

配置Nginx的深入指南

# 精通Nginx

## Mastering Nginx

[ 瑞士 ] Dimitri Aivaliotis 著  
陶利军 译

# 精通 Nginx

[瑞士] Dimitri Aivaliotis 著

陶利军 译

人 民 邮 电 出 版 社

北 京

## 版权声明

Copyright ©2013 Packt Publishing. First published in the English language under the title *Mastering NGINX*  
All rights reserved.

本书由英国 **Packt Publishing** 公司授权人民邮电出版社出版。未经出版者书面许可，对本书的任何部分不得以任何方式或任何手段复制和传播。

版权所有，侵权必究。

- 
- ◆ 著 [瑞士] Dimitri Aivaliotis
  - 译 陶利军
  - 责任编辑 陈冀康
  - 责任印制 张佳莹
  - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
  - 邮编 100164 电子邮件 315@ptpress.com.cn
  - 网址 <http://www.ptpress.com.cn>
  - 北京 印刷有限公司印刷
  - ◆ 开本: 800×1000 1/16
  - 印张: 17.25
  - 字数: 340 千字 2015 年 2 月第 1 版
  - 印数: 1 – 0000 册 2015 年 2 月北京第 1 次印刷

著作权合同登记号 图字: 01-2013-7461 号

---

定价: 00.00 元

读者服务热线: (010)81055410 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京崇工商广字第 0021 号

# 内容提要

Nginx 是一个高性能的轻量级 Web 服务器，本书从配置文件的角度出发，介绍了多种关于 Nginx 配置的技巧。

本书以模块化风格写成，几乎每一章都是一个独立的模块，读者将能够自由地在各个模块间切换阅读。全书分两部分，第一部分用 8 章内容介绍了安装 Nginx 及第三方模块、配置指南、使用 mail 模块、Nginx 作为反向代理、反向代理高级话题、Nginx Http 服务器、Nginx 的开发以及故障排除技巧；第二部分用 4 个附录的形式介绍了指令参考、Rewrite 规则指南、Nginx 社区以及 Solaris 系统下的网络调优。

本书适合在安装和配置服务器方面有经验的系统管理员或系统工程师，阅读本书不需要任何 Nginx 使用经验，相信这本书会帮助你更好地完成任务。

# 作者简介

**Dimitri Aivaliotis** 在瑞士苏黎世的一个主机托管商担任系统架构师。他的职业生涯，从为学校构建基于 Linux 的计算机网络到为银行构建双数据中心的高可用性基础设施和在线门户网站。他在解决客户问题上已经花费了 10 年的时间，并且在这条路上发现了 Nginx。他每天使用 Nginx 为他的客户提供 Web 访问、代理和流媒体服务。

Dimitri 以最优异的成绩获得了伦斯勒理工学院的理科学士，并且获得了佛罗里达州立大学管理信息系统的理科硕士。

这是他的第一本书。

---

我要感谢 John Blackwell 和 Phil Margolis 阅读了早期的手稿草稿。在这本书上他们的批评和建议对我帮助很大。我还要感谢技术审评提供有益的意见反馈，指出我犯的错误。任何遗留的错误都是我的。

Packt Publishing 团队对该项目从一开始就给予了真诚的支持，在过往最后期限的黑暗日子里，作为一个作者他们给了我鼓励。

Nginx, Inc. 团队的支持填补了我理解 Nginx 如何工作的空白，没有他们我就不能够写出这本书。

特别衷心地感谢我的家人，我的妻子和孩子不得不接受我将许多时间花在写作上，在此期间他们的耐心令我感动。

---

# 审稿人简介

**Yasir Adnan** 居住在孟加拉国的首都达卡，他是一名计算机科学专业的学生，也是一个自由程序员。他曾在移动和 Web 应用程序方面工作。他现在主要开发手机应用程序，他的邮件地址为 [yasiradnan@outlook.com](mailto:yasiradnan@outlook.com)。

**Andrew Alexeev** 是 Nginx, Inc. 的创始人之一，Nginx, Inc. ——Nginx web 服务器背后的高性能 web 公司。在加入 Nginx, Inc. 之前，2011 之初，Andrew 曾在互联网行业和各种各样企业的 ICT 部门任职。Andrew 持有 St. Petersburg Electrotechnical University 的电子学学位证书和 Antwerp Management School 的 MBA 证书。

**Antonio P.P. Almeida** (@perusio) 自从他在破旧的基于迅驰 1.3 GHz 的便携式电脑上开发 Drupal 后，他就着迷于 Nginx，Apache 对资源的胃口必然导致了 Nginx。他一直学习在各种可能的应用程序中如何让 Nginx 获取更大功效，特别是 Nginx 配置语言的所有微妙之处。他生活和工作在巴黎，除了 Nginx 外，他还着迷于难以理解的意大利中世纪音乐、电影以及如何使 Drupal 更好。

**Rainer Duffner** 获得了德国康斯坦茨 University of Applied Science 大学信息系统的学位，最近在 EveryWare AG 就职于系统工程师，这是一个帮助客户将 FreeBSD、Linux 和 Solaris 服务器获取最大功效的公司。

他居住在瑞士苏黎世的一个小镇上，喜欢在空闲的时间里骑山地自行车环绕苏黎世和瑞士的山脉。

---

我要感谢 Dimitri 利用时间很好地检查这本书，一直以来他都是一个很好的资源。

---

# 前言

Nginx 是一个高性能的 Web 服务器，在它的设计上使用的系统资源非常少。有很多 how-to 和示例配置文件在互联网上出现，这会澄清 Nginx 配置的浑水，这样做你将会学习到在各种环境中如何调整 Nginx，以及一些配置模糊的选项的配置，以便设计一个符合你需求的配置文件。

在你已经理解了如何根据自己的需求来构建一个配置文件后，你就不再需要复制-粘贴配置片段了。这是一个过程，而且会有曲折，但是本书中有关技巧的解释，会使你觉得手写 Nginx 配置文件是一件很舒服的事情。万一事情不像你期望的那样工作，你将能够独立调试该问题，或者至少能够寻求帮助，而不会觉得自己好像都没有尝试过。

这本书是以现代风格所写，这种设计帮助你尽可能快地获取信息。几乎每一章都是一个独立的模块，你可以根据你的需要自由地跳到任何地方获取更深入的特定主题。如果你觉得错过了某些主要的东西，那么你可以返回去读取前面的章节。它们在这种方式下构建以帮助你的配置文件一步步成长。

## 本书涵盖的内容

第 1 章，安装 Nginx 和第三方模块，教你如何在选择的操作系统上安装 Nginx，以及在你的安装中如何包含第三方模块。

第 2 章，配置向导，讲解了 Nginx 的配置文件格式，你将会学到每一个不同区段的配置，如何配置全局参数，及 location 的用处。

第 3 章，使用 Mail 模块，探索 Nginx 的邮件代理模块，详细介绍了配置的方方面面。在本章中还有一个认证服务的代码例子。

第 4 章，Nginx 作为反向代理，介绍反向代理的原理，并且描述 Nginx 如何充当该角色。

第 5 章，反向代理高级话题，深入研究使用 Nginx 作为反向代理解决可伸缩及性能问题。

第 6 章，Nginx Http 服务器，描述如何使用各种模块，包括通过 Nginx 解决常见的 Web 问题。

第 7 章，Nginx 与开发者，展示了 Nginx 如何与你的应用程序集成，以便更快地将内容交付给你的用户。

第 8 章，故障排除技术，研究了一些常见的配置文件，一旦出现问题如何调试，及调优性能的一些建议。

附录 A，指令参考，提供了一个方便的配置指令参考，这些指令贯穿了全书，也有一些以前没有覆盖到的指令。

附录 B，Rewrite 规则向导，描述如何使用 Nginx 的 rewrite 规则模块，并且描述了将 Apache 格式的 rewrite 规则转换为 Nginx 的 rewrite 规则的一些步骤。

附录 C，团体，介绍了可以在线上搜寻到的更多资源。

附录 D，针对 Solaris 操作系统在网络下的调优，详述了有必要存在的 Solaris 10 版本以上系统下的网络调优。

## 使用这本书你需要做的

任何现代 Linux PC 都能够充分地运行本书中的实例代码。用户代码实例在每一章都给出了安装操作指南，基本上归纳如下。

- ◆ 构建环境：编译器、头文件，等等。
- ◆ Nginx：最新版本较好。
- ◆ Ruby：最好从 <https://rvm.io> 安装。
- ◆ Perl：默认版本较好。

## 谁需要这本书

这本书适用于在安装和配置服务器方面有经验的系统管理员或系统工程师，以满足特定的需求。你不需要已经有使用 **Nginx** 的经验。

## 约定

在本书中，你将会看到一些风格的文本，便于区别两种不同的信息。这里有一些这些格式的例子，并且有解释说明。

代码字符以下列格式显示: "Nginx will attempt to build a dependent library statically if you include a `—with-<library>=<path>` option to configure."

代码块设置如下。

```
$ export BUILD_DIR=`pwd`  
$ export Nginx_INSTALLDIR=/opt/nginx  
$ export VAR_DIR=/home/www/tmp  
$ export LUAJIT_LIB=/opt/luajit/lib  
$ export LUAJIT_INC=/opt/luajit/include/luajit-2.0
```

在我们想提请你注意代码块的一个特定部分时，对有关的线或项目设置粗体。

```
$ export BUILD_DIR=`pwd`  
$ export Nginx_INSTALLDIR=/opt/nginx  
$ export VAR_DIR=/home/www/tmp  
$ export LUAJIT_LIB=/opt/luajit/lib  
$ export LUAJIT_INC=/opt/luajit/include/luajit-2.0
```

任何命令行输入或者输出书写如下。

```
$ mkdir $HOME/build  
$ cd $HOME/build && tar xzf nginx-<version-number>.tar.gz
```

新的术语和重要的字以粗体显示。你在屏幕上看到的字，例如，菜单或者对话框，会出现类似这样的版本: "clicking the **Next** button moves you to the next screen"。



警告或者重要的事项出现在这样的盒子中。



提示和技巧以这种方式显示。

## 读者反馈

从读者来的反馈总是受欢迎的，它让我们知道你对本书的想法——什么是你喜欢的，什么是不喜欢的。对于我们而言读者反馈非常重要，它使我们的课题发挥更大的作用。

向我们发送一般的反馈，只需要简单地发送电子邮件到 [feedback@packtpub.com](mailto:feedback@packtpub.com)，并且在邮件的主题部分提及本书的名称就可以了。

如果你有擅长的主题，并且你对写作或者投稿感兴趣，那么你可以看看我们的作者向导 [www.packtpub.com/authors](http://www.packtpub.com/authors)。

## 客户支持

现在你自豪地成为 Packt 书的所有人了，对于你的购买我们有更多的事情帮助你。

### 下载示例代码

你购买所有 Packt 书的示例代码都可以从 <http://www.PacktPub.com> 下载，如果你从别的地方购买了这本书，你可以访问 <http://www.PacktPub.com/support>，并且注册，那么代码会以邮件的形式直接发送给你。

### 勘误表

尽管我们非常小心地确保我们内容的准确性，但是错误还是会发生的。如果你在我们的任何一本书中发现了错误——可能是文本的错误，也可能是代码的错误，并把它向我们报告，我们将非常感谢。通过这样，你可以使其他读者免于读取错误，并且帮助我们在本书的后续的版本中更正。如果你想查找任何勘误表，请通过 <http://www.packtpub.com/support> 访问它

们，选择你要找的书的名称，点击 **errata submission form** 链接，就会进入本书详细的勘误表。一旦你的勘误表被校验，那么你的提交将会被接受，并且在我们的网站上的勘误表将会被更新，或者添加到任何存在的勘误表中，在标题部分的勘误表下。通过从 <http://www.packtpub.com/support> 选择标题，任何存在的勘误表都可以被查看。

## 盗版

在互联网上，对于有版权材料的盗版行为一直是一个持续的跨越全媒体的问题。在 Packt，我们非常重视版权和许可，如果你在互联网上偶然发现以任何形式非法拷贝我们的文字的地方，请立即提供给我们地址或者是网站的名字，以便我们能够追究补救办法。

请联系我们 [copyright@packtpub.com](mailto:copyright@packtpub.com)，并且附上涉嫌盗版的材料的链接。

我们非常感谢你帮助保护我们的作者，并且我们会持续给读者带来有价值的内容。

## 问题

如果有任何方面关于本书的问题，你可以联系我们 [questions@packtpub.com](mailto:questions@packtpub.com)，我们将会尽量解决。

# 目录

- 第 1 章 安装 Nginx 及第三方模块 ..... 1
  - 1.1 使用包管理器安装 Nginx ..... 2
    - 1.1.1 CentOS ..... 2
    - 1.1.2 Debian ..... 3
  - 1.2 从源代码安装 Nginx ..... 3
    - 1.2.1 准备编译环境 ..... 4
    - 1.2.2 从源代码编译 ..... 4
  - 1.3 配置 Web 或者 mail 服务器 ..... 6
    - 1.3.1 邮件代理的配置选项 ..... 6
    - 1.3.2 指定路径的配置选项 ..... 7
  - 1.4 使用各种模块 ..... 8
  - 1.5 查找并安装第三方模块 ..... 11
  - 1.6 组合在一起 ..... 12
  - 1.7 总结 ..... 14
- 第 2 章 配置指南 ..... 15
  - 2.1 基本配置格式 ..... 15
  - 2.2 Nginx 的全局配置参数 ..... 16
  - 2.3 使用 include 文件 ..... 17
  - 2.4 Http 的 server 部分 ..... 18
    - 2.4.1 客户端指令 ..... 18
    - 2.4.2 文件 I/O 指令 ..... 19
    - 2.4.3 Hash 指令 ..... 20

