

Pro PHP Application Performance

Tuning PHP Web Projects for Maximum Performance

高性能PHP 应用开发

[美] Armando Padilla 著
Tim Hawkins

盛海艳 刘霞 译

- Yahoo公司技术专家力作
- PHP性能优化修炼秘籍
- 掌握各种重构技术和最佳实践



人民邮电出版社
POSTS & TELECOM PRESS



Armando Padilla

专注于PHP技术已有13年，领导并全面参与了基于LAMP的多个网络应用程序。Armando目前是Yahoo的高级工程师，曾负责多个著名的高流量应用，例如2010年冬奥会和2010年南非世界杯网站、Yahoo手机新闻应用程序。



Tim Hawkins

早在1993年就创建了loot.com站点，这是世界上最早的分门站点之一。之后他帮助Yahoo欧洲公司打造了很多关键产品，例如搜索、本地搜索、邮件、即时通信软件和社会化网络等。他目前在美国的一家大型电子零售公司管理庞大的海外团队，负责开发和部署下一代电子商务应用程序。

Apress®

图灵社区: www.ituring.com.cn

反馈/投稿/推荐信箱: contact@turingbook.com

热线: (010)51095186转604

分类建议 计算机/网络技术/PHP

人民邮电出版社网址: www.ptpress.com.cn

Pro PHP Application Performance (Using PHP 5.4 Projects for Maximum Performance)

高性能PHP应用开发

全球已有超过百万的程序员从事PHP开发，而任何认真的程序员均需要了解如何提升PHP项目的性能。本书专门研究了这一课题。

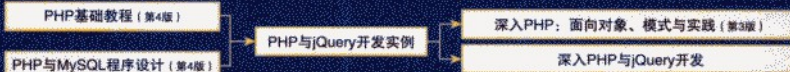
本书将深入探讨在应用程序运行中起重要作用的所有技术和组件。现在是数秒间就决定能否留住用户的时代，所有人都必须把优化作为项目路线图的必备环节。但到底应该分析应用程序中的哪些组件呢？应该怎样优化，又该如何测量应用程序的执行性能呢？这些正是本书要回答的问题。

本书的内容还包括：为什么应该优化某个特定的组件，为什么优化某个函数会比优化另一个更有效，如何寻找和使用面向开源社区的优化工具，如何部署缓存软件和Web服务器软件。此外，本书还会讲解更多高级技巧，包括：

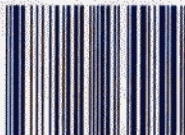
- ◆ 使用Xdebug来分析一些没有实现最佳运行效率的函数；
- ◆ 比较不同的PHP函数所执行的opcode，从而搜索到运行效率最高的函数；
- ◆ 当应用程序正在为用户提供服务时，使用strace来分析Apache。

读完本书后，读者会对从哪里开始优化形成完整的认识。最重要的是，在未来优化PHP应用程序时，将会拥有趁手的工具来助一臂之力。

图灵PHP图书阅读路线图



ISBN 978-7-115-26495-4



9 787115 264954 >

ISBN 978-7-115-26495-4

定价: 39.00元



TURING

图灵程序设计丛书

Web开发系列

Pro PHP Application Performance
Tuning PHP Web Projects for Maximum Performance

高性能PHP 应用开发

[美] Armando Padilla 著
Tim Hawkins
盛海艳 刘霞 译

人民邮电出版社
北京

图书在版编目(CIP)数据

高性能PHP应用开发 / (美) 帕蒂拉 (Padilla, A.),
(美) 霍金斯 (Hawkins, T.) 著; 盛海艳, 刘霞译. —
北京: 人民邮电出版社, 2011. 11

(图灵程序设计丛书)

书名原文: Pro PHP Application
Performance: Tuning PHP Web Projects for Maximum
Performance

ISBN 978-7-115-26495-4

I. ①高… II. ①帕… ②霍… ③盛… ④刘… III.
①PHP语言—程序设计 IV. ①TP312

中国版本图书馆CIP数据核字(2011)第201252号

内 容 提 要

本书是一本广受好评的 PHP 性能优化方面的图书, 通过介绍 PHP 的原理和相关的工具集来实现调优性能的目的。它分析和研究了 Web 应用程序的前端和后端, 并系统地提升了其性能和运行效率。本书还介绍了 PHP 编码最佳实践的运用以及如何使用工具来应用缓存技术。另外书中也涉及了对 Web 服务器的优化和数据库的优化。

本书适合 PHP 开发人员阅读。

图灵程序设计丛书 高性能PHP应用开发

-
- ◆ 著 [美] Armando Padilla, Tim Hawkins
译 盛海艳 刘霞
责任编辑 傅志红
执行编辑 李盼
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷
- ◆ 开本: 800×1000 1/16
印张: 11.75
字数: 284千字 2011年11月第1版
印数: 1—3 500册 2011年11月北京第1次印刷
著作权合同登记号 图字: 01-2011-0805号
ISBN 978-7-115-26495-4
-

定价: 39.00元

读者服务热线: (010)51095186转604 印装质量热线: (010)67129223

反盗版热线: (010)67171154

译者序

作为一种HTML内嵌式语言，PHP的历史可追溯至1994年，它是一种在服务器端执行的脚本语言，其风格有点类似C语言，但除了融合它自己独创的语法和C语言的语法之外，其中也有Java语法和Perl语法的影子，因此可称得上是集几大语言精华于一身的优秀语言。正如我们所熟知的，它最出众的特点是能够快速执行动态网页。

回顾一下PHP的发展历史，可粗略地分为几个阶段。1994年，Rasmus Lerdorf创建了PHP，这位PHP之父最初只是用它来统计自己网站的流量。1995年，他以PHP Tools这个名称对外发布了第一个版本，这就是最早的PHP 1.0，功能也十分简单。随着PHP的使用越来越广泛，大量开发人员开始加入到PHP的开发阵营中。在同年，PHP 2.0发布了。这一版加入了对MySQL的支持，从此PHP作为开发动态网页的卓越工具也得到了业内的一致认可。1997年和2000年分别又发布了PHP 3和PHP 4，大量新特性出现在这两个版本中。目前大多数人使用的主流版本是PHP 5，这是在2008年发布的。但无论你使用的是哪个版本，都会从本书中获益，因为本书的核心是基本原理，可以适用在所有版本上。

本书的结构很清晰，分为两大部分，Web应用程序的前端和后端。第一部分针对前端主要介绍了如何发现和消除浏览器在呈现网页过程中的瓶颈，此外还介绍了很多PHP编码的最佳实践以及如何运用缓存技术。第二部分则介绍后端，详细讲解了很多类型的Web服务器软件以及如何优化它们。

本书从基准测试技术开始讲起，开篇即引入了两个测试工具：Apache Benchmark 和Siege。然后由浅入深依次介绍了PHP代码优化、Opcode缓存、变量缓存、Apache Web服务器优化，以及最后的数据库优化。值得注意的是，基准测试几乎贯穿了整本书的所有章节，大部分章节中都穿插了ab和Siege的使用，并给出了详细的图解来说明测试的过程和结果，这是本书的一大特色。

毫无疑问，本书最适合PHP开发人员阅读，但同时也适用于对优化大型网络应用程序感兴趣的各类人员，包括项目经理、工程师和测试人员。

最后，衷心感谢人民邮电出版社图灵公司各位编辑在翻译工作中给予的帮助和宝贵意见。由于译者水平有限，在翻译过程中难免会出现一些错误，恳请读者批评指正。

谨以此书献给我的家人、朋友还有我的狗 Snoopy。

——Armando Padilla

谨以此书献给我的伙伴 Ester，他总是承受我埋头工作时的坏脾气，并忍受我残缺不全的回答。

——Tim Hawkins

引 言

如果你我是同道中人，你可能正在当地书店里拿起本书，或者正在网上读这份简介，尝试找找对这本书的感觉。要么你是一位好学的PHP工程师，急于探究构建大型应用程序的奥妙之处，要么你刚刚承担了一项支持高流量PHP应用程序的开发任务。这本书将非常适合像你这样对PHP有透彻了解并且熟悉这种语言的PHP开发人员——一位想深入研究PHP原理和工具集并且想揭开PHP脚本神秘面纱的人。

本书的目的是全面介绍在优化PHP应用程序时所需考虑的组件。它涵盖了所有这些组件，从JavaScript到正在运行应用程序的Web服务器软件。

本书分为两大部分，Web应用程序的前端和后端。第一部分介绍前端，帮助你确定在呈现过程中浏览器遇到的瓶颈以及如何消除这些瓶颈。这一部分还涉及PHP编码最佳实践的运用，如何使用很多可用的工具来应用缓存技术。第二部分介绍后端，介绍了很多类型的Web服务器软件，如何优化软件，以及优化数据库的技巧。

概述

下面是各章节内容的详细说明。

第1章：基准测试技术

我们首先确定测量应用程序性能所需的工具。我们将学习业界最受欢迎的两个基准测试工具的安装、结果读取和应用，这两个工具即Apache Benchmark（简称ab）和Siege。还将介绍如何使用并发以及特定时间段内的模拟负载来运行模拟负载实验。

第2章：提高客户端下载和呈现性能

应用程序性能不仅仅与PHP代码有关。这一章将重点介绍浏览器如何呈现内容。我们将学习对JavaScript进行基准测试的可用工具，测量浏览器尝试加载的数据量，以及查看浏览器加载内容的效率。我们还将学会如何通过安装和使用Firebug、Page Speed和Yahoo!的YSlow来完成这些工作。使用这些工具，我们可通过确认JavaScript、Image的性能是否提高来优化一个简单的网页。这一章内容比较浅显，所以并不要求你是JavaScript方面的专家。

第 3 章：PHP 代码优化

这一章开始研究PHP代码。我们将学习一些有助于提高PHP性能的最佳编码实践，还将了解如何构造一个快速运行的for循环，如何使用最佳PHP函数来包含文件，并且将重点了解如何使用和安装VLD、strace和Xdebug。在安装VLD和strace之后，就可以对Opcode以及运行PHP脚本所需的Apache C级别的处理进行分析了。使用Xdebug分析代码，我们将发现PHP代码自身的瓶颈所在。

第 4 章：Opcode 缓存

了解PHP的生命周期对于优化来说是非常重要的，所以这一章将介绍生命周期。我们将学习在用户请求期间PHP所执行的操作步骤，并找出有哪些地方可以使用Opcode缓存器进行优化。还将学习如何安装和配置Opcode缓存器，如APC、XCache和eAccelerator，同时还将对前后脚本进行基准测试，以便查看缓存Opcode后的收益。

第 5 章：变量缓存

在第4章介绍的缓存信息的基础上，我们将介绍变量缓存工具，如Memcached，以及使用APC存储信息。我们将学习如何安装、配置和实现一个简单的示例以便使你熟悉软件，另外还将介绍一个使用数据库结果集的实际示例。

第 6 章：选择正确的 Web 服务器

一直以来Apache都是一枝独秀，它是大型部署实际的标准。但最近该领域还出现了一些令人兴奋的新面孔。这一章将详细介绍Apache，并将其与新出现的Lighttpd和Nginx做一下对比。

第 7 章：优化 Web 服务器和内容交付

Apache是非常出色的Web服务器程序包，虽然它是开箱即用的，但稍加调整并且掌握一些技巧后，它的性能、持久性和稳定性会更高，它也会发挥出它真实的功能。这一章还将讨论如何对其进行扩展，以支持更高流量和用户负载的一些秘诀。

第 8 章：数据库优化

在大多数Web应用程序中，数据库服务器都发挥着极其重要的作用。这一章将讨论优化mysql数据库服务器的相关内容，同时还将提供使系统保持最佳状态的方法和工具。

致 谢

不言而喻，我要感谢Apress的工作人员Jennifer Blackwell和Michelle Lowman，他们给了我写这本书的机会。还要感谢数不清的开发人员和系统管理员，在很多个深夜，他们为我解答了无数个与本书主题相关的问题。感谢你们。

——Armando Padilla

感谢Rasmus Lerdorf，是他启动了这本书的写作工作，他也教给了我许多APC的高超技巧，还要感谢雅虎欧洲公司的前任同事们，他们教会了我要有远见。

——Tim Hawkins

目 录

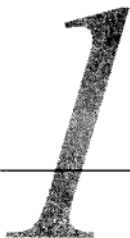
第 1 章 基准测试技术	1
1.1 PHP 应用程序栈	1
1.2 基准测试实用工具	2
1.3 定义请求/响应生命周期	3
1.4 Apache Benchmark	4
1.4.1 安装 Apache Benchmark	4
1.4.2 运行 Apache Benchmark	5
1.4.3 弄清响应的含义	6
1.4.4 ab 选项标记	8
1.4.5 ab 陷阱	11
1.5 Siege	12
1.5.1 安装 Siege	12
1.5.2 运行 Siege	13
1.5.3 分析结果	13
1.5.4 Siege 选项标记	15
1.5.5 测试很多 URL	15
1.6 影响基准测试数字	16
1.6.1 地理位置	16
1.6.2 旅行的数据包	16
1.6.3 响应的大小	16
1.6.4 代码复杂性	17
1.6.5 浏览器行为	18
1.6.6 Web 服务器设置	18
1.7 小结	19
第 2 章 提高客户端下载和呈现性能	20
2.1 优化响应的重要性	21
2.2 Firebug	21
2.2.1 安装 Firebug	22
2.2.2 Firebug 性能选项卡	22
2.2.3 Console 选项卡	23
2.2.4 Net 选项卡	25
2.3 YSlow	26
2.3.1 YSlow v2 规则集	26
2.3.2 安装 YSlow	27
2.3.3 启动 YSlow	28
2.4 Page Speed	30
2.4.1 安装 Page Speed	31
2.4.2 运行中的 Page Speed	31
2.5 优化工具	32
2.5.1 JavaScript 优化	33
2.5.2 JavaScript 的放置位置	33
2.5.3 精简 JavaScript	36
2.6 精简工具	37
2.7 YUI Compressor	38
2.8 Closure Compiler	38
2.8.1 减少资源请求	39
2.8.2 使用服务器端压缩	39
2.9 图像压缩	39
2.10 Smush.it	40
2.11 小结	42
第 3 章 PHP 代码优化	43
3.1 PHP 最佳实践	43
3.1.1 PHP 的经济性	45
3.1.2 require 与 require_once	45
3.1.3 提前计算循环长度	47
3.1.4 使用 foreach、for、while 循环访问数组元素	49
3.1.5 文件访问	50
3.1.6 更快速地访问对象属性	52
3.2 使用 VLD、strace 和 Xdebug 一探 究竟	54

3.2.1 用 VLD 查看 Opcode 函数	54	6.1.4 你在托管服务中托管	103
3.2.2 使用 strace 进行 C 级跟踪	56	6.1.5 你正在使用不常见的 PHP 扩展	103
3.3 发现瓶颈	58	6.2 Web 服务器的使用情况图表	103
3.3.1 Xdebug 2: PHP 调试工具	58	6.3 Web 服务器请求的处理	104
3.3.2 验证安装	60	6.4 Web 服务器硬件	105
3.3.3 安装基于 GUI 的工具	61	6.5 对 Web 服务器进行分类	106
3.4 小结	64	6.6 Apache HTTPD	106
第 4 章 Opcode 缓存	65	6.6.1 Apache Daemon 命令行	107
4.1 回顾路线图	65	6.6.2 Apache 多处理模块	108
4.2 PHP 的生命周期	66	6.7 了解 Apache 模块	109
4.3 Opcode 缓存工具	68	6.7.1 添加动态 Apache 模块	110
4.3.1 Alternative PHP Cache	68	6.7.2 删除动态 Apache 模块	110
4.3.2 XCache	75	6.8 关于 Apache 的最后几点	111
4.3.3 用 XCache 缓存	76	6.9 lighttpd	111
4.3.4 XCache 设置	77	6.9.1 安装 lighttpd	111
4.3.5 eAccelerator	78	6.9.2 lighttpd 配置设置	113
4.3.6 eA 设置	82	6.9.3 比较静态负载内容	114
4.4 小结	84	6.9.4 在 lighttpd 上安装 PHP	115
第 5 章 变量缓存	85	6.10 Nginx	118
5.1 应用程序的性能路线图	85	6.10.1 安装 Nginx	118
5.2 实现变量缓存的价值	86	6.10.2 Windows 安装	121
5.3 示例项目: 创建表	87	6.11 Nginx 作为静态 Web 服务器	122
5.3.1 获取记录	88	6.11.1 安装 FastCGI PHP	123
5.3.2 计算读取数据库的开销	89	6.11.2 Nginx 基准测试	124
5.4 APC 缓存	93	6.12 小结	126
5.4.1 将数据添加到缓存中	93	第 7 章 优化 Web 服务器和内容交付	127
5.4.2 对 APC 进行基准测量	94	7.1 测定 Web 服务器的性能	127
5.5 Memcached	96	7.2 了解应用程序的内存占用情况	129
5.5.1 安装 Memcached	96	7.3 优化 Apache 中的进程	130
5.5.2 启动 Memcached 服务器	97	7.3.1 控制 Apache 客户端 (Prefork MPM)	131
5.5.3 在 PHP 中使用 Memcached	97	7.3.2 优化内存使用和防止产生交 换	131
5.6 小结	101	7.4 其他 Apache 配置调整	131
第 6 章 选择正确的 Web 服务器	102	7.4.1 使用 .htaccess 文件和 AllowOverride	132
6.1 选择适合你的 Web 服务器程序包	103	7.4.2 使用 FollowSymLinks	133
6.1.1 安全性和稳定性非常重要	103	7.4.3 使用 DirectoryIndex	133
6.1.2 找到具有丰富知识的工程师非 常重要	103		
6.1.3 你的网站主要是静态内容	103		

7.4.4 关闭 HostnameLookup	133	8.2.2 InnoDB: 专业级的选择	147
7.4.5 启用 Keep-Alive	134	8.2.3 选择存储引擎	148
7.4.6 使用 mod_deflate 压缩 内容	134	8.3 了解 MySQL 如何使用内存	148
7.5 扩展到单台服务器之外	135	8.3.1 InnoDB 与 MyISAM 内存使 用的比较	149
7.5.1 使用 Round-Robin DNS	135	8.3.2 每服务器与每连接 (线程) 内存使用的比较	149
7.5.2 使用负载均衡器	135	8.4 查找配置文件	150
7.5.3 使用直接服务器返回	137	8.4.1 Mysqltuner.pl: 优化数据库 服务器的内存	151
7.5.4 在服务器场的成员之间共享 会话	138	8.4.2 示例服务器可能出现的问题	154
7.5.5 与共享文件系统共享资产	139	8.4.3 优化 InnoDB	155
7.5.6 与独立资产服务器共享资产	140	8.5 找到有问题的查询	155
7.5.7 与内容分发网络共享资产	140	8.6 分析有问题的查询	157
7.6 使用分布式架构的陷阱	141	8.7 PHP 数据库应用程序的建议	158
7.6.1 缓存一致性问题	141	8.7.1 保持独立的读写连接	158
7.6.2 缓存版本问题	141	8.7.2 默认使用“utf8”(多字节 Unicode) 字符集	158
7.6.3 用户 IP 地址跟踪	142	8.7.3 使用“UTC”日期格式	159
7.6.4 多米诺骨牌或级联失败效应	143	8.8 小结	160
7.6.5 部署失败	143		
7.7 监视应用程序	144	附录 A 在 Windows 上安装 Apache、 MySQL、PHP 和 PECL	161
7.8 小结	144	附录 B 在 Linux 上安装 Apache、 MySQL、PHP 和 PECL	174
第 8 章 数据库优化	145		
8.1 MySQL 简介	146		
8.2 了解 MySQL 存储引擎	146		
8.2.1 MyISAM: 原始引擎	147		

第1章

基准测试技术



电话响了，一个声音在另一端大声地说：“嘿！为什么这个应用程序不能支持200个并发用户呢？”你深吸一口气，尽你所能以一位资深PHP人员的语调低声回答：“奇怪，我看看出了什么问题，一会儿给你答复。”回想起这场对话的前几周。那时你的任务是构建一个由数据库驱动的PHP应用程序，据各方所说系统需求描述了一个简单的PHP应用程序。作为一名经验丰富的PHP开发人员，你开始编写代码，创建基本的体系结构层、PHP后端、CSS、JavaScript，并且还因为精通Photoshop，你还创建了图形布局并且发布了这个应用程序产品。

随着应用程序越来越受欢迎，网站访问用户与日俱增，很多抱怨也接踵而至。所有抱怨都指向了一个类似的问题，那就是网站无法响应或者响应速度太慢。现在，你看着代码，并考虑如何找到最后那点儿瓶颈和影响速度的代码并把它们除掉，不管问题到底是不是由代码引起的。你最终还是遇到了这个问题——当有50、100、200或300个并发用户同时请求主机上的文档时，你的Web应用程序的性能如何？此外，你如何在这样的流量负载下测试性能？

本章将深入考查两个用于基准测试的开源工具，它们不仅能够帮助回答这些问题，而且当我们对本书所讨论的应用程序进行性能增强时，它们还能测量性能的变化情况。我们要使用的这两个工具是Apache Benchmark (ab) 和Siege。

我们将学习如何安装这两个工具，读取结果以及使用工具来获取不同类型的内容——从简单的HTML到大型图像。最后还将学习一些其他的基础知识，包括HTTP请求/响应生命周期是如何被处理的，一个请求到底执行了什么操作，以及在请求主机资源的时候应该主要关注哪些可能引起延迟（也称作“lag”）的区域。

但首先让我们来了解一下PHP应用程序栈和贯穿本书始终的一个方法。

1.1 PHP 应用程序栈

每个PHP应用程序都有一个栈，可视化后，类似于图1-1。

大多数PHP应用程序都是在浏览器中显示给用户的，它们的显示使用了JavaScript (JS) 格式的前端代码、层叠样式表 (CSS)、Flash、其他前端技术以及诸如图像之类的资源。前端（如PHP应用程序栈的最上面一块所示）可帮助用户在Web应用程序中导航以及触发PHP层。PHP层包含特定于该应用程序的业务逻辑，并且在使用外部存储系统的情况下，它通常会与数据库或Web服

务交互以获取动态数据。最后，所有基于Web的PHP应用程序都有一个共同点：必须将它们安装在诸如Apache或Nginx这样的Web服务器上，而这些服务器也要安装在某个操作系统上。

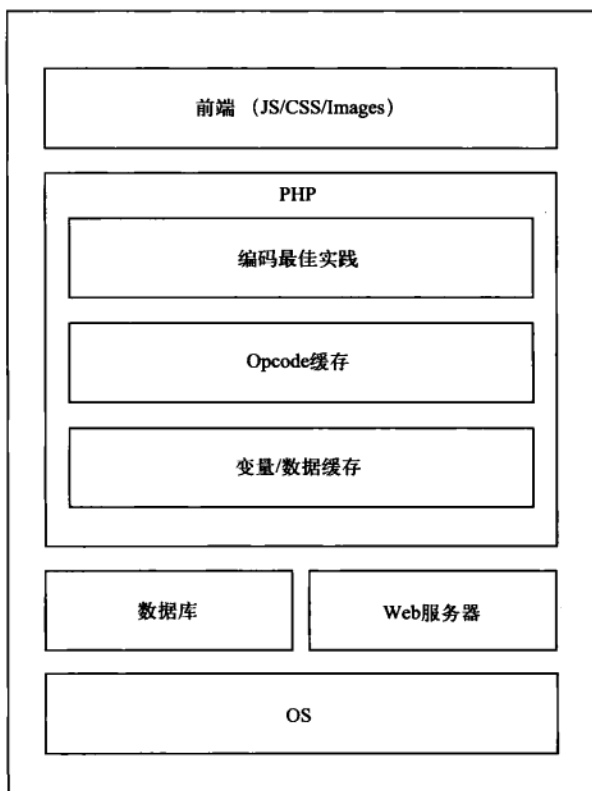


图1-1 PHP应用程序栈以及本书各章的划分

图1-1还表示出了本书所涉及内容的具体划分。PHP应用程序中的每一层都可以进行优化，而且每一层都是后续章的基础。从前端到Web服务器，本书将介绍图中所示的每一层，但我们需要一个工具，它不仅可用于测量当前未经修改的应用程序的执行情况，而且还可用于测量在对其性能进行提升之后它的执行情况。Apache Benchmark和Siege都满足了这样的要求。

1.2 基准测试实用工具

ab和siege同属于一组Web服务器基准测试工具，这些工具提供在各种不同的模拟用户请求发生时有关Web服务器响应的统计信息。它们允许我们模拟任意数量的请求Web服务器上某个特定Web文档的用户，更重要的是，它允许我们模拟任意数量的用户同时访问Web服务器上的文档

(并发请求)。

例如，每个工具提供的信息都与下列内容有关：

- 响应一个请求所花费的总时间；
- 来自服务器的总响应大小；
- Web服务器每秒可以处理的请求总数。

这些工具所不能完成的就是测试功能。这些工具仅能测试在特定Web服务器上运行的单个Web文档的请求。

之所以选择ab和siege，乃是由于以下原因。

- 易于使用：ab和siege都只有一行用于键入，同时只有少量选项。这意味着从一开始就很容易学习。
- 易于安装：它们都非常容易安装，并且只需要很少的安装时间。
- 基于命令行：大多数开发人员都在UNIX或Windows服务器上使用命令行。

1.3 定义请求/响应生命周期

让我们通过考查生命周期来快速探究一下HTTP请求/响应所执行的操作。首先，我们需要了解HTTP请求是什么，以及它所执行的操作，因为这些工具会利用一个请求的生命周期来帮助测量应用程序的性能。

HTTP请求是用户或工具尝试从Web服务器获取内容时所采取的操作。典型的HTTP请求包含请求正在尝试访问的主机信息、浏览器信息以及对Web服务器有用的其他信息。图1-2显示了用户个人计算机的HTTP请求/响应的过程。

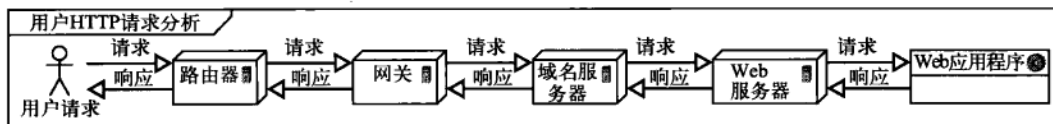


图1-2 HTTP请求生命周期

上图显示了一个简单的用户HTTP请求的过程，它向Web服务器请求获取内容。该请求是从用户的计算机生成的，依次要经过用户的主路由器（如果有的话）、ISP网关和域名服务器（DNS），并在DNS中会查找与请求的域名相关联的IP，然后到达具有指定IP的Web服务器，并最终请求Web应用程序生成特定内容。

该生命周期的第二部分是HTTP响应。一旦请求到达Web服务器之后，Web服务器便通过获取并格式化用户请求数据来准备响应；然后Web服务器就会将数据打包成多个数据包，并以相反的顺序沿着用户请求的相同路径将这些数据包发送给用户。如果数据足够大，则采用多个数据包发送，检查在传送期间是否有错误并由浏览器重建，然后浏览器便可开始其呈现过程。所有这些步骤必须发生在浏览器呈现网页之前。

这些步骤中的每个步骤都会使最终用户的网页执行速度减慢。接下来要讨论的工具可用来测试应用程序的响应时间，从而测试其最佳状态。

1.4 Apache Benchmark

Apache Benchmark (ab) 工具是最著名的基准测试工具之一，它是默认的Apache安装的一部分，能够通过模拟对特定URL的任意数量请求来对Web服务器进行负载测试。ab工具提供以下信息：

- 传输的总数据大小（以字节为单位）；
- Web服务器在模拟流量下每秒可以支持的请求总数；
- 完成一个请求所花费的最长时间（以毫秒为单位）；
- 完成一个请求所花费的最短时间（以毫秒为单位）。

使用ab工具还可以运行很多不同的负载模拟，例如：

- 对Web文档的模拟请求；
- 指定时间内的请求；
- 打开Keep-Alive时的请求。

最重要的是，Apache Benchmark是独立于Apache Web服务器的，从而可以在运行ab的同时使运行此工具的计算机上的Web服务器处于非活动状态。

1.4.1 安装 Apache Benchmark

在下面两节中，我们将介绍如何在基于Windows和Unix的系统上安装运行ab工具所需的文件。

1. Unix和Mac安装

如果用的是*nix操作系统，则会有很多安装Apache的选项。可以通过ports、yum、apt-get安装或只是下载源文件并安装。表1-1显示了安装命令的完整列表。

表1-1 使用存储库来安装Apache Web服务器

存 储 库	命 令
yum	yum install apache2
ports	sudo port install apache2
apt-get	apt-get install apache2

Mac用户可以在终端上使用MacPorts并执行表1-1所示的基于ports的命令。

2. Windows安装

Windows用户可在浏览器中打开<http://httpd.apache.org/>。加载此页之后，单击页面左侧的“Download from a mirror”（从镜像下载）链接，找到适合你的系统的相应下载程序包，即Windows 32 Binary版本，然后下载。编写本书时，最新的Apache版本为2.2.X。

当程序包下载完之后，就可以通过运行安装向导在系统的任意位置上安装该软件。我将Apache安装在默认位置C:\Program Files\Apache Software Foundation，但也可以安装在系统的任意位置。此处所选择的位置就是APACHE_HOME引用所指向的位置。

现在，打开目录<APACHE_HOME>\Apache2.2\bin。应该可以看到类似于图1-3的文件和目录的集合。

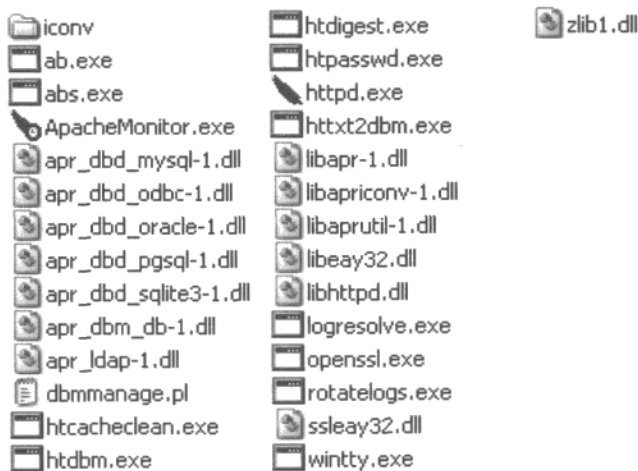


图1-3 安装好后的Windows Apache的bin目录

你已成功安装ab了工具，现在让我们使用它。

1.4.2 运行 Apache Benchmark

我们要运行的第一个基准测试是在www.example.com域上的简单测试。这个基本测试的主要目的就是让你熟悉以下工具的语法，查看所有的可用选项以及查看完整的响应。

所有ab命令的组成遵循此结构：

```
ab [options] [full path to web document]
```

我们将使用ab语法模拟单个请求。打开命令/shell终端并键入以下命令：

```
ab -n 1 http://www.example.com/
```

这条命令只使用了option部分的一个选项，即n，它表示要在指定的URL上执行的请求数。在这个示例中，ab只请求Web文档一次，但n的值可以是小于50 000的任意数字。默认情况下，n设置为1。

该命令的下一部分是URL部分。对于刚刚执行的ab命令，URL为http://www.example.com/。如果选择测试此域中的某个文档，如test.php（并不存在），则要测试的URL将为http://www.example.com/test.php。

让我们回到用于执行ab命令的命令/shell终端。到现在为止，你已经执行了这个命令，并且屏幕上满是ab工具所返回的数字和常规数据。你的输出应该类似于图1-4。

```
$ ./ab -n 1 http://www.example.com/
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 www.example.com (be patient).....done


Server Software:      Apache/2.2.3
Server Hostname:      www.example.com
Server Port:          80

Document Path:        /
Document Length:      574 bytes

Concurrency Level:    1
Time taken for tests:  0.094 seconds
Complete requests:    1
Failed requests:       0
Write errors:          0
Total transferred:    860 bytes
HTML transferred:     574 bytes
Requests per second:  10.67 [#/sec] (mean)
Time per request:     93.750 [ms] (mean)
Time per request:     93.750 [ms] (mean, across all concurrent requests)
Transfer rate:        8.96 [Kbytes/sec] received


Connection Times (ms)
      min   mean[+/-sd] median   max
Connect:    47    47   0.0      47    47
Processing:  47    47   0.0      47    47
Waiting:    47    47   0.0      47    47
Total:      94    94   0.0      94    94
```

图1-4 URL http://www.example.com的ab响应

注意 测试其他计算机时，请小心谨慎并且同时限制对Web服务器以及测试所发出的请求数量。我们都不想损害任何无问题的服务器而陷入真正的麻烦中。

1.4.3 弄清响应的含义

如果你头一次见识刚刚显示的输出中的响应，或者就算以前看到过，还是可能会觉得它有点难以理解。我们来看看其中一些非常重要的项目，以及一些在优化代码时有助于了解优化效果的项目。

图1-4中的数据分为4个主要部分，如图1-5所示。

1. 服务器信息

服务器信息部分包含Web服务器运行的软件。在我们的示例中，软件为Apache 2.2.3。数据包含在第一个字段即Server Software中。该字段的值可能会因为网站所使用的Web服务器软件而存在差异。由于Web管理员使用的安全措施，该字段的值也可能会返回一些你不熟悉的内容。

```

Server Software:      Apache/2.2.3
Server Hostname:      www.example.com
Server Port:          80

```

服务器信息

```

Document Path:        /
Document Length:      574 bytes

```

文档信息

```

Concurrency Level:    1
Time taken for tests:  0.094 seconds
Complete requests:    1
Failed requests:      0
Write errors:         0
Total transferred:    860 bytes
HTML transferred:     574 bytes
Requests per second:  10.67 [#/sec] (mean)
Time per request:     93.750 [ms] (mean)
Time per request:     93.750 [ms] (mean, across all concurrent requests)
Transfer rate:        8.96 [Kbytes/sec] received

```

连接信息

```

Concurrency Level:    1
Time taken for tests:  0.094 seconds
Complete requests:    1
Failed requests:      0
Write errors:         0
Total transferred:    860 bytes
HTML transferred:     574 bytes
Requests per second:  10.67 [#/sec] (mean)
Time per request:     93.750 [ms] (mean)
Time per request:     93.750 [ms] (mean, across all concurrent requests)
Transfer rate:        8.96 [Kbytes/sec] received

```

连接指标细目分类

图1-5 ab结果的各个部分

下面两个字段（即Server Hostname和Server Port）含有我们在其上运行模拟的主机名和Web服务器侦听的端口号。

2. 脚本信息

ab响应的第二部分包含有关运行模拟的Web文档的信息。Document Path包含请求的文档，而Document Length则包含所有HTML、图像、CSS、JS以及响应中任何内容的字节数总和。

3. 连接信息

连接信息部分包含信息的主体。它回答了诸如“请求收到响应需要多长时间？”、“返回了多少数据？”之类的问题，最重要的是它回答了“处理文档时Web服务器可以支持多少用户？”。

表1-2提供了此部分数据的完整列表和描述。现在，让我们看一看突出显示的行，这些行包含我们最关心的字段。

表1-2 ab响应描述

字 段	描 述	示 例 值
Concurrency Level	所进行的并发请求总数	1,2,3,...,n, 其中n为任意数字
Time taken for tests	运行所花费的总时间	000.000秒
Complete requests	模拟的请求总数中已完成的请求总数	1,2,3,...,n, 其中n为任意数字

(续)

字 段	描 述	示 例 值
Failed requests	模拟的请求总数中失败的请求总数	1,2,3,...,n, 其中n为任意数字
Write errors	使用写入数据时遇到的错误总数	1,2,3,...,n, 其中n为任意数字
Non-2xx responses	未收到HTTP成功响应的请求总数(200)	1,2,3,...,n, 其中n为任意数字
Total transferred	整个模拟的响应中传输的总数据, 大小包 括标头数据	725个字节
HTML transferred	整个模拟传输的内容正文的总大小	137 199个字节
Requests per second	每秒支持的请求总数	5.68 [# /秒] (平均值)
Time per request	满足一个请求需要花费的总时间	176.179毫秒
Time per request	满足所有并发请求中的一个请求需要花 费的总时间	176.179毫秒
Transfer rate	每秒收到的字节总数 (KB)	766.27 [KB/秒]

HTML transferred、Requests per second以及Time per request都是关键字段。这些字段使我们能够大概了解Web服务器为一个请求返回的数据量、Web服务器一秒可以处理的请求总数以及一个请求成功地收到来自Web服务器的响应所花费的总时间。

在本书中, 我们的目标是成功降低HTML transferred, 提高Requests per second并且降低Time per request值。

4. 连接指标细目分类

最后一个部分包含一个表, 其中包含Connect、Processing、Waiting以及Total字段。这些字段告诉我们请求在每个过程状态中所需的时间。我们最感兴趣的是Total字段及其最大、最小值列。这两列提供响应一个请求所需花费的最长和最短时间的数据。下面让我们看看ab为我们提供的可选标记。

1.4.4 ab 选项标记

ab包含大量有用的可选标记。使用这些标记, 可以将为HTML表格格式化响应、设置cookie、设置基本身份验证信息以及设置内容类型, 另外还有其他选项。表1-3显示了可选标记的完整列表。

表1-3 可选标记

标 记	描 述
-A <username>:<password>	用于提供服务器身份验证信息。用户名和密码用“:”分隔。发送的字符串采用base64编码
-c <concurrency number>	一次模拟的请求数。默认情况下设置为1。数量不得大于n值
-C cookie-name=value	可重复的标记, 包含cookie信息
-d	隐藏 “percentage served within XX[ms] table”

(续)

标 记	描 述
-e	要创建的.csv文件的路径。该文件包含运行的基准测试的结果,该结果分为两列,即Percentage和Time in ms。建议采用“gnuplot”文件
-g	要创建的“gnuplot”或TSV文件的路径。基准测试的输出将保存到该文件中
-h	显示要用于ab的选项列表
-H custom-header	采用字段值对形式发送有效标头和请求
-i	执行HEAD请求,而不是默认的GET请求
-k	启用Keep-Alive功能。允许通过一个HTTP会话满足多个请求。默认情况下,该功能处于禁用状态
-n requests	要执行的请求总数
-p POST-file	包含用于HTTP POST请求的数据的文件路径。内容应该包含由&分隔的键=值对
-P username:password	采用Base64编码的字符串。字符串包含基本身份验证,以及由“:”分隔的用户名和密码
-q	执行多于100个请求时隐藏进度输出
-s	使用https协议,而非默认的http协议——不建议这样做
-S	隐藏中位数和标准偏差值
-t timelimit	指定了这个值以后,基准测试的时间不会超过指定的值。默认情况下无时间限制
-v verbosity-level	数值为2及以上将打印警告和信息;为3将打印HTTP响应代码;4及以上将打印标头信息
-V	显示ab工具的版本号
-w	采用HTML表格打印结果
-x <table-attributes>	表示HTML属性的字符串,使用-w时将放置在<table>标记中
-X proxy[:port]	指定要使用的代理服务器。代理端口是可选的
-y <tr-attributes>	表示HTML属性的字符串,使用-w时将放置在<tr>标记中
-z <td-attributes>	表示HTML属性的字符串,使用-w时将放置在<td>标记中

为了实现优化PHP脚本的目标,我们需要将少数的选项调到零点。这些选项如下所示。

- n: 要模拟的请求数;
- c: 要模拟的并发请求数;
- t: 执行模拟所需要的时间。

初始安装ab之后,使用标记n运行模拟。现在让我们使用其他标记并看一看www.example.com网站包含的初始基准测试数据。

1. 并发测试

根据Web应用程序的不同,用户在应用程序上的时间可以从几秒钟到几分钟不等。如果你的站点非常吸引人,或者是受到了恶意用户的DOS攻击,进入的用户流可能大幅波动,从很小的流量到很高的流量。你需要模拟真实的流量来测试网站是否能够应付这样的情况。

我们将模拟一个并发测试，同时对Web服务器进行10个并发请求，直到进行到100个请求为止。使用c标记时的一项警告是让使用的值小于要进行的请求总数n。如果值等于n，则只会并发请求所有n个请求。为此，我们执行如下命令。

```
ab -n 100 -c 10 http://www.example.com/
```

运行该命令之后，应该得到一个类似于图1-6的响应。

```
$ ./ab -n 100 -c 10 http://www.example.com/
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 www.example.com (be patient).....done

Server Software:      Apache/2.2.3
Server Hostname:      www.example.com
Server Port:          80

Document Path:        /
Document Length:      438 bytes

Concurrency Level:    10
Time taken for tests:  4.469 seconds
Complete requests:    100
Failed requests:       0
Write errors:          0
Total transferred:    72500 bytes
HTML transferred:     43800 bytes
Requests per second:  22.38 [#/sec] (mean)
Time per request:     446.875 [ms] (mean)
Time per request:     44.688 [ms] (mean, across all concurrent requests)
Transfer rate:         15.84 [Mbytes/sec] received

Connection Times (ms)
      min      mean[+/-sd] median   max
Connect:    31      44  14.4      47     94
Processing:  47    379  72.1     391    500
Waiting:    31    215  122.7    219    500
Total:      94    423  69.5     438    547

Percentage of the requests served within a certain time (ms)
 50%    438
 75%    453
 80%    469
 90%    469
 95%    500
 98%    531
 99%    547
100%    547 (longest request)
```

图1-6 www.example.com的并发模拟结果

通过模拟的并发请求，我们可以看到Request per second字段，并注意到Web服务器每秒可以支持22.38个请求（用户）。分析Connection Metrics的Total min和max列，我们注意到在10个并发请求的指定流量负载下，最快的响应为94毫秒，而最慢的请求花了547毫秒。

但是，我们知道流量不会只是持续1、2或3秒，高流量可能会持续几分钟、几小时，甚至是几天。下面让我们运行一个模拟来测试一下。

2. 时间测试

我们注意到每天接近中午时，网站会达到流量高峰，这个高峰会持续10分钟。在这种情况下，Web服务器执行得如何？我们将要使用的下一个标记就是t标记。使用t标记，可以检查在任何时

间内Web服务器执行的情况。

下面我们使用以下命令来模拟10个用户在20秒的时间内同时访问网站：

```
ab -c 10 -t 20 http://www.example.com/
```

该命令不包含n标记，但默认情况下包含该标记并且使用t选项时ab将其值设置为50 000。某些情况下，使用t选项时，最大请求数可能会达到50 000，此时模拟结束。

此ab命令完成其模拟之后，将会获得类似于图1-7所示的数据。

```
$ ./ab -c 10 -t 20 http://www.example.com/
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 www.example.com (be patient)
Finished 427 requests


Server Software:      Apache/2.2.3
Server Hostname:      www.example.com
Server Port:          80

Document Path:        /
Document Length:       438 bytes

Concurrency Level:    10
Time taken for tests:  20.125 seconds
Complete requests:    427
Failed requests:       0
Write errors:          0
Total transferred:    311025 bytes
HTML transferred:     187902 bytes
Requests per second:  21.22 [#/sec] (mean)
Time per request:     471.311 [ms] (mean)
Time per request:     47.131 [ms] (mean, across all concurrent requests)
Transfer rate:        15.09 [Kbytes/sec] received


Connection Times (ms)
              min      mean[+/-sd] median    max
Connect:        31       45   14.7      47     141
Processing:    281      412   88.8     391    1828
Waiting:        31      278  113.0     297     656
Total:         328      457   90.8     438    1859


Percentage of the requests served within a certain time (ms)
 50%    438
 66%    453
 75%    469
 80%    484
 90%    531
 95%    594
 98%    641
 99%    688
100%   1859 (longest request)
```

图1-7 www.example.com/的基准测试结果（20秒内10个并发用户）

此模拟的结果指出，当10个并发用户在20秒的时间段内请求Web文档时性能会下降。最快速的满足请求花费了328毫秒，而最慢速的请求花费了1859毫秒（1.8秒）。

1.4.5 ab 陷阱

使用ab时有几点警告。如果你再看看刚刚执行的这个命令，就会注意到在域名的结尾有一个反斜杠。如果你不请求该域中的特定文档，则这个反斜杠是必需的。ab还可能会由于其传递给

Web服务器的用户代理值而被某些Web服务器阻止，因此在某些情况下，可能收不到任何数据。如果要解决后面这个问题，请使用可用的选项标记之一-H，来提供你的请求中的自定义浏览器标头信息。

若要通过Chrome浏览器模拟请求，可以使用以下ab命令：

```
ab -n 100 -c 5 -H "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/534.2 (KHTML, like Gecko) Chrome/6.0.447.0 Safari/534.2" http://www.example.com
```

1.5 Siege

我们将使用的第二个基准测试工具为Siege。与ab一样，Siege可以模拟Web托管文档的用户流量，但与ab不同的是，Siege可以对文本文件中指定的URL列表运行负载测试。它还可以在执行其他请求之前让某个请求休眠，从而让你感觉某个用户在转移到Web应用程序的下一个文档之前正在读取该文档。

1.5.1 安装 Siege

可以通过从官方网站www.joedog.org/index/siege-home或<http://freshmeat.net/projects/siege>下载源代码或者使用存储库（如port或aptitude）以及以下所示的命令之一安装Siege：

```
sudo port install siege
```

或者

```
sudo aptitude install siege
```

使用其中一个命令，Siege将自动安装所有必需的程序包。在编写本书时，Siege的最新稳定版本为2.69。

很遗憾，如果没有Cygwin，Windows用户将不能使用Siege。如果你使用的是Windows，请先下载Cygwin并安装该软件，然后再尝试安装和运行Siege。安装完Cygwin之后，使用本部分中概述的步骤安装Siege。

如果你决定使用源代码安装，可能会在下载程序包时遇到问题，此时，请打开一个终端窗口并键入以下内容。

```
wget ftp://ftp.joedog.org/pub/siege/siege-latest.tar.gz
```

该命令会将此程序包下载到你的系统上。下载完此程序包之后，执行以下命令：

- ❑ `tar xvfz siege-latest.tar.gz`
- ❑ `cd siege-2.69/`
- ❑ `./configure`
- ❑ `make`
- ❑ `sudo make install`

这些命令将配置源代码、创建安装程序包并最终在系统上安装此程序包。安装完成后，将你的目录位置更改为/usr/local/bin/。你应该会在此目录中看到Siege脚本。

下面，我们继续进行并在域www.example.com上运行一个简单测试，以便查看示例结果。

1.5.2 运行 Siege

第一个示例是对www.example.com的一个简单负载测试。与ab一样，Siege遵循特定的语法格式。

```
siege [options] [URL]
```

使用Siege格式，我们将模拟一个负载测试，5个并发用户在10秒内访问网站www.example.com。需要说明一下，使用Siege时的并发被称为事务。因此我们要模拟的测试是使用Siege命令让Web服务器在10秒的时间内一次满足5个同时发生的事务：

```
siege -c 5 -t10S http://www.example.com/
```

该命令使用两个选项标记：并发标记c和时间标记t。使用并发标记，可以通过X（在本例中为5）个用户同时访问网站来测试请求。数字可以是任意数字，只要运行测试的系统可以支持这样一个任务即可。t标记以秒（S）、分钟（M）或小时（H）为单位指定时间，并且数字和字母之间没有空格。

运行此命令之后，你应该会看到类似于图1-8的输出。

```
Lifting the server siege...      done.
Transactions:                  102 hits
Availability:                  100.00 %
Elapsed time:                   9.71 secs
Data transferred:              0.04 MB
Response time:                 0.02 secs
Transaction rate:              10.50 trans/sec
Throughput:                    0.00 MB/sec
Concurrency:                   0.24
Successful transactions:       102
Failed transactions:           0
Longest transaction:           0.03
Shortest transaction:          0.02
```

图1-8 www.example.com的Siege响应（10秒内5个并发请求）

1.5.3 分析结果

与ab结果一样，Siege工具的结果也分为几个部分。具体来说，结果集有两个部分需要分析：

- 单个请求详细信息；
- 测试指标。

1. 单个请求详细信息

单个请求详细信息部分显示了该工具创建和运行的所有请求。每一行都代表一个唯一的请求

并且包含三列，如图1-9所示。

```

** SIEGE 2.69
** Preparing 5 concurrent users for battle.
The server is now under siege...
HTTP/1.1 200 0.03 secs: 438 bytes ==> /
HTTP/1.1 200 0.03 secs: 438 bytes ==> /
HTTP/1.1 200 0.02 secs: 438 bytes ==> /
HTTP/1.1 200 0.03 secs: 438 bytes ==> /

```

图1-9 Siege请求数据

该输出包含运行的初始Siege命令的请求示例。各列所代表的内容如下：

- HTTP响应状态代码；
- 完成请求所需的总时间；
- 作为响应所收到的数据总量（不包括标头数据）。

2. 测试指标

测试指标部分包含有关整个负载测试的信息。表1-4列出并描述了所有字段，你可以仔细查看一下。我们只对Data transferred、Transaction rate、Longest transaction以及Shortest transaction感兴趣。我们将重点介绍结果中的这些特定属性，因为它们显示出了优化应用程序的效果。

表1-4 Siege测试指标部分描述

字段名称	描 述	示 例 值
Transactions	已完成的事务总数	102 hits
Availability	能够请求Web文档的时间	100.00%
Elapsed Time	完成测试所需的总时间	9.71 secs
Data transferred	响应中数据的总大小（不包括标头数据）	0.04M
Response time	整个测试过程中的平均响应时间	0.02 secs
Transaction rate	每秒要满足的事务总数	10.50 trans/sec
Throughput	处理数据和响应所需的总时间	0.00 MB/sec
Concurrency	Concurrency是同时连接的平均数，该数字升高时服务器性能降低	5
Successful transactions	整个测试过程中所执行的成功的事务总数	102
Failed transactions	整个测试过程中遇到的失败的事务总数	0
Longest transaction	满足一个请求所需的最长时间	0.03
Shortest transaction	满足一个请求所需的最短时间	0.02

Data transferred 部分包含每个请求收到的响应的总大小（以MB为单位）。Transaction rate帮助我们了解当Web服务器在我们命令指定的负载下运行时可以满足的并发事务数（同时发生的请求）。在本例中，Web服务器在10秒之内5个并发请求的负载下时，Web服务器可以每秒满足10.50个事务。

Shortest transaction和Longest transaction字段告诉我们满足一个请求所需的最短时间（以秒为单位）以及满足一个请求所需的最长时间（以秒为单位）。

1.5.4 Siege 选项标记

Siege也包含很多可选标记，如果感兴趣，可以使用以下命令查看这些标记：

```
siege -h
```

1.5.5 测试很多 URL

下面我们重点介绍两个新的标记：“internet”标记（i）和“file”标记（f）。

当使用t和i标记时，Siege在某个文本文件中随机选择一个URL并请求该Web文档。尽管无法保证访问该文本文件中的所有URL，但是能够保证这是一个真实的测试，用于模拟网站上的用户活动。

