Community Experience Distilled

学习 Nginx HTTP Server（中文版）

（法）Clément Nedelcu 著
陶利军 译

清华大学出版社

学习 Nginx HTTP Server（中文版）

Nginx服务器/反向代理配置指南

指令和模块配置完整参考

Nginx（发音同“engine x”）是一款轻量级的Web 服务器／反向代理服务器及电子邮件（IMAP/POP3）代理服务器。最初由俄罗斯程序员Igor Sysoev开发，供俄罗斯第二大门户网站和搜索引擎Rambler使用。Nginx具有诸多优点，如稳定性高，功能丰富，示例配置文件丰富，占有内存少和并发能力强，等等。目前，使用Nginx的网站有新浪、网易、腾讯、人人网、开源中国社区以及国内几个重要的视频网站和博客网站。

本书具有以下特色：

- 下载和构建Nginx
- Nginx基本配置：语法、结构和语义
- 轻松创建虚拟主机配置
- 掌握如何激活、配置和使用所有模块
- 用Nginx Rewrite模块建立高级重写规则
- 设置Nginx，使其通过FastCGI与PHP, Python等结合使用
- 配置Nginx，将其用作现有HTTP服务器的前端
- 用Nginx替换Apache
- 通过3个实例演示如何移植Apache重写规则
- 通过详细的指令参考进行配置
- 故障排除提示

ISBN 978-7-302-27089-8
9 787302 270898 >
定价：49.00元

学习 Nginx HTTP Server
(中文版)

(法) Clément Nedelcu 著

陶利军 译

清华大学出版社

北 京

内 容 简 介

本书是 Nginx 新手管理员和资深管理员的理想读物。对于初学者，可从中学习如何以快速而安全的方式安装 Nginx 并对各个模块进行配置。对于有经验的管理员，它提供了不同视角的解决方案。书中提供 Nginx 所有模块和指令的完整参考，解释了如何用 Nginx 取代现有服务器，如何将 Nginx 配置为现有服务器的前端系统。完成本书的阅读后，读者能够轻松实现 Nginx 服务器，提升 Web 应用的速度。

北京市版权局著作权合同登记号　图字：01-2011-3512
本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

学习 Nginx HTTP Server(中文版)/(法)内德尔库(Nedelcu, C.)著；陶利军译.
--北京：清华大学出版社，2012
书名原文：Nginx HTTP Server
ISBN 978-7-302-27089-8

Ⅰ. ①学…　Ⅱ. ①内… ②陶…　Ⅲ. ①互联网络—网络服务器　Ⅳ. ①TP368.5

中国版本图书馆 CIP 数据核字(2011)第 207803 号

责任编辑：文开琪
封面设计：杨玉兰
责任校对：周剑云
责任印制：李红英
出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编**：100084
社 总 机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京密云胶印厂
经　销：全国新华书店
开　本：178×233　**印　张**：22.25　**字　数**：436 千字
版　次：2012 年 1 月第 1 版　　**印　次**：2012 年 1 月第1 次印刷
印　数：1～4000
定　价：49.00 元

产品编号：041269-01

关于审阅者

Pascal Charest

担任 Les Laboratoires Phoenix 资深首席顾问——一个系统性能信息咨询公司，总部设在加拿大，他们使用先进的算法和自由软件，他被称为“主题专家”，管理基础设施项目，带领运作，并执行过程确认。

在过去的一年，他的任务包括为一个大型的北美投资集团重新设计存储系统，管理电信级运营，一个杰出的电信行业的国际网络成员。他还领导了好几个本地创业行动，并且通过自定义的可扩展性的云计算解决方案/网络基础设施解决需求。他也是一个自由软件/社区倡导者，并经常在有关信息系统的可扩展性问题会议上讲话。他的邮件地址为 pascal.charest@labsphoenix.com，可以通过这个地址来与他联系。

感谢 Catherine，我亲爱的，感谢你所做的一切，所以我不用这样做。

Manlio Perillo

在意大利生活，在伊尔皮尼亚地区，那不勒斯附近。他目前是自由程序员，主要使用 Python 和 Nginx 开发 Web 应用程序。在 2008 年，他开始致力于 WSGI(Python Web 服务器网关接口)、基于 Nginx 的执行以及一些其他的开源项目，详见 *http://bitbucket.org/mperillo/*。

关于作者

Clément Nedelcu

出生并成长于法国，在英国、法国和中国的大学接受过教育。他曾在中国江苏科学技术大学担任计算机教师。他同时兼任法国企业的技术顾问，他精通 Web 和 Microsoft .NET 开发以及 Linux 服务器管理。自 2005 年以来，他在业余时间一直管理着一个大型的网站，最终领略到 Nginx 的好处，由此开博讨论 Nginx，由此有了本书……

作者的博客地址：*http://cnedelcu.net*。他的博客包含 Nginx 和其他 Web 开发话题的相关文档。

我要感谢我的女友、我的家庭和我的朋友，在写作阶段，他们非常支持我。本书献给 Martin Fjordvald，在我的服务器快要死掉的时候，是他第一个向我介绍 Nginx。特别感谢 Maxim Dounin、Jérémie Bertrand、Shaun James、Zhang Yichun、Brendan 和 Freenode IRC Nginx 频道所有的人。

目　录

前　言

Web 服务器市场长期以来的领头羊是 Apache，这是一个众所周知的事实。根据最近的调查，截至 2009 年 10 月，超过 45%的 Web 服务器都是由这个有着 15 年悠久历史的开源应用提供的。然而，过去几个月同样的一份报告，也揭示了一个新的竞争对手崛起：Nginx，一个来自俄罗斯的轻量级 HTTP 服务器——读作“engine X”。对于这个新生事物，疑问也不少：为什么博客世界对它的出现表现得如此兴奋？是什么原因造成如此多的服务器管理员自 2009 年开始纷纷切换到 Nginx？这个小软件成熟到足以运行高流量网站了吗？

首先，Nginx 并不是想象的那么年轻。它最初开始于 2002 年，该项目由一个独立开发人员实现，他就是 Igor Sysoev，当时的目的是满足一个访问流量极大的俄罗斯门户网站。该网站在 2008 年 9 月每天收到的 HTTP 请求高达 50 亿。该 Web 应用现在由一些非常流行的网站用来提供 Web 服务，例如 WordPress、Hulu、SourceForge 等许多网站。事实证明，Nginx 是一个非常高效的、轻量级的但功能强大的 Web 服务器。在本书中，你会发现 Nginx 的许多功能，会理解为什么这么多管理员都会放弃 Apache 而信赖这个新的 HTTP 服务器。

在许多方面，Nginx 都比竞争对手更高效。首先最重要的一点是速度。通过利用异步套接字，Nginx 在收到请求时，不会派生出与请求一样多的子进程。考虑减轻 CPU 负载和内存消耗，每个核(core)一个进程足以处理数千个连接。其次是易用——与其他 Web 服务器解决方案(例如 Apache)相比较，配置文件的读取和调整都十分简单。只需几行就足以建立一个完整的虚拟主机配置。最后但同样重要的是模块性。Nginx 不但是一个在类 BSD 许可下发布的完全开源项目，它还自带一个强大的插件系统——称为“模块”。原始发布归档文件中包含种类繁多的模块，许多第三方模块都能从网上下载。总的来说，Nginx 集速度、效率和能力于一体，提供了成功 Web 服务器的完美组合。迄今为止，相对于 Apache 来说，它是最好的选择。

尽管 Nginx 从 0.7.52 版后开始支持 Windows，但是众所周知，对于生产网站来说，首选的还是基于 Linux 的发布。对于本书中所描述的各个过程，我们假设你的网站基于 Linux 操作系统，例如 Debian、Fedora、CentOS、Mandriva 或者其他众所周知的发布版本。

本书主题

第 1 章“准备工作环境” 提供了一个基本的方法，帮助读者认识 Linux 命令行环境，本书自始至终都会使用这个方法。

第 2 章“下载和安装 Nginx” 通过下载和安装 Nginx，也包括先决条件的下载和安装，以帮助读者了解整个安装过程。

第 3 章“Nginx 的基本配置” 帮助你认识 Nginx 的基本配置和设置 Core 模块。

第 4 章“HTTP 配置” 详细讲述 HTTP 核心模块，包括主要的配置模块和指令。

第 5 章“模块配置” 帮助你认识 Nginx 的许多官方模块，包括 Rewrite 模块和 SSI 模块。

第 6 章“Nginx 与 PHP、Python” 解释了如何设置 PHP 和其他第三方应用程序(如果你对动态网站感兴趣)以通过 FastCGI 与 Nginx 协同工作。

第 7 章“Nginx 和 Apache” 教你设置 Nginx 为反向代理服务器，使其与 Apache 一同提供服务。

第 8 章“从 Apache 到 Nginx” 详细指导读者如何从 Apache 切换到 Nginx。

附录 A“指令索引” 列出并描述了所有配置指令，按照字母顺序排序。模块提供的指令在相关章节中也有描述。

附录 B 列出了模块的参考。

附录 C“疑难解答” 讨论了管理员在配置 Nginx 时可能面对的最常见的一些问题。

本书前提条件

Nginx 是一个自由、开源的软件，它可以运行于各种不同的操作系统——基于 Linux、Mac OS、Windows 操作系统，等等。因此，就软件而言，并没有真正的需求。不过，本书尤其是前两章，我们将在 Linux 环境下工作，所以会优先考虑运行基于 Linux 的操作系统。针对 Nginx 的编译安装有哪些前提条件，则在第 2 章讲述。

本书读者对象

无论是 Nginx 的初学者，还是有经验的管理员，都可以将本书视为良师益友。对于前者，本书将带领你经历一个完整的过程——从下载、编译、安装，到配置各种模块，

直到完整建立一个轻量级的 HTTP 服务器；对于后者，它提供来自不同角度的方法，能帮助你充分利用现有的基础设施。随着本书描述的深入，还将提供一个完整的 Nginx 模块和指令参考，解释如何利用 Nginx 取代现有服务器或让 Nginx 作为现有服务器的前端服务器。

本书排版约定

在本书中，你将发现我们通过大量文本样式来区分不同类型的信息。下面有一些示例及其相关的说明。

代码通常以等宽字体显示。格式如下：

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

当我们希望您特别注意代码块中的某些部分时，会将相关的部分加粗显示，如下所示：

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

所有命令行输入或输出都采用以下形式：

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
/etc/asterisk/cdr_mysql.conf
```

新的术语和重要的词语都采用粗体显示。屏幕上所看到的单词，如菜单或对话框中的，在文中将采用这样的格式：“单击 **Next** 按钮移到下一屏。”

[警告或重要说明出现在这样的提示框中。]

[提示和技巧出现在这样的提示框中。]

特别说明，为便于读者参考原书索引(见 305 页)，中文版有意与原文对照，所以有的页面可能比较松，有意保留了一些空白。

读者反馈

我们始终欢迎读者的反馈意见。请让我们知道您对这本书的看法——喜欢哪些地

方，或者不喜欢哪些地方。读者反馈对我们出版好书至关重要。

向我们发送普通的读者反馈，只需发送邮件到 feedback@packtpub.com 并在邮件主题中提及该书的书名即可。

如果有您需要并希望我们出版的书，请按照 www.packtpub.com 提供的 SUGGEST A TITLE 表单，向我们提交相关说明，或者发送邮件到 suggest@packtpub.com。

如果有您感兴趣或有经验写作或参与写作的议题，请阅读 www.packtpub.com/authors 提供的著译者说明。

读者支持

现在，您已经自豪地拥有了 Packt 的书，我们将通过大量的支持来让您从图书中收获更多。

勘误表

尽管我们已经竭尽全力确保内容的准确性，但错误在所难免。如果您在我们的任何一本书中发现了问题——可能是文本错误或者是代码错误，我们感谢您向我们报告错误。通过这样的方式，您可以帮助其他读者减少挫败并帮助我们改进这本书后续版本。如果您发现任何错误，请访问 http://www.packtpub.com/support 报告您发现的错误，具体作法是选择相应的书名称，然后点击勘误表提交表的链接，填写详细的错误信息。一旦您提交的勘误表被核实，就会被我们接受并被上传到我们的网站上，或者增加到现有勘误表中，就在这本书的“勘误表”这个部分。在 http://www.packtpub.com/support 选择书名，就可以查看所有已经提交的勘误表。

隐私

互联网上，受版权保护作品的侵权情况屡禁不止。在 Packt，我们非常谨慎地保护我们的知识产权和许可。如果您发现互联网上有我们所出版的作品的任何形式的任何非法副本，请为我们提供地址或网站名称，以便我们迅速采取补救措施。

如果发现可疑的盗版或者侵权作品，请通过邮件将链接地址发送到 copyright@packtpub.com 告知我们。

我们非常感激您帮助我们保护作者以及保护我们向您提供高价值内容的能力。

疑难解答

如果您对于本书有任何问题，都可以发送邮件到 questions@packtpub.com，我们将极尽全力帮助您解决。

第 1 章
准备工作环境

在这一章中，我们将指导你通过这些步骤来准备工作环境，包括计算机和网站服务器。要想建立一个功能齐全的 Nginx，需要理解许多事情，尤其是使用 Windows 操作系统的读者。

本章主题

- 设置终端仿真器，用于通过命令行界面连接到远程服务器
- 基本的 Linux 命令行工具，供不同阶段使用
- 介绍 Linux 文件系统结构
- 系统管理工具
- 管理文件和权限

设置终端仿真器

对于过去十五年天天都在 Windows 操作系统下工作的我们来说，这个想法可能又使我们回到一个好的、老的基于命令行的界面，看起来好像稍微有点原始，但尽管如此，这对于大多数服务器管理员来说是必要的。第一个准备工作是下载和安装一个 SSH 客户端(Secure Shell，SSH)，这是一个网络协议，通过这个协议能够使两个设备进行安全的通信，安全的保证在于交换的数据是加密的。它的作用显而易见，就是用于连接远程系统的 shell，换句话说，就是在不损害安全的情况下能够控制服务器。

查找并下载 PuTTY

在 Windows 操作系统下，对于使用 SSH 的访问者来说，PuTTY 是一个使用广泛的终端仿真器。因此，可以在网上找到大量的文章和其他文档，该软件会提供各种不同的功能。我们将只涵盖与这里相关的功能——配置 PuTTY 连接服务器，输入文本，使用复制和粘贴命令。但应该知道还有很多免费的开源工具能够建立 SSH 通道，连接到 Telnet、Rlogin 甚至是原始 TCP 通信(raw TCP communication)，等等。

可以直接从作者的网站下载 PuTTY：

http://www.chiark.greenend.org.uk/~sgtatham/putty/

它只有一个单独的.EXE 程序，不需要任何额外的文件。它的所有数据都保存在 Windows 注册表中，所以不会用配置文件来塞满你的系统。

建立会话

在继续阅读之前，首先确定下列元素：

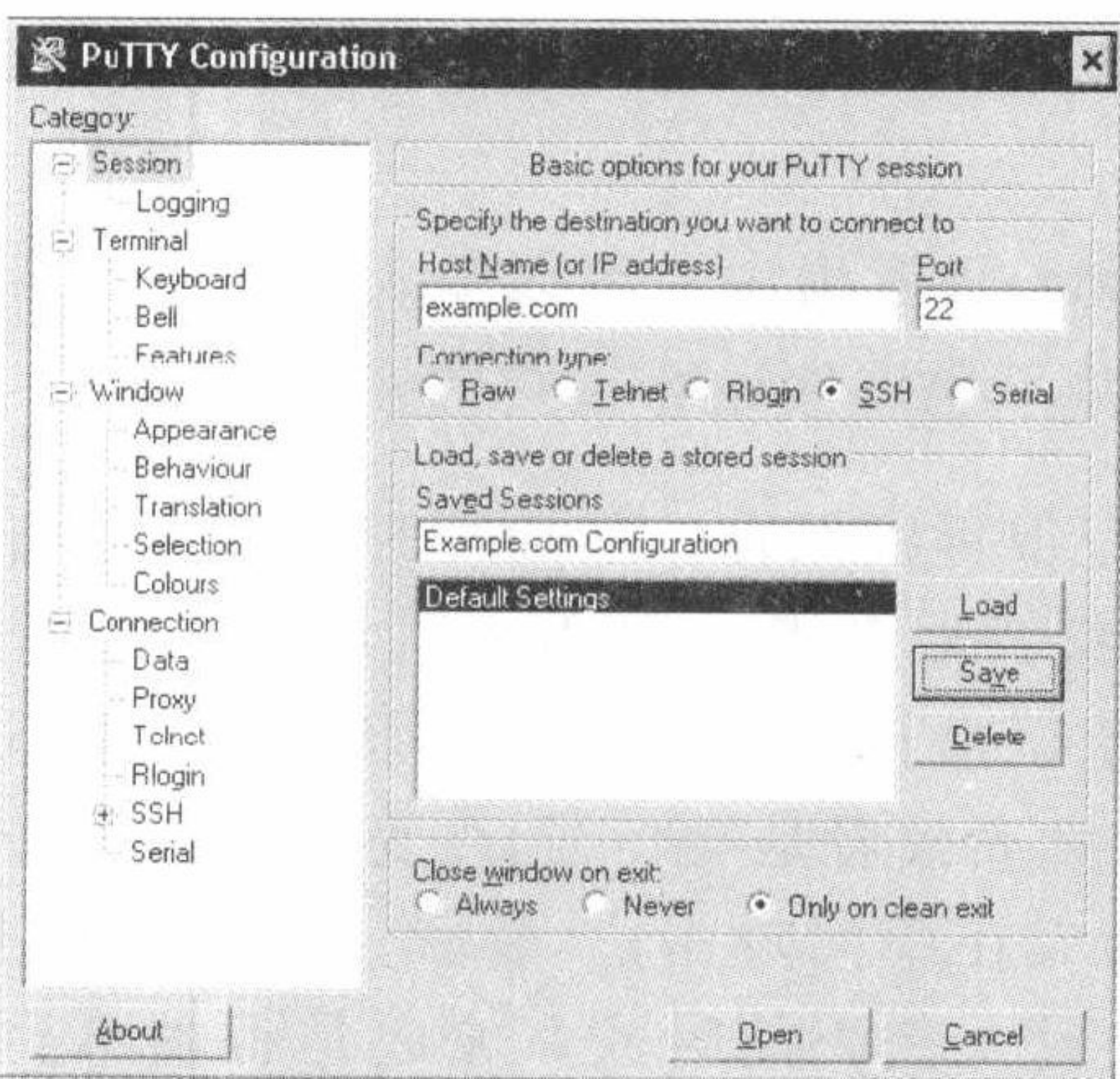
- 准备连接的主机名或 IP 地址；
- SSH 守护进程的端口号，除非已知有其他端口号，否则该服务默认的是端口号是 22；
- 系统的一个用户帐号；
- 帐号的密码。

让我们快速看一下 PuTTY 的主要窗口(见下图)。

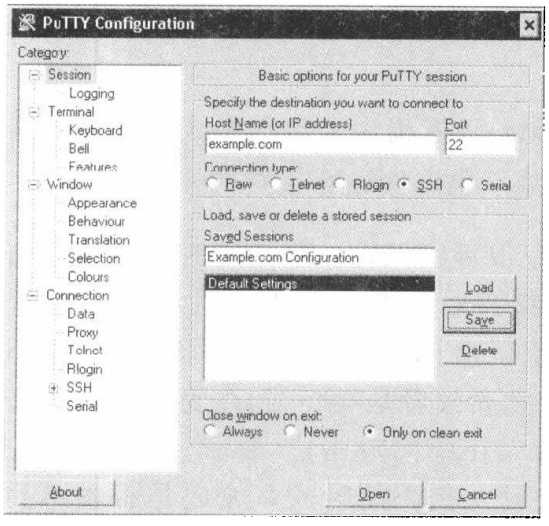
PuTTY 会保存 session 中的设置，因此在完成各种参数配置后，确定指定该会话的名称后再单击 Save 按钮，如上图所示。

在默认的 PuTTY 界面上，需要填入想要连接的远程服务器的**主机名**(或者 **IP 地址**)，然后配置运行在远程服务器上的 SSH 服务提供的端口号，SSHD 默认的端口号是 22。这里另外有几个设置选项，可能有用。

- 在 **Window** 设置组，可能需要调整一些参数，例如终端窗口的大小和能够回显的行数。
- 在 **Window | Appearance** 设置组，可以改变在终端窗口中显示的字体、字形及字号的大小，也包括光标选项。
- 在 **Window | Translation** 设置组，可以指定不同的字符集，这对运行 UTF-8 字符集的服务器非常有用。


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
example.com
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Example.com Configuration
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel

让我们快速看一下 PuTTY 的主要窗口(见下图)。



PuTTY 会保存 session 中的设置，因此在完成各种参数配置后，确定指定该会话的名称后再单击 Save 按钮，如上图所示。

在默认的 PuTTY 界面上，需要填入想要连接的远程服务器的**主机名**(或者 **IP 地址**)，然后配置运行在远程服务器上的 SSH 服务提供的端口号，SSHD 默认的端口号是 22。这里另外有几个设置选项，可能有用。

- 在 **Window** 设置组，可能需要调整一些参数，例如终端窗口的大小和能够回显的行数。
- 在 **Window | Appearance** 设置组，可以改变在终端窗口中显示的字体、字形及字号的大小，也包括光标选项。
- 在 **Window | Translation** 设置组，可以指定不同的字符集，这对运行 UTF-8 字符集的服务器非常有用。

- 在 **Connection** 设置组，可开启 **TCP keepalives** 功能，以防止 TCP 超时而导致断开。
- 在 **Connection | Data** 设置组中，可以输入要连接的系统的帐户，然而由于安全的原因，PuTTY 并不允许你保存用户的密码。

一旦完成会话的配置，记得再次保存，然后再开始单击主窗口上的 **Open** 按钮建立连接。如果是第一次连接到服务器，则需要通过接受服务器端的指纹(fingerprint)来证实它的真实性。如果以后再连接到同一台服务器，则不会再看到确认的信息，除非服务器设置(例如主机名或者是端口号)改变，或者是安全性被损坏，你连接到了一个中间的服务器(有人在中间攻击)。最后，提示注册(除非你激活自动登录选项)和密码。请注意，在键入密码的时候，密码是根本不会显示在屏幕上的——甚至不会以星号显示，因此要确保键入的密码准确无误，然后按 Enter 键。

使用 PuTTY 和 shell

如果你以前从来没有使用过 PuTTY 或者系统 shell，可以看看下面的详细介绍，涉及终端仿真器主窗口的一些行为。

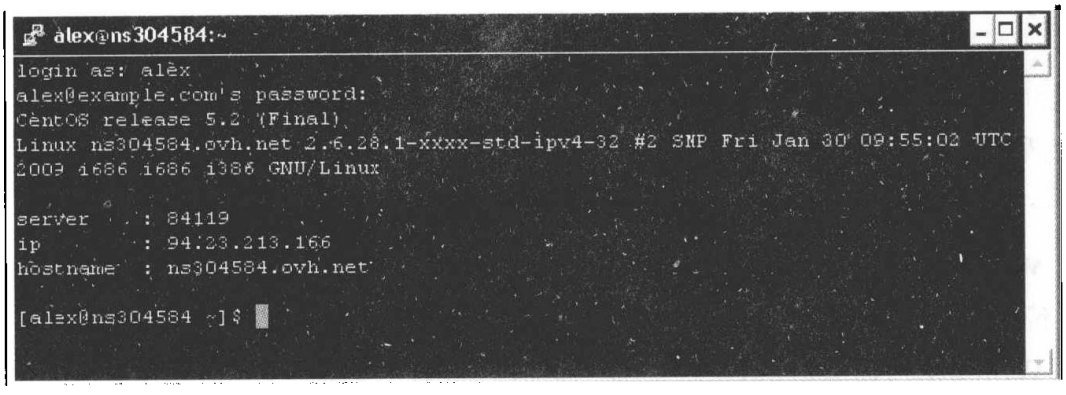
- 在终端窗口中，使用鼠标光标选择的文本在你释放鼠标左键的时候会自动复制到剪贴板。
- 在窗口的任何区域，只要单击鼠标右键就会将剪贴板中的文本粘贴到终端窗口。

alex@ns304584:~

```
login as: alex
alex@example.com's password:
CentOS release 5.2 (Final)
Linux ns304584.ovh.net 2.6.28.1-xxxx-std-ipv4-32 #2 SMP Fri Jan 30 09:55:02 UTC
2009 i686 i686 i386 GNU/Linux

server   : 84119
ip       : 94.23.213.166
hostname : ns304584.ovh.net

[alex@ns304584 ~]$
```

- 在 **Connection** 设置组，可开启 **TCP keepalives** 功能，以防止 TCP 超时而导致断开。
- 在 **Connection | Data** 设置组中，可以输入要连接的系统的帐户，然而由于安全的原因，PuTTY 并不允许你保存用户的密码。

一旦完成会话的配置，记得再次保存，然后再开始单击主窗口上的 **Open** 按钮建立连接。如果是第一次连接到服务器，则需要通过接受服务器端的指纹(fingerprint)来证实它的真实性。如果以后再连接到同一台服务器，则不会再看到确认的信息，除非服务器设置(例如主机名或者是端口号)改变，或者是安全性被损坏，你连接到了一个中间的服务器(有人在中间攻击)。最后，提示注册(除非你激活自动登录选项)和密码。请注意，在键入密码的时候，密码是根本不会显示在屏幕上的——甚至不会以星号显示，因此要确保键入的密码准确无误，然后按 Enter 键。



使用 PuTTY 和 shell

如果你以前从来没有使用过 PuTTY 或者系统 shell，可以看看下面的详细介绍，涉及终端仿真器主窗口的一些行为。

- 在终端窗口中，使用鼠标光标选择的文本在你释放鼠标左键的时候会自动复制到剪贴板。
- 在窗口的任何区域，只要单击鼠标右键就会将剪贴板中的文本粘贴到终端窗口。

- 使用快捷键 Ctrl+C 并不会把文本复制到剪贴板，而是中断一个正在执行的程序。如果你意外运行一个超长时间执行的命令[1]，则可以通过这个快捷方式来结束程序，从而再次出现 shell。
- 万一从服务器断开连接，则右击终端窗口的标题栏，打开一个菜单，然后选择“重新开始会话”。
- 在命令行键入文件名的时候，可以通过 Tab 键尝试自动补全文件名。在使用 Tab 补全的过程中，如果听到“哔哔”的声响，这可能有两个原因：不是你键入的文件名片段(文件名的一部分)在系统中不存在，就是在系统中找到多个文件。如果是属于后一种情况，那么快速敲 Tab 键两次就会看到一个与已输入部分的文件名匹配的一个文件列表。注意，这个功能并非一直有效，这依赖于服务器运行的系统。

基本的 shell 命令

连接到服务器并打开一个终端窗口是一回事，能够实际使用则是另一回事。如果从来没有在 Linux 下工作过，你可能会发现这部分内容对你特别有帮助，它会帮助你了解最基本的和有用的命令。这里介绍的很多命令都将用于后面的章节，但是你很快会意识到在这里有很多命令需要使用。

文件和目录管理

在 BASH (Bourne-Again SHell, GNU/Linux 发布时的默认 shell)和微软的 Windows 命令提示行有很多相似之处，主要的相似在于我们使用的工作目录概念。shell 提示输入一个命令，那么该命令会在当前的工作目录中执行。第一次登录 shell 帐户时，会进入你的 home 目录，这个文件夹一般包含你的个人文件，是一个私有的空间，如果没有特别指定访问权限，系统上的其他用户是不可进入该目录查看的。

① 所谓的超长时间是与你预计的时间相比较而言。

下表列出了一些最有用的基本命令，它们用于文件和目录的管理。

命令名称	描　述
pwd	显示当前所在的工作目录 [alex@example.com ~]$ pwd /home/alex
cd	改变目录 [alex@example.com ~]$ cd images [alex@example.com images]$ pwd /home/alex/images [alex@example.com images]$ cd /tmp [alex@example.com tmp]$ pwd /tmp 对于 cd，有一些有用的快捷方式： ● 键入 cd 或 cd ~ 总能够让你回到 home 目录，然而更通用的用法是“cd ~”，因为可以通过使用“cd ~/images”，进入用户主目录下的 image 目录 ● 键入“cd ..”会让你回到本目录的上一级目录，注意， cd 和“..”之间有空格 ● 键入“cd .”，没有任何效果。然而要注意，“.”代表当前目录，例如可以使用“cd ./images”，进入当前目录下的 images 目录
ls	列出当前目录或指定目录下的所有文件和目录 [alex@example.com ~]$ ls images photo2.jpg photo.jpg shopping.txt 试一下“**ls -l**”，显示与文件及目录相关的详细信息。选项“**-a**”显示隐藏文件和系统文件

续表

命令名称	描　述
mkdir	新建一个新目录 [alex@example.com ~]$ mkdir documents [alex@example.com ~]$ cd documents [alex@example.com documents]$ mkdir /tmp/alex [alex@example.com documents]$ cd /tmp/alex [alex@example.com alex]$ pwd /tmp/alex 一般的命令行应用程序没有输出任何文本的情况就是一个成功的操作。如果有错误发生，就显示一个信息
cp	文件复制 命令语法：cp [选项] 源文件 目标文件 [alex@example.com ~]$ cp photo2.jpg photo3.jpg
mv	移动或重命名文件 命令语法：mv [选项] 源文件 目标文件 文件重命名： [alex@example.com ~]$ mv photo3.jpg photo4.jpg 把文件移到另一个目录： [alex@example.com ~]$ mv photo4.jpg images/
rm	删除一个文件或者一个目录。使用-r 选项会使用递归功能： [alex@example.com ~]$ rm photo.jpg [alex@example.com ~]$ ls images photo2.jpg shopping.txt [alex@example.com ~]$ rm -r images/ [alex@example.com ~]$ ls photo2.jpg shopping.txt 使用这个命令要慎重，尤其是以管理员的身份(这里指的是系统管理员)登录时，被删除的文件是不能恢复的[1]。一个简单的“**rm -rf /**”命令足以“洗白”你的整个文件系统

① 可以通过专用的软件进行恢复，不过也不是十分有把握。

locate	在整个文件系统上找出指定的文件，该命令直接与 updatedb 关联： [alex@example.com ~]$ locate photo2.jpg /home/alex/photo2.jpg /home/jesse/holiday_photo2.jpg 注意： 命令 locate 完全依赖于索引，如果新建了一个文件，你是找不到它的，除非你使用下面的命令执行过数据库升级
updatedb	更新文件数据库。注意，该命令需要使用管理员权限，因此，一般通过 cron job，就是计划任务(相当于 Windows 操作系统中的任务管理)来实现一个定时更新： [alex@example.com ~]$ mkdir "Holidays in France" [alex@example.com ~]$ locate France No file found: a database update is required. 一旦以管理员的帐户进入，就可以执行该命令： [root@example.com ~]# updatedb [root@example.com ~]# locate France /home/alex/Holidays in France 这样便可找到新建文件
man	显示指定命令的使用帮助 [alex@example.com ~]$ man ls

alex@ns304584:~

```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information about the FILEs (the current directory by default).  Sort
       entries alphabetically if none of -cftuvSUX nor --sort.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .
:
```

命令名称	描　述

续表

命令名称	描　述
locate	在整个文件系统上找出指定的文件，该命令直接与 updatedb 关联： [alex@example.com ~]$ locate photo2.jpg /home/alex/photo2.jpg /home/jesse/holiday_photo2.jpg 注意： 命令 locate 完全依赖于索引，如果新建了一个文件，你是找不到它的，除非你使用下面的命令执行过数据库升级
updatedb	更新文件数据库。注意，该命令需要使用管理员权限，因此，一般通过 cron job，就是计划任务(相当于 Windows 操作系统中的任务管理)来实现一个定时更新： [alex@example.com ~]$ mkdir "Holidays in France" [alex@example.com ~]$ locate France No file found: a database update is required. 一旦以管理员的帐户进入，就可以执行该命令： [root@example.com ~]# updatedb [root@example.com ~]# locate France /home/alex/Holidays in France 这样便可找到新建文件
man	显示指定命令的使用帮助 [alex@example.com ~]$ man ls alex@ns304584:~ LS(1) User Commands LS(1) NAME ls - list directory contents SYNOPSIS ls [OPTION]... [FILE]... DESCRIPTION List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort. Mandatory arguments to long options are mandatory for short options too. -a, --all do not ignore entries starting with .

最后，可以使用 clear 命令来清除屏幕上的所有文本，然后重新开始。

用户和组管理

对于管理员来说，首要的困扰就是在他们的系统上有哪些用户，这些用户要访问哪些资源。在这方面，基于 Unix 的操作系统提供了一个详细的用户和组的管理机制。

超级用户帐户

每一个操作系统中都有一个超级用户帐户，经常需要执行管理员级别的任务，这个帐户通常叫 root，然而在一些操作系统上也叫其他名字(例如 admin 或者甚至是 toor)。超级用户能够访问系统中所有的文件和目录，有权读取、编辑和执行所有的文件，也可以改变文件的属性和访问权限。但是还是不推荐用机器的超级用户长期连接远程机器。实际上，一些操作系统(例如 Ubuntu)甚至都不允许你这么做。计算机安全的一个最基本的原理，即最小权限，你永远不允许做你分外的事情。换句话说，如果你只打算给一个使用你计算机的用户上网和使用 Open Office 编写文档的能力，那么为什么给他能够访问系统配置目录的权限呢？对用户授予比需要更多的权限只会导致系统的安全性和完整性被损坏。正是由于这个原因，所以强烈推荐你建立一个用户帐号，这样做不仅方便限制机器上的个人用户，也是为了确保应用程序要运行在一个具有明确界定的安全环境中。

用户帐户

在 Linux 文件系统中的配置目录[①]中保存着系统用户的列表：/etc/passwd，然而用户的密码并没有保存在该文件中。由于安全的原因，在大多数情况下，这些用户密码保存在一个独立文件/etc/shadow 中，不过该文件还是提供了每一位用户的一些信息。在文件 passwd 中，每一行对应一个用户，语法格式如下，而且必须遵循此格式：

```
name:password:ID:group ID:comment:home directory:login shell
```

① 一般是/etc 目录。

在实际的文件中，“password”部分被“x”所替代，它指出密码实际保存在文件/etc/shadow 中。

增加一个新的用户就是只是在文件/etc/passwd 中添加这么一行。然而，你可能会发现手动处理稍微有点麻烦，在这种情况下，你会很高兴地去学习一个能够自动操作的程序，而且发布的版本中也包括这样的工具，即 **useradd**。

对于这个命令，基本的语法是 useradd username，这将建立一个新的用户帐户，同时包括默认的设置(当然也可以定制)。默认的设置包括：一个 **home** 目录，位于/home；没有有效期；默认的组；登录的 Bash shell。如果添加的帐户是用来运行服务，例如 Nginx，则建议不要授予该用户帐户 shell 的访问权限，因此你要确定登录 shell 设置为 **nologin**(通常的设置 **/sbin/nologin**)，因此，应该用以下格式命令添加该用户：

```
useradd --shell /sbin/nologin nginx
```

也可以定制该用户 home 目录的位置，比如定位到 Nginx 的安装目录[①]：

```
useradd --shell /sbin/nologin --home-dir /usr/local/nginx nginx
```

最后这个 nginx 指的是要创建的这个用户帐户的名称。

在帐户建立完成后，如果想对该帐户修改一些参数，可以使用 **usermod** 命令。允许对帐户名重命名，修改帐户密码，移动 home 目录到其他位置，等等。最后，可能还想删除一个用户帐户，可以通过命令 **userdel** 来完成，格式很简单：**userdel username**，使用**-r** 选项还可以同时删除 home 目录。

① 实际上这样写更稳妥：**useradd -s /sbin/nologin -d /usr/local/nginx nginx**。

alex@ns304584:~

```
[alex@ns304584 ~]$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

在实际的文件中，“password”部分被“x”所替代，它指出密码实际保存在文件/etc/shadow 中。

```
alex@ns304584:~
[alex@ns304584 ~]$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

增加一个新的用户就是只是在文件/etc/passwd 中添加这么一行。然而，你可能会发现手动处理稍微有点麻烦，在这种情况下，你会很高兴地去学习一个能够自动操作的程序，而且发布的版本中也包括这样的工具，即 **useradd**。

对于这个命令，基本的语法是 useradd username，这将建立一个新的用户帐户，同时包括默认的设置(当然也可以定制)。默认的设置包括：一个 **home** 目录，位于/home；没有有效期；默认的组；登录的 Bash shell。如果添加的帐户是用来运行服务，例如 Nginx，则建议不要授予该用户帐户 shell 的访问权限，因此你要确定登录 shell 设置为 **nologin**(通常的设置 **/sbin/nologin**)，因此，应该用以下格式命令添加该用户：

```
useradd --shell /sbin/nologin nginx
```

也可以定制该用户 home 目录的位置，比如定位到 Nginx 的安装目录[①]：

```
useradd --shell /sbin/nologin --home-dir /usr/local/nginx nginx
```

最后这个 nginx 指的是要创建的这个用户帐户的名称。

在帐户建立完成后，如果想对该帐户修改一些参数，可以使用 **usermod** 命令。允许对帐户名重命名，修改帐户密码，移动 home 目录到其他位置，等等。最后，可能还想删除一个用户帐户，可以通过命令 **userdel** 来完成，格式很简单：**userdel username**，使用**-r** 选项还可以同时删除 home 目录。

① 实际上这样写更稳妥：**useradd -s /sbin/nologin -d /usr/local/nginx nginx**。

记住，所有这些命令都可以通过 man 命令获取详细信息，从而得到帮助，例如：man useradd。

组管理

除了用户帐户外，基于 Unix 的操作系统还提供了更高级的资源管理机制——用户组。其目的在于让同一个组的成员对某一文件或目录有共同的访问权限，每一个进入该组的用户都继承该组的权限。尽管用户可以属于其他组(secondary group)，但是一个用户帐户至少属于一个组——主要组(primary group)。

在实践中，用户组的列表保存在文件/etc/group 中，在文件中，每一行代表一个组，它的语法描述如下：

```
group name:password:group ID:user list
```

用户组密码很少使用，在该文件中实际存储的是一个“x”，它表示该组没有密码。在每一行的结尾，你会看到属于该组的用户列表。下图是一个生产环境服务器的用户组文件。

此外，如果想在系统中建立一个新的用户组，则有两个选择：(1)在文件/etc/group 中添加新的一行；(2)使用专用的 groupadd 命令来添加，它的语法很简单：groupadd groupname，它也有一些典型的参数，可以通过 man groupadd 了解。

类似于系统用户管理，也可以找到命令 groupmod 和 groupdel，分别用于修改和删除组。更重要的是，如何将一个用户添加到一个用户组？有两种方法：(1)手工编辑/etc/group 文件，将用户名添加到相应的组，在组的末尾添加上希望添加的用户；(2)可以使用下列命令来完成：

```
usermod --append --groups groupname username
```

alex@ns304584:~

```
[alex@ns304584 ~]$ cat /etc/group | grep ,
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
lp:x:7:daemon,lp
psaadm:x:2521:psaadm,sw-cp-server
psaserv:x:2522:apache,psaftp,psaadm
[alex@ns304584 ~]$
```

记住，所有这些命令都可以通过 man 命令获取详细信息，从而得到帮助，例如：man useradd。

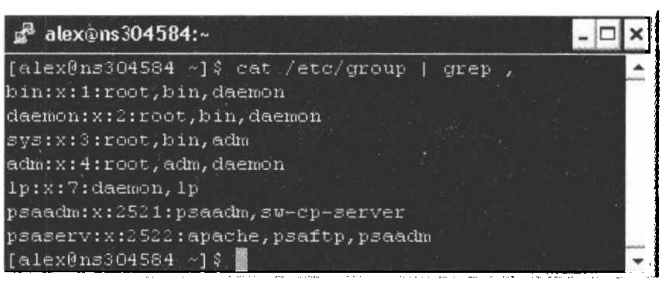
组管理

除了用户帐户外，基于 Unix 的操作系统还提供了更高级的资源管理机制——用户组。其目的在于让同一个组的成员对某一文件或目录有共同的访问权限，每一个进入该组的用户都继承该组的权限。尽管用户可以属于其他组(secondary group)，但是一个用户帐户至少属于一个组——主要组(primary group)。

在实践中，用户组的列表保存在文件/etc/group 中，在文件中，每一行代表一个组，它的语法描述如下：

```
group name:password:group ID:user list
```

用户组密码很少使用，在该文件中实际存储的是一个“x”，它表示该组没有密码。在每一行的结尾，你会看到属于该组的用户列表。下图是一个生产环境服务器的用户组文件。



此外，如果想在系统中建立一个新的用户组，则有两个选择：(1)在文件/etc/group 中添加新的一行；(2)使用专用的 groupadd 命令来添加，它的语法很简单：groupadd groupname，它也有一些典型的参数，可以通过 man groupadd 了解。

类似于系统用户管理，也可以找到命令 groupmod 和 groupdel，分别用于修改和删除组。更重要的是，如何将一个用户添加到一个用户组？有两种方法：(1)手工编辑/etc/group 文件，将用户名添加到相应的组，在组的末尾添加上希望添加的用户；(2)可以使用下列命令来完成：

```
usermod --append --groups groupname username
```

你可能会指定一个或多个组。如果将选项--append 移走，会得到以下结果：在命令中指定的组取而代之该用户现有的组。最后，可以通过命令 groups 来显示一个用户都属于哪些组。

程序和进程

在 shell 中运行一个程序不是简单地键入该文件的文件名。对于 Bash 处理执行二进制和脚本文件的方法，有些不易察觉的细节，你需要理解一下。

运行应用程序

想执行程序或者脚本的时候，需要面对三种不同的情况。

- **要执行的程序就在当前工作目录下。**

  解决方法：在要执行的文件前面加上“./”(句点和斜线)，这么做的原因是强迫 shell 在当前的工作目录中查找要执行的文件。

  示例如下：

  ```
  [alex@example.com ~]$ cd programs
  [alex@example.com programs]$ ./my-app
  ```

- **要执行的程序不在当前的工作目录下，但你知道该文件的路径。**

  解决方法：键入该文件的完整路径。

  示例如下：

  ```
  [alex@example.com ~]$ /home/alex/programs/my-app
  ```

- **要执行的程序位于环境变量 PATH 指定的值内。**

  解决方法：直接键入文件名，而不用指出其路径。

  示例如下：

  开始使用 nano 文本编辑器，通常能够在系统目录 /usr/bin 找到(/usr/bin 在 PATH 内)。

  ```
  [alex@example.com ~]$ nano
  ```

注意，在运行 shell 命令时，shell 的提示符是不会再出现的，除非该命令执行完毕。这种情况下会是个问题，如果有一个漫长的处理过程，在这个过程中你将无法做任何事情，因此，你可能会想在开始运行一个程序后，让它在后台运行，而不是在 shell 界面下完成执行，这也很容易，在执行的命令行后添加一个“&”字符即可。

示例如下：

```
[alex@example.com tmp]$ cp home.avi ~/movies/ &
[6] 2629
[alex@example.com tmp]$ [6] Done cp home.avi ~/movies/ &
```

一旦发送命令，该进程的 pid(进程标识符)就会显示在屏幕上，并且回到 shell 提示符。一旦执行完毕，除了用于开启该进程的原始命令之外，还会出现一条信息，指出该命令执行完毕，

系统服务

许多运行在后台的应用程序(通常指的是可以通过 **services 命令**来启动的程序)，都不是通过在命令后简单加上一个&字符，而是通过一个复杂的脚本来管理它们的启动和停止。这些脚本可以放置在各自的目录内，但最普遍的是在目录**/etc/init.d**内。

一些 Linux 发布版本中，例如 Red Hat, Fedora, CentOS 和 Mandriva，提供了一个 service 脚本，可以通过它来控制服务。通过使用 service name command 语法来实现，在这里，service 是脚本命令，name 指要操作的服务，command 是下表中的命令之一。如果在你所用的发布版本中没有提供 service 脚本，也可以使用类似这样的语法来执行：/etc/init.d/name command。注意，init.d 脚本不一定提供所有这些常用的命令。

start	启动指定的服务
stop	通过干净的方式(clean way)停止指定的服务
restart	重新启动指定的服务
reload	重新装载指定服务的配置文件
status	显示指定服务的状态

尝试一下 service --status-all，它会列出所有系统服务的当前状态。

命令名称	描述

alex@ns304584:~

```
[root@ns203544 ~]# service named restart
Stopping named :                                           [  OK  ]
Starting named :                                           [  OK  ]
[root@ns203544 ~]# /etc/init.d/mysqld status
mysqld (pid 12933) is running...
[root@ns203544 ~]# service httpd reload
Reloading httpd:                                           [  OK  ]
[root@ns203544 ~]#
```

示例如下：

```
[alex@example.com tmp]$ cp home.avi ~/movies/ &
[6] 2629
[alex@example.com tmp]$ [6] Done cp home.avi ~/movies/ &
```

一旦发送命令，该进程的 pid(进程标识符)就会显示在屏幕上，并且回到 shell 提示符。一旦执行完毕，除了用于开启该进程的原始命令之外，还会出现一条信息，指出该命令执行完毕，

系统服务

许多运行在后台的应用程序(通常指的是可以通过 **services 命令**来启动的程序)，都不是通过在命令后简单加上一个&字符，而是通过一个复杂的脚本来管理它们的启动和停止。这些脚本可以放置在各自的目录内，但最普遍的是在目录**/etc/init.d**内。

一些 Linux 发布版本中，例如 Red Hat, Fedora, CentOS 和 Mandriva，提供了一个 service 脚本，可以通过它来控制服务。通过使用 service name command 语法来实现，在这里，service 是脚本命令，name 指要操作的服务，command 是下表中的命令之一。如果在你所用的发布版本中没有提供 service 脚本，也可以使用类似这样的语法来执行：/etc/init.d/name command。注意，init.d 脚本不一定提供所有这些常用的命令。

命令名称	描　述
start	启动指定的服务
stop	通过干净的方式(clean way)停止指定的服务
restart	重新启动指定的服务
reload	重新装载指定服务的配置文件
status	显示指定服务的状态

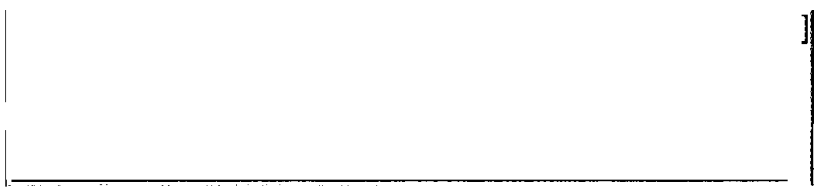
尝试一下 service --status-all，它会列出所有系统服务的当前状态。

进程管理

正如以前提到的，系统会为运行在计算机上的每一个进程提供一个数值，这个数值就是进程标识符(pid)。毫无疑问，pid 对于各种不同的环境都很重要，有些重要性可能还需要你去进一步发现。

查找 pid

首先，如何查找发现一个进程的 pid？尽管有许多方法可以得到它，但是使用最多的还是一个简单的工具——ps。它有很多选项(可以和管道机制组合)能够帮你找到进程的各种详细信息。



ps aux | grep sshd 命令可以分为三部分。

1. ps aux 命令列出当前系统中运行的所有进程。
2. | (pipe)是重定向管道。重定向命令的输出，将管道命令之前的命令输出重定向到管道命令后的命令中，作为该命令的输入。运行 ps aux 会产生一个很长的进程列表，但是因为这里只需要显示一个进程，因此使用了 grep 过滤命令。
3. grep sshd 从 ps aux 命令的输出中获取只包含指定关键字的行。换句话说，grep 负责的是过滤，保留了包括 sshd 的行。

alex@ns304584:~

```
[alex@ns304584 ~]$ ps aux | grep sshd
root     11757  0.0  0.1  11808  3052 ?        Ss   16:26   0:00 sshd: alex [priv]
alex     11820  0.0  0.0  11968  1744 ?        S    16:26   0:00 sshd: alex@pts/0
alex     12167  0.0  0.0   4156   660 pts/0    S+   16:27   0:00 grep sshd
root     32243  0.0  0.0   7036   992 ?        Ss   Oct29   0:00 /usr/sbin/sshd
[alex@ns304584 ~]$
```

[alex@example ~]$ ps aux

The result is not displayed on the standard output. Instead, it is "piped" to the next command.

```
apache   12604  8.5  0.6  28436 12660 ?        S    18:05   0:37 /var/usr/bin/php-cgi
nobody   13133  0.0  5.9 171684 120976 ?       Ssl  Sep30  23:10 memcached -d -m 128 -l 127.0.0.1
apache   13916 11.6  0.6  28504 12512 ?        S    18:10   0:20 /var/usr/bin/php-cgi
root     14005  0.0  0.2  34884  5204 ?        Ss   Oct17   1:11 /usr/bin/spamd --username=popuser
...
```

(pipe)

grep php

Grep filters the input: only lines containing "php" are retained

```
apache   12604  8.5  0.6  28436 12660 ?        S    18:05   0:37 /var/usr/bin/php-cgi
apache   13916 11.6  0.6  28504 12512 ?        S    18:10   0:20 /var/usr/bin/php-cgi
```

进程管理

正如以前提到的，系统会为运行在计算机上的每一个进程提供一个数值，这个数值就是进程标识符(pid)。毫无疑问，pid 对于各种不同的环境都很重要，有些重要性可能还需要你去进一步发现。

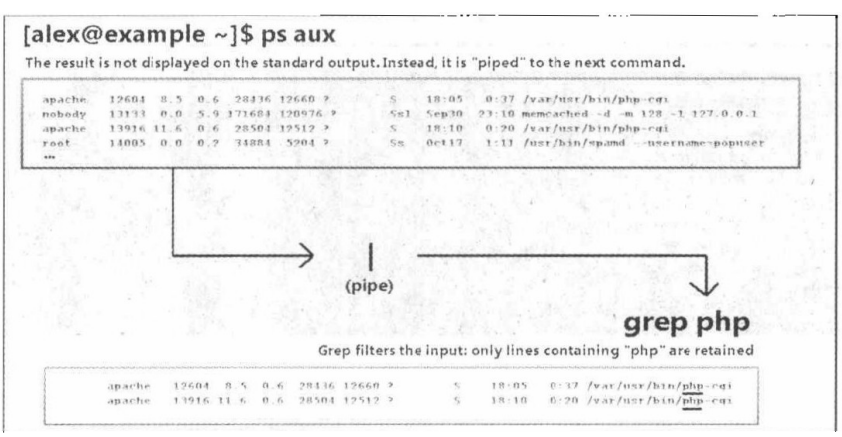
查找 pid

首先，如何查找发现一个进程的 pid？尽管有许多方法可以得到它，但是使用最多的还是一个简单的工具——ps。它有很多选项(可以和管道机制组合)能够帮你找到进程的各种详细信息。

```
alex@ns304584:~
[alex@ns304584 ~]$ ps aux | grep sshd
root     11757  0.0  0.1  11808  3052 ?        Ss   16:26   0:00 sshd: alex [priv]
alex     11820  0.0  0.0  11968  1744 ?        S    16:26   0:00 sshd: alex@pts/0
alex     12167  0.0  0.0   4156   660 pts/0    S+   16:27   0:00 grep sshd
root     32243  0.0  0.0   7036   992 ?        Ss   Oct29   0:00 /usr/sbin/sshd
[alex@ns304584 ~]$
```

ps aux | grep sshd 命令可以分为三部分。

1. ps aux 命令列出当前系统中运行的所有进程。
2. | (pipe)是重定向管道。重定向命令的输出，将管道命令之前的命令输出重定向到管道命令后的命令中，作为该命令的输入。运行 ps aux 会产生一个很长的进程列表，但是因为这里只需要显示一个进程，因此使用了 grep 过滤命令。
3. grep sshd 从 ps aux 命令的输出中获取只包含指定关键字的行。换句话说，grep 负责的是过滤，保留了包括 sshd 的行。



管理员最好的朋友——top

如果你的系统上运行着一个高流量的网站，那么另一个工具特别有用，它就是top，它列出了在当前系统下运行的所有进程，并且列出了进程的 pid，默认顺序以进程对 CUP 的使用情况排序[1]。这个显示每秒钟会自动刷新一次，直到你中断它的执行(例如可以通过执行 Ctrl+C 键中断)或者是通过按 Q 键退出应用程序。可以通过这个工具了解和追踪最消耗资源的进程。

大写字母部分提供了大量有用的统计数据，主要是当前系统上资源的使用情况，例如正常运行的时间、正在活动的用户、平均负载、内存和 CPU 的使用情况等。

杀进程

如果一个正在执行的命令发生错误并且不返回提示符，可以使用一种解决方法，即按快捷键 Ctrl+C 中断这个应用程序。通过 kill 命令，同样的操作能够应用于后台进程。有一点需要说明，不能在 kill 之后指定进程的名称，而是为它提供该程序的pid，这么做的原因显而易见，同一个程序可能会被执行多次，它们的进程名相同，但每次执行的 pid 却不同，所以一个程序名不一定总是对应于独一无二的进程：

```
[alex@example.com ~]$ kill 12075
```

① 如果想以其他指标排序，可以查看 man top，该命令有许多用法。

alex@ns304584:~

```
[alex@ns304584 ~]$ top
top - 16:31:32 up 125 days,  7:43,  5 users,  load average: 0.44, 0.55, 0.54
Tasks: 237 total,   2 running, 235 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.3%us,  1.2%sy,  0.0%ni, 95.4%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2048448k total,  1925172k used,   123276k free,   135600k buffers
Swap:  1044208k total,    79180k used,   965028k free,  1123808k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
12075 apache    20   0 27972  11m 5808 R   51  0.6   0:21.46 php-cgi
12144 apache    20   0 28104  12m 5804 S   22  0.6   0:29.15 php-cgi
13594 alex      20   0  2316  996  708 R    2  0.0   0:00.01 top
    1 root      20   0  2060  236  160 S    0  0.0   0:16.22 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.87 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   8:34.16 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:01.27 migration/1
    6 root      15  -5     0    0    0 S    0  0.0   8:55.97 ksoftirqd/1
    7 root      15  -5     0    0    0 S    0  0.0   0:04.22 events/0
    8 root      15  -5     0    0    0 S    0  0.0   0:17.37 events/1
```

管理员最好的朋友——top

如果你的系统上运行着一个高流量的网站，那么另一个工具特别有用，它就是 top，它列出了在当前系统下运行的所有进程，并且列出了进程的 pid，默认顺序以进程对 CUP 的使用情况排序[①]。这个显示每秒钟会自动刷新一次，直到你中断它的执行(例如可以通过执行 Ctrl+C 键中断)或者是通过按 Q 键退出应用程序。可以通过这个工具了解和追踪最消耗资源的进程。

```
alex@ns304584:~
[alex@ns304584 ~]$ top
top - 16:31:32 up 125 days,  7:43,  5 users,  load average: 0.44, 0.55, 0.54
Tasks: 237 total,   2 running, 235 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.3%us,  1.2%sy,  0.0%ni, 95.4%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2048448k total,  1925172k used,   123276k free,   135600k buffers
Swap:  1044208k total,    79180k used,   965028k free,  1123808k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
12075 apache    20   0 27972  11m 5808 R   51  0.6   0:21.46 php-cgi
12144 apache    20   0 28104  12m 5804 S   22  0.6   0:29.15 php-cgi
13594 alex      20   0  2316  996  708 R    2  0.0   0:00.01 top
    1 root      20   0  2060  236  160 S    0  0.0   0:16.22 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.87 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   8:34.16 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:01.27 migration/1
    6 root      15  -5     0    0    0 S    0  0.0   8:55.97 ksoftirqd/1
    7 root      15  -5     0    0    0 S    0  0.0   0:04.22 events/0
    8 root      15  -5     0    0    0 S    0  0.0   0:17.37 events/1
```

大写字母部分提供了大量有用的统计数据，主要是当前系统上资源的使用情况，例如正常运行的时间、正在活动的用户、平均负载、内存和 CPU 的使用情况等。

杀进程

如果一个正在执行的命令发生错误并且不返回提示符，可以使用一种解决方法，即按快捷键 Ctrl+C 中断这个应用程序。通过 kill 命令，同样的操作能够应用于后台进程。有一点需要说明，不能在 kill 之后指定进程的名称，而是为它提供该程序的 pid，这么做的原因显而易见，同一个程序可能会被执行多次，它们的进程名相同，但每次执行的 pid 却不同，所以一个程序名不一定总是对应于独一无二的进程：

```
[alex@example.com ~]$ kill 12075
```

① 如果想以其他指标排序，可以查看 man top，该命令有许多用法。

此外，如果执行该命令没有任何结果输出，也没有必要担心。实际上，如果输入的是一个无效的 pid，那么可能 kill 命令才会提示你。kill 命令是简单的发送一个信号给指定进程，这并不一定意味着该进程已成功停止。如果该程序被锁定，那么该信号不会响应进程的执行，程序将继续运行。不要急，有一种简单的处理方法能让你恢复信心，即选项“-9”，它能够强迫进程立即结束。

```
[alex@example.com ~]$ kill -9 12075
```

最后，正如你所想的，在某些时候你可能需要依次结束多个进程。例如，需要杀掉 Apache 产生的所有进程。如果是这样，我们可以使用一个稍微不同的命令——**killall**，它与 **kill** 命令稍有不同，**killall** 命令接受进程名作为参数，而不是 pid。

```
[alex@example.com ~]$ killall httpd
```

了解 Linux 文件系统

Linux 操作系统有自己组织文件的方法，也是一种非常特殊的组织方法，近似于历史悠久的文件系统等级标准 FHS(Filesystem Hierarchy Standard)。根据 FHS 官方的文档，这个标准的作用是：

- 软件预定(predict)安装文件和目录的位置；
- 用户预定(predict)安装文件和目录的位置。

尽管 FHS 的原始标准规范是在 1933 年公布的，但它仍然供现代发布的 Linux 使用，不过略有修改，使用的是修订版本。

目录结构

不像 Windows 操作系统一样——在 Windows 操作系统中，所有文件路径总是以一个字母开头(可以想象一下，如果系统中有多达 26 个驱动器的时候会是什么情况？)——基于 FHS 的文件系统有一个共同的父目录，该父目录叫根 (root)目录，也就是众所周知的“/”(斜线字符)。所有文件和目录(不管是设备、驱动器或分区，它们挂载于本地)都是根目录的子目录(children)，因此在本书中，所有的绝对路径总是以斜线开始的(见下表)。

让我们现在开始运行 cd/，要了解通过 FHS 定义的许多子目录，接着运行 ls。请注意，该目录结构纯粹是传统的目录结构，你可以在该目录下或者其他目录下放置你自己的文件和建立更多的目录。

路径名称	描　述
/	根目录：不要与/root 混淆，根目录下通常没有文件，尽管没有任何人阻止你这么做
/bin	二进制文件目录：系统中的可执行二进制文件或脚本文件通常都放在该目录，该目录对于系统上的所有用户可见。例如，一些常用的命令 ls、cp 或者 mv 等都可以在该目录下找到
/boot	boot 目录：存放系统启动时的关键文件
/dev	设备：该目录存放设备文件和特殊文件，更多的信息参见下一节
/etc	配置文件目录：系统中服务和应用程序的配置文件目录，你可能会经常浏览这个目录，例如，在需要编辑 Nginx 服务器的设置或添加虚拟主机的时候
/home	主目录：该目录包含本系统上除 root 用户外其他所有用户的主目录。例如，目录/home/alex 为系统用户 alex 的 home 目录
/lib	库文件目录：用于存放/bin 和 /sbin 目录下的二进制文件需要的共享库文件和内核模块
/media	可移动介质目录：该目录能使你轻松访问可移动介质的目录，挂载点用来挂载 CD-ROM、USB 等设备
/mnt	临时挂载文件系统：该目录适合管理员临时挂载一个文件系统
/opt	可选的软件安装包：理论上，这个目录应该安装一些应用程序文件和附加的安装包(在安装系统是没有默认安装)，但是在实际使用中，该目录基本不使用
/proc	内核和进程信息的虚拟文件系统：该目录提供了访问虚拟文件系统的桥梁，它包含各种状态信息和关于所有进程的详细信息
/root	root 用户的 home 目录：用户 root，也就是众所周知的超级用户(Superuser)，它不像普通用户一样存储在/home 目录下的 home 目录，取而代之的是/root 目录，即斜线“/”和 root，注意，这里的 root 不能和根(root)目录“/”混淆
/sbin	系统二进制文件目录：该目录中的工具供系统管理员使用，因此一般情况下只有 root 用户才可访问。例如，程序 ifconfig，halt，service 等其他许多的程序都能在这里找到
/srv	服务器数据：该目录用于存放来自于系统的服务产生的数据。就像其他目录一样，它只是占用一个目录而已，该目录很少使用
/tmp	临时文件命令：在程序执行时，不需要保护的文件应该放在这里。实际上，许多操作系统在重新启动后就会清除该目录中的内容

续表

路径名称	描　述
/usr	只读的用户数据：该目录是第二个层次分级目录，它提供了只读的共享数据。/usr 目录包含下列目录： ● **/usr/bin**　不重要的二进制命令和脚本文件，所有用户均可使用(例如 wget，gzip，firefox 等) ● **/usr/include**　该目录下是头文件，如来自 C 的库文件，有些程序在编译时会包含该目录下的这些文件 ● **/usr/lib**　由/usr/bin 和/usr/sbin 下的程序调用的库文件 ● **/usr/sbin**　不重要的系统二进制命令和脚本文件，对于所有的用户是可用的(例如，useradd 和 ntpdate 等) ● **/usr/share**　与系统结构体系无关的数据文件 ● **/usr/src**　内核源代码和安装应用程序的源代码 ● **/usr/X11R6**　与 X Window 系统 (v11 release 6)-相关的文件 ● **/usr/local**　一个第三层的目录结构，仅用于本地数据
/var	变化的文件：在运行应用程序或是服务中，这些文件会发生预期变化，例如，logfiles，cache，spool，等等。它有自己的层次结构： ● **/var/lib**　应用程序或更普遍的说法是操作系统依赖的变量状态信息。注意，MySql 数据库文件的变量信息通常存储在/var/lib/mysql ● **/var/lock**　锁文件(lock file)用于应用程序之间的同步资源访问 ● **/var/log**　由程序、服务或内核产生的日志文件 ● **/var/mail**　基于用户邮件的文件。在更多的系统中，/var/mail 也只是一个简单的快捷方式，实际文件位于/var/spool/mail 中 ● **/var/run**　运行中的可变数据。在系统重新启动后原有的数据将消失，该目录提供的信息是从系统被启动开始到现在的有关信息 ● **/var/spool**　这个目录放的文件是要处理的文件，例如邮件和打印作业 ● **/var/tmp**　一个占位符，用于存放系统重新启动时不应删除的临时文件

特殊文件和设备

你可能在目录结构中注意到，Linux 操作系统有一个预留目录(/dev)，用于存放“设备文件”。事实上，这个目录包含的元素称为节点，每一个节点在系统上代表为一个不同的设备，它们是实际的硬件设备或者是伪设备，无论哪种情况，作为文件系统的一部分将它们列出来，其目的是配合程序和服务的输入和输出，使它们更加便利——软件开发人员能够方便地访问这些设备，因为他们要读写文件。应该学习设备文件，很多时候需要使用它，或者说迟早会使用到它们。

设备类型

在/dev 目录内，可能会有大量有效的设备，不幸的是，它们中大多数设备都有一个不出名的名字，几乎不可能看出其用途，设备文件的命名采用的是 Linux 操作系统的协定。由于在系统中可能存在很多设备，因此我们将仅识别出最常见的一些。设备的名称由一个前缀组成，按照惯例，这个前缀用于定义设备的类型，并且如果同一类型的设备有多个出现在当前系统中，那么在设备之后会有一个数字或者是字母，以示区分。

下面列出最常见设备文件类型的常规前缀：

- **cdrom** CD 和 DVD-ROM 驱动器
- **fd** 软盘驱动器
- **hd** IDE 连接设备，例如硬盘驱动器和 CD-ROM
- **md** Metadisks 和 RAID 设备，例如硬盘
- **ram** RAM 磁盘
- **sd** SCSI 连接大容量存储设备
- **usb** USB 连接设备

alex@ns304584:/dev

```
[alex@ns304584 ~]$ cd /dev
[alex@ns304584 dev]$ ls
console             ptyc8  ptyrf  ptyx6   tty2   ttyc3  ttyra  ttywd
core                ptyc9  ptys0  ptyx7   tty20  ttyc4  ttyrb  ttywe
                    ptyca  ptys1  ptyx8   tty21  ttyc5  ttyrc  ttywf
cpu_dma_latency     ptycb  ptys2  ptyx9   tty22  ttyc6  ttyrd  ttyx0
dac960_gam          ptycc  ptys3  ptyxa   tty23  ttyc7  ttyre  ttyx1
                    ptycd  ptys4  ptyxb   tty24  ttyc8  ttyrf  ttyx2
dlm-control         ptyce  ptys5  ptyxc   tty25  ttyc9  ttys0  ttyx3
dlm-monitor         ptycf  ptys6  ptyxd   tty26  ttyca  ttyS0  ttyx4
dlm_plock           ptyd0  ptys7  ptyxe   tty27  ttycb  ttys1  ttyx5
fd                  ptyd1  ptys8  ptyxf   tty28  ttycc  ttyS1  ttyx6
full                ptyd2  ptys9  ptyy0   tty29  ttycd  ttys2  ttyx7
```

特殊文件和设备

你可能在目录结构中注意到，Linux 操作系统有一个预留目录(/dev)，用于存放“设备文件”。事实上，这个目录包含的元素称为节点，每一个节点在系统上代表为一个不同的设备，它们是实际的硬件设备或者是伪设备，无论哪种情况，作为文件系统的一部分将它们列出来，其目的是配合程序和服务的输入和输出，使它们更加便利——软件开发人员能够方便地访问这些设备，因为他们要读写文件。应该学习设备文件，很多时候需要使用它，或者说迟早会使用到它们。

设备类型

在/dev 目录内，可能会有大量有效的设备，不幸的是，它们中大多数设备都有一个不出名的名字，几乎不可能看出其用途，设备文件的命名采用的是 Linux 操作系统的协定。由于在系统中可能存在很多设备，因此我们将仅识别出最常见的一些。设备的名称由一个前缀组成，按照惯例，这个前缀用于定义设备的类型，并且如果同一类型的设备有多个出现在当前系统中，那么在设备之后会有一个数字或者是字母，以示区分。

```
alex@ns304584:/dev
[alex@ns304584 ~]$ cd /dev
[alex@ns304584 dev]$ ls
console          ptyc8  ptyrf  ptyx6   tty2   ttyc3  ttyra  ttywd
core             ptyc9  ptys0  ptyx7   tty20  ttyc4  ttyrb  ttywe
                 ptyca  ptys1  ptyx8   tty21  ttyc5  ttyrc  ttywf
cpu_dma_latency  ptycb  ptys2  ptyx9   tty22  ttyc6  ttyrd  ttyx0
dac960_gam       ptycc  ptys3  ptyxa   tty23  ttyc7  ttyre  ttyx1
                 ptycd  ptys4  ptyxb   tty24  ttyc8  ttyrf  ttyx2
dlm-control      ptyce  ptys5  ptyxc   tty25  ttyc9  ttys0  ttyx3
dlm-monitor      ptycf  ptys6  ptyxd   tty26  ttyca  ttyS0  ttyx4
dlm_plock        ptyd0  ptys7  ptyxe   tty27  ttycb  ttys1  ttyx5
fd               ptyd1  ptys8  ptyxf   tty28  ttycc  ttyS1  ttyx6
full             ptyd2  ptys9  ptyy0   tty29  ttycd  ttys2  ttyx7
```

下面列出最常见设备文件类型的常规前缀：

- **cdrom** CD 和 DVD-ROM 驱动器
- **fd** 软盘驱动器
- **hd** IDE 连接设备，例如硬盘驱动器和 CD-ROM
- **md** Metadisks 和 RAID 设备，例如硬盘
- **ram** RAM 磁盘
- **sd** SCSI 连接大容量存储设备
- **usb** USB 连接设备

伪设备

列在目录/dev 目录下的一些设备并不相当于实际的硬件设备，相反，它们是为了提供给管理员和开发者访问特殊资源的简单输入或是输出。出于这个原因，我们称它们为“伪设备”(Pseudo device)。下表列出最常用的伪设备，简单描述如下。

伪 设 备	描 述
/dev/null	**Null 设备** 这个伪设备经常被称为黑洞(black hole)，因此它的目的是忽略发送给它的一切数据。向它写数据时，它始终报告写操作是成功的；向它读取数据时，该设备总是返回没有数据。如果你不想要将一个程序输出重定向到任何地方，这是特别有用的。 换句话说，如果你想确保一个命令执行，但又不想把文字输到屏幕上，则可以像下面这样： `[alex@example.com ~]$ cat shopping.txt > /dev/null`
/dev/random /dev/urandom	**随机数发生器** 流设备产生随机数数据流。/dev/random 产生真实的随机数，而 /dev/urandom 提供伪随机数。它们产生的是二进制数据，来自 /dev/random 和 /dev/urandom 的数字不能够显示在终端控制台(它们看起来像是垃圾数据流)，这些设备通常由开发人员使用，希望收集到可信(reliable)的随机数

/dev/full	**Full 设备**
	该伪设备是一个流设备，向它写入时，会返回错误，因为它总被认为是满的；用户或程序读取它时，它返回一个无限空字符流。伪设备/dev/full 为管理员和程序员提供了一种操作，一旦被触发就会返回错误。
	`[alex@example.com ~]$ echo Hello! > /dev/full`
	`~bash: echo: write error: No space left on device`
	建议进一步参见书后的译者注 1

续表

伪 设 备	描 述
/dev/zero	**Zero 数据** 与/dev/null 极其相似。写完成后，zero 伪设备总是提供成功的返回代码。然而，在从这个设备上读取时，它输出一个无限空字符流。 有各种各样的例子可以证明读设备/dev/zero 有用，例如，作为程序的输入为它提供数据，将生成一个指定大小的文件，或者是为了格式化存储设备而写入

挂接存储驱动器

正如你可能在前面几节已经注意到，在/dev 目录中，有些可用的设备是存储设备，例如硬盘存储器、固态硬盘(SSD)、软驱或 CD-ROM。然而访问它们提供的内容不只是简单地使用 cd 命令进入驱动器访问，存储驱动器需要挂接到文件系统，换句话说，驱动器需要附加到一个固定的目录。

```
[alex@example.com ~]$ cd /dev/md1
~bash: cd: /dev/md1: is not a directory.
[alex@example.com ~]$ mount /dev/md1 /mnt/alexdrive
[alex@example.com ~]$ cd /mnt/alexdrive
[alex@example.com alexdrive]$ ls
Documents Music Photos Videos boot.ini
```

命令 mount 允许你将一个驱动器(第一个参数，/dev/md1)附加到系统中现有的目录(第二个参数)。一旦驱动器被挂载，你就可以使用文件系统的任何指令访问该驱动器。

 在现代的Linux发行版本中，CD-ROM和其他一些常见设备能够由系统自动挂接。

如果你想获取当前挂接系统中设备的挂接情况，使用 mount 命令来完成这个工作——它会告诉你系统中每一个被挂接的驱动器及使用中的文件系统，如下图所示。

如果想让一个驱动器在开机的时候能够自动挂接，或者简单设置一个目录作为一个驱动器的默认挂载点，则需要编辑/etc/fstab 文件，这需要你具有管理员的权限，该文件是一个简单的文本文件，可以使用文本编辑器(例如 nano)来打开。然而，该文件遵循一个特殊的语法，一些不知不觉的改变可能会招来很多足以毁灭系统的可能，**fstab** 更详细的语法可以在 tuxfiles.org 之类的网站找到。

最后，如果需要从一个正在使用的计算机移除一个设备(例如，移除 USB 存储器)，那么你首先需要使命令将其卸载，卸载设备的命令是：

```
[alex@example.com ~]$ umount /dev/usb1
```

注意，命令的第一个参数可以是驱动器的名称或者是挂载点，处理的结果相同。

文件和 inode

在提及 Unix 操作系统的时候，“文件系统”的概念普遍被误解。由于这些系统遵循 FHS，所以它们使用一个共同的目录层次结构重组了所有文件和设备，然而，存储设备可能有自己独立的磁盘文件系统，一个包含大量存储设备(硬盘驱动器，CD-ROM，等等)的磁盘文件系统被设计为组织文件。Windows 操作系统更喜欢使用 FAT, FAT32 或 NTFS，然而，在 Linux 操作系统下，默认且也是最受推荐的文件系统是 EXT3。EXT3 有许多特征，对于管理员来说，要想完全理解操作系统提供的 EXT3，必须掌握 FHS。

alex@ns304584:~

```
[alex@ns304584 ~]$ mount
/dev/md1 on / type ext3 (rw,errors=remount-ro)
/dev/proc on /proc type proc (rw)
/dev/sys on /sys type sysfs (rw)
/dev/devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/md2 on /var type ext3 (rw,usrquota,grpquota)
/dev/shm on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
tmpfs on /var/usr/local/psa/handlers/before-local type tmpfs (rw)
tmpfs on /var/usr/local/psa/handlers/before-queue type tmpfs (rw)
tmpfs on /var/usr/local/psa/handlers/before-remote type tmpfs (rw)
tmpfs on /var/usr/local/psa/handlers/info type tmpfs (rw)
tmpfs on /var/usr/local/psa/handlers/spool type tmpfs (rw,mode=0770,gid=31)
```

如果你想获取当前挂接系统中设备的挂接情况，使用 mount 命令来完成这个工作——它会告诉你系统中每一个被挂接的驱动器及使用中的文件系统，如下图所示。

```
alex@ns304584:~
[alex@ns304584 ~]$ mount
/dev/md1 on / type ext3 (rw,errors=remount-ro)
/dev/proc on /proc type proc (rw)
/dev/sys on /sys type sysfs (rw)
/dev/devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/md2 on /var type ext3 (rw,usrquota,grpquota)
/dev/shm on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
tmpfs on /var/usr/local/psa/handlers/before-local type tmpfs (rw)
tmpfs on /var/usr/local/psa/handlers/before-queue type tmpfs (rw)
tmpfs on /var/usr/local/psa/handlers/before-remote type tmpfs (rw)
tmpfs on /var/usr/local/psa/handlers/info type tmpfs (rw)
tmpfs on /var/usr/local/psa/handlers/spool type tmpfs (rw,mode=0770,gid=31)
```

如果想让一个驱动器在开机的时候能够自动挂接，或者简单设置一个目录作为一个驱动器的默认挂载点，则需要编辑/etc/fstab 文件，这需要你具有管理员的权限，该文件是一个简单的文本文件，可以使用文本编辑器(例如 nano)来打开。然而，该文件遵循一个特殊的语法，一些不知不觉的改变可能会招来很多足以毁灭系统的可能，**fstab** 更详细的语法可以在 tuxfiles.org 之类的网站找到。

最后，如果需要从一个正在使用的计算机移除一个设备(例如，移除 USB 存储器)，那么你首先需要使命令将其卸载，卸载设备的命令是：

```
[alex@example.com ~]$ umount /dev/usb1
```

注意，命令的第一个参数可以是驱动器的名称或者是挂载点，处理的结果相同。

文件和 inode

在提及 Unix 操作系统的时候，“文件系统”的概念普遍被误解。由于这些系统遵循 FHS，所以它们使用一个共同的目录层次结构重组了所有文件和设备，然而，存储设备可能有自己独立的磁盘文件系统，一个包含大量存储设备(硬盘驱动器，CD-ROM，等等)的磁盘文件系统被设计为组织文件。Windows 操作系统更喜欢使用 FAT, FAT32 或 NTFS，然而，在 Linux 操作系统下，默认且也是最受推荐的文件系统是 EXT3。EXT3 有许多特征，对于管理员来说，要想完全理解操作系统提供的 EXT3，必须掌握 FHS。

EXT3 文件系统

不像 Windows 系统中古老的 FAT32 文件系统只允许文件大小为 4 GB，EXT3 文件系统下单个文件的大小限制为 16T(依赖于块的大小)。此外，在驱动器上使用 EXT3 文件系统最大的存储空间为 32 TB，因此你应该无忧无虑地使用许多年，除非使用存储器的能力猛涨。EXT3 有趣的特点之一是，它设计的数据存储方式，能使文档碎片降到最低限度，同时又不会影响系统性能，因此，不需要对驱动器进行碎片整理。

文件名

EXT3 文件系统可以接受长度为 256 个字符的文件名，尽管扩展名通常显示文件内容的类型，例如 a.txt 文件表示一个文本文件，它应该包含有文本内容，a.mp3 文件用户表示一个音乐文件，等等，但是 EXT3 不需要文件的扩展名。有一个重要的事实需要注意，即在 EXT3 的文件系统中，文件名区分大小写——可能你已经发现，在同一个目录中，文件名“SHOPPPING.TXT”，“Shopping.txt”或“shopping.txt”分别表示不同的文件。

节点

Linux 的文件系统(例如 EXT3 文件系统)会为每一个文件存储大量的信息，而这种信息被分为逻辑信息和物理信息——存储在称为 **inode** 特定结构中的信息和实际的文件数据。

一些被包含在 inode 中的数据指出系统如何恢复驱动器上文件的内容，但这还不是全部——inode 还包含文件权限，用户和组的所有权，文件大小，访问和修改时间，等等。注意，它不包含实际的文件名。

在磁盘上的每一个节点都有一个唯一的标识符，这个标识符被称为节点号(inode number)或 i-number，可以在各种情况下使用。可以通过使用 ls -i 检索文件的节点数(见下图)。

alex@ns304584:~

```
[alex@ns304584 ~]$ ls
photo3.jpg  photo.jpg  shopping.txt
[alex@ns304584 ~]$ ls -i
1048590                    1048591 photo.jpg
1048592 photo2.jpg         1048593 shopping.txt
[alex@ns304584 ~]$
```

EXT3 文件系统

不像 Windows 系统中古老的 FAT32 文件系统只允许文件大小为 4 GB，EXT3 文件系统下单个文件的大小限制为 16T(依赖于块的大小)。此外，在驱动器上使用 EXT3 文件系统最大的存储空间为 32 TB，因此你应该无忧无虑地使用许多年，除非使用存储器的能力猛涨。EXT3 有趣的特点之一是，它设计的数据存储方式，能使文档碎片降到最低限度，同时又不会影响系统性能，因此，不需要对驱动器进行碎片整理。

文件名

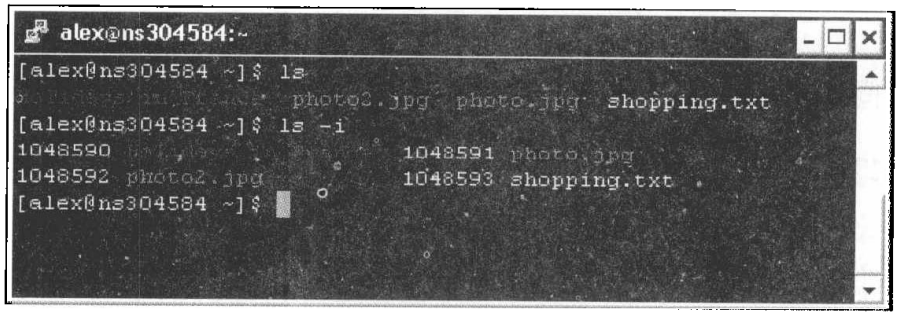
EXT3 文件系统可以接受长度为 256 个字符的文件名，尽管扩展名通常显示文件内容的类型，例如 a.txt 文件表示一个文本文件，它应该包含有文本内容，a.mp3 文件用户表示一个音乐文件，等等，但是 EXT3 不需要文件的扩展名。有一个重要的事实需要注意，即在 EXT3 的文件系统中，文件名区分大小写——可能你已经发现，在同一个目录中，文件名“SHOPPPING.TXT”，“Shopping.txt”或“shopping.txt”分别表示不同的文件。

节点

Linux 的文件系统(例如 EXT3 文件系统)会为每一个文件存储大量的信息，而这种信息被分为逻辑信息和物理信息——存储在称为 **inode** 特定结构中的信息和实际的文件数据。

一些被包含在 inode 中的数据指出系统如何恢复驱动器上文件的内容，但这还不是全部——inode 还包含文件权限，用户和组的所有权，文件大小，访问和修改时间，等等。注意，它不包含实际的文件名。

在磁盘上的每一个节点都有一个唯一的标识符，这个标识符被称为节点号(inode number)或 i-number，可以在各种情况下使用。可以通过使用 ls -i 检索文件的节点数(见下图)。



atime, ctime 和 mtime

元数据中包含一个 inode，你会发现有关文件的三个不同时间戳，分别为：atime, ctime, 和 mtime(见下表的描述)。

时间戳	描述
atime	**访问时间** 表示文件最后被访问的时间和日期。每一次应用程序或服务使用系统调用读取一个文件时，文件的访问时间都会更新
mtime	**修改时间** 表示文件被修改的时间和日期。文件的内容发生改变时，文件的修改日期将随之更新
ctime	**更改时间** 表示文件上次被修改的时间和日期，这个时间戳关系到文件属性(换句话说，文件的节点 inode 的改动)和文件数据

必须理解 **mtime** 和 **ctime** 之间的不同，修改时间(Modification time)仅关系到文件的数据，而更改时间(Change time)会同时跟踪文件属性和数据。下面有一些常见的例子说明了这三种机制。

文件访问时间(atime)

```
[alex@example.com ~]$ nano shopping.txt
```

文件被文本编辑器打开后，它的内容也就被访问。同时该文件的访问时间更新。

文件更改时间(ctime)

```
[alex@example.com ~]$ chmod 0755 script.sh
```

文件权限被更新(chmod 命令在后面的章节中讲述)，inode 中的信息也随之改变，这个命令执行的结果是文件的更改时间被更新。

```
[alex@example.com ~]$ echo "- a pair of socks" >> shopping.txt
```

文件修改时间(mtime)

文件数据被更新，因此，文件的修改时间和文件的改变时间都随之更新。

正如你可能已经注意到的，在 inode 中没有文件和目录建立的时间记录，因此如果想要查找文件是什么时候创建的，将不可能。至于为什么把这样一个重要的因素排除在外，目前还不清楚。如果想知道与文件相关的所有时间戳，可以使用 stat 命令：

```
[alex@example.com ~]$ stat shopping.txt
```

对于 SSD 固态驱动器用户很重要

事实证明，SSD 固态驱动器(Solid-State Drive)会使文件系统的访问时间功能下降，能够使驱动器性能下降显著，因为无论何时读取一个文件，都需要更新其 inode。因此，对于频繁进行写操作的情况，使用这种类型的存储设备显然是一个重大的问题。放心，对于这个已经存在的问题，也有一个简单的解决方法，即禁用文件的访问时间更新。这可以通过一个挂接选项来解决——noatime。将该选项在文件/etc/fstab 中指定(如果想永久不变地添加)。你可以简单用 noatime ssd 查找更多的文档，这归功于 Kevin Burton 的重大发现。

符号链接和硬链接

Linux 中的符号链接相当于 Windows 操作系统中的快捷方式，但仍然有些不同，需要解释一下，最重要的是，读写该文件的应用程序的访问实际上会影响到链接的目标(即被链接的文件)，而不是链接本身。然而，cp 或 rm 影响的是连接而不是目标。

建立链接是通过 ln -s 命令来完成的，这里有一个例子可以帮助你理解符号链接：

```
[alex@example.com ~]$ ln -s shoppinglist.txt link_to_list
[alex@example.com ~]$ ls
link_to_list photo.jpg photo2.jpg shoppinglist.txt
[alex@example.com ~]$ cat link_to_list
- toothpaste- a pair of socks
[alex@example.com ~]$ rm link_to_list
[alex@example.com ~]$ ls
photo.jpg photo2.jpg shoppinglist.txt
```

正如你看到的，可以通过符号连接读取被链接的文件。如果删除该链接，目标文件不会受到影响，复制操作也是一样的(是链接自身的复制，而不是目标文件)。除了与 Windows 快捷方式不同之外，另一个不同是它们可以使用相对路径链接文件。这使嵌入归档文件内的链接特别有用——用户可能提取文件到系统上的任何位置，使用绝对路径部署快捷方式将毫无意义。

最后， Windows 快捷方式有包括额外的元数据的能力，这允许用户选择一个图标，分配键盘的快捷方式，等等。然而，符号链接是简单的链接目标文件的路径及其本身，它们不提供同样的能力。

在 Windows 中，另一个类型的链接无效，那么就硬链接(hard link)，它们的功能稍微有点不同，硬链接表现为连接到文件的数据，两个或更多链接可以链接到存储器上的同一个数据，当这些链接中的任何一个被删除后，数据自身不受影响，其他的链接仍然指向该数据，如果把它们链接的目标文件也删除，则在最后一个链接被删除后，数据也将从存储器中移除。

下面举例说明，让我们为 shoppinglist.txt 文件建立一个硬链接，使用同样的命令 ln，但是没有开关选项-s：

```
[alex@example.com ~]$ ln shoppinglist.txt hard_link_to_list
```

如果决定删除文件 shoppinglist.txt，但是 hard_link_to_list 仍然留在这里，它指向的数据仍然有效，另外，新建的链接对于一些命令(例如 ls)，被作为一个实际的文件。如果运行 ls 命令计算所占的这个目录总文件大小，会发现链接文件的大小会被加上，如果 shoppinglist.txt 文件自身的大小占用磁盘空间为 5KB，那么被 ls 报告的总数为 10KB——5KB 为 shoppinglist.txt 文件自身的大小，5KB 为链接自身的大小。然而，一些工具，例如 du(用于显示磁盘使用量，后面会讲到)，能够更进一步报告实际占用的存储空间。

文件处理

下一个学习方向是 Linux shell，如在命令行界面熟练处理文件，通过这些简单的工具能够完成很多操作：编辑文件，压缩文件和目录，修改文件属性，等等。但是，现在让我们从一个更简单的话题出发——显示文件。

读取文件

首先，需要理解与我们一起工作的终端，换句话说，在这里不可能使用图形数据，只能够将文本显示在屏幕上，在这个意义上，就是处理指定文本文件了，不能够将二进制文件显示在屏幕上，例如，图表、视频等其他格式的二进制数据文件。在终端上显示文本文件最简单的方法是使用 cat 命令，例如，下图所示的例子：

但是 cat 命令还能够执行更复杂的操作(例如，将多个输入的文件连接为一个文件)，它格式简单，语法为 cat filename，文件 filename 的内容将显示在标准的输出设备上——换句话说，在这里指的就是终端显示屏，也就是你的显示器。

如果重新使用 grep 机制，前面在进程管理部分接触过 grep，那么对于下面的过滤，能够获得感兴趣的内容：

```
[alex@example.com ~]$ cat /etc/fstab | grep sys
/dev/sys /sys /sysfs defaults 0 0
```

正如你看到的，管道命令输出到命令 grep，在 grep 命令后设置指定过滤的字符串，任何没有包括指定字符串的行都不会显示。

为了以不同的方式显示文本，也可以将管道输出指向其他程序。例如，你的文件碰巧是一个大的文本文件，它可能不太合适当前的终端窗口，那么解决的方法为将管道转向 more 命令，例如下图所示的例子。

alex@ns304584:~

```
[alex@ns304584 ~]$ cat /etc/fstab
/dev/md1        /       ext3    errors=remount-ro       0       1
/dev/md2        /var    ext3    defaults,usrquota,grpquota      0       2
/dev/sda3       none    swap    defaults        0       0
/dev/sdb3       none    swap    defaults        0       0
/dev/devpts             /dev/pts                devpts  gid=5,mode=620  0 0
/dev/shm                /dev/shm                tmpfs   defaults        0 0
/dev/proc               /proc                   proc    defaults        0 0
/dev/sys                /sys                    sysfs   defaults        0 0
[alex@ns304584 ~]$
```

读取文件

首先，需要理解与我们一起工作的终端，换句话说，在这里不可能使用图形数据，只能够将文本显示在屏幕上，在这个意义上，就是处理指定文本文件了，不能够将二进制文件显示在屏幕上，例如，图表、视频等其他格式的二进制数据文件。在终端上显示文本文件最简单的方法是使用 cat 命令，例如，下图所示的例子：

```
alex@ns304584:~
[alex@ns304584 ~]$ cat /etc/fstab
/dev/md1        /       ext3    errors=remount-ro       0       1
/dev/md2        /var    ext3    defaults,usrquota,grpquota      0       2
/dev/sda3       none    swap    defaults        0       0
/dev/sdb3       none    swap    defaults        0       0
/dev/devpts             /dev/pts                devpts  gid=5,mode=620  0 0
/dev/shm                /dev/shm                tmpfs   defaults        0 0
/dev/proc               /proc                   proc    defaults        0 0
/dev/sys                /sys                    sysfs   defaults        0 0
[alex@ns304584 ~]$
```

但是 cat 命令还能够执行更复杂的操作(例如，将多个输入的文件连接为一个文件)，它格式简单，语法为 cat filename，文件 filename 的内容将显示在标准的输出设备上——换句话说，在这里指的就是终端显示屏，也就是你的显示器。

如果重新使用 grep 机制，前面在进程管理部分接触过 grep，那么对于下面的过滤，能够获得感兴趣的内容：

```
[alex@example.com ~]$ cat /etc/fstab | grep sys
/dev/sys /sys /sysfs defaults 0 0
```

正如你看到的，管道命令输出到命令 grep，在 grep 命令后设置指定过滤的字符串，任何没有包括指定字符串的行都不会显示。

为了以不同的方式显示文本，也可以将管道输出指向其他程序。例如，你的文件碰巧是一个大的文本文件，它可能不太合适当前的终端窗口，那么解决的方法为将管道转向 more 命令，例如下图所示的例子。

这样可以更好的控制文件流——它显示的行数和终端所能容纳的行数一样，直到你按下回车键(Return key)显示下一屏，按下 Q 或 Ctrl+C 返回 shell 提示符。

更妙的是 less 命令，允许你在文档流中上下滚动，它不像 cat 那样需要管道符，而 less 命令独自就可以完成：

```
[alex@example.com ~]$ less /etc/php.ini
```

编辑文件

如果你是一个长时间使用微软 Windows 或 Mac OS 操作系统的用户，那么学习这么一个高级命令行文本编辑器确实会感到很吃惊，以下提到的这几个命令行文本编辑器在大多数 Linux 发行版本中都有提供：vim, emacs, nano 等。这里有一个问题——即应该选择哪一个文本编辑器？既然你读到这里，那么你最好的选择应该是 nano，这已在前面的章节中提到。

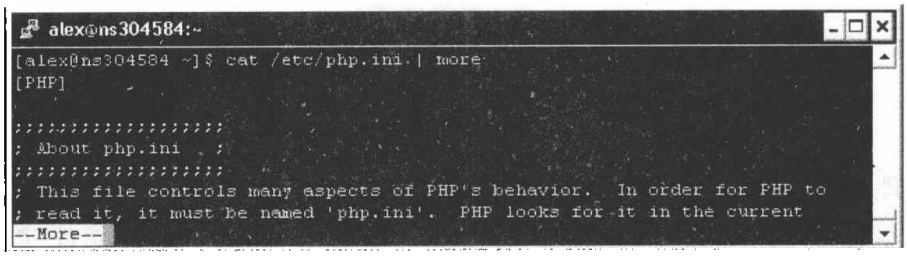
nano 是一个用户界面友好的文本编辑器，与它一起提供的是大量有趣的功能——高亮度语法、文本查找和替换及快捷键(keyboard shortcut)。不像它的竞争对手一样，通常需要一个长的学习过程，与其他文本编辑器有明显区别的是，nano 的界面很直观，如下图所示。

alex@ns304584:~

```
[alex@ns304584 ~]$ cat /etc/php.ini | more
[PHP]

;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;
; This file controls many aspects of PHP's behavior.  In order for PHP to
; read it, it must be named 'php.ini'.  PHP looks for it in the current
--More--
```

alex@ns304584:~

```
  GNU nano 2.0.9              File: /etc/php.ini

[PHP]

;;;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;;;
; This file controls many aspects of PHP's behavior.  In order for PHP to
; read it, it must be named 'php.ini'.  PHP looks for it in the current

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Tex ^T To Spell
```



这样可以更好的控制文件流——它显示的行数和终端所能容纳的行数一样，直到你按下回车键(Return key)显示下一屏，按下 Q 或 Ctrl+C 返回 shell 提示符。

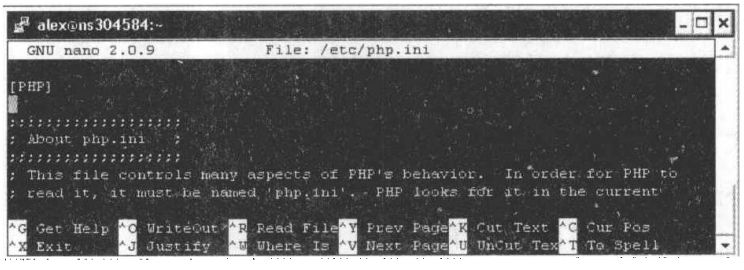
更妙的是 less 命令，允许你在文档流中上下滚动，它不像 cat 那样需要管道符，而 less 命令独自就可以完成：

```
[alex@example.com ~]$ less /etc/php.ini
```

编辑文件

如果你是一个长时间使用微软 Windows 或 Mac OS 操作系统的用户，那么学习这么一个高级命令行文本编辑器确实会感到很吃惊，以下提到的这几个命令行文本编辑器在大多数 Linux 发行版本中都有提供：vim, emacs, nano 等。这里有一个问题——即应该选择哪一个文本编辑器？既然你读到这里，那么你最好的选择应该是 nano，这已在前面的章节中提到。

nano 是一个用户界面友好的文本编辑器，与它一起提供的是大量有趣的功能——高亮度语法、文本查找和替换及快捷键(keyboard shortcut)。不像它的竞争对手一样，通常需要一个长的学习过程，与其他文本编辑器有明显区别的是，nano 的界面很直观，如下图所示。



由于在这个命令行控制的界面下没有鼠标光标，所以界面通过快捷键来控制，各种有效的操作都显示在底部的命令栏中。编辑完成后，保存(Ctrl+O)并退出(Ctrl+X)。注意，在底部有效的命令列表中，字符“^”表示控制(Control)组合键(^G 代表 Ctrl+G，^O 代表 Ctrl+O，等等)。

然而，向文件写入还有其他的方法，所使用的命令不需要任何界面，其中一个可行的方法就是重定向机制，它允许你为输入或输出流指定一个位置，通过 shell 命令进行交互，换句话说，在默认情况下，文字显示在屏幕上，但是你可以使用这种机制指定到其他的地方。对于重定向机制来说，最常见的用法是将一个命令的输出写入到一个文件。下面的例子说明了它的语法：

```
[alex@example.com ~]$ ls /etc > files_in_etc.txt
```

该命令执行后通常没有任何文本输出到屏幕，实际上文本已经被保存到你指定的文件中，符号“>”允许你把文本写到文件中，如果指定的文件已经存在，原始的内容会被删除和替换。在这个例子中，我们列出了/etc 目录下的文件，并且作为一个结果保存在一个文本文件中；如果使用符号“>>”，可将上面命令的输出添加到指定的文件(如果不存在，则新建一个)：

```
[alex@example.com ~]$ ls /etc/init.d >> files_in_etc.txt
```

/etc/init.d 目录中的文件都是文本文件，将该列表追加到一个文本文件。

最后，touch 命令用于更新一个文件的访问日期和修改日期，虽然它实际上并没有编辑文件的内容，但功能确实如此：

```
[alex@example.com ~]$ touch shopping.txt
```

压缩和归档

虽然 ZIP 和 RAR 压缩格式很流行并且广泛应用于互联网，它们都是专利技术软件，所以并非 Linux 世界的主流选择，其他格式，例如 Gzip 和 bzip2 是幸运的。当然，对于 Linux，ZIP 和 RAR 解决方案都有。但是你会发现在 Linux 系统下，多数项目下载的归档文件都是.tar.gz 或 .tar.bz2 文件。

正确读取文件名，这里有两个扩展名——tar 和 gz (或 bz2)，第一部分指出将文件组成到一起的方法，第二部分显示使用的压缩算法，**Tar**(是 **Tape archive**，磁带归档之意)，它是一个把多个文件连接为单个文件的工具，被压缩的单个文件称为 tarball，一旦这个 tarball 建立，就可以给定一个压缩选项，提供各种各样的压缩算法。该工具在大多数 Linux 发布版本中都有，然而在一些很小的系统中，可能必须用系统中的包管理器手动安装(后面有介绍)它。

使用 gzip 和 bz2 建立 tarball 的压缩包的语法各自如下：