2.4.4	Socket 指令 .....	21
2.4.5	示例配置文件 .....	22
2.5	虚拟 server 部分 .....	22
2.6	Locations——where, when, how .....	26
2.7	mail 的 server 部分 .....	28
2.8	完整的样本配置文件 .....	30
2.9	总结 .....	31
<b>第 3 章</b>	<b>使用 mail 模块 .....</b>	<b>33</b>
3.1	基本代理服务 .....	33
3.1.1	POP3 服务 .....	35
3.1.2	IMAP 服务 .....	36
3.1.3	SMTP 服务 .....	36
3.1.4	使用 SSL/TLS .....	37
3.1.5	完整的 mail 示例 .....	40
3.2	认证服务 .....	42
3.3	与 Memcached 结合 .....	49
3.4	解释日志文件 .....	52
3.5	操作系统的限制 .....	54
3.6	总结 .....	55
<b>第 4 章</b>	<b>Nginx 作为反向代理 .....</b>	<b>57</b>
4.1	反向代理简介 .....	58
4.1.1	代理模块 .....	59
4.1.2	upstream 模块 .....	64
4.2	Upstream 服务器的类型 .....	67
4.2.1	单个 upstream 服务器 .....	67
4.2.2	多个 upstream 服务器 .....	69
4.2.3	非 Http 型 upstream 服务器 .....	70
4.3	将 if 配置转换为一个更现代的解释 .....	72
4.4	使用错误文件来处理 upstream 的问题 .....	76
4.5	确定客户端的真实 IP 地址 .....	78
4.6	总结 .....	78

---

<b>第 5 章 反向代理高级话题 .....</b>	<b>81</b>
5.1 实现安全隔离 .....	82
5.1.1 使用 SSL 对流量进行加密 .....	82
5.1.2 使用 SSL 进行客户端身份验证 .....	84
5.1.3 基于原始 IP 地址阻止流量 .....	86
5.2 孤立应用程序的扩展 .....	89
5.3 反向代理服务器的性能调优 .....	92
5.3.1 缓冲 .....	92
5.3.2 缓存 .....	94
5.3.3 压缩 .....	100
5.4 总结 .....	103
<b>第 6 章 Nginx Http 服务器 .....</b>	<b>105</b>
6.1 Nginx 的系统结构 .....	106
6.2 Http 的核心模块 .....	106
6.2.1 server .....	107
6.2.2 日志 .....	108
6.2.3 文件查找 .....	111
6.2.4 名字解析 .....	113
6.2.5 客户端交互 .....	115
6.3 使用 limit 指令防止滥用 .....	117
6.4 约束访问 .....	121
6.5 流媒体文件 .....	126
6.6 预定义变量 .....	127
6.7 使用 Nginx 和 PHP-FPM .....	129
6.8 将 Nginx 和 uWSGI 连接在一起工作 .....	141
6.9 总结 .....	143
<b>第 7 章 Nginx 的开发 .....</b>	<b>145</b>
7.1 缓存集成 .....	145
7.1.1 应用程序没有缓存 .....	146
7.1.2 使用数据库缓存 .....	148
7.1.3 使用文件做缓存 .....	150
7.2 动态修改内容 .....	154

---

7.2.1	addition 模块	154
7.2.2	sub 模块	155
7.2.3	xslt 模块	156
7.3	使用服务器端包含 (Server Side Includes, SSI)	157
7.4	在 Nginx 中的决策	159
7.5	创建安全链接	163
7.6	生成图像	165
7.7	跟踪网站的访问者	169
7.8	防止意外的代码执行	170
7.9	总结	171
<b>第 8 章</b>	<b>故障排除技巧</b>	<b>173</b>
8.1	分析日志文件	173
8.1.1	错误日志文件格式	174
8.1.2	错误日志文件条目实例	175
8.2	配置高级日志记录	178
8.2.1	调试日志记录	178
8.2.2	使用访问日志文件进行调试	185
8.3	常见的配置错误	188
8.3.1	使用 if 取代 try_files	188
8.3.2	使用 if 作为主机名切换	189
8.3.3	不使用 server 部分的配置追求更好的效果	190
8.4	操作系统限制	192
8.4.1	文件描述符限制	192
8.4.2	网络限制	194
8.5	性能问题	195
8.6	使用 Stub Status 模块	197
8.7	总结	198
<b>附录 A</b>	<b>指令参考</b>	<b>199</b>
<b>附录 B</b>	<b>Rewrite 规则指南</b>	<b>241</b>
B.1	介绍 rewrite 模块	241
	创建新的 rewrite 规则	245
B.2	转换 Apache 的重写规则	247

---

B.2.1 规则 #1: 使用 <code>try_files</code> 替代目录和文件存在性检测	247
B.2.2 规则 #2: 使用 <code>location</code> 替代匹配 <code>REQUEST_URI</code>	248
B.2.3 规则 #3: 使用 <code>server</code> 替代匹配 <code>R Http_HOST</code>	249
B.2.4 规则 #4: 变量检查使用 <code>if</code> 替代 <code>RewriteCond</code>	251
B.3 总结	252
附录 C <b>Nginx</b> 的社区	253
C.1 邮件列表	253
C.2 IRC 频道	254
C.3 Web 资源	254
C.4 撰写好的 bug 报告	254
C.5 总结	255
附录 D <b>Solaris</b> 系统下的网络调优	257



# 第 1 章

## 安装 Nginx 及第三方模块

Nginx 最初的设计是成为一个 Http 服务器，一个能够解决 C10K 问题的 Http 服务器。关于 C10K 这个问题，Daniel Kegel 在 <http://www.kegel.com/c10k.html> 页面有具体描述，设计一个同时连接处理 10000 连接的 web 服务器。为了实现这个目标，Nginx 通过基于事件的处理机制并且操作系统也要使用相应的事件机制，便可以解决 C10K 问题。

在我们开始探索如何配置 Nginx 之前，首先我们要安装它，这一章将详细讲述如何安装 Nginx，以及如何获取正确的模块并且安装和配置它们。Nginx 是模块化设计的，并且有非常丰富的社区第三方模块，它们的设计者通过创建这些模块为 core Nginx 增添了功能，我们可以在编译安装 Nginx 时将它们添加到 Nginx 服务器。

在本章中，我们涉及到以下内容。

- ◆ 使用包管理器安装 Nginx。
- ◆ 通过源代码安装 Nginx。
- ◆ 配置 Nginx 为 Web 或者 Mail 服务器。
- ◆ 使用各种模块。
- ◆ 查找并安装第三方模块。
- ◆ 组合在一起。

## 1.1 使用包管理器安装 Nginx

使用包管理器安装 Nginx 的机会是你所使用的操作系统已经提供了 Nginx 的安装包。使用包管理器安装 Nginx 的方法很简单，只需要使用包管理器安装命令就可以了。

◆ Linux (基于 deb)

```
sudo apt-get install nginx
```

◆ Linux (基于 rpm)

```
sudo yum install nginx
```

◆ FreeBSD

```
sudo pkg_install -r nginx
```



命令 `sudo` 表示的是通过操作系统中的超级用户 ('root') 权限执行的命令。如果操作系统支持 **RBAC (Role-based access control)**，那么可以用一个不同的命令，例如 'pfexec'，来达到同样的目的。

通过上面的命令来安装的 Nginx 都会被安装到操作系统的标准位置下。如果使用操作系统的安装包安装 Nginx，那么通过上面的命令来安装是最好的方式。

Nginx core 团队也提供了二进制的标准版本，可以从 <http://nginx.org/en/download.html> 页面下载可用的版本。没有发布 Nginx 二进制版本的系统用户（例如，CentOS），可用使用下面的指导来安装预测试、预编译二进制版本。

### 1.1.1 CentOS

通过创建下面的文件在系统中添加 Nginx 仓库的 yum 配置。

```
sudo vi /etc/yum.repos.d/nginx.repo
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/6/$basearch/
gpgcheck=0
enabled=1
```

然后通过执行以下命令来安装 Nginx。

```
sudo yum install nginx
```

也可以按照前面介绍的 URL 下载 Nginx 发布版本安装。

### 1.1.2 Debian

通过从 [http://nginx.org/keys/nginx\\_signing.key](http://nginx.org/keys/nginx_signing.key) 下载 Nginx 并安装签名 key，将该签名 key 添加到系统的 apt keyring 中。

```
sudo apt-key add nginx_signing.key
```

将 nginx.org 仓库追加到 `/etc/apt/sources.list` 文件末尾。

```
vi /etc/apt/sources.list  
deb http://nginx.org/packages/debian/ squeeze nginx  
deb-src http://nginx.org/packages/debian/ squeeze nginx
```

然后执行下面的命令安装 Nginx。

```
sudo apt-get update  
sudo apt-get install nginx
```

如果所使用的操作系统在它可用的安装包中没有包括 Nginx，或者是所包含的版本太旧不能满足需要，或者是 [nginx.org](http://nginx.org) 没有提供所需要的安装包，或者是你想使用“development”版本的 Nginx，那么可以使用从源代码编译的方法来安装 Nginx。

## 1.2 从源代码安装 Nginx

Nginx 代码提供了两个独立的下载分支——标准版和开发版。开发分支是一个正处于积极开发状态的版本。在这个版本中将会有一些新的功能被集成到其中，在标准版中是找不到这些功能的。当一个“开发”版被发布时，它会经历同样的 QA 和作为标准版的一组类似功能的测试。因此无论哪一个分支都可以用于生产环境。两者主要的不同在于对第三方模块的支持。在开发版本里内部的 API 可能会改变，而标准版本却保持不变，因此为了向下兼容第三方模块，在标准版本中第三方模块都可以有效使用。

## 1.2.1 准备编译环境

为了从源代码编译 Nginx，系统需要具备某些必要的条件。除了编译器之外，如果想启用 SSL 支持和能够使用 `rewrite` 模块，那么还需要提供相应的 **OpenSSL** 和 **PCRE (Perl Compatible Regular Expressions)** 库及开发头文件。这依赖于系统，也有可能系统中这些必要条件已经被默认安装了，如果没有安装，那么需要从其安装包安装或者是从源码下载并解压安装，但是要在 Nginx 的配置文件中指定它们在系统中安装的位置。

如果在配置文件中使用了 `-with-<library>=<path>` 选项，那么 Nginx 将试图建立一个静态的依赖库。如果你想的是 Nginx 不依赖于系统的任何其他部分，也可能是你想多榨取一点 nginx 二进制额外的性能，那么你可能会使用构建静态库的做法。如果你使用的外部库的功能只能从某个版本起（例如，NPN[Next Protocol Negotiation] TLS 扩展从 OpenSSL 1.0.1 版本有效），那么你就不得不将其指定到特定版本解压后的源代码路径。

根据你自己的喜好，可能还会提供其他的、可选的安装包。这些安装包包括 MD5 和 SHA-1 以支持哈希算法、zlib 压缩库、libatomic 库。在 Nginx 中，很多地方会使用到哈希算法，例如，为了计算 URI 哈希从而计算缓存 key。zlib 压缩库被用于投递 gzip 压缩的内容。如果 `atomic_ops` 库有效，那么 Nginx 将会使用它实现自动内存更新操作，以便实现高性能的内存锁定代码。

## 1.2.2 从源代码编译

可以从 <http://nginx.org/en/download.html> 地址下载 Nginx，从该页面找到 `.tar.gz` 或者 `.zip` 格式的下载分支，按照下面的步骤将下载的安装包解压到一个临时的目录中。

```
$ mkdir $HOME/build
$ cd $HOME/build && tar xzf nginx-<version-number>.tar.gz
```

使用下面的命令来配置 Nginx。

```
$ cd $HOME/build/nginx-<version-number> && ./configure
```

然后使用下面的命令编译并且安装。

```
$ make && sudo make install
```

下表的配置选项将帮助您设计出自己的 Nginx 二进制。这些选项对 Nginx 都是有效的，模块可以被独立激活。

在编译自己的二进制 Nginx 时，你会有很大的灵活性来包含你仅使用的功能。你已经说定使用哪个用户运行 Nginx 了吗？你要使用默认的 logfile 位置，以便不用在 Nginx 的配置文件明确地说明它们吗？表 1-1 所示是配置选项列表，通过它来帮助你设计你自己的 nginx 命令。

表 1-1 通用配置选项

选项	解释
--prefix=<path>	Nginx 安装的根路径，所有其他的路径都要依赖于该选项
--sbin-path=<path>	指定 Nginx 二进制文件的路径。如果没有指定，那么这个路径会依赖于 --prefix 选项
--conf-path=<path>	如果在命令行没有指定配置文件，那么将会通过这里指定的路径，Nginx 将会去那里查找它的配置文件
--error-log-path=<path>	指定错误文件的路径，Nginx 将会往其中写入错误日志文件，除非有其他的配置
--pid-path=<path>	指定的文件将会写入 Nginx master 进程的 pid，通常在 /var/run 下
--lock-path=<path>	共享存储器互斥锁文件的路径
--user=<user>	worker 进程运行的用户
--group=<group>	worker 进程运行的组
--with-file-aio.	为 FreeBSD 4.3 +和 Linux 2.6.22 +系统启用异步 I/O
--with-debug	这个选项用于启用调试日志。在生产环境的系统中不推荐使用该选项