为了指定要使用的文件，我们使用标记f。默认情况下，Siege使用的文件位于SIEGE_HOME/etc/urls.txt中，但你也可以更改该路径，方法是将该标记设置为文本文件的位置。

URL格式和文件

下面我们将使用两个命令来执行接下来的测试。在系统的任意位置创建一个测试文件。将文件放在HOME_DIR/urls.txt下并在该文件中放置三个URL，格式遵循代码清单1-1所示的Siege URL格式。代码清单1-2显示了完整的示例urls.txt文件。

代码清单1-1 Siege URL格式结构

```
[protocol://] [servername.domain.xxx] [:portnumber] [/directory/file]
```

代码清单1-2 urls.txt文件

```
http://www.example.com/  
http://www.example.org/  
http://www.example.net/
```

三个URL位于三个不同的域中。正常情况下不会采用这种方式，而是在列表中将要请求的Web文档置于同一域中。

下面我们使用以下命令运行此测试：

```
siege -c 5 -t10S -i -f HOME_DIR/urls.txt
```

正如你所看到的，输出与图1-8非常类似，唯一的不同在于要测试的URL是从urls.txt文件中随机选择的。

现在已经运行了ab和Siege，你可能想知道哪些因素会影响这些数字。接下来，我们将介绍一下这方面的内容。

1.6 影响基准测试数字

最终影响响应时间并影响基准测试数字的五个方面如下：

- 地理位置和网络问题；
- 响应大小；
- 代码处理；
- 浏览器行为；
- Web服务器配置。

1.6.1 地理位置

Web服务器的地理位置对于用户体验到的响应时间非常重要。如果Web服务器位于美国，而用户位于中国、欧洲或拉丁美洲，那么请求到达目的地需穿越很远的距离，再等待Web服务器获取该文档，然后再回到位于这些国家的用户又要穿越遥远的距离，这些都会影响你感知到的Web应用程序的速度。

问题是路由器、服务器的总数，并且在某些情况下还必须穿洋过海才能到达目的地——在本例中，目的地是你的网站。用户通过的路由器/服务器越多，请求到达Web应用程序所花的时间就越长，并且Web应用程序的响应到达用户所花的时间也越长。

1.6.2 旅行的数据包

某些情况下，数据包也会产生开销。如前所述，当采用数据包（可管理的小块数据）的形式将Web服务器的响应发送回用户时，用户的系统必须检查是否有错误，然后再重构该消息。如果任何一个数据包包含错误，则会对Web服务器进行自动请求，从发现错误的数据包开始请求所有数据包——这样便迫使你考虑数据的大小。数据越小，服务器需要创建以及发送回用户的数据包数量就越少。

1.6.3 响应的大小

下面我们看一看数据大小如何影响数据到达其目的地所花费的时间。如果我们的网站向页面呈现1MB的内容，这意味着Web服务器需要将1MB的数据发送给用户——这需要相当多的数据包！根据用户的连接速率，对于非常小的内容，完成请求所需的响应时间要少得多。

为了说明这一点，我们将对较大图像的请求和较小图像的请求进行基准测试，并比较它们的响应时间。

用于获取较大图像的ab命令如下：

```
ab -n 1 http://farm5.static.flickr.com/4011/4225950442_864042b26a_s.jpg
```

用于获取较小图像的ab命令如下：

```
ab -n 1 http://farm5.static.flickr.com/4011/4225950442_864042b26a_b.jpg
```

当我们分析图1-10和1-11所示的响应信息时，有三个项目非常引人注目，它们是：Document Length、Total min 和Total max时间。与较大图像的请求相比，满足较小图像的请求花费更少的时间，如Total max和Total min值所示。换句话说，用户请求的数据越小，响应速度越快。

```
Document Path:      /4011/4225950442_864042b26a_s.jpg
Document Length:    3407 bytes

Concurrency Level:  1
Time taken for tests: 0.188 seconds
Complete requests:  1
Failed requests:     0
Write errors:        0
Total transferred:  3906 bytes
HTML transferred:   3407 bytes
Requests per second: 5.33 [#/sec] (mean)
Time per request:    187.500 [ms] (mean)
Time per request:    187.500 [ms] (mean, across all concurrent requests)
Transfer rate:       20.34 [Kbytes/sec] received

Connection Times (ms)
      min    mean[+/-sd] median    max
Connect:    78      78   0.0      78     78
Processing:  94      94   0.0      94     94
Waiting:     94      94   0.0      94     94
Total:      172     172   0.0     172    172
```

图1-10 对较小图像请求的响应

```
Document Path:      /4011/4225950442_864042b26a_b.jpg
Document Length:    308235 bytes

Concurrency Level:  1
Time taken for tests: 1.109 seconds
Complete requests:  1
Failed requests:     0
Write errors:        0
Total transferred:  308731 bytes
HTML transferred:   308235 bytes
Requests per second: 0.90 [#/sec] (mean)
Time per request:    1109.375 [ms] (mean)
Time per request:    1109.375 [ms] (mean, across all concurrent requests)
Transfer rate:       271.77 [Kbytes/sec] received

Connection Times (ms)
      min    mean[+/-sd] median    max
Connect:    63      63   0.0      63     63
Processing: 1047    1047  0.0    1047    1047
Waiting:     94      94   0.0      94     94
Total:      1109    1109  0.0    1109    1109
```

图1-11 对较大图像请求的响应

在后面的章节中，我们将介绍如何减小响应数据大小，方法是分析Web应用程序的内容（无论是图像、JavaScript文件还是CSS文件），以确定你可以最小化以及优化的内容和位置。

1.6.4 代码复杂性

文档必须执行的逻辑也会影响响应。在我们的初始测试中，这并没有成为一个问题，因为我们测试的是非常简单的静态HTML页面，但随着我们添加PHP（用于交互的数据库和/或要调用的

Web服务),我们无意中增加了满足请求所花费的时间,因为每个外部交互和PHP进程都会产生开销。在后面的章节中,我们将介绍如何减少这些开销。

1.6.5 浏览器行为

浏览器对网站的响应性也起到举足轻重的作用。每个浏览器都有自己呈现JavaScript、CSS和HTML的方法,而这会使用户体验的总响应时间增加几毫秒,甚至几秒。

1.6.6 Web 服务器设置

最后,Web服务器及其配置也可能会增加响应请求所花费的时间。默认情况下(开箱即用),大多数Web服务器都不包含最优设置并且需要技术精湛的工程师来修改配置文件和内核设置。为了测试对Web服务器的简单改进,我们需要超前一点,在启用Keep-Alive设置的同时测试Web服务器。我们将在后面的章节中对Web服务器配置进行更详细的讨论。

启用Keep-Alive设置,Web服务器可以打开特定数量的连接,然后服务器可以保持这些连接处于打开状态以满足其他传入的请求。这样便不再需要Web服务器为每个传入请求打开一个连接,满足请求后再关闭该连接,通过减少这样的开销可以加快应用程序的速度并且减少Web服务器必须处理的进程数量,从而增加了可以支持的用户数。

下面让我们捕获可用于比较的基准数据。运行以下命令:

```
ab -c 5 -t 10 http://www.example.com/
```

之后再运行此命令:

```
ab -c 5 -t 10 -k http://www.example.com/
```

此命令包含Keep-Alive标记k。该标记允许Web服务器保持5个并发连接处于打开状态并允许其他连接通过它们,从而减少Web服务器创建新连接所花费的时间。相连的图1-12和图1-13展示了它们的比较。

```

Concurrency Level:      5
Time taken for tests:    10.063 seconds
Complete requests:      184
Failed requests:         0
Write errors:            0
Total transferred:      133400 bytes
HTML transferred:       80592 bytes
Requests per second:    18.29 [#/sec] (mean)
Time per request:       273.438 [ms] (mean)
Time per request:       54.688 [ms] (mean, across all concurrent requests)
Transfer rate:          12.95 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median   max
Connect:    16    53 220.9   31    3031
Processing: 125   214 437.3   141   3141
Waiting:    31   142 385.6    94   3141
Total:      156   267 487.7   188   3188

```

图1-12 10秒内5个并发用户的ab测试结果


```

Concurrency Level:      5
Time taken for tests:    10.000 seconds
Complete requests:      1412
Failed requests:         0
Write errors:            0
Keep-Alive requests:    1412
Total transferred:      1023700 bytes
HTML transferred:       618456 bytes
Requests per second:    141.20 [#/sec] (mean)
Time per request:       35.411 [ms] (mean)
Time per request:       7.082 [ms] (mean, across all concurrent requests)
Transfer rate:          99.97 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median   max
Connect:    0      0  2.5      0    47
Processing: 16    35 56.1     31  1234
Waiting:    16    35 56.1     31  1234
Total:      16    35 56.3     31  1234

```

图1-13 使用Keep-Alive的ab测试结果

对比这两个图并参考Requests per second、Total min和Total max，可以清晰地看出使用Keep-Alive大大增加了Web服务器每秒可以满足的请求数，同时还加快了响应时间。

有了用于度量优化代码是否成功的工具作为坚实的基础，下面将开始进行性能优化。

1.7 小结

本章的目标是介绍可用于执行基准测试的工具，在后面的章节中，我们再介绍用于特定优化目的的每种工具的重要特点。

本章介绍了如何使用、安装Apache Benchmark和Siege工具以及如何分析数据，还介绍了影响基准测试数字，转而影响用户请求的响应时间的4个主要项目。最后学习了有关HTTP请求生命周期的内容，并知道了通过了解HTTP请求的处理过程也会有助于你进行优化。

在上一章中，你已经学到了如何使用Apache Benchmark（简称ab）和Siege工具来测量响应时间，并确定网页在不同流量负载下响应的快慢。有了这些结果，就可以确定Web服务器能否在承受大量负载的时候支持足够多的页面，也可以确定响应速度的快慢。如果生活简单到每个人都只使用一种终端来访问网页，那么我们只需保留这些结果并跳过这一章。遗憾的是，现实要复杂得多，Web浏览器也是如此。现在，我们要重点讨论一个基本的组件，它也是对任何在线PHP应用程序都极为关键的组件，那就是应用程序的前端。在本章中，我们将着重研究你的应用程序在用户浏览器中的性能表现。

如图2-1所示，前端是PHP应用程序的第一层，也是我们在本章中将会讨论的组件。我们需要首先优化这个组件，因为它是用户访问应用程序时接触到的第一项技术。



图2-1 PHP应用程序组件栈

在本章中，我们将从Web浏览器的角度分析Web服务器的响应。具体来说，我们将使用Firefox浏览器中的可用工具来分析当浏览器请求某个页面时，Web服务器发送的响应。

本章将重点介绍以下几种工具。第一个工具集（Firebug、YSlow和Page Speed）可提供以下信息来帮助我们分析响应：

- Web服务器发送的响应的细节信息；
- 分析JavaScript中的前端逻辑；
- 浏览器将读取的资源的逐项列表；
- 浏览器获取和接收资源所花费的时间；
- 对如何优化响应的建议。

第二个工具集（YUI Compressor、Closure Compiler和Smush.it）将帮助我们优化响应。简言之，我们可以使用这些工具压缩JavaScript、CSS文件以及网页所需的图像。

在讲述这两个工具集时，我们将介绍如何安装这些工具并读取结果。完成工具的安装后，将采用一些建议来优化服务器的响应。

2.1 优化响应的重要性

事先说明一下，本章不要求你详细了解JavaScript（JS）、层叠样式表（CSS）或图像优化。如果你对这个话题感兴趣，市场上有许多关于JS和CSS优化的优秀图书，这里你要做的是优化PHP！在这个大前提下，我们将讨论影响Web浏览器呈现时间的因素，以及可快速应用于网页并且效果显著的高级优化技术。阅读本章之前你需要知道的是，每种技术在Web应用程序中起到什么作用，它们都放置在网页的什么位置，以及来自Web服务器的响应可能（经常）会涉及对这些资源的引用。只有这些。

那么，为什么用一整章的篇幅来讲解如何测量和优化这些技术以及来自Web服务器的响应呢？因为如果不优化响应，用户会始终认为你的网页不够快。

例如，某个用户正在加载一个网页，该网页的总大小为3MB。响应中含有30个未缓存的大图片、臃肿的CSS和无数个JavaScript文件，而这些东西网页根本就不需要。不管你在优化PHP代码方面付出了多少努力，取得了多少进步，用户依然需要完成3MB下载的响应才能看到这个网页。如果使用标准的DSL调制解调器（1Mbit/s），那么下载一个3MB的文件需要1分钟。Juniper Research的一项调查表明，用户等待页面加载的平均时间最多为4秒钟。所以，在1分钟的下载时间中，56秒是难以忍受的，这会让你失去这个用户。

2.2 Firebug

正如任何Web开发人员/设计人员会告诉你的一样，在创建跨浏览器应用程序时，Firebug曾不止一次地帮助他们渡过难关——我知道我是这样的。Firebug是Web开发人员广泛使用的一个插件，可以提供关于网页的DOM、CSS、JavaScript的详细信息以及对响应优化最为重要的资源请求信息。该插件会检索你目前浏览的具体网页上的相关信息，并在不同的选项卡中展示这些信息。

注意 Firebug中含有许多对网页设计人员大有帮助的选项卡，但其超出了本书的范围。如果了解更多有关该工具的信息，请访问<http://getfirebug.com/>，获取Firebug可用功能的完整列表。

2.2.1 安装 Firebug

在写这本书的时候，Firebug的最新版本是1.5.x，它只能安装在Firefox 3.5~3.6的浏览器中。该插件有一个轻量级版本（FirebugLite），其中包含了支持其他浏览器的功能有限子集，但我还是推荐安装完整版本。

要安装此插件，请打开Firefox浏览器，并通过以下链接打开Firebug首页或Mozilla附加组件Firebug页面。

□ Firebug首页：<http://getfirebug.com/>

□ Mozilla附加组件Firebug页面：<https://addons.mozilla.org/en-US/firefox/addon/1843/>

网页加载后，单击“Install Firebug for Firefox”（在Firefox上安装Firebug）或“Add to Firefox”（添加Firefox）按钮，然后在弹出的安装窗口中单击“Allow”（允许）按钮。安装完Firebug后，需要重启浏览器才能使用该插件。

2.2.2 Firebug 性能选项卡

成功安装Firebug后，单击浏览器窗口右下角的小虫子图标启动Firebug，如图2-2所示。



图2-2 在Firefox中启动Firebug

此操作将启动Firebug并打开控制台窗口。你会注意到，有许多选项卡可供使用，它们可以让你深入了解任何网页的结构和样式。我们将重点讨论两个选项卡：Console和Net，如图2-3所示。

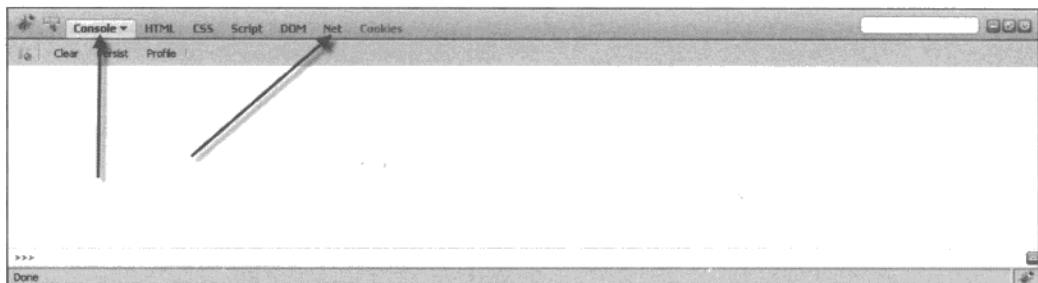


图2-3 Firebug的Console和Net选项卡位置

2.2.3 Console 选项卡

Console选项卡显示目前浏览的网页上所有的JavaScript日志信息、警告或错误。在某些情况下，Web开发人员可通过这个窗口使用`console.log()`追踪自己的JavaScript。此选项卡还提供了一种分析此页面上执行的JavaScript的方式，我们很快就能看到。

JavaScript的分析结果将显示在一个含有9栏的表格中。每栏名称及其详细描述完整列表请参见表2-1。

表2-1 Firebug JavaScript 性能分析工具的栏描述

栏 名 称	描 述
Function	所调用的函数
Calls	调用页面上特定函数的总次数
Percent	该函数内部所花费时间在全部函数集合所花费时间中所占的百分比
Own Time	特定函数内部所花费的时间（以毫秒为单位）
Time	执行函数所花费的时间（以毫秒为单位）
Avg	执行函数所花费的平均时间（以毫秒为单位）——Calls栏Time栏
Min	执行函数所花费的最短时间（以毫秒为单位）
Max	执行函数所花费的最长时间（以毫秒为单位）
File	包含所调用函数的静态文件

除了表2-1中列出的信息之外，JavaScript性能分析工具中还包含了函数调用的总次数以及JavaScript花费的总时间，如图2-4所示。

在网页上运行JavaScript性能分析工具

让我们来分析一个简单的网页。打开Firebug控制台，加载Google主页，www.google.com。当页面完全加载后，单击Console、Profile，然后输入一个单词（我输入的是php）来调用Ajax。再次单击Profile，停止性能分析并显示结果。你应该会看到类似图2-4中所示的界面。

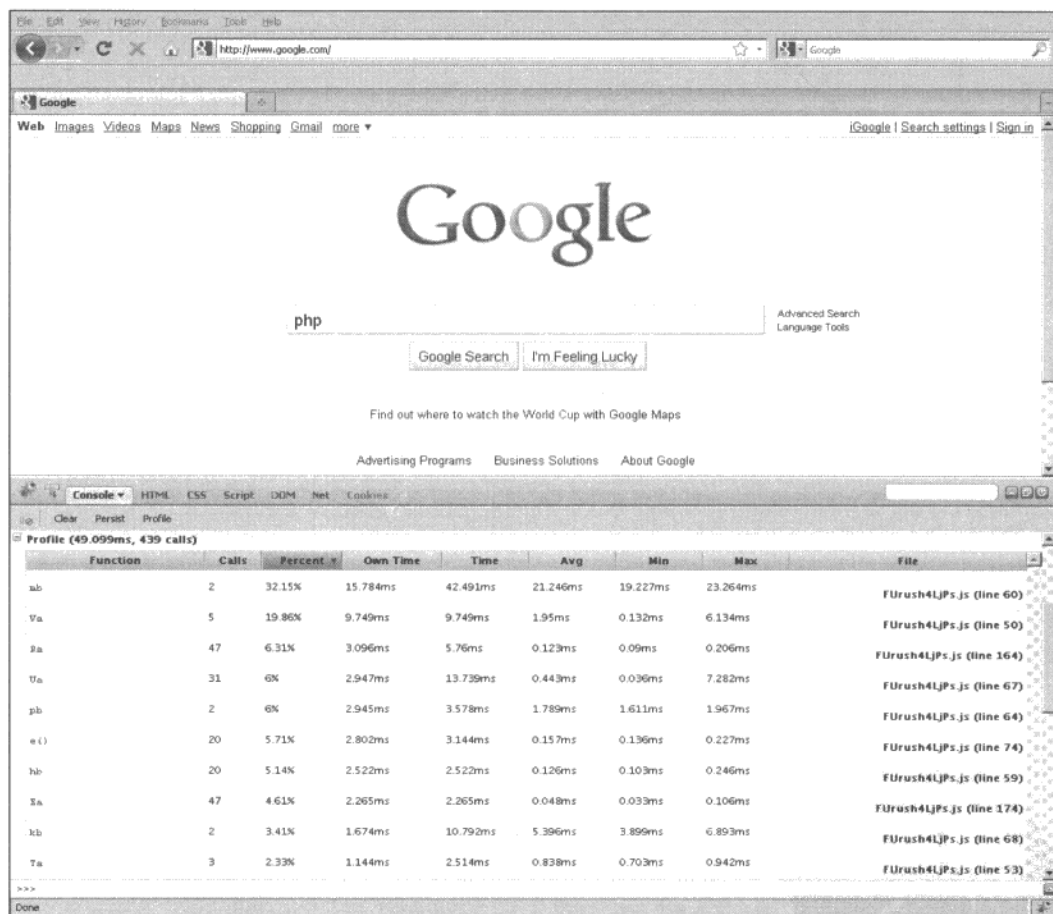


图2-4 Firebug检索在Google.com上运行JavaScript性能分析工具的结果

根据性能分析工具结果的表格，我们从图2-4中可以一眼看到，对JavaScript函数的调用有439次，而执行函数集合的总时间为49.09毫秒。我们还可以看到，需要花费较长时间执行的函数是mb，这个函数所花费的时间占了总时间的32%。既然本书不会详细讨论应如何优化JavaScript代码，我们将停在这里，你需要知道的是，你现在可以走到JavaScript开发人员面前，拿出一些有用的指标，以找出潜在的瓶颈并准确定位应该从哪里开始优化JavaScript代码。

2.2.4 Net 选项卡

Net选项卡显示了在Web服务器返回响应后浏览器发起的网络调用的详细信息。Net选项卡将结果显示在一个表中，这个表包括的项列在表2-2中，此外还包含下列内容：

- ☐ 请求总数；
- ☐ 响应数据总大小；
- ☐ 接收和呈现响应所花费的总时长；
- ☐ 响应头（Response Header）信息。

表2-2 Firebug的Net选项卡的栏描述

栏 名 称	描 述
URL	所获取的资源（内容）以及所使用的HTTP请求
Status	HTTP响应代码
Domain	从中获取资源的域的域名
Size	资源的总大小
Timeline	彩色的进度条，它表示浏览器何时请求了资源

在网页上使用Net选项卡，并再次加载Google主页，可以得到如图2-5所示的结果。

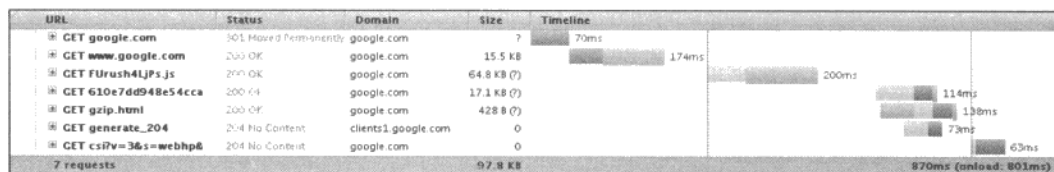


图2-5 在Google.com网页上使用Net选项卡的结果

该表含有浏览器对呈现页面所需资源的调用的逐项列表。你可以将鼠标悬停在Timeline栏中的每个资源上，查看浏览器在获取指定资源时所执行操作的详细分析，以及每种颜色所代表的含义。表2-3列出了每种颜色所代表的含义。

表2-3 在Net结果中颜色的含义

颜 色	描 述
紫色	浏览器正在等待资源
灰色	正在接收资源
红色	正在发送数据
绿色	正在连接
蓝色	DNS查找

按顺序阅读结果,首先是一个重定向。我在URL中输入了google.com而不是www.google.com,之后被重定向到www.google.com。完成该重定向用了70毫秒,而在重定向时,浏览器处于等待响应的状态并且未接收任何数据。这是图2-5中所示的第一行。70毫秒过后,浏览器被重定向,但需要再等待64毫秒才能接收数据。请记住,在浏览器等待数据时,用户也一样在等待着浏览器窗口能够呈现一些内容。在322毫秒的标记处,浏览器会拦截所有操作,同时执行JavaScript,72毫秒后浏览器开始接收内容,如图2-5所示的第三行。让我们跳过其他资源,直接看最后一项;根据结果,我们可以确定,最后的资源是在807毫秒后获取的。

通过Net选项卡提供的信息,我们不但可以开始减少所调用资源的次数,还可以减少响应数据的大小等。性能优化工程师已制定了一些规则,我们可以将这些规则用在网页上,以加快用户的网页呈现过程。在下一节中,我们将使用一个运用这些规则的工具,来帮助我们测评网页对这些规则的遵守程度。

2.3 YSlow

YSlow是一个网页性能分析工具,它是由Yahoo性能优化团队创建的。该工具是Firebug的附加件,且必须在Firefox中使用。与Firebug不同,YSlow采用了一系列规则来为网页评分。目前有两种版本的规则:默认的YSlow v2规则,基于全部22条规则为网页评分;以及Classic v1规则,根据23条规则中的13条为网页评分。

2.3.1 YSlow v2 规则集

YSlow v2中的22条网站优化规则包含以下内容:

- ☐ CSS优化;
- ☐ 图像优化;
- ☐ JavaScript优化;
- ☐ 服务器优化。

YSlow运用22条规则,针对每条规则计算出一个分数(用字母表示),并算出整个网页优化水平的整体评分。字母“A”表示网页中的大部分内容均遵循这22条规则并已对性能进行了优化。而字母“F”则表示该网页没有进行优化。

除了这些规则,YSlow还提供了可用来优化CSS、JavaScript和HTML的在线工具参考。在本章的后面,我们将会用到其中的一些工具。

在此插件中,规则集至关重要,因此现在让我们来了解其中的一些规则。

1. CSS优化规则

在开始进行CSS优化时,有一些所用规则的要点:

- ☐ 将CSS样式放置在HTML文档顶部;
- ☐ 避免某种CSS表达式;
- ☐ 精简CSS文件。

遵循这三条规则，可以删除所有不必要的空白处（精简）来减少CSS文件的大小，也可以将CSS文件放置在HTML文档顶部以加速浏览器的呈现过程。

2. 图像优化规则

出于设计考虑，如今的所有网站都会使用图像，而遵循YSlow的一些评分规则来优化这些图像可以减少网页的加载时间。

- 使用所要求的图像大小，而不是在HTML中使用宽度和高度属性重新调整图像尺寸。
- 在可能的情况下创建子画面（sprite）。

遵循第一条规则，我们可以通过使用与网页匹配的图像尺寸来减少响应数据大小。第二条规则将一个页面上的图片整合，在某些情况下成为单一文件，从而可以减少浏览器需要获取的图片总数。

3. JavaScript优化

正如本章开头所述，我们可以优化JavaScript。下面是YSlow用来检查JavaScript的三条规则。

- 将JavaScript放置在HTML底部。
- 精简JavaScript。
- 将JavaScript做成外部文件。

同样，我们希望优化响应数据的大小以及浏览器为用户呈现页面内容的方式。将JavaScript放置在HTML文件底部，浏览器将不会因加载和执行JavaScript而延迟呈现HTML。从之前在Google主页上使用Firebug - Net选项卡这个例子中就可以看出这一点。

像精简CSS一样，要精简JavaScript，可以删除空白的空间，从而减少JavaScript文件和响应数据的大小。

4. 服务器优化

在服务器优化中，YSlow会检查服务器响应头中的几个元素，如以下几个方面：

- 服务器是否采用了Gzip/bzip2压缩；
- DNS查找是否有所减少；
- 是否实现了Etag。

该规则只是YSlow所用22条规则的示例。要获取规则及其详细说的完整列表，请安装YSlow并开始为网页评分。

2.3.2 安装 YSlow

YSlow仅适用于Windows和Unix系统上的Firefox。在撰写本书时，YSlow 1.5.4版本已稳定，并可在Mozilla网站上获取，网址是<https://addons.mozilla.org/en-US/firefox/addon/5369/>。

登录该页面后，单击“Add to Firefox”（添加到Firefox）按钮；插件安装完毕后，重新启动浏览器。就是这么简单。

2.3.3 启动 YSlow

启动YSlow有两种方法。可以单击浏览器右下角的YSlow徽标，如图2-6所示；也可以打开Firebug控制台，然后单击YSlow选项卡，如图2-7所示。现在，打开YSlow控制台，让我们为Google主页评分。

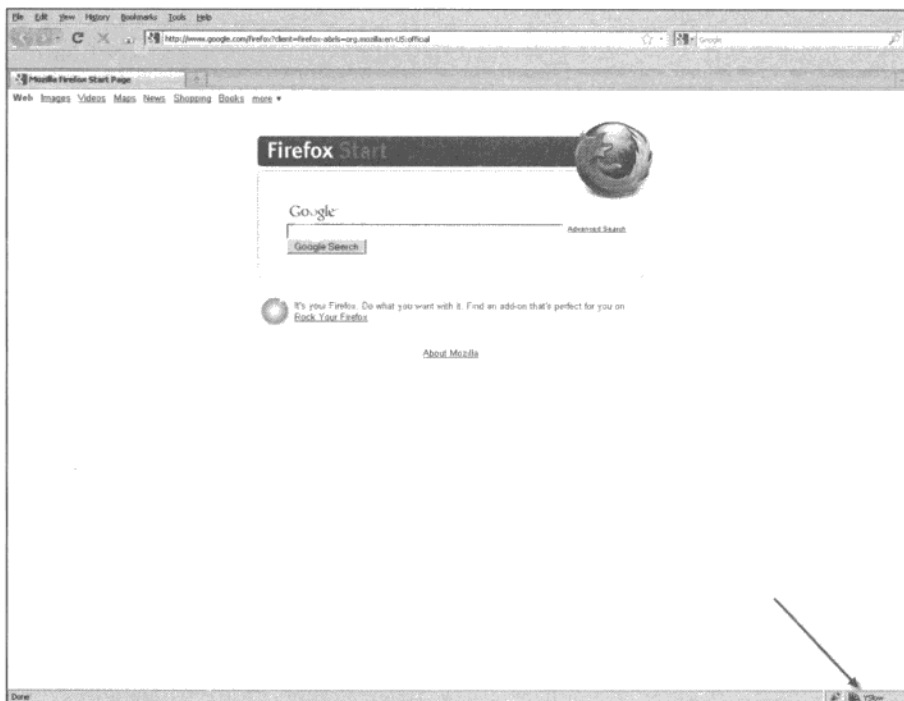


图2-6 使用YSlow图标启动YSlow

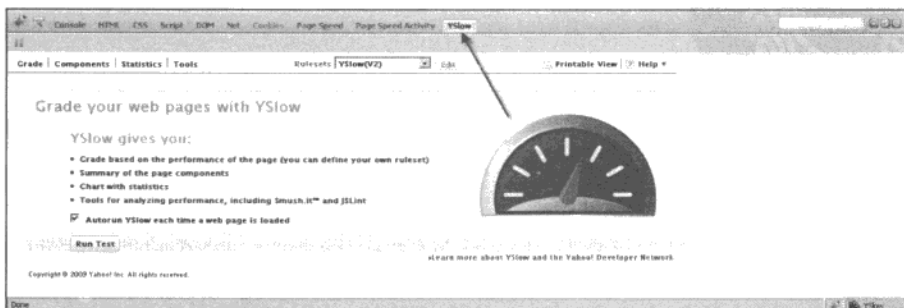


图2-7 从Firebug控制台启动YSlow

1. Grade选项卡

我们将继续使用Google主页。加载Google主页：www.google.com。Google主页加载完成后，单击YSlow工具中的“Run Test”（运行测试）；如果未选中Grade选项卡，请单击该选项卡。你将看到类似于图2-8所示的屏幕。

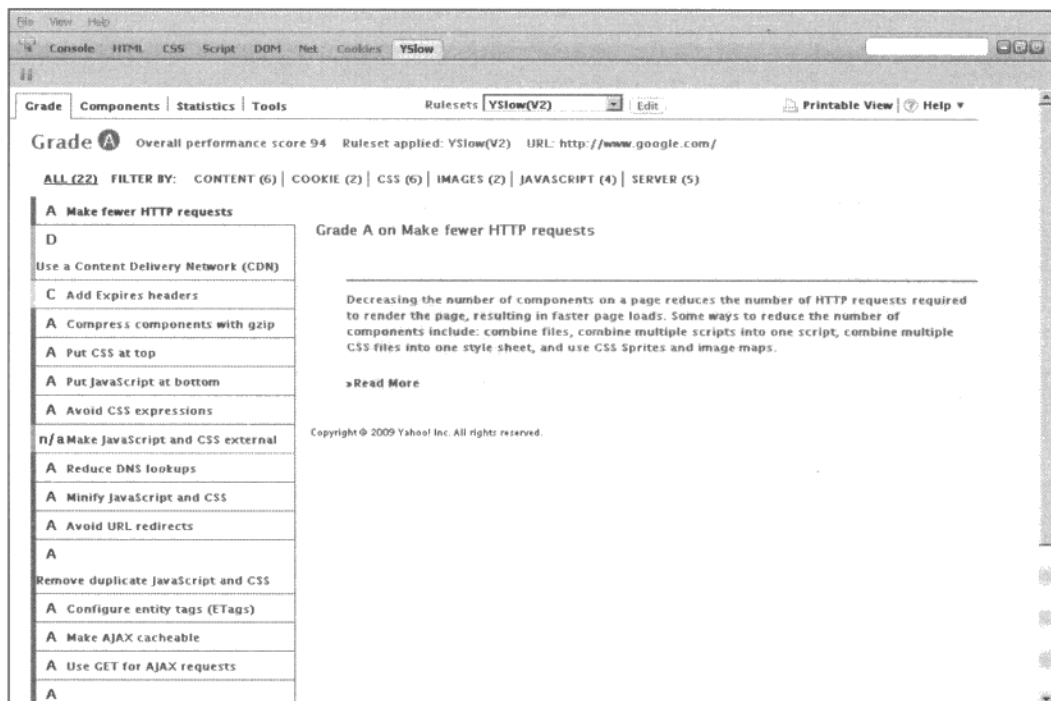


图2-8 Google.com的YSlow结果

根据YSlow工具的结果，Google主页的总体评分为“A”，性能得分为94/100。

YSlow包含6个结果筛选器，分别是Content、Cookie、CSS、Images、JavaScript和Server。筛选器选项卡显示在页面评分的下方；单击一个选项卡，则对应的筛选器将会显示规则，描述将要进行的优化的类型及其优化的作用，还会显示对应筛选器对该页面的总体评分。

回到图2-8所示的结果，Google在实现内容传送网络方面得分为“D”，在包含“Expires”头部设置方面得分为“C”，“Expires”允许网页在浏览器缓存中存储图像和内容以备日后使用。

2. Statistics选项卡

Statistics选项卡与Grade选项卡类似，更深入地展示出如何提高网页呈现性能。Statistics选项卡包含关于缓存使用的信息，并为我们提供了一个非常简单的方法来了解缓存在优化呈现功能中如何发挥重要作用。图2-9展示了Google.com的统计结果。

请注意这两张图表。一张图表描述了所请求资源的总数以及缓存为空时（用户初次访问网页时）所请求的总大小。右边的第二张图表显示的是缓存已满时（原有缓存）的信息。

对比的结果是缓存减少了两个资源请求，而资源大小则减少了65.4KB。有了原有缓存，浏览器可以减少从Web服务器上下载的网页内容，而获取存储在本地的网页内容，从而可提高网页加载的速度。我们将在本书后面的第4章和第5章进一步讨论缓存。

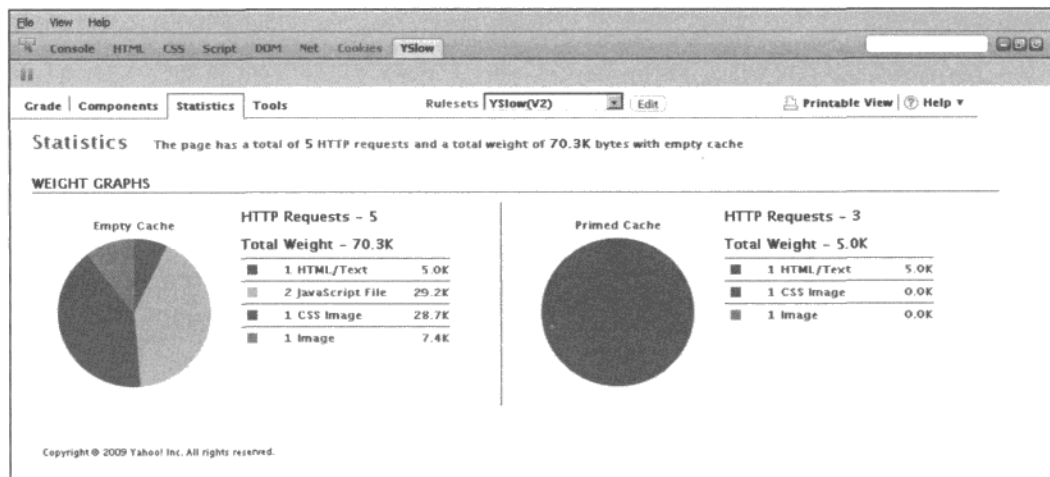


图2-9 Google统计结果图表

3. Tools选项卡

下一个选项卡是Tools选项卡。此选项卡含有一个工具列表，可以帮助我们优化在网页上使用的JavaScript、CSS和图片。我们将在本章后面继续讨论这个选项卡，现在，让我们暂时将其搁在一边。

YSlow可以对我们在某个网页上应用优化技术的水平进行评分，并且让我们更深入地理解优化，而我主要用它来对特定性能调整进行评分并阅读相关信息来了解这些调整的工作情况。有些信息YSlow无法提供，比如我可以从哪里节约资源大小，我可以更新哪些文件。它并没有那么透明，它的工作方式类似于黑盒工具。下面，让我们讨论一下与其类似的另一个工具Google的Page Speed，它的工作方式类似于白盒工具。

2.4 Page Speed

Google的Page Speed也是用于Firefox/Firebug的插件，与YSlow一样，它也为网页运用优化技术的水平进行评分。与YSlow不同的是，Page Speed提供的信息包括它对哪些文件进行了分析、如何优化文件、在网页的什么地方进行优化以及衡量优化可以提高多少性能的指标。

除未使用的CSS), Page Speed能识别出哪些CSS元素未被使用,并预计文件中3.3KB的内容可以被删除,这就将文件大小减少了46.5%。再次强调,减少响应数据大小和获取的资源数量可以提高用户方的网页呈现性能。

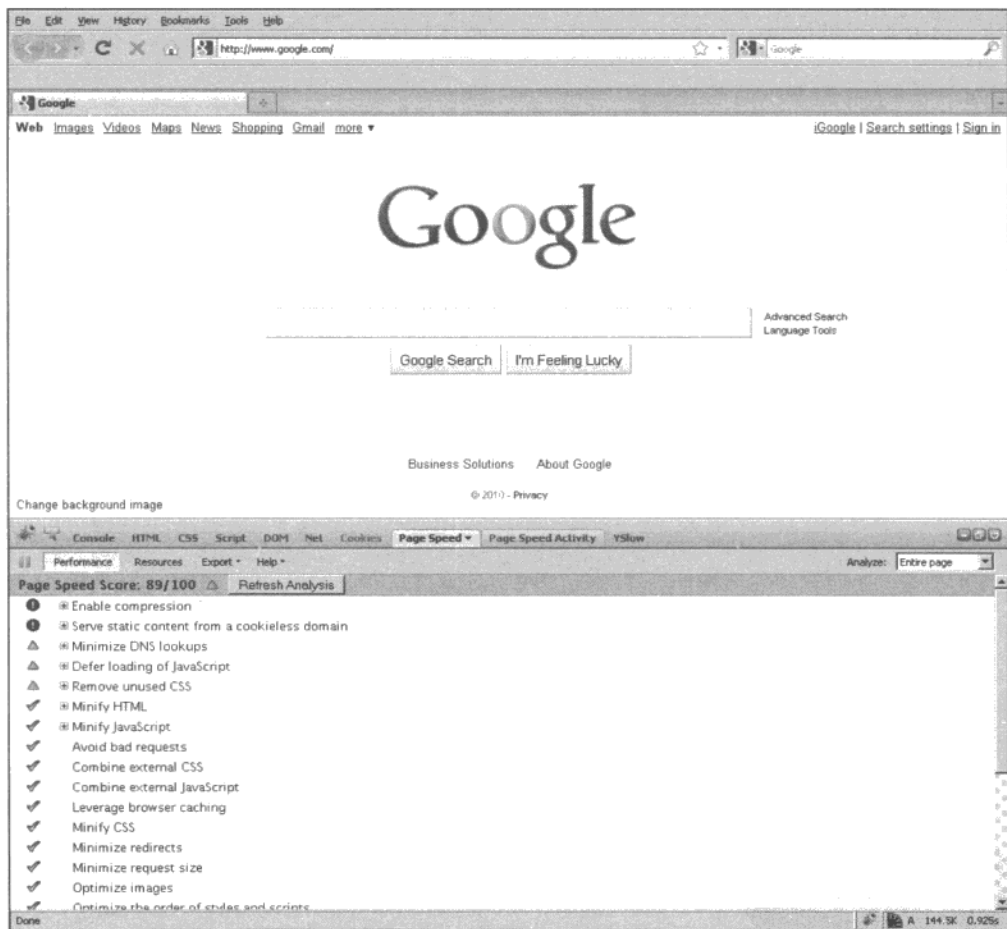


图2-11 Google.com的Page Speed结果

2.5 优化工具

除了YSlow中的Tools选项卡,上面提及的工具均未提供优化页面上JS、CSS或图像的途径。这些工具仅是衡量手段,用于评估我们的资源以及对这些资源的优化效果(我们将在后面的章节学习到)。

2.5.1 JavaScript 优化

每个开发人员都需要首先理解的是,终端用户并不会只使用一种浏览器。用户可以使用并且会使用如今市面上可用的多种浏览器:Internet Explorer (IE) 6/7/8、Firefox、Chrome、Opera和Safari,这只是其中几个。

每个浏览器都用自己专有的JavaScript引擎来处理JavaScript。到目前为止,Chrome 6浏览器已实现了V8引擎,这是一个C++引擎,可直接安装在使用了ECMAScript的Chrome浏览器上,这一点已经在ECMA 262第三版中做了说明,请参考V8网站<http://code.google.com/p/v8/>了解相关的其他信息。Safari 4使用的是字节码引擎SquirrelFish (<http://webkit.org/blog/189/announcing-squirrelfish/>),Firefox 3.5+使用了TraceMonkey (<https://wiki.mozilla.org/JavaScript:TraceMonkey>),而Opera则使用Carakan。

由于每个浏览器都在使用自己的JavaScript引擎,因此可以确定的是应用程序中的JavaScript在不同的浏览器中会有不同的性能。使用某种浏览器的用户也许比使用其他浏览器的用户体验到更快的加载速度。幸好,我们有可以运行的基准,能够测试出哪个浏览器的JavaScript引擎更快。

使用SunSpider JavaScript基准测试 (<http://www2.webkit.org/perf/sunspider-0.9/sunspider.html>),可以比较各个主流浏览器运行JavaScript函数集的性能。该测试不能运行任何特殊的API,也不能与DOM进行交互。它只能测试JavaScript的核心功能,举几个例子,比如加密、日期、数学、字符串和正则表达式功能。图2-12给出了在5种浏览器(IE 8、Firefox 3.6、Chrome 7、Safari 5和Opera 10)上运行该测试的结果。

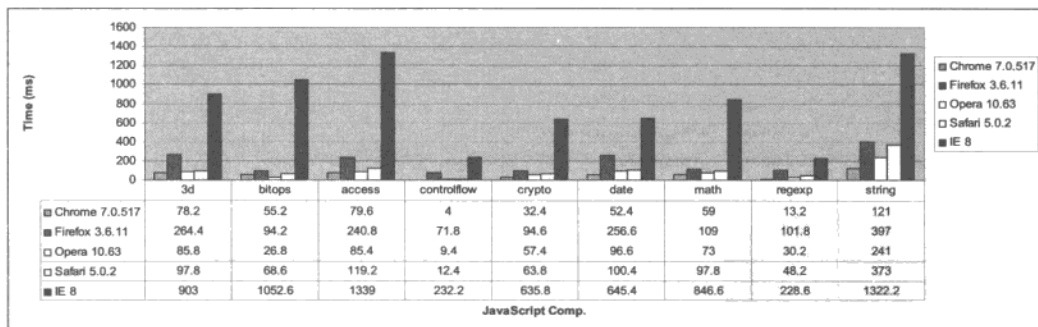


图2-12 SunSpider基准测试结果

图2-12中所示的图表展示了SunSpider基准测试对5种主流浏览器进行测试的结果。这5种浏览器运行在奔腾4、3.20Ghz系统上,操作系统为Windows XP,测试结果表明,Chrome的JavaScript引擎速度最快,也就是说JavaScript在这个浏览器中的执行速度最快。

2.5.2 JavaScript 的放置位置

如前所述,JavaScript所放的位置对应用程序在浏览器中的加载至关重要。JavaScript的放置

位置之所以如此重要,是因为浏览器在遇到<script>标记时的反应。每当浏览器遇到一个<script>标记时,它将会中断该网页上所有进一步的呈现,因此用户会看到并只能看到等待中的页面内容(参见代码清单2-1)。如果将所有JavaScript移到HTML文件底部,则HTML将可以呈现得更快(参见代码清单2-2)。

代码清单2-1 JavaScript文件中的不佳放置位置

```
<html>
<head>
<title>JavaScript Example</title>

<script type="text/javascript">
function addItem()
{
    var items = ['Apache', 'Nginx', 'Lighty'],
        localDoc = document; //local document object
    var node = '',
        i = 0,
        li = '';

    node = localDoc.getElementById('webservers');
    for(i; i<items.length; i++)
    {
        li = document.createElement('li');
        li.innerHTML = items[i];
        node.appendChild(li);
    }
}
</script>

</head>
<body>
<div id="main">

    <h3>Technology List</h3>
    <ul id="techlist">
        <li>JavaScript</li>
        <li>CSS</li>
        <li>Images</li>
        <li>PHP</li>
    </ul>

    <ul id="webservers"></ul>

</div>

<script type="text/javascript">
```



```

addItem();
</script>
</body>
</html>

```

代码清单2-2 优化过的JavaScript放置位置

```

<html>
<head>
<title>JavaScript Example</title>
</head>
<body>
<div id="main">

    <h3>Technology List</h3>
    <ul id="techlist">
        <li>Javascript</li>
        <li>CSS</li>
        <li>Images</li>
        <li>PHP</li>
    </ul>

    <ul id="webservers"></ul>

</div>

<script type="text/javascript">
function addItem()
{

    var items = ['Apache', 'Nginx', 'Lighty'],
        localDoc = document; //local document object
    var node = '',
        i = 0,
        li = '';

    node = localDoc.getElementById('webservers');
    for(i; i<items.length; i++)
    {
        li = document.createElement('li');
        li.innerHTML = items[i];
        node.appendChild(li);
    }

}

addItem();
</script>
</body>
</html>

```

代码清单2-1和代码清单2-2中所显示的HTML和JavaScript代码含有使用HTML标记ul和li的一系列Web技术。其中的HTML代码还包含一个空的div标记，将由JavaScript对其进行更新。JavaScript代码中包含一个单独的函数，它负责创建DOM元素、将每个Web服务器放于li标记中并将li附加到空的Web服务器的ul标记后面。

使用Firebug测试代码清单2-1和代码清单2-2，结果如图2-13和图2-14所示。

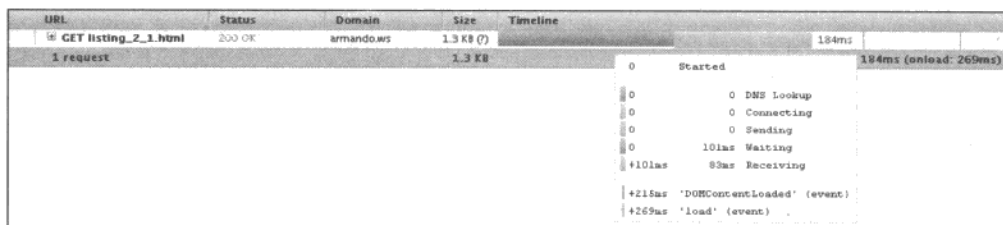


图2-13 代码清单2-1的Firebug Net测试结果

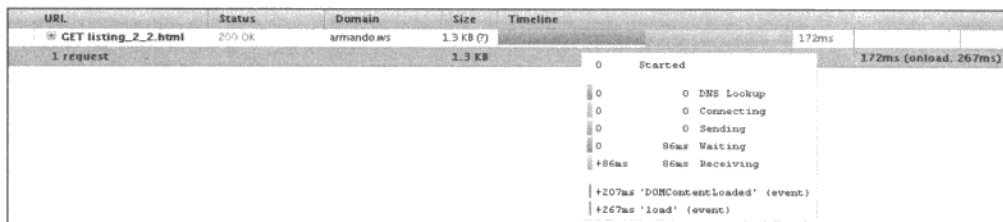


图2-14 代码清单2-2的Firebug Net测试结果

图2-13和图2-14均没有使用缓存，而且模拟了用户对页面的初次访问请求。在图2-13中，页面的呈现时间为269毫秒，而代码清单2-1中所示代码呈现页面的时间为267毫秒。此外，使用代码清单2-2，浏览器拦截页面呈现的时间也从101毫秒减少到了86毫秒。其中所用的JavaScript和HTML代码均为最小量，这表示该页面已经简化过。在较大的HTML/JavaScript页面中，这种改进会达到更好的效果。

2.5.3 精简 JavaScript

减少响应数据大小可减少浏览器等待加载页面内容有效负载所需的时间，从而直接提高性能。内容的有效负载包含浏览器呈现页面所需的一切资源。有效负载越小，页面下载越快。精简操作可通过删除JavaScript文件中的空白空间、移除无用代码、去除注释、缩小变量、删除不必要的代码（某些压缩工具可实现）来减少JavaScript文件的大小。

我们可以通过代码清单2-3中所示的JavaScript来演示精简的过程。代码清单2-3中所示的JavaScript代码对代码清单2-2中所示的代码进行了扩展，添加了注释并将for循环中实现的功能移到了新增函数中。JavaScript代码也从HTML中移除，并放置于单独的文件listing2_8.js中。

代码清单2-3 使用JavaScript为无序列表添加新条目

```

/**
 * Add Items within the items array to a ul
 * html tag
 */
function addItems()
{
    var items = ['Apache', 'Nginx', 'Lighty'],
        node = '',
        i = 0,
        li = '';

    node = document.getElementById('webservers');
    appendList(items, node);

}

/**
 * Foreach item within the array, create a li node
 * and append to the node provided.
 */
function appendList(items, node)
{
    for(i; i<items.length; i++)
    {
        li = document.createElement('li');
        li.innerHTML = items[i];
        node.appendChild(li);
    }
}

```

使用可用的精简工具（如YUI Compressor等），将文件大小从525字节减少到278字节。代码清单2-4显示了精简后的最终文件。

代码清单2-4 YUI Compressor的输出文件

```

function addItems(){var items=["Apache","Nginx","Lighty"],node="",i=0,li="";
node=document.getElementById("webservers");appendList(items,node);}function
appendList(items,node){for(i;i<items.length;i++){li=document.createElement("li");
li.innerHTML=items[i];node.appendChild(li);}}

```

当然，得到的精简文件的效果并不是十分显著。从573字节降至255字节不会被认为解决了瓶颈问题的根源，来看看现实世界中的JavaScript文件，比如JavaScript框架jQuery。jQuery的主页提供了两个版本的框架，文件大小为155KB的jQuery未精简版本以及仅24KB的精简版本，后者比前者少了131KB，也就是降低了84.51%的大小。