```
tar czvf archive.tar.gz [file1 file2…]
tar cjvf archive.tar.bz2 [file1 file2…]
```

习惯上，Linux 用户不会把多个文件归档在一起，而是首先将文件收集到一个唯一的目录中，然后再归档该目录。因此，在用户解压归档文件时，只有一个条目出现在目录列表中。设想一下解压 ZIP 文件，将一个 ZIP 文件解压到 Windows 桌面，你愿意所有的文件单独出现在桌面上还是整洁优美地收集到一个单独的目录中？无论哪种方式，语法是一样的，就看要归档文件还是目录。

命令 tar 当然也能够执行相反的操作——解压文件，然而，由于在压缩时使用的算法不同，命令的参数也稍有不同：

```
tar xzvf archive.tar.gz
tar xjvf archive.tar.bz2
```

注意，tar.gz 文件也可创建为**.tgz**，tar.bz2 也可创建为**.tbz**。其他能够 tar 压缩格式有：LZMA (.tar.lzma)和 compress (.tar.z)，但是它们都已经废弃不用了。

如果偶然发现 RAR 或 ZIP，也可以解压缩这种文件，下载和安装 Linux 版的 unrar 或 unzip 即可。它们的语法很简单：

```
unrar x file.rar
unzip file.zip
```

系统管理工具

既然你打算安装并配置 Nginx，那么我们假设你就是服务器的管理员，在你的系统中建立这样一个重要的组成部分，需要充分理解 Linux 系统的管理概念和工具。

以超级管理员身份运行命令

正如我们在“超级帐户”部分讨论的，使用最少特权原则(the principle of least privilege)是很重要的。在这一方面，登录系统时，尽量少用 root 权限，如果经常使用 root 登录，会在许多方面令系统置于危险之中。首先，网络通信被截获，如果他们截获一个简单的用户帐户，那么潜在的电脑黑客所造成的破坏将大大减少；其次，每个人都会写错别字，如果意外键入“rm –rf / root/file.x”(注意 在–rf 和/之间有一个空格)，这个命令“rm –rf / root/file.x”就会变成为“rm –rf /”和“rm –rf root/file.x”，后果将难以想象！它会清除根目录“/”下所有的文件！因此在所有情况下，要让登录系统的用户具有适当的系统权限。

如果你以一个普通用户登录，那么如何执行管理员级别的任务或需要特定管理员权限的任务？这里有两个可能的回答，使用 **su** 和 **sudo**。

su 命令

su，是 substitute user 的缩写，它是一个命令，允许你使用指定的帐户来开始一个会话，如果没有指定帐户，则使用 root 帐户。需要输入你所用帐户的密码(除非你已经以 root 身份登录系统而想接替其他用户的帐户，这时将不再输入密码)：

```
[alex@example.com ~]$ su - root
Password :
[root@example.com ~]# nano /etc/fstab
```

从这时开始，你就已经作为 root 登录了。你可以运行命令和管理任务。完成后，可以键入 exit 命令退出当前会话。

```
[root@example.com ~]# exit
exit
[alex@example.com ~]$
```

你可能已经注意到，su 命令中用户名的前面使用了一个连字符“-”——它表示实际上是为用户建立一个 shell 会话，这会继承它所有的个人设置和环境变量，如果忽略连字符，则会停留在当前目录，并保护用户帐户最初登录所用的所有设置。

sudo 命令

尽管它的名字非常类似于 su，但 sudo 在一个完全不同的方式下工作。它只是使用指定的用户来执行一个命令，而不建立完整的会话，默认情况下是超级用户，语法如下：

```
sudo nano /etc/fstab
```

命令 su 和 sudo 在功能上的主要不同在于：使用命令 sudo 执行命令时，系统提示输入自己的帐户密码。我已经能听到你的尖叫——为什么我没有 root 的密码却能够获取 root 的权限？原来在/etc/sudoers 配置文件中，该文件指定了允许使用 sudo 的用户列表。更重要的是，指定了允许执行的命令，此外，所有用户的行为都被记录在日志文件中，包括失败的 sudo 尝试。

默认情况下，sudoers 文件中没有用户，因此，必须先以 root 用户登录，然后再将指定的用户添加到/etc/sudoers 文件，这个文件遵循一个严格的语法，因此，对于它可以使用一个专用的工具 **visudo**，其源于众所周知的 vi 文本编辑器，visudo 检测文件的语法然后保存文件，并确保该文件没有被同时编辑。

Visudo 是一种扩展功能，vi 则工作在两种模式下：命令模式和插入模式。在插入模式下可以直接编辑文档，按 Esc 键切换到命令行模式，输入命令来控制程序本身。在开始使用 visudo 时，按 I 键切换到插入模式，然后进行必要的修改，例如，在文件的末尾添加一个新的 sudo 用户：

```
alex ALL=(ALL) ALL
```

在文件 sudoers 中，这种定义授予 alex 用户所有的命令权限。编辑完毕后，按 Esc 键进入命令行模式，再键入如下命令来保存修改：

```
:w
```

再键入以下命令退出 visudo：

```
:q
```

如果希望退出但又不想保存修改，则键入以下命令：

```
:q!
```

关于 vi 或 visudo 的更多信息，可使用 man 命令来了解(或者如果你熟悉行话，那就是 RTFM！)。

系统检查和维护

现在，你完成了管理服务器的必要条件，可以执行实际的管理任务了。第一组任务涉及系统资源，在系统发生变化时——例如安装软件包(后面的章节会有讲解)——总是要检测系统的当前状态，查看是否有足够的磁盘空间和有效的内存空间。

磁盘剩余空间

命令 df 用于检查挂接驱动器的可用空间(见下图)：

选项-h 允许你以人类可读格式显示大小。应该经常检查磁盘有效空间：如果遭遇运行空间不足，应用程序可能出现任何情况(换句话说，就是难以理解的错误信息)。

磁盘使用量

如果注意到磁盘满了，但又不知道为什么，则可以求助于 du 命令，它特别有用。它允许你显示指定目录下每一个子目录占用的空间。

alex@ns304584:~

```
[alex@ns304584 ~]$ df -h
Filesystem            Size  Used  Avail.  Use%Mounted on
/dev/md1               9,9G  552M  8,8G    7% /
/dev/md2              449G   31G  395G    8% /var
/dev/shm             1001M     0 1001M    0% /dev/shm
tmpfs                1001M     0 1001M    0% /var/usr/local/psa/handlers/before-local
tmpfs                1001M     0 1001M    0% /var/usr/local/psa/handlers/before-queue
tmpfs                1001M     0 1001M    0% /var/usr/local/psa/handlers/before-remote
tmpfs                1001M     0 1001M    0% /var/usr/local/psa/handlers/info
tmpfs                1001M     0 1001M    0% /var/usr/local/psa/handlers/spool
[alex@ns304584 ~]$
```

alex@ns304584:/usr

```
[alex@ns304584 usr]$ sudo du -h --max-depth=1
Password :
288M    ./bin
4,0K    ./etc
4,0K    ./games
107M    ./include
1,6M    ./kerberos
935M    ./lib
42M     ./libexec
1,7G    ./local
40K     ./man
37M     ./sbin
1,6G    ./share
56K     ./src
8,0K    ./X11R6
4,6G    .
[alex@ns304584 usr]$
```

系统检查和维护

现在，你完成了管理服务器的必要条件，可以执行实际的管理任务了。第一组任务涉及系统资源，在系统发生变化时——例如安装软件包(后面的章节会有讲解)——总是要检测系统的当前状态，查看是否有足够的磁盘空间和有效的内存空间。

磁盘剩余空间

命令 df 用于检查挂接驱动器的可用空间(见下图)：

```
alex@ns304584:~
[alex@ns304584 ~]$ df -h
Filesystem            Size  Used Avail. Use%Mounted on
/dev/md1              9,9G  552M  8,8G   7% /
/dev/md2              449G   31G  395G   8% /var
/dev/shm             1001M     0 1001M   0% /dev/shm
tmpfs                1001M     0 1001M   0% /var/usr/local/psa/handlers/before-local
tmpfs                1001M     0 1001M   0% /var/usr/local/psa/handlers/before-queue
tmpfs                1001M     0 1001M   0% /var/usr/local/psa/handlers/before-remote
tmpfs                1001M     0 1001M   0% /var/usr/local/psa/handlers/info
tmpfs                1001M     0 1001M   0% /var/usr/local/psa/handlers/spool
[alex@ns304584 ~]$
```

选项-h 允许你以人类可读格式显示大小。应该经常检查磁盘有效空间：如果遭遇运行空间不足，应用程序可能出现任何情况(换句话说，就是难以理解的错误信息)。

磁盘使用量

如果注意到磁盘满了，但又不知道为什么，则可以求助于 du 命令，它特别有用。它允许你显示指定目录下每一个子目录占用的空间。

```
alex@ns304584:/usr
[alex@ns304584 usr]$ sudo du -h --max-depth=1
Password :
288M    ./bin
4,0K    ./etc
4,0K    ./games
107M    ./include
1,6M    ./kerberos
935M    ./lib
42M     ./libexec
1,7G    ./local
40K     ./man
37M     ./sbin
1,6G    ./share
56K     ./src
8,0K    ./X11R6
4,6G    .
[alex@ns304584 usr]$
```

这里再次使用-h，指定以人类可读的格式显示大小。如果不使用选项--max-depth，du 会从当前目录开始递归地浏览系统中的所有文件(换句话说，就是遍历指定目录下的所有文件)。现在，便可以轻松跟踪占用系统太多存储空间的目录了。

空闲内存

命令 free 显示当前系统内存的使用情况。它不但显示物理内存还显示 swap 的状态，也包括系统缓冲状态。可以使用-m 开关参数以兆字节显示或-k 以千字节显示。

软件包

基本的命令行用法？有！用户和组的管理？有！系统是否有足够的内存和磁盘存储空间？有！看起来你已准备安装新的软件套件和组件。基本上有三种方式，我们将从最简单到最复杂，逐一研究它们。

软件包管理

包管理器是一个工具，方便你对系统上的软件包进行管理，让你下载并安装 (软件包)、更新、卸载它们，等等。

在 Linux 世界中，有很多种不同的安装包系统，这往往与特定的发布有关——RPM 用于基于 Red Hat 系统的发布，APT 用于基于类 Debian 的系统发布，简单的 TGZ 包用于 Slackware，等等。在这里，我们只介绍前两个(即 RPM 和 APT)，它们也是我们最常用的两个。

对于使用 RPM 的系统，yum 是使用最多的包管理工具；对于 APT，则是与发布版本一起提供的 apt-get 工具。尽管它们的语法稍微不同，但是这两个程序的功能基本上相同——指定一个包的名称，它们将在线下载该软件并且自动安装它。

下面的例子中，显示了如何使用 yum 在计算机上安装 PHP：

```
[root@example.com ~]# yum install php
```

使用 apt-get：

```
[root@example.com ~]# apt-get install php
```

所有需要的组成部分，例如库文件或其他软件都将首先下载和安装，然后再处理请求的软件包。不需要你做任何别的事情，确认即可。同样，也可以使用这两个工具的其中一个来进行升级和移除操作。

下载和手动安装软件包

请注意，目前只有有限数量的软件包可以通过这些包管理器来进行管理，这依赖于它们的仓库(repositories)“存放”的数据包种类。该仓库来自 Linux 发行，其使用规范通常比较严格，软件开发人员不能始终用它们来发布自己的软件。因此，有许多应用程序无法在默认的仓库中找到(但可以使用自定义仓库)，这意味着你不能够使用包管理器来安装它们。

面对这种情况，有两种选择：从网上找一个安装包或者从源代码编译安装，下面详细介绍。第一个方法通常是访问包含你要安装软件的官方网站，然后找到发布的 RPM 安装包(或者是用于 Debian 系统的 DEB 包)，再下载安装。

使用 wget 下载工具下载，然后再用 **rpm -ivh** 命令安装该包：

```
[alex@example.com ~]$ wget ftp://example2.com/mysqlclient.rpm
(Download successful)
[alex@example.com ~]$ sudo rpm -ivh mysqlclient.rpm
```

对于 DEB 安装包，用 **dpkg -i** 命令安装：

```
[alex@example.com ~]$ wget ftp://example2.com/mysqlclient.deb
(Download successful)
[alex@example.com ~]$ sudo dpkg -i mysqlclient.deb
```

注意，这种方法不会处理依赖性，由于需要的库文件无法在系统中找到，所以应用程序可能无法正确安装。在这种情况下，只能自己安装它们(库文件)。

从源代码安装

最后一种方法，这种方法始终有效，即不管 Linux 的发行版本，下载应用程序的源代码并且编译它。这种方法有它自己的优势——通常可以配置大量的选项，如果你自己是个开发者，甚至还可以编辑它的源代码，从另一方面讲，它需要安装许多开发包(编译器和库等)，在编译中可能会由于一些不太重要但又模糊不清的原因导致编译失败——丢失组件，没有需要的库的版本，等等。

一般做法是下载一个包含源代码的.tar.gz 归档文件，然后再解压该文件，进入解压后的目录，再执行三个命令分别为 configure，make 和 make install。在下面的例子中，我们下载一个最新版本的 nano，并安装它：

```
[alex@example.com ~]$ wget http://www.nano-
editor.org/dist/v2.0/nano-2.0.9.tar.gz
(Download successful)
[alex@example.com ~]$ tar zxvf nano-2.0.9.tar.gz
(Extraction successful)
[alex@example.com ~]$ cd nano-2.0.9
[alex@example.com nano-2.0.9]$ ./configure
(Configuration complete)
[alex@example.com nano-2.0.9]$ make
(Build successful)
[alex@example.com nano-2.0.9]$ sudo make install
(Install successful)
```

视软件安装的过程而定，输出的二进制可能被复制到/usr/bin 目录(或者是 PATH 环境变量中找到的其他目录)，但有时必须亲自去找。

有些应用程序需要更具体的编译命令和程序，这些描述通常在一个 readme 文件中说明。在编译一个应用程序之前，不应该忽视该文件，应该好好读读它。

文件和权限

基于 Unix 的操作系统使用一个复杂的权限机制来管理访问文件和目录。在这类系统中，目录实际上也被视为特殊文件，涉及权限的时候，它们的工作方式是一样的。

理解文件权限

对于文件，有三种类型的访问：读文件、写文件和执行文件。每类访问都被定义为用户、用户组和其他用户组。权限的具体情况可以使用命令“ls –l”查看。

```
[alex@example.com photos]$ ls -l
total 2
drwxrwxrwx 2 alex alex 4096 oct 31 11:35 Holidays in France
-rw-rw-r-- 1 alex alex 8 oct 31 09:21 photo2.jpg
```

第一列提供一个文件权限的特征表示，它包含如下字符：

- 第 1 个字符：文件类型(-: 文件, d: 命令, l: 链接; 或者是其他类型)；
- 第 2 个到第 4 个字符：**R**-读, **w**-写, **x**-执行 针对属主；
- 第 5 个到第 7 个字符：**R**-读, **w**-写, **x**-执行 针对所属组；
- 第 8 个到第 10 个字符：**R**-读, **w**-写, **x**-执行 针对其他用户。

目录权限

目录具有特定的属性：粘着位(sticky bit)和设置组位 ID(set group ID)，SGID。第 1 个位能够确保放在该目录中的文件只能由它们的用户(和 root 用户)删除。第 2 个位用于给目录添加 sgid 权限，使该目录下创建文件或文件的所属组继承该目录的所属组。

目录的权限不同于一般的文件权限：

- *x* 位指定是否可以进入该目录(例如使用 cd)；
- *r* 允许目录中的内容被列出(例如使用 ls)；
- *w* 位指定是否允许在该目录中新建文件(或者删除已有的文件)。

八进制表示法

你肯定已经在其他的地方见过它：它的用法告诉你改变一个目录的权限为 0755，或者甚至是 777，给定的数字实际上是八进制，以八进制的方式来表示文件和目录的权限，这种格式由 3 或者 4 个数字从 0 到 7，这里的 0 表示没有权限，7 表示所有的权限。

- 第 1 个数字位是一个可选和指示属性(例如粘贴位)，经常不会特别指定或设置为 0；
- 第 2 个数字位表示文件属主的许可权限；
- 第 3 个数字位表示文件属组的许可权限；
- 第 4 个数字位表示其他用户的许可权限。

从 0 到 7 数字值的计算方法是：每一个属性都有一个权重，所有的权重加在一起构成一个总值。权重：0 表示没有任何属性；1 表示“r”，2 表示“w”，4 表示“x”。因此，每个属性变化有自己的八进制表示，如下表所示。

权限(r, x, w)			权　重	八进制表示
-	-	-	0 + 0 + 0	0
r	-	-	1 + 0 + 0	1
-	w	-	0 + 2 + 0	2
r	w	-	1 + 2 + 0	3
-	-	x	0 + 0 + 4	4
r	-	x	1 + 0 + 4	5
-	w	x	0 + 2 + 4	6
r	w	x	1 + 2 + 4	7

要让每一个人(文件属主、属组及其他用户)有全部的权限，则设置为 rwxrwxrwx，八进制表示为：777。

改变权限

除了万能的超级用户之外，其他的用户只能改变属于自己文件的权限，修改的工具是使用众所周知的 chmod 工具，它有两个主要语法变种：可以指定一个完整的八进制值权限来重置文件，也可以修改特定属性。

使用一个八进制值：

```
[alex@example.com ~]$ chmod 777 photo2.jpg
```

第一个参数是一个八进制值，第二个是文件或目录的名字。
第二个语法比较复杂：

```
chmod who+/-what filename
```

第一个参数(who， +/- 或 what)由三个元素组成：

- who 是一个"u" (user/owner)， "g" (group)， "o" (others) 和 "a" (all)的组合，如果忽略这部分，新的属性就会应用到所有的用户(all)；
- +/- 如果想授予那些权限，就使用“+”；如果想剥夺那些权限，就用“-”；
- what 是一个"r" (read)，"w" (write)和 "x" (execute)的组合。

下面是使用这种语法时可能使用到的一些例子：

```
chmod +x script.sh:
```

给予该脚本文件执行的权限(所有的用户均有执行的权限)；

```
chmod go-rwx photo.jpg:
```

除了属主外，其他人没有访问权限；

```
chmod a-w shopping.txt:
```

没有人能够编辑该文件，甚至属主也不可以。
注意，使用开关参数-R 可以递归修改一个目录的权限：

```
chmod -R g+rx photos:
```

该目录 photos 能够被属组内的任何用户访问，目录内的所有文件都可以查看。

改变用户和组的所有权

命令 chown 和 chgrp 分别允许你改变文件的属主和属组。第一个命令 chown 只能由超级用户执行，这显而易见是由于安全的原因。只要是该文件的属主，任何用户都可以改变自己文件的属组。
命令 chown 的语法如下：

```
chown user filename
```

在这里，user 是这个文件 filename 的属主。
关于 chgrp:

```
chgrp group filename
```

同前面一样，group 是新属组的组名，指定的文件属于新的组。
还有一种使用方法，chown 支持下列语法：

```
chown user:group filename
```

类似于 chmod，这两个命令接受参数-R，允许你递归添加改变。以下是一些可能用到的情况：

```
chown alex photo.jpg
```

作为 root 用户执行，文件 photo.jpg 的新属主是 alex；

```
chown -R root photos
```

作为 root 用户执行，photos 目录及其目录下的所有文件和子目录都将属于 root 用户；

```
chown alex:students shopping.txt
```

同时改变文件的属主和属组，属主为 alex，属组为 students；

```
chgrp guests shopping.txt
```

将文件 shopping.txt 的属组改为 guests；

```
chgrp -R applications /etc/apps
```

/etc/apps 目录及其目录下的所有文件和子目录都将属于 applications 用户。

小结

最后一部分是权限，它也标志着本章的结束，概述了一个网管应该使用的命令和定期执行的任务。使用 shell 命令主要是记忆相关的命令名称和参数，习惯使用它之后，会变得更高效，不久以后，当你再返回使用 Windows 时，有时甚至会发现自己打开一个命令行终端来执行简单的任务！不管怎么样，有了所有的组成部分，就需要开始下一步——下载和安装 Nginx Web 服务器应用程序了。在第 2 章的末尾，你将有一个能够工作、并且能够载入服务器上默认网页的 Web 服务器。

第 2 章

下载和安装 Nginx

在这一章，我们将继续采取必要的步骤，向着建立一个有用的 Nginx 前进，这一刻是保证 Web 服务器能够顺利运作的关键时刻——在安装 Nginx Web 服务器时有一些必要的库和工具，这些库和工具的选择取决于编译 Nginx 时选择的参数决定，但需要你在将 Nginx 的代码编译为二进制执行命令之选择安装这些必要的组件，并进行其他系统相关的配置。

本章主题

- 下载和编译安装 Nginx 之前的必要程序
- 下载一个合适的 Nginx 源代码版本
- 配置 Nginx 编译时的选项
- 通过 init 脚本来控制应用程序
- 配置系统启动时自动启动 Nginx

准备先决条件

正如你想到的，我们下载该程序的一个源代码包并且手动编译，而不是使用包管理工具，例如 Yum，Aptitude 或 Yast 来安装。这么做有两个原因。首先，在 Linux 发布版本中，该包可能无效，实际上，很少有提供下载和自动安装的仓库 (repositories)，即便有，大部分包含的也是过期的版本。其次，更重要的是，有一个不得不提的事实，我们需要在编译时对多种重要的选项进行配置，也正是基于这种情况，才有了不得不手动编译安装的结果，因此也就致使你需要在系统上安装这些工具和库文件，在 Nginx 编译时根据需要进行处理。

依赖于编译时的模块选择，可能需要不同的安装前提条件。在这里，我们将指导你安装最常见的工具和库，例如 GCC、PCRE、zlib 和 OpenSSL。

GCC——GNU 编译器集合

Nginx 是一个由 C 语言编写的程序，因此首先需要在系统上安装一个编译工具，例如 GNU 的 GCC，GCC 通常由大多数 Linux 的发行版本安装，但如果因为某种原因没有安装，那么这一步必须进行。

 GCC 是一个开源编译器集合，是用于处理各种各样的语言：C、C++、Java、Ada、FORTRAN，等等。在 Linux 世界中，它是最通用的编译器。它支持大量的处理器，例如 x86、AMD64、PowerPC、ARM、MIPS 等。

首先确定系统上是否已经安装 GCC，如下所示：

```
[alex@example.com ~]$ gcc
```

如果有下面的输出，则表明 GCC 已经正确安装在系统上，你可以跳过这一步，进行下面的步骤：

```
gcc: no input files
```

如果收到下面的消息，则必须安装和编译 GCC：

```
~bash: gcc: command not found
```

GCC 可以使用默认包管理器的仓库(repositories)来安装，包管理器的选择依赖于你使用的 Linux 发布版本，包管理器有不同的实现：yum 是基于 Red Hat 的发布版本；apt 用于 Debian 和 Ubuntu；yast 用于 SuSE Linux；等等。这里是典型处理方式，使用 yum 或 apt 来下载和安装 GCC：

```
[root@example.com ~]# yum install gcc
```

如果使用 apt-get：

```
[root@example.com ~]# apt-get install gcc
```

如果使用其他安装包管理器，那么语法可能有所不同，你可能需要使用 man 工具来查找相关文档。无论哪种方式，你的安装包管理程序能够正确下载和安装 GCC，之后自动解决依赖关系。

PCRE 库

在 Nginx 编译需要 PCRE(Perl Compatible Regular Expression)，因为 Nginx 的 Rewrite 模块和 HTTP 核心模块会使用到 PCRE 正则表达式语法，在后面，你将发现这一点。需要安装两个安装包 pcre 和 pcre-devel。第一个安装包提供编译版本的库，而第二个提供开发阶段的头文件和编译项目的源代码，这正是我们需要的理由。可以使用下面示例中的命令来安装这两种软件包。

使用 yum 来安装：

```
[root@example.com ~]# yum install pcre pcre-devel
```

或者安装与 PCRE 相关的所有的安装包：

```
[root@example.com ~]# yum install pcre*
```

如果用 apt-get：

```
[root@example.com ~]# apt-get install libpcre3 libpcre3-dev
```

如果这些安装包已经安装在系统上，你会收到一条信息 Nothing to do(见下图)，换句话说，安装包管理器没有安装或升级任何部分。

alex@ns304584:~

```
[root@ns304584 ~]# yum install pcre pcre-devel
Loading "fastestmirror" plugin
Loading mirror speeds from cached hostfile
 * rpmfusion-free-updates: download1.rpmfusion.org
 * adobe-linux-i386: linuxdownload.adobe.com
 * updates: mirror.ovh.net
 * base: mirror.ovh.net
 * addons: mirror.ovh.net
Excluding Packages in global exclude list
Finished
Setting up Install Process
Parsing package install arguments
Package pcre - 6.6-2.el5_1.7.i386 is already installed.
Package pcre-devel - 6.6-2.el5_1.7.i386 is already installed.
Nothing to do
[root@ns304584 ~]#
```

PCRE 库

在 Nginx 编译需要 PCRE(Perl Compatible Regular Expression)，因为 Nginx 的 Rewrite 模块和 HTTP 核心模块会使用到 PCRE 正则表达式语法，在后面，你将发现这一点。需要安装两个安装包 pcre 和 pcre-devel。第一个安装包提供编译版本的库，而第二个提供开发阶段的头文件和编译项目的源代码，这正是我们需要的理由。可以使用下面示例中的命令来安装这两种软件包。

使用 yum 来安装：

```
[root@example.com ~]# yum install pcre pcre-devel
```

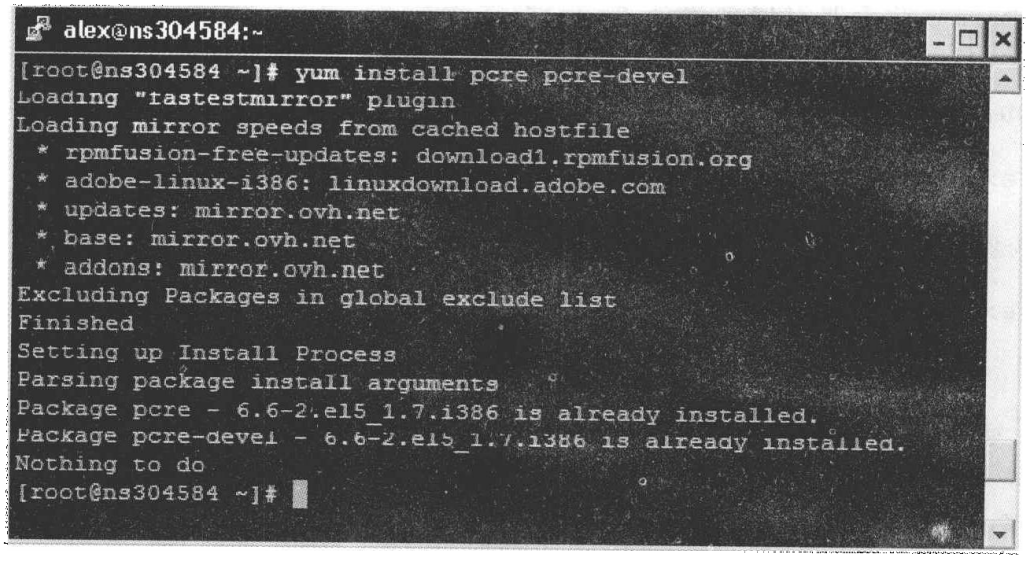
或者安装与 PCRE 相关的所有的安装包：

```
[root@example.com ~]# yum install pcre*
```

如果用 apt-get：

```
[root@example.com ~]# apt-get install libpcre3 libpcre3-dev
```

如果这些安装包已经安装在系统上，你会收到一条信息 Nothing to do(见下图)，换句话说，安装包管理器没有安装或升级任何部分。



zlib 库

zlib 库提供了开发人员的压缩算法，在 Nginx 的各种模块中需要使用 gzip 压缩。你可以再次使用包管理程序来安装这个组件，如同安装 PCRE 一样，同样需要安装库和它的源代码：zlib 和 zlib-devel。

使用 yum：

```
[root@example.com ~]# yum install zlib zlib-devel
```

使用 apt-get：

```
[root@example.com ~]# apt-get install zlib1g zlib1g-dev
```

这些安装包会快速安装完毕，并且没有依赖问题。

OpenSSL

OpenSSL 项目是一个协作开发健壮的、商业级的、全功能的、开源工具执行于安全套接层(SSL v2/v3)和传输层安全(TLS v1)的协议，也是一个完整强壮的通用加密库。该项目的管理者是来自世界范围内社区的志愿者，他们使用互联网表达、计划和开发 OpenSSL 工具箱，有关文档可以查看 *http://www.openssl.org*。

在 Nginx 中，如果服务器提供安全网页时则会用到 OpenSSL 库，我们需要安装库文件和它的开发安装包。安装 openssl 和 openssl-devel 过程如下。

使用 yum：

```
[root@example.com ~]# yum install openssl openssl-devel
```

使用 apt-get：

```
[root@example.com ~]# apt-get install openssl openssl-dev
```

注意你所在国家的法律和法规。一些国家不允许使用强壮的密码算法，对于你的使用，OpenSSL 和 Nginx 项目的作者、出版社和开发者将不承担相关法律责任。

现在已经安装完成所有的前提条件，你可以开始准备下载和编译 Nginx 的源代码了。

下载 Nginx

这种处理下载的方式会让我们发现各种资源，管理员可自行支配、所有与 Nginx 相关联的网站、社区(communities)和维基(wiki)。我们也将快速讨论你见到的不同版本，最后选择一个最合适你安装的版本。

网站和资源

尽管 Nginx 是一个新且刚刚成长起来的项目，但是在互联网上已经有相当数量的资源和积极的社区管理员和开发者。官方网站是 *www.nginx.net*，相当的简单，除了最新版本的下载链接，并不提供更多的信息和文档，而与之相反的是，你会在官方维基网站(*wiki.nginx.org*)上发现很多有趣的文档和例子。


Install


Modules


Addons


Configure


Community


Resources

Main

Nginx is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. Written by Igor Sysoev in 2005, Nginx now hosts nearly 6% (13M) of all domains worldwide.

Nginx is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption.

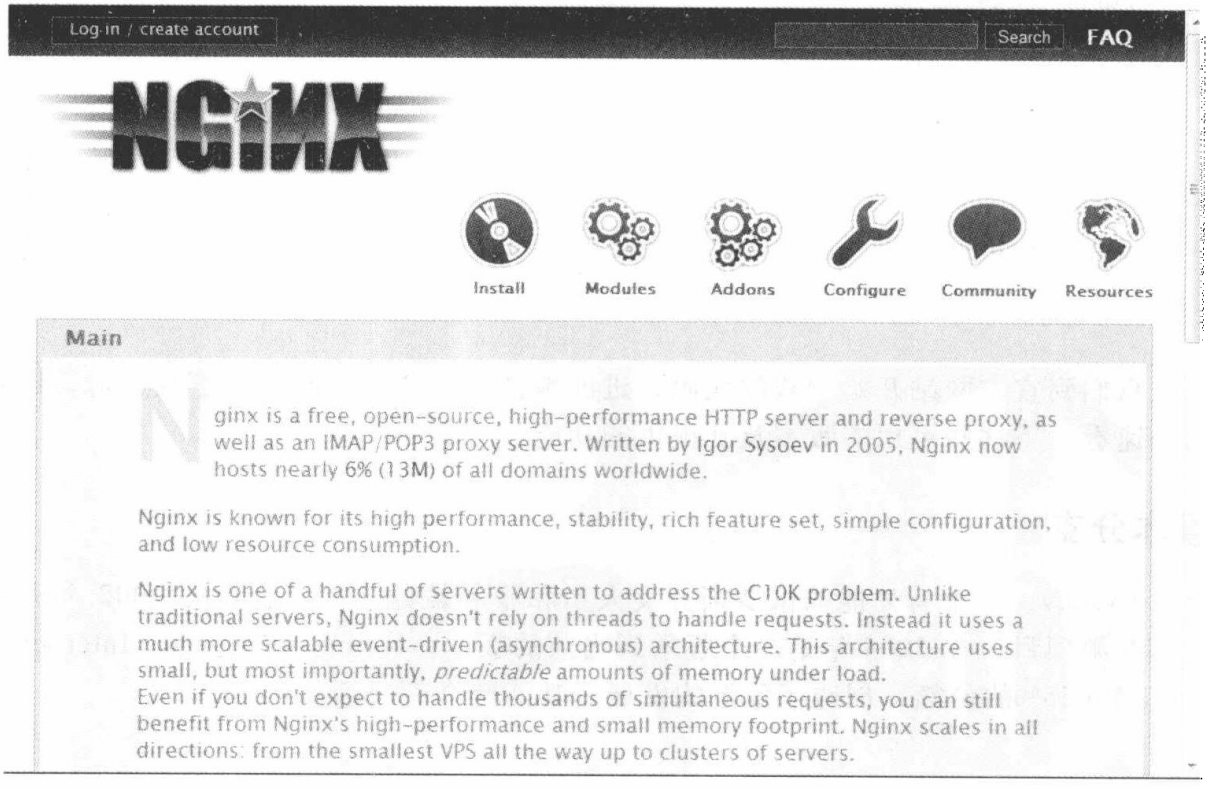
Nginx is one of a handful of servers written to address the C10K problem. Unlike traditional servers, Nginx doesn't rely on threads to handle requests. Instead it uses a much more scalable event-driven (asynchronous) architecture. This architecture uses small, but most importantly, *predictable* amounts of memory under load.
Even if you don't expect to handle thousands of simultaneous requests, you can still benefit from Nginx's high-performance and small memory footprint. Nginx scales in all directions: from the smallest VPS all the way up to clusters of servers.

下载 Nginx

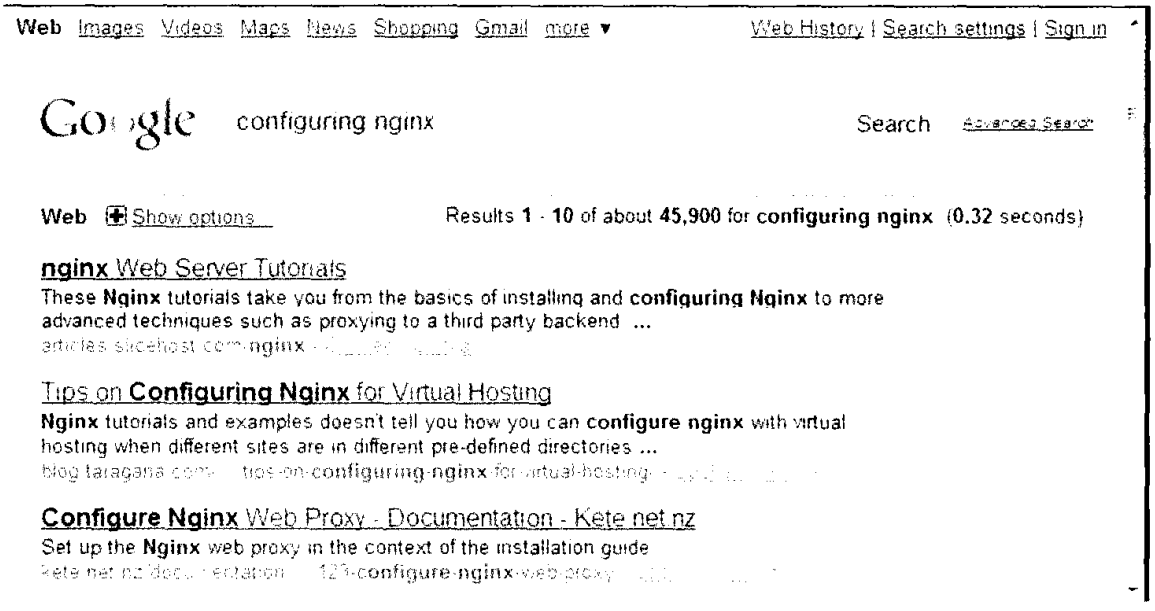
这种处理下载的方式会让我们发现各种资源，管理员可自行支配、所有与 Nginx 相关联的网站、社区(communities)和维基(wiki)。我们也将快速讨论你见到的不同版本，最后选择一个最合适你安装的版本。

网站和资源

尽管 Nginx 是一个新且刚刚成长起来的项目，但是在互联网上已经有相当数量的资源和积极的社区管理员和开发者。官方网站是 *www.nginx.net*，相当的简单，除了最新版本的下载链接，并不提供更多的信息和文档，而与之相反的是，你会在官方维基网站(*wiki.nginx.org*)上发现很多有趣的文档和例子。



如下图所示，维基提供了大量的文档和配置例子——而且已被证明在许多环境下对你非常有用。如果有具体的问题想问，可以使用论坛(*forum.nginx.org*)，很快就会有积极的社区用户回答你的问题。另外，Nginx 邮件列表，会将你询问的任何问题转发到 Nginx 论坛，这也将证实了 Nginx 有一个优秀的资源库。如果你需要直接帮助，可以在 Freenode 的 Nginx IRC 频道(*http://webchat.freenode.net/*)寻求帮助，总有一群人在有规律的提供帮助。还有其他有趣的资源信息——blogosphere，对于你执行的一个简单的兴趣查询，会返回大量的博客文档，例如 Nginx 的配置和模块。



现在我们到官方网站开始下载源代码，进而编译和安装 Nginx，在下载之前，让我们快速看一下这些可用的版本及其提供的功能。

版本分支

Igor Sysoev，一个有才能的俄罗斯开发人员和服务器管理员，最早从 2002 年开始这个开源项目，在 2004 年第一个版本到当前的版本，Nginx 现在已经在 Internet 上占据了 6.55%的份额，创造了稳定的发展，功能健全而灵活。

这个项目目前有三种版本支流。

- 稳定版本　这是通常被推荐使用的版本，开发者和用户认可的版本，但是通常有点滞后于开发版本。当前最新的稳定版本为 0.7.66，发布于 2010 年 7 月 7 日。
- 开发版本　这是用于下载的最新版本可见版本，虽然它足以在生产环境中部署，但是你可能还是会发现有 bug，因此，还是推荐使用稳定版本，尽管它没有最新版本的功能。当前最新的开发版本是 0.8.40，发布于 2010 年 7 月 7 日。
- 旧版版本　如果你因为某些原因对老版本感兴趣，那么你会找到两个版本，有一个旧版本和一个旧的稳定版本：0.5.38 和 0.6.39。

一个常见问题是开发版本“在生产服务器上是否足够稳定？” Cliff Wells，nginx.org 维基网站的创始人和社区维护人员认为：“我一般使用和推荐最新的开发版本”早期的用户很少报告危险的问题。具体选择什么版本由你决定，本书所给的指令都是适用的，不管发布的是什么版本，因为 Nginx 开发者们决定新版本的向后兼容性。你可能会发现很多关于版本改变的信息、bug 修复，这些均可以在官方网站的更改日志页中找到。

功能

作为稳定版本 0.7.66，Nginx 提供了一个令人印象深刻的多样化功能，和你想象的相反，不是所有的都关系到 HTTP 内容。这里有一个 web 分支(branch)的主要功能列表，引自官方网站 nginx.net：

- 处理静态文件、索引文件(index file)和自动索引(auto indexing)，打开文件描述符缓冲；
- 通过缓冲加速反向代理，简单的负载均衡和容错；
- 加速远程支持 FastCGI 服务器，简单的负载均衡和容错；

- 模块化结构。过滤器包括 gzip 压缩，字节服务(byte rang)，分块响应(chunked responses)，XSLT，SSI 和图片大小过滤器。在单个页中能够并行处理多重 SSI 包含，如果通过 FastCGI 或 proxied 服务器，这些均可处理。
- SSL 和 TLS SNI 支持(具有 SNI(Server Name Indication) 的 TLS，在服务器上做虚拟主机需要使用 TLS)。

Nginx 也能够作为邮件代理服务器，这方面内容在本书中没有详述。

1. 用户在外部使用 HTTP 认证服务器重定向到 IMAP/POP3 后端服务器。
2. 用户在外部使用 HTTP 认证服务器，连接重定向到内部的 SMTP 后台服务器。
3. 认证方法如下。
   - POP3：USER/PASS、APOP、AUTH LOGIN/PLAIN/CRAM-MD5
   - IMAP：LOGIN、AUTH LOGIN/PLAIN/CRAM-MD5
   - SMTP：AUTH LOGIN/PLAIN/CRAM-MD5
4. 支持 SSL。
5. 支持 STARTTLS 和 STLS 。

Nginx 能够被许多计算机结构和操作系统编译，像 Windows, Linux, Mac OS, FreeBSD 及 Solaris 操作系统，也可以运行在 32 位或 64 位的结构下。

下载并解压

一旦决定选择使用的版本，就到 nginx.net 找到相关 URL，然后下载到 home 目录，下载的安装包中包括要编译的源代码，可以通过 wget 来下载：

```
[alex@example.com ~]$ mkdir src && cd src
[alex@example.com src]$ wget http://nginx.org/download/nginx-
0.7.66.tar.gz
```

我们将使用 0.7.66 版本，这是 2010 年 6 月 7 日发布的最新稳定版本。下载完成后可在当前目录下解压安装包：

```
[alex@example.com src]$ tar zxf nginx-0.7.66.tar.gz
```

你已经成功下载并解压了 Nginx 安装包，现在，为了适应操作系统，为了获得二进制的运行，要对 Nginx 进行编译配置处理。

配置选项

创建一个应用程序通常分为三步，从源代码到配置、编译和安装编译。配置步骤允许你选择许多选项，这些选项在完成程序的建立安装后不可编辑，因此它直接影响该项目的二进制文件。所以，这一步非常重要，如果想在后来避免意外，必须仔细，例如在一个无计划的目录中缺乏指定的模块或文件。

该过程是添加某些开关选项(switch)到配置(configure)脚本，该脚本来就在源代码安装包中，在下面的内容中我们将了解到，可以激活的开关选项有三种类型，但我们首先研究最容易的。

容易的方法

如果是因为某种原因，你不想打扰配置步骤，例如只是测试或者是简单的安装，因为将来还会重新编译安装该程序，所以你只是简单地使用 configure 脚本，而并没有使用任何开关选项。执行下列三个命令就可以建立和安装工作版(能够进行 Web 服务)的 Nginx：

```
[alex@example.com nginx-0.7.66]# ./configure
```

运行了该命令后，就开始了一个长时间的程序验证过程，以便确定系统包含所有必要的组成成分。如果配置过程失败，请再次检测先决条件，现在看来，它是引发错误的最常见原因。要想查看为什么该命令会失败，也可以求助于“objs/autoconf.err”文件，该文件提供了更详细的报告：

```
[alex@example.com nginx-0.7.66]# make
```

命令 make 对应用程序进行编译，如果配置部分运行良好，那么这一部分不会出现任何错误。

```
[root@example.com nginx-0.7.66]# make install
```

这是最后一步，复制编译后的文件(也包括资源文件)到安装目录，默认情况下是目录“/usr/local/nginx”，这一步可能需要以 root 的身份登录系统，然后来执行安装的操作，这依赖于你对当前用户授予“/usr/local”的权限。

为此，如果你没有经过自定义配置就创建了该应用程序，那么你的这个冒险便会错过很多功能，例如可选择的模块和我们即将认识的其他模块。

路径选项

在运行 configure 脚本命令的时候，可能开启一些开关选项，例如，需要对 Nginx 各种组成成分指定目录或文件路径。会注意配置脚本提供的开关选项，可能会因下载的版本而不同。下面列举的选项都在稳定发布版本 0.7.66 中有效。如果使用的是其他版本，可以运行“./configure --help”命令列出有效的开关变量，以便于安装的选择。典型的做法是使用开关命令在命令行添加一些文本，例如，使用开关选项 --conf-path：

```
[alex@example.com nginx-0.7.66]# ./configure --conf-
path=/etc/nginx/nginx.conf
```

下表中，全部列出配置时的开关选项。

选　项	用　法	默 认 值
--prefix=...	指定安装 Nginx 的基础目录	**/usr/local/nginx** 注意：如果你在配置时使用了相对路径，则连接到基础目录。 示例： 指定 **--conf-path=conf/nginx.conf** 则配置文件会在目录： /usr/local/nginx/conf/nginx.conf
--sbin-path=...	Nginx 二进制文件安装的路径	<prefix>/sbin/nginx
--conf-path=...	主要配置文件放置的目录	<prefix>/conf/nginx.conf
--error-log -path=...	错误日志存放的路径。 错误日志在配置文件中须配置得非常正确，该路径只应用于你在配置文件中没有指定任何错误的日志指令时	<prefix>/logs/error.log

续表

选　项	用　法	默 认 值
--pid-path=...	指定 Nginx 的 pid 文件的路径。可以在配置文件中指定 pid 文件的路径，如果没有具体的指定，则使用在这里对该选项指定的该路径	**<prefix>/logs/nginx.pid** 注意：该 pid 文件是一个简单的文本文件，它包含进程的标识符。该文件应该放置在一个清晰可见的位置，以便其他应用程序能够很容易找到运行该程序的 pid
--lock-path=...	锁文件(lock file)的存放路径。同样，该文件也可以在配置文件中指定，但是，如果在配置文件中没有指定，则使用该值	**<prefix>/logs/nginx.lock** 注意：锁文件允许其他应用程序确定是否一个程序在运行，就 Nginx 来说，它用于确定该进程没有被启动两次
--with-perl_modules_path=...	定义 Perl 模块的路径。如果需要包含另外的 Perl 模块，必须定义该参数	
--with-perl=...	Perl 二进制文件的路径。用于执行 Perl 脚本。如果想执行一个 Perl 脚本，必须设置该路径	
--http-log-path=...	定义被访问文件的日志文件存放路径。该路径只用于在配置文件中没有定义访问日志的情况	**<prefix>/logs/access.log**
--http-client-body-temp-path=...	该目录用于存储客户端请求产生的临时文件	**<prefix>/client_body_temp**
--http-proxy-temp-path=...	该目录用于代理存储临时文件	**<prefix>/proxy_temp**
--http-fastcgi-temp-path=...	指定用于 HTTP FastCGI 模块使用的临时文件的存放	**<prefix>/fastcgi_temp**
--builddir=...	指定创建应用程序的位置	

先决条件选项

先决条件的格式有库文件和二进制文件。到现在，你应该已经把它们安装在系统中了，然而即使它们已经安装在系统中，可能有时候配置脚本还是不能找到它们的位置。原因有多种，例如，如果它们安装在非标准路径中。为了修复这些问题，可以使用下表所列的开关项来指出它们所在的路径。在杂项中，将其他的依赖选择组织在一起。

--with-cc=...	指定一个备用的 C 编译器的位置
--with-cpp=...	指定一个备用的 C 预处理器的位置
--with-cc-opt=...	定义额外的选项，然后在命令行传递给 C 编译器
--with-ld-opt=...	定义额外的选项，然后在命令行传递给 C 连接器
--with-cpu-opt=...	指定不同的目标处理器结构，可以是下列值：pentium，pentiumpro，pentium3，pentium4，athlon，opteron，sparc32，sparc64 和 ppc64

--without-pcre	不使用 PCRE 库。这个设置不推荐使用，因为它会移除对正则表达式的支持，从而使 Rewrite 模块失去作用
--with-pcre	强制使用 PCRE 库
--with-pcre=...	允许指定 PCRE 库的源代码
--with-pcre-opt=...	用于建立 PCRE 库的另外的选项

--with-md5=...	指定 MD5 库源代码的路径
--with-md5-opt=...	用于建立 MD5 库的另外选项
--with-md5-asm	为建 MD5 库使用汇编语言源代码

--with-sha1=...	指定 SHA1 库的源代码
--with-sha1-opt=...	用于建立 SHA1 库的另外选项
--with-sha1-asm	为建 SHA1 库使用汇编语言源代码

--with-zlib=...	指定 zlib 库的源代码
--with-zlib-opt=...	用于建立 zlib 库的另外的选项
--with-zlib-asm=...	使用汇编语言最大限度地优化下列目标结构：Pentium，pentiumpro

编译选项

PCRE 选项

MD5 选项

SHA1 选项

zlib 选项

先决条件选项

先决条件的格式有库文件和二进制文件。到现在，你应该已经把它们安装在系统中了，然而即使它们已经安装在系统中，可能有时候配置脚本还是不能找到它们的位置。原因有多种，例如，如果它们安装在非标准路径中。为了修复这些问题，可以使用下表所列的开关项来指出它们所在的路径。在杂项中，将其他的依赖选择组织在一起。

编译选项	
--with-cc=...	指定一个备用的 C 编译器的位置
--with-cpp=...	指定一个备用的 C 预处理器的位置
--with-cc-opt=...	定义额外的选项，然后在命令行传递给 C 编译器
--with-ld-opt=...	定义额外的选项，然后在命令行传递给 C 连接器
--with-cpu-opt=...	指定不同的目标处理器结构，可以是下列值：pentium，pentiumpro，pentium3，pentium4，athlon，opteron，sparc32，sparc64 和 ppc64
PCRE 选项	
--without-pcre	不使用 PCRE 库。这个设置不推荐使用，因为它会移除对正则表达式的支持，从而使 Rewrite 模块失去作用
--with-pcre	强制使用 PCRE 库
--with-pcre=...	允许指定 PCRE 库的源代码
--with-pcre-opt=...	用于建立 PCRE 库的另外的选项
MD5 选项	
--with-md5=...	指定 MD5 库源代码的路径
--with-md5-opt=...	用于建立 MD5 库的另外选项
--with-md5-asm	为建 MD5 库使用汇编语言源代码
SHA1 选项	
--with-sha1=...	指定 SHA1 库的源代码
--with-sha1-opt=...	用于建立 SHA1 库的另外选项
--with-sha1-asm	为建 SHA1 库使用汇编语言源代码
zlib 选项	
--with-zlib=...	指定 zlib 库的源代码
--with-zlib-opt=...	用于建立 zlib 库的另外的选项
--with-zlib-asm=...	使用汇编语言最大限度地优化下列目标结构：Pentium，pentiumpro

--with-openssl=...	指定 OpenSSL 库的源代码路径
--with-openssl-opt=...	为建立 OpenSSL 库的另外的选项

模块选项

模块，将在第 4 章和后面几章讨论，在编译 Nginx 之前需要对模块进行选择，一些模块默认是开启的，有些模块需要手动开启，详细情况参考下文。

默认开启的模块

下表列出的模块默认情况下是开启的，可以通过下列开关项关闭这些模块。

--without-http_charset_module	禁用 Charset 模块，该模块用于对网页重新编码
--without-http_gzip_module	禁用 Gzip 压缩模块
--without-http_ssi_module	禁用服务器端包含模块
--without-http_userid_module	禁用用户 ID 模块。该模块为用户通过 cookie 验证身份
--without-http_access_module	禁用访问模块，对于指定的 IP 段，允许访问配置
--without-http_auth_basic_module	禁用基本的认证模块
--without-http_autoindex_module	禁用自动索引模块
--without-http_geo_module	禁用 Geo 模块，该模块允许你定义依赖于 IP 地址段的变量
--without-http_map_module	禁用 Map 模块，该模块允许你声明 map 区段
--without-http_referer_module	禁用 Referer 控制模块
--without-http_rewrite_module	禁用 Rewrite 模块
--without-http_proxy_module	禁用代理模块。该模块用于向其他服务器传输请求

默认开启的模块	描述

OpenSSL 选项

续表

OpenSSL 选项	
--with-openssl=...	指定 OpenSSL 库的源代码路径
--with-openssl-opt=...	为建立 OpenSSL 库的另外的选项

模块选项

模块，将在第 4 章和后面几章讨论，在编译 Nginx 之前需要对模块进行选择，一些模块默认是开启的，有些模块需要手动开启，详细情况参考下文。

默认开启的模块

下表列出的模块默认情况下是开启的，可以通过下列开关项关闭这些模块。

默认开启的模块	描　述
--without-http_charset_module	禁用 Charset 模块，该模块用于对网页重新编码
--without-http_gzip_module	禁用 Gzip 压缩模块
--without-http_ssi_module	禁用服务器端包含模块
--without-http_userid_module	禁用用户 ID 模块。该模块为用户通过 cookie 验证身份
--without-http_access_module	禁用访问模块，对于指定的 IP 段，允许访问配置
--without-http_auth_basic_module	禁用基本的认证模块
--without-http_autoindex_module	禁用自动索引模块
--without-http_geo_module	禁用 Geo 模块，该模块允许你定义依赖于 IP 地址段的变量
--without-http_map_module	禁用 Map 模块，该模块允许你声明 map 区段
--without-http_referer_module	禁用 Referer 控制模块
--without-http_rewrite_module	禁用 Rewrite 模块
--without-http_proxy_module	禁用代理模块。该模块用于向其他服务器传输请求

--without-http_fastcgi_module	禁用 FastCGI 模块。该模块是用于与 FastCGI 进程配合工作
--without-http_memcached_module	禁用 Memcached 模块。该模块是用于与 memcached 守护进程配合工作
--without-http_limit_zone_module	禁用 Limit Zone 模块。该模块是用于根据定义的 zone 来限制约束对资源的使用
--without-http_limit_req_module	禁用 Limit Requests 模块。该模块允许你限制每个用户请求的总数
--without-http_empty_gif_module	禁用 Empty Gif 模块。该模块用于在内存中提供一个空白的 GIF 图像
--without-http_browser_module	禁用 Browser 模块。该模块用于解释用户代理字符串
--without-http_upstream_ip_hash_module	禁用 Upstream 模块。该模块用于配置负载均衡结构

默认禁用的模块

下表列出的开关选项允许你开启默认禁用的模块。

--with-http_ssl_module	开启 SSL 模块，支持使用 HTTPS 协议的网页
--with-http_realip_module	开启 Real IP 的支持，该模块用于从客户请求的头数据中读取 real IP 地址
--with-http_addition_module	开启 Addition 模块，该模块允许你追加或前置数据(prepend data)到响应的主体部分
--with-http_xslt_module	开启 XSLT 模块的支持，该模块实现 XSL 转化为 XML 文档
--with-http_image_filter_module	开启 Image Filter 模块，该模块是让你修改图像。注意：如果想编译该模块，需要在系统中安装 libgd 库

默认开启的模块	描述

默认禁用的模块	描述

续表

默认开启的模块	描　　述
--without-http_fastcgi_module	禁用 FastCGI 模块。该模块是用于与 FastCGI 进程配合工作
--without-http_memcached_module	禁用 Memcached 模块。该模块是用于与 memcached 守护进程配合工作
--without-http_limit_zone_module	禁用 Limit Zone 模块。该模块是用于根据定义的 zone 来限制约束对资源的使用
--without-http_limit_req_module	禁用 Limit Requests 模块。该模块允许你限制每个用户请求的总数
--without-http_empty_gif_module	禁用 Empty Gif 模块。该模块用于在内存中提供一个空白的 GIF 图像
--without-http_browser_module	禁用 Browser 模块。该模块用于解释用户代理字符串
--without-http_upstream_ip_hash_module	禁用 Upstream 模块。该模块用于配置负载均衡结构

默认禁用的模块

下表列出的开关选项允许你开启默认禁用的模块。

默认禁用的模块	描　　述
--with-http_ssl_module	开启 SSL 模块，支持使用 HTTPS 协议的网页
--with-http_realip_module	开启 Real IP 的支持，该模块用于从客户请求的头数据中读取 real IP 地址
--with-http_addition_module	开启 Addition 模块，该模块允许你追加或前置数据(prepend data)到响应的主体部分
--with-http_xslt_module	开启 XSLT 模块的支持，该模块实现 XSL 转化为 XML 文档
--with-http_image_filter_module	开启 Image Filter 模块，该模块是让你修改图像。注意：如果想编译该模块，需要在系统中安装 libgd 库

--with-http_geoip_module	开启 GeoIP 模块，该模块通过使用 MaxMind's GeoI 二进制数据库来获取客户端在地理上的分布。 注意：如果希望编译该模块，需要在系统中安装 libgeoip 库
--with-http_sub_module	开启 Substitution 模块，该模块用于在网页中替换文本
--with-http_dav_module	开启 WebDAV 模块 (Distributed Authoring and Versioning via Web)
--with-http_flv_module	开启 FLV 模块，该模块用于专门处理.flv(flash 视频)文件
--with-http_gzip_static_module	开启 Gzip 静态模块，该模块用于发送预压缩的文件
--with-http_random_index_module	开启 Random Index 模块。该模块用于挑选一个随机的文件作为该目录的 index
--with-http_secure_link_module	开启 Secure Link 模块，该模块用于在 URL 中检测关键字的存在
--with-http_stub_status_module	开启 Stub Status 模块，该模块会产生一个服务器状态和信息页
--with-google_perftools_module	开启 Google 性能工具模块

杂项

下表列出的是在配置脚本中有效的一些选项，例如，邮件代理服务功能或时间管理功能。

--with-mail	开启邮件服务代理(mail server proxy)模块,支持 POP3, IMAP4 和 SMTP。该功能默认禁用
--with-mail_ssl_module	开启邮件代理服务对 SSL 的支持。该功能默认禁用
--without-mail_pop3_module	在邮件代理下禁用 POP3 功能。在开启邮件代理模块后该功能默认启用
--without-mail_imap_module	对邮件代理服务器禁用 IMAP4 模块，在开启邮件代理模块后该功能默认启用
--without-mail_smtp_module	对于邮件代理服务器禁用 SMTP 模块，在开启邮件代理模块后该功能默认启用

默认禁用的模块	描述

邮件服务代理

续表

默认禁用的模块	描　　述
--with-http_geoip_module	开启 GeoIP 模块，该模块通过使用 MaxMind's GeoI 二进制数据库来获取客户端在地理上的分布。 注意：如果希望编译该模块，需要在系统中安装 libgeoip 库
--with-http_sub_module	开启 Substitution 模块，该模块用于在网页中替换文本
--with-http_dav_module	开启 WebDAV 模块 (Distributed Authoring and Versioning via Web)
--with-http_flv_module	开启 FLV 模块，该模块用于专门处理.flv(flash 视频)文件
--with-http_gzip_static_module	开启 Gzip 静态模块，该模块用于发送预压缩的文件
--with-http_random_index_module	开启 Random Index 模块。该模块用于挑选一个随机的文件作为该目录的 index
--with-http_secure_link_module	开启 Secure Link 模块，该模块用于在 URL 中检测关键字的存在
--with-http_stub_status_module	开启 Stub Status 模块，该模块会产生一个服务器状态和信息页
--with-google_perftools_module	开启 Google 性能工具模块

杂项

下表列出的是在配置脚本中有效的一些选项，例如，邮件代理服务功能或时间管理功能。

邮件服务代理	
--with-mail	开启邮件服务代理(mail server proxy)模块,支持 POP3, IMAP4 和 SMTP。该功能默认禁用
--with-mail_ssl_module	开启邮件代理服务对 SSL 的支持。该功能默认禁用
--without-mail_pop3_module	在邮件代理下禁用 POP3 功能。在开启邮件代理模块后该功能默认启用
--without-mail_imap_module	对邮件代理服务器禁用 IMAP4 模块，在开启邮件代理模块后该功能默认启用
--without-mail_smtp_module	对于邮件代理服务器禁用 SMTP 模块，在开启邮件代理模块后该功能默认启用

--with-rtsig_module	开启 rtsig 模块，使用 rtsig 作为事件通知机制。
--with-select_module	开启 select 模块，使用 select 作为事件通知机制。默认情况下，该模块是开启的，除非系统有一种更好的方式发现——kqueue，epoll，rtsig 或 poll
--without-select_module	禁用 select 模块
--with-poll_module	开启 poll 模块，该模块使用 poll 作为事件通知机制。默认情况下，如果有效，该模块是开启的。除非系统上有一种更好的方式发现——kqueue，epoll 或 rtsig
--without-poll_module	禁用 poll 模块

--user=…	指定启动 Nginx 进程的默认用户。这个设置仅用于在配置文件中省略 user 指令来指定用户的情况
--group=…	指定启动 Nginx 进程默认的用户组。这个设置仅用于在配置文件中省略使用 group 指令来指定用户的情况

--with-ipv6	开启对 IPv6 的支持
--without-http	禁用 HTTP 服务
--without-http-cache	禁用 HTTP 缓冲功能
--add-module=PATH	通过指定的路径编译添加第三方模块。如果希望编译多个模块，那么该选项可以无限次使用
--with-debug	开启记录额外的调试信息

配置举例

这里有一些 configuration 脚本命令的例子，可能应用于各种不同的情况，在这些例子中，路径开关选项被忽略，因为在每个系统中它们都被明确指定，保留默认值只是为了能够正常工作。

事件管理：

允许你为 Nginx 定时器选择事件通知系统，仅适用于高级用户

用户和组选项

其他选项

续表

事件管理： 允许你为 Nginx 定时器选择事件通知系统，仅适用于高级用户	
--with-rtsig_module	开启 rtsig 模块，使用 rtsig 作为事件通知机制。
--with-select_module	开启 select 模块，使用 select 作为事件通知机制。默认情况下，该模块是开启的，除非系统有一种更好的方式发现——kqueue，epoll，rtsig 或 poll
--without-select_module	禁用 select 模块
--with-poll_module	开启 poll 模块，该模块使用 poll 作为事件通知机制。默认情况下，如果有效，该模块是开启的。除非系统上有一种更好的方式发现——kqueue，epoll 或 rtsig
--without-poll_module	禁用 poll 模块
用户和组选项	
--user=…	指定启动 Nginx 进程的默认用户。这个设置仅用于在配置文件中省略 user 指令来指定用户的情况
--group=…	指定启动 Nginx 进程默认的用户组。这个设置仅用于在配置文件中省略使用 group 指令来指定用户的情况
其他选项	
--with-ipv6	开启对 IPv6 的支持
--without-http	禁用 HTTP 服务
--without-http-cache	禁用 HTTP 缓冲功能
--add-module=PATH	通过指定的路径编译添加第三方模块。如果希望编译多个模块，那么该选项可以无限次使用
--with-debug	开启记录额外的调试信息

配置举例

这里有一些 configuration 脚本命令的例子，可能应用于各种不同的情况，在这些例子中，路径开关选项被忽略，因为在每个系统中它们都被明确指定，保留默认值只是为了能够正常工作。

请注意，这些配置没有包含第三方模块，有关安装附加模块的更多信息，请参考第 5 章。

关于 prefix 开关选项

在配置期间，应该特别注意--prefix 开关选项，将来许多配置指令(我们将在后面的章节进一步尝试)都基于你在这时选择的路径。记住，一旦二进制文件编译完成，prefix 就不能再变，然而这并不是一个决定性的问题，因为仍然可以使用绝对路径。

如果你打算与时俱进，想要升级 Nginx，以便使用一个新的发布版，默认的 prefix(如果没有通过使用--prefix 开关选项去覆盖)是/usr/local/nginx——一个不包含版本号的路径，因此，升级 Nginx 的时候，如果没有指定不同的 prefix，新安装的文件就会覆盖先前安装的文件，这种做法可能存在其他问题，因为这有可能会清除你的配置文件和正在运行的二进制文件。

因此推荐使用不同的 prefix，利用 prefix 指明每一个版本号，例如：

```
./configure --prefix=/usr/local/nginx-0.7.66
```

另外，为了使将来简单一些，可以为/usr/local/nginx 建立一个符号链接来指向/usr/local/nginx-0.7.66，一旦升级，该链接便会指向新的/usr/local/nginx-newer.version。这会(例如)使 init 脚本总是能够使用最新安装的 Nginx 版本。

普通的 HTTP 和 HTTPS 服务器

第一个例子描述的是这样一种情况，为 HTTP 服务开启 HTTPS 服务，并包含最重要的功能和模块，而与邮件相关的选项都被禁用：

```
./configure --user=www-data --group=www-data / --with-
http_ssl_module --with-http_realip_module
```

可以看出，命令行相当简单，大多数开关选项都被省略。原因是：默认的配置是相当高效的，并且大多数模块已被启用。你只需要为 HTTPS 协议包含 http_ssl 模块即可。考虑到 Nginx 服务器可能被用作后端运行，所以顺便又添加了“real IP”模块，该模块用于检索访问者的 IP 地址。

开启所有的模块

以下这一种情况：整个安装包。开启所有的模块支持，由自己来决定是否在运行时使用它们。

```
./configure --user=www-data --group=www-data /
            --with-http_ssl_module  /
            --with-http_realip_module  /
            --with-http_addition_module    /
            --with-http_xslt_module /
            --with-http_image_filter_module  /
            --with-http_geoip_module /
            --with-http_sub_module         /
            --with-http_dav_module /
            --with-http_flv_module  /         /
            --with-http_gzip_static_module   /
            --with-http_random_index_module  /
            --with-http_secure_link_module   /
            --with-http_stub_status_module
```

这种配置开启了最广泛的配置选择，第 4 章～第 7 章提供了关于模块配置的详细信息。

在这种安装下，所有可选模块都被开启，因此需要安装额外的库，例如 libgeoip(用于 Geo IP 模块)；libgd(用于 Image Filter 模块)；libxml2 和 libxslt(用于 XSLT 模块)。可以通过系统中的安装包管理器来安装这些先决条件，例如可以通过 yum 来安装 libxml2，也可以通过 apt-get 来安装 libxml2。

邮件服务器代理

最后一个建立的配置稍微有点特殊，它用于开启邮件代理功能，这是 Nginx 不被重视的一面，以下配置开启与该功能相关的所有模块：

```
./configure --user=www-data --group=www-data  \
            --with-mail --with-mail_ssl_module
```

如果愿意彻底禁用 HTTP 服务功能，而只是将 Nginx 用于邮件代理，可以添加一个--without-http 开关选项。

注意，在前面列出的命令中，www-data 用于运行 Nginx 进程的用户和组(user 和 group)，所以工作进程将以此组合来运行。因此，系统上必须有这样的用户和用户组。添加用户和用户组的相关信息，可以参阅第 1 章。

建立配置的问题

在某些情况下，configure 命令可能会失败——在一个很长的检查列表后，你可能会在终端上收到一些错误的信息，很多情况下(但不是全部)，这些错误与丢失先决条件或没有指定路径相关。

在这种情况下，需要继续下面的工作，即仔细校验，以确保有所有需要编译的应用程序，并且要随意翻阅 objs/autoconf.err 文件，该文件详细记录了编译出错的相关问题，它是在 configure 进程进行期间产生的，它会详细告诉你进程在哪里出了问题。

请确保先决条件

基本的四个先决条件是：GCC, PCRE, zlib 和 OpenSSL。最后是三个库，针对每个库，必须安装两个安装包：一个是库自身，另一个是开发源代码。确定你已经安装两者。请参阅在本章开始部分的先决条件。注意其他先决条件，例如其他的扩展模块可能需要 LibXML2 或 LibXSLT，例如 HTTP XSLT 模块。

如果肯定已经正确安装所有的先决条件，那么造成编译错误的原因是 configure 脚本无法定位先决条件文件的位置。在这种情况下，确定你包含的开关选项与相关的文件路径相关联。

例如，下列开关项允许你指定 OpenSSL 库文件的位置：

```
./configure [...] --with-openssl=/usr/lib64
```

所以，配置脚本会在前面指定的路径寻找 OpenSSL 库。

目录的存在和可写性

一定记得检查明显的错误，每一个人迟早都会犯一些最初级的错误。确定一下对于存放 Nginx 文件的目录，运行配置和编译脚本的用户是否有读和写的权限，也要确定一下在配置脚本各个开关项中指定的路径是否存在，是否有效。

最后，所有的问题都解决后，需要看一下“configuration summary”，差不多类似于下图。

编译和安装

配置过程相当重要——它会产生一个 makefile，依赖于应用程序所选择的开关项，会根据需求在系统上执行一个较长时间的检查。一旦 configure 脚本成功执行，你就可以继续编译 Nginx。

编译该项目就是在该项目的源代码目录下执行 make 命令：

```
[alex@example.com nginx-0.7.66]$ make
```

一个成功的 build 编译应该会出现最后的信息：

```
make[1]: leaving directory followed by the project source path.
```

此外，在编译时可能发生的问题，大多数可以归因于找不到先决条件或指定的路径无效。如果是这些原因，可以再次运行 configure 脚本，然后再重新检查开关选项和所有的先决条件选项。也可能是你下载了比较新的版本，新版本可能不向后兼容，针对这种情况，最好的选择就是访问官方网站找老一点的版本。

alex@extremejeux:~/src/nginx-0.7.66

```
Configuration summary
  + using system PCRE library
  + using system OpenSSL library
  + md5: using OpenSSL library
  + sha1 library is not used
  + using system zlib library

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

[alex@extremejeux nginx-0.7.66]$
```

最后，所有的问题都解决后，需要看一下“configuration summary”，差不多类似于下图。

```
alex@extremejeux:~/src/nginx-0.7.66
Configuration summary
  + using system PCRE library
  + using system OpenSSL library
  + md5: using OpenSSL library
  + sha1 library is not used
  + using system zlib library

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

[alex@extremejeux nginx-0.7.66]$
```

编译和安装

配置过程相当重要——它会产生一个 makefile，依赖于应用程序所选择的开关项，会根据需求在系统上执行一个较长时间的检查。一旦 configure 脚本成功执行，你就可以继续编译 Nginx。

编译该项目就是在该项目的源代码目录下执行 make 命令：

```
[alex@example.com nginx-0.7.66]$ make
```

一个成功的 build 编译应该会出现最后的信息：

```
make[1]: leaving directory followed by the project source path.
```

此外，在编译时可能发生的问题，大多数可以归因于找不到先决条件或指定的路径无效。如果是这些原因，可以再次运行 configure 脚本，然后再重新检查开关选项和所有的先决条件选项。也可能是你下载了比较新的版本，新版本可能不向后兼容，针对这种情况，最好的选择就是访问官方网站找老一点的版本。

如果一个编译进程成功完成，需要准备下一步——安装应用程序：

```
[alex@example.com nginx-0.7.66]$ make install
```

命令 make install 会执行 makefile 文件中的安装部分。换句话说，它执行了一些简单的操作，例如复制二进制文件和配置文件到指定的安装目录。如果用于存储 HTML 文件和日志文件的目录不存在，则新建这些目录。make install 这一步一般不会有问题，除非系统遇到一些异常情况，例如存储空间或内存不足。

在目录/usr/local/下安装应用程序时，可能需要 root 的权限，这依赖于你对目录的权限设置。

控制 Nginx 服务

眼下，你应该成功安装并建立了 Nginx，默认的位置为/usr/local/nginx，因此我们将来的例子都将基于这个目录。

守护进程和服务

下一步，显然是执行 Nginx， 然而，在这么做之前，重要的是了解一下这个应用程序的性质。计算机应用程序有两个类型——有的需要用户在前端输入命令后立即运行；有的则不然，运行于后台。

Nginx 属于后者，即经常提及的作为守护进程运行的那种程序。守护进程的名称后通常带有“d”字样。这里有几个例子——httpd 是 HTTP 服务器的守护进程；named 是域名服务器的守护进程；crond 是任务调度程序的守护进程。然而，需要注意，Nginx 不是这种情况。从命令行启动 Nginx 的时候，守护进程会立即返回到命令行提示符，在大多数情况下，用不着向终端屏幕输出数据。

因此，启动 Nginx 的时候，屏幕上不会出现任何文本信息，并且提示符会立即返回。虽然这似乎令人吃惊，但这是一个好的迹象，相反，这意味着你的守护进程已经在正确运行，说明你的配置没有任何错误。

用户和组

一个最普遍的麻烦来源是，对 Nginx 设置的是一个无效的访问权限的时候——由于用户或用户组的错误配置，经常会报告“403 Forbidden”错误，因为 Nginx 不能访问你需要的文件。

根据进程的功能，可能需要两个不同级别的进程权限：

1. Nginx 的 master 进程，由 root 启动，在大多数类 Unix 系统上，由 root 帐户开启的进程允许在开放任何端口的 TCP 套接字，但是其他用户启动的程序只能监听在 1024 以上的端口。如果不是以 root 帐户启动 Nginx，将无法得到标准的端口(如 80 或 443)，即无法启动。而且，通过 user 指令在配置文件中指定的用户和组用于工作进程，在这里不予考虑。
2. Nginx 的工作进程，由配置文件中 user 指令(详见第 3 章)指定的帐户开始运行，配置文件中的设置优先于在配置时使用 configure 脚本开关选项指定的用户。如果没有做任何指定，工作进程将以用户 nobody 开始，用户组为 nobody 组(或 nogroup，具体取决于操作系统)。

Nginx 命令行开关项

Nginx 二进制文件接收命令行参数，用于执行各种操作，控制后台进程。为了获取该命令的全部参数列表，可以请求 help 帮助，使用下面的命令即可：

```
[alex@example.com ~]$ cd /usr/local/nginx/sbin
[alex@example.com sbin]$ ./nginx -h
```

alex@example:/usr/local/nginx/sbin

```
[alex@example sbin]$ ./nginx -?
nginx version: nginx/0.7.66
Usage: nginx [-?hvVt] [-s signal] [-c filename] [-p prefix] [-g directives]

Options:
  -?,-h         : this help
  -v            : show version and exit
  -V            : show version and configure options then exit
  -t            : test configuration and exit
  -s signal     : send signal to a master process: stop, quit, reopen, reload
  -p prefix     : set prefix path (default: /usr/local/nginx/)
  -c filename   : set configuration file (default: conf/nginx.conf)
  -g directives : set global directives out of configuration file

[alex@example sbin]$
```

用户和组

一个最普遍的麻烦来源是，对 Nginx 设置的是一个无效的访问权限的时候——由于用户或用户组的错误配置，经常会报告“403 Forbidden”错误，因为 Nginx 不能访问你需要的文件。

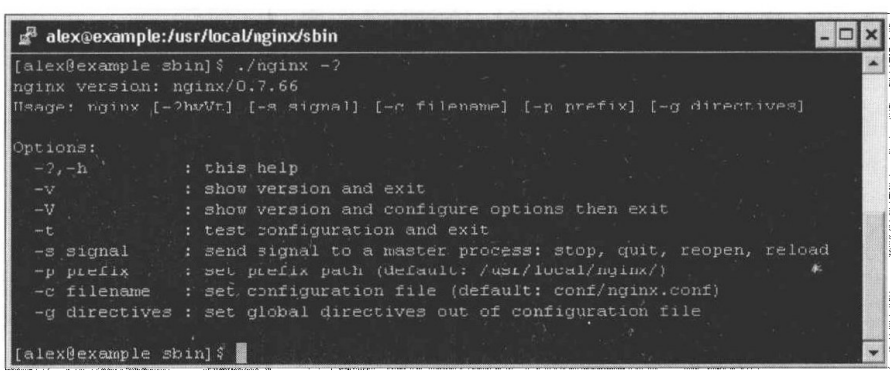
根据进程的功能，可能需要两个不同级别的进程权限：

1. Nginx 的 master 进程，由 root 启动，在大多数类 Unix 系统上，由 root 帐户开启的进程允许在开放任何端口的 TCP 套接字，但是其他用户启动的程序只能监听在 1024 以上的端口。如果不是以 root 帐户启动 Nginx，将无法得到标准的端口(如 80 或 443)，即无法启动。而且，通过 user 指令在配置文件中指定的用户和组用于工作进程，在这里不予考虑。
2. Nginx 的工作进程，由配置文件中 user 指令(详见第 3 章)指定的帐户开始运行，配置文件中的设置优先于在配置时使用 configure 脚本开关选项指定的用户。如果没有做任何指定，工作进程将以用户 nobody 开始，用户组为 nobody 组(或 nogroup，具体取决于操作系统)。

Nginx 命令行开关项

Nginx 二进制文件接收命令行参数，用于执行各种操作，控制后台进程。为了获取该命令的全部参数列表，可以请求 help 帮助，使用下面的命令即可：

```
[alex@example.com ~]$ cd /usr/local/nginx/sbin
[alex@example.com sbin]$ ./nginx -h
```



下面将描述这些开关项的作用，一些用于控制守护进程，一些用于在应用配置上执行各种操作。

启动和停止守护进程

可以通过不带任何参数的 Nginx 二进制文件来启动 Nginx。如果该守护进程已经运行，就会有一条消息指出已经有一个套接字在指定端口监听：

```
[emerg]: bind() to 0.0.0.0:80 failed (98: Address already in
use) […]
[emerg]: still could not bind().
```

除了这一点，你可以控制这个守护进程，可以停止它，重启它，或只是重新载入配置文件。控制是通过 nginx-s 命令向进程发送信号来实现的(详见下表)。

命　令	描　述
nginx –s stop	立即停止守护进程(使用 TERM 信号)
nginx –s quit	温和地停止守护进程 (使用 QUIT 信号)
nginx –s reopen	重新打开日志文件
nginx –s reload	重新载入配置文件

注意，在开始运行这个守护进程、停止它或执行前面说的任何操作时，会首先解析和确认配置文件，如果配置文件无效，不管提交什么命令，都会失败，甚至是试图停止一个守护进程。换句话说，如果配置文件无效，兴许无法停止 Nginx 服务。

有一种替代方法可以终止该进程，只适用于危急情况，即使用 kill 或 killall 命令：

```
[alex@example.com ~]$ killall nginx
```

测试配置文件

可以想象，如果不断调整配置，这一点点的细节可能成为一个重要的问题。在配置文件中，任何不起眼的错误都可能导致你丧失对服务的控制权——可能无法以正常的方式停止服务，显而易见，服务拒绝运行。

因此，在很多情况下，下列命令是很有用的，可以检测语法、合法性和配置文件的完整性：

```
[alex@example.com ~]$ /usr/local/nginx/sbin/nginx -t
```

开关选项-t 代表测试配置文件。Nginx 将重新解析配置文件，让你知道配置文件是否有效。下图显示了一个无效的配置，测试的结果是一个失败的测试。

一个有效的配置文件并不意味着 Nginx 必然能够启动，然而，可能还有额外的问题，例如套接字问题、无效的路径或不正确的访问权限。

显而易见，必须能够熟练控制配置文件。生产环境中的服务器，这样做是很危险的，要不惜任何代价避免这种情况。既然这样，一个最好的实践是将新配置文件放在一个单独的临时文件中，然后再对该文件进行测试。Nginx 提供了一个-c 选项，便可进行这种测试：

```
[alex@example.com sbin]$ ./nginx -t -c /home/alex/test.conf
```

该命令将解析配置文件/home/alex/test.conf，确定它作为 Nginx 配置文件的有效性。此后，再确定新的有效配置文件，然后重新载入服务器配置文件：

```
[alex@example.com sbin]$ cp /home/alex/test.conf
/usr/local/nginx/conf/nginx.conf
cp: erase 'nginx.conf' ? yes
[alex@example.com sbin]$ ./nginx -s reload
```

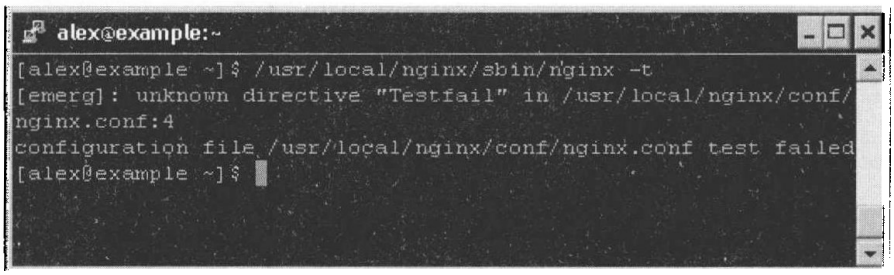
其他开关选项

其他开关选项在许多情况下迟早都会用到：-V 选项，不但会告诉你当前 Nginx 的版本号，更重要的是还会提醒你在 configure 脚本那一步所添加的开关选项，换句话说，该开关选项将显示你在使用运行 configure 脚本配置的各种开关选项(如下图所示)：

alex@example:~

```
[alex@example ~]$ /usr/local/nginx/sbin/nginx -t
[emerg]: unknown directive "Testfail" in /usr/local/nginx/conf/
nginx.conf:4
configuration file /usr/local/nginx/conf/nginx.conf test failed
[alex@example ~]$
```

开关选项-t 代表测试配置文件。Nginx 将重新解析配置文件，让你知道配置文件是否有效。下图显示了一个无效的配置，测试的结果是一个失败的测试。



一个有效的配置文件并不意味着 Nginx 必然能够启动，然而，可能还有额外的问题，例如套接字问题、无效的路径或不正确的访问权限。

显而易见，必须能够熟练控制配置文件。生产环境中的服务器，这样做是很危险的，要不惜任何代价避免这种情况。既然这样，一个最好的实践是将新配置文件放在一个单独的临时文件中，然后再对该文件进行测试。Nginx 提供了一个-c 选项，便可进行这种测试：

```
[alex@example.com sbin]$ ./nginx -t -c /home/alex/test.conf
```

该命令将解析配置文件/home/alex/test.conf，确定它作为 Nginx 配置文件的有效性。此后，再确定新的有效配置文件，然后重新载入服务器配置文件：

```
[alex@example.com sbin]$ cp /home/alex/test.conf
/usr/local/nginx/conf/nginx.conf
cp: erase 'nginx.conf' ? yes
[alex@example.com sbin]$ ./nginx -s reload
```

其他开关选项

其他开关选项在许多情况下迟早都会用到：-V 选项，不但会告诉你当前 Nginx 的版本号，更重要的是还会提醒你在 configure 脚本那一步所添加的开关选项，换句话说，该开关选项将显示你在使用运行 configure 脚本配置的各种开关选项(如下图所示)：

可以看出，Nginx 在安装配置时只使用了开关选项--with-http_ssl_module。为什么它如此重要呢？如果试图使用一个在 configure 脚本安装配置时没有包含的模块，那么如果开启了这类模块的指令，会导致配置文件中发生错误(这时会觉得前面的说法很有意义)。你的第一个反应是对来自语法错误的错误报告感到困惑，你的第二个反应是你是否建立了该模块！运行一下“nginx -V”命令即可回答这个问题。

此外，-g 选项用于指定额外的配置指令，即使配置文件中并没有包含：

```
[alex@example.com sbin]$ ./nginx -g "timer_resolution 200ms";
```

添加 Nginx 作为系统服务

在这一小节，我们将创建一个脚本，该脚本将改变 Nginx 守护进程，让 Nginx 以系统服务的形式启动。这将获得的主要有两个成果：该守护进程将由标准的命令控制，更重要的是，它会在系统启动时自动启动。

System V 脚本

发展到今天，大多数基于 Linux 的操作系统，使用的是 System-V 风格的 init 守护进程，换句话说，它们的启动处理由 init 进程管理，其管理功能在一定程度上继承了基于 System V 的 Unix 操作系统。

该守护进程根据运行级别(run level)的原则，系统的运行级别表示当前计算机的状态。下表描述了各种不同运行级别及其含义。

0	系统停止
1	单用户模式 (援救模式)
2	多用户模式，不支持 NFS
3	完整的多用户模式

alex@localhost:~

```
[alex@localhost ~]$ /usr/local/nginx/sbin/nginx -V
nginx version: nginx/0.7.66
built by gcc 4.1.2 20080704 (Red Hat 4.1.2-44)
TLS SNI support disabled
configure arguments: --with-http_ssl_module
[alex@localhost ~]$
```

运行级别	状态

```
alex@localhost:~
[alex@localhost ~]$ /usr/local/nginx/sbin/nginx -V
nginx version: nginx/0.7.66
built by gcc 4.1.2 20080704 (Red Hat 4.1.2-44)
TLS SNI support disabled
configure arguments: --with-http_ssl_module
[alex@localhost ~]$
```

可以看出，Nginx 在安装配置时只使用了开关选项--with-http_ssl_module。为什么它如此重要呢？如果试图使用一个在 configure 脚本安装配置时没有包含的模块，那么如果开启了这类模块的指令，会导致配置文件中发生错误(这时会觉得前面的说法很有意义)。你的第一个反应是对来自语法错误的错误报告感到困惑，你的第二个反应是你是否建立了该模块！运行一下“nginx -V”命令即可回答这个问题。

此外，-g 选项用于指定额外的配置指令，即使配置文件中并没有包含：

```
[alex@example.com sbin]$ ./nginx -g "timer_resolution 200ms";
```

添加 Nginx 作为系统服务

在这一小节，我们将创建一个脚本，该脚本将改变 Nginx 守护进程，让 Nginx 以系统服务的形式启动。这将获得的主要有两个成果：该守护进程将由标准的命令控制，更重要的是，它会在系统启动时自动启动。

System V 脚本

发展到今天，大多数基于 Linux 的操作系统，使用的是 System-V 风格的 init 守护进程，换句话说，它们的启动处理由 init 进程管理，其管理功能在一定程度上继承了基于 System V 的 Unix 操作系统。

该守护进程根据运行级别(run level)的原则，系统的运行级别表示当前计算机的状态。下表描述了各种不同运行级别及其含义。

运行级别	状　态
0	系统停止
1	单用户模式 (援救模式)
2	多用户模式，不支持 NFS
3	完整的多用户模式

续表

运行级别	状　态
4	没有使用
5	图形界面模式
6	重启系统

可以手动运行一个 run level：使用“telinit 0”可以关闭计算机，使用“telinit 6”重启。

对于每一个运行级别(run level)的转换，都会有一组服务被执行，这是一个关键的概念。注意：系统停止时，它的运行级别为 0；一旦打开计算机，就会从运行级别 0 转换到计算机默认启动级别。系统默认启动级别是在系统中（在/etc/inittab 文件）配置的，并且默认启动级别也依赖于你所使用的发布版本：Debian 和 Ubuntu 使用的是运行级别 2；Red Hat 和 Fedora 使用的是 3 或 5；CentOS 和 Gentoo 使用的是 3，等等。

让我们总结一下，开启运行 CentOS 的计算机时，它会从运行级别 0 转换到 3，这个转换包含启动所有计划运行在运行级别为 3 的服务。这里会有一个问题，如何将一个服务安排到指定运行级别？对于每一个运行级别，在某一个目录中都包含要执行的脚本(见下图)。

/etc/rc.d

Name	Ext	Size	Changed	Rights	Owner
..			12/29/2009 4:36:09 PM	rwxr-xr-x	root
init.d			10/18/2009 3:31:43 PM	rwxr-xr-x	root
rc0.d			1/1/2010 6:55:49 PM	rwxr-xr-x	root
rc1.d			1/1/2010 6:55:49 PM	rwxr-xr-x	root
rc2.d			1/1/2010 6:55:49 PM	rwxr-xr-x	root
rc3.d			1/1/2010 6:55:49 PM	rwxr-xr-x	root
rc4.d			1/1/2010 6:55:49 PM	rwxr-xr-x	root
rc5.d			1/1/2010 6:55:49 PM	rwxr-xr-x	root
rc6.d			1/1/2010 6:55:49 PM	rwxr-xr-x	root

如果进入这些目录(rc0.d, rc1.d, to rc6.d)，你找不到任何真实的文件，相反的，它们的符号连接会指向存放在 init.d 目录下的脚本名。服务的启动脚本确实存储在 init.d 目录下，而是被链接通过工具放在适当的运行级别目录下。

什么是 init 脚本?

init 脚本，是众所周知的作为启动服务的脚本，sysv 脚本是一个基于某一标准的脚本，该脚本将通过一些命令来控制一个应用程序的 start、stop 或者其他操作，这将触发两个级别。首先，在计算机启动的时候，如果该服务的启动计划级别正是系统运行的级别，那么 init 守护进程将运行该脚本(带有启动参数)；对于你来说，另一种可能就是手动执行脚本，即是通过 shell 调用该脚本来执行，这种可能性已经在第 1 章介绍过：

```
[root@example.com ~]# service httpd start
```

或者如果系统没有提供 service 命令：

```
[root@example.com ~]# /etc/init.d/httpd start
```

脚本必须接受至少两个命令：start 和 stop，分别用于启动和停止该服务。然而，作为一个系统管理员，为了扩大操作行为领域，经常提供更多的功能选项会很有趣，例如提供一个 reload 参数以便重新载入服务的配置文件，或者是提供一个 restart 参数，先 stop 再 start 服务。

注意，既然“service httpd start”和“/etc/init.d/httpd start”其实在做同一件事，而第二个命令适用于所有的操作系统，我们将不再进一步提及 service 命令，而是专门使用/etc/init.d/方法来控制 Nginx。

为 Nginx 建立 init 脚本

以此方式我们将创建一个用于启动和停止 Nginx 守护进程的 shell 脚本，同时还要加入重启动和重新载入配置文件功能。这里的目的不是讨论 Linux 的 shell 脚本编程，因此我们只提供一个写好的 init 源代码脚本，除了脚本命令就是一些注释，以帮助理解脚本。

首先，选择一种文本编辑器建立一个名为 nginx 的文件，并且保存在目录/etc/init.d/的下面(在其他一些系统下，/etc/init.d/实际上是/etc/rc.d/init.d/符号链接)。仔细将下列脚本复制到新建文件中，请确保更改路径，使其符合实际安装位置。

保存该脚本到 init.d 目录下需要 root 权限。

```
#! /bin/sh
# Author: Ryan Norbauer http://norbauerinc.com
# Modified: Geoffrey Grosenbach http://topfunky.com
# Modified: Clement NEDELCU
# Reproduced with express authorization from its contributors
set -e
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DESC="nginx daemon"
NAME=nginx
DAEMON=/usr/local/nginx/sbin/$NAME
SCRIPTNAME=/etc/init.d/$NAME

# If the daemon file is not found, terminate the script.
test -x $DAEMON || exit 0

d_start() {
   $DAEMON || echo -n " already running"
}

d_stop() {
   $DAEMON -s quit || echo -n " not running"
}
d_reload() {
   $DAEMON -s reload || echo -n " could not reload"
}

case "$1" in
start)
   echo -n "Starting $DESC: $NAME"
       d_start
       echo "."
;;
stop)
   echo -n "Stopping $DESC: $NAME"
       d_stop
       echo "."
;;
reload)
   echo -n "Reloading $DESC configuration..."
       d_reload
   echo "reloaded."
```

```
;;
restart)
    echo -n "Restarting $DESC: $NAME"
        d_stop
# Sleep for two seconds before starting again, this should give
the
# Nginx daemon some time to perform a graceful stop.
        sleep 2
        d_start
        echo "."
;;
*)
        echo "Usage: $SCRIPTNAME {start|stop|restart|reload}" >&2
        exit 3
;;
esac

exit 0
```

更多信息请参见书后译者注 2。

安装 Nginx 的 init 脚本

将该脚本文件放在 init.d 目录下，并不意味着工作完成，还有其他步骤需要执行，然后才能开启该服务(脚本所启动的服务，这里当然指的是 Nginx)。首先，需要确定脚本可执行，到现在为止，我们的脚本文件还只是一个系统拒绝运行的文本文件，通过下面的 chmod 命令来授予该脚本可执行权限：

```
[root@example.com ~]# chmod +x /etc/init.d/nginx
```

注意，如果新建的文件是以 root 用户的权限建立的，则需要以 root 身份登录后才可以修改文件的权限。

到现在，你已经能够使用“service nginx start”或“/etc/init.d/nginx start”命令来启动服务了，也包括停止、重新启动或重新载入服务器配置文件。最后一步是让该脚本能够在适当的运行级自动启动。遗憾的是，要这么做，只能完全依赖于操作系统，我们覆盖了最广泛的两种操作系统家族：一是基于 Debian 的 Debian/Ubuntu/other 家族；二是源于 Red Hat 的 Red Hat/Fedora/CentOS/other 家族。

基于 Debian 的发布

对于第一种情况，通过一个简单的命令就能完成系统级别初始化脚本：

```
[root@example.com ~]# update-rc.d -f nginx defaults
```

该命令将在默认系统运行级别上建立链接，默认级别是指重启(reboot)和关闭(shutdown)级别，对于这两种级别，该脚本会执行 stop 参数，对于其他级别，则传递 start 参数而运行。现在可以重启系统，并且在启动列表中看到 Nginx 服务正在启动。

基于 Red Hat 的发布

基于 Red Hat 系统的家族，命令不同，但是你可以获得其他工具来管理系统的启动，可以通过下面的命令来添加服务：

```
[root@example.com ~]# chkconfig --add nginx
```

一旦执行完该命令，就可以检验该服务的运行级别：

```
[root@example.com ~]# chkconfig --list nginx
Nginx 0:off 1:off 2:on 3:off 4:on 5:on 6:off
```

另一个用来管理系统服务的有用工具是 ntsysv，它列出了所有能够计划在系统启动时执行的服务，可以开启它们，也可以禁用它们。

注意，必须首先运行“chkconfig --add nginx”命令，否则 nginx 不会出现在上图所示的服务列表中。

alex@localhost:/

```
ntsysv 1.3.â                                                          â
           â What services should be automatically started?           â
           â                                                          â
           â              [ ] nfs                 â                   â
           â              [*] nfslock             â                   â
           â              [*] nginx               â                   â
           â              [ ] nscd                ââ                  â
           â              [ ] ntalk               â                   â
           â              [ ] ntpd                â                   â
           â              [ ] pand                â                   â
           â              [*] pcscd               â                   â
           â                                                          â
           â        ááááááá                ááááááááááá              â
           â        á Ok á                  á Cancel á                â
           â        ááááááá                ááááááááááá              â
Press <F1> â                                                          â
```

基于 Debian 的发布

对于第一种情况，通过一个简单的命令就能完成系统级别初始化脚本：

```
[root@example.com ~]# update-rc.d -f nginx defaults
```

该命令将在默认系统运行级别上建立链接，默认级别是指重启(reboot)和关闭(shutdown)级别，对于这两种级别，该脚本会执行 stop 参数，对于其他级别，则传递 start 参数而运行。现在可以重启系统，并且在启动列表中看到 Nginx 服务正在启动。

基于 Red Hat 的发布

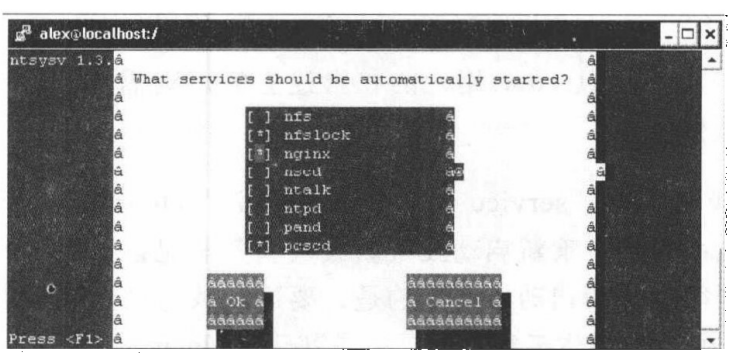
基于 Red Hat 系统的家族，命令不同，但是你可以获得其他工具来管理系统的启动，可以通过下面的命令来添加服务：

```
[root@example.com ~]# chkconfig --add nginx
```

一旦执行完该命令，就可以检验该服务的运行级别：

```
[root@example.com ~]# chkconfig --list nginx
Nginx 0:off 1:off 2:on 3:off 4:on 5:on 6:off
```

另一个用来管理系统服务的有用工具是 ntsysv，它列出了所有能够计划在系统启动时执行的服务，可以开启它们，也可以禁用它们。



注意，必须首先运行“chkconfig --add nginx”命令，否则 nginx 不会出现在上图所示的服务列表中。

小结

本章涉及许多重要的知识点，首先是编译 Nginx 之前需要做的所有事情，然后是帮助我们选择使用适当的版本——是使用稳定版还是使用一个先进但可能不稳定的版本？之后，我们下载源代码并且根据需要配置编译它，例如开启或者是禁用一些功能和模块，例如，SSL，GeoIP，等等，这一步之后，源代码被编译并且将应用程序安装到你在系统中指定的目录，我们创建了一个初始化脚本，并且修改了系统引导顺序来安排服务的启动。从这时起，Nginx 已经安装在你的服务器中，并在系统启动时自动启动。你的 Web 服务器可以使用了，尽管它不能满足最基本的功能——提供一个网站。在 Web 服务器上建立网站的第一步是建立一个配置文件，下一章将概述 Nginx 的基本配置，并且教你如何基于未来的用户和系统资源来优化 Nginx 性能。

第 3 章

Nginx 的基本配置

本章将开始为 Web 服务器建立一个适当的配置。为此，我们首先需要着手处理配置文件的语法问题。我们首先需要理解配置文件中的各种指令，然后通过这些指令可以优化服务器，针对不同的流量模式和硬件设置。最后建立一些测试页，用来确定每一件事都正确完成，即配置文件是有效的。本章只处理基本配置指令，下一章将详细讨论高级话题，例如，HTTP 模块配置和用法，建立虚拟主机，等等。

本章主题

- 配置文件的语法
- 基本的配置指令
- 建立一个适合自己的配置
- 测试 Web 服务
- 测试和维护服务器

配置文件的语法

配置文件一般是一个文本文件，可以由管理员编辑，当然也可以由程序解析。通过为一组指令指定值，便定义了程序的行为，基于 Linux 的操作系统，大部分应用程序都是基于一个大的、复杂的配置文件，这往往会使管理变成噩梦。Apache, PHP, MySQL, Qmail 和 Bind——所有这些名称都意味着不好的回忆，但没办法，所有这些应用程序都有自己的配置文件，并且各有自己的语法和格式这已成为一个事实。PHP 使用 Windows 格式的.ini 文件，sendmail 使用 M4 宏处理器来编译配置文件，Zabbix 能够从 MySQL 数据库抽取配置，等等。遗憾的是，没有建立一个标准！同样，你将要使用 Nginx，需要学习它的新语法，它又有新的特点，有自己的词汇。

另一方面(这是它的优势之一)，配置 Nginx 原来是如此简单，相比之下，至少比

Apache 或者其他主流的 Web 服务器简单，仅需要掌握一些机制——指令、区段(block)和整体逻辑结构，实际配置过程中，大部分都是为指令填写值。

配置指令

Nginx 的配置文件被描述为一个具有一定逻辑结构的一组指令列表。应用程序的整个行为通过修改这些指令的值即可实现。

默认情况下，Nginx 使用了一个主要的配置文件，定义该文件的步骤在第 2 章下载和安装的“建立配置”小节中有描述，如果你没有编辑配置文件的路径和 prefix 选项，那么配置文件在系统中的位置应该是“/usr/local/nginx/conf/nginx.conf”。现在让我们快速看一下最前面的几行(如下图所示)。

仔细看看前两行：

```
#user nobody;
worker_processes 1;
```

正如你所知，字符“#”意味着第一行是注释行，换句话说，就是一块 Nginx 将忽略的文本，不管该指令是否有值，其唯一目的是要通过那个用户来打开并读取文件。可以在一行的开始使用“#”，也可以在“#”之后跟随指令。

alex@extremejeux: ~/src/nginx-0.7.66/conf

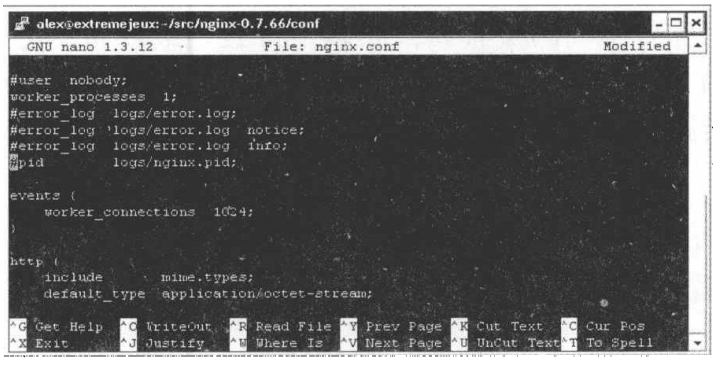
```
  GNU nano 1.3.12          File: nginx.conf                    Modified

#user  nobody;
worker_processes  1;
#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;
#pid        logs/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Apache 或者其他主流的 Web 服务器简单，仅需要掌握一些机制——指令、区段(block)和整体逻辑结构，实际配置过程中，大部分都是为指令填写值。

配置指令

Nginx 的配置文件被描述为一个具有一定逻辑结构的一组指令列表。应用程序的整个行为通过修改这些指令的值即可实现。

默认情况下，Nginx 使用了一个主要的配置文件，定义该文件的步骤在第 2 章下载和安装的“建立配置”小节中有描述，如果你没有编辑配置文件的路径和 prefix 选项，那么配置文件在系统中的位置应该是“/usr/local/nginx/conf/nginx.conf”。现在让我们快速看一下最前面的几行(如下图所示)。



仔细看看前两行：

```
#user nobody;
worker_processes 1;
```

正如你所知，字符“#”意味着第一行是注释行，换句话说，就是一块 Nginx 将忽略的文本，不管该指令是否有值，其唯一目的是要通过那个用户来打开并读取文件。可以在一行的开始使用“#”，也可以在“#”之后跟随指令。

第二行是一个实际的声明——一条指令，第一位(worker_processes)是一个设置键，你可以附加一个或多个参数，在这里，参数是 1，它标志着 Nginx 作为单个工作进程(关于该指令更多的功能介绍参见后文)。

指令总是以分号结尾(;)。

每一条指令都有它特别的功能，Nginx 通过定义这些指令来实现特定的功能。另外，每个指令也有不同的语法，例如，指令 worker_process 只接受一个数字值，而指令 user 会让你指定两个字符串值——一个是用户帐户(Nginx 的 worker 进程便使用该用户运行)，第二个是用户组。

Nginx 通过模块的方法来构建而成，因此，每一个模块都提供一组指令。最根本的指令是 Nginx 核心模块部分，将在本章详细介绍。至于其他模块的其他指令，我们将在后面进行探讨。

组织和包含

在前面的截图中，你肯定已经注意到一个特别的指令——include：

```
include mime.types;
```

顾名思义，该指令用来执行一个对特定文件的包含，换句话说，就是在该配置文件的内容中，将被插入的文件通过 include 指令插入到这个确定(确切)的位置上。这里有一个实际的例子可以帮助我们理解：

nginx.conf：

```
user nginx nginx;
worker_processes 4;
include other_settings.conf;
```

other_settings.conf：

```
error_log logs/error.log;
pid logs/nginx.pid;
```

以下是 Nginx 解释配置文件的最终结果：

```
user nginx nginx;
worker_processes 4;
error_log logs/error.log;
pid logs/nginx.pid;
```

前面例子中包括递归处理，在这种情况下，使用 include 指令，在主配置文件中通过该指令将文件 other_settings.conf 包括进来，而在该文件中又包含其他的文件。在最初的配置文件设置中，使用了两个文件——nginx.conf 和 mime.type。然而，对于高级配置来说，至少有五个文件，如下表所示。

标准名称	描　述
nginx.conf	应用程序的基本配置文件
mime.types	一个文件扩展列表文件，它们与 MIME 类型关联
fastcgi.conf	与 FastCGI 相关的配置文件
proxy.conf	与 Proxy 相关的配置文件
sites.conf	配置 Nginx 提供的网站，也包括众所周知的虚拟主机。推荐每一个域建立一个单独的文件

这些文件名是按照惯例来定义的，实际上，完全可以使用其他文件名，例如，可以将 FastCGI 和 Proxy 重新组合到有一个普通的文件中，文件名可以为 proxy_and_fastcgi_config.conf。

注意，include 指令支持文件名替换，换句话说，使用通配符“*”的文件名，这里的“*”可以匹配零个、一个或者多个连续的字符[①]：

```
include sites/*.conf;
```

这将包含 sites 目录下扩展名为.conf 的文件，这种机制允许你为自己的每一个网站建立一个单独的文件，然后就像使用上面的指令一样，通过一条指令一次性将它们全部包括进来。

在包含一个文件时要仔细点——如果指定的文件不存在，配置检测会失败，Nginx 将无法启动：

```
[alex@example sbin]# ./nginx -t
[emerg]:open()"/usr/local/nginx/conf/dummyfile.conf"failed (2:No
such file or directory) in /usr/local/nginx/conf/nginx.conf:48
```

很明显，先前的配置不是很合适，由于在 include 指令中使用了通配符，从而导致包含不存在的文件。

然而，如果在配置文件插入“include dummy*.conf”，然后再测试它(看一下这种格式在你的系统上是否有配置)，看一下会有什么情况发生：

```
[alex@example sbin]# ./nginx -t
the configuration file/usr/local/nginx/conf/nginx.conf syntax is ok
configuration file/usr/local/nginx/conf/nginx.conf test is successful
```

---

① 注意，结尾一定要有分号“;”,否则会出现莫名其妙的报错，如：

```
[emerg]: unexpected "}" in /usr/local/nginx0.8/conf/nginx.conf:28
```

或者

```
[emerg]: directive "include" is not terminated by ";" in
/usr/local/nginx0.8/conf/ nginx.conf:7
```

等等。

指令块

指令由模块提供——如果你激活一个新模块，便会有一组特定的指令变为有效。模块也可以使指令块(block)[1]可用，例如，下面这个是配置文件中的一个逻辑框架：

```
events {
    worker_connections 1024;
}
```

这个 events 区段(block)可以在默认的配置文件中找到，它来自于 events 模块，该模块提供的指令也只能在这个区段中使用，例如前面的例子，worker_connections 只能放在 events 区段才有意义。然而有一个重要的例外——一些文件可以放在配置文件的根部(最开始的部分)，因为它可以给服务器一个全局的效果。配置文件的根部也被认为是主要的区段。

本章将详述核心模块中的块和指令，模块是服务器顺利运作的根本。可选模块(它们在默认情况下是否启用)将在后面的章节中讨论。

注意，在某些情况下，不同区段能够互相嵌套，例如：

```
http {
    server {
        listen 80;
        server_name example.com;
        access_log /var/log/nginx/example.com.log;
        location ^~ /admin/ {
        index index.php;
        }
    }
}
```

这个例子展示了如何在网站配置 Nginx，以便于辨别 http 区段(而不是，比如 imap，如果想利用邮件服务器的代理功能)。

在这个 http 区段，可以声明一个或多个 server 区段，一个 server 区段允许配置一个虚拟主机。在这个例子中，server 区段包含一些配置，这些设置通过一个主机 HTTP 头(Host HTTP header)应用于所有的请求以便于正确配置 example.com。

---

① block，区段。在本书的翻译中，“块”和“区段”混合使用，有的时候觉得“块”合适，而有时有觉的是“区段”合适，但表达的是同样的内容。

在这个 server 区段内，可以插入一个或多个 location 区段，当你需要对特定的路径进行 URI 匹配时，这些 location 区段允许你对这些路径单独设置。详情请参见第 4 章。

最后同样重要的是配置的继承。在一个区段中嵌套其他区段，那么被嵌套的区段会继承其父区段的设置。access_log 指令(在这个例子中，在 server 区段级别定义)指定对这个服务器的所有 HTTP 请求都会记录在这个文本文件中。

这个条件在 location 子区段仍然成立，但可以重新设置 access_log 指令来禁用继承：

```
[…]
    location ^~ /admin/ {
        index index.php;
        access_log off;
    }
[…]
```

这样，日志虽然不能够记录 /admin/，但它仍可以记录该网站所有的访问日志。对 access_log 设置的值，在 server 级别的区段设置的值将由 location 级别的区段设置的值所覆盖。

高级语言规则

Nginx 配置文件的语法有助于理解某些语法规则，如果以前没有在 Nginx 下工作过，可能会觉得难以理解。

接受特有语法的指令

突然看到这种复杂语法，可能会觉得难以理解：

```
rewrite^/(.*)\.(png|jpg|gif)$/image.php?  file=$1&format=$2  last;
```

指令的语法往往会详细指出指令的用法。listen 指令只能够接受一个端口号，用来打开一个监听套接字，location 块或 rewrite 指令支持复杂的表达式，是为了配置特殊模式。所有这些语法，都将在各自的章节中通过具体的指令一并解释。

在后面我们将学习一个模块(即 Rewrite 模块)，它允许你在配置文件中使用更高级的逻辑结构，通过使用 if，set，break 和 return 指令以及各种变量来选择处理客户端的请求。通过使用这些新的元素，配置文件开始变得像编程脚本。不管怎么说，我们认识的模块越多，语法也会越丰富。

指令值的单位

最后，可以使用下列单位来指定配置文件所在环境中所用指令值的单位。

- k 或 K: 千字节
- m 或 M: 兆字节

因此，下面两种语法是正确的，同时也是相等的：

```
client_max_body_size 2M;
client_max_body_size 2048k;
```

下面是可以指定的时间值，可以使用其缩写形式：

- ms: Milliseconds(毫秒)
- s: Seconds(秒)
- m: Minutes(分钟)
- h: Hours(小时)
- d: Days(天)
- w: Weeks(星期)
- M: Months (30 天)(月)
- y: Years (365 天)(年)

遇到用一个时间段作为指示值的情况，将特别有用：

```
client_body_timeout 3m;
client_body_timeout 180s;
client_body_timeout 180;
```

注意，默认的时间单位是秒，在前面的例子中，最后两行的结果一样。

变量

模块提供各种变量，变量用于指定变量值，例如，Nginx 的 HTTP 核心模块定义了 $nginx_version 变量。在设置 log_format 指令时，在下面的格式中，可以包含各种各样的变量：

```
[…]
location ^~ /admin/ {
    access_log logs/main.log;
    log_format main '$pid - $nginx_version - $remote_addr';
}
[…]
```

注意，一些指令不允许使用任何变量：

```
error_log logs/error-$nginx_version.log;
```

这是一个有效的配置指令，但它只产生一个名为“error-$nginx_version.log”的文件，并不解析变量。

字符串值

将字符串用作指令值，可以有三种格式，首先，可以没有引号：

```
root /home/example.com/www;
```

然而，如果使用的是特殊字符，例如，空格符(" ")、分号(;)或者是花括号({和})，就需要使用单引号或双引号将其括起：

```
root '/home/example.com/my web pages';
```

无论使用单引号或双引号，Nginx 都认为没有什么区别。

基本模块指令

在本小节中，我们将需要仔细看看基本模块。我们尤其感兴趣的是回答两个问题——什么是基本模块和有哪些可用指令。

什么是基本模块?

