支持许多不同的hash算法。要了解有关*mhash*的更多信息，可以参考PHP在线手册中的*mhash*资料或者*mhash*的主页*http://mhash.sourceforge.net/*。

参见

技巧20.9中通过隐藏的表单变量使用hash的例子；`md5()`函数的文档（*http://www.php.net/md5*）和mhash扩展的文档（*http://www.php.net/mhash*）。

18.10 加密和解密数据

问题

你想使用多种流行算法中的一种算法对数据进行加密和解密。

方案

使用PHP的*mcrypt*扩展：

```
<?php

$algorithm = MCRYPT_BLOWFiSH;
$key = 'That golden key that opens the palace of eternity.';
$data = 'The chicken escapes at dawn. Send help with Mr. Blue.';
$mode = MCRYPT_MODE_CBC;

$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode),
                       MCRYPT_DEV_URANDOM);
```

```
$encrypted_data = mcrypt_encrypt($algorithm, $key, $data, $mode, $iv);
$plain_text = base64_encode($encrypted_data);
echo $plain_text . "\n";

$encrypted_data = base64_decode($plain_text);
$decoded = mcrypt_decrypt($algorithm, $key, $encrypted_data, $mode, $iv);
echo $decoded . "\n";

?>
NNB9WnuCYjyd3Y7vUh7XDfWFCWnQYOBsMehHNmBHbGOdJ3cM+yghABb/XyrJ+w3xz9tms74/a70=
The chicken escapes at dawn. Send help with Mr. Blue.
```

讨论

*mcrypt*扩展是*mcrypt*库（一种实现了多种不同加密算法的库）的一个接口。数据的加密和解密是通过mcrypt_encrypt()和mcrypt_decrypt()完成的。这两个函数都接受五个参数。其中第一个参数是使用的算法。调用mcrypt_list_algorithms()会返回你系统中安装的*mcrypt*都支持哪些算法。此外，表18-1中还列出了*mcrypt*支持的所有算法。第二个参数是加密使用的密钥。第三个参数是要加密或解密的数据。第四个参数是加密或解密的模式。（调用mcrypt_list_modes()会返回被支持的模式）。第五个参数是初始化微量（IV,initialization vector），这个参数在某些模式下由加密或解密过程使用。

表18-1中列出了*mcrypt*库支持的全部算法，包括用于指明算法的常量值、密钥及加密块的字节大小和算法以及是否被*libmcrypt* 2.2.x和2.4.x支持。

表18-1：mcrypt算法常量

算法常量	描述	键大小	块大小	2.2.x	2.4.x
MCRYPT_3DES	Triple DES	168 (112 effective)	64	Yes	Yes
MCRYPT_TRIPLEDES	Triple DES	168 (112 effective)	64	No	Yes
MCRYPT_3WAY	3way (Joan Daemen)	96	96	Yes	No
MCRYPT_THREEWAY	3way	96	96	Yes	Yes
MCRYPT_BLOWFiSH	Blowfish (Bruce Schneier)	上至 448	64	No	Yes
MCRYPT_BLOWFiSH_COMPAT	与其他实现兼容的Blowfish	上至 448	64	No	Yes
MCRYPT_BLOWFiSH_128	Blowfish	128	64	Yes	No
MCRYPT_BLOWFiSH_192	Blowfish	192	64	Yes	—
MCRYPT_BLOWFiSH_256	Blowfish	256	64	Yes	No
MCRYPT_BLOWFiSH_448	Blowfish	448	64	Yes	No
MCRYPT_CAST_128	CAST（Carlisle Adams和Stafford Tavares）	128	64	Yes	Yes

表18-1：mcrypt算法常量（续）

算法常量	描述	键大小	块大小	2.2.x	2.4.x
MCRYPT_CAST_256	CAST	256	128	Yes	Yes
MCRYPT_CRYPT	One-rotor Unix crypt	104	8	—	Yes
MCRYPT_ENIGNA	One-rotor Unix crypt	104	8	No	Yes
MCRYPT_DES	美图数据加密标准	56	64	Yes	Yes
MCRYPT_GOST	苏联 Gosudarstvennyi 标准（“国定标准”）	256	64	Yes	Yes
MCRYPT_IDEA	国际加密算法	128	64	Yes	Yes
MCRYPT_LOKI97	LOKI97（Lawrie Brown, Josef Pieprzyk）	128, 192, or 256	64	Yes	Yes
MCRYPT_MARS	MARS (IBM)	128-448	128	No	Yes
MCRYPT_PANAMA	PANAMA（Joan Daemen, Craig Clapp）	—	Stream	No	Yes
MCRYPT_RC2	Rivest Cipher 2	8-1024	64	No	Yes
MCRYPT_RC2_1024	Rivest Cipher 2	1024	64	Yes	No
MCRYPT_RC2_128	Rivest Cipher 2	128	64	Yes	No
MCRYPT_RC2_256	Rivest Cipher 2	256	64	Yes	No
MCRYPT_RC4	Rivest Cipher 4	Up to 2048	Stream	Yes	No
MCRYPT_ARCFOUR	Non-trademarked RC4 compatible	Up to 2048	Stream	No	Yes
MCRYPT_ARCFOUR_IV	Arcfour with Initialization Vector	Up to 2048	Stream	No	Yes
MCRYPT_RC6	Rivest Cipher 6	128, 192, or 256	128	No	Yes
MCRYPT_RC6_128	Rivest Cipher 6	128	128	Yes	No
MCRYPT_RC6_192	Rivest Cipher 6	192	128	Yes	No
MCRYPT_RC6_256	Rivest Cipher 6	256	128	Yes	No
MCRYPT_RIJNDAEL_128	Rijndael（Joan Daemen, Vincent Rijmen）	128	128	Yes	Yes
MCRYPT_RIJNDAEL_192	Rijndael	192	192	Yes	Yes
MCRYPT_RIJNDAEL_256	Rijndael	256	256	Yes	Yes
MCRYPT_SAFERPLUS	SAFER+ (based on SAFER)	128, 192, or 256	128	Yes	Yes
MCRYPT_SAFER_128	Secure And Fast Encryption Routine with strengthened key schedule	128	64	Yes	Yes

表18-1：mcrypt算法常量（续）

算法常量	描述	键大小	块大小	2.2.x	2.4.x
MCRYPT_SAFER_64	Secure And Fast Encryption Routine with strengthened key	64	64	Yes	Yes
MCRYPT_SERPENT	Serpent（Ross Anderson, Eli Biham, Lars Knudsen）	128, 192, or 256	128	No	Yes
MCRYPT_SERPENT_128	Serpent	128	128	Yes	No
MCRYPT_SERPENT_192	Serpent	192	128	Yes	No
MCRYPT_SERPENT_256	Serpent	256	128	Yes	No
MCRYPT_SKIPJACK	U.S. NSA Clipper Escrowed Encryption Standard	80	64	No	Yes
MCRYPT_TWOFiSH	Twofish (Counterpane Systems)	128, 192, or 256	128	No	Yes
MCRYPT_TWOFiSH_128	Twofish	128	128	Yes	No
MCRYPT_TWOFiSH_192	Twofish	192	128	Yes	No
MCRYPT_TWOFiSH_256	Twofish	256	128	Yes	No
MCRYPT_WAKE	Word Auto Key加密（David Wheeler）	256	32	No	Yes
MCRYPT_XTEA	扩展的Tiny加密算法（David Wheeler, Roger Needham）	128	64	Yes	Yes

除了加密和解密的数据之外，所有其他参数在加密和解密时都必须一致。如果使用了一种需要初始化向量的模式，只需明确地将其连同加密文本一起传递即可。

不同的加密模式适用于不同的环境。Cipher Block Chaining（CBC）模式将数据分块加密，并使用前一块中的加密数据（也就是密钥）计算下一块数据的加密值。初始化向量影响的是第一块数据的加密值。 Cipher Feedback（CFB）和Output Feedback（OFB）也会使用初始化向量，但是这两种模式的加密单元要比分块的还小。要注意的是，如果使用OFB模式时加密单元比其分块尺寸小，可能会导致某些安全问题。Electronic Code Book（ECB）模式则是通过不连续的分块方式对数据进行加密，而数据块之间不存在相互依赖关系。 ECB模式不使用初始化向量。但由于在使用同一密钥的情况下相同的明文总会生成相同的密文，因此重复使用这种模式时的安全性较之其他模式要低。表18-2中列出了设置各种模式的常量。

表18-2：mcrypt中的模式常量

模式常量	说明
MCRYPT_MODE_ECB	电子代码图书（Electronic Code Book）模式。
MCRYPT_MODE_CBC	密码块链接（Cipher Block Chaining）模式。
MCRYPT_MODE_CFB	密码反馈（Cipher Feedback）模式。
MCRYPT_MODE_OFB	带8位反馈的输出反馈（Output Feedback）模式。
MCRYPT_MODE_NOFB	带*n*位反馈的输出反馈模式，其中*n*是所用算法的块大小（仅限*libmcrypt* 2.4或更高版本）。
MCRYPT_MODE_STREAM	流式密码（Stream Cipher）模式，针对RC4 和 WAKE 算法（仅限*libmcrypt* 2.4或更高版本）。

不同的算法使用的块大小也不同。可以通过调用mcrypt_get_block_size()得知某种特定算法的块大小。类似地，初始化向量的大小也是由算法和模式决定的。而使用mcrypt_create_iv()和mcrypt_get_iv_size()会很容易生成适当的随机初始化向量：

```
<?php

$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode), MCRYPT_DEV_URANDOM);

?>
```

mcrypt_create_iv()的第一个参数是向量的大小，第二个参数是随机数。随机数包括三种选择：MCRYPT_DEV_RANDOM读取自伪设备（pseudodevice）*/dev/random*，MCRYPT_DEV_URANDOM读取自伪设备*/dev/urandom*，而MCRYPT_RAND则由内部随机数生成器生成。并非所有操作系统都支持随机生成的伪设备。因此，为了取得非重复的随机数流，要保证在使用MCRYPT_RAND之前先调用srand()。

本技巧中的代码和例子与*mcrypt* 2.4兼容。PHP的*mcrypt*接口能够同时支持*mcrypt* 2.2和*mcrypt* 2.4，但对这两个版本的支持程度是不一样的。对于*mcrypt* 2.4，PHP只支持下列函数：mcrypt_ecb()，mcrypt_cbc()，mcrypt_cfb()，mcrypt_ofb()，mcrypt_get_key_size(),mcrypt_get_block_size()，mcrypt_get_cipher_name()和mcrypt_create_iv()。而在使用*mcrypt* 2.2加密或者解密数据时，需要根据你想使用的模式调用相应的mcrypt_*MODE*()函数，并给其传递一个说明如何加密和解密的参数。下面的代码是本方案中代码的*mcrypt* 2.2版本：

```
<?php

$algorithm = MCRYPT_BLOWFiSH;
$key  = 'That golden key that opens the palace of eternity.';
$data = 'The chicken escapes at dawn. Send help with Mr. Blue.';
```

```
$iv = mcrypt_create_iv(mcrypt_get_block_size($algorithm),
                       MCRYPT_DEV_URANDOM);

$encrypted_data = mcrypt_cbc($algorithm, $key, $data, MCRYPT_ENCRYPT, $iv);
$plain_text = base64_encode($encrypted_data);
echo $plain_text . "\n";

$encrypted_data = base64_decode($plain_text);
$decoded = mcrypt_cbc($algorithm, $key, base64_decode($plain_text), MCRYPT_DECR
YPT, $iv);
echo $decoded . "\n";

?>
```

参见

*mcrypt*扩展的文档（*http://www.php.net/mcrypt*），*mcrypt*库的参考资料（*http://mcrypt.hellug.gr/*）。选择适当的算法并安全地使用需要精心地规划：要了解有关crypt以及其使用的密码算法，可以查看PHP在线手册的*mcrypt*部分、*mcrypt*的主页和*/dev/random*及*/dev/urandom*指南页面。另外，关于密码术的好书有Bruce Schneier著的《Applied Cryptography》（Wiley）和Douglas R.Stinson著的《Cryptography: Theory and Practice》（Chapman & Hall）。

18.11 把加密的数据保存到文件或数据库

问题

你想将加密的数据保存起来，以备将来由web服务器取得并进行解密。

方案

将解密数据时所需的信息（例如算法、加密模式和初始化向量）连同加密数据一起保存，但不包括密钥：

```
<?php

/* 加密数据 */
$algorithm  = MCRYPT_BLOWFiSH;
$mode = MCRYPT_MODE_CBC;
$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode), MCRYPT_DEV_URANDO
M);
$ciphertext = mcrypt_encrypt($algorithm, $_POST['key'], $_POST['data'], $mode,
$iv);
```

```
/* 保存加密的数据 */
$st = $db->prepare('INSERT
            INTO   noc_list (algorithm, mode, iv, data)
            VALUES (?, ?, ?, ?)');
$st->execute(array($algorithm, $mode, $iv, $ciphertext));

?>
```

在对数据解密时，使用保存的数据和用户提供的密钥：

```
<?php

$row = $db->query('SELECT *
                   FROM   noc_list
                   WHERE  id = 27')->fetch();
$plaintext = mcrypt_decrypt($row->algorithm,
                            $_POST['key'],
                            $row['data'],
                            $row['mode'],
                            $row['iv']);

?>
```

讨论

例18-1中的*save-crypt.php*脚本把加密后的数据保存到了文件中。

例18-1：save-crypt.php

```
<?php

function show_form() {
    $html = array();
    $html['action'] = htmlentities($_SERVER['PHP_SELF'], ENT_QUOTES, 'UTF-8');

    print<<<FORM
<form method="POST" action="{$html['action']}">
<textarea name="data" rows="10" cols="40">Enter data to be encrypted here.</textarea>
<br />
Encryption Key: <input type="text" name="key" />
<br />
<input name="submit" type="submit" value="Save" />
</form>
FORM;
}

function save_form() {
    $algorithm  = MCRYPT_BLOWFiSH;
    $mode = MCRYPT_MODE_CBC;

    /* 加密数据 */
    $iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode), MCRYPT_DEV_URANDOM);
    $ciphertext = mcrypt_encrypt($algorithm,
```

```
                                    $_POST['key'],
                                    $_POST['data'],
                                    $mode,
                                    $iv);

    /* 保存加密的数据 */
    $filename = tempnam('/tmp','enc') or exit($php_errormsg);
    $file = fopen($filename, 'w') or exit($php_errormsg);
    if (FALSE === fwrite($file, $iv.$ciphertext)) {
        fclose($file);
        exit($php_errormsg);
    }

    fclose($file) or exit($php_errormsg);

    return $filename;
}

if (isset($_POST['submit'])) {
    $file = save_form();
    echo "Encrypted data saved to file: $file";
} else {
    show_form();
}

?>
```

例18-2中是对应的程序 —— *get-crypt.php*，该程序接受一个文件名和一个密钥并生成解密数据。

例18-2：get-crypt.php

```
<?php

function show_form() {
    $html = array();
    $html['action'] = htmlentities($_SERVER['PHP_SELF'], ENT_QUOTES, 'UTF-8');

    print<<<FORM
<form method="POST" action="{$html['action']}">
Encrypted File: <input type="text" name="file" />
<br />
Encryption Key: <input type="text" name="key" />
<br />
<input name="submit" type="submit" value="Display" />
</form>
FORM;

function display() {
    $algorithm  = MCRYPT_BLOWFiSH;
    $mode = MCRYPT_MODE_CBC;

    $file = fopen($_POST['file'], 'r') or exit($php_errormsg);
    $iv = fread($file, mcrypt_get_iv_size($algorithm, $mode));
```

```
    $ciphertext = fread($file, filesize($_POST['file']));
    fclose($fh);

    $plaintext = mcrypt_decrypt($algorithm, $_POST['key'], $ciphertext, $mode,$iv);
    echo "<pre>$plaintext</pre>";
}

if (isset($_POST['submit'])) {
    display();
} else {
    show_form();
}

?>
```

在这两个程序当中，加密算法和模式都采取了硬编码的方式写在了代码中，所以这些信息就用不着保存到文件中了。而文件中包含的是初始化向量及紧随其后的加密数据。因为mcrypt_get_iv_size()实际上会返回解密程序需要在完整的IV后面读取多少个字节数，所以不需要在初始化向量（IV）后面添加分隔符。

通过使用本技巧中介绍的方式对文件进行加密，可以在攻击者取得访问保存该文件服务器访问权限的情况下对文件进行保护。如果没有正确的密钥或者强大的计算能力，攻击者将无法读取文件。然而，如果加密的数据和加密密钥在你的服务器和用户的web浏览器之间毫无戒备地传输，那么这些加密文件所提供的安全性也会大打折扣。对于能够窃听或监控网络通信的人而言，他们都可以在数据被加密之前将其截获。为了防止数据被窃听，可以使用SSL。

当你使用本技巧中的方式在web服务器端加密数据的时候，另外一种风险来自于在对数据进行加密并写入到文件之前其内容的可见性。对于具有服务器root或administrator权限的人，他实际上可以查看web服务器进程所使用的内存，并有可能窥视到尚未加密的数据以及密钥。而如果操作系统将web服务器进程的内存映像交换到磁盘上，那就有可能在磁盘交换文件中看到未加密的数据。这种攻击既不容易实现，也很难破除。只要加密数据保存到文件中，即使是拥有web服务器root权限的人也会很难分辨，但如果攻击者能够在写入文件之前就偷看到未加密的数据，那么这种情况下加密所提供的保护也将是有限的。

参见

技巧18.13中有关SSL和保护在网络上传输数据的内容；mcrypt_encrypt()函数的文档（*http://www.php.net/mcrypt-encrypt*），mcrypt_decrypt()函数的文档（*http://www.php.net/mcrypt-decrypt*），mcrypt_create_iv()函数的文档（*http://www.php.net/mcrypt-*

create-iv）和mcrypt_get_iv_size()函数的文档（*http://www.php.net/mcrypt-get-iv-size*）。

18.12 在网站之间共享加密数据

问题

你想与另外一个网站安全地交换加密数据。

方案

如果其他网站需要从你的站点中取得数据，那么把这些数据放在一个设置了密码保护的页面中，也可以将相应的数据放到加密的表单中，带或不带密码保护取决于你。如果你想把数据放到另外一个网站上，可以通过post方法将加密的数据提交到一个受密码保护的URL中。

讨论

下面的页面需要一个用户名和密码，然后加密并显示包含昨天账户活跃度的文件内容：

```
<?php

$user = 'bank';
$password = 'fas8uj3';

if ($_SERVER['PHP_AUTH_USER'] != $user ||
    $_SERVER['PHP_AUTH_PW'] != $password) {
    header('WWW-Authenticate: Basic realm="Secure Transfer"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You must supply a valid username and password for access.";
    exit;
}

header('Content-type: text/plain; charset=UTF-8');
$filename = strftime('/usr/local/account-activity.%Y-%m-%d', time() - 86400);
$data = implode('', file($filename));

$algorithm  = MCRYPT_BLOWFiSH;
$mode = MCRYPT_MODE_CBC;
$key  = "There are many ways to butter your toast.";

/* 加密数据 */
$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode), MCRYPT_DEV_URANDOM);
$ciphertext = mcrypt_encrypt($algorithm, $key, $data, $mode, $iv);
```

```
echo base64_encode($iv.$ciphertext);

?>
```

以下是获取加密数据并进行解密的相应代码：

```
<?php

$user = 'bank';
$password = 'fas8uj3';
$algorithm  = MCRYPT_BLOWFiSH;
$mode = MCRYPT_MODE_CBC;
$key  = "There are many ways to butter your toast.";

$file = fopen("http://$user:$password@bank.example.com/accounts.php", 'r')
        or exit($php_errormsg);
$data = '';

while (!feof($file)) {
    $data .= fgets($file, 1048576);
}

fclose($file) or exit($php_errormsg);
$binary_data = base64_decode($data);
$iv_size = mcrypt_get_iv_size($algorithm, $mode);
$iv = substr($binary_data, 0, $iv_size);
$ciphertext = substr($binary_data, $iv_size, strlen($binary_data));

echo mcrypt_decrypt($algorithm, $key, $ciphertext, $mode, $iv);

?>
```

这里的恢复程序与加密程序所做的完全相同，只是方向相反。它通过用户名和密码取得Base64编码的加密数据，并进行Base64解码，然后分离出初始化向量。最后，对数据进行解密并输出。

在前面的例子当中，用户名和密码仍然是以明文的形式在网络中传输的，除非连接始终使用SSL。不过，如果使用了SSL，可能也就不需要对文件的内容进行加密了。我们在例子中同时介绍了密码提示和文件加密，只是为了说明其实现过程。

不管怎样，同时使用密码保护和文件加密在有一种情况下还是有用的，就是当收到的文件不自动解密时。自动化的程序会在取得加密文件后保持其加密状态，并将其放在某个地方以备后面使用。而解密所需的密钥也就不需要保存到恢复程序中了。

参见

技巧18.13中有关SSL和保护在网络上传输数据的内容。`mcrypt_encrypt()`函数的文档

（*http://php.net/mcrypt-encrypt*）和mcrypt_decrypt()函数的文档（*http://php.net/mcrypt-decrypt*）。

18.13 检测SSL

问题

你想知道收到的请求是否基于SSL。

方案

测试$_SERVER['HTTPS']的值：

```
<?php

if ('on' == $_SERVER['HTTPS']) {
  echo 'The secret ingredient in Coca-Cola is Soylent Green.';
} else {
  echo 'Coca-Cola contains many delicious natural and artificial flavors.';
}

?>
```

讨论

SSL是在比HTTP更低的层次上发挥作用的。Web服务器和浏览器会根据各自的能力和能够通过安全连接传递的HTTP信息，协商决定建立适当的安全连接。对于截取数据的攻击者来说，所截取的数据只是一些无法看懂且没有意义的字节码。

不同的web服务器对使用SSL有不同的要求，所以在使用SSL之前一定要仔细阅读服务器的文档。而让PHP基于SSL来运行，则不需要改变什么。

除了基于$_SERVER['HTTPS']修改代码之外，还可以通过SSL连接来设置需要交换的cookie。如果setcookie()的最后一个参数是TRUE，那么浏览器只会通过安全连接将cookie发送回服务器：

```
<?php

/* Set an SSL-only cookie named "sslonly" with value "yes" that expires at the
   end of the current browser session. */

setcookie('sslonly', 'yes', '', '/', 'example.org', true);

?>
```

尽管浏览器只通过SSL连接把cookie发送回服务器，但服务器在将cookie发送到浏览器（当在页面中调用setcookie()的时候）时，则不管对设置cookie页面的请求是否是基于SSL的。如果你在cookie中设置了敏感的数据，就必须保证同样只在SSL请求中设置这个cookie。但是，也不要忘记cookie中的数据在用户计算机中是未加密的。

参见

setcookie()函数的文档（*http://php.net/setcookie*）。

18.14 通过GPG加密电子邮件

问题

你想发送加密的电子邮件。例如，你通过自己的网站接受了一个订单，然后需要给供应商发送一封包含要处理的订单详细资料的电子邮件。通过对这封电子邮件进行加密，就可以避免在网络中以明文方式传递诸如信用卡号码之类的敏感信息。

方案

在发送电子邮件之前，对邮件信息的主体内容使用GNU Privacy Guard（GPG）进行加密：

```
<?php

$message_body = escapeshellarg($message_body);
$gpg_path     = '/usr/local/bin/gpg';
$sender       = 'web@example.com';
$recipient    = 'ordertaker@example.com';
$home_dir     = '/home/web';
$user_env     = 'web';

$cmd = "echo $message_body | HOME=$home_dir USER=$user_env $gpg_path " .
       '--quiet --no-secmem-warning --encrypt --sign --armor ' .
       "--recipient $recipient --local-user $sender";

$message_body = `$cmd`;

mail($recipient, 'Web Site Order', $message_body);

?>
```

而这封电子邮件可以通过GPG、Pretty Good Privacy（PGP）或支持上述任何一个程序的电子邮件客户插件进行解密。

讨论

PGP是一种流行的公共密钥加密程序，而GPG是基于PGP的开源程序。因为使用PGP经常会受到各种专利权及控制问题的困扰，所以一般来说使用GPG更随意一些。

本方案中的代码调用了*/usr/local/bin/gpg*来加密保存在`$message_body`中的信息，它使用了属于`$sender`的私人密钥和属于`$recipient`的公共密钥。这意味着，只有`$recipient`能够解密这封电子邮件，并且当完成解密时，也会知道该邮件来自`$sender`。

设置环境变量`HOME`和`USER`是为了告诉GPG到哪里去找它们的密钥：*$HOME/.gnupg/secring.gpg*。而`--quiet`和`--no-secmem-warning`选项则是为了消除GPG可能会生成的警告。`--encrypt`和`--sign`选项是告诉 GPG 同时加密并标记信息。通过加密使得信息对除了收件人之外的任何人都难辨其意义，而通过标记则会使收件人知道是谁生成的这些信息以及生成的时间。`--armor`选项会保证生成纯文本而不是二进制的输出，以便密文适合电子寄送。

一般来说，私人密钥是通过口令（passphrase）来保护的。如果受口令保护的私人密钥被攻击者得到，除非攻击知道口令，否则就不能通过私人密钥来加密信息。GPG会在对信息加密时提示输入口令。不过在本技巧中，我们没有给私人密钥`$sender`设置口令。如果我们设置了口令，那么如果没有人每次输入口令，网站就不会发送新的订单邮件。而把口令保存到文件中，在每次需要加密时提供给GPG并不会比前面不设置口令具有更高的安全性。

在加密时使用没有口令保护的密钥有一个缺点，就是得到这个私密密钥的攻击者有可能会向你的订单处理程序发送一些虚假的订单邮件。当然，这种风险是容易控制的。由于订单首先能够通过网站提交，所以之前就有了向订单程序中插入不良信息的可乘之机。捕捉有害订单的任何处理程序同样也能够被这些虚假的订单所触发。同样地，一旦发觉密钥失窃，就会对失窃的漏洞进行修复，而此时更换一个新的密钥就能轻易地将攻击者拒之门外。

参见

GNU Privacy Guard的主页*http://gnupg.org/*和MIT PGP分发网站*http://web.mit.edu/network/pgp.html*。

第19章

国际化和本地化

19.0 概述

虽然用PHP编程的人们都要学习一些英语，并且最终要掌握函数名和语法结构，但事实上，通过PHP能够创建几乎适用于任何语言的应用程序。的确有一些应用程序需要供讲多种不同语言的人使用。而通过PHP提供的对国际化和本地化的支持，让一个讲法语的人编写的程序能够被讲德语的人使用并非难事。

所谓国际化［通常简写为I18N（注1）］，是指对一个只为某个地区设计的程序进行重构，使其能够在更多地区使用的过程。而所谓本地化［通常简写为L10N（注2）］，则是指在一个面向国际化的程序中增加对一个新地区应用支持的过程。

地区是一组描述世界上某一特定区域文本格式和语言习惯的设置集合，这些设置可以分成六个类别：

LC_COLLATE
: 此设置控制文本的顺序——在字母顺序中，哪个字母在先，哪个字母在后。

LC_CTYPE
: 此设置控制大小写字母之间的转换，也包括哪个字符属于哪个字符类，如字母数字式字符等。

LC_MONETARY
: 此设置描述货币信息的首选格式，诸如用哪个字符表示小数点和如何标识负数。

注1：单词“internationalization”在首字母“i”和末字母“n”之间有18个字母。

注2：单词“localization”在首字母“l”和末字母“n”之间有10个字母。

LC_NUMERIC

此设置描述数字信息的首选格式，诸如如何对数字分组和用哪个字符作为千分位分隔符。

LC_TIME

此设置描述时间和日期信息的首选格式，诸如月和日的名称以及是24小时制还是12小时制。

LC_MESSAGES

在需要以多种语言显示信息时此类别包含程序要使用的文本消息。

还有一个元类别 —— LC_ALL，它包含了所有类别。

地区名通常由三个部分构成：第一部分，是一个强制性的、表示语言的缩写，例如“en”表示英语或“pt”表示葡萄牙语。第二部分，跟在一个下划线之后，是一个可选的国家说明符，用于区分讲同一种语言的不同国家，例如“en_US”表示美国英语，而“en_UK”表示英国英语，或者“pt_BR”表示巴西葡萄牙语，而“pt_PT”表示葡萄牙葡萄牙语。最后一部分，跟在一个句点之后，是一个可选的字符集说明符，例如“zh_TW.Big5”表示我国台湾省使用Big5字符集。虽然大多数地区名都遵循这一约定，但也有个别不遵循这个约定的情况。在使用地区名过程中的一个主要问题就是一些地区命名太过随意。查询和设置地区的主题在技巧19.1 ~ 19.3中讨论。

如果要对文本、日期和时间以及货币等进行正确的本地化，必须使用几种不同的技术。本地化也可以用于你程序中使用的外部实体中，如图像和包含文件等。有关本地化的内容在技巧19.4 ~ 19.8中介绍。

对于处理大量本地化数据的系统，将在19.9和19.10中讨论。19.9中介绍了控制数据的一些简单方式，而技巧19.10则介绍了GNU的*gettext* —— 一个提供本地化支持的、功能完整的工具集。

技巧19.11 ~ 19.13讨论了如何保证你的程序能够正确地与多种字符编码协同工作，例如能够正常地处理诸如à l'Opéra-Théâtre和（日本字符）等字符串。要实现这一点的一种方式就是对程序要处理的所有文本进行UTF-8编码。这种编码方案能够像ISO-8859-1一样处理西方字符，同样也能处理世界上其他各种记录系统中的字符。这几个技巧聚焦于通过UTF-8提供无缝的、语言无关的用户体验。

正在开发中的PHP 6将在对Unicode字符集的支持方面进行极大的改进，包括更加有效地操作多字节字符串以及对地区系统的彻底修补。Andrei Zmievski的“PHP 6 and

Unicode”演讲稿（*http://www.gravitonic.com/talks/*）中，对PHP 6中与Unicode相关的内容进行了介绍。

19.1 列举有效的地区

问题

你想知道自己的系统都支持哪些地区。

方案

使用*locale*程序列举出可用的地区，用*locale-a*可以输出你的系统所支持地区。

讨论

在Linux和Solaris系统中，可以在*/usr/bin/locale*中找到有效的`locale`。在Windows XP系统中，可用的地区位于“控制面板”中“区域和语言选项”的“区域选项”中“标准和格式”下拉菜单中。

对于其他系统，查看有效地区的方式会有所不同。例如，BSD支持地区但并不提供列举地区的*locale*程序。BSD的地区信息通常保存在*/usr/share/locale*中，因此在这个目录中有可能找到可用的地区列表。

虽然地区系统有助于解决许多本地化的问题，但由于标准化程度不够，所以使用不便之处也经常会令人沮丧。不仅不同的系统支持的地区会有所不同，甚至对于相同的地区都会使用不同的名称。

参见

你的系统的*locale(1)*参考页。Windows系统用作地区名的语言字符串列表：（*http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_language_strings.asphttp://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_language_strings.asp.*）。Windows系统用作地区名的国家/地区字符串列表：（*http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_country_strings.asp*）。

19.2 使用特定的地区

问题

你想让PHP使用特定的地区设置。

方案

以相应的类别和地区调用setlocale()。下面示范了如何对所有类别都使用es_MX（Mexican Spanish）地区：

```
setlocale(LC_ALL,'es_MX');
```

下面示范了如何在时间和日期格式中使用de_AT（Austrian German）地区：

```
setlocale(LC_TIME,'de_AT');
```

在Windows系统中，等价的地区名分别是Spanish_Mexico和German_Austria。

讨论

如果想查找当前地区且不改变它，可以如例19-1所示的以0作为地区参数传递给setlocale()。

例19-1：取得当前地区

```
<?php
print setlocale(LC_ALL,0);
?>
```

很多系统也支持使用常见地区的别名，并放在诸如*/usr/share/locale/locale.alias*这样的文件中。该文件中可能会包含像下面这样很多行：

```
russian         ru_RU.ISO-8859-5
slovak          sk_SK.ISO-8859-2
slovene         sl_SI.ISO-8859-2
slovenian       sl_SI.ISO-8859-2
spanish         es_ES.ISO-8859-1
swedish         sv_SE.ISO-8859-1
```

每一行的第一列是别名，第二列是相应别名指向的地区和字符集。可以在调用setlocale()时使用别名而不是与别名对应的字符串。例如，可以使用“setlocale(LC_ALL,'swedish');”来代替“setlocale(LC_ALL,'sv_SE.ISO-8859-1');”。

通过调用setlocale()也可以在Windows系统中修改地区设置。我们在技巧19.1中已经介绍过了，地区名可能会不一样。如果PHP是在多线程的环境中运行，那么修改地区则可能导致意想不到的结果。因为调用setlocale()会修改当前进程中所有线程的地区设置。也就是说，在一个线程的地区设置被修改后，其他线程中正在运行的脚本也会立即应用相同的修改。因此，如果需要使用setlocale()，最好考虑使用单线程的服务器设置。

参见

技巧19.3中有关设置默认地区的介绍；setlocale()函数的文档（*http://www.php.net/setlocale*）。

19.3 设置默认地区

问题

你想要设置一个可供你的所有PHP程序使用的地区。

方案

在通过auto_prepend_file配置指令开始加载文件之前，调用setlocale()来设置你想要的地区，如例19-2所示。

例19-2：设置默认地区

```
<?php
setlocale(LC_ALL,'es_US');
?>
```

如果想使用系统的环境变量中设置的默认地区，可以将null作为地区参数传递给setlocale()，如例19-3所示。

例19-3：基于系统环境设置默认的地区

```
<?php
setlocale(LC_ALL,null);
?>
```

讨论

如果不调用setlocale()，即使在你启动Web服务器或PHP二进制文件（binary）之前设置了相应的环境变量，PHP也不会改变其地区设置。例如，在把环境变量LC_ALL设置为es_US之后，PHP仍然会运行在其默认的C地区。

参见

技巧19.2中有关使用特定地区的介绍；setlocale()函数的文档（*http://www.php.net/setlocale*）和auto_prepend_file函数的文档（*http://www.php.net/manual/en/configuration.directives.php#ini.auto-prepend-file*）。

19.4 本地化文本消息

问题

你想要以适合本地的语言显示文本消息。

方案

维护一个单词和短语的消息目录，在打印消息之前从消息目录中取出相应的字符串。例19-4所示为一个包含美国英语和英国英语中食品名称的简单的消息目录，以及一个从该目录中抽取单词的函数。

例19-4：简单的消息目录

```
<?php
$messages = array ('en_US' =>
             array(
              'My favorite foods are' => 'My favorite foods are',
              'french fries' => 'french fries',
              'candy'        => 'candy',
              'potato chips' => 'potato chips',
              'eggplant'     => 'eggplant'
             ),
            'en_UK' =>
             array(
              'My favorite foods are' => 'My favourite foods are',
              'french fries' => 'chips',
              'candy'        => 'sweets',
              'potato chips' => 'crisps',
              'eggplant'     => 'aubergine'
             )
            );

function msg($s) {
    global $LANG, $messages;
    if (isset($messages[$LANG][$s])) {
        return $messages[$LANG][$s];
    } else {
        error_log("l10n error: LANG: $lang, message: '$s'");
    }
}
?>
```

讨论

例19-5使用这个消息目录打印出了相应的食品列表。

例19-5：使用消息目录

```
<?php
$LANG = 'en_UK';
print msg('My favorite foods are').":\n";
print msg('french fries')."\n";
print msg('potato chips')."\n";
print msg('candy')."\n";
?>
```

例19-5的输出结果：

```
My favourite foods are:
chips
crisps
sweets
```

如果想在例19-5中得到美国英语而非英国英语的叫法，只要把$LANG设置为en_US即可。

可以将msg()信息检索函数与sprintf()组合起来保存需要替换其中值的短语。例如，考虑一下英语中的这句话“I am 12 years old.”，在西班牙语中对应的句子是“Tengo 12 años.”。对于西班牙短语不能通过分别变换“I am”，数字12和“years old”来构建。相反，可以将它们以sprintf()式的字符串保存到信息分类表中，如例19-6所示。

例19-6：sprintf()式的消息目录

```
<?php
$messages = array ('en_US' => array('I am X years old.' => 'I am %d years old.'),
                   'es_US' => array('I am X years old.' => 'Tengo %d a?os.')
            );
?>
```

例19-7将msg()的结果作为一个格式化字符串传递给sprintf()。

例19-7：使用sprintf()式的消息目录

```
<?php
$LANG = 'es_US';
print sprintf(msg('I am X years old.'),12);
?>
```

例19-7的输出结果：

```
Tengo %d años.
```

对于不同语言短语中需要替换值顺序不同的情况，sprintf()也支持更改参数的顺序。具体实现如例19-8所示。

例19-8：改变消息目录参数的顺序

```
<?php
$messages = array ('en_US' =>
                    array('I am X years and Y months old.' =>
                          'I am %d years and %d months old.'),
                   'es_US' =>
                    array('I am X years and Y months old.' =>
                          'Tengo %2$d meses y %1$d a?os.')
            );
?>
```

不管是哪种语言，调用sprintf()时的参数顺序都是一样的（即先是年份，然后是月份），如例19-9所示。

例19-9：在参数顺序可变的情况下使用消息目录

```
<?php
$LANG = 'es_US';
print sprintf(msg('I am X years and Y months old.'),12,7);
$LANG = 'es_US';
print sprintf(msg('I am X years and Y months old.'),12,7);
?>
```

例19-9的输出结果：

```
I am 12 years and 7 months old.
Tengo 7 meses y 12 años.
```

在格式化的字符串中，%2$告知sprintf()使用第二个参数，而%1$则告知它使用第一个参数。

这些短语可以保存为一个函数的返回值，而不是数组中的字符串。用函数来保存短语也就不再需要使用sprintf()了。例19-10显示了一些返回整个句子的函数。

例19-10：消息目录函数

```
<?php
// 英文版
function i_am_X_years_old($age) {
 return "I am $age years old.";
}

// 西班牙语版
function i_am_X_years_old($age) {
 return "Tengo $age a?os.";
}
?>
```

如果一部分消息目录是以数组形式保存的，而另一部分消息目录保存在了函数中，那么对于这种语言的消息目录来说，对象就是一种理想的容器。例19-11包含了一个基本的对象和两个简单的消息目录。

例19-11：消息目录对象

```
class pc_MC_Base {
  public $messages;
  public $lang;

  public function msg($s) {
    if (isset($this->messages[$s])) {
      return $this->messages[$s];
    } else {
      error_log("l10n error: LANG: $this->lang, message: '$s'");
    }
  }

}

class pc_MC_es_US extends pc_MC_Base {
   public $lang = 'es_US';
   public $messages = array ('chicken' => 'pollo',
                             'cow'     => 'vaca',
                             'horse'   => 'caballo'
                            );

  public function i_am_X_years_old($age) {
    return "Tengo $age a?os ";
  }
}

class pc_MC_en_US extends pc_MC_Base {
    public $lang = 'en_US';
    public $messages = array ('chicken' => 'chicken',
                              'cow'     => 'cow',
                              'horse'   => 'horse'
                              );

  public function i_am_X_years_old($age) {
    return "I am $age years old.";
  }
}
?>
```

每个消息目录对象都通过扩展pc_MC_Base类得到msg()方法，然后定义自己的消息（在其构造器中）和自己返回短语的函数。例19-12中使用了这个消息目录对象来以西班牙语输出相应的文本。

例19-12：使用消息目录对象

```
<?php
$MC = new pc_MC_es_US;
```

```
print $MC->msg('cow');
print $MC->i_am_X_years_old(15);
?>
```


如果想用英语来输出相同内容的文本，只需要将$MC指定为pc_MC_en_US对象，而不是pc_MC_es_US对象即可，其他的代码仍然保持不变。

参见

第7章中“概述”部分有关对象技巧化的讨论；sprintf()函数的文档（*http://www.php.net/sprintf*）。

19.5 本地化日期和时间

问题

你想以符合某个地区习惯的方式显示日期和时间。

方案

使用strftime()的%c格式化字符串：print strftime('%c');。这个格式化字符串能够以符合某地区习惯的方式显示完整的日期和时间戳。

也可以将strftime()格式化字符串作为消息保存到消息目录当中，并将结果传递给strftime()，如例19-13所示。

例19-13：通过strftime()使用消息目录

```
<?php
$MC = new pc_MC_es_US;
print strftime($MC->msg('%Y-%m-%d'));
?>
```

讨论

%c格式化字符串告诉strftime()以与当前地区使用习惯相符的首选表示方法返回日期和时间。因而，会因使用的地区不同而产生不同的结果：

```
Sat May 13 13:53:31 2006      // 默认的 C 地区
Sam 13 Mai 2006 13:53:31 EDT  // de_AT 地区
sam 13 mai 2006 13:53:31 EDT  //  fr_FR 地区
```

由%c生成的格式化时间字符串虽然符合地区习惯，但却不够灵活。例如，你要是只想取得时间字符串，则必须给strftime()传递不同的格式化字符串。而这些格式化字符串本身则是因地区而异的。在某些地区，会用带有适当的 A.M./P.M. 的数字1 ~ 12来表示小时，而其他地区则会用数字0 ~ 23来表示小时。要适当地显示针对某个地区的时间字符串，则需要为包含你想要的每个时间格式的$messages数组增加一个元素。对于一个特定的时间格式，如%H:%M，它的键在每个地区都是相同的，而值则有可能不同，如%H:%M用于实行24小时制的地区，%I:%M%P用于实行12小时制的地区。然后，查询出适当的格式化字符串并将其传递给strftime()，如例19-14所示。

例19-14：通过消息目录实现时间格式化

```
<?php
$MC = new pc_MC_es_US;

print strftime($MC->msg('%H:%M'));
?>
```

注意，对地区的修改不会改变时区；也就是说，它只改变显示结果的格式。

参见

技巧3.4中有关strftime()可以接受的格式化字符串的讨论；技巧3.11中涉及通过程序改变时区设置的内容；strftime()函数的文档（*http://www.php.net/strftime*）。

19.6 本地化货币值

问题

你想以符合地区惯例的格式显示货币金额。

方案

使用money_format()函数生成经过适当格式化的字符串。例19-15所示为一些money_format()能够识别的格式化字符。

例19-15：使用money_format()进行格式化

```
<?php
$income = 5549.3;
$debit  = -25.95;

$formats = array('%i', // 国际
                 '%n', // 国家
```

```
                    '%+n', // + 和 -
                    '%(n', // () 负数
                    );

setlocale(LC_ALL, 'en_US');
foreach ($formats as $format ) {
    print "$income @ $format = " .
        money_format($format,$income) .
        "\n";
    print "$debit @ $format = " .
        money_format($format,$debit) .
        "\n";
}
?>
```

例19-15的输出结果为：

```
5549.3 @ %i = USD 5,549.30
-25.95 @ %i = -USD 25.95
5549.3 @ %n = $5,549.30
-25.95 @ %n = -$25.95
5549.3 @ %+n = $5,549.30
-25.95 @ %+n = -$25.95
5549.3 @ %(n = $5,549.30
-25.95 @ %(n = ($25.95)
```

在PHP 4.3.0之前或者在Windows系统中无法使用money_format()函数。如果不能使用money_format()，那么可以使用例19-17中的pc_format_currency()函数来生成一个经过适当格式化字符串。例19-16所示为pc_format_currecy()函数的用法。

例19-16：使用 pc_format_currency()

```
<?php
setlocale(LC_ALL,'turkish');
print pc_format_currency(-12345678.45);
?>
```

例19-16的输出结果为：

```
-12.345.678,45 TL
```

讨论

money_format()与sprintf()或strftime()类似，它们都接受格式化字符串和值作为参数。其中格式化字符串中的特殊顺序用于说明如何对值进行格式化。如同sprintf()要求格式化序列必须遵循特定的次序（百分号、间隔、类型说明符等等）一样，money_format()同样要求每个格式化序列以一定的顺序出现，即百分号、标记、宽

度、左精确度、右精确度、转换字符。只有百分号和转换字符是强制的。表19-1列出了money_format()能够接受的格式化字符。

表19-1：money_format()的格式化字符

类别	格式化字符	说明
标志	*=c*	使用单个字符“*c*”作为填充字符。默认是空格
标志	^	不要使用分组字符
标志	+	使用针对地区的“+”和“-”字符来格式化正数和负数。无论使用“+”还是“(”标记，这都是默认值
标志	(	用“(”和“)”环绕负值
标志	!	不要在输出字符串中包含货币符号
标志	-	所有字段左对齐。如果不存在这个字符，则所有字段右对齐
宽度	*width*	指定最小字段的宽度。默认的最小字段宽度为0
左精度	*#precision*	如果小数点左侧数位的精度不够，则使用填充字符填充宽度
右精度	*.precision*	在格式以前，将小数点右侧数位舍入到*precision*精度。如果*precision*是0，那么不论小数点还是其右侧的数字都不会输出。默认与本地设置一致
转换字符	i	使用国际化的货币格式。这通常意味着是三个字符的代码，如用作货币符号的USD
转换字符	n	使用国家的货币格式。这通常意味着一个适合本地化的值，如用作货币符号的$
转换字符	%	%直接量