2.6 精简工具

这里，我们将介绍两个JavaScript精简工具，Yahoo的YUI Compressor以及Google的Closure Compiler。我们将分别使用这两种工具来压缩jQueryJavaScript框架。

2.7 YUI Compressor

Yahoo的YUI Compressor可以在YSlow的Firebug附加件中使用，也可以下载该工具的Java (.jar) 版本并进行操作。在撰写本书时，YUI Compressor的最新稳定版本是2.4.2，并可用于Linux和Windows系统，此工具可以从YUI Compressor的官方网站 (<http://developer.yahoo.com/yui/compressor/>) 下载。如果不想下载，也可以打开Firebug，单击YSlow，单击Tools，然后单击“All JS Minified (所有精简的JS)”。这种方式唯一的缺点是，需要将结果复制并粘贴到自己的文件中。

完成该工具的下载后，将其解压缩。jar文件将存放在build目录中：build/yuicompressor-2.4.2.jar。要精简某个JavaScript文件，可以使用代码清单2-5所示的命令。

代码清单2-5 用YUI Compressor进行精简

```
java -jar yuicompressor-2.4.2.jar -o js-mini.js js-to-minify.js
```

代码清单2-6中所示的命令将执行YUI Compressor的jar文件，它使用了输出标志-o。输出标志将指定存放精简文件的位置。代码清单2-6所示命令中的第二个文件是存放要精简的文件的位置。在这个例子中，要精简的文件存放在当前的工作目录中，其名称是js-to-minify.js。要查看可用选项的完整列表，请运行命令java -jar yuicompressor-2.4.2.jar -h。

用这个工具精简JQuery网站上未精简的JQuery文件。未精简的开发文件大小是155KB，运行以下命令：

代码清单2-6 用YUI Compressor精简JQuery

```
java -jar yuicompressor-2.4.2.jar -o jquery-min.js jquery.js
```

该命令将生成一个新文件jquery-mini.js，其大小是78KB。

2.8 Closure Compiler

另一种可用的精简工具是Google的Closure Compiler工具。这个工具以JavaScript文件作为输入，如其官方网站 (<http://code.google.com/closure/compiler>) 所述，“编译于JavaScript而胜于JavaScript”。Closure Compiler有以下三种使用方式：

- 使用可以下载到本地硬盘驱动器上的Java jar文件；
- 通过官方网站上的在线工具进行操作；
- 使用RESTful API。

要获取其他信息，可以访问官方网站。

为了运行下面的例子，需要下载包含此工具的压缩文件。请访问其官方网站进行下载，单击右侧“How do I start?”（如何开始？）框下方的“Download the application”（下载应用程序）链接，然后将下载的内容解压缩。解压后的文件要确保存在.jar文件，因为我们将用它来压缩JQuery文件。

让我们使用Closure Compiler来精简未修改的jQuery文件。运行代码清单2-7所示的命令，将生成大小为71KB的jquery-min-cc.js。

代码清单2-7 使用Closure Compiler精简jQuery

```
java -jar compiler.jar --js jquery.js --js_output_file jquery-min-cc.js
```

2.8.1 减少资源请求

通过减少向Web服务器请求的数量可以减少浏览器必须通过Internet获取资源的次数。在处理JavaScript时，可以将网页所需的所有JavaScript文件合并成一个文件，从而轻松实现减少资源请求的目的。这里需要注意的是，要确保这些文件以正确的顺序合并在一起，即某个文件所需的功能应该出现在使用该功能的文件之前。

2.8.2 使用服务器端压缩

文件压缩是另一种减少JavaScript以及CSS文件大小的方法。对已精简过的JavaScript或CSS文件进行压缩可进一步减小它。如今使用得最广泛的压缩工具是GZip。我们可以对比368.6KB文件的两个版本：未压缩但已精简的版本和已精简的+gzip的版本，比较结果如表2-4所示。

表2-4 使用压缩功能的对比结果

压 缩	大 小
无	368.6KB
已精简	88.2KB
精简+gzip	30.05KB

结果表明，该文件精简+gzip版本的大小要远远小于其他两个文件。在第8章中，我们将进一步讨论相关内容。

2.9 图像压缩

并非所有的图像压缩都是一样的，也并非所有的图像压缩都能满足网页的特定需求。应用图像时采用的压缩类型至关重要，它会影响浏览器在开始加载页面前需要下载的信息量。使用正确的压缩方式可以减少响应数据大小。

我们在第1章中曾简要地描述了图像压缩，对一个小图像和一个大图像分别运用ab基准测试来比较响应数据的大小及响应时间，结果表明，较大的图像会因其响应数据大而增加响应时间。对于不同的文件类型，如JPEG、GIF和PNG，有一个通用的规则，即使用GIF做页面上的徽标和小图标，用JPEG呈现照片或高品质图像，其他的一切图像均使用PNG格式。如果你已经在运用这条规则，那么你可以再进一步，使用Web压缩工具（比如Yahoo! 的Smush.it）来压缩图像。

2.10 Smush.it

Smush.it是Yahoo! 开发的无损压缩工具, 可以进一步减少图像文件的大小。可通过www.smushit.com访问该工具, 你可以上传文件, 也可以使用一个URL列表来引用它们(如图2-15所示)。文件大小的限制为1MB。



图2-15 Smush.it主页

我们将使用300×300的小图像来演示Smush.it的用法, 如图2-16所示。



图2-16 在网页上使用的300×300原始图像

原始文件是一个100KB的JPEG。在对该图像使用Smush.it后, 生成的JPEG大小降至98.46KB, 减少了1.96%, 如图2-17所示。

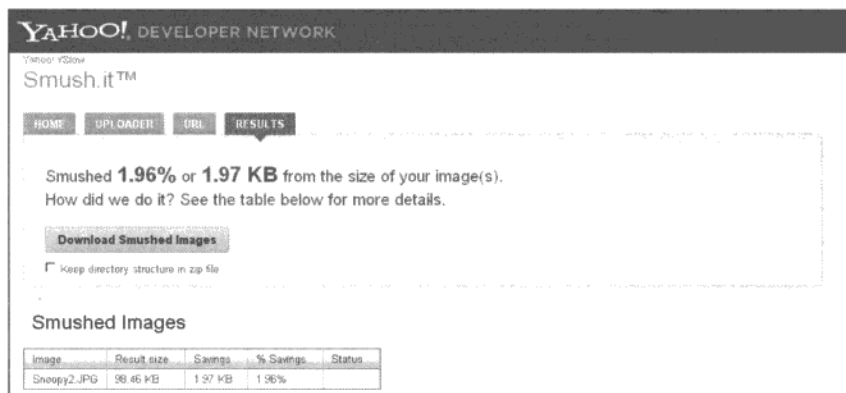


图2-17 使用Smush.it压缩原始图像

这个例子中仅使用了一个图像，所以压缩前后的变化不大。但如果是显示15个图像的网站会怎样？如果对每个这样的图像都做出少量压缩，能够节省多少空间？我们可以使用Firebug和YSlow来测试这种情况。使用CNN主页进行测试，打开Firebug，单击YSlow，然后单击Tools选项卡，最后单击“All Smush.it”链接，这样可以从YSlow中运行Smush.it。运行结果如图2-18所示。

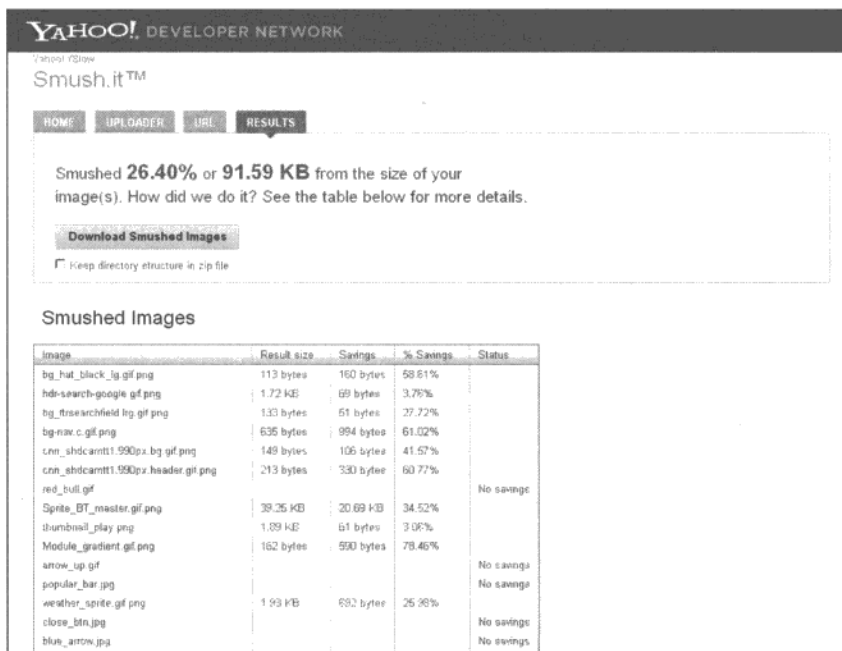


图2-18 对CNN主页运行Smush.it的结果

结果证明了千字节(KB)级别的减少量,在这个例子中,CNN主页确实因为使用Smush.it而获益。主页上完整的图片集共减少了26.4%的大小,也就是从响应数据的大小减少了91.59KB。

现在,我们有了可以衡量前端网页优化效果的准则和工具以及实现优化的各种工具,可以在下一章中讨论向下一层的内容了,通过使用PHP性能分析工具并应用最新的最佳实践来关注一下驱动网页用户视图的PHP代码。

2.11 小结

本章介绍了三个性能测试工具:Firebug、YSlow和Page Speed。我们学习了如何在Firebug中以及如何通过Firefox中的快速启动图标打开这三种工具。我们通过一个小的JavaScript示例以及Google主页上使用的JavaScript代码学到了如何开始使用和阅读概要分析工具的结果。通过Firebug的Net选项卡,你也学会了如何读取输出数据并了解了浏览器遇到<script>标记时会做出的行为。此外,你还知道了如何使用YSlow的22条评分标准以及Page Speed的标准为网页性能评分。

你还学到了如何运用测量网页执行性能的工具来实现JavaScript和图像的性能调整。性能调整包括使用YUI Compressor和Google Closure Compiler等工具精简JavaScript。你还了解了将多个JavaScript文件合并成一个文件以减少资源请求的好处,比较了使用Gzip进行的服务器端压缩,并利用图像压缩工具Smush.it压缩示例图像和CNN主页上的图像,从而了解了使用这种工具可以减少的响应数据大小。

看完这本书后，你将会学到一些非常好的优化技巧，比如Opcode缓存、数据库优化，以及使用了Memcached的数据缓存。但在介绍这些内容之前，我们需要学习如何应用最佳的编码实践。知道应该何时使用某个特定PHP函数而不是另一个，并且能够对代码进行少量逐渐地改进从而提高PHP代码的性能。

在前面的章节中，我们着重介绍了如何通过使用Firebug、Page Speed和Yslow来优化内容呈现并减少用户所需下载的数据量。在许多情况下，对返回给用户的响应和内容进行优化就足以让大部分用户满意。但在更多情况下，你会希望更进一步去优化PHP代码本身。

虽然显得有些老套，但我要引用一句中国古代的谚语：“授人以鱼，只救一时之急；授人以渔，则可解一生之需。”我之所以提起这句话，乃是因为这就是本章意图。的确，我们会研究并应用最新的最佳实践，但我会更进一步，教你如何成为一个PHP优化领域的“渔夫”！你将学到如何安装Vulcan Logic Dumper (VLD) 并分析其提供的结果，从而判断特定的函数调用之所以会比其他调用更快的原因。另外，你将了解如何安装strace并在Apache的Web服务器调用某个PHP脚本时，用其查看该脚本调用的系统级操作。最后，你还将学会通过Xdebug提供的结果找到代码的瓶颈。Xdebug是一个概要分析工具，你将在本章中学习如何安装和使用它。

3.1 PHP 最佳实践

最近，我发现许多优秀的应用软件都未能应用一些基本的最佳实践（这些实践正是你将在本章中学到的），从而无法在速度上有所突破。其中的一些技巧已被人们使用和测试过了，它们已经经过了多年的实践检验。所以，人们不使用它们着实让我吃了一惊。

如果要深入了解某个最佳实践存在了多长时间，可以看下面的例子：Rasmus Lerdorf（对，正是PHP的创造者）在其关于PHP优化的演讲中详尽的陈述了使用PHP函数`require`而不是`require_once`，他在2004年就已首次提出了这一实践。然而直到今天，也很少有应用程序遵循这个最佳实践。但是你很幸运，因为我们将会讲到它。

提示 要获取关于PHP相关性能说明的列表，请访问<http://talks.php.net/index.php/PHP>。

下面，我们将讲述应用PHP 5.3.2的最佳优化实践。PHP已经完成了许多性能调整增强，因此

旧版本的PHP也将受益于这些编码实践。

- 使用require与使用require_once的对比。
- 提前计算一个for循环的长度。
- 比较for、foreach和while循环在访问数组元素时的性能。
- 使用file_get_contents访问文件。

除此之外，还有一些其他的优化技术，比如使用逗号而不是句点来连接字符串（如代码清单3-1所示），或者在字符串包含变量时使用双引号而不是单引号（如代码清单3-2所示）。但是，这些调整对性能的提高极其有限，所以在这里，我们只是将其作为例子简单介绍一下。

代码清单3-1 使用逗号连接字符串

```
<?php
echo "Hi ". "there. ". "how are ". "you?"; //Slow
echo "Hi ", "there. ", "how are ", "you?"; //Faster...slightly
```

代码清单3-2 字符串包含变量时使用双引号

```
<?php
$name = "Snoopy Padilla";
echo 'Hi there, '.$name; //Slower
echo "Hi there, $name"; //Faster...slightly
```

再次强调，在优化PHP代码时，这些选择不应该是重点关注的对象。

回到我们的PHP性能路线图，PHP最佳实践是PHP层中的初始模块，如图3-1所示。



图3-1 PHP性能路线图

PHP层还包含其他的模块，比如变量/数据缓存和Opcode缓存，我们将在其他章节中讨论这些模块，在这一章中，我们将专注于研究可以在PHP代码中实现的小改进。

在开始之前，你需要了解PHP的“经济性”，以及调用函数的成本及其响应时间之间的权衡考虑。

3.1.1 PHP 的经济性

取决于PHP脚本的不同，可能会含有许多对内部PHP函数和自定义函数的调用，甚至还会调用对象内部的实例化和调用方法。从PHP经济的角度考虑，每一个对这些函数的调用都会产生成本，而与其他函数相比，一些函数的调用和执行更为昂贵。脚本在生产环境内准备就绪后，从根本上讲，你就已经准备好支付你的成本了，而支付的“货币”就是响应时间。

产生的成本越高（许多函数调用），响应时间就会越长。概括地说，函数调用得越多，应用程序就会越慢。

3.1.2 require 与 require_once

使用require还是require_once？这是个老问题了。在学完本节后，你可以根据其性能判断应该使用哪一个了。

require和require_once都是PHP函数，开发人员可以使用它们在某个特定的脚本中导入外部PHP文件以供使用。你可以根据应用程序的复杂程度调用一次或若干次require_once/require。使用require（而不是require_once）可以提高应用程序的性能。

由于在导入PHP脚本时将进行大量的操作状态(stat)调用，因此require要快于require_once。如果你请求的文件位于目录/var/shared/htdocs/myapp/models/MyModels/ClassA.php下，则操作系统会在到达ClassA.php之前的每个目录中进行一次stat调用。在这个例子中，共进行6次stat调用。require也会发起stat调用，但是次数较少。越少的函数调用，代码的运行速度就越快。

我们将使用代码清单3-3中所示的代码比较这两种方法。代码清单3-3使用了4个文件，均通过PHP函数require_once请求并导入。

代码清单3-3 加载四个外部类文件的基本代码

```
<?php
require_once("ClassA.php");
require_once("ClassB.php");
require_once("ClassC.php");
require_once("ClassD.php");

echo "Only testing require_once.";
```

所请求的类均显示在代码清单3-4中，它们仅是声明了自己，并不包含其他功能。

代码清单3-4 类A到类D

```

<?php
class A
{
}

class B
{
}

class C
{
}

class D
{
}

```

以上4个空类均可帮助我们模拟一个需要在主脚本中使用外部PHP文件的PHP脚本。我们排除了任何额外的函数调用，专注于使用require_once()的文件加载。

将每个类放于单独的文件中：ClassA.php、ClassB.php、ClassC.php、ClassD.php，并将这些文件与代码清单3-3所示的代码保存在一起。重新启动Web服务器，然后使用第1章介绍的ab工具运行如代码清单3-5所示的ab命令。

代码清单3-5 用于测试require_once和require函数的ab命令

```
ab -c 10 -n 100000 localhost/index.php
```

代码清单3-5中的ab命令模拟了10万个请求，同一时间有5个并发请求。命令终止后，应该可以看到类似于图3-2所示的结果。

```

Concurrency Level:      10
Time taken for tests:    9.938 seconds
Complete requests:      10000
Failed requests:         0
Write errors:           0
Total transferred:      212000 bytes
HTML transferred:       29000 bytes
Requests per second:    1000.63 [#/sec] (mean)
Time per request:       9.9375 [ms] (mean)
Time per request:       9.938 [ms] (mean, across all concurrent requests)
Transfer rate:          21.32 [Kbytes/sec] received

Connection Times (ms)
  min   mean    +-----+  median    max
Connect:  0      9      8.5       16       47
Processing: 16    89    20.6      94      266
Waiting:  0     60    28.1      63      266
Total:    16    98    21.5      94      266

Percentage of the requests served within a certain time (ms)
 50%:  94
 66%:  94
 75%: 109
 80%: 109
 90%: 109
 95%: 125
 98%: 172
 99%: 203
100%: 266 (longest request)

```

图3-2 ab -n 100000 localhost/index.php的结果

使用此ab命令测试require_once(), 我们可以看到, 响应时间为99毫秒。我们的结果还表明, 该脚本每秒可以支持100.63个请求。

现在, 让我们将require_once() 函数调用改为require(), 如代码清单3-6所示。重新启动Web服务器, 然后重新运行代码清单3-5中的ab命令。结果应类似于代码清单3-6。

代码清单3-6 使用require的已优化代码

```
<?php
require("ClassA.php");
require("ClassB.php");
require("ClassC.php");
require("ClassD.php");

echo "Only testing require_once.";
```

图3-3所示的新结果表明, 每秒可支持的请求数量有所提升, 从100.63增加到了105.44。这个结果还说明, 代码的响应时间从最初的99.37毫秒降到了94.84毫秒, 减少了5毫秒。

```
Concurrency Level:      10
Time taken for tests:    9.484 seconds
Complete requests:      1000
Failed requests:         0
Write errors:           0
Total transferred:      217000 bytes
HTML transferred:       29000 bytes
Requests per second:    105.44 [#/sec] (mean)
Time per request:       94.844 [ms] (mean)
Time per request:       9.484 [ms] (mean, across all concurrent requests)
Transfer rate:          22.34 [Kbytes/sec] received

Connection times (ms)
  min  mean[+/-sd] median   max
Connect:    0    8   8.1    16    31
Processing: 16   85  13.0    78   203
Waiting:    0   56  23.7    63   203
Total:      16   93  13.2    94   203

Percentage of the requests served within a certain time (ms)
 50%    94
 66%    94
 75%    94
 80%    94
 90%   109
 95%   109
 98%   109
 99%   125
100%   203 (longest request)
```

图3-3 使用require函数的ab结果

3.1.3 提前计算循环长度

在进入循环之前计算循环长度是另一项可以使用的优化技术。

代码清单3-7中所示的代码是一个简单的for循环, 它将遍历数组\$item并分10次计算出数值。要确定哪些地方可以进行优化(在这个例子中是减少函数调用次数), 你需要一步一步地分析代码都做了什么。

代码清单3-7 未优化的for循环

```
<?php
$item = array(1,2,3,4,5,6,7,8,9,10);
```

```
for($i=0; $i<count($items); $i++)
{
    $x = 10031981 * $i;
}
```

让我们看一下代码清单3-7中的代码，主要注意for循环的逻辑，PHP按以下方式执行该for循环：

- (1) 将*\$i*初始化为0，从索引0开始循环，使用count()检查数组长度，*\$i*增加1。
- (2) 迭代0完成后，开始索引1，使用count()检查数组长度，*\$i*增加1。
- (3) 迭代1完成后，开始索引2，使用count()检查数组长度，*\$i*增加1。

代码继续运行，直至到达数组元素的末尾。迭代一开始，问题便存在了。每次开始一个新索引的迭代时，都必须调用函数count()才能确定数组长度。在代码清单3-7所示的代码中，count()被调用了10次，有9次都是多余的。这些不必要的调用都可以被替换，只要在到达for循环之前调用一次count()就可以了。更新后的优化代码如代码清单3-8所示。

代码清单3-8 已优化的for循环

```
<?php
$items = array(1,2,3,4,5,6,7,8,9,10);
$count = count($items);

for($i=0; $i<$count; $i++)
{
    $x = 10031981 * $i;
}
```

代码清单3-8中的代码产生的结果与代码清单3-7的相同，但是函数调用的次数从10次降到了1次。调用的次数越少，PHP脚本的运行速度越快。

我们可以使用PHP函数microtime()来准确计算出减少9次count()函数调用可以节省多少时间。使用代码清单3-7所示的代码，并增加另一个for循环，执行该代码10万次，代表10万个用户请求该PHP脚本。对代码清单3-7中代码所做的变动在代码清单3-9中以粗体标记了出来。

代码清单3-9 未优化的for循环基准代码

```
<?php
$items = array(1,2,3,4,5,6,7,8,9,10);

$start = microtime();
for($x=0; $x<100000; $x++){
    for($i=0; $i<count($items); $i++)
    {
        $j = 100381*$i;
    }
}
echo microtime()-$start;
```

执行十次该代码并计算结果的平均值，我们得出，10万次循环的总执行时间为0.046毫秒。重新启动Web服务器，现在，让我们测试代码清单3-10所示代码，其中包含优化了的for循环。

代码清单3-10 优化的for循环基准代码

```
<?php
$items = array(1,2,3,4,5,6,7,8,9,10);
$count = count($items);

$start = microtime();
for($x=0; $x<100000; $x++){
    for($i=0; $i<$count; $i++)
    {
        $j = 100381*$items[$i];
    }
}
echo microtime()-$start;
```

我们将再次运行10次代码并获取平均值。使用此代码，我们可以看到，for循环的平均执行时间为0.0095，减少了0.036毫秒。优化过的代码快了0.036毫秒。

3

3.1.4 使用 foreach、for、while 循环访问数组元素

访问数组数据的方法是可以优化的，方法就是使用foreach循环替代while或for循环。优化数据访问的方法至关重要，因为很多Web应用程序需要从数据库或XML文件中读取数据并且必须遍历每条记录后才能将数据显示给用户。为了演示这种优化，我们将使用代码清单3-11中所示的代码。

代码清单3-11 在代码清单3-8所示代码中使用foreach

```
<?php
$items = array_fill(0, 100000, '12345678910');

$start = microtime();
reset($items);
foreach($items as $item)
{
    $x = $item;
}
echo microtime()-$start;
```

代码清单3-11创建了一个数组\$items，其中包含10万个元素，每个元素含有155个字节的字符串，代表数据库中的典型数据。之后，该代码段设置了开始时间并使用foreach循环访问数组的每个元素，最后，我们以毫秒为单位显示所用时间。我们连续执行了10次代码清单3-11中所示的代码，然后计算出每次执行时间的平均值，其结果是0.0078毫秒。

以代码清单3-11为基础，修改这段代码，使用while循环，而不是foreach循环。代码清单3-12中粗体显示的代码代表的是我们对其所做的修改。

代码清单3-12 在代码清单3-11所示代码中使用while循环

```
<?php
$items = array_fill(0, 100000, '12345678910');
```

```
$start = microtime();
reset($items);
$i=0;
while($i<100000)
{
    $x = $items[$i];
    $i++;
}
echo microtime()-$start;
```

重新启动Web服务器并运行该代码10次以后，我们再次计算平均执行时间。用while循环访问数组中单个元素的平均时间为0.0099毫秒。

最后一组循环比较的是for循环。我们使用代码清单3-13所示代码，按照同样的基准循环流程，重新启动Web服务器、执行代码10次并计算平均结果。

代码清单3-13 在代码清单3-11所示代码中使用for循环

```
<?php
$items = array_fill(0, 100000, '12345678910');

$start = microtime();
reset($items);
for($i=0; $i<100000; $i++)
{
    $j = $items[$i];
}
echo microtime()-$start;
```

上述三种循环基准的结果列于表3-1中。表中结果可以证明，使用foreach循环访问数组元素是最好的方法。

表3-1 10万个元素的数组的PHP循环平均执行时间

循环类型	平均执行时间
foreach	0.0078毫秒
while	0.0099毫秒
for	0.0105毫秒

3.1.5 文件访问

PHP有四种从文件中读取数据的方法：fread()、file_get_contents()、file()和readfile()。file_get_contents()、readfile()和fread()以字符串形式返回数据，而file()则将文件中的数据作为数组返回，文件中的每一行均是数组中的一个元素。虽然这四个方法都可以读取文件内容，但只有file_get_contents将文件缓存到内存中，以便更快地进行读/写操作，这种方式称为内存映射。使用内存映射后，file_get_contents在支持其运行的系统中读取小文件时，性能有了大幅提高。

我们将在两个场景中对`fread()`和`file_get_contents()`这两个方法进行比较，一个场景是以字符串形式返回3.9KB小文件中的数据，另一个则是返回2.3MB大文件中的数据（如代码清单3-14所示）。

代码清单3-14 使用`file_get_contents()`从3.9KB大小的文件中读取内容——访问10万次

```
<?php
$fileToFetch = "<YOUR_3.9KB_FILE_STORED_LOCALLY>";

//Access the file 100000 times
$start = microtime();

for($i=0; $i<100000; $i++)
{
    $fileContent = file_get_contents($fileToFetch);
}

$end = microtime()-$start;
echo $end.'ms<br>';
```

代码清单3-14将变量`$fileToFetch`设置为我们用于此次测试的小文件，其含有3.9KB的本地存储数据。如果你打算自己运行代码，请将`<YOUR_3.9KB_FILE_STORED_LOCALLY>`替换为你本地存储文件的路径。此段代码用`microtime()`设置开始时间，并使用`for`循环来运行代码，循环中使用`file_get_contents()`访问该文件并读取其内容10万次。最后，我们计算出执行时间，并将其显示在屏幕上。

执行完10次该PHP代码后，我们使用每次执行的结果计算平均执行时间。使用`file_get_contents`的平均执行时间为0.3730毫秒。

现在让我们来修改代码清单3-14中所示的代码，使用`fopen`和`fread()`访问同一个3.9KB文件并读取其中的数据。更新后的代码如代码清单3-15所示。

代码清单3-15 使用`fread()`从3.9KB大小的文件中读取内容——访问10万次

```
<?php
$fileToFetch = "<YOUR_3.9KB_FILE_STORED_LOCALLY>";

//Access the file 100000 times
$start = microtime();

for($i=0; $i<100000; $i++)
{
    $fileHandler = fopen($fileToFetch, 'r');
    $fileContent = fread($fileHandler, filesize($fileToFetch));
}

$end = microtime()-$start;
echo $end.'ms<br>';
```

以粗体显示的部分有两行代码使用`fopen()`方法创建了一个读取文件处理语句，然后第二行代码在`fread()`方法中使用了该语句。运行这段代码10次，计算出平均执行时间，结果为0.1108

毫秒。比较fread()和file_get_contents(),使用fread()读取数据的时间快了0.2622毫秒,也就是性能上有了70%的提高。

使用代码清单3-16和代码清单3-17所示代码比较同一个函数读取2.3MB大文件的性能,我们可以看到, file_get_contents的表现更佳,平均执行时间为0.012毫秒,而fread()的平均执行时间则为0.019毫秒。这种改进微乎其微,但每一点提升都有其价值所在。表3-2对比了这四种方法的平均执行时间。

代码清单3-16 从2.3MB大小的文件中读取内容——只访问一次

```
<?php
$fileToFetch = "<YOUR_2.3MB_FILE_STORED_LOCALLY>";

$start = microtime();

$fileHandler = fopen($fileToFetch, 'r');
$fileContent = fread($fileHandler, filesize($fileToFetch));

$end = microtime()-$start;
echo $end.'ms<br>';
```

代码清单3-17 从2.3MB大小的文件中读取内容——只访问一次

```
<?php
$fileToFetch = "<YOUR_2.3MB_FILE_STORED_LOCALLY>";

$start = microtime();

$fileContent = file_get_contents($fileToFetch);

$end = microtime()-$start;
echo $end.'ms<br>';
```

表3-2 对不同文件类型的平均执行时间结果

文件读取类型	平均执行时间	文件类型
file_get_contents()	0.3730毫秒	小
fread()	0.1108毫秒	小
file_get_contents()	0.012毫秒	大
fread()	0.019毫秒	大

3.1.6 更快速地访问对象属性

这是在这一章中少数几个我不愿意下笔撰写的主题之一,因为它违反了常规的面向对象编码实践:封装。由于这可以让性能得到100%的提升,因此,由你来决定是否要实现它。

在PHP 5.0和更高版本中,你可以使用类,而且如果用某个类创建了复杂对象,那么你同时也创建了类属性,比如代码清单3-18所示代码。

代码清单3-18 使用了封装的基准Person类

```
<?php
class Person
{
    private $_gender = NULL;
    private $_age = NULL;

    public function getGender()
    {
        return $this->_gender;
    }

    public function getAge()
    {
        return $this->_age;
    }

    public function setGender($gender)
    {
        $this->_gender = $gender;
    }

    public function setAge($age)
    {
        $this->_age = $age;
    }
}

$personObj = new Person();

$start = microtime();
for($i=0; $i<100000; $i++)
{
    $personObj->getGender();
}
echo microtime()-$start.'ms';
```

如代码清单3-18所示，类Person含有类属性age（年龄）以及gender（性别）。它还包含两个方法，取值函数和赋值函数（getter和setter）。这个类的设计采用了封装概念，我们创建方法并将属性设置为私有。此段代码还包含了测试。它运行了10万次循环，实例化Person对象并使用取值函数getGender()读取gender属性。

运行代码清单3-18中所示代码10次，平均的执行时间为0.0443毫秒。现在，让我们删除封装，使用代码清单3-19所示代码重新进行测试。

代码清单3-19 直接读取类属性的基准Person类

```
<?php
class Person
{
```

```
public $_gender = NULL;
public $_age = NULL;

}

$personObj = new Person();

$start = microtime();
for($i=0; $i<100000; $i++)
{
    //Average: 0.0205617ms
    $personObj->_gender;
}

echo microtime()-$start.'ms';
```

代码清单3-19所示代码的平均执行时间为0.0205毫秒，比使用封装的代码快了0.0238毫秒，也就是有了53%的性能提高。

将代码基准化并应用不同的函数是极佳的测试方法，可以衡量出哪种函数最适合使用。唯一的缺点是，我们还是不知道为什么一个函数比另一个快。要了解这个原因，需要研究Opcode并分析哪些函数正在执行以及多少个操作已被调用。而要做到这一点，我们需要引入VLD和strace。

3.2 使用 VLD、strace 和 Xdebug 一探究竟

你现在处于“学习如何钓鱼”的阶段。在本节中，我们将研究每个PHP脚本编译成的Opcode一直到它被Web服务器执行的过程。要让你的系统正确执行PHP脚本，则脚本中的每个函数都会被转换成Opcode中的操作——比如ECHO、CONCAT、ADD_VAR、ASSIGN等操作。在下一章中，我们将详细介绍PHP转换成Opcode的方式；但现在，我们将重点对其进行分析。我们将使用Vulcan Logic Dumper（也就是反汇编程序）在系统层级上分析不同函数的行为。

3.2.1 用 VLD 查看 Opcode 函数

VLD由Derick Rethans编写，并且仅可在基于UNIX的系统上使用。VLD是Zend Engine的插件，可显示脚本在执行时使用的所有Opcode。我们可以使用VLD看到底层操作，包括每个函数都在做什么，正在进行哪些系统调用，最重要的是我们可以用VLD比较表面上看起来类似的PHP函数，也就是轻量vs.重量函数调用。

安装VLD

本章所涉及的许多工具都需要PECL。如果你还没有安装PECL，请安装，它会帮助你节约设置工具的时间和精力。如果你有PECL，让我们执行代码清单3-20所示的PECL命令来安装VLD的beta测试版，在编写本书时，其版本号是vld-0.10.1。

代码清单3-20 使用PECL安装VLD

```
pecl install channel://pecl.php.net/vld-0.10.1
```

完成VLD的安装后,需要使用字符串extension=vld.so将vld.so文件添加到php.ini文件中。打开你的php.ini文件,加入此字符串并保存更改。

我们将使用在本章开头简要介绍过的简单优化技术(在连接字符串时,使用','而不是'.'),让你熟悉此工具的使用。我们将比较代码清单3-21和代码清单3-22中所示代码的系统调用,这两个代码段均使用echo打印同一个字符串,但连接字符串的值并不相同。

代码清单3-21 使用“.”的Echo

```
<?php
echo "Hello"." " . "World!";
```

代码清单3-22 使用“,”的Echo

```
<?php
echo "Hello"," ", "World!";
```

将代码清单3-21中代码保存到文件echo1.php中,将代码清单3-22中代码保存到文件echo2.php中。在shell终端中执行以下命令运行VLD:

```
php -dvld.active=1 echo1.php
php -dvld.active=1 echo2.php
```

在分别执行完以上命令后,你应该能看到类似于图3-4和图3-5的输出。

```
function name: (null)
number of ops: 5
compiled vars: none
line  # * op                                fetch      ext return operands
-----
  2    0 > EXT_STMT
      1    CONCAT                          ~0         'Hello', '+'
      2    CONCAT                          ~1         ~0, 'World%21'
      3    ECHO                             ~1
  3    4  > RETURN                          1

branch: # 0; line: 2- 3; sop: 0; eop: 4
path #1: 0,
```

图3-4 echo1.php的VLD输出

图3-4中包含了每次运行代码清单3-21中所示PHP脚本时所执行的Opcode信息。输出中含有执行的操作的个数、所有变量集输出(compiled vars)以及一个列表,表中包含执行操作的PHP代码行号、执行操作的次数(#)以及所执行操作的名称(op)。

参考VLD提供的输出,我们可以重点关注关键项的数量、op的个数以及执行顺序。number of ops(OP数)标明了代码运行时在Opcode层级执行的操作总个数。在这个例子中,所执行操作的总数是5。向下看结果,参考所列表格,我们可以看到所执行的系统级函数的完整列表。回送“Hello World!”需要两个CONCAT(连接调用)以及一个ECHO调用。现在,让我们看一下使用逗号连接字符串的代码的VLD输出,如图3-5所示。

```

function name: (null)
number of ops: 5
compiled vars: none
line   # * op                                fetch      ext return operands
-----
  2     0 > EXT_STMT
        1   ECHO                                'Hello'
        2   ECHO                                '+'
        3   ECHO                                'World%21'
  3     4 > RETURN                                1

branch: # 0; line: 2- 3; sop: 0; eop: 4
path #1: 0.

```

图3-5 echo2.php的VLD输出——使用逗号连接字符串

参考图3-5中的number of ops, 我们可以看到, 它与图3-4的操作调用次数相同。所不同的是, 这里只用到了ECHO操作, 而没有用到CONCAT操作。图3-5中只包含了比CONCAT调用的成本更低的ECHO调用。

无论脚本的大小如何, VLD都可以用来转储PHP脚本的Opcode。使用代码清单3-11所示代码, 生成的Opcode如图3-6所示。

```

filename:      /var/www/foreach_test.php
function name: (null)
number of ops: 20
compiled vars: !0 = $items, !1 = $start, !2 = $item, !3 = $x
line   # * op                                fetch      ext return operands
-----
  2     0 > SEND_VAL                                0
        1   SEND_VAL                                1000000
        2   SEND_VAL                                '12345678910'
        3   DO_FCALL                                3      'array_fill'
        4   ASSIGN                                !0, $0
  5     5   DO_FCALL                                0      'microtime'
        6   ASSIGN                                !1, $2
  7     7   SEND_REF
        8   DO_FCALL                                1      'reset'
  8     9 > FE_RESET                                $5      !0, ->15
       10 >> FE_FETCH                                $6      $5, ->15
       11 > ZEND_OP_DATA
       12   ASSIGN                                !2, $6
 10    13   ASSIGN                                !3, !2
 11    14 > JMP
       15 > SWITCH_FREE                                $5
 14    16   DO_FCALL                                0      'microtime'
       17   SUB                                ~10     $9, !1
       18   ECHO                                ~10
 15    19 > RETURN                                1

```

图3-6 代码清单3-11所示PHP代码的Opcode

3.2.2 使用 strace 进行 C 级跟踪

我们要使用的另一个工具是strace, 它可以在Apache请求中跟踪所调用的C级函数, 包括PHP级进程。

我们将使用前面提到的一个简单优化技术“使用require而不是require_once或include_once”来测试这个工具。在下面的小节中，你将学到如何在系统上安装strace。

安装strace

使用strace的一个要求是拥有基于Unix的系统并有自由重启Apache的权限。我假设你已经安装了Apache开发环境，所以现在我们将直接安装strace。要安装strace，你有三种选择。

□ 从<http://sourceforge.net/projects/strace>下载源文件，并直接在你的系统中生成文件。

□ 使用apt-get: apt-get install strace安装strace。

□ 使用yum: yum install strace安装strace。

第二和第三个选项的安装过程简单方便，因此建议使用它们。安装完strace后，让我们将strace与Apache进行绑定。你必须执行以下这条命令启动Apache：

```
apache2ctl -X -k [restart OR start].
```

这将以调试模式启动Apache，启动时使用的是单一流程而非子流程。

启动Apache后，你需要通过定位Apache流程ID将strace绑定至Apache。要执行此操作，请打开一个shell窗口并执行这条命令：

```
ps auxw | grep www-data
```

最后，你可以使用以下命令将strace绑定至Apache。完成上述步骤后，不要关闭终端窗口并保持其可见。在执行后续步骤时，该窗口将显示Apache响应HTTP请求时strace的稳定输出流。

```
strace -p <processeID>
```

请加载http://<你运行strace的开发环境>/<使用require_once的代码>.php链接以熟悉strace的使用。你应该可以看到类似图3-7所示的输出。

```
getcwd("/var/www", 4096) = 9
lstat("/var/www/./ClassA.php", {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
lstat("/var/www/ClassA.php", {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
open("/var/www/ClassA.php", O_RDONLY) = 10
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
mmap(NULL, 24, PROT_READ, MAP_SHARED, 10, 0) = 0x7f8de40b4000
munmap(0x7f8de40b4000, 24) = 0
close(10) = 0
getcwd("/var/www", 4096) = 9
lstat("/var/www/./ClassB.php", {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
lstat("/var/www/ClassB.php", {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
open("/var/www/ClassB.php", O_RDONLY) = 10
```

图3-7 使用require_once代码的strace输出片段

图3-7所示的输出片段概括了PHP脚本执行的C级操作。我们重点关注单一的require_once方法调用，看一下图3-7中的第2、3、4行。这几行说明了进行require_once调用时所执行的C级操作。两个lstat调用将ClassA.php导入到你的PHP文件中。

现在加载含有代码清单3-6中所示代码的链接: <http://<你运行strace的开发环境>/<使用require的代码>.php>, 这一次, 你应该会看到类似图3-8所示的响应。

```
getcwd("/var/www", 4096) = 9
lstat("/var/www/./ClassA.php", {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
open("/var/www/ClassA.php", O_RDONLY) = 10
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
mmap(NULL, 24, PROT_READ, MAP_SHARED, 10, 0) = 0x7f8de4135000
munmap(0x7f8de4135000, 24) = 0
close(10) = 0
getcwd("/var/www", 4096) = 9
lstat("/var/www/./ClassB.php", {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
open("/var/www/ClassB.php", O_RDONLY) = 10
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
fstat(10, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
mmap(NULL, 24, PROT_READ, MAP_SHARED, 10, 0) = 0x7f8de4135000
```

图3-8 使用require代码的strace输出片段

图3-8包含了导入ClassA.php所执行的C级操作片段, 其输出的大部分均与图3-7完全一致, 只是缺少了一个lstat操作。strace的输出表明, 对/var/www/ClassA.php的lstat调用将不会被执行, 因此加快了文件的导入速度, 也提高了你的PHP脚本性能。

3.3 发现瓶颈

在编写代码时, 瓶颈应该是需要考虑的关键问题之一。它们是在你的应用程序中占用了与其他部分相比更多时间和处理资源的层。在连接网络服务或I/O功能(比如打开本地存储的XML文件)时, Web应用程序往往会在数据库连接层中出现瓶颈。在本节中, 我们将使用一种工具, 帮助我们识别代码中的瓶颈。

3.3.1 Xdebug 2: PHP 调试工具

Xdebug是面向PHP的调试器和概要分析工具。除了更多的调试信息外, Xdebug还可以为开发人员提供如下的信息:

- ☐ PHP脚本的内存消耗;
- ☐ 对某个函数执行的调用总数;
- ☐ 函数内部花费的总时间;
- ☐ 某个函数的完整栈跟踪。

现在, 让我们安装这个工具, 并用它运行一些例子, 来发现我们代码中存在的瓶颈。

1. 安装Xdebug

安装Xdebug有两种方法: 使用PECL或从源文件生成。我将介绍使用PECL安装该工具的方法,

因为它的安装速度更快，并且也没有那么多隐患。要安装它，你需要安装PHP 5.0及以上版本和PECL。安装完这两样后，请运行代码清单3-23所示的命令。

代码清单3-23 使用PECL安装Xdebug

```
pecl install xdebug
```

完成Xdebug的安装后，你需要让PHP在每次运行时对你的脚本进行概要分析。有两种方法可供选择：

- 更新php.ini文件；
- 在PHP脚本的开头处添加声明。

我们会在后面介绍这两种方法，前一种适用于拥有对PHP环境完全访问权限的用户，第二种方法则适合无法访问php.ini文件的用户使用。

2. 更新PHP.ini文件

要自动运行Xdebug，你需要开启这个扩展。打开php.ini文件并添加属性zend_extension_ts，如代码清单3-24所示。代码中的文本包含的是绝对路径，指向所需加载的Xdebug线程安全的扩展文件。

代码清单3-24 安装Xdebug扩展

```
[PHP_Xdebug]
zend_extension_ts="FULL PATH TO php_xdebug file"
```

此外，在使用Xdebug之前，你还需要在php.ini中指定5个Xdebug的特定属性，如表3-3所示。

表3-3 Xdebug属性

Xdebug属性	说 明
xdebug.profiler_enabled	开启 (1) 或关闭 (0) 性能分析工具
xdebug.profiler_output_dir	放置cachegrind文件的目录。默认设置为/tmp
xdebug.profiler_append	对PHP脚本发出新请求时，覆盖文件 (1)。默认情况下此设置为关 (0)
xdebug.profiler_output_name	所用的文件名称
xdebug.profiler_enable_trigger	允许性能分析工具开始使用一个GET/POST或COOKIE变量：XDEBUG_PROFILE。Xdebug.profiler_enable必须设置为0

将代码清单3-24和代码清单3-25中的数据复制到你的PHP.ini文件中，然后重新启动Web服务器。

代码清单3-25 PHP.ini属性

```
xdebug.profiler_enable = 1
xdebug.profiler_enable_trigger = 1
xdebug.profiler_output_dir="ABSOLUTE PATH TO XDEBUG LOGS DIRECTORY"
xdebug.profiler_append=0n
xdebug.profiler_output_name = "cachegrind"
```

3.3.2 验证安装

设置好你的环境后, 创建一个phpinfo()文件, 以此来检查扩展是否已安装。如果扩展已成功安装, 你应该可以看到如图3-9所示的Xdebug信息。

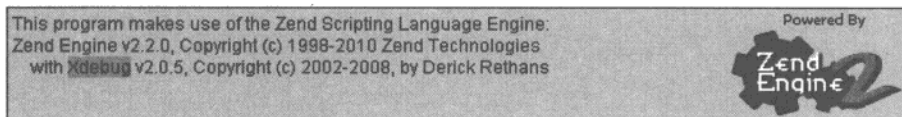


图3-9 安装了Xdebug的phpinfo打印结果

运行第一个性能分析工具

在将Xdebug应用于PHP代码之前, 你是体会不到它有多么强大的。我们将对一小段代码进行概要分析, 以熟悉运行Xdebug的流程并分析部分输出结果 (如代码清单3-26所示)。

代码清单3-26 PHP代码示例

```
<?php
function bar($items)
{
    for($i=0; $i<count($items); $i++)
    {
        if(isInt($items[$i]))
        {
            echo 10*20*$items[$i];
        }
    }
}

function isInt($value)
{
    if(is_int($value))
    {
        return true;
    }
    else
    {
        return false;
    }
}

$sints = array(1,2,"E",4,5,6,"T",8,9,"o");
$sints2 = array(0,1,2,3,4,5,6,7,8,9);

bar($sints);
bar($sints2);

echo "Done!";
```

代码清单3-26所示的代码包含两个函数，`bar()`和`isInt()`。函数`bar()`接受一个数组作为参数，并调用`isInt()`函数来验证数组元素是否是整数，如果该值是一个整数，则将其乘以10和20。如此操作20次，这是数组`$ints`和`$ints2`中含有的元素总数。

开启Xdebug设置并运行上述代码，然后查看`xdebug.profiler_output_dir`属性中所使用的目录。你应该会看到目录中出现了一个文件。

打开这个文件你会看到一个目录。会看到类似代码清单3-27所示的文件。这是`cachegrind`输出文件，你可以看出，它们似乎有点令人费解。我们需要一个工具来解读这种输出文件并为我们提供看得懂的信息。

代码清单3-27 Xdebug概要分析输出文件

```
version: 0.9.6
cmd: C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\index.php
part: 1

events: Time

fl=php:internal
fn=php::count
4 1

fl=php:internal
fn=php::is_int
15 1

fl=C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\index.php
fn=isInt
6 110
cfn=php::is_int
calls=1 0 0
15 1
...
```

3.3.3 安装基于 GUI 的工具

Xdebug是一个独立工具，作为调试器，它表现很出色，但是它另外需要一个有力的概要分析工具。有两个版本的基于GUI的工具：Windows版本的WinCacheGrind和Linux（KDE）版本的KDECacheGrind，后者也可以使用Cygwin安装在Windows上。

与KDECacheGrind不同，WinCacheGrind的功能有限并且存在一些缺陷。此外，WinCacheGrind还缺少直观图表来帮助我们查看每个函数调用的执行方式以及每次调用所花费的时间。

1. 安装WinCacheGrind

安装Windows版本的WinCacheGrid非常容易。先在SourceForge上下载可执行文件，网址是<http://sourceforge.net/projects/wincachegrind/>。下载完成后，双击可执行文件即可加载WinCacheGrid文件了。

2. 安装KCacheGrind

使用KDE的Linux用户可以从以下网址下载tar.gz文件：<http://kcachegrind.sourceforge.net/html/Download.html>。下载完成后，将文件解压缩到所选目录，然后运行代码清单3-28中所示的命令。

代码清单3-28 KCacheGrind安装命令

```
./configure
make
make install OR sudo make install
```

大功告成。

3. 分析数据

运行代码清单3-27中的代码，在创建了cachegrind文件后，使用工具WinCacheGrind或KCacheGrind打开该文件。我将使用WinCacheGrind。

根据cachegrind输出文件所提供的信息，处理PHP脚本花费的总时间为4.4毫秒，而PHP脚本执行了64次函数调用。

我们将看到的第一个视图含有以下信息：所执行的函数调用（如图3-10所示）、函数内部花费的平均时间、累计平均时间、函数内部花费的总时间、总累积时间、函数调用的总次数。

Function	Avg Self	Avg Cum	Total Self	Total Cum	Calls
(main)	0.2ms	4.6ms	0.2ms	4.6ms	1
bar	1.6ms	2.2ms	3.2ms	4.4ms	2
isInt	-	-	1.2ms	1.2ms	20
php::count	-	-	-	-	22
php::is_int	-	-	-	-	20

图3-10 代码清单3-26中代码的函数概要分析信息

图3-10共显示了五个函数调用，其中也包括我们的根函数，也就是调用其他函数的顶层主函数。根据收集到的信息可知，count()函数被调用了22次，isInt()被调用了20次，而is_int同样也被调用了20次。我们可以根据这些数字快速识别出频繁被调用的函数。从该图可以读出的另一条信息是完成一个函数所花费的总时间。在这个例子中，php::count()函数的调用次数最多。

保持WinCacheGrind工具开启状态，单击php::is_int函数。你应该可以看到如图3-11所示的窗口视图。此窗口显示的信息包括函数花费的时间、累计时间、被调用的函数、调用函数的文件以及栈跟踪。

Num.	Self	Cum.	Called by	Called from	Stack trace
1	-	-	isInt	index.php (15)	isInt < bar < (main)
2	-	-	isInt	index.php (15)	isInt < bar < (main)
3	-	-	isInt	index.php (15)	isInt < bar < (main)
4	-	-	isInt	index.php (15)	isInt < bar < (main)
5	-	-	isInt	index.php (15)	isInt < bar < (main)
6	-	-	isInt	index.php (15)	isInt < bar < (main)
7	-	-	isInt	index.php (15)	isInt < bar < (main)
8	-	-	isInt	index.php (15)	isInt < bar < (main)
9	-	-	isInt	index.php (15)	isInt < bar < (main)
10	-	-	isInt	index.php (15)	isInt < bar < (main)
11	-	-	isInt	index.php (15)	isInt < bar < (main)
12	-	-	isInt	index.php (15)	isInt < bar < (main)
13	-	-	isInt	index.php (15)	isInt < bar < (main)
14	-	-	isInt	index.php (15)	isInt < bar < (main)
15	-	-	isInt	index.php (15)	isInt < bar < (main)
16	-	-	isInt	index.php (15)	isInt < bar < (main)
17	-	-	isInt	index.php (15)	isInt < bar < (main)
18	-	-	isInt	index.php (15)	isInt < bar < (main)
19	-	-	isInt	index.php (15)	isInt < bar < (main)

图3-11 php::is_int函数调用结果

图3-11所示的数据分为6列。我们要看的是最后一列“Stack trace”（栈跟踪）。每一行数据均代表对函数的一次调用，因此，我们需要确定调用是如何到达函数的。从图中的数据可以看出，调用过程从主PHP流开始，到bar()函数，最后到达函数isInt()。在这里，共需要三次跳转才能到达我们所需的函数，而其实现的只是检查某个值是否是整数。

