HZ BOOKS
华章IT

老男孩教育
oldboyedu.com

Linux/Unix
技术丛书

Linux

Learn Linux Operation with Old Boy

Web Cluster Practice

跟老男孩学Linux运维

Web集群实战

老男孩 ◎著

资深运维架构实战专家及教育培训界顶尖专家十多年的运维实战经验总结，系统讲解网站集群架构的框架模型以及各个节点的企业级搭建和优化。

实战性强，不仅讲解了Web集群所涉及的各种技术，还针对整个集群中的每个网络服务节点给出解决方案，并指导你细致掌握Web集群的运维规范和方法。

机械工业出版社
China Machine Press

Linux/Unix技术丛书

跟老男孩学Linux运维：Web集群实战

老男孩　著

ISBN：978-7-111-52983-5

本书纸版由机械工业出版社于2016年出版，电子版由华章分社（北京华章图文信息有限公司，北京奥维博世图书发行有限公司）全球范围内制作与发行。

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @华章数媒

微信公众号 华章电子书（微信号：hzebook）

目录

前言

为什么要写这本书

不知不觉接触Linux（之前用的是Solaris）已经有16个年头了，在这16年的运维职业生涯中，我走了相当多的弯路，特别是头两年，相当迷茫、彷徨，最要命的是无论怎么努力学习和坚持，就是感觉自己没有入门。那时，优秀的学习书籍寥寥无几，工作中也无人指导，更没有规范的培训机构，一遇到服务器故障就会无所适从，无数次都是在痛苦的挣扎中度过的，也有无数次想要放弃学习Linux。

后来我慢慢地积累了一些经验，有了自己的运维心得。在我的运维技术有了质的飞跃之后，我开始酝酿一套Linux培训体系课程，最初的目的就是希望大家不要重走我走过的弯路，因为这条路充满了荆棘，一不小心就可能走不出来了。

经过一段时间的酝酿，我将自己的想法写成一份项目策划书，发给了公司领导，希望公司能够开展IT培训相关的业务，而我可以负责这块业务。领导首肯了我的策划书，但是在接下来的日子没有任何行动，可能是觉得时机不够成熟吧。

但是我的心却被策划书给拽走了，于是开始了我的兼职IT培训生

涯，这就是“老男孩IT教育”的前身。在多年的培训过程中我发现，很多小伙伴因为条件的限制无法到北京现场学习，虽然我们也录制了大量网络视频，但还是有网友非常希望老男孩能够把讲课的内容整理成书，以便深入学习。看到小伙伴们热切的期盼，我心动了，于是开始计划把讲课的内容整理成书，让全国的小伙伴都能够从中受益。

但是由于培训讲课的排期很紧，课程很多，平时还要在企业里工作，而且写书和讲课也是不同的路数，因此写书计划被一次次地搁浅，直至遇到了她——机械工业出版社华章公司的Lisa，正是因为她的执着、包容、鼓励，使得我有足够的信心和动力完成此书，并且即将策划与Linux运维实战相关的更多图书，这些书后续会一一与大家见面。

目前，全球进入了“互联网+”时代，越来越多的传统企业都在考虑通过网络提供产品和服务，包括互联网+教育、互联网+金融、互联网电商、互联网+出租车、互联网+保险等。而互联网的背后就是Linux技术的时代（包括移动互联网在内），掌握Linux运维技术已经成为每一个IT技术人员的必经之路，本书的中小规模网站集群架构实战就是构建在Linux系统上的高性能、高并发企业级网站集群架构上的解决方案！

读者对象

·Linux系统管理员和运维工程师

·互联网网站开发及数据库管理人员

·网络管理员和项目实施工程师

·Linux相关售前售后技术工程师

·开设Linux相关课程的大中专院校

·对Linux感兴趣的人群

如何阅读本书

本书针对中小规模网站集群的搭建、部署、优化进行了详细讲解，全书可分为三大部分，其中第一部分介绍与Linux相关的基础且重要的知识，第二部分针对当下流行的Web环境架构（LNMP）的搭建及企业级Web优化等进行了讲解，第三部分介绍Web集群后端的数据存储及Web集群前端的负载均衡和高可用。如果你是一名经验丰富的资深Linux用户，可以直接阅读第二部分内容；如果你是一名Linux初学者，请务必从第1章的基础知识开始学习。

第一部分为基础篇（第1~4章），简单地介绍了Linux的历史沿革、Linux的企业级选型、学习环境的搭建、Linux的企业级系统安装、Linux系统的基础优化，以及远程连接Linux及客户端SSH的设置等，最后比较深入地讲解了HTTP协议和WWW服务相关知识，为读者搭建企业级

Web集群环境做好了准备。

第二部分为Web服务篇（第5~8章），着重讲解了Linux、Nginx、MySQL、PHP（LNMP）等当下流行的Web环境架构的搭建、开源blog网站产品的安装部署、Web优化等知识。为读者搭建企业级完整的网站Web集群架构做好了准备。

第三部分为集群篇（第9~15章），着重讲解了Web集群后端的MySQL数据库、Web集群共享存储NFS、Nginx反向代理负载均衡、Keepalived高可用、Memcached缓存及session共享、Nagios企业级监控等技术实战，最后为读者规划了一个中等规模的网站集群架构解决方案。

勘误和支持

由于作者的水平有限，加之编写的时间仓促，书中难免会出现一些错误或者不准确的地方，不妥之处在所难免，恳请读者批评指正。你可以将书中的错误发布到我专门为本书准备的博客地址处：http://oldboy.blog.51cto.com/2561410/1713128，或者在我的微博（http://weibo.com/oldboy8）上留言。同时如果你遇到任何问题，可以加入我为本书提供的两个QQ交流群（339128815和226199808），我将尽量为读者提供最满意的解答。书中所需的各种工具及程序文件也都将发布在上述QQ群及我的博客网站上，我也会将本书的勘误等及时更

新。如果你有更多的宝贵意见，也欢迎你发送邮件至我的邮箱（oldboy@oldboyedu.com），我很期待能够听到你们的真挚反馈。

致谢

首先要感谢伟大的Linux系统开发者Linus Torvalds，是他开创了一款影响我一生的软件。

感谢李泳谊、王洪志，感谢你们在百忙之中为本书供图！

感谢王硕导师及每一个运维课程班级的班长及班干部，感谢你们替我分担老男孩IT教育众多学员的批改作业、答疑、就业指导和管理工作。

感谢老男孩IT教育的每一位学员——是你们的长期支持使得老男孩IT教育的业绩蒸蒸日上，让我有较多时间持续写作。感谢你们对老男孩IT教育的支持。

感谢我的同事Python自动化开发课程的Alex（李杰）老师和武老师，高级架构师课程的赵班长老师，高薪运维就业课程的张耀助教老师，课程顾问歪歪老师、小雨老师、飞雪老师、环宇老师及其他未提及名字的老师，正是你们辛勤努力的工作，使得我有时间完成此书。

感谢中网志腾的郭威和实利通和的王斐和梁露，感谢你们提供给我

的多台DELL物理服务器，使得本书得以高效顺利地完成！

感谢森华易腾的陆锦云女士及其同事，感谢你们提供IDC机房带宽资源并长期支持，使得本书得以顺利完成！

感谢机械工业出版社华章公司的编辑Lisa，感谢你的执着支持、包容和鼓励，在近一年的时间中始终支持我的写作，是你的鼓励和帮助引导我能顺利完成全部书稿。

最后要感谢我的父母、家人，感谢你们将我培养成人，并时时刻刻为我灌输着信心和力量！

谨以此书，献给支持老男孩IT教育的每一位朋友、学员，以及众多热爱Linux运维技术的朋友。

第1章　Linux系统介绍与环境搭建准备

本章以操作系统的介绍作为开篇，首先介绍操作系统的基础概念及操作系统的原理；然后，带领读者了解UNIX的发展史以及市面上常见的UNIX系统版本，并对UNIX/Linux诞生及发展情况进行了说明，附带介绍了发展过程中的关键人物；之后，讲解需要重点掌握的GNU、GPL等名词知识，并对本书“主人公”Linux的优秀特性、常见的Linux发行版本及不同场景下的选择进行了分析；最后，带读者了解互联网常用的两个重点Linux版本：CentOS和Red Hat Linux，并完成CentOS Linux的基本环境搭建准备工作。

1.1 Linux简介

1.1.1 什么是操作系统

如果被问到什么是操作系统，可能很多初学者都会一脸茫然。虽然我们都知道平时一直在用的Windows XP、Windows 7、Windows 8其实就是操作系统，却无法准确给出操作系统的定义，或者向提问者解释清楚什么是操作系统。

操作系统，英文名称为Operating System，简称OS，是计算机系统中必不可少的基础系统软件，它是应用程序运行及用户操作必备的基础环境支撑，是计算机系统的核心。

操作系统的作用是管理和控制计算机系统中的硬件和软件资源，它除了直接管理计算机系统的各种硬件资源（如CPU、内存、磁盘等）以外，还会对系统资源供需的优先顺序进行管理。此外，操作系统还可以控制设备的输入、输出及操作网络与管理文件系统等事务。同时，它也负责对计算机系统中各类软件资源进行管理（例如各类应用软件的安装、运行环境设置等）。图1-1是操作系统与计算机硬件、软件之间的关系示意图。

综上所述，可以给操作系统一个基本的定义：

操作系统是计算机系统中必不可少的基础系统软件，它的作用是管理和控制计算机系统中的硬件和软件资源，合理地组织计算机系统的工作流程，以便有效地利用这些资源为使用者提供一个功能强大、使用方便的操作环境。它在计算机系统（硬件）与使用者之间起到接口的作用。

上面的定义听起来是不是有些复杂？那老男孩就来帮助大家更简单快速地理解什么是操作系统。操作系统就是处于用户与计算机系统硬件之间用于传递信息的系统程序软件。例如：操作系统会在接收到用户输入的信息后，将其传给计算机系统硬件核心进行处理，然后再把处理结果返回给使用者。图1-2是简单理解操作系统作用的示意图。

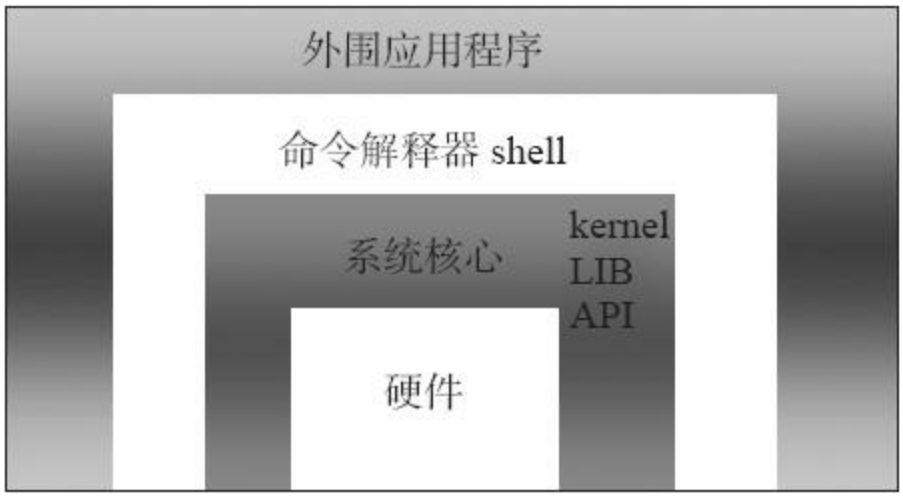


图1-1　操作系统与计算机软硬件关系示意图

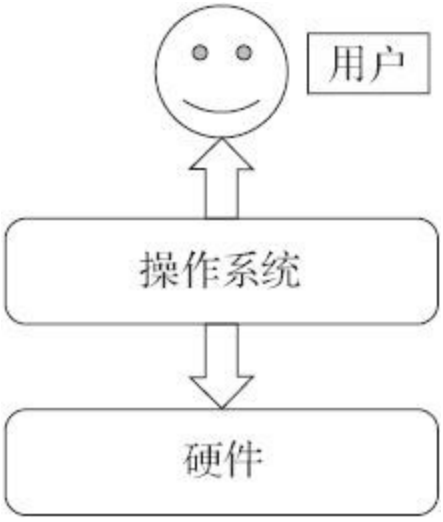


图1-2　简单理解操作系统作用的示意图

目前PC（Intel x86系列）上比较常见的操作系统有Windows、Linux、DOS、UNIX等。

1.1.2 什么是Linux

与大家熟知的Windows操作系统软件一样，Linux也是一个操作系统软件。但是与Windows不同的是，Linux是一套开放源代码程序的，并可以自由传播的类UNIX操作系统软件（UNIX系统是Linux系统的前身，具备很多优秀特性）。其在设计之初，就是基于Intel x86系列CPU架构计算机的。它是一个基于POSIX [1] 的多用户、多任务并且支持多线程和多CPU的操作系统。

Linux是由世界各地成千上万的程序员设计和开发实现的。当初开发Linux系统的目的就是建立不受任何商业化软件版权制约的、全世界都能自由使用的类UNIX操作系统兼容产品。在过去的20年里，Linux系统主要被应用于服务器端、嵌入式开发和个人PC桌面3大领域，其中服务器端领域是重中之重。

我们熟知的大型、超大型互联网企业（百度、新浪、淘宝等）都使用Linux系统作为其服务器端的程序运行平台，全球及国内排名前十的网站使用的主流系统几乎都是Linux系统。

从上面的内容可以看出，Linux操作系统之所以如此流行，是因为它具有如下一些特点：

·是开放源代码的程序软件，可自由修改。

·UNIX系统兼容，具备几乎所有UNIX的优秀特性。

·可自由传播，无任何商业化版权制约。

·适合Intel等x86 CPU系列架构的计算机。

 技巧：好的总结习惯很重要，学会主动对阶段性的知识进行小结是学好Linux运维的重要习惯。

[1] POSIX全称为Portable Operating System Interface，中文翻译为可移植操作系统接口，POSIX标准定义了操作系统应该为应用程序提供的接口标准。

1.2 Linux的起源

1.2.1 UNIX的历史

说到Linux的起源，就不得不提到Linux之前的UNIX系统。UNIX系统于1969年在AT&T的贝尔实验室诞生，20世纪70年代它逐步盛行，这期间又产生了一个比较重要的分支，就是大约1977年诞生的BSD（Berkeley Software Distribution）系统。从BSD系统开始，各大厂商及商业公司根据自身公司的硬件架构，并以BSD系统为基础进行UNIX系统的研发，从而产生了各种版本的UNIX系统，例如：SUN公司的Solaris、IBM公司的AIX、HP公司的HP UNIX等。图1-3给出了UNIX系统诞生、发展的时间及版本分支介绍，供读者参考。

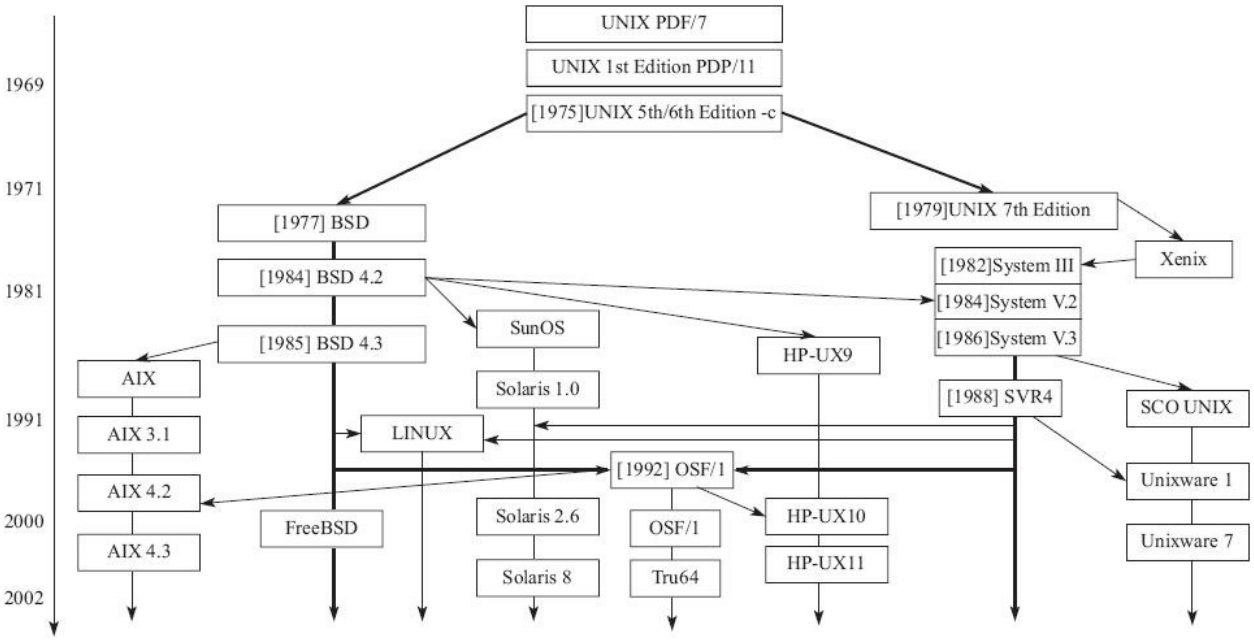


图1-3　UNIX诞生及版本分支发展简略图解

在图1-3中可以看到，本书的“主人公”Linux系统诞生于1991年左右，可以说Linux是从UNIX发展而来的。

1.2.2 UNIX的5大优秀特性

细心的读者应该看到前文曾提到“类UNIX操作系统兼容产品”或“类UNIX操作系统软件”吧？可为什么都要开发类UNIX系统呢？那是因为UNIX是一个非常棒的操作系统，它很像一个非常聪明伶俐但不太听话的孩子，而开发者们在开发系统时，一方面想要继承它的“聪明伶俐”，另一方面又想改善它不听话的一面，故而会有如此考虑。

那么UNIX都有哪些“聪明伶俐”的特点呢？一起来看看吧。

1.技术成熟，可靠性高

使用UNIX系统时，即使连续运行若干年也无需重启，它依然可以工作得非常好。毫不夸张地说，只要计算机硬件不坏，UNIX就很难出现问题。

2.极强的可伸缩性

UNIX支持的CPU处理器体系架构非常多，包括Intel/AMD及HP-PA、MIPS、PowerPC、UltraSPARC、ALPHA等RISC芯片，以及SMP、MPP等技术。


提示：可能是由于早期各大厂商都基于UNIX进行适合自己的

硬件开发，因此，UNIX支持的CPU架构才更多。

3.强大的网络功能

Internet互联最重要的协议TCP/IP就是在UNIX上开发和发展起来的。此外，UNIX还支持很多常用的网络通信协议，如NFS、DCE、IPX/SPX、SLIP、PPP等。

4.强大的数据库支持能力

Oracle、DB2、Sybase、Informix等大型数据库，都把UNIX作为其主要的数据库开发和运行平台，一直到目前为止，依然如此。

5.强大的开发功能

正是UNIX促使了C语言的诞生，并相互促进与发展，成为当时工程师的首选操作系统和开发环境。互联网早期有重大意义的软件新技术的出现几乎都在UNIX上，例如：TCP/IP、WWW、Java、XML等。在互联网早期具有重大意义的软件及新技术几乎都出现在UNIX上。

1.2.3 UNIX操作系统的革命

20世纪70年代中后期，由于各厂商及商业公司开发的UNIX及内置软件都是针对自己公司特定硬件的，因此在其他公司的硬件上基本上无法直接运行，而且当时没有人对开发基于x86架构CPU的系统感兴趣。另外，20世纪70年代末，UNIX又面临了突如其来地被AT&T回收版权等重大问题，特别是要求禁止对学生群体提供UNIX系统源代码，这一度引起当时UNIX业界的恐慌，也因此产生了商业纠纷。

UNIX面临版权回收，以及代码不开源等的问题，直接或间接导致了新的类UNIX系统的诞生，以及自由软件运动的建立和发展。

1984年，Richard Stallman发起了开发自由软件的运动，并成立了自由软件基金会（Free Software Foundation，FSF）和GNU项目。当时发起这个自由软件运动和创建GNU项目的目的其实很简单，就是想开发一个类似UNIX系统，并且是自由软件的完整操作系统，也就是要解决20世纪70年代末UNIX版权及软件源代码面临闭源的问题，这个系统叫做GNU操作系统。

也是在20世纪80年代初期，同样是由于之前的UNIX系统版权和源代码限制等问题，当时大学里教学UNIX系统的束缚很大。因此，一个大学的教授（名字为Andrew Tanenbaum，谭邦宁），大概于1984年开始

着手编写新的用于教学的UNIX系统，目标是开发新UNIX系统，使其尽可能地与原有的UNIX系统兼容，并且可以运行于x86 PC平台，这个系统的名字为Minix。

老男孩补充：由于谭邦宁开发这个Minix系统的目的只是用于教学，因此，Minix系统的功能无法满足商用的需求，但是Minix的产生对于Linux的诞生是至关重要的，且看下文。

1.2.4 Linux的诞生

Linux系统的诞生始于芬兰赫尔辛基大学的一位计算机系名为Linus Torvalds的学生。在大学期间，他接触到了学校的UNIX系统，但是当时的UNIX系统仅为一台主机，且对应了多个终端，使用时存在操作等待时间很长等一些问题，无法满足年轻的Linus Torvalds的使用需求。因此他萌生了自己开发一个UNIX的想法。不久，他找到了前文提到的谭邦宁教授开发的用于教学的Minix操作系统，和我们现在一样，他把Minix安装到了他的I386个人计算机上。此后，Torvalds又陆续阅读了Minix系统的源代码，从Minix系统中学到了很多重要的系统核心程序设计理念和设计思想，从而逐步开始了Linux系统雏形的设计和开发。

Linux的标志和吉祥物为一只名字叫作Tux的企鹅——Torvalds’UNIX，如图1-4所示。

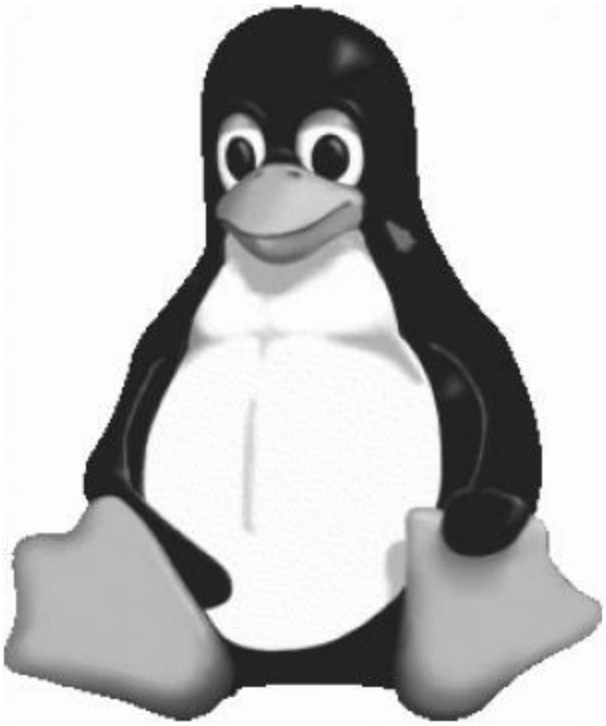


图1-4　企鹅Tux

1.2.5 Linux的发展历程

1.Linux的发展历程简介

1984年，Andrew S.Tanenbaum开发了用于教学的UNIX系统，命名为Minix。

1989年，Andrew S.Tanenbaum将Minix系统运行于x86的PC平台。

1990年，芬兰赫尔辛基大学学生Linus Torvalds首次接触Minix系统。

1991年，Linus Torvalds开始在Minix上编写各种驱动程序等操作系统内核组件。

1991年年底，Linus Torvalds公开了Linux内核源码0.02版（http://www.kernel.org ），注意，这里公开的Linux内核源码并不是我们现在使用的Linux系统的全部，而仅仅是Linux内核（kernel）部分的代码。

1993年，Linux 1.0版发行，Linux转向GPL版权协议。

1994年，Linux的第一个商业发行版Slackware问世。

1996年，美国国家标准技术局的计算机系统实验室确认Linux版本1.2.13（由Open Linux公司打包）符合POSIX标准。

1999年，Linux的简体中文发行版问世。

2000年后，Linux系统日趋成熟，涌现大量基于Linux服务器平台的应用，并广泛应用于基于ARM技术的嵌入式系统中。

2.Linux发展历程中相关人物

我们一定要向前辈们致以深深的敬意，没有他们，就没有今天优秀的Linux系统（如图1-5所示）。

Richard Stallman
（自由软件与 GNU 项目发起人）

Andrew S. anenbaum
（Minix 开发者）

Linus Torvalds
（Linux 之父）

图1-5　Linux系统诞生发展过程中关键代表人物

1.3 Linux核心概念知识

1.3.1 自由软件与FSF

1.自由软件

简单地理解，自由软件的核心就是没有商业化软件版权制约，源代码开放，可无约束自由传播。

 注意：自由软件强调的是权利问题，而非是否免费的问题。大家一定要理解这个概念，自由软件中的自由是“言论自由”中的“自由”，而不是“免费啤酒”中的“免费”。

自由意味着freedom，而免费意味着free，这是完全不同的概念。例如：Red Hat Linux自由但不免费，CentOS Linux是自由且免费的。

自由软件关乎使用者运行、复制、发布、研究、修改和改进该软件的自由。更精确地说，自由软件赋予软件使用者四种自由[1]：

·不论目的为何，有运行该软件的自由。

·有研究该软件如何运行，以及按需改写该软件的自由。当然，取得该软件源代码为达成此目的之前提。

·有重新发布拷贝的自由。

·有改进该软件，以及向公众发布改进的自由，这样整个社群都可受惠。同样，取得该软件的源码为达成此目的之前提。

2.FSF

FSF（Free Software Foundation）的中文意思是自由软件基金会，是Richard Stallman于1984年发起和创办的。FSF的主要项目是GNU项目。它的目的是建立可自由发布和可移植的类UNIX操作系统产品。GNU项目本身产生的主要软件包括：Emacs编辑软件、gcc编译软件、bash命令解释程序和编程语言，以及gawk（GNU’s awk）等。

[1] 此部分内容参考自GNU官方网站。

1.3.2　GNU知识

GNU的全称为GNU’s not unix，意思是“GNU不是UNIX”。GNU计划又称革奴计划，是由Richard Stallman在1984年公开发起的，是FSF的主要项目（如图1-6所示）。前面已经提到过，这个项目的目标是建立一套完全自由的和可移植的类UNIX操作系统。

图1-6　GNU相关图片纪念

GNU类UNIX操作系统是由一系列应用程序、系统库和开发工具构成的软件集合，并加上了用于资源分配和硬件管理的内核。

但是GNU自己的内核Hurd仍在开发中，离实用还有一定的距离。因此，这个GNU系统并没有流行起来。现在的GNU系统通常是使用Linux系统的内核，加上GNU项目贡献的一些组件，以及其他相关程序

组成的，这样的组合被称为GNU/Linux操作系统。

到1991年Linux内核发布的时候，GNU项目已经完成了除系统内核之外的各种必备软件的开发。在Linus Torvalds和其他开发人员的努力下，GNU项目的部分组件又运行到了Linux内核之上，例如：GNU项目里的Emacs、gcc、bash、gawk等，至今都是Linux系统中很重要的基础软件。

1.3.3 GPL知识

1.GPL

GPL全称为General Public License，中文名为通用公共许可，是一个最著名的开源许可协议，开源社区最著名的Linux内核就是在GPL许可下发布的。GPL许可是由自由软件基金会创建的。

1984年，Richard Stallman发起开发自由软件的运动后不久，在其他人的协作下他创立了通用公共许可证（GPL），这对推动自由软件的发展起到至关重要的作用，那么，这个GPL到底是什么意思呢？

简单地理解，GPL许可的核心是保证任何人有共享和修改自由软件的自由，任何人有权取得、修改和重新发布自由软件的源代码权利，但都必须同时给出具体更改的源代码。

虽然整个Linux内核是基于GNU通用公共许可的，但是Linux内核并不是GNU计划的一部分，这一点请读者不要混淆。

2.LGPL

LGPL（Lesser General Public License）相对于GPL较为宽松，允许不公开全部源代码，为基于Linux平台开发商业软件提供了更广阔的空

间。对于该知识点，本书仅提及一下，有兴趣的读者可到网上查询相关信息。

1.3.4 Linux系统组成

Linux操作系统的核心为Linus Torvalds开发的kernel，Linux内核之上的组件分为几部分：一部分是GNU组件，如Emacs、gcc、bash、gawk等；另一些重要组成部分则来自于加利福尼亚大学伯克利分校的BSD UNIX项目和麻省理工学院的X Windows系统项目，以及在这之后成千上万的程序员开发的应用程序等（见表1-1）。正是Linux内核与GNU项目、BSD UNIX及MIT的X11（X Windows）的结合，才使得整个Linux操作系统很快形成，并得到了发展，进而组成了今天优秀的Linux系统。

Linux操作系统=Linux内核+GNU软件及系统软件+必要的应用程序

表1-1　Linux系统各组成部分的贡献人员

Linux 内核	GNU 组件（gcc、bash）	其他必要应用程序
开发者 Linus Torvalds	项目发起人 Richard Stallman	BSD UNIX 和 X Windows 以及成千上万的程序员

Linux系统的核心组成原理示意图参照图1-1。

1.4 Linux的特点

1.4.1 Linux为什么受欢迎

Linux以高效和灵活著称。Linux运行于PC上，可以实现几乎全部的UNIX特性，同时具有多任务、多用户的能力，支持多线程、多CPU。Linux是在GNU通用公共许可（GPL）权限下免费获得的，是一个符合POSIX标准的操作系统。

Linux操作系统软件包不仅包括完整的Linux操作系统，而且还包括了文本编辑器、高级语言编译器，以及X-Windows图形用户界面等应用软件，使用Linux也可以像使用Windows 7、Windows 8一样，通过窗口、图标和菜单对系统进行操作，当然，这是Linux个人桌面领域的应用，在服务器领域绝大多数场景下都还是使用命令行、文本模式操作Linux的。

Linux系统之所以受到广大计算机爱好者的喜爱，主要原因有两个：

一是，Linux属于自由软件，用户不用支付任何费用就可以获得系统和系统的源代码，并且可以根据自己的需要对源代码进行必要的修改，无偿使用，无约束地自由传播。

二是，Linux具有UNIX的全部优秀特性，任何使用UNIX操作系统或想要学习UNIX操作系统的人，都可以通过学习Linux来了解UNIX，同样可以获得UNIX中的几乎所有优秀功能，并且Linux系统更开放，社区开发和全世界的使用者也更活跃。

1.4.2 Linux更多特点介绍

还记得前文对Linux操作系统特性的小结么？除了那些特点以外，其实Linux还具有如下一些特点：

·可以说Linux是UNIX在PC上的克隆版，仿UNIX内核构建，几乎与UNIX指令集向下完全兼容。

·是一个完善的支持多用户、多任务、多进程、多CPU的系统。

·具有很高的系统稳定性与可靠性。

·具有很高的系统安全性。

·有完善的网络服务，支持HTTP、FTP、SMTP、POP、SAMBA、SNMP、DNS、DHCP、SSH、TELNET等。

·是基于GNU许可，自由开放的系统。

·有大量第三方免费应用程序。

·得到了众多业界厂商支持，如IBM、Oracle、Intel、HP、MOTO、Google等。

·有完善的大型数据库平台，包括Oracle、DB/2、Sybase、

MySQL、PostgreSQL等。

·有完善的图形用户界面，包括GNOME、KDE等。

·有完善的开发平台，包括C/C++、Java、Perl等，支持各类图形界面API，如GTK+、QT等。

1.5　Linux的应用领域

1.5.1　IT服务器Linux系统应用领域

如今的IT服务器领域是Linux、UNIX、Windows三分天下，Linux系统可谓后起之秀，特别是最近几年来，服务器端Linux操作系统不断地扩大市场份额，每年增长势头迅猛，并且开始对Windows及UNIX服务器市场的地位构成严重威胁。图1-7是国内服务器端各个系统使用百分比的一个参考饼图。

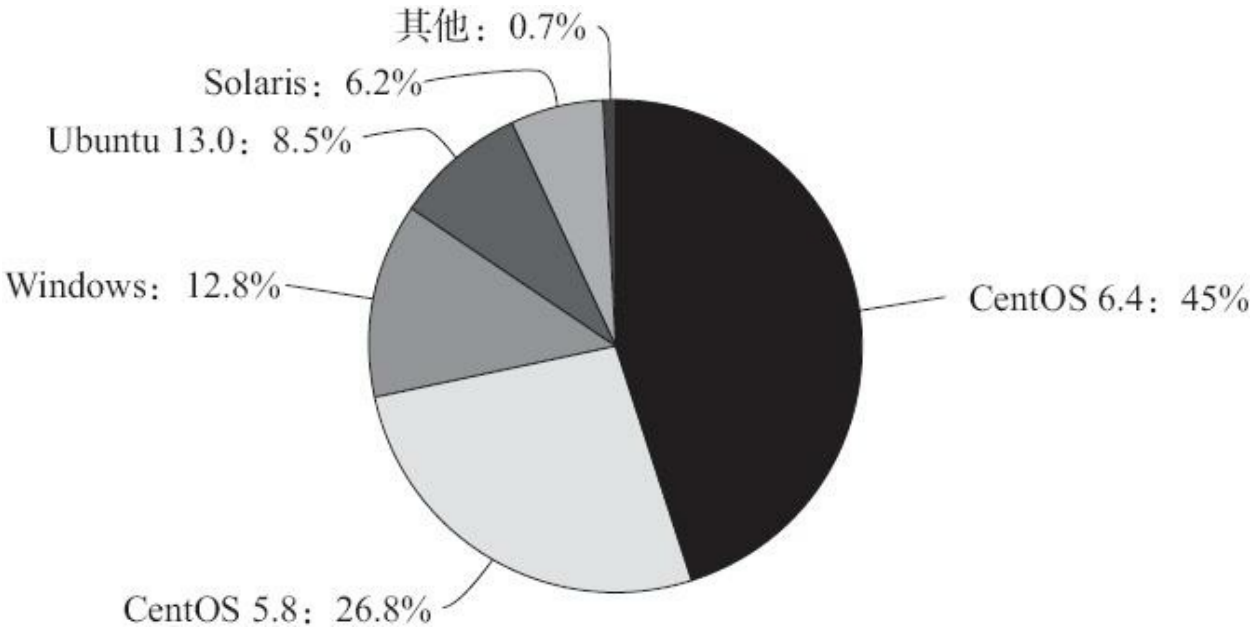


图1-7　服务器端系统使用百分比图

从图1-7中可以看出，Linux占80%左右（包括CentOS、Ubuntu

等），Windows占12.8%，Solaris占6.2%。可见，在未来的服务器领域市场里Linux是大势所趋。

Linux作为企业级服务器的应用十分广泛，利用Linux系统可以为企业构架WWW服务器、数据库服务器、负载均衡服务器、邮件服务器、DNS服务器、代理服务器（透明网关）、路由器等，不但使企业降低了运营成本，同时还获得了Linux系统带来的高稳定性和高可靠性，且无须考虑商业软件的版权问题。

随着Linux在服务器领域的广泛应用，近几年来，该系统已经渗透到电信、金融、政府、教育、银行、石油等各个行业，同时各大硬件厂商也相继支持Linux操作系统。这一切表明，Linux在服务器市场前景光明。同时，大型、超大型互联网企业（百度、新浪、淘宝等）都在使用Linux系统作为其服务器端的程序运行平台，全球及国内排名前十的网站使用的几乎都是Linux系统，Linux已经逐步渗透到各个领域的企业里。

1.5.2 嵌入式Linux系统应用领域

由于Linux系统开放源代码，功能强大、可靠、稳定性强、灵活，而且具有极大的伸缩性，再加上它广泛支持大量的微处理器体系结构、硬件设备、图形支持和通信协议，因此，在嵌入式应用的领域里，从因特网设备（路由器、交换机、防火墙、负载均衡器等）到专用的控制系统（自动售货机、手机、PDA、各种家用电器等），Linux操作系统都有很广阔的应用市场。特别是经过这几年的发展，它已经成功地跻身于主流嵌入式开发平台。例如，在智能手机领域，Android Linux已经在智能手机开发平台牢牢地占据了一席之地。嵌入式系统是另一个应用领域，不是本书讨论的主要话题，读者若对此感兴趣，可参考相关文章和书籍。

1.5.3　个人桌面Linux应用领域

所谓个人桌面系统，其实就是我们在办公室使用的个人计算机系统，例如：Windows XP、Windows 7、MAC等。Linux系统在这方面的支持也已经非常好了，完全可以满足日常的办公及家用需求，例如：

·浏览器上网浏览（例如：Firefox浏览器）。

·办公室软件（Open Office，兼容微软Office软件）处理数据。

·收发电子邮件（例如：ThunderBird软件）。

·实时通信（例如：QQ等）。

·文字编辑（例如：vi、vim、emac）。

·多媒体应用。

虽然Linux个人桌面系统的支持已经很广泛了，但是在当前的桌面市场份额还远远无法与Windows系统竞争，这其中的最大障碍可能不在于Linux桌面系统产品本身，而在于用户的使用观念、操作习惯和应用技能，以及曾经在Windows上开发的软件的移植问题。

1.5.4 本书讲解的Linux领域说明

本书主要讲解Linux系统服务器端的知识、技术，以及企业生产运维的实践经验。

下面为本书讨论的技术核心及什么企业单位会用到本书的技术知识：

·服务器领域的Linux运维技术。

·基于X86 CPU架构的计算机硬件的Linux系统。

·面向互联网的企业，或者即将将业务转移到互联网的企业。

1.6　如何选择Linux的发行版

1.6.1　Linux的发行版本介绍

Linux内核（kernel）版本主要有4个系列，分别为Linux kernel 2.2、Linux kernel 2.4、Linux kernel 2.6、Linux kernel 3.x，更多更新的内核版本请浏览https://www.kernel.org/ 。

Linux的发行商包括Slackware、Red Hat、Debian、Fedora、TurboLinux、Mandrake、SUSE、CentOS、Ubuntu、红旗、麒麟……

下面来看看其中几个重要的发行版本。

·Red Hat：Red Hat Linux 9.0的内核为2.4.20。在版本9.0后，Red Hat不再遵循GPL协议，成为收费产品（但仍开源），发展的新版本依次为Red Hat 3.x、Red Hat 4.x、Red Hat 5.x、Red Hat 6.x、Red Hat 7.x。

·Fedora：Red Hat的一个分支，仍遵循GPL协议，可以认为是Red Hat预发布版。

·CentOS（Community Enterprise Operating System）：Red Hat的另一个重要分支，以Red Hat所发布的源代码重建符合GPL许可协议的Linux系统，即将Red Hat Linux源代码的商标（LOGO）及非自由软件部

分去除后再编译而成的版本。目前CentOS已被Red Hat公司收购，但仍开源免费。CentOS Linux是国内互联网公司使用最多的Linux系统版本，也是本书的“主人公”，本书后面所有的内容讲解都是基于CentOS这个操作系统的，绝大部分内容几乎无需任何修改即可适合其他Linux操作系统版本。

提示：有关Linux操作系统，记住Red Hat、CentOS、Ubuntu、Fedora、SUSE、Debian等即可。对于Red Hat与CentOS的区别和联系，有时会被企业面试官问到，需要重点了解。

1.6.2 选择适合的Linux系统学习

在了解了Linux版本及应用领域之后，接下来就要定位我们到底该往哪方向发展了。如果你想做一个网站的后端运维工程师，那么就沿着服务器领域的路线；如果你想进入嵌入式领域，那么就要学习嵌入式领域的技能；如果你对桌面Linux系统感兴趣，那么可以深入桌面系统领域。选择的领域不同，学习和要掌握的技能自然就会有差别，因此，这个选择就显得很重要了。当你阅读本书的时候，相信你已经选择了IT服务器Linux系统应用领域。没错，我们整本书的主要内容就是基于Linux服务器应用领域而写的。

1.Linux发行版本应用场景

在确定了自己的发展路线后，针对Linux系统选择一个合适的版本就显得尤为重要了。事实上，这个问题也正是大多数初学者最为头疼的一个问题，对于此，笔者的建议如下：

如果你是一个Linux爱好者，想选择一个桌面系统，并且既不想用盗版，又不想花太多钱购买商业系统软件，那么可以选择Ubuntu桌面系统。如果你需要服务器端的Linux系统，想用一个比较稳定的服务器系统，或者说目标就是进入企业从事Linux运维工作，那么建议你选择CentOS或Red Hat。在这两者当中又应首选CentOS，因为目前市场的趋

势是这样的，CentOS社区非常活跃。如果对系统稳定性、安全性有更高的要求，或者是特殊使用偏好的用户，可以考虑Debian或FreeBSD。如果特别痴迷于新技术体验和追求最新的软件版本，可以选择Fedora，但要容忍Fedora潜在的新技术软件的Bug和系统稳定性的问题。如果喜欢更好的中文环境支持，可以选择麒麟Linux……本书主要侧重服务器领域，并且根据国内互联网企业的市场需求，本书所选择的Linux版本为CentOS，当然所讲的技术也适合Red Hat Linux及其他大部分Linux系列，CentOS是当前国内互联网企业服务器端实际应用最多的系统。

其实Linux虽然发行版本众多，但是系统的核心及大部分外围基础应用软件都是相同的或仅仅是简单的变种，只要学会其中一种，即可触类旁通，因此不建议读者同时学习多个系统，更不建议在工作中同时使用多个系统版本，这不但会浪费自己的学习时间，也影响企业业务的稳定，增加无谓的维护成本。前面已说明，本书将以企业最常用的CentOS系统为主进行讲解，但是想学习其他版本的Linux的读者，也同样适合阅读本书，因为它们的使用方法绝大部分都是相同的。

 提示：当前，绝大多数互联网公司和企业的Linux操作系统平台为CentOS和Red Hat，通常情况下，只要能掌握这两个操作系统的应用就可以胜任绝大多数公司的相关运维工作。对于其他版本的操作系统，包括UNIX，想要再学也会轻车熟路，因为它们之间的差异很小。

2.选择CentOS Linux的版本

本书讲解的Linux运维技术主要是基于CentOS x86_64 Linux的，绝大部分知识几乎无需任何修改同样也适用于Red Hat Linux等同源或类似Linux系统版本。

下面是CentOS的主流版本在国内互联网企业的使用现状说明。

·CentOS 5系列：占25%左右，主流版本有CentOS 5.5、CentOS 5.8、CentOS 5.10、CentOS 5.11，不推荐新手学习。

·CentOS 6系列：占45%左右，主流版本有CentOS 6.2、CentOS 6.4、CentOS 6.6、CentOS 6.7，推荐新手学习。

·CentOS 7系列：刚刚发布不久，目前极少企业正式使用。根据企业的主流应用进行选择才是明智的，不要盲目选择最高的系统版本。

综上所述，老男孩推荐学习当下企业的主流应用，即CentOS 6系列，这其中又首选CentOS 6.6。因为只要大版本和企业相同即可，小版本的差别对学习来说几乎无影响，故而可以选择该系列最新的小版本，即CentOS 6.6 x86_64来学习。其实，只要学透一种版本，再学别的系统版本自然就会触类旁通了。

已参加工作的运维人员选择CentOS系统的版本建议：

如果你在企业有决定权，那么老男孩建议你选择CentOS 6.4或CentOS 6.6的64位x86_64系统，目前很多大中型公司使用CentOS 5或

CentOS 6系列，或者将这两个系列并行使用，并正在向CentOS 6系列过渡。选主流的版本软件本身会更稳定，经过企业长时间生产考验，遇到问题网上解决方案资料也更多。

大公司的局部新业务可能慢慢会试着用CentOS 6.6或CentOS 7，由于新业务相对单一，重要性可能也没那么高，所以局部业务可以先用新系统测试，等测试一段时间后稳定了，企业人员也熟悉了新系统较长时间后，就可以慢慢向CentOS 6.6甚至CentOS 7大量迁移了，这个试用的过程很重要，切勿跟风瞬间做大量的版本升级，企业里运维工作还是以稳定优先，没有需求也可不升级。

中小型新公司或新业务可能会直接用最新版CentOS 7 64位，这样的公司里很多为从业新手，可能没有CentOS 5或者CentOS 6系列的体验和应用经验，其实，直接选择CentOS 7是比较盲目的。最新版一般潜在的问题隐患会较多，且不可预知，多数企业未使用，你选择使用后遇到问题不好解决，可能无人可问，也无处查资料。一些运维新人经常这么做。看过本书的读者要注意，不要犯这样的错误。不过最新版的性能和功能等往往还是不错的，对于真正的选择方法，还是看企业的实际需求，以及内部运维人员对特定版本的熟练程度。先试水然后逐步过渡，是大公司的选择思路。

 面试技巧：大家被面试官问及使用的是什么操作系统时，一定要一次性说出来（系统版本、内核版本、32位还是64位），例如：我的

工作中使用的是CentOS 6.6 x86_64位Linux系统，内核版本为2.6.32-504，这才是一个合格的Linux运维人员的表现。

1.7 搭建学习Linux的运维环境

1.7.1 虚拟机软件介绍

简单地说，虚拟机（Virtual Machine）软件就是一套特殊的软件，它可以作为操作系统独立运行，也可以运行于操作系统之上。若是运行于系统之上的虚拟机软件，在一台计算机（PC或笔记本电脑等）上安装虚拟机软件后，就可以模拟若干台相对独立的虚拟PC设备，并且可以在每台虚拟的PC设备上安装运行操作系统，运行网络服务，与真实的计算机设备几乎无任何使用差别。

使用时，需要先在计算机上安装虚拟机软件（例如：VMware Workstation），然后通过安装的虚拟机软件创建一个或多个虚拟机系统（即虚拟的计算机设备），最后在这些虚拟的计算机设备上安装操作系统并进行启动配置，最终实现在一台计算机上“同时”运行多个虚拟机设备系统。

另外，还可以将这些虚拟的系统连成局域网，用来部署网站集群架构等更深层次的运维技术，这样的虚拟环境在后文会涉及。图1-8为安装VMware Workstation虚拟机软件后打开的软件界面。

在图1-8中，展示的是在Windows 2007桌面操作系统上安装的虚拟

机软件VMware，这里通过配置VMware虚拟了8台PC设备，且分别在这8台PC上安装Linux系统（实际学习中是先安装一台，其他的系统可以通过第一台进行克隆）。这8个虚拟机同时在一台计算机上独立运行，几乎互不干扰，并且可同在一个局域网内，还可以互相通信。

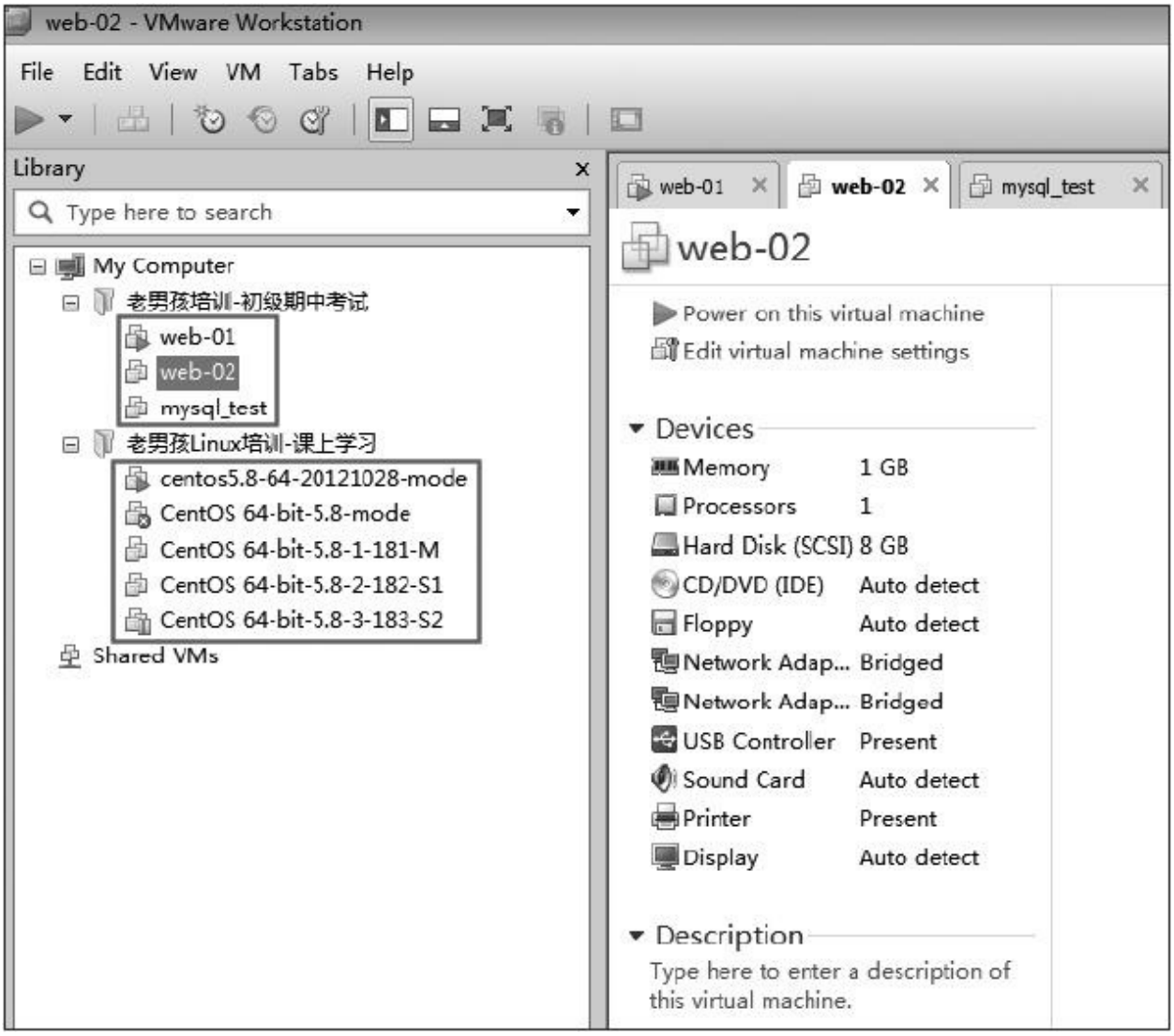


图1-8　Windows 2007系统安装VMware Workstation8虚拟软件窗口

经过前面对虚拟机软件的介绍，相信读者应该知道虚拟机软件到底

是什么了吧。下面，老男孩介绍一些网友常用的虚拟机软件，如表1-2所示。

表1-2　常用的虚拟机软件及选择建议

虚拟机软件	特点及选择建议
VMware Workstation	工作站版虚拟化软件，简单、易用，适合搭建学习环境
KVM/Xen Linux 的虚拟化	服务器级虚拟化软件，适合企业虚拟化应用，复杂，不适合学习环境
Virtual PC	MAC 平台可以用
VirtualBox	开源的虚拟机软件

1.7.2 通过虚拟机软件学习Linux运维

通过虚拟机软件是初学者学习Linux运维的最佳方式。

在与部分网友和学生的交流中，老男孩发现，很多初学者都认为，学习Linux就必须把自己的计算机安装成Linux系统或必须有真正的服务器设备。这实际上是一些机构、书籍或网络文章给人传导的错误思想和思维。其实，学习Linux最简单、最实用的环境就是虚拟机环境（例如：通过VMware Workstation等软件学习）。老男孩这样说的原因有如下几点：

·利用虚拟机软件搭建Linux学习环境简单，容易上手，最重要的是利用虚拟机模拟出来的Linux与真实的Linux几乎没有任何区别。如果购买服务器，动则一两万元，不是一般的网友所能承受的，而且声音很大，很费电。如果用PC和笔记本电脑搭建Linux（包括双系统共存方式），那就完全是Linux环境了，不但加大你的学习难度（例如：无法用Word记笔记和正常浏览网页），也与实际的工作环境相差很远，即南辕北辙了，企业里运维人员的工作环境绝大多数都是在Windows桌面系统下通过SSH工具（SecureCRT/Xshell）远程连接千百里之外的Linux服务器进行管理和维护的。因此，用虚拟机软件来搭建环境是最接近企业工作环境的。

·搭建Linux集群等大规模环境有时需要同时开启几台虚拟机（每台虚拟机仅需256~512MB内存、6~8GB的硬盘空间即可），此时如果是用服务器或者自己的计算机安装Linux，则很难满足学习要求，购买多台服务器就更不现实了。事实上，仅仅用价值四五千元的个人笔记本电脑就可以轻松满足搭建中小规模Linux集群架构的学习需求。

·如果用虚拟机学习，只要计算机配置高一点，就可以同时开启多个Linux虚拟机，在上班、回家的路上，带着笔记本电脑即可随时学习。如果是多台真实计算机和服务器设备，就无法移动了。当然有读者会说可以放机房里，但这个代价也太大了。大多数学习者很难有这样的资源。

·使用虚拟机系统环境，我们可以随意对虚拟系统进行任何的设置和更改操作，甚至可以格式化虚拟机系统硬盘，进行重新分区等操作，而且完全不用担心会丢掉有用数据，因为虚拟机是系统上运行的一个虚拟软件，对虚拟机系统的任何操作都相当于在操作虚拟机的虚拟机设备和系统，不会影响计算机上的真实数据。

综上所述，老男孩想给大家的建议就是，踏实地用虚拟机学习吧，学习Linux运维，几乎99.9%的知识都与硬件设备无关，我们不要给自己设置太多的限制和门槛，那样会影响学习Linux的进度，从而可能丧失学习Linux的兴趣。

当然了，如果在学习的过程中有条件可以摸一摸真正的服务器设备。在实际教学中，我们会让学生摸到服务器，不仅可以进行RAID制作，还可以给真实的物理服务器安装系统。总之一句话，如果没有设备，在计算机上安装虚拟机一样可以搭建逼近工作环境的学习环境，当然如果有设备配合虚拟机学习，学习效果更佳。

企业真正服务器硬件手把手介绍请见 http://v.qq.com/page/g/x/y/g016789xvxy.html 。

1.7.3　选择适合自己的虚拟机软件

1.选择适合的虚拟机软件

如果你使用的是Windows系统，那么，老男孩推荐你使用VMware Workstation；如果是Mac OS平台，可以选择Virtual PC；如果你用的是Ubuntu系统，可以选择Xen、KVM、VMware（Linux版本）。本书将以在Windows 2007系统上安装VMware Workstation 8.0为例，为大家讲解Linux运维技术，同时会在DELL R710真实服务器环境中进行测试，其他环境的搭建大同小异，进入Linux里面几乎无差别。

2.虚拟机软件对硬件的要求

虚拟机软件的原理是利用宿主机物理硬件资源虚拟PC设备，因此对物理机硬件的要求比较高，其中最主要的是内存、硬盘和CPU资源。首先，宿主机物理内存要足够大，最低在4GB以上，因为在创建虚拟设备时，要为每个虚拟机分配一定的内存资源（一般最小为256MB，实际学习时可以设置在256~1024MB之间）和硬盘空间（默认8GB即可），SSD固态盘最好，同样也要分配CPU资源，CPU最好是I5以上，例如：每个虚拟机分配一核CPU。当然了，多个虚拟机系统也可以同时占用一个核，在日常学习Linux时，如果不进行大量并行安装软件等消耗CPU资源的操作，使用虚拟机环境还是非常舒服的。

1.7.4　安装与使用VMware虚拟机软件

1.对VMware Workstation版本的建议

表1-3中给出的是选择VMware Workstation版本的建议。仅为建议，非必须照做。

表1-3　VMware Workstation版本选择建议

物理机（宿主机）系统版本	VMware Workstation 版本
Windows XP（不推荐）	VMware Workstation 5.5（不推荐）
Windows 7（推荐）	VMware Workstation 8.0（推荐）
Windows 8（不推荐）	VMware Workstation 9.0/10.0/11.0（不推荐）

 提示：本书写作时，Windows 10发布了，不建议新手使用最新系统作为宿主机学习Linux，因为会遇到各种“坑”，走前人走过的路才是最佳的学习思路，学会后再变通就会好多了，因此，推荐大家使用Windows 7系统和VMware Workstation8.0虚拟机软件，与老男孩使用的一致最佳。

2.虚拟机软件的安装

在Windows系统下安装了适合Windows版本的VMware Workstation后，就可以在VMware Workstation上创建虚拟机了。之后运行创建的虚拟机，在虚拟机上安装CentOS Linux操作系统，这个安装过程与在实际

生产环境下安装是一样的。

VMware Workstation虚拟机软件的安装很简单，只需按照Windows常规方法，持续按“下一步”即可完成，这里不再描述，如果你遇到了问题，可以在前言中寻找老男孩为本书建立的问题反馈交流群。

1.7.5　创建一个虚拟机的实践

1）虚拟机软件安装完毕后，双击桌面上的VMware Workstation图标，或者在开始程序菜单里找到VMware Workstation图标来启动VM应用程序，如图1-9所示。

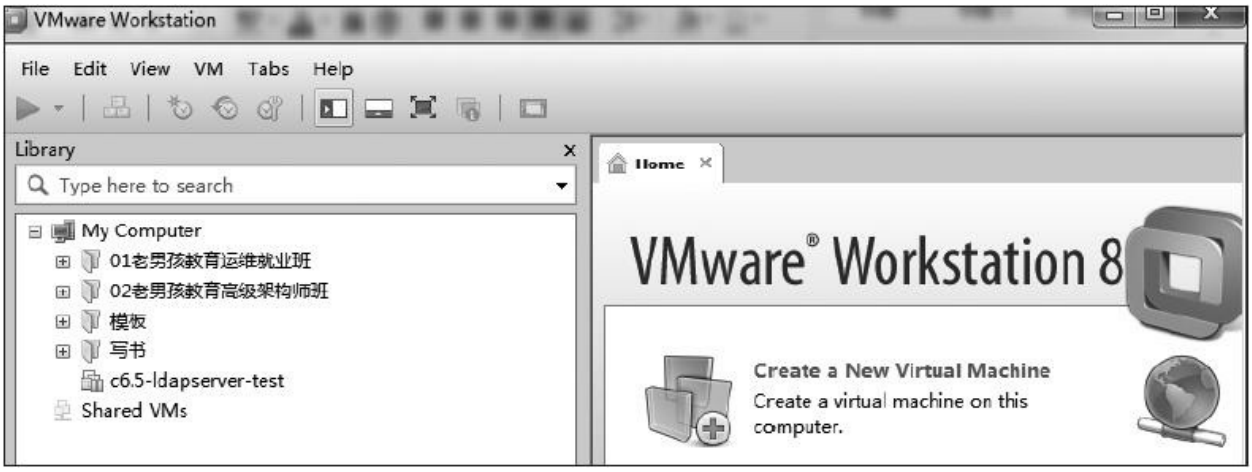


图1-9　VMware Workstation虚拟机软件主界面

2）接下来按Ctrl+N快捷键（或者点选菜单栏File→New virtual machine）创建一个新虚拟机，此时，会出现新建虚拟机向导窗口，如图1-10所示。

3）在图1-10所示的虚拟机创建向导界面中，单选Custom（advanced），即自定义安装，然后单击Next按钮继续，弹出如图1-11所示的界面。

图1-10　虚拟机创建向导界面

图1-11　虚拟机硬件兼容和设备限制说明界面

4）图1-11描述了虚拟机硬件兼容和虚拟设备大小限制，在图右侧的Limitations下面，可以发现软件对要创建的虚拟机的硬件大小会有限制，例如：内存最大为64GB、CPU为8 processor limit、磁盘为2TB disk size limit等，不过，这些并不会影响学习，保留默认选择即可。然后单击Next按钮继续，此时会出现如图1-12所示的窗口。

5）在图1-12所示的界面中，“为虚拟机选择如何安装系统”这一步骤很关键。请选择界面最下面标示的“I will install the operating system later.”，表示创建虚拟机后不再默认安装系统，即创建虚拟机后，我们

可以手动选择镜像或将光盘放入虚拟光驱自行安装。如果选择了“Installer disc image file”，则在创建完虚拟机后就会自动安装系统。这虽然很方便，但是会增加太多的安装包，并且会自动分区，这样就不是企业环境安装的标准了。选择完毕，然后单击Next按钮继续，会出现如图1-13所示的界面。

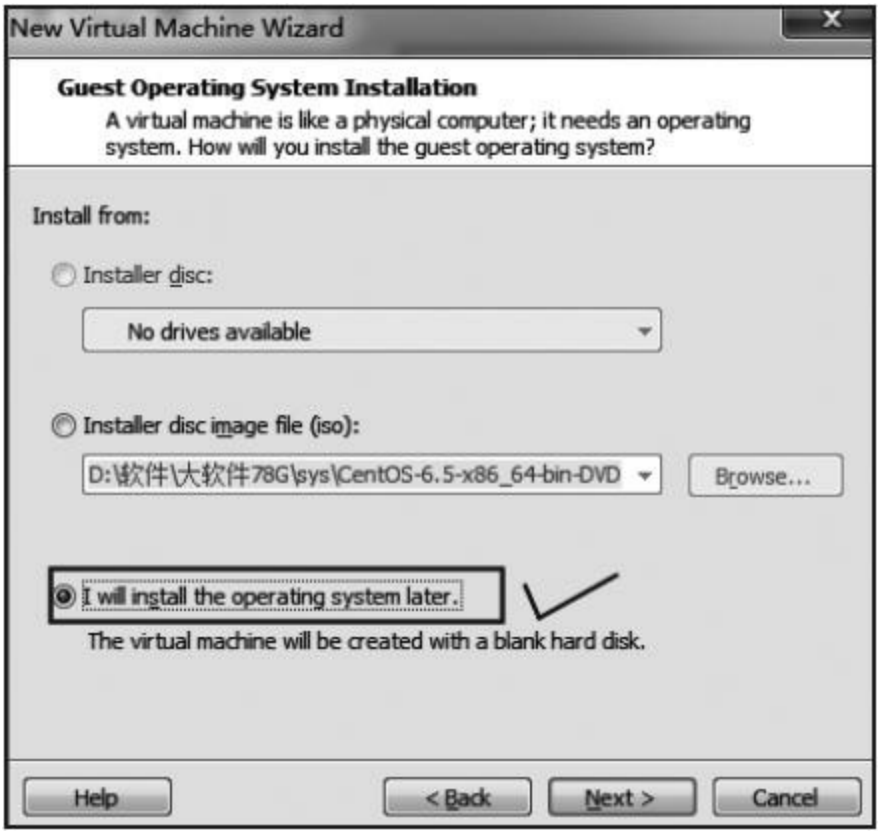


图1-12　为虚拟机选择如何安装系统

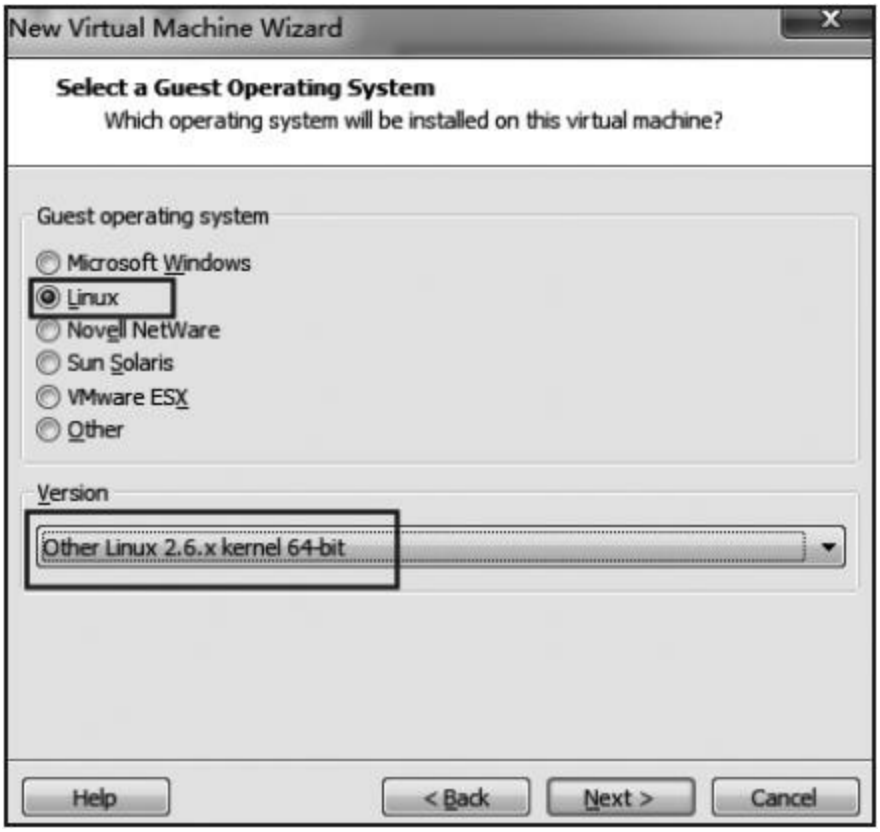


图1-13　为虚拟机选择系统类型及内核版本

6）“为虚拟机选择系统类型及内核版本”也很关键，这一步是选择系统类型及内核版本进行系统安装，如果选错了，可能无法正确安装系统。

这里选择Linux系统类型，内核为“Other Linux 2.6.x kernel 64-bit”，也就是说，要安装的系统为CentOS 6.6 x86_64系统。也可以在系统类型中直接选择CentOS 64bit（但不推荐，因为笔者没有这样选过，运维思想：多走自己走过的正确的路）。选择完毕，然后单击Next按钮继续，出现如图1-14所示的界面。

7）在图1-14所示的界面中，要为虚拟机起名并选择安装程序的路径，路径要选择大一点的宿主机磁盘分区，剩余空间至少要大于20GB，配置完毕后单击Next按钮继续，出现的界面如图1-15所示。

图1-14　为虚拟机起名及选择程序安装路径

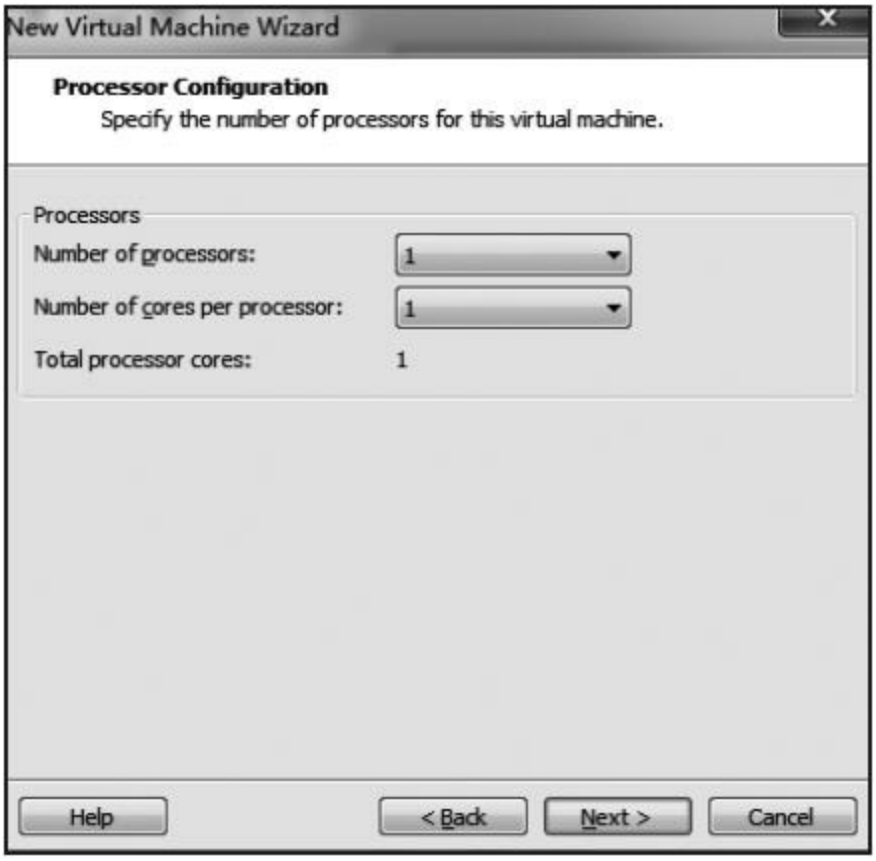


图1-15　为虚拟机选择CPU资源

8）在如图1-15所示的界面，要选择虚拟机对应CPU的颗数，以及每个CPU的核数，默认都是1，选此即可。我们仅仅是用虚拟机来进行学习、实验，不要求虚拟机设备资源有多好。单击Next按钮继续，出现的界面如图1-16所示。

注意：CentOS 6系列x86_64在虚拟机上设置CPU时可能会出现故障，这是因为笔记本电脑或台式机硬件CPU需要支持虚拟化技术，但有些系统硬件默认是不支持的，可以通过调整BIOS使其支持，不过，某些旧机器可能只能安装32位的Linux系统。详见本节结尾处。

9）“为虚拟机选择内存资源大小”也很关键，默认选择的虚拟机内存大小为384MB，建议调整为1024MB。之后，单击Next按钮继续，出现的界面如图1-17所示。

内存大小设置分析：

·创建虚拟机时内存至少大于512MB，最好是1GB以上，否则安装系统可能报错，因为从CentOS 6开始，多数都用图形界面安装系统了，因此，占用的内存会比较大。

·安装Linux之后，启动虚拟机时内存最好在512MB以上，最低是256MB，否则也可能报错。

·如果是CentOS 5系列，安装时内存为256MB，启动时100MB左右即可。CentOS 6系列安装和启动都会占用更多的内存。

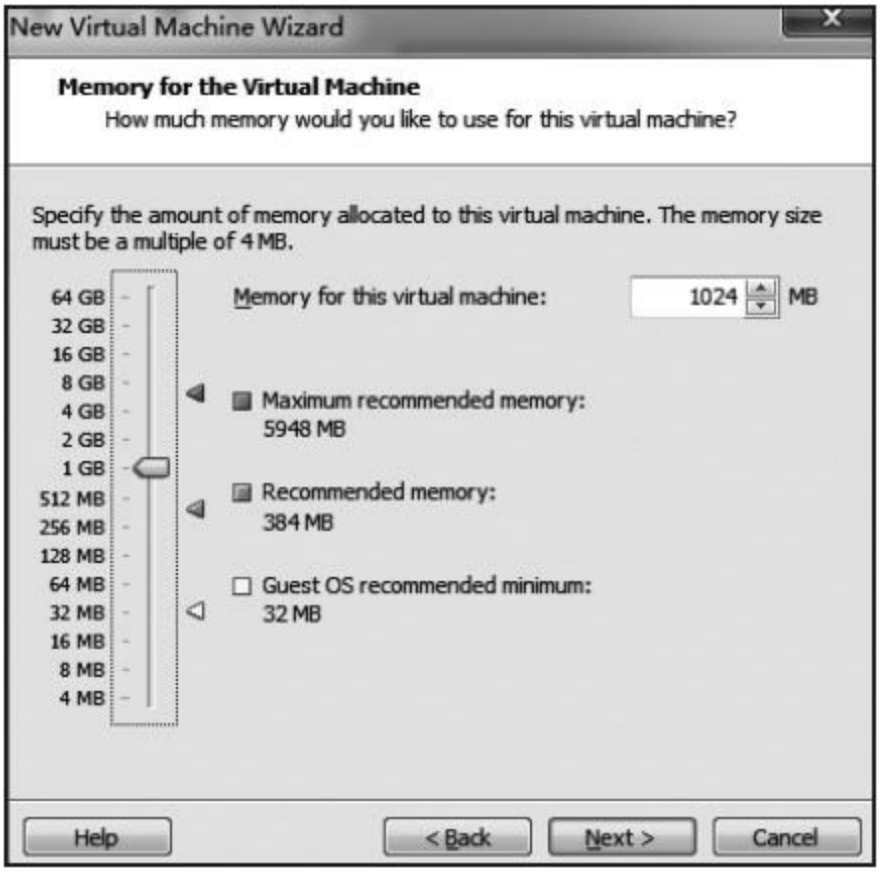


图1-16　为虚拟机选择内存资源大小

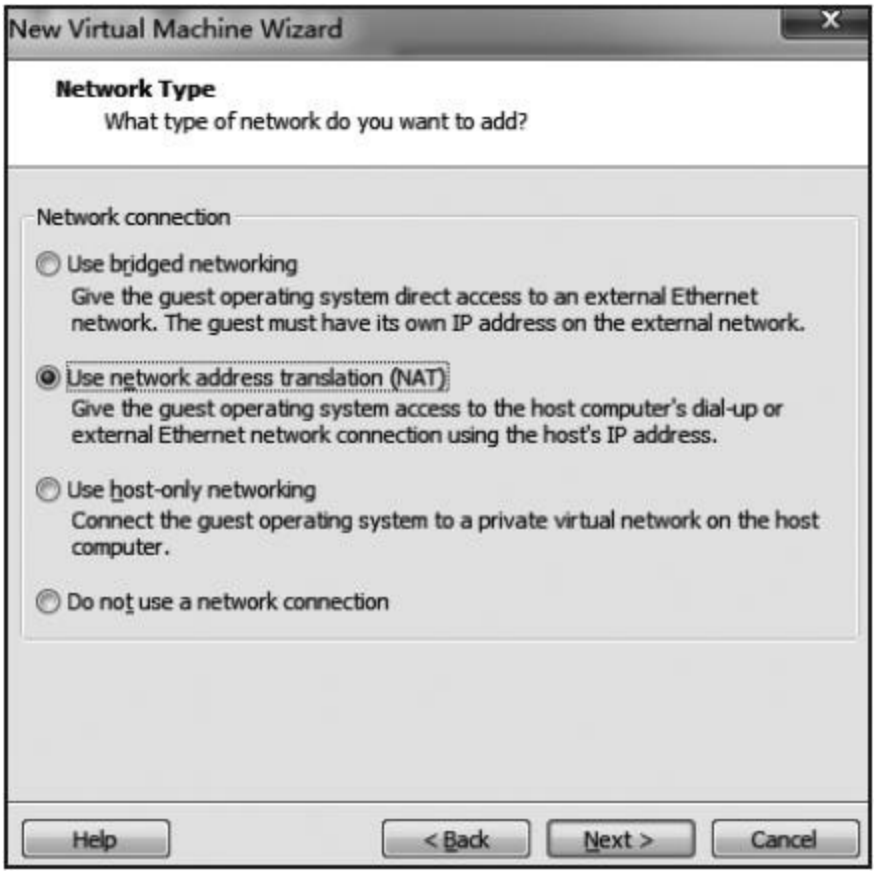


图1-17　为虚拟机选择网络类型

10）“为虚拟机选择网络类型”极其关键。VMware虚拟机常见的网络类型有Bridged（桥接）、NAT、Host-only（仅主机）3种，在分析如何选择之前，先要简单给大家介绍一下这三种网络类型。

·NAT

NAT（Network Address Translation，网络地址转换），NAT模式是比较简单的实现虚拟机上网的方式。简单地理解，NAT模式的虚拟机就是通过宿主机（物理计算机）上网和交换数据的。

在NAT模式下，虚拟机的网卡连接到宿主机的VMnet8上。此时系统的VMware NAT Service服务就充当了路由器，负责将虚拟机发送到VMnet8的包进行地址转换之后发送到实际的网络上，再将实际网络上返回的包进行地址转换后通过VMnet8发送给虚拟机。VMware DHCP Service负责为虚拟机分配IP地址。NAT网络类型的原理逻辑图如图1-18所示。

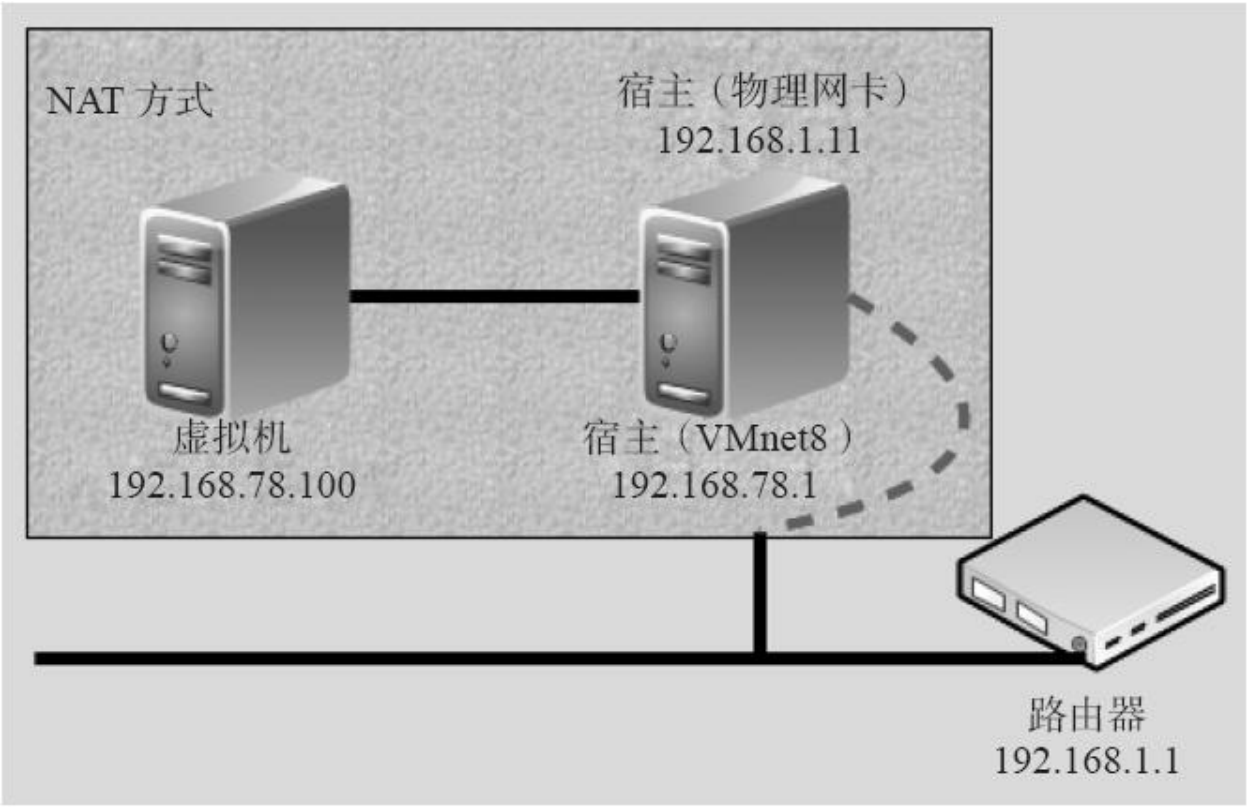


图1-18　VMware NAT网络模式原理逻辑图

NAT网络特别适合于家庭里计算机直接连接网线的情况，当然办公室的局域网环境也是适合的，其优势是不会与其他物理主机IP冲突，且在没有路由器的环境下也可以通过SSH NAT连接虚拟机学习，换了网络

环境虚拟机IP等不受影响，这是老男孩推荐的选择。

·Bridged（桥接模式）

桥接模式可以简单理解为通过物理主机网卡架设一座桥，从而连入实际的网络中。因此，虚拟机可以被分配与物理主机相同网段的独立IP，所有网络功能和网络中的真实机器几乎完全一样。桥接模式下的虚拟机和网内真实计算机所处的位置是一样的。

在Bridged模式下，计算机设备创建的虚拟机就像一台真正的计算机一样，它会直接连接到实际的网络上，逻辑上上网与宿主机（计算机设备）没有联系。Bridged网络类型的原理逻辑图如图1-19所示。

Bridged网络类型特别适合于局域网环境，其优势是虚拟机像一台真正的主机一样，缺点是可能会与其他物理主机IP冲突，并且在与宿主机交换数据时，都会经过实际的路由器，当不考虑NAT模式的时候，就选择这个桥接模式，在桥接模式下换了网络环境后所有虚拟机的IP都会受影响。

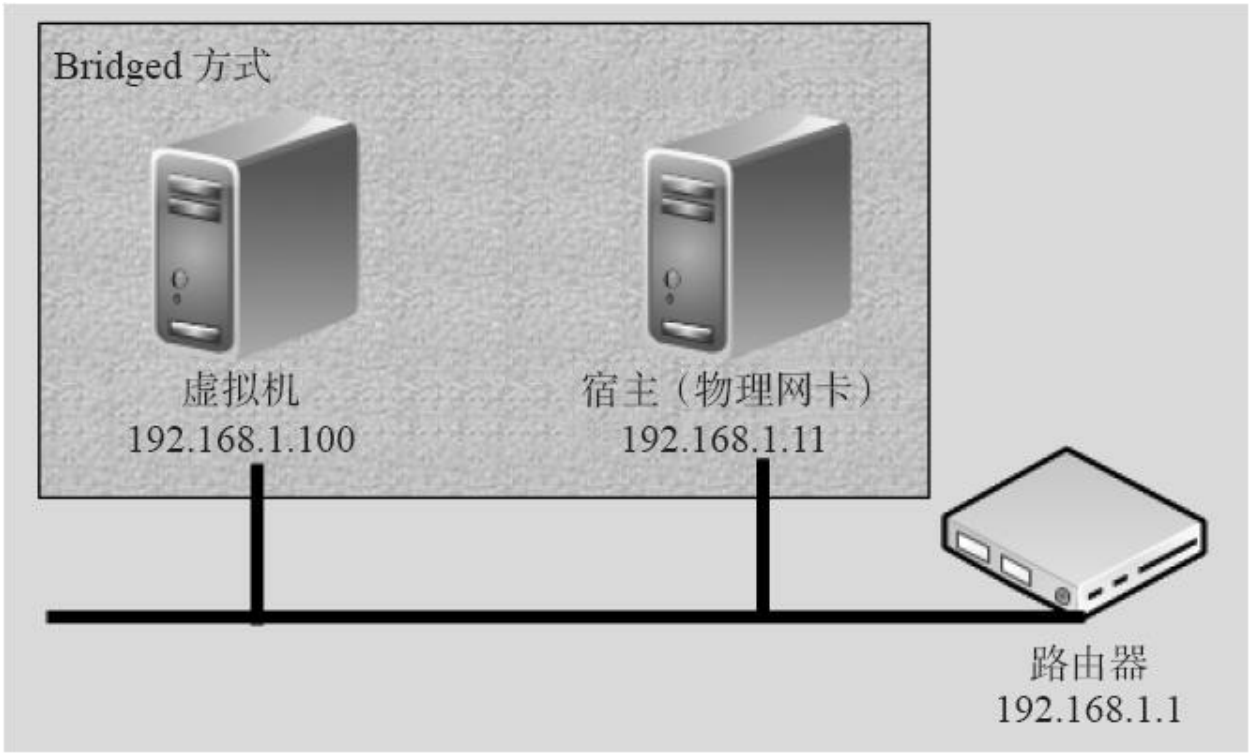


图1-19　VMware Bridged网络模式原理逻辑图

·Host-only（仅主机）

在Host-only模式下，虚拟机的网卡会连接到宿主的VMnet1上，但宿主系统并不为虚拟机提供任何路由服务，因此虚拟机只能与宿主机进行通信，不能连接到实际网络上，即无法上网。Host-only网络类型的原理逻辑图如图1-20所示。

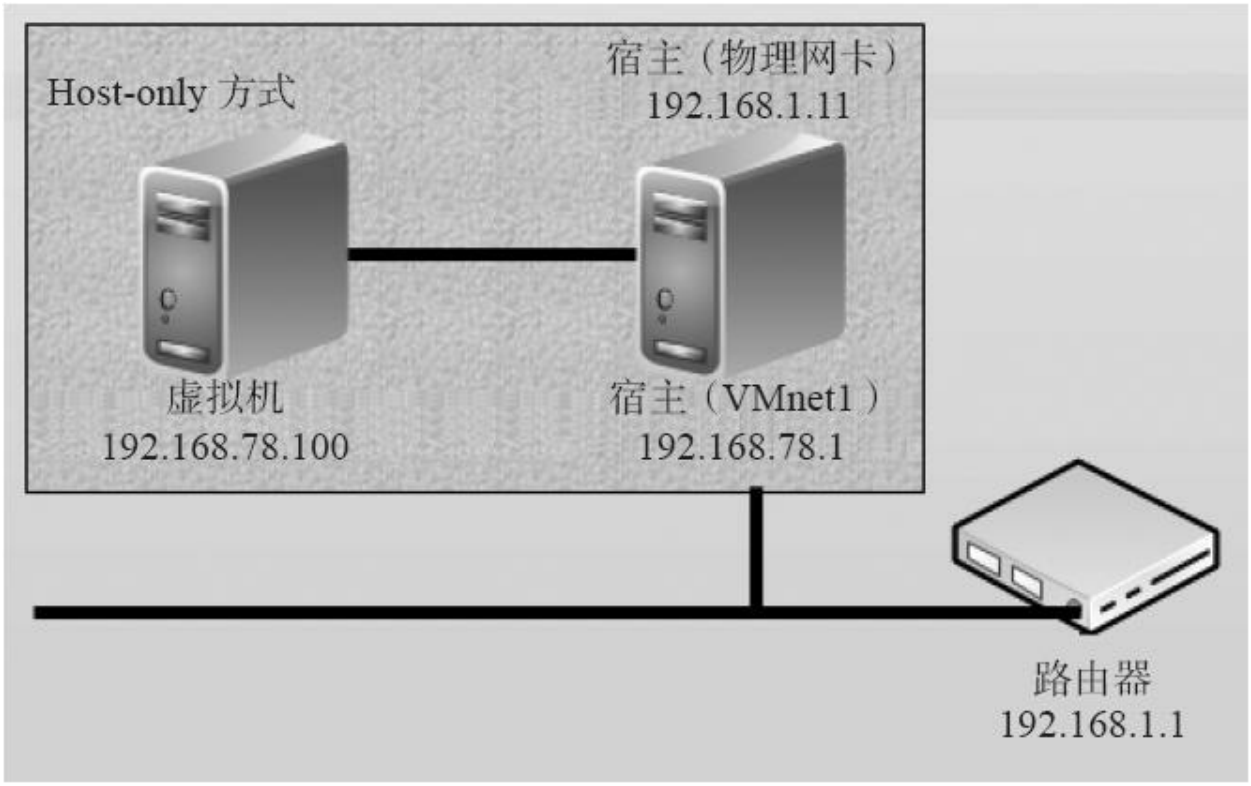


图1-20　VMware Host-only网络模式原理逻辑图

老男孩的写书环境是办公室环境，有物理路由器，不经常换网络环境，因此，这里选择以桥接模式为例为大家讲解（如图1-19所示）。在选择了网络类型后，单击Next按钮继续，出现的界面如图1-21所示。

11）在图1-22所示的界面选择虚拟机的I/O控制器类型，采用默认类型即可，选择完毕，单击Next按钮继续，出现的界面如图1-23所示。

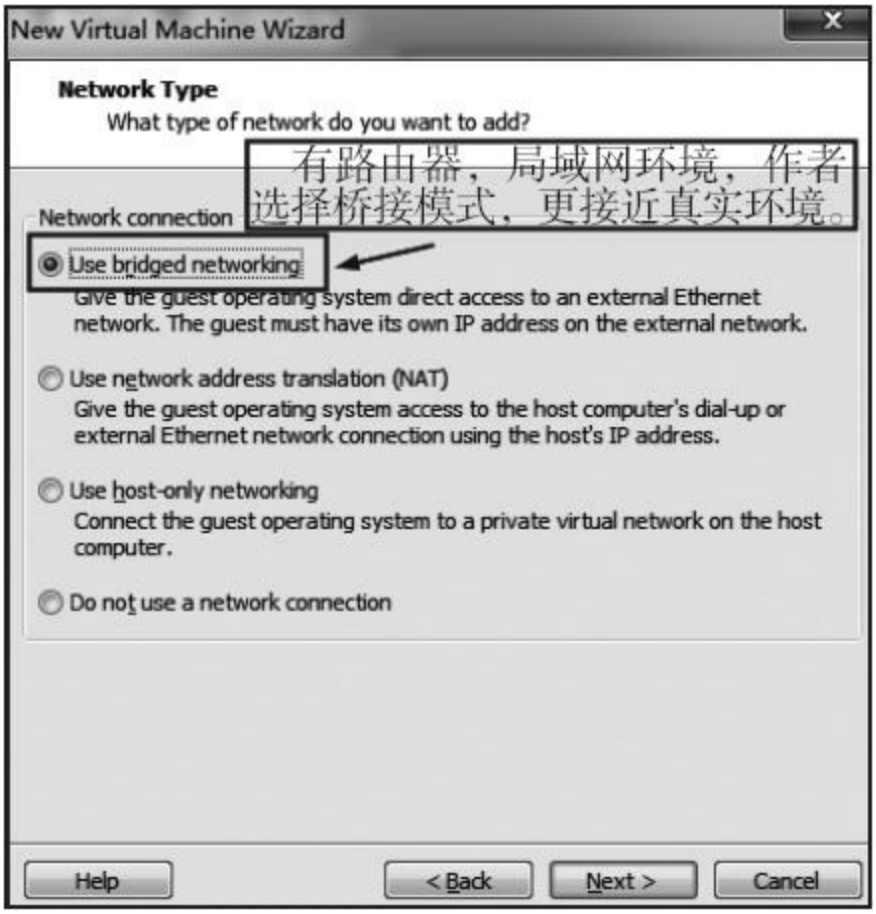


图1-21　为虚拟机选择桥接网络类型

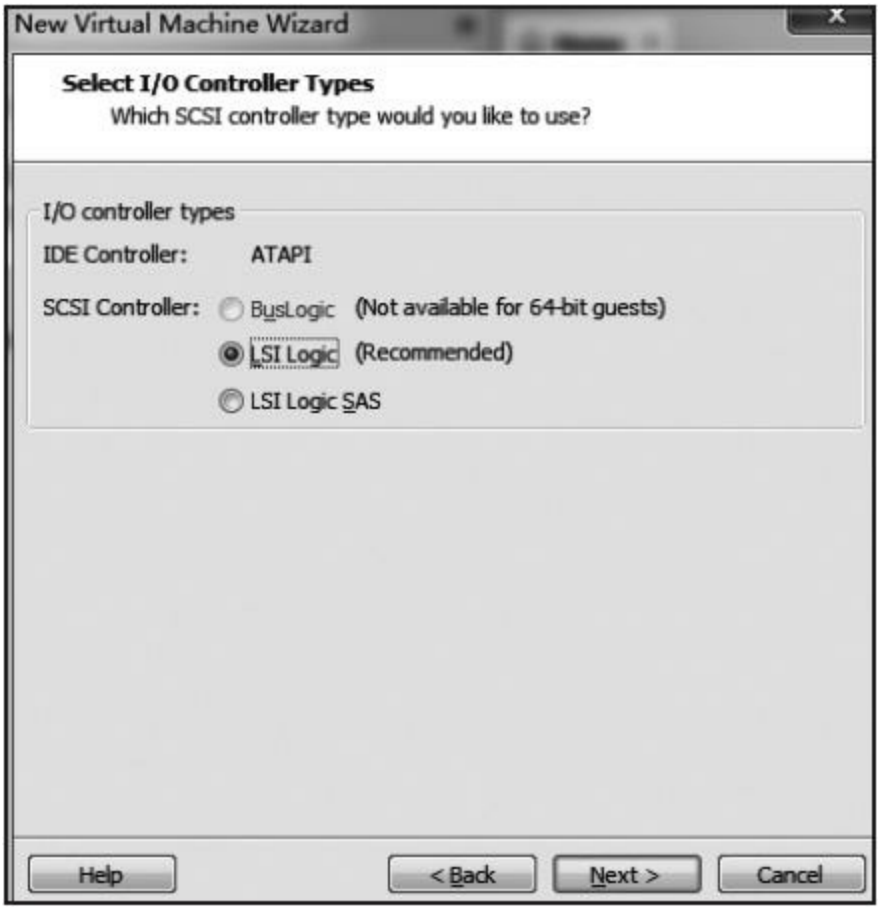


图1-22　为虚拟机选择I/O控制器类型

12）在选择虚拟机磁盘类型时，采用默认的SCSI即可（VM高版本会有SAS选项，不过建议不要选），选择完毕，单击Next按钮继续，会出现如图1-24所示界面。

图1-23　为虚拟机选择虚拟机磁盘类型

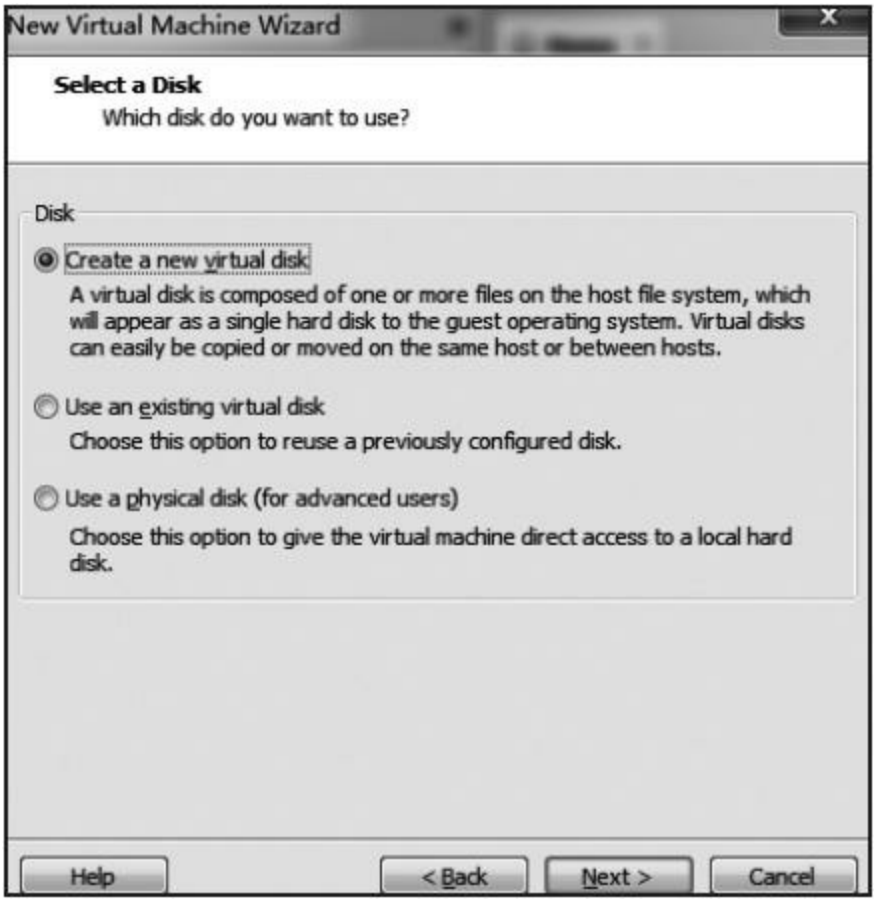


图1-24　为虚拟机选择使用的磁盘

13）在选择虚拟机使用的磁盘时，采用默认的“Create a new virtual disk”，即创建一个虚拟磁盘。选择完毕，单击Next按钮继续，会弹出如图1-25所示的界面。

14）在选择虚拟机使用的磁盘文件名时，仍然采用默认配置。选择完毕，单击Next按钮继续，出现的界面如图1-26所示。

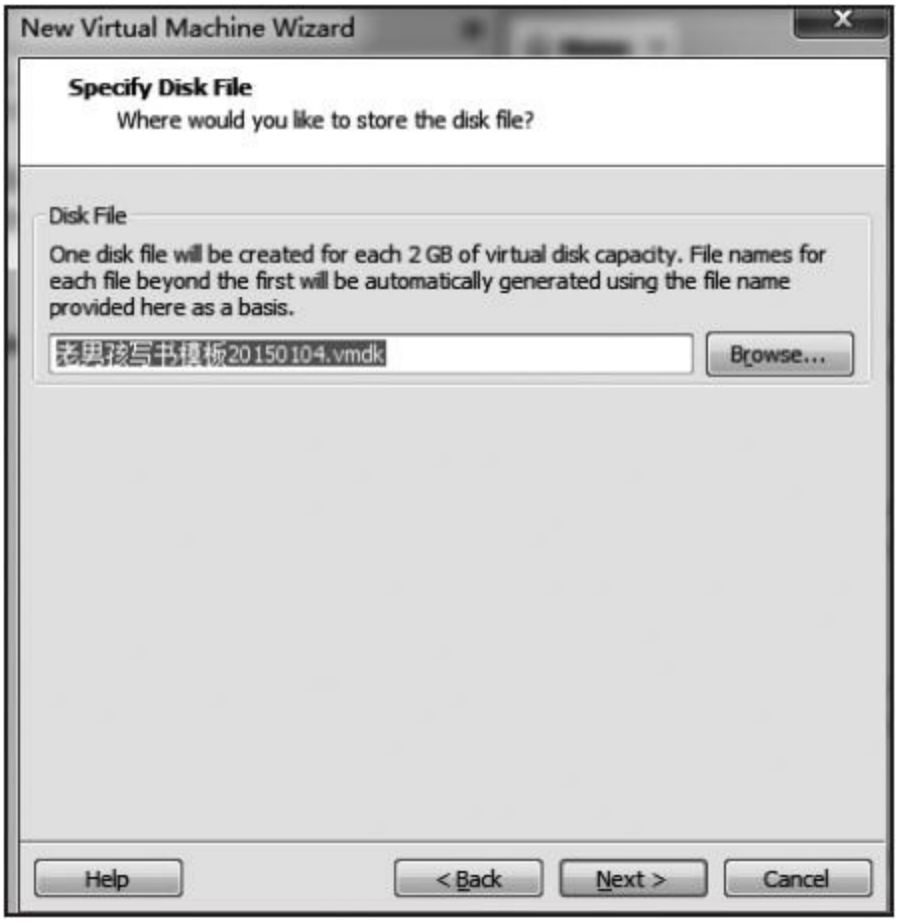


图1-25　为虚拟机选择磁盘文件

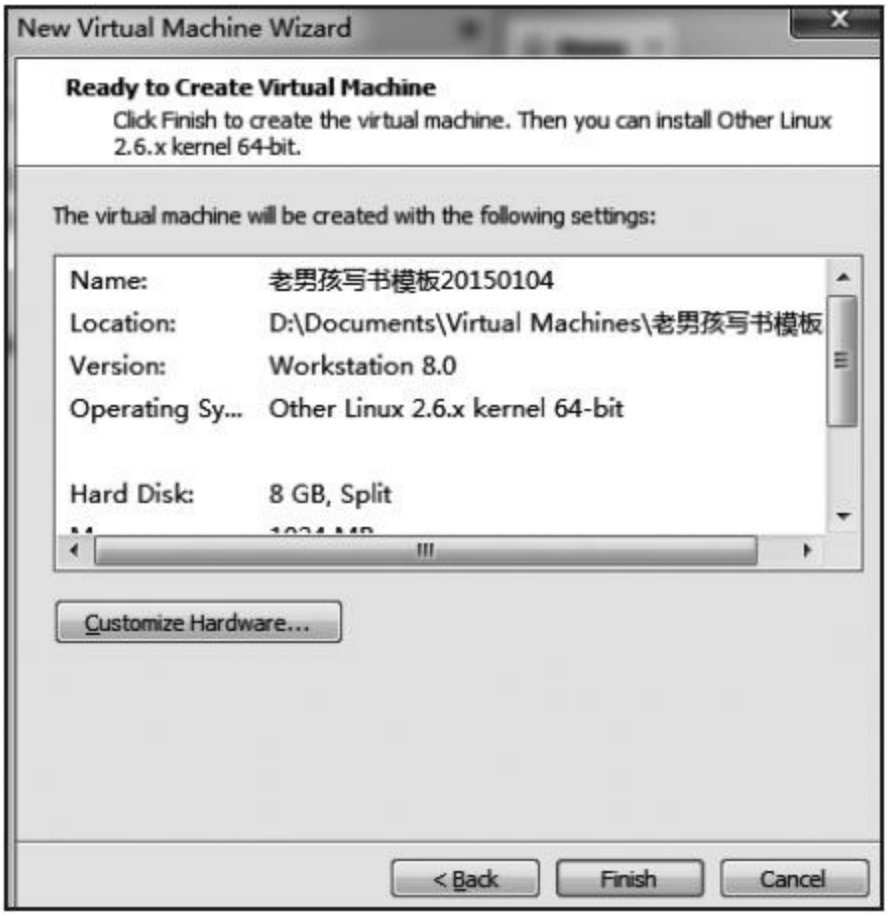


图1-26　显示虚拟机所有配置选项信息

15）如图1-26所示界面用于显示配置的虚拟机的所有选项信息，可以通过滚动条下拉查看，单击左下方的“Customize Hardware”可以自定义添加其他硬件，如磁盘、网卡等，这里保留默认。选择完毕，单击Next按钮继续，会出现如图1-27所示界面。

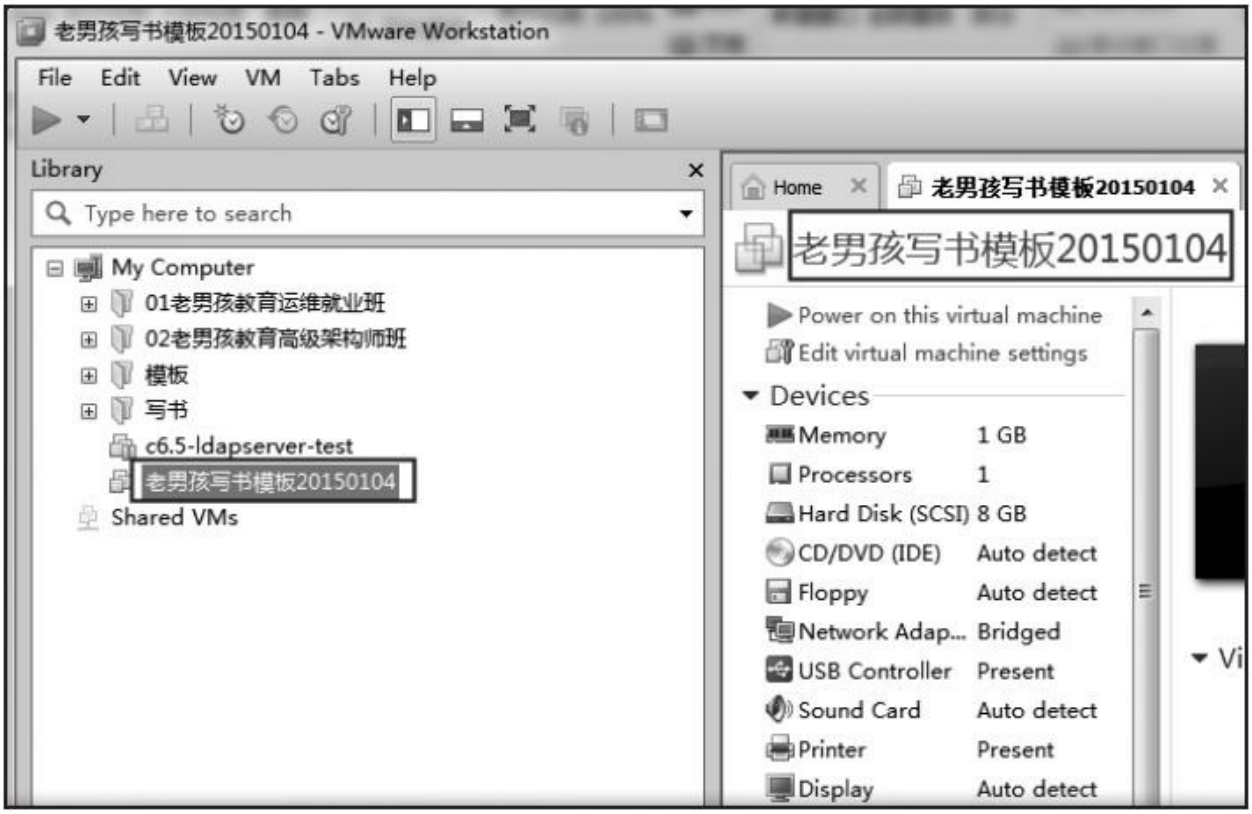


图1-27　创建虚拟机后的主界面

16）图1-27所示的界面中展示的是创建虚拟机后的界面信息，左边是虚拟机的名称，右边是虚拟机的实际配置。在这个窗口，可以为创建好的虚拟机配置光驱，把CentOS镜像文件（CentOS-6.6-x86_64-bin-DVD1.iso，需要单独下载，如果官方网站上没有CentOS 6.6，下载CentOS 6.7也可以，只要是6系列的都可以）放到光驱里就可以启动虚拟机安装Linux系统了。点选窗口右边的光驱选项，出现的界面如图1-28所示。

图1-28　配置CentOS 6.6 ISO镜像文件载入光驱

17）接下来就剩下启动虚拟机安装系统了。单击“Power on this virtual machine”即可，如图1-29所示。

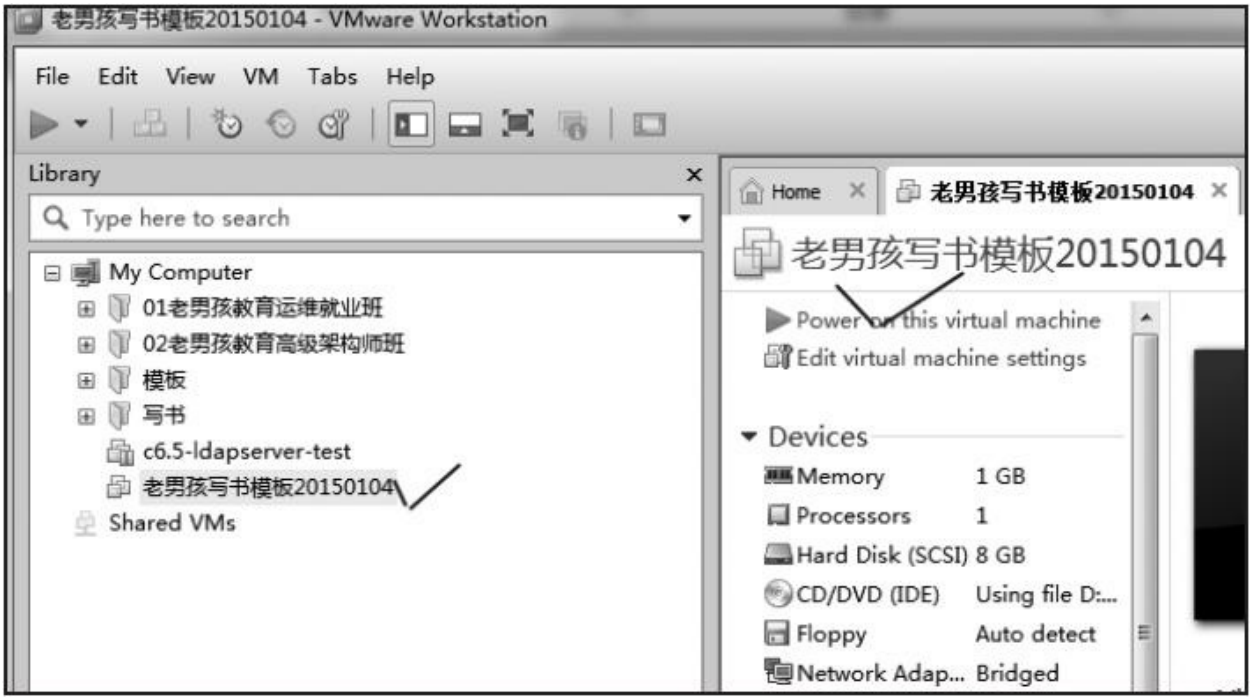


图1-29　准备启动虚拟机安装CentOS 6.6界面

启动虚拟机后可能遇到的问题及应对方法

在老男孩实际教学过程中，发现有些同学的笔记本电脑默认使用VMware软件创建虚拟机后，在虚拟机中不支持64位CentOS系统的安装，如图1-30所示。

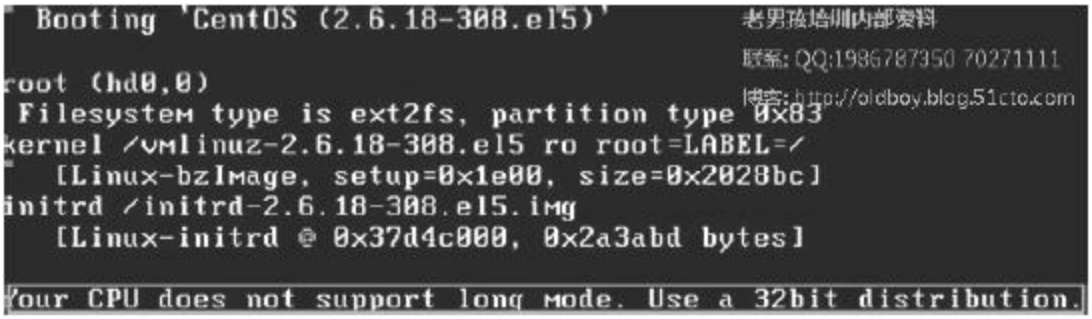


图1-30　虚拟机开机提示不支持CentOS 6.6 64位系统

解决方法：

进入笔记本电脑或台式机的BIOS，找到类似下面的选项进行调整，不同的计算机会略有不同。这里以Thinkpad计算机为例说明，具体情况如图1-31和图1-32所示。

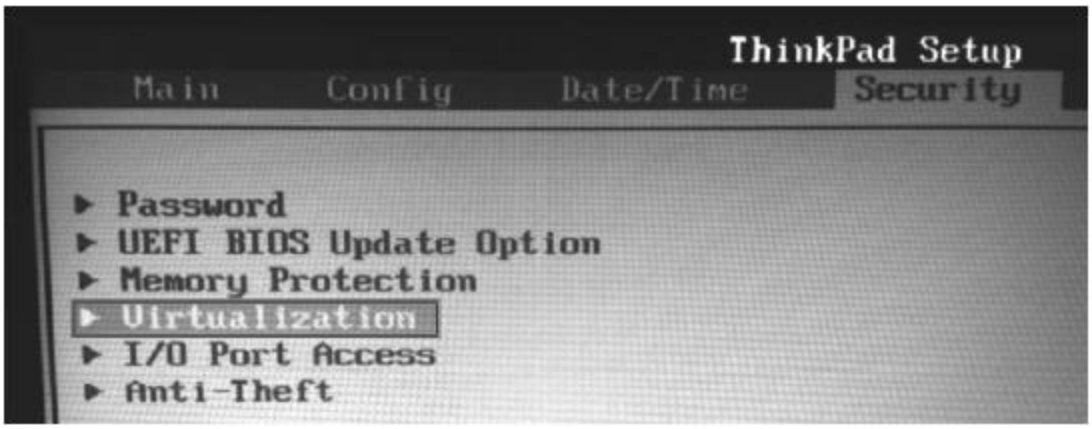


图1-31　调整宿主机（计算机）BIOS，支持64位系统安装

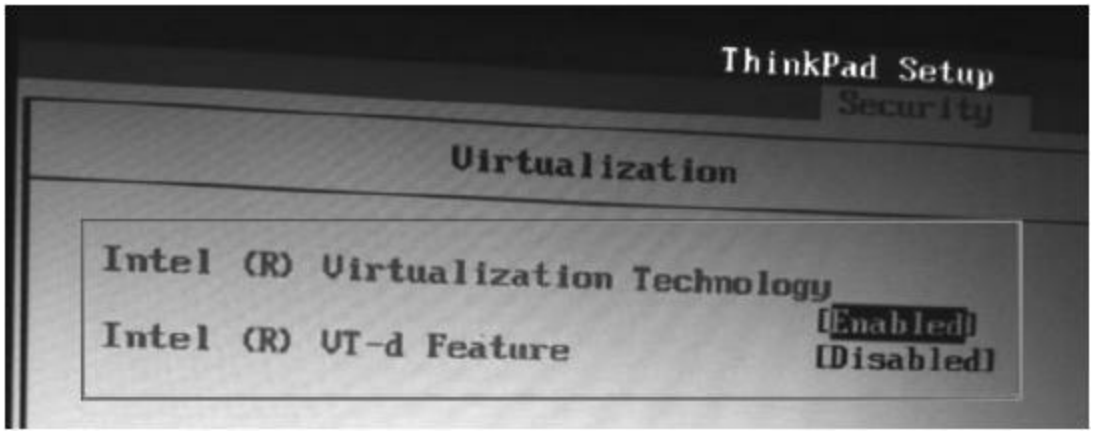


图1-32　调整宿主机（计算机）BIOS，支持64位系统安装

在图1-32中，两个选项都改成Enabled，然后保存。如果没有意外，就可以安装64位操作系统了。如果重启计算机后依然不行，那就是计算机硬件有问题了，只能安装32位系统，或者更换为支持64位系统的硬件。

到此为止，VMware虚拟机的准备工作就全部做完了，在下一章将为大家讲解安装Linux系统的相关知识。

1.8 本章重点回顾

1）了解什么是操作系统及操作系统简单原理图。

2）了解UNIX的发展历史。

3）了解市场上的常见UNIX系统版本。

4）了解UNIX及Linux诞生发展的几个关键人物。

5）重点了解GNU、GPL的知识。

6）了解Linux系统的特点。

7）了解Linux系统的常见发行版本，不同场景选择。

8）重点了解CentOS和Red Hat的区别和联系。

9）了解CentOS各个版本的应用场景及企业应用情况。

10）学会搭建学习Linux的环境。

注意：最好能口头表达出上述内容。

1.9 本章知识相关考试题

1）请详细描述GNU的相关知识和历史事件。

2）请描述什么是GPL及GPL的内容细节。

3）企业工作中如何选择各Linux发行版？

4）Red Hat Linux和CentOS Linux有什么区别和联系？

5）请说出你认为Linux受欢迎的3个以上特点。

1.10 本章参考资料

·操作系统介绍资料

http://baike.baidu.com/view/880.htm

·自由软件基金会

http://www.gnu.org/philosophy/free-sw.html

·GNU与GPL知识

http://www.gnu.org/home.zh-cn.html

·GPL协议英文版

http://www.gnu.org/licenses/gpl.html

·虚拟机及虚拟机软件的知识

http://zh.wikipedia.org/wiki/%E8%99%9A%E6%8B%9F%E6%9C%BA

·老男孩针对本章内容给出的对应视频精品资料

http://edu.51cto.com/course/course_id-839.html

第2章　企业级CentOS 6.6操作系统安装

2.1　下载CentOS系统ISO镜像

2.1.1　下载CentOS系统ISO镜像的说明

要安装CentOS系统，就必须有CentOS系统软件安装程序，可以通过浏览器访问CentOS的官方站点http://www.centos.org ，然后在导航栏找到“Downloads→Mirrors”链接，单击进入后即可下载，或者打开如下的地址直接选择国内的高速镜像站点进行下载：

·64位：http://mirrors.aliyun.com/centos/6.6/isos/x86_64/

·32位：http://mirrors.aliyun.com/centos/6.6/isos/i386/


提示：如果CentOS 6.6的下载地址过期无法下载，可以直接下载CentOS 6.7或更新的6系列其他版本，方法是输入http://mirrors.aliyun.com/centos/ 进入地址后进行选择。

目前，国内比较稳定的开源镜像软件下载地址为aliyun镜像地址（http://mirrors.aliyun.com/ ）和网易镜像地址（http://mirrors.163.com/ ），推荐国内用户使用这两个站点提供的镜像，直接打开CentOS的官方

站点会比较慢，甚至会打不开。

下载完成后，得到的是CentOS操作系统的ISO系统软件安装程序，该程序分为32位和64位两种版本，文件主要为DVD格式（早期还有CD格式），扩展名为.iso，软件程序文件名见表2-1。

表2-1　软件程序文件格式

文件格式	i386（32 位系统）	x86_64（64 位系统）
DVD 格式	CentOS-6.6-i386-bin-DVD1.iso CentOS-6.6-i386-bin-DVD2.iso	CentOS-6.6-x86_64-bin-DVD1.iso CentOS-6.6-x86_64-bin-DVD2.iso


提示：目前绝大多数企业生产环境，用的都是64位DVD格式的系统镜像。

2.1.2　下载后有关ISO镜像的使用说明

正如前面提到的，扩展名为“.iso”格式的操作系统文件就是所谓的镜像文件。这种镜像文件一般是用来刻录光盘的，文件个头比较大，单个CD文件可能就有700MB之多，而单个DVD文件可能会高达4GB大小，因此建议不要使用浏览器（例如IE/Firefox）下载，因为中途一旦断网就会前功尽弃。推荐使用类FTP客户端程序或迅雷等下载工具来下载，例如Filezilla，这样就不需要担心因断网而前功尽弃的问题了。曾经老男孩就因为心存侥幸而吃了大亏，白白浪费了许多时间。

需要说明的是，这种ISO镜像文件不能直接以数据格式刻录成为CD/DVD，这样是无法实现引导安装系统的！必须要用刻录程序的镜像刻录功能，将ISO文件以“镜像文件格式”刻录成CD/DVD盘才行。

那么，在不同的场景，该如何使用ISO镜像呢？这里给大家几个不同场景下的使用建议：

·如果是在单机物理服务器上安装系统，可能需要购买空白DVD光盘刻录成DVD。当然，制作U盘镜像引导安装也是可以的。

·如果是在VMware/virtualbox等虚拟软件环境下学习，则不需要刻录成DVD光盘或制作成U盘。在创建完虚拟机后，直接指定ISO镜像文

件路径到虚拟机的光驱，即可实现引导安装系统。

·如果是服务器数量比较多的工作环境，一般也不需要刻录成DVD光盘或制作U盘，大多数都会通过ISO镜像部署无人值守网络批量安装系统服务。

若是确定要刻录光盘，老男孩也有两个建议：

·推荐使用DVD格式的ISO文件来进行刻录，这样只需一两张DVD就够了，多数情况下刻录第一个镜像文件即可，第二个镜像文件很少用到，因为我们在安装系统时都采用最小安装原则，第一个镜像文件的软件包就足够了；如果是U盘引导，配置也不难，这里就不多介绍了，请读者参阅相关资料或老男孩的博文。

·不建议使用CD格式的ISO文件来刻录，因为那样需要七八张CD盘，很费劲。截止到本书写作时，aliyun中也已看不到CD格式的镜像文件了。

安装Linux系统的常见引导方式有如下几种：

·光盘引导安装。

·U盘引导安装。

·网络安装（需要网卡支持，现在主流网卡都支持）。

虚拟机环境可以直接使用ISO镜像，安装方式可以是上面三种方式中的任何一种。

2.1.3 为什么企业环境要选择64位操作系统

目前绝大多数企业生产环境中，使用的都是64位CentOS系统，那么32位与64位系统到底有什么不同？为什么要选择64位系统呢？这就要从32位与64位系统的定位和区别讲起。

区别之一：当初设计时的定位不同。64位操作系统的设计定位是满足机械设计和分析、三维动画、视频编辑和创作，以及科学计算和高性能计算应用程序等领域，这些应用领域的共同特点就是需要有大量的系统内存和浮点性能。简单地说，64位操作系统是为高科技人员使用本行业特殊软件的运行平台而设计的，而32位操作系统是为普通用户设计的。

区别之二：安装要求配置不同。64位操作系统只能安装在64位计算机上（CPU必须是64位的），并且只在针对64位的软件时才能发挥其最佳性能。32位操作系统既可以安装在32位（32位CPU）计算机上，也可以安装在64位（64位CPU）计算机上。当然，此时32位的操作系统是无法发挥64位硬件性能的。

区别之三：运算速度不同。64位CPU GPRs（General-Purpose Registers，通用寄存器）的数据宽度为64位，64位指令集可以运行64位数据指令，也就是说处理器一次可提取64位数据（只要两个指令，一次

提取8字节的数据），比32位提高了一倍（32位需要4个指令，一次只能提取4字节的数据），性能会相应提升。

区别之四：寻址能力不同。 64位处理器的优势还体现在操作系统对内存的控制上。由于地址使用的是特殊整数，因此一个ALU（算术逻辑运算器）和寄存器可以处理更大的整数，也就是更大的地址。比如，Windows 7 x64 Edition支持高达128 GB的物理内存和16TB的虚拟内存，而32位的CPU和操作系统理论上最大只可支持4GB的内存，实际上也就是3.2GB左右的内存，当然32位操作系统是可以通过扩展来支持大内存的，扩展所采用的是PAE技术。

 提示：若要用一句话概括32位与64位操作系统的区别，那就是64位操作系统的CPU运算速度更快，支持更大的内存使用，可以发挥更大更好的硬件性能，提升业务工作效率。

2.1.4 如何区分已安装的系统是32位还是64位

在Linux系统中查看系统版本为32位还是64位的方法如下。

方法1为标准的查看方法。命令及输出如下：

```
[root@www ~]# uname -m
x86_64
```

上述输出结果中带有x86_64字样，说明该系统为64位。再来看下面的命令及输出：

```
[oldboy@web ~]$ uname -a
Linux web 2.6.18-164.el5 #1 SMP Thu Sep 3 03:


33:


56 EDT 2009 i686 i686 i386 GNU/Linux
```

上述输出结果中带有i386/i686字样，说明该系统为32位。

还可以通过命令uname-a查看更多的信息。

方法2为网友寻思出的方法。命令及输出如下：

```
 [root@www ~]# ls -d /lib64
/lib64
```

输出结果中存在lib64目录，则表示为64位的系统。

2.1.5　在学习与工作中如何选择操作系统

1.工作场景

当前，绝大多数服务器厂商如Dell、HP、IBM的服务器既支持32位又支持64位系统。在老男孩的生产场景中，为了发挥硬件的最佳性能，尤其是对大内存的利用率，6年前就已经是100%使用64位操作系统了。目前，在正式的生产环境使用32位系统的企业已寥寥无几了，甚至绝大多数IT人员的笔记本电脑安装的Windows系统都是64位系统了。

2.学习场景

对于初学Linux的读者来说，使用32位或64位的系统都是可以的（两者会略有区别，但不是很大），但还是建议大家尽可能地用64位的系统，与企业应用环境保持一致，这样才能提升学习效率与学习效果。老男孩在工作及教学中发现，有部分硬件（相对较老）默认无法支持VMware等64位的虚拟化软件，个别初学者甚至在尽可能地调整计算机的BIOS设置及CPU虚拟化支持选项后，仍无法支持VMware的64位虚拟能力，对于这样的硬件，就只能使用32位的系统了。虽然这对配置个别网络服务有一点点影响，但其实对于学习来说，问题不太大，可以忽略不计。当然，如果手头宽裕，还是建议购买支持64位CPU虚拟化能力的计算机为好，新的主流计算机都是支持64位CPU虚拟化能力的。

有关调整计算机BIOS及CPU虚拟化支持选项的步骤，在第1章安装虚拟化软件VMware时已经讲解过，这里不再重复。

2.2 CentOS 6.6操作系统安装准备

2.2.1 单台物理服务器安装系统准备

对于单台物理服务器，在正式安装操作系统之前，需要先确认以下两个问题，以便能够顺利安装系统。

·服务器光驱和系统安装光盘（需要把ISO文件刻成DVD光盘）都可用。

·服务器的各个硬件都能被CentOS 6系统支持（当前市场上大多数品牌的服务器都已支持CentOS 6）。

确认完毕，就可以使用DVD光盘安装系统了，若考虑采用U盘安装的方式，请读者自行学习。

2.2.2　虚拟机学习安装系统准备

作为初学者，在安装CentOS 6系统之前，通常会在Windows系统上事先安装好虚拟机软件，例如VMware，然后打开运行安装的虚拟机软件，选择创建一个虚拟机，并配置好硬件，最后找到与创建的这个虚拟机对应的虚拟光盘驱动器。准备就绪，就可以将CentOS系统的ISO镜像文件加载进来了。图2-1即为VMware虚拟机加载CentOS 6.6 ISO镜像文件的界面。在该界面右侧，将默认的物理光驱“Use physical drive”，改为使用ISO镜像文件“Use ISO image file”，然后单击右边的“Browse...”，找到事先下载好的CentOS 6.6的第一个镜像文件并加载进来即可。

提示：安装系统时，绝大多数情况下只需要使用第一个ISO镜像文件，因为在企业实际环境中，安装系统时绝大多数都采用最小化安装原则，第二个镜像文件很少用到。

图2-1　VMware虚拟机加载ISO镜像图

2.3　开始安装CentOS 6.6操作系统

2.3.1　安装CentOS 6.6操作系统的过程

加载完CentOS 6.6的第一个ISO镜像文件后，开启/重启计算机或虚拟机，此时，系统会进行自检，自检完毕就会出现安装系统时的引导界面，如图2-2所示。

图2-2　CentOS 6.6开机安装系统引导界面

提示：如果是在虚拟机上安装CentOS系统，需要进入虚拟机界面操作，如果需要退出虚拟机界面，可以按快捷键Ctrl+Alt。

1.选择系统引导方式

在图2-2所示的引导界面中，可以看到共有5种引导方式，这5种方式对应的中英文含义见表2-2。

表2-2　CentOS 6.6的5种引导方式

序号	CentOS 6.6 的引导方式	解释说明
1	Install or upgrade an existing system	安装新系统或升级已经存在的系统
2	Install system with basic video driver	安装带有基本视频驱动程序的系统
3	Rescue installed system	修复已经安装的系统（故障修复）
4	Boot from local drive	从本地驱动器引导系统
5	Memory test	内存测试

其中，第一种（新服务器安装或已有服务器升级）和第三种（系统故障恢复）引导方式比较有用，其他几个选项用途不是很大，可以忽略。

此处选择第一项“Install or upgrade an existing system”，即默认选项，然后按Enter（回车）键进入图形安装界面。咦，为什么不是文本方式呢？这是因为CentOS 6无法用文本模式完全定制化安装，即在文本模式下无法定制磁盘分区，这是CentOS 6与之前各版本的不同之处。因此，在安装CentOS 6的系统时，通常都选择图形安装。

2.检查安装光盘介质

确定引导方式后，进入如图2-3所示的界面，如果需要检查光盘介质，选择“OK”，否则按Tab键选择“Skip”，这里直接按Tab键选

择“Skip”，然后按Enter键继续。

图2-3　检查安装光盘介质图

提示：检查光盘介质所用的时间一般都比较长，而且必要性不大。因此，若能保证光盘是正常的，建议略过检查。

去机房之前可准备多套系统光盘，防止因光盘介质有问题，影响系统安装。当然，还可以通过U盘或网络方式安装系统，这样就不需要光盘了。如果是虚拟机安装系统就更不需要检测介质了，如果介质有问题，那就是ISO文件的问题，重新下载就好了。

3.进入安装下一步界面

忽略光盘介质检查后，进入如图2-4所示的界面。要特别注意，如果采用的是VMware Workstation创建的虚拟机，其安装界面有可能会显

示不完整，例如：可能看不见界面里的“Next”按键。此时，可通过按VMware窗口的最大化按钮（如图2-5所示），使界面最大化来解决。然后单击“Next”继续。

图2-4　调整VMware显示最大化后的安装界面图

图2-5　调整VMware最大化显示安装界面

提示：Ctrl+Alt快捷键用于从虚拟机内向虚拟机外切换，系统也会提示这个快捷键的，如果已经进入窗口，需要按VMware窗口的最大化按钮就要使用Ctrl+Alt快捷键退出来再单击。