由于money_format()有赖于strfmon()系统功能的支持，所以只有该系统功能有效时才能够使用它。而Windows不支持strfmon()系统功能。

例19-17中的pc_format_currency()函数从localeconv()中得到货币格式化信息，然后使用number_format()和一些逻辑运算来构造正确的字符串。

例19-17：pc_format_currency

```
<?php
function pc_format_currency($amt) {
    // 取得符合地区惯例的货币格式信息
    $a = localeconv();

    // 计算$amt的符号并去掉该符号
    if ($amt < 0) { $sign = -1; } else { $sign = 1; }
```

```
    $amt = abs($amt);
    // 通过适当分组，小数点和小数位来格式化$amt
    $amt = number_format($amt,$a['frac_digits'],$a['mon_decimal_point'],
                         $a['mon_thousands_sep']);

    // 求出放置货币符号以及正号或负号的位置
    $currency_symbol = $a['currency_symbol'];
    // $amt >= 0成立吗?
    if (1 == $sign) {
        $sign_symbol  = 'positive_sign';
        $cs_precedes  = 'p_cs_precedes';
        $sign_posn    = 'p_sign_posn';
        $sep_by_space = 'p_sep_by_space';
    } else {
        $sign_symbol  = 'negative_sign';
        $cs_precedes  = 'n_cs_precedes';
        $sign_posn    = 'n_sign_posn';
        $sep_by_space = 'n_sep_by_space';
    }
    if ($a[$cs_precedes]) {
        if (3 == $a[$sign_posn]) {
            $currency_symbol = $a[$sign_symbol].$currency_symbol;
        } elseif (4 == $a[$sign_posn]) {
            $currency_symbol .= $a[$sign_symbol];
        }
        // 货币符号前置
        if ($a[$sep_by_space]) {
            $amt = $currency_symbol.' '.$amt;
        } else {
            $amt = $currency_symbol.$amt;
        }
    } else {
        // 货币符号后置
        if ($a[$sep_by_space]) {
            $amt .= ' '.$currency_symbol;
        } else {
            $amt .= $currency_symbol;
        }
    }
    if (0 == $a[$sign_posn]) {
        $amt = "($amt)";
    } elseif (1 == $a[$sign_posn]) {
        $amt = $a[$sign_symbol].$amt;
    } elseif (2 == $a[$sign_posn]) {
        $amt .= $a[$sign_symbol];
    }
    return $amt;
}
?>
```

对于正、负金额而言，pc_format_currency()中放置货币符号与记录正确位置的代码几乎是相同的 —— 只不过使用的是由localeconv()返回数组中的不同元素而已。该数组返回的相应元素如表19-2所示。

表19-2：由localeconv()返回的与货币相关的信息

数组元素	说明
currency_symbol	本地货币符号
mon_decimal_point	货币中的小数点字符
mon_thousands_sep	货币中的千位分隔符
positive_sign	正号
negative_sign	负号
frac_digits	带小数点的数值
p_cs_precedes	如果currency_symbol位于正值之前是1，之后是0
p_sep_by_space	如果用空格分隔currency_symbol和正值是1，否则是0
n_cs_precedes	如果currency_symbol位于负值之前是1，之后是0
n_sep_by_space	如果用空格分隔currency_symbol和负值是1，否则是0
p_sign_posn	正号位置： • 如果圆括号括住数量和currency_symbol，是0 • 如果正号字符串位于数量和currency_symbol之前，是1 • 如果正号字符串位于数字和currency_symbol之后，是2 • 如果正号字符串直接位于currency_symbol之前，是3 • 如果正号字符串直接位于currency_symbol之后，是4
n_sign_posn	负号位置：可能的值与p_sign_posn相同

参见

技巧2.10中同样也讨论到的money_format()的内容；money_format()函数的文档（*http://www.php.net/money_format*），localeconv()函数的文档（*http://www.php.net/localeconv*）和number_format()函数的文档（*http://www.php.net/number-format*）。

19.7 本地化图像

问题

你想让包含文本的图像中的文本以适合本地的语言显示。

方案

为你想支持的每个地区建立一个目录，并建立一个全局图像目录保存不包含本地化信息

的图像。然后将需要本地化的图像拷贝一份保存到相应地区的目录中。要保证不同目录中图像的文件名相同。最后，不采取直接输出图像URL的方式，而是使用类似技巧19.4中输出符合地区习惯文本的msg()那样的包装函数。

讨论

例19-18中的img()包装函数首先会查找特定地区版本的图像是否存在，然后才会查找全局图像。如果两者都不存在，它会向错误日志中写入一条错误消息。

例19-18：查找本地化图像

```
<?php
$image_base_path = '/usr/local/www/images';
$image_base_url  = '/images';

function img($f) {
    global $LANG;
    global $image_base_path;
    global $image_base_url;

    if (is_readable("$image_base_path/$LANG/$f")) {
        return "$image_base_url/$LANG/$f";
    } elseif (is_readable("$image_base_path/global/$f")) {
        return "$image_base_url/global/$f";
    } else {
        error_log("l10n error: LANG: $lang, image: '$f'");
    }
}
```

img()函数既需要知道文件系统（$image_base_path）中的图像文件路径，也需要知道基于你的站点URL（*/images*）的图像路径。它用前者来测试该文件是否可读，用后者来为图像构造适当的URL。

每个本地化目录中相同图像的本地化副本必须使用相同的名称。例如，包含文本“New!”的黄色脉冲星图片不论是在*images/en_US*目录还是在*images/es_US*目录中都应该命名为*new.gif*，即使*images/es_US/new.gif*图像中位于黄色脉冲星背景上面的文本是“Nuevo!”也必须如此。

不要忘了对图像标签中的alt属性值进行本地化。例19-19输出了一个完整的本地化<img/>元素。

例19-19：本地化的<img>元素

```
<?php
print '<img src="' . img('cancel.png') . '" ' .
      'alt="' . msg('Cancel') . '"/>';
?>
```

如果某幅特定图像的本地化版本大小不同，那么同时也要把图像的高度和宽度值保存到消息目录中。例19-20输出带有height和width属性的<img/>元素。

例19-20：带高度和宽度属性的本地化<img/>元素

```
<?php
print '<img src="' . img('cancel.png') . '" ' .
      'alt="' . msg('Cancel') . '" ' .
      'height="' . msg('img-cancel-height') . '" ' .
      'width="' . msg('img-cancel-width') . '"/>';
?>
```

其中img-cancel-height和img-cancel-width的本地化消息并不是文本字符串，而是用于描述每个地区的*canel.png*图像大小的整数值。

参见

技巧19.4中有关特定地区消息目录的内容。

19.8 本地化包含文件

问题

你想在你的页面中包含针对特定地区的文件。

方案

在确定了包含针对哪个地区的文件后，将include_path修改为例19-21中所示的代码。

例19-21：通过修改include_path实现包含本地化文件

```
<?php
$base = '/usr/local/php-include';
$LANG = 'en_US';

$include_path = ini_get('include_path');
ini_set('include_path',"$base/$LANG:$base/global:$include_path");
?>
```

讨论

在例19-21中，$base变量中保存的是要包含的本地化文件的基准目录名。非本地化文件保存在$base的全局子目录中，而本地化文件则保存在以相应地区（例如，*en_US*）命名的子目录中。在包含路径中，前置本地化目录和全局目录，会使PHP在查询包含文件时

首先查找这两个位置。将本地化目录放在前面可以保证只在缺少本地化文件时才会加载非本地化文件。

这个技术与技巧19.7中介绍的img()函数类似。不过，这里可以利用PHP的include_path特性对目录进行自动搜索。为保证最大效用，要尽量在代码中提前重置include_path，更好的做法是：对于每次请求通过auto_prepend_file实现在文件顶部加载。

参见

include_path指令的文档（*http://www.php.net/manual/en/configuration.directives.php#ini.include-path*）和auto_prepend_file指令的文档（*http://www.php.net/manual/en/configuration.directives.php#ini.auto-prepend-file*）。

19.9 管理本地化资源

问题

你想实现对多个消息目录和图像的跟踪。

方案

有两种技术可以简化对本地化资源的管理。第一种技术是为每种语言创建一个对象。例如，基于现有的American English扩展创建Canadian English。这样对于与原始语言不同的地方，只需在新对象中修改单词和短语就可以了。

第二种技术：跟踪在新语言中仍需转换的短语，在新语言对象中保留与基准语言中具有相同值的存根。通过查找哪些值在基准语言和新语言中相同，就可以生成需要转换的单词和短语列表。

讨论

例19-22中的*catalog-compare.php*程序会输出两个目录中相同的消息，同时也会输出一个目录没有而另一个目录中有的消息。

例19-22：catalog-compare.php

```
<?php

if (! (isset($_SERVER['argv'][1]) && isset($_SERVER['argv'][2]))) {
    die("Specify two locales to compare.");
```

```
}

$base = 'pc_MC_'.$_SERVER['argv'][1];
$other  = 'pc_MC_'.$_SERVER['argv'][2];

require_once 'pc_MC_Base.php';
require_once "$base.php";
require_once "$other.php";

$base_obj = new $base;
$other_obj = new $other;

/* 检查其他类与基类中相同的消息
 * 或者基类中有但其他类中没有的消息 */
foreach ($base_obj->messages as $k => $v) {
    if (isset($other_obj->messages[$k])) {
        if ($v == $other_obj->messages[$k]) {
            print "SAME: $k\n";
        }
    } else {
        print "MISSING: $k\n";
    }
}

/* 检查其他类中有而基类中没有的消息 */
foreach ($other_obj->messages as $k => $v) {
    if (! isset($base_obj->messages[$k])) {
        print "MISSING (BASE): $k\n";
    }
}
```

在使用这个程序时，需要把每个消息目录对象放在一个与对象同名的文件中（例如，*pc_MC_en_US*类应该放在名为*pc_MC_es_US.php*的文件中，而*pc_MC_es_US*类应该放在名为*pc_MC_es_US.php*的文件中）。然后，就可以在命令行中以这两个地区名作为参数来调用该程序了：

```
% php catalog-compare.php en_US es_US
```

在Web环境下，基于不同的请求原则使用不同的地区和消息目录非常有用。所用的地区可能来自浏览器（在Accept-Language头部信息中），或者由服务器明确地设置（对于不同的虚拟主机有可能设置为以不同的语言显示相同的内容）。如果相同的代码需要基于每次请求原则选择一个消息目录，那么可以像19-23所示的那样对该消息目录类进行技巧化。

例19-23：技巧化消息目录类

```
<?php
// $locale 来自头部信息或虚拟主机名
$classname = "pc_MC_$locale";
```

```
require_once 'pc_MC_Base.php';
require_once $classname.'.php';

$MC = new $classname;
?>
```


参见

技巧19.4中有关消息目录的内容；技巧7.19中有关查找对象的方法和属性的信息。

19.10 使用 gettext

问题

你想要一个创建、管理和部署消息目录的完整系统。

方案

使用PHP的*gettext*扩展，通过该扩展可以使用符合GNU的*gettext*实用程序。例19-24使用了*gettext*函数来输出来自定制消息目录的信息。

例19-24：使用gettext

```
<?php
// 定义一个能找到 "animals" 分类的目录
bindtextdomain('animals','/home/translator/custom/locale');
// 将 'animals' 分类作为默认值
textdomain('animals');


$languages = array('en_US','fr_FR','de_DE');
foreach ($languages as $language) {
    // 改为相应的地区
    setlocale(LC_ALL, $language);
    // 并取得本地化字符串
    print gettext('Monkey');
    print "\n";
}
?>
```

例19-24的输出结果为：

```
Monkey
Singe
Affe
```

讨论

*gettext*是简化应用程序生成多语言消息的一组工具。使用--with-gettext选项编译PHP，能够使函数从*gettext*格式的消息目录中检索合适的文本，而且还可以通过很多外部工具来编辑这些消息目录。

在*gettext*中，消息按照域进行分类，某个特定域中的所有消息都保存在同一个文件当中。bindtextdomain()会告诉*gettext*到哪里去找某个特定域的消息目录。下面的调用：

```
bindtextdomain('animals','/home/translator/custom/locale')
```

表示en_US地区中animals域的消息目录位于以下文件中：

/home/translator/custom/locale/en_US/LC_MESSAGES/animals.mo

textdomain('animals')函数将默认的域设置为animals。调用gettext()就会从默认的域中检索消息。另外一些函数，如dgettext()，可以实现从其他域中检索消息。在调用gettext()（或dgettext()）时，会返回针对当前地区的相应消息。如果分类表中没有与传进参数匹配的当前地区消息，gettext()（或dgettext()）则只返回其参数。因此，如果还没有转换所有消息，你的代码就会以英语（或者其他基准语言）输出那些未转换的消息。

通过textdomain()来设置默认的域，可以简化后面检索该域中消息的操作，因为只需调用gettext('Good morning')，而不用调用dgettext('domain','Good morning')。不过，如果你觉得gettext('Good morning')还是太长，那么可以使用gettext()函数的非正式别名:_()。因此，gettext('Good morning')就可以简写成_('Good morning')。

在*gettext*的网站上，有关于管理编程人员与转换人员之间信息流以及如何有效使用*gettext*的详细而有益的资料。其中也包括能够用于管理信息分类表的其他工具（如一个特殊的GNU Emacs模式）的相关信息。

应该注意*gettext*的一个不足是：它不是线程安全的。如果在一台多线程Web服务器中使用*gettext*，可能会遇到修改一个线程中的设置而影响到其他线程的问题。

参见

*gettext*扩展的文档（*http://www.php.net/gettext*），*gettext*库（*http://www.gnu.org/software/gettext/gettext.html*），Joao Prado Maia的文章“Gettext”（*http://www.onlamp.com/pub/a/php/2002/06/13/php.html*）中所探讨的如何使用诸如xgettext和msgfmt等GNU实用程序生成和维护信息分类表的内容。

19.11 设置传出数据的字符编码

问题

你想要保证浏览器能够正确地处理你的程序发送的UTF-8编码的文本内容。

方案

将PHP的default_encoding配置指令设置为utf-8。这样就能保证PHP传出的HTML响应的Content-Type头部中包含charset=utf-8信息段。该信息段告诉Web浏览器按照UTF-8编码来解析页面内容。

讨论

设置default_encoding会提醒Web浏览器应该使用UTF-8编码来解析你的页面内容。不过，你仍然有责任运用适当的函数来保证页面内容完全使用UTF-8编码。技巧19.13中详细介绍了如何使用UTF-8编码。

如果你不能修改default_encoding配置指令，也可以像例19-25所示的那样自己通过header()函数发送一个适当的Content-Type头部信息。

例19-25：设置字符编码

```
<?php
header('Content-Type: text/html;charset=utf-8');
?>
```

参见

技巧19.13中有关生成UTF-8编码文本的更多内容。

19.12 设置输入数据的字符编码

问题

你想确保进入你程序中的数据流保持一致的字符编码方式，以便对它们进行适宜地处理。例如，你想将所有通过表单提交的数据都视为以UTF-8进行了编码。

方案

虽然不能保证浏览器一定会遵从你发给它们的有关字符编码的指令，但还是有一些办法能够使行为规范的浏览器正常地遵循这些规则。

首先，按照技巧19.11中的指示让你的程序告诉浏览器它所发送的文本内容编码为UTF-8。而且，带有charset信息段的Content-Type头部信息，对于浏览器在提交表单时使用该头部信息指定的字符编码方式进行编码也是个不错的暗示。

其次，在输出的<form/>元素中包含accept-charset="utf-8"属性。尽管并不是所有的Web浏览器都支持该属性，但多数情况下这一属性都能命令浏览器将用户在表单中键入的数据发送到服务器之前，先进行UTF-8编码。

讨论

通常情况下，浏览器所提交的数据与生成包含表单的页面所用的编码方式是一致的。因此，只要你将输出标准化为UTF-8，那么你就可以理所当然地认为能够得到UTF-8编码的输入。accept-charset <form/>属性是HTML 4.0规范的一部分，但并没有被所有HTML解析器（如Web浏览器）实现。

参见

技巧19.11中有关发送UTF-8编码输出的内容：accept-charset<form/>属性的详细介绍（*http://www.w3.org/TR/REC-html40/interact/forms.html#adef-accept-charset*）。

19.13 操纵UTF-8编码的文本

问题

你想在你的程序中使用以UTF-8编码的文本。例如，你想正确地计算多字节字符串的长度，并确保所有文本都以正确的UTF-8编码字符输出。

方案

为了满足对多种UTF-8兼容性的需求，使用一组PHP函数的组合。

如果mbstring扩展有效，则使用其字符串函数来进行UTF-8编码的字符串操作。例19-26使用mb_strlen()函数分别计算了两个UTF-8编码的字符串的字符数目。

例19-26：使用mb_strlen()

```
<?php
// 设置正确的编码
mb_internal_encoding('UTF-8');
// ö是双字节
$name = 'Kurt Gödel';
// 下面每个长字符都是三个字节
$dinner ='불고기';

$name_len_bytes = strlen($name);
$name_len_chars = mb_strlen($name);

$dinner_len_bytes = strlen($dinner);
$dinner_len_chars = mb_strlen($dinner);

print "$name is $name_len_bytes bytes and $name_len_chars chars\n";
print "$dinner is $dinner_len_bytes bytes and $dinner_len_chars chars\n";
?>
```

例19-26的输出结果为：

```
Kurt Gödel is 11 bytes and 10 chars
불고기 is 9 bytes and 3 chars
```

在PHP 5中默认有效的iconv扩展，也提供了一些能够处理多字节字符串的函数，如例19-27所示。

例19-27：使用iconv

```
<?php
// 正确地设置编码
iconv_set_encoding('internal_encoding','UTF-8');
// ö是双字节
$name = 'Kurt Gödel';
// 下面每个长字符都是三个字节
$dinner ='불고기';

$name_len_bytes = strlen($name);
$name_len_chars = iconv_strlen($name);

$dinner_len_bytes = strlen($dinner);
$dinner_len_chars = iconv_strlen($dinner);

print "$name is$name_len_bytes bytes and $name_len_chars chars\n";
print "$dinner is$dinner_len_bytes bytes and $dinner_len_chars chars\n <br/>";

print "The seventh character of $name is " . iconv_substr($name,6,1) . "\n";
print "The last two characters of $dinner are " . iconv_substr($dinner,-2);
?>
```

此外，对于像htmlentities()和htmlspecialchars()这样的函数使用第三个可选的参数，也可让它们将输入的数据作为UTF-8编码的内容来处理，如例19-28所示。

例19-28：UTF-8的HTML编码

```
<?php
$encoded_name = htmlspecialchars($_POST['name'], ENT_QUOTES, 'UTF-8');
$encoded_dinner = htmlentities($_POST['dinner'], ENT_QUOTES, 'UTF-8');
?>
```

讨论

至少在PHP 6发布之前，我们都要始终关注对字符进行适当编码的代价有多大。如果你按照技巧19.11和19.12的指示去做，那么进入到你程序中的数据应该是UTF-8编码的，而浏览器也会正确处理你的程序输出的UTF-8编码的内容。不过，你也因此肩负了两项职责：以适合UTF-8编码的方式操纵字符串和生成UTF-8编码的文本内容。

如果你已经接受了众所周知的国际化理论，那么实现第一项责任就很容易，因为一个字符不是一个字节。而PHP对这一公理的特有理解是，PHP的字符串函数只知道字节，而不知道字符。例如，strlen()函数只计算某个字符串的字节数，而不是其包含的字符数。在使用ISO-8859-1编码以前，这还不是问题 —— 因为在包含256个字符的字符集中每个字符就占用一个字节。但是对于UTF-8编码的字符而言，可能会使用1~4个字节来表示一个字符。mbstring和iconv扩展为一些依照字符到字符原则而不是字节到字节原则行事的字符串函数提供了可以选择的余地。表19-3中列出了这些函数。

表19-3：基于字符的函数

正则表达式函数	mbstring 函数	iconv 函数
strlen()	mb_strlen()	iconv_strlen()
strpos()	mb_strpos()	iconv_strpos()
strrpos()	mb_strrpos()	iconv_strrpos()
substr()	mb_substr()	iconv_substr()
strtolower()	mb_strtolower()	-
strtoupper()	mb_strtoupper()	-
substr_count()	mb_substr_count()	-
ereg()	mb_ereg()	-
eregi()	mb_eregi()	-
ereg_replace()	mb_ereg_replace()	-
eregi_replace()	mb_eregi_replace()	-
split()	mb_split()	-
mail()	mb_send_mail()	-

为了让mbstring适当地处理字符串，应该告诉它使用的是UTF-8编码方式。如例19-26所示，可以通过使用mb_internal_encoding()函数的脚本来实现这一点。或者，也可以在系统范围内设置这个值，即将mbstring.internal_encoding配置指令设置为UTF-8。

使用iconv也有类似的要求。为此，可以使用例19-27中的iconv_set_encoding()函数或者设置iconv.internal_encoding配置指令。

mbstring对正则表达式函数的ereg系列提供了备选方案。不过，对于PCRE（preg_*()）的正则表达式函数，你始终可以使用UTF-8字符串。修饰符u告诉preg函数其中的模式字符串使用的是UTF-8编码，并且启用了在模式中使用多种Unicode属性的支持。例19-29使用了“小写字母”Unicode属性来分别计算两个字符串中小写字母的个数。

例19-29：正则表达式匹配UTF-8

```
<?php
$name = 'Kurt Gödel';
$dinner = 불고기

$name_lower = preg_match_all('/\p{Ll}/u',$name,$match);
$dinner_lower = preg_match_all('/\p{Ll}/u',$dinner,$match);

print "There are $name_lower lowercase letters in $name. \n";
print "There are $dinner_lower lowercase letters in $dinner. \n";
?>
```

例19-29的输出结果为：

```
There are 7 lowercase letters in Kurt Gödel.
There are 3 lowercase letters in 불고기
```

还有一些函数可以用于在其他字符编码与UTF-8编码之间进行转换。utf8_encode()和utf8_decode()函数可以将字符串在ISO-8859-1编码和UTF-8编码之间进行转换。由于ISO-8859-1在很多情况下都是默认的编码方案，因此通过这两个函数可以非常方便地将非UTF-8数据处理成一致的编码格式。例如，pspell扩展使用的字典通常都有自己以ISO-8859-1编码条目。在例19-30中，utf8_encode()函数是把pspell_suggest()的输入结果转换成经过UTF-8适当编码的字符串所必需的。

例19-30：对ISO-8859-1字符串应用UTF-8编码方案

```
<?php
$lang = isset($_GET['lang']) ? $_GET['lang'] : 'en';
$word = isset($_GET['word']) ? $_GET['word'] : 'asparagus';

$ps = pspell_new($lang);
$check = pspell_check($ps, $word);
```

```
print htmlspecialchars($word,ENT_QUOTES,'UTF-8');
print $check ? ' is ' : ' is not ';
print ' found in the dictionary.';
print '<hr/>';

if (! $check) {
    $suggestions = pspell_suggest($ps, $word);
    if (count($suggestions)) {
        print 'Suggestions: <ul>';
        foreach ($suggestions as $suggestion) {
            $utf8suggestion = utf8_encode($suggestion);
            $safesuggestion = htmlspecialchars($utf8suggestion,
                                               ENT_QUOTES,'UTF-8');
            print "<li>$safesuggestion</li>";
        }
        print '</ul>';
}
?>
```

将严格的字符编码看作一种与HTML实体编码类似的任务，可能更有助于认识其重要意义。在各种情况下，为了使文本适应特定环境必须对其进行格式化处理。有了实体编码，通常意味着外部源程序要通过htmlentities()或htmlspecialchars()来实现数据检索。而有了字符编码，则意味着在对数据进行处理（如使用字符相关的函数进行字符串操作）之前先将其转换为UTF-8，并且在输出数据之前保证字符串完成了UTF-8编码。

参见

技巧19.11和19.12中为接收和发送UTF-8编码的字符串和如何在程序中进行设置的内容；mbstring的文档（*http://www.php.net/mbstring*），iconv的文档（*http://www.php.net/iconv*），htmlentities()函数的文档（*http://www.php.net/htmlentities*），htmlspecialchars()函数的文档（*http://www.php.net/htmlspecialchars*），PCRE模式的语法规则（*http://www.php.net/reference.pcre.pattern.syntax*），utf8_encode()函数的文档（*http://www.php.net/utf8_encode*）和utf8_decode()函数的文档（*http://www.php.net/utf8_decode*）。

其他有关管理PHP和字符集方面的比较好的背景资料还有：

- Peter Linsley的文章《An Overview on Globalizing Oracle PHP Applications》（*http://www.oracle.com/technology/tech/php/pdf/globalizing_oracle_php_applications.pdf*）
- PHP WACT Wiki中的字符集/字符编码问题（*http://www.phpwact.org/php/i18n/charsets*）

- Tim Bray 的文章《Characters vs. Bytes》（*http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF*）
- Jukka Korpela的文章《A Tutorial on Character Code Issues》（*http://www.cs.tut.fi/~jkorpela/chars.html*）

第20章

错误处理，故障排除和测试

20.0 概述

把那些花费时间开发Web应用程序的人称为程序员很容易造成误解，因为实际上这些人会把大部分的“编程”时间都花在程序排错上。无论你是在修改编写的代码还是重构在庞大开发环境中很少执行的程序块，几乎都是花费大量的工作时间进行故障排除和测试，一遍又一遍的查错和反复测试。

乱哄哄的聚会，近乎令人发疯的、通宵达旦的排错会议很可能会在你的工作描述中漏掉——该员工从事相当有趣的工作？而事实却是错误、问题、调试和测试构成了程序员生活中的重要组成部分。如果你能够以良好的习惯和技术直面这些问题，那么你就有可能将排错时间减到最少，而尽可能地将时间花在有价值的事情上面。

遗憾的是，许多开发者都不肯在培养错误处理、故障排查和测试能力方面下工夫——但愿你不会犯同样的错误。假如你采用的是被人们亲切称之为悲观编程（pessimistic programming）的工作方式，那么你很可能会在一开始就为将来的错误作出谋划——而你的程序也会从那时起做好能够被适当处理的准备。

技巧20.1到20.11涉及错误处理，包括查找错误来源、确定发生错误时的特征、向最终用户隐藏错误和记录错误，以便在错误发生后能够主导发言踊跃的排错会议。

技巧20.12介绍了如何使用Xdebug——一个开源PHP扩展，该扩展除了支持实时地对代码逐行进行排错，还包含一组强大的代码剖析（code-profiling）功能。

技巧20.13、20.14和20.15对PHP中的单元测试问题进行了探讨，并示范了如何将修复错误变成测试套件，以保证一劳永逸地修改同类错误。

技巧20.16向你介绍了XAMPP，通过它很容易在本地计算机中建立测试环境，从而可以在确定问题来源时拥有一个沙箱（sandbox）环境，不必担心对产品网站造成破坏。

长期以来，培养良好的排错和测试习惯都是开发者们一再逃避的事情。不要再等到下一个项目开始才关注养成良好的习惯了，否则，你将永远与良好的习惯无缘。

20.1 发现并修复解析错误

问题

由于致命的解析错误导致了你的脚本无法运行，你想快速地找出问题所在，然后继续编码。

方案

检查PHP解析器报告出现问题的那一行。如果那一行没问题，以倒推的方式在程序中继续查找，直到找出问题行。

或者使用支持PHP的开发环境，它会在你编码时即时提示语法错误，而且也可以在出现解析错误时对错误来源进行跟踪。

讨论

与大多数编程语言相似，PHP解析器对于脚本编写的方式总喜欢吹毛求疵。如果没有按照应有的格式编写代码，PHP解析器将会中断解析并通知你出现了错误，这就是所说的解析错误。

将下面有错误的程序：

```
<?php
if isset($user_firstname) {
  print "Howdy, $user_firstname!";
} else {
  print "Howdy!";
}
?>
```

保存到名为howdy.php的文件中并运行该文件，PHP会显示下面的错误信息：

```
Parse error: syntax error, unexpected T_ISSET, expecting '(' in
/var/www/howdy.php on line 2
```

根据这条信息，我们知道第二行代码中存在问题，更具体地说，是有关意外的T_ISSET的语法错误。

在PHP将脚本解析为计算机能够理解的格式时，它会把每一行分割成称之为记号（token）的程序块。PHP认可的记号有很多，而且它也知道什么记号应该以什么顺序出现在PHP代码行当中。在上面的解析错误信息中，意外的T_ISSET意味着PHP解析器在那里遇到了不被支持的T_ISSET记号。

继续看解析错误信息，我们发现PHP解析器想要一个T_ISSET标记中缺少的'('。再回过头来看看代码的第二行，的确，if与isset()函数之间缺少一个左括号。

一些支持PHP开发的编码工具能够在运行代码之前对这类问题给出警告。图20-1显示了我们的错误程序在Zend Studio 5中的情形，其中包含了对将来可能出现的解析错误的事先警告。

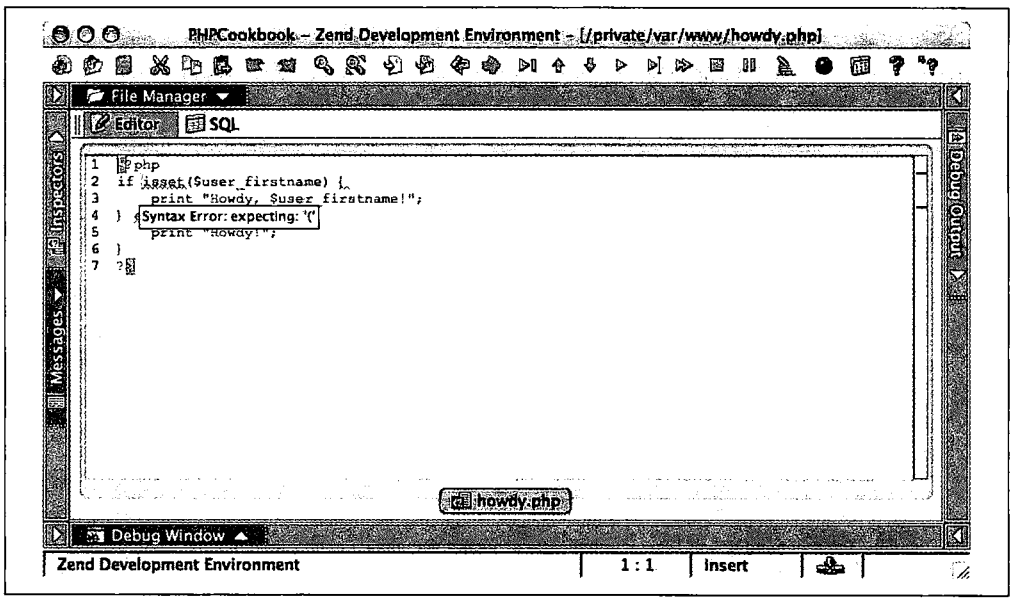

图20-1：Zend Strudio 5在错误发生前察觉并给出警示

并不是所有的错误都会如解析错误那样直接指出问题所在行。有时，错误可能出现在报告的问题出处的前几行中，由于解析器在解析那些行时并没有认为其中有错误，所以不会生成报告，但在解析错误信息中仍然会指出真正错误行中的问题。

如果你很难找到错误来源并且不能使用错误排查工具来帮你发现错误的原因，记住对于任何尝试失败的情况，注释都是你的朋友。在问题所在行被解析错误信息引用之前，给代码块加上必要的注释，然后重新运行令人厌恶的脚本。通过这一排查过程，你将最终发现导致问题的代码行。

参见

PHP解析器标记cheatsheet（*http://www.php.net/tokens*）。

20.2 创建自己的异常类

问题

你想控制如何（或是否）向用户显示错误信息，即使你正在使用几个第三方库，而每个库都有自己处理错误的视图。

方案

利用PHP 5对异常的支持创建你自己的异常处理程序，从而对第三方库中发生的错误按照你的想法进行处理。

```
class CustomException extends Exception
{
  public function __construct($message, $code = 0) {
    // 确保赋值适当
    parent::__construct($message, $code);

    // 记录已知信息
    $msg = "-------------------------------------------------\n";
    $msg .= __CLASS__ . ": [{$this->code}]: {$this->message}\n";
    $msg .= $this->getTraceAsString() . "\n";
    error_log($msg);
  }

  // 重载 __toString() 方法以抑制任何“标准”输出
  public function __toString() {
    return $this->printMessage();
  }

  // 将错误代码映射为输出信息或模板
  public function printMessage() {

    $usermsg = '';
    $code = $this->getCode();

    switch ($code) {
      case SOME_DEFiNED_ERROR_CODE:
        $usermsg = 'Ooops! Sorry about that.';
        break;
      case OTHER_DEFiNED_ERROR_CODE:
        $usermsg = "Drat!";
        break;
      default:
```

```
            $usermsg = file_get_contents('/templates/general_error.html');
            break;
        }
        return $usermsg;
    }

    // 定义静态 exception_handler 执行默认的异常处理
    public static function exception_handler($exception) {
        throw new CustomException($exception);
    }

}

// 保证捕获任何异常
set_exception_handler('CustomException', 'exception_handler');

try {
    $obj = new CoolThirdPartyPackage();
} catch (CustomException $e) {
    echo $e;
}
```

讨论

PHP自PHP 5中引入了异常的概念。异常是其他许多语言都常用的结构，这种结构主要用于稳妥（gracefully）地处理无法预料的错误情况。异常对于你在脚本中使用了第三方库代码，而又对这些代码在不可预知的环境下的行为无法100%地预料到的情况非常有用。比如丢失数据库连接、远程API服务器反应迟钝或者随机性的相似行为等。

异常在脚本中提供了一个可以借之创建沙箱区域的try/catch结构，在该结构中可能会处理极其严重的问题，但却不会对其他代码造成任何伤害。

```
try {
    // 执行某些任务
    $obj = new CoolThing();
} catch (CustomException $e) {
    // 这里，CoolThing并不酷
    print $e;
}
```

为什么在PHP 5提供了功能完善的异常类的情况下，还要使用自定义的异常处理程序呢？因为默认的异常类并没有真正对不可预知的结果实现妥善地处理。它只能输出与常规的错误没有什么区别的错误信息。如果你想对这些不幸的事件实现真正地灵活处理，那么通过自定义的异常处理程序才能做到根据特定的情况来决定具体做什么。

在上面的CustomException类中，包含着两个目标。第一个是记录你能觉察到的任何信息；第二个是要从用户的角度做到尽可能地酷。

其中__construct()方法通过调用父类的构造器（默认异常类的构造器）来设置异常，以保证自定义的异常对象能够使用所有可能设置的值。

然后，马上通过调用error_log()来尽可能地记录完整的信息。你也可以用自己选择的错误记录函数来代替调用error_log()。不过，与稳妥地处理错误的目标保持一致，也要保证你选择的错误记录函数不会导致另一个错误。例如，你想记录的错误信息可能是数据库连接失败，而此时不会将该错误信息记录到同一台数据库服务器。

这样，CustomException类就完成了在调用代码中输出错误的功能。但是，这却不是必要的行为。你甚至可以简单地写出像下面这样的try/catch代码块：

```
try {
  // 执行某些任务
  $obj = new CoolThing();
} catch (CustomException $e) {
  // 这里，CoolThing并不酷
  $e->redirectToOhNoPage();
}
```

其中的catch (CustomException $e)意味着会技巧化一个CustomException类并将该技巧赋给变量$e。此时，$e只是一个保存着某些预定义值和与导致异常问题相关方法的对象，然而从另外的角度来看，这个对象并不同于常规的对象，它也可能会如你想象的那样简单，也可能非常复杂。

标准的错误处理程序同异常之间的一个主要区别就在于恢复这个概念。本技巧中迄今用到的这个例子与你可能已经非常熟悉的PHP 4中set_error_handler()的用法具有很好的参照性。这个概念就是在自定义的异常处理程序中，可以包含当捕获到自定义异常时自动运行的清理例程。该例程能够完美地执行清理工作，并且优雅地退出。

在应用程序流的执行过程中也可以运用异常轻易地从错误中恢复。例如，try块可以同时带有多个catch块，这样就比一大堆if/else/else/else块显得更整洁：

```
try {
  // 执行某些任务
    $obj = new CoolThing();
} catch (PossibleException $e) {
    // 我们认为这种情况可能会发生
    print "<!-- caught exception $e! -->";
    $obj = new PlanB();
} catch (AnotherPossibleException $e) {
    // 我们知道这种情况也可能发生
    print "<!-- aha! caught exception $e -->";
    $obj = new PlanC();
} catch (CustomException $e) {
    // 如果仍然有问题，执行清理工作
```

```
        $e->cleanUp();
        $e->bailOut();
    }
```

在这个例子中，我们能够在不脱离代码块执行流程的基础上，使用try/catch结构来检测异常条件 —— 除非发生未预料到的问题。如果未能按照该程序执行流程从已知的各种可能情况中恢复过来，还可以选择如何处理自定义的异常。甚至还能在catch块内部抛出一个新的异常，从而影响到包含当前执行代码块的try/catch结构的异常起泡顺序。

参见

技巧8.9中更多有关记录错误的内容。异常的相关文档（*http://www.php.net/exceptions*）。

20.3 输出栈轨迹

问题

你想知道程序中特定的地方出现了什么问题，以及导致该问题的原因何在。

方案

使用debug_print_backtrace()。

```
function stooges() {
  print "woo woo woo!\n";
  larry();
}

function larry() {
  curly();
}

function curly() {
  moe();
}

function moe() {
  debug_print_backtrace();
}

stooges();
```

这样会输出：

```
woo woo woo!
#0  moe() called at [backtrace.php:14]
```

```
#1  curly() called at [backtrace.php:10]
#2  larry() called at [backtrace.php:6]
#3  stooges() called at [backtrace.php:21]
```

讨论

debug_backtrace()函数是在PHP 4.3.0中引入的，PHP 5.0.0中进而又推出了方便的debug_print_backtrace()函数。这一对组合使我们能够在调用某个特定的函数之前就可以快速地获得有关程序当前所处状态的直观认识。

你的应用程序越是复杂，就越能从这个轨迹返回函数中获得更有用的信息。在对大型代码库进行故障排除时，你有可能会使用像Xdebug这样的全功能扩展，或者使用PHPEdit、Zend Studio等支持设置断点、巡回代码块、观察变量变化等功能的集成开发环境（IDE，integrated development environment），来更迅速地实现成功的错误追踪（bug-hunting）。

如果你不想通过代码到处声明'Here I am on line ' . __LINE__;得到流水账似的输出，而只是想知道某些关键信息，那么debug_backtrace()和（或）debug_print_backtrace()同样适合你的需要。

如果你使用的是PHP 4，但又想使用只有PHP 5中才有的debug_print_backtrace()功能，那么可以使用PEAR的PHP_Compat兼容包。PHP_Compat中提供了一个与PHP 5的本地debug_print_backtrace()功能等价的实现。

参见

debug_backtrace()函数的文档（*http://www.php.net/debug-backtrace*）和debug_print_backtrace()函数的文档（*http://www.php.net/debug-print-backtrace*），PEAR的PHP_Compat包（*http://pear.php.net/package/PHP_Compat*），Zend Studio IDE（*http://www.zend.com/products/zend_studio*），PHPEdit IDE（*http://www.waterproof.fr/products/PHPEdit/*）。

20.4 读取配置变量

问题

你想得到某个PHP配置设置的值。

方案

使用ini_get()。

```
// 找出包含路径
$include_path = ini_get('include_path');
```

讨论

若要一次得到全部配置变量的值，调用ini_get_all()。该函数返回的是一个包含变量的关联数组，数组的每个元素自身也是一个关联数组。二级数组中包含三个元素：全局设置值、本地值和访问代码。

```
// 把所有配置变量保存到一个关联数组中
$vars = ini_get_all();
print_r($vars['include_path']);
Array
(
    [global_value] => .:/usr/local/lib/php/
    [local_value] => .:/usr/local/lib/php/
    [access] => 7
)
```

其中，global_value是在*php.ini*文件中设置的值，local_value是为了适应Web服务器的配置文件、任何相关的*.htaccess*文件以及当前脚本的变化而经过调整的值。access的值是一个表示在什么位置上能够修改这个值的数字常量。表20-1对access的可能值给予了解释。注意，在这种情况下access有可能会使人误解，事实上该设置值虽然总是能够访问到，但却并不一定能够被调整。

表20-1：access值

值	PHP 常量	含义
1	PHP_INI_USER	任何脚本，使用ini_set()。
2	PHP_INI_PERDIR	目录层次，使用*.htaccess*。
4	PHP_INI_SYSTEM	系统层次，使用*php.ini*或*httpd.conf*。
7	PHP_INI_ALL	任何地方：脚本、目录和系统。

值6表示该设置既可以在目录级也可以在系统级进行修改，因为2+4=6。实际上，没有变量可以只在PHP_INI_USER或PHP_INI_PERDIR级别进行修改，所有变量都能够在PHP_INI_SYSTEM级进行修改，所以返回值的范围基本上都是4、6或7。

也可以通过给ini_get_all()传递一个扩展的名称，获得属于该扩展的变量。

```
// 只返回session模块特定的变量
$session = ini_get_all('session');
```

按照约定，某个扩展的变量将以该扩展的名称加点作为前缀。所以，全部session变量都以session.开始，而所有的Java变量都以java.开始。

因为ini_get()返回的是配置指令的当前值，所以如果你想知道*php.ini*文件中的初始值，则需要使用get_cfg_var()。

```
$original = get_cfg_var('sendmail_from'); // 我们的地址改了吗?
```

get_cfg_var()返回的值与ini_get_all()返回数组中的global_value元素的值是相同的。

参见

技巧20.5中有关设置配置变量的内容。ini_get()函数的文档（*http://www.php.net/ini-get*）、ini_get_all()函数的文档（*http://www.php.net/ini-get-all*）和get_cfg_var()函数的文档（*http://www.php.net/get-cfg-var*）。有关配置变量、默认值以及何时能够修改的完整指令列表（*http://www.php.net/manual/en/ini.php*）。

20.5 设置配置变量

问题

你想改变某个PHP配置设置的值。

方案

使用ini_set()。

```
// 向包含路径中添加一个目录
ini_set('include_path', ini_get('include_path') . ':/home/fezzik/php');
```

讨论

ini_set()并不能永久地改变配置变量的值。修改后的新值只在包含调用ini_set()的请求中有效。如果想永久地改变配置变量，还需要在*php.ini*文件中修改相应的值。

对有些变量的修改是没有意义的，比如asp_tags。因为在调用int_get()修改其设置时，其作用已经发生了。如果无法修改变量的值，ini_set()返回false。

但是，在某些页面中修改配置变量也是很有用的。例如，如果你是在命令行中运行脚本，可以将html_errors设置为关闭。

要想把变量重置为初始值，可以使用ini_restore()。

```
ini_restore('sendmail_from'); // 回到默认值
```

参见

技巧20.4中有关取得配置变量值的内容。ini_set()函数的文档（*http://www.php.net/ini-set*）和ini_restore()函数的文档（*http://www.php.net/ini-restore*）。

20.6 对用户隐藏错误信息

问题

你不想让用户看到PHP的错误信息。

方案

在*php.ini*或者服务器的配置文件中设置下列值：

```
display_errors =off
log_errors     =on
```

如果你拥有编辑服务器端*php.ini*文件的权限，也可以使用ini_set()来设置这些值：

```
ini_set('display_errors', 'off');
ini_set('log_errors', 'on');
```

这些设置用于告诉PHP不要将错误信息作为HTML发送给浏览器，而是将其保存到服务器的错误日志中。

讨论

在log_errors设置为启用时，错误信息就会被写入到服务器的错误日志中。如果你想把PHP的错误信息写入到一个单独的文件中，可以将error_log配置指令设置为该文件的路径名：

```
error_log   = /var/log/php.error.log
```

或者：

```
ini_set('error_log', '/var/log/php.error.log');
```

如果error_log被设置为syslog，PHP的错误信息将会通过*syslog(3)*发送到系统日志程序（Unix）或者事件日志（Windows）。

虽然有许多错误信息 —— 如用户填写的表单不正确 —— 应该通知用户，但仍然有必要将一些会反映出你的脚本中存在问题的内部错误信息对用户隐藏起来。原因有二：首先，这些错误信息看起来会令人（专家级用户）怀疑你的程序不专业或者导致人们（初级水平的用户）的疑惑不解。如果在将数据保存到数据库的过程中出现了问题，你可以检查数据库返回的查询代码并向用户显示致歉信息，恳请用户稍后再试。而将PHP返回的含糊不清的错误信息直接发送给用户显然无助于提升你网站的信用。