在开始通过合并函数进行重构以减少跳转次数之前，请先问问自己，这么做是否值得。在大多数情况下，函数被分解成易于管理的多个代码段，以帮助你完成具体的任务。将函数逻辑合并在一起只会造成混乱，因此，请在重构前仔细思考。

在我们的例子中，这种情况下，让我们进行重构并重新运行脚本，以生成新的输出文件。更新后的代码如代码清单3-29所示。

代码清单3-29 优化后的代码——合并了函数逻辑并在循环前计算出循环次数

```
<?php
function bar($items)
{
    $count = count($items);
    for($i=0; $i<$count; $i++)
    {
        if(is_int($items[$i]))
        {
            echo 10*20*$items[$i];
        }
    }
}

$sints = array(1,2,"E",4,5,6,"T",8,9,"o");
$sints2 = array(0,1,2,3,4,5,6,7,8,9);

bar($sints);
bar($sints2);

echo "Done!";
```

代码清单3-29中包含了优化后的代码，函数isInt()被移到了函数bar()中，而count()则被放到了for循环外。执行所示代码，将生成的cachegrind输出结果放到GUI工具中，累计时间应该会有显著减少。

现在的累计时间是1.9毫秒，而不是4.4毫秒，函数调用的总次数也从64次降至44次，效果非常明显。

此外，我们还消除了三次跳转的开销（这三次跳转仅仅是为了查看某个值是否为整数）。现在，在调用is_int()之前，我们只需进行两次跳转，如图3-12所示。

Num.	Self	Cum.	Called by	Called from	Stack trace
1	-	-	bar	index.php (7)	bar < (main)
2	-	-	bar	index.php (7)	bar < (main)
3	-	-	bar	index.php (7)	bar < (main)
4	-	-	bar	index.php (7)	bar < (main)
5	-	-	bar	index.php (7)	bar < (main)
6	-	-	bar	index.php (7)	bar < (main)
7	-	-	bar	index.php (7)	bar < (main)

图3-12 到达is_int函数的栈跟踪

Xdebug可以对PHP脚本进行深入分析并能对任何应用程序进行微调，加快其运行速度。熟练掌握了一些最佳实践后，让我们开始研究如何运用缓存进一步优化代码。

3.4 小结

本章以用于对PHP应用程序进行概要分析的两个工具为基础，讨论了一些当前的PHP最佳实践，这些实践可以提高应用程序的性能，比如以下几个方面：

- 使用require代替require_once；
- 提前计算for循环的长度；
- 在访问数组中的元素时，使用foreach代替while；
- 在访问小文件中的数据时，使用fread()；大文件则使用file_get_contents；
- 更快地访问对象属性。

此外，你还学习了如何深入挖掘，使用VLD分析PHP Opcode；同时你还安装了strace，用其查看每个函数执行的系统调用，确定哪个函数的性能更好。你还了解了关于Xdebug的知识：如何安装Xdebug和使用基于GUI的工具WinCacheGrind和KDECacheGrind。

现在，我们将深入研究Opcode，包括如何生成Opcode、Opcode是什么以及如何对其进行缓存。

现在，要加快PHP脚本的运行速度，我们要做的是在对PHP脚本进行请求时删除所有不必要的进程。我们可以通过删除在PHP生命周期中无需常规执行的进程来优化应用程序，从而使其对用户的任何请求做出更快的响应。

在上一章中，我们简要提到了Opcode，并对其进行了分析。在这一章中，我们将更深入地研究Opcode，通过细致查看PHP生命周期的每个步骤并查明由Zend引擎执行的PHP脚本有什么变化。你还将了解到关于Opcode缓存的信息以及缓存是如何用于加速PHP生命周期的。最后，你将学到以下缓存工具：

- ❑ Alternative PHP Cache (APC)
- ❑ XCache
- ❑ eAccelerator

我们会分别在Windows和Unix系统中安装上述所有缓存解决方案，并提供基准测量数据以确定使用Opcode缓存的好处。我们还将用一些篇幅描述每个缓存解决方案的设置，帮助深入理解如何调整设置以进一步提高PHP应用程序的性能。

4.1 回顾路线图

在继续之前，我们需要从一个高层视角来概括一下我们目前在PHP应用程序优化中所处的位置。图4-1是本书开头展示的路线图，它描述了如今大部分PHP应用程序的每一层。在这张路线图上，我们已越来越接近现代化的PHP应用程序，现在，我们处在PHP Opcode缓存部分，也就是图4-1的阴影区域。

Opcode是PHP非常重要的一个组件，在开始对其进行缓存前，我们需要知道它是如何生成的。在上一章中，我们说明了什么是Opcode，但是没有涉及它的生成方式。现在，我们将对此进行说明，让我们先详细回顾PHP的生命周期。



图4-1 PHP应用程序优化图

4.2 PHP 的生命周期

无论用命令行还是从Web服务器上请求PHP脚本时，PHP必须执行所需的5个步骤，如图4-2中的文字框所示。Zend引擎必须从文件系统中读取文件、扫描其词典和表达式、解析文件、创建要执行的计算机代码（称为Opcode），最后执行Opcode。

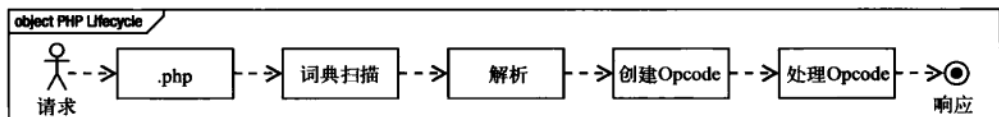


图4-2 PHP的生命周期

PHP的生命周期非常短暂，但是，每次对脚本发出请求时，它必须执行图4-2中列出的所有步骤。每次在针对特定PHP脚本的请求到达时，即使该PHP脚本的内容没有任何变化，Zend引擎也

必须重新创建该文件的Opcode。对于脚本的初次请求，这是必要的，但后续请求则无需如此操作。如果实现Opcode缓存，我们就可以省略以下3个步骤以缩短PHP的生命周期，从而提高应用程序的性能：

- (1) 词典扫描；
- (2) 解析；
- (3) 创建Opcode。

优化后的最终PHP生命周期如图4-3所示。

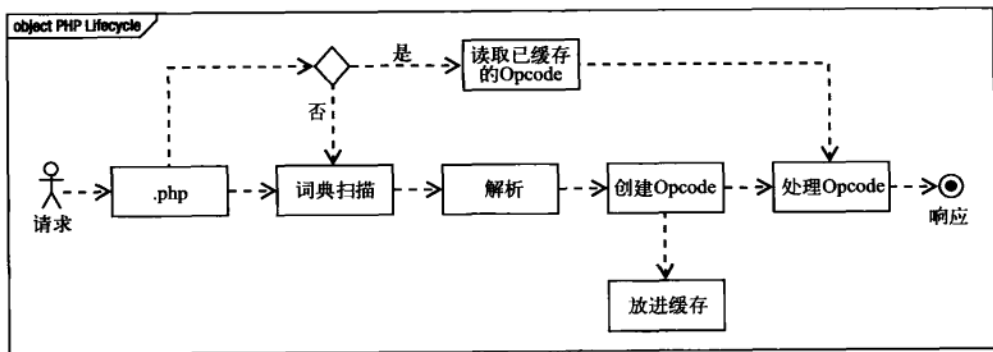


图4-3 应用了Opcode缓存的PHP生命周期

注意 你应该很熟悉缓存理论了，如果不是，也不用担心。概括地说，缓存是在共享内存中放置数据以供过后读取的技术，其访问速度比实际从硬盘中读取数据要快得多。

图4-3中包含了在调用PHP脚本时请求可以采用的两种途径：对脚本的初次请求所采取的初始路径以及后续脚本请求所采取的路径。在初次用户请求中，会采用如图4-2所示的路径，但在执行词典扫描前还将执行其他两个步骤。Zend引擎执行的额外步骤是在缓存内部检查已生成的Opcode。如果之前没有生成Opcode，则引擎将继续进行词典扫描，直至到达创建Opcode这一步。生成Opcode后，将其放置在Opcode缓存中，然后继续。执行这个步骤，则之后对这个PHP脚本的任何请求都可以从共享内存中读取Opcode，而不用自行创建Opcode。在这条路径中，进程在分析之前需要多执行两个步骤，这可能会稍微降低PHP脚本初次请求的速度。

这么做的真正好处在于，后续请求可以采用第二条路径。第二条路径从请求.php文件开始，然后检查共享内存，确定Opcode是否已被成功缓存。由于初始请求创建了Opcode并将其存储在缓存中，所以，Zend引擎可以从缓存中获取Opcode并使用它。这样，要满足用户的请求，PHP的生命周期可以省去三个步骤：词典扫描、解析和创建Opcode，如图4-3所示。

现在，让我们看看适用于PHP的缓存工具。

4.3 Opcode 缓存工具

在接下来的几节中，我们将研究在真实PHP项目中已被有效使用的三种Opcode缓存技术：Alternative PHP Cache（APC）、XCache和eAccelerator（eA）。其中一些Opcode缓存解决方案（比如APC）能够缓存的不仅仅是Opcode，但是在本章中，我们将仅讲述缓存Opcode方面的内容。

4.3.1 Alternative PHP Cache

Alternative PHP Cache（APC）是一个PECL扩展，可用于Unix和Windows服务器。APC直接安装在Zend引擎上，如果请求正在进行且没有过期，它可以提供一种缓存系统，将该请求重定向至已缓存的Opcode。APC使用共享内存和映射表来为特定PHP脚本获取Opcode。

1. 安装APC

如前所述，APC可用于基于Windows和Unix的系统。在撰写本书时，APC的最新版本是3.1.3，并可使用PECL进行安装。我们将使用PECL安装APC，建议你在继续之前先检查系统上是否安装了PECL。如果你的系统尚未安装PECL，请查看附录中安装PECL的完整步骤指南。

● 在Unix系统上安装

如果你已安装了PECL，则安装APC很容易。在基于Unix的系统上，打开一个shell窗口，然后运行以下命令：`sudo pecl install apc`。PECL将从互联网上读取所需的软件包及其关联程序。

完成安装后，你需要更新正在使用的`php.ini`文件，无论是Web服务器加载的`php.ini`还是命令行工具使用的`php.ini`文件都需要更新。打开`php.ini`文件并添加代码清单4-1中的代码行。这样，PHP可以将APC的扩展作为PHP的扩展加载并设置两个APC设置，我们将在本节后面讨论到。保存对文件的修改，然后重新启动Web服务器使更改生效。

警告 撰写本书时，我在为Linux上安装APC的过程中，在这一步遇到了问题。如果你也遇到了类似于以下提示的错误，可以尝试从源文件生成代码包。

```
'/tmp/pear/temp/APC/php_apc.c:959: error: duplicate 'static'
make: *** [php_apc.lo] Error 1
ERROR: 'make' failed'
```

也可以运行以下命令安装APC的beta版。

```
sudo pecl install apc-beta
```

代码清单4-1 在PHP中加载APC扩展

```
extension=apc.so
apc.enabled=1
apc.stat=1
```

为了确保APC已正确安装,请打开你习惯使用的编辑器,然后创建一个phpinfo脚本,通常类似于代码清单4-2中所示的代码。

代码清单4-2 简单的phpinfo脚本

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

如果APC已正确安装,你应该可以看到PHP的扩展APC在phpinfo页面加载的设置,如图4-4所示。

apc

APC Support		enabled
Version	3.1.3p1	
MMAP Support	Enabled	
MMAP File Mask	no value	
Locking type	pthread mutex Locks	
Revision	\$Revision: 286798 \$	
Build Date	Jul 25 2010 15:55:58	

Directive	Local Value	Master Value
apc.cache_by_default	On	On

图4-4 phpinfo页面显示的APC扩展

2. 在Windows系统上安装

在基于Windows的计算机上安装APC有点麻烦。如果不使用基于Windows的计算机,可以跳到下一节。Windows用户必须为自己的系统找到一个预编译的(.dll) APC扩展。运行PECL的安装命令很可能会出现如代码清单4-3所示的错误提示。

代码清单4-3 在Windows上用PECL进行安装

```
running: msdev APC.dsp /MAKE "APC - Release"
ERROR: Did not understand the completion status returned from msdev.exe.
```

以下链接可能会对你有所帮助: <http://downloads.php.net/pierre/>。里面有你需要用到的.dll文件。在这个例子中,我下载了软件包hp_apc-3.0.19-5.2-Win32-VC6-x86.zip,然后解压缩。解压缩后,我将文件php_apc.dll放到了PHP扩展文件夹中。使用安装PHP时默认的PHP目录位置,完整路径是C:\Program Files\PHP\ext。如果该目录不存在,可以引用php.ini文件的设置extension_dir,从而确定指向扩展目录的路径。PHP将用此处指定的目录加载任意的外部模块/扩展。

最后,打开正在使用的php.ini文件,将代码清单4-4中所示的代码行添加到文件末尾。保存更改并重新启动Web服务器。

代码清单4-4 Windows系统中php.ini的APC设置

```
extension=php_apc.dll
```

```
apc.enabled=1
apc.stat=1
```

代码清单4-4将加载.dll文件并设置两个APC设置,我们将在本章后面更详细地介绍。如果.dll文件不在PHP的ext目录中,则扩展设置必须包含指向该文件的完整路径。

要验证安装是否成功,可创建一个phpinfo PHP脚本,包含代码清单4-3中的代码,然后使用Web浏览器加载该页面。如果APC已正确安装,你应该可以看到类似于图4-4的界面。

3. 使用APC

使用APC很简单,你可以像往常一样创建PHP脚本,然后让用户请求该脚本。对脚本的初始请求将写入缓存,也就是说,在进行脚本的初始请求或对原有PHP脚本进行更改时,Zend引擎将在后台创建Opcode并将生成的Opcode使用APC存储到共享内存中。APC在后台工作,并且没有使用任何特殊函数来缓存Zend引擎所生成的Opcode(但是APC含有一些用于缓存可变数据的函数,我们将在第5章讲到)。

现在,让我们使用代码清单4-5所示代码来运行示例,以确认在应用程序中使用APC的好处。代码清单4-5所示的代码创建了一个包含1万个元素的数组,然后将该数据显示在屏幕上。

代码清单4-5 使用ab进行测试的示例代码

```
<?php
$max = 10000;
$x = 0;
$array = array();
while($x < $max)
{
    $array[$x] = $x;
    $x++;
}

foreach($array as $z)
{
    echo "$z<br/>";
}
```

我们需要查看在将APC应用于PHP脚本后代码性能的提高。要查看性能提高效果,需要进行两次测试,初始测试在未启用APC时模拟应用程序的速度,第二次测试则开启APC,以测量代码性能上的变化。测试将使用ab模拟1000次请求,请求的并发数为5个,如代码清单4-6所示。

代码清单4-6 测试APC的ab命令

```
ab -n 1000 -c 5 http://localhost/test.php
```

在运行ab模拟前,我们需要禁用APC,方法是將apc.enabled设置为“0”,然后重新启动Web服务器使更改生效。请确保APC已被禁用,方法是加载phpinfo页面,确定其不再显示APC设置数据。

执行该命令五次之后，我选取了最佳结果，如图4-5所示。

```

Concurrency Level:      5
Time taken for tests:    4.013 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      89081000 bytes
HTML transferred:       88890000 bytes
Requests per second:    249.18 [#/sec] (mean)
Time per request:       20.066 [ms] (mean)
Time per request:       4.013 [ms] (mean, across all concurrent requests)
Transfer rate:          21676.93 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0      2
Processing:     11   20   5.7     18     47
Waiting:        0    7   3.0      6     26
Total:         11   20   5.7     18     47

```

图4-5 未使用Opcode缓存时，代码清单4-4所示代码的ab结果

结果表明，Web服务器可以在30.06毫秒完成模拟加载，并且每秒可以满足249.18个请求，平均每个并发请求的执行时间为4.013毫秒。现在，开启APC并进行同样的测试。

再次打开php.ini文件，并将apc.enabled设置从0改为1。此操作将启动APC。重新启动Web服务器，让其加载新的设置。再次运行ab命令，应该可以看到类似于图4-6的结果。

```

Concurrency Level:      5
Time taken for tests:    3.934 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      89081000 bytes
HTML transferred:       88890000 bytes
Requests per second:    254.20 [#/sec] (mean)
Time per request:       19.670 [ms] (mean)
Time per request:       3.934 [ms] (mean, across all concurrent requests)
Transfer rate:          22113.60 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0     15
Processing:     11   20   5.8     17     50
Waiting:        0    7   2.8      5     24
Total:         11   20   5.8     17     51

```

图4-6 使用Opcode缓存时，代码清单4-4所示代码的ab结果

使用APC的好处应该显而易见了。未使用Opcode缓存时，每秒可执行249.18个请求，而现在，使用Opcode缓存后，我们的Web服务器每秒可以执行254.20个请求。Web服务器的响应时间也有所减少。使用APC默认设置的初始数值，我们的性能有了一些提高。要判断速度是否可以进一步加快，需要深入研究APC设置并决定应该启用或禁用哪些APC默认设置来提高性能。

4. APC设置

APC为开发人员提供了可以从`php.ini`文件中控制APC的设置。在配置APC时,我们已见过了其中几个设置,如代码清单4-1和4-4所示。在这两段代码中,我们设置了`apc.enabled`和`apc.stat`。

`apc.enabled`设置使用整数型数值0或1来禁用或启用APC。默认情况下,该设置为1。另一个设置`apc.stat`则可以检查对PHP脚本的更改,无论是初始请求还是后续请求,脚本更改都会在请求脚本时被缓存。显然,此设置增加了Opcode缓存生命周期的开销,而且在大多数情况下,可以很安全地禁用`apc.stat`。请记住,禁用该设置后,每当需要对已缓存的PHP脚本进行更改时,均需重新启动Web服务器。

表4-1中列出了其他设置,其中包括最常用的设置以及它们的说明。完整的设置列表可以从www.php.net/apc获取。

表4-1 常用的APC设置

设置名称	说 明
<code>apc.cache_by_default</code>	默认启用缓存。1表示“启用”状态。0表示“禁用”状态
<code>apc.filters</code>	根据逗号分隔的POSIX正则表达式判断文件需要缓存还是不需要缓存。以 <code>a+</code> 开头的正则表达式将强制APC不缓存与此正则表达式匹配的任何文件。以 <code>a-</code> 开头的正则表达式将强制APC缓存与此正则表达式匹配的任何文件
<code>apc.stat</code>	启用或禁用APC对于所请求PHP脚本是否有更改的检查。每次调用脚本时均会执行此过程。如果禁用该设置,在对PHP脚本进行任意更改后均需要重新启动Web服务器,以清除缓存并更改脚本内容。0将禁用该设置。1将启用该设置。默认值为1
<code>apc.enabled</code>	启用或禁用APC缓存。1将启用APC。0将禁用APC。默认值为1
<code>apc.shm_size</code>	设置APC允许使用的共享内存大小。此值以兆字节为单位
<code>apc.shm_segments</code>	设置可用的共享内存段总数
<code>apc.include_once_override</code>	启用或禁用对 <code>include_once</code> 和 <code>require_once</code> 的优化。启用该设置时,可减少PHP内部函数进行的额外系统调用。1将启用该设置。0将禁用该设置。该设置的默认状态为禁用
<code>apc.optimization</code>	设置优化级别。将此值设置为0将禁用优化功能,而设置为较高数值可提高优化级别
<code>apc.num_files_hint</code>	设置你认为需要缓存的文件数。默认值是1000。如果不确定文件数,可以设置为0。设置为接近真实文件数的数值往往可以提高一些性能
<code>apc.ttl</code>	设置文件存储在缓存中的过期时间,以秒为单位。到达过期时间时,符合过期时间条件的文件将会从缓存中清除
<code>apc.write_lock</code>	开启该设置将强制单个进程缓存特定的脚本。适用于必须缓存多个文件的大流量Web服务器或应用程序

我们可以使用上表中所列的一些新设置,再次调整初始设置,方法是打开`php.ini`文件,按照代码清单4-7配置APC设置。

代码清单4-7 配置设置的使用示例

```
;APC
extension=apc.so
apc.enabled=on
```

```

apc.shm_size=16
apc.include_once_override=1
apc.write_lock=1
apc.optimization=9
apc.stat=0
apc.num_files_hint=5

```

代码清单4-7中的设置及其数值实现了以下功能：使用`apc.enabled`设置开启了APC，将共享内存大小设置为16兆字节，开启了`include_once`优化，开启了强制写入，设置了优化级别，使用`apc.stat`禁用了修改时间检查并使用APC设置`apc.num_files_hint`将缓存文件总数设置为5。保存此`php.ini`文件并重新启动Web服务器，然后使用`phpinfo`页面验证新设置已被正确配置，如图4-7所示。

Directive	Local Value	Master Value
<code>apc.cache_by_default</code>	On	On
<code>apc.canonicalize</code>	On	On
<code>apc.coredump_unmap</code>	Off	Off
<code>apc.enable_cli</code>	Off	Off
<code>apc.enabled</code>	On	On
<code>apc.file_md5</code>	Off	Off
<code>apc.file_update_protection</code>	2	2
<code>apc.filters</code>	<i>no value</i>	<i>no value</i>
<code>apc.gc_ttl</code>	3600	3600
<code>apc.include_once_override</code>	On	On
<code>apc.lazy_classes</code>	Off	Off
<code>apc.lazy_functions</code>	Off	Off
<code>apc.max_file_size</code>	1M	1M
<code>apc.mmap_file_mask</code>	<i>no value</i>	<i>no value</i>
<code>apc.num_files_hint</code>	5	5
<code>apc.preload_path</code>	<i>no value</i>	<i>no value</i>
<code>apc.report_autofilter</code>	Off	Off
<code>apc.rfc1867</code>	Off	Off
<code>apc.rfc1867_freq</code>	0	0
<code>apc.rfc1867_name</code>	APC_UPLOAD_PROGRESS	APC_UPLOAD_PROGRESS
<code>apc.rfc1867_prefix</code>	upload_	upload_
<code>apc.rfc1867_ttl</code>	3600	3600
<code>apc.shm_segments</code>	1	1
<code>apc.shm_size</code>	16	16
<code>apc.stat</code>	Off	Off
<code>apc.stat_ctime</code>	Off	Off
<code>apc.ttl</code>	0	0
<code>apc.use_request_time</code>	On	On
<code>apc.user_entries_hint</code>	4096	4096
<code>apc.user_ttl</code>	0	0
<code>apc.write_lock</code>	On	On

图4-7 优化后的APC设置

5. APC管理工具

APC自带了管理工具，便于开发人员查看APC缓存的工作状况。管理工具中提供的信息包括APC正在运行的设置、为缓存分配的总大小、使用量、缓存脚本的总数及其名称，开发人员可以在一个网页界面中查看所有的更新。

6. 安装管理工具

每个APC安装程序均包含一个apc.php文件，用于安装此网页界面。该文件是运行此网页界面唯一所需的文件，并且必须直接安装在Web服务器上以便访问。

如果你是从源文件安装APC的，或者如果你使用的是Windows系统，则需要下载安装包。该文件就在安装包里。相反，如果你一直是跟着我的操作从发布源进行安装，则可以使用find命令或Unix的locate命令定位该文件。也可以尝试在路径/usr/share/php/apc.php中查找该文件。

找到该文件后，将其复制并粘贴至你的Web服务器。这样，便可以用浏览器访问以下链接来读取PHP脚本了：http://YOUR_HOST/apc.php，其中的YOUR_HOST可以是本地主机，也可以是你目前用于开发工作的主机。

接下来，我们需要更新脚本本身，通过更新脚本中的常变量ADMIN_PASSWORD。设置这个密码后，在登录到网页界面时你将能够看到更多的功能，比如清空缓存。保存更改并重新启动Web服务器。加载链接http://YOUR_HOST/apc.php，你应该可以看到如图4-8所示的网页界面。

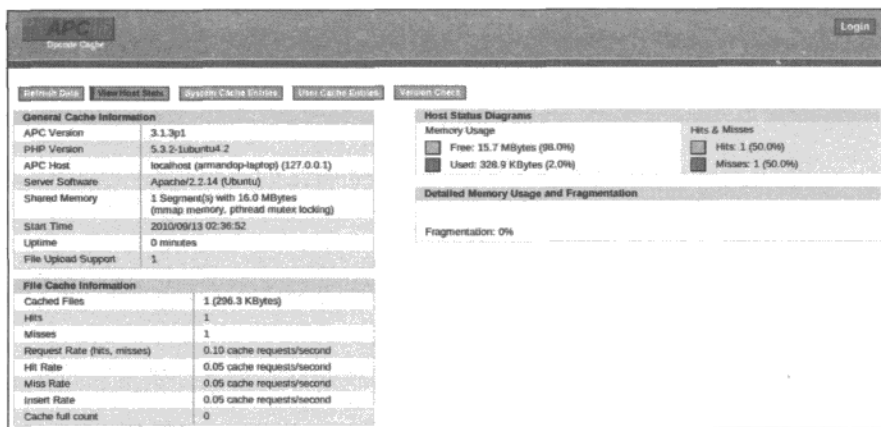


图4-8 APC管理工具主页

主页上包含常用的缓存信息，比如Web服务器正在运行的APC和PHP版本信息、Web服务器软件名称、共享内存类型、文件缓存信息、内存使用信息以及其他你可能需要的有用数据。在主页的页面顶部还有五个按钮。这些子选项可以提供更多信息，比如哪些脚本已被缓存（如图4-9所示）、用户缓存项目信息、对任意更新版本的查询，在你登录后，还会显示一个清空Opcode缓存的选项以及Pre-Directory输入列表，如图4-9所示。

Script Filename	Hits	Size	Last accessed	Last modified	Created at	Deleted at
/var/www/apc.php	4	303380	2010/09/13 02:46:09	2010/09/13 02:33:24	2010/09/13 02:45:45	
/var/www/test.php	0	4164	2010/09/13 02:45:58	2010/09/13 00:01:32	2010/09/13 02:45:58	
/var/www/info.php	0	4164	2010/09/13 02:46:01	2010/07/24 22:19:08	2010/09/13 02:46:01	
/var/www/foreach_test.php	0	4164	2010/09/13 02:46:04	2010/08/26 23:16:44	2010/09/13 02:46:04	

图4-9 APC管理工具在登录后的“System Cache Entries”（系统缓存输入）部分

APC是一个非常棒的Opcode缓存工具，但它不是唯一的工具。要获得良好性能，我们必须了解其他的工具，然后再决定使用哪个工具最能满足我们的需求。下一个要研究的工具是XCache。

4.3.2 XCache

XCache是另一种在PHP中使用的Opcode缓存工具。像APC一样，XCache在共享内存中存储Opcode，并使用缓存的Opcode来响应对PHP脚本的请求。

XCache也与APC一样，可以在基于Unix的系统和Windows系统上使用。在撰写本书时，XCache 1.2.X是最稳定的版本。在这里，我将安装XCache 1.2.2并用其测试Opcode缓存。

1. 在Unix系统上安装

XCache可以从任意资源库中下载安装，如果你希望用源文件安装，也可以从其官方网站<http://xcache.lighttpd.net>上获取。我建议你在用源文件安装前，先尝试使用表4-2中所示的一个命令从资源库中安装。这么操作可以自动下载并安装你的系统可能需要的任何关联程序。

表4-2 从资源库中安装XCache的Unix命令

发布系统	命 令
Red Hat/Fedora	yum install xcache
Debian/Ubuntu	sudo apt-get install php5-xcache

XCache的安装完成后，请务必重新启动Web服务器，并使用phpinfo脚本验证XCache是否已正确加载。如果该扩展已正确安装并加载，你应该可以看到类似于图4-10所示的输出结果。

2. 在Windows系统上安装

与在Unix系统上安装相比，在Windows系统上安装XCache需要执行更多的步骤。你需要与Windows和PHP版本都相匹配的已编译扩展。由于PECL不包含XCache的Windows安装，因此我们需要在XCache的官方网站上查找合适的.dll文件。加载网站<http://xcache.lighttpd.net/pub/ReleaseArchive>，然后根据你的PHP版本，下载合适的软件包。比如，要安装使用PHP 5.3的XCache，可以下载最新的XCache二进制文件xcache-1.3.0-php-5.3.0-win32-vc9-x86.zip。

XCache

XCache Support	enabled
Version	1.3.0
Modules Built	cacher optimizer coverage assembler encoder decoder
Readonly Protection	N/A
Cache Init Time	2010-07-26 21:48:51
Cache Instance Id	1178
Opcache Cache	enabled, 16,777,216 bytes, 1 split(s), with 8192 slots each
Variable Cache	disabled
Shared Memory Schemes	mmap
Coverage Auto Dumper	disabled

图4-10 phpinfo页面中的XCache扩展信息

成功下载该文件后，解压缩该文件包并将文件php_xcache.dll复制到ext目录中。打开php.ini文件，将代码清单4-8中的文本添加进去，以便PHP将文件php_xcache.dll作为一个线程安全的扩展加载。

代码清单4-8 Windows的XCache设置

```
[XCache]
zend_extension_ts=php_xcache.dll
```

重新启动Web服务器，在你的浏览器中加载phpinfo脚本，以确保该扩展能够正确加载。如果该扩展已正确安装，应该可以看到如图4-10所示的XCache扩展设置。

4.3.3 用 XCache 缓存

在任何PHP应用程序中应用XCache就像应用APC一样简单。要创建并存储Opcode并不要求使用任何函数，只要一个简单的请求就可以创建并存储Opcode了。为了确定XCache Opcode缓存的有效性，我们将用到代码清单4-4中所示的代码、代码清单4-5中所示的ab命令，以及图4-5和图4-6分别所示的禁用APC结果和启用APC结果。

图4-11显示了执行ab命令后所产生的结果。

```
Concurrency Level:      5
Time taken for tests:    3.852 seconds
Complete requests:      1000
Failed requests:         0
Write errors:           0
Total transferred:      89081000 bytes
HTML transferred:       88890000 bytes
Requests per second:    259.59 [#/sec] (mean)
Time per request:       19.261 [ms] (mean)
Time per request:       3.852 [ms] (mean, across all concurrent requests)
Transfer rate:          22582.66 [Kbytes/sec] received

Connection Times (ms)
      min  mean[+/-sd] median   max
Connect:    0     0   0.3      0     9
Processing:  0    19   6.7     17    53
Waiting:    0     7   3.6      6    30
Total:      8    19   6.7     17    54
```

图4-11 使用XCache时，代码清单4-4所示代码的ab结果

对图4-5所示的没有进行Opcode缓存的应用程序，我们的新结果可以在每秒内满足整整10个请求。我们也能看到，其响应速度比平均水平要快得多。

4.3.4 XCache 设置

XCache还包含了一组很棒的配置选项，我们可以用它们自定义XCache。表4-3列出了这些设置的完整列表。理解本章中提及的每个Opcode缓存工具的每一个设置是非常重要的，因为一些设置既可以加快运行速度，也有可能使速度降低。

表4-3 XCache配置的设置

设 置	说 明
xcache.admin.user	(String) 管理认证用户名。默认设置为“mOo”
xcache.admin.pass	(String) 管理认证密码。默认设置为“<empty string>”。此值应该是md5(你的密码)
xcache.admin.enable_auth	(String) 启用或禁用管理站点的身份验证。默认设置为“on”
xcache.test	(String) 启用或禁用测试功能
xcache.coredump_dir	(String) 在遇到故障时，放置核心转储的目录。必须是PHP可写入的目录。保留为空代表禁用
xcache.cacher	(Boolean) 启用或禁用Opcode缓存。默认为启用
xcache.size	(int) 所用共享缓存的大小。如果为0，缓存将无法使用
xcache.count	(int) 缓存被分割的“块”数。默认设置为1
xcache.slots	哈希表提示。数字越大，哈希表内进行的搜索速度就越快。此值越高，所需内存也越多
xcache.ttl	(int) Opcode文件的生存时间。如果将此值设置为0，则将无限期缓存
xcache.gc_interval	(秒) 触发垃圾回收的时间间隔。默认设置为0
xcache.var_size	(int) 变量大小
xcache.var_count	(int) 变量个数
xcache.var_slots	可变数据槽设置
xcache.var_ttl	(秒) 可变数据的生存时间。默认设置为0
xcache.var_maxttl	(秒) 处理变量时最大的生存时间
xcache.var_gc_interval	(秒) 垃圾回收的生存时间
xcache.readonly_protection	(Boolean) 启用ReadonlyProtection时可用。注意，这会减慢工具运行速度，但更安全
xcache.mmap_path	(String) 用于只读保护的文件路径。这将限制两个PHP组共享同一个/tmp/cache目录
xcache.optimizer	(Boolean) 启用或禁用优化。默认状态为禁用
xcache.coverager	(Boolean) 启用覆盖范围数据集合。启用后会减慢运行过程
xcache.coveragedump_directory	(String) 放置数据集合信息的目录位置。默认使用目录/tmp/pcovis

4.3.5 eAccelerator

我们要研究的最后一个Opcode缓存工具是eAccelerator(eA)，它的工作方式与APC和XCache非常相似。eA由Dmitry Stogov创建，最初是Turck MMcache项目的组成部分。与APC和XCache一样，eA也在共享内存中存储缓存的内容，但它同时还提供了一个单独的选项，可以将缓存数据存储在磁盘上。

在撰写本书时，这个工具最稳定的版本是0.9.6.1，我们将在本章后面的部分使用这个版本演示安装过程并衡量使用PHP时性能的提高。eA 0.9.6.1适用于PHP 4以及PHP 5的所有版本，并且可以安装在基于Windows和Unix的系统上。完整版文档以及源文件可以直接从官方网站上下载，网址是www.eaccelerator.net。

在研究如何使用eA提高PHP性能之前，我们要在Unix和Windows系统上安装这个工具。如果你使用的是Unix系统，请继续阅读，否则可以跳到“在Windows系统上安装”这一节。

1. 在Unix系统上安装

要在Unix系统上安装eA，可以在shell窗口中执行表4-4所列的命令之一，也可以从官方网站上下载并安装源代码。在本节中，我将采取后一种方法，但你也可以使用分发版命令进行安装，两种方法并没有什么区别。

表4-4 可用于安装eA的分发版命令

发布系统	命令
Red Hat/Fedora	<code>yum install php-eaccelerator</code>
Ubuntu	<code>sudo apt-get install php5-eaccelerator</code>

打开一个shell窗口，然后运行代码清单4-9中所示的命令。此命令将从eAccelerator网站上下载源代码并用tar解压缩bz2文件。

代码清单4-9 eA下载和解压缩命令

```
wget http://bart.eaccelerator.net/source/0.9.6.1/eaccelerator-0.9.6.1.tar.bz2
tar xvjf eaccelerator-0.9.6.1.tar.bz2
```

解压缩源代码后，将所得内容放在任意位置，然后在包含源代码的目录中执行代码清单4-10中所示的命令。

代码清单4-10 eA安装命令

```
phpize
./configure
make
sudo make install
```

命令执行完毕后，你应该可以在输出结果中看到两个目录，如图4-12所示。其中一个路径含有安装的库的位置，第二个路径是指向共享位置的路径。如果你要以Zend扩展形式安装，那么在后面的步骤中，将需要用到这两个目录的位置。

```

-----
Libraries have been installed in:
    /home/armandop/Downloads/eaccelerator-0.9.6.1/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 'LD_LIBRARY_PATH' environment variable
  during execution
- add LIBDIR to the 'LD_RUN_PATH' environment variable
  during linking
- use the '-Wl,-rpath -Wl,LIBDIR' linker flag
- have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
-----
Installing shared extensions:      /usr/lib/php5/20090626+libs/

```

图4-12 eA安装的输出

● 创建缓存文件夹

如前所述，有两种存储缓存内容的方法。我们可以在共享内存中存储缓存，也可以使用磁盘，将缓存保存在本地存储目录中。我们可以将eaccelerator.keys的值设置为shm_and_disk、shm_only或disk_only，以指定所用选项。如果使用默认值shm_and_disk，eA会首先尝试在共享内存中存储缓存内容，但如果共享内存中没有空间，则eA会将缓存内容存入磁盘。如果使用其他两个选项shm_only和disk_only，eA将只能使用指定的位置进行存储。

我们将使用默认值，所以需要有一个位置来放置缓存Opcode。默认情况下，eA会尝试将缓存内容存储在目录/tmp/eaccelerator中。我们建议你更改此位置并将其从/tmp目录中删除，因为该目录会在每次系统重新启动时被清空。eA推荐在/var/中创建目录，完整的目录位置可以是/var/cache/eaccelerator。

要创建目录位置，可以运行命令mkdir -p /var/cache/eaccelerator，该命令会创建一个完整的目录路径，然后使用命令chmod 0777 /var/cache/eaccelerator改变该目录的安全权限。

eA安装完毕并创建好缓存目录后，就可以将eA集成到PHP中了。

● 将eAccelerator作为PHP扩展安装

有几种方法可以用来安装eA。我们可以将eA作为一个PHP扩展进行安装，也可以将其作为zend_extension或zend_extension_ts（线程安全）安装。在本书中，我们将eA作为PHP扩展进行安装。

为了使PHP能够使用eAccelerator，必须更新php.ini文件。找到你正在使用的php.ini文件，然后将代码清单4-11中所示文本添加到文件末尾。

代码清单4-11 eA的php.ini设置

```

extension="eaccelerator.so"
eaccelerator.shm_size="16"
eaccelerator.cache_dir="/var/cache/eaccelerator"

```

```

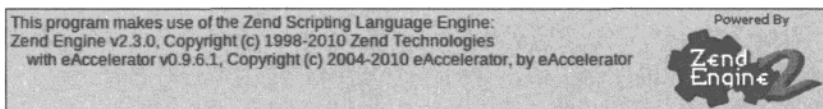
eaccelerator.enable="1"
eaccelerator.optimizer="1"
eaccelerator.check_mtime="1"
eaccelerator.debug="0"
eaccelerator.filter=""
eaccelerator.shm_max="0"
eaccelerator.shm_ttl="0"
eaccelerator.shm_prune_period="0"
eaccelerator.shm_only="0"
eaccelerator.compress="1"
eaccelerator.compress_level="9"

```

代码清单4-11包含在运行eA前需要定义的一系列设置。在配置其他13个eA设置时，一起加载.so文件是最理想的设置之一，但你可以设置的并不仅是这些。表4-5列出了这些设置及其说明的完整列表。我们将在本节后面更详细地研究这些设置。

● 确保已成功安装eA

要确保已成功安装eA，需要执行两个步骤。第一步是使用phpinfo脚本来确认PHP扩展是否已成功安装。只要创建一个phpinfo脚本，保存它，然后在浏览器中加载，你应该可以看到页面上显示的eA信息，如图4-13所示。



eAccelerator

eAccelerator support	enabled
Version	0.9.6.1
Caching Enabled	true
Optimizer Enabled	true
Check mtime Enabled	true
Memory Size	16,777,176 Bytes
Memory Available	16,772,488 Bytes
Memory Allocated	4,688 Bytes

图4-13 eA的phpinfo()设置

第二步是确保已成功创建缓存目录结构。我们需要测试缓存目录的权限级别是否设置正确。如果你一直按照我的说明操作，请打开目录/var/cache/eaccelerator，或者打开你在php.ini文件的eaccelerator.cache_dir设置中指定的缓存目录。如果所有的设置均正确，应该可以看到以0、1、2、3、…、9命名的一系列目录。eA创建的文件在保存到磁盘时，会存储在这里显示的目录中。让我们继续，使用eA进行Opcode缓存。

下一节将介绍如何在Windows系统中安装eA。如果要直接开始学习基准测试并了解使用eA的好处，你可以跳过下一节。

2. 在Windows系统上安装

eAccelerator的官方网站上含有实时更新的网站列表,为可能在Windows上运行的任意版本的PHP提供了许多eA二进制文件的编译版本。要在Windows上安装eA,你需要下载适用于你所安装的PHP版本的二进制文件,可以访问以下网址获取: <http://eaccelerator.net/wiki/InstallFromBinary>。

单击其中一个链接,下载你的PHP版本所需的正确.dll文件。文件下载完毕后,将其放置在PHP扩展文件夹中。你不一定非得将该文件放置在此位置中,但是把所有扩展保存在同一个位置中是一个好习惯。如果你通过php.net网站的安装向导将PHP安装在了默认位置,那么扩展文件夹的位置应该是C:\Program Files\PHP\ext。如果你自定义了目录,请将文件放置在其中。

● 创建eA目录

eA为我们提供了两种存储缓存内容的方法:将其存储在共享内存中,或在Web服务器的指定目录中存储缓存内容。如果使用默认的方法,eA将首先尝试将数据存储在共享内存中。但是,如果共享内存没有剩余空间,则eA会将内容存储在我们即将创建的目录中。

在C:\Program Files\Apache Software Foundation\Apache2.2中创建目录cache\eaccelerator。这样,其他应用程序也可以使用这个缓存目录,并且该目录不允许Web访问,因而可以保证其安全性。已成功安装该目录后,你需要修改php.ini文件。

● 更新php.ini

打开你的Web服务器正在使用的php.ini文件,将代码清单4-12中所示的eA设置添加到末尾,保存更改,然后重新启动Web服务器。代码清单4-12中所示的设置允许PHP使用zend_extension_ts键加载.dll文件,还可以通过设定其他几个设置来运行eA。这里列出的设置仅供参考,你可以用你认为有用的设置进行替换。我们将在本节后面详细描述代码清单4-12中列出的每个设置。表4-5列出了可以使用的eA设置完整列表。

代码清单4-12 Windows系统的php.ini eA设置

```
[PHP_eA]
zend_extension_ts="C:\Program Files\PHP\ext\eAccelerator0961_5.3.3_ts.dll"
eaccelerator.shm_size="16"
eaccelerator.cache_dir="C:\Program Files\Apache Software Foundation\Apache2.2\cache"
eaccelerator.enable="1"
eaccelerator.optimizer="1"
eaccelerator.check_mtime="1"
eaccelerator.debug="0"
eaccelerator.filter=""
eaccelerator.shm_max="0"
eaccelerator.shm_ttl="0"
eaccelerator.shm_prune_period="0"
eaccelerator.shm_only="0"
eaccelerator.compress="1"
eaccelerator.compress_level="9"
```

如果在用PHP重新启动Web服务器时,所有的设置均已正确配置并且.dll文件也已放置在服务器中,那么在创建并加载phpinfo文件时,应该可以看到类似于图4-13所示的输出结果。

eA设置正确后,让我们来看看我们可用的设置,并对初始设置进行一些修改。

4.3.6 eA 设置

与其他Opcode缓存软件一样，eA中提供了许多可用于控制该软件的设置。我们可以在eA中设置可用共享内存的大小以及存储在内存中的文件的过期时间，还可以使用过滤设置来实现只缓存某些特定文件，以上仅列举了几个例子。在安装eA时，我们已经见过了其中的一些设置，对于同样的设置列表，我们将重点讨论代码清单4-7和代码清单4-8中列出的一些关键设置。此外，我们还将更新我们的安装程序，以提高性能。

我们要研究的第一个设置是`eaccelerator.check_mtime`，eA使用该设置检查缓存内容的修改次数以及其对应的非缓存内容的修改次数。如果两个修改次数不相同，eA会尝试缓存已更新的文件。启用此设置后，每当有对文件的请求时，即使文件没有更新，eA亦将强制执行此检查。显然，这种对修改次数的比较增加了请求的开销，所以我们应该禁用此设置。而禁用这个设置的缺点是在每次对PHP脚本进行更新后，你都需要重新启动eA。

我们要看的下一个设置是`eaccelerator.filter`。这个设置可以减少eA应该缓存的文件类型。例如，如果你只想缓存`.phtml`文件，可以将此键值设置为`*.phtml`。此外，也可以在此键值前放置一个“`!`”，代表该值的否定值，如`!*.phtml *.php`。此值表示eA将仅缓存扩展名为`.php`的文件，而不会缓存扩展名为`.phtml`的文件。

通过以下三个设置，我们可以控制缓存大小并知道在没有可用空间时会如何处理缓存内容：`eaccelerator.shm_max`、`eaccelerator.shm_ttl`和`eaccelerator.shm_prune_period`。`eaccelerator.shm_max`设置的是eA可用于缓存的可用空间大小。此设置以兆字节为单位，当超出设定值后，将会用到以下两个设置：`eaccelerator.shm_ttl`和`eaccelerator.shm_prune_period`。

`eaccelerator.shm_ttl` 和 `eaccelerator.shm_prune_period` 用于确定在达到`shm_max`设定值后，应移除哪些内容。第一个设置`eaccelerator.shm_ttl`包含的是最后的访问时间，以秒为单位。如果某个文件在经过`eaccelerator.shm_ttl`秒后未被请求，则在没有可用空间时，其内容将从缓存中删除。而`eaccelerator.shm_prune_period`则是内容应该保存在缓存中的时长。

你还可以在`php.ini`文件中使用表4-5中列出的其他设置。表4-5含有设置的名称、何时使用该设置的简短描述及其默认设置。我建议你仔细阅读这张表格，然后再决定应该启用或禁用哪些功能。

表4-5 eA设置

设 置	说 明	默 认 值
<code>eaccelerator.shm_size</code>	设置共享内存的大小。以兆字节为单位	0兆字节
<code>eaccelerator.cache_dir</code>	设置缓存目录的位置。在开启磁盘缓存后，eA将在此目录中存储预编译的代码、会话数据以及内容	<code>/tmp/eaccelerator</code>
<code>eaccelerator.enable</code>	启用或禁用eA。设置为0，禁用eA。设置为1，启用eA	启用：1

(续)

设 置	说 明	默 认 值
eaccelerator.optimizer	启用后,可以加快代码执行速度。在编译脚本时将运行此优化程序。设置为1,启用优化。设置为0,禁用优化	启用: 1
eaccelerator.debug	启用 eA 日志。日志消息将被放在 eaccelerator.log_file 设置指定的文件中。设置为1,启用日志。设置为0,禁用日志	禁用: 0
eaccelerator.log_file	指定 eA 用于存放日志消息的文件位置。若未指定日志文件,所有日志条目将被放置于 stderr 中;如果使用了 Apache Web 服务器,日志条目会放置在 Apache 的日志文件中	没有默认值
eaccelerator.name_space	附加到 eA 所生成的键值开头的字符串	Web 服务器的主机名
eaccelerator.check_mtime	每次请求脚本时,允许 eA 检查该文件自上次缓存后是否被修改	启用: 1
eaccelerator.filter	指定需要缓存或不需要缓存的文件。例如: “*.php” 将缓存所有以 .php 扩展名结尾的文件。要排除某种模式,请在该模式前加上 “!”	“”, 缓存所有的 php 文件
eaccelerator.shm_max	设置可以放置在缓存中的最大容量。任何大于此处指定大小的文件将不会被缓存。此值以字节为单位。设置为0将禁用最大容量,允许缓存任意大小的文件	0, 禁用最大容量。
eaccelerator.shm_ttl	当到达共享内存的最大容量时,此设置指定的时间内未被访问过的文件将会被移除。此值应该是整数,以秒为单位。设置为0,则表示没有过期时间	禁用: 0
eaccelerator.shm_prune_period	当到达共享内存的最大容量时,shm_prune_period 秒前创建的文件将被移除。设置为0,表示所有缓存均不会被移除	禁用: 0
eaccelerator.shm_only	启用或禁用磁盘上保存缓存脚本。设置为0,表示允许磁盘缓存	启用: 0
eaccelerator.compress	启用或禁用压缩功能。设置为1,表示启用压缩功能。设置为0,则禁用压缩功能	启用: 1
eaccelerator.compress_level	设置缓存的压缩级别。设置为9,表示使用最大压缩	9: 最大压缩
eaccelerator.keys	为此类缓存设置缓存类型。可用的值包括 shm_and_disk (在共享内存和磁盘中存储)、shm (如果共享内存可用,则在其中存储,否则在磁盘上存储)、shm_only (仅在共享内存中存储)、disk_only (仅使用磁盘缓存)、none (无缓存)	shm_and_disk
eaccelerator.sessions	为此类缓存设置缓存类型。可用的值包括 shm_and_disk (在共享内存和磁盘中存储)、shm (如果共享内存可用,则在其中存储,否则在磁盘上存储)、shm_only (仅在共享内存中存储)、disk_only (仅使用磁盘缓存)、none (无缓存)	Shm_and_disk

(续)

设 置	说 明	默 认 值
eaccelerator.content	为此类缓存设置缓存类型。可用的值包括 shm_and_disk (在共享内存和磁盘存储)、shm (如果共享内存可用, 则在其中存储, 否则在磁盘上存储)、shm_only (仅在共享内存中存储)、disk_only (仅使用磁盘缓存)、none (无缓存)	Shm_and_disk
eaccelerator.admin.name	eA管理工具的用户名	N/A
eaccelerator.admin.password	eA管理工具的密码	N/A

4.4 小结

本章讲述了PHP的生命周期, 并概述了Opcode是如何生成的。你还学到了在对某个文件进行后续请求时, Opcode缓存可以省掉PHP生命周期中的三个步骤, 从而提高代码性能。在第3章打下的基础上, 我们在本章中学习了可用于缓存Opcode的各种工具。我们安装了以下列出的缓存工具、对缓存Opcode的好处进行了基准测试并检查了每个设置, 以确定哪些设置可以进行修改, 从而提高性能缓存工具如下所示:

- ☐ APC
- ☐ XCache
- ☐ eAccelerator

特别地, 我们讨论了对APC和eA禁用文件修改检查, 因为该功能会在创建Opcode时增加额外的步骤, 我们还禁用了在APC内部使用的优化级别。最后, 我们了解并安装了APC管理工具, 我们可以使用这个工具控制和查看内存消耗和已缓存的PHP文件, 也可以方便地清除Opcode缓存, 所有功能都可以在一个网页界面上完成。

在下一章中, 我们将学习缓存内容、清除诸如连接数据库之类的繁杂流程以及为每个使用请求获取内容。

功能丰富的现代Web应用程序都会用某种方法让用户与应用程序进行交互。如今，最流行的交互方法是在常用的社交网络Web应用程序上更新自己的状态。在几乎所有情况下，这些用户交互都会保存在稳定的外部存储解决方案中，并且这些内容会在列表页面中重新显示给用户。

在这样的请求中，可以改善的地方不胜枚举，比如数据库软件（第9章将会讲到）、连接数据库的方式以及处理此类进程结果的方式。本章将重点关注如何优化内容读取操作以及如何通过缓存来重复利用数据查询结果、复杂算法结果或任意数据。

我们将研究两种不同的缓存软件：APC和Memcached。你将学到如何安装Memcached（我们已经在第4章安装了APC），使用代码示例应用变量缓存，以及对其给应用程序（我们即将创建的）带来何种影响进行基准测试。此应用程序将使用MySQL数据库来存储1万条记录（这属于中等大小的记录集）并在一个页面上显示这些记录。我们还将从更高层的角度研究Web应用程序请求中的缓存，以进一步理解缓存在整个请求中的作用。