4.安装过程语言选择

进入图2-6所示的语言选择界面后，可对安装过程语言进行选择，这里保留默认选项“English”，单击“Next”继续。

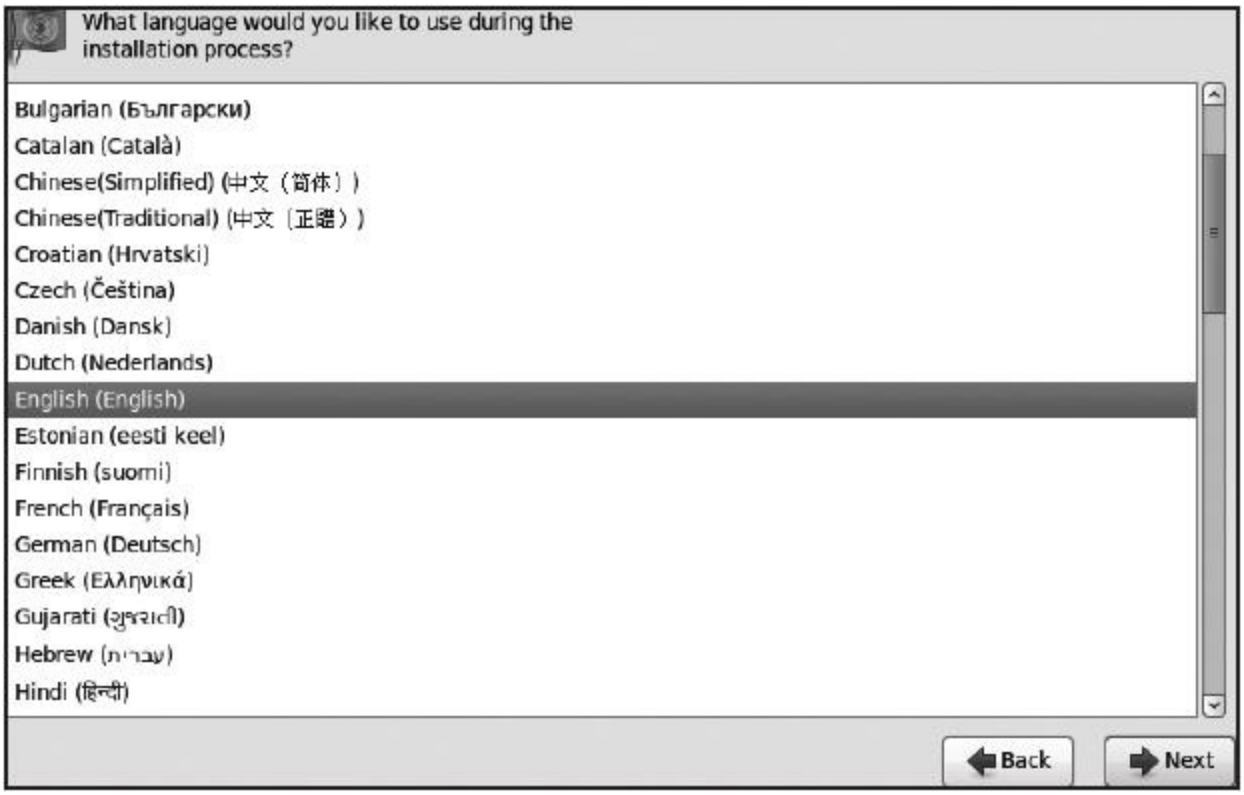


图2-6　安装过程语言选择界面图

提示：此处仅仅是安装过程中的语言显示，不建议选中文，要学会多亲近英文，至少不要逃避，只有这样才能学好Linux运维。

5.选择键盘布局

进入如图2-7所示的选择键盘布局界面后，选择保留默认选项“U.S.English”，单击“Next”继续。

6.选择合适的物理设备

进入如图2-8所示的选择合适的物理设备界面后，会看到两个选项。如果是普通的服务器，默认选择第一个“Basic Storage Devices”即可，第二个是用于特殊存储设备的，例如SANs（ISCSI）等。选项的含义，英文已经解释得非常详细了，不再赘述。这里保留默认选项“Basic Storage Devices”，单击“Next”继续。

图2-7　选择键盘布局图

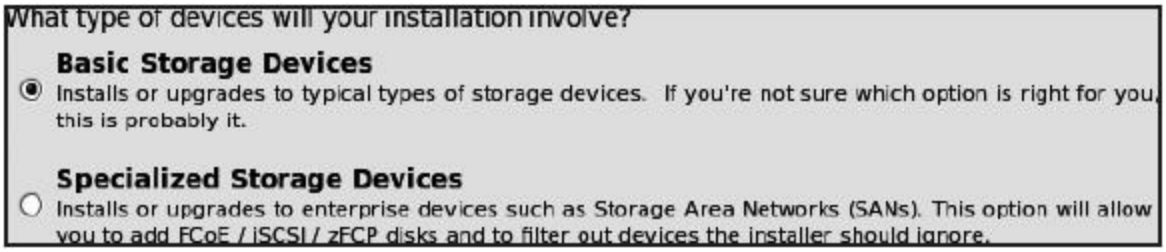


图2-8　选择合适的物理设备

7.初始化硬盘提示

进入如图2-9所示的初始化硬盘警告界面后，会看到相应的警告信息，单击“Yes，discard any data”继续。

提示：这一步会格式化服务器的硬盘，即删除硬盘内的所有数据，要确认物理硬盘内的数据是否有用。如果是虚拟机就是格式化虚拟磁盘（不会丢失虚拟机所在的宿主机的数据）。

8.初始化主机名及配置网络

现在，进入如图2-10所示的“初始化主机名及配置网络”界面。

图2-9　初始化硬盘警告

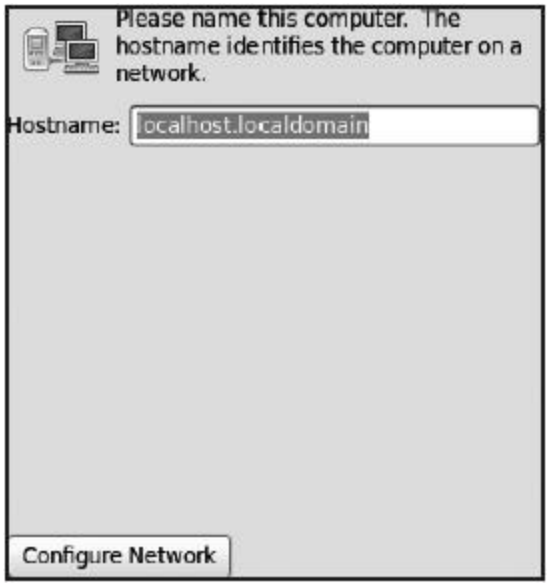


图2-10　初始化主机名及配置网络

（1）为系统设置主机名

在图2-10中，左上角的“Hostname”表示配置主机名。在右边对应的选框里，会有默认的localhost.localdomain主机名，删除之，设置自己的主机名，这里用www作为主机名。

提示：不建议保留默认Hostname主机名，设置一个规范的主机名，会显得更专业，这是运维的原则，而且也可避免对后面业务服务的安装配置产生影响。自定义主机名时可以使用普通的字符串（例如www），或者完整的FQDN名（例如www.etiantian.org ）。最好由简单字母或字母带数字字符（以字母开头，不要只用数字，可带下划线等）组成，不建议用特殊字符。

（2）配置网卡及连接网络（可选）

单击图2-10左下角的“Configure Network”按钮，会弹出一个网络连接（Network connections）的窗口，在弹出的窗口中选择“System eth0”，然后单击右边的“Edit”按钮，此时会弹出一个“Editing System eth0”的窗口，在该窗口勾选“Connect automatically”复选框，设置eth0网卡自动连接，并选择“IPv4 Settings”，然后根据自己的网络情况配置对应的IP地址、子网掩码、网关和DNS。配置好后，如果希望保存，可以单击“Apply”使配置生效，完整的配置过程如图2-11所示。


提示：介绍这一步骤只是为了让读者先对网卡及网络配置方法有一个大概了解，这里先忽略，保留到系统安装后再来配置。

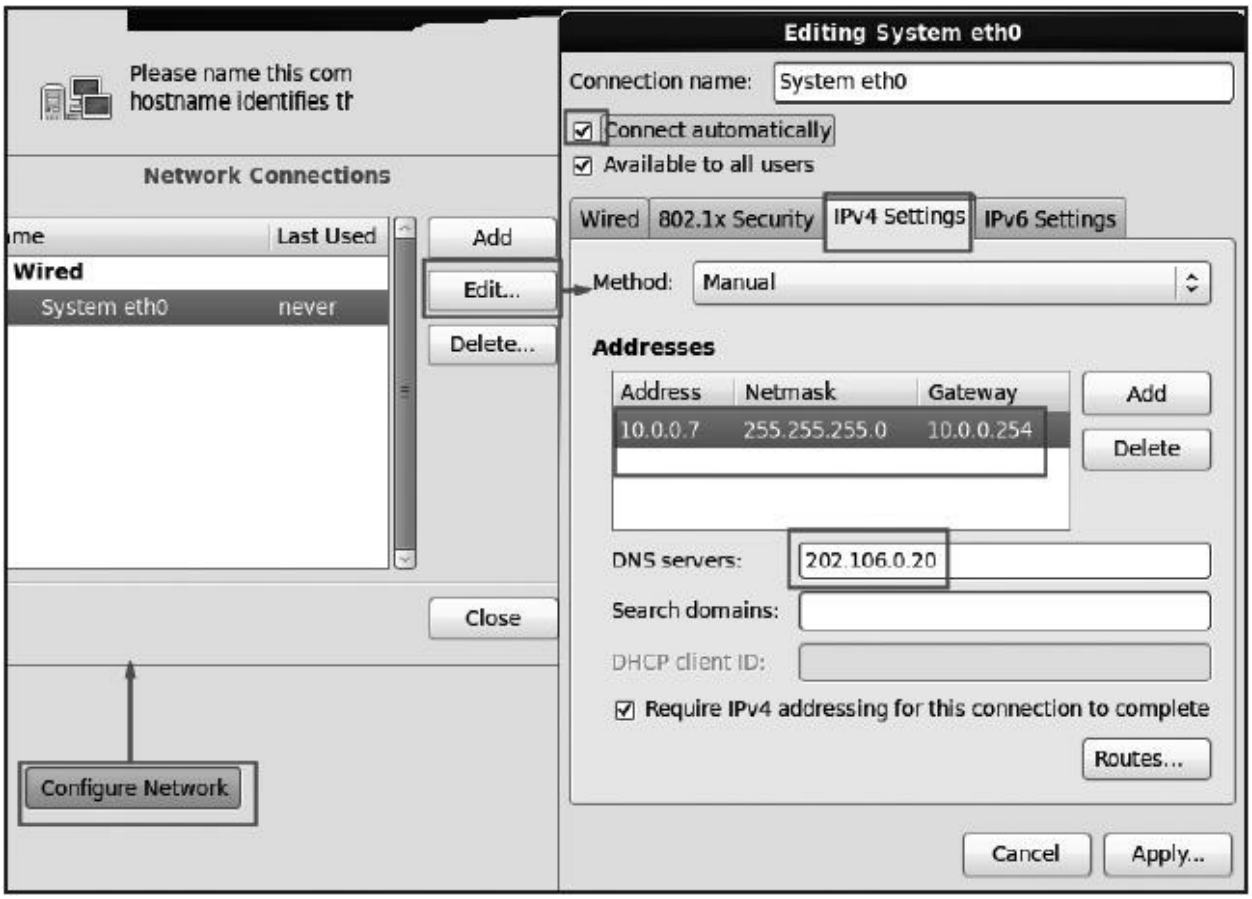


图2-11　配置网卡及连接网络（可选）

9.系统时钟及时区设置

在如图2-12所示的“系统时钟及时区设置”界面中，选择“Asia/Shanghai”，然后取消“System clock uses UTC”前的对勾，最后选择“Next”继续。

图2-12　系统时钟及时区设置

提示：这里的时区要注意一下。不同的时区会有不一样的日期/时间显示。这可能会造成业务数据的时间不一致，所以，要为系统选择正确的时区才行，建议选择“亚洲/上海”。要特别注意的是，复选框“System clock uses UTC”与所谓的夏令时有关，不必选择，不然可能

会对时区造成影响，导致系统显示时间与本地时间不同，当然这些都是可以在安装完成以后再根据需要修改的。

10.设置超级用户root口令

在如图2-13所示的界面中，输入两次root用户的口令，然后单击右下角“Next”继续。

提示：如果是生产环境，root口令要尽量复杂。比如，设置8位以上包含数字、字母大小写，甚至是特殊字符的口令。在企业运维工作中，安全是至关重要的一环，要从每一件小事做起。

如果密码过于简单会出现如图2-14所示的提示。可以用“Use Anyway”强行设置简单的密码，但不推荐，如果是学习环境，还是建议使用简单密码，否则经常忘记密码也是一件不舒服的事情，很多新手都会遇到这个问题。

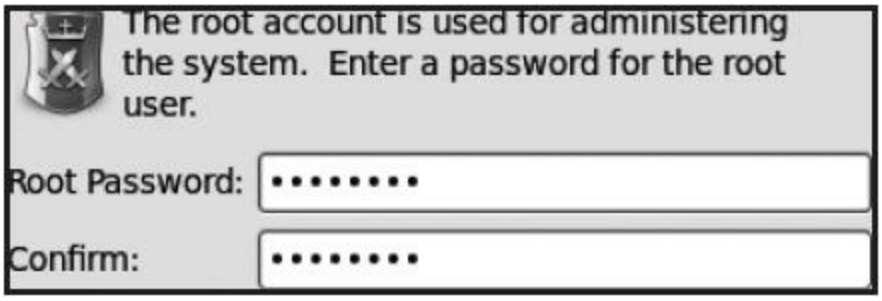


图2-13　root用户口令设置

图2-14　密码过于简单的提示

2.3.2　磁盘分区类型选择与磁盘分区配置过程

1.选择系统安装磁盘空间类型

经过上一节的步骤后，会进入如图2-15所示的磁盘分区方式选择界面。

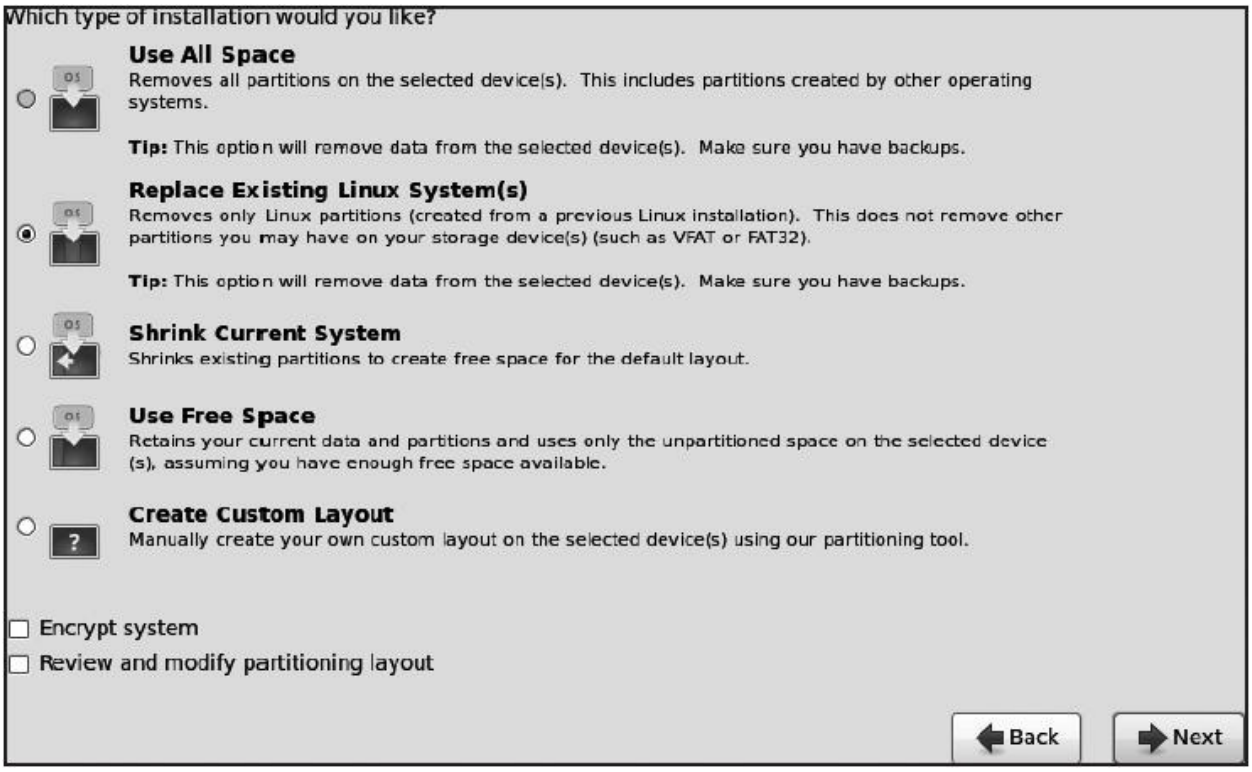


图2-15　系统磁盘空间选择

CentOS 6.6磁盘有如下五种分区方式。

·Use All Space：删除当前磁盘内的所有分区，包括其他系统创建的

分区。

·Replace Existing Linux System（s）：删除当前磁盘内的所有Linux分区，而不删除其他系统创建的分区，这是默认的选项 。

·Shrink Current System：利用（挤压）分区上存在的所有空闲空间，创建系统默认的分区布局。

·Use Free Space：使用未使用的分区空间。

·Create Custom Layout：自定义分区方式。

这里选择最后一种分区分式“Create Custom Layout”，即由管理员自行定制分区，因为它更灵活。然后单击界面右下角的“Next”继续。

2.选择“Create Custom Layout”自定义磁盘分区

选择“Create Custom Layout”后进入的界面如图2-16所示，可以看到磁盘总空间为8189MB，磁盘设备名为sda，剩余空间8189MB，说明还没有进行分区，磁盘的模式为VMware，表示是使用VMware虚拟机进行安装的。


提示：图2-16所示的界面（自定义的分区布局界面）下面有一排按钮，我们可以利用这些按钮来执行添加或删除分区等操作。其中Create表示新建分区，Edit表示编辑修改分区，Delete表示删除分区。具

体作用读者可自行尝试。

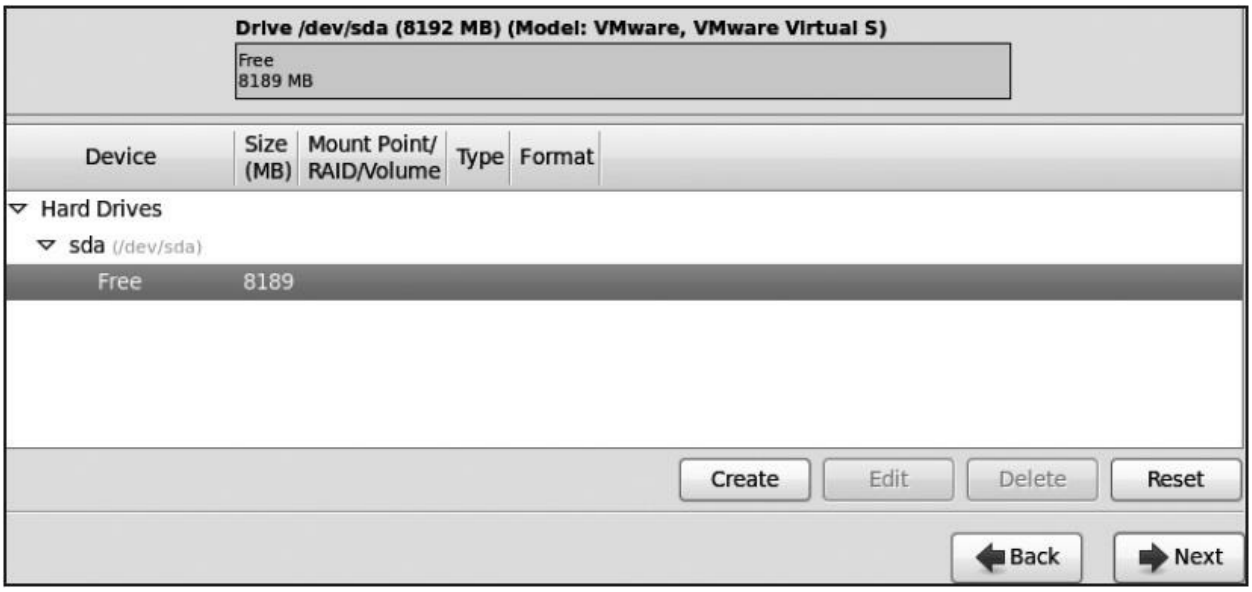


图2-16　自定义的分区图

3.按企业生产标准定制磁盘分区

根据图2-16可知，当前的计算机只有一块磁盘，设备名为sda（如果有多块盘，会显示sda、sdb、sdc等）。选中磁盘sda下的“free 8189”这一行，如图2-16所示，然后单击界面右下角的第一个按钮“Create”，开始创建磁盘分区，进入如图2-17所示的界面。

在图2-17中，默认的选项为“Standard Partition”，意思是创建标准分区，也是我们要选择的项。其他的例如“RAID Partition”的意思是RAID（磁盘冗余阵列）分区。在企业中RAID功能一般是通过物理硬件来完成的，硬件RAID卡的效率更高，操作系统的RAID功能性能差，冗余也受限于操作系统，因此企业很少用。而LVM的意思是逻辑卷管

理，它可以对设置好的分区大小进行动态调整，其前提是所有的分区格式都需要事先做成LVM格式，即分区标号为8e。企业环境的分区一般都是按需求事先规划好的，极少有后续调整的需求，且LVM的性能与标准分区及硬件RAID卡相比还是有一定的差距的，因此，如果没有特殊需求，不建议选择LVM和操作系统软RAID功能分区。

选择创建标准分区（Standard Partition）后，单击界面右下角的“Create”按钮，得到如图2-18所示的界面。

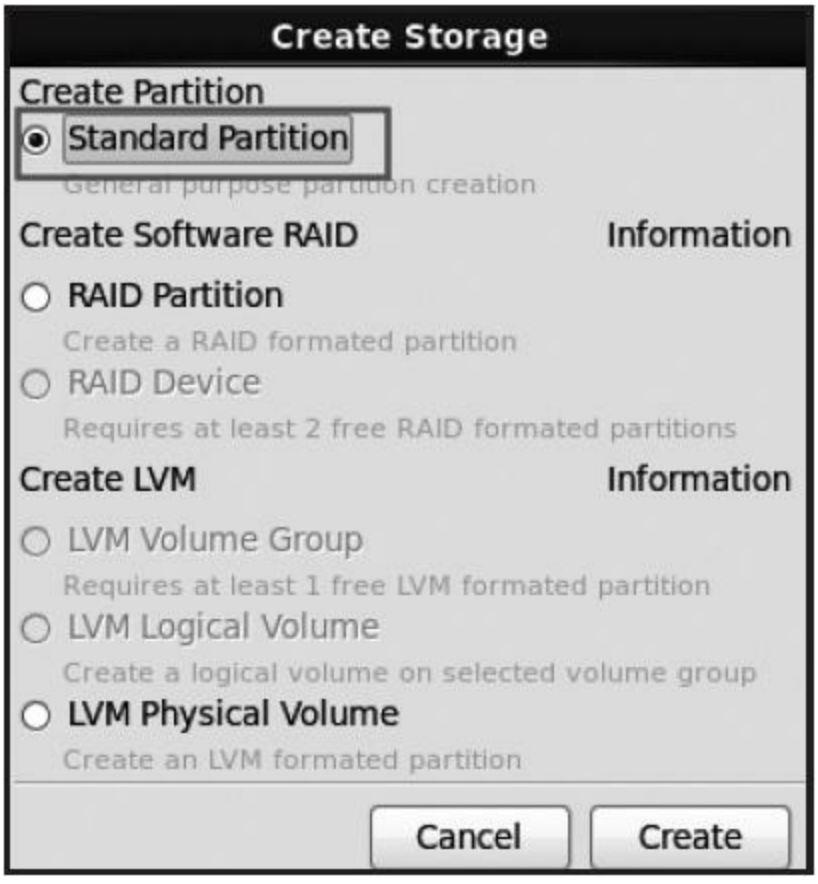


图2-17　创建分区界面

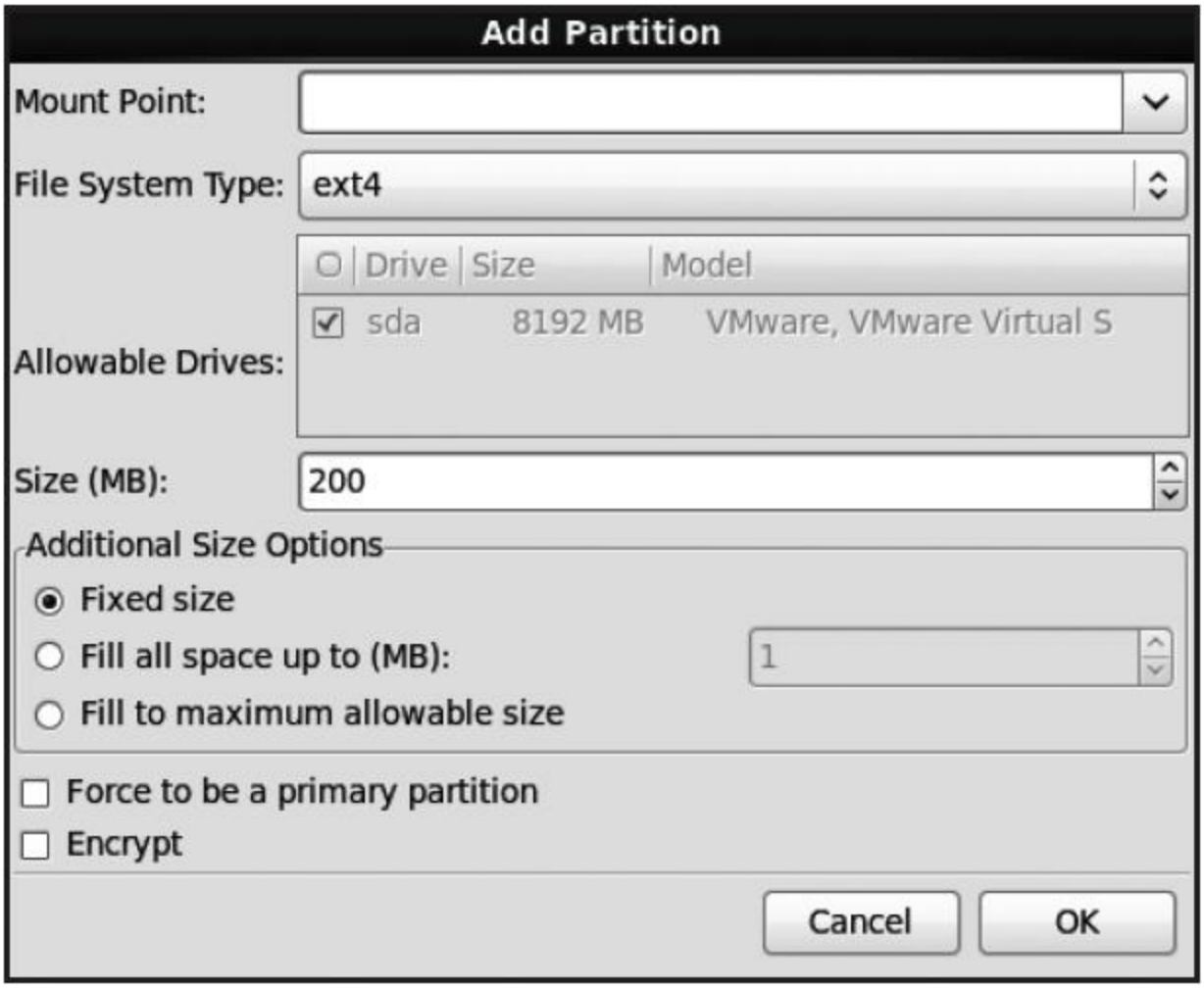


图2-18　添加分区界面图

Linux系统磁盘分区知识简介

在分区之前，需要先简单介绍一下磁盘分区相关知识，便于大家更好地理解学习。

1）磁盘在使用前一般要先分区。

2）磁盘分区有主分区、扩展分区和逻辑分区之分。一块磁盘最多

可以有4个主分区，其中一个主分区的位置可以用一个扩展分区替换，在这个扩展分区内可以划分多个逻辑分区。

3）如果规划的分区数量超过4个，则分区组合为3primary+1extend或2p+1e或1p+1e。

4）一块磁盘最多只能有一个扩展分区，扩展分区不能直接使用，必须在扩展分区上划分逻辑分区，然后格式化（创建文件系统），之后才能存取数据或安装系统。

磁盘分区命名及编号方式

（1）以设备名命名

在Linux系统中，磁盘设备对应于系统中的特殊文件，这些特殊文件放在“/dev”目录中，不同的设备对应的设备名称如下：

·系统的第一块IDE接口的硬盘称为/dev/hda。

·系统的第二块IDE接口的硬盘称为/dev/hdb。

·系统的第一块SCSI接口的硬盘称为/dev/sda。

·系统的第二块SCSI接口的硬盘称为/dev/sdb。

（2）使用数字编号

为了表示不同的分区，通常会用数字进行编号，比如：

·系统的第一块IDE接口硬盘的第1个分区称为/dev/hda1。

·系统的第一块IDE接口硬盘的第5个分区称为/dev/hda5。

·系统的第二块SCSI接口硬盘的第1个分区称为/dev/sdb1。

·系统的第二块SCSI接口硬盘的第5个分区称为/dev/sdb5。

需要注意的是，在对分区编号时，数字1~4只能留给主分区或扩展分区使用，逻辑分区（在扩展分区基础上建立）编号只能从5开始。

在对Linux系统设置了分区之后，还要在分区上创建文件系统才能安装系统，这个在安装时由系统自行完成创建。

Linux系统对分区的基本要求

1）最少要有一个根（/）分区，用来存放系统文件及程序。其大小至少在5GB以上。

2）要有一个swap（交换）分区，它的作用相当于Windows里的虚拟内存，swap分区的大小一般为物理内存容量的1.5倍（内存<8GB）。但当系统物理内存大于8GB时，则swap分区配置8~16GB即可，太大无用，浪费磁盘空间。swap分区不是必需的，但是大多数情况应该设置，个别企业的数据库应用场景不分swap。

3）/boot分区，这是Linux系统的引导分区，用于存放系统引导文件，如Linux内核等。所有文件的总大小一般只有几十MB，并且以后也不会增大太多。因此，该分区可以设置为100~200MB，这个分区也不是必需的。

企业生产场景中Linux系统的分区方案

常规的分区方案如下。

方案1：针对网站集群架构中的某个节点服务器分区，该服务器上的数据有多份（其他节点也有）且数据不太重要，建议的分区方案如下。

/boot：设置为100~200MB。

swap：物理内存的1.5倍，当内存大于或等于8GB时，配置为8~16GB即可。

/：剩余硬盘空间大小（/usr、/home、/var等分区和“/”共用一个分区，这相当于在Windows系统中只有一个C盘，所有数据和系统文件都放在一起）。

方案2：针对数据库及存储角色的服务器分区，该服务器的业务有大量重要的数据，建议分区方案如下。

/boot：设置为100~200MB。

/：大小设置为50~200GB，只存放系统相关文件，网站等的业务数据不放在这里。

swap：物理内存的1.5倍，当内存大于或等于8GB时，配置为8~16GB即可。

/data：剩余硬盘空间大小，放数据库及存储服务等重要数据。当然，data的名称也可以换成别的名字。

本方案其实就是把重要数据单独分区，便于备份和管理。

方案3：针对大网站或门户级别企业的服务器进行分区。

/boot：大小设置为100MB。

swap：物理内存的1.5倍，当内存大于或等于8GB时，配置为8~16GB即可。

/：大小设置为50~200GB，只存放系统相关文件，网站等的业务数据不存放在这里。

剩余的磁盘空间保留，不再进行分区，将来分配给不同的使用部门，由他们自己根据需求再分！

此种分区方案更灵活，比较适合业务线比较多、需求不确定的大企业使用。

对于分区，有网友还给出了如下方案：

/boot、swap、/、/usr、/home、/var

这种分区方案是典型的没有主见的分区方式，有太多的额外分区（/usr、/home、/var），有没有必要且不说，管理起来也很麻烦，这与一个家庭就2~3口人，买了100平方的房子，非要隔成几个房间同一道理，老男孩极不推荐这样的分区方案。

如果说怕某个分区满了会影响系统运行，这样的分区想法是错的。第一，硬盘空间是固定的，分区多了，比只分一个区肯定更容易满；第二，在企业应用里，业务不可用和服务器宕机的危害几乎差不多，因此，分区少一些，然后对所有分区进行监控报警是目前多数规范企业的选择。

本文采用常规的服务器分区方案，即分为/boot、swap、/三个分区，注意分区的先后顺序。

关于磁盘分区就介绍这么多。下面继续介绍安装系统的操作。

1）创建第一个分区，/boot分区

在“Add Partition”（添加分区）的界面中（如图2-18所示），需要分别手工输入分区挂载点目录、选定文件系统类型、对哪个磁盘操作（如果服务器有多个磁盘则要特别注意），以及指定创建的分区大小。操作完成后的界面如图2-19所示。之后单击“OK”继续。

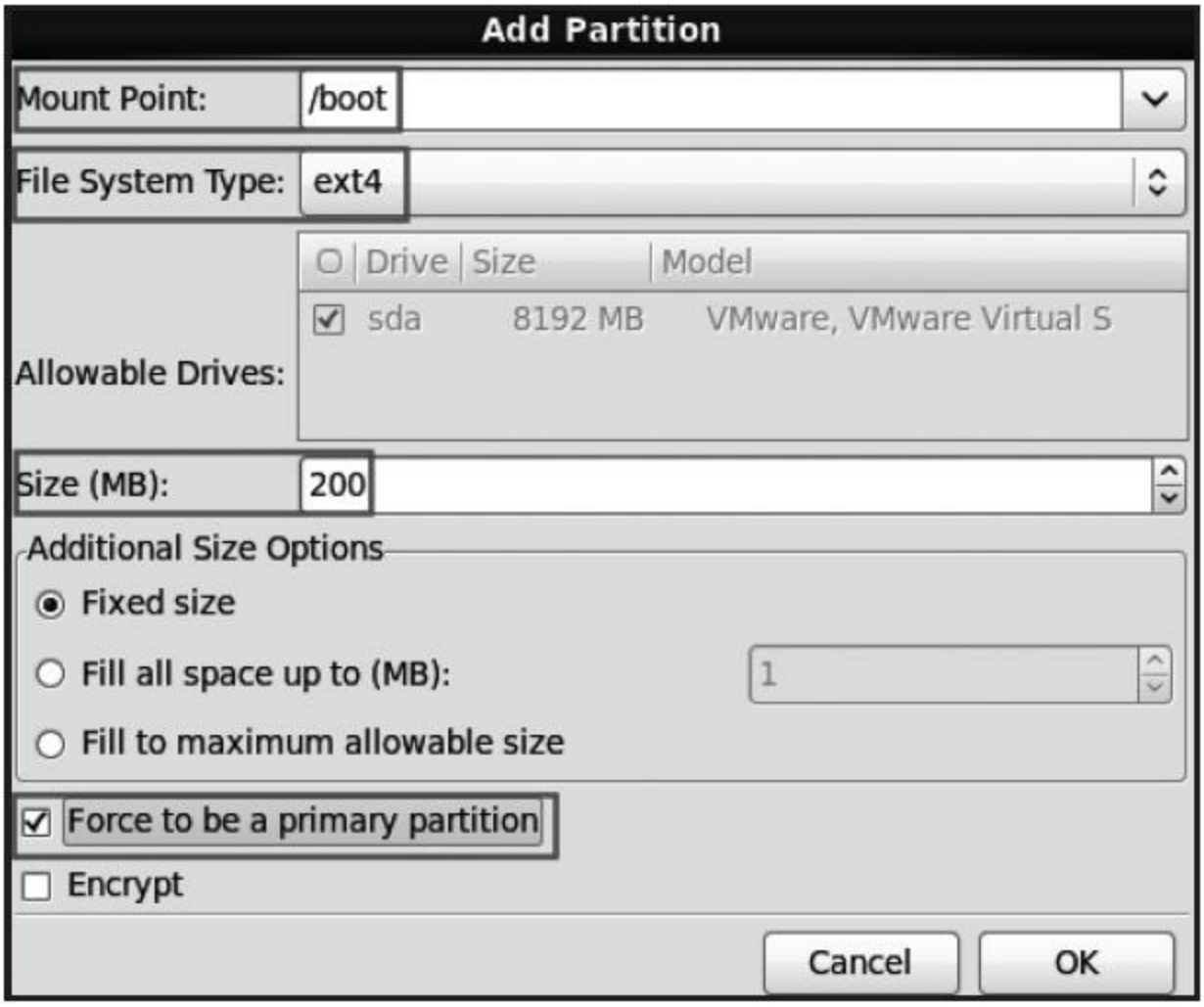


图2-19　/boot分区配置过程图

界面选项说明如下。

1）Mount Point的意思是挂载点，这是Linux下访问磁盘分区的入

口，即如果要往/boot分区（/dev/sda1）写入数据，就必须通过/boot入口来写入，这一点与Windows是不同的。

2）File System Type的意思是文件系统类型，就像Windows的fat32/ntfs一样，磁盘分区只有在设置了文件系统类型格式化并挂载上挂载点后，分区才能存放数据。目前有如下一些文件系统类型。

·ext2/ext3/ext4：适合Linux的文件系统类型。由于ext3文件系统多了日志记录功能，因此系统恢复起来会更快速，ext4是ext3的升级，效率更加高，因此建议使用默认的ext4类型，而不要使用ext2/ext3。

·physical volume（LVM）：一种弹性调整文件系统大小的机制，可以让文件系统变大或变小，而不改变原有文件数据的内容，功能不错，但性能会下降。

·software RAID：利用Linux系统的特性，用软件仿真出磁盘阵列的功能。

·swap：内存交换空间。由于swap并不会使用到目录树的挂载，因此用swap就不需要指定挂载点。

·vfat：同时被Linux与Windows所支持的文件系统类型。如果主机硬盘上同时存在Windows与Linux两种操作系统，有数据交换需求，可以使用该文件系统。

·xfs：一个文件系统类型，在CentOS 7中将被作为默认的文件系统类型而替换ext4。

3）Force to be a primary partition的意思是强制主分区，是可选项。由于所有的分区未过4个，因此可勾选该项。

创建/boot分区后的结果如图2-20所示。

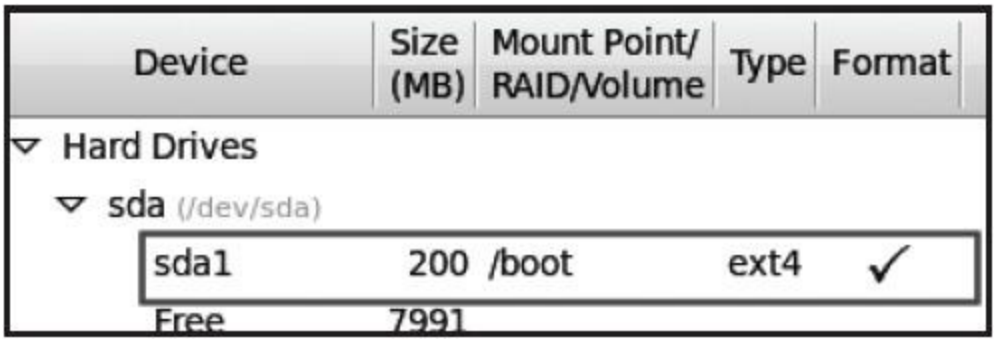


图2-20　创建/boot分区后的结果图

2）创建swap交换分区

在图2-20中，选中磁盘sda下面的“free 7991”这一行，然后单击右下角第一个按钮“Create”，再次进入“Add Partition”操作界面，创建swap交换分区，同创建/boot分区一样，swap分区的设置也比较简单，整个配置内容和过程如图2-21和图2-22所示。

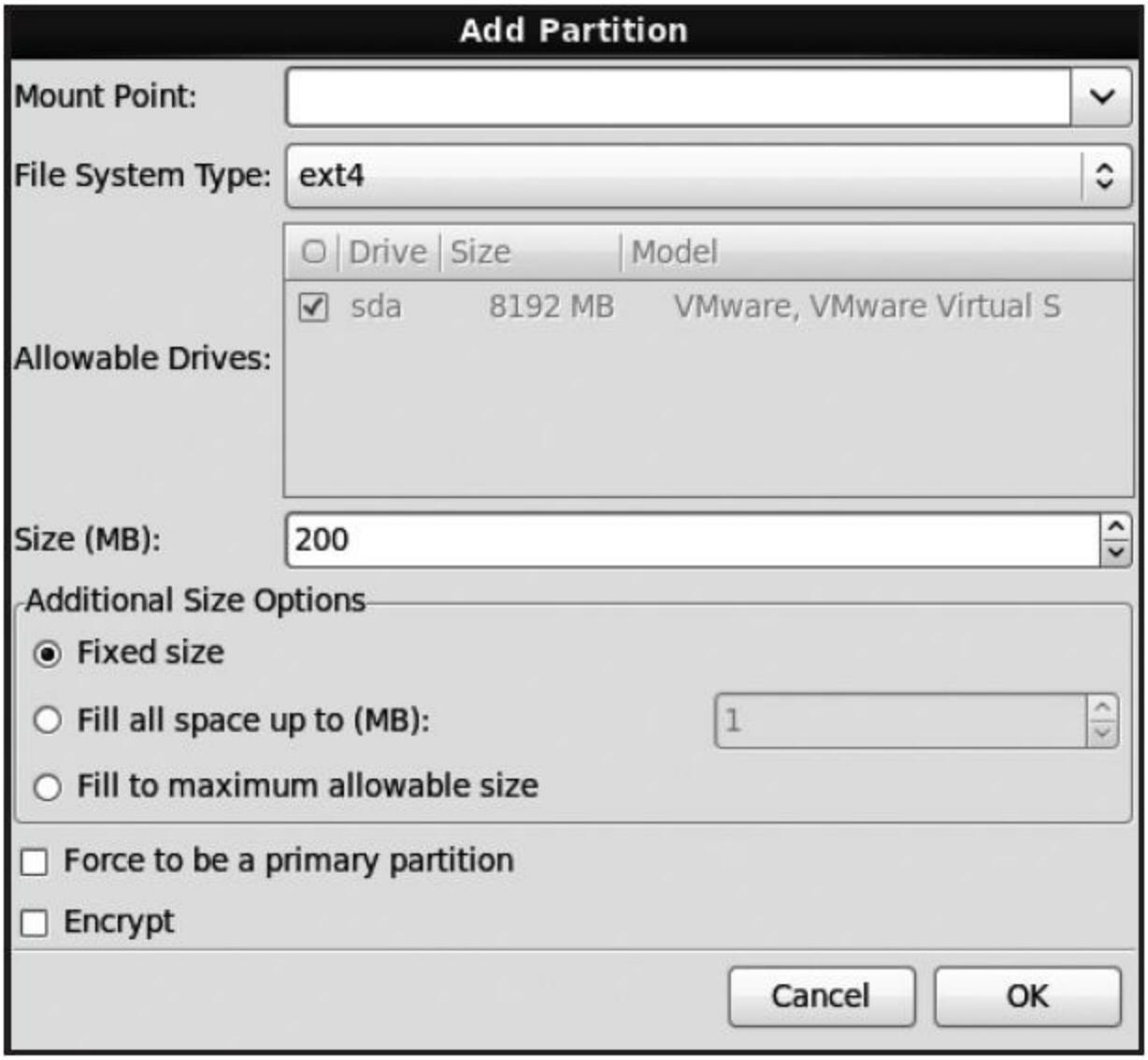

Add Partition
Mount Point:
File System Type:
ext4
Drive
Size
Model
sda
8192 MB
VMware, VMware Virtual S
Allowable Drives:
Size (MB):
200
Additional Size Options
Fixed size
Fill all space up to (MB):
1
Fill to maximum allowable size
Force to be a primary partition
Encrypt
Cancel
OK

图2-21　添加分区界面

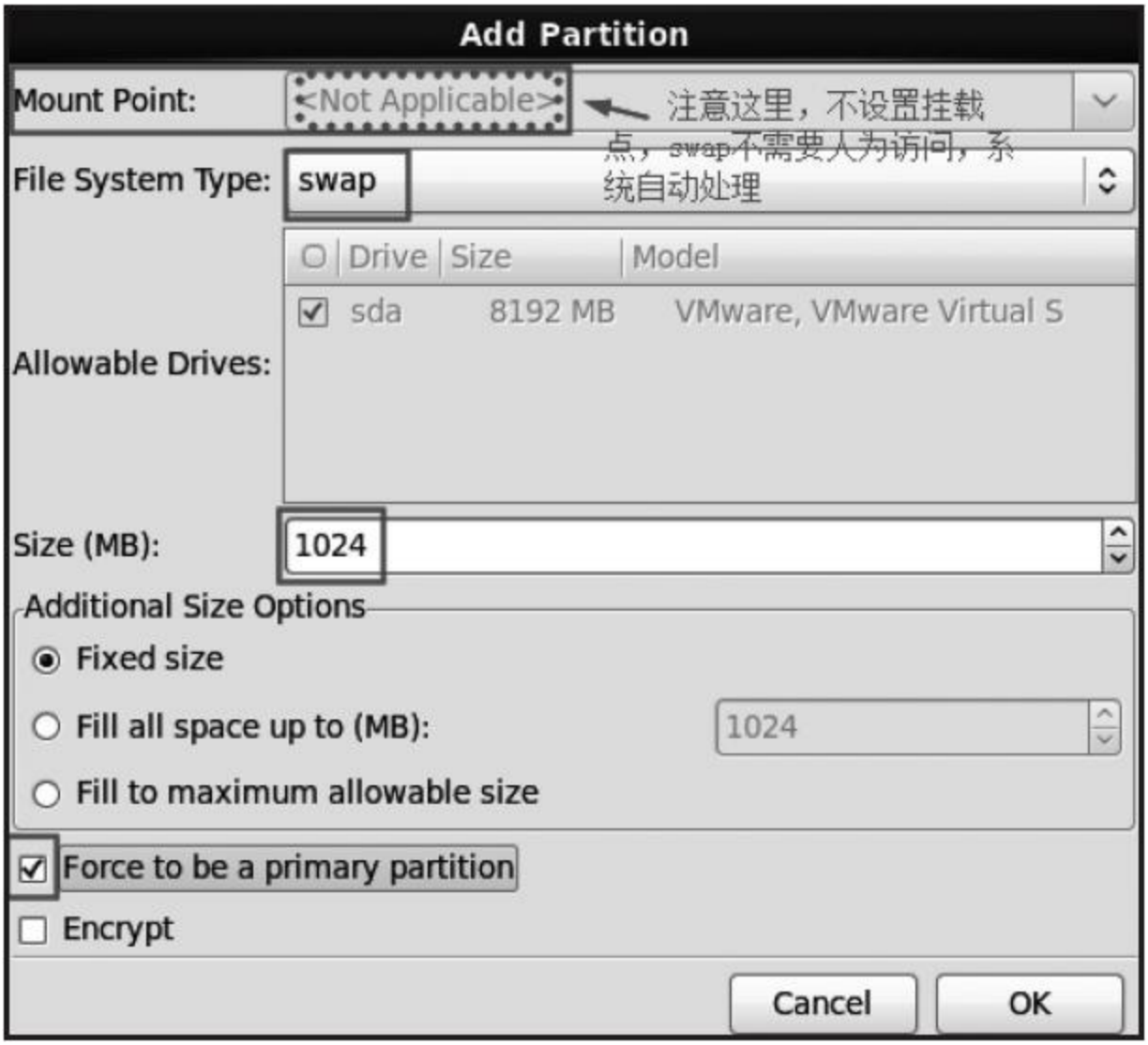


图2-22　swap分区配置过程图

swap分区配置说明：

1）swap分区不配置Mount Point挂载点，swap不需要人为访问，交给系统自动处理。

2）swap分区的大小一般为物理内存容量的1.5倍（内存<8GB）。但当系统物理内存大于8GB时，swap分区配置8~16GB即可，太大无

用，还浪费磁盘空间。

3）如果是学习环境，为了节省空间，swap设置得小一点也没有问题。

创建swap分区后的结果如图2-23所示。

有关交换分区swap的说明：

Linux系统分区时，常规的分区方案为/、/boot、swap，其中swap不是挂载点（/、/boot都是挂载点），而是分区类型，类似ext4的一个文件系统，如果在挂载点输入框处输入“/swap”创建swap分区就错了。

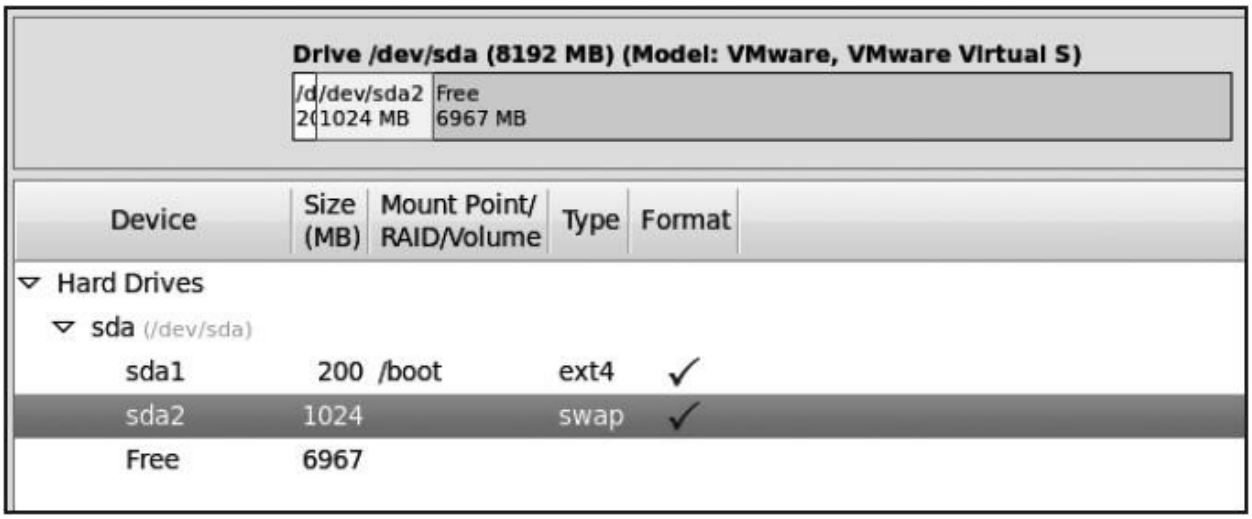


图2-23　创建/boot和swap分区后的结果图

3）创建/（根）分区

在图2-23中，选中磁盘sda下面的“free 6967”这一行，然后单击右下

角第一个按钮“Create”，再次进入“Add Partition”操作界面，创建/（根）分区，同创建/boot分区一样，可轻松完成/（根）分区的设置，整个操作过程如图2-24和图2-25所示。

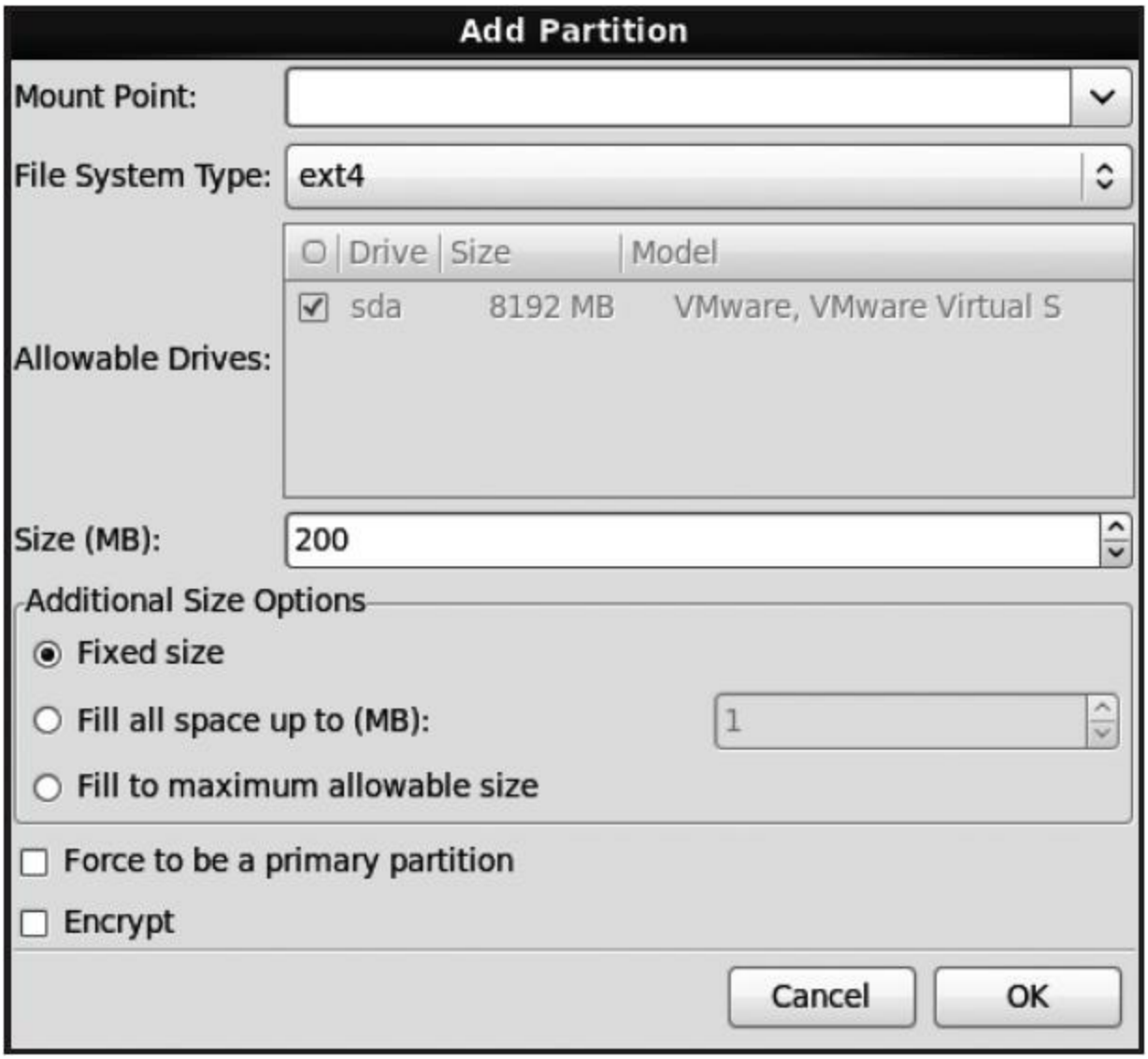


图2-24　添加分区图

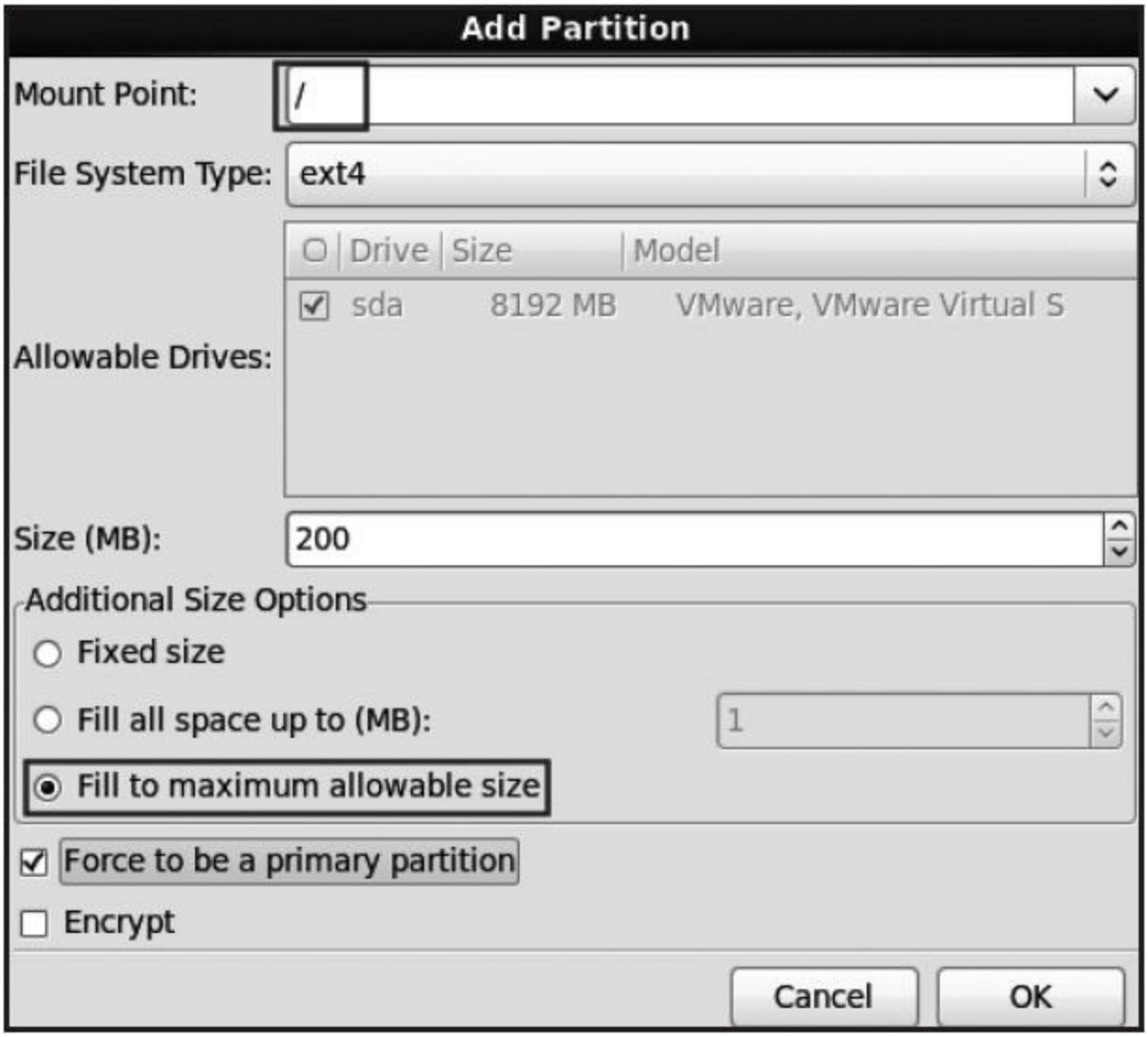


图2-25　/（根）分区配置图

根（/）分区配置

由于根分区是最后一个分区，因此把剩余空间都给了它，对应的选项为“Fill to maximum allowable size”。这时，原来的“Size（MB）”右边的数字框内设置的数字就不起作用了。

创建/（根）分区后的结果如图2-26所示。

创建完所有的分区后，单击界面右下角的“Next”继续。此时系统需要对配置的分区格式化，而且会出现格式化警告窗口，单击“Format”（格式化）继续，这个过程中还可能提示写入分区表，在写入配置到磁盘的窗口，单击“write changes to disk”继续。整个过程如图2-27所示。

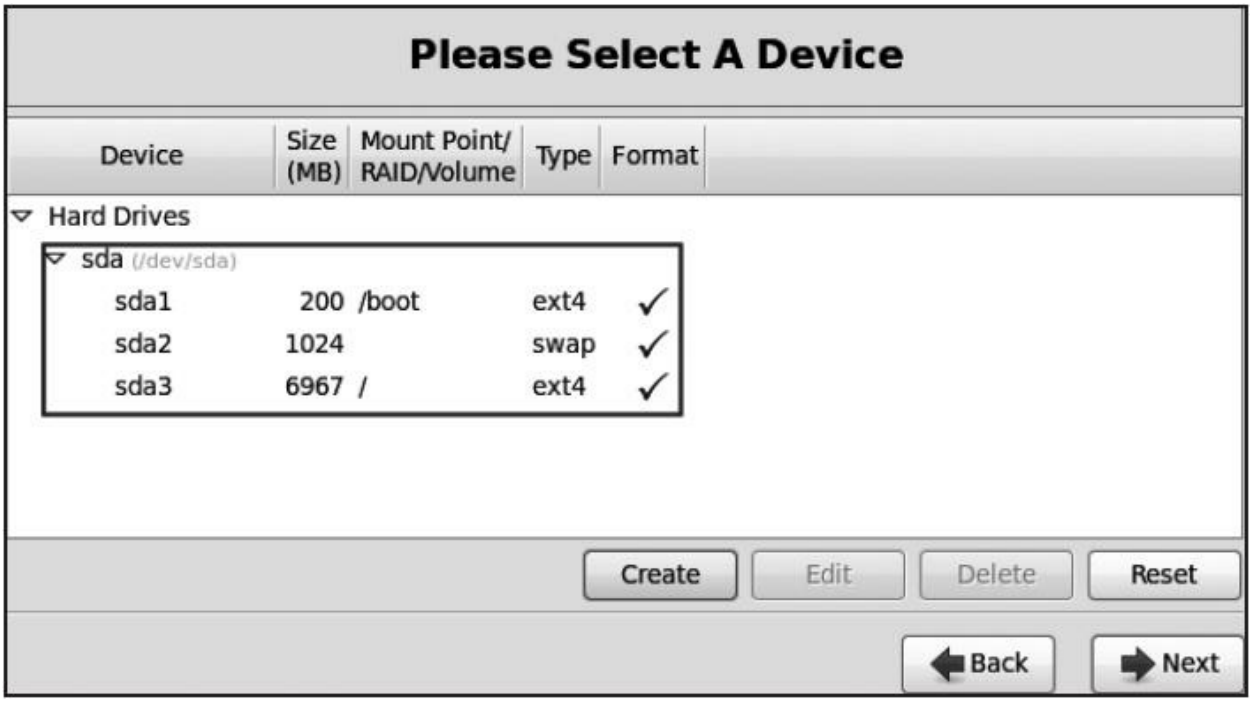


图2-26　创建/boot、swap和/（根）分区后的总图

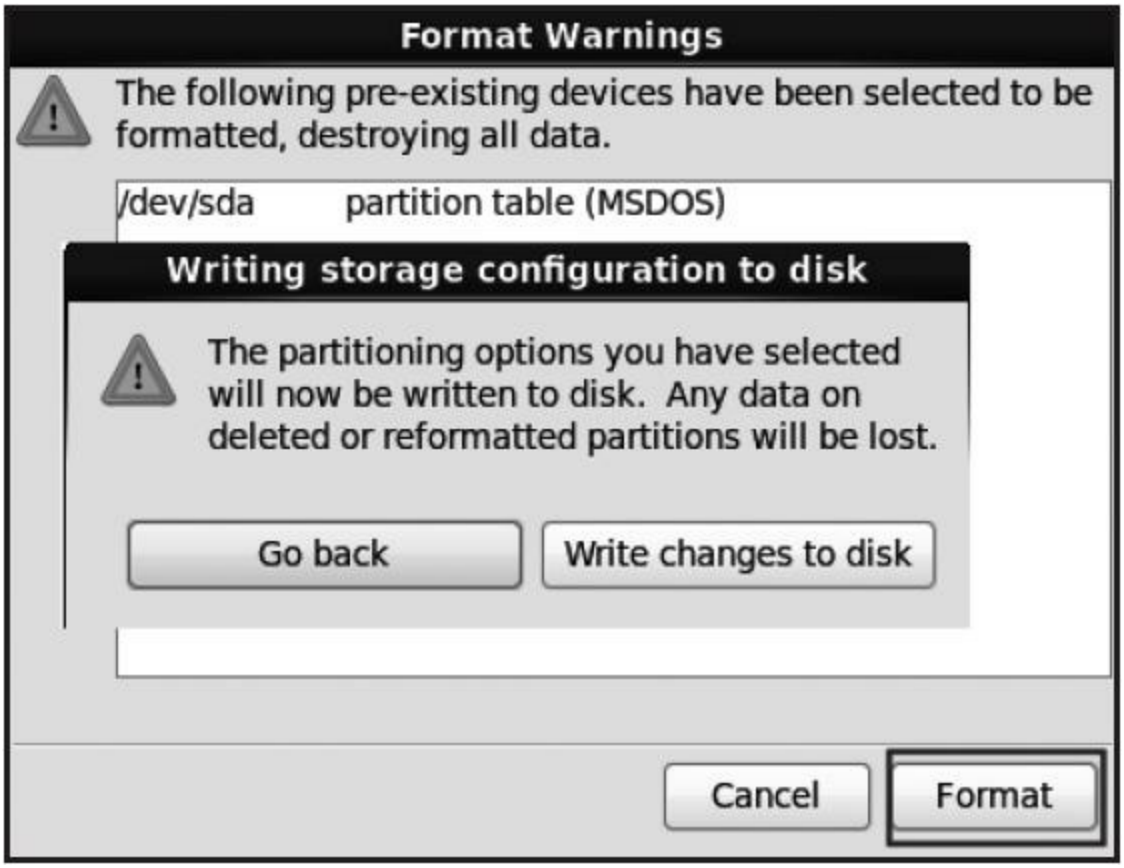


图2-27　格式化警告和写入磁盘的图

提示：这里采用的是生产环境中集群节点下的节点服务器的分区方式，即系统坏掉后硬盘数据不需要保留。此分区方式也适合大多数生产环境的服务器，如果是数据库及存储等有重要数据的特殊业务服务，一般会单独划分存放数据的分区，如/data。

除了/boot、swap和/三个分区外，还可以加/usr、/home、/var等分区，具体要根据服务器的需求来决定，一般情况下，只配置这三个分区就足够了。这种分区方案最大优点就是简单，使用方便，可批量安装部署，而且不会存在有的分区满了，有的分区还剩余很多空间又不能被利

用的情况（LVM的情况在这里先不阐述）。

该分区方案的缺点是如果系统坏了，重新安装系统时，数据都在/（根）分区，导致数据备份很麻烦，如果设置了/usr、/home、/var等分区，即使系统出了故障，也可以直接在/（根）分区安装系统，这样并不会破坏其他分区的数据。当然，刚才也说了，如果是不存在备份数据的集群的节点，那采用这种分区方案是很明智的，不需要特别担心某个分区暴满的问题。

2.3.3　CentOS 6.6系统安装包组的选择与配置过程

1.启动引导设备的配置

分区格式化后，会进入如图2-28所示的界面，系统默认使用GRUB作为启动加载器，引导程序默认在MBR下，然后按“Next”键继续。

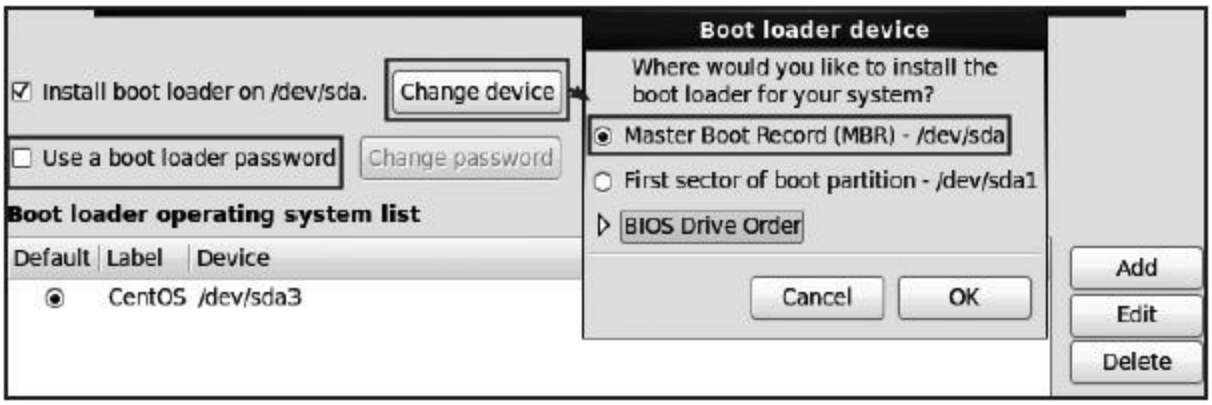


图2-28　Boot Loader配置

提示：如果想防止别人接触到计算机破解root密码，以及执行一些特殊操作，可以勾选图中“Use a boot loader password”，为GRUB引导菜单设置密码，此功能在安装系统后也可以设置，后文有讲解。

2.系统安装类型选择及自定义额外包组

现在进入如图2-29所示的界面。在该图中，上半部分是系统定制的各种安装类型选项，默认是“Desktop”，这里选择“Minimal”，即最小化

安装，下半部分是在确定系统安装类型后，想额外添加的软件包组选择项，这里选择“Customize now”，即立即自定义。

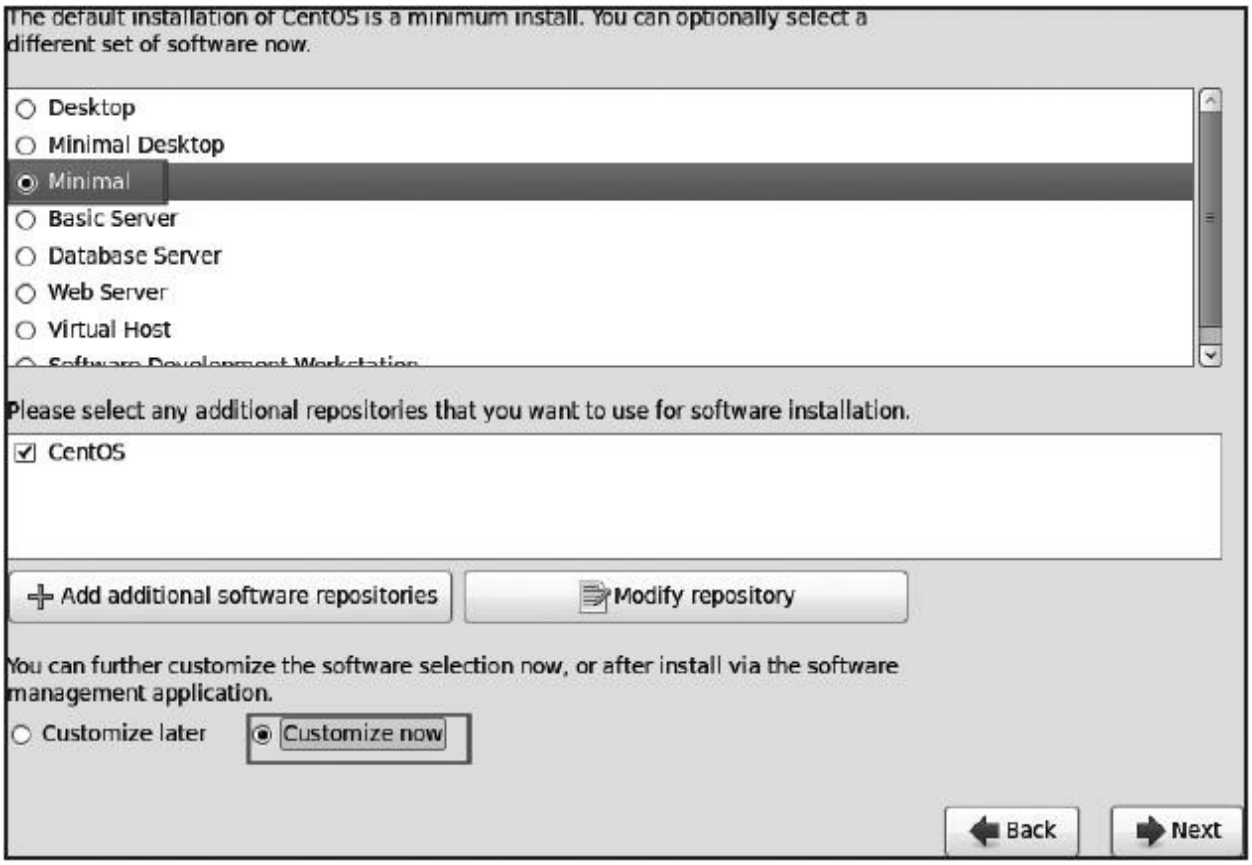


图2-29　系统安装包类型选择及自定义额外包组

选择自定义额外的包组界面后，首先需要选择左边大的分类，然后再勾选右边对应的包组（注意，使用默认的包组内容即可）。选择结果如图2-30和图2-31所示。

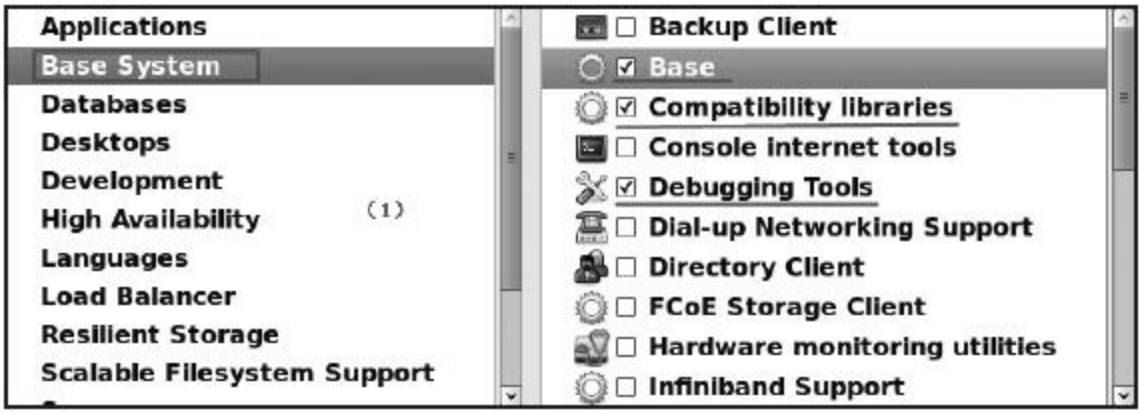


图2-30　自定义安装包组选择界面1

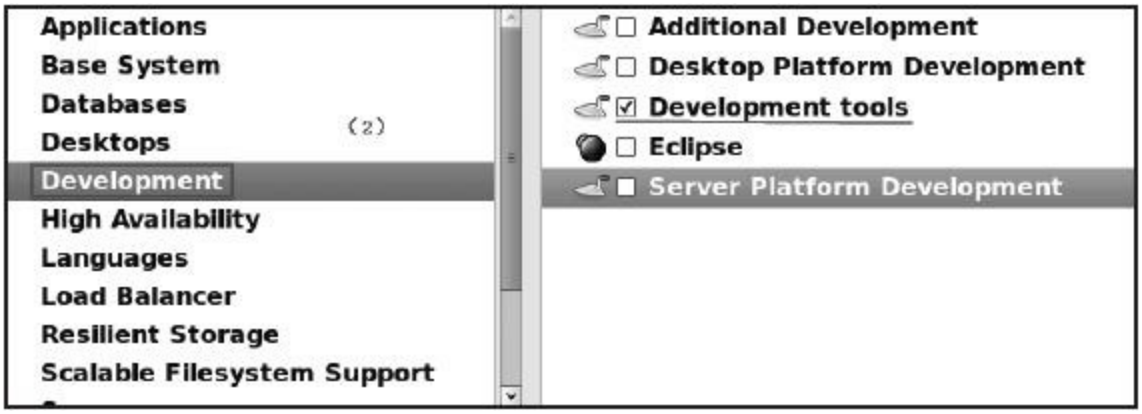


图2-31　自定义安装包组选择界面2

提示：根据经验，选择安装包时应该采用最小化原则，即不需要的或不确定是否需要的就不安装，这样可以最大程度上确保系统安全。

如果安装过程中落下了部分包组，或者是其他伙伴安装时没选，在安装完系统后可以以如下方式补上未安装的包组：

例如：补充安装"Compatibility libraries"、"Base"、"Development tools"3个包组的命令为：

```
yum groupinstall "Compatibility libraries" "Base" "Development tools"
```

补充安装"debugging Tools"、"Dial-up Networking Support"2个包组的命令为：

```
yum groupinstall "debugging Tools" "Dial-up Networking Support"
```

可以通过“yum groupinfo包组名”查看具体安装的包组组件。

下面是安装了CentOS 6.6后，登录系统时查看到的选包的情况：

```
yum grouplist:


Installed Groups:


   Base
   Compatibility libraries
   Debugging Tools
   Development tools
   E-mail server
   Graphical Administration Tools
   Hardware monitoring utilities
   Legacy UNIX compatibility
   Networking Tools
   Performance Tools
   Perl Support
   Scientific support
   Security Tools
```

可以看到，除了我们选择的以外，系统还默认安装了一些软件包组。

3.万事俱备，开始安装系统

安装系统的界面如图2-32和图2-33所示。

图2-32　安装一开始过程图

图2-33　安装一半以后的过程图

提示：在安装过程中，会显示总共需要安装包的数量及当前已安装完的包的数量，还会显示当前安装包的包名和大小，以及总的安装

进度等信息。

4.安装结束

系统安装结束后，出现的界面如图2-34所示。此时单击“Reboot”键即可重启系统。

图2-34　安装过程结束

2.4　系统安装后的基本配置

2.4.1　重启系统过程中的引导过程介绍

系统安装结束后，取出安装时使用的DVD系统盘（如果是虚拟机镜像文件，此时无法取出），按“Reboot”键重新启动系统。首先进入如图2-35所示的CentOS开机引导界面。

图2-35　CentOS开机引导界面

此时如果按下键盘的Esc键，可以像CentOS 5一样查看系统引导过程的细节，如图2-36所示。

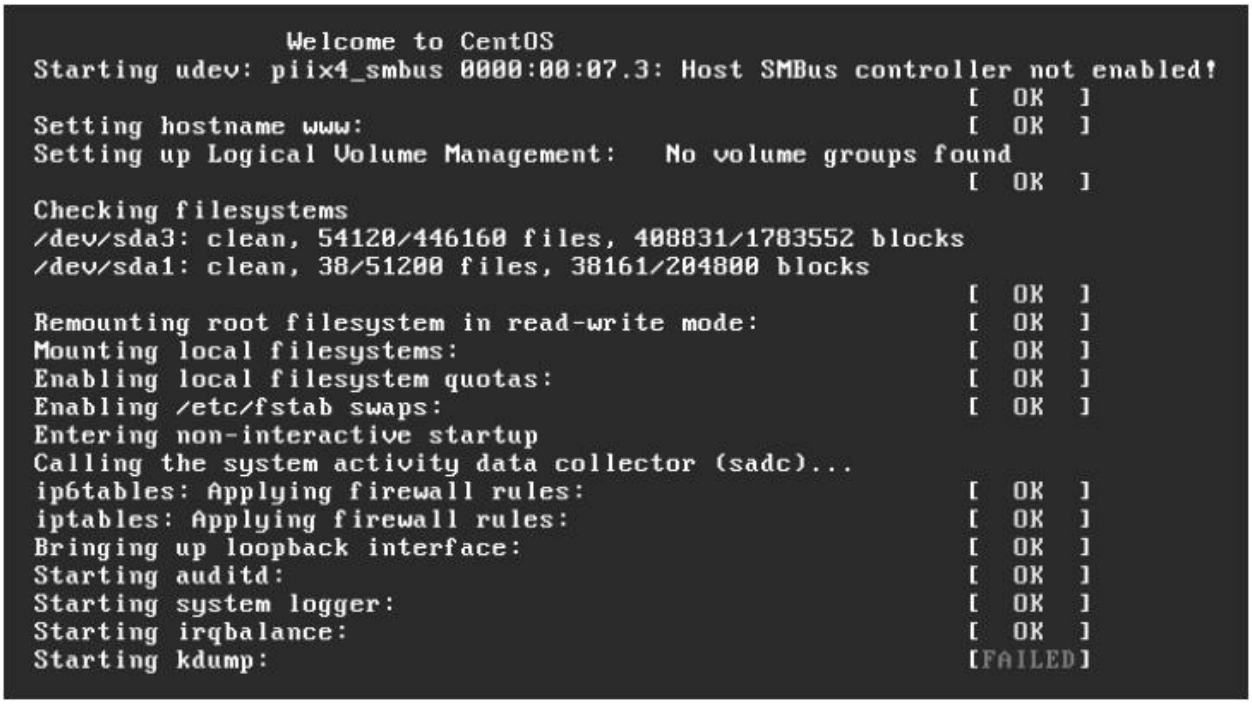


图2-36　系统引导过程细节界面

系统引导及程序服务加载完毕后，会出现登录界面，如图2-37所示。

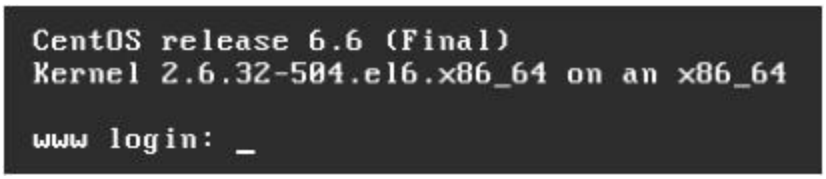


图2-37　系统登录界面

提示：登录界面中的英文内容为CentOS的版本及内核的当前版本。

Linux内核版本号为：

[root@www~]#uname-r

2.6.32-504.el6.x86_64

这里补充介绍一下各数字和字母代表的含义：第一个数字2表示主版本号，有结构性变化才更改；接下来的数字6表示次版本号，新增功能时才变化，一般奇数表示测试版，偶数表示开发版；接下来的32表示对次版本的修订次数或补丁包数；504代表编译的次数，每次编译可对少数程序优化或修改；e16用来表示版本的特殊信息，有较大的随意性；el代表企业版Linux；pp代表测试版；fc代表fedora core；rc代表候选版本；x86_64表示64位。

在图2-37中显示的www为主机名，login为登录提示。

2.4.2　登录系统

现在，把光标定位到“login：”提示后面，然后输入root超级用户名，按Enter键，并按提示输入密码（注意密码是不显示的），过程如图2-38所示。

登录到系统后的界面如图2-39所示。

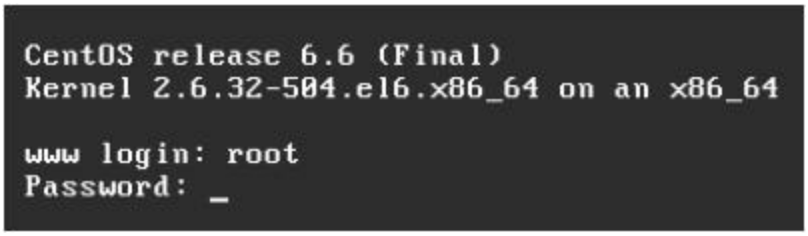


图2-38　登录过程图

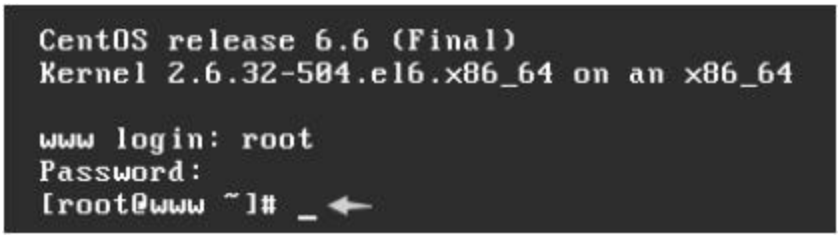


图2-39　登录系统后的界面图

提示：“[root@www~]#”里的#号为超级管理员root输入命令的提示符，在#号后面可以输入命令进行系统管理。

在企业的工作环境中，都是在这种提示符下输入命令进行管理的，而不是像Windows系统那样采用图形界面进行管理，这一点对于从

Windows转过来的读者尤为重要。命令管理的初期会有些麻烦，让人不适应，等熟练以后就会发现命令管理更直观、更简洁，效率也更高，这也是Linux的强大之处。

2.4.3　配置网卡和设置网络联网

在2.3.1节的第8步“初始化主机名及配置网络”中，已经讲解过如何配置网卡了，只不过当时没有具体选择，现在可以配置了，如果读者已经在前面配置过了，这里可以忽略。

1.通过setup命令设置网卡

1）在系统命令行下输入setup命令，然后按回车键，如图2-40所示。

2）通过按tab键及对应的键依次选择进入如下几个窗口，为了节省篇幅，老男孩把几个操作进行了合并，并进行了排序，如图2-41所示。

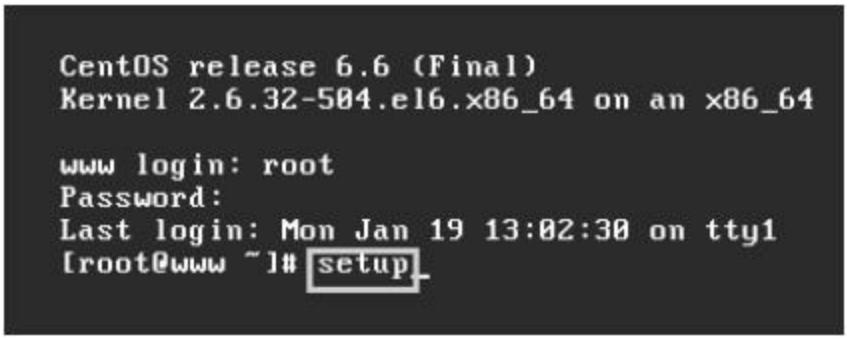


图2-40　配置网卡图

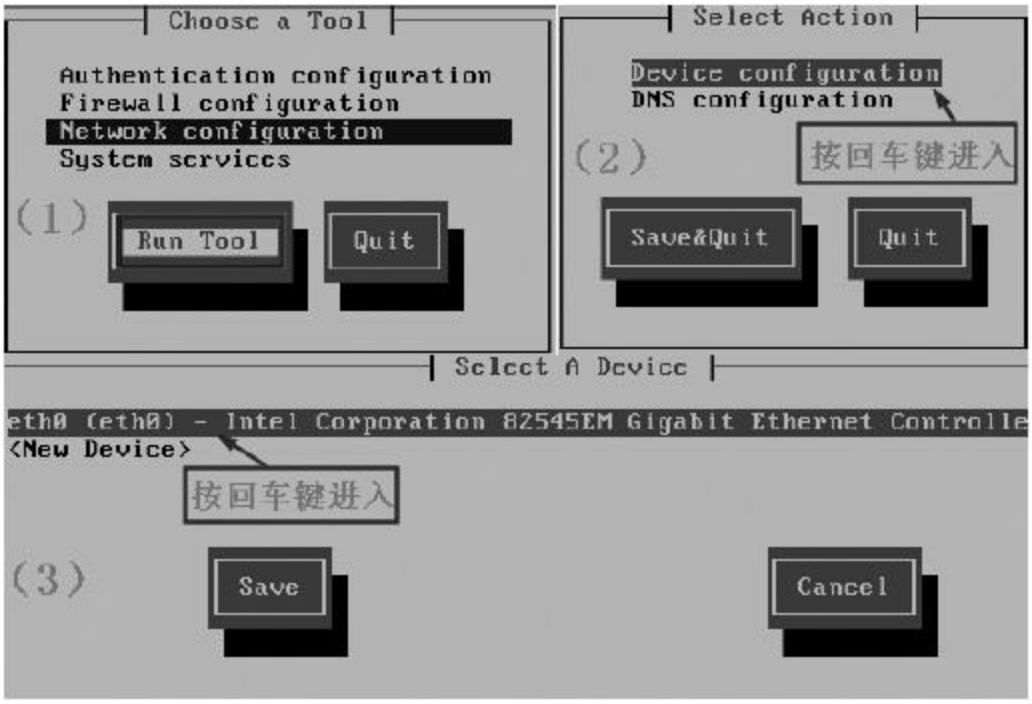


图2-41　配置网卡过程图

提示：整个配置过程依次为：执行setup命令→Network configuration→Device configuration→eth0，配置完依次单击OK→Save→Save&Quit→Quit，这样就退出到命令行界面了。

该配置实质上就是修改网卡的配置文件/etc/sysconfig/network-scripts/ifcfg-eth0内容。

2.VMware虚拟机系统下的设置方法

1）虚拟机NAT模式下的网卡设置

如果虚拟机的宿主机是拨号上网的，则虚拟机的网卡最好使用NAT模式，VMware软件设置虚拟网卡为NAT模式的配置如图2-42所示。

对应的网卡配置如图2-43所示。

提示：

·NAT模式下要对Use DHCP的位置进行设置。

·无需配置IP、子网掩码和网关，因为DHCP会自动分配。

·需要配置本地主DNS和辅DNS解析地址。

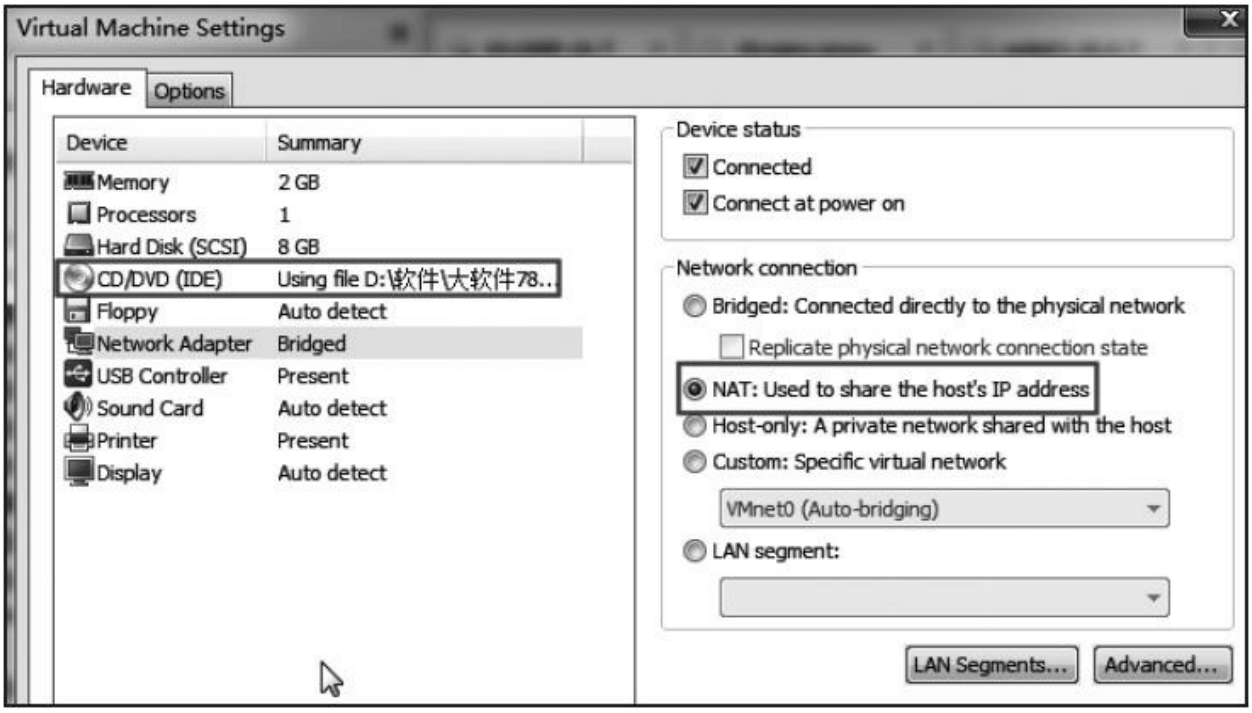


图2-42　VMware软件设置虚拟网卡为NAT模式

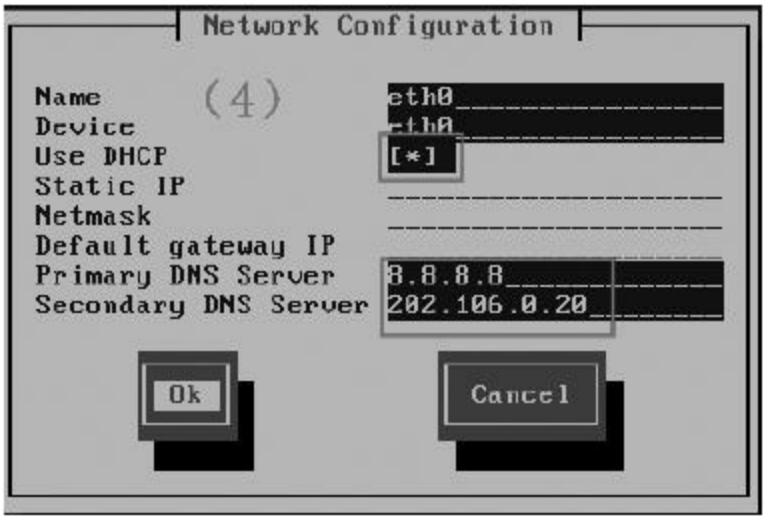


图2-43　图形网卡配置说明

配置完成后，依次单击OK→Save→Save&Quit→Quit，即可退出到命令行界面。

然后依次执行如下命令：

（1）执行命令ifup eth0，启动网卡。

（2）执行命令ifconfig eth0，查看获取的IP。

（3）执行命令ping baidu.com，检测网络是否通畅。

详细的操作过程如图2-44所示。

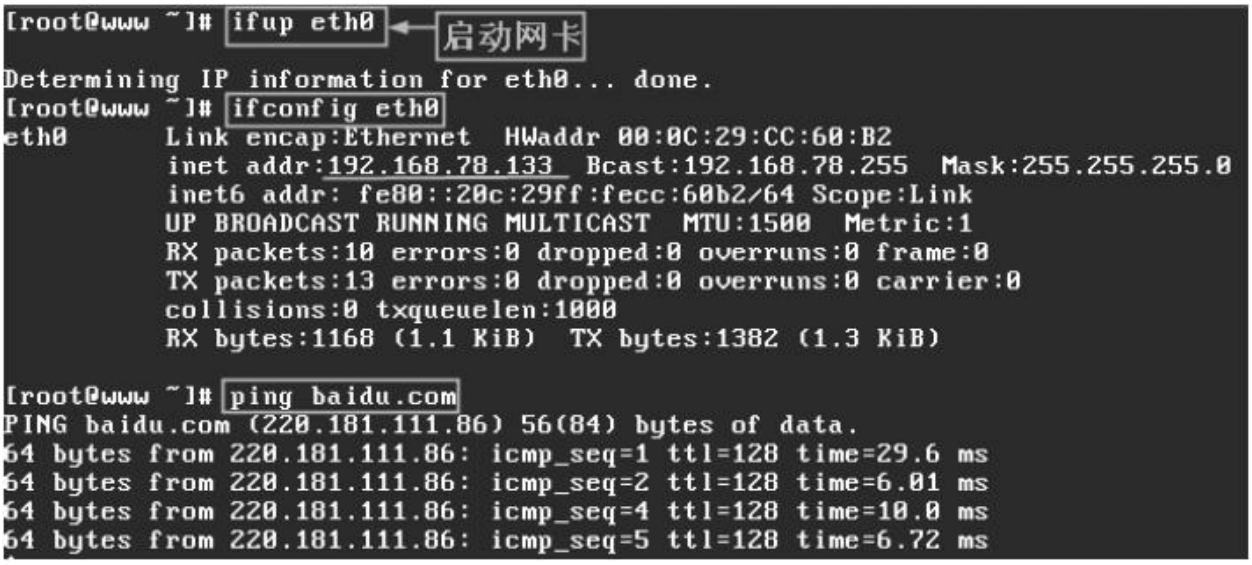


图2-44　启动网卡联网测试过程图

此时就可以用SSH工具（例如SecureCRT、xshell等SSH工具）连接系统了。

2）虚拟机桥接（bridged）模式下的网卡设置

如果虚拟机的宿主机在局域网内，是通过路由器上网的，则虚拟机的网卡除了可以使用NAT模式外，还可以使用桥接模式。VMware软件设置虚拟网卡为桥接模式的配置如图2-45所示。

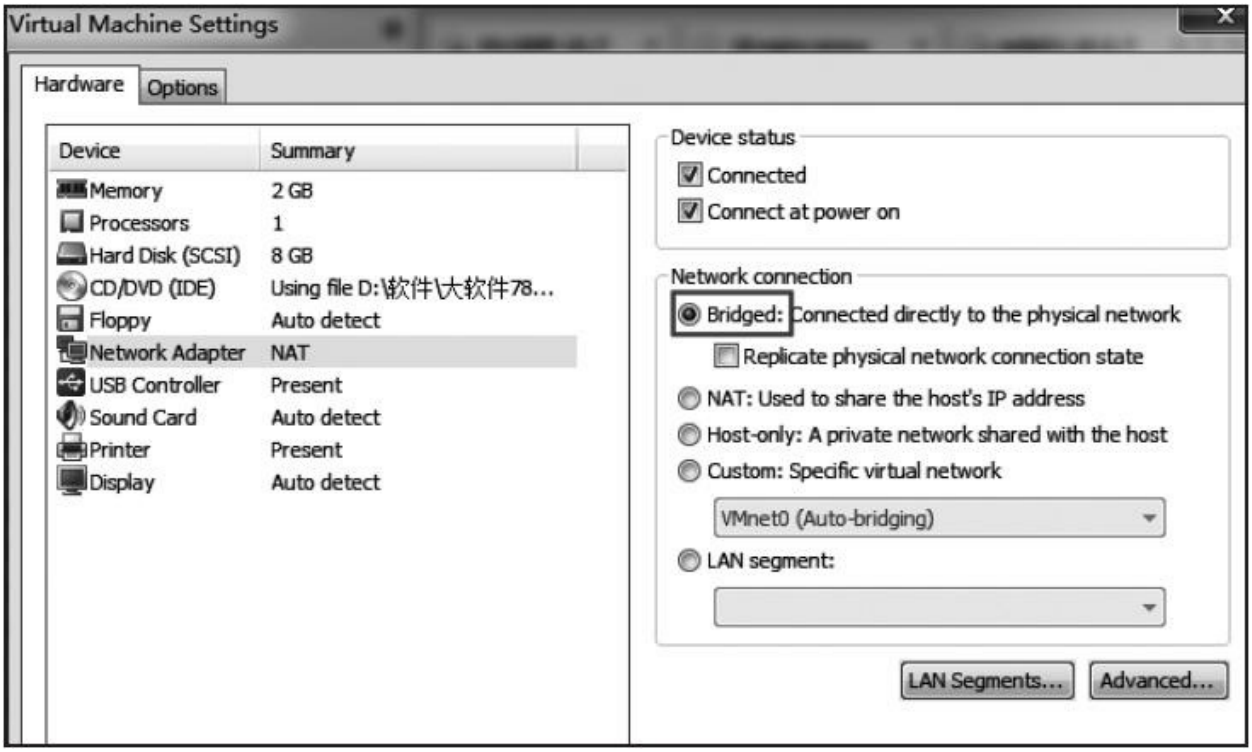


图2-45　VMware软件设置虚拟网卡为桥接模式

对应的网卡配置如图2-46所示。

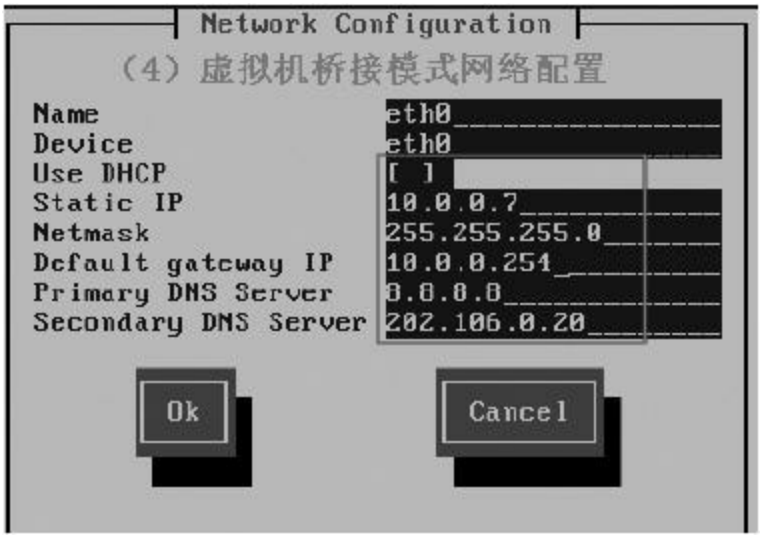


图2-46　网络配置图

提示：

·在桥接模式下Use DHCP的位置可以取消，当然也可以利用DHCP自动分配，这样更简单。

·如果考虑自己设置固定IP，则需配置IP、子网掩码、网关，此时禁止DHCP分配。

·需要配置本地主DNS和辅DNS解析地址。

配置完成后，与NAT配置一样，依次单击OK→Save→Save&Quit→Quit，这样就退出到命令行界面了。

然后依次执行如下命令：

（1）执行命令ifup eth0，启动网卡。

（2）执行命令ifconfig eth0，查看设置或获取的IP。

（3）执行命令ping baidu.com，检测网络是否通畅。

详细操作过程如图2-47所示。

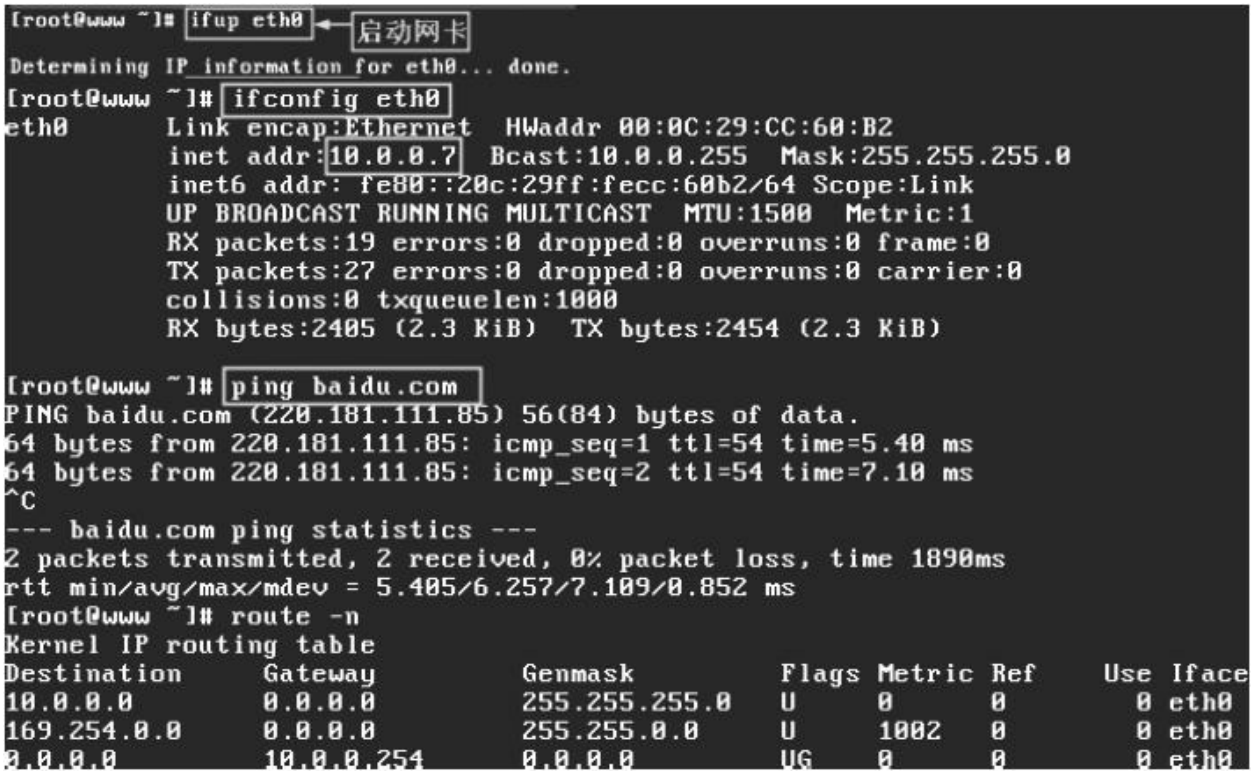


图2-47　启动网卡联网测试过程图

3.物理服务器系统的网络设置方法

物理服务器系统的网络设置同虚拟机桥接网卡模式完全一样，因此，这里就不赘述了，读者可以参考虚拟机桥接网卡模式的配置过程。

4.系统无法联网的故障排除方法

1）首先要确认系统的安装方式是虚拟机还是物理服务器。

如果是虚拟机，同时还要考虑虚拟机所在的宿主机联网方式，如果宿主机是采用直接拨号方式上网，虚拟网卡的模式就选NAT模式，如果宿主机采用路由器局域网方式上网，那么虚拟机的网卡选择bridged和

NAT两种模式都可以。如果是物理服务器的话，直接配置就可以了。参考虚拟机桥接的配置。

2）如果系统安装方式检查无误，仍然不能上网，继续确认网卡实际配置是否正确。

如果虚拟机是NAT模式，需要使用DHCP自动获取IP；如果是虚拟机bridged桥接模式，则需要根据虚拟机所在的宿主机联网方式（可以查看对比）设定IP、子网掩码及网关，更方便的方法还是使用DHCP获取IP。不管是采用bridged模式还是采用NAT模式，最好都手工指定DNS。

提示：采用NAT模式上网时，要注意宿主机Windows的计算机管理、服务和应用程序对应的VMware的DHCP和NAT服务是否开启（如图2-48所示）。

图2-48　宿主机Windows的VMware DHCP和NAT服务

如果是物理服务器，检查IP、子网掩码、网关和DNS等设置是否正

确，可与局域网内其他可上网的机器进行一一对比，或者找网管询问。

3）上面两类问题都排除后，就需要确认网卡的配置文件了。

使用setup命令配置网卡的过程实际上就是修改了如下的网卡配置文件，因此，可以通过查看网卡的配置文件，确认其与想要配置的是否一致。

下面展示了如何查看网卡配置文件，以及网卡配置项的含义。

```
[root@www ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0          ←第一块网卡逻辑设备名，第二块为

eth1，有些系统也会以

em等

                     字符标识。

HWADDR=00:

0c:

29:

a5:

3f:
```

39 ←以太网硬件地址，即

MAC地址，如果是

VMware克隆的虚拟机

无法启动网卡，可以毫不犹豫地删除此项。

TYPE=Ethernet ←上网类型，目前基本都是以太网。

UUID=176582f7-d198-4e4f-aab0-34ab10d17247
←通用唯一识别码

（

Universally Unique Identifier），

如果是

VMware克隆的虚拟机无法启动网卡，可以去除此项。

ONBOOT=no ←这个地方要设置为

yes，才能保证下次开机启动时激活网卡设备。

NM_CONTROLLED=yes ←是否通过

NetworkManager管理网卡设备。

BOOTPROTO=none ←启动协议，获取配置方式，有

none|bootp|dhcp三个选项。

IPADDR=10.0.0.8 ←这是虚拟机桥接模式，局域网

Linux服务器的固定

IP。

NETMASK=255.255.255.0 ←子网掩码，用来规划网络位和主机位，一般为

255.255.255.0。

DNS2=8.8.8.8 ←第二个

DNS，这里默认会覆盖，以及优先于

/etc/resolv.conf
的配置生效。

GATEWAY=10.0.0.254 ←局域网上网网关地址。

DNS1=202.106.0.20 ←主

DNS，这里默认会覆盖，以及优先于

```
/etc/resolv.conf的

                    配置生效


IPV6INIT=no          ←是否支持


IPV6
USERCTL=no
```

提示：更多网卡配置相关知识请执行less/usr/share/doc/initscripts-*/sysconfig.txt查看。

无论虚拟机采用的是NAT模式还是桥接模式，或者是物理服务器，都要修改网卡启动项配置，把“ONBOOT=no”修改为“ONBOOT=yes”，使得下一次开机时网卡可以自启动。命令如下：

```
[root@www ~]# sed -i 's#ONBOOT=no#ONBOOT=yes#g' /etc/sysconfig/network-scripts/ifcf
[root@www ~]# grep ONBOOT /etc/sysconfig/network-scripts/ifcfg-eth0
ONBOOT=yes
```

重启网卡的命令如下：

```
[root@www ~]# ifdown eth0 && ifup eth0
Determining if ip address 10.0.0.7 is already in use for device eth0...
```

提示：尽量不要使用/etc/init.d/network restart重启网卡，因为这条命令会影响所有的网卡。

查看IP设置的命令如下：

```
[root@www ~]# ifconfig eth0
eth0      Link encap:


Ethernet  HWaddr 00:


0C:


29:


CC:


60:


B2
          inet addr:


10.0.0.7  Bcast:


10.0.0.255  Mask:


255.255.255.0
...省略若干
```

查看默认网关设置的命令如下：

```
[root@www ~]# route -n
Kernel IP routing table
Destination   Gateway       Genmask         Flags Metric Ref    Use Iface
10.0.0.0      0.0.0.0       255.255.255.0   U     0      0        0 eth0
169.254.0.0   0.0.0.0       255.255.0.0     U     1002   0        0 eth0
```

```
0.0.0.0         10.0.0.254      0.0.0.0         UG    0      0        0 eth0
```

查看DNS设置的命令如下：

```
[root@www ~]# cat /etc/resolv.conf;


 generated by /sbin/dhclient-script
search localdomain
nameserver 8.8.8.8
nameserver 202.106.0.20
```

如果以上配置都正确，一般来说上网就没有问题了。

提示：老男孩在使用CentOS 6.6时，发现在/etc/sysconfig/network-scripts/ifcfg-eth0网卡文件配置的DNS设置会覆盖/etc/resolv.conf中的设置，即使/etc/sysconfig/network-scripts/ifcfg-eth0网卡文件不设置DNS，只在/etc/resolv.conf中配置DNS设置，执行/etc/init.d/network restart重启网卡命令仍会清除/etc/resolv.conf的DNS设置。这点请读者注意，建议在/etc/sysconfig/network-scripts/ifcfg-eth0网卡文件中配置DNS，而不是在/etc/resolv.conf中设置，以免遇到上网无法解析的问题。

2.4.4 更新系统，打补丁到最新

与Windows打补丁类似，Linux也可以定期更新系统软件。

Linux下安装软件的优秀工具叫做yum，它是CentOS Linux下最好用的包管理器和安装软件包的工具，用起来很方便。Linux的二进制软件包一般是rpm包，类似Windows下的exe程序。

通过yum工具可以很方便地安装rpm软件包，默认获取rpm包的软件配置是从国外CentOS官方源及其镜像地址下载的。因此，通过yum工具安装软件，速度会比较慢，因此需要把默认获取rpm包的地址改成国内的yum源地址。

CentOS安装完毕后，首先应该修改更新源，并升级到最新的系统。修改更新yum源的命令如下：

```
CentOS 6
cp /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.ori
wget -O /etc/yum.repos.d/CentOS-Base.repo http:


//mirrors.163.com/.help/CentOS 6-Base-163.repo
```

提示：老男孩在写书过程中，刚开始选择了aliyun上的yum源，但遇到了更新错误，后来改成网易的就好了。大家遇到问题时，不妨更

换yum源。下面为阿里云的yum源配置参考：http://mirrors.aliyun.com/help/centos 。

然后，使用如下命令将系统更新到最新状态：

```
[root@www ~]# ll /etc/pki/rpm-gpg/总用量


 16
-rw-r--r--. 1 root root 1706 10月


 23 2014 RPM-GPG-KEY-CentOS-6
-rw-r--r--. 1 root root 1730 10月


 23 2014 RPM-GPG-KEY-CentOS-Debug-6
-rw-r--r--. 1 root root 1730 10月


 23 2014 RPM-GPG-KEY-CentOS-Security-6
-rw-r--r--. 1 root root 1734 10月


 23 2014 RPM-GPG-KEY-CentOS-Testing-6
[root@www ~]# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY*
[root@www ~]# yum update -y
```

 提示：也可以使用yum upgrade-y，此时大约会有127MB大小的更新包，这个更新是可选操作。

一般在首次安装系统时可以执行yum update-y，如果是在生产线已经应用的业务服务器系统中就不要轻易更新了，以免导致业务服务异常。若遇到了特殊的漏洞（例如2014年4月发生的openssl心脏出血及2014年10月发生的bash漏洞，可以单独对指定的问题软件进行升级处

理。yum upgrade与yum update的作用和区别很小，想了解的读者可执行man yum进行查看。

2.4.5　额外安装一些有用的软件包

按照上文选择最小化安装时，会有一些有用的工具包没有安装进去，此时就可以安装这些软件工具了，命令如下：

```
[root@www ~]#yum install tree telnet dos2unix sysstat lrzsz nc nmap -y
```

另外，如果在安装时落下了某些需要的软件包组，可以执行如下命令来安装。

```
[root@www ~]#yum grouplist <==查看所有包组名称，包括已安装的和未安装的。

[root@www ~]#yum groupinstall "Development Tools" <==指定包组名安装，注意要带双引号。
```

提示：yum grouplist用于查看包组列表。

到此为止，整个CentOS 6.6的安装就告一段落了，大家在用虚拟机学习时可以保留这个模板机的配置，然后每次克隆新的虚拟机（推荐使用链接克隆节省空间）用来学习，当然，也可以用快照的方式来设定及回滚操作过的设置。

2.5 本章重点回顾

1）安装Linux系统前，要事先规划好如何分区，做好安装包选择。

2）Linux系统的分区知识及企业环境不同业务场景及服务器角色对应的分区方案。

3）用练习机安装时可设置/、/boot、swap三个分区，这也是工作中常用的分区方法。如果是在工作环境中，对于数据库及存储的服务器还可以再分出一个/data分区。

门户网站一般的分区方案：假设服务器内存为16GB，硬盘为1TB。

/boot分区：100~200MB

swap分区：物理内存的1.5~2倍，如果内存大于16GB，可以配置为8~16GB。

/分区：80~200GB

剩余空间不分，保留给使用的人根据业务中的具体问题进行划分。

4）若要调整开机启动设备的顺序，必须要重新启动并进入BIOS进行调整。

5）安装CentOS 6.6时使用的是图形界面，而不是文本界面。

6）安装过程中进入分区后，请以“自定义的分区结构”来处理自己规划的分区方式。

7）一般要求swap应该是1.5~2倍的物理内存大小。即使没有swap，依旧能够安装和运行Linux操作系统。

8）CentOS 5.x-6.x的引导程序为grub，最好选择安装在MBR中。

9）安装软件包组的选择是关键，不能太多也不能太少。

10）若连不上Internet，可以尝试查看网卡是否启动，上网模式是否正确，以及IP、网关、DNS等的设置是否正确。

2.6 本章知识相关考试题

1）32位和64位系统的区别是什么？

2）请描述Linux分区的知识（包括设备名、主分区、扩展分区、文件系统类型等）。

3）什么是挂载点，挂载点的作用是什么？

4）企业场景下如何针对不同的业务服务器规划分区方案？

5）企业场景下Linux系统安装如何尽可能地最小化选包？

6）企业场景下若线上运行的系统缺少部分包组，如何补救？

第3章　CentOS 6.6连接管理及优化

3.1　远程连接Linux系统管理

3.1.1　为什么要远程连接Linux系统

很多机构的培训教学，都是直接在虚拟机界面上讲解知识，这就导致非常多的学生只熟悉如何在虚拟机界面上操作，对实际的工作场景并不熟悉。事实上，在实际的工作场景中，虚拟机界面或物理服务器本地的窗口都是很少能够接触到的，因为服务器装完系统后，都要拉到IDC机房托管，如果购买了云主机，更碰不到服务器本地显示器了，此时只能通过远程连接的方式管理Linux系统。因此，在装好Linux系统后，学习Linux运维的第一步应该是配置好客户端软件远程连接Linux系统进行管理。

另外，需要指出的是，学习Linux的最佳方式就是利用虚拟机来学习，而不是安装双系统或直接抛弃Windows系统改装Linux系统。通过模拟生产环境来学习是非常必要的，而虚拟机环境正是上面几种方式中最接近企业真实运维工作环境的，当然，如果能有一个真实的物理机放在局域网或机房里就更好了。

3.1.2　远程连接Linux的原理

1.SSH远程连接介绍

当前，在几乎所有的互联网企业环境中，最常用的提供Linux远程连接服务的工具就是SSH软件了，SSH分为SSH客户端和SSH服务器端两部分。其中，SSH服务器端包含的软件程序主要有openssh和openssl，在Linux系统中可以用如下方法查询SSH服务器端工具的安装情况。

```
[root@www ~]# rpm -qa openssh openssl
openssl-1.0.1e-30.el6.x86_64
openssh-5.3p1-104.el6.x86_64
```

提示：openssh是提供SSH服务的程序，openssl是为SSH提供连接加密的程序。

2.SSH服务器端介绍

启动Linux系统时，默认情况下，SSH服务器端程序就会随系统一起启动，SSH服务是一个守护进程（daemon），它在系统后台永久运行并时刻响应所有来自SSH客户端的连接请求。SSH服务器端的进程名为sshd，负责实时监听远程SSH客户端的连接请求并进行处理，这些请求一般包括公共密钥认证、密钥交换、对称密钥加密和非安全连接等。SSH服务是Linux系统优化时需要保留开机自启动的服务之一。

3.SSH客户端介绍

SSH客户端最常用的工具就是Windows平台上运行的SecureCRT了，该工具安装很简单，按提示一步步操作即可安装完毕，然后打开并注册软件，运行就可以使用了。除SecureCRT客户端软件之外，还有xshell、putty及Linux下的SSH客户端软件。

4.SSH协议介绍

SSH服务器端和SSH客户端之间的交流都是通过SSH协议来实现的。SSH协议是Secure Shell Protocol的简写，由IETF网络工作小组（Network Working Group）制定。在进行数据传输之前，SSH先通过加密技术对联机数据包进行加密处理，然后再进行数据传输，这样就可确保传递的数据安全。

SSH是专为远程登录会话和其他网络服务提供的安全性协议。利用SSH协议可以有效地防止远程管理过程中的信息泄露，在当前的生产环境中，绝大多数企业普遍采用SSH协议服务来代替传统的不安全的远程联机服务软件，如telnet（23端口，非加密的）等。

SSH协议有两个不兼容的版本，分别是SSH 1.x和SSH 2.x。

OpenSSH同时支持SSH 1.x和SSH 2.x。用SSH 2.x的客户端程序不能连接到SSH 1.x的服务程序上，SSH 2.x比SSH 1.x更安全，默认情况下服

务器端通过SSH 2.x协议提供服务。

5.SSH客户端和SSH服务器端远程连接原理示意图

图3-1为SSH客户端和SSH服务器端远程连接的形象示意图，表3-1为SSH服务远程连接需要设定的五个要素。

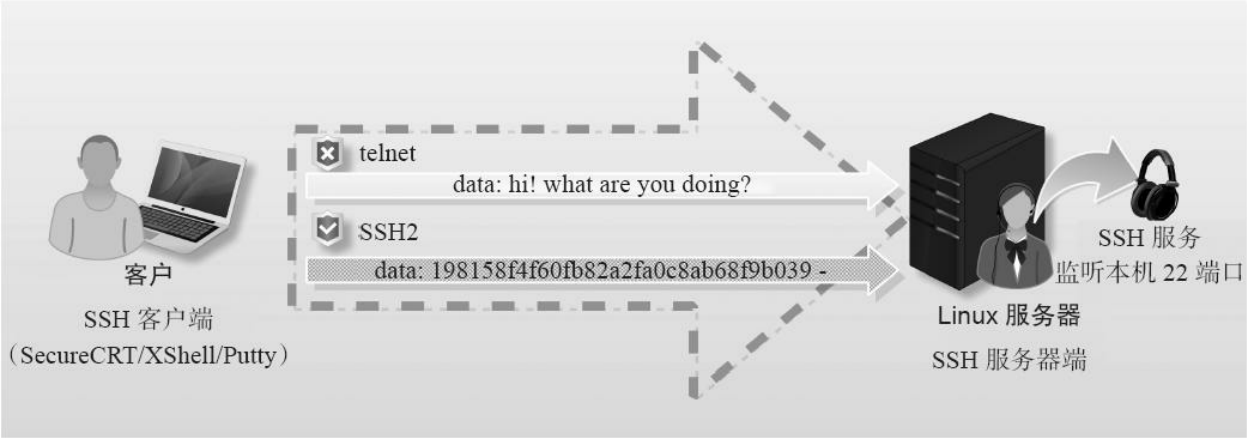


图3-1　SSH通信示意图

表3-1　SSH服务远程连接需要设定的五个要素

协议	IP	端口	用户名	密码
SSH（加密的）	10.0.0.7	22	oldboy	oldboy
telnet（未加密）	10.0.0.7	23	oldboy	oldboy

6.SSH远程连接故障排查示例

假设远程Linux服务器的IP地址为10.0.0.7，现在要进行连接故障排查。

首先，查看远端服务是否畅通，注意：操作命令的机器为客户端计

算机命令行，基本检查思路如下。

（1）利用ping命令检查（客户端执行）

具体命令为：

```
ping 10.0.0.7          ←此命令适合

Linux和

Windows
```

这里通过ping命令从客户端发包到服务器，看看是否有数据包返回，从而确定物理链路是不是通畅。打个比喻就是想开车去玩，先确认到目的地的高速路是否修通了。


提示：也可以通过tracert-d 10.0.0.7（仅适合Windows）或traceroute 10.0.0.7-n（适合Linux）命令跟踪路由情况，这两个跟踪路由的命令一般用于广域网。

（2）利用telnet或nmap命令检查（客户端执行）

具体命令为：

```
telnet 10.0.0.7 22
```

或

```
nmap 10.0.0.7 -p 22      仅适合

Linux，需要安装该软件包后才能使用。
```

通过该命令可以查看连接服务器端10.0.0.7的22端口是不是处于开通状态，因为SSH服务默认开启的是22端口。同样可以比喻为想开车去玩，在高速路通了以后，要确认旅游景点是否开放。

（3）检查iptables等防火墙策略是否阻挡了连接（服务器端执行）

具体命令为：

```
/etc/init.d/iptables status
```

初学者在学习期间最好关掉iptables，等日后学习了iptables后再开启。打个比喻即确认是否因为天气原因例如大雾或下雪，高速路遇阻无法通行了。

7.SSH远程连接故障排查思路逻辑图

为了让读者能更形象地了解SSH远程连接故障排查思路，特用图解说明，如图3-2所示。

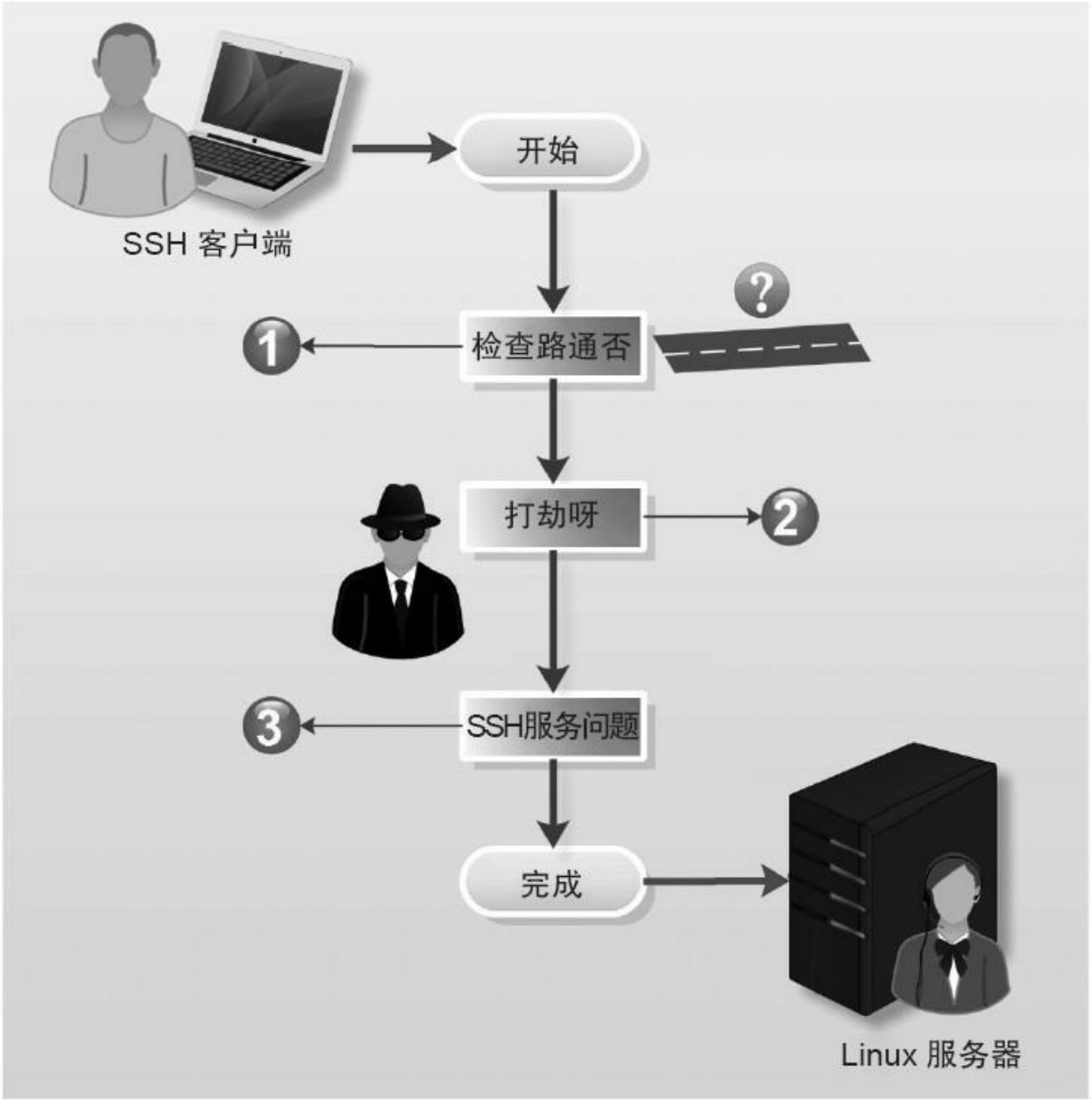


图3-2　SSH远程连接故障排查逻辑图

表3-2为SSH故障排查思路小结。

表3-2　SSH故障排查思路

第 1 步：物理链路是否有问题，比喻为“高速路是否修通？”（客户端执行）	
ping 10.0.0.7	排查客户端到服务器端线路问题，ping 是常用的网络连通性检查工具（路通否）
tracert -d 10.0.0.7	路由跟踪命令，也可以检查路是否通畅，-d 是不进行反向解析
第 2 步：SSH 服务是否有问题，比喻为“旅游景点是否开放？”（客户端执行）	
telnet 10.0.0.7 22	判断 SSH 服务器默认的 22 端口是否打开（客户端执行）。一看端口是否打开，二看端口是否改变了，三看防火墙等
nmap 10.0.0.7 -p 22	也能达到和 telnet 相同的效果，但 nmap 仅在 Linux 中使用。
第 3 步：是不是防火墙阻挡，即是不是下雪封路，高速路遇阻无法通行了（服务器端查看）	
/etc/init.d/iptables stop	Linux 防火墙 iptables，可能好心办坏事，阻挡了远程连接，学习环境建议关掉防火墙，生产中按需开启

3.1.3　远程连接Linux的客户端工具介绍

在介绍远程连接Linux的工具之前，先回忆一下Windows下曾经用过的远程连接服务器的工具，其中包括远程桌面mstsc、vnc、pcanywhere等，在Windows服务器端这些工具分别对应不同的服务进程来响应客户端的连接请求。如果读者熟悉这些工具，就可以发现，这些工具远程连接后都是图形管理的，而Linux系统的远程连接工具多数是用命令行管理的，例如SecureCRT。

远程连接Linux服务器的常见工具有SecureCRT、Xshell、Putty等，其中最常用的是SecureCRT和Xshell。这些客户端工具在Linux服务器端对应着相同的SSH服务进程sshd，即远程连接都使用SSH协议，当然它们也支持其他协议，如telnet。

事实上，远程连接Linux服务器也有图形显示工具，例如vnc工具。不过，通常只在特殊的工作场景才会临时用到图形远程管理，例如安装Oracle的场景，绝大多数企业场景都无须桌面管理，而且安装Oracle也不是必须要用图形管理，只要能把图形导出来显示就行了，实现工具有xmanger、x-win32等。

3.1.4　如何选择远程连接Linux的工具

在互联网企业里，通常都习惯使用命令行管理Linux，所以选择命令行管理工具是第一位的。

若Linux服务器端守护程序选择了SSH服务，则整个连接过程都会加密，因此比较安全。连接Linux服务器的客户端如果是Windows系统，可以选择SecureCRT或Xshell工具，推荐使用SecureCRT，理由是其历史悠久、简单、用的人多。至于Linux，就用openssh自带的SSH客户端工具即可。如果需要用图形远程管理，建议选择vnc。相关工具列表见表3-3。