其次，将错误信息显示给用户会导致安全隐患。基于你使用的数据库及错误的类型，很可能在错误信息中包含着如何登录数据库或者服务器的信息，并且还可能暴露数据库的结构。恶意用户会利用这些信息对你的网站实施攻击。

例如，如果你的数据库服务器当机了，而你使用的是mysql_connect()与其通讯，PHP则会生成下列警告信息：

```
<br>
<b>Warning</b>:  Can't connect to MySQL server on 'db.example.com' (111) in
<b>/www/docroot/example.php</b> on line <b>3</b><br>
```

如果把这则信息发送给用户的浏览器，那么用户就会知道你的数据库服务器名为*db.example.com*并可能尝试对其进行攻击。

参见

技巧8.9中有关如何记录错误信息的内容；技巧20.5中有关通过ini_set()设置配置值的更多信息。PHP配置指令的文档（*http://www.php.net/configuration*）。

20.7 调谐错误处理

问题

你想修改某个页面中错误记录的灵敏度，这样你就可以控制哪些类型的错误会被报告。

方案

使用error_reporting()来调整PHP报告错误的类型。

```
error_reporting(E_ALL);                 // 全部
error_reporting(E_ERROR | E_PARSE);     // 仅限主要问题
error_reporting(E_ALL & ~E_NOTICE);     // 除提示之外的所有错误
```

讨论

每一个错误都有与之关联的错误类型。例如，如果以一个字符串作为参数调用array_pop()，PHP会报告“This argument needs to be an array”，因为该函数只能对数组进行操作。与这一报告信息相关的错误类型是E_NOTICE —— 一种非致命的运行时间问题。

在默认情况下，错误类型的报告级别是E_ALL & ~E_NOTICE，这意味着除提示之外的所有错误。其中，&等同于逻辑与，~等同于逻辑非。但是，在*php-recommended.ini*配置文件中，将错误报告的级别设置为了E_ALL，即全部错误类型。

PHP 5中引入了一个新的错误级别 —— E_STRICT。在开发过程中启用E_STRICT可以让PHP始终提供能够改进代码的方法，还会在使用不推荐的函数时收到警告，并附带一些有关编码的最新和最佳的推荐方式。E_STRICT是唯一不包含在E_ALL中的错误级别。因此要想在开发过程中最大程度的涵盖各种错误类型，就需要将错误报告级别设置为E_ALL | E_STRICT。

标记为提示的错误信息是比警告的严重程度低的运行时间问题。提示并不一定意味出错，但是会指出潜在的问题。E_NOTICE的一个例子是“Undefined variable”，这个问题发生在你使用尚未赋值的变量的情况下：

```
// 产生 E_NOTICE 提示
foreach ($array as $value) {
    $html .= $value;
}

// 不会生成任何错误信息
$html = '';
foreach ($array as $value) {
    $html .= $value;
}
```

在前面第一种情况下，第一次执行foreach循环时，$html是未定义的。所以，在追加该变量值时，PHP会提示你追加的是一个未定义的变量。在第二种情况下，通过在循环之前将一个空字符串赋给$html就避免了E_NOTICE错误。由于变量默认的值是空字符串，

因此前面这两段代码最终会生成相同的结果。由于我们可能会拼错变量的名称，所以E_NOTICE错误是很有用的，比如：

```
foreach ($array as $value) {
    $hmtl .= $value; // 噢！应该是 $html
}

$html = ''
foreach ($array as $value) {
    $hmtl .= $value; // 噢！应该是 $html
}
```

通过自定义的错误处理函数，能够基于错误类型解析错误并采取相应的行动。有关错误类型的完整列表见表20-2。

表20-2：错误类型

值	常量	描述	能否捕获
1	E_ERROR	无法补救的错误	No
2	E_WARNING	可恢复的错误	Yes
4	E_PARSE	解析程序错误	No
8	E_NOTICE	可能的错误	Yes
16	E_CORE_ERROR	类似E_ERROR，但由 PHP核心生成	No
32	E_CORE_WARNING	类似E_WARNING，但由PHP核心生成	No
64	E_COMPILE_ERROR	类似E_ERROR，但由PHP核心生成	No
128	E_COMPILE_WARNING	类似E_WARNING，但由Zend Engine生成	No
256	E_USER_ERROR	类似E_ERROR，但通过调用trigger_error()触发	Yes
512	E_USER_WARNING	类似E_WARNING，但通过调用trigger_error()触发	Yes
1024	E_USER_NOTICE	类似E_NOTICE，但通过调用trigger_error()触发	Yes
2047	E_ALL	除了E_STRICT之外类型	N/A
2048	E_STRICT	运行时提示，PHP有关改进代码质量的修改建议（PHP 5以后）	N/A

其中能够捕获的错误表示可以由通过set_error_handler()注册的函数来处理。而其他错误则表示问题可能比较严重，如果由用户来处理恐怕会不安全，因此必须由PHP来处理。

参见

技巧20.8中示范的如何定义自定义错误处理函数的内容；error_reporting()函数的文档

（*http://www.php.net/error-reporting*）和set_error_handler()函数的文档（*http://www.php.net/set-error-handler*）；有关错误的更多信息（*http://www.php.net/manual/en/ref.errorfunc.php*）。

20.8 使用自定义错误处理函数

问题

你想创建一个自定义的错误处理函数，以便控制PHP报告错误的方式。

方案

通过set_error_handler()来设置自己的错误函数。

```
set_error_handler('pc_error_handler');

function pc_error_handler($errno, $error, $file, $line) {
    $message = "[ERROR][$errno][$error][$file:$line]";
    error_log($message);
}
```

讨论

通过自定义的错误处理函数能够基于错误的类型来解析错误并采取相应的措施。技巧20.7的表20-2中列出了所有的错误类型。

向set_error_handler()中传递一个函数名，PHP就会把所有错误都交由该函数来处理。自定义的错误处理函数可以接受五个参数。第一个参数是错误的类型，如表示E_NOTICE的8。第二个参数是由错误抛出的信息，如“Undefined variable:html”。第三和第四个参数是PHP发现错误所在的文件名称和行数。最后一个参数保存着在当前作用域中定义的所有变量及其值的数组。

例如，在下面的代码中，$html在没有被赋予初始值的情况下就进行追加操作。

```
error_reporting(E_ALL);
set_error_handler('pc_error_handler');

function pc_error_handler($errno, $error, $file, $line, $context) {
    $message = "[ERROR][$errno][$error][$file:$line]";
    print "$message";
    print_r($context);
}
```

```
$form = array('one','two');

foreach ($form as $line) {
    $html .= "<b>$line</b>";
}
```

当生成“Undefined variable”错误时，pc_error_handler()会输出：

```
[ERROR][8][Undefined variable:  html][err-all.php:16]
```

在输出初始错误信息后，pc_error_handler()还会输出一个包含所有全局变量、环境变量、请求变量以及session变量的数组。

表20-2是能够捕获的错误表示可以由用set_error_handler()注册的函数进行处理。而其他错误则表示问题可能比较严重，如果由用户来处理恐怕会不安全，因此必须由PHP来处理。

参见

技巧20.7中列出的不同错误类型信息。set_error_handler()函数的文档（*http://www.php.net/set-error-handler*）。

20.9 记录错误

问题

你想将程序错误保存到日志中，这些错误可能是解析器错误和未找到文件、错误的数据库查询和断开的连接等各种错误。

方案

使用error_log()将错误写入日志中。

```
// LDAP 错误
if (ldap_errno($ldap)) {
    error_log("LDAP Error #" . ldap_errno($ldap) . ": " . ldap_error($ldap));
}
```

讨论

记录错误有利于排除程序故障，准确的错误记录有助于修正错误。应该始终做到对出错原因进行记录。

```
$r = mysql_query($sql);
if (! $r) {
    $error = mysql_error();
    error_log('[DB: query @'.$_SERVER['REQUEST_URI']."][$sql]: $error");
} else {
    // 处理结果
}
```

如果只是简单的记录错误的发生，而不记录相关的支持性信息，那么就会降低错误记录的可用性：

```
$r = mysql_query($sql);
if (! $r) {
    error_log("bad query");
} else {
    // 处理结果
}
```

另一种有用的技术就是在错误信息中包含__FiLE__，__LINE__，__FUNCTION__，__CLASS__和__METHOD__等“魔术”常量：

```
error_log('['.__FiLE__.']['.__LINE__."]: $error");
```

其中，__FiLE__常量指的是当前文件名，而__LINE__指的是当前行数。

__FUNCTION__常量是在PHP 4.3.0版中加入的。从该版本起到其他的PHP 4.x系列版本，__FUNCTION__常量都是以小写形式返回当前的函数名，但从PHP 5开始，该常量改为返回声明时候的函数名。__CLASS__常量返回当前的类名，该常量也是在PHP 4.3.0版中加入的。__CLASS__与_FUNCTION__返回值的大小写形式在PHP 4.x与PHP 5这两个版本是一致的。

PHP 5.0.0 中引入了__METHOD__常量，该常量返回当前类方法的名称。而且，是依据该方法声明时的大小写形式，返回相同大小写形式的方法名称。

参见

技巧20.6中对用户隐藏错误信息的讨论。error_log()函数的文档（*http://www.php.net/error-log*）。“魔术”常量的文档（*http://www.php.net/manual/en/language.constants.predefined.php*）。

20.10 消除“headers already sent”错误

问题

你想通过header()或setcookie()发送一个HTTP头部信息，但PHP报告“headers already sent”错误信息。

方案

这种错误发生在调用header()或setcookie()之前发送非头部信息输出的情况下。因此，需要修改代码将任何输出都放在发送头部信息之后。

```
// 正确的
setcookie("name", $name);
print "Hello $name!";

// 错误的
print "Hello $name!";
setcookie("name", $name);

// 正确的
<?php setcookie("name",$name); ?>
<html><title>Hello</title>
```

讨论

HTTP信息包含头部和主体信息，而且是按照先后顺序发送给客户端的。如果首先发送了主体信息，那么就不能再发送头部信息了。所以，如果你在输出HTML后才调用setcookie()，PHP是无法发送相应的Cookie头部信息的。

同样地，需要删除包含文件附带的任何空白符。因为如果包含文件中带有位于<?php ?>标签之外的空白行，那么这个空白行就会被发送到浏览器，所以需要使用trim()来删除文件中前导或后置的空白行。

```
$file = '/path/to/file.php';

// 备份
copy($file, "$file.bak") or die("Can't copy $file: $php_errormsg);

// 读取并删除首尾空白
$contents = trim(join('',file($file)));

// 写入
$fh = fopen($file, 'w')  or die("Can't open $file for writing: $php_errormsg);
if (-1 == fwrite($fh, $contents)) { die("Can't write to $file: $php_errormsg);}
fclose($fh)              or die("Can't close $file: $php_errormsg);
```

按照一个目录接一个目录的原则来对文件进行处理，比按照一个文件接一个文件的原则来处理更方便。技巧24.7中介绍了如何处理一个目录的中所有文件。

保证包含文件中不会带有任何附加空白符的另一种完全合法的手段是不写关闭的?>标签。如果包含文件中是纯PHP代码，那么这种方法就能保证你不用再对该文件中的空白符进行处理。有关这一方法更多的信息，请参考*http://www.php.net/manual/en/language.basic-syntax.instruction-separation.php*。

如果你不想因为空行而中断头部信息的发送，也可以打开输出缓冲。输出缓冲能够防止PHP随时将所有输出都发送到客户端。如果对输出进行了缓冲，就可以任意对头部信息和主体信息进行组合。但是对用户而言，这样就好像是服务器在处理请求时需要花费更长的时间，因为他们需要更长的时间才能看到浏览器显示输出结果。

参见

技巧8.12中有关输出缓冲的讨论；技巧24.7中有关处理一个目录中的所有文件的介绍；header()函数的文档（*http://www.php.net/header*）。

20.11 记录调试信息

问题

你想通过向输出变量中添加说明文字来辅助排除故障，也想在生产和调试模式之间轻松地来回切换。

方案

将一个基于定义好的常量有条件地输出信息的函数，放在以auto_prepend_file配置设置包含的页面文件当中。将下列代码保存到*debug.php*文件中。

```
// 开启调试模式
define('DEBUG',true);

// 普通的调试函数
function pc_debug($message) {
    if (defined('DEBUG') && DEBUG) {
        error_log($message);
    }
}
```

在*php.ini*或你站点的*.htaccess*文件中设置auto_prepend_file指令。

```
auto_prepend_file=debug.php
```

然后，就可以在代码中调用pc_debug()来输出调试信息了。

```
$sql = 'SELECT color, shape, smell FROM vegetables';
pc_debug("[sql: $sql]"); // 只有DEBUG为true时才输出
$r = mysql_query($sql);
```

讨论

代码调试是代码编写过程中一个必要步骤，有很多技术可以帮助你快速地找到并解决问题。其中多数涉及到包含确保代码正确性的支持平台（scaffolding）。程序越复杂，支持平台就越有必要。Fred Brooks在他的《The Mythical Man-Month（中文书名：人月神话。译者注）》（Addison-Wesley）中推断“支持平台中有近一半代码会出现在产品中”。提前且适当地规划，能够保证你以清晰而高效的方式将支持平台整合到程序设计逻辑当中。但这也要求你必须做到事先度量和记录，以及如何对支持平台收集的数据进行分类等进行深入地规划。

一种筛选信息的技术是为不同的调试注释类型指定不同的优先级，之后调试函数只输出那些高于当前优先级的信息。

```
define('DEBUG',2);

function pc_debug($message, $level = 0) {
    if (defined('DEBUG') && ($level > DEBUG) {
        error_log($message);
    }
}

$sql = 'SELECT color, shape, smell FROM vegetables';
pc_debug("[sql: $sql]", 1); // 不输出，因为 1 < 2
pc_debug("[sql: $sql]", 3); // 输出，因为 3 > 2
```

另一种技术是编写包含附加信息的包装函数，对性能调谐提供帮助，比如，优化查询数据库所需的时间。

```
function db_query($sql) {
    if (defined('DEBUG') && DEBUG) {
        // 如果DEBUG启用，则开始对查询执行时间记时
        $DEBUG_STRING = "[sql: $sql]<br>\n";
        $starttime = microtime(true);
    }

    $r = mysql_query($sql);

    if (! $r) {
        $error = mysql_error();
```

```
            error_log('[DB: query @'.$_SERVER['REQUEST_URI']."][$sql]: $error");
        } elseif (defined(DEBUG) && DEBUG) {
            // 查询没有失败且DEBUG开启，则结束计时
            $endtime = microtime(true);
            $elapsedtime = $endtime - $starttime;
            $DEBUG_STRING .= "[time: $elapsedtime]<br>\n";
            error_log($DEBUG_STRING);
        }

        return $r;
    }
```

在这个例子中，不仅将SQL记录到了错误日志，同时还记录了MySQL执行该请求时所用的时间。这样你就会发现某些查询是否会耗费过多的时间。技巧21.1中更详细地讨论了有关计算代码执行时间和PHP 4中与microtime(true)兼容的备选方案的内容。

最后，可能你会想到要整合PEAR的日志包，该包为针对抽象的记录系统提供了一个有效的框架。PEAR Log预定义了八个日志级别：PEAR_LOG_EMERG，PEAR_LOG_ALERT，PEAR_LOG_CRIT，PEAR_LOG_ERR，PEAR_LOG_WARNING，PEAR_LOG_NOTICE，PEAR_LOG_INFO和PEAR_LOG_DEBUG。这个日志包为自定义错误日志提供了强大的选项分类机制，包括把错误记录到SQLite和（或）浏览器弹出窗口。

参见

define()函数的文档（*http://www.php.net/define*）；defined()函数的文档（*http://www.php.net/defined*）和error_log()函数的文档（*http://www.php.net/error-log*）。Frederick P.Brooks著的《The Mythical Man-Month》（Addison-Wesley）。PEAR Log的主页（*http://pear.php.net/package/Log*）。

20.12 使用扩展的调试程序

问题

你想在脚本运行期间交互地进行调试。

方案

通过使用Xdebug的远程调试客户端程序，能够以交互的方式检查数据结果、设置断点，进入、移出或跳过代码片段。

讨论

Xdebug扩展为辅助开发提供了很多有用的功能，包括与Kcachegrind兼容的代码剖析。在本技巧中，我们主要讨论Xdebug的交互调试能力。为了验证本技巧中介绍的内容，你需要能够编译并安装一个Zend扩展，也就是说必须要有权编辑系统中的*php.ini*文件。PHP的dl()扩展加载函数不能用于Xdebug。本技巧中的例子适用于Xdebug 2.0.0版。

安装Xdebug扩展是一个直观的过程。你可以从源代码开始，也可以使用pecl命令。

```
% pecl install xdebug-beta
```

当对该扩展编译并安装后，接着需要编辑*php.ini*文件并提供到*xdebug.so*模块的完整路径，比如zend_extension = /usr/lib/php/extensions/no-debug-non-zts-20050922/xdebug.so。

要进行交互调试，还需要下载一份Xdebug源代码的拷贝。默认情况下，*pecl*安装过程中不会安装绑定的Xdebug应用程序客户端。这是因为在多数情况下，xdebug.so模块都安装在一台远程服务器上，而*debugclient*工具一般都安装在一台单独的机器中。不过，在本地Web服务器的测试环境中同时运行PHP和xdebug.so模块也没有任何问题。

在下载与通过*pecl*命令安装的模块版本一致的Xdebug源代码后，先解压缩然后将当前目录修改为debugclient分配的目录名，并发出以下命令。

```
% cd debugclient
% ./configure
% make
% sudo make install
```

*debugclient*的二进制文件会安装到/usr/local/bin。可以通过运行*debugclient*命令来对安装结果进行测试。你可能会看到类似如下的信息。

```
% debugclient
Xdebug Simple DBGp client (0.9.1)
Copyright 2002-2005 by Derick Rethans.

Waiting for debug server to connect.
```

默认情况下，*debugclient*工具会通过9000端口侦听来自启用了Xdebug的PHP脚本的连接，也可以通过使用-p切换命令来使*debugclient*通过其他端口来侦听，命令格式如下：

```
% debugclient -p port
```

只要保证你的脚本或*php.ini*文件中的设置与你设置的*debugclient*侦听端口一致即可。

通过一个PHP脚本的命令行调用（invocation）就可以启动Xdebug所支持的交互调试，也可以通过向运行Xdebug和PHP脚本的Web服务器传递一个适当的值来启动。为了在命令行中开始交互会话，需要在触发脚本之前设置一些环境变量。

```
% export XDEBUG_CONFiG="idekey=session_name remote_enable=1"
% php myscript.php
```

如果要调试的是一个运行在一台远程服务器中的网页，只要向请求URL中添加XDEBUG_SESSION_START=name，就能设置一个名为XDEBUG_SESSION_START的cookie。当设置了XDEBUG_SESSION_START的get或post变量，或存在XDEBUG_SESSION cookie时，Xdebug扩展就会尝试连接到在*php.ini*的xdebug.remote_host字段中指定的*debugclient*。

连接成功后，剩下的步骤基本都是相同的，在交互过程中采取的主要操作如表20-3所示。

表20-3：常见的Xdebug命令

命令	说明
run	启动或恢复执行脚本，直到遇到断点或者脚本执行完毕
step_into	步进下一条语句。如果调用了一个函数，Xdebug会在该函数的第一条语句处暂停
step_over	步进下一条语句。如果放出step_over命令时所在的行调用了一个函数，调试程序会在放出该命令的同一个作用域中调用函数后停止
step_out	步出当前作用域并在从当前函数中返回时暂停语句执行
stop	立即结束脚本执行
set_breakpoint	在调试会话中设置一个新断点
property_value	取得一个属性值
context_get	在给定的栈深度的环境中返回一个属性数组。如果省略栈深度，则使用当前的栈深度

记住了这些选项后，就可以开始调试程序了。例20-1示范了一个向访问者说hello的极为复杂的程序，其过于复杂的地方主要表现在它没有表明应该如何问候到达网站的访问者，而只是为我们提供了一些在调试交互中跨函数调用的选项。

例20-1：过度复杂的问候程序。

```
<?php
$user = 'Curly';

function sayHello($user, $greeting = 'Hello, %s!') {
  if (!validUser($user)) {
    $greeting = 'Hey! What are you doing here, %s?!';
```

```
  }
  printf($greeting, $user);
}

function validUser($user) {
  // 在此进行验证
  return true;
}

sayHello($user);
?>
```

我们会在通过浏览器触发该程序后对其进行交互调试。如果你需要设置本地测试环境的指导，可以参考技巧20.16。首先，启动*debugclient*。

```
% debugclient
Xdebug Simple DBGp client (0.9.1)
Copyright 2002-2005 by Derick Rethans.

Waiting for debug server to connect.
```

现在，加载包含例20-1中程序的文件overhello.php，并通过浏览器来访问该页面，要保证在查询字符串后面添加正确的Xdebug值 —— 例如，http://localhost/overhello.php?XDEBUG_SESSION_ START=foo。你所等候的*debugclient*应该注册该连接并输出以下信息：

```
Connect
<init fileuri="file:///Users/clay/Sites/overhello.php" language="PHP"
protocol_version="1.0" appid="4148" idekey="foo"><engine version="2.0.0beta5">
<![CDATA[Xdebug]]></engine><author><![CDATA[Derick Rethans]]></author><url>
<![CDATA[http://xdebug.org]]></url><copyright><![CDATA[Copyright (c)
2002-2005 by Derick Rethans]]></copyright></init>
(cmd)
```

你会注意到浏览器并没有生成任何输出 —— 它只是在等待你继续进行调试会话。为此，我们每次进入程序中的一行。Xdebug使用的DBGp协议需要每个请求同时都要发送一个标识符，所以下列命令会传递一个我们已经通过返回服务器启动的标识符。

```
(cmd) step_into -i foo
<response command="step_into" transaction_id="foo" status="break" reason="ok">
</response>
(cmd)
```

这个响应的含义是Xdebug扩展收到了你的请求并向前前进了一行，同时暂停执行并等待下一次调试请求。为了确认当前所执行的位置，可以在任何时候发出stack_get请求。

```
(cmd) stack_get -i foo
<response command="stack_get" transaction_id="foo"><stack where="{main}"
```

```
level="0" type="file" filename="file:///Users/clay/Sites/overhello.php"
lineno="1"></stack></response>
(cmd)
```

来自stack_get的响应告诉我们，现在我们已经到达了脚本的{main}部分。换句话说，当前位置不是函数或类的内部，而是正运行通过主文件，并且是在第一行，我们再移动到另一行。

```
(cmd) step_into -i foo
<response command="step_into" transaction_id="foo" status="break"
reason="ok"></response>
(cmd)
```

*debugclient*再次返回了ok响应，也再次将状态设置为中断并等待我们发送其他指令。那么对这一行我们应该做什么呢？

```
(cmd) context_get -i foo
<response command="context_get" transaction_id="foo">
<property name="user" fullname="$user" type="uninitialized">
</property></response>
(cmd)
```

我们看到通过请求环境，$user变量浮出水面。DBGp协议在返回变量值时非常谨慎，除非明确的请求否则不会返回值。即便如此，你也会发现这些值都是经过Base64编码的，这样就可以保护*debugclient*免受意外数据的困扰：

```
(cmd) property_value -n user -i foo
<response command="property_value" transaction_id="foo"
type="null"></response>
(cmd)
```

我们命令Xdebug给出变量$user的值，而它返回的是一个null。为什么？这是因为在到达第二行后之前$user的值实际上还没有设置，而*debugclient*一开始就读取了第二行。因此，我们要先发送另一个step_into命令，再来请求property_value：

```
(cmd) step_into -i foo
<response command="step_into" transaction_id="foo"
status="break" reason="ok"></response>
(cmd) property_value -n user -i foo
<response command="property_value" transaction_id=
"foo" type="string" encoding="base64">
<![CDATA[Q3VybHk=]]></response>
(cmd)
```

这一次得到了经过Base64编码的$user值，但我们怎样才能知道其中真正的内容呢？只需在一个单独的终端窗口中对该值进行解码即可：

```
% php -r "echo base64_decode('Q3VybHk=');"
Curly
```

到现在为止，一切正常。然而，如果像这样一次一行地调试一个大型的程序恐怕就会成为一件痛苦而费时的事情了。那么，我们先回到采取操作之前。为此，我们需要设置断点并让程序运行到断点为止。

```
(cmd) breakpoint_set -i foo -t call -m sayHello
<response command="breakpoint_set" transaction_id="foo" id="41480001"></response>
(cmd)
```

-t开关是告诉Xdebug我们要在调用类型处设置一个断点，而-m开关则表示在调用sayHello函数的地方设置中断。这个响应为断点设置了一个唯一的ID，通过该ID（或者一个完全实现了DBGp协议的IDE）我们可以引用该断点。现在可以运行该程序到它再次暂停，并再次取得我们的定向：

```
(cmd) run -i foo
<response command="run" transaction_id="foo" status="break" reason="ok"></response>
(cmd) stack_get -i foo
<response command="stack_get" transaction_id="foo"><stack where="sayHello" level="0"
type="file" filename="file:///Users/clay/Sites/overhello.php" lineno="5"></stack>
<stack where="{main}" level="1" type="file" filename=
"file:///Users/clay/Sites/overhello.php" lineno="16"></stack></response>
(cmd)
```

在这个的例子程序中，我们知道sayHello函数是在第5行定义，在第16行被调用的，所以这个响应与我们所知的程序情况是一致的，而且它还返回到了声明函数的地方，因而我们可以进入该函数并检查其中是否存在问题。我们现在就来试一下：

```
(cmd) step_into -i foo
<response command="step_into" transaction_id="foo" status="break" reason="ok">
</response>
(cmd) stack_get -i foo
<response command="stack_get" transaction_id="foo"><stack where="sayHello"
level="0" type="file" filename="file:///Users/clay/Sites/overhello.php"
lineno="5"></stack><stack where="{main}" level="1" type="file" filename=
"file:///Users/clay/Sites/overhello.php" lineno="16"></stack></response>
(cmd) context_get -i foo
<response command="context_get" transaction_id="foo"><property name="greeting"
 fullname="$greeting" address="60271800" type="string" encoding="base64">
<![CDATA[SGVsbG8sICVzIQ==]]></property><property name="user" fullname="$user"
address="60264872" type="string" encoding="base64"><![CDATA[Q3VybHk=]]>
</property></response>
(cmd)
```

这一系列命令向我们表明，程序目前已经运行到了第16行，而且已经调用了sayHello函数。我们在该脚本第5行的开始，查看对sayHello调用，并没有给它传递任何参数，而且

正准备进入if()块通过运行validUser()来检查$user是否有效。因为简单的程序还没有完成，所以先跳过这个块，而不要为运行尚未完成的函数而浪费时间。在跳过这个块之后，再检查我们到了哪。

```
(cmd) step_over -i foo
<response command="step_over" transaction_id="foo" status="break"
reason="ok"></response>
(cmd) stack_get -i foo
<response command="stack_get" transaction_id="foo">
<stack where="sayHello" level="0" type="file" filename=
"file:///Users/clay/Sites/overhello.php" lineno="8">
</stack><stack where="{main}" level="1" type="file" filename=
"file:///Users/clay/Sites/overhello.php" lineno="16">
</stack></response>
(cmd)
```

如你所见，我们现在位于if()块后面的第8行。下面让程序正常运行以便完成整个程序。

```
(cmd) run -i foo
<response command="run" transaction_id="foo" status="stopped"
reason="ok"></response>
(cmd)
```

最后，run命令返回了一个stopped状态，这是因为程序在运行其他部分时没有遇到任何额外的断点。

当然，这个例子是比较简单的，但是通过它我们足以看到如何使用Xdebug来操纵你的程序并实时地查看各种情况。你可能也发现使用支持Xdebug的IDE会更比使用*debugclient*简单一些，比如使用自由编辑器软件WeaverSlave。然而，通过*debugclient*能够轻易地并且独立地提供大量对你应用程序进行深入分析的功能。

参见

Xdebug的文档（*http://www.xdebug.org/*）；DBGp协议的文档（*http://www.xdebug.org/docs-dbgp.php*）；支持Xdebug的编辑器，WeaverSlave的相关资料（*http://weaverslave.ws/*）。

20.13 编写单元测试

问题

你正在从事一个扩展了一组核心功能的项目，而你想要一种随着项目进展能保证一切正常的简单方式。

方案

编写一个用于测试函数或类的核心功能的单元测试，使其在发生某些中断时向你提供警告信息。

使用PHP-QA的.phpt测试系统的例子如下：

```
--TEST--
str_replace() function
--FILE--
<?php
$str = 'Hello, all!';
var_dump(str_replace('all', 'world', $str));
?>
--EXPECT--
string(13) "Hello, world!"
```

讨论

在PHP中可以通过许多方式编写单元测试。可能一连串简单的.phpt测试就能满足你的需要，但你也可能会需要像PHPUnit或SimpleTest这样的更加结构化的测试方案。我们会分别对他们进行讨论，但必须先回答一个问题：为什么要首先写一个单元测试呢？

不论使用什么语言来编写程序都非常类似于剥洋葱皮，只不过方向相反而已。从洋葱的中心开始，一层一层地包起来，直到形成最终产品——洋葱。

核心上包裹的层数越多，核心延续期望的功能就越重要。而保证程序核心延续预期功能——特别是在修改以后——的最简单方式就是贯穿单元测试。

在前面的例子中，我们用一个字符串替换另一个字符串进而成功地测试了str_replace()函数。该测试不会关心str_replace()是如何写的——关键问题在于它能否按照重现原则如期实现相应功能。

运行.phpt测试的最简单方式就是将测试内容保存到扩展名为.phpt（例如：str_replace.phpt）的文件中，然后使用PEAR内置的.phpt执行工具，像下面这样：

```
% pear run-tests str_replace.phpt
```

对此你会看到类似下面的输出结果。

```
Running 1 tests
PASS str_replace() function[str_replace.phpt]
TOTAL TIME: 00:00
1 PASSED TESTS
0 SKIPPED TESTS
```

你可以通过创建多个*.phpt*文件并运行这些文件来测试你代码功能中的多方面特性：

```
% pear run-tests *.phpt
```

要了解.phpt文件的详细结构信息，可以访问*http://qa.php.net/write-test.php*。

也可以通过PHPUnit/PHPUnit2单元测试框架来编写单元测试。PHPUnit2只对PHP 5有效，而PHPUnit则能在PHP 4或PHP 5以下的版本中使用。

上面的例子可以编写为如下的PHPUnit2测试。

```
require_once 'PHPUnit2/Framework/TestCase.php';

class StrreplaceTest extends PHPUnit2_Framework_TestCase
{
    public function testStrreplaceWorks()
    {
        $str = 'Hello, all!';
        $this->assertEquals('Hello, world!', str_replace('all', 'world', $str));
    }
}
```

将这些代码保存到一个名为*StrreplaceTest.php*的文件中。只要安装了PHPUnit2，就可以像下面这样来运行该测试：

```
% phpunit StrreplaceTest
```

以上命令会查找名为*StrreplaceTest.php*的文件，并运行其中定义的测试。

PHPUnit是一个非常强大的单元测试框架，它能够做的远不止这个例子中运行的测试这么简单。在*http://www.phpunit.de*中可以找到PHPUnit的完整文档资料。

另一种流行的单元测试框架是SimpleTest。默认情况下，这个框架的测试是在浏览器（尽管也可以通过命令行）中运行的。如果使用SimpleText来编写我们对`str_replace()`函数的测试，可能会写成下面这样：

```
require_once 'simpletest/unit_tester.php';
require_once 'simpletest/reporter.php';

class TestStrreplace extends UnitTestCase
{
    function testStrreplace()
    {
        $str = 'Hello, all!';
        $this->assertEqual('Hello, world!', str_replace('all', 'world', $str));
    }
}
```

从这个例子中可以看到，SimpleTest与PHPUnit非常类似。不过，SimpleTest框架与PHPUnit的不同之处在于它的功能全面。在决定哪一个框架最适合你的需要之前，最好还是对这两个框架进行一次彻底的比较。要学习有关SimpleTest的更多内容，可以访问*http://www.lastcraft.com/simple_test.php*。

参见

.phpt单元测试的文档（*http://qa.php.net/write-test.php*），PHPUnit和PHPUnit2的资料（*http://www.phpunit.de*）。SimpleTest的资料（*http://www.lastcraft.com/simple_test.php*）。

20.14 编写单元测试套件

问题

你想做到按照正规的原则，方便地运行多个单元测试。

方案

将多个单元测试包装为一个称作单元测试套件的组中。

讨论

几乎没有任何程序能够简单到仅通过一个单元测试就能满足其生命周期中的所有测试需要。随着时间的推移和程序规模的增大，必然会出现越来越多的测试需求。有的可能是为了测试新功能，而有的可能是为了验证修复的错误是否仍然处于被修复的状态。

当你的测试库增大到足够多时，就会发现把它们组织成一组单元测试套件反而更方便。所谓单元测试套件，本质上就是对一组测试进行包装，通过引用该包装的名称即可运行所有测试。

使用SimpleTest框架，我们可以创建一个测试套件，在PHP中，测试包括str_replace函数在内的更多功能，可以把许多与字符串相关的函数都放到一个单独的文件中。例如，在一个名为string_tests.php的文件中，我们可能会增加如下内容。

```
class TestStringfunctions extends UnitTestCase
{
    function testStrreplace()
    {
```

```
        $str = 'Hello, all!';
        $this->assertEqual('Hello, world!', str_replace('all', 'world', $str));
    }

    function testSubstr()
    {
        $str = 'Hello, all!';
        $this->assertEqual('e', substr($str, 1, 1));
    }
}
```

这样我们通过运行TestStringfunctioins类就可以完成两项测试。我们可以再创建一个名为array_tests.php的类似文件，在其中定义如下内容。

```
class TestArrayfunctions extends UnitTestCase
{
    function testArrayflip()
    {
        $array = ('foo' => 'bar', 'cheese' => 'hotdog');
        $flipped = array_flip($array);
        $this->assertEqual('foo', reset($flipped));
    }

    function testArraypop()
    {
        $array = ('foo' => 'bar', 'cheese' => 'hotdog');
        $popped = array_pop($array);
        $this->assertEqual('hotdog', $popped);
        $this->assertEqual(1, sizeof($array));
    }
}
```

有了4个可以运行的测试，就可以把它们组织成一个套件，以便检查确认程序正常运行所需要的功能，调用一次执行所有测试。我们的测试套件可能看起来如下：

```
require_once 'simpletest/unit_tester.php';
require_once 'simpletest/reporter.php';

$test = new GroupTest('All tests');
$test->addTestFile('string_tests.php');
$test->addTestFile('array_tests.php');

if (TextReporter::inCli()) {
    exit ($test->run(new TextReporter()) ? 0 : 1);
} else {
    $test->run(new HtmlReporter());
}
```

将这些代码保存到名为test_suite.php的文件中，然后就可以在一个正确地安装了PHP以及在脚本中将路径设置为正确映射SimpleTest安装位置的浏览器中运行该文件了。

如果是使用PHP CLI在shell中运行，会得到类似如下的结果。

```
% php test_suite.php
All tests
OK
Test cases run: 4/4. Failures: 0, Exceptions: 0
```

通过以上手段，你可以构建起包含大量测试但仍然能够通过一条命令触发所有测试的自动测试系统。

注意当测试运行于CLI模式时三元操作符的使用，这种运行单元测试组件的方法可以允许脚本在作为外部自动测试脚本运行时，返回成功或失败的条件。

参见

SimpleTest的文档（*http://www.lastcraft.com/simple_test.php*）；PHPUnit和PHPUnit2的资料（*http://www.phpunit.de*），其中包含在PHPUnit中如何创建测试套件的信息。

20.15 对网页应用单元测试

问题

你的应用程序不能分割成小的可测试程序块，或者你只想将单元测试应用到你的访问者能够看到的网站中。

方案

以SimpleTest的WebTestCase类为中心编写一系列单元测试去测试你网站的最终输出。

创建一个文件来测试你网站 —— *example.com*中的有关内容。在一个名为exampledotcom_tests.php的文件中，增加以下内容。

```
class TestOfExampledotcom extends WebTestCase
{
    // 基本主页加载
    function testHomepageLoading()
    {
        $this->assertTrue($this->get('http://www.example.com/'));
    }

    // 测试对 "FAQ"的点击以及FAQ生成的结果
    function testFaq()
    {
```

```
        $this->get('http://www.example.com/');
        $this->clickLink('FAQ');
        $this->assertTitle('Example FAQ');
        $this->assertWantedPattern('/have a question\?/i');
    }
}
require_once 'simpletest/web_tester.php';
require_once 'simpletest/reporter.php';

$test = new GroupTest('Web site tests');
$test->addTestFile('exampledotcom_tests.php');

if (TextReporter::inCli()) {
    exit ($test->run(new TextReporter()) ? 0 : 1);
} else {
    $test->run(new HtmlReporter());
}
```

讨论

如果你要处理的是一个全部或部分由PHP程序代码驱动的站点，有时候很难写出一个能够测试压缩功能（encapsulated functionality）的小型单元测试。也许你只是想保证网站能够正确运行，如果不能，那么就进行调试。

无论你的基本代码能否分割成单元测试，网页单元测试都是一种方便而且实用的技术。毕竟，真实的环境并不总是与你的测试环境一致，所以建立一个守护作业（cron job）来运行网页测试套件是非常有益的。

SimpleTest的`WebTestCase`支持对导航、内容、cookie和表单处理的测试。甚至还能在框架集（framesets）中选择框架并在特定的框架内执行测试。

如果你的站点中大量使用了JavaScript，那么可以关注一下Selenium，它是一个开源的测试框架，用于在浏览器中测试完整的用户体验。Selenium提供了一种基于浏览器的IDE，能够记录自动测试生成的操作并对基于Windows、Mac OS X和Linux的浏览器提供了广泛的支持。

参见

SimpleTest的文档（*http://www.lastcraft.com/simple_test.php*）。Selenium项目的页面（*http://www.openqa.org/selenium/*）。

20.16 设置测试环境

问题

你想在对PHP脚本进行测试时，不用担心破坏网站或者污染你的生产开发环境。

方案

使用XAMPP在你的桌面电脑中为你的程序设置测试环境。

讨论

为Web应用程序设置本地运行环境的复杂性经常会使开发人员裹足不前。其结果经常是对最佳开发实践的弱化，比如在产品网站中作为一个特权用户来编辑文件——永远是不可取的！

XAMPP项目为四个平台 —— Windows 98/NT/2000/XP, Mac OS X, Linux（SuSE/RedHat/Mandrake/Debian）和Solaris —— 提供了一键安装的解决方案。这个包中包含了Apache，MySQL，PHP和PEAR，phpMyAdmin以及eAccelerator的同期版本。

通过简单的、步进式的安装过程，XAMPP项目将会令你在你的本地计算机中创建Web应用程序运行环境变成了点击鼠标的过程。

要处理大型数据集？令人遗憾的是，需要开发处理大量频繁更新内容网站的开发人员发现，由于缺乏良好的测试数据而不得不眼睁睁地看着自己最佳的开发方法流于形式。不要让这种事情发生在你的身上！你只需要写一个简单的在本地映射你的数据结构的脚本，然后通过使用完整数据集的子集快照（subset snapshot）来预先更新你的本地数据拷贝。这样一来，你就可以获得一个相关数据的当前拷贝，而且这个拷贝的数据量对于测试和开发者来说也足够用了。

参见

XAMPP项目的主页*http://www.apachefriends.org/en/xampp.html*。

第21章

性能调谐和负载测试

21.0 概述

PHP本身的速度是相当快的。一般来说，PHP程序中比较慢的部分必须与外部资源进行交互，等待数据库查询执行完成或者从远程URL中抽取内容等。也就是说，你的PHP程序本身可能不能实现它应有的效率。本章所介绍的就是如何发现和修复代码中存在的性能问题的技术。

在软件工程领域中，对在开发过程中开始优化的最佳时间问题有许多争论。优化进行得过早，有可能会因为过多地陷入不重要的细节问题而浪费时间；而优化得太晚，可能又会发现自己不得不重新编写程序中的大段代码。

对于这种两难的问题，一个保险的办法就是养成善于处理小问题的良好习惯，而其益处最终会体现出来。例21-1中所示为在PHP 5.1.2中生成与MD5散列完全相同的三种方式。

例21-1：生成散列的三种方式

```
// PHP中基本的md5() 函数
$hashA = md5('optimize this!');

// 作为mhash扩展的MD5
$hashB = bin2hex(mhash(MHASH_MD5, 'optimize this!'));

// 在PHP 5.1.2+中通过hash()函数实现的MD5
$hashC = hash('md5', 'optimize this!');
```

$hashA、$hashB和$hashC的结果全都是83f0bb25be8de9106700840d66f261cf。然而，第三种方式的速度是PHP基本md5()函数速度的两倍。

但是对于类似这样的密集测试（head-to-head tests）进行优化需要考虑一些问题，比如必须知道该函数在你的程序中被调用的次数和替代方案的可读性及可维护性。

例如，在选择散列函数时，如果你需要在PHP 5.1.2之前的版本中运行代码，那么只能是全部使用md5()，或者根据PHP的版本（以及是否安装了mhash扩展）检查来决定使用哪个函数。md5()与hash()之间的绝对时间差大约是十分之一毫秒。如果你需要一次性地计算成千上万个散列，那么插入额外的计算例程来选择最快的函数就是有意义的。但是，如果只是为了省下微不足道的一小点时间，而增加许多额外的复杂性就不值得了。

优化并不是在真空中进行的。当你对你的代码采取优化手段时，不仅可以调整原来的执行时间，也会影响到代码的规模、可读性和可维护性。几乎总是有必要的场合会需要极快的执行时间。而更常见也是更有价值的是节省编程和故障排除的时间。你必须在处理各种优化难题的过程中积极平衡这些利害关系。

本章首先介绍几种与你的开发例程集成的简单分析方法。技巧21.1介绍了如何计算函数的执行时间。技巧21.2详细讨论了计算整个程序运行时间的步骤。在技巧21.3中可以学到如何更进一步地运用这些简单的方法，其中包括使用排错程序扩展对程序进行分析。

技巧21.4中有关于如何对你的网站进行压力测试的简要介绍，会提醒你比代码本身更需要性能调谐的内容：网络潜伏以及硬件问题。

许多PHP脚本中最突出的瓶颈问题是滥用正则表达式。技巧21.5介绍了不使用高开销的正则表达式就能解决文本匹配问题的一些方法。

技巧21.6中介绍了几个PHP加速器，解释了流行的开源和商业加速器程序中常用的手段，以及如何通过使用它们而获益。

21.1 计算函数执行时间

问题

你有一个函数，且想知道执行这个函数需要多长时间。

方案

比较函数执行前后的毫秒数，就能知道函数执行所花费的时间：

```
<?php

// 创建一个无意义的长字符串
$long_str = uniqid(php_uname('a'), true);
```

```
// 从这里开始计时
$start = microtime(true);

// 测试函数
$md5 = md5($long_str);

$elapsed = microtime(true) - $start;

echo "That took $elapsed seconds.\n";
?>
```

讨论

要确定执行一个单独的函数所需的时间，可能不需要一个完整的基准程序包，比如PEAR Benchmark（在技巧21.2中有介绍）。作为替代方案，也可以通过microtime()函数得到你想要的信息。

参见

技巧3.13中讨论的microtime()函数的用法，包括如何在PHP 4中使用它的介绍；microtime()函数的文档（*http://www.php.net/microtime*）；技巧21.2中有关PEAR Benchmark的信息。

21.2 计算程序执行时间

问题

你想对一段程序代码进行粗略分析，看一看执行每条语句需要多长时间。

方案

使用PEAR Benchmark模块：

```
<?php
require_once 'Benchmark/Timer.php';

$timer =& new Benchmark_Timer(true);

$timer->start();
// 此处是一些设置代码
$timer->setMarker('setup');
// 此处是更多的已执行代码
$timer->setMarker('middle');
```

```
// 此处仍然有很多代码
$timer->setmarker('done');
// 最后还有一点代码
$timer->stop();

$timer->display();
?>
```

讨论

调用setMarker()记录时间。display()方法用于输出记号列表、设置每个记号的时间以及从前一个记号起经过的时间。

```
------------------------------------------------------------------------
marker       time index              ex time                  perct
------------------------------------------------------------------------
Start        1029433375.42507400     -                        0.00%
------------------------------------------------------------------------
setup        1029433375.42554800     0.00047397613525391      29.77%
------------------------------------------------------------------------
middle       1029433375.42568700     0.00013899803161621      8.73%
------------------------------------------------------------------------
done         1029433375.42582000     0.00013303756713867      8.36%
------------------------------------------------------------------------
Stop         1029433375.42666600     0.00084602832794189      53.14%
------------------------------------------------------------------------
total        -                       0.0015920400619507       100.00%
------------------------------------------------------------------------
```