如表 1-2 所示，可以优化编译，这正是下表中的选项的用武之地。

表 1-2 优化配置选项

选项	说明
--with-cc=<path>	如果想设置一个不在默认 PATH 下的 C 编译器
--with-cpp=<path>	设置 C 预处理器的相应路径

续表

选项	说明
<code>--with-cc-opt=&lt;options&gt;</code>	指定必要的 include 文件路径，可能 <code>d(-I&lt;path&gt;)</code> 指出，也可能是优化 ( <code>-O4</code> ) 和指定一个 64 位构建
<code>--with-ld-opt=&lt;options&gt;</code>	包含连接器库的路径 ( <code>-L&lt;path&gt;</code> ) 和运行路径 ( <code>-R&lt;path&gt;</code> )
<code>--with-cpu-opt=&lt;cpu&gt;</code>	通过该选项为特定的 CPU 构建 Nginx

## 1.3 配置 Web 或者 mail 服务器

Nginx 是一个独一无二的高性能 Web 服务器，它也被设计成为一个邮件代理服务器。根据你构建 Nginx 的目标，可将其配置成一个 Web 加速器、Web 服务器、邮件代理，或者是集所有为一体。你可以将任何服务安装在一个二进制文件中，这样做的好处是可以通过配置文件来设置 Nginx 服务器的角色，或者根据需要在高性能的环境中安装一个瘦身的二进制 Nginx 文件。

### 1.3.1 邮件代理的配置选项

表 1-3 是邮件模块独有的配置选项。

表 1-3 mail 配置选项

选项	说明
<code>--with-mail</code>	该选项用于启用 mail 模块，该模块默认没有被激活
<code>--with-mail_ssl_module</code>	为了代理任何一种类型的使用 SSL/TLS 的 mail，激活该模块
<code>--without-mail_pop3_module</code>	在启用 mail 模块后，单独地禁用 POP3 模块
<code>--without-mail_imap_module</code>	在启用 mail 模块后，单独地禁用 IMAP 模块
<code>--without-mail_smtp_module</code>	在启用 mail 模块后，单独地禁用 SMTP 模块
<code>--without-http</code>	该选项将会完全禁用 http 模块，如果你只想支持 mail，那么可以使用它

对于典型的 mail 代理，我推荐将 Nginx 配置为：

```
$ ./configure --with-mail --with-mail_ssl_module --with-openssl=${BUILD_DIR}/openssl-1.0.1c
```

对于邮件服务器来说，现在几乎每一个邮件服务器的安装都需要安装 SSL/TLS，并且没有一个邮件代理启用了预期功能的用户。我推荐静态编译 OpenSSL，以便对操作系统中的 OpenSSL 库没有依赖性。在前面使用的变量 BUILD\_DIR 需要提前设置。

### 1.3.2 指定路径的配置选项

表 1-4 显示了 http 模块有效的配置选项，从激活 Perl 模块到指定临时目录的位置。

表 1-4 Http 配置选项

选项	说明
<code>--without-http-cache</code>	在使用 upstream 模块时，Nginx 能够配置本地缓存内容。这个选项能够禁用缓存
<code>--with-http_perl_module</code>	Nginx 配置能够扩展使用 Perl 代码。这个选项启用这个模块（然而使用这个模块会降低性能）
<code>--with-perl_modules_path=&lt;path&gt;</code>	对于额外嵌入的 Perl 模块，使用该选项指定该 Perl 解析器的路径。也可以通过配置选项来指定 Perl 模块解析器的位置
<code>--with-perl=&lt;path&gt;</code>	如果在默认的路径中没有找到 Perl，那么指定 Perl（5.6 版本以上）的路径
<code>--http-log-path=&lt;path&gt;</code>	Http 访问日志的默认路径
<code>--http-client-body-temp-path=&lt;path&gt;</code>	从客户端收到请求后，该选项设置的目录用于作为请求体临时存放的目录。如果 WebDAV 模块启用，那么推荐设置该路径为同一文件系统上的目录作为最终的目的地
<code>--http-proxy-temp-path=&lt;path&gt;</code>	在使用代理后，通过该选项设置存放临时文件路径
<code>--http-fastcgi-temp-path=&lt;path&gt;</code>	设置 FastCGI 临时文件的目录
<code>--http-uwsgi-temp-path=&lt;path&gt;</code>	设置 uWSGI 临时文件的目录
<code>--http-scgi-temp-path=&lt;path&gt;</code>	设置 SCGI 临时文件的目录

## 1.4 使用各种模块

在 Nginx 发布的版本中，除了 http 和 mail 模块之外，还有其他一些模块。这些模块在默认安装中没有被安装，但是可以在编译安装时适当地配置选项 `--with-<module-name>_module` 来启用相应的选项，如表 1-5 所示。

表 1-5 Http 模块配置选项

选项	说明
<code>--with-http_ssl_module</code>	如果需要对流量进行加密，那么可以使用到这个选项，在 URLs 中开始部分将会是 https（需要 OpenSSL 库）
<code>--with-http_realip_module</code>	如果你的 Nginx 在七层负载均衡器或者是其他设备之后，它们将 Http 头中的客户端 IP 地址传递，那么你将需要启用这个模块。在多个客户处于一个 IP 地址的情况下使用
<code>--with-http_addition_module</code>	这个模块作为一个输出过滤器，使你能够在请求经过一个 location 前或者后时在该 location 本身添加内容
<code>--with-http_xslt_module</code>	该模块用于处理 XML 响应转换，基于一个或者多个 XSLT 格式（需要 libxml2 和 libxslt 库）
<code>--with-http_image_filter_module</code>	该模块被作为图像过滤器使用，在将图像投递到客户之前进行处理（需要 libgd 库）
<code>--with-http_geoip_module</code>	使用该模块，能够设置各种变量以便在配置文件中的区段使用，基于地理位置查找客户端 IP 地址（需要 MaxMfind GeoIP 库和相应的预编译数据库文件）
<code>--with-http_sub_module</code>	该模块实现了替代过滤，在响应中用一个字符串替代另一个字符串
<code>--with-http_dav_module</code>	启用这个模块将激活使用 WebDAV 的配置指令。请注意，这个模块也只在有需要使用的基础上启用，如果配置不正确，它可能带来安全问题

续表

选项	说明
<code>--with-http_flv_module</code>	如果需要提供 Flash 流媒体视频文件, 那么该模块将会提供伪流媒体
<code>--with-http_mp4_module</code>	这个模块支持 H.264/AAC 文件伪流媒体
<code>--with-http_gzip_static_ module</code>	当被调用的资源没有 .gz 结尾格式的文件时, 如果想支持发送预压缩版本的静态文件, 那么使用该模块
<code>--with-http_gunzip_module</code>	对于不支持 gzip 编码的客户, 该模块用于为客户解压缩预压缩内容
<code>--with-http_random_index_ module</code>	如果你想提供从一个目录中随机选择文件的索引文件, 那么这个模块需要被激活
<code>--with-http_secure_link_ module</code>	该模块提供了一种机制, 它会将一个哈希值链接到一个 URL 中, 因此, 只有那些使用正确的密码能够计算链接
<code>--with-http_stub_status_ module</code>	启用这个模块后会收集 Nginx 自身的状态信息。输出的状态信息可以使用 RRDtool 或类似的东西来绘制成图

正如你所看到的, 所有这些模块都是建立在 Http 模块的基础之上的, 它们提供了额外的功能。在编译时启用这些模块根本不会影响到运行性能, 以后在配置使用这些模块时性能会产生影响。

因此, 对于网络加速器/代理, 就配置选项来说, 我想提出以下建议。

```
$ ./configure --with-http_ssl_module --with-http_realip_module --with-  
http_geoip_module --with-http_stub_status_module --with-openssl=${BUILD_  
DIR}/openssl-1.0.1c
```

及 Web 服务器。

```
$ ./configure --with-http_stub_status_module
```

不同之处在于它们面对的客户, 处于 Web 加速角色时, 会考虑到 SSL 请求的终结, 也包括处理代理客户和基于客户来源决策。处于 Web 服务角色时, 则仅需要提供默认文件访

问能力。

我总是推荐启用 `stub_status` 模块，这是因为它提供了收集 Nginx 如何执行、对其度量的一个方法。

## 不再使用的模块

有些 `http` 模块通常情况下是激活的，但是可以通过设置适当的 `--without-<module-name>_module` 选项禁用它们。如果在配置中不使用这些模块，如表 1-6 所示，那么你可以禁用它们。

表 1-6 禁用的配置选项

说明	说明
<code>--without-http_charset_module</code>	该字符集模块负责设置 <b>Content-Type</b> 响应头，以及从一个字符集转换到另一个字符集
<code>--without-http_gzip_module</code>	<b>gzip</b> 模块作为一个输出过滤器，在将内容投递到客户时对内容进行压缩
<code>--without-http_ssi_module</code>	该模块是一个过滤器，用于处理 <b>SSI</b> 包含。如果启用了 <b>Perl</b> 模块，那么额外的 <b>SSI</b> 指令 ( <b>perl</b> ) 可用
<code>--without-http_userid_module</code>	<b>userid</b> 模块能够使得 Nginx 设置 <b>cookies</b> ，用于客户标识。变量 <code>\$uid_set</code> 和 <code>\$uid_got</code> 可以记录用户跟踪
<code>--without-http_access_module</code>	<b>access</b> 模块基于 <b>IP</b> 控制访问 <b>location</b>
<code>--without-http_auth_basic_ module</code>	该模块通过 <b>Http</b> 基本身份验证限制访问
<code>--without-http_autoindex_ module</code>	如果一个目录中没有 <b>index</b> 文件，那么 <b>autoindex</b> 模块能够收集这个目录列出文件
<code>--without-http_geo_module</code>	该模块能够让你基于客户端 <b>IP</b> 地址设置配置变量，然后根据这些变量的值采取行动
<code>--without-http_map_module</code>	<b>map</b> 模块能够让你映射一个变量到另一个变量
<code>--without-http_split_clients_ module</code>	该模块创建用于 <b>A/B</b> 测试的变量
<code>--without-http_referer_module</code>	该模块能够让 Nginx 阻止基于 <b>Referer</b> <b>Http</b> 头的请求

续表	
说明	说明
--without-http_rewrite_module	通过 rewrite 模块能够让你基于变量条件改变 URI
--without-http_proxy_module	使用 proxy 模块允许 Nginx 将请求传递到其他服务器或者服务器组
--without-http_fastcgi_module	FastCGI 模块能够让 Nginx 将请求传递到 FastCGI 服务器
--without-http_uwsgi_module	这个模块能够使得 Nginx 将请求传递到 uWSGI 服务器
--without-http_scgi_module	SCGI 模块能够让 Nginx 将请求传递到 SCGI 服务器
--without-http_memcached_module	该模块能够使得 Nginx 与一个 memcached 服务器进行交互，将响应放置到变量查询中
--without-http_limit_conn_module	该模块能够使得 Nginx 基于某些键，通常是 IP 地址，设置连接限制
--without-http_limit_req_module	通过该模块，Nginx 能够限制每个用户的请求率
--without-http_empty_gif_module	在内存中产生一个 1 像素×1 像素的透明 GIF 图像
--without-http_browser_module	browser 模块允许基于 User-Agent Http 请求头配置，变量的设置基于在该头中发现的版本
--without-http_upstream_ip_hash_module	该模块定义了一组可以与不同的代理模块结合使用的服务器

## 1.5 查找并安装第三方模块

由于有多个开源项目，所以在 Nginx 周围就会有一个活跃的开发社区。由于 Nginx 的模块化特性，这个社区能够开发和发布模块，从而为 Nginx 提供额外的功能。它们涵盖了广泛的应用，所以着手开发自己的模块之前应该看看有什么可用模块。

安装第三方模块的过程相当简单，步骤如下。

1. 定位你想要使用的模块（在 <https://github.com> 或者是 <http://wiki.nginx.org/3rdPartyModules> 查找）。

2. 下载该模块。
3. 解压缩源代码安装包。
4. 如果有 README 文件，那么阅读 README 文件，查看在安装中是否有依赖安装。
5. 通过 `/configure-add-module=<path>` 选项配置使用该模块。

这个过程会给你的 Nginx 二进制文件与模块附加这个功能。

需要注意的是，很多第三方模块是实验性质的，因此在将这些模块用于生成系统之前要测试使用这些模块。另外请记住，Nginx 的开发版本中可能会有 API 的变化，会导致第三方模块出现问题。

安装说明详见：<http://wiki.nginx.org/HttpLuaModule#Installation>，我们将在下一部分将该模块作为一个安装第三方库的示例使用。

特别应该提到的是 ngx\_lua 这个第三方模块，ngx\_lua 模块提供了启用 Lua 的功能，而不是像 Perl 一样在配置时嵌入式脚本语言。该模块对于 Perl 模块来说最大的优点就是它的无阻塞性，并与其他第三方模块紧密集成。对于它的安装说明完整的描述详见：<http://wiki.nginx.org/HttpLuaModule#Installation>。