表3-3　SSH服务器端及客户端工具列表

服务角色	文本命令行管理工具	图形管理工具
Linux 服务器端	sshd（openssh openssl）	vnc server
客户端	SecureCRT 或 Xshell、Putty、SSH	vnc client、xmanger、x-win32

3.2 SSH客户端常用工具SecureCRT

3.2.1 SecureCRT工具介绍

SecureCRT是一款支持SSH（SSH1和SSH2）协议的终端仿真软件，常被用来运行于Windows下远程登录UNIX或Linux服务器主机。

SecureCRT软件功能强大，不仅仅支持SSH协议，同时还支持Telnet、RLogin、Serial和TAPI等协议，它有非常多的功能，这里就不一一介绍了，常用功能可看下文实践。

与SecureCRT类似功能的SSH软件还有Xshell、Putty等。SecureCRT、Xshell、Putty等都仅仅是客户端软件，一般被用于Windows客户端的计算机，因此，无论选择哪款客户端SSH工具，对于学习来说都是可以的，读者可以根据自身习惯进行选择，这里老男孩以比较大众化的SecureCRT为例为读者阐述SecureCRT的常用优秀功能。

3.2.2 SecureCRT工具安装说明

由于是在Windows下进行安装，因此操作步骤很简单，按提示一路单击“下一步”即可完成，这里就不一一叙述了，不过要提醒的是，请注意看SecureCRT软件目录下的帮助，要注册SecureCRT后才可以运行使用。

3.2.3　配置SecureCRT连接Linux主机

在笔记本或客户端计算机上正确安装SecureCRT SSH连接工具后，通过开始程序或快捷菜单找到SecureCRT并打开，则会看到如图3-3所示的界面。

在图3-3所示的界面，单击工具栏中的第二个快捷按钮（即“快速连接”按钮），则会进入如图3-4所示的窗口。

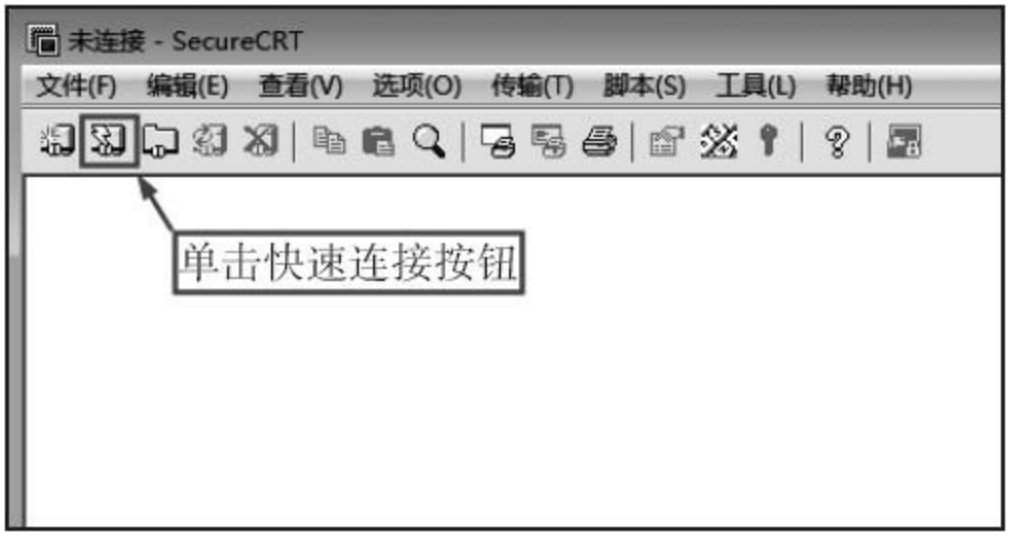


图3-3　SecureCRT界面

图3-4　快速连接服务器配置端口

在该窗口中，首先选择连接的协议为SSH2（也可以选择其他协议），然后在下面的选框中，配置Linux系统的IP地址及默认的端口号22，接下来配置连接的用户名，默认是Linux超级管理员root，最后在右下角勾选“保存会话”和“在一个标签中打开”，并单击“连接”，此时会进入如图3-5所示的提示框。

这里是主机密钥验证，只有在第一次连接时才会出现这个提示框，单击“接受并保存”按钮，然后进入如图3-6所示的提示窗口。

图3-5　新建主机密钥提示

图3-6　提示输入口令框

在“口令”栏旁边的选框敲入安装Linux系统时设置的root密码，然后勾选左下角的“保存口令”，最后单击“确定”按钮。如果顺利的话，应该会出现登录成功后的Linux Shell命令行提示窗口，如图3-7所示。

此时就可以像在VMware虚拟机里或物理机连接显示器那样管理Linux系统了，你甚至感受不到是远程管理，因为都是文本操作，操作非常流畅，哪怕是距离服务器几千公里。

3.2.4　通过SSH工具连接Linux主机的常见问题

如果你的操作比较规范，且是在局域网内，那么一般情况下不会有连接问题；如果你的操作有疏漏之处，而且是广域网或拨号上网，那么就有可能存在一些问题。下面就与大家分享一下部分常见问题的产生原因及解决办法。

1.连接超时问题

如果完成上一节介绍的设置步骤后依然连接不上Linux主机，可能会出现连接超时的提示，如图3-8所示。

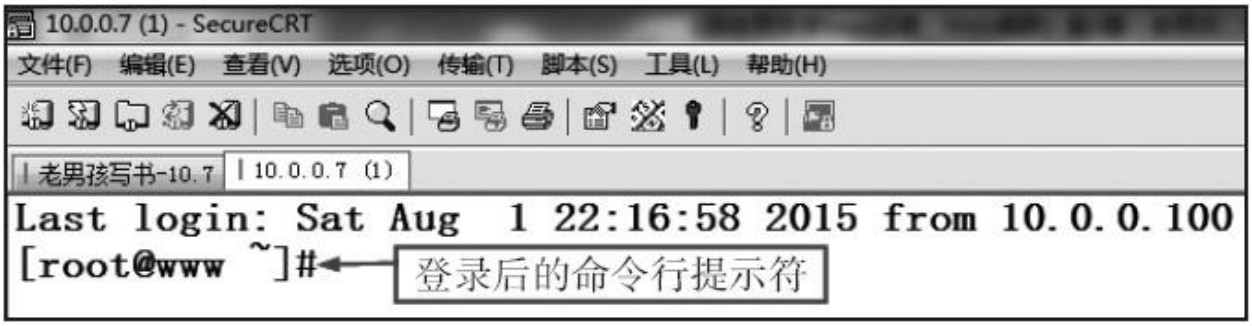


图3-7　正确登录系统后的窗口及命令行

图3-8　连接服务器超时提示

如果遇到这个问题，请检查Linux系统和客户端计算机端的网络连接是不是通畅的，最简单的方法就是在客户端计算机上ping一下Linux主机的IP。

ping的实际命令如下：

```
C：

\Users\oldboy>ping 10.0.0.7正在

 Ping 10.0.0.7 具有

 32 字节的数据：

来自

 10.0.0.100 的回复：

 无法访问目标主机。

来自

 10.0.0.100 的回复：

 无法访问目标主机。

提示：这是不通的状态
```

```
C：

\Users\oldboy>ping 10.0.0.7正在

 Ping 10.0.0.7 具有

 32 字节的数据：

来自

 10.0.0.7 的回复：

 字节

=32 时间

<1ms TTL=64来自

 10.0.0.7 的回复：

 字节

=32 时间

<1ms TTL=64提示：这是通的状态
```

如果未能连通Linux主机的IP，则可通过如下方法进行检查：

1）Linux主机本身是不是正确设置或获取到了IP，并且这个IP与客户端计算机的IP是不是在同一个网络里。

2）客户端计算机和Linux主机之间的22端口是不是被Linux系统的防火墙给阻挡了。关闭Linux防火墙的方法为：

```
[root@www ~]# /etc/init.d/iptables stop                    <==最好连续执行两遍
```

2.SSH端口问题

如果出现的是远程系统拒绝连接的提示（如图3-9所示），则要从SSH端口查找原因。

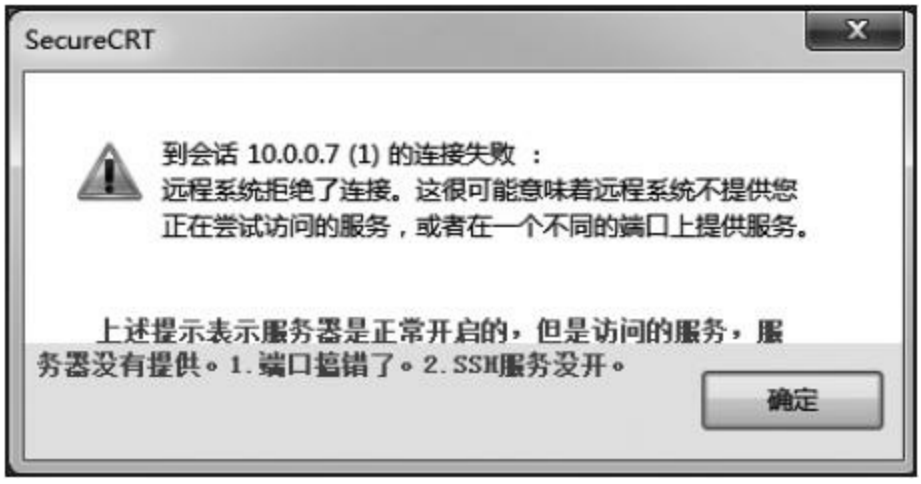


图3-9　由于端口问题导致的连接失败图

可从如下几个方面来查找原因。

1）sshd服务程序是否正确开启，确认的示例命令如下：

```
[root@www ~]# ps -ef|grep sshd|grep -v grep
root   364   1  0 Oct06      00:


00:


00 /usr/sbin/sshd <==有这一行表示服务正常
```

2）sshd服务的默认端口22是不是被更改了，确认的示例命令如下：

```
[root@www ~]# netstat -lntup|grep sshd
tcp    0    0 0.0.0.0:


22      0.0.0.0:


*           LISTEN      364/sshd
tcp    0    0 : : :


22           : : :


*               LISTEN      364/sshd提示:


22为端口，如果被改就是其他数字了。
```

3）是否因开启了iptables防火墙而导致禁止连接，确认的示例代码

如下：

```
[root@www ~]# /etc/init.d/iptables stop
iptables:


 Setting chains to policy ACCEPT:


 filter          [  OK  ]
iptables:


 Flushing firewall rules:


                            [  OK  ]
iptables:


 Unloading modules:


                                  [  OK  ]
[root@www ~]# /etc/init.d/iptables stop
```

4）客户端SecureCRT是不是配错了连接的端口或IP。

如果你熟悉telnet命令，也可以在客户端计算机上执行telnet 10.0.0.722来检查连通性。SSH端口问题大体上都是由上述4类情况引起的，更多原因可以到网上搜索，或者加入本书前言提供的官方QQ群进行交流。

3.2.5　调整SecureCRT终端显示和回滚缓冲区大小

磨刀不误砍柴工，为了更方便地学习Linux，首先得对终端进行一些调整，步骤如下：通过SecureCRT顶部菜单中的“选项”→“会话选项”，打开会话选项 窗口，然后单击左边菜单“终端”→“仿真”，并勾选窗口右边的“ANSI颜色 ”项，再到“终端”右边的选框里选择“Linux”，最后设置回滚缓冲区为32000，并单击“确定”即可（如图3-10所示）。

图3-10　会话选项图

调整终端显示和回滚缓冲区的说明如下：

·调整“终端”→“仿真”的ANSI颜色，并且将终端选择为Linux（有网友选择Xterm，也是可以的），目的是让Linux命令行看起来更舒服，如果开发Shell/Python程序，更利于代码展示，比如会高亮显示代码等。

·调整回滚缓冲区的目的是当操作内容过多时，想回看操作过的记录，可以向上翻得更远一些。

3.2.6　调整字体及光标颜色

单击用户管理界面左边菜单中的“外观”，在右侧内容区会显示与“窗口和文本外观”相关的选项，如图3-11所示。

图3-11　调整字体光标等

在图3-11所示的界面中单击“颜色”，在基本颜色处选择“绿色”，然后单击“确定”按钮，如图3-12所示。

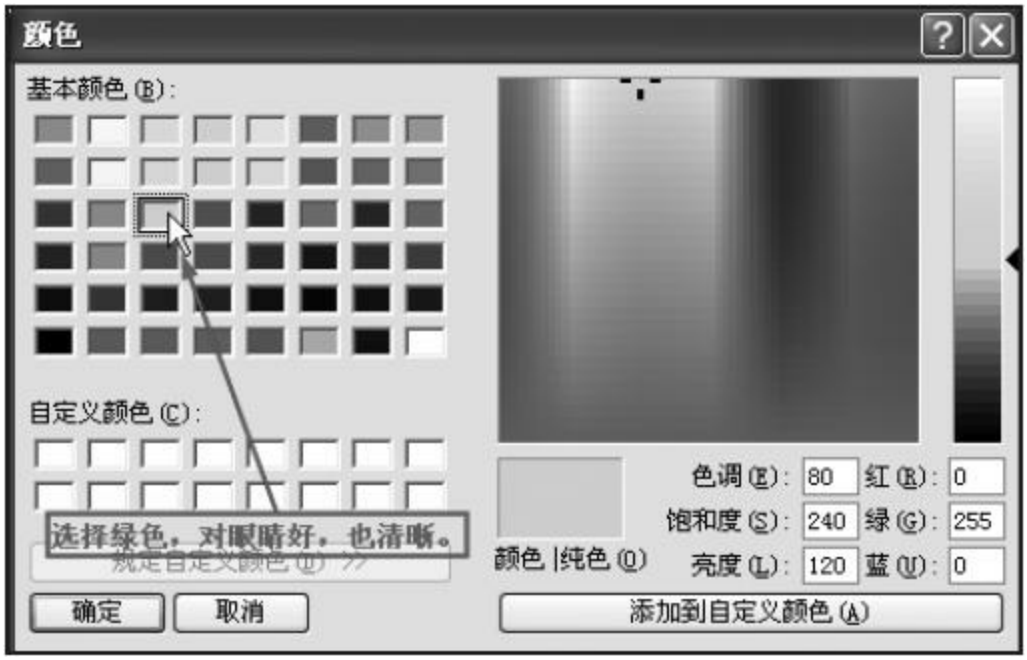


图3-12　光标颜色调整

提示：如果觉得字体够大，可以不调整，不过，给光标换个颜色还是不错的，定位光标时会比较清晰快速。

3.2.7　配置记录SSH操作的日志及输出

设置在SecureCRT中记录执行命令及屏幕输出的日志文件，可以更方便地查询曾经操作过的配置、命令及结果输出等（对于大家的学习和工作都有用处）。

日志文件的格式为：%H_%Y%M%D.log（日志文件名以主机IP、年、月、日的形式记录）。

单击用户管理界面左边菜单的“日志文件”，在右侧内容区会显示与“日志文件”相关的选项，配置内容如图3-13所示。

图3-13　配置SSH日志图

配置说明如下：

·在图3-13中，一定要勾选“在连接上启动记录”，如果不勾选则不会记录。

·要勾选“追加到文件”。

·如果单独配置这部分内容，需要退出连接重新登录，上述配置才能生效。

提示：还可以将记录文件名配置为更为详细的格式。如，%H_%S_%Y%M%D%h%m.log，在图3-13所示的日志文件中配置即可。

3.2.8 配置本地机器上传和下载的目录

在用户管理界面，单击左边菜单的“Xmodem/Zmodem”，然后就可在右侧的目录区中设置文件的上传和下载目录了，如图3-14所示。

上传和下载的路径可以设置为同一个，但所选择的路径必须在系统中存在才行。

设置完毕后，就可以通过SecureCRT连接的Linux命令行经由rz上传文件到Linux系统了，通过“sz文件名”则可以下载文件到上述配置的路径里，从而实现客户端计算机和Linux主机的文件传输。

图3-14　rz、sz上传和下载Windows默认路径

下面来看一下有关上传和下载命令的说明。

1.rz、sz命令的安装方法

第一种方法：安装系统时选包含rz、sz命令的包组，即 ☑ **Dial-up Networking Support** 。

第二种方法：安装系统后通过执行yum install lrzsz-y或yum groupinstall"Dial-up Networking Support"-y命令来安装。

2.上传命令rz

上传内容时，执行rz命令，如果希望覆盖服务器上的同名内容上传，可加-y参数，输入“rz-y”命令后直接按回车键，会打开一个上传文件的窗口，然后从这个窗口浏览找到需要传输的文件进行上传，在这个上传的窗口内选择的文件是客户端计算机本地要上传到Linux的文件。

3.下载命令sz

下载内容时，执行命令“sz filename”，如果希望覆盖本地的同名内容下载，则可输入“sz-y filename”命令，“sz-y”命令后面的“filename”为命令行Linux主机当前目录下的文件，例如：sz oldboy.log。使用sz或“sz-y文件名”命令从服务器上下载文件时，默认的客户端下载路径就

是图3-14所设置的下载路径地址。

4.使用rz、sz命令的注意事项

·只能上传和下载文件，不能上传和下载目录，如果是目录需要打包成文件再传。

·上传的文件可以是计算机里的任意文件，下载的文件会下载到SecureCRT配置的对应下载路径目录中。

·执行rz命令按回车键后出现的窗口中，一定不要勾选最下方的“以ASCII方式上传文件”，否则会遇到问题，如图3-15所示。

图3-15　上传文件注意事项图

5.其他工具

除了rz、sz等传输文件命令外，还可以用ftp、sftp（SSH服务）等工具来传输文件。

3.2.9　实现批量部署和管理功能

要实现这个功能有一个前提，就是前面介绍连接时所讲的要确保所有的标签在同一个SecureCRT窗口打开（如图3-16所示）。

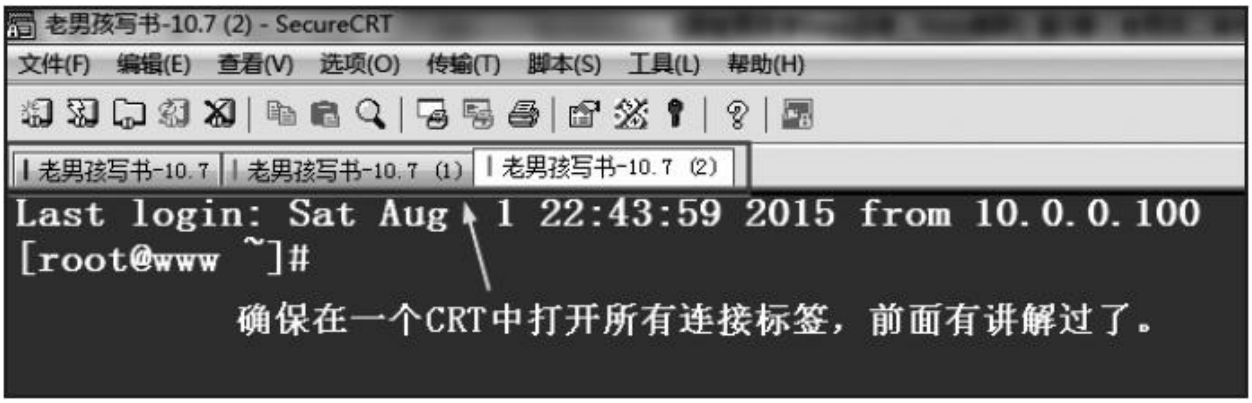


图3-16　多标签批量管理登录窗口

如果不是在同一个标签中打开的，请按图3-16和图3-17所示的进行调整：打开SecureCRT连接窗口，选中一个连接标签，然后勾选右下角的“在一个标签中打开”按钮，这样以后这个连接就会在一个窗口中打开了。

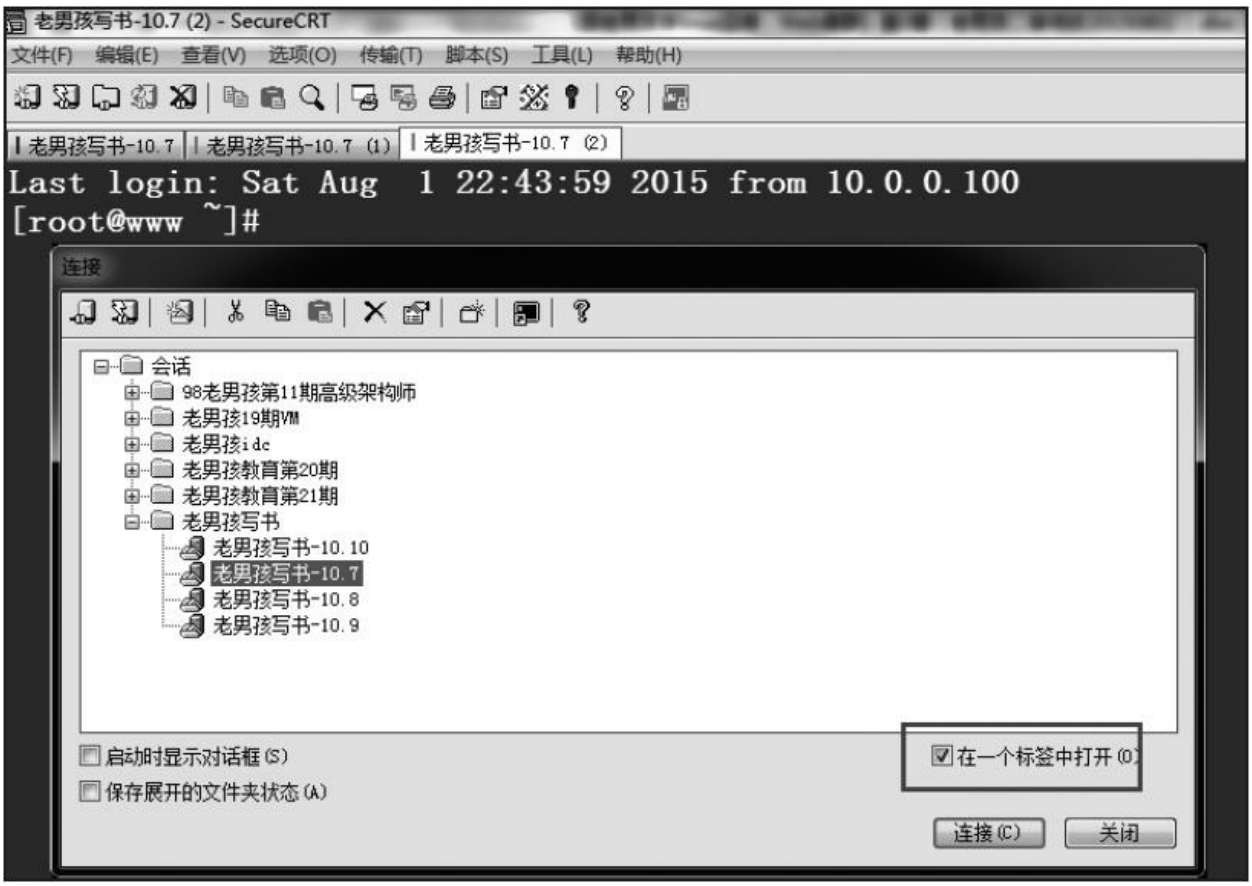


图3-17　在一个标签中打开设置图

之后，把如图3-18所示的窗口缩小（不要最大化，否则交谈窗口按钮是灰色的），并单击菜单中的“查看”→“交谈窗口”，就会出现如图3-19所示的底部空白窗口。

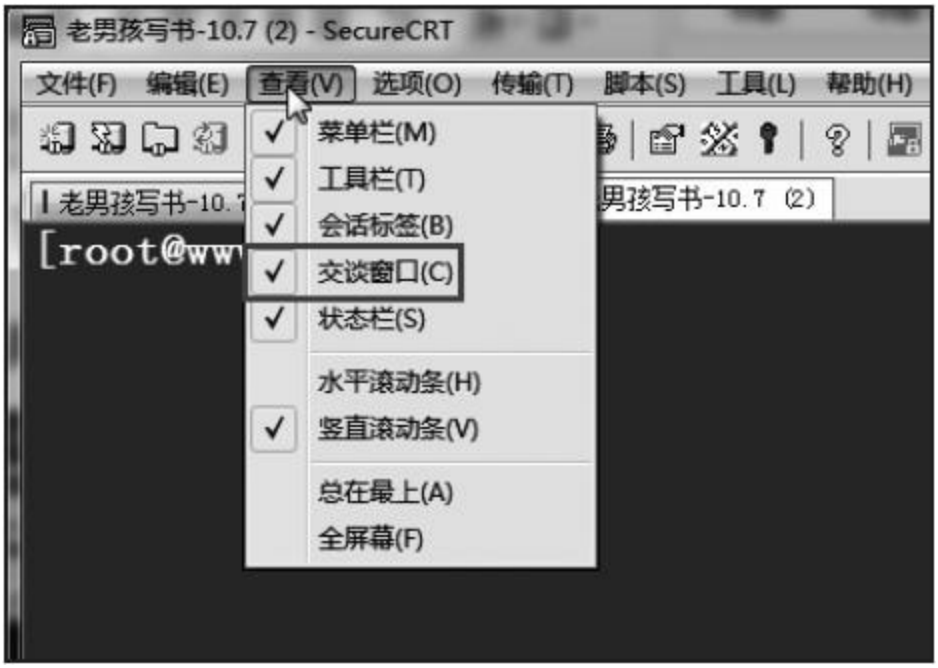


图3-18　设置批量管理输入框

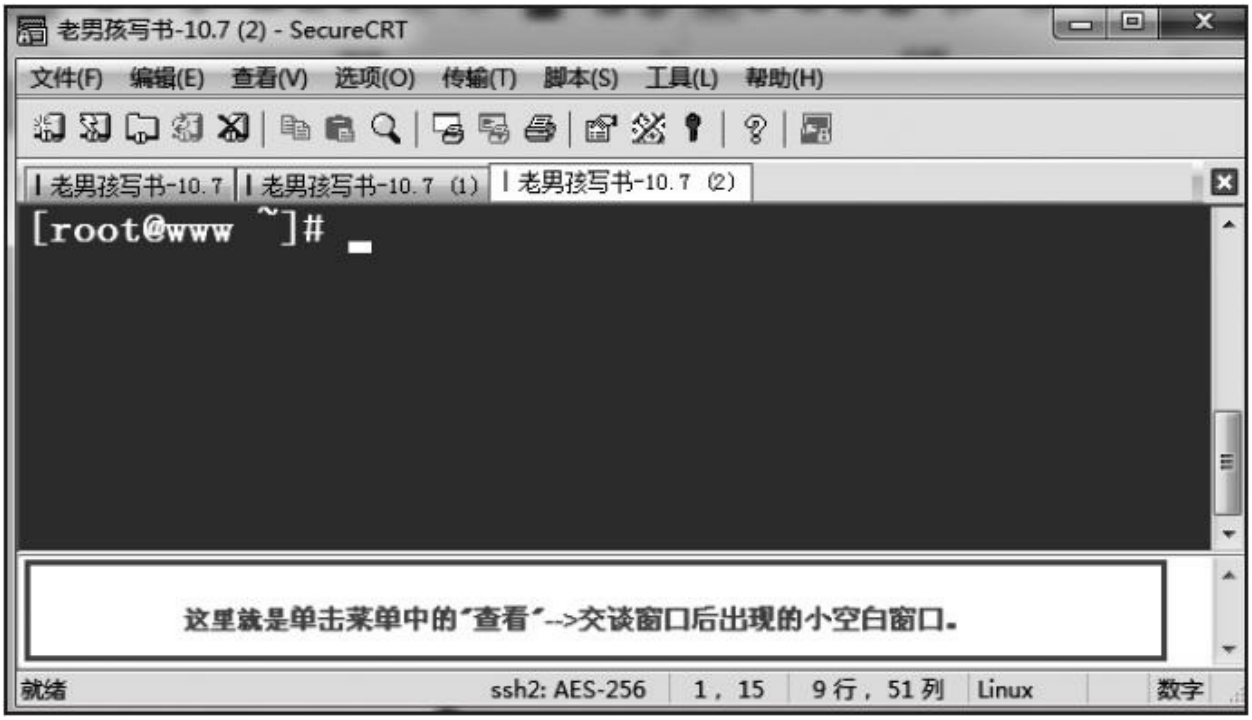


图3-19　批量执行命令设置窗口

在空白窗口处单击鼠标右键，然后选择“将交谈发送到所有标签”，

如图3-20所示。

图3-20　快捷菜单

最后批量部署执行命令的结果如图3-21所示。

提示：如果要批量部署或执行任务的服务器为数十台，就可以利用SecureCRT的这个小功能，非常不错，简单易用，可以替代一些大型的批量部署软件。如果是大规模服务器数量，则可以用saltstack等批量管理工具。

特别需要注意的是，上述批量管理的操作，不能使用交互的命令，例如vi/vim、rz等。但可以变通使用，例如vi/vim命令可以用echo、cat、sed替代。

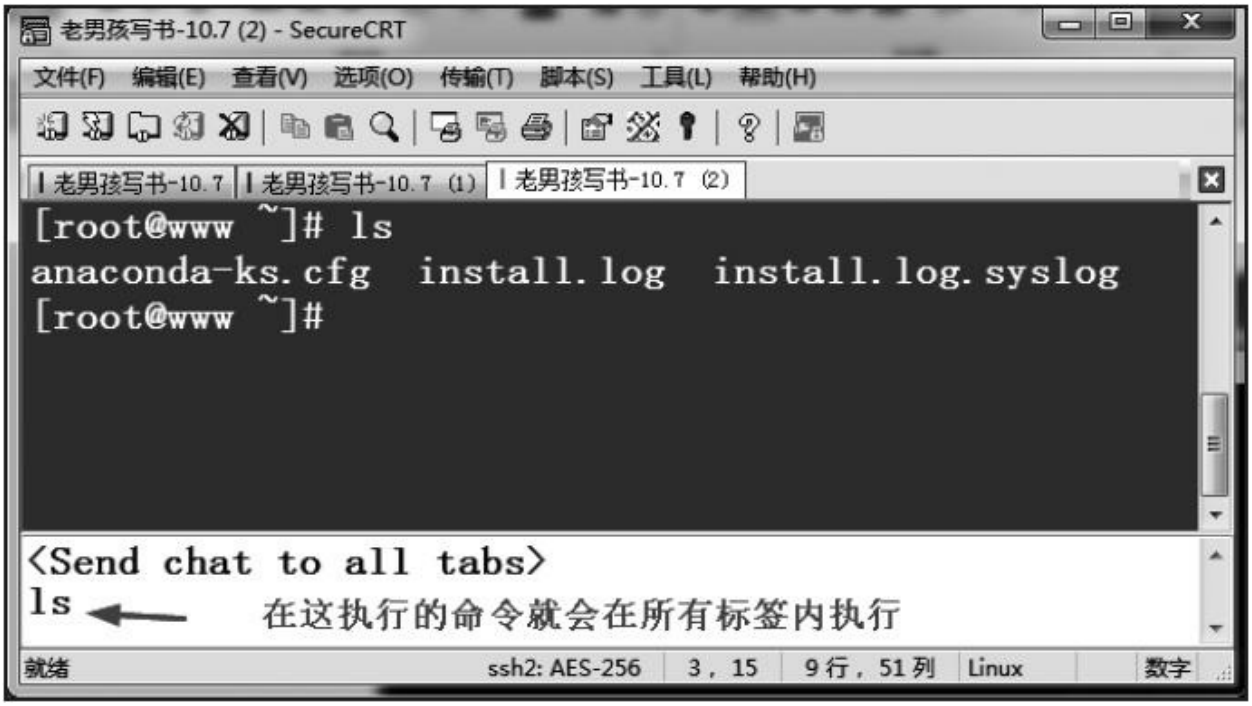


图3-21　批量执行命令演示图

3.2.10　配置SecureCRT标签路径

工作久了，SecureCRT的标签就会有很多，当重装系统或换客户端计算机时，这些标签就会丢失，因此需要对标签进行拷贝以实现备份，可找到全局选项 对配置文件夹的目录进行配置，如图3-22所示。

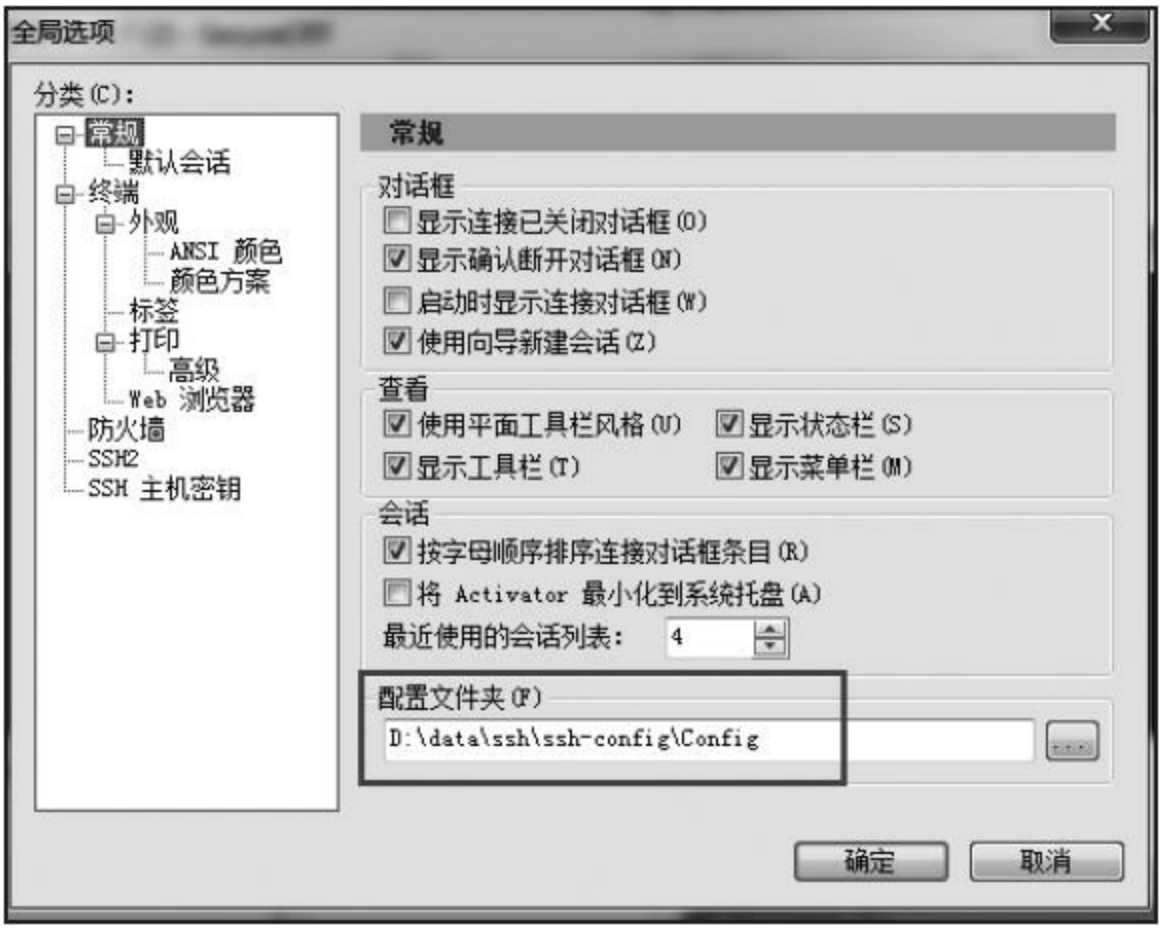


图3-22　SecureCRT标签信息存放路径

3.2.11　配置标签模板

前面做了非常多的配置，若是再新建标签连接服务器，如何继承这些配置呢？

方法1：通过对一个已经配置好的标签实施复制、粘贴操作来新建标签（如图3-23所示），这样它就继承老标签的配置，只需要再修改相应的特殊配置即可，如IP、会话名字等。

方法2：在全局选项里配置上述的各种功能设置（如图3-24所示），这样，每次新添加标签时都会使用全局的默认设置。

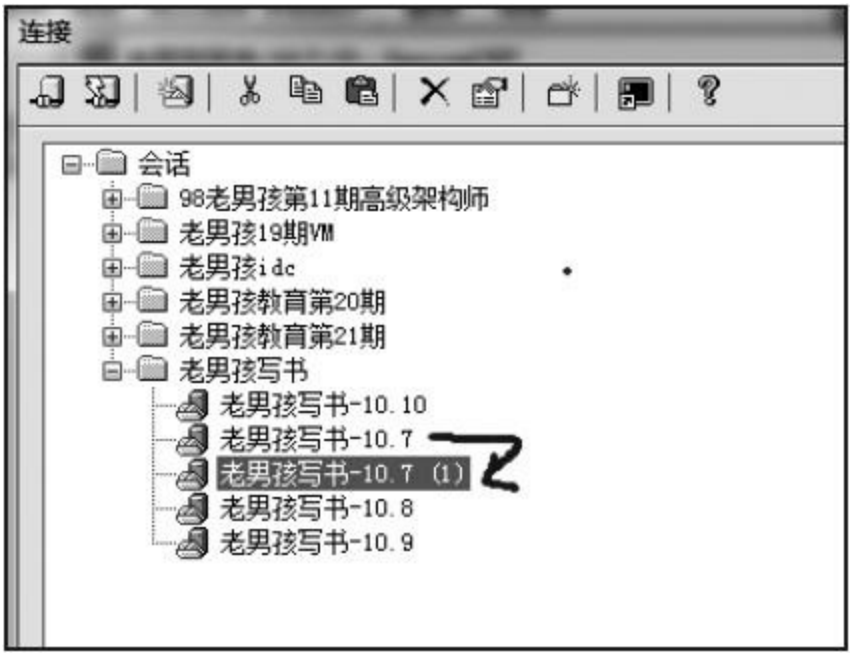


图3-23　复制配置好的标签

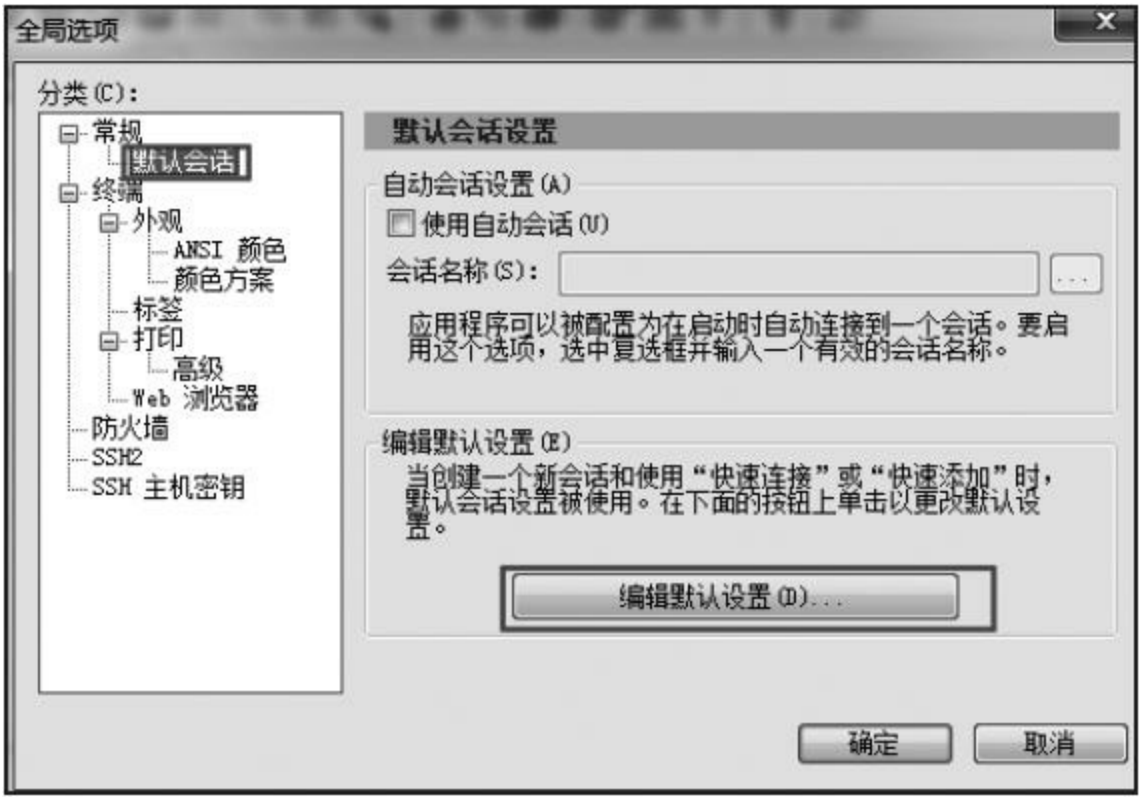


图3-24　全局调整标签配置永久生效

3.2.12　调整命令行颜色方案（目录和注释）

默认情况下命令行界面的目录和文件内容注释都是深蓝色的，看不清楚，因此，这个功能的调节就显得很有必要了。方法是依次选择“SecureCRT菜单”→“选项”→“全局选项”，然后在“全局选项”左侧单击“ANSI颜色”，右侧就出现相应的选项（如图3-25所示），现在将右侧深蓝的颜色调整为浅蓝。

图3-25　调整目录等的元素颜色显示

调整后可以发现CRT窗口目录的颜色变了，如图3-26所示。

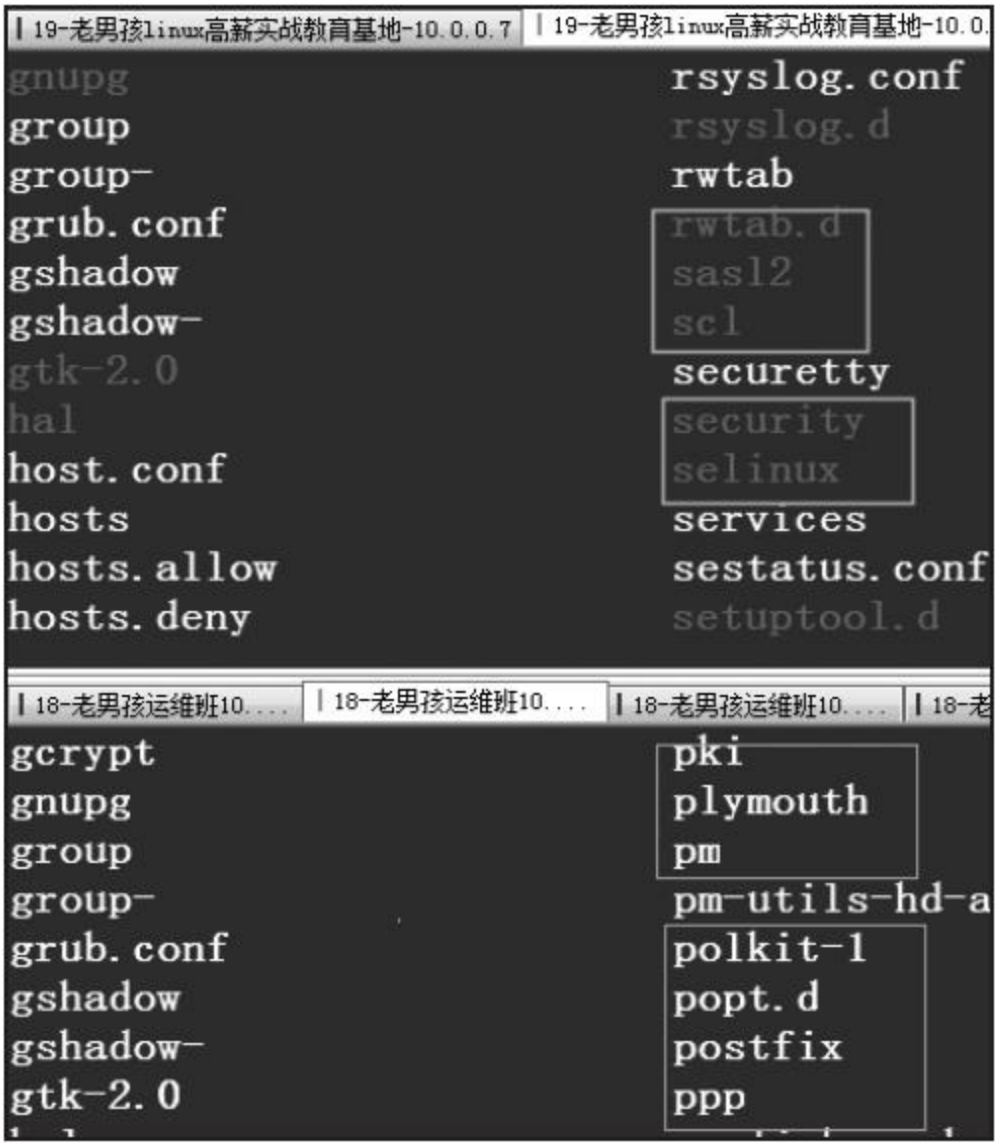


图3-26　颜色清晰效果

Shell脚本内中注释部分的颜色也有了相应的变化，如图3-27所示。

图3-27　编写脚本清晰效果

3.3 Linux系统应用管理

管理Linux系统之前，先来查看一下当前Linux系统的版本、内核等信息。命令如下：

```
[root@www ~]# cat /etc/redhat-release
CentOS release 6.6 （

Final）

      ←这是系统版本信息

[root@www ~]# uname -r
2.6.32-504.el6.x86_64          ←这是内核

kernel的版本号

[root@www ~]# uname -m
x86_64                    ←这表示为

64位系统
```

3.3.1 添加普通用户账号

可使用如下命令添加一个普通用户账号，并为其设置口令：

```
[root@www ~]# useradd oldboy
[root@www ~]# passwd oldboy
Changing password for user oldboy.
New password:



BAD PASSWORD:


 it is too simplistic/systematic ←提示密码太简单了，但可以不理会



BAD PASSWORD:


 is too simple
Retype new password:



passwd:


 all authentication tokens updated successfully.
```

 提示：一般情况下，在企业生产环境中应尽量避免直接到root用户下操作，除非有超越普通用户权限的系统维护需求。

还可通过下面的命令一步到位地设置密码（其中，oldboy为用户名，密码为qq：31333741）。

```
echo "qq:

31333741"|passwd --stdin oldboy && history -c
```

尝试切换用户角色，命令如下：

```
[root@www ~]# su - oldboy    <==由

root管理员，切换到普通用户

oldboy
[oldboy@www ~]$ whoami       <==查看当前用户是什么


oldboy
[oldboy@www ~]$ su - root    <==切回到

root用户

Password:
```

说明：

1）超级用户root切换到普通用户下面，无需输入对应用户密码，这相当于“皇帝”去“大臣”家里。

2）普通用户切换到root或其他普通用户下，需要输入切换的对应用

户密码。

3）普通用户的权限比较小，只能进行基本的系统信息查看等操作，无法更改系统配置和管理服务。

4）$符号是普通用户的命令行提示符，#符号是超级管理员的提示符。示例如下：

```
[oldboy@www ~]$      ←普通用户

oldboy对应的提示符

[root@www ~]#      ←超级管理员

root对应的提示符
```

5）提示符@前面的字符代表当前用户（可用whoami查询），后面的为主机名（可用hostname查询），~所在的位置是窗口当前用户所在的路径。示例如下：

```
[oldboy@www ~]$      ←

oldboy为当前用户，

www为主机名，
```

~表示当前目录，即家目录。

6）Linux命令提示符由PS1环境变量控制。示例如下：

```
[root@www ~]# set|grep PS1      ←注意

PS1是大写的
```

这里的PS1='[\u@\h\W]\$'，可以通过全局变量配置/etc/profile文件调整PS1='[\u@\h\W\t]\$'。

想进一步了解切换用户的命令su，可以参看老男孩的博客“由su和su-的区别谈学习Linux运维方法”，网址为：http://oldboy.blog.51cto.com/2561410/1053606 。

3.3.2　基本的Linux命令操作示例

在了解Linux命令之前，先看看命令操作语法格式图，如图3-28所示。

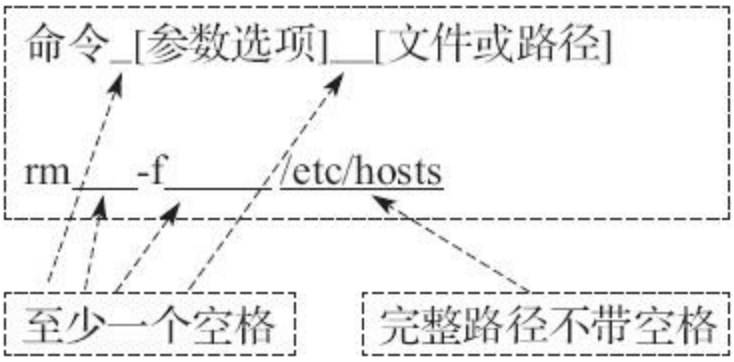


图3-28　Linux命令操作语法格式图

在Linux系统运维工作中，常用操作大多都是在命令行下实现的。

```
[root@oldboy ~]#                    ←就是在这样的提示符下敲命令。
```

Linux系统命令操作语法的格式：

```
命令
 [参数选项
] [文件或路径
```

```
]
rm   -f          /etc/hosts
```

说明：

·中括号（[]）表示可选，即命令的参数及路径文件是可选的。

·参数选项表示使用一个命令的不同功能。

例1：在根“/”下创建一个目录data。

可采用如下方法：

```
[root@www ~]# mkdir /data      ←此命令就是答案了。


[root@www ~]# ls -ld /data
drwxr-xr-x 2 root root 4096 8月


   3 14:


33 /data此命令是查看


data目录是否创建成功，


ls是


list列表的意思。


-ld是
```

ls的常用参数，是以长格式查看目录的属性

除了上面的方法外，还可以用下面的方法：

[root@www ~]# rm -r /data　　←删除前面建立的

/data目录。

rm：是否删除目录

"/data"?

y　　←输入

y表示确认删除，

y是

yes的缩写。

[root@www ~]# cd /　　←进入到根目录

/下，强调下

"/"在

linux里是根的意思，

即所有目录的顶点。

[root@www /]# mkdir data ←创建

data目录，注意不是

date，也不带斜线

"/"了，

因为已经进入了

"/"。

[root@www /]# ls -ld data ←查看创建

data目录本身

drwxr-xr-x 2 root root 4096 8月

3 14:

37 data


提示：Windows下的目录路径样式为d：\data\，而Linux下的路径就是/data/。

记忆方法：创建目录的命令可以理解为英文全称make directory，缩写后就是mkdir。

例1用到的Linux命令知识点如下。

·cd后跟相对路径或绝对路径，表示进入对应目录，cd/表示进入根目录下。记忆方法===>change directory，中文意思是变换目录。

·相对路径概念：不从“/”开始，而是从当前目录开始的路径。如data/test、mnt/oldboy。

·绝对路径概念：从“/”开始的路径，就叫绝对路径。
如/data/test、/mnt/oldboy。

·mkdir[-mp][目录名称]：表示建立目录，默认可以不加参数，如果实现特殊功能可加如下参数。

·-m：这个参数用来指定要创建目录的权限，但很少用，一般是创建目录后用chmod来处理。如mkdir-m 777/tmp/test。即建立一个权限为777的目录。

·-p：这个参数用来递归创建目录，如mkdir-p/data/test/，这是mkdir的常用参数。

例2：在/data目录下面建立一个文件，名字为oldboy.txt。

方法1：

```
[root@www /]# touch /data/oldboy.txt      ←在
/data目下创建
oldboy.txt。

[root@www /]# ls -l /data/oldboy.txt      ←查看创建的
oldboy.txt，注意，没有用到
                                          上文提到的
-d参数了，这是为什么呢？因为要
                                          查看的目标文件不是目录，所以不用
-d参数了

-rw-r--r-- 1 root root 0 8月
   3 14:
46 /data/oldboy.txt
```

方法2：

```
[root@www /]# cd /data
[root@www data]# touch oldboy.txt      ←如果同名文件存在，不会提示也不会覆盖，

                                       会更新

oldboy.txt文件的时间戳。

[root@www data]# ls -l oldboy.txt
-rw-r--r-- 1 root root 0 8月

   3 14:

47 oldboy.txt
                                       ←比前文的

oldboy.txt时间新
```

例2用到的Linux命令知识点如下：

·touch[文件名]：因为touch是“摸”的意思，所以该命令表示“摸”一下文件，其作用是如果文件不存在，就建立新文件，如果存在，就改变文件的访问时间信息。

例3：为oldboy.txt增加内容“I am studying linux.”。

方法1：常规编辑方法。

在data目录下执行“vi oldboy.txt”命令进入vi编辑器（默认为命令模

式），单击a或i进入编辑模式后敲入内容“I am studying linux.”，然后按键盘上的Esc键退出编辑模式（进入命令模式），最后敲“：wq命令”保存并退出。

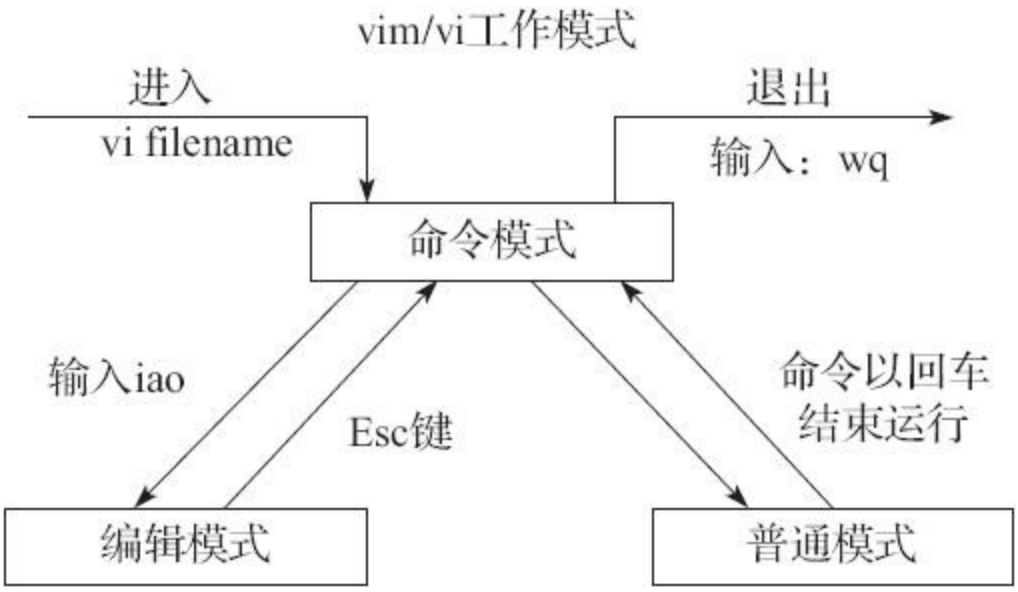


图3-29　vi/vim编辑器工作模式简图

图3-29为vi/vim编辑器的工作模式简图。

方法2：往文件尾部插入内容的常用命令是echo，注意，内容都是放到文件尾部的。

```
[root@oldboy ~]# echo 'I am studying linux.'>>/data/oldboy.txt←这个在企业里很常用：内容

echo命令使用单引号把引号内的内容通过

">>"输出重定向符号追加到

/data/oldboy.txt，注意：追加就是增加，不是覆盖。
```

```
[root@oldboy ~]# cat /data/oldboy.txt          ←检查追加的结果

I am studying linux.
```

例3用到的Linux命令知识点如下。

·vi：类似Windows记事本的Linux下的常用编辑器，如果想功能更丰富可以用vim替代vi，vi/vim的内部有很多命令技术点要掌握，例如方法1提到的i就是insert，进入插入编辑模式状态的意思，Esc键是切换到命令模式，wq就是写入后退出编辑器的意思（w是write，q为quit）。命令模式下敲冒号“：”，后面才可以接命令参数。

·echo：一个打印输出内容的常用命令，配合“>”或“>>”可以清除文件内容及追加内容到文件尾部，这个命令须掌握。“>”意思为重定向，会清除文件里所有旧的数据，然后增加echo命令后的内容，“>>”为内容追加，只在文件尾部追加需要的内容。

例4：把oldboy.txt文件拷贝到/tmp下。

方法如下：

```
[root@www data]# cp /data/oldboy.txt /tmp/      ←这个就是拷贝的命令了

[root@www data]# ls -l /tmp/oldboy.txt
```

```
-rw-r--r-- 1 root root 21 8月

   3 15:

07 /tmp/oldboy.txt
[root@www data]# cat /tmp/oldboy.txt
I am studying linux.提示：有的读者加参数

-a ，

-p等也可以。拷贝目录并保持属性。更多信息可以执行

man cp进行查看
```

例4用到的Linux命令知识点如下。

在cp[-a i f p r u][源文件][目标文件]的第一个中括号中可加入如下参数。

·-a：相当于-dpr。

·-d：若源文件为链接文件（link file），则复制链接文件属性而非档案本身。

·-p：连同档案的属性一起复制过去，而非使用默认属性。

·-r：递归，用于复制目录。

·-f：强制，若目标档案已经存在且无法开启，则移除后再尝试。

·-i：若目标文件已经存在，在覆盖时会先询问。

·-u：若目标文件存在，则目标文件比源文件旧时才复制。

 提示：如果要复制的源文件有多个，那么目的文件在最后，且必须是目录。cp的重要参数是-a、-p、-r。

例5：把/data目录移动到/root下。

方法如下：

```
[root@www data]# pwd          ←打印当前路径


/data
[root@www data]# cd /          ←切换到根目录下


[root@www /]# mv data /root/      ←移动


data目录到


/root下，移动就是剪切的意思


[root@www /]# ls -ld /root/data/
drwxr-xr-x 2 root root 4096 8月


   3 14:
```

```
46 /root/data/
```

例5用到的Linux命令知识点如下。

在mv[-i f u][源目录/源文件][目的目录/目的文件]的第一个中括号中可加入如下参数。

·-i：如果目的文件已存在，询问是否覆盖。

·-f：强制执行，不会询问。

·-u：若目的文件存在，则比源文件新时才会移动。

 提示：如果要移动的源文件有多个，那么目的文件放在最后，且必须是目录。

例6：进入root目录下的data目录，删除oldboy.txt。

方法如下：

```
[root@www /]# cd /root/data/
[root@www data]# rm -f oldboy.txt      ←删除文件，使用


-f参数，快速删除，不提示是否删除


[root@www data]# ls
```

删除命令rm相当于remove（移除）的缩写，删除文件一般用“rm oldboy.txt”，此法会提示你是否确定删除，推荐初学者使用。用“rm-f oldboy.txt”删除很快，但是不提示用户进行确认，直接删除，比较危险，容易出错，不推荐初学者使用。

另外，对于文件的删除禁止使用“rm-fr文件名”，这种是杀鸡用牛刀的做法，必须禁止，最多用“rm-f文件”。rm-fr一般用来强制删除目录，非常危险，太多高手发生过严重的错误。


提示：初学者尽量不要用rm命令，如果必须要删除，可以用mv命令替代rm，也就是把文件移动到/tmp下，然后等/tmp分区将要满时一次性删除，使用mv移动文件到/tmp的方法是学习Windows下回收站的功能，减少犯错的机率。这种把/tmp目录当作回收站使用的做法，对初学者来说是一个不错的选择。

前面以问答的形式带大家学习了mkdir、cd、touch、cp、mv、rm等Linux基本命令，捎带学习了pwd、cat、ls、echo等基础命令，怎么样？Linux的命令是不是很简单！更多命令的应用请读者参考其他书籍，或者老男孩即将出版的命令基础书籍或运维班课程。

3.4　安装Linux系统后调优及安全设置

3.4.1　关闭SELinux功能

SELinux（Security-Enhanced Linux）是美国国家安全局（NSA）对于强制访问控制的实现，这个功能让系统管理员又爱又恨，这里我们还是把它给关闭了吧，至于安全问题，后面通过其他手段来解决，这也是大多数生产环境的做法，如果非要开启也是可以的。关闭方式如下。

1）修改配置文件，使关闭SELinux永久生效：

```
[root@www ~]# sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
<==修改配置文件可使配置永久生效，但必须要重启系统，此步是


sed快速修改方法，也可以通过


vi编辑修改此文件。



[root@www ~]# grep SELINUX=disabled /etc/selinux/config
SELINUX=disabled               <==检查替换结果为


disabled就表示编辑成功了
```

2）临时关闭SELinux，可在命令行执行如下命令：

```
[root@www ~]# setenforce
usage:

  setenforce [ Enforcing | Permissive | 1 | 0 ]←数字

0表示

Permissive，即给出警告提示，但不会阻止操作，相当于

disabled。

数字

1表示

Enforcing，即表示

SELinux为开启状态。

[root@@www ~]# setenforce 0     <==临时将

SELinux调成

Permissive状态。

[root@@www ~]# getenforce     <==查看

SELinux当前状态
```

```
Permissive
```

命令说明如下。

·setenforce：用于命令行管理SELinux的级别，后面的数字表示设置对应的级别。

·getenforce：查看SELinux当前的级别状态。

 提示：修改配置SELinux后，要想使其生效，必须要重启系统。因此，可配合使用setenforce 0这个临时使其关闭的命令，这样在重启前后都可以使SELinux关闭生效，也就是说无须立刻重启服务器了，在生产场景下Linux机器是不能随意重启的。

3.4.2　设定运行级别为3（文本模式）

设定运行级别（runlevel）为3，即表示使用文本命令行模式管理Linux，如果你是按照本书前面章节所述的那样完成的系统安装，则无需设置，检查一下即可。检查命令如下：

```
[root@www ~]# grep 3:

initdefault /etc/inittab
id:

3:

initdefault:

            <==这里的

3就是

Linux默认的运行级别，如果有需求可以将其

                        修改为其他级别。工作中常用

3级别，即文本模式。

[root@www ~]# runlevel      <==查看当前系统运行级别
```

```
N 3
[root@www ~]# init 5      <==切换运行级别为


5，只有在安装了桌面程序时才可以执行


                        startx命令切换
```

命令说明如下。

·runlevel：查看当前系统运行级别。

·init：切换运行级别，后面接对应级别的数字，例如：init 6就是重启Linux服务器。

课外作业：

·掌握Linux系统的7种运行级别。

·掌握Linux系统从开机到登录之前的启动流程。

3.4.3 精简开机系统自启动

与Windows系统一样，在Linux服务器运行的过程中，也会有很多无用的软件服务默认运行，这些服务占用了很多系统资源，而且也带来了安全隐患，因此要关闭。那么，企业生产场景的Linux主机到底需要保留哪些开机自启动的服务呢？

1.重要的开机自启动服务

根据老男孩多年的经验，企业环境新装Linux系统之后有必要保留的开机自启动服务有5个，具体如下。

·sshd：远程连接Linux服务器时需要用到这个服务程序，所以必须要开启，否则Linux服务器就无法提供远程连接服务了。

·rsyslog：日志相关软件，这是操作系统提供的一种机制，系统的守护程序通常会使用rsyslog程序将各种信息写到各个系统日志文件中，在CentOS 6以前此服务的名字为syslog。

·network：系统启动时，若想激活/关闭各个网络接口，则应（必须）考虑开启此服务。

·crond：该服务用于周期性地执行系统及用户配置的任务计划。有

要周期性执行的任务时，就要开启，此服务几乎是生产场景必须要用的一个软件。

·sysstat：sysstat是一个软件包，包含监测系统性能及效率的一组工具，这些工具对于我们收集系统性能数据很有帮助，比如CPU使用率、硬盘和网络吞吐数据等，对这些数据的收集和分析，有利于判断系统运行是否正常，所以它是提高系统运行效率、安全运行服务器的得力助手。

sysstat软件包集成的主要工具为：

·iostat工具提供CPU使用率及硬盘吞吐效率的数据。

·mpstat工具提供与单个或多个处理器相关的数据。

·sar工具负责收集、报告并存储系统活跃的信息。


提示：上述5个服务是安装完系统后建议保留的开机自启动服务，也几乎是一切生产服务器必须保留的开机自启动服务。将来还可以根据服务器的业务使用场景调整相应的自启动服务。

2.设置开机自启动服务的常见方法

1）执行命令，然后手动选择处理的方法

方法1：执行ntsysv命令，然后在弹出的窗口中进行设置。

方法2：执行setup命令→system service，然后在弹出的窗口中进行设置。

 提示：初学者可以选择上述两种方法中的一种来完成设置。

2）通过一行命令或Shell脚本进行设置

若你是有一定Linux经验的读者，可以进一步了解通过命令或脚本来一键完成设置的思路。

在快速设置前，先来查看默认情况下Linux系统开启的服务有哪些，由于我们工作在文本模式3级别，因此只需要查找3级别上开启的服务即可。查看命令如下：

```
[root@www ~]# LANG=en#先调整成英文字符集，以方便下面命令过滤中文字符串


[root@www ~]# chkconfig --list|grep 3:


on
abrt-ccpp      0:


off  1:


off  2:


off  3:


on  4:
```

```
off  5:


on  6:


off
abrtd          0:


off  1:


off  2:


off  3:


on  4:


off  5:


on  6:


off
acpid          0:


off  1:


off  2:


on   3:


on  4:
```

```
on   5:

on  6:

off
atd             0:

off  1:

off  2:

off  3:

on  4:

on   5:

on  6:

off
auditd          0:

off  1:

off  2:

on   3:

on  4:

on   5:
```

```
on  6:


off
blk-availability      0:


off  1:


on   2:


on  3:


on   4:


on  5:


on    6:


off
cpuspeed       0:


off  1:


on   2:


on   3:


on  4:


on   5:


on  6:
```

off
crond   0:

off 1:

off 2:

on   3:

on 4:

on   5:

on 6:

off <==这是要保留的

haldaemon   0:

off 1:

off 2:

off 3:

on 4:

on   5:

on 6:

```
off
ip6tables      0:


off  1:


off  2:


on   3:


on  4:


on   5:


on  6:


off
iptables       0:


off  1:


off  2:


on   3:


on  4:


on   5:


on  6:
```

```
off
irqbalance     0:


off  1:


off  2:


off  3:


on  4:


on   5:


on  6:


off
kdump          0:


off  1:


off  2:


off  3:


on  4:


on   5:


on  6:


off
lvm2-monitor   0:
```

```
off  1:


on   2:


on   3:


on  4:


on   5:


on  6:


off
mdmonitor      0:


off  1:


off  2:


on   3:


on  4:


on   5:


on  6:


off
messagebus     0:


off  1:
```

```
off  2:


on   3:


on  4:


on   5:


on  6:


off
netfs           0:


off  1:


off  2:


off  3:


on  4:


on   5:


on  6:


off
network         0:


off  1:
```

```
off  2:

on   3:

on  4:

on   5:

on  6:

off <==这是要保留的


postfix        0:

off  1:

off  2:

on   3:

on  4:

on   5:

on  6:

off
rsyslog        0:

off  1:
```

```
off  2:

on   3:

on  4:

on   5:

on  6:

off <==这是要保留的


sshd            0:

off  1:

off  2:

on   3:

on  4:

on   5:

on  6:

off <==这是要保留的


sysstat         0:
```

```
off  1:

on   2:

on   3:

on  4:

on   5:

on  6:

off <==这是要保留的

udev-post      0:

off  1:

on   2:

on   3:

on  4:

on   5:

on  6:

off
```

提示：可以看到，默认情况下开启了很多服务，我们需要保留开启的所有服务也包含在其中

了解了系统在3级别上开启的服务后，就可以通过命令快速实现配置了，下面就正式介绍

第一种快速处理方法：先全关闭，再开启需要保留的。

操作思路：先将3级别文本模式下默认开启的服务都关闭，然后开启需要开启的服务。

操作命令如下：

```
LANG=en
for oldboy in `chkconfig --list|grep 3:

on|awk '{print $1}'`;

do chkconfig --level 3 $oldboy off;

done
for oldboy in crond network rsyslog sshd sysstat ;

do chkconfig --level 3 $oldboy on;

done
chkconfig --list|grep 3:

on
```

操作过程如下：

```
[root@www ~]# LANG=en                #<==临时调整字符集为英文

[root@www ~]# for oldboy in `chkconfig --list|grep 3:

on|awk '{print $1}'`;

do chkconfig --level 3 $oldboy off;

done          #<==关掉所有开启的服务

[root@www ~]# for oldboy in crond network rsyslog sshd sysstat ;

do chkconfig --level 3 $oldboy on;

done
<==开启需要开启的服务

[root@www ~]# chkconfig --list|grep 3:

on <==查看设置结果

crond           0:

off   1:

off   2:

on    3:

on    4:

on    5:

on    6:

off
network         0:
```

```
off   1:

off   2:

on    3:

on    4:

on    5:

on    6:

off
rsyslog         0:

off   1:

off   2:

on    3:

on    4:

on    5:

on    6:

off
sshd            0:

off   1:

off   2:

on    3:

on    4:

on    5:

on    6:
```

```
off
sysstat         0:

off   1:

off   2:

off   3:

on    4:

off   5:

off   6:

off
```

第二种快速处理方法：一条命令搞定，Shell循环实现。

操作思路：默认情况下开机需要保留的服务都已经是开启状态了，因此，只需把3级别下

操作过程如下：

```
[root@www ~]# for oldboy in `chkconfig --list|grep "3:

on"|awk '{print $1}'|grep -vE "crond|network|sshd|rsyslog|sysstat"`;

do chkconfig $oldboy off;

done
[root@www ~]# chkconfig --list|grep 3:

on
crond           0:
```

```
off   1:

off   2:

on    3:

on    4:

on    5:

on    6:

off
network         0:

off   1:

off   2:

on    3:

on    4:

on    5:

on    6:

off
rsyslog         0:

off   1:

off   2:

on    3:

on    4:

on    5:

on    6:
```

```
off
sshd            0:

off   1:

off   2:

on    3:

on    4:

on    5:

on    6:

off
sysstat         0:

off   1:

off   2:

off   3:

on    4:

off   5:

off   6:

off
```

第三种快速处理方法：不要Shell循环语句也一条命令搞定。

操作思路：默认情况下开机需要保留的服务都已经是开启状态了，因此，只需把3级别

操作命令如下：

```
chkconfig --list|grep 3:

on|grep -vE "crond|sshd|network|rsyslog|sysstat" |awk '{print "chkconfig " $1 " off"}'|bash
```

操作过程为先拼接所有要操作的命令字符串，命令如下：

```
[root@www ~]# chkconfig --list|grep 3:

on|grep -vE "crond|sshd|network| rsyslog|sysstat" |awk '{print "chkconfig " $1 " off"}'
chkconfig abrt-ccpp off
chkconfig abrtd off
chkconfig acpid off
chkconfig atd off
chkconfig auditd off
chkconfig blk-availability off
chkconfig cpuspeed off
chkconfig haldaemon off
chkconfig htcacheclean off
chkconfig sysstat off
```

然后将拼接的所有要操作的命令字符串通过bash运行，如下：

```
[root@www ~]#chkconfig --list|grep 3:

on|grep -vE "crond|sshd|network| rsyslog|sysstat" |awk '{print "chkconfig " $1 " off"}'|bash
```

用下面的命令也可：

```
[root@www ~]# chkconfig --list|grep 3:
```

```
on|grep -vE "crond|sshd|network| rsyslog|sysstat" |awk '{print $1}'|sed -r 's# (
.*)
#chkconfig \1 off#g'|bash
```

提示：有很多朋友经常问老男孩，到底将哪些服务作为开机自启动服务合适？其实这个问题

3.4.4 关闭iptables防火墙

关闭防火墙的目的是让初学者学习更方便，将来学了iptables技术后可再统一开启。在企业环境中，一般只有配置外网IP的Linux服务器才需要开启防火墙，但即使有外网IP，高并发、高流量的业务服务器仍然不能开启防火墙，因为开启后会有较大性能损失，导致网站访问速度很慢，这种情况下只能在前端加更好的硬件防火墙了。

关闭防火墙的具体操作过程如下：

```
[root@www ~]# /etc/init.d/iptables stop
iptables:


 Setting chains to policy ACCEPT:


 filter          [  OK  ]
iptables:


 Flushing firewall rules:


                          [  OK  ]
iptables:


 Unloading modules:


                                [  OK  ]
[root@www ~]# /etc/init.d/iptables stop#←重复执行确认已关闭
```

```
[root@www ~]# chkconfig iptables off#←关闭开机自启动命令，前面已经关闭这里就无需执行


[root@www ~]# chkconfig --list|grep ipt
iptables        0:

off   1:

off   2:

off   3:

off   4:

off   5:

off   6:

off
```

3.4.5 Linux系统安全最小原则说明

最小化原则 对Linux系统安全来说极其重要，即多一事不如少一事。具体包括如下几个方面：

·安装Linux系统最小化，即选包最小化，yum安装软件包也要最小化，无用的包不装。

·开机自启动服务最小化，即无用的服务不开启。

·操作命令最小化。例如：能用“rm-f test.txt”就不用“rm-fr test.txt”。

·登录Linux用户最小化。平时没有特殊需求不登录root，用普通用户登录即可。

·普通用户授权权限最小化，即只给用户必需的管理系统的命令。

·Linux系统文件及目录的权限设置最小化，禁止随意创建、更改、删除文件。

3.4.6　更改SSH服务器端远程登录的配置

Windows服务器的默认远程管理端口是3389，管理员用户是administrator，普通用户是guest。Linux的管理用户是root，普通用户默认有很多个，远程连接默认端口22。这些通常有IT经验的人都知道。那么黑客是否也知道呢？他们当然知道，甚至比我们更清楚，所以，为了系统安全，必须隐藏或更改上述默认配置。更改配置的命令如下：

```
[root@www ~]#cp /etc/ssh/sshd_config /etc/ssh/sshd_config.ori
#→更改配置前进行备份，是系统管理员的一个良好的习惯。



[root@www ~]#vi /etc/ssh/sshd_config  #→编辑


sshd_config
####by oldboy#2011-11-24##
Port 52113
PermitRootLogin no
PermitEmptyPasswords no
UseDNS no
GSSAPIAuthentication no
####by oldboy#2011-11-24##
```

其中，sshd_config修改的相关参数说明见表3-4。

表3-4　修改的参数说明

参　　数	说　　明
Port	指定 sshd 守护进程监听的端口号，默认为 22。默认在本机的所有网络接口上监听，也可以通过 ListenAddress 指定只在某个特定的接口上监听。 端口范围：0 ~ 65535，不能与已有的服务器端口冲突。 一般建议改为比 1024 大的端口
PermitEmptyPasswords	是否允许密码为空的用户远程登录。默认为“no”
PermitRootLogin	是否允许 root 登录。可用值如下：“yes”（默认）表示允许；“no”表示禁止；“without-password”表示禁止使用密码认证登录；“forced-commands-only”表示只有在指定了 command 选项的情况下才允许使用公钥认证登录，同时其他认证方法全部被禁止，这个值常用于做远程备份之类的事情
UseDNS	指定 sshd 是否应该对远程主机名进行反向解析，以检查此主机名是否与其 IP 地址真实对应。默认值为“yes”。 建议改成“no”，否则可能会导致 SSH 连接很慢
GSSAPIAuthentication no	解决 Linux 之间使用 SSH 远程连接慢的问题

注：Linux下SSH远程连接服务慢的解决方案请见老男孩的博客：http://oldboy.blog.51cto.com/2561410/1300964 。

将以上信息更改后，保存退出。

执行如下命令重启sshd，使修改的配置生效：

```
[root@www ~]# /etc/init.d/sshd reload重新载入

 sshd:

                                                [确定

]或者


[root@www ~]# /etc/init.d/sshd restart
Stopping sshd:

                                              [  OK  ]
```

```
Starting sshd:                                             [  OK  ]
```

提示：reload为平滑重启，不影响正在SSH连接的其他用户，因此要好于restart。

此时远程连接Linux，就仅有oldboy用户可通过52113端口远程连接到系统中了（本地登录的用户不受影响）。

除了手工操作之外，也可以通过命令行批量修改对应的参数，命令集如下：

```
  echo "#--------sshConfig修改

ssh默认登录端口，禁止

root登录

----------#"
  \cp /etc/ssh/sshd_config /etc/ssh/sshd_config.`date +"%F%H%M%S"`
  sed -i 's%#Port 22%Port 52113%' /etc/ssh/sshd_config
  sed -i 's%#PermitRootLogin yes%PermitRootLogin no%' /etc/ssh/sshd_config
  sed -i 's%#PermitEmptyPasswords no%PermitEmptyPasswords no%' /etc/ssh/sshd_config
  sed -i 's%#UseDNS yes%UseDNS no%' /etc/ssh/sshd_config
  sed -i 's%GSSAPIAuthentication yes%GSSAPIAuthentication no%' /etc/ssh/sshd_config
  egrep "UseDNS|52113|RootLogin|EmptyPass|GSSAPIAuthentication" /etc/ssh/sshd_config
  /etc/init.d/sshd reload
```

还可以通过sed命令快速增加参数内容，如下：

```
[root@www ssh]# sed -ir '13 iPort 52113\nPermitRootLogin no\nPermitEmptyPasswords no
[root@www ssh]# sed -n '13,
```

```
17p' sshd_config
Port 52113
PermitRootLogin no
PermitEmptyPasswords no
UseDNS no
GSSAPIAuthentication no
```

注意，在重启sshd后，当前的CRT仍然可以连接服务器，直到退出重新登录为止。如下：

```
[root@www ~]# netstat -an|grep 10.0.0.100  <==ip是客户机的

IP。

tcp      0     52 10.0.0.7:

22       10.0.0.100:

52255           ESTABLISHED
tcp      0      0 10.0.0.7:

22       10.0.0.100:

53242           ESTABLISHED
```

退出CRT之后，登录会显示无法连接（如图3-30所示）。

此时要修改登录用户和登录端口才行，如图3-31所示。

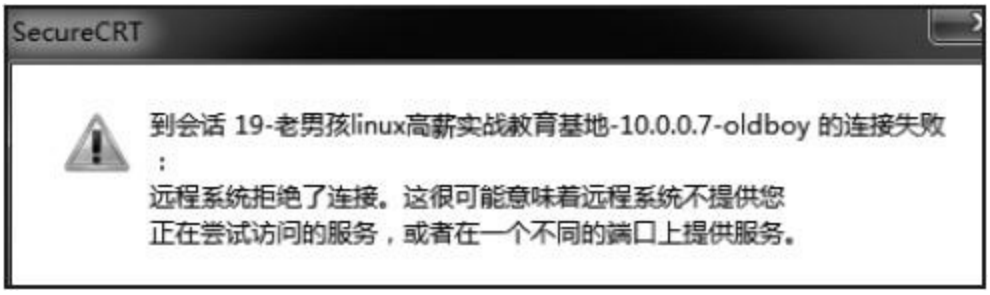


图3-30　新窗口无法远程连接SSH

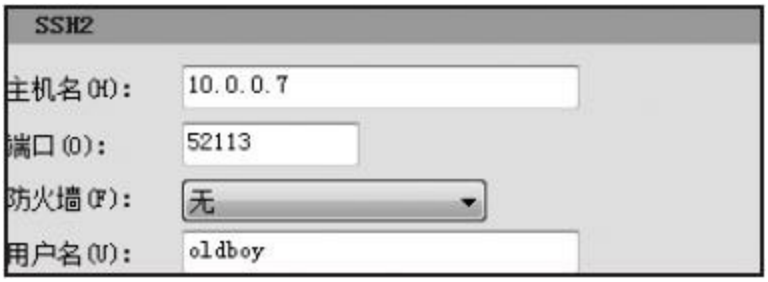


图3-31　修改登录端口和用户

配置完毕，重新连接后的结果如下：

```
[oldboy@www ~]$ whoami
oldboy
```

如果需要系统管理可以切换到root用户，切换命令如下：

```
[oldboy@www ~]$ su - root      #←注意，此处需要的是


root用户的密码



Password:



[root@www ~]# whoami
root
```

除了前面介绍的安全知识以外，还有更高级的SSH安全策略，具体如下。

1）更改SSH监听的IP，使其仅监听内网IP。命令如下：

```
[root@web01 ~]# sed -n '13,


20p' /etc/ssh/sshd_config
####by oldboy#2011-11-24##
Port 52113
PermitRootLogin no
PermitEmptyPasswords no
UseDNS no
GSSAPIAuthentication no
ListenAddress 10.0.0.7:


52113      #←企业仅指定监听本机内网


IP地址


####by oldboy#2011-11-24##
```

2）通过防火墙限制仅能使用内网IP连接此服务器。限制命令如下：

```
iptables -I INPUT -p tcp --dport 52113 -s 10.0.0.0/24 -j ACCEPT
#←默认规则为


DROP时的限制


SSH命令
```

3）通过拨号到VPN服务器，然后从局域网访问这些服务器，提升安全性。其实Linux主机安全就像我们居住的房子安全一样，其原理示意图如图3-32所示。

4）工作中的系统安全重点就是互联网TCP/IP连接、对外的Web服务器端口http 80和https 443的安全控制。

3.4.7　利用sudo控制用户对系统命令的使用权限

为了安全及管理的方便，可将需要root权限的普通用户加入sudo管理，这样用户就可以通过自己的普通账户登录，利用root的权限来管理系统了，当然也就不需要有root账号及密码了。

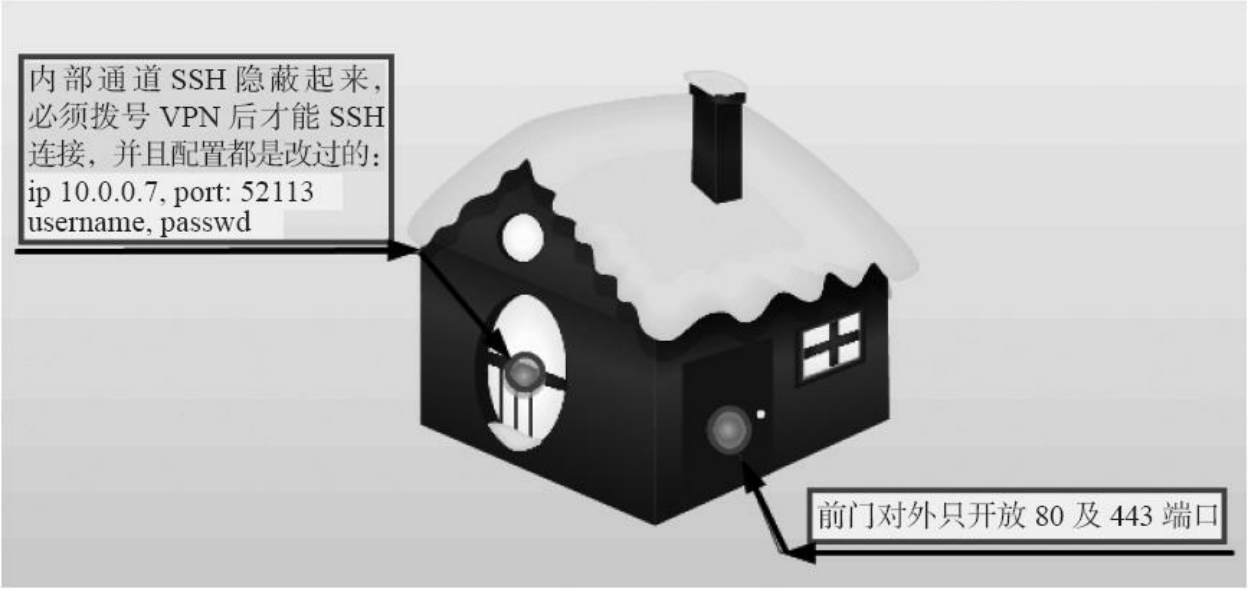


图3-32　Linux服务器安全形象示意图

执行如下visudo命令，即可打开sudo的配置文件进行编辑。

```
[root@wwwentos ~]#visudo  #→相当于直接编辑

/etc/sudoers，但用命令方式更安全，推荐
```

在/etc/sudoers文件的大约第98行下面，添加需要提升为root权限的普通用户名及对应权限，格式如下：

```
visudo或者

vi /etc/sudoers，在

98行下面加入，也可以在其他位置加入。


root    ALL=（

ALL）

                        ALL <==此行是

98行


oldboy  ALL=（

ALL）

                    /usr/sbin/useradd,

/usr/sbin/userdel
```

上述内容不同列对应的说明见表3-5。

表3-5　sudo提权配置说明

用户或组	机器 =（授权角色）	可以执行的命令
user	MACHINE=	COMMANDS
oldboy	ALL=(ALL)	/usr/sbin/useradd,/usr/sbin/userdel

提示：如果oldboy用户被授予上述权限，那么它可在所有的机器上以所有的角色运行useradd、userdel命令。

如果是针对用户组，则对应的授权命令如下：

```
%用户组

  机器

=（授权使用哪个角色的权限）

  /usr/sbin/useradd
```

通过sudo进行授权管理系统的目的：既能让运维人员干活，又不能让其威胁系统安全，其实就是前面讲的用户权限最小化原则，还可以审计用户使用sudo的提权操作命令。

为了管理方便，这里暂时给oldboy授权all权限，即可以管理整个系统。

详细操作过程为：输入visudo找到第98行，并在该行下面放入如下内容。

```
oldboy  ALL=（
```

```
ALL)

        NOPASSWD:

 ALL
#→这个配置结尾的

ALL表示

oldboy可拥有完全的系统管理权限，

NOPASSWD表示提权执行命令时不提示密码
```

配置完成后要进行检查，命令如下：

```
[root@www ~]# grep oldboy /etc/sudoers
oldboy  ALL=(

ALL)

        NOPASSWD:

 ALL
```

也可以使用如下快速操作命令增加sudo授权，仅限于批量管理的情况：

```
\cp /etc/sudoers /etc/sudoers.ori
echo "oldboy  ALL=(


ALL)


 NOPASSWD:


 ALL " >>/etc/sudoers
tail -1 /etc/sudoers
visudo -c #→直接追加内容没有语法检查，因此要单独执行语法检查命令
```

将以上信息更改后，保存退出。

此时再以oldboy用户登录系统时，就可以通过执行类似sudo ls-l/root（sudo后面加正常命令）的命令以root用户的权限管理系统了，如下：

```
[oldboy@www ~]$ ls /root
ls:


 cannot open directory /root:


 Permission denied
#→可以看到，


oldboy用户是无法直接访问访问


/root家目录的


[oldboy@www ~]$ sudo ls /root
```

```
anaconda-ks.cfg  install.log  install.log.syslog
#→通过


sudo命令，使得


oldboy用户具备了访问


/root目录的权限
```

说明：

·通过sudo授权管理后，所有用户执行授权的特殊权限格式为“sudo 命令”。

·如果需要切换到root执行相关操作，可以通过“sudo su-”命令，注意，此命令提示的密码为当前用户的密码，而不是root的密码。

·执行“sudo-l”命令可以查看当前用户被授予的sudo权限集合。如下：

```
[oldboy@oldboy ~]$ sudo -l
[sudo] password for oldboy:


Matching Defaults entries for oldboy on this host:


User oldboy may run the following commands on this host:
```

```
    (
ALL)
 /usr/sbin/useradd,
  (
ALL)
 /usr/sbin/userdel
```

·对于Linux系统bash的内置命令，一般无法进行sudo授权，例如cd命令。

sudo授权与su切换的原理示意图如图3-33所示。

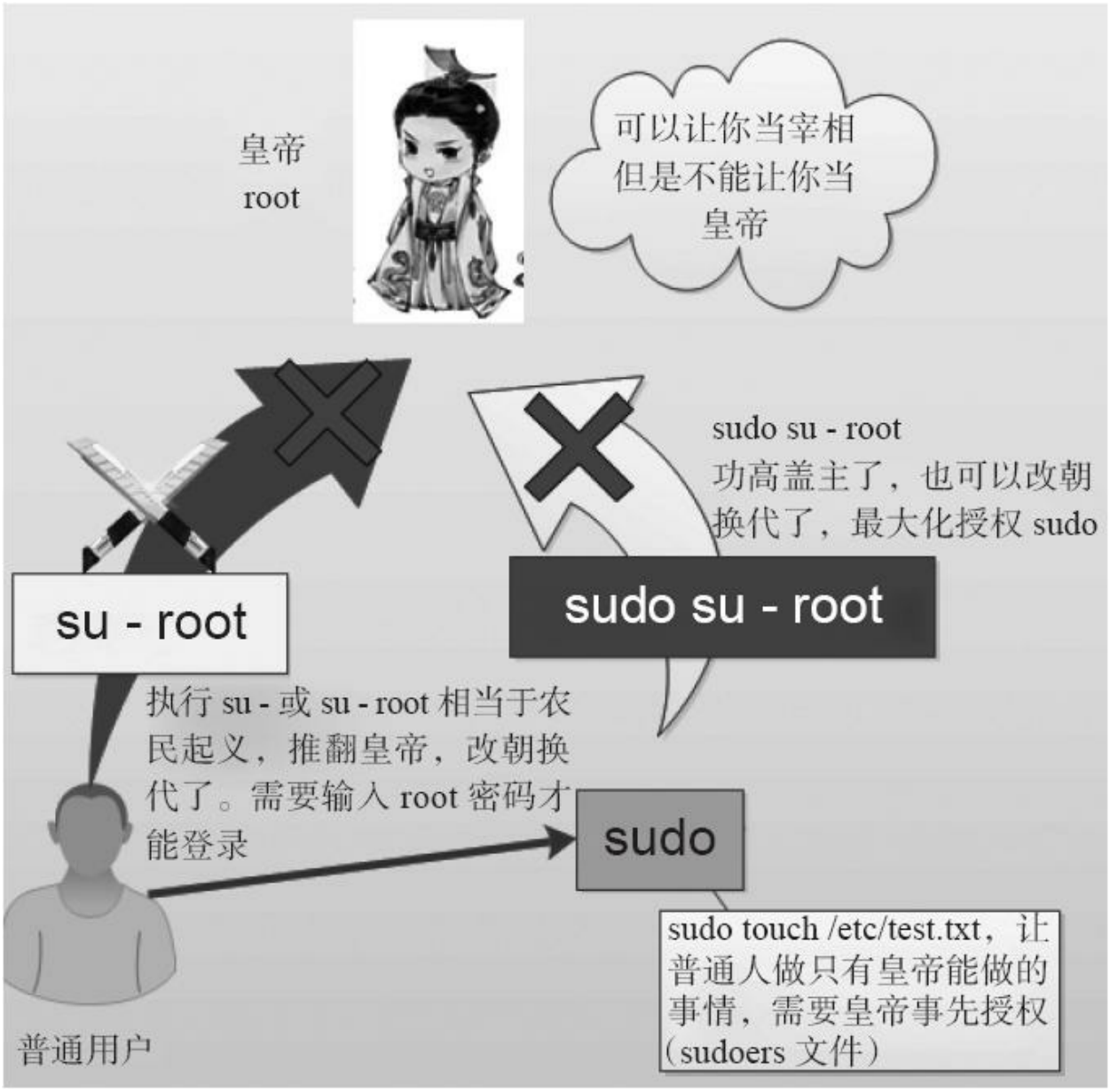


图3-33　su与sudo用户角色切换原理图

在生产环境中，通常会禁止root远程登录，不过，会为每个运维人员建立一个普通账号，然后根据运维人员的需求，通过sudo控制登录系统的权限，事实证明这是一个不错的权限管理方式。当然，在笔者的生产环境中还使用了ldap统一认证登录及授权管理的方式。即只要有一个

账号和密码，全公司多个机房系统内通行无阻（系统登录、SVN、VPN等），有关这部分内容的讲解，在老男孩教学的高级课程中会有涉及。在此，大家了解一下即可。

普通用户的环境变量问题：在早期的Centos 5系统中，普通用户执行系统管理相关命令会遭遇到环境变量问题，导致找不到执行的命令（CentOS 6以后的系统已经不存在这个问题了）。

解决办法：先对比root和oldboy用户下默认的PATH环境变量，命令如下。

```
root@wwwentos ~# echo $PATH
/application/jdk/bin:


/application/jdk/jre/bin:


/usr/kerberos/sbin:


/usr/kerberos/bin:


/usr/local/sbin:


/usr/local/bin:


/sbin:


/bin:
```

```
/usr/sbin:

/usr/bin:

/bin:

/root/bin
[oldboy@centos oldboy]$ echo $PATH
/application/jdk/bin:

/application/jdk/jre/bin:

/usr/kerberos/bin:

/usr/local/bin:

/bin:

/usr/bin:

/bin:

/home/oldboy/bin:

/home/oldboy/bin
```

经过对比可发现普通用户上几个关键的环境变量（/usr/local/sbin：/sbin：/usr/sbin：）是导致找不到执行命令的原因（除非带全路径执行）。可通过如下命令来解决此问题：

[oldboy@centos ~]$ vi ~/.bash_profile   #→编辑

~/.bash_profile环境变量文件，把：

/usr/local/sbin：

/sbin：

/usr/sbin添加到

PATH环境变量里，注意，每个路径之间用冒号分隔。

PATH=$PATH：

$HOME/bin：

/home/oldboy/scripts：

/usr/local/sbin：

/sbin：

/usr/sbin
#→加粗部分：

/usr/local/sbin：

/sbin：

/usr/sbin即为添加的内容

```
export PATH
[oldboy@centos ~]$ source ~/.bash_profile #→使添加的内容生效



[oldboy@@centos ~ $ echo $PATH
/application/jdk/bin:


/application/jdk/jre/bin:


/usr/kerberos/bin:


/usr/local/bin:


/bin:


/usr/bin:


/bin:


/home/oldboy/bin:


/home/oldboy/bin:


/home/oldboy/scripts:


/home/oldboy/bin:


/home/oldboy/scripts:


/home/oldboy/bin:
```

```
/home/oldboy/scripts:

/usr/local/sbin:

/sbin:

/usr/sbin
```

---

sudo授权对于bash的内置命令处理，是一个难题，因为内置命令没有实体文件和路径，不过一般都有解决方法，例如可以使用sudo ls替代sudo cd，有的网友使用sudo bash后再使用内置命令，这是很危险的，不推荐。

3.4.8 Linux中文显示设置

此项优化为可选项，即调整Linux系统的字符集设置，那么，什么是字符集呢？简单地说，字符集就是一套文字符号及其编码。目前Linux下常用的字符集有如下几种。

·GBK：定长，双字节，不是国际标准，支持的系统不少，实际企业用的不多。

·UTF-8：非定长，1~4字节，广泛支持，MySQL也使用UTF-8，企业广泛使用。

可通过快捷的命令方式在/etc/sysconfig/i18n中添加如下内容，使其支持中文显示：

```
[root@www ~]# cat /etc/sysconfig/i18n
LANG="en_US.UTF-8"
SYSFONT="latarcyrheb-sun16"
[root@www ~]# cp /etc/sysconfig/i18n /etc/sysconfig/i18n.ori
[root@www ~]# echo ' LANG="zh_CN.UTF-8"'  >/etc/sysconfig/i18n
#→相当于用


vi /etc/sysconfig/i18n 添加


LANG="zh_CN.UTF-8"内容


[root@www ~]#  source /etc/sysconfig/i18n #→使上文修改生效
```

```
[root@www ~]# echo $LANG
zh_CN.UTF-8
```

提示：

·注意“zh_CN.UTF-8”的大小写字母。

·这个中文显示配置要与你自己的SSH客户端的配置一致。默认情况下就是老男孩演示的配置。

·调整SSH客户端CRT的字符集，使其与Linux服务器端一致（UTF-8），如图3-34所示。

图3-34　客户端CRT字符集设置

查看字符集设置的结果，如下：

```
[root@www ~]# cat /etc/sysconfig/i18n
LANG="zh_CN.UTF-8"
```

登录Linux系统查看中文字符是否正常显示的步骤如下：

1）将服务器端字符集（/etc/sysconfig/i18n）改为LANG="zh_CN.UTF-8"。

2）将客户端字符集（CRT）调整为UTF-8。

3）命令行执行setup命令，看到原来中文乱码的窗口不乱了，正确显示中文字符了（如图3-35所示）。

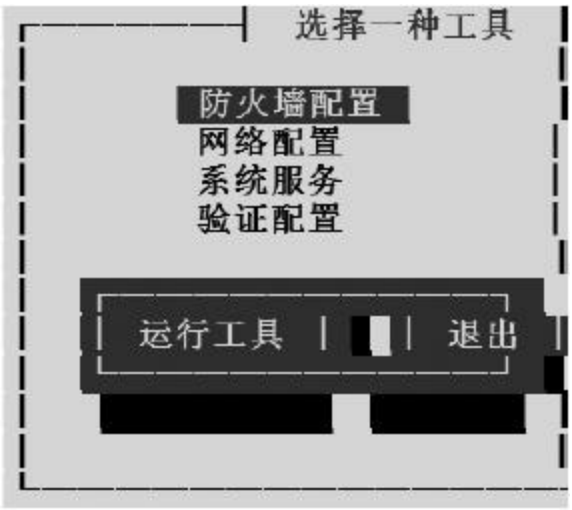


图3-35　客户端setup命令的窗口

3.4.9　设置Linux服务器时间同步

先看一下Windows系统的时间同步设置，如图3-36所示。由于计算机有了该设置，所以，你会发现计算机时间比手表的时间更准确。

Linux系统的时间同步服务为ntp服务。

我们可以手动同步互联网时间到本地Linux主机，命令如下：

```
[root@www ~]# /usr/sbin/ntpdate time.nist.gov
 4 Feb 14:


21:


30 ntpdate[19433]:


 adjust time server 24.56.178.140 offset -0.114353 sec←如果时间服务器连不上，可以去网上搜


/usr/sbin/ntpdate ntp.sjtu.edu.cn
[root@www ~]# which ntpdate
/usr/sbin/ntpdate <==注意这个路径，


Centos5为


/sbin
```

图3-36　Windows时间同步设置

利用定时任务crond把上述的命令每5分钟自动执行一次，命令如下：

```
[root@www ~]# echo '#time sync by oldboy at 2010-2-1' >>/var/spool/cron/root
[root@www ~]# echo '*/5 * * * * /usr/sbin/ntpdate time.nist.gov >/dev/null 2>&1' >>/
[root@www ~]# crontab -l
#time sync by oldboy at 2010-2-1
*/5 * * * * /usr/sbin/ntpdate time.nist.gov >/dev/null 2>&1
#→这个命令其实就是写一个定时任务，相当于执行


crontab -e然后加入内容：
```

```
*/5 * * * * /usr/sbin/ntpdate time.windows.com >/dev/null 2>&1保存退出
```

提示：在机器数量少时，可利用以上定时任务进行互联网时间同步，如果机器数量大，那么最好是在网内部署一个时间同步服务器ntp server，然后让自己的网内服务器的时间都与ntp server同步就可以了。

当前，小规模时间同步架构的示意图如图3-37所示，即网络内部不搭建时间服务器了，所有内部服务器都同步外部互联网已有的时间服务器。

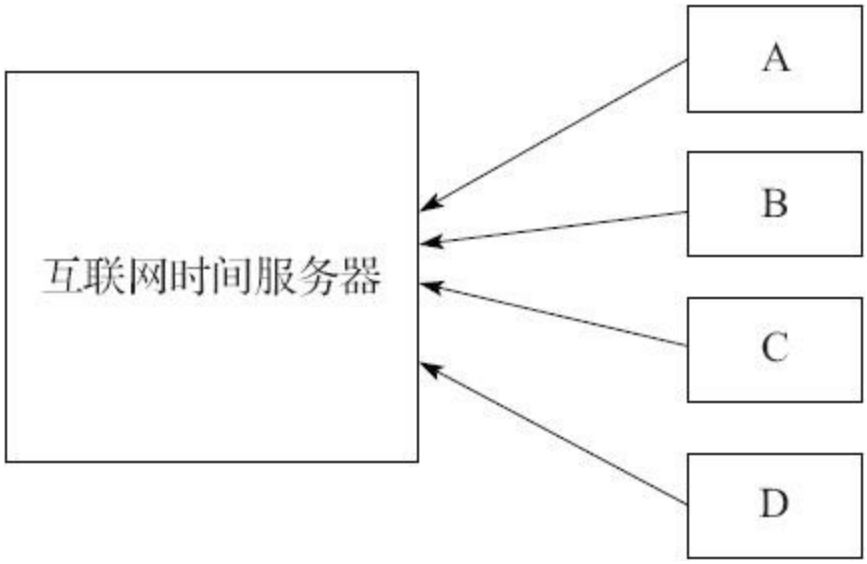


图3-37　小规模集群时间同步架构

大规模集群时间同步架构的示意图如图3-38所示，其中ntpserver1和

ntpserver2互备，为内部的时间服务器，由这两台服务器与外部时间服务器时间同步，然后内部的所有服务器都与ntpserver1和ntpserver2进行时间同步。

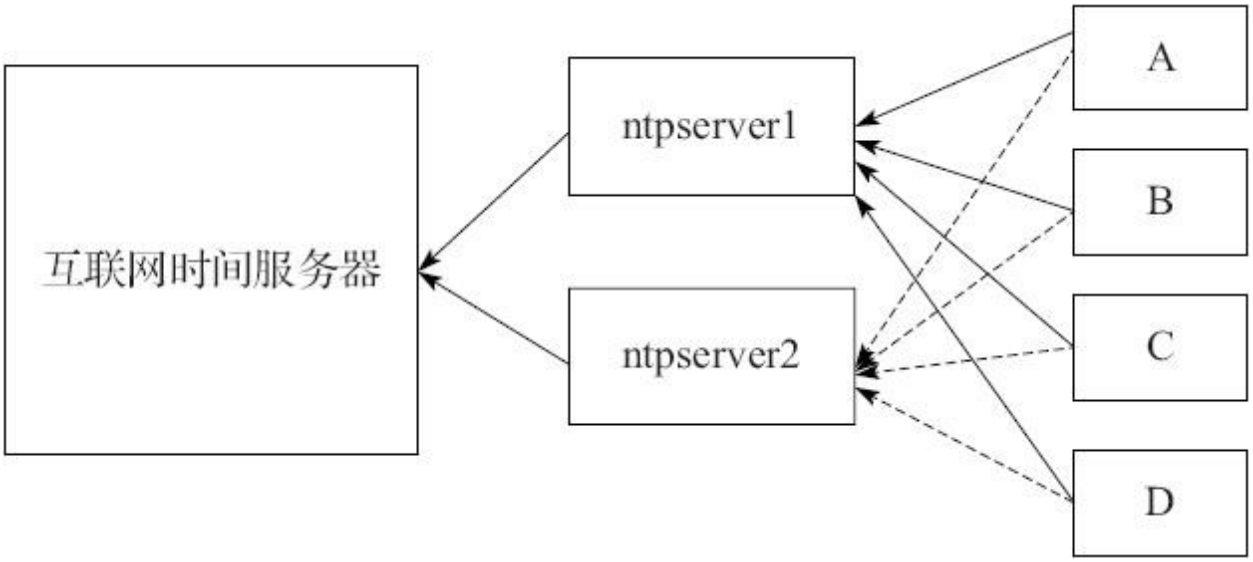


图3-38　大规模时间同步架构

有关ntp服务，此处不再深入讲解了。大家可以自行查阅相关文章。

3.4.10　历史记录数及登录超时环境变量设置

1.设置闲置账号超时时间

设置闲置账号超时时间的示例命令如下，注意此处的配置仅临时生效。

```
[root@www ~]# export TMOUT=10
[root@www ~]#
timed out waiting for input:


 auto-logout     #←


10秒提示超时
```

2.设置Linux的命令行历史记录数

设置Linux命令行的历史记录数示例命令如下，注意此处的配置仅临时生效。

```
[root@www ~]# export HISTSIZE=5
[root@www ~]# history
   42  whoami
   43  TMOUNT=10                    #连接的超时时间


   44  TMOUT=10
   45  HISTSIZE=5
   46  history
```

设定用户的命令行历史记录文件（~/.bash_history）记录指定命令数量的示例命令如下，注意此处的配置仅临时生效。

```
[root@www ~]# export HISTFILESIZE=5
[root@www ~]# cat ~/.bash_history
ifdown eth0
ls
ls
rz
ls                                  #←可以看到文件里记录命令的数量也变少了
```

把上述命令放入配置文件，使其可以永久生效，命令如下。

```
[root@www ~]# echo 'export TMOUT=300' >>/etc/profile
[root@www ~]# echo 'export HISTSIZE=5' >>/etc/profile
[root@www ~]# echo 'export HISTFILESIZE=5' >>/etc/profile
[root@www ~]# tail -3 /etc/profile
export TMOUT=300
export HISTSIZE=5
export HISTFILESIZE=5
[root@www ~]# source /etc/profile #←使得配置文件生效
```

在上述命令中，涉及的系统控制变量说明如下。

·TMOUNT=10：连接的超时时间控制变量。

·HISTSIZE=5：命令行的历史记录数量变量。

·HISTFILESIZE=10：历史记录文件的命令数量变量（~/.bash_history）。

 提示：实际工作中类似的变量还有不少，大家可以根据企业工作需求选择使用，最好执行前加上export命令。更多Linux系统的环境变量，大家可以执行man bash查询。

3.4.11　调整Linux系统文件描述符数量

文件描述符是由无符号整数表示的句柄，进程使用它来标识打开的文件。文件描述符与包括相关信息（如文件的打开模式、文件的位置类型、文件的初始类型等）的文件对象相关联，这些信息被称作文件的上下文。文件描述符的有效范围是0到OPEN_MAX。

对于内核而言，所有打开的文件都是通过文件描述符引用的。当打开一个现有文件或创建一个新文件时，内核向进程返回一个文件描述符。当读或写一个文件时，使用open或creat返回的文件描述符标识该文件，并将其作为参数传递给read或write，文件描述符定义的内容如下。

```
/* Standard file descriptors. */
#define STDIN_FILENO 0 /* Standard input. */
#define STDOUT_FILENO 1 /* Standard output. */
#define STDERR_FILENO 2 /* Standard error output. */
```

查看Linux服务器文件描述符设置的情况可以使用ulimit-n命令，文件描述符大小默认是1024。

```
[root@www ~]# ulimit -n
1024
```

对于高并发的业务Linux服务器来说，这个默认的设置值是不够的，需要调整。

调整方法1：

执行vim/etc/security/limits.conf命令，在文件结尾加上如下一行。

```
*               -       nofile          65535
```

或者直接执行如下一行命令追加上述内容到文件尾部：

```
[root@www ~]# echo '*   -   nofile    65535 ' >>/etc/security/limits.conf
[root@www ~]# tail -1 /etc/security/limits.conf
*               -       nofile          65535
```

配置完成后，需重新登录才可以生效，查看如下：

```
[root@www ~]# ulimit -n
65535
```

调整方法2：

直接把ulimit-SHn 65535命令加入/etc/rc.local，用以设置每次开机启动时配置生效，命令如下。

```
cat >>/etc/rc.local<<EOF
#-S use the 'soft' resource limit
#-H  use the 'hard' resource limit
#-n  the maximum number of open file descriptors
ulimit -HSn 65535
#-s the maximum stack size
ulimit -s 65535
EOF
```

3.4.12　Linux服务器内核参数优化

所谓Linux服务器内核参数优化[1]，主要是指在Linux系统中针对业务服务应用而进行的系统内核参数调整，优化并无一定的标准。下面以生产环境下Linux常见的内核优化为例进行讲解，仅供大家参考。

优化方法是执行vi/etc/sysctl.conf命令到文件结尾，然后拷贝如下内容并保存。

```
net.ipv4.tcp_fin_timeout = 2
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_keepalive_time = 600
net.ipv4.ip_local_port_range = 4000    65000
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.route.gc_timeout = 100
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_synack_retries = 1
net.core.somaxconn = 16384
net.core.netdev_max_backlog = 16384
net.ipv4.tcp_max_orphans = 16384
#以下参数是对


iptables防火墙的优化，防火墙不开会提示，可以忽略不理



net.nf_conntrack_max = 25000000
net.netfilter.nf_conntrack_max = 25000000
net.netfilter.nf_conntrack_tcp_timeout_established = 180
net.netfilter.nf_conntrack_tcp_timeout_time_wait = 120
net.netfilter.nf_conntrack_tcp_timeout_close_wait = 60
net.netfilter.nf_conntrack_tcp_timeout_fin_wait = 120
```

将上面的内核参数值加入/etc/sysctl.conf文件中，然后执行如下命令

使之生效：

```
[root@www ~]# sysctl -p
net.ipv4.ip_forward = 0
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
kernel.sysrq = 0
kernel.core_uses_pid = 1
net.ipv4.tcp_syncookies = 1
error:


 "net.bridge.bridge-nf-call-ip6tables" is an unknown key
error:


 "net.bridge.bridge-nf-call-iptables" is an unknown key
error:


 "net.bridge.bridge-nf-call-arptables" is an unknown key
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.shmmax = 68719476736
kernel.shmall = 4294967296
net.ipv4.tcp_fin_timeout = 2
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_keepalive_time = 600
net.ipv4.ip_local_port_range = 4000    65000
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.route.gc_timeout = 100
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_synack_retries = 1
net.core.somaxconn = 16384
net.core.netdev_max_backlog = 16384
net.ipv4.tcp_max_orphans = 16384
net.nf_conntrack_max = 25000000
net.netfilter.nf_conntrack_max = 25000000
error:


 "net.netfilter.nf_conntrack_tcp_timeout_established" is an unknown key
error:


 "net.netfilter.nf_conntrack_tcp_timeout_time_wait" is an unknown key
error:
```

```
 "net.netfilter.nf_conntrack_tcp_timeout_close_wait" is an unknown key
error:


 "net.netfilter.nf_conntrack_tcp_timeout_fin_wait" is an unknown key
```

如果是在CentOS 6环境中，必须开启ip6tables服务才不会出现上面所示的报错，其实报错也可以暂时不理，这是针对防火墙的优化，而此时防火墙并没有开启，将来开启了就没问题了。

sysctl.conf内核文件中的参数含义见表3-6。

表3-6　sysctl.conf内核文件中常用参数含义

参　数	说　明
net.ipv4.tcp_fin_timeout	表示套接字由本端要求关闭，这个参数决定了它保持在 FIN-WAIT-2 状态的时间，默认值是 60 秒。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_fin_timeout 60
net.ipv4.tcp_tw_reuse	表示开启重用。允许将 TIME-WAIT sockets 重新用于新的 TCP 连接，默认值为 0，表示关闭。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_tw_reuse 0
net.ipv4.tcp_tw_recycle	表示开启 TCP 连接中 TIME-WAIT sockets 的快速回收。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_tw_recycle，默认为 0，表示关闭。 提示：reuse 和 recycle 这两个参数是为防止生产环境下 Web、Squid 等业务服务器 time_wait 网络状态数量过多设置的
net.ipv4.tcp_syncookies	表示开启 SYN Cookies 功能。当出现 SYN 等待队列溢出时，启用 Cookies 来处理，可防范少量 SYN 攻击，Cent OS 5 系列默认值为 1，表示开启，因此这个参数也可以不添加。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_syncookies，默认为 1
net.ipv4.tcp_keepalive_time	表示当 keepalive 启用时，TCP 发送 keepalive 消息的频度。默认是 2 小时，建议改为 10 分钟。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_keepalive_time，默认为 7200 秒
net.ipv4.ip_local_port_range	该选项用来设定允许系统打开的端口范围，即用于向外连接的端口范围。 该参数对应系统路径为：/proc/sys/net/ipv4/ip_local_port_range 32768 61000
net.ipv4.tcp_max_syn_backlog	表示 SYN 队列的长度，默认为 1024，建议加大队列的长度为 8192 或更多，这样可以容纳更多等待连接的网络连接数。该参数为服务器端用于记录那些尚未收到客户端确认信息的连接请求最大值。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_max_syn_backlog
net.ipv4.tcp_max_tw_buckets	表示系统同时保持 TIME_WAIT 套接字的最大数量，如果超过这个数值，TIME_WAIT 套接字将立刻被清除并打印警告信息。默认为 180000，对于 Apache、Nginx 等服务器来说可以将其调低一点，如改为 5000 ~ 30 000，不同业务的服务器也可以给大一点，比如 LVS、Squid。 此项参数可以控制 TIME_WAIT 套接字的最大数量，避免 Squid 服务器被大量的 TIME_WAIT 套接字拖死。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_max_tw_buckets
net.ipv4.tcp_synack_retries	参数的值决定了内核放弃连接之前发送 SYN+ACK 包的数量。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_synack_retries，默认值为 5
net.ipv4.tcp_syn_retries	表示在内核放弃建立连接之前发送 SYN 包的数量。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_syn_retries 5

（续）

参　数	说　明
net.ipv4.tcp_max_orphans	用于设定系统中最多有多少个 TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这个数值，孤立连接将立即被复位并打印出警告信息。这个限制只是为了防止简单的 DoS 攻击。不能过分依靠这个限制甚至人为减小这个值，更多的情况是增加这个值。 该参数对应系统路径为：/proc/sys/net/ipv4/tcp_max_orphans 65536
net.core.somaxconn	该选项默认值是 128，这个参数用于调节系统同时发起的 TCP 连接数，在高并发的请求中，默认的值可能会导致链接超时或重传，因此，需要结合并发请求数来调节此值。 该参数对应系统路径为：/proc/sys/net/core/somaxconn 128
net.core.netdev_max_backlog	表示当每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许发送到队列的数据包最大数。 该参数对应系统路径为：/proc/sys/net/core/netdev_max_backlog，默认值为 1000

网络状态说明及优化命令和优化细节参考资料请看：

```
http:

//yangrong.blog.51cto.com/6945369/1321594        ←老男孩教育的优秀学生博文


http:

//oldboy.blog.51cto.com/2561410/1336488
```

[1] 本优化适合Apache、Nginx、Squid等多种Web应用，特殊的业务有可能需要略做调整。

3.4.13　定时清理邮件服务临时目录垃圾文件

CentOS 5系列的系统会默认安装Sendmail服务，因此邮件临时存放地点的路径为/var/spool/clientmqueue/。

CentOS 6默认情况下没有安装Sendmail服务，而是改装了Postfix服务，因此邮件临时存放地点的路径为/var/spool/postfix/maildrop/。

以上两个目录很容易被垃圾文件填满，导致系统的inode数量不够用，从而无法存放文件。

手动清理的方法如下：

```
[root@oldboy ~]# find /var/spool/clientmqueue/ -type f|xargs rm -f
                                              ←适合

Centos5的

Sendmail服务

[root@oldboy ~]# find /var/spool/postfix/maildrop/ -type f|xargs rm -f
                                              ←适合

CentOS 6的

Postfix服务
```

定时清理的方法为：将上述命令写成脚本，然后做成定时任务，每日凌晨0点执行一次。

下面以CentOS 6为例讲解，命令如下：

```
[root@oldboy ~]# mkdir -p /server/scripts
[root@oldboy ~]# echo "find /var/spool/postfix/maildrop/ -type f|xargs rm -f" >/serv
[root@oldboy ~]# cat /server/scripts/del_file.sh
find /var/spool/postfix/maildrop/ -type f |xargs rm -f
[root@oldboy ~]# echo "00 00 * * * /bin/sh /server/scripts/del_file.sh >/dev/null 2>
[root@oldboy ~]# crontab -l
#del clientmqueue files by oldboy at 2010-09-26
00 00 * * 0 /bin/sh /server/scripts/del_file.sh >/dev/null 2>&1
[root@web01 ~]# df -i ←查看磁盘


inode总量、剩余量、使用量



Filesystem     Inodes IUsed  IFree IUse% Mounted on
/dev/sda3      479552 55781 423771   12% /
tmpfs          127032     1 127031    1% /dev/shm
/dev/sda1       51200    38  51162    1% /boot
```

3.4.14　隐藏Linux版本信息显示

在登录到Linux主机本地（非CRT连接）前，会显示系统的版本和内核，如图3-39所示。

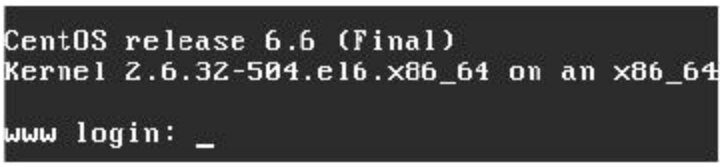


图3-39　登录Linux前的终端显示图

登录后执行如下命令，将产生登录前终端显示内容的实际存放文件。

```
[root@www ~]# cat /etc/issue
CentOS release 6.6 (


Final)



Kernel \r on an \m
[root@www ~]# cat /etc/issue.net
CentOS release 6.6 (


Final)



Kernel \r on an \m
```

执行如下命令清除Linux系统版本及内核信息：

```
[root@www ~]# > /etc/issue
[root@www ~]# cat /etc/issue
[root@www ~]# > /etc/issue.net
[root@www ~]# cat /etc/issue.net
```

3.4.15　锁定关键系统文件，防止被提权篡改

要锁定关键系统文件，必须对账号密码文件及启动文件加锁，防止被篡改。上锁命令如下：

```
chattr +i /etc/passwd /etc/shadow /etc/group /etc/gshadow /etc/inittab
```


提示：上锁后，所有用户都不能对文件修改删除。

解锁的命令如下：

```
chattr -i /etc/passwd /etc/shadow /etc/group /etc/gshadow /etc/inittab
```


提示：上锁后，如需临时操作，可以解锁后对文件进行修改，之后再上锁。

如果想更加安全，可以把chattr改名转移，防止被黑客利用。命令如下：

```
[root@oldboylinux ~]# mv /usr/bin/chattr /usr/bin/oldboy1
[root@oldboylinux ~]# chattr -i /etc/passwd /etc/shadow /etc/group /etc/gshadow /etc
-bash:


 /usr/bin/chattr:


 没有那个文件或目录
```

```
[root@oldboylinux ~]# oldboy1 -i /etc/passwd /etc/shadow /etc/group /etc/gshadow /et
[root@oldboylinux ~]# lsattr /etc/passwd ←查看被上锁后的文件属性
```

```
----i--------e- /etc/passwd提示：
```

```
chattr有很多参数，例如
```

```
-a等，如果读者要了解更多，请执行
```

```
man chattr查看
```

3.4.16　清除多余的系统虚拟账号

操作前要先根据公司系统提供的服务确定哪些账号不需要使用，如果不确定就不要操作了，一般情况下，一个规范的系统提供的服务都比较少，因此，系统中默认的绝大多数虚拟用户都可以删掉，例如：bin、adm、lp、halt、mail、uucp、operator、games、gopher、ftp、dbus、vcsa、abrt、ntp、saslauth、postfix、tcpdump等。这些用户本身也是无法登录的，因此，此项优化不是必须的。

3.4.17　为grub菜单加密码

为grub菜单加密码的目的是防止他人修改grub进行内核等启动设置，以及用单用户模式启动进行破解root密码等操作。实际上此步可以在安装系统的过程中设定，安装系统后的具体设定步骤如下。

（1）先利用/sbin/grub-md5-crypt产生一个MD5密码串，命令如下：

```
[root@www ~]# /sbin/grub-md5-crypt
Password:



Retype password:



$1$hoY96$dM9GlbjKLbi/GV8J9neOm1
```

（2）修改grub.conf文件，命令如下：

```
[root@www ~]# vi /etc/grub.conf
1 # grub.conf generated by anaconda
      2 #
      3 # Note that you do not have to rerun grub after making changes to this file
      4 # NOTICE:



  You have a /boot partition.  This means that
      5 #          all kernel and initrd paths are relative to /boot/,



 eg.
      6 #          root  (
```

hd0，

0）

```
 7 #          kernel /vmlinuz-version ro root=/dev/sda3
 8 #          initrd /initrd-[generic-]version.img
 9 #boot=/dev/sda
10 default=0
11 timeout=5
12 splashimage=（
```

hd0，

0）

/grub/splash.xpm.gz

```
13 hiddenmenu
14 password --md5 $1$hoY96$dM9GlbjKLbi/GV8J9neOm1
#注意：
```

password要加在

splashimage和

title之间，否则可能无法生效

```
15 title oldboy's CentOS 6 （
```

2.6.32-504.el6.x86_64）

```
16         root （
```

hd0，

```
0)


        17          kernel /vmlinuz-2.6.32-504.el6.x86_64 ro root=UUID=2e902291-a225-42
           73-8bd9-c9da961c65e9 rd_NO_LUKS rd_NO_LVM LANG=en_US.UTF-8 rd_NO_MD SYSFONT
           =latarcyrheb-sun16 crashkernel=auto  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM r
           hgb quiet
        18          initrd /initramfs-2.6.32-504.el6.x86_64.img
```

设置完成后，下次开机需要管理grub时就会提示输入密码，如图3-40所示。

图3-40　开机后的grub菜单

3.4.18　禁止Linux系统被ping

此项优化不是必须的，而且有时我们自己也会通过ping来检查服务器是否异常，对于要求很高的中小企业服务器，设置禁止ping也是可以的。从安全角度来说，禁止ping还是会增加系统安全的。禁止ping的命令如下：

```
[root@www ~]# echo "net.ipv4.icmp_echo_ignore_all=1" >> /etc/sysctl.conf
[root@www ~]# tail -1 /etc/sysctl.conf
net.ipv4.icmp_echo_ignore_all=1
[root@www ~]# sysctl -p
```

其实这个禁止ping的方法不是最佳策略，因为禁止ping后，我们自己也无法通过ping检查了。还原上述禁止ping的操作，因为工作中用得较少，命令如下：

```
删除

/etc/sysctl.conf中的

net.ipv4.icmp_echo_ignore_all=1，保存后，

并执行如下命令

echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_all
```

比较好的策略是通过iptables设置让特定的IP可以ping，如让内网用户ping，其他外部用户不能ping。例如：在默认策略为drop的情况下，可以执行如下命令，使10.0.0.0/24网段允许ping。

```
iptables -t filter  -I INPUT -p icmp --icmp-type 8 -i eth0 -s 10.0.0.0/24 -j ACCEPT
```

3.4.19　升级具有典型漏洞的软件版本

截止到作者写书时，openssl、openssh、bash存在很多软件漏洞，在企业场景需要对其进行修复，操作步骤如下。

1）首先查看相关软件的版本号，命令如下：

```
[root@www ~]# rpm -qa openssl openssh bash
openssl-1.0.1e-30.el6.x86_64
bash-4.1.2-29.el6.x86_64
openssh-5.3p1-104.el6.x86_64
```

2）升级已知漏洞的软件版本到最新，命令如下：

```
[root@www ~]# yum install openssl openssh bash -y
[root@www ~]# rpm -qa openssl openssh bash
openssh-5.3p1-104.el6_6.1.x86_64
bash-4.1.2-29.el6.x86_64
openssl-1.0.1e-30.el6_6.5.x86_64
```

3.5 Linux基础优化与安全重点小结

1）不用root登录管理系统，而以普通用户身份登录，通过sudo授权管理。

2）更改默认的远程连接SSH服务器端口，禁止root用户远程连接，甚至更改SSH服务只监听内网IP。

3）定时自动更新服务器的时间，使其与互联网时间同步。

4）配置yum更新源，从国内更新源下载安装软件包。

5）关闭SELinux及iptables（在工作场景中，如果有外部IP一般要打开iptables，高并发、高流量的服务器可能无法开启）。

6）调整文件描述符的数量，进程及文件的打开都会消耗文件描述符数量。

7）定时自动清理邮件临时目录垃圾文件，防止磁盘的inodes数被小文件占满（注意CentOS 6和CentOS 5要清除的目录不同）。

8）精简并保留必要的开机自启动服务（如crond、sshd、network、rsyslog、sysstat）。

9）Linux内核参数优化/etc/sysctl.conf，执行sysctl-p生效。

10）更改系统字符集为“zh_CN.UTF-8”，使其支持中文，防止出现乱码问题。

11）锁定关键系统文件，

如/etc/passwd、/etc/shadow、/etc/group、/etc/gshadow、

/etc/inittab，处理以上内容后把chattr、lsattr改名为oldboy并转移，这样就安全多了。

12）清空/etc/issue、/etc/issue.net，去除系统及内核版本登录前的屏幕显示。

13）清除多余的系统虚拟用户账号。

14）为grub引导菜单加密码。


提示：上述仅为安装Linux后的一些基础优化，更多针对不同业务服务器的优化思路见http://oldboy.blog.51cto.com/2561410/988726 。“Linux基础优化（老男孩学生活学活用）”见 http://lspgyy.blog.51cto.com/5264172/1308977 。

3.6 有关VMware虚拟机的使用问题

在VMware虚拟环境，做完上述操作后的Linux主机可以做一个快照，然后当作模板机永久保留，以后再需要学习时，可以克隆一个新的机器使用，系统安装和优化是较为简单的操作，没必要多次重复操作。不过，在克隆虚拟机及使用的过程中也会遇到一些问题。比如，在老男孩的教学过程中，发现有学生通过VMware 8的完全克隆功能快速创建了一台版本为CentOS 6.6的Linux虚拟机，但遇到了如下问题无法解决。

创建后的症状：启动之后使用ifconfig，发现无IP地址，只有回环地址为127.0.0.1。因此考虑是否因为MAC/IP地址及主机名都与克隆前的源主机相同（源主机采用手动方式配置的IP）而产生此问题。

修改IP地址和MAC地址后，执行下面的重启等命令后依然无济于事：

```
/etc/init.d/network restart
ifup eth0
```

最终的解决办法如下。

1）编辑网卡对应的eth0的配置文件：vi/etc/sysconfig/network-scripts/ifcfg-eth0，删除配置文件中HWADDR地址及UUID所在的行，如下：

```
HWADDR=00:

0c:

29:

08:

28:

9f
UUID=cee39dbb-6a10-4425-9daf-768b6e79a9c9
```

 提示：当然也可以根据实际的HWADDR和UUID修改，而不删除。见/etc/udev/rules.d/70-persistent-net.rules文件内容帮助。

eth0网卡文件修改后，代码如下：

```
[root@oldboy ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED=yes
BOOTPROTO=none
IPADDR=10.0.0.7
NETMASK=255.255.255.0
DNS2=8.8.8.8
GATEWAY=10.0.0.254
DNS1=10.0.0.254
IPV6INIT=no
USERCTL=no
```

2）清空如下文件：

```
> /etc/udev/rules.d/70-persistent-net.rules。
```

提示：机器名可以不改

3）重启系统，执行reboot或在VM外重启Linux。

原因猜测：这是VM克隆为了防止源机器和克隆机器启动网络配置地址冲突而做的保护策略，或者开发者自己也没有考虑到这个问题。

3.7 本章重点回顾

1）远程连接Linux的原理及相关软件介绍，重点掌握SecureCRT或Xshell。

2）SecureCRT的高效操作配置。

3）Linux系统的大量基本优化技巧。

4）VMware克隆虚拟机无法联网的故障排查解决。

3.8 本章知识相关考试题

1）企业场景面试题：Linux系统如何优化？

2）企业场景面试题：SSH服务连不上，如何排查？

第4章　Web服务基础

4.1　HTTP服务的重要基础

4.1.1　用户访问网站基本流程

我们每天都会使用Web客户端上网浏览网页。最常见的Web客户端就是Web浏览器，如通用的微软Internet Explorer（IE），以及技术人员偏爱的火狐浏览器、谷歌浏览器等 。当我们在Web浏览器里输入网站地址（例如：www.etiantian.org ）时，很快就会看到网站的内容。这一切似乎看起来很神奇，那么在其背后到底是怎样的实现流程呢？也许普通的上网者无需关注，但作为一个IT技术人员，特别是合格的Linux运维人员，就需要清晰的掌握了。

下面老男孩就为大家揭晓从客户端用户在Web浏览器里输入网站地址，到看到网站内容的完整访问流程。

第一步：客户端用户在浏览器里输入www.etiantian.org 网站地址，回车后，系统首先会查找系统本地的DNS缓存及hosts文件信息，确定是否存在www.etiantian.org 域名对应的IP解析记录，如果有就直接获取IP地址，然后去访问这个IP地址对应域名www.etiantian.org 的服务器。一