基本模块提供的指令允许你定义 Nginx 基本功能的变量，在编译时它们不能被禁用，因此它们提供的指令和区段总是有效。有下面三个基本模块。

- **核心模块(Core module)**——基本特征和指令，例如进程管理和安全。
- **事件模块(Events module)**——可以让你在 Nginx 内部机制配置网络使用能力。
- **配置模块(Configuration module)**——提供包含机制。

这些模块提供了很多指令，我们将对这些命令中个别语法和默认值加以详细阐述。

Nginx 进程结构

开始详细讲述基本配置指令之前，有必要理解一下 Ngnix 的进程结构，即 Nginx 在后台如何工作。虽然应用程序只是一个二进制文件(显然是轻量级的后台进程)，但是在运行时功能却相当复杂。

alex@example: /home/alex

```
[alex@example ~]$ ps fuax | grep nginx
root     15645  0.0  0.1  3940  508 ?        Ss   04:13   0:00 nginx: master process /us
nginx    15646  0.0  0.4  4280 1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15647  0.0  0.4  4280 1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15648  0.0  0.3  4200 1000 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15649  0.0  0.4  4280 1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15650  0.0  0.4  4280 1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15651  0.0  0.4  4280 1028 ?        S    04:13   0:00  \_ nginx: worker process
```

什么是基本模块?

基本模块提供的指令允许你定义 Nginx 基本功能的变量，在编译时它们不能被禁用，因此它们提供的指令和区段总是有效。有下面三个基本模块。

- **核心模块(Core module)**——基本特征和指令，例如进程管理和安全。
- **事件模块(Events module)**——可以让你在 Nginx 内部机制配置网络使用能力。
- **配置模块(Configuration module)**——提供包含机制。

这些模块提供了很多指令，我们将对这些命令中个别语法和默认值加以详细阐述。

Nginx 进程结构

开始详细讲述基本配置指令之前，有必要理解一下 Ngnix 的进程结构，即 Nginx 在后台如何工作。虽然应用程序只是一个二进制文件(显然是轻量级的后台进程)，但是在运行时功能却相当复杂。

```
alex@example: /home/alex
[alex@example ~]$ ps fuax | grep nginx
root     15645  0.0  0.1   3940   508 ?        Ss   04:13   0:00 nginx: master process /us
nginx    15646  0.0  0.4   4280  1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15647  0.0  0.4   4280  1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15648  0.0  0.4   4280  1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15649  0.0  0.4   4280  1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15650  0.0  0.4   4280  1032 ?        S    04:13   0:00  \_ nginx: worker process
nginx    15651  0.0  0.4   4280  1028 ?        S    04:13   0:00  \_ nginx: worker process
```

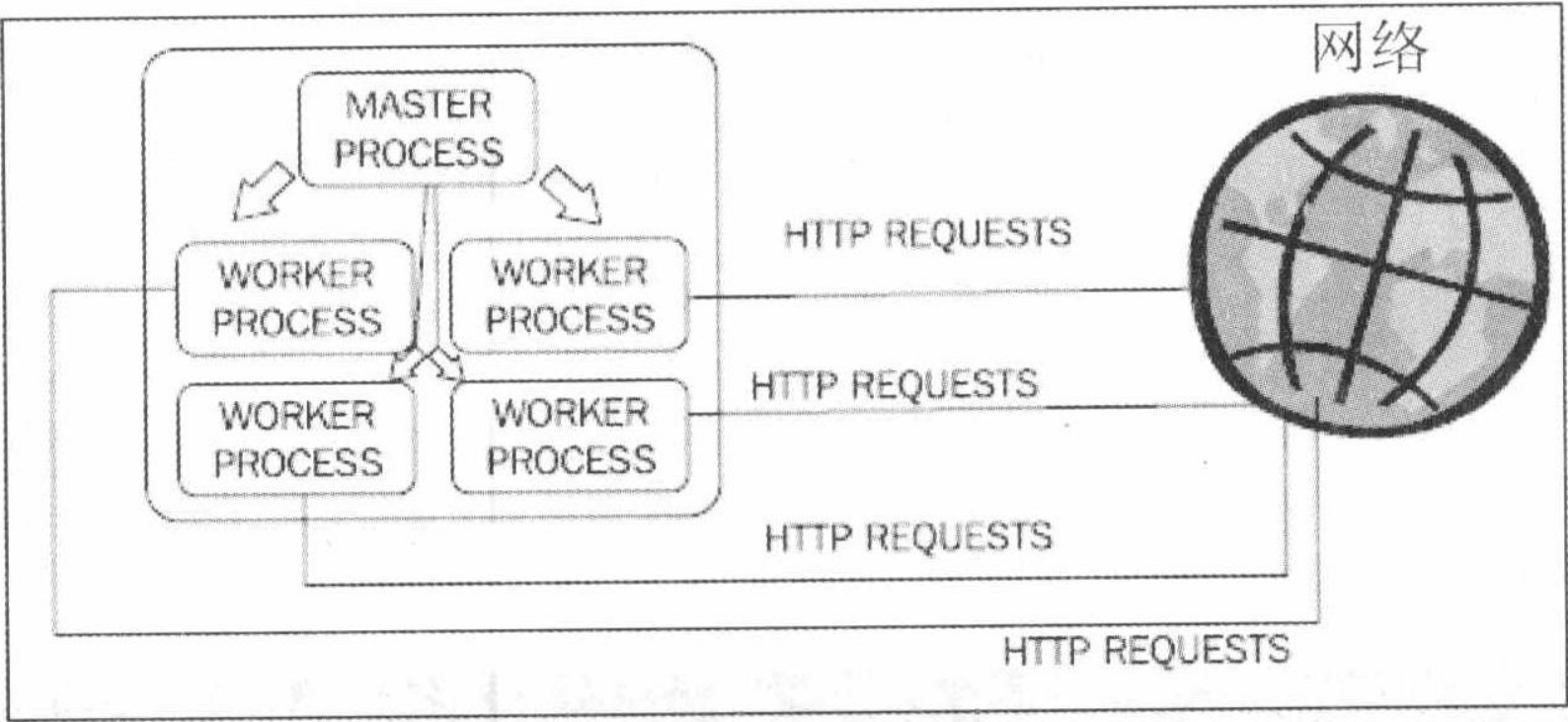
现在启动 Nginx，一个独一无二的进程存在于内存——**master 进程**，如果该服务在系统启动时有 init 脚本启动，那么它会使用当前的用户和用户组来运行——通常为 root/root，master 进程本身不处理任何客户端的请求，而它的功能是用来产生进程——worker 进程，这些 worker 进程以在配置文件中指定的用户和用户组来运行，可以定义大量 worker 进程以及每个 worker 进程的最大连接数。

核心模块指令

下表列出的指令在核心(core)模块中有效，大部分指令必须放在配置文件的根部[①]，而且只能使用一次。然而，有些指令在多种情况下有效，表中会有相应说明，指明该指令会在哪些环境中可用，在配置文件的根部只能使用一次。

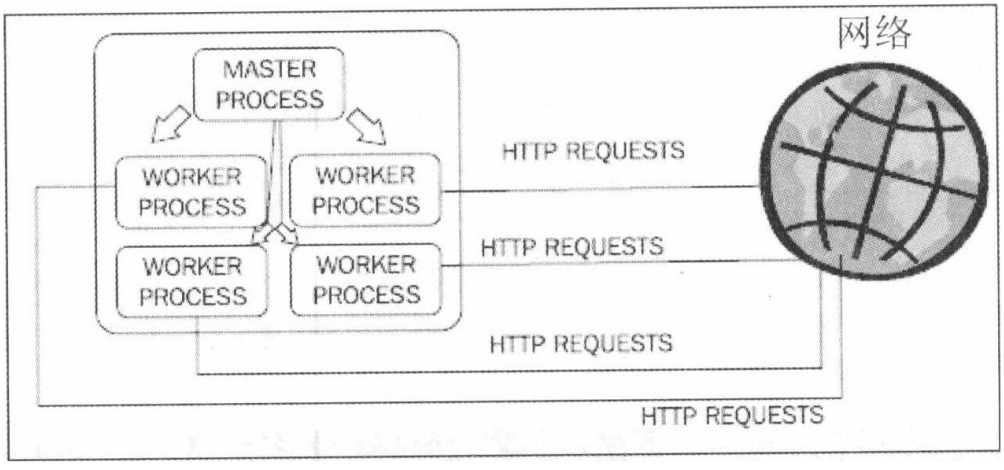
daemon	值：on 或 off 语法：<daemon on;> 默认值：on 启用或禁用守护进程模式。如果禁用它，该程序就不能在后台启动，当它在 shell 下运行时，将留在前端运行

① 即配置文件的开始部分。


网络
MASTER PROCESS
WORKER PROCESS
WORKER PROCESS
WORKER PROCESS
WORKER PROCESS
HTTP REQUESTS
HTTP REQUESTS
HTTP REQUESTS
HTTP REQUESTS

指令和使用环境	语法和描述

现在启动 Nginx，一个独一无二的进程存在于内存——**master 进程**，如果该服务在系统启动时有 init 脚本启动，那么它会使用当前的用户和用户组来运行——通常为 root/root，master 进程本身不处理任何客户端的请求，而它的功能是用来产生进程——worker 进程，这些 worker 进程以在配置文件中指定的用户和用户组来运行，可以定义大量 worker 进程以及每个 worker 进程的最大连接数。



核心模块指令

下表列出的指令在核心(core)模块中有效，大部分指令必须放在配置文件的根部[①]，而且只能使用一次。然而，有些指令在多种情况下有效，表中会有相应说明，指明该指令会在哪些环境中可用，在配置文件的根部只能使用一次。

指令和使用环境	语法和描述
daemon	值：on 或 off 语法：<daemon on;> 默认值：on 启用或禁用守护进程模式。如果禁用它，该程序就不能在后台启动，当它在 shell 下运行时，将留在前端运行

① 即配置文件的开始部分。

debug_points	值：stop 或 abort 语法：debug_points stop; 默认值：None 激活调试点。当一个调试点为了附加一个调试器来改变方向而使用 stop 来打断应用程序；使用 abort 来放弃一个调试点并且建立一个内核转储文件[1]
env	语法： env MY_VARIABLE; env MY_VARIABLE=my_value; 允许（重)定义环境变量
error_log 使用环境：main, http, server 和 location	语法：error_log /file/path level; 默认值：logs/error.log error 这里的级别有下列值：debug, info, notice, warn, error 和 crit(详细程度由高到低：debug 提供了全部日志记录，crit 仅报告关键错误)。 能够提供不同的错误日志级别：应用程序、HTTP 服务、虚拟主机和虚拟主机目录。 通过重定向，日志记录可以重定向到/dev/null，在配置文件的根部使用下列指令可以禁用错误日志记录： error_log /dev/null crit;
lock_file	语法： 文件路径 lock_file logs/nginx.lock; 默认值：在编译时定义 使用 lock 文件是为了互拆现象。默认是不使用，除非在编译时已启用
log_not_found 使用环境：main，http，server 和 location	值：on 或 off 语法：log_not_found on; 默认值：on 开启或禁用记录 404 错误(404 not found)，如果日志中填满了无法访问 favicon.ico 或 robots.txt 文件而产生的 404 错误，可以考虑将其关掉
master_process	值：on 或 off 语法：master_process on; 默认值：on 如果设置为 on，Nginx 将开启多个进程：一个主进程(即 master 进程)和 worker 进程；如果禁用，Nginx 会以独一无二的进程来运行。该指令仅被用于测试，作为一个 master 进程——客户端无法连接到你的服务器

① 这一类指令对于我们这些最终用户其实没什么用，它们大都是被开发者所使用。

指令和使用环境	语法和描述

续表

指令和使用环境	语法和描述
debug_points	值：stop 或 abort 语法：debug_points stop; 默认值：None 激活调试点。当一个调试点为了附加一个调试器来改变方向而使用 stop 来打断应用程序；使用 abort 来放弃一个调试点并且建立一个内核转储文件[1]
env	语法： env MY_VARIABLE; env MY_VARIABLE=my_value; 允许（重)定义环境变量
error_log 使用环境：main, http, server 和 location	语法：error_log /file/path level; 默认值：logs/error.log error 这里的级别有下列值：debug, info, notice, warn, error 和 crit(详细程度由高到低：debug 提供了全部日志记录，crit 仅报告关键错误)。 能够提供不同的错误日志级别：应用程序、HTTP 服务、虚拟主机和虚拟主机目录。 通过重定向，日志记录可以重定向到/dev/null，在配置文件的根部使用下列指令可以禁用错误日志记录： error_log /dev/null crit;
lock_file	语法： 文件路径 lock_file logs/nginx.lock; 默认值：在编译时定义 使用 lock 文件是为了互拆现象。默认是不使用，除非在编译时已启用
log_not_found 使用环境：main，http，server 和 location	值：on 或 off 语法：log_not_found on; 默认值：on 开启或禁用记录 404 错误(404 not found)，如果日志中填满了无法访问 favicon.ico 或 robots.txt 文件而产生的 404 错误，可以考虑将其关掉
master_process	值：on 或 off 语法：master_process on; 默认值：on 如果设置为 on，Nginx 将开启多个进程：一个主进程(即 master 进程)和 worker 进程；如果禁用，Nginx 会以独一无二的进程来运行。该指令仅被用于测试，作为一个 master 进程——客户端无法连接到你的服务器

① 这一类指令对于我们这些最终用户其实没什么用，它们大都是被开发者所使用。

续表

指令和使用环境	语法和描述
pid	语法： 文件路径 pid logs/nginx.pid; 默认值：在编译时定义 用于存放 Nginx 守护进程的 pid 文件路径。默认值为编译时配置的路径
ssl_engine	语法： 字符串 ssl_engine enginename; 默认值：无 这里的 enginename 是一个系统的有效硬件 SSL 加速器。要检测有效的 SSL 加速器装置，可以在 shell 中使用下面的命令： Openssl engine –t
thread_stack_size	语法： 数字(大小) thread_stack_size 1m; 默认值：None 定义线程堆栈的大小，请参考下面的 worker_threads 指令
timer_resolution	语法： 数字(时间) timer_resolution 100ms; 默认值：无 控制系统调用 gettimeofday()时间间隔，它会通过调用该函数来与内部时钟同步。如果这个值没有指定，在每一次内核事件通知后时钟都会被刷新
user	语法： user username groupname; user username; 默认值：在编译时定义，如果没有定义，则使用 Nginx 的 master 进程的用户和用户组。 该指令允许你定义一个用户及其相应的个组，Nginx 的 worker 进程将使用这些设置来运行
worker_threads	语法： Numeric worker_threads 8; 默认值：无 为每一个 worker 进程定义一定数量的线程。 警告！线程在默认情况下禁用。作者声明“该代码已经被破解。”

worker_cpu_affinity	语法： worker_cpu_affinity 1000 0100 0010 0001; worker_cpu_affinity 10 10 01 01; worker_cpu_affinity; 默认值：无 该指令与 worker_processes 协同工作，它可以让你的 Worker 进程影响 CPU 内核，数字序列和 worker 进程一样多。 如果你配置 Nginx 使用 3 个 worker 进程，那么就会有三个数字组合(可以认为是三个块)，对于双核 CPU，每一个块有两位数字：worker_cpu_affinity 01 01 10; ● 第一块 (01) 指示第一个 worker 进程会受第二个内核的影响； ● 第二块(01) 指示第二个 worker 进程会受第二个内核的影响； ● 第三块(10)指示第三个 worker 进程会受第一个内核的影响。 请注意，这种亲和力(affinity)只建议针对多核处理器，而非超线程或类似技术的处理器 指令：worker_cpu_affinity 语法：worker_cpu_affinity cpumask [cpumask...] 默认值：无 说明：该选项仅用于 Linux，该选项允许你将 worker 进程绑定到一个 CPU 上，通过调用 sched_setaffinity()。 例如： worker_processes 4; worker_cpu_affinity 0001 0010 0100 1000; 将每一个进程绑定到一颗 CPU。 worker_processes 2; worker_cpu_affinity 0101 1010; 将第一个 worker 绑定到 CPU0/CPU2； 第二个 worker 绑定到 CPU1/CPU3。 这适用于超线程(HTT)CPU。

指令和使用环境	描　述

续表

指令和使用环境	描　述
worker_cpu_affinity	语法： worker_cpu_affinity 1000 0100 0010 0001; worker_cpu_affinity 10 10 01 01; worker_cpu_affinity; 默认值：无 该指令与 worker_processes 协同工作，它可以让你的 Worker 进程影响 CPU 内核，数字序列和 worker 进程一样多。 如果你配置 Nginx 使用 3 个 worker 进程，那么就会有三个数字组合(可以认为是三个块)，对于双核 CPU，每一个块有两位数字：worker_cpu_affinity 01 01 10; ● 第一块 (01) 指示第一个 worker 进程会受第二个内核的影响； ● 第二块(01) 指示第二个 worker 进程会受第二个内核的影响； ● 第三块(10)指示第三个 worker 进程会受第一个内核的影响。 请注意，这种亲和力(affinity)只建议针对多核处理器，而非超线程或类似技术的处理器 指令：worker_cpu_affinity 语法：worker_cpu_affinity cpumask [cpumask...] 默认值：无 说明：该选项仅用于 Linux，该选项允许你将 worker 进程绑定到一个 CPU 上，通过调用 sched_setaffinity()。 例如： worker_processes　4; worker_cpu_affinity 0001 0010 0100 1000; 将每一个进程绑定到一颗 CPU。 worker_processes　2; worker_cpu_affinity 0101 1010; 将第一个 worker 绑定到 CPU0/CPU2； 第二个 worker 绑定到 CPU1/CPU3。 这适用于超线程(HTT)CPU。

worker_priority	语法： 数字 worker_priority 0; 默认值：0 定义 worker 进程的优先级，从-20(最高级)到 19(最低级)，默认值为 0。注意，内核进程运行在-5 优先级，因此不推荐设置-5 或较小。
worker_processes	语法： 数字 worker_processes 4; 默认值：1 定义 worker 进程的数量，Nginx 可将请求的处理分到多个 worker 进程。 默认值为 1，如果属于 CPU 多核，推荐增加该值。 此外，如果一个进程由于慢的 I/O 操作被阻塞，那么进入的请求会转交给其他的 worker 进程
worker_rlimit_core	语法： 数字(大小) worker_rlimit_core 100m; 默认值：无 定义每个 worker 进程的内核文件大小
worker_rlimit_nofile	语法： 数字 worker_rlimit_nofile 10000; 默认值：无 定义一个 worker 进程可以同时处理的文件数量
worker_rlimit_sigpending	语法： 数字 worker_rlimit_sigpending 10000; 默认值：无 定义每个用户(调用进程的用户 ID)能够被排入队列的信号(signal)数量。如果队列(queue)满，会由于这个限制而导致信号(signals)被忽略
working_directory	语法： 目录路径 working_directory /usr/local/nginx/; 默认值：在配置编译时通过 prefix 开关选项指定 这是 worker 进程工作的目录，仅用于定义内核(core)文件的位置。对于该目录，worker 进程用户(user 指令指定的用户)必须有写的权限，用于能够写入内核(core 文件)

指令和使用环境	描述

续表

指令和使用环境	描　述
worker_priority	语法： 数字 worker_priority 0; 默认值：0 定义 worker 进程的优先级，从-20(最高级)到 19(最低级)，默认值为 0。注意，内核进程运行在-5 优先级，因此不推荐设置-5 或较小。
worker_processes	语法： 数字 worker_processes 4; 默认值：1 定义 worker 进程的数量，Nginx 可将请求的处理分到多个 worker 进程。 默认值为 1，如果属于 CPU 多核，推荐增加该值。 此外，如果一个进程由于慢的 I/O 操作被阻塞，那么进入的请求会转交给其他的 worker 进程
worker_rlimit_core	语法： 数字(大小) worker_rlimit_core 100m; 默认值：无 定义每个 worker 进程的内核文件大小
worker_rlimit_nofile	语法： 数字 worker_rlimit_nofile 10000; 默认值：无 定义一个 worker 进程可以同时处理的文件数量
worker_rlimit_sigpending	语法： 数字 worker_rlimit_sigpending 10000; 默认值：无 定义每个用户(调用进程的用户 ID)能够被排入队列的信号(signal)数量。如果队列(queue)满，会由于这个限制而导致信号(signals)被忽略
working_directory	语法： 目录路径 working_directory /usr/local/nginx/; 默认值：在配置编译时通过 prefix 开关选项指定 这是 worker 进程工作的目录，仅用于定义内核(core)文件的位置。对于该目录，worker 进程用户(user 指令指定的用户)必须有写的权限，用于能够写入内核(core 文件)

Events 模块

与 Events 模块一起提供的指令可以用来配置网络机制，指令的一些参数会对 Nginx 应用程序的性能产生重要的影响。

下面列出的所有指令必须放在 events 区段(块)，events 模块的配置部分放在配置文件根部：

```
user nginx nginx;
master_process on;
worker_processes 4;
events {
    worker_connections 1024;
    use epoll;
}
[...]
```

这些指令不能在别处使用。如果非要这么做，配置文件将无法通过测试。

accept_mutex	值：on 或 off accept_mutex on; 默认值：on 启用或禁用使用一个接受互斥(互斥现象)锁来打开套接字监听
accept_mutex_delay	语法： 数字(时间) accept_mutex_delay 500ms; 默认值：500 milliseconds 定义一个 worker 进程在尝试再次获取资源之前应该等待多长时间。如果指令 accept_mutex 设置为 off，则不使用该值(指的是指令 accept_mutex_delay 的值)
connections	该指令已被 worker_connections 取代，不赞成再使用

指令	描述

Events 模块

与 Events 模块一起提供的指令可以用来配置网络机制，指令的一些参数会对 Nginx 应用程序的性能产生重要的影响。

下面列出的所有指令必须放在 events 区段(块)，events 模块的配置部分放在配置文件根部：

```
user nginx nginx;
master_process on;
worker_processes 4;
events {
    worker_connections 1024;
    use epoll;
}
[...]
```

这些指令不能在别处使用。如果非要这么做，配置文件将无法通过测试。

指　令	描　述
accept_mutex	值：on 或 off accept_mutex on; 默认值：on 启用或禁用使用一个接受互斥(互斥现象)锁来打开套接字监听
accept_mutex_delay	语法： 数字(时间) accept_mutex_delay 500ms; 默认值：500 milliseconds 定义一个 worker 进程在尝试再次获取资源之前应该等待多长时间。如果指令 accept_mutex 设置为 off，则不使用该值(指的是指令 accept_mutex_delay 的值)
connections	该指令已被 worker_connections 取代，不赞成再使用

debug_connection	语法： IP 地址 或 CIDR 段 debug_connection 172.63.155.21; debug_connection 172.63.155.0/24; 默认值：无 对于匹配客户端的 IP 或 IP 区段，要记录详细的日志信息，这些信息存放在 error_log 指令指定的文件中，开启 debug 级别的详细日志 注意：为了使用这项功能，在配置编译 Nginx 时使用开关变量 debug
multi_accept	语法： on 或 off multi_accept off; 默认值：off 定义 Nginx 是否立刻接受从所有监听队列进入的连接
use	值：/dev/poll，epoll， eventport，kqueue，rtsig 或 select use kqueue; 默认值：在编译时定义 在有效的模型中选择 event 的模型类型(在编译时开启)，但是 Nginx 会自动选择最合适的一个。 ● select：默认的标准模块，如果 OS 不支持更有效的模型，则使用它(这种模型也是在 Windows 下仅有的一种模型) ● poll：在自动选择上它优先于 select，但在所有的系统上都无效。 ● kqueue：一种在 FreeBSD 4.1+和 OpenBSD 2.9+和 NetBSD 2.0, MacOS X 操作系统下性能高效的模型； ● epoll：一种基于 Linux 2.6+ 操作系统下有效的模型； ● rtsig：实时信号，对于 Linux 2.2.19 有效，但不适用于高流量情况，作为默认，系统设置仅允许 1 024 个队列信号； ● /dev/poll: 一种用于 Solaris 7 11/99+，HP/UX 11.22+，IRIX 6.5.15+和 Tru64 UNIX 5.1A+系统的高效模型； ● Eventport: 用于 Solaris 10 的一种高效的模型，但需要安全补丁

指令	描述

续表

指　令	描　述
debug_connection	语法： IP 地址 或 CIDR 段 debug_connection 172.63.155.21; debug_connection 172.63.155.0/24; 默认值：无 对于匹配客户端的 IP 或 IP 区段，要记录详细的日志信息，这些信息存放在 error_log 指令指定的文件中，开启 debug 级别的详细日志 注意：为了使用这项功能，在配置编译 Nginx 时使用开关变量 debug
multi_accept	语法： on 或 off multi_accept off; 默认值：off 定义 Nginx 是否立刻接受从所有监听队列进入的连接
use	值：/dev/poll，epoll， eventport，kqueue，rtsig 或 select use kqueue; 默认值：在编译时定义 在有效的模型中选择 event 的模型类型(在编译时开启)，但是 Nginx 会自动选择最合适的一个。 ● select：默认的标准模块，如果 OS 不支持更有效的模型，则使用它(这种模型也是在 Windows 下仅有的一种模型) ● poll：在自动选择上它优先于 select，但在所有的系统上都无效。 ● kqueue：一种在 FreeBSD 4.1+和 OpenBSD 2.9+和 NetBSD 2.0, MacOS X 操作系统下性能高效的模型； ● epoll：一种基于 Linux 2.6+ 操作系统下有效的模型； ● rtsig：实时信号，对于 Linux 2.2.19 有效，但不适用于高流量情况，作为默认，系统设置仅允许 1 024 个队列信号； ● /dev/poll: 一种用于 Solaris 7 11/99+，HP/UX 11.22+，IRIX 6.5.15+和 Tru64 UNIX 5.1A+系统的高效模型； ● Eventport: 用于 Solaris 10 的一种高效的模型，但需要安全补丁

续表

指　令	描　述
worker_connections	语法： 数字 worker_connections 1024; 接受值：无 定义一个 worker 进程能够同时连接的数量

Configuration 模块

Nginx 的 Configuration 模块是一个简单的模块，它提供的 include 指令能够将其他文件包含在 Nginx 配置文件中。在配置文件中的任何地方均可插入该指令，并且在它后面只有一个参数——文件路径：

```
include /file/path.conf;
include sites/*.conf;
```

注意，如果没有指定绝对路径，那么文件的路径将和配置文件的目录相关，换言之，Nginx 会认为与配置文件在同一目录下。默认为“include sites/example.conf”，它实际上包含以下文件：

```
/usr/local/nginx/conf/sites/example.conf.
```

适合你需求的配置文件

随着一个由基本模块开始的长指令列表，就目标流量而言，对于你的硬件，我们预想一个能够应付你的需求配置。在本小节中，我们将第一次近距离看看默认的配置文件，以便理解每一项设置的含义。

理解默认的配置文件

为什么 Nginx 能够在众多的 Web 服务器中占有一席之地，因为它是极端轻量级、最优化和最简单并且也是最高效的 Web 服务器。例如，默认的配置文件效率较高，在许多情况下，都用不着改变初始的设置。

现在，我们将开始学习默认的配置文件，主配置文件为 nginx.conf，然而你会发现这个文件几乎是空的。原因在于，当一个指令没有出现在配置文件中时，则使用其默认值。因此，我们也将考虑原始设置中指令的默认值。

```
user root root;
worker_processes 1;
worker_priority 0;
error_log logs/error.log error;
log_not_found on;
events {
    accept_mutex on;
    accept_mutex_delay 500ms;
    multi_accept off;
    worker_connections 1024;
}
```

这个配置文件也许在别的地方也适用，但你得需要着手解决一些问题。

必要的调整

我们将检查一些需要立即改变配置指令，将其修改为你可能设置的值。

- **user root root;**

  该指令指定工作过程在开始时以 root 的用户运行。对这样的进程授予对整个文件的全部系统权限是很危险的。你需要在系统上建立一个新的帐户，并且在这里使用它。关于创建用户和用户组，请参阅第 1 章。

  推荐值(现在假定为你在系统上创建了用户和用户组)：

  ```
  user nginx nginx;
  ```

  ```
  worker_processes 1;
  ```

  这个设置，你只能开启一个工作过程，这意味着所有的请求都将由这个单一的进程来处理(Nginx 当前的版本不是多线程)，这也意味着所有的执行都将委托给 CPU 的一个核。强烈推荐增加该值，至少应该为 CPU 的每个核分配一个工作进程。推荐值(假定你的 CPU 为 4 核)：worker_processes 4;

- **`worker_priority 0;`**

  默认情况下，worker 进程会在一个适当的优先级启动。如果系统同时在执行其他的任务，你可能想授予 Nginx worker 进程更高一些优先级。在这种情况下，应该降低——减小该值，提高优先级。值的范围为-20(优先级最高)～19(优先权最低)，在这里没有推荐值，完全取决于具体环境。然而，设置的值不应该低于-5，因为它是内核进程的默认优先级：

- **`log_not_found on;`**

  该指令指定 Nginx 是否记录 404 错误。然而对于找不到资源，这些错误可以提供有用的信息，这些错误记录中多是由于浏览者尝试访问一个 *favicon*(按照惯例一般是网站的 favicon.ico)，或者是 robots 试图访问一个索引结构(robots.txt)。推荐将 log_not_found 设置为 off，因为就常规文件而言，这样的错误记录会使日志文件变得非常零乱，然而不要在 server 级别将 log_not_found 的值设置为 off。注意，该指令是 HTTP Core 模块的组成部分，更多信息请参考第 4 章。

- **`worker_connections 1024;`**

  如果将工作进程设置为 4，每一个进程接受 1024 个连接，那么服务器将能够同时“款待”4096 连接。这时需要调整硬件设置：在服务器上配置更多的内存和 CPU，使服务能够同时完成更多的连接请求。

适当选择硬件

我们现在建立三个不同设置的服务器：一个标准的配置，将一个利用常规硬件组成的服务器用在适当的网站；一个低流量的设置旨在适当优化硬件的性能；最后，在一个适当设置的在高流量情况下的生产环境下的服务器。

总是难以为计算机的性能进行分类，首先，因为每个环境都有自己的资源。如果你在一个大公司工作，那么和一个网管讨论一个功能强大的计算机，这两者是不具有相同的含义，一个独立的网站而另一个或许是由第三方的 Web 主机来提供。第二，因为计算机每年都在变得强大：更快的 CPU，廉价的内存和新技术的崛起(SSD)。所以，下面给出的规格在这里只供参考，需要加以调整，以便适于你自己的情况和你的计算机所在的时代。对于各种指令的推荐值可以直接基于前面的说明——CPU 的每个核一个 worker 进程，最大连接数基于 RAM，等等。

CPU：双核 RAM：2 GB 请求：~ 1/s	CPU：四核 RAM：4 GB 请求：~ 50/s	CPU：八核 RAM：12 GB 请求：~1000/s

worker_processes 2;
worker_rlimit_nofile1024;
worker_priority -5;
worker_cpu_affinity 01 10;
events {
multi_accept on;
worker_connections 128;
}
	worker_processes 4;
	worker_rlimit_nofile 8192;
	worker_priority 0;
	worker_cpu_affinity
	0001 0010 0100 1000;
	events {
	multi_accept off;
	worker_connections 1024;
	}
		worker_processes 8;
		worker_priority 0;
		events {
		multi_accept off;
		worker_connections 8192;
		}

也就是说 worker 进程和连接限制的数量，对于第一个问题，如果设置的不适当，可能会使得 CPU 的某一个核发生混乱，而其他的却没有使用或没有充分利用。要确定 worker_processes 匹配你的 CPU 内核数量。

对于第二个问题，如果设置太低，可能会导致连接被拒绝，如果设置的太高，可能会内存溢出，导致整个系统的崩溃，不幸的是，一次方程是计算不出 worker_connections 指令的值，你需要基于机器的配置做流量估算。

低流量的配置	标准配置	高流量配置

推荐值

总是难以为计算机的性能进行分类，首先，因为每个环境都有自己的资源。如果你在一个大公司工作，那么和一个网管讨论一个功能强大的计算机，这两者是不具有相同的含义，一个独立的网站而另一个或许是由第三方的 Web 主机来提供。第二，因为计算机每年都在变得强大：更快的 CPU，廉价的内存和新技术的崛起(SSD)。所以，下面给出的规格在这里只供参考，需要加以调整，以便适于你自己的情况和你的计算机所在的时代。对于各种指令的推荐值可以直接基于前面的说明——CPU 的每个核一个 worker 进程，最大连接数基于 RAM，等等。

低流量的配置	标准配置	高流量配置
CPU：双核 RAM：2 GB 请求：~ 1/s	CPU：四核 RAM：4 GB 请求：~ 50/s	CPU：八核 RAM：12 GB 请求：~1000/s
推荐值		

worker_processes 2;
worker_rlimit_nofile1024;
worker_priority -5;
worker_cpu_affinity 01 10;
events {
multi_accept on;
worker_connections 128;
}
	worker_processes 4;
	worker_rlimit_nofile 8192;
	worker_priority 0;
	worker_cpu_affinity
	0001 0010 0100 1000;
	events {
	multi_accept off;
	worker_connections 1024;
	}
		worker_processes 8;
		worker_priority 0;
		events {
		multi_accept off;
		worker_connections 8192;
		}

也就是说 worker 进程和连接限制的数量，对于第一个问题，如果设置的不适当，可能会使得 CPU 的某一个核发生混乱，而其他的却没有使用或没有充分利用。要确定 worker_processes 匹配你的 CPU 内核数量。

对于第二个问题，如果设置太低，可能会导致连接被拒绝，如果设置的太高，可能会内存溢出，导致整个系统的崩溃，不幸的是，一次方程是计算不出 worker_connections 指令的值，你需要基于机器的配置做流量估算。

测试服务器

服务器的基本配置现在已经确定，在下面的章节中，将进行 HTTP 模块的配置和如何建立一个虚拟主机。但是现在，需要确定一下我们的设置是否正确，是否适合生产环境。

建立测试服务器

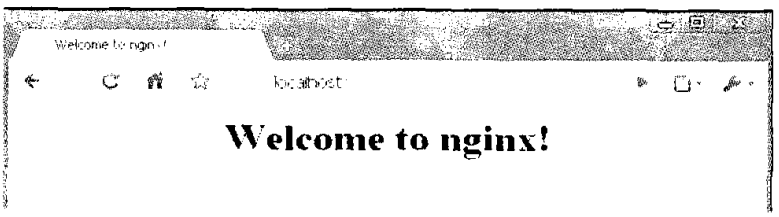
为了执行简单的测试，例如通过浏览器连接到 Web 服务器，我们需要为 Nginx 创建一个 Web 站点，以便提供 Web 服务。在默认的安装包内，有一个简单的测试页，在 html 目录内(/usr/local/nginx/html/index.html)，并且原始的 Nginx 配置文件 nginx.conf 也包含对该网页的访问。我们现在感兴趣的是下面这一部分：

```
http {
    include mime.types;
    default_type application/octet-stream;
    sendfile on;
    keepalive_timeout 65;
    server {
        listen 80;
        server_name localhost;
        location / {
            root html;
            index index.html index.htm;
    }
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root html;
    }
}
```

Nginx 以这一部分配置为基础作为一个网站来提供 Web 服务：

- 通过 80 端口监听 tcp 套接字
- 访问地址 *http://localhost/*
- 主页为 index.html

关于这些指令更详细的阐述，请参考第 4 章，通过访问 *http://localhost/*，点燃(fire up)了你幸运的网站，如下图所示。



随后将出现欢迎信息，如果没有看到这个欢迎消息，则需要再次检查配置文件，修改配置，以便确定再次载入配置文件会有好消息。

性能测试

对 Nginx 的设置完成基本功能和结构的配置之后，你可能会想进行一些测试，例如重新载入配置文件，再次运行测试，再次编辑配置文件，等等。在理想的情况下，要避免在运行 Nginx 服务的机器上运行测试工具，因为这会影响实际的效果，使测试值有偏差。

眼下，人们可能会质疑性能测试的相关性，一方面，虚拟主机和模块都没有完全配置，你的网站可能会使用 FastCGI 应用程序(PHP 和 Python 等)；另一方面，我们正在进行的测试是服务器原始的性能，没有额外的组成成分，例如，没有确定完全使用 CPU 内核，因此，在将服务器投入生产之前，总还有许多地方需要完善。

在这里，我们使用三个工具来评估服务的性能。这三个应用程序专门用于负载测试，由于其来源不同，所以测试方式也不相同：

- httperf　一个众所周知的开源工具，由 HP 开发，仅使用于 Linux 操作系统；
- autobench　由 Perl 封装的 httperf，改善了测试机制并且产生详细的报告；
- OpenWebLoad　规模较小的开放源码的负载测试工具。

这些工具的背后原理都是产生大量的 HTTP 请求，进而产生复杂的访问请求，最终达到全面测试服务器和研究的结果。

Httperf

httperf 是一个简单的命令行工具，官方网址为 *http://www.hpl.hp.com/research/linux/httperf/*，可以从官网下载并安装，源代码以归档文件 tar.gz 形式发布，需要编译安装，它的编译方法是标准的方法：./configure，make 和 make install。安装之后，可以使用下面的命令：

```
[alex@example ~]$ httperf --server 192.168.1.10 --port 80 --uri
/index.html --rate 300 --num-conn 30000 --num-call 1 --timeout 5
```

在前面的命令行中，用实际环境中的值替换以下参数：

- --server　你所测试的网站名(主机名、域名或 IP 地址)
- --uri　指定下载的文件
- --rate　每秒发送的请求
- --num-conn　连接的总数
- --num-call　每个连接发送的请求数
- --timeout　超时时间

如下图所示，在这个例子中，httperf 将重复下载 *http://192.168.1.10/index.html*，每秒 300 次，总共请求 30 000 次。

alex@example: /home/alex

```
Maximum connect burst length: 6298

Total: connections 21767 requests 21710 replies 21710 test-duration 14.692 s

Connection rate: 1481.6 conn/s (0.7 ms/conn, <=1022 concurrent connections)
Connection time [ms]: min 1.4 avg 563.4 max 3922.6 median 197.5 stddev 988.4
Connection time [ms]: connect 397.6
Connection length [replies/conn]: 1.000

Request rate: 1477.7 req/s (0.7 ms/req)
Request size [B]: 72.0

Reply rate [replies/s]: min 1942.8 avg 2077.3 max 2211.8 stddev 190.2 (2 samples)
Reply time [ms]: response 165.7 transfer 0.0
Reply size [B]: header 215.0 content 151.0 footer 0.0 (total 366.0)
Reply status: 1xx=0 2xx=21702 3xx=0 4xx=0 5xx=8

CPU time [s]: user 0.22 system 8.88 (user 1.5% system 60.5% total 62.0%)
Net I/O: 633.5 KB/s (5.2*10^6 bps)

Errors: total 78290 client-timo 57 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 78233 addrunavail 0 ftab-full 0 other 0
```

Httperf

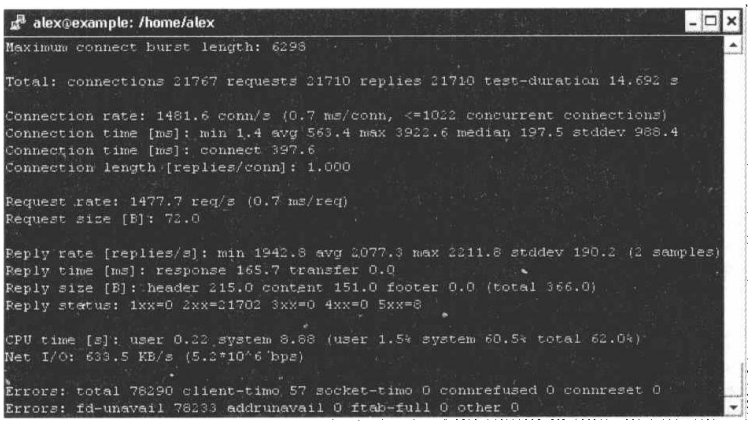
httperf 是一个简单的命令行工具，官方网址为 *http://www.hpl.hp.com/research/linux/httperf/*，可以从官网下载并安装，源代码以归档文件 tar.gz 形式发布，需要编译安装，它的编译方法是标准的方法：./configure，make 和 make install。安装之后，可以使用下面的命令：

```
[alex@example ~]$ httperf --server 192.168.1.10 --port 80 --uri
/index.html --rate 300 --num-conn 30000 --num-call 1 --timeout 5
```

在前面的命令行中，用实际环境中的值替换以下参数：

- --server　你所测试的网站名(主机名、域名或 IP 地址)
- --uri　指定下载的文件
- --rate　每秒发送的请求
- --num-conn　连接的总数
- --num-call　每个连接发送的请求数
- --timeout　超时时间

如下图所示，在这个例子中，httperf 将重复下载 *http://192.168.1.10/index.html*，每秒 300 次，总共请求 30 000 次。



结果表明：响应时间和成功的请求数量。如果成功率为 100%或响应时间接近 0ms，则增加请求，并且再次运行测试，直到服务器显示疲软迹象。测试结果如果开始显得有点不完美，那么就要适当的调整配置指令并且再次运行测试。

autobench

autobench 是一个 Perl 脚本，它能使 httperf 得以充分利用——它会连续测试和自动增长请求频率，直到服务器变得饱和。autobench 的一个有趣功能是它能够产生一个.tsv 报告，你可以使用各种应用程序打开并产生图形。

可以在作者个人网站下载该软件：*http://www.xenoclast.org/autobench/*，下载完成后解压归档文件，然后运行 make 和 make install 命令。

尽管它还支持一次测试多个主机，但这里只使用更简单的单机测试。它的命令行类似于 httperf 的执行：

```
[alex@example ~]$ autobench --single_host --host1 192.168.1.10
\
        --uri1 /index.html --quiet --low_rate 20  \
        --high_rate 200 --rate_step 20 --num_call 10  \
        --num_conn 5000 --timeout 5 --file results.tsv
```

可以配置如下开关变量：

- --host1：所测试网站的名称(主机名、域名或 IP)
- --uri1：指定下载的文件
- --quiet：不在屏幕上显示 httperf 信息
- --low_rate：在测试开始时每秒的连接数
- --high_rate：在测试结束时每秒的连接数
- --rate_step：连接数在每一次测试后的增长频率
- --num_call：每个连接发送的请求数
- --num_conn：连接的总数
- --timeout：超时时间
- --file：将测试结果导出到一个指定的文件（.tsv 文件）

一旦测试结束，会得到一个.tsv 文件，可以用一个应用程序(例如微软的 Excel)打开。下图是一个测试服务器的测试结果生成的图形(注意，这个状态报告中包含 10 个连续的状态)：

1 2 3 4 5 6 7 8 9 10

测试步骤

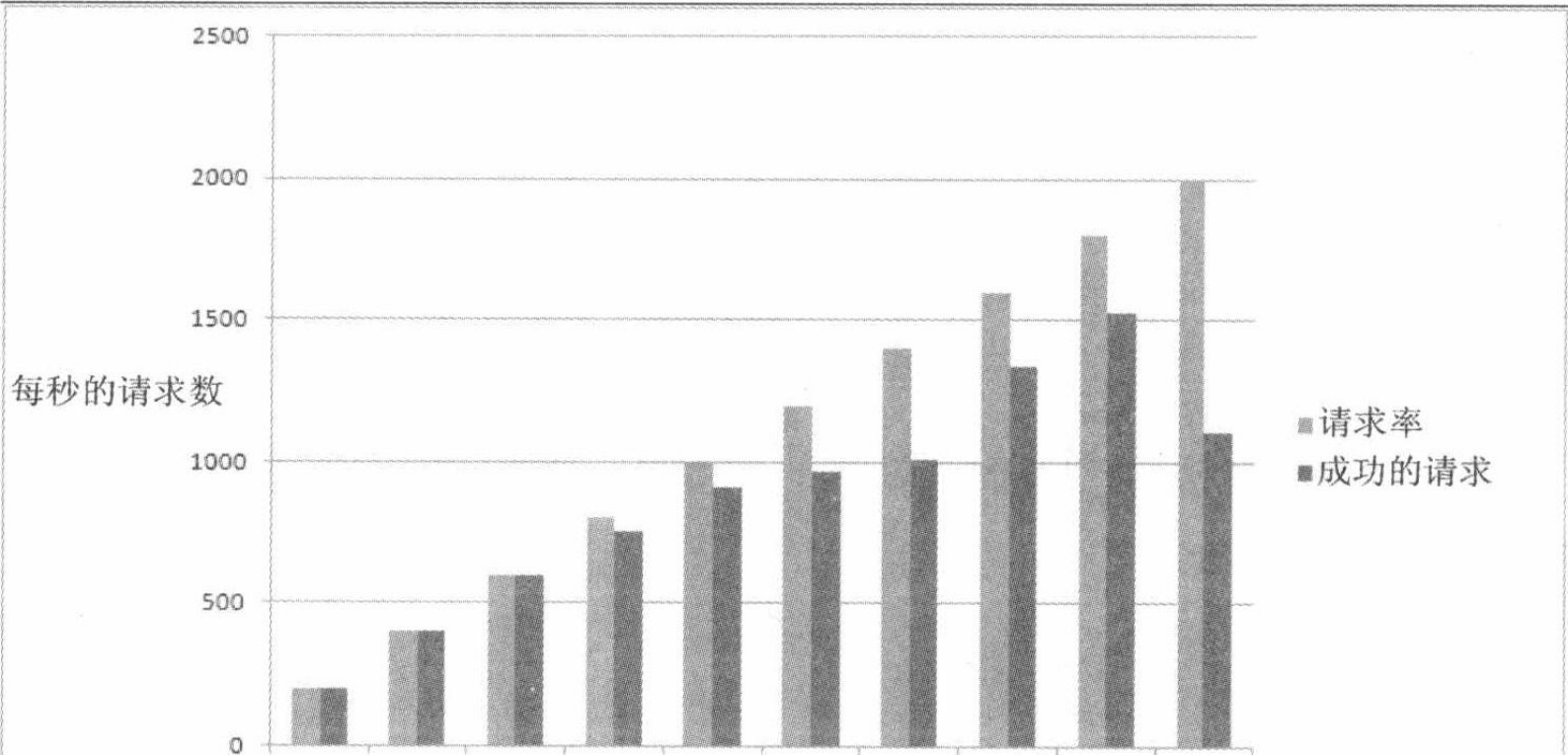
从图中可以看出，该服务器在没有任何丢失的情况下，每秒支持 600 个请求，一旦超过这个限制，一些连接会被丢弃，因为 Nginx 超出负荷便无法处理。但是在第 9 步中，它能够每秒成功处理 1500 个请求。

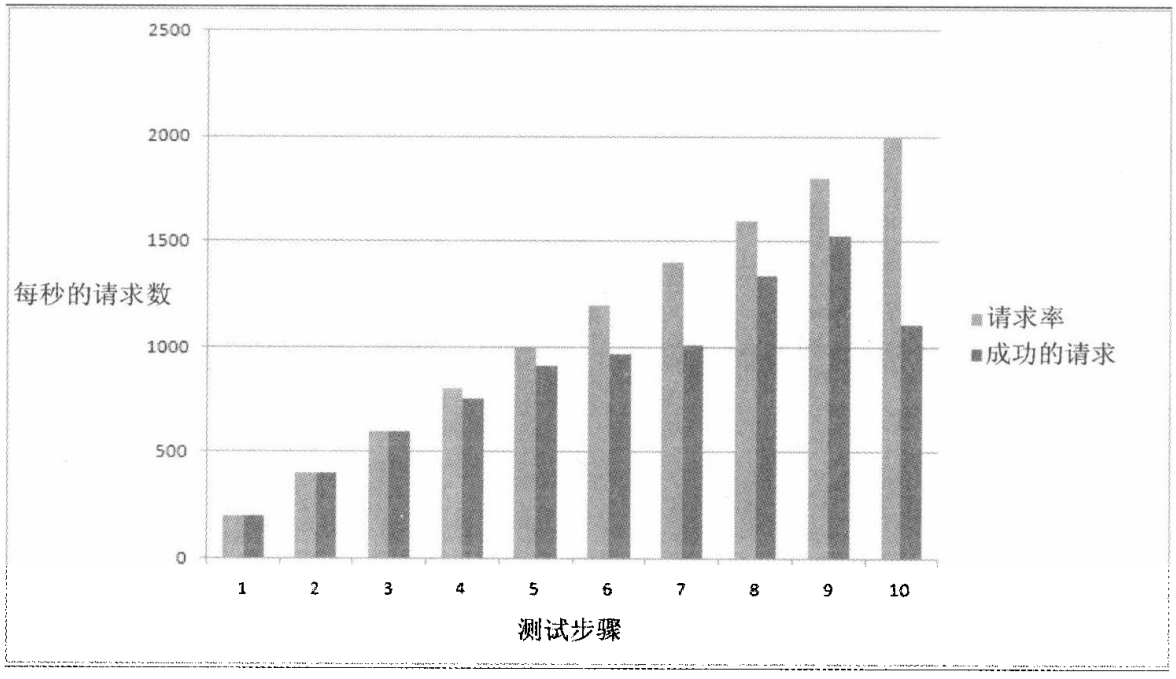
 警告：这些测试都是来自于虚拟机，不能反映生产环境中的真实能力。

OpenWebLoad

OpenWebLoad 是一个自由的开源系统，它适用于 Linux 和 Windows 平台，最早开发于 2000 年。它提供了另一种方法，即在服务器上抛出大量请求，看能够正确处理多少，它只是使用数量可变的连接，发送尽可能多的请求，每一秒发送报告。

可以从官方网站下载：*http://openwebload.sourceforge.net*，源代码以归档文件.tar.gz 形式下载，解压后执行./configure，make 和 make install。


2500
2000
1500
1000
500
0
每秒的请求数
请求率
成功的请求



从图中可以看出，该服务器在没有任何丢失的情况下，每秒支持 600 个请求，一旦超过这个限制，一些连接会被丢弃，因为 Nginx 超出负荷便无法处理。但是在第 9 步中，它能够每秒成功处理 1500 个请求。

警告：这些测试都是来自于虚拟机，不能反映生产环境中的真实能力。

OpenWebLoad

OpenWebLoad 是一个自由的开源系统，它适用于 Linux 和 Windows 平台，最早开发于 2000 年。它提供了另一种方法，即在服务器上抛出大量请求，看能够正确处理多少，它只是使用数量可变的连接，发送尽可能多的请求，每一秒发送报告。

可以从官方网站下载：*http://openwebload.sourceforge.net*，源代码以归档文件.tar.gz 形式下载，解压后执行./configure，make 和 make install。

它的用法比前两个工具简单：

```
[alex@example ~]$ openload example.com/index.html 10
```

URL 中的第一个参数为测试网页；第二个参数为打开的连接数。

如图所示，每秒产生一个新的结果行，请求不断地发送，直到你按下 Enter 键，然后紧跟着显示一个摘要。下面是有关输出的解释：

- **Tps**(每秒处理数) 一个处理相当于一个完整的请求(一个来回)
- **MaTps** 过去 20 秒的 Tps 平均值
- **Resp Time** 过去 1 秒的平均值
- **Err** (错误率) 服务器返回的响应不是预料的 200 时，会发生错误
- **Count** 处理的总数

为了给设置建立平稳的配置，可以反复折腾并发连接数。下表是三个不同数量的连接，结果不言自明。

同时连接数	1	20	1000
每秒处理数(Tps)	67.54	205.87	185.07
平均响应时间	14 ms	91 ms	596 ms

C:\WINDOWS\system32\cmd.exe

```
C:\>openload.exe example.com/index.html 10
URL: http://example.com:80/index.html
Clients: 10
MaTps 210.37, Tps 210.37, Resp Time  0.046, Err   0%, Count   211
MaTps 211.51, Tps 221.78, Resp Time  0.045, Err   0%, Count   433
MaTps 212.69, Tps 223.33, Resp Time  0.045, Err   0%, Count   657
MaTps 213.38, Tps 219.56, Resp Time  0.046, Err   0%, Count   877
MaTps 214.63, Tps 225.87, Resp Time  0.044, Err   0%, Count  1104
MaTps 215.20, Tps 220.34, Resp Time  0.045, Err   0%, Count  1325
Total TPS: 216.40
Avg. Response time:  0.045 sec.
Max Response time:   0.097 sec
Total Requests:    1325
Total Errors:         0

C:\>
```

指标	测试1	测试2	测试3

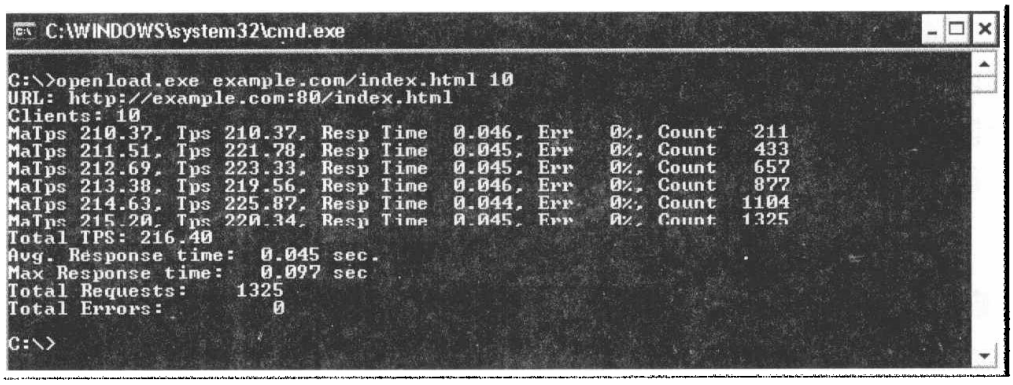
它的用法比前两个工具简单：

```
[alex@example ~]$ openload example.com/index.html 10
```

URL 中的第一个参数为测试网页；第二个参数为打开的连接数。

如图所示，每秒产生一个新的结果行，请求不断地发送，直到你按下 Enter 键，然后紧跟着显示一个摘要。下面是有关输出的解释：

- **Tps**(每秒处理数)　一个处理相当于一个完整的请求(一个来回)
- **MaTps**　过去 20 秒的 Tps 平均值
- **Resp Time**　过去 1 秒的平均值
- **Err** (错误率)　服务器返回的响应不是预料的 200 时，会发生错误
- **Count**　处理的总数



为了给设置建立平稳的配置，可以反复折腾并发连接数。下表是三个不同数量的连接，结果不言自明。

指　　标	测试 1	测试 2	测试 3
同时连接数	1	20	1000
每秒处理数(Tps)	67.54	205.87	185.07
平均响应时间	14 ms	91 ms	596 ms

连接太少，结果 Tps 低，然而，响应时间较佳；太多的连接产生相当的高 Tps，但是响应时间较长。因此，需要找到一个合适的中间值，使连接和响应都比较合适。

平滑升级 Nginx

在许多情况下，需要替代 Nginx 二进制文件，例如，在编译一个新版本时，希望将它用于生产环境中，或者只想启用一个新的模块，而重新编译程序。在这种情况下，大多数管理员都会停止服务器，复制新的二进制文件以覆盖旧的二进制文件，然后再重新启动 Nginx。然而在这种情况下没有考虑到一个问题，在大多数网站，会有一些情况，服务的正常运行时间至关重要，为避免丢失连接而不惜任何代价。幸运的是，Nginx 做到了这一点，Nginx 嵌入了一种机制，允许你切换二进制而不中断正常运行时间——实现 0%的请求丢失。只要仔细执行下列步骤即可。

1. 用新的 Nginx 二进制替代旧的 Nginx 二进制(默认/usr/local/nginx/sbin/nginx) 。
2. 找到 Nginx 的 master 进程的 pid，例如，通过使用 ps x | grep nginx | grep master 或查找 pid 文件内的值。
3. 给 master 进程发送一个 USR2 (12)信号——kill -USR2 ***，***替换为第(2)步中找到的 pid。这将开始升级，对旧的.pid 文件重命名，运行新的二进制文件。
4. 发送一个 WINCH (28)信号到旧的 master 进程——kill -WINCH ***，***替换为第(2)步中找到的 pid，这将使旧的 worker 进程平滑关闭。
5. 确定所有旧的 worker 进程已终止，然后再给旧的 master 进程发送 QUIT 信号——kill -QUIT ***，**替换为第(2)步中找到的 pid。

祝贺你！你已经成功完成了 Nginx 升级并没有失去任何一个连接。

小结

本章首次提供通过学习语法和核心模块指令的方法来接触配置结构，这些模块指令对于服务器总体性能起着重要作用，然后，为了适合你个人的需要，我们通过一系列的调节，提高性能的测试，让你将服务调整到最佳状态。这仅仅是个开始——然而，实际上，从现在开始我们所做的任何事都与建立配置结构相关。下一章，我们将进一步探索模块系统详细研究高级指令。

第 4 章

HTTP 配置

眼下，我们已经拥有了一个工作中的 Nginx——不但已将它安装在系统并设置为系统启动时自动启动 Nginx，而且还使用基本指令进行了组织和优化。现在该进行配置了，首先从 HTTP 核心模块入手。该模块主要涉及 HTTP 配置——它允许你建立站点来提供 Web 服务，同时也支持虚拟主机。

本章主题

- 介绍 HTTP 核心模块
- http/server/location 结构
- HTTP 核心模块指令
- HTTP 核心模块变量
- 深入 location 区段

HTTP 核心模块

HTTP 核心模块包含 HTTP 服务器所有基本的区段(block)、指令和变量。如果在配置编译时包括了它，那么它默认是启用的(具体描述参考第 2 章)，但如果你将该模块禁用，实际上也是可选的——可以在配置编译时(自己定制)不包含该模块，这样做的结果是完全禁用 HTTP 的功能，并且所有的 HTTP 模块都将不能编译。显而易见，如果你购买了本书，那么一定是对 Nginx 提供的 Web 服务感兴趣，因此你会启用这个功能。

该模块是 Nginx 中最大的一个模块——它提供了许多变量和指令，为了理解所有这些新的元素，理解如何使用它们，我们首先需要理解一下 http，server 和 location 这三个主要的区段结构。

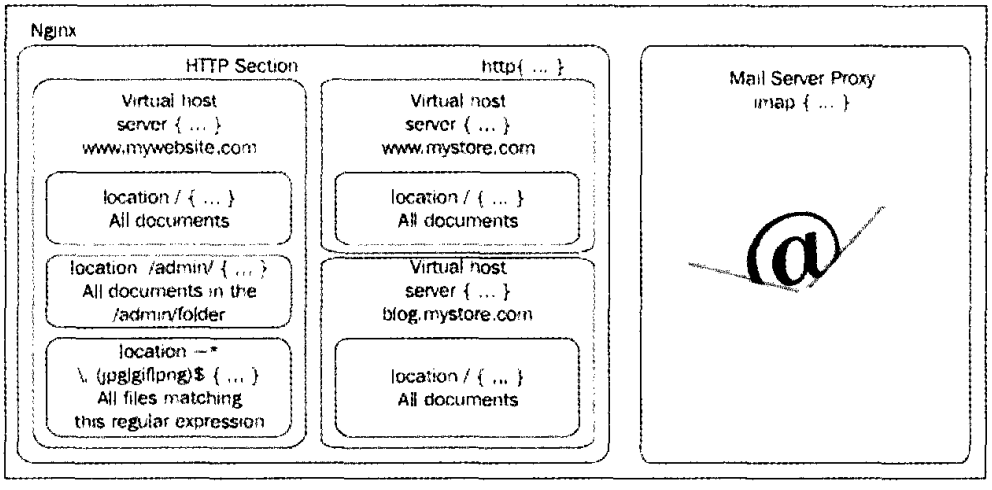
区段的结构

在第 3 章中，我们通过学习 Nginx 默认的配置文件——由一系列指令和值组成，没有任何组织结构——认识了核心模块，然后接触到 Events 模块，其时介绍了第一个区段(events)。这个区段仅由它所提供的指令使用。

随着它的出现，HTTP 模块提出了三个新的逻辑区段(块)：

- http　该区段嵌入配置文件的根部，在这个区段中允许定义指令和嵌入 HTTP 相关模块的区段，尽管本意并非如此。该块可以插入多次——但如果发生这种情况，在后面区段中的指令值会取代前面指令的值。
- server　这个区段允许你声明一个站点。换句话说，通过设置它，你能够设定网站(通过主机名，例如 www.mywebsite.com)，这样通过 Nginx 实现公认的服务器，并且让它使用你自己的配置。该区段只能用在 http 区段中。
- location　定义一组设置，应用于网站的一个特定位置，本章的下一部分进一步详细描述 location 区段。该区段能够用于 server 区段，也能嵌套在其他 location 中。

下图概述了最后的结构，它提供了两个基本的例子，相当于实际的环境。



在 HTTP 小节，通过 http 区段，涵盖整个 Web 相关的配置，它包含一个或多个 server 区段，定义了域和子域(你所管理的)。对于这些站点的每一个网站来说，你有可能定义 location 区段，以便将额外的设置添加给特定的 URI 请求或是 URI 匹配模式。

记住继承应用的原理，如果在 http 区段做了设置(例如，gzip，使用了 gzip 压缩)，那么这个设置将会持续执行到 server 和 location 区段中，如下所示：

```
http {
    # 在 http 区段支持 gzip 压缩
    gzip on;
    server {
        server_name localhost;
        listen 80;
        # 在这里 gzip 仍然开启
        location /downloads/ {
            gzip off;
            # 该指令只应用于/downloads/目录下找到的文件
        }
    }
}
```

模块指令

在这三个层次中的每一个层次，都可以插入指令，这些指令将影响 Web 服务器的行为。在下面分组专题中介绍的指令列表都由主要的 HTTP 模块引入，对于每一条指令，都指出它的使用环境——一些指令不能用于某些级别，例如，在 location 区段插入一个 server_name 指令是不合理的。在这里，下表指示了每一条指令允许使用的级别——http 块、server 块、location 块，另外还有 if 块，以及后面介绍的 Rewrite 模块。

注意：这个文档适用于 0.7.66 标准版，将来升级后，一些指令的语法或许会提供新的功能，在这里不做讨论。

套接字和主机的配置

通过对下面这组指令的设置，将配置虚拟主机。在实践中，可以根据主机名或 IP 地址和端口的组合来具体创建一个 server 区段。另外，可以通过一些指令来配置 TCP 套接字选项，以便调整网络设置。

listen 使用环境：server	指定用于提供 Web 服务站点监听的套接字所使用的 IP 地址和/或端口号，网站一般都在 80 端口提供服务(这也是默认值)。 语法：listen [address][:port] [其他选项]; 其他选项: ● default：指定该 server 区段被用于默认的网站，在该 IP 地址和端口上接受任何客户端的请求 ● ssl：指定的网站提供 SSL 服务 ● 其他选项依赖于 bind 和 listen 系统调用：backlog=num, rcvbuf=size, sndbuf=size, accept_filter=filter，deferred 和 bind 例如[1]: listen 192.168.1.1:80; listen 127.0.0.1; listen 80 default; listen [::a8c9:1234]:80; # IPv6 地址要放在方括号之间 listen 443 ssl;

① 0.7.x 语法:

listen address:port [default [backlog=num | rcvbuf=size | sndbuf=size | accept_filter=filter | deferred | bind | ssl]]

0.8.x 语法:

listen address:port [default_server [backlog=num | rcvbuf=size | sndbuf=size | accept_filter=filter | deferred | bind | ssl]]

指令和使用环境	描述

套接字和主机的配置

通过对下面这组指令的设置，将配置虚拟主机。在实践中，可以根据主机名或 IP 地址和端口的组合来具体创建一个 server 区段。另外，可以通过一些指令来配置 TCP 套接字选项，以便调整网络设置。

指令和使用环境	描　述

listen
使用环境：server
	指定用于提供 Web 服务站点监听的套接字所使用的 IP 地址和/或端口号，网站一般都在 80 端口提供服务(这也是默认值)。
	语法：listen [address][:port] [其他选项];
	其他选项:
	● default：指定该 server 区段被用于默认的网站，在该 IP 地址和端口上接受任何客户端的请求
	● ssl：指定的网站提供 SSL 服务
	● 其他选项依赖于 bind 和 listen 系统调用：backlog=num, rcvbuf=size, sndbuf=size, accept_filter=filter，deferred 和 bind
	例如[1]:
	`listen 192.168.1.1:80;`
	`listen 127.0.0.1;`
	`listen 80 default;`
	`listen [::a8c9:1234]:80;`
	`# IPv6 地址要放在方括号之间`
	`listen 443 ssl;`

① 0.7.x 语法:

```
listen address:port [ default [ backlog=num | rcvbuf=size | sndbuf=size |
accept_filter=filter | deferred | bind | ssl ] ]
```

0.8.x 语法:

```
listen address:port [ default_server [ backlog=num | rcvbuf=size |
sndbuf=size | accept_filter=filter | deferred | bind | ssl ] ]
```

server_name
使用环境：server
	在 server 区段中指定一个或多个主机名(hostname)，Nginx 收到 HTTP 请求时，它会与所有的 server 区段相比较，然后找到与客户端请求 header 中的 Host 相匹配的 server 区段。第一个与主机名匹配的 server 块将被选中。
	B 计划：如果没有 server 区段与客户端所请求匹配的主机匹配， Nginx 会选择第一个 server 区段，匹配该区段监听参数(例如，listen *:80，它是所有对 80 端口的请求)。另外，就优先级而言，具有 default 选项的 listen 优先级最高，它会被第一个选择。
	注意，该指令接受通配符，也包括正则表达式(在这种情况下，主机名将以~字符开始)。
	语法：server_name hostname1 [hostname2…];
	例如：
	`server_name www.website.com;`
	`server_name www.website.com website.com;`
	`server_name *.website.com;`
	`server_name .website.com;`
	`#combines*.website.com` 和 `website.com`
	`server_name *.website.*;`
	`server_name ~^\.example\.com$;`
	注意：可以将一个空字符串作为指令的值，为了匹配所有其他 server 不匹配的客户端请求 header 中的 host，但是至少有一个常规的名称(或者“_”，一个虚拟主机名)：
	`server_name website.com "";`
	`server_name _ "";`
	请见书后的译者注 3。

server_name_in_redirect 使用环境：http, server, location	该指令用于内部重定向(关于内部重定向的更多信息，请参阅后面的 Rewrite 模块)。如果设置为开启，Nginx 将使用 server_name 指令中指定的第一个主机名来进行重定向；如果设置为关闭，Nginx 将使用客户端 HTTP 请求头中 Host 的值进行重定向。 语法：on 或 off 默认值：on `server_name_in_redirect on;`

指令和使用环境	描　述

续表

指令和使用环境	描　　述

server_name
使用环境：server
	在 server 区段中指定一个或多个主机名(hostname)，Nginx 收到 HTTP 请求时，它会与所有的 server 区段相比较，然后找到与客户端请求 header 中的 Host 相匹配的 server 区段。第一个与主机名匹配的 server 块将被选中。
	B 计划：如果没有 server 区段与客户端所请求匹配的主机匹配，Nginx 会选择第一个 server 区段，匹配该区段监听参数(例如，listen *:80，它是所有对 80 端口的请求)。另外，就优先级而言，具有 default 选项的 listen 优先级最高，它会被第一个选择。
	注意，该指令接受通配符，也包括正则表达式(在这种情况下，主机名将以~字符开始)。
	语法：server_name hostname1 [hostname2…];
	例如：
	`server_name www.website.com;`
	`server_name www.website.com website.com;`
	`server_name *.website.com;`
	`server_name .website.com;`
	`#combines*.website.com` 和 `website.com`
	`server_name *.website.*;`
	`server_name ~^\.example\.com$;`
	注意：可以将一个空字符串作为指令的值，为了匹配所有其他 server 不匹配的客户端请求 header 中的 host，但是至少有一个常规的名称(或者“_”，一个虚拟主机名)：
	`server_name website.com "";`
	`server_name _ "";`
	请见书后的译者注 3。

server_name_in_redirect 使用环境：http, server, location	该指令用于内部重定向(关于内部重定向的更多信息，请参阅后面的 Rewrite 模块)。如果设置为开启，Nginx 将使用 server_name 指令中指定的第一个主机名来进行重定向；如果设置为关闭，Nginx 将使用客户端 HTTP 请求头中 Host 的值进行重定向。 语法：on 或 off 默认值：on `server_name_in_redirect on;`

server_names_hash_max_size 使用环境：http	Nginx 使用哈希表(hash table)来进行变量数据收集，旨在加速请求进程。该指令用于定义存储服务器名称数量哈希表的最大值。如果服务器使用的主机名总数超过 512 个，则需要增加该值。 语法：数值 默认值：512 `server_names_hash_max_size 512;`
server_names_hash_bucket_size 使用环境：http	定义在服务器名称哈希表中一个条目(或叫记录——译者注)的最大长度。如果服务器名称(或者是叫做机器名)长度大于 32 个字符，必须增加该值。 语法：数值 默认值：32 (或 64，或 128，这取决于处理器缓存规格) `server_names_hash_bucket_size 32;`
port_in_redirect 使用环境：http, server, location	就重定向来说，该指令定义 Nginx 是否将端口重定向后的端口添加在 URL 中。 语法：on 或 off 默认值：on `port_in_redirect on;`
tcp_nodelay 使用环境：http, server, location	开启或关闭使用 TCP_NODELAY 套接字选项，仅用于 keep-alive 连接。 下文引自 Linux 文档中套接字编程的相关内容： TCP_NODELAY 是一个特殊的作用，它禁用了 Nagle 缓冲算法。这种情况仅用于应用程序经常发送小的数据包，而不立即获得响应，但在这里及时投递是必需的(最常见的例子就是移动鼠标)。 使用 TCP_NODELAY 只出于一个特殊的目的，禁用 Nagle 缓冲算法。 语法：on 或 off 默认值：on `tcp_nodelay on;`

指令和使用环境	描述

续表

指令和使用环境	描　述
server_names_hash_max_size 使用环境：http	Nginx 使用哈希表(hash table)来进行变量数据收集，旨在加速请求进程。该指令用于定义存储服务器名称数量哈希表的最大值。如果服务器使用的主机名总数超过 512 个，则需要增加该值。 语法：数值 默认值：512 `server_names_hash_max_size 512;`
server_names_hash_bucket_size 使用环境：http	定义在服务器名称哈希表中一个条目(或叫记录——译者注)的最大长度。如果服务器名称(或者是叫做机器名)长度大于 32 个字符，必须增加该值。 语法：数值 默认值：32 (或 64，或 128，这取决于处理器缓存规格) `server_names_hash_bucket_size 32;`
port_in_redirect 使用环境：http, server, location	就重定向来说，该指令定义 Nginx 是否将端口重定向后的端口添加在 URL 中。 语法：on 或 off 默认值：on `port_in_redirect on;`
tcp_nodelay 使用环境：http, server, location	开启或关闭使用 TCP_NODELAY 套接字选项，仅用于 keep-alive 连接。 下文引自 Linux 文档中套接字编程的相关内容： TCP_NODELAY 是一个特殊的作用，它禁用了 Nagle 缓冲算法。这种情况仅用于应用程序经常发送小的数据包，而不立即获得响应，但在这里及时投递是必需的(最常见的例子就是移动鼠标)。 使用 TCP_NODELAY 只出于一个特殊的目的，禁用 Nagle 缓冲算法。 语法：on 或 off 默认值：on `tcp_nodelay on;`

tcp_nopush 使用环境：http, server, location	开启或禁用 TCP_NOPUSH(FreeBSD)或 TCP_CORK (Linux)的 socket 选项。注意，该选项仅作用于 sendfile 指令已启用的情况。如果 tcp_nopush 设置为 on，那么 Nginx 将尝试在单个 TCP 数据包中发送整个 HTTP 响应头。 语法：on 或 off 默认值：off `tcp_nopush off;`
sendfile 使用环境：http, server, location	如果这条指令被启用，Nginx 将使用 sendfile 内核来调用处理文件传递。如果禁用该指令，那么 Nginx 将自己处理文件传递。该选项可能会影响到服务器的性能，具体取决于被传输文件的物理位置(例如 NFS)。[①] 语法：on 或 off 默认值：off `sendfile off;`
sendfile_max_chunk 使用环境：http, server	本指令用于定义 sendfile 每一次调用数据的最大值。 语法：数值(size) 默认值：0
send_lowat 使用环境: http, server	该选项允许你在 TCP 套接字中利用 SO_SNDLOWAT 标志，这个选项仅在 FreeBSD 操作系统下可用。它的功能在于定义了用做输出操作的缓冲区大小。 语法：数值 (大小) 默认值：0
reset_timedout_connection 使用环境：http, server, location	当一个客户端连接超时，其相关的信息可能保留在内存，正是依赖于这种状态，相关的信息才得以留存。然而启用该指令后，如果连接超时，将清除所有与内存的关联。 语法：on 或 off 默认值：off `reset_timedout_connection off;`

① 有关 sendfile，请查阅 Linux 内核的相关知识，充分理解这个机制才能明白这里的意思。

指令和使用环境	描　述

续表

指令和使用环境	描　　述
tcp_nopush 使用环境：http, server, location	开启或禁用 TCP_NOPUSH(FreeBSD)或 TCP_CORK (Linux)的 socket 选项。注意，该选项仅作用于 sendfile 指令已启用的情况。如果 tcp_nopush 设置为 on，那么 Nginx 将尝试在单个 TCP 数据包中发送整个 HTTP 响应头。 语法：on 或 off 默认值：off tcp_nopush off;
sendfile 使用环境：http, server, location	如果这条指令被启用，Nginx 将使用 sendfile 内核来调用处理文件传递。如果禁用该指令，那么 Nginx 将自己处理文件传递。该选项可能会影响到服务器的性能，具体取决于被传输文件的物理位置(例如 NFS)。[①] 语法：on 或 off 默认值：off sendfile off;
sendfile_max_chunk 使用环境：http, server	本指令用于定义 sendfile 每一次调用数据的最大值。 语法：数值(size) 默认值：0
send_lowat 使用环境: http, server	该选项允许你在 TCP 套接字中利用 SO_SNDLOWAT 标志，这个选项仅在 FreeBSD 操作系统下可用。它的功能在于定义了用做输出操作的缓冲区大小。 语法：数值 (大小) 默认值：0
reset_timedout_connection 使用环境：http, server, location	当一个客户端连接超时，其相关的信息可能保留在内存，正是依赖于这种状态，相关的信息才得以留存。然而启用该指令后，如果连接超时，将清除所有与内存的关联。 语法：on 或 off 默认值：off reset_timedout_connection off;

① 有关 sendfile，请查阅 Linux 内核的相关知识，充分理解这个机制才能明白这里的意思。

路径和文档

下表描述的指令用来配置文档的根目录，每一个网站都有这样的根目录，用于存放Web服务器提供的服务内容，例如，主页、错误页等。

root 使用环境：http，server，location, if	定义文档的根目录，该目录包含你希望为访问者提供的内容。 语法：目录路径 默认值：html `root /home/website.com/public_html;`

alias
 	
使用环境：location
	alias 指令只能够放在 location 区段中，它为 Nginx 指定的文件路径提供别名，使 Nginx 为指定的请求找到访问的文件。
	作为一个例子，考虑下面的配置：
	`http {`
	`  server {`
	`  server_name localhost;`
	`  root /var/www/website.com/html;`
	`  location /admin/ {`
	`  alias /var/www/locked/;`
	`  }`
	`  }`
	`}`
	收到一个 *http://localhost/* 请求时，文件由目录/var/www/website.com/html/来提供。然而，如果 Nginx 收到的请求是 *http://localhost/admin/*，文件则由目录/var/www/locked/来提供。此外，root 指令的值是不会改变的，在动态的脚本中这一步不可见。此外，root 指令指定的文件根目录值是不能够改变的。
	语法：目录(不要忘记尾部的 /) 或文件路径

指令和使用环境	描述

路径和文档

下表描述的指令用来配置文档的根目录，每一个网站都有这样的根目录，用于存放Web服务器提供的服务内容，例如，主页、错误页等。

指令和使用环境	描　　述
root 使用环境：http，server，location, if	定义文档的根目录，该目录包含你希望为访问者提供的内容。 语法：目录路径 默认值：html `root /home/website.com/public_html;`

alias
使用环境：location
	alias 指令只能够放在 location 区段中，它为 Nginx 指定的文件路径提供别名，使 Nginx 为指定的请求找到访问的文件。
	作为一个例子，考虑下面的配置：
	`http {`
	`  server {`
	`  server_name localhost;`
	`  root /var/www/website.com/html;`
	`  location /admin/ {`
	`  alias /var/www/locked/;`
	`  }`
	`  }`
	`}`
	收到一个 *http://localhost/* 请求时，文件由目录/var/www/website.com/html/来提供。然而，如果 Nginx 收到的请求是 *http://localhost/admin/*，文件则由目录/var/www/locked/来提供。此外，root 指令的值是不会改变的，在动态的脚本中这一步不可见。此外，root 指令指定的文件根目录值是不能够改变的。
	语法：目录(不要忘记尾部的 /) 或文件路径

error_page
使用环境：http，server，location，if
	允许你修改访问 URI 所产生的 HTTP 响应代码并选择性地将它替换为其他代码。
	语法：error_page code1 [code2 …] [=replacement code] [=@block | URI]
	例如：
	`error_page 404 /not_found.html;`
	`error_page 500 501 502 503 504 /server_error.html;`
	`error_page 403 http://website.com/;`
	`error_page 404 @notfound;`
	`# 跳到命名的 location 区段`
	`error_page 404 =200 /index.html;`
	`#在出现 404 错误时，重定向到 index.html，`
	`#并且会有 200 OK 响应代码`

if_modified_since 使用环境：http，server，location	定义 Nginx 如何处理 HTTP 头中的 If-Modified-Since，这种头多数是搜索引擎蜘蛛的请求(例如 Google 页面爬虫)，会指出上一次的时间和日期，如果被请求的文件从上次爬行后没有修改过，那么服务器简单地返回一个“304 Not Modified response”代码，而不包括任何 body。 该指令接受下面三个值： ● off：忽略 If-Modified-Since 头 ● exact：如果在 HTTP 头中指定的时间和日期与实际请求的文件的修改日期准确匹配，则返回“304 Not Modified”；如果文件修改日期在指定之前或有意隐瞒，文件通常会正常送达(200 OK 响应)。 ● before：如果在 HTTP 头中指定的时间和日期在被请求文件修改日期之前或一样，则返回“304 Not Modified”。 语法：if_modified_since off | exact | before 默认值：exact `if_modified_since exact;`

指令和使用环境	描　述

续表

指令和使用环境	描　述
error_page 使用环境：http，server，location，if	允许你修改访问 URI 所产生的 HTTP 响应代码并选择性地将它替换为其他代码。 语法：error_page code1 [code2 …] [=replacement code] [=@block | URI] 例如： error_page 404 /not_found.html; error_page 500 501 502 503 504 /server_error.html; error_page 403 http://website.com/; error_page 404 @notfound; # 跳到命名的 location 区段 error_page 404 =200 /index.html; #在出现 404 错误时，重定向到 index.html， #并且会有 200 OK 响应代码
if_modified_since 使用环境：http，server，location	定义 Nginx 如何处理 HTTP 头中的 If-Modified-Since，这种头多数是搜索引擎蜘蛛的请求(例如 Google 页面爬虫)，会指出上一次的时间和日期，如果被请求的文件从上次爬行后没有修改过，那么服务器简单地返回一个“304 Not Modified response”代码，而不包括任何 body。 该指令接受下面三个值： ● off：忽略 If-Modified-Since 头 ● exact：如果在 HTTP 头中指定的时间和日期与实际请求的文件的修改日期准确匹配，则返回“304 Not Modified”；如果文件修改日期在指定之前或有意隐瞒，文件通常会正常送达(200 OK 响应)。 ● before：如果在 HTTP 头中指定的时间和日期在被请求文件修改日期之前或一样，则返回“304 Not Modified”。 语法：if_modified_since off | exact | before 默认值：exact if_modified_since exact;

index
使用环境：http，server，location
	定义一个默认的页面，如果在请求中没有指定文件名(换句话说，就是 index 页面)，Nginx 就会使用该页面提供服务。可以指定多个文件名，但使用的是第一个找到的文件。如果没有找到指定文件，Nginx 要么尝试产生一个自动的文件索引(这需要 autoindex 指令的支持，请检测 HTTP Autoindex 模块)，要么返回 403 错误页面。
	你也可以随意插入一个绝对路径的文件名(例如 /page.html，基于 root 目录指定的路径)，但只能作为指令中的最后一个参数。
	语法：index file1 [file2…] [absolute_file];
	默认值：index.html
	`index index.php index.html index.htm;`
	`index index.php index2.php /catchall.php;`

recursive_error_pages 使用环境：http，server，location	有的时候，通过指令 error_page 提供的错误页面本身也发生了错误，在这种情况下，指令 error_page 会被再次使用(递归)。该指令开启或禁用递归错误页面。 语法：on 或 off 默认值：off `recursive_error_pages off;`

try_files
使用环境：location
	试图找到指定的文件(参数 1 到 N－1)，如果指定的这些文件存在的文件都不是，就跳到命名 location 区段(最后一个参数)或者是指定的 URI。
	语法：后跟命名 location 区段或 URI 的多个文件路径。
	例如：
	`location / {`
	`  try_files $uri $uri.html $uri.php $uri.xml @proxy;`
	`}`
	`  # 下列是"named location block"`
	`location @proxy {`
	`  proxy_pass 127.0.0.1:8080;`
	`}`
	在这个例子中，Nginx 设法正常提供文件，如果请求的 URI 不符合任何现有的文件，Nginx 会给 URI 附加.html，然后再次尝试提供文件服务。如果仍然失败，则依次尝试.php 和.xml。最后，如果所有这些可能的文件都失败，则由另一个 location 块(@proxy)处理该请求