我们将以这个模块为例在下一节中介绍如何安装第三方模块。

## 1.6 组合在一起

现在你已经大概了解了各种配置选项，接下来你可以根据自己的需要设计一个二进制文件。下面的例子中，指定了 `prefix`、`user`、`group`，某些路径禁用了某些模块，启用了一些其他模块，包括一些第三方模块。

```
$ export BUILD_DIR=`pwd`
$ export Nginx_INSTALLDIR=/opt/nginx
$ export VAR_DIR=/home/www/tmp
$ export LUAJIT_LIB=/opt/luajit/lib
$ export LUAJIT_INC=/opt/luajit/include/luajit-2.0

$ ./configure \
    --prefix=${Nginx_INSTALLDIR} \
    --user=www \
    --group=www \
    --http-client-body-temp-path=${VAR_DIR}/client_body_temp \
```

```

--http-proxy-temp-path=${VAR_DIR}/proxy_temp \
--http-fastcgi-temp-path=${VAR_DIR}/fastcgi_temp \
--without-http_uwsgi_module \
--without-http_scgi_module \
--without-http_browser_module \
--with-openssl=${BUILD_DIR}/../openssl-1.0.1c \
--with-pcre=${BUILD_DIR}/../pcre-8.32 \
--with-http_ssl_module \
--with-http_realip_module \
--with-http_sub_module \
--with-http_flv_module \
--with-http_gzip_static_module \
--with-http_gunzip_module \
--with-http_secure_link_module \
--with-http_stub_status_module \
--add-module=${BUILD_DIR}/ngx_devel_kit-0.2.17 \
--add-module=${BUILD_DIR}/ngx_lua-0.7.9

```

接下来跟随的大量输出显示了在您的系统上能找到什么样的配置，概要打印出来，如下所示。

```

Configuration summary
+ using PCRE library: /home/builder/build/pcre-8.32
+ using OpenSSL library: /home/builder/build/openssl-1.0.1c
+ md5: using OpenSSL library
+ sha1: using OpenSSL library
+ using system zlib library

nginx path prefix: "/opt/nginx"
nginx binary file: "/opt/nginx/sbin/nginx"
nginx configuration prefix: "/opt/nginx/conf"
nginx configuration file: "/opt/nginx/conf/nginx.conf"
nginx pid file: "/opt/nginx/logs/nginx.pid"
nginx error log file: "/opt/nginx/logs/error.log"
nginx http access log file: "/opt/nginx/logs/access.log"
nginx http client request body temporary files: "/home/www/tmp/
client_body_temp"
nginx http proxy temporary files: "/home/www/tmp/proxy_temp" nginx http
fastcgi temporary files: "/home/www/tmp/fastcgi_temp"

```

如上所示，`configure` 找到了所有我们要查找的条目，并且按照我们的喜好设置了路径。

现在，你可以构建你的 Nginx 并且安装它了，正如本章一开始提到的。

## 1.7 总结

本章介绍了各种 Nginx 的有效模块，通过编译你自己的二进制文件，你可以定制 Nginx 能够为你提供哪些功能。构建和安装软件对于你来说应该不会陌生，所以在创造一个构建环境或确保所有依赖关系都存在上不会花很多的时间。一个 Nginx 的安装应该是按照你的需要，能随时启用或禁用模块，正如你看到的，启用或者是禁用一个模块应该感到很容易。

接下来我们将介绍基本的 Nginx 配置概述，以便感受一下在通常情况下 Nginx 是如何配置的。

# 第 2 章

## 配置指南

Nginx 的配置文件的格式非常合乎逻辑。学习这种格式以及如何使用每个部分是基础，这将有助于你手工创建一个配置文件。通过这一章的下列讨论话题将帮助你达到这个目的。

- ◆ 基本配置格式。
- ◆ Nginx 全局配置参数。
- ◆ 使用 include 文件。
- ◆ Http 的 server 部分。
- ◆ 虚拟服务器部分。
- ◆ location——在哪儿，什么时候，怎么样。
- ◆ mail 的 server 部分。
- ◆ 完整的示例配置文件。

### 2.1 基本配置格式

基本的 Nginx 配置文件由若干个部分组成。每一个部分都是通过下列方法定义的。

```
<section> {  
  
    <directive> <parameters>;  
  
}
```

需要注意的是每一个指令行都由分号结束 (;)，这标记着一行的结束。大括号 ({} ) 实际上表示一个新上下文(context)，但是大多数情况下我们将它们作为“节、部分(section)”来读。

## 2.2 Nginx 的全局配置参数

全局配置部分被用于配置对整个 server 都有效的参数和前一个章节中的例外格式。全局部分可能包含配置指令，例如，user 和 worker\_processes，也包括“节、部分(section)”。例如，events，这里没有大括号 ({} ) 包围全局部分。

在全局部分中，最重要的配置指令都在表 2-1 中，这些指令将会是你处理的最重要部分。

表 2-1 全局配置指令

指令	说明
user	使用这个参数来配置 worker 进程的用户和组。如果忽略 group，那么 group 的名字等于该参数指定的用户的用户组
worker_processes	指定 worker 进程启动的数量。这些进程用于处理客户的连接。选择一个正确的数量取决于服务器环境、磁盘子系统和网络基础设施。一个好的经验法则是设置该参数的值与 CPU 绑定的负载处理器核心的数量相同，并用 1.5~2 之间的数乘以这个数作为 I/O 密集型负载
error_log	error_log 是所有错误写入的文件。如果在其他区段中没有设置其他的 error_log，那么这个日志文件将会记录所有的错误。该指令的第二个参数指定了被记录错误的级别(debug, info, notice, warn, error, crit, alert, emerg)。注意，debug 级别的错误只有在编译时配置了 --with-debug 选项才可以使用
pid	设置记录主进程 ID 的文件，这个配置将会覆盖编译时的默认配置
use	该指令用于指示使用什么样的连接方法，这个配置将会覆盖编译时的默认配置，如果配置该指令，那么需要一个 events 区段。通常不需要覆盖，除非是当编译时的默认值随着时间的推移产生错误时才需要被覆盖设置
worker_connections	该指令配置一个工作进程能够接受并发连接的最大数。这个连接包括，客户连接和向上游服务器的连接，但并不限于此。这对于反向代理服务器尤为重要，为了达到这个并发性连接数量，需要在操作系统层面进行一些额外调整

下面是一个使用这些指令的简短的例子。

```
# we want nginx to run as user 'www'
user www;

# the load is CPU-bound and we have 12 cores
worker_processes 12;

# explicitly specifying the path to the mandatory error log
error_log /var/log/nginx/error.log;

# also explicitly specifying the path to the pid file
pid /var/run/nginx.pid;

# sets up a new configuration context for the 'events' module
events {

    # we're on a Solaris-based system and have determined that nginx
    # will stop responding to new requests over time with the default
    # connection-processing mechanism, so we switch to the second-best
    use /dev/poll;
    # the product of this number and the number of worker_processes
    # indicate s how many simultaneous connections per IP:port pair are
    # accepted
    worker_connections 2048;

}
```

这一部分应该放置在 `nginx.conf` 文件的顶部。

## 2.3 使用 include 文件

在 Nginx 的配置文件中，`include` 文件可以在任何地方，以便增强配置文件的可读性，并且能够使得部分配置文件重新使用。使用 `include` 文件，要确保被包含的文件自身有正确的 Nginx 语法，即配置指令和块（blocks），然后指定这些文件的路径。

```
include /opt/local/etc/nginx/mime.types;
```

在路径中出现通配符，表示可以配置多个文件。

```
include /opt/local/etc/nginx/vhost/*.conf;
```

如果没有给定全路径，那么 Nginx 将会依据它的主配置文件路径进行搜索。

Nginx 测试配置文件很容易，通过下面的命令来完成。

```
nginx -t -c <path-to-nginx.conf>
```

该命令将测试 Nginx 的配置文件，包括 include 文件，但是它只检查语法错误。

## 2.4 Http 的 server 部分

在 Http 中，server 部分或者是 Http 配置 context 是可用的，除非在编译安装 Nginx 时没有包含 Http 模块（也就是使用了--without-http）。这部分控制了 Http 模块的方方面面，是使用最多的一个部分。

本部分的指令用于处理 Http 连接，因此该模块提供了相当数量的指令。为了更容易理解这些指令我们将它们划分为不同的类型来讲述。

### 2.4.1 客户端指令

如表 2-2 所示，这一组指令用于处理客户端连接本身的各个方面，以及不同类型的客户端。

表 2-2 Http 客户端指令

指令	说明
chunked_transfer_encoding	在发送给客户端的响应中允许禁用 Http/1.1 标准的块传输编码
client_body_buffer_size	为了阻止临时文件写到磁盘，可以通过该指令为客户端请求体设置缓存大小，默认的缓存大小为两个内存页面
client_body_in_file_only	用于调试或者是进一步处理客户端请求体。该指令能够将客户端请求体强制写入到磁盘文件
client_body_in_single_buffer	为了减少拷贝的操作，使用该指令强制 Nginx 将整个客户端请求体保存在单个缓存中
client_body_temp_path	定义一个命令路径用于保存客户端请求体

续表

指令	说明
client_body_timeout	指定客户体成功读取的两个操作之间的时间间隔
client_header_buffer_size	为客户端请求头指定一个缓存大小，当请求头大于 1kB 时会用到这个设置
client_header_timeout	该超时是读取整个客户端头的时间长度
client_max_body_size	定义允许最大的客户端请求头，如果大于这个设置，那么客户端将会是 413 (Request Entity Too Large) 错误
keepalive_disable	对某些类型的客户端禁用 keep-alive 请求功能
keepalive_requests	定义在一个 keep-alive 关闭之前可以接受多少个请求
keepalive_timeout	指定 keep-alive 连接持续多久。第二个参数也可以设置，用于在响应头中设置 “Keep-Alive” 头
large_client_header_buffers	定义最大数量和最大客户端请求头的大小
msie_padding	为了填充响应的大小至 512 字节，对于 MSIE 客户端，大于 400 的状态代码会被添加注释以便满足 512 字节，通过启用该命令可以阻止这种行为
msie_refresh	对于 MSIE 客户端，可启用发送一个 refresh 头，而不是重定向

2.4.2 文件 I/O 指令

这些指令用于控制 Nginx 如何投递静态文件，以及如何管理文件描述符参见表 2-3。

表 2-3 Http 文件 I/O 指令

指令	说明
aio	启用异步文件 I/O。该指令对于现代版本的 FreeBSD 和发布的 Linux 都有效。在 FreeBSD 系统下，aio 可能被用于 sendfile 预加载数据。在 Linux 下需要 directio 指令，自动禁用 sendfile

续表

指令	说明
directio	用于启用操作系统特定的标志或者功能提供大于给定参数的文件。 在 Linux 系统下使用 aio 时需要使用该指令
directio_alignment	设置 directio 的算法。默认值为 512，通常足够了，但是在 Linux 的 XFS 下推荐增加为 4K
open_file_cache	配置一个缓存用于存放打开的文件描述符、目录查询和文件查询错误
open_file_cache_errors	按照 open_file_cache，启用文件查询错误缓存
open_file_cache_min_uses	open_file_cache 缓存的文件描述符保留在缓存中，使用该指令配置最少使用文件描述符的次数
open_file_cache_valid	指定对 open_file_cache 缓存有效性检查的时间间隔
postpone_output	指定 Nginx 发送给客户端最小的数值，如果可能的话，没有数据会发送，直到达到此值
read_ahead	如果可能的话，内核将预读文件到设定的参数大小。目前支持 FreeBSD 和 Linux（Linux 会忽略大小）
sendfile	使用 sendfile（2）直接复制数据从一个到另一个文件描述符
sendfile_max_chunk	设置在一个 sendfile（2）拷贝中最大数据的大小，这是为了阻止 worker “贪婪”

### 2.4.3 Hash 指令

如表 2-4 所示，这组 hash 指令控制 Nginx 分配给某些变量多大的静态内存。在启动和重新配置时，Nginx 会计算需要的最小值。在 Nginx 发出警告时，你几乎只需要调整一个 \*\_hash\_max\_size 指令的参数值就可以达到效果。\*\_hash\_bucket\_size 变量被设置了默认值，以便满足多处理器缓存行降低检索所需要的检索查找，因此基本不需要改变，额外更详细的内容参考 <http://nginx.org/en/docs/hash.html>。

表 2-4 Http hash 指令

指令	说明
server_names_hash_bucket_size	指定用于保存 server_name 哈希表大小的“桶”
server_names_hash_max_size	指定的 server_name 哈希表的最大大小
types_hash_bucket_size	指定用于存放哈希表的“桶”的大小
types_hash_max_size	指定哈希类型表的最大大小
variables_hash_bucket_size	它指定用于存放保留变量“桶”的大小
variables_hash_max_size	指定存放保留变量最大哈希值的大小

2.4.4 Socket 指令

如表 2-5 所示，这些指令描述了 Nginx 如何设置创建 TCP 套接字的变量选项。

表 2-5 Http 套接字指令

指令	说明
lingering_close	指定如何保持客户端的连接，以便用于更多数据的传输
lingering_time	在使用 lingering_close 指令的连接中，使用该指令指定客户端连接为了处理更多的数据需要保持打开连接的时间
lingering_timeout	结合 lingering_close，该指令显示 Nginx 在关闭客户端连接之前，为获得更多数据会等待多久
reset_timedout_connection	使用这个指令之后，超时的连接将会被立即关闭，释放相关的内存。默认的状态是处于 FIN_WAIT1，这种状态将会一直保持连接
send_lowat	如果非零，Nginx 将会在客户端套接字尝试减少发送操作
send_timeout	在两次成功的客户端接收响应的写操作之间设置一个超时时间
tcp_nodelay	启用或者禁用 TCP_NODELAY 选项，用于 keep-alive 连接
tcp_nopush	仅依赖于 sendfile 的使用。它能够使得 Nginx 在一个数据包中尝试发送响应头，以及在数据包中发送一个完整的文件