5.1 应用程序的性能路线图

我们将不再讨论缓存Opcode的PHP生命周期以及PHP脚本的逻辑和代码，而关注用户所请求的缓存数据。这样可以更深入地研究应用程序栈，从而发现可以完全或部分消除的瓶颈问题。在如图5-1所示的应用程序栈中，本章将重点研究变量缓存，也就是灰色框中的内容。

如前所述，变量缓存可以缓存任意数据，而这并不包括PHP逻辑或源代码，它们的缓存将通过Opcode缓存来实现。变量缓存能缓存方法中的PHP逻辑及数据库查询调用的结果，一般来讲，它可以缓存进程中的任意数据结果。



图5-1 应用程序栈

5.2 实现变量缓存的价值

通过使用一个一般的Web应用程序，可以从高层视角来理解变量缓存的重要性。如今大多数的Web应用程序均会使用某种类型的稳定存储解决方案——数据库(包括云存储)和/或平面文件。因此，在从这些外部资源获取数据时，应用程序的运行速度会有所降低(请参阅第3.1.1节)。要理解其原因，我们需要参考图5-2。

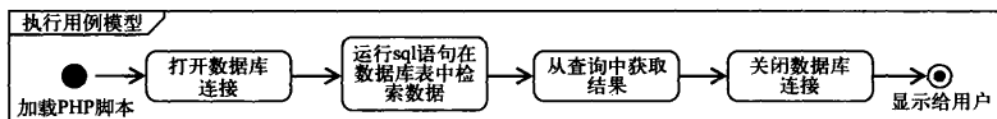


图5-2 没有缓存的情况下从数据库中获取数据

图5-2是PHP脚本从数据库中读取数据的典型应用程序流。此图的流程包括打开数据库连接、运行任意SQL语句、读取SQL语句找到的结果、关闭数据库连接，最后在HTML页面上将所获内容显示给用户。在这个例子中，上述每个步骤都存在瓶颈。数据库软件可能未调整为最佳的运行设置，SQL语句使用的数据表可能未被优化，数据库驱动程序可能不是最佳选择——这其中有太多需要注意的地方。如果不使用缓存，用户在每次请求PHP脚本时都会碰到所有的瓶颈问题，每一次都会导致性能的降低。如果使用缓存来存储SQL语句的结果，这种性能损失就不复存在了。

通过使用缓存并在应用程序流中采取几个额外步骤能够提高PHP的性能。如图5-3所示，之前的简单应用程序流现在包含了每个请求均必须执行的额外步骤。其中的两个步骤是初始请求和后续请求均必须实现的。在初始请求中，应用程序将检查缓存，以确定是否已存储了数据库查询的数据。因为这是初始请求，所以缓存中没有可取数据，这将触发“缓存未命中”流程。未发现缓存数据即属于“缓存未命中”的情况。由于“缓存未命中”，所以需要开启数据库连接、执行SQL语句、获取查询结果等常规步骤。接下来这个流程引入了第二个新步骤。数据从结果集中提取后，会被放入缓存，以待后续的用户请求使用。

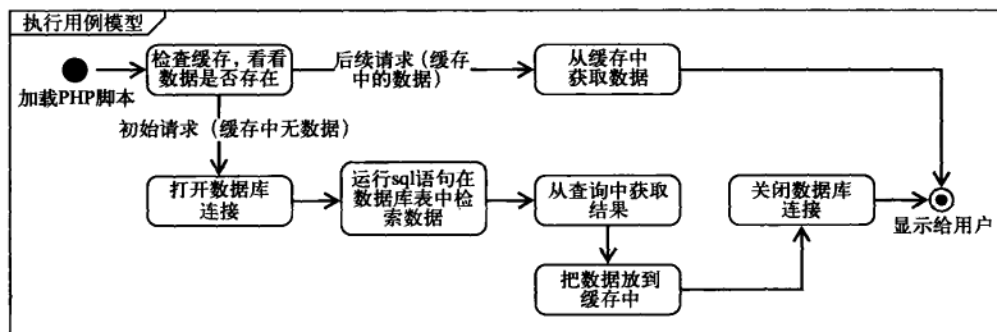


图5-3 从缓存中读取数据

在分析用户对PHP脚本的后续请求时可以看出使用缓存的好处。再次参考图5-3，我们继续查看后续用户请求的应用程序流。当用户加载PHP脚本时，初始步骤是在缓存中检查数据，该操作触发了“缓存命中”流程；缓存数据被找到且仍然有效。触发缓存命中流程后，可以跳过许多步骤，包括开启数据库连接、执行SQL语句、从结果集中获取数据、将数据添加到缓存、关闭数据库连接。我们将从共享内存中读取数据，用这种方式获取内容更为快速，效果更佳，因为我们不需要处理外部驱动器、网络问题和软件等事项。

这种方法并不仅限于从数据库中读取数据。如前所述，我们可以对平面文件中存储的数据以及过程密集型函数/方法的结果进行缓存。我们将在下面几个小节中创建一个使用数据库的小型应用程序，以此来实践如图5-3所示的应用程序流。

5.3 示例项目：创建表

要达到使用缓存的最佳效果，我们将创建一个由数据库驱动的小型Web应用程序。该应用程序将在一个HTML表格中显示数据库中存储的1万条记录。通过创建这个非常简单的应用程序，我们可以很容易看出应用缓存解决方案将如何提高程序性能。我们将依次对应用程序在没有数据库层、有数据库层以及使用缓存时的性能进行基准测试。首先，创建和准备数据库及数据表。

安装数据库并不是本书的重点内容，因此这里并不会详细讨论这个话题。但附录A中附带了如何安装MySQL的说明。建议你现在就安装数据库，因为第8章内容也需要数据库。可以在www.mysql.com上免费下载安装程序。这个例子中测试使用的是MySQL 5.1。

使用代码清单5-1中的SQL语句创建数据库以及一个数据表chapter5，表中含有一个整数列num。

代码清单5-1 创建数据库和表的SQL语句

```
CREATE DATABASE pro_php_perf;
USE pro_php_perf;
CREATE TABLE chapter5 (num int(8) NOT NULL);
```

成功创建数据库和表后，我们需要在表中放入1万条记录。复制并执行代码清单5-2中的代码。

代码清单5-2 在表中插入1万个数字型值的PHP代码

```
<?php
/**
 * Insert 10,000 random numerical values into table.
 *
 */
$username = 'YOUR_USERNAME_HERE';
$password = 'YOUR_PASSWORD_HERE';
$host     = 'YOUR_DB_HOST_HERE';
$dbName   = 'YOUR_DATABASE_NAME_HERE';

//Open connection.
$dbHandler = mysqli_connect($host, $username, $password, $dbName)
```

```

        or die('Connection error: '.mysqli_error());

//Connect to db.
mysqli_select_db($dbHandler, $dbName)
    or die ('Could not connect to db: '.mysqli_error());

//Add in 10000 records into db.
$i=0;
while($i<10000)
{
    //Generate random number
    $num = rand(1,1000000);

    //Insert into table
    $statement = "INSERT INTO chapter5 (num) VALUES ($num)";
    mysqli_query($dbHandler, $statement)
        or die ('Error executing statement: '.mysqli_error());

    ++$i;
}

//Close connection.
mysqli_close($dbHandler);

```

代码清单5-2所示的代码设置了连接信息，并使用PHP函数mysqli开启一个数据库连接，使用INSERT语句插入1万个随机的数字型值，最后关闭连接。执行代码后，运行代码清单5-3所示的SQL语句以确定数据已成功被插入表格中。

代码清单5-3 计算表中记录数的SQL语句

```
SELECT COUNT(num) FROM chapter5;
```

如果结果正常，继续下一步。如果遇到任何问题，确保已根据数据库的用户设置更新了连接信息并检查是否存在SQL错误。

5.3.1 获取记录

在数据表中存放了数据后，可以创建代码从表中读取所有数据，没错，是全部1万条数据，如代码清单5-4所示。首先，对不使用数据库层、仅循环访问含有1万个元素的数组的PHP代码进行性能测试。这有助于测量使用数据库时产生的开销。

代码清单5-4 没有数据库开销的PHP代码

```

<?php
$records = array_fill(0, 10000, 50000);

$stable = "<table border='1'><tr><td>Array Elements</td></tr>";

//Display the data.
foreach($records as $record)
{

```

```

$stable .= "<tr><td>$record</td></tr>";

}

$stable .= "</table>";

echo $stable;

```

代码清单5-4中包含了PHP代码以及用来显示记录的HTML标记，但并未包含任何数据库连接，也没有任何取数据的开销。取而代之的是，代码在数组\$records中创建了1万个元素，然后通过foreach循环来循环读取数组中的元素并在HTML表格中显示数据。

初始基准测试将分析代码清单5-4中所示的代码，这将作为衡量未来结果的基准。使用代码清单5-5中所示的ab命令，模拟1000个请求的流量负载，每5个请求并发。

代码清单5-5 ab命令

```
ab -n 1000 -c 5 http://localhost/Listing5_4.php
```

执行该命令后，应该可以看到类似于图5-4所示的ab结果。

```

Concurrency Level:      5
Time taken for tests:    3.759 seconds
Complete requests:      1000
Failed requests:         0
Write errors:            0
Total transferred:      230265384 bytes
HTML transferred:       230074193 bytes
Requests per second:    266.06 [#/sec] (mean)
Time per request:       18.793 [ms] (mean)
Time per request:       3.759 [ms] (mean, across all concurrent requests)
Transfer rate:          59829.11 [Kbytes/sec] received

Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:        0        0   0.1      0      2
Processing:      8       19   6.5     17     43
Waiting:        7       16   5.7     14     42
Total:          8       19   6.5     17     43

```

图5-4 代码清单5-2所示代码的ab结果

结果表明，没有数据库层时，Web服务器的响应速度可达266.06个请求/秒，每个请求可在18.79毫秒内响应。结果还表明，并发请求的响应时间是3.75毫秒。现在，添加数据库层并进行基准测试。

5.3.2 计算读取数据库的开销

现在，对代码清单5-6中所示的PHP脚本进行基准测试。所示代码将使用数据库层来连接数据库，并从之前创建的表chapter5中读取所有记录。但是，这段代码将不会根据结果集显示或创建HTML表格。相反，我们将继续使用数组\$records。这样，便能对使用数据库的开销进行基

准测试了。

按照代码清单5-6更新代码。

代码清单5-6 从表中读取1万条记录的PHP脚本

```
<?php
/**
 * Benchmark Database overhead
 *
 */
$username = 'YOUR_USERNAME_HERE';
$password = 'YOUR_PASSWORD_HERE';
$host      = 'YOUR_DB_HOST_HERE';
$dbName    = 'YOUR_DATABASE_NAME_HERE';

//Open connection.
$dbHandler = mysql_connect($host, $username, $password, $dbName)
    or die('Connection error: '.mysql_error($dbHandler));

//Connect to db.
mysql_select_db($dbName, $dbHandler)
    or die('Could not connect to db: '.mysql_error($dbHandler));

//Fetch the records from the Database.
$stmt = "SELECT num FROM chapter5 ORDER BY num DESC";
$results = mysql_query($stmt, $dbHandler)
    or die('Could not run SQL: '.mysql_error($dbHandler));

//Close connection.
mysql_close($dbHandler);

$records = array_fill(0, 10000, 50000);

$table = "<table border='1'><tr><td>Array Elements</td></tr>";

//Display the data.
foreach($records as $record)
{
    $table .= "<tr><td>$record</td></tr>";
}

$table .= "</table>";

echo $table;
```

代码清单5-6中的代码首先设置了连接信息。如果你一直跟随着本书操作，请将这些设置中的数值更改为你个人的数据库设置。此段代码会打开数据库连接、创建以降序顺序读取所有数值的SQL语句、关闭数据库连接、在\$records数组中填入1万个元素，最后调用foreach循环，在HTML表格中显示所有结果。对更新的代码运行代码清单5-5中的ab命令，将产生如图5-5所示的结果。


```

Concurrency Level:      5
Time taken for tests:    4.786 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      230249000 bytes
HTML transferred:       230058000 bytes
Requests per second:    208.96 [#/sec] (mean)
Time per request:       23.928 [ms] (mean)
Time per request:       4.786 [ms] (mean, across all concurrent requests)
Transfer rate:          46985.91 [Kbytes/sec] received

```

```

Connection Times (ms)
              min  mean[+/-sd] median  max
Connect:        0    0   0.1      0    1
Processing:    10   24   8.9     22   60
Waiting:        9   21   7.9     19   59
Total:         10   24   9.0     22   60

```

图5-5 代码清单5-4所示代码的ab结果

新结果表明，性能有了显著下降。Web服务器每秒仅可支持208.96个请求，每个请求的响应时间为23.92毫秒。对比图5-4所示的结果可知，在使用数据库连接和记录读取后，每秒处理的请求数减少了58个，也就是下降了21.46%，并且响应时间也增加了27.32%。这表明了从外部数据源（这里是数据库）获取数据的相关成本。

在继续之前，让我们更新代码，充分利用结果集，并使用其内容在HTML表中显示数据。完整代码如代码清单5-7所示。

代码清单5-7 从数据库中读取并显示全部1万条记录

```

<?php
/**
 * No Database overhead.
 */
$usename = 'root';
$password = 'password';
$host = 'localhost';
$dbName = 'pro_php_perf';

//Open connection.
$dbHandler = mysql_connect($host, $usename, $password, $dbName)
    or die('Connection error: '.mysql_error($dbHandler));

//Connect to db.
mysql_select_db($dbName, $dbHandler)
    or die('Could not connect to db: '.mysql_error($dbHandler));

//Fetch the records from the Database.
$stmt = "SELECT num FROM chapter5 ORDER BY num DESC";
$results = mysql_query($stmt, $dbHandler)
    or die('Could not run SQL: '.mysql_error($dbHandler));

```

```
//Close connection.
mysql_close($dbHandler);

//Add to collection
$records = array();
while($record = mysql_fetch_object($results))
{

    $records[] = $record->num;

}

//Display
$stable = "<table border='1'><tr><td>Array Elements</td></tr>";

foreach($records as $record)
{

    $stable .= "<tr><td>$record</td></tr>";

}

$stable .= "</table>";

echo $stable;
```

使用代码清单5-5中所示的ab命令对整个代码进行基准测试，结果如图5-6所示。

```
Document Path:      /Chapter5/Listing_5_7.php
Document Length:    238932 bytes

Concurrency Level:   5
Time taken for tests: 13.614 seconds
Complete requests:   1000
Failed requests:     0
Write errors:        0
Total transferred:   239123000 bytes
HTML transferred:    238932000 bytes
Requests per second: 73.46 [#/sec] (mean)
Time per request:    68.067 [ms] (mean)
Time per request:    13.614 [ms] (mean, across all concurrent requests)
Transfer rate:       17153.45 [Kbytes/sec] received

Connection Times (ms)
      min  mean[+/-sd] median   max
Connect:    0      0   0.2      0      4
Processing: 26    68  23.2     66    136
Waiting:    24    62  21.5     59    123
Total:      26    68  23.2     66    136
```

图5-6 代码清单5-7所示代码的ab结果

将完整逻辑添加到我们将使用的应用程序示例中，每秒处理的请求总数降到了73.46，每个请求的响应时间为68.067毫秒。这组数字并不理想：每秒可处理的请求总数降低了65%，每个请求的响应时间也增加了184%。让我们继续，使用APC来改善这种结果。

5.4 APC 缓存

在第4章中，我们采用了APC专门来缓存PHP Opcode。在这一章中，我们将使用APC来存储信息。为此，我们将重点关注PHP APC内部方法的子集，如表5-1所示。在开始之前，请确保已完整安装APC PHP扩展。第4章中介绍了安装APC的全部步骤。

表5-1 APC函数

函 数	参 数	说 明
apc_add()	apc_add(String key, Mixed var, int Expiration Time)	如果指定键不存在，则使用其将内容添加到缓存中
apc_fetch()	apc_fetch(Mixed key)	在缓存中获取某个键的内容。如果未找到该键，则返回false
apc_store()	apc_store(String key, Mixed var, int Expiration Time)	使用指定键将某个数值存储在缓存中。如果该键存在，则将替换原有数值
apc_exists()	apc_exists(mixed keys)	检查指定键是否存在于缓存中。如果该键存在，返回true，否则返回false
apc_delete()	apc_delete(String key)	从缓存中移除指定键。如果操作成功，返回true，否则返回false

注意 可以在www.php.net/apc上查看其他的APC函数。

表中列出的函数可以实现的功能有：将数据添加到共享内存中、使用指定键从共享内存中读取数据、检查确定某个键是否存在、以及从缓存中移除与某个键相关的内容。

5.4.1 将数据添加到缓存中

我们将利用表5-1中列出的部分函数创建一个小型页面计数器，如代码清单5-8所示，然后，再将其应用到由数据库驱动的Web应用程序中。

代码清单5-8 页面计数器：将内容添加到APC缓存

```
<?php
/**
 * Example visitor counter using APC.
 *
 **/
if(!$counter = apc_fetch('myCounter'))
{
    $counter = 1;
    //Add the new value to memcached
    apc_add('myCounter', $counter, 120);
}
else
{
```

```

$counter++;

//Update the counter in cache
apc_store('myCounter', $counter, 120);

}

echo "Your visitor number: ".$counter;

```

代码清单5-8可以作为使用APC数据缓存的示例，这段代码含有一个有效的页面计数器，在用户初次访问此PHP脚本的两分钟内，动态记录用户数。此网页计数器首先调用`apc_fetch()`来检查键`myCounter`是否存在于缓存中，并读取其数据。如果该键值不存在，则变量`$counter`被设置为1，然后再共享内存APC缓存中插入新键`myCounter`。如果键值存在，则计数器将仅读取当前缓存中的数据，并将数值加1，然后使用`apc_store()`更新在键值`myCounter`中存储的数据。最后，将访客计数展示给用户。

这个例子还在`apc_add()`中使用了第三个参数。我们将其设定为整型数值120秒（2分钟），从而请求缓存信息在2分钟内保持有效。数据过期后，缓存键值将在下个请求中更新。

我们接下来将使用APC将缓存应用到数据库驱动的应用程序示例中。

5.4.2 对 APC 进行基准测量

我们将要把刚才学到的APC函数应用到数据库驱动的应用程序中，把这些函数应用于代码中有瓶颈的地方，并连接数据库并读取数据。如代码清单5-9所示对代码进行更新。

代码清单5-9 对代码清单5-7中代码应用APC缓存

```

<?php
/**
 * Listing 5.7 with APC applied.
 */
$username = 'YOUR_USERNAME_HERE';
$password = 'YOUR_PASSWORD_HERE';
$host      = 'YOUR_DB_HOST_HERE';
$dbName    = 'YOUR_DATABASE_NAME_HERE';

if(! $records = apc_fetch('orderedNumbers'))
{
    //Open connection.
    $dbHandler = mysql_connect($host, $username, $password, $dbName)
        or die('Connection error: '.mysql_error($dbHandler));

    //Connect to db.
    mysql_select_db($dbName, $dbHandler)
        or die ('Could not connect to db: '.mysql_error($dbHandler));
}

```

```

//Fetch the records from the Database.
$statement = "SELECT num FROM chapter5 ORDER BY num DESC";
$results   = mysql_query($statement, $dbHandler)
            or die ('Could not run SQL: '.mysql_error($dbHandler));

//Close connection.
mysql_close($dbHandler);

//Place into array.
$records = array();
while($record = mysql_fetch_object($results))
{

    $records[] = $record->num;

}

//Add to cache for 2 minutes
apc_store('orderedNumbers', $records, 120);

}

//Display
$stable = "<table border='1'><tr><td>Array Elements</td></tr>";

foreach($records as $record)
{

    $stable .= "<tr><td>$record</td></tr>";

}

$stable .= "</table>";

echo $stable;

```

注意 如果你跟随本书做了每个代码示例，请确保删除php.ini文件中的“;”以开启APC，然后重新启动Web服务器。

这段代码只进行了微调，以粗体字显示。与代码清单5-7中的代码一样，我们增加了函数apc_add()和apc_fetch()。让我们对这段代码应用ab，以确定APC缓存可以在哪些方面提高性能。图5-7显示了ab模拟的结果。

图5-7显示的是运行ab后得到的结果。将图5-6与图5-7进行比较，可以看到服务器在满足的请求数量方面有所改进，而Web服务器响应单一请求的速度也有所提高。现在，服务器可以满足的用户数量增长了72.51%，而响应单个请求的总时间也下降了42%。使用APC提高了应用程序的性能。

```

Concurrency Level:      5
Time taken for tests:    7.891 seconds
Complete requests:      1000
Failed requests:         0
Write errors:           0
Total transferred:      259047000 bytes
HTML transferred:       258856000 bytes
Requests per second:    126.73 [#/sec] (mean)
Time per request:       39.454 [ms] (mean)
Time per request:       7.891 [ms] (mean, across all concurrent requests)
Transfer rate:          32059.41 [Kbytes/sec] received

```

```

Connection Times (ms)
              min    mean[+/-sd] median    max
Connect:        0      0   0.7      0      18
Processing:    15     39  13.3     39     86
Waiting:        0     19   9.1     17     61
Total:         15     39  13.4     39     86

```

图5-7 启用APC缓存的ab结果

运行时间的缩短以及目前支持用户数的上升得益于从内存中读取数据，我们不再需要重复打开数据库连接、执行SQL语句并读取数据这个三步流程。在这个例子中，用户可以直接获取数据，并且在2分钟内无须执行以上三步。要进一步提高脚本的性能，还可以缓存所生成的HTML，从内存中读取整张表格。

APC不是唯一可用于缓存数据的工具。Memcached是另一种可用的工具。

5.5 Memcached

与本章和第4章所涉及的其他缓存工具不同，Memcached是原本用于PHP的缓存解决方案。2003年，Brad Fitzpatrick为Web应用程序Livejournal开发出了Memcached之后，Memcached得到了开源社区的鼎力支持。现在，它已广泛应用于各种主流Web应用程序中，比如维基百科、Flickr、Twitter和YouTube等。

在下面几节中，我们将讲述Memcached的安装过程、研究PHP Memcached函数并调整设置，使其达到最佳性能。

5.5.1 安装 Memcached

目前，Memcached的稳定版本是1.4.5，可从www.memcached.org上下载。安装Memcached有两种选择。可以从上述网站下载.tgz源文件，也可以使用apt-get或yum命令从发布端口中安装。

如果你的系统安装了apt-get，可以运行命令apt-get install php5-memcached。

该命令将安装Memcached所需的全部关联程序和软件包。重新启动Web服务器，创建一个PHP脚本phpinfo()，然后用Web浏览器加载该文件，以确认PHP可以成功加载Memcached扩展。如果所有程序均已正确安装，应该可以看到Memcached PHP设置，如图5-8所示。

memcached

memcached support	enabled
Version	1.0.0
libmemcached version	0.31
Session support	yes
igbinary support	no

图5-8 phpinfo页面上的Memcached信息

另一种安装Memcached的方法是使用源文件。下载源文件，然后在终端窗口中执行代码清单5-10所示的命令。

代码清单5-10 用源文件安装Memcached

```
./configure
make
make install
```

完成Memcached的安装后，重新启动Web服务器并加载phpinfo脚本以确保所有程序已正确安装。你应该可以看到如图5-8所示的Memcached设置。

5.5.2 启动 Memcached 服务器

指定分配给Memcached的内存容量及其运行的端口，以启动Memcached。Memcached默认侦听11211端口，但可以使用-p参数改变端口号。执行命令/usr/bin/memcached -m 512 -p 11211启动Memcached。

5.5.3 在 PHP 中使用 Memcached

PHP已在PHP Memcached类中捆绑了一系列的Memcached方法。我们将重点研究表5-2列出的方法子集，以便让你熟悉这个工具。你可以使用表中列出的方法向缓存中添加内容、从缓存中提取内容、更新缓存内容、清除所有内容和从缓存中删除某个指定的项。

表5-2 Memcached方法

方 法	参 数	说 明
Memcached::add()	add (String key, mixed Value, int Expiration Time in seconds)	向缓存中添加一个新键/值。如果该键已存在，则会失败
Memcached::get()	get (String key, Callback function, float cas_token)	从缓存中读取指定键的内容。如果键不存在，则返回false
memcached::set()	set (String key, Mixed value, int Expiration Time in seconds)	设置指定键的内容。与add()不同，即使指定键存在，该方法也不会失败
memcached::flush()	flush (int time)	从缓存中删除所有键和内容
memcached::delete()	delete (String key, int Time in seconds)	从缓存中移除指定键。如果设置了时间参数，则在指定时间到达前，将忽略所有向缓存中添加该键的请求

注意 Memcached可用方法的完整列表可以从www.php.net/memcached的PHP文档中获取。

1. 连接至memcached服务器

在将数据保存至新memcached服务器之前，需要打开一个指向服务器的连接，就像打开一个数据库连接或创建一个文件处理程序一样。代码清单5-11展示了一段简单的PHP 5代码示例，可以连接到在本地运行的Memcached服务器。

代码清单5-11 连接到Memcached服务器

```
<?php
/**
 * Example visitor counter using Memcached.
 *
 **/
$memHost = 'localhost';
$memPort = 11211;

$memCached = new Memcached();
$memCached->addServer($memHost, $memPort);
```

我们使用变量\$memHost来存储将要连接的服务器，并在变量\$memPort中存储memcached服务器侦听的端口。然后，实例化Memcached类，用方法addServer()将localhost服务器添加至Memcached服务器池中。

2. 将数据添加到缓存中

与前面介绍APC的章节一样，我们将利用代码清单5-8所示的访客计数器代码介绍如何使用Memcached函数。代码清单5-12列出了更新的访客计数器代码。

代码清单5-12 在页面计数器中使用memcached

```
<?php
/**
 * Example visitor counter using Memcached.
 *
 **/
$memHost = 'localhost';
$memPort = 11211;

$memCached = new Memcached();
$memCached->addServer($memHost, $memPort);

if(!$counter = $memCached->get('myCounter'))
{
    $counter = 1;
    //Add the new value to memcached
    $memCached->add('myCounter', $counter, 120);
}
else
```



```

{

    $counter++;

    //Update the counter in cache
    $memCached->set('myCounter', $counter, 120);

}

echo "Your visitor number: ".$counter;

```

其中的逻辑与代码清单5-8是一样的：使用if-else语句检查在缓存中是否存在myCounter的值。如果缓存中有这个值，则将此值加1，并将新值存储到缓存中。否则，对要存储的内容进行初始化，然后将值存入缓存内的新键中。

在使用Memcached时，首先创建Memcached对象，使用方法addServer()添加memcached服务器，使用Memcached方法get()确定共享内存中是否存在数据；如果没有数据，则使用变量\$counter初始化数据并用Memcached::add()将该数据添加到新键myCounter中，其初始值为1。

3. 对Memcached进行基准测试

我们将使用代码清单5-7所示的代码对应用Memcached时的结果进行基准测试。代码清单5-13中更新了代码。

代码清单5-13 对代码清单5-5中的代码应用Memcached

```

<?php
$username = 'YOUR_USERNAME';
$password = 'YOUR_PASSWORD';
$host      = 'YOUR_HOST';
$dbName    = 'YOUR_DB';

$memHost = 'localhost';
$memPort = 11211;

$memCached = new Memcached();
$memCached->addServer($memHost, $memPort);

if(!$records = $memCached->get('myRecords'))
{
    //Open connection.

    $dbHandler = mysql_connect($host, $username, $password, $dbName)
        or die('Connection error: '.mysql_error($dbHandler));

    //Connect to db.
    mysql_select_db($dbName, $dbHandler)
        or die('Could not connect to db: '.mysql_error($dbHandler));

    //Fetch the records from the Database.
    $statement = "SELECT num FROM chapter5 ORDER BY num DESC";
    $results   = mysql_query($statement, $dbHandler)

```

```

        or die ('Could not run SQL: '.mysql_error($dbHandler));

//Close connection.
mysql_close($dbHandler);

$records = array();
while($record = mysql_fetch_object($results))
{

    $records[] = $record->num;

}

//Add the data into Memcached
$memCached->add('myRecords', $records, 120);
}

//Display
$stable = "<table border='1'><tr><td>Array Elements</td></tr>";

foreach($records as $record)
{

    $stable .= "<tr><td>$record</td></tr>";

}

$stable .= "</table>";

echo $stable;

```

使用代码清单5-5所示的ab命令，结果表明，在使用缓存解决方案时，性能有所提高。使用缓存，应用程序可以支持更多的用户并能加快脚本的运行速度。在图5-9所示的新结果中，每秒处理的请求上升到了215.46个，比未使用缓存的应用程序增加了193%。每个请求花费的时间也从68.06毫秒减少到了23.20毫秒，速度提高了65%。

```

Concurrency Level:      5
Time taken for tests:    4.641 seconds
Complete requests:      1000
Failed requests:         0
Write errors:           0
Total transferred:      259047000 bytes
HTML transferred:       258856000 bytes
Requests per second:    215.46 [#/sec] (mean)
Time per request:       23.206 [ms] (mean)
Time per request:       4.641 [ms] (mean, across all concurrent requests)
Transfer rate:          54505.72 [Kbytes/sec] received

```

```

Connection Times (ms)
      min  mean[+/-sd] median   max
Connect:    0      0   0.2      0      6
Processing: 10     23   7.8     22     47
Waiting:    0     11   4.9      9     37
Total:      10     23   7.8     22     47

```

图5-9 代码清单5-13所示代码的ab结果

5.6 小结

本章重点研究PHP代码本身。你在阅读第2章时已经明白，优化不仅仅只关于PHP，而是涉及PHP运行的所有层。要让应用程序实现最佳性能，首先要使用工具集测量性能，然后进行客户端JavaScript和CSS优化，之后在PHP中使用缓存。第4章和第5章介绍并应用了缓存解决方案，不仅用于缓存Opcode，还包括运行应用程序所需的大量数据。我们已讲述了一些基础知识，包括什么是缓存以及缓存在PHP应用程序中的作用。

本章重点介绍了APC以及Memcached。我们学习了缓存所涉及的一些术语，使用内置的PHP函数实现了存储、读取以及从APC中清除缓存，安装并使用了Memcached，最重要的是，我们还进行了简单的实验，用于测试APC和Memcached对性能有怎样的提高。在剩余的几节中，我们将关注用于运行应用程序的软件。

因为所有入站请求都由Web服务器软件负责接收，并将所请求的任务委派给PHP引擎，最后发送响应，因此PHP应用程序的速度在很大程度上与Web服务器软件的性能是紧密相关的。所以，我们需要通过消除不必要的进程来防止Web服务器降低应用程序的速度。

本章将介绍Web服务器的工作原理，并帮助你确定哪个主要Web服务器程序包最适合你的应用程序。我们将看看一直很流行的带有mod_php的Apache Web服务器，并且研究一下最新出现的lighttpd和Nginx。我们将分析这些服务器好用的原因，以及如何在这些服务器上安装FastCGI PHP。我们还要看一看针对每种服务器类型的几个简单的基准测试，其中涉及静态内容和PHP内容。

但在深入论述之前，我们将看一看选择正确的Web服务器程序包的一些通用准则，以及所有类型共有的Web服务器的一些方面，如使用情况、请求处理以及硬件。

首先要弄清我们的位置，请参阅图6-1，该图显示我们现在处于应用程序性能路线图的Web服务器部分。



图6-1 PHP应用程序组件栈

在应用程序栈中，Web服务器这个部分非常重要，因为它的执行速度将影响对用户响应的速度。现在，我们继续讨论所有Web服务器通用的一些注意事项。

6.1 选择适合你的 Web 服务器程序包

如何确定应该使用哪个程序包没有硬性的规则。但是，我们会尽量提供几点建议帮助你选择。

6.1.1 安全性和稳定性非常重要

毫无疑问，Apache是可供选择的最好用的Web服务器程序包。它的流行意味着人们会不断地开发出稳定和安全补丁，用来修复所发现的任何漏洞。如果你需要确保安装的安全性并且在发现新漏洞时及时安装补丁，那么Apache可能是最佳选择。

6.1.2 找到具有丰富知识的工程师非常重要

尽管Nginx和lighttpd都是主流，但是在社区中它们并不像古老的Apache服务器那样知名。如果你寄希望于能够找到和雇用具有一流的Web服务器技术的工程师，那么Apache会再次成为最佳选择。

6.1.3 你的网站主要是静态内容

如果你运行的是一个照片或视频托管网站，则不妨好好考虑lighttpd或Nginx这些新出现的工具，它们可以提高静态对象处理的性能。

6.1.4 你在托管服务中托管

传统上，很多托管服务都放在提供托管应用程序的组件中。在采用依赖非Apache的其他服务器的设计之前，请查询一下你的托管服务提供商是否支持这项服务。

6.1.5 你正在使用不常见的 PHP 扩展

大多数PHP扩展的编写者都假定PHP将运行在带有mod_php模块的Apache上。lighttpd和Nginx使用FCGI模式下的扩展来托管PHP解释器，这些扩展通常没有做过测试。在使用这些可选的Web服务器程序包之前，请检查你的应用程序所需的所有PHP扩展是否在FCGI模式下正常工作。

6.2 Web 服务器的使用情况图表

选择Web服务器程序包时应该考虑的一个因素是它的流行程度。如果一个Web服务器非常流行，则意味着有很多人使用它，人们会发现错误，也会提供支持服务，等等。不太流行的Web服

务器的曝光程度可能也不如流行的服务器（参见表6-1）。

表6-1 Web服务器程序包流行程度（来源：Netcraft 2010）

供 应 商	产 品	托管的站点（百万）	百 分 比
Apache	Apache	111	54
Microsoft	IIS	50	24
Igor Sysoev	Nginx	16	8
Google	GWS	15	7
Lighttpd	lighttpd	1	0.46

但是，在放弃某些服务器（如Nginx或lighttpd）之前，你应该知道，由于它们自身性质的缘故，以及它们作为快速的静态内容服务器的良好口碑，它们经常被用作较大型域的“支持服务器”，只负责处理静态资产（图像、.css、.js），而且不一定会记录在Netcraft图表中。

6.3 Web 服务器请求的处理

Web服务器必须按照HTTP规范指定的步骤对入站请求执行一系列定义明确的处理。为了生成所需的输出，我们将这一系列事件称为“请求处理管道”（Request Processing Pipeline），因为它描述了处理请求所需执行的操作系列或管道。

尽管在每个Web服务器程序包中这些事件发生方式的细节有很大不同，但是它们一般都遵循相同的模式。

- 请求侦听器：该组件负责从浏览器获取入站网络连接以及从套接字读取请求。
 - 请求解析：获取请求并将其解析为可以由其他Web服务器组件轻松解释的数据结构；大多数Web服务器将解析后的数据以“请求对象”的形式来公开，所请求的目标就是在服务器上运行的应用程序。它还负责一些解码工作，例如对Cookie Jar进行解码并使其可用作值词典。
 - 输入筛选器：该组件负责需对请求进行所需的转换。如果Web服务器能够执行任何URL重写，那么通常会在此阶段执行。
 - URI映射：这是将输入请求中的URI映射为Web服务器中的物理目录和文件的位置。所有安全、位置或目录选项都在此阶段应用。
 - 请求处理：Web服务器从磁盘获取页面的内容。也可能从缓存的文件（如果Web服务器支持对象缓存）获取。
 - 输出筛选器：对于动态语言（如PHP），输出筛选器用于将文件的源代码转换为输出HTML，如果需要的话，它也会进行压缩。其他输出编码（如分块）也在此时应用。
 - 输出传输：管道中的最后阶段；检查返回给用户Web浏览器的内容的传输。
- 图6-2显示了对托管文档的典型请求。



图6-2 请求处理管道

浏览器对在指定的URL上的托管文档执行一个请求时，Web服务器将侦听特定端口号下的所有进入的请求。当Web服务器遇到对PHP文件的请求时，它要么打开一个连接（如果尚未达到其限制），要么用当前打开的任何连接满足该请求。然后Web服务器通过PHP引擎处理该请求、发送响应，完成后最终关闭该连接。在此过程中，Web服务器可以执行其他操作，如将把请求中的错误（如果有）记录到访问日志中以及在发送到浏览器之前压缩数据。

如果必须多次打开和关闭连接，或者上述有些无关步骤并不重要但却占用了宝贵的CPU时间，那么所描述的这个过程可能很快就会成为速度瓶颈。

尽管这些步骤可能只花费几毫秒的时间，但是这些时间累加起来，直到这些步骤完成，用户的浏览器才会收到结果。如果我们可以通过去除某些步骤或减少执行某些步骤所需的时间来加快该过程，便可以加快响应，从而提高应用程序速度。

6.4 Web 服务器硬件

硬件很重要。如果你没有最理想的硬件但仍期望同时满足成千上万名访问者，那么本章中告诉你的技巧可能不会起到很大的作用。但出于同样原因，我也不建议你去购买当下最好的硬件。我要说的是根据你的经济能力进行购买并通过使用本书来充分利用你的硬件。使用最佳设置配置

你的Web服务器并使用你的当前硬件，可以为你节省资金。

总之，拥有备用内存，可以创建更多进程来处理进入的请求，并且拥有多个CPU可以提高Web服务器的性能。由于市场上存在各种不同类型的Web服务器系统，因此我们将在以下部分中深入研究不同类型的Web服务器。

6.5 对Web服务器进行分类

并非所有Web服务器都相同。如前所述，你的硬件设置取决于你运行Web服务器的方式。某些Web服务器有多个CPU，且带有超大容量的内存（达到GB数量级），而有些Web服务器只有一个CPU，它带有的内存也十分有限。由于这些服务器硬件的布置各不相同，因此有两种不同类型的Web服务器值得深入讨论：

□ Prefork/Fork

□ Threaded

Apache是最知名的Web服务器，人们通常将其部署为prefork Web服务器，但也可以使用其他模式（请参见前面的“Apache多处理模块”）。prefork Web服务器允许Web服务器创建一个不以原有进程为基础的单进程（fork）。作为prefork服务器类型运行的Web服务器倾向于创建一个进程池，供入站的用户使用。我们推荐单CPU服务器使用这种类型的Web服务器。在表6-2中可以找到每种类型Web服务器的完整描述。我们还建议任何使用链接库（如数据库集成、文本处理等）的应用程序也使用这种类型的Web服务器，因为这些第三方扩展中的大部分未证明是线程安全的，在重负载下运行可能会导致一些问题。

表6-2 Web服务器类型

Web服务器类型	说 明
Prefork	基于进程的Web服务器；使用fork进程来满足每个进入的请求
Threaded	基于线程的Web服务器；使用线程来满足每个进入的请求

在单个CPU上，将prefork与thread和worker相比时，prefork服务器类型的性能非常高。随着并发级别的提高，prefork仍然保持稳定，而其他服务器类型的处理时间会增加。再次说明，所有这些都取决于你所使用的服务器类型及其配置。

接下来我们将讨论这两种类型服务器的异同点。根据每种类型的优点以及你的硬件和处理需求，就可以确定选择将Apache配置为哪种类型。

6.6 Apache HTTPD

Apache Web服务器是目前为止最广泛应用于开发和生产环境的Web服务器。在撰写本书的时候，它的最新版本是2.2，该版本可用于Unix和Windows系统。Apache的绝妙之处在于设置阵列（array of settings），它允许你在其配置文件中进行调整。默认情况下，配置文件包含运行应用程序使用的设置以及不使用的设置，但当你的网站流量增加时，其中的一些设置可能不是太理想。

注意 用于Apache服务以及它使用的配置文件的名称通常存在某些混淆。对于基于Debian的分发（例如Ubuntu），程序包的名称为“apache2”，并且在其服务名称以及其配置文件“apache2.conf”的名称中会同时使用该名称。对于基于Red Hat的分发（例如Fedora等），进程名称为“httpd”并且拥有一个相应的“httpd.conf”配置文件。这个差别贯穿始终，包括用于配置、日志记录的目录名称。在本书中，我们使用的是“apache2”，但在从Red Hat派生的系统上，请替换为“httpd”。

有两种安装Apache的方法：从程序包安装或从源文件安装。我们将针对两个最常用的分发系列Debian（SUSE、Ubuntu）或Red Hat（Red Hat、CentOS、Fedora）来介绍如何从程序包安装。

附录A包含在Windows上将Apache作为完整LAMP栈的一部分安装的详细说明，附录B则提供了在Linux服务器或工作站上安装类似说明。

6.6.1 Apache Daemon 命令行

在深入研究配置文件的修改之前，我们先简要介绍一下命令行选项。通过使用命令选项，我们可以进一步了解已加载的模块，检查Web服务器类型，测试对配置文件所做的更改是否生效以及其他内容。表6-3列出了可用的命令行选项的完整列表。

表6-3 Apache 2 命令行参数

参 数	描 述
-D name	设置名称以便在<IfDefine name>指令中使用
-d directory	设置ServerRoot
-f file	设置ServerConfigFile
-C "directive"	在读取配置文件之前对指令进行处理
-c	在读取配置文件之后对指令进行处理
-e level	显示日志级别为“level”的启动错误
-E file	将启动错误记录到“file”文件中
-v	显示版本号
-V	显示编译设置
-h	显示可用的命令行选项
-l	列出已编译的模块
-L	列出可用的配置指令
-t -D DUMP_VHOSTS	显示解析的设置，目前只有VHOSTS
-S	-t -D DUMP_VHOSTS的快捷方式
-t -D DUMP_MODULES	显示已加载的模块
-M	-t -D DUMP_MODULES的快捷方式
-t	对配置文件运行语法检查

这些命令行参数可以帮助你了解已安装的内容（如果Web服务器不是由你亲自安装的）。若

要检查模块，请运行 `apache2ctl -M` 命令。你应该会得到系统中所有已加载模块的代码清单，例如 `core.c`、`prefork.c` 以及 `http_core.c`。该命令还为我们提供其他关于 Apache 当前配置的 Web 服务器类型（如 `prefork`）的信息。

默认情况下，Apache 把最重要的文件安装到它的安装目录中。在 Unix 系统上，该目录为 `/etc/apache2`，而对于 Windows 用户，该目录则为 `C:\Program Files\Apache Software Foundation\Apache2.2`（如果使用默认的安装值）。

在该目录中，你将发现 `apache2.conf`。这个文件包含一个配置设置列表，列出了从加载文件的目录（`DocumentRoot`）到 Web 服务器在某一时刻可以同时支持的用户数的各种设置。如下一节要讨论的，通过修改某些设置，你可以配置 Web 服务器以满足你的需要。

稍后在第 7 章，我们还将介绍一些设置，通过修改它们可以优化 Web 服务器从而实现更高性能。

6.6.2 Apache 多处理模块

Apache 使用一组多处理模块（MPM）确定它将使用哪些进程和内存模块来处理每个请求。请求首先被放到请求队列中，然后通过 MPM 分派给将负责处理它们的进程。Apache 自带一组用于不同 OS 架构和处理模型的 MPM。

首先，通过使用 `apache2ctl -M` 或 `httpd -M` 命令来确定加载的模块，以此来确保你的系统使用的是正确的 MPM。在输出中，你应该会看到表 6-4 所示的模块之一。

表 6-4 Apache 中可用的 MPM 列表

操作系统	MPM
BeOS	BeOS
Netware	Mpm_netware
OS/2	Mpm_os2
Unix	Prefork、Perchild、Threadpool、Worker
Windows	Mpm_winnt

如果你的系统未运行正确的模块，可以使用 `-with-mpm=<MPM Value>` 在编译期更改该设置。可用的 Unix MPM 如下。

- **Prefork**：Apache 预先创建一组子进程。
- **Perchild**：Prefork 的一个变体，它允许为子进程设置单独的进程权限。
- **Threadpool**：Apache 使用多线程来实现请求处理程序——对于大多数 Unix 系统，不建议使用，原因是无法保证线程安全。
- **Worker**：Prefork 和 Threadpool 的混合，其中每个子进程都支持多个线程。

在基于 Unix 的系统上，几乎 99% 的人都使用 Prefork MPM，一些人在他们的专用应用程序中使用 Worker MPM。一般来说，任何多线程的 MPM 都是不推荐的，因为很多绑定到 Apache 和 PHP 解释器的库文件未证明是线程安全的，并且在多线程的环境中执行时可能会产生错误的结果。为了简便起见，我们将只讨论 Prefork MPM。

Prefork MPM

Prefork是目前为止最常用的MPM形式，并且在大多数Unix Apache分发版中是默认安装的。启动时，Apache将创建固定数量的子进程来处理请求，然后根据需要请求路由到每个子进程，并且将根据访问服务器的请求数的增加或减少，在已定义的限制之间扩大或减小子进程池。

Apache将循环利用池中的子进程，直到达到每个子进程的最大请求数，此时它将销毁子进程并产生一个新的进程来替换该进程。设计这种机制的目的就是防止那些已经占用很多内存或者有其他问题的进程被永久锁定到系统中。一旦一个子进程处理完为它分配的请求，它就会死亡并重生。

6.7 了解 Apache 模块

MPM是一个可加载的组件，它控制如何分派和处理请求，以及使用哪个内存/进程模块来处理请求。Apache模块实际执行对请求的处理。Apache有一个内置的模块，称为core_module，该模块提供了一些用来处理静态内容的基本功能。它负责在磁盘上查找要服务的对象并将该对象发送回客户端。

Apache使用了“挂钩”（hook）机制，具体来说，在每个允许动态加载或动态模块编辑的阶段，Apache都使用一组挂钩将自己插入到执行链中，并添加额外的功能。图6-3显示了Apache模块如何将自己附加到处理管道中，以及如何在标准核心处理中添加或替换组件。通常，Apache模块是根据其可加载的文件名（如mod_php、mod_rewrite或mod_deflate）命名的。

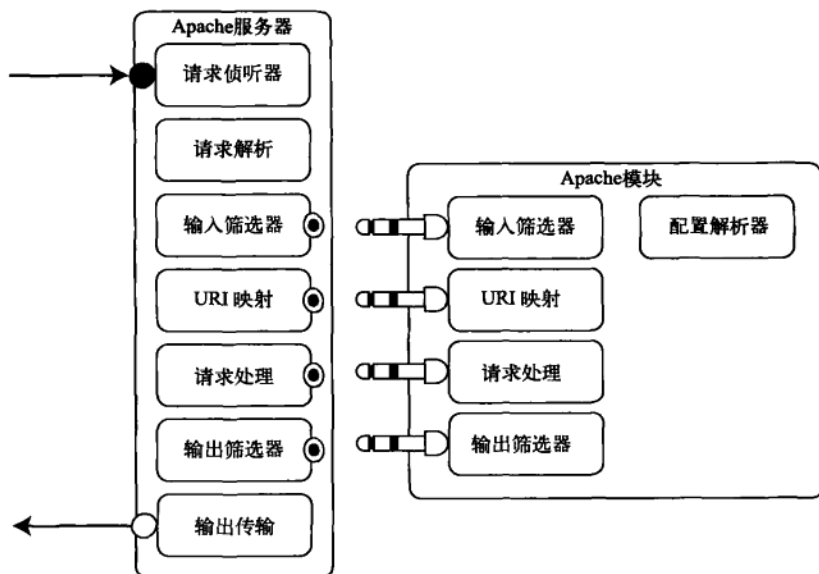


图6-3 附加到处理管道的Apache模块

我们可以在每个请求中加载和调用多个模块，每个模块将自己放置在处理链的相应部分中，以便能够处理之前模块的输出。

用`mod_php`模块举个例子，这是一个负责将PHP源代码转换为HTML输出的模块，它可以通过输出筛选器这个阶段将其输入传递给`mod_deflate`，然后`mod_deflate`使用`gzip`压缩来压缩输出的模块。每个模块都有一个输出筛选器阶段，并且阶段按照正确的顺序连接到一起。

6.7.1 添加动态 Apache 模块

根据系统部署Apache方式的不同，有多种启用动态模块的方法。通常，启用模块的方法是向Apache配置文件中添加一个“LoadModule”声明。但是，有几种“惯例”可以使得该操作更加简便，你在安装的时候可能就用到了其中的一种。

1. 使用`conf.d`目录

基于Red Hat的分发版有一个名为`/etc/httpd/conf.d`的目录，该目录包含在配置文件中，紧随LoadModule部分之后。Apache启动时，会自动包含该目录中扩展名为`.conf`的任何文件。这样，你就可以将包含“LoadModule”行，同时包含配置模块所需的任何默认全局指令的文件，保留在该目录中。

2. 使用模块管理帮助程序

基于Debian的分发版（如Ubuntu）有一种机制，它对上述的单独模块配置目录进行了扩展，添加了一个用于管理其内容的工具。目录`/etc/apache2/mods-available`包含分发版中每个可用模块的单独的配置文件，另一个目录`/etc/apache2/mods-enabled`包含`/etc/apache2/mods-available`中模块的符号链接。若要启用模块，例如，用于映射URL的模块`mod_rewrite`，可以执行以下命令。

```
$sudo a2enmod rewrite
```

该命令将在`/etc/apache2/mods-enabled`目录中创建所需的符号链接。

6.7.2 删除动态 Apache 模块

尽管向Apache的安装中添加正确的模块很重要，但是，保持模块列表简短，以及仅使用运行应用程序所需的模块也不能忽视。再次使用`-M`，你可以检查安装的模块，或者检查配置文件。使用Windows Apache配置，你可以通过在文件中查找单词“LoadModule”来检查模块列表。若要移去模块，可以查找包含你要禁用的模块名称的“LoadModule”行（模块名称位于LoadModule的右侧），然后只需在该行的开头放置一个注释标记代码“#”，该模块便不会再加载了。删除该行并不好，因为以后你可能想还原该模块，注释掉该文件是比较安全的做法。完成之后，保存更改并重新启动Apache。

1. 使用`conf.d`目录删除模块

如果你的分发版使用`conf.d`目录，那么删除模块会很容易，只需将文件重命名为类似于`mod_rewrite.conf.bak`的名称，然后重新启动Apache即可。

2. 使用模块管理帮助程序来删除模块

在Debian (Ubuntu) 分发版上，可以使用模块帮助程序删除模块配置文件的链接。

```
$sudo a2dismod rewrite
```

然后重新启动Apache。

6.8 关于 Apache 的最后几点

毫无疑问，Apache是目前为止功能最全面的Web服务器程序包。它拥有大量用户，并且在互联网上广泛使用，最大型的网站大多用它来提供内容。

它是最出色的多面手并且拥有卓越的支持，但全面性是需要付出代价的，那就是它无法成为最快速的解决方案。如果你的应用程序不需要Apache提供的灵活性和广泛支持，那么也可以选择其他的Web服务器，下面我们就来看一下哪种可能适合你。

6.9 lighttpd

如果要对lighttpd的特点进行一下概括，可参见它的分发版支持站点，这个站点的第一段话便对lighttpd做了很好的概括：安全性、速度、合规性以及灵活性——所有这些都是对lighttpd（发音为lighty）的最好描述，它的出现正在迅速改变Web服务器效率的定义，因为它是针对高性能的环境进行设计和优化的。与其他Web服务器相比，它占用的内存较少，并且可以有效管理CPU负载，同时它还拥有高级特性集（FastCGI、SCGI、Auth、输出压缩、URL重写以及更多），对于每个受到负载问题困扰的服务器，lighttpd都是完美的解决方案。

lighttpd是由Jan Kneschke编写的，它最初是作为一个研究“c10K”问题，以及如何在一个服务器上创建可以支持10 000个并发连接的Web服务的实验系统。由于这一点，它作为针对静态内容的快速Web服务器赢得了赞誉。

6.9.1 安装 lighttpd

可以使用资源库中的程序包或者通过网站上的源代码来安装lighttpd。此Web服务器有两个版本，一个版本基于Unix，另一个版本是Windows二进制文件。我们会详细介绍在每种系统上安装此Web服务器的过程。请跳至最适合你的部分——这样你并不会错过任何信息。

1. 在Unix上安装lighttpd

如前所述，lighttpd适合于资源库中的大多数基于Unix的系统。当前它可用于Debian、Ubuntu、OpenSUSE以及其他系统。用于安装该程序包的资源库和命令的完整列表如表6-5所示。

我将通过运行第二行的命令在Ubuntu系统上使用Aptitude资源库进行全新安装。

默认情况下，lighttpd随下列功能一起安装（如果使用资源库安装）：

☐ IPv6

☐ Zlib

- ❑ Bzip2
- ❑ Crypt
- ❑ SSL
- ❑ mySQL
- ❑ Memcached
- ❑ SQLite

表6-5 可用于lighttpd安装的资源库

资源库	命令
Zypper	zypper install lighttpd
Aptitude	apt-get install lighttpd lighttpd-doc
Yum	yum install lighttpd
Emerge	emerge lighttpd

此外,通过使用lighttpd -v命令,你还可以深入了解此Web服务器的配置以及其他设置。lighttpd命令的完整列表如表6-6所示。修改配置文件时将显示这些命令。

有3个目录是你应该知道的——log、www和配置目录。在我的安装中, www目录,也就是用于放置所有Web应用程序的目录,它的位置是/var/www。Log目录位于/var/log/lighttpd/,包含Web服务器配置设置的目录位于/etc/lighttpd/。

2. 在Windows上安装lighttpd

直到最近, Windows的lighttpd版本还需要在系统中安装Cygwin。你可以在<http://en.wlmp-project.net/downloads.php>网站上获得最新的二进制版本的此Web服务器安装。该网站提供了最新的稳定版1.4.X,它可用于Windows 2000、XP、2003、Vista、2008以及Windows 7。

若要实现快速、无缝的安装,请下载安装向导并按照所示的步骤执行操作。安装完此Web服务器之后,你将拥有一个包含很多项的目录,如图6-4所示。



图6-4 lighttpd目录结构

该目录包含很多非常重要的文件。LightTPD.exe可以用来启动Web服务器。Htdocs目录是放置Web应用程序的位置，logs目录包含错误和访问日志，conf目录包含所有可用的配置选项。

若要启动此Web服务器，请打开目录bin并双击Service-Install.exe文件。（注意，这将安装一个服务，从而每次启动操作系统时都启动lighttpd，这一点非常重要。）

该文件将打开命令行提示符并运行如图6-5所示的一系列项。

```

OLMP Project ToolKit - LightIPD Service Installation Tool
*****
Copyright (C) 2006-2009.
OLMP Project IEAM / D-Club Soft.

- Copy Service executable to 'C:\WINDOWS\LIGHTSRC.exe'
- Install LightIPD Service
- Add Service values to Registry
- Add LightIPD to Windows Firewall

LightIPD Service installation successful.

Do you want to start LightIPD Service? (Y/N) Y

- Start LightIPD Service
The LightIPD service is starting.
The LightIPD service was started successfully.

Press any key to continue...