指令和使用环境	描　述

续表

指令和使用环境	描　　述

index
使用环境：http，server，location
	定义一个默认的页面，如果在请求中没有指定文件名(换句话说，就是 index 页面)，Nginx 就会使用该页面提供服务。可以指定多个文件名，但使用的是第一个找到的文件。如果没有找到指定文件，Nginx 要么尝试产生一个自动的文件索引(这需要 autoindex 指令的支持，请检测 HTTP Autoindex 模块)，要么返回 403 错误页面。
	你也可以随意插入一个绝对路径的文件名(例如 /page.html，基于 root 目录指定的路径)，但只能作为指令中的最后一个参数。
	语法：index file1 [file2…] [absolute_file];
	默认值：index.html
	`index index.php index.html index.htm;`
	`index index.php index2.php /catchall.php;`

recursive_error_pages 使用环境：http，server，location	有的时候，通过指令 error_page 提供的错误页面本身也发生了错误，在这种情况下，指令 error_page 会被再次使用(递归)。该指令开启或禁用递归错误页面。 语法：on 或 off 默认值：off `recursive_error_pages off;`

try_files
使用环境：location
	试图找到指定的文件(参数 1 到 N－1)，如果指定的这些文件存在的文件都不是，就跳到命名 location 区段(最后一个参数)或者是指定的 URI。
	语法：后跟命名 location 区段或 URI 的多个文件路径。
	例如：
	`location / {`
	`  try_files $uri $uri.html $uri.php $uri.xml @proxy;`
	`}`
	`  # 下列是"named location block"`
	`location @proxy {`
	`  proxy_pass 127.0.0.1:8080;`
	`}`
	在这个例子中，Nginx 设法正常提供文件，如果请求的 URI 不符合任何现有的文件，Nginx 会给 URI 附加.html，然后再次尝试提供文件服务。如果仍然失败，则依次尝试.php 和.xml。最后，如果所有这些可能的文件都失败，则由另一个 location 块(@proxy)处理该请求

客户端请求

下表描述的是 Nginx 处理客户端请求。在其他的事项中，可以配置 keep-alive 机制，并且可能将客户端的请求日志记录到文件中。

keepalive_requests 使用环境：http，server，location	设置在单个 keep-alive 的连接下提供的最大请求数量。 语法：数值 默认值：100 `keepalive_requests 100;`

keepalive_timeout
使用环境：http，server，location
	该指令定义一个 keep-alive 的时长，换句话说，keep-alive 能够使客户端到服务器的连接在一定时间内持续有效。在这个时间内，客户端对服务器的访问不需要再次建立连接或重新连接。
	第二个(可选)参数作为 keep-alive：timeout=HTTP 响应头传递，预期的效果是让这一段时间过去后让客户端浏览器自身关闭连接。
	注意，有些浏览器忽略这个设置，例如，Internet Explorer 会在 60 秒后自动关闭连接。
	语法：keepalive_timeout time1 [time2];
	默认值：75
	`keepalive_timeout 75;`
	`keepalive_timeout 75 60;`

send_timeout 使用环境：http，server，location	设置超时时间。一旦超过这个设置的时间，Nginx 会关闭一个不活动的连接。一个连接变为非活动状态的那一刻起，客户端停止传输数据。 语法：时间值 (秒) 默认值：60 `send_timeout 60;`

指令和使用环境	描述

客户端请求

下表描述的是 Nginx 处理客户端请求。在其他的事项中，可以配置 keep-alive 机制，并且可能将客户端的请求日志记录到文件中。

指令和使用环境	描　　述
keepalive_requests 使用环境：http，server，location	设置在单个 keep-alive 的连接下提供的最大请求数量。 语法：数值 默认值：100 `keepalive_requests 100;`

keepalive_timeout
使用环境：http，server，location
	该指令定义一个 keep-alive 的时长，换句话说，keep-alive 能够使客户端到服务器的连接在一定时间内持续有效。在这个时间内，客户端对服务器的访问不需要再次建立连接或重新连接。
	第二个(可选)参数作为 keep-alive：timeout=HTTP 响应头传递，预期的效果是让这一段时间过去后让客户端浏览器自身关闭连接。
	注意，有些浏览器忽略这个设置，例如，Internet Explorer 会在 60 秒后自动关闭连接。
	语法：keepalive_timeout time1 [time2];
	默认值：75
	`keepalive_timeout 75;`
	`keepalive_timeout 75 60;`

send_timeout 使用环境：http，server，location	设置超时时间。一旦超过这个设置的时间，Nginx 会关闭一个不活动的连接。一个连接变为非活动状态的那一刻起，客户端停止传输数据。 语法：时间值 (秒) 默认值：60 `send_timeout 60;`

client_body_in_file_only 使用环境：http，server，location	如果开启该指令，那么进入的 HTTP 请求的主体部分被存储在实际的磁盘文件中。这个客户端发送的主体(client body)相当于客户端的 HTTP 请求的原始数据，只是除去了头部(换句话说，就是 POST 请求发送的内容)。文件存为纯文本。 该指令接受下面三个值： ● off 不将请求的主体存储在文件； ● clean 将请求的主体存储在文件，但是处理请求后就移除； ● on 将请求的主体(request body)存储在文件，但是处理请求后不移除(不推荐这么设置，除非用于调试目的)； 语法：client_body_in_file_only on|clean|off 默认值：off client_body_file_only off;
client_body_in_single_buffer 使用环境：http, server, location	定义 Nginx 是否将请求的主体存储在内存中单一的缓冲区中。 语法：on 或 off 默认值：off client_body_in_single_buffer off;
client_body_buffer_size 使用环境：http, server, location	指定持有客户端请求主体的缓存大小。如果超过这个大小，那么主体(或者至少是主体的部分)将被写到磁盘。 注意，如果 client_body_in_file_only 指令开启，那么请求的主体总是会被存储在磁盘上的文件中，而不再管它们的大小(即不再管是否它们的大小与缓存合适)。 语法：数值 默认值：8 KB 或 16 KB(2 个内存页)，具体取决于结构 client_body_buffer_size 8K;

指令和使用环境	描述

续表

指令和使用环境	描　　述
client_body_in_file_only 使用环境：http，server，location	如果开启该指令，那么进入的 HTTP 请求的主体部分被存储在实际的磁盘文件中。这个客户端发送的主体(client body)相当于客户端的 HTTP 请求的原始数据，只是除去了头部(换句话说，就是 POST 请求发送的内容)。文件存为纯文本。 该指令接受下面三个值： ● off　不将请求的主体存储在文件； ● clean　将请求的主体存储在文件，但是处理请求后就移除； ● on　将请求的主体(request body)存储在文件，但是处理请求后不移除(不推荐这么设置，除非用于调试目的)； 语法：client_body_in_file_only on|clean|off 默认值：off `client_body_file_only off;`
client_body_in_single_buffer 使用环境：http, server, location	定义 Nginx 是否将请求的主体存储在内存中单一的缓冲区中。 语法：on 或 off 默认值：off `client_body_in_single_buffer off;`
client_body_buffer_size 使用环境：http, server, location	指定持有客户端请求主体的缓存大小。如果超过这个大小，那么主体(或者至少是主体的部分)将被写到磁盘。 注意，如果 client_body_in_file_only 指令开启，那么请求的主体总是会被存储在磁盘上的文件中，而不再管它们的大小(即不再管是否它们的大小与缓存合适)。 语法：数值 默认值：8 KB 或 16 KB(2 个内存页)，具体取决于结构 `client_body_buffer_size 8K;`

client_body_temp_path 使用环境：http, server, location	允许定义临时文件的路径，该文件用于存储客户端请求的主体部分。 另外一个选项可以将这些文件分散到一个层次结构中，最多三层。 语法：client_body_temp_path path [level1] [level2] [level3] 默认值：client_body_temp client_body_temp_path /tmp/nginx_rbf; client_body_temp_path temp 2; # Nginx 将建立一个两数位的目录来存放 #请求体(request #body)文件 client_body_temp_path temp 1 2 4; # Nginx 将建立一个 3 级目录 #(第一级: 1 个数字, 第二级: # 2 个数字,第三级: 4 个数字)
client_body_timeout 使用环境：http，server，location	在读取一个客户端请求主体的时候，定义一个非活动的超时间隔。在一个连接变为无效时，从那一刻起停止传输数据，如果客户端请求延时到达，Nginx 将返回一个 408 表示 HTTP 请求超时错误。 语法：时间值(秒) 默认值：60 send_timeout 60;
client_header_buffer_size 使用环境：http，server，location	该指令用于定义 Nginx 将缓存区分配给请求的头部的大小。通常 1 就够了，然而，在有些情况下，请求的头部(header)可能包含大的 cookie 数据或较长的 URI。如果属于这种情况， Nginx 会将一个或多个缓冲区分配给请求头使用(大的缓冲区的大小通过 large_client_header_buffers 来指定)。 语法：数值 默认值：1 K client_header_buffer_size 1K;
client_header_timeout 使用环境：http，server，location	在读一个客户端请求头时，定义不活动超时的时长。一个连接变为无效的连接的那时起，客户端就停止传输数据。如果延迟到达，Nginx 返回一个 HTTP 错误表示 408 请求超时。 语法：时间值 (秒) 默认值：60 send_timeout 60;

指令和使用环境	描述

续表

指令和使用环境	描　　述
client_body_temp_path 使用环境：http, server, location	允许定义临时文件的路径，该文件用于存储客户端请求的主体部分。 另外一个选项可以将这些文件分散到一个层次结构中，最多三层。 语法：client_body_temp_path path [level1] [level2] [level3] 默认值：client_body_temp client_body_temp_path /tmp/nginx_rbf; client_body_temp_path temp 2; # Nginx 将建立一个两数位的目录来存放 #请求体(request #body)文件 client_body_temp_path temp 1 2 4; # Nginx 将建立一个 3 级目录 #(第一级: 1 个数字, 第二级: # 2 个数字,第三级: 4 个数字)
client_body_timeout 使用环境：http，server，location	在读取一个客户端请求主体的时候，定义一个非活动的超时间隔。在一个连接变为无效时，从那一刻起停止传输数据，如果客户端请求延时到达，Nginx 将返回一个 408 表示 HTTP 请求超时错误。 语法：时间值(秒) 默认值：60 send_timeout 60;
client_header_buffer_size 使用环境：http，server，location	该指令用于定义 Nginx 将缓存区分配给请求的头部的大小。通常 1 就够了，然而，在有些情况下，请求的头部(header)可能包含大的 cookie 数据或较长的 URI。如果属于这种情况， Nginx 会将一个或多个缓冲区分配给请求头使用(大的缓冲区的大小通过 large_client_header_buffers 来指定)。 语法：数值 默认值：1 K client_header_buffer_size 1K;
client_header_timeout 使用环境：http，server，location	在读一个客户端请求头时，定义不活动超时的时长。一个连接变为无效的连接的那时起，客户端就停止传输数据。如果延迟到达，Nginx 返回一个 HTTP 错误表示 408 请求超时。 语法：时间值 (秒) 默认值：60 send_timeout 60;

client_max_body_size 使用环境：http，server，location	利用该指令定义客户端请求体的最大值，如果超过这个值，Nginx 会返回 413 请求实体太大的 HTTP 错误。如果打算让用户向 HTTP 服务器上传文件，那么这个设置很重要。 语法：Size value 默认值：1 M; `client_max_body_size 1M;`
large_client_header_buffers 使用环境：http，server，location	定义数量和缓冲区的大小，以便存储客户端的请求。假如默认的缓存(client_header_buffer_size)不足。头(header)的每一行必须与单个缓存相合适，如果请求的 URI 行大于单个 buffer 的大小，那么 Nginx 就会返回“414 Request URI too large”错误；如果另一个头行超过单个 buffer 的大小，那么 Nginx 会返回一个“400 Bad request”错误。 语法：large_client_header_buffers amount size 默认值：4 个 4K 或 8K 的缓冲(1 个内存页，大小依赖于你的结构) `large_client_header_buffers 4 4K;`
lingering_time 使用环境：http，server，location	该指令会将一个请求体运用到客户端请求。上传的数据一旦超过 max_client_body_size 指定的值，Nginx 就会立即发送“413 Request entity too large”的 HTTP 响应错误。然而，大多数浏览器不管哪个通知，继续上传数据。该指令定义了 Nginx 应该等待的时间总数，包括从发送这个错误响应之后到关闭该连接之前的时间。 语法：数值(时间) 默认值：30 seconds

续表

指令和使用环境	描述

续表

指令和使用环境	描　述
client_max_body_size 使用环境：http，server，location	利用该指令定义客户端请求体的最大值，如果超过这个值，Nginx 会返回 413 请求实体太大的 HTTP 错误。如果打算让用户向 HTTP 服务器上传文件，那么这个设置很重要。 语法：Size value 默认值：1 M; `client_max_body_size 1M;`
large_client_header_buffers 使用环境：http，server，location	定义数量和缓冲区的大小，以便存储客户端的请求。假如默认的缓存(client_header_buffer_size)不足。头(header)的每一行必须与单个缓存相合适，如果请求的 URI 行大于单个 buffer 的大小，那么 Nginx 就会返回“414 Request URI too large”错误；如果另一个头行超过单个 buffer 的大小，那么 Nginx 会返回一个“400 Bad request”错误。 语法：large_client_header_buffers amount size 默认值：4 个 4K 或 8K 的缓冲(1 个内存页，大小依赖于你的结构) `large_client_header_buffers 4 4K;`
lingering_time 使用环境：http，server，location	该指令会将一个请求体运用到客户端请求。上传的数据一旦超过 max_client_body_size 指定的值，Nginx 就会立即发送“413 Request entity too large”的 HTTP 响应错误。然而，大多数浏览器不管哪个通知，继续上传数据。该指令定义了 Nginx 应该等待的时间总数，包括从发送这个错误响应之后到关闭该连接之前的时间。 语法：数值(时间) 默认值：30 seconds

lingering_timeout 使用环境：http，server，location	该指令定义 Nginx 在客户端关闭之前，两个读操作之间的等待时间。 语法：数值(时间) 默认值：5 seconds
ignore_invalid_headers 使用环境：http，server	如果禁用该指令，Nginx 会返回一个 400 错误的 HTTP 请求，以免请求头被错误形成。 语法：on 或 off 默认值：on

MIME 类型

Nginx 提供了两个特有的指令，这两个指令用于帮助你配置 MIME 的类型：types 和 default_type，用于对 Nginx 提供的文档定义默认 MIME 类型，这个设置会影响响应数据包 HTTP 头中的 Content-Type。

types
使用环境：http，server，location
	该指令允许你在 MIME 类型和文件扩展名之间建立联系。它实际上是接受特定语法的区段：
	`types {`
	`  mimetype1 extension1;`
	`  mimetype2 extension2 [extension3…];`
	`  […]`
	`}`
	Nginx 在提供文件服务时，它会检查文件的扩展名，以便决定 MIME 的类型。然后，在响应数据包中发送，MIME 的类型会在 HTTP 的头中 Content-Type 部分发送。这个头通常影响着浏览器对客户端的处理。例如，如果 MIME 类型是 application/octet-stream，那么浏览器会下载该文件而非显示它；如果 MIME 类型是 text/plain，那么该文件在浏览器中会被作为纯文本显示而不用 HTML 解释提供。
	Nginx 包括一套基本 MIME 类型，作为一个单独的文件(mime.types)保存。要包括该文件，使用如下的指令即可：
	`include mime.types;`

指令和使用环境	描述

指令和使用环境	描述

续表

指令和使用环境	描　述
lingering_timeout 使用环境：http，server，location	该指令定义 Nginx 在客户端关闭之前，两个读操作之间的等待时间。 语法：数值(时间) 默认值：5 seconds
ignore_invalid_headers 使用环境：http，server	如果禁用该指令，Nginx 会返回一个 400 错误的 HTTP 请求，以免请求头被错误形成。 语法：on 或 off 默认值：on

MIME 类型

Nginx 提供了两个特有的指令，这两个指令用于帮助你配置 MIME 的类型：types 和 default_type，用于对 Nginx 提供的文档定义默认 MIME 类型，这个设置会影响响应数据包 HTTP 头中的 Content-Type。

指令和使用环境	描　述

types
使用环境：http，server，location
	该指令允许你在 MIME 类型和文件扩展名之间建立联系。它实际上是接受特定语法的区段：
	`types {`
	`  mimetype1 extension1;`
	`  mimetype2 extension2 [extension3…];`
	`  […]`
	`}`
	Nginx 在提供文件服务时，它会检查文件的扩展名，以便决定 MIME 的类型。然后，在响应数据包中发送，MIME 的类型会在 HTTP 的头中 Content-Type 部分发送。这个头通常影响着浏览器对客户端的处理。例如，如果 MIME 类型是 application/octet-stream，那么浏览器会下载该文件而非显示它；如果 MIME 类型是 text/plain，那么该文件在浏览器中会被作为纯文本显示而不用 HTML 解释提供。
	Nginx 包括一套基本 MIME 类型，作为一个单独的文件(mime.types)保存。要包括该文件，使用如下的指令即可：
	`include mime.types;`

types
使用环境：http，server，location
	该文件已经包含大多数重要的文件扩展名，因此可能不用修改编辑它。如果你提供的文件扩展名没有包含在该文件中，则使用默认的类型，例如下面通过 default_type 指令指出的类型。
	
	注意，可以通过重新声明来覆盖一个指定类型，一个有用的例子是强迫一个目录中的文件都用于下载而非显示：
	
	`http {`
	`  include mime.types;`
	`  […]`
	`  location /downloads/ {`
	`  # 移除所有的 MIME 类型`
	`  types { }`
	`  default_type application/octet-stream;`
	`  }`
	`  […]`
	`}`
	
	注意，如果它们的扩展名众所周知，例如.html 或.txt，一些浏览器会忽略 MIME 类型，可以仍然显示文件。如果 mime.types 没有被包括，那么默认值为：
	
	`types {`
	`  text/html html;`
	`  image/gif gif;`
	`  image/jpeg jpg;`
	`}`

default_type 使用环境：http、server 和 location	定义默认的 MIME 类型。Nginx 提供一个文件时，该文件的类型(通过扩展名来辨别)将与 MIME 文件中声明的类型相匹配，然后返回 MIME 类型的值，该值将填入 HTTP 响应头中的 Content-Type 部分。如果提供的文件扩展名与任何所知的 MIME 类型都不匹配，那么响应头中的值将由 default_type 的值提供。 语法：MIME type 默认值：text/plain `default_type text/plain;`
types_hash_max_size 使用环境：http、server 和 location	在 MIME 类型哈希表中定义一个条目的最大值。 默认值：4 k 或 8 k (CPU 缓冲的一行)

指令和使用环境	描述

续表

指令和使用环境	描　　述

types
使用环境：http，server，location
	该文件已经包含大多数重要的文件扩展名，因此可能不用修改编辑它。如果你提供的文件扩展名没有包含在该文件中，则使用默认的类型，例如下面通过 default_type 指令指出的类型。
	
	注意，可以通过重新声明来覆盖一个指定类型，一个有用的例子是强迫一个目录中的文件都用于下载而非显示：
	
	`http {`
	`  include mime.types;`
	`  […]`
	`  location /downloads/ {`
	`  # 移除所有的 MIME 类型`
	`  types { }`
	`  default_type application/octet-stream;`
	`  }`
	`  […]`
	`}`
	
	注意，如果它们的扩展名众所周知，例如.html 或.txt，一些浏览器会忽略 MIME 类型，可以仍然显示文件。如果 mime.types 没有被包括，那么默认值为：
	
	`types {`
	`  text/html html;`
	`  image/gif gif;`
	`  image/jpeg jpg;`
	`}`

default_type 使用环境：http、server 和 location	定义默认的 MIME 类型。Nginx 提供一个文件时，该文件的类型(通过扩展名来辨别)将与 MIME 文件中声明的类型相匹配，然后返回 MIME 类型的值，该值将填入 HTTP 响应头中的 Content-Type 部分。如果提供的文件扩展名与任何所知的 MIME 类型都不匹配，那么响应头中的值将由 default_type 的值提供。 语法：MIME type 默认值：text/plain `default_type text/plain;`
types_hash_max_size 使用环境：http、server 和 location	在 MIME 类型哈希表中定义一个条目的最大值。 默认值：4 k 或 8 k (CPU 缓冲的一行)

限制和约束

客户端尝试访问服务器上的某一特定 location 或文档时，可以通过这一组指令添加额外的约束。注意，在第 5 章将介绍其他用于访问约束的指令。

limit_except
使用环境：location

该指令允许你阻止使用所有 HTTP 方法，除非明确允许一些命令。

在一个 location 区段内，你可能想约束一些 HTTP 使用方法，例如禁止客户端发送 POST 请求。需要定义两个元素：首先是不禁止的方法(允许的方法，所有其他方法将被禁止)；其次是受此影响的访问者。

```
location /admin/ {
    limit_except GET {
        allow 192.168.1.0/24;
        deny all;
    }
}
```

在这个例子中，约束添加到 location 区段的 /admin/，所有的访问者只允许使用 GET 方法。而访问者需要为本地 IP 地址，通过 allow 指令指定的客户端(详情参见 HTTP Access 模块)则不受此限制。如果一个访问者使用禁用的方法， Nginx 会返回 403 禁止 HTTP 错误。注意，GET 方法暗含 HEAD 方法(如果允许 GET，意味着允许 GET 和 HEAD 这两种方法)。

它的语法比较特别:

```
limit_except METHOD1 [METHOD2…] {
allow | deny | auth_basic | auth_basic_user_file
| proxy_pass | perl;
}
```

就是说，limit_except 区段接受一些特定的指令。例如，不能将 rewrite 指令插入 limit_except 区段。
可插入的指令列表如下：

```
limit_except METHOD1 [METHOD2…] {
  allow|deny|auth_basic|auth_basic_user_file
| proxy_pass | perl;
}
```

因此，如果想进一步了解这些指令(allow，deny，auth_basic...)，可以查看提供相关指令的模块，这些模块分布在各章

指令和使用环境	描述

限制和约束

客户端尝试访问服务器上的某一特定 location 或文档时，可以通过这一组指令添加额外的约束。注意，在第 5 章将介绍其他用于访问约束的指令。

指令和使用环境	描　　述

limit_except
使用环境：location
	该指令允许你阻止使用所有 HTTP 方法，除非明确允许一些命令。
	
	在一个 location 区段内，你可能想约束一些 HTTP 使用方法，例如禁止客户端发送 POST 请求。需要定义两个元素：首先是不禁止的方法(允许的方法，所有其他方法将被禁止)；其次是受此影响的访问者。
	
	`location /admin/ {`
	`  limit_except GET {`
	`  allow 192.168.1.0/24;`
	`  deny all;`
	`  }`
	`}`
	
	在这个例子中，约束添加到 location 区段的 /admin/，所有的访问者只允许使用 GET 方法。而访问者需要为本地 IP 地址，通过 allow 指令指定的客户端(详情参见 HTTP Access 模块)则不受此限制。如果一个访问者使用禁用的方法， Nginx 会返回 403 禁止 HTTP 错误。注意，GET 方法暗含 HEAD 方法(如果允许 GET，意味着允许 GET 和 HEAD 这两种方法)。
	
	它的语法比较特别:
	
	`limit_except METHOD1 [METHOD2…] {`
	`allow | deny | auth_basic | auth_basic_user_file`
	`| proxy_pass | perl;`
	`}`
	
	就是说，limit_except 区段接受一些特定的指令。例如，不能将 rewrite 指令插入 limit_except 区段。
	可插入的指令列表如下：
	
	`limit_except METHOD1 [METHOD2…] {`
	`  allow|deny|auth_basic|auth_basic_user_file`
	`| proxy_pass | perl;`
	`}`
	
	因此，如果想进一步了解这些指令(allow，deny，auth_basic...)，可以查看提供相关指令的模块，这些模块分布在各章

limit_rate 使用环境：http，server，location，if	允许对个别客户端进行传输率限制，传输率以每秒传输的字节数来表示，示例如下： `limit_rate 500k;` 前面例子中，限制为每秒 500k/s，如果客户端打开两个连接，那么该客户端被允许的传输率为 2 * 500k。 语法：大小值 默认值：无
limit_rate_after 使用环境：http，server，location，if	定义在指令 limit_rate 生效之前传输数据的总数，示例如下： `limit_rate_after 10m;` 在前面的例子中，Nginx 将以最大的速率发送 10MB 的数据，超过这个大小后，通过 limit_rate 指令限制的值将起作用(参考前文)。类似于指令 limit_rate，这个设置只应用于单个连接。 语法：大小值 默认值：无

satisfy
使用环境：location
	该指令定义了客户端是否需要所有访问条件都有效(satisfy all)或者是至少一个有效(satisfy any)才能被允许访问。
	`location /admin/ {`
	`  allow 192.168.1.0/24;`
	`  deny all;`
	`  auth_basic "Authentication required";`
	`  auth_basic_user_file conf/htpasswd;`
	`}`
	在前面的例子中，客户端能够访问该资源必须具备两个条件。
	①通过 allow 和 deny 指令(HTTP Access 模块)，我们将只允许具有本地 IP 的客户端，其他 IP 的客户端将拒绝访问。
	②通过 auth_basic 和 auth_basic_user_file 指令(HTTP Auth_basic 模块)，我们将只允许能够提供用户名和密码的用户访问。
	使用“satisfy all”，客户端必须满足这两个条件才能够访问该资源；使用“satisfy any”，客户端只需满足任意一个条件就可以访问该资源。
	语法：satisfy any | all
	默认值：all
	`satisfy all;`

指令和使用环境	描述

续表

指令和使用环境	描　　述
limit_rate 使用环境：http，server，location，if	允许对个别客户端进行传输率限制，传输率以每秒传输的字节数来表示，示例如下： `limit_rate 500k;` 前面例子中，限制为每秒 500k/s，如果客户端打开两个连接，那么该客户端被允许的传输率为 2 * 500k。 语法：大小值 默认值：无
limit_rate_after 使用环境：http，server，location，if	定义在指令 limit_rate 生效之前传输数据的总数，示例如下： `limit_rate_after 10m;` 在前面的例子中，Nginx 将以最大的速率发送 10MB 的数据，超过这个大小后，通过 limit_rate 指令限制的值将起作用(参考前文)。类似于指令 limit_rate，这个设置只应用于单个连接。 语法：大小值 默认值：无

satisfy
使用环境：location
	该指令定义了客户端是否需要所有访问条件都有效(satisfy all)或者是至少一个有效(satisfy any)才能被允许访问。
	`location /admin/ {`
	`  allow 192.168.1.0/24;`
	`  deny all;`
	`  auth_basic "Authentication required";`
	`  auth_basic_user_file conf/htpasswd;`
	`}`
	在前面的例子中，客户端能够访问该资源必须具备两个条件。
	①通过 allow 和 deny 指令(HTTP Access 模块)，我们将只允许具有本地 IP 的客户端，其他 IP 的客户端将拒绝访问。
	②通过 auth_basic 和 auth_basic_user_file 指令(HTTP Auth_basic 模块)，我们将只允许能够提供用户名和密码的用户访问。
	使用“satisfy all”，客户端必须满足这两个条件才能够访问该资源；使用“satisfy any”，客户端只需满足任意一个条件就可以访问该资源。
	语法：satisfy any | all
	默认值：all
	`satisfy all;`

internal
使用环境：
location
	该指令指定 location 区段只能用于内部访问，换句话说，由该指令指定的资源对外部请求是不能访问的。
	`server {`
	`  [...]`
	`  server_name .website.com;`
	`  location /admin/ {`
	`  internal;`
	`  }`
	`}`
	前面的配置中，客户端将无法浏览 *http://website.com/admin/*，这样的访问将遇到 404 没有找到的错误。要访问该资源，只有一个方法，即通过内部重定向(详情参见 Rewrite 模块)

文件处理和缓存

在坚实的基础之上建立网站很重要，文件访问和缓存对 Web 服务起着关键的作用。基于这个观点，Nginx 允许我们通过下表所示的指令进行调整。

direction 使用环境：http，server，location	如果启用该指令，需要指定一个值，大于该值的文件将通过 Direct I/O 系统机制读取。这便允许 Nginx 从存储驱动器上读取数据并且直接放入内存，而不会经过复杂的中间缓冲处理。启用该指令后，sendfile 指令将自动禁用，它们不能够一起使用。 语法：大小值 或 off 默认值：off `Direction 5m;`

指令和使用环境	描述

指令和使用环境	描述

续表

指令和使用环境	描　　述

internal
使用环境：
location
	该指令指定 location 区段只能用于内部访问，换句话说，由该指令指定的资源对外部请求是不能访问的。
	`server {`
	`[…]`
	`server_name .website.com;`
	`location /admin/ {`
	`internal;`
	`}`
	`}`
	前面的配置中，客户端将无法浏览 *http://website.com/admin/*，这样的访问将遇到 404 没有找到的错误。要访问该资源，只有一个方法，即通过内部重定向(详情参见 Rewrite 模块)

文件处理和缓存

在坚实的基础之上建立网站很重要，文件访问和缓存对 Web 服务起着关键的作用。基于这个观点，Nginx 允许我们通过下表所示的指令进行调整。

指令和使用环境	描　　述
direction 使用环境：http，server，location	如果启用该指令，需要指定一个值，大于该值的文件将通过 Direct I/O 系统机制读取。这便允许 Nginx 从存储驱动器上读取数据并且直接放入内存，而不会经过复杂的中间缓冲处理。启用该指令后，sendfile 指令将自动禁用，它们不能够一起使用。 语法：大小值 或 off 默认值：off `Direction 5m;`

open_file_cache 使用环境：http，server，location	该指令允许启用缓存，该缓存用于存储被打开文件的相关信息，通俗的说，实际上不是存储文件内容本身，而是与文件相关的信息，例如：文件描述符（文件大小，修改时间，等等）；文件和目录存在的信息；文件错误，例如，权限读取错误，文件没有找到，等等。注意，这些可以通过 open_file_cache_errors 指令来禁用。 该指令接受两个参数。 ①max=X，这里的 X 是能够缓存的最大条目。如果达到这个数量，老的条目将被删除，从而给新的条目留下存储空间； ②可选参数 inactive=Y，这里的 Y 是设定每一个被缓冲的条目可存储的时间长度。默认情况下，Nginx 将等待 60s，然后条目被清除。如果访问的是缓存条目，则重置其定时器（换句话说，就是将这个值再次改为 60s），如果一个条目被访问的次数多于指令 open_file_cache_min_uses 指定的值，缓存的条目则不会被清理(直到 Nginx 用完缓存空间而决定清除旧的条目)。 语法：open_file_cache max=X [inactive=Y] | off 默认值：off `open_file_cache max=5000 inactive=180;`
open_file_cache_errors 使用环境：http，server，location	开启或禁用缓存文件错误，由指令 open_file_cache 开启对文件的缓存(参考前文)。 语法：on 或 off 默认值：off `open_file_cache_errors on;`

指令和使用环境	描述

续表

指令和使用环境	描　　述
open_file_cache 使用环境：http，server，location	该指令允许启用缓存，该缓存用于存储被打开文件的相关信息，通俗的说，实际上不是存储文件内容本身，而是与文件相关的信息，例如：文件描述符（文件大小，修改时间，等等）；文件和目录存在的信息；文件错误，例如，权限读取错误，文件没有找到，等等。注意，这些可以通过 open_file_cache_errors 指令来禁用。 该指令接受两个参数。 ①max=X，这里的 X 是能够缓存的最大条目。如果达到这个数量，老的条目将被删除，从而给新的条目留下存储空间； ②可选参数 inactive=Y，这里的 Y 是设定每一个被缓冲的条目可存储的时间长度。默认情况下，Nginx 将等待 60s，然后条目被清除。如果访问的是缓存条目，则重置其定时器（换句话说，就是将这个值再次改为 60s），如果一个条目被访问的次数多于指令 open_file_cache_min_uses 指定的值，缓存的条目则不会被清理(直到 Nginx 用完缓存空间而决定清除旧的条目)。 语法：open_file_cache max=X [inactive=Y] | off 默认值：off `open_file_cache max=5000 inactive=180;`
open_file_cache_errors 使用环境：http，server，location	开启或禁用缓存文件错误，由指令 open_file_cache 开启对文件的缓存(参考前文)。 语法：on 或 off 默认值：off `open_file_cache_errors on;`

open_file_cache_min_uses 使用环境：http，server，location	为了保护条目，该指令定义条目总访问次数。默认情况下，在 open_file_cache 中的条目会在闲置一段时间后(默认是 60s)被清除，但如果该条目是活动的，则可以阻止 Nginx 移除该缓存条目。 `open_file_cache_min_uses 3;` 如果一个缓存条目访问的次数超过 3 次，就会变为永久活动条目，而不会被移除，除非是 Nginx 决定清除旧的条目而释放一些空间后才会删除。 语法：数值 默认值：1
open_file_cache_valid 使用环境：http，server，location	启用文件缓存机制很重要，但是缓存的信息很快就变得过时，尤其对于移动迅速的文件系统(fast-moving filesystem)。从这个角度讲，过了一段时间后，信息需要重新校验，该指令指定在缓存重新有效之前 Nginx 将等待的时间。 语法：时间值(秒) 默认值：60 `open_file_cache_valid 60;`

其他指令

下表所示的指令与 Web 服务器的各个方面有关：记录，URI 组成，DNS，等等。

log_not_found 使用环境：http，server，location	开启或禁用 404 没有找到的 HTTP 错误，如果日志中充满 404 错误，却由于找不到 favicon.ico 或 robots.txt 文件，那么你可以将这个选项关闭掉。 语法：on 或 off 默认值：on `log_not_found on;`
log_subrequest 使用环境：http，server，location	启用或禁用记录子请求，被内部指令(参考 Rewrite 模块)或是 SSI 请求(参考 Server Side Includes 模块)触发的请求。 语法：on 或 off 默认值：off `log_subrequest off;`

指令和使用环境	描　述

指令和使用环境	描述

续表

指令和使用环境	描　　述
open_file_cache_min_uses 使用环境：http，server，location	为了保护条目，该指令定义条目总访问次数。默认情况下，在 open_file_cache 中的条目会在闲置一段时间后(默认是 60s)被清除，但如果该条目是活动的，则可以阻止 Nginx 移除该缓存条目。 `open_file_cache_min_uses 3;` 如果一个缓存条目访问的次数超过 3 次，就会变为永久活动条目，而不会被移除，除非是 Nginx 决定清除旧的条目而释放一些空间后才会删除。 语法：数值 默认值：1
open_file_cache_valid 使用环境：http，server，location	启用文件缓存机制很重要，但是缓存的信息很快就变得过时，尤其对于移动迅速的文件系统(fast-moving filesystem)。从这个角度讲，过了一段时间后，信息需要重新校验，该指令指定在缓存重新有效之前 Nginx 将等待的时间。 语法：时间值(秒) 默认值：60 `open_file_cache_valid 60;`

其他指令

下表所示的指令与 Web 服务器的各个方面有关：记录，URI 组成，DNS，等等。

指令和使用环境	描　　述
log_not_found 使用环境：http，server，location	开启或禁用 404 没有找到的 HTTP 错误，如果日志中充满 404 错误，却由于找不到 favicon.ico 或 robots.txt 文件，那么你可以将这个选项关闭掉。 语法：on 或 off 默认值：on `log_not_found on;`
log_subrequest 使用环境：http，server，location	启用或禁用记录子请求，被内部指令(参考 Rewrite 模块)或是 SSI 请求(参考 Server Side Includes 模块)触发的请求。 语法：on 或 off 默认值：off `log_subrequest off;`

merge_slashes
使用环境：http，server，location
	启用该指令将在 URI 中实现一个具有合并多个连续斜线的效果，在以下情况中，这个指令特别有用：
	`server {`
	`  [...]`
	`  server_name website.com;`
	`  location /documents/ {`
	`  type { }`
	`  default_type text/plain;`
	`  }`
	`}`
	默认情况下，如果客户端试图访问 *http://website.com//documents/*(注意，在 URI 中有两个//)，那么 Nginx 将返回一个 404 错误代码，表示文件没有找到的 HTTP 错误。如果启用该指令，则两个斜线合并为一个，然后会与 location 匹配。
	语法：on 或 off
	默认值：off
	`merge_slashes off;`

msie_padding 使用环境：http，server，location	该指令是专门为 Microsoft Internet Explorer 浏览器设计的，对于错误页面来说(错误代码为 400 或更高值)，如果响应体(response body)少于 512 字节，这些浏览器会显示字节的错误页，有时候会由服务器提供更多的信息页。如果开启该选项，状态代码为 400 或更高值的响应体将填满到 512 个字节。 语法：on 或 off 默认值：off `msie_padding off;`
msie_refresh 使用环境：http，server，location	这是另一个 MSIE 特殊指令，它在 HTTP 响应代码为“301 Moved permanently”和“302 Moved temporarily”时生效。如果将该指令开启，则 Nginx 发送给运行 MSIE 浏览器的客户端的响应体(response body)中会包含一个 refresh meta 标签(<meta http-equiv="Refresh"…>)，旨在将浏览器请求的资源重定向到一个新的位置。 语法：on 或 off 默认值：off `msie_refresh off;`

续表

指令和使用环境	描述

续表

指令和使用环境	描　述

merge_slashes
使用环境：http，server，location
	启用该指令将在 URI 中实现一个具有合并多个连续斜线的效果，在以下情况中，这个指令特别有用：
	`server {`
	`  [...]`
	`  server_name website.com;`
	`  location /documents/ {`
	`  type { }`
	`  default_type text/plain;`
	`  }`
	`}`
	默认情况下，如果客户端试图访问 *http://website.com//documents/*(注意，在 URI 中有两个//)，那么 Nginx 将返回一个 404 错误代码，表示文件没有找到的 HTTP 错误。如果启用该指令，则两个斜线合并为一个，然后会与 location 匹配。
	语法：on 或 off
	默认值：off
	`merge_slashes off;`

msie_padding 使用环境：http，server，location	该指令是专门为 Microsoft Internet Explorer 浏览器设计的，对于错误页面来说(错误代码为 400 或更高值)，如果响应体(response body)少于 512 字节，这些浏览器会显示字节的错误页，有时候会由服务器提供更多的信息页。如果开启该选项，状态代码为 400 或更高值的响应体将填满到 512 个字节。 语法：on 或 off 默认值：off `msie_padding off;`
msie_refresh 使用环境：http，server，location	这是另一个 MSIE 特殊指令，它在 HTTP 响应代码为“301 Moved permanently”和“302 Moved temporarily”时生效。如果将该指令开启，则 Nginx 发送给运行 MSIE 浏览器的客户端的响应体(response body)中会包含一个 refresh meta 标签(<meta http-equiv="Refresh"…>)，旨在将浏览器请求的资源重定向到一个新的位置。 语法：on 或 off 默认值：off `msie_refresh off;`

resolver 使用环境：http，server，location	指定域名服务器，Nginx 会使用它将主机名解析到 IP，反之亦然。 语法：IP address 默认值：None (系统默认) `resolver 127.0.0.1; #使用本地 DNS`
resolver_timeout 使用环境：http，server，location	域名查询超时时间。 语法：时间值 (秒) 默认值：30 `resolver_timeout 30s;`
server_tokens 使用环境：http，server，location	该指令允许你确定 Nginx 是否告知客户端其运行的版本号。Nginx 有两个位置能够指出其版本号： ①在服务器响应头中(例如 nginx/0.7.66)。如果设置 server_tokens 的值为 off，服务器的头将只指示为 Nginx； ②在错误页上，Nginx 会在页脚指出版本号。如果设置 server_tokens 为 off，那么错误页的页脚之后显示 nginx。 如果运行着 Nginx 的老版本，并且不再打算对其升级，那么隐藏其版本号会是个不错的想法 语法：on 或 off 默认值：on `server_tokens on;`
underscores_in_headers 使用环境：http，server	在自定义 HTTP 头名称中启用或禁用下划线。如果将该指令设置为 on，那么在下列例子中，Nginx 会认为头是有效的：test_header: value。 语法：on 或 off 默认值：off `underscores_in_headers off;`
variables_hash_max_size 使用环境：http	该指令定义了变量哈希表存放变量条目的最大数量。如果服务器配置使用的变量总数超过 512 个，则需要增加该值。 语法：时间值 默认值：512

指令和使用环境	描　述

续表

指令和使用环境	描　述
resolver 使用环境：http，server，location	指定域名服务器，Nginx 会使用它将主机名解析到 IP，反之亦然。 语法：IP address 默认值：None (系统默认) `resolver 127.0.0.1; #使用本地 DNS`
resolver_timeout 使用环境：http，server，location	域名查询超时时间。 语法：时间值 (秒) 默认值：30 `resolver_timeout 30s;`
server_tokens 使用环境：http，server，location	该指令允许你确定 Nginx 是否告知客户端其运行的版本号。Nginx 有两个位置能够指出其版本号： ①在服务器响应头中(例如 nginx/0.7.66)。如果设置 server_tokens 的值为 off，服务器的头将只指示为 Nginx； ②在错误页上，Nginx 会在页脚指出版本号。如果设置 server_tokens 为 off，那么错误页的页脚之后显示 nginx。 如果运行着 Nginx 的老版本，并且不再打算对其升级，那么隐藏其版本号会是个不错的想法 语法：on 或 off 默认值：on `server_tokens on;`
underscores_in_headers 使用环境：http，server	在自定义 HTTP 头名称中启用或禁用下划线。如果将该指令设置为 on，那么在下列例子中，Nginx 会认为头是有效的：test_header: value。 语法：on 或 off 默认值：off `underscores_in_headers off;`
variables_hash_max_size 使用环境：http	该指令定义了变量哈希表存放变量条目的最大数量。如果服务器配置使用的变量总数超过 512 个，则需要增加该值。 语法：时间值 默认值：512

variables_hash_bucket_size 使用环境：http	定义存放在变量哈希表中变量的最大长度，如果变量名长于 64 个字符，则必须增大该值。 语法：数值 默认值：64 (或 32 或 128 依赖于处理器缓存)

post_action
使用环境：http，server，location，if
	定义一个请求完成之后的动作。请求完成之后，Nginx 将调用该 URI。
	语法：URI 或命名 location 区段
	例如：
	`location /payment/ {`
	`  post_action /scripts/done.php;`
	`}`

模块变量

HTTP 核心模块引入了大量变量，你可以使用指示值范围内的值，但要小心使用，因为只有少数指令接受变量定义的值。如果在一个不接受该参数的指令中使用了该参数，则不会有任何错误报告，相反，它会显示变量名称的原始文本。

在这里会碰到三种类型的变量。第一是客户端请求头(request header)中发送的值；第二是为响应客户端而发送响应头(response header)中的值；第三是有 Nginx 产生的各种变量。

请求头

Nginx 可以让你访问客户端的 HTTP 请求头(request header)，下表中的变量，可以日后用于配置文件中。

$http_host	HTTP 头中 Host 的值，这个值是一个字符串，它指出客户端设法要到达主机的主机名
$http_user_agent	HTTP 头中 User-Agent 的值，该字符串指出客户端使用的 Web 浏览器
$http_referer	HTTP 头中 Referer 的值，该字符串告诉服务器从哪个页面链接而来

指令和使用环境	描　　述

变量名称	描述

续表

指令和使用环境	描　　述
variables_hash_bucket_size 使用环境：http	定义存放在变量哈希表中变量的最大长度，如果变量名长于 64 个字符，则必须增大该值。 语法：数值 默认值：64 (或 32 或 128 依赖于处理器缓存)

post_action
使用环境：http，server，location，if
	定义一个请求完成之后的动作。请求完成之后，Nginx 将调用该 URI。
	语法：URI 或命名 location 区段
	例如：
	`location /payment/ {`
	`  post_action /scripts/done.php;`
	`}`

模块变量

HTTP 核心模块引入了大量变量，你可以使用指示值范围内的值，但要小心使用，因为只有少数指令接受变量定义的值。如果在一个不接受该参数的指令中使用了该参数，则不会有任何错误报告，相反，它会显示变量名称的原始文本。

在这里会碰到三种类型的变量。第一是客户端请求头(request header)中发送的值；第二是为响应客户端而发送响应头(response header)中的值；第三是有 Nginx 产生的各种变量。

请求头

Nginx 可以让你访问客户端的 HTTP 请求头(request header)，下表中的变量，可以日后用于配置文件中。

变量名称	描　　述
$http_host	HTTP 头中 Host 的值，这个值是一个字符串，它指出客户端设法要到达主机的主机名
$http_user_agent	HTTP 头中 User-Agent 的值，该字符串指出客户端使用的 Web 浏览器
$http_referer	HTTP 头中 Referer 的值，该字符串告诉服务器从哪个页面链接而来

$http_via	HTTP 头中 Via 的值，该值会告知我们客户端可能使用的代理
$http_x_forwarded_for	HTTP 头中 X-Forwarded-For 的值，如果客户端在代理之后，通过该变量能够取得真实的客户端 IP 地址
$http_cookie	HTTP 头中 cookie 的值，该变量中包含客户端发送的 cookie 数据
$http_...	获取其他通过客户端发送的头信息可以使用$http_再跟一个头名称，名称要小写，将头名称(header name)中的短横线“-”换为下划线“_”

响应头

以类似的方法，可以访问 HTTP 响应头(response header)，响应头由服务器端发往客户端，这些变量不一定总是一直有效——响应发送后，它们将只带来一个值，例如，在将消息写往日志的时候。

$sent_http_content_type	HTTP 头中 Content-Type 的值，指出被传递资源的 MIME 类型
$sent_http_content_length	HTTP 头中 Content-Length 的值，它会告知响应体(response body)的长度
$sent_http_location	HTTP 头中 Location 的值，它指出想要访问的 location 资源与原始请求(original request)中指定 location 的不同
$sent_http_last_modified	HTTP 头中 Last-Modified 的值，相当于修改请求资源的日期
$sent_http_connection	HTTP 头中 Connection 的值，它定义了连接是否持续有效或者是已关闭
$sent_http_keep_alive	HTTP 头中 Keep-Alive 的值，定义了提供持续连接的时间长度
$sent_http_transfer_encoding	HTTP 头中 Transfer-Encoding 的值，该变量指出有关响应体(response body)所使用编码方法的相关信息(例如压缩，gzip)
$sent_http_cache_control	HTTP 头中 Cache-Control 的值，该变量会告诉客户端浏览器是否对资源进行缓存
$sent_http_...	获取其他发送到客户端的头信息，可以使用$sent_http_再跟一个头名称，名称要小写，将头名称(header name)中的短横线“-”换为下划线“_”

变量	描述

变量名称	描　述

续表

变　　量	描　　述
$http_via	HTTP 头中 Via 的值，该值会告知我们客户端可能使用的代理
$http_x_forwarded_for	HTTP 头中 X-Forwarded-For 的值，如果客户端在代理之后，通过该变量能够取得真实的客户端 IP 地址
$http_cookie	HTTP 头中 cookie 的值，该变量中包含客户端发送的 cookie 数据
$http_...	获取其他通过客户端发送的头信息可以使用$http_再跟一个头名称，名称要小写，将头名称(header name)中的短横线“-”换为下划线“_”

响应头

以类似的方法，可以访问 HTTP 响应头(response header)，响应头由服务器端发往客户端，这些变量不一定总是一直有效——响应发送后，它们将只带来一个值，例如，在将消息写往日志的时候。

变量名称	描　　述
$sent_http_content_type	HTTP 头中 Content-Type 的值，指出被传递资源的 MIME 类型
$sent_http_content_length	HTTP 头中 Content-Length 的值，它会告知响应体(response body)的长度
$sent_http_location	HTTP 头中 Location 的值，它指出想要访问的 location 资源与原始请求(original request)中指定 location 的不同
$sent_http_last_modified	HTTP 头中 Last-Modified 的值，相当于修改请求资源的日期
$sent_http_connection	HTTP 头中 Connection 的值，它定义了连接是否持续有效或者是已关闭
$sent_http_keep_alive	HTTP 头中 Keep-Alive 的值，定义了提供持续连接的时间长度
$sent_http_transfer_encoding	HTTP 头中 Transfer-Encoding 的值，该变量指出有关响应体(response body)所使用编码方法的相关信息(例如压缩，gzip)
$sent_http_cache_control	HTTP 头中 Cache-Control 的值，该变量会告诉客户端浏览器是否对资源进行缓存
$sent_http_...	获取其他发送到客户端的头信息，可以使用$sent_http_再跟一个头名称，名称要小写，将头名称(header name)中的短横线“-”换为下划线“_”

Nginx 产生的变量

除了 HTTP 头变量外，Nginx 提供了大量的变量，涉及请求，像前面讲的一样将会被处理，也可以在当前的配置中设置使用。

$arg_XXX	允许你访问查询字符串(GET 参数)，这里的 XXX 替换为具体的参数
$args	所有结合在一起的字符串查询参数
$binary_remote_addr	作为二进制数据的客户端 IP 地址(4 个字节)
$body_bytes_sent	在响应 body 中发送的字节数
$content_length	相当于 HTTP 头的 Content-Length
$content_type	相当于 HTTP 头的 Content-Type
$cookie_XXX	允许你访问 cookie 数据，这里的 XXX 替换为具体的参数
$document_root	对当前的请求返回 root 指令指定的值
$document_uri	返回当前请求的 URI，如果内部重定向被执行，那么它将不同于原始的 URI 请求，它与变量$uri 相同
$host	该变量相当于 HTTP 请求头中的 Host，Nginx 自身会给这个变量赋予一个值，这种情况用于在原始的请求头中没有提供 Host
$hostname	返回服务器的系统名称
$is_args	如果定义$args 变量，$is_args 相当于？如果$args 为空，那么$is_args 也为空
$limit_rate	返回每一个连接的速率，这个速率是通过 limit_rate 指令来定义的，可以通过设置这个指令来编辑这个变量(设置的指令有 Rewrite 模块提供) `set $limit_rate 128k;`
$nginx_version	返回正在运行的 Nginx 的版本
$pid	返回 Nginx 进程标识符
$query_string	如同$args
$remote_addr	返回客户端的 IP 地址
$remote_port	返回客户端套接字的端口
$remote_user	如果使用认证，则返回客户名称
$realpath_root	在客户端请求中返回文档的根目录，在处理符号链接时，将连接到实际的路径
$request_body	返回客户端请求的 body，或 body 为空

变量名称	描　述

Nginx 产生的变量

除了 HTTP 头变量外，Nginx 提供了大量的变量，涉及请求，像前面讲的一样将会被处理，也可以在当前的配置中设置使用。

变量名称	描　　述
$arg_XXX	允许你访问查询字符串(GET 参数)，这里的 XXX 替换为具体的参数
$args	所有结合在一起的字符串查询参数
$binary_remote_addr	作为二进制数据的客户端 IP 地址(4 个字节)
$body_bytes_sent	在响应 body 中发送的字节数
$content_length	相当于 HTTP 头的 Content-Length
$content_type	相当于 HTTP 头的 Content-Type
$cookie_XXX	允许你访问 cookie 数据，这里的 XXX 替换为具体的参数
$document_root	对当前的请求返回 root 指令指定的值
$document_uri	返回当前请求的 URI，如果内部重定向被执行，那么它将不同于原始的 URI 请求，它与变量$uri 相同
$host	该变量相当于 HTTP 请求头中的 Host，Nginx 自身会给这个变量赋予一个值，这种情况用于在原始的请求头中没有提供 Host
$hostname	返回服务器的系统名称
$is_args	如果定义$args 变量，$is_args 相当于？如果$args 为空，那么$is_args 也为空
$limit_rate	返回每一个连接的速率，这个速率是通过 limit_rate 指令来定义的，可以通过设置这个指令来编辑这个变量(设置的指令有 Rewrite 模块提供) `set $limit_rate 128k;`
$nginx_version	返回正在运行的 Nginx 的版本
$pid	返回 Nginx 进程标识符
$query_string	如同$args
$remote_addr	返回客户端的 IP 地址
$remote_port	返回客户端套接字的端口
$remote_user	如果使用认证，则返回客户名称
$realpath_root	在客户端请求中返回文档的根目录，在处理符号链接时，将连接到实际的路径
$request_body	返回客户端请求的 body，或 body 为空

$request_body_file	如果请求体(request body)被保存(参见client_body_in_file_only)，则该变量会指出临时文件的路径
$request_completion	如果请求完成将返回OK，否则为空字符串
$request_filename	返回在当前请求中提供的全文件名
$request_method	该变量指出在请求中使用的HTTP方法，例如GET或POST
$request_uri	相当于原始的URI请求，在整个处理过程中保持不变(不像$document_uri/$uri)
$scheme	返回http或https，取决于客户端的请求
$server_addr	返回服务器的IP地址。每一个变量都需要系统调用，这一点需要注意，在高流量的设置设置中，可能会影响系统的总体性能
$server_name	表示指令server_name的值，在处理请求时使用
$server_port	表示服务器接收请求数据的套接字端口
$server_protocol	返回协议和版本号，通常为HTTP/1.0或HTTP/1.1
$uri	等同于$document_uri

Location 区段

我们已经确定Nginx提供的服务，可以通过三个级别调整配置文件：在协议级别(http区段)、在server级别(server区段)及在请求的URI级别(location区段)，现在让我们详细讲述后者。

Location 修饰符

Nginx允许你定义location区段，通过指定的模式与客户端请求的URI相匹配：

```
server {
    server_name website.com;
    location /admin/ {
        # The configuration you place here only applies to
        http://website.com/admin/
    }
}
```

续表

变量名称	描述

续表

变量名称	描　述
$request_body_file	如果请求体(request body)被保存(参见client_body_in_file_only)，则该变量会指出临时文件的路径
$request_completion	如果请求完成将返回OK，否则为空字符串
$request_filename	返回在当前请求中提供的全文件名
$request_method	该变量指出在请求中使用的HTTP方法，例如GET或POST
$request_uri	相当于原始的URI请求，在整个处理过程中保持不变(不像$document_uri/$uri)
$scheme	返回http或https，取决于客户端的请求
$server_addr	返回服务器的IP地址。每一个变量都需要系统调用，这一点需要注意，在高流量的设置设置中，可能会影响系统的总体性能
$server_name	表示指令server_name的值，在处理请求时使用
$server_port	表示服务器接收请求数据的套接字端口
$server_protocol	返回协议和版本号，通常为HTTP/1.0或HTTP/1.1
$uri	等同于$document_uri

Location 区段

我们已经确定Nginx提供的服务，可以通过三个级别调整配置文件：在协议级别(http区段)、在server级别(server区段)及在请求的URI级别(location区段)，现在让我们详细讲述后者。

Location 修饰符

Nginx允许你定义location区段，通过指定的模式与客户端请求的URI相匹配：

```
server {
    server_name website.com;
    location /admin/ {
        # The configuration you place here only applies to
        http://website.com/admin/
    }
}
```

前面是一个简单的目录，可以使用一个复杂的模式。下面是 location 区段的语法：

```
location [=|~|~*|^~|@] pattern { ... }
```

第一个可选参数是一个符号，称作 location 修饰符，它用来定义 Nginx 的匹配模式，也定义非常自然的模式(简单字符串或正则表达式)。下表描述的是不同行为。

=	URI 的定位必须与指定的模式精确匹配。该模式在这里限定为一个简单的文本字符串，不能使用正则表达式：
	`server {`
	`  server_name website.com;`
	`  location = /abcd {`
	`  [...]`
	`  }`
	`}`
	对该 location 区段配置的访问：
	● 可用 *http://website.com/abcd* (严格匹配)
	● 可用 *http://website.com/ABCD* (区分大小写，如果操作系统区分大小写)
	● 可用 *http://website.com/abcd?param1&param2* (不管查询字符串参数)
	● 不可用 *http://website.com/abcd/* (结尾斜杠)
	● 不可用 *http://website.com/abcde* (在指定的模式后添加额外的字符)

修饰符	描述

前面是一个简单的目录，可以使用一个复杂的模式。下面是 location 区段的语法：

```
location [=|~|~*|^~|@] pattern { ... }
```

第一个可选参数是一个符号，称作 location 修饰符，它用来定义 Nginx 的匹配模式，也定义非常自然的模式(简单字符串或正则表达式)。下表描述的是不同行为。

修饰符	描述

=	URI 的定位必须与指定的模式精确匹配。该模式在这里限定为一个简单的文本字符串，不能使用正则表达式：
	`server {`
	`  server_name website.com;`
	`  location = /abcd {`
	`  […]`
	`  }`
	`}`
	对该 location 区段配置的访问：
	● 可用 *http://website.com/abcd* (严格匹配)
	● 可用 *http://website.com/ABCD* (区分大小写，如果操作系统区分大小写)
	● 可用 *http://website.com/abcd?param1&param2* (不管查询字符串参数)
	● 不可用 *http://website.com/abcd/* (结尾斜杠)
	● 不可用 *http://website.com/abcde* (在指定的模式后添加额外的字符)

(无)	URI 的定位必须以指定模式开始，不可以使用正则表达式。
	`server {`
	`  server_name website.com;`
	`  location /abcd {`
	`  [...]`
	`  }`
	`}`
	对该 location 区段配置的访问：
	● 可用 *http://website.com/abcd* (严格匹配)
	● 可用 *http://website.com/ABCD*
	(区分大小写，如果你的操作系统使用了区分大小写的文件名)
	● 可用 *http://website.com/abcd?param1&param2*
	(不管查询字符串参数)
	● 可用 *http://website.com/abcd/* (结尾斜杠)
	● 可用 *http://website.com/abcde* (在指定的模式后添加额外的字符)

~	客户端请求的 URI 与指定的正则表达式匹配必须区分大小写。
	`server {`
	`  server_name website.com;`
	`  location ~ ^/abcd$ {`
	`  [...]`
	`  }`
	`}`
	在本例中使用正则表达式 ^/abcd$，指定匹配模式必须以/开始(^),然后再跟 abc 字符，结尾($)为 d 字符。因此，对该 location 区段配置的访问：
	● 可用 *http://website.com/abcd* (严格匹配)
	● 不可用 *http://website.com/ABCD* (区分大小写)
	● 可用 *http://website.com/abcd?param1&param2*
	(不管查询字符串参数)
	● 不可用 *http://website.com/abcd/*
	(结尾斜杠) 因为指定了正则表达式
	● 不可用 *http://website.com/abcde*
	(额外字符) 因为指定了正则表达式
	注意：有的操作系统(例如 Windows)，~ 和 ~*大小写不敏感，是因为操作系统本身对字符大小写不敏感

修饰符	描述

续表

修饰符	描述

(无)	URI 的定位必须以指定模式开始，不可以使用正则表达式。
	`server {`
	`  server_name website.com;`
	`  location /abcd {`
	`  [...]`
	`  }`
	`}`
	对该 location 区段配置的访问：
	● 可用 *http://website.com/abcd* (严格匹配)
	● 可用 *http://website.com/ABCD*
	(区分大小写，如果你的操作系统使用了区分大小写的文件名)
	● 可用 *http://website.com/abcd?param1&param2*
	(不管查询字符串参数)
	● 可用 *http://website.com/abcd/* (结尾斜杠)
	● 可用 *http://website.com/abcde* (在指定的模式后添加额外的字符)

~	客户端请求的 URI 与指定的正则表达式匹配必须区分大小写。
	`server {`
	`  server_name website.com;`
	`  location ~ ^/abcd$ {`
	`  [...]`
	`  }`
	`}`
	在本例中使用正则表达式 ^/abcd$，指定匹配模式必须以/开始(^),然后再跟 abc 字符，结尾($)为 d 字符。因此，对该 location 区段配置的访问：
	● 可用 *http://website.com/abcd* (严格匹配)
	● 不可用 *http://website.com/ABCD* (区分大小写)
	● 可用 *http://website.com/abcd?param1&param2*
	(不管查询字符串参数)
	● 不可用 *http://website.com/abcd/*
	(结尾斜杠) 因为指定了正则表达式
	● 不可用 *http://website.com/abcde*
	(额外字符) 因为指定了正则表达式
	注意：有的操作系统(例如 Windows)，~ 和 ~*大小写不敏感，是因为操作系统本身对字符大小写不敏感

~*	对客户端请求的 URI 与指定的正则表达式匹配，必须不区分大小写。
	`server {`
	`  server_name website.com;`
	`  location ~* ^/abcd$ {`
	`  [...]`
	`  }`
	`}`
	在本例中使用的正则表达式与前面的类似。因此，对该 location 区段配置的访问：
	● 可用 *http://website.com/abcd* (严格匹配)
	● 可用 *http://website.com/ABCD* (大小写不敏感)
	● 可用 *http://website.com/abcd?param1&param2* (不管查询字符串参数)
	● 不可用 *http://website.com/abcd/* (结尾斜杠) 因为指定了正则表达式
	● 不可用 *http://website.com/abcde* (额外字符) 因为指定了正则表达式

^~	类似于无标志(no symbol)行为，URI 的定位必须以指定模式开始。不同的是，如果模式匹配，那么 Nginx 就停止搜索其他模式(参考下文)
@	定义命名 location 区段，这些区段客户端不能够访问，只可以由内部产生的请求来访问，例如 try_files 或 error_page

查找顺序和优先级

既然通过不同的模式可以定义多个 location 区段，则需要认识到一点，当 Nginx 收到一个请求的时候，它会搜索 location 区段，以找出最佳的请求 URI 匹配：

```
server {
   server_name website.com;
      location /files/ {"
         # 可用于任何以"/files/"开始的请求
         # 例如/files/doc.txt, /files/, /files/temp/
      }
      location = /files/ {
         # 可用于与"/files/"精确匹配的请求
```

续表

修饰符	描述

续表

修饰符	描述

~*	对客户端请求的 URI 与指定的正则表达式匹配，必须不区分大小写。
	`server {`
	`  server_name website.com;`
	`  location ~* ^/abcd$ {`
	`  [...]`
	`  }`
	`}`
	在本例中使用的正则表达式与前面的类似。因此，对该 location 区段配置的访问：
	● 可用 *http://website.com/abcd* (严格匹配)
	● 可用 *http://website.com/ABCD* (大小写不敏感)
	● 可用 *http://website.com/abcd?param1&param2* (不管查询字符串参数)
	● 不可用 *http://website.com/abcd/* (结尾斜杠) 因为指定了正则表达式
	● 不可用 *http://website.com/abcde* (额外字符) 因为指定了正则表达式

^~	类似于无标志(no symbol)行为，URI 的定位必须以指定模式开始。不同的是，如果模式匹配，那么 Nginx 就停止搜索其他模式(参考下文)
@	定义命名 location 区段，这些区段客户端不能够访问，只可以由内部产生的请求来访问，例如 try_files 或 error_page

查找顺序和优先级

既然通过不同的模式可以定义多个 location 区段，则需要认识到一点，当 Nginx 收到一个请求的时候，它会搜索 location 区段，以找出最佳的请求 URI 匹配：

```
server {
    server_name website.com;
        location /files/ {"
            # 可用于任何以"/files/"开始的请求
            # 例如/files/doc.txt, /files/, /files/temp/
        }
        location = /files/ {
            # 可用于与"/files/"精确匹配的请求
```

```
            # 不能够用于到对/files/doc.txt 的访问请求
            # 但是仅可以 /files/
        }
}
```

当客户端访问 http://website.com/files/doc.txt 的时候，第一个 location 区段被使用，然而当客户端访问 http://website.com/files/的时候，第二个 location 区段将被使用(即便是第一个 location 匹配)，因为它的优先权高于第一个 location(它是精确匹配)。由此可见，匹配的顺序与在配置文件中的建立顺序(将“/files/”块放在“=/files/”前)是不相关的，Nginx 将在特定的顺序中搜索模式匹配。

1. 带有=修饰符的 location 区段　如果指定字符串与请求的 URI 精确匹配，则 Nginx 使用该 location 的设置。
2. 没有修饰符的 location 区段　如果指定字符串与请求的 URI 精确匹配，则 Nginx 使用该 location 的设置。
3. 带有^~修饰符的 location 区段　如果指定字符串与请求的 URI 开始匹配，则 Nginx 使用该 location 的设置。
4. 带有~ 或 ~*修饰符的 location 区段　如果正则表达式与请求的 URI 匹配，则 Nginx 使用该 location 的设置。
5. 没有修饰符的 location 区段　如果指定字符串与 URI 请求开始(beginning of the requested URI)匹配，则 Nginx 使用该 location 的设置。

在这种情况下，修饰符“^~”开始变得有意义，我们可以设想一下，下列情况会让它变得有用。

情况 1

```
server {
    server_name website.com;
    location /doc {
        […] # 由"/doc"开始的请求
    }
    location ~* ^/document$ {
        […] # 请求精确匹配"/document"
    }
}
```

客户端请求 http://website.com/document 的时候，可能会感到惊奇，哪一个 location 区段会被使用？确实如此，这个请求将有两个区段匹配，同样，答案并不在于块在配置文件中出现的顺序。

情况 2

```
server {
    server_name website.com;
    location /document {
        […] #由"/document"开始的请求
    }
    location ~* ^/document$ {
        […] # 精确匹配"/document"请求
    }
}
```

遗留下来的是同样的问题：当客户端发送一个下载 *http://website.com/document* 的请求时会发生什么？这里有一个技巧。在第一个块中指定的字符串现在精确匹配客户端的 URI 请求，因此，相比而言，Nginx 比正则表达式更优先使用第一个区段。

情况 3

```
server {
    server_name website.com;
    location ^~ /doc {
        […] # 由"/doc"开始的请求
    }
    location ~* ^/document$ {
        […] #请求精确匹配"/document"
    }
}
```

最后一种情况是使用“^~”修饰符。客户端访问 *http://website.com/document* 时，哪一个区段将被使用？回答是：第一个区段。因为“^~”的优先级高于“~*”，因此，任何以/doc 开头的 URI 都将受到第一个区段的影响，即使是第二个区段中的正则表达式与请求的 URI 匹配。

小结

在本章中，我们自始至终都在学习 Nginx HTTP 配置。首先，我们学会了通过声明 server 区段来建立虚拟主机，然后，我们认识了 HTTP Core 模块的指令和变量，它们可以插入哪些区段。最后，理解了 location 区段的调整机制。

这些工作已完成。现在，你的服务器实际上已经提供了网站。我们打算进行下一步，更深入地认识其他模块，它们能够使 Nginx 提供更强大的功能。第 5 章将进入高级话题，例如 Rewrite 和 SSI 模块，以及 HTTP 服务的其他部分。

第 5 章
模块配置

Nginx 真正的魅力在于它的模块，整个应用程序建立在一个模块化系统之上，在编译时可以对每一个模块进行启用或禁用。在这些模块中，有的功能简单(例如 *Autoindex* 模块，它会对目录中的文件产生一个列表)；有的模块则改变了你对 web 服务器的认识(例如，Rewrite 模块)；开发者还邀请你自己建立自己的模块；在本章结尾会快速概述一下第三方模块。

本章主题

- Rewrite 模块，使用它你可以更多地改写 URI
- SSI 模块，服务器端脚本语言
- 在 Nginx 建立时默认启用的模块
- 在编译时必须启用的可选模块
- 快速了解第三方模块

Rewrite 模块

特别是这个模块，对于 Nginx 来说，相比于其他模块，对它进行一个简单的指令设置，会带来了更多功能，它定义了一个新的处理请求级别，本章将锁定这个功能进行讲解。

最初，该模块的目的(顾名思义)是执行 URL 重定向，这个机制允许你去掉带有恶意的 URL，包含多个参数，例如 *http://example.com/article.php?id=1234&comment=32*——这样的网址不会提供任何信息并且对访问者毫无意义。链接到 Web 站点时会包含有用的信息，它会指明你正在访问的网页。即尽可能告诉用户该 URL 中大致的内容，一个实例性的 URL 会是这样：*http://website.com/article-1234-32-US-economy-strengthens.html*。如此一来，不但会使访问者更有兴趣，还有利于搜索引擎——对于搜索引擎优化(SEO)，这也是一个关键的因素。

这种机制的原理其实很简单——在请求到达之后、在提供文件之前，客户端的请求包含重定向的 URI。一旦被重写，为了找到适用于本请求的配置，URI 会与相应的 location 区段相匹配，有关详情参见后文。

正则表达式

首先且也是最重要的是，需要理解正则表达式，事实上，URL 重写由 rewrite 指令来执行，该指令接受一个模式(正则表达式)，后跟一个替换 URI。

这是一个庞大的话题——有很多书专门解释它的来龙去脉。然而，我们将要通过简化的方法来证实，但又将充分利用该机制。

目的

首先要回答的一个问题是：正则表达式有何用途？简而言之——其主要目的是验证(verify)一个字符串匹配模式。此模式是用一种特殊的语言编写的，它允许你定义极其复杂而准确的规则。

hello	^hello$	是	字符串由 h (^h)字符开始，后跟 e，l，l，字符，并以 o (o$)结尾
hell	^hello$	不	字符串由 h (^h)字符开始，后跟 e，l，l，字符，但不以 o 结尾
Hello	^hello$	视情况而定	如果引擎执行区分大小写，则字符串不会匹配该模式

如果使用的模式很复杂，例如，有一个邮件地址验证模式：^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$，会使你更有兴趣。从程序角度验证电子邮件的格式需要大量的代码。然而在这里，所有的工作可以用一个正则表达式模式匹配来解决。

PCRE 语法

Nginx 所用的语法源于 Perl 兼容正则表达式(PCRE)库，这是在编译前首先要安装的(如第 2 章所述)，除非禁用 PCRE 库的模块。这是正则表达式最常用的形式，你在这里所学的所有东西适用于其他语言。

字符串	模式	是否匹配	解　　释

这种机制的原理其实很简单——在请求到达之后、在提供文件之前，客户端的请求包含重定向的 URI。一旦被重写，为了找到适用于本请求的配置，URI 会与相应的 location 区段相匹配，有关详情参见后文。

正则表达式

首先且也是最重要的是，需要理解正则表达式，事实上，URL 重写由 rewrite 指令来执行，该指令接受一个模式(正则表达式)，后跟一个替换 URI。

这是一个庞大的话题——有很多书专门解释它的来龙去脉。然而，我们将要通过简化的方法来证实，但又将充分利用该机制。

目的

首先要回答的一个问题是：正则表达式有何用途？简而言之——其主要目的是验证(verify)一个字符串匹配模式。此模式是用一种特殊的语言编写的，它允许你定义极其复杂而准确的规则。

字符串	模式	是否匹配	解　　释
hello	^hello$	是	字符串由 h (^h)字符开始，后跟 e，l，l，字符，并以 o (o$)结尾
hell	^hello$	不	字符串由 h (^h)字符开始，后跟 e，l，l，字符，但不以 o 结尾
Hello	^hello$	视情况而定	如果引擎执行区分大小写，则字符串不会匹配该模式

如果使用的模式很复杂，例如，有一个邮件地址验证模式：^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$，会使你更有兴趣。从程序角度验证电子邮件的格式需要大量的代码。然而在这里，所有的工作可以用一个正则表达式模式匹配来解决。

PCRE 语法

Nginx 所用的语法源于 Perl 兼容正则表达式(PCRE)库，这是在编译前首先要安装的(如第 2 章所述)，除非禁用 PCRE 库的模块。这是正则表达式最常用的形式，你在这里所学的所有东西适用于其他语言。

在其最简单的形式中，一个模式由一个字符组成(例如 x)，我们能够对这个模式进行字符串匹配。有匹配模式 x 的例子吗？当然，例如包含字符 x 的例子。可以是前面指定的字符——(模式[a-z]匹配从 a 到 z 的任何字符)，也可以是字母和数字的组合([a-z0-9])。表达式 hell[a-z0-9]匹配的结果可以是：hello 和 hell4，但不可以是 hell 和 hell!。

我们使用的字符 [和]叫元字符，对模式有特殊的效果。这种元字符共有 11 个字符，它们扮演着不同的角色。如果想在建立的模式中包含这些元字符中的某一个字符，需要在该字符前使用转义符“\”。

^ 开始 (beginning)	字符“^”之后的实体(entity)，必须在被匹配的字符串开始部分找到 例如： ^h 匹配的字符串：hello，h，hh 不能匹配的字符串：character，ssh
$ 结束(end)	在字符“$”前的实体(entity)，必须在匹配的字符串尾找到。 例如：e$ 匹配的字符串：sample, e, file 不能匹配的字符串：extra, shell
. 任何字符 (any)	匹配任何字符： 例如：hell 匹配的字符串：hello，hellx，hell5，hell! 不能匹配的字符串：hell，helo
[ ] 组(set)	匹配指定字符集内的任何字符。 语法：[a-z] 表示一个区间,[abcd] 表示一组,[a-z0-9] 两个区间 例如：hell[a-y123] 匹配的字符串：hello，hell1，hell2，hell3 不能匹配的字符串：hellz，hell4，heloo
[^] 否定组 (negate set)	匹配任何不包括在指定字符集内的字符串： 例如：hell[^a-np-z0-9] 匹配的字符串：hello，hell; 不能匹配的字符串：hella，hell5

元字符	描述

在其最简单的形式中，一个模式由一个字符组成(例如 x)，我们能够对这个模式进行字符串匹配。有匹配模式 x 的例子吗？当然，例如包含字符 x 的例子。可以是前面指定的字符——(模式[a-z]匹配从 a 到 z 的任何字符)，也可以是字母和数字的组合([a-z0-9])。表达式 hell[a-z0-9]匹配的结果可以是：hello 和 hell4，但不可以是 hell 和 hell!。

我们使用的字符 [和]叫元字符，对模式有特殊的效果。这种元字符共有 11 个字符，它们扮演着不同的角色。如果想在建立的模式中包含这些元字符中的某一个字符，需要在该字符前使用转义符“\”。

元 字 符	描　　述
^ 开始 (beginning)	字符“^”之后的实体(entity)，必须在被匹配的字符串开始部分找到 例如： ^h 匹配的字符串：hello，h，hh 不能匹配的字符串：character，ssh
$ 结束(end)	在字符“$”前的实体(entity)，必须在匹配的字符串尾找到。 例如：e$ 匹配的字符串：sample, e, file 不能匹配的字符串：extra, shell
. 任何字符 (any)	匹配任何字符： 例如：hell 匹配的字符串：hello，hellx，hell5，hell! 不能匹配的字符串：hell，helo
[ ] 组(set)	匹配指定字符集内的任何字符。 语法：[a-z] 表示一个区间,[abcd] 表示一组,[a-z0-9] 两个区间 例如：hell[a-y123] 匹配的字符串：hello，hell1，hell2，hell3 不能匹配的字符串：hellz，hell4，heloo
[^] 否定组 (negate set)	匹配任何不包括在指定字符集内的字符串： 例如：hell[^a-np-z0-9] 匹配的字符串：hello，hell; 不能匹配的字符串：hella，hell5

| 或(alternation)	匹配符号“|”之前或之后的实体。 例如：hello|welcome 匹配的字符串：hello，welcome，helloes，awelcome 不能匹配的字符串：hell，ellow，owelcom
( ) 分组(grouping)	组成一组用于匹配的实体(entity)，常用符号“|”协助完成。 例如：^(hello|hi) there$ 匹配的字符串：hello there，hi there 不能匹配的字符串：hey there，ahoy there
\ 转义(escape)	允许你对一个特殊字符转义(即将具有特殊意义的字符变为普通的文本字符)。 例如：Hello 匹配的字符串：Hello.，Hello. How are you?，Hi! Hello... 不能匹配的字符串：Hello，Hello，how are you?

量词

可以通过使用有限的字符来表达简单的匹配。下表所示的量词允许你扩展实体匹配使用量词可以定制匹配的次数。

* 0 或多次	字符“*”之前的实体被发现 0 次或多次 例如：he*llo 匹配的字符串：hllo，hello，heeeello 不能匹配的字符串：hallo，ello
+ 1 或多次	字符 “+”之前的实体被发现 1 次或多次。 例如：he+llo 匹配的字符串：hello，heeeello 不能匹配的字符串：hllo，helo
? 0 或 1 次	字符“？”之前的字符必须被发现 0 或 1 次。 例如：he?llo 匹配的字符串：hello，hllo 不能匹配的字符串: heello，heeeello
{x} x 次	字符“{}”之前的字符必须匹配指定的 x 次数。 例如：he{3}llo 匹配的字符串：heeello，oh heeello there! 不能匹配的字符串：hello，heello，heeeello

元字符	描述

量词	描述

续表

元 字 符	描　　述
| 或(alternation)	匹配符号“|”之前或之后的实体。 例如：hello|welcome 匹配的字符串：hello，welcome，helloes，awelcome 不能匹配的字符串：hell，ellow，owelcom
( ) 分组(grouping)	组成一组用于匹配的实体(entity)，常用符号“|”协助完成。 例如：^(hello|hi) there$ 匹配的字符串：hello there，hi there 不能匹配的字符串：hey there，ahoy there
\ 转义(escape)	允许你对一个特殊字符转义(即将具有特殊意义的字符变为普通的文本字符)。 例如：Hello 匹配的字符串：Hello.，Hello. How are you?，Hi! Hello... 不能匹配的字符串：Hello，Hello，how are you?

量词

可以通过使用有限的字符来表达简单的匹配。下表所示的量词允许你扩展实体匹配使用量词可以定制匹配的次数。

量　　词	描　　述
* 0 或多次	字符“*”之前的实体被发现 0 次或多次 例如：he*llo 匹配的字符串：hllo，hello，heeeello 不能匹配的字符串：hallo，ello
+ 1 或多次	字符 “+”之前的实体被发现 1 次或多次。 例如：he+llo 匹配的字符串：hello，heeeello 不能匹配的字符串：hllo，helo
? 0 或 1 次	字符“？”之前的字符必须被发现 0 或 1 次。 例如：he?llo 匹配的字符串：hello，hllo 不能匹配的字符串: heello，heeeello
{x} x 次	字符“{}”之前的字符必须匹配指定的 x 次数。 例如：he{3}llo 匹配的字符串：heeello，oh heeello there! 不能匹配的字符串：hello，heello，heeeello

{x,} 至少 x 次	字符“{}”的字符必须至少找到指定的 x 次数(即可以多于 x 次)。 例如：he{3}llo 匹配的字符串：heeello，heeeeeeello 不能匹配的字符串：hllo，hello,heello
{x,y} x 到 y 次	字符“{}”的字符找到的次数必须介于 x 和 y 次之间。 例如：he{2，4}llo 匹配的字符串：heello，heeello，heeeello 不能匹配的字符串：hello，heeeeello

正则表达式的元字符“{”和“}”可能会和 Nginx 配置文件中区段(block)定界符“{ }”有冲突。因此，如果想写一个包括花括号的正则表达式，需要将模式放在引号之间(单引号或双引号)：

```
rewrite  hel{2,}o   /hello.php;    # invalid
rewrite "hel{2,}o"  /hello.php;    # vaild
rewrite 'hel{2,}o'  /hello.php;    # vaild
```

捕获

正则表达式机制的最后一个功能是能够捕获子表达式，放在圆括号“( )”之间的任何文本，在被捕获后都能用于后面的匹配处理。下表列出一些例子以便说明其原理。

^(hello|hi) (sir|mister)$	hello sir	$1 = hello $2 = sir
^(.*)$	nginx rocks	$1 = nginx rocks
^(.{1,3})([0-9]{1,4})([?!]{1,2})$	abc1234!?	$1 = abc $2 = 1234 $3 = !?

量词	描述

模式	字符串	捕获

续表

量　　词	描　　述
{x,} 至少 x 次	字符“{}”的字符必须至少找到指定的 x 次数(即可以多于 x 次)。 例如：he{3}llo 匹配的字符串：heeello，heeeeeeello 不能匹配的字符串：hllo，hello,heello
{x,y} x 到 y 次	字符“{}”的字符找到的次数必须介于 x 和 y 次之间。 例如：he{2，4}llo 匹配的字符串：heello，heeello，heeeello 不能匹配的字符串：hello，heeeeello

正则表达式的元字符“{”和“}”可能会和 Nginx 配置文件中区段(block)定界符“{ }”有冲突。因此，如果想写一个包括花括号的正则表达式，需要将模式放在引号之间(单引号或双引号)：

```
rewrite  hel{2,}o   /hello.php;    # invalid
rewrite "hel{2,}o"  /hello.php;    # vaild
rewrite 'hel{2,}o'  /hello.php;    # vaild
```

捕获

正则表达式机制的最后一个功能是能够捕获子表达式，放在圆括号“( )”之间的任何文本，在被捕获后都能用于后面的匹配处理。下表列出一些例子以便说明其原理。

模　　式	字 符 串	捕　　获
^(hello|hi) (sir|mister)$	hello sir	$1 = hello $2 = sir
^(.*)$	nginx rocks	$1 = nginx rocks
^(.{1,3})([0-9]{1,4})([?!]{1,2})$	abc1234!?	$1 = abc $2 = 1234 $3 = !?

在 Nginx 表达式中使用正则表达式的时候，例如，在一个 location 区段的环境中，这些被捕获的缓冲数据(这些数据均放在缓冲区中)可以用于后面的指令：

```
server {
    server_name website.com;
    location ~* ^/(downloads|files)/(.*)$ {
    add_header Capture1 $1;
    add_header Capture2 $2;
    }
}
```

在前面的例子中，location 区段的正则表达式将匹配请求的 URI，这里将用到一些 URL：/downloads/file.txt, /files/archive.zip 或者甚至是/files/docs/report.doc。有两部分被抓获：$1 将包含 downloads 或 files，$2 将包含/downloads/ 或 /files/之后的任何东西。注意这里使用的是指令 add_header(语法：add_header header_name header_value，参考 HTTP headers 模块部分)，它在客户端响应 header 中插入自定义字段，这里是为了演示。

内部请求

Nginx 区分内部请求和外部请求。外部请求直接源于客户端，URI 尽量与 location 区段匹配。

```
server {
    server_name website.com;
    location = /document.html {
    deny all; # example directive
    }
}
```

客户端请求 *http://website.com/document.html* 时，将直接进入前面的 location 区段。与此相对，内部请求会被 Nginx 通过特定的指令触发。

在默认的 Nginx 模块中，有些内部指令有能力处理内部请求：error_page，index，rewrite，try_files，add_before_body，add_after_body(来自 Addition 模块)，include SSI 命令，等等。

有两种不同类型的内部请求。

- **内部重定向**(Internal redirect) Nginx 在内部重定向客户端的请求。URI 被改变，请求可能匹配到其他的 location 区段，并且根据不同的设置做出相应的选择。最常见的内部重定向是使用一个改写指令，对请求的 URI 进行重写。
- **子请求**(Sub-request) 另外一种触发内部请求而产生内容的是子请求(请求与被请求产生的内容)。一个简单的例子是使用 Addition 模块，指令 add_after_body 允许你在原始 URI 之后指定一个 URI，该 URI 将被处理，处理的结果将插入到主体(body)中。SSI 模块也可以通过 include 命令来利用子请求插入内容。

error_page

该指令的详细介绍可参见 HTTP 核心模块，它用于定义发生指定的错误代码时使用 error_page 来定义服务器的行为。最简单的形式是使用一个错误代码改变 URI：

```
server {
    server_name website.com;
    error_page 403 /errors/forbidden.html;
    error_page 404 /errors/not_found.html;
}
```

客户端在尝试访问一个 URI 而触发以上任何一个错误时，Nginx 会提供一个相当于错误代码的网页。事实上，它不是向客户端发送错误网页——它实际上要完成一个基于新 URI 的请求。因此，转到一个不同位置，如下面的例子所示：

```
server {
    server_name website.com;
    root /var/www/vhosts/website.com/httpdocs/;
    error_page 404 /errors/404.html;
    location /errors/ {
        alias /var/www/common/errors/;
        internal;
    }
}
```

当一个客户端试图载入一个不存在的文档时，最初会收到 404 错误。我们使用 error_page 指令来指定 404 错误建立一个内部重定向到/errors/404.html，因此，Nginx 会产生一个新的请求——提供 URI /errors/404.html 网页。请求的这个 URI 将进入“location /errors/”区段，从而导致这里的配置被应用。

下面是原始的但经过裁剪的请求信息，节选自这种机制下的调试信息(注意，记录日志必须设置为 debug 才能够捕获这些信息，详情可参阅 error_log 指令)。

```
->http request line: "GET /page.html HTTP/1.1"
->http uri: "/page.html"
->test location: "/errors/"
->using configuration ""
->http filename:
"/var/www/vhosts/website.com/httpdocs/page.html"
-> open() "/var/www/vhosts/website.com/httpdocs/page.html"
failed (2: No such file or directory), client: 127.0.0.1,
server: website.com, request: "GET /page.html HTTP/1.1", host:
"website.com"
->http finalize request: 404, "/page.html?" 1
->http special response: 404, "/page.html?"
->internal redirect: "/errors/404.html?"
->test location: "/errors/"
->using configuration "/errors/"
->http filename: "/var/www/common/errors/404.html"
->http finalize request: 0, "/errors/404.html?" 1
```

注意，在 location 区段中使用 internal 指令来禁止客户端访问/errors/目录，因此该 location 只能够被内部重定向访问。

该机制同 index 指令一样(详情参考后面的 index 模块)——如果客户端没有提供请求文件的路径，Nginx 将尝试提供有内部重定向触发指定的 index 页。

rewrite

前面所述的指令 error_page，不是 rewrite 的一部分，但它详细描述了 Nginx 处理请求的方法，在这个功能中提供了一个可靠的应用。类似于 error_page 如何处理重定向到其他的 location，内部重定向也是通过 rewrite 指令重写 URI 来产生的。

```
server {
    server_name website.com;
    root /var/www/vhosts/website.com/httpdocs/;
    location /storage/ {
        internal;
        alias /var/www/storage/;
    }
    location /documents/ {
```

```
        rewrite ^/documents/(.*)$ /storage/$1;
    }
}
```

客户端查询 *http://website.com/documents/file.txt*，开始匹配第二个 location 区段(location /documents/)，然而，该区段包含一个 rewrite 指令，它会将请求的 URI 由/documents/file.txt 转换为/storage/file.txt。这个 URI 转换会重新初始化进程——新的 URI 将匹配另一个 location 区段，这时，第一个 location 区段(location /storage/)将配置这个 URI(/storage/file.txt)。

再次快速看一下这种机制下的调试信息：

```
->http request line: "GET /documents/file.txt HTTP/1.1"
->http uri: "/documents/file.txt"
->test location: "/storage/"
->test location: "/documents/"
->using configuration "/documents/"
->http script regex: "^/documents/(.*)$"
->"^/documents/(.*)$" matches "/documents/file.txt", client:
127.0.0.1, server: website.com, request: "GET
/documents/file.txt HTTP/1.1", host: "website.com"
->rewritten data: "/storage/file.txt", args: "", client:
127.0.0.1, server: website.com, request: "GET
/documents/file.txt HTTP/1.1", host: "website.com"
->test location: "/storage/"
->using configuration "/storage/"
->http filename: "/var/www/storage/file.txt"
->HTTP/1.1 200 OK
->http output filter "/storage/test.txt?"
```

无限循环

各种不同的语法和指令很容易让人迷惑。更糟糕的是，如果发生这种情况(例如，如果你的重写规则是多余的)，很可能会引起内部指令重定向从而导致无限循环：

```
server {
    server_name website.com;
    location /documents/ {
        rewrite ^(.*)$ /documents/$1;
    }
}
```

你认为这个例子会很好地运行，但这个配置实际上会触发一个内部重定向(从/documents/anything 到/documents//documents/anything)，然而，由内部重定向而来的请求将再次被内部重定向，由/documents//documents/anything 变为/documents//documents//documents/anything，如此反复。

下面是这个例子的调试日志(部分摘要)：

```
->test location: "/documents/"
->using configuration "/documents/"
->rewritten data: "/documents//documents/file.txt", [...]
->test location: "/documents/"
->using configuration "/documents/"
->rewritten data: "/documents//documents//documents/file.txt"
[...]
->test location: "/documents/"
->using configuration "/documents/"
->rewritten data: -
>"/documents//documents//documents//documents/file.txt" [...]
->[...]
```

你可能惊奇这种循环是否会无限进行下去——回答是否定的。这种循环的次数仅限于 10。只允许 10 次内部循环，一旦超过这个限制， Nginx 会产生一个 500 内部错误(500 Internal Server Error)。

服务器端包含(SSI)

可能的子请求来源是服务器端包含(SSI)模块，即在将响应数据包发送至客户端之前，服务器端解析一些文档，有点类似于 PHP 或其他预处理器。

例如在一个普通的 HTML 文件中，你有可能插入标签，由 Nginx 解释为相应的命令：

```
<html>
<head>
      <!--# include file="header.html" -->
</head>
<body>
      <!--# include file="body.html" -->
</body>
</html>
```

Nginx 将处理这两个命令，在这种情况下，它会读取 head.html 和 body.html 这两个文件的内容，并将它们插入源代码文档，然后发送至客户端。

有些命令可供你支配，它们在本章的 SSI 模块这一小节中有详细介绍。现在我们感兴趣的一个命令是 include——它把一个文件插入到另一个文件。

```
<!--# include virtual="/footer.php?id=123" -->
```

指定的文件不只是从静态的位置打开并读取，事实上，Nginx 会处理整个子请求，响应体被插入源代码中，取代原有的 **include** 标签。

条件结构

Rewrite 模块引进了一组新的指令和区段，其中之一就是 if 条件结构：

```
server {
    if ($request_method = POST) {
    [···]
    }
}
```

这种结构使我们能够根据指定条件添加配置。如果条件为真，那么配置将被应用，否则将不会被应用。

在形成一个条件时，会使用下表描述的不同语法。

无	如果指定的变量或者数据不等于空字符串或由 0 开始的字符串，则条件为真。
	`if ($string) {`
	`  [...]`
	`}`

=, !=	如果“=”前的变量与“=”后的值相等，那么条件为真。下面的例子被读作“如果 request_method 等于 POST”，就应用“[···]”中的配置。
	
	`if ($request_method = POST) {`
	`  [...]`
	`}`
	“!=”正好与“=”相反。下面的配置中：“如果请求的方法不同于 GET，那么就会应用[···]中的配置。”
	
	`if ($request_method != GET) {`
	`  [...]`
	`}`

操作符	描述

有些命令可供你支配，它们在本章的 SSI 模块这一小节中有详细介绍。现在我们感兴趣的一个命令是 include——它把一个文件插入到另一个文件。

```
<!--# include virtual="/footer.php?id=123" -->
```

指定的文件不只是从静态的位置打开并读取，事实上，Nginx 会处理整个子请求，响应体被插入源代码中，取代原有的 **include** 标签。

条件结构

Rewrite 模块引进了一组新的指令和区段，其中之一就是 if 条件结构：

```
server {
    if ($request_method = POST) {
    […]
    }
}
```

这种结构使我们能够根据指定条件添加配置。如果条件为真，那么配置将被应用，否则将不会被应用。

在形成一个条件时，会使用下表描述的不同语法。

操作符	描述

无	如果指定的变量或者数据不等于空字符串或由 0 开始的字符串，则条件为真。
	`if ($string) {`
	`  [...]`
	`}`

=, !=	如果“=”前的变量与“=”后的值相等，那么条件为真。下面的例子被读作“如果 request_method 等于 POST”，就应用“[…]”中的配置。
	`if ($request_method = POST) {`
	`  [...]`
	`}`
	“!=”正好与“=”相反。下面的配置中：“如果请求的方法不同于 GET，那么就会应用[…]中的配置。”
	`if ($request_method != GET) {`
	`  [...]`
	`}`

~,
~*,
!~,
!~*
	如果符号“~”之前的参数与其后面的指定模式相匹配，那么条件为真。
	`if ($request_filename ~ "\.txt$") {`
	`  [...]`
	`}`
	“~”大小写敏感(区分大小写)；“~*”大小写不敏感；而符号“！”的功能在于否定匹配：
	`if ($request_filename !~* "\.php$") {`
	`  [...]`
	`}`
	注意，可以将缓存中捕获的正则表达式插入到这里。
	`if ($uri ~ "^/search/(.*)$") {`
	`  set $query $1;`
	`  rewrite ^ http://google.com/search?q=$query;`
	`}`

-f, !-f	“-f”测试指定文件是否存在：
	`if (-f $request_filename) {`
	`  [...]  # if the file exists`
	`}`
	使用“!-f”测试文件的不存在性：
	`if (!-f $request_filename) {`
	`  [...]  #if the file does not exist`
	`}`

-d, !-d	类似于“-f ”的操作，用于测试一个目录是否存在
-e, !-e	类似于“-f ”的操作，用于测试一个文件、目录或符号链接是否存在
-x, !-x	类似于“-f ”的操作，用于测试文件是否存在和是否可以执行

到 0.7.66 版本为止，还没有 **else-**或 **else if-**等类似的指令。然而使用其他指令一样能够允许你有效的控制顺序流。

你可能想知道：使用 if 区段与使用 location 区段有什么优势呢？其实，在下面的例子中，这两个区段具有同样的结果：

```
if ($uri ~ /search/) {
    [...]
}
    location ~ /search/ {
    [...]
}
```

操作符	描述

续表

操作符	描　述

~,
~*,
!~,
!~*
	如果符号“~”之前的参数与其后面的指定模式相匹配，那么条件为真。
	`if ($request_filename ~ "\.txt$") {`
	`  [...]`
	`}`
	“~”大小写敏感(区分大小写)；“~*”大小写不敏感；而符号“！”的功能在于否定匹配：
	`if ($request_filename !~* "\.php$") {`
	`  [...]`
	`}`
	注意，可以将缓存中捕获的正则表达式插入到这里。
	`if ($uri ~ "^/search/(.*)$") {`
	`  set $query $1;`
	`  rewrite ^ http://google.com/search?q=$query;`
	`}`

-f, !-f	“-f”测试指定文件是否存在：
	`if (-f $request_filename) {`
	`  [...]  # if the file exists`
	`}`
	使用“!-f”测试文件的不存在性：
	`if (!-f $request_filename) {`
	`  [...]  #if the file does not exist`
	`}`

-d, !-d	类似于“-f ”的操作，用于测试一个目录是否存在
-e, !-e	类似于“-f ”的操作，用于测试一个文件、目录或符号链接是否存在
-x, !-x	类似于“-f ”的操作，用于测试文件是否存在和是否可以执行

到 0.7.66 版本为止，还没有 **else-**或 **else if-**等类似的指令。然而使用其他指令一样能够允许你有效的控制顺序流。

你可能想知道：使用 if 区段与使用 location 区段有什么优势呢？其实，在下面的例子中，这两个区段具有同样的结果：

```
if ($uri ~ /search/) {
    [...]
}
    location ~ /search/ {
    [...]
}
```

事实上，主要的不同在于能够使用在这两个区段的指令不同——有些指令可以用在 if 区段，而有的则不可以，正如你在指令列表中注意到的，在 location 区段中几乎所有的指令都可以使用。总的来说，最好的方法就是在 if 区段中插入来源于 Rewrite 模块的指令，而其他的指令不是这样用的。

指令

Rewrite 模块提供了一组指令，这些指令不只是重写 URI。下表描述的这些指令根据使用环境情况来使用。

rewrite 使用环境： Server，location， if	正如我们以前讨论的，该指令允许你对当前请求的 URI 进行重写，因此对客户端的请求会被重新设置。 语法：rewrite regexp replacement [flag]; 这里的“regexp”是一个 URI 正则表达式，它的目的是匹配后面的“replacement”。 “flag”可以取下面的值。 ● last：当前的重写规则应该是最后应用。在这条规则应用之后，新的 URI 被 Nginx 处理并且会查找一个 location 区段。然而，后面的 rewrite 指令将会被忽略。 ● break：当前的重写规则被应用，但是对于被修改的 URI(不重新搜索匹配的 location 区段)，Ngin 不会初始化新的请求。就是说，不再重新启动搜索其他匹配的 location 区段，后面的 rewrite 指令将会被忽略。 ● redirect：返回 302Moved temporarily HTTP 重定向响应，并且通过一个替代的 URI 设置作为某一 location 的头(header)。(浏览器显示跳转后的 URL 地址) ● permanent：返回 301Moved permanently HTTP 重定向响应，被替代的 URI 作为 location 头的值。(浏览器显示跳转后的 URL 地址) ● 如果你指定一个以 http://开始的 URI 作为替代的 URI，Nginx 将自动使用 redirect 标签

指令和使用环境	描述

事实上，主要的不同在于能够使用在这两个区段的指令不同——有些指令可以用在 if 区段，而有的则不可以，正如你在指令列表中注意到的，在 location 区段中几乎所有的指令都可以使用。总的来说，最好的方法就是在 if 区段中插入来源于 Rewrite 模块的指令，而其他的指令不是这样用的。