Benckmark模块中也包含Benchmark_Iterate类，这个类可以用于记录一个单独函数执行多次的时间：

```
<?php
require 'Benchmark/Iterate.php';

$timer =& new Benchmark_Iterate;

// 要计时的示例函数
function use_preg($ar) {
    for ($i = 0, $j = count($ar); $i < $j; $i++) {
        if (preg_match('/gouda/',$ar[$i])) {
            // 是gouda
        }
    }
}

// 另一个要计时的示例函数
function use_equals($ar) {
    for ($i = 0, $j = count($ar); $i < $j; $i++) {
        if ('gouda' == $ar[$i]) {
            // 是gouda
        }
```

```
    }
}

// 运行 use_preg() 1000 次
$timer->run(1000,'use_preg',
            array('gouda','swiss','gruyere','muenster','whiz'));
$results = $timer->get();
print "Mean execution time for use_preg(): $results[mean]\n";

// 运行 use_equals() 1000 次
$timer->run(1000,'use_equals',
            array('gouda','swiss','gruyere','muenster','whiz'));
$results = $timer->get();
print "Mean execution time for use_equals(): $results[mean]\n";
?>
```

其中的Benckmark_Iterate::get()方法返回一个关联数组。该数组中的mean元素保存着每一次迭代相应函数所使用的平均时间。iterations元素保存着迭代的次数。每次函数迭代执行的时间保存在一个带有整数键的数组元素。例如，第一次迭代的时间保存在$results[1]中，而第37次迭代的时间保存在$results[37]中。

如果要自动记录每一行PHP代码执行所经过的时间，可以使用declare结构和ticks指令：

```
function profile($display = false) {
    static $times;

    switch ($display) {
    case false:
        // 将当前时间添加到记录的时间列表中
        $times[] = microtime();
        break;
    case true:
        // 以微秒为单位返回花费的时间
        $start = array_shift($times);

        $start_mt = explode(' ', $start);
        $start_total = doubleval($start_mt[0]) + $start_mt[1];

        foreach ($times as $stop) {
            $stop_mt = explode(' ', $stop);
            $stop_total = doubleval($stop_mt[0]) + $stop_mt[1];
            $elapsed[] = $stop_total - $start_total;
        }

        unset($times);
        return $elapsed;
        break;
    }
}
```

```
// 注册tick处理程序
register_tick_function('profile');

// 记录开始时间
profile();

// 执行代码，记录每条语句的执行时间
declare (ticks = 1) {
    foreach ($_SERVER['argv'] as $arg) {
        print strlen($arg);
    }
}

// 输出花费的时间
$i = 0;
foreach (profile(true) as $time) {
    $i++;
    print "Line $i: $time\n";
}
```

你可以通过ticks指令来基于可重复原则对一个代码块执行一个函数。赋给ticks的数值表示在使用register_tick_function()注册的函数执行之前要执行多少条语句。

在前面的例子中，我们注册了一个单独的函数profile()，并通过这个函数在declare块中执行每条语句。如果$_SERVER['argv']中有两个元素，profile()就会执行四次：包括foreach每次循环的时间和print strlen($arg)代码行每次执行的时间。

也可以将过程设置为每执行三条语句调用两个函数：

```
register_tick_function('profile');
register_tick_function('backup');

declare (ticks = 3) {
    // 代码……
}
```

还可以给注册的函数传递额外的参数，该参数可以是对象方法而非常规函数：

```
// 给profile()传入 "parameter"
register_tick_function('profile', 'parameter');

// 调用 $car->drive();
$car = new Vehicle;
register_tick_function(array($car, 'drive'));
```

如果你想执行一个对象方法，可以将该对象及方法名封装在数组中传递。这样register_tick_function()就可以知道你引用的是一个对象而非函数。

调用unregister_tick_function()可以从tick函数列表中删除注册的函数：

```
unregister_tick_function('profile');
```

参见

PEAR Benchmark类的有关信息（*http://pear.php.net/package/Benchmark*）；register_tick_function()函数的文档（*http://www.php.net/register-tick-function*），unregister_tick_function()函数的文档（*http://www.php.net/unregister-tick-function*）和declare结构的文档（*http://www.php.net/declare*）。

21.3 通过排错扩展进行代码剖析

问题

你想用一个健壮的解决方案来剖析自己的应用程序，以便能够持续地监控程序的哪个部分最耗费时间。

方案

使用剖析（profiling）和排错扩展，比如Advanced PHP Debugger（APD）或Xdebug，这两个扩展都在PECL知识库中。

在安装了APD之后，就可以通过在你的脚本顶部加上adp_set_pprof_trace()来在一个可配置目录中生成跟踪文件。对该跟踪文件进行分析，就能得到PHP脚本在执行过程中如何分配时间的详细信息：

```
<?php
$dumpdir = '/tmp';
$processid = posix_getpid();
ini_set('apd.dumpdir', $dumpdir);

// 准备输出一个基本的报告
$dumpfile = $dumpdir . '/pprof.' . $processid . '.0';

// 开始跟踪
apd_set_pprof_trace();

// 我们将要剖析的函数
function pc_longString() {
    return uniqid(php_uname('a'), true);
}

function pc_md5($str) {
    return md5($str);
}
```

```
function pc_mhashmd5($str) {
    return bin2hex(mhash(MHASH_MD5, $str));
}

function pc_hashmd5($str) {
    return hash('md5', $str);
}

// 运行这些函数
$str = pc_longString();

$md5 = function_exists('md5') ? pc_md5($str) : false;
$md5 = function_exists('mhash') ? pc_mhashmd5($str) : false;
$md5 = function_exists('hash') ? pc_hashmd5($str) : false;

echo "now run:\n";
echo "  /usr/bin/pprofp -R $dumpfile\n";
echo "to view a report.\n";
```

由运行pprofp工具（这一工具是作为APD包的一部分安装的），会得到类似如下的结果：

```
% /usr/bin/pprofp -R /tmp/pprof.16704.0

Trace for /home/clay/phpckbk2/apd/md5.php
Total Elapsed Time = 0.00
Total System Time  = 0.00
Total User Time    = 0.00


        Real         User        System              secs/     cumm
%Time (excl/cumm)  (excl/cumm) (excl/cumm) Calls    call    s/call  Memory Usage Name
--------------------------------------------------------------------------------
100.0 0.00 0.00  0.00 0.00  0.00 0.00        1  0.0005    0.0042            0 main
77.8 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0033    0.0033     33554432 apd_set_pprof_trace
4.6 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0000    0.0002            0 pc_longString
2.6 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0001    0.0001            0 php_uname
2.3 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0000    0.0001            0 pc_mhashmd5
1.5 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0001    0.0001            0 uniqid
1.5 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0000    0.0001            0 pc_md5
1.5 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0001    0.0001            0 mhash
1.3 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0001    0.0001            0 md5
1.3 0.00 0.00  0.00 0.00  0.00 0.00    3  0.0000    0.0000          0 function_exists
1.2 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0000    0.0001            0 pc_hashmd5
1.0 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0000    0.0000            0 hash
0.6 0.00 0.00  0.00 0.00  0.00 0.00      1  0.0000    0.0000            0 bin2hex
```

讨论

安装了APD之后，启动一次在程序运行时保存报告的剖析会话就是一件简单的事了。由APD生成的输出文件可以放在任何你认为合适的地方，如果你想在脚本中整合一些有条件的跟踪执行，甚至可以把这个文件放在一个动态位置中，只需在调用

apd_set_pprof_trace()之前通过ini_set('apd.dumpdir','*/path/to/writable/dump/directory*')来挑选一个位置即可。

生成的pprof文件是可以使用计算机来处理的，其中保存着你的脚本运行过程的详细信息。被保存在apd.dumpdir位置的输出文件的命名遵守pprof.*process* ID.*file number*的格式约定。这个文件的file number部分由APD内部决定，该部分会为每个跟踪的进程创建一系列必需的顺序文件。

通过绑定的*pprofp*命令解释（shell）应用程序来处理这些输出文件，能够得到一些诸如哪些部分会占用程序执行的大部分时间之类的基本信息。而知道了运行时间最长的函数，也就等于知道了对程序采取优化措施的目标。

除了执行时间的基本报告信息之外，*pprof*文件也可以通过*pprof2calltree*转换工具转换为Kcachegrind兼容的文件。Kcachegrind是一个能够深入到应用程序中查找瓶颈问题根源的GUI程序。

还有一些其他的剖析扩展；而为了最有效地剖析并排除程序问题，我们建议你逐个尝试一下，再决定哪一个最适合你的需要。

参见

APD的文档（*http://www.php.net/manual/en/ref.apd.php*）；Xdebug剖析和排错扩展的资料（*http://www.xdebug.org/*）；DBG扩展的资料（*http://dd.cron.ru/dbg/*）；Kcachegrind可视化剖析工具的资料（*http://kcachegrind.sourceforge.net*）。

21.4 对网站进行压力测试

问题

你想知道如何使你的网站顺利地执行繁重的负载。

方案

使用压力测试和基准工具来模拟多种级别的负载。

讨论

人们经常会混淆压力测试和基准测试，而认识到这两者之间的区别是至关重要的。

对一个网站进行基准测试通常是一种由单独的开发者完成的临时活动。其中最常用的工具是Apache HTTP 服务器的基准测试工具 —— *ab*，该工具用来测试一台HTTP服务器每秒钟能够响应多少次请求。例如：

```
% /usr/bin/ab -n 1000 -c 100 -k
www.example.com/test.php
```

这个测试会以100次并发请求为一组，返回基于1000次对*http://www.example.com/test.php*请求的平均响应时间的报告。

这种测试虽然也有价值 —— 它能够帮你合理地估算在正常负载的情况下服务器每秒钟能够响应多少次请求，但它却不能告诉你当整个Web程序面临繁重的负载时会出现什么情况。毕竟它一次只能测试一个URL。

压力测试是一种能够中断你的Web应用程序的测试技术。通过对断点的测试，你就能够识别并修复应用程序中的弱点，或者对何时需要购置新硬件设备有更好的判断。当这种技术与代码剖析共同使用时，你还可以明确程序中的哪一部分应该优先剥离，也就是说，你是否需要在添置更多的前台Web服务器之前为你的数据库集群添置更多的服务器？

压力测试的一个极好的开源工具是*Siege*。可以通过配置，让Siege从一个配置文件中读取多个URL并按照顺序（回归测试）逐个运行这些URL，也可以让它读取一个列表或者一些URL并随机地选中其中一个来执行 —— 这更接近于网站的真实应用。Siege也可以像*ab*一样一次只测试一个URL。

如果你不能在系统中安装Siege，那么Lincoln Stein的*torture.pl*脚本是一个很好的替代方案。*torture.pl*中融合了许多Siege中的设计理念，而且还能生成与Siege类似的报告。

参见

Siege的源代码和文档（*http://www.joedog.org/JoeDog/Siege*）；*ab*的文档（*http://httpd.apache.org/docs/2.0/programs/ab.html*）；*torture.pl*的源代码和文档（*http://stein.cshl.org/~lstein/torture/*）。

21.5 避免使用正则表达式

问题

你想通过优化字符串匹配操作来提升脚本的性能。

方案

用速度更快的字符串和字符类型函数代替不必要的正则表达式。

讨论

不必要计算的一种常见来源就是在没有必要的情况下使用正则表达式，比如，在验证一个提交表单中的用户名是否只包含字母和数字字符时。

解决这一问题的常见方法是使用正则表达式：

```
<?php
if (!preg_match('/^[a-z0-9]*$/i', $username)) {
  echo 'please enter a valid username.';
}
?>
```

而同样的测试可以通过ctype_alnum()函数以更快的速度完成。

通过使用技巧21.1中的代码计时技术，我们可以将ctype_alnum()的执行速度与正则表达式的执行速度进行比较：

```
<?php
$username = 'foo411';

$start = microtime(true);

if (!preg_match('/^[a-z0-9]*/i', $username)) {
    echo 'please enter a valid username';
}

$regextime = microtime(true) - $start;

$start = microtime(true);

if (!ctype_alnum($username)) {
    echo 'please enter a valid username';
}

$ctypetime = microtime(true) - $start;

echo "preg_match took:  $regextime seconds\n";
echo "ctype_alnum took: $ctypetime seconds\n";
?>
```

输出结果如下：

```
preg_match took:  0.000163078308105 seconds
ctype_alnum took: 9.05990600586E-06 seconds
```

我们看到，ctype_alnum()的速度明显更快一些；9.05990600586E-06等于0.00000906秒，这个速度是preg_match()正则表达式的18倍，而结果实际上是一样的。

对于一个复杂的应用程序，用等价的替代方案取代不必要的正则表达式的累计结果，会使其性能得到显著提高。

当你在编码并决定是否需要使用正则表达式时的一块试金石是，你所执行的匹配能否以一个简明的语句来说明。没错，有一些匹配，如“是一个有效的电子邮件地址的字符串”，如果没有复杂的正则表达式就无法实现真正的验证。然而，“检查字符串A中是否包含字符串B”就可以通过其他手段来测试，这种情况最终可能只需要一个非常简单的测试，而不需要正则表达式：

```
$haystack = 'The quick brown fox jumps over the lazy dog';
$needle = 'lazy dog';

// 最慢
if (ereg($needle, $haystack)) echo 'match!';

// 慢
if (preg_match("/$needle/", $haystack)) echo 'match!';


// 快
if (strstr($haystack, $needle)) echo 'match!';

// 最快
if (strpos($haystack, $needle) !== false) echo 'match!';
```

在决定使用正则表达式之前先仔细地确定ctype和字符串函数是否更有效肯定是有益的，特别是当你在处理一段重复循环的代码时。

参见

ctype函数的文档（*http://www.php.net/manual/en/ref.ctype.php*），字符串函数的文档（*http://www.php.net/manual/en/ref.strings.php*）；正则表达式函数的文档（*http://www.php.net/manual/en/ref.pcre.php*）。

21.6 使用加速器

问题

你想增强PHP应用程序的性能。

方案

安装一个能够进行代码高速缓存的PHP加速器，以避免PHP根据每个请求把脚本编码成操作码。

讨论

为了跳过对每次请求的编译步骤，PHP代码加速器会通过将PHP脚本的编码版保存在磁盘或者共享内存中，来实现其大部分显而易见的魔法操作。

当通知PHP解释器运行特定的程序时，它会读取程序的源代码并将其编译为简洁的内部表现形式。然后，再执行编译结果中包含的指令。当脚本执行完成时，解释器就会丢弃编译结果。

相对而言，加速器则会将编译后的指令保存起来。当下一次PHP解释器根据请求要执行同一程序时，加速器就会介入并检查是否保存了该程序的编译结果。如果有，它就会通知PHP解释器不用再重新编译，只要直接执行保存的编译结果即可。可以通过配置使加速器根据不同的条件来更新编译结果，比如当初始程序改变时或者只有明确通知它需要更新时。

三个最流行的免费PHP加速器是Alternative PHP Cache（APC）、eAccelerator和ionCube PHP Accelerator（PHPA）。同样免费的Zend Optimizer也能通过明显地纠正常见但效率偏低的编码习惯实现对性能的小幅提升。但是，Zend Optimizer并不是加速器。

这些加速器可能会有一些影响你选择的重要差别。比如Windows支持问题、PHP兼容性、Zend Optimizer兼容性以及超出代码缓存的可操作性问题等等，都可能会影响到你选用哪一种加速器。基准测试表明，所有这些加速器对性能的提升作用大致相当，所以决定性的因素很可能就是以上提到的这些因素之一。

APC和eAccelerator是正在开发中的开源项目（eAccelerator是已经停止开发的Turck MMCache项目的派生项目）。ionCube加速器是一个商业产品。通常，开发者对于报告错误的响应速度也是影响你决定使用哪一个加速器的因素之一。要确保检查每个加速器报告错误的当前状态，以及这些问题将如何基于你使用的PHP版本以及程序遵循的编码类型（面向对象或者过程式）对你的程序产生的影响。

最后，很重要的一点就是要知道加速器的兼容性通常都会落后于新发布的PHP版本。如果运行一个最新而且最强大的PHP版本对你而言非常重要，你可能会发现基于现有加速器所提供的功能几乎没有使用加速器的必要。

参见

APC的网站（*http://pecl.php.net/package/apc*）；eAccelerator的网站（*http://www.eaccelerator.net/*）；PHPA的网站（*http://www.php-accelerator.co.uk/*）。

第22章

正则表达式

22.0 概述

正则表达式是进行模式匹配和文本操纵的一种复杂而强大的工具。虽然正则表达式不及纯粹的文本匹配速度那么快，但却极其灵活。正则表达式通过其简单 —— 虽然简洁但满是符号 —— 的语法构造模式能够匹配几乎任何可以想象出来的字符组合。如果你的网站需要从文本文件式的数据源（如体育比赛结果、新闻标题或其他频繁更新的内容）中取得数据，那么正则表达式就能够帮你从这些数据源中提取出有意义的信息。

本章会对基本的正则表达式语法作一简明介绍，然后集中讨论PHP中提供的使用正则表达式的函数。如果想了解更多有关正则表达式的细节，可以参考PHP在线手册的PCRE部分（*http://www.php.net/pcre*）和David Sklar著的《Learning PHP 5》一书的附录B。如果想系统地学习正则表达式，请阅读Jeffrey E.F. Friedl著的《Mastering Regular Expressions》这本全面介绍正则表达式的书。

通过正则表达式在纯文本和HTML之间相互转换是非常方便的。幸运的是，由于这些功能的可用性很强，因而PHP内置了许多处理类似任务的函数。在技巧9.10中我们介绍了转义HTML实体的函数，在技巧13.14中介绍了去掉HTML标签的方法，而在技巧13.12和13.13中也分别介绍了如何将纯文本转换成HTML以及将HTML转换为纯文本。在技巧9.4中详细讨论了运用正则表达式匹配并验证电子邮件地址的方法。

几年来，正则表达式的应用领域已经从其最初的基本应用日益向整合应用发展。因而，PHP也提供了两组不同的正则表达式函数。第一组中包含的是传统的（或POSIX）函数，每个函数的名称都以`ereg`（意为“扩展”，而且`ereg`函数本身也是对原始功能集合的一个扩展）作为开头。另一组中包含的是Perl兼容的函数，函数名都以前缀`preg`（意为Perl兼容的正则表达式）开头。

preg函数使用了一个模仿Perl语言正则表达式功能的库。这样你就能像使用Perl语言一样通过正则表达式来实现很多强大的功能，比如非贪婪匹配、向前和向后断言匹配，甚至递归模式匹配等。

一般来说，没有任何理由再使用ereg函数了。因为它们所提供的功能有限，而且比preg函数的速度更慢。然而，由于ereg函数在PHP中先于preg函数存在已经有很多年了，因此许多程序员在旧程序代码中或者出于习惯仍然使用着ereg函数。值得庆幸的是，这两组函数的原型是相同的，所以在这两组函数之间互相转换并不会带来多大的问题（我们在技巧22.1中列举了如何实现这一点并消除几个主要的难题）。

可以把正则表达式想象成一门非常严格的编程语言。而这门正则表达式语言的唯一任务就是匹配文本模式。在正则表达式的模式当中，多数字符都用于匹配字符自身。也就是说，正则表达式rhino匹配包含着五个字符序列的字符串rhino。而奇特的事情都是由一群称为元字符的标点和符号搞出来的。这些元字符的作用不是直接与自身匹配，而是对正则表达式匹配程序下达命令。

最常用的元字符包含句点（.）、星号（*）、加号（+）和问号（?）（如果想让元字符直接与自身匹配，需要在元字符前面加上反斜杠）。

- 句点的意思是“匹配任意字符”。因此，模式.at匹配bat、cat甚至rat。
- 星号的意思是“匹配0或多个前面的对象”（迄今为止，我们知道的唯一对象就是字符）。
- 加号的含义类似于星号，但是指“匹配1或多个前面的对象”。因此，.+at匹配brat、sprat甚至catestrophe中的cat，但不匹配at。如果要匹配at，必须将+换成*。
- 问号的意思是“前面的对象是可选的”。也就是说，它匹配0或1个前面的对象。color既匹配color也匹配colour。

如果想对多个字符应用*或+，需要把相应的字符序列放在一个圆括号中构成一个对象。通过圆括号可以将字符组织成更复杂的模式，也能捕获与其中对象匹配的部分文本序列。捕获的文本序列能够被preg_replace()引用，进而对字符串进行修改，而所有捕获的匹配能够被保存在一个作为preg_match()和preg_match_all()的第三个参数传递的数组变量中。preg_match_all()函数与preg_match()是类似的，但前者会找到字符串中包含的所有匹配项，而不是像后者那样只找到第一个匹配项。例22-1中示范了preg_match()、preg_match_all()和preg_replace()函数的使用。

例22-1：使用preg函数

```
<?php
if (preg_match('{<title>.+</title>}', $html)) {
    // 页面中包含标题元素
}

if (preg_match_all('/<li>/', $html, $matches)) {
    print 'Page has ' . count($matches[0]) . " list items\n";
}

// 变粗体为斜体
$italics = preg_replace('/(<\/?)b(>)/', '$1i$2', $bold);
?>
```

如果你想用一组特殊的字符来匹配字符串，可以通过将这些字符放到方括号中来创建一个字符类。字符类[aeiou]与a,e,i,o及u中的任何一个字符都匹配。也可以在方括号中使用范围来创建字符类。字符类[a-z]匹配所有的小写英文字母。而字符类[a-zA-Z0-9]则匹配所有数字和英文字母。字符类[a-zA-Z0-9_]匹配数字、英文字母和下划线。

到现在为止，我们所看到的模式，匹配的都是包含符合模式描述的任意字符串。换句话说，[a-z0-9]+不仅会匹配grapefruit和c3p0，同样也匹配grr!!!和*******p。所有这四个字符串都符合[a-z0-9]+的描述 —— “一个数字或小写英文字母出现一次或多次”。

通过锚定（anchoring）模式可以使其按照只匹配模式所描字符串的原则进行匹配。脱字符号（^）和美元符号（$）分别用于锚定模式的开始和结束。没有它们，匹配会发生在字符串的任何地方。所以，尽管[a-z0-9]+的含义是“一个数字或小写英文字母出现一或多次”，但^[a-z0-9]+则意味着“以一个或多个数字或者小写英文字母开头”，[a-z0-9]+$意味着“以一个或多个数字或者小写英文字母结末”，而^[a-z0-9]+$的含义就成了“仅包含一个或多个数字或者小写英文字母”。例22-2示范了一些字符类的用法。

例22-2：使用字符类和锚点进行匹配

```
<?php
$thisFileContents = file_get_contents(__FILE__);
// http://php.net/language.variables 给出了一个用于在
// php中验证变量名的正则表达式。模式开始的\$用于匹配$
//字母$
$matchCount = preg_match_all('/\$[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*/',
                             $thisFileContents, $matches);
print "Matches: $matchCount\n";
foreach ($matches[0] as $variableName) {
    print "$variableName\n";
}
?>
```

例22-2 输出使用的每个变量名称：

```
Matches: 8
$thisFileContents
$matchCount
$thisFileContents
$matches
$matchCount
$matches
$variableName
$variableName
```

有时候，通过定义目标来确定目标会更容易一些，因此可以采取这种思路。具体方法是编写一个不包含要匹配内容的字符类，然后在前面加上脱字符号（^）。如果脱字符号位于字符类的外部，它表示的是匹配一个字符串开始的模式；而如果脱字符号位于字符类的内部（即方括号的内部开头。译者注），它的含义就成了“匹配除了这个方括号中列出的字符以外的字符”。例如，字符类[^aeiou]匹配除了小写的英语元音字母之外的其他任何字符。

注意，[aeiou]的反义词并不是[bcdfghjklmnpqrstvwxyz]。字符类[^aeiou]同样会匹配大写的英语元音字母，如：AEIOU；会匹配数字，如：123；会匹配网址URL，如：http://www.cnpq.br/；甚至会匹配表情符号 :)。

竖线符号（|），也称为管道（pipe），用于分隔二选一的模式。例22-3使用了管道符号在一个文本块中查找多种可能的文件名。

例22-3：通过|进行匹配

```
<?php
$text = "The files are cuddly.gif, report.pdf, and cute.jpg.";
if (preg_match_all('/[a-zA-Z0-9]+\.(gif|jpe?g)/',$text,$matches)) {
    print "The image files are: " . implode(',',$matches[0]);
}
?>
```

例22-3的输出结果为：

```
The image files are: cuddly.gif,cute.jpg
```

以上我们所介绍的内容只是正则表达式的一些基本知识。在后面的技巧中我们还会介绍其他一些细节内容，此外在PHP的网站（*http://www.php.net/pcre*）上也有一些有关Perl兼容正则表达式的非常有用的信息。其中，页面最后的“Pattern Modifiers”和“Pattern Syntax”链接到的页面中包含了特别详尽和丰富的内容。

22.1 从ereg转换到preg

问题

你想从使用ereg函数转换到使用preg函数。

方案

首先，必须在模式中添加分隔符：

```
preg_match('/pattern/', 'string')
```

对于eregi()所完成的对大小写不敏感的匹配，使用/i修饰符来代替：

```
preg_match('/pattern/i', 'string');
```

当使用整数而不是字符串作为模式或者替换值时，将数字转换为十六进制并使用转义序列来指定它：

```
$hex = dechex($number);
preg_match("/\x$hex/", 'string');
```

讨论

ereg和preg两者之间有几个重要的区别。首先，在使用preg函数时 —— 和在Perl中一样 —— 模式不仅仅要包含字符串模式，同时也需要定界符。因此，模式的形式为/pattern/（注1）。所以，

```
ereg('pattern', 'string');
```

就变成了：

```
preg_match('/pattern/', 'string');
```

在选择模式定界符时，一定不要把定界符放在正则表达式的模式中，否则会导致该模式提前结束。如果没有办法避免这个问题，就必须使用反斜杠对位于模式中的定界符进行转义。可以调用addcslashes()来代替手工添加反斜杠。

注1：像{pattern}，<pattern>，|pattern|，#pattern#，或者任何你喜欢的分隔符。如果你使用了一个成对的分隔符像(，<，[，或者{作为起始分隔符，那么PHP会希望将相应的),>，]或者}作为结束分隔符。如果你使用另外的字符作为起始分隔符，那么PHP希望用相同的字母作为结束分隔符。

例如，如果你使用/作为定界符：

```
$ereg_pattern = '<b>.+</b>';
$preg_pattern = addcslashes($ereg_pattern, '/');
```

那么$preg_pattern的值现在就是 <b>.+<\/b>。

preg函数中也没有一连串的大小写不敏感的函数。而是使用不区分大小写的修饰符。在转换时，需要将

```
eregi('pattern', 'string');
```

改为：

```
preg_match('/pattern/i', 'string');
```

即在结束定界符的后面加i。

最后，还有一个不太清晰的区别。如果你在ereg_replace()中使用数字（不是字符串）作为模式或替换值，则会被假定你引用的是该字符的ASCII编码值。因此，由于9是制表符（即\t）的ASCII编码值，下面的代码会在每一行的开头插入一个制表符。

```
$tab = 9;
$replaced = ereg_replace('^', $tab, $string);
```

下面是如何转换结尾换行符的例子。

```
$converted = ereg_replace(10, 12, $text);
```

如果要在ereg函数中消除这种默认的作用，可以使用数字字符代替数字值：

```
$tab = '9';
```

另一方面，preg_replace()则把数字9看成是数字9，而非制表符的替代符号。如果要对这些字符编码进行转换并在preg_replace()中使用，需要先将相应的编码值转换为十六进制并在前面加上前缀\x。例如9要转换为\x9或\x09，而12要转换成\x0c。不过，也可以用\t、\r和\n来分别表示制表符、回车符和换行符。

参见

ereg()函数的文档（*http://www.php.net/ereg*）；preg_match()函数的文档（*http://www.php.net/preg-match*）和addcslashes()函数的文档（*http://www.php.net/addcslashes*）。

22.2 匹配单词

问题

你想从一个字符串中提取出所有单词。

方案

关键是仔细地定义单词的标准。在完成定义之后，就可以用相应的特定字符类型创建正则表达式了：

```
/\S+/                    // 不是空白符的任何字符
/[A-Z'-]+/i              // 所有大小写字母、撇号和连字符
```

讨论

一个看似简单的问题“什么是单词？”却令人意想不到的复杂。虽然Perl兼容的正则表达式中内置了单词字符类型 —— 用\w表示，但关键是真正理解在PHP中是如何定义一个单词的。否则，得到的结果很难令人满意。

正常情况下，由于直接取自Perl关于单词的定义，因此\w包括所有字母、数字和下划线，也就是说，a_z也是一个单词，但电子邮件地址php@example.com却不是。

在本技巧中，我们只涉及英语单词，而不讨论其他语言中单词的概念。因为Perl兼容的正则表达式会根据当前地区来定义它的设置，所以通过修改地区能够改变字母的定义，相应地也就重新定义了单词的含义。

为了解决这一问题，你可能会想到在字符类中明确地列举属于你认为是单词的那些字符。如果需要这样做的话，可以使用\xdd来添加非标准的字符，其中dd是一个字符的十六进制编码。

参见

技巧19.2中有关设置地区的信息和技巧19.13中有关PCRE的regex函数如何使用UTF-8编码字符的讨论。

22.3 查找第n个匹配项

问题

你想查找与模式匹配的第*n*个匹配项，而不是第一个匹配项。

方案

使用preg_match_all()将所有匹配项提取到一个数组中，然后再按照自己的愿望挑选出特定的匹配项，如例22-4所示。

例22-4：查找第n个匹配项

```
<?php
$todo = "1. Get Dressed 2. Eat Jelly 3. Squash every week into a day";

preg_match_all("/\d\. ([^\d]+)/", $todo, $matches);

print "The second item on the todo list is: ";
// $matches[1] 是包含由([^\d]+)捕获的每个子字符串的数组
print $matches[1][1];

print "The entire todo list is: ";
foreach($matches[1] as $match) {
    print "$match\n";
}
?>
```

讨论

由于preg_match()函数会在它找到第一个匹配项后停止继续匹配，所以如果要查询其他匹配项则需要使用preg_match_all()。preg_match_all()函数返回它找到的完整模式匹配的项目个数。如果没有找到匹配，则返回0。如果遇有错误发生，如模式中存在语法问题，则返回false。

preg_match_all()的第三个参数是一个用于保存与模式匹配的各个子字符串信息的数组。对于例22-4而言，$matches[0]中保存的是与模式/\d\. ([^\d]+)/匹配的$todo部分：①Get Dressed,②Eat Jelly，和③Squash every week into a day。

$matches数组中第二个元素是保存着与每个括入圆括号中的子模式匹配的文本数组。例22-4中的模式只有一个子模式 ([^\d]+)，所以$matches[1]保存的就是一个与该子模式匹配的字符串数组： Get Dressed, Eat Jelly，和Squash every week into a day。

如果还存在第二个子模式，那么与第二个子模式匹配的子字符串将会保存在$matches[2]中，第三个子模式的匹配项会保存在$matches[3]中，依此类推。

preg_matche_all()除了能够返回同时包含完整匹配然后是子匹配的单独数组外，也能按匹配项返回多个带有子匹配项的数组。要触发这一功能，可以将PREG_SET_ORDER作为第四个参数传递。这一功能在你构建了多个用于捕获的子模式，而且希望能够一次一组地迭代每个子模式时是相当有用的，具体过程如例22-5所示。

例22-5：对子模式进行分组

```
<?php
$todo = "
first=Get Dressed
next=Eat Jelly
last=Squash every week into a day
";

preg_match_all("/([a-zA-Z]+)=(.*)/", $todo, $matches, PREG_SET_ORDER);

foreach ($matches as $match) {
    print "The {$match[1]} action is {$match[2]} \n";
}
?>
```

例22-5的输出结果为：

```
The first action is Get Dressed
The next action is Eat Jelly
The last action is Squash every week into a day
```

通过传递PREG_SET_ORDER，位于foreach循环中的每个$match的值分别包含着对应的子模式匹配项：$match[0]中保存着匹配的整个字符串，$match[1]中保存的是“=”以前的字符串，而$match[2]中保存的是“=”后面的字符串。

参见

preg_match_all()函数的文档（*http://www.php.net/preg-match-all*）。

22.4 选择进行贪婪或非贪婪匹配

问题

你希望模式尽可能地匹配最小的字符串而不是最大的字符串。

方案

在表示数量的元字符后面添加?来修改模式中这一部分的匹配策略，如例22-6所示。

例22-6：使数量元字符匹配尽量少的字符

```
<?php
// 查找所有 <em>emphasized</em> 部分
preg_match_all('@<em>.+?</em>@', $html, $matches);
?>
```

或者以模式修饰符U结尾将所有表示数量的元字符的匹配策略由贪婪（“匹配尽可能多的字符”）转换为非贪婪（“匹配尽可能少的字符”）。例22-7中的代码与例22-6中的代码的功能是一样的。

例22-7：使数量元字符匹配尽量少的字符

```
<?php
// 查找所有 <em>emphasized</em> 部分
preg_match_all('@<em>.+</em>@U', $html, $matches);
?>
```

讨论

在默认情况下，PHP中的所有正则表达式数量元字符串都是贪婪的。例如，模式.+，它匹配“, one or more characters, ”，匹配字符串I simply love your work。贪婪的正则表达式只能找到一个匹配项，这是因为它在匹配了开始的标签后，.+会使它尽可能多地匹配后续的字符，直到遇到最后一个闭标签才停止。也就是说，.+模式会匹配love your work。

而另一方面，非贪婪的正则表达式则会找到两个匹配项。第一个标签同前面一样会被匹配，但之后的.+会尽可能早地停止匹配，因此只匹配love。第二个匹配随即开始，在第二轮中.+又匹配了work。

例22-8所示为贪婪和非贪婪模式的用法。

例22-8：贪婪匹配与非贪婪匹配

```
<?php
$html = 'I simply <em>love</em> your <em>work</em>';
// 贪婪
$matchCount = preg_match_all('@<em>.+</em>@', $html, $matches);
print "Greedy count: " . $matchCount . "\n";
// 非贪婪
$matchCount = preg_match_all('@<em>.+?</em>@', $html, $matches);
print "First non-greedy count: " . $matchCount . "\n";
// 非贪婪
$matchCount = preg_match_all('@<em>.+</em>@U', $html, $matches);
```

```
print "Second non-greedy count: " . $matchCount . "\n";
?>
```

例22-8的输出结果为：

```
Greedy count: 1
First non-greedy count: 2
Second non-greedy count: 2
```

贪婪匹配也可以称为最大化匹配，非贪婪匹配也可以称为最小化匹配，因为这两种策略要么是尽可能地匹配最多的字符，要么就是尽可能地匹配最少的字符。

ereg()和ereg_replace()函数始终都是贪婪的。因此，能够在贪婪和非贪婪策略之中进行选择是选择PCRE函数的另外一个原因。

虽然非贪婪匹配对于简单的HTML解析用处很大，但如果你的标记代码不是100%都合法而且还存在一些无法配对的之类的标签（注2），那么采取这种匹配策略就会出现问题。如果你的目标只是从一个文本块中删除全部（或部分）HTML标签，最好还是不要选用正则表达式，而是使用内置的strip_tags()函数，这个函数不仅速度快而且效果好。你可以参考技巧13.14中更多的相关内容。

最后，虽然非贪婪匹配的思想源于Perl，但U修饰符并不与Perl兼容，它只在PHP的Perl兼容正则表达式中有效。而且，它不仅会将所有数量元字符由贪婪转换为非贪婪，而且也能进行相反的转换。也就是说，要想在一个末尾带/U的模式中得到贪婪的数量元字符，只须在后面再加上一个?即可。同样的方式通常也会把一个贪婪的数量元字符转换为非贪婪的数量元字符。

参见

技巧22.6中有关在HTML标签中捕获文本的更多内容。技巧13.14中有关剥掉HTML标签的讨论。preg_match_all()函数的文档（*http://www.php.net/preg-match-all*）。

22.5 找到一个文件中与模式匹配的所有行

问题

你想找到一个文件中与既定模式匹配的所有行。

注2：即使对于合法的HTML标记代码同样有可能遇到麻烦。例如，在注释中出现了加粗标签。虽然HTML解析器会忽略注释中的标签，但正则表达式模式却不会。

方案

使用preg_grep()将该文件读入到一个数组中。

讨论

有两种方式可以完成这一任务。例22-9速度快，但占用内存较多。它使用了file()函数将文件中的每一行都放到了一个数组中，并通过preg_grep()过滤掉了非匹配的行。

例22-9：快速查询匹配特定模式的行

```
$pattern = "/\bo'reilly\b/i"; // 只限 O'Reilly 的书
$ora_books = preg_grep($pattern, file('/path/to/your/file.txt'));
```

例22-10速度要慢一些，但占用内存较少。它每次读取文件中的一行，并在读取后通过preg_match()来检查该行是否与模式匹配。

例22-10：有效地查询与特定模式匹配的行

```
$fh = fopen('/path/to/your/file.txt', 'r') or die($php_errormsg);
while (!feof($fh)) {
    $line = fgets($fh);
    if (preg_match($pattern, $line)) { $ora_books[ ] = $line; }
}
fclose($fh);
```

由于例22-9中的代码一次读取了所有文件的内容，因而大约会比例22-10的速度快三倍，但例22-10通过以分行方式解析文件做到了占用更少的内存空间。一定要记住的是，由于这两种方法都是针对文本中的个别的行进行操作，所以就无法使用跨行匹配文件的模式。

参见

技巧23.5中有关将文件读取为字符串的介绍。preg_grep()函数的文档（*http://www.php.net/preg-grep*）。

22.6 在HTML标签中捕获文本

问题

你想在HTML标签中捕获文本。例如，你想在一个HTML文档中找到所有标题标签。

方案

将HTML文件读入到一个字符串中，并在模式中使用非贪婪匹配策略，如例22-11所示。

例22-11：捕获HTML标题

```
<?php
$html = file_get_contents('example.html');
preg_match_all('@<h([1-6])>(.+?)</h\1>@is', $html, $matches);
foreach ($matches[2] as $text) {
    print "Heading: $text \n";
}
?>
```

讨论

通过简单的正则表达式很难对HTML实现可靠的解析。这正是使用XHTML的好处之一，因为显而易见XHTML是容易验证和解析的。

例如，例22-11中模式无法处理位于标题标签中的属性，它只能找到匹配的标题。所以对于<h1>Dr. Strangelove</h1>是没问题的，因为文本包含在了<h1></h1>中。但对于<h2>How I Learned to Stop Worrying and Love the Bomb</h3>则不行，因为开标签是<h2>，而闭标签却不是。

这种技术也可以用于在良好结构的和标签中查找文本，如例22-12所示。

例22-12：从HTML标签中提取文本

```
<?php
$html = file_get_contents('example.html');
preg_match_all('@<(strong|em)>(.+?)</\1>@is', $html, $matches);
foreach ($matches[2] as $text) {
    print "Text: $text \n";
}
?>
```

然而，例22-12对于嵌套的标题元素是无能为力的。如果*example.html*中包含Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb，那么例22-12则不会将内部的文本作为一个独立的项目来提取。

这在例22-11中是可以实现的。因为标题是块级元素，它们之间相互嵌套是不合法的。但是，对于行内（inline）元素，中嵌套则是合法的。

在解析少量的HTML代码时使用正则表达式在一定程度上是有用的，尤其是在HTML代码的结构很规范（或者是你自己编写）的情况下。对于更一般化也比较可靠的HTML解析，可以使用tidy扩展，该扩展为流行的HTML清理库libtidy提供了一个接口。当tidy对

HTML进行清理后，可以使用它提供的方法访问文档中的各个部分。或者，如果你通过tidy把HTML转换成了XHTML，就可以使用SimpleXML或DOM扩展的所有XML文档操作能力来对你的HTML文档随意进行分割组合。

参见

技巧13.9中有关在网页中添加标记的信息和技巧13.11中有关从HTML文档中提取链接的介绍。preg_match()函数的文档（*http://www.php.net/preg-match*）和tidy的文档（*http://www.php.net/tidy*）。

22.7 阻止对子模式匹配文本的捕获

问题

你想在模式中使用圆括号来分组，但不想让与圆括号中的子模式匹配的文本出现在包含匹配文本的数组中。

方案

将?:放在紧挨着开始圆括号的后面，如例22-13所示。

例22-13：阻止文本捕获

```
<?php
$html = '<link rel="icon" href="http://www.example.com/icon.gif"/>
<link rel="prev" href="http://www.example.com/prev.xml"/>
<link rel="next" href="http://www.example.com/next.xml"/>';

preg_match_all('/rel="(prev|next)" href="([^"]*?)"/', $html, $bothMatches);
preg_match_all('/rel="(?:prev|next)" href="([^"]*?)"/', $html, $linkMatches);

print '$bothMatches is: '; var_dump($bothMatches);
print '$linkMatches is: '; var_dump($linkMatches);

?>
```

在例22-13中，$bothMatches中包含rel和href属性的值。而$linkMatches中则只包含href属性值。该代码会输出以下结果：

```
$bothMatches is: array(3) {
  [0]=>
  array(2) {
    [0]=>
    string(49) "rel="prev" href="http://www.example.com/prev.xml""
```

```
    [1]=>
    string(49) "rel="next" href="http://www.example.com/next.xml""
  }
  [1]=>
  array(2) {
    [0]=>
    string(4) "prev"
    [1]=>
    string(4) "next"
  }
  [2]=>
  array(2) {
    [0]=>
    string(31) "http://www.example.com/prev.xml"
    [1]=>
    string(31) "http://www.example.com/next.xml"
  }
}
$linkMatches is: array(2) {
  [0]=>
  array(2) {
    [0]=>
    string(49) "rel="prev" href="http://www.example.com/prev.xml""
    [1]=>
    string(49) "rel="next" href="http://www.example.com/next.xml""
  }
  [1]=>
  array(2) {
    [0]=>
    string(31) "http://www.example.com/prev.xml"
    [1]=>
    string(31) "http://www.example.com/next.xml"
  }
}
```

讨论

在子模式可选的情况下使用阻止捕获非常有用。因为可选的子模式可能不会出现在包含捕获文本的数组中，所以它可能会改变其捕获的文本块数。这通常会使根据给定的索引来引用特定的匹配文本块变得不好操作。但将可选的子模式设置为不捕获可以解决这个问题。例22-14所示为这一差别。

例22-14：不捕获可选子模式

```
<?php
$html = '<link rel="icon" href="http://www.example.com/icon.gif"/>
<link rel="prev" title="Previous" href="http://www.example.com/prev.xml"/>
<link rel="next" href="http://www.example.com/next.xml"/>';

preg_match_all('/rel="(?:prev|next)"(?: title="[^"]+?")? href=
"([^"]*?)"/', $html, $linkMatches);
```

```
print '$bothMatches is: '; var_dump($linkMatches);
?>
```

参见

PCRE Pattern Syntax的文档（*http://php.net/reference.pcre.pattern.syntax*）。

22.8 转义正则表达式中的特殊字符

问题

你想让一个正则表达式中的*或+直接匹配其自身，而不是作为元字符出现。这种情况适用于你让用户键入要在正则表达式中使用的搜索字符串。

方案

使用preg_quote()来对Perl兼容的正则表达式元字符进行转义处理。

```
<?php
$pattern = preg_quote('The Education of H*Y*M*A*N K*A*P*L*A*N').':(\d+)';
if (preg_match("/$pattern/",$book_rank,$matches)) {
    print "Leo Rosten's book ranked: ".$matches[1];
}
?>
```

使用quotemeta()来转义POSIX元字符。

```
$pattern = quotemeta('M*A*S*H').':[0-9]+';
if (ereg($pattern,$tv_show_rank,$matches)) {
    print 'Radar, Hot Lips, and the gang ranked: '.$matches[1];
}
```

讨论

以下是preg_quote()转义的元字符：

```
. \ + * ? ^ $ [ ] () { } < > = ! | :
```

下面是quotemeta()转义的元字符：

```
. \ + * ? ^ $ [ ] ()
```

这两个函数都是用反斜杠来转义元字符。

quotemeta()函数不能转义全部POSIX元字符。字符{，}和|也都是有效的元字符，但不会被转义。这也是另一个使用preg_match()而不使用ereg()的原因。

也可以给preg_quote()传递要转义的字符作为其第二个参数。而将你选择的模式定界符（一般是/）作为这个参数非常合适，这样就能将它进行转义。这在你将用户输入合并为正则表达式时是非常重要的一环。下面的代码会从一个表单中读取$_GET['search_term']的值并在$s中搜索以$_GET['search_term']开头的词。

```
$search_term = preg_quote($_GET['search_term'],'/');
if (preg_match("/\b$search_term/i",$s)) {
   print 'match!';
}
```