般第一次请求时，DNS缓存是没有解析记录的，而hosts多在内部临时测试时使用。

第二步：如果客户端本地DNS缓存及hosts文件没有www.etiantian.org域名对应的解析记录，那么，系统会把浏览器的解析请求发送给客户端本地设置的DNS服务器地址（通常称此DNS为LDNS，即Local DNS）解析，如果LDNS服务器的本地缓存有对应的解析记录就会直接返回IP地址给客户端，如果没有，则LDNS会负责继续请求其他的DNS服务器。

第三步：LDNS从DNS系统的（"."）根开始请求对www.etiantian.org域名的解析，并针对各个层级的DNS服务器系统进行一系列的查找，最终会查找到etiantian.org域名对应的授权DNS服务器，而这个授权DNS服务器正是企业购买域名时用于管理域名解析的服务器，这个授权服务器会有www.etiantian.org 对应的IP解析记录。如果此时没有，就表示企业的域名管理人员没有为www.etiantian.org 域名做解析设置，即网站还没架设好。

第四步：etiantian.org域名的授权DNS服务器会把www.etiantian.org对应的最终IP解析记录（例如：1.1.1.1）发给LDNS。

第五步：LDNS把来自授权DNS服务器www.etiantian.org 对应的IP解析记录发给客户端浏览器，并且它会把该域名和IP的对应解析缓存起

来，以便下一次更快地返回相同解析请求的记录，这些缓存记录在指定的时间（DNS TTL值控制）内不会过期。

第六步：客户端浏览器获取了www.etiantian.org 的对应IP地址，接下来，浏览器会请求获得IP地址对应的网站服务器，网站服务器接收到客户的请求并响应处理（此处的处理可能是数百台集群的服务器系统，也可能是一台云主机），将客户请求的内容返回给客户端浏览器。至此，一次访问浏览网页的完整过程就完成了。

提示：上述仅仅是客户端用户第一次访问网站的基本过程，连续访问后，系统本地和LDNS层级都会有缓存记录，再访问时流程就会有些变化，会直接取本地缓存记录，这样访问过程就很快了。在上述整个访问流程里，包含了DNS的解析流程及HTTP协议的通信原理等重要的技术点，后者在下文会有详细说明。

客户端用户第一次访问网站的整个流程解析如图4-1所示。

提示：

1）查看Windows客户端本地缓存的DNS解析记录的命令如下。

```
C:

\ >ipconfig /displaydns←意思为
```

Display the contents of the DNS Resolver Cache（显示

DNS CACHE内容），“

/displaydns”前要有空格

2）清除Windows客户端本地缓存的DNS解析记录的命令如下。

C：

\ >ipconfig /flushdns　　←意思为

Purges the DNS Resolver cache（清除

DNS CACHE

内容），“

/flushdns”前要有空格

3）Windows系统下hosts域名解析记录的位置如下。

C：

\Windows\System32\drivers\etc\hosts

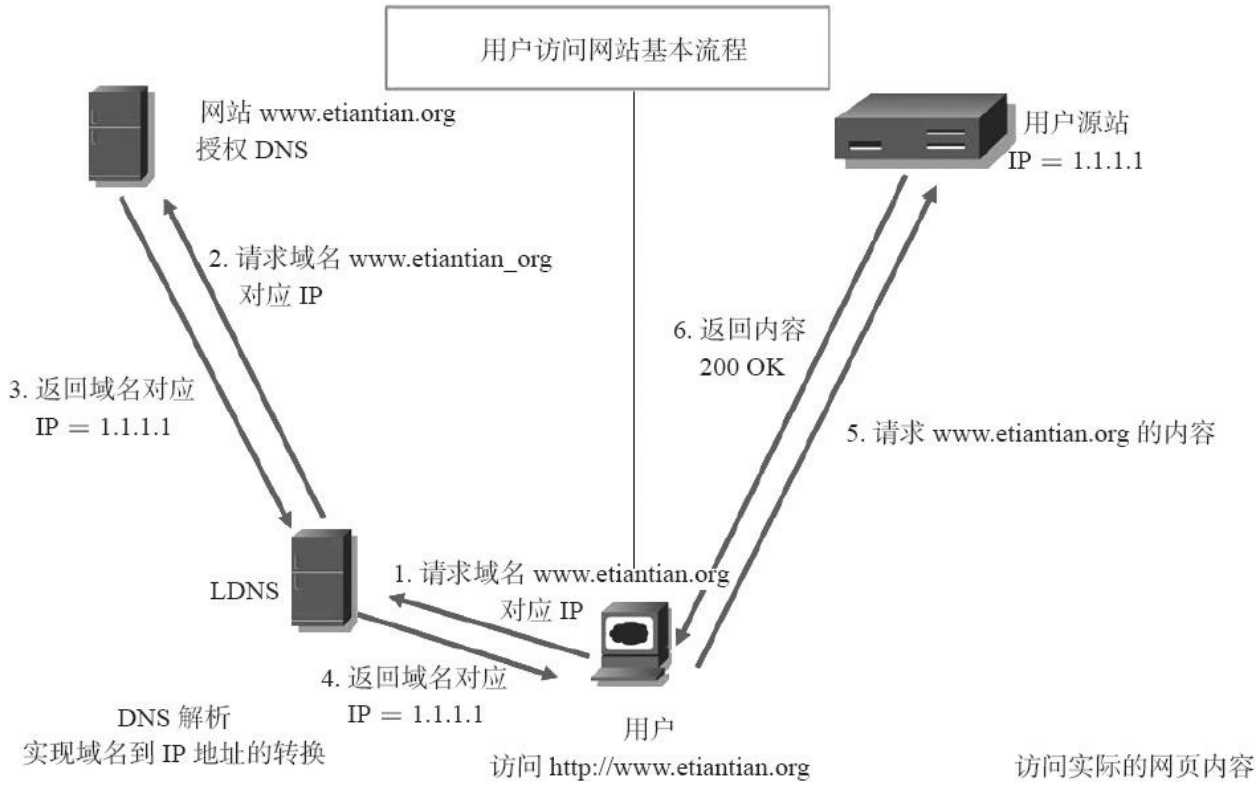


图4-1　用户访问Web网站的基本流程

4.1.2 DNS系统解析基本流程

1.DNS简介

了解完用户访问网站的基本流程后，再来了解下DNS解析的基本流程，这是企业针对运维岗位进行招聘时经常会面试的问题，因此，必须要熟练掌握。

DNS，全称为Domain Name System，它在一个网站运行中起到了至关重要的作用，其主要作用是把网站域名解析为对应的IP地址，例如：把www.etiantian.org解析为对应的IP地址记录，如1.1.1.1，这个从域名到IP的解析过程，称作A记录，即Address Record。

DNS系统除了负责这个最重要的A记录解析外，还有很多的功能呢！例如：

·设置CNAME别名记录，这个别名解析功能常被CDN加速服务商应用。

·设置MX邮件记录，这个MX记录功能，在购买或搭建邮件服务时会被用到。

·设置PTR记录，反向解析，即把IP地址解析为对应的域名，和A记

录的解析相反，此功能在邮件服务等业务中会用到。

更多的DNS功能，请参阅其他书籍或老男孩的高级架构师课程内容。

DNS系统的架构类似于一棵倒挂着的树（和Linux系统目录结构类似），它的顶点也是根（"."），只不过这个根是用点（"."）来表示的，不是目录的根斜线（"/"）。整个DNS系统的树状结构如图4-2所示。

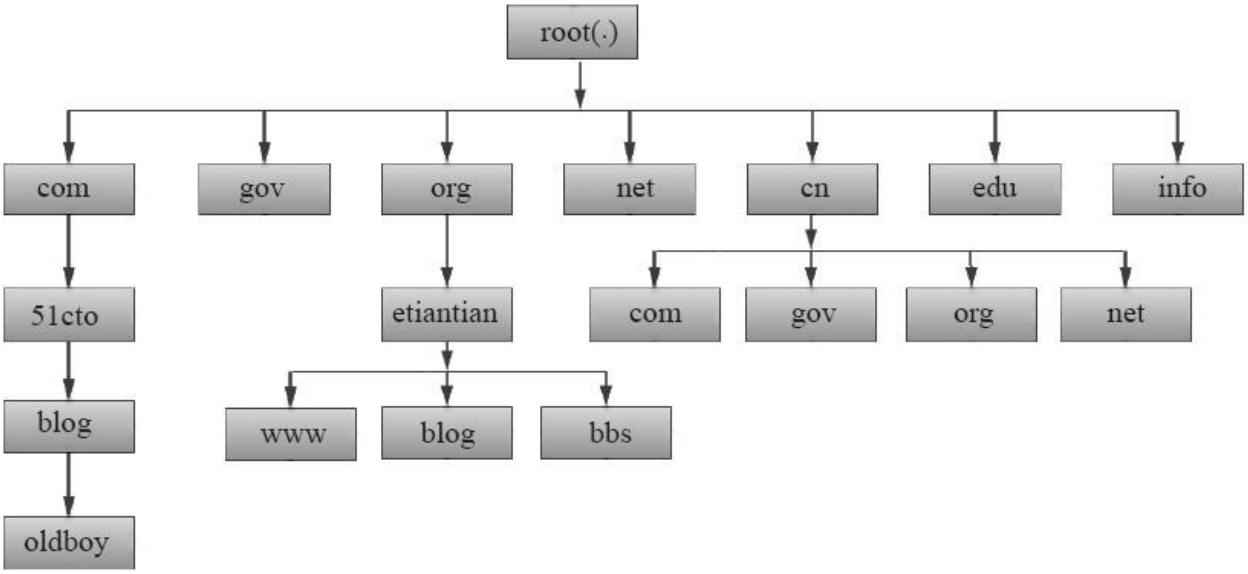


图4-2　DNS系统树状结构图

2.DNS解析流程

（1）DNS解析流程说明

在4.1.1节，我们了解了用户访问网站的基本实现流程，那么客户端是怎样一步步通过各个层级的DNS，获取到域名所对应的IP的呢？这里

给大家做一个较为详细的说明。

DNS的解析流程实际上就是从用户在客户端浏览器中输入网站地址并按回车开始的，一直持续到获取域名对应的IP，整个过程可分为如下几个步骤。

第一步：客户端用户在浏览器里输入www.etiantian.org 网站地址后回车，系统首先会查找系统本地的DNS缓存及hosts文件信息，确定是否存在www.etiantian.org 域名对应的IP解析记录，如果有就直接获取到IP地址，然后去访问这个IP地址对应的www.etiantian.org 域名的服务器。一般第一次请求时，DNS缓存是没有解析记录的，而hosts多为内部临时测试使用。

第二步：如果客户端本地DNS缓存及本地hosts文件没有www.etiantian.org 域名对应的解析记录，那么，系统会把浏览器的解析请求发送给在客户端本地设置的DNS服务器地址（通常称此DNS为LDNS，即Local DNS）解析，如果LDNS服务器的本地缓存有对应的解析记录就会直接返回IP地址给客户端；如果没有，则LDNS会负责继续请求其他的DNS服务器。

第三步：LDNS从DNS系统的（"."）根开始请求对www.etiantian.org 域名的解析，根DNS服务器在全球一共有13台，根服务器下面是没有www.etiantian.org 域名解析记录的，但是根下面有www.etiantian.org

对应的顶级域.org的解析记录，因此，根会把.org对应的DNS服务器地址返回给LDNS。

第四步：LDNS获取到.org对应的DNS服务器地址后，就会去.org服务器请求www.etiantian.org 域名的解析，而.org服务器下面也没有www.etiantian.org 域名对应的解析记录，但是有etiantian.org域名的解析记录，因此，.org服务器会把etiantian.org对应的DNS服务器地址返回给LDNS。

第五步：同理，LDNS获取到etiantian.org对应的DNS服务器地址后，就会去etiantian.org服务器请求对www.etiantian.org 域名的解析，etiantian.org域名对应的DNS服务器是该域名的授权DNS服务器，这个DNS服务器正是企业购买域名时用于管理解析的服务器（也可能是自建的授权DNS服务器），这个服务器会有与www.etiantian.org 对应的IP解析记录，如果此时没有，就表示企业的域名人员没有为www.etiantian.org 域名做解析，即网站还没架设好。

第六步：etiantian.org域名DNS服务器会把www.etiantian.org 对应的IP解析记录（例如1.1.1.1）发给LDNS。

第七步：LDNS把来自授权DNS服务器的与www.etiantian.org 对应的IP解析记录发给客户端浏览器，并且LDNS会在本地把域名和IP的对应解析记录缓存起来，以便下一次更快地返回相同解析请求的记录。至

此，整个DNS的解析流程就完成了。

（2）DNS解析流程图

上述整个DNS解析流程如图4-3所示。

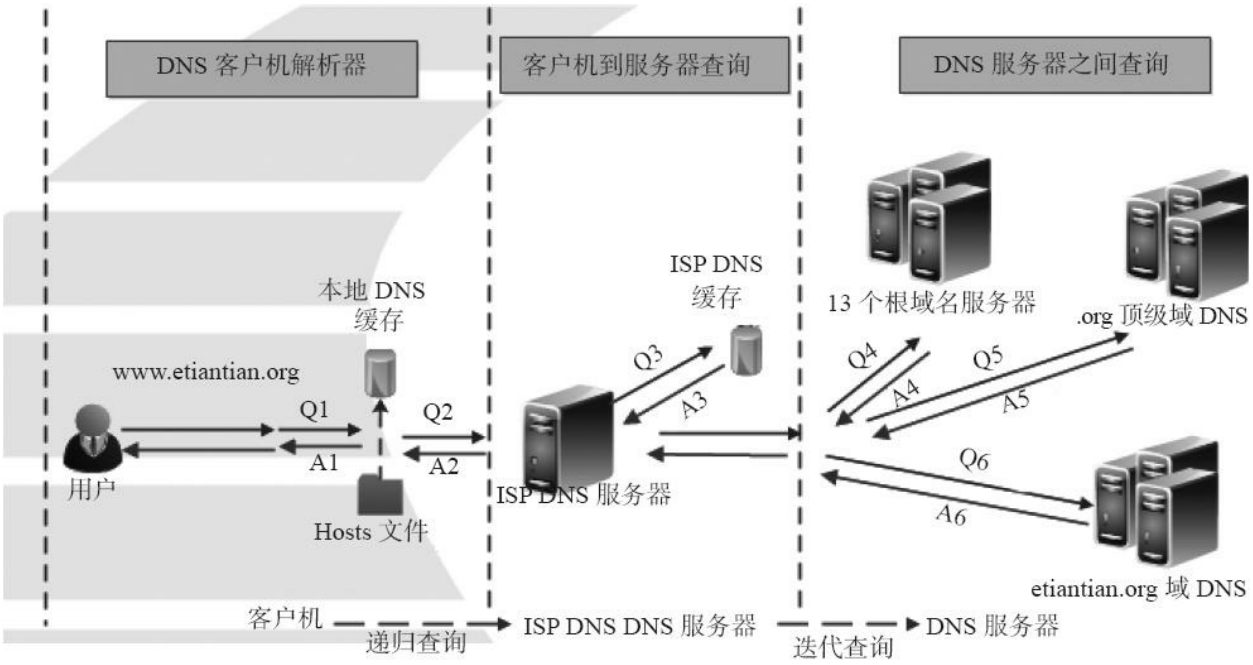


图4-3　DNS解析流程

（3）通过实践理解DNS解析流程

下面通过dig命令来帮助大家进一步理解DNS解析流程。

```
[root@oldboy ~]# dig +trace www.etiantian.org;

<<>> DiG 9.3.6-P1-RedHat-9.3.6-20.P1.el5_8.2 <<>> +trace www.etiantian.org; ;

global options:
```

```
  printcmd
.                       25991   IN      NS      m.root-servers.net.
.                       25991   IN      NS      b.root-servers.net.
.                       25991   IN      NS      g.root-servers.net.
.                       25991   IN      NS      k.root-servers.net.
.                       25991   IN      NS      f.root-servers.net.
.                       25991   IN      NS      l.root-servers.net.
.                       25991   IN      NS      d.root-servers.net.
.                       25991   IN      NS      c.root-servers.net.
.                       25991   IN      NS      i.root-servers.net.
.                       25991   IN      NS      h.root-servers.net.
.                       25991   IN      NS      j.root-servers.net.
.                       25991   IN      NS      e.root-servers.net.
.                       25991   IN      NS      a.root-servers.net.; ;


 Received 436 bytes from 192.168.120.13#53(


192.168.120.13)


 in 1 ms
# a-m一共


13个


root DNS服务器，


m.root-servers.net根


DNS负责了此次的解析



org.                    172800  IN      NS      b0.org.afilias-nst.org.
org.                    172800  IN      NS      d0.org.afilias-nst.org.
org.                    172800  IN      NS      b2.org.afilias-nst.org.
org.                    172800  IN      NS      c0.org.afilias-nst.info.
org.                    172800  IN      NS      a0.org.afilias-nst.info.
org.                    172800  IN      NS      a2.org.afilias-nst.info.; ;


 Received 437 bytes from 202.12.27.33#53(
```

m.root-servers.net）

 in 136 ms
#解析

org的一共有

6个

DNS服务器，

.org的

b0.org.afilias-nst.org. DNS负责了此次的解析

```
etiantian.org.          86400   IN      NS      dns.bizcn.com.
etiantian.org.          86400   IN      NS      dns.cnmsn.net.; ;
```

 Received 89 bytes from 199.19.54.1#53（

b0.org.afilias-nst.org）

 in 331 ms
#解析

etiantian.org的一共有

2个

DNS服务器，授权

DNS dns.bizcn.com负责了最终的

A记录解析

```
www.etiantian.org.      600     IN      A       118.26.204.147
etiantian.org.          600     IN      NS      dns.cnmsn.net.
etiantian.org.          600     IN      NS      dns.bizcn.com.; ;
```

Received 169 bytes from 218.93.205.110#53（

dns.bizcn.com）

in 39 ms

4.2 HTTP协议

4.2.1 HTTP协议简介

HTTP协议，全称为HyperText Transfer Protocol，中文名为超文本传输协议，是互联网中最常用的一种网络协议。HTTP的重要应用之一是WWW服务。设计HTTP协议最初的目的就是提供一种发布和接收HTML（一种页面标记语言）页面的方法。

HTTP协议是互联网上常用的通信协议之一。它有很多的应用，但最流行的就是用于Web浏览器和Web服务器之间的通信，即WWW应用或称Web应用。

WWW，全称为World Wide Web，常称为Web，中文译为“万维网”。它是目前互联网上最受用户欢迎的信息服务形式。HTTP协议的WWW服务应用的默认端口为80，另外一个加密的WWW服务应用https的默认端口为443，主要用于网银、支付等和钱相关的业务。当今，HTTP服务、WWW服务、Web服务三者的概念已经混淆了，在本书中也视为相同，都是指当下最常见的网站服务应用。

4.2.2 HTTP协议版本

HTTP协议从诞生之初到现在已经经历了若干个版本，其中最主要的版本为HTTP/1.0、HTTP/1.1。HTTP/1.0是第一个得到广泛使用的版本，而HTTP/1.1为当前使用的主流版本。

1.HTTP/1.0简介

HTTP/1.0是第一个得到广泛使用的HTTP版本。HTTP/1.0在HTTP/0.9的基础上增加了HTTP请求头，可支持更多的请求方法，并且能对多媒体对象进行处理。HTTP/1.0使得包含生动图片的Web页面和交互式表格成为可能，而正是这些页面和表格促使互联网被人们广泛地接受。HTTP/1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后即断开TCP连接，服务器不跟踪每个客户，也不记录过去的请求。

2.HTTP/1.1简介

HTTP/1.1的重点是修复HTTP设计中的缺陷，从可扩展性、缓存处理、带宽优化、持久连接、host头、错误通知、消息传递、内容协商等多个方面都做了相关改进。HTTP/1.1是当前互联网主流的HTTP版本。

在连接方面，HTTP/1.1支持持久连接，在一个TCP连接上可以传送

多个HTTP请求和响应，减少了建立和关闭连接的消耗和时间延迟。

在请求头方面，HTTP/1.1增加了更多的请求头和响应头信息，用以增强HTTP功能。例如：host主机头功能，可以让Web浏览器使用主机头名来明确表示要访问服务器上的哪个Web站点，这样就可以使用Web服务器在同一个IP地址和端口号上配置多个虚拟Web站点。

HTTP/1.1的持久连接，也需要增加新的请求头来帮助实现，例如，Connection请求头的值为Keep-Alive时，表示客户端通知服务器返回本次请求结果后保持连接；Connection请求头的值为close时，表示客户端通知服务器返回本次请求结果后关闭连接。HTTP/1.1还提供了与身份认证、状态管理和Cache缓存等机制相关的请求头和响应头等。

4.2.3 HTTP请求方法

在HTTP通信中，每个HTTP请求报文都包含一个方法。用以告诉Web服务器端需要执行哪些具体的动作，这些动作包括：获取指定Web页面、提交内容到服务器、删除服务器上资源文件等，这些HTTP请求报文中包含的方法被称为HTTP请求方法。其中，常用的HTTP请求方法见表4-1。

表4-1 常用的HTTP请求方法

HTTP 方法	作用描述
GET	客户端请求指定资源信息，服务器返回指定资源
HEAD	只请求响应报文中的 HTTP 首部
POST	将客户端的数据提交到服务器，例：注册表单
PUT	用从客户端向服务器传送的数据取代指定的文档内容
DELETE	请求服务器删除 Request-URI 所标识的资源
MOVE	请求服务器将指定的页面移至另一个网络地址

4.2.4 HTTP状态码

1.HTTP状态码介绍

HTTP状态码（HTTP Status Code）是用来表示Web服务器响应HTTP请求状态的数字代码。每当Web客户端向Web服务器发送一个HTTP请求时，Web服务器都会返回一个状态响应代码。这个状态码是一个三位数字代码，作用是告知Web客户端此次请求是否成功，或者是否要采取其他的动作方式。

HTTP协议1.1版本中的状态码可以分为五大类，如表4-2所示。

表4-2　不同范围的状态码及其对应的作用

状态码范围	作用描述
100 ~ 199	用于指定客户端应相应的某些动作
200 ~ 299	用于表示请求成功
300 ~ 399	用于已经移动的文件，并且常被包含在定位头信息中指定新的地址信息
400 ~ 499	用于指出客户端的错误
500 ~ 599	用于指出服务器的错误

通过表4-2可知道，HTTP响应的状态码种类很多，但是在实际工作场景中，经常遇到的状态码却不多，老男孩把生产场景下常见的重要状态码及对应的作用整理为表4-3供读者学习参考[1]。

表4-3　生产场景常见的状态码及其对应的作用

状态代码	详细描述说明
200-OK	服务器成功返回网页，这是成功的 HTTP 请求返回的标准状态码
301-Moved Permanently	永久跳转，所请求的网页将永久跳转到被设定的新位置，例如：从 etiantian.org 跳转到 www.etiantian.org
403-Forbidden	禁止访问，虽然这个请求是合法的，但是服务器端因为匹配了预先设置的规则而拒绝响应客户端的请求，此类问题一般为服务器或服务权限配置不当所致
404-Not Found	服务器找不到客户端请求的指定页面，可能是客户端请求了服务器上不存在的资源所致

（续）

状态代码	详细描述说明
500-Internal Server Error	内部服务器错误，服务器遇到了意料不到的情况，不能完成客户的请求。这是一个较为笼统的报错，一般为服务器的设置或内部程序问题导致。例如：SELinux 开启，而又没有为 HTTP 设置规则许可，客户端访问就是 500
502-Bad Gateway	坏的网关，一般是代理服务器请求后端服务时，后端服务不可用或没有完成响应网关服务器。这通常为反向代理服务器下面的节点出问题所致
503-Service Unavailable	服务当前不可用，可能是服务器超载或停机维护导致的，或者是反向代理服务器后面没有可以提供服务的节点
504-Gateway Timeout	网关超时，一般是网关代理服务器请求后端服务时，后端服务没有在特定的时间内完成处理请求。多数是服务器过载导致没有在指定的时间内返回数据给前端代理服务器

2.HTTP状态码的命令行查看

可以通过curl命令（附带相关参数）在Linux命令行查看HTTP响应的数字状态码，命令如下：

```
[root@oldboy ~]# curl -I www.etiantian.org
HTTP/1.1 200 OK            ←


 200即为状态码


Server:


 nginx/1.6.2
Date:
```

```
 Mon,

 09 Mar 2015 05:

42:

25 GMT
Content-Type:

 text/html;

 charset=utf-8
Connection:

 keep-alive
Vary:

 Accept-Encoding
[root@oldboy ~]# curl -I -s -w %{http_code} -o /dev/null www.etiantian.org
200                ←

 200即为状态码
```

提示：

1）学习Linux时，要能在搭建HTTP服务时模拟出这些状态码对应的状态，这样在实际生产场景中遇到相关问题才能够快速解决掉。

2）HTTP/1.0中只定义了16个状态响应码，而在HTTP/1.1中新增了

24个状态响应码。

[1] 生产环境常见HTTP状态码的博客文章见http://oldboy.blog.51cto.com/2561410/716294。

4.2.5 HTTP报文

HTTP报文中有很多行内容，这些行的字段都是由一些ASCII码串组成，但各个字段的长度是不同的。HTTP报文可分为两种，一种是从Web客户端发往Web服务器的HTTP报文，称为请求报文（Request Message）。另外一种是从Web服务器发往Web客户端的报文，称为响应报文（Response Message），HTTP的请求和响应报文的格式类似。

1.HTTP请求报文（Request Message）介绍

HTTP请求报文由请求行、请求头部（header）、空行和请求报文主体几个部分组成，表4-4给出了请求报文的一般格式。

表4-4 HTTP请求报文格式说明

报文格式	报文信息
请求行	请求方法 URL 协议版本
请求头	字段名 1：值 1 字段名 2：值 2 …… 例如： Accept: image/gif, image/jpeg Accept-Language: zh-cn ……
空行	空白无内容
请求报文主体	GET 方法没有请求报文主体，POST 方法才有

下面对HTTP请求报文的每个部分逐一阐述。

（1）请求行

请求行是请求报文的第一行，用来说明客户端想要做什么。内容由请求方法字段、URL字段和HTTP协议版本字段组成，它们之间用空格分隔。下面以GET/index.html HTTP/1.1为例来说明请求报文的起始请求行信息详情，见表4-5。

表4-5　请求报文的起始请求行信息

请求方法字段示例	URL 字段示例	HTTP 协议版本
GET	/index.html	HTTP/1.1

（2）请求头部

请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“：”分隔。请求头部的作用是通过客户端把请求的相关信息告诉给服务器，常见的请求头部信息见表4-6。

表4-6　常见的请求头部信息

请求头信息	说　明
Accept: image/gif, image/jpeg	媒体类型
Accept-Language: zh-cn	语言类型
Accept-Encoding: gzip, deflate	支持压缩
User-Agent: Mozilla/4.0（compatible；MSIE 6.0；Windows NT；...）	客户端类型
Host: www.etiantian.org	主机名

与请求报文相关的最常用的请求头是Content-Type和Content-Length。

（3）空行

最后一个请求头部信息之后是一个空行，通过发送回车符和换行符，通知Web服务器空行以下不会有请求头部的信息了。

（4）请求报文主体

请求报文主体中包括了要发送给Web服务器的数据信息。请求报文主体不会应用于HTTP的GET命令方法，而是应用于POST方法。POST方法适用于需要客户填写表单的场合。请求报文的主体信息此处就不再举例了。

2.HTTP响应报文（Response Message）介绍

HTTP响应报文由起始行、响应头部（header）、空行和响应报文主体这几个部分组成，和HTTP请求报文格式类似。表4-7给出了HTTP响应报文的一般格式。

表4-7　HTTP响应报文的一般格式

报文格式	报文信息
起始行	协议及版本号、数字状态码、状态信息
响应头部	字段名 1：值 1 字段名 2：值 2 …… 例如： Content-Type: text/html；charset=utf-8 Content-Length:78 ……
空行	空白无内容
响应报文主体	＜ html ＞ ＜ head ＞＜ title ＞ oldboy's blog ＜ /title ＞＜ /head ＞ ＜ body ＞ I am oldboy, mysql blog is http://oldboy.blog.51cto.com ＜ /body ＞ ＜ /html ＞

下面对响应报文的每个部分逐一阐述。

（1）起始行

响应报文的起始行也叫状态行，用来说明服务器响应客户端请求的状况。一般为协议及版本号、数字状态码、状态情况。例如：HTTP/1.1 200 OK。

（2）响应头部

和请求报文类似，起始行的后面一般有若干个头部字段。每个头部字段都包含一个名字和一个值，两者之间用冒号分隔。头部结尾也是以一个空行结束的。常见的头部信息有：

Content-Type：

```
 text/html;

 charset=utf-8
Content-Length:

78.....
```

（3）空行

最后一个响应头部信息之后是一个空行，通过发送回车符和换行符，通知客户端空行下文无头部信息了。

（4）响应报文主体

响应报文主体中装载了要返回给客户端的数据。这些数据可以是文本，也可以是二进制的（如图片、视频），下面是响应报文主体的html格式文本数据示例。

```
<

html>

<

head><

title>
```

```
oldboy's blog<

/title><

/head>

<

body>

I am oldboy,

mysql blog is http:

//oldboy.blog.51cto.com<

/body>

<

/html>
```

3.一个简单的请求报文和应答报文示例

图4-4为HTTP报文请求应答流程图，从该图中的示例可以看到，Web客户端发送了一条HTTP请求报文，请求资源

http://www.etiantian.org/index.html，请求报文的起始行中有一个GET命令，资源名称为/index.html。使用的是HTTP/1.1协议，由于请求的方法为GET，因此请求报文不需要有主体，因为从服务器上获取（GET）一个简单的页面不需要在请求报文的主体中发送请求数据。

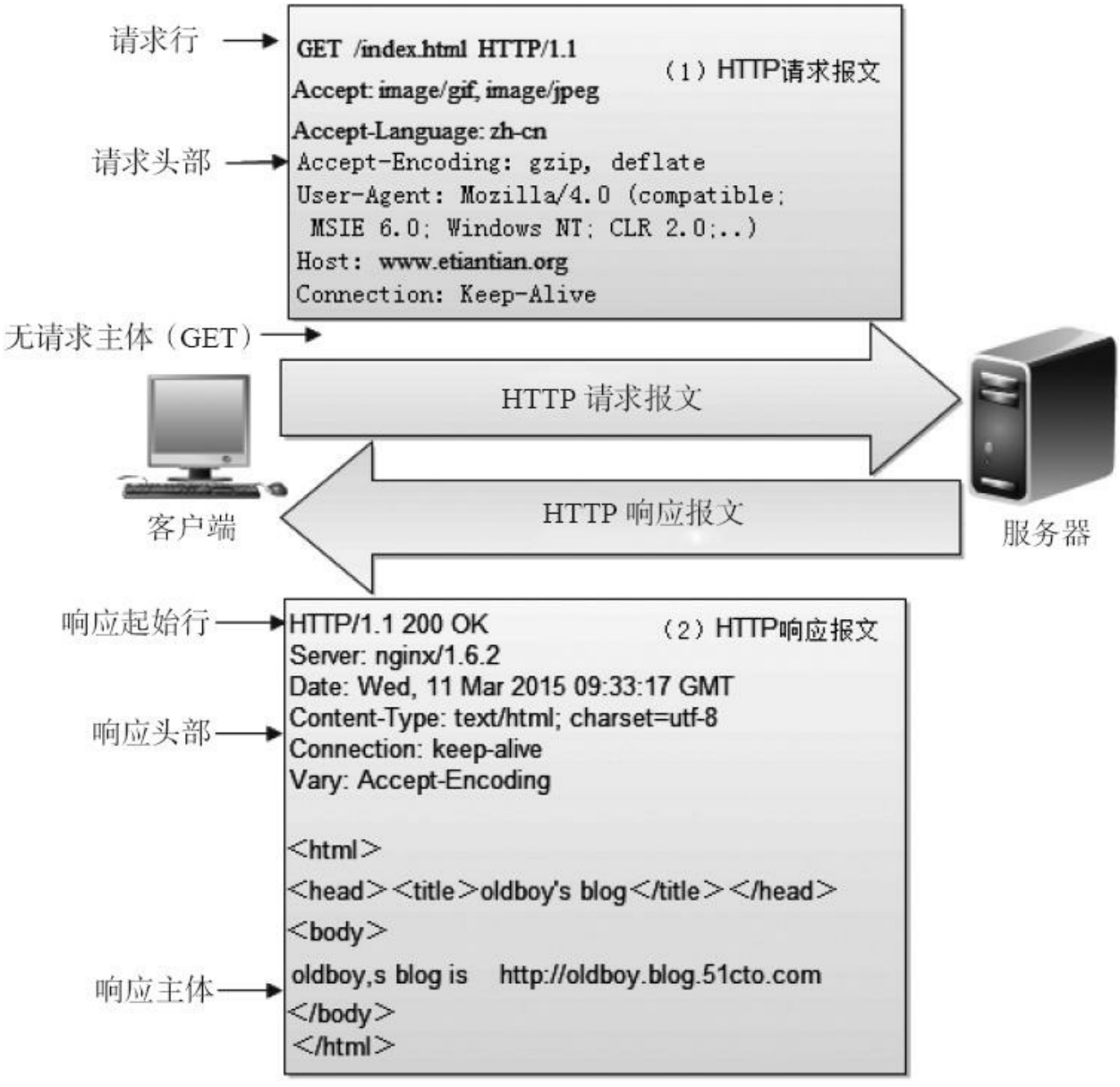


图4-4　HTTP报文请求应答流程详图

Web服务器接受到请求后，返回一条HTTP响应报文。这条响应报文中包含了HTTP的版本号（HTTP/1.1）、成功的状态码（200）、状态描述（OK），以及若干响应头部字段，在所有这些内容之后跟着包含了所请求资源的响应主体。Content-Length首部说明了响应主体的长度，Content-Type首部说明了文档的MIME类型。

4.2.6 HTTP协议原理及重点分析

HTTP协议属于OSI模型中的第七层应用层协议，HTTP协议的重要应用就是WWW服务应用，下面就以WWW服务应用为例介绍HTTP协议的通信原理。以HTTP协议进行通信时，需要有客户端（即终端用户）和服务器端（即Web服务器），在Web客户端向Web服务器发送请求报文之前，先要通过TCP/IP协议在Web客户端和服务器之间建立一个TCP/IP连接。整个HTTP协议请求的工作流程如下：

1）终端客户在Web浏览器地址栏输入访问地址http://www.etiantian.org/index.html 。

2）Web浏览器请求DNS服务器把域名www.etiantian.org 转换成Web服务器的IP地址，此处的解析过程就是DNS解析的原理流程，前面已经讲过了，此处不再赘述。

3）Web浏览器将端口号（默认是80）从访问地址（URL）中解析出来。

4）Web浏览器通过解析后的IP地址及端口号与Web服务器之间建立一条TCP连接。

5）建立TCP连接后，Web浏览器向Web服务器发送一条HTTP请求

报文，请求报文的内容格式及信息细节前面已经讲过了，此处不再赘述。

6）Web服务器响应并读取浏览器的请求信息，然后返回一条HTTP响应报文，响应报文的内容格式及信息细节前文也已经讲过了，此处不再赘述。

7）Web服务器关闭HTTP连接，关闭TCP连接，Web浏览器显示访问的网站内容到屏幕上。

上述就是HTTP协议通信过程，整个通信原理的重要知识点有：

·用户访问网站的流程。

·DNS解析流程细节。

·建立TCP连接发送HTTP报文的过程。

·HTTP请求报文的细节。

·HTTP响应报文的细节。

·Web服务器请求后端集群的细节（此处后文会讲到）。

如果读者在企业面试谈到HTTP原理时能把上述5点细节说清楚，则大事必成！

事实上，DNS解析原理、HTTP协议原理都是高薪面试的重点，是高级运维必会知识，这里对其中的重要知识点进行汇总，如下：

·HTTP协议位于OSI模型中第7层应用层。

·HTTP协议的重要应用是WWW服务。

·用户上网流程、DNS解析流程。

·DNS解析获取到IP后，建立TCP连接，然后发送HTTP请求的细节和服务器响应细节。

·HTTP请求报文与HTTP响应报文知识。

·到达HTTP服务后，请求后端集群节点的流程为Nginx→FastCGI→PHP→（数据库、存储等）。

4.3 HTTP资源

4.3.1 媒体类型

互联网上的数据有很多不同的类型，Web服务器会把通过Web传输的每个对象都打上MIME类型（即MIME type）的数据格式标签。最初设计MIME（Multipurpose Internet Mail Extension，多用途因特网邮件扩展）是为了解决在不同的电子邮件系统之间搬移报文时存在的问题。MIME在电子邮件系统中工作得非常好，后来，HTTP也支持了这个功能，用它来描述数据并标记不同的数据内容类型。

当Web服务器响应HTTP请求时，会为每一个HTTP对象数据加一个MIME类型。当Web浏览器获取到服务器返回的对象时，会去查看相关的MIME类型，并进行相应处理。

MIME类型存在于HTTP响应报文的响应头部信息里，它是一种文本标记，表示一种主要的对象类型和一个特定的子类型，中间由一条斜杠来分隔。表4-8为生产场景最常见的MIME类型。

表4-8　生产场景最常见的MIME类型

MIME 类型	文件类型
text/html	html、htm、shtml 文本类型
text/css	css 文本类型
text/xml	xml 文本类型
image/gif	gif 图像类型
image/jpeg	jpeg、jpg 图像类型
application/javascript	js 文本类型
text/plain	txt 文本类型
application/json	json 文本类型
video/mp4	mp4 视频类型
video/quicktime	mov 视频类型
video/x-flv	flv 视频类型
video/x-ms-wmv	wmv 视频类型
video/x-msvideo	avi 视频类型

可以从www的重要服务软件Nginx的配置文件conf目录下，查看其支持的媒体类型，相关命令及内容如下：

```
[root@nginx conf]# less mime.types
types {
    text/html                             html htm shtml;

    text/css                              css;

    text/xml                              xml;

    image/gif                             gif;

    image/jpeg                            jpeg jpg;

    application/javascript                js;
```

. . . 省略

. . .

4.3.2 URL介绍

URL，全称为Uniform Resource Location，中文翻译为统一资源定位符，也被称为网页地址（网址）。如同门牌一样，它是因特网上标准的资源唯一地址。通俗地说，URL是Internet上用来描述信息资源的字符串，主要用在各种WWW客户端和服务器程序上。URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。严格来讲，每个URL都是一个URI，它标识一个互联网资源，并指定对其进行操作或取得该资源的方法。

URL的格式由下列三部分组成：

第一部分是协议，例如：http。

第二部分是主机资源服务器IP地址或域名（端口号），例如：www.etiantian.org 。

第三部分是主机资源的具体地址，如目录和文件名等，例如：oldboy/index.html 。

第一部分和第二部分之间用“：//”符号隔开，第二部分和第三部分用“/”符号隔开。第一部分和第二部分是不可缺少的，第三部分可以省略。表4-9列举了一个标准的URL及说明。

表4-9　标准的URL及说明

协议	分隔符号	IP 地址域名	分隔符号	资源目录地址
http	://	www.etiantian.org	/	oldboy/index.html
http	://	www.oldboyedu.com	/	video/index.html

4.3.3 URI介绍

URI，全称为Uniform Resource Identifier，中文翻译为统一资源标识符，是一个用于标识某一互联网资源名称的字符串。这个字符串在世界范围内唯一标识并定位某一个信息资源。互联网上每个可用的数据资源（如HTML、图片、视频等）皆通过统一资源标识符进行定位。表4-10给出了网站URI（URL是URI的子集）说明；表4-11给出了指向一个用户邮箱的URI。

表4-10　网站URI说明

协议	分隔符号	IP 地址域名	分隔符号	资源目录地址
http	://	www.etiantian.org	/	oldboy/index.html
http	://	www.oldboyedu.com	/	video/index.html

表4-11　指向一个用户邮箱的URI

协议（服务形式）	分隔符号	用户名	分隔符号	域名
mailto	:	oldboy	@	etiantian.org

大多数读者都熟悉URL，而不是URI。URL是URI命名机制的一个子集。

4.3.4 静态网页资源

1.静态网页资源介绍

在网站设计中，纯粹HTML格式的网页（可以包含图片、视频、JS（前端功能实现）、CSS（样式）等）通常被称为“静态网页”，早期（大约2000年左右）的网站大多都是由静态网页制作的。静态网页是相对于动态网页而言的，是指没有后台数据库、不含程序（如PHP、JSP、ASP）、不可交互的网页。

2.静态网页资源特点

静态网页资源的特点是，开发者编写的是什么，它显示的就是什么，一旦编写完成，就不会有任何改变。静态网页的维护和更新相对比较麻烦，每个不同的网页都需要单独编辑更新，静态网页一般适用于更新较少的宣传展示型网站（例如：酒、家具、水果等的宣传网站），是早期（2000年左右）很多中小网站展示的形式。

静态网页资源的对应程序及资源文件的常见扩展名为：

·纯文本类程序或文件，如.htm、.html、.xml、.shtml、.js、.css等。

·图片类文件或数据文档，

如.jpg、.gif、.png、.bmp、.txt、.doc、.ppt等。

·视频类流媒体文件，如.mp4、.swf、.avi、.wmv、.flv等。

静态网页资源有几个重要的特征：

1）每个页面都有一个固定的URL地址，且URL一般以.htm、.html、.shtml等常见形式为后缀，而且地址中不含有问号“？”或“&”等特殊符号。

2）网页内容一经发布到网站服务器上，无论是否有用户访问，每个网页的内容都是保存在网站服务器文件系统上的，也就是说，静态网页是实实在在保存在服务器上的文件实体，每个网页都是一个独立的文件。

3）网页内容是固定不变的，因此，容易被搜索引擎收录（容易被用户找到）（优点）。

4）因为网页没有数据库的支持，所以在网站制作和维护方面的工作量较大，当网站信息量很大时，完全依靠静态网页比较困难（缺点）。

5）网页的交互性较差，在程序的功能实现方面有较大的限制（缺点）。

6）网页程序在用户浏览器端解析，如IE浏览器，程序解析效率很高，由于服务器端不进行解析，并且不需要读取数据库，因此服务器端可以接受更多的并发访问。当客户端向服务器请求数据时，服务器会直接从磁盘文件系统上返回数据（不做任何解析），待客户端拿到数据后，在浏览器端解析并展现出来（优点）。

网站静态页面的特点就相当于在餐馆吃火锅，餐馆把原材料和工具都给你准备好，你自己只需要涮着吃就行，不需要大厨给你炒菜做菜了。因此，对于餐馆来讲，服务顾客的效率大大提高了。由于静态页面不需要在服务器端解析，因此服务器的压力也大大减轻了。

3.静态网页语言

常见的静态网页语言有HTML、JS、CSS、XML、SHTML等，具体语言的特点不在本书范围内，读者如果感兴趣可以参考相关网页开发的书籍。

回顾一下静态网页的核心特点，如下：

1）客户浏览器端解析程序。

2）不需要读取数据库，性能和效率很高。

3）后端没有数据库支持，和用户的交互性差，功能实现很差。

4.有关静态网页的架构思想

在高并发、高访问量的场景下做架构优化，涉及的关键环节就是把动态网页转成静态网页，而不是直接请求数据库和动态服务器，并且可以把静态内容推送到前端缓存（或CDN）中提供服务，这样就可以提升用户体验，节约服务器和维护成本。

4.3.5　动态网页资源

1.动态网页资源介绍

所谓的动态网页是与静态网页相对而言的，也就是说，动态网页的URL后缀不是.htm、.html、.shtml、.xml、.js、.css等静态网页的常见扩展名形式，而是.asp、.aspx、.php、.js、.do、.cgi等形式的，并且一般在动态网页网址中会有标志性的符号——“？，&”，此外，在大多数情况下后端都需要有数据库支持。图4-5为动态网页地址的形式。

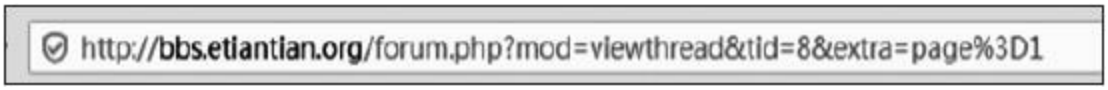


图4-5　动态网址形式

2.动态网页资源特点

1）网页扩展名后缀常见为：.asp、.aspx、.php、.jsp、.do、.cgi等。

2）网页一般以数据库技术为基础，大大降低了网站维护的工作量。

3）采用动态网页技术的网站可以实现更多的功能，如用户注册、用户登录、在线调查、投票、用户管理、订单管理、发博文等。

4）动态网页并不是独立存在于服务器上的网页文件，当用户请求服务器上的动态程序时，服务器解析这些程序并可能通过读取数据库来返回一个完整的网页内容。

5）动态网页中的“？”在搜索引擎的收录方面存在一定的问题，搜索引擎一般不会从一个网站的数据库中访问全部网页，或者出于技术等方面的考虑，搜索蜘蛛一般不会去抓取网址中“？”后面的内容，因此在企业通过搜索引擎进行推广时，需要针对采用动态网页的网站做一定的技术处理（伪静态技术），以便适应搜索引擎的抓取要求。

程序在服务器端解析，这相当于顾客点餐，饭店厨师做饭做菜，耗时长、效率低。在这个过程中，会消耗大量的CPU和内存、I/O等资源，并且多数还要提供读取数据库等服务，因此，其访问效率远不如静态网页，在服务器端解析动态程序的服务常见的有PHP引擎、Java容器（Tomcat、Resin、Jboss、Weblogic）等。

3.有关动态网页的架构思想

一般来说，静态网页的性能效率是动态网页的10~30倍。且动态网站效率很差，并发能力也很低，在高并发场景中，应尽可能转换成静态网页提供服务。动态转静态几乎是所有高并发网站必备的架构方案思路，也是高级架构师的职责所在。

此外，动态转静态也要根据业务需求设计，例如，对于更新频繁的

网站，如果设计不好就可能会产生数据不一致的情况，即用户看到的数据不是网站最新的内容，而是静态的内容。

4.3.6 伪静态网页

1.伪静态网页介绍

顾名思义，伪静态就是通过某些技术（如rewrite）把动态网页的URL地址伪装成静态网页URL地址，但实质上用户访问的还是动态网页，只不过看起来是符合静态网页地址特征的，因此，用户及某些搜索引擎会误认为是静态网页。

2.伪静态网页特点

从网站的URL地址看，伪静态表面上看起来是静态内容（如地址结尾带html），但这其实是通过rewrite规则实现的URL地址重写。改写后的URL地址规范、美观，有利于搜索引擎抓取，以及提升用户访问体验。如：http://oldboy.blog.51cto.com/2561410/803606 和 http://www.discuz.net/forum-3967-1.html 这两个地址都是伪静态的。因为伪静态网页还是动态网页，所以从性能上考虑，伪静态功能不但没有提升网站性能，反而会降低网站的性能。这一点读者要理解。可能有些读者就会有疑问了，为什么动态网页不能直接转成静态网页呢？

3.伪静态网页作用

由于搜索引擎无法正确读取带参数的动态网页内容，以致网页中很

多具有丰富信息的页面无法被搜索引擎收录，因此就有了伪静态技术。它的作用是把动态网页URL改写成静态网页的URL，虽然消除了动态网页中的参数，但是并不需要生成任何网页，仅仅是改变了网页地址路径。这样做的主要目的有两个，一是让搜索引擎收录网站内容，让更多的用户访问企业网站内容。二是提升用户访问体验，动态地址带特殊符号等的URL看起来也不如静态网页地址舒服。

4.伪静态网页的缺点

动态网页伪装成静态网页，虽然可以让搜索引擎收录，并且可提升用户访问体验，但伪静态网页并不能提升网站的访问效率，从理论上说还会降低网站的性能，没有动态转静态网页效率高，不过，对于一些并发不是很大的网站来说，或者是硬件资源充足的网站来说，伪静态还是一个不错的功能。

现在很多大型的网站都采用了动态网页生成静态网页的技术，用于消除动态网页中的参数，使搜索引擎收录更多的内容，达到优化网站的效果。当网站遇到访问瓶颈时，如果有技术力量将动态网页转化成静态网页，那就再好不过了，但是，有些更新频率很快的业务并不适合这么做。此外，如果将动态转为静态有很大难度，这时也可以考虑将其设置伪静态。相关架构内容可以参考老男孩高级架构师课程内容。

5.伪静态网页小结

·利用rewrite技术将动态网页伪装成静态网页（URL地址改写）。

·便于搜索引擎搜录，提升用户访问量及用户体验。

·访问性能没有提升，并且转换成伪静态会消耗资源，因此性能反而会下降。

·尽可能地将动态网页转换成真正的静态页面。

·并发量不是很大或动态更新过于频繁时，用rewrite实现伪静态也是不错的。

·伪静态网页的实现过程，一般由产品运营提出需求，开发和运维共同实现。

课外作业： 请在学习完如何部署开源站点discuz（bbs）和wordpress（blog）程序后，把这两个网站的访问地址做成伪静态（提示，官方网站上有说明）。

4.3.7　生产Web架构优化实战方案

由于静态网页程序在客户端解析，大大降低了服务器端的访问压力，因此在实际高并发的网站架构中，可以考虑把用户请求的数据解析后存成静态文件放于磁盘中或内存中，从而降低动态服务器的压力，节约企业成本，提升用户体验。有关高并发网站架构从动态转静态的内容，可以参考老男孩的文章《浅谈千万级PV/IP规模高性能高并发网站架构》：http://oldboy.blog.51cto.com/2561410/736710 。

下面介绍几个不用更改企业业务产品代码就可以实现高并发网站架构从动态转静态的例子。

1.门户新闻业务

新闻网站的特点是一旦发布完成，几乎不会再改动网页内容。因此，新闻业务内容的静态化相对比较简单。

第一步：程序要支持发布的内容由动态转成静态的功能。

第二步：运营编辑人员发布新闻网页后，后台程序立刻将动态网页生成静态文件。

第三步：运维人员通过发布或事件触发把运营编辑生成的静态网页

发布到事先搭建好的公司缓存集群服务器上，或者把静态内容同步到购买的全国所有CDN服务器节点上，然后，再提供给用户提供访问浏览。

2.视频网站业务

视频网站和新闻网站类似，特点都是一旦发布完成，几乎不会再改动网页内容。因此，实现视频业务网站高效访问也很简单。

以优酷视频网为例，用户在上传视频时，需要经历转码→审核的过程（大概1个小时）。此外，一些热点视频也可能会被提前推送同步到CDN的核心节点或全国所有CDN服务器节点，这样用户访问时才会更快。

3.Blog/BBS/SNS/微博社区业务/电商（如淘宝、京东）

这几类业务由动态转静态是比较困难的，因为，用户发布内容后，可能会随时更新并查看，对于这种情况，一般会通过异步的方式来处理，例如通过消息中间件技术加上NoSQL集群技术来实现转换。较为详细的说明见博客文章《浅谈千万级PV/IP规模高性能高并发网站》（http://oldboy.blog.51cto.com/2561410/736710 ）和（http://edu.51cto.com/course/course_id-3093.html ）。

4.4 网站流量度量术语

4.4.1 IP

IP（独立IP），即Internet Protocol，这里指独立IP数，独立IP数是指不同IP地址的计算机访问网站时被计的总次数。独立IP数是衡量网站流量的一个重要指标。一般一天内（00：00-24：00）相同IP地址的客户端访问网站页面只会被计为一次，记录独立IP的时间可为一天或一个月，目前通用的标准为“一天”。

假设有部分同学在老男孩教育的局域网中同时打开了老男孩博客（http://oldboy.blog.51cto.com ），请问对于51CTO网站是几个独立IP？答：是一个独立IP。

这是因为，国内所有的公司几乎都是采用局域网共享上网的，即通过路由器NAT地址转换上网，每个计算机在局域网内的私有IP是不同的，但是在外网上，就必须由路由器把每个私网地址转换成了路由器接口的固定公网IP（多IP映射暂不考虑），所以说，对于网站来说一天内多个相同IP的客户端访问会被计为一个独立IP。

再假设一个客户机用户通过ADSL等直接拨号上网，但是上网的时候偶尔掉线，一共重新拨号了3次（相近时间重新拨号IP地址相同的几

率是极小的），然后每次都继续打开老男孩的博客地址，请问此时，网站独立IP数是多少？答：是3个独立IP。

由此可见，通过独立IP数度量网站访问量，和实际的访问情况不是很匹配。国内的企业、学校等多数是用NAT上网的，一个独立IP背后可能有数十上百个客户端访问。独立IP数虽然不是很准确，但却是IT技术人员比较关心的一个衡量网站的指标。

4.4.2 PV

PV（访问量）即Page View，中文翻译为页面浏览，即页面浏览量或单击量，不管客户端是不是相同，也不管IP是不是相同，用户只要访问网站页面就会被计算PV，一次计一个PV。

PV的具体度量方法就是从客户浏览器发出一个对Web服务器的请求（Request），Web服务器接到这个请求后，将该请求对应的网页（Page）发送给浏览器，这样就产生了一个PV。这里有一个问题，就是只要这个请求发送给了浏览器，无论这个页面是否完全打开（或下载完成），都会被计数（1次为1个PV），一般为了防止用户快速刷PV，很多网站会把PV的统计程序放在页面的最下面。

用PV衡量网站时，PV数反映的是浏览某网站的页面数量，每刷新一次页面也算一次。因此，可以说PV数与来访用户的数量成正比，但PV数并不是真正的页面来访者数量，而是网站被访问的页面数量，因为一个来访者可能产生多个PV。

问：如果一个用户要访问赶集网或58同城租房，你觉得用户可能会产生多少PV？

答：平均可能会有十几到几十个PV，一个来访者访问网站的PV数

的多少是和网站提供的业务直接相关的。对于分类网站，用户浏览网站可能是为了找房子、找工作，因此一个用户访问的页面会很多，自然PV也就会很多。

PV（Page View）是网站被访问的页面数量的一个指标，但不能直接知道有多少人访问了这个网站。

一个来访者访问网站，可能产生若干PV数，但是独立IP数就只有1个，因此，如果对比一个网站的独立IP数和PV数，不难看出，PV数一定会大于等于独立IP数，其比例视网站的业务而定，对于分类门户，可能会达到10：1，甚至更多。

4.4.3 UV

UV（独立访客）即Unique Visitor，同一个客户端（PC或移动端）访问网站被计为一个访客。一天（00：00-24：00）内相同的客户端访问同一个网站只计一次UV。UV一般是以客户端Cookie等技术作为统计依据的，实际统计会有误差。

考虑到一台客户端计算机可能会有多人使用，因此，UV（独立访客）实际上并不一定是独立的自然人访问。

4.4.4　企业网站对IP、PV、UV的度量

1.对IP的度量

·分析所有Web服务器的访问日志信息，对IP地址段去重后计数，这是IT人员的基本计算手段。

·在网站的每一个（所有）页面结尾，嵌入JS等统计程序代码，待用户加载网页后，IP即传给统计IP的服务器，这种方法一般被第三方统计公司或在企业内部开发日志分析程序时使用。

·用第三方大家比较信任的统计工具，例如：谷歌的统计（GA）。

IP的统计方法简单、易用，因此，成了多数网站衡量网站流量的重要指标之一。

2.对PV的度量

·分析Web服务的访问日志（需要排除JS、CSS及各种图片的日志信息），只计算HTML、PHP等页面数量。

·在网站的每一个页面结尾，嵌入JS等统计程序代码，待用户加载网页后，访问数量即传给统计PV的服务器，这种方法一般被第三方统计公司或在企业内部开发日志分析程序时使用。

·用第三方大家比较信任的统计工具，例如：谷歌的统计（GA）。

PV的统计方法也很简单、易用，因此也是多数网站衡量网站流量的重要指标之一。

 提示：IP和PV概念，统计方法也是Linux运维人员要掌握的重点。

3.对UV的度量

·通过客户端HTTP请求报文分析。

一个客户端会多次请求网站服务器，每次HTTP请求都会携带客户端自身的大量信息，比如：IP地址、请求发出的时间、浏览器版本，操作系统版本等。网站服务器对这些请求进行分析，如果这些请求满足一些共同特征，比如来自同一个IP地址，且浏览器版本和操作系统版本相同，请求时间又相近等，那么就可以认为这些请求是来自于同一个客户端，那么多个页面访问也只算一个UV。共同特征的定义是由服务器方决定的。通常，用IP地址+其他特征共同来定义的情况较多。但此种度量方法无法解决以下问题，例如：多个人的计算机软硬件雷同，并且是一个公司或学校的人；多个人共用一个计算机等情况。

·通过Cookie鉴别。

当客户端第一次访问某个网站服务器时，网站服务器会给这个客户

端的计算机发出一个Cookie，通常放在这个客户端计算机的C盘当中。在这个Cookie中会分配一个独一无二的编号，这其中会记录一些访问服务器的信息，如访问时间、访问了哪些页面，等等。当你下次再访问这个服务器的时候，服务器就可以直接从你的计算机中找到上一次放进去的Cookie文件，并且对其进行一些更新，但那个独一无二的编号是不会变的。如果在一定时间内，服务器发现2个来访者对应的是一个编号，那么自然可以认为它来源于同一个来访者了，于是就计算1个UV。

使用Cookie的方法要比分析客户端HTTP请求头部信息更精确些。但也存在一些问题，比如：有的客户端为保证更高级别的安全，关闭了Cookie的功能；或者是有些客户端设置了在退出页面时自动删除Cookie，抑或你经常自己去手动删除Cookie，那么这个方法就不那么精确了。

因此，以上两个方法都只能得到近似的UV，而不是绝对精确的。

UV的度量相对IP和PV来说，不但麻烦，而且要开发比较复杂的程序系统才能得到期望的结果，因此，在Linux运维领域大家提及得较少，一般企业市场及运营人员可能会更多关注网站的UV。

4.4.5　IP、PV、UV的区别

针对该主题，下面以一个访问示例来讲解吧。

假设某城市的一个网吧里，有10个人都进入了www.etiantian.org 的网站，每个人平均访问了5个页面，但是这个网吧的对外出口是一个公网IP（注意：也可以配置多个IP出口，此处不计特殊情况），所以，对于etiantian网站来说，只会计算一个独立IP访问，但是因为网吧里有10人在访问www.etiantian.org 的网站，并且平均都访问了5次，因此，对于etiantian网站来说，PV数就是10×5=50个PV，而因为有10个人访问，就是10个不同的客户端访问，因此，UV（独立访客）为10。

那么，在此访问示例中，网站独立IP数为1个，PV数为50个，UV（独立访客）为10个。

通过上述结果，我们不难得出一个结论，一个网站的独立IP数量要比网站实际访问的PV数量小得多。通常情况下（国内互联网环境下），网站的UV数也会大于独立IP数。

PV数高说明访问的页面数多，但是不一定就代表来访者多；但PV数一定与来访者的数量成正比，不过，PV并不直接决定页面的真实来访者数量。比如在访问某网站时，一个人也可通过不断地刷新页面，制

造出非常高的PV数。PV数多，用户访问网站页面的总数量多，通常服务器的压力会大一些。

4.4.6 并发连接

1.网站并发连接

在面试过程中，Linux运维人员经常会被问到：你的公司网站最大并发是多少？

那么到底什么是并发？怎么理解并发呢？

笔者查阅了很多资料，但还是没有找到让人信服的确切说法。下面是网上的一些说法：

A种理解：网站服务器每秒能够接收的最大用户请求数。

B种理解：网站服务器每秒能够响应的最大用户请求数。

C种理解：网站服务器在单位时间内能够处理的最大连接数。

虽然A、B的理解占IT人员中的大多数，但是，按照老男孩的理解，C种理解更为准确一些。

列举一个单位时间段的例子说明：

我们去餐馆吃饭（如图4-6所示），餐馆里一共有10张桌，每张桌最多坐4个人同时吃饭，那么按一般人的理解，这个餐馆能够接收的并

发吃饭人数为10×4，即40个并发，其实这里就没有考虑时间问题，1秒并发可以是40个，10分钟内并发也是40个。因为这里还有一个因素，就是每个人吃饭时长的问题，如果平均每个人10分钟吃完，那么可以说10分钟内，这个餐馆的并发为40个，而不是每秒钟并发40个，因为，第一秒可以是40个人同时进来，但是第二秒就无人可进了（满员了），如果说10分钟并发是40个，下一个10分钟还是40个，第三个10分钟还可以是40个。这就是老男孩对并发的理解，即网站服务器在单位时间内能够处理的最大连接数。

再列举一个同一个时刻的示例：

高速公路每个方向都有两条车道（如图4-7所示），那么，同一时刻并发的车辆为两辆，并且并发可以永远为2，如果按秒计算，每秒的并发可能就有十几辆，这个例子和餐馆不同，因为高速路处理并发不需要处理时间。但是对于Web服务器来讲，是需要花费时间处理请求的，这个请求可能是1秒或数秒，因此说，并发不应该只是用户访问的请求数，而应是服务器同时处理的并发数，并且单位时间不一定是1秒，可能是一个连接处理周期内的连接数。

图4-6　餐馆并发图解

图4-7　高速车道并发图

对于网站服务器来说，所谓的并发就是单位时间内，服务器能够同时处理的最大连接数，因为有的请求1秒结束，有的请求可能10秒才结束（业务程序及配置不同），因此，网站并发不是客户端每秒的并发请求数，而是服务器在一段时间内（1秒或数秒内）可以处理的最大连接数，这个连接既包含正在建立的连接，也包含已经建立的连接。

例如：某网站的并发是5000。意味着单位时间内（理解为1秒或数秒内），正在处理的连接数，正在建立的连接数，加起来一共是5000个。

下面是国外学者对网站并发数的计算公式及参考说明：

Request Per Second+Simultaneous Browser connections+Thinking Time=Concurrent User

其中：

·Concurrent User表示网站并发用户总数。

·Request Per Second[RPS]表示每秒请求数（吞吐量）。

·Simultaneous Browser connections[SBC]表示并发浏览连接数。

·Thinking Time表示平均用户思考时间。

2.其他服务并发连接

（1）QPS（Query Per Second，每秒查询率）

每秒查询率QPS是用于衡量一个特定的查询服务器在规定时间内所处理流量多少的标准。运维工作中，DNS系统及数据库等服务的查询性能经常用每秒查询率来衡量。

（2）IOPS（Input/Output Operations Per Second）

IOPS即每秒进行读写（I/O）操作的次数，多用于数据库等场合，衡量随机访问的性能。存储端的IOPS性能和主机端的I/O是不同的，IOPS是指存储每秒可接受多少次主机发出的访问，主机的一次I/O需要多次访问存储才可以完成。例如，主机写入一个最小的数据块，也要经过“发送写入请求、写入数据、收到写入确认”三个步骤，也就是3个存储端访问。

4.4.7　常见企业网站排名及PV/IP访问量

表4-12给出了知名网站访问量的信息参考。

表4-12　知名网站访问量信息参考

网　　站	独立 IP 万 / 日	PV 数万 / 日	网站并发级别	机器数量
www.51cto.com	582 000	1 338 600	10 000	数十台
www.ganji.com	1 734 000	13 872 000	10 000 ~ 30 000	几百台
www.58.com	1 398 000	22 927 200	10 000 ~ 30 000	几百台
www.weibo.com	30 180 000	166 593 600	几十万	千台
www.taobao.com	46 620 000	489 510 000	几十万 ~ 百万	万台
www.jd.com	6 108 000	98 949 600	数万	千台
www.163.com	10 320 000	79 154 400	十万	千台
www.suning.com	930 000	7 254 000	10 000 ~ 30 000	百台

提示：以上数据是从第三方网站 http://alexa.chinaz.com/alexa_more.aspx 查找所得，查找时间大约2015年7月，不同的统计程序差别也很大，有一定误差，实际最高日访问量应该要比此表大，因为网站访问量也与节假日等有关，另外统计的误差和chinaz.com的统计方法有关，后面的最大并发及机器数量级别为作者根据访问量及业务类型估算而来，不代表网站的实际情况，仅供初学者参考。

4.4.8　有关网站度量Linux企业运维的常见面试题

常见的面试题如下：

1）请问你如何理解网站并发？

2）你们公司网站访问量是多少？是怎么计算的？

一定要理解IP、PV、并发量这3个点的知识，在回答时才能有的放矢，这三个点的多少决定面试时说多大的架构，对于没有经验的新手不能在介绍有几万PV的同时描述数十台的集群架构，这样就尴尬了。

关于网站访问指标的计算，可以考虑：

·运维部门的日志分析。

·开发在页面嵌入的JS程序（用于统计、收集、分析）。

·运营市场通过第三方公司提供的工具进行统计，例如GA统计。

4.5　WWW服务软件介绍

4.5.1　WWW软件全球使用排名参考

截止到作者写稿时为止，WWW应用软件存世量在数十种以上，不同的WWW应用软件在不同时间点的排名参考图4-8、图4-9和表4-13。

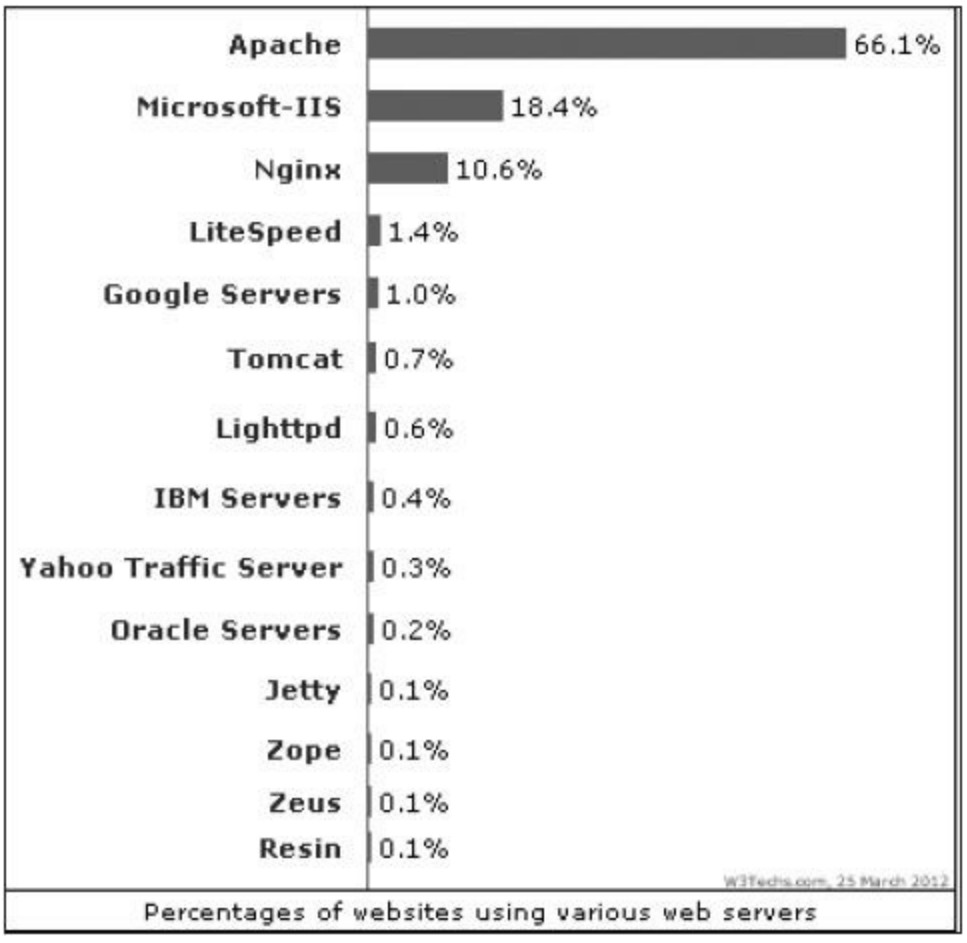


图4-8　2012-03-25日WWW软件的排名

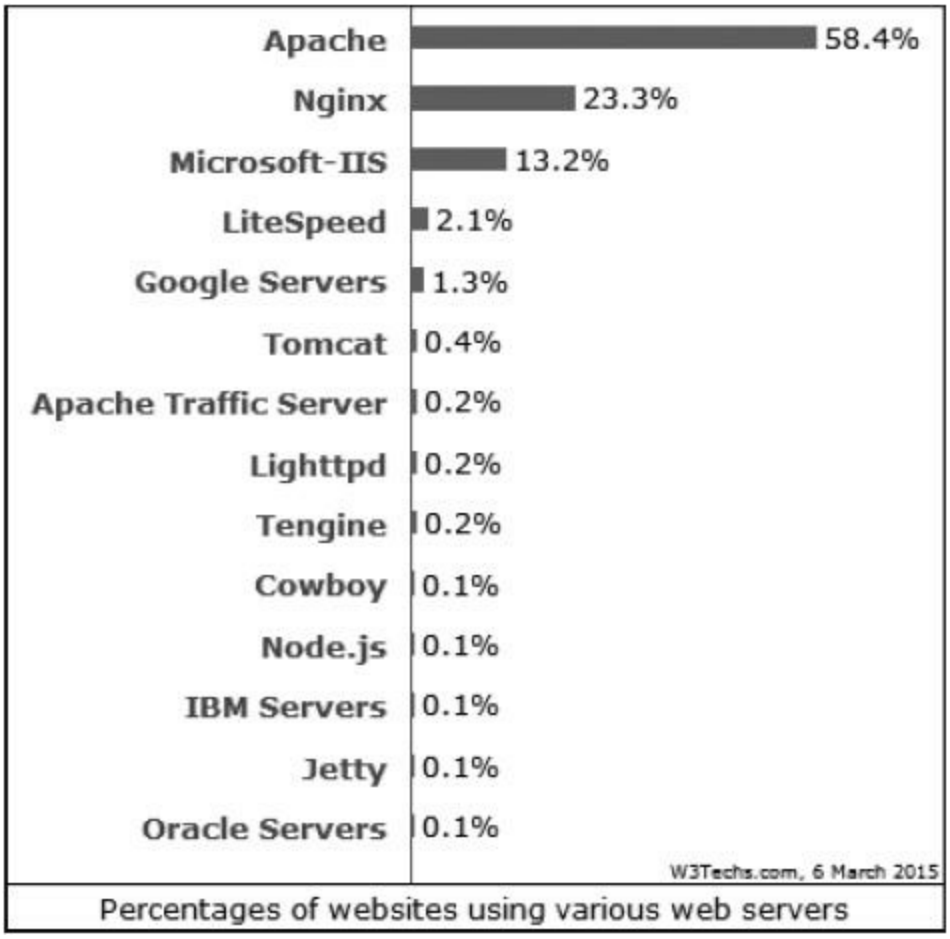


图4-9　3年后的2015-03-25日WWW软件的排名

从上述趋势变化[1]不难发现，Apache虽然份额最大，但是有逐年下降趋势，而Nginx这个后起之秀上升趋势显著。另外，Nginx的分支Tengine，也从看不见身影发展到逐渐占有一定份额了。

表4-13　排名变化对比

WWW 软件	2012 年	2015 年	趋势变化
Apache	66.6%	58.4%	下降近 9 个百分点
IIS	18.4%	13.3%	下降近 5 个百分点
Nginx	10.6%	23.3%	增加近 13 个百分点
Tengine	无排名	0.2%	增加近 0.2 个百分点
Tomcat	0.7%	0.4%	下降了 0.3 个百分点

[1] 上述数据来自 http://w3techs.com/technologies/overview/web_server/all。

4.5.2　当前互联网主流Web服务说明

1.当前互联网主流Web服务软件

常用来提供静态Web服务的软件有如下三种。

·Apache：这是中小型Web服务的主流，Web服务器中的老大哥。

·Nginx：大型网站Web服务主流，曾经Web服务器中的初生牛犊，现已长大。Nginx的分支Tengine（http://tengine.taobao.org/ ）目前也在飞速发展。

·Lighttpd：这是一个不愠不火的优秀Web软件，社区不活跃，静态解析效率很高。在Nginx流行前，它是大并发静态业务的首选，国内百度贴吧、豆瓣等众多网站都有Lighttpd奋斗的身影。

2.常用来提供动态服务的软件

·PHP（FastCGI）：大中小型网站都会使用，动态网页语言PHP程序的解析容器。它可配合Apache解析动态程序，不过，这里的PHP不是FastCGI守护进程模式，而是mod_php5.so（module）。也可配合Nginx解析动态程序，此时的PHP常用FastCGI守护进程模式提供服务。

·Tomcat：中小企业动态Web服务主流，互联网Java容器主流（如

jsp、do）。

·Resin：大型动态Web服务主流，互联网Java容器主流（如jsp、do）。

·IIS（Internet information services）：微软Windows下的Web服务软件（如asp、aspx）。

4.5.3 WWW静态程序服务软件Apache

Apache软件有几个重要的版本系列，分别为：Apache 1.3、Apache 2.0、Apache 2.2、Apache 2.4等，其中，Apache 1.3和Apache 2.0系列已经成为过去时，官方的网站也看不见其踪影了，目前主流的Apache为2.2系列，正在向Apache 2.4系列过渡。如果没有特别的要求，建议读者当下使用Apache 2.2系列。

官方地址：http://www.apache.org/ 。

4.5.4 WWW静态服务软件Nginx

Nginx的版本只有一个系列，但是版本更新很快，仅仅半年就有数个版本，这也可看出其社区的活跃程度，具体内容参见官方文档地址：http://www.nginx.org/ 及http://www.nginx.org/en/docs/ ，截至本书写稿时，其最新稳定版为1.6.2。

4.5.5 WWW动态服务软件Resin

Resin，官方号称是世界上最快的Web服务，是大型动态Web服务主流，为互联网Java程序的解析容器，百度、人人都曾用过Resin。

其发展的主要版本为Resin3.0、Resin3.1、Resin4.0，目前较多企业使用Resin 3.1，正在向Resin 4.0过渡。

图4-10为官方Resin、Nginx和HTTPD每秒请求数的速度对比。

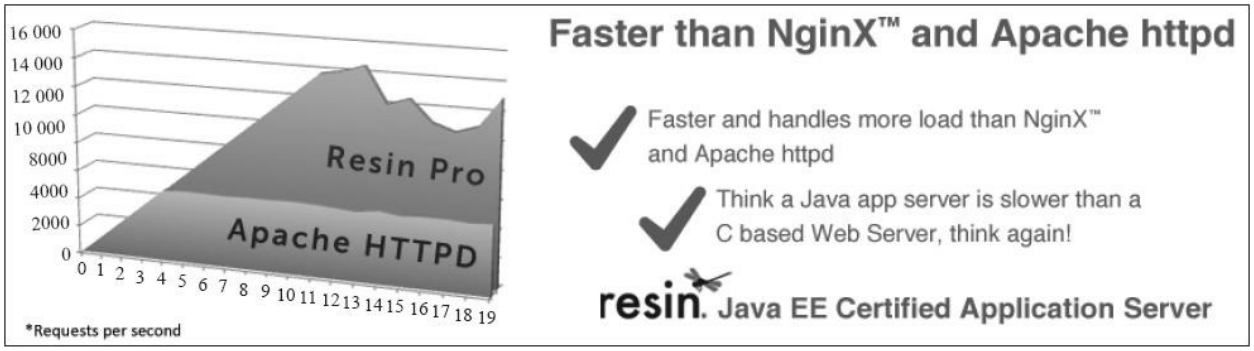


图4-10　Resin、Nginx和HTTPD每秒请求数的速度对比

Resin官方地址：

http://www.caucho.com/download/

http://caucho.com/resin-3.1/doc/

http://caucho.com/resin/doc/install-apache.xtp

4.5.6　WWW动态服务软件Tomcat

Tomcat一直是中小企业动态Web服务的主流，常用作解析Java程序的容器，其版本发展变化如表4-14。

表4-14　Tomcat版本变化对应表

Servlet/JSP Spec	Apache Tomcat version	Actual release revision	Minimum Java Version
3.0/2.2	7.0.x	7.0.26	1.6
2.5/2.1	6.0.x	6.0.35	1.5
2.4/2.0	5.5.x	5.5.35	1.4
2.3/1.2	4.1.x（archived）	4.1.40（archived）	1.3
2.2/1.1	3.3.x（archived）	3.3.2（archived）	1.1

目前企业使用的主流版本有Tomcat 6系列和Tomcat 7系列，官方也已经推出了更新的Tomcat 8.0系列。

Tomcat官方地址：

http://tomcat.apache.org/whichversion.html

http://tomcat.apache.org/

4.5.7 WWW动态服务软件PHP

PHP软件是大中小型网站程序前台页面开发的首选，存世的开源软件众多，也是中小企业网站开发的首选，它是动态网页语言PHP程序的解析容器。PHP的版本系列有PHP 5.2、PHP 5.3、PHP 5.4、PHP 5.5、PHP 5.6，其中最经典的版本为PHP 5.2系列，企业应用的主流版本可以说是百花争艳。

值得注意的是PHP提供解析的方式，在配合Apache解析动态程序时，它用的是mod_php5.so（module）模块；在配合Nginx解析动态程序时，常用FastCGI守护进程模式提供服务。

官方网站：http://php.net/ 。

4.6 本章重点回顾

（1）用户访问网站基本流程。

（2）DNS系统解析原理。

（3）HTTP协议通信原理，包括HTTP协议、请求报文、响应报文、状态码等相关知识。

（4）动态、静态概念特点及伪静态技术。

（5）动态转静态Web优化方案。

（6）IP、PV、UV的概念和区别。

（7）并发的概念理解。

（8）了解常用WWW服务软件特点，如Apache、Nginx、PHP（FastCGI）、Tomcat、Resin等。

4.7　本章知识相关面试考试题

（1）请描述DNS系统的解析原理。

（2）请描述HTTP协议的工作原理。

（3）请问你公司的网站访问量是多少？

（4）请说出HTTP状态码200、301、403、404、500、502、504代表的意义。

4.8 本章参考资料

·HTTP-Hypertext Transfer Protocol

http://www.w3.org/Protocols/

·HTTP/1.1

http://www.w3.org/Protocols/rfc2616/rfc2616.html

·统一资源标识符（URI）

http://zh.wikipedia.org/wiki/%E7%BB%9F%E4%B8%80%E8%B5%84

·统一资源定位符（URL）

http://zh.wikipedia.org/w/index.phptitle=%E7%BB%9F%E4%B8%80%

第5章　Nginx Web服务应用

本章主要对Nginx Web服务软件进行介绍，涉及Nginx的基础、特性、配置部署、优化，以及企业中的日常运维管理和应用。作为HTTP服务软件的后起之秀，Nginx与它的老大哥Apache相比有很多改进之处，比如，在性能上，Nginx占用的系统资源更少，能支持更多的并发连接（特别是静态小文件场景下），达到更高的访问效率；在功能上，Nginx不但是一个优秀的Web服务软件，还可以作为反向代理负载均衡及缓存服务使用；在安装配置上，Nginx更为方便、简单、灵活，可以说，Nginx是一个极具发展潜力的Web服务软件。下面就开始详细介绍Nginx Web服务软件。

5.1 Nginx介绍

5.1.1 Nginx是什么

如果你听说或使用过Apache软件，那么很快就会熟悉Nginx软件，与Apache软件类似，Nginx（“engine x”）是一个开源的，支持高性能、高并发的WWW服务和代理服务软件。它是由俄罗斯人Igor Sysoev开发的，最初被应用在俄罗斯的大型网站www.rambler.ru 上。后来作者将源代码以类BSD许可证的形式开源出来供全球使用。

Nginx因具有高并发（特别是静态资源）、占用系统资源少等特性，且功能丰富而逐渐流行起来。

在功能应用方面，Nginx不但是一个优秀的Web服务软件，还具有反向代理负载均衡功能和缓存服务功能。在反向代理负载均衡功能方面，它类似于大名鼎鼎的LVS负载均衡及Haproxy等专业代理软件，但是Nginx部署起来更为简单、方便；在缓存服务功能方面，它又类似于Squid等专业的缓存服务软件。

Nginx可以运行在UNIX、Linux、BSD、Mac OS X、Solaris，以及Microsoft Windows等操作系统中。随着Nginx在国内很多大型网站中的稳定高效运行，近两年它也逐渐被越来越多的中小型网站所使用。当前

流行的Nginx Web组合被称为LNMP或LEMP（即Linux Nginx MySQL PHP），其中LEMP里的E取自Nginx（“engine x”）。

Nginx的官方介绍见http://nginx.org/en/ 。

5.1.2 Nginx软件的使用排名

图5-1为Nginx软件在全球WWW服务软件中的使用排名。

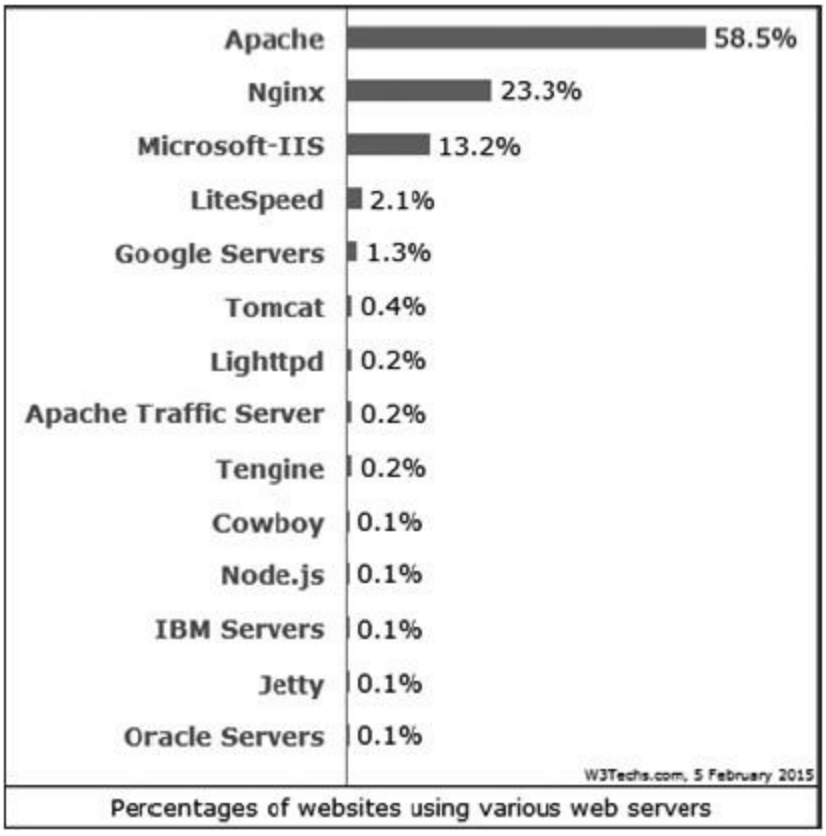


图5-1 Web Server排行 [1]

根据图5-1可以看出，到目前为止，Nginx已经成为全球使用数量排名第二的Web服务软件。其中，2012年Nginx市场占有率为12.6%，2013年市场占有率为14.5%，截至作者写作时，市场占有率达到了23.3%，可以看到Nginx的市场占有率有逐年快速递增的趋势！

图5-2为Web软件排行，其对应的数据如图5-3所示。

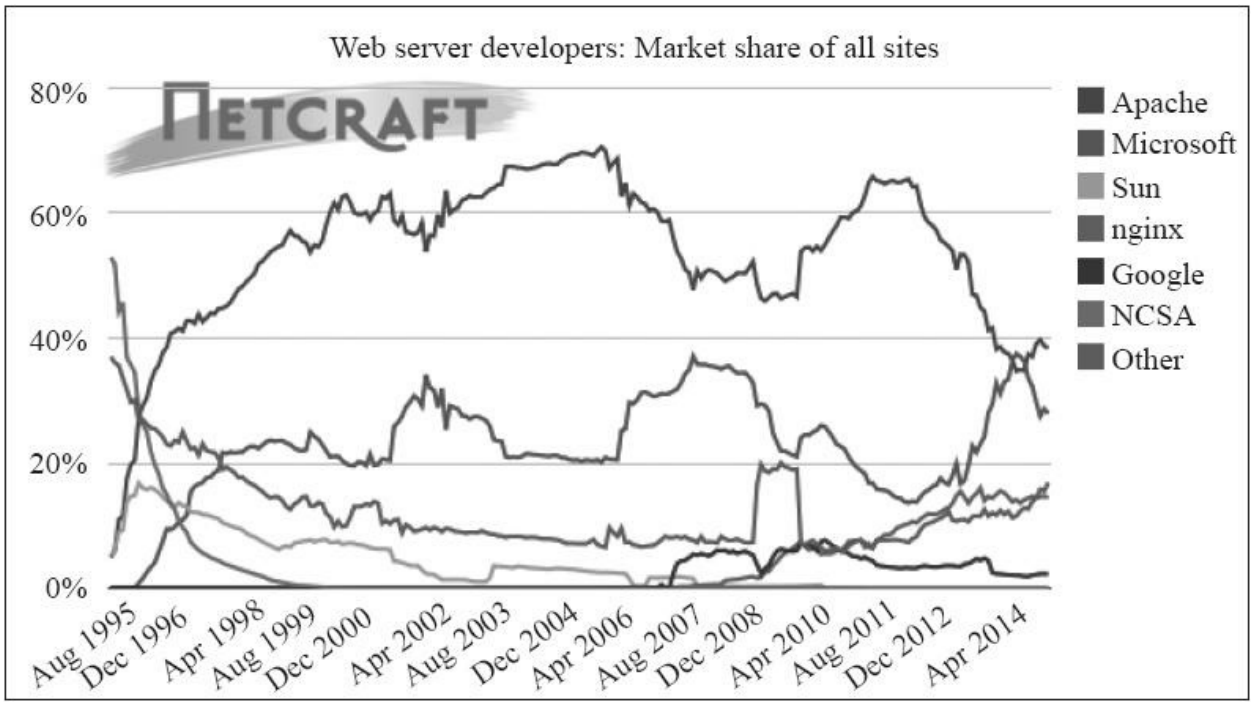


图5-2　Web软件排行 [2]

Developer	February 2015	Percent	March 2015	Percent	Change
Apache	342,480,920	38.77%	337,175,536	38.39%	-0.38
Microsoft	253,484,221	28.69%	245,496,533	27.95%	-0.74
nginx	130,093,899	14.73%	127,191,696	14.48%	-0.25
Google	20,238,057	2.29%	20,097,702	2.29%	-0.00

图5-3　Web软件排行数据

在不同的信息来源中，上面展示的各个指标数据会有一定的差别，但不管怎样都能看出Nginx的市场占有份额已经不容小觑了，作为IT从业人员，掌握Nginx软件的企业级应用是当务之急。

[1] Web Server的排行信息来源于http://w3techs.com/technologies/overview/web_server/all。

[2] 图5-2和图5-3的Web Server排行信息来源于http://news.netcraft.com/。

5.1.3 Nginx的重要特性及应用场合

1.Nginx的重要特性

Nginx在诞生之初的功能较弱，随着近几年各种功能特性逐渐增多并日趋成熟，很多大型网站也写了一些功能模块并开源出来，这使得Nginx变得越来越强大了。Nginx具备如下基本特性。

·可针对静态资源高速高并发访问及缓存。

·可使用反向代理加速，并且可进行数据缓存。

·具有简单负载均衡、节点健康检查和容错功能。

·支持远程FastCGI服务的缓存加速。

·支持FastCGI、Uwsgi、SCGI、Memcached Servers的加速和缓存。

·支持SSL、TLS、SNI。

·具有模块化的架构：过滤器包括gzip压缩、ranges支持、chunked响应、XSLT、SSI及图像缩放等功能。在SSI过滤器中，一个包含多个SSI的页面，如果经由FastCGI或反向代理处理，可被并行处理。

它所具备的其他WWW服务特性如下：

·支持基于名字、端口及IP的多虚拟主机站点。

·支持Keep-alive和pipelined连接。

·可进行简单、方便、灵活的配置和管理。

·支持修改Nginx配置，并且在代码上线时，可平滑重启，不中断业务访问。

·可自定义访问日志格式，临时缓冲写日志操作，快速日志轮询及通过rsyslog处理日志。

·可利用信号控制Nginx进程[1]。

·支持3xx-5xx HTTP状态码重定向。

·支持rewrite模块，支持URI重写及正则表达式匹配。

·支持基于客户端IP地址和HTTP基本认证的访问控制。

·支持PUT、DELETE、MKCOL、COPY及MOVE等较特殊的HTTP请求方法。

·支持FLV流和MP4流技术产品应用。

·支持HTTP响应速率限制。

·支持同一IP地址的并发连接或请求数限制。

·支持邮件服务代理。

 提示：以上信息参考自http://nginx.org/en/。

读者面试时可能需要解答如下Nginx HTTP服务器的特色及优点。

·支持高并发：能支持几万并发连接（特别是静态小文件业务环境）。

·资源消耗少：在3万并发连接下，开启10个Nginx线程消耗的内存不到200MB。

·可以做HTTP反向代理及加速缓存，即负载均衡功能，内置对RS节点服务器健康检查功能，这相当于专业的Haproxy软件或LVS的功能。

·具备Squid等专业缓存软件等的缓存功能。

·支持异步网络I/O事件模型epoll（Linux 2.6+）。

2.Nginx软件的主要企业功能应用

（1）作为Web服务软件

Nginx是一个支持高性能、高并发的Web服务软件，它具有很多优秀的特性，作为Web服务器，与Apache相比，Nginx能够支持更多的并

发连接访问，但占用的资源却更少，效率更高，在功能上也强大了很多，几乎不逊色于Apache。

（2）反向代理或负载均衡服务

在反向代理或负载均衡服务方面，Nginx可以作为Web服务、PHP等动态服务及Memcached缓存的代理服务器，它具有类似专业反向代理软件（如Haproxy）的功能，同时也是一个优秀的邮件代理服务软件（最早开发这个产品的目的之一就是作为邮件代理服务），但是Nginx的代理功能还是相对简单了些，特别是不支持TCP的代理（本书写作过程中Nginx 1.9.0已发布，已开始支持TCP的代理了，看来Nginx的代理功能也在逐渐增强）。

（3）前端业务数据缓存服务

在Web缓存服务方面，Nginx可通过自身的proxy_cache模块实现类Squid等专业缓存软件的功能。

Nginx的这三大功能（Web服务、反向代理或负载均衡服务、前端业务数据缓存服务）是国内使用Nginx的主要场景，特别是前两个。

[1] 见http://nginx.org/en/docs/control.html#logs。

5.2 Nginx Web服务

本章主要侧重于Nginx作为Web服务的应用。

5.2.1 Nginx Web服务介绍

Nginx安装简单，配置文件简洁，而且配置灵活。近两年来，Nginx在国内互联网领域的使用日趋火热，越来越多的网站开始使用，如淘宝、阿里、京东、小米、网易、新浪、赶集等。

Nginx作为Web服务器的主要应用场景包括：

·使用Nginx运行HTML、JS、CSS、小图片等静态数据（此功能类似Lighttpd软件）。

·Nginx结合FastCGI运行PHP等动态程序（例如使用fastcgi_pass方式）。

·Nginx结合Tomcat/Resin等支持Java动态程序（常用proxy_pass方式）。

5.2.2 Nginx与其他Web软件产品的对比说明

先来看看Apache软件的特点，如下：

·Apache 2.2版本非常稳定强大，据官方说，Apache 2.4版本性能更强。

·Prefork模式取消了进程创建开销，性能很高。

·处理动态业务数据时，因关联到后端的引擎和数据库，瓶颈不在Apache上。

·高并发时消耗系统资源相对多一些。

·基于传统的select模型，高并发能力有限。

·支持扩展库，可通过DSO、apxs方法编译安装额外的插件功能，不需要重新编译Apache。

·功能多，更稳定，更安全，插件也多。

·市场份额在逐年递减。

再来看看Nginx软件的特点，如下：

·基于异步网络I/O模型（epoll、kqueue）。

·具备支持高性能，高并发的特性，并发连接可达数万。

·对小文件（小于1MB的静态文件）高并发支持很好，性能很高。

·不支持类似Apache的DSO模式，扩展库必须编译进主程序（缺点）。

·进程占用系统资源比较低。

·支持Web、反向Proxy、Cache三大重点功能，并且都很优秀。

·市场份额在逐年快速增加。

最后是Lighttpd的特点，如下：

·基于异步网络I/O模型，性能、并发都与Nginx相近。

·扩展库是SO模式，比Nginx灵活。

·目前国内的使用率比较低，安全性没有Apache和Nginx好。

·通过插件（mod_secdownload）可实现文件URL地址加密（优点）。

·社区不活跃，市场份额较低。

5.2.3 Web服务产品性能对比测试

图5-4为各种Web服务产品在静态数据访问性能上的对比，从图中可看出，处理静态小文件（小于1MB）时，Nginx和Lighttpd比Apache更有优势，Nginx处理小文件的优势明显，Lighttpd综合最强。由于测试环境等差异，结论会有误差，仅供读者参考。

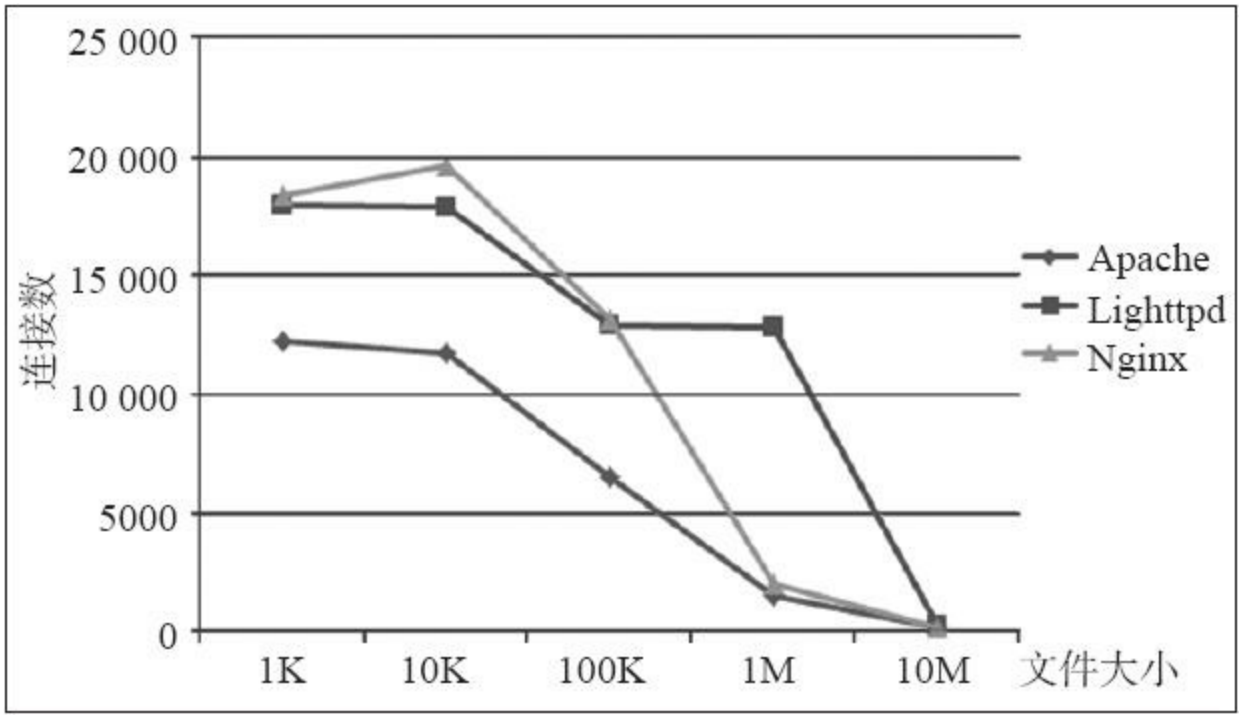


图5-4　主流Web服务静态数据性能对比图

图5-5是各类Web服务器在动态数据性能上的对比，从图中可以看出，在处理动态数据时，三者的差距不大，Apache更有优势一点。这是因为处理动态数据的能力取决于PHP（Java）和后端数据库的服务能

力，也就是说瓶颈不在Web服务器上。一般情况下普通PHP引擎支持的并发连接参考值为300~1000，Java引擎和数据库的并发连接参考值则为300~1500。业务场景及网站架构不同，并发连接数也会有上下浮动，这些数字仅供初学者参考。

5.2.4 为什么Nginx总体性能比Apache高

Nginx使用最新的epoll（Linux 2.6内核）和kqueue（freebsd）异步网络I/O模型，而Apache使用的是传统的select模型 。目前Linux下能够承受高并发访问的Squid、Memcached软件采用的都是epoll模型 。

处理大量连接的读写时，Apache所采用的select网络I/O模型比较低效。下面用两个通俗的比喻来解释Apache采用的select模型和Nginx采用的epoll模型之间的区别。

第一个比喻： 假设你在大学读书，住的宿舍楼有很多房间，你的朋友要来找你。select版宿管大妈 就会带着你的朋友到各房间挨个去找，直到找到你为止。而epoll版宿管大妈 会先记下每位入住同学的房间号，你的朋友来找你时，只需告诉你的朋友你住在哪个房间即可，不用亲自带着你的朋友满宿舍楼找人了。如果同时来了100个人，都要找自己住这栋楼的同学，select版和epoll版宿管大妈，谁的效率更高，就很明显了。

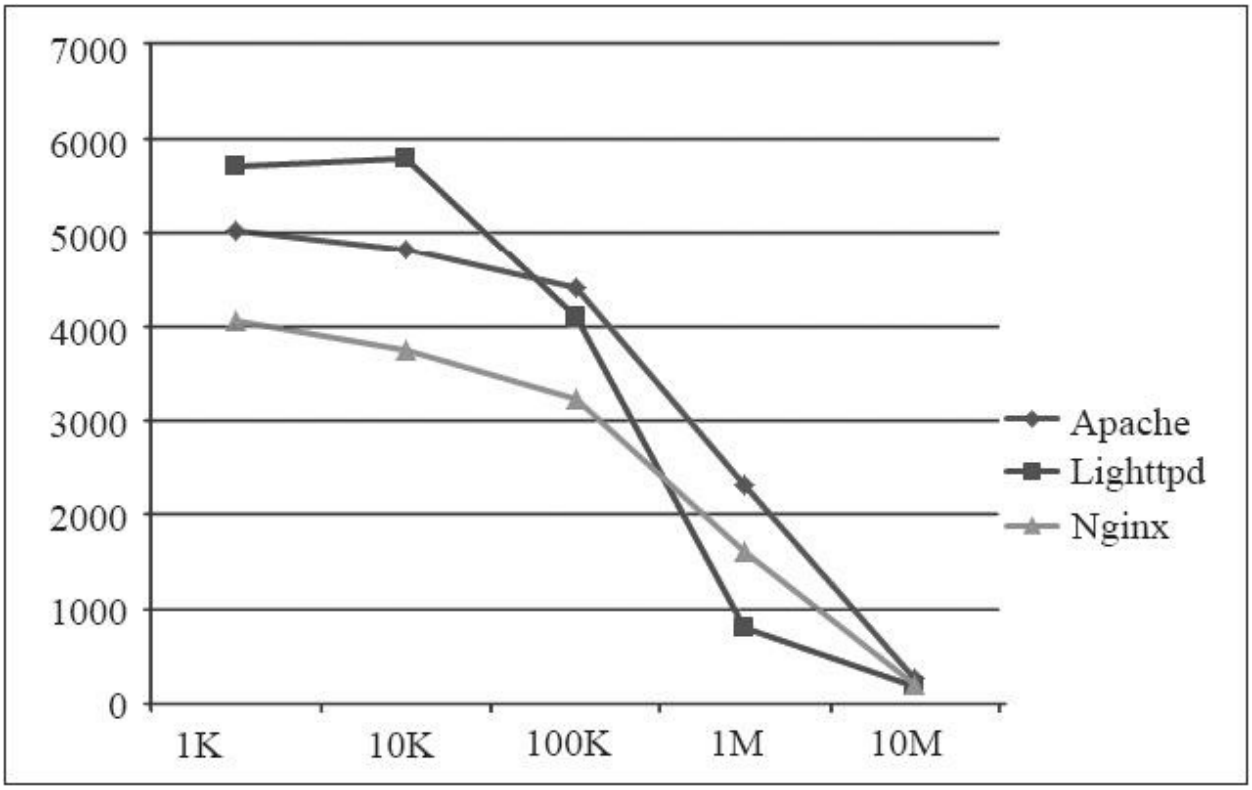


图5-5　主流Web动态数据性能对比图

第二个比喻： select的调用复杂度是线性的，即O（n）。举个例子，一个保姆照看一群孩子，如果把孩子是否需要尿尿比作网络I/O事件，select的作用就好比这个保姆挨个询问每个孩子“你要尿尿吗？”如果孩子回答是，保姆则把孩子领出来放到另外一个地方。当所有孩子询问完之后，保姆领着这些要尿尿的孩子去上厕所（处理网络I/O事件）。在epoll机制下，保姆不再需要挨个询问每个孩子是否需要尿尿。取而代之的是，如果孩子需要尿尿，他就自己主动站到事先约定好的地方，而保姆的职责就是查看事先约定好的地方是否有孩子。如果有小孩，则领着孩子去上厕所（网络事件处理）。因此，epoll的这种机制，能够高效地处理成千上万的并发连接，而且性能不会随着连接数增加而下降太

多。

Apache select和Nginx epoll的技术对比如表5-1所示。

表5-1　Apache select和Nginx epoll的技术对比

指　　标	select	epoll
性能	随着连接数的增加性能急剧下降。处理成千上万并发连接数时，性能很差	随着连接数的增加，性能基本上没有下降。处理成千上万并发连接时，性能很好
连接数	连接数有限制，处理的最大连接数不超过1024。如果要处理的连接数超过1024个，则需要修改FD_SETSIZE宏，并重新编译	连接数无限制
内在处理机制	线性轮询	回调callback
开发复杂性	低	中

限于篇幅，更多的select和epoll原理及深入区别请查询老男孩的博客（http://oldboy.blog.51cto.com ）。

5.2.5 如何正确选择Web服务器

虽然国内很多人都在使用Nginx，但是Apache、Lighttpd这两个Web服务器同样非常强大且实用，尤其是Apache，到目前为止仍是全球使用最广泛的Web服务软件。

在实际工作中，我们需要根据业务需求来选择合适的业务服务软件，有关Web服务，老男孩的选择建议如下。

·静态业务：若是高并发场景，尽量采用Nginx或Lighttpd，二者首选Nginx。

·动态业务：理论上采用Nginx和Apache均可，建议选择Nginx，为了避免相同业务的服务软件多样化，增加额外维护成本。动态业务可以由Nginx兼做前端代理，再根据页面元素的类型或目录，转发到后端相应的服务器进行处理。

·既有静态业务又有动态业务：采用Nginx。

此外，如果并发不是很大，又对Apache很熟悉，采用Apache也是可以的，Apache 2.4版本也很强大，并发连接数也有所增加。总的来说，在满足需求的前提下，首先选择自己最擅长的软件，若发现了更好的软件，可在掌握新软件之后逐步替换。虽然动态和静态业务都倾向于选择

Nginx，但是大前提是自己要熟练掌握Nginx。切记，在工作中不要盲目选择软件，这可能最终会导致自己无法控制局面，从而给企业带来灾难性的损失。

5.3 编译安装Nginx

到目前为止，还未发现操作系统ISO镜像自带，或者默认情况下yum可以直接通过rpm包方法安装Nginx的情况。如果要使用yum安装Nginx，则需要配置epel yum源或去官方寻找。接下来重点讲Nginx的编译方式安装，如果是大规模的安装，可以先根据自身的业务需求定制好rpm包，然后放到yum仓库里通过yum来安装，有关rpm包的定制及yum仓库的搭建，请看老男孩的博客：http://user.qzone.qq.com/49000448/blog/1426987479 。

1.安装Nginx所需的pcre库

pcre的全称为perl compatible regular expressions，中文译为“perl兼容正则表达式”，官方站点为http://www.pcre.org/ ，安装pcre库是为了使Nginx支持具备URI重写功能的rewrite模块，如果不安装pcre库，则Nginx无法使用rewrite模块功能，Nginx的rewrite模块功能几乎是企业应用必须的。安装pcre库的过程如下。

1）查看当前Linux系统环境，命令如下：

```
[root@www ~]# cat /etc/redhat-release
CentOS release 6.6 (


Final)
```

```
[root@www ~]# uname -r
2.6.32-504.el6.x86_64
[root@www ~]# uname -m
x86_64#<==64位系统
```

2）采用yum安装方式安装pcre（老男孩推荐的安装方法），命令如下：

```
[root@www ~]# yum install pcre pcre-devel -y
[root@www ~]# rpm -qa pcre pcre-devel
```

---

yum安装操作后检查安装结果，命令如下：

```
[root@www ~]# rpm -qa pcre pcre-devel
pcre-devel-7.8-6.el6.x86_64
pcre-7.8-6.el6.x86_64提示：


yum安装的


pcre版本有些低，不过一般情况不影响使用
```

也可采用编译方法安装pcre，比较复杂不推荐。

2.安装Nginx

Nginx的英文官方网站是http://nginx.org/ ，在这里可以查看Nginx的各个软件版本信息。Nginx软件有三种版本：稳定版、开发版和历史稳

定版。开发版更新较快，包含最新的功能和bug的修复，但同时也可能会遇到新的bug，开发版的更新一旦稳定下来，就会被纳入稳定版中。但是，有些新功能不会被加到旧的稳定版中去。稳定版本的更新较慢，但是软件bug也会较少，可以作为企业生产环境的首选，因此通常建议选择使用稳定版。当然，在实际工作中，选择稳定版时，尽量避免使用最新的版本，选择比已出来的最新版晚6~10个月的版本比较好。本书中考虑到从写书到出版会有较长时间间隔，因此选择的是写书期间推出的最新稳定版（nginx-1.6.3 stable）。

Nginx的安装非常简单，具体的操作过程及屏幕输出如下。

（1）检查并安装Nginx基础依赖包pcre-devel、openssl-devel

要想正确安装Nginx，首先必须安装好pcre-devel、openssl-devel包，因此先要检查这些Nginx基础依赖包是否安装，操作命令如下：

```
[root@www ~]# rpm -qa pcre-devel pcre
pcre-devel-7.8-6.el6.x86_64       #<==pcre的


devel包已经安装！



pcre-7.8-6.el6.x86_64             #<==pcre包已经安装！



[root@www ~]# rpm -qa openssl-devel openssl
openssl-1.0.1e-30.el6.x86_64       #<==这里没有
```

```
opensll“

devel”字符串的包
```

提示：名称中带有“devel”字符串的软件包是必须要安装的。

（2）安装openssl-devel

Nginx在使用HTTPS服务的时候要用到此模块，如果不安装openssl相关包，安装Nginx的过程中会报错。安装openssl-devel的命令及检查命令如下：

```
[root@www ~]# yum install -y openssl openssl-devel
[root@www ~]# rpm -qa openssl openssl-devel
openssl-devel-1.0.1e-30.el6.8.x86_64
openssl-1.0.1e-30.el6.8.x86_64
```

（3）开始安装Nginx

操作命令如下：

```
mkdir -p /home/oldboy/tools
#-p选项表示不提示目录是否存在，循环向下创建所有层级目录，如果存在就会忽略。

#建立一个工具目录来固定存放安装的各种软件，这里老男孩使用个人用户

oldboy家目录下的
```

tools
cd /home/oldboy/tools
#进入

/home/oldboy/tools目录

wget -q http:

//nginx.org/download/nginx-1.6.3.tar.gz
#下载软件包，进入

http:

//nginx.org/download/ 复制对应版本的链接地址。提示，如果发现

Nginx软件下载地址已不可用，可能版本已更新，可去官方地址

http:

//www.nginx.org下载。

```
ls -l nginx-1.6.3.tar.gz
useradd nginx -s /sbin/nologin -M
tar xf nginx-1.6.3.tar.gz
cd nginx-1.6.3
./configure  --user=nginx --group=nginx --prefix=/application/nginx-1.6.3/ --with-ht
make
make install
ln -s /application/nginx-1.6.3 /application/nginx
#这条
```

ln命令的意义十分深远重大。这可是生产环境的经验。

#将

Nginx安装路径通过软链接的方式更改为

/application/nginx/，方便人员使用。

#安装时指定版本号路径是为了便于查看区分当前使用的

Nginx版本，也方便以后升级。

#内部人员使用路径

/application/nginx/。

#当

Nginx软件升级编译成带新版本号的版本后，删除原来软链接，再重新建立新的到

/application/nginx/的软链接就好。

#程序中如果有引用

Nginx路径的地方，不需要做任何更改，因为升级后访问路径还是

```
/application/nginx/
cd ../
```

检查链接及目录状态，命令如下：

```
ll /application|grep nginx
ls -l /application/nginx/
```

编译Nginx软件时，可以使用./configure--help查看相关参数帮助。下面是本次编译时指定的参数及简单说明：

```
--prefix=PATH  set installation prefix                    #←设置安装路径。

--user=USER    set non-privileged user for worker processes #←进程用户权限。

--group=GROUP  set non-privileged group for worker processes #←进程用户组权限。

--with-http_stub_status_module  enable ngx_http_stub_status_module
                                                  #←激活状态信息。

--with-http_ssl_module           enable ngx_http_ssl_module #←激活

ssl功能。
```

提示：Nginx的大部分模块功能都会默认编译到软件中，不需要单独指定编译参数。

下面是安装的操作过程。

```
[root@www ~]# mkdir -p /home/oldboy/tools
[root@www ~]# cd /home/oldboy/tools
[root@www tools]# wget -q http:

//nginx.org/download/nginx-1.6.3.tar.gz
```

```
[root@www tools]# ls -l nginx-1.6.3.tar.gz
-rw-r--r-- 1 root root 804164 11月


 23 15:


26 nginx-1.6.3.tar.gz
[root@www tools]# useradd nginx -s /sbin/nologin -M
[root@www tools]# tar xf nginx-1.6.3.tar.gz
[root@www tools]# cd nginx-1.6.3
[root@www nginx-1.6.3]#./configure  --user=nginx --group=nginx --prefix=/applicatior
[root@www nginx-1.6.3]# make
[root@www nginx-1.6.3]# make install
[root@www nginx-1.6.3]# ln -s /application/nginx-1.6.3 /application/nginx
[root@www nginx-1.6.3]# ls -l /application/nginx/总用量


 16
drwxr-xr-x. 2 root root 4096 8月


  13 11:


19 conf
drwxr-xr-x. 2 root root 4096 8月


  13 11:


19 html
drwxr-xr-x. 2 root root 4096 8月


  13 11:


19 logs
drwxr-xr-x. 2 root root 4096 8月


  13 11:


19 sbin
```

在安装环节中，如果遇到如下错误：

```
------------------------------------------------------------------------
./configure:


 error:


 SSL modules require the OpenSSL library.
You can either do not enable the modules,


 or install the OpenSSL library
into the system,


 or build the OpenSSL library statically from the source
with nginx by using --with-openssl=<path> option.
------------------------------------------------------------------------
```

则解决方法是执行命令：yum install openssl openssl-devel-y。

到此，Nginx的安装工作就完成了。

3.启动并检查安装结果

安装完Nginx后，并不能直接对外提供服务，需要先启动Nginx服务才行，具体操作如下。

（1）启动前检查配置文件语法

命令如下：

```
[root@www tools]# /application/nginx/sbin/nginx -t
nginx:
```

```
 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
```

提示：在启动服务前检查语法非常重要，可以防止因配置错误导致网站重启或重新加载配置等对用户的影响。

（2）启动Nginx服务

启动命令如下：

```
[root@www tools]# /application/nginx/sbin/nginx
```

（3）查看Nginx服务对应的端口是否成功启动

命令如下：

```
[root@www tools]# lsof -i :


80
COMMAND   PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nginx   10586  root   6u   IPv4  21304      0t0  TCP *:


http  (


LISTEN)
```

```
nginx   10587 nginx   6u   IPv4  21304      0t0  TCP *:

http  (

LISTEN)
```

也可以通过netstat-lnt|grep 80查看，命令如下：

```
[root@www tools]# netstat -lnt|grep 80
tcp      0      0 0.0.0.0:

80          0.0.0.0:

*              LISTEN
```

（4）检查Nginx启动的实际效果

在Windows下通过浏览器检测的方式如下。

打开浏览器输入http://10.0.0.8（10.0.0.8为安装Nginx服务器的IP地址），然后回车，如果看到如图5-6所示的内容，就表示Nginx已经启动了。

图5-6　成功访问Nginx服务Web界面图

在Linux下可使用如下wget命令检测。

```
[root@www tools]# wget 127.0.0.1
--2015-04-03 14:


24:


44--  http:


//127.0.0.1/正在连接


 127.0.0.1:


80... 已连接。


已发出


 HTTP 请求，正在等待回应


... 200 OK长度:
```

```
612 [text/html]正在保存至：

 “

index.html”

100%[======================================>] 612         --.-K/s   in 0s
2015-04-03 14：

24：

44 （

71.6 MB/s）

 - 已保存

 “

index.html”

 [612/612]）
```

也可使用curl命令检测，如下：

```
[root@www tools]# curl 127.0.0.1
<!

DOCTYPE html>
<html>
```

```
<head>
<title>Welcome to nginx!


</title>
<style>
    body {
        width:


 35em;


        margin:


 0 auto;


        font-family:


 Tahoma,


 Verdana,


 Arial,


 sans-serif;


    }
</style>
</head>
<body>
<h1>Welcome to nginx!


</h1>
<p>If you see this page,


 the nginx web server is successfully installed and
```

```
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http:


//nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http:


//nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

以上3种方法都可以检测Nginx的安装及浏览是否正常。


提示：10.0.0.8为服务器的IP地址，也可以通过netstat-lnt|grep 80、lsof-i：80检查Nginx服务器端口（默认为80）来判断Nginx是否已启动，或者通过ps-ef|grep nginx检查Nginx服务进程来判断，当然最稳妥的方法还是通过URL地址来检查。在后面的监控服务中还会提到这个问题。

4.Nginx启动的疑难杂症汇总

问题1：启动Nginx时有如下报错“nginx：[emerg]getpwnam（"nginx"）failed”。

解答：这是因为没有对应的Nginx服务用户，执行useradd nginx-s/sbin/nologin-M创建Nginx用户即可。为了让读者理解问题，重现上述错误过程，命令如下：

```
[root@www tools]# pkill nginx
[root@www tools]# userdel nginx
[root@www tools]# /application/nginx/sbin/nginx
nginx:


 [emerg] getpwnam(


"nginx")


 failed
[root@www tools]# useradd nginx -s /sbin/nologin -M
[root@www tools]# /application/nginx/sbin/nginx
```

提示：重现错误过程排错，是优秀运维人员的重要本领之一，因为企业场景遇到问题，经常以临时性快速解决为主，线下重现来根治，这时就需要模拟重现错误了。

问题2：编译安装pcre编译软件时，gcc不全导致报错（本文使用yum安装不存在此问题）。

报错信息如下：

```
[root@gjlin2 pcre-8.30]# make && make install
make  all-am
make[1]:


 Entering directory `/home/gjlin/tools/pcre-8.30'
 CXX    pcrecpp.lo
libtool:


 compile:


 unrecognized option `-DHAVE_CONFIG_H'
libtool:
```

```
 compile:


 Try `libtool --help' for more information.
make[1]:


 *** [pcrecpp.lo] 错误


 1
make[1]:


 Leaving directory `/home/gjlin/tools/pcre-8.30'
make:


 *** [all] 错误


2
```

解答：执行“yum-y install gcc-c++”命令安装gcc-c++依赖包。

问题3：如何查看Nginx编译时的参数？

解答：可采用如下命令查看：

```
[root@www conf]# ../sbin/nginx -V
nginx version:


 nginx/1.6.3
built by gcc 4.4.7 20120313 (


Red Hat 4.4.7-11)
```

```
  (

GCC)


TLS SNI support enabled
configure arguments:


 --user=nginx --group=nginx --prefix=/application/nginx-1.6.3 --with-http_stub_statu
```

问题4：浏览器、wget或curl等软件访问不了Nginx页面。

解答：此类问题的排查思路又分在Nginx服务器端排查和在客户端排查。服务器端的排查过程如下。

首先关闭SELinux，命令如下：

```
 [root@www tools]# setenforce 0       #<==这是临时关闭


selinux的方法。


setenforce:


 SELinux is disabled       #<==如果之前就已经是关闭状态，则显示


disabled。


[root@www tools]# grep SELINUX=disabled /etc/selinux/config
#<==永久关闭的方法就是编辑
```

/etc/selinux/config，然后将

SELINUX= enforcing改为

SELINUX=disabled地点，前文讲过，此处不再赘述，仅仅检查结果

SELINUX=disabled

然后检查防火墙。命令如下：

```
[root@www tools]# /etc/init.d/iptables stop
#←这是关闭防火墙的命令，如果是有外网

IP的生产环境，请允许

80端口的访问，而不是关闭防火墙，

#允许命令如下：

iptables -I INPUT -p tcp --dport 80 -j ACCEPT
Flushing firewall rules：

 [  OK  ]
Setting chains to policy ACCEPT：

 filter [  OK  ]
Unloading iptables modules：

 [  OK  ]
[root@www tools]# chkconfig iptables off
#←非正式环境可以禁止防火墙开机自启动，便于学习调试
```

```
Nginx服务。等掌握防火墙技术再开启才好。


[root@www tools]# /etc/init.d/iptables status
#←查看


iptables的当前状态


Firewall is stopped.
```

再到服务器本地检查端口、进程和URL。此过程又分4步。

一是确认端口80是否存在，命令如下：

```
[root@www tools]# netstat -lnt|grep 80
tcp      0      0 : : :


80                : : :


*                    LISTEN
```

二是查看是否有http进程存在，命令如下：

```
[root@www tools]# ps -ef|grep nginx
root    10637  1      0 14:


29        00:


00:


00 nginx:
```

```
 master process /application/
nginx/sbin/nginx
nginx    10638  10637  0 14:


29        00:


00:


00 nginx:


 worker process
root     10668  1075   0 14:


40 pts/0  00:


00:


00 grep nginx提示：默认情况仅有一个


Nginx进程
```

三是在服务器本地进行wget http://10.0.0.8测试。

注意：如果第一步和第二步都不符合要求，就不用进行第三步骤了。直接执行步骤4即可。

这里把本地服务器当作客户端来模拟用户检查HTTP服务，此处的检查比步骤一和步骤二的更准确，同时可以排除防火墙的干扰，如果可

正常访问，说明Nginx服务没问题，那么就是网络或防火墙等的问题了。

第四步是查看Nginx的错误日志，确定是否有特殊异常。

如果以上四步全都检查了，基本上问题就可迎刃而解了，如果是步骤一和步骤二异常，则可以查看Web错误日志，获取信息。例如，刚才重现的错误在错误日志里的提示如下：

```
[root@www tools]# cat /application/nginx/logs/error.log
2015/07/03 14:


29:


20 [emerg] 10628#0:


 getpwnam (


"nginx")


 failed
```

 注意：遇到问题时要在第一时间看屏幕返回的提示和Nginx服务的error.log，获取有效信息以便解决问题。这是运维工程师必须具备的基本技能！在实际教学工作中，发现有不少学生，一遇到问题，无论大小，不经过思考就到处去问。这样的学习方法非常不好，养成看屏幕输出和错误日志的习惯才是学习的正确方法，经过思考后还无法解决问

题，再去请教别人才是最好的。

下面介绍客户端排查的思路。

第一步，在客户端上ping服务器端IP，命令如下。

```
ping 10.0.0.8            ←排除物理线路问题影响
```

第二步，在客户端上telnet服务器端IP、端口，命令如下：

```
telnet 10.0.0.8 80       ←排除防火墙等得影响
```

第三步，在客户端使用wget命令检测，如下：

```
wget 10.0.0.8（

curl -I 10.0.0.8）←模拟用户访问，排除

http服务自身问题，根据输出再排错
```

 提示：以上三步是客户端访问网站异常排查的重要三部曲。

5.小试牛刀：部署一个Web站点

Nginx的默认站点目录是Nginx安装目录/application/nginx/下的html目录，这可以从Nginx主配置文件/application/nginx/conf/nginx.conf中查到，内容如下：

```
[root@www ~]# grep html /application/nginx/conf/nginx.conf
            root   html;

                    ←

 这个就是默认的站点目录

html，就

                                    是

/application/nginx/html
            index  index.html index.htm;

      ←这是站点的首页文件
```

提示：此处的/application/nginx/是/application/nginx-1.6.3的软链接。

此时，如果要部署网站业务数据，只需把开发好的程序全部放到/application/nginx/html目录下面即可。

这里进入/application/nginx/html下，删除掉Nginx默认的首页index.html（原始内容为Welcome to Nginx等），然后新建立一个index.html，加入如下网页内容，并保存：

```
<html>
<head><title>oldboy,



s Nginx server blog.</title></head>
<body>
Hi,



I am oldboy. My blog address is
<a href="http:



//oldboy.blog.51cto.com">http:



//oldboy.blog.51cto.com</a>
</body>
</html>
```

操作过程如下：

```
[root@www ~]# cd /application/nginx/html/
[root@www html]# rm -f index.html
[root@www html]# vi index.html           #<==切换到编辑模式下，加入如下内容



<html>
<head><title>oldboy,



s Nginx server blog.</title></head> <body> Hi,



I am oldbo
y. My blog address is
<a href="http:
```

```
//oldboy.blog.51cto.com">http:


//oldboy.blog.51cto.com</a>
</body>
</html>
```

切换到命令模式保存文件后查看，命令如下：

```
[root@www html]# cat index.html
<html>
<head><title>oldboy,


s Nginx server blog.</title></head> <body> Hi,


I am oldboy. My blog address is
<a href="http:


//oldboy.blog.51cto.com">http:


//oldboy.blog.51cto.com</a>
</body>
</html>
```

此时，打开浏览器输入http://10.0.0.8，然后回车，应该可以看到如图5-7所示的内容。

图5-7　简单Web站点内容

我们可以手工编写HTML程序，创建类似的更多更复杂的网站程序。看到这里，有些读者可能不以为然，话说15年前的老男孩就是用纯手工敲HTML程序，创建了大量的网站代码。如今的互联网中，这种HTML纯静态的网站已经不能满足需求了，也因此产生了ASP、PHP、JSP、Java等语言。在后面的章节中会逐步深入说明相关环境的部署，此处不必纠结，跟着本书内容走即可，有关动静态的介绍请参考本书前面章节。

5.4 Nginx技术的深入剖析

5.4.1 Nginx软件功能模块说明

Nginx软件之所以强大，是因为它具有众多的功能模块，下面列出了企业常用的重要模块。

（1）Nginx核心功能模块（Core functionality）

Nginx核心功能模块负责Nginx的全局应用，主要对应主配置文件的Main区块和Events区块区域，这里有很多Nginx必须的全局参数配置。有关核心功能模块的详细信息，请看官网，地址为：http://nginx.org/en/docs/ngx_core_module.html 。

（2）标准的http功能模块集合

这些标准的http功能模块，虽然不是Nginx软件所必需的，但都是很常用的，因此绝大部分默认情况都会自动安装到Nginx软件中（见表5-2），不建议读者擅自改动，保留软件的默认配置就好，除非你明确知道你在做什么，有什么额外影响。

在生产环境中，配置、调整及优化Nginx软件，主要就是根据这些模块的功能修改相应的参数来实现的。通过官方地

址http://nginx.org/en/docs/ 可以查看到上述及更多模块的详细使用帮助。

表5-2　企业场景常用的Nginx http功能模块汇总

Nginx http 功能模块	模块说明
ngx_http_core_module	包括一些核心的 http 参数配置，对应 Nginx 的配置为 HTTP 区块部分
ngx_http_access_module	访问控制模块，用来控制网站用户对 Nginx 的访问
ngx_http_gzip_module	压缩模块，对 Nginx 返回的数据压缩，属于性能优化模块
ngx_http_fastcgi_module	FastCGI 模块，和动态应用相关的模块，例如 PHP
ngx_http_proxy_module	proxy 代理模块
ngx_http_upstream_module	负载均衡模块，可以实现网站的负载均衡功能及节点的健康检查
ngx_http_rewrite_module	URL 地址重写模块
ngx_http_limit_conn_module	限制用户并发连接数及请求数模块
ngx_http_limit_req_module	根据定义的 key 限制 Nginx 请求过程的速率
ngx_http_log_module	访问日志模块，以指定的格式记录 Nginx 客户访问日志等信息
ngx_http_auth_basic_module	Web 认证模块，设置 Web 用户通过账号、密码访问 Nginx
ngx_http_ssl_module	ssl 模块，用于加密的 http 连接，如 https
ngx_http_stub_status_module	记录 Nginx 基本访问状态信息等的模块

5.4.2 Nginx的目录结构说明

为了让大家更清晰地了解Nginx软件的全貌，有必要介绍下Nginx安装后整体的目录结构及文件功能。可执行tree/application/nginx/命令来查看，其内容如下：

```
[root@www ~]# tree /application/nginx/
                          #<==如果


tree命令找不到需要


yum intall tree -y安装



/application/nginx/
|-- client_body_temp
|-- conf                    #这是


Nginx所有配置文件的目录，极其重要



|   |-- fastcgi.conf          #fastcgi相关参数的配置文件



|   |-- fastcgi.conf.default     #fastcgi.conf的原始备份



|   |-- fastcgi_params          #fastcgi的参数文件



|   |-- fastcgi_params.default
|   |-- koi-utf
```

```
|   |-- koi-win
|   |-- mime.types          #媒体类型，前面章节讲解过


|   |-- mime.types.default
|   |-- nginx.conf          #这是
```

Nginx默认的主配置文件

```
|   |-- nginx.conf.default
|   |-- scgi_params          #scgi相关参数文件，一般用不到


|   |-- scgi_params.default
|   |-- uwsgi_params               #uwsgi相关参数文件，一般用不到


|   |-- uwsgi_params.default
|   `-- win-utf
|-- fastcgi_temp               #fastcgi临时数据目录


|-- html     #这是编译安装时
```

Nginx的默认站点目录，前面已说明，类似

Apache的默认站点

htdocs目录

```
|   |-- 50x.html     #     错误页面优雅替代显示文件，例如：出现
```

502错误时会调用此页面

```
    #     error_page   500 502 503 504  /50x.html;
```

```
|   `-- index.html     #     默认的首页文件，在实际环境中，大家习惯用（注意不是必须）
```

index.html、

index.php、

index.jsp来做网站的首页文件。首页文件名字是在

nginx.conf中事先定义好的。

```
|-- logs      #这是
```

Nginx默认的日志路径，包括错误日志及访问日志

```
|   |-- access.log      #     这是
```

Nginx的默认访问日志文件，使用

tail -f access.log，

可以实时观看网站用户访问情况信息

```
|   |-- error.log      #     这是
```

Nginx的错误日志文件，如果

Nginx出现启动故障等问题，一定要看看这个错误日志

```
|   `-- nginx.pid      #     Nginx的
```

pid文件，

Nginx进程启动后，会把所有进程的

ID号写到此文件