指令

Rewrite 模块提供了一组指令，这些指令不只是重写 URI。下表描述的这些指令根据使用环境情况来使用。

指令和使用环境	描　　述
rewrite 使用环境： Server，location， if	正如我们以前讨论的，该指令允许你对当前请求的 URI 进行重写，因此对客户端的请求会被重新设置。 语法：rewrite regexp replacement [flag]; 这里的“regexp”是一个 URI 正则表达式，它的目的是匹配后面的“replacement”。 “flag”可以取下面的值。 ● last：当前的重写规则应该是最后应用。在这条规则应用之后，新的 URI 被 Nginx 处理并且会查找一个 location 区段。然而，后面的 rewrite 指令将会被忽略。 ● break：当前的重写规则被应用，但是对于被修改的 URI(不重新搜索匹配的 location 区段)，Ngin 不会初始化新的请求。就是说，不再重新启动搜索其他匹配的 location 区段，后面的 rewrite 指令将会被忽略。 ● redirect：返回 302Moved temporarily HTTP 重定向响应，并且通过一个替代的 URI 设置作为某一 location 的头(header)。(浏览器显示跳转后的 URL 地址) ● permanent：返回 301Moved permanently HTTP 重定向响应，被替代的 URI 作为 location 头的值。(浏览器显示跳转后的 URL 地址) ● 如果你指定一个以 http://开始的 URI 作为替代的 URI，Nginx 将自动使用 redirect 标签

rewrite
使用环境：
Server，location，if
	• 注意，访问者请求的 URI 被该指令处理为相对的 URI：不包含主机名和协议。例如 一个这样的请求 *http://website.com/documents/page.html*，会被处理为 /documents/page.html。
	• 解码：URI 相当于这样一个请求 *http://website.com/my%20page.htm*，会变为/my page.html
	• 不包含参数：对于这样一个请求 *http://website.com/page.php?id=1&p=2*，URI 将是/page.php。在改写 URI 的时候，你不需要考虑在替代的 URI 中包含的参数——Nginx 会替你做这个。如果希望 Nginx 在改写的 URI 中不包含参数，那么在替代的 URI 结尾插入一个“？”字符：rewrite ^/search/(.*)$ /search.php?q=$1?。
	示例：
	`rewrite ^/search/(.*)$ /search.php?q=$1;`
	`rewrite ^/search/(.*)$ /search.php?q=$1?;`
	`rewrite ^ http://website.com;`
	`rewrite ^ http://website.com permanent;`

break
使用环境：
server，
location，if
	该指令被用于阻止进一步的 rewrite 指令匹配。经过这一点后，URI 就被锁定，不能再改变。
	例如：
	`if (-f $uri) {`
	`  break;  # break if the file exists`
	`}`
	`if ($uri ~ ^/search/(.*)$) {`
	`  set $query $1;`
	`  rewrite ^ /search.php?q=$query?;`
	`}`
	在本例中，改写 /search/anything 类似查询为 /search.php?q=anything，然而，如果请求的文件存在(例如 /search/index.html)，break 指令会阻止 Nginx 改写 URI

指令和使用环境	描述

续表

指令和使用环境	描　述

rewrite
使用环境：
Server，location，if
	● 注意，访问者请求的 URI 被该指令处理为相对的 URI：不包含主机名和协议。例如 一个这样的请求 *http://website.com/documents/page.html*，会被处理为 /documents/page.html。
	● 解码：URI 相当于这样一个请求 *http://website.com/my%20page.htm*，会变为/my page.html
	● 不包含参数：对于这样一个请求 *http://website.com/page.php?id=1&p=2*，URI 将是/page.php。在改写 URI 的时候，你不需要考虑在替代的 URI 中包含的参数——Nginx 会替你做这个。如果希望 Nginx 在改写的 URI 中不包含参数，那么在替代的 URI 结尾插入一个“？”字符：rewrite ^/search/(.*)$ /search.php?q=$1?。
	示例：
	`rewrite ^/search/(.*)$ /search.php?q=$1;`
	`rewrite ^/search/(.*)$ /search.php?q=$1?;`
	`rewrite ^ http://website.com;`
	`rewrite ^ http://website.com permanent;`

break
使用环境：
server，
location，if
	该指令被用于阻止进一步的 rewrite 指令匹配。经过这一点后，URI 就被锁定，不能再改变。
	例如：
	`if (-f $uri) {`
	`  break;  # break if the file exists`
	`}`
	`if ($uri ~ ^/search/(.*)$) {`
	`  set $query $1;`
	`  rewrite ^ /search.php?q=$query?;`
	`}`
	在本例中，改写 /search/anything 类似查询为 /search.php?q=anything，然而，如果请求的文件存在(例如 /search/index.html)，break 指令会阻止 Nginx 改写 URI

return
使用环境：server，location，if
	中断请求处理过程，并且返回一个指定的 HTTP 状态代码。
	语法：return code;
	这里的 code 可以从下列状态码中选择：204, 400, 402 到 406, 408, 410, 411, 413, 416 和 500 到 504。另外，为了返回一个“HTTP 200 OK”状态代码，你可以使用 Nginx 特殊的代码 444(非标准状态码)，这样不会发送任何头(header)和主体(body)数据。
	例如：
	`if ($uri ~ ^/admin/) {`
	`  return 403;`
	`# 由于 Nginx 已经完成了请求`
	`# 因此下面的指令就不再执行`
	`  rewrite ^ http://website.com;`
	`}`

set
使用环境：server，location，if
	初始化或重定义一个变量。注意，有些变量是不可以被重定义的，例如，不能修改$uri。
	语法：set $variable value;
	例如：
	`set $var1 "some text";`
	`if ($var1 ~ ^(.*) (.*)$) {`
	`  set $var2 $1$2;  #concatenation`
	`  rewrite ^ http://website.com/$var2;`
	`}`

uninitialized_variable_warn 使用环境：http，server，location，if	如果设置为 **on**，那么在所用配置文件中有变量未被初始化的时候，Nginx 会发出消息记录。 语法：**on** 或 **off** `uninitialized_variable_warn on;`
rewrite_log 使用环境：http，server，location，if	如果设置为 **on**，Nginx 将在“**notice**” 错误级别(参考 **error_log** 指令)对 rewrite 引擎处理的每一个操作发出日志消息。 语法：on 或 off 默认值：off `rewrite_log off;`

指令和使用环境	描述

续表

指令和使用环境	描　　述

return
使用环境：server，location，if
	中断请求处理过程，并且返回一个指定的 HTTP 状态代码。
	语法：return code;
	这里的 code 可以从下列状态码中选择：204, 400, 402 到 406, 408, 410, 411, 413, 416 和 500 到 504。另外，为了返回一个“HTTP 200 OK”状态代码，你可以使用 Nginx 特殊的代码 444(非标准状态码)，这样不会发送任何头(header)和主体(body)数据。
	例如：
	`if ($uri ~ ^/admin/) {`
	`  return 403;`
	`# 由于 Nginx 已经完成了请求`
	`# 因此下面的指令就不再执行`
	`  rewrite ^ http://website.com;`
	`}`

set
使用环境：server，location，if
	初始化或重定义一个变量。注意，有些变量是不可以被重定义的，例如，不能修改$uri。
	语法：set $variable value;
	例如：
	`set $var1 "some text";`
	`if ($var1 ~ ^(.*) (.*)$) {`
	`  set $var2 $1$2;  #concatenation`
	`  rewrite ^ http://website.com/$var2;`
	`}`

uninitialized_variable_warn 使用环境：http，server，location，if	如果设置为 **on**，那么在所用配置文件中有变量未被初始化的时候，Nginx 会发出消息记录。 语法：**on** 或 **off** `uninitialized_variable_warn on;`
rewrite_log 使用环境：http，server，location，if	如果设置为 **on**，Nginx 将在“**notice**”错误级别(参考 **error_log** 指令)对 rewrite 引擎处理的每一个操作发出日志消息。 语法：on 或 off 默认值：off `rewrite_log off;`

通用重写规则

这里有一组重写规则，基本能够满足动态网站的需要，美化他们的网页链接，这得归功于 URL 重写机制。由于每一个网站的不同性，显而易见，你需要根据特定的位置进行调整。

执行搜索

这个重写规则的目的是为了搜索查询。搜索 URL 中包含的关键字。

输入的 URI	http://website.com/search/some-search-keywords
重写后的 URI	http://website.com/search.php?q=some-search-keywords
重写规则	rewrite ^/search/(.*)$ /search.php?q=$1?;

用户个人资料页面

大多数允许访问者注册的动态网站都提供了一个可以查看个人资料的网页。这种格式的 URL 可以用，它包含用户 ID 和用户名。

输入的 URI	http://website.com/user/31/James
重写后的 URI	http://website.com/user.php?id=31&name=James
重写规则	rewrite^/user/([0-9]+)/(.+)$/user.php?id=$1&name=$2?;

多个参数

有些网站对字符串参数使用不同的语法，例如 通过使用斜线“/”来分隔非命名参数。

输入的 URI	http://website.com/index.php/param1/param2/param3
重写后的 URI	http://website.com/index.php?p1=param1&p2=param2&p3=param3
重写规则	rewrite^/index.php/(.*)/(.*)/(.*)$/index.php?p1=$1&p2=$2&p3=$3?;

类维基百科格式

许多网站现在都采用维基百科引入的 URL 格式：一个前缀目录，后跟文章名称。

输入的 URI	http:// website.com/wiki/Some_keyword
重写后的 URI	http://website.com/wiki/index.php?title=Some_keyword
重写规则	rewrite ^/wiki/(.*)$ /wiki/index.php?title=$1?;

新网站的文章

这种 URL 结构经常被新网站使用，在 URL 中包含有文章内容的指示。这种格式由一个文章标识符，后跟一个斜线“/”和一个关键字列表。关键字经常可以忽略，不包含在重写的 URI 中。

输入的 URI	http://website.com/33526/us-economy-strengthens
重写后的 URI	http://website.com/article.php?id=33526
重写规则	rewrite ^/([0-9]+)/.*$ /article.php?id=$1?;

讨论板

绝大部分现代的电子公告板(bulletin board)现在使用更漂亮的 URL，它通过两个参数来显示如何建立这个查看话题(*topic view*)——话题(topic)标识符和出发点(starting post)。再次请求时，关键字会被忽略。

输入的 URI	http://website.com/topic-1234-50-some-keywords.html
重写的 URI	http://website.com/viewtopic.php?topic=1234&start=50
重写规则	rewrite^/topic-([0-9]+)-([0-9]+)-(.*)\.html$ /viewtopic.php?topic=$1&start=$2?;

SSI 模块

SSI 即服务器端包含(Server Side Includes)，它实际上是一种由 Nginx 解释的服务器端编程语言。它的名称基于一个事实，即该语言的大部分功能在使用时都以包含命令(include)的形式写入。早在 20 世纪 90 年代，为了提供动态网页，这样的语言就被使用了——从带有客户端脚本的简单的静态.html 文件到包含服务器端处理的复杂页面。在 HTML 源代码中，网站管理员现在能够插入服务器端解释的指令，包括更高级的预处理器，例如 PHP 或 ASP。

SSI 最著名的例证是 quote of the day。为了在网站每个网页中插入一个新的 quote of the day，网站管理员将不得不编辑每一个网页的 HTML 源代码，以手动替换过去的应用。使用服务器端包含，一个简单的命令足以简化这项任务：

```
<html>
   <head><title>My web page</title></head>
<body>
      <h1>Quote of the day: <!--# include file="quote.txt" -->
      </h1>
</body>
</html>
```

你只需要插入一个新的引用，编辑被引用文件 quoted.text 的内容。所有的网页将自动升级这个应用。直到今天，所有主流 Web 服务器(Apache，IIS，Lighttpd 等)都支持服务器端包含。

模块指令和变量

插入文件的实际内容都由 Nginx 解析，这会带来一个主要问题——Nginx 应该为什么样的文件解析 SSI 命令？如果是解析二进制文件，例如图像文件(.gif，.jpg，.png)或其他类型的媒体文件。需要使用该模块介绍的指令设法确保正确配置 Nginx。

ssi 使用环境：http，server，location，if	对文件启用 SSI 命令解析。Nginx 仅解析相当于通过 ssi_types 指令选择的 MIME 类型。 语法：on 或 off 默认值：off `ssi on;`
ssi_types 使用环境：http，server，location	定义可以被 SSI 解析的 MIME 文件类型，类型 text/html 总是支持的。 语法：ssi_types type1 [type2] [type3...]; 默认值：text/html `ssi_types text/plain;`

指令和使用环境	描述

SSI 最著名的例证是 quote of the day。为了在网站每个网页中插入一个新的 quote of the day，网站管理员将不得不编辑每一个网页的 HTML 源代码，以手动替换过去的应用。使用服务器端包含，一个简单的命令足以简化这项任务：

```
<html>
   <head><title>My web page</title></head>
<body>
      <h1>Quote of the day: <!--# include file="quote.txt" -->
      </h1>
</body>
</html>
```

你只需要插入一个新的引用，编辑被引用文件 quoted.text 的内容。所有的网页将自动升级这个应用。直到今天，所有主流 Web 服务器(Apache，IIS，Lighttpd 等)都支持服务器端包含。

模块指令和变量

插入文件的实际内容都由 Nginx 解析，这会带来一个主要问题——Nginx 应该为什么样的文件解析 SSI 命令？如果是解析二进制文件，例如图像文件(.gif，.jpg，.png)或其他类型的媒体文件。需要使用该模块介绍的指令设法确保正确配置 Nginx。

指令和使用环境	描　述
ssi 使用环境：http，server，location，if	对文件启用 SSI 命令解析。Nginx 仅解析相当于通过 ssi_types 指令选择的 MIME 类型。 语法：on 或 off 默认值：off `ssi on;`
ssi_types 使用环境：http，server，location	定义可以被 SSI 解析的 MIME 文件类型，类型 text/html 总是支持的。 语法：ssi_types type1 [type2] [type3...]; 默认值：text/html `ssi_types text/plain;`

ssi_silent_errors 使用环境：http，server，location	一些 SSI 命令可能会产生错误。发生这种情况的时候，Nginx 会在该命令的位置输出一条信息——“an error occurred wnile processing the directive.” 启用此选项，消息就不会出现了。 语法：on 或 off 默认值：off ssi_silent_errors off;
ssi_value_length 使用环境：http，server，location	SSI 命令有参数，可以接受一个值(例如，<!--# include file="value" -->)。该参数用于定义 Nginx 接受的最大长度。 语法：Numeric 默认值：256(字符数) ssi_value_length 256;
ssi_ignore_recycled_buffers 使用环境：http，server，location	设置为 on 的时候，该指令会阻止 Nginx 使用循环缓冲区。 语法：on 或 off 默认值：off
ssi_min_file_chunk 使用环境：http，server，location	如果缓冲区的大小大于 ssi_min_file_chunk，则数据被存储在一个文件中，然后再通过 sendfile 发送。其他情况下，直接从内存发出。 语法：数值 (大小) 默认值：1 024

一个急需注意的问题是如何使用 SSI 来管理自己的资源——通过在 location 或 server 区段级别中开启 SSI 模块，至少你应该够解析所有的 text/html 文件(实际上几乎所有的页面都会显示在客户端的浏览器中)，为了使 Nginx 的 SSI 模块更有效地工作，可考虑禁用不需要解析的文件。

首先，所有包含 SSI 命令的页面要使用.shtml (Server HTML)扩展名。然后，在配置文件中，在 location 级别的区段中，在特定的条件下开启 SSI 支持。提供的文件名必须以.shtml 结尾：

```
server {
    server_name website.com;
    location ~* \.shtml$ {
        ssi on;
    }
}
```

指令和使用环境	描述

续表

指令和使用环境	描　　述
ssi_silent_errors 使用环境：http，server，location	一些 SSI 命令可能会产生错误。发生这种情况的时候，Nginx 会在该命令的位置输出一条信息——“an error occurred wnile processing the directive.” 启用此选项，消息就不会出现了。 语法：on 或 off 默认值：off ssi_silent_errors off;
ssi_value_length 使用环境：http，server，location	SSI 命令有参数，可以接受一个值(例如，<!--# include file="value" -->)。该参数用于定义 Nginx 接受的最大长度。 语法：Numeric 默认值：256(字符数) ssi_value_length 256;
ssi_ignore_recycled_buffers 使用环境：http，server，location	设置为 on 的时候，该指令会阻止 Nginx 使用循环缓冲区。 语法：on 或 off 默认值：off
ssi_min_file_chunk 使用环境：http，server，location	如果缓冲区的大小大于 ssi_min_file_chunk，则数据被存储在一个文件中，然后再通过 sendfile 发送。其他情况下，直接从内存发出。 语法：数值 (大小) 默认值：1 024

一个急需注意的问题是如何使用 SSI 来管理自己的资源——通过在 location 或 server 区段级别中开启 SSI 模块，至少你应该够解析所有的 text/html 文件(实际上几乎所有的页面都会显示在客户端的浏览器中)，为了使 Nginx 的 SSI 模块更有效地工作，可考虑禁用不需要解析的文件。

首先，所有包含 SSI 命令的页面要使用.shtml (Server HTML)扩展名。然后，在配置文件中，在 location 级别的区段中，在特定的条件下开启 SSI 支持。提供的文件名必须以.shtml 结尾：

```
server {
    server_name website.com;
    location ~* \.shtml$ {
        ssi on;
    }
}
```

一方面，提交给 Nginx 的所有 HTTP 请求都将通过另外的正则表达式进行模式匹配；另一方面，静态的 HTML 文件或被其他程序解释的文件(例如 .php)没有必要解析。

最后，SSI 模块带来了两个变量：

- $date_local 根据当前系统的时区返回当前的时间；
- $date_gmt 根据服务器的时区返回当前的 GMT 时间。

SSI 命令

一旦在网站开启了 SSI 引擎，就可以着手写第一个动态 HTML 页面。此外，原理简单——在网站使用正常的 HTML 代码，在 HTML 代码中插入 SSI 命令。

这些命令使用特定的语法——初看，它们像一般的 HTML 注释：<!-- A comment -->，那是它的好处——如果你对你的文件意外禁用了 SSI 解析功能，那么 SSI 命令不能出现在客户端的浏览器，它们只能够作为 HTML 注释出现在源代码中。完整的语法为

```
<!--# command param1="value1" param2="value2" … -->
```

文件包含

Server Side Include 模块的主要命令显然就是 include 命令，它有两种不同的格式。

首先，你能够使用一个简单的文件包含，使用 include file 指令：

```
<!--# include file="header.html" -->
```

该命令会产生一个 HTTP 子请求(由 Nginx 处理)。产生的响应体会被插入命令本身的位置替换掉命令部分。

第二种情况是使用 include virtual 命令：

```
<!--# include virtual="/sources/header.php?id=123" -->
```

这种情况下服务器也会执行一个子请求，所不同的地方在于使用这种方法 Nginx 需要获取一个指定的文件(使用 include file 时，参数 wait 将自动开启)。其实，这两个参数能够插入 include 命令标签内。默认情况下，所有的 SSI 请求并行地同时发出请求，在高负荷的环境中会引起下载缓慢和超时。作为选择，你能够使用 wait="yes"参数来指定，让 Nginx 等待请求完成之后才移到其他 include。

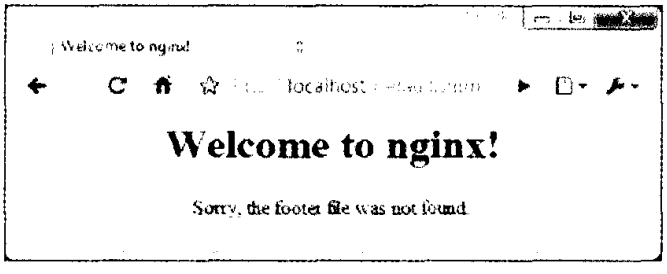
```
<!--# include virtual="header.php" wait="yes" -->
```

如果 include 命令的结果是空的或触发了一个错误(404，500，等等)，Nginx 会插入一个与其相对应的 HTML 错误页：<html>[…]404 Not Found</body></html>。

该消息显示在准确的位置(插入 include 命令的位置)。如果你想演示一下这个行为，可以建立一个命名块(block)，将 block 连接到 include 指令，那么一旦错误发生，该 block 的内容将显示在 include 命令的位置上。

```
<html>
<head><title>SSI Example</title></head>
<body>
<center>
<!--# block name="error_footer" -->Sorry, the footer file was
not found.<!--# endblock -->
<h1>Welcome to nginx</h1>
<!--# include virtual="footer.html" stub="error_footer" -->
</center>
</body>
</html>
```

在客户端浏览器的输出结果如下图所示。



可以看出，error_footer 块(block)的内容被插入 include 命令的位置，就在<h1>标签之后。

使用变量

Nginx 的 SSI 模块还提供了使用变量的能力。echo 命令用于显示一个变量(换句话说，在最终的 HTML 源代码中插入该变量值)：

```
<!--# echo var="variable_name" -->
```

该命令接受三个参数：

- var 要显示的变量名称，例如，REMOTE_ADDR 用来显示一个客户端的 IP 地址。
- default 以防变量为空设置的一个显示字符串。如果不指定该参数，输出将为(none)。
- ncoding 字符串的编码方式。能够接受的值有 none(没有特定的编码)；url(就像 URL 一样编码文本——一个空格变为 20%，等等)；entity(使用 HTML 实体：& 变为 &amp)。

也可以使用 set 命令设置自己的变量：

```
<!--# set var="my_variable" value="your value here" -->
```

value 参数由引擎自己解析，因此，你可以使用已有的变量：

```
<!--# echo var="MY_VARIABLE" -->
<!--# set var="MY_VARIABLE" value="hello" -->
<!--# echo var="MY_VARIABLE" -->
<!--# set var="MY_VARIABLE" value="$MY_VARIABLE there" -->
<!--# echo var="MY_VARIABLE" -->
```

这里是前面例子中的代码执行后，Nginx 输出的结果(相对于三个 echo 命令)：

```
(none)
hello
hello there
```

条件结构

下面的命令集将允许你包含文本或其他命令(依赖于条件)。条件结构可以使用下面的语法建立：

```
<!--# if expr="expression1" -->
[…]
<!--# elif expr="expression2" -->
[…]
<!--# else -->
[…]
<!--# endif -->
```

表达式可以定制三种不同方式：

- 检查一个变量：<!--# if expr="$variable" -->。类似于在 Rewrite 模块中的 if 块，如果变量不为空，那么条件为真；
- 比较两个字符串：<!--# if expr="$variable = hello" -->。如果第一个字符串和第二个字符串相等，那么条件为真。使用“!=”而不是“=”，如果第一个字符串不等于第二个字符串，那么条件为真；
- 正则表达式模式匹配：<!--# if expr="$variable = /pattern/" -->。注意，模式必须以“/”字符封闭。否则，会被视为一个简单的字符串。例如，<!--# if expr= "$MY_VARIABLE = /^/documents//" -->。类似的比较，否定条件则使用“!=”。

在一个条件块(condition block)中插入的内容能够包含常规的 HTML 代码，或另外的 SSI 指令，有一个例外——不能用在 if 块中。

配置

Nginx 提供的 SSI 命令，最后也最重要的(只有一次)是 config 命令了，它允许你配置两个简单的参数。

首先，当 SSI 引擎遇到错误时弹出的消息，可能由于格式有误的标签引起，也可能是无效的表达式，默认情况下，Nginx 会显示“an error occurred while processing the directive”，如果你想显示其他信息，则键入以下内容：

```
<!--# config errmsg="Something terrible happened" -->
```

另外，你还可以配置日期的格式(由$date_local 和 $date_gmt 返回的时间格式)，使用 timefmt 参数：

```
<!--# config timefmt="%A, %d-%b-%Y %H:%M:%S %Z" -->
```

这里指定的字符串被作为 strftime 这个 C 函数的格式字符串传递。关于可使用的格式字符串，更多的信息可以参考 strftime C 语言函数的文档，位于 *http://www.opengroup.org/onlinepubs/009695399/functions/strftime.html*。

其他模块

本章的前半部分覆盖了 Nginx 两个主要的模块，也就是 Rewrite 模块和 SSI 模块。这里还有许多模块，这么多模块使 Web 服务器的功能丰富起来，通过主题的形式对这些模块进行重组。

在这一小节描述的模块中，一些是在 Nginx 默认建立的时候就被包含在内，而有些不是。这表示，除非你在配置编译 Nginx 时指定包含这些模块(就像在第 2 章描述的那样)，否则它们对你来说无效(即不可用)。

站点访问和日志记录

下列这个模块允许你配置访问者如何访问网站和服务器如何记录访问请求。

Index 模块

Index 模块提供了一个简单的 index 指令，它允许你定义，如果客户端请求中没有指定文件名(换句话说，就是定义网站的 index 页)，Nginx 将提供默认网页。可以指定多个文件名，提供的则是第一个找到的文件。如果指定文件没有找到，那么如果 autoindex 指令被开启(核查 HTTP Autoindex 模块)，Nginx 会尝试产生一个自动的 index 文件，或者返回一个“403 Forbidden”错误页。

可选的方式，你可以插入一个绝对路径的文件(例如/page.html)，但是只能作为该指令的最后一个参数。

语法:

```
index file1 [file2…] [absolute_file];
```

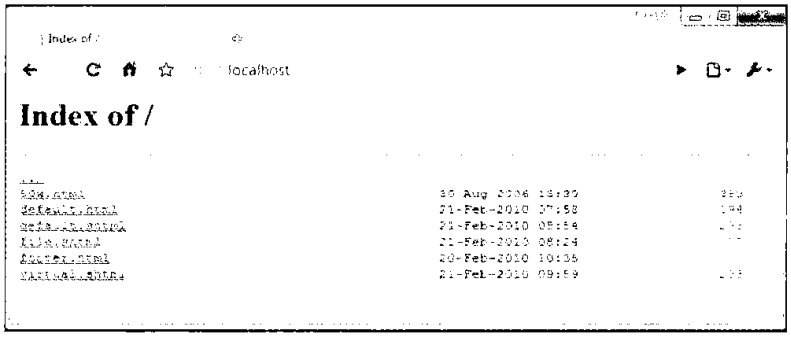
默认值：index.html.

```
index index.php index.html index.htm;
index index.php index2.php /catchall.php;
```

该指令的使用环境：http，server，location。

Autoindex 模块

如果 Nginx 不能为请求的目录提供一个 index 网页，那么默认的行为是返回一个“403 Forbidden HTTP”错误页。使用下图所示的指令设置，你能够得到一个自动的文件列表，它列出了被请求目录下所有的文件及目录。



由三列组成，每列显示文件的一个信息——文件名、文件的日期和时间和文件的大小(字节)。

指令和使用环境	描　　述
autoindex 使用环境：http，server，location	启用或禁用自动目录列表，用于丢失 index 网页的目录。 语法：on 或 off
autoindex_exact_size 使用环境：http，server，location	如果设置为 on，该指令会确保列出的文件大小以字节显示，否则会使用其他单位，例如 KB，MB 或 GB。 语法：on 或 off 默认值：on

autoindex_localtime 使用环境：http，server，location	默认情况下，该指令设置为 off，因此文件列出的日期和时间都是作为 GMT 时间。如果设置为 on，那么会使用本地服务器的时间。 语法：on 或 off 默认值：off

Random index 模块

该模块提供了一个简单的指令，那就是 random_index，它能够用于 location 区段中，可以为 Nginx 返回一个 index 网页，这个 index 网页是依据目录中的文件和子目录而随机生成的。

默认编译中，该模块不会包含在内。

语法：on 或 off

Log 模块

该模块用于控制 Nginx 访问日志的行为。对于系统管理员来说，这是一个核心模块，因为它允许你分析在 Nginx 运行时期内客户端的请求行为。它包含三个重要的指令，如下表所示。

access_log 使用环境：http，server，location	该指令定义了访问日志文件的路径，访问日志条目的格式，通过选择模板名称或启用访问日志文件。 语法：access_log path [format [buffer=size]] | off; 关于该指令语法的一些注意事项： ● 使用“access_log off”将在当前级别[1]中禁止访问日志的记录； ● 格式参数相当于一个利用 log_format 指令声明的一个模板，下面有描述； ● 如果没有指定格式参数，就应用默认的格式(combined)将会被使用 ● 可以在文件路径中使用变量

① 要理解这里的级别，它指的是 http 级别、server 级别还 location 级别。

指令和使用环境	描述

指令和使用环境	描述

续表

指令和使用环境	描　　述
autoindex_localtime 使用环境：http，server，location	默认情况下，该指令设置为 off，因此文件列出的日期和时间都是作为 GMT 时间。如果设置为 on，那么会使用本地服务器的时间。 语法：on 或 off 默认值：off

Random index 模块

该模块提供了一个简单的指令，那就是 random_index，它能够用于 location 区段中，可以为 Nginx 返回一个 index 网页，这个 index 网页是依据目录中的文件和子目录而随机生成的。

默认编译中，该模块不会包含在内。

语法：on 或 off

Log 模块

该模块用于控制 Nginx 访问日志的行为。对于系统管理员来说，这是一个核心模块，因为它允许你分析在 Nginx 运行时期内客户端的请求行为。它包含三个重要的指令，如下表所示。

指令和使用环境	描　　述
access_log 使用环境：http，server，location	该指令定义了访问日志文件的路径，访问日志条目的格式，通过选择模板名称或启用访问日志文件。 语法：access_log path [format [buffer=size]] | off; 关于该指令语法的一些注意事项： ● 使用“access_log off”将在当前级别[1]中禁止访问日志的记录； ● 格式参数相当于一个利用 log_format 指令声明的一个模板，下面有描述； ● 如果没有指定格式参数，就应用默认的格式(combined)将会被使用 ● 可以在文件路径中使用变量

① 要理解这里的级别，它指的是 http 级别、server 级别还 location 级别。

log_format
使用环境：http，server，location
	定义一个模板，用于描述访问日志中一个条目包含的内容。该模板由 access_log 指令使用。
	语法：
	log_format template_name format_string;
	默认的模板为 combined，与下例匹配：
	`log_format combined '$remote_addr - $remote_user [$time_local] '"$request" $status $body_bytes_sent '"$http_referer" "$http_user_agent"';`
	#other example
	`log_format simple '$remote_addr $request';`

open_log_file_cache 使用环境：http，server，location	为日志文件描述符配置缓存[①]。更多信息请参考 Core 模块的指令 open_file_cache 语法： open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time] | off; 它的参数类似于指令 open_file_cache 或相关指令的参数，差别仅在于这个只用于访问日志

Log 模块也带来了一些新的变量，但是只有在写日志条目时能使用：

- $connection 连接号
- $pipe 如果请求时管道，那么该变量被设置为'p'
- $time_local 本地时间(在写入日志条目时的时间)
- $msec 本地时间(在写入日志条目时的时间)微秒
- $request_time 处理请求时间的总长度，以微秒
- $status 响应状态码
- $bytes_sent 发送到客户端字节的总数
- $body_bytes_sent 发送到客户端响应体总的字节数
- $apache_bytes_sent 类似于$body_bytes，相当于 Apache 的 mod_log_config 指令的%B 参数
- $request_length 请求体的长度

① 将频繁使用的文件描述符保存到 open_log_file_cache 中。

指令和使用环境	描述

续表

指令和使用环境	描　　述

log_format
使用环境：http，server，location
	定义一个模板，用于描述访问日志中一个条目包含的内容。该模板由 access_log 指令使用。
	语法：
	log_format template_name format_string;
	默认的模板为 combined，与下例匹配：
	`log_format combined '$remote_addr - $remote_user [$time_local] '"$request" $status $body_bytes_sent '"$http_referer" "$http_user_agent"';`
	#other example
	`log_format simple '$remote_addr $request';`

open_log_file_cache 使用环境：http，server，location	为日志文件描述符配置缓存[①]。更多信息请参考 Core 模块的指令 open_file_cache 语法： open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time] | off; 它的参数类似于指令 open_file_cache 或相关指令的参数，差别仅在于这个只用于访问日志

Log 模块也带来了一些新的变量，但是只有在写日志条目时能使用：

- $connection　连接号
- $pipe　如果请求时管道，那么该变量被设置为'p'
- $time_local　本地时间(在写入日志条目时的时间)
- $msec　本地时间(在写入日志条目时的时间)微秒
- $request_time　处理请求时间的总长度，以微秒
- $status　响应状态码
- $bytes_sent　发送到客户端字节的总数
- $body_bytes_sent　发送到客户端响应体总的字节数
- $apache_bytes_sent　类似于$body_bytes，相当于 Apache 的 mod_log_config 指令的%B 参数
- $request_length　请求体的长度

① 将频繁使用的文件描述符保存到 open_log_file_cache 中。

限制和约束

下列模块允许你对网站上的文档进行访问管理——需要用户认证，匹配一组规则，或简单约束某些访问者。

auth_basic 模块

auth_basic 模块提供基本的认证功能。它有两个指令，你可以将它们用在网站指定的 location(server)区段中，这样就可以通过用户名和密码认证来约束用户访问。

```
location /admin/ {
    auth_basic "Admin control panel";
    auth_basic_user_file access/password_file;
}
```

第一个指令是 auth_basic，它可以设置为 off，或设置为文本消息，通常简称为“认证请求”(authentication challenge)，当一个客户端试图访问受保护的资源时，这个文本消息将会在 web 浏览器的用户名/密码的对话框上显示。

第二个指令是 auth_basic_user_file，定义密码文件的相对路径(与配置文件目录相对)。密码文件的格式行的语法为 username:password[:comment]。密码必须由 crypt(3)函数加密，例如 可以使用 Apache 的命令行工具 htpasswd 来生成加密密码。

Access 模块

该模块带来了两个重要的指令：allow 和 deny。它们允许你通过指定 IP 或 IP 范围来允许或拒绝被访问的资源。

两个指令有同样的语法：allow IP | CIDR | all，这里的 IP 就是 IP 地址，CIDR 是一个 IP 范围(CIDR 语法)，all 表示所有客户端的 IP。

```
location {
    allow 127.0.0.1; # 允许本地 IP 访问
    deny all; # 禁止所有其他 IP 地址
}
```

注意，规则的处理顺序为自上而下——如果你的第一条指令是“deny all”，那么随后所有例外的 allow 指令都没有任何效果。相反的情况也是如此——如果你开始使用了“allow all”，那么随后所有的 deny 指令都没有任何效果，因为你已经允许了所有的 IP 地址。

Limit zone 模块

该模块的机制比普通模块稍微复杂。它允许你定义一个 zone，然后可以通过限制该 zone 而达到限制服务器的最大并发连接数。

第一步使用 limit_zone 定义一个 zone：

- 通过 limit_zone zone_name $variable memory_max_size 来定义；
- zone_name 是一个随意给定的区域(zone)名；
- $variable 是一个变量，该变量的功能用于区别一个客户端和另一个客户端，一般使用$binary_remote_addr——二进制格式的 IP 地址(比 ASCII 更高效)；
- memory_max_size 用于设定分配给存储会话状态表内存的最大值。

下列例子基于客户端 IP 地址定义了一些 zone：

```
limit_zone myzone $binary_remote_addr 10m;
```

定义了 zone 之后可以使用 limit_conn 来限制连接：

```
limit_conn zone_name connection_limit;
```

应用于前面的例子便是：

```
location /downloads/ {
   limit_conn myzone 1;
}
```

结果，共享同一个$binary_remote_addr 的请求受到连接的限制(同时只能有一个连接)。如果达到限制，所有其他的连接请求收到的响应都是“503 Service unavailable[①]。

Limit request 模块

以同样的方式，Limit request 模块允许你限制被定义的 zone 的请求总数。

通过 limit_req_zone 指令定义一个 zone，它的语法不同于 Limit zone 模块中的对应指令：

```
limit_req_zone $variable zone=name:max_memory_size rate=rate;
```

指令的参数相同，除了结尾的 rate：用以表示每秒的请求数(r/s)或每分钟的请求数(r/m)。该请求率将应用到在 zone 内定义的客户端。要把 zone 添加到 location，可使用 limit_req 指令：

---

① 指令 limit_conn 的语法：

limit_conn zone 名字最大连接数

该指令有两个参数，第一个是通过指令 limit_zone 定义的区域名字；第二个是限制的连接数。

```
limit_req zone=name burst=burst [nodelay];
```

burst 参数定义了最大可能的突发请求——当收到客户端的请求数超过 zone 内定义的限制时，响应在一定程度上会延时。在一定程度上，burst 定义的值就是只能同时接受最大数量的请求值。如果超过这个限制，那么 Nginx 会返回“503 Service Unavailable”的 HTTP 错误响应。

```
limit_req_zone $binary_remote_addr zone=myzone:10m rate=2r/s;
[…]
    location /downloads/ {
    limit_req zone=myzone burst=10;
}
```

内容和编码

下面这组模块提供的功能会对服务器为客户端提供的内容产生影响，或者通过修改的方法来对响应进行编码，或者从头产生(从无到有，从零开始)响应的编码方式。

Empty GIF 模块

该模块的目的就是能够从内存提供一个 1×1 的透明 GIF 的指令。这样的文件时常被网页设计师用来调整其网站的外观。通过使用这个指令，你能够直接从内存中获取一个空的 GIF 文件，而不用从存储空间读取和处理实际的 GIF 文件。

要使用这个功能，在你选择的 location 块中插入 empty_gif 指令即可：

```
location = /empty.gif {
        empty_gif;
}
```

FLV 模块

该模块能够提供一个简单的功能，在提供 Flash Video (FLV)文件时会变得有用。它会解析一个特殊请求参数 start，该参数用来指出客户端希望下载的那个部分的偏移量(offset)。FLV 文件必须通过 URI: video.flv?start=XXX 来访问。

 默认编译的 Nginx 不包含这个模块。

要使用这个功能，只需简单地在你选定的 location 中插入指令 flv：

```
location ~* \.flv {
    flv;
}
```

HTTP headers 模块

该模块提供了两个指令，它们会影响发送到客户端的响应头。

首先，add_header Name value 让你在响应头中添加一个新行。语法为：add_header name：value 。该行仅被添加到响应代码为 200，204，301，302 和 304 响应头中。你可以在 value 参数中插入变量。

另外，指令 expires 允许你控制发送到客户端 Expires 和 Cache-Control HTTP header 的值，它影响的响应代码和前面的一样(200，204，301，302 和 304)。它接受以下值之一。

- off　两个头都不修改。
- 一个时间值　文件的有效期设置为“当前的时间”+“你指定的时间”。例如，“expires 24h”将返回一个从现在开始向前推 24 小时的时间值。
- epoch　将文件的有效期设置为 January 1, 1970。Cache-Control header 被设置为 no-cache。
- max　将文件的有效期设置为 December 31, 2037。“Cache-Control header”被设置为缓存 10 年。

Addition 模块

Addition 模块(参见本章末尾说明)允许你(通过简单指令)在 HTTP 响应体之前或者之后添加内容。


默认编译的 Nginx 不包含这个模块。

有两个主要指令：

```
add_before_body file_uri;
add_after_body file_uri;
```

按照以前的规定，Nginx 引发一个子查询来获取指定的 URI。另外，假如 location 区段类型指定不够，你可以追加定义文件内容的类型(默认为 text/html)：

```
addition_types mime_type1 [mime_type2…];
```

注意：0.7.64 版本之前，源代码中以前的指令都被拼写错误了——使用的是 addtion_types。在 0.7.65 版本中被修正了。
更多详情请参见书后的译者注 4。

Substitution 模块

顺着前面模块的路线，Substitution 模块允许你从响应体中查找和直接替换文本，语法格式如下：

```
sub_filter: searched_text replacement_text;
```

另外两个指令提供了更大的灵活性：

- sub_filter_once (on 或 off, 默认为 on) 只替换一次，替换文本第一次完成后就停止。
- sub_filter_types (默认为 text/html) 可以被替换的 MIME 类型，将符合条件的文本替换。


默认编译的 Nginx 不包含这个模块。

更多补充信息见书后译者注 5。

Gzip filter 模块

该模块允许你在将响应体发送到客户端之前将响应体利用 Gzip 算法压缩。要启用 Gzip 压缩，使用 gzip 指令(on 或 off)可以使用在 http，server，location，甚至 if 级别(但是不推荐)即可。下表所示的指令会帮助你配置过滤器选项。

gzip_buffers 使用环境：http，server，location	设定存储被压缩响应头的缓存数量和大小。 语法：gzip_buffers amount size; 默认值：gzip_buffers 44k (或 8k，依赖于具体的操作系统)
gzip_comp_level 使用环境：http，server，location	设定算法的压缩级别。指定值的范围从 1(最低的压缩级别，占用 CPU 较少，运算较快)到 9(最高的压缩级别，占用较多的 CPU，运算较慢)。 语法：数值 默认值：1
gzip_disable 使用环境：http，server，location	对于与指定的正则表达式匹配的用户代理，它们的 HTTP 请求头禁用 Gzip 压缩。 语法：正则表达式 默认值：无
gzip_http_version 使用环境：http，server，location	对指定的协议版本使用 Gzip 压缩。 语法：1.0 或 1.1 默认值：1.1
gzip_min_length 使用环境：http，server，location	如果响应体的长度低于指定的值，那么就不使用压缩。 语法：数值(大小) 默认值：0

指令使用环境	描述

Gzip filter 模块

该模块允许你在将响应体发送到客户端之前将响应体利用 Gzip 算法压缩。要启用 Gzip 压缩，使用 gzip 指令(on 或 off)可以使用在 http，server，location，甚至 if 级别(但是不推荐)即可。下表所示的指令会帮助你配置过滤器选项。

指令使用环境	描　　述
gzip_buffers 使用环境：http，server，location	设定存储被压缩响应头的缓存数量和大小。 语法：gzip_buffers amount size; 默认值：gzip_buffers 44k (或 8k，依赖于具体的操作系统)
gzip_comp_level 使用环境：http，server，location	设定算法的压缩级别。指定值的范围从 1(最低的压缩级别，占用 CPU 较少，运算较快)到 9(最高的压缩级别，占用较多的 CPU，运算较慢)。 语法：数值 默认值：1
gzip_disable 使用环境：http，server，location	对于与指定的正则表达式匹配的用户代理，它们的 HTTP 请求头禁用 Gzip 压缩。 语法：正则表达式 默认值：无
gzip_http_version 使用环境：http，server，location	对指定的协议版本使用 Gzip 压缩。 语法：1.0 或 1.1 默认值：1.1
gzip_min_length 使用环境：http，server，location	如果响应体的长度低于指定的值，那么就不使用压缩。 语法：数值(大小) 默认值：0

gzip_proxied 使用环境：http，server，location	启用或禁用 Gzip 压缩，用于从代理(参考后面章节中反向代理部分)服务器上接收到的响应体。 该指令接受下列参数，有些可以组合使用： ● off/any 对所有的请求启用/禁用压缩功能 ● expired 如果 Expires header 阻止缓存，则启用压缩 ● no-cache/no-store/private 如果 Cache-Control header 被设置为 no-cache，no-store 或 private，则启用压缩 ● no_last_modified 假如 Last-Modified header 没有设置，则启用压缩 ● no_etag 假如 ETag header 没有设置，则启用压缩 ● auth 假如设置了 Authorization header，则启用压缩
gzip_types 使用环境：http，server，location	除了默认的 text/html MIME 类型外，对其他类型也启用压缩功能。 语法：gzip_types mime_type1 [mime_type2…] 默认值：text/html (不能够被禁用)
gzip_vary 使用环境：http，server，location	向响应数据包中添加 Vary：Accept-Encoding HTTP 头。 语法：on 或 off 默认值：off
gzip_window 使用环境：http，server，location	设置用于 Gzip 操作的窗口缓冲的大小(windowBits 参数)。该指令所使用的值是从 Zlib 库调用的功能。 语法：数值(大小) 默认值：MAX_WBITS(来源于 Zlib 库)
gzip_hash 使用环境：http，server，location	设置分配给内部压缩状态(memLevel 参数)的内存总数。该指令所使用的值是从 Zlib 库调用的功能。 语法：数值 (大小) 默认值：MAX_MEM_LEVEL (来源于 Zlib 库)

指令使用环境	描述

续表

指令使用环境	描　　述
gzip_proxied 使用环境：http，server，location	启用或禁用 Gzip 压缩，用于从代理(参考后面章节中反向代理部分)服务器上接收到的响应体。 该指令接受下列参数，有些可以组合使用： ● off/any　对所有的请求启用/禁用压缩功能 ● expired　如果 Expires header 阻止缓存，则启用压缩 ● no-cache/no-store/private　如果 Cache-Control header 被设置为 no-cache，no-store 或 private，则启用压缩 ● no_last_modified　假如 Last-Modified header 没有设置，则启用压缩 ● no_etag　假如 ETag header 没有设置，则启用压缩 ● auth　假如设置了 Authorization header，则启用压缩
gzip_types 使用环境：http，server，location	除了默认的 text/html MIME 类型外，对其他类型也启用压缩功能。 语法：gzip_types mime_type1 [mime_type2…] 默认值：text/html (不能够被禁用)
gzip_vary 使用环境：http，server，location	向响应数据包中添加 Vary：Accept-Encoding HTTP 头。 语法：on 或 off 默认值：off
gzip_window 使用环境：http，server，location	设置用于 Gzip 操作的窗口缓冲的大小(windowBits 参数)。该指令所使用的值是从 Zlib 库调用的功能。 语法：数值(大小) 默认值：MAX_WBITS(来源于 Zlib 库)
gzip_hash 使用环境：http，server，location	设置分配给内部压缩状态(memLevel 参数)的内存总数。该指令所使用的值是从 Zlib 库调用的功能。 语法：数值 (大小) 默认值：MAX_MEM_LEVEL (来源于 Zlib 库)

postpone_gzipping 使用环境：http，server，location	在开始进行 Gzip 压缩前定义一个最小的数据阈值。 语法：大小 (数值) 默认值：0
gzip_no_buffer 使用环境：http，server，location	默认情况下，在将数据发送到客户端之前，Nginx 会等待，直到至少有一个缓冲(由 gzip_buffers 定义)被数据填满。如果开启这个指令，则禁用缓冲。 语法：on 或 off 默认值：off

Gzip static 模块

该模块为 Gzip 过滤机制(filter mechanism)添加一个简单的功能——当 gzip_static 指令(on 或 off)开启，Nginx 将自动查找.gz 文件——先查找一下，是否有该文件的压缩形式文件。这种机制允许 Nginx 发送之前压缩过的文档，而不是 Nginx 在百忙之中对每一个请求的文本都进行压缩。


默认编译的 Nginx 不包含这个模块。

如果一个客户端请求“/documents/page.html”，Nginx 会首先检测是否存在“/documents/page.html.gz”文件。如果找到.gz 文件，就将它提供给客户端。注意，即使在提供请求的文件之后，Nginx 也不会自己产生.gz 文件。

Charset filter 模块

通过设置 Charset filter 模块，可以更精确地控制响应体的字符集。你不但能够使用指令 charset 来设置参数，就是 Content-Type HTTP header 的编码(例如 text/html; charset=utf-8)，而且 Nginx 也能够使用指定的编码方法对数据进行自动重新编码。

charset 使用环境：http，server，location，if	该指令会为响应的 Content-Type 头添加指定的编码。如果指定的编码与 source_charset 设置的不同，Nginx 会重新对文档进行编码。 语法：charset encoding | off; 默认值：off 示例：charset utf-8;

指令使用环境	描述

指令使用环境	描　述

续表

指令使用环境	描　　述
postpone_gzipping 使用环境：http，server，location	在开始进行 Gzip 压缩前定义一个最小的数据阈值。 语法：大小 (数值) 默认值：0
gzip_no_buffer 使用环境：http，server，location	默认情况下，在将数据发送到客户端之前，Nginx 会等待，直到至少有一个缓冲(由 gzip_buffers 定义)被数据填满。如果开启这个指令，则禁用缓冲。 语法：on 或 off 默认值：off

Gzip static 模块

该模块为 Gzip 过滤机制(filter mechanism)添加一个简单的功能——当 gzip_static 指令(on 或 off)开启，Nginx 将自动查找.gz 文件——先查找一下，是否有该文件的压缩形式文件。这种机制允许 Nginx 发送之前压缩过的文档，而不是 Nginx 在百忙之中对每一个请求的文本都进行压缩。

默认编译的 Nginx 不包含这个模块。

如果一个客户端请求“/documents/page.html”，Nginx 会首先检测是否存在“/documents/page.html.gz”文件。如果找到.gz 文件，就将它提供给客户端。注意，即使在提供请求的文件之后，Nginx 也不会自己产生.gz 文件。

Charset filter 模块

通过设置 Charset filter 模块，可以更精确地控制响应体的字符集。你不但能够使用指令 charset 来设置参数，就是 Content-Type HTTP header 的编码(例如 text/html; charset=utf-8)，而且 Nginx 也能够使用指定的编码方法对数据进行自动重新编码。

指令使用环境	描　　述
charset 使用环境：http，server，location，if	该指令会为响应的 Content-Type 头添加指定的编码。如果指定的编码与 source_charset 设置的不同，Nginx 会重新对文档进行编码。 语法：charset encoding | off; 默认值：off 示例：charset utf-8;

source_charset 使用环境：http，server，location，if	定义初始响应的编码，如果该指令的值与在 charset 指令中指定的值不同，Nginx 会重新对文档进行编码。 语法：source_charset encoding;
override_charset 使用环境：http，server，location，if	Nginx 从代理或 FastCGI 网关收到一个响应时，该指令定义是否检测字符编码和进行重写。 语法：on 或 off 默认值：off
charset_types 使用环境：http，server，location	定义可用的 MIME 类型以便用于重新编码。 语法：charset_types mime_type1 [mime_type2…]; 默认值：text/html，text/xml，text/plain，text/vnd.wap.wml，application/x-javascript，application/rss+xml
charset_map 使用环境：http	重新定义字符编码表。表的每一行包括两个被执行的 16 进制编码，可以在 Nginx 默认配置目录下找到 koi8-r 字符集(koi-win 和 koi-utf)。 语法：charset_map src_encoding dest_encoding {...}

Memcached 模块

Memcached 是一个通过守护进程方式来运行的应用程序，它能够通过套接字连接。其主要目的，正如其名称所示，提供了一个高效的分布式键/值的内存缓存系统。Nginx Memcached 模块提供了下表所示的指令允许你配置访问 Memcached 守护进程。

memcached_pass 使用环境：location，if	定义 Memcached 守护进程的主机名和端口。 语法：memcached_pass hostname:port; 示例：memcached_pass localhost:11211;
memcached_connect_timeout 使用环境：http，server，location	定义连接超时，单位为毫秒(默认值为 60 000)。 示例：memcached_connect_timeout 5000;
memcached_send_timeout Context：http，server，location	定义数据写操作超时，单位为毫秒(默认值为 60 000)。 示例：memcached_send_timeout 5,000;

指令使用环境	描述

指令和使用环境	描述

续表

指令使用环境	描　　述
source_charset 使用环境：http，server，location，if	定义初始响应的编码，如果该指令的值与在 charset 指令中指定的值不同，Nginx 会重新对文档进行编码。 语法：source_charset encoding;
override_charset 使用环境：http，server，location，if	Nginx 从代理或 FastCGI 网关收到一个响应时，该指令定义是否检测字符编码和进行重写。 语法：on 或 off 默认值：off
charset_types 使用环境：http，server，location	定义可用的 MIME 类型以便用于重新编码。 语法：charset_types mime_type1 [mime_type2…]; 默认值：text/html，text/xml，text/plain，text/vnd.wap.wml，application/x-javascript，application/rss+xml
charset_map 使用环境：http	重新定义字符编码表。表的每一行包括两个被执行的 16 进制编码，可以在 Nginx 默认配置目录下找到 koi8-r 字符集(koi-win 和 koi-utf)。 语法：charset_map src_encoding dest_encoding {...}

Memcached 模块

Memcached 是一个通过守护进程方式来运行的应用程序，它能够通过套接字连接。其主要目的，正如其名称所示，提供了一个高效的分布式键/值的内存缓存系统。Nginx Memcached 模块提供了下表所示的指令允许你配置访问 Memcached 守护进程。

指令和使用环境	描　　述
memcached_pass 使用环境：location，if	定义 Memcached 守护进程的主机名和端口。 语法：memcached_pass hostname:port; 示例：memcached_pass localhost:11211;
memcached_connect_timeout 使用环境：http，server，location	定义连接超时，单位为毫秒(默认值为 60 000)。 示例：memcached_connect_timeout 5000;
memcached_send_timeout Context：http，server，location	定义数据写操作超时，单位为毫秒(默认值为 60 000)。 示例：memcached_send_timeout 5,000;

memcached_read_timeout 使用环境：http，server，location	定义数据读操作超时，单位为毫秒(默认值为60 000)。 例如：memcached_read_timeout 5,000;
memcached_buffer_size 使用环境：http，server，location	定义读和写缓冲的大小。 例如：memcached_buffer_size 8k;
memcached_next_upstream 使用环境：http，server，location	当指令 memcached_pass 被连接到一个 upstream 区段时(参考 Upstream 模块)，该指令会定义匹配条件，只有条件匹配才会跳到下一个 upstream 模块。 语法：可选值有 error、timeout、invalid_response、not_found 或 off 默认值：error timeout 例如：memcached_next_upstream off;

另外，你需要定义变量$memcached_key，用来放置或者从缓存中获取该元素的键。例如，可以使用“set $memcached_key $uri”或“$memcached_key $uri?$args”。

注意，Nginx 的 Memcached 模块只能从缓存中检索数据，并不存储请求的结果。在缓存中存储数据应该由服务器端脚本来完成，你只需要确定在服务器端脚本和 Nginx 配置中使用同样的命名方案。作为一个例子，我们决定使用 memcached 来从缓存中检索数据，如果请求的 URI 没有找到，那么该请求将被传递给代理(详情可参考第 7 章)：

```
server {
    server_name example.com;
     […]
    location / {
        set $memcached_key $uri;
        memcached_pass 127.0.0.1:11211;
        error_page 404 @notcached;
    }
    location @notcached {
        internal;
        # 如果文件没有被找到，那么转发请求到代理
        proxy_pass 127.0.0.1:8080;
    }
}
```

指令和使用环境	描　述

续表

指令和使用环境	描　　述
memcached_read_timeout 使用环境：http，server，location	定义数据读操作超时，单位为毫秒(默认值为60 000)。 例如：memcached_read_timeout 5,000;
memcached_buffer_size 使用环境：http，server，location	定义读和写缓冲的大小。 例如：memcached_buffer_size 8k;
memcached_next_upstream 使用环境：http，server，location	当指令 memcached_pass 被连接到一个 upstream 区段时(参考 Upstream 模块)，该指令会定义匹配条件，只有条件匹配才会跳到下一个 upstream 模块。 语法：可选值有 error、timeout、invalid_response、not_found 或 off 默认值：error timeout 例如：memcached_next_upstream off;

另外，你需要定义变量$memcached_key，用来放置或者从缓存中获取该元素的键。例如，可以使用“set $memcached_key $uri”或“$memcached_key $uri?$args”。

注意，Nginx 的 Memcached 模块只能从缓存中检索数据，并不存储请求的结果。在缓存中存储数据应该由服务器端脚本来完成，你只需要确定在服务器端脚本和 Nginx 配置中使用同样的命名方案。作为一个例子，我们决定使用 memcached 来从缓存中检索数据，如果请求的 URI 没有找到，那么该请求将被传递给代理(详情可参考第 7 章)：

```
server {
    server_name example.com;
     […]
    location / {
        set $memcached_key $uri;
        memcached_pass 127.0.0.1:11211;
        error_page 404 @notcached;
    }
    location @notcached {
        internal;
        # 如果文件没有被找到，那么转发请求到代理
        proxy_pass 127.0.0.1:8080;
    }
}
```

Image filter 模块

该模块提供图像处理功能，通过 GD Graphics Library(也称 gdlib)来实现。

默认编译的 Nginx 不包含这个模块。

确定在 location 区段使用(employ)下表所示的指令，只过滤图像文件。例如 location ~* \.(png|jpg|gif)$ { ... }。

image_filter 使用环境：location	让我们在将图像发送到客户端之前使用转换。这里有四种有效的选项： ● test 确保请求的文档是图像文件，如果测试失败，则返回一个“415 Unsupported media type ”HTTP 错误消息，说明类型不支持。 ● size 以一个简单的 JSON 响应组成格式表示图像的信息，信息包括大小和类型(例如，{ "img": {"width":50,"height":50,"type":"png"}}) 如果文件无效，则返回一个简单的{}。 ● resize width height 调整为指定尺寸的图片。 ● crop width height 按照规定选择图像的指定部分。 语法： image_filter (test|size|resize width height| crop width height 默认值：无 示例：image_filter resize 200 100;
image_filter_buffer 使用环境：http，server，location	设置处理图像的最大值。 语法：image_filter_buffer size 默认值：image_filter_buffer 1m;
image_filter_jpeg_quality 使用环境：http，server，location	设置输出 JPEG 图像的质量。 语法：image_filter_jpeg_quality [1...100] 默认值：image_filter_jpeg_quality 75;

指令和使用环境	描述

Image filter 模块

该模块提供图像处理功能，通过 GD Graphics Library(也称 gdlib)来实现。

默认编译的 Nginx 不包含这个模块。

确定在 location 区段使用(employ)下表所示的指令，只过滤图像文件。例如 location ~* \.(png|jpg|gif)$ { ... }。

指令和使用环境	描　述
image_filter 使用环境：location	让我们在将图像发送到客户端之前使用转换。这里有四种有效的选项： ● test　确保请求的文档是图像文件，如果测试失败，则返回一个“415 Unsupported media type ”HTTP 错误消息，说明类型不支持。 ● size　以一个简单的 JSON 响应组成格式表示图像的信息，信息包括大小和类型(例如，{ "img": {"width":50,"height":50,"type":"png"}}) 如果文件无效，则返回一个简单的{}。 ● resize width height　调整为指定尺寸的图片。 ● crop width height　按照规定选择图像的指定部分。 语法： image_filter (test|size|resize width height| crop width height 默认值：无 示例：image_filter resize 200 100;
image_filter_buffer 使用环境：http，server，location	设置处理图像的最大值。 语法：image_filter_buffer size 默认值：image_filter_buffer 1m;
image_filter_jpeg_quality 使用环境：http，server，location	设置输出 JPEG 图像的质量。 语法：image_filter_jpeg_quality [1...100] 默认值：image_filter_jpeg_quality 75;

XSLT 模块

在提供给客户端之前，XSLT 允许你对 XML 文件或者从后台服务器(代理和 FastCGI 等)收到的响应应用 XSLT 转换。该模块的指令如下表所示。

默认编译的 Nginx 不包含这个模块。

xml_entities 使用环境：http，server，location	指定一个包含象征元素的 DTD 文件(XML 实体)。 语法：文件路径 示例：xml_entities xml/entities.dtd;
xslt_stylesheet 使用环境：location	指定一个使用它自己参数的 XSLT 模板文件路径，变量可以插入这些参数。 语法：xslt_stylesheet template [param1] [param2...]; 示例：xslt_stylesheet xml/sch.xslt param=value;
xslt_types 使用环境：http，server，location	定义除 text/xml 以外可转换的 MIME 类型。 语法：MIME 类型 示例：xslt_types text/xml text/plain;

与访问者相关的模块

下面这组模块提供了附加功能，这些功能能够帮助你查找出关于访问者的更多信息，例如通过解析客户端的请求头来了解访问者浏览器的名称和版本号，请求分配标识符，等等。

Browser 模块

为了建立供日后配置使用的各种变量，Browser 模块解析了客户端请求的 User-Agent HTTP 头。产生的三个变量如下：

- $modern_browser　如果客户端浏览器被识别为一个现代的浏览器，则该变量值便由指令 modern_browser_value 定义；
- $ancient_browser　如果客户端浏览器被识别为一个旧的浏览器，则该变量的值便由指令 ancient_browser_value 定义；
- $msie　如果客户端使用的是 IE 浏览器，则这个变量被设置为 1。

指令和使用环境	描述

XSLT 模块

在提供给客户端之前，XSLT 允许你对 XML 文件或者从后台服务器(代理和 FastCGI 等)收到的响应应用 XSLT 转换。该模块的指令如下表所示。


默认编译的 Nginx 不包含这个模块。

指令和使用环境	描　　述
xml_entities 使用环境：http，server，location	指定一个包含象征元素的 DTD 文件(XML 实体)。 语法：文件路径 示例：xml_entities xml/entities.dtd;
xslt_stylesheet 使用环境：location	指定一个使用它自己参数的 XSLT 模板文件路径，变量可以插入这些参数。 语法：xslt_stylesheet template [param1] [param2...]; 示例：xslt_stylesheet xml/sch.xslt param=value;
xslt_types 使用环境：http，server，location	定义除 text/xml 以外可转换的 MIME 类型。 语法：MIME 类型 示例：xslt_types text/xml text/plain;

与访问者相关的模块

下面这组模块提供了附加功能，这些功能能够帮助你查找出关于访问者的更多信息，例如通过解析客户端的请求头来了解访问者浏览器的名称和版本号，请求分配标识符，等等。

Browser 模块

为了建立供日后配置使用的各种变量，Browser 模块解析了客户端请求的 User-Agent HTTP 头。产生的三个变量如下：

- $modern_browser　如果客户端浏览器被识别为一个现代的浏览器，则该变量值便由指令 modern_browser_value 定义；
- $ancient_browser　如果客户端浏览器被识别为一个旧的浏览器，则该变量的值便由指令 ancient_browser_value 定义；
- $msie　如果客户端使用的是 IE 浏览器，则这个变量被设置为 1。

为了帮助 Nginx 识别 web 浏览器，区分现代的浏览器和旧的浏览器，你需要插入多条 ancient_browser 和 modern_browser 指令。

```
modern_browser opera 10.0;
```

这个例子中，如果用户的 User-Agent HTTP 头包含 Opera 10.0，那么客户端浏览器就被视为现代浏览器。

更多补充说明参见书后的译者注 6。

Map 模块

就像 Browser 模块一样，Map 模块允许你建立值的映射，这依赖于变量。

```
Map $uri $variable {
    /page.html 0;
    /contact.html 1;
    /index.html 2;
    default 0;
}
rewrite ^ /index.php?page=$variable;
```

注意，map 指令只能插入 http 区段。在这个例子中，$variable 可能会有三个不同的值，如果$uri 被设置为“/page.html”，那么$variable 现在的定义是“0”；如果$uri 被设置为“/contact.html”，那么$variable 现在是“1”；如果$uri 被设置为“/index.html”，那么$variable 的值将等于 2。对于所有其他情况(默认)，$variable 的值被设置为 0，最后的指令根据变量$variable 改写 URL。除了默认值外，map 指令接受另一个特殊的关键字 hostnames，它允许你使用通配符来匹配主机名，例如 *.domain.com。

另外两个指令允许你调整 Nginx 的内存管理机制：

- map_hash_max_size　设置保存变量关系哈希表的最大值。
- map_hash_bucket_size　设置在 map 中一个条目的最大值。

更多补充说明参见书后的译者注 7。

Geo 模块

该模块的目的是提供一个近似于 map 指令的功能——影响基于客户端数据的一个变量(例如，IP 地址)，语法稍微有点不同，它允许你指定 IP 范围(CIDR 格式)：

```
geo $variable {
default unknown;
127.0.0.1 local;
123.12.3.0/24 uk;
92.43.0.0/16 fr;
}
```

关于该模块的更多说明，请参见书后的译者注 8。

GeoIP 模块

顾名思义，该模块与前边一些模块有相似之处，这个可选模块使用 MaxMind (*www.maxmind.com*)GeoIP 二进制数据库，为你提供了有关访问者精确的地理信息。你需要从 MaxMind 的网站上下载数据库文件，并且放在你的 Nginx 目录下。

默认编译的 Nginx 不包含这个模块。

然后，你只需要通过下面的任意一条命令指定数据库的路径：

```
geoip_country country.dat; # 国家信息数据库

geoip_city city.dat;       # 城市信息数据库
```

第一条指令开启了三个变量：$geoip_country_code(两个字母的国家代码)；$geoip_country_code3(三个字母的国家代码)；$geoip_country_name(国家的全名)。第二个指令同样包含这三个变量，但是它提供的是其他信息：$geoip_region，$geoip_city，$geoip_postal_code，$geoip_city_continent_code，$geoip_latitude 和 $geoip_longitude。

Geolp 的实例可参见书后的译者注 9。

UserID filter 模块

该模块利用发布 cookie 的方法为客户端提供一个标识符。在配置中，该标识符能够从变量$uid_got 和 $uid_set 进一步来访问。

userid 使用环境：http，server，location	开启或禁用发布及记录 cookie。 该指令可接受四个值： ● on 使用 v2 版的 cookie 并记录它们 ● v1 使用 v1 版的 cookie 并记录它们 ● log 不用发送 cookie 数据，但是记录进入的 cookie ● off 不发送 cookie 数据 语法：userid on|v1|log|off 默认值：userid off; 示例：useid on;
userid_service 使用环境：http，server，location	该指令用于指定发布 cookie 服务器的 IP 地址。 语法：userid_service ip; 默认值：服务器的 IP 地址
userid_name 使用环境：http，server，location	定义发布的 cookie 名称。 语法：userid_name name; 默认值：用户标识符 示例：userid_name uid;
userid_domain Context：http，server，location	定义发布 cookie 的域。 语法：userid_domain domain; 默认值：无(域名部分不发送) 示例：userid_domain example.com;
userid_path 使用环境：http，server，location	定义 cookie 的路径部分。 语法：userid_path path; 默认值：/ 示例：userid_path /;
userid_expires 使用环境：http，server，location	定义 cookie 的生存期。 语法：userid_expires date | max; 默认值：无 示例：userid_expires 365d;
userid_p3p 使用环境：http，server，location	给 P3P 头指派一个值，该值将和 cookie 一同发送。 语法: userid_p3p data; 默认值：无 示例：userid_p3p 'policyref="/w3c/p3p.xml",CP="CUR ADM OUR NOR STA NID"';

指令和使用环境	描述

UserID filter 模块

该模块利用发布 cookie 的方法为客户端提供一个标识符。在配置中，该标识符能够从变量$uid_got 和 $uid_set 进一步来访问。

指令和使用环境	描　　述
userid 使用环境：http，server，location	开启或禁用发布及记录 cookie。 该指令可接受四个值： ● on　使用 v2 版的 cookie 并记录它们 ● v1　使用 v1 版的 cookie 并记录它们 ● log　不用发送 cookie 数据，但是记录进入的 cookie ● off　不发送 cookie 数据 语法：userid on|v1|log|off 默认值：userid off; 示例：useid on;
userid_service 使用环境：http，server，location	该指令用于指定发布 cookie 服务器的 IP 地址。 语法：userid_service ip; 默认值：服务器的 IP 地址
userid_name 使用环境：http，server，location	定义发布的 cookie 名称。 语法：userid_name name; 默认值：用户标识符 示例：userid_name　uid;
userid_domain Context：http，server，location	定义发布 cookie 的域。 语法：userid_domain domain; 默认值：无(域名部分不发送) 示例：userid_domain　example.com;
userid_path 使用环境：http，server，location	定义 cookie 的路径部分。 语法：userid_path path; 默认值：/ 示例：userid_path　/;
userid_expires 使用环境：http，server，location	定义 cookie 的生存期。 语法：userid_expires date | max; 默认值：无 示例：userid_expires　365d;
userid_p3p 使用环境：http，server，location	给 P3P 头指派一个值，该值将和 cookie 一同发送。 语法: userid_p3p data; 默认值：无 示例：userid_p3p　'policyref="/w3c/p3p.xml",CP="CUR ADM OUR NOR STA NID"';

Referer 模块

该模块有一个简单的指令：valid_referers。它的目的是检查来自客户端请求的 Referer HTTP 头，并且可能会拒绝基于该值的请求。如果 referrer 被认为无效，那么$invalid_referer 设置为 1。在有效的 referrers 列表中，你可以使用下面的三种值：

- None　缺(absence) 被认为是有效的；
- Blocked　伪装的 referrer (例如 XXXXX) 也被认为是有效的；
- A server name　指定服务器名称被认为是有效的 referrer。

例如，下面定义的变量$invalid_referer，如果发现 referrer 无效，则会返回一个错误代码：

```
valid_referers none blocked *.website.com *.google.com;
    if ($invalid_referer) {
    return 403;
}
```

要知道，欺骗 Referer HTTP 头是一个非常简单的过程，因此检查客户端请求的 referrer 不能作为一个安全措施。
更多补充信息请见书后译者注 10。

Real IP 模块

该模块提供了一个简单的功能——它会通过在 X-Real-IP HTTP 头中指定的 IP 来代替客户端的 IP 地址，如果 Nginx 作为一个后端服务器(从本质上讲，它的效果类似于 Apache 的 mod_rpaf，详情参考第 7 章)，用于客户端访问一个在代理之后的 web 站点或者是用于从适当的头(header)中检索 IP 地址。要使用该功能，第一步是插入 real_ip_header 指令来定义被利用的 HTTP 头——X-Real-IP 或 X-Forwarded-For，第二步是定义被信任的 IP 地址，换句话说，就是被允许使用这些头的客户端。这可多亏了 set_real_ip_from 指令，它能够接受 IP 地址和 CIDR 范围。

```
real_ip_header X-Forwarded-For;
set_real_ip_from 192.168.0.0/16;
set_real_ip_from 127.0.0.1;
```


默认编译的 Nginx 不包含这个模块。

有关该模块的更多内容，请参见书后译者注 11。

SSL 和安全

Nginx 通过 SSL 模块提供了安全的 HTTP 功能，但也提供了另外一个叫 Secure Link 的模块，它能够在另一种方法下为你的网站和你的访问者提供安全帮助。

SSL 模块

SSL 模块能够提供 HTTPS 的支持，尤其是在 SSL/TLS 上的 HTTP。它能够通过提供证书来给网站提供安全访问，使用一个证书和下面的指令定义的其他参数就可以实现。

默认编译的 Nginx 不包含这个模块。

指令及其描述如下表所示。

ssl 使用环境：http，server	在指定的服务器开启 HTTPS。该指令相当于对应的 listen 443 ssl 或 listen port ssl，但不如它们普遍。 语法：on 或 off 默认值：ssl off;
ssl_certificate 使用环境：http，server	设置 PEM 证书(certificate)的路径。 语法：文件路径
ssl_certificate_key 使用环境：http，server	设置 PEM secret key 文件的路径。 语法：文件路径
ssl_client_certificate 使用环境：http，server	设置客户端 PEM certificate 的路径。 语法：文件路径
ssl_dhparam 使用环境：http，server	设置 Diffie-Hellman 参数文件的路径。 语法：文件路径
ssl_protocols 使用环境：http，server	指定使用的协议。 语法：ssl_protocols [SSLv2] [SSLv3] [TLSv1]; 默认值：ssl_protocols SSLv2 SSLv3 TLSv1;
ssl_ciphers 使用环境：http，server	指定使用的密码。可以列出有效的密码(cipher)，密码的生成可以在 shell 中使用命令 openssl ciphers 语法：ssl_ciphers cipher1[:cipher2...]; 默认值：ssl_ciphers ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
ssl_prefer_server_ciphers 使用环境：http，server	指定是否服务器密码优先于客户端密码。 语法：on 或 off 默认值：off

指令和使用环境	描　　述

SSL 和安全

Nginx 通过 SSL 模块提供了安全的 HTTP 功能，但也提供了另外一个叫 Secure Link 的模块，它能够在另一种方法下为你的网站和你的访问者提供安全帮助。

SSL 模块

SSL 模块能够提供 HTTPS 的支持，尤其是在 SSL/TLS 上的 HTTP。它能够通过提供证书来给网站提供安全访问，使用一个证书和下面的指令定义的其他参数就可以实现。

默认编译的 Nginx 不包含这个模块。

指令及其描述如下表所示。

指令和使用环境	描　述
ssl 使用环境：http，server	在指定的服务器开启 HTTPS。该指令相当于对应的 listen 443 ssl 或 listen port ssl，但不如它们普遍。 语法：on 或 off 默认值：ssl off;
ssl_certificate 使用环境：http，server	设置 PEM 证书(certificate)的路径。 语法：文件路径
ssl_certificate_key 使用环境：http，server	设置 PEM secret key 文件的路径。 语法：文件路径
ssl_client_certificate 使用环境：http，server	设置客户端 PEM certificate 的路径。 语法：文件路径
ssl_dhparam 使用环境：http，server	设置 Diffie-Hellman 参数文件的路径。 语法：文件路径
ssl_protocols 使用环境：http，server	指定使用的协议。 语法：ssl_protocols [SSLv2] [SSLv3] [TLSv1]; 默认值：ssl_protocols SSLv2 SSLv3 TLSv1;
ssl_ciphers 使用环境：http，server	指定使用的密码。可以列出有效的密码(cipher)，密码的生成可以在 shell 中使用命令 openssl ciphers 语法：ssl_ciphers cipher1[:cipher2...]; 默认值：ssl_ciphers ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
ssl_prefer_server_ciphers 使用环境：http，server	指定是否服务器密码优先于客户端密码。 语法：on 或 off 默认值：off

续表

指令和使用环境	描　　述
ssl_verify_client 使用环境：http，server	开启校验客户端提供的证书。 语法：on 或 off 默认值：off
ssl_verify_depth 使用环境：http，server	指定校验客户端证书链(chain)的深度。 语法：数值 默认值：1
ssl_session_cache 使用环境：http，server	配置用于 SSL 会话的缓存。 语法：off，none，builtin:size 或者 shared:name:size 默认：off(禁用 SSL 会话) 更多详情可参见书后译者注 12。
ssl_session_timeout 使用环境：http，server	开启 SSL 会话功能后，该指令定义使用会话的期限(即超时)。 语法：时间值 默认值：5 分钟

另外，可以使用下列有效变量：

- $ssl_cipher　指出当前请求使用的密码；
- $ssl_client_serial　指出客户端证书的序列号；
- $ssl_protocol　指出当前请求所使用的协议；
- $ssl_verify　如果客户端证书校验成功，那么设置该变量为 SUCCESS；
- $ssl_client_s_dn 和 $ssl_client_i_dn　指出客户端证书的主题(Subject)值和客户端证书发行者 DN；
- $ssl_client_cert 和 $ssl_client_raw_cert　返回客户端证书数据，这是为第二个变量提供的原始数据。

创建 SSL 证书

虽然 SSL 模块尽可能地提供了很多可能性，但其实在大多数情况下只用两个指令就可以建立一个安全的网站。下面将帮助你通过使用 SSL 证书来为网站配置 Nginx(在这个例子中，网站名称确定为 *secure.website.com*)。在此之前，要确保你已经具备可供支配的组成部分：

- 一个.key 文件，通过下列命令来产生(其他级别的加密工具也行)：

  ```
  openssl genrsa -out secure.website.com.key 1024
  ```

- 一个.csr 文件，通过下列命令产生：

  ```
  openssl req -new -key secure.website.com.key -out
  secure.website.com.csr
  ```

- 被作为认证授权网站证书文件，例如，*secure.website.com.crt*。(注意：为了从 CA 获取证书，你需要提供你的.csr 文件。)
- 由 CA 发行的证书文件，如果是从 GoDaddy.com 购买的证书，则会是这种形式，例如 *gd_bundle.crt*。

第一步是合并网站的证书和 CA 证书，通过下列命令将它们合并在一起：

```
cat secure.website.com.crt gd_bundle.crt > combined.crt
```

然后可以准备配置 Nginx，在 server 区段内配置安全内容：

```
server {
    listen 443;
    server_name secure.website.com;
    ssl on;
    ssl_certificate /path/to/combined.crt;
    ssl_certificate_key /path/to/secure.website.com.key;
    […]
}
```

Secure link 模块

Secure link 模块完全与 SSL 模块无关，该模块提供了一个基本保护，其做法是在允许一个用户访问一个资源之前，在 URL 中检查是否有一个特定的散列值：

```
location /downloads/ {
    secure_link_secret "secret";
    if ($secure_link = "") {
        return 403;
    }
    rewrite ^ /downloads/$secure_link break;
}
```

通过这个配置，要想访问在目录/downloads/的文档，必须提供一个包含有被请求文件之文件名和指令 secure_link_secret 指定的值。常规的访问，例如 *http://website.com/downloads/file.zip*，将以 403 错误告终。

默认编译的 Nginx 不包含这个模块。

包含在 URL 中的正确的哈希值是一个 MD5 哈希值，这个值由下列公式计算得出：MD5(文件名+ secure_link_secret 指令值)。在前面的例子中，如果客户端希望下载文件/downloads/file.zip，那么在他们访问的 URL 中需要提供一个 MD5 值，该值由前面的公式计算及 MD5(file.zipsecret)。在这里，最终的 URL 为：

```
http://website.com/downloads/63666cbff4e08672ebbb0ed3e7c2f011/file.zip
```

如果 URI 没有包含正确的哈希值，那么变量$secure_link 的值为空，否则它会被设

置为请求的文件名，并且能够应用于重写规则中。
有关该模块的更多详情，请参见本书最后的译者注 13。

其他杂项模块

剩下的三个模块为可选模块(都需要在编译时启用)，它们提供了其他高级功能。

Stub status 模块

Stub status 模块被设计为提供有关服务器当前的状态信息，例如，活动连接总数，被处理的请求总数，等等。要使用该模块，在 location 区段添加 stub_status 指令即可。所有匹配 location 区段的请求都将得到状态页：

```
location = /nginx_status {
    stub_status on;
    allow 127.0.0.1;  # 你可能想保护该信息
    deny all;
}
```


默认编译的 Nginx 不包含这个模块。

Nginx 处理的结果如下所示：

```
Active connections: 1
server accepts handled requests
10 10 23
Reading: 0 Writing: 1 Waiting: 0
```

这个结果很有趣，注意一些服务器监控方法，例如 Monitorix 通过模块 stub status 提供对 Nginx 的支持——每隔一定的间隔就解析 Nginx 服务器的状态。
有关该模块的更多详情，可参考书后的译者注 14。

Google-perftools 模块

该模块是用于 Nginx 工作进程的(Google Performance Tools)Google 工具性能分析机制。该工具生成一个报告，生成的报告是基于性能分析的可执行代码，有关该项目的更多信息，可以从官方网站了解：*http://code.google.com/p/google-perftools/*。

默认编译的 Nginx 不包含这个模块。

为了开启这个功能，你需要指定由指令 google_perftools_profiles 产生的报告文件的路径：

```
google_perftools_profiles logs/profiles;
```

WebDAV 模块

WebDAV 是一个众所周知的 HTTP 协议的扩展，网站上 HTTP 被设计为为访问者提供资源下载(即读数据)时，WebDAV 提供了扩展功能，为 Web 服务器提供写操作，例如，建立文件和目录，移动和复制文件，等等。Nginx 的 WebDAV 模块实现了 WebDAV 协议的一个小子集。

默认编译的 Nginx 不包含这个模块。

该模块的指令如下表所示。

指令和使用环境	描　　述
dav_methods 使用环境：http，server，location	选择你要使用(支持)的 DAV 方法 语法：dav_methods [off | [PUT] [DELETE] [MKCOL] [COPY] [MOVE]]; 默认值：off
dav_access 使用环境：http，server，location	在当前的级别(指 http，server，location。)中定义访问权限。 语法：dav_access [user:r|w|rw] [group:r|w|rw] [all:r|w|rw]; 默认值：dav_access user:rw;
create_full_put_path 使用环境：http，server，location	该指令定义了当客户请求在一个不存在的目录下新建文件时的行为。如果设置为 on，则新建这个不存在的目录；如果设置为 off，则新建文件失败。 语法：on 或 off 默认值：off
min_delete_depth 使用环境：http，server，location	该指令定义了在执行删除操作时(删除文件或目录)指定的最小 URI 深度。 语法：数值 默认值：0

有关该模块的更多详情，可参见书后的译者注 16。

第三方模块

在过去的几年中，Nginx 社区一直在成长、变大，有第三方开发人员写出了很多其他模块。这些模块可以从官方的 wiki 站点下载：*http://wiki.nginx.org/nginx3rdPartyModules*。

目前可用的模块提供广泛的功能：

- Access Key 模块保护文档，它的方式类似于 Secure link 模块，作者 *Mykola Grechukh*；
- Fancy Indexes 模块改进 Nginx 产生的自动目录列表，作者 Adrian Perez de Castro；
- Headers More 模块提高了 HTTP header 灵活性，作者 Yichun Zhang (agentzh)；
- 还有为 Web 服务器各个不同部分编写的更多功能模块。

要将第三方的模块集成到 Nginx 中，需要按照下面的简单三步去做。

1. 下载相关功能模块的.tar.gz 归档文件；
2. 使用命令 tar xzf module.tar.gz 解压。
3. 通过命令./configure --add-module=/module/source/path [...]配置编译 Nginx。

一旦完成编译安装该应用程序，就可以使用该模块，就像 Nginx 的其他模块一样使用，它们有自己的变量和指令。

如果你自己对编写 Nginx 模块感兴趣，Evan Miller 发布过一本优秀的“攻略”：*Emiller' s Guide to Nginx Module Development*。完整的指南查阅他的个人网站：*http://www.evanmiller.org/*。

小结

通过本章的阅读，我们了解了 Nginx 的模块，这些模块有助于提高或优化 Web 服务器的配置。就功能、虚拟主机的处理方式和配置方式而言，Nginx 从其他 Web 服务器中脱颖而出，足以说服了许多管理员使用它。

另外有三个模块没有介绍，第一是 FastCGI 模块，将在第 7 章中介绍，它允许配置网关，用于其他应用程序，如 PHP 或 Python；第二是 proxy 模块，允许你设计复杂的设置，将在第 7 章中讲述；最后是 Upstream 模块，它用于前两个模块，因此会与前两个模块一起详述。

第 6 章

Nginx 与 PHP、Python

21 世纪的前 10 年是服务器端技术的十年。在过去的十年中，绝大多数网站由静态的 HTML 内容转为高度而且完全动态的页面，从用户交互的角度来看，这种技术将网络带入一个全新的级别。软件解决方案的迅速崛起，包括开源，一些软件变得足够成熟能够处理高流量的网站。在本章中，我们将研究 Nginx 与其他应用程序的交互(配合)能力。我们保留了这两个应用，是基于两个不同的原因：第一个原因很明显，PHP，市场占有率高——根据 Nginx 服务的市场调查，到 2008 年，将近 33%的网络服务器由 PHP 驱动；第二个原因是 Python——与 Nginx 一同工作的安装配置方法简单(并且 Python 的应用也越来越广泛——译者注)。这种轻松的机制能够将其应用到其他应用程序，如 Perl 或 Rails 框架下的 Ruby。

本章主题

- 认识 CGI 和 FastCGI
- Nginx 的 FastCGI 模块
- 通过 Upstream 模块实现负载均衡
- 设置 PHP 和 PHP-FPM
- 设置 Python 和 Django
- 配置 Nginx 和 PHP、Python 一同工作

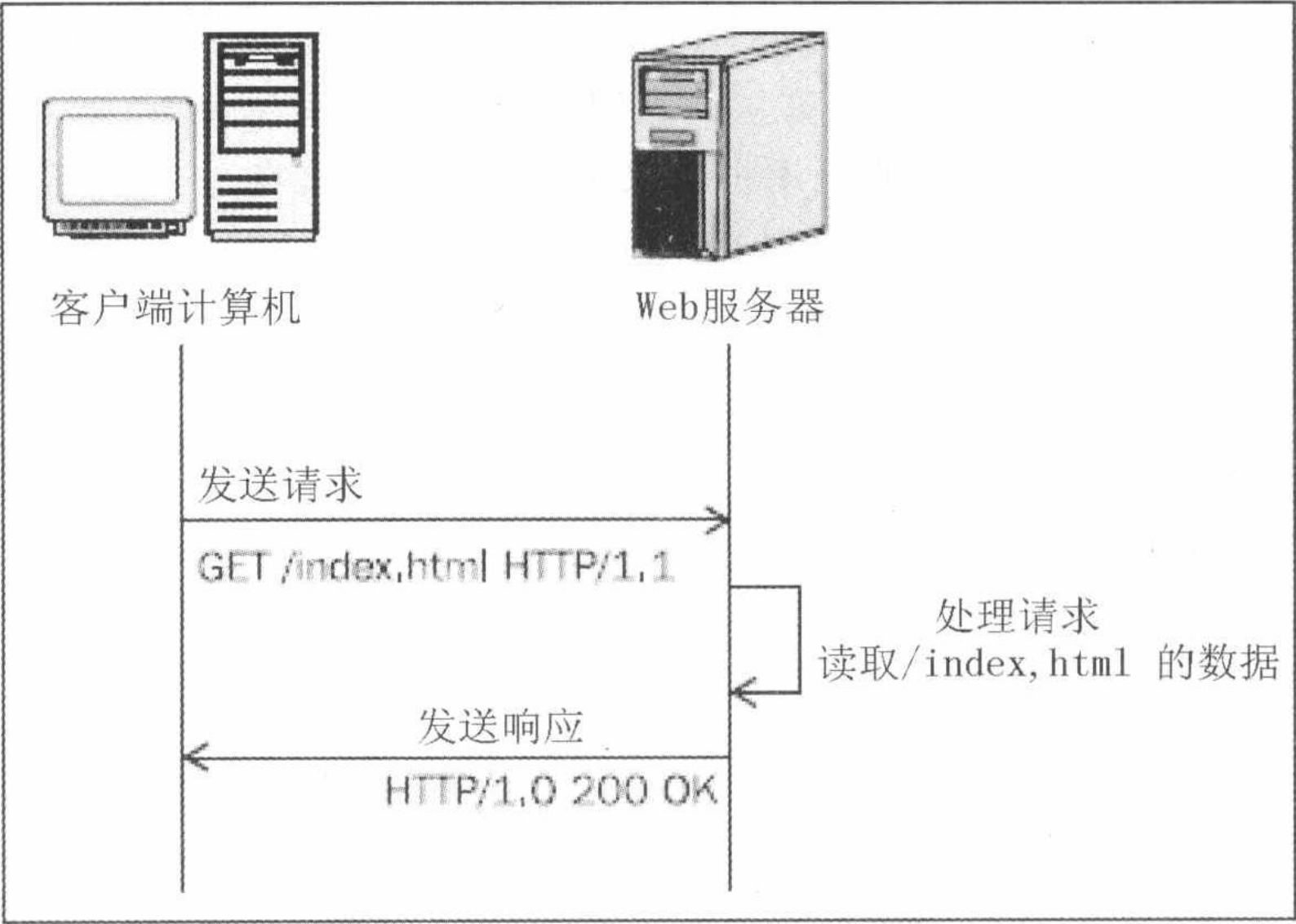
FastCGI 入门

顾名思义，FastCGI 实际上是以 CGI 变化而来的。因此，首先要解释 CGI，下面通过介绍 FastCGI 来详细了解。

理解 Web 服务的机制

如下图所示，Web 服务器最初的目的是响应客户端的请求，提供存储在驱动器上的文件。客户端发送一个请求要求下载文件，服务器响应该请求，处理并发送给客户端适当的响应代码，如果文件能够正常提供，则响应——200 OK；如果文件无法找到，则响应——404 及其他的响应。

这种机制自从万维网问世之初就已经开始使用，现在仍然如此。然而，如前所述，静态网站正在逐步被抛弃，而费用花在动态网站上，动态网站包含通过应用程序处理的脚本(例如这些应用程序中的 PHP 和 Python)。于是，Web 服务机制发展成为下图所示的形式。

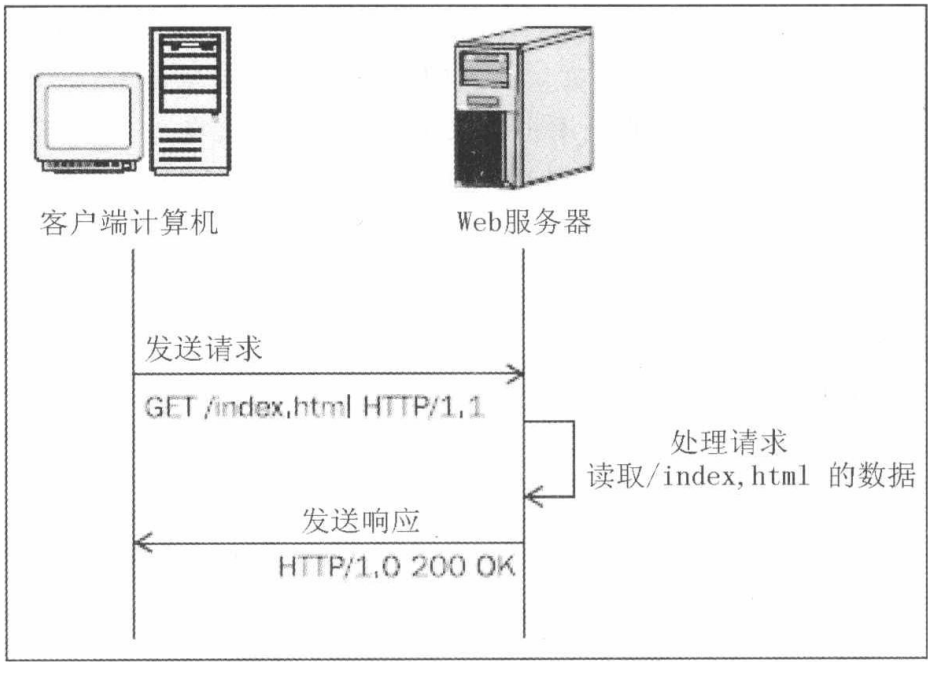

客户端计算机
Web服务器
发送请求
GET /index.html HTTP/1.1
处理请求
读取/index.html 的数据
发送响应
HTTP/1.0 200 OK

FastCGI 入门

顾名思义，FastCGI 实际上是以 CGI 变化而来的。因此，首先要解释 CGI，下面通过介绍 FastCGI 来详细了解。

理解 Web 服务的机制

如下图所示，Web 服务器最初的目的是响应客户端的请求，提供存储在驱动器上的文件。客户端发送一个请求要求下载文件，服务器响应该请求，处理并发送给客户端适当的响应代码，如果文件能够正常提供，则响应——200 OK；如果文件无法找到，则响应——404 及其他的响应。



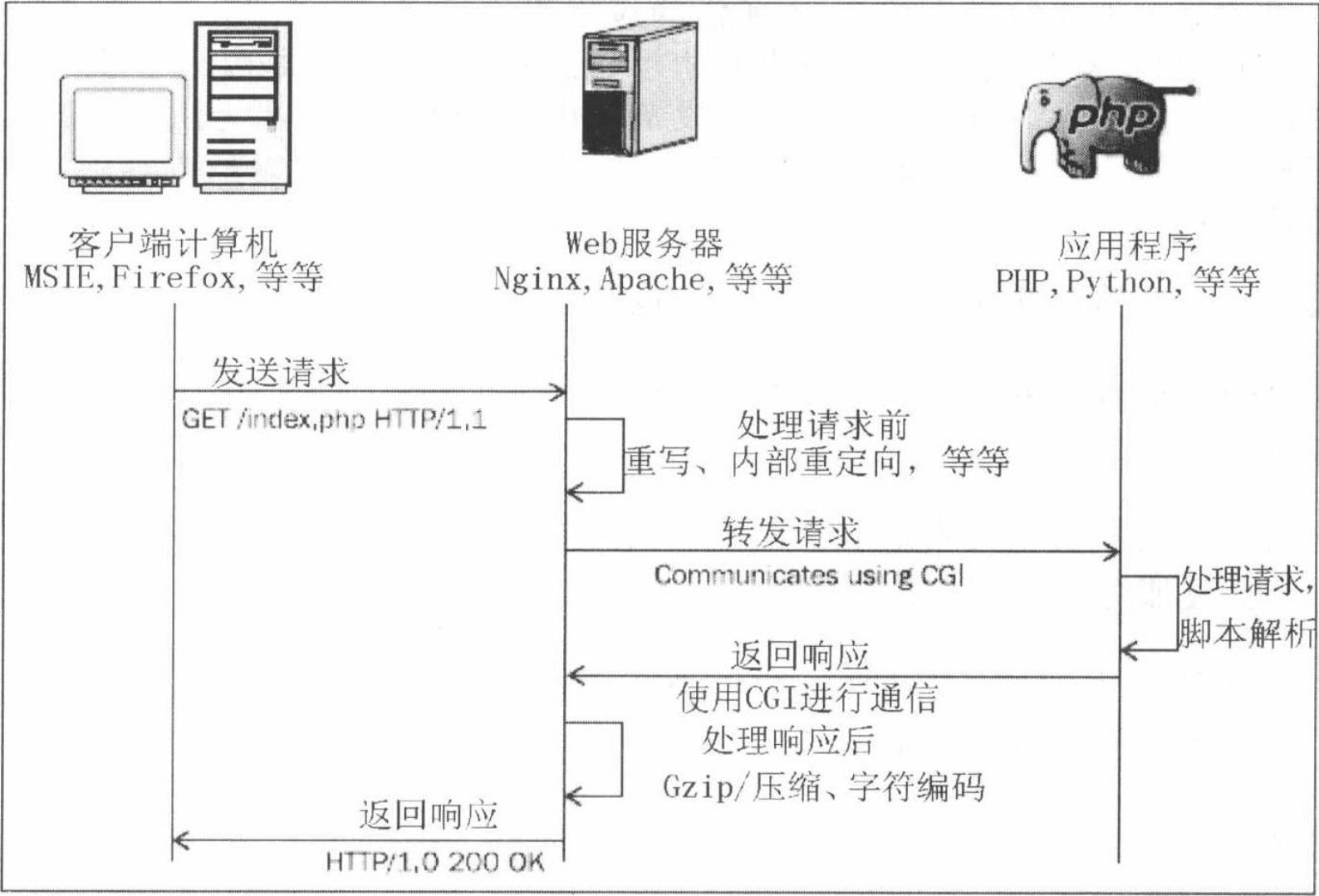
这种机制自从万维网问世之初就已经开始使用，现在仍然如此。然而，如前所述，静态网站正在逐步被抛弃，而费用花在动态网站上，动态网站包含通过应用程序处理的脚本(例如这些应用程序中的 PHP 和 Python)。于是，Web 服务机制发展成为下图所示的形式。

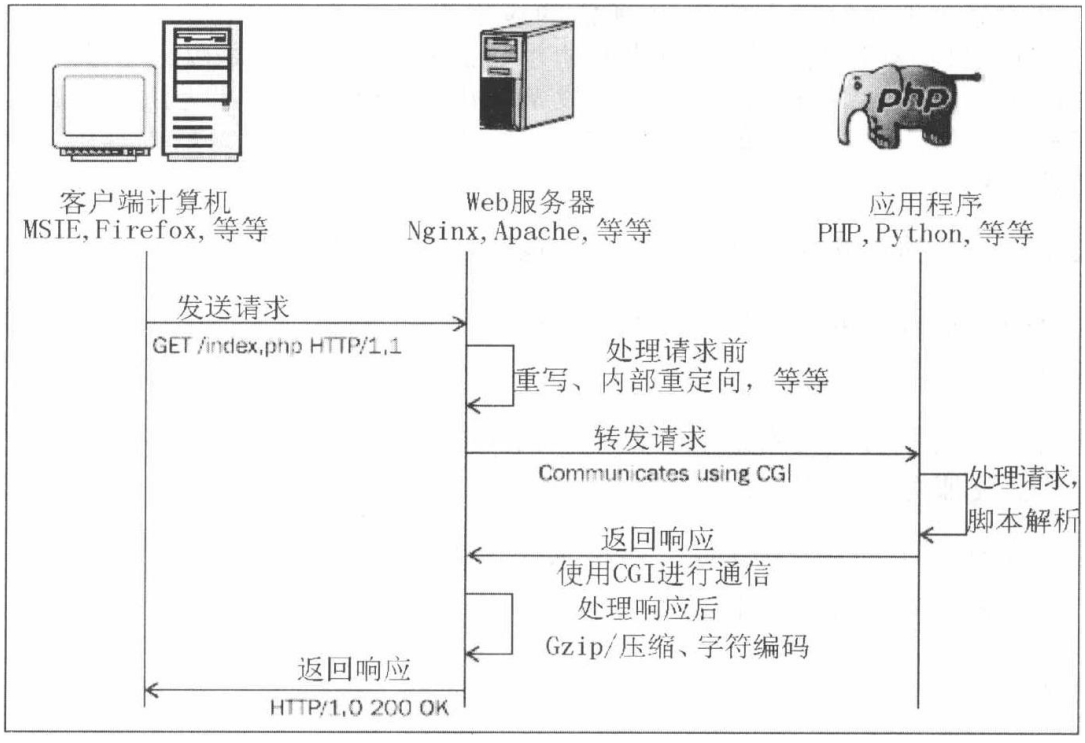
客户端试图访问动态页面时，Web 服务器收到请求，接着转发给第三方的应用程序，应用程序独立处理完该脚本，然后再将结果返回给产生响应的 Web 服务器，最后转发响应至客户端。