当一位《私家侦探马格南》（译注1）的爱好者在搜索框中键入了t.c（译注2）时，通过使用preg_quote()可以保证正则表达式能够得到适当地解释。而如果不使用preg_quote()，结果可能会匹配tic、tucker以及任何第一个字母是t和第三个字母是c的单词。而将定界符通过preg_quote()来转义（作为其第二个参数）也能保证用户输入的正斜杠，如CP/M，被正确地处理。

参见

preg_quote()函数的文档（*http://www.php.net/preg-quote*）和quotemeta()函数的文档（*http://www.php.net/quotemeta*）。

22.9 通过模式分隔符读取记录

问题

你想从一个文件中读取多条记录，其中每条记录都是以正则表达式能够匹配的分隔符分隔的。

方案

将整个文件中的内容读取到一个字符串中，然后使用正则表达式进行分割：

```
$filename = '/path/to/your/file.txt';
```

译注1：1982年发行的美国电视连续剧。

译注2：剧中人物，Roger E. Mosley饰。

```
$fh = fopen($filename, 'r') or die($php_errormsg);
$contents = fread($fh, filesize($filename));
fclose($fh);

$records = preg_split('/[0-9]+\) /', $contents);
```

讨论

这会将一个编号列表打散，并将单独的列表项目保存到相应的数组元素中。所以，如果你有一个类似如下的列表：

```
1) Gödel
2) Escher
3) Bach
```

最终会得到一个包含四个元素的数组，其中的第一个元素为空。这是因为preg_split()假设定界符位于项目之间，但这里的例子中，数字位于项目之前：

```
Array
(
    [0] =>
    [1] => Gödel
    [2] => Escher
    [3] => Bach
)
```

从某种角度来看，这可能也是一种特征，而非bug，因为第*n*个元素保存第*n*个项目。不过，为了简化数组，你可以删去第一个元素：

```
$records = preg_split('/[0-9]+\) /', $contents);
array_shift($records);
```

另一个你可能会想要的修正是从元素中剥离换行符并以空字符串取代之。

```
$records = preg_split('/[0-9]+\) /', str_replace("\n",'',$contents));
array_shift($records);
```

PHP不允许你将输入记录的分隔符改成换行符之外的其他字符，因此这项技术对于分开以字符串分隔的记录也是很有用的。不过，如果你是基于字符串而不是正则表达式来分离，则为了提高效率可以用explode()来代替preg_split()。

参见

技巧23.5中从一个文本中读取内容的讨论。技巧1.11中有关解析CSV文件的介绍。

22.10 在正则表达式中使用PHP函数

问题

你想通过一个PHP函数来处理匹配的文本。例如，你想对子模式捕获的HTML实体进行解码处理。

方案

使用preg_replace_callback()函数。不使用替换模式，而是给该函数传递一个回调函数。这个回调函数会接收到一个包含与子模式匹配的内容的数组，返回经过适当替换的字符串。例22-15对位于<code></code>之间的实体进行了解码。

例22-15：通过回调函数生成替换字符串

```
<?php
$html = 'The &lt;b&gt; tag makes text bold: <code>&lt;b&gt;bold&lt;/b&gt;</code>';
print preg_replace_callback('@<code>(.*?)</code>@','decode', $html);

// $matches[0] 整个匹配的字符串
// $matches[1] 是第一个子模式捕获的字符串
function decode($matches) {
    return html_entity_decode($matches[1]);
}
?>
```

例22-15的输出结果为：

```
The &lt;b&gt; tag makes text bold: <b>bold</b>
```

讨论

preg_replace_callback()的第二个参数指定的是在计算替换字符串时调用的函数。与PHP在任何地方使用的“回调”伪类型（pseudotype）都一样，这个参数可以是一个字符串也可以是一个数组，用字符串指定的是函数名。如果想使用对象技巧的方法作为回调，则需要传递一个数组，数组的第一个元素保存着对象技巧，第二个元素保存着一个包含方法名的字符串。如果想使用一个静态类方法作为回调，要传递的数组中也包含两个元素：类名和方法名。

回调函数会接收到一个参数 —— 包含匹配内容的数组。该数组的元素0中保存的总是与整个模式匹配的文本。如果传递给preg_replace_callback()的模式中包含括入圆括号中

的子模式，那么与这些子模式匹配的文本就分别保存在数组的其他元素中。匹配数组是通过数字键来访问的，即使模式中包含命名的子模式也是如此。

PHP的preg_replace_callback()主页中建议使用create_function()创建匿名函数用作回调。虽然这可能很方便，但如果对create_function()的调用与对preg_replace_callback的调用内联（inline）并且位于一个循环当中，则会强烈地消耗内存。如果你想在preg_replace_callback()中使用匿名函数，可以调用create_function()一次，将匿名的回调函数保存到一个变量中。然后，将该变量作为回调函数传递给preg_replace_callback()。例22-16使用了匿名函数来对文件中的每一行实现类似例22-15中的转换。

例22-16：通过匿名函数生成替换字符串

```
<?php
$callbackFunction = create_function('$matches',
                          'return html_entity_decode($matches[1]);');
$fp = fopen('html-to-decode.html','r');
while (! feof($fp)) {
    $line = fgets($fp);
    print preg_replace_callback('@<code>(.*?)</code>@',$callbackFunction, $line);
}
fclose($fp);
?>
```

preg_replace_callback()的替换方案是使用模式修饰符e。使用了该修饰符，会使得替换字符串被作为PHP代码来求值。虽然有此方案，但由于下面将要解释的与回溯引用有关的原因，我们还是建议你使用preg_replace_callback()。

例22-17使用模式修饰符来实现与例22-15中相同的实体解码处理。

例22-17：对匹配文本中的实体进行解码

```
<?php
$html = 'The &lt;b&gt; tag makes text bold: <code>&lt;b&gt;bold&lt;/b&gt;</code>';
print preg_replace('@<code>(.*?)</code>@e',"html_entity_decode('$1')", $html);
?>
```

当你使用e修饰符并在替换字符串中包含了回溯引用时，问题可能就有些复杂了。这就引出了需要重视（在某些情况下，可能是需要解决）的多级转义的问题。

第一级转义是因为你在任何时候构造一个字符串时都会发生的PHP的常规行为（注意：在技巧22-17中，因为$1不是有效的常规变量名，所以即使整个替换字符串由双引号作为定界符，其中的$也不需要进行转义）。

第二级转义是有关在替换字符串中如何对回溯引用替换部分进行定界的问题。在例22-17中，html_entity_decode('$1')会变成html_entity_decode('<code><b&g

t;bold</code>')。也就是说，会以一个参数——单引号字符串——调用html_entity_decode()。

不论单引号还是双引号，在捕获的匹配文本中都是以反斜杠进行了转义的。而当回溯引用替换部分自身包含单、双引号时就需要认真对待了。我们以例22-18为例。

例22-18：回溯引用替换部分的引号转义

```
<?php
$html = "<code>&lt;b&gt; It's bold &lt;/b&gt;</code>";
print preg_replace('@<code>(.*?)</code>@e',"html_entity_decode('$1')", $html);
print "\n";

$html = '<code>&lt;i&gt; "This" is italic. &lt;/i&gt;</code>';
print preg_replace('@<code>(.*?)</code>@e',"html_entity_decode('$1')", $html);
print "\n";
?>
```

例22-18的输出结果为：

```
<b> It's bold </b>
<i> \"This\" is italic. </i>
```

不知道为什么，第二行中出现了两个反斜杠。实际上，这是因为e修饰符起作用的结果。我们在前面提到过，不论单引号还是双引号，在捕获的匹配文本中都以反斜杠进行了转义。也就是说，在例22-18中对preg_replace()的第一次调用中，用于计算替换字符串的是：

```
html_entity_decode('<code>&lt;b&gt; It\'s bold &lt;/b&gt;</code>)
```

html_entity_decode()接收到了一个带单引号的字符串，该字符串中还包含一个用反斜杠转义的单引号。一切正常，因为'It\'s'实际上正是It's。然而，在第二次调用preg_replace()时，问题就出现了。用于计算的字符串这次变成了：html_entity_decode ('<code><i> \"This\" is italic. </i></code>')，即在一个单引号引用的字符串中，位于双引号之前的反斜杠并不代表直接量反斜杠，而是两个字符序列\"。

为了解决这个问题，需要在代码中对执行替换计算的部分使用str_replace()来将\"替换成"（不要使用stripslashes()，因为这也会删掉其他字符串前面的反斜杠，而这不是我们所希望的）。例22-19使用了一个实现替换的函数来封装html_entity_decode()。

例22-19：修复在回溯引用替换部分中的引号转义问题

```
<?php
$html = "<code>&lt;b&gt; It's bold &lt;/b&gt;</code>";
print preg_replace('@<code>(.*?)</code>@e',"preg_html_entity_decode('$1')", $ht
ml);
```

```
print "\n";

$html = '<code>&lt;i&gt; "This" is italic. &lt;/i&gt;</code>';
print preg_replace('@<code>(.*?)</code>@e',"preg_html_entity_decode('$1')", $html);
print "\n";

function preg_html_entity_decode($s) {
    $s = str_replace('\\"','"', $s);
    return html_entity_decode($s);
}
?>
```

例22-19通过使用preg_html_entity_decode()函数保证了结果的正确输出：

```
<b> It's bold </b>
<i> "This" is italic. </i>
```

最后一个有关转义和模式修饰符e需要注意的问题是，在你的替换计算表达式内部，要切记使用单引号（而不是双引号）来对任何包含回溯引用值的字符串进行定界。换句话说，要用preg_html_entity_decode('$1')，而不能用preg_html_entity_decode("$1")。如果回溯引用值中包含着一些类似有效变量名称的字符串时，使用双引号就会出现问题，如例22-20所示。

例22-20：变量名和双引号字符串

```
<?php
$text = '<code>if ($temperature &lt; 12) { fever(); }</code>';
print "Good: \n";
print preg_replace('@<code>(.*?)</code>@e',"preg_html_entity_decode('$1')", $text);
print "\n Bad: \n";
print preg_replace('@<code>(.*?)</code>@e','preg_html_entity_decode("$1")'  , $text);

function preg_html_entity_decode($s) {
    $s = str_replace('\\"','"', $s);
    return html_entity_decode($s);
}
?>
```

例22-20的输出结果为：

```
Good:
if ($temperature < 12) { fever(); }
 Bad:

Notice: Undefined variable: temperature in example.php(6) : regexp code on line
 1
if ( < 12) { fever(); }
```

通过适当地引用，第一个preg_replace()的结果如我们所料：对$text进行的唯一修改是用<替换<。而带有双引号的$1的第二个preg_replace()，则被迫中断。因为PHP

解析器认为传递给preg_html_entity_decode()的字符串是"if ($temperature < 12) { fever();}"，由于带双引号，所以PHP解释器会尝试将$temperature替换为相应的变量值，而实际上，根本不存在那个变量。

因此，“在preg_replace()中使用修饰符e”这个例子的寓意是双重的：对于以反斜杠转义的双引号字符串和在代码表达式中使用单引号定界字符串避免意外发生变量插值现象是正确的；而正是聪明的引用和插值行为使得preg_replace_callback()成为一种更友好的选择。

参见

preg_replace_callback()函数的文档（*http://www.php.net/preg_replace_callback*）；preg_replace()函数的文档（*http://www.php.net/preg_replace*）；create_function()函数的文档（*http://www.php.net/create_function*）和回调伪类型的文档（*http://www.php.net/language.pseudo-types#language.types.callback*）。

第23章

文件

23.0 概述

Web应用程序中的输入输出流一般都发生在浏览器、服务器和数据库之间。但是，在许多情况下也会涉及到文件的处理。当对从远程网页获得的信息进行本地处理时、在没有数据库的情况下存储数据时以及为与其他程序共享保存的信息时，都要用到文件。而且，当PHP不仅仅作为一种网页处理工具而出现时，文件输入/输出（I/O）函数变得更有用武之地。

PHP中文件输入/输出的接口与C类似，但却没有那么复杂。用于标识读取和写入文件的基础单元是文件句柄。该句柄用以识别指向特定文件的连接，而且对文件的操作也要通过它来完成。本章主要包括如何在PHP中打开、关闭和操纵文件，以及在打开文件后能够对文件做哪些处理等内容。第24章中则包括了处理目录和文件元数据（如权限）的相关内容。

例23-1中的代码打开了*/tmp/cookie-data*并将一个特殊的cookie写入到文件中。

例23-1：将数据写入文件

```
<?php
$fh = fopen('/tmp/cookie-data','w')      or die("can't open file");
if (-1 == fwrite($fh,$_COOKIE['flavor'])) { die("can't write data"); }
fclose($fh)                              or die("can't close file");
?>
```

其中，fopen()函数尝试在打开文件成功后，返回一个文件句柄。如果该函数无法打开文件（例如，权限不符），则返回false并生成一个E_WARNIGN-type错误。技巧23.1～23.3介绍了打开文件的几种方式。

在例23-1中，fwrite()把flavor cookie的值写入到了文件句柄中，并返回所写入的字节数。如果该函数无法向文件中写入字符串（例如，磁盘空间不足），则返回-1。

最后，fclose()关闭文件句柄。虽然每个请求最后都会自动关闭文件，但明确地关闭打开的所有文件仍然是一个值得提倡的好习惯。因为这样能够保证在命令行环境中调用代码时不会出现问题，并能及时地释放系统资源。而且，你还能对fclose()函数返回的代码进行检查。由于缓冲的数据在调用fclose()之前可能不会真正写入到磁盘上，所以有时候fclose()也会返回“磁盘已满”的错误。

与其他过程一样，PHP必须拥有适当的权限才能对一个文件进行读写操作。这个问题在命令行环境中通常很直观，但当在Web服务器中运行脚本时却有可能会导致混淆。你的Web服务器（连同PHP脚本）可能会为一个致力于Web服务的特殊用户（或者，也可能是一个普通用户）而运行。为了保证安全，该用户通常对它所能访问的文件具有受限权限。如果你的脚本在对一个文件进行操作时遇到了问题，要确认一下Web服务器的用户或组 —— 不是你的 —— 拥有执行该文件操作的权限。有的Web服务设置可能会以你的身份来运行脚本，在那种情形下，你应该保证你的脚本不会意外地对不属于你网站的个人文件进行读写操作。

因为多数文件处理函数在遇到错误时只返回false，所以要想了解错误的更多信息就必须做一些额外的工作。在track_errors配置指令设为on的情况下，每个错误的信息都会被放到全局变量$php_errormsg中。如果将这个变量包含到你的错误输出代码中会有助于排除故障，如例23-2所示。

例23-2：使用文件相关的错误信息

```
<?php
$fh = fopen('/tmp/cookie-data','w')      or die("can't open: $php_errormsg");
if (-1 == fwrite($fh,$_COOKIE['flavor'])) { die("can't write: $php_errormsg") };
fclose($fh)                              or die("can't close: $php_errormsg");
?>
```

如果你对*/tmp/cookie-data*没有写入的权限，例23-2会以以下错误输出终止：

```
can't open: fopen("/tmp/cookie-data", "w") - Permission denied
```

Windows和Unix对待文件的方式不同。因此，要保证文件存取代码在Unix和Windows系统中都能正常有效，必须当心行分隔符并保证路径名正确。

在Windows中，一个行分隔符包含两个字符：ASCII 13（回车符）后跟ASCII 10（移行或换行符）。在Unix系统中，则只需要一个ASCII 10即可。这些打字时代起的名字解释了为什么当你输出一个带有Unix分隔符的文件时会得到“阶梯状”的文本。想象一下如果将这些字符的名字作为命令发送给一台打字机或者单字符打印机的压纸滚筒会怎样。

回车会派滚筒返回到一行的开始位置，而换行则使滚筒上的纸上提一行（译注1）。当一台未经配置的打印机碰到带有Unix分隔符的文件时，会忠实地执行指令并在每一行的末尾进行换行。虽然前进了一行但却没有把水平打印位置移动到左页边距处。所以，下面一行会从前一行结束的位置下方开始，就形成了“阶梯状”文本。

PHP函数使用换行符作为一行的分隔符（如fgets()），因为无论在Windows还是Unix平台中行的末尾都有换行符，所以这些函数对这两大平台都是适用的。

如果要删除行分隔符，可以使用PHP函数rtrim()，如例23-3所示。

例23-3：清理尾部空白

```
<?php
$fh = fopen('/tmp/lines-of-data.txt','r') or die($php_errormsg);
while($s = fgets($fh)) {
    $s = rtrim($s);
    // 对 $s 进行某些处理
}
fclose($fh)                                  or die($php_errormsg);
?>
```

这个函数会删除每一行末尾的空白符，包括ASCII 13和 ASCII 10（也包括制表符和空格符）。如果你想保留行尾的某些空白符，但仍然想删除回车符和换行符，可以给rtrim()传递一个应该删除字符的字符串，该字符串以外的其他字符则会保留，如例23-4所示。

例23-4：清理行尾字符

```
<?php
$fh = fopen('/tmp/lines-of-data.txt','r') or die($php_errormsg);
while($s = fgets($fh)) {
    $s = rtrim($s, "\r\n");
    // 对 $s 进行某些处理
}
fclose($fh)                                  or die($php_errormsg);
?>
```

对于路径名中的目录分隔符，Unix和Windows也不一致。Unix使用斜杠（/），而Windows使用反斜杠（\）。不过，由于Windows版的PHP也能够识别作为目录分隔符的/，所以这个问题不算大。例如，例23-5同样能够成功地输出*C:\Alligator\Crocodile Menu.txt*中的内容。

例23-5：在Windows中使用正斜杠分隔目录

```
<?php
$fh = fopen('c:/alligator/crocodile menu.txt','r') or die($php_errormsg);
```

译注1：见“为什么键盘上‘回车键’叫回车呢？‘回车’一词的由来”*http://ks.cn.yahoo.com/question/?qid=1306072001951*。

```
while($s = fgets($fh)) {
    print $s;
}
fclose($fh)                                    or die($php_errormsg);
?>
```

例23-5同时也利用了Windows系统不区分文件名大小写的事实。但是，Unix系统对文件名大小写是敏感的。

某些换行符所导致的问题不仅仅存在于代码要读写的文件中，也同样存在于源代码文件中。如果是多个人共同开发一个项目，同样必须保证每个开发者都要在其编辑器中使用相同的换行符。

当你打开一个文件之后，PHP也提供了许多用于处理其中数据的工具。PHP与类似C语言输入/输出接口的原则一致，从一个文件中读取数据的两个最基本的函数是`fread()` —— 用于读取指定数目的字节，和`fgets()` —— 每次读取一行（根据指定的任意字节数）。例23-6中处理的行长度为256字节。

例23-6：从文件中读取数据行。

```
<?php
$fh = fopen('orders.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    $s = fgets($fh,256);
    process_order($s);
}
fclose($fh)              or die($php_errormsg);
?>
```

如果*orders.txt*中每行长度为300字节，第一次调用`fgets()`只返回前面的256字节。第二次调用`fgets()`会返回剩下的44字节并当遇到换行符时停止。第三次调用`fgets()`时又会移动到文件中的下一行。如果没有指定第二个参数，`fgets()`会读取一行末尾之前的数据（在PHP 4.2.0版之前，必须要指定行的长度。从PHP 4.2.0到4.3.0，未指长度时默认值为1024）。

许多基于文件内容的操作，如随机提取一行数据（见技巧23.8），如果将整个文件读取到一个字符串或数组中，那么从概念上会更简单一些（而且需要的代码也更少）。其中，`file_get_content()`会把全部文件内容读取到一个字符串中，而`file()`函数则会把文件的每一行放到一个数组中。当然，简化操作的另一方面意味着增大内存消耗。这对于将PHP作为一个服务器模块的情况是非常有害的。一般来说，当一个进程（比如嵌入了PHP代码的Web服务器进程）得到分配的内存（如PHP把全部文件内容都读取到一个字符串或者数组中）后，会在其终止时将内存返回给操作系统。这意味着，作为Apache服务器模块之一的PHP在调用`file_get_contents()`读取一个1MB大小的文件时，相应

的Apache进程会增加1MB内存的占用，而且直到该进程终止时才会释放。这样的过程只要重复数次，就会导致服务器效率降低。所以，即使你有一次性处理整个文件的再好理由，也要先搞清楚增加内存占用的后果再去实施。

技巧23.17到23.19涉及在PHP程序内部运行其他程序的话题。有些程序执行操作器（program execution operators）或函数提供了几种运行程序的方式，包括整批读取输出（backticks）、读取输出的最后一行（system()）。PHP可以使用管道（pipes）来运行程序，既能够传递输入，也能够读取其输出。由于管道是通过标准的输入/输出函数（fgets()和fread()）来执行读操作的，所以你可以决定自己想要的输入，同时也能在读取输入块的过程中执行其他任务。同样地，通过管道执行写操作是借助fputs()和fwrite()实现的，因此你能够以任意增量向程序中传递输入。

管道同常规文件一样也存在权限的问题。也就是说，PHP必须具有打开程序的执行权限。如果在打开一个管道时遇到了问题，尤其是当PHP作为一个特殊的Web服务器用户运行时，一定要确保该用户有权限执行打开管道的那个程序。

23.1 创建或打开一个本地文件

问题

你想打开一个本地文件，以读取其中的数据或向其中写入数据。

方案

使用fopen()，如例23-7所示。

例23-7：打开一个文件

```
<?php
$fh = fopen('file.txt','r') or die("can't open file.txt: $php_errormsg");
?>
```

讨论

fopen()的第一个参数是要打开的文件，第二个参数是打开文件的模式。模式用于指定对相应的文件能够执行什么操作（读和/或写）、文件打开后文件指针的位置（在文件开始处还是结尾处）、是否在打开以后缩短为零长度以及是否当文件不存在时创建相应的文件等等，如表23-1所示。

表23-1：fopen()打开文件的模式

模式	可读？	可写？	文件指针	截断？	创建？
r	是	否	开始	否	否
r+	是	是	开始	否	否
w	否	是	开始	是	是
w+	是	是	开始	是	是
a	否	是	结尾	否	是
a+	是	是	结尾	否	是
x	否	是	开始	否	是
x+	是	是	开始	否	是

如果文件已经存在，则x和x+模式会返回false并生成一个警告信息。这两个模式在PHP 4.3.2及以后版本中是有效的。

在非POSIX系统中，如Windows系统，当打开二进制文件时需要在模式中添加一个b，如例23-8所示，否则读写会因NUL（ASCII 0）字符而失败。

例23-8：可靠的读取二进制文件

```
<?php
$fh = fopen('c:/images/logo.gif','rb');
?>
```

即使Unix系统在模式中不加b的情况下能够很好地处理二进制文件，但加上b总是有益的。因为这样就能使你的代码获得更高的可移植性并能够同时在Unix和Windows系统中顺畅运行。

如果想进一步对文件进行操作，则需要把fopen()返回的文件句柄传递给其他输入/输出函数，如fgets()，fputs()和 fclose()。

也可以通过给fopen()传递第三个参数，来告诉它在*php.ini*中指定的include_path中搜索要打开的文件。例23-9所示为如何在include_path中搜索*file.inc*。

例23-9：打开include_path中的文件

```
<?php
$fh = fopen('file.inc','r',true) or die("can't open file.inc: $php_errormsg");
?>
```

参见

fopen()函数的文档（*http://www.php.net/fopen*）。

23.2 创建一个临时文件

问题

你需要一个临时文件来保存一些数据。

方案

如果文件只需在脚本运行期间存在，可以使用tmpfile()来创建，如例23-10所示。

例23-10：通过tmpfile()创建一个临时文件

```
<?php
$temp_fh = tmpfile();
// 向临时文件中写入一些数据
fputs($temp_fh,"The current time is ".strftime('%c'));
// 当脚本终止时该文件销毁
exit(1);
?>
```

如果这个文件需要维持更长时间，则需要通过tempnam()生成一个文件名，再用fopen()打开，如例23-11所示。

例23-11：通过tmpnam()创建一个临时文件

```
<?php
$tempfilename = tempnam('/tmp','data-');
$temp_fh = fopen($tempfilename,'w') or die($php_errormsg);
fputs($temp_fh,"The current time is ".strftime('%c'));
fclose($temp_fh) or die($php_errormsg);
?>
```

讨论

tmpfile()函数使用一个唯一的名称创建文件并返回一个文件句柄。当以该文件句柄调用fclose()或者脚本终止运行时，文件被删除。

作为替代方案，tempnam()会生成一个文件名。这个函数接受两个参数：第一个参数是目录，第二个参数是文件名的前缀。如果指定的目录不存在或不可写，tempnam()会使用系统临时文件夹 —— Unix中的TMPDIR环境变量或Windows中的TMP环境变量。例23-12所示为tempnam()如何生成文件。

例23-12：通过tempnam()生成一个文件名

```
<?php
$tempfilename = tempnam('/tmp','data-');
```

```
print "Temporary data will be stored in $tempfilename";
?>
```

例23-12的输出结果如下：

```
Temporary data will be stored in /tmp/data-GawVoL
```

由于PHP以这种方式生成临时文件，即使你的脚本从未明确的打开生成的文件，也会以tempnam()返回的文件名创建该文件并留空。这样就可以保证在调用tempnam()和调用fopen()期间不会再有其他程序以相同的文件名重新创建这个文件。

参见

tmpfile()函数的文档（*http://www.php.net/tmpfile*）和tempnam()函数的文档（*http://www.php.net/ tempnam*）。

23.3 打开远程文件

问题

你想打开一个可以通过HTTP或FTP存取的文件。

方案

给fopen()传递要打开文件的URL，如例23-13所示。

例23-13：打开一个远程文件

```
<?php
$fh = fopen('http://www.example.com/robots.txt','r') or die($php_errormsg);
?>
```

讨论

当fopen()接收到以*http://*开头的文件名时，它会以HTTP/1.0 GET请求的方式来取得给定的页面（尽管同时也传递了一个处理虚拟主机的Host:头部信息）。而且通过文件句柄只能访问响应的主体内容，而不能访问头部信息。通过HTTP协议打开文件时，只能读，不能写。

当fopen()接收到以*ftp://*开头的文件名时，它会返回一个通过被动FTP模式（passive-

mode）获得的指定文件的指针。你可以通过FTP读取或者写入文件，但不能对文件同时读写。

如果想通过fopen()打开一个需要用户名和密码的URL，可以像例23-14所示的那样在URL中嵌入相应的认证信息。

例23-14：通过FTP或HTTP及密码打开文件

```
<?php
$fh = fopen('ftp://username:password@ftp.example.com/pub/Index','r');
$fh = fopen('http://username:password@www.example.com/robots.txt','r');
?>
```

fopen()打开远程文件是通过一个称为流封装（stream wrapper）的PHP特性实现的。这一特性默认是启用的，但通过将*php.ini*中或Web服务器配置文件中的allow_url_fopen设置为off可以禁用它。所以，如果你不能通过fopen()打开远程文件，那么检查一下你的服务器配置可能会找到症结所在。

参见

技巧13.1到13.7中讨论的有关检索URL的内容；fopen()函数的文档（*http://www.php.net/fopen*）和流包装的文档（*http://www.php.net/features.remote-files*及*http://www.php.net/wrappers*）。

23.4 从标准输入中读取数据

问题

你想从命令行中的标准输入读取数据。例如，获得用户的键盘输入或者通过管道传送到PHP程序中的数据。

方案

使用fopen()打开*php://stdin*，如例23-15所示。

例23-15：从标准输出中读取数据

```
<?php
$fh = fopen('php://stdin','r') or die($php_errormsg);
while($s = fgets($fh)) {
    print "You typed: $s";
}
?>
```

讨论

技巧25.3讨论了在命令行环境中获得键盘输入数据的详细过程。在Web环境下，由于信息并不是通过标准输入获取的，所以从标准输入中读取数据并不十分有用。HTTP post和上传文件请求的主体内容是由PHP解析并保存在专用变量中的。对于非上传文件post请求的主体内容，我们在技巧8.7中介绍过，也可以通过php://input流读取到。

参见

技巧25.3在命令行环境中获取键盘输入内容的讨论；技巧8.7中有关读取POST请求主体内容的讨论；fopen()函数的文档（*http://www.php.net/fopen*）。

23.5 把文件内容读取到字符串中

问题

你想把一个文件的全部内容加载到一个变量中。例如，你想确定一个文件中的文本是否与一个正则表达式匹配。

方案

使用file_get_contents()，如例23-16所示。

例23-16：把一个文件的内容读取到字符串中

```
<?php
$people = file_get_contents('people.txt');
if (preg_match('/Names:.*(David|Susannah)/i',$people)) {
    print "people.txt matches.";
}
?>
```

讨论

如果你想要对文件中的内容进行处理，那么使用file_get_contents()非常合适。但是，如果你只是想将文件中的全部内容输出，可以使用比先读到字符串中再打印字符串更容易（也更有效）的方式。PHP为此提供了两个函数：第一个是fpassthru($fh)，这个函数会输出文件句柄$fh中的所有内容，然后再关闭文件句柄；第二个是readfile($filename)，它会直接输出$filename中的全部内容。

可以通过readfile()来对不需要显示的图像进行封装。例23-17中的程序确保了被请求的图像来自一周之内。

例23-17：显示最新的图像

```
<?php
$image_directory = '/usr/local/images';

if (preg_match('/^[a-zA-Z0-9]+\.(gif|jpe?g)$/',$image,$matches) &&
    is_readable($image_directory."/$image") &&
    (filemtime($image_directory."/$image") >= (time() - 86400 * 7))) {

  header('Content-Type: image/'.$matches[1]);
  header('Content-Length: '.filesize($image_directory."/$image"));

  readfile($image_directory."/$image");

} else {
  error_log("Can't serve image: $image");
}
?>
```

为了保证封装的有效性，保存图像的目录 —— $image_directory —— 必须位于Web服务器的文档根目录之外。否则，用户就有可能直接访问到图像文件。上面的代码从三个方面对图像文件进行了测试。首先，$image保存的文件名只由字母和数字构成，并且以*.gif*，*.jpg*或*.jpeg*结尾。我们要保证诸如“和/”之类的字符不会出现在文件名中，这样可以避免恶意用户从指定的目录之外取得文件。其次，我们通过is_readable()来确保程序可以读取这个文件。最后，我们通过filemtime()取得文件的修改时间并确保该时间在86400×7秒之前的后面（译注2）。因为一天是86400秒，所以86400×7是一周。如果所有这些条件测试都通过，就可以准备发送图像了。首先，我们将两个头部信息 —— 图像的MIMI类型和文件大小 —— 发送给浏览器。然后，我们使用readfile()将整个文件的内容发送给用户。

参见

filesize()函数的文档（*http://www.php.net/filesize*），fread()函数的文档（*http://www.php.net/fread*），fpassthru()函数的文档（*http://www.php.net/fpassthru*）和readfile()函数的文档（*http://www.php.net/readfile*）。

译注2：即从当前时间起一周时间之内。

23.6 计算文件中的行数、段数或记录数

问题

你想计算一个文件中包含多少行、多少个段落或者多少条记录。

方案

使用fgets()来计算行数，如例23-18所示。因为该函数一次读取文件中的一行，所以通过计算在到达文件末尾前调用该函数的次数就可以得到行数。

例23-18：计算一个文件中包含多少行

```
<?php
$lines = 0;

if ($fh = fopen('orders.txt','r')) {
  while (! feof($fh)) {
    if (fgets($fh)) {
      $lines++;
    }
  }
}
print $lines;
?>
```

要计算段落数，则只在读到一个空行的情况下增加计数器的值，如例23-19所示。

例23-19：计算一个文件中包含多少段落

```
<?php
$paragraphs = 0;

if ($fh = fopen('great-american-novel.txt','r')) {
  while (! feof($fh)) {
    $s = fgets($fh);
    if (("\n" == $s) || ("\r\n" == $s)) {
      $paragraphs++;
    }
  }
}
print $paragraphs;
?>
```

要计算记录数，只在读取的那一行中恰好包含记录分隔符和空白符时增加计数器的值。在例23-20中，记录分隔符包含在$record_separator中。

例23-20：计算一个文件中包含多少记录

```
<?php
$records = 0;
$record_separator = '--end--';

if ($fh = fopen('great-american-textfile-database.txt','r')) {
  while (! feof($fh)) {
    $s = rtrim(fgets($fh));
    if ($s == $record_separator) {
      $records++;
    }
  }
}
print $records;
?>
```

讨论

在例23-18中，$lines当fgets()返回true时就会加1。在fgets()从头到尾逐行读取一个文件中的内容时，它会返回所取得的第一行内容。当读取到最后一行时它返回false，所以$lines的累计数不会有错。此时，由于已经满足了EOF条件，feof()会返回true，循环终止。

例23-19虽然对简单的文件非常合适，但如果文件中存在长空行字符串或当文件中不存在两个连续的换行符时有可能得出意想不到结果。这些问题可以通过基于preg_split()的函数来进行补救。如果文件比较小，能够读到内存中，可以使用例23-21中的函数pc_split_paragraphs()。这个函数能够返回包含文件中每个段落的数组。

例23-21：pc_split_paragraphs()

```
<?php
function pc_split_paragraphs($file,$rs="\r?\n") {
    $text = file_get_contents($file);
    $matches = preg_split("/(.*?$rs)(?:$rs)+/s",$text,-1,
                          PREG_SPLIT_DELIM_CAPTURE|PREG_SPLIT_NO_EMPTY);
    return $matches;
}
?>
```

例23-21中，文件的内容根据两个或多个连续的换行符断开并返回到$matches数组中。默认的记录分隔正则表达式\r?\n能匹配Windows和Unix中的换行符。

如果由于文件过大无法一次读到内存中，可以使用例23-22中的pc_split_paragraphs_largefile()函数，这个函数以16KB的块为单位读取文件。

例23-22：pc_split_paragraphs_largefile()

```
<?php
function pc_split_paragraphs_largefile($file,$rs="\r?\n") {
    global $php_errormsg;

    $unmatched_text = '';
    $paragraphs = array();

    $fh = fopen($file,'r') or die($php_errormsg);

    while(! feof($fh)) {
        $s = fread($fh,16384) or die($php_errormsg);
        $text_to_split = $unmatched_text . $s;

        $matches = preg_split("/(.*?$rs)(?:$rs)+/s",$text_to_split,-1,
                              PREG_SPLIT_DELIM_CAPTURE|PREG_SPLIT_NO_EMPTY);

        // 如果上一块不是以两个记录分隔符结尾，
        // 将其保存到下一部分的前面
        $last_match = $matches[count($matches)-1];
        if (! preg_match("/$rs$rs\$/",$last_match)) {
            $unmatched_text = $last_match;
            array_pop($matches);
        } else {
            $unmatched_text = '';
        }

        $paragraphs = array_merge($paragraphs,$matches);
    }

    // 当读取完所有部分后，如果仍有最后一个不以记录分隔符结尾的块
    // 将其视为一个段落
    if ($unmatched_text) {
        $paragraphs[] = $unmatched_text;
    }
    return $paragraphs;
}
?>
```

这个函数使用了与pc_split_paragraphs()中相同的正则表达式，将文件内容拆分成相应的段落。当它发现从文件中读取的块中包含段落的结尾时，会将块中文本的剩余部分保存到$unmatched_text中，并将其放在读取的下一个块的前面，即这个块会在文件中下一个段落的开始处包含这些未匹配的文本。

例23-20中的记录计算函数通过fgets()算出了每一行的长度。如果你能提供一个行长度的合理上限，stream_get_line()能够提供计算记录数目的更精确的方式。这个函数在读取一行时以达到一定字节数或者遇到特定定界符为准。在例23-23中，将记录分隔符作为定界符提供给了这个函数。

例23-23：通过stream_get_line()计算一个文件中有多少记录

```
<?php
$records = 0;
$record_separator = '--end--';

if ($fh = fopen('great-american-textfile-database.txt','r')) {
    $done = false;
    while (! $done) {
        $s = stream_get_line($fh, 65536, $record_separator);
        if (feof($fh)) {
            $done = true;
        } else {
            $records++;
        }
  }
}
print $records;
?>
```

例23-23假设每个记录不会超过64KB（65536字节）。每次调用stream_get_line()返回一个记录，不包含记录分隔符。当stream_get_line()前进到最后一个记录分隔符时，表明它到达了文件的末尾处，所以$done被设置为true，并停止计算记录数目。

参见

fgets()函数的文档（*http://www.php.net/fgets*），feof()函数的文档（*http://www.php.net/feof*），preg_split()函数的文档（*http://www.php.net/preg-split*）和stream_get_line()函数的文档（*http:// www.php.net/stream_get_line*）。

23.7 处理文件中的每一个词

问题

你想对文件中出现的每一个单词分别进行处理。例如，你想建立一个有关计算机用语与文档用语类似性的语汇索引。

方案

通过fgets()读取每一行，将每行分为单词，对每个单词进行处理，如例23-24所示。

例23-24：处理文件中的每一个单词

```
<?php
$fh = fopen('great-american-novel.txt','r') or die($php_errormsg);
```

```
while (! feof($fh)) {
    if ($s = fgets($fh)) {
        $words = preg_split('/\s+/',$s,-1,PREG_SPLIT_NO_EMPTY);
        // 处理单词
    }
}
fclose($fh) or die($php_errormsg);
?>
```

讨论

例23-25对文件中的平均词长进行了计算。

例23-25：计算平均单词长度

```
<?php
$word_count = $word_length = 0;

if ($fh = fopen('great-american-novel.txt','r')) {
  while (! feof($fh)) {
    if ($s = fgets($fh)) {
      $words = preg_split('/\s+/',$s,-1,PREG_SPLIT_NO_EMPTY);
      foreach ($words as $word) {
        $word_count++;
        $word_length += strlen($word);
      }
    }
  }
}

print sprintf("The average word length over %d words is %.02f characters.",
              $word_count,
              $word_length/$word_count);
?>
```

处理单词会因如何定义一个“单词”而产生不同的结果。本技巧中的代码使用了Perl兼容正则表达式引擎的\s空白元字符，包含空格符、制表符、回车符和进纸符。而技巧1.5中是基于空格符来将一行文本分成多个单词的，在该技巧中那样做是没问题的，因为分开的单词还须通过空格符重新连接起来。在Perl兼容引擎中同样也包含词边界断言（\b），用于匹配单词字符（字母和数字字符）与非单词字符（其他字符）之间的位置。使用\b同使用\s来界定单词在对待嵌入在单词中的标点符号时会存在显著区别。比如，6 o'clock在使用空格符拆分时是两个单词（6和o'clock），而在根据词边界来拆分时则是四个单词（6，o，'和clock）。

参见

技巧22.2中有关使用正则表达式匹配单词的讨论；技巧1.5中有关将一行文本拆分成多个

单词的内容；fgets()函数的文档（*http://www.php.net/fgets*）；preg_split()函数的文档（*http://www.php. net/preg-split*）和Perl兼容正则表达式扩展的文档（*http://www.php. net/pcre*）。

23.8 从文件中随机提取一行

问题

你想从文件中随机提取一行文本。例如，你想从一个谚语文件中选择显示条目。

方案

使用例23-26中提供的pc_randomint()函数，该函数对文件中的所有行都分配了相同的被选机会。

例23-26：随机地找到文件中的一行

```
<?php
function pc_randomint($max = 1) {
  $m = 1000000;
  return ((mt_rand(1,$m * $max)-1)/$m);
}
?>

$line_number = 0;

$fh = fopen('sayings.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    if ($s = fgets($fh)) {
        $line_number++;
        if (pc_randomint($line_number) < 1) {
            $line = $s;
        }
    }
}
fclose($fh) or die($php_errormsg);
?>
```

讨论

pc_randomint()函数会计算一个位于0和$max之间的十进制数，包括0但不包含$max。在读取每一行的同时，行计数器会自增1，而pc_randomint()也会生成一个位于0和$line_number之间的随机数。如果生成的随机数小于1，则将当前行作为选中行。在读取完所有行之后，被选中作为随机行的最后一行保存在$line中。

这个算法巧妙地保证了在包含n行的文件中，每一行都有1/n的几率被选中，而且也不用把所有n行都保存到内存中。

参见

mt_rand()函数的文档（*http://www.php.net/mt-rand*）。

23.9 随机化处理文件中的所有行

问题

你想对一个文件中的所有行随机处理重新排序。比如，你有一个文件保存了各种有趣的引语，而你想随便挑出其中一句。

方案

通过file()将文件中的所有行读取到一个数组中，然后打乱数组中元素的顺序，如例23-27所示。

例23-27：对一个文件中的所有行进行随机化处理

```
<?php
$lines = file('quotes-of-the-day.txt');
$lines = shuffle($lines);
?>
```

讨论

通过shuffle()函数可以将数组中的元素进行随机地重新排序处理，因此在调用shuffle()函数后，就可以将$lines[0]作为选中显示的引语了。

参见

技巧4.20中有关shuffle()函数的介绍；shuffle()函数的文档（*http://www.php.net/shuffle*）。

23.10 处理长度可变的文本字段

问题

你想从一个文件中读取分隔的文本字段。例如，你可能有一个由数据库程序生成的文件，其中每条记录保存为文件中的一行，而每条记录中的字段之间通过制表符隔开。你想对这个文件进行解析并将结果保存到一个数组中。

方案

如例23-28所示，读取文件中的每一行，然后根据其分隔符进行拆分。

例23-28：处理长度可变的文本字段

```
<?php
$delim = '|';

$fh = fopen('books.txt','r') or die("can't open: $php_errormsg");
while (! feof($fh)) {
    $s = rtrim(fgets($fh));
    $fields = explode($delim,$s);
    // …… 对数据进行某些处理 ……
}
fclose($fh) or die("can't close: $php_errormsg");
?>
```

讨论

要对保存在*books.txt*中的下列数据进行处理：

```
Elmer Gantry|Sinclair Lewis|1927
The Scarlatti Inheritance|Robert Ludlum|1971
The Parsifal Mosaic|Robert Ludlum|1982
Sophie's Choice|William Styron|1979
```

可以像例23-29中所示的那样分别对每条记录进行拆分。

例23-29：处理图书列表

```
<?php
$fh = fopen('books.txt','r') or die("can't open: $php_errormsg");
while (! feof($fh)) {
    $s = rtrim(fgets($fh));
    list($title,$author,$publication_year) = explode('|',$s);
    // …… 对数据进行某些处理 ……
}
fclose($fh) or die("can't close: $php_errormsg");
?>
```

如果你为fgets()提供了一个行长度参数，这个参数至少要与最长记录的长度相同，才能保证记录不会截掉。

调用rtrim()也是必需的，因为fgets()读取行时会包含末尾的空白字符。如果不使用rtrim()，会导致$publication_year的末尾带有一个换行符。

参见

技巧1.14中有关将字符串分隔成片段的讨论；技巧1.11和1.13中有关分析逗号分隔以及固定宽度数据的内容；explode()函数的文档（*http://www.php.net/explode*）和rtrim()函数的文档（*http://www. php.net/rtrim*）。

23.11 读取配置文件

问题

你想通过配置文件来对程序进行初始设置。

方案

使用parse_ini_file()，如例23-30所示。

例23-30：解析一个配置文件

```
<?php
$config = parse_ini_file('/etc/myapp.ini');
?>
```

讨论

函数parse_ini_file()能够读取像PHP的主配置文件*php.ini*一样结构化的配置文件。该函数不会将配置文件中的设置应用到PHP程序中，而只是将文件中的值返回到一个数组中。

例如，当parse_ini_file()解析包含以下内容的文件时：

```
; physical features
eyes=brown
hair=brown
glasses=yes
```

```
; other features
name=Susannah
likes=monkeys,ice cream,reading
```

返回的数组如下：

```
Array
(
    [eyes] => brown
    [hair] => brown
    [glasses] => 1
    [name] => Susannah
    [likes] => monkeys,ice cream,reading
)
```

我们注意到，配置文件中的空行以及以“;”开头的行都会被忽略，而其他name=value形式的行则会被保存到一个数组元素中，并且以每行的name作为键，以value作为值。如果值为on或yes，则在数组中会保存为1；如果值为off或no，则在数组中保存为空字符串。

如果想按部分解析配置文件，可以向parse_ini_file()传递1作为第二个参数，配置文件中的各个部分是以方括号中的词相互区分的：

```
[physical]
eyes=brown
hair=brown
glasses=yes

[other]
name=Susannah
likes=monkeys,ice cream,reading
```

如果这些内容保存在文件*/etc/myapp.ini*中，那么，

```
$conf = parse_ini_file('/etc/myapp.ini',1);
```

会把如下数组保存到$conf中：

```
Array
(
    [physical] => Array
        (
            [eyes] => brown
            [hair] => brown
            [glasses] => 1
        )

    [other] => Array
        (
            [name] => Susannah
```

```
            [likes] => monkeys,ice cream,reading
        )

)
```