## 2.4.5 示例配置文件

下面是一个 Http 配置部分的例子。

```
http {  
  
    include          /opt/local/etc/nginx/mime.types;  
  
    default_type     application/octet-stream;  
  
    sendfile on;  
  
    tcp_nopush on;  
  
    tcp_nodelay on;  
  
    keepalive_timeout 65;  
  
    server_names_hash_max_size 1024;  
  
}
```

在 `nginx.conf` 文件中上面的这部分内容跟随在全局配置指令之后。

## 2.5 虚拟 server 部分

任何由关键字 `server` 开始的部分都被称作“虚拟服务器”部分。它描述的是一组根据 `server_name` 指令逻辑分割的资源，这些虚拟服务器响应 Http 请求，因此它们都包含在 `http` 部分中。

一个虚拟服务器由 `listen` 和 `server_name` 指令组合定义，`listen` 指令定义了一个 IP 地址/端口组合或者是 UNIX 域套接字路径。

```
listen address[:port];  
listen port;  
listen unix:path;
```

如表 2-6 所示，`listen` 指令唯一地标识了在 Nginx 下的套接字绑定，此外还有一些其他的可选参数。

表 2-6 listen 指令的参数		
参数	说明	注解
default_server	定义这样一个组合： address:port 默认的请求被绑定在此	
Setfib	为套接字监听设置相应的 FIB	仅支持 FreeBSD，不支持 UNIX 域套接字
backlog	在 listen() 的调用中设置 backlog 参数调用	在 FreeBSD 系统中默认值为 -1，而在其他的系统中为 511
rcvbuf	在套接字监听中设置 SO_RCVBUF 参数	
sndbuf	在套接字监听中设置 SO_SNDBUF 参数	
accept_filter	设置接受的过滤器： dataready 或者 httpready dataready	仅支持 FreeBSD
deferred	设置 accept() 调用的 TCP_DEFER_ ACCEPT	仅支持 Linux
bind	为 address:port 套接字对打开一个单独的 bind() 调用	如果任何其他特定套接字参数被使用，那么一个单独的 bind() 将会被隐式地调用
ipv6only	设置 IPV6_V6ONLY 参数的值	只能在一个全新的开始设置。不支持 UNIX 域套接字
ssl	表明该端口仅接受 Https 的连接	允许更紧凑的配置
so_keepalive	为 TCP 监听套接字配置 keepalive	

server\_name 指令是相当简单的，但可以用来解决一些配置问题。它的默认值为""，这意味着 server 部分没有 server\_name 指令，对于没有设置 Host 头字段的请求将会匹配该 server 处理。这种情况可用于，例如，丢弃这种缺乏 Host 头的请求。

```
server {  
  
    listen 80;  
  
    return 444;  
  
}
```

在这个例子中使用的 Http 非标准代码 444 将会使得 Nginx 立即关闭一个连接。

除了普通的字符串之外，Nginx 也接受通配符作为 `server_name` 的参数。

◆ 通配符可以替代部分子域名：\*.example.com。

◆ 通配符可以替代顶级域部分：www.example.\*。

◆ 一种特殊形式将匹配子域或域本身：

.example.com（匹配\*.example.com也包括example.com）。

通过在域名前面加上波浪号（~），正则表达式也可以被作为参数应用于 `server_name`。

```
server_name ~ ^www\.example\.com$;  
server_name ~ ^www(\d+)\.example\.(com)$;
```

后一种形式是利用捕获，可以在以后引用中进一步配置（用\$1，\$2等）指令中使用。

对于一个特定的请求，确定哪些虚拟服务器提供该请求的服务时，应该遵循下面的逻辑。

1. 匹配 IP 地址和 `listen` 指令指定的端口。
2. 将 Host 头字段作为一个字符串匹配 `server_name` 指令。
3. 将 Host 头字段与 `server_name` 指令值字符串的开始部分做匹配。
4. 将 Host 头字段与 `server_name` 指令值字符串的尾部分做匹配。
5. 将 Host 头字段与 `server_name` 指令值进行正则表达式匹配。
6. 如果所有 Host 头匹配失败，那么将会转向 `listen` 指令标记的 `default_server`。
7. 如果所有的 Host 头匹配失败，并且没有 `default_server`，那么将会转向第一个 `server` 的 `listen` 指令，以满足第 1 步。

这个逻辑体现在下面的图 2-1 中。

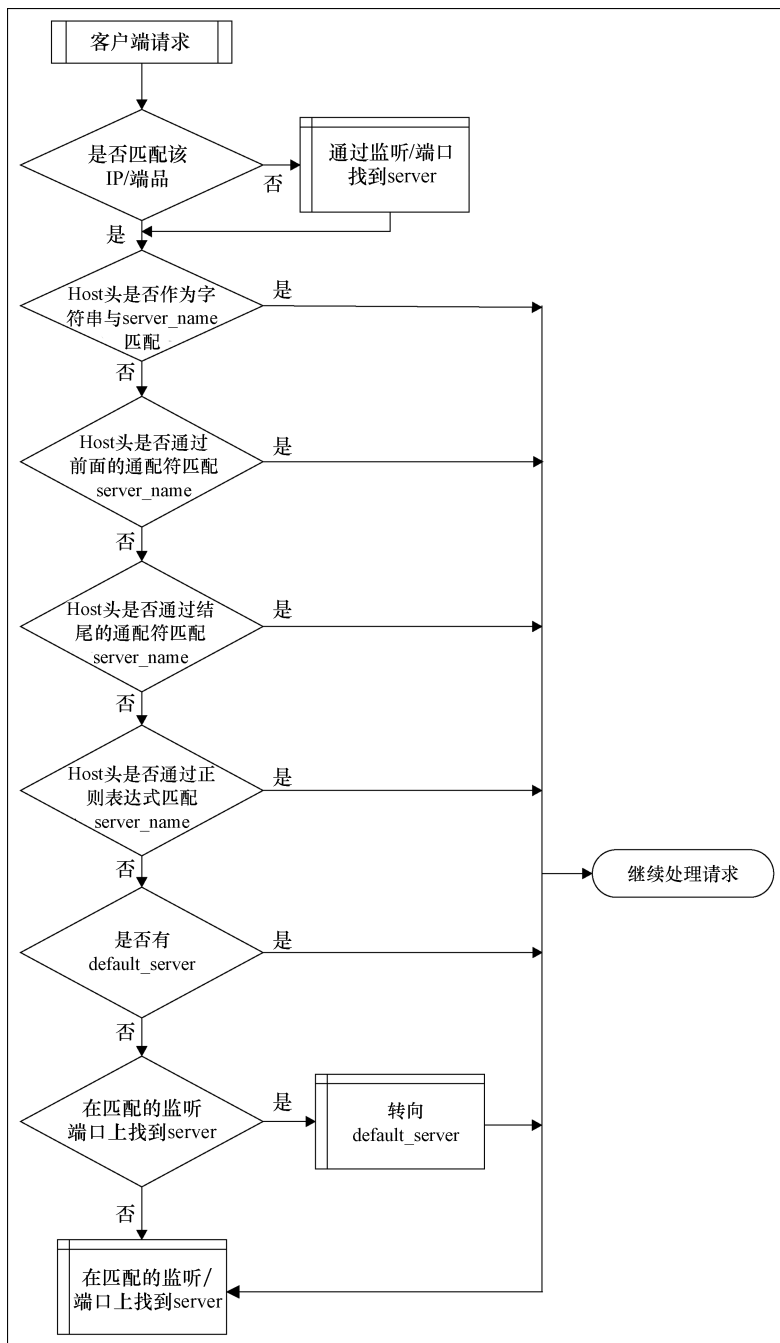


图 2-1 Host 头匹配流程图

`default_server` 被用于处理其他方式没有处理的请求。因此推荐总是明确地设置 `default_server`，以便这些没有被处理的请求通过这种定义的方式处理。

除了这个用法外，`default_server` 也可以使用同样的 `listen` 指令配置若干个虚拟服务器。这里设置的任何指令都将会在匹配的 `server` 区段有效。

## 2.6 Locations——where, when, how

`location` 指令可以用在虚拟服务器 `server` 部分，并且意味着提供来自客户端的 `URI` 或者内部重定向访问。除少数情况外，`location` 也可以被嵌套使用，它们被作为特定的配置尽可能地处理请求。

`location` 定义如下。

```
location [modifier] uri {...}
```

或者是命名 `location`。

```
location @name {...}
```

命名 `location` 仅对内部访问重定向，在进入一个 `location` 之前它会保留被请求的 `URI` 部分。命名 `location` 只能够在 `server` 级别定义。

表 2-7 中的修饰符会影响 `location` 的处理。

表 2-7 `location` 修饰符

修饰符	处理方式
<code>=</code>	使用精确匹配并且终止搜索
<code>~</code>	区分大小写的正则表达式匹配
<code>~*</code>	不区分大小写的正则表达式匹配
<code>^~</code>	如果该 <code>location</code> 是最佳的匹配，那么对于匹配这个 <code>location</code> 的字符串不再进行正则表达式检测。注意这不是一个正则表达式匹配——它的目的是优先于正则表达式的匹配

当一个请求进入时，`URI` 将会被检测匹配一个最佳的 `location`。

- ◆ 没有正则表达式的 `location` 被作为最佳的匹配，独立于含有正则表达式的 `location` 顺序。
- ◆ 在配置文件中按照查找顺序进行正则表达式匹配，在查找到第一个正则表达式匹

配之后结束查找，那么就由这个最佳的 location 提供请求处理。

这里比较匹配描述的是解码 URI，例如，在 URI 中的 "%20"，将会匹配 location 中的 " "(空格)。

命名 location 仅可以在内部重定向的请求中使用。

表 2-8 中的指令仅在 location 中使用。

表 2-8 仅用于 location 的指令

指令	说明
alias	定义 location 的其他名字，在文件系统中能够找到。如果 location 指定了一个正则表达式，alias 将会引用正则表达式中定义的捕获。alias 替代 location 中匹配的部分，没有匹配的部分将会在文件系统中搜索。当配置改变一点，配置中使用 alias 则会有脆弱的表现，因此推荐使用 root，除非是为了找到文件而需要修改
internal	指定一个仅用于内部请求的 location（其他指定定义的重定向、rewrite 请求、error 请求等）
limit_except	限定一个 location 可以执行的 Http 操作（GET 也包括 HEAD）

另外，http 部分的其他指令也可以在 location 中指定，参考附录 A 指令参考有完整列表。

指令 try\_files 在这里也值得一提，它也可以用在 server 部分，但是最常见的还是在 location 部分中。try\_files 指令将会按照给定它的参数列出的顺序进行尝试，一个被匹配的将会被使用。

它经常被用于从一个变量去匹配一个可能的文件，然后将处理传递到一个命名 location，看下面的例子。

```
location / {  
  
    try_files $uri $uri/ @mongrel;  
  
}  
  
location @mongrel {
```

```
    proxy_pass http://appserver;

}
```

在这里有一个隐含的目录索引，如果给定的 URI 作为一个文件没有被找到，那么处理将会通过代理被传递到 appserver。我们将会在本书的其他部分讨论如何最好地使用 location、try\_files 和 proxy\_pass 来解决特定的问题。

除以下前缀外，locations 可以被嵌套。

- ◆ 具有 "=" 前缀。
- ◆ 命名 location。

最佳实践表明正则表达式 location 被嵌套在基于字符串的 location，看下面的例子。

```
# ffirst, we enter through the root
location / {

    # then we ffind a most-specific substring
    # note that this is not a regular expression
    location ^~ /css {

        # here is the regular expression that then gets matched
        location~* /css/.*\.css$ {

        }

    }

}
```

## 2.7 mail 的 server 部分

mail 服务部分，或者是 mail 的配置内容部分，仅在构建 Nginx 时使用了 mail 模块（--with-mail）才有效。这个部分控制了 mail 模块的所有方面。

作为 mail 模块允许配置影响代理邮件连接的所有方面，也可以为每个 server 指定。这个 server 也可以接受 listen 和 server\_name 指令，这些指令我们在 http server 部分已经看过了。

Nginx 能够代理 IMAP、POP3 和 SMTP 协议，表 2-9 中列出了该模块有效的指令。

表 2-9 Mail 模块指令

指令	说明
auth_http	指定提供的认证方式，用于 POP3/IMAP 用户认证使用。这个功能将会在第 3 章详细讨论
imap_capabilities	指示后端服务器支持 IMAP4 功能
pop3_capabilities	指示后端服务器支持 POP3 功能
protocol	指示虚拟 server 支持的协议
proxy	该指令将会简单地启用或者禁用 mail 代理
proxy_buffer	该指令设置用于代理连接缓冲，在高于默认值时需要通过该指令设置
proxy_pass_error_message	在后端认证进程向客户端发出一个有用的错误消息的情况下，这个设置很有用
proxy_timeout	如果超时需要超过默认的 24 个小时，那么该指令需要配置
xclient	SMTP 协议允许基于 IP/HELO/LOGIN 参数检查，它们通过 XCLIENT 命令传递。该指令使 Nginx 传递这种消息