```

图6-5 Windows安装过程窗口

这个过程完成后，按任意键并打开http://localhost/，可看到如图6-6所示的lighttpd欢迎页面。

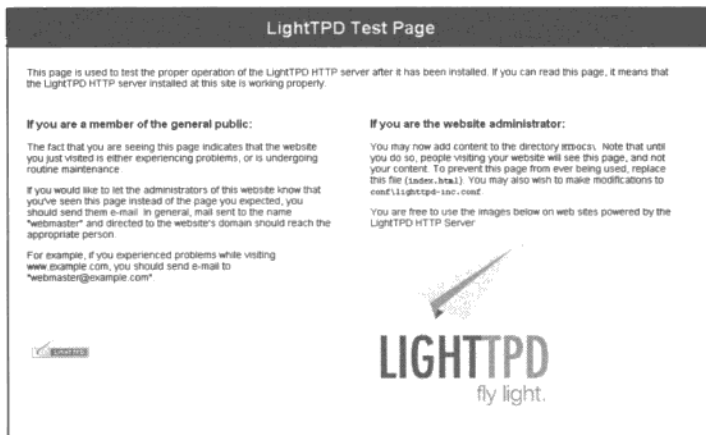


图6-6 Windows lighttpd欢迎页面

6.9.2 lighttpd 配置设置

由于配置文件是本节的重点，我们将多花点时间进行讨论，我们需要熟悉配置文件以及测试配置更改的工具。接下来，我们将学习lighttpd命令行命令。

打开一个命令窗口并键入 `lighttpd -h`。在基于 Windows 的系统上,键入 `LightTPD.exe -h`。这将显示可以运行的命令列表。例如,若要加载不同的配置文件,可以使用 `-f` 标志。通过使用该标志,可以加载位于系统中任何位置的任何配置文件。若要查看 `lighttpd` 的版本,可以使用 `-v` 标志。这些命令的完整列表如表 6-6 所示。

表 6-6 lighttpd 的命令行选项

标 志	说 明
<code>-f <name></code>	配置文件的完整路径
<code>-m <name></code>	模块目录的完整路径
<code>-p</code>	显示解析的配置文件
<code>-t</code>	测试使用了 <code>-f</code> 的配置文件
<code>-D</code>	不转到后台
<code>-v</code>	版本
<code>-V</code>	编译期间选项的列表
<code>-h</code>	“帮助”菜单

现在,让我们看一下配置文件。配置文件包含可提高 `lighttpd` 性能的所有可用的选项。它包含很多信息,例如 Web 目录所在的位置、要使用 PHP 处理或排除的文件、要在启动时加载的模块以及要侦听的端口等。

打开配置文件。配置文件中的默认设置只是所有可用配置中很小的一部分。`server.modules` 包含一个要使用的模块列表(用逗号分隔)。默认情况下,将安装 `mod_access`、`mod_alias`、`mod_accesslog` 以及 `mod_compress`。尽量保持此列表简短是保证应用程序最佳性能的关键。`server.document-root` 设置包含 Web 目录的完整路径, `server.errorlog` 包含 Web 服务器错误日志的完整路径。未放入配置文件中的设置有 `server.max-connections`、`server.max-fds` 以及 `server-max-keep-alive-idle`。使用这些设置,我们可以设置最大连接总数、设置文件描述符(文件处理程序)的最大数量以及断开空闲连接的最大秒数。

<http://redmine.lighttpd.net/projects/lighttpd/wiki/Docs:ConfigurationOptions> 上显示了配置设置的完整列表。

6.9.3 比较静态负载内容

我们希望优化我们的 Web 服务器,因此我们需要了解它在出厂时的速度。为此,我将根据本书开头部分提到的规范,使用运行 Ubuntu 的计算机在 `lighttpd` 上运行一个基准测试。我也在这台计算机上运行过其他基准测试,所以要确保每次运行测试时都重新启动此 Web 服务器。

我将运行下面的 `ab` 测试,该测试通过同时在 Apache 和 `lighttpd` 上的 500 个并发请求来模拟 1000 个请求。目标是选择使应用程序运行速度最优的 Web 服务器。

代码清单 6-1 显示了运行此基准测试所使用的命令行。

代码清单6-1 通过500个并发请求来模拟1000个连接的ab测试命令

```
Ab -n1000 -c 500 http://localhost/
```

在lighttpd上模拟负载的结果如图6-7所示。

```

Concurrency Level:      500
Time taken for tests:    0.074 seconds
Complete requests:      1000
Failed requests:         0
Write errors:           0
Total transferred:      435000 bytes
HTML transferred:       177000 bytes
Requests per second:    13450.80 [#/sec] (mean)
Time per request:       37.172 [ms] (mean)
Time per request:       0.074 [ms] (mean, across all concurrent requests)
Transfer rate:          5713.96 [Kbytes/sec] received

Connection Times (ms)
      min   mean[+/-sd] median   max
Connect:    0     5   5.1      1     14
Processing:  6    10   4.4      8     22
Waiting:    4    10   4.6      8     22
Total:      7    15   7.2     15     29

```

图6-7 静态HTML文件上的基准测试结果

为什么我们要测试静态内容呢？如果你拥有一个包含四台服务器的服务器集群，可以用一台服务器（特别是轻型服务器）来处理所有图像和静态内容，用另外三台服务器来分担PHP内容负载。

现在，让我们比较一下运行PHP脚本的Web服务器。为此，我们需要使用FastCGI安装PHP。

6.9.4 在 lighttpd 上安装 PHP

在lighttpd上安装PHP非常简单，并且可以使用CGI或FastCGI PHP版本来完成。我建议使用FastCGI，其原因显而易见（顾名思义）。

FastCGI PHP可用于Unix和Windows，因此我将详细介绍这两种方法。首先我们看一下要安装的Unix版本。对于Windows用户，可以跳至有关在Windows环境中安装FastCGI PHP的下一部分。如果你是Unix用户，你将需要安装大多数资源库中都提供的php5-cgi程序包。如果使用Ubuntu或Debian，则可以运行如代码清单6-2所示的命令。

代码清单6-2 使用Aptitude安装FastCGI PHP

```
apt-get install php5-cgi
```

如果此程序包没有遇到任何问题，你应该会在/etc/php5目录中得到一个新目录。打开/etc/php5/cgi/php.ini文件，添加代码清单6-3中所示的文本并保存更改。为了确保fcgi版本的PHP正确设置php变量\$_SERVER['PATH_INFO']的值，必须进行此更改。某些应用程序使用了此变量，其默认行为会复制一个bug（这是先前fcgi实现中的一个bug）。

代码清单6-3 在php.ini文件中打开FastCGI

```
#[FastCGI PHP]
cgi.fix_pathinfo = 1
```

现在，我们需要将Web服务器配置为当它遇到扩展名为.php的文件时，使用FastCGI处理所有文件。打开位于conf目录中的lighttpd.conf文件。在我的安装中，此文件位于/etc/lighttpd/目录中。打开此文件并添加代码清单6-4中的文本。

代码清单6-4 更新lighttpd.conf，添加FastCGI模块

```
...
server.modules = (
    ...
    "mod_fastcgi"
)
...
```

在此文件中，将代码清单6-5中所示的文本添加到文件的结尾。

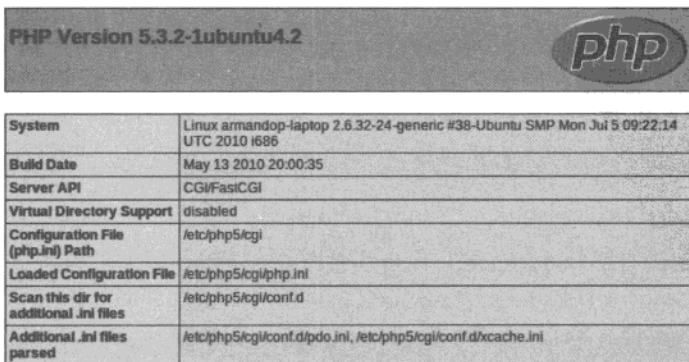
代码清单6-5 fastcgi.server设置

```
...
fastcgi.server = (".php" => ((
    "bin-path" => "/path/to/your/php-cgi",
    "socket" => "/tmp/php.socket"
)))
...
```

代码清单6-5设置了fastcgi.server设置。这个设置声明所有.php文件都应该使用php-cgi二进制文件（文件的位置是在“bin-path”中设置的）。保存此文件并重新启动服务器。

1. 验证PHP安装

若要验证安装是否成功，可创建一个phpinfo.php PHP文件并将它放在web-root目录中。从浏览器中请求该文件，如果一切成功，你应该会看到类似于图6-8的内容。



PHP Version 5.3.2-1ubuntu4.2	
System	Linux armandop-laptop 2.6.32-24-generic #38-Ubuntu SMP Mon Jul 5 09:22:14 UTC 2010 i686
Build Date	May 13 2010 20:00:35
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/etc/php5/cgi/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
Additional .ini files parsed	/etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/xcache.ini

图6-8 lighttpd phpinfo页

2. 对PHP内容进行基准测试

下面我们将使用代码清单6-6中所示的代码运行我们的ab测试并获取结果。这些结果不仅仅可以帮助我们使用每个Web服务器的默认设置来比较结果，而且还可帮助我们在下一部分测量调整的结果。

代码清单6-1显示了我将用于此测试的ab命令，代码显示在代码清单6-6中。

代码清单6-6 要测试的代码段

```
<?php
$max = 10000;
$x = 0;
$array = array();

while($x < $max)
{
    $array[$x] = $x;
    $x++;
}

foreach($array as $z)
{
    echo "$z<br/>";
}
```

运行此测试 5 次之后，我得到的最高结果如图6-9所示。

```
Concurrency Level:      500
Time taken for tests:    3.945 seconds
Complete requests:      1000
Failed requests:         0
Write errors:           0
Total transferred:      89124898 bytes
HTML transferred:       88963530 bytes
Requests per second:    253.47 [#/sec] (mean)
Time per request:       1972.640 [ms] (mean)
Time per request:       3.945 [ms] (mean, across all concurrent requests)
Transfer rate:          22060.80 [Kbytes/sec] received
```

图6-9 在lighttpd服务器上对代码清单6-6运行ab测试的结果

图6-8中所示的结果表明服务器达到的每秒最大请求数是253.47，并且每个请求的平均时间为1972.640毫秒（1.9秒）。尽管对于单台服务器来说，这些都是比较可观的数字，但是让我们看一看如何调整服务器设置才能获得更好的性能。

3. 设置调整

我们将增加文件描述符的个数，去除获取文件的开销，并为我们的系统设置所需的PHP进程数。

我们所要做的第一件事就是增加文件描述符的个数。文件描述符是一种文件处理程序，用户（或一个请求）可以通过它来访问服务器中的指定文件。如果Web服务器的文件描述符不够用，则会向用户返回错误，并且你的错误日志将很快被填满以下日志内容：

```
(server.1345)socket enabled again
(server.1391)sockets disabled, connection limit reached.
```

为了消除这个问题，我们增加`server.max-fds`的值。默认情况下，`lighttpd`将该值设置为1024（大多数情况下）。此时，我们的服务器只能处理512个连接（ $1024/2 = 512$ ）。`Lighttpd`网站建议在比较繁忙的服务器上将该值增加到2048。这将允许最多1024个连接。若要提高文件描述符的最大数量，可使用`server.max-fds`属性，将它的值设为2048并且还要设置`server.max-connections=1024`。

我们可以进行的下一个配置更改是通过禁用、缓存或使用FAM来控制`stat`调用，从而去除每个请求多次调用`stat()`的开销。使用`server.state-cache-engine`参数，可以将该值设置为`disable`、`simple`或`fam`。

6.10 Nginx

我们要介绍的最后一个HTTP Web服务器是Nginx（engine-x）。Nginx不仅仅是一个HTTP Web服务器，而且还可以用作反向代理和IMAP/POP3邮件服务器。安装Nginx的目标是确定PHP脚本在此Web服务器下执行的效果。根据Nginx的官方网站（www.nginx.org）的说法，Nginx是Igor Sysoev在2002年创建的。它还托管了全球6.55%的域并大力宣传Wordpress.com和Hulu（作为其用户）。到目前为止，最新的稳定版本为0.7.x。之前的版本仍然可用，并且版本0.7.x可用于Windows和Unix系统。

与基于进程的Web服务器Apache不同，Nginx是异步的Web服务器。这意味着Nginx将产生非常少的线程来支持并发请求或者不产生线程，而Apache Web服务器的每个并发请求都需要一个新线程。因此，Nginx所提供的最出色功能之一就是在高流量负载下只占用很少量的内存。

6.10.1 安装 Nginx

Nginx既可用于Windows，也可用于Unix，在官方网站上可以找到这两个版本。我将首先在基于Unix的系统上安装此Web服务器，之后在基于Windows的系统上安装。你可以直接跳至你的系统需要的部分，这并不会错过任何重要的信息。在这两种情况下，我们都将参考Nginx的网站<http://wiki.nginx.org>。

1. 在Unix上安装Nginx

目前，资源库中提供了我们安装的大多数程序包。现在，我们将通过在命令窗口中运行表6-7所示的`apt-get`命令在基于Ubuntu的系统中安装Nginx。请参阅此表并使用与你的操作系统相对应的命令。

表6-7 安装Nginx的命令

操作系统	命 令
Red Hat/Fedora	<code>yum install nginx</code>
Ubuntu/Debian	<code>apt-get install nginx</code>

执行适用于你的系统的命令即可正确安装所需的程序包。如果你遇到问题，请读一读输出，因为很多时候它都会包含有关缺少哪个程序包或遇到了什么问题的信息。

也可以选择从源代码安装。如果你希望从源代码安装Nginx，请打开你的浏览器并加载<http://wiki.nginx.org/NginxInstall>页面。有三个选项：稳定版本、正开发的版本和历史版本。你选择适当程序包后，下载该程序包并在你的本地驱动器中展开它，然后运行代码清单6-7所示的命令。

代码清单6-7 从源代码安装Nginx的命令

```
./configure [compile-time options]
make
sudo make install
```

此时，Nginx应该已经安装在你的系统上并且可以使用了。

2. 编译期选项

默认情况下，使用资源库命令来安装Nginx会安装代码清单6-8所示的配置设置。

代码清单6-8 默认配置设置

```
conf-path=/etc/nginx/nginx.conf
--error-log-path=/var/log/nginx/error.log
--pid-path=/var/run/nginx.pid
--lock-path=/var/lock/nginx.lock
--http-log-path=/var/log/nginx/access.log
--http-client-body-temp-path=/var/lib/nginx/body
--http-proxy-temp-path=/var/lib/nginx/proxy
--http-fastcgi-temp-path=/var/lib/nginx/fastcgi
--with-debug
--with-http_stub_status_module
--with-http_flv_module
--with-http_ssl_module
--with-http_dav_module
--with-http_gzip_static_module
--with-http_realip_module
--with-mail
--with-mail_ssl_module
--with-ipv6
--add-module=/build/builddd/nginx-0.7.65/modules/nginx-upstream-fair
```

默认配置包含很多重要信息，如错误日志的路径、访问日志的路径、配置文件、SSL支持、邮件支持以及已启用的服务器状态页等。若要更改这些设置，你有两个选项：对conf-path中指定的配置文件进行修改或使用一些编译期选项重新编译。表6-8列出了最常用的编译期选项以及默认情况下所安装的配置选项的描述。在<http://wiki.nginx.org/NginxInstallOptions>网站上可以找到完整列表。

表6-8 Nginx 编译期设置

设 置	说 明
--prefix=<path>	所有其他设置将使用的相对路径，默认情况下设置为 /usr/local/nginx.
--conf-path=<path>	配置文件的位置，默认值为 <prefix>/conf/nginx.conf
--pid-path=<path>	nginx.pid的路径，默认值为 <prefix>/logs/nginx.pid
--error-log-path=<path>	所使用错误日志的路径；默认值为 <prefix>/logs/error.log
--http-log-path=<path>	所使用访问日志的路径；默认值为 <prefix>/logs/access.log
--user=<user>	将运行Nginx的默认用户；默认值为 “nobody”
--group=<group>	将运行Nginx的默认组；默认值为 “nobody”
--lock-path=<path>	锁定文件的路径
--http-client-body-temp-path=<path>	HTTP客户端临时请求文件的路径；默认值为 <prefix>/client_body_temp
--http-proxy-temp-path=<path>	HTTP临时代理文件的路径；默认值为 <prefix>/proxy_temp
--http-fastcgi-temp-path=<path>	FastCGI临时文件的路径；默认值为 <prefix>/fastcgi_temp
--without-http	关闭HTTP服务器
--with-debug	打开调试日志
--with-http_stub_status_module	打开服务器状态页
--with-http_flv_module	打开flv模块
--with-http_ssl_module	打开ssl模块
--with-http_dav_module	打开dav模块
--with-http_gzip_static_module	打开gzip模块
--with-http_realip_module	打开realip模块
--with-mail	打开IMAP4/POP3/SMTP代理模块
--with-mail_ssl_module	打开mail ssl模块
--add-module=<path>	位于指定路径中的第三方模块

3. 验证安装并启动Nginx

若要启动Nginx,请在命令窗口中执行代码清单6-9所示的命令。表6-9列出了其他命令行选项。

代码清单6-9 启动Nginx

```
Nginx
```

表6-9 Nginx命令行选项

选 项	描 述
-s [stop quit reopen reload]	允许停止、退出、重新打开或重新加载Nginx Web服务器
-h	可用选项的列表
-v	显示版本号
-V	显示配置选项
-t	测试配置文件；做出更改时非常有用
-p <prefix>	设置前缀路径
-c <filepath>	设置要使用的配置文件
-g <directives>	设置全局指令

运行此Web服务器之后，加载URL <http://localhost/>。你应该会看到一个类似于图6-10所示的页面。



图6-10 Nginx的欢迎屏幕

警告 如果你已打开任何其他Web服务器，请确保此时将其关闭。

6.10.2 Windows 安装

在Windows系统上安装时，可以使用Nginx的二进制文件格式。与Unix版本一样，Nginx有三个版本，即稳定版本、正开发的版本和历史版本。最新的稳定版为0.7.67，可以在<http://nginx.org/en/download.html>上找到。将该文件下载到适当的目录下，最好是路径名中不包含空格的目录，然后将文件解压缩。C:\nginx是一个不错的位置。

解压缩完成后，你应该得到图6-11所示的目录结构。

在该目录中，你将找到nginx.exe可执行文件并且该目录中包含6个不同的目录：

- ☐ conf
- ☐ contrib
- ☐ docs
- ☐ html
- ☐ logs
- ☐ temp



图6-11 Windows Nginx
目录结构

Conf目录包含配置文件，配置文件中包含Nginx设置和FastCGI设置。Html目录是用于放置应用程序的位置。logs目录包含Nginx输出的所有日志，temp目录包含Nginx需要创建的所有临时文件。

若要启动此服务器，只需双击位于该目录中的nginx.exe文件并等待其加载。启动Web服务器之后，通过在浏览器中访问http://localhost/来确保此Web服务器安装正确。验证是否显示图6-9。

6.11 Nginx 作为静态 Web 服务器

现在已安装了Nginx，让我们来比较一下Nginx和Apache Web服务器，看看哪个Web服务器对于我们的应用程序来说性能最好。我们将通过运行代码清单6-10所示的命令在Ubuntu系统上测试Nginx的默认安装与Apache的默认安装。

代码清单6-10 ab命令

```
ab -n 1000 -c 500 http://localhost/
```

代码清单6-10所示的命令将通过在保持500个并发连接的情况下请求静态的HTML页面1000次来测试我们的服务器。在给定此负载的情况下，Apache Web服务器的结果如图6-12所示，Nginx的结果如图6-13所示。

```
Concurrency Level:      500
Time taken for tests:    4.207 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      487596 bytes
HTML transferred:       190098 bytes
Requests per second:    237.67 [#/sec] (mean)
Time per request:       2103.724 [ms] (mean)
Time per request:       4.207 [ms] (mean, across all concurrent requests)
Transfer rate:          113.17 [Kbytes/sec] received
```

图6-12 Apache Web服务器的ab结果

```
Concurrency Level:      500
Time taken for tests:    0.074 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      372860 bytes
HTML transferred:       155530 bytes
Requests per second:    13535.83 [#/sec] (mean)
Time per request:       36.939 [ms] (mean)
Time per request:       0.074 [ms] (mean, across all concurrent requests)
Transfer rate:          4928.68 [Kbytes/sec] received
```

图6-13 Nginx Web服务器的ab结果

在这些结果中，我们只对两项感兴趣：每个请求的时间以及每秒的请求数。出于优化的考虑，我们只关注每个请求的时间。Nginx Web服务器将静态文件的响应时间从2103.724毫秒（2秒）（使用Apache）减少到36.939毫秒，减少了2.06秒。

使用Nginx可以满足更多的用户，因此可以最大限度地利用每个Web服务器。我们再来看一下每秒请求数的结果，Nginx可以满足13 535.83个用户，而Apache可以满足237.67个用户。增加了13 298个用户。前面我们仅使用静态内容测试了这些结果，现在让我们看一看使用默认设置在运行PHP时Apache和Nginx的测量结果。为此，我们需要在Nginx上安装PHP。

注意 这里并不是讨论“我的Web服务器比你的更好”。它们只是在我的本地计算机上使用每个Web服务器的默认设置进行的基准测试。

6.11.1 安装 FastCGI PHP

若要在Nginx上使用PHP，我们需要使用FastCGI版本的PHP。为了安装FastCGI版本的PHP，需要在系统上安装php5-cgi程序包以及spawn-fcgi。对于Unix用户，使用代码清单6-11中所示的命令。

代码清单6-11 安装php5-cgi的命令

```
apt-get install php5-cgi
apt-get install spawn-fcgi
```

安装完此程序包之后，你应该得到路径/etc/php5/cgi。进入该目录并打开php.ini文件。在文件底部进行代码清单6-12所示的修改。为了确保fcgi版本的PHP正确设置php变量\$_SERVER['PATH_INFO']的值，必须进行此更改。某些应用程序使用了此变量，其默认行为是产生一个bug（这是先前fcgi实现中的一个旧bug）。

代码清单6-12 修改php.ini文件

```
Cgi.fix_pathinfo=1
```

安装完这两个程序包之后，我们需要启动FastCGI进程。为此，我们运行代码清单6-13中的命令。

代码清单6-13 启动FastCGI

```
/usr/bin/spawn-fcgi -a 127.0.0.1 -p 9000 -u www-data -g www-data -f /usr/bin/php5-cgi -P
/var/run/fastcgi-php.pid
```

代码清单6-13所示的命令使用6个可选命令。第一个选项是-a，它指定了运行此进程的主机的IP。由于是在本地运行此进程，因此我输入了本地IP。第二个选项是-p，它指定了要侦听的端口号。第三个选项指定了将运行此进程的用户，第四个选项是此进程所属的组，第五个选项指定二进制文件的位置，最后一个选项指定了我们要使用的.pid文件的位置。

成功生成此进程之后，你需要更新Nginx配置文件。为此，打开位于/etc/nginx/sites-available/default中的Nginx配置文件。删除被注释掉的FastCGI信息，该信息类似于代码清单6-14中的文本。

代码清单6-14 删除注释掉的FastCGI信息

```
location ~\.php$ {

    fastcgi_pass    127.0.0.1:9000;
    fastcgi_index   index.php;
    fastcgi_param   SCRIPT_FILENAME /var/www/nginx-default$fastcgi_script_name;
    include fastcgi_params;

}
```

所示的代码段通过使用在127.0.0.1:9000中侦听到的FastCGI进程来指定Nginx对任何.php文件都使用FastCGI。保存更改并重新启动服务器。

验证FastCGI安装

创建一个PHP信息页 (phpinfo page) 并将其放置在/var/www/nginx-default/目录中。执行完该操作之后, 加载文件<http://localhost/info.php>。应该看到图6-14。

如果一切都已正确安装, 则你应该看到作为服务器API行的值“CGI/FastCGI”, 如图6-14所示。

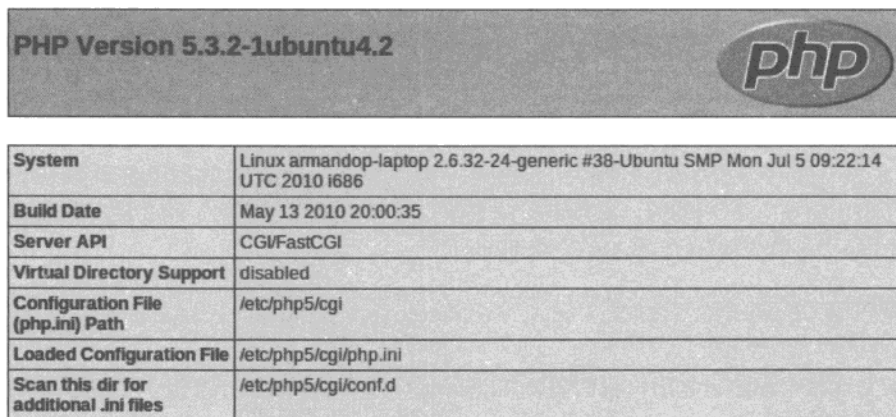


图6-14 安装了FastCGI PHP的PHP信息页

6.11.2 Nginx 基准测试

为了测试执行某个脚本的效果, 我们将使用本书前面已经测试过的代码, 如代码清单6-15所示。

代码清单6-15 要测试的PHP代码段

```
<?php
$max = 10000;
$x = 0;
$array = array();
```

```
while($x < $max)
{
    $array[$x] = $x;
    $x++;
}

foreach($array as $z)
{
    echo "$z<br/>";
}
```

该代码创建了一个包含10 000个元素的数组，并将这些元素显示在屏幕上。在Apache和Nginx上运行相同的ab命令产生的结果如图6-15和图6-16所示。

```
Server Software:      Apache/2.2.16
Server Hostname:      localhost
Server Port:          80

Document Path:        /test.php
Document Length:      88890 bytes

Concurrency Level:     500
Time taken for tests:   16.961 seconds
Complete requests:     1000
Failed requests:        0
Write errors:          0
Total transferred:     89081000 bytes
HTML transferred:     88890000 bytes
Requests per second:   58.96 [#/sec] (mean)
Time per request:      8480.315 [ms] (mean)
Time per request:      16.961 [ms] (mean, across all concurrent requests)
Transfer rate:         5129.12 [Kbytes/sec] received
```

图6-15 在Apache Web服务器上使用默认设置运行代码清单6-15的ab结果

```
Server Software:      nginx/0.7.67
Server Hostname:      localhost
Server Port:          80

Document Path:        /test.php
Document Length:      88890 bytes

Concurrency Level:     500
Time taken for tests:   9.152 seconds
Complete requests:     1000
Failed requests:        0
Write errors:          0
Total transferred:     89048000 bytes
HTML transferred:     88890000 bytes
Requests per second:   109.27 [#/sec] (mean)
Time per request:      4575.926 [ms] (mean)
Time per request:      9.152 [ms] (mean, across all concurrent requests)
Transfer rate:         9502.00 [Kbytes/sec] received
```

图6-16 在Nginx Web服务器上使用默认设置运行代码清单6-15的ab结果

图6-15所示的结果再次表明处理优化时我们需要查看两个重要项：每秒的请求数以及每个请求的时间。在这个结果中，Apache服务器每个请求的时间为8480.315毫秒（8.5秒）。这意味着用户需要等待将近10秒的时间才能得到服务器的响应，然后浏览器才开始显示数据。对比一下该数字与图6-16所示的Nginx结果，我们发现Nginx服务器每个请求的时间为4575.926毫秒（4.5秒），快了50%。

6.12 小结

本章进一步研究了PHP应用程序。我们了解了不同类型服务器（即prefork、threaded和event）的知识，你还知道了当以更快的速度向用户呈现内容时，为什么硬件非常重要。

本章还包含很多其他主题，如有用的Apache配置设置和命令以及如何去除Apache进程中的某些步骤以缩短加载时间。我们还略微谈到了其他两种最流行的Web服务器lighttpd和Nginx。你学会了如何安装、配置和运行几个ab测试，以测试这两个服务器是否能够满足你的应用程序的要求。

我们鼓励你使用本章提供的信息来确定哪个Web服务器程序包最适合你的特定应用程序。同时也鼓励你通过使用本章介绍的一些机制来实验和验证你所做出的选择是否正确。

应用程序的最重要组件之一就是托管它的Web服务器。无论你怎么优化应用程序，如果不同时优化Web服务器，那么你将无法实现全部的性能优化目标。

本章将介绍如何为Web服务创建“应用程序概貌”，以及如何利用它来确保Web服务器的正确配置。

我们将看一下如何通过去除不必要的文件操作来克服一些导致性能下降的常见问题。

有时，单独一台Web服务器可能无法提供所需的性能水平或吞吐量，因此你可能就要把应用程序部署到多个服务器上，以此来提高性能。我们将探讨应用程序需要满足哪些需求才能运行在一个由多台Apache服务器组成的集群上。

我们还将探讨可用来分割应用程序以获得较高性能级别的一些方法。

对于专业的托管应用程序来说，监视也是一个重要的需求。我们将看一下有哪些监视系统可用来帮助我们时刻监视系统的状态。

最后，我们将介绍一些可用于加快内容交付的基础架构选项和服务，并帮助你了解一下有哪些常见的缺陷以及如何克服它们。

注意 我们仍然在探讨Web服务器层，因此请看一下上一章的图6-1，回顾它在应用程序栈模型中的位置。

7.1 测定 Web 服务器的性能

在第1章中，我们介绍了如何使用基准测试工具来测定Web服务的性能。这种方法很适用于静态测试，你可以通过它来很好地了解服务运行的情况。负载测试并不适合在生产服务器上经常使用，而我们在决策之前却通常需要确定服务器在真实负载下的性能。这时，我们一般采用ApacheTop，它是我们用来检查Web服务器性能的主要工具。

使用实时访问日志文件分析器 ApacheTop

ApacheTop可以通过apt-get安装在Debian系统上，并且可以以.rpm形式安装在Red Hat/

CentOS/Fedora系统上。我们可以使用以下命令在基于Debian的系统（Ubuntu）上安装ApacheTop：

```
$sudo apt-get install apachetop
```

在Red Hat/CentOS/Fedora上的安装可以通过源文件进行，使用以下命令：

```
mkdir ~/maketemp
cd ~/maketemp
wget http://www.webta.org/apachetop/apachetop-0.12.6.tar.gz
sudo yum install readline-devel
sudo yum install ncurses-devel
sudo yum install pcre-devel
tar xvfz apachetop-0.12.6.tar.gz
cd apachetop-0.12.6
./configure
make
sudo make install
```

ApacheTop是一个实时访问的日志文件分析器，它的行为方式类似于对访问日志进行跟踪，但也提供了一定程度的分析。

通过指定Web服务器访问日志文件的路径来启动ApacheTop，如代码清单7-1所示。

代码清单7-1 ApacheTop输出

```
$sudo apachetop -f /var/log/apache2/access_log

last hit: 13:48:40      atop runtime: 0 days, 00:01:25      13:49:02
All:          68 reqs ( 1.2/sec)      1462.5K ( 25.7K/sec)      21.5K/req
2xx:         67 (98.5%) 3xx:          0 ( 0.0%) 4xx:          1 ( 1.5%) 5xx:          0 ( 0.0%)
R ( 30s):      15 reqs ( 0.5/sec)      288.9K ( 9859.7B/sec)      19.3K/req
2xx:          15 ( 100%) 3xx:          0 ( 0.0%) 4xx:          0 ( 0.0%) 5xx:          0 ( 0.0%)

REQS REQ/S    KB KB/S URL
1 0.05  4.7  0.2 */tags/view/tag/Lincoln+Smart+Dodge+Lotus+Subaru
1 0.05 20.9  0.9 /media/4c7b73ba0dc48dc965820300
1 0.05 27.7  1.3 /media/4c7b74f90dc48d7668200000
1 0.05 35.0  1.6 /media/4c7b72680dc48dc965770200
1 0.05 20.2  0.9 /media/4c7b78270dc48d7668240200
1 0.05 25.1  1.1 /media/4c7b778d0dc48d7668b10100
1 0.03 17.5  0.6 /media/4c7b6de30dc48dab64840000
1 0.03 29.3  1.0 /media/4c7b6c6d0dc48d0e5f370500
1 0.03 25.3  0.8 /media/4c7b6dbd0dc48dab64670000
1 0.03 16.1  0.5 /media/4c7b6e240dc48dab64a20000
1 0.03 31.6  1.1 /media/4c7b6e690dc48dab64d20000
1 0.03 24.9  0.9 /media/4c7b6e8f0dc48dab64ec0000
1 0.03  5.6  0.2 /
1 0.04  2.4  0.1 /feed/top
1 0.04  2.6  0.1 /tags/top
```

ApacheTop将显示自从它启动以来所累积的所有2xx、3xx、4xx和5xx状态码的请求率，还将提供这些状态码的前30秒的统计数据，你可以通过这些信息来确定负载是正在上升还是下降。它还详细列出了被点击得最多的URL以及它们的点击率。该信息非常有用，因为它可以帮助你确定

应该首先优化哪些页面。

可以使用 `-T n` 命令行选项更改R显示中ApacheTop累积结果的周期。这样便可以将周期从默认的30秒更改为较长的时间（若Web服务器流量较低）。这样做的目的是显示出有代表性的流量，以便可以确定你的网站上单击排名前十的URL以及它们的点击率。

ApacheTop完成加载之后，可以使用“?”键显示命令列表，这些命令有很多用处，包括修改显示、从URL切换到引用链接、放大以及检查某个URL的详细统计信息。

过滤机制可以帮助我们确定网站中点击率最高的网页的分布情况，它是最有用的特性之一。例如，如果你键入“f”、“a”、“u”（表示filter=>add=>url），然后键入“.php”，则ApacheTop会将显示的URL限制为以“.php”结尾的URL。

建议你花一些时间来熟悉ApacheTop，以便了解应用程序的概貌并弄清点击率最高的页面，这样便可以将优化重点放在那些带来最大利益的网页上。你应该通过使用情况了解使用率最高的十大页面。确保你已经知道了这些页面以及它们所占的使用率百分比。

例如，你可能发现获得的内容类似于表7-1所示。

表7-1 示例应用程序按请求数量排名前十的URL

页 面	请求数量	百分比（四舍五入）
/（主页）	2000	33
/news.php	1500	25
/blog.php	1000	17
/video.php	600	10
/topuser.php	400	6
/message.php	150	2
/message_send.php	100	2
/message_read.php	90	2
/user_profile.php	80	2
/feedback.php	15	1
总计	5935	100

这种类型的对应关系将告诉你应用程序中所有“操作”的位置，并且让你知道需要关注的位置。

7.2 了解应用程序的内存占用情况

使用ApacheTop，我们可以了解应用程序的“概貌”。现在我们需要弄清什么是内存占用（memory footprint），以便可以确保在Web服务器中安装有足够的内存，或者配置服务器以避免产生磁盘交换（swapping）。

如果应用程序中包含常见的页脚，则可以添加注释，以便直观地查看之前标识的每个点击率较高的页面的峰值内存使用情况。

添加如下内容：

```
<!-- Memory usage: <?php echo memory_get_peak_usage(1); ?> -->
```

这将在每个页面的结尾处放置一个注释，从而显示出在创建该页面时占用了多少内存。查看每个被标识为点击率高的页面，然后使用“查看源代码”来得到峰值内存使用情况并把这个值记录下来。加权平均值是用百分比（用因子表示，例如33% => 0.33）乘以请求内存使用，如表7-2所示。

表7-2 向十大URL中添加内存占用和权重

页 面	请 求 数	百分比（四舍五入）	每个请求占用的内存	内存的加权平均值
/（主页）	2000	33	10MB	3.3MB
/news.php	1500	25	15MB	3.75MB
/blog.php	1000	17	16MB	2.72MB
/video.php	600	10	8MB	0.8MB
/topuser.php	400	6	17MB	1.02MB
/message.php	150	2	9MB	0.18MB
/message_send.php	100	2	6MB	0.12MB
/message_read.php	90	2	8MB	0.16MB
/user_profile.php	80	2	14MB	0.28MB
/feedback.php	15	1	3MB	0.03MB
总计	5935	100	106MB	12.36MB

这告诉我们，对于正常的流量混合来说，每个请求的加权平均内存消耗为12.36MB。因此，如果拥有1GB内存的Web服务器，并且操作系统及其他内容已预留了200MB，那么剩下用于应用程序的内存数量将为800MB。将该值除以加权平均值得到 $800/12.36 = 64.72$ 。这就是可以同时处理而不会耗尽内存以及导致系统产生磁盘交换的十大并发请求的内存数量。

应该注意的是，我们在此使用了前10个请求作为一般经验法则。如果发现在表的末尾URL的请求百分比并未减少到一个较小的数字（比如说1%），那么就可能需要扩大该表来包含更多的URL。当ApacheTop创建按请求排序的视图时，它会自动忽略URL后面的任何查询字符串参数，因此由查询字符串所导致的URL变化并不会产生大量额外的项。我们的目的不是对系统中所有可能的URL进行极为精确的评估（在大型网站上这可能会运行好几百页），而只是充分覆盖主要路径即可。

7.3 优化 Apache 中的进程

在第6章中，我们介绍了Apache Web服务器如何处理请求，以及如何使用MPM将请求发送给应用程序。

这一节将介绍如何优化Prefork MPM（最常见的MPM）的设置，以确保它不会过度使用内存以及产生磁盘交换。

7.3.1 控制 Apache 客户端 (Prefork MPM)

上述Prefork MPM的请求处理行为是通过一系列配置指令来控制的。表7-3显示了Apache的Ubuntu分发版的默认值；其他分发版可能有它们自己的默认值。

大多数分发版都是在主配置文件httpd.conf中设置这些值；但某些分发版（如Ubuntu）会将这些值放置在一个单独的文件中，该文件位于Apache配置文件夹的“extra”子目录中。例如，在Ubuntu上，这些值位于/etc/apache2/conf/extra/httpd-mpm.conf中。你可能还需要恢复httpd.conf中的一个原先被注释掉的包含文件行，以便加载它们。

表7-3 用于控制Prefork MPM行为的Apache指令

指 令	说 明	默 认 值
StartServers	该指令控制Apache启动时将生成的客户端数。由于创建客户端有开销，因此最好是拥有足够的客户端来处理空闲（idle-level）的流量	5
MinSpareServers	正常情况下，如果请求数减少并且Apache无法提供拥有这些请求的正当理由，则Apache会终止一些客户端。该指令对Apache将保持活动的客户端数设置了一个较低的限制。它不应低于StartServers	5
MaxSpareServers	该指令设置Apache开始放弃客户端的点。如果活动客户端数比当前正在处理的并发请求数多10个以上，那么Apache将开始终止和放弃客户端，直到这个数量达到MinSpareServers值。该值设置得太低，会导致Apache封杀客户端进程，因此小心使用此设置	10
MaxClients	MaxClients指令设置Apache将产生的最大子进程数，从而设置它可以处理的最大同时或并发请求数	150
MaxRequestsPerChild	该指令设置子进程在被终止和重新产生之前将处理的最大请求数。如果设置为0，则该进程是永久存在的，永远不会死亡	0

7.3.2 优化内存使用和防止产生交换

现在，你已经了解了设置MPM配置所需的全部信息。从之前的章节中，我们了解到对于实际情况中使用的混合页面而言，有着1GB内存的服务器最多可以支持65个并发请求。

我们应该更改的第一个指令是MaxClients，将其设置为65以便不会过度使用内存。

为了确保在负载上运行时不会封杀客户端，大概应该将StartServers设置为65的一半，即30，同时也将MinSpareServers设置为30。最后，将MaxSpareServers设置为40，这是一个适当的值，可防止封杀客户端。

7.4 其他 Apache 配置调整

Apache Web服务器有很多配置选项，用于控制其行为的每个方面。Apache的默认出厂配置的目的是提供一个“开包即用”的方便配置，但在分发版配置文件中的很多默认值可能会降低性能，如果你不需要某个特殊功能，则可以避免使用它。

最好先了解一下这些用于处理请求的“方便的功能”到底有多少，以便可以确定它们对应用程序性能的影响，以及是否应该避免使用这些功能。

7.4.1 使用.htaccess 文件和 AllowOverride

在第3章，我们介绍了require_once函数的使用（它引入了对操作系统“lstat”函数的额外调用，因此减慢了页面的呈现速度）。而启用“AllowOverride”指令（它允许使用.htaccess文件）也存在类似的开销。

.htaccess文件是Apache主配置文件的补充文件，它的作用相当于为每个请求打个“补丁”，该文件可以放置在应用程序的任何目录中，用于为存储在该位置及其子目录中的内容建立自定义配置。

“AllowOverride”指示Apache检查包含有要处理的脚本或文件的目录及其每个父目录，看看是否存在影响这个当前请求的其他Apache配置指令的“.htaccess”文件。但如果已启用“AllowOverride”，则即使没有使用.htaccess文件，也仍然要进行该检查，目的是确定是否存在.htaccess文件，因为这会导致多个操作系统调用的开销。

如果正在使用.htaccess文件，请考虑将配置指令移动到Apache的主配置文件中，主配置文件仅在HTTP服务器启动时或一个新的HTTPD客户端被创建时加载一次（而不是每个请求都加载）。如果需要为不同的目录保留不同的指令，则要考虑把它们放在<Directory ...> ... </Directory>标记中，以保留控制指定目录的功能。

如果正在使用一些限定形式的共享主机，并且没有对整个Apache配置文件的访问权限，那么可能必须使用.htaccess文件。但一般来说，为了最大限度地提高性能，应该避免使用该文件和配置指令。毫无疑问，你应该尽力关闭这些指令，以确保实现最大化的性能。

在下面的代码清单中，我们创建了一个简单的映射到www.static.local的静态服务器vhost，并在dir1/dir2/dir3的文档根中创建了一个三级深的路径。在最深的目录中，我们放置了一个名为pic.jpg的文件，大小大约为6KB。代码清单7-2显示了将AllowOverride选项设置为“None”时，系统在siege下的性能。代码清单7-3显示了在AllowOverride选项设置为“All”时，相同测试的结果。

代码清单7-2 禁用AllowOverride指令时处理静态对象

```
$siege -c 300 -t 30S http://www.static.local/dir1/dir2/dir3/pic.jpg
.....
Lifting the server siege      done.
Transactions:                15108 hits
Availability:                100.00 %
Elapsed time:                29.66 secs
Data transferred:           100.01 MB
Response time:               0.00 secs
Transaction rate:            509.37 trans/sec
Throughput:                  3.37 MB/sec
Concurrency:                 12.99
```

```

Successful transactions:      15108
Failed transactions:          0
Longest transaction:         0.14
Shortest transaction:         0.00

```

代码清单7-3 启用AllowOverride指令时处理静态对象

```

$ siege -c 300 -t 30S http://www.static.local/dir1/dir2/dir3/pic.jpg
.....
Lifting the server siege...      done.
Transactions:                   14440 hits
Availability:                   100.00 %
Elapsed time:                   29.67 secs
Data transferred:               95.58 MB
Response time:                  0.02 secs
Transaction rate:               486.69 trans/sec
Throughput:                     3.22 MB/sec
Concurrency:                   11.87
Successful transactions:        14440
Failed transactions:            0
Longest transaction:            1.06
Shortest transaction:           0.00

```

这些结果表明，在处理静态对象时，关闭该选项与启用该选项的性能差异约为5%。

7.4.2 使用 FollowSymlinks

与刚刚所描述的AllowOverride指令一样，FollowSymlinks选项也需要额外的操作系统调用，目的是确定是否存在symlink。当不需要时应该禁用它，这在性能方面会有一点提高。

7.4.3 使用 DirectoryIndex

另一个可能会在无意间产生操作系统调用的地方是DirectoryIndex指令。当请求的URL指向一个目录而非某一具体文件时，该指令指定了一个默认文件列表（采用空格分隔的格式）。Apache按照在该指令中指定的顺序搜索默认文件。一定要把使用最频繁的应用程序的名称放在此列表中的第一个。例如，如果是PHP应用程序，该选项应如下设置：

```
DirectoryIndex index.php index.html
```

如果文件顺序错误，则Web服务器会对目录的每个请求执行对index.html的不必要的搜索。对于主页来说，这非常重要，因为大部分流量访问的都是主页，并且不可避免地会间接引用index.php。

7.4.4 关闭 HostnameLookup

我们在本书的前面部分介绍了DNS查找。DNS查找将获取域名并查找其映射的IP。每次IP地址不存在时都会发生该过程，并且这个检查会增加延迟。

大部分Apache分发版默认都会关闭此功能，而如果未关闭，则可能会产生非常不利的影响。为了关闭此功能，我们需要对配置文件的HostnameLookup键进行更改。该指令可能已经设置为“关闭”，但如果不是这样，则将其更改为“关闭”并重新启动服务器。

7.4.5 启用 Keep-Alive

Keep-Alive的作用是使Web服务器支持永久连接。启用Keep-Alive指令，Apache可以支持每个TCP连接的多个HTTP请求。这是一个需要设置的重要指令，因为如果启用Keep-Alive，则Apache会在打开和关闭连接时不使用内存。去除此开销，我们会再次提高应用程序的速度。

若要启用Keep-Alive，请打开配置文件并找到Keep-Alive指令。在某些情况下，该指令已设置为“启用”。如果没有启用，只需将其值设置为“启用”，保存更改并重新启动Apache。

7.4.6 使用 mod_deflate 来压缩内容

HTTP协议允许使用压缩传输编码。这样不仅仅会加快可压缩文件（如html、js或css文件）的传输，而且还会减少传输应用程序所占用的带宽。如果你的流量非常大并且需要为出站带宽付费，则该功能可以帮助你减少费用。

mod_deflate是Apache 2.x服务器附带的一个标准模块，该模块非常容易设置和使用。若要启用该模块，请确保在你的Apache配置文件中恢复下面这行被注释掉的内容。请注意，具体路径可能与此处所显示的路径有所不同，但原理是一样的。

```
LoadModule deflate_module /usr/lib/apache2/modules/mod_deflate.so
```

如果使用的是基于Debian的分发版（如Ubuntu），有一个不需要编辑配置文件即可启用此模块的机制，就是使用以下命令来启用它。

```
$sudo a2enmod deflate
```

然后，重新启动Apache服务器来加载该模块。如果想要压缩从服务器发往浏览器的所有支持压缩的文本、HTML或XML，可向vhost配置中添加以下指令。

```
AddOutputFilterByType DEFLATE text/html text/plain text/xml
```

但是有一点，一些比较老的浏览器声明支持压缩传输，但却破坏了对标准的支持，因此以下指令将阻止mod_deflate对发送到这些有问题的客户端的文件的压缩。

```
BrowserMatch ^Mozilla/4 gzip-only-text/html
BrowserMatch ^Mozilla/4\.0[678] no-gzip
BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
```

若要测试压缩是否正常工作，请重新启动服务器，使用Firefox和Firebug访问主页，并使用Net面板进行检查，该面板就是使用gzip内容编码传输的主页PHP生成的HTML。