```
|-- proxy_temp       #临时目录

|-- sbin      #这是
```

Nginx命令的目录，如

Nginx的启动命令

```
nginx
|   `-- nginx      #Nginx的启动命令

nginx
|-- scgi_temp      #临时目录

`-- uwsgi_temp      #临时目录

9 directories,

 21 files
```

 提示：所有结尾为default的文件都是备份文件，其他未做注释的目录，为老男孩在多年的生产环境中较少用到的目录，因此建议初学者也不要理会，学习要抓重点，先把这些常用的都学会再研究其他的。当然，大家如果有需求可以查阅相关文档进一步了解。

5.4.3 Nginx主配置文件nginx.conf

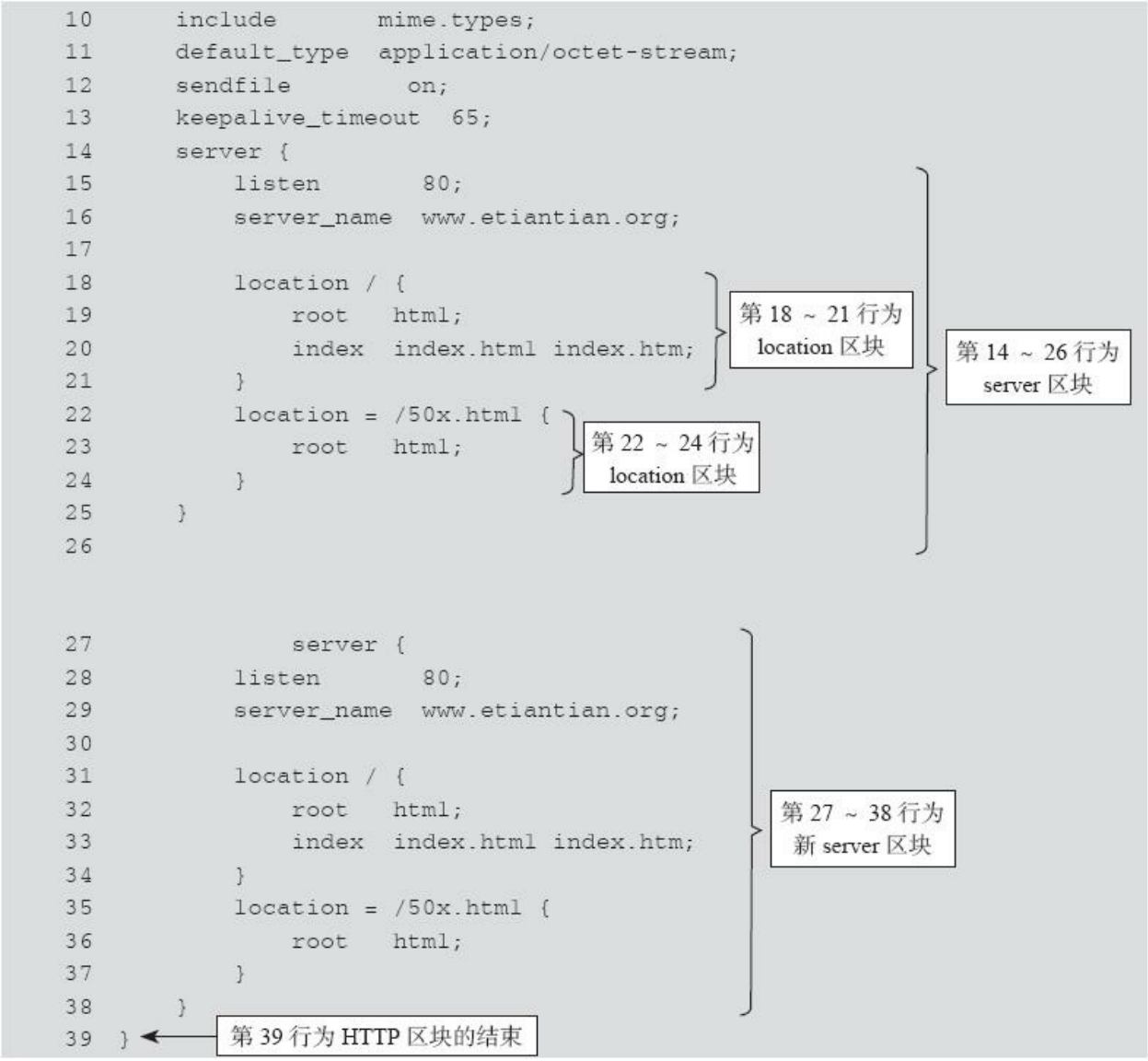
Nginx主配置文件nginx.conf是一个纯文本类型的文件（其他配置文件大多也是如此），它位于Nginx安装目录下的conf目录中，整个配置文件是以区块的形式组织的。一般，每个区块以一个大括号“{}”来表示，区块可以分为几个层次，整个配置文件中，Main区位于最上层，在Main区下面可以有Events区、HTTP区等层级，在HTTP区中又包含有一个或多个server区，每个server区中又可有一个或多个location区，Nginx整个配置文件nginx.conf的主体框架为：

```
1  worker_processes  1;
2  error_log  logs/error.log;
3  pid        logs/nginx.pid;
4
5  events {
6      worker_connections  1024;
7  }
8
9  http {
```

1 ~ 3 行为 Main 区，Nginx 核心功能模块

第 5 ~ 7 行为 events 区，Nginx 核心功能模块

第 9 行是 HTTP 区开始，Nginx http 核心模块



nginx.conf是Nginx最重要的配置文件之一，必须要搞定它。下面针对默认主配置文件nginx.conf的参数做详细的中文解释。

提示：有英文能力的读者一定不要错过英文的注释。

Nginx的配置文件有数百行，我们先去掉所有的默认注释行，最小化讲解Nginx的核心配置参数，其他相关参数在后续学习相关功能时再

一一讲解。

去掉所有注释行后，其形式如下：

```
[root@www conf]# egrep -v "#|^$" nginx.conf.default      ←去掉包含

#号和空行的内容

worker_processes  1;

                    ←

 worker进程的数量

events {                              ←

 事件区块开始

    worker_connections  1024;

                ←每个

worker进程支持的最大连接数

}                                     ←事件区块结束

http {                                ←
```

HTTP区块开始

```
    include       mime.types;
```

←

Nginx支持的媒体类型库文件

```
    default_type  application/octet-stream;
```

←

默认的媒体类型

```
    sendfile        on;
```

←开启高效传输模式

```
    keepalive_timeout  65;
```

←连接超时

```
    server {
```

←第一个

Server区块开始，表示一个独立的虚拟主机站点

```
        listen       80;
```

←

提供服务的端口，默认

80

```
        server_name  localhost;
```

←提供服务的域名主机名

```
        location / {              ←第一个
```

location区块开始

```
            root   html;
```

←站点的根目录，相当于

Nginx的安装目录

```
            index  index.html index.htm;
```

←

默认的首页文件，多个用空格分开

```
        }           ←第一个
```

location区块结果

```
        error_page   500 502 503 504  /50x.html;
```

←

出现对应的

http状态码时，使用

50x.html回应客户

```
        location = /50x.html {          ←
```

location区块开始，访问

50x.html

```
            root   html;
```

←

指定对应的站点目录为

html

```
        }
    }
}          ←
```

HTTP区块结束

整个Nginx配置文件的核心框架如下：

```
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    server {
        listen       80;



        server_name  localhost;



        location / {
            root   html;



            index  index.html index.htm;



        }
    }
}
```

5.4.4 Nginx其他配置文件

如果是配合动态服务（例如PHP服务），Nginx软件还会用到扩展的FastCGI相关配置文件，这个配置是通过在nginx.conf主配置文件中嵌入include命令来实现的，不过默认情况是注释状态，不会生效。

fastcgi.conf配置文件的初始内容如下：

```
[root@www conf]# cat fastcgi.conf
fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_script_name;

fastcgi_param  QUERY_STRING       $query_string;

fastcgi_param  REQUEST_METHOD     $request_method;

fastcgi_param  CONTENT_TYPE       $content_type;

fastcgi_param  CONTENT_LENGTH     $content_length;

fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;

fastcgi_param  REQUEST_URI        $request_uri;

fastcgi_param  DOCUMENT_URI       $document_uri;
```

```
fastcgi_param  DOCUMENT_ROOT      $document_root;

fastcgi_param  SERVER_PROTOCOL    $server_protocol;

fastcgi_param  HTTPS              $https if_not_empty;

fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;

fastcgi_param  SERVER_SOFTWARE    nginx/$nginx_version;

fastcgi_param  REMOTE_ADDR        $remote_addr;

fastcgi_param  REMOTE_PORT        $remote_port;

fastcgi_param  SERVER_ADDR        $server_addr;

fastcgi_param  SERVER_PORT        $server_port;

fastcgi_param  SERVER_NAME        $server_name;

# PHP only,

 required if PHP was built with --enable-force-cgi-redirect
fastcgi_param  REDIRECT_STATUS    200;
```

fastcgi_params默认配置文件的内容如下：

```
 [root@www conf]# cat fastcgi_params
fastcgi_param  QUERY_STRING       $query_string;

fastcgi_param  REQUEST_METHOD     $request_method;

fastcgi_param  CONTENT_TYPE       $content_type;

fastcgi_param  CONTENT_LENGTH     $content_length;

fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;

fastcgi_param  REQUEST_URI        $request_uri;

fastcgi_param  DOCUMENT_URI       $document_uri;

fastcgi_param  DOCUMENT_ROOT      $document_root;

fastcgi_param  SERVER_PROTOCOL    $server_protocol;

fastcgi_param  HTTPS              $https if_not_empty;

fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;
```

```
fastcgi_param  SERVER_SOFTWARE    nginx/$nginx_version;

fastcgi_param  REMOTE_ADDR        $remote_addr;

fastcgi_param  REMOTE_PORT        $remote_port;

fastcgi_param  SERVER_ADDR        $server_addr;

fastcgi_param  SERVER_PORT        $server_port;

fastcgi_param  SERVER_NAME        $server_name;

# PHP only,

 required if PHP was built with --enable-force-cgi-redirect
fastcgi_param  REDIRECT_STATUS    200;


[root@C64 conf]# diff fastcgi_params fastcgi.conf
1a2
> fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_script_name;
```

上述未做注释的目录或文件是比较少用的，有关动态扩展配置，后文讲到PHP服务时再来讲解，初学者跟随本书的进度学习即可。当然，大家如果有需求可以查阅相关文档进一步了解。

5.5 Nginx虚拟主机配置实战

5.5.1 虚拟主机的概念和类型介绍

1.虚拟主机概念

所谓虚拟主机，在Web服务里就是一个独立的网站站点，这个站点对应独立的域名（也可能是IP或端口），具有独立的程序及资源目录，可以独立地对外提供服务供用户访问。

这个独立的站点在配置里是由一定格式的标签段标记的，对于Apache软件来说，一个虚拟主机的标签段通常被包含在<VirtualHost></VirtualHost>内，而Nginx软件则使用一个server{}标签来标示一个虚拟主机。一个Web服务里可以有多个虚拟主机标签对，即可以同时支持多个虚拟主机站点。

2.虚拟主机类型

常见的虚拟主机类型有如下几种。

（1）基于域名的虚拟主机

所谓基于域名的虚拟主机，意思就是通过不同的域名区分不同的虚

拟主机，基于域名的虚拟主机是企业应用最广的虚拟主机类型，几乎所有对外提供服务的网站使用的都是基于域名的虚拟主机，例如：www.etiantian.org 。

（2）基于端口的虚拟主机

同理，所谓基于端口的虚拟主机，意思就是通过不同的端口来区分不同的虚拟主机，此类虚拟主机对应的企业应用主要为公司内部的网站，例如：一些不希望直接对外提供用户访问的网站后台等，访问基于端口的虚拟主机，地址里要带有端口，例如：http://www.etiantian.org:9000 。

（3）基于IP的虚拟主机

同理，所谓基于IP的虚拟主机，意思就是通过不同的IP区分不同的虚拟主机，此类虚拟主机对应的企业应用非常少见。一般不同的业务需要使用多IP的场景都会在负载均衡器上进行VIP绑定，而不是在Web上绑定IP来区分不同的虚拟机。

三种虚拟主机类型均可独立使用，也可以混合使用，读者应把基于域名的虚拟主机类型当做重点来学习掌握，其他的两个类型了解即可。

5.5.2　基于域名的虚拟主机配置实战


说明：本节内容在生产场景中是最常用到的，因此，读者应熟练掌握。

1.配置基于域名的nginx.conf内容

这里使用grep过滤命令来生成基础的Nginx主配置文件nginx.conf，然后根据生成的初始配置进行修改，使其成为所需的形式，具体命令为：

```
[root@www ~]# cd /application/nginx/conf/
[root@www conf]# diff nginx.conf.default nginx.conf
                          #<==初始时这两个配置文件是一致的。



[root@www conf]# egrep -v "#|^$" nginx.conf.default >nginx.conf
                          #<==过滤包含


#号和空行，生成新文件


nginx.conf
```

或者干脆直接创建新的配置文件nginx.conf，然后编辑，输入如下内容：

```
[root@www conf]# vim nginx.conf      #<==使用
```

vim要好于

vi，可以实现

{}高亮显示。

```
worker_processes  1;

events {
    worker_connections  1024;

}
http {
    include       mime.types;

    default_type  application/octet-stream;

    sendfile        on;

    keepalive_timeout  65;

    server {
        listen       80;

        server_name  www.etiantian.org;

        location / {
            root   html/www;
```

```
            index  index.html index.htm;


        }
    }
}
```

提示：上述配置为一个基于www.etiantian.org 域名的站点，这里省略了一切无关的配置参数，让大家学习理解更简单轻松，虚拟主机的关键部分就是server{}标签大括号里的内容。

2.创建域名对应的站点目录及文件

此处配置的是基于域名的虚拟主机，即创建对应的域名站点目录及文件。命令如下：

```
[root@www conf]# mkdir ../html/www -p #<==../表示上级目录，即


/application/nginx目录



[root@www conf]# echo "http:


//www.etiantian.org" >../html/www/index.html
[root@www conf]# cat ../html/www/index.html
http:


//www.etiantian.org
```

上述命令的作用是创建了一个html/www站点目录，对应于虚拟主机配置文件里root根目录的html/www设置（root html/www；）。然后生成一个默认的首页文件index.html，文件的内容是“http://www.etiantian.org ”。

3.检查语法并重新加载Nginx

先检查修改过的Nginx配置文件语法是否正确：

```
[root@www conf]# ../sbin/nginx -t
nginx:

 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:

 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful提示：

syntax is ok”及“

successful”，说明语法正确，如果有问题则需要调整配置，直到语法检查正确为止。

操作前做检查在工作场景十分重要
```

然后平滑重启Nginx，即重新加载配置文件。

```
[root@www conf]# ../sbin/nginx -s reload提示：如果没有启动

Nginx，则无法
```

```
reload。
```

再检查Nginx重新加载后的情况，例如进程和端口是否OK。

```
[root@www conf]# ps -ef|grep nginx
root     10637     1  0 Apr03       00:


00:


00 nginx:


 master process /application/
nginx/sbin/nginx
nginx    12603 10637  0 11:


47       00:


00:


00 nginx:


 worker process
root     12605 12547  0 11:


48 pts/0  00:


00:


00 grep nginx
[root@www conf]# netstat -lntp|grep 80
```

```
tcp        0      0 0.0.0.0:

80          0.0.0.0:

*      LISTEN    10637/nginx
IPv4   14989    0t0   TCP *:

http  (

LISTEN)
```

 提示：出现上述结果表示Nginx服务是OK的，在工作场景下，操作后进行快速检查十分重要。

最后测试域名站点配置的访问结果。这里又分Linux客户端和Windows客户端。

下面是针对Linux客户端的访问：

```
[root@www conf]# echo "10.0.0.8 www.etiantian.org" >>/etc/hosts
[root@www conf]# tail -1 /etc/hosts
10.0.0.8 www.etiantian.org
[root@www conf]# curl www.etiantian.org
http:


//www.etiantian.org
```

 提示：不要忘了在访问的客户端做hosts解析。

下面针对Windows客户端浏览器进行访问。如果域名没有做正式的DNS解析，可在笔记本电脑上编辑hosts文件，添加hosts记录解析。

Windows客户端hosts文件的通用路径为：%systemroot%\system32\drivers\etc\hosts，在老男孩的WindowsXP/Windows7系统上，hosts路径为：C：\WINDOWS\system32\drivers\etc\hosts。

 说明：hosts文件一般被比喻为本地的DNS文件，其功能是把指定域名解析成对应的IP，多个域名可以对应一个IP，默认情况下hosts文件中的配置解析优先于DNS服务器。开发测试等环节会普遍应用到这个hosts文件，简单而方便。如果经常使用该文件，可以在桌面或任务栏建立个hosts快捷方式，省得每次找起来费劲，同时也有小软件或插件，可以帮你实现快速修改。

最终添加的域名和所配置的机器IP对应解析的配置为：

```
10.0.0.8 www.etiantian.org
#这个


www.etiantian.org 是虚拟主机里的域名配置，


10.0.0.8是


Web服务器的
```

IP，还可以一个

IP对应多个域名，空格隔开即可

说明：win32 hosts的路径位置为：%SYSTEMROOT%\System32\drivers\etc\hosts。

可采用如下步骤打开编辑器：开始→运行→放入上述路径，回车选择编辑器。 也可以在开始→运行后，输入drivers或system32，然后在文件夹上敲文件的开头字母逐层查找。

遇到hosts文件无法编辑的时候，需要在开始菜单找到记事本，右键使用管理员权限打开记事本，然后在记事本里寻找路径打开编辑hosts文件。

配置好hosts解析后，最好在dos提示符下检查一下解析效果。

C：

\Users\oldboy>ping www.etiantian.org正在

Ping www.etiantian.org [10.0.0.8] 具有

32 字节的数据：

来自

```
10.0.0.8 的回复:

字节

=32 时间

<1ms TTL=64
```

---

ping www.etiantian.org域名返回10.0.0.8就表示对了。

最后在浏览器中输入域名http://www.etiantian.org ，查看浏览结果，如图5-8所示。

图5-8　第一个域名访问结果

出现上述结果表示一切正常，如果你的配置出现了问题，只需要细致地查看每一个字符的配置及各个步骤是不是都做对了。是不是很简单呢？

小测试：　在继续阅读下文之前，先留个测试，如果要新增域名分别为bbs.etiantian.org和blog.etiantian.org的两个虚拟主机，其网站根目录分别为html/bbs、html/blog，当访问各自的域名站点时，显示各自域名

对应的目录下文件内容（内容即为对应域名），该怎么做？请大家试试看能否自己搞定。

4.配置多个基于域名的虚拟主机

相信此刻聪明的你已经搞定了上面的小测试。不过老男孩还是给大家演示一下步骤，你可以顺便看看我的配置是不是和你的一致。

（1）增加新域名对应的配置

前面已经增加了一个www.etiantian.org 虚拟主机的配置，下面再增加两个虚拟主机的配置，站点域名分别为bbs.etiantian.org和blog.etiantian.org。增加的配置一定要在nginx.conf的http{}区块内，最好放在www.etiantian.org 虚拟主机配置的下面，增加的内容如图5-9所示。

```
server {
    listen       80;
    server_name  bbs.etiantian.org;
    location / {
        root   html/bbs;
        index  index.html index.htm;
    }
}
server {
    listen       80;
    server_name  blog.etiantian.org;
    location / {
        root   html/blog;
        index  index.html index.htm;
    }
}
```

图5-9　Nginx基于域名的虚拟主机图


提示：注意增加内容的位置是在nginx.conf的http{}区块内，即第一个server结束标签“}”的下面。增加技巧为：复制www.etiantian.org虚拟主机的整个标签段，在文件结尾最后一个“}”前粘贴修改即可。

在整体格式上，新增域名虚拟主机的配置和前文配置过的www.etiantian.org 域名的虚拟主机是一样的，区别就是server_name和root参数的配置不同，此时基于3个域名的完整nginx.conf配置文件的内容为：

```
[root@www conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;


    default_type  application/octet-stream;


    sendfile        on;


    keepalive_timeout  65;


    server {
        listen       80;
```

```
    server_name  www.etiantian.org;



    location / {
        root   html/www;



        index  index.html index.htm;



    }
}
server {
    listen       80;



    server_name  bbs.etiantian.org;



    location / {
        root   html/bbs;



        index  index.html index.htm;



    }
}
server {
    listen       80;



    server_name  blog.etiantian.org;



    location / {
        root   html/blog;
```

```
            index  index.html index.htm;


        }
    }
}
```

（2）创建新虚拟主机站点对应的目录和文件

下面的命令用于创建上述两个新增域名分别对应的站点目录及文件。

```
[root@www conf]# mkdir ../html/bbs  ../html/blog -p
[root@www conf]# echo "http:


//bbs.etiantian.org" >../html/bbs/index.html
[root@www conf]# echo "http:


//blog.etiantian.org" >../html/blog/index.html
```

检查配置的结果，命令如下：

```
[root@www conf]# cat ../html/bbs/index.html
http:


//bbs.etiantian.org
[root@www conf]# cat ../html/blog/index.html
http:


//blog.etiantian.org
```

如果熟悉Shell编程的话，上述多个站点内容可以使用一条脚本命令

搞定，还可以执行后检查结果，如下：

```
[root@www conf]# for n in www blog bbs;


do
mkdir -p ../html/$n;


echo "http:


//${n}.etiantian.org" >../html/$n/index.html;


cat ../html/$n/index.html;


done
```

 提示：怎么样？Shell编程很强大吧，有关Shell编程的内容请参考老男孩的其他书及博客。

下面检查站点的目录结构：

```
[root@www conf]     #LANG=en
[root@www conf]     #tree ../html/ 如果没有


tree命令，需要


yum install tree -y安装。
```

```
../html/
|-- 50x.html
|-- bbs          #<==bbs站点目录



|   `-- index.html
|-- blog          #<==blog站点目录



|   `-- index.html
|-- index.html
`-- www          #<==www站点目录



    `-- index.html
3 directories,


 5 files
```

（3）重新加载Nginx配置

每次更改Nginx的配置都需要管理员重新加载，使配置生效，这是因为Nginx在启动时已把所有配置信息都加载到内存中了，若更改了磁盘上的nginx.conf配置文件，需要把新的配置重新加载到内存中，使其生效。这样设计的目的是大幅度提升Nginx服务的访问性能。检查语法及重新加载配置的完整命令如下：

```
[root@www conf]# /application/nginx/sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@www conf]# /application/nginx/sbin/nginx -s reload
```

（4）在客户端测试

同样分为Linux客户端和Windows客户端。在Linux客户端的测试如下：

```
[root@www conf]# tail -1 /etc/hosts     #<==查看配置的


hosts文件



10.0.0.8 www.etiantian.org bbs.etiantian.org blog.etiantian.org
[root@www conf]# curl www.etiantian.org
http:



//www.etiantian.org
[root@www conf]# curl bbs.etiantian.org
http:



//bbs.etiantian.org
[root@www conf]# curl blog.etiantian.org
http:



//blog.etiantian.org
```

成功！还是Linux客户端测试Web更简单、直接。

在Windows客户端的测试如下。

修改C:\WINDOWS\system32\drivers\etc\hosts，命令如下：

```
10.0.0.8    www.etiantian.org  bbs.etiantian.org blog.etiantian.org
```

浏览器检查结果如图5-10所示。

图5-10　新增两个域名访问结果合图

出现上述结果，表示成功完成新增2个域名的虚拟主机配置，如果遇到问题，解决办法请阅读前文故障排错章节，验证完毕后，备份当前的配置留着以后用，备份命令如下：

```
[root@www conf]# /bin/cp nginx.conf  nginx.conf_BaseName
```

5.5.3 基于端口的虚拟主机配置实战

基于端口的虚拟主机在生产环境中不多见，仅偶尔会用到，一般为公司内部人员提供访问，如OA系统、网站程序的后台、CMS发布后台、MySQL的Web客户端phpmyadmin等，使用特殊端口多是从安全上考虑的。下面讲下基于端口的虚拟主机相关配置部署。

1.配置虚拟主机监听的端口

如果要配置基于端口的虚拟主机，就需要为每个虚拟主机配置不同的端口。这里以上述基于域名的3个虚拟主机为例进行讲解。首先，编辑nginx.conf主配置文件，然后把每个虚拟主机的“listen 80；”这个配置行的80数字端口修改掉，内容见下文，注意server_name域名位置可以不做任何变更，哪怕是相同域名也可以，因为，基于端口的虚拟主机就是通过端口来唯一区别不同的虚拟主机的，只要端口不同就是不同的虚拟主机。

2.修改虚拟主机配置

经过修改后，完整的基于端口的多个虚拟主机配置如下：

```
[root@www conf]# cat nginx.conf
worker_processes  1;
```

```
events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    server {
        listen       80;



        server_name  www.etiantian.org;



        location / {
            root   html/www;



            index  index.html index.htm;



        }
    }
    server {
        listen       81;


          #←由基于域名的
```

80改为基于端口的

```
81
        server_name  bbs.etiantian.org;



        location / {
            root   html/bbs;



            index  index.html index.htm;



        }
    }
    server {
        listen       82;


          #←由基于域名的
```

80改为基于端口的

```
82
        server_name  blog.etiantian.org;



        location / {
            root   html/blog;



            index  index.html index.htm;



        }
    }
}
```

以上端口可以任意改，只要不和已有的服务冲突即可，原则上应该是大于1024小于65535的任意端口。注释编辑完成后保存配置。

3.检查语法重新加载配置生效

执行5.5.2节中的“系列检查确认动作”，然后重启Nginx。

```
[root@www conf]# /application/nginx/sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@www conf]# /application/nginx/sbin/nginx -s reload
[root@www conf]# netstat -lntup|grep nginx
tcp     0    0 0.0.0.0:


81        0.0.0.0:


*        LISTEN    10637/nginx
tcp     0    0 0.0.0.0:


82        0.0.0.0:


*        LISTEN    10637/nginx
tcp     0    0 0.0.0.0:


80        0.0.0.0:


*        LISTEN    10637/nginx
```

现在可以看到3个端口了。

4.测试不同端口的访问结果

通过浏览器访问如下3个地址，查看结果如图5-11所示。

```
http:

//www.etiantian.org:

80      #←端口为

80可以省略不写


http:

//blog.etiantian.org:

81
http:

//bbs.etiantian.org:

82
```

图5-11　基于端口的3个站点访问内容展示

浏览到的内容如果和URL地址栏里的域名部分一样，就表示配置正确了。

Nginx虚拟主机官方帮助的网址为：http://Nginx.org/en/docs/http/request_processing.html 。

5.5.4　基于IP的虚拟主机配置实战

基于IP的虚拟主机在生产环境中的应用更为少见，因此，本节的内容读者了解即可。

1.在服务器网卡上增加多个IP

既然要配置基于IP的虚拟主机，就需要让每个虚拟主机有不同的IP地址，此处以增加辅助IP的形式临时在eth0网卡上增加2个不同的IP，命令如下：

```
[root@www conf]# ip addr add 10.0.0.9/24 dev eth0
[root@www conf]# ip addr add 10.0.0.10/24 dev eth0
```

当然，也可以使用ifconfig添加别名IP，但考虑到该命令在后续系统版本中将要去掉了，因此使用更先进的IP命令生成IP地址。

增加完毕后，使用如下命令检查配置生效结果：

```
 [root@www conf]# ip add|grep 10.0.0          #←过滤网卡上的

3个


IP
    inet 10.0.0.8/24 brd 10.0.0.255 scope global eth0
    inet 10.0.0.9/24 scope global secondary eth0
    inet 10.0.0.10/24 scope global secondary eth0
[root@www conf]# ping 10.0.0.10           #←
```

ping检查

PING 10.0.0.10 （

10.0.0.10）

56（

84）

bytes of data.
64 bytes from 10.0.0.10:

icmp_seq=1 ttl=64 time=0.023 ms
[root@www conf]# ping 10.0.0.9 #←

ping检查

PING 10.0.0.9 （

10.0.0.9）

56（

84）

bytes of data.
64 bytes from 10.0.0.9:

icmp_seq=1 ttl=64 time=0.029 ms

2.增加虚拟主机配置

基于IP的虚拟主机实际配置示例如下。这是一个端口和IP混合的虚拟主机示例，读者可以自行修改，使其仅仅基于IP，即每个虚拟主机的server_name字段都换成IP地址。

```
[root@www conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    server {
        listen       10.0.0.8:



80;
```

```
        server_name  www.etiantian.org;
```

此处也可以改成对应

```
IP 10.0.0.8
        location / {
            root   html/www;

            index  index.html index.htm;

        }
    }
    server {
        listen       10.0.0.9:

81;

        server_name  bbs.etiantian.org;
```

此处也可以改成对应

```
IP 10.0.0.9
        location / {
            root   html/bbs;

            index  index.html index.htm;

        }
    }
    server {
        listen       10.0.0.10:

82;
```

```
        server_name  blog.etiantian.org;

 此处也可以改成对应

IP 10.0.0.10
        location / {
            root   html/blog;


            index  index.html index.htm;


        }
    }
}
```

重新加载Nginx服务使修改的配置生效，此处忽略，读者别忘记。

3.客户端浏览器中访问方法

打开浏览器分别输入如下内容：

```
http:

//10.0.0.8:

80,


http:

//10.0.0.9:
```

```
    81;

    http:

    //10.0.0.10:

    82;
```

然后回车，如果可以分别看到如图5-12所示的内容，就表示配置成功了。

图5-12　基于IP、端口、域名混合配置多站点展示

提示：基于IP的虚拟主机配置在生产环境中不经常使用，如果需要不同的IP对应不同的服务，可在网站前端的负载均衡器上配置。因此，读者了解基于IP的虚拟主机是咋回事就OK了。

5.5.5 Nginx配置虚拟主机的步骤

Nginx配置虚拟主机的步骤如下（适合各类虚拟主机类型）：

1）增加一个完整的server标签段到结尾处。注意，要放在http的结束大括号前，也就是将server标签段放入http标签。

2）更改server_name及对应网页的root根目录，如果需要其他参数，可以增加或修改。

3）创建server_name域名对应网页的根目录，并且建立测试文件，如果没有index首页，访问会出现403错误。

4）检查Nginx配置文件语法，平滑重启Nginx服务，快速检查启动结果。

5）在客户端对server_name处配置的域名做host解析或DNS配置，并检查（ping域名看返回的IP是否正确）。

6）在Win32浏览器中输入地址访问，或者在Linux客户端做hosts解析，用wget或curl接地址访问。

Nginx虚拟主机的官方帮助网址为：http://Nginx.org/en/docs/http/request_processing.html 。

5.5.6　企业场景中重启Nginx后的检测策略

在企业运维实战场景中，每一个配置操作处理完毕后都应该进行快速有效的检查，这是一个合格运维人员的良好习惯。在老男孩的实际工作中，启动Nginx的同时，还会调用脚本通过获取header信息或模拟用户访问指定URL（wget等方式）来自动检查Nginx的启动是否正常，最大限度地保证服务重启后，能迅速确定网站情况，而无须手工敲命令查看。这样，如果配置有问题（非语法问题，语法问题已经用-t参数检查过了），就可以迅速使用上一版备份的配置文件覆盖回来，使得影响用户的时间最短。

下面是一个重启Nginx服务后，检查启动是否正常的脚本，可以把它包含在Nginx启动脚本（需要另行开发）里的start等启动位置。

```
[root@www conf]# cat check_url.sh
#!


/bin/sh
#author:


oldboy  20100325 QQ31333741
#------function split------
. /etc/init.d/functions
function checkURL ()


{
        checkUrl=$1
        echo 'check url start ...'
```

```
        judge= (


$ (


curl -I -s --connect-timeout 2 ${checkUrl}|head -1|tr " " "\n") )



        if [[ "${judge[1]}" == '200' && "${judge[2]}" == 'OK' ]]
           then
                action "${checkUrl}" /bin/true
        else
                action "${checkUrl}" /bin/false
                echo -n "retrying again...";


sleep 3;



                judgeagain= (


$ (


curl -I -s --connect-timeout 2 ${checkUrl}|head -1|tr
"\r" "\n") )



                if [[ "${judgeagain[1]}" == '200' && "${judgeagain[2]}"=='OK' ]]
                then
                        action "${checkUrl},


retried again" /bin/true
                else
                        action "${checkUrl},


retried again" /bin/false
                fi
        fi
        sleep 1;
```

```
}
#usage method
checkURL http:



//www.etiantian.org
[root@www conf]# echo "10.0.0.8 www.etiantian.org" >>/etc/hosts
[root@www conf]# tail -1 /etc/hosts
10.0.0.8 www.etiantian.org
[root@www conf]# sh check_url.sh
check url start ...
http:



//www.etiantian.org                                    [确定



]
```

此脚本的功能是检测访问www网站时的响应header是否返回200，以及状态是否为OK。当检测失败时，3秒后再检测一次，然后，格式化检测结果并输出，如果有多个域名，可以对上述命令传参的不同域名循环检查。检测方法可以是端口、URI、响应header等，具体将根据业务需求进行选择。对Shell不是很了解的读者，暂时可以略过，不影响学习本书的内容。

上面给出的是一个非常简单的例子，但在实际工作中，启动服务后的检查脚本可能会比较复杂，例如：淘宝网的业务会在重启Nginx后立刻执行脚本，对数个和Web关联的API接口进行检查，确保启动Nginx后，自身及所有关联业务正常。

5.6 Nginx常用功能配置实战

5.6.1 规范优化Nginx配置文件

大家如果了解Apache软件，就会知道Apache主配置包含虚拟主机子文件的方法，这里也借鉴了Apache的这种包含方法。

Nginx的主配置文件为nginx.conf，主配置文件包含的所有虚拟主机的子配置文件会统一放入extra目录中，虚拟主机的配置文件按照网站的域名或功能取名，例如www.conf、bbs.conf、blog.conf等。当然，如果虚拟主机的数量不是很多，也可以把多个虚拟主机配置成一个单独的配置文件，仅仅和Nginx的主配置文件nginx.conf分离开即可。

这里使用的参数是include，下面先看看它的语法：

```
include file | mask;
```

它可以放置在Nginx配置中的任何位置。用法示例如下[1]：

```
include mime.types;

include www.conf;
```

```
        #<==包含单个文件

include vhosts/*.conf;

    #<==包含

vhosts下所有以

conf结尾的文件
```

下面是优化Nginx配置的实战方案。

具体实施步骤如下：

```
[root@www conf]# mkdir extra
[root@www conf]# /bin/cp nginx.conf_BaseName nginx.conf
                    #<==以基于域名的虚拟主机为例



[root@www conf]# sed -n '10,


17p' nginx.conf
#<==打印


www.etiantian.org虚拟主机配置内容



    server {
        listen       80;
```

```
        server_name  www.etiantian.org;



        location / {
            root   html/www;



            index  index.html index.htm;



        }
    }
[root@www conf]# sed -n '10,


17p' nginx.conf >extra/www.conf
#<==把


www.etiantian.org虚拟主机配置内容写入


extra/www.conf
[root@www conf]# sed -n '18,


25p' nginx.conf
#<==打印


bbs.etiantian.org虚拟主机配置内容



    server {
        listen       80;



        server_name  bbs.etiantian.org;



        location / {
            root   html/bbs;
```

```
            index  index.html index.htm;


        }
    }
[root@www conf]# sed -n '18,


25p' nginx.conf >extra/bbs.conf
#<==把


bbs.etiantian.org虚拟主机配置内容写入


extra/bbs.conf
[root@www conf]# sed -n '26,


33p' nginx.conf
#<==打印


blog.etiantian.org虚拟主机配置内容



    server {
        listen       80;


        server_name  blog.etiantian.org;


        location / {
            root   html/blog;


            index  index.html index.htm;


        }
```

```
    }
[root@www conf]# sed -n '26,


33p' nginx.conf >extra/blog.conf
#<==把


blog.etiantian.org虚拟主机配置内容写入


extra/blog.conf
```

删除主配置文件nginx.conf中所有虚拟主机的配置（包含server{}标签），这里是10到33行的内容，需要提前查好行号，打印确认无误后再删除。

```
[root@www conf]# sed -i '10,


33d' nginx.conf
[root@www conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;


    default_type  application/octet-stream;


    sendfile        on;
```

```
    keepalive_timeout  65;


}
```

把虚拟主机独立配置文件www.conf、bbs.conf、blog.conf的信息包含到nginx.conf里，这样就把主配置和各个虚拟主机配置分离了，具体包含的配置内容如下：

```
include extra/www.conf;


include extra/bbs.conf;


include extra/blog.conf;
```

快速操作的过程如下。

1）操作前配置文件内容，命令如下：

```
[root@www conf]# cat -n nginx.conf
     1worker_processes  1;



     2events {
     3    worker_connections  1024;



     4}
     5  http {
```

```
    6    include       mime.types;

    7    default_type  application/octet-stream;

    8    sendfile        on;

    9    keepalive_timeout  65;

   10}
```

2）执行下面的插入命令。

```
[root@www conf]# sed -i '10 i include extra/www.conf;

\ninclude extra/bbs.conf;

\ninclude extra/blog.conf;

'  nginx.conf
```

上述sed插入命令生效的结果是在下面的配置中增加三行包含虚拟主机文件的配置：

```
[root@www conf]# cat -n nginx.conf
     1worker_processes  1;


     2events {
     3    worker_connections  1024;
```

```
4}
5http {
6    include       mime.types;

7    default_type  application/octet-stream;

8    sendfile        on;

9    keepalive_timeout  65;

10include extra/www.conf;

11include extra/bbs.conf;

12include extra/blog.conf;

13}
```


提示：手工改也可以啊，呵呵！不过sed还是很高大上的命令，要会用才行。

3）重新加载配置，并测试更改后的结果。

命令如下：

```
[root@www conf]# ../sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@www conf]# ../sbin/nginx -s reload
[root@www conf]# tail -1 /etc/hosts
10.0.0.8 www.etiantian.org bbs.etiantian.org blog.etiantian.org
[root@www conf]# curl www.etiantian.org
http:


//www.etiantian.org
[root@www conf]# curl bbs.etiantian.org
http:


//bbs.etiantian.org
[root@www conf]# curl blog.etiantian.org
http:


//blog.etiantian.org
```

优化Nginx配置文件后进行网站访问，一切正常！

通过在主配置文件中加上include包含的配置，可以让Nginx的配置更加简单、清晰、规范。修改后的最终配置文件的内容如下：

```
[root@www conf]# cat -n nginx.conf
     1worker_processes  1;



     2events {
     3    worker_connections  1024;



     4}
     5http {
```

```
     6    include       mime.types;

     7    default_type  application/octet-stream;

     8    sendfile        on;

     9    keepalive_timeout  65;

    10include extra/www.conf;

    11include extra/bbs.conf;

    12include extra/blog.conf;

    13}
[root@www conf]# tree extra/
extra/├──

 bbs.conf├──

 blog.conf└──

 www.conf
0 directories,

 3 files
[root@www conf]# cat extra/www.conf
    server {
        listen       80;
```

```
        server_name  www.etiantian.org;



        location / {
            root   html/www;



            index  index.html index.htm;



        }
    }
[root@www conf]# cat extra/bbs.conf
    server {
        listen       80;



        server_name  bbs.etiantian.org;



        location / {
            root   html/bbs;



            index  index.html index.htm;



        }
    }
[root@www conf]# cat extra/blog.conf
    server {
        listen       80;



        server_name  blog.etiantian.org;



        location / {
            root   html/blog;
```

```
            index  index.html index.htm;


        }
    }
```

如果需要新增虚拟主机站点，可以按照上述配置步骤在extra下创建新的虚拟主机文件，然后在主配置文件nginx.conf中包含在extra下创建的新虚拟主机文件。

如果虚拟主机数量过多，也可以按业务分类，例如将几个同业务的虚拟主机配置放到一个文件里，这样不至于因虚拟主机太多而导致很零碎，一切都可以灵活地调整。运维思想很关键，机器是死的，人是活的，要学会多变通才好。

[1] 参考自：http://nginx.org/en/docs/ngx_core_module.html#include。

5.6.2 Nginx虚拟主机的别名配置

1.虚拟主机的别名介绍及配置

所谓虚拟主机别名，就是为虚拟主机设置除了主域名以外的一个或多个域名名字，这样就能实现用户访问的多个域名对应同一个虚拟主机网站的功能。

以www.etiantian.org 域名的虚拟主机为例，为其增加一个别名etiantian.org，使得访问etiantian.org时，在该域名出现的网站内容和访问www.etiantian.org 得到的结果是一样的，这是企业场景中活生生的基本需求配置。

具体配置内容及对比见表5-3。

表5-3 虚拟主机别名的配置前后对比

原始 www 虚拟主机配置（修改前 extra/www.conf 内容）	带别名虚拟主机配置（修改后 extra/www.conf 内容）

[root@www conf]# cat extra/www.conf
#www virtualhost by oldboy
server {
listen 80;
server_name www.etiantian.org;
location / {
root html/www;
index index.html index.htm;
}
}
	[root@www conf]# cat extra/www.conf
	#www virtualhost by oldboy
	server {
	listen 80;
	server_name www.etiantian.org etiantian.org;
	location / {
	root html/www;
	index index.html index.htm;
	}
	}
	# 提示：仅在 server_name 所在行的行尾增加了 etiantian.org 内容！

重新加载配置并测试访问结果。

1）重新加载配置，命令如下：

```
[root@www conf]# ../sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@www conf]# ../sbin/nginx -s reload
```

2）进行配置域名解析（Linux客户端），命令如下：

```
[root@www conf]# tail -1 /etc/hosts
10.0.0.8 www.etiantian.org bbs.etiantian.org blog.etiantian.org etiantian.org
[root@www conf]# ping etiantian.org
PING www.etiantian.org (


10.0.0.8)


 56(


84)


 bytes of data.
64 bytes from www.etiantian.org (


10.0.0.8):


 icmp_seq=1 ttl=64 time=0.033 ms
```

3）测试访问结果（Linux客户端），命令如下：

```
[root@www conf]# curl etiantian.org
http:


//www.etiantian.org
[root@www conf]# curl www.etiantian.org
http:


//www.etiantian.org
```

访问etiantian.org所出现的内容和www.etiantian.org 是一样的。在工作中可以通过别名的配置实现多域名访问一个站点，例如baidu.com和www.baidu.com 就是一样的内容，当然这里的实现方法除了别名，还有rewrite 301跳转的配置思路。

2.虚拟主机别名在生产中的使用场景案例

多数企业网站希望访问www.etiantian.org 和etiantian.org时，所浏览的是同一个页面，若有这类需求，就可以让etiantian.org以别名的方式出现，这时两个域名都要解析到服务器的IP地址。

在老男孩的生产环境中，曾经还利用别名来监控集群下面RS的URL是否正常。如：

```
server_name  www.etiantian.org www1.etiantian.org www2.etiantian.org;
```

可以在监控服务器里配置hosts来监控RS www1.etiantian.org、www2.etiantian.org等地址是否正常，进而判断每一台机器的www.etiantian.org 是否正常。如果不使用别名则很难通过域名URL的方式检测判断节点下面的机器是否正常（因为这些集群节点的域名是同一个）。

5.6.3 Nginx状态信息功能实战

1.Nginx status介绍

Nginx软件的功能模块中有一个ngx_http_stub_status_module模块，这个模块的主要功能是记录Nginx的基本访问状态信息，让使用者了解Nginx的工作状态，例如连接数等信息。要使用状态模块，在编译Nginx时必须增加http_stub_status_module模块来支持。

可通过如下方法检查编译安装Nginx时是否设定了上述模块：

```
[root@www conf]# ../sbin/nginx -V      #<==检查编译安装时设置的编译参数

nginx version:

 nginx/1.6.3
built by gcc 4.4.7 20120313 (

Red Hat 4.4.7-11)

  (

GCC)

TLS SNI support enabled
configure arguments:
```

```
 --user=nginx --group=nginx --prefix=/application/nginx-1.6.3 --with-http_stub_statu
```

```
--with-http_stub_status_module就对了
```

2.配置Nginx status

具体配置过程如下。

1）生成状态配置，并增加状态配置参数

操作命令如下：

```
cat >>/application/nginx/conf/extra/status.conf<<EOF
0##status
server{
    listen  80;



    server_name  status.etiantian.org;



    location / {
      stub_status on;



      access_log   off;



    }
  }
EOF
sed -i '13 i include extra/status.conf;
```

```
'  nginx.conf
```

下面是操作过程及结果检查：

```
[root@www conf]# cat >>/application/nginx/conf/extra/status.conf<<EOF
> ##status
> server{
>     listen  80;



>     server_name  status.etiantian.org;



>     location / {
>       stub_status on;



>       access_log   off;



>     }
>   }
> EOF
[root@www conf]# cat extra/status.conf
0##status
server{
    listen  80;



    server_name  status.etiantian.org;



    location / {
      stub_status on;


          #<==打开状态信息开关



      access_log   off;
```

```
    }
  }
[root@www conf]# sed -i '13 i include extra/status.conf;


'  nginx.conf
[root@www conf]# cat -n nginx.conf
     1worker_processes  1;



     2events {
     3    worker_connections  1024;



     4}
     5http {
     6    include       mime.types;



     7    default_type  application/octet-stream;



     8    sendfile        on;



     9    keepalive_timeout  65;



    10include extra/www.conf;



    11include extra/bbs.conf;



    12include extra/blog.conf;



    13include extra/status.conf;
```

```
        #<==不要忘了增加包含文件的配置到主配置文件


nginx.conf
    14}
```

2）检查语法，并重启服务

命令如下：

```
[root@www conf]# ../sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@www conf]# ../sbin/nginx -s reload
```

3）配置hosts解析（Windows客户端）

命令如下：

```
10.0.0.8    www.etiantian.org  bbs.etiantian.org blog.etiantian.org
10.0.0.8 etiantian.org status.etiantian.org
```

4）测试访问结果

访问情况如图5-13所示。

图5-13　查看Nginx状态信息图

也可以用location的方式实现状态信息配置，例如在任意一个虚拟主机里，为server标签增加如下配置：

```
location /nginx_status {
stub_status on;

access_log off;

allow 10.0.0.0/24;

            #<==设置允许和禁止的

IP段访问

deny all;

                        #<==设置允许和禁止的

IP段访问

}
```

3.Nginx status显示结果详解

通常，打开http://status.etintian.org 时会有如图5-13所示的状态信息显示，可以通过压力测试工具获取这些数值，结果说明如下。

```
Active connections：


2872
#<==表示


Nginx 正处理的活动连接数有


2872个。



server accepts handled requests
29431211 29431211 110298687
```

Reading：80 Writing：35 Waiting：2757

其中，第一个server表示Nginx启动到现在共处理了29431211个连接；

第二个accepts表示Nginx启动到现在共成功创建了29431211次握手；

请求丢失数=（握手数-连接数），可以看出，本次状态显示没有丢

失请求。

第三个handled requests，表示总共处理了110298687次请求；

Reading为Nginx读取到客户端的Header信息数。

Writing为Nginx返回给客户端的Header信息数。

Waiting为Nginx已经处理完正在等候下一次请求指令的驻留连接。在开启keep-alive的情况下，这个值等于active-（reading+writing）

特别提示：为了安全起见，这个状态信息要防止外部用户查看。

5.6.4 为Nginx增加错误日志（error_log）配置

Nginx软件会把自身运行的故障信息及用户访问的日志信息记录到指定的日志文件里。

1.Nginx错误日志信息介绍

配置记录Nginx的错误信息是调试Nginx服务的重要手段，属于核心功能模块（ngx_core_module）的参数，该参数名字为error_log，可以放在Main区块中全局配置，也可以放置不同的虚拟主机中单独记录。

error_log的语法格式及参数语法说明如下：

```
error_log    file          level;
关键字        日志文件      错误日志级别
```

其中，关键字error_log不能改变，日志文件可以指定任意存放日志的目录，错误日志级别常见的有[debug|info|notice|warn|error|crit|alert|emerg]，级别越高，记录的信息越少，生产场景一般是warn|error|crit这三个级别之一，注意不要配置info等较低级别，会带来巨大磁盘I/O消耗。

error_log的默认值为：

```
#default:

 error_log logs/error.log error;
```

可以放置的标签段为：

```
#context:

 main,

 http,

 server,

 location
```

参考资料：http://nginx.org/en/docs/ngx_core_module.html#error_log

。

2.Nginx错误日志配置

编辑主配置文件nginx.conf，增加错误日志的配置方法如下：

```
worker_processes  1;

error_log  logs/error.log;
```

```
    #<==非常简单，一般配置这一行即可。


events {
    worker_connections  1024;


}
http {
    include       mime.types;


    default_type  application/octet-stream;


    sendfile        on;


    keepalive_timeout  65;


include extra/www.conf;


include extra/bbs.conf;


include extra/blog.conf;


include extra/status.conf;


}
```

 提示：灰底加粗的配置为增加的日志配置内容。

5.7 Nginx访问日志（access_log）

5.7.1 Nginx访问日志介绍

Nginx软件会把每个用户访问网站的日志信息记录到指定的日志文件里，供网站提供者分析用户的浏览行为等，此功能由ngx_http_log_module模块负责。对应的官方地址为：http://nginx.org/en/docs/http/ngx_http_log_module.html 。

5.7.2　访问日志参数

Nginx的访问日志主要由表5-4中的两个参数控制。

表5-4　控制日志的参数

参　数	说　明
log_format	用来定义记录日志的格式（可以定义多种日志格式，取不同名字即可）
access_log	用来指定日志文件的路径及使用何种日志格式记录日志

Nginx日志格式中默认的参数配置如下：

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';
```

Nginx记录日志的默认参数配置如下：

```
access_log  logs/access.log  main;
```

5.7.3　访问日志配置说明

1.日志格式的定义说明

先来看其语法：

定义语法

```
log_format name string ...;
```

其配置位置在http标签内。

日志格式说明如下：

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';
```

其中，log_format为日志格式关键参数，不能变。

main是为日志格式指定的标签，记录日志时通过这个main标签选择指定的格式。其后所接的所有内容都是可以记录的日志信息，具体见表5-5。注意，所有的日志段以空格分隔，一行可以记录多个，不同列的

意义也在表5-5中进行了说明。

表5-5　Nginx日志变量说明

Nginx 日志变量	说　明
$remote_addr	记录访问网站的客户端地址
$http_x_forwarded_for	当前端有代理服务器时，设置 Web 节点记录客户端地址的配置，此参数生效的前提是代理服务器上也进行了相关的 x_forwarded_for 设置
$remote_user	远程客户端用户名称
$time_local	记录访问时间与时区
$request	用户的 http 请求起始行信息
$status	http 状态码，记录请求返回的状态，例如：200、404、301 等
$body_bytes_sents	服务器发送给客户端的响应 body 字节数
$http_referer	记录此次请求是从哪个链接访问过来的，可以根据 referer 进行防盗链设置
$http_user_agent	记录客户端访问信息，例如：浏览器、手机客户端等

在没有特殊要求的情况下，采用默认的配置即可，更多可以设置的记录日志信息的变量见：

http://nginx.org/en/docs/http/ngx_http_log_module.html

2.记录日志的access_log参数说明

下面是有关access_log参数的官方说明。

语法如下：

```
access_log path [format [buffer=size [flush=time]] [if=condition]];

access_log path format gzip[=level] [buffer=size] [flush=time] [if=condition];

access_log syslog:
```

```
server=address[,

parameter=value] [format [if=condition]];
```

---

buffer=size为存放访问日志的缓冲区大小，flush=time为将缓冲区的日志刷到磁盘的时间，gzip[=level]表示压缩级别，[if=condition]表示其他条件。一般的场景中，这些参数都无须配置，极端优化时才可能会考虑这些参数。

access_log off中的off，表示不记录访问日志。

默认配置：access_log logs/access.log combined；

放置位置在http、server、location、if in location、limit_except中。

5.7.4　访问日志配置实战

编辑主配置文件nginx.conf，配置日志的格式，如下：

```
[root@www conf]# sed -n '21,

23 s/#//gp' nginx.conf.default
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
```

把上述内容放到nginx.conf的http标签的首部，如下：

```
[root@www conf]# cat -n nginx.conf
     1worker_processes  1;



     2error_log  logs/error.log;



     3events {
     4    worker_connections  1024;



     5}
     6http {
     7    include       mime.types;



     8    default_type  application/octet-stream;
```

```
 9    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
10                      '$status $body_bytes_sent "$http_referer" '
11                      '"$http_user_agent" "$http_x_forwarded_for"';


12    sendfile        on;


13    keepalive_timeout  65;


14include extra/www.conf;


15include extra/bbs.conf;


16include extra/blog.conf;


17include extra/status.conf;


18}
```

然后在每个虚拟主机里进行配置，使其使用上述格式记录用户访问日志。以www.etiantian.org 站点为例，命令如下：

```
[root@www conf]# cat extra/www.conf
#www virtualhost by oldboy
    server {
        listen       80;


        server_name  www.etiantian.org etiantian.org;
```

```
        location / {
            root   html/www;


            index  index.html index.htm;


        }
        access_log logs/access_www.log main;


    }
```

如果不指定日志格式就会用默认的combined格式记录日志。

接下来，要检查语法，重新加载配置，命令如下：

```
[root@www conf]# ../sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@www conf]# ../sbin/nginx -s reload
```

下面用浏览器模拟用户访问生成日志，在服务器上查看日志结果，命令如下：

```
[root@www conf]# curl www.etiantian.org
http:


//www.etiantian.org
[root@www conf]# ls -l ../logs/access_www.log
```

```
-rw-r--r-- 1 root root 2559 4月


   5 12:


12 ../logs/access_www.log
[root@www conf]# curl www.etiantian.org
http:


//www.etiantian.org
[root@www conf]# tail -1  ../logs/access_www.log
10.0.0.8 - - [05/Apr/2015:


12:


15:


28 +0800] "GET / HTTP/1.1" 200 25 "-" "curl/7.19.7 (


x86_64-redhat-linux-gnu)


 libcurl/7.19.7 NSS/3.15.3 zlib/1.2.3 libidn/1.18 libssh2/1.4.2" "-"
```

使用谷歌浏览器访问的日志结果如下：

```
[root@www conf]# tail -5  ../logs/access_www.log
10.0.0.100 - - [05/Apr/2015:


12:


16:


00 +0800] "GET / HTTP/1.1" 200 25 "-" "Mozilla/5.0 (
```

```
Windows NT 6.1;

 WOW64)

 AppleWebKit/537.36 (

KHTML,

 like Gecko)

 Chrome/39.0.2171.95 Safari/537.36" "-"
10.0.0.100 - - [05/Apr/2015:

12:

16:

00 +0800] "GET /favicon.ico HTTP/1.1" 404 570 "-" "Mozilla/5.0 (

Windows NT 6.1;

 WOW64)

 AppleWebKit/537.36 (

KHTML,

 like Gecko)

 Chrome/39.0.2171.95 Safari/537.36" "-"
```

读者可以根据生成的日志对比上述log_format的日志格式，看是否对应。

定义的日志格式如下：

```
'$remote_addr - $remote_user [$time_local] "$request" ' '$status $body_bytes_sent "$
                    '"$http_user_agent" "$http_x_forwarded_for"';
```

真实的日志内容如下：

```
10.0.0.100 - - [05/Apr/2015:

12:

16:

00 +0800] "GET / HTTP/1.1" 200 25 "-" "Mozilla/5.0 (

Windows NT 6.1;

 WOW64)

 AppleWebKit/537.36 (

KHTML,

 like Gecko)
```

```
Chrome/39.0.2171.95 Safari/537.36" "-"
```

对应说明如下：

·$remote_addr对应的是真实日志里的10.0.0.100，即客户端的IP。

·$remote_user对应的是第二个中杠“-”，没有远程用户，所以用“-”填充。

·[$time_local]对应的是[05/Apr/2015：12：16：00+0800]。

·"$request"对应的是"GET/HTTP/1.1"。

·$status对应的是200状态码，200表示正常访问。

·$body_bytes_sent对应的是25字节，即响应body的大小。

·"$http_referer"对应的是"-"，由于是直接打开域名浏览的，因此，referer没有值。

·"$http_user_agent"对应的是"Mozilla/5.0（Windows NT 6.1；WOW64）AppleWebKit/537.36（KHTML，like Gecko）Chrome/39.0.2171.95 Safari/537.36"。

·"$http_x_forwarded_for"对应的是"-"，因为Web服务没有使用代理，因此此处为"-"。

下面针对日志配置进行深入说明。

可以在记录日志参数中加上buffer和flush选项，这样可在高并发场景下提升网站访问性能。加该选项的命令如下：

```
access_log path [format [buffer=size [flush=time]] [if=condition]];

access_log path format gzip[=level] [buffer=size] [flush=time] [if=condition];

access_log syslog:

server=address[,

parameter=value] [format [if=condition]];

access_log off;
```

具体配置如下：

```
[root@www conf]# cat extra/www.conf
#www virtualhost by oldboy
    server {
        listen       80;

        server_name  www.etiantian.org etiantian.org;

        location / {
```

```
            root   html/www;

            index  index.html index.htm;

        }
        #access_log logs/access_www.log main;

        access_log logs/access_www.log main gzip buffer=32k flush=5s;

        #access_log off;

    }
```

更多内容参考：http://nginx.org/en/docs/http/ngx_http_log_module.html 。

5.7.5 Nginx访问日志轮询切割

默认情况Nginx会把所有的访问日志生成到一个指定的访问日志文件access.log里，但这样一来，时间长了就会导致日志个头很大，不利于日志的分析和处理，因此，有必要对Nginx日志、按天或按小时进行切割，使其分成不同的文件保存。这里使用按天切割的方法。

具体切割脚本如下：

```
[root@www logs]# cat /server/script/cut_nginx_log.sh
#!


/bin/sh
Dateformat=`date +%Y%m%d`
Basedir="/application/nginx"
Nginxlogdir="$Basedir/logs"
Logname="access_www"
[ -d $Nginxlogdir ] && cd $Nginxlogdir||exit 1
[ -f ${Logname}.log ]||exit 1
/bin/mv ${Logname}.log ${Dateformat}_${Logname}.log
$Basedir/sbin/nginx -s reload
```

注意：脚本实现切割Nginx日志的思想为将正在写入的Nginx日志（access_www.log）改名为带日期的格式文件（20150417_access_www.log），然后平滑重新加载Nginx，生成新的Nginx日志（access_www.log）。

下面通过定时任务实现每天00点整定时执行/server/script/cut_nginx_log.sh切割日志。

操作命令如下：

```
cat >>/var/spool/cron/root << EOF
#cut nginx access log by oldboy
00 00 * * * /bin/sh /server/script/cut_nginx_log.sh >/dev/null 2>&1
EOF
```

操作后结果如下：

```
[root@www logs]# crontab -l
#time sync by oldboy at 2010-2-1
*/5 * * * * /usr/sbin/ntpdate time.nist.gov >/dev/null 2>&1
#cut nginx access log by oldboy
00 00 * * * /bin/sh /server/script/cut_nginx_log.sh >/dev/null 2>&1
```

最终日志切割效果如下：

```
[root@www logs]# ll总用量


 28
-rw-r--r-- 1 root root     0 4月


  18 00:


06 access.log
-rw-r--r-- 1 root root 92456 4月


  18 00:


07 access_www_20150418.log
-rw-r--r-- 1 root root  5648 4月


  18 00:
```

```
07 access_www_20150419.log
-rw-r--r-- 1 root root     0 4月


  19 00:


00 access_www.log
-rw-r--r-- 1 root root 18064 4月


  19 00:


00 error.log
-rw-r--r-- 1 root root     5 4月


   7 19:


09 nginx.pid
```

Nginx常用的日志收集及分析工具有rsyslog、awstats、flume、ELK（Elasticsearch logstash Kibana）、storm等。如果读者有需求，可以自行研究或参考老男孩的其他课程、书籍或网上资料。更多常用的开源运维工具见http://oldboy.blog.51cto.com/2561410/775056 。

5.8 Nginx location

5.8.1 location作用

location指令的作用是根据用户请求的URI来执行不同的应用。URI的知识前面章节已经讲解过，其实就是根据用户请求的网站地址URL进行匹配，匹配成功即进行相关的操作。

下面是官方提供的常见的location匹配语法。

5.8.2 location语法

location使用的语法为：

```
location [ = | ~ | ~* | ^~ ] uri {
 ...
}
```

表5-6是对location语法的说明。上述语法中的URI部分是关键，这个URI可以是普通的字符串地址路径，或者是正则表达式，匹配成功则执行后面大括号里的相关指令。正则表达式的前面还可以有“~”或“~*”等特殊的字符。

表5-6　对location语法列表说明

location	[=|~|~*|^~|@]	uri	{ ... }
指令	匹配标识	匹配的网站网址	匹配 URI 后要执行的配置段

匹配这两种特殊字符“~”或“~*”的区别为：“~”用于区分大小写（大小写敏感）的匹配；“~*”用于不区分大小写的匹配。还可以用逻辑操作符“！”对上面的匹配取反，即“！~”和“！~*”。此外，“^~”的作用是在进行常规的字符串匹配检查之后，不做正则表达式的检查，即如果最明确的那个字符串匹配的location配置中有此前缀，那么不做正则表达式的检查。

5.8.3 location匹配示例

下面是一组典型的location匹配，是官方的例子。

```
location = / {
    [ configuration A ]
}
location / {
    [ configuration B ]
}
location /documents/ {
    [ configuration C ]
}
location ^~ /images/ {
    [ configuration D ]
}
location ~* \.(
gif|jpg|jpeg)
$ {
    [ configuration E ]
}
```

在上述location配置中，当用户请求“/”时，将匹配configuration A，当用户请求“/index.html”时，将匹配configuration B；当用户请求“/documents/document.html”时，将匹配configuration C；当用户请求“/images/1.gif”时，将匹配configuration D；当用户请求“/documents/1.jpg”时，将匹配configuration E。

表5-7描述了不同URI对应的配置情况。

表5-7　不同URI对应的配置

用户请求的 URI	完整的 URL 地址	匹配的配置
/	http://www.etiantian.org/	configuration A
/index.html	http://www.etiantian.org/	configuration B
/documents/document.html	http://www.etiantian.org/documents/document.html	configuration C
/images/1.gif	http://www.etiantian.org/images/1.gif	configuration D
/documents/1.jpg	http://www.etiantian.org/documents/1.jpg	configuration E

5.8.4 location匹配实战

下面是官方给出的location示例，我们通过该示例来验证不同的location标签生效的顺序。Nginx的配置文件如下：

```
[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;



        server_name  www.etiantian.org etiantian.org;



        root   html/www;



        location / {
           return 401;



        }
        location = / {
            return 402;



        }
        location /documents/ {
            return 403;



        }
        location ^~ /images/ {
            return 404;
```

```
        #matches any query beginning with /images/ and halts searching,


    #so regular expressions will not be checked.
    #匹配任何以
```

/images/开头的查询并且停止搜索。任何正则表达式匹配将不会被检查。

```
    #"^~" 这个前缀的作用是在常规的字符串匹配检查之后，不做正则表达式的检查，即如果最明确的那个字符
```

location配置中有此前缀，那么不会做正则表达式的检查

```
}
        }
        location ~* \.(
```

gif|jpg|jpeg)

```
$ {
      #matches any request ending in gif,


 jpg,


 or jpeg. However,


 all
      #requests to the /images/ directory will be handled.
      #匹配任何以


 gif、
```

jpg 或

```
 jpeg 结尾的请求


            return 500;


        }
        access_log logs/access_www.log main gzip buffer=32k flush=5s;


    }
```

检查语法并使修改的配置生效，命令如下：

```
[root@www extra]# ../../sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@www extra]# ../../sbin/nginx -s reload
```

然后以Linux客户端为例对上述location匹配进行真实测试，配置hosts文件如下：

```
[root@www extra]# tail -1 /etc/hosts
10.0.0.8 www.etiantian.org bbs.etiantian.org blog.etiantian.org etiantian.org
```

实验结果如下：

```
[root@www extra]# curl -s -o /dev/null -I -w "%{http_code}\n" http:
```

```
//www.etiantian.org
402
[root@www extra]# curl -s -o /dev/null -I -w "%{http_code}\n" http:


//www.etiantian.org/
402
[root@www extra]# curl -s -o /dev/null -I -w "%{http_code}\n" http:


//www.etiantian.org/index.html
401
[root@www extra]# curl -s -o /dev/null -I -w "%{http_code}\n" http:


//www.etiantian.org/documents/document.html
403
[root@www extra]# curl -s -o /dev/null -I -w "%{http_code}\n" http:


//www.etiantian.org/images/1.gif
404
[root@www extra]# curl -s -o /dev/null -I -w "%{http_code}\n" http:


//www.etiantian.org/documents/1.jpg
500
[root@www extra]# curl -s -o /dev/null -I -w "%{http_code}\n" http:


//www.etiantian.org/oldboy/
401
[root@www extra]# curl -s -o /dev/null -I -w "%{http_code}\n" http:


//www.etiantian.org/abc/
401
```

表5-8为用户请求的URI相关说明。

表5-8　用户请求的URI说明

用户请求的 URI	设置的状态码	说　明

当为空或 / 时	返回 402，即匹配了：
	location = / {
	return 402;
	}
	和上述官方示例说明表 5-7 匹配（configuration A）一致。
		= 的精确匹配优先级最高，无论放置的顺序如何，它都将优先被匹配并执行

/index.html 或任意不匹配其他 location 的字符串	返回 401，即匹配了：
	location / {
	return 401;
	}
	和上述官方示例说明表 5-7 匹配（configuration B）一致。
		/ 为默认匹配，即如果没有匹配上其他的 location，则最后匹配“默认匹配”的部分

/documents/document.html	返回 403，即匹配了：
	location /documents/ {
	return 403;
	}
	和上述官方示例说明表 5-7 匹配（configuration C）一致。
		此部分为路径匹配，即匹配了路径 /documents/，注意后面的 /documents/1.jpg，这表示没有匹配此处的 location，而是匹配了结尾的 1.jpg

/images/1.gif	返回 404，即匹配了：
	location ^~ /images/ {
	return 404;
	}
	和上述官方示例说明表 5-7 匹配（configuration D）一致。
		此部分为路径匹配，但是前面增加了特殊字符 ^~，所以优先匹配路径，而没有匹配结尾的 1.gif

/documents/1.jpg	返回 500，即匹配了：
	location ~* \.(gif|jpg|jpeg)$ {
	return 500;
	}
	和上述官方示例说明表 5-7 匹配（configuration E）一致。
		此部分匹配了 1.jpg，属于扩展名匹配，虽然有 /documents/，但还是匹配了扩展名

从以上多个location的配置匹配可以看出匹配的优先顺序，具体见表5-9。

表5-9　不用URI及特殊字符组合匹配的顺序说明

顺序	不用 URI 及特殊字符组合匹配	匹配说明
1	“location = / {”	精确匹配 /
2	“location ^~ /images/ {”	匹配常规字符串，不做正则匹配检查
3	“location ~* \.(gif|jpg|jpeg)$ {”	正则匹配
4	“location /documents/ {”	匹配常规字符串，如果有正则，则优先匹配正则
5	“location / {”	所有 location 都不能匹配后的默认匹配

5.9 Nginx rewrite

5.9.1 什么是Nginx rewrite？

和Apache等Web服务软件一样，Nginx rewrite的主要功能也是实现URL地址重写。Nginx的rewrite规则需要PCRE软件的支持，即通过Perl兼容正则表达式语法进行规则匹配。前文在安装Nginx软件时就已经安装了这个PCRE软件，同时也让Nginx支持了rewrite的功能，默认参数编译时，Nginx就会安装支持rewrite的模块，但是，也必须要有PCRE软件的支持。

5.9.2 Nginx rewrite语法

1.rewrite指令语法

指令语法：rewrite regex replacement[flag]；

默认值：none

应用位置：server、location、if

rewrite是实现URL重写的关键指令，根据regex（正则表达式）部分的内容，重定向到replacement部分，结尾是flag标记。下面是一个简单的URL rewrite跳转的例子：

```
rewrite ^/ (

.*)

 http:

//www.etiantian.org/$1 permanent;
```

在上述指令中，rewrite为固定关键字，表示开启一条rewrite匹配规则，regex部分是^/（.*），这是一个正则表达式，表示匹配所有，匹配

成功后跳转到http://www.etiantian.org/$1 。这里的$1是取前面regex部分括号里的内容，结尾的permanent；是永久301重定向标记，即跳转到后面的http://www.etiantian.org/$1 地址上。

2.regex常用正则表达式说明

表5-10为regex常用正则表达式字符的一部分。

3.rewrite指令结尾的flag标记说明

rewrite指令的最后一项参数为flag标记，该标记说明见表5-11。

表5-10　regex常用正则表达式字符

字符	描　　述
\	将后面接着的字符标记为一个特殊字符或一个原义字符或一个向后引用。例如，“\n”匹配一个换行符，序列“\\”和“\$”则匹配“$”
^	匹配输入字符串的起始位置，如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配“\n”或“r”之后的位置
$	匹配输入字符串的结束位置，如果设置了 RegExp 对象的 Multiline 属性，$ 也匹配“\n”或“\r”之前的位置
*	匹配前面的字符零次或多次，例如，ol* 能匹配“o”及“olll”，* 等价于 {0,}
+	匹配前面的字符一次或多次，例如，“ol+”能匹配“ol”及“oll”，但不能匹配“o”，.+ 等价于 {1,}
?	匹配前面的字符零次或一次，例如，“do(es)?”可以匹配“do”或“does”中的“do”，.? 等价于 {0,1} 当该字符紧跟在任何一个其他限制符（*,+?,{n},{n},{n,m}）的后面时，匹配模式是非贪婪模式的，非贪婪模式会尽可能少地匹配所搜索的字符串，而默认的贪婪模式则会尽可能多地匹配所搜索的字符串，例如，对于字符串“oooo”，“o+?”将匹配单个“o”，而“o+”将匹配所有“o”
.	匹配除“\n”之外的任何单个字符，要匹配包括“\n”在内的任何字符，请使用像“[.\n]”这样的模式
(pattern)	匹配括号内的 pattern，并可以在后面获取对应的匹配，常用 $0...$9 属性获取小括号中的匹配内容。要匹配圆括号字符，请使用“\(”或“\)”

表5-11　rewrite指令最后一项参数flag标记的说明

flag 标记符号	说　　明
last	本条规则匹配完成后，继续向下匹配新的 location URI 规则
break	本条规则匹配完成即终止，不再匹配后面的任何规则
redirect	返回 302 临时重定向，浏览器地址栏会显示跳转后的 URL 地址
permanent	返回 301 永久重定向，浏览器地址栏会显示跳转后的 URL 地址

在以上的flag标记中，last和break用来实现URL重写，浏览器地址栏的URL地址不变，但在服务器端访问的程序及路径发生了变化。redirect和permanent用来实现URL跳转，浏览器地址栏会显示跳转后的URL地址。

last和break标记的实现功能类似，但二者之间有细微的差别，使用alias指令时必须用last标记，使用proxy_pass指令时要使用break标记。last标记在本条rewrite规则执行完毕后，会对其所在的server{.....}标签重新发起请求，而break标记则会在本条规则匹配完成后，终止匹配，不再匹配后面的规则。

更多知识请参考后面关于Nginx反向代理负载均衡的章节。

5.9.3 Nginx rewrite的企业应用场景

Nginx的rewrite功能在企业里应用非常广泛：

·可以调整用户浏览的URL，使其看起来更规范，合乎开发及产品人员的需求。

·为了让搜索引擎收录网站内容，并让用户体验更好，企业会将动态URL地址伪装成静态地址提供服务。

·网站换新域名后，让旧域名的访问跳转到新的域名上，例如：让京东的360buy换成了jd.com。

·根据特殊变量、目录、客户端的信息进行URL跳转等。

5.9.4 Nginx rewrite 301跳转

以往我们是通过别名的方式实现etiantian.org和www.etiantian.org 访问同一个地址的，事实上，除了这个方式以外，还可以使用Nginx rewrite 301跳转的方式来实现。实现的配置如下：

```
[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;


        server_name  etiantian.org;


        rewrite ^/ (

.*)

 http:

//www.etiantian.org/$1 permanent;


        #<==当用户访问

etiantian.org及下面的任意内容时，都会通过这条

rewrite跳转到
```

```
www.etiantian.org对应的地址


    }
     server {
         listen       80;


         server_name  www.etiantian.org;


         location / {
             root   html/www;


             index  index.html index.htm;


         }
         access_log logs/access_www.log main gzip buffer=32k flush=5s;


     }
```

浏览器访问效果见图5-14。

图5-14　Nginx rewrite 301跳转图

5.9.5 实现不同域名的URL跳转

这一节通过示例来讲解不同域名的URL跳转。

例1：实现访问http://blog.etiantian.org 时跳转到http://www.etiantian.org/blog/oldboy.html 。

外部跳转时使用这种方法，可让浏览器地址变为跳转后的地址，另外，要事先设置http://www.etiantian.org/blog/oldboy.html 有结果输出，不然会出现401等权限错误。

1.配置Nginx rewrite规则

跳转前，http://blog.etiantian.org 对应站点的配置如下：

```
[root@www extra]# cat blog.conf
    server {
        listen       80;



        server_name  blog.etiantian.org;



        location / {
            root   html/blog;



            index  index.html index.htm;
```

```
        }
        if  (

 $http_host ~* "^(

.*)

\.etiantian\.org$")

 {
            set $domain $1;

            rewrite ^(

.*)

 http:

//www.etiantian.org/$domain/oldboy.html break;

            }
    }
```

要配置的规则内容为：

```
        if  (

 $http_host ~* "^(

.*)
```

```
\.etiantian\.org$")

 {
           set $domain $1;

           rewrite ^(
.*)

 http:

//www.etiantian.org/$domain/oldboy.html break;

           }
```

跳转后，http://www.etiantian.org/blog/oldboy.html 地址对应的站点配置如下：

```
[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;

        server_name  www.etiantian.org etiantian.org;

        location / {
            root   html/www;

            index  index.html index.htm;
```

```
        }
        access_log logs/access_www.log main gzip buffer=32k flush=5s;


    }
```

2.客户端访问测试效果

Windows下浏览器访问跳转后的结果如图5-15所示。

图5-15　实现不同域名的URL跳转效果图

例2：实现访问http://etiantian.org/bbs 时跳转到http://bbs.etiantian.org。

在etiantian.org下设置Nginx rewrite规则，如下：

```
[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;


        server_name  www.etiantian.org etiantian.org;


        location / {
```

```
            root   html/www;

            index  index.html index.htm;

        }
        rewrite ^(
.*)
/bbs/ http:
//bbs.etiantian.org break;

        access_log logs/access_www.log main gzip buffer=32k flush=5s;

    }
[root@www extra]# cat bbs.conf
#bbs virtualhost by oldboy
    server {
        listen       80;

        server_name  bbs.etiantian.org;

        location / {
            root   html/bbs;

            index  index.html index.htm;

        }
    }
```

访问前后的效果对比如图5-16所示

图5-16　不同域名跳转效果图

以下是有关rewrite特殊flag标记last与break的说明：

在根location（即location/{……}）中或server{……}标签中编写rewrite规则，建议使用last标记，而在普通的location（例location/oldboy/{……}或if{}）中编写rewrite规则，则建议使用break标记。

5.10　Nginx访问认证

有时，在实际工作企业要求我们为网站设置访问账号和密码权限，这样操作后，只有拥有账号密码的用户才可以访问网站内容，访问验证效果如图5-17所示。

图5-17　Nginx身份验证窗口

这种使用账号密码才可以访问网站的功能主要应用在企业内部人员访问的地址上，例如：企业网站后台、MySQL客户端phpmyadmin、企业内部的CRM、WIKI网站平台等。下面介绍一个配置示例：

```
location / {
    auth_basic           "closed site";


    auth_basic_user_file conf/htpasswd;
```

```
}
```

其中，有两个参数需要说明。

·auth_basic

语法：auth_basic string|off；

默认值：auth_basic off；

使用位置：http、server、location、limit_except

·auth_basic_user_file

语法：auth_basic_user_file file；

默认值：—

使用位置：http、server、location、limit_except

auth_basic_user_file参数后接认证密码文件，file的内容如下：

```
# comment
name1:


password1
name2:


password2:
```

```
comment
name3:


password3
```

可以使用Apache自带的“htpasswd”或“openssl passwd”命令设置用户和密码到认证文件里，注意，密码是加密的。相关内容见官网：http://nginx.org/en/docs/http/ngx_http_auth_basic_module.html

下面进行配置实战。选择一个虚拟主机配置在server标签里，配置内容如下：

```
    auth_basic            "oldboy training";


    auth_basic_user_file /application/nginx/conf/htpasswd;
```

注意：/application/nginx/conf/是认证文件路径，htpasswd是存放账号及密码的文件。

www虚拟主机的位置配置如下：

```
[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;
```

```
        server_name  www.etiantian.org etiantian.org;


        location / {
            root   html/www;


            index  index.html index.htm;


            auth_basic           "oldboy training";


            auth_basic_user_file /application/nginx/conf/htpasswd;


        }
        access_log logs/access_www.log main gzip buffer=32k flush=5s;


    }
```

说明：

·auth_basic　"oldboy training"；用于设置认证提示字符串"oldboy training"。

·auth_basic_user_file/application/nginx/conf/htpasswd；用于设置认证的密码文件，即用户输入账户密码后，Nginx会到这个文件中对比用户的输入是否正确，进而决定是否允许用户访问网站。

生成认证账号和密码的步骤如下。

1）获取htpasswd设置账号和密码，命令如下：

```
/usr/bin/which:

 no htpasswd in (

/usr/local/sbin:

/usr/local/bin:

/sbin:

/bin:

/usr/sbin:

/usr/bin:

/root/bin)


[root@www extra]# yum install httpd -y
[root@www extra]# which htpasswd
/usr/bin/htpasswd
```

2）创建账号密码，此账号密码就是用户访问网站时需要输入的。

操作命令如下：

```
htpasswd -bc /application/nginx/conf/htpasswd oldboy 123456
chmod 400 /application/nginx/conf/htpasswd
chown nginx /application/nginx/conf/htpasswd
```

操作过程如下：

```
[root@www extra]# htpasswd -bc /application/nginx/conf/htpasswd oldboy 123456
Adding password for user oldboy
[root@www extra]# chmod 400 /application/nginx/conf/htpasswd
[root@www extra]# chown nginx /application/nginx/conf/htpasswd
[root@www extra]# cat /application/nginx/conf/htpasswd
oldboy:


HWSXCFixaoiZ6      #<==看到了吧，密码是加密的。
```

3）重新加载Nginx使配置修改生效，命令如下：

```
/application/nginx/sbin/nginx -t
/application/nginx/sbin/nginx -s relaod
```

4）进行浏览器访问测试，见图5-18。

根据提示输入账号密码后回车，如图5-19所示。

验证后可以访问到网站内容，如图5-20所示。

如果密码不对就不能访问网站内容，而是提示重新输入账号密码，超过指定的次数或放弃密码验证时会给出如图5-21所示的报错信息。

图5-18　最终Nginx身份验证窗口

图5-19　输入账号及密码

图5-20　通过验证后的浏览器内容

图5-21　未通过验证的浏览器报错

5.11 Nginx相关问题的解答

问题1：Tengine和Nginx是什么关系

Tengine是淘宝开源Nginx的分支，官方站点为http://tengine.taobao.org/ 。

问题2：说明访问Nginx时出现状态码“403 forbidden”的原因。

原因之一是Nginx配置文件里没有配置默认首页参数，或者首页文件在站点目录下没有如下内容：

```
index  index.php index.html index.htm;
```

问题模拟示例如下：

```
[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;


        server_name  www.etiantian.org;


        location / {
            root   html/www;
```

```
            #index  index.html index.htm;


          #<==注释首页文件配置



        }
        access_log off;



    }
[root@www extra]# ../../sbin/nginx -s reload
[root@www extra]# tail -1 /etc/hosts
10.0.0.8 www.etiantian.org bbs.etiantian.org blog.etiantian.org etiantian.org
[root@www extra]# ll ../../html/www/                    总用量


 12
drwxr-xr-x 2 root root 4096 4月


  15 14:


20 blog
-rw-r--r-- 1 root root    4 4月


  17 17:


11 index.html     #<==存在首页文件



drwxr-xr-x 2 root root 4096 4月


  15 14:


19 oldboy
[root@www extra]# curl -I -s 10.0.0.8|head -1
HTTP/1.1 403 Forbidden     #<==Nginx没有指定首页文件的参数，因此访问
```

Nginx时不会把

index.html当成首页，所以报

403错误

原因之二是站点目录下没有配置文件里指定的首页文件如下。

```
[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;



        server_name  www.etiantian.org;



        location / {
            root   html/www;



            index  index.html index.htm;


          #<==配置首页文件配置



        }
        access_log off;



    }
[root@www extra]# ../../sbin/nginx -s reload
[root@www extra]# rm -f ../../html/www/index.html      #<==删除物理首页文件
```

```
[root@www extra]# curl -I -s 10.0.0.8|head -1
HTTP/1.1 403 Forbidden
```

若是因以上两个出现的问题，可以通过一个参数来解决，命令如下：

```
autoindex on;



[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;



        server_name  www.etiantian.org;



        location / {
            root   html/www;



            autoindex on;

     #<==当找不到首页文件时，会展示目录结构，这个功能一

                                般不要用，除非有需求



        }
        access_log off;



    }
```

其效果如图5-22所示。

原因之三是站点目录或内部的程序文件没有Nginx用户访问权限，如下：

```
[root@www extra]# echo test > ../../html/www/index.html
[root@www extra]# chmod 700 ../../html/www/index.html
                          #<==设置



700让



nginx用户无权读取



[root@www extra]# ls -l ../../html/www/index.html
-rwx------ 1 root root 5 4月



  17 17:



15 ../../html/www/index.html
[root@www extra]# curl -I -s 10.0.0.8|head -1
HTTP/1.1 403 Forbidden           #<==403错误



[root@www extra]# chmod 755 ../../html/www/index.html
                          #<==设置



755让



nginx用户有权读取



[root@www extra]# curl -I -s 10.0.0.8|head -1
HTTP/1.1 200 OK                #<==200 OK了
```

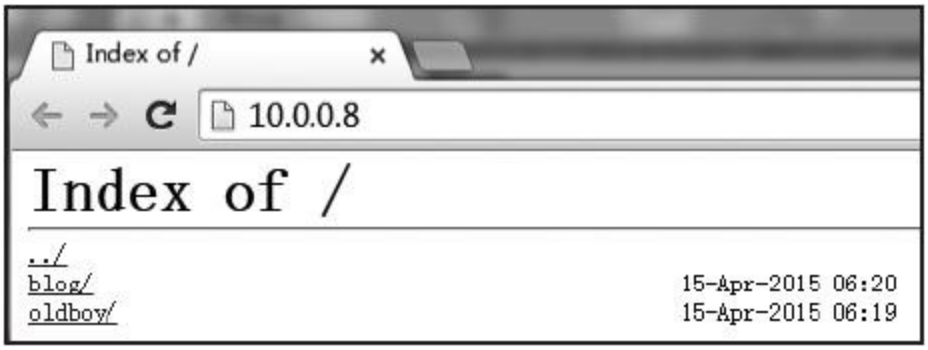


图5-22　页面效果

原因之四是Nginx配置文件中设置了allow、deny等权限控制，导致客户端没有访问权限，如下：

```
[root@www extra]# cat www.conf
#www virtualhost by oldboy
    server {
        listen       80;



        server_name  www.etiantian.org;



        location / {
            root   html/www;



            index  index.html index.htm;



            allow 192.168.1.0/24;



            deny all;
```

```
        }
        access_log off;



    }
[root@www extra]# curl -I -s 10.0.0.8|head -1
HTTP/1.1 200 OK                #<==设置


755让


nginx用户有权读取



[root@www extra]# ../../sbin/nginx -s reload
[root@www extra]# curl -I -s 10.0.0.8|head -1
HTTP/1.1 403 Forbidden
```

提示：上述出现403错误的原因并不是Nginx才有的，Apache服务的Forbidden 403问题同样也是这几个原因导致的，只是参数细节略有区别而已，相关资料请见http://oldboy.blog.51cto.com/2561410/581383 。

5.12 本章重点回顾

1）Nginx的特性优点。

2）主流Web动态静态性能对比。

3）Apache select和Nginx epoll模型的区别（面试常考）。

4）虚拟主机概念及类型分类详解。

5）基于域名和端口虚拟主机的介绍及搭建。

6）Nginx错误、访问日志，以及访问日志切割。

7）Nginx访问状态信息介绍及配置实践。

8）Nginx location介绍及配置实践。

9）Nginx rewrite介绍及配置实践。

10）Nginx Web访问认证介绍及配置实践。

第6章　企业级LNMP环境应用实践

6.1　LNMP应用环境

6.1.1　LNMP介绍

大约在2010年以前，互联网公司最常用的经典Web服务环境组合就是LAMP（即Linux、Apache、MySQL、PHP），近几年随着Nginx Web服务的逐渐流行，又出现了新的Web服务环境组合——LNMP或LEMP，其中LNMP为Linux、Nginx、MySQL、PHP等首字母的缩写，而LEMP中的E则表示Nginx，它取自Nginx名字的发音（engine x）。现在，LNMP已经逐渐成为国内大中型互联网公司网站的主流组合环境，因此，我们必须熟练掌握LNMP环境的搭建、优化及维护方法。

6.1.2 LNMP组合工作流程

在深入学习LNMP组合之前，有必要先来了解一下LNMP环境组合的基本原理，也就是它们之间到底是怎样互相调度的？

当LNMP组合工作时，首先是用户通过浏览器输入域名请求Nginx Web服务，如果请求是静态资源，则由Nginx解析返回给用户；如果是动态请求（.php结尾），那么Nginx就会把它通过FastCGI接口（生产常用方法）发送给PHP引擎服务（FastCGI进程php-fpm）进行解析，如果这个动态请求要读取数据库数据，那么PHP就会继续向后请求MySQL数据库，以读取需要的数据，并最终通过Nginx服务把获取的数据返回给用户，这就是LNMP环境的基本请求顺序流程（如图6-1和图6-2所示）。这个请求流程是企业使用LNMP环境的常用流程。

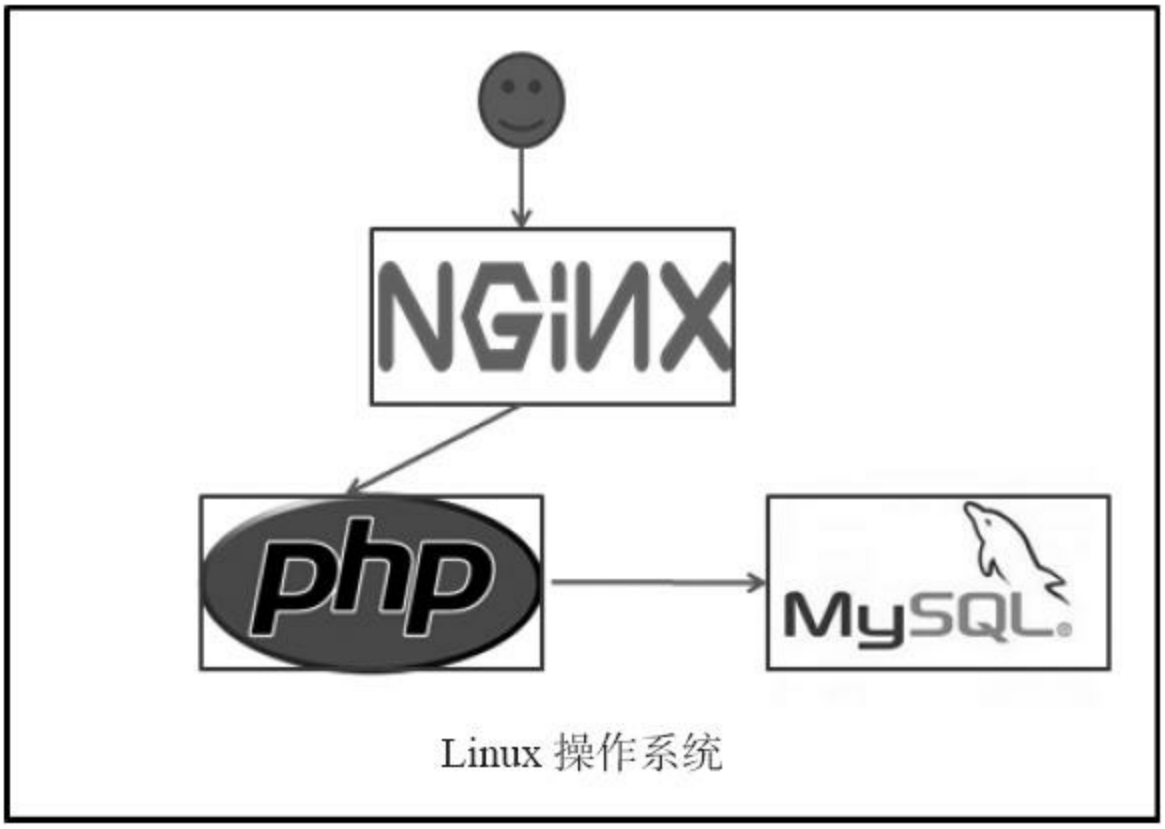


图6-1　LNMP组合调用关系逻辑图

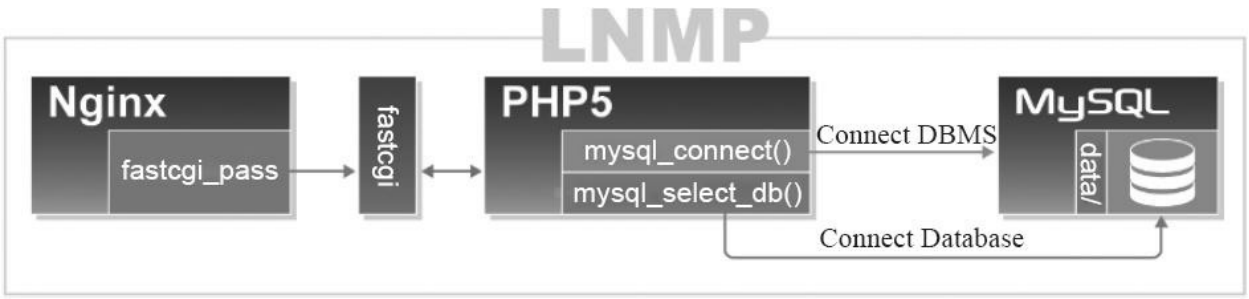


图6-2　LNMP组合FastCGI方式调用PHP、MySQL的关系逻辑图

6.2 LNMP之MySQL数据库

6.2.1 MySQL数据库介绍

MySQL是互联网领域里非常重要的、深受广大用户欢迎的一款开源关系型数据库软件，由瑞典MySQL AB公司开发与维护。2006年，MySQL AB公司被SUN公司收购，2008年，SUN公司又被传统数据数据库领域大佬甲骨文（Oracle）公司收购。因此，MySQL数据库软件目前属于Oracle公司，但仍是开源的，Oracle公司收购MySQL的战略意图显而易见，其自身的Oracle数据库继续服务于传统大中型企业，而利用收购的MySQL抢占互联网领域数据库份额，完成其战略布局。

MySQL是一种关系型数据库管理软件，关系型数据库的特点是将数据保存在不同的二维表中，并且将这些表放入不同的数据库中，而不是把所有数据统一放在一个大仓库里，这样的设计增加了MySQL的读取速度，灵活性和可管理性也得到了很大提高。访问及管理MySQL数据库的最常用标准化语言为SQL结构化查询语言。

6.2.2 为什么选择MySQL数据库

目前，绝大多数使用Linux操作系统的互联网企业都使用MySQL作为后端的数据库，从大型的BAT门户，到电商门户平台、分类门户平台等无一例外。那么，MySQL数据库到底有哪些优势和特点，让大家毫不犹豫地选择它呢？

原因可能有以下几点：

·性能卓越、服务稳定，很少出现异常宕机。

·开放源代码且无版权制约，自主性强、使用成本低。

·历史悠久，社区及用户非常活跃，遇到问题，可以很快获取到帮助。

·软件体积小，安装使用简单，并且易于维护，安装及维护成本低。

·支持多种操作系统，提供多种API接口，支持多种开发语言，特别是对流行的PHP语言无缝支持。

·品牌口碑效应，使得企业无需考虑就直接用之。

更多的MySQL介绍见后面的章节。

6.2.3　安装MySQL数据库

1.安装概览

MySQL有几种不同的产品线，且每种产品线又有很多不同的版本，这里选择当前企业使用最广的社区版MySQL 5.5系列作为LNMP的组合环境数据库平台。

MySQL的安装方法也有很多，常见的方法见表6-1。

表6-1　企业场景MySQL安装方式一览

序号	MySQL 安装方式	特点说明
1	yum/rpm 包安装	特点是简单、速度快，但是没法定制安装，入门新手常用这种方式
2	二进制安装	解压软件，简单配置后就可以使用，不用安装，速度较快，专业 DBA 喜欢这种方式。软件名如：mysql-5.5.32-linux2.6-x86_64.tar.gz
3	源码编译安装	特点是可以定制安装[⊖]，但是安装时间长，例如：字符集安装路径，等等。软件名如：mysql-5.5.32.tar.gz
4	源码软件结合 yum/rpm 安装	把源码软件制作成符合要求的 rpm，放到 yum 仓库里，然后通过 yum 来安装。结合了上面 1 和 3 的优点，即安装快速，可任意定制参数，但是安装者也需要具备更深能力。本书结尾有 rpm 定制包的内容介绍

注：提示：MySQL 5.5产品系列和早期的MySQL 5.0/5.1系列属于不同的产品线，因此，安装方式不同，后文有讲解。

安装MySQL的注意事项如下：

1）建议和前面章节介绍的Nginx服务安装在同一台机器上。

2）重视操作过程的报错输出，有错误要解决掉再继续，不能忽略掉编译中的错误。

2.安装步骤介绍

1）创建mysql用户的账号

首先以root身份登录到Linux系统中，然后执行如下命令创建mysql组及用户账号：

```
[root@www ~]# groupadd mysql
[root@www ~]# useradd -s /sbin/nologin -g mysql -M mysql
```

useradd命令的参数简要说明如下：

·-s/sbin/nologin表示禁止该用户登录，只需要角色存在即可，加强安全。

·-g mysql指定mysql用户属于mysql组。

·-M表示不创建用户家目录，因为没有需要。

·groupadd和useradd这两条命令也可以用useradd-s/sbin/nologin-M mysql替代。

下面检查刚刚创建的mysql用户和组，命令如下：

```
[root@www ~]# tail -1 /etc/passwd
mysql:


x:


501:


501: :


/home/mysql:


/sbin/nologin
[root@www ~]# id mysql
uid=501 (


mysql)


 gid=501 (


mysql)


 组


=501 (


mysql)
```

检查输出，可以看到mysql用户和组已经成功创建。

现在建立存放所有安装软件的固定目录，命令如下。老男孩喜欢用

普通用户家目录，如果已经有了直接进入即可。

```
[root@www ~]# mkdir -p /home/oldboy/tools
[root@www ~]# cd /home/oldboy/tools/
```

2）获取MySQL软件包

MySQL软件包的下载地址为：http://mysql.ntu.edu.tw/Downloads/，如果地址已变更无法下载，可以去官方下载，或者通过老男孩的云空间下载。

读者可以通过ftp或rz命令从本地把已经下载好的MySQL软件包上传到Linux系统中，或者找到下载的网址直接使用wget命令下载。

本例以rz命令从本地安装来演示部署过程。执行时如提示无rz命令，可先执行yum install lrzsz-y安装之。执行结果如下：

```
[root@www tools]# cd /home/oldboy/tools/
[root@www tools]# rz
rz waiting to receive.开始


 zmodem 传输。


 按


 Ctrl+C 取消


正在传输
```

```
 mysql-5.5.32-linux2.6-x86_64.tar.gz...
  100%   182346 KB 5525 KB/s 00:


00:


11       0 错误


[root@www tools]# ls -sh mysql-5.5.32-linux2.6-x86_64.tar.gz
179M mysql-5.5.32-linux2.6-x86_64.tar.gz
```

提示：本例以MySQL二进制的方式来讲解，后面章节会使用编译方式指导大家安装。在生产场景中，两种方法都是可以用的，其应用场景一般没什么太大差别。不同之处在于，二进制的安装包较大，名字和源码包也有些区别。

MySQL二进制安装包和源码包名称见表6-2。

表6-2　MySQL二进制和源码包

MySQL 软件	软件名
MySQL 二进制安装包	mysql-5.5.32-linux2.6-x86_64.tar.gz
MySQL 源码安装包	mysql-5.5.32.tar.gz

3）采用二进制方式安装MySQL

（1）解压并移动MySQL二进制软件包到指定的安装路径，命令如下：

```
[root@www tools]# tar xf mysql-5.5.32-linux2.6-x86_64.tar.gz
                                    #<==解压
```

```
[root@www tools]# mkdir -p /application/      #<==创建安装目录，如果有就不用创建了


[root@www tools]# mv mysql-5.5.32-linux2.6-x86_64 /application/mysql-5.5.32
                                    #<==移动并改名目录
```

（2）创建软链接，生成去掉版本号的访问路径并查看，命令如下：

```
[root@www tools]# ln -s /application/mysql-5.5.32/ /application/mysql
                                    #<==设置软链接


[root@www tools]# ls -l /application/          #<==查看配置后的结果


total 8
lrwxrwxrwx  1 root root   26 Apr 22 15:

32 mysql -> /application/mysql-5.5.32/
                                    #<==MySQL软链接


drwxr-xr-x 13 root root 4096 Apr 22 15:

32 mysql-5.5.32
                                    #<==MySQL安装路径


lrwxrwxrwx  1 root root   24 Apr  3 12:

30 nginx -> /application/nginx-1.6.2
```

```
                                                    #<==和

Nginx一台机器

drwxr-xr-x 11 root root 4096 Apr  6 15：

34 nginx-1.6.2提示：二进制安装包，仅需要解压就可以了，不需要执行

cmake/configure，

make，

make install等过程
```

当安装LNMP一体化环境时，MySQL数据库要装在Nginx所在的机器上。如果MySQL和Nginx不在一台机器上，那么，Nginx服务器上的MySQL数据库软件包只要解压移动到/application目录，改名为mysql就可以了，不需要进行后面的初始化配置。

在非一体的LNMP环境（即Nginx和MySQL不在一台机器上），编译PHP环境时，也是需要MySQL数据库环境的，但是高版本的PHP，例如5.3版本以上，内置了PHP需要的MySQL程序，因此，对于此类版本就不需要在Nginx服务器上安装MySQL软件了，只需要在编译PHP时指定相关参数即可。这个PHP的编译参数为--with-mysql=mysqlnd，表示PHP程序在编译时会调用内置的MySQL的库，详细说明见后文。

4）初始化MySQL配置文件my.cnf

命令如下：

```
[root@www tools]# cd /application/mysql/
[root@www mysql]# ls -l support-files/*.cnf
-rw-r--r-- 1 7161 wheel  4691 Jun 19  2013 support-files/my-huge.cnf
-rw-r--r-- 1 7161 wheel 19759 Jun 19  2013 support-files/my-innodb-heavy-4G.cnf
-rw-r--r-- 1 7161 wheel  4665 Jun 19  2013 support-files/my-large.cnf
-rw-r--r-- 1 7161 wheel  4676 Jun 19  2013 support-files/my-medium.cnf
-rw-r--r-- 1 7161 wheel  2840 Jun 19  2013 support-files/my-small.cnf
[root@www mysql]# /bin/cp support-files/my-small.cnf /etc/my.cnf
```


提示：

·support-files下有my.cnf的各种配置样例。懂英文的读者可以阅读一下，里面的注释非常详细。

·使用cp全路径/bin/cp，可实现拷贝而不出现替换提示，即如果有重名文件会直接覆盖。

·本例为测试安装环境，因此，选择参数配置小的my-small.cnf配置模版，如果是生产环境可以根据硬件选择更高级的配置文件，上述配置文件模版对硬件的要求从低到高依次为：

```
my-medium.cnf==>my-small.cnf==>my-large.cnf==>my-huge.cnf==>my-innodb-heavy-4G.cnf
```

5）初始化MySQL数据库文件

初始化命令如下：

```
[root@www mysql]# mkdir -p /application/mysql/data
                #<==建立
```

MySQL数据文件目录

```
[root@www mysql]# chown -R mysql.mysql /application/mysql/
                #<==授权
```

mysql用户管理

MySQL的安装目录

```
[root@www mysql]# /application/mysql/scripts/mysql_install_db --basedir=
/application/mysql --datadir=/application/mysql/data --user=mysql
                #<==初始化
```

MySQL数据库文件，会有很多信息提示，如果没有

ERROR级别

的错误，会有两个

OK的字样，表示初始化成功，否则就要解决初始化的问题

WARNING:

```
 The host 'www' could not be looked up with resolveip.
This probably means that your libc libraries are not 100 % compatible
with this binary MySQL version. The MySQL daemon,
```

```
 mysqld,
```

```
 should work
normally with the exception that host name resolving will not work.
This means that you should use IP addresses instead of hostnames
when specifying MySQL privileges !



Installing MySQL system tables...
OK
Filling help tables...
OK
To start mysqld at boot time you have to copy
support-files/mysql.server to the right place for your system
PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !



To do so,



 start the server,



 then issue the following commands:



/application/mysql/bin/mysqladmin -u root password 'new-password'
/application/mysql/bin/mysqladmin -u root -h www password 'new-password'
Alternatively you can run:



/application/mysql/bin/mysql_secure_installation
which will also give you the option of removing the test
databases and anonymous user created by default.  This is
strongly recommended for production servers.
See the manual for more instructions.
You can start the MySQL daemon with:



cd /application/mysql ;



 /application/mysql/bin/mysqld_safe &
You can test the MySQL daemon with mysql-test-run.pl
cd /application/mysql/mysql-test ;
```

```
 perl mysql-test-run.pl
Please report any problems with the /application/mysql/scripts/mysqlbug script!
```

以上命令的主要作用是生成了如下数据库文件：

```
[root@www mysql]# tree /application/mysql/data/
/application/mysql/data/
|-- mysql
|   |-- columns_priv.MYD
|   |-- columns_priv.MYI
|   |-- columns_priv.frm
|   |-- db.MYD
|   |-- db.MYI
|   |-- db.frm
|   |-- event.MYD
|   |-- event.MYI
|   |-- event.frm
|   |-- func.MYD
|   |-- func.MYI
|   |-- func.frm
|   |-- general_log.CSM
|   |-- general_log.CSV...省略若干...
```

这些MySQL数据文件是MySQL正确运行所必需的基本数据库文件，其功能是对MySQL权限、状态等进行管理。

3.初始化故障排错集锦

错误示例1：WARNING：The host'mysql'could not be looked up with resolveip.

需修改主机名的解析，使其和uname-n一样，修改后的结果如下：

```
[root@www mysql]# grep `uname -n` /etc/hosts
127.0.0.1               www localhost.localdomain localhost
```

错误示例2：ERROR：1004 Can't create file'/tmp/#sql300e_1_0.frm'（errno：13）

下面是初始化数据库时可能会遇到的错误，原因是/tmp目录的权限有问题。

解决办法为处理/tmp目录，如下：

```
[root@www mysql]# ls -ld /tmp
drwxrwxrwt. 4 root root 4096 Apr 22 16:


43 /tmp
[root@www mysql]# chmod -R 1777 /tmp/
```

此故障必须解除，否则，后面会出现登录不了数据库等各种问题。

6.2.4　配置并启动MySQL数据库

1）设置MySQL启动脚本，命令如下：

```
[root@www mysql]# cp support-files/mysql.server  /etc/init.d/mysqld
#<==拷贝


MySQL启动脚本到


MySQL的命令路径



[root@www mysql]# chmod +x /etc/init.d/mysqld
#<==使脚本可执行
```

2）MySQL二进制默认安装路径是/usr/local/mysql，启动脚本里是/usr/local/mysql的路径都需要替换，命令如下：

```
[root@www mysql]# sed -i 's#/usr/local/mysql#/application/mysql#g' /application/mysc
```

3）启动MySQL数据库，命令如下：

```
[root@www mysql]# /etc/init.d/mysqld start
Starting MySQL.. SUCCESS!
```

这是启动数据库的规范方法之一，还可以使用/application/mysql/bin/mysqld_safe--user=mysql&启动。这个命令结尾的“&”符号，作用是在后台执行MySQL服务，命令执行完还需要按下回车才能进入命令行状态。


注意：如果已执行上面启动命令，还想尝试下面的命令，请先执行/etc/init.d/mysqld stop或killalll mysqld（生产环境尽量不要用killall）结束MySQL进程。

4）检查MySQL数据库是否启动，命令如下：

```
[root@www mysql]# netstat -lntup|grep mysql
tcp      0    0 0.0.0.0:

3306        0.0.0.0:

*      LISTEN      9446/mysqld
```

如发现3306端口没起来，请tail-100/application/mysql/data/机器名.err查看日志信息，看是否有报错信息，然后根据相关错误提示进行调试。经常查看服务运行日志是个很好的习惯，也是高手的习惯，你要不要成为高手？嘿！

5）查看MySQL数据库启动结果日志，命令如下：

```
[root@www mysql]# tail -10 /application/mysql/data/www.err
InnoDB:
```

```
 Creating foreign key constraint system tables
InnoDB:


 Foreign key constraint system tables created
150422 17:


09:


47  InnoDB:


 Waiting for the background threads to start
150422 17:


09:


48 InnoDB:


 5.5.32 started;


 log sequence number 0
150422 17:


09:


48 [Note] Server hostname  (


bind-address) :


 '0.0.0.0';


 port:
```

 3306
150422 17:

09:

48 [Note]   - '0.0.0.0' resolves to '0.0.0.0';

150422 17:

09:

48 [Note] Server socket created on IP:

 '0.0.0.0'.
150422 17:

09:

48 [Note] Event Scheduler:

 Loaded 0 events
150422 17:

09:

48 [Note] /application/mysql/bin/mysqld:

 ready for connections.
Version:

 '5.5.32' socket:

```
 '/tmp/mysql.sock'  port:


 3306  MySQL Community Server  (


GPL)
```

本例只查看了错误日志的命令及错误日志中的内容，省略了大部分日志内容。

6）设置MySQL开机自起动，命令如下：

```
[root@www mysql]# chkconfig --add mysqld
[root@www mysql]# chkconfig mysqld on
[root@www mysql]# chkconfig --list mysqld
mysqld          0:


off   1:


off   2:


on    3:


on    4:


on    5:


on    6:


off
```

提示：也可以将启动命令/etc/init.d/mysqld start放到/etc/rc.local里面。

7）配置mysql命令的全局使用路径，命令如下：

```
[root@www mysql]# echo 'export PATH=/application/mysql/bin:


$PATH' >>/etc/profile
#<==注意，


echo后是单引号，双引号是不行滴。这是为什么呢？本书后文讲到


shell编程时会介绍。



[root@www mysql]# tail -1 /etc/profile
export PATH=/application/mysql/bin:


$PATH
[root@www mysql]# source /etc/profile
#<==执行


source使上一行添加到


/etc/profile中的内容直接生效。



#<==以上两条命令的用途为定义


mysql全局路径，实现在任意路径执行


mysql命令。
```

```
[root@www mysql]# echo $PATH
/application/mysql/bin:


/usr/local/sbin:


/usr/local/bin:


/sbin:


/bin:


/usr/sbin:


/usr/bin:


/root/bin
#<==查看设置的结果
```

提示：如果不配置mysql的全局路径，则无法直接敲mysql等命令管理数据库，而只能采用/application/mysql/bin/mysql这样的带着路径的方式敲命令。 还有另外一个办法，就是把/application/mysql/bin下面的命令拷贝到已经是全局系统命令路径的/usr/local/sbin下。

由全局路径配置不当导致的问题案例见文章：

http://oldboy.blog.51cto.com/2561410/1122867 。

8）登录MySQL测试，命令如下：

```
[root@www mysql]# mysql      #<==直接敲就进入数据库了，而且身份还是

root。

Welcome to the MySQL monitor.  Commands end with ;

 or \g.
Your MySQL connection id is 1
Server version:

 5.5.32 MySQL Community Server  (

GPL)

Copyright  (

c)

 2000,

 2013,

 Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;

' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
#<==你还可以使用如下三种命令登录：
```

```
mysql -uroot -p,


mysql -uroot,


mysql -uroot -p '密码


'
mysql> show databases;


     #<==查看当前所有的数据库



+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set （


0.00 sec）


mysql> select user（）；


     #<==查看当前的登录用户



+----------------+
| user（）


         |
+----------------+
| root@localhost |
```

```
+----------------+
1 row in set  (


0.00 sec)



mysql> quit
Bye
```

登录故障排查说明：

如果这里提示无法登录，排除了数据库启动问题后，则很可能是数据库初始化文件有问题 。例如：

```
[root@www mysql]# mysql
ERROR 1045  (


28000) :


 Access denied for user 'root'@'localhost'  (


using password:


 NO)
```

上述问题的3种可能原因及其解决方法如下：

1）删除并重新初始化数据库，出现此问题一般都是因为数据库初始化有问题。

2）或者是MySQL数据库文件损坏了，也可能是MySQL数据目录权限有问题。

3）检查主机名对应主机IP解析是否正确。

MySQL安装完成后，默认情况下，管理员账号root是无密码的，这个必须设置。

6.2.5 MySQL安全配置

1）为MySQL的root用户设置密码，命令如下：

```
[root@www mysql]# mysqladmin -u root password 'oldboy123'     #<==更改默认密码。

[root@www mysql]# mysql                          #<==无法直接登录了

ERROR 1045 （

28000）：

 Access denied for user 'root'@'localhost' （

using password：

 NO）

[root@www mysql]# mysql -uroot -p                #<==新的登录方式

Enter password：

                                 #<==输入新密码

oldboy123
Welcome to the MySQL monitor.  Commands end with ；
```

```
 or \g.
Your MySQL connection id is 4
Server version:


 5.5.32 MySQL Community Server  (


GPL)



Copyright  (


c)


 2000,


 2013,


 Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;


' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

2）清理无用的MySQL用户及库，命令如下：

```
mysql> select user,


host from mysql.user;



+------+-----------+
| user | host      |
```

```
+------+-----------+
| root | 127.0.0.1 |
| root | : :


1       |
|      | localhost |
| root | localhost |
|      | www       |
| root | www       |
+------+-----------+
6 rows in set  (


0.00 sec)



mysql> drop user "root"@": :


1";



Query OK,


 0 rows affected  (


0.00 sec)



mysql> drop user ""@"localhost";



Query OK,


 0 rows affected  (


0.00 sec)
```

```
mysql> drop user ""@"www";

Query OK,

 0 rows affected  (

0.00 sec)

mysql> drop user "root"@"www";

Query OK,

 0 rows affected  (

0.00 sec)

mysql> select user,

host from mysql.user;

+------+-----------+
| user | host      |
+------+-----------+
| root | 127.0.0.1 |
| root | localhost |
+------+-----------+
2 rows in set  (

0.00 sec)

mysql>
```

```
mysql> flush privileges;

Query OK,

 0 rows affected  (

0.00 sec)
```

下面介绍一个故障排查案例：使用drop命令删除不了用户。估计不少同学都有过类似的经历。

可能是大写及特殊Linux主机名导致的，如下：

```
mysql> drop user ''@'MySQL';

ERROR 1396  (

HY000) :

 Operation DROP USER failed for ' '@'mysql'
```

解决办法如下：

```
mysql> delete from mysql.user where user='' and host='MySQL';

Query OK,
```

```
 1 row affected （

0.00 sec）

mysql> flush privileges；

Query OK，

 0 rows affected （

0.00 sec）
```

现在可删除无用的数据库了，如下：

```
mysql> drop database test；

Query OK，

 0 rows affected （

0.02 sec）

mysql> show databases；

+--------------------+
| Database           |
```

```
+--------------------+
| information_schema |
| mysql              |
+--------------------+
2 rows in set （


0.00 sec）
```

好了，初学者能学到这里就很好了。恭喜，可以休息一下！以便更好地学习后面的技术。关于MySQL，后面还会有介绍，下一节先来了解一下FastCGI。

6.3 FastCGI介绍

6.3.1 什么是CGI

CGI的全称为“通用网关接口”（Common Gateway Interface），为HTTP服务器与其他机器上的程序服务通信交流的一种工具，CGI程序须运行在网络服务器上。

传统CGI接口方式的主要缺点是性能较差，因为每次HTTP服务器遇到动态程序时都需要重新启动解析器来执行解析，之后结果才会被返回给HTTP服务器。这在处理高并发访问时几乎是不可用的，因此就诞生了FastCGI。另外，传统的CGI接口方式安全性也很差，故而现在已经很少被使用了。

6.3.2 什么是FastCGI

FastCGI是一个可伸缩地、高速地在HTTP服务器和动态脚本语言间通信的接口（在Linux下，FastCGI接口即为socket，这个socket可以是文件socket，也可以是IP socket），主要优点是把动态语言和HTTP服务器分离开来。多数流行的HTTP服务器都支持FastCGI，包括Apache、Nginx和Lighttpd等。

同时，FastCGI也被许多脚本语言所支持，例如当前比较流行的脚本语言PHP。FastCGI接口采用的是C/S架构，它可以将HTTP服务器和脚本解析服务器分开，同时还能在脚本解析服务器上启动一个或多个脚本来解析守护进程。当HTTP服务器遇到动态程序时，可以将其直接交付给FastCGI进程来执行，然后将得到的结果返回给浏览器。这种方式可以让HTTP服务器专一地处理静态请求，或者将动态脚本服务器的结果返回给客户端，这在很大程度上提高了整个应用系统的性能。

FastCGI的重要特点如下：

·HTTP服务器和动态脚本语言间通信的接口或工具。

·可把动态语言解析和HTTP服务器分离开。

·Nginx、Apache、Lighttpd，以及多数动态语言都支持FastCGI。

·FastCGI接口方式采用C/S结构，分为客户端（HTTP服务器）和服务器端（动态语言解析服务器）。

·PHP动态语言服务器端可以启动多个FastCGI的守护进程（例如php-fpm（fcgi process mangement））。

·HTTP服务器通过（例如Nginx fastcgi_pass）FastCGI客户端和动态语言FastCGI服务器端通信（例如php-fpm）。

6.3.3 Nginx FastCGI的运行原理

Nginx不支持对外部动态程序的直接调用或者解析，所有的外部程序（包括PHP）必须通过FastCGI接口来调用。FastCGI接口在Linux下是socket，为了调用CGI程序，还需要一个FastCGI的wrapper（可以理解为用于启动另一个程序的程序），这个wrapper绑定在某个固定的socket上，如端口或文件socket。当Nginx将CGI请求发送给这个socket的时候，通过FastCGI接口，wrapper接收到请求，然后派生出一个新的线程，这个线程调用解释器或外部程序处理脚本来读取返回的数据；接着，wrapper再将返回的数据通过FastCGI接口，沿着固定的socket传递给Nginx；最后，Nginx将返回的数据发送给客户端，这就是Nginx+FastCGI的整个运作过程。详细的过程如图6-3所示。

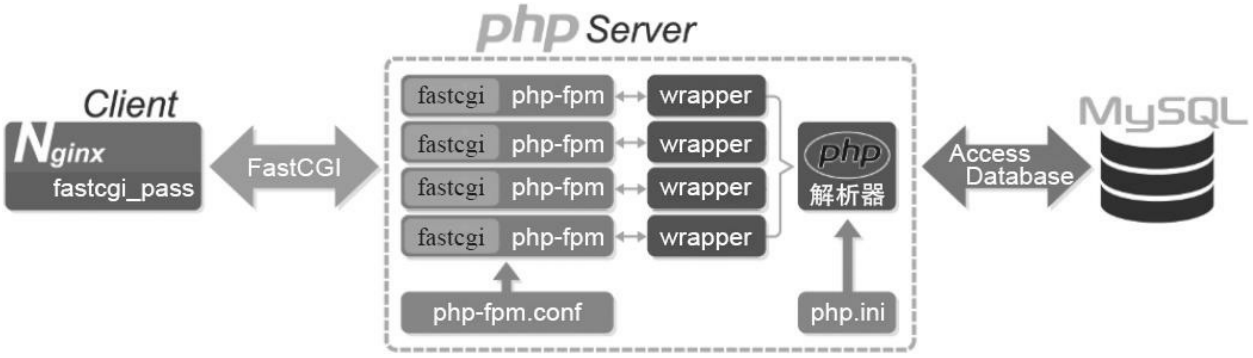


图6-3　Nginx结合PHP FastCGI运行的原理图

FastCGI的主要优点是把动态语言和HTTP服务器分离开来，使Nginx专门处理静态请求及向后转发的动态请求，而PHP/PHP-FPM服务

器则专门解析PHP动态请求。

6.4　LNMP之PHP（FastCGI方式）服务的安装准备

6.4.1　检查Nginx及MySQL的安装情况

1）检查确认Nginx及MySQL的安装路径，命令如下：

```
[root@www mysql]# ls -ld /application/mysql/
drwxr-xr-x 13 root root 4096 Apr 22 15:



32 /application/mysql/
[root@www mysql]# ls -ld /application/nginx/
drwxr-xr-x 11 root root 4096 Apr  6 15:



34 /application/nginx/
```

2）检查端口及启动情况，命令如下：

```
[root@www mysql]# netstat -lntup|grep -E "80|3306"
tcp     0      0 0.0.0.0:



3306        0.0.0.0:



*    LISTEN      9446/mysqld
tcp     0      0 0.0.0.0:



80          0.0.0.0:



*    LISTEN      3881/nginx
```

3）测试访问Nginx及MySQL是否OK，命令如下：

```
a.测试



nginx
[root@www /]# wget 127.0.0.1
...省略部分



...
Saving to:



 `index.html'
100%[==============================================>] 24   --.-K/s   in 0s
2012-04-07 22:



14:



19 (



1.17 MB/s)



 - `index.html' saved [24/24]
b.测试



mysql
[root@www mysql]# mysql -uroot -p
Enter password:



Welcome to the MySQL monitor.  Commands end with ;



 or \g.
Your MySQL connection id is 5
Server version:



 5.5.32 MySQL Community Server (
```

```
GPL)


...省略部分


...
Type 'help;


' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

如果访问结果和上述操作结果一致，就表明Nginx及MySQL的安装一切正常。

6.4.2　检查安装PHP所需的lib库

PHP程序在开发及运行时会调用一些诸如zlib、gd等函数库，因此需要确认lib库是否已经安装，执行命令如下：

```
rpm -qa zlib-devel libxml2-devel libjpeg-devel libjpeg-turbo-devel libiconv-devel
rpm -qa freetype-devel libpng-devel gd-devel libcurl-devel libxslt-devel
```

提示：libjpeg-turbo-devel是早期的libjpeg-devel的新名字，libcurl-devel是早期的curl-devel的新名字。

执行过程如下：

```
[root@www conf]# rpm -qa zlib-devel libxml2-devel libjpeg-devel libjpeg-turbo-devel
zlib-devel-1.2.3-29.el6.x86_64
[root@www conf]# rpm -qa freetype-devel libpng-devel gd-devel libcurl-devel libxslt·
```

提示：

·每个lib一般都会存在对应的以“*-devel*”命名的包，安装lib对应的-devel包后，对应的lib包就会自动安装好，例如安装gd-devel时就会安装gd。

·这些lib库不是必须安装的，但是目前的企业环境下一般都需要安装。否则，PHP程序运行时会出现问题，例如验证码无法显示等。

执行下面命令安装相关的lib软件包：

```
yum install zlib-devel libxml2-devel libjpeg-devel libjpeg-turbo-devel libiconv-deve
yum install freetype-devel libpng-devel gd-devel libcurl-devel libxslt-devel libxslt
```

需要说明的是，yum安装时采用libjpeg-devel这个名字也是可以的，但是安装完libjpeg-devel，查询包名就要用libjpeg-turbo-devel，同样，安装时，curl-devel对应查询包名为libcurl-devel。

安装后的结果如下：

```
[root@www conf]# rpm -qa zlib-devel libxml2-devel libjpeg-turbo-devel libiconv-devel
libxml2-devel-2.7.6-20.el6.x86_64
libjpeg-turbo-devel-1.2.1-3.el6_5.x86_64
zlib-devel-1.2.3-29.el6.x86_64提示：仅缺少



libiconv-devel包



[root@www tools]# rpm -qa freetype-devel libpng-devel gd-devel libcurl-devel libxslt
freetype-devel-2.3.11-15.el6_6.1.x86_64
gd-devel-2.0.35-11.el6.x86_64
libpng-devel-1.2.49-1.el6_2.x86_64
libcurl-devel-7.19.7-46.el6.x86_64
libxslt-devel-1.1.26-2.el6_3.1.x86_64
```

从以上结果看出，仅有libiconv-devel这个包没有安装，因为默认的yum源没有此包，后面会编译安装。

当然，也可以一个一个地yum安装或通过源文件手工编译安装，这是七八年前老男孩曾经用过的方法，后来发现太麻烦就想办法改成了现在的安装方式。毕竟效率第一嘛！

6.4.3 安装yum无法安装的libiconv库

由于这个安装非常简单，就不一一讲解了，按照下述命令复制执行即可。

```
mkdir -p /home/oldboy/tools
cd /home/oldboy/tools
wget http:


//ftp.gnu.org/pub/gnu/libiconv/libiconv-1.14.tar.gz
tar zxf libiconv-1.14.tar.gz
cd libiconv-1.14
./configure --prefix=/usr/local/libiconv
make
make install
cd ../
```


技巧：可以复制多行命令，一次输入执行。

libiconv相关信息地址为：http://www.gnu.org/software/libiconv/ （获得软件包的途径为打开g.cn，输入download libiconv）。老男孩曾经用过的上一个稳定版为libiconv-1.10.tar.gz。

6.4.4 安装libmcrypt库

这是一个使用动态加载的模块化的libmcrypt。libmcrypt对于在程序运行时添加/移除算法是有用的。limbcrypt-nm目前不再被官方支持，其软件地址为http://mcrypt.hellug.gr/lib/ ，编译PHP的过程中，libmcrypt库不是必须要安装的包。

很多网友采用的是很复杂的编译安装libmcrypt的方法，本文带着大家选择更简单的yum安装方法。

在CentOS默认的yum源里没有libmcrypt-devel，因此需要事先配置epel第三方yum源，具体命令如下：

```
wget -O /etc/yum.repos.d/epel.repo http:

//mirrors.aliyun.com/repo/epel-6.repo
```

下面就可以安装PHP相关包libmcrypt对应的包libmcrypt-devel了，命令如下：

```
yum -y install libmcrypt-devel
```

建议读者使用这个更简单的yum方法，抛弃网上流行的编译安装方法！

6.4.5　安装mhash加密扩展库

mhash是基于离散数学 原理不可逆向的PHP加密方式扩展库，其在默认情况下不会开启。mhash可以用于创建校验数值、消息摘要、消息认证码，以及无需原文的关键信息保存（如密码）等。它为PHP提供了多种散列算法，如MD5、SHA1、GOST等。可以通过MHASH_hashname（）查看其支持的算法有哪些。

不过，需要注意的是[1]：

·该扩展不能提供最新的散列算法。

·该扩展结果原则上运算不可逆。

同理，我们选择更简单的yum安装方法来安装mhash。安装PHP需要的相关包mhash的命令如下：

```
yum -y install mhash  #<==需提前安装

epel源，前文已经安装过

epel源不再赘述。
```

建议读者使用这个更简单的yum方法，抛弃网上流行的编译安装方法！

[1] 参考资料：http://baike.baidu.com/view/9549224.htm。

6.4.6　安装mcrypt加密扩展库

PHP程序员在编写代码程序时，除了要保证代码的高性能之外，还有一点是非常重要的，那就是程序的安全性保障。PHP除了自带的几种加密函数外，还有功能更全面的PHP加密扩展库mcrypt和mhash。

其中，mcrypt扩展库可以实现加密解密功能，就是既能将明文加密，也可以将密文还原。

可以说，mcrypt是PHP里面重要的加密支持扩展库，该库在默认情况下不开启。

mcrypt库支持20多种加密算法和8种加密模式，具体可以通过函数mcrypt_list_algorithms（）和mcrypt_list_modes（）来显示[1]。

以往很多网友及老男孩选择的是编译安装mcrypt的方法。这里选择更简单的yum安装方法安装mcrypt。安装PHP需要的相关包mcrypt的命令如下：

```
yum -y install mcrypt  #<==需提前安装

epel源，前文已经安装过

epel源不在累述。
```

建议读者使用这个更简单的yum安装方法，抛弃网上流行的编译安装方法！

[1] 参考：http://baike.baidu.com/view/9537042.htm。

6.5 开始安装PHP（FastCGI方式）服务

6.5.1 获取PHP软件包

我们可以使用wget方式下载PHP软件包，也可以下载到本地计算机，再上传到Linux中，这里以本地下载后上传到Linux系统为例进行讲解。

下面使用rz命令，通过php-5.3.27.tar.gz本地上传，当然也可以去网上下载，地址为：http://cn.php.net 。

```
[root@www tools]# cd /home/oldboy/tools
[root@www tools]# rz -y
```

Linux下的下载命令为：

```
wget http:


//cn.php.net/get/php-5.3.27.tar.gz/from/cn2.php.net/mirror
wget http:


//cn2.php.net/get/php-5.3.27.tar.gz/from/this/mirror
```

6.5.2 解压配置PHP

执行如下命令解压并配置PHP软件：

```
[root@www tools]# tar zxf php-5.3.27.tar.gz
[root@www tools]# cd php-5.3.27
[root@www php-5.3.27]# ./configure \
--prefix=/application/php5.3.27 \
--with-mysql=/application/mysql \
--with-iconv-dir=/usr/local/libiconv \
--with-freetype-dir \
--with-jpeg-dir \
--with-png-dir \
--with-zlib \
--with-libxml-dir=/usr \
--enable-xml \
--disable-rpath \
--enable-safe-mode \
--enable-bcmath \
--enable-shmop \
--enable-sysvsem \
--enable-inline-optimization \
--with-curl \
--with-curlwrappers \
--enable-mbregex \
--enable-fpm \
--enable-mbstring \
--with-mcrypt \
--with-gd \
--enable-gd-native-ttf \
--with-openssl \
--with-mhash \
--enable-pcntl \
--enable-sockets \
--with-xmlrpc \
--enable-zip \
--enable-soap \
--enable-short-tags \
--enable-zend-multibyte \
--enable-static \
--with-xsl \
--with-fpm-user=nginx \
--with-fpm-group=nginx \
--enable-ftp特别强调：上述每行结尾的换行符反斜线（


\）之后不能再有任何字符包括空格
```

执行上述命令后，最后的正确输出提示如图6-4所示。

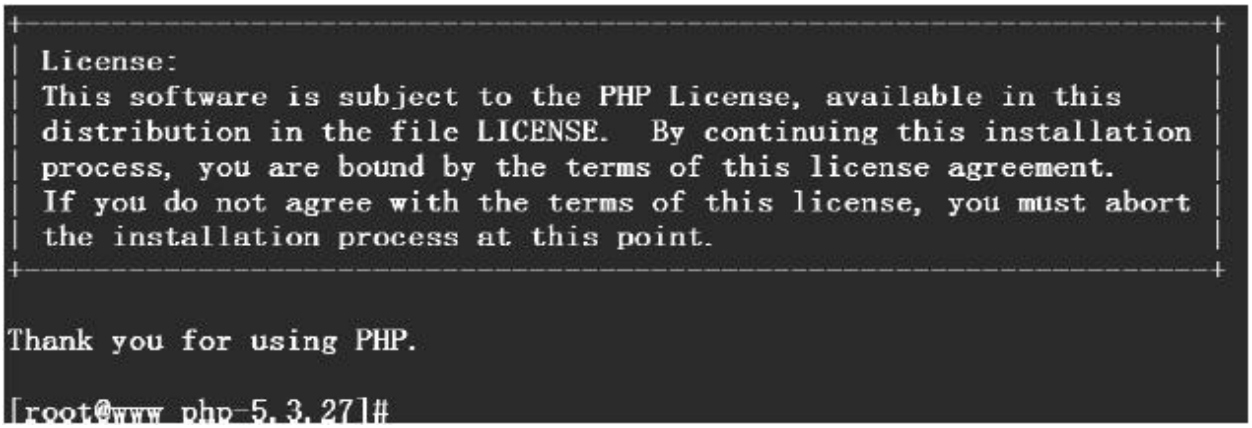


图6-4　编译PHP软件./configure，结束后正确输出提示

PHP FastCGI模式的设置说明：如果是PHP5.3及以上版本，所使用的编译参数为--enable-fpm，如果是PHP5.2版本，编译参数则为--enable-fastcgi--enable-fpm--enable-force-cgi。

对于上面的命令，部分参数说明如下。

·--prefix=/application/php5.3.27：表示指定PHP的安装路径为/application/php5.3.27。

·--with-mysql=/application/mysql：表示需要指定MySQL的安装路径，安装PHP需要的MySQL相关内容。当然，如果没有MySQL软件包，也可以不单独安装，这样的情况可使用--with-mysql=mysqlnd替代--with-mysql=/application/mysql，因为PHP软件里已经自带了连接MySQL的客户端工具。

·--with-fpm-user=nginx：nginx表示指定PHP-FPM进程管理的用户为Nginx，此处最好和Nginx服务用户统一。

·--with-fpm-group=nginx：表示指定PHP-FPM进程管理的组为Nginx，此处最好与Nginx服务用户组统一。

·--enable-fpm：表示激活PHP-FPM方式服务，即以FastCGIF方式运行PHP服务。

另外，针对此命令还有如下一些说明：

·可以执行./configure–help命令来详细查看以上各参数的用途。

·以上配置中的"\"反斜线表示换一行输入。

其他需要MySQL相关包场景的PHP对应编译参数如下：

```
--enable-mysqlnd \
--with-pdo-mysql=mysqlnd \
--with-mysqli=mysqlnd \
```

在PHP的安装过程中，经常会因为各种原因出现非常多的错误，下面针对configure过程的错误进行分析。

报错示例：configure：error：xslt-config not found.Please reinstall the libxslt>=1.1.0 distributio。

原因是前面要求安装的lib库没装上，解决办法如下：

```
[root@www php-5.3.27]# yum install libxslt-devel -y #<==前文已经提前安装了。
```

6.5.3 编译PHP

正确执行前文配置PHP软件的./configure系列命令后，就可以编译PHP软件了，具体操作过程如下：

```
[root@www php-5.3.27]# ln -s /application/mysql/lib/libmysqlclient.so.18  /usr/lib64
[root@www php-5.3.27]# touch ext/phar/phar.phar    #<==有网友用


mkdir是不对的。



[root@www php-5.3.27]# make
#make最后的正确提示



Build complete.
Don't forget to run 'make test'.
```

在编译PHP的过程中，同样可能出现非常多的错误，下面将针对make过程中的错误进行分析。

报错示例1：

```
Generating phar.php
/home/oldboy/tools/php-5.3.27/sapi/cli/php:


 error while loading shared libraries:


 libmysqlclient.so.18:
```

```
 cannot open shared object file:


 No such file or directory
make:


 *** [ext/phar/phar.php] Error 127
```

报错原因：这个原因是很常见的，即PHP找不到指定的库，一般是相应库的路径不对导致的。

解决办法如下：

```
[root@www php-5.3.27]# ln -s /application/mysql/lib/libmysqlclient.so.18  /usr/lib64
[root@www php-5.3.27]# make
Build complete.
Don't forget to run 'make test'.
[root@www php-5.3.27]#
```

报错示例2：

```
[root@www php-5.3.27]# make
Generating phar.phar
chmod:


 cannot access `ext/phar/phar.phar':


 No such file or directory
make:


 [ext/phar/phar.phar] Error 1  (


ignored)
```

```
Build complete.
Don't forget to run 'make test'.
```

报错原因：上述报错提示的是不能访问ext/phar/phar.phar这个文件或目录，其原因不太好确定，也许是PHP编译bug，但是可以使用如下方法解决这个问题。

解决办法如下：

```
[root@www php-5.3.27]# touch ext/phar/phar.phar
```

如果在PHP编译时使用--with-mysql=mysqlnd替代--with-mysql=/application/mysql，则没有上述示例1和示例2的错误发生。

如果实在无法继续安装，请按照本书介绍的步骤重新安装LNMP环境。

6.5.4 安装PHP生成文件到系统

披荆斩棘后，如果执行make命令不再出现错误提示，就可以使用下面的命令安装PHP了。

```
[root@www php-5.3.27]# make install
```

make install正确结束后的最后两行内容如下：

```
/home/oldboy/tools/php-5.3.27/build/shtool install -c ext/phar/phar.phar /applicatio
ln -s -f /application/php5.3.27/bin/phar.phar /application/php5.3.27/bin/phar
Installing PDO headers:


            /application/php5.3.27/include/php/ext/pdo/
```

6.5.5　配置PHP引擎配置文件php.ini

1）设置软链接以方便访问，命令如下：

```
[root@www php-5.3.27]# ln -s /application/php5.3.27 /application/php
[root@www php-5.3.27]# ls -l /application/php
lrwxrwxrwx 1 root root 22 Apr 22 20:


38 /application/php -> /application/php5.3.27
```

2）查看PHP配置默认模板文件，命令如下：

```
[root@www php-5.3.27]# ls php.ini*
php.ini-development  php.ini-production
```

请注意以上两文件的异同之处，可通过diff或vimdiff命令比较，或者使用其他文本比较工具。这里为便于讲解，使用Windows下的Beyond Compare工具给出区别（如图6-5所示）。

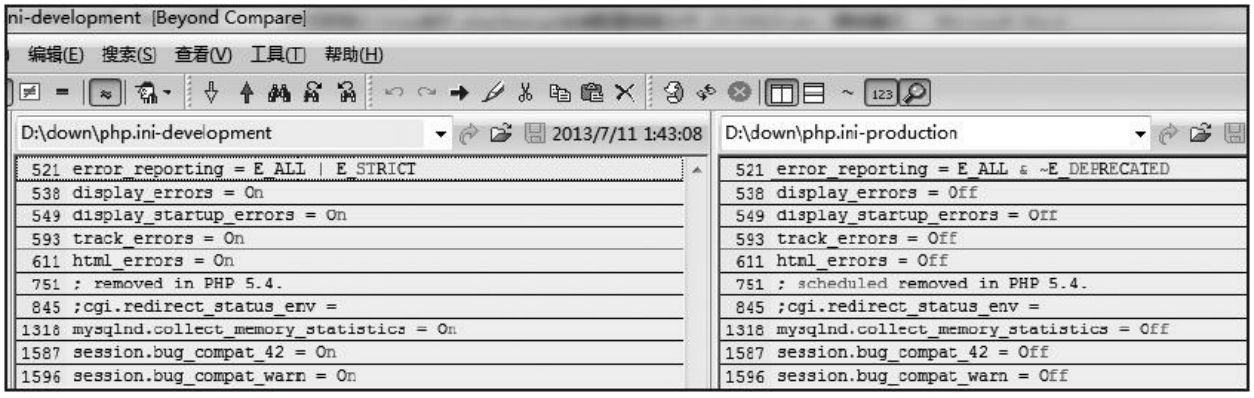


图6-5　生产场景和测试场景下php.ini的配置对比

从对比结果可以看出，开发环境更多的是开启日志、调试信息，而生产环境都是关闭状态。

3）拷贝PHP配置文件到PHP默认目录，并更改文件名称为php.ini，命令如下：

```
[root@www php-5.3.27]# cp php.ini-production /application/php/lib/php.ini
[root@www php-5.3.27]# ls -l /application/php/lib/php.ini
-rw-r--r-- 1 root root 69627 Apr 22 20:


42 /application/php/lib/php.ini
```

php.ini参数的说明见附录C。

6.5.6　配置PHP服务（FastCGI方式）的配置文件php-fpm.conf

配置命令如下：

```
[root@www php-5.3.27]# cd /application/php/etc/
[root@www etc]# ls
pear.conf  php-fpm.conf.default
[root@www etc]# cp php-fpm.conf.default php-fpm.conf
```

关于php-fpm.conf，暂时可用默认的配置，先把服务搭好，后文会对php-fpm.conf进行优化。

6.5.7　启动PHP服务（FastCGI方式）

1）启动PHP服务php-fpm，命令如下：

```
[root@www etc]# /application/php/sbin/php-fpm
```

2）检查PHP服务php-fpm的进程及启动端口的情况，命令如下：

```
[root@www etc]# ps -ef|grep php-fpm
root     26718     1  0 20:

45         00:

00:

00 php-fpm:

 master process (

/application/php5.3.27/etc/php-fpm.conf)

nginx    26719 26718  0 20:

45         00:

00:

00 php-fpm:
```

```
 pool www
nginx    26720 26718  0 20:


45         00:


00:


00 php-fpm:


 pool www
root     26722  7506  0 20:


45 pts/0    00:


00:


00 grep php-fpm
[root@www etc]# lsof -i :


9000 #<==默认


9000端口提供服务



COMMAND   PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
php-fpm 26718  root  7u  IPv4 187248  0t0  TCP localhost:


cslistener （


LISTEN）
```

```
php-fpm 26719 nginx  0u  IPv4 187248  0t0  TCP localhost:
cslistener (
LISTEN)

php-fpm 26720 nginx  0u  IPv4 187248  0t0  TCP localhost:
cslistener (
LISTEN)
```

6.6　配置Nginx支持PHP程序请求访问

6.6.1　修改Nginx配置文件

此处的环境为前文安装的Nginx环境，可按照如下步骤编辑Nginx的主配置文件nginx.conf。

1）查看Nginx当前的配置，命令如下：

```
[root@www etc]# cd /application/nginx/conf/
[root@www conf]# cp nginx.conf nginx.conf.02
[root@www conf]# cat nginx.conf
worker_processes  1;



error_log  logs/error.log;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
```

```
    sendfile        on;

    keepalive_timeout  65;

include extra/www.conf;

include extra/bbs.conf;

include extra/blog.conf;

include extra/status.conf;

}
```

2）PHP解析，这里以blog为例讲解，内容如下：

```
[root@www conf]# cat extra/blog.conf
    server {
        listen       80;

        server_name  blog.etiantian.org;

        location / {
            root   html/blog;

            index  index.html index.htm;
```

```
        }
        if  (


 $http_host ~* "^(


.*)


\.etiantian\.org$")


 {
           set $domain $1;


           rewrite ^(


.*)


 http:


//www.etiantian.org/$domain/oldboy.html break;


          }
    }
```

最终blog虚拟主机的完整配置如下：

```
[root@www conf]# cat extra/blog.conf
    server {
        listen       80;
```

```
        server_name  blog.etiantian.org;


        location / {
            root   html/blog;


            index  index.html index.htm;


        }
        location ~ .*\.（

php|php5）

$ {
            root html/blog;

     #<==这里配置如果不好，很容易出现

404错误，此处也

                                可以把两个

location里的

root html/blog;

                                合成一个，然后放到两个

location的外面
```

```
            fastcgi_pass  127.0.0.1:
9000;

            fastcgi_index index.php;

            include fastcgi.conf;

        }
    }
```

6.6.2 检查并启动Nginx

可通过如下命令检查Nginx配置文件的语法：

```
[root@www conf]# ../sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.2/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.2/conf/nginx.conf test is successful
[root@www conf]# ../sbin/nginx -s reload
```

此步在生产环境很关键，如不提前检查语法，重启后发现语法错误会导致Nginx无法提供服务，给用户访问体验带来不好的影响。

6.6.3　测试LNMP环境生效的情况

（1）测试PHP解析请求是否OK

1）进入指定的默认站点目录后，编辑index.php，添加如下内容：

```
[root@www conf]# cd ../html/blog/
[root@www blog]# echo "<php phpinfo () ;


 >" >test_info.php
[root@www blog]# cat test_info.php
<php phpinfo () ;


 >
```

以上代码为显示PHP配置信息的简单PHP文件代码。

注意：对于初学者来说，以上内容最好手工录入而不要拷贝，否则可能会导致意外结果。

2）调整Windows下的host解析（10.0.0.8为当前的机器IP），命令如下：

```
10.0.0.8    www.etiantian.org  bbs.etiantian.org blog.etiantian.org
```

3）打开浏览器，输入http://blog.etiantian.org/test_info.php ，即可打开如图6-6所示的界面。

图6-6　phpinfo的信息页面

出现上述界面，表示Nginx配合PHP解析已经正常。

（2）针对Nginx请求访问PHP，然后对PHP连接MySQL的情况进行测试

编辑test_mysql.php，加入如下内容：

```
<php
    //$link_id=mysql_connect(


'主机名


',
```

```
'用户
',
'密码
');

     $link_id=mysql_connect(
'localhost',
'root',
'oldboy123')
 or mysql_error();

    //$link_id=mysql_connect(
'localhost',
'test',
'');

     if(
$link_id)
```

```
{
        echo "mysql successful by oldboy!