如果 Nginx 在编译时支持了 SSL (`--with-mail_ssl_module`)，那么表 2-10 所示指令在前面的 mail 模块中可以使用。

表 2-10 Mail 模块的 SSL 指令

指令	说明
ssl	表明该部分支持 SSL 处理
ssl_certificate	为该虚拟 server 指定 PEM 编码的 SSL 证书路径
ssl_certificate_key	为该虚拟 server 指定 PEM 编码的 SSL 密码密钥
ssl_ciphers	指定在该虚拟 server 中支持密码 (OpenSSL 格式)
ssl_prefer_server_ciphers	表明 SSLv3 和 TLSv1 服务器密码是客户端的首选
ssl_protocols	指示使用的 SSL 协议
ssl_session_cache	指定 SSL 缓存，以及其是否应该在所有的 worker 进程中共享
ssl_session_timeout	客户端能够使用多久相同的 SSL 参数，提供的参数被存储在该缓存中

## 2.8 完整的样本配置文件

以下示例是一个样本配置文件，它包括了在本章讨论的各个不同方面。请注意，不要复制粘贴该样本配置文件，因为它很可能不是你需要的配置，而只是显示了一个完整配置文件的架构而已。

```
user www;

worker_processes 12;

error_log /var/log/nginx/error.log;

pid /var/run/nginx.pid;

events {

    use /dev/poll;

    worker_connections 2048;

}

http {

    include      /opt/local/etc/nginx/mime.types;

    default_type application/octet-stream;

    sendfile on;

    tcp_nopush on;

    tcp_nodelay on;

    keepalive_timeout 65;

    server_names_hash_max_size 1024;
```

```
server {  
  
    listen 80;  
  
    return 444;  
  
}  
  
server {  
  
    listen 80;  
  
    server_name www.example.com;  
  
    location / {  
  
        try_files $uri $uri/ @mongrel;  
  
    }  
  
    location @mongrel {  
  
        proxy_pass http://127.0.0.1:8080;  
  
    }  
  
}  
  
}
```

## 2.9 总结

在本章，我们看到了如何构建 Nginx 的配置文件。模块化的本质值得思考，从某种意义上讲，Nginx 本身也是模块化的。全局的配置区段负责各个方面，对于 Nginx 来说是一个整体。Nginx 负责处理的每一种协议单独成为一个部分。我们还可以通过在这些协议配置内（http 或者 mail）指定 server 来定义每一个请求如何被处理，以便请求被路由到特定的 IP 地址和端口上。在 http 区段中，使用 locations 来匹配 URI 请求，这些 locations

可以嵌套使用，或者按其他顺序使用，以确保请求被路由到正确的文件系统区域或者应用程序服务器。

本章没有涵盖编译到二进制 Nginx 命令中各种模块提供的配置选项，这些额外的指令将会遍及本书的始终，因此特定的模块被用于解决一个问题。对于 Nginx 配置中的变量也没有解释，这些变量也将会在本书后边的内容中讨论。本章的焦点是基于 Nginx 的基本配置。

# 第 7 章

## Nginx 的开发

纵观全书，到目前为止，我们已经看到如何为多种不同环境配置 Nginx。我们还没有做的是看一下能够为应用程序开发人员提供的配置。有些方法能够使得 Nginx 直接集成到应用程序。下面的部分我们将来探讨这些可能性。

- ◆ 集成缓存。
- ◆ 动态修改内容。
- ◆ 使用服务器端包含 SSI。
- ◆ Nginx 中的决策。
- ◆ 创建安全链接。
- ◆ 生成图像。
- ◆ 跟踪网站访问者。
- ◆ 防止意外代码执行。

### 7.1 缓存集成

在提供静态内容方面 Nginx 是一流的服务器。它的设计目标是只使用最少的系统资源来支持 100000 多个并发访问。将一个动态 Web 程序集成到这样一个架构的服务器中可能意味着会对 Nginx 服务器的性能造成影响。我们尽可能多地并发连接，但是这并不意味着我们不能给我们的用户一个发动的网络体验。

缓存在第 5 章“反向代理高级问题”中介绍过。在这一部分中，我们将深入地研究 Nginx 的缓存机制，将其整合到 Web 应用程序中。你的 Web 应用程序可能已经缓存到一定的程度，可能已经在你数据库中预提供了一个页面，使这一个“昂贵”的操作不会在每一个访问中都执行。或者更好一点的是你的应用程序可提前将页面写入文件系统中，以便它们能够达到只是简单地类似于 Nginx 提供静态文件一样的性能。无论您的应用程序的缓存机制已经有或者没有，Nginx 都提供了一种方法，将它集成到服务器。

### 7.1.1 应用程序没有缓存

当你的应用程序确实没有缓存可言时，Nginx 仍然可以帮助加快用户的响应时间。代理和 fastcgi 模块能够使用缓存功能，为此你只需在 Nginx 的配置文件中添加 proxy\_cache\_\* 或者 fastcgi\_cache\_\* 指令就可以了。proxy\_cache\_\* 指令在第 5 章“反向代理高级问题”中有描述，fastcgi\_cache\_\* 命令在第 6 章“Nginx Http 服务器”中有描述。

在这里，我们将介绍如何扩展你的应用程序，以指导 Nginx 如何缓存单个页面，可以通过发送给 Nginx 的头来实现，你可以使用标准的 Expires 和 Cache-Control 头，或者指定 X-Accel-Expires 头，Nginx 仅为缓存解释而不传递给客户端。这个头允许应用程序控制 Nginx 缓存文件时间的长度。对于通常生存时间较长的对象来说，这样很容易使它们过期。

假设你有一个新的应用程序，加载页面时遭受着缓慢的延时，这种情况的导致可能有多种原因，但是经过分析，你确定从存储在数据库中的内容到每个网页都要实时渲染生成。当一个用户访问站点时，在一个完整呈现页面投递到用户之前，这将导致新的数据库连接被打开，多个 SQL 查询生成，页面被解析出来。由于在后台系统的应用程序中有多个连接，这种架构不能利用一种更合理的渲染策略轻易地重组。

由于存在这些限制，你可以决定使用以下缓存策略。

- ◆ 首页缓存 1 分钟，因为它所包含的链接文章及文件列表经常更新。
- ◆ 每篇文章都将被缓存 1 天，因为一旦写完它们将不会改变，但是我们又不希望缓存被填满，因此需要移除一些旧的缓存内容以便满足空间的需要。
- ◆ 尽快能地缓存所有图像，因为从磁盘检索图像文件是一个比较“昂贵”的操作。

我们将为 Nginx 添加以下配置来支持这些策略。

```
http {
```

```
# here we configure two separate shared memory zones for the keys/metadata
# and filesystem paths for the cached objects themselves
proxy_cache_path /var/spool/nginx/articles keys_zone=ARTICLES:16m
levels=1:2 inactive=1d;

proxy_cache_path /var/spool/nginx/images keys_zone=IMAGES:128m
levels=1:2 inactive=30d;

# but both paths still lie on the same filesystem as proxy_temp_
path
proxy_temp_path /var/spool/nginx;

server {

    location / {

        # this is where the list of articles is found
        proxy_cache_valid 1m;

    }

    location /articles {

        # each article has a URI beginning with "/articles"
        proxy_cache_valid 1d;

    }

    location /img {

        # every image is referenced with a URI under "/img"
        proxy_cache_valid 10y;

    }

}
```

这是按照我们的需要设置的，对于遗留没有设置缓存支持的应用程序我们已经启用了缓存。

## 7.1.2 使用数据库缓存

如果你的应用程序当前在数据库中缓存了预呈送的页面，那么不需要花费更多额外的努力就能够将这些页面放在 Memcached 实例中。

Nginx 能够直接从 Memcached 缓存中回复请求，其逻辑如图 7-1 所示。

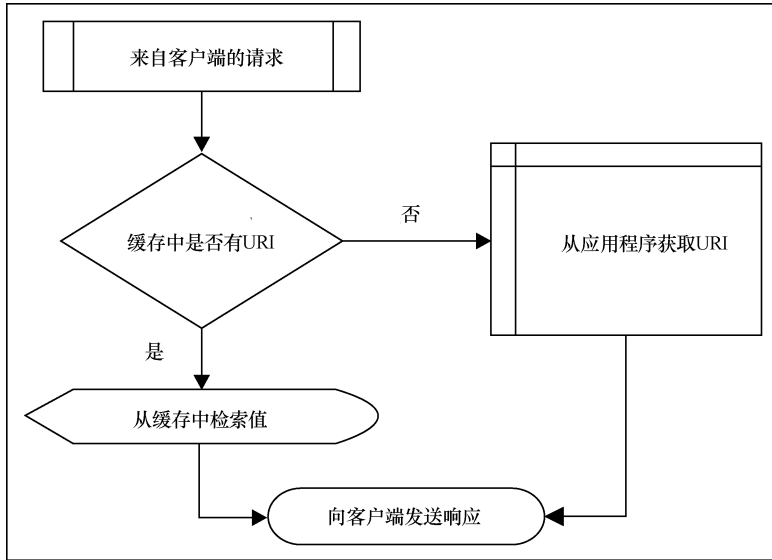


图 7-1 缓存示意图

接口很简单，使得它尽可能灵活。Nginx 在存储中查找一个 key，如果找到，那么将会把该值返回给客户端。构造一个合适的 key 来配置，我们将在下面讨论。在那个 key 下存储的值设计在 Nginx 范围之外，这个工作属于应用程序的。

确定使用 key 是一件非常简单的事情，对于没有特别要求的资源，最好的 key 是使用 URI 自身，对 \$memcached\_key 变量就是这么设置的。

```
location / {  
  
    set $memcached_key $uri;  
  
    memcached_pass 127.0.0.1:11211;  
  
}
```

如果你的应用程序读取请求参数来构造，那么\$memcached\_key 还应该包括：

```
location / {  
  
    set $memcached_key "$uri?$args";  
  
    memcached_pass 127.0.0.1:11211;  
  
}
```

如果请求的 key 当前不存在，那么 Nginx 将需要一种方法从应用程序请求该页面，希望该应用程序将 key/value 对写入 Memcached，以便下次请求能够直接从内存调用。如果请求的 key 在 Memcached 中没有被找到，那么 Nginx 将会报告一个“Not Found”错误，因此最好的方法是传递请求至应用程序使用 error\_page 指令来实现，并且由一个 location 来处理请求。我们也应该包括“Bad Gateway”和“Gateway Timeout”错误代码，万一 Memcached 不响应 key 的查找。

```
server {  
  
    location / {  
  
        set $memcached_key "$uri?$args";  
  
        memcached_pass 127.0.0.1:11211;  
  
        error_page 404 502 504 = @app;  
  
    }  
  
    location @app {  
  
        proxy_pass 127.0.0.1:8080;  
  
    }  
  
}
```

记住在 error\_page 指令的参数后使用等号 (=)，Nginx 将使用最后一个参数来替换返回代码，这样就会使得我们能够在返回错误代码的情况下又成为了一次正常的响应。

表 7-1 描述了 Memcached 模块可用的指令，该模块默认被编译到 Nginx 的二进制文件中。

表 7-1	Memcached 模块指令
指令	说明
memcached_buffer_size	用于设置来自 Memcached 响应的缓存大小。该响应被同步发送给客户端
memcached_connect_timeout	在向 Memcached 服务器产生一个请求后，Nginx 将会等待接受该请求的最长时间
memcached_next_upstream	设定在什么条件下将请求传递给下一个服务器，可以指定下面一个或者多个参数。 <ul style="list-style-type: none"><li>◆ error: 在同 Memcached 通信的过程中发生错误。</li><li>◆ timeout: 在同 Memcached 通信的过程中发生超时。</li><li>◆ invalid_response: Memcached 服务器返回了空的或者其他无效的响应。</li><li>◆ not_found: 在该实例上没有找到响应的 key。</li><li>◆ off: 禁止将请求传递到下一个 Memcached 服务器</li></ul>
memcached_pass	指定 Memcached 服务器的名字或者 IP 地址和端口号，也可能是在 upstream 中声明的一个服务器组
memcached_read_timeout	该指令用于指定一个时间长度，该时间长度是在一个连接关闭之前从 Memcached 服务器两次成功读操作的时间
memcached_send_timeout	该指令用于指定一个时间长度，该时间长度是在一个连接关闭之前从对 Memcached 服务器两次成功写操作时间

7.1.3 使用文件做缓存

假设你的应用程序以文件形式写了预呈现的页面，你知道每一个文件应该多久有效。你可以在任何代理和客户端之间配置 Nginx 投递指示每一个文件的某些头，以及缓存文件多久。在这种方式下，你为你的用户启用了本地缓存，而没有改变一行代码。

你可以通过设置 Expires 和 Cache-Control 头来实现，这些都是标准的 Http 头。客户端和 Http 代理“了解”这些头。不需要改变你的应用程序，如表 7-2 所示你只需要在 Nginx 的配置文件中相应的 location 中设置这些头就可以了。Nginx 提供 expires 和 add\_header 指令使得设置起来比较方便。