图7-1显示了配置mod_deflate和访问一个URL（返回一个文本或HTML文件）之后的Firebug Net面板。响应头中的“Content-Encoding”字段显示该内容确实是已被压缩了。

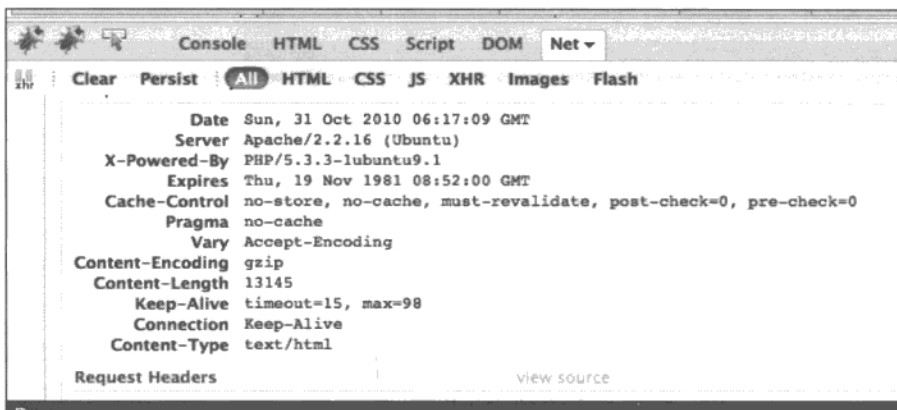


图7-1 显示gzip的Content-Encoding值的Firebug

7.5 扩展到单台服务器之外

无论你对应用程序或系统配置应用了多少优化，如果你的应用程序是成功的，你就需要对它进行扩展，因为单台计算机的能力无法再满足你的需要。为了在分布式模式下运行，应用程序必须满足很多“要求”。重新设计你的应用程序以便在Web服务器“场”中运行，乍看起来这个想法似乎有点吓人，但幸运的是PHP和LAMP栈中的组件提供了大量的分布式支持。

在这一节中，你将了解其中一些需求以及如何简单轻松地实现它们。

7.5.1 使用 Round-Robin DNS

在多台Web服务器之间分布流量的最简单方法是使用“round-robin DNS”。这需要为服务器集群中的每台Web服务器的主机名设置一个“A”记录。DNS服务将按照随机的顺序将地址列表发送到每个客户端，从而允许将请求在服务器场中的所有成员之间进行简单分布。

这种机制的优点是它不需要任何其他硬件或Web系统的配置。其缺点如下所示。

- ❑ 如果其中一个Web服务器出现故障，仍然会有流量发给它。没有一种机制可以检测发生故障的服务器并将流量路由到其他服务器。
- ❑ 任何系统配置的更改要想“复制”到整个DNS系统，将花费大量时间。如果你想添加或删除服务器，则这些更改可能需要多达3天的时间才能完全生效。

7.5.2 使用负载均衡器

负载均衡器是一个在以群集或场形式运行的一组服务器之间分布请求的设备。它的作用是使得从用户浏览器的角度来看，服务器场就像是单一一台服务器一样。

图7-2显示了使用负载均衡器聚合多台Web服务器性能的系统的典型布局。

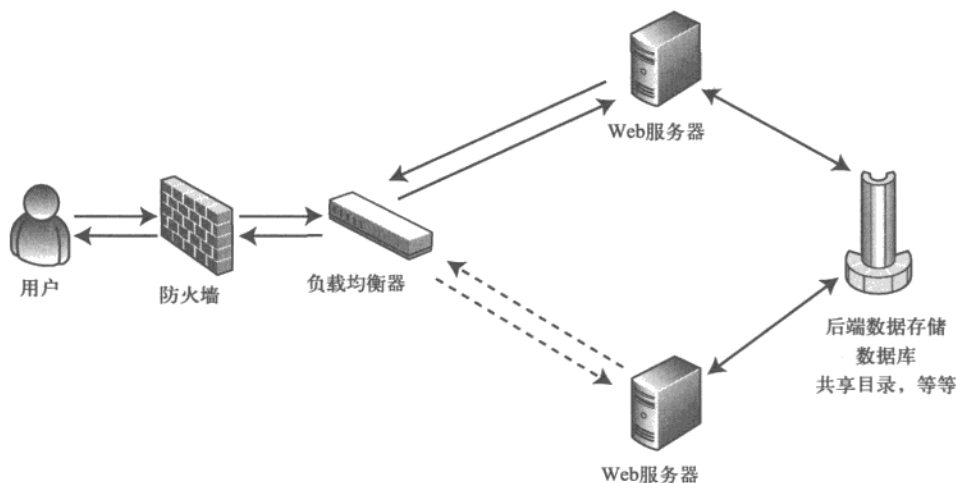


图7-2 典型的分布式Web应用程序

负载均衡器的种类有很多，有基于硬件的，也有基于软件的。一般来说，负载均衡器分为四类。

- ❑ 完全基于软件的解决方案：这些解决方案，如Linux虚拟服务器项目（www.linuxvirtualserver.org），可以直接在Web场的每台服务器上运行一个负载均衡服务。这种解决方案的缺点是，它需要对每台Web服务器上的网络接口进行自定义配置，因此它无法用于很多基于云或主机托管的解决方案。
- ❑ 使用一个单独的负载均衡服务器的软件解决方案：你可以在单独的计算机（位于Web场“前面”）上运行一个软件负载均衡器。与mod_proxy一起运行的产品（如HAProxy、Squid和Apache）允许你构建自己的负载均衡设备。
- ❑ 物理的负载均衡设备：拥有自己的负载均衡设备的另一种方法是使用F5、Coyote Point、Cisco以及Juniper提供的商用设备。这些设备通常提供很多其他功能，如缓存、SSL终端以及I/O优化。
- ❑ 负载均衡服务：很多基于云的解决方案都提供负载均衡服务，这些服务可以将单个IP地址映射到多台Web服务器。Amazon Elastic Load Balancer就是一个服务，它支持在Amazon EC2云中托管的实例之间进行负载均衡。

负载均衡器可以提供比上述的简单的round-robin DNS解决方案更复杂的负载分布。通常，它们都提供以下分布方法。

- ❑ Round-robin：类似于DNS分布方法。
- ❑ 最少连接：使用最少的活动连接数将请求发送至Web服务器。
- ❑ 最少负载：很多负载均衡器都提供一个询问Web服务器的机制，用于确定其当前负载，并且将新的请求分布到负载最少的服务器。

□ 最少延迟：负载均衡器使用一个移动平均线监视器（moving average monitor）来监视请求，将请求发送至显示响应最快的服务器。这是一种不直接轮询服务器而确定负载的方法。

□ 随机：负载均衡器将请求路由到随机的服务器。

此外，负载均衡器将监视Web服务器，看是否存在没有对请求做出响应的服务器，或者对指向它们的状态或负载监视请求没有做出适当响应的服务器，然后将这些服务器标记为“已宕机”并停止将请求路由给它们。

很多商用和开源负载均衡器通常支持的另一个功能就是“黏滞会话（Sticky Sessions）”。负载均衡器将尝试尽可能用同一服务器来满足一个用户的请求，目的是减少计算机之间交换会话状态信息的需要。但是，你应该注意，在高负载的情况下，使用黏滞会话可能会导致负载分布不均。

甚至在负载峰值超过整个Web服务器场的容量时，负载均衡器也可以提供帮助。负载均衡器通常具有定义“溢出”服务器的功能。服务器场中的每台服务器都可以设置一个最大的连接数，当所有服务器的所有连接都在使用中时，可以将其他请求路由到溢出服务器上的固定页面。

溢出服务器可以提供一个信息页，该页面告诉用户该服务现在正处于峰值负载并且请用户稍后返回，例如，如果网站提供新闻服务，可以把溢出服务器的信息页设置为显示五个头条新闻的简单HTML。这样便可以应对诸如9/11或迈克尔·杰克逊逝世之类的情况，在此类情况下，由于公众对信息的巨大需求，大多数新闻服务都人满为患。这时用静态HTML新闻页面可以分担来自静态服务器的很大一部分连接。

还可以使用溢出服务器来托管“网站维护”页面，当整个服务器场由于更新或维护而整体离线时，可以把这个维护画面显示给用户。

7.5.3 使用直接服务器返回

随着流量的增加以及你向Web服务器场中添加越来越多的服务器，可能会出现另一个性能问题，那就是网页的呈现速度会受到限制。在刚刚所描述的简单分布式Web应用程序中，来自用户浏览器的请求以及来自Web服务器的响应都必须经过负载均衡器解决方案的处理。Linux虚拟服务器解决方案由于其设计的性质而不受此限制，但大多数解决方案都有此限制。

现在人们已经研究出一种称为直接服务器返回（direct server return, DSR）的技术，该技术绕过Web服务器响应的负载均衡系统，从Web服务器将响应直接写入用户的浏览器。这属于一个网络级的操作，因此你的应用程序不会感知到有什么差别。但这意味着负载均衡器只处理请求，这些请求大部分比响应小。

图7-3显示了当服务器场被配置为使用直接服务器返回（DSR）时的数据流。

此外，像Linux虚拟服务器解决方案一样，这种解决方案需要对Web服务器的网络接口进行特殊配置，并且需要来自负载均衡器的特定支持，以便直接传递信息从而实现直接返回连接。因此，它不适合基于云的或虚拟主机的解决方案，这些解决方案通常只有有限的机会来更改其与硬件接口的方式。

由于DSR的配置对于每个负载均衡解决方案和供应商来说区别很大，因此在本书的范围内将不介绍设置和配置。如果你觉得它对你有用，可以咨询你的主机提供商。

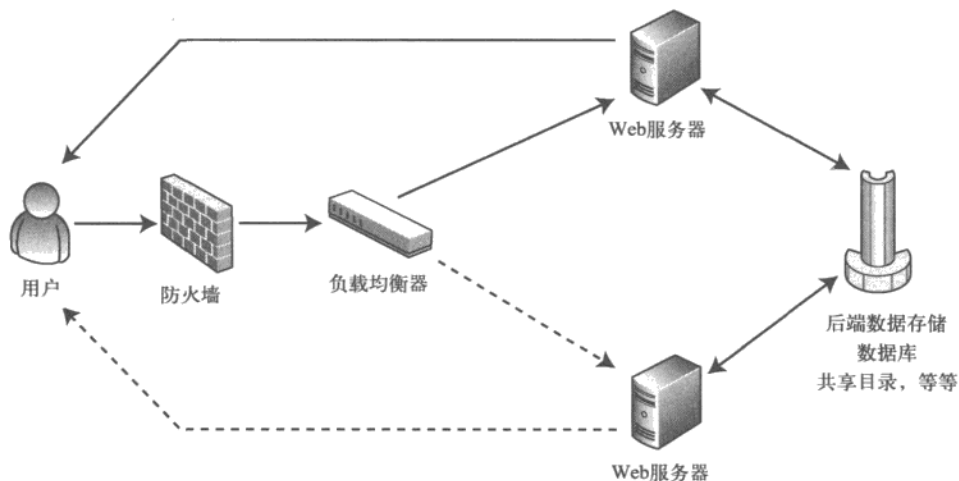


图7-3 使用直接服务器返回的分布式Web应用程序

7.5.4 在服务器场的成员之间共享会话

简单的静态网站不需要会话，也就不需要采取特殊操作来确保会话在服务器场中的所有计算机之间分布正确。但是，如果你的网站支持各种登录行为，则需要维护会话，并且你需要确保这些会话在服务器场中的正确共享。

默认情况下，PHP使用基于文件的会话存储来建立其会话。Web服务器本地磁盘上的目录用于存储序列化的会话数据，cookie（默认值为PHPSESSID）用于保持客户端浏览器和文件中会话数据之间的关联。

当你分布你的应用程序时，必须确保所有Web服务器都能访问每个用户的同一会话数据。实现该操作的方法主要有三种。

(1) Memcache：使用一个共享的 Memcache 实例来存储会话数据。当你使用PECL来安装Memcache扩展时，系统将提示你是否希望安装会话支持。如果安装，则系统允许你将`session.save_handler`设置为“Memcache”并且将保持共享状态。

(2) 共享目录中的文件：你可以使用基于文件的会话状态存储（`session.save_handler="files"`），只要你确保将`session.save_path`设置为在所有计算机之间共享的目录即可。通常，在这些情况下使用NFS来共享文件夹。

(3) 数据库：你可以将会话ID用作唯一键值来创建一个用户会话处理程序，用于序列化往返于后端数据库服务器的数据。

使用特定的共享策略之前，你需要检查一下，在你的PHP版本中是否支持该方法。使用`phpinfo`列出可以在你的安装上使用的会话扩展的详细信息。

进行检查，以确保为你选择的方法安装了合适的“注册保存处理程序”（Register Save

Handler)。图7-4显示了在你的phpinfo页面中应该出现的内容，你的Memcache扩展是否安装正确以及Memcache保存处理程序是否已正确注册。

session

Session Support	enabled	
Registered save handlers	files user memcache sqlite	
Registered serializer handlers	php php_binary wddx	

Directive	Local Value	Master Value
session.auto_start	Off	Off
session.bug_compet_42	Off	Off
session.bug_compet_warn	Off	Off
session.cache_expire	180	180
session.cache_limiter	nocache	nocache
session.cookie_domain	no value	no value
session.cookie_httponly	Off	Off
session.cookie_lifetime	0	0
session.cookie_path	/	/
session.cookie_secure	Off	Off
session.entropy_file	no value	no value
session.entropy_length	0	0
session.gc_divisor	1000	1000
session.gc_maxlifetime	1440	1440
session.gc_probability	1	1
session.hash_bits_per_character	5	5
session.hash_function	0	0
session.name	PHPSESSID	PHPSESSID
session.referer_check	no value	no value
session.save_handler	files	files
session.save_path	no value	no value
session.serialize_handler	php	php
session.use_cookies	On	On
session.use_only_cookies	On	On
session.use_trans_sid	0	0

图7-4 phpinfo中的会话扩展部分

7.5.5 与共享文件系统共享资产

除了组成应用程序的PHP文件之外，你通常还需要处理其他资产，如图像、视频、.css文件以及.js文件。尽管你可以向每台Web服务器部署任何固定的资产，但是如果你支持用户生成的内容并允许用户上传视频、图像以及其他资产，则你必须确保这些内容可用于所有Web服务器。完成此操作的最简单方法是在所有Web服务器之间保留一个共享的目录结构，并将用户内容存储映射到该目录。与共享会话文件的情况一样，你可以使用NFS在计算机之间共享挂载点 (mount point)。

在诸如Amazon EC2这样的服务中，你可以使用一个外部的S3 bucket来存储固定的资产以及用户贡献的资产。由于S3支持通过一个简单的URL来引用存储文件，因此S3 bucket还可以用来处理文件，从而减轻Web服务器的负担。

7.5.6 与独立资产服务器共享资产

处理共享资产的另一个策略是将共享资产放到专为处理静态文件而优化的独立系统上。尽管 Apache 是一个非常出色的综合 Web 处理解决方案，但它的灵活性和复杂性意味着它通常不是一个高性能的静态内容分发最佳解决方案。一般情况下，其他解决方案（如 lighttpd 和 Nginx）通常能以更高的速度提供静态内容。在第 6 章，我们看到了处理静态内容时 lighttpd 和 Nginx 是多么高效。

7.5.7 与内容分发网络共享资产

内容分发网络（CDN）是按层次结构分布的缓存代理服务器网络，它内置有地理负载均衡功能。主要目的是将不经常更改的文件缓存到尽可能接近用户浏览器的计算机中。为此，每个网络都维护了一个巨大的缓存服务器网络，这些服务器分布在互联网周围的几个关键点。

地理 DNS 系统查找离你的网站用户最近的缓存服务器，获取并缓存静态资产的副本，以供你的用户使用。然后，系统将使用最近的缓存副本来满足对该资产的后续请求，而不会将请求一直发送回你的 Web 服务器。通过从距离你的用户最近的 CDN 缓存服务器处理这些请求，可以大大缩短呈现页面的时间。

图 7-5 显示了一个 CDN 如何缓存靠近用户内容的简化图表。你可以控制对网站的哪些部分进行缓存，以及哪些部分直接通过你的系统来传递。

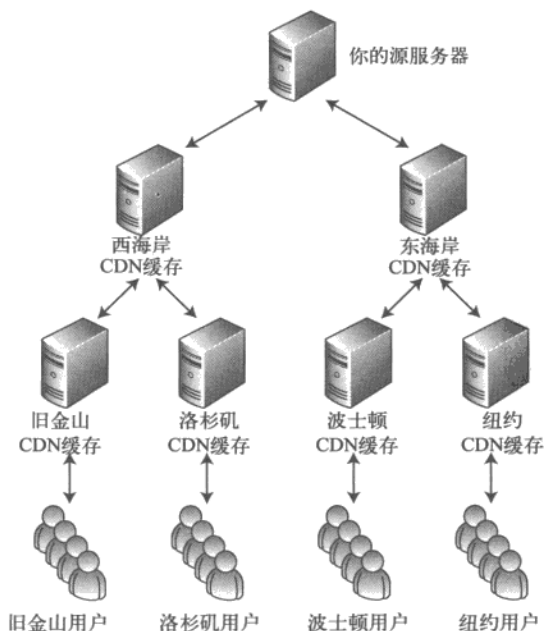


图 7-5 通过 CDN 进行地理分发的示例

一些典型的CDN系统包括以下内容。

- ❑ Akamai: 最知名、最全面的CDN解决方案之一, 实际上由于其成本, 它并不适合中小型网站。
- ❑ CD Networks: 与Akamai一样, 该内容分发网络是专为大型部署设计的。
- ❑ Limelight: 另一个知名的CDN系统, Limelight还提供资产的远程存储和分发。
- ❑ Amazon CloudFront: 一个简单的CDN, 它与Amazon EC2/S3进行了集成, 它最著名的特点是无需签订合同, 随时付款即可使用, 但不如之前提到的解决方案使用得广泛。

7.6 使用分布式架构的陷阱

在多台服务器之间分布你的应用程序可能会导致一些问题, 在计划时你应该注意这些问题。下面我们详细介绍一下可能发生的一些最常见的问题。

7.6.1 缓存一致性问题

通常, 在很多应用程序中都需要保持应用程序级的缓存, 例如, 缓存RSS源。如果缓存机制不是在Web服务器场的所有成员之间共享, 那么你可能会看到一些缓存一致性的影响。

如果你使用某个共享缓存机制, 例如Memcache (每个成员都彼此连接), 那么你将不会受到任何影响。但是, 如果你的缓存机制使用每个Web服务器上的本地资源, 如本地文件系统或APC缓存, 那么缓存在每个计算机中的数据可能不会同步。

这可能会导致当用户从一个服务器切换到另一个服务器时呈现给用户的视图不一致。当用户刷新主页时, 可能会连接到不同服务器上缓存的RSS源, 而这些RSS源将处于不同的状态。

你应该尽可能在Web服务器场上使用共享缓存机制, 或者确保缓存在本地缓存中的数据具有较长的数据持久性, 以便把影响减至最低。

7.6.2 缓存版本问题

如果你使用CDN来分发和缓存静态资产或用户生成的资产, 那么你需要确保在你更改分发的任何文件的内容时, 要么更改文件名称, 要么发出一条相应的命令来刷新CDN的旧版本文件。如果你不这样做, 那么当你发布新版本的应用程序时, 用户将看到你的新页面设计, 但图像、.js文件或.css文件却是旧的。

减少这些问题的另一个常用方法是用版本号来命名资产(例如, /assets/v5/img/logo.jpg), 并且每次发布时递增版本号。你无需获得每个版本的单独副本。简单的重写规则将使Apache忽略此差异, 但将迫使CDN重新缓存资产。

使用下面所示的mod_rewrite规则, 可以令你的Web服务器忽略URL的版本元素。

```
RewriteEngine on
RewriteRule ^/assets/v[.A-Za-z0-9_-]+/(.*) /assets/$1 [PT]
```

现在, 对以下URL的任何请求将访问/assets/img/logo.jpg上的相同资产。

```
/assets/v1/img/logo.jpg  
/assets/vtesting/img/logo.jpg  
/assets/v1234/img/logo.jpg
```

若要在你的代码中使用此规则，只需使用每次发布时递增的全局版本号来生成你的全部资产 URL，如下所示：

```

```

如果你正在使用版本控制系统（如子版本）来管理你的代码，那么你甚至可以考虑使用应用程序资源库的版本号作为版本号。

另一个实现资产版本控制的常用方法是使用基于版本号的查询字符串，也就是说，在资产文件引用的结尾添加“?nnn”。但是，该方法并不适合所有 CDN 系统，尤其是 CloudFront，它会忽略 URI 上的查询字符串。

例如，使用查询字符串方法，你可以通过插入以下命令来创建版本化的徽标引用。使用这种方法，你无需使用重写规则，但不保证该方法适合所有 CDN 系统。

```

```

7.6.3 用户 IP 地址跟踪

使用分布式服务器场（在一组服务器前面有一个负载均衡器）的一个风险是每个 Web 服务器所“看到”的 IP 地址（Web 服务器将这个 IP 地址看作是请求的来源）并不是客户端浏览器的 IP 地址，而是负载均衡系统的地址。这可能会有一些不利之处。

我们应该注意到，很多内容分发网络都有同样的问题，因为这些网络都屏蔽了客户端浏览器的 IP 地址。这些情况下的解决方案将因你使用的网络而异。

可能遇到如下的问题。

- 如果你使用日志文件分析器来为你的市场营销或产品管理团队提供统计数据，那么该日志文件分析器可能会由于缺少客户端 IP 地址而发生混淆，并且可能无法计算单独用户的正确数量以及访问你的网站的正确次数。大多数负载均衡器和代理都可以这样配置：在它们传递给 Web 服务器的请求中插入一个“X-Forwarded-For”标头，该标头包含用户浏览器的真实 IP 地址。之后可以将 Web 服务器配置为使用该值，而不是其日志文件中的正常 IP 地址，从而恢复统计系统辨别单独用户的能力。
- 如果你使用防火墙规则或其他特定于应用程序的机制用来拦截或者跟踪每个 Web 服务器中的 IP 地址，那么它们可能会再次由于缺少直接的源 IP 地址而发生混淆。尽管基于应用程序的机制可以使用类似于上面所述的方法来获取正确的 IP 地址，但防火墙规则（例如地址阻止）通常无法使用 CDN 或负载均衡器所发送的数据，因为它们是在网络层进行操作的，因此不了解 HTTP 标头。幸运的是，大多数解决方案都具有在 Web 服务器外定义规则的能力，也就是在服务或设备本身中定义，但是所使用的方法却是因解决方案而异的，这些内容不在本书讨论范围之内。

对于插入了“X-Forwarded-For”标头的情况，你可以更改一下标准的Apache访问日志的格式，把该标头包含进来，以取代网络IP地址，具体的方法是在虚拟主机描述中使用下面的定义。把它放在定义日志文件的指令前面。

```
LogFormat "%(X-Forwarded-For)i %l %u %t \"%r\" %s %b \"%{Referer}i\" \"%{User-agent}i\""
forwarded
CustomLog "logs/mysite-access_log" forwarded
```

如果你希望你的所有主机都使用此格式而无需每个主机分别声明，那么请将该定义添加到httpd.conf文件中，直接放在你定义的虚拟主机前面。

```
LogFormat "%(X-Forwarded-For)i %l %u %t \"%r\" %s %b \"%{Referer}i\" \"%{User-agent}i\""
combined
```

7.6.4 多米诺骨牌或级联失败效应

当你使用Web服务器场时，你必须密切注意服务器上的负载因子。如果由于你的疏忽，没有正确地增加服务器来满足需要支持的负载量，那么可能会发生所谓的“多米诺骨牌失败效应”。

假定你拥有一个由两个服务器组成的Web服务器场，基于负载平均值和并发请求等，每个服务器的负载为其容量的60%。

这两台服务器中的一台服务器发生故障将导致该服务器上的整个负载转移到另一台服务器。这将导致后者试图去处理相当其总能力120%的负载，从而引发它也会发生失败。多米诺骨牌效应就是在这里开始的。

当你设计一个Web场时，必须确保它可以容忍一个或多个计算机发生故障，具体情况取决于场的大小。如果使用两台计算机，那么必须监视负载因子。如果负载超过50%，那么应该计划向该场中添加额外的计算机。你必须做好计划，估计出最多有多少台机器可能会同时发生失败，然后确保服务器场中有足够的容量可用来支持这些失败机器转移出来的负载，否则你很容易陷入此失败模式。

还要记住，你不仅仅需要考虑到失败的情况。使用多个计算机的最大好处之一是通过依次使每个计算机脱机，你有机会执行滚动升级或维护。

7.6.5 部署失败

如果你只有一台服务器，那么部署失败的话会马上显示出来——你的网站或服务不再工作。但是如果有一个Web场，那么可能由于某种原因某台计算机的部署或更新不工作，除非你在任何更改之后测试每台计算机，否则你可能无法立即察觉到哪台计算机不工作。

当从负载均衡器查看时，特别是当负载均衡器开启了“Keep-Alive”并把连接路由到同一台服务器时，你的服务看起来工作正常。但其他用户可能会看到随机失败或永久失败，具体情况取决于用户连接到的服务器。因此一定要确保你的部署过程包含一个单独检查每个服务器的步骤。为此，最好是拥有一个映射到每个服务器（例如www1.example.com、www2.example.com）的DNS项，以便可以执行此验证步骤。

7.7 监视应用程序

部署完应用程序并顺利运行之后，你必须保持它的良好运行。你可能只需使用我们在本章开头所述的步骤定期重新访问你的应用程序并确定你是否需要扩展应用程序硬件。最好的解决方案是安装监视系统。

如果安装了实时监视系统，应用程序执行得如何，你一看便知。大多数监视系统还能够对系统的任何参数超过你设置的限制时触发“警报”。监视系统不仅仅给了你系统运行良好的信心，而且还会作为一个早期警告系统在出现问题时提醒你。

例如，如果其中一个 Web 服务器宕机，你可以让监视系统向你发送电子邮件或短消息来提醒这个问题。

为你推荐一些监视系统

监视系统足够写一整本书了。由于安装和设置出色的监视解决方案这个主题不在本书范围内，因此我们将只列出一些较常用的开源解决方案供你选择。

- ❑ Ganglia：一个实时监视系统，特别适合监视服务器阵列或服务器场，并且还提供有关各个服务器的性能统计信息。Ganglia 能够“累积”统计信息以提供在某个场中运行的一组服务器的综合统计信息。有关详细信息，请访问 <http://ganglia.sourceforge.net/>。
- ❑ Cacti：推荐的另一个出色的实时监视工具，它的特点是有大量可用的“探针”，可用于监视应用程序栈的每个部分。有关详细信息，请访问 www.cacti.net/。
- ❑ Nagios：开源监视系统之父，在系统可用性监视方面非常出色——它拥有巨大的“探针”库。有关详细信息，请访问 www.nagios.org/。

7.8 小结

在本章中，我们学习了如何确定应用程序的请求的大致情况，以及如何据此确定其内存占用。我们使用该信息来限制系统上的进程，以防止磁盘交换。

我们还检查了一些可以提高性能的配置文件的调整，特别是处理静态对象时，专注于优化 PHP 性能的工程师通常会忽略这个事实。

我们还介绍了当性能需要超出单个服务器的容量并且需要采用分布式方式运行时，我们可以执行的操作。

此外，我们还介绍了有哪些选择可以帮助分担处理静态资产（如图像、.js 文件和 .css 文件）的责任。

最后，我们介绍了可以从开源社区获得的一些监视工具，使用这些工具，你可以密切关注 Web 服务器的运行状况和性能，你可以确信，一旦出现问题，这些工具就会提前提示你。

介绍数据库优化的书数不胜数，其中很大一部分是讲述MySQL优化的。本书仅仅用一章的篇幅来讨论数据库优化，因此无论如何也无法全面地讨论这个主题。

我们将重点介绍与PHP程序员最相关的优化知识，并看看能否提供一些工具和经验规则来帮助你处理基本的优化问题，以避免应用程序中埋下的性能问题会导致日后付出昂贵代价来修复。

在本章中，我们将了解MySQL是如何使用内存的，并推荐一个用来优化MySQL服务器配置以实现其性能最大化的工具。我们还将介绍如何确定CPU或磁盘系统是否被占用得过多而导致资源耗尽。

最后还将探讨如何发现有问题的查询并确定它们的处理方式，以及如何优化索引以便消除不必要的磁盘I/O操作。

从图8-1中可以看出，数据库层是到达操作系统之前的最后一个应用程序层。由于它在应用程序中的基础地位，整个应用程序的性能通常与数据库服务器的性能有着直接的关系。因此，重点要防止数据库过载并且确保它已经通过优化而实现了最高的运行效率。



图8-1 PHP应用程序组件栈

8.1 MySQL 简介

MySQL是一个关系数据库管理系统（RDBMS），它的作用是使用结构化查询语言（SQL）对存储在各种存储引擎中的数据进行访问。

MySQL源代码遵守“GNU 通用公共许可证”，但也有商用许可的版本，后者提供企业级支持。

MySQL激发了很多派生项目，这些项目各自侧重于特定应用程序的各个不同方面。这些项目包括MariaDB、OurDelta、Drizzle和Percona Server（XtraDB）。

Percona、Google和Facebook等组织都提供了专业的分发版或补丁集，它们为标准MySQL分发版中添加了更多的扩展选项。

MySQL可以用于所有主流操作系统，包括Linux、Windows、Mac OS X、Solaris、FreeBSD，等等。

8.2 了解 MySQL 存储引擎

MySQL是一个分层的应用程序。通信、查询解析、优化，以及数据存储与访问分别由不同的层来处理（见图8-2）。

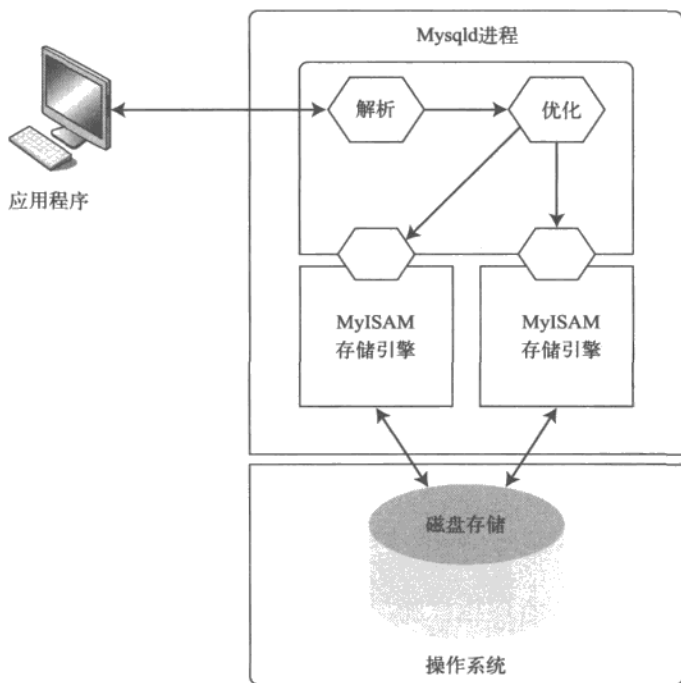


图8-2 mysql进程和存储引擎

MySQL的一大特色是具有多个数据存储和访问层（称为“存储引擎”）。默认的分发版自带很多这样的存储引擎。最有名的两个引擎是“MyISAM”和“InnoDB”。

其他一些引擎主要是为专用功能设计的，一般不会受到广泛的关注，但MyISAM和InnoDB是在大多数的安装版本中被大量使用的，因此值得我们进一步探讨。

我们不会一一介绍这两种引擎的所有细节，而是只列出它们的主要优缺点，这样既可以帮助你理解它们的优点和缺点，也可以帮助你选择应该使用哪一个。

遗憾的是，在这里简单地说“xxx”优于“yyy”并进行简单推荐是行不通的，每种引擎都有很多属性，这些属性使得它们或多或少地只适合特定的应用程序。

你必须在开发周期初期做出决定，这可能就会大大改变构造、设置和部署系统的方式。

8.2.1 MyISAM：原始引擎

MyISAM是随MySQL一起开发的原始存储引擎。设计它的目的是使用唯一键（每个记录有一个唯一键）在主要发生读取操作的工作负载中快速检索记录。如果站点中95%~100%的操作都是读取操作，那么毫无疑问该引擎就是最佳解决方案。但需要注意的是，该引擎也有一些缺点，如表8-1所示。

表8-1 MyISAM的优缺点

优 点	缺 点
能够快速查询唯一键	表级别锁定，如果你的应用程序写入操作的总时间占到5%以上，则表锁定会减慢应用程序的速度
支持全文索引	不支持事务，没有“开始=>提交/中止”能力
选择count(*)的速度很快	有持久性问题，表崩溃可能需要冗长的修复操作才能使其恢复联机
磁盘空间占用较少	

MyISAM是非事务性引擎，因此它无法回滚失败的事务或查询。

8.2.2 InnoDB：专业级的选择

InnoDB是一个符合ACID（原子性、一致性、隔离性、持久性）的存储引擎，它包括版本控制和日志功能，并且具有提交、回滚和防止数据损坏的灾难恢复功能。InnoDB还实现了行级锁定和一致的非锁定读取，这可以显著提高多用户并发和性能。InnoDB把用户数据存储存储在集群索引中，以减少最流行的查询类型（即基于主键的查询）的磁盘I/O操作。为了保持数据完整性，InnoDB还支持外键和引用完整性约束。

你可以把MyISAM的表与InnoDB表一起使用，即使在同一数据库中也可以这样做。表8-2显示了InnoDB存储引擎的主要优缺点。

表8-2 InnoDB的优缺点

优 点	缺 点
支持事务，可以放弃和回滚查询。崩溃不会导致数据损坏	从xxxx查询中选择count(*)的速度大大降低
具有行级锁定，并发写入同一表的不同行不会被序列化	没有全文索引
支持全部ACID功能的版本控制	自动递增字段必须是表中的第一个字段，当迁移的时候可能会导致问题
支持多种联机备份策略	占用更多磁盘空间
提高了应用程序在高负载、大量连接下的并发能力	一些简单查询表单的速度可能低于MyISAM，但复杂的、多个表的查询速度则超过MyISAM

8.2.3 选择存储引擎

如前所述，究竟是选择MyISAM还是选择InnoDB是一个复杂的问题，但我们可以提供一些简单的经验规则，它们将有助于你轻松选择。以下几个小节提供了选择的一些依据。

1. 当应用程序执行的大部分是读取操作时（大于95%）

当你查看应用程序中的读写比例时，如果发现主要是读取操作，而对表的改写只是偶尔才发生，那么应该选择MyISAM。它在主要处理读取操作的工作负荷中的运行速度较快，但缺乏对表的广泛写入，由于写入操作很少，因此由于MyISAM缺乏行级锁定而导致的性能问题很少。

2. 当事务性和一致性非常重要时

毫无疑问，此时InnoDB肯定是正确的选择。MyISAM不支持事务并且无法回滚失败的更新来保持一致性。

3. 当你有一个包含很多连接表的复杂模式时

此时InnoDB再次成为最佳选择。InnoDB支持引用完整性检查（例如外键约束），这是确保大型复杂架构保持完好的一个重要功能。

此外，InnoDB支持事务的能力还确保当你更新具有约束关系的多个表时，更新部分的任何问题都会触发整个更新回滚——这也是引用完整性的另一个重要要求。

4. 当不间断操作非常重要时

如果你需要拥有24×7全天候运行，那么建议你选择InnoDB。MyISAM没有防止数据崩溃的日志、版本控制和记录功能，并且几乎所有MyISAM备份解决方案都需要某种形式的停机时间，即使只有片刻。

8.3 了解 MySQL 如何使用内存

MySQL对内存来者不拒，你给它的内存越多，它的性能就越好，直到达到某个点。这个点就是MySQL的速度超过数据“工作集”的速度，超过这个点之后，无论你再为它分配多少内存，它的性能几乎都不会再有提高。

“工作集”就是经常使用的数据集，你可能拥有一个15GB的数据库用于存放新文章，但如果人们在你的搜索界面中最多只搜索两星期之内的文章，那么“工作集”就是发布日期小于14天的所有文章。一旦这个数据集可以轻松地放到内存中，性能的提高也基本上就到了极限，特别是当拥有一个好的索引集时。

如果要让MySQL使用系统中安装的所有内存，就必须将其配置为使用这些内存。

在下一小节中，我们将介绍用于控制内存使用的指令，这些指令将直接影响性能。

8.3.1 InnoDB 与 MyISAM 内存使用的比较

MySQL配置文件提供了很多指令，可用于控制服务器的内存占用。这些可设置的信息可以分为以下几个大类的指令。

- 影响缓冲区和缓存大小的指令，这些指令适用于所有存储引擎。
- 仅影响MyISAM存储引擎的指令。
- 仅影响InnoDB存储引擎的指令，通常这些指令以“innodb_”开头。
- 控制各种资源（如连接数量等）限制的指令。
- 定义属性（如字符集、路径等）的指令。

如果只使用某一种存储引擎，那就只需考虑优化该引擎使用的内存即可。但当使用MySQL时，则可能会在同一服务器中混合使用多种存储引擎，甚至在同一数据库中会有一些表存储在不同的引擎中，因此可能需要在两个不同的引擎之间分配内存的使用。如果拥有一个混合的存储集，并且没有更好的理由使用两个中较小的那一个，那么将数据库转换为一个单一存储引擎可能会比较好，这样可以使很多事情更易于管理。我们会在后面看到，执行备份时，混合存储引擎还会限制一些选项的使用。

8.3.2 每服务器与每连接（线程）内存使用的比较

必须记住，当在配置文件中配置内存缓冲区和缓存的大小时，一些内存结构是按照每个连接或每个线程来分配的（见图8-3）。当连接到MySQL的连接数增加时，MySQL将使用更多的内存，因此一定要把从应用程序连接到数据库服务器的打开的连接数减至最少，这一点非常重要。

让我们看一看MySQL如何在动态（基于连接）内存使用和固定（基于实例）内存使用之间划分内存分配。

每活动连接（动态）使用的内存数量如下所示：

```
per_connection_memory =
    read_buffer_size           // memory for sequential table scans
    +read_rnd_buffer_size      // Memory for buffering reads
    +sort_buffer_size          // Memory for in mem sorts
    +thread_stack              // Per connection memory
    +join_buffer_size          // Memory for in mem table joins
```

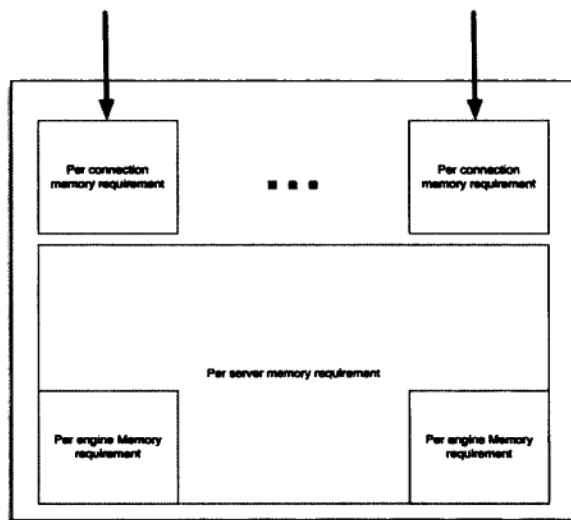


图8-3 mysqld的内存分配概览

每服务器（固定）使用的内存数量如下所示：

```
per_server_memory =
    tmp_table_size                // memory for all temp tables
    +max_heap_table_size          // max size of single temp table
    +key_buffer_size              // memory allocated for index blocks
    +innodb_buffer_pool_size      // main cache for InnoDB data
    +innodb_additional_mem_pool_size // InnoDB record structure cache
    +innodb_log_buffer_size       // log file write buffer
    +query_cache_size             // compiled statement cache
```

MySQL可以使用的最大内存定义如下：

```
max_memory = (per_connection_memory * max_connections) + per_server_memory
```

由此可以很容易地看出，如果数据量很大，那么必须提供足够的内存才能确保将尽可能多的操作放在内存中，以便减少昂贵的磁盘读写操作。但通常会忽略一个方面，这就是MySQL的每个连接都需要使用内存，因此，如果有很多连接到MySQL服务器上的Web服务器，而且每个Web服务器都有大量的活动连接，那么你可能需要提供足够的内存来支持它们。

现在，我们将继续介绍如何确定需要分配给这些内存缓冲区的内存数量。

8.4 查找配置文件

在介绍优化内存使用之前，首先需要确定MySQL配置文件的位置，以便应用优化过程中的任何更改。但遗憾的是，有关该文件应该放置的位置有很多不同的观点。

标准MySQL后台程序会在表8-3列出的位置中查找其配置文件。如果文件存在于多个位置，那么它就会按从上到下的顺序加载这些文件，较低文件中的指令优先于较高文件中的指令。

表8-3 MySQL将搜索配置文件的标准位置

文 件 名	用 途
/etc/my.cnf	全局数据库选项
/etc/mysql/my.cnf	全局数据库选项（截止MySQL 5.1.15）
[SYSCONFDIR]/my.cnf	全局数据库选项
\$MYSQL_HOME/my.cnf	特定于服务器的选项
[CUSTOM]	通过--defaults-extra-file=指定的文件
~/.my.cnf	特定于用户的选项

每个文件都是以标准.ini文件格式列出的，每个文件都包含多个部分，如“[mysqld]”，每个部分中都有指令。

更改配置文件之后，重新启动服务器使更改生效。

8.4.1 Mysqldtuner.pl: 优化数据库服务器的内存

在这一小节中，我们将对现有数据库服务器的性能进行评估，并介绍如何使用几种简单的技巧和工具来确定服务器是否配置正确。

为了执行这项任务，我们将使用几个常用或易于获得的工具来收集有关目标系统状态的信息。

第一项有用的信息就是最上面的“top”这一行所显示的服务器信息。它告诉我们有关服务器上负载数量的详细信息，如果正在发生交换，则会告诉我们mysqld进程正在使用的总内存数量（见图8-4）。

```
top - 09:38:13 up 235 days, 2:39, 1 user, load average: 0.62, 0.56, 0.44
Tasks: 69 total, 1 running, 65 sleeping, 0 stopped, 3 zombie
Cpu(s): 4.3%us, 0.0%sy, 0.0%ni, 86.5%id, 9.0%wa, 0.0%hi, 0.2%si, 0.0%st
Mem: 17927580k total, 17851776k used, 75804k free, 185020k buffers
Swap: 0k total, 0k used, 0k free, 4989364k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 16570 mysql    15   0 12.9g 11g 6032 S    9  66.4   3567:29 mysqld
    1 root      15   0 10304  800  672 S    0   0.0    0:07.93 init
```

图8-4 负载规模适度的MySQL服务器的top命令输出

从刚刚显示的信息我们可以看出，该服务器的负载并不大。负载因子仅为0.62、0.56和0.44，在这台机器这是一个非常低的负载水平。

该计算机完全没有发生交换，并且mysqld的CPU和内存使用分别为9%和66%，这种情况也非常好。它没有占用大量CPU，并且使用的系统内存数量也比较合理。如果系统使用了过量的CPU或内存（发生交换），那么我们的重点将是减小MySQL的最大内存占用以防止溢出。这台服务器装备有17.5GB的内存。

现在让我们运行*iostat*工具来看看I/O性能。*iostat*应该可以从你的分发版软件资源库中安装，并且所有分发版应该都提供了该工具。我们将使用该工具的-d -c和-x选项来开启设备、CPU以及扩展的状态信息。图8-5显示了在我们的示例服务器上运行*iostat*时所产生的输出。

```
$ iostat -d -c -x 2
Linux 2.6.21.7-2.fc8xen-ec2-v1.0 (db64)           Wednesday, 03 November, 2010

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           9.83    0.05    2.68    1.29    0.08   86.06

Device:            rrqm/s   wrqm/s     r/s     w/s    ...  await  svctm   %util
sdal                0.00    3.46    0.02    1.33    ...   0.99   0.09   0.01
sdb                 0.77    7.26    1.14    2.29    ...  50.77   4.79   1.64

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           5.84    0.00    0.00   11.68    0.00   82.49

Device:            rrqm/s   wrqm/s     r/s     w/s    ...  await  svctm   %util
sdal                0.00   15.38    0.00    3.08    ...    0.00   0.00   0.00
sdb                 0.00    9.74    1.03   17.44    ...  269.78  25.58  47.23
```

图8-5 负载规模适度的MySQL服务器的*iostat -d -c -x 2*输出

从这个*iostat*报告中我们可以看出，数据库文件（sdb）所在的驱动器的利用率在1.64%~47.23%范围内变化。但这并不需要特别关注。这个值（%util）显示了在该示例中正在被使用的磁盘通道的I/O性能。我们的驱动器并没有过载——%iowait值也确认了这一点，%iowait值表示系统等待I/O操作完成的时间。该时间范围为1.29%~11.68%，这再次告诉我们所有一切都非常好——应用程序并没有受到I/O的限制。

如果你看到较高的%util或%iowait值，那么应该查看一下驱动器的性能，或者看看打开的一些MySQL内存缓冲区以减少MySQL服务器命中磁盘所需的时间。但该系统没有显示I/O性能的任何问题，因此我们可以继续进行。

现在，你可以开始检查MySQL服务器进程的内部情况了。我们将引入一个非常有用的开源工具，即mysqltuner.pl，该工具可以减轻配置和检查数据库服务器的大量工作。虽然有了它之后并不代表你可以不用去了解深层的进程优化知识，但作为一款出色的工具，它可以帮助你快速检查服务器的设计是否合理，并且定位那些明显有问题的错误配置。这个脚本还会检查计算机的最近使用情况并根据实际的使用提出可以提高性能的改进之处。

你可以在运行的服务器上运行此脚本。它就不会对系统造成破坏，也不会给系统造成明显负担，因此可以使用它对你的计算机进行定期的健康检查。

若要安装该脚本，需要将它下载到数据库服务器。该脚本的创作者已注册了一个网址mysqltuner.pl，并且主页便是该脚本。因此要下载和安装它，只需按照以下步骤执行即可。

```
$wget mysqltuner.pl -O mysqltuner.pl
$chmod +x mysqltuner.pl
```

现在就应该能够运行此处所示的脚本了。系统将提示你输入用户名和密码（这个用户名需要有数据库服务器访问权限）。输入之后，该脚本将检查你的服务器并产生类似于图8-6所示的输出。

```

$ ./mysqltuner.pl

>> MySQLTuner 1.0.1 - Major Hayden <major@mhtx.net>
>> Bug reports, feature requests, and downloads at http://mysqltuner.com/
>> Run with '--help' for additional options and output filtering
Please enter your MySQL administrative login: root
Please enter your MySQL administrative password:
[!!!] Successfully authenticated with no password - SECURITY RISK!

----- General Statistics -----
[--] Skipped version check for MySQLTuner script
[OK] Currently running supported MySQL version 5.0.45-log
[OK] Operating on 64-bit architecture

----- Storage Engine Statistics -----
[--] Status: -Archive -BDB -Federated +InnoDB -ISAM -NDBCluster
[--] Data in MyISAM tables: 589M (Tables: 649)
[--] Data in InnoDB tables: 11G (Tables: 138)
[!!!] Total fragmented tables: 32

----- Performance Metrics -----
[--] Up for: 19d 21h 25m 22s (237M q [138.423 qps], 13M conn, TX: 688B, RX: 30B)
[--] Reads / Writes: 91% / 9%
[--] Total buffers: 10.1G global + 21.4M per thread (500 max threads)
[!!!] Maximum possible memory usage: 20.5G (119% of installed RAM)
[OK] Slow queries: 0% (2K/237M)
[OK] Highest usage of available connections: 7% (39/500)
[OK] Key buffer size / total MyISAM indexes: 512.0M/204.4M
[OK] Key buffer hit rate: 100.0% (45M cached / 6K reads)
[!!!] Query cache efficiency: 0.4% (487K cached / 112M selects)
[OK] Query cache prunes per day: 0
[OK] Sorts requiring temporary tables: 2% (133K temp sorts / 6M sorts)
[!!!] Temporary tables created on disk: 38% (5M on disk / 14M total)
[OK] Thread cache hit rate: 99% (141 created / 13M connections)
[!!!] Table cache hit rate: 1% (512 open / 36K opened)
[OK] Open file limit used: 29% (738/2K)
[OK] Table locks acquired immediately: 99% (4M immediate / 4M locks)
[!!!] InnoDB data size / buffer pool: 11.1G/8.0G

----- Recommendations -----
General recommendations:
  Run OPTIMIZE TABLE to defragment tables for better performance
  Reduce your overall MySQL memory footprint for system stability
  Temporary table size is already large - reduce result set size
  Reduce your SELECT DISTINCT queries without LIMIT clauses
  Increase table cache gradually to avoid file descriptor limits
Variables to adjust:
  *** MySQL's maximum memory usage is dangerously high ***
  *** Add RAM before increasing MySQL buffer variables ***
  query_cache_limit (> 512M, or use smaller result sets)
  table_cache (> 512)
  innodb_buffer_pool_size (>= 11G)

```

图8-6 一台负载适中的服务器上mysqltuner.pl的输出

在后台，这个用于优化的脚本会访问数据库服务器并运行两个查询，即“SHOW STATUS”和“SHOW VARIABLES”。前者会返回一些统计信息，包括服务器的当前性能、查询数量、缓存

效率，等等。后者则将检索所有内部数据结构的大小。根据这两组信息，优化脚本计算出刚刚所示的结果并使用一组内置规则来提出建议。

示例中所显示的是一台中等规模的数据库服务器，它支持一个日流量为每天约35 000名独立用户的网站，因此服务器每天接到相当多查询请求。

尽管该服务器执行得不错，但它仍然暴露了很多问题，我们将对这些问题进行检查，并确定可以采取哪些措施来处理它们。

8.4.2 示例服务器可能出现的问题

前面的mysqldtuner.pl报告突出了两个可能发生的問題，其中一个可能非常严重，而另一个可能不会太严重。

第一个问题是系统被配置为使用了太多的内存。系统声明它将使用119%的系统内存，但经过对服务器内存使用情况检查，我们发现它只使用了66%的内存。那么这说明了什么？

线索来自于报告中的下面这几行：

```
[--] Total buffers: 10.1G global + 21.4M per thread (500 max threads)
[!!] Maximum possible memory usage: 20.5G (119% of installed RAM)

[OK] Highest usage of available connections: 7% (39/500)
```

还记得前面讲过的用于计算最大内存使用的公式吗？公式中指定每个连接使用的内存为21.4MB。由于我们配置了最大连接数为500，总计的内存就约为10.7GB。每台服务器的固定内存消耗为10.1GB，因此MySQL可能消耗的总内存为20.8GB，比计算机中实际安装的17.5GB内存超过了大约15%。