";


    }else{
        echo mysql_error();


    }
//这是

php单行注释


/* 这是

php多行注释


 */
>
```

配置如下：

```
[root@www blog]# pwd
/application/nginx/html/blog
[root@www blog]# cat test_mysql.php
<php
        $link_id=mysql_connect(


'localhost',


'root',
```

```
'oldboy123')

 or mysql_error () ;

        if (

$link_id)

{
                echo "mysql successful by oldboy !

";

        }else{
                echo mysql_error () ;

        }
>
```

正常的结果如图6-7所示。

图6-7　PHP访问MySQL的正常信息显示

到此，LNMP的组合已基本搭建完毕。当然，我们还没有做相关的优化，此部分见后面的章节。

6.7　部署一个blog程序服务

6.7.1　开源博客程序WordPress介绍

WordPress是一套利用PHP语言和MySQL数据库开发的开源免费的blog（博客，网站）程序，用户可以在支持PHP环境和MySQL数据库的服务器上建立blog站点。它的功能非常强大，拥有众多插件，易于扩充功能。其安装和使用也都非常方便。目前WordPress已经成为搭建blog平台的主流，很多发布平台都是根据WordPress二次开发的，如果你也想像他们一样拥有自己的blog，可购买网上的域名及空间，然后搭建LNMP环境，部署WordPress程序后就可以轻松成就自己的梦想了。

 注意：WordPress是单用户个人博客，与blog.51cto.com的多用户博客是有区别的。

6.7.2　WordPress博客程序的搭建准备

1.MySQL数据库配置准备

1）登录MySQL数据库，操作如下：

```
[root@www ~]# mysql -uroot -p
Enter password:



Welcome to the MySQL monitor.  Commands end with ;



 or \g.
Your MySQL connection id is 8
Server version:



 5.5.32 MySQL Community Server  (



GPL)



Copyright  (



c)



 2000,



 2013,



 Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
```

```
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;

' or '\h' for help. Type '\c' to clear the current input statement.
```

2）创建一个专用的数据库WordPress，用于存放blog数据，操作如下：

```
mysql> create database wordpress;

     #<==创建一个数据库，名字为

wordpress
Query OK,

 1 row affected （

0.00 sec）

mysql> show databases like 'wordpress';

     #<==查看

+----------------------+
| Database （

wordpress）

 |
+----------------------+
| wordpress            |
```

```
+----------------------+
1 row in set （

0.02 sec）
```

3）创建一个专用的WordPress blog管理用户，命令如下：

```
mysql> grant all on wordpress.*  to wordpress@'localhost' identified by '123456';

Query OK,

 0 rows affected （

0.05 sec）
```

当数据库和PHP服务不在一台机器上，可以执行如下命令授权：

```
#grant all on wordpress.*  to wordpress@'10.0.0.%' identified by '123456';

  10.0.0.%为客户端地址段。

mysql> flush privileges;

                #<==刷新权限，使得创建用户生效

Query OK,
```

```
 0 rows affected （

0.03 sec）

mysql> show grants for wordpress@'localhost';

                                #<==查看用户对应的权限

+---------------------------------------------------------------------------+
| Grants for wordpress@localhost                                            |
+---------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'wordpress'@'localhost' IDENTIFIED BY PASSWORD '*6BB4837EB74
| GRANT ALL PRIVILEGES ON `wordpress`.* TO 'wordpress'@'localhost'          |
+---------------------------------------------------------------------------+
2 rows in set （

0.02 sec）

mysql> select user,

host from mysql.user;

     #<==查看数据库里创建的

wordpress用户

| user      | host      |
+-----------+-----------+
| root      | 127.0.0.1 |
| root      | localhost |
| wordpress | localhost |          #<==只允许本机通过
```

```
wordpress用户访问数据库



+-----------+-----------+
3 rows in set （


0.00 sec）



mysql> quit
Bye
```

2.Nginx及PHP环境配置准备

1）选择前文配置好的支持LNMP的blog域名对应的虚拟主机，命令如下：

```
[root@www ~]# cd /application/nginx/conf/extra/
[root@www extra]# cat blog.conf
    server {
        listen       80;



        server_name  blog.etiantian.org;



        location / {
            root   html/blog;



            index  index.php index.html index.htm;


                                    #<==补充一个首页文件
```

```
index.php
        }
        location ~ .*\.(

php|php5)

$ {
            root html/blog;

            fastcgi_pass  127.0.0.1:

9000;

            fastcgi_index index.php;

            include fastcgi.conf;

        }
    }
[root@www extra]# ../../sbin/nginx -s reload     #<==重新加载
```

2）获取WordPress博客程序，并放置到blog域名对应虚拟主机的站点目录下，即html/blog，操作命令如下：

```
[root@www extra]# cd ../../html/blog/
[root@www blog]# ls -sh wordpress-4.1-zh_CN.tar.gz
6.4M wordpress-4.1-zh_CN.tar.gz          #<==浏览

www.wordpress.org，下载博客程序
```

```
[root@www blog]# tar xf wordpress-4.1-zh_CN.tar.gz
                                #<==解压软件包



[root@www blog]# ls
index.html  test_mysql.php  wordpress  wordpress-4.1-zh_CN.tar.gz
[root@www blog]# rm -f index.html test_mysql.php
                                #<==删除无用文件



[root@www blog]# mv wordpress/* .     #<==把


wordpress目录的内容移动到


blog根目录



[root@www blog]# /bin/mv wordpress-4.1-zh_CN.tar.gz /home/oldboy/tools/
                                #<==移走源程序，备份起来



[root@www blog]# ls -l               #<==完整的


blog程序内容


总用量


 180
-rw-r--r--  1 nobody 65534   418 9月


  25 2013 index.php
-rw-r--r--  1 nobody 65534 19930 4月


  10 2014 license.txt
-rw-r--r--  1 nobody 65534  6575 12月
```

 23 09:

30 readme.html
drwxr-xr-x 2 nobody 65534 4096 4月

  23 12:

02 wordpress
-rw-r--r-- 1 nobody 65534 4951 8月

  21 2014 wp-activate.php
drwxr-xr-x 9 nobody 65534 4096 12月

 23 09:

30 wp-admin
-rw-r--r-- 1 nobody 65534   271 1月

   9 2012 wp-blog-header.php
-rw-r--r-- 1 nobody 65534 5008 11月

 27 04:

17 wp-comments-post.php
-rw-r--r-- 1 nobody 65534 2898 12月

 23 09:

30 wp-config-sample.php
drwxr-xr-x 5 nobody 65534 4096 12月

 23 09:

30 wp-content

```
-rw-r--r--  1 nobody 65534  2956 5月

  13 2014 wp-cron.php
drwxr-xr-x 12 nobody 65534  4096 12月

 23 09:

29 wp-includes
-rw-r--r--  1 nobody 65534  2380 10月

 25 2013 wp-links-opml.php
-rw-r--r--  1 nobody 65534  2714 7月

   8 2014 wp-load.php
-rw-r--r--  1 nobody 65534 33435 12月

 17 06:

19 wp-login.php
-rw-r--r--  1 nobody 65534  8252 7月

  17 2014 wp-mail.php
-rw-r--r--  1 nobody 65534 11115 7月

  18 2014 wp-settings.php
-rw-r--r--  1 nobody 65534 25152 12月

  1 05:

23 wp-signup.php
-rw-r--r--  1 nobody 65534  4035 12月

  1 05:

23 wp-trackback.php
```

```
-rw-r--r--  1 nobody 65534  3032 2月


  10 2014 xmlrpc.php
[root@www blog]# chown -R nginx.nginx ../blog/
#<==授权


nginx及


php服务访问


blog站点目录，此授权不是非常安全，是临时的办法，更规范的做法见后文


LNMP优化部分。



[root@www blog]# ls -l                #<==最终博客程序目录及权限设置


总用量


 180
-rw-r--r--  1 nginx nginx   418 9月


  25 2013 index.php
-rw-r--r--  1 nginx nginx 19930 4月


  10 2014 license.txt
-rw-r--r--  1 nginx nginx  6575 12月


 23 09:


30 readme.html
drwxr-xr-x  2 nginx nginx  4096 4月


  23 12:
```

```
02 wordpress
-rw-r--r--  1 nginx nginx  4951 8月


  21 2014 wp-activate.php
drwxr-xr-x  9 nginx nginx  4096 12月


 23 09:


30 wp-admin
-rw-r--r--  1 nginx nginx   271 1月


   9 2012 wp-blog-header.php
-rw-r--r--  1 nginx nginx  5008 11月


 27 04:


17 wp-comments-post.php
-rw-r--r--  1 nginx nginx  2898 12月


 23 09:


30 wp-config-sample.php
drwxr-xr-x  5 nginx nginx  4096 12月


 23 09:


30 wp-content
-rw-r--r--  1 nginx nginx  2956 5月


  13 2014 wp-cron.php
drwxr-xr-x 12 nginx nginx  4096 12月


 23 09:
```

```
29 wp-includes
-rw-r--r--  1 nginx nginx  2380 10月


 25 2013 wp-links-opml.php
-rw-r--r--  1 nginx nginx  2714 7月


   8 2014 wp-load.php
-rw-r--r--  1 nginx nginx 33435 12月


 17 06:


19 wp-login.php
-rw-r--r--  1 nginx nginx  8252 7月


  17 2014 wp-mail.php
-rw-r--r--  1 nginx nginx 11115 7月


  18 2014 wp-settings.php
-rw-r--r--  1 nginx nginx 25152 12月


  1 05:


23 wp-signup.php
-rw-r--r--  1 nginx nginx  4035 12月


  1 05:


23 wp-trackback.php
-rw-r--r--  1 nginx nginx  3032 2月


  10 2014 xmlrpc.php
```

这样，准备工作就做好了。

6.7.3　开始安装blog博客程序

很多开源程序都支持浏览器傻瓜式的界面安装，为了让初学者易懂，此处也采用界面安装的方式讲解。

1）打开浏览器输入blog.etiantian.org（提前做好host或DNS解析），回车后，出现如图6-8所示的内容。

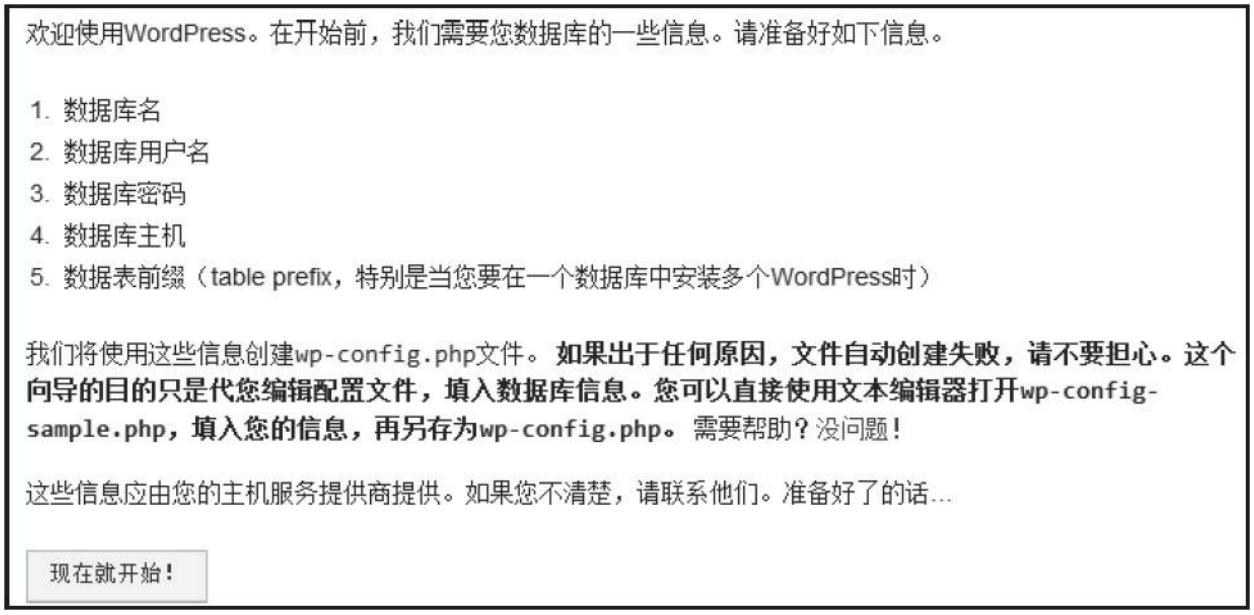


图6-8　博客程序安装开始

2）仔细阅读页面的文字信息后，单击“现在就开始”按钮继续，然后在出现的页面表单上填写相应的内容，如图6-9所示。

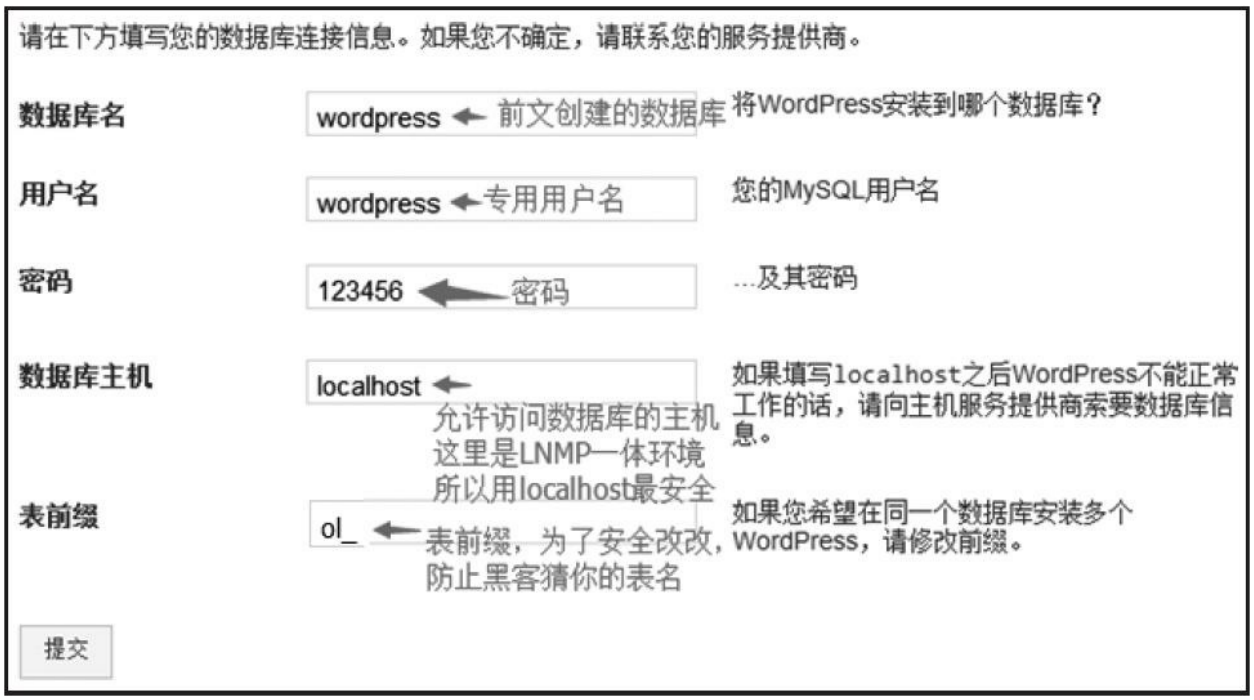


图6-9　连接数据库信息表单

3）在页面表单里填好内容后，单击结尾的“提交”按钮继续，得到如图6-10所示界面。

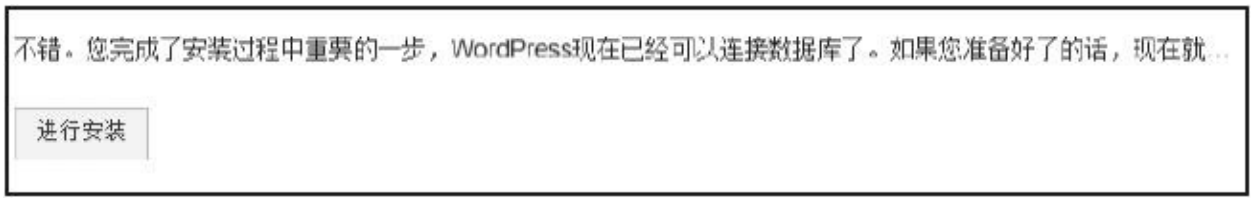


图6-10　开始安装博客

4）出现如图6-10所示的界面，表示可以安装WordPress了，单击“进行安装”按钮继续，进入如图6-11所示的界面。

图6-11　博客管理平台的信息

5）根据界面提示设置blog站点的信息后，单击“安装WordPress”按钮继续。

出现如图6-12所示的信息提示，表示已经成功安装了WordPress博

客。

下面是blog博客的简单使用方法。

1）在图6-12所示的界面中，输入账号和密码，然后单击“登录”按钮即可登录blog网站的管理后台（如图6-13所示）。

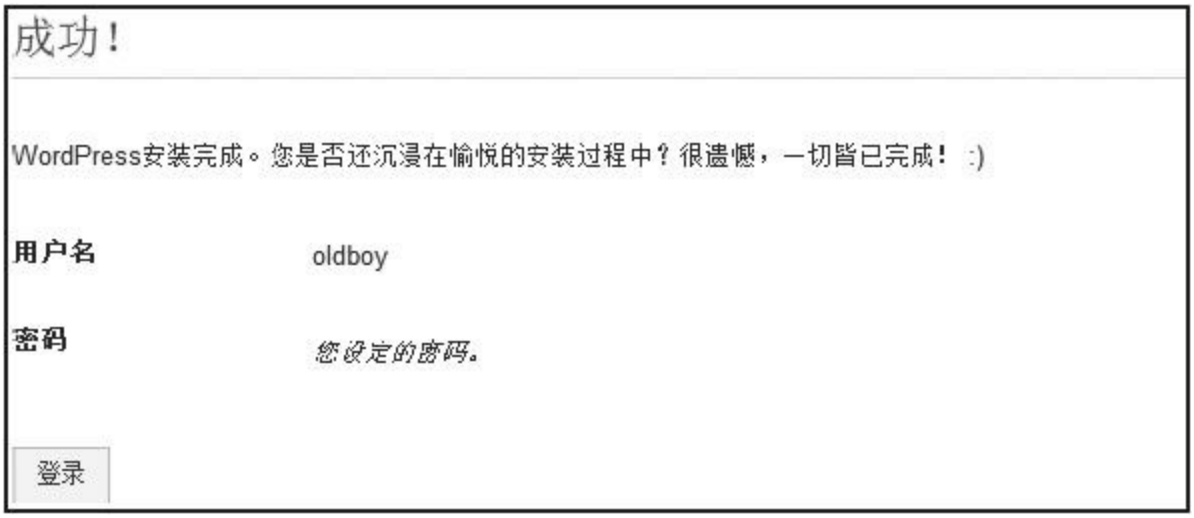


图6-12　博客程序成功安装

图6-13　登录后台

2）图6-14为登录后台后的界面，左边是管理菜单。

图6-14　登录后的WordPress后台

3）单击“文章”→“写文章”菜单发布博文，如图6-15和图6-16所示。

图6-15　发文章导航

4）单击右上角的“发布”按钮发布博文，发布成功后，如图6-17所示。

图6-16　发布博客文章的界面

图6-17　博客文章发布成功

5）单击图6-17中的“查看文章”或直接在浏览器输入blog.etiantian.org，查看发布后的文章内容（如图6-18所示）。

这样就看到发布的文章了，更多有关后台管理的内容，留给大家发掘吧！至此，LNMP环境搭建及博客程序部署全部完成。

6.7.4　实现WordPress博客程序URL静态化

实现此功能时，首先要在WordPress后台依次单击设置→固定链接→自定义结构，然后输入下面的代码，并保存更改（如图6-19所示）。

```
/archives/%post_id%.html说明：

%post_id% 是数据库对应博文内容的唯一

ID，例如

423
```

图6-18　查看发布的博客文章

修改过程如图6-19所示。

图6-19　博客伪静态配置图

接着，在Nginx配置文件的server容器中添加下面的代码：

```
        location / {
        if  (

-f $request_filename/index.html)

{
            rewrite  (

.*)

 $1/index.html break;
```

```
        }
        if  (

-f $request_filename/index.php)

{
            rewrite  (

.*)

 $1/index.php;

        }
        if  (!

-f $request_filename)

{
            rewrite  (

.*)

 /index.php;

         }
        }
```

实际案例如下：

```
[root@www extra]# cat blog.conf
```

```
    server {
        listen       80;

        server_name  blog.etiantian.org;

        root   html/blog;

        index  index.php index.html index.htm;

        location / {
        if (

-f $request_filename/index.html)

{
            rewrite (

.*)

 $1/index.html break;

        }
        if (

-f $request_filename/index.php)

{
            rewrite (

.*)
```

```
 $1/index.php;

        }
        if  (!

-f $request_filename)

{
            rewrite  (

.*)

 /index.php;

         }
        }
        location ~ .*\. (

php|php5)

$ {
            fastcgi_pass  127.0.0.1:

9000;

            fastcgi_index index.php;

            include fastcgi.conf;

        }
    }
```

最后，检查语法并重新加载Nginx服务，操作如下：

```
[root@www extra]# ../../sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.2/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.2/conf/nginx.conf test is successful
[root@www extra]# ../../sbin/nginx -s reload
```

现在可通过浏览器访问了，如图6-20所示。

图6-20　博客伪静态页面图

6.8 有关使用高版本PHP 5.5的说明

考虑到本书的部分读者可能希望使用更高版本的PHP软件，本节给出了PHP 5.5.26的config配置编译参数，在PHP 5.3.27和PHP 5.5.26中，除了编译参数略有区别外，其他运维方面的使用几乎是一样的。下面是PHP 5.5.26的config配置编译参数：

```
./configure \
--prefix=/application/php5.5.26 \
--with-mysql=/application/mysql/ \
--with-pdo-mysql=mysqlnd \
--with-iconv-dir=/usr/local/libiconv \
--with-freetype-dir \
--with-jpeg-dir \
--with-png-dir \
--with-zlib \
--with-libxml-dir=/usr \
--enable-xml \
--disable-rpath \
--enable-bcmath \
--enable-shmop \
--enable-sysvsem \
--enable-inline-optimization \
--with-curl \
--enable-mbregex \
--enable-fpm \
--enable-mbstring \
--with-mcrypt \
--with-gd \
--enable-gd-native-ttf \
--with-openssl \
--with-mhash \
--enable-pcntl \
--enable-sockets \
--with-xmlrpc \
--enable-soap \
--enable-short-tags \
--enable-static \
--with-xsl \
--with-fpm-user=nginx \
--with-fpm-group=nginx \
--enable-ftp \
--enable-opcache=no
```

6.9 本章重点回顾

1）LNMP的组合中各组件工作调度逻辑关系。

2）Nginx与PHP通过FastCGI模式通信的原理。

3）LNMP环境的企业级搭建。

4）WordPress博客程序的安装搭建。

6.10 本章参考资料

·Nginx官方网站：http://nginx.org

·PHP官方网站：http://php.net

·MySQL官方网站：http://mysql.com

第7章　PHP服务缓存加速优化实战

7.1　PHP缓存加速器介绍与环境准备

7.1.1　PHP缓存加速器介绍

1.操作码介绍及缓存原理

当客户端请求一个PHP程序时，服务器的PHP引擎会解析该PHP程序，并将其编译为特定的操作码（Operate Code，简称opcode）文件，该文件是执行PHP代码后的一种二进制表示形式。默认情况下，这个编译好的操作码文件由PHP引擎执行后丢弃。而操作码缓存（Opcode Cache）的原理就是将编译后的操作码保存下来，并放到共享内存里，以便在下一次调用该PHP页面时重用它，避免了相同代码的重复编译，节省了PHP引擎重复编译的时间，降低了服务器负载，同时减少了CPU和内存开销。

2.PHP缓存加速软件介绍

为了提高PHP引擎的高并发访问及执行速度，产生了一系列PHP缓存加速软件。这些软件设计的目的就是缓存前文提到的PHP引擎解析过的操作码文件，以便在指定时间内有相同的PHP程序请求访问时，不再

需要重复解析编译，而是直接调用缓存中的PHP操作码文件，这样就提高了动态Web服务的处理速度，从而提升了用户访问企业网站的整体体验。

7.1.2 LAMP环境PHP缓存加速器的原理

下面简单介绍Apache环境的PHP缓存加速器原理。

在LAMP环境中，Apache服务是使用libphp5.so响应处理PHP程序请求的，整个流程大概如下：

1）Apache接收客户的PHP程序请求，并根据规则过滤之。

2）Apache将PHP程序请求传递给PHP处理模块libphp5.so。

3）PHP引擎定位磁盘上的PHP文件，并将其加载到内存中解析。

4）PHP处理模块libphp5.so将PHP源代码编译成为opcode。

5）PHP处理模块libphp5.so执行opcode，然后把opcode缓存起来。

6）Apache接收客户端新的PHP程序请求，PHP引擎直接读取缓存执行opcode文件，并将结果返回。在这一次任务中，就无第4步的编译解析了，从而提升了PHP编译解析效率。

PHP缓存加速器解决的是上述第5步的问题，默认情况下PHP会将opcode内容执行后丢弃，这里却通过PHP缓存加速软件，将opcode内容缓存了下来，目的是当有重复请求时，不需要再重复编译解析PHP程序

代码，因为在高并发高访问量的网站上，大量的重复编译会消耗很多的系统资源和时间，而这也就会成为瓶颈，既影响了处理速度，又加重了服务器的负载，为了解决此问题，PHP缓存加速器就这样诞生了。

图7-1是LAMP环境下PHP请求及操作码缓存过程的原理示意图。

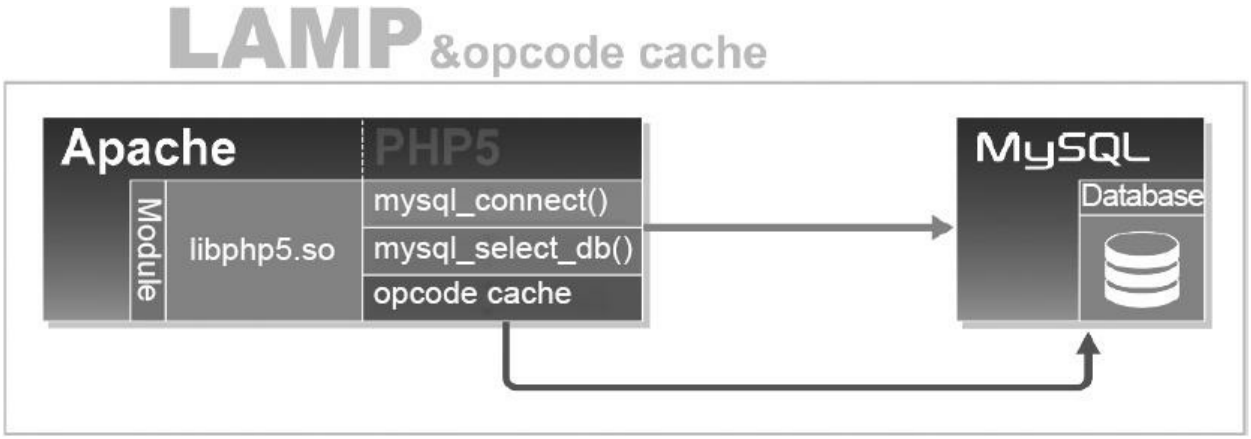


图7-1　LAMP经典网站架构PHP操作码缓存原理示意图

7.1.3 LNMP环境PHP缓存加速器的原理详解

在LNMP环境中，PHP引擎不再使用libphp5.so模块了，而是启动了独立的FCGI即php-fpm进程，由它监听来自Nginx的PHP程序请求，并交给PHP引擎解析处理，整个执行流程大概如下：

1）Nginx接收客户端的PHP程序访问请求。

2）Nginx根据扩展名等过滤规则将PHP程序请求传递给解析PHP的FCGI（php-fpm）进程。

3）PHP FPM进程调用PHP解析器读取站点磁盘上的PHP文件，并加载到内存中。

4）PHP解析器将PHP程序编译成为opcode文件，然后把opcode缓存起来。

5）PHP FPM引擎执行opcode树后，返回数据给Nginx，进而返回客户端。

6）Nginx接收客户新的PHP程序请求，PHP FPM引擎就会直接读取缓存中的opcode并执行，将结果返回。该过程中无需第4步操作，从而提升了PHP编译解析效率。

图7-2为LNMP环境下PHP请求及操作码缓存过程的原理示意图。

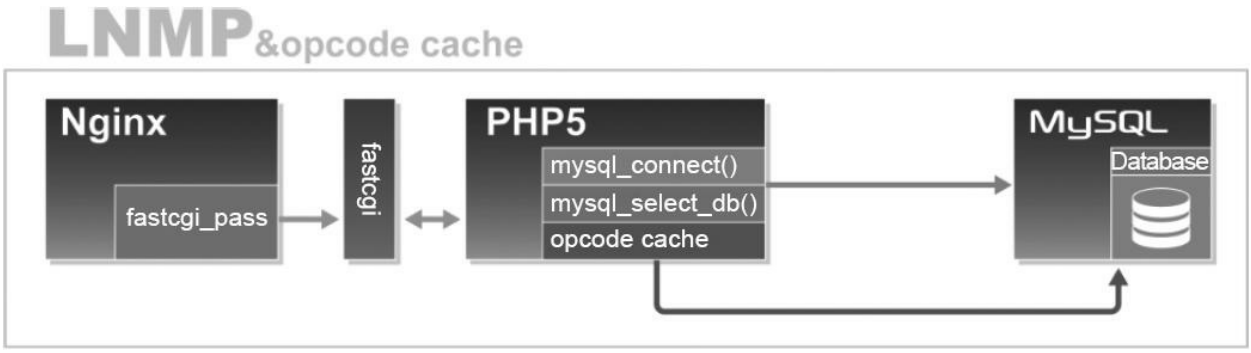


图7-2　LNMP经典网站架构PHP操作码缓存原理示意图

7.1.4 PHP缓存加速器软件种类及选择建议

PHP缓存加速器软件常见的种类有XCache、eAccelerator、APC（Alternative PHP Cache），ZendOpcache等，那么，在企业环境我们要如何选择PHP缓存加速器软件呢？

事实上，任选其一即可，没必要都安装上，都安装也可能会发生冲突。总的建议就是根据企业的业务需求及选择前的压力测试结果，或者根据个人的经验偏好选择。不过，老男孩建议首选XCache，其次是eAccelerator，如果想尝新，可以选择ZendOpcache，限于篇幅下文未对APC加速软件[1]做深入讲解。

首选XCache的原因如下：

·经过测试，XCache效率更高、速度更快。

·XCache软件开发社区更活跃，最新版2014年底发布。

·支持更高版本的PHP，例如PHP 5.5、PHP 5.6。

次选eAccelerator的原因如下：

·安装及配置参数更简单，加速效果也不错。

·文档资料较多，但官方对软件的更新很慢，社区不活跃。

·仅适合PHP版本5.4以下的程序。

选择ZendOpcache的原因如下：

·是PHP官方研发的新一代缓存加速软件，以后的发展潜力可能会很好，PHP 5.5以前的版本可以通过ZendOpcache软件以插件扩展的方式安装，从PHP 5.5版本开始已经整合到PHP软件里了，编译时只需指定一个参数即可，例如：--enable-opcache。

·ZendOpcache可能是未来的缓存加速首选，现在的稳定性还有待检验，小规模环境下PHP 5以前的版本可以通过插件式安装使用，PHP 5以上的版本可以直接指定参数编译使用。若可以忍受ZendOpcache的各种未知问题的话，也可以尝试使用。

[1] APC全称为Alternative PHP Cache，也是一款不错的PHP缓存加速软件，本章未对APC加速软件做深入讲解，仅仅在此处提及，读者如果对APC很感兴趣，可以参阅相关资料文档：http://pecl.php.net/packages.php和http://pecl.php.net/package/APC。

7.1.5　PHP缓存加速器安装环境准备

1.LNMP基础Web环境准备

在安装PHP的扩展及缓存加速软件之前，需要先安装好LNMP的完整环境，例如：能配置出现phpinfo信息的界面，表示PHP服务正常安装，同时最好可以编写一个调用数据库的简单PHP程序，例如test_mysql.php（见前文），进而确认MySQL数据库是否正常。在前面的章节里，已经详细讲解了LNMP环境的安装、配置及部署方法，此处不再多提。

当前LNMP环境软件的各个版本信息如表7-1所示。

表7-1　LNMP环境软件及版本说明

软件	版　本
Linux	CentOS 6.6 64bit
Nginx	1.6.3
MySQL	5.5.32 64bit
PHP	5.3.27（PHP 5.5 版本见本章结尾）

如果上述软件的版本对不上，在安装PHP的扩展软件时可能会遇到一些小问题。因此，建议在学习中使用的版本尽量和本书给出的版本一致，操作也要保持一致，否则可能会出现额外的问题，影响学习进度，等按照书上的操作完成了部署后，再去变换版本操作。这样的学习方法

是最好的。

2.检查LNMP的软件版本

1）查看Linux内核及版本相关信息，命令如下：

```
[root@www ~]# cat /etc/redhat-release
CentOS release 6.6 (


Final)



[root@www ~]# uname -r
2.6.32-504.el6.x86_64
[root@www ~]# uname -m
x86_64
```

2）查看Nginx Web版本相关信息，命令如下：

```
[root@www ~]# /application/nginx/sbin/nginx -v
nginx version:


 nginx/1.6.3
```

3）查看PHP服务版本相关信息，命令如下：

```
[root@www ~]# /application/php/bin/php -v
PHP 5.3.27 (


cli)


  (
```

```
built:


 Apr 22 2015 20:


23:


57)


Copyright  (


c)


 1997-2013 The PHP Group
Zend Engine v2.3.0,


 Copyright  (


c)


 1998-2013 Zend Technologies
```

4）查看MySQL服务版本相关信息，命令如下：

```
[root@www ~]# mysqladmin -uroot -poldboy123 version
mysqladmin  Ver 8.42 Distrib 5.5.32,


 for linux2.6 on x86_64
Copyright  (


c)
```

2000,

2013,

Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Server version   5.5.32
Protocol version   10
Connection   Localhost via UNIX socket
UNIX socket   /tmp/mysql.sock
Uptime:

8 days 18 hours 52 min 32 sec
Threads:

1 Questions:

2263 Slow queries:

0 Opens:

204 Flush tables:

1 Open tables:

4 Queries per second avg:

0.002

7.1.6　有关LNMP环境扩展软件的部署说明

1.LNMP缓存加速特别提示

不管是Apache还是Nginx，最后都是通过PHP提供动态程序解析的，因此，不管是Apache的libphp5.so模块方式，还是Nginx的FCGI的PHP服务方式，最终在PHP引擎上的优化是一致的，即都是基于PHP服务（php.ini）的，因此，如无特殊说明，本节以后的环境安装和优化均适用于LNMP和LAMP，有需要使用Apache服务的读者学习本章也能得到参考。

2.解决部分加速软件的Perl编译问题

在下面各类软件的安装编译过程中，如果不解决Perl的一些环境问题可能会带来意想不到的安装错误或警告。为了避免出现这些问题导致前功尽弃，下面把一些拦路虎提前告诉大家，并解决之。

（1）配置环境变量LC_ALL

配置环境变量LC_ALL的过程如下：

```
[root@www ~]# echo 'export LC_ALL=C'>> /etc/profile
#←设置环境变量，解决后面

Perl程序插件的编译问题。符号
```

```
">>"表示向文件追加内容



[root@www ~]# tail -1 /etc/profile
#←查看是否正确追加了



export LC_ALL=C环境配置



export LC_ALL=C
[root@www ~]# source /etc/profile
#←使增加的环境变量配置生效



[root@www ~]# echo $LC_ALL
C #←查看生效结果
```

如果不设置该变量，在安装某些加速软件时，可能会遇到如下警告（安装eAccelerator时遇到警告）：

```
[root@www eaccelerator-0.9.6.1]# /application/php/bin/phpize
Configuring for:



PHP Api Version:



         20090626
Zend Module Api No:



      20090626
Zend Extension Api No:
```

```
   220090626
perl:


 warning:


 Setting locale failed.
perl:


 warning:


 Please check that your locale settings:


        LANGUAGE =  (


unset) ,


        LC_ALL =  (


unset) ,


        LANG = "zh_cn.gb18030"
    are supported and installed on your system.
perl:


 warning:


 Falling back to the standard locale  (


"C")


.
perl:
```

warning:

Setting locale failed.
perl:

warning:

Please check that your locale settings:

LANGUAGE = (

unset),

LC_ALL = (

unset),

LANG = "zh_cn.gb18030"
are supported and installed on your system.
perl:

warning:

Falling back to the standard locale (

"C")

.解决方法：见前文

（2）安装Perl相关软件依赖

需要提前安装Perl相关软件依赖软件包，执行yum-y install perl-CPAN或yum-y install perl-devel，任意一个即可，大约依赖17个包，提前解决后面安装软件时可能遇到的报错问题。

如果不安装上述软件包，在后面安装ImageMagick时可能会报错。后文安装ImageMagick时有相应的报错说明。

7.2 安装PHP缓存加速器扩展

7.2.1 安装PHP eAccelerator缓存加速模块

1.eAccelerator缓存加速插件说明

eAccelerator是一个免费的、开放源代码的PHP加速、优化及缓存的扩展插件软件，它可以缓存PHP程序编译后的中间代码文件（opcode）、session数据等，降低PHP程序在编译解析时对服务器的性能开销。eAccelerator还可以加快PHP程序的执行速度，降低服务器负载压力，使PHP程序代码执行效率提高1~10倍。

eAccelerator会把编译好的PHP程序存放在共享内存里，然后每次从内存里调用执行，可以设定把一些不适合放在内存里缓存的编译结果存储到磁盘上，默认情况下，磁盘和内存缓存都会被eAccelerator使用。

eAccelerator诞生于2004年，前身是Turck MMCache，因为开发者进入了Zend公司工作，所以开发eAccelerator的人继承了Turck MMCache的一些特性，从而设计出了eAccelerator加速器。

eAccelerator算是一个“老牌”的缓存加速软件，曾经在结合PHP引擎解析时被广泛使用，成熟稳定，目前代码更新不活跃，因此，使用的企

业逐渐减少，但eAccelerator仍是一款值得信赖的缓存加速软件。XCache的官方也称赞eAccelerator是不错的opcode缓存器。

eAccelerator的最新版为0.9.6.1，支持的PHP最新版本为PHP 5.3及以前5系列的版本。

早期的0.9.5版本支持PHP 4和PHP 5.2以前的版本。

eAccelerator最新版0.9.6.1版的下载地址为：

https://github.com/eaccelerator/eaccelerator/downloads 。

2.eAccelerator插件安装过程

具体的安装命令集如下：

```
cd /home/oldboy/tools/
wget https:



//github.com/downloads/eaccelerator/eaccelerator/eaccelerator-0.9.6.1.tar.bz2
tar xf eaccelerator-0.9.6.1.tar.bz2
cd eaccelerator-0.9.6.1
/application/php/bin/phpize
./configure --enable-eaccelerator=shared --with-php-config=/application/php/bin/php
#<==configure的参数路径要正确配置，特别是后面的



--with-php-config参数对应的



PHP安装目录地址，几乎所有的



PHP扩展都要指定这个参数，请读者注意，后面不再提及
```

```
make
make install
cd ../
ls /application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
```

安装操作详细过程如下：

```
[root@www ~]# cd /home/oldboy/tools/
[root@www tools]# wget https:



//github.com/downloads/eaccelerator/eaccelerator/eaccelerator-0.9.6.1.tar.bz2
[root@www tools]# ls eaccelerator-0.9.6.1.tar.bz2
eaccelerator-0.9.6.1.tar.bz2
[root@www tools]# tar xf eaccelerator-0.9.6.1.tar.bz2
[root@www tools]# cd eaccelerator-0.9.6.1
[root@www eaccelerator-0.9.6.1]# /application/php/bin/phpize
Configuring for:



PHP Api Version:



         20090626
Zend Module Api No:



      20090626
Zend Extension Api No:



   220090626
[root@www eaccelerator-0.9.6.1]# ./configure --enable-eaccelerator=shared --with-php
[root@www eaccelerator-0.9.6.1]# make
...skip...
Build complete.
Don't forget to run 'make test'.
[root@www eaccelerator-0.9.6.1]# make install
Installing shared extensions:



/application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
[root@www eaccelerator-0.9.6.1]# ls /application/php5.3.27/lib/php/extensions/no-deb
eaccelerator.so  #<==最后生成了
```

```
eaccelerator.so模块就表示

eaccelerator成功安装

[root@www eaccelerator-0.9.6.1]# cd ..
```

下面是实践得出的注意事项及说明：

·PHP 5.3.XX可用eaccelerator-0.9.6版本，如果使用0.9.5.2版本在make阶段会报错：make：***[optimize.lo]Error 1。

·PHP 5.2.XX可用eaccelerator-0.9.5.2版本。

 提示：上述测试验证了eAccelerator的官方说明，请读者选择时注意。

下面是学生遇到的问题故障。

问题：出现Cannot find config.m4.错误。

```
[root@www tools]# /application/php/bin/phpize
Cannot find config.m4.
Make sure that you run '/application/php/bin/phpize' in the top level source directo
```

解答：需要切换到eaccelerator路径下执行/application/php/bin/phpize。

7.2.2　安装PHP XCache缓存加速模块

1.XCache缓存加速插件说明

XCache是一个开源的、又快又稳定的PHP opcode缓存器/优化器，其项目leader曾经是Lighttpd（和Nginx类似的高速Web服务软件）的开发成员之一。XCache把PHP程序编译后的数据（opcode）缓存到共享内存里，避免相同的程序重复编译。用户请求相同的PHP程序时，可以直接使用缓存中已编译好的数据，从而提高PHP的访问速度，通常可以提升2~5倍，并大幅降低服务器负载开销。

很多公司使用XCache，它已经能在大流量/高负载的生产环境中稳定运行，与同类型的opcode缓存器相比在各个方面都更胜一筹，例如：社区活跃、快速开发、能够快速跟进PHP的版本更新等。

当前稳定版本为3.1.x（全面支持PHP 5.1~5.5）和3.2.x（2014年底发布，全面支持PHP 5.1~5.6）。

XCache软件详情请参考：

http://xcache.lighttpd.net 或http://xcache.lighttpd.net/wiki/Introduction

。

2.XCache插件的安装过程

本章以XCache最新稳定版本3.2.x（全面支持PHP 5.1~5.6）为例进行讲解，其安装步骤及过程如下。

具体的安装命令集如下：

```
cd /home/oldboy/tools/
wget http:



//xcache.lighttpd.net/pub/Releases/3.2.0/xcache-3.2.0.tar.bz2
tar xf xcache-3.2.0.tar.bz2
cd xcache-3.2.0
/application/php/bin/phpize
./configure --enable-xcache   --with-php-config=/application/php/bin/php-config
make
make install
ls -l /application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
cd ..
```

详细的安装过程如下：

```
[root@www tools]# cd /home/oldboy/tools/
[root@www tools]# wget http:



//xcache.lighttpd.net/pub/Releases/3.2.0/xcache-3.2.0.tar.bz2
[root@www tools]# ls xcache-3.2.0.tar.bz2
xcache-3.2.0.tar.bz2
[root@www tools]# tar xf xcache-3.2.0.tar.bz2
[root@www tools]# cd xcache-3.2.0
[root@www xcache-3.2.0]# /application/php/bin/phpize
Configuring for:



PHP Api Version:



         20090626
Zend Module Api No:
```

```
      20090626
Zend Extension Api No:



   220090626
[root@www xcache-3.2.0]# ./configure --enable-xcache   --with-php-config=/applicatio
[root@www xcache-3.2.0]# make
...skip...
Build complete.
Don't forget to run 'make test'.
[root@www xcache-3.2.0]# make install
Installing shared extensions:



     /application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
[root@www xcache-3.2.0]# ls -l /application/php5.3.27/lib/php/extensions/no-debug-no
total 1052
-rwxr-xr-x 1 root root 417069 May  1 12:



48 eaccelerator.so
-rwxr-xr-x 1 root root 658468 May  1 14:



33 xcache.so
                    #<==最后生成了



xcache.so模块，表示



xcache安装成功



[root@www xcache-3.2.0]# cd ../
```

7.2.3 PHP官方加速插件ZendOpcache

1.ZendOpcache插件说明

上面讲解了目前常见的PHP缓存加速插件：APC、eAccelerator、XCache，从PHP 5.5开始，官方已经集成了新一代的缓存加速插件，其名字为ZendOpcache，功能和前三者相似但又有少许不同，据官方说，ZendOpcache缓存速度更快。

这几个PHP加速插件的主要原理基本相同，就是把PHP执行后的数据缓存到内存中从而避免重复的编译过程，使其能够直接使用缓存中已编译的代码，从而提高速度，降低服务器负载。它们的效率是显而易见的，一些大型的CMS，每次打开一个页面要调用数十个PHP文件，执行数万行代码，效率可想而知，安装上述加速器后，打开页面的速度明显加快。

PHP 5.5以上版本，支持ZendOpcache很简单，只需在编译PHP 5.5时加上--enable-opcache就行了。其实，在PHP5.5版本以前，ZendOpcache也有独立的软件，并且也支持低版本的PHP 5.2.*、PHP5.3.*、PHP 5.4.*。下面就以PHP 5.3版本为例讲解ZendOpcache软件，以PHP扩展插件的方式介绍安装步骤。

官方下载地址为：http://pecl.php.net/package/ZendOpcache 。

2.ZendOpcache插件安装过程

这里以当前的最新稳定版本：zendopcache-7.0.5.tgz为例进行介绍，由于我们使用的PHP版本为5.3，因此需要以PHP扩展的插件的方式安装，不能使用PHP编译直接加参数（--enable-opcache）的方式（PHP 5.5以上才可以），操作步骤及过程如下。

具体的安装命令集如下：

```
cd /home/oldboy/tools/
wget -q http:


//pecl.php.net/get/zendopcache-7.0.5.tgz
tar xf zendopcache-7.0.5.tgz
cd zendopcache-7.0.5
/application/php/bin/phpize
./configure --enable-opcache --with-php-config=/application/php/bin/php-config
make
make install
ls -l /application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
cd ..
```

详细的安装过程如下：

```
[root@www tools]# cd /home/oldboy/tools/
[root@www tools]# wget -q http:


//pecl.php.net/get/zendopcache-7.0.5.tgz
[root@www tools]# ls zendopcache-7.0.5.tgz
zendopcache-7.0.5.tgz
[root@www tools]# tar xf zendopcache-7.0.5.tgz
[root@www tools]# cd zendopcache-7.0.5
[root@www zendopcache-7.0.5]# /application/php/bin/phpize
Configuring for:
```

```
PHP Api Version:



         20090626
Zend Module Api No:



      20090626
Zend Extension Api No:



   220090626
[root@www zendopcache-7.0.5]# ./configure --enable-opcache --with-php-config=/applic
[root@www zendopcache-7.0.5]# make
...skip...
Build complete.
Don't forget to run 'make test'.
[root@www zendopcache-7.0.5]# make install
Installing shared extensions:



 /application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
[root@www zendopcache-7.0.5]# ls -l /application/php5.3.27/lib/php/extensions/no-deb
total 1536
-rwxr-xr-x 1 root root 417069 May  1 12:



48 eaccelerator.so
-rwxr-xr-x 1 root root 491814 May  1 15:



06 opcache.so
                    #<==最后生成了



opcache.so模块就表示



opcache安装成功



-rwxr-xr-x 1 root root 658468 May  1 14:



33 xcache.so
[root@www zendopcache-7.0.5]# cd ..
```

7.3 安装数据库缓存及其他PHP扩展插件

7.3.1 安装PHP Memcached扩展插件

1.Memcached缓存软件说明

Memcached是一个开源的、支持高性能、高并发及分布式的内存缓存服务软件，从名称上看，前3个字符的单词Mem就是内存的意思，而后面5个字符的单词Cache就是缓存的意思，最后字符d，是daemon的意思，表示服务器端进程模式服务。

Memcached服务分为服务器端和客户端两部分，其中，服务器端软件的名字形如Memcached-1.4.13.tar.gz，客户端软件的名字形如Memcache-2.27.tar.gz。

Memcached诞生于2003年，最初由LiveJournal的Brad Fitzpatrick开发完成。Memcache是整个项目的名称，而Memcached是服务器端的主程序名，因其协议简单，且支持高并发而被广泛使用。

在传统场景下，多数Web应用都将数据保存到RDBMS中，www服务器从中读取数据并在浏览器中显示。但随着数据量的增大、访问的集中，就会出现RDBMS的负担加重、数据库响应缓慢、网站打开延迟等

问题。

这时就需要Memcached了。Memcached是高性能的分布式内存缓存服务。使用Memcached的主要目的是，通过在自身内存中缓存数据库的查询结果，减少数据库访问次数，以提高动态Web应用的速度，提高网站架构的并发能力和可扩展性。

Memcached服务通过在事先规划好的系统内存空间中临时缓存数据库中的各类数据，以达到减少前端业务对数据库的直接高并发访问，从而提升大规模网站集群中动态服务的并发访问能力。

生产场景的Memcached服务一般被用来保存网站中经常被读取的对象或数据，就像我们的客户端浏览器把经常访问的网页缓存起来一样，通过内存缓存来存取对象或数据要比磁盘存取快很多，因为磁盘是机械的介质，因此，在当今的IT企业中，Memcached的应用范围很广。

图7-3是Memcached缓存架构逻辑图。

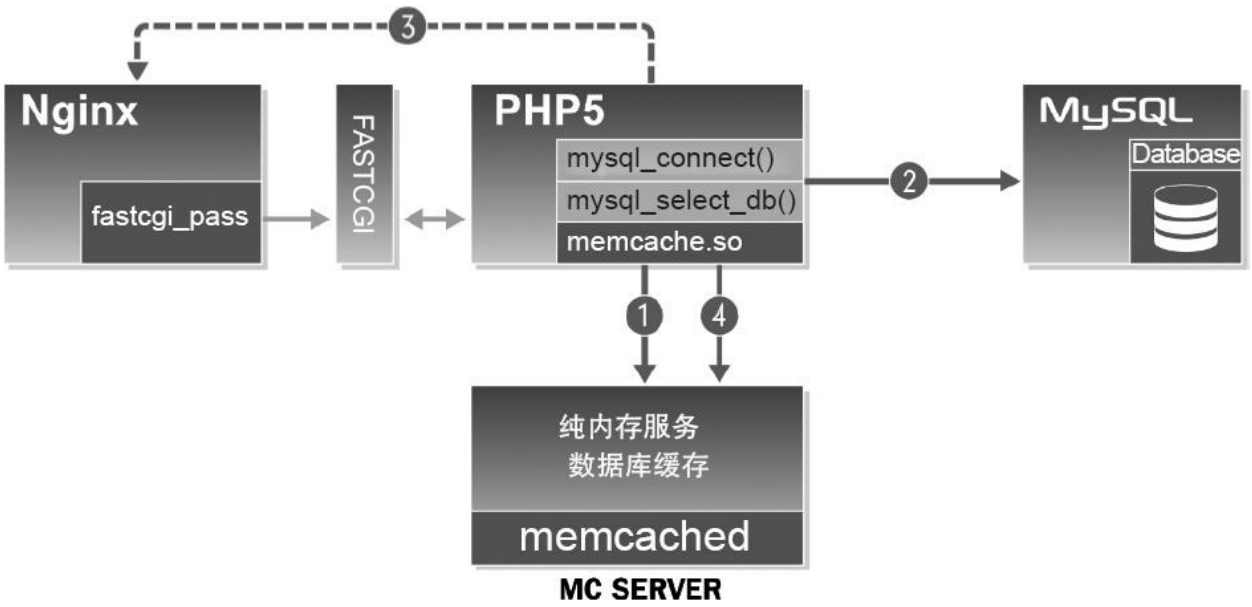


图7-3　LNMP环境Memcached客户端缓存工作原理逻辑图

Memcached服务的工作步骤如下。

第一步：程序首先检查客户端请求的数据在Memcached服务的缓存中是否存在，如果存在，直接把请求的数据返回，不再请求后端数据库。

第二步：如果请求的数据在Memcached缓存中不存在，则程序会去Memcahced后端的数据库服务。

第三步：把从数据库中取到的数据返回给客户端。

第四步：同时把新取到的数据库的数据缓存一份到Memcached服务缓存中，下次同样的请求就直接从Memcached服务缓存返回数据，从而减轻数据库的访问压力。

2.Memcached缓存PHP扩展插件安装

前文已经提过，Memcached分为服务器端软件和客户端插件两部分，本文是Memcached客户端PHP的扩展插件（memcache-2.2.7.tgz）在PHP环境中的安装，用于访问Memcached服务器端数据。

PHP的Memcached扩展插件下载地址为：http://pecl.php.net/package/memcache 。

PHP的Memcached客户端扩展插件安装命令集如下：

```
cd /home/oldboy/tools/
wget -q http:


//pecl.php.net/get/memcache-2.2.7.tgz
tar zxf memcache-2.2.7.tgz
cd memcache-2.2.7
/application/php/bin/phpize
./configure  --enable-memcache  --with-php-config=/application/php/bin/php-config
make
make install
cd ../
```

如果安装的是memcache-2.2.4.tgz，可能会报错：make：*** [memcache.lo]Error 1。

解决方法为使用如下命令：

```
cp memcache.loT memcache.lo
```

整个操作过程如下：

```
[root@www tools]# cd /home/oldboy/tools/
[root@www tools]# wget -q http:



//pecl.php.net/get/memcache-2.2.7.tgz
[root@www tools]# ls -l memcache-2.2.7.tgz
-rw-r--r-- 1 root root 36459 Mar 30 15:



05 memcache-2.2.7.tgz
[root@www tools]# tar zxf memcache-2.2.7.tgz
[root@www tools]# cd memcache-2.2.7
[root@www memcache-2.2.7]# /application/php/bin/phpize
Configuring for:




PHP Api Version:



         20090626
Zend Module Api No:



      20090626
Zend Extension Api No:



   220090626
[root@www memcache-2.2.7]# ./configure  --enable-memcache  --with-php-config=/applic
[root@www memcache-2.2.7]# make
...skip...
Build complete.
Don't forget to run 'make test'.
[root@www memcache-2.2.7]# make install
Installing shared extensions:



 /application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
[root@www memcache-2.2.7]# cd ..
[root@www tools]# ls -l /application/php5.3.27/lib/php/extensions/no-debug-non-zts-2
total 1780
-rwxr-xr-x 1 root root 417069 May  1 12:



48 eaccelerator.so
-rwxr-xr-x 1 root root 246728 May  1 15:



51 memcache.so
#<==最后生成了
```

memcache.so模块，表示

memcache扩展插件成功安装

-rwxr-xr-x 1 root root 491814 May 1 15:

06 opcache.so
-rwxr-xr-x 1 root root 658468 May 1 14:

33 xcache.so

7.3.2　安装PDO_MYSQL扩展模块

1.PDO_MYSQL扩展插件说明

PDO扩展为PHP访问数据库定义了一个轻量级一致性的接口，它提供了一个数据访问抽象层，这样，无论使用的是什么数据库，都可以通过一致的函数执行查询并获取数据。

PDO_MYSQL扩展插件下载地址为：http://pecl.php.net/get/PDO_MYSQL-1.0.2.tgz 。[1]

2.PDO_MYSQL扩展插件的安装过程

PDO_MYSQL的安装有两种方法，一种是插件方式安装，另一种是编译PHP时加入PDO_MYSQL支持，直接指定PHP的对应PDO_MYSQL编译参数安装，例如：--with-pdo-mysql=mysqlnd，同时PHP的环境也可以不装MySQL软件，直接指定如下参数--with-mysql=mysqlnd，即可让PHP支持连接MySQL数据库。当然了，建议读者跟着本书的演示进行，先把路走通了，再去测试老男孩说明方案中提到的方法。此处采用工作中常用的PHP扩展插件方式安装。

完整安装PDO_MYSQL的命令集如下：

```
cd /home/oldboy/tools
```

```
wget -q http:


//pecl.php.net/get/PDO_MYSQL-1.0.2.tgz
tar xf PDO_MYSQL-1.0.2.tgz
cd PDO_MYSQL-1.0.2/
/application/php/bin/phpize
./configure  --with-php-config=/application/php/bin/php-config --with-pdo-mysql=/app
#configure的参数路径要正确配置。


/application/mysql为


MySQL的安装路径，要先安装好


MySQL，本文讲解的是


LNMP一体的环境



make
make install
ls /application/php5.3.27/lib/php/extensions/no-debug-zts-20090626/
cd ../
```

安装操作过程如下：

```
[root@www tools]# wget http:


//pecl.php.net/get/PDO_MYSQL-1.0.2.tgz
[root@www tools]# ls -l PDO_MYSQL-1.0.2.tgz
-rw-r--r-- 1 root root 14778 Mar 30 15:


05 PDO_MYSQL-1.0.2.tgz
[root@www tools]# tar xf PDO_MYSQL-1.0.2.tgz
[root@www tools]# cd PDO_MYSQL-1.0.2
[root@www tools]# /application/php/bin/phpize
[root@www PDO_MYSQL-1.0.2]# ./configure  --with-php-config=/application/php/bin/php-
[root@www PDO_MYSQL-1.0.2]# make
...skip...
Build complete.
Don't forget to run 'make test'.
```

```
[root@www PDO_MYSQL-1.0.2]# make install
Installing shared extensions:



/application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
[root@www PDO_MYSQL-1.0.2]# ls -l
/application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
total 1936
-rwxr-xr-x 1 root root 417069 May  1 12:



48 eaccelerator.so
-rwxr-xr-x 1 root root 246728 May  1 15:



51 memcache.so
-rwxr-xr-x 1 root root 491814 May  1 15:



06 opcache.so
-rwxr-xr-x 1 root root 156004 May  1 16:



05 pdo_mysql.so
                    #<==出现



pdo_mysql.so模块，表示



pdo_mysql成功安装



-rwxr-xr-x 1 root root 658468 May  1 14:



33 xcache.so
[root@www PDO_MYSQL-1.0.2]# cd ..
```

如果遇到下面的警告可以忽略，继续安装。

```
[root@www PDO_MYSQL-1.0.2]# /application/php/bin/phpize
Configuring for:
```

PHP Api Version:

20090626
Zend Module Api No:

20090626
Zend Extension Api No:

220090626
config.m4:

104:

warning:

AC_CACHE_VAL (

pdo_inc_path,

...) :

suspicious cache-id,

must contain _cv_ to be cached
../../lib/autoconf/general.m4:

1974:

AC_CACHE_VAL is expanded from...
../../lib/autoconf/general.m4:

1994:

AC_CACHE_CHECK is expanded from...
aclocal.m4:

2754:

PHP_CHECK_PDO_INCLUDES is expanded from...
config.m4:

104:

the top level
config.m4:

104:

warning:

AC_CACHE_VAL (

pdo_inc_path,

...) :

suspicious cache-id,

must contain _cv_ to be cached
../../lib/autoconf/general.m4:

1974:

AC_CACHE_VAL is expanded from...
../../lib/autoconf/general.m4:

```
1994:


 AC_CACHE_CHECK is expanded from...
aclocal.m4:


2754:


 PHP_CHECK_PDO_INCLUDES is expanded from...
config.m4:


104:


 the top level
```

---

[1] 技巧：使用谷歌搜索关键字“PDO_MYSQL-1.0.2.tgz download”。

7.4　安装其他的PHP扩展插件模块

7.4.1　安装图像处理程序及imagick扩展模块

1.安装ImageMagick图像软件

ImageMagick是一套功能强大、稳定而且免费的工具集和开发包，可以用来读、写和处理超过89种基本格式的图片文件，包括流行的tiff、jpeg、gif、png、pdf，以及PhotoCD等。利用ImageMagick，可以根据Web应用程序的需要动态生成图片，还可以对一个（或一组）图片进行改变大小、旋转、锐化、减色或增加特效等操作，并将操作的结果以相同格式或其他格式保存。对图片的操作，即可以通过命令行进行，也可以用C/C++、Perl、Java、PHP、Python或Ruby编程来完成。同时ImageMagick提供了一个高质量的2D工具包，部分支持SVG。现在，ImageMagic的主要精力集中在加强性能、减少bug，以及提供稳定的API和ABI上。

ImageMagick的常见功能如下：

·将图片从一个格式转换成另一个格式，包括直接转换成图标。

·可以改变图片尺寸，旋转、锐化（sharpen）、减色，设置图片特

效。

·对图片设置各种尺寸缩略图。

·将图片设置为可以适应于Web背景的透明图片。

·将一组图片做成gif动画，直接convert。

·将几张图片做成一张组合图片。

·在一个图片上写字或画图形，带文字阴影和边框渲染。

·给图片加边框或框架。

·取得一些图片的特性信息。

它几乎包括了gimp可以实现的所有常规插件功能，甚至包括各种曲线参数的渲染功能。ImageMagick的下载地址为：http://download.chinaunix.net/download/0001000/95.shtml ，请提前下载好放到指定服务器的目录下。

安装ImageMagick的命令集如下：

```
ls -l ImageMagick-6.7.9-9.tar.xz
tar xf ImageMagick-6.7.9-9.tar.xz
cd ImageMagick-6.7.9-9
./configure
make && make install
cd ../
#提示：此图像软件安装时
```

```
make步骤耗时较长
```

具体安装过程如下：

```
[root@www tools]# ls -l ImageMagick-6.7.9-9.tar.xz
-rw-r--r-- 1 root root 8363720 May  1 16:



16 ImageMagick-6.7.9-9.tar.xz
[root@www tools]# tar xf ImageMagick-6.7.9-9.tar.xz
[root@www tools]# cd ImageMagick-6.7.9-9
[root@www ImageMagick-6.7.9-9]# ./configure
[root@www ImageMagick-6.7.9-9]# make && make install
[root@www ImageMagick-6.7.9-9]# cd ../提示：此步不是安装



PHP的扩展，因此，没有生成



.so的文件
```

下面来看看ImageMagick安装报错及解决方法。

问题1：make步骤出错。

示例如下：

```
cd PerlMagick && /usr/bin/perl Makefile.PL
Can't locate ExtUtils/MakeMaker.pm in @INC （



@INC contains:



 /usr/local/lib64/perl5 /usr/local/share/perl5 /usr/lib64/perl5/vendor_perl /usr/sha
```

```
 at Makefile.PL line 24.
BEGIN failed--compilation aborted at Makefile.PL line 24.
make[1]:


 *** [PerlMagick/Makefile] Error 2
make[1]:


 Leaving directory `/home/oldboy/tools/ImageMagick-6.5.1-2'
make:


 *** [all] Error 2
```

解决方案：采用yum install perl-devel-y命令。

解决思路：

方法一，根据如下内容：

```
Can't locate ExtUtils/MakeMaker.pm in @INC (

@INC contains:

 /usr/local/lib64/perl5 /usr/local/share/perl5 /usr/lib64/perl5/vendor_perl /usr/sha

 at Makefile.PL line 24.
```

可以看到，上述内容中有Makefile.PL、/usr/lib64/perl5/vendor_perl和Perl语言的字样，因此可以试着使用yum install perl-devel-y命令安装相关包，看看是否可以解决问题。前文讲解过PHP基础依赖包的安装方

法，遇到问题首先要考虑安装软件依赖的devel包。多思考水平自然就提高了，哪怕思考得不对也能锻炼思维，尝试的次数多了，准确率就提高了。

方法二，寻找唯一并且有特征的错误提示去搜索，如下：

```
[PerlMagick/Makefile] Error 2
```

图7-4和图7-5是通过搜索得到的内容，利用搜索引擎快速搜索需要的信息，是运维人员重要的能力，一般报错问题都可以通过搜索解决，谷歌的搜索效果最好。

图7-4　ImageMagick安装报错搜索结果

图7-5　ImageMagick安装报错搜索解决答案参考

问题2：出现如下报错信息“gcc：internal compiler error：Killed（program cc1）”

解决方法：通过查看dmesg发现下述错误信息：

[2517343.500178]Out of memory：Kill process 5051（cc1）score 632 or sacrifice child

原因：安装VM虚拟机时设置的内存小了（400MB太小，大于

512MB就没事了），加1GB的VM内存较好。

2.安装imagick PHP扩展插件

imagick插件工作需要ImageMagick软件的支持，所以，必须要先安装ImageMagick，否则会报错。

imagick插件是一个可以供PHP调用ImageMagick功能的扩展模块。使用这个扩展可以使PHP具备和ImageMagick相同的功能。

安装了ImageMagick图像程序后，再安装PHP的扩展imagick插件，才能使用ImageMagick提供的api进行图片的创建与修改、压缩等操作，因为它们都集成在imagick这个PHP扩展中。

其安装命令集如下（需要提前下载好放到指定服务器的目录下）：

```
tar zxf imagick-2.3.0.tgz
cd imagick-2.3.0
/application/php/bin/phpize
./configure --with-php-config=/application/php/bin/php-config
#configure的参数路径要正确配置。



make
make install
ls /application/php5.3.27/lib/php/extensions/no-debug-zts-20090626
cd ../
```

安装过程如下：

```
[root@www tools]# tar zxf imagick-2.3.0.tgz
[root@www tools]# cd imagick-2.3.0
```

```
[root@www imagick-2.3.0]# /application/php/bin/phpize
Configuring for:



PHP Api Version:



         20090626
Zend Module Api No:



      20090626
Zend Extension Api No:



   220090626
[root@www imagick-2.3.0]# make
Build complete.
Don't forget to run 'make test'.
[root@www imagick-2.3.0]# make install
Installing shared extensions:



 /application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
[root@www imagick-2.3.0]# ls -l /application/php5.3.27/lib/php/extensions/no-debug-n
total 2984
-rwxr-xr-x 1 root root  417069 May  1 12:



48 eaccelerator.so
-rwxr-xr-x 1 root root 1073033 May  1 16:



35 imagick.so
                                #<==最后生成了



imagick.so模块就对了



-rwxr-xr-x 1 root root  246728 May  1 15:



51 memcache.so
-rwxr-xr-x 1 root root  491814 May  1 15:



06 opcache.so
```

```
-rwxr-xr-x 1 root root  156004 May  1 16:


05 pdo_mysql.so
-rwxr-xr-x 1 root root  658468 May  1 14:


33 xcache.so
```

7.4.2　检查所有PHP扩展插件模块安装的成果

到此为止，常见的PHP扩展插件安装得就差不多了，下面看看安装的成果吧。

```
[root@www tools]# ls -l /application/php5.3.27/lib/php/extensions/no-debug-non-zts-2
total 2984
-rwxr-xr-x 1 root root  417069 May  1 12:


48 eaccelerator.so
-rwxr-xr-x 1 root root 1073033 May  1 16:


35 imagick.so
-rwxr-xr-x 1 root root  246728 May  1 15:


51 memcache.so
-rwxr-xr-x 1 root root  491814 May  1 15:


06 opcache.so
-rwxr-xr-x 1 root root  156004 May  1 16:


05 pdo_mysql.so
-rwxr-xr-x 1 root root  658468 May  1 14:


33 xcache.so
```

当前一共有6个常用扩展模块，其他的若有需要以后可以后续安装。其中，eaccelerator.so、opcache.so、xcache.so属于同类软件，生产环境中安装其中一种即可，否则，可能会引起同时使用冲突，本书全都

介绍了，目的是让大家了解方法。另外，pdo_mysql.so、imagick.so属于功能软件，可选安装，memcache.so是数据库缓存软件，可选安装。

7.5　配置PHP加速与缓存相关的扩展插件模块

7.5.1　配置Memcache/PDO_MYSQL/imagick模块生效

1.修改PHP的配置文件php.ini

修改php.ini的配置文件过程如下：

1）执行vi/application/php/lib/php.ini命令，编辑查找extension_dir="./"参数，修改为extension_dir="/application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/"，这个extension_dir对应的路径就是前文编译的模块所在的路径。


提示：默认的PHP配置文件路径为/application/php/lib/php.ini，可以通过在编译PHP时添加参数指定php.ini的配置路径--with-config-file-path=/application/php5.3.27/lib/etc，如果不指定编译路径，默认为/application/php/lib/。

快速操作命令的方法如下：

```
cd /application/php/lib/
```

```
/bin/cp php.ini php.oldboy.20150501                    #<==修改前备份


sed -i 's#;

 extension_dir = "./"#extension_dir = "/application/php5.3.27/lib/php/extensions/no·


grep "extension_dir =" php.ini                         #<==替换后，过滤检查结果
```

2）在vim命令模式下按Shift+G键跳到文件结尾，增加如下几行，然后保存：

```
extension = memcache.so
extension = pdo_mysql.so
extension = imagick.so
```

快速操作命令的方法如下：

```
cat >> /application/php/lib/php.ini<<EOF;


--cache ext start by oldboy 2015-05-01--
extension = memcache.so
extension = pdo_mysql.so
extension = imagick.so;


--cache ext end by oldboy 2015-05-01---
EOF
tail -5 /application/php/lib/php.ini提示：注释不是


"#", 是
```

```
";


"
```

操作过程如下（老男孩学习思想：在使用中记忆Linux命令）：

```
[root@oldboy tools]# cd /application/php/lib/
[root@oldboy lib]# cp php.ini php.ini.oldboy.20150501 #<==操作前养成备份习惯



[root@oldboy lib]# cp php.ini php.ini.tmp
                          #<==搞不清楚可以先用测试文件测试



[root@oldboy lib]# sed -i 's#;


 extension_dir = "./"#extension_dir = "/application/php5.3.27/lib/php/extensions/no·
[root@oldboy lib]# grep extension_dir php.ini.tmp
                          #<==测试文件测试


OK了，同样命令应用到正式环境



extension_dir = "/application/php5.3.27/lib/php/extensions/no-debug-non-zts-2009062(
...skip...
[root@oldboy lib]# grep "extension_dir =" php.ini ;


 extension_dir = "./"
...skip...
 [root@oldboy lib]# sed -i 's#;


 extension_dir = "./"#extension_dir = "/application/php5.3.27/lib/php/extensions/no·
[root@oldboy lib]# grep "extension_dir =" php.ini
                          #<==操作后养成检查习惯



extension_dir = "/application/php5.3.27/lib/php/extensions/no-debug-non-zts-2009062(
...skip...
```

```
[root@www lib]# cat >> /application/php/lib/php.ini<<EOF
> ;


--cache ext start by oldboy 2015-05-01--
> extension = memcache.so
> extension = pdo_mysql.so
> extension = imagick.so
> ;


--cache ext end by oldboy 2015-05-01---
> EOF
[root@www lib]# tail -5 /application/php/lib/php.ini;


--cache ext start by oldboy 2015-05-01--
extension = memcache.so
extension = pdo_mysql.so
extension = imagick.so;


--cache ext end by oldboy 2015-05-01---
```

2.检查配置的相关模块生效情况

（1）重启PHP服务，编写测试程序phpinfo

首先，重启PHP服务，并检查模块生效情况，命令如下：

```
[root@www lib]# pkill php-fpm
[root@www lib]# ps -ef|grep php-fpm|grep -v grep
[root@www lib]# /application/php/sbin/php-fpm
[root@www lib]# ps -ef|grep php-fpm|grep -v grep|wc -l
3
```

然后在前文提到的blog程序的站点目录下面增加如下phpinfo.php代码文件：

```
[root@www extra]# pwd
/application/nginx/conf/extra
```

```
[root@www extra]# cat blog.conf
    server {
        listen       80;



        server_name  blog.etiantian.org;



        location / {
            root   html/blog;



            index  index.php index.html index.htm;



        }
        location ~ .*\. (


php|php5)



$ {
            root html/blog;



            fastcgi_pass  127.0.0.1:


9000;



            fastcgi_index index.php;



            include fastcgi.conf;



        }
    }
```

命令如下：

```
cat >>/application/nginx/html/blog/view_info.php<<EOF
<php phpinfo () ;



>
EOF
[root@www extra]# cat /application/nginx/html/blog/view_info.php
<php phpinfo () ;



>
```

（2）检查Memcached扩展插件

配好客户端的host解析，然后在浏览器中输入http://blog.etiantian.org/view_info.php 页面的地址，出现的内容如图7-6所示。

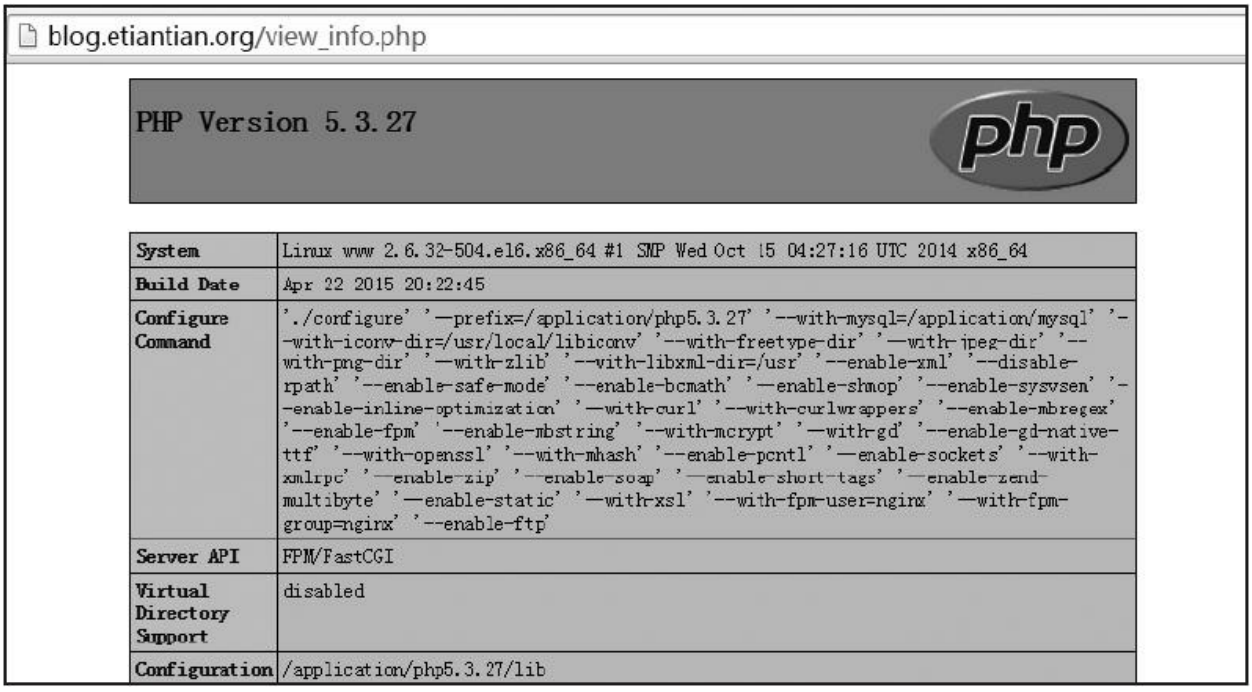


图7-6　PHP的phpinfo信息展示

通过快捷键Ctrl+F进行页面搜索，如果查找到如图7-7所示的内容，表示Memcached插件已生效。

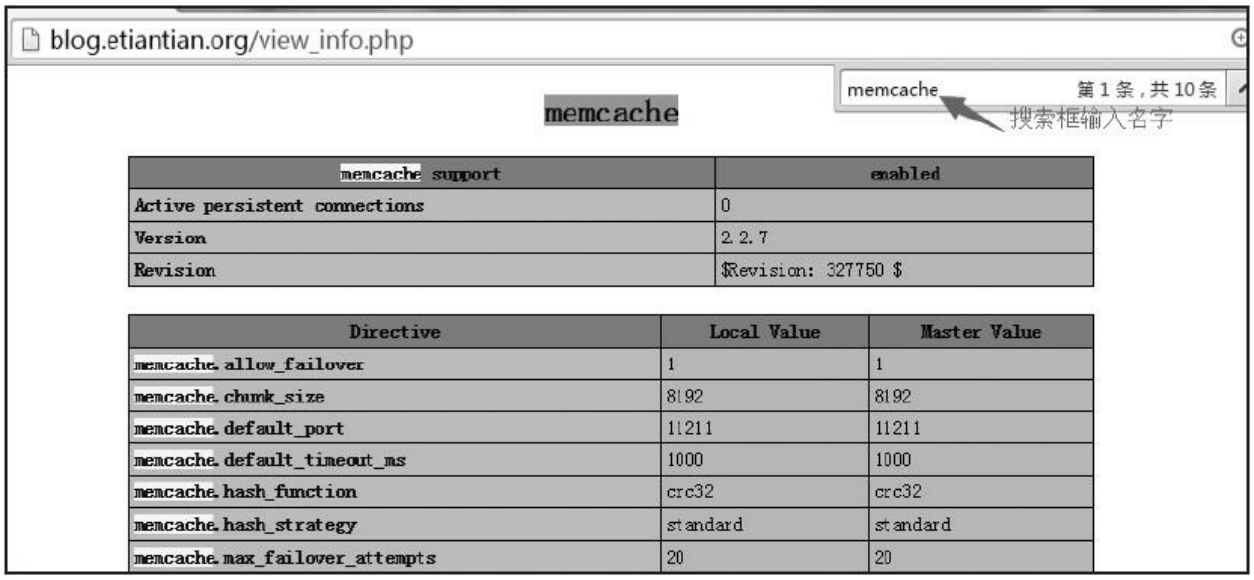


图7-7　PHP的扩展插件memcache客户端展示

（3）检查PDO_MYSQL扩展插件

同理，搜索检查PDO_MYSQL扩展插件，如图7-8所示。

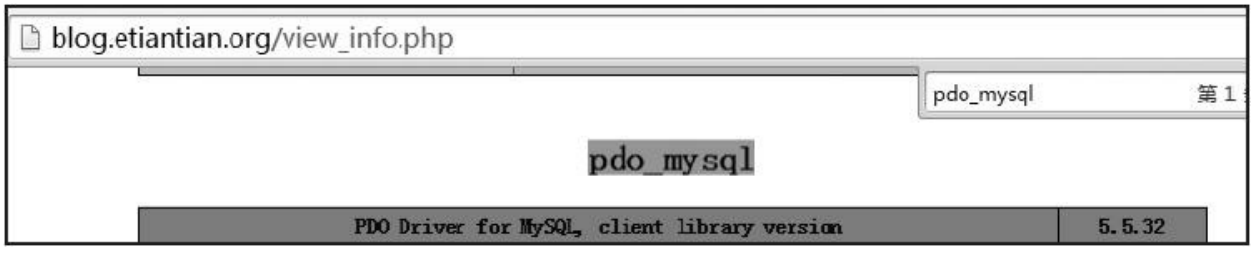


图7-8　PHP的扩展插件pdo_mysql

（4）检查IMAGICK扩展插件

同理，搜索检查IMAGICK扩展插件，如图7-9所示。

到此为止，pdo_mysql.so、imagick.so、memcache.so这三个PHP的扩展插件就全部安装及配置完毕，后面将配置其余的缓存插件。

7.5.2　配置eAccelerator插件生效并优化参数

1.配置eAccelerator缓存目录

配置命令1：　配置eAccelerator缓存目录，命令如下：

```
mkdir -p /tmp/eaccelerator
#<===此目录可以用


tmpfs内存文件系统或


SSD固态硬盘来存储。



chown -R nginx.nginx /tmp/eaccelerator提示：上面


chown 后的用户是


nginx的用户


.
```

操作过程如下：

```
[root@www extra]# cd /application/php/lib/
[root@www lib]# mkdir -p /tmp/eaccelerator
[root@www lib]# chown -R nginx.nginx /tmp/eaccelerator
[root@www lib]# ls -ld /tmp/eaccelerator/
drwxr-xr-x 2 nginx nginx 4096 May  1 17:
```

```
42 /tmp/eaccelerator/
```

imagick

imagick

imagick module	enabled
imagick module version	2.3.0
imagick classes	Imagick, ImagickDraw, ImagickPixel, ImagickPixelIterator
ImageMagick version	ImageMagick 6.7.9-9 2015-05-01 Q16 http://www.imagemagick.org
ImageMagick copyright	Copyright (C) 1999-2012 ImageMagick Studio LLC
ImageMagick release date	2015-05-01
ImageMagick Number of supported formats:	187
ImageMagick Supported formats	3FR, A, AAI, AI, ART, ARW, AVI, AVS, B, BMP, BMP2, BMP3, C, CAL, CALS, CANVAS, CAPTION, CIN, CIP, CLIP, CMYK, CMYKA, CR2, CRW, CUR, CUT, DCM, DCR, DCX, DDS, DFONT, DNG, DPX, EPDF, EPI, EPS, EPS2, EPS3, EPSF, EPSI, ERF, FAX, FITS, FRACTAL, FTS, G, G3, GIF, GIF87, GRADIENT, GRAY, HALD, HDR, HISTOGRAM, HRZ, HTM, HTML, ICB, ICO, ICON, INFO, INLINE, IPL, JNG, JPEG, JPG, K, K25, KDC, LABEL, M, M2V, M4V, MAC, MAP, MAT, MATTE, MEF, MIFF, MNG, MONO, MOV, MP4, MPC, MPEG, MPG, MRW, MSL, MSVG, MTV, MVG, NEF, NRW, NULL, O, ORF, OTB, OTF, PAL, PALM, PAM, PANGO, PATTERN, PBM, PCD, PCDS, PCL, PCT, PCX, PDB, PDF, PDFA, PEF, PES, PFA, PFB, PFM, PGM, PICON, PICT, PIX, PJPEG, PLASMA, PNG, PNG24, PNG32, PNG8, PNM, PPM, PREVIEW, PS, PS2, PS3, PSB, PSD, PWP, R, RADIAL-GRADIENT, RAF, RAS, RGB, RGBA, RGBO, RLA, RLE, SCR, SCT, SFW, SGI, SHTML, SR2, SRF, STEGANO, SUN, SVG, SVGZ, TEXT, TGA, THUMBNAIL, TILE, TIM, TTC, TTF, TXT, UIL, UYVY, VDA, VICAR, VID, VIFF, VST, WBMP, WMV, WPG, X, X3F, XBM, XC, XCF, XPM, XPS, XV, XWD, Y, YCbCr, YCbCrA, YUV

Directive	Local Value	Master Value
imagick.locale_fix	0	0
imagick.progress_monitor	0	0

图7-9　PHP的扩展插件imagick

配置命令2：　配置eAccelerator参数，命令如下：

```
cat >> /application/php/lib/php.ini<<EOF
[eaccelerator]
```

```
extension=eaccelerator.so
eaccelerator.shm_size="64"
eaccelerator.cache_dir="/tmp/eaccelerator"
eaccelerator.enable="1"
eaccelerator.optimizer="1"
eaccelerator.check_mtime="1"
eaccelerator.debug="0"
eaccelerator.filter=""
eaccelerator.shm_max="0"
eaccelerator.shm_ttl="3600"
eaccelerator.shm_prune_period="3600"
eaccelerator.shm_only="0"
eaccelerator.compress="1"
eaccelerator.compress_level="9"
EOF
```

操作后检查配置结果，命令如下：

```
[root@www lib]# tail -15 php.ini
[eaccelerator]
extension=eaccelerator.so
eaccelerator.shm_size="64"
eaccelerator.cache_dir="/tmp/eaccelerator"
eaccelerator.enable="1"
eaccelerator.optimizer="1"
eaccelerator.check_mtime="1"
eaccelerator.debug="0"
eaccelerator.filter=""
eaccelerator.shm_max="0"
eaccelerator.shm_ttl="3600"
eaccelerator.shm_prune_period="3600"
eaccelerator.shm_only="0"
eaccelerator.compress="1"
eaccelerator.compress_level="9"
```

下面的表7-2给出了eAccelerator配置参数的详细说明。

表7-2　eAccelerator配置参数详解

eAccelerator 参数	解释说明
[eaccelerator]	开始 eAccelerator 加速模块配置
extension=eaccelerator.so	加载 eAccelerator 加速模块，路径相对于 extension_dir 的配置
eaccelerator.shm_size="64"	存储缓存数据的共享内存大小，如果为 0，则最大值看内核配置 /proc/sys/kernel/shmmax
eaccelerator.cache_dir="/tmp/eaccelerator"	磁盘缓存存储路径，缓存内容为 precompiled code、session data、content 和 user entries。默认路径为 "/tmp/eaccelerator"
eaccelerator.enable="1"	eAccelerator 缓存生效开关，1 为默认值，即生效；0 为关闭，不生效
eaccelerator.optimizer="1"	加速 PHP 代码执行速度，1 为默认值，表示激活；0 为不激活。用于缓存前的代码加速
eaccelerator.check_mtime="1"	检查缓存修改时间，决定代码是否需要重新编译，1 为激活，是默认值
eaccelerator.debug="0"	缓存加速调试，0 为关闭，1 为打开，打开后可以看到缓存命中信息
eaccelerator.filter=""	设定对象是否缓存规则，空表示不设定
eaccelerator.shm_max="0"	可以被放置的缓存最大值，0 是不限制
eaccelerator.shm_ttl="3600"	缓存文件的生存期
eaccelerator.shm_prune_period="3600"	当共享内存空间不够时，从共享内存中移除老数据的时间周期
eaccelerator.shm_only="0"	是否允许缓存数据到磁盘，0 为允许，但是对于 session data and content caching 无影响
eaccelerator.compress="1"	是否开启压缩，1 为开启
eaccelerator.compress_level="9"	压缩级别，9 为最高

根据内容指定是否缓存到共享内存或磁盘的其他参数：

```
eaccelerator.keys     = "shm_and_disk" #<==控制

keys缓存位置

eaccelerator.sessions = "shm_and_disk" #<==控制

sessions缓存位置

eaccelerator.content  = "shm_and_disk" #<==控制内容缓存位置
```

上述3个参数可选的值为：

```
shm_and_disk:

cache data in shared memory and on disk （

default value）

shm:

cache data in shared memory or on disk if shared memory is full or data size greater
shm_only:

cache data in shared memory
disk_only:

cache data on disk
none:

don't cache data
```

更多信息请参考https://github.com/eaccelerator/eaccelerator/wiki/Settings 。

2.检查eAccelerator加速配置情况

 提示：如果仅检查加速情况配置，可以不重启Apache。

执行如下PHP命令，测试缓存的配置情况：

```
[root@www lib]# /application/php/bin/php -v
PHP 5.2.6 (

cli)

  (

built:

 Jul  7 2010 08:

03:

45)

Copyright (

c)

 1997-2008 The PHP Group
Zend Engine v2.2.0,

 Copyright (

c)

 1998-2008 Zend Technologies
with eAccelerator v0.9.5.2,
```

```
 Copyright  (


c)


 2004-2006 eAccelerator,


 by eAccelerator
[root@www lib]# /application/php/bin/php -v
PHP 5.3.27  (


cli)


  (


built:


 Apr 22 2015 20:


23:


57)



Copyright  (


c)


 1997-2013 The PHP Group
Zend Engine v2.3.0,


 Copyright  (


c)
```

```
 1998-2013 Zend Technologies
    with eAccelerator v0.9.6.1,


 Copyright  (


c)


 2004-2010 eAccelerator,


 by eAccelerator
```

重启PHP服务的命令如下：

```
[root@www lib]# pkill php-fpm
[root@www lib]# ps -ef|grep php-fpm|grep -v grep
[root@www lib]# /application/php/sbin/php-fpm
[root@www lib]# ps -ef|grep php-fpm|grep -v grep|wc -l
3
```

现在通过phpinfo检查eAccelerator插件结果，如图7-10所示。

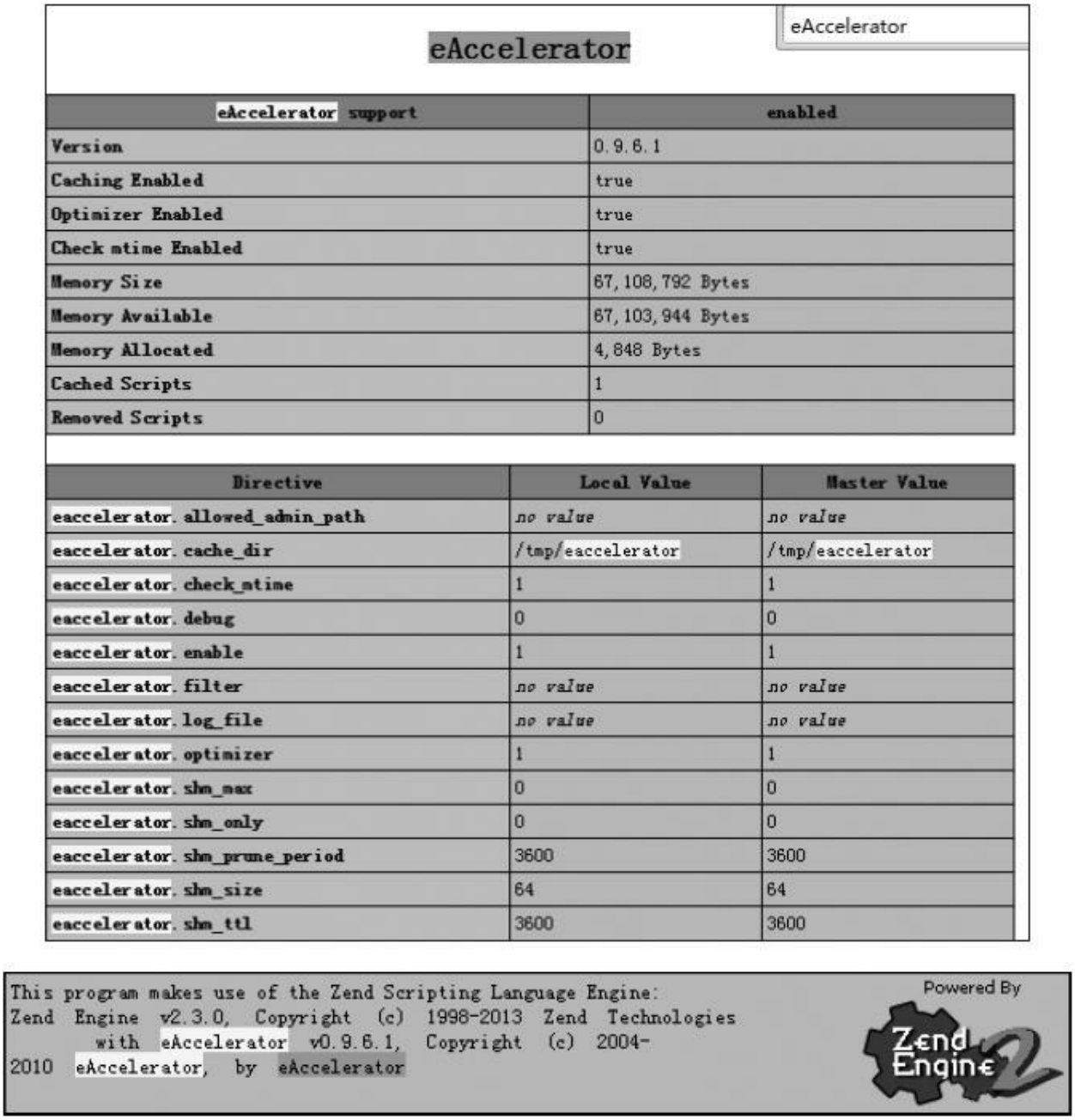
此时看看缓存目录/tmp/eaccelerator，结果如下：

```
[root@www lib]# ls -l /tmp/eaccelerator/
total 64
drwxrwxrwx 18 root root 4096 May  1 17:


46 0
drwxrwxrwx 18 root root 4096 May  1 17:


46 1
drwxrwxrwx 18 root root 4096 May  1 17:
```

```
46 2
drwxrwxrwx 18 root root 4096 May  1 17:


46 3
drwxrwxrwx 18 root root 4096 May  1 17:


46 4
drwxrwxrwx 18 root root 4096 May  1 17:


46 5
drwxrwxrwx 18 root root 4096 May  1 17:


46 6
drwxrwxrwx 18 root root 4096 May  1 17:


46 7
drwxrwxrwx 18 root root 4096 May  1 17:


46 8
drwxrwxrwx 18 root root 4096 May  1 17:


46 9
drwxrwxrwx 18 root root 4096 May  1 17:


46 a
drwxrwxrwx 18 root root 4096 May  1 17:


46 b
drwxrwxrwx 18 root root 4096 May  1 17:


46 c
drwxrwxrwx 18 root root 4096 May  1 17:


46 d
drwxrwxrwx 18 root root 4096 May  1 17:
```

```
46 e
drwxrwxrwx 18 root root 4096 May  1 17:


46 f
```

可以看到/tmp/eaccelerator/缓存目录下也有内容了。以上两个检查可以确认配置是否生效。



eAccelerator

eAccelerator

eAccelerator support	enabled
Version	0.9.6.1
Caching Enabled	true
Optimizer Enabled	true
Check mtime Enabled	true
Memory Size	67,108,792 Bytes
Memory Available	67,103,944 Bytes
Memory Allocated	4,848 Bytes
Cached Scripts	1
Removed Scripts	0

Directive	Local Value	Master Value
eaccelerator.allowed_admin_path	*no value*	*no value*
eaccelerator.cache_dir	/tmp/eaccelerator	/tmp/eaccelerator
eaccelerator.check_mtime	1	1
eaccelerator.debug	0	0
eaccelerator.enable	1	1
eaccelerator.filter	*no value*	*no value*
eaccelerator.log_file	*no value*	*no value*
eaccelerator.optimizer	1	1
eaccelerator.shm_max	0	0
eaccelerator.shm_only	0	0
eaccelerator.shm_prune_period	3600	3600
eaccelerator.shm_size	64	64
eaccelerator.shm_ttl	3600	3600

图7-10　PHP的扩展插件eAccelerator生效

3.访问PHP页面测试检查eAccelerator加速情况

重启PHP服务后，在浏览器里访问PHP页面，如出现访问phpinfo页面，就会有下面的缓存文件（其实上面已经访问过了）。

```
[root@www lib]# find /tmp/eaccelerator/ -type f|xargs file
/tmp/eaccelerator/0/0/eaccelerator-1502.1114:


 data
```


提示：eaccelerator-1502.1114就是Cache的内容，而且是phpinfo的页面缓存内容，类型为data。

4.使用tmpfs优化eAccelerator缓存目录

tmpfs是一种基于内存的文件系统，通常使用tmpfs作为数据临时存储，比本地磁盘存储快很多，此方法适用于临时使用的各类缓存场景。例如：上传图片时很多软件默认在/tmp下临时缓存切图、存放session数据，则可以让/tmp使用tmpfs文件系统来加快访问效率。本书将/tmp/eaccelerator挂载到tmpfs文件系统上，让访问缓存的数据更快。具体操作方法如下：

```
[root@www lib]# mount -t tmpfs  -o size=16m tmpfs /tmp/eaccelerator
#<==创建


16M大小的


tmpfs类型文件系统挂载到


/tmp/eaccelerator
[root@www lib]# df -h            #<==检查挂载情况


Filesystem      Size  Used Avail Use% Mounted on
```

```
/dev/sda3       7.1G  3.7G  3.1G  56% /
tmpfs           497M     0  497M   0% /dev/shm
/dev/sda1       190M   27M  153M  15% /boot
tmpfs            16M     0   16M   0% /tmp/eaccelerator
[root@www lib]# grep eacc /proc/mounts  #<==检查挂载情况



tmpfs /tmp/eaccelerator tmpfs rw,


relatime,


size=16384k 0 0
[root@www lib]# tail -1 /etc/fstab #<==配置永久挂载，生产场景


size可以根据实际情况调整



tmpfs                   /tmp/eaccelerator       tmpfs   size=16m        0 0
[root@www lib]# umount /tmp/eaccelerator/
[root@www lib]# grep eacc /proc/mounts
[root@www lib]# mount -a #<==测试永久挂载



[root@www lib]# grep eacc /proc/mounts
tmpfs /tmp/eaccelerator tmpfs rw,


relatime,


size=16384k 0 0
```

搞定！

7.5.3　配置XCache插件加速

 提示：XCache和eAccelerator功能相近，安装一个即可。考虑到知识的完整性，本节将其作为知识点来讲解，配置之前应删除eAccelerator的所有配置。

1.xcache.ini参数说明

xcache软件的解压目录/home/oldboy/tools/xcache-3.2.0/下存在一个名字为xcache.ini的配置文件，即XCache的配置文件。下面的表7-3给出了xcache配置文件参数的说明。

表7-3　XCache配置文件参数详解

XCache 配置文件参数	解释说明
[xcache-common] extension = xcache.so	加载 xcache.so，路径相对于 extension_dir 的配置。 自 3.0 版本开始不再支持使用 zend_extension 加载 XCache 的方式
[xcache.admin] xcache.admin.enable_auth = On	激活管理员认证
xcache.admin.user = "mOo" xcache.admin.pass = "md5 encrypted password"	指定 XCache 管理员用户名和密码，密码根据 http://xcache.lighttpd.net/demo/cacher/mkpassword.php 地址产生，留空表示禁止管理页面
[xcache]	开始 XCache 缓存参数配置段，下面所有的初始值即为默认值，除非明确说明
xcache.shm_scheme = "mmap"	设置 XCache 如何从系统分配共享内存
xcache.size = 60M	0 为禁止缓存，非 0 则启用缓存。需要注意系统所允许的 mmap 最大值
xcache.count = 1	指定将 Cache 切分成多少块，官方推荐设置为服务器 CPU 的数量 [root@www lib]# grep -c processor /proc/cpuinfo 1
xcache.slots = 8K	hash 槽个数的参考值，缓冲超过此数值时也没有任何问题（you can always store count(items) > slots）
xcache.ttl = 0	设定 Cache 对象的生存期 TTL（time to live），0 为永不过期 *****
xcache.gc_interval = 0	回收器扫描过期的对象回收内存空间的间隔，0 为不扫描，其他值的单位是秒
xcache.var_size = 4M xcache.var_count = 1 xcache.var_slots = 8K xcache.var_ttl = 0 xcache.var_gc_interval = 300	这几个值和上面的几个类似，只不过用于变量缓存，而不是 opcode 缓存
; N/A for /dev/zero xcache.readonly_protection = Off	如果启用了该参数，将略微降低性能，但会提高一定的安全系数。这个选项对于 xcache.mmap_path = /dev/zero 无效
xcache.mmap_path = "/dev/zero"	对于 *nix, xcache.mmap_path 是一个文件路径而非目录。如果要启用该参数，请使用 "/tmp/xcache" 这样的路径，而不是 "/dev/*"。如果开启了 xcache.readonly_protection 参数，不同进程组的 PHP 将不会共享同一个 /tmp/xcache 路径
xcache.coredump_directory = ""	当 XCache crash 后，是否把数据保存到指定路径
xcache.disable_on_crash = Off	当 XCache 发生 crash 时，自动关闭 XCache 缓存

更多参数说明请看配置文件对应注释说明或参考官方文档：

http://xcache.lighttpd.net/wiki/XcacheIni 。

2.修改php.ini配置XCache

1）先在配置XCache参数前加个配置分界符，配置命令如下：

```
cd /application/php/lib
echo >>php.ini
echo ';


xcache config by oldboy 20150501---------------------' >>php.ini
tail -2 php.ini
```

执行过程如下：

```
[root@www lib]# cd /application/php/lib
[root@www lib]# echo >>php.ini
[root@www lib]# echo ';


xcache config by oldboy 20150501-------------' >>php.ini
[root@www lib]# tail -2 php.ini;


xcache config by oldboy 20150501----------------------
```

2）编辑xcache.ini，修改XCache的配置参数，调整的关键参数见表7-4。

表7-4　调整关键的参数

需要调整的关键参数		调整后的关键参数数值	
xcache.size	= 60M	xcache.size	= 256M
xcache.count	= 1	xcache.count	= 2
xcache.ttl	= 0	xcache.ttl	= 86400
xcache.gc_interval	= 0	xcache.gc_interval	= 3600
xcache.var_size	= 4M	xcache.var_size	= 64M

以上参数需要根据生产硬件的大小，以及业务数据的访问量来调整，

```
[root@www lib]# vim /home/oldboy/tools/xcache-3.2.0/xcache.ini
```

修改相应配置参数后保存退出。

3）将修改后的xcache.ini合并到php.ini结尾。命令如下：

```
[root@www lib]# cat /home/oldboy/tools/xcache-3.2.0/xcache.ini >>php.ini
```

修改后，整个xcache.ini的内容如下：

```
[root@www lib]# tail -85 php.ini|egrep -v "^;


|^$"
[xcache-common]
extension = xcache.so
[xcache.admin]
xcache.admin.enable_auth = On
xcache.admin.user = "mOo"
xcache.admin.pass = "md5 encrypted password"
[xcache]
xcache.shm_scheme =        "mmap"
xcache.size  =               256M
xcache.count =                 2
xcache.slots =                 8K
xcache.ttl   =                 86400
xcache.gc_interval =           3600
xcache.var_size  =             64M
xcache.var_count =             1
xcache.var_slots =             8K
xcache.var_ttl   =             0
xcache.var_maxttl   =          0
xcache.var_gc_interval =     300
xcache.var_namespace_mode =    0
xcache.var_namespace =         ""
xcache.readonly_protection = Off
xcache.mmap_path =    "/dev/zero"
xcache.coredump_directory =   ""
xcache.coredump_type =         0
xcache.disable_on_crash =    Off
```

```
xcache.experimental =        Off
xcache.cacher =               On
xcache.stat   =               On
xcache.optimizer =           Off
[xcache.coverager]
xcache.coverager =           Off
xcache.coverager_autostart =  On
xcache.coveragedump_directory = ""
```

由于这里是演示环境，因此部分参数未进行调整，读者可根据生产要求自行调整。

3.检查XCache加速情况配置

再次执行PHP的命令，查看缓存的生效情况，命令如下：

```
[root@www lib]# /application/php/bin/php -v
PHP Warning:


  Cannot load module 'XCache' because conflicting module 'eAccelerator' is already l
PHP 5.3.27 (


cli)


  (


built:


 Apr 22 2015 20:


23:


57)
```

```
Copyright  (

c)

 1997-2013 The PHP Group
Zend Engine v2.3.0,

 Copyright  (

c)

 1998-2013 Zend Technologies
    with eAccelerator v0.9.6.1,

 Copyright  (

c)

 2004-2010 eAccelerator,

 by eAccelerator
```

看到了吧，上面提示说XCache和eAccelerator冲突。在实际工作中，这两个软件可以任选一个。此时可以把eAccelerator的配置参数删除，保留XCache参数，命令如下：

```
[root@www lib]#cp php.ini php.ini-include-eacc-xcache
[root@www lib]# sed -n '1924,

1938p' php.ini
[eaccelerator]
extension=eaccelerator.so
```

```
eaccelerator.shm_size="64"
eaccelerator.cache_dir="/tmp/eaccelerator"
eaccelerator.enable="1"
eaccelerator.optimizer="1"
eaccelerator.check_mtime="1"
eaccelerator.debug="0"
eaccelerator.filter=""
eaccelerator.shm_max="0"
eaccelerator.shm_ttl="3600"
eaccelerator.shm_prune_period="3600"
eaccelerator.shm_only="0"
eaccelerator.compress="1"
eaccelerator.compress_level="9"
[root@www lib]# sed -i '1924,


1938d' php.ini     #<==使用


sed删除，请注意行号
```

要和配置文件相对应

```
[root@www lib]# /application/php/bin/php -v
PHP 5.3.27 （


cli）


  （


built：


 Apr 22 2015 20：


23：


57）
```

```
Copyright (
c)
 1997-2013 The PHP Group
Zend Engine v2.3.0,
 Copyright (
c)
 1998-2013 Zend Technologies
    with XCache v3.2.0,
 Copyright (
c)
 2005-2014,
 by mOo
    with XCache Cacher v3.2.0,
 Copyright (
c)
 2005-2014,
 by mOo
```

XCache和eAccelerator均使用系统的共享内存作为存储空间，因

此，有必要调整系统的共享内存大小参数。下面介绍对应的XCache和eAccelerator内核调优方法，命令如下：

```
[root@www lib]# tail /etc/sysctl.conf
kernel.msgmnb = 65536
# Controls the maximum size of a message,


 in bytes
kernel.msgmax = 65536
# Controls the maximum shared segment size,


 in bytes
kernel.shmmax = 68719476736
# Controls the maximum number of shared memory segments,


 in pages
kernel.shmall = 4294967296
```

现在重启PHP服务，然后通过phpinfo界面检查XCache插件结果，如图7-11所示。

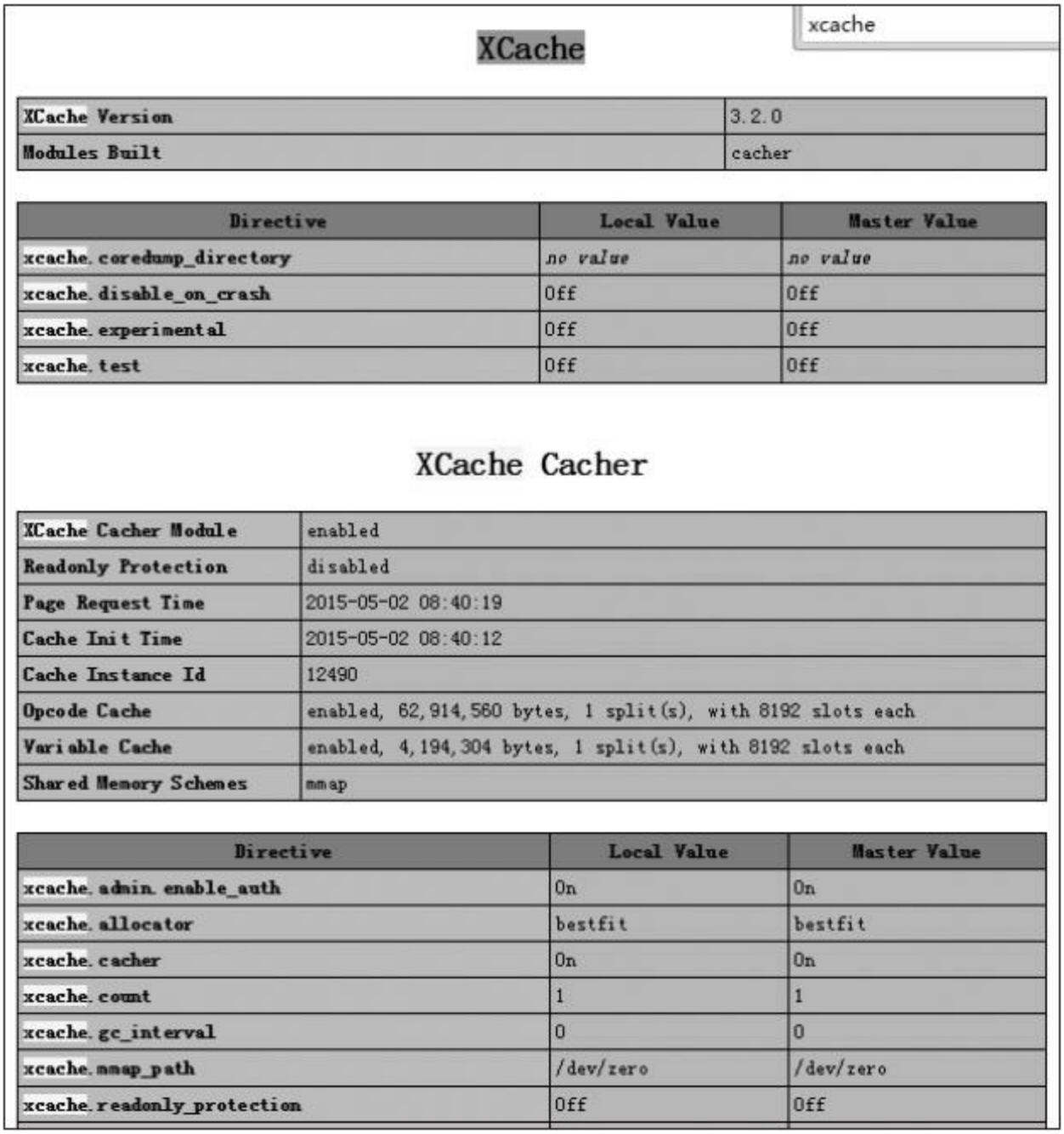
4.配置Web界面查看XCache缓存加速信息

使用Linux命令行md5sum命令，或者打开浏览器输入地址：http://xcache.lighttpd.net/demo/cacher/mkpassword.php ，通过输入字符串生成Xcache管理员的密码。这里使用如下md5sum命令生成密文密码。

```
[root@oldboyedu lib]# echo -n "123456"|md5sum
e10adc3949ba59abbe56e057f20f883e  -
```

使用echo打印123456，通过md5sum返回结果为e10adc3949ba59abbe56e057f20f883e，然后编辑php.ini文件，在结尾的XCache配置部分把这一串密文内容按照如下参数进行修改：

```
[xcache.admin]
xcache.admin.enable_auth = On
xcache.admin.user = "oldboy"
xcache.admin.pass = "e10adc3949ba59abbe56e057f20f883e" #<==对应密码为


123456
```



xcache

XCache

XCache Version	3.2.0
Modules Built	cacher

Directive	Local Value	Master Value
xcache.coredump_directory	*no value*	*no value*
xcache.disable_on_crash	Off	Off
xcache.experimental	Off	Off
xcache.test	Off	Off

XCache Cacher

XCache Cacher Module	enabled
Readonly Protection	disabled
Page Request Time	2015-05-02 08:40:19
Cache Init Time	2015-05-02 08:40:12
Cache Instance Id	12490
Opcode Cache	enabled, 62,914,560 bytes, 1 split(s), with 8192 slots each
Variable Cache	enabled, 4,194,304 bytes, 1 split(s), with 8192 slots each
Shared Memory Schemes	mmap

Directive	Local Value	Master Value
xcache.admin.enable_auth	On	On
xcache.allocator	bestfit	bestfit
xcache.cacher	On	On
xcache.count	1	1
xcache.gc_interval	0	0
xcache.mmap_path	/dev/zero	/dev/zero
xcache.readonly_protection	Off	Off

图7-11　PHP的扩展插件xcache生效

然后复制XCache软件下面的缓存加速管理PHP程序到站点目录下，命令如下：

```
[root@www lib]# cd /home/oldboy/tools/xcache-3.2.0
```

```
[root@www xcache-3.2.0]# cp -a htdocs/ /application/nginx/html/blog/xadmin
[root@www xcache-3.2.0]# chown -R nginx.nginx /application/nginx/html/blog/xadmin
[root@oldboyedu xcache-3.2.0]# pkill php-fpm
[root@oldboyedu xcache-3.2.0]# /application/php/sbin/php-fpm
```

访问http://blog.etiantian.org/xadmin/index.php，弹出验证框，需要输入密码（如图7-12所示）。

图7-12　XCache缓存Web管理登录界面

登录后的相关信息展示如图7-13所示。

图7-13　XCache缓存Web展示界面

有关缓存XCache的状态、命中等相关信息都可以通过这个XCache管理界面查看。

7.5.4　配置ZendOpcache插件加速

1.配置ZendOpcache参数

在php.ini的最后面加入下面几行：

```
[opcache]
zend_extension=/application/php5.3.27/lib/php/extensions/no-debug-non-zts-
20090626/opcache.so;


extension=opcache.so
opcache.memory_consumption=32
opcache.interned_strings_buffer=8
opcache.max_accelerated_files=1000
opcache.revalidate_freq=60
opcache.fast_shutdown=1
opcache.enable_cli=1
```

检查配置配置文件，查看配置结果，命令如下：

```
[root@www lib]# tail php.ini
[opcache]
zend_extension=/application/php5.3.27/lib/php/extensions/no-debug-non-zts-
20090626/opcache.so;


extension=opcache.so     #<==这种方法不能用了



opcache.memory_consumption=128
opcache.interned_strings_buffer=8
opcache.max_accelerated_files=4000
opcache.revalidate_freq=60
opcache.fast_shutdown=1
opcache.enable_cli=1
```

确认此结果是否和自己希望配置的一致，操作后进行检查是运维人员的一个优秀习惯。

2.检查ZendOpcache生效情况

下面使用PHP命令检查生效结果：

```
[root@www lib]# /application/php/bin/php -v
PHP 5.3.27 (

cli)

  (

built:

 Apr 22 2015 20:

23:

57)

Copyright (

c)

 1997-2013 The PHP Group
Zend Engine v2.3.0,

 Copyright (
```

```
c)


 1998-2013 Zend Technologies
    with XCache v3.2.0,


 Copyright  (


c)


 2005-2014,


 by mOo
    with Zend OPcache v7.0.5,


 Copyright  (


c)


 1999-2015,


 by Zend Technologies
    with XCache Cacher v3.2.0,


 Copyright  (


c)