Web 服务器与应用程序按次序通信，CGI 协议最早是在 20 世纪 90 年代发明的。

CGI 通用网关接口

按照 RFC 3875 的规定(CGI protocol v1.1)，国际互联网学会(ISOC)的计划，通用网关接口(CGI)允许一个 HTTP 服务和 CGI 脚本负担责任——响应客户端的 requests.[...]。服务器(在这里当然指的是 Nginx)响应管理连接、数据传送、传输及与客户端相关的网络问题，而 CGI 脚本处理诸如数据访问和文档处理等应用程序问题。


客户端计算机
MSIE, Firefox, 等等
Web服务器
Nginx, Apache, 等等
应用程序
PHP, Python, 等等
发送请求
GET /index.php HTTP/1.1
处理请求前
重写、内部重定向，等等
转发请求
Communicates using CGI
处理请求，
脚本解析
返回响应
使用CGI进行通信
处理响应后
Gzip/压缩、字符编码
返回响应
HTTP/1.0 200 OK



客户端试图访问动态页面时，Web 服务器收到请求，接着转发给第三方的应用程序，应用程序独立处理完该脚本，然后再将结果返回给产生响应的 Web 服务器，最后转发响应至客户端。

Web 服务器与应用程序按次序通信，CGI 协议最早是在 20 世纪 90 年代发明的。

CGI 通用网关接口

按照 RFC 3875 的规定(CGI protocol v1.1)，国际互联网学会(ISOC)的计划，通用网关接口(CGI)允许一个 HTTP 服务和 CGI 脚本负担责任——响应客户端的 requests.[...]。服务器(在这里当然指的是 Nginx)响应管理连接、数据传送、传输及与客户端相关的网络问题，而 CGI 脚本处理诸如数据访问和文档处理等应用程序问题。

CGI 是一个协议，它描述了 Web 服务器(Nginx)和应用程序网关(PHP, Python, 等等)之间的信息交换。在实践中，当 Web 服务器收到一个应该转发到该程序网关的请求时，只是简单地执行所需的命令(所需的应用程序)，例如/usr/bin/php。客户端请求的详细信息(例如，用户代理和其他请求信息)通过命令行参数或环境变量来传递，然而实际的数据则通过标准的输入设备利用 POST 或 PUT 方法来传递，再调用应用程序，然后将处理完的文本内容写入标准的输出设备，最后内容由 Web 服务器重新捕获。

这种技术看上去简单并且足够有效，但它也存在一些明显的不足。

1. 为每一个请求产生一个唯一的进程，从一个请求到另一个请求，内存和其他的信息将全部丢失；
2. 开启一个进程会消耗系统的资源，大而重的并发请求(每一个产生的进程)数量很快会使服务器一团糟；
3. 很难设计一个架构，从而把 web 服务器和应用程序网关放在不同的服务器上。如果需要，也并非不可能。

FastCGI

上面提到的问题表明 CGI 协议相关的服务器效率低下，为了寻求解决的方法，20 世纪 90 年代中期开源市场(Open Market)产生了由 CGI 到 FastCGI 的演变。在过去的十五年中，这种技术已经变成主要的标准，许多 Web 服务器现在都提供这种功能——甚至包括商业软件，例如 Microsoft IIS。

虽然目的一样，但 FastCGI 在 CGI 的基础上做了重大的改进，它基于下面的原理。

- FastCGI 使用了能够处理多个请求的持续进程，而不是针对每个请求都产生新的进程。
- Web 服务器和应用程序网关通信是通过 TCP 套接字或 POSIX 本地 IPC 套接字进行的，因此，两个进程可能在网络上的不同计算机中。
- Web 服务器将客户端请求转发到应用程序网关，并使用单个连接来收到响应，其他的请求也可以跟进，从而不用再为其他的请求建立连接。注意，在大部分 Web 服务器上，包括 Nginx 和 Apache，FastCGI 的执行不支持(或至少不全部支持)多路技术。
- FastCGI 是一个基于套接字的协议，因此它能够适用于任何平台及任何编程语言。

在这一章中，我们将通过 FastCGI 设置 PHP 和 Python，你可能会发现这种机制有些类似于其他应用程序，例如 Perl 或 Rails 框架下的 Ruby。

设计一个 FastCGI 驱动结构实际上没有想象的那么复杂。只要有 Web 服务和处理应用的程序在运行，那么唯一的难点就是建立这两部分的连接。根据这个观点，第一步是配置 Nginx 和 FastCGI 应用程序通信，FastCGI 与 Nginx 的和谐共处由 FastCGI 来完成。这一小节详细讲述该模块提供的指令。

主要指令

在默认情况下，FastCGI 模块包含在已编译的 Nginx 中，你不用在编译时手动开启。通过下列指令，允许你配置 Nginx 将请求传递给 FastCGI 应用程序。注意，Nginx 的配置文件目录下有一个名为 fastcgi_params 的文件，该文件中定义的指令值在大多数情况下都有效。

fastcgi_pass
使用环境：
location，if
	指定什么请求应该转递给 FastCGI 服务器，具体做法是先定义 fastcgi_pass，然后再在 location 中应用。fastcgi_pass 有以下两种语法：
	● 对于 TCP 套接字
	`fastcgi_pass hostname:port;`
	● 对于 Unix 套接字
	`fastcgi_pass unix:/path/to/fastcgi.socket;`
	也可以参考 upstream 块(详情参见后文):
	● fastcgi_pass myblock;
	示例：
	`fastcgi_pass localhost:9000;`
	`fastcgi_pass 127.0.0.1:9000;`
	`fastcgi_pass unix:/tmp/fastcgi.socket;`
	`# 使用 upstream 块`
	`upstream fastcgi {`
	`  server 127.0.0.1:9000;`
	`  server 127.0.0.1:9001;`
	`}`
	`location ~* \.php$ {`
	`  fastcgi_pass fastcgi;`
	`}`

指令和使用环境	描述

在这一章中，我们将通过 FastCGI 设置 PHP 和 Python，你可能会发现这种机制有些类似于其他应用程序，例如 Perl 或 Rails 框架下的 Ruby。

设计一个 FastCGI 驱动结构实际上没有想象的那么复杂。只要有 Web 服务和处理应用的程序在运行，那么唯一的难点就是建立这两部分的连接。根据这个观点，第一步是配置 Nginx 和 FastCGI 应用程序通信，FastCGI 与 Nginx 的和谐共处由 FastCGI 来完成。这一小节详细讲述该模块提供的指令。

主要指令

在默认情况下，FastCGI 模块包含在已编译的 Nginx 中，你不用在编译时手动开启。通过下列指令，允许你配置 Nginx 将请求传递给 FastCGI 应用程序。注意，Nginx 的配置文件目录下有一个名为 fastcgi_params 的文件，该文件中定义的指令值在大多数情况下都有效。

指令和使用环境	描　　述

fastcgi_pass
使用环境：
location，if
	指定什么请求应该转递给 FastCGI 服务器，具体做法是先定义 fastcgi_pass，然后再在 location 中应用。fastcgi_pass 有以下两种语法：
	● 对于 TCP 套接字
	`fastcgi_pass hostname:port;`
	● 对于 Unix 套接字
	`fastcgi_pass unix:/path/to/fastcgi.socket;`
	也可以参考 upstream 块(详情参见后文):
	● fastcgi_pass myblock;
	示例：
	`fastcgi_pass localhost:9000;`
	`fastcgi_pass 127.0.0.1:9000;`
	`fastcgi_pass unix:/tmp/fastcgi.socket;`
	`# 使用 upstream 块`
	`upstream fastcgi {`
	`  server 127.0.0.1:9000;`
	`  server 127.0.0.1:9001;`
	`}`
	`location ~* \.php$ {`
	`  fastcgi_pass fastcgi;`
	`}`

fastcgi_param
使用环境：
http，server，
location
	允许配置传递给 FastCGI 服务器的请求。对于所有 FastCGI 的要求，有两个参数是必须的 SCRIPT_FILENAME 和 QUERY_STRING。
	
	示例：
	
	`fastcgi_param SCRIPT_FILENAME /home/website.com/www$fastcgi_script_name;`
	`fastcgi_param QUERY_STRING $query_string;`
	
	对于 POST 请求，增加的参数为 REQUEST_METHOD、CONTENT_TYPE 和 CONTENT_LENGTH：
	
	`fastcgi_param REQUEST_METHOD $request_method;`
	`fastcgi_param CONTENT_TYPE $content_type;`
	`fastcgi_param CONTENT_LENGTH $content_length;`
	
	fastcgi_params 指定的文件，可以在 Nginx 配置文件的目录中找到，它包含所有必要的参数定义。如果没有 SCRIPT_FILENAME，必须在 Nginx 配置文件中为每一个 FastCGI 指定配置。
	语法：fastcgi_param PARAM value;

fastcgi_pass_header 使用环境：http， server， location	指定传递到 FastCGI 服务器额外的头(header)信息。 语法：fastcgi_pass_header headername; 示例：fastcgi_pass_header Authorization;
fastcgi_hide_header 使用环境：http， server，location	指定 FastCGI 服务器应该隐藏的头信息(即 Nginx 没有转发的头信息)。 语法：fastcgi_hide_header headername; 示例：fastcgi_hide_header X-Forwarded-For;

指令和使用环境	描述

续表

指令和使用环境	描　述

fastcgi_param
使用环境：
http，server，
location
	允许配置传递给 FastCGI 服务器的请求。对于所有 FastCGI 的要求，有两个参数是必须的 SCRIPT_FILENAME 和 QUERY_STRING。
	
	示例：
	
	`fastcgi_param SCRIPT_FILENAME /home/website.com/www$fastcgi_script_name;`
	`fastcgi_param QUERY_STRING $query_string;`
	
	对于 POST 请求，增加的参数为 REQUEST_METHOD、CONTENT_TYPE 和 CONTENT_LENGTH：
	
	`fastcgi_param REQUEST_METHOD $request_method;`
	`fastcgi_param CONTENT_TYPE $content_type;`
	`fastcgi_param CONTENT_LENGTH $content_length;`
	
	fastcgi_params 指定的文件，可以在 Nginx 配置文件的目录中找到，它包含所有必要的参数定义。如果没有 SCRIPT_FILENAME，必须在 Nginx 配置文件中为每一个 FastCGI 指定配置。
	语法：fastcgi_param PARAM value;

fastcgi_pass_header 使用环境：http， server， location	指定传递到 FastCGI 服务器额外的头(header)信息。 语法：fastcgi_pass_header headername; 示例：fastcgi_pass_header Authorization;
fastcgi_hide_header 使用环境：http， server，location	指定 FastCGI 服务器应该隐藏的头信息(即 Nginx 没有转发的头信息)。 语法：fastcgi_hide_header headername; 示例：fastcgi_hide_header X-Forwarded-For;

fastcgi_index 使用环境：http，server，location	FastCGI 服务不支持自动目录索引——如果 URI 请求以/结尾，Nginx 会将 fastcgi_index 指定的值附加在/之后。 语法：fastcgi_index 文件名; 示例：fastcgi_index index.php;
fastcgi_ignore_client_abort 使用环境：http，server，location	该指令允许你定义以下情况：如果客户端放弃它们对服务器端的请求，会发生什么；如果打开(turn on)该指令，Nginx 会忽略放弃的请求并且完成处理请求；如果关闭(turn off)该指令，Nginx 不会忽略放弃的请求,它会中断请求处理并且放弃与 FastCGI 服务器的通信。 语法：on 或 off 默认值：off
fastcgi_intercept_errors 使用环境：http，server，location	定义 Nginx 是否处理网关返回的错误或直接向客户端返回错误页。(注意：错误处理通过 Nginx 的指令 error_page 来完成) 语法：on 或 off 默认值：off
fastcgi_read_timeout 使用环境：http，server，location	定义应用程序 FastCGI 的响应超时时间。如果在这个时间段内 Nginx 没有收到响应，则返回“504 Gateway Timeout”的 HTTP 错误。 语法：数值(s) 默认值：60 秒
fastcgi_connect_timeout 使用环境：http，server，location	定义后端服务器连接超时。这个超时和读取/发送(read/send)超时不同——如果 Nginx 已经连接到后台服务器，fastcgi_connect_timeout 将不再适用。 语法：时间值 (秒) 默认值：60 秒

指令和使用环境	描　述

续表

指令和使用环境	描　　述
fastcgi_index 使用环境：http，server，location	FastCGI 服务不支持自动目录索引——如果 URI 请求以/结尾，Nginx 会将 fastcgi_index 指定的值附加在/之后。 语法：fastcgi_index 文件名; 示例：fastcgi_index index.php;
fastcgi_ignore_client_abort 使用环境：http，server，location	该指令允许你定义以下情况：如果客户端放弃它们对服务器端的请求，会发生什么；如果打开(turn on)该指令，Nginx 会忽略放弃的请求并且完成处理请求；如果关闭(turn off)该指令，Nginx 不会忽略放弃的请求,它会中断请求处理并且放弃与 FastCGI 服务器的通信。 语法：on 或 off 默认值：off
fastcgi_intercept_errors 使用环境：http，server，location	定义 Nginx 是否处理网关返回的错误或直接向客户端返回错误页。(注意：错误处理通过 Nginx 的指令 error_page 来完成) 语法：on 或 off 默认值：off
fastcgi_read_timeout 使用环境：http，server，location	定义应用程序 FastCGI 的响应超时时间。如果在这个时间段内 Nginx 没有收到响应，则返回“504 Gateway Timeout”的 HTTP 错误。 语法：数值(s) 默认值：60 秒
fastcgi_connect_timeout 使用环境：http，server，location	定义后端服务器连接超时。这个超时和读取/发送(read/send)超时不同——如果 Nginx 已经连接到后台服务器，fastcgi_connect_timeout 将不再适用。 语法：时间值 (秒) 默认值：60 秒

续表

指令和使用环境	描　　述
fastcgi_send_time out 使用环境：http，server，location	定义发送数据到后台服务器的超时时间。超时时间没有被应用到整个响应延时，而是在两个写操作之间。 语法：时间值(秒) 默认值：60 秒

fastcgi_split_path_info
使用环境：location
	这个指令对于以下形式的 URL 特别有用：
	`http://website.com/page.php/param1/param2/`
	该指令会根据指定的正则表达式将路径信息拆分开：
	`fastcgi_split_path_info ^(.+\.php)(.*)$;`
	这会影响下面两个变量：
	● $fastcgi_script_name　实际的文件名(脚本文件)将被执行(在本例中是 page.php)
	● $fastcgi_path_info　URL 中脚本名之后的部分(在本例中为/param1/param2/)
	这些参数能够进一步供定义使用：
	`fastcgi_param SCRIPT_FILENAME /home/website.com/www$fastcgi_script_name;`
	`fastcgi_param PATH_INFO $fastcgi_path_info;`
	语法：正则表达式

fastcgi_store 使用环境：http，server，location	提供一个简单的缓存存储(cache store)，用于存储 FastCGI 应用程序的响应，作为文件存储在驱动器上。再有同样的 URI 请求时，文档就直接从缓存中提供，而不用再将请求转发到 FastCGI 应用程序。 该指令用于开启或禁用缓存存储。 语法：on 或 off

fastcgi_store_access 使用环境：http，server，location	该指令定义在使用环境中对缓存文件的访问权限。 语法： fastcgi_store_access [user:r|w|rw] [group:r|w|rw] [all:r|w|rw]; 默认值：fastcgi_store_access user:rw;
fastcgi_temp_path 使用环境：http，server，location	设置临时文件和缓存文件的路径。 语法：文件路径 示例： fastcgi_temp_path /tmp/nginx_fastcgi;
fastcgi_max_temp_file_size 使用环境：http，server，location	指定临时文件的最大值。如果该指令的值设置为 0，那么对于 FastCGI 请求将禁止使用临时文件，或者指定临时文件的最大值。 默认值：1 GB 语法：大小值 示例：fastcgi_max_temp_file_size 5m;
fastcgi_temp_file_write_size 使用环境：http，server，location	提供使用存储驱动器上的临时文件时，需要设置写缓冲区的大小，该指令就是来完成这个工作的。 语法：大小值(Size value) 默认值：2 * proxy_buffer_size
fastcgi_buffers 使用环境：http，server，location	设置缓冲区的数量和大小，用于从 FastCGI 应用程序读取响应数据。[①] 语法：fastcgi_buffers 数量 大小; 默认值：8 缓冲区，每个大小为 4k 或 8k，具体取决于平台 示例：Fastcgi_buffers 8 4k;
fastcgi_buffer_size 使用环境：http，server，location	设置缓冲区的大小，用于读取从 FastCGI 应用程序响应数据包开始的部分，通常包含简单的头数据。默认值相当于 1 个缓冲区大小，如前面指令(fastcgi_buffers)的定义。 语法：大小值(Size value) 示例：fastcgi_buffer_size 4k;

① 该指令有两个参数，第一个设置缓冲区的数量，第二个设置缓冲区的大小。

指令和使用环境	描述

续表

指令和使用环境	描　述
fastcgi_store_access 使用环境：http，server，location	该指令定义在使用环境中对缓存文件的访问权限。 语法： fastcgi_store_access [user:r|w|rw] [group:r|w|rw] [all:r|w|rw]; 默认值：fastcgi_store_access user:rw;
fastcgi_temp_path 使用环境：http，server，location	设置临时文件和缓存文件的路径。 语法：文件路径 示例： fastcgi_temp_path /tmp/nginx_fastcgi;
fastcgi_max_temp_file_size 使用环境：http，server，location	指定临时文件的最大值。如果该指令的值设置为 0，那么对于 FastCGI 请求将禁止使用临时文件，或者指定临时文件的最大值。 默认值：1 GB 语法：大小值 示例：fastcgi_max_temp_file_size 5m;
fastcgi_temp_file_write_size 使用环境：http，server，location	提供使用存储驱动器上的临时文件时，需要设置写缓冲区的大小，该指令就是来完成这个工作的。 语法：大小值(Size value) 默认值：2 * proxy_buffer_size
fastcgi_buffers 使用环境：http，server，location	设置缓冲区的数量和大小，用于从 FastCGI 应用程序读取响应数据。[①] 语法：fastcgi_buffers 数量 大小; 默认值：8 缓冲区，每个大小为 4k 或 8k，具体取决于平台 示例：Fastcgi_buffers 8 4k;
fastcgi_buffer_size 使用环境：http，server，location	设置缓冲区的大小，用于读取从 FastCGI 应用程序响应数据包开始的部分，通常包含简单的头数据。默认值相当于 1 个缓冲区大小，如前面指令(fastcgi_buffers)的定义。 语法：大小值(Size value) 示例：fastcgi_buffer_size 4k;

① 该指令有两个参数，第一个设置缓冲区的数量，第二个设置缓冲区的大小。

fastcgi_send_lowat 使用环境：http，server，location	对于 TCP 套接字，该选项允许你使用 SO_SNDLOWAT 标志，仅在 FreeBSD 下。该值定义了在缓冲区中用于输出操作的最小值。 语法：数值(大小) 默认值：0
fastcgi_pass_request_body fastcgi_pass_request_headers 使用环境：http，server，location	定义是否将请求体分别——特别是请求头(request header)——自传递到后台服务器。 语法：on 或 off; 默认值：on
fastcgi_ignore_headers 使用环境：http，server，location	Nginx 阻止处理从后台服务器响应的四种头信息(之一)： ● X-Accel-Redirect ● X-Accel-Expires ● Expires ● Cache-Control 语法：fastcgi_ignore_headers header1 [header2…];

fastcgi_next_upstream
使用环境：http，server，location
	fastcgi_pass 连接到一个 upstream 区段时，该指令定义了请求被抛弃并重新发送到下一个 upstream 服务区段的情况。
	
	该指令接受下列值中的任意组合：
	● error——通信中或试图与一个服务器通信时发生错误
	● timeout——传输或连接尝试中发生超时；
	● invalid_header——后端服务器返回一个空的或无效的响应；
	● http_500
	● http_502
	● http_503
	● http_504
	● http_404——以防这类 HTTP 错误发生，Nginx 切换到下一个 upstream
	● off——禁止使用下一个 upstream server
	示例：
	`fastcgi_next_upstream error timeout http_504;`
	`fastcgi_next_upstream timeout invalid_header;`

指令和使用环境	描　述

续表

指令和使用环境	描　　述
fastcgi_send_lowat 使用环境：http，server，location	对于 TCP 套接字，该选项允许你使用 SO_SNDLOWAT 标志，仅在 FreeBSD 下。该值定义了在缓冲区中用于输出操作的最小值。 语法：数值(大小) 默认值：0
fastcgi_pass_request_body fastcgi_pass_request_headers 使用环境：http，server，location	定义是否将请求体分别——特别是请求头(request header)——自传递到后台服务器。 语法：on 或 off; 默认值：on
fastcgi_ignore_headers 使用环境：http，server，location	Nginx 阻止处理从后台服务器响应的四种头信息(之一)： ● X-Accel-Redirect ● X-Accel-Expires ● Expires ● Cache-Control 语法：fastcgi_ignore_headers header1 [header2…];
fastcgi_next_upstream 使用环境：http，server，location	fastcgi_pass 连接到一个 upstream 区段时，该指令定义了请求被抛弃并重新发送到下一个 upstream 服务区段的情况。 该指令接受下列值中的任意组合： ● error——通信中或试图与一个服务器通信时发生错误 ● timeout——传输或连接尝试中发生超时； ● invalid_header——后端服务器返回一个空的或无效的响应； ● http_500 ● http_502 ● http_503 ● http_504 ● http_404——以防这类 HTTP 错误发生，Nginx 切换到下一个 upstream ● off——禁止使用下一个 upstream server 示例： fastcgi_next_upstream error timeout http_504; fastcgi_next_upstream timeout invalid_header;

fastcgi_catch_stderr 使用环境：http，server，location	允许你拦截一些发送到 stderr(标准的错误输出流设备)的错误信息并且将其存储在 Nginx 错误日志。 语法：fastcgi_catch_stderr filter; 示例：fastcgi_catch_stderr "PHP Fatal error:"

FastCGI 缓存

一旦正确配置 Nginx 与 FastCGI 应用程序一同工作，你就可以随意利用下面的指令通过设置一个缓存系统来改善或提升服务器的总体性能。

fastcgi_cache 使用环境：http，server，location	定义一个 cache 区域(zone)。对区域(zone)定义标识符是为了进一步在其他指令中使用。 语法：fastcgi_cache zonename; 示例：fastcgi_cache cache1;
fastcgi_cache_key 使用环境：http，server，location	该指令用来定义缓存 key，换句话说，就是为了区分不同的缓存条目。如果一个缓存的 key 设置为$uri，那么所有带有类似$uri 的请求将相当于同样的缓存条目。对于大多数动态网站来说这还不够，还需要在缓存 key 中包含查询字符参数，以便/index.php 和/index.php?page=contact 不会指向同一个缓存条目。 语法：fastcgi_cache_key key; 示例：fastcgi_cache "$scheme$host$request_uri $cookie_user";
fastcgi_cache_methods 使用环境：http，server，location	定义能够缓存的适当 HTTP 方法。GET 和 HEAD 默认包含在内，不可以禁止，例如，你可以开启对 POST 请求的缓存。 语法：fastcgi_cache_methods METHOD; 示例：fastcgi_cache_methods POST;
fastcgi_cache_min_uses 使用环境：http，server，location	定义在一个请求有资格(eligible)缓存之前被击中的最少次数，默认情况下，一个请求的响应一次被击中后就会被缓存(如果下一个请求具有同样缓存 key，那么它将会收到这个被缓存的响应)。 语法：数值(Numeric value) 示例：fastcgi_cache_min_uses 1;

指令和使用环境	描述

指令和使用环境	描　述

续表

指令和使用环境	描　　述
fastcgi_catch_stderr 使用环境：http，server，location	允许你拦截一些发送到 stderr(标准的错误输出流设备)的错误信息并且将其存储在 Nginx 错误日志。 语法：fastcgi_catch_stderr filter; 示例：fastcgi_catch_stderr "PHP Fatal error:"

FastCGI 缓存

一旦正确配置 Nginx 与 FastCGI 应用程序一同工作，你就可以随意利用下面的指令通过设置一个缓存系统来改善或提升服务器的总体性能。

指令和使用环境	描　　述
fastcgi_cache 使用环境：http，server，location	定义一个 cache 区域(zone)。对区域(zone)定义标识符是为了进一步在其他指令中使用。 语法：fastcgi_cache zonename; 示例：fastcgi_cache cache1;
fastcgi_cache_key 使用环境：http，server，location	该指令用来定义缓存 key，换句话说，就是为了区分不同的缓存条目。如果一个缓存的 key 设置为$uri，那么所有带有类似$uri 的请求将相当于同样的缓存条目。对于大多数动态网站来说这还不够，还需要在缓存 key 中包含查询字符参数，以便/index.php 和/index.php?page=contact 不会指向同一个缓存条目。 语法：fastcgi_cache_key key; 示例：fastcgi_cache "$scheme$host$request_uri $cookie_user";
fastcgi_cache_methods 使用环境：http，server，location	定义能够缓存的适当 HTTP 方法。GET 和 HEAD 默认包含在内，不可以禁止，例如，你可以开启对 POST 请求的缓存。 语法：fastcgi_cache_methods METHOD; 示例：fastcgi_cache_methods POST;
fastcgi_cache_min_uses 使用环境：http，server，location	定义在一个请求有资格(eligible)缓存之前被击中的最少次数，默认情况下，一个请求的响应一次被击中后就会被缓存(如果下一个请求具有同样缓存 key，那么它将会收到这个被缓存的响应)。 语法：数值(Numeric value) 示例：fastcgi_cache_min_uses 1;

fastcgi_cache_path
使用环境：http，server，location
	指定用于存储缓存文件的目录，也包括其他参数。
	语法：
	fastcgi_cache_path path [levels=numbers keys_zone=name: size inactive=time max_size=size];
	其他参数如下：
	● levels——指定子目录的深度(通常为 1：2 足够)
	● keys_zone——让你使用先前通过指令 fastcgi_cache 指定的 zone 名称，并且可以指明占用内存的大小
	● inactive——如果缓存的响应在指定时间期限内没有被使用，那么它将从缓存中删除
	● max_size——定义整个缓存的最大值。
	示例：
	fastcgi_cache_path /tmp/nginx_cache levels=1:2 zone=zone1:10m inactive=10m max_size=200M;

fastcgi_cache_use_stale 使用环境：http，server，location	定义 Nginx 在某些情况下(关于网关)是否提供过期的缓存数据。如果使用该指令的 timeout 参数，并且是网关超时，那么 Nginx 将提供缓存数据 语法： fastcgi_cache_use_stale [updating] [error] [timeout] [invalid_header] [http_500]; 示例： fastcgi_cache_use_stale error timeout;
fastcgi_cache_valid 使用环境：http，server，location	该指令允许你对各种不同的响应代码定制不同的缓存时间。与 404 错误代码相关的条目可以缓存 1 分钟，而代码 200 OK 响应设置为 10 分钟或者更多。该指令可以插入多次： fastcgi_cache_valid 404 1m; fastcgi_cache_valid 500 502 504 5m; fastcgi_cache_valid 200 10; 语法：fastcgi_cache_valid code1[code2...]time;

这里是一个完整的 Nginx FastCGI 缓存配置示例，使用了大多数与缓存有关的指令(如前所述的指令)：

```
fastcgi_cache phpcache;
fastcgi_cache_key "$scheme$host$request_uri";
    # $request_uri 包含请求参数(例如 such as /page.php?arg=value)
fastcgi_cache_min_uses 2;
    # 击中 (hit),将由缓存的响应来提供请求
```

指令和使用环境	描述

续表

指令和使用环境	描　　述

fastcgi_cache_path
使用环境：http，server，location
	指定用于存储缓存文件的目录，也包括其他参数。
	语法：
	fastcgi_cache_path path [levels=numbers keys_zone=name: size inactive=time max_size=size];
	其他参数如下：
	● levels——指定子目录的深度(通常为 1∶2 足够)
	● keys_zone——让你使用先前通过指令 fastcgi_cache 指定的 zone 名称，并且可以指明占用内存的大小
	● inactive——如果缓存的响应在指定时间期限内没有被使用，那么它将从缓存中删除
	● max_size——定义整个缓存的最大值。
	示例：
	fastcgi_cache_path /tmp/nginx_cache levels=1:2 zone=zone1:10m inactive=10m max_size=200M;

fastcgi_cache_use_stale 使用环境：http，server，location	定义 Nginx 在某些情况下(关于网关)是否提供过期的缓存数据。如果使用该指令的 timeout 参数，并且是网关超时，那么 Nginx 将提供缓存数据 语法： fastcgi_cache_use_stale [updating] [error] [timeout] [invalid_header] [http_500]; 示例： fastcgi_cache_use_stale error timeout;
fastcgi_cache_valid 使用环境：http，server，location	该指令允许你对各种不同的响应代码定制不同的缓存时间。与 404 错误代码相关的条目可以缓存 1 分钟，而代码 200 OK 响应设置为 10 分钟或者更多。该指令可以插入多次： fastcgi_cache_valid 404 1m; fastcgi_cache_valid 500 502 504 5m; fastcgi_cache_valid 200 10; 语法：fastcgi_cache_valid code1[code2...]time;

这里是一个完整的 Nginx FastCGI 缓存配置示例，使用了大多数与缓存有关的指令(如前所述的指令)：

```
fastcgi_cache phpcache;
fastcgi_cache_key "$scheme$host$request_uri";
    # $request_uri 包含请求参数(例如 such as /page.php?arg=value)
fastcgi_cache_min_uses 2;
    # 击中 (hit),将由缓存的响应来提供请求
```

```
fastcgi_cache_path /tmp/cache levels=1:2 keys_zone=phpcache:10m
inactive=30m max_size=500M;
fastcgi_cache_use_stale updating timeout;
fastcgi_cache_valid 404 1m;
fastcgi_cache_valid 500 502 504 5m;
```

既然这些指令差不多适用于每一个虚拟主机，你可能想把这些指令保存到单独的文件中(fastcgi_cache)，然后再在适当的地方包含(include)：

```
server {
    server_name website.com;
    location ~* \.php$ {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_param SCRIPT_FILENAME
/home/website.com/www$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_script_name;
        include fastcgi_params;
        include fastcgi_cache;
    }
}
```

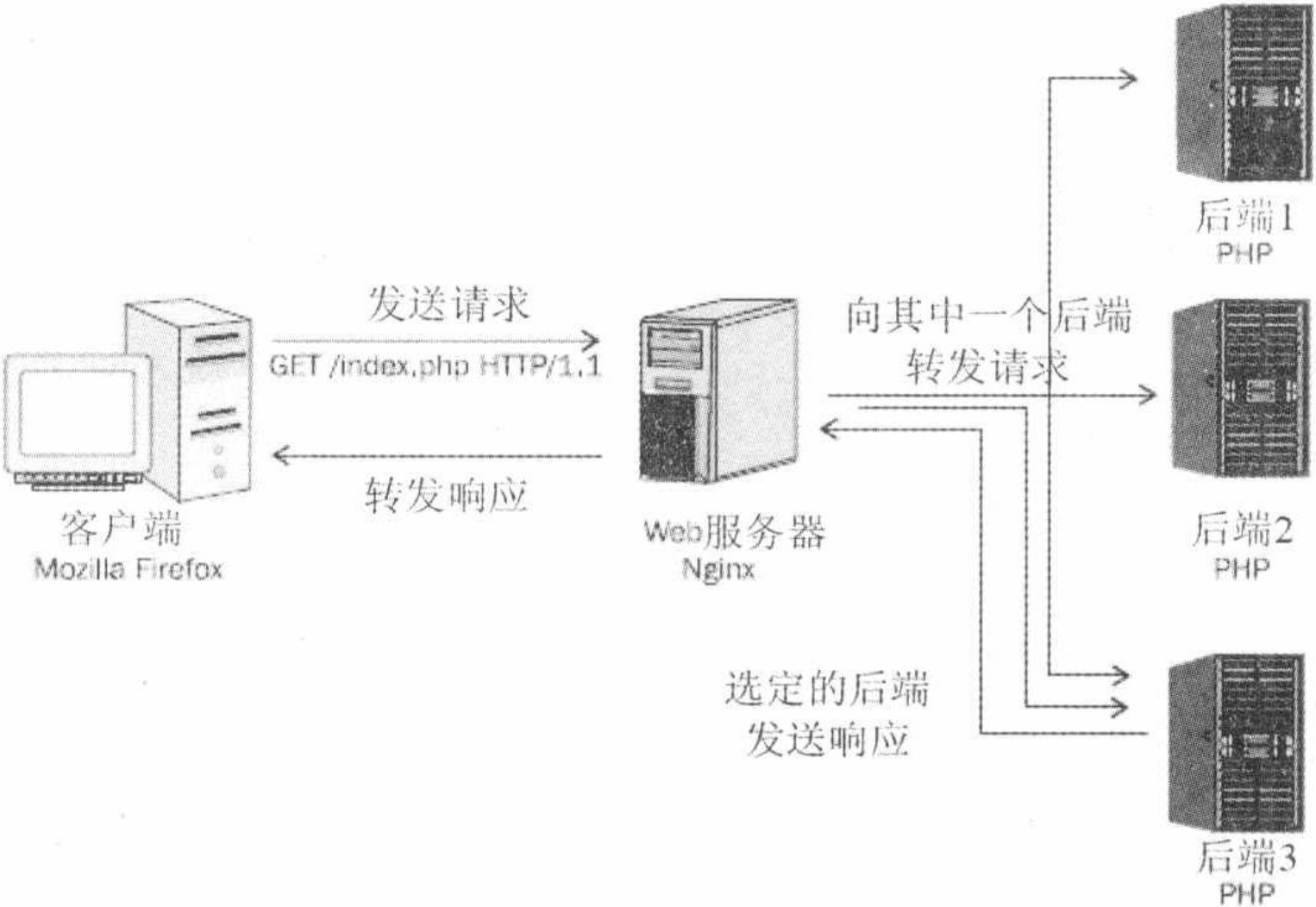
Upstream 块

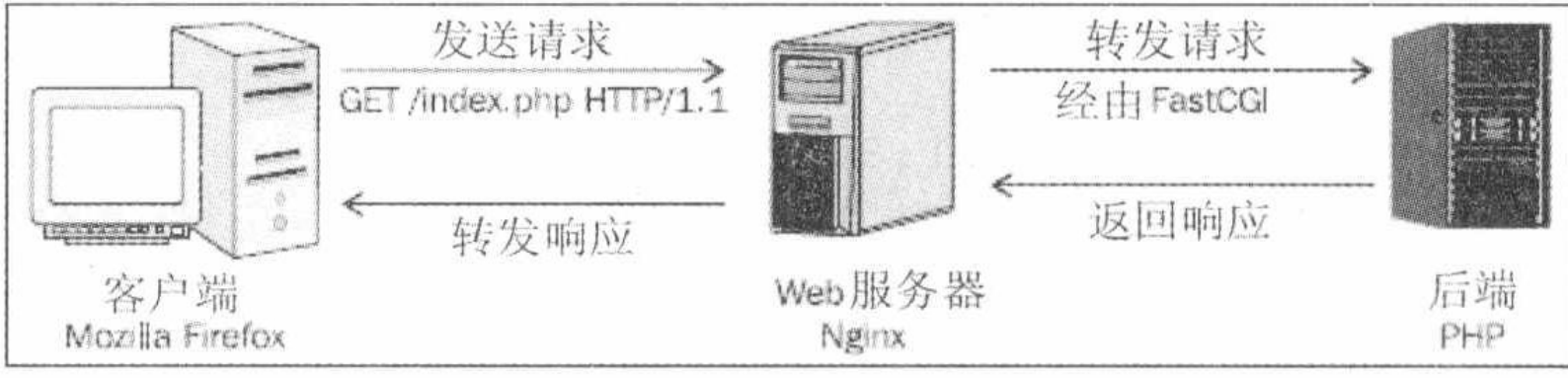
通过使用 FastCGI，也包括你将在下一章认识到的 Proxy 模块，Nginx 将请求转向后端服务器。它与进程通信是通过 FastCGI 或者是简单的类似于规则的 HTTP 客户端行为[1]。在负载平衡的架构情况下，无论哪一种方法，后端服务器(FastCGI 应用程序，其他 Web 服务器，等等)都有可能是位于不同的服务器。

现在，对于应用程序(例如 PHP)来说，一般的问题是它们相当耗资源，尤其是 CPU。基于这种情况，你可能会因此而迫使自己在多个服务器之间找到平衡，结果便有下图所示的结构。

在图中所示的情况中，Nginx 要连接多个后台服务器。要完成这样的配置，需要使用一个新的模块，即 Upstream 模块。

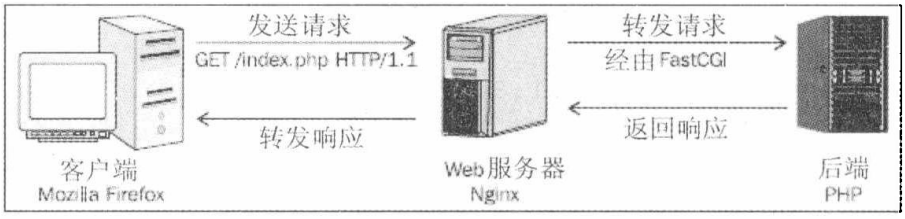
① 换句话说，就是正常的 HTTP 客户端访问。


发送请求
GET /index.php HTTP/1.1
转发响应
客户端
Mozilla Firefox
Web服务器
Nginx
向其中一个后端
转发请求
选定的后端
发送响应
后端1
PHP
后端2
PHP
后端3
PHP

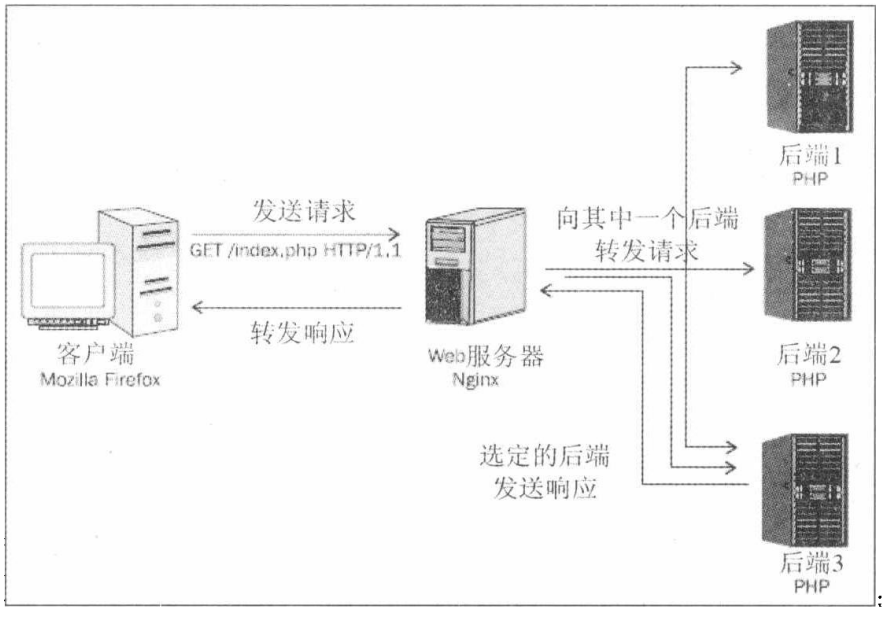

发送请求
GET /index.php HTTP/1.1
转发响应
客户端
Mozilla Firefox
转发请求
经由 FastCGI
返回响应
Web服务器
Nginx
后端
PHP

Upstream 块

通过使用 FastCGI，也包括你将在下一章认识到的 Proxy 模块，Nginx 将请求转向后端服务器。它与进程通信是通过 FastCGI 或者是简单的类似于规则的 HTTP 客户端行为[①]。在负载平衡的架构情况下，无论哪一种方法，后端服务器(FastCGI 应用程序，其他 Web 服务器，等等)都有可能是位于不同的服务器。



现在，对于应用程序(例如 PHP)来说，一般的问题是它们相当耗资源，尤其是 CPU。基于这种情况，你可能会因此而迫使自己在多个服务器之间找到平衡，结果便有下图所示的结构。



在图中所示的情况中，Nginx 要连接多个后台服务器。要完成这样的配置，需要使用一个新的模块，即 Upstream 模块。

① 换句话说，就是正常的 HTTP 客户端访问。

upstream 模块的语法

该模块允许你声明命名 upstream 区段来定义一个服务器列表：

```
upstream phpfpm {
    server 192.168.0.50:9000;
    server 192.168.0.51:9000;
    server 192.168.0.52:9000;
}
```

定义 FastCGI 配置后，就可以连接 upstream 区段：

```
server {
    server_name website.com;
        location ~* \.php$ {
            fastcgi_pass phpfpm;
             [...]
    }
}
```

在前面的设置中，符合条件的 FastCGI 请求将被转发到后端服务器，这个工作由 upstream 来完成，它会在 upstream 区段中选择一个服务器。

有一个问题可能你要问，Nginx 如何决定哪一个后端服务器为每一个请求服务？答案很简单：在 upstream 模块中，默认的方法是轮询，然而，这种方法并不是最好的。这样会出现来自同一个访问者的两个请求可能会被不同的两个服务器处理，由此可能会带来很多问题。例如，一个 PHP 会话存储在一个后台服务器，而会话却不会复制到其他服务器。

要确定同一个访问者发出的请求总是由同一个后端服务器来处理，可以使用 ip_hash 选项，在 upstream 区段中声明即可：

```
upstream phpfpm {
    ip_hash;
    server 192.168.0.50:9000;
    server 192.168.0.51:9000;
    server 192.168.0.52:9000;
}
```

根据前面的设置，这将实现基于访问者 IP 地址的分配请求方法，但它仍然是一种轮询的算法。然而，你很快会意识到，客户端的 IP 地址有时会因为种种原因而更换 IP 地址：动态 IP 刷新、代理切换、Tor(一种翻墙工具)等。因此，ip_hash 机制并不能完全保证客户端总能连接到同一台 upstream 服务器。

server 指令

可以在 upstream 区段的 server 指令中放一些可以接受的其他参数，这些参数影响着 Nginx 服务器对后台服务器的选择。

- weight=n——这里可以设置一个值，代表这个服务器的权重。如果建立一个包含两个后端服务器的 upstream，且设置第一个的权重为 2，那么它被选中的次数是另一个的两倍：

```
upstream php {
    server 192.168.0.1:9000 weight=2;
    server 192.168.0.2:9000;
}
```

如果在 upstream 块中使用 ip_hash 模式，这个 weight 选项会被忽略。

- max_fails=n——定义可以发生通信错误的最大次数(在 fail_timeout 参数定义的时间期限内)。超过这个次数，Nginx 就会使这个服务器不再起作用。
- fail_timeout=n——定义一个时间期限，在这个期限内统计出一个最大的失败次数，如果 Nginx 在这段时间(fail_timeout 设定的时间)内与后端服务器通信失败的次数超过参数 max_fails 设定的次数，则认为这个服务器不再起作用。
- down——如果想把一个后端服务器标记为离线状态，不再使用该服务器，则可以使用该参数。该参数仅适用于指令 ip_hash 开启时。
- backup——如果标记一个后端服务器作为备份服务器，那么 Nginx 将不会使用该服务器，直到其他服务器(没有被标记备份服务器)全部宕机或无效时才启用。

这些参数都是可选参数，它们可以结合使用：

```
upstream phpbackend {
    server localhost:9000 weight=5;
    server 192.168.0.1 max_fails=5 fail_timeout=60s;
    server unix:/tmp/backend backup;
}
```

Nginx + PHP

我们现在打算通过 FastCGI 来配置 PHP + Nginx，在设计这个安装步骤时有一些步骤非常特别，最大的麻烦可能是不能够使用当前建立的 PHP[1]。这个问题将在本章讨论。

结构

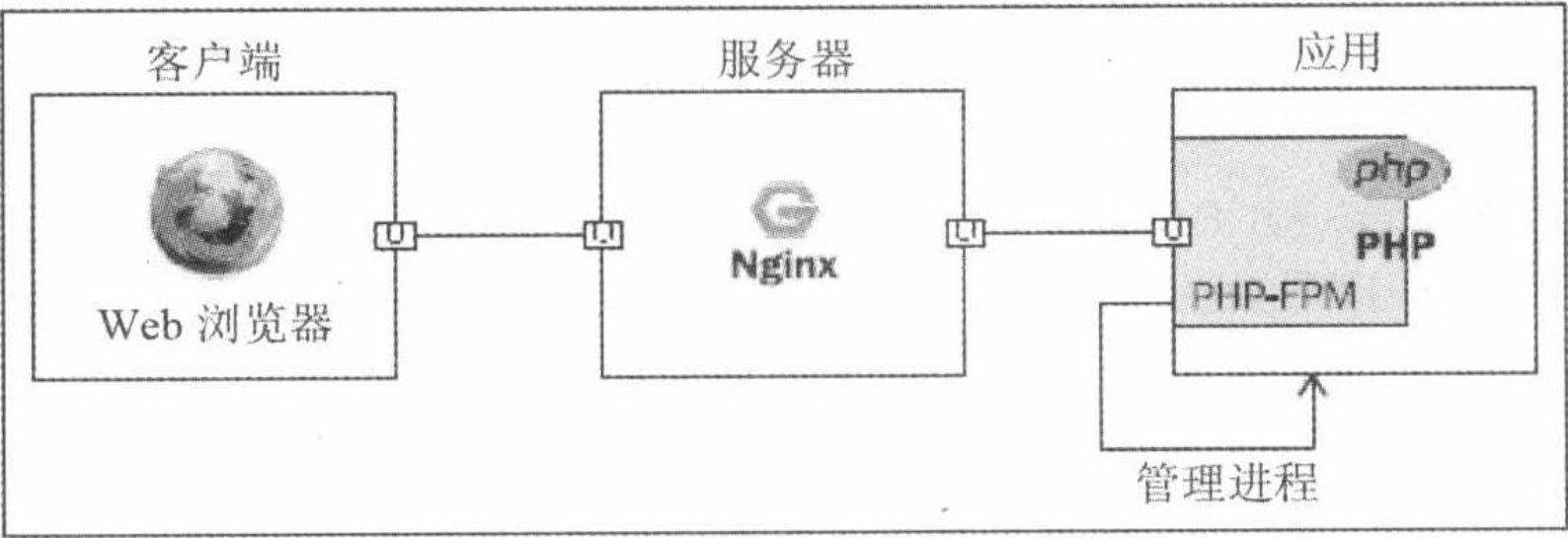
在开始安装过程之前，理解一下 PHP 与 Nginx 是如何交互的。我们已经确定 FastCGI 是一个运行在套接字的通信协议的服务，因此它暗指一个客户端和一个服务器，客户端显然是 Nginx，作为服务器，当然实际上就是正要安装的十分复杂的 PHP。

默认情况下，PHP 支持 FastCGI 协议。PHP 二进制命令能够处理脚本并且能够通过套接字与 Nginx 交互。然而，我们打算使用另外的成分来提高进程的全面管理。

网上提供了几种解决方案，包括实际的 PHP 脚本(打开套接字并且支持 FastCGI 协议)，我们选择的方法是 **PHP-FPM**(PHP FastCGI 进程管理，PHP FastCGI Process Manager)，就功能和性能而言，这是当前公认的效率最高的方法，就是建立可能会有些复杂。

PHP-FPM 将 FastCGI 带到一个全新的水平，它的很多功能将在后文描述。

① 换言之，在安装 PHP 与 Nginx 之前可能你的系统中已经安装了 PHP，并且可能还在使用，那么这个安装会给你以前安装的 PHP 在使用上带来麻烦。


客户端
服务器
应用
Web 浏览器
Nginx
php
PHP
PHP-FPM
管理进程

Nginx + PHP

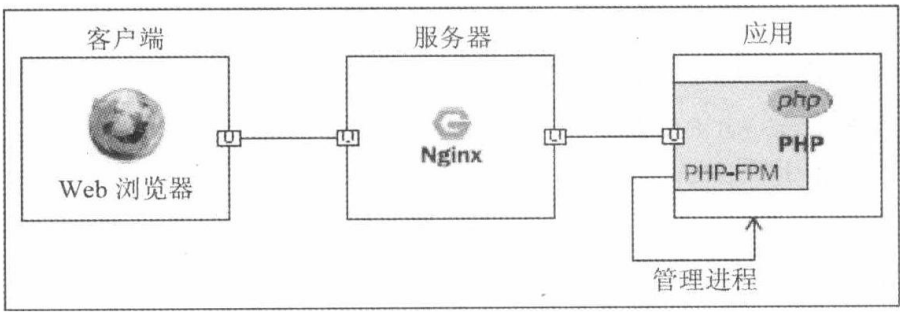
我们现在打算通过 FastCGI 来配置 PHP + Nginx，在设计这个安装步骤时有一些步骤非常特别，最大的麻烦可能是不能够使用当前建立的 PHP[①]。这个问题将在本章讨论。

结构

在开始安装过程之前，理解一下 PHP 与 Nginx 是如何交互的。我们已经确定 FastCGI 是一个运行在套接字的通信协议的服务，因此它暗指一个客户端和一个服务器，客户端显然是 Nginx，作为服务器，当然实际上就是正要安装的十分复杂的 PHP。

默认情况下，PHP 支持 FastCGI 协议。PHP 二进制命令能够处理脚本并且能够通过套接字与 Nginx 交互。然而，我们打算使用另外的成分来提高进程的全面管理。

网上提供了几种解决方案，包括实际的 PHP 脚本(打开套接字并且支持 FastCGI 协议)，我们选择的方法是 **PHP-FPM**(PHP FastCGI 进程管理，PHP FastCGI Process Manager)，就功能和性能而言，这是当前公认的效率最高的方法，就是建立可能会有些复杂。



PHP-FPM 将 FastCGI 带到一个全新的水平，它的很多功能将在后文描述。

① 换言之，在安装 PHP 与 Nginx 之前可能你的系统中已经安装了 PHP，并且可能还在使用，那么这个安装会给你以前安装的 PHP 在使用上带来麻烦。

PHP-FPM

正如前面的示意图，PHP-FPM 实际上本身并不是程序。它当前的形式是作为一个补丁，你需要将它添加到原始的 PHP 源代码中。将它完全集成到 PHP 的优势在于尽量减少内存使用和 CPU 过载，这就是将其作为一个单独应用程序的原因。不方便的是你不得不为它建立一个特别的 PHP，因为你无法利用当前已经安装的 PHP。

PHP-FPM 在 PHP 结构中使用了新的元素。

- 自动地进程化 PHP，使它变成后台进程
- 提供一个用于管理 PHP 进程的命令行脚本，你可以启动/停止/重启动/重新载入正在监听连接的 PHP-CGI 进程。该脚本非常类似于正常的 service 脚本：php-fpm start，php-fpm stop，php-fpm reload 等等
- 还有更多优点，例如，改良的日志记录，IP 地址约束，等等。

设置 PHP 和 PHP-FPM

在这一部分，我们将详细讲述如何下载和编译一个全新的 PHP，更重要的是添加 PHP-FPM 补丁。

下载并解压

需要建立一个全新的 PHP，在本书写作时，从 4.4.7 到 5.3.x，各种 PHP 版本都支持。访问官方网站：*www.php.net*，下载这些版本，然后从 *www.php-fpm.org* 下载适当的 PHP-FPM 补丁版本。

```
[root@website.com ~]# wget http://php.net/get/php-
5.3.0.tar.gz/from/www.php.net/mirror
[root@website.com ~]# wget http://php-fpm.org/downloads/php-5.3.0-
fpm-0.5.12.diff.gz
```

下载完成之后，执行 tar 对 PHP 归档文件进行解压：

```
[root@website.com ~]# tar xzf php-5.3.0.tar.gz
```

打补丁

下载的第二个归档文件是 PHP-FPM 补丁，需要添加到适当的 PHP 版本。现在，下载的是 PHP 5.3.0，所以需要 PHP 5.3.0 的 PHP-FPM。在下列的命令中，通过读取当前目录中的补丁然后再由管道命令“|”、patch 命令来完成为指定目录下的 PHP 打补丁：

```
[root@website.com ~]# gzip -cd php-5.3.0-fpm-0.5.12.diff.gz |
patch -d php-5.3.0 -p1
Upon executing this command, you should see a long list of
patched files:
patching file configure
patching file configure.in
patching file libevent/aclocal.m4
[…]
patching file sapi/cgi/Makefile.frag
[root@website.com ~]#
```

必要条件

对于创建具有 PHP-FPM 的 PHP，有两个主要的必要条件——libevent 和 libxml 开发库。如果系统上尚未安装它们，则需要通过系统数据包管理程序来安装。

例如，基于 Red Hat 系统和其他系统，使用 Yum 管理器来安装：

```
yum install libevent-devel libxml2-devel
```

对于 Ubuntu、Debian 和其他系统，使用 Apt 或 Aptitude 来安装：

```
aptitude install libxml2-dev libevent-dev
```

创建 PHP

安装完成所有的依赖包之后，就可以开始创建 PHP，类似于先前安装其他的应用程序和库，基本需要三个命令：configure、make 和 make install。你会意识到，如果已经将 PHP 安装到系统，那么这会安装一个应用程序的新实例，新安装的不会覆盖旧的，但这需要在安装时使用不同的路径。

在这里，第一步(configure)是决定性的，你需要启用 PHP-FPM 选项，这是为了让 PHP 包含我们需要的功能，configure 有很多选项，通过这些选项你可以包含选择的功能，一些是必须开启的重要功能，例如，数据库交互功能、正则表达式、支持文件压缩、结合 web 服务器，等等。所有可能的 configure 选项可以使用下列命令来列出：

```
[root@website.com php-5.3.0]# ./configure --help
```

被使用的选项越少，那你也就会意识到大量的功能将会被失去，如果你愿意包含其他的元素，那么会需要额外的依赖包，这里没有写相关的文档。在这里只是使用了--enable-fpm 开关选项：

```
[root@website.com php-5.3.0]# ./configure --enable-fpm
```

下一步是建立一个应用程序并且同时安装它：

```
[root@website.com php-5.3.0]# make all install
```

这个过程需要一段时间，具体取决于系统。

安装之后的配置

开始配置新安装的 PHP。例如，拷贝 php.ini 到先前设置的位置，将先前设置的覆盖掉。下一步是配置 PHP-FPM。打开 php-fpm.conf 文件，该文件位于/usr/local/etc/下面。

该文件包含一些重要的指令，后面将重用这些指令：

- 编辑用于 Unix 套接字和进程的用户和组
- PHP-FPM 监听的 IP 地址和端口号
- 提供同时处理请求的总数
- 允许连接到 PHP-FPM 的 IP 地址

运行和控制

对 PHP-FPM 配置文件做适当的修改后，可以通过下面的命令启动：

```
[root@website.com ~]# /usr/sbin/php-fpm start
```

如果一切按计划进行，你会看到如下所示的消息：

```
Starting php_fpm done
```

此外，用于控制进程的命令如下：

```
php-fpm stop;           # 停止 PHP-FPM
php-fpm quit;           # 平滑关闭 PHP-FPM
php-fpm restart;        # 重启动 PHP-FPM
php-fpm reload;         # 重新载入配置文件
php-fpm logrotate;      # 执行日志轮换
```

Nginx 配置

如果已经成功启动 PHP-FPM，就可以调整配置文件，从而通过配置文件将这两部分连接起来。在下面的 server 区段中，是一个简单有效的模板，可以将其作为自己的网站配置的基础：

```
server {
    server_name .website.com;          # 服务器名称，接受 www 访问
    listen 80;                         # 监听端口为 80
    root /home/website/www;            # 文档的 root 目录
    index index.php;                   # 默认的请求文件名：
    index.php
    location ~* \.php$ {               # 对于以.php 结尾的请求
        fastcgi_pass 127.0.0.1:9000;   # 指定监听的地址和端口号
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
                                           #转给 PHP-FPM 的文档路径
        fastcgi_param PATH_INFO $fastcgi_script_name;
                                       #转给 PHP-FPM 的脚本名
        include fastcgi_params;        #包含 FastCGI 其他相关配置
    }
}
```

保存该配置文件后，重新载入配置文件：/usr/local/nginx/sbin/nginx -s reload 或者是 service nginx reload。在网站的 root 目录下建立一个简单的脚本文件，看 PHP 能否正确解释：

```
[root@website.com ~]# echo "<?php phpinfo(); ?>" >/home/
website/www/index.php
```

点击你喜欢的 Web 浏览器并且输入 http://localhost/(或者你网站的 URL)，你可能会看到如下图所示的页面，PHP 服务器的信息页。

注意，如果文件或目录的访问权限配置不当，你可能偶尔会遇到 403 禁止 HTTP 错误。如果出现这种情况，应确定你在 php-fpm.conf 配置文件中正确指定了用户和组以便能使 PHP 读取。

Nginx 与 Python

Python 是一种流行的面向对象编程语言，在许多平台上可见，从基于 Unix 的系统到 Windows，对于 Java 和 Microsoft .NET 平台也有效。如果对 Python 与 Nginx 一同工作感兴趣，说明你已经知道 Python 能够做什么了。我们打算使用 Python 作为服务端的编程语言，借助于 Django 结构。

Django

Django 是一个开源的 Web 开发结构，Python 是看准了它开发 Web 简单容易，它的口号陈词是“完美主义最后期限的 Web 框架”，更多的信息可访问它的官方网站：*www.djangoproject.com*。

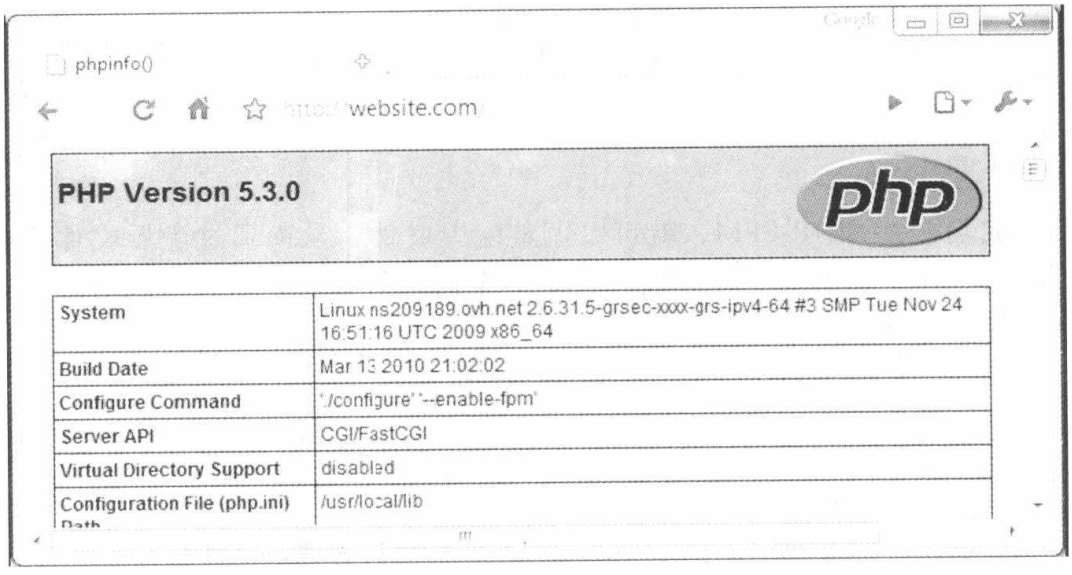
phpinfo()

http://website.com

PHP Version 5.3.0

System	Linux ns209189.ovh.net 2.6.31.5-grsec-xxxx-grs-ipv4-64 #3 SMP Tue Nov 24 16:51:16 UTC 2009 x86_64
Build Date	Mar 13 2010 21:02:02
Configure Command	'./configure' '--enable-fpm'
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/lib

点击你喜欢的 Web 浏览器并且输入 http://localhost/(或者你网站的 URL)，你可能会看到如下图所示的页面，PHP 服务器的信息页。



注意，如果文件或目录的访问权限配置不当，你可能偶尔会遇到 403 禁止 HTTP 错误。如果出现这种情况，应确定你在 php-fpm.conf 配置文件中正确指定了用户和组以便能使 PHP 读取。

Nginx 与 Python

Python 是一种流行的面向对象编程语言，在许多平台上可见，从基于 Unix 的系统到 Windows，对于 Java 和 Microsoft .NET 平台也有效。如果对 Python 与 Nginx 一同工作感兴趣，说明你已经知道 Python 能够做什么了。我们打算使用 Python 作为服务端的编程语言，借助于 Django 结构。

Django

Django 是一个开源的 Web 开发结构，Python 是看准了它开发 Web 简单容易，它的口号陈词是“完美主义最后期限的 Web 框架”，更多的信息可访问它的官方网站：*www.djangoproject.com*。

在其他令人感兴趣的功能中，例如动态管理界面(dynamic administrative interface)、缓存框架(caching framework)和单元测试，Django 提供了 FastCGI 管理，它会使我们通过 Nginx 来运行 Python 脚本更简单。

设置 Python 和 Django

我们将在 Linux 操作系统上安装 Python 和 Django，除了其他的先决条件，过程比较顺利，主要就是要运行两个几乎不会带来任何麻烦的命令。

Python

Python 可以通过安装包管理器来安装。要安装它，可以运行下面的命令。对于基于 Red Hat 和其他的系统，可以使用 Yum 管理工具：

```
yum install python python-devel
```

对于 Ubuntu、Debian 和其他系统，可以使用 Apt 和 Aptitude 工具：

```
aptitude install python python-dev
```

安装包管理器会自己解决它们的依赖包。

Django

为了安装 Django，我们将使用一个不同的方法。为了确定用的是最新版本，我们将直接从 Django SVN 下载一个源代码安装包。

SVN 取自 Subversion 的首字母，是一个代码版本管理软件。它主要的目的是在一个项目开发中维持一个协作工作环境，保护源代码和其他文件的历史版本。通过与 SVN 仓库(repository)连接，你能够下载一个项目指定版本的源代码。

第一步是安装 Subversion，工具允许你与 Django 的存放仓库(repositor)保存同步，对于基于 Red Hat 及其他系统，使用 Yum 安装包管理器：

```
yum install subversion
```

对于 Ubuntu、Debian 和其他系统，可以使用 Apt 和 Aptitude 工具：

```
aptitude install subversion
```

安装包管理器会解决它们自身的依赖包。

一旦 Subversion 安装完毕，就可以下载 django 的源文件。在指定的目录下，下载并安装 Django：

```
[root@website.com ~]# mkdir django && cd django
[root@website.com django]# svn co
http://code.djangoproject.com/svn/django/trunk/
[…]
[root@website.com django]# cd trunk
[root@website.com trunk]# python setup.py install
```

最后，运行 Python FastCGI 管理器最后还需要一个组件 flup 库，它提供了 FastCGI 协议的一个具体实现。对于 Red Hat 及其他系统，使用 Yum 作为安装包管理器(EPEL 仓库必须开启，否则需要从源代码建立)：

```
yum install python-flup
```

对于 Ubuntu、Debian 和其他系统，可以使用 Apt 和 Aptitude 工具：

```
aptitude install python-flup
```

启动 FastCGI 进程管理

这里不会详述如何通过 Django 框架新建网站。一旦完成这部分，你会发现一个名 manage.py 的 Python 脚本，它会给你带来默认的项目模板。将该文件移到指定目录下，运行下面的命令：

```
[root@website.com www]# python manage.py runfcgi method=prefork
host=127.0.0.1 port=9000 pidfile=/var/run/ django.pid
```

如果当前配置的每一项都正确，并且依赖性也正确安装，那么运行该命令后不会产生任何输出。这通常是一个好的信号，FastCGI 进程管理器现在已经运行在后台服务器等待客户端的连接。你可以使用 ps 命令验证这个正在运行的应用程序，例如，执行 ps aux | grep python。现在我们必须做的事情是在 Nginx 配置文件中设置虚拟主机。

Nginx 的配置文件

Nginx 的配置文件类似于 PHP:

```
server {
    server_name .website.com;
    listen 80;
    root /home/website/www;
    index index.html;
    location / {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_script_name;
        include fastcgi_params;
    }
}
```

小结

无论使用 PHP、Python 或其他 CGI 应用程序，你现在都应该有一个清晰的想法，即如何使 Nginx 能够获取并处理你的脚本。对于主流的编程语言和 FastCGI 协议，由于它们具有公认的执行效率，在网上有各种各样的实现版本，现在开始替代服务器集成(server-integrated)方法，例如，Apache 的 mod_php, mod_wsgi，等等。

然而，如果因为你已经有一个功能不错的系统架构(例如 Apache 的 mod_php)，而对这些服务应用程序直接连接 Nginx 不确定(可能缺乏自信心或者没有把握)，那么你可以考虑下一章提供的选择——在现有 Apache 下安装 Nginx，然后让它们协同工作。

第 7 章

Nginx 和 Apache

如果你正在读本书，想必已经知道 Apache 在全球 Web 服务器中占有 55%的份额(截至 2010 年初 Netcraft 的调查)。事实上，许多对 Nginx 感兴趣的管理员是在遭遇 Apache 生产缓慢、复杂的配置、有时无应答、安全问题等之后，才想到换掉 Apache，例如换用 Nginx。然而，有一种往往被忽视的可能—起初听起来有点不太自然(有点牵强)—同时运行 Nginx 和 Apache。了解实际情况后，你会知道该解决方案提供了大量的有利因素(也就是优势)，寻求快速而高效解决方案的管理员尤其能体会到这一点。

本章主题

- 介绍反向代理机制
- 反向代理机制的优势和劣势
- 认识 Nginx 的代理模块
- 配置 Nginx 与 Apache 协同工作
- 重新配置 Apache 作为后端服务器
- 其他调整和注意事项

Nginx 作为反向代理

首先要澄清一点：本章描述的反向代理机制并不是最优解决方法。在下面几种情况下还存在一些问题。

- 已经安装的 Apache 有一个复杂配置文件而且很难移植到 Nginx 或你没有时间或不想完全切换到 Nginx。

- 系统运行着一个前端系统管理面板，例如 Parallels Plesk, cPanel 或其他解决方法，它们会自动产生 Apache 配置文件；
- 项目或机构需要的功能对 Apache 有效但 Nginx 无法提供时。

在其他大多数情况下，完全切换到 Nginx 是有顺序的，第 8 章对此提供了一个很好的描述。

理解问题根源

反向代理主要针对一个问题——Apache 总体提供的速度。由于 Apache 在内存中载入了大量的模块和其他的组成部分(针对每一个收到的 HTTP 请求)，同一时间涌进来大量的请求可能会迅速使服务器陷入混乱。有人说 Apache 是在损失优化和处理速度而致力于功能，在实践中，这种结果导致过多的内存和 CPU 开销，与其相对，Nginx 被证实是轻量级的、稳定的，能够提供大量的请求(与 Apache 相比使用最少的内存和 CPU)。

我们要做什么呢？在回答这个问题之前，分析一下服务器提供的内容有哪些类型。让我们访问一个每天有数百万访问量的网站：www.yahoo.com，我们分析的网站是一个访问量很大的网站，然而它不能够完全代表万维网，Yahoo!主页只是一个可以代表问题的完美例证。

普通用户访问 yahoo.com 的时候，他的网络浏览器实际上不得不下载大量的数据，浏览器下载的不同数据如下表所示。

HTML 源代码	1	157.6KB	52.5KB
Javascript (.js)代码文件和库	6	382.1KB	112.3KB
层叠样式表文件(.css)	3	256.8KB	42.8KB
Flash 动画(.swf)	2	61.4KB	61.4KB
从 CSS 连接的文件(.png 和.gif)	18	43.0KB	43.0KB
普通的图像文件(.gif 和.jpg)	11	73.3KB	73.3KB
总计	41	974.2KB	385.3KB

媒体类型	文件/请求 统计	大小总计	Gzip 压缩后大小总计

- 系统运行着一个前端系统管理面板，例如 Parallels Plesk, cPanel 或其他解决方法，它们会自动产生 Apache 配置文件；
- 项目或机构需要的功能对 Apache 有效但 Nginx 无法提供时。

在其他大多数情况下，完全切换到 Nginx 是有顺序的，第 8 章对此提供了一个很好的描述。

理解问题根源

反向代理主要针对一个问题——Apache 总体提供的速度。由于 Apache 在内存中载入了大量的模块和其他的组成部分(针对每一个收到的 HTTP 请求)，同一时间涌进来大量的请求可能会迅速使服务器陷入混乱。有人说 Apache 是在损失优化和处理速度而致力于功能，在实践中，这种结果导致过多的内存和 CPU 开销，与其相对，Nginx 被证实是轻量级的、稳定的，能够提供大量的请求(与 Apache 相比使用最少的内存和 CPU)。

我们要做什么呢？在回答这个问题之前，分析一下服务器提供的内容有哪些类型。让我们访问一个每天有数百万访问量的网站：www.yahoo.com，我们分析的网站是一个访问量很大的网站，然而它不能够完全代表万维网，Yahoo!主页只是一个可以代表问题的完美例证。

普通用户访问 yahoo.com 的时候，他的网络浏览器实际上不得不下载大量的数据，浏览器下载的不同数据如下表所示。

媒体类型	文件/请求 统计	大小总计	Gzip 压缩后大小总计
HTML 源代码	1	157.6KB	52.5KB
Javascript (.js)代码文件和库	6	382.1KB	112.3KB
层叠样式表文件(.css)	3	256.8KB	42.8KB
Flash 动画(.swf)	2	61.4KB	61.4KB
从 CSS 连接的文件(.png 和.gif)	18	43.0KB	43.0KB
普通的图像文件(.gif 和.jpg)	11	73.3KB	73.3KB
总计	41	974.2KB	385.3KB

这些数据取自 2010 年 3 月 20 日的快照，可能会因地理位置、访问日期和其他标准而导致结果稍有不同。

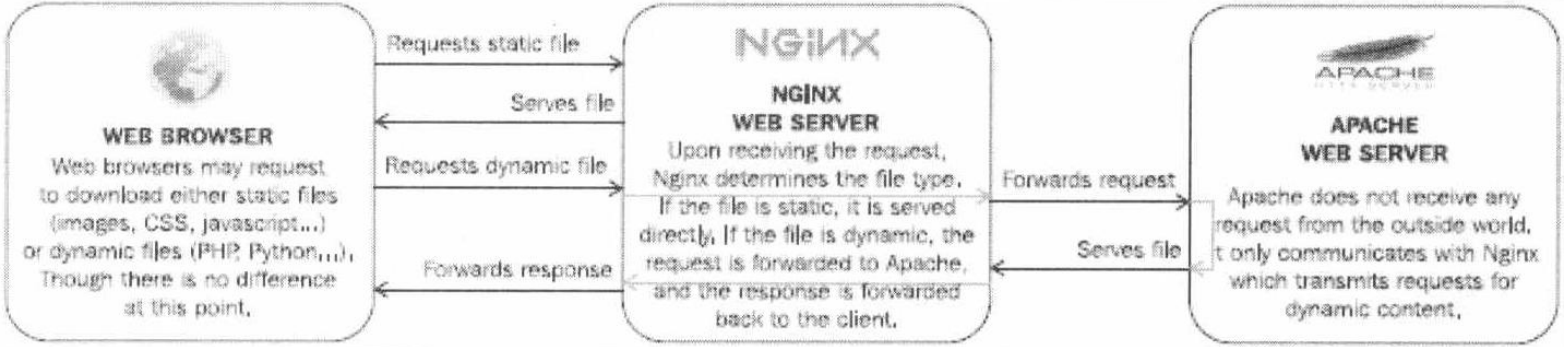
下载数据总量不是很惊人——毕竟也就 385.3KB(包括 cookie 和其他总开销也就 400～450KB)，在许多国家的高速以太网连接中这个数据量不到一秒钟就能传输完。

在这种情况下更大的问题是服务器必须处理的请求数量，对第一次来的访问者，对没有使用任何数据缓存的 Web 浏览器载入该网页，这个 Web 服务器要处理 41 个 HTTP 请求。幸运的是，大部分人都有文件缓存，但是如果你需要更新内容，就不会很好，此外，缓存数据总是要过期的。

你的 Web 服务器能在 1 秒钟内处理 41 个 HTTP 请求吗？能够处理 41 000 (1000 页面查看/秒)吗？能够处理 410 000 吗？如果是这样，你可能有支持这样负载的基础设施。二者择一，使用 Nginx 情况会好些——正如你注意到的，41 个请求中有 40 个是静态元素——图像文件、CSS、JavaScript 代码文件，等等——让 Nginx 提供这些文件以提高速度。我们可以设计一种结构，让 Nginx 提供静态文件而由 Apache 处理动态内容。

反向代理机制

稍微有点像前面的 FastCGI 结构，我们打算将 Nginx 作为前端服务器来运行，换句话说，Nginx 将直接与外面的世界通信，但 Apache 将作为后端服务器来运行，仅与 Nginx 进行数据交换。如下图所示。


WEB BROWSER
Web browsers may request to download either static files (images, CSS, javascript...) or dynamic files (PHP, Python,...). Though there is no difference at this point.
Requests static file
Serves file
Requests dynamic file
Forwards response
NGINX
NGINX WEB SERVER
Upon receiving the request, Nginx determines the file type. If the file is static, it is served directly. If the file is dynamic, the request is forwarded to Apache, and the response is forwarded back to the client.
Forwards request
Serves file
APACHE
APACHE WEB SERVER
Apache does not receive any request from the outside world. It only communicates with Nginx which transmits requests for dynamic content.

这些数据取自 2010 年 3 月 20 日的快照，可能会因地理位置、访问日期和其他标准而导致结果稍有不同。

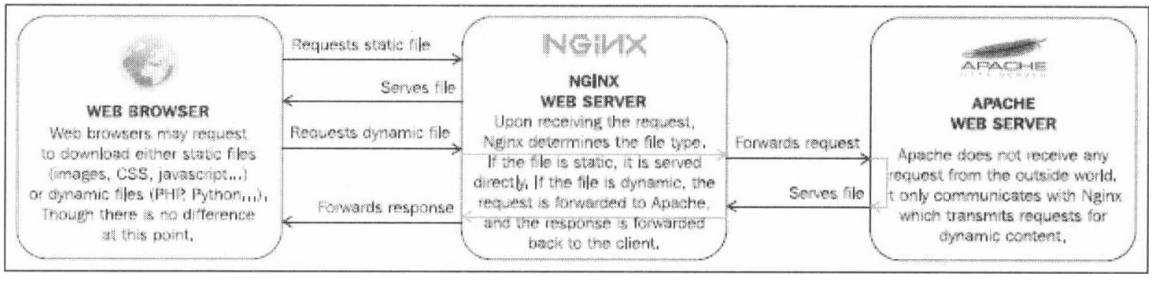
下载数据总量不是很惊人——毕竟也就 385.3KB(包括 cookie 和其他总开销也就 400～450KB)，在许多国家的高速以太网连接中这个数据量不到一秒钟就能传输完。

在这种情况下更大的问题是服务器必须处理的请求数量，对第一次来的访问者，对没有使用任何数据缓存的 Web 浏览器载入该网页，这个 Web 服务器要处理 41 个 HTTP 请求。幸运的是，大部分人都有文件缓存，但是如果你需要更新内容，就不会很好，此外，缓存数据总是要过期的。

你的 Web 服务器能在 1 秒钟内处理 41 个 HTTP 请求吗？能够处理 41 000 (1000 页面查看/秒)吗？能够处理 410 000 吗？如果是这样，你可能有支持这样负载的基础设施。二者择一，使用 Nginx 情况会好些——正如你注意到的，41 个请求中有 40 个是静态元素——图像文件、CSS、JavaScript 代码文件，等等——让 Nginx 提供这些文件以提高速度。我们可以设计一种结构，让 Nginx 提供静态文件而由 Apache 处理动态内容。

反向代理机制

稍微有点像前面的 FastCGI 结构，我们打算将 Nginx 作为前端服务器来运行，换句话说，Nginx 将直接与外面的世界通信，但 Apache 将作为后端服务器来运行，仅与 Nginx 进行数据交换。如下图所示。



这里有两个 Web 服务器运行和处理请求。

- Nginx 放在前端，作为前端服务器(换句话说，作为反向代理)，接收从外面进来的所有请求，它会对请求进行过滤，如果请求的资源是静态文件，则直接由 Nginx 提供给客户端；请求的资源是动态页面，则转发给 Apache。
- Apache 作为后端服务器运行，它只与 Nginx 通信。它可以与前端服务器位于同一台计算机上，在这种情况下，要保证 Nginx 的 80 端口有效，必须对监听端口进行编辑。也可以在不同的机器上使用多个后端服务器(使用 upstream 区段，参考第 6 章)，以便负担负载。

这两个进程的通信并没有使用 FastCGI，Nginx 充当的是一个简单的代理服务器——它从客户端收到 HTTP 请求并且转发到后端服务器(充当一个 HTTP 客户端)。因此，这里没有使用新的协议或复杂的软件，这种机制由 Nginx 提供的代理(Proxy)模块来实现。这种 Nginx 的代理模块处理机制将在后面的章节中讲述。

优势和劣势

将 Nginx 作为前台服务器，Apache 充当后端，主要目标是提高服务速度。正如我们设置，从客户端来的大量请求，请求中的静态文件由速度快的 Nginx 提供，因此从客户端和服务器端来看，总体性能得以显著提高。

在过去，Apache 的安全问题相当多，因而推出新的版本。因此，必须保持系统最新，才能保证有一个完全安全的 Web 服务器。但更公平的说法是，Apache 是一个比较流行的 Web 服务器，因而发现的 bug 和安全问题可能更多。与此相对，Nginx 的最新稳定版本，迄今为止，显然是安全的。Nginx 的作者在对新功能的更新上非常注重安全问题，在少有的新功能更新中别无选择的注重了安全问题。

最后，如果采用这种解决方案，在涉及 Apache 的配置时，你会觉得特别容易设置，几乎不做任何修改。只需要改端口。但如果将 Apache 和 Nginx 安装在不同的服务器上，这个端口也可以不改。如果你已经花几个小时来配置 Apache 在服务器端与预处理程序(例如 PHP, Python 或其他程序)协同工作，那么对于你的这些设置，照现状看，是特别有用的。

在另一方面，仍旧将动态请求交给 Apache，这显然比 Nginx + FastCGI 慢，所以，最优解决方案是完全切换到 Nginx 而省略 Apache。

除此以外，既然 Nginx 作为前端服务器安装，那么它必然包含接受从客户端发来的原始请求(*raw* request)，即 Nginx 接收的 URI 是原始格式，这将导致 Nginx 产生“困惑”(confusion)——它无法区分动态内容和静态内容。对于这个问题，有两个选择——要么将重写规则移植到 Nginx，要么重定向任何以 404 为结果的请求到 Apache 后端服务器。为了解释后者，大多数像这样的/articles/43515-us-economy-strengthens.html 请求不会相当于你系统上的任何文件——这意味着要重写。然后，可以从内部 Nginx 的配置检查是否存在这样的文件，如果文件不存在，就将该请求重定向到 Apache。

最后，同样重要的是，本章后面将进一步讨论，可能会有一些和控制面板(control panel)相关的问题，例如 Parallels Plesk，cPanel 及其他工具。这些面板对管理员很有用，它们能够使一些麻烦的任务自动化进行——例如，在 Apache 中添加虚拟主机，建立电子邮件帐户，配置 DNS 守护进程，等等。有下面两个主要问题。

- 这些控制面板允许根据变化为服务器配置添加改变，它们会自动为服务器产生一个有效的配置文件。不幸的是，迄今为止，这些控制面板只提供兼容 Apache，不会产生 Nginx 的配置文件。因此任何改变对 Nginx 都无效。
- 无论是完全用 Nginx 替代 Apache，还是使用反向代理机制，Nginx 通常最终都运行在 80 端口，控制面板软件不知道这个事实，所以在产生配置文件的时候它会很“固执”(stubborn)，将系统的复位 Apache 的 80 端口，这将与 Nginx 造成冲突。

这两个问题都将在本章后文继续讨论。

Nginx 代理模块

与第 6 章类似，第一步是建立一个新的结构，在这里将认识一个新的模块，Nginx 默认将该代理(Proxy)模块建立在内，通过该模块允许你将客户端的 HTTP 请求转发到后端服务器。我们将从以下几方面配置该模块。

- 后端服务器基本的地址和端口信息
- 缓存、缓冲区和临时文件配置项

- 限制、超时和错误行为配置指令
- 其他配置选项

所有这些选项都是有效的，通过本章介绍的指令进行配置。

主要指令

首先的一组指令允许你建立基本的配置，例如，向 location 中的后端服务器传递的信息，以及如何传递。

指令和使用环境	描　述

proxy_pass
使用环境：location, if
	指定转发到后端服务器的请求，在 location 中指示：
	● 对于 TCP 套接字，语法如下：
	proxy_pass http://hostname:port;
	● 对于 Unix 套接字，语法如下：
	proxy_pass http://unix:/path/to/file.socket;
	也可以指向 upstream 区段：
	proxy_pass http://myblock;
	对于安全流量，还可以使用 https://。另外，URI 部分也可以使用允许的变量。
	示例：
	proxy_pass http://localhost:8080;
	proxy_pass http://127.0.0.1:8080;
	proxy_pass http://unix:/tmp/nginx.sock;
	proxy_pass https://192.168.0.1;
	proxy_pass http://localhost:8080/uri/;
	proxy_pass http://unix:/tmp/nginx.sock:/uri/;
	proxy_pass http://$server_name:8080;
	# 使用 upstream 区段
	upstream backend {
	server 127.0.0.1:8080;
	server 127.0.0.1:8081;
	}
	location ~* \.php$ {
	proxy_pass http://backend;
	}

proxy_method 使用环境：http, server, location	允许最主要的 HTTP 请求方法转发至后端服务器，例如，如果指定 POST，那么转发到后端服务器的所有请求将是 POST 请求。 语法： proxy_method method; 示例： proxy_method POST;
proxy_hide_header 使用环境：http, server, location	默认情况下，当 Nginx 准备从后端服务器转发至客户端响应的时候，会忽略一些头信息：Date, Server, X-Pad 和 X-Accel-*。使用这个指令，可以隐藏更多的客户端头行(header line)。可以插入该指令多次，每次一个头(header)名。 语法： proxy_hide_header header_name; 示例： proxy_hide_header Cache-Control;
proxy_pass_header 使用环境：http, server, location	结合前面的指令，该指令强制一些被忽略的头传递到客户端。 语法： proxy_pass_header headername; 示例： proxy_pass_header Date;
proxy_pass_request_body proxy_pass_request_headers 使用环境：http, server, location	定义是否分别将请求体(request body)和额外的请求头(request header)传递给后端服务器。 语法：on 或 off; 默认值：on

指令和使用环境	描述

续表

指令和使用环境	描　　述
proxy_method 使用环境：http, server, location	允许最主要的 HTTP 请求方法转发至后端服务器，例如，如果指定 POST，那么转发到后端服务器的所有请求将是 POST 请求。 语法： proxy_method method; 示例： proxy_method POST;
proxy_hide_header 使用环境：http, server, location	默认情况下，当 Nginx 准备从后端服务器转发至客户端响应的时候，会忽略一些头信息：Date, Server, X-Pad 和 X-Accel-*。使用这个指令，可以隐藏更多的客户端头行(header line)。可以插入该指令多次，每次一个头(header)名。 语法： proxy_hide_header header_name; 示例： proxy_hide_header Cache-Control;
proxy_pass_header 使用环境：http, server, location	结合前面的指令，该指令强制一些被忽略的头传递到客户端。 语法： proxy_pass_header headername; 示例： proxy_pass_header Date;
proxy_pass_request_body proxy_pass_request_headers 使用环境：http, server, location	定义是否分别将请求体(request body)和额外的请求头(request header)传递给后端服务器。 语法：on 或 off; 默认值：on

proxy_redirect 使用环境：http, server, location	允许改写出现在 HTTP 头却被后端服务器触发重定向的 URL。 语法：off, default, 或选择的 URL ● off——重定向被转到实际的位置； ● default——proxy_pass 指令的值被用作主机名和当前文档的路径附加； 注意，proxy_redirect 必须插在 proxy_pass 指令之后，因为配置文件解析顺序是由上而下的。 ● URL——利用其他 URI 替代 URL 的一部分 另外，在替代的 URL 中可以使用变量。 示例： proxy_redirect off; proxy_redirect default; proxy_redirect http://localhost:8080/ http://example.com/; proxy_redirect http://localhost:8080/wiki/ /w/; proxy_redirect http://localhost:8080/ http://$host/;
proxy_next_upstream 使用环境：http, server, location	proxy_pass 连接到一个 upstream 块的时候，该指令定义了这样一种情况：请求自愿放弃并且重新发送到下一个该区段的下一个 upstream server。该指令接受下列值的组合： ● error——在通信中或试图连接到一个服务器时发生错误； ● timeout——在传输中或试图连接时发生超时； ● invalid_header——后端服务器返回为空或无效的响应 http_500, http_502, http_503,http_504, http_404 在这些情况下发生这样的 HTTP 错误，Nginx 将切换到下一个 upstream ● off: 禁止从上游服务器使用下一个 upstream 服务器 示例： proxy_next_upstream error timeout http_504; proxy_next_upstream timeout invalid_header;

指令和使用环境	描　述

续表

指令和使用环境	描　　述
proxy_redirect 使用环境：http, server, location	允许改写出现在 HTTP 头却被后端服务器触发重定向的 URL。 语法：off, default, 或选择的 URL ● off——重定向被转到实际的位置； ● default——proxy_pass 指令的值被用作主机名和当前文档的路径附加； 注意，proxy_redirect 必须插在 proxy_pass 指令之后，因为配置文件解析顺序是由上而下的。 ● URL——利用其他 URI 替代 URL 的一部分 另外，在替代的 URL 中可以使用变量。 示例： proxy_redirect off; proxy_redirect default; proxy_redirect http://localhost:8080/ http://example.com/; proxy_redirect http://localhost:8080/wiki/ /w/; proxy_redirect http://localhost:8080/ http://$host/;
proxy_next_upstream 使用环境：http, server, location	proxy_pass 连接到一个 upstream 块的时候，该指令定义了这样一种情况：请求自愿放弃并且重新发送到下一个该区段的下一个 upstream server。该指令接受下列值的组合： ● error——在通信中或试图连接到一个服务器时发生错误； ● timeout——在传输中或试图连接时发生超时； ● invalid_header——后端服务器返回为空或无效的响应 http_500, http_502, http_503,http_504, http_404 在这些情况下发生这样的 HTTP 错误，Nginx 将切换到下一个 upstream ● off: 禁止从上游服务器使用下一个 upstream 服务器 示例： proxy_next_upstream error timeout http_504; proxy_next_upstream timeout invalid_header;

缓存、缓冲和临时文件

理想情况下，尽量应该减少转发到后端服务器的请求的数量，下表列出的指令将帮助你建立一个缓存系统，同时还包括缓冲区选项和 Nginx 处理临时文件的方法。

指令和使用环境	描述
proxy_buffer_size 使用环境：http，server，location	设置缓冲区的大小，该缓冲区用于存放读取来自后端服务器响应数据的开始部分，通常包含简单的头(header)数据。 默认值相对于 1 个缓冲区的大小，由前面的指令(proxy_buffers)定义。 语法：数值(大小) 示例： proxy_buffer_size 4k;
proxy_buffering 使用环境：http，server，location	定义是否缓冲后端服务器的响应。如果设置为 on，那么 Nginx 将把响应数据存储在内存中，使用内存空间来提供缓冲。如果缓冲用完，响应数据就存储到临时文件；如果设置为 off，响应就直接转发到客户端。 语法：on 或 off 默认值：on
proxy_buffers 使用环境：http，server，location	设置缓冲数量和大小，用于存放从后端服务器读取的响应数据。 语法：proxy_buffers 数量 大小; 默认值：8 个缓冲，每个缓冲 4k 或 8k，具体取决于平台 示例：fastcgi_buffers 8 4k;
proxy_busy_buffers_size Context：http，server，location	在缓冲区中，收到的后端数据堆积超过该指令指定的数值，缓冲就会被刷新，并且数据被发送至客户端。 语法：数值 (size) 默认值：2 * proxy_buffer_size
proxy_cache 使用环境：http，server，location	定义一个缓存 zone。指定 zone 的标识符，以便在将来的指令中使用。 语法：proxy_cache *zonename*; 示例：proxy_cache cache1;

指令和使用环境	描述

缓存、缓冲和临时文件

理想情况下，尽量应该减少转发到后端服务器的请求的数量，下表列出的指令将帮助你建立一个缓存系统，同时还包括缓冲区选项和 Nginx 处理临时文件的方法。

指令和使用环境	描　述
proxy_buffer_size 使用环境：http，server，location	设置缓冲区的大小，该缓冲区用于存放读取来自后端服务器响应数据的开始部分，通常包含简单的头(header)数据。 默认值相对于 1 个缓冲区的大小，由前面的指令(proxy_buffers)定义。 语法：数值(大小) 示例： proxy_buffer_size 4k;
proxy_buffering 使用环境：http，server，location	定义是否缓冲后端服务器的响应。如果设置为 on，那么 Nginx 将把响应数据存储在内存中，使用内存空间来提供缓冲。如果缓冲用完，响应数据就存储到临时文件；如果设置为 off，响应就直接转发到客户端。 语法：on 或 off 默认值：on
proxy_buffers 使用环境：http，server，location	设置缓冲数量和大小，用于存放从后端服务器读取的响应数据。 语法：proxy_buffers 数量 大小; 默认值：8 个缓冲，每个缓冲 4k 或 8k，具体取决于平台 示例：fastcgi_buffers 8 4k;
proxy_busy_buffers_size Context：http，server，location	在缓冲区中，收到的后端数据堆积超过该指令指定的数值，缓冲就会被刷新，并且数据被发送至客户端。 语法：数值 (size) 默认值：2 * proxy_buffer_size
proxy_cache 使用环境：http，server，location	定义一个缓存 zone。指定 zone 的标识符，以便在将来的指令中使用。 语法：proxy_cache *zonename*; 示例：proxy_cache cache1;

续表

proxy_cache_key 使用环境：http，server，location	该指令定义缓存 key，换句话说，是为了区别缓冲的各项。如果缓存 key 的值设置为$uri，那么所有带有这个$uri 的请求将作为单个缓存条目，换句话说，就是由缓存中的同一个条目来提供访问。但这对于一个动态网站是不够的——还需要在缓存 key 中包括查询字符串参数，以避免/index.php 和/index.php?page=contact 指向同一个缓存条目。 语法：proxy_cache_key key; 示例： proxy_cache_key "$scheme$host$request_uri $cookie_user";
proxy_cache_path 使用环境：http	指出用于存放缓存文件的目录，还有其他参数。 语法：proxy_cache_path path [levels=numbers keys_zone=name:size inactive=time max_size=size]; 其他参数： ● levels——指示子目录的深度(通常 1:2 足够)； ● keys_zone——该参数允许你使用先前通过指令 proxy_cache 声明的 zone，并且指明它在内存中使用的大小； ● inactive——如果一个缓存的响应在指定时间期限内没有被使用，则从缓存中移除该响应； ● max_size——定义整个缓存的最大值 示例： proxy_cache_path/tmp/nginx_cache levels=1:2 zone=zone1:10m inactive=10m max_size=200M;
proxy_cache_methods 使用环境：http，server，location	定义有资格缓存的 HTTP 方法[1]。GET 和 HEAD 默认包含，不能禁用，可以(例如)开启缓存 POST 请求。 语法：proxy_cache_methods METHOD; 示例：proxy_cache_methods POST;

① 换句话说就是能够被缓存的 HTTP 方法。

指令和使用环境	描　述

续表

指令和使用环境	描　　述
proxy_cache_key 使用环境：http，server，location	该指令定义缓存 key，换句话说，是为了区别缓冲的各项。如果缓存 key 的值设置为$uri，那么所有带有这个$uri 的请求将作为单个缓存条目，换句话说，就是由缓存中的同一个条目来提供访问。但这对于一个动态网站是不够的——还需要在缓存 key 中包括查询字符串参数，以避免/index.php 和/index.php?page=contact 指向同一个缓存条目。 语法：proxy_cache_key key; 示例： proxy_cache_key "$scheme$host$request_uri $cookie_user";
proxy_cache_path 使用环境：http	指出用于存放缓存文件的目录，还有其他参数。 语法：proxy_cache_path path [levels=numbers keys_zone=name:size inactive=time max_size=size]; 其他参数： ● levels——指示子目录的深度(通常 1:2 足够)； ● keys_zone——该参数允许你使用先前通过指令 proxy_cache 声明的 zone，并且指明它在内存中使用的大小； ● inactive——如果一个缓存的响应在指定时间期限内没有被使用，则从缓存中移除该响应； ● max_size——定义整个缓存的最大值 示例： proxy_cache_path/tmp/nginx_cache levels=1:2 zone=zone1:10m inactive=10m max_size=200M;
proxy_cache_methods 使用环境：http，server，location	定义有资格缓存的 HTTP 方法[1]。GET 和 HEAD 默认包含，不能禁用，可以(例如)开启缓存 POST 请求。 语法：proxy_cache_methods METHOD; 示例：proxy_cache_methods POST;

① 换句话说就是能够被缓存的 HTTP 方法。

proxy_cache_min_uses 使用环境：http，server，location	定义在一个请求有资格(eligible)缓存之前被击中(hit)的最少次数。默认情况下，一个请求的响应击中一次后就会被缓存(下一个具有同样缓存 key 的请求将收到缓存响应) 语法：数值 示例：proxy_cache_min_uses 1;
proxy_cache_valid 使用环境：http，server，location	该指令允许你对各种不同的响应代码定制不同的缓存时间。你可以缓存 404 错误代码相关条目 1 分钟，相反，代码 200 OK 响应设置为 10 分钟或者更多。该指令可以插入多次。示例： proxy_cache_valid 404 1m; proxy_cache_valid 500 502 504 5m; proxy_cache_valid 200 10; 语法：proxy_cache_valid code1 [code2…] time;
proxy_cache_use_stale 使用环境：http，server，location	定义 Nginx 在某些情况下(关于网关)是否提供过期的缓冲数据。如果使用该指令的 timeout 参数，则在网关超时，Nginx 将提供缓存数据。 语法：proxy_cache_use_stale [updating] [error] [timeout] [invalid_header] [http_500]; 示例：proxy_cache_use_stale error timeout;
proxy_max_temp_file_size 使用环境：http，server，location	如果将该指令的值设置为 0，会禁止使用临时文件；或者可以指定一个缓冲文件的最大值。 语法：大小值 默认值：1 GB 示例：proxy_max_temp_file_size 5m;
proxy_temp_file_write_size 使用环境：http，server，location	设置 Nginx 提供使用存储驱动器上的临时文件这种机制时，可以使用该指令设置写缓冲区的大小。 语法：大小值(Size value) 默认值：2 * proxy_buffer_size
proxy_temp_path 使用环境：http，server，location	设置临时文件和缓存文件的路径。 语法：proxy_temp_path path [level1 [level2…]] 示例： proxy_temp_path /tmp/nginx_proxy; proxy_temp_path /tmp/cache 1 2;