对程序进行初始化的另一种手段是将配置项保存为一个合法的PHP文件，通过require加载这个文件。如果文件*config.php*包含如下内容：

```
<?php

// 自然特征
$eyes = 'brown';
$hair = 'brown';
$glasses = 'yes';

// 其他特征
$name = 'Susannah';
$likes = array('monkeys','ice cream','reading');
?>
```

那么，通过简单的require 'config.php';就可以调用变量 $eyes，$hair，$glasses，$name和$likes的值了。

由require加载的配置文件必须是合法的PHP文件 —— 包含开始标签<?php和结束标签?>。而且，*config.php*中的变量名是明确设置的，不是像使用parse_ini_file()时被保存在数组中。对于简单的配置文件，这种技术不需要太复杂的语法，但是如果包含类似例23-31所示的语句，就可以在配置文件中嵌入相应的逻辑，这是比较有用的。

例23-31：配置文件中嵌入的逻辑

```
<?php

$time_of_day = (date('a') == 'am') ? 'early' : 'late';

?>
```

参见

parse_ini_file()函数的文档（*http://www.php.net/parse-ini-file*）。

23.12 不通过临时文件而实现对文件的修改

问题

你想在修改一个文件时不使用临时文件保存修改的内容。

方案

通过file_get_contents()读取文件内容，进行修改，然后再通过file_put_contents()回写到文件中。这个过程以例23-32所示。

例23-32：适当地修改文件

```
<?php
$contents = file_get_contents('pickles.txt');
$contents = strtoupper($contents);
file_put_contents('pickles.txt', $contents);
?>
```

讨论

例23-33将把以星号或斜杠强调的文本转换成HTML的<b>或<i>标签。

例23-33：适当地将一个文件HTML化

```
<?php
$contents = file_get_contents('message.txt');
// 将 *word* 转换成 <b>word</b>
$contents = preg_replace('@\*(.*?)\*@i','<b>$1</b>',$contents);
// 将 /word/ 转换成 <i>word</i>
$contents = preg_replace('@/(.*?)/@i','<i>$1</i>',$contents);
file_put_contents('message.txt', $contents);
```

由于添加HTML标签会使文件体积变大，所以整个文件必须读到内存中进行处理。如果对文件的修改会使每一行收缩（或保持长度不变），该文件就可以被逐行地处理，以节省内存。例23-34所示将以<b>和<i>标记的文本转换成以星号和斜杠标记的文本的过程。

例23-34：适当地将一个文件文本化

```
<?php
$fh = fopen('message.txt','r+')         or die($php_errormsg);

// 计算要读取的字节数
$bytes_to_read = filesize('message.txt');

// 初始化保存文件位置的变量
$next_read = $last_write = 0;

// 如果仍有要读的字节，继续
while ($next_read < $bytes_to_read) {

    /* 转移到下次要读取的位置，读取一行，并保存
     * 下一次要读取的位置 */
    fseek($fh,$next_read);
    $s = fgets($fh)                     or die($php_errormsg);
    $next_read = ftell($fh);
```

```
    // 将 <b>word</b> 转换成 *word*
    $s = preg_replace('@<b[^>]*>(.*?)</b>@i','*$1*',$s);
    // 将 <i>word</i> 转换成 /word/
    $s = preg_replace('@<i[^>]*>(.*?)</i>@i','/$1/',$s);

    /* 转移到上一次写入结束的位置，写入
     * 转换后的行，并保存下一次写入的位置 */
    fseek($fh,$last_write);
    if (-1 == fwrite($fh,$s))            { die($php_errormsg); }
    $last_write = ftell($fh);
}

// 按写入的长度对文件进行截取
ftruncate($fh,$last_write)            or die($php_errormsg);

// 关闭文件
fclose($fh)                             or die($php_errormsg);
```

参见

技巧13.12和13.13中有关在纯文本和HTML文档之间转换的更多信息；fseek()函数的文档（*http:// www.php.net/fseek*），rewind()函数的文档（*http://www.php.net/rewind*），ftruncate()函数的文档（*http://www.php.net/ftruncate*），file_get_contents()函数的文档（*http://www.php.net/file_get_contents*）和file_put_contents()函数的文档（*http://www.php.net/file_put_contents*）。

23.13 将缓冲内容刷出到一个文件中

问题

你想将所有缓冲的数据强制写入到一个文件句柄中。

方案

使用fflush()，如例23-35所示。

例23-35：强制刷出缓冲内容

```
<?php
fwrite($fh,'There are twelve pumpkins in my house.');
fflush($fh);
?>
```

例23-35确保了将“There are twelve pumpkins in my house.”写入到$fh中。

讨论

为了获得更高的效率，系统输入/输出库通常不会在接到你的指令时就向文件中写入数据，而是先将数据成批保存在缓冲器中，然后再一次性地全部写入到磁盘上。使用fflush()会强制缓冲器中任何等待写入的内容立即被实际写入到磁盘上。

刷出缓冲内容在生成存取或活动日志时尤其有用。通过在每个要写入到日志文件中的信息之后调用fflush()，可以确保任何监控该日志文件的程序或人能尽早地发现该信息。

参见

fflush()函数的文档（*http://www.php.net/fflush*）。

23.14 写入标准输出

问题

你想将数据写入到标准输出中。

方案

使用echo或print()输出，如例23-36所示。

例23-36：写入到标准输出

```
<?php
print "Where did my pastrami sandwich go?";
echo  "It went into my stomach.";
?>
```

讨论

print()是一个函数，echo是一个语言结构。也就是说，print()会返回一个值，而echo则不会。你可以在一个比较大的表达式中包含print()但不能包含echo，如例23-37所示。

例23-37：echo对print

```
<?php
// 正常
(12 == $status) ? print 'Status is good' : error_log('Problem with status!');

// 出现解析错误
(12 == $status) ? echo 'Status is good' : error_log('Problem with status!');
?>
```

如果你使用文件函数“$fh = fopen('php://stdout','w')”或“die($php_errormsg);”则需要以“*php://stdout*”作为文件名。

如果需要抽象输出目标或者希望在输出到标准输出的同时将数据写入到文件中，那么通过文件句柄而不是简单地通过print()或echo向标准输出中写入数据是很有用的，具体细节请参考技巧23.15。

也可以通过打开*php://stderr*: $fh = fopen('php://stderr','w');来写入到标准错误中。

参见

技巧23.15有关同时写入到多个文件句柄的内容；echo的文档（*http://www.php.net/echo*）和print()函数的文档（*http://www.php.net/print*）。

23.15 同时写入到多个文件句柄

问题

你想将输出发送到多个文件句柄中。例如，你想同时将日志信息显示在屏幕并保存到日志文件中。

方案

将输出包装在迭代文件句柄的循环中，如例23-38所示。

例23-38：pc_multi_fwrite()

```
<?php
function pc_multi_fwrite($fhs,$s,$length=NULL) {
  if (is_array($fhs)) {
    if (is_null($length)) {
      foreach($fhs as $fh) {
        fwrite($fh,$s);
      }
    } else {
      foreach($fhs as $fh) {
        fwrite($fh,$s,$length);
      }
    }
  }
}

$fhs['file'] = fopen('log.txt','w') or die($php_errormsg);
$fhs['screen'] = fopen('php://stdout','w') or die($php_errormsg);
```

```
pc_multi_fwrite($fhs,'The space shuttle has landed.');
?>
```

讨论

如果你不想（或总想）给fwrite()函数传递一个长度参数，可以从pc_multi_fwrite()函数中去掉相应的检测。例23-39中的版本没有考虑$length参数。

例23-39：未涉及$length的pc_multi_fwrite()

```
<?php
function pc_multi_fwrite($fhs,$s) {
  if (is_array($fhs)) {
    foreach($fhs as $fh) {
      fwrite($fh,$s);
    }
  }
}
?>
```

参见

fwrite()函数的文档（*http://www.php.net/fwrite*）。

23.16 转义Shell中的元字符串

问题

你需要在一个命令行中合并外部数据，但必须对特殊字符进行转义，以免发生意外。例如，你想将用户输入作为一个参数传递给程序。

方案

使用escapeshellarg()处理参数，用escapeshellcmd()处理程序名称，如例23-40所示。

例23-40：转义Shell中的元字符串

```
<?php
system('ls -al '.escapeshellarg($directory));
system(escapeshellcmd($ls_program).' -al');
?>
```

讨论

在命令行中使用未转义的字符是很危险的。记住：永远不要将用户原始的输入传递给PHP的壳执行（shell-execution）函数，必须要对命令中和参数中的字符进行适当转义才能传递使用。这一点非常重要。在命令行中执行来自Web表单的命令是非同小可的，也不是我们愿意推荐的。然而，的确有些时候你需要执行一个外部程序，所以对命令和参数进行转义是必要的。

escapeshellarg()要求以单引号引住参数（并对存在于单引号中的任何字符进行转义）。例23-41使用了escapeshellarg()来打印一个特定进程的过程状态。

例23-41：使用 escapeshellarg()

```
<?php
system('/bin/ps '.escapeshellarg($process_id));
?>
```

通过escapeshellarg()，即使在ID中存在意想不到字符（例如，一个空格）的情况下仍能保证正确的显示这一过程，同时也能避免无意运行的命令。如果$process_id中包含“1; rm -rf/”，那么system ("/bin/ps $process_id")不仅会显示进程1的状态，而且也会执行*rm -rf/*命令。

但是，system('/bin/ps '.escapeshellarg($process_id))运行命令“*/bin/ps 1; rm -rf*”，会产生一个错误，因为“1-semicolon-space-rm-space-hyphen-rf”并非一个合法的ID。

类似地，escapeshellcmd()会防止执行无意执行的命令。命令“system("/usr/local/bin/formatter-$which_ program");”会根据$which_program的值来选择运行相应的程序。

例如，如果$which_program的值是pdf 12，脚本就会以参数12来运行*/usr/local/bin/formatter-pdf*。而如果$which_program的值是pdf 12; 56，那么脚本会以参数12来运行*/usr/local/bin/formatter-pdf*，但也会运行程序56——但这会导致错误。

要保证成功地向*formatter-pdf*传递参数，就需要运行escapeshellcmd()：“system(escapeshellcmd ("/usr/local/bin/formatter-$which_program"));”。这样，在运行*/usr/local/bin/formatter-pdf*的时候就会向其中传递两个参数：12和56。

参见

system()函数的文档（*http://www.php.net/system*）；escapeshellarg()函数的文档

（*http://www.php. net/escapeshellarg*）和escapeshellcmd()函数的文档（*http://www.php. net/escapeshellcmd*）。

23.17 向程序传递输入数据

问题

你想将输入数据传递给运行于PHP脚本中的外部程序。例如，你可能会使用一个通过运行外部程序来编制文本索引的数据库，并希望将文本传递给那个程序。

方案

使用popen打开一个通向相应程序的管道，通过fputs()或fwrite()将数据写入到管道中，最后再以pclose()关闭管道，如例23-42所示。

例23-42：向程序传递输入数据

```
<?php
$ph = popen('/usr/bin/indexer --category=dinner','w') or die($php_errormsg);
if (-1 == fputs($ph,"red-cooked chicken\n"))    { die($php_errormsg); }
if (-1 == fputs($ph,"chicken and dumplings\n")) { die($php_errormsg); }
pclose($ph)                                    or die($php_errormsg);
?>
```

讨论

例23-43使用popen()调用*nsupdate*命令，该命令会向域名服务器（nameservers）提交动态DNS更新（Dynamic DNS Update）请求。

例23-43：使用popen()调用nsupdate命令

```
<?php
$ph = popen('/usr/bin/nsupdate -k keyfile')                 or die($php_errormsg);
if (-1 == fputs($ph,"update delete test.example.com A\n")) { die($php_errormsg); }
if (-1 == fputs($ph,"update add test.example.com 5 A 192.168.1.1\n"))
                                                            { die($php_errormsg); }
pclose($ph)                                                 or die($php_errormsg);
?.
```

在例23-43中，通过popen()向*nsupdate*发送了两个命令：第一个命令用于删除*test.example.com* A记录，第二个命令用于添加一个IP地址为192.168.1.1的*test.example.com* A记录。

参见

popen()函数的文档（*http://www.php.net/popen*）和pclose()函数的文档（*http://www.php.net/ pclose*）。RFC 2136规范中有关动态DNS的说明（*http://www.faqs.org/rfcs/rfc2136.html*）。

23.18 从程序中读取标准输出

问题

你想从一个程序中读取输出的数据。例如，你想得到诸如*route(8)*这类提供网络信息的系统实用程序输出的数据。

方案

要读取程序输出的完整内容，使用backtick(`)操作符，如例23-44所示。

例23-44：通过backtick运行一个程序

```
<?php
$routing_table = `/sbin/route`;
?>
```

要读取增量输出的内容，可以通过用popen()打开一个管道的方式，如例23-45所示。

例23-45：从popen()中读取输出

```
<?php
$ph = popen('/sbin/route','r') or die($php_errormsg);
while (! feof($ph)) {
    $s = fgets($ph)            or die($php_errormsg);
}
pclose($ph)                    or die($php_errormsg);
?>
```

讨论

backtick操作符（在安全模式下是无效的）会执行一个程序，并将程序输出的全部内容作为一个字符串返回。在一个配置为448MB内存的Linux系统中，命令`$s = '/usr/bin/free';`会将下列多个字符串保存到`$s`中：

```
             total       used       free     shared    buffers     cached
Mem:         48620     446384       2236          0      68568     163040
-/+ buffers/cache:     214776     233844
Swap:       136512          0     136512
```

如果程序生成的输出很多，则从管道中读取一行数据的效率会更高。如果你准备将基于管道的输出以格式化数据的形式输出到浏览器，那么可以按照取得的形式进行输出。例23-46以一个HTML表格的形式将最近登录Unix系统的信息输出，其中使用了*/usr/bin/last*命令。

例23-46：通过popen()输出最近登录信息

```
<?php
// 打印表头
print<<<_HTML_
<table>
<tr>
 <td>user</td><td>login port</td><td>login from</td><td>login time</td>
 <td>time spent logged in</td>
</tr>
_HTML_;

// 打开通往 /usr/bin/last 的管道
$ph = popen('/usr/bin/last','r') or die($php_errormsg);
while (! feof($ph)) {
    $line = fgets($ph) or die($php_errormsg);

    // 不处理空行或者结尾的信息行
    if (trim($line) && (! preg_match('/^wtmp begins/',$line))) {
        $user = trim(substr($line,0,8));
        $port = trim(substr($line,9,12));
        $host = trim(substr($line,22,16));
        $date = trim(substr($line,38,25));
        $elapsed = trim(substr($line,63,10),' ()');

        if ('logged in' == $elapsed) {
            $elapsed = 'still logged in';
            $date = substr_replace($date,'',-5);
        }

        print "<tr><td>$user</td><td>$port</td><td>$host</td>";
        print "<td>$date</td><td>$elapsed</td></tr>\n";
    }
}
pclose($ph) or die($php_errormsg);

print '</table>';
?>
```

参见

popen()函数的文档（*http://www.php.net/popen*）；pclose()函数的文档（*http://www.php.net/pclose*）；backtick操作符的文档（*http://www.php.net/language.operators.execution*）和安全模式的资料（*http://www.php.net/features.safe-mode*）。

23.19 从程序中读取标准错误信息

问题

你想从一个程序中读取输出的错误信息。例如，你想捕捉由*strace(1)*显示的系统调用信息。

方案

通过在传递给popen()的命令行指令中添加2>&1将标准的错误重定向到标准输出，再通过以r模式打开管道来读取标准输出，这个过程如例23-47所示。

例23-47：读取标准错误

```
<?php
$ph = popen('strace ls 2>&1','r') or die($php_errormsg);
while (!feof($ph)) {
    $s = fgets($ph)               or die($php_errormsg);
}
pclose($ph)                       or die($php_errormsg);
?>
```

讨论

无论是在Unix的*sh*还是Windows的*cmd.exe*壳程序（shells）中，标准错误都是文件描述符2，而标准输出都是文件描述符1。在命令中加上2>&1就是指示壳程序将正常发送给文件描述符2（标准错误）的内容重定向到文件描述符1（标准输出）。fgets()用于读取标准错误和标准输出。

这个技术虽然能够读取标准错误，但却不会提供与标准输出相区别的方式。如果只想读取标准错误，必须要阻止通过管道返回的标准输出。这在Unix中是通过将其重定向到*/dev/null*，在Windows中重定向到NUL来实现的，如例23-48所示。

例23-48：重定向标准输出

```
<?php
// Unix: 只读取标准错误
$ph = popen('strace ls 2>&1 1>/dev/null','r') or die($php_errormsg);

// Windows: 只读取标准错误
$ph = popen('ipxroute.exe 2>&1 1>NUL','r') or die($php_errormsg);
?>
```

参见

popen()函数的文档；*popen(3)*的在线手册（manpage）中有关系统通过popen()使用的shell程序的详细资料；Unix系统中*sh(1)*在线手册的Redirection部分中有关shell重定向的信息；Windows系统帮助的命令参考（Command Reference）部分中有关重定向的介绍。

23.20 锁定文件

问题

你想拥有一个文件的排他权，以防止在对其进行读取和更新时被修改。例如，如果你想将留言信息保存到一个文件中，两个用户应该能够在不相互干扰对方留言的情况下同时添加留言。

方案

使用flock()对文件执行“劝告式（advisory）”锁定，如例23-49所示。

例23-49：使用劝告式文件锁定

```
<?php
$fh = fopen('guestbook.txt','a')         or die($php_errormsg);
flock($fh,LOCK_EX)                       or die($php_errormsg);
fwrite($fh,$_POST['guestbook_entry'])    or die($php_errormsg);
fflush($fh)                              or die($php_errormsg);
flock($fh,LOCK_UN)                       or die($php_errormsg);
fclose($fh)                              or die($php_errormsg);
?>
```

讨论

flcok()提供的文件锁定方式称为“劝告式”锁定，是因为flock()不会真正阻止其他进程打开锁定的文件，它提供的只是一种使进程之间能够以自动协作方式对文件进行存取的机制。所有需要对flock()锁定的文件进行存取操作的程序，都需要对文件设置并释放锁定状态，以保证文件锁定有效。

通过flock()可以设定两种锁定方式：排他锁和共享锁。所谓排他锁，是通过将LOCK_EX作为第二个参数传递给flock()实现的，在这种锁定方式下，特定文件每次只能由一个进程控制。而所谓共享锁，通过将LOCK_SH作为第二个参数传递给flock()实现，在共享锁定方

式下，特定文件每次可能由多个进程控制。在写入文件之前，应该使用排他锁。在读取文件之前，应该使用共享锁。

如果你在代码中某个位置使用flock()锁定了一个文件，那么这个文件在代码中任何位置都会处于锁定状态。例如，如果你的一部分程序在写入文件之前使用LOCK_EX设置了一个排他锁，那么如果在程序其他部分中需要从这个文件中读取内容，必须确保使用LOCK_SH将文件的锁定方式设置为共享锁。如果你没有那么做，则试图读取这个文件的进程就会在另一个进程正在写入该文件时看到正在写入的内容。

在释放锁定的文件时，将LOCK_UN作为第二个参数来调用flock()。当你解锁一个文件时，重要的一点就是要在解锁之前先调用fflush()将所有缓冲的数据刷出到要写入的文件当中。其他进程不应该在数据被写入文件之前对该文件设置锁定状态。

默认情况下，flock()在得到一个锁之前会处于阻塞（block）状态。如果不想让它处于阻塞状态，可以将LOCK_NB作为第二个参数。非阻塞锁定状态如例23-50所示。

例23-50：非阻塞锁定

```
<?php
$fh = fopen('guestbook.txt','a')         or die($php_errormsg);
$tries = 3;
while ($tries > 0) {
    $locked = flock($fh,LOCK_EX | LOCK_NB);
    if (! $locked) {
        sleep(5);
        $tries--;
    } else {
        // 不再执行循环
        $tries = 0;
    }
}
if ($locked) {
    fwrite($fh,$_POST['guestbook_entry']) or die($php_errormsg);
    fflush($fh)                             or die($php_errormsg);
    flock($fh,LOCK_UN)                    or die($php_errormsg);
    fclose($fh)                             or die($php_errormsg);
} else {
    print "Can't get lock.";
}
?>
```

当锁定状态为非阻塞时，即使不能取得锁，flock()也会立即返回。前面的例子分三次尝试取得*guestbook.txt*的锁，每次尝试之间休眠5秒钟。

通过flock()对文件加锁并不是任何环境下都有效，比如在某些NFS实现和旧版本的Windows系统中。要在这些情况下模仿文件锁定，可以使用目录作为一个排他锁指示

器。排他锁指示器指的是一个单独的、空文件夹，它的存在表示数据文件是被锁定的。在打开数据文件之前，可以创建一个锁文件夹，而在完成对数据文件的操作后再删除这个锁文件夹即可。其他方面，有关文件存取的代码就都相同了，如例23-51所示。

例23-51：通过mkdir()来模仿锁定状态

```
<?php
// 循环到能够生成锁定文件夹为止
$locked = 0;
while (! $locked) {
    if (@mkdir('guestbook.txt.lock',0777)) {
        $locked = 1;
    } else {
        sleep(1);
    }
}
$fh = fopen('guestbook.txt','a')         or die($php_errormsg);

if (-1 == fwrite($fh,$_POST['guestbook_entry'])) {
    rmdir('guestbook.txt.lock');
    die($php_errormsg);
}
if (! fclose($fh)) {
    rmdir('guestbook.txt.lock');
    die($php_errormsg);
}
rmdir('guestbook.txt.lock')              or die($php_errormsg);
```

之所以使用文件夹而不使用文件来标识锁定状态，是因为当要创建的文件夹已经存在时，mkdir()函数就不能再创建它。这样就能通过检查锁指示器是否存在来决定何时创建它。但是，在创建了文件夹之后的任何错误捕获（trapping）都必须在退出之前通过删除文件夹来清除。如果不删除文件夹，那么将来的进程都不可能再通过创建这个文件夹来取得锁状态。

如果你使用文件而不是文件夹来作为锁指示器，那么相应的代码可能会与例23-52中的代码类似。

例23-52：带有错误倾向的文件锁

```
$locked = 0;
while (! $locked) {
    if (! file_exists('guestbook.txt.lock')) {
        touch('guestbook.txt.lock');
        $locked = 1;
    } else {
        sleep(1);
    }
}
```

例23-52会在负载加重时出现问题，因为它是通过file_exists()来检查锁定状态（文件

是否存在），然后再通过touch()来创建锁文件的。当第一个进程调用完file_exists()后，第二个进程有可能会在第一个进程调用touch()之前先调用touch()。而后两个进程都会认为得到了文件的排他锁，而实际上哪个进程也没有得到。而如果使用mkdir()就不会在检查和创建之间存在时间间隙（时间差），所以可以保证生成文件夹的进程得到排他权限。

参见

flock()函数的文档（*http://www.php.net/flock*）。

23.21 读写自定义文件类型

问题

你想使用PHP的标准文件存取函数访问可能不是保存在文件中的数据。例如，你想使用文件存取函数在共享内存中执行读写操作，或者想要对已经读取但尚未传送到PHP程序的文件内容进行处理。

方案

编写一个流封装程序对往返于PHP和你的自定义位置或自定义格式之间的数据进行处理。所谓的流封装程序是一个类，这个类必须实现PHP需要访问你的自定义数据流的各种方法，包括：打开、关闭、读取、写入等等。一个特定的包装程序应该通过特定的前缀进行注册。当把文件名传递给fopen()、include()或者其他任何PHP文件处理函数时要使用相应的前缀，以保证调用必要的流封装程序。

PEAR的Stream_SHM模块实现了一个对共享内存执行读写操作的流封装程序。例23-53所示为如何使用这个流封装程序。

例23-53：使用自定义的流封装程序

```
<?php
require_once 'Stream/SHM.php';
stream_register_wrapper('shm','Stream_SHM') or die("can't register shm");
$shm = fopen('shm://0xabcd','c');
fwrite($shm, "Current time is: " . time());
fclose($shm);
?>
```

讨论

流封装程序不仅对于操作非文件数据源比较方便，而且还能用于对向PHP传递途中的文件内容进行预处理。Mike Naberezny为此举了一个在应用到模板时的例子。当把short_open_tags设置为off时，在模板中输出一个对象技巧变量代码相当冗长<?php echo $this->property; ?>。Miker的解决方案通过使用一个流封装程序使得@字符能够代表$this->（译注3）。例23-54是这个流封装程序的代码，例23-55是一个例子模板，而例23-56是一个如何将前两者综合到一起应用的小示例。

例23-54：为简洁模板而开发的流封装程序

```
<?php
/**
 * Stream wrapper to convert markup of mostly PHP templates into PHP prior to
include().
 *
 * Based in large part on the example at
 * http://www.php.net/manual/en/function.stream-wrapper-register.php
 *
 * @author Mike Naberezny (@link http://mikenaberezny.com)
 * @author Paul M. Jones  (@link http://paul-m-jones.com)
 */
class ViewStream {
    /**
     * Current stream position.
     *
     * @var int
     */
    private $pos = 0;

    /**
     * Data for streaming.
     *
     * @var string
     */
    private $data;

    /**
     * Stream stats.
     *
     * @var array
     */
    private $stat;


    /**
     * Opens the script file and converts markup.
     */
    public function stream_open($path, $mode, $options, &$opened_path) {
```

译注3：大大简化了相应的操作。

```
    //获取视频脚本源
    $path = str_replace('view://', '', $path);
    $this->data = file_get_contents($path);

    /**
     * If reading the file failed, update our local stat store
     * to reflect the real stat of the file, then return on failure
     */
    if ($this->data===false) {
        $this->stat = stat($path);
        return false;
    }

    /**
     * Convert <?= ?> to long-form <?php echo ?>
     *
     * We could also convert <%= like the real T_OPEN_TAG_WITH_ECHO
     * but that's not necessary.
     *
     * It might be nice to also convert PHP code blocks <? ?> but
     * let's quit while we're ahead.  It's probably better to keep
     * the <?php for larger code blocks but that's your choice.  If
     * you do go for it, explicitly check for <?xml as this will
     * probably be the biggest headache.
     */
    if (! ini_get('short_open_tag')) {
        $find = '/\<\?\= (.*)? \?>/';
        $replace = "<?php echo \$1 ?>";
        $this->data = preg_replace($find, $replace, $this->data);
    }

    /**
     * Convert @$ to $this->
     *
     * We could make a better effort at only finding @$ between <?php ?>
     * but that's probably not necessary as @$ doesn't occur much in the wild
     * and there's a significant performance gain by using str_replace().
     */
    $this->data = str_replace('@$', '$this->', $this->data);

    /**
     * file_get_contents() won't update PHP's stat cache, so performing
     * another stat() on it will hit the filesystem again.  Since the file
     * has been successfully read, avoid this and just fake the stat
     * so include() is happy.
     */
    $this->stat = array('mode' => 0100777,
                        'size' => strlen($this->data));

    return true;
}


/**
 * Reads from the stream.
```

```
     */
    public function stream_read($count) {
        $ret = substr($this->data, $this->pos, $count);
        $this->pos += strlen($ret);
        return $ret;
    }


    /**
     * Tells the current position in the stream.
     */
    public function stream_tell() {
        return $this->pos;
    }


    /**
     * Tells if we are at the end of the stream.
     */
    public function stream_eof() {
        return $this->pos >= strlen($this->data);
    }


    /**
     * Stream statistics.
     */
    public function stream_stat() {
        return $this->stat;
    }


    /**
     * Seek to a specific point in the stream.
     */
    public function stream_seek($offset, $whence) {
        switch ($whence) {
            case SEEK_SET:
                if ($offset < strlen($this->data) && $offset >= 0) {
                $this->pos = $offset;
                    return true;
                } else {
                    return false;
                }
                break;

            case SEEK_CUR:
                if ($offset >= 0) {
                    $this->pos += $offset;
                    return true;
                } else {
                    return false;
                }
                break;
```

```
            case SEEK_END:
                if (strlen($this->data) + $offset >= 0) {
                    $this->pos = strlen($this->data) + $offset;
                    return true;
                } else {
                    return false;
                }
                break;

            default:
                return false;
        }
    }
}
?>
```

例23-55：针对流包装程序的简单模板

```
<html> <?= @$hello ?> </html>
```

例23-56：模板与流包装程序综合应用的示例

```
<?php
/** Stream wrapper */
require_once dirname(__FILE__) . DIRECTORY_SEPARATOR . 'ViewStream.php';

/**
 * A very dumb template class just to demonstrate the concept.
 *
 * @author Mike Naberezny
 * @link   http://mikenaberezny.com/archives/40
 * @link   http://phpsavant.com
 */
class IdiotSavant {
    public function __construct() {
        if (!in_array('view', stream_get_wrappers())) {
            stream_wrapper_register('view', 'ViewStream');
        }
    }

    public function render($filename) {
        include 'view://' . dirname(__FILE__) . DIRECTORY_SEPARATOR . $filename.
'.phtml';
    }
}

// 创建新视图
$view = new IdiotSavant();

//将变量"hello" 指定给视图的作用域
$view->hello = 'Hello, World!';

// 根据模板呈现视图输出<html> Hello, World! </html>"
$view->render('ExampleTemplate');
```

参见

stream_register_wrapper()函数的文档（*http://www.php.net/stream_register_wrapper*）；PEAR的Stream_SHM模块的文档（*http://pear.php.net/package/stream_shm*）；Mike Naberezny的博客文章“Symfony Templates and Ruby's ERb”（*http://www.mikenaberezny.com/archives/40*）。

23.22 读写压缩文件

问题

你想对压缩文件进行读写操作。

方案

使用compress.zlib或compress.bzip2流封装程序和标准的文件函数。例23-57从一个gzip压缩文件中读取数据。

例23-57：从压缩文件中读取数据

```
<?php
$fh = fopen('compress.zlib://lots-of-data.gz','r') or die("can't open: $php_err
ormsg");
while ($line = fgets($fh)) {
    // $line 是未压缩数据的下一行
}
fclose($fh) or die("can't close: $php_errormsg");
?>
```

讨论

compress.zlib流封装程序提供了对以gzip算法压缩的文件访问的权限。而compress.bzip2流封装程序则提供了对以bzip2算法压缩的文件访问的权限。通过这两个流封装程序都能实现对压缩文件的读、写和追加操作。要启用zlib和bzip2压缩流，在构建PHP时分别使用--withzlib和--withbz2。

除了对本地压缩文件提供访问支持的流封装程序之外，还有一些能够对特定的流进行压缩（或解压缩）的流过滤器（filters）。zlib.deflate和zlib.inflate过滤器会根据zlib“deflate”算法对数据进行压缩和解压缩。bzip2.compress和bzip2.uncompress过滤器则使用的都是bzip2算法。

每个流过滤器在创建之后都必须应用到一个流。例23-58所示为使用bzip2流过滤器从一个URL中读取压缩的数据。

例23-58：从URL中读取压缩的数据

```
<?php
$fp = fopen('http://www.example.com/something-compressed.bz2','r');
stream_filter_append($fp, 'bzip2.uncompress');
while (! feof($fp)) {
    $data = fread($fp);
    // 对 $data 进行某些操作;
}
fclose($fp);
?>
```

参见

压缩流封装程序的文档（*http://www.php.net/wrappers.compression*）；压缩过滤器的文档（*http:// www.php.net/filters.compression*）和stream_filter_append()函数的文档（*http://www.php.net/ stream_filter_append*）；RFCs 1950中*zlib*算法的详细介绍（*http://www.faqs.org/rfcs/rfc1950.html*）和1951中的资料（*http://www.faqs.org/rfcs/rfc1951.html*）。

第24章

目录

24.0 概述

文件系统中除了保存文件的实际内容之外，也保存着许多与文件相关的附加信息。这些信息包括像文件大小、目录和访问权限等这样的细节。当对文件进行处理时，可能你还会对文件的元数据进行操作。PHP为我们提供了多种读取和操作目录、目录项以及文件属性的函数。与PHP中其他文件相关的部分一样，这些函数与C语言中完成相同任务的函数类似，只是进行了某种程度的简化。

文件是通过信息节点来组织的。每个文件（以及文件系统的其他部分，例如目录、设备和连接）都拥有自己的信息节点。这个信息节点包含一个指向文件数据块位置和所有与文件相关元数据的指针。目录的数据块中保存着目录中文件的名称以及每个文件的信息节点。

PHP提供了一些用于查看目录中包含什么文件的方法。其中DirectoryIterator类（在PHP 5及以后版本中有效）提供了一个用于遍历目录的面向对象的综合接口。例24-1使用DirectoryIterator输出了一个目录中所有文件的名称。

例24-1：使用DirectoryIterator

```
<?php
foreach (new DirectoryIterator('/usr/local/images') as $file) {
    print $file->getPathname() . "\n";
}
?>
```

而opendir()、readdir() 和 closedir()函数，如例24-2所示，提供了针对同样任务的

程序上的手段。使用opendir()可以取得一个目录的句柄，readdir()用来迭代文件，而closedir()关闭目录句柄（注1）。

例24-2：程序性的目录迭代

```
<?php
$d = opendir('/usr/local/images') or die($php_errormsg);
while (false !== ($f = readdir($d))) {
    print $f . "\n";
}
closedir($d);
?>
```

在本章中，我们一般会使用DirectoryIterator来举例。

文件系统中保存的不仅是文件和目录。在Unix系统中，还会保存符号连接（symbolic links）文件。这些特殊文件的内容是指向另外文件的指针，可以删除连接而不影响连接指向的文件。若要创建一个符号连接，可以如例24-3所示，使用symlink()。

例24-3：生成一个符号连接

```
<?php
symlink('/usr/local/images','/www/docroot/images') or die($php_errormsg);
?>
```

例24-3中的代码在*/www/docroot*中创建了一个指向*/usr/local/images*的名为*images*的符号连接。

如果想找到与文件、目录或链接相关的信息，必须通过相应的信息节点来查找。stat()函数可以为你从一个信息节点中取得元数据。技巧24.2中详细讨论了stat()函数的使用。PHP中也有许多在内部使用了stat()函数可以提供有关文件的某些具体信息。这些函数如表24-1所示。

表24-1：文件信息函数

函数名	函数提供什么文件信息？
file_exists()	该文件是否存在
fileatime()	最后访问时间
filectime()	最后修改元数据的时间
filegroup()	组（数字）

注1：PHP也有一个dir()类，其中的方法映射了这些程序上的手段（打开、读取、关闭）。但由于DirectoryIterator更加强大，所以如果你希望使用面向对象（OO）的接口，还是用它比较合适。

表24-1：文件信息函数（续）

函数名	函数提供什么文件信息？
fileinode()	信息节点数
filemtime()	内容的最后修改时间
fileowner()	所有者（数字）
fileperms()	权限（十进制，数字）
filesize()	大小
filetype()	类型（fifo、char、dir、block、link、file、未知）
is_dir()	是否是目录
is_executable()	是否可执行
is_file()	是否是常规文件
is_link()	是否是一个符号连接
is_readable()	是否可读
is_writable()	是否可写

在Unix系统中，文件权限表示文件的所有者、文件组中的用户以及所有用户能够对文件执行什么操作。操作是指读取、写入和执行。对程序而言，执行意味着运行程序的能力；对目录而言，执行是在目录中搜索文件的能力。

Unix中的权限还包括setuid位、setgid位和sticky位。setuid位意味着一个程序在运行时，要使用其所有者的用户ID。setgid位意味着该程序要使用其组的组ID运行。对于一个目录，setgid位意味着在默认情况下，该目录中的新文件会在与目录相同的组中创建。sticky位对于共享文件的目录有用，因为它可以防止在一个目录内拥有写入权限的非超级用户删除该目录中的文件，除非他们拥有该文件或者该目录。

当通过chmod()设置权限（见技巧24.3）时，权限必须以一个八进制数表示。这个数有四位数字：第一位数是针对文件的任何特殊设置（如setuid或setgid），第二位数是用户权限 —— 用户能做什么，第三位数是组权限 —— 文件组中的用户能做什么，第四位数是通用（world）权限 —— 所有用户能做什么。在为每一位数计算适当的值时，可以将表24-2中的值相加从而得到你想要的权限。例如，0644的权限值意味着没有特殊设置（第一位是0），文件的所有者可以读写该文件（第二位6等于4<读>+2<写>），文件组中的用户可以读取该文件（第三位4），而所有其他用户能够读取这个文件（第四位4）。而4644这个权限值除了设置了setuid位，其他都与前面分析的相同。

表24-2：文件权限值

值	权限含义	特别设置含义
4	读	setuid
2	写	setgid
1	执行	sticky

新创建的文件和目录的权限会受一个叫做*umask*设置的影响。该设置是一个在文件或目录初始权限值（0666及0777）基础上删除或屏蔽（masked）后的权限值。例如，如果umask是0022，那么通过touch()或fopen()新创建文件的默认权限就是0644，而通过mkdir()新创建目录的默认权限就是0755。可以通过umask()取得或设置umask值。即调用这个函数时它会返回当前的umask值，如果在调用时给它传递了一个参数，就能够将umask值修改为参数所指定的值。例24-4所示为如何对新创建的文件进行限制，以防止除了文件所有者（和超级用户）之外的任何人访问该文件。

例24-4：改变默认文件权限

```
$old_umask = umask(0077);
touch('secret-file.txt');
umask($old_umask);
```

在例24-4中，第一次调用umask()对组和其他任何人屏蔽了所有权限。当创建文件后，再次调用umask()恢复umask以前的值。当PHP作为一个服务器模块运行时，它会在每次调用结尾处恢复umask的默认值。Windows在建立文件权限和所有者关系时使用不同（而且更强大）的系统，因此PHP的umask()函数（同其他与权限有关的函数一样）在Windows中是无效的。

24.1 获取并设置文件时间戳

问题

你想知道某个文件的最后访问和更新时间，或者你想修改一个文件的访问或更新时间。例如，你想让网站中的每个页面都显示它们的更新时间。

方案

函数fileatime()、filemtime()和filectime()返回一个文件的最后访问时间、更新时间和元数据修改时间，如例24-5所示。

例24-5：取得文件时间戳

```
<?php
$last_access = fileatime('larry.php');
$last_modification = filemtime('moe.php');
$last_change = filectime('curly.php');
?>
```

文件的更新时间可以通过touch()来修改。如果不带第二个参数，touch()将更新时间设置为当前日期和时间。要将文件的更新时间设置为特定的值，将表示纪元时间戳的值作为第二个参数传递给touch()即可。例24-6修改了两个文件的更新时间，但不会修改文件的内容。

例24-6：修改文件更新时间

```
<?php
touch('shemp.php');           // 将更新时间设置为现在
touch('joe.php',$timestamp);  // 将更新时间设置为 $timestamp
?>
```

讨论

fileatime()函数返回的是一个文件因为读或写被打开的最后时间。filemtime()函数返回的是一个文件内容被修改的最后时间。filectime()函数返回文件内容或元数据（例如所有者或权限）被修改的最后时间。每个函数返回的都是一个纪元时间戳。

例24-7中的代码输出了你网站中一个页面的最后更新时间。

例24-7：输出网站的最后更新时间

```
<?php
print "Last Modified: ".strftime('%c',filemtime($_SERVER['SCRIPT_FILENAME']));
?>
```

参见

fileatime()函数的文档（*http://www.php.net/fileatime*）；filemtime()函数的文档（*http://www.php. net/filemtime*）和filectime()函数的文档（*http://www.php.net/filectime*）。

24.2 获取文件信息

问题

你想读取一个文件的元数据——例如，权限和所有者关系。

方案

如例24-8所示，使用stat()函数返回一个包含文件相关信息的数组。

例24-8：获取文件信息

```
<?php
$info = stat('harpo.php');
?>
```

讨论

stat()函数返回的文件信息数组中既包含数字索引的元素，也包含字符串索引的元素。这些元素的细节如表24-3所示。

表24-3：由stat()返回的信息

数字索引	字符串索引	值
0	dev	设备
1	ino	信息节点
2	mode	权限
3	nlink	连接数
4	uid	所有者的用户ID
5	gid	组的组ID
6	rdev	信息设备的设备类型（在Windows中是-1）
7	size	大小（字节）
8	atime	最后访问时间（纪元时间戳）
9	mtime	内容最后修改时间（纪元时间戳）
10	ctime	内容或元数据的最后修改时间（纪元时间戳）
11	blksize	I/O 的块大小（在Windows中是-1）
12	blocks	分配给该文件的块数

返回数组的mode元素中包含以基于10的整数表示的权限。由于权限通常都是通过象征性的符号（如，*ls*的-rw-r--r--输出）或一个八进制整数（如0644）来表示的，所以这个值很容易被混淆。可以像例24-9所示的那样，通过base_convert()将这个权限值转换为更容易理解的八进制权限值格式。

例24-9：转换文件权限值

```
<?php
$file_info = stat('/tmp/session.txt');
```

```
$permissions = base_convert($file_info['mode'],10,8);
?>
```

在例24-9中，$permission中保存的是六位八进制数字。例如，*ls*显示有关*/tmp/session.txt*的下列信息：

```
-rw-rw-r--    1 sklar    sklar          12 Oct 23 17:55 /tmp/session.txt
```

那么$file_info['mode']是33204，$permission是100664。其中最后三位数字（664）分别表示用户（读和写）、组（读和写）和其他（读）用户对该文件的权限。第三位数字0，意味着该文件没有设置setuid位或setgid位。最左边的10意味着该文件是一个常规文件（而非一个套接字<socket>、符号连接或其他特殊的文件）。

由于stat()返回的数组中同时包含数字和字符串索引的元素，所以使用foreach来迭代一遍这个数组会得到每个值的两份拷贝。因此，可以使用一个for循环取得返回数组的元素0 12的值即可。

如果在一个符号连接上调用stat()，返回的是与该符号连接指向的文件有关的信息。如果想得到符号连接本身的信息，则要使用lstat()。

与stat()类似的是fstat()，这个函数以一个文件句柄（由fopen()或popen()返回）作为参数。

PHP的stat()函数使用的是底层的*stat(2)*系统调用，这样就产生了浪费。为了将开销降到最低，PHP会缓存调用*stat(2)*的结果。所以，如果你在一个文件上调用了stat()，在修改了这个文件的权限后，再对这个文件调用stat()时仍然会得到相同的结果。为了强制PHP重载文件的元数据，可以调用clearstatcache()，这样就会刷掉PHP缓冲的信息。PHP同样也为其他返回文件元数据的函数进行缓冲，这些函数包括：file_exists()、fileatime()、filectime()、filegroup()、fileinode()、filemtime()、fileowner()、fileperms()、filesize()、filetype()、fstat()、is_dir()、is_executable()、is_file()、is_link()、is_readable()、is_writable()和lstat()。

参见

stat()函数的文档（*http://www.php.net/stat*）；lstat()函数的文档（*http://www.php.net/lstat*），fstat()函数的文档（*http://www.php.net/fstat*）和clearstatcache()函数的文档（*http://www.php.net/clearstatcache*）。

24.3 修改文件权限或所有者

问题

你想修改一个文件的权限或所有者。例如，你想防止其他用户看到包含敏感数据的文件。

方案

使用chmod()修改文件的权限，如例24-10所示。

例24-10：修改文件权限

```
<?php
chmod('/home/user/secrets.txt',0400);
?>
```

使用chown()修改文件的所有者，使用chgrp()修改文件的组。这两个函数的使用见例24-11。

例24-11：修改文件的所有者和组

```
<?php
chown('/tmp/myfile.txt','sklar');            //指定用户名
chgrp('/home/sklar/schedule.txt','soccer'); // 指定组名

chown('/tmp/myfile.txt',5001);               // 指定用户 uid
chgrp('/home/sklar/schedule.txt',102);      // 指定组 gid
?>
```

讨论

传递给chmod()的权限必须以八进制数指定。

超级用户可以修改任何文件的权限、所有者和组。其他用户都是受限制的，他们只能修改自己文件的权限和组，但不能修改所有者。另外，非超级用户在修改文件的组时，也只能将文件修改为其所在的组。

函数chmod()、chgrp()和chown()在Windows系统中是无效的。

参见

chmod()函数的文档（*http://www.php.net/chmod*）；chown()函数的文档（*http://www.php.net/chown*）和chgrp()函数的文档（*http://www.php.net/chgrp*）。

24.4 按其组成部分分割文件名

问题

你想找到一个文件的路径和文件名。例如，你想在包含现有文件的同一目录中创建文件。

方案

使用basename()取得文件名，dirname()取得路径，如例24-12所示。

例24-12：获取路径中的各个部分

```
<?php
$full_name = '/usr/local/php/php.ini';
$base = basename($full_name);  // $base 是 "php.ini"
$dir  = dirname($full_name);   // $dir 是 "/usr/local/php"
?>
```