也就是说，如果我们允许将连接数增加到其最大值500，那么计算机将被迫发生交换。但现在被使用的最大连接数仅为7%（39）。服务器已经运行了20天，最大连接数从未超过39。因此，我们可以采取的第二个操作就是将最大连接数从500减至100，这样就把每个连接池的总内存减少到2.14GB。现在系统的17.5GB内存完全能够满足这12.24 GB的总内存需求，并且也消除了它进入交换状态的风险。

现在，系统给我们留下了4GB的备用内存，因此，我们可以进行的第二个改进就是将innodb_buffer_pool从8GB增加到11GB，这样整个工作集就能够轻松地放到内存中了。如你所见，这就是优化程序脚本的建议之一。

```
innodb_buffer_pool_size (>= 11G)
```

最后，优化脚本建议我们增加两个缓冲区，于是把最后的这1GB分配给这两个缓冲区。服务器又重新回到最佳状态。

```
query_cache_limit (> 512M, or use smaller result sets)
table_cache (> 512)
```

但是，应该注意的是，这里的优化是根据服务器到目前为止所遇到的负载和查询混合进行的。

每个月运行一次这个优化程序是个好主意，这样可以在出现问题的时候尽早知道，以便尽早升级硬件（例如增加内存）。

8.4.3 优化 InnoDB

InnoDB对内存很敏感，因此为它配置正确的内存数量非常重要。如果它没有足够的内存来缓冲一个合理规模的数据集，那么它的性能可能迅速恶化。下面我们提供几个非常适合中等规模DB服务器的InnoDB内存设置的简单示例。

以下设置基于一个16GB的数据库服务器，这是很常见的大小。

- ❑ `innodb_file_per_table`：默认情况下，InnoDB为每个数据库创建一个文件并用这个文件来管理数据库的表。这意味着如果表的大小先增大然后又缩小，则很难恢复磁盘空间。设置该选项将使InnoDB对每个表使用一个独立的数据存储文件。如果你想在现有数据库上更改此设置，应该备份该数据库，然后删除它，再更改选项，然后重新启动服务器，最后从备份中重新加载该数据库。
- ❑ `innodb_buffer_pool_size=`：如果你只使用InnoDB表，可以把这一项设置为可用内存的70%左右。如果你混合使用MyISAM和InnoDB，则将它减小一点以便给MyISAM留一些空间。
- ❑ `innodb_log_buffer_size=4M`：4M的大小可以满足绝大部分记录的需求，而且提供了一个合理的性能。如果你拥有很大的文本字段或blob字段，或者记录非常大，可以把这个值稍微提高一点。
- ❑ `innodb_log_file_size=256M`：这是推荐的值，可以在恢复数据库的速度与保持较高的运行时性能之间取得良好的平衡。
- ❑ `innodb_flush_log_at_trx_commit=2`：这一项控制将日志文件刷新到磁盘的频率。如果你可以容忍在发生崩溃时丢失一些记录，那么可以把它的值设为2来减少磁盘写入的数量、加快性能以及减少驱动器上的I/O负担。

8.5 找到有问题的查询

优化服务器配置只是提高性能的方法之一。到目前为止，最常见的性能问题的根源是查询结构不合理或者丢失了索引。

MySQL有一个内置的用于将低性能查询记录到日志文件中的机制，因此这些查询可以被识别出来并进行优化。若要启用MySQL的这个慢速查询日志功能，需要向服务器配置文件中添加以下两行，将其添加到“[mysqld]”部分下。

```
[mysqld]
log-slow-queries = /var/log/mysql/mysql-slow.log
long_query_time=1
```

你可以根据需要替换成你自己的路径，但要确保MySQL进程可以写入该文件夹。`long_query_time`指令设置了一个阈值，超过这个阈值的查询将被归类为“执行速度慢”的查询。我们给出的

示例为1秒，因此运行时间超过1秒的任何查询都会记录到慢速查询日志文件中。

让我们看一个例子。在我的数据库系统中，有一个名为“articles”的表，该表中有一个“article_title”字段。我的应用程序需要获取按标题排序的前10个文章标题，以便它可以一次翻阅10篇文章。界面支持按照管理工具中显示的每个列进行排序，按照文章标题排序也是其中的一个。管理员抱怨当他切换到按标题排序视图时，应用程序变得很慢并且从一页翻到另一页很费劲。

尽管我们都猜测可能是这个查询的问题，但是使用工具来帮助我们确定和修复问题是一个很有用的过程，因为这也有可能是一个更为复杂的问题。

应用程序用于获取第一页的列视图数据的查询命令如下：

```
SELECT article_title from articles order by article_title limit 10;
```

当我在MySQL查询工具中运行我的查询时，得到图8-7所示的结果。

```
mysql> select article_title from articles order by article_title limit 10;
+-----+
| article_title |
+-----+
| " I want to beat Ferguson's United " |
| "...one more triumph for the crass stupidity rapidly replacing culture in this country..." |
| "A bad day at the office"... |
| "A case Metzelder will be no more" |
| "A Smarter (and Cost-Efficient) Way to Fight Crime" |
| "A Strike Fit To Win Any Game Of Football" |
| "A Win, A Win, My Kingdom For A Win" |
| "Action" Jackson Asiku will carry the hopes of two nations on Friday night |
| "Al Arabi Sports" logo unveiled |
| "All" or Nothing |
+-----+
10 rows in set (6.92 sec)

mysql>
```

图8-7 排序后的文章标题查询示例（优化之前）

时间为6.9秒——看来不是太好。如果这是提供管理屏幕的查询，那么我们的文章管理员将花费较长的时间翻阅所有文章。因此，让我们来看看慢速查询日志。

```
# Time: 101103 23:03:00
# User@Host: root[root] @ localhost []
# Query_time: 6.920107 Lock_time: 0.000111 Rows_sent: 10 Rows_examined: 123675
SET timestamp=1288796580;
select article_title from articles order by article_title limit 10;
```

日志告诉我们，为了找到我的10个记录并将它们发送给我，它需要从文章表（数据库表的整个内容）中读取123675个记录，由于文章记录的页面大小也非常大（因为它包含所有文章文本），那么它无疑要从磁盘中读取很多数据。事实上，在我们的这个例子中，仅该表中的数据就达到了近400MB。

通常，你将使用慢速查询日志文件来查找导致问题的查询，在这个例子中，我们已经发现了这个问题或者系统已经将该问题报告给我们，并且使用该日志证实了我们的怀疑。

在下一节中，我们将使用一些MySQL的内置工具来查明查询速度如此之慢的原因，并确定如何纠正这个问题。

8.6 分析有问题的查询

找到有问题的查询之后，我们可以使用MySQL中内置的实用工具来显示MySQL服务器检索数据时执行的步骤。这将为我们提供线索，帮助我们找到问题的位置。

MySQL有一个查询分析器，它能够对查询进行检查，并使用它所检查的表的相关信息来检查表本身的一些统计数据 and 索引列表。此分析的结果称为“执行计划”（execution plan），它列出了执行查询所要执行的步骤。

我们可以指示MySQL来显示“执行计划”而不是运行查询，这样，我们就可以看到它将执行的操作。为此，我们使用“Explain”语法。在任何查询的前面添加“Explain”将返回内部执行计划的表示。

```
mysql> explain select article_title from articles order by article_title limit 10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	articles	ALL	NULL	NULL	NULL	NULL	123675	Using filesort

那么我们从此表示中了解了哪些内容呢？重要的信息是“using filesort”以及123675这个行数，它基本上表示获取表中所有数据的副本并使用quicksort对其排序，以便我们可以确定前10个记录并将它们发回。

在“article_title”字段上添加索引将大大提高性能，因为服务器不必创建临时表并对内容进行排序。通过遍历索引中已排序的前10项而不是未排序的数据文件，我们将能够确定需要按正确顺序发送的记录。因此，我们通过以下代码向表中添加一个索引：

```
CREATE INDEX title_idx on articles (article_title);
```

在我们的查询上再次运行explain，可以看到此时已使用我们的索引进行了排序。

```
mysql> explain select article_title from articles order by article_title limit 10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	articles	index	NULL	title_idx	767	NULL	10	Using index

你可以看出已读取的行数下降到10，这个数字与我们的查询请求的数字相同。

现在，如果我们再次执行查询，可以看到应用索引的效果（图8-8）。

查询的性能有了很大的提高。现在执行查询的速度非常快，以至于MySQL都无法显示出查询的持续时间（显示为0.00秒）。因此，现在我们的管理工具将以按文章标题排序的模式压缩各个页面，管理员的烦恼解除了，并且欠了我一个大人情。

```
mysql> select article_title from articles order by article_title limit 10;
+-----+-----+
| article_title |
+-----+-----+
| " I want to beat Ferguson's United " |
| "...one more triumph for the crass stupidity rapidly replacing culture in this country..." |
| "A bad day at the office"... |
| "A case Metzelder will be no more" |
| "A Smarter (and Cost-Efficient) Way to Fight Crime" |
| "A Strike/Fit To Win Any Game Of Football" |
| "A Win, A Win, My Kingdom For A Win" |
| "Action" Jackson Asiku will carry the hopes of two nations on Friday night |
| "Al Arabi Sports" logo unveiled |
| "All" or Nothing |
+-----+-----+
10 rows in set (0.00 sec)

mysql>
```

图8-8 文章标题查询示例（优化后）

8.7 PHP 数据库应用程序的建议

开始对你的应用程序进行编码之前，应该认真考虑几个设计问题。在很多情况下，开发人员倾向于使用MySQL出厂设置中提供的默认值。开发人员做出的模式和配置选择最终往往不可避免地成为系统投入使用后的默认值。如果从一开始就设置错了，那么当应用程序投入使用后你第一次面对这些问题时，再修复所发现的错误可能就意味着付出高昂的代价。

8.7.1 保持独立的读写连接

开始就创建两个数据库连接是一个好的方法，一个用于读取，一个用于写入，并且允许不同的数据库服务器连接它们。如果你只有一个服务器，则将它们设置为彼此相同。

当你进行应用程序的编码时，可以把更改数据的任何查询（UPDATE、INSERT、DELETE等）都写成使用写入连接，纯SELECT或读取查询则一律使用读取连接。

如果你需要升级你的应用程序，则可以将数据库服务器分离到其他计算机上，并通过复制来连接它们。但若要实现这一工作，必须确保所有写入都指向你的主要服务器，所有读取都指向适当的从属服务器。

通过使用两个连接，可以轻松重新配置你的应用程序以支持大量不同的扩展选项，使用一个或多个从属服务器来增加查询带宽。从一开始就实现这种方案只需要很少的努力，但之后却会大大增加你的选择。

8.7.2 默认使用“utf8”（多字节 Unicode）字符集

现如今，你应该使用“utf8”字符集编写所有应用程序的存储和页面呈现功能。存储密度的开销非常小，而且对于普通的ASCII文本，则完全没有开销。但如果除了英语之外，你还需要支持其他备选语言的存储，或者必须处理非英语国家名字然后存放在你的数据集中，那么你需要

具备这个能力。启动服务之后，将一个庞大的数据库从默认的ISO-8859-1格式转换为“utf8”是一项令人望而生畏而且耗时的任务，还不如从一开始就在所有地方都使用“utf8”，这会让你轻松很多。

通过在MySQL配置文件中设置一些参数，可以强制所有新的数据库、表和文本字段在默认情况下都采用“utf8”创建。

```
[mysqld]
collation_server=utf8_unicode_ci
character_set_server=utf8
skip-character-set-client-handshake
```

最后一条指令告诉服务器不执行与客户端的字符集协商（negotiation）。通过设置此选项，可以确保所有客户端和连接都设置为采用“utf8”操作，而无需在连接到此数据库服务器的每个服务器的my.cnf文件上专门配置这项内容。为了确保与MySQL服务器交互的所有服务都采用一致的行为，这是一个好方法，并且还可以确保在读取或写入服务器时所有的数据都无需进行字符集转换（这种转换也会影响性能）。

直接设定默认的字符集而关闭协商功能还有另一个好处，那就是你过后不再需要向连接发送“set NAMES utf8”语句来确保它切换到utf8。尽管该语句非常小并且执行的速度也很快，但它需要往返于服务器，并且如果你将数据库连接设置为utf8，很多PHP框架都会在每次查询之前自动发送该语句。通过上面所示的配置，你可以不必设置连接字符集并且不必重复发送该语句。

前面的指令还指明了当你创建一个没有任何特定字符集模式属性的数据库或表模式时会如何处理。你所创建的没有“[DEFAULT] CHARACTER SET utf8”属性的实体都将自动被设置为“utf8”。但是，需要注意的是，如果你使用一个工具（如phpMyAdmin）来操纵你的模式，请务必小心，确保该工具不要应用自己的默认设置并且不要创建那些会覆盖你刚刚设置的默认选择的模式属性。

8.7.3 使用“UTC”日期格式

同样，使用常用的日期格式来存储日期时间值也是一个好方法，这样可以轻松比较日期和时间，而不必担心时区。在PHP中，在UTC与本地时区之间来回转换是非常容易的。

为了将MySQL设置为在UTC时区中操作，必须确保在你的MySQL实例中安装了该时区支持。这是一个关于特定时区信息的数据库，而这些信息通常都不是默认安装的。

若要在MySQL中安装时区支持，首先必须查找你的操作系统时区数据库。在大多数Linux系统上，它位于/usr/share/zoneinfo中。找到后，可以使用MySQL提供的“mysql_tzinfo_to_sql”实用工具将时区信息转换为适合加载到你的MySQL系统中的SQL脚本。

```
$ mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
Warning: Unable to load '/usr/share/zoneinfo/Asia/Riyadh87' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/Asia/Riyadh88' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/Asia/Riyadh89' as time zone. Skipping it.
```

你可能会看到上面所示的一些警告，可以安全地忽略它们。

将时区数据库加载到MySQL中之后，可以修改MySQL配置文件，将UTC指定为默认时区。打开配置文件，向该文件的“[mysqld]”部分中添加以下指令。

```
[mysqld]  
default-time-zone=UTC
```

重新启动MySQL服务器，它现在默认的所有日期/时间值都应该是UTC。如果由于某种原因你的MySQL服务器未启动，可能是时区数据库未加载，因此它无法设置默认时区。检查mysqld.log文件看是否出现这种情况，如果是，则再次检查时区信息数据库的安装。如果你没有安装时区支持数据库而尝试设置默认时区，MySQL将不会启动。

PHP有很多功能可用于在UTC日期格式与其他格式之间进行来回转换，其中很多功能对“区域”敏感，因此我们很容易创建允许用户设置首选时区的用户界面，并且可以把所有日期/时间都显示为用户所在的本地时区的时间。

8.8 小结

本章介绍了MySQL服务器整体结构，以及它是如何支持可插入式的存储引擎的。我们了解了这些存储引擎的特征，以及如何选择适用于我们的应用程序的存储引擎。我们还研究了MySQL如何使用内存并学习了一些技术将MySQL可用的内存配置为最佳状态。还学习了如何检测低效的查询，并了解了分析并纠正它们的过程。

在整本书中，我们介绍了如何在应用程序栈的每个阶段诊断性能问题，从PHP运行时性能到应用程序代码，再到Web服务器，最终到数据库服务器本身。在每个阶段，我们都介绍了如何改善系统以消除性能瓶颈，也介绍了能够把性能影响降至最小的各种编码策略。

编写快速、高效、可伸缩的代码并不是说只需运用一组经验规则就可以了，而应该是深入了解应用程序的每个阶段背后所发生情况，编码的目的是避免可能阻碍你的应用程序的瓶颈。

我们希望本书所述的工具和技巧将帮助你理解应用程序的一些深层次问题，并帮助你形成一个对如何编写优秀应用程序的更全面的理解。

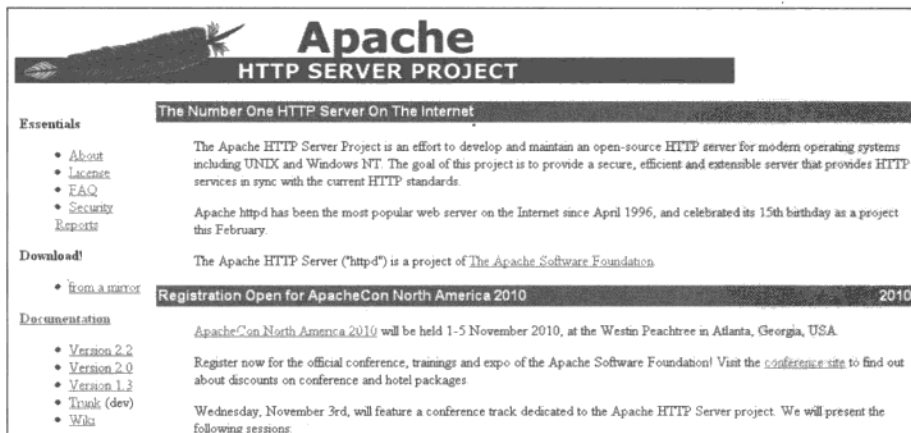
在Windows上安装Apache、MySQL、PHP和PECL

在整本书中，我们概述了优化各种Web服务器和数据库配置的方法，但并未一步一步详细介绍安装这些工具的过程。本附录作为一个参考，可帮助你安装以下工具：

- ☐ Apache 2.2
- ☐ MySQL 5.0
- ☐ PHP 5.0
- ☐ PECL

A.1 安装 Apache

编写本书时，Apache已发布了其免费Web服务器的2.2.x版本。请访问站点<http://httpd.apache.org>，单击左侧的“Download! from a mirror”链接，如图A-1所示。



图A-1 Apache HTTP服务器项目主页

选择图A-2中所示的镜像站点之一。

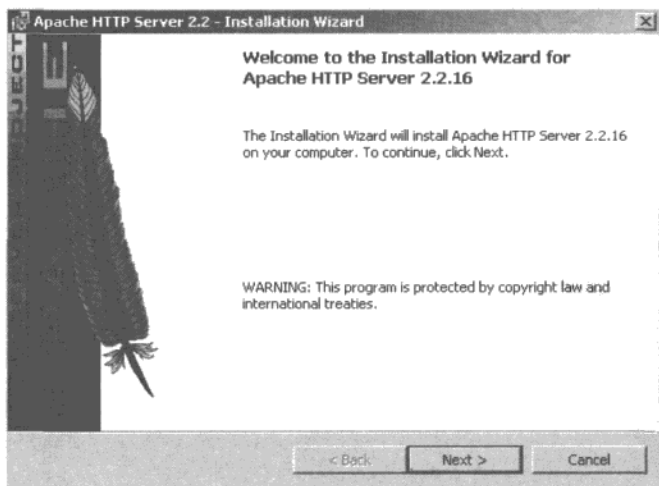


图A-2 Apache Server Windows和Unix下载链接

Windows用户应该下载Windows安装程序文件apache_2.2.x-win32-x86-no_ssl-r2.msi。

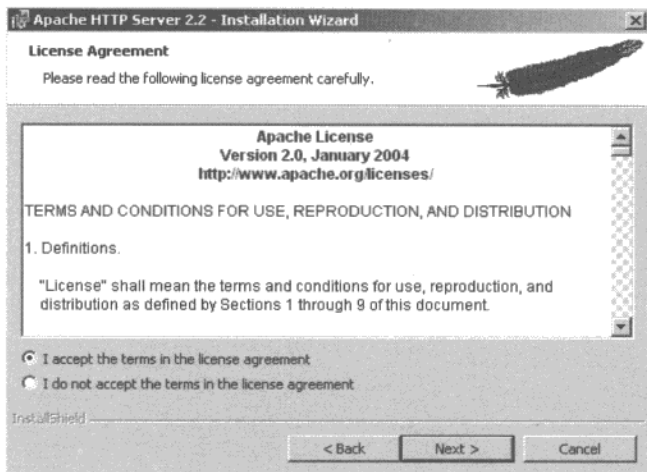
下载完该文件之后，打开安装程序。初始窗口将是一个安全警告，根据你的Windows版本，忽略该警告并单击“运行”按钮开始安装。

下一个窗口是Apache安装窗口。如果你安装了早期版本的Apache，则可能会出现另一个弹出窗口，该窗口要求你开始新安装之前先删除之前的安装。如果你不想升级，则跳过涉及Apache的这些步骤。如果未安装早期版本，则单击初始窗口中的“Next”按钮，如图A-3所示。



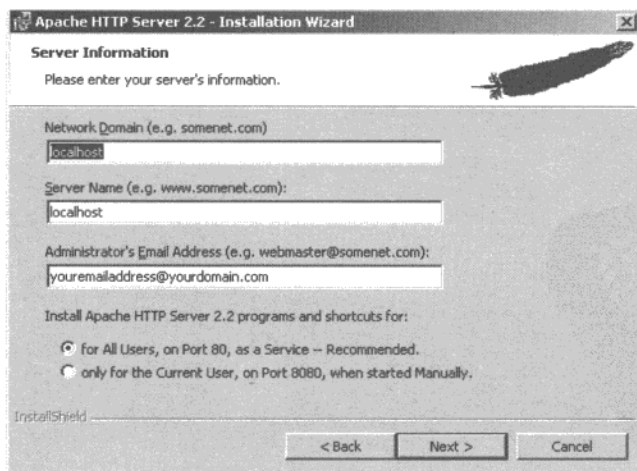
图A-3 Apache安装欢迎窗口

此时，选择“I accept the terms in the license agreement”并单击窗口中的“Next”，如图A-4所示。出现下一个窗口之后，再次单击“Next”按钮。



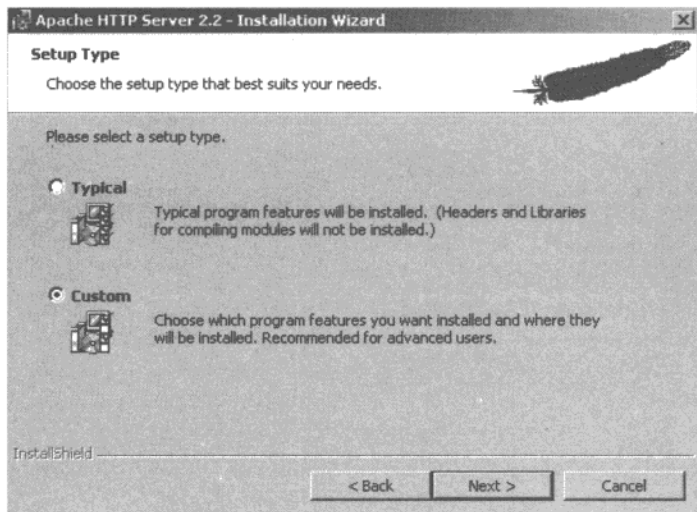
图A-4 Apache服务器条款和条件窗口

很快我们来到了如图A-5所示的窗口。这里，我们需要填写所有字段。由于这将在我们的桌面上运行的Web服务器，因此可以在“Network Domain”和“Server Name”字段中分别添加任何网络域名和服务器名称。我选择在这两个字段中都输入“localhost”。在“Administrator's Email Address”字段中，输入你的电子邮件地址，然后单击“Next”。



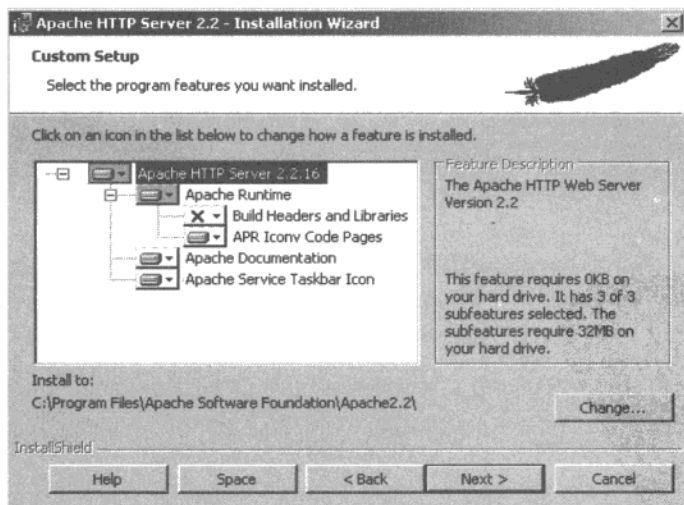
图A-5 Apache Server 信息设置窗口

接下来，我们将开始安装该软件，但我们需要告诉Apache安装向导安装该软件的位置。在图A-6所示的窗口中选择安装位置。单击“Custom”单选按钮，然后单击“Next”。



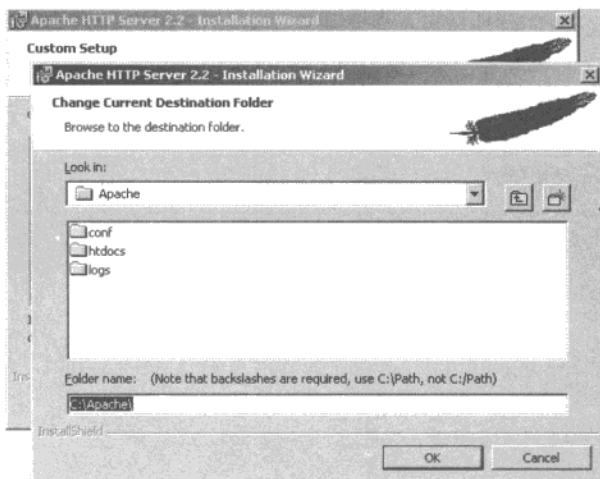
图A-6 Apache 安装类型窗口

在图A-7中所示的屏幕中，单击“Change”按钮。



图A-7 Apache 自定义安装窗口

为了便于在整本书中使用，我建议你将安装目录更改为C:\Apache；当然，你也可以将Apache保存在其他位置。为了便于以后参考，请记住从此处开始将C:\Apache称为APACHE_HOME。单击“OK”，然后单击“Next”（图A-8）。

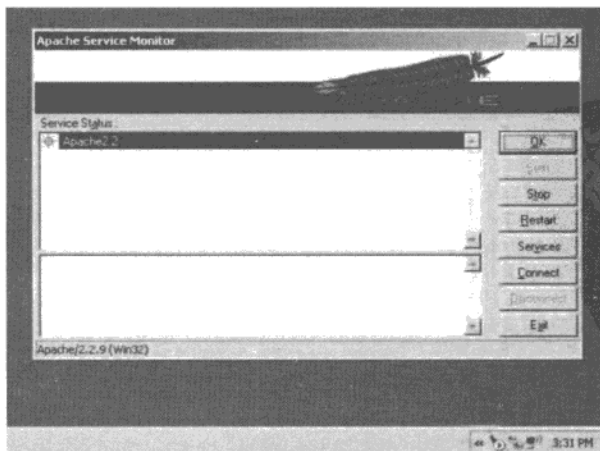


图A-8 Apache Change 当前安装位置窗口

现在到了最后一个窗口，单击“Install”，窗口关闭，这一切就这么简单。我们已成功在计算机上安装了Web服务器。

安装 Apache 之后

如果在安装Apache期间没有任何错误，此时你应该会在任务栏中看到“Apache Monitor”图标，如图A-9所示。右键单击该图标并单击“Open Apache Monitor”。使用此工具可以启动和停止Web服务器。



图A-9 Apache Monitor的Windows任务图标

现在, 让我们确保我们的计算机上正在运行 Apache。为此, 我们需要从 Web 浏览器中调用 Apache, 打开你最喜欢的 Web 浏览器并键入 `http://localhost`。此时你应该会看到 Apache 欢迎主页, 如图 A-10 所示。如果你有任何问题并且无法看到该页面, 请查看 Apache 错误日志, 该日志位于 `APACHE_HOME/logs/error.log` 中。常常可以在此处找到你的问题, 并且只需阅读保存到这些文件中的错误即可轻松解决。

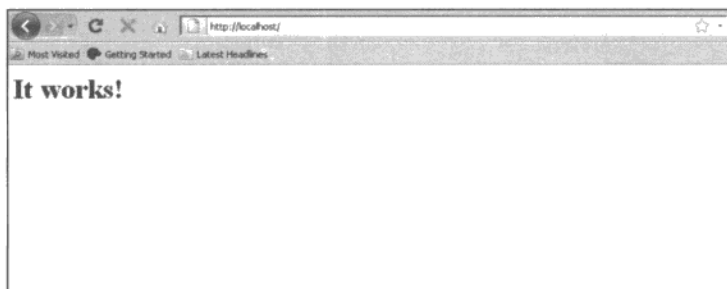


图 A-10 Apache Server 欢迎页面

A.2 安装 MySQL

安装 MySQL 也非常简单。访问网站 `www.mysql.com`, 单击顶部菜单栏上的“Downloads(GA)”链接下载最新的软件。到达图 A-11 所示的屏幕之后, 单击左侧菜单栏中的“MySQL Server”链接。

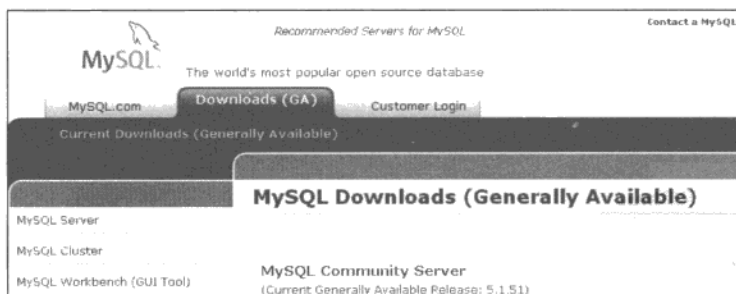


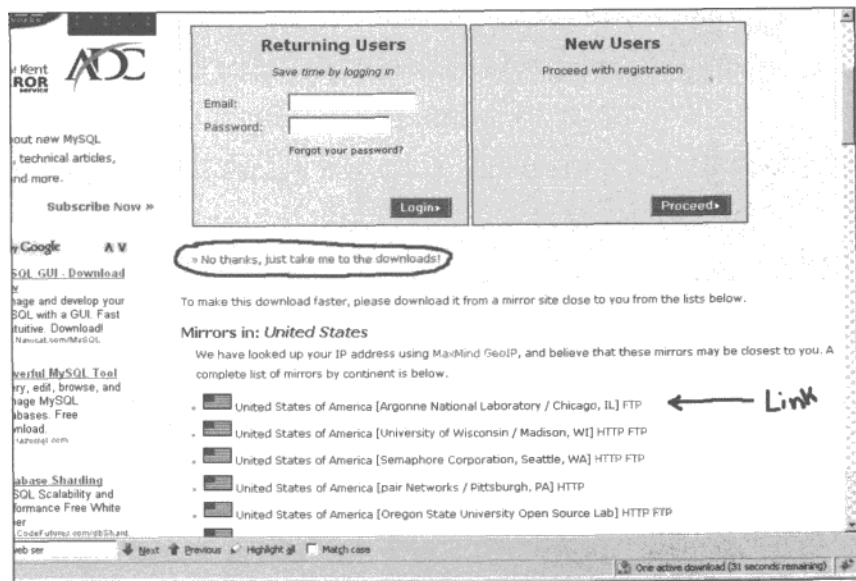
图 A-11 MySQL Downloads 主页

你需要向下滚动网页, 直到看到图 A-12 所示的部分。

与 Apache 一样, MySQL 也为我们提供了在 Windows 或 Unix 中安装的选项。Windows 用户应该下载 Windows 安装程序 `Windows ZIP/Setup.EXE (x86)`, Unix 用户应该从下拉菜单中选择“UNIX OS”下载适当的 Unix 风格安装程序。选择程序包之后, 系统将要求你登录你的账户。单击图 A-13 中所示的“No thanks, just take me to the downloads!”, 此时将显示一个镜像链接列表。选择一个镜像链接并开始下载。

Windows (x86, 32-bit), MSI Installer (mysql-5.1.51-win32.msi)	5.1.51	105.9M	Download
Windows (x86, 32-bit), MSI Installer Essentials - Recommended (mysql-essential-5.1.51-win32.msi)	5.1.51	38.9M	Download
Windows (x86, 64-bit), MSI Installer Essentials - Recommended (mysql-essential-5.1.51-win64.msi)	5.1.51	31.5M	Download
Windows (x86, 64-bit), MSI Installer (mysql-5.1.51-win64.msi)	5.1.51	99.0M	Download

图A-12 Mysql Windows下载链接



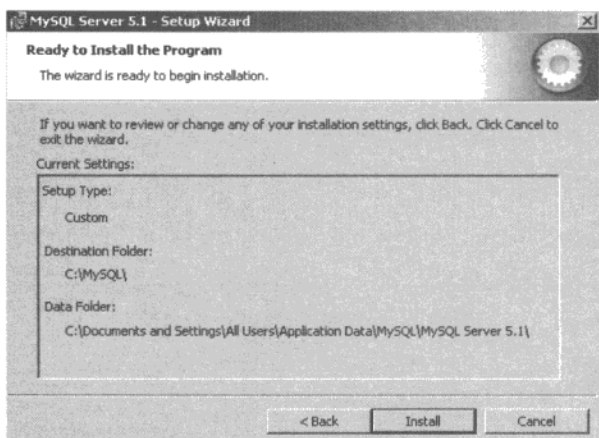
图A-13 Mysql下载安装程序登录窗口

下载完成后，打开.zip文件并运行安装文件来安装MySQL。在初始的欢迎窗口中，只需单击“Next”。

“Setup Type”窗口中显示了典型安装或自定义安装的选项。单击“Custom”单选按钮，然后单击“Next”。这样便可以在你选择的目录中安装MySQL。

显示“Custom Setup”窗口之后，单击“Change”并键入MYSQL_HOME目录。为简便起见，我们将在C:\mysql下安装MySQL文件。在后面的章节中，我们将此路径称为MYSQL_HOME。

单击“OK”，然后单击“Next”。到达“Ready to Install the Program”窗口（图A-14）之后，单击“Install”并观看MySQL安装。如果系统出现其他屏幕，请单击“Next”。

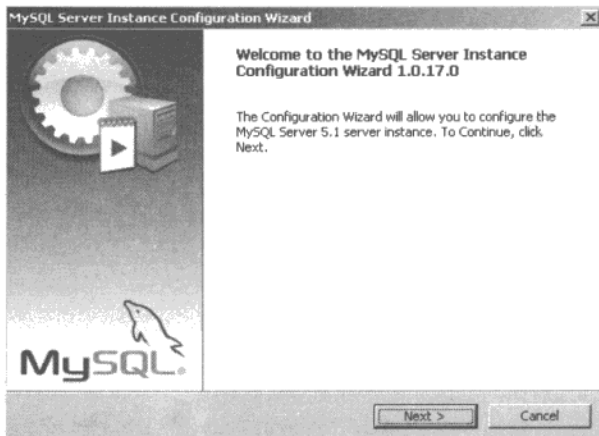


图A-14 MySQL已准备好安装

如果在MySQL安装过程中没有任何错误，则表示MySQL已安装在我们的计算机上并且将弹出一个配置窗口。现在让我们完成配置MySQL实例的步骤。

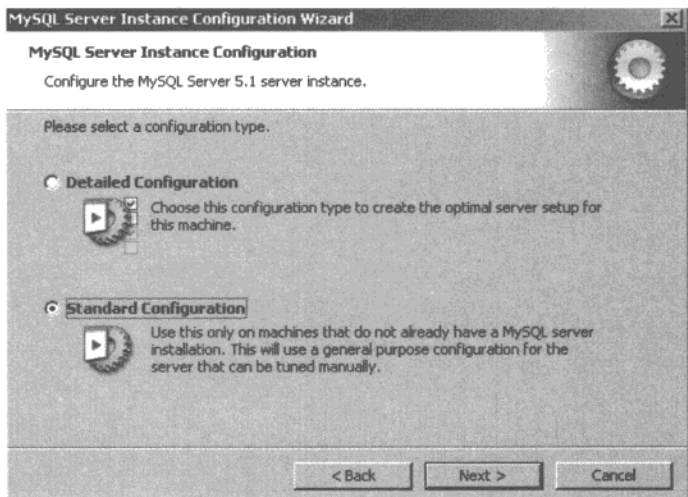
配置 MySQL

配置MySQL只需要一分钟时间。在初始的第一个窗口上，单击“Next”开始配置过程（图A-15）。



图A-15 MySQL配置欢迎窗口

在图A-16所示的窗口中，单击“Standard Configuration”按钮以加快配置过程，然后单击“Next”。



图A-16 MySQL配置类型窗口

在下一个屏幕中，我们需要接受已为我们选定的默认设置“Install as Windows Service”（如图A-17所示），然后单击“Next”。



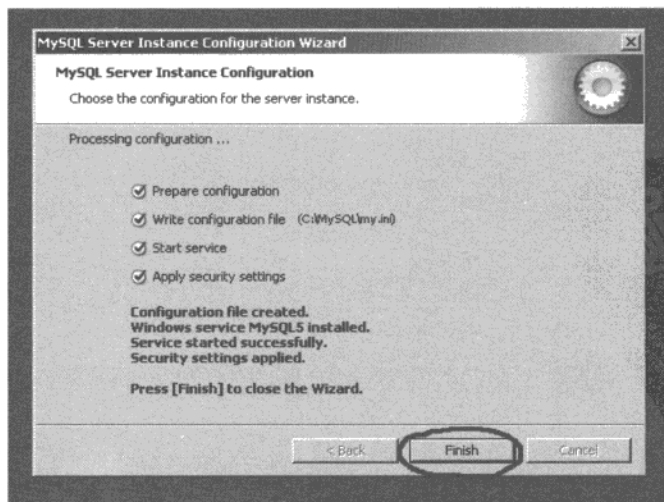
图A-17 MySQL 配置Windows选项窗口

最后一个窗口是为安装设置密码。在这两个字段中都输入一个密码，保留“Modify Security Settings”选中，然后单击“Next”（图A-18）。



图A-18 MySQL 配置安全选项窗口

最后，单击“Execute”按钮，注意观察出现选中标记。如果安装和配置成功完成，则你会看到一个MySQL窗口，该窗口中在四个小气泡中的选中标记，表示没有错误，如图A-19所示。恭喜——我们已经完成安装。单击“Finish”并休息一下。



图A-19 MySQL 处理配置窗口

A.3 安装 PHP

PHP安装程序可以从www.php.net下载。而且与目前为止的大多数下载一样，有下载Unix安装程序或Windows可执行文件的选项。在安装PHP时，如果你使用的是Windows环境，则下载.zip文件，而不是Windows .exe安装程序。若要查找.zip文件，请单击“Stable Releases”标题下页面右侧“Current PHP 5 Stable”链接并查找“Windows Binaries”标题下的.zip程序包。单击该链接之后，将为你呈现一个可从中下载的镜像列表。单击你所在国家中的镜像链接即可开始下载。.zip文件包含增加的扩展以及我们需要的库。

PHP安装程序完成下载之后，将这些文件解压缩到你选择的目录。我在目录C:\PHP5\下安装这些文件，我建议你也采用同样的目录，以便使用整本书。

设置 PHP5 和 MySQL

解压缩完所有文件之后，我们需要修改Apache安装的httpd.conf文件。转到目录APACHE_HOME/conf并打开该文件。

该文件允许手动配置Apache。由于默认情况下Apache不知道如何解释PHP文件，因此，我们需要告诉Apache当用户对.php文件进行请求时它需要使用哪个解释器。如果我们在安装过程中错过了此步骤，可以随时尝试在任何浏览器上加载.php文件，该浏览器将提示我们下载该文件或在某些情况下只在页面上显示PHP代码。因此，让我们告诉Apache使用哪个解释器。在本例中，解释器就是我们所安装的PHP引擎。在该文件的结尾，键入以下文本：

```
AddType application/x-httpd-php .php
PHPIniDir "C:/PHP/"
LoadModule php5_module "C:/PHP/php5apache2_2.dll"
```

注意 通过在Windows命令窗口（运行cmd）或Unix命令窗口中使用命令apachectl start、apachectl stop或apachectl restart，我们可以启动、停止或重新启动Apache。当然，你还可以单击系统托盘中的“Apache Monitor”图标，然后选择“Apache2 Stop”或“Apache2 Restart”。

保存该文件并重新启动Apache。通过使用Apache Service Monitor来完成该操作。显示Apache Service Monitor窗口，单击右侧的“Restart”按钮。如果一切运行正常，你将看到一个成功消息。

若要测试PHP安装，参考下一节“Creating a phpinfo() Script”创建一个phpinfo PHP脚本。如果一切运行正常，那么此时你应该会看到Apache正在将PHP转换为类似于图A-20的网页。

A.4 创建 phpinfo()脚本


在整本书中，我们都引用了一个phpinfo()脚本，该脚本允许你检查是否正确安装了PHP扩展以及是否正确设置了PHP配置。该PHP脚本包含不到三行代码，如代码清单A-1所示。

代码清单A-1 .phpinfo()脚本代码

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

若要创建phpinfo()脚本, 打开一个文本文件并复制代码清单A-1中所示的代码。将该文件另存为info.php并将其放置在你的Web服务器中。如果你已经按照附录中Apache的安装说明执行操作, 请将该文件放置在APACHE_HOME/htdocs位置中。

在浏览器中加载http://localhost/info.php文件, 以查看当前安装的PHP使用的PHP设置。你将会看到一个HTML页面, 该页面中包含如图A-20所示的信息。



System	Windows NT ARMANDO1 5.1 build 2600 (Windows XP Professional Service Pack 3) i586
Build Date	Jul 21 2010 20:25:38
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--disable-isapi" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=D:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=.obj" "--enable-com-dotnet" "--with-mcrypt=static"

图A-20 PHP信息页

向下滚动该页, 查看不同的配置设置以及当前PHP安装使用的模块。你可能会发现一个已经安装了模块和服务器设置的列表, 以及该页中加载php.ini文件的位置。

A.5 安装 PECL

PECL是一个PHP扩展资源库, 包含一些PHP开发人员可用的最佳PHP扩展。从加密到XML解析, 完整的扩展列表可以在其官方网站http://pecl.php.net上找到。

本书中概述的大部分扩展都可以使用PECL及其命令行工具快速、轻松地安装。在这里, 我们将通过初始安装PHP的一个程序包资源库PEAR在Windows上安装PECL。

安装PECL的起始步骤是安装PEAR。如前所述, PEAR是PHP的一个程序包资源库。若要开始安装PEAR, 需要在系统上安装PHP。到现在为止, 你应该已在你的PHP_HOME目录中安装了PHP。

在“开始→运行”窗口中键入cmd来打开一个命令行窗口。该窗口打开后, 导航到安装PHP的目录, 并键入php go-pear.bat。这为你安装PEAR以及PECL。如果安装过程中没有发生

任何错误，那么现在PECL应该已安装完成。若要验证此安装，请在命令行窗口中键入`pecl`。应该会看到如图A-21所示的帮助菜单。

```

Commands:
build          Build an Extension From C Source
bundle        Unpacks a Pecl Package
channel-add    Add a Channel
channel-alias  Specify an alias to a channel name
channel-delete Remove a Channel From the List
channel-discover Initialize a Channel from its server
channel-info   Retrieve Information on a Channel
channel-login  Connects and authenticates to remote channel server
channel-logout Logs out from the remote channel server
channel-update Update an Existing Channel
clear-cache    Clear Web Services Cache
config-create  Create a Default configuration file
config-get     Show One Setting
config-help    Show Information About Setting
config-set     Change Setting
config-show    Show All Settings
convert        Convert a package.xml 1.0 to package.xml 2.0 format
cvsdiff        Run a "cvs diff" for all files in a package
cvstag         Set CVS Release Tag
download       Download Package
download-all  Downloads each available package from the default channel
info           Display information about a package
install        Install Package
list           List Installed Packages In The Default Channel

```

图A-21 PECL帮助菜单

在Linux上安装Apache、MySQL、PHP和PECL

在附录A中，我们介绍了如何在Windows上安装完整的WAMP栈。在本附录中，我们将介绍如何在Linux上安装同样的LAMP栈。

由于Linux有很多特色和不同版本，因此我们将只重点介绍两个最大的发行版：

- 采用Fedora 14格式且基于RPM/yum，它代表了基于Red Hat、CentOS以及Fedora的发行版；
- 基于Debian (Ubuntu 10.10)。

通过主要在两个平台上安装，我们希望涵盖基于Linux的大部分开发工作站设置，因为本书主要是面向开发人员的。

B.1 Fedora 14

Fedora是一个基于RPM/yum的Linux发行版，由Fedora Project开发并由Red Hat提供支持。Fedora与Red Hat RHEL和CentOS发行版使用了同一种打包方法和文件系统布局，通常在高可靠性的Web服务中使用。

Fedora 14是最新的版本，代码名称为“Laughlin”，是在2010年11月2日发布的。

可以使用代码清单B-1中所示的终端命令序列在Fedora 14上创建一个LAMP栈，该栈包含本书中所讨论的所有组件，而且还包括所描述的所有工具。

代码清单B-1 Fedora 14的安装指令

```
$su -
# Install Apache
$yum -y install httpd

# Install Mysql
$yum -y install mysql-server
$yum -y install mysql

# Install PHP
$yum -y install php
$yum -y install php-mysql
```

```

$yum -y install php-pecl-apc
$yum -y install php-gd
$yum -y install php-pecl-xdebug

# Install phpinfo page on http://localhost/phpinfo.php
$echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php

# Install Memcache and php extension
$yum -y install memcached
$yum -y install php-pecl-memcache

# Install Benchmarking and monitoring tools. Note: iostat is in sysstat package
$yum -y install apachetop
$yum -y install siege
$yum -y install sysstat

# Set services to start at boot time
$chkconfig mysqld
$chkconfig memcached
$chkconfig httpd

# Start the services.
$service httpd start
$service mysqld start
$service memcached start
$exit

```

组件版本和位置

使用上述过程在Fedora 14上安装的应用程序组件的版本如表B-1所示。在本书出版时这些版本信息都是正确的。程序包的后续更新可能会导致版本号大于所示的版本。

表B-1 Fedora 14的组件版本和配置文件位置

组 件	属 性	值
Apache	版本	Apache/2.2.17 (Fedora)
	配置目录	/etc/httpd/conf
	默认访问日志	/var/log/httpd/access_log
	默认错误日志	/var/log/httpd/error_log
	默认文档根	/var/www/html
Mysql	版本	5.1.52
	配置文件	/etc/my.cnf
	默认Socket	/var/lib/mysql/mysql.sock
	数据目录	/var/lib/mysql
	日志文件	/var/log/mysqld.log
PHP	版本	5.3.3
	默认PHP.ini文件	/etc/php.ini
	配置目录	/etc/php.d

B.2 Ubuntu 10.10

Ubuntu是一个基于Debian GNU/Linux发行格式的Linux分发版。名称Ubuntu采用来自南非的基于Bantu^①的语言，表示“humanity toward others”。

尽管也存在服务器版本，但Ubuntu主要用作桌面版本，这使它成为一个理想的开发人员工作站操作系统，通常集成了最新的开发工具和IDE。很多商用工具也是专门针对Ubuntu发行的。

Ubuntu版本10.10的代码名称为“Maverick Meerkat”，是在2010年10月10日发布的。

可以使用代码清单B-2中所示的终端命令序列在Ubuntu 10.10上创建一个LAMP栈，该栈包含本书中所讨论的所有组件，而且还包括所描述的所有工具。

代码清单B-2 Ubuntu 10.10的安装指令

```
# Install Configuration manager
$ sudo apt-get install tasksel

# Install basic LAMP stack
$ sudo tasksel install lamp-server

# Install additional PHP extensions
$ sudo apt-get install php-apc
$ sudo apt-get install php5-xdebug

# Install Benchmarking and monitoring tools. Note: iostat is in sysstat package
$ sudo apt-get install apachetop
$ sudo apt-get install sysstat
$ sudo apt-get install siege

# Install memcache and php extension
$ sudo apt-get install memcached
$ sudo apt-get install php5-memcache

# Install Pear/PecL
$ sudo apt-get install php-pear

# Create phpinfo page at http://localhost/phpinfo.php
$ sudo bash -c "echo \"<?php phpinfo(); ?>\" > /var/www/phpinfo.php"

# Restart web server to ensure all modules are loaded.
$ sudo service apache2 restart
```

B.2.1 组件版本和位置

通过上述步骤在Ubuntu 10.10上安装的应用程序组件的版本如表B-2所示。在出版时这些版本信息都是正确的。程序包的后续更新可能会导致版本号大于所示的版本。

① 南非一个小部落的名字。——译者注

表B-2 Ubuntu 10.10的组件版本和配置文件位置

组 件	属 性	值
Apache	版本	Apache/2.2.17 (Fedora)
	配置目录	/etc/httpd/conf
	默认访问日志	/var/log/httpd/access_log
	默认错误日志	/var/log/httpd/error_log
	默认文档根	/var/www/html
Mysql	版本	5.1.52
	配置文件	/etc/my.cnf
	默认Socket	/var/lib/mysql/mysql.sock
	数据目录	/var/lib/mysql
	日志文件	/var/log/mysqld.log
PHP	版本	5.3.3
	默认 PHP.ini文件	/etc/php.ini
	配置目录	/etc/php.d

B.2.2 Tasksel

代码清单B-2中的安装脚本使用一个名为“tasksel”的工具，该工具是Ubuntu特有的工具。Tasksel允许安装预先定义的一组程序包，该程序包包含特殊的配置，其中一个配置是“LAMP server”。如果你想查看此配置中所安装的准确的程序包列表，请执行以下命令：

```
$ tasksel --task-packages lamp-server
```

B.3 PECL

正如附录A中提到的，PECL是一个PHP扩展资源库，它包含一些PHP开发人员可用的最佳PHP扩展。从加密到XML解析，扩展的完整列表可以在其官方网站<http://pecl.php.net>上找到。

Unix用户很幸运。如前所述，若要使用PECL，需要安装PEAR，但对于Unix用户，安装PHP版本4.3.0或更高版本时会自动安装PEAR。如果你确定没有安装PEAR，请从<http://pear.php.net/go-pear> as go-pear.php下载go-pear.php文件并运行命令php go-pear.php在命令窗口中运行该脚本。安装完成后，在命令窗口中运行命令pear和pecl验证PEAR和PECL是否已安装。

版 权 声 明

Original English language edition, entitled *Pro PHP Application Performance: Turning PHP Web Projects for Maximum Performance* by Armando Padilla, Tim Hawkins, published by Apress, 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705 USA.

Copyright © 2010 by Armando Padilla and Tim Hawkins. Simplified Chinese-language edition copyright © 2011 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由 Apress L.P. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

[General Information]

书名=高性能PHP应用开发

作者=(美)帕蒂拉,(美)霍金斯著

页数=178

出版社=北京市:人民邮电出版社

出版日期=2011.11

SS号=12865980

DX号=000008194868

URL=<http://book1.duxiu.com/bookDetail.jsp?dxNumber=000008194868&d=4F539A0581119449720728BFC2B1BCDA>