 2005-2014,


 by mOo
```

可以看到ZendOpcache已经生效，并且貌似和XCache相处得比较融洽，不过工作中是否多选，还要慎重选择。

现在重启PHP服务，并通过phpinfo界面检查XCache插件结果，如图7-14和图7-15所示。

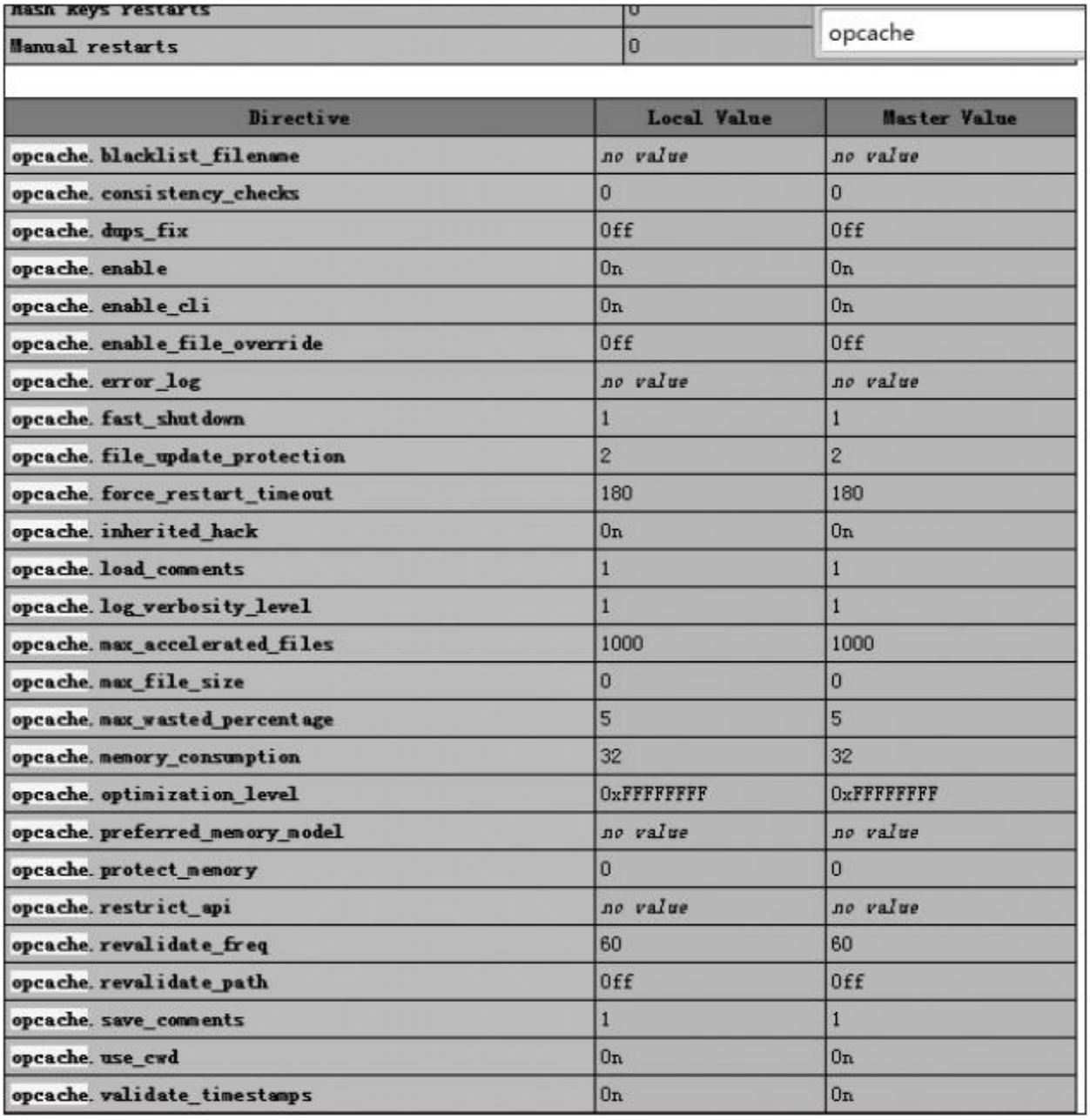
Zend OPcache

Opcode Caching	Up and Running
Optimization	Enabled
Startup	OK
Shared memory model	mmap
Cache hits	0
Cache misses	1
Used memory	116128
Free memory	33438304
Wasted memory	0
Cached scripts	1
Cached keys	1
Max keys	1979
OOM restarts	0
Hash keys restarts	0
Manual restarts	0

图7-14　PHP的扩展插件OPcache生效图1

3.ZendOpcache配置参数说明

表7-5中针对OPcache的部分重要参数进行了说明。



Hash keys restarts	0
Manual restarts	0

opcache

Directive	Local Value	Master Value
opcache.blacklist_filename	*no value*	*no value*
opcache.consistency_checks	0	0
opcache.dups_fix	Off	Off
opcache.enable	On	On
opcache.enable_cli	On	On
opcache.enable_file_override	Off	Off
opcache.error_log	*no value*	*no value*
opcache.fast_shutdown	1	1
opcache.file_update_protection	2	2
opcache.force_restart_timeout	180	180
opcache.inherited_hack	On	On
opcache.load_comments	1	1
opcache.log_verbosity_level	1	1
opcache.max_accelerated_files	1000	1000
opcache.max_file_size	0	0
opcache.max_wasted_percentage	5	5
opcache.memory_consumption	32	32
opcache.optimization_level	0xFFFFFFFF	0xFFFFFFFF
opcache.preferred_memory_model	*no value*	*no value*
opcache.protect_memory	0	0
opcache.restrict_api	*no value*	*no value*
opcache.revalidate_freq	60	60
opcache.revalidate_path	Off	Off
opcache.save_comments	1	1
opcache.use_cwd	On	On
opcache.validate_timestamps	On	On

图7-15　PHP的扩展插件OPcache生效图2

表7-5　OPcache重要参数说明

OPcache 参数	解释说明
opcache.memory_consumption=128	OPcache 共享内存空间大小，用于存放 precompiled PHP code，默认为 64，单位为 Mbytes
opcache.interned_strings_buffer=8	默认值为 4，interned strings 内存的数量，单位是 M
opcache.max_accelerated_files=4000	默认值为 2000，OPcache 散列表的 key 的最大数量
opcache.revalidate_freq=60	默认值为 2，检查文件时间戳的频率，用于共享内存分配的变化
opcache.fast_shutdown=1	默认值为 0，如果激活，一个快速的关闭队列将被用来加速代码
opcache.enable_cli=1	默认值为 0，激活 PHP CLI 的 OPcache，用于测试和调试

更多的OPcache参数可以查看安装目录下的README，本书的路径为/home/oldboy/tools/zendopcache-7.0.5/README。

说明：ZendOpcache是PHP官方的新一代的缓存加速软件，PHP 5.5以前可以通过ZendOpcache软件以插件扩展的方式安装，从PHP 5.5版本开始已经整合到PHP软件里，编译时只需指定一个参数即可，例如：--enable-opcache。

ZendOpcache可能是未来的首选，现在的稳定性还有待检验。在小规模环境下，PHP 5以上的版本可以使用。如果可以忍受其未知的问题也可以使用。

PHP 5.5及以上版本PHP安装编译的完整参数如下：

```
tar zxf php-5.5.13.tar.gz
cd php-5.5.13
./configure --prefix=/application/php --with-mysql --with-jpeg-dir --with-png-dir --
make && make install
```

在编译参数里指定--enable-opcache即可编译。

7.6　生产环境PHP扩展插件的安装建议

1.PHP的安装插件表格列表

常见的PHP扩展插件及其说明见表7-6。

表7-6　常见的PHP扩展插件列表

PHP EXT module	说　　明	备　　注
eaccelerator-0.9.5.2.tar.tar	适合 PHP5.3 以前的版本，PHP 缓存加速	可选 PHP 扩展插件
eaccelerator-0.9.6.tar.bz2	适合 PHP5.3 版本，PHP 缓存加速	可选 PHP 扩展插件
ImageMagick.tar.gz	常用图像处理程序，属功能应用	非 PHP 的扩展插件
imagick-2.3.0.tgz	需要先装图像处理程序，属功能应用	可选 PHP 扩展插件
memcache-2.2.7.tgz	memcache 客户端数据库缓存优化用	可选 PHP 扩展插件
PDO_MYSQL-1.0.2.tgz	PHP 数据库访问插件，属功能应用	可选 PHP 扩展插件
xcache-3.2.0.tar.bz2	支持 PHP5.1-5.6，PHP 缓存加速	可选 PHP 扩展插件
zendopcache-7.0.5.tgz	支持 PHP5.3-5.4，PHP 缓存加速	可选 PHP 扩展插件

2.生产环境插件的安装建议

1）对于功能性插件，如果业务产品不需要使用，可以暂时不考虑安装，例如PDO_MYSQL\memcache\imagick等。如果不清楚是否需要，最好还是装上，有备无患。

2）对于性能优化插件，eAccelerator、XCache、ZendOpcache、APC可以安装任一种，具体情况看实际业务需求，在选择时最好能搭建相关环境进行压力测试，然后根据实际测试结果来选择，用数据说话很

重要。

老男孩也是这么用的，一般的环境，上述4种优化插件都可以使用，效果都还可以。

3.PHP加速插件的测试对比

表7-7为相关PHP加速插件的测试结果对比参考（无opcache）。

下面是针对PHP加速器比较结果进行的总结。

·通过测试得出，eAccelerator在请求时间和内存占用综合方面是最好的。

·通过测试得出，使用加速器比无加速器在请求时间快了3倍左右。

·通过各个官方观察，XCache的更新是最快的，这也说明它是最有发展前景的。

表7-7　PHP加速插件的测试结果对比

	请求时间（秒）	单次请求时间（毫秒）	最大内存占用（MB）	最小内存占用（MB）
None	10.41	96.08	24	24
APC	30.45	32.84	21	21
eAccelerator	31.26	31.99	23	18
XCache	30.28	33.02	29	19

以上是总结结果，也许有人会疑惑到底用哪个加速器好呢？我只能

告诉你，首先，用一定比不用好，其次每个加速器还有一些可以调优的参数，所以要根据系统环境而定。此外，XCache和ZendOPcache这两款加速器的潜力还是很大的，可以多关注一下。

7.7 补充知识

7.7.1 phpize是什么

安装PHP扩展插件的时候，常常有这样一条命令：/application/php/bin/phpize，可能有人会问phpize有什么用？

事实上，phpize是用来扩展PHP扩展模块的，通过phpize可以建立PHP的外挂模块。比如想在原来编译好的PHP中加入Memcached等扩展模块，可以使用phpize工具。

PHP的官方说明地址为http://php.net/manual/en/install.pecl.phpize.php。

那么，要如何使用phpize呢？

编译PHP后，其bin目录下会有phpize这个脚本文件。在编译要添加的扩展模块之前，执行以下phpize就可以了。比如现在想在PHP中加入memcached扩展模块，那么要做的只是执行如下的Memcached客户端软件安装命令：

```
cd /home/oldboy/tools/
wget -q http:
```

```
//pecl.php.net/get/memcache-2.2.7.tgz
tar zxf memcache-2.2.7.tgz
cd memcache-2.2.7
/application/php/bin/phpize #<==在执行


configure前执行这个命令



./configure  --enable-memcache  --with-php-config=/application/php/bin/php-config
make
make install
cd ../
```

 注意：./configure后面可以指定的是php-config文件的路径。

这样编译就完成了，还需要做的是在php.ini文件中加入如下两行：

```
extension_dir = "/application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626
extension = memcache.so
```

提示：上述两行配置的作用是加载使得Memcached客户端配置生效。

 提示：extension_dir的路径就是memcache.so模块文件所在的路径。

7.7.2 PHP指定MySQL的编译参数带来的问题

如果在编译PHP时指定了--with-mysql=mysqlnd和--with-pdo-mysql=mysqlnd的参数，例如：

```
    ./configure\
--prefix=/application/php5.3.27\
--with-mysql=mysqlnd\
--with-mysqli=mysqlnd\
--with-pdo-mysql=mysqlnd\
    ...省略其他参数...
```

那么在生产中可能会遇到socket连接问题，解决方法是在php.ini里加入如下命令：

```
pdo_mysql.default_socket=/application/php5.3.27/tmp/mysql.sock
```

7.8　PHP缓存加速压力测试练习

分别安装ZendOpcache、eacc、XCache缓存加速插件，通过压测软件对比三者的缓存效率。

测试方法：

1）不装任何加速插件和分别安装某一个缓存加速软件。

2）可用压力测试软件webbench、loadruner。

3）用压力测试方法，通过数据看看到底哪个加速器好。

7.9 本章参考资料

·eAccelerator官方地址

http://eaccelerator.net

https://github.com/eaccelerator/eaccelerator/wiki

https://github.com/eaccelerator/eaccelerator/downloads

·XCache官方资料

http://xcache.lighttpd.net 。

http://xcache.lighttpd.net/wiki/Introduction

·ZendOpcache官方资料

http://pecl.php.net/package/ZendOpcache

第8章　企业级Nginx Web服务优化实战

8.1　Nginx基本安全优化

8.1.1　调整参数隐藏Nginx软件版本号信息

一般来说，软件的漏洞都和版本有关，这个很像汽车的缺陷，同一批次的要有问题就都有问题，别的批次可能就都是好的。因此，我们应尽量隐藏或消除Web服务对访问用户显示各类敏感信息（例如Web软件名称及版本号等信息），这样恶意的用户就很难猜到他攻击的服务器所用的是否有特定漏洞的软件，或者是否有对应漏洞的某一特定版本，从而加强Web服务的安全性。这在武侠小说里，就相当于隐身术，你隐身了，对手就很难打着你了。

想要隐身，首先要了解所使用软件的版本号，对于Linux客户端，可通过命令行查看Nginx版本号，最简单的方法就是在Linux客户端系统命令行执行如下curl命令：

```
[root@oldboy ~]# curl -I 10.0.0.7
HTTP/1.1 200 OK
Server:


 nginx/1.6.3    #<==这里很清晰地暴露了
```

Web版本号（

1.6.3）及软件名称（

nginx）

Date:

Thu,

09 Oct 2014 01:

58:

51 GMT
Content-Type:

text/html
Content-Length:

18
Last-Modified:

Thu,

25 Sep 2014 20:

01:

01 GMT
Connection:

```
 keep-alive
ETag:


 "5424747d-12"
Accept-Ranges:


 bytes
```

在Windows客户端上，通过浏览器访问Web服务时，若找不到页面，默认报错的信息如图8-1所示。

图8-1　找不到对应地址的错误页面显示

以上虽然是不同的客户端，但是都获得了Nginx软件名称，而且查到了Nginx的版本号，这就使得Nginx Web服务的安全存在一定的风险，因此，应隐藏掉这些敏感信息或用一个其他的名字将其替代。例如，下面是百度搜索引擎网站Web软件的更名做法：

```
[root@oldboy ~]# curl -I baidu.com
HTTP/1.1 200 OK
Date:
```

Tue，

21 Oct 2014 03：

31：

01 GMT
Server：

Apache　　#<==将

Web服务软件更名为了

Apache，并且版本号也去掉了

[root@oldboy ~]# curl -I -s www.baidu.com|grep Server
Server：

BWS/1.1　　#<==将

Web服务软件更名为了

BWS，并且版本号改为

1.1（闭源的软件名称

和版本就无所谓了）

门户网站尚且如此，我们也学着隐藏或改掉应用服务软件名和版本

号吧！

事实上，还可以通过配置文件加参数来隐藏Nginx版本号。

编辑nginx.conf配置文件增加参数，实现隐藏Nginx版本号的方式如下。

在Nginx配置文件nginx.conf中的http标签段内加入“server_tokens off；”参数，如下：

```
http
{
    ......
    server_tokens off;


    ......
}
```

此参数放置在http标签内，作用是控制http response header内的Web服务版本信息的显示，以及错误信息中Web服务版本信息的显示。

server_tokens参数的官方说明如下：

```
syntax:

     server_tokens on | off;

 #<==此行为参数语法，
```

on为开启状态，

off为关闭状态

default：

server_tokens on；

#<==此行意思是不配置该参数，软件默认情况的结果

context：

http，

server，

location #<==此行为

server_tokens参数可以放置的位置

参数作用：激活或禁止

Nginx的版本信息显示在报错信息和

Server的响应首部位置中

Enables or disables emitting of nginx version in error messages and in the"Server" r

---

官方资料地址：http://nginx.org/en/docs/http/ngx_http_core_module.html 。

配置完毕后保存，重新加载配置文件，再次通过curl查看，结果如下：

```
[root@oldboy conf]# /application/nginx/sbin/nginx -s reload
[root@oldboy conf]# curl -I 10.0.0.7
HTTP/1.1 200 OK
Server:


 nginx                      #<==版本号已消失



Date:


 Thu,


 09 Oct 2014 02:


03:


32 GMT
Content-Type:


 text/html
Content-Length:


 18
Last-Modified:


 Thu,
```

```
 25 Sep 2014 20:


01:


01 GMT
Connection:


 keep-alive
ETag:


 "5424747d-12"
Accept-Ranges:


 bytes
```

此时，浏览器的报错提示中没有了版本号，如图8-2所示。修改成功。

图8-2　无版本号的错误页面显示

8.1.2　更改源码隐藏Nginx软件名及版本号

隐藏了Nginx版本号后，更进一步，可以通过一些手段把Web服务软件的名称也隐藏起来，或者更改为其他Web服务软件名以迷惑黑客。但软件名字的隐藏修改，一般情况下不会有配置参数和入口，Nginx也不例外，这可能是由于商业及品牌展示等原因，软件提供商不希望使用者把软件名字隐藏起来。因此，此处需要更改Nginx源代码，具体的解决方法如下。

第一步是依次修改3个Nginx源码文件。

修改的第一个文件为nginx-1.6.3/src/core/nginx.h，如下：

```
[root@oldboy core]# sed -n '13,


17p' nginx.h
#define NGINX_VERSION      "1.6.3" #<==修改为想要显示的版本号，如


2.2.23
#define NGINX_VER          "nginx/" NGINX_VERSION
                          #<==将


nginx修改为想要修改的软件名称，如


OWS
#define NGINX_VAR          "NGINX"     #<==将


nginx修改为想要修改的软件名称，
```

如

```
OWS（

Oldboy Web Server）

#define NGX_OLDPID_EXT     ".oldbin"
```

修改后的结果如下：

```
[root@oldboy core]# sed -n '13,

17p' nginx.h
#define NGINX_VERSION      "2.2.23"  #<==已修改为想要显示的版本号

2.2.23
#define NGINX_VER          "OWS/" NGINX_VERSION #<==已修改为想要显示的名字

OWS
#define NGINX_VAR          "OWS"#<==已修改为想要显示的名字

OWS
#define NGX_OLDPID_EXT     ".oldbin"
```

修改的第二个文件是nginx-1.6.3/src/http/ngx_http_header_filter_module.c的第49行，需要修改的字符串如下：

```
[root@oldboy http]# grep -n 'Server:
```

```
 nginx'  ngx_http_header_filter_module.c
49:


static char ngx_http_server_string[] = "Server:


 nginx " CRLF;


 #<==修改本行结尾的


nginx。
```

通过sed替换修改，当然，也可以通过编辑器进行如下修改：

```
[root@oldboy http]# sed -i 's#Server:


 nginx#Server:


 OWS#g' ngx_http_header_filter_module.c
#<==把


Server:


 nginx替换为


Server:


 OWS，注意
```

```
Nginx字符串很多，所以加上

Server:

 nginx一起替换。当然也可以指定行替换，这样就可以直接替换

nginx字符串了。

sed -i '49 s#nginx#OWS#g' ngx_http_header_filter_module.c
```

修改后的结果如下：

```
[root@oldboy http]# grep -n 'Server:

 OWS'  ngx_http_header_filter_module.c
49:

static char ngx_http_server_string[] = "Server:

 OWS" CRLF;

 #<==变成

OWS了。
```

修改的第三个文件是nginx-1.6.3/src/http/ngx_http_special_response.c，对外页面报错时，它会控制是否展示敏感信息。这里输出修改前的信息ngx_http_special_response.c中

的第21~30行，如下：

```
[root@oldboy http]# sed -n '21,

30p' ngx_http_special_response.c
static u_char ngx_http_error_full_tail[] =
"<hr><center>" NGINX_VER "</center>" CRLF#<==此行需要修改

"</body>" CRLF
"</html>" CRLF;

static u_char ngx_http_error_tail[] =
"<hr><center>Nginx</center>" CRLF#<==此行需要修改

"</body>" CRLF
```

将上述内容中的“"<hr> "NGINX_VER" "CRLF”修改为“"<hr> "NGINX_VER"（http://oldboy.blog.51cto.com） "CRLF”，然后将“"<hr> Nginx "CRLF”修改为“"<hr> OWS "CRLF”。

修改后的结果如下：

```
[root@oldboy http]# sed -n '21,

30p' ngx_http_special_response.c
static u_char ngx_http_error_full_tail[] =
"<hr><center>" NGINX_VER "  (

http:
```

```
//oldboy.blog.51cto.com)


</center>" CRLF#<==此行是定义对外展示的内容



"</body>" CRLF
"</html>" CRLF;



static u_char ngx_http_error_tail[] =
"<hr><center>OWS</center>" CRLF           #<==此行将对外展示的


Nginx名字更改为


OWS
"</body>" CRLF
```

第二步是修改后编译软件，使其生效。

修改后再编译安装软件，如果是已安装好的服务，需要重新编译Nginx，配好配置，启动服务。

再次使浏览器出现404错误，然后看访问结果，如图8-3所示。

图8-3　去掉软件名字和版本的404页面图

如图8-3所示，Nginx的软件和版本名都被改掉了，并且加上了我们自己的博客地址。

再看看Linux curl命令响应头部信息，如下：

```
[root@oldboy http]# curl -I bbs.etiantian.org/oldboy/
HTTP/1.1 404 Not Found
Server:


 OWS/2.2.23                #<==也更改了


Date:


 Thu,


 12 Feb 2015 07:


00:


52 GMT
Content-Type:
```

```
 text/html
Content-Length:

 198
Connection:

 keep-alive
```

说明：

·提升网站安全，要从最简单、最短板、最低点入手解决问题，门打开着，即使给窗户安装再结实的护栏也没有意义。

·向有经验的人及优秀的网站公司学习。

·学会看官方文档，根据一手资料去分析。

·命令输出结果中含有需要过滤或要保留的内容时，命令自身可能有参数可直接实现。

掌握技术思想比解决问题本身更重要，见老男孩的博客：http://oldboy.blog.51cto.com/2561410/1196298 。

8.1.3 更改Nginx服务的默认用户

为了让Web服务更安全，要尽可能地改掉软件默认的所有配置，包括端口、用户等。例如：本书前文的Linux基础安全中，就更改过SSH服务的22端口，并且禁止了root SSH连接等。

下面就来更改Nginx服务的默认用户。

首先，查看Nginx服务对应的默认用户。一般情况下，Nginx服务启动后，默认使用的用户是nobody，查看默认的配置文件，如下：

```
[root@oldboy conf]# grep '#user' nginx.conf.default
#user  nobody;
```

为了防止黑客猜到这个Web服务的用户，我们需要更改成特殊的用户名，例如nginx或特殊点的inca，但是这个用户必须是系统里事先存在的，下面以nginx用户为例进行说明。

（1）为Nginx服务建立新用户

为Nginx服务建立新用户的操作过程如下：

```
useradd nginx -s /sbin/nologin -M
#<==不需要有系统登录权限，应当禁止其登录能力，相当于
```

```
apache里的用户


id nginx #<==检查用户
```

（2）配置Nginx服务，让其使用刚建立的nginx用户

更改Nginx服务默认使用的用户，方法有两种：

第一种为直接更改配置文件参数，将默认的#user nobody；改为如下内容：

```
user  nginx nginx;
```

如果注释或不设置上述参数，默认为nobody用户，不推荐使用nobody用户名，最好采用一个普通用户，此处用大家习惯的、前文建立好的nginx用户。

第二种方法为直接在编译Nginx软件时指定编译的用户和组，命令如下：

```
./configure  --user=nginx --group=nginx --prefix=/application/nginx1.6.3 --with-http
```

提示：前文在编译Nginx服务时，就是这样带着参数的，因此无论配置文件中是否加参数，默认都是nginx用户。

（3）检查更改用户的效果

重新加载配置后，检查Nginx服务进程的对应用户，如下：

```
[root@oldboy conf]# ps -ef|grep nginx|grep -v grep
root      1428     1  0 09:


56         00:


00:


00 nginx:


 master process /
application/nginx/sbin/nginx
nginx     1610  1428  0 10:


03         00:


00:


00 nginx:


 worker process
nginx     1611  1428  0 10:


03         00:
```

```
00:

00 nginx:

 worker process
```

通过查看上述更改后的Nginx进程，可以看到worker processes进程对应的用户都变成了nginx。所以，我们有理由得出结论，上述的两种方法都可设置Nginx的worker进程运行的用户。当然，Nginx的主进程还是以root身份运行的，本章后面也会有更改root主进程服务用户的深度安全优化与架构技巧讲解。

8.2 根据参数优化Nginx服务性能

8.2.1 优化Nginx服务的worker进程个数

在高并发、高访问量的Web服务场景，需要事先启动好更多的Nginx进程，以保证快速响应并处理大量并发用户的请求。

这类似于开饭店，在营业前，需要事先招聘一定数量的服务员准备接待顾客，但这里就有一个问题，如果饭店对客流没有正确预估，就会导致一些问题发生，例如：服务员人数招聘多了，但是客流很少，那么服务员就可能很闲，没事干，饭店的成本也高了；如果客流很大，而服务员人数少了，可能就接待不过来顾客，导致顾客吃饭体验差。因此，饭店要根据客户的流量及并发量来调整接待的服务人员数量，然后根据监测顾客量变化及时调整到最佳的配置。

Nginx服务就相当于饭店，网站用户就相当于顾客，Nginx的进程就相当于服务员，下面就来优化Nginx进程的个数。

1.优化Nginx进程对应的配置

优化Nginx进程对应Nginx服务的配置参数如下：

```
worker_processes 1;
```

```
#<==指定了

Nginx要开启的进程数，结尾的数字就是进程的个数
```

上述参数调整的是Nginx服务的worker进程数，Nginx有Master进程和worker进程之分，Master为管理进程，真正接待“顾客”的是worker进程。

2.优化Nginx进程个数的策略

前面已经讲解过，worker_processes参数大小的设置最好和网站的用户数量相关联，可如果是新配置，不知道网站的用户数量该怎么办呢？

搭建服务器时，worker进程数最开始的设置可以等于CPU的核数，且worker进程数要多一些，这样起始提供服务时就不会出现因为访问量快速增加而临时启动新进程提供服务的问题，缩短了系统的瞬时开销和提供服务的时间，提升了服务用户的速度。高流量高并发场合也可以考虑将进程数提高至CPU核数*2，具体情况要根据实际的业务来选择，因为这个参数除了要和CPU核数匹配外，也和硬盘存储的数据及系统的负载有关，设置为CPU的核数是一个好的起始配置，这也是官方的建议。

3.查看Web服务器CPU硬件资源信息

下面介绍查看Linux服务器CPU总核数的方法。

通过/proc/cpuinfo可查看CPU个数及总核数。查看CPU总核数的示例如下：

```
[oldboy@oldboy ~]$ grep processor /proc/cpuinfo|wc -l
4 #<==表示为


1颗


CPU四核



[root@oldboy ~]# grep -c processor /proc/cpuinfo
4 #<==表示为


1颗


CPU四核
```

查看CPU总颗数的示例如下：

```
[oldboy@oldboy ~]$ grep 'physical id' /proc/cpuinfo|sort|uniq|wc -l
1 #<==对


physical id去重计数，表示


1颗


CPU
```

通过执行top命令，然后按数字1，即可显示所有的CPU核数，如下：

```
[root@oldboy-server ~]# top 按

1显示多核

cpu
top - 11:

31:

10 up 608 days,

 16:

04,

  2 users,

  load average:

 0.00,

 0.00,

 0.00
Tasks:

 121 total,
```

```
   1 running,

 114 sleeping,

   6 stopped,

   0 zombie
Cpu0  :

  0.2%us,

  0.1%sy,

  0.0%ni,

 99.2%id,

  0.5%wa,

  0.0%hi,

  0.0%si,

  0.0%st
Cpu1  :

  0.1%us,

  0.0%sy,
```

0.0%ni,

99.5%id,

0.3%wa,

0.0%hi,

0.0%si,

0.0%st
Cpu2 :

0.2%us,

0.1%sy,

0.0%ni,

99.4%id,

0.3%wa,

0.0%hi,

0.0%si,

0.0%st
Cpu3 :

0.2%us,

```
  0.1%sy,

  0.0%ni,

 99.4%id,

  0.3%wa,

  0.0%hi,

  0.0%si,

  0.0%st
Mem:

   8173172k total,

  8126340k used,

    46832k free,

   419508k buffers
Swap:

  4192956k total,

      156k used,

  4192800k free,

  6681084k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
```

```
    1 root      15   0 10368  680  572 S  0.0  0.0   0:

01.94 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:

02.48 migration/0
#<==这是单

CPU四核的信息

例如：

CPU核数为

4，就配置

worker_processes 4
```

4.实践修改Nginx配置

假设服务器的CPU颗数为1颗，核数为4核，则初始的配置可通过查看默认的nginx.conf里的worker_processes数来了解，命令如下：

```
[root@oldboy conf]# grep worker_processes  nginx.conf
worker_processes  1;
```

这里修改参数值为CPU的总核数4，然后重新加载Nginx服务。

修改配置的方法如下：

```
[root@oldboy conf]# sed  -i 's#worker_processes  1#worker_processes  4#g' nginx.con1
[root@oldboy conf]# grep worker_processes  nginx.conf
worker_processes  4;
```

提示：可以通过

vi修改

优雅重启Nginx，使修改生效，如下：

```
[root@oldboy ~]# /application/nginx/sbin/nginx -t
nginx:

 the configuration file /application/nginx1.6.3/conf/nginx.conf syntax is ok
nginx:

 configuration file /application/nginx1.6.3/conf/nginx.conf test is successful
[root@oldboy ~]# /application/nginx/sbin/nginx -s reload
```

现在检查修改后的worker进程数量，如下：

```
[root@oldboy ~]# ps -ef|grep nginx|grep -v grep
root      1428     1  0 09:

56         00:

00:

00 nginx:

 master process /application/nginx/sbin/nginx
```

```
nginx     1832  1428  0 10:

24         00:

00:

00 nginx:

 worker process
nginx     1833  1428  0 10:

24         00:

00:

00 nginx:

 worker process
nginx     1834  1428  0 10:

24         00:

00:

00 nginx:

 worker process
nginx     1835  1428  0 10:

24         00:

00:
```

```
00 nginx:

 worker process
```

从“worker_processes 4”可知，worker的进程数为4个。Nginx Master主进程不包含在这个参数内，Nginx Master的主进程为管理进程，负责调度和管理worker进程。

有关worker_processes参数的官方说明如下：

```
syntax:

      worker_processes number;

 #<==此行为参数语法，

number为数量

default:

      worker_processes 1;

      #<==此行意思是不配置该参数，软件默认情况数量为

1
context:

      main                        #<==此行为
```

worker_processes参数可以放置的位置

worker_processes为定义

worker进程数的数量，建议设置为

CPU的核数或

CPU核数

*2，具体情况要根据实际的业务来选择，因为这个参数，除了要和

CPU核数匹配外，和硬盘存储的数据以系统的负载也有关，设置为

CPU的个数或核数是一个好的起始配置。

From ：

http:

//nginx.org/en/docs/ngx_core_module.html

8.2.2 优化绑定不同的Nginx进程到不同的CPU上

默认情况下，Nginx的多个进程有可能跑在某一个CPU或CPU的某一核上，导致Nginx进程使用硬件的资源不均，本节的优化是尽可能地分配不同的Nginx进程给不同的CPU处理，达到充分有效利用硬件的多CPU多核资源的目的。

在优化不同的Nginx进程对应不同的CPU配置时，四核CPU服务器的参数配置参考如下：

```
worker_processes    4;

worker_cpu_affinity 0001 0010 0100 1000;

#<== worker_cpu_affinity就是配置

Nginx进程与

CPU亲和力的参数，即把不同的进程分给不同的

CPU处理。这里

0001 0010 0100 1000是掩码，分别代表第

1、
```

2、

3、

4核

CPU，由于

worker_processes进程数为

4，因此，上述配置会把每个进程分配一核

CPU处理，默认情况下进程不会绑定任何

CPU，参数位置为

main段。

八核CPU服务器的参数配置参考如下：

```
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000
```

```
worker_cpu_affinity 0001 0010 0100 1000 0001 0010 0100 1000;
```

---

worker_cpu_affinity参数的官方说明如下：

```
syntax:

      worker_cpu_affinity cpumask ...;

     #<==此行为

cpu亲和力参数语法，

cpumask为

cpu掩码

default:

      —

                              #<==默认不设置

context:

      main                          #<==此行为

worker_cpu_
affinity参数可以放置的位置
```

worker_cpu_affinity的作用是绑定不同的worker进程数到一组CPU上。通过设置bitmask控制进程允许使用的CPU，默认worker进程不会绑

定到任何CPU。

下面是绑定的示例配置。

```
worker_processes    4;

worker_cpu_affinity 0001 0010 0100 1000;

binds each worker process to a separate CPU,

 while
worker_processes    2;

worker_cpu_affinity 0101 1010;

binds the first worker process to CPU0/CPU2,

 and the second worker process to CPU1/CPU3. The second example is suitable for hype
From :

 http:

//nginx.org/en/docs/ngx_core_module.html by oldboy
```

下面是压力测试配置结果。配置前的压力测试结果为：

```
top - 16:

30:
```

17 up 41 min,

2 users,

load average:

2.30,

1.50,

1.35
Tasks:

104 total,

4 running,

100 sleeping,

0 stopped,

0 zombie
Cpu0 :

6.7%us,

9.3%sy,

0.0%ni,

4.3%id,

0.0%wa,

5.0%hi,

74.7%si,

0.0%st
Cpu1 :

18.3%us,

26.6%sy,

0.0%ni,

48.8%id,

0.0%wa,

0.0%hi,

6.3%si,

0.0%st
Cpu2 :

15.9%us,

24.9%sy,

0.0%ni,

54.8%id,

0.0%wa,

0.0%hi,

4.3%si,

0.0%st
Cpu3 :

16.0%us,

27.0%sy,

0.0%ni,

53.3%id,

0.0%wa,

0.0%hi,

3.7%si,

0.0%st
top - 16:

30:

38 up 41 min,

2 users,

load average:

2.65,

1.63,

1.39
Tasks:

104 total,

3 running,

101 sleeping,

0 stopped,

0 zombie
Cpu0 :

7.7%us,

11.3%sy,

0.0%ni,

4.7%id,

0.0%wa,

5.0%hi,

71.3%si,

0.0%st
Cpu1 :

17.3%us,

29.2%sy,

0.0%ni,

47.8%id,

0.3%wa,

0.0%hi,

5.3%si,

0.0%st
Cpu2 :

17.7%us,

24.0%sy,

0.0%ni,

53.7%id,

0.0%wa,

0.0%hi,

4.7%si,

0.0%st

Cpu3 :

17.7%us,

27.7%sy,

0.0%ni,

51.3%id,

0.3%wa,

0.0%hi,

3.0%si,

0.0%st

top - 16:

30:

```
57 up 42 min,
  2 users,
  load average:
 2.48,
 1.65,
 1.41
Tasks:
 104 total,
   5 running,
  99 sleeping,
   0 stopped,
   0 zombie
Cpu0  :
  6.0%us,
 11.4%sy,
  0.0%ni,
  4.7%id,
```

0.0%wa,

5.0%hi,

72.9%si,

0.0%st
Cpu1 :

17.9%us,

26.9%sy,

0.0%ni,

50.5%id,

0.0%wa,

0.0%hi,

4.7%si,

0.0%st
Cpu2 :

16.9%us,

26.8%sy,

0.0%ni,

51.7%id,

0.0%wa,

0.0%hi,

4.6%si,

0.0%st
Cpu3 :

16.6%us,

27.6%sy,

0.0%ni,

51.8%id,

0.0%wa,

0.0%hi,

4.0%si,

0.0%st
top - 16:

31:

```
22 up 42 min,
  2 users,
  load average:
 2.65,
 1.76,
 1.45
Tasks:
 104 total,
   4 running,
 100 sleeping,
   0 stopped,
   0 zombie
Cpu0  :
  7.7%us,
 12.7%sy,
  0.0%ni,
  5.3%id,
```

0.0%wa,

5.0%hi,

69.3%si,

0.0%st
Cpu1 :

19.4%us,

28.4%sy,

0.0%ni,

47.2%id,

0.0%wa,

0.0%hi,

5.0%si,

0.0%st
Cpu2 :

17.0%us,

26.3%sy,

0.0%ni,

```
 52.3%id,
  0.0%wa,
  0.0%hi,
  4.3%si,
  0.0%st
Cpu3  :
 16.9%us,
 25.6%sy,
  0.0%ni,
 54.2%id,
  0.0%wa,
  0.0%hi,
  3.3%si,
  0.0%st
```

通过观察，我们发现配置前不同CPU的使用率相对平均。

配置worker_cpu_affinity，如下：

```
[root@www ~]# grep worker_cpu nginx.conf
worker_cpu_affinity 0001 0010 0100 1000;
```

压力测试的命令及结果如下：

```
[root@www ~]#webbench -c 20000 -t 180 http:

//10.0.0.190/
Webbench - Simple Web Benchmark 1.5
Copyright (

c)

 Radim Kolar 1997-2004,

 GPL Open Source Software.
Benchmarking:

 GET http:

//10.0.0.190/
20000 clients,

 running 180 sec.
Speed=1521283 pages/min,

 3588193 bytes/sec.
Requests:

 3661785 susceed,

 902066 failed.
```

查看CPU调度结果如下：

```
top - 16:
24:
58 up 36 min,
  2 users,
  load average:
 1.65,
 1.58,
 1.34
Tasks:
 104 total,
   4 running,
 100 sleeping,
   0 stopped,
   0 zombie
Cpu0  :
  4.7%us,
  8.0%sy,
  0.0%ni,
  7.3%id,
  0.0%wa,
```

5.3%hi,

74.8%si,

0.0%st

Cpu1 :

16.9%us,

32.2%sy,

0.0%ni,

43.2%id,

0.0%wa,

0.0%hi,

7.6%si,

0.0%st

Cpu2 :

16.6%us,

26.2%sy,

0.0%ni,

52.2%id,

0.0%wa,

0.0%hi,

5.0%si,

0.0%st

Cpu3 :

16.9%us,

28.2%sy,

0.0%ni,

49.8%id,

0.0%wa,

0.0%hi,

5.0%si,

0.0%st
top - 16:

25:

36 up 36 min,

2 users,

load average:

1.83,

1.64,

1.37
Tasks:

104 total,

2 running,

102 sleeping,

0 stopped,

0 zombie
Cpu0 :

5.0%us,

8.1%sy,

0.0%ni,

9.7%id,

0.0%wa,

5.0%hi,

72.1%si,

0.0%st

Cpu1 :

17.6%us,

31.2%sy,

0.0%ni,

43.9%id,

0.0%wa,

0.0%hi,

7.3%si,

0.0%st

Cpu2 :

16.0%us,

26.7%sy,

0.0%ni,

52.7%id,

0.0%wa,

0.0%hi,

4.7%si,

0.0%st
Cpu3 :

17.9%us,

28.6%sy,

0.0%ni,

48.8%id,

0.0%wa,

0.0%hi,

4.7%si,

0.0%st
top - 16:

26:

09 up 37 min,

2 users,

load average:

2.01,

1.71,

1.40
Tasks:

104 total,

4 running,

100 sleeping,

0 stopped,

```
   0 zombie
Cpu0  :

  6.3%us,

  9.0%sy,

  0.0%ni,

 10.6%id,

  0.0%wa,

  4.7%hi,

 69.4%si,

  0.0%st
Cpu1  :

 19.3%us,

 32.6%sy,

  0.0%ni,

 41.9%id,

  0.0%wa,

  0.0%hi,

  6.3%si,

  0.0%st
Cpu2  :

 17.9%us,

 26.9%sy,

  0.0%ni,
```

51.8%id,

0.0%wa,

0.0%hi,

3.3%si,

0.0%st
Cpu3 :

19.7%us,

30.0%sy,

0.0%ni,

46.3%id,

0.0%wa,

0.0%hi,

4.0%si,

0.0%st
top - 16:

26:

22 up 37 min,

2 users,

load average:

2.21,

1.77,

1.42
Tasks:

```
 104 total,
   4 running,
 100 sleeping,
   0 stopped,
   0 zombie
Cpu0  :
  5.3%us,
  8.3%sy,
  0.0%ni,
 10.3%id,
  0.0%wa,
  5.0%hi,
 71.0%si,
  0.0%st
Cpu1  :
 19.3%us,
 29.2%sy,
  0.0%ni,
 45.8%id,
  0.0%wa,
  0.0%hi,
  5.6%si,
  0.0%st
```

```
Cpu2  :
 15.9%us,
 24.3%sy,
  0.0%ni,
 55.8%id,
  0.0%wa,
  0.0%hi,
  4.0%si,
  0.0%st
Cpu3  :
 16.7%us,
 27.0%sy,
  0.0%ni,
 52.7%id,
  0.0%wa,
  0.0%hi,
  3.7%si,
  0.0%st
```

通过观察，我们发现配置后不同CPU的使用率相对平均，和测试前变化不大。因此可知，默认就是比较平均

另外（taskset-retrieve or set a process’s CPU affinity）命令本身也有分配CPU的功能，这里留给大家测试（例

```
    -c,
```

--cpu-list
      specify a numerical list of processors instead of a bitmask.
      The list may contain multiple items,

separated by comma,

and
      ranges. For example,

0,

5,

7,

9-11.

8.2.3 Nginx事件处理模型优化

Nginx的连接处理机制在不同的操作系统会采用不同的I/O模型，在Linux下，Nginx使用epoll的I/O多路复用模型，在Freebsd中使用kqueue的I/O多路复用模型，在Solaris中使用/dev/poll方式的I/O多路复用模型，在Windows中使用的是icop，等等。

要根据系统类型选择不同的事件处理模型，可供使用的选择有“use[kqueue|rtsig|epoll|/dev/poll|select|poll]；”。本书使用的是CentOS 6.6 Linux，因此将Nginx的事件处理模型调整为epoll模型。

具体的配置参数如下：

```
events
#<==events指令是设定


Nginx的工作模式及连接数上限



{
  use epoll;


#<==use是一个事件模块指令，用来指定


Nginx的工作模式。


Nginx支持的工作模式有
```

select、

poll、

kqueue、

epoll、

rtsig和

/dev/poll。其中

select和

poll都是标准的工作模式，

kqueue和

epoll是高效的工作模式，不同的是

epoll用在

Linux平台上，而

kqueue用在

BSD系统中。对于

Linux系统

```
Linux 2.6+的内核，推荐选择

epoll工作模式，这是高性能高并发的设置

}
```

根据Nginx的官方文档建议，也可以不指定事件处理模型，Nginx会自动选择最佳的事件处理模型服务。

events区块及use事件处理参数的官方说明如下：

```
syntax:

      events { ... }          #<==语法配置

default:

      —

                  #<==缺省没有设置

context:

      main                #<==events标签的放置位置，放在

main段
```

events区块是一个用来设置连接进程的区块，例如：设置Nginx的网络I/O模型，以及连接数等。

use事件的处理参数说明如下：

```
syntax:

    use method:

        #<==网络模型配置,

method选择模型之一

default:

    —

            #<==缺省没有设置

context:

    events              #<==网络模型配置放置于

events区块内
```

对于使用连接进程的方法，通常不需要进行任何设置，Nginx会自

动选择最有效的方法。

 提示：事件处理模型参数需要在events区块配置。

8.2.4　调整Nginx单个进程允许的客户端最大连接数

接下来，调整Nginx单个进程允许的客户端最大连接数，这个控制连接数的参数为worker_connections。

worker_connections的值要根据具体服务器性能和程序的内存使用量来指定（一个进程启动使用的内存根据程序确定），如下：

```
events  #<==events指令是设定

Nginx的工作模式及连接数上限


{
worker_connections 20480;


#<==worker_connections也是个事件模块指令，用于定义

Nginx每个进程的最大连接数，默认是

1024。最大客户端连接数由

worker_processes和

worker_connections决定，即
```

Max_client= worker_processes*worker_connections。

进程的最大连接数受

Linux系统进程的最大打开文件数限制，在执行操作系统命令“

ulimit -HSn 65535”或配置相应文件后，

worker_connections的设置才能生效

下面是worker_connections的官方说明。

参数语法：worker_connections number

默认配置：worker_connections 512

放置位置：events

说明：worker_connections用来设置一个worker process支持的最大并发连接数，这个连接数包括了所有连接，例如：代理服务器的连接、客户端的连接等，实际的并发连接数除了受worker_connections参数控制外，还和最大打开文件数worker_rlimit_nofile有关（见下文），Nginx总并发连接=worker数量*worker_connections。

参考资料：http://nginx.org/en/docs/ngx_core_module.html 。

8.2.5 配置Nginx worker进程最大打开文件数

接下来，调整配置Nginx worker进程的最大打开文件数，这个控制连接数的参数为worker_rlimit_nofile。该参数的实际配置如下：

```
worker_rlimit_nofile 65535;

#<==最大打开文件数，可设置为系统优化后的

ulimit -HSn的结果，在第

3章中，调整系统文件描述符和这个问题有相同之处
```

下面是worker_rlimit_nofile number的官方说明。

参数语法：worker_rlimit_nofile number

默认配置：无

放置位置：主标签段

说明：此参数的作用是改变worker processes能打开的最大文件数。

参考资料：http://nginx.org/en/docs/ngx_core_module.html

8.2.6 优化服务器域名的散列表大小

先要将确切名字和通配符名字存储在散列表中。散列表和监听端口关联，每个端口都最多关联到三张表：确切名字的散列表、以星号起始的通配符名字的散列表和以星号结束的通配符名字的散列表。散列表的尺寸在配置阶段进行了优化，可以以最小的CPU缓存命中失败来找到名字。Nginx首先会搜索确切名字的散列表，如果没有找到，则搜索以星号起始的通配符名字的散列表，如果还是没有找到，继续搜索以星号结束的通配符名字的散列表。因为名字是按照域名的字节来搜索的，所以搜索通配符名字的散列表比搜索确切名字的散列表慢。注意.nginx.org存储在通配符名字的散列表中，而不在确切名字的散列表中。由于正则表达式是一个一个串行测试的，因此该方式也是最慢的，而且不可扩展。

鉴于以上原因，请尽可能使用确切的名字。举个例子，如果使用nginx.org和www.nginx.org 来访问服务器是最频繁的，那么将它们明确地定义出来就更为有效，命令如下：

```
server {
    listen       80;

    server_name  nginx.org  www.nginx.org  *.nginx.org;

    ...
```

```
}
```

下面这种方法相比更简单，但是效率也更低：

```
server {
    listen       80;

    server_name  .nginx.org;

    ...
}
```

如果定义了大量的名字，或者定义了非常长的名字，那就需要在HTTP配置块中调整server_names_hash_max_size和server_names_hash_bucket_size的值。server_names_hash_bucket_size的默认值可能是32或64，也可能是其他值，这取决于CPU的缓存行的长度。如果这个值是32，那么定义“too.long.server.name.nginx.org”作为虚拟主机名就会失败，此时会显示下面的错误信息：

```
could not build the server_names_hash,

you should increase server_names_hash_bucket_size:

 32
```

出现了这种情况，那就需要将设置值扩大一倍，命令如下：

```
http {
    server_names_hash_bucket_size  64;

    ...
```

如果定义了大量名字，会得到如下另外一个错误信息：

```
could not build the server_names_hash,

you should increase either server_names_hash_max_size:

 512
or server_names_hash_bucket_size:

 32
```

那么应该先尝试设置server_names_hash_max_size的值，此值差不多等于名字列表的名字总量。如果还不能解决问题，或者服务器启动非常缓慢，再尝试增加server_names_hash_bucket_size的值，具体信息如下：

```
server_names_hash_max_size 512;

                #默认是

512kb，一般要查看系统给出确切

的值。这里一般是

cpu L1 的
```

```
4-5倍
```

---

server_names_hash_max_size的官方说明如下：

```
syntax:

   server_names_hash_max_size size;

     #<==参数语法

default:

  server_names_hash_max_size 512;

     #<==参数默认大小

context:

  http #<==仅能放置在

http标签段
```

参数作用：设置存放域名（server names）的最大散列表大小。细节见http://nginx.org/en/docs/ngx_core_module.html 。

第二个参数如下：

```
  server_names_hash_bucket_size 128;

#<==不能带单位！配置主机时必须设置该值，否则无法运行

Nginx，或者无法通过测试。该设置与

server_names_hash_max_size共同控制保存服务器名的

HASH表，

hash bucket size总是等于

hash表的大小，并且是一路处理器缓存大小的倍数。若

hash bucket size等于一路处理器缓存的大小，那么在查找键时，最坏的情况下在内存中查找的次数为

2。第一次是确定存储单元的地址，第二次是在存储单元中查找键值。若报出

hash max size 或

 hash bucket size的提示，则需要增加

server_names_hash_max_size的值
```

server_names_hash_bucket_size的官方说明如下：

```
syntax:

        server_names_hash_bucket_size size;

             #<==参数语法

default:

        server_names_hash_bucket_size 32|64 |128;

       #<==参数默认大小

context:

        http#<==仅能放置在

http标签段
```

参数作用：设置存放域名（server names）的最大散列表的存储桶（bucket）的大小。默认值依赖CPU的缓存行。细节见 http://nginx.org/en/docs/ngx_core_module.html 。

8.2.7　开启高效文件传输模式

1.设置参数：sendfile on；

sendfile参数用于开启文件的高效传输模式。同时将tcp_nopush和tcp_nodelay两个指令设置为on，可防止网络及磁盘I/O阻塞，提升Nginx工作效率。

sendfile参数的官方说明如下：

```
syntax:

        sendfile on | off;

                        #<==参数语法


default:

        sendfile off;

                                #<==参数默认大小


context:

        http,
```

```
server,

location,

if in location     #<==可以放置的标签段
```

参数作用：激活或禁用sendfile（）功能。sendfile（）是作用于两个文件描述符之间的数据拷贝函数，这个拷贝操作是在内核之中的，被称为“零拷贝”，sendfile（）比read和write函数要高效很多，因为，read和wrtie函数要把数据拷贝到应用层再进行操作。相关控制参数还有sendfile_max_chunk，读者可以自行查询。细节见 http://nginx.org/en/docs/http/ngx_http_core_module.html#sendfile 。

2.设置参数：tcp_nopush on；

tcp_nopush参数的官方说明如下：

```
syntax:

     tcp_nopush on | off;

                  #<==参数语法

default:
```

```
        tcp_nopush off;

                        #<==参数默认大小

    context:

        http,

     server,

     location                #<==可以放置的标签段
```

参数作用：激活或禁用Linux上的TCP_CORK socket选项，此选项仅仅当开启sendfile时才生效，激活这个.tcp_nopush参数可以允许把http response header和文件的开始部分放在一个文件里发布，其积极的作用是减少网络报文段的数量。细节见 http://nginx.org/en/docs/http/ngx_http_core_module.html 。

8.2.8　优化Nginx连接参数，调整连接超时时间

1.什么是连接超时

先来个比喻吧，某饭店请了服务员招待顾客，但是现在饭店不景气，此时，为多余的服务员发工资使得成本被提高，想减少饭店开支成本就得解雇服务员。

这里的服务员就相当于Nginx服务建立的连接，当服务器建立的连接没有接收处理请求时，可在指定的时间内就让它超时自动退出。还有当Nginx和FastCGI服务建立连接请求PHP时，如果因为一些原因（负载高、停止响应），FastCGI服务无法给Nginx返回数据，此时可以通过配置Nginx服务参数使其不会死等，因为前面用户还等着它返回数据呢，例如，可设置为如果请求FastCGI 10秒内不能返回数据，那么Nginx就中断本次请求，向用户汇报取不到数据的错误。

2.连接超时的作用

·将无用的连接设置为尽快超时，可以保护服务器的系统资源（CPU、内存、磁盘）。

·当连接很多时，及时断掉那些已经建立好的但又长时间不做事的连接，以减少其占用的服务器资源，因为服务器维护连接也是消耗资源

的。

·有时黑客或恶意用户攻击网站，就会不断地和服务器建立多个连接，消耗连接数，但是啥也不干，大量消耗服务器的资源，此时就应该及时断掉这些恶意占用资源的连接。

·LNMP环境中，如果用户请求了动态服务，则Nginx就会建立连接，请求FastCGI服务以及后端MySQL服务，此时这个Nginx连接就要设定一个超时时间，在用户容忍的时间内返回数据，或者再多等一会后端服务返回数据，具体的策略要根据具体业务进行具体分析。当然了，后端的FastCGI服务及MySQL服务也有对连接的超时控制。

简单地说，连接超时是服务的一种自我管理、自我保护的重要机制。

3.连接超时带来的问题，以及不同程序连接设定知识

服务器建立新连接也是要消耗资源的，因此，超时设置得太短而并发很大，就会导致服务器瞬间无法响应用户的请求，导致用户体验下降。

企业生产有些PHP程序站点会希望设置成短连接，因为PHP程序建立连接消耗的资源和时间相对要少些。而对于Java程序站点来说，一般建议设置长连接，因为Java程序建立连接消耗的资源和时间更多，这是

语言运行机制决定的。

4.Nginx连接超时的参数设置

（1）设置参数：keepalive_timeout 60；

用于设置客户端连接保持会话的超时时间为60秒。超过这个时间，服务器会关闭该连接，此数值为参考值。

keepalive_timeout参数的官方说明如下：

```
syntax:

        keepalive_timeout timeout [header_timeout];

       #<==参数语法


default:

        keepalive_timeout 75s;

                    #<==参数默认大小


context:

        http,

 server,
```

```
location                    #<==可以放置的标签段
```

参数作用：keep-alive可以使客户端到服务器端已经建立的连接一直工作不退出，当服务器有持续请求时，keep-alive会使用已经建立的连接提供服务，从而避免服务器重新建立新连接处理请求。

此参数设置一个keep-alive（客户端连接在服务器端保持多久后退出），其单位是秒，和HTTP响应header域的“Keep-Alive：timeout=time”参数有关，这些header信息也会被客户端浏览器识别并处理，不过有些客户端并不能按照服务器端的设置来处理，例如：MSIE大约60秒后会关闭keep-alive连接。细节见 http://nginx.org/en/docs/http/ngx_http_core_module.html 。

（2）设置参数：tcp_nodelay on；

用于激活tcp_nodelay功能，提高I/O性能。

tcp_nodelay参数的官方说明如下：

```
syntax:

    tcp_nodelay on | off;

                #<==参数语法
```

```
default:

      tcp_nodelay on;

                        #<==参数默认大小


context:

      http,

 server,

 location                  #<==可以放置的标签段
```

参数作用：默认情况下当数据发送时，内核并不会马上发送，可能会等待更多的字节组成一个数据包，这样可以提高I/O性能。但是，在每次只发送很少字节的业务场景中，使用tcp_nodelay功能，等待时间会比较长。

参数生效条件：激活或禁用TCP_NODELAY选项，当一个连接进入keep-alive状态时生效。细节见 http://nginx.org/en/docs/http/ngx_http_core_module.html 。

（3）设置参数：client_header_timeout 15；

用于设置读取客户端请求头数据的超时时间。此处的数值15，其单位是秒，为经验参考值。

client_header_timeout参数的官方说明如下：

```
syntax:

      client_header_timeout time;

                  #<==参数语法

default:

      client_header_timeout 60s;

                   #<==参数默认大小

context:

      http,

 server                                  #<==可以放置的标签段
```

参数作用：设置读取客户端请求头数据的超时时间。如果超过这个时间，客户端还没有发送完整的header数据，服务器端将返回“Request time out（408）”错误，此处的知识原理请参考前文HTTP原理来理解，

可指定一个超时时间，防止客户端利用http协议进行攻击。细节见 http://nginx.org/en/docs/http/ngx_http_core_module.html 。

（4）设置参数：client_body_timeout 15；

用于设置读取客户端请求主体的超时时间，默认值是60。

client_body_timeout参数的官方说明如下：

```
syntax:

      client_body_timeout time;

     #<==参数语法


default:

      client_body_timeout 60s;

      #<==默认值是

60秒


context:

      http,

 server,
```

```
location        #<==可以放置的标签段
```

参数作用：设置读取客户端请求主体的超时时间。这个超时仅仅为两次成功的读取操作之间的一个超时，非请求整个主体数据的超时时间，如果在这个超时时间内，客户端没有发送任何数据，Nginx将返回“Request time out（408）”错误，默认值是60，请参考前文http原理部分理解此参数，这样效果更好。细节见 http://nginx.org/en/docs/http/ngx_http_core_module.html 。

（5）设置参数：send_timeout 25；

用于指定响应客户端的超时时间。这个超时仅限于两个连接活动之间的时间，如果超过这个时间，客户端没有任何活动，Nginx将会关闭连接，默认值为60秒，可以改为参考值25秒。

send_timeout参数的官方说明如下：

```
syntax:

        send_timeout time:

                #<==参数语法

default:
```

```
        send_timeout 60s;
                #<==默认值是
60秒

context:
        http,
 server,
 location      #<==可以放置的标签段
```

参数作用：设置服务器端传送HTTP响应信息到客户端的超时时间，这个超时仅仅为两次成功握手后的一个超时，非请求整个响应数据的超时时间，如在这个超时时间内，客户端没有接收任何数据，连接将被关闭。细节见http://nginx.org/en/docs/http/ngx_http_core_module.html。

下面结合HTTP原理，画图（如图8-4所示）讲解上述几个超时参数。

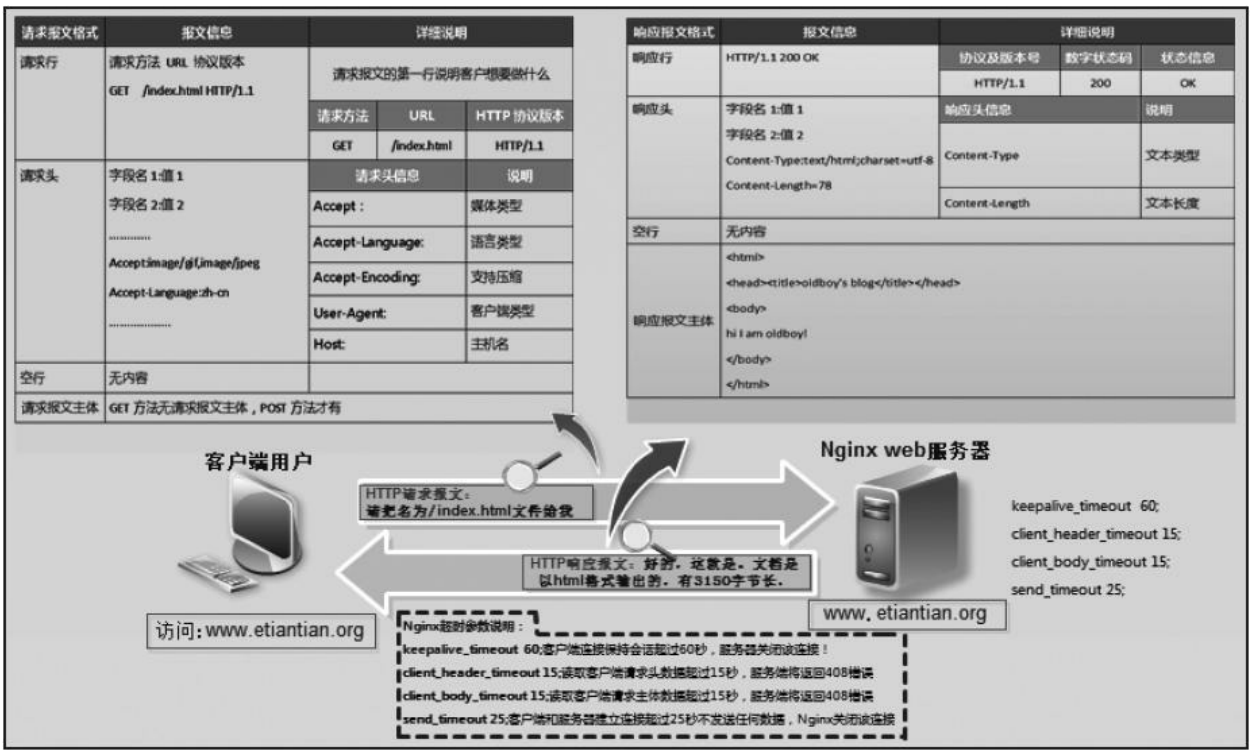


图8-4　结合HTTP原理，画图讲解Nginx配置超时参数图

8.2.9　上传文件大小的限制（动态应用）

下面介绍如何调整上传文件的大小（http Request body size）限制。

首先，在Nginx的主配置文件里加入如下参数：

```
client_max_body_size 8m;
```

具体大小根据公司的业务做调整，如果不清楚就先设置为8m吧，有关客户端请求主体的解释在HTTP原理一节已经解释过了，一般情况下，HTTP的post方法在提交数据时才会携带请求主体信息。

client_max_body_size参数的官方说明如下：

```
syntax:

     client_max_body_size size;

#<==参数语法

default:

     client_max_body_size 1m;
```

```
  #<==默认值是

1m
context:

      http,

 server,

 location    #<==可以放置的标签段
```

参数作用：设置最大的允许的客户端请求主体大小，在请求头域有“Content-Length”，如果超过了此配置值，客户端会收到413错误，意思是请求的条目过大，有可能浏览器不能正确显示。设置为0表示禁止检查客户端请求主体大小。此参数对提高服务器端的安全性有一定的作用。细节见http://nginx.org/en/docs/http/ngx_http_core_module.html 。

8.2.10　FastCGI相关参数调优（配合PHP引擎动态服务）

FastCGI参数是配合Nginx向后请求PHP动态引擎服务的相关参数，要想较好地理解参数细节，需要熟悉了解前面章节介绍的Nginx FastCGI客户端配合php-fpm（FastCGI服务器端）的原理。Nginx FastCGI工作的逻辑图如图8-5所示。

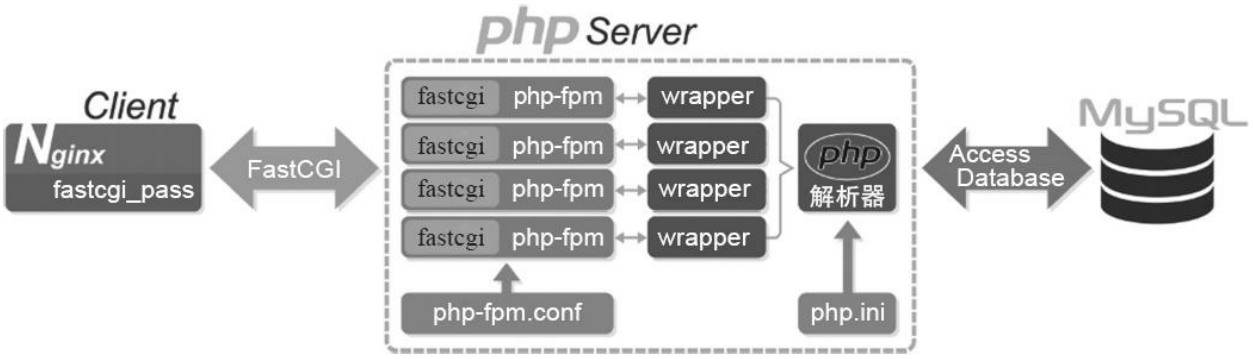


图8-5　Nginx FastCGI模式工作原理图

此处讲解的参数均为Nginx FastCGI客户端向后请求PHP动态引擎服务（php-fpm（FastCGI服务器端））的相关参数，属于Nginx的配置参数。表8-1是Nginx FastCGI常见参数的说明。

表8-1　Nginx FastCGI常见参数列表说明

Nginx FastCGI 相关参数	说明
fastcgi_connect_timeout	表示 Nginx 服务器和后端 FastCGI 服务器连接的超时时间，默认值为 60 秒，这个参数值通常不要超过 75 秒，因为建立的连接越多，消耗的资源就越多
fastcgi_send_timeout	设置 Nginx 允许 FastCGI 服务器端返回数据的超时时间，即在规定时间之内后端服务器必须传完所有的数据，否则，Nginx 将断开这个连接。其默认值为 60 秒
fastcgi_read_timeout	设置 Nginx 从 FastCGI 服务器端读取响应信息的超时时间，表示连接建立成功后，Nginx 等待后端服务器的响应时间，是 Nginx 已经进入后端的排队之中等候处理的时间
fastcgi_buffer_size	这是 Nginx FastCGI 的缓冲区大小参数，设定用来读取从 FastCGI 服务器端收到的第一部分响应信息的缓冲区大小，这里的第一部分通常会包含一个小的响应头部。默认情况下，这个参数的大小是由 fastcgi_buffers 指定的一个缓冲区的大小
fastcgi_buffers	设定用来读取从 FastCGI 服务器端收到的响应信息的缓冲区大小和缓冲区数量，默认值为 fastcgi_buffers 8 4k|8k;。 指定本地需要用多少和多大的缓冲区来缓冲 FastCGI 的应答请求。如果一个 PHP 脚本所产生的页面大小为 256KB，那么会为其分配 4 个 64KB 的缓冲区来缓存；如果页面大小大于 256KB，那么大于 256KB 的部分会缓存到 fastcgi_temp 指定的路径中，但是这并不是好方法，因为内存中的数据处理速度要快于硬盘。一般这个值应该为站点中 PHP 脚本所产生的页面大小的中间值，如果站点大部分脚本所产生的页面大小为 256KB，那么可以把这个值设置为“16 16k”、“4 64k”等
proxy_busy_buffers_size	用于设置系统很忙时可以使用的 proxy_buffers 大小，官方推荐的大小为 proxy_buffers*2
fastcgi_busy_buffers_size	用于设置系统很忙时可以使用的 fastcgi_buffers 大小，官方推荐的大小为 fastcgi_buffers *2。 默认值为 fastcgi_busy_buffers_size 8k|16k;
fastcgi_temp_file_write_size	FastCGI 临时文件的大小，可设置为 128 ~ 256KB
fastcgi_cache oldboy_nginx	表示开启 FastCGI 缓存并为其指定一个名称。开启缓存非常有用，可以有效降低 CPU 的负载，并且防止 502 错误的发生，但是开启缓存也可能引起其他问题，要根据具体情况来选择
fastcgi_cache_path	示例：fastcgi_cache_path /data/ngx_fcgi_cache levels=2:2 keys_zone= ngx_fcgi_cache:512min active=1d max_size=40g; fastcgi_cache 缓存目录，可以设置目录前列层级，比如 2:2 会生成 256*256 个子目录，keys_zone 是这个缓存空间的名字，cache 是用多少内存（这样热门的内容，nginx 会直接放入内存，提高访问速度），inactive 表示默认失效时间，max_size 表示最多用多少硬盘空间，需要注意的是 fastcgi_cache 缓存是先写在 fastcgi_temp_path 再移到 fastcgi_cache_path 中去的，所以这两个目录最好在同一个分区，从 0.8.9 之后可以在不同的分区，不过还是建议放在同一分区

（续）

Nginx FastCGI 相关参数	说明
fastcgi_cache_valid	示例：fastcgi_cache_valid 200 302 1h; 用来指定应答代码的缓存时间，实例中的值表示将 200 和 302 应答缓存一个小时。 示例：fastcgi_cache_valid 301 1d; 将 301 应答缓存 1 天。 示例：fastcgi_cache_valid any 1m; 将其他应答缓存设置为 1 分钟
fastcgi_cache_min_uses	示例：fastcgi_cache_min_uses 1; 设置请求几次之后响应将被缓存，1 表示一次即被缓存
fastcgi_cache_use_stale	示例：fastcgi_cache_use_stale error timeout invalid_header http_500; 定义在哪些情况下使用过期缓存
fastcgi_cache_key	示例：fastcgi_cache_key $request_method://$host$request_uri;fastcgi_cache_ key http://$host$request_uri; 定义 fastcgi_cache 的 key，示例中以请求的 URI 作为缓存的 key，Nginx 会取这个 key 的 md5 作为缓存文件，如果设置了缓存散列目录，Nginx 会从后往前取相应的位数做为目录。 注意一定要加上 $request_method 作为 cache key，否则如果先请求的为 head 类型，后面的 GET 请求返回为空

FastCGI Cache资料见 http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html#fastcgi_cache 。

FastCGI常见参数的Nginx配置示例如下：

```
[root@nginx conf]# cat nginx.conf
worker_processes  4;


worker_cpu_affinity 0001 0010 0100 1000;


worker_rlimit_nofile 65535;


user nginx;
```

```
events {
    use epoll;

    worker_connections  10240;

}
http {
    include       mime.types;

    default_type  application/octet-stream;

    sendfile        on;

    tcp_nopush on;

    keepalive_timeout  65;

    tcp_nodelay on;

    client_header_timeout 15;

    client_body_timeout 15;

    send_timeout 15;
```

```
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    server_tokens off;

fastcgi_connect_timeout 240;

fastcgi_send_timeout 240;

fastcgi_read_timeout 240;

fastcgi_buffer_size 64k;

fastcgi_buffers 4 64k;

fastcgi_busy_buffers_size 128k;

fastcgi_temp_file_write_size 128k;

#fastcgi_temp_path /data/ngx_fcgi_tmp;

fastcgi_cache_path /data/ngx_fcgi_cache levels=2:

2 keys_zone=ngx_fcgi_cache:

512m inactive=1d max_size=40g;
```

```
    #web..............
    server {
        listen       80;



        server_name  blog.etiantian.org;



        root   html/blog;



        location / {
            root   html/blog;



            index  index.php index.html index.htm;



                }
        location ~ .*\. (

php|php5)


$
        {
         fastcgi_pass  127.0.0.1:

9000;



         fastcgi_index index.php;



         include fastcgi.conf;
```

```
        fastcgi_cache ngx_fcgi_cache;

        fastcgi_cache_valid 200 302 1h;

        fastcgi_cache_valid 301 1d;

        fastcgi_cache_valid any 1m;

        fastcgi_cache_min_uses 1;

        fastcgi_cache_use_stale error timeout invalid_header http_500;

        fastcgi_cache_key http:

//$host$request_uri;


         }
         access_log  logs/web_blog_access.log  main;


           }
upstream blog_etiantian{
    server 10.0.0.8:

8000 weight=1;


}
    server {
        listen       8000;
```

```
        server_name  blog.etiantian.org;



        location / {
            proxy_pass http:


//blog_etiantian;



            proxy_set_header Host  $host;



            proxy_set_header X-Forwarded-For  $remote_addr;



              }
         access_log  logs/proxy_blog_access.log  main;



          }
}
```

为了让读者更加容易理解上述参数的作用及原理，老男孩为大家画了一个原理图，如图8-6所示。

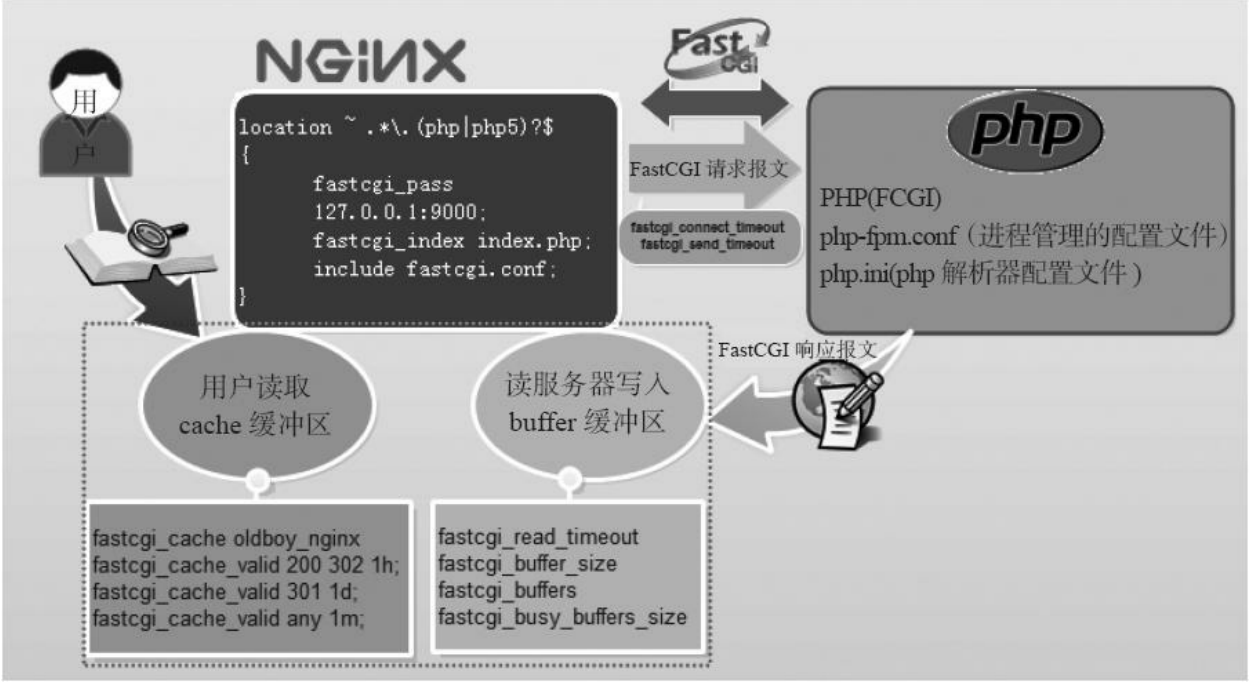


图8-6　根据HTTP原理及FastCGI原理讲解FastCGI参数优化

Nginx的FastCGI的相关参数和反向代理proxy的相关参数非常接近，大家在理解后文Nginx反向代理的参数时，可以结合此处FastCGI的参数进行对比理解。

8.2.11　配置Nginx gzip压缩实现性能优化

1.Nginx gzip压缩功能介绍

Nginx gzip压缩模块提供了压缩文件内容的功能，用户请求的内容在发送到用户客户端之前，Nginx服务器会根据一些具体的策略实施压缩，以节约网站出口带宽，同时加快数据传输效率，来提升用户访问体验。

2.Nginx gzip压缩的优点

·提升网站用户体验：发送给用户的内容小了，用户访问单位大小的页面就加快了，用户体验提升了，网站口碑就好了。

·节约网站带宽成本：数据是压缩传输的，因此节省了网站的带宽流量成本，不过压缩时会稍微消耗一些CPU资源，这个一般可以忽略。

此功能既能提升用户体验，又能使公司少花钱，一举多得。对于几乎所有的Web服务来说，这是一个非常重要的功能，Apache服务也有此功能。

3.需要和不需要压缩的对象

·纯文本内容压缩比很高，因此，纯文本的内容最好进行压缩，例

如：html、js、css、xml、shtml等格式的文件。

·被压缩的纯文本文件必须要大于1KB，由于压缩算法的特殊原因，极小的文件压缩后可能反而变大。

·图片、视频（流媒体）等文件尽量不要压缩，因为这些文件大多都是经过压缩的，如果再压缩很可能不会减小或减小很少，或者有可能增大，同时压缩时还会消耗大量的CPU、内存资源。

4.参数介绍及配置说明

此压缩功能与早期Apache服务的mod_deflate压缩功能很相似，Nginx的gzip压缩功能 依赖于ngx_http_gzip_module模块，默认已安装。

对应的压缩参数说明如下：

```
#压缩配置

gzip on;

#<==开启

gzip压缩功能。

gzip_min_length  1k;
```

#<==设置允许压缩的页面最小字节数，页面字节数从

header头的

Content-Length中获取。默认值是

0，表示不管页面多大都进行压缩。建议设置成大于

1K，如果小于

1K可能会越压越大。

```
  gzip_buffers     4 16k;
```

#<==压缩缓冲区大小。表示申请

4个单位为

16K的内存作为压缩结果流缓存，默认值是申请与原始数据大小相同的内存空间来存储

gzip压缩结果。

```
  gzip_http_version 1.1;
```

#<==压缩版本（默认

1.1，前端为

squid2.5时使用

1.0），用于设置识别

HTTP协议版本，默认是

1.1，目前大部分浏览器已经支持

GZIP解压，使用默认即可。

```
  gzip_comp_level 2;
```

#<==压缩比率。用来指定

gzip压缩比，

1压缩比最小，处理速度最快；

9压缩比最大，传输速度快，但处理最慢，也比较消耗

CPU资源。

```
  gzip_types  text/plain application/x-javascript text/css application/xml;
```

#<==用来指定压缩的类型，“

text/html”类型总是会被压缩，这个就是

HTTP原理部分讲的媒体类型。

```
    gzip_vary on;

#<==vary header支持。该选项可以让前端的缓存服务器缓存经过

gzip压缩的页面，例如用

Squid缓存经过

Nginx压缩的数据
```

完整的配置如下：

```
    gzip on;
    gzip_min_length  1k;
    gzip_buffers     4 32k;
    gzip_http_version 1.1;
    gzip_comp_level 9;
    #gzip_types    text/plain application/x-javascript text/css application/xml;
#<== 老的不正确写法
    gzip_types  text/css text/xml application/javascript;
    gzip_vary on;
```

不同的Nginx版本中，gzip_types的配置可能会有不同，上述配置示例适合Nginx 1.6.3。对应的文件类型，请查看安装目录下的mime.types文件。

更多官方资料请看http://nginx.org/en/docs/http/ngx_http_gzip_module.html 。

5.Nginx压缩配置效果检查

可通过火狐浏览器加yslow插件查看gzip压缩及expires缓存结果。提前安装好火狐浏览器，并且安装好yslow插件，开启监控，然后打开LNMP时安装的博客地址，就可以看到如图8-7所示的压缩结果。

↑ TYPE	SIZE (KB)	GZIP (KB)	COOKIE RECEIVED (bytes)	COOKIE SENT (bytes)	HEADERS	URL	EXPIRES (Y/M/D)
⊞ doc (1)	0.06K						
⊟ js (1)	72.1K						
js	72.1K	24.0K				http://blog.etiantian.org/oldboy/jquery.js	2012/10/20

图8-7　检查Nginx gzip压缩效果图

6.重要的前端网站调试工具介绍

常见的前端网站调试工具调试工具有如下几种。

·Google浏览器（Chrome）：通过该浏览器直接按F12键即可查看压缩及缓存结果，另外，谷歌浏览器（Chrome）上也可以直接安装yslow插件（如图8-8所示）。

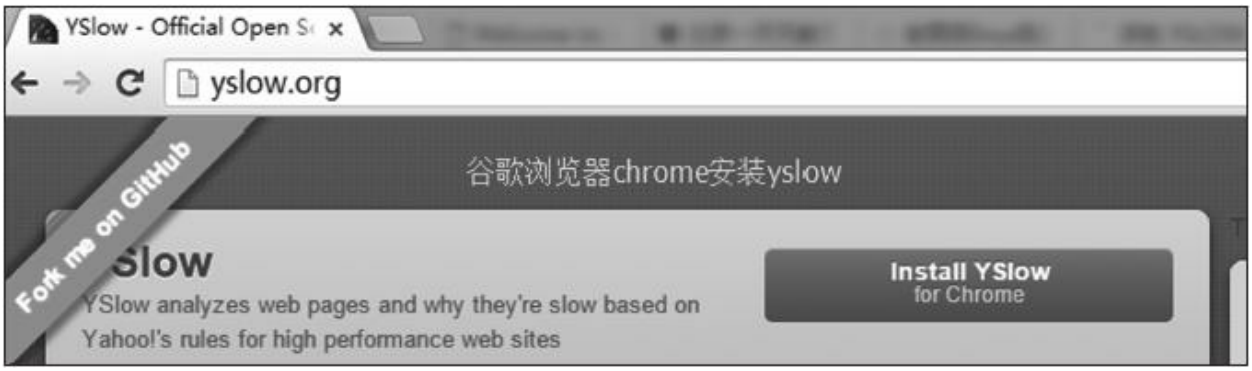


图8-8　调试工具yslow.org网站

·火狐浏览器：在该浏览器上安装firebug、yslow（http://yslow.org/），即可进行调试（如图8-9所示）。

·IE浏览器：在该浏览器上安装httpwatch即可进行调试（省略）。

图8-9　调试工具yslow安装方法图

8.2.12　配置Nginx expires缓存实现性能优化

1.Nginx expires功能介绍

简单地说，Nginx expires的功能就是为用户访问的网站内容设定一个过期时间，当用户第一次访问这些内容时，会把这些内容存储在用户浏览器本地，这样用户第二次及以后继续访问该网站时，浏览器会检查加载已经缓存在用户浏览器本地的内容，就不会去服务器下载了，直到缓存的内容过期或被清除为止。

更深入地理解：expires的功能就是允许通过Nginx配置文件控制HTTP的"Expires"和"Cache-Control"响应头部内容，告诉客户端浏览器是否缓存和缓存多久以内访问的内容。这个expires模块控制Nginx服务器应答时的expires头内容和Cache-Control头的max-age指令。缓存的有效期可以设置为相对于源文件的最后修改时刻或客户端的访问时刻。

这些HTTP头向客户端表明了内容的有效性和持久性。如果客户端本地有内容缓存，则内容就可以从缓存（除非已经过期）而不是从服务器中读取，然后客户端会检查缓存中的副本，看其是否过期或失效，以决定是否重新从服务器获得内容更新。

2.Nginx expires作用介绍

在网站的开发和运营中，视频、图片、CSS、JS等网站元素的更改机会较少，特别是图片，这时可以将图片设置在客户浏览器本地缓存365天或3650天，而将CSS、JS、html等代码缓存10~30天。这样用户第一次打开页面后，会在本地的浏览器按照过期日期缓存相应的内容，下次用户再打开类似的页面时，重复的元素就无需下载了，从而加快用户访问速度。用户的访问请求和数据减少了，也可节省大量的服务器端带宽。此功能同Apache的expires功能类似。

3.Nginx expires功能优点

·expires可以降低网站的带宽，节约成本。

·加快用户访问网站的速度，提升用户访问体验。

·服务器访问量降低了，服务器压力就减轻了，服务器成本也会降低，甚至可以节约人力成本。

对于几乎所有的Web服务来说，这是非常重要的功能之一，Apache服务也有此功能。

4.Nginx expires配置详解

前面已经介绍了expires的功能原理，接下来就来配置Nginx expires的功能。这里以location标签为例进行讲解，通过location URI规则将需要缓存的扩展名列出来，然后指定缓存时间。如果针对所有内容设置缓

存，也可以不用location。Nginx默认安装了expires功能。

（1）根据文件扩展名进行判断，添加expires功能范例

范例1：

```
location ~ .*\.(
gif|jpg|jpeg|png|bmp|swf)
$
{
    expires      3650d;

}
```

该范例的意思是当用户访问网站URL结尾的文件扩展名为上述指定类型的图片时，设置缓存3650天，即10年。

范例2：

```
location ~ .*\.(
js|css)
$
{
    expires      30d;

}
```

该范例的意思是当用户访问网站URL结尾的文件扩展名为js、css类型的元素时，设置缓存30天，即1个月。

上述两个范例的实际配置结果如下：

```
###www
    server {
        listen       80;



        server_name  www.etiantian.org;



        location / {
            root   html/www;



            index  index.html index.htm;



        }
        location ~ .*\.(


gif|jpg|jpeg|png|bmp|swf)


$
        {
           expires      10y;



           root   html/www;



        }
       location ~ .*\.(
```

```
js|css)

$
       {
          expires      30d;


       }
        access_log  /app/logs/www_access.log  main;


    }
```

特别注意，location内容一般放到虚拟主机配置中，即server标签中。

（2）根据URI中的路径（目录）进行判断，添加expires功能范例

范例3：

```
## Add expires header according to URI (

path or dir)

.
location ~ ^/ (

images|javascript|js|css|flash|media|static)

/ {
    expires 360d;
```

```
}
```

该范例的意思是当用户访问网站URL中包含上述路径（例：images、js、css，这些在服务器端是程序目录）时，把访问的内容设置缓存360天，即1年。

（3）单个文件添加expires功能的范例

下面的命令会给robots.txt机器人文件设置过期时间（这里为7天），在这7天并不记录404错误日志。

```
location ~ (
robots.txt)
 {
          expires 7d;
          break;
}
```

5.Nginx expires配置效果检查

检查Nginx expires的方法和检查Nginx gzip的方法相同。

通过火狐浏览器加yslow插件查看gzip压缩及expires缓存结果时，要

提前安装好火狐浏览器，并且要安装好yslow插件，开启监控，然后打开LNMP时安装的博客地址（带有图片、JS、CSS），就可以看到如图8-10所示的缓存结果了。

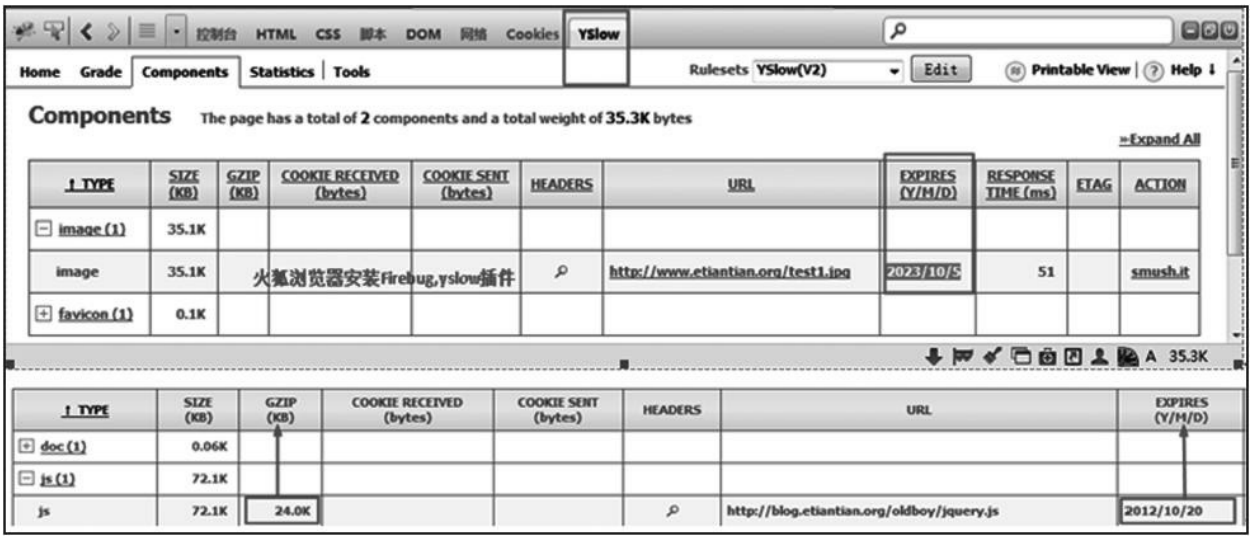


图8-10　查看expires是否生效图

在Linux客户端可通过如下curl命令查看图片URL的缓存header信息：

```
[root@oldboy ~]# curl -I http:


//blog.etiantian.org/23.jpg
HTTP/1.1 200 OK
Server:


 nginx
Date:


 Fri,
```

21 Sep 2012 02:

07:

18 GMT
Content-Type:

image/jpeg
Content-Length:

59491
Last-Modified:

Mon,

13 Aug 2012 02:

26:

18 GMT
Connection:

keep-alive
ETag:

"502865ca-e863"
Expires:

Mon,

19 Sep 2022 02:

07:

```
18 GMT #<==缓存的过期时间

Cache-Control:

 max-age=315360000        #<==缓存的总时间，单位为秒

Accept-Ranges:

 bytes
```

6.Nginx expires功能缺点及解决方法

几乎所有的事物都是有两面性，没有十全十美的人和事。Nginx expires功能也不例外，虽然这个功能很好，但是也会给企业带来一些困惑。

当网站被缓存的页面或数据更新了，此时用户端看到的可能还是旧的已经缓存的内容，这样就会影响用户体验，那么如何解决这个问题呢？解决办法如下。

第一，对于经常需要变动的图片等文件，可以缩短对象缓存时间，例如：谷歌和百度的首页的图片经常根据不同的日期换成一些节日的图，所以这里可以将这个图片设置为缓存期为1天。

第二，当网站改版或更新时，可以在服务器将缓存的对象改名（网

站代码程序）。

·对于网站的图片、附件，一般不会被用户直接修改，用户层面上的修改图片，实际上是重新传到服务器，虽然内容一样但是是一个新的图片名了。

·网站改版升级会修改JS、CSS元素，若改版时对这些元素改了名，会使得前端的CDN及用户端需要重新缓存内容。

7.企业网站缓存日期曾经的案例参考

若企业的业务和网站访问量不同，那么网站的缓存期时间设置也是不同的，比如，如下企业所用的缓存日期就是不一样的。

·51CTO：1周

·新浪：15天

·京东：25年

·淘宝：10年

8.企业网站有可能不希望被缓存的内容

·广告图片，用于广告服务，都缓存了就不好控制展示了。

·网站流量统计工具（JS代码），都缓存了流量统计就不准了。

·更新很频繁的文件（google的logo），这个如果按天，缓存效果还是显著的。

8.3 Nginx日志相关优化与安全

8.3.1 编写脚本实现Nginx access日志轮询

当用户请求一个软件时，绝大多数软件都会记录用户的访问情况，Nginx服务也不例外。Nginx软件目前还没有类似Apache的通过cronolog或rotatelog对日志分割处理的功能，但是，运维人员可以利用脚本开发、Nginx的信号控制功能或reload重新加载，来实现日志的自动切割、轮询。

详细操作过程如下。

1）配置日志切割脚本，命令如下：

```
[root@oldboy ~]# mkdir /server/scripts/ -p
[root@oldboy ~]# cd /server/scripts/
[root@oldboy scripts]# vim cut_nginx_log.sh
cd /application/nginx/logs &&\
/bin/mv www_access.log www_access_$(


date +%F -d -1day)


.log      #<==将日志按日期改成前一天的名称


/application/nginx/sbin/nginx -s reload      #<==重新加载


nginx使得触发重新生成访问
```

```
日志文件

提示：实际上脚本的功能很简单，就是改名日志，然后加载

nginx，重新生成文件记录日志
```

2）将这段脚本保存后加入到服务器端的定时任务配置里，让此脚本在每天凌晨0点执行，就可以实现日志的每天分割功能，操作结果如下：

```
[root@oldboy scripts]# crontab -e      #<==打开编辑功能后，加入如下内容


#cut nginx access log by oldboy at 201409
00 00 * * * /bin/sh /server/scripts/cut_nginx_log.sh >/dev/null 2>&1
```

此处意思是每天凌晨0点执行后面的/server/scripts/cut_nginx_log.sh脚本，>/dev/null 2>&1表示任何输出都不要。这个是Linux的基础服务之一，具体请参考老男孩的其他基础课程或相关资料。

最终切割后的日志效果如下：

```
[root@oldboy scripts]# ll /application/nginx/logs/总用量


 27752
-rw-r--r--. 1 root root 20241716 10月
```

```
  9 12:

03 access.log
-rw-r--r--. 1 root root    27428 10月

 10 04:

22 error.log
-rw-r--r--. 1 root root        5 10月

  9 09:

56 nginx.pid
-rw-r--r--. 1 root root      264 9月

 26 05:

08 www_access_2014-09-25.log
-rw-r--r--. 1 root root        0 9月

 26 05:

12 www_access_2014-09-26.log
-rw-r--r--. 1 root root        0 9月

 28 00:

00 www_access_2014-09-27.log
-rw-r--r--. 1 root root    48717 9月

 28 03:

11 www_access_2014-09-28.log
-rw-r--r--. 1 root root  8072741 10月
```

```
 9 17:

25 www_access_2014-10-09.log
```

说明：这里按照不同的日期生成日志。

8.3.2 不记录不需要的访问日志

在实际工作中，对于负载均衡器健康节点检查或某些特定文件（比如图片、JS、CSS）的日志，一般不需要记录下来，因为在统计PV时是按照页面计算的，而且日志写入太频繁会消耗大量磁盘I/O，降低服务的性能。

具体配置方法如下：

```
location ~ .*\.(
js|jpg|JPG|jpeg|JPEG|css|bmp|gif|GIF)
$ {
    access_log off;
}
```

这里用location标签匹配不记录日志的元素扩展名，然后关掉日志。

8.3.3 访问日志的权限设置

假如日志目录为/app/logs，则授权方法如下：

```
chown -R root.root /app/logs
chmod -R 700 /app/logs
```

不需要在日志目录上给Nginx用户读或写许可，但很多网友都没注意这个问题，他们把该权限直接给了Nginx或Apache用户，这就成为安全隐患。

8.4 Nginx站点目录及文件URL访问控制

8.4.1 根据扩展名限制程序和文件访问

Web 2.0时代，绝大多数网站都是以用户为中心的，例如：bbs、blog、sns产品，这几个产品都有一个共同特点，就是不但允许用户发布内容到服务器，还允许用户发图片甚至上传附件到服务器上，由于为用户开了上传的功能，因此给服务器带来了很大的安全风险。虽然很多程序在上传前会做一定的控制，例如：文件大小、类型等，但是，一不小心就会被黑客钻了空子，上传了木马程序。

下面将利用Nginx配置禁止访问上传资源目录下的PHP、Shell、Perl、Python程序文件，这样用户即使上传了木马文件也没法执行，从而加强了网站的安全。

范例1：配置Nginx，禁止解析指定目录下的指定程序。

```
location ~ ^/images/.*\.(
php|php5|sh|pl|py)
$
        {
           deny all;
```

```
        }
location ~ ^/static/.*\. (

php|php5|sh|pl|py)

$
        {
           deny all;



        }
location ~* ^/data/ (

attachment|avatar)

/.*\. (

php|php5)

$
    {
        deny all;



    }
```

对上述目录的限制必须写在Nginx处理PHP服务配置的前面，如下：

```
location ~ .*\. (

php|php5)
```

```
$
{
  fastcgi_pass  127.0.0.1:

9000;

  fastcgi_index index.php;

  include fcgi.conf;

}
```

范例2：Nginx下配置禁止访问*.txt和*.doc文件。

实际配置信息如下：

```
location ~* \.(

txt|doc)

$ {
    if  (

-f $request_filename)

 {
    root /data/www/www;

    #rewrite ...
```

..可以重定向到某个

```
URL
    break;



    }
}
location ~* \.(
```

```
txt|doc)
```

```
${
    root /data/www/www;


    denyall;


}
```

8.4.2 禁止访问指定目录下的所有文件和目录

范例1：配置禁止访问指定的单个或多个目录。

禁止访问单个目录的命令如下：

```
location ~ ^/ (

static)

/ {
        deny all;

}
location ~ ^/static {
        deny all;

}
```

禁止访问多个目录的命令如下：

```
location ~ ^/ (

static|js)

 {
        deny all;
```

```
}
```

范例2：禁止访问目录并返回指定的HTTP状态码，命令如下：

```
      server {
          listen       80;


          server_name  www.etiantian.org etiantian.org;


              root   /data0/www/www;


              index  index.html index.htm;


              access_log  /app/logs/www_access.log  commonlog;


              location /admin/ { return 404;


    }
              location /templates/ { return 403;


    }
}
```

作用：禁止访问目录下的指定文件，或者禁止访问指定目录下的所有内容。

最佳应用场景：对于集群的共享存储，一般是存放静态资源文件，

所以可禁止执行指定扩展名的程序，例：.php、.sh、.pl、.py。

8.4.3 限制网站来源IP访问

下面介绍如何使用ngx_http_access_module限制网站来源IP访问。

案例环境：phpmyadmin数据库的Web客户端，内部开发人员用的。

范例1：禁止某目录让外界访问，但允许某IP访问该目录，且支持PHP解析，命令如下：

```
location ~ ^/oldboy/ {
    allow 202.111.12.211;

    deny all;

}
    location ~ .*\.(
php|php5)
$ {
            fastcgi_pass  127.0.0.1:
9000;

            fastcgi_index index.php;

            include fastcgi_params;
```

```
            fastcgi_param  SCRIPT_FILENAME $document_root$fastcgi_script_name;

    }
}
```

范例2：限制指定IP或IP段访问，命令如下：

```
location / {
    deny 192.168.1.1;

    allow 192.168.1.0/24;

    allow 10.1.1.0/16;

    deny all;

}
```

参考：http://nginx.org/en/docs/http/ngx_http_access_module.html 。

企业问题案例：Nginx做反向代理的时候可以限制客户端IP吗？

解答：可以，具体方法如下。

方法1：使用if来控制，命令如下：

```
 if  (

 $remote_addr = 10.0.0.7 )

 {
 return 403;

 }
if  (

 $remote_addr = 218.247.17.130 )

 {
 set $allow_access_root 'true';

 }
 http:

//nginx.org/en/docs/varindex.html
```

方法2：利用deny和allow只允许IP访问，命令如下：

```
location / {
    root   html/blog;

    index  index.php index.html index.htm;

    allow 10.0.0.7;

    deny all;
```

```
    }
```

方法3：只拒绝某些IP访问，命令如下：

```
location / {
    root   html/blog;

    index  index.php index.html index.htm;

    deny 10.0.0.7;

    allow all;

    }
```

注意事项：

·deny一定要加一个IP，否则会直接跳转到403，不再往下执行了，如果403默认页是在同一域名下，会造成死循环访问。

·对于allow的IP段，从允许访问的段位从小到大排列，如127.0.0.0/24的下面才能是10.10.0.0/16，其中：

·24表示子网掩码：255.255.255.0

·16表示子网掩码：255.255.0.0

·8表示子网掩码：255.0.0.0

·以deny all；结尾，表示除了上面允许的，其他的都禁止。如：

```
deny 192.168.1.1;

allow 127.0.0.0/24;

allow 192.168.0.0/16;

allow 10.10.0.0/16;

denyall;
```

8.4.4 配置Nginx，禁止非法域名解析访问企业网站

问题：Nginx如何防止用户IP访问网站（恶意域名解析，也相当于是直接IP访问企业网站）？

方法1：让使用IP访问网站的用户，或者恶意解析域名的用户，收到501错误，命令如下：

```
server {
listen 80 default_server;


server_name _;


return 501;


}
```

说明：直接报501错误，从用户体验上不是很好。

方法2：通过301跳转到主页，命令如下：

```
server {
listen 80 default_server;
```

```
server_name _;

rewrite ^(
.*)
 http:
//blog.etiantian.org/$1 permanent;

}
```

方法3：发现某域名恶意解析到公司的服务器IP，在server标签里添加以下代码即可，若有多个server则要多处添加。

```
if  (
$host !
~ ^www/.eduoldboy/.com$)
{
    rewrite ^(
.*)
  http:
```

```
    //www.eduoldboy.com$1 permanent;


    }
```

上面代码的意思是如果header信息的host主机名字段非www.eduoldboy.com， 就301跳转到www.eduoldboy.com 。

8.5　Nginx图片及目录防盗链解决方案

1.什么是资源盗链

简单地说，就是某些不法网站未经许可，通过在其自身网站程序里非法调用其他网站的资源，然后在自己的网站上显示这些调用的资源，达到填充自身网站的效果。这一举动不仅浪费了调用资源网站的网络流量，还造成其他网站的带宽及服务压力吃紧，甚至宕机。

下面通过示意图阐述资源被盗链原理，如图8-11所示。

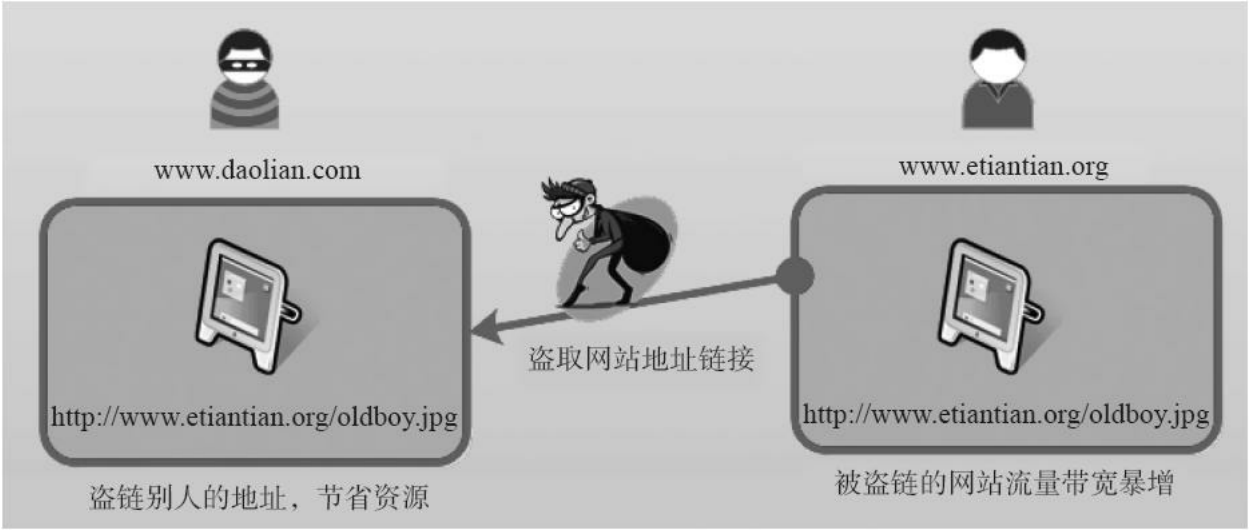


图8-11　资源被盗链原理详细示意图

2.网站资源被盗链带来的问题

若网站图片及相关资源被盗链，最直接的影响就是网络带宽占用加

大了，带宽费用多了，网络流量也可能忽高忽低，Nagios/Zabbix等报警服务频繁报警，类似图8-12所示。

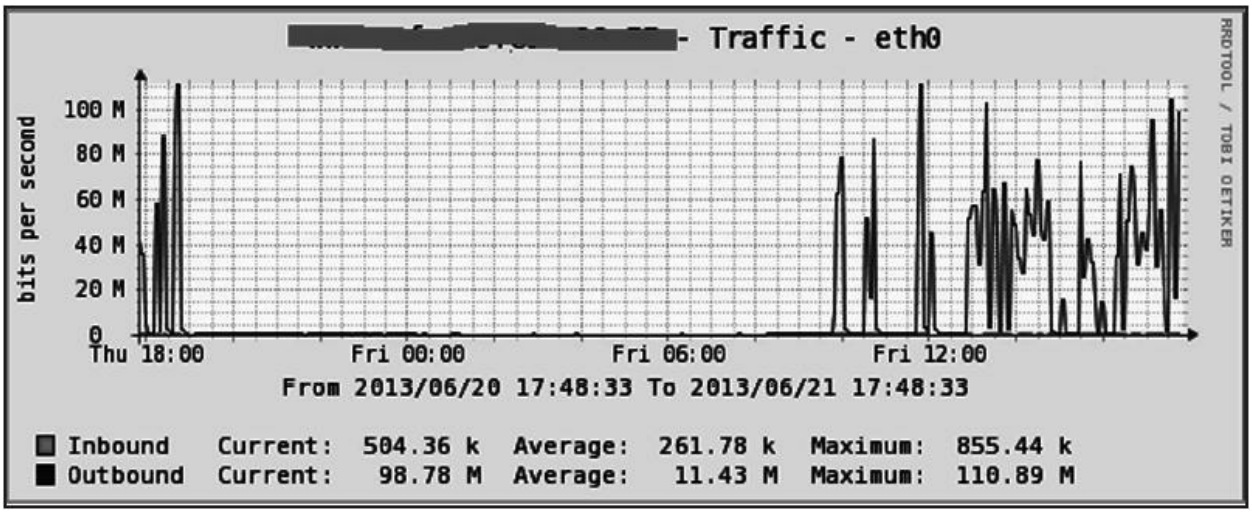


图8-12　因盗链导致的流量飙升图

最严重的情况就是网站的资源被非法使用，使网站带宽成本加大和服务器压力加大，这有可能导致数万元的损失，且网站的正常用户访问也会受到影响。

3.企业真实案例：网站资源被盗链，出现严重问题

某日，接到从事运维工作的朋友的紧急求助，其公司的CDN源站，源站的流量没有变动，但CDN加速那边的流量无故超了好几个GB，不知道怎么处理。

该故障的影响：由于是购买的CDN网站加速服务，因此虽然流量多了几个GB，但是业务未受影响。只是，这么大的异常流量，持续下去

可直接导致公司无故损失数万元。解决这个问题可体现运维的价值。

那么这样的问题如何及时发现，又如何处理呢？

本节给大家讲述几个发现问题的方法，如何处理盗链，后文会详细讲解。

第一，对IDC及CDN带宽做监控报警。

第二，作为高级运维或运维经理，每天上班的重要任务，就是经常查看网站流量图，关注流量变化，关注异常流量。

第三，对访问日志做分析，迅速定位异常流量，并且和公司市场推广等保持较好的沟通，以便调度带宽和服务器资源，确保网站正常的访问体验。

更多企业案例及实战解决方案见老男孩的博文：http://oldboy.blog.51cto.com/2561410/909696 。

4.常见防盗链解决方案的基本原理

（1）根据HTTP referer实现防盗链

在HTTP协议中，有一个表头字段叫referer，使用URL格式来表示是哪里的链接用了当前网页的资源。通过referer可以检测访问的来源网页，如果是资源文件，可以跟踪到显示它的网页地址，一旦检测出来源

不是本站，马上进行阻止或返回指定的页面。

HTTP referer是header的一部分，当浏览器向Web服务器发送请求时，一般会带上referer，告诉服务器我是从哪个页面链接过来的，服务器借此获得一些信息用于处理。Apache、Nginx、Lighttpd三者都支持根据HTTP referer实现防盗链，referer是目前网站图片、附件、html等最常用的防盗链手段。图8-13是referer防盗链的基本原理图。

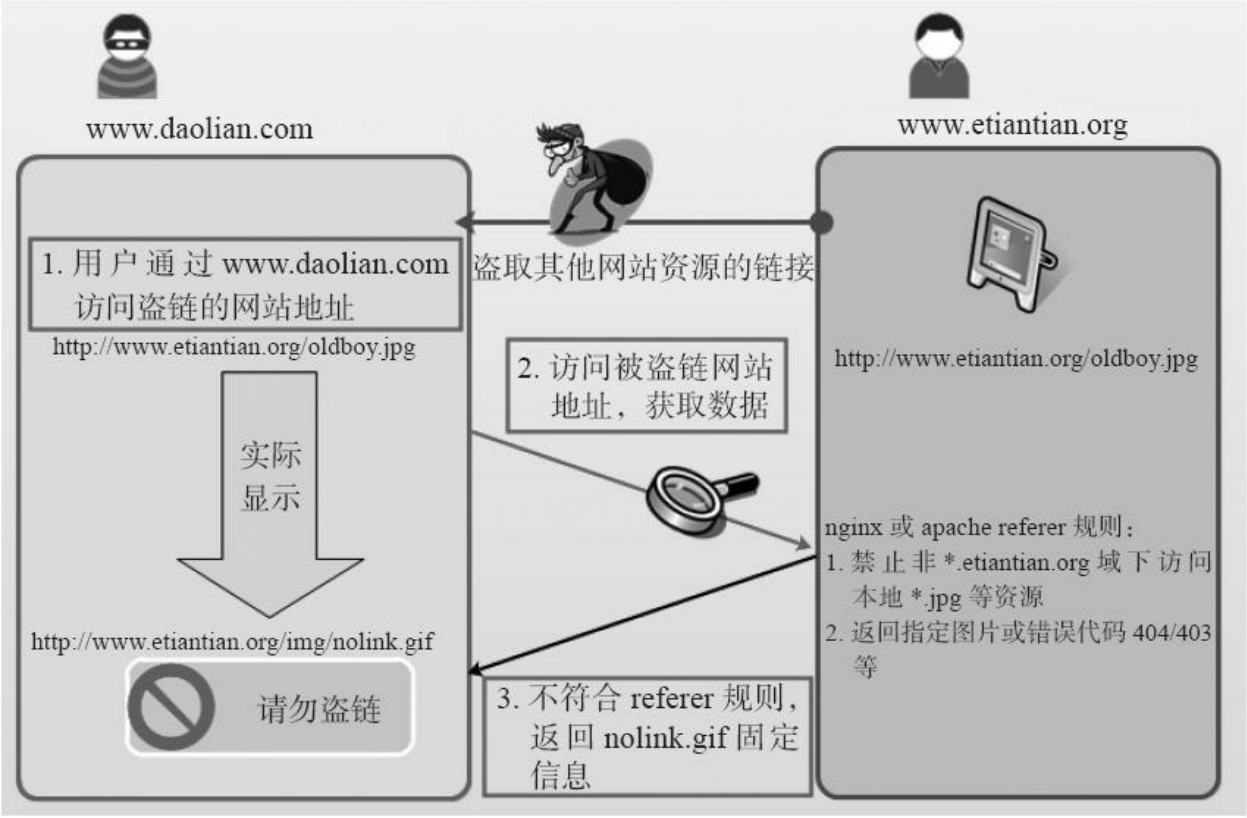


图8-13　通过referer解决盗链问题基本原理

（2）根据cookie防盗链

对于一些特殊的业务数据，例如流媒体应用通过ActiveX显示的内

容（例如，Flash、Windows Media视频、流媒体的RTSP协议等），因为它们不向服务器提供referer header，所以若采用上述的referer的防盗链手段，就达不到想要的效果。

对于Flash、Windows Media视频这种占用流量较大的业务数据，防盗链是比较困难的，此时可以采用Cookie技术，解决Flash、Windows Media视频等的防盗链问题。

例如：ActiveX插件不传递referer，但会传递Cookie，可以在显示ActiveX的页面的<head></head>标签内嵌入一段JavaScript代码，设置“Cookie：Cache=av”如下：

```
<script> document.cookie="Cache=av;

domain=domain.com;

path=/";

 </script>
```

然后就可以通过各种手段来判断这个Cookie的存在，以及验证其值的操作了。

根据Cookie来防盗链的技术非本书的内容，读者了解即可，如果企业确实有需要，可以阅读其他书籍或进入交流群获取这部分的知识。

（3）通过加密变换访问路径实现防盗链

此种方法比较适合视频及下载类业务数据的网站。例如：Lighttpd有类似的插件mod_secdownload来实现此功能。先在服务器端配置此模块，设置一个固定用于加密的字符串，比如oldboy，然后设置一个url前缀，比如/mp4/，再设置一个过期时间，比如1小时，然后写一段PHP代码，利用加密字符串和系统时间等通过md5算法生成一个加密字符串。最终获取到的文件的URL链接中会带有一个时间戳和一个加密字符的md5数值，在访问时系统会对这两个数据进行验证。如果时间不在预期的时间段内（如1小时内）则失效；如果时间戳符合条件，但是加密的字符串不符合条件也会失效，从而达到防盗链的效果。

PHP代码实例如下：

```
<php
$secret = "oldboy";

    // 加密字符串，必须和

lighttpd.conf里的保持一致

$uri_prefix = "/mp4/";

    // 虚拟的路径、前缀，必须和

lighttpd.conf里的保持一致
```

```
$file = "/test.mp4";

     // 实际文件名，必须加

"/"斜杠


$timestamp = time();

     // current timestamp
$t_hex = sprintf(

"%08x",

 $timestamp);


$m = md5(

$secret.$file.$t_hex);


printf(

'%s',

 $uri_prefix,

 $m,

 $t_hex,
```

```
    $file,
    $file);
        //生成
    url地址串
    >
```

根据Lighttpd的插件mod_secdownload来进行防盗链的技术非本书的内容，读者了解即可，如果企业确实有需要，可以阅读其他书籍，或者进入交流群获取这部分的知识。

5.Nginx Web服务实现防盗链实战

在默认情况下，只需要进行简单的配置，即可实现防盗链处理。请看下面的实例。

（1）利用referer，并且针对扩展名rewrite重定向

下面的代码为利用referer且针对扩展名rewrite重定向，即实现防盗链的Nginx配置。

```
    location ~* \.(
    jpg|gif|png|swf|flv|wma|wmv|asf|mp3|mmf|zip|rar)
```

```
$ {
  valid_referers none blocked *.etiantian.org etiantian.org;


if  (

$invalid_referer)

 {
    rewrite ^/ http:

//www.etiantian.org/img/nolink.jpg;


  }
}
```

 提示：要根据自己公司的实际业务（是否有外链的合作），进行域名设置。

设置expires的方法如下：

```
[root@oldboy bbs]# cat /application/nginx/conf/extra/www.conf
    server {
        listen       80;


        server_name  www.etiantian.org;


            root   html/www;
```

```
            index  index.html index.htm;

            access_log  logs/www_access.log main;

#Preventing hot linking of images and other file types
location ~* ^.+\. (

gif|jpg|png|swf|flv|rar|zip)

$ {
    valid_referers none blocked server_names *.etiantian.org etiantian.org;

    if  (

$invalid_referer)

 {
        rewrite ^/ http:

//bbs.etiantian.com/img/nolink.jpg;

    }
    access_log off;

    root html/www;

    expires 1d;

    break;
```

```
    }
}
```

（2）利用referer，并且针对站点目录过滤返回错误码

针对目录的方法如下：

```
location /images {
  root /data0/www/www;


  valid_referers none blocked *.etiantian.org etiantian.org;


if  (


$invalid_referer)


{
    return 403;


  }
}
```

在上面这段防盗链设置中，分别针对不同文件类型和不同的目录进行了设置，读者可以根据自己的需求进行类似设定。下面是上述代码的说明：

·“jpg|gif|png|swf|flv|wma|wmv|asf|mp3|mmf|zip|rar”表示对

以.jpg、.gif、.png、.swf、.flv、.wma、.wmv、.asf、.mp3、.mmf、.zip和.rar为后缀的文件实行防盗链处理。

·“*.etiantian.org etiantian.org”表示这个请求可以正常访问上面指定的文件资源。

·if{}中内容的意思是如果地址不是上面指定的地址就跳转到通过rewrite指定的地址，也可以直接通过return返回403错误。

·return 403为自定义的http返回状态码。

·rewrite^/http://www.etiantian.org/img/nolink.jpg；”表示显示一张防盗链图片。

·access_log off；表示不记录访问日志，减轻压力。

·expires 3d指的是所有文件设置3天的浏览器缓存。

6.NginxHttpAccessKeyModule实现防盗链介绍

如果不怕麻烦，有条件实现的话，推荐使用NginxHttpAccessKeyModule。

其运行方式是：如download目录下有一个file.zip的文件。对应的URI是http://www.abc.com/download/file.zip ，使用ngx_http_accesskey_module模块后就成了

http://www.abc.com/download/file.zipkey=09093abeac094，只有正确地给定了key值，才能下载download目录下的file.zip，而且key值是与用户的IP相关的，这样就可以避免被盗链了。据说，现在NginxHttpAccessKeyModule连迅雷都可以防了，读者可以尝试一下。

7.在产品设计上解决盗链方案

产品设计时，处理盗链问题可将计就计，为网站上传的图片增加水印。例如：图8-14就是为网站上传的图片增加的水印。

图8-14　网站被盗链后的提示图片示例

为图片添加版权水印是很有效的方法。网站直接转载图片一般是为了快捷，但是对于有水印的图片，很多站长是不愿意转载的。

8.Nginx防盗链花絮

下面实战模拟演示盗链。

1）假定blog.etiantian.com是非法盗链的网站域名，先编写如下的

oldboy.html程序：

```
<html>
<head>
<title>老男孩教育



</title>
</head>
<body bgcolor=green>老男孩的博客!