指令和使用环境	描述

续表

指令和使用环境	描　　述
proxy_cache_min_uses 使用环境：http，server，location	定义在一个请求有资格(eligible)缓存之前被击中(hit)的最少次数。默认情况下，一个请求的响应击中一次后就会被缓存(下一个具有同样缓存 key 的请求将收到缓存响应) 语法：数值 示例：proxy_cache_min_uses 1;
proxy_cache_valid 使用环境：http，server，location	该指令允许你对各种不同的响应代码定制不同的缓存时间。你可以缓存 404 错误代码相关条目 1 分钟，相反，代码 200 OK 响应设置为 10 分钟或者更多。该指令可以插入多次。示例： proxy_cache_valid 404 1m; proxy_cache_valid 500 502 504 5m; proxy_cache_valid 200 10; 语法：proxy_cache_valid code1 [code2…] time;
proxy_cache_use_stale 使用环境：http，server，location	定义 Nginx 在某些情况下(关于网关)是否提供过期的缓冲数据。如果使用该指令的 timeout 参数，则在网关超时，Nginx 将提供缓存数据。 语法：proxy_cache_use_stale [updating] [error] [timeout] [invalid_header] [http_500]; 示例：proxy_cache_use_stale error timeout;
proxy_max_temp_file_size 使用环境：http，server，location	如果将该指令的值设置为 0，会禁止使用临时文件；或者可以指定一个缓冲文件的最大值。 语法：大小值 默认值：1 GB 示例：proxy_max_temp_file_size 5m;
proxy_temp_file_write_size 使用环境：http，server，location	设置 Nginx 提供使用存储驱动器上的临时文件这种机制时，可以使用该指令设置写缓冲区的大小。 语法：大小值(Size value) 默认值：2 * proxy_buffer_size
proxy_temp_path 使用环境：http，server，location	设置临时文件和缓存文件的路径。 语法：proxy_temp_path path [level1 [level2…]] 示例： proxy_temp_path /tmp/nginx_proxy; proxy_temp_path /tmp/cache 1 2;

限制、超时设定和错误

下表列出的指令将帮助你定义超时行为，同时也包括与后端服务器通信的各种限制设置。

proxy_connect_timeout 使用环境：http，server，location	定义后端服务器连接超时。这和读取/发送(read/send)超时不一样，如果 Nginx 已经连接到后端服务器，那么该指令是不合适的。 语法：时间值（秒） 示例：proxy_connect_timeout 15;
proxy_read_timeout 使用环境：http，server，location	从后端服务器读取数据超时。这个超时不应用于全部响应延迟，而是两个读操作的响应延时。 语法：时间值（秒） 默认值：60 示例：proxy_read_timeout 60;
proxy_send_timeout 使用环境：http，server，location	发送数据到后端服务器超时。这个超时不应用于全部响应延迟，而是两个写操作的响应延时。 语法：时间值（秒） 默认值：60 示例：proxy_send_timeout 60;
proxy_ignore_client_abort 使用环境：http，server，location	如果设置为 on，即使客户端放弃请求，Nginx 也会继续处理代理请求；在另外一种情况下(也就是设置为 off)，当客户端放弃请求的时候，Nginx 也会放弃它对后端服务器的请求。 默认值：off
proxy_intercept_errors 使用环境：http，server，location	默认情况下，Nginx 返回所有的错误页(HTTP 状态代码 400 及更高)，由后端服务器直接发送到客户端。如果该指令设置为 on，那么错误代码被解析，能够与指令 error_page 指定的值相匹配。 默认值：off
proxy_send_lowat 使用环境：http，server，location	对于 TCP 套接字，该选项允许你使用 SO_SNDLOWAT 标志(仅在 FreeBSD 下)。该值定义缓冲区中用于输出操作的最小值。 语法：数值（大小） 默认值：0

指令和使用环境	描述

限制、超时设定和错误

下表列出的指令将帮助你定义超时行为，同时也包括与后端服务器通信的各种限制设置。

指令和使用环境	描　述
proxy_connect_timeout 使用环境：http，server，location	定义后端服务器连接超时。这和读取/发送(read/send)超时不一样，如果 Nginx 已经连接到后端服务器，那么该指令是不合适的。 语法：时间值（秒） 示例：proxy_connect_timeout 15;
proxy_read_timeout 使用环境：http，server，location	从后端服务器读取数据超时。这个超时不应用于全部响应延迟，而是两个读操作的响应延时。 语法：时间值（秒） 默认值：60 示例：proxy_read_timeout 60;
proxy_send_timeout 使用环境：http，server，location	发送数据到后端服务器超时。这个超时不应用于全部响应延迟，而是两个写操作的响应延时。 语法：时间值（秒） 默认值：60 示例：proxy_send_timeout 60;
proxy_ignore_client_abort 使用环境：http，server，location	如果设置为 on，即使客户端放弃请求，Nginx 也会继续处理代理请求；在另外一种情况下(也就是设置为 off)，当客户端放弃请求的时候，Nginx 也会放弃它对后端服务器的请求。 默认值：off
proxy_intercept_errors 使用环境：http，server，location	默认情况下，Nginx 返回所有的错误页(HTTP 状态代码 400 及更高)，由后端服务器直接发送到客户端。如果该指令设置为 on，那么错误代码被解析，能够与指令 error_page 指定的值相匹配。 默认值：off
proxy_send_lowat 使用环境：http，server，location	对于 TCP 套接字，该选项允许你使用 SO_SNDLOWAT 标志(仅在 FreeBSD 下)。该值定义缓冲区中用于输出操作的最小值。 语法：数值（大小） 默认值：0

其他指令

最后，在 Proxy 模块中最后一组有效的指令没有归类，如下表所示。

proxy_headers_hash_max_size 使用环境：http，server，location	为了加速处理请求，Nginx 使用哈希表来存储代理的头信息，这个指令定义了该表的最大值。如果与后端服务器通讯使用的头(header)大于 512 个头，必须增加该值。 语法：数值（大小） 默认值：512
proxy_headers_hash_bucket_size 使用环境：http，server，location	设置在代理头哈希表(proxy headers hash table)中最长的 header 名字，如果代理头名称超过 64 个字符，必须增加该值。 语法：数值 默认值：64
proxy_ignore_headers 使用环境：http，server，location	阻止 Nginx 处理从后端服务器响应的四种头：X-Accel-Redirect，X-Accel-Expires，Expires 和 Cache-Control。 语法：proxy_ignore_headers header1 [header2…];
proxy_set_body 使用环境：http，server，location	允许你设置一个静态的请求体(static request body)，用于调试目的。变量可以用作指令的值。 语法：字符串值（任何值） 示例：proxy_set_body test;
proxy_set_header 使用环境：http，server，location	该指令允许你重新定义代理 header 值再转到后端服务器。可以声明多次。 语法：proxy_set_header Header Value; 示例：proxy_set_header Host $host;

指令和使用环境	描述

其他指令

最后，在 Proxy 模块中最后一组有效的指令没有归类，如下表所示。

指令和使用环境	描　述
proxy_headers_hash_max_size 使用环境：http，server，location	为了加速处理请求，Nginx 使用哈希表来存储代理的头信息，这个指令定义了该表的最大值。如果与后端服务器通讯使用的头(header)大于 512 个头，必须增加该值。 语法：数值（大小） 默认值：512
proxy_headers_hash_bucket_size 使用环境：http，server，location	设置在代理头哈希表(proxy headers hash table)中最长的 header 名字，如果代理头名称超过 64 个字符，必须增加该值。 语法：数值 默认值：64
proxy_ignore_headers 使用环境：http，server，location	阻止 Nginx 处理从后端服务器响应的四种头：X-Accel-Redirect，X-Accel-Expires，Expires 和 Cache-Control。 语法：proxy_ignore_headers header1 [header2…];
proxy_set_body 使用环境：http，server，location	允许你设置一个静态的请求体(static request body)，用于调试目的。变量可以用作指令的值。 语法：字符串值（任何值） 示例：proxy_set_body test;
proxy_set_header 使用环境：http，server，location	该指令允许你重新定义代理 header 值再转到后端服务器。可以声明多次。 语法：proxy_set_header Header Value; 示例：proxy_set_header Host $host;

proxy_store 使用环境：http，server，location	指定是否将后端服务器的响应存储为一个文件。存储的响应文件能够重新使用，用于为其他的请求提供访问。 可能的值：on，off 或是与文档的根目录或别名(root 或是 alias)相关联的路径。也可以设置为 on，并定义 proxy_temp_path 指令。 示例： proxy_temp_path on; proxy_temp_path /temp/store;
proxy_store_access 使用环境：http，server，location	该指令用于定义文件的访问权限，而所访问的文件是存储了客户端请求访问而产生响应的内容。 语法： proxy_store_access [user:[r|w|rw]] [group:[r|w|rw]] [all:[r|w|rw]]; 示例： proxy_store_access user:rw group:rw all:r;

变量

代理模块提供了几个变量，这些变量可以插入不同的 location。例如，在指令 proxy_set_header 中或基于日志的指令中，例如 log_format。这些可用的变量如下：

- $proxy_host——包含当前请求的后端服务器的名称
- $proxy_port——包含当前请求的后端服务器的端口
- $proxy_add_x_forwarded_for——该变量包含 *X-Forwarded-For* 请求头的值，后跟远程客户端的 IP 地址，这两个值由逗号分隔。如果 X-Forwarded-For 请求头难以获取，那么该变量只包含一个远程客户端的 IP 地址。
- $proxy_internal_body_length——请求体(request body) 的长度或 0。请求体的长度由指令 proxy_set_body 设置。

配置 Apache 和 Nginx

讨论代理模块之后，我们知道这个模块允许我们建立反向代理配置结构，现在该把这些原理转化为实践了。基本配置主要包括两部分。一部分涉及 Apache，另一部分涉及 Nginx。对于你决定应用的这些修改顺序，它们没有任何区别。

指令和使用环境	描述

续表

指令和使用环境	描　述
proxy_store 使用环境：http，server，location	指定是否将后端服务器的响应存储为一个文件。存储的响应文件能够重新使用，用于为其他的请求提供访问。 可能的值：on，off 或是与文档的根目录或别名(root 或是 alias)相关联的路径。也可以设置为 on，并定义 proxy_temp_path 指令。 示例： proxy_temp_path on; proxy_temp_path /temp/store;
proxy_store_access 使用环境：http，server，location	该指令用于定义文件的访问权限，而所访问的文件是存储了客户端请求访问而产生响应的内容。 语法： proxy_store_access [user:[r|w|rw]] [group:[r|w|rw]] [all:[r|w|rw]]; 示例： proxy_store_access user:rw group:rw all:r;

变量

代理模块提供了几个变量，这些变量可以插入不同的 location。例如，在指令 proxy_set_header 中或基于日志的指令中，例如 log_format。这些可用的变量如下：

- $proxy_host——包含当前请求的后端服务器的名称
- $proxy_port——包含当前请求的后端服务器的端口
- $proxy_add_x_forwarded_for——该变量包含 *X-Forwarded-For* 请求头的值，后跟远程客户端的 IP 地址，这两个值由逗号分隔。如果 X-Forwarded-For 请求头难以获取，那么该变量只包含一个远程客户端的 IP 地址。
- $proxy_internal_body_length——请求体(request body) 的长度或 0。请求体的长度由指令 proxy_set_body 设置。

配置 Apache 和 Nginx

讨论代理模块之后，我们知道这个模块允许我们建立反向代理配置结构，现在该把这些原理转化为实践了。基本配置主要包括两部分。一部分涉及 Apache，另一部分涉及 Nginx。对于你决定应用的这些修改顺序，它们没有任何区别。

注意，我们选择以 Apache 为特例描述这个修改过程，但这种方法能够应用于其他任何 HTTP 服务器。只有一点不同，即必须具体编辑配置部分和指令。除此以外，应用反向代理的原理可以使用，不管使用的是什么服务器。

重新配置 Apache

对于将要编辑的 Apache 配置文件，为了允许 Apache 和 Nginx 能够协同工作，那么有两个主要的方面。但首先弄清楚我们从哪里来，以及我们的目标是什么。

配置概述

服务器的结构如下：

- 一个运行在 80 端口的应用服务器，例如 Apache；
- 一个基于服务器端处理动态脚本的应用程序，例如 PHP，通过 CGI，FastCGI 或服务器模块与 Web 服务器进行通讯。

新的配置——我们即将转向的配置，类似于下列设置：

- Nginx 运行于 80 端口；
- Apache 或其他 Web 服务器运行在不同的端口，只接受从本地套接字的请求；
- 脚本处理应用程序保持不变。

可以看出，只更改两个主要的配置，然后应用于 Apache 或其他正在运行的 Web 服务器。首先，改变端口号，为了避免与 Nginx 发生冲突，Nginx 要作为前端服务器运行；其次，(尽管这是一个可选项)，可能想不让 Apache 接受外来的请求，只接受 Nginx 转发的请求。这两个配置步骤将在下文详细介绍。

重新设置端口号

取决于 Web 服务器的建立的方式(手工建立，还是由服务器面板管理软件自动配置，如 cPanel 和 Plesk 等)，你可能会发现有很多配置文件，主要的配置文件经常位于/etc/httpd/conf/ 或 /etc/apache2/，这可能很大程度上依赖于配置结构，一些服务器面板管理软件为每一个虚拟主机建立有额外的文件。

在 Apache 中需要替换下列三个主要的元素：

- Listen 指令，默认设置为 80 端口，必须换用其他端口(例如 8080)。该指令通常在主配置文件中；
- 你必须弄清楚下列目前配置指令(在主配置文件中)：NameVirtualHost A.B.C.D:8080，这里的 A.B.C.D 是在服务器上完成通讯的网卡的 IP 地址；
- 刚才选择的端口需要配置到所有虚拟主机配置部分，描述如下。

虚拟主机部分必须由下面的模板：

```
<VirtualHost A.B.C.D:80>
    ServerName example.com
    ServerAlias www.example.com
    [...]
</VirtualHost>
```

改为：

```
<VirtualHost A.B.C.D:8080>
    ServerName example.com:8080
    ServerAlias www.example.com
    [...]
</VirtualHost>
```

在这个例子中，A.B.C.D 是虚拟主机的 IP 地址，example.com 是虚拟主机的主机名，前两行的端口必须编辑。

只接受本地的请求

有很多方法限定 Apache 只接受本地的请求，拒绝外部的访问。但是首先，为什么要这么做？作为一个放在客户端和 Apache 之间的扩展层，Nginx 从安全方面提供了某种帮助，使访问者不再直接访问 Apache，这就降低了潜在的危险(所有 Web 服务器普遍存在的安全问题)。总的来说，只允许客户端访问你的前端服务器并非一个坏的想法。

一种方法，在主配置文件中改变网卡的监听范围。指令 Listen 在 Apache 中让你指定一个端口，但也可以指定 IP 地址，但在默认情况下，不选择 IP 地址导致它接受所有从网卡来的请求。只要用指令“Listen 127.0.0.1:8080”来代替指令“Listen 8080”， Apache 就只在这个本地的 IP 地址监听了。如果 Nginx 和 Apache 不在同一台主机上，则需要指定具体的 IP 地址，该 IP 地址配置在安装 Apache 的主机上，并且是用于与安装有 Nginx 的主机进行通信的网卡的 IP 地址。

另一种方法作为一种可替代的方法，是建立一个虚拟主机，然后限定这个虚拟主机：

```
<VirtualHost A.B.C.D:8080>
    ServerName example.com:8080
    ServerAlias www.example.com
    […]
    Order deny,allow
    allow from 127.0.0.1
    allow from 192.168.0.1
    deny all
</VirtualHost>
```

这里使用了 Apache 的 allow 和 deny 指令，用来限定访问该虚拟主机的 IP 地址。如此一来，便可以更精细地配置，这对于有些情况尤其有用，例如在网站无法只用 Nginx 提供服务时。

修改完配置后，不要忘记重新载入配置文件，以便应用新的配置，例如 service httpd reload 或/etc/init.d/httpd reload。

配置 Nginx

建立一个能够与 Apache 协同工作的 Nginx，只需几步简单的配置，后文将会更准确地进行调整。

开启代理选项

第一步是在 location 区段中开启接受代理请求。由于指令 proxy_pass 不能够放在 http 或 server 级别，所以要在单独需要转发的地方包含 proxy_pass 指令。通常，一个 location / { 后备区段足够了，除了包含匹配 break 的 location 区段外，它将匹配所有请求。

这里有一个简单的示例，它使用了单个静态的 Apache，和 Nginx 安装在同一台机器上：

```
server {
    server_name.example.com;
    root /home/example.com/www;
     […]
    location / {
        proxy_pass http://127.0.0.1:8080;
    }
}
```

在下面的例子中，我们使用一个 upstream 区段，以便指定多个服务器，这在第 6 章也出现过：

```
upstream apache {
    server 192.168.0.1:80;
    server 192.168.0.2:80;
    server 192.168.0.3:80 weight=2;
    server 192.168.0.4:80 backup;
}
    server {
        server_name .example.com;
        root /home/example.com/www;
        […]
        location / {
            proxy_pass http://apache;
    }
}
```

至此，通过这样的配置文件，所有请求都被代理到后端服务器，我们打算将内容分为两类：

- 动态文件(dynamic file)——在发送到客户端之前需要处理的文件，例如 PHP、Perl 和 Ruby 脚本，由 Apache 提供；
- 静态文件(static file)——其他所有不需要处理的内容，例如图像、CSS 文件、静态的 HTML 文件和媒体，直接由 Nginx 提供。

按照这种方式，我们不得不进行内容分离。通过某种方式使这两种内容能够由这两类服务器分别提供。

内容分离

为了完成这种分离，使用两种不同的 location 块即可：一个用来匹配动态文件扩展(名)；另一个包含所有的其他文件。下面的例子解析了对.php 文件的请求转发至代理：

```
server {
    server_name .example.com;
    root /home/example.com/www;
    […]
    location ~* \.php.$ {
        # 代理所有以.php*结尾的 URI 请求
        # (includes PHP, PHP3, PHP4, PHP5…)
        proxy_pass http://127.0.0.1:8080;
    }
    location / {
        #这里为其他静态内容的选项
        #例如 缓存控制，别名…
        expires 30d;
    }
}
```

这种方法虽然简单，但在使用 URL 重定向时会引起麻烦。很多 Web 2.0 的网站现在使用链接，可能会隐藏文件扩展名，例如 *http://example.com/articles/us-economy-strengthens/*，有的甚至使用链接取代文件扩展名，如 *http://example.com/us-economy-strengthens.html*。

创建反向代理配置时，有两个选项。

- 将 Apache 重写规则(通常在网站根目录下.htaccess 文件中找到)移植到 Nginx，Nginx 需要知道请求文件的实际扩展名，并将该请求正确代理到 Apache。
- 如果不愿意移植 Apache 重写规则，那么对于这样的请求，Nginx 的默认行为是返回 404 错误。然而，可以通过多种方法修改这种行为。例如，通过指令 error_page 来处理 404 错误，或在提供文件之前先测试文件是否存在。

下面是两种解决方法的详细情况。

下面是这个机制的实现，使用的是 error_page 指令：

```
server {
    server_name .example.com;
```

```
    root /home/example.com/www;
    […]
    location / {
        # 这里是提供的静态内容
        expires 30d;
        […]
        # 对于 404 错误，提交查询到@proxy
        #命名 location 区段
        error_page 404 @proxy;
    }
    location @proxy {
        proxy_pass http://127.0.0.1:8080;
    }
}
```

另一种方法是使用 if 指令，if 指令来自 rewrite 模块：

```
server {
    server_name .example.com;
    root /home/example.com/www;
     […]
    location / {
        # 如果被请求的文件扩展名为.php,
        # 那么转交到 Apache 查询
        if ($request_filename ~* \.php.$) {
        break; # 阻止进一步改写(rewrite)
        proxy_pass http://127.0.0.1:8080;
    }
        # 如果被请求的文件不存在
        # 那么转交到 Apache 查询
    if (!-f $request_filename) {
        break; # 阻止进一步改写(rewrite)
        proxy_pass http://127.0.0.1:8080;
    }
    # 这里提供静态文件
    expires 30d;
    }
}
```

两种解决方法没有真正的性能差异，因此它们会传递同样数量的请求到后端服务器。如果想获得最佳性能，应该继续将 Apache 重定向规则移植到 Nginx。

高级配置

眼下，我们使用 proxy 模块提供的指令，它提供了很多功能供我们使用，优化我们的设计。下表列出了一些设置，尽管我们需要逐个核实，但对于反向代理配置十分有用。由于它们可以多次使用，因此可以将它们放在一个单独的配置文件中，然后再包含到 location 区段。

首先建立一个 proxy.conf 文本文件，将该文件放在 Nginx 配置目录，在文件中插入下面的指令。然后在每一个 if 块的 location 区段中(这些区段是将请求转发到后端服务器或者 upstream 区段)proxy_pass 指令后插入以下这一行：

```
include proxy.conf;
```

对于一些设置的建议值：

proxy_redirect off;	设置为 off 后，让 Nginx 以“as it is”方式重定向到客户端，即对响应本身不做任何处理
proxy_set_header Host $host;	正如配置文件中指定的一样，转发到后端服务器的请求中的 Host HTTP 头默认为代理的主机名。这样的设置令 Nginx 可以换而使用客户端请求中的原始主机名
proxy_set_header X-Real-IP $remote_addr;	由于后端服务器从 Nginx 收到请求，所以这时与后端服务器通讯的 IP 将是 Nginx 的 IP 地址而不是原始客户端的 IP 地址。使用这种设置能够转发真实的客户端 IP 地址到一个新的头(header)，即 X-Real-IP 头
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;	类似于前面的 header。只是如果客户端在他/她自己的一端使用了代理，那么客户端实际的 IP 地址应该包含在 X-Forwarded-For 请求头中，使用 $proxy_add_x_forwarded_for 来确保用于套接字通讯的 IP 地址，可能是原始客户端的 IP 地址(代理后面的客户端)转交到后端服务器
client_max_body_size 10m;	限制客户端请求体的最大值为 10M。实际上，这里设置的值只是一个参考值，请确保调整的值与后端服务器的水平相同。否则，一个正确收到请求的(被 Nginx 处理后)将不能够成功的转发到后端服务器

设　置	描　述

高级配置

眼下，我们使用 proxy 模块提供的指令，它提供了很多功能供我们使用，优化我们的设计。下表列出了一些设置，尽管我们需要逐个核实，但对于反向代理配置十分有用。由于它们可以多次使用，因此可以将它们放在一个单独的配置文件中，然后再包含到 location 区段。

首先建立一个 proxy.conf 文本文件，将该文件放在 Nginx 配置目录，在文件中插入下面的指令。然后在每一个 if 块的 location 区段中(这些区段是将请求转发到后端服务器或者 upstream 区段)proxy_pass 指令后插入以下这一行：

```
include proxy.conf;
```

对于一些设置的建议值：

设　置	描　述
proxy_redirect off;	设置为 off 后，让 Nginx 以“as it is”方式重定向到客户端，即对响应本身不做任何处理
proxy_set_header Host $host;	正如配置文件中指定的一样，转发到后端服务器的请求中的 Host HTTP 头默认为代理的主机名。这样的设置令 Nginx 可以换而使用客户端请求中的原始主机名
proxy_set_header X-Real-IP $remote_addr;	由于后端服务器从 Nginx 收到请求，所以这时与后端服务器通讯的 IP 将是 Nginx 的 IP 地址而不是原始客户端的 IP 地址。使用这种设置能够转发真实的客户端 IP 地址到一个新的头(header)，即 X-Real-IP 头
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;	类似于前面的 header。只是如果客户端在他/她自己的一端使用了代理，那么客户端实际的 IP 地址应该包含在 X-Forwarded-For 请求头中，使用 $proxy_add_x_forwarded_for 来确保用于套接字通讯的 IP 地址，可能是原始客户端的 IP 地址(代理后面的客户端)转交到后端服务器
client_max_body_size 10m;	限制客户端请求体的最大值为 10M。实际上，这里设置的值只是一个参考值，请确保调整的值与后端服务器的水平相同。否则，一个正确收到请求的(被 Nginx 处理后)将不能够成功的转发到后端服务器

client_body_buffer_size 128k;	定义用于持有(hold)请求体(request body)内存缓冲的大小，超过该值后，内容就被保存到临时文件中。根据你的访问者期望发送请求的大小来进行调整，类似于 client_max_body_size
proxy_connect_timeout 15;	如果与代理服务器协同工作的后端服务器在本地网络，请务必保持此值尽可能低(这里是 15 秒，但是这个值取决于平均负荷)。不管怎样，此指令的最大值为 75 秒
proxy_send_timeout 15;	确保定义一个用于写操作的超时时间(在一个与后端服务器通讯中两个写操作之间的超时)
proxy_read_timeout 15;	除了读操作外，同前面的指令类似

其他许多指令都可以在这里设定，不过，设置默认值最合适。

其他步骤

如果想完善自己的反向代理架构，另外还有一些你可能感兴趣的步骤。有三个问题要讨论：关于 IP 地址的讨论，如何确保后端服务器收到正确的 IP 地址；对于这样的设置，如何处理 HTTPS 请求；关于服务器面板控制软件的几点说明，cPanel, Plesk，等等。

转发正确的 IP 地址

现在，由于各种各样的原因，有相当一部分网站利用访问者的 IP 地址来做下面的事情：

- 存储访问者在博客或论坛上发表评论的 IP；
- 基于地理目标的广告或其他服务；
- 对指定范围内的 IP 进行服务限制。

对于这些网站，确保其 Web 服务能正确收到访问者的 IP 很重要。

设置	描述

续表

设　置	描　述
client_body_buffer_size 128k;	定义用于持有(hold)请求体(request body)内存缓冲的大小，超过该值后，内容就被保存到临时文件中。根据你的访问者期望发送请求的大小来进行调整，类似于 client_max_body_size
proxy_connect_timeout 15;	如果与代理服务器协同工作的后端服务器在本地网络，请务必保持此值尽可能低(这里是 15 秒，但是这个值取决于平均负荷)。不管怎样，此指令的最大值为 75 秒
proxy_send_timeout 15;	确保定义一个用于写操作的超时时间(在一个与后端服务器通讯中两个写操作之间的超时)
proxy_read_timeout 15;	除了读操作外，同前面的指令类似

其他许多指令都可以在这里设定，不过，设置默认值最合适。

其他步骤

如果想完善自己的反向代理架构，另外还有一些你可能感兴趣的步骤。有三个问题要讨论：关于 IP 地址的讨论，如何确保后端服务器收到正确的 IP 地址；对于这样的设置，如何处理 HTTPS 请求；关于服务器面板控制软件的几点说明，cPanel, Plesk，等等。

转发正确的 IP 地址

现在，由于各种各样的原因，有相当一部分网站利用访问者的 IP 地址来做下面的事情：

- 存储访问者在博客或论坛上发表评论的 IP；
- 基于地理目标的广告或其他服务；
- 对指定范围内的 IP 进行服务限制。

对于这些网站，确保其 Web 服务能正确收到访问者的 IP 很重要。

如前所述，由于 Apache 或更普遍的说法是后端服务器，后端服务器使用该 IP 地址作为套接字进行通信，将在设计中出现的该 IP 地址是 Nginx 服务器所在主机的 IP。对于这个问题，我们已经讨论了解决方案——为了在 X-Real-IP 头中转发客户端的 IP，在配置中插入“proxy_set_header X-Real-IP $remote_addr”指令。

很不幸，这些 Web 应用软件不能用 X-Real-IP 头来配置。远程客户端的 IP 地址会以某种方式被换为其他值，在涉及 Apache 的时候，使用模块 mod_rpaf，它的详细安装和配置不在这里讨论，更多信息可访问其官方网站：*http://stderr.net/apache/rpaf/*。

SSL 问题及解决方案

如果网站打算提供安全网页，必须以某种方式允许访问者通过 443 端口上的 SSL 连接到你的基础设施(infrastructure)。对此，可能有两个解决方案——要么根本不用 Nginx 而保持 Apache SSL 配置不变改，要么配置 Nginx 在 443 端口接受 SSL 通信。

第一种解决方案明显简单——不改虚拟主机配置(不在 Apache 配置文件中改变虚拟主机的端口)，网站完全向外开放(外网的客户端可以访问)，除非后端服务器基于本地网络的其他机器上。

另一种方案是配置 Nginx 接受安全连接是通过 SSL 模块，详情参见第 5 章。一旦在 server 区段正确配置，就能够建立一个配置——把安全的请求转发到后端的 Apache 服务器。注意，如果后端服务器与前端的 Nginx 基于同一台机器，必须编辑配置文件，以避免前端和后端端口冲突。

服务器面板控制问题

许多服务器管理员依赖于面板控制软件，因为这样能简化他们的工作。高端的软件解决方案(例如 Parallels Plesk 或 cPanel)，能够为许多应用服务器(网站服务、电子邮件、数据库等)产生配置文件。遗憾的是，它们作为一个特有的 Web 服务器应用程序，都只支持 Apache，目前还不支持 Nginx。

如果你按照配置反向代理进程的步骤，会注意到有时必须手动编辑 Apache 的配置文件。我们替换监听端口和编辑或插入一些配置指令。显然，使用面板控制软件产生的文件时，我们手工修改这些控制，软件是不知道的，因此会擦掉(erase)我们的修改。重新启动 Apache 的时候，迎接你的将是冲突和错误信息。

在这一点上，每一次重新建立配置文件后，再次应用改变的时候，没有其他的解决方法。随着 Nginx 的日益普及，开发者在他们的软件将有望推行全面 Nginx 的支持，或者至少允许编辑它们的配置文件，根据作为反向代理需要设置 Nginx。

小结

Nginx 的反向代理配置为我们的架构带来许多优势，而且还可以配置。然而，在构建过程中你会遇到一些障碍，尤其是通过面板控制软件管理服务器时。此外，由于没有对所有请求使用反向代理，你将无法充分利用 Nginx。

如果正在寻求一个更有效的解决方案，可以考虑完全取代 Apache 的 Nginx。下一章将详细说明这一过程，按部就班，从虚拟主机到重写规则再到 FastCGI。

第 8 章

从 Apache 到 Nginx

每一个有经验的系统管理员都会告诉你同样的故事——Web 基础设施工作良好，客户端能够在一个理想的速度下进行访问，你最不想做的一件事情就是修改原有的架构，它会花掉你几天、几个星期甚至几个月的时间。(这是为什么呢？)网站人气逐渐旺起来的时候，问题往往会不可避免地发生(上述问题不是主流问题，因此不能作为证据)，无论怎样修改已有的配置文件，都收效甚微，最终你不得不寻求一种解决方法。这时候，你有多个原因想抛弃原有的 Web 服务器而全面采用 Nginx 来提供服务。无论你决定把 Nginx 作为唯一一台更高效的服务器，还是作为一个代理服务器，或就只是一个服务器，因为你可能想一次性去掉 Apache。本章将指导你用后者(Nginx)取代前者(Apache)。

本章主题

- 深入比较 Apache 和 Nginx
- 一个移植 Apache 配置的完整指南
- 如何把 Apache 的重写规则(rewrite rule)移植到 Nginx
- 一些流行 Web 应用程序的重写规则

Nginx 对 Apache

本章将回答一些关于 Nginx 的主要问题——它是如何从其他服务器中脱颖而出的？与 Apache 比较如何？你是否在使用 Apache 之前用过 Nginx 或考虑用它来替代当前的 Web 服务器？你为什么决定使用 Nginx，而不使用几乎占有全球互联网网站一半份额的 Apache？

特征

通过第 8 章详细说明反向代理的配置，这个特征存在与否并没有什么了不起，鉴于 Nginx 也只会区分静态和动态内容，结果它只提供静态内容的请求，而将动态内容转发至后端的服务器。不管怎样，开始考虑用 Nginx 来完全替代现有服务器的时候，弄清楚它有哪些能耐。如果你计划的架构需要特殊组成部分，那么首先你通常要做的是检查应用程序的功能，下面列出一些主要的功能，描述与 Apache 相比 Nginx 是如何工作的。

内核和功能

请求管理 Web 服务如何处理请求？	事件驱动结构 使用异步套接字接受请求，不使用单独的线程处理，旨在减少内存和 CPU 开销	同步套接字、线程和进程 每一个请求是一个单独的进程或线程，使用同步套接字
设计语言 Web 服务使用何种语言编写？	C C 语言是显而易见的低级语言，它提供了更多的内存管理	C 和 C++ 尽管 Apache 是用 C 语言编写的，但是许多模块使用的是 C++
可移植性 支持何种操作系统？	多平台 Nginx 能够运行在 Windows、GNU/Linux、Unix、BSD、Mac OS X 和 Solaris	多平台 Apache 能够运行在 Windows、GNU/Linux、Unix、BSD、Mac OS X、Solaris、Novell NetWare、OS/2、TPF、OpenVMS、eCS、AIX、z/OS、HP-UX 等
诞生年月 从开发到现在有多久了？	2002 Nginx 比 Apache 年轻，它是为更现代的时代而准备的	1994 Apache 是许多开源项目中的一个，开始于 20 世纪 90 年代，它是造就当今万维网的元老

特　征	Nginx	Apache

特征

通过第 8 章详细说明反向代理的配置，这个特征存在与否并没有什么了不起，鉴于 Nginx 也只会区分静态和动态内容，结果它只提供静态内容的请求，而将动态内容转发至后端的服务器。不管怎样，开始考虑用 Nginx 来完全替代现有服务器的时候，弄清楚它有哪些能耐。如果你计划的架构需要特殊组成部分，那么首先你通常要做的是检查应用程序的功能，下面列出一些主要的功能，描述与 Apache 相比 Nginx 是如何工作的。

内核和功能

特　征	Nginx	Apache
请求管理 Web 服务如何处理请求？	事件驱动结构 使用异步套接字接受请求，不使用单独的线程处理，旨在减少内存和 CPU 开销	同步套接字、线程和进程 每一个请求是一个单独的进程或线程，使用同步套接字
设计语言 Web 服务使用何种语言编写？	C C 语言是显而易见的低级语言，它提供了更多的内存管理	C 和 C++ 尽管 Apache 是用 C 语言编写的，但是许多模块使用的是 C++
可移植性 支持何种操作系统？	多平台 Nginx 能够运行在 Windows、GNU/Linux、Unix、BSD、Mac OS X 和 Solaris	多平台 Apache 能够运行在 Windows、GNU/Linux、Unix、BSD、Mac OS X、Solaris、Novell NetWare、OS/2、TPF、OpenVMS、eCS、AIX、z/OS、HP-UX 等
诞生年月 从开发到现在有多久了？	2002 Nginx 比 Apache 年轻，它是为更现代的时代而准备的	1994 Apache 是许多开源项目中的一个，开始于 20 世纪 90 年代，它是造就当今万维网的元老

一般的功能

这一部分主要对比 Apache 和 Nginx 之间的不同，而不是列出各种功能，这个在前面已经讲过了。

HTTPS 支持 Web 服务器可以提供安全的网页吗？	作为模块支持 如果想支持 HTTPS，需要确保将正确的模块编译到 Nginx	作为模块支持 Apache 在默认情况下通过 include 来支持 HTTPS 模块
虚拟主机 在 Web 服务器主机上可以在同一台计算机上支持多个网站？	生来就支持 Nginx 生来就支持虚拟主机，但是默认的配置不支持每一台虚拟主机一个配置文件(更多细节本章会进一步说明)	生来就支持 Apache 生来就支持虚拟主机，每一个目录包含一个配置文件(.htaccess)
CGI 支持 Web 浏览支持 CGI 和 FastCGI 吗？	仅支持 FastCGI Nginx 通过一个模块支持 FastCGI，在编译时会默认包含在内	支持 CGI 和 FastCGI 通过在 Apache 中载入模块，两种协议都可使用。
系统模块 Web 服务器如何处理模块？	静态模块系统 模块必须在编译时包含在内	动态模块系统 模块可以在配置文件中动态的载入和卸载

一般而言，Apache 的模块数更多，显而易见，它能够提供更多功能。它的很多功能，甚至是核心应用程序的内核都是模块化的。此时，Apache 官方网站的模块涉及 500 多个，适用于各种版本，相对而言，是比 Nginx 的 80 个模块多出很多。造成这种现实状况的主要原因如下。

功　　能	Nginx	Apache

一般的功能

这一部分主要对比 Apache 和 Nginx 之间的不同，而不是列出各种功能，这个在前面已经讲过了。

功　能	Nginx	Apache
HTTPS 支持 Web 服务器可以提供安全的网页吗？	作为模块支持 如果想支持 HTTPS，需要确保将正确的模块编译到 Nginx	作为模块支持 Apache 在默认情况下通过 include 来支持 HTTPS 模块
虚拟主机 在 Web 服务器主机上可以在同一台计算机上支持多个网站？	生来就支持 Nginx 生来就支持虚拟主机，但是默认的配置不支持每一台虚拟主机一个配置文件(更多细节本章会进一步说明)	生来就支持 Apache 生来就支持虚拟主机，每一个目录包含一个配置文件(.htaccess)
CGI 支持 Web 浏览支持 CGI 和 FastCGI 吗？	仅支持 FastCGI Nginx 通过一个模块支持 FastCGI，在编译时会默认包含在内	支持 CGI 和 FastCGI 通过在 Apache 中载入模块，两种协议都可使用。
系统模块 Web 服务器如何处理模块？	静态模块系统 模块必须在编译时包含在内	动态模块系统 模块可以在配置文件中动态的载入和卸载

一般而言，Apache 的模块数更多，显而易见，它能够提供更多功能。它的很多功能，甚至是核心应用程序的内核都是模块化的。此时，Apache 官方网站的模块涉及 500 多个，适用于各种版本，相对而言，是比 Nginx 的 80 个模块多出很多。造成这种现实状况的主要原因如下。

灵活性和团队

在两个应用程序的家庭中(family)做一个诚实的比较，这是另一个批判标准。在今天的计算机科学界，不能够简单的关注应用程序提供的原始功能，也要考虑其他的问题。

例如：

- 如果遇到问题，我去哪里寻求帮助？
- 我是否打算查找功能相关文档？
- 将来会有更多模块？
- 该项目是否仍然活跃并有开发人员进行更新？
- 是否有很多管理员对服务器的安全进行过测试？

如果服务器广泛流行，这些问题一般不言自明。就 Apache 来说，说它是一个主流的应用程序这还不够，它的文档很容易找到，开发人员这些年发布了上百个模块，而且十五年以来它都定期升级。

Nginx 怎么样？它对这些问题的立场如何？这绝对是一个敏感的问题。首先，有一些固定集中信息的网站，例如官方的 wiki。如果你有一个关于 Apache 的问题，一个简单的搜索就足以找到很多准确解答问题的文章；然而，如果你有一个关于 Nginx 的问题，可能不得不求助于网上论坛的新闻组或邮件列表。

在更新和安全方面，虽然 Nginx 由其作者 Igor Sysoev 经常升级，但这些升级很少包括安全修补，服务器从一开始就已经建立在坚实可靠的基础之上。尽管它的使用不如 Apache 普及，但 Nginx 用于一些非常流行的网上平台，例如 SourceForge，WordPress，ImageShack，等等。对 Web 服务高性能领域的这些贡献，赋予它不可否认的正确性。

性能

一般而言，性能和基于团体问题是重要的，能够改变一切的方面是性能，管理员天生倾向于喜欢能够提供给最终用户最佳舒适的服务器，判断标准的特征是最小时间的网页载入时间和最大 RPM(每秒的请求率)率。

第 3 章提供了第一个测试 HTTP 服务性能的方法。为了建立直接的性能比较，同样的测试也可以应用于 Apache。事实上，许多管理员和技术人员的博客中已经这样做了，总的趋势是在各个方面都毫无疑问地赞成 Nginx。

- Nginx 的 RPS 率一般比较高，有时是 Apache 的两倍，换句话说，Nginx 在同样的时间间隔能够提供的网页数是 Apache 提供的网页数的两倍。
- Nginx 响应时间较低(lower)——请求数增长时，Apache 提供网页会变得越来越慢(slower)。
- 对于提供同样的请求时，Apache 使用的带宽略多于 Nginx。这有两方面的原因：不是 Apache 产生更多流量开销，就是通过更多占用有效带宽来快速传递数据(仍不能确定这些假设哪个最有效)。

综上所述，在性能这一方面，Nginx 毫不费劲地赢了 Apache。这足以让很多服务器改用这个轻量级的俄罗斯 Web 服务器。

使用

Nginx 以出色性能遥遥领先于 Apache 的原因，恰好是编写 Nginx 的动机。起初，Igor Sysoev 创建的 Nginx 就是为俄罗斯 Web 站(*www.rambler.ru*)提供流量非常高的访问，每天会收到上亿的请求数量。Apache 的设计者们在 20 世纪 90 年代着手设计 Apache 的时候，可能不是 Apache 设计师的原计划中一部分。

更通俗地说，Nginx 的设计目的是解决 C10k 问题。这个问题指明了一个共同的评论点，根据当前的计算机技术状态和网络扩展性仅允许一台计算机(从主流产业)续连接 10 000(同时并发的网络连接)，因为操作系统和软件的限制。然而，现在该值不再有代表性，由于技术进步，其时，这个问题被认为是非常严重的，因此引发主流 Web 服务器的开发，例如 Lighttpd, Cherokee，显然还有 Nginx。

结论

Nginx 社区有一个名言，对这种情况的概括相当准确：

“Apache 就像 Microsoft 的 Word，它有一百万个选项，但你只需要六个。Nginx 只做了这六件事，但是它做的这六件事中有五件事比 Apache 做的快 50 倍。”

——Chris Lea, ***ChrisLea.com***

另一个帮助 Nginx 建立声誉的显著证明是：

“我目前使用 Nginx 做反向代理，在一个服务器上每天超过千万的 HTTP 请求”(算一下就是每秒几百次的请求)。在我精心配置下，在最高峰负荷下，它使用了 15 MB RAM 和 10%的 CPU(FreeBSD 6)。在负载相同的情况下，Apache 已经倒下了(在大约使用 1000 进程后，天知道需要多少内存)；Pound 也倒下了(太多的线程，对所有线程堆栈使用 400MB+的内存)；Lighty 每小时泄露的内存多余 20 MB(使用更多的 CPU，但不太多)。

——Bob Ippolito, ***MochiMedia.com***

如果你为一个大型项目购买资源，但可供支配的资源有限，可选择 Nginx 作为一个优秀的解决方案。如果 Web 服务知识和托管有限，最初可选择 Apache，但是当你迎接成功的时候，你、你的服务器和你的访问者可能最后会发现不一致的地方。

移植 Apache 配置

就是这样，你已经受够了 Apache，最后决定完全切换到 Nginx。从现在开始，要执行相当多的步骤，首先是适应先前的配置，从某种程度上讲，要确定现有的网站的工作和切换后的工作旗鼓相当。

指令

下表将概括一些常用的 Apache 配置指令，试图提供 Nginx 中相当的或可替换的命令。表中命令的顺序遵循默认的 Apache 配置文件中的顺序。

ServerTokens Apache 允许你在请求头中配置服务器的操作系统和软件的名称及版本信息	**server_tokens** 在 Nginx 中，通过该指令可以启用或禁用服务器信息的发送，该指令来源于主 HTTP 模块
ServerRoot 允许你定义服务器的 root 目录，该目录包含配置文件目录和日志目录	**--prefix 编译时的选项** Nginx 在编译时通过使用配置脚本的--prefix 开关变量或者是在执行启动时使用-p 命令行选项
PidFile 定义应用程序的 pid 文件路径	**pid** 准确地相当于指令 pid
TimeOut 该指令定义三个元素：GET 请求的最大执行时间；在 POST 和 PUT 请求中两个 TCP 数据包之间的最大延时；两个 TCP‘ACK’数据包之间的最大延时	**多个指令** ● 这里有多个指令可实现类似的行为： ● send_timeout:定义允许客户端两次读取操作的最大延时； ● client_body_timeout:设置用于读取客户端请求时间的最大值； ● bodyclient_header_timeout:设置读取客户端请求时间的最大值
KeepAlive, MaxKeepAliveRequests, KeepAliveTimeout 这三个指令控制 Apache 的 keep-alive 行为	**keepalive_timeout, keepalive_requests** 这两个指令直接与 Apache 中相应的指令相同，此外，如果想完全禁用 keep-alives，将 keepalive_timeout 或 keepalive_requests 的值设置为 0 即可
Listen 定义 Apache 用于监听连接的网卡接口和端口	**listen** 在 Nginx 中，该指令只能在虚拟主机级别(server 区段)中定义
LoadModule 通过这个指令，Apache 提供了动态载入模块的功能	**--with_****_module** Nginx 不能动态载入模块，所有需要的模块要在编译时包括在内，一旦成为 Nginx 的一部分就不能再禁用

Apache 指令	对应的 Nginx 指令

Apache 指令	对应的 Nginx 指令
ServerTokens Apache 允许你在请求头中配置服务器的操作系统和软件的名称及版本信息	**server_tokens** 在 Nginx 中，通过该指令可以启用或禁用服务器信息的发送，该指令来源于主 HTTP 模块
ServerRoot 允许你定义服务器的 root 目录，该目录包含配置文件目录和日志目录	**--prefix 编译时的选项** Nginx 在编译时通过使用配置脚本的--prefix 开关变量或者是在执行启动时使用-p 命令行选项
PidFile 定义应用程序的 pid 文件路径	**pid** 准确地相当于指令 pid
TimeOut 该指令定义三个元素：GET 请求的最大执行时间；在 POST 和 PUT 请求中两个 TCP 数据包之间的最大延时；两个 TCP‘ACK’数据包之间的最大延时	**多个指令** ● 这里有多个指令可实现类似的行为： ● send_timeout:定义允许客户端两次读取操作的最大延时； ● client_body_timeout:设置用于读取客户端请求时间的最大值； ● bodyclient_header_timeout:设置读取客户端请求时间的最大值
KeepAlive, MaxKeepAliveRequests, KeepAliveTimeout 这三个指令控制 Apache 的 keep-alive 行为	**keepalive_timeout, keepalive_requests** 这两个指令直接与 Apache 中相应的指令相同，此外，如果想完全禁用 keep-alives，将 keepalive_timeout 或 keepalive_requests 的值设置为 0 即可
Listen 定义 Apache 用于监听连接的网卡接口和端口	**listen** 在 Nginx 中，该指令只能在虚拟主机级别(server 区段)中定义
LoadModule 通过这个指令，Apache 提供了动态载入模块的功能	**--with_****_module** Nginx 不能动态载入模块，所有需要的模块要在编译时包括在内，一旦成为 Nginx 的一部分就不能再禁用

Include 文件包含指令，支持通配符	**include** 与 Apache 的功能相同
User, Group 允许你定义用于运行守护进程的用户和组	**user** Nginx 的 user 指令让你指定运行 Nginx 的用户和组
ServerAdmin, ServerSignature 让你指定服务器管理员的邮件地址，签名邮件上会显示的错误和诊断页面	**没有相等的指令** 在 0.7.66 版本的时候，Nginx 没有相似的命令，错误页或其他的信息不能够通过邮件地址发送给服务器管理员
UseCanonicalName 定义 Apache 如何建立子引用 URL	**没有直接的指令** 尽管没有与 Apache 直接对应的命令，但可以通过具体的模块来设定(proxy, FastCGI,等等)
DocumentRoot 定义 Apache 提供文件的 root 目录，该指令可以用在服务器和虚拟主机级别使用	**root** 该指令能够插入 http, server, location 和 if 区段，用于定义文档的 root 目录
DirectoryIndex,IndexOptions,IndexIgnore 定义目录索引和文件列表选项	**index,autoindex,random_index,** fancyindex (第三方) Nginx 也提供好多种选项用于管理索引
AccessFileName 指定文件名为.htaccess 的文件，文件包含在网页上动态执行的文件	**没有对应的指令** Nginx 在版本 0.7.66 时还没有.htaccess 这样功能的文件。详情参见后文
TypesConfig, DefaultType 定义 MIME 类型选项	**types, default_type** 尽管语法不同，但对应的指令存在于 Nginx 中
HostNameLookups 允许查找客户端 IP 地址的主机名，用于日志记录或访问控制目的	**没有对应的指令** Nginx 在版本 0.7.66 时还没有对应的功能

Apache 指令	对应的 Nginx 指令

续表

Apache 指令	对应的 Nginx 指令
Include 文件包含指令，支持通配符	**include** 与 Apache 的功能相同
User, Group 允许你定义用于运行守护进程的用户和组	**user** Nginx 的 user 指令让你指定运行 Nginx 的用户和组
ServerAdmin, ServerSignature 让你指定服务器管理员的邮件地址，签名邮件上会显示的错误和诊断页面	**没有相等的指令** 在 0.7.66 版本的时候，Nginx 没有相似的命令，错误页或其他的信息不能够通过邮件地址发送给服务器管理员
UseCanonicalName 定义 Apache 如何建立子引用 URL	**没有直接的指令** 尽管没有与 Apache 直接对应的命令，但可以通过具体的模块来设定(proxy, FastCGI,等等)
DocumentRoot 定义 Apache 提供文件的 root 目录，该指令可以用在服务器和虚拟主机级别使用	**root** 该指令能够插入 http, server, location 和 if 区段，用于定义文档的 root 目录
DirectoryIndex,IndexOptions,IndexIgnore 定义目录索引和文件列表选项	**index,autoindex,random_index,** fancyindex (第三方) Nginx 也提供好多种选项用于管理索引
AccessFileName 指定文件名为.htaccess 的文件，文件包含在网页上动态执行的文件	**没有对应的指令** Nginx 在版本 0.7.66 时还没有.htaccess 这样功能的文件。详情参见后文
TypesConfig, DefaultType 定义 MIME 类型选项	**types, default_type** 尽管语法不同，但对应的指令存在于 Nginx 中
HostNameLookups 允许查找客户端 IP 地址的主机名，用于日志记录或访问控制目的	**没有对应的指令** Nginx 在版本 0.7.66 时还没有对应的功能

ErrorLog, LogLevel, LogFormat, CustomLog 激活日志记录和格式设置	**access_log, log_format** Nginx 也允许设置种类繁多的选项，但它们组合在较少的指令中
Alias, AliasMatch, ScriptAlias 目录别名选项	**alias** Nginx 也提供了对应的 alias 指令，但没有提供其他两个指令

模块

我们早在第 2 章的时候就了解到，在 Nginx 中模块不能动态载入，必须在编译时包含在内。另外，一旦它们被完全编译和集成到主二进制的 nginx 文件，在运行时也不能够禁止。因此，在编译 Nginx 的时候，需要仔细考虑怎样选择模块。

如果你担心选择的模块会影响到 Nginx 服务器性能，那么你应该意识到唯一有明显差异的模块是来自于“过滤模块(filter modules)”。如果选择的模块是对请求和/或响应的内容做过滤，那么这个模块总是被激活使用。过滤模块的例子：Addition, Charset, Gzip, SSI, 等等。就非过滤模块来说(例如 Autoindex, FastCGI, Stub Status,等等)，如果没有使用它们的指令，那么该模块处理程序永远不会执行。

下表列出了 Apache 和 Nginx 共有的一些模块，注意，可能会有相当(但作用相同)的模块，但它们不会也没有必要提供完全相同的功能，因此在这种情况下，指令很可能并不相同。应该在它们各自的区段中查阅这些模块的文档。

mod_auth_basic	auth_basic	默认包含	--without-http_auth_basic_module
mod_autoindex	autoindex	默认包含	--without-http_autoindex_module
mod_charset_lite	charset	默认包含	--without-http_charset_module
mod_dav	dav	可选	--with-http_dav_module

续表

Apache 指令	对应的 Nginx 指令

Apache 模块	Nginx 模块	状　况	配置选项

续表

Apache 指令	对应的 Nginx 指令
ErrorLog, LogLevel, LogFormat, CustomLog 激活日志记录和格式设置	**access_log, log_format** Nginx 也允许设置种类繁多的选项，但它们组合在较少的指令中
Alias, AliasMatch, ScriptAlias 目录别名选项	**alias** Nginx 也提供了对应的 alias 指令，但没有提供其他两个指令

模块

我们早在第 2 章的时候就了解到，在 Nginx 中模块不能动态载入，必须在编译时包含在内。另外，一旦它们被完全编译和集成到主二进制的 nginx 文件，在运行时也不能够禁止。因此，在编译 Nginx 的时候，需要仔细考虑怎样选择模块。

如果你担心选择的模块会影响到 Nginx 服务器性能，那么你应该意识到唯一有明显差异的模块是来自于“过滤模块(filter modules)”。如果选择的模块是对请求和/或响应的内容做过滤，那么这个模块总是被激活使用。过滤模块的例子：Addition, Charset, Gzip, SSI, 等等。就非过滤模块来说(例如 Autoindex, FastCGI, Stub Status,等等)，如果没有使用它们的指令，那么该模块处理程序永远不会执行。

下表列出了 Apache 和 Nginx 共有的一些模块，注意，可能会有相当(但作用相同)的模块，但它们不会也没有必要提供完全相同的功能，因此在这种情况下，指令很可能并不相同。应该在它们各自的区段中查阅这些模块的文档。

Apache 模块	Nginx 模块	状　况	配置选项
mod_auth_basic	auth_basic	默认包含	--without-http_auth_basic_module
mod_autoindex	autoindex	默认包含	--without-http_autoindex_module
mod_charset_lite	charset	默认包含	--without-http_charset_module
mod_dav	dav	可选	--with-http_dav_module

mod_deflate	gzip	默认包含	--without-http_gzip_module
mod_expires	headers	默认包含	不能禁用
mod_fcgid	fastcgi	默认包含	--without-http_fastcgi_module
mod_headers	Headers	默认包含	不能禁用
mod_include	ssi	默认包含	--without-http_ssi_module
mod_log_config	log	默认包含	不能禁用
mod_proxy	proxy	默认包含	--without-http_proxy_module
mod_rewrite	rewrite	默认包含	--without-http_rewrite_module
mod_ssl	ssl	可选	--with-http_ssl_module
mod_status	stub_status	可选	--with-http_stub_status_module
mod_substitute	sub	可选	--with-http_sub_module
mod_uid	userid	默认包含	--without-http_userid_module

虚拟主机和配置部分

就像 Nginx 允许你在各种级别(http, server, location, if)定义配置设置一样，Apache 也有自己的设置部分。配置示例如下表所示。

配置部分

下表提供了从 Apache 移到 Nginx 配置区段的移植。一些 Apache 部分没有与 Nginx 直接相等的配置，但是对于大多数情况，使用一些稍微不同点的语法就可以实现相同的行为。

Default	http	这个设置在 Apache 配置文件的根部分，相当于在 Nginx 配置文件的根部分设置，也可以将这些设置放在 http 区段(相对于其他模块，例如 mail 或 imap 用于邮件代理服务器的功能)
<VirtualHost>	server	Apache 在 <VirtualHost> 部分的设置，在 Nginx 的配置文件中应该放置在 server 区段部分

Apache 模块	Nginx 模块	状况	配置选项

Apache 部分	Nginx 部分	描　述

续表

Apache 模块	Nginx 模块	状　况	配置选项
mod_deflate	gzip	默认包含	--without-http_gzip_module
mod_expires	headers	默认包含	不能禁用
mod_fcgid	fastcgi	默认包含	--without-http_fastcgi_module
mod_headers	Headers	默认包含	不能禁用
mod_include	ssi	默认包含	--without-http_ssi_module
mod_log_config	log	默认包含	不能禁用
mod_proxy	proxy	默认包含	--without-http_proxy_module
mod_rewrite	rewrite	默认包含	--without-http_rewrite_module
mod_ssl	ssl	可选	--with-http_ssl_module
mod_status	stub_status	可选	--with-http_stub_status_module
mod_substitute	sub	可选	--with-http_sub_module
mod_uid	userid	默认包含	--without-http_userid_module

虚拟主机和配置部分

就像 Nginx 允许你在各种级别(http, server, location, if)定义配置设置一样，Apache 也有自己的设置部分。配置示例如下表所示。

配置部分

下表提供了从 Apache 移到 Nginx 配置区段的移植。一些 Apache 部分没有与 Nginx 直接相等的配置，但是对于大多数情况，使用一些稍微不同点的语法就可以实现相同的行为。

Apache 部分	Nginx 部分	描　述
Default	http	这个设置在 Apache 配置文件的根部分，相当于在 Nginx 配置文件的根部分设置，也可以将这些设置放在 http 区段(相对于其他模块，例如 mail 或 imap 用于邮件代理服务器的功能)
<VirtualHost>	server	Apache 在 <VirtualHost> 部分的设置，在 Nginx 的配置文件中应该放置在 server 区段部分

<Location> <LocationMatch>	location	在 Apache<Location>和<LocationMatch>(正则表达式)中的行为可以被 Nginx 中的 location 区段重现
无	if	Nginx 通过 if 区段提供了动态条件结构(dynamic conditional structure)，在 Apache 中没有完全相等的指令，要说最能靠近相当的那就是来自于 Rewrite 模块的 RewriteCond 指令了
<Directory> <DirectoryMatch> <Files> <FilesMatch>	无	Apache 允许你添加设置到指定的位置(对本地文件系统)，而 Nginx 仅提供了基于每个 URI 设置
<IfDefine>	无	在启动时，如果指定条件为真，那么会应用一组指令。这个功能在 Nginx 中无效
<IfModule>	无	如果指定的模块被载入，那么在启动时会应用一组指令。由于 Nginx 不支持动态载入模块，因此功能也无效
<Proxy> <ProxyMatch>	无	将一组指令应用到被代理的资源(通过指定通配符 URI 或正则表达式)上。在 Nginx 中没有相应的部分

创建虚拟主机

在 Apache 中，虚拟主机是可选项，可以在配置文件的根部定义服务器设置选项：

```
Listen 80
ServerName example.com
ServerAlias www.example.com
DocumentRoot "/home/example.com/www"
[…]
```

Apache 部分	Nginx 部分	描　述

续表

Apache 部分	Nginx 部分	描　述
<Location> <LocationMatch>	location	在 Apache<Location>和<LocationMatch>(正则表达式)中的行为可以被 Nginx 中的 location 区段重现
无	if	Nginx 通过 if 区段提供了动态条件结构(dynamic conditional structure)，在 Apache 中没有完全相等的指令，要说最能靠近相当的那就是来自于 Rewrite 模块的 RewriteCond 指令了
<Directory> <DirectoryMatch> <Files> <FilesMatch>	无	Apache 允许你添加设置到指定的位置(对本地文件系统)，而 Nginx 仅提供了基于每个 URI 设置
<IfDefine>	无	在启动时，如果指定条件为真，那么会应用一组指令。这个功能在 Nginx 中无效
<IfModule>	无	如果指定的模块被载入，那么在启动时会应用一组指令。由于 Nginx 不支持动态载入模块，因此功能也无效
<Proxy> <ProxyMatch>	无	将一组指令应用到被代理的资源(通过指定通配符 URI 或正则表达式)上。在 Nginx 中没有相应的部分

创建虚拟主机

在 Apache 中，虚拟主机是可选项，可以在配置文件的根部定义服务器设置选项：

```
Listen 80
ServerName example.com
ServerAlias www.example.com
DocumentRoot "/home/example.com/www"
[…]
```

然而，这种情况只适用于你打算在服务器上只运行一个网站时，或者是你想定义一个默认设置用于接受与其他虚拟主机访问规则都不匹配的访问。

而在 Nginx 中，运行在同一个服务器上的所有网站必须放在 server 区段中，在这里允许你创建虚拟主机，相当于 Apache 的<VirtualHost>部分。下表描述了从 Apache 的<VirtualHost>到 Nginx 的 server 区段的转译。

<VirtualHost 12.34.56.78:80>	server {
ServerName example.com:80 ServerAlias www.example.com	listen 12.34.56.78:80; server_name example.com www.example.com;
UseCanonicalName Off	# 没有对应的设置
SuexecUserGroup user group	# 没有对应的设置
ServerAdmin "admin@example.com"	# 没有对应的设置
DocumentRoot /home/example.com/www	root /home/example.com/www;
CustomLog /home/example.com/logs/ access_log cust	access_log /home/example.com/logs/access_log cust; #注意，cust 格式必须在前面用 log_format 指令声明过
ErrorLog /home/example.com/logs/ error_log	error_log /home/example.com/logs/error_log;

<Location /documents/>
Options +Indexes
</Location>
	location /documents/ {
	autoindex on;
	}

<IfModule mod_ssl.c> SSLEngine off </IfModule>	# Nginx 中没有与 IfModule 指令相等的指令 ssl off;

Apache 虚拟主机	Nginx 虚拟主机

然而，这种情况只适用于你打算在服务器上只运行一个网站时，或者是你想定义一个默认设置用于接受与其他虚拟主机访问规则都不匹配的访问。

而在 Nginx 中，运行在同一个服务器上的所有网站必须放在 server 区段中，在这里允许你创建虚拟主机，相当于 Apache 的<VirtualHost>部分。下表描述了从 Apache 的<VirtualHost>到 Nginx 的 server 区段的转译。

Apache 虚拟主机	Nginx 虚拟主机
<VirtualHost 12.34.56.78:80>	server {
ServerName example.com:80 ServerAlias www.example.com	listen 12.34.56.78:80; server_name example.com www.example.com;
UseCanonicalName Off	# 没有对应的设置
SuexecUserGroup user group	# 没有对应的设置
ServerAdmin "admin@example.com"	# 没有对应的设置
DocumentRoot /home/example.com/www	root /home/example.com/www;
CustomLog /home/example.com/logs/ access_log cust	access_log /home/example.com/logs/access_log cust; #注意，cust 格式必须在前面用 log_format 指令声明过
ErrorLog /home/example.com/logs/ error_log	error_log /home/example.com/logs/error_log;

<Location /documents/>
Options +Indexes
</Location>
	location /documents/ {
	autoindex on;
	}

<IfModule mod_ssl.c> SSLEngine off </IfModule>	# Nginx 中没有与 IfModule 指令相等的指令 ssl off;

<Directory /home/example.com/www>
<IfModule mod_fcgid.c>
<Files ~ (\.php)>
SetHandler fcgid-script
FCGIWrapper /usr/bin/ php-cgi .php
Options +ExecCGI
allow from all
</Files>
</IfModule>
Options -Includes -ExecCGI
</Directory>
	#在 Nginx 中没有与 Directory 部分相当的设置
	#location 区段只能应用于基于每个 URI 设置 location
	#区段的设置应用于所有对虚拟主机 root 目录
	#请求。我们将它应用到.php 文件设置：
	location ~ \.php {
	# 插入 FCGI 设置
	fastcgi_pass 127.0.0.1:9000;
	fastcgi_param SCRIPT_FILENAME /home/example.com/ www$fastcgi_script_name;
	fastcgi_param PATH_INFO $fastcgi_script_name;
	include fastcgi_params; # 额外的 FastCGI 的设置
	}
	#没有直接相等的其他指令，或 Nginx 不需要

</VirtualHost>	}

前面的转译指导适用于常规虚拟主机，并没有提供安全网页，建立一个使用 SSL 的虚拟主机后，情况就有所不同了。下表的指令聚焦于 SSL 相关指令，但是上表中的指令仍旧可以使用。

<VirtualHost 12.34.56.78:443>	server {
ServerName example.com:443 ServerAlias www.example.com	listen 12.34.56.78:443; server_name example.com www.example.com;
SSLEngine on SSLVerifyClient none SSLCertificateFile /home/ example.com/cert/certchL9435	ssl on; ssl_verify_client off; ssl_certificate /home/example.com/cert/cert.pem; ssl_certificate_key /home/example.com/cert/cert.key;

Apache 虚拟主机	Nginx 虚拟主机

Apache 虚拟主机	Nginx 虚拟主机

续表

Apache 虚拟主机	Nginx 虚拟主机

<Directory
/home/example.com/www>
<IfModule mod_fcgid.c>
<Files ~ (\.php)>
SetHandler fcgid-script
FCGIWrapper /usr/bin/
php-cgi .php
Options +ExecCGI
allow from all
</Files>
</IfModule>
Options -Includes -ExecCGI
</Directory>
	#在 Nginx 中没有与 Directory 部分相当的设置
	#location 区段只能应用于基于每个 URI 设置
	location
	#区段的设置应用于所有对虚拟主机 root 目录
	#请求。我们将它应用到.php 文件设置：
	location ~ \.php {
	# 插入 FCGI 设置
	fastcgi_pass 127.0.0.1:9000;
	fastcgi_param SCRIPT_FILENAME /home/example.com/
	www$fastcgi_script_name;
	fastcgi_param PATH_INFO $fastcgi_script_name;
	include fastcgi_params; # 额外的 FastCGI 的设置
	}
	#没有直接相等的其他指令，或 Nginx 不需要

</VirtualHost>	}

前面的转译指导适用于常规虚拟主机，并没有提供安全网页，建立一个使用 SSL 的虚拟主机后，情况就有所不同了。下表的指令聚焦于 SSL 相关指令，但是上表中的指令仍旧可以使用。

Apache 虚拟主机	Nginx 虚拟主机
<VirtualHost 12.34.56.78:443>	server {
ServerName example.com:443 ServerAlias www.example.com	listen 12.34.56.78:443; server_name example.com www.example.com;
SSLEngine on SSLVerifyClient none SSLCertificateFile /home/ example.com/cert/certchL9435	ssl on; ssl_verify_client off; ssl_certificate /home/example.com/cert/cert.pem; ssl_certificate_key /home/example.com/cert/cert.key;

续表

<Directory /home/example.com/www> SSLRequireSSL </Directory>	# Nginx 中没有相应的规定
</VirtualHost>	}

.htaccess 文件

这一小节将着手处理一个棘手的问题——文件.htaccess，根本问题就是共享主机(shared hosting)。在 Nginx 中确实没有这种机制，而是提供了不同的解决方法。

提示 .htaccess 文件

.htaccess 是一个小的独立的配置文件，允许网管员放在网站的每一个单独的目录中，在请求访问接到一个特定的目录时，Apache 会检查这类文件存在，并将它应用于请求的上下文中。下图所示为允许网管员在多个级别添加独立的设置。

Apache 虚拟主机	Nginx 虚拟主机

/home/example.com/www

- .htaccess
  about.php
  index.php
  [...]

/home/example.com/www/downloads

- .htaccess
  file.zip
  report.doc
  [...]

/home/example.com/www/downloads/protected

- .htaccess
  finances.xls
  sales.xls
  [...]

续表

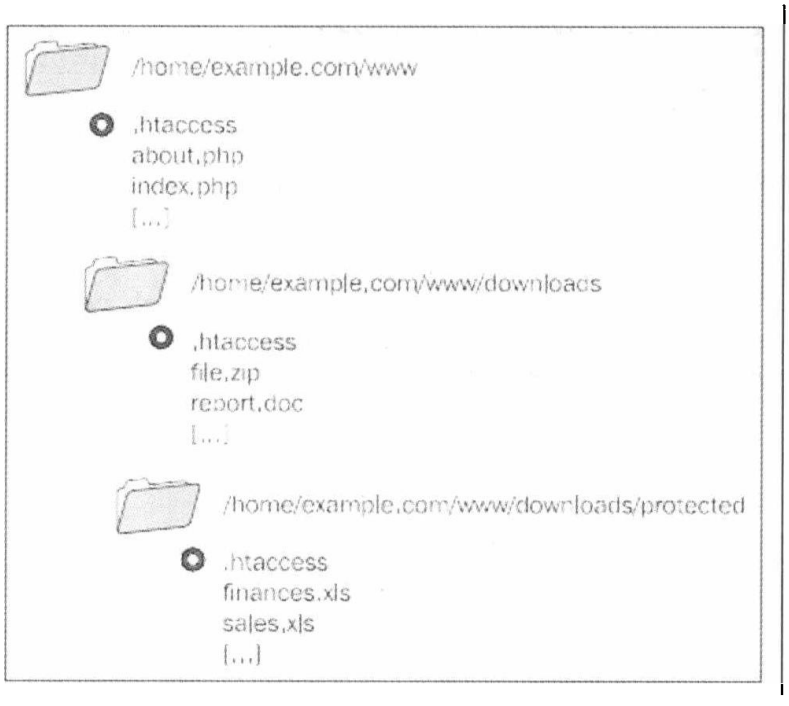
Apache 虚拟主机	Nginx 虚拟主机
<Directory /home/example.com/www> SSLRequireSSL </Directory>	# Nginx 中没有相应的规定
</VirtualHost>	}

.htaccess 文件

这一小节将着手处理一个棘手的问题——文件.htaccess，根本问题就是共享主机(shared hosting)。在 Nginx 中确实没有这种机制，而是提供了不同的解决方法。

提示 .htaccess 文件

.htaccess 是一个小的独立的配置文件，允许网管员放在网站的每一个单独的目录中，在请求访问接到一个特定的目录时，Apache 会检查这类文件存在，并将它应用于请求的上下文中。下图所示为允许网管员在多个级别添加独立的设置。



在一个客户端请求/downloads/protected/finances.xls，这三个.htaccess 文件将以下列顺序应用：

1. /home/example.com/www/.htaccess
2. /home/example.com/www/downloads/.htaccess
3. /home/example.com/www/downloads/protected/.htaccess

如果在/www/.htaccess 和 /www/downloads/.htaccess 中定义相同的设置，那么优先的将是最后读取的.htaccess 文件中的设置，换句话说就是后一个.htaccess 文件优先于前一个.htaccess 文件。

Nginx 的对等实现

遗憾的是，在 Nginx 中没有这种机制，然而，我们可以最大限度地使用可支配的指令，从而找到替代的解决方案。

在 Apache 中，对于.htaccess 主要有三种用法：

- 对于特定目录的访问，建立访问和认证规则；
- 在顶级目录定义重写规则(通常不指定目录)；
- 为模块设置标志，例如 mod_php, mod_perl 或 mod_python。

当我们提到后者(这里指 Apache)，当预处理器被设置为 Apache 的模块时，唯一有效的是使用标志(flag)。如果你的服务器通过 CGI 或 FastCGI 运行 PHP，那么标志将不会被认可，从而产生 500 Internal Server Error 错误。就我们而言，Nginx 连接这样的应用程序只能通过 FastCGI 或 HTTP，因此标志(flag)是不允许的。

依据虚拟主机的声明而定，这里有两种解决方法能够使类.htaccess 行为生效，或者说至少是在一定意义上的相似。

如果打算在唯一的配置文件中列出所有的虚拟主机，那么第一种解决方法就是利用 include 指令在 server 区段插入。在下面的例子中，被包含的虚拟主机配置文件在目录/home/example.com/www/下，server 的 root 为/home/example.com/www，因此，可以将/www/视为文档的 root 目录，那么被包含的文件就有可能会被客户端下载，因此不要忘记这一点：

```
Server {
    listen 80;
```

```
    server_name .example.com;
    root /home/example.com/www;
    […]
    # 包含附加的配置文件
    location / {
        include /home/example.com/www/.ngconf*;
    }
    # 如果有人尝试下载这类文件，则拒绝下载
    location ~ \.ngconf {
        return 404;
    }
}
```

这将包含目录/www/下任何以.ngconf 开头的文件名。注意，通配符在 include 指令中。如果指定的文件名没有通配符，那么如果该文件丢失，Nginx 会认为配置为无效；如果使用通配符，那么即使这样的文件不存在，Nginx 也不会产生任何错误。

然后在文件.ngconf 中包含涉及虚拟主机自身的指令：

```
autoindex off; # 禁止显示目录列表
location /downloads/ {
    autoindex on; # 允许目录/downloads/产生列表
}
[…]
```

对于 Web 托管服务提供商，这个解决方案似乎相对安全，因为这将只允许网站管理员定义与 location 相关的设置(防止重要的修改，例如使用不同的端口，不同的主机名，等等)。然而，需要注意的是，如果一个网站的管理员建立了一个无效的.ngconf 文件，Nginx 会拒绝重新载入配置文件，直到问题被修复。

另一种方案是，把虚拟主机的声明放在一个单独的文件中，放在每一个虚拟主机的根路径下。在这种情况下，只需要 Nginx 主配置文件中的以下指令：

```
include /home/*/www/.ngconf;
```

于是，文件.ngconf 只得包含一个完整的虚拟主机声明，包括端口和服务器名字，这种解决方法仅被考虑为你自己能够完全管理服务器，你永远不应该允许外部 Web 站点管理员对你的服务器有这么多的控制权。

不过，Apache 和 Nginx 之间仍旧有显著的差异。

1. 对于每次客户端的请求，Apache 都将.htaccess 的设置应用并进行处理。
2. Nginx 只在重新载入配置文件时应用.ngconf 文件(例如 service nginx reload)。

目前，对于最后这个问题还没有解决办法，Nginx 不允许动态改变配置。

重写规则

HTTP 服务器切换最让人头痛的是重写规则，不幸的是，Nginx 不能直接与 Apache 重写规则兼容，主要有以下两方面：

1. 通常，重写规则放在.htaccess 文件中，正如前面所讨论的，Nginx 没有提供这样的机制，因此重写规则就只能放在不同的 location 中；
2. 改写操作和条件的语法迥然不同，并且需要调整。

本章将深入探讨将 apache 规则迁移到 Nginx 过程中遇到的问题，并且为一些主要的 Web 应用提供预先写好的规则。

一般意见

在 Nginx 中研究实际的案例之前，让我们以一些重要重写规则的建议开始。

位置

通过对 Nginx 的描述，我们可以有把握地说，对于共享主机(shared hosting)的托管公司，Nginx 不是最适当的 Web 服务器。缺乏.htaccess 文件，导致几乎不可能对主机上的网站有各自的设置，其中包括重写规则。虽然在上一节中已提供替代解决方案，但不是最佳的解决方案，因为在每次更改后都需要重新载配置文件。将它们成功添加之后，如果整个配置文件不包括任何错误，那么只有重新加载才能通过。

第一个问题的后果是，必须迁移重写规则，无论哪一个文件包含虚拟主机的配置，它们必须直接放入虚拟主机的 server 或 location 区段。在 Apache 中，重写规则应该放置在某处，例如/home/example.com/www/.htaccess；然而，在 Nginx 中，需要将它们作为虚拟主机配置文件的一部分，例如 /usr/local/nginx/conf/nginx.conf。

语法

将重写规则移植到 Nginx 时，Apache 有两个指令很重要，它没有其他相应的指令。并非故意不支持，而是它们的行为已经成为 Nginx 提供的等价部分(换句话说，Nginx 已经提供了这样的功能)。

1. RewriteCond 允许你定义条件，然后重写匹配的 URL；
2. RewriteRule 通过指定正则表达式、重写的 URL 和一组标志具体执行 URL 重写。

第一个指令 RewriteCond 相当于 Nginx 的 if，它的作用是在应用重写规则之前进行条件校验，下面的例子中，在重写 URL 前确定一下请求的文件是否存在：

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule . /index.php [L]
```

Nginx 与之相当的是 if 和 rewrite，如下所示:

```
if (!-f $request_filename) {
    rewrite . /index.php last;
}
```

想在多个条件下使用改写时，会有点复杂：Nginx 的 if 语句在表达式中只支持一个条件，不允许 if 块嵌套。那么 Nginx 如何重现类似如下行为呢？

```
RewriteCond %{REQUEST_FILENAME} !-f      # 文件肯定不存在
RewriteCond %{REQUEST_FILENAME} !-d      # 目录肯定不存在
RewriteRule . /index.php [L]             # 重写 URL
```

对于这个具体问题，有一个简单的逻辑解决办法——我们将使用多个 if 块来影响变量。在下面的例子中，在经过前面两个 if 块后，第三个进入检测的 if 块，其变量值由前面的两个变量值来决定：

```
set $check "";
# 如果指定的文件不存在，则设置$check 为 "A"
if (!-f $request_filename) {
    set $check "A";
}
#如果指定的目录不存在，则设置$check 为$check + B
if (!-d $request_filename) {
    set $check "${check}B";
}
```

```
# 如果$check 在两个 if 块中都被影响，则执行重写
if ($check = "AB") {
    rewrite . /index.php last
}
```

注意这两个特殊的重写条件(-f 用来测试文件是否存在；-d 用来测试目录是否存在)，Nginx 已经提供了一个将两者合起来测试的符号：-e。因此，快速解决方案如下：

```
if (!-e $request_filename) {
    rewrite . /index.php last;
}
```

除了测试文件或目录是否存在，-e 也检测符号链接(如果指定的文件名相当于现有链接的话)。

总的来说，有关 Rewrite 模块更多的信息，请参阅第 5 章。

重写规则

Apache 的 RewriteRule 指令直接对应于 Nginx 的 rewrite 指令，然而，它们之间还是有一点微妙的差异——在 Nginx 中 URI 由“/”符号开始，尽管如此，仍然需要简单的转译：

```
RewriteRule ^downloads/(.*)$ download.php?url=$1 [QSA]
```

前面的 Apache 规则转化如下：

```
rewrite ^/downloads/(.*)$ /download.php?url=$1;
```

注意，标志[QSA]是告诉 Apache 将查询参数添加到重写的 URL，然而，这个操作对 Nginx 来说是默认的。若要阻止 Nginx 附加查询参数，在 URL 尾部插入一个符号“？”即可：

```
rewrite ^/downloads/(.*)$ /download.php?url=$1?;
```

Apache 的指令 RewriteRule 允许其他的标志，这些对应于 Nginx 指定的标志，详见第 5 章的相关描述。

WordPress

WordPress 对你来说可能并不陌生。从 2009 年 9 月以来，这个非常流行的开源博客应用程序在全球有 2 亿多网站使用。它由 PHP 和 MySQL 技术实现，它兼容 Nginx 的方便易用。不错，如果不是为了不重写规则，这种说法是完全真实的。

该 Web 应用提供的.htaccess 文件放在网站的 root 目录：

```
# BEGIN WordPress
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule . /index.php [L]
</IfModule>
# END WordPress
```

相较而言，第一个例子比较容易理解和转换到 Nginx。事实上，大多数重写规则处理过程包含三个步骤。

1. 检查被请求的 URI 是否相当于已经存在的文件。在这种情况下，正常提供。
2. 检查被请求的 URI 是否相当于一个目录。在这种情况下，正常提供。
3. WordPress 利用内部的 PHP 脚本自己分析 URI(通过检查 $_SERVER ["REQUEST_URI"]变量)改写到 index.php。

由于不需要操心大量的复杂规则，URL 由 PHP 脚本自己解析，因此转换为 Nginx 相当容易。下面是一个 Nginx 虚拟主机的完整实例，但是移除了其他不相关的指令：

```
server {
    listen 80;
    server_name blog.example.com;
    root /home/example.com/blog/www;
    index index.php;
    location / {
    # 如果请求的 URI 没有与任何存在的文件、目录或符号链接，
    # 则重写 URL 到 index.php
        if (!-e $request_filename) {
        rewrite ^ /index.php last;
    }
}

    # 对于所有的 PHP 请求，通过 FastCGI 把它们传递到 PHP-FPM
    # 详情参阅第 6 章
    location ~ \.php$ {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_param SCRIPT_FILENAME
/home/example.com/blog/www$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_script_name;
        include fastcgi_params; # 包含附加的 FCGI 参数
    }
}
```

MediaWiki

正如其名称所示，Media Wiki 是一个 Web 引擎，授权于著名的 Wikipedia，一个在线开放的百科全书。当前它是一个开源软件，每一个人都可以下载并且在本地的服务器安装它。该程序也能够作为 CMS(内容管理软件)使用，一些大公司（例如 Novell）已经发现它是一个可靠的解决方案。

和 WordPress 相反，MediaWiki 没有提供预先写好的用于美化 URL 的.htaccess 文件，取而代之的是，MediaWiki 官方网站上提供了各种各样的方法，所有文档用 wiki 文章的形式呈现。网站管理员可以直接修改 Apache 主配置文件，使解决方法生效。然而，也有简单的解决方案，可以省去这样的麻烦。这里没有特别的 Apache 解决方法，下面的三个 Nginx 重写规则足以满足需要。

- 重定向默认请求(例如请求 URI 的作为/)到/wiki/Main_Page；
- 重定向所有/wiki/abcd 形式的 URI 到实际的/w/index.php?title=abcd，不要忘记在请求的 URL 后添加参数；
- 确保对/wiki 的请求重定向到主页/w/index.php。

下面是一个完整的虚拟主机配置示例，包括重写规则：

```
server {
    listen 80;
    server_name wiki.example.com;
    root /home/example.com/wiki/www;
    location / {
        index index.php;
        rewrite ^/$ /wiki/Main_Page permanent;
    }
    # 两个重写规则
    rewrite ^/wiki/([^?]*)(?:\?(.*))? /w/index.php?title=$1&$2;
    rewrite ^/wiki /w/index.php;
```

```
    # 这里是你的 FCGI 的配置
    location ~ \.php$ {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME
/home/example.com/wiki/www$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

vBulletin

论坛在 21 世纪初逐渐流行起来，很多流行的 Web 应用程序也开始出现，例如 vBulletin，phpBB 或 Invision Board。它们中的大多数论坛软件都卷入这场声势浩大的浪潮，并且自吹自擂 URL 完全友好地支持 SEO。不幸的是，重写规则经常以.htaccess 文件的形式出现。的确，vBulletin 的开发者在 Apache 2 和 IIS 中已经选择提供了重写规则，毫无悬念的是他们忘记了 Nginx。让我们给他们一个教训，下表描述了一个解决方法，用于将 Apache 的重写规则转化为 Nginx 的重写规则。

Apache 规则	Nginx 规则
RewriteEngine on	# 没有必要

RewriteCond %{REQUEST_FILENAME} -s [OR]
RewriteCond %{REQUEST_FILENAME} -l [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^.*$ - [NC,L]
	# 如果请求的 URI 相当于系统中存在的文件、目录或链接，则不重写
	if (-e $request_filename) {
	break;
	}

RewriteRule ^threads/.* showthread.php [QSA]	rewrite ^/threads/.*$/showthread.php last;
RewriteRule ^forums/.* forumdisplay.php [QSA]	rewrite ^/forums/.*$/forumdisplay.php last;
RewriteRule ^members/.* member.php [QSA]	rewrite ^/members/.*$ /members.php last;
RewriteRule ^blogs/.* blog.php [QSA]	rewrite ^/blogs/.*$ /blog.php last;
ReWriteRule ^entries/.* entry.php [QSA]	rewrite ^/entries/.*$ /entry.php last;

RewriteCond %{REQUEST_FILENAME} -s [OR] RewriteCond %{REQUEST_FILENAME} -l [OR] RewriteCond %{REQUEST_FILENAME} -d RewriteRule ^.*$ - [NC,L]	# 出于某种原因，在 vBulletin 提供的.htaccess 文件中，同样的规则出现了两次。但不需要在 Nginx 中插入两次。
RewriteRule ^(?:(.*?)(?:/|$))(.*\|$)$ $1.php?r=$2 [QSA]	rewrite ^/(?:(.*?)(?:/|$))(.*\|$)$ /$1.php?r=$2 last;

小结

从 Apache 切换到 Nginx，可能一开始看起来很复杂。在这个过程中涉及许多步骤，如果没有信心和充分准备，你可能会一筹莫展。您需要了解 Nginx 目前的局限：不能在运行时改变配置，没有.htaccess 或类似功能。Nginx 完全没有 Apache 那么多的模块，至少目前还没有。最后同样重要的是，必须转换所有网站重写规则，因此，它确实需要相当多工作。但是，这个很小的代价可以换得一个可以确保长期稳定和可扩展的服务器，你和你的访问者也不会后悔，因为它一般都会提高负载和响应速度。

Apache 规则	Nginx 规则

续表

Apache 规则	Nginx 规则
RewriteCond %{REQUEST_FILENAME} -s [OR] RewriteCond %{REQUEST_FILENAME} -l [OR] RewriteCond %{REQUEST_FILENAME} -d RewriteRule ^.*$ - [NC,L]	# 出于某种原因，在 vBulletin 提供的.htaccess 文件中，同样的规则出现了两次。但不需要在 Nginx 中插入两次。
RewriteRule ^(?:(.*?)(?:/|$))(.*\|$)$ $1.php?r=$2 [QSA]	rewrite ^/(?:(.*?)(?:/|$))(.*\|$)$ /$1.php?r=$2 last;

小结

从 Apache 切换到 Nginx，可能一开始看起来很复杂。在这个过程中涉及许多步骤，如果没有信心和充分准备，你可能会一筹莫展。您需要了解 Nginx 目前的局限：不能在运行时改变配置，没有.htaccess 或类似功能。Nginx 完全没有 Apache 那么多的模块，至少目前还没有。最后同样重要的是，必须转换所有网站重写规则，因此，它确实需要相当多工作。但是，这个很小的代价可以换得一个可以确保长期稳定和可扩展的服务器，你和你的访问者也不会后悔，因为它一般都会提高负载和响应速度。

附录 A
指令索引

下表列出的指令来自于 Nginx 所有可用的第三方模块，每一条指令都有一个简单的概述。如果一个模块没有被默认包含，那么表中指出的指令会标记“*”。在相应的章节中会找到更详细的信息。表中指令按字母顺序排列。

accept_mutex：启用或禁用互斥 第 3 章，“时间模块”小节	Events
accept_mutex_delay：设置延时接受互斥 第 3 章，“时间模块”小节	Events
access_log：定义访问日志设置 第 5 章，“站点的访问和日志记录”小节	Log
add_after_body：在响应体之后添加内容 第 5 章，“内容和编码”小节	Addition*
add_before_body：在响应体之前添加内容 第 5 章，“内容和编码”小节	Addition*
add_header：为响应添加头 第 5 章，“内容和编码”小节	Headers
alias：设置目录别名 第 4 章，“路径和文档”小节	HTTP Core
allow：允许一个 IP 地址或一个 IP 范围访问特定的资源 第 5 章，“限制和约束”小节	Access

指令	模块

附录 A
指令索引

下表列出的指令来自于 Nginx 所有可用的第三方模块，每一条指令都有一个简单的概述。如果一个模块没有被默认包含，那么表中指出的指令会标记“*”。在相应的章节中会找到更详细的信息。表中指令按字母顺序排列。

指　令	模　块
accept_mutex：启用或禁用互斥 第 3 章，“时间模块”小节	Events
accept_mutex_delay：设置延时接受互斥 第 3 章，“时间模块”小节	Events
access_log：定义访问日志设置 第 5 章，“站点的访问和日志记录”小节	Log
add_after_body：在响应体之后添加内容 第 5 章，“内容和编码”小节	Addition*
add_before_body：在响应体之前添加内容 第 5 章，“内容和编码”小节	Addition*
add_header：为响应添加头 第 5 章，“内容和编码”小节	Headers
alias：设置目录别名 第 4 章，“路径和文档”小节	HTTP Core
allow：允许一个 IP 地址或一个 IP 范围访问特定的资源 第 5 章，“限制和约束”小节	Access

ancient_browser：如果请求用户代理与一个指定的字符串匹配，则会影响到$ancient_browser 第 5 章，“与访问者相关”小节	Browser
ancient_browser_value：设置影响变量$ancient_browser 的值 第 5 章，“与访问者相关”小节	Browser
auth_basic：设置显示在基本身份验证对话框的文字信息 第 5 章，“限制和约束”小节	Auth Basic
auth_basic_user_file：定义其中包含基本身份验证的用户名和密码文件 第 5 章，“限制和约束”小节	Auth Basic
autoindex：开启自动目录索引 第 5 章，“站点的访问和日志记录”小节	Autoindex
autoindex_exact_size：以字节为单位自动显示目录索引文件的大小 第 5 章，“站点的访问和日志记录”小节	Autoindex
autoindex_localtime：启用或禁用文件的日期调整，以符合服务器的本地时间 第 5 章，“站点的访问和日志记录”小节	Autoindex
break：阻止对访问网址的进一步改写 第 5 章，“Rewrite 模块指令”指令	Rewrite
charset：在 HTTP 头中设置 Content-Type 字符集类型 第 5 章，“内容和编码”小节	Charset
charset_map：定义重新编码的字符表 第 5 章，“内容和编码”小节	Charset
charset_types：定义可被 charset 选择的 MIME 类型 第 5 章，“内容和编码”小节	Charset
client_body_buffer_size：指定用于客户端请求体缓冲区的大小 第 4 章，“客户端请求”小节	HTTP Core

指令	模块

续表

指　令	模　块
ancient_browser：如果请求用户代理与一个指定的字符串匹配，则会影响到$ancient_browser 第 5 章，“与访问者相关”小节	Browser
ancient_browser_value：设置影响变量$ancient_browser 的值 第 5 章，“与访问者相关”小节	Browser
auth_basic：设置显示在基本身份验证对话框的文字信息 第 5 章，“限制和约束”小节	Auth Basic
auth_basic_user_file：定义其中包含基本身份验证的用户名和密码文件 第 5 章，“限制和约束”小节	Auth Basic
autoindex：开启自动目录索引 第 5 章，“站点的访问和日志记录”小节	Autoindex
autoindex_exact_size：以字节为单位自动显示目录索引文件的大小 第 5 章，“站点的访问和日志记录”小节	Autoindex
autoindex_localtime：启用或禁用文件的日期调整，以符合服务器的本地时间 第 5 章，“站点的访问和日志记录”小节	Autoindex
break：阻止对访问网址的进一步改写 第 5 章，“Rewrite 模块指令”指令	Rewrite
charset：在 HTTP 头中设置 Content-Type 字符集类型 第 5 章，“内容和编码”小节	Charset
charset_map：定义重新编码的字符表 第 5 章，“内容和编码”小节	Charset
charset_types：定义可被 charset 选择的 MIME 类型 第 5 章，“内容和编码”小节	Charset
client_body_buffer_size：指定用于客户端请求体缓冲区的大小 第 4 章，“客户端请求”小节	HTTP Core

client_body_in_file_only：强制 Nginx 在所有情况下将客户端请求体存储为一个文件 第 4 章，“客户端请求”小节	HTTP Core
client_body_in_single_buffer：定义是否将客户端请求体存储在一个缓冲区 第 4 章，“客户端请求”小节	HTTP Core
client_body_temp_path：设置用于临时存储客户端请求文件的路径 第 4 章，“客户端请求”小节	HTTP Core
client_body_timeout：设置读客户端请求体超时 第 4 章，“客户端请求”小节	HTTP Core
client_header_buffer_size：设置分配给请求头的缓冲大小 第 4 章，“客户端请求”小节	HTTP Core
client_header_timeout：设置读取客户端请求头超时 第 4 章，“客户端请求”小节	HTTP Core
client_max_body_size：设置客户端请求体的最大值 第 4 章，“客户端请求”小节	HTTP Core
connection_pool_size：定义用于连接内存池的使用空间大小 第 4 章，“套接字和主机的配置”	HTTP Core
connections：不赞成使用 (参考 worker_connections) 第 3 章，“事件模块”小节	Events
create_full_put_path：启用或禁用用于 PUT 请求的递归目录建立(建立全路径) 第 5 章，“其他杂项模块”小节	DAV*
daemon：启用或禁用守护进程模式 第 3 章，“核心模块指令”小节	Core
dav_access：对当前的级别定义访问权限 第 5 章，“其他杂项模块”小节	DAV*

指令	模块

续表

指　令	模　块
client_body_in_file_only：强制 Nginx 在所有情况下将客户端请求体存储为一个文件 第 4 章，“客户端请求”小节	HTTP Core
client_body_in_single_buffer：定义是否将客户端请求体存储在一个缓冲区 第 4 章，“客户端请求”小节	HTTP Core
client_body_temp_path：设置用于临时存储客户端请求文件的路径 第 4 章，“客户端请求”小节	HTTP Core
client_body_timeout：设置读客户端请求体超时 第 4 章，“客户端请求”小节	HTTP Core
client_header_buffer_size：设置分配给请求头的缓冲大小 第 4 章，“客户端请求”小节	HTTP Core
client_header_timeout：设置读取客户端请求头超时 第 4 章，“客户端请求”小节	HTTP Core
client_max_body_size：设置客户端请求体的最大值 第 4 章，“客户端请求”小节	HTTP Core
connection_pool_size：定义用于连接内存池的使用空间大小 第 4 章，“套接字和主机的配置”	HTTP Core
connections：不赞成使用 (参考 worker_connections) 第 3 章，“事件模块”小节	Events
create_full_put_path：启用或禁用用于 PUT 请求的递归目录建立(建立全路径) 第 5 章，“其他杂项模块”小节	DAV*
daemon：启用或禁用守护进程模式 第 3 章，“核心模块指令”小节	Core
dav_access：对当前的级别定义访问权限 第 5 章，“其他杂项模块”小节	DAV*

dav_methods：选择要启用的 DAV 方法 第 5 章，“其他杂项模块”小节	DAV*
debug_connection：对指定的 IP 或 IP 范围开启详细的日志记录 第 3 章，“事件模块”小节	Events
debug_points：启用或禁用调试点 第 3 章，“核心模块指令”小节	Core
default_type：为 Nginx 提供的文件设置默认的 MIME 类型 第 4 章，“MIME 类型”小节	HTTP Core
deny：拒绝一个 IP 或 IP 范围访问一个资源 第 5 章，“限制和约束”小节	Access
directio：启用或禁用直接使用 I/O 第 4 章，“文件处理和缓存”小节	HTTP Core
empty_gif：有内存提供一个空的 gif 文件 第 5 章，“内容和编码”小节	Empty GIF
env：定义或重新定义环境变量 第 3 章，“核心模块指令”小节	Core
error_log：指定错误日志设置 第 3 章，“核心模块指令”小节	Core
error_page：定义错误行为的具体代码 第 4 章，“路径和文档”小节	HTTP Core
expires：控制缓存响应数据包中发送的头信息 第 5 章，“内容和编码”小节	Headers
fastcgi_buffer_size：设置 FastCGI 响应缓冲区的大小 第 6 章，“主要指令”小节	FastCGI
fastcgi_buffers：设置用于和后台 FastCGI 通信的缓冲数量和大小 第 6 章，“主要指令”小节	FastCGI
fastcgi_cache：定义一个 FastCGI 缓存区域 第 6 章，“FastCGI 缓存”小节	FastCGI

指令	模块

续表

指　　令	模　　块
dav_methods：选择要启用的 DAV 方法 第 5 章，“其他杂项模块”小节	DAV*
debug_connection：对指定的 IP 或 IP 范围开启详细的日志记录 第 3 章，“事件模块”小节	Events
debug_points：启用或禁用调试点 第 3 章，“核心模块指令”小节	Core
default_type：为 Nginx 提供的文件设置默认的 MIME 类型 第 4 章，“MIME 类型”小节	HTTP Core
deny：拒绝一个 IP 或 IP 范围访问一个资源 第 5 章，“限制和约束”小节	Access
directio：启用或禁用直接使用 I/O 第 4 章，“文件处理和缓存”小节	HTTP Core
empty_gif：有内存提供一个空的 gif 文件 第 5 章，“内容和编码”小节	Empty GIF
env：定义或重新定义环境变量 第 3 章，“核心模块指令”小节	Core
error_log：指定错误日志设置 第 3 章，“核心模块指令”小节	Core
error_page：定义错误行为的具体代码 第 4 章，“路径和文档”小节	HTTP Core
expires：控制缓存响应数据包中发送的头信息 第 5 章，“内容和编码”小节	Headers
fastcgi_buffer_size：设置 FastCGI 响应缓冲区的大小 第 6 章，“主要指令”小节	FastCGI
fastcgi_buffers：设置用于和后台 FastCGI 通信的缓冲数量和大小 第 6 章，“主要指令”小节	FastCGI
fastcgi_cache：定义一个 FastCGI 缓存区域 第 6 章，“FastCGI 缓存”小节	FastCGI

fastcgi_cache_key： 该指令用来定义缓存 key 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_cache_methods：为 FastCGI 缓存设置适当的 HTTP 方法 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_cache_path：为指定的区域(zone)配置 FastCGI 缓存选项 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_cache_use_stale：定义是否在某些情况下使用过时的缓存数据 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_cache_valid：该指令允许你对各种不同的响应代码定制不同的缓存时间 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_catch_stderr：允许你拦截一些发送到标准错误设备(stderr)的错误信息并将它们存储在 Nginx 的错误日志中 第 6 章，“主要指令”小节	FastCGI
fastcgi_connect_timeout：定义后端服务器连接超时 第 6 章，“主要指令”小节	FastCGI
fastcgi_hide_header：跳过 FastCGI 头 第 6 章，“主要指令”小节	FastCGI
fastcgi_ignore_client_abort：当客户端放弃请求的时候，设置 FastCGI 模块行为 第 6 章，“主要指令”小节	FastCGI
fastcgi_ignore_headers：Nginx 阻止处理从后台服务器响应的四种头信息(之一)：X-Accel-Redirect, X-Accel-Expires, Expires, Cache-Control 第 6 章，“主要指令”小节	FastCGI
fastcgi_index：为 FastCGI 指定目录索引 第 6 章，“主要指令”小节	FastCGI