表 7-2 头修改指令	
指令	说明
add_header	在 Http 响应代码 200、204、206、301、302、303、304 或者 307 的响应头中添加字段
expires	添加或者修改 Expires 和 Cache-Control 头。参数可以是一个可选的 modified 参数，跟随一个时间，或者是 epoch、max、off 其中之一。如果单独设置了 time，Expires 头将会设置为当前的时间加上 time 指定的时间数值。Cache-Control 将被设置为 max-age=t，这里的 t 是指定的一个时间参数，单位为秒。如果 modified 参数设置为一个靠前的时间，那么 Expires 头被设置为文件的修改时间加上 time 参数指定的时间。如果 time 包含一个@，指定的时间将被解释为一天的时间，例如，@12h 是中午 12 点。epoch 确切的日期和时间被定义为 Thu, 01 Jan 1970 00:00:01 GMT。max 设置 Expires 为 Thu, 31 Dec 2037 23:55:55 GMT，Cache-Control 为 10 年。任何负值都将会把 Cache-Control 设置为 no-cache

知道你的应用程序生成的是什么文件，你就可以适当地设置这些头文件。让我们来看一个示例应用程序，其中的主页面被缓存 5 分钟，所有的 JavaScript 和 CSS 文件缓存 24 小时，每个 HTML 页面缓存 3 天，而每个图像要尽可能时间长。

```
server {

    root /home/www;

    location / {

        # match the index.html page explicitly so the *.html below
        # won't match the main page
        location = /index.html    {

            expires 5m;

        }

        # match any file ending in .js or .css (Javascript or CSS files)
        location ~* /\.*(js|css)$ {

            expires 24h;
```

```
    }

    # match any page ending in .html
    location ~* /\.html$ {

        expires 3d;

    }

}

# all of our images are under a separate location (/img)
location /img {

    expires max;

}

}
```

看到了这个配置如何设置头，我们在浏览器中看一下每一个 `location` 是什么样的。每一个现代的浏览器都有内置的或者是可用的插件来查看请求头和响应头。下面的快照是使用 Chrome 在这些 `location` 中的显示结果。

◆ **主页(index.html):** Expires 头设置的时间比 Date 晚 5 分钟。Cache-Control 头的 `max-age` 参数为 300 秒。



◆ **CSS 文件:** Expires 头设置的时间比 Date 晚 24 小时。Cache-Control 头的 `max-age` 参数为 86400 秒。

```
▼ Response Headers    view parsed
HTTP/1.1 200 OK
Server: nginx/1.2.2
Date: Sat, 15 Dec 2012 19:07:43 GMT
Content-Type: text/plain
Content-Length: 69
Last-Modified: Sat, 15 Dec 2012 18:31:33 GMT
Connection: keep-alive
Expires: Sun, 16 Dec 2012 19:07:43 GMT
Cache-Control: max-age=86400
Accept-Ranges: bytes
```

◆ **HTML 文件：**Expires 头设置的时间比 Date 晚 3 天。Cache-Control 头的 max-age 参数为 259200 秒。

```
▼ Response Headers    view parsed
HTTP/1.1 200 OK
Server: nginx/1.2.2
Date: Sat, 15 Dec 2012 19:10:16 GMT
Content-Type: text/html
Content-Length: 170
Last-Modified: Sat, 15 Dec 2012 18:39:12 GMT
Connection: keep-alive
Expires: Tue, 18 Dec 2012 19:10:16 GMT
Cache-Control: max-age=259200
Accept-Ranges: bytes
```

◆ **图像文件：**Expires 头设置为 Thu, 31 Dec 2037 23:55:55 GMT。Cache-Control 头的 max-age 参数设置为 315360000 秒。

```
▼ Response Headers    view parsed
HTTP/1.1 200 OK
Server: nginx/1.2.2
Date: Sat, 15 Dec 2012 19:07:43 GMT
Content-Type: image/jpeg
Content-Length: 26246
Last-Modified: Sat, 15 Dec 2012 18:28:41 GMT
Connection: keep-alive
Expires: Thu, 31 Dec 2037 23:55:55 GMT
Cache-Control: max-age=315360000
Accept-Ranges: bytes
```

只需在适当的位置设置一个指令 expires，我们便可以确保我们的预呈现的文件按照设置的生存期在本地缓存。

## 7.2 动态修改内容

有时候可能需要对来自于应用程序的内容做后期处理，也许你可能想在前端服务器页面上的某一位置上添加一个字符串然后再投递到客户端，或者想在呈现的页面上执行一个转换。Nginx 提供了 3 个可用的模块：addition、sub 和 xslt。

### 7.2.1 addition 模块

模块 addition 作为一个过滤器能够在响应的前或者后添加文本。默认编译的 Nginx 中没有包含该模块，因此如果想使用该功能，你必须在编译 Nginx 时添加 `--with-http-addition_module` 参数。该过滤器的原理是引用一个子请求，由该子请求附加一个请求或者是在开始放置一个请求。

```
server {
    root /home/www;
    location / {
        add_before_body /header;
        add_after_body /footer;
    }
    location /header {
        proxy_pass http://127.0.0.1:8080/header;
    }
    location /footer {
        proxy_pass http://127.0.0.1:8080/footer;
    }
}
```

表 7-3 是 addition 模块的指令列表。

表 7-3 Http addition 模块指令

指令	说明
add_before_body	在响应体之前添加了子请求处理结果
add_after_body	在响应体之后添加了子请求处理结果
addition_types	列出除了 text/html 之外其他的 MIME 类型，如果设置为*号，那么将会启用所有的 MIME 类型

## 7.2.2 sub 模块

该模块可以作为一个过滤器来替换一个文本为另一个文本。在默认情况下编译的 Nginx 没有包含该模块，因此如果你想使用这个模块功能，那么必须在 `configure` 时添加 `--with-http_sub_module` 参数。

该模块相当容易工作，你可以使用 `sub_filter` 指令指定一个字符串和一个被替代的字符串，过滤器在匹配时大小写不敏感。

```
location / {  
  
    sub_filter </head> '<meta name="frontend" content="web3"></head>';  
  
}
```

在前面的例子中，我们在页面的头中添加了一个新的 `meta` 标签，该标签通过 Nginx 来添加。也可能会有匹配一次以上的情况，要实现这种情况，就需要设置 `sub_filter_once` 指令为 `off`。这样对于在页面中使用绝对链接替代相对链接很有用。例如：

```
location / {  
  
    sub_filter_once off;  
  
    sub_filter '
```

表 7-7 SSI 命令

命令	参数	参数说明
block		定义一个小节, 该小节可以被 include 命令应用, 结尾是<!--#endblock-->
	name	小节的名称

续表

命令	参数	参数说明
config		设置全局参数，在整个 SSI 处理中都会被使用
	errmsg	配置字符串在 SSI 处理出错的时候将会抛出。默认值是[an error occurred while processing the directive]
	timefmt	传递给 strftime()的字符串，用于格式化时间戳在其他命令中使用。默认格式为 %A, %d-%b-%Y %H:%M:%S %Z
echo		输出一个变量的值
	var	变量名称，它的值将会被输出
	encoding	变量使用的编码方法。它的值是 none、url 和 entity 的三者之一。默认值为 entity
	default	如果变量没有定义，那么这个值将会输出。如果没有定义，none 为默认值
if		评估一个条件。如果条件为 true，小节将会被封闭。支持的次序 f、elsif、else 和 endif
	expr	表达式，用于评估真实性。 ◆ 变量存在性 (expr="\$var") ◆ 文本比较 (expr="\$var = text" 或者 expr="\$var != text") ◆ 正则表达式匹配 (expr="\$var = /regex/" 或者 expr="\$var != / regex/")
include		输出子请求的结果
	file	include 包括的文件名称
	virtual	包括子请求的 URI
	stub	被包含的区段，而不是一个空的内容，或者在处理中有错误
	wait	如果在同一个页面上有多个 include 命令，且设置了这个参数，那么它们将会被按照顺序处理
	set	如果在 virtual 中产生到 proxy_pass 或者 memcached_pass location 的子请求，那么结果可以被存储在该参数设置的变量中
set		创建一个变量并且为变量设置值
	var	设置变量的名字
	value	设置变量的值

一个 SSI 文件无非是用这些嵌入了注释的命令组成的 HTML 文件。这样一来，如果 ssi 在某一位位置不能够包含一个文件时，那么 HTML 部分仍将会呈现，尽管有些不完整。

下面 SSI 是调用一个子请求来呈现页眉、页脚和页面中的菜单的一个示例。

```
<html>
<head>
  <title>*** SSI test page ***</title>
  <link rel="stylesheet" href="/css/layout.css" type="text/css"/>
  <!--# block name="bofilerplate" -->
  <p>...</p>
  <!--# endblock -->
</head>
<body>
  <div id="header">
    <!--# include virtual="/render/header?page=$uri"
    stub="bofilerplate" -->
  </div>
  <div id="menu">
    <!--# include virtual="/render/menu?page=$uri"
    stub="bofilerplate" -->
  </div>
  <div id="content">
    <p>This is the content of the page.</p>
  </div>
  <div id="footer">
    <!--# include virtual="/render/footer?page=$uri"
    stub="bofilerplate" -->
  </div>
</body>
</html>
```

这段代码用于提供一些默认的内容，在这种情况下，错误将会在子请求中处理。

在处理逻辑上如果这些原语不能提供足够的灵活性，那么你可以使用嵌入式的 Perl 模块来解决任何其他处理或者你可能需要的配置。

## 7.4 在 Nginx 中的决策

你可能发现自己尝试以某种并不能够使用的方式扭曲 Nginx 的配置指令。这种情况经常在配置中看到，在使用许多 if 检查模仿某种逻辑链的情形中。一个好的选择是使用 Nginx

的嵌入 Perl 模块，使用这个模块，你将能够使用 Perl 的灵活性实现你的配置目标。

在默认情况下安装 Nginx 时，perl 模块是没有被安装在 Nginx 的运行二进制文件中的，因此如果要使用该模块，那么需要在 `configure` 时指定 `--with-http_perl_module` 选项，也要确保 Perl 在构建时使用了 `-Dusemultiplicity=yes`（或 `-Dusethreads=yes`）和 `-Dusemymalloc=no` 参数。Nginx 重载配置文件随着时间的推移将会导致 perl 模块内存泄漏，所以这最后一个参数用来帮助减轻这一问题。

在 Nginx 中使用了内置的 Perl 模块之后，那么就可以使用表 7-8 中的指令了。

表 7-8 Perl 模块指令

指令	说明
perl	在一个 location 中激活 perl 指令。它的参数是一个处理器的名字或者一个描述子程序的字符串
perl_modules	为 Perl 模块指定一个额外的搜索路径
perl_require	指明一个 Perl 模块，每次在 Nginx 重新配置后将会重新载入。对于单独的模块可能需要指定多次
perl_set	安装一个 Perl 处理程序，用于设置一个变量。这个参数是处理程序的名字或者一个描述子程序的字符串

在 Nginx 配置文件中编写 Perl 脚本时，你会使用 `$r` 对象代表该请求。这个对象的方法如下。

- ◆ `$r->args`：请求参数。
- ◆ `$r->filename`：被 URI 应用的文件名。
- ◆ `$r->has_request_body(handler)`：如果有请求体，处理程序将被调用。
- ◆ `$r->allow_ranges`：在响应中启用字节范围。
- ◆ `$r->discard_request_body`：丢弃请求体。
- ◆ `$r->header_in(header)`：指定的请求头值。
- ◆ `$r->header_only`：Nginx 只向客户端返回响应头。
- ◆ `$r->header_out(header, value)`：设置指定的响应头。
- ◆ `$r->internal_redirect(uri)`：在 Perl 处理程序执行完成之后将会对指定

的 URI 生成重定向。

- ◆ `$r->print (text)`: 将指定的文本发送给客户端。
- ◆ `$r->request_body`: 在内存中的请求体。
- ◆ `$r->request_body_file`: 如果被写入到一个临时文件, 那么会返回请求体的内容。
- ◆ `$r->request_method`: 请求的 **Http** 方法。
- ◆ `$r->remote_addr`: 客户端的 IP 地址。
- ◆ `$r->flush`: 立即向客户端发送数据。
- ◆ `$r->sendfile (name[, offset[, length]])`: 发送指定的文件到客户端, 可选参数有 `offset` 和 `length`, 在 Perl 处理程序执行完成之后发送。
- ◆ `$r->send_http_header ([type])`: 向客户端发送响应头, 还可以指定可选的内容类型。
- ◆ `$r->status (code)`: 设置 **Http** 响应的状态代码。
- ◆ `$r->sleep (milliseconds, handler)`: 设置处理程序执行的定时器, 在这个设定的时间完成之后, **Nginx** 将会继续处理其他的请求, 而定时器仍在运行。
- ◆ `$r->unescape (text)`: 解码后的 URI 编码文本。
- ◆ `$r->uri`: 请求中的 URI。
- ◆ `$r->variable (name[, value])`: 设置一个命名的本地请求变量或者是设置一个指定的值。

Perl 模块也可以用在 SSI 中, 使用 Perl 的 SSI 命令格式如下。

```
<!--# perl sub="module::function" arg="parameter1" arg="parameter2"
... -->
```

我们来看一下使用 Perl 模块的例子, 我们的目标是传递请求到不同的上游服务器 (upstream server), 由请求的 URI 第一个字母决定传递到哪个服务器。

在 **Nginx** 中, 我们可以实现一系列的 location, 但它会更简洁地表示为一个 Perl 处理程序。

第一步是在 Perl 处理器中定义一个 Action。

```
# upstreammapper.pm

# name our package
package upstreammapper;

# include the nginx request methods and return code definitions
use nginx;

# this subroutine will be called from nginx
sub handler {

    my $r = shift;

    my @alpha = ("a".. "z");

    my %upstreams = ();

    # simplistically create a mapping between letter and
    # an IP which is between 10 and 35 of that network
    foreach my $idx (0..$#alpha) {

        $upstreams{ $alpha[$idx] } = $idx + 10;

    }

    # get the URI into an array
    my @uri = split(//, $r->uri);

    # so that we can use the ffirst letter as a key
    my $ip = "10.100.0." . $upstreams{ $uri[1] };

    return $ip;

}

1;

__END__
```