使用pathinfo()取得目录名、文件名（base name）和扩展名并保存到一个关联数组中，如例24-13所示。

例24-13：获取路径组件及文件扩展名

```
<?php
$info = pathinfo('/usr/local/php/php.ini');
// $info['dirname'] 中是 "/usr/local/php"
// $info['basename'] 中是 "php.ini"
// $info['extension'] 中是 "ini"
?>
```

讨论

要在包含现有文件的同一目录中创建一个文件，可以使用dirname()找到目录，并将该目录传递给tempnam()，这个过程见例24-14。

例24-14：在特定的位置创建一个临时文件

```
<?php
$dir = dirname($existing_file);
$temp = tempnam($dir,'temp');
$temp_fh = fopen($temp,'w');
?>
```

dirname()函数在与特殊的常量__FILE__组合使用时特别有用，这一常量包含着当前文件的完整路径名，这个路径名与当前执行的PHP脚本不同。如果*/usr/local/alice.php*中包含*/usr/local/bob.php*，那么bob.php中的__FILE__就是*/usr/local/bob.php*。当我们想在同一

目录中包含（include）或请求（require）脚本作为一个特定文件，但不知道目录路径，而且该目录也不必出现在包含路径中时__FILE__的这种行为特别有用。例24-15所示为该过程。

例24-15：包含与当前文件相关的文件

```
<?php
$currentDir = dirname(__FILE__);
include $currentDir . '/functions.php';
include $currentDir . '/classes.php';
?>
```

如果例24-15中代码包含在目录*/usr/local*下，那么它会包含*/usr/local/functions.php*和*/usr/local/classes.php*。在你向其他用途分配代码时这一技术尤其有用。通过使用这一技术，就不再需要为了使分发后的代码正确运行而必须进行配置或修改包含路径了。

之所以使用诸如basename()、dirname()和pathinfo()这样的函数比简单地根据/来分割完整的文件名具有更高的可移植性，因为这些函数使用的是操作系统认可的分隔符。即，在Windows系统中，这些函数会将/和\都看成是文件和目录分隔符，而在其他平台上，则只会使用/。

PHP没有提供能够将basenaem()、dirname()和pathinfo()返回的组件组合成一个完整文件名的内置函数。如果你想实现这一功能，就必须将这些组件通过和内置的DIRECTORY_SEPARATOR常量（在Unix中是/，在Windows中是\）进行组合。

参见

basename()函数的文档（*http://www.php.net/basename*）；dirname()函数的文档（*http://www.php. net/dirname*）；pathinfo()函数的文档（*http://www.php.net/pathinfo*）和__FILE__常量的文档（*http://www.php.net/language.constants.predefined*）。

24.5 删除文件

问题

你想删除一个文件。

方案

使用unlink()，如例24-16所示。

例24-16：删除文件

```
<?php
$file = '/tmp/junk.txt';
unlink($file) or die ("can't delete $file: $php_errormsg");
?>
```

讨论

unlink()函数只能删除PHP进程用户能够删除的文件。如果你在使用unlink()时遇到了问题，可能需要检查一下操作文件的权限以及运行PHP脚本的方式。

参见

unlink()函数的文档（*http://www.php.net/unlink*）。

24.6 复制或移动文件

问题

你想复制或移动一个文件。

方案

使用copy()复制文件，如例24-17所示。

例24-17：复制文件

```
<?php
$old = '/tmp/yesterday.txt';
$new = '/tmp/today.txt';
copy($old,$new) or die("couldn't copy $old to $new: $php_errormsg");
?>
```

使用rename()移动文件，如例24-18所示。

例24-18：移动文件

```
<?php
$old = '/tmp/today.txt';
$new = '/tmp/tomorrow.txt';
rename($old,$new) or die("couldn't move $old to $new: $php_errormsg");
?>
```

讨论

在Unix系统中，如果使用的是PHP 4.3.3之前的版本，rename()不能跨文件系统移动文件。为了解决这个问题，需要先把文件复制到新位置，然后再删除原文件，这个过程如例24-19所示。

例24-19：跨文件系统移动文件

```
<?php
if (copy("/tmp/code.c","/usr/local/src/code.c")) {
  unlink("/tmp/code.c");
}
?>
```

如果要复制或移动多个文件，可以通过循环来调用copy()或rename()，而且在调用这两个函数时，每次只能对一个文件进行操作。

参见

copy()函数的文档（*http://www.php.net/copy*）和rename()函数的文档（*http://www.php.net/rename*）。

24.7 处理同一目录中的所有文件

问题

你想对一个目录中的所有文件进行处理。例如，你想在表单中创建一个列出目录中所有文件的<select />下拉选择框。

方案

使用DirectoryIterator类来取得目录中的每一个文件，如例24-20所示。

例24-20：处理一个目录中的所有文件

```
<?php
echo "<select name='file'>\n";
foreach (new DirectoryIterator('/usr/local/images') as $file) {
    echo '<option>' . htmlentities($file) . "</option>\n";
}
echo '</select>';
?>
```

讨论

DirectoryIterator会为目录中的每一个元素生成一个值。这个值是一个带有许多便捷特征的对象。而该对象的字符串表现形式就是目录元素的文件名（没有前面的路径）。例如，如果*/usr/local/images*包含文件*cucumber.gif*和*eggplant.png*，例24-20就会输出如下结果：

```
<select name='file'>
<option>.</option>
<option>..</option>
<option>cucumber.gif</option>
<option>eggplant.png</option>
</select>
```

由此可见，DirectoryIterator会为所有目录元素生成一个对象，包含.（当前目录）和..（父目录）。幸运的是，这个对象带有能够帮助我们识别它们的一些方法。其中，当对象是.或..时，isDot()返回true。例24-21使用isDot()避免了这两个元素出现在输出结果中。

例24-21：从输出中删除.和..

```
<?php
echo "<select name='file'>\n";
foreach (new DirectoryIterator('/usr/local/images') as $file) {
    if (! $file->isDot()) {
        echo '<option>' . htmlentities($file) . "</option>\n";
    }
}
echo '</select>';
?>
```

表24-4列出了DirectoryIterator生成对象的其他方法。

表24-4：获取DirectoryIterator对象信息的方法

方法名	返回值	示例
isDir()	该元素是否是一个目录	false
isDot()	该元素是否是.或者..	false
isFile()	该元素是否是一个常规文件	true
isLink()	该元素是否是一个连接	false
isReadable()	该元素是否可读	true
isWritable()	该元素是否可写	true
isExecutable()	该元素是否可执行	false
getATime()	访问该元素的最后时间	1144509622

表24-4：获取DirectoryIterator对象信息的方法（续）

方法名	返回值	示例
getCTime()	该元素的创建时间	1144509600
getMTime()	该元素的最后修改时间	1144509620
getFilename()	该元素的文件名（没有前导路径）	eggplant.png
getPathname()	该元素的完整路径	/usr/local/images/eggplant.php
getPath()	该元素的前导路径	/usr/local/images
getGroup()	该元素的组ID	500
getOwner()	该元素的所有者ID	1000
getPerms()	以八进制数表示的该元素的权限	16895
getSize()	该元素的大小	328742
getType()	该元素的类型（dir、file、link等）	file
getInode()	该元素的信息节点数	28720

表24-4中函数所报告的数据与表24-1中函数报告的数据同样都来自于对底层系统的调用，所以有关Unix和Windows系统中存在差别的警告仍然适用。

参见

DirectoryIterator类的文档（*http://www.php.net/~helly/php/ext/spl/classDirectoryIterator.html*）。

24.8 生成与模式匹配的文件名列表

问题

你想寻找文件名与一个模式匹配的所有文件。

方案

使用DirectoryIterator类的FilterIterator子类。这个FilterIterator子类需要通过自己的accept()方法来决定是否接受一个特定的值。例24-22中的代码只接受扩展名为常见图像的文件名。

例24-22：使用FilterIterator子类

```
<?php
class ImageFilter extends FilterIterator {
    public function accept() {
        return preg_match('@\.(gif|jpe?g|png)$@i',$this->current());
    }
}
foreach (new ImageFilter(new DirectoryIterator('/usr/local/images')) as $img) {
    print "<img src='".htmlentities($img)."'/>\n";
}
?>
```

讨论

FilterIterator子类封装了一个DirectoryIterator类的技巧，并且只允许某些有资格的元素出现。它根据accept()方法返回true或false来决定某个特定元素（通过$this->current()访问到）是否具备资格。在例24-22中，accept()使用了一个正则表达式进行筛选，但你在自己的代码中可以使用其他逻辑。

如果你的模式能通过一个简单的壳（shell）“glob”（例如，*.*）来表示，那么可以使用glob()函数获取匹配的文件名。例24-23在一个特定的目录中查找所有文本文件。

例24-23：使用glob()

```
<?php
foreach (glob('/usr/local/docs/*.txt') as $file) {
   $contents = file_get_contents($file);
   print "$file contains $contents\n";
}
?>
```

glob()函数会返回一个匹配完整路径名的数组。如果没有与模式匹配的文件，glob()返回false。

参见

技巧24.9中递归地迭代一个目录中所有文件的详细介绍；FilterIterator类的文档（*http://www.php.net/~ helly/php/ext/spl/classFilterIterator.html*）和glob()函数的文档（*http://www.php.net/glob*）；有关壳模式匹配的信息（*http://www.gnu.org/software/bash/manual/bashref.html#SEC35*）。

24.9 递归地处理同一目录中的所有文件

问题

你想对一个目录及其所属子目录中的全部文件逐一执行某些操作。例如，你想知道一个目录中的所有文件占用了多大的磁盘空间。

方案

使用RecursiveDirectoryIterator和RecursiveIteratorIterator。其中，RecursiveDirectoryIterator扩展了DirectoryIterator，增加了能够访问子目录中元素的getChildren()方法。RecursiveIteratorIterator则用于对RecursiveDirectoryIterator返回到列表中的层次结构进行平整（flattens）处理。例24-24计算了一个目录中文件的总体大小。

例24-24：递归地处理一个目录中的所有文件

```
<?php
$dir = new RecursiveDirectoryIterator('/usr/local');
$totalSize = 0;
foreach (new RecursiveIteratorIterator($dir) as $file) {
    $totalSize += $file->getSize();
}
print "The total size is $totalSize.\n";
?>
```

讨论

RecursiveDirectoryIterator生成的（传递给RecursiveIteratorIterator的）对象与通过DirectoryIterator得到的对象相同，因此表24-4中列出的所有方法对这个对象都是有效的。

参见

RecursiveDirectoryIterator类的文档（*http://www.php.net/~helly/php/ext/spl/classRecursiveDirectoryIterator.html*）和RecursiveIteratorIterator类的文档（*http://www.php.net/~helly/php/ext/spl/classRecursiveIteratorIterator.html*）。

24.10 创建新目录

问题

你想创建一个新目录。

方案

使用mkdir()，如例24-25所示。

例24-25：创建新目录

```
<?php
mkdir('/tmp/apples',0777) or die($php_errormsg);
?>
```

讨论

mkdir()的第二个参数是新目录的权限模式，该参数必须是一个八进制数。这个权限值会通过减掉当前的umask从而得到新创建目录的权限。也就是说，如果当前umask是0002，调用mkdir('/tmp/apples',0777)会将创建目录的权限设置为0775（用户和组可以读取、写入和执行，其他人只能读取和执行）。

默认情况下，mkdir()只在存在父目录的时候才会创建目录。例如，如果*/usr/local/images*不存在，就不能创建*/usr/local/images/puppies*。如果想同时创建目录及其父目录，可以将true作为第三个参数传递给mkdir()。这样该函数就可以通过类似递归的方式创建任何缺少的目录。

参见

mkdir()函数的文档（*http://www.php.net/mkdir*）。

24.11 删除目录及其内容

问题

你想删除一个目录及其中的所有内容，包括子目录和子目录中的内容。

方案

使用RecursiveDirectoryIterator和RecursiveIteratorIterator，并在其父目录之前指定应该列出的子元素（文件和子目录），如例24-26所示。

例24-26：删除一个目录

```
<?php
function obliterate_directory($dir) {
    $iter = new RecursiveDirectoryIterator($dir);
    foreach (new RecursiveIteratorIterator($iter, RecursiveIteratorIterator::
CHILD_FIRST) as $f) {
        if ($f->isDir()) {
            rmdir($f->getPathname());
        } else {
            unlink($f->getPathname());
        }
    }
    rmdir($dir);
}

obliterate_directory('/tmp/junk');
?>
```

讨论

很显然，删除文件是一种有风险的操作。由于PHP内置的目录移除函数 —— rmdir()，只能删除空目录，而unlink()则不能接受壳（shell）通配符，所以必须通过CHILD_FIRST常量告诉RecursiveIteratorIterator在父目录之前提供子元素。

但是，该常量在PHP 5.1以前是无效的。如果你使用的是PHP的早期版本，可以使用例24-27中的函数达到同样目的。

例24-27：在不通过RecursiveIteratorIterator的情况下删除一个目录

```
<?php
function obliterate_directory($dir) {
    foreach (new DirectoryIterator($dir) as $file) {
        if ($file->isDir()) {
            if (! $file->isDot()) {
                obliterate_directory($file->getPathname());
            }
        } else {
            unlink($file->getPathname());
        }
    }
    rmdir($dir);
}
?>
```

参见

rmdir()函数的文档（*http://www.php.net/rmdir*）和RecursiveIteratorIterator类的文档（*http://www.php.net/~helly/php/ext/spl/classRecursiveIteratorIterator.html*）。

24.12 编程：Web服务器目录列表

例24-28（本技巧后半部分）中的*web-ls.php*程序提供了一种展示你Web服务器根目录中所有文件视图的方式，并对结果进行了与Unix的*ls*命令相同的格式化处理。其中的文件名都加上了链接以便于下载每个文件，而目录名也都加了链接以便能够在每个目录中浏览，具体效果如图24-1所示。

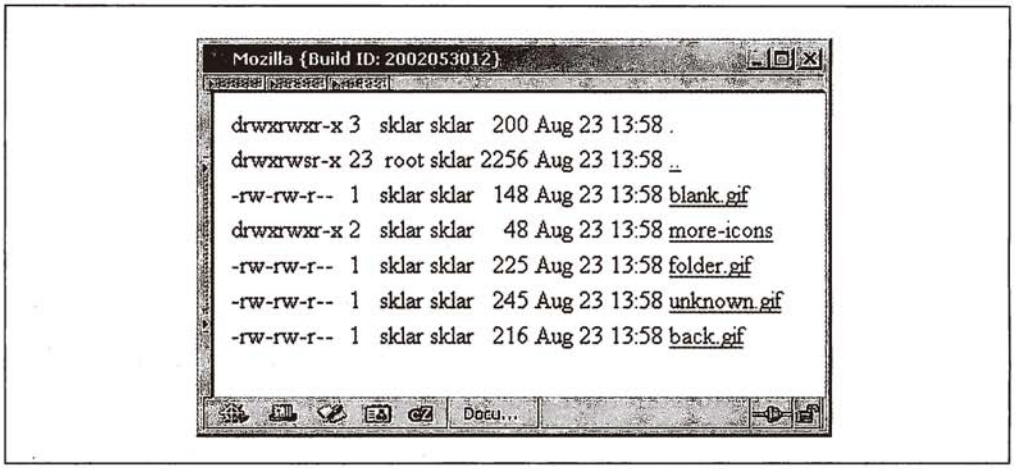

图24-1：Web目录列表

例24-28中的多数代码行都致力于以一种容易阅读的方式来表现文件的权限，而该程序的心脏部分则是结尾处的foreach循环。DirectoryIterator会为目录中的每个项目生成一个对象。该对象的几个方法提供了与文件相关的信息，最后通过printf()输出经过格式化的文件信息。

其中的mode_string()函数及其使用的常量用于将一个文件的模式（例如，35316）由八进制形式转换为容易认读的字符串形式（例如，-rwsrw-r--）。

例24-28：web-ls.php

```
<?php

/* 用于确定文件权限及类型的位屏蔽（Bit masks），下面列出的
 * 名字和值是与POSIX兼容的，个别系统可能会有自己的
```

```
 * 扩展名。
 */

define('S_IFMT',0170000);   // mask for all types
define('S_IFSOCK',0140000); // type: socket
define('S_IFLNK',0120000);  // type: symbolic link
define('S_IFREG',0100000);  // type: regular file
define('S_IFBLK',0060000);  // type: block device
define('S_IFDIR',0040000);  // type: directory
define('S_IFCHR',0020000);  // type: character device
define('S_IFiFO',0010000);  // type: fifo
define('S_ISUID',0004000);  // set-uid bit
define('S_ISGID',0002000);  // set-gid bit
define('S_ISVTX',0001000);  // sticky bit
define('S_IRWXU',00700);    // mask for owner permissions
define('S_IRUSR',00400);    // owner: read permission
define('S_IWUSR',00200);    // owner: write permission
define('S_IXUSR',00100);    // owner: execute permission
define('S_IRWXG',00070);    // mask for group permissions
define('S_IRGRP',00040);    // group: read permission
define('S_IWGRP',00020);    // group: write permission
define('S_IXGRP',00010);    // group: execute permission
define('S_IRWXO',00007);    // mask for others permissions
define('S_IROTH',00004);    // others: read permission
define('S_IWOTH',00002);    // others: write permission
define('S_IXOTH',00001);    // others: execute permission

/* mode_string() 是一个辅助函数。它将一个表示文件类型
 * 和权限的八进制模式转换成一个对应的、由10个字符组成
 * 的字符串形式。它是GNU文件实用程序包 (GNU fileutils package)
 * mode_string()函数的PHP版
 */
$mode_type_map = array(S_IFBLK => 'b', S_IFCHR => 'c',
                       S_IFDIR => 'd', S_IFREG => '-',
                       S_IFiFO => 'p', S_IFLNK => 'l',
                       S_IFSOCK => 's');
function mode_string($mode) {
    global $mode_type_map;
    $s = '';
    $mode_type = $mode & S_IFMT;
    // 添加类型字符
    $s .= isset($mode_type_map[$mode_type]) ?
          $mode_type_map[$mode_type] : '?';

  // 设置用户权限
  $s .= $mode & S_IRUSR ? 'r' : '-';
  $s .= $mode & S_IWUSR ? 'w' : '-';
  $s .= $mode & S_IXUSR ? 'x' : '-';

  // 设置组权限
  $s .= $mode & S_IRGRP ? 'r' : '-';
  $s .= $mode & S_IWGRP ? 'w' : '-';
  $s .= $mode & S_IXGRP ? 'x' : '-';
```

```
    // 设置其他权限
    $s .= $mode & S_IROTH ? 'r' : '-';
    $s .= $mode & S_IWOTH ? 'w' : '-';
    $s .= $mode & S_IXOTH ? 'x' : '-';

    // 针对 set-uid, set-gid 和 sticky 调整执行字母
    if ($mode & S_ISUID) {
        // 'S' 表示 set-uid 但不能被所有者执行
        $s[3] = ($s[3] == 'x') ? 's' : 'S';
    }

    if ($mode & S_ISGID) {
        // 'S' 表示 set-gid 但不能被组用户执行
        $s[6] = ($s[6] == 'x') ? 's' : 'S';
    }

    if ($mode & S_ISVTX) {
        // 'T' 表示 sticky 但不能被其他用户执行
        $s[9] = ($s[9] == 'x') ? 't' : 'T';
    }

    return $s;
}

// 如果未指定则从文档根目录开始
$dir = isset($_GET['dir']) ? $_GET['dir'] : '';

// 在文件系统中查找 $dir
$real_dir = realpath($_SERVER['DOCUMENT_ROOT'].$dir);
// 根据真实路径解决正斜杠和反斜杠问题
// 以便传递文档根目录
$real_docroot = realpath($_SERVER['DOCUMENT_ROOT']);

// 确保 $real_dir is 位于文档根目录中
if (! (($real_dir == $real_docroot) ||
       ((strlen($real_dir) > strlen($real_docroot)) &&
        (strncasecmp($real_dir,$real_docroot.DIRECTORY_SEPARATOR,
        strlen($real_docroot.DIRECTORY_SEPARATOR)) == 0)))) {
    die("$dir is not inside the document root");
}

// 通过删除开始处的文档根目录规范化 $dir
$dir = substr($real_dir,strlen($real_docroot)+1);

// 是否打开了一个目录?
if (! is_dir($real_dir)) {
    die("$real_dir is not a directory");
}

print '<pre><table>';

// 读取目录中的每个条目
foreach (new DirectoryIterator($real_dir) as $file) {
    // 将uid转换成用户名
    if (function_exists('posix_getpwuid')) {
```

```
    $user_info = posix_getpwuid($file->getOwner());
} else {
    $user_info = $file->getOwner();
}

// 将gid转换成组名
if (function_exists('posix_getgrid')) {
    $group_info = $file->getGroup();
} else {
    $group_info = $file->getGroup();
}

// 为保证可读性对日期进行格式化
$date = date('M d H:i',$file->getMTime());

// 将八进制模式转换成易认读的字符串形式
$mode = mode_string($file->getPerms());

$mode_type = substr($mode,0,1);
if (($mode_type == 'c') || ($mode_type == 'b')) {
    /* if it's a block or character device, print out the major and
     * minor device type instead of the file size */
    $statInfo = lstat($file->getPathname());
    $major = ($statInfo['rdev'] >> 8) & 0xff;
    $minor = $statInfo['rdev'] & 0xff;
    $size = sprintf('%3u, %3u',$major,$minor);
} else {
    $size = $file->getSize();
}

// 为文件名两侧添加 <a href="">
// 当前目录不加链接
if ('.' == $file->getFilename()) {
    $href = $file->getFilename();
} else {
    // 在父目录链接中不包含 ".."
    if ('..' == $file->getFilename()) {
        $href = urlencode(dirname($dir));
    } else {
        $href = urlencode($dir) . '/' . urlencode($file);
    }

    /* 除了 "/" 之外的所有字符都应该进行url编码 */
    $href = str_replace('%2F','/',$href);

    // 通过web-ls浏览其他目录
    if ($file->isDir()) {
        $href = sprintf('<a href="%s?dir=/%s">%s</a>',
                        $_SERVER['PHP_SELF'],$href,$file);
    } else {
        // 链接文件以便下载
        $href= sprintf('<a href="%s">%s</a>',$href,$file);
    }

    // 如果是一个链接，同时显示链接目标提示
```

```
        if ('l' == $mode_type) {
            $href .= ' -&gt; ' . readlink($file->getPathname());
        }
    }

    // 输出文件信息
    printf('<tr><td>%s</td><td align="right">%s</td>
            <td align="right">%s</td><td align="right">%s</td>
            <td align="right">%s</td><td>%s</td></tr>',
           $mode,                // 格式化的模式字符串
           $user_info['name'],   // 所有者的用户名
           $group_info['name'],  // 组名称
           $size,                // 文件大小（或设备号）
           $date,                // 最后更新日期和时间
           $href);               //浏览或者供下载的链接
}

print '</table></pre>';
?>
```

24.13 编程：网站搜索

可以将例24-29中的*site-search.php*程序作为基于文件的中小型规模网站的搜索引擎。

例24-29：site-search.php

```
<?php
class SiteSearch {
    public $bodyRegex = '';
    protected $seen = array();


    public function searchDir($dir) {
        // 保存匹配页面的数组
        $pages = array();

        // 保存执行递归操作的目录的数组
        $dirs = array();

        // 将目录标记为已经查看过以保证不再重复查看
        $this->seen[realpath($dir)] = true;

        try {
            foreach (new RecursiveIteratorIterator(
            new RecursiveDirectoryIterator($dir)) as $file) {
                if ($file->isFile() && $file->isReadable() &&
                (! isset($this->seen[$file->getPathname()]))) {
                    // 将其标记为已经查看过
                    // 以便返回时跳过
                    $this->seen[$file->getPathname()] = true;

                    // 将文件内容加载到$text
                    $text = file_get_contents($file->getPathname());
```

```
                    // 如果搜索项位于主体（body）定界符内
                    if (preg_match($this->bodyRegex,$text)) {

                    // 通过从完整路径中删除文档根目录
                    // 为文件建立相对URI
                    $uri = substr_replace($file->getPathname(),'',0,strlen
                    ($_SERVER['DOCUMENT_ROOT']));

                    // 如果页面中有标题，找到它
                    if (preg_match('#<title>(.*?)</title>#Sis',$text,$match)) {
                        // 并将标题和URI添加到 $pages
                        array_push($pages,array($uri,$match[1]));
                    } else {
                        // 否则，使用URI作为标题
                        array_push($pages,array($uri,$uri));
                    }
                }
                }
            }
        } catch (Exception $e) {
            // 在打开目录时出现了问题
        }
        return $pages;
    }
}

// 根据匹配页面的标题按字母顺序进行排序的辅助函数
function by_title($a,$b) {
        return ($a[1] == $b[1]) ?
               strcmp($a[0],$b[0]) :
               ($a[1] > $b[1]);
}

// SiteSearch 对象执行搜索
$search = new SiteSearch();

// 保存与搜索项目匹配的页面的数据
$matching_pages = array();
// 要搜索的文档根目录下面的目录
$search_dirs = array('sports','movies','food');
// 用于搜索文件的正则表达式。"S"模式修饰符
// 告诉PCRE引擎为提高效率
// "学习" regex
$search->bodyRegex = '#<body>(.*' . preg_quote($_REQUEST['term'],'#').
                     '.*)</body>#Sis';

// 将每个目录中匹配的文件添加$matching页面中
foreach ($search_dirs as $dir) {
    $matching_pages = array_merge($matching_pages,
                                  $search->searchDir($_SERVER['DOCUMENT_ROOT'].'/'.$dir));
}

if (count($matching_pages)) {
    // 按照标题对匹配页面排序
```

```
    usort($matching_pages,'by_title');
    print '<ul>';
    // 输出带有指向该页面链接的标题
    foreach ($matching_pages as $k => $v) {
        print sprintf('<li> <a href="%s">%s</a>',$v[0],$v[1]);
    }
    print '</ul>';
} else {
    print 'No pages found.';
}
?>
```

这个程序用于按照搜索项（$_REQUEST['term']）在文档根目录下指定的目录集合中搜索所有文件。搜索的目标目录集合由$search_dirs指定。该程序也会递归地搜索相应的子目录并跟踪符号连接，但也为了避免死循环保持了已经搜索过的文件和目录的记录。

如果找到包含搜索项的一些页面，程序就会输出包含指向这些页面的链接列表，而且列表会按照页面标题的字母顺序进行排序。如果某个页面中不包含标题（<title>和</title>标签之间的内容），则会使用该页面文档根目录的相对URI。

程序搜索的目标是每个文件中位于<body>和</body>标签之间的内容。如果在你页面的<body>标签内部有许多你想排除在搜索之外的文本内容，那么可以将应该搜索的文本通过特殊的HTML注释包围起来，然后修改$body_regex来查找这些标签。最终你的页面可能与例24-30中所示的页面有一些类似。

例24-30：HTML页面示例

```
<body>

// 菜单、头部（页眉）等的HTML代码

<!-- search-start -->

<h1>Aliens Invade Earth</h1>

<h3>by H.G. Wells</h3>

<p>Aliens invaded earth today. Uh Oh.</p>

// 更多内容

<!-- search-end -->

// 页脚等元素的HTML代码

</body>
```

如果只想基于HTML注释中的标题、作者和内容来搜索匹配项，需要像例24-31所示的那样修改$search->bodyRegex变量。

例24-31：对应的正则表达式

```
$search->bodyRegex = '#<!-- search-start -->(.*' . preg_quote($_REQUEST['term'],'#').
            '.*)<!-- search-end -->#Sis';
```

如果你不想让搜索项匹配页面中HTML或PHP标签中的文本，可以如例24-32所示，在载入要搜索文件内容的代码中添加一个对strip_tags()的调用。

例24-32：去掉HTML和PHP标签

```
// 将文件内容载入到 $text
$text= strip_tags(file_get_contents($file->getPathname()));
```

第25章

命令行PHP

25.0 概述

PHP是为Web编程而创建的，而且一直也主要用于这一目的。不过，PHP其实也可以当作通用的脚本语言来使用。比如，当你需要与Web应用程序共享代码时，PHP用作命令行中的脚本就特别有用。如果你的网站中有一个论坛，你可能想每隔几分钟或几小时运行一次扫描新帖的程序，并在发现新帖中包含某些关键字时提醒你。如果用PHP来编写这个扫描程序就能够实现与你的主论坛应用程序共享相关的论坛代码。这样做不仅可以节省你的时间，而且还有助于避免维护开销的膨胀。

从4.3版开始，PHP安装时就会包含一个命令行界面（command-line interface，CLI）程序。这个二进制CLI程序类似于Web服务器模块和二进制的CGI，但是与它们有着某些重要的区别，CLI显得与shell更加友好。在CLI中，一些配置指令设置了硬编码的值；例如，html_errors指令设置为false，而implicit_flush设置为true。max_execution_time指令设置为0，是为了不限制程序运行的时间。最后，register_argc_argv也设置为true。这就意味着你能够通过$argv和$argc这两个数组找到参数信息，而不必使用$_SERVER['argv']和$_SERVER['argc']了。有关处理参数的问题将在技巧25.1和25.2中讨论。

在运行脚本时，需要将脚本文件名称作为参数传递：

```
% php scan-discussions.php
```

在Unix系统中，也可以通过在脚本的顶部使用“hash-bang”语法来自动地运行PHP解释程序。如果PHP的二进制程序位于*/usr/local/bin*，那么脚本的第一行就要写成下面这样：

```
#!/usr/local/bin/php
```

然后，只要你拥有执行该文件的权限，就可以通过在命令行中键入脚本的名称来运行该脚本了。

如果你想将类和函数同时用于Web程序和命令行，那么抽象的代码就必须反映这两种环境下的差别。比如，分别采用HTML和纯文本输出，或者是否有权使用Web服务器设置环境变量等。一个有用的策略就是检测php_sapi_name()返回的值是否为cli。根据这一检测，就可以像下面这样分流脚本的行为：

```
if ('cli' == php_sapi_name()) {
  print "Database error: ".mysql_error()."\n";
} else {
  print "Database error.<br/>";
  error_log(mysql_error());
}
```

这里的代码不仅基于它所在的运行环境调整了输出格式（\n与
），而且也改变了信息的去向。在命令行中，运行程序的人最好能够看到直接来自MySQL的错误消息，而在Web程序中，则没有必要让用户看到可能敏感的数据。相反，代码会输出一个普通的错误消息，并将详细的消息保存到服务器的错误日志中，以便专门查阅。

在命令行中，通过使用-d标记，可以在不修改*php.ini*文件的基础上指定自定义的INI选项。例如，下面是如何启用输出缓冲的命令：

```
% php -d output_buffering=1 scan-discussions.php
```

CLI二进制程序也接受-r参数。当这个参数后面是不带<?php和?>标签的某些PHP代码时，CLI二进制程序就会运行相应的代码。例如，下面是输出当前时间的命令：

```
% php -r 'print strftime("%c");'
```

如果想查看CLI二进制程序的完整选项列表，使用-h命令：

```
% php -h
```

最后，CLI二进制程序将针对标准I/O流的句柄定义为常量STDIN、STDOUT和STDERR。你可以在fopen()中直接使用这些常量而不用自己再创建文件句柄了。

```
// 从标准输出读取
$input = fgets(STDIN,1024);

// 写入到标准输出
fwrite(STDOUT,$jokebook);

// 写入到标准错误
fwrite(STDERR,$error_code);
```

25.1 解析程序参数

问题

你想要处理在命令行中传递的参数。

方案

在$argc中查找参数的数量，在$argv中查找参数的值。第一个参数，$argv[0]，保存的是运行脚本的名称：

```
if ($argc != 2) {
    die("Wrong number of arguments: I expect only 1.");
}

$size = filesize($argv[1]);

print "I am $argv[0] and report that the size of ";
print "$argv[1] is $size bytes.";
```

讨论

为了基于由命令行传递标记来设置选项，可以如例25-1中所示的那样从1到$argc循环遍历$argv。

例25-1：解析命令行参数

```
<?php
for ($i = 1; $i < $argc; $i++) {
    switch ($argv[$i]) {
    case '-v':
        // 设置标记
        $verbose = true;
        break;
    case '-c':
        // 前进到下一个参数
        $i++;
        // 如果已经设置，保存相应的值
        if (isset($argv[$i])) {
            $config_file = $argv[$i];
        } else {
            // 如果没有指定文件名称，则退出
            die("Must specify a filename after -c");
        }
        break;
    case '-q':
        $quiet = true;
        break;
```

```
        default:
            die('Unknown argument: '.$argv[$i]);
            break;
        }
    }
    ?>
```

在这个例子中，参数-v和-q是设置$verbose和$quiet的标记，而参数-c后面需要跟一个字符串。该字符串会被指定给$config_file。

参见

技巧25.2中更多通过*getopt*解析参数的内容；变量$argc和$argv的文档（*http://www.php.net/reserved.variables*）。

25.2 通过getopt解析程序参数

问题

你想要解析程序的选项，这些选项可能会以或短或长的形式，甚至组合的形式指定。

方案

使用PEAR的Console_Getopt类。该类的getopt()方法既可以解析-a或-b这样的短结构选项，也可以解析--alice或--bob这样的长结构选项：

```
$o = new Console_Getopt;

// 接受 -a、 -b 和 -c
$opts = $o->getopt($argv,'abc');

// 接受 --alice 和 --bob
$opts = $o->getopt($argv,'',array('alice','bob'));
```

讨论

要解析短结构选项，将命令行参数数组和一个用于指定有效选项的字符串传递给Console_Getopt::getopt()。下面的例子接受-a、 -b或-c这样的参数，可以单独出现，也可以成组出现：

```
$o = new Console_Getopt;
$opts = $o->getopt($argv,'abc');
```

对于前面的选项字符串abc，下面都是能够接受的选项：

```
% program.php -a -b -c
% program.php -abc
% program.php -ab -c
```

getopt()方法返回一个数组。该数组中元素的第一个元素是一个列表，其中保存着在命令行中指定的所有解析的选项以及它们的值。第二个元素中保存的是在命令行中指定的、传递给getopt()的不符合规范的选项。例如，如果前面的程序运行如下：

```
% program.php -a -b sneeze
```

那么$opts中的元素及值如下：

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => a
                    [1] =>
                )
            [1] => Array
                (
                    [0] => b
                    [1] =>
                )
        )
    [1] => Array
        (
            [0] => program.php
            [1] => sneeze
        )
)
```

在规范字符串中的选项后面加一个冒号表示该选项需要一个值，加两个冒号则表示值是可选的。因此ab:c::的含义就是a不能有值，b必须有值，而c在指定的情况下可以有值。如果使用这个规范字符串，那么通过以下命令再次调用相应程序：

```
% program.php -a -b sneeze
```

得到的$opts如下：

```
Array
(
    [0] => Array
        (
            [0] => Array
```

```
            (
                [0] => a
                [1] =>
            )
        [1] => Array
            (
                [0] => b
                [1] => sneeze
            )
    )
[1] => Array
    (
        [0] => program.php
    )
)
```

由于这次sneeze被看成了b的值，所以不再出现在数组未解析的选项中。注意，包含未解析选项的数组中总会包含程序的名称。

要解析长结构选项参数，需要给getopt()传递一个描述你想得到的参数的数组。将每个参数放到一个数组元素（没有前置的--）中并通过在后面加“=”标明是强制参数，或者加“==”标明是一个可选的参数。这个数组是getopt()的第三个参数。第二个参数（描述短结构参数的字符串）根据你想不想同时解析短结构参数可以留空，也可以不留空。下面这个例子接受一个没有值的debug参数，一个强制必须有值的name参数和一个值可选的size参数：

```
require 'Console/Getopt.php';
$o = new Console_Getopt;
$opts = $o->getopt($argv,'',array('debug','name=','size=='));
```

以下命令都是运行这个程序的有效命令：

```
% program.php --debug
% program.php --name=Susannah
% program.php --name Susannah
% program.php --debug --size
% program.php --size=56 --name=Susannah
% program.php --name --debug
```

之所以最后一条命令也是合法的（即使达到预期的效果），是因为它将--debug看成了name参数的值，而并没有认为设置了debug参数。命令行中的值与参数之间只能通过“=”或空格来分隔。

对于长结构的参数，getopt()会在解析返回的数组中包含前导的--。例如，当运行如下命令时：

```
% program.php --debug --name=Susannah
```

$opts会被设置为：

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => --debug
                    [1] =>
                )
            [1] => Array
                (
                    [0] => --name
                    [1] => Susannah
                )
        )
    [1] => Array
        (
            [0] => program.php
        )
)
```

这个代码使用$argv作为命令行参数的数组，这在默认情况下不会有问题。Console_Getopt提供了一个名为readPHPArgv()的方法，用于在$argv和$HTTP_SERVER_VARS['argv']中查找命令行参数。使用这个方法时，需要将该方法的调用结果传递给getopt()：

```
require 'Console/Getopt.php';
$o = new Console_Getopt;
$opts = $o->getopt($o->readPHPArgv(),'',array('debug','name=','size=='));
```

当遇到错误发生时 —— 例如没有指定必要的参数 —— getopt和readPHPArgv()都会返回一个Getopt_Error对象。Getopt_Error扩展了PEAR_Error基类，所以你可以使用熟悉的方法来处理错误：

```
require 'Console/Getopt.php';
$o = new Console_Getopt;
$opts = $o->getopt($o->readPHPArgv(),'',array('debug','name=','size=='));

if (PEAR::isError($opts)) {
    print $opts->getMessage();
} else {
    // 处理选项
}
```

参见

技巧25.1中不通过*getopt*解析程序选项的内容；Console_Getopt类的文档（*http://pear.php.net/manual/en/core.console.getopt.php*）。

25.3 读取键盘输入

问题

你需要读取某些用户输入的内容。

方案

使用fopen()，以特殊的文件名“php://stdin:”作为参数：

```
print "Type your message. Type '.' on a line by itself when you're done.\n";

$fh = fopen('php://stdin','r') or die($php_errormsg);
$last_line = false;  $message = '';
while (! $last_line) {
    $next_line = fgets($fp,1024);
    if (".\n" == $next_line) {
      $last_line = true;
    } else {
      $message .= $next_line;
    }
}

print "\nYour message is:\n$message\n";
```

如果安装了Readline扩展，也可以使用readline()：

```
$last_line = false; $message = '';
while (! $last_line) {
    $next_line = readline();
    if ('.' == $next_line) {
        $last_line = true;
    } else {
        $message .= $next_line."\n";
    }
}

print "\nYour message is:\n$message\n";
```

讨论

只要通过fopen()取得了指向标准输入设备（stdin）的文件句柄，就可以使用所有标准的文件读取函数（fread()、fgets()等）来处理输入的内容。本方案中使用的是fgets()，该函数一次返回一行输入。如果使用fread()，那么输入仍然还需要以换行符结尾，以便fread()能够返回。例如，如果运行了如下代码：

```
$fh = fopen('php://stdin','r') or die($php_errormsg);
$msg = fread($fh,4);
print "[$msg]";
```

并键入了tomato和一个换行符，那么输出将是[toma]。同指定的一样，fread()只从标准输入设备中读取了四个字符。但即使如此，仍然需要一个换行符作为从等待键盘输入状态中返回的信号。

Readline扩展提供了一个面向GNU Readline库的接口。其readline()函数每次返回一行，不包含结尾的换行符。Readline允许用户进行Emacs式和vi式的行编辑操作，也可以利用这一点来保存以前键入的命令：

```
$command_count = 1;
while (true) {
    $line = readline("[$command_count]--> ");
    readline_add_history($line);
    if (is_readable($line)) {
        print "$line is a readable file.\n";
    }
    $command_count++;
}
```

这个例子显示了一个带有行前增量数字的提示。由于每一行都通过readline_add_history()添加到了Readline历史记录中，只要在提示状态下按“向上”和“向下”箭头键，就能翻出前一次键入的命令行。

参见

fopen()函数的文档（*http://www.php.net/fopen*），fgets()函数的文档（*http://www.php.net/fgets*），fread()函数的文档（*http://www.php.net/fread*），Readline扩展的文档（*http://www.php.net/readline*）和Readline库的文档（*http://cswww.cns.cwru.edu/php/chet/readline/rltop.html*）。

25.4 针对输入文件的每一行运行PHP代码

问题

你想读取一个完整的文件并基于文件的每一行执行PHP代码。例如，你想创建一个使用PHP的Perl兼容正则表达式引擎的grep的命令行版。

方案

使用-R命令行标记来处理标准输入：

```
% php -R 'if (preg_match("/$argv[1]/", $argn)) print "$argn\n";'
      php
      < /usr/share/dict/words

ephphatha
```

如果想在某一行之前或之后执行一个代码块，可以分别使用-B和-E选项：

```
% php -B '$count = 0;'
      -R 'if (preg_match("/$argv[1]/", $argn)) $count++;'
      -E 'print "$count\n";'
      php
      < /usr/share/dict/words

1
```

讨论

有时候你可能想通过命令行调用PHP快速地处理一个文件，该任务可能是一个独立的项目，也可能是一系列管道命令之一。这种方式能够对数据进行快捷的转换。

用PHP来实现这一点很容易，只需使用三个命令行标记和两个特殊的变量即可，它们分别是-R、-B、-E、$argn和$argi。

其中-R标记用于指定要对文件中每一行执行PHP代码。在这个代码块中，可以访问保存于$argn变量相应行的文本。

举一个基本的例子，下面的PHP脚本接受HTML输入、剥离标签并将输出结果：

```
php -R 'print strip_tags($argn) . "\n"; ' < index.html
```

由于PHP能够自动地剥离输入结尾处的换行符，所以这个脚本不仅显示strip_tags($argn)的结果，而且也回送了一个换行符。

这个脚本操作的文件是作为标准输出传递的*index.html*。目前还没有其他指定要处理文件的机制。

下面的例子稍微复杂一些，这是一个grep的简化版，展示了如何通过$argv数组接收输入的参数。

```
% php -R 'if (preg_match("/$argv[1]/", $argn)) print "$argn\n";'
      php
      < /usr/share/dict/words

ephphatha
```

传递给preg_match()的第一个值是/$argv[1]/，它是传递给脚本的第一个参数。在这个例子中是php，因此这个脚本就会在*/usr/share/dict/words*文件中搜索所有包含php的单词。

而找到的结果——ephphatha是一个亚拉姆语（Aramaic）单词，含义为“be opened.”。

除了针对个别的行执行处理外，有时候可能你也需要执行初始化或清理代码。为此需要使用-B和-E标记。

在grep例子的基础上，下面的代码用于计算匹配行的总数：

```
% php -B '$count = 0;'
      -R 'if (preg_match("/$argv[1]/", $argn)) $count++;'
      -E 'print "$count\n";'
      php
      < /usr/share/dict/words

1
```

在-B部分，将$count初始化为0。接着，如果有匹配项的话，则在-R部分加上1。最后，在-E部分将结果输出。

如果要计算匹配行的百分比，除了使用总数外，还要用到$argi：

```
% php -B '$count = 0;'
      -R 'if (preg_match("/$argv[1]/", $argn)) $count++;'
      -E 'print "$count/$argi\n";'
      php
      < /usr/share/dict/words

1/234937
```

$argi变量包含文件中当前的行数，因此在-E部分，它保存的就是所有行的总数。

参见

在命令行中使用PHP的文档（*http://www.php.net/features.commandline*）。

25.5 读取密码

问题

你需要从命令行中读取输入但尚未回送的字符串——例如，输入密码。

方案

在Unix系统中，使用*/bin/stty*来触发对键入字符的回送：

```
// 关掉回送
`/bin/stty -echo`;

// 读取密码
$password = readline();

// 重新触发回送
`/bin/stty echo`;
```

在Windows系统中，使用FFi扩展来访问*msvcrt.dll*中的_getch()：

```
$ffi = new FFi("[lib='msvcrt.dll'] int _getch();");

while(true) {
    // 从键盘获得一个字符输入
    $c = chr($ffi->_getch());
    if ( "\r" == $c ||  "\n" == $c ) {
        // 如果是换行符，中断循环，说明已经取得了密码
        break;
    } elseif ("\x08" == $c) {
        /* 如果是空格符，从 $password 中删除前一个字符*/
        $password = substr_replace($password,'',-1,1);
    } elseif ("\x03" == $c) {
        // 如果是 Control-C，清除 $password 并中断循环
        $password = NULL;
        break;
    } else {
        // 否则，将字符添加到密码中
        $password .= $c;
    }
}
```

讨论

在Unix系统中，使用*/bin/stty*来控制输出过程，因此键入的字符在读取密码过程中不会被回送到屏幕显示。Windows系统中没有*/bin/stty*，所以通过外来函数接口（Foreign Function Interface，FFi）扩展在微软C运行时库*msvcrt.dll*中调用_getch()。_getch()函数只读取字符，但不会将字符回送到屏幕。它返回的是所读取字符的ASCII码，所以需要用chr()转换成字符，然后再基于键入的字符分别执行不同的操作。如果是换行符或回车符，说明密码已经键入完成，因此应该中断循环；如果是空格符，则从密码的结尾删除一个字符；如果是一个Ctrl-C中断，则将密码设置为NULL并退出循环。如果以上条件都不匹配，则将字符连接到$password。在最后退出循环时，$password保存的就是键入的密码。