指令	模块

续表

指　令	模　块
fastcgi_cache_key： 该指令用来定义缓存 key 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_cache_methods：为 FastCGI 缓存设置适当的 HTTP 方法 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_cache_path：为指定的区域(zone)配置 FastCGI 缓存选项 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_cache_use_stale：定义是否在某些情况下使用过时的缓存数据 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_cache_valid：该指令允许你对各种不同的响应代码定制不同的缓存时间 第 6 章，“FastCGI 缓存”小节	FastCGI
fastcgi_catch_stderr：允许你拦截一些发送到标准错误设备(stderr)的错误信息并将它们存储在 Nginx 的错误日志中 第 6 章，“主要指令”小节	FastCGI
fastcgi_connect_timeout：定义后端服务器连接超时 第 6 章，“主要指令”小节	FastCGI
fastcgi_hide_header：跳过 FastCGI 头 第 6 章，“主要指令”小节	FastCGI
fastcgi_ignore_client_abort：当客户端放弃请求的时候，设置 FastCGI 模块行为 第 6 章，“主要指令”小节	FastCGI
fastcgi_ignore_headers：Nginx 阻止处理从后台服务器响应的四种头信息(之一)：X-Accel-Redirect，X-Accel-Expires，Expires，Cache-Control 第 6 章，“主要指令”小节	FastCGI
fastcgi_index：为 FastCGI 指定目录索引 第 6 章，“主要指令”小节	FastCGI

fastcgi_intercept_errors：定义 FastCGI 后端服务器产生错误是否返回“as is”。Nginx 是否处理被网关返回的错误或是直接向客户端返回错误页 第 6 章，“主要指令”小节	FastCGI
fastcgi_max_temp_file_size：设置临时文件的最大值 第 6 章，“主要指令”小节	FastCGI
fastcgi_next_upstream：该指令指这样一种情况，当客户端请求被抛弃(be abandoned)时，请求将被转发到下一个 FastCGI 服务器 第 6 章，“主要指令”小节	FastCGI
fastcgi_param：配置被传递到后端服务器的的 FastCGI 头 第 6 章，“主要指令”小节	FastCGI
fastcgi_pass：指定 FastCGI 使用指定的 location，以便转发到后端 FastCGI 服务器 第 6 章，“主要指令”小节	FastCGI
fastcgi_pass_header：将 FastCGI 忽略的头重新启用 第 6 章，“主要指令”小节	FastCGI
fastcgi_pass_request_body：定义请求体是否应该传递到后端服务器 第 6 章，“主要指令”小节	FastCGI
fastcgi_pass_request_headers：定义额外的请求头是否应该传递到后端服务器 第 6 章，“主要指令”小节	FastCGI
fastcgi_read_timeout：设置从后端服务器读取超时 第 6 章，“主要指令”小节	FastCGI
fastcgi_send_lowat：允许你在 FastCGI 通信中使用 TCP 套接字标志 SO_SNDLOWAT，仅用于 FreeBSD 第 6 章，“主要指令”小节	FastCGI
fastcgi_send_timeout：定义对于发送数据到后台服务器的超时时间 第 6 章，“主要指令”小节	FastCGI
fastcgi_split_path_info：根据正则表达式分裂 URI 路径 第 6 章，“主要指令”小节	FastCGI

指令	模块

续表

指　令	模　块
fastcgi_intercept_errors：定义 FastCGI 后端服务器产生错误是否返回“as is”。Nginx 是否处理被网关返回的错误或是直接向客户端返回错误页 第 6 章，“主要指令”小节	FastCGI
fastcgi_max_temp_file_size：设置临时文件的最大值 第 6 章，“主要指令”小节	FastCGI
fastcgi_next_upstream：该指令指这样一种情况，当客户端请求被抛弃(be abandoned)时，请求将被转发到下一个 FastCGI 服务器 第 6 章，“主要指令”小节	FastCGI
fastcgi_param：配置被传递到后端服务器的的 FastCGI 头 第 6 章，“主要指令”小节	FastCGI
fastcgi_pass：指定 FastCGI 使用指定的 location，以便转发到后端 FastCGI 服务器 第 6 章，“主要指令”小节	FastCGI
fastcgi_pass_header：将 FastCGI 忽略的头重新启用 第 6 章，“主要指令”小节	FastCGI
fastcgi_pass_request_body：定义请求体是否应该传递到后端服务器 第 6 章，“主要指令”小节	FastCGI
fastcgi_pass_request_headers：定义额外的请求头是否应该传递到后端服务器 第 6 章，“主要指令”小节	FastCGI
fastcgi_read_timeout：设置从后端服务器读取超时 第 6 章，“主要指令”小节	FastCGI
fastcgi_send_lowat：允许你在 FastCGI 通信中使用 TCP 套接字标志 SO_SNDLOWAT，仅用于 FreeBSD 第 6 章，“主要指令”小节	FastCGI
fastcgi_send_timeout：定义对于发送数据到后台服务器的超时时间 第 6 章，“主要指令”小节	FastCGI
fastcgi_split_path_info：根据正则表达式分裂 URI 路径 第 6 章，“主要指令”小节	FastCGI

fastcgi_store：定义 FastCGI 缓存存储设置 第 6 章，“主要指令”小节	FastCGI
fastcgi_store_access：设置 FastCGI 缓存存储访问权限 第 6 章，“主要指令”小节	FastCGI
fastcgi_temp_file_write_size：将临时文件保存到磁盘驱动器时，设置写缓存的大小 第 6 章，“主要指令”小节	FastCGI
fastcgi_temp_path：设置 FastCGI 临时文件的路径 第 6 章，“主要指令”小节	FastCGI
flv：支持 FLV 文件请求 第 5 章，“内容和编码”小节	FLV*
geo：定义一个基于客户端 IP 地址来源的 map 值 第 5 章，“与你的访问者相关”小节	Geo
geoip_city：设置基于城市 IP 的数据库路径 第 5 章，“与你的访问者相关”小节	Geo IP*
geoip_country：设置基于国家 IP 数据库的路径 第 5 章，“与你的访问者相关”小节	Geo IP*
google_perftools_profiles：设定 Google 性能分析工具产生的日志存放文件名 第 5 章，“其他杂项模块”小节	Google Perftools*
gzip_buffers：定义用于存储被压缩的响应头的缓存数量和大小 第 5 章，“内容和编码”小节	Gzip
gzip_comp_level：定义对响应数据包的压缩级别 第 5 章，“内容和编码”小节	Gzip
gzip_disable：对于与指定的正则表达式匹配的用户代理 HTTP 请求头禁用 Gzip 压缩 第 5 章，“内容和编码”小节	Gzip
gzip_hash：设置分配给内部压缩状态(memLevel 参数)的内存总数 第 5 章，“内容和编码”小节	Gzip

指令	模块

续表

指　　令	模　　块
fastcgi_store：定义 FastCGI 缓存存储设置 第 6 章，“主要指令”小节	FastCGI
fastcgi_store_access：设置 FastCGI 缓存存储访问权限 第 6 章，“主要指令”小节	FastCGI
fastcgi_temp_file_write_size：将临时文件保存到磁盘驱动器时，设置写缓存的大小 第 6 章，“主要指令”小节	FastCGI
fastcgi_temp_path：设置 FastCGI 临时文件的路径 第 6 章，“主要指令”小节	FastCGI
flv：支持 FLV 文件请求 第 5 章，“内容和编码”小节	FLV*
geo：定义一个基于客户端 IP 地址来源的 map 值 第 5 章，“与你的访问者相关”小节	Geo
geoip_city：设置基于城市 IP 的数据库路径 第 5 章，“与你的访问者相关”小节	Geo IP*
geoip_country：设置基于国家 IP 数据库的路径 第 5 章，“与你的访问者相关”小节	Geo IP*
google_perftools_profiles：设定 Google 性能分析工具产生的日志存放文件名 第 5 章，“其他杂项模块”小节	Google Perftools*
gzip_buffers：定义用于存储被压缩的响应头的缓存数量和大小 第 5 章，“内容和编码”小节	Gzip
gzip_comp_level：定义对响应数据包的压缩级别 第 5 章，“内容和编码”小节	Gzip
gzip_disable：对于与指定的正则表达式匹配的用户代理 HTTP 请求头禁用 Gzip 压缩 第 5 章，“内容和编码”小节	Gzip
gzip_hash：设置分配给内部压缩状态(memLevel 参数)的内存总数 第 5 章，“内容和编码”小节	Gzip

gzip_http_version：对指定的 HTTP 协议版本使用 Gzip 压缩 第 5 章，“内容和编码”小节	Gzip
gzip_min_length：设置被 Gzip 压缩的最小响应数据包 第 5 章，“内容和编码”小节	Gzip
gzip_no_buffer：使用该指令会禁用 Gzip 对响应数据包的压缩 第 5 章，“内容和编码”小节	Gzip
gzip_proxied：启用或禁用 Gzip 对从代理收到的响应体进行压缩 第 5 章，“内容和编码”小节	Gzip
gzip_static：启用响应数据包预压缩模块 第 5 章，“内容和编码”小节	Gzip static*
gzip_types：设置能够被 Gzip 压缩的 MIME 类型 第 5 章，“内容和编码”小节	Gzip
gzip_vary：启用或禁用在响应数据包中包括 Vary HTTP 头 第 5 章，“内容和编码”小节	Gzip
gzip_window：为 Gzip 操作设置缓冲的窗口(windowBits argument) 第 5 章，“内容和编码”小节	Gzip
if_modified_since：定义 Nginx 如何处理 If-Modified-Since HTTP 头 第 4 章，“路径和文档”小节	HTTP Core
ignore_invalid_headers：假如请求头格式错误，如果禁用该指令，则返回“400 Bad Request HTTP error” 第 4 章，“客户端请求”小节	HTTP Core
image_filter：对图像应用转化 第 5 章，“内容和编码”小节	Image Filter*
image_filter_buffer：设置图像的最大值 第 5 章，“内容和编码”小节	Image Filter*

指令	模块

续表

指　令	模　块
gzip_http_version：对指定的 HTTP 协议版本使用 Gzip 压缩 第 5 章，“内容和编码”小节	Gzip
gzip_min_length：设置被 Gzip 压缩的最小响应数据包 第 5 章，“内容和编码”小节	Gzip
gzip_no_buffer：使用该指令会禁用 Gzip 对响应数据包的压缩 第 5 章，“内容和编码”小节	Gzip
gzip_proxied：启用或禁用 Gzip 对从代理收到的响应体进行压缩 第 5 章，“内容和编码”小节	Gzip
gzip_static：启用响应数据包预压缩模块 第 5 章，“内容和编码”小节	Gzip static*
gzip_types：设置能够被 Gzip 压缩的 MIME 类型 第 5 章，“内容和编码”小节	Gzip
gzip_vary：启用或禁用在响应数据包中包括 Vary HTTP 头 第 5 章，“内容和编码”小节	Gzip
gzip_window：为 Gzip 操作设置缓冲的窗口(windowBits argument) 第 5 章，“内容和编码”小节	Gzip
if_modified_since：定义 Nginx 如何处理 If-Modified-Since HTTP 头 第 4 章，“路径和文档”小节	HTTP Core
ignore_invalid_headers：假如请求头格式错误，如果禁用该指令，则返回“400 Bad Request HTTP error” 第 4 章，“客户端请求”小节	HTTP Core
image_filter：对图像应用转化 第 5 章，“内容和编码”小节	Image Filter*
image_filter_buffer：设置图像的最大值 第 5 章，“内容和编码”小节	Image Filter*

image_filter_jpeg_quality：设置 JPEG 图像过滤输出的质量 第 5 章，“内容和编码”小节	Image Filter*
include：包含外部的配置文件 第 3 章，“核心模块指令”小节	Core
index：设置目录的默认 index 第 5 章，“站点访问和日志记录”	Index
internal：约束 location 区段只能用于内部子请求和重定向 第 4 章，“限制和约束”小节	HTTP Core
keepalive_requests：置单个 keep-alive 的连接下提供的最大请求数量 第 4 章，“客户端请求”小节	HTTP Core
keepalive_timeout：关闭一个 keep-alive 连接之前 Nginx 等待的最长时间 第 4 章，“客户端请求”小节	HTTP Core
large_client_header_buffers：为客户端较大的请求头(larger header)设置缓冲值 第 4 章，“客户端请求”小节	HTTP Core
limit_conn：限制每个 zone 的连接总数 第 5 章，“限制和约束”小节	Limit Zone
limit_except：设置允许的 HTTP 方法 第 4 章，“限制和约束”小节	HTTP Core
limit_rate：限制每个连接的传输率 第 4 章，“限制和约束”小节	HTTP Core
limit_rate_after：超过指定的大小限制后，那么传输率就会受到限制 第 4 章，“限制和约束”小节	HTTP Core
limit_req：限制每个 zone 的连接总数 第 5 章，“限制和约束”小节	Limit Req
limit_req_zone：定义一个被 limit_req 使用的 zone 第 5 章，“限制和约束”小节	Limit Req

指令	模块

续表

指　令	模　块
image_filter_jpeg_quality：设置 JPEG 图像过滤输出的质量 第 5 章，“内容和编码”小节	Image Filter*
include：包含外部的配置文件 第 3 章，“核心模块指令”小节	Core
index：设置目录的默认 index 第 5 章，“站点访问和日志记录”	Index
internal：约束 location 区段只能用于内部子请求和重定向 第 4 章，“限制和约束”小节	HTTP Core
keepalive_requests：置单个 keep-alive 的连接下提供的最大请求数量 第 4 章，“客户端请求”小节	HTTP Core
keepalive_timeout：关闭一个 keep-alive 连接之前 Nginx 等待的最长时间 第 4 章，“客户端请求”小节	HTTP Core
large_client_header_buffers：为客户端较大的请求头(larger header)设置缓冲值 第 4 章，“客户端请求”小节	HTTP Core
limit_conn：限制每个 zone 的连接总数 第 5 章，“限制和约束”小节	Limit Zone
limit_except：设置允许的 HTTP 方法 第 4 章，“限制和约束”小节	HTTP Core
limit_rate：限制每个连接的传输率 第 4 章，“限制和约束”小节	HTTP Core
limit_rate_after：超过指定的大小限制后，那么传输率就会受到限制 第 4 章，“限制和约束”小节	HTTP Core
limit_req：限制每个 zone 的连接总数 第 5 章，“限制和约束”小节	Limit Req
limit_req_zone：定义一个被 limit_req 使用的 zone 第 5 章，“限制和约束”小节	Limit Req

limit_zone：定义一个被 limit_conn 使用的 zone 第 5 章，“限制和约束”小节	Limit Zone
lingering_time：当客户提交了一个超过最大允许的值时，通过该指令定义其处理行为 第 4 章，“客户端请求”小节	HTTP Core
lingering_timeout：设定在关闭客户端连接之前，对于两个读操作之间的间隔，Nginx 应该等待的时间总数 第 4 章，“客户端请求”小节	HTTP Core
listen：指定监听的套接字端口 第 4 章，“套接字和主机配置”小节	HTTP Core
lock_file：设置 lock 文件的路径 第 3 章，“内核模块指令”小节	Core
log_format：定义记录访问日志条目格式 第 5 章，“站点的访问和日志记录”小节	Log
log_not_found：启用或禁用记录 404 错误 第 4 章，“其他指令”小节	HTTP Core
log_subrequest：启用或禁用在日志文件中记录子请求 第 4 章，“其他指令”小节	HTTP Core
map：定义一个 map 值用来匹配一个变量，结果会被存储在其他的变量中 第 5 章，“与访问者相关”小节	Map
map_hash_bucket_size：设置 map 条目的最大值 第 5 章，“与访问者相关”小节	Map
map_hash_max_size：设置 map 中条目的最大值 第 5 章，“与访问者相关”小节	Map
master_process：启用或禁用 master 主进程 第 3 章，“核心模块指令”	Core
memcached_buffer_size：设置 memcached 数据缓存大小 第 5 章，“内容和编码”小节	Memcached
memcached_connect_timeout：设置 memcached 连接超时值 第 5 章，“内容和编码”小节	Memcached
memcached_next_upstream：用于 memcached 的配置，设置用于切换到下一个 upstream server 的条件 第 5 章，“内容和编码”小节	Memcached

指令	模块

续表

指　令	模　块
limit_zone：定义一个被 limit_conn 使用的 zone 第 5 章，“限制和约束”小节	Limit Zone
lingering_time：当客户提交了一个超过最大允许的值时，通过该指令定义其处理行为 第 4 章，“客户端请求”小节	HTTP Core
lingering_timeout：设定在关闭客户端连接之前，对于两个读操作之间的间隔，Nginx 应该等待的时间总数 第 4 章，“客户端请求”小节	HTTP Core
listen：指定监听的套接字端口 第 4 章，“套接字和主机配置”小节	HTTP Core
lock_file：设置 lock 文件的路径 第 3 章，“内核模块指令”小节	Core
log_format：定义记录访问日志条目格式 第 5 章，“站点的访问和日志记录”小节	Log
log_not_found：启用或禁用记录 404 错误 第 4 章，“其他指令”小节	HTTP Core
log_subrequest：启用或禁用在日志文件中记录子请求 第 4 章，“其他指令”小节	HTTP Core
map：定义一个 map 值用来匹配一个变量，结果会被存储在其他的变量中 第 5 章，“与访问者相关”小节	Map
map_hash_bucket_size：设置 map 条目的最大值 第 5 章，“与访问者相关”小节	Map
map_hash_max_size：设置 map 中条目的最大值 第 5 章，“与访问者相关”小节	Map
master_process：启用或禁用 master 主进程 第 3 章，“核心模块指令”	Core
memcached_buffer_size：设置 memcached 数据缓存大小 第 5 章，“内容和编码”小节	Memcached
memcached_connect_timeout：设置 memcached 连接超时值 第 5 章，“内容和编码”小节	Memcached
memcached_next_upstream：用于 memcached 的配置，设置用于切换到下一个 upstream server 的条件 第 5 章，“内容和编码”小节	Memcached

memcached_pass：配置访问 memcached 第 5 章，“内容和编码”小节	Memcached
memcached_read_timeout：设置 memcached 读数据超时 第 5 章，“内容和编码”小节	Memcached
memcached_send_timeout：设置 memcached 写数据超时 第 5 章，“内容和编码”小节	Memcached
merge_slashes：在 URL 中开启或禁用合并双斜线“//” 第 4 章，“其他指令”小节	HTTP Core
min_delete_depth：当使用 DELETE 命令的进行操作的时候，该指令定义被删除文件或目录的最小 URI 深度 第 5 章，“其他杂项模块”小节	DAV*
modern_browser：如果请求的用户代理匹配指定的字符串，则会影响变量$modern_browser 的值 第 5 章，“与你的访问者相关”小节	Browser
modern_browser_value：设置 $modern_browser 的值 第 5 章，“与访问者相关”小节	Browser
msie_padding：针对 MSIE 浏览器开启填充功能 第 4 章，“其他指令”小节	HTTP Core
msie_refresh：用于为 MSIE 家族浏览器开启特殊重定向 第 4 章，“其他指令”小节	HTTP Core
multi_accept：启用或禁用从队列中立刻接受多个连接 第 3 章，“事件模块”小节	Events
open_file_cache：定义打开文件缓存存储设置 第 4 章，“文件处理和缓存”小节	HTTP Core
open_file_cache_errors：定义是否应该在打开的文件缓存存储中缓存文件错误 第 4 章，“文件处理和缓存”小节	HTTP Core

指令	模块

续表

指　令	模　块
memcached_pass：配置访问 memcached 第 5 章，“内容和编码”小节	Memcached
memcached_read_timeout：设置 memcached 读数据超时 第 5 章，“内容和编码”小节	Memcached
memcached_send_timeout：设置 memcached 写数据超时 第 5 章，“内容和编码”小节	Memcached
merge_slashes：在 URL 中开启或禁用合并双斜线“//” 第 4 章，“其他指令”小节	HTTP Core
min_delete_depth：当使用 DELETE 命令的进行操作的时候，该指令定义被删除文件或目录的最小 URI 深度 第 5 章，“其他杂项模块”小节	DAV*
modern_browser：如果请求的用户代理匹配指定的字符串，则会影响变量$modern_browser 的值 第 5 章，“与你的访问者相关”小节	Browser
modern_browser_value：设置 $modern_browser 的值 第 5 章，“与访问者相关”小节	Browser
msie_padding：针对 MSIE 浏览器开启填充功能 第 4 章，“其他指令”小节	HTTP Core
msie_refresh：用于为 MSIE 家族浏览器开启特殊重定向 第 4 章，“其他指令”小节	HTTP Core
multi_accept：启用或禁用从队列中立刻接受多个连接 第 3 章，“事件模块”小节	Events
open_file_cache：定义打开文件缓存存储设置 第 4 章，“文件处理和缓存”小节	HTTP Core
open_file_cache_errors：定义是否应该在打开的文件缓存存储中缓存文件错误 第 4 章，“文件处理和缓存”小节	HTTP Core

open_file_cache_min_uses：定义一个文件要被保留在缓存中最少被访问的次数 第 4 章，“文件处理和缓存”小节	HTTP Core
open_file_cache_valid：设置缓存的校验间隔 第 4 章，“文件处理和缓存”小节	HTTP Core
open_log_file_cache：为日志文件描述符配置缓存 第 5 章，“站点的访问和日志记录”小节	Log
override_charset：覆盖掉从 proxy 或 FastCGI 收到的文档的 charset 设置 第 5 章，“内容和编码”小节	Charset
pid：设置 pid 文件的路径 第 3 章，“核心模块指令”小节	Core
port_in_redirect：启用或禁用包括端口号的内部重定向 第 4 章，“套接字和主机配置”小节	HTTP Core
post_action：定义一个请求完成之后的动作。请求完成之后，Nginx 将会调用该 URI 第 4 章，“其他指令”小节	HTTP Core
postpone_gzipping：定义一个最小数据限制，达到这个限制就可以开始使用 Gzip 压缩 第 5 章，“内容和编码”小节	HTTP Core
postpone_output：推迟发送响应数据包，这个指令定义了在每一个数据包中发送数据的大小 第 4 章，“套接字和主机配置”小节	HTTP Core
proxy_buffer_size：设置后端服务器响应缓冲区大小 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_buffering：启用或禁用端服务器响应缓冲 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_buffers：设置用于后端服务器通讯的缓冲数量和大小 第 7 章，“缓存、缓冲和临时文件”小节	Proxy

指令	模块

续表

指　令	模　块
open_file_cache_min_uses：定义一个文件要被保留在缓存中最少被访问的次数 第 4 章，“文件处理和缓存”小节	HTTP Core
open_file_cache_valid：设置缓存的校验间隔 第 4 章，“文件处理和缓存”小节	HTTP Core
open_log_file_cache：为日志文件描述符配置缓存 第 5 章，“站点的访问和日志记录”小节	Log
override_charset：覆盖掉从 proxy 或 FastCGI 收到的文档的 charset 设置 第 5 章，“内容和编码”小节	Charset
pid：设置 pid 文件的路径 第 3 章，“核心模块指令”小节	Core
port_in_redirect：启用或禁用包括端口号的内部重定向 第 4 章，“套接字和主机配置”小节	HTTP Core
post_action：定义一个请求完成之后的动作。请求完成之后，Nginx 将会调用该 URI 第 4 章，“其他指令”小节	HTTP Core
postpone_gzipping：定义一个最小数据限制，达到这个限制就可以开始使用 Gzip 压缩 第 5 章，“内容和编码”小节	HTTP Core
postpone_output：推迟发送响应数据包，这个指令定义了在每一个数据包中发送数据的大小 第 4 章，“套接字和主机配置”小节	HTTP Core
proxy_buffer_size：设置后端服务器响应缓冲区大小 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_buffering：启用或禁用端服务器响应缓冲 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_buffers：设置用于后端服务器通讯的缓冲数量和大小 第 7 章，“缓存、缓冲和临时文件”小节	Proxy

proxy_busy_buffers_size：设置用于繁忙的后端服务器的缓冲大小 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache：定义代理缓存 zone 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_key：设置 key，用于缓存被代理的请求 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_methods：设置可被代理缓存的 HTTP 方法 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_min_uses：设置缓存的条目被使用的次数，达到这个数值则会受到保护而不会被重缓存中清除出去 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_path：为指定的 zone 配置代理缓存选项 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_use_stale：在某些环境下定义缓存的数据是否过时 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_valid：设置缓存特定响应码的有效期 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_connect_timeout：为连接后台服务器设置连接超时 第 7 章，“限制、超时设定和错误”小节	Proxy
proxy_headers_hash_bucket_size：设置代理头哈希表(proxy headers hash table)中最长的头名字 第 7 章，“其他指令”小节	Proxy
proxy_headers_hash_max_size：在代理头哈希表中设置条目的最大值 第 7 章，“其他指令”小节	Proxy
proxy_hide_header：跳过指定的 header，用于反向代理 第 7 章，“主要指令”小节	Proxy

指令	模块

续表

指　令	模　块
proxy_busy_buffers_size：设置用于繁忙的后端服务器的缓冲大小 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache：定义代理缓存 zone 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_key：设置 key，用于缓存被代理的请求 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_methods：设置可被代理缓存的 HTTP 方法 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_min_uses：设置缓存的条目被使用的次数，达到这个数值则会受到保护而不会被重缓存中清除出去 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_path：为指定的 zone 配置代理缓存选项 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_use_stale：在某些环境下定义缓存的数据是否过时 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_cache_valid：设置缓存特定响应码的有效期 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_connect_timeout：为连接后台服务器设置连接超时 第 7 章，“限制、超时设定和错误”小节	Proxy
proxy_headers_hash_bucket_size：设置代理头哈希表(proxy headers hash table)中最长的头名字 第 7 章，“其他指令”小节	Proxy
proxy_headers_hash_max_size：在代理头哈希表中设置条目的最大值 第 7 章，“其他指令”小节	Proxy
proxy_hide_header：跳过指定的 header，用于反向代理 第 7 章，“主要指令”小节	Proxy

proxy_ignore_client_abort：当客户端放弃请求时，设置代理模块的行为 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_ignore_headers：阻止 Nginx 处理指定的头 第 7 章，“其他指令”小节	Proxy
proxy_intercept_errors：定义是否后端服务器产生错误应该返回“as is” 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_max_temp_file_size：设置临时文件的最大值 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_method：允许设置转发至后台服务器的 HTTP 方法用 第 7 章，“主要指令”小节	Proxy
proxy_next_upstream：定义跳过 upstream server 的条件 第 7 章，“主要指令”小节	Proxy
proxy_pass：在指定的 location 中，使反向代理指向一个后端服务器 第 7 章，“主要指令”小节	Proxy
proxy_pass_header：对于反向代理，禁用跳过指定的头 第 7 章，“主要指令”小节	Proxy
proxy_pass_request_body：允许请求体被传递到后端服务器 第 7 章，“主要指令”小节	Proxy
proxy_pass_request_header：允许其他的请求头传递到后端服务器 第 7 章，“主要指令”小节	Proxy
proxy_read_timeout：为后台服务器通信设置读超时 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_redirect：启用或禁用由后台服务器产生的重定向 第 7 章，“主要指令”小节	Proxy
proxy_send_lowat：对于 TCP 套接字通讯，在与后端服务器通信中允许你使用 SO_SNDLOWAT 标志，仅在 FreeBSD 下有效 第 7 章，“缓存、缓冲和临时文件”小节	Proxy

指　令	模　块

续表

指　令	模　块
proxy_ignore_client_abort：当客户端放弃请求时，设置代理模块的行为 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_ignore_headers：阻止 Nginx 处理指定的头 第 7 章，“其他指令”小节	Proxy
proxy_intercept_errors：定义是否后端服务器产生错误应该返回“as is” 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_max_temp_file_size：设置临时文件的最大值 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_method：允许设置转发至后台服务器的 HTTP 方法用 第 7 章，“主要指令”小节	Proxy
proxy_next_upstream：定义跳过 upstream server 的条件 第 7 章，“主要指令”小节	Proxy
proxy_pass：在指定的 location 中，使反向代理指向一个后端服务器 第 7 章，“主要指令”小节	Proxy
proxy_pass_header：对于反向代理，禁用跳过指定的头 第 7 章，“主要指令”小节	Proxy
proxy_pass_request_body：允许请求体被传递到后端服务器 第 7 章，“主要指令”小节	Proxy
proxy_pass_request_header：允许其他的请求头传递到后端服务器 第 7 章，“主要指令”小节	Proxy
proxy_read_timeout：为后台服务器通信设置读超时 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_redirect：启用或禁用由后台服务器产生的重定向 第 7 章，“主要指令”小节	Proxy
proxy_send_lowat：对于 TCP 套接字通讯，在与后端服务器通信中允许你使用 SO_SNDLOWAT 标志，仅在 FreeBSD 下有效 第 7 章，“缓存、缓冲和临时文件”小节	Proxy

proxy_send_timeout：为后端服务器设置写超时 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_set_body：设置请求体，用于调试目的 第 7 章，“主要指令”小节	Proxy
proxy_set_header：设置额外的头数据，用于调试目的 第 7 章，“其他指令”小节	Proxy
proxy_store：为每个代理通讯开启缓存存储 第 7 章，“其他指令”小节	Proxy
proxy_store_access：对缓存设置访问权限 第 7 章，“其他指令”小节	Proxy
Proxy_temp_file_write_size：当写临时文件的时候，设置缓存的大小 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_temp_path：设置用于代理的临时文件路径 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
random_index：启用或禁用选择一个随机的文件作为该目录的 index 第 5 章，“站点的访问和日志记录”小节	Random Index*
real_ip_header：设置 HTTP 头，用于替代 IP 地址 第 5 章，“与你的访问者相关”小节	Real IP*
recursive_error_pages：启用或禁用使用递归错误网页(使用 error_page 指定) 第 4 章，“路径和文档”小节	HTTP Core
request_pool_size：定义分配给请求的内存池空间(pool memory space) 第 4 章，“套接字和主机配置”小节	HTTP Core

指令	模块

续表

指　　令	模　　块
proxy_send_timeout：为后端服务器设置写超时 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_set_body：设置请求体，用于调试目的 第 7 章，“主要指令”小节	Proxy
proxy_set_header：设置额外的头数据，用于调试目的 第 7 章，“其他指令”小节	Proxy
proxy_store：为每个代理通讯开启缓存存储 第 7 章，“其他指令”小节	Proxy
proxy_store_access：对缓存设置访问权限 第 7 章，“其他指令”小节	Proxy
Proxy_temp_file_write_size：当写临时文件的时候，设置缓存的大小 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
proxy_temp_path：设置用于代理的临时文件路径 第 7 章，“缓存、缓冲和临时文件”小节	Proxy
random_index：启用或禁用选择一个随机的文件作为该目录的 index 第 5 章，“站点的访问和日志记录”小节	Random Index*
real_ip_header：设置 HTTP 头，用于替代 IP 地址 第 5 章，“与你的访问者相关”小节	Real IP*
recursive_error_pages：启用或禁用使用递归错误网页(使用 error_page 指定) 第 4 章，“路径和文档”小节	HTTP Core
request_pool_size：定义分配给请求的内存池空间(pool memory space) 第 4 章，“套接字和主机配置”小节	HTTP Core

reset_timedout_connection：启用或禁用连接超时后删除连接信息 第 4 章，“套接字和主机配置”小节	HTTP Core
resolver：设置 DNS 的 IP 地址 第 4 章，“其他指令”小节	HTTP Core
resolver_timeout：设置解析主机名超时 第 4 章，“其他指令”小节	HTTP Core
return：中断请求并且返回指定的代码 第 5 章，“Rewrite 模块指令”小节	Rewrite
rewrite：改写 URL 第 5 章，“Rewrite 模块指令”小节	Rewrite
rewrite_log：启用或禁用 rewrite 在 notice 的日志级别上发出日志消息 第 5 章，“Rewrite 模块指令”小节	Rewrite
root：为虚拟主机或虚拟路径设置文档的根目录 第 4 章，“路径和文档”小节	HTTP Core
satisfy：定义访问资源的条件 第 4 章，“限制和约束”小节	HTTP Core
secure_link_secret：为安全 URL 设置密语 第 5 章，“SSL 和安全”小节	Secure Link*
send_lowat：在 BSD 系统下启用或禁用 TCP 套接字标志 SO_SNDLOWA，用于同客户端通讯 第 4 章，“套接字和主机配置”小节	HTTP Core
send_timeout：在上一个数据包被收到之后，在 Nginx 关闭一个客户端连接之前的这个时间值 第 4 章，“客户端请求”小节	HTTP Core
sendfile：启用或禁用调用 sendfile 内核处理文件传输 第 4 章，“套接字和主机配置”小节	HTTP Core
sendfile_max_chunk：每一个被 sendfile 处理的数据块的最大值 第 4 章，“套接字和主机配置”小节	HTTP Core
server：在 upstream 区段声明一个 server 条目 第 6 章，“Upstream 块”小节	Upstream

指令	模块

续表

指　令	模　块
reset_timedout_connection：启用或禁用连接超时后删除连接信息 第 4 章，“套接字和主机配置”小节	HTTP Core
resolver：设置 DNS 的 IP 地址 第 4 章，“其他指令”小节	HTTP Core
resolver_timeout：设置解析主机名超时 第 4 章，“其他指令”小节	HTTP Core
return：中断请求并且返回指定的代码 第 5 章，“Rewrite 模块指令”小节	Rewrite
rewrite：改写 URL 第 5 章，“Rewrite 模块指令”小节	Rewrite
rewrite_log：启用或禁用 rewrite 在 notice 的日志级别上发出日志消息 第 5 章，“Rewrite 模块指令”小节	Rewrite
root：为虚拟主机或虚拟路径设置文档的根目录 第 4 章，“路径和文档”小节	HTTP Core
satisfy：定义访问资源的条件 第 4 章，“限制和约束”小节	HTTP Core
secure_link_secret：为安全 URL 设置密语 第 5 章，“SSL 和安全”小节	Secure Link*
send_lowat：在 BSD 系统下启用或禁用 TCP 套接字标志 SO_SNDLOWA，用于同客户端通讯 第 4 章，“套接字和主机配置”小节	HTTP Core
send_timeout：在上一个数据包被收到之后，在 Nginx 关闭一个客户端连接之前的这个时间值 第 4 章，“客户端请求”小节	HTTP Core
sendfile：启用或禁用调用 sendfile 内核处理文件传输 第 4 章，“套接字和主机配置”小节	HTTP Core
sendfile_max_chunk：每一个被 sendfile 处理的数据块的最大值 第 4 章，“套接字和主机配置”小节	HTTP Core
server：在 upstream 区段声明一个 server 条目 第 6 章，“Upstream 块”小节	Upstream

server_name：设置虚拟主机的名字 第 4 章，“套接字和主机配置”小节	HTTP Core
server_name_in_redirect：启用或禁用服务器名字用于内部重定向 第 4 章，“套接字和主机配置”小节	HTTP Core
server_names_hash_bucket_size：设置在哈希表中服务器名字的最大值 第 4 章，“套接字和主机配置”小节	HTTP Core
server_names_hash_max_size：设置在哈希表中服务器名字的最大数量 第 4 章，“套接字和主机配置”小节	HTTP Core
server_tokens：启用或禁用显示 Nginx 的版本号信息 第 4 章，“其他指令”小节	HTTP Core
set：设置变量的值 第 5 章，“Rewrite 模块指令”小节	Rewrite
set_real_ip_from：通过声明 IP 地址来定义一个受信任的服务器 第 5 章，“与你的访问者相关”小节	Real IP*
source_charset：为文档设置初始的编码方式 第 5 章，“内容和编码”小节	Charset
ssi：激活服务器端包含 第 5 章，“SSI 模块指令和变量”小节	SSI
ssi_ignore_recycled_buffers：阻止 Nginx 使用循环内存 第 5 章，“SSI 模块指令和变量”小节	SSI
ssi_min_file_chunk：定义用于 SSI 请求的缓冲和存储设置 第 5 章，“SSI 模块指令和变量”小节	SSI
ssi_silent_errors：定义对 SSI 错误是否保持沉默(silent) 第 5 章，“SSI 模块指令和变量”小节	SSI
ssi_types：定义 SSI 能够解析的 MIME 类型 第 5 章，“SSI 模块指令和变量”小节	SSI

指令	模块

续表

指　令	模　块
server_name：设置虚拟主机的名字 第 4 章，“套接字和主机配置”小节	HTTP Core
server_name_in_redirect：启用或禁用服务器名字用于内部重定向 第 4 章，“套接字和主机配置”小节	HTTP Core
server_names_hash_bucket_size：设置在哈希表中服务器名字的最大值 第 4 章，“套接字和主机配置”小节	HTTP Core
server_names_hash_max_size：设置在哈希表中服务器名字的最大数量 第 4 章，“套接字和主机配置”小节	HTTP Core
server_tokens：启用或禁用显示 Nginx 的版本号信息 第 4 章，“其他指令”小节	HTTP Core
set：设置变量的值 第 5 章，“Rewrite 模块指令”小节	Rewrite
set_real_ip_from：通过声明 IP 地址来定义一个受信任的服务器 第 5 章，“与你的访问者相关”小节	Real IP*
source_charset：为文档设置初始的编码方式 第 5 章，“内容和编码”小节	Charset
ssi：激活服务器端包含 第 5 章，“SSI 模块指令和变量”小节	SSI
ssi_ignore_recycled_buffers：阻止 Nginx 使用循环内存 第 5 章，“SSI 模块指令和变量”小节	SSI
ssi_min_file_chunk：定义用于 SSI 请求的缓冲和存储设置 第 5 章，“SSI 模块指令和变量”小节	SSI
ssi_silent_errors：定义对 SSI 错误是否保持沉默(silent) 第 5 章，“SSI 模块指令和变量”小节	SSI
ssi_types：定义 SSI 能够解析的 MIME 类型 第 5 章，“SSI 模块指令和变量”小节	SSI

ssi_value_length：定义 SSI 的 tag 值最大长度 第 5 章，“SSI 模块指令和变量”小节	SSI
ssl：为虚拟主机提供 HTTPS 第 5 章，“SSL 和安全”小节	SSL*
ssl_certificate：设置 PEM 证书文件的路径 第 5 章，“SSL 和安全”小节	SSL*
ssl_certificate_key：设置密钥文件的路径 第 5 章，“SSL 和安全”小节	SSL*
ssl_ciphers：设置被 SSL 引擎使用的密码 第 5 章，“SSL 和安全”小节	SSL*
ssl_client_certificate：设置客户端 PEM 证书文件的路径 第 5 章，“SSL 和安全”小节	SSL*
ssl_dhparam：设置 DH 文件的路径 第 5 章，“SSL 和安全”小节	SSL*
ssl_engine：指定想要使用的 SSL 引擎名字 第 3 章，“核心模块指令”小节	Core
ssl_prefer_server_ciphers：指定是否服务器密码优先于客户端密码 第 5 章，“SSL 和安全”小节	SSL*
ssl_protocols：设置被 SSL 引擎使用的协议 第 5 章，“SSL 和安全”小节	SSL*
ssl_session_cache：配置 SSL 会话缓存设置 第 5 章，“SSL 和安全”小节	SSL*
ssl_session_timeout：为 SSL 会话设置超时 第 5 章，“SSL 和安全”小节	SSL*
ssl_verify_client：启用或禁用校验客户端证书 第 5 章，“SSL 和安全”小节	SSL*
ssl_verify_depth：设置证书的校验深度 第 5 章，“SSL 和安全”小节	SSL*
stub_status：开启或禁用 stub 状态信息 第 5 章，“其他杂项模块”	Stub Status*

指令	模块

续表

指　令	模　块
ssi_value_length：定义 SSI 的 tag 值最大长度 第 5 章，“SSI 模块指令和变量”小节	SSI
ssl：为虚拟主机提供 HTTPS 第 5 章，“SSL 和安全”小节	SSL*
ssl_certificate：设置 PEM 证书文件的路径 第 5 章，“SSL 和安全”小节	SSL*
ssl_certificate_key：设置密钥文件的路径 第 5 章，“SSL 和安全”小节	SSL*
ssl_ciphers：设置被 SSL 引擎使用的密码 第 5 章，“SSL 和安全”小节	SSL*
ssl_client_certificate：设置客户端 PEM 证书文件的路径 第 5 章，“SSL 和安全”小节	SSL*
ssl_dhparam：设置 DH 文件的路径 第 5 章，“SSL 和安全”小节	SSL*
ssl_engine：指定想要使用的 SSL 引擎名字 第 3 章，“核心模块指令”小节	Core
ssl_prefer_server_ciphers：指定是否服务器密码优先于客户端密码 第 5 章，“SSL 和安全”小节	SSL*
ssl_protocols：设置被 SSL 引擎使用的协议 第 5 章，“SSL 和安全”小节	SSL*
ssl_session_cache：配置 SSL 会话缓存设置 第 5 章，“SSL 和安全”小节	SSL*
ssl_session_timeout：为 SSL 会话设置超时 第 5 章，“SSL 和安全”小节	SSL*
ssl_verify_client：启用或禁用校验客户端证书 第 5 章，“SSL 和安全”小节	SSL*
ssl_verify_depth：设置证书的校验深度 第 5 章，“SSL 和安全”小节	SSL*
stub_status：开启或禁用 stub 状态信息 第 5 章，“其他杂项模块”	Stub Status*

sub_filter：在响应头中搜索和替换文本 第 5 章，“内容和编码”小节	Substitution*
sub_filter_once： 定义 sub_filter 是否搜索和替换仅一次后就停止 第 5 章，“内容和编码”小节	Substitution*
sub_filter_types：定义可用于搜索和替换的 MIME 类型 第 5 章，“内容和编码”小节	Substitution*
tcp_nodelay：在 keep-alive 连接中启用或禁用 TCP_NODELAY 选项 第 4 章，“套接字和主机配置”小节	HTTP Core
tcp_nopush：在 keep-alive 连接中，启用和禁用 TCP_NOPUSH (BSD) 或 TCP_CORK (Linux)套接字选项 第 4 章，“套接字和主机配置”小节	HTTP Core
thread_stack_size：设置线程堆栈的大小 第 3 章，“核心模块指令”小节	Core
timer_resolution：内部时钟同步间隔 第 3 章，“核心模块指令”小节	Core
try_files：如果试图访问的文件没有找到，那么跳到一个命名块 第 4 章，“路径和文档”小节	HTTP Core
types：与文件扩展名匹配的 MIME 类型 第 4 章，“MIME 类型”小节	HTTP Core
types_hash_bucket_size： 在 MIME 类型哈希表定义一个条目的最大值 第 4 章，“MIME 类型”小节	HTTP Core
types_hash_max_size：定义 MIME 类型哈希表中最大的条目数 第 4 章，“MIME 类型”小节	HTTP Core
underscores_in_headers：在 HTTP 头名字中启用或禁用下划线 第 4 章，“其他指令”小节	HTTP Core

指令	模块

续表

指　令	模　块
sub_filter：在响应头中搜索和替换文本 第 5 章，“内容和编码”小节	Substitution*
sub_filter_once： 定义 sub_filter 是否搜索和替换仅一次后就停止 第 5 章，“内容和编码”小节	Substitution*
sub_filter_types：定义可用于搜索和替换的 MIME 类型 第 5 章，“内容和编码”小节	Substitution*
tcp_nodelay：在 keep-alive 连接中启用或禁用 TCP_NODELAY 选项 第 4 章，“套接字和主机配置”小节	HTTP Core
tcp_nopush：在 keep-alive 连接中，启用和禁用 TCP_NOPUSH (BSD) 或 TCP_CORK (Linux)套接字选项 第 4 章，“套接字和主机配置”小节	HTTP Core
thread_stack_size：设置线程堆栈的大小 第 3 章，“核心模块指令”小节	Core
timer_resolution：内部时钟同步间隔 第 3 章，“核心模块指令”小节	Core
try_files：如果试图访问的文件没有找到，那么跳到一个命名块 第 4 章，“路径和文档”小节	HTTP Core
types：与文件扩展名匹配的 MIME 类型 第 4 章，“MIME 类型”小节	HTTP Core
types_hash_bucket_size： 在 MIME 类型哈希表定义一个条目的最大值 第 4 章，“MIME 类型”小节	HTTP Core
types_hash_max_size：定义 MIME 类型哈希表中最大的条目数 第 4 章，“MIME 类型”小节	HTTP Core
underscores_in_headers：在 HTTP 头名字中启用或禁用下划线 第 4 章，“其他指令”小节	HTTP Core

uninitialized_variable_warn：启用或禁用记录使用没有被初始化的变量的指令 第 5 章，“Rewrite 模块指令”小节	Rewrite
upstream：定义一个 upstream 区段，用于负载均衡结构 第 6 章，“Upstream 块”小节	Upstream
use：设置首选的事件模型 第 3 章，“事件模块”小节	Events
user：设置运行 worker 进程的用户和组 第 3 章，“核心模块指令”小节	Core
userid：启用或禁用 userid 模块 第 5 章，“与你的访问者相关”小节	User ID
userid_domain：设置分配 cookie 的域 第 5 章，“与你的访问者相关”小节	User ID
userid_expires：设置 cookie 生存期 第 5 章，“与你的访问者相关”小节	User ID
userid_name：设置分配的 cookie 名称 第 5 章，“与你的访问者相关”小节	User ID
userid_p3p：设置 p3p 头的值 第 5 章，“与你的访问者相关”小节	User ID
userid_path：设置 cookie 的路径小节 第 5 章，“与你的访问者相关”小节	User ID
userid_service：该指令用于定义发布 cookie 服务器的 IP 地址 第 5 章，“与你的访问者相关”小节	User ID
valid_referers： 用于在 location 中定义 referrer 第 5 章，“与你的访问者相关”小节	Referrer
variables_hash_bucket_size：定义变量哈希表中的最大的变量长度 第 4 章，“其他指令”小节	HTTP Core
variables_hash_max_size：定义变量哈希表的最大值 第 4 章，“其他指令”小节	HTTP Core

指令	模块

续表

指　令	模　块
uninitialized_variable_warn：启用或禁用记录使用没有被初始化的变量的指令 第 5 章，“Rewrite 模块指令”小节	Rewrite
upstream：定义一个 upstream 区段，用于负载均衡结构 第 6 章，“Upstream 块”小节	Upstream
use：设置首选的事件模型 第 3 章，“事件模块”小节	Events
user：设置运行 worker 进程的用户和组 第 3 章，“核心模块指令”小节	Core
userid：启用或禁用 userid 模块 第 5 章，“与你的访问者相关”小节	User ID
userid_domain：设置分配 cookie 的域 第 5 章，“与你的访问者相关”小节	User ID
userid_expires：设置 cookie 生存期 第 5 章，“与你的访问者相关”小节	User ID
userid_name：设置分配的 cookie 名称 第 5 章，“与你的访问者相关”小节	User ID
userid_p3p：设置 p3p 头的值 第 5 章，“与你的访问者相关”小节	User ID
userid_path：设置 cookie 的路径小节 第 5 章，“与你的访问者相关”小节	User ID
userid_service：该指令用于定义发布 cookie 服务器的 IP 地址 第 5 章，“与你的访问者相关”小节	User ID
valid_referers： 用于在 location 中定义 referrer 第 5 章，“与你的访问者相关”小节	Referrer
variables_hash_bucket_size：定义变量哈希表中的最大的变量长度 第 4 章，“其他指令”小节	HTTP Core
variables_hash_max_size：定义变量哈希表的最大值 第 4 章，“其他指令”小节	HTTP Core

worker_connections：定义每个 worker 进程同时处理的连接数 第 3 章，“事件模块”小节	Events
worker_cpu_affinity：定义 worker 进程和 CPU 内核的亲和力 第 3 章，“核心模块指令”小节	Core
worker_priority：设置 worker 进程的优先级 第 3 章，“核心模块指令”小节	Core
worker_processes：设置 worker 进程的数量 第 3 章，“核心模块指令”小节	Core
worker_rlimit_core：为每个 worker 进程设置核心文件的大小 第 3 章，“核心模块指令”小节	Core
worker_rlimit_nofile：设置一个 worker 进程能够同时使用文件的总数 第 3 章，“核心模块指令”小节	Core
worker_rlimit_sigpending：定义一个 worker 进程在队列中能够容纳信号(signal)的总数 第 3 章，“核心模块指令”小节	Core
worker_threads：启用线程不推荐) 第 3 章，“核心模块指令”小节	Core
working_directory：为 worker 进程设置工作目录 第 3 章，“核心模块指令”小节	Core
xml_entities：指定包含象征性元素定义的 DTD 文件 第 5 章，“内容和编码”小节	XSLT*
xslt_stylesheet：指定 XSLT 模板文件的路径 第 5 章，“内容和编码”小节	XSLT*
xslt_types：设置 MIME 类型，可以被应用于转化的类型 第 5 章，“内容和编码”小节	XSLT*

指令	模块

续表

指　　令	模　　块
worker_connections：定义每个 worker 进程同时处理的连接数 第 3 章，“事件模块”小节	Events
worker_cpu_affinity：定义 worker 进程和 CPU 内核的亲和力 第 3 章，“核心模块指令”小节	Core
worker_priority：设置 worker 进程的优先级 第 3 章，“核心模块指令”小节	Core
worker_processes：设置 worker 进程的数量 第 3 章，“核心模块指令”小节	Core
worker_rlimit_core：为每个 worker 进程设置核心文件的大小 第 3 章，“核心模块指令”小节	Core
worker_rlimit_nofile：设置一个 worker 进程能够同时使用文件的总数 第 3 章，“核心模块指令”小节	Core
worker_rlimit_sigpending：定义一个 worker 进程在队列中能够容纳信号(signal)的总数 第 3 章，“核心模块指令”小节	Core
worker_threads：启用线程不推荐) 第 3 章，“核心模块指令”小节	Core
working_directory：为 worker 进程设置工作目录 第 3 章，“核心模块指令”小节	Core
xml_entities：指定包含象征性元素定义的 DTD 文件 第 5 章，“内容和编码”小节	XSLT*
xslt_stylesheet：指定 XSLT 模板文件的路径 第 5 章，“内容和编码”小节	XSLT*
xslt_types：设置 MIME 类型，可以被应用于转化的类型 第 5 章，“内容和编码”小节	XSLT*

附录 B
模 块 参 考

本附录总结了 Nginx 稳定版 0.7.66 的有效模块，对于每一个模块，只简单描述一下，也提供一些具体的介绍，并指出在哪一章能够找到更详细的信息。模块的列举按照字母顺序。

标记有*的模块为可选模块，建立 Nginx 的时候，如果没有额外使用配置开关项指出，那么它将不会被编译在 Nginx 中。启用或禁用模块的相应配置开关将在每个模块中详细讨论。

Access

设置允许或拒绝访问资源，它基于 IP 地址或地址范围。

关键指令：allow, deny

配置开关项：--without-http_access_module 禁用该模块。

第 5 章，“限制和约束”小节

Addition*

允许你指定在响应体之前或之后添加的内容。

关键指令：add_before_body, add_after_body

配置开关项：--with-http_addition_module 启用该模块

第 5 章，“内容和编码”小节

Auth_basic module

允许你在指定位置中设置基本认证。

关键指令：auth_basic, auth_basic_user_file

配置开关项：--without-http_auth_basic_module 启用该模块。

第 5 章，“限制和约束”小节

Autoindex

在没有索引文件的情况下，允许自动生成目录的文件列表。

关键指令：autoindex

配置开关项：--without-http_autoindex_module 禁用该模块。

第 5 章，“站点访问和日志记录”小节

Browser

解析 User-Agent HTTP 头，在解析的结果中产生变量。

关键指令：modern_browser, ancient_browser

配置开关项：--without-http_browser_module 禁用该模块。

第 5 章，“与访问者相关”小节

Charset

提供重新对网页内容编码的功能。

关键指令：charset, override_charset

配置开关项：--without-http_charset_module 禁用该模块

第 5 章，“内容和编码”小节

Core

提供内核功能，例如 守护进程和套接字处理。

关键指令：worker_processes, user

配置开关项：该模块默认激活而且不能禁用。

第 3 章，“核心模块指令”小节

DAV*

开启 WebDAV (Web-based Distributed Authoring and Versioning) 支持。

关键指令：dav_methods, dav_access

配置开关项：--with-http_dav_module 启用本模块。

第 5 章，“其他杂项模块”小节

Empty GIF

允许直接从内存提供一个空白的 GIF 文件。

关键指令：empty_gif

配置开关项：--without-http_empty_gif_module 禁用该模块。

第 5 章，“内容和编码”小节

Events

允许你选择和配置连接事件模型。

关键指令：worker_connections

配置开关项：该模块默认激活而且不能禁用。

第 3 章，“Events 模块”小节

FastCGI

开启 FastCGI 支持。

关键指令：fastcgi_pass, fastcgi_param

配置开关项：--without-http_fastcgi_module 禁用该模块。

第 6 章，“FastCGI 模块”小节

FLV*

支持 FLV 文件。

关键指令：flv

配置开关项：--with-http_flv_module 启用本模块。

第 5 章，“内容和编码”小节

Geo

影响一个类似于 map 指令的值，受影响的是一个 IP 或者是 IP 范围。

关键指令：geo

配置开关项：--without-http_geo_module 禁用该模块。

第 5 章，“与访问者相关”小节

Geo IP*

支持 MaxMind's GeoIP 数据库。

关键指令：geoip_country, geoip_city

配置开关项：--with-http_geoip_module 启用本模块。

第 5 章，“与访问者相关”小节

Google-perftools*

支持 Google 性能分析工具。

关键指令：google_perftools_profiles

配置开关项：--with-google_perftools_module 启用本模块。

第 5 章，“其他杂项模块”小节

Gzip

允许使用 Gzip 压缩算法压缩响应体。

关键指令：gzip, gzip_comp_level

配置开关项：--without-http_gzip_module 禁用该模块。

第 5 章，“内容和编码”小节

Gzip Static*

开启预压缩响应文件检测。

关键指令：gzip_static

配置开关项：--with-http_gzip_static_module 启用该模块。

第 5 章，“内容和编码”小节

Headers

允许定义专用的 HTTP 响应头。

关键指令：add_header, expires

配置开关项：该模块默认激活而且不能禁用。

第 5 章，“内容和编码”小节

HTTP Core

提供核心的 HTTP 功能。

关键指令：listen, server_name, 等等。

配置开关项：--without-http 禁用所有与 HTTP 有关的功能。

第 4 章，“HTTP 核心模块”小节

Image Filter*

通过 GDLib 提供图像转换功能。

关键指令：image_filter

配置开关项：--with-http_image_filter_module 启用该模块。

第 5 章，“内容和编码”小节

Index

允许定义一个文件，作为本目录的 index。

关键指令：index

配置开关项：该模块默认激活而且不能禁用。

第 5 章，“站点访问和日志记录”小节

Limit Requests

允许对定义的 zone 限制请求。

关键指令：limit_req, limit_req_zone

配置开关项：--without-http_limit_req_module 禁用该模块。

第 5 章，“限制和约束”小节

Limit Zone

允许对定义的 zone 限制连接。

关键指令：limit_zone, limit_conn

配置开关项：--without-http_limit_zone_module 禁用该模块。

第 5 章，“限制和约束”小节

Log

提供访问日志的转换功能。

关键指令：access_log, log_format

配置开关项：该模块默认激活而且不能禁用。

第 5 章，“站点访问和日志记录”小节

Map

影响建立在键和值的映射(map)变量。

关键指令：map

配置开关项：--without-http_map_module 禁用该模块。

第 5 章，“与访问者相关”小节

Memcached

提供用于同 memcached 交互的指令（内存缓存守护进程)。

关键指令：memcached_pass

配置开关项：--without-http_memcached_module 禁用该模块。

第 5 章，“内容和编码”小节

Proxy

提供反向代理功能。

关键指令：proxy_pass, proxy_set_header，等等

配置开关项：--without-http_proxy_module 禁用该模块。

第 7 章，“代理模块”小节

Random index*

允许选择一个随机文件作为本目录的 index。

关键指令：random_index

配置开关项：--with-http_random_index_module 启用本模块。

第 5 章，“站点访问和日志记录”小节

Real IP*

当使用 Nginx 作为后端服务器时，允许从头中(header)检索真正的客户端 IP 地址。

关键指令：set_real_ip_from, real_ip_header

配置开关项：--with-http_realip_module 启用本模块。

第 5 章，“与访问者相关”小节

Referer

允许建立一个 HTTP 白名单(whitelist)。

关键指令：valid_referers

配置开关项：--without-http_referer_module 禁用该模块。

第 5 章，“与访问者相关”小节

Rewrite

提供 URL 重定向功能。

关键指令：rewrite, if, return, break,等等

配置开关项：--without-http_rewrite_module 禁用该模块。

第 5 章，“Rewrite 模块”小节。

Secure Link*

在 URL 位置中提供基于哈希值的链路验证。

关键指令：secure_link_secret

配置开关项：--with-http_secure_link_module 启用本模块。

第 5 章，“SSL 和安全”小节

SSI

提供服务器端包含功能。

关键指令：ssi, ssi_types

配置开关项：--without-http_ssi_module 禁用该模块。

第 5 章，“SSI 模块”小节

SSL*

提供 HTTP SSL 支持。

关键指令：ssl, ssl_certificate, 等等

配置开关项：--with-http_ssl_module 启用本模块。

第 5 章，“SSL 和安全”小节

Stub status*

提供服务状态信息功能。

关键指令：stub_status

配置开关项：--with-http_stub_status_module 启用本模块。

第 5 章，“其他杂项模块”小节

Substitution*

允许在 web 页中替代内容。

关键指令：sub_filter, sub_filter_once

配置开关项：--with-http_sub_module 启用本模块。

第 5 章，“内容和编码”小节

Upstream

允许设置负载均衡结构。

关键指令：upstream, server

配置开关项：--without-http_upstream_ip_hash_module 只是禁用了 ip_hash 指令。Upstream 模块默认自身被包含在内，不能够被禁用。

第 6 章，“Upstream 模块”小节

User ID

允许设置 cookie 来标识访问者。

关键指令：userid, userid_domain

配置开关项：--without-http_userid_module 禁用该模块。

第 5 章，“与访问者相关”小节

XSLT*

允许在响应体中应用 XSLT 模板。

关键指令：xslt_stylesheet, xml_entities

配置开关项：--with-http_xslt_module 启用本模块。

第 5 章，“内容和编码”小节

附录 C

疑 难 解 答

即使你一字不漏地读完全书，遗憾的是，你仍然会碰到各种各样的问题，从简单的配置错误到一个或者多个模块偶尔出现的异常行为。在这个附录中，我们试图为 Nginx 新手管理员提供一些常见问题的解决方法。

本附录主题

- 一个基本指导，包含一般故障排除技巧；
- 如何解决最常见的一些安装问题；
- 处理“403 Forbidden HTTP 错误”；
- 为什么你的配置不能正确应用；
- 关于 if 块的行为。

处理一般性问题

在我们开始之前，只要遇到 Nginx 问题，就应该确保依照以下的建议，因为它们通常能够提供很好的解决方案。

检查访问权限

Nginx 管理员遇到的许多错误是由于非法访问权限引起的。在两个阶段，可以指定运行 Nginx 的用户和组：

- 在安装配置时，通过 configure 命令指定一个默认用户和组(参见第 2 章)。
- 在配置文件中，使用 user 指令来指定用户和组，这个指令会覆盖你在 configure 时指定的值。

如果 Nginx 期望访问的文件没有正确的权限，换句话说，Nginx 在指定的用户和组的权限下不能够读取文件(由此延伸不能写，例如，用于保存临时文件的目录)，在这些情况下，Nginx 不能正确提供文件，所以要检查文件权限。

测试配置文件

一个常见的错误是管理员经常自信地修改配置文件中的一些内容却经常不保留备份，然后就重新载入 Nginx，以便应用新的配置文件，如果配置文件中包含语法或语义的错误，那么应用程序将拒绝重新载入。更糟糕的是，在重新启动 Nginx 服务器后，如果 Nginx 停止，它将拒绝启动。在所有情况下，记住下面的忠告：

- 总是保存一个使用中的配置文件的备份；
- 在重新载入或重启动 Nginx 之前，测试一下配置文件，使用命令 nginx -t 来测试当前的配置文件，或者使用 nginx -t -c/path/to/config/file.conf 来测试指定的配置文件；
- 使用重新载入来替代重新启动服务——宁愿选择“service nginx reload”而别选择“service nginx restart”(nginx -s reload 替代 nginx -s stop && start)。

重载服务否？

你会惊讶地得知这种情况经常发生——最复杂的局势有最简单的解决方案。在想破脑袋之前，在急着上论坛之前或 IRC 请求帮助之前，先从最简单的验证开始。

你可能需要花两个小时创建虚拟主机配置，正确保存了文件，然后开始用浏览器检验。但要记住另外的一步——Nginx 不像 Apache，它不支持动态修改.htaccess 配置或类似情况。因此，花一些时间用“service nginx reload”来确保重新载入 Nginx，或者使用“/etc/init.d/nginx reload”或者是“/usr/local/nginx/sbin/nginx -s reload”，不要忘记测试事先的配置。

检查日志

通常，你的问题没有必要在互联网上寻找答案——很有可能答案已经在 Nginx 的日志文件中给出了。你可能需要检查两种日志文件的变化——首先是访问日志文件，这些本身包含有关请求的信息：请求方法和 URI，Nginx 标记的 HTTP 响应代码，等等，具体取决于定义的日志格式。

然而更重要的是，错误日志是一个信息金矿，它依赖于你定义的级别(对于错误诊断，更多的信息请参考 error_log 指令)，Nginx 将提供详细的内置功能。例如，你将能够看到请求 URI 转化为实际的文件系统路径，对于调试重写规则，这是一个很大的帮助。错误日志应该放在/logs/目录下(所配置的 Nginx 相对目录下)，默认为/usr/local/nginx/logs。

安装问题

尝试安装 Nginx 或第一次运行它的时候，一般有三种错误来源：

- 缺少一些先决条件或指定的源路径无效，关于先决条件的详细的描述，请参考第 2 章。
- 在正确安装 Nginx 之后，在网站使用 SSL。在配置那一步时是否正确包含了 SSL 模块？详见第 2 章。
- Nginx 拒绝启动，并且输出类似“[emerg] 3629#0: open() "/path/to/logs/access.log" failed (2: No such file or directory)”的信息，在这种情况下，Nginx 试图访问打开一个文件，例如 日志文件，是无法访问的，这就是无效权限问题引起或无效路径，例如当你指定的日志文件目录在系统中根本就不存在时。

自定义 403Forbidden 错误页

如果你决定使用 allow 和 deny 指令来允许或决绝访问服务器上的资源，那么被拒绝访问的客户端通常会转向“403 Forbidden”错误页，你要为客户精心设置一个自定义的、用户界面友好的 403 错误页面，让他们了解一下他们为什么被拒绝访问。不幸的是，如果自定义页面不能工作，而作为客户端，仍然会得到默认的 Nginx 403 错误页面。

```
server {
    [...]
    allow 192.168.0.0/16;
    deny all;
    error_page 403 /error403.html;
}
```

问题很简单——Nginx 也拒绝访问你定义的 403 错误页！在这种情况下，你需要指定一个访问规则，让 location 区段的访问优先匹配你的网页。可以使用下面的方法，只允许访问自定义 403 错误页面：

```
server {
    […]
    location / {
        error_page 403 /error403.html;
        allow 192.168.0.0/16;
        deny all;
    }
    location = /error403.html {
        allow all;
    }
}
```

如果定义的不仅仅是一个错误页面而是更多，那么你可以指定一个 location 区段来匹配所有错误页面的文件名：

```
server {
    […]
    location / {
        error_page 403 /error403.html;
        error_page 404 /error404.html;
        allow 192.168.0.0/16;
        deny all;
    }
    location ~ "^/error[0-9]{3}\.html$" {
        allow all;
    }
}
```

现在，所有访问者都可以看到你自定义的错误页面。

location 区段的优先权

在同一个 server 区段中使用多个 location 区段经常会发生问题——你的配置不能够像你想象的那样应用。作为一个例子，你想定义的行为是让所有请求图像的客户端请求都转向代理：

```
location ~* \.(gif|jpg|jpeg|png)$ {
        # 匹配任何请求 GIF/JPG/JPEG/PNG 类型的文件
    proxy_pass http://imageserver;  # 由代理转给后端服务器
}
```

以后，你决定启用的/images/目录自动索引，因此，你决定创建一个新的location区段，用来匹配由/images/打头的所有请求：

```
location ^~ /images/ {
    # matches any request that starts with /images/
    autoindex on;
}
```

使用这种配置，当客户端请求下载/images/square.gif 的时候，Nginx 将只应用第二个 location 区段。为什么不是第一个呢？原因在于 location 区段的处理有自己特定的顺序。更多关于 location 区段优先级的信息，请参考第 4 章。

if 区段的问题

在某些情况下，如果不是特别需要，不管你使用的 Nginx 如何建立，都应该避免使用 if 区段。

低效率的语句

有些使用 if 区段不适当的例子，在某些程度上，无用的检查让存储驱动器陷入危险之中。

```
location / {
    # Redirect to index.php if the requested file is not
found
    if (!-e $request_filename) {
        rewrite ^ index.php last;
    }
}
```

对于这样一个配置，被 Nginx 收到的每一个请求为了查找被请求的文件名将触发对目录树的完全校验，因此需要系统调用多个存储磁盘访问。如果你测试/usr/local/nginx/html/hello.html，Nginx 将检测/，/usr，/usr/local，/usr/local/nginx 等等。不管怎样，你应该避免采取这样的声明，例如，通过过滤预先声明的类型(例如，通过进行这种检查，仅当所请求文件与特定扩展名匹配时)：

```
location / {
    # Filter file extension first
    if ($request_filename !~ "\.(gif|jpg|jpeg|png)" {
        break;
   }
```

```
    if (!-f $request_filename) {
        rewrite ^ index.php last;
    }
}
```

非预期行为

if 块最好用于简单的环境，因为它的行为有时候让人惊讶。撇开 if 语句不能嵌套的事实，在下列情况下可能存在问题：

```
# Two consecutive statements with the same condition:
location / {
    if ($uri = "/test.html") {
        add_header X-Test-1 1;
        expires 7;
    }
    if ($uri = "/test.html") {
        add_header X-Test-1 1;
    }
}
```

在这种情况下，第一个 if 块被忽略，只有第二个被处理。然而，如果你在第一个块插入一个 Rewrite 模块指令，例如 rewrite，break 或 return，第一个块将被处理而第二个块将被忽略。

还有其他一些情况使用 if 也会造成问题。

- try_files 和 if 语句放在同一个 location 区段是不被推荐的，在大多数情况下，指令 try_files 会被忽略；
- 一些指令在理论上可以用在 if 块，但会造成严重问题，例如，proxy_pass 和 fastcgi_pass。应该在 location 区段中使用它们；
- 避免在 location 区段中使用 if 块，如果这个 location 块根据其修饰符来捕获正则表达式的话。

这些问题源于 Nginx 配置建立在声明性语言基础之上，但 *Rewrite* 模块的指令，(例如，if，rewrite，return，或 break)使其看起来像脚本。总的来说，你应该尽量避免在 if 块中使用其他模块的指令。

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

译者注

译者注 1：

注意：

```
[root@cms01 ~]# uname -a
Linux cms01 2.6.9-5.ELsmp #1 SMP Wed Jan 5 19:30:39 EST 2005
i686 i686 i386 GNU/Linux
[root@cms01 ~]# echo "Hello! " > /dev/full
[root@cms01 ~]# echo  Hello!  > /dev/full
[root@cms01 ~]#
```

这是我的测试结果，不会有任何返回。因此，要根据具体的系统区使用该设备。

译者注 2：

还要根据实际情况对上述内容进行修改，如果是在 Red Hat 的系统中，可能还得添加类似如下内容：

```
# chkconfig:   - 85 15
# description:  Nginx is an HTTP(S) server, HTTP(S) reverse \
# proxy and IMAP/POP3 proxy server
# processname: nginx
# config:      /usr/local/nginx0.8/conf/nginx.conf
# config:      /etc/sysconfig/nginx
# pidfile:     /var/run/nginx.pid
```

具体实例如下：

```
[root@mail init.d]# more nginx
#!/bin/sh
#
# nginx - this script starts and stops the nginx daemon
#
# chkconfig:   - 85 15
# description:  Nginx is an HTTP(S) server, HTTP(S) reverse \
#               proxy and IMAP/POP3 proxy server
# processname: nginx
# config:      /usr/local/nginx0.8/conf/nginx.conf
```

```
# config:      /etc/sysconfig/nginx
# pidfile:     /var/run/nginx.pid

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
[ "$NETWORKING" = "no" ] && exit 0

nginx="/usr/local/nginx0.8/sbin/nginx"
prog=$(basename $nginx)

NGINX_CONF_FILE="/usr/local/nginx0.8/conf/nginx.conf"

[ -f /etc/sysconfig/nginx ] && . /etc/sysconfig/nginx

lockfile=/var/lock/subsys/nginx

make_dirs() {
   # make required directories
   user=`nginx -V 2>&1 | grep "configure arguments:" | sed
's/[^*]*--user=\([^ ]*\).*/\1/g' -`
   options=`$nginx -V 2>&1 | grep 'configure arguments:'`
   for opt in $options; do
       if [ `echo $opt | grep '.*-temp-path'` ]; then
           value=`echo $opt | cut -d "=" -f 2`
           if [ ! -d "$value" ]; then
               # echo "creating" $value
               mkdir -p $value && chown -R $user $value
           fi
       fi
   done
}

start() {
    [ -x $nginx ] || exit 5
    [ -f $NGINX_CONF_FILE ] || exit 6
    make_dirs
    echo -n $"Starting $prog: "
    daemon $nginx -c $NGINX_CONF_FILE
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
    return $retval
```

```
}

stop() {
    echo -n $"Stopping $prog: "
    killproc $prog -QUIT
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
    return $retval
}

restart() {
    configtest || return $?
    stop
    sleep 1
    start
}

reload() {
    configtest || return $?
    echo -n $"Reloading $prog: "
    killproc $nginx -HUP
    RETVAL=$?
    echo
}

force_reload() {
    restart
}

configtest() {
  $nginx -t -c $NGINX_CONF_FILE
}

rh_status() {
    status $prog
}

rh_status_q() {
    rh_status >/dev/null 2>&1
}

case "$1" in
    start)
        rh_status_q && exit 0
        $1
        ;;
    stop)
        rh_status_q || exit 0
        $1
        ;;
    restart|configtest)
        $1
```

```
        ;;
    reload)
        rh_status_q || exit 7
        $1
        ;;
    force-reload)
        force_reload
        ;;
    status)
        rh_status
        ;;
    condrestart|try-restart)
        rh_status_q || exit 0
            ;;
    *)
        echo     $"Usage:     $0     {start|stop|status|restart|
condrestart|try-restart|reload|force-reload|configtest}"
        exit 2
esac
```

译者注 3：

该指令执行两个动作。

(1) 将进入的 HTTP 请求的主机头与 Nginx 配置文件中各个 server { ... }区段比较，并且选择第一个匹配的 server 区段——即确定虚拟主机。服务器名称(Server name)按照以下顺序处理：①全域名，静态域名；②开始部分使用通配符的域名，例如 *.example.com；③结尾部分使用通配符的域名，例如：www.example.* ；④带有正则表达式的域名。

如果没有找到匹配的 server，则按照下面的顺序在配置文件中选择一个 server { ... }：

① 匹配 listen 指令标记为[default|default_server]的 server 区段；②匹配 listen 指令或隐含 listen 80)的第一个 server 区段(或隐含 listen 80)。

(2) 如果设置 server_name_in_redirect，则设置用于 HTTP 重定向的服务器名称。示例如下：

```
server {
   server_name   example.com  www.example.com;
}
```

第一个名称为服务器的基本名称，将使用默认的机器名(hostname)。可以使用“*”来替代域名的第一部分和最后一部分：

```
server {
   server_name  example.com  *.example.com  www.example.*;
}
```

以上前两个名称(example.com 和 *.example.com)可以合为一个：

```
server {
    server_name  .example.com;
}
```

也可以在服务器名称中使用正则表达式，在名称前加一个波浪符号“~”，示例如下：

```
server {
    server_name www.example.com  ~^www\d+\.example\.com$;
}
```

从 nginx 0.7.12 版开始，支持空服务器名称，能理解没有“Host”值的头(header)，请注意，大多数浏览器总是发送一个“Host header”，如果使用 IP 访问，那么“Host header”将包含该 IP。指定一个能够包含所有(catch-all)的访问区段，请参考 listen 指令的 default_server 标志(flag)。

```
server {
  server_name "";
}
```

从 nginx 0.8.25 版本起，可以在指令 server_name 中使用：

```
server {
    server_name   ~^(www\.)?(?<domain>.+)$;
    location / {
        root  /sites/$domain;
  }
}
```

一些老版本的 PCRE 提供这种语法，如果有任何问题，可尝试下列语法：

```
server {
    server_name   ~^(www\.)?(?P<domain>.+)$;
    location / {
        root  /sites/$domain;
    }
}
```

译者注 4：

概述：

该模块会在当前 location 内容之前或者之后添加其他 location。它是作为一个输出过滤器来执行的，主请求内容和对其他 location 子请求的内容不会被完全缓存，并且仍然以流的方式发送给客户端。因为在发送 HTTP 头的时候，最终响应体的长度还不能确定——HTTP chunked(HTTP 数据块)编码总在这里被使用。

该模块默认情况下不被编译，如果要启用该模块，则在配置编译时使用配置选项——with-http_addition_module 来编译。

例如：

```
location / {
        add_before_body   /before_action;
        add_after_body    /after_action;
}
```

限定：

注意，0.8.17 为止，如果当前 location 自身被作为子请求，则将不会有内容添加。考虑下面的例子：

```
location /foo {
    add_before_body /bar;
    }
    location /bar {
    add_before_body  /baz;
}
```

那么访问/foo 时将不会把子请求/bar 的内容插入。同时还要注意的是，在这两条指令中只能使用字符串而不能使用变量。因此以下配置：

```
location / {
    set $before_action  /before_action;
    add_before_body     $before_action;
}
```

正如预期的(尽管配置文件能够正确载入)，将不能正常工作。

该模块的指令如下表所示。

指令使用环境	描　　述
add_before_body 使用环境： http，server， location	该指令允许在响应体之前添加 URI 内容，然后再作为子请求来处理。 语法：add_before_body uri 默认值：no
add_after_body 使用环境： http，server， location	该指令允许在响应体之后添加 URI 内容，然后再作为子请求来处理。 语法：add_after_body uri 默认值：no
addition_types 使用环境： http，server， location	该指令允许你添加指定的 MIME-类型(默认为 text/html)。 语法：addition_types mime-type [mime-type ...] 默认值：text/html

译者注 5：

例如：

```
location / {
    sub_filter </head>
        '</head><scriptlanguage="javascript"src="$script"></script>';
    sub_filter_once on;
}
```

另外，命令 sub_filter 的默认值为 none，这三条指令的使用环境为：http, server, location。

译者注 6：

该模块建立一些变量，变量的值依赖于请求头中的“User-agent”。产生的三个变量如上所述。

示例：

根据浏览器的不同而选择 index 文件：

```
modern_browser_value "modern.";
modern_browser msie 5.5;
modern_browser gecko 1.0.0;
modern_browser opera 9.0;
modern_browser safari 413;
```

```
modern_browser konqueror 3.0;
index index.${modern_browser}html index.html;
```

重定向旧的浏览器：

```
modern_browser msie 5.0;
modern_browser gecko 0.9.1;
modern_browser opera 8.0;
modern_browser safari 413;
modern_browser konqueror 3.0;
modern_browser unlisted;
ancient_browser Links Lynx Netscape4;

if ($ancient_browser){
   rewrite  ^  /ancient.html;
}
```

该模块提供的指令如下表所示：

指令和使用环境	描　述
ancient_browser 使用环境：http，server，location	功能：当在“User-agent”字段中被识别出的浏览器为旧的浏览器，该指令会指定出一些子链，其中，有一个特别的行“netscape4”相当于正则表达式"^Mozilla/[1-4] "。 语法：ancient_browser line [line...] 默认值：no
ancient_browser_value 使用环境：http，server，location	功能：该指令为变量 ancient_browser 指定值。 语法：ancient_browser_value line 默认值：ancient_browser_value 1
modern_browser 使用环境：http，server，location	功能：该指令指定哪一个版本的浏览器被作为现代的浏览器。当前可指定的值(浏览器)有：msie, gecko (基于 Mozilla) opera, safari, konqueror。还可以指定版本号，格式为(按大小排序)：X, X.X, X.X.X, 或 X.X.X.X，它们中各自的最大值为——4000, 4000.99, 4000.99.99, 和 4000.99.99.99。还有一个特殊值“unlisted”，它表示被考虑为现代浏览器，而没有使用指令 modern_browser 和 ancient_browser 指出的浏览器。在头中没有包括“User-agent”的浏览器被考虑为古老的浏览器(ancient)，除非在“modern_browser unlisted”中指出。 语法：modern_browser browser version|unlisted 默认值：no
modern_browser_value 使用环境： modern_browser_value 1	功能：该指令为变量$modern_browser 指令一个值。 功能：http，server，location 语法：modern_browser_value line

示例：

注意，下面提供的指令 ancient_browser 的使用方法被终止，因此不再使用：

```
ancient_browser "MSIE 4.0" "MSIE 5.0" "MSIE 5.5" "MSIE 6.0";
```

使用实例(来自 *https://gist.github.com/228769*)：

```
#
# Supported browsers
#

modern_browser gecko 1.9;

# note that Safari related directives match
# Chrome, Mobile Safari, Palm Pre and other WebKit-based
browsers here, too, #thanks to all of them using almost
identical User-Agent strings
modern_browser safari 3.0;
modern_browser safari 3.1;
modern_browser safari 3.2;
modern_browser safari 4.0;

modern_browser msie 7.0;
modern_browser msie 8.0;

modern_browser  unlisted;

#
# Non-supported browsers
#

ancient_browser msie 4.0;
ancient_browser msie 5.0;
ancient_browser msie 5.5;
ancient_browser msie 6.0;

# Here is how one can disable support of a specific browser
ancient_browser opera 7;
ancient_browser opera 8;
ancient_browser opera 9;
ancient_browser opera 10;

ancient_browser konqueror 3;
ancient_browser konqueror 4;

ancient_browser Links Lynx Netscape4;
```

```
#
# Now, to the fun part. If we perform a rewrite on every
# request here, images won't be displayed, so we have to
# do some extra work besides just
#
# if ($ancient_browser){
#   rewrite  ^  /unsupported_browser.html;
#   break;
# }

set $unsupported_browser_rewrite    do_not_perform;

if ($ancient_browser){
  set $unsupported_browser_rewrite   perform;
}

if ($uri !~* /\.(jpeg|jpg|png|ico|gif|js|css)$/) {
  set $unsupported_browser_rewrite   do_not_perform;
}

if ($unsupported_browser_rewrite = perform){
  rewrite  ^  /unsupported_browser.html;
  break;
}
```

仅支持最新的版本的 Chrome, Firefox, Internet Explorer, Safari, Mobile Safari 和 Palm Pre。

又如(本例来自互联网)，以下这个配置用于对手机浏览器的判断：

```
modern_browser  unlisted;

ancient_browser "GoBrowser";
ancient_browser "MIDP";
ancient_browser "WAP";
ancient_browser "UP.Browser";
ancient_browser "Smartphone";
ancient_browser "Obigo";
ancient_browser "Mobile";
ancient_browser "AU.Browser";
ancient_browser "wxd.Mms";
ancient_browser "WxdB.Browser";
ancient_browser "CLDC";
ancient_browser "UP.Link";
ancient_browser "KM.Browser";
ancient_browser "UCWEB";
```

```
ancient_browser "SEMC-Browser";
ancient_browser "Mini";
ancient_browser "Symbian";
ancient_browser "Palm";
ancient_browser "Nokia";
ancient_browser "Panasonic";
ancient_browser "MOT-";
ancient_browser "SonyEricsson";
ancient_browser "NEC-";
ancient_browser "Alcatel";
ancient_browser "Ericsson";
ancient_browser "BENQ";
ancient_browser "BenQ";
ancient_browser "Amoisonic";
ancient_browser "Amoi";
ancient_browser "Capitel";
ancient_browser "PHILIPS";
ancient_browser "SAMSUNG";
ancient_browser "Lenovo";
ancient_browser "Mitsu";
ancient_browser "Motorola";
ancient_browser "SHARP";
ancient_browser "WAPPER";
ancient_browser "LG-";
ancient_browser "LG/";
ancient_browser "EG900";
ancient_browser "CECT";
ancient_browser "Compal";
ancient_browser "kejian";
ancient_browser "Bird";
ancient_browser "BIRD";
ancient_browser "G900/V1.0";
ancient_browser "Arima";
ancient_browser "CTL";
ancient_browser "TDG";
ancient_browser "Daxian";
ancient_browser "DBTEL";
ancient_browser "Eastcom";
ancient_browser "EASTCOM";
ancient_browser "PANTECH";
ancient_browser "Dopod";
ancient_browser "Haier";
ancient_browser "HAIER";
ancient_browser "KONKA";
ancient_browser "KEJIAN";
ancient_browser "LENOVO";
ancient_browser "Soutec";
```

```
ancient_browser "SOUTEC";
ancient_browser "SAGEM";
ancient_browser "SEC";
ancient_browser "SED-";
ancient_browser "EMOL";
ancient_browser "INNO55";
ancient_browser "ZTE";
ancient_browser "iPhone";
ancient_browser "Android";
ancient_browser "Windows CE";
ancient_browser "DX";
ancient_browser "TELSON";
ancient_browser "TCL";
ancient_browser "oppo";
ancient_browser "ChangHong";
ancient_browser "MALATA";
ancient_browser "KTOUCH";
ancient_browser "TIANYU";
ancient_browser "TOUCH";
ancient_browser "MAUI";
ancient_browser "J2ME";
ancient_browser "BlackBerry";
ancient_browser "yulong";
ancient_browser "coolpad";

if ( $ancient_browser )
{
proxy_pass http://m.xxx.com;
}
```

译者注 7：

Map 模块的三个指令如下表所示。

指令和使用环境	描　　述
map 使用环境：http	语法：map $var1 $var2 {...} 默认值：无 说明：map 指令有三个特殊值 default、hostnames 和 include。 ● default：指定无匹配内容是的匹配值； ● hostnames：它允许你使用通配符来匹配主机名； ● include：包含一个文件，该值可以使用多次。
map_hash_max_size 使用环境：http	语法：map_hash_max_size size 默认值：map_hash_max_size 2048
map_hash_bucket_size 使用环境：http	语法：map_hash_bucket_size n 默认值：map_hash_bucket_size 32/64/128

译者注 8：

摘要：该模块会建立变量，变量值依赖于客户端端的 IP 地址。

指令和使用环境	描　述
geo 使用环境：http	语法：geo [$ip_variable] $variable { ... } 默认值：无 说明：指令描述依赖的值是基于客户的 IP 地址。默认情况下，该 IP 地址是通过查找到的$remote_addr，但是自从 0.7.27 版本起，它可以使用指定的变量。

```
geo  $arg_remote_addr $geo {
...;
}
```

地址可以指定为 CIDR 格式。此外还有四个特定的值，举例说明：

- delete　删除指定的网络(0.7.23);
- default　变量的值，如果客户端的地址不相当于任何设定的地址，那么将会被改写为默认的 0.0.0.0/0；
- include　包含地址和值信息的文本文件，作为应用，这些文件可以包含进来；
- proxy　指定代理服务器的地址(0.8.7+)。
- ranges　使用范围的格式指定 IP 地址(0.7.23)。该值必须在 geo 的配置环境中第一行使用。

```
geo  $country  {
default          no;
include          conf/geo.conf;
127.0.0.0/24      us;
127.0.0.1/32      ru;
10.1.0.0/16       ru;
192.168.1.0/24    uk;
}
```

在文件 conf/geo.conf 中：

```
10.2.0.0/16       ru;
192.168.2.0/24    ru;
```

对于值的选择，将选择最大值。例如 127.0.0.1 将会获得"ru"，而不是 "us"。有关 IP 范围的例子：

```
geo  $country  {
ranges;
default                    no;
127.0.0.0-127.0.0.0        us;
127.0.0.1-127.0.0.1        ru;
127.0.0.1-127.0.0.255      us;
10.1.0.0-10.1.255.255      ru;
192.168.1.0-192.168.1.255  uk;
}
```

(翻译自 *http://wiki.nginx.org/HttpGeoModule*)

译者注 9：

```
geoip_country_file /absolute/path/to/GeoIP.dat;
location /geoip/ {
rewrite .* /?country=$geoip_country_code;
}
```

geoip_country: 使用环境：http	语法：geoip_country path/to/db.dat; 默认值：无 说明：这个指令决定参观者 IP 所在国家需要使用的.dat 文件，设置后该模块会创建以下变量： ● $geoip_country_code 两个字母的国家代码，例如"RU", "US"； ● $geoip_country_code3 三个字母的国家代码，例如"RUS", "USA"； ● $geoip_country_name 国家的完整名称，例如"Russian Federation", "United States"。
geoip_city: 使用环境：http	语法：geoip_city path/to/db.dat; 默认值：无 说明：这个指令决定参观者 IP 所在国家、地区和城市需要使用的.dat 文件，设置后该模块会创建以下变量： ● $geoip_country_code 两个字母的国家代码，例如"RU", "US"。 ● $geoip_country_code3 三个字母的国家代码，例如"RUS", "USA"。 ● $geoip_country_name 国家的完整名称，例如"Russian Federation", "United States"。 ● $geoip_region 地区的名称(类似于省、地区、州、行政区，联邦土地等)，例如"Moscow City", "DC"。 ● $geoip_city 城市名称，例如"Moscow", "Washington"。 ● $geoip_postal_code 邮政编码。 如果只需要国家名，则可以使用 geoip_country 数据库(1.1MB)，因为 geoip_city 数据库大小为 43MB，并且它们在进行查找时是完全缓存到内存中的。(节选自互联网)

指令和使用环境	描述

```
geo  $country  {
ranges;
default                    no;
127.0.0.0-127.0.0.0        us;
127.0.0.1-127.0.0.1        ru;
127.0.0.1-127.0.0.255      us;
10.1.0.0-10.1.255.255      ru;
192.168.1.0-192.168.1.255  uk;
}
```

(翻译自 *http://wiki.nginx.org/HttpGeoModule*)

译者注 9：

```
geoip_country_file /absolute/path/to/GeoIP.dat;
location /geoip/ {
rewrite .* /?country=$geoip_country_code;
}
```

指令和使用环境	描　　述
geoip_country: 使用环境：http	语法：geoip_country path/to/db.dat; 默认值：无 说明：这个指令决定参观者 IP 所在国家需要使用的.dat 文件，设置后该模块会创建以下变量： ● $geoip_country_code 两个字母的国家代码，例如"RU", "US"； ● $geoip_country_code3 三个字母的国家代码，例如"RUS", "USA"； ● $geoip_country_name 国家的完整名称，例如"Russian Federation", "United States"。
geoip_city: 使用环境：http	语法：geoip_city path/to/db.dat; 默认值：无 说明：这个指令决定参观者 IP 所在国家、地区和城市需要使用的.dat 文件，设置后该模块会创建以下变量： ● $geoip_country_code 两个字母的国家代码，例如"RU", "US"。 ● $geoip_country_code3 三个字母的国家代码，例如"RUS", "USA"。 ● $geoip_country_name 国家的完整名称，例如"Russian Federation", "United States"。 ● $geoip_region 地区的名称(类似于省、地区、州、行政区，联邦土地等)，例如"Moscow City", "DC"。 ● $geoip_city 城市名称，例如"Moscow", "Washington"。 ● $geoip_postal_code 邮政编码。 如果只需要国家名，则可以使用 geoip_country 数据库(1.1MB)，因为 geoip_city 数据库大小为 43MB，并且它们在进行查找时是完全缓存到内存中的。(节选自互联网)

译者注 10：

命令：valid_referers

语法：valid_referers [none|blocked|server_names] ...

默认值：none

使用环境：server，location

译者注 11：

该模块允许客户端 IP 地址由一个请求头(例如 X-Real-IP 或 X-Forwarded-For)变为一个值。如果 Nginx 工作在一些 L7 负载均衡之后，这将是很有用的，Nginx 收到的请求是一个本地 IP，但是代理添加的请求头是用客户端 IP 添加的。

该模块不在 Nginx 默认编译中。要想使用它，需要使用 configure 选项--with-http_realip_module。

注意："你要建立一个受信任的代理列表(见下面)，在头中(header)的第一个 IP 地址是一个不被信任的 IP 地址，它将作为客户端 IP 地址。"来源于 Apache 模块 mod_extract 的 README 文件，信息十分丰富。

示例：

```
set_real_ip_from   192.168.1.0/24;
set_real_ip_from   192.168.2.1;
real_ip_header    X-Real-IP;
```

指令和使用环境	描　　述
set_real_ip_from 使用环境：http，server，location	功能：该指令描述了真实的 IP 地址(也可以称为是可信的 IP，其实就是代理服务器的 IP 地址)，这(些)IP 地址将被用于准确替换——在请求转发时被从 Header 信息中替换掉。 语法：set_real_ip_from [the address|CIDR] 默认值：无
real_ip_header 使用环境：http，server，location	功能：设置用来替换的 IP 地址的头名字，换句话说，就是使用哪个头来替换 IP 地址。 语法：real_ip_header [X-Real-IP|X-Forwarded-For] 默认值：real_ip_header X-Real-IP

译者注 12：

ssl-session-cache 指令的配置有：

- off 硬关闭：Nginx 对客户端明确表明会话不能重新使用；
- none 软关闭：Nginx 对客户端说会话可以重新使用，但是 Nginx 实际上从来不重新使用它们。这是实际上是为某些使用 ssl_session_cache 的邮件客户端提供的一种灵活方案，这样可以使的在邮件代理和 HTTP 服务器中使用。
- builtin OpenSSL 内置的缓存，是在内部仅能被一个进程使用，缓存大小是在会话总数中指定的。注意：如果要使用这种方法可能会出现内存碎片问题，在使用的时候请考虑这一点。请看下面的“References”。
- shared 缓存被所有的工作进程共享。缓存的大小按字节计算：1MB，这个大小能够容纳大约 4000 个会话。对于每一个被共享的缓存必须给定一个(随意的)名字。一个被冠名的缓存可以用在几个不同的虚拟主机 (virtual host)中，换句话说，就是相同名称的缓存可以用在不同名称的虚拟主机中。

可能会同时使用两个缓存类型：builtin 和 shared，例如：

```
ssl_session_cache  builtin:1000 shared:SSL:10m;
```

然而要记住，仅使用 shared 类型，例如，没有 builtin 类型时，shared 类型会更有效些。
如果 ssl_verify_client 设置为“on”或“optional”，那么对于 Nginx 的版本来说，低于 0.8.34 版本时，该指令不能设置为“none”或“off”。

译者注 13：

该模块在编译时没有默认包含，要想使用该功能，需要在编译安装是指定该模块，即在编译时添加选项--with-http_secure_link_module。

示例：

```
location /prefix/ {
    secure_link_secret secret_word;
          # 如果哈希值不正确，$secure_link 便为一个空的字符串
    if ($secure_link = "") {
    return 403;
    }
          # 这里是必须的，否则会得到一个 404 错误
    rewrite ^ /prefix/$secure_link break;
}
```

指令：secure_link_secret

语法: secure_link_secret secret_word

默认值：无

使用环境：location

说明：该指令指定了一个密码字(secret word)，后者在校验请求时使用。完整的受保护链接遵循下列格式：

```
/prefix/MD5 hash/reference
```

这里的 MD5 通过前面介绍的公式计算出，通过这一系列的组合来保护链接。例如，要对文件 top_secret_file.pdf 的链接进行保护，文件位于目录 p 下，那么可以像下面这样配置 Nginx：

```
location /p/ {
    secure_link_secret segredo;
          # 如果哈希值不正确，$secure_link 便为一个空字符串
    if ($secure_link = "") {
    return 403;
    }
          # 这里是必须的，否则会得到一个 404 错误
    rewrite ^ /p/$secure_link break;
}
```

可以通过使用 openssl 命令工具来计算 MD5 哈希值，我们将把文件名 top_secret_file.pdf 和密码字 segredo 联系起来以保护链接：

```
    echo -n 'top_secret_file.pdfsegredo' | openssl dgst -md5
```

注意，在这个命令中，top_secret_file.pdf 和 segredo 之间没有空格！否则会获得这样的哈希值：0849e9c72988f118896724a0502b92a8。现在你可以访问这个受保护的文件了，使用的 URL 为 *http://example.com/p/0849e9c72988f118896724a0502b92a8/top_secret_file.pdf*。

注意前缀，即受保护链接的路径，必须与根路径不同，即不能从根路径开始。只能在非根的路径中使用 secure_link 指令。因此，下面的用法是不合法的：

```
location / {
       # 这是绝对错误的！这是一个根路径！
    secure_link_secret segredo;
    [...]
}
```

译者注 14：

指令：stub_status

语法：stub_status on

默认值：无

使用环境：location

示例：

```
Active connections: 291
server accepts handled requests
16630948 16630948 31070465
Reading: 6 Writing: 179 Waiting: 106
```

解释：

- active connections　所有打开的连接，包括到后端服务器的连接；
- server accepts handled requests　nginx 接受了 16 630 948 个连接，处理了 16 630 948 个连接(没有人仅接受 close 状态)，处理了 31 070 465 个请求(每个连接 1.8 个请求)；
- reading　nginx 读取请求头；
- writing　nginx 读取请求体，处理请求，或者向客户端写响应报告；
- waiting　仍保持(keep-alive)的连接，实际上是活动连接(reading + writing) 。

译者注 16：

该模块在编译时没有被默认包含。要想使用该功能，在编译时添加选项--with-http_dav_module。

示例：

```
location / {
    root     /data/www;
    client_body_temp_path  /data/client_temp;
    dav_methods  PUT DELETE MKCOL COPY MOVE;
    create_full_put_path   on;
    dav_access           group:rw  all:r;

    limit_except  GET {
    allow  192.168.1.0/32;
    deny   all;
  }
}
```

{
  "before_pdg2pic_conversion": {
    "filename": "MTI5MjMwMjZf5a2m5LmgTmdpbnggSFRUUCBTZXJ2ZXIoLnppcA==",
    "filename_decoded": "12923026_\u5b66\u4e60Nginx HTTP Server(.zip",
    "filesize": 18511567,
    "md5": "c60d5143a08485f5585ec9c01b075ba1",
    "header_md5": "0000ea7d67dc139ba91813ae5b6997ac",
    "sha1": "0524092f0b37b0c7c8380f7067391097c9d7504d",
    "sha256": "73ae6f87792636d96cef6c32eb1d4d832e62450b9e68738d12bd35652d430b6e",
    "crc32": 983265024,
    "zip_password": "",
    "uncompressed_size": 19149570,
    "pdg_dir_name": "",
    "pdg_main_pages_found": 346,
    "pdg_main_pages_max": 346,
    "total_pages": 355,
    "total_pixels": 10746720
  },
  "after_pdg2pic_conversion": {
    "filename": "MTI5MjMwMjZf5a2m5LmgTmdpbnggSFRUUCBTZXJ2ZXIoLnppcA==",
    "filename_decoded": "12923026_\u5b66\u4e60Nginx HTTP Server(.zip",
    "filesize": 70354411,
    "md5": "fe8de7ff1c810c087aa128c2f37ddf50",
    "header_md5": "d7387041c163dfe624b8c603b762c16f",
    "sha1": "d82fe35d95bfde12ab7647ea35251a242a1d66a9",
    "sha256": "5d1eec95ae07f08b4681213e15a6a407dd8e32ed0e8e2333f9c7e54433d614bd",
    "crc32": 2053530499,
    "zip_password": "",
    "uncompressed_size": 73628512,
    "pdg_dir_name": "",
    "pdg_main_pages_found": 346,
    "pdg_main_pages_max": 346,
    "total_pages": 355,
    "total_pixels": 1893834152
  },
  "pdf_generation_missing_pages": false
}