然后再设置 Nginx 使用这个模块做映射。

```
http {

    # this path is relative to the main configuration file
    perl_modules perl/lib;

    perl_require upstreammapper.pm;
```

```
# we'll store the result of the handler in the $upstream variable
perl_set $upstream upstreammapper::handler;
```

然后将请求传递到正确的上游服务器。

```
location / {

    include proxy.conf;

    proxy_pass http://$upstream;

}

}
```

我们已经看到了在 Perl 处理程序中实施一些简单的配置逻辑，几乎任何类型的特殊要求都可以以类似的方式来完成。



在 Perl 处理程序中，请求处理尽可能地明确，当 Nginx 必须等待一个 Perl 处理程序完成时，负责处理该请求的整个 worker 将会被阻塞。因此，任何 I/O 或者与 DNS 相关的任务应该在 Perl 处理程序之外进行。

## 7.5 创建安全链接

可能由于某种原因你需要对网站的某些内容进行保护，但是对于这些内容你又不想集成完整的用户认证系统。在这种情况下，可以启用 Nginx 的 `secure_link` 模块，在编译安装 Nginx 时添加 `--with-http_secure_link` 就可以了，再通过使用 `secure_link_secret` 指令以及其相应变量 `$secure_link` 来实现。

模块 `secure_link` 通过计算 `secure_link_secret` 指令之后提供的安全字的 MD5 哈希值工作。如果这个哈希值与 URI 中找到的哈希值匹配，那么 `$secure_link` 变量将会被设置为哈希值且为 URI 的一部分；如果没有匹配，那么 `$secure_link` 变量的值将会被设置为空字符串。

一种可能使用的情形是使用密码生成的 MD5 哈希值作为下载页面的一个链接，然后将密码配置在 Nginx 的配置文件中，以便能够使得用户访问这些链接。这个词和页面要定期更换，以防止以后再次调用被保存的链接，下面的例子说明了这种情况。

我们首先设定一个密码为 `supersecret`，然后再生成它的 MD5 哈希值作为想要使用

的哈希值。

```
$ echo -n "alphabet_soup.pdfsupersecret" |md5sum
8082202b04066a49a1ae8da9ec4feba1 -
$ echo -n "time_again.pdfsupersecret" |md5sum
5b77faadb4f5886c2ffb81900a6b3a43 -
```

接下来我们创建链接的 HTML 代码。

```
<a href="/downloads/8082202b04066a49a1ae8da9ec4feba1/alphabet_soup.
pdf">alphabet soup</a>
<a href="/downloads/5b77faadb4f5886c2ffb81900a6b3a43/time_again.
pdf">time again</a>
```

如果在 Nginx 的配置文件中添加 `secure_link_secret` 指令，并且是使用同样的密码，那么这些链接是有效的。

```
# any access to URIs beginning with /downloads/ will be protected
location /downloads/ {

    # this is the string we used to generate the hashes above
    secure_link_secret supersecret;

    # deny access with a Forbidden if the hash doesn't match
    if ($secure_link = "") {

        return 403;

    }

    try_files /downloads/$secure_link =404;

}
```

要确保链接没有哈希值将不会工作，我们可以在 HTML 中添加一个额外的链接。

```
<a href="/downloads/bare_link.pdf">bare link</a>
```

调用这个链接时会报告是“403 Forbidden”错误。



对于解决前面描述的 `secure_Link` 模块这类技术问题，Nginx 自身就提供了一个可选的方法，参见 <http://wiki.nginx.org/HttpSecureLinkModule>。

## 7.6 生成图像

你可以配置 Nginx 处理一些简单的图像转换，而不是编写一个图像处理模块。如果你的图像处理需求就像旋转图像、调整其大小或裁剪它一样简单，那么 Nginx 完全可以胜任。

要使用这个功能，你需要安装 libgd 库，并且要在编译 Nginx 时加入 image\_filter 模块（--with-http\_image\_filter\_module），如果你这样做了，那么你就可以使用下列在列表中的指令了。



GD 库（libgd）是一个用 C 语言写的图像生成库。它经常被编成语言组合使用为站点生成图像，例如，PHP 或者 Perl。Nginx 的 image\_filter 模块使用 libgd 提供创建一个简单的图像调整大小的代理的能力，我们会在下面的例子中讨论。

表 7-9 图像过滤指令

指令	说明
<code>empty_gif image_filter</code>	<p>使用该指令会在相应的 location 中生成 1x1 的透明 GIF 文件。根据以下参数变换图像。</p> <ul style="list-style-type: none"><li>◆ <code>off</code>: 关闭图像变换。</li><li>◆ <code>test</code>: 确保响应是 GIF、JPEG 或 PNG 图像。如果不是，返回错误 415（不支持的媒体类型）。</li><li>◆ <code>size</code>: 发送关于 JSON 格式的图像信息。</li><li>◆ <code>rotate</code>: 逆时针旋转图像，90、180 或 270 度。</li><li>◆ <code>resize</code>: 在给定的宽度和高度下成比例的缩小。为了只减小一个方面的尺寸，可以将另一个方面指定为“-”。</li><li>◆ <code>crop</code>: 按照给定的最大边长减小图像，长宽是给定的，任何多余的部分沿其他边缘被削减。为了减少一个维度的大小，可以将另一个维度的大小设置为“-”。如果通过 <code>rotate</code> 组合，那么在缩小之前会旋转。发生错误返回 415 代码（不支持的媒体类型）</li></ul>
<code>image_filter_buffer</code>	<p>处理图像所使用的缓冲大小。如果需要更多的内存，那么服务器将会返回 415 错误（不支持的媒体类型）</p>

续表

指令	说明
image_filter_jpeg_quality	处理结果是 JPEG 图像的质量品质。不推荐超过 95
image_filter_sharpen	通过这个百分比增加了处理后图像的清晰度
image_filter_transparency	禁用 GIF 和 PNG 图像保持透明度变换。默认 on，保留透明度

注意，`empty_gif` 指令不是 `image_filter` 模块的一个部分，但是包含在默认安装的 Nginx 中。使用这些指令，我们能够构造一个缩放图像的模块如下。

```
location /img {

    try_files $uri /resize/$uri;

}

location ~* /resize/(?<name>.*)(?<width>[[:digit:]]*)x(?<height>[[:digit:]]*)\.(?<extension>gif|jpe?g|png)$ {

    error_page 404 = /resizer/$name.$extension?width=$width&height=$height;

}

location /resizer {

    image_filter resize $arg_width $arg_height;

}
```

在上面的配置片段中，首先尝试提供一个按照 URI 请求的图像，如果没有找到合适的名字的图像，就会移动到 `/resize` 位置。`/resize` 这个 `location` 是像一个正则表达式那样定义的 `location`，以便我们能够捕获我们希望大小的图像。注意我们使用命名捕获组来创建有意义的变量名字，然后将这些传递到 `/resize` 中，以便我们能够有作为 URI 的原始文件的名称和作为命名参数的宽和高。

我们组合 Nginx 的 `proxy_store` 或者 `proxy_cache` 功能来保存调整大小后的图像，以便同样 URI 的其他请求不再需要“击中” `image_filter` 模块。

```
server {

    root /home/www;

    location /img {

        try_files $uri /resize/$uri;

    }

    location /resize {

        error_page 404 = @resizer;

    }

    location @resizer {

        internal;

        proxy_pass http://localhost:8080$uri;

        proxy_store /home/www/img$request_uri;

        proxy_temp_path /home/www/tmp/proxy_temp;

    }

}

server {

    listen 8080;

    root /home/www/img;

    location ~* /resize/(?<name>.*)(?<width>[[:digit:]]*)
x(?<height>[[:digit:]]*)\.(?<extension>gif|jpe?g|png)$ {

        error_page 404 = /resizer/$name.$extension?width=$width&heigh
```

```
        t=$height;

    }

    location /resizer {

        image_filter resize $arg_width $arg_height;

    }

}
```

正如你在 `image_filter` 模块指令表中看到的，由该模块返回的任何错误代码都是 415。我们可以“抓住”这个代码将其替换为一个空的 GIF，以便最终用户仍旧获取的是一个图像而不是一个错误消息。

```
location /thumbnail {

    image_filter resize 90 90;

    error_page 415 = @empty;

}

location = @empty {

    access_log off;

    empty_gif;

}
```

尺寸参数 `image_filter` 特别值得一提，在将这个参数配置在 `location` 中后，投递给客户端的不是图像本身而是有关图像本身的一些信息。这个用法在你的应用程序中在调用调整或者修剪 URI 之前，对于发现有关图像的元数据信息很有用。

```
location /img {

    image_filter size;

}
```

结果是一个 JSON 对象，例如：

```
{ "img" : { "width": 150, "height": 200, "type": "png" } }
```

## 7.7 跟踪网站的访问者

跟踪特定网站访问者，一个不太显眼的方式是使用 `userid` 模块。这个模块会设置用于识别唯一客户端的 `cookie`。这些 `cookies` 的值通过 `$uid_set` 变量应用获取。当同一用户返回到该网站，该 `cookie` 仍然有效，该值在变量 `$uid_got` 中有效。看下面的这个例子如何使用这些变量。

```
http {

    log_format useridcomb '$remote_addr - $uid_got [$time_local] '
                          '"$request" $status $body_bytes_sent '
                          '"$http_referer" "$http_user_agent"';

    server {

        server_name .example.com;

        access_log logs/example.com-access.log useridcomb;

        userid      on;

        userid_name  uid;

        userid_domain example.com;

        userid_path  /;

        userid_expires 365d;

        userid_p3p     'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR
        STA NID"';

    }

}
```

这些指令汇总于表 7-10。

表 7-10 UserID 模块指令

指令	说明
Userid	<p>通过下面的参数来激活该模块。</p> <ul style="list-style-type: none"><li>◆ on: 启用版本 2 的 cookies，并且记录它们。</li><li>◆ v1: 启用版本 1 的 cookies，并且记录它们。</li><li>◆ log: 不发送 cookies，但是记录进入的 cookies。</li><li>◆ off: 不发送 cookies，也不记录到日志</li></ul>
userid_domain	配置域设置 cookie
userid_expires	设置 cookie 的生存期。如果使用了关键字 max，那么将会为浏览器设置生存期到 31 Dec 2037 23:55:55 GMT
userid_name	设置 cookie 的名字（默认为 uid）
userid_p3p	配置 P3P 头，用于使用 p3p（Platform for Privacy Preferences）协议的站点
userid_path	设置 cookie 的路径
userid_service	设置服务 cookie 标识。例如，如果是版本二，那么 cookies 将会设置为服务器的 IP 地址

## 7.8 防止意外的代码执行

当试图构造一个你希望它做什么的配置时，你可能会无意中允许了你所不愿意的。看看下面的配置块。

```
location~* \.php {

    include fastcgi_params;

    fastcgi_pass 127.0.0.1:9000;

}
```

从这个配置中可以看出，我们对 PHP 文件的请求都将会被传递到 FastCGI 服务器响应并处理这些请求。如果给定传入的文件仅是 PHP 文件，那么这个配置是没问题的，但是由于 PHP 编译和配置的不同性可能结果并非总是如此。如果用户上传的是包含 PHP 文件的相同目录结构，那么这将会是一个问题。用户可能会阻止以 .php 为后缀的文件上传，但允许上传 .jpg、.png 和 .gif 文件。恶意用户可以上传嵌入 PHP 代码的图像文件，这样的结果是导致 FastCGI 服务器通过传递一个上传的文件名的 URI 来执行这段代码。

要阻止这类事情发生，可以设置 PHP 的参数 `cgi.fix_pathinfo` 的值为 0，也可以在 Nginx 配置文件中添加以下类似配置。

```
location ~* \.php {  
  
    try_files $uri =404;  
  
    include fastcgi_params;  
  
    fastcgi_pass 127.0.0.1:9000;  
  
}
```

在上面的配置中我们使用 `try_files`，以确保将请求传递到 FastCGI 服务器处理之前 PHP 文件确实存在。



请记住，你应该评估你的配置，查看配置是否适合您的目标。如果只有几个文件，你最好通过明确指定的 PHP 文件的方式来执行，而不是使用正则表达式的 `location` 和相应的 `try_files` 方式。

## 7.9 总结

Nginx 提供了多种方法来支持开发人员将高性能的 web 服务器集成到他们的应用中，我们看到了集成旧的和新的应用程序下的各种可能。为了尽快地投递 Web 网页，Nginx 提供了被动和主动的方法使用缓存。

我们还探讨了 Nginx 如何通过添加或替换文本来操纵响应。Nginx 也可以使用 SSI。我们看到了将这些命令集成到普通的文本文件中的一个方法。然后，我们验证了在 Nginx 中

嵌入了强大的 Perl 功能。在 Nginx 中图像转换模块也可能被使用，我们研究了如何设置一个唯一的 cookie 来跟踪网站访问者。本章我们围绕关于如何阻止意外程序运行讨论。总体上，当 Nginx 作为 web 服务器时，开发者有相当多的工具与 Nginx 协同工作。

在下一章，我们将探讨故障排除技术，在出现不按预期方式工作时解决问题。