FFi扩展是PECL的一部分。Windows用户可以在*http://pecl4win.php.net/ext.php/php_ffi.dll*下载一个打包好（pre-built）的DLL文件。要确保所使用的FFi版本高于0.3，否则这里的代码将不会正常运行。

下面的代码用于显示Login:和Password:提示，并将键入的密码与保存在*/etc/passwd*中的相应加密密码进行比较。这个程序要求你的系统未使用密码屏蔽：

```
print "Login: ";
$fh = fopen('php://stdin','r')   or die($php_errormsg);
$username = rtrim(fgets($fh,64)) or die($php_errormsg);

preg_match('/^[a-zA-Z0-9]+$/',$username)
    or die("Invalid username: only letters and numbers allowed");


print 'Password: ';
`/bin/stty -echo`;
$password = rtrim(fgets($fh,64)) or die($php_errormsg);
`/bin/stty echo`;
print "\n";

// 不需要再从键盘读取了
fclose($fh);

// 在 /etc/passwd 中找到相应的行
$fh = fopen('/etc/passwd','r')   or die($php_errormsg);
$found_user = 0;
while (! ($found_user || feof($fh))) {
    $passwd_line = fgets($fh,256);
    if (preg_match("/^$username:/",$passwd_line)) {
        $found_user = 1;
    }
}
fclose($fh);
```

```
$found_user or die ("Can't find user \"$username\"");

// 从 /etc/passwd 中解析正确的行
$passwd_parts = split(':',$passwd_line);

/* 对键入的密码加密，并与 /etc/passwd 中的密码进行比较 */
$encrypted_password = crypt($password,
                            substr($passwd_parts[1],0,CRYPT_SALT_LENGTH));

if ($encrypted_password == $passwd_parts[1]) {
    print "login successful";
} else {
    print "login unsuccessful";
}
```

参见

readline()函数的文档（*http://www.php.net/readline*），chr()函数的文档（*http://www.php.net/chr*），FFi的文档（*http://pecl.php.net/ffi*）和_getch()函数的文档（*http://msdn.microsoft.com/library/en-us/vccore98/HTML/_crt__getch.2c_._getche.asp*）；对于Unix系统，可以看一下系统的*stty(1)*参考页。

25.6 编程：命令解释程序

例25-2中的*command-shell.php*程序（本技巧后半部分）会提供类似Shell的提示，让你能够交互地执行PHP代码。这个函数通过readline()从命令行中读取输入，然后通过eval()函数来运行输入的指令。在默认情况下，会在键入每一行指令后马上运行。但对于多行模式（通过-m或-multiline指定），则会不停地读取每一行，直到在行中键入了“.”时才会返回累积的代码。

此外，*command-shell.php*使用了Readline的自动完成（word-completion）功能辅助键入PHP函数。如果在键入几个字符后按Tab键，你就会看到与以往键入的字符匹配的函数名列表。

这个程序适合于交互地运行一些小型代码片断或用于测试各种命令。而且，在每一行中定义的变量、函数和类会在程序退出之前都保持有定义，借此可以对不同的数据库查询进行测试，例如：

```
% php command-shell.php
[1]> require 'DB.php';

[2]> $dbh = DB::connect('mysql://user:pwd@localhost/phpc');
```

```
[3]> print_r($dbh->getAssoc('SELECT sign,planet,start_day FROM zodiac WHERE
element
LIKE "water"'));
Array
(
    [Cancer] => Array
        (
            [0] => Moon
            [1] => 22
        )
    [Scorpio] => Array
        (
            [0] => Mars
            [1] => 24
        )
    [Pisces] => Array
        (
            [0] => Neptune
            [1] => 19
        )
)
```

*command-shell.php*程序的代码如例25-2中所示。

例25-2：command-shell.php

```
// 加载 readline 库
if (! function_exists('readline')) {
    dl('readline.'. (((strtoupper(substr(PHP_OS,0,3))) == 'WIN')?'dll':'so'))
        or die("Readline library required\n");
}

// 加载 Console_Getopt 类
require 'Console/Getopt.php';

$o = new Console_Getopt;
$opts = $o->getopt($o->readPHPArgv(),'hm',array('help','multiline'));

// 如果参数错误，提示使用方法消息并停止执行
if (PEAR::isError($opts)) {
    print $opts->getMessage();
    print "\n";
    usage();
}

// 默认键入后按行执行
$multiline = false;

foreach ($opts[0] as $opt) {
    // 删除任何前置的 -s
    $opt[0] = preg_replace('/^-+/','',$opt[0]);

    // 检查参数的第一个字符
    switch($opt[0][0]) {
    case 'h':
```

```
        // 显示帮助
        usage();
        break;
    case 'm':
        $multiline = true;
        break;
    }
}

// 设置错误显示
ini_set('display_errors',false);
ini_set('log_errors',true);

// 建立 readline 完成表
$functions = get_defined_functions();
foreach ($functions['internal'] as $k => $v) {
    $functions['internal'][$k] = "$v(";
}
function function_list($line) {
    return $GLOBALS['functions']['internal'];
}
readline_completion_function('function_list');

$cmd = '';
$cmd_count = 1;

while (true) {
    // 从用户获得一行输入
    $s = readline("[$cmd_count]> ");
    // 将其添加到命令历史记录中
    readline_add_history($s);
    // If we're in multiline mode:
    if ($multiline) {
        // 如果只键入了一个 "."
        if ('.' == rtrim($s)) {
            // 通过 eval() 执行代码
            eval($cmd);
            // 清除累积的代码
            $cmd = '';
            // 增加命令数目
            $cmd_count++;
            // 在新行启动下一次提示
            print "\n";
        } else {
            /* 否则，向累积的代码中添加一个换行符添加一个新换行符
               避免 // 样式的注释将其余键入的行注释掉
            */
            $cmd .= $s."\n";;
        }
    } else {
        // 如果不处于多行模式，通过 eval() 执行
        eval($s);
        // 增加命令数目
        $cmd_count++;
        // 在新行启动下一次提示
```

```
        print "\n";
    }
}

// 显示有用的使用信息
function usage() {
    $my_name = $argv[0];

    print<<<_USAGE_
Usage: $my_name [-h|--help] [-m|--multiline]

  -h, --help: display this help
  -m, --multiline: execute accumulated code when "." is entered
                   by itself on a line. The default is to execute
                   each line after it is entered.

_USAGE_;
    exit(-1);
}
```

第26章

PEAR和PECL

26.0 概述

PEAR是PHP Extension and Application Repository（PHP扩展和应用程序仓库）的简写，它是一个协同工作的开源类集。开发者可以使用PEAR类来解析XML，实现验证系统功能，生成SOAP请求，发送带附件的MIME邮件以及完成其他多种多样常见（也可能不一定常见）的任务。当然，梨（pear）也是一种好吃的水果。

PECL是PHP Extension Community Library（PHP扩展社区库）的简写。PECL的发音应该类似“pickle”，是用C语言编写的一系列PHP扩展的统称。这些扩展就如同随着主PHP版本发布而分发的那些扩展一样，只不过它们具有专门的关注点 —— 例如与libssh2库或者ImageMagick图形库的接口。

要了解有关PEAR的全面信息，请阅读介绍PEAR手册；登录*http://pear.php.net*找出新的PEAR包。这个PEAR网站中也包含指向邮件列表文档的链接和RSS源，以便你能方便地获知新程序包的发布信息。

在主PHP版本中只绑定了一些核心的PEAR包。不过，安装PEAR的过程也是对程序的调用，这个程序叫*pear*，通过它来下载并安装补充的PEAR包是很容易的，这个程序也以简单的PEAR安装程序而著称。技巧26.1中介绍了如何使用PEAR安装程序。

此外，通过PEAR安装程序还能针对你个人的项目使用PEAR类管理基础结构。通过按照PEAR的格式创建自己的包，你的用户就能使用*pear*来下载并安装来自你的项目网站的文件。如果你通过这种方式分发的包比较多，可能就要考虑架设运行一台正式的PEAR通道服务器（channel server）了。PEAR安装程序也支持各种各样基于通道的特性，这些特性包含在本章各个技巧中。

安装PEAR需要PHP 4.2.0或更高版本，而且最好能通过`--with-zlib`配置标记来构建

PHP。PEAR包是以经过gzip压缩的tar文档形式存在的，也可能会是未经压缩的tar文档形式。如果是在启用了zlib的PHP环境下安装这些包，会更加方便。

本章介绍了如何找到你想使用的PEAR包，以及如何在你的机器中安装这些包。由于PEAR及PEAR通道提供了许多包，所以需要有一种方便的查找方式。技巧26.2介绍了几种查找PEAR包的不同途径。在找到包的名字并确定了它所处的通道服务器之后，技巧26.3继续讨论了如何查看包的资料和信息。

当定位了所要使用的包之后，就需要运行*pear*将这个包传输到你的机器上，然后在服务器的正确位置来安装它。安装PEAR包和PECL扩展分别是技巧26.4和26.7的主题。技巧26.5中讲解了如何发现你机器中安装的包需要升级，以及如何安装最新版本的内容。如果你想删除一个包，请参见技巧26.6中提供的示例。

PHP从4.3.0版开始在默认情况下会安装PEAR，因此如果你运行的是一个比这个版本还要新的PHP版本，就能直接使用PEAR而不必安装（注1）。自从最初包含的PHP 4.3.0至今，PEAR已经发生了显著的改变，因此我们强烈建议你将PEAR升级到1.4.9或更高版本。要了解如何升级PEAR，请参见技巧26.5。如果你更想知道如何安装纯净的程序，可以参考一下技巧26.1。

在安装PHP的过程中，PEAR会将*pear*安装到与*php*相同的目录下，PEAR包会被放在*prefix/lib/php*目录中（注2）。如果想把PEAR安装到其他目录，可以配置PHP时设置`--with-pear=DIR`配置项。可能你也会安装多个PEAR的技巧，这样就可以在共享服务器环境中派上用场了。技巧26.1中也介绍了如何安装多个PEAR技巧的细节。

在安装了一个PEAR包之后，通过调用require_once在PHP脚本中使用它。例如，下面示范了如何包含Net_Dig包：

```
require_once 'Net/Dig.php';
```

一般来说，如果包名中包含下划线，需要将下划线替换成斜线并在后面加上*.php*扩展名。

有的包中可能会提供多个类文件，而其中一些可能需要在特定的场合下才能使用。SOAP包就是一个很好的例子。在包含这个包时不是简单加载*SOAP.php*，而是要根据脚本的需要包含*SOAP/Client.php*或*SOAP/Server.php*。可以通过阅读相应包的文档来决定在某些特定的场合下应该包含哪些特定的包。

注1：　如果你通过`--disable-cli`禁止了PHP命令行版，PHP则不会安装PEAR。

注2：　也可能是*/usr/local/lib/php*, */usr/lib/php*，或者在有些Linux分发版本中是*/usr/share/php*。

因为PEAR包是作为常规的PHP文件来包含的，所以要保证包含PEAR包的目录位于include_path中。如果没有，include_once和require_once将无法找到PEAR的类文件。

如果要查看一些特定PEAR包的用法说明和示例，可以访问PEAR的网站*http://pear.php.net/packages.php*，其中许多包都有最终用户文档以及示例。其他的也至少会包含一套提供用户示例的API文档。如果这些资料无法满足你的需要，还可再看一看相应包的PHP文件的顶部区域，多数情况下也会包含一个用法示例。

PECL扩展的文档并不总能很容易地找到。在PHP手册中包含一些非常好的PECL扩展文档，如ClibPDF扩展（*http://www.php.net/cpdf*），而有些PECL扩展可能根本就没有文档，而用法说明则必须通过阅读与PECL网站绑定的源代码中的PHP测试脚本自行收集。在某些极端的情况下，你只能通过阅读相应扩展的源代码来获得一个该扩展能实现什么功能的整体印象。

PEAR同PHP的组合提供了一套数量可观的可重用代码库，从而使这两者成为普遍存在的PHP社区中一笔巨大的财富。

26.1 使用PEAR安装程序

问题

你想使用PEAR安装程序：*pear*。通过这个程序可以安装新的包、完成包的升级和取得关于现有PEAR包的相关信息。

方案

通过PEAR安装程序执行一个命令，在命令行中键入命令名作为第一个参数：

```
% pear command
```

讨论

下面所示为如何通过list命令列出所有安装的PERA包（注3）：

```
% pear list
Installed packages, channel pear.php.net:
```

注3：　在*pear*的早期版本中，该命令为list-installed。

```
========================================
Package                    Version State
Archive_Tar                1.3.1   stable
Console_Getopt             1.2     stable
DB                         1.7.6   stable
DB_DataObject              1.8.4   stable
Date                       1.4.6   stable
File_Passwd                1.1.6   stable
HTML_Common                1.2.2   stable
HTML_QuickForm             3.2.5   stable
HTML_QuickForm_Controller  1.0.5   stable
HTML_Template_IT           1.1     stable
HTTP_Request               1.3.0   stable
HTTP_Session               0.5.1   beta
MDB2                       2.0.0   stable
Mail_Mime                  1.3.1   stable
Net_Socket                 1.0.6   stable
Net_URL                    1.0.14  stable
Net_UserAgent_Detect       2.0.1   stable
PEAR                       1.4.9   stable
PHP_Compat                 1.5.0   stable
Pager                      2.4.0   stable
XML_Parser                 1.2.7   stable
XML_RPC                    1.4.0   stable
XML_RSS                    0.9.9   beta
XML_Serializer             0.18.0  beta
XML_Util                   1.1.1   stable
```

如果想了解所有有效的PEAR命令，使用help。其中有许多命令也采用了简写名称；例如，list也可以简写为l。这些简写的命令通常使用了命令名称的前几个字母。不过，由于PEAR安装程序提供了很多命令，因此最好还是认真地看一下简写命令列表直到把它们全都记住。可以通过下面命令查看*pear*简写命令的列表：

```
% pear help shortcuts
```

*pear*中同时包含了使用和开发PEAR包的命令，为此，许多命令可能并非你所需要的。例如，package命令，其作用是创建一个新PEAR包。如果你只想运行其他人的包，那么完全可以忽略这个命令。表26-1中列出了一些常用的PEAR安装程序命令。

表26-1：常用的PEAR安装程序命令

命令名称	简写	说明
install	i	下载并安装包
upgrade	up	升级安装的包
uninstall	un	删除安装的包
list	l	列出安装的包
list-upgrades	lu	列出针对已安装包的全部有效更新

表26-1：常用的PEAR安装程序命令（续）

命令名称	简写	说明
channel-discover	di	从所在服务器中初始化一个备用的PEAR通道
list-channels	lc	列出所有本地化配置的PEAR通道
search	sp	搜索包

与所有解释程序一样，如果你想运行*pear*，首先必须拥有执行它的权限。如果作为root用户可以运行*pear*，但作为一个常规用户时却不行，就需要确认一下组或者执行位（world-execution bit）是否进行了有效的设置。类似地，为了进行某种操作，*pear*会在包含PEAR文件的目录中创建一个锁定文件。因此，你还必须拥有那个目录中扩展名为*.lock*的文件写入的权限。

在寻找PEAR包的位置时，需要运行`config-get php_dir`命令。通过`config-get ini_get('include_path')`可以查看PHP内部`include_path`的值，也可以查看*php.ini*文件。如果由于处于共享的主机环境中而不能修改*php.ini*文件，可以在包含任何PEAR文件时在你脚本顶部的`include_path`中添加相应的目录。要了解有关在PHP内部设置配置变量的更多内容，请参看技巧20.5。

如果你的网络中包含HTTP代理服务器，可以通过以下命令配置PEAR以使用该代理服务器：

```
% pear config-set http_proxy proxy.example.com:8080
```

可以使用下列格式配置PEAR安装程序：

```
% pear set-config setting value
```

其中的*setting*是要修改参数的名称，而*value*则是要设置的新值。要查看当前所有的设置，使用`config-show`命令。

```
% pear config-show
Configuration (channel pear.php.net):
=====================================
Auto-discover new Channels     auto_discover    <not set>
Default Channel                default_channel  pear.php.net
HTTP Proxy Server Address      http_proxy       <not set>
PEAR server [DEPRECATED]       master_server    pear.php.net
Default Channel Mirror         preferred_mirror pear.php.net
Remote Configuration File       remote_config     <not set>
PEAR executables directory     bin_dir          /usr/local/bin
PEAR documentation directory   doc_dir          /usr/local/lib/php/doc
PHP extension directory        ext_dir          /usr/lib/php/extensions/
   no-debug-non-zts-20020429
```

```
PEAR directory                  php_dir          /usr/local/lib/php
PEAR Installer cache directory  cache_dir        /tmp/pear/cache
PEAR data directory             data_dir         /usr/local/lib/php/data
PHP CLI/CGI binary              php_bin          /usr/local/bin/php
PEAR test directory             test_dir         /usr/local/lib/php/test
Cache TimeToLive                cache_ttl        3600
Preferred Package State         preferred_state  beta
Unix file mask                  umask            22
Debug Log Level                 verbose          1
PEAR password (for              password         <not set>
maintainers)
Signature Handling Program      sig_bin          /usr/local/bin/gpg
Signature Key Directory         sig_keydir       /etc/pearkeys
Signature Key Id                sig_keyid        <not set>
Package Signature Type          sig_type         gpg
PEAR username (for              username         <not set>
maintainers)
User Configuration File         Filename         /home/foo/.pearrc
System Configuration File       Filename         /etc/pear.conf
```

使用config-help命令可以得到每个配置选项的简明描述。

如果你没有安装PEAR，或者你是在一个共享主机的环境中，并且不能通过升级或者其他方式来修改系统端的PEAR配置，那么你可能需要通过引导程序获得一个最新的PEAR副本。一般来说，只要你是针对共享环境来配置这个最新的PEAR的副本，那么在共享主机上的安装过程就不应该出现问题。可能你想使用一个别出心裁的目录，如*/home/exampleuser/pear/php*来保存PEAR包文件，而不是将其保存在传统的安装位置。这也没问题，只要你的include_path值正确地映射了这个目录就可以。

在通过命令行引导一个最新的PEAR副本时，需要运行下列命令：

```
% lynx -source http://go-pear.org | php -q
Welcome to go-pear!

Go-pear will install the 'pear' command and all the files needed by
it.  This command is your tool for PEAR installation and maintenance.

Go-pear also lets you download and install the PEAR packages bundled
with PHP: DB, Net_Socket, Net_SMTP, Mail, XML_Parser.

If you wish to abort, press Control-C now, or press Enter to
continue:
```

这样就会从PEAR网站下载一个PHP脚本，并将其交给PHP执行。该程序会下载运行pear所需的全部文件并升级安装运行。

在某些Unix系统中，需要运行的可能不是*lynx*命令，而是*link*命令。如果你安装了PHP的命令行版，可能也需要删除-q标记 —— CLI程序会自动禁用（suppresses）HTTP

头部。如果*go-pear*看起来运行不够顺利，可以在传输的PHP命令中添上`-d output buffering=off`。

Windows系统中的安装过程分为两步：

```
C:\> php-cli -r 'readfile("http://go-pear.org");' > go-pear
C:\> php-cli go-pear
```

运行*go-pear*脚本需要PHP 4.2或更高版本。对Windows系统中的安装过程来说，*php-cli*就是PHP的命令行版。

参见

PEAR在线文档中与安装程序有关的部分内容（*http://pear.php.net/manual/en/installation.php*）。

26.2 查找PEAR包

问题

你想得到一个有关PEAR包的列表。通过这样一个列表，你就能够在了解更多内容的基础上决定安装哪个包。

方案

在*http://pear.php.net/packages.php*上浏览包，或者在*http://pear.php.net/search.php*中搜索包。使用*pear*的`remote-list`命令也可以得到一个PEAR包的列表，或者通过`search`命令来搜索相应的包。在PEAR的通道目录网站*http://www.upear.com/*或*http://pear.php.net/channels/*中可以找到PEAR通道服务器的列表。

讨论

有几种查看有效的PEAR或PEAR兼容包的途径。首先，在PEAR的官方网站*http://pear.php.net/packages.php*上面通过目录式的风格来浏览包的列表。在该网站上，可以按照单独的PEAR类别来查看PEAR包。

另一种途径是通过网址*http://pear.php.net/search.php*来搜索列表。该搜索页面允许你通过包名、作者、类别以及发布日期进行搜索。

你还可以通过PEAR安装程序获得包列表，这需要使用下面所示的remote-list命令：

```
% pear remote-list
Channel pear.php.net Available packages:
=========================================
Package                         Version
Auth_HTTP                       2.1.6
Auth                            1.3.0
Auth_SASL                       1.0.1
LiveUser                        0.16.11
Auth_PrefManager                1.1.4
Auth_RADIUS                     1.0.4

...

XML_Indexing                    0.3.6
XML_Feed_Parser                 0.3.0beta
XML_RPC2                        0.0.7
XML_Query2XML                   0.6.0
```

其中remote-list的简写形式是rl。

也可以通过-c标记在remote-list命令后指定要查询的通道，来从兼容PEAR的通道服务器中查询有效的包。为此，必须首先让PEAR安装程序知道替代的通道服务器。例如：

```
% pear channel-discover pearified.com
Adding Channel "pearified.com" succeeded
Discovery of channel "pearified.com" succeeded
% pear remote-list -c pearified
Retrieving data...0%....50%....
Channel pearified Available packages:
=====================================
Package                           Version
Editors_FCKeditor                 2.2.0
Editors_TinyMCE                   2.0.1
Icons_Silk                        1.3.0

...

Role_Web                          1.1.1
SimpleTest                        1.0.0
Smarty                            2.6.8
```

在从命令行中搜索包名时，需要使用search命令：

```
% pear search auth
Retrieving data...0%....50%....
Matched packages, channel pear.php.net:
=======================================
Package             Stable/(Latest)          Local
Auth                1.3.0/(1.3.0 stable)     1.3.0 Creating an authentication
                                             system.
```

```
Auth_HTTP           2.1.6/(2.1.6 stable)     HTTP authentication
Auth_PrefManager    1.1.4/(1.1.4 stable)     1.1.4 Preferences management class
Auth_PrefManager2   -n/a-/(2.0.0dev1 alpha)  Preferences management class
Auth_RADIUS         1.0.4/(1.0.4 stable)     Wrapper Classes for the RADIUS PECL.
Auth_SASL           1.0.1/(1.0.1 stable)     Abstraction of various SASL mechanism
                                             responses
```

这里采用了大小写敏感的方式搜索包名，并返回相应的包名、最新稳定版本，任何处于dev、alpha或beta状态的最新版本，你在本地安装的版本（如果有）以及关于包的简要说明。

参见

技巧26.3中介绍了有关查找包的更多信息。

26.3 查找有关包的信息

问题

你想收集有关某个包的信息，例如有关包的用途、维护者、包的版本号以及该包的发布协议等的描述。

方案

如果这个包安装在你的机器中，那么可以使用PEAR安装程序的info命令。

```
% pear info Net_URL
```

否则，使用remote-info命令：

```
% pear remote-info SOAP
```

也可以通过查看包的主页（*http://pear.php.net*）来获得相关信息。

讨论

info命令提供的是有关某个包的摘要信息：

```
% pear info Net_URL
About Net_URL-1.0.14
====================
Provides                Classes:
```

```
Package               Net_URL
Summary               Easy parsing of Urls
Description           Provides easy parsing of URLs and their
                      constituent parts.
Maintainers           Richard heyes <richard@php.net> (lead)
Version               1.0.14
Release Date          2004-06-19
Release License       BSD
Release State         stable
Release Notes         Whitespace
Package.xml Version   1.0
Last Installed Version - None -
Last Modified         2006-05-08
```

如果你没有安装包，可以通过远程服务器查询相关描述：

```
% pear remote-info Net_URL
Package details:
================
Latest      1.0.14
Installed   - no -
Package     Net_URL
License     BSD
Category    Networking
Summary     Easy parsing of Urls
Description Provides easy parsing of URLs and their
            constituent parts.
```

这个请求显示的信息稍有不同。它没有包含发布日期，但却包含了通常的PEAR类别和这个包的最新版本号。

在相关包的主页中可以找到更完整的信息，其中还提供了指向更早发布版本的链接、更新日志以及该包的CVS仓库（repository）的可浏览权限。还能查看到包的下载统计数字信息。图26-1所示为一个包信息的示例页面。

参见

技巧26.2中介绍的搜索包的方法。

26.4 安装PEAR包

问题

你想安装一个PEAR包。

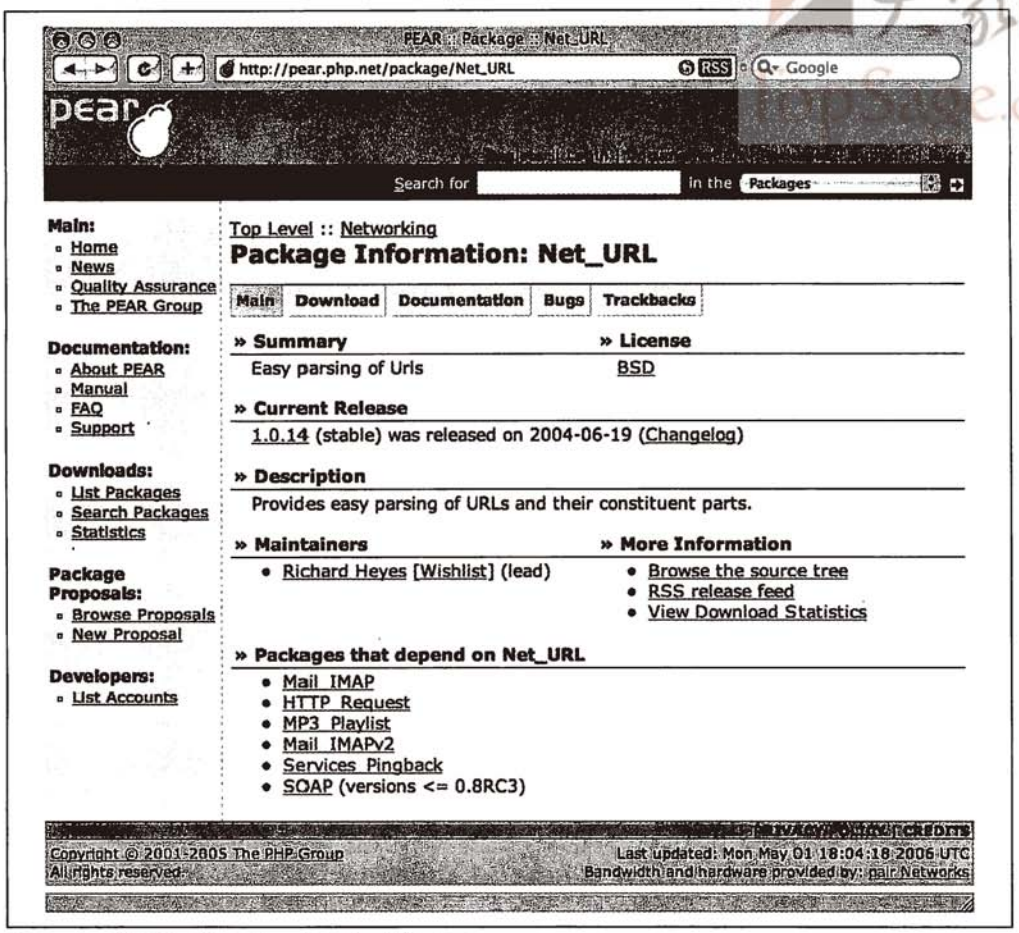

图26-1：PEAR网站中 Net_URL包的信息页面

方案

通过PEAR安装程序从相应的PEAR通道服务器中下载并安装这个包：

```
% pear install Package_Name
```

也可以指定从其他的PEAR通道安装：

```
% pear install channel/Package_Name
```

还可以从互联网中的任何位置安装包：

```
% pear install http://pear.example.com/Package_Name-1.0.0.tgz
```

下面是如何安装位于本地的包的副本：

```
% pear install Package_Name-1.0.0.tgz
```

讨论

在安装PEAR包时，必须拥有保存包的目录的写入权限，这个目录默认是*/usr/local/lib/php/*。

也可以同时请求安装多个包：

```
% pear install XML_Parser XML_Tree
downloading XML_Parser-1.2.7.tgz ...
Starting to download XML_Parser-1.2.7.tgz (12,939 bytes)
.....done: 12,939 bytes
downloading XML_Tree-1.1.tgz ...
Starting to download XML_Tree-1.1.tgz (4,826 bytes)
...done: 4,826 bytes
install ok: channel://pear.php.net/XML_Tree-1.1
install ok: channel://pear.php.net/XML_Parser-1.2.7
```

在安装包时，PEAR会检查所有必需的PHP函数以及新包要使用的PEAR包。如果检查失败，PEAR则会报告相应的依赖关系：

```
% pear install MDB2_Driver_Mysql
 install MDB2_Driver_Mysql
Did not download dependencies: pear/MDB2, use --alldeps or --onlyreqdeps to
download automatically
pear/MDB2_Driver_mysql requires package "pear/MDB2" (version >= 2.0.1)
No valid packages found
install failed
```

正如你在以上错误信息中看到的，PEAR安装程序没有试图下载要使用的内容。此默认行为就是假设你不想安装或更新相关的内容。最方便的安装命令就是-o，这是对--onlyreqdeps的简写，这个命令表示要安装所有必要的内容。

使用了-o命令后，安装成功：

```
% pear install -o MDB2_Driver_Mysql
downloading MDB2_Driver_mysql-1.0.1.tgz ...
Starting to download MDB2_Driver_mysql-1.0.1.tgz (22,240 bytes)
........done: 22,240 bytes
downloading MDB2-2.0.1.tgz ...
Starting to download MDB2-2.0.1.tgz (91,219 bytes)
...done: 91,219 bytes
install ok: channel://pear.php.net/MDB2-2.0.1
install ok: channel://pear.php.net/MDB2_Driver_mysql-1.0.1
```

如果你想忽略必要的内容，可以使用-n或-nodeps命令告诉安装程序忽略缺少的内容继续安装。

参见

技巧26.7中关于安装PECL包的内容；技巧26.5中有关更新现有包的更多内容；技巧26.6中有关删除包的讨论。

26.5 升级PEAR包

问题

你想获得包含新增功能及错误修复的最新版本，而需要对系统中的包进行升级。

方案

找到可用的升级包，然后通过以下命令对该包进行升级：

```
% pear list-upgrades
% pear upgrade -o Package_Name
```

讨论

通过PEAR安装程序将某个包升级到新版本是一项简单的任务。如果你知道哪个具体的包已经废弃，就可以直接对其进行升级。不过，可能你会认为定期地检查一下是否有新版本发布比较合适。

为此，可以使用list-upgrades命令，该命令会输出一个表，其中包括包所在的通道服务器、包名、本地版本号及状态、远程升级包的版本号及状态，以及需要下载升级文件的大小：

```
% pear list-upgrades
pear.php.net Available Upgrades (stable):
=========================================
Channel       Package             Local           Remote          Size
pear.php.net  HTML_Table          1.6.1 (stable)  1.7.0 (stable)  13.7kB
pear.php.net  HTML_Template_IT    1.1.3 (stable)  1.1.4 (stable)  19.7kB
pear.php.net  Log                 1.9.3 (stable)  1.9.5 (stable)  37kB
pear.php.net  Mail                1.1.9 (stable)  1.1.10 (stable) 16.5kB
pear.php.net  MDB2                2.0.0 (stable)  2.0.1 (stable)  90kB
pear.php.net  Pager               2.3.6 (stable)  2.4.1 (stable)  31kB
```

```
pear.php.net  PEAR              1.4.8 (stable)  1.4.9 (stable)  277kB
pear.php.net  Services_Weather  1.3.2 (stable)  1.4.0 (stable)  53kB
```

如果你的包当前都是最新版本，*pear*就会输出：

```
Channel pear.php.net: No upgrades available
```

如果要更新特定的包，则需要使用upgrade命令。例如：

```
% pear upgrade MDB2
Did not download dependencies: pear/PEAR, use --alldeps or --onlyreqdeps to
download automatically
downloading MDB2-2.0.1.tgz ...
Starting to download MDB2-2.0.1.tgz (91,219 bytes)
.....................done: 91,219 bytes
upgrade ok: channel://pear.php.net/MDB2-2.0.1
```

list-upgrades的简写方式是lu，用于列出最新的更新供你选择。

PEAR网站上提供了包含新包和增强（upgraded）包列表的RSS源，可以访问*http://pear.php.net/rss.php*订阅该源。此外，在PEAR的网站和许多其他PEAR通道服务器上也提供了包含新包和增强包列表的聚合RSS源，可以参考*http://pearified.com/planet.xml*。

参见

技巧26.4和26.7中关于安装PEAR和PECL包的相关内容。技巧26.6中有关删除包的方法。技巧12.12中关于如何解析RSS源的更多内容。

26.6 卸载PEAR包

问题

你想从系统中删除一个PEAR包。

方案

使用uninstall命令告诉PEAR安装程序删除不想要包：

```
% pear uninstall Pager
uninstall ok: channel://pear.php.net/MDB2-2.4.1
```

讨论

卸载包后，整个包的内容都会从系统中被删除。如果你想重新安装这个包，必须使用与安装其他新包一样的方式。

如果你想删除的包与其他包有依存关系，PEAR会警告你并中断卸载过程。例如，我们先查看一下PEAR安装情况的示例：

```
% pear list
Installed packages, channel pear.php.net:
=========================================
Package           Version State
Archive_Tar       1.3.1   stable
DB                1.7.6   stable
HTML_Common       1.2.2   stable
HTML_Table        1.7.0   stable
MDB2              2.0.1   stable
MDB2_Driver_mysql 1.0.1   stable
PEAR              1.4.9   stable
XML_Parser        1.2.7   stable
XML_Tree          1.1     stable
```

现在，我们试着卸载MDB2包：

```
% pear uninstall MDB2
pear/MDB2 cannot be uninstalled, other installed packages depend on this
package
```

当然，你也可以通过-n标记或--nodeps标记命令PEAR安装程序忽略包之间存在的依存关系而进行强制卸载，但采取这种做法时最好要谨慎行事。

如果你想让升级的包自动回滚到上一个版本，通过uninstall是做不到的。同样，PEAR也不允许在安装高版本的包之后再安装低版本的包。但是，如果想强制用旧版的包覆盖新版的包，可以使用install -f或install --force来达到目的：

```
% pear install DB-1.7.5
Skipping package "pear/DB", already installed as version 1.7.6
No valid packages found
install failed

% pear install -f DB-1.7.5
downloading DB-1.7.5.tgz ...
Starting to download DB-1.7.5.tgz (124,767 bytes)
.........................done: 124,767 bytes
install ok: channel://pear.php.net/DB-1.7.5
```

uninstall的简写方式是un。

参见

技巧26.4和技巧26.7中有关安装PEAR和PECL包的更多内容。

26.7 安装PECL包

问题

你想安装一个PECL包，该包是以C语言编写的，作为PHP扩展存在，能够在PHP内部调用。

方案

首先要确保你拥有全部必需的扩展库，然后使用PEAR安装程序的绑定（bundled）命令*pecl*：

```
% pecl install mailparse
```

如果要在PHP中使用这个扩展，需要在*php.ini*文件中添加对应的一行配置指令：

```
extension=mailparse.so
```

讨论

安装PECL包的前端过程虽然与安装以PHP编写的PEAR包的安装过程几乎没有区别，但幕后执行的任务却迥然不同。由于PECL扩展是用C语言编写的，所以安装程序必须对这个扩展执行编译，并根据已经安装的PHP版本进行配置。所以，在目前的条件下，如果你使用的是Unix系统而且安装了必要的开发工具；或者使用的是Windows系统并且有MSDev（MS-Develop Studio，微软公司提供的软件开发平台，可以使用VC++和VJ++等工具），那么就可以安装PECL包。

与基于PHP的PEAR包不同，PECL扩展不会在缺乏编译该扩展必须的库文件时提醒你。它将确保正确安装这些必须文件的任务交给了你。如果你在安装PECL扩展时遇到了问题，可以参阅一下程序包中附带的README文件或者其他文档资料。安装程序会把这些参考文档放在PEAR层次下的*docs*目录中。

在安装PECL扩展时，*pecl*命令会下载分发的文件、提取其中的内容、运行*phpize*命令并按照你机器中安装的PHP版本进行配置，然后生成并安装这个扩展。安装过程中也可能会提示你选择库文件保存的位置：

```
% pecl install mailparse
downloading mailparse-2.1.1.tgz ...
Starting to download mailparse-2.1.1.tgz (35,883 bytes)
.........done: 35,883 bytes
9 source files, building
running: phpize
Configuring for:
PHP Api Version:         20031224
Zend Module Api No:      20041030
Zend Extension Api No:   220040412

...

Build complete.
(It is safe to ignore warnings about tempnam and tmpnam).

running: make INSTALL_ROOT="/var/tmp/pear-build-root/install-mailparse-2.1.1"
install
Installing shared extensions:
/var/tmp/pear-build-root/install-mailparse-2.1.1/usr/lib/php/20041030/
running: find "/var/tmp/pear-build-root/install-mailparse-2.1.1" -ls
8306920    4 drwxr-xr-x   3 root     root       4096 May 1  16:40 /var/tmp/pear-build-root/↵
   install-mailparse-2.1.1
4522201 4 drwxr-xr-x      3 root     root       4096 May 1 16:40 /var/tmp/pear-build-root/↵
   install-mailparse-2.1.1/usr
4522202 4 drwxr-xr-x      3 root     root       4096 May 1 16:40 /var/tmp/pear-build-root/↵
   install-mailparse-2.1.1/usr/lib
4522203 4 drwxr-xr-x      3 root     root       4096 May 1 16:40 /var/tmp/pear-build-root/↵
   install-mailparse-2.1.1/usr/lib/php
8306938 4 drwxr-xr-x      2 root     root       4096 May 1 16:40 /var/tmp/pear-build-root/↵
   install-mailparse-2.1.1/usr/lib/php/20041030
8306939 140 -rwxr-xr-x    1 root     root     136671 May 1 16:40 /var/tmp/pear-build-root/↵
   install-mailparse-2.1.1/usr/lib/php/20041030/mailparse.so

Build process completed successfully
Installing
'/var/tmp/pear-build-root/install-mailparse-2.1.1//usr/lib/php/20041030/mailparse.so'
install ok: channel://pecl.php.net/mailparse-2.1.1
You should add "extension=mailparse.so" to php.ini
```

PECL扩展与以PHP编写的PEAR包分别保存在不同的位置。如果你想运行*pecl*，必须对PHP扩展目录拥有写入的权限。为此，你可能会选择安装PHP时所使用的同样身份来安装这些包。同样地，还必须检查一下对这些文件的执行权限——因为多数PEAR文件是不可执行的，而且你的umask值可能也没有对这些可执行文件提供正确的权限设置。

如果你是在Windows环境下运行PHP和PECL，可能你会更愿意从*http://pecl4win.php.net/*上面下载针对PECL扩展重新编译的DLL文件。

PHP的`dl()`可以用于在运行期间加载扩展，但是这个函数目前是不推荐使用的。如果可能的话，可以考虑通过*php.ini*文件中的配置来启用新安装的扩展。

参见

技巧26.4中有关安装PEAR包的信息；技巧26.5中关于对现有包进行升级的更多内容；技巧26.6中卸载包的过程；面向Windows系统的PECL仓库（*http://pecl4win.php.net*）。

计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

Java 一览无余：Java 视频教程 | Java SE | Java EE

.Net 技术精品资料下载汇总：ASP.NET 篇

.Net 技术精品资料下载汇总：C#语言篇

.Net 技术精品资料下载汇总：VB.NET 篇

撼世出击：C/C++编程语言学习资料尽收眼底 电子书+视频教程

Visual C++(VC/MFC)学习电子书及开发工具下载

Perl/CGI 脚本语言编程学习资源下载地址大全

Python 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载

数据库管理系统(DBMS)精品学习资源汇总：MySQL 篇 | SQL Server 篇 | Oracle 篇

平面设计优秀资源学习下载 | Flash 优秀资源学习下载 | 3D 动画优秀资源学习下载

最强 HTML/xHTML、CSS 精品学习资料下载汇总

最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总

UML 学习电子资下载汇总 软件设计与开发人员必备

经典 LinuxCBT 视频教程系列 Linux 快速学习视频教程一帖通

天罗地网：精品 Linux 学习资料大收集(电子书+视频教程) Linux 参考资源大系

Linux 系统管理员必备参考资料下载汇总

Linux shell、内核及系统编程精品资料下载汇总

UNIX 操作系统精品学习资料<电子书+视频>分类总汇

FreeBSD/OpenBSD/NetBSD 精品学习资源索引 含书籍+视频

Solaris/OpenSolaris 电子书、视频等精华资料下载索引

作者简介

Adam Trachtenberg是eBay咨询平台（Platform Evangelism）的资深管理者，他在那里向全世界的开发者和商界人士传布eBay平台的理念。在加入eBay之前，Trachtenberg在与他人共同创办的两家公司Student.Com和TVGrid.Com中担任副总裁。在这两家公司任职期间，他领导了网站前、中、后台的设计和开发。Trachtenberg自1997年开始使用PHP，同时也是《Upgrading to PHP 5》的作者和《PHP Cookbook》的合著者——这两本书都由O'Reilly Media出版。他住在加利福尼亚州的旧金山，博客网址是*http://www.trachtenberg.com*，并拥有哥伦比亚大学的学士和MBA学位。

David Sklar是Ning的一名软件架构师，该公司研发了一个用于创建和共享社区应用程序的软件平台。除了《PHP Cookbook》，他还是《Learning PHP 5》（O'Reilly）和《Essential PHP Tools》（Apress）的作者。David经常在各种会议上发表演讲，包括O'Reilly Open Source Conventioin、O'Reilly Emerging Technology Conference、USENIX和许多PHP会议。他的博客网址是*http://www.sklar. com/blog*，同时还在维护PX（*http://px.sklar.com*）——他于1996年创建的一个PHP代码的交换网站。David拥有耶鲁大学的计算机科学学位，他住在纽约，擅长骑术，喜欢吃咸烤扇贝，酷爱首都艺术博物馆中18～19世纪的美国家居用品。

动物介绍

本书封面的动物是一只加拉帕哥斯陆鬣蜥（一种产于南美洲和西印度群岛的大蜥蜴），曾经繁盛于加拉帕哥斯群岛（位于厄瓜多尔西部），这种鬣蜥被19世纪初的殖民者们证实味道很鲜美。后来，家畜被引进到这些群岛上，进一步破坏了这些爬行动物的生存环境和食物链。今天，在圣地亚哥（智利首都）岛上鬣蜥已经销声匿迹，而其他岛屿上也只有少数得以幸存下来。

作为南美大陆鬣蜥的远亲，加拉帕哥斯陆鬣蜥能长到三英尺长，雄性鬣蜥大概有30磅重。它们坚韧的黄色鳞甲表面分布着白色、黑色、褐色的斑点，显得"锈迹斑斑"。这些蜥蜴就如同旧时神话中传说的灵异——龙一般，有着长长的尾巴、带利爪的脚和刺状的发冠。不过，事实上它们对人类是无害的。

大陆鬣蜥生活在比较干燥的地区，喜欢早晨出来晒太阳。但是到了中午，它们会跑到仙人掌、石头和树的阴影下乘凉。晚上，为了保存体内热量，它们又会钻到地下的洞穴里睡大觉。

这些爬行动物从不挑食，它们通常以生长缓慢的植物以及灌木为主食，有时也会用一些掉在地上的果子和仙人掌树充饥。这些植物为它们提供了大部分必需的水分，然而如果有可能的话，它们也会喝上一些新鲜的水作为补给。

根据体型大小，鬣蜥的成熟期在8～15岁之间。它们会在一段时期内聚集、交配，而这个交配期在每个岛上都不一样。交配之后，雌性鬣蜥会迁徙到适宜的地区安家。通常，雌性鬣蜥会在洞中产下2～20枚蛋。产蛋之后，它们会继续在封闭的洞口附近守护，以防其他雌性鬣蜥也到同一个洞穴中产蛋。

幼鬣蜥会在85～110天之后孵化出世，还要再花上大约一周的时间才能从洞中破土而出。正常情况下，如果孵化成功，而且出生第一年食物充足，没有诸如鹰、白鹭、苍鹭以及蛇等当地食肉动物的侵袭，它们就有可能活到超过60岁。事实上，由于野猫等食肉动物的捕食，这些幼鬣蜥必须确保幸存下来并长到至少3～4岁才能使自己长到野猫们无法捕食的个头。