<br>我的博客是


<a href="http:


//oldboy.blog.51cto.com" target="_blank">博客地址


</a>
<img src="http:


//www.etiantian.org/stu.jpg">
</body>
</html>
```

这个非法盗链的访问地址是http://blog.etiantian.com/oldboy.html，网站里会加载www.etiantian.org 网站的图片stu.jpg，即盗用该网站图片并消耗了www.etiantian.org 正常网站的带宽。

2）假定我们维护的网站为www.etiantian.org，设定一张存在的图片的地址为www.etiantian.org/stu.jpg，此时发现被blog.etiantian.com 盗链了。通过查看流量，分析日志看referfer，就可以看到被盗链的具体情

况。

下面是查看Web用户的http_referer的实战演示。

Nginx日志格式为www.etiantian.org，其内容如下：

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';
```

当盗链的网站blog.etiantian.com 访问我们的站点时，记录的日志如下：

```
[root@nginx conf]# tail -f  ../logs/www_access.log
10.0.0.125 - - [10/Mar/2015:

19:

34:

15 +0800] "GET /stu.jpg HTTP/1.1" 200 68080 "http:

//blog.etiantian.com/oldboy.html" "Mozilla/5.0 (

Windows NT 6.1;

 WOW64;

 rv:
```

```
29.0)

 Gecko/20100101 Firefox/29.0" "-"
```

我们自己正常访问用户网站时，站点记录的日志如下：

```
10.0.0.125 - - [10/Mar/2015:

19:

37:

45 +0800] "GET /img/nolink.jpg HTTP/1.1" 304 0 "-" "Mozilla/4.0 (

compatible;

 MSIE 8.0;

 Windows NT 6.1;

 WOW64;

 Trident/4.0;

 SLCC2;

 .NET CLR 2.0.50727;

 .NET CLR 3.5.30729;
```

```
 .NET CLR 3.0.30729;

 Media Center PC 6.0;

 .NET4.0C;

 .NET4.0E;

 InfoPath.3)

" "-"
```

对比可知，盗链的网站多出了一个http_referer的信息，即“http://blog.etiantian.com/oldboy.html ”，表示使用我们网站资源的用户来自于该网站，也就是该站盗链了网站www.etiantian.org 资源。

可在www.etiantian.org 网站下设置防盗链，Nginx的方法如下：

```
#Preventing hot linking of images and other file types
location ~* ^.+\.(

jpg|png|swf|flv|rar|zip)

$ {
    valid_referers none blocked *.etiantian.org etiantian.org;

    if (
```

```
$invalid_referer)

 {
        rewrite ^/ http:

//bbs.etiantian.org/img/nolink.gif;


    }
    root html/www;


}
```

 提示：如果referers不是*.etiantian.org etiantian.org；，给它一个http://bbs.etiantian.org/img/nolink.gif 。

此外，给盗链网站显示的图片域名不要和被盗链的网站域名一样。也就是http://bbs.etiantian.org/img/nolink.gif 里面的bbs.etiantian.org不能是www.etiantian.org。

3）单独搭建一个虚拟机主机bbs.etiantian.org，指定盗链后展示的图片为http://bbs.etiantian.org/img/nolink.gif 。

上面演示的盗链操作步骤汇总如下：

①将www.etiantian.org/stu.jpg 指定被盗链的图片地址。

②将http://bbs.etiantian.org/img/nolink.gif 指定盗链后展示的图片。

③输入非法盗链网站提供给用户的URL，为http://blog.etiantian.com/oldboy.html 。

在没有设置盗链之前，显示www.etiantian.org/stu.jpg ，设置盗链后，会显示http://bbs.etiantian.org/img/nolink.gif 对应的图片，但地址还是盗链的地址。

8.6 Nginx错误页面的优雅显示

8.6.1 生产环境常见的HTTP状态码列表

企业生产环境常见的HTTP状态码列表如表8-2所示。

表8-2 常见的HTTP状态码列表

状态代码及英文描述	代码描述
200 - OK - Standard response for successful HTTP requests.	服务器成功返回网页，这是成功的 http 请求返回的标准状态码
301 - Moved Permanently - This and all future requests should be directed to the given.	永久跳转，所有请求的网页将永久跳转到被设定的新位置，例如：从 etiantian.org 跳转到 www.etiantian.org
403 - Forbidden - forbidden request (matches a deny filter) => HTTP 403 - The request was a legal request, but the server is refusing to respond to it.	禁止访问，这个请求是合法的，但是服务器端因为匹配了预先设置的规则而拒绝响应客户端的请求，此类问题一般为服务器权限配置不当所致
404 - Not Found - The requested resource could not be found but may be available again in the future.	服务器找不到客户端请求的指定页面，可能是客户端请求了服务器不存在的资源所导致的
500 - Internal Server Error - internal error in haproxy => HTTP 500 - A generic error message, given when no more specific message is suitable.	内部服务器错误，服务器遇到了意料不到的情况，不能完成客户的请求。这是一个较为笼统的报错，一般为服务器的设置或内部程序问题所致
502 - Bad Gateway - the server returned an invalid or incomplete response => HTTP 502 - The server was acting as a gateway or proxy and received an invalid response from the upstream server.	坏的网关，一般是代理服务器请求后端服务时，后端服务不可用或没有完成响应网关服务器。一般为代理服务器下面的节点出了问题所致
503 - Service Unavailable - no server was available to handle the request => HTTP 503 - The server is currently unavailable (because it is overloaded or down for maintenance).	服务当前不可用，可能为服务器超载或停机维护所致，或者是代理服务器后面没有可以提供服务的节点
504 - Gateway Timeout - the server failed to reply in time => HTTP 504 - The server was acting as a gateway or proxy and did not receive a timely response from the upstream server.	网关超时，一般是网关代理服务器请求后端服务时，后端服务没有在特定的时间内完成处理请求，一般为服务器过载所致，没有在指定的时间内返回数据给代理服务器

参考资料请见http://oldboy.blog.51cto.com/2561410/716294 。

8.6.2　为什么要配置错误页面优雅显示

在网站的运行过程中，可能因为页面不存在或系统过载等原因，导致网站无法正常响应用户的请求，此时Web服务会返回系统默认的错误码，或者很不友好的页面（如图8-15所示）。

图8-15　访问不到内容时出现的404界面

我们可以将404、403等的错误信息页面重定向到网站首页或其他事先指定的页面，提升网站的用户访问体验。

例如：老男孩的博客所在的网站，在这方面的处理就不是很严谨，给用户不好的页面体验。

范例1：对错误代码403实行本地页面跳转，命令如下：

```
###www
    server {
        listen       80;



        server_name  www.etiantian.org;



        location / {
            root   html/www;



            index  index.html index.htm;



        }
        error_page  403  /403.html;


  #<==当出现


403错误时，会跳转到


403.html页面



    }
```

上面的/403.html是相对于站点根目录html/www的。

范例2：对错误代码404实行本地页面优雅显示，命令如下：

```
    server {
        listen       80;
```

```
        server_name www.oldboyedu.com;


        location / {
            root   html/www;


            index  index.html index.htm;


            error_page  404              /404.html;


        #<==当出现
```

404错误时，会跳转到

404.html页面

```
            access_log  /app/logs/bbs_access.log  commonlog;


        }
}
```

代码中的/404.html是相对于站点根目录html/www的。

范例3：50x页面放到本地单独目录下，进行优雅显示。

```
# redirect server error pages to the static page /50x.html
error_page   500 502 503 504  /50x.html;
```

```
location = /50x.html {
    root   /data0/www/html;


}
```

这里指定单独的站点目录存放到50x.html文件中。

范例4：改变状态码为新的状态码，并显示指定的文件内容，命令如下：

```
error_page 404 =200 /empty.gif;


    server {
        listen       80;


        server_name www.linuxpeixun.com;


        location / {
            root   /data0/www/bbs;


            index  index.html index.htm;


            fastcgi_intercept_errors on;


            error_page  404 =200              /ta.jpg;
```

```
            access_log  /app/logs/bbs_access.log  commonlog;

        }
}
```

范例5：错误状态码URL重定向，命令如下：

```
server {
        listen       80;

        server_name www.oldboyedu.com;

        location / {
            root   html/www;

            index  index.html index.htm;

          error_page   404  http:

//oldboy.blog.51cto.com;

        #<==当出现

404错误时，会跳转到指定的

URL http:

//oldboy.blog.51cto.com页面显示给用户，这个
```

```
URL一般是企业另外的可用地址


            access_log  /app/logs/bbs_access.log  commonlog;


        }
}
```

代码中的/404.html是相对于站点根目录html/www的。

范例6：将错误状态码重定向到一个location，命令如下：

```
location / {
    error_page 404 = @fallback;



}
location @fallback {
    proxy_pass http:


//backend;



}
```

官方说明如下：

```
syntax:


      error_page code ... [=[response]] uri;


default:
```

—

context:

http,

server,

location,

if in location
Defines the URI that will be shown for the specified errors. These directives are ir
Example:

```
error_page   404              /404.html;

error_page   500 502 503 504  /50x.html;
```

Furthermore,

it is possible to change the response code to another using the "=code" syntax,

for example:

```
error_page 404 =200 /empty.gif;
```

If an error response is processed by a proxied server,

```
 or a FastCGI server,

 and the server may return different response codes  (

e.g.,

 200,

 302,

 401 or 404) ,

 it is possible to respond with a returned code:

error_page 404 = /404.php;

It is also possible to use redirects for error processing:

error_page 403         http:

//example.com/forbidden.html;

error_page 404 =301    http:

//example.com/notfound.html;

In this case,
```

```
 the response code 302 is returned to the client. It can only be changed to one of t

301,

 302,

 303 and 307)

.
If there is no need to change URI during internal redirection it is possible to pass



location / {
    error_page 404 = @fallback;



}
location @fallback {
    proxy_pass http:

//backend;



}
If the uri processing led to an error,

 the HTTP status code indicating last occurred problem will be returned.
```

阿里门户网站天猫的Nginx优雅显示配置案例如下：

```
error_page   500 501 502 503 504  http:

//err.tmall.com/error2.html;
```

```
error_page  400 403 404 405 408 410 411 412 413 414 415 http:

//err.tmall.com/error1.html;
```

8.7 Nginx站点目录文件及目录权限优化

1.单机LNMP环境目录权限严格控制措施

为了保证网站不遭受木马入侵，所有站点目录的用户和组都应该为root，所有的目录权限是755；所有的文件权限是644。设置如下：

```
[root@www ~]# ls -l /var/html/blog/|tail -5
-rw-r--r-- 1 root root  7712 5月


   2 2012 wp-mail.php
-rw-r--r-- 1 root root  9916 4月


  27 2012 wp-settings.php
-rw-r--r-- 1 root root 18299 4月


  21 2012 wp-signup.php
-rw-r--r-- 1 root root  3700 1月


   9 2012 wp-trackback.php
-rw-r--r-- 1 root root  2788 2月


  17 2012 xmlrpc.php
```

以上的权限设置可以防止黑客上传木马，以及修改站点文件，但是，合理的网站用户上传的内容也会被拒之门外。那么如何让合法的用户可以上传文件，而又不至于被黑客利用攻击呢？

如果是单机的LNMP环境，站点目录和文件属性设置如下。

先把所有的目录权限设置为755，所有的文件权限设置为644，所用的目录，以及文件用户和组都是root；然后把用户上传资源的目录权限设置为755，将用户和组设置为Nginx服务的用户；最后针对上传资源的目录做资源访问限制（前文已述）。

部分公司所采用的授权方式不是很安全，常见的有如下两种：

·chmod-R 777/sitedir

·chown-R nginx.nginx/sitedir

上述两种授权方法虽然不能说错误，但是没有做到授权最小化，会给网站带来非常大的安全隐患，特别是木马入侵的时候。

在比较好的网站业务架构中，应把资源文件，包括用户上传的图片、附件等服务和程序服务分离，最好把上传程序服务也分离出来，这样就可以从容地按照上文所述进行安全授权了。

2.Nginx企业网站集群超级安全设置

结合Linux权限体系及Nginx大型集群架构进行配置，严格控制针对Nginx目录的访问才能降低网站被入侵的风险。比如，可根据图8-16中的企业集群架构逻辑图和不同角色提供的不同服务来严格控制不同服务器的Nginx目录权限。

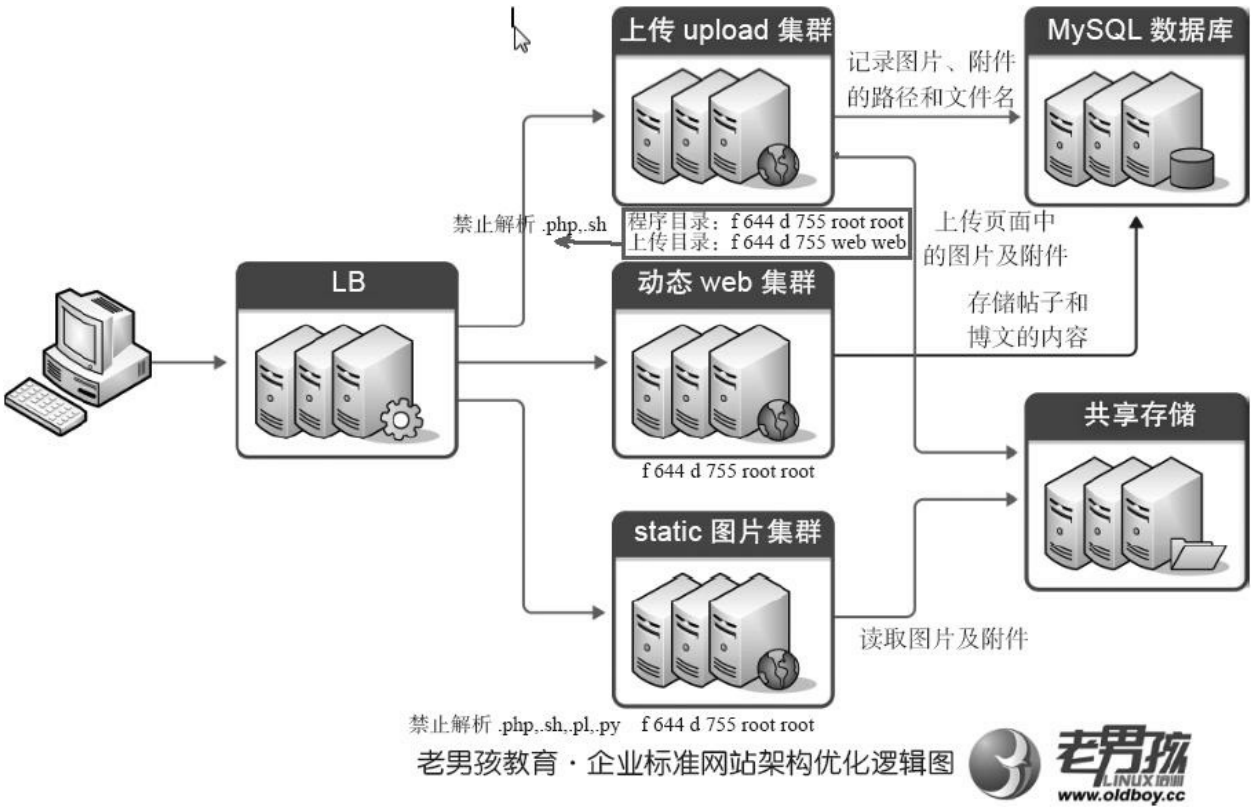


图8-16　专业的Nginx集群架构角色逻辑图

表8-3为集群架构中不同于前面Web业务的权限管理细化。

表8-3　集群架构中不同角色的授权具体思路说明

服务器角色	权限处理	安全系数
动态 Web 集群	目录权限 755，文件权限 644，所用的目录，以及文件用户和组都是 root。环境为 Nginx+PHP	文件不能被改，目录不能被写入，安全系数 10
static 图片集群	目录权限 755，文件权限 644，所用的目录，以及文件用户和组都是 root。环境为 Nginx	文件不能被改，目录不能被写入，安全系数 10
上传 upload 集群	目录权限 755，文件权限 644，所用的目录，以及文件用户和组都是 root。特别：用户上传的目录设置为 755，用户和组使用 Nginx 服务配置的用户	文件不能被改，目录不能被写入，但是用户上传的目录允许写入文件且需要通过 Nginx 的其他功能来禁止读文件，安全系数 8

做到上述的设置后，网站服务在系统层面被入侵的风险就大大降低

了。

8.8 Nginx防爬虫优化

1.robots.txt机器人协议介绍

Robots协议（也称为爬虫协议、机器人协议等）的全称是“网络爬虫排除标准”（Robots Exclusion Protocol），网站通过Robots协议告诉搜索引擎哪些页面可以抓取，哪些页面不能抓取。有关Robots.txt的配置，本书不会详细讲解。

2.机器人协议八卦

2011年10月25日，京东商城正式将一淘网的搜索爬虫屏蔽，以防止一淘网对其内容进行抓取，具体措施如图8-17所示。

2008年9月8日，淘宝网宣布封杀百度爬虫，百度忍痛遵守爬虫协议。因为一旦破坏协议，用户的隐私和利益就无法得到保障，搜索网站就谈不上人性关怀。

淘宝的robots.txt设置如下：

```
http:


//www.taobao.com/robots.txt
User-agent:


 Baiduspider
```

```
Disallow:

 /
User-agent:

 baiduspider
Disallow:

 /
```

图8-18为taobao.com Robots.txt设置情况。

2012年8月，360综合搜索被指违反robots协议（如图8-19所示）。

3.Nginx防爬虫优化配置

我们可以根据客户端的user-agents信息，轻松地阻止指定的爬虫爬取我们的网站。下面来看几个案例。

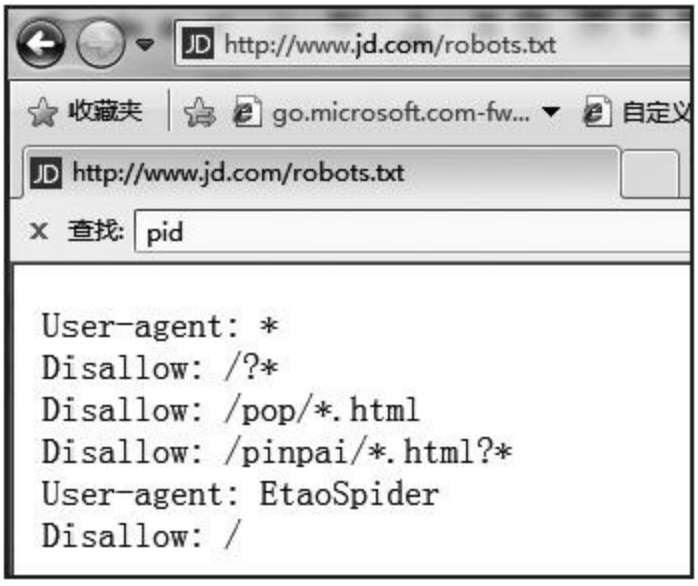


图8-17　jd.com Robots.txt设置情况

图8-18　taobao.com Robots.txt设置情况

图8-19　360.cn Robots.txt设置情况

范例1：阻止下载协议代理，命令如下：

```
## Block download agents ##
if  (


$http_user_agent ~* LWP: :


Simple|BBBike|wget)


 {
    return 403;
```

```
}
```

 说明：如果用户匹配了if后面的客户端（例如wget），就返回403。

这里根据$http_user_agent获取客户端agent，然后判断是否允许或返回指定错误码。

范例2：添加内容防止N多爬虫代理访问网站，命令如下：

这些爬虫代理使用“|”分隔，具体要处理的爬虫可以根据需求增加或减少，添加的内容如下：

```
if (

$http_user_agent ~*
"qihoobot|Baiduspider|Googlebot|Googlebot-Mobile|Googlebot-Image|Mediapartners-Googl

 Slurp China|YoudaoBot|Sosospider|Sogou spider|Sogou web spider|MSNBot")

 {
return 403;

}
```

范例3：测试禁止不同的浏览器软件访问。

示例代码如下：

```
if  (

$http_user_agent ~* "Firefox|MSIE")



{
rewrite ^ (

.*)

 http:

//blog.etiantian.org/$1 permanent;



}
```

如果浏览器为Firefox或IE，就会跳转到http://blog.etiantian.org 。

8.9 利用Nginx限制HTTP的请求方法

在前文的HTTP原理一节，讲解了很多HTTP方法，其中最常用的HTTP方法为GET、POST，我们可以通过Nginx限制HTTP请求的方法来达到提升服务器安全的目的，例如，让HTTP只能使用GET、HEAD和POST方法的配置如下：

```
#Only allow these request methods
    if  (


$request_method !


~ ^ (


GET|HEAD|POST)


$ )


 {
        return 501;


    }
#Do not accept DELETE,


 SEARCH and other methods
```

在前文中，当上传服务器上传数据到存储服务器时，用户上传写入

的目录就不得不给Nginx对应的用户相关权限，这样一旦程序有漏洞，木马就有可能被上传到服务器挂载的对应存储服务器的目录里，虽然我们也做了禁止PHP、SH、PL、PY等扩展名的解析限制，但还是会遗漏一些想不到的可执行文件。对于这种情况，该怎么办呢？事实上，还可以通过限制上传服务器的Web服务（可以具体到文件）使用GET方法，防止用户通过上传服务器访问存储内容，让访问存储渠道只能从静态或图片服务器入口进入。例如，在上传服务器上限制HTTP的GET方法的配置如下：

```
## Only allow GET request methods ##
    if (

$request_method ~* ^(

GET)

$ )

 {
        return 501;

    }
```


提示：还可以加一层location，更具体地限制文件名。

实际效果如图8-20所示。

图8-20　禁止HTTP GET方法的实际访问效果

8.10　使用CDN做网站内容加速

8.10.1　什么是CDN

CDN的全称是Content Delivery Network，中文意思是内容分发网络。简单地讲，通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的Cache服务器内，通过智能DNS负载均衡技术，判断用户的来源，让用户就近使用与服务器相同线路的带宽访问Cache服务器，取得所需的内容。例如：天津网通用户访问天津网通Cache服务器上的内容，北京电信访问北京电信Cache服务器上的内容。这样可以有效减少数据在网络上传输的时间，提高访问速度。

CDN是一套全国或全球的分布式缓存集群，其实质是通过智能DNS判断用户的来源地域及上网线路，为用户选择一个最接近用户地域，以及和用户上网线路相同的服务器节点，因为地域近，且线路相同，所以，可以大幅提升用户浏览网站的体验。

CDN产生背景之一：BGP机房虽然可以提升用户体验，但是价格昂贵，对于用户来说，CDN的诞生可以提供比BGP机房更好的体验（让同一地区、同一线路的用户访问和当地同一线路的网站），BGP机房和普通机房有将近5~10倍的价格差。CDN多使用单线的机房，根据用户的线

路及位置，为用户选择靠近用户的位置，以及相同的运营商线路，不但提升了用户体验，价格也降下来了。

CDN的价值：

·为架设网站的企业省钱。

·提升企业网站的用户访问体验（相同线路、相同地域、内存访问）。

·可以阻挡大部分流量攻击，例如：DDOS攻击。

经常有朋友问老男孩，日100万PV的架构如何设计？对于这样的问题，下面简单为大家解答思路：首先应尽量考虑把网站数据放到CDN中缓存，这样计算在网站的总流量、总访问量后减去CDN的访问流量，剩下的访问量规模需要的架构才是我们需要设计考虑的。一个良好的网站架构设计，访问量尽量都交给CDN。

8.10.2 CDN的特点

CDN就是一个具备根据用户区域和线路智能调度的分布式内存缓存集群。其特点如下：

·通过服务器内存缓存网站数据，提高了企业站点（尤其含有大量图片、视频等的站点）的访问速度，并大大提高企业站点的稳定性（省钱且提升用户体验）。

·用户根据智能DNS技术自动选择最适合的Cache服务器，降低了不同运营商之间互联瓶颈造成的影响，实现了跨运营商的网络加速，保证不同网络中的用户都能得到良好的访问质量。

·加快了访问速度，减少了原站点的带宽。

·用户访问时从服务器的内存中读取数据，分担了网络流量，同时减轻了原站点负载压力等。

·使用CDN可以分担源站的网络流量，同时可以减轻原站点的负载压力，并降低黑客入侵及各种DDOS攻击对网站的影响，保证网站有较好的服务质量。

下面是通过curl命令访问163网站的header信息，可以看到163网站

的首页就使用了CDN进行加速。

```
[oldboy@student~]$curl-Iwww.163.com
HTTP/1.1200OK
Expires:


Sat,


25Feb201202:


58:


57GMT
Date:


Sat,


25Feb201202:


57:


37GMT
Server:


nginx
Content-Type:


text/html;


charset=GBK
Transfer-Encoding:


chunked
Vary:
```

Accept-Encoding
Cache-Control:

max-age=80
Vary:

User-Agent
Vary:

Accept
Age:

60
X-Via:

1.1zb200:

80 (

CdnCacheServerV2 0) ,

1.1jxq206:

8107 (

CdnCacheServerV2.0)

Connection:

keep-alive

上面的“（CdnCacheServerV2.0）”表示163首页使用了CDN加速。

8.10.3　企业使用CDN的基本要求

首先要说的是，不是所有的网站都可以一上来就能用CDN的。要加速的业务数据应该存在独立的域名，例如：img1-4.etiantian.org/video1-4.etiantian.org，业务内容图片、附件、JS、CSS等静态元素，这样的静态网站域名才可以使用CDN。

下面来看一个DNS解析范例。DNS服务器加速前的A记录如下：

```
;

A records
img.etiantian.org        IN  A          124.106.0.21（企业服务器的

IP）
```

删除上面的记录，命令如下：

```
img.etiantian.org        IN  A          124.106.0.21（服务器的

IP）
```

然后，做下面的别名解析。

```
;


 CNAME records
img1.etiantian.org     IN CNAME bbs
img.etiantian.org          3M  IN CNAME     img.etiantian.org.cachecn.com.
```

这个img.etiantian.org.cachecn.com.地址必须是事先由CDN公司配置好的CDN公司的域名。国内较大的CDN提供商为网宿、蓝讯、快网。

8.11 Nginx程序架构优化

解耦是开发人员中流行的一个名词，简单地说就是把一堆程序代码按照业务用途分开，然后提供服务，例如：注册登录、上传、下载、浏览列表、商品内容页面、订单支付等都应该是独立的程序服务，只不过在客户端看来是一个整体而已。如果中小公司做不到上述细致的解耦，起码也要让下面的几个程序模块独立。

·网页页面服务。

·图片附件及下载服务。

·上传图片服务。

上述三者的功能尽量分离。分离的最佳方式是分别使用独立的服务器（需要改动程序），如果程序实在不易更改，次选方案是在前端负载均衡器Haproxy/Nginx上，根据URI（例如目录或扩展名）过滤请求，然后抛给后面对应的服务器。

例如：根据扩展名分发，请求http://www.etiantian.org/a/b.jpg 就应抛给图片服务器（独立的静态服务器最适合使用CDN）；根据URL路径分发，请求http://www.etiantian.org/upload/index.php 就应抛给上传服务器。不符合上面两个要求的，默认抛给Web服务器。


说明：可以部署3台服务器，人为分布请求服务器。当然了，这适合并发比较高、服务器较多的情况。程序架构分离了，效率、安全性都会提高很多。

8.12　使用普通用户启动Nginx（监牢模式）

8.12.1　为什么要让Nginx服务使用普通用户

默认情况下，Nginx的Master进程使用的是root用户，worker进程使用的是Nginx指定的普通用户，使用root用户跑Nginx的Master进程有两个最大的问题：

·管理权限必须是root，这就使得最小化分配权限原则遇到难题。

·使用root跑Nginx服务，一旦网站出现漏洞，用户就可以很容易地获得服务器的root权限。

因此，老男孩想出了一种不用给开发人员，甚至普通运维人员管理员权限，就可以很好地管理Nginx服务的方法，具体内容将在下节为大家讲解。

8.12.2　给Nginx服务降权的解决方案

解决方案如下：

·给Nginx服务降权，用inca用户跑Nginx服务，给开发及运维设置普通账号，只要与inca同组即可管理Nginx，该方案解决了Nginx管理问题，防止root分配权限过大。

·开发人员使用普通账户即可管理Nginx服务及站点下的程序和日志。

·采取项目负责制度，即谁负责项目维护，出了问题就是谁负责。

很多公司开发和运维为root权限争得不可开交，甚至大打出手。

参考资料：到底要不要给开发人员管理服务器的权限？（http://down.51cto.com/data/844517 ）。

8.12.3 给Nginx服务降权实战

本优化属架构优化（同样适合其他软件），通过Nginx启动命令的-c参数指定不同的Nginx配置文件，可以同时启动多个实例，并使用普通的用户运行服务。

Nginx安装后的启动命令路径为“/application/nginx/sbin/nginx”，可以通过加-h参数查看相关参数的用法，命令及结果如下：

```
[root@oldboy ~]# /application/nginx/sbin/nginx -h
nginx version:


 nginx/1.3.4
Usage:


 nginx [-hvVtq] [-s signal] [-c filename] [-p prefix] [-g directives]
Options:




  -,


-h         :


 this help
  -v            :


 show version and exit#<==显示


nginx版本号后退出
```

-V : show version and configure options then exit#<==显示nginx版本号和配置选项后退出

-t : test configuration and exit#<==测试配置文件是否正确，在运行时需要重新加载配置。此命令非常重要

-q : suppress non-error messages during configuration testing

-s signal : send signal to a master process: stop, quit, reopen, reload
#<==发送信号给一个master进程，这里的reload参数很重要，是优雅重启

```
nginx的参数，类似

apache的

graceful参数


  -c filename   :

 set configuration file （

default:

 conf/nginx.conf）

#<==使用指定的配置文件而不是

 conf 目录下的

 nginx.conf这里我们就是通过

nginx -c 路径的功能来实现跑多实例

nginx
```

较常用的方法是使服务跑在指定用户的家目录下面，这样相对比较安全，同时有利于批量业务部署和上线。

配置普通用户启动Nginx的过程如下。

1）添加用户并创建相关目录和文件，操作如下：

```
[root@www ~]# useradd inca
[root@www ~]# su - inca
[inca@www ~]$ pwd
/home/inca
[inca@www ~]$ mkdir conf logs www
[inca@www ~]$ cp /application/nginx/conf/mime.types ~/conf/
[inca@www ~]$ echo inca >www/index.html
```

2）配置Nginx配置文件。配置后的查看命令如下：

```
[inca@www ~]$ cat conf/nginx.conf
worker_processes  4;



worker_cpu_affinity 0001 0010 0100 1000;



worker_rlimit_nofile 65535;



error_log  /home/inca/logs/error.log;



user inca inca;



pid        /home/inca/logs/nginx.pid;



events {
    use epoll;



    worker_connections  10240;
```

```
}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';



    #web.fei fa daolian..............
    server {
        listen       8080;



        server_name  www.etiantian.org;



        root   /home/inca/www;



        location / {
            index  index.php index.html index.htm;



                }
        access_log  /home/inca/logs/web_blog_access.log  main;



            }
}
```

```
 [inca@www ~]$ LANG=en
[inca@www ~]$ tree
.
|-- conf
|   |-- mime.types
|   `-- nginx.conf
|-- logs
`-- www
    `-- index.html
3 directories,


 3 files
```

说明如下：

·所有参数的值，带路径的都要改成/home/inca。

·特权用户root使用的80端口，改为普通用户使用的端口，在1024以上，这里为8080。

3）启动Nginx，命令如下：

```
[inca@www ~]$ ps -ef|grep nginx|grep -v grep
[inca@www ~]$ /application/nginx/sbin/nginx -c /home/inca/conf/nginx.conf &>/dev/nul
[1] 4736
[inca@www ~]$ ps -ef|grep nginx|grep -v grep
inca      4738     1  0 13:


02         00:


00:


00 nginx:


 master process /appli-cation/nginx/sbin/nginx -c /home/inca/conf/nginx.conf #<==主进
```

inca用户了

inca   4739 4738 0 13:

02   00:

00:

00 nginx:

 worker process
inca   4740 4738 0 13:

02   00:

00:

00 nginx:

 worker process
inca   4741 4738 0 13:

02   00:

00:

00 nginx:

 worker process
inca   4742 4738 0 13:

02   00:

```
00:


00 nginx:


 worker process
[inca@www ~]$ ss -lntup|grep nginx
tcp LISTEN 0 128 *:


8080   *:


*      users:  ( (


"nginx",


4738,


5) ,  (


"nginx",


4739,


5) ,  (


"nginx",


4740,


5) ,  (
```

```
"nginx",

4741,

5）, （

"nginx",

4742,

5））

#特别提示：此处启动

nginx，如果不定向到空会显示一些提示，不是错误，可以通过加

&>/dev/null定向到空，从而忽略不见。

[inca@www ~]$ killall nginx
[inca@www ~]$ killall nginx
nginx:

 no process killed
[inca@www ~]$ /application/nginx/sbin/nginx -c /home/inca/conf/nginx.conf
nginx:

 [alert] could not open error log file:

 open（）

 "/application/nginx-1.6.2/logs/error.log" failed （
```

```
13:

 Permission denied)

2015/04/26 13:

06:

44 [warn] 4762#0:

 the "user" directive makes sense only if the master process runs with super-user pr

 ignored in /home/inca/conf/nginx.conf:

5
```

4）查看访问，命令如下：

```
[inca@www ~]$ lsof -i :

8080
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nginx   4764 inca    5u  IPv4 333019      0t0  TCP *:

webcache  (

LISTEN)

nginx   4765 inca    5u  IPv4 333019      0t0  TCP *:
```

```
webcache (
LISTEN)

nginx   4766 inca    5u  IPv4 333019      0t0  TCP *:
webcache (
LISTEN)

nginx   4767 inca    5u  IPv4 333019      0t0  TCP *:
webcache (
LISTEN)

nginx   4768 inca    5u  IPv4 333019      0t0  TCP *:
webcache (
LISTEN)
```

配置好运行解析，浏览器带端口访问结果如图8-21所示。

图8-21　成功展示页面内容

5）解决普通端口非80提供服务的问题。

用负载均衡器解决Web服务非80端口的转换问题，负载均衡器可为Haproxy、Nginx、F5等。

本解决方案的优点如下：

·给Nginx服务降权，让网站更安全。

·按用户设置站点权限，使站点更独立（无需虚拟化隔离）。

·开发不需要用root即可完整管理服务及站点。

·可实现责任划分，即网络问题属于运维的责任，网站打不开就是开发的责任，或者两者共同承担。

8.13 控制Nginx并发连接数量

ngx_http_limit_conn_module这个模块用于限制每个定义的key值的连接数，特别是单IP的连接数。

不是所有的连接数都会被计数。一个符合计数要求的连接是整个请求头已经被读取的连接。

控制Nginx并发连接数量参数的说明如下：

1）limit_conn_zone参数：

语法：limit_conn_zone key zone=name：size；

上下文：http

用于设置共享内存区域，key可以是字符串、Nginx自带变量或前两个组合，如$binary_remote_addr、$server_name。name为内存区域的名称，size为内存区域的大小。

2）limit_conn参数：

语法：limit_conn zone number；

上下文：http、server、location

用于指定key设置最大连接数。当超过最大连接数时，服务器会返回503（Service Temporarily Unavailable）错误。

1.限制单IP并发连接数

Nginx的配置文件如下：

```
[root@oldboy ~]# cat /application/nginx/conf/nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    limit_conn_zone $binary_remote_addr zone=addr:

10m;



    server {
```

```
        listen       80;

        server_name   www.etiantian.org;

        location / {
            root   html;

            index  index.html index.htm;

            limit_conn addr 1;

     #<==限制单

    IP的并发连接为

    1
        }
    }
}
```

在客户端10.0.0.5使用Apache的ab测试工具进行测试。

测试1：模拟并发连接1，访问10次服务器，即执行ab-c 1-n 10 http://10.0.0.3/进行测试。

注意：-c为并发数，-n为请求总数，10.0.0.3为Nginx的IP地址。

测试过程中查看Nginx的访问日志，结果如下：

```
[root@oldboy ~]# tailf /application/nginx/logs/access.log
10.0.0.5 - - [14/Sep/2015:


11:


50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:


50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:


50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:


50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:
```

```
50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:


50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:


50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:


50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:


50:


31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:
```

```
11:

50:

31 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
```

根据上述日志可以看出当并发为1时，返回值都是200，即访问正常。

测试2：模拟并发连接2，访问10次服务器，即执行ab-c 2-n 10 http://10.0.0.3/进行测试。

测试过程中查看Nginx的访问日志，结果如下：

```
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:
```

11:

52:

15 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"

```
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

52:

15 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
```

可以看到状态码200与503间隔1：1出现，即Nginx已经做了并发连接限制，对超过限制的请求返回503。

测试3：模拟并发连接3，访问10次服务器，即执行ab-c3-n 10 http://10.0.0.3/进行测试。

测试过程中查看Nginx的访问日志，结果如下：

```
10.0.0.5 - - [14/Sep/2015:

11:

53:

59 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

53:

59 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

53:

59 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

53:
```

59 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

53:

59 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

53:

59 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

53:

59 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

53:

59 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

11:

```
53:


59 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


11:


53:


59 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
```

由于采用的样本小，所以出现的次数并不均衡。但看其中间数据，200与503出现的次数为1：2，即Nginx已经做了并发连接限制，对超过限制的请求返回503。

以上功能的应用场景之一是用于服务器下载，命令如下：

```
location /download/ {
    limit_conn addr 1;



}
```

上面的命令限制访问download下载目录的连接数，该连接数为1。

2.限制虚拟主机总连接数

不仅可以限制单IP的并发连接数，还可以限制虚拟主机总连接数，

甚至可以对两者同时限制。Nginx的配置文件如下：

```
[root@oldboy ~]# cat /application/nginx/conf/nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    limit_conn_zone $binary_remote_addr zone=addr:


10m;



    limit_conn_zone $server_name zone=perserver:


10m;



    server {
        listen       80;
```

```
        server_name   www.etiantian.org;


        location / {
            root   html;


            index  index.html index.htm;


            #limit_conn addr 1;


            limit_conn perserver 2;

    #<==设置虚拟主机连接数为


2
        }
    }
}
```

提示：Nginx的内部变量列表见官网http://nginx.org/en/docs/varindex.html 。

测试：执行ab-c 5-n 1000 http://10.0.0.3/进行测试，即并发连接数为5，访问1000次。

统计日志中200与503出现的次数（测试前清空日志，测完将日志复制到root家目录）如下：

```
[root@oldboy ~]# grep -c 200 access.log
```

```
634
[root@oldboy ~]# grep -c 503 access.log
366
```

结论：出现的次数近似为2：1。

至此，Nginx限制连接数的应用实践就讲解完毕了。

更多内容可参考：http://nginx.org/en/docs/http/ngx_http_limit_conn_module.html 。

8.14 控制客户端请求Nginx的速率

ngx_http_limit_req_module模块用于限制每个IP访问每个定义key的请求速率。limit_req_zone参数说明如下。

语法：limit_req_zone key zone=name：size rate=rate；

上下文：http

用于设置共享内存区域，key可以是字符串、Nginx自带变量或前两个组合，如$binary_remote_addr。name为内存区域的名称，size为内存区域的大小，rate为速率，单位为r/s，每秒一个请求。

limit_req参数说明如下：

语法：limit_req zone=name[burst=number][nodelay]；

上下文：http、server、location

这里运用了令牌桶原理，burst=num，一共有num块令牌，令牌发完后，多出来的那些请求就会返回503。

换句话说，一个银行，只有一个营业员，银行很小，等候室只有5个人的位置。因此，营业员一个时刻只能为一个人提供服务，剩下的不超过5个人可以在银行内等待，超出的人不提供服务，直接返回503。

nodelay默认在不超过burst值的前提下会排队等待处理，如果使用此参数，就会处理完num+1次请求，剩余的请求都视为超时，返回503。

用于测试的Nginx配置文件如下：

```
[root@oldboy ~]# cat /application/nginx/conf/nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    limit_req_zone $binary_remote_addr zone=one:

10m rate=1r/s;



    #<==以请求的客户端

IP作为
```

key值，内存区域命名为

one，分配

10m内存空间，访问速率限制为

1秒

1次请求（

request）

```
server {
        listen       80;

        server_name   www.etiantian.org;

        location / {
            root   html;

            index  index.html index.htm;

            limit_req zone=one burst=5;

    #<==使用前面定义的名为
```

one的内存空间，队列值为

```
5，即可以有


5个请求排队等待



        }
    }
}
```

测试1：执行ab-c 4-n 1000 http://10.0.0.3/和ab-c 5-n 1000 http://10.0.0.3/进行测试，命令及结果如下：

```
[root@oldboy ~]# tailf /application/nginx/logs/access.log
10.0.0.5 - - [14/Sep/2015:


13:


50:


19 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


13:


50:


20 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


13:


50:
```

```
21 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

50:

22 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

50:

23 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

50:

24 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

50:

25 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3".....
```

可以发现，访问日志中的时间和请求是1秒钟1条请求，证明配置生效。

测试2：执行ab-c 6-n 1000 http://10.0.0.3/进行测试。

统计日志中的200与503出现的次数（测试前已清空日志，测完将日志复制到root家目录）如下：

```
[root@oldboy ~]# grep -c 200 access.log
6
[root@oldboy ~]# grep -c 503 access.log
994
[root@oldboy ~]# more access.log
10.0.0.5 - - [14/Sep/2015:



13:



48:



42 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:



13:



48:



42 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:



13:



48:
```

42 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

48:

42 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

48:

42 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

48:

42 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"……省略若干……

……省略若干……

10.0.0.5 - - [14/Sep/2015:

13:

48:

42 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

48:

42 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

48:

42 +0800] "GET / HTTP/1.0" 503 212 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

48:

43 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

48:

44 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:

13:

```
48:


45 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


13:


48:


46 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
10.0.0.5 - - [14/Sep/2015:


13:


48:


47 +0800] "GET / HTTP/1.0" 200 612 "-" "ApacheBench/2.3"
```

通过过滤排重及部分日志，可以看到第一个请求返回200，为正常处理，剩余的994次超过限制的请求在1秒内执行完成，但是都返回了503。

具体过程原理为：Nginx在第1秒先处理第一个请求，同时接下来的5个请求等待排队，剩下的所有（994次）请求返回503。接着第2秒到第6秒处理等待的5个请求。

更多内容可参考：http://nginx.org/en/docs/http/ngx_http_limit_req_module.html 。

8.15 本章重点回顾

1）安全优化：隐藏Nginx软件名及版本号。

2）性能加安全优化：连接超时参数及FastCGI相关参数调优。

3）性能优化：gzip压缩功能及调试查看方法。

4）性能优化：expires缓存功能及调试查看方法。

5）安全优化：集群中各角色服务站点目录权限控制策略。

6）安全优化：站点目录下所有的文件和目录访问控制。

7）性能加安全优化：robots.txt协议及防爬虫优化解决方案。

8）性能加安全优化：静态资源防盗链解决方案。

9）用户体验优化：错误页面优雅显示方法。

10）安全优化：限制http请求方法。

11）性能加安全优化：CDN加速知识。

12）安全优化：监牢模式运行Nginx方案策略。

13）性能加安全优化：Nginx并发连接数及请求速率控制。

第9章　MySQL数据库企业级应用实践

9.1　概述

9.1.1　MySQL介绍

前面已经介绍过，MySQL属于传统关系型数据库产品，它开放式的架构使得用户选择性很强，同时社区开发与维护人数众多。其功能稳定，性能卓越，且在遵守GPL协议的前提下，可以免费使用与修改，也为MySQL的推广与使用带来了更多的利好。在MySQL成长与发展过程中，支持的功能逐渐增多，性能也不断提高，对平台的支持也越来越多。

MySQL是一种关系型数据库管理系统，关系型数据库的特点是将数据保存在不同的表中，再将这些表放入不同的数据库中，而不是将所有数据统一放在一个大仓库里，这样的设计增加了MySQL的读取速度，而且灵活性和可管理性也得到了很大提高。访问及管理MySQL数据库的最常用标准化语言为SQL结构化查询语言。

9.1.2 MariaDB数据库的诞生背景介绍

自甲骨文公司收购MySQL后，其在商业数据库与开源数据库领域市场的占有份额都跃居第一，这样的格局引起了业内很多的人士的担忧，因为商业数据库的老大有可能将MySQL闭源。为了避免Oracle将MySQL闭源，而无开源的类MySQL数据库可用，MySQL社区采用分支的方式来避开这个风险。MariaDB数据库就这样诞生了，MariaDB是一个向后兼容，可能在以后替代MySQL的数据库产品，其官方地址为https://mariadb.org/。不过，这里还是建议大家选择更稳定、使用更广泛的MySQL数据库，可以先测试MariaDB数据库，等使用的人员更多一些，社区更活跃后再考虑使用为好。

9.2 MySQL多实例介绍

在本书第6章，已经针对MySQL数据库进行了介绍，并说明了为什么要选择MySQL数据库，以及MySQL数据库在Linux系统下的多种安装方式，同时讲解了MySQL的二进制方式单实例安装、基础优化等内容，本章将为大家讲解更为实用的MySQL多实例安装、主从复制集群等重要应用实践。

9.2.1　什么是MySQL多实例

简单地说，MySQL多实例就是在一台服务器上同时开启多个不同的服务器端口（如：3306、3307），同时运行多个MySQL服务进程，这些服务进程通过不同的socket监听不同的服务器端口来提供服务。

这些MySQL多实例共用一套MySQL安装程序，使用不同的my.cnf（也可以相同）配置文件、启动程序（也可以相同）和数据文件。在提供服务时，多实例MySQL在逻辑上看起来是各自独立的，它们根据配置文件的对应设定值，获得服务器相应数量的硬件资源。

打个比方吧，MySQL多实例就相当于房子的多个卧室，每个实例可以看作一间卧室，整个服务器就是一套房子，服务器的硬件资源（CPU、Mem、Disk）、软件资源（Centos操作系统）可以看作房子的卫生间、厨房、客厅，是房子的公用资源。若你是北漂的小伙伴，与朋友一起租房，相信对此能更好地理解。大家蜗居在一起，在自己的卧室休息，出来活动时肯定是要共用上述公共资源的。这样该明白MySQL多实例了吧。

图9-1是MySQL多实例示意图。

其实很多网络服务都是可以配置多实例的，例如Nginx、Apache、

Haproxy、Redis、Memcahe等。这在门户网站使用得很广泛。

9.2.2 MySQL多实例的作用与问题

这一节首先来看看MySQL多实例的作用，具体如下。

·有效利用服务器资源

当单个服务器资源有剩余时，可以充分利用剩余的资源提供更多的服务，且可以实现资源的逻辑隔离。

·节约服务器资源

当公司资金紧张，但是数据库又需要各自尽量独立地提供服务，而且，需要主从复制等技术时，多实例就再好不过了。

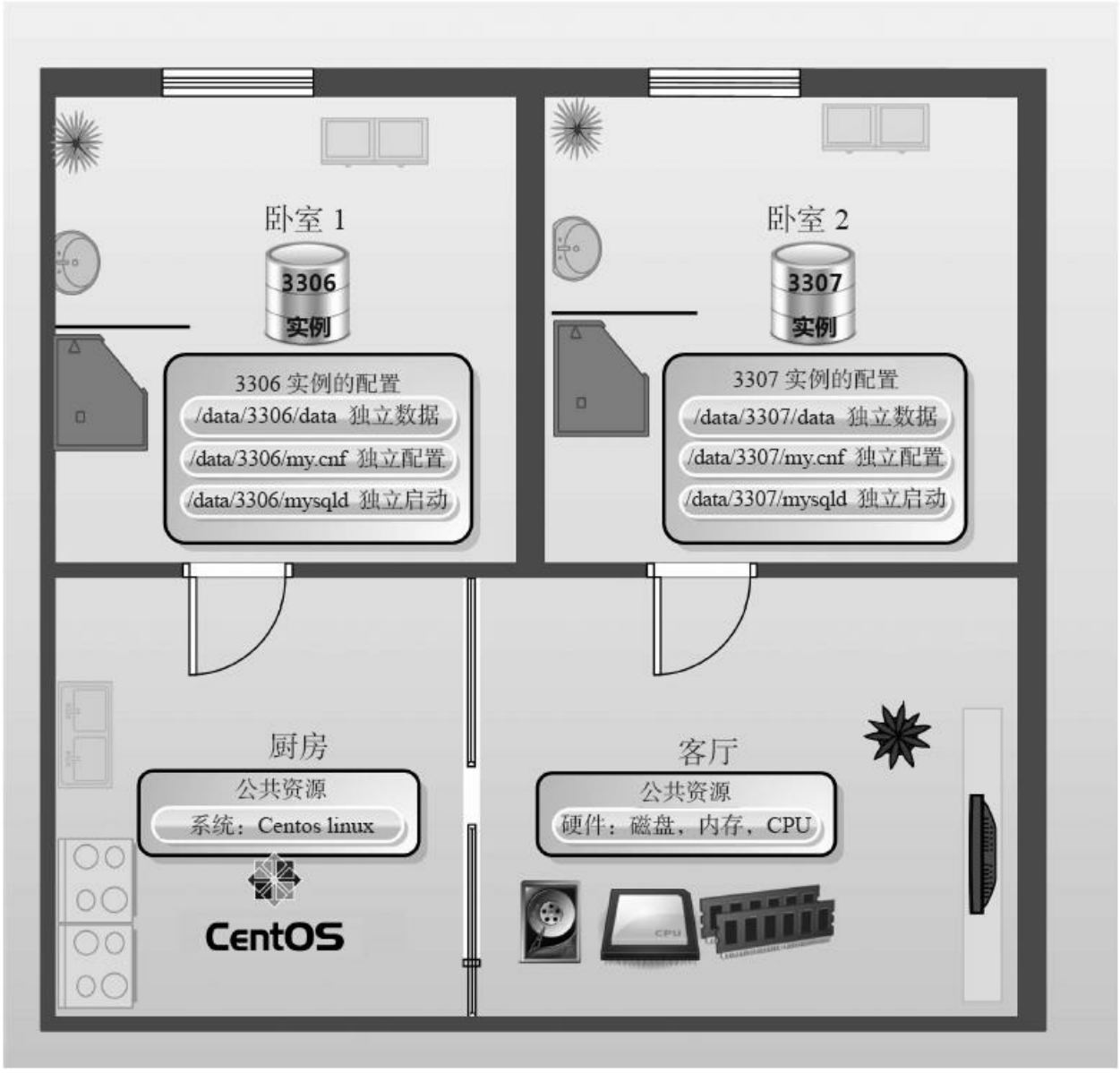


图9-1　MySQL多实例示意图

MySQL多实例有它的好处，但也有其弊端，比如，会存在资源互相抢占的问题。

当某个数据库实例并发很高或有SQL慢查询时，整个实例会消耗大量的系统CPU、磁盘I/O等资源，导致服务器上的其他数据库实例提供

服务的质量一起下降。这就相当于大家住在一个房子的不同卧室一样，早晨起来上班，都要刷牙、洗脸等，这样卫生间就会长期占用，其他人要等待一样。不同实例获取的资源是相对独立的，无法像虚拟化一样完全隔离。

9.3 MySQL多实例的生产应用场景

1.资金紧张型公司的选择

若公司资金紧张，公司业务访问量不太大，但又希望不同业务的数据库服务各自尽量独立地提供服务而互相不受影响，同时，还需要主从复制等技术提供备份或读写分离服务，那么，多实例就再好不过了。例如：可以通过3台服务器部署9~15个实例，交叉做主从复制、数据备份及读写分离，这样就可达到9~15台服务器每个只装一个数据库才有的效果。这里要强调的是，所谓的尽量独立是相对的。

2.并发访问不是特别大的业务

当公司业务访问量不太大的时候，服务器的资源基本上都浪费了，这时就很适合多实例的应用，如果对SQL语句的优化做得比较好，MySQL多实例会是一个很值得使用的技术，即使并发很大，合理分配好系统资源，搭配好服务，也不会有太大问题。

3.门户网站应用MySQL多实例场景

门户网站通常都会使用多实例，因为配置硬件好的服务器，可节省IDC机柜空间，同时，跑多实例也会减少硬件资源跑不满的浪费。比如，百度公司的很多数据库都是多实例，不过，一般是从库多实例，例

如某部门中使用的IBM服务器为48核CPU，内存96GB，一台服务器跑3~4个实例；此外，新浪网使用的也是多实例，内存48GB左右。


说明：据老男孩调查，新浪网的数据库单机1~4个数据库实例的居多，其中又数1~2个的最多，因为大业务占用的机器比较多。服务器是DELL R510的居多，CPU是E5210，48GB内存，磁盘12×300G SAS，做RAID10，此为门户网站的服务器配置参考，希望能给读者购买服务器及部署实例一点数据帮助。

另外，新浪网站安装数据库时，一般采用编译安装的方式，并且会在优化之后做成rpm包，以便统一使用[1]。

[1] 老男孩面授课程中会讲解rpm包定制及yum仓库搭建技术。读者也可以参考老男孩内部的360云盘分享：http://yunpan.cn/cjJ4rWaT8bVRS，提取码为1e89，里面是rpm包制作的内部免费视频与yum仓库搭建速成，以及相关打包知识。

9.4 MySQL多实例常见的配置方案

9.4.1 单一配置文件、单一启动程序的多实例部署方案

下面是MySQL官方文档提到的单一配置文件、单一启动程序多实例部署方案，老男孩不推荐此方案，这里仅作为知识点提及，后文不再涉及此方案的说明。my.cnf配置文件示例（MySQL手册里提到的方法）如下：

```
[mysqld_multi]
mysqld     = /usr/bin/mysqld_safe
mysqladmin = /usr/bin/mysqladmin
user       = mysql
[mysqld1]
socket     = /var/lib/mysql/mysql.sock
port       = 3306
pid-file   = /var/lib/mysql/mysql.pid
datadir    = /var/lib/mysql/
user       = mysql
[mysqld2]
socket     = /mnt/data/db1/mysql.sock
port       = 3302
pid-file   = /mnt/data/db1/mysql.pid
datadir    = /mnt/data/db1/
user       = mysql
skip-name-resolve
server-id=10
default-storage-engine=innodb
innodb_buffer_pool_size=512M
innodb_additional_mem_pool=10M
default_character_set=utf8
character_set_server=utf8
#read-only
relay-log-space-limit=3G
expire_logs_day=20
```

启动程序的命令如下：

```
mysqld_multi --config-file=/data/mysql/my_multi.cnf start 1,

2
```

该方案的缺点是耦合度太高，一个配置文件，不好管理。工作开发和运维的统一原则为降低耦合度。

9.4.2 多配置文件、多启动程序的部署方案

多配置文件、多启动程序部署方案，是本文主要讲解的方案，也是老男孩常用并极力推荐的多实例方案。下面来看配置示例。

以下是老男孩已部署好的MySQL双实例的目录信息及文件注释说明：

```
[root@MySQL /]# tree /data
/data
|-- 3306
|   |-- data    ←

3306实例的数据文件

|   |-- my.cnf ←

3306实例的配置文件

|   `-- mysql   ←

3306实例的启动文件

`-- 3307
    |-- data    ←

3307实例的数据文件
```

```
    |-- my.cnf ←

3307实例的配置文件

    `-- mysql  ←

3307实例的启动文件

4 directories,

 4 files
```

 提示：这里的配置文件my.cnf、启动程序mysql都是独立的文件，数据文件data目录也是独立的。

多实例MySQL数据库的安装和前文讲解的单实例没有任何区别，因此，读者如果有前文单实例的安装环境，那么可以直接略过9.5.1节的内容。

9.5　安装并配置多实例MySQL数据库

9.5.1　安装MySQL多实例

1.安装MySQL需要的依赖包和编译软件

（1）安装MySQL需要的依赖包

安装MySQL之前，最好先安装MySQL需要的依赖包，不然后面会出现很多报错信息，到那时还得再回来安装MySQL的依赖包。安装命令如下：

```
[root@MySQL ~]# yum install ncurses-devel libaio-devel -y
[root@MySQL ~]# rpm -qa ncurses-devel libaio-devel
libaio-devel-0.3.107-10.el6.x86_64
ncurses-devel-5.7-3.20090208.el6.x86_64
```


提示：安装后检查，如果出现两行信息表示安装成功。

（2）安装编译MySQL需要的软件

首先通过网络或老男孩提供的云盘或QQ群下载相关的cmake软件，上传到Linux里，也可找到网络下载地址后直接在Linux里使用wget下载，然后进行如下操作：

```
[root@MySQL tools]# ls -lh cmake-2.8.8.tar.gz
```

```
-rw-r--r-- 1 root root 5.5M 6月

   7 15:

48 cmake-2.8.8.tar.gz      #<==此软件需要提前下载好
```

```
[root@MySQL tools]# tar xf cmake-2.8.8.tar.gz
[root@MySQL tools]# cd cmake-2.8.8
[root@MySQL cmake-2.8.8]# ./configure                #<==无需加任何参数
```

```
[root@MySQL cmake-2.8.8]# gmake
[root@MySQL cmake-2.8.8]# gmake install
[root@MySQL cmake-2.8.8]# which cmake
/usr/local/bin/cmake
[root@MySQL cmake-2.8.8]# cd ../
```

2.开始安装MySQL

为了让大家学习更多的MySQL技术，本文选择了以相对复杂的源代码安装为例来讲解MySQL多实例的安装。大型公司一般都会将MySQL软件定制成rpm包，然后放到yum仓库里，使用yum安装，中小企业里的二进制和编译安装的区别不大。

（1）建立MySQL用户账号

首先以root身份登录到Linux系统中，然后执行如下命令创建mysql用户账号：

```
[root@MySQL ~]# useradd -s /sbin/nologin -M mysql      #<==默认会创建和
```

```
mysql用户同名的组


[root@MySQL ~]# id mysql
uid=500（

mysql）

 gid=500（

mysql）

 组

=500（

mysql）
```

根据上述输出结果，可以看到mysql用户和组已经成功创建。

（2）获取MySQL软件包

MySQL软件包的下载地址为：http://dev.mysql.com/downloads/mysql/ （如果地址变更无法下载，可以去官方下载或通过老男孩提供的云空间下载）。可以把软件下载到客户端计算机本地后使用rz等工具传到Linux里，或者找到网络下载地址后直接在Linux里使用wget下载。

提示：本例以MySQL编译的方式来讲解，前面章节已经带领大家使用二进制方式安装过了。在生产场景中，二进制和源码包两种安装方法都是可以用的，其应用场景一般没什么差别。不同之处在于，二进制的安装包较大，名字和源码包也有些区别，二进制安装过程比源码更快。

MySQL源码包和二进制安装包的名称见表9-1。

表9-1　MySQL二进制和源码包

MySQL 软件	软件名
MySQL 源码安装包	mysql-5.5.32.tar.gz（本章选择的安装包）
MySQL 二进制安装包	mysql-5.5.32-linux2.6-x86_64.tar.gz

（3）采用编译方式安装MySQL

配置及编译安装的步骤如下：

```
[root@MySQL tools]# tar zxf mysql-5.5.32.tar.gz
[root@MySQL tools]# cd mysql-5.5.32
[root@MySQL mysql-5.5.32]# cmake . -DCMAKE_INSTALL_PREFIX=/application/
mysql-5.5.32 \
-DMYSQL_DATADIR=/application/mysql-5.5.32/data \
-DMYSQL_UNIX_ADDR=/application/mysql-5.5.32/tmp/mysql.sock \
-DDEFAULT_CHARSET=utf8 \
-DDEFAULT_COLLATION=utf8_general_ci \
-DEXTRA_CHARSETS=gbk,


gb2312,


utf8,


ascii \
-DENABLED_LOCAL_INFILE=ON \
```

```
-DWITH_INNOBASE_STORAGE_ENGINE=1 \
-DWITH_FEDERATED_STORAGE_ENGINE=1 \
-DWITH_BLACKHOLE_STORAGE_ENGINE=1 \
-DWITHOUT_EXAMPLE_STORAGE_ENGINE=1 \
-DWITHOUT_PARTITION_STORAGE_ENGINE=1 \
-DWITH_FAST_MUTEXES=1 \
-DWITH_ZLIB=bundled \
-DENABLED_LOCAL_INFILE=1 \
-DWITH_READLINE=1 \
-DWITH_EMBEDDED_SERVER=1 \
-DWITH_DEBUG=0
#提示，编译时可配置的选项很多，具体可参考官方文档



[root@MySQL mysql-5.5.32]# make
[root@MySQL mysql-5.5.32]# make install
```

下面为MySQL安装路径设置不带版本号的软链接/application/mysql，操作步骤如下：

```
[root@MySQL mysql-5.5.32]# ln -s /application/mysql-5.5.32/ /application/mysql
#补充：如果系统里有我们曾经讲解过的单实例安装的数据库文件和启动程序，最好停掉或删除，以免产生冲突



[root@MySQL mysql-5.5.32]# ls /application/mysql/
bin      data  include         lib  mysql-test  scripts  sql-bench
COPYING  docs  INSTALL-BINARY  man  README      share    support-files
```

如果上述操作未出现错误，查看/application/mysql/目录下有内容，则MySQL5.5.32源代码包采用cmake方式的安装就算成功了。

9.5.2　创建MySQL多实例的数据文件目录

在老男孩的企业中，通常以/data目录作为MySQL多实例总的根目录，然后规划不同的数字（即MySQL实例端口号）作为/data下面的二级目录，不同的二级目录对应的数字就作为MySQL实例的端口号，以区别不同的实例，数字对应的二级目录下包含MySQL的数据文件、配置文件及启动文件等。

下面以配置3306、3307两个实例为例进行讲解。创建MySQL多实例的目录如下：

```
[root@MySQL ~]# mkdir -p /data/{3306,


3307}/data
[root@MySQL ~]# tree /data/
/data/
|-- 3306        ←


3306实例的目录


|   `-- data    ←


3306实例的数据文件目录


`-- 3307        ←
```

```
3307实例的目录

    `-- data    ←

3307实例的数据文件目录

4 directories,

 0 files
```

提示：

1）mkdir-p/data/{3306，3307}/data相当于mkdir-p/data/3306/data；mkdir-p/data/3307/data两条命令。

2）如果是创建多个目录，可以增加如3308、3309这样的目录名，在生产环境中，一般为3~4个实例为佳。

9.5.3　创建MySQL多实例的配置文件

MySQL数据库默认为用户提供了多个配置文件模板，用户可以根据服务器硬件配置的大小来选择。

```
[root@MySQL mysql-5.5.32]# ls -l support-files/my*.cnf
-rw-r--r-- 1 root root  4759 7月


  20 17:


28 support-files/my-huge.cnf
-rw-r--r-- 1 root root 19809 7月


  20 17:


28 support-files/my-innodb-heavy-4G.cnf
-rw-r--r-- 1 root root  4733 7月


  20 17:


28 support-files/my-large.cnf
-rw-r--r-- 1 root root  4744 7月


  20 17:


28 support-files/my-medium.cnf
-rw-r--r-- 1 root root  2908 7月


  20 17:
```

```
28 support-files/my-small.cnf
```

 注意：

1）关于mysql my.cnf中参数的调优，由于篇幅关系本书不会详细介绍，有需要的读者请参考老男孩教育的相关课程。

2）support-files下有mysql my.cnf的各种配置样例，里面的注释非常详细，不过是英文的。

上面是单实例的默认配置文件模板，如果配置多实例，和单实例会有不同。为了让MySQL多实例之间彼此独立，要为每一个实例建立一个my.cnf配置文件和一个启动文件MySQL，让它们分别对应自己的数据文件目录data。

首先，通过vim命令添加配置文件内容，命令如下：

```
vim /data/3306/my.cnf
vim /data/3307/my.cnf
```

不同的实例需要添加的my.cnf内容会有区别，具体见表9-2，其中的配置由官方的配置模板修改而来。当然，在实际工作中，我们是拿早已配置好的模板来进行修改的，可以通过rz等方式上传配置文件模板my.cnf文件到相关目录下。

表9-2　MySQL 3306、3307实例配置文件对比

MySQL3306 实例	MySQL 3307 实例
[client] port = **3306** socket = /data/**3306**/mysql.sock [mysql] no-auto-rehash [mysqld] user = mysql port = **3306**	[client] port = **3307** socket = /data/**3307**/mysql.sock [mysql] no-auto-rehash [mysqld] user = mysql port = **3307**

（续）

MySQL3306 实例	MySQL 3307 实例
socket = /data/3306/mysql.sock	socket = /data/3307/mysql.sock
basedir = /application/mysql	basedir = /application/mysql
datadir = /data/3306/data	datadir = /data/3307/data
open_files_limit = 1024	open_files_limit = 1024
back_log = 600	back_log = 600
max_connections = 800	max_connections = 800
max_connect_errors = 3000	max_connect_errors = 3000
table_cache = 614	table_cache = 614
external-locking = FALSE	external-locking = FALSE
max_allowed_packet =8M	max_allowed_packet =8M
sort_buffer_size = 1M	sort_buffer_size = 1M
join_buffer_size = 1M	join_buffer_size = 1M
thread_cache_size = 100	thread_cache_size = 100
thread_concurrency = 2	thread_concurrency = 2
query_cache_size = 2M	query_cache_size = 2M
query_cache_limit = 1M	query_cache_limit = 1M
query_cache_min_res_unit = 2k	query_cache_min_res_unit = 2k
#default_table_type = InnoDB	#default_table_type = InnoDB
thread_stack = 192K	thread_stack = 192K
#transaction_isolation = READ-COMMITTED	#transaction_isolation = READ-COMMITTED
tmp_table_size = 2M	tmp_table_size = 2M
max_heap_table_size = 2M	max_heap_table_size = 2M
long_query_time = 1	long_query_time = 1
pid-file = /data/3306/mysql.pid	pid-file = /data/3307/mysql.pid
relay-log = /data/3306/relay-bin	relay-log = /data/3307/relay-bin
relay-log-info-file = /data/3306/relay-log.info	relay-log-info-file = /data/3307/relay-log.info
binlog_cache_size = 1M	binlog_cache_size = 1M
max_binlog_cache_size = 1M	max_binlog_cache_size = 1M
max_binlog_size = 2M	max_binlog_size = 2M
key_buffer_size = 16M	key_buffer_size = 16M
read_buffer_size = 1M	read_buffer_size = 1M
read_rnd_buffer_size = 1M	read_rnd_buffer_size = 1M
bulk_insert_buffer_size = 1M	bulk_insert_buffer_size = 1M
lower_case_table_names = 1	lower_case_table_names = 1
skip-name-resolve	skip-name-resolve
slave-skip-errors = 1032,1062	slave-skip-errors = 1032,1062
replicate-ignore-db=mysql	replicate-ignore-db=mysql
server-id = 1	server-id = 3
innodb_additional_mem_pool_size = 4M	innodb_additional_mem_pool_size = 4M
innodb_buffer_pool_size = 32M	innodb_buffer_pool_size = 32M

（续）

MySQL3306 实例	MySQL 3307 实例
innodb_data_file_path = ibdata1: 128M: autoextend innodb_file_io_threads = 4 innodb_thread_concurrency = 8 innodb_flush_log_at_trx_commit = 2 innodb_log_buffer_size = 2M innodb_log_file_size = 4M innodb_log_files_in_group = 3 innodb_max_dirty_pages_pct = 90 innodb_lock_wait_timeout = 120 innodb_file_per_table = 0 [mysqldump] quick max_allowed_packet = 2M [mysqld_safe] log-error=/data/3306/mysql_oldboy3306.err pid-file=/data/3306/mysqld.pid	innodb_data_file_path = ibdata1:128M: autoextend innodb_file_io_threads = 4 innodb_thread_concurrency = 8 innodb_flush_log_at_trx_commit = 2 innodb_log_buffer_size = 2M innodb_log_file_size = 4M innodb_log_files_in_group = 3 innodb_max_dirty_pages_pct = 90 innodb_lock_wait_timeout = 120 innodb_file_per_table = 0 [mysqldump] quick max_allowed_packet = 2M [mysqld_safe] log-error=/data/3307/mysql_oldboy3307.err pid-file=/data/3307/mysqld.pid

最终完成后的多实例根/data目录结果如下：

```
[root@MySQL /]# tree /data
/data
|-- 3306
|   |-- data
|   `-- my.cnf#这个就是


3306实例的配置文件



`-- 3307
    |-- data
    `-- my.cnf#这个就是


3307实例的配置文件



4 directories,
```

```
2 files
```

有关配置文件的参数说明，可以根据my-innodb-heavy-4G.cnf或查找相关资料，进行完整注释。

9.5.4　创建MySQL多实例的启动文件

MySQL多实例启动文件的创建和配置文件的创建几乎一样，也可以通过vim命令来添加，如下：

```
vim /data/3306/mysql
vim /data/3307/mysql
```

需要添加的MySQL启动文件内容见表9-3。当然，在实际工作中我们是拿早已配置好的模板来进行修改的，可以通过rz等方式上传配置文件模板MySQL文件到相关目录下。

表9-3　3306与3307 MySQL实例启动文件对比

3306 MySQL 实例启动文件

```
#!/bin/sh
##########################################
#this scripts is created by oldboy at
2007-06-09
#site:http://www.etiantian.org
#blog:http://oldboy.blog.51cto.com
##########################################
#init
port=3306
mysql_user="root"
mysql_pwd="oldboy123" #<== 这里将来要修改为
和数据库密码一致。
CmdPath="/application/mysql/bin"
mysql_sock="/data/${port}/mysql.sock"
#startup function
function_start_mysql()
{
    if [ ! -e "$mysql_sock" ];then
      printf "Starting MySQL...\n"
        /bin/sh ${CmdPath}/mysqld_safe
--defaults-file=/data/${port}/my.cnf 2>&1 >
/dev/null &
    else
      printf "MySQL is running...\n"
      exit
    fi
}

#stop function
function_stop_mysql()
{
    if [ ! -e "$mysql_sock" ];then
       printf "MySQL is stopped...\n"
       exit
    else
       printf "Stoping MySQL...\n"
       ${CmdPath}/mysqladmin -u ${mysql_
user} -p${mysql_pwd} -S /data/${port}/
mysql.sock shutdown
    fi
}
```

3307 MySQL 实例启动文件

```
#!/bin/sh
##########################################
#this scripts is created by oldboy at
2007-06-09
#site:http://www.etiantian.org
#blog:http://oldboy.blog.51cto.com
##########################################
#init
port=3307
mysql_user="root"
mysql_pwd="oldboy123" #<== 这里将来要修改为
和数据库密码一致。
CmdPath="/application/mysql/bin"
mysql_sock="/data/${port}/mysql.sock"
#startup function
function_start_mysql()
{
    if [ ! -e "$mysql_sock" ];then
      printf "Starting MySQL...\n"
        /bin/sh ${CmdPath}/mysqld_safe
--defaults-file=/data/${port}/my.cnf 2>&1 >
/dev/null &
    else
      printf "MySQL is running...\n"
      exit
    fi
}

#stop function
function_stop_mysql()
{
    if [ ! -e "$mysql_sock" ];then
       printf "MySQL is stopped...\n"
       exit
    else
       printf "Stoping MySQL...\n"
       ${CmdPath}/mysqladmin -u ${mysql_
user} -p${mysql_pwd} -S /data/${port}/
mysql.sock shutdown
    fi
}
```

（续）

3306 MySQL 实例启动文件	3307 MySQL 实例启动文件

#restart function
function_restart_mysql()
{
printf "Restarting MySQL...\n"
function_stop_mysql
sleep 2
function_start_mysql
}
 	
case $1 in
start)
function_start_mysql
;;
stop)
function_stop_mysql
;;
restart)
function_restart_mysql
;;
*)
printf "Usage: /data/${port}/mysql {start|stop|restart}\n"
esac
	#restart function
	function_restart_mysql()
	{
	printf "Restarting MySQL...\n"
	function_stop_mysql
	sleep 2
	function_start_mysql
	}
	
	case $1 in
	start)
	function_start_mysql
	;;
	stop)
	function_stop_mysql
	;;
	restart)
	function_restart_mysql
	;;
	*)
	printf "Usage: /data/${port}/mysql {start|stop|restart}\n"
	esac

最终完成后的多实例根/data目录结果如下：

```
[root@MySQL ~]# tree /data
/data
|-- 3306
|   |-- my.cnf  #<==3306实例的配置文件


|   `-- mysql   #<==3306实例的启动文件


`-- 3307
    |-- my.cnf  #<==3307实例的配置文件


    `-- mysql   #<==3307实例的启动文件
```

```
2 directories,

 4 files
```

需要特别说明一下，在多实例启动文件中，启动MySQL不同实例服务，所执行的命令实质是有区别的，例如，启动3306实例的命令如下：

```
mysqld_safe --defaults-file=/data/3306/my.cnf 2>&1 > /dev/null &
```

启动3307实例的命令如下：

```
mysqld_safe --defaults-file=/data/3307/my.cnf 2>&1 > /dev/null &
```

下面看看在多实例启动文件中，停止MySQL不同实例服务的实质命令。

停止3306实例的命令如下：

```
mysqladmin -u root -poldboy123 -S /data/3306/mysql.sock shutdown
```

停止3307实例的命令如下：

```
mysqladmin -u root -poldboy123 -S /data/3307/mysql.sock shutdown
```

9.5.5　配置MySQL多实例的文件权限

1）通过下面的命令，授权mysql用户和组管理整个多实例的根目录/data。

```
[root@MySQL /]# chown -R mysql.mysql /data
[root@MySQL /]# find /data -name mysql|xargs ls -l
-rw-r--r-- 1 mysql mysql 1307 Jul 15  2013 /data/3306/mysql
-rw-r--r-- 1 mysql mysql 1307 Jul 21  2013 /data/3307/mysql
```

2）通过下面的命令，授权MySQL多实例所有启动文件的mysql可执行，设置700权限最佳，注意不要用755权限，因为启动文件里有数据库管理员密码，会被读取到。

```
[root@MySQL /]# find /data -name mysql|xargs chmod 700
```

检查上述命令的处理结果，看其权限处理是否完成，如下：

```
[root@MySQL /]# find /data -name mysql -exec ls -l {} \;


-rwx------ 1 mysql mysql 1307 Jul 21  2013 /data/3307/mysql
-rwx------ 1 mysql mysql 1307 Jul 15  2013 /data/3306/mysql
```

从输出看，权限已调整完毕，这里是引导读者在学习中使用记忆命令，其结果相当于chmod 700/data/3306/mysql，chmod 700/data/3307/mysql的命令组合。

9.5.6　MySQL相关命令加入全局路径的配置

1.配置全局路径意义

如果不为MySQL的命令配置全局路径，就无法直接在命令行输入mysql这样的命令，只能用全路径命令（/application/mysql/bin/mysql），这种带着路径输入命令的方式很麻烦。

2.配置MySQL全局路径的方法

1）确认mysql命令所在路径，命令如下：

```
[root@MySQL /]# ls /application/mysql/bin/mysql
/application/mysql/bin/mysql
```

2）在PATH变量前面增加/application/mysql/bin路径，并追加到/etc/profile文件中，命令如下：

```
[root@MySQL /]# echo 'export PATH=/application/mysql/bin:


$PATH' >>/etc/profile
#←注意，


echo后是单引号呦，双引号是不行的



[root@MySQL /]# tail -1 /etc/profile
export PATH=/application/mysql/bin:
```

$PATH
[root@MySQL /]# source /etc/profile
#←执行

source使上一行添加到

/etc/profile中，内容直接生效

#←以上命令的用途为定义

mysql全局路径，实现在任意路径执行

mysql命令

[root@MySQL /]# echo $PATH
/application/mysql/bin:

/usr/local/sbin:

/usr/local/bin:

/sbin:

/bin:

/usr/sbin:

/usr/bin:

/root/bin

```
#←执行

echo $PATH有

/application/mysql/bin输出表示配置成功。
```

 提示：更简单的设置方法为用下面命令做软链接：ln-s/application/mysql/bin/*/usr/local/sbin/，把mysql命令所在路径链接到全局路径/usr/local/sbin/的下面。

3.因MySQL环境变量配置顺序导致的错误案例

特别强调：务必把MySQL命令路径放在PATH路径中其他路径的前面，否则，可能会导致使用的mysql命令和编译安装的mysql命令不是同一个，进而产生错误。下面给出了MySQL路径配置问题导致的错误案例：http://oldboy.blog.51cto.com/2561410/1122867 ，此案例就是使用yum安装的MySQL客户端命令访问了编译安装的服务器端，从而出现问题。

9.5.7　初始化MySQL多实例的数据库文件

上述步骤全都配置完毕后，就可以初始化数据库文件了，这个步骤其实也可以在编译安装MySQL之后就操作，只不过放到这里更合理一些。

（1）初始化MySQL数据库

初始化命令如下：

```
cd /application/mysql/scripts  #←注意和

MySQL 5.1的路径不同，

MySQL 5.1不在

MySQL bin路径下了

./mysql_install_db --basedir=/application/mysql --datadir=/data/3306/data --user=mys
./mysql_install_db --basedir=/application/mysql --datadir=/data/3307/data  --user=my
```


提示：--basedir=/application/mysql为MySQL的安装路径，--datadir为不同的实例数据目录。

操作过程如下：

```
[root@MySQL /]# cd /application/mysql/scripts
[root@MySQL scripts]# ./mysql_install_db --basedir=/application/mysql --datadir=/dat
```

MySQL数据库文件，会有很多信息提示，如果没有

ERROR级别的错误，有两个

OK的字样，表示初始化成功，否则就要解决初始化的问题

```
Installing MySQL system tables...
OK  #<==如果此处有两个
```

OK，就表示初始化成功

```
Filling help tables...
OK  #<==如果此处有两个
```

OK，就表示初始化成功

```
To start mysqld at boot time you have to copy
support-files/mysql.server to the right place for your system
PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
```

To do so，

start the server，

then issue the following commands：

```
/application/mysql/bin/mysqladmin -u root password 'new-password'
/application/mysql/bin/mysqladmin -u root -h MySQL password 'new-password'
Alternatively you can run：
```

```
/application/mysql/bin/mysql_secure_installation
which will also give you the option of removing the test
databases and anonymous user created by default.  This is
strongly recommended for production servers.
See the manual for more instructions.
You can start the MySQL daemon with:



cd /application/mysql ;



 /application/mysql/bin/mysqld_safe &
You can test the MySQL daemon with mysql-test-run.pl
cd /application/mysql/mysql-test ;



 perl mysql-test-run.pl
Please report any problems with the /application/mysql/scripts/mysqlbug script!



[root@MySQL scripts]# ./mysql_install_db --basedir=/application/mysql --datadir=/dat
WARNING:



 The host 'MySQL' could not be looked up with resolveip.
This probably means that your libc libraries are not 100 % compatible
with this binary MySQL version. The MySQL daemon,



 mysqld,



 should work
normally with the exception that host name resolving will not work.
This means that you should use IP addresses instead of hostnames
when specifying MySQL privileges !



Installing MySQL system tables...
OK #<==如果此处有两个



OK，就表示初始化成功
```

```
Filling help tables...
OK #<==如果此处有两个


OK，就表示初始化成功


...省略若干行...
```

（2）初始化数据库的原理及结果说明

初始化数据库的实质就是创建基础的数据库系统的库文件，例如：生成MySQL库表等。

初始化数据库后查看对应实例的数据目录，可以看到多了如下文件：

```
[root@MySQL scripts]# tree /data
/data
|-- 3306
|   |-- data
|   |   |-- mysql
|   |   |   |-- columns_priv.MYD
|   |   |   |-- columns_priv.MYI
|   |   |   |-- columns_priv.frm
|   |   |   |-- db.MYD
|   |   |   |-- db.MYI
|   |   |   |-- db.frm...省略部分...
```

（3）初始化故障排错集锦

示例1：给出了警告信息“WARNING：The host'MySQL'could not be

looked up with resolveip.”

警告是可以忽略的，如果非要解决则需修改对主机名的解析，使其和uname-n命令的结果一样，如下：

```
[root@MySQL scripts]# grep `uname -n` /etc/hosts
127.0.0.1               MySQL localhost.localdomain localhost
```

示例2：给出错误提示“ERROR：1004 Can't create file'/tmp/#sql300e_1_0.frm'（errno：13）”。

在执行初始化数据库命令时可能就会遇到这样的错误，错误提示如下：

```
ERROR:

 1004  Can't create file '/tmp/#sql300e_1_0.frm'  (

errno:

 13)


120406 15:

47:

02 [ERROR] Aborting
120406 15:
```

```
47:


02 [Note] /application/mysql/libexec/mysqld:


 Shutdown complete
Installation of system tables failed!


  Examine the logs in
/application/mysql/data for more information.
```

根据提示可知，/tmp目录不能创建文件，所以解决办法为给/tmp目录增加权限，如下：

```
[root@MySQL scripts]# ls -ld /tmp
drwxrwxrwt. 4 root root 4096 Apr 22 16:


43 /tmp
[root@MySQL scripts]# chmod -R 1777 /tmp/ #<==1777是


/tmp的默认权限，改动过此目录权限会报错
```

本示例的故障必须要解除，否则，后面会出现登录不了MySQL数据库等各种问题。

9.5.8　启动MySQL多实例数据库

下面来看看启动MySQL多实例的命令。

第一个实例3306的启动命令如下：

```
[root@MySQL ~]# /data/3306/mysql start
Starting MySQL...
```

第二个实例3307的启动命令如下：

```
[root@MySQL ~]# /data/3307/mysql start
Starting MySQL...
```

现在，检查MySQL多实例数据库是否成功启动，命令及结果如下：

```
 [root@MySQL ~]# netstat -lntup|grep 330
tcp    0    0 0.0.0.0:


3306        0.0.0.0:


*      LISTEN      18058/mysqld
tcp    0    0 0.0.0.0:


3307        0.0.0.0:


*      LISTEN      18777/mysqld
```

从输出中可以看到，3306和3307实例均已正常启动。

9.5.9 MySQL多实例启动故障排错说明

如果MySQL多实例有服务没有启动，排查办法如下：

·如果发现没有显示MySQL对应实例的端口，请稍微等待几秒再检查，MySQL服务的启动比Web服务慢一些。

·如果还不行，请查看MySQL服务对应实例的错误日志，错误日志路径在my.cnf配置的最下面定义。例如，3306实例的错误日志为：

```
[root@MySQL /]# grep log-error my.cnf|tail -1
log-error=/data/3306/mysql_oldboy3306.err
```

那么，可以执行tail-100/data/3306/mysql_oldboy3306.err检查MySQL错误日志，如下：

```
[root@MySQL /]# tail -100 /data/3306/mysql_oldboy3306.err
120412 15:


23:


33 mysqld_safe Starting mysqld daemon with databases from /data/3306/data
120412 15:


23:


33 [Warning] '--log-long-format' is deprecated and will be
removed in a future release. Please use '--log-short-format' instead.
120412 15:
```

23:

33 [Warning] '--log_slow_queries' is deprecated and will be removed in a future rele
120412 15:

23:

33 [Warning] --myisam_max_extra_sort_file_size is deprecated and does nothing in thi
/usr/local/mysql/libexec/mysqld:

Can't find file:

'./mysql/plugin.frm' (

errno:

13)

120412 15:

23:

33 [ERROR] Can't open the mysql.plugin table. Please run mysql_upgrade to create it.
120412 15:

23:

34 InnoDB:

Initializing buffer pool,

```
 size = 32.0M
120412 15:


23:


34  InnoDB:


 Completed initialization of buffer pool
120412 15:


23:


34  InnoDB:


 Operating system error number 13 in a file operation.
InnoDB:


 The error means mysqld does not have the access rights to
InnoDB:


 the directory.
InnoDB:


 File name ./ibdata1
InnoDB:


 File operation call:


 'create'.
InnoDB:


 Cannot continue operation.
120412 15:
```

```
23:

34 mysqld_safe mysqld from pid file /data/3306/mysqld.pid ended
```

·细看所有执行命令返回的屏幕输出，不要忽略关键的输出内容。

·辅助查看系统日志/var/log/messages。

·如果MySQL关联了其他服务，要同时查看相关服务的日志。

·仔细阅读，重新查看操作的步骤是否正确，书写的命令及字符是否正确。

经常查看各种服务运行日志是一个很好的习惯，也是成长为高手的必经之路。

9.6　配置及管理MySQL多实例数据库

1.配置MySQL多实例数据库开机自启动

服务的开机自启动很关键，MySQL多实例的启动也不例外，把MySQL多实例的启动命令加入/etc/rc.local，实现开机自启动，命令如下：

```
[root@MySQL ~]# echo "#mysql multi instances" >>/etc/rc.local
[root@MySQL ~]# echo "/data/3306/mysql start" >>/etc/rc.local
[root@MySQL ~]# echo "/data/3307/mysql start" >>/etc/rc.local
[root@MySQL ~]# tail -3 /etc/rc.local
#mysql multi instances
/data/3306/mysql start
/data/3307/mysql start
```

提示：要确保MySQL脚本可执行呦！

2.登录MySQL测试

测试命令如下：

```
[root@MySQL ~]# mysql -S /data/3306/mysql.sock
#←直接敲就进来了，而且身份还是


root。但是多了


-S /data/3306/mysql.sock，用于区别登录不同的实例


Welcome to the MySQL monitor.  Commands end with ;
```

```
 or \g.
Your MySQL connection id is 1
Server version:


 5.5.32-log Source distribution
Copyright (


c)


 2000,


 2013,


 Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;


' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
mysql> show databases;


      #←查看当前所有的数据库



+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (


0.00 sec)
```

```
mysql> select user();

     #←查看当前的登录用户

+----------------+
| user()

         |
+----------------+
| root@localhost |
+----------------+
1 row in set  (

0.00 sec)

mysql>
```

3.MySQL多实例数据库的管理方法

MySQL安装完成后，默认情况下，MySQL管理员的账号root是无密码的。登录不同的实例需要指定不同实例的mysql.sock文件路径，这个mysql.sock是在my.cnf配置文件里指定的。

下面是无密码情况下登录数据库的方法，关键点是-S参数及后面指定的/data/3306/mysql.sock，注意，不同实例的sock虽然名字相同，但是路径是不同的，因此是不同的文件。

```
mysql -S /data/3306/mysql.sock
mysql -S /data/3307/mysql.sock
```

下面是重启对应实例数据库的命令。

```
/data/3306/mysql stop
/data/3306/mysql start
```

4.MySQL安全配置

MySQL管理员的账号root密码默认为空，极不安全，可以通过mysqladmin命令为MySQL不同实例的数据库设置独立的密码，命令如下：

```
[root@MySQL ~]# mysqladmin -u root -S /data/3306/mysql.sock password 'oldboy123'
#←为

mysql设置密码

[root@MySQL ~]# mysql -uroot -p -S /data/3306/mysql.sock     #←无法直接登录了

Enter password:

ERROR 1045 （

28000）:

 Access denied for user 'root'@'localhost' （

using password:
```

```
 YES)


[root@MySQL ~]# mysql -uroot -p -S /data/3306/mysql.sock      #←新的登录方式


Enter password:


                                    #←输入新密码


oldboy123
Welcome to the MySQL monitor.  Commands end with ;


 or \g.
Your MySQL connection id is 5
Server version:


 5.5.32-log Source distribution...省略若干...


mysql>
```

 提示：3307实例的设置方法和3306实例的相同，只是连接时的mysql、sock路径不同而已，这已提醒多次，大家要注意。

下面介绍带密码登录不同实例数据库的方法。

登录3306实例的命令如下：

```
mysql -uroot -poldboy123 -S /data/3306/mysql.sock
```

登录3307实例的命令如下：

```
mysql -uroot -poldboy123 -S /data/3307/mysql.sock
```

 提示：基础弱的读者，在测试时尽量保证多实例的密码相同，可以减少麻烦，后面还原数据库时会覆盖密码！

若要重启多实例数据库，也需要进行相应的如下配置。在重启数据库前，需要调整不同实例启动文件里对应的数据库密码。

```
[root@MySQL ~]# sed -n '13p' /data/3306/mysql
mysql_pwd="oldboy" #<==这是登录数据库时使用的密码，是前面启动文件


mysql里的默认定义的配置



[root@MySQL ~]# sed -i '13 s#oldboy#oldboy123#g' /data/3306/mysql  /data/3307/mysql
[root@MySQL ~]# sed -n '13p' /data/3306/mysql
mysql_pwd="oldboy123"
[root@MySQL ~]# sed -n '13p' /data/3307/mysql
mysql_pwd="oldboy123"
```

如果当前的MySQL启动文件权限为755或644，则需要降低MySQL启动文件的权限，因为有密码，被其他用户看到会很不安全，批量调整权限及属主的命令如下：

```
[root@MySQL ~]# find /data -type f -name "mysql" -exec chmod 700 {} \;


[root@MySQL ~]# find /data -type f -name "mysql" -exec chown root.root {} \;


[root@MySQL ~]# find /data -type f -name "mysql" -exec ls -l {} \;
```

```
-rwx------ 1 root root 1307 Jul 21  2013 /data/3307/mysql
-rwx------ 1 root root 1335 Jul 20 20:


57 /data/3306/mysql
```

多实例下正常停止数据库的命令如下：

```
/data/3306/mysql stop
#<==由于选择了


mysqladmin shutdown的停止方式，所以停止数据库时需要在启动文件里配置数据库的密码



/data/3306/mysql start
```

 提示：禁止使用pkill、kill-9、killall-9等命令强制杀死数据库，这会引起数据库无法启动等故障的发生。企业血的教训案例请看http://oldboy.blog.51cto.com/2561410/1431161 。

5.如何再增加一个MySQL的实例

若在3306和3307实例的基础上，再增加一个MySQL实例，该怎么办？下面给出增加一个MySQL 3308端口实例的命令集合：

```
mkdir -p /data/3308/data
\cp /data/3306/my.cnf  /data/3308/
\cp /data/3306/mysql  /data/3308/
sed -i 's/3306/3308/g' /data/3308/my.cnf
sed -i 's/server-id = 1/server-id = 8/g' /data/3308/my.cnf
sed -i 's/3306/3308/g' /data/3308/mysql
chown -R mysql:
```

```
mysql /data/3308
chmod 700 /data/3308/mysql
cd /application/mysql/scripts
./mysql_install_db --datadir=/data/3308/data --basedir=/application/mysql --user=mys
chown -R mysql:



mysql /data/3308
egrep "server-id|log-bin" /data/3308/my.cnf
/data/3308/mysql start
sleep 5
netstat -lnt|grep 3308
#提示：最好把



server-id按照



IP地址最后一个小数点的数字设置



#成功标志：多了一个启动的端口



3308
[root@MySQL scripts]# netstat -lnt|grep 330
tcp        0      0 0.0.0.0:



3306                0.0.0.0:



*                   LISTEN
tcp        0      0 0.0.0.0:



3307                0.0.0.0:



*                   LISTEN
tcp        0      0 0.0.0.0:



3308                0.0.0.0:



*                   LISTEN
```

如果配置以后，服务启动后却没有运行起来，别忘了一定要看MySQL错误日志呦，在/data/3308/my.cnf最下面有错误日志路径地址。本例作为作业留给读者自己去实践，看看能否独自搞定这个实例。

6.多实例MySQL登录问题分析

（1）多实例本地登录MySQL

多实例本地登录一般通过socket文件来指定具体登录到哪个实例，此文件的具体位置是在MySQL编译过程或my.cnf文件中指定的。在本地登录数据库时，登录程序会通过socket文件来判断登录的是哪个数据库实例。

例如：通过mysql-uroot-p'oldboy123'-S/data/3307/mysql.sock可知，登录的是3307这个实例。mysql.sock文件是MySQL服务器端与本地MySQL客户端进行通信的UNIX套接字文件。

（2）远程连接登录MySQL多实例

远程登录MySQL多实例中的一个实例时，通过TCP端口（port）来指定所要登录的MySQL实例，此端口的配置是在MySQL配置文件my.cnf中指定的。

例如：在mysql -uoldboy -p'oldboy' -h 10.0.0.7 -P 3307中，-P为端口

参数，后面接具体的实例端口，端口是一种“逻辑连接位置”，是客户端程序被分派到计算机上特殊服务程序的一种方式，强调提前在10.0.0.7上对oldboy用户做了授权。

9.7 MySQL主从复制介绍

MySQL数据库的主从复制方案，与使用scp/rsync等命令进行的文件级别复制类似，都是数据的远程传输，只不过MySQL的主从复制是其自带的功能，无需借助第三方工具，而且，MySQL的主从复制并不是数据库磁盘上的文件直接拷贝，而是通过逻辑的binlog日志复制到要同步的服务器本地，然后由本地的线程读取日志里面的SQL语句，重新应用到MySQL数据库中。

9.7.1 概述

MySQL数据库支持单向、双向、链式级联、环状等不同业务场景的复制。在复制过程中，一台服务器充当主服务器（Master），接收来自用户的内容更新，而一个或多个其他的服务器充当从服务器（Slave），接收来自主服务器binlog文件的日志内容，解析出SQL，重新更新到从服务器，使得主从服务器数据达到一致。

如果设置了链式级联复制，那么，从服务器（Slave）本身除了充当从服务器外，也会同时充当其下面从服务器的主服务器。链式级联复制类似A→B→C的复制形式。

图9-2和图9-3为单向主从复制架构逻辑图，此架构只能在Master端进行数据写入。

图9-2　一主一从逻辑图

图9-4为双向主主复制逻辑架构图，此架构可以在Master1端或Master2端进行数据写入，或者两端同时写入数据（需要特殊设置）。

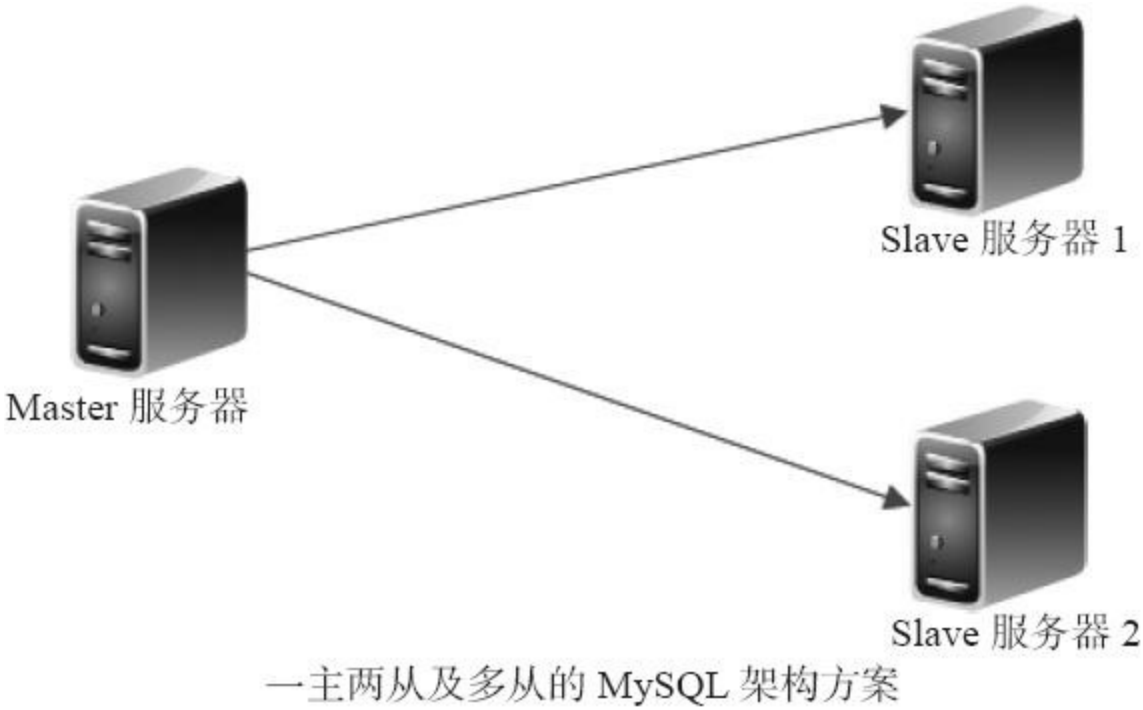


图9-3　一主多从逻辑图

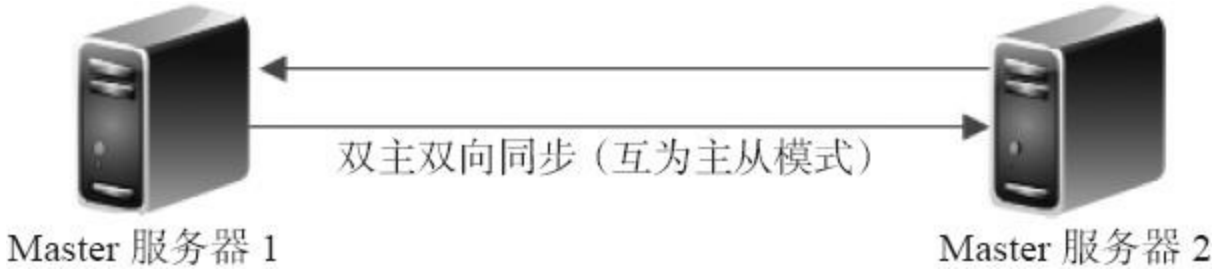


图9-4　双向主主复制逻辑图

图9-5为线性级联单向双主复制逻辑架构图，此架构只能在Master1端进行数据写入，工作场景中，Master1和Master2作为主主互备，Slave1作为从库，中间的Master2需要做特殊的设置。

图9-5　线性级联单向双主复制逻辑图

图9-6为环状级联单向多主同步逻辑架构图，任意一个点都可以写入数据，此架构比较复杂，属于极端环境下的“作品”，一般场景应慎用。

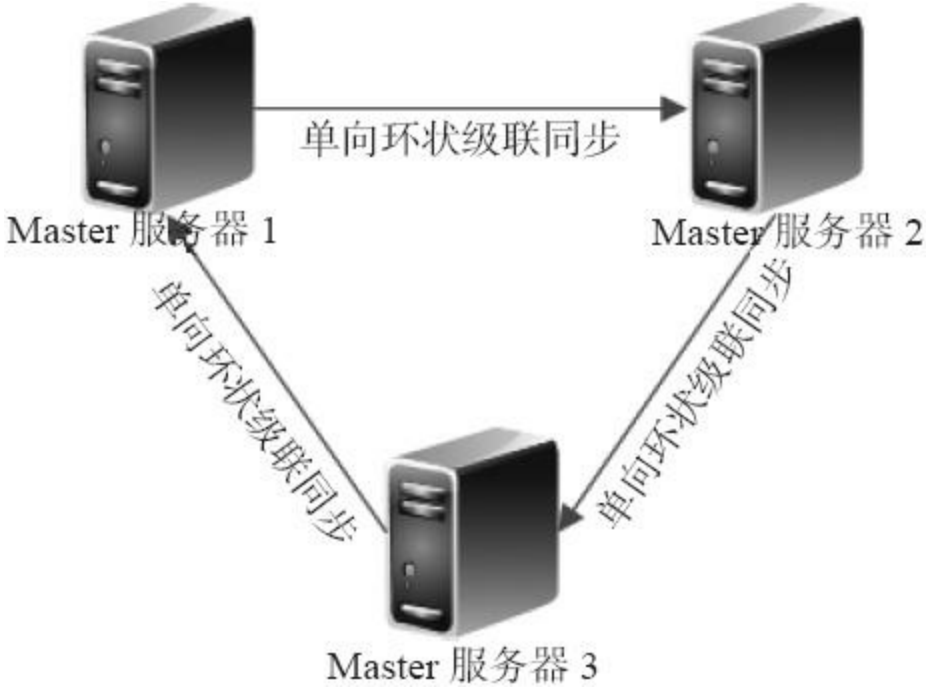


图9-6　环状级联单向多主同步逻辑图

此外，MySQL官方手册也给出了常见的复制架构图，如图9-7所示。

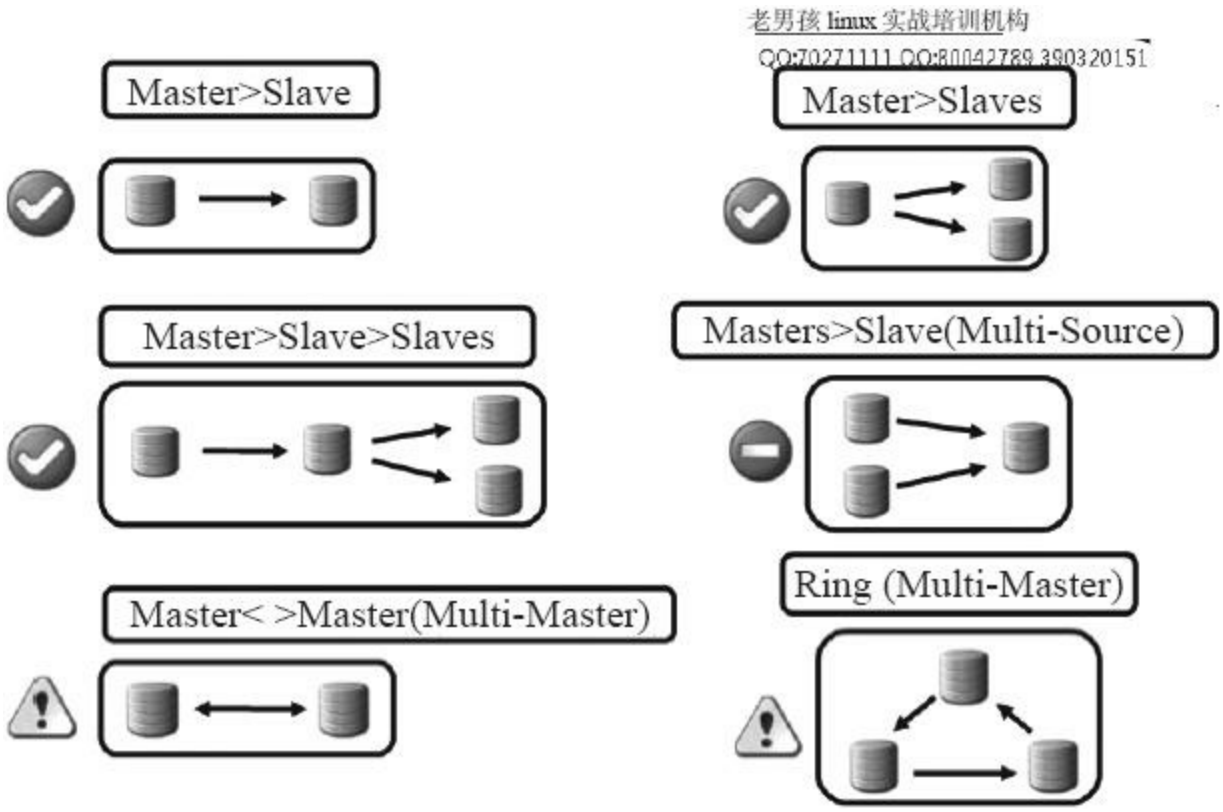


图9-7　MySQL官方的常见复制架构图

在当前的生产工作中，MySQL主从复制都是异步的复制方式，即不是严格实时的数据同步，但是正常情况下给用户的体验是实时的。

9.7.2 MySQL主从复制的企业应用场景

MySQL主从复制集群功能使得MySQL数据库支持大规模高并发读写成为可能，同时有效地保护了物理服务器宕机场景的数据备份。

应用场景1：从服务器作为主服务器的实时数据备份

主从服务器架构的设置可以大大加强MySQL数据库架构的健壮性。例如：当主服务器出现问题时，我们可以人工或设置自动切换到从服务器继续提供服务，此时从服务器的数据与宕机时的主数据库几乎是一致的。

这类似NFS存储数据通过inotify+rsync同步到备份的NFS服务器，只不过MySQL的复制方案是其自带的工具。

利用MySQL的复制功能进行数据备份时，在硬件故障、软件故障的场景下，该数据备份是有效的，但对于人为地执行drop、delete等语句删除数据的情况，从库的备份功能就没用了，因为从服务器也会执行删除的语句。

应用场景2：主从服务器实现读写分离，从服务器实现负载均衡

主从服务器架构可通过程序（PHP、Java等）或代理软件（mysql-

proxy、Amoeba）实现对用户（客户端）的请求读写分离，即让从服务器仅仅处理用户的select查询请求，降低用户查询响应时间，以及同时读写在主服务器上带来的访问压力。对于更新的数据（例如update、insert、delete语句），则仍然交给主服务器处理，确保主服务器和从服务器保持实时同步。

百度、淘宝、新浪等绝大多数的网站都是用户浏览页面多于用户发布内容，因此通过在从服务器上接收只读请求，就可以很好地减轻主库的读压力，且从服务器可以很容易地扩展为多台，使用LVS做负载均衡效果就非常棒了，这就是传说中的数据库读写分离架构。上述架构的逻辑图如图9-8所示。

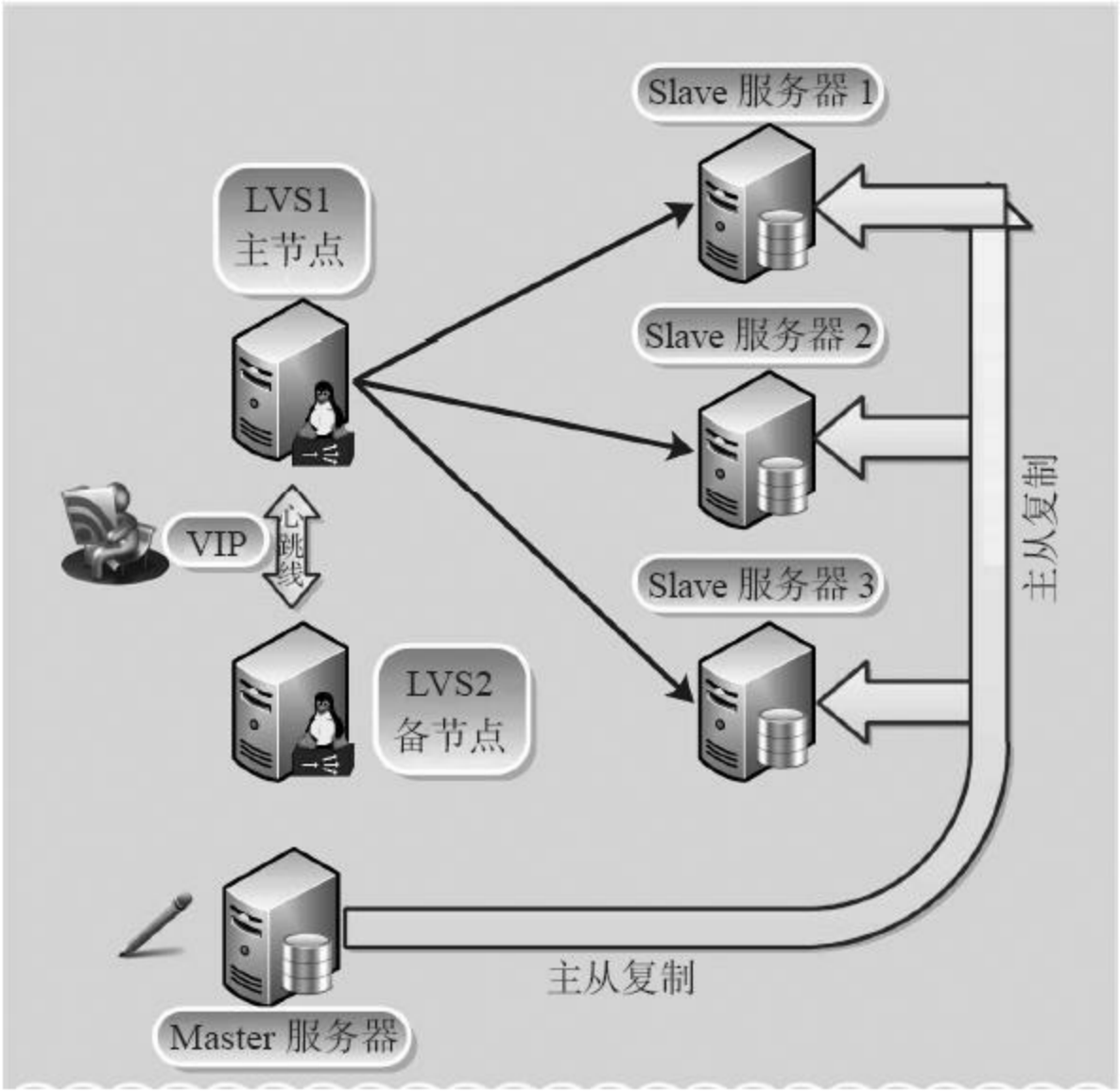


图9-8　一主多从读写分离及从库负载均衡架构逻辑图

应用场景3：把多个从服务器根据业务重要性进行拆分访问

可以把几个不同的从服务器，根据公司的业务进行拆分。例如：有为外部用户提供查询服务的从服务器，有内部DBA用来数据备份的从服务器，还有为公司内部人员提供访问的后台、脚本、日志分析及供开发

人员查询使用的从服务器。这样的拆分除了减轻主服务器的压力外，还可以使数据库对外部用户浏览、内部用户业务处理及DBA人员的备份等互不影响。具体可以用图9-9所示的简单架构来说明。

9.7.3　实现MySQL主从读写分离的方案

（1）通过程序实现读写分离（性能和效率最佳，推荐）

PHP和Java程序都可以通过设置多个连接文件轻松地实现对数据库的读写分离，即当语句关键字为select时，就去连接读库的连接文件，若为update、insert、delete时，则连接写库的连接文件。

通过程序实现读写分离的缺点就是需要开发人员对程序进行改造，使其对下层不透明，但这种方式更容易开发和实现，适合互联网业务场景。

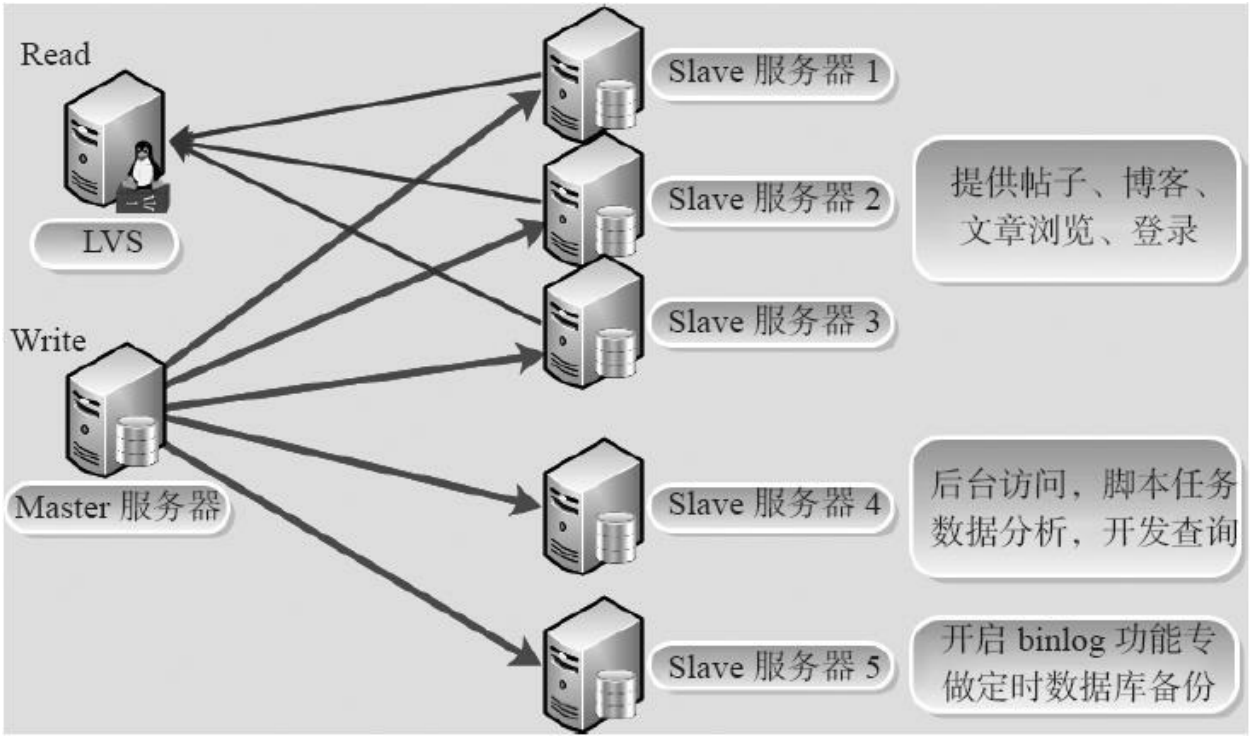


图9-9　MySQL主从复制根据业务重要性拆分从库方案

（2）通过开源的软件实现读写分离

MySQL-proxy、Amoeba等代理软件也可以实现读写分离功能，这些软件的稳定性和功能一般，不建议生产使用。绝大多数公司常用的还是在应用端发程序实现读写分离。

（3）大型门户独立开发DAL层综合软件

百度、阿里等大型门户都有开发牛人，会花大力气开发适合自己业务的读写分离、负载均衡、监控报警、自动扩容、自动收缩等一系列功能的DAL层软件，此部分可以参考老男孩架构师分布式数据库集群的课程内容。

MySQL读写分离的基本逻辑图如图9-10所示。

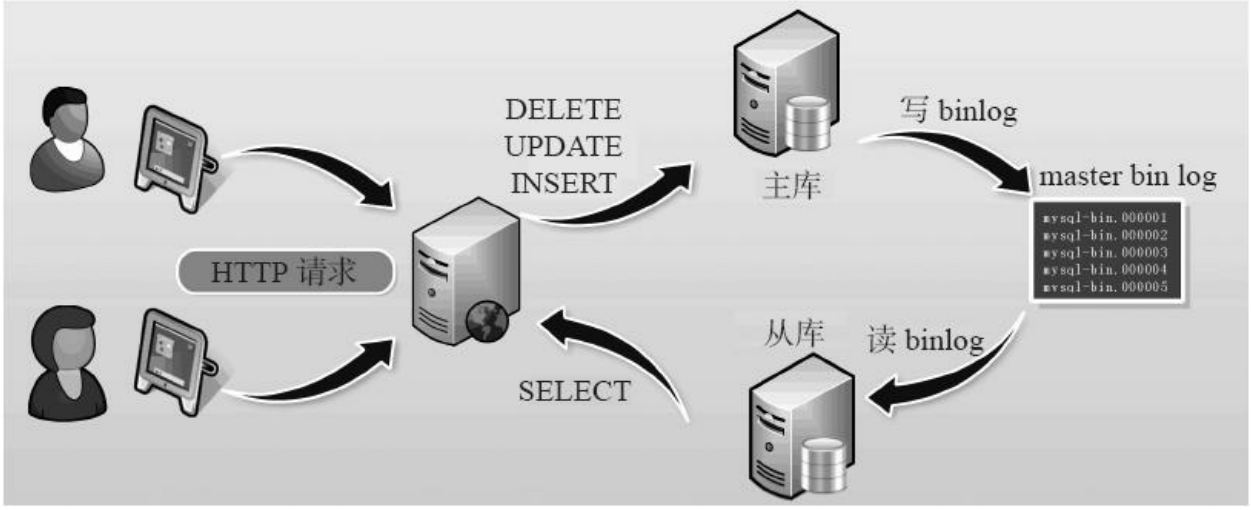


图9-10　MySQL读写分离的基本逻辑图

9.7.4 MySQL主从复制原理介绍

MySQL的主从复制是一个异步 的复制过程（虽然一般情况下感觉是实时的），数据将从一个MySQL数据库（我们称之为Master）复制到另一个MySQL数据库（我们称之为Slave），在Master与Slave之间实现整个主从复制的过程是由三个线程参与完成的。其中有两个线程（SQL线程和I/O线程）在Slave端，另外一个线程（I/O线程）在Master端。

要实现MySQL的主从复制，首先必须打开Master端的binlog记录功能，否则就无法实现。因为整个复制过程实际上就是Slave从Master端获取binlog日志，然后再在Slave上以相同顺序执行获取的binlog日志中所记录的各种SQL操作。

要打开MySQL的binlog记录功能，可通过在MySQL的配置文件my.cnf中的mysqld模块（[mysqld]标识后的参数部分）增加“log-bin”参数选项来实现，具体信息如下。

```
[mysqld]
log-bin = /data/3306/mysql-bin
```

 提示：有些读者因把log-bin放在了配置文件结尾，而不是[mysqld]标识后，从而导致配置复制不成功。

9.7.5 MySQL主从复制原理过程详细描述

下面简单描述MySQL Replication的复制原理过程。

1）在Slave服务器上执行start slave命令开启主从复制开关，开始进行主从复制。

2）此时，Slave服务器的I/O线程会通过在Master上已经授权的复制用户权限请求连接Master服务器，并请求从指定binlog日志文件的指定位置（日志文件名和位置就是在配置主从复制服务时执行change master命令指定的）之后开始发送binlog日志内容。

3）Master服务器接收到来自Slave服务器的I/O线程的请求后，其上负责复制的I/O线程会根据Slave服务器的I/O线程请求的信息分批读取指定binlog日志文件指定位置之后的binlog日志信息，然后返回给Slave端的I/O线程。返回的信息中除了binlog日志内容外，还有在Master服务器端记录的新的binlog文件名称，以及在新的binlog中的下一个指定更新位置。

4）当Slave服务器的I/O线程获取到Master服务器上I/O线程发送的日志内容、日志文件及位置点后，会将binlog日志内容依次写到Slave端自身的Relay Log（即中继日志）文件（MySQL-relay-bin.xxxxxx）的最末

端，并将新的binlog文件名和位置记录到master-info文件中，以便下一次读取Master端新binlog日志时能够告诉Master服务器从新binlog日志的指定文件及位置开始请求新的binlog日志内容。

5）Slave服务器端的SQL线程会实时检测本地Relay Log中I/O线程新增加的日志内容，然后及时地把Relay Log文件中的内容解析成SQL语句，并在自身Slave服务器上按解析SQL语句的位置顺序执行应用这些SQL语句，并在relay-log.info中记录当前应用中继日志的文件名及位置点。

经过了上面的过程，就可以确保在Master端和Slave端执行了同样的SQL语句。当复制状态正常时，Master端和Slave端的数据是完全一样的。当然，MySQL的复制机制也有一些特殊情况，具体请参考官方的说明，大多数情况下，大家不用担心。

图9-11为MySQL Replication的复制原理逻辑图。

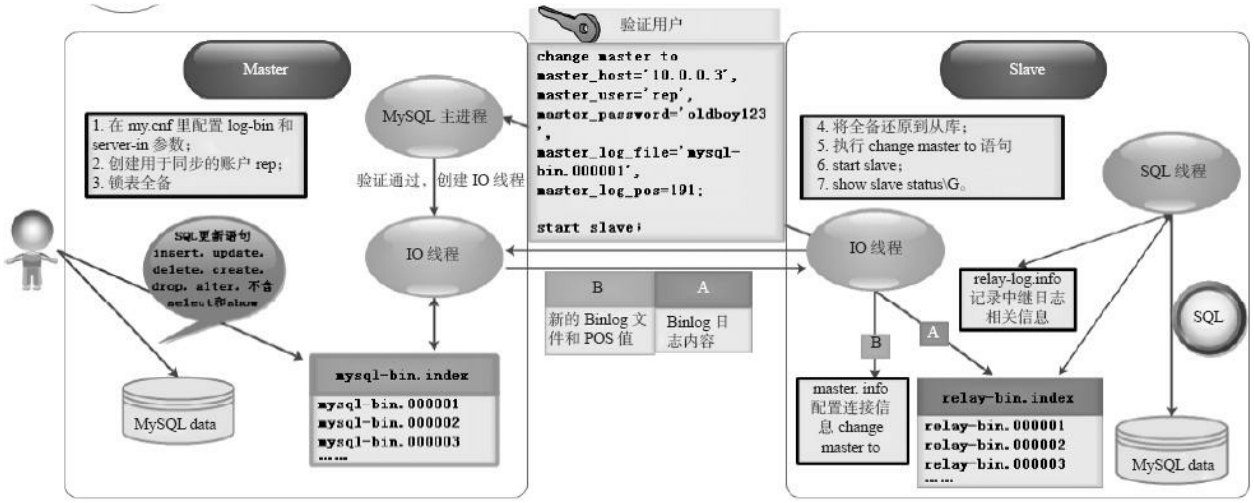


图9-11　MySQL主从复制基本原理逻辑图

 特别说明：当企业面试MySQL主从复制原理时，不管是面试还是笔试，都要尽量画图表达，而不是口头讲或文字描述，面试时可以找黑板或拿出纸来给面试官详细讲解。

下面针对MySQL主从复制原理的重点进行小结：

·主从复制是异步的逻辑的SQL语句级的复制。

·复制时，主库有一个I/O线程，从库有两个线程，即I/O和SQL线程。

·实现主从复制的必要条件是主库要开启记录binlog功能。

·作为复制的所有MySQL节点的server-id都不能相同。

·binlog文件只记录对数据库有更改的SQL语句（来自主数据库内容的变更），不记录任何查询（如select、show）语句。

9.8 MySQL主从复制实践

9.8.1 主从复制实践准备

1.主从复制数据库实战环境准备

MySQL主从复制实践对环境的要求比较简单，可以是单机单数据库多实例的环境，也可以是两台服务器，每个机器一个独立数据库的环境。本文以单机数据库多实例的环境为例讲解。实例端口信息查看如下：

```
[root@MySQL ~]# ss -lnt|grep 330
LISTEN     0      128                        *:


3306                    *:


*
LISTEN     0      128                        *:


3307                    *:


*
LISTEN     0      128                        *:


3308                    *:


*
```

提示：这里把3306实例作为主库，3307实例作为从库，如果根据前面的内容配置了MySQL多实例环境，直接开启多实例环境使用即可。

2.定义主从复制需要的服务器角色

主库及从库IP、端口信息如下：

```
主库（

mysql master）

  :

[ip 为

10.0.0.7 port 为

3306]从库

1（

mysql slave）

 :

[ip 为

10.0.0.7 port 为
```

```
3307]从库

2（

mysql slave）

 :

[ip 为

10.0.0.7 port 为

3308]
```

这里的主从复制技术是针对前面的内容以单机数据库多实例环境来讲的。一般情况下，小企业在做常规的主从复制时，主从服务器多数在不同的机器上，并且监听的端口均为默认的3306。虽然不在同一个机器上，但是步骤和过程却是一样的。

读者在掌握了单数据库多实例的同步方法后，可以自己适当扩展，完成异机相同端口之间的主从复制。

3.数据库中英文名称约定

MySQL主库，也可称为Master，本文对应服务的端口号为3306。

MySQL从库1，也可称为Slave1，本文对应服务的端口号为3307。

MySQL从库2，也可称为Slave2，本文对应服务的端口号为3308。

下面的内容中可能把主库称为Master，把从库称为Slave，或者反过来称呼，代表的意思都是一个。

9.8.2　在主库Master上执行操作配置

1.设置server-id值并开启binlog功能参数

根据前文介绍的MySQL主从复制原理我们知道，要实现主从复制，关键是要开启binlog日志功能，所以，首先来打开主库的binlog日志参数。

1）修改主库的配置文件。执行vi/data/3306/my.cnf，编辑多实例3306的my.cnf配置文件，按如下内容修改两个参数：

```
[mysqld]
server-id = 1                  #<==用于同步的每台机器或实例

server-id都不能相同。

log-bin = /data/3306/mysql-bin      #<==加粗可省略
```

提示：

·上面的两个参数要放在my.cnf中的[mysqld]模块下，否则会出错。

·server-id的值使用服务器IP地址最后一个小数点后面数字，如19，目的是避免不同机器或实例ID重复（不适合多实例）。

·0<server-id<2^{32} -1的自然数。

·要先在my.cnf配置文件中查找相关参数，并按要求修改。若发现不存在，再添加参数，切记，参数不能重复。

·修改my.cnf配置后需要重启数据库，命令为：/data/3306/mysql restart，注意要确认真正重启了。

2）检查配置参数后的结果，如下：

```
[root@MySQL ~]# egrep "server-id|log-bin" /data/3306/my.cnf
log-bin = /data/3306/mysql-bin <== log-bin后面也可以不带等号内容，


Mysql会使用默认日志



server-id = 1
```

3）重启主库MySQL服务，命令如下：

```
[root@MySQL ~]# /data/3306/mysql restart
Restarting MySQL...
```

4）登录数据库，检查参数的更改情况，如下：

```
[root@MySQL ~]# mysql -uroot -poldboy123 -S /data/3306/mysql.sock
Welcome to the MySQL monitor.  Commands end with ;


 or \g.
Your MySQL connection id is 2
Server version:
```

```
 5.5.32-log MySQL Community Server （

GPL）

Copyright （

c）

 2000，

 2013，

 Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help；

' or '\h' for help. Type '\c' to clear the current input statement.
mysql> show variables like 'server_id'；

+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 1     |  #<==通过检查得知配置的

server_id为

1
+---------------+-------+
1 row in set （

0.00 sec）
```

```
mysql> show variables like 'log_bin';


+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |  #<==binlog功能已开启


+---------------+-------+
1 row in set （


0.00 sec）
```

这样，binlog功能就开启了。

2.在主库上建立用于主从复制的账号

根据主从复制的原理，从库要想和主库同步，必须有一个可以连接主库的账号，并且这个账号是主库上创建的，权限是允许主库的从库连接并同步数据。

1）登录MySQL 3306实例主数据库，命令如下：

```
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3306/mysql.sock
```

2）建立用于从库复制的账号rep，命令如下：

```
mysql> grant replication slave on *.* to 'rep'@'10.0.0.%' identified by 'oldboy123';
```

Query OK，

0 rows affected （

0.00 sec）

#replication slave为

mysql同步的必须权限，此处不要授权

all权限

. 表示所有库所有表，也可以指定具体的库和表进行复制。例如

oldboy.test中，

oldboy为库，

test为表

'rep'@'10.0.0.%' rep为同步账号。

10.0.0.%为授权主机网段，使用了

%表示允许整个

10.0.0.0网段以

```
rep用户访问

# identified by 'oldboy123';

 oldboy123为密码，实际环境下设置得复杂些为好

mysql> flush privileges;

           #<==刷新权限，使授权的权限生效

Query OK,

 0 rows affected （

0.02 sec）
```

3）检查主库创建的rep复制账号命令及结果如下：

```
mysql> select user,

host from mysql.user;

+-----------+-----------+
| user      | host      |
+-----------+-----------+
| rep       | 10.0.0.%  |     #<==出现这一行表示复制账号已经配置好了
```

```
| root      | 127.0.0.1 |
| root      | localhost |
+-----------+-----------+
3 rows in set （


0.01 sec）



mysql> select user，


host from mysql.user where user='rep'；



+------+----------+
| user | host     |
+------+----------+
| rep  | 10.0.0.% |
+------+----------+
1 row in set （


0.00 sec）



mysql> show grants for rep@'10.0.0.%'；



+-------------------------------------------------------------------------+
| GRANT REPLICATION SLAVE ON *.* TO 'rep'@'10.0.0.%' IDENTIFIED BY PASSWORD '*FE288
```

rep账号权限正是配置的主从复制需要的权限

```
 REPLICATION SLAVE
+-------------------------------------------------------------------------+
1 row in set （


0.00 sec）
```

3.实现对主数据库锁表只读

1）对主数据库锁表只读（当前窗口不要关掉）的命令如下：

```
mysql> flush table with read lock;

Query OK,

 0 rows affected （

0.00 sec）
```

 提示：在引擎不同的情况下，这个锁表命令的时间会受下面参数的控制。锁表时，如果超过设置时间不操作会自动解锁。

默认情况下自动解锁的时长参数值如下：

```
mysql> show variables like '%timeout%';

<==截取部分内容如下

+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| interactive_timeout        | 28800 |
| wait_timeout               | 28800 |
+----------------------------+-------+
10 rows in set （
```

```
0.00 sec)
```


提示：有关这两个参数，请读者自行测试。

2）锁表后查看主库状态。可通过当前binlog日志文件名和二进制binlog日志偏移量来查看，结果如下。

注意，show master status；命令显示的信息要记录在案，后面的从库导入全备后，继续和主库复制时就是要从这个位置开始。

```
mysql> show master status;


+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000008 |      342 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set （


0.00 sec)
```

或者新开命令行窗口，用如下命令查看锁表后的主库binlog位置点信息：

```
[root@MySQL ~]# mysql -u root -p'oldboy123' -S /data/3306/mysql.sock -e "show master
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000008 |      342 |              |                  |
```

```
+-----------------+----------+-------------+-----------------+
```

3）锁表后，一定要单开一个新的SSH窗口，导出数据库的所有数据，如果数据量很大（50GB以上），并且允许停机，可以停库直接打包数据文件进行迁移，那样更快。

```
[root@MySQL ~]# mkdir /server/backup/ -p
[root@MySQL ~]# mysqldump -uroot -p'oldboy123' -S /data/3306/mysql.sock --events -A


date +%F）


.sql.gz
#注意：


-A 表示备份所有库；


-B表示增加


 use DB 和


 drop等（导库时会直接覆盖原有的）


[root@MySQL ~]# ls -l /server/backup/mysql_bak.$（


date +%F）


.sql.gz
-rw-r--r-- 1 root root 144341 7月


  27 15：
```

```
37 /server/backup/mysql_bak.2015-07-27.sql.gz
```

为了确保导出数据期间，数据库没有数据插入，导库完毕可以再次检查主库状态信息，结果如下：

```
[root@MySQL ~]# mysql -u root -p'oldboy123' -S /data/3306/mysql.sock -e "show master
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000008 |      342 |              |                  |
+------------------+----------+--------------+------------------+
```


提示：若无特殊情况，binlog文件及位置点和锁表后导出数据前是一致的，即没有变化。

导出数据完毕后，解锁主库，恢复可写，命令如下。因为主库还要对外提供服务，不能一直锁定不让用户访问。

```
mysql>unlock tables;
```

可能会有读者因为锁表后的binlog位置问题犯迷糊，实际上做从库前，无论主库更新了多少数据，最后从库都可以从上面show master status的位置很快赶上主库的进度。

4.把主库导出的MySQL数据迁移到从库

这里常用的命令有scp、rsync等，可将备份的数据往异地拷贝。

下面主要讲解单数据库多实例的主从配置，也就是说，mysqldump备份的3306实例的数据和要恢复的3307实例在一台机器上，因此无需异地复制拷贝。先查看主库导出的数据，如下：

```
[root@MySQL ~]# ls -l /server/backup/mysql_bak.$(


date +%F)


.sql.gz
-rw-r--r-- 1 root root 144341 7月


  27 15:


37 /server/backup/mysql_bak.2015-07-27.sql.gz
```

9.8.3　在MySQL从库上执行的操作过程

1.设置server-id值并关闭binlog功能参数

数据库的server-id一般在一套主从复制体系内是唯一的，这里从库的server-id要和主库及其他从库的不同，并且要注释掉从库的binlog参数配置，如果从库不做级联复制，并且不作为备份用，就不要开启binlog，开启了反而会增加从库磁盘I/O等的压力。

但是，有以下两种情况需要打开从库的binlog记录功能，记录数据库更新的SQL语句：

·级联同步A→B→C中间的B时，就要开启binlog记录功能。

·在从库做数据库备份，数据库备份必须要有全备和binlog日志，才是完整的备份。

（1）修改配置文件，配置从库1的相关参数

执行vi/data/3307/my.cnf，编辑my.cnf配置文件，按如下内容修改两个参数：

```
[mysqld]
server-id = 3  #<==调整等号后的数值，和任何一个数据库实例都不同
```

提示：

·上面两参数要放在my.cnf中的[mysqld]模块下，否则会出错。

·server-id的值可使用服务器IP地址最后一个小数点后面的数字，如179，目的是避免不同机器或实例ID重复（不适合多实例）。

·要先在文件中查找相关参数按要求修改。若发现不存在，再添加参数，切记，参数不能重复。

·修改my.cnf配置后需要重启数据库，命令为：/data/3307/mysql restart，注意要确认真正重启了。

（2）检查配置参数后的结果

命令如下：

```
[root@MySQL ~]# egrep "server-id|log-bin" /data/3307/my.cnf
server-id = 3
```

（3）重启3307的从数据库

命令如下：

```
[root@MySQL ~]# /data/3307/mysql stop
Stoping MySQL...
[root@MySQL ~]# /data/3307/mysql start
Starting MySQL...
[root@MySQL ~]# ss -lnt|grep 3307
```

```
LISTEN     0      128                          *:

3307                       *:

*
```

（4）登录数据库检查参数的改变情况

命令如下：

```
[root@MySQL ~]# mysql -uroot -poldboy123 -S /data/3307/mysql.sock
Welcome to the MySQL monitor.  Commands end with ;

 or \g.
Your MySQL connection id is 3
Server version:

 5.5.32 MySQL Community Server  (

GPL)

Copyright  (

c)

 2000,

 2013,

 Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

```
Type 'help;


' or '\h' for help. Type '\c' to clear the current input statement.
mysql> show variables like 'log_bin';



+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | OFF   |
+---------------+-------+
1 row in set （


0.00 sec）



mysql> show variables like 'server_id';



+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 3     |
+---------------+-------+
1 row in set （


0.00 sec）
```

2.把从主库mysqldump导出的数据恢复到从库

操作命令如下：

```
[root@MySQL ~]# cd /server/backup/
[root@MySQL backup]# ls -l总用量


 144
-rw-r--r-- 1 root root 144341 7月
```

```
  27 15:


37 mysql_bak.2015-07-27.sql.gz
[root@MySQL backup]# gzip -d mysql_bak.2015-07-27.sql.gz #<==解压目标数据库备份，源文件会


[root@MySQL backup]# ll总用量


 520
-rw-r--r-- 1 root root 528662 7月


  27 15:


37 mysql_bak.2015-07-27.sql
[root@MySQL backup]# mysql -uroot -p'oldboy123' -S /data/3307/mysql.sock < mysql_bak
<==这是把数据还原到


3307实例的命令
```

提示：如果备份时使用了-A参数，则在还原数据到3307实例时，登录3307实例的密码也会和3306主库的一致，因为3307实例的授权表MySQL也被覆盖了。

3.登录3307从库，配置复制参数

1）MySQL从库连接主库的配置信息如下：

```
CHANGE MASTER TO
MASTER_HOST='10.0.0.7',
```

#<==这里是主库的

IP
MASTER_PORT=3306,

#<==这里是主库的端口，从库端口可以和主库不同

MASTER_USER='rep',

#<==这里是主库上建立的用于复制的用户

rep
MASTER_PASSWORD='oldboy123',

#<==这里是

rep用户的密码

MASTER_LOG_FILE='mysql-bin.000008',

#<==这里是

show master status时查看到的二进制日志文件名称，注意不能多空格

MASTER_LOG_POS=342;

#<==这里是

show master status时查看到的二进制日志偏移量，注意不能多空格

提示：字符串用单引号括起来，数值不用引号，注意内容前后不能有空格。

2）登录数据库后，去掉上述语句中的注释，执行如下：

```
CHANGE MASTER TO
MASTER_HOST='10.0.0.7',

MASTER_PORT=3306,

MASTER_USER='rep',

MASTER_PASSWORD='oldboy123',

MASTER_LOG_FILE='mysql-bin.000008',

MASTER_LOG_POS=342;
```

提示：这个步骤的参数一定不能错，否则，数据库复制配置会失败。

也可以不登录数据库内部命令行，在Linux命令行快速执行

CHANGE MASTER的语句实现相应的功能。下面的语句特别适合在脚本中一键配置从库使用：

```
    mysql -uroot -p'oldboy123' -S /data/3307/mysql.sock<< EOF
CHANGE MASTER TO
    MASTER_HOST='10.0.0.7',

    MASTER_PORT=3306,

    MASTER_USER='rep',

    MASTER_PASSWORD='oldboy123',

    MASTER_LOG_FILE='mysql-bin.000008',

    MASTER_LOG_POS=342;

EOF
```

整个配置MySQL从库连接主库的信息操作过程如下：

```
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3307/mysql.sock<< EOF
> CHANGE MASTER TO
> MASTER_HOST='10.0.0.7',

> MASTER_PORT=3306,
```

```
> MASTER_USER='rep',

> MASTER_PASSWORD='oldboy123',

> MASTER_LOG_FILE='mysql-bin.000008',

> MASTER_LOG_POS=342;

> EOF
```

上述操作的原理实际上是把用户密码等信息写入从库新的master.info文件中。

```
[root@MySQL ~]# ll /data/3307/data/master.info
-rw-rw---- 1 mysql mysql 76 7月

  27 15:

46 /data/3307/data/master.info
[root@MySQL ~]# cat /data/3307/data/master.info
18
mysql-bin.000008     #<==这里是

show master status时查看到的二进制日志文件名称，注意不能多空格

342                #<==这里是

show master status时查看到的二进制日志偏移量，注意不能多空格
```

```
10.0.0.7            #<==这里是主库的


ip
rep                  #<==这里是主库上建立的用于复制的用户


rep
oldboy123             #<==这里是


rep用户的密码


3306                  #<==这里是主库的端口


60
0...省略若干...
```

9.8.4　启动从库同步开关，测试主从复制配置情况

（1）启动从库主从复制开关，并查看复制状态

相关语句如下：

```
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3307/mysql.sock -e "start slave;


"
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3307/mysql.sock -e "show slave s


"
*************************** 1. row ***************************
               Slave_IO_State:


 Waiting for master to send event
                  Master_Host:


 10.0.0.7
                  Master_User:


 rep
                  Master_Port:


 3306
                Connect_Retry:


 60
              Master_Log_File:


 mysql-bin.000001
          Read_Master_Log_Pos:
```

471
              Relay_Log_File:

relay-bin.000002
               Relay_Log_Pos:

253
       Relay_Master_Log_File:

mysql-bin.000001
            Slave_IO_Running:

Yes
           Slave_SQL_Running:

Yes
             Replicate_Do_DB:

         Replicate_Ignore_DB:

mysql
          Replicate_Do_Table:

      Replicate_Ignore_Table:

     Replicate_Wild_Do_Table:

 Replicate_Wild_Ignore_Table:

                  Last_Errno:

```
                  Last_Error:

                Skip_Counter: 0
         Exec_Master_Log_Pos: 471
             Relay_Log_Space: 403
             Until_Condition: None
              Until_Log_File:
               Until_Log_Pos: 0
          Master_SSL_Allowed: No
          Master_SSL_CA_File:
          Master_SSL_CA_Path:
             Master_SSL_Cert:
           Master_SSL_Cipher:
```

```
                Master_SSL_Key:

         Seconds_Behind_Master:

 0
Master_SSL_Verify_Server_Cert:

 No
                Last_IO_Errno:

 0
                Last_IO_Error:

               Last_SQL_Errno:

 0
               Last_SQL_Error:

  Replicate_Ignore_Server_Ids:

             Master_Server_Id:

 1
```

主从复制是否成功，最关键的为下面的3项状态参数：

```
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3307/mysql.sock -e "show slave s
             Slave_IO_Running:

 Yes
            Slave_SQL_Running:
```

```
Yes
       Seconds_Behind_Master:

0
```

·Slave_IO_Running：Yes，这个是I/O线程状态，I/O线程负责从从库到主库读取binlog日志，并写入从库的中继日志，状态为Yes表示I/O线程工作正常。

·Slave_SQL_Running：Yes，这个是SQL线程状态，SQL线程负责读取中继日志（relay-log）中的数据并转换为SQL语句应用到从数据库中，状态为Yes表示I/O线程工作正常。

·Seconds_Behind_Master：0，这个是复制过程中从库比主库延迟的秒数，这个参数很重要，但企业里更准确地判断主从延迟的方法为：在主库写时间戳，然后从库读取时间戳，和当前数据库时间进行比较，从而认定是否延迟。

有关show slave status结果的说明，请大家查看MySQL手册。

（2）测试主从复制结果

在主库上写入数据，然后观察从库的数据状况。

```
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3306/mysql.sock -e "create datab
```

```
"
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3307/mysql.sock -e "show databas


"
+--------------------+
| Database （


oldboy）


 |
+--------------------+
| oldboy             |
+--------------------+
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3306/mysql.sock -e "drop databas


"
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3307/mysql.sock -e "show databas


"
```

根据测试可以判断，主从库是同步的。

9.8.5 MySQL主从复制问题汇总

故障1：主库show master status；没返回状态结果，如下：

```
mysql> show master status;


Empty set （


0.00 sec）
```

解答：上述问题是主库binlog功能开关没开或没生效导致的。

正确开启binlog功能的配置结果如下：

```
[root@MySQL ~]# grep "log-bin" /data/3306/my.cnf
log-bin = /data/3306/mysql-bin
[root@MySQL ~]# mysql -uroot -p'oldboy123' -S /data/3306/mysql.sock -e "show variabl


"
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
1 row in set （


0.00 sec）
```

提示：配置文件里的参数和show variables里的参数可能不一样，例如，my.cnf里的log-bin和show里的log_bin长相就是不一样的。

故障2：出现错误信息“Last_IO_Error：Got fatal error 1236 from master when reading data from binary log：'Could not find first log file name in binary log index file'”。

解答：上面故障的原因是执行CHANGE MASTER命令时，某一个参数的值多了空格，如下：

CHANGE MASTER TO

MASTER_HOST='10.0.0.7'，

MASTER_PORT=3306，

MASTER_USER='rep'，

MASTER_PASSWORD='oldboy123'，

MASTER_LOG_FILE='dmysql-bin.000008d'，#<==内容的两端不能有空格。

MASTER_LOG_POS=342；

故障3：服务无法启动。

故障语句如下：

```
[root@MySQL ~]# /data/3306/mysql start
MySQL is running...
[root@MySQL ~]# ps -ef|grep mysql
root      1636  1595  0 11:


47 pts/0    00:


00:


00 grep --color=auto mysql
```

解决：其原因是启动脚本里对mysql.sock是否存在做了判断，示例如下。如果存在mysql.sock，就认为服务运行是个小bug。读者可以自行更改启动脚本来解决。

```
[root@MySQL ~]# rm -f /data/3306/mysql.sock /data/3306/*.pid
[root@MySQL ~]# /data/3306/mysql start
Starting MySQL...
[root@MySQL ~]# mysql -uroot -poldboy123 -S /data/3306/mysql.sock
```

9.8.6 MySQL主从复制配置步骤小结

MySQL主从复制配置完整步骤如下。

1）准备两台数据库环境或单台多实例环境，确定能正常启动和登录。

2）配置my.cnf文件：主库配置log-bin和server-id参数；从库配置server-id，该值不能和主库及其他从库一样，一般不开启从库log-bin功能。注意，配置参数后要重启才能生效。

3）登录主库，增加从库连接主库同步的账户，例如：rep，并授权replication slave同步的权限。

4）登录主库，整库锁表flush table with read lock（窗口关闭后即失效，超时参数设置的时间到了，锁表也失效），然后show master status查看binlog的位置状态。

5）新开窗口，在Linux命令行备份导出原有的数据库数据，并拷贝到从库所在的服务器目录。如果数据库数据量很大，并且允许停机，可以停机打包，而不用mysqldump。

6）导出主库数据后，执行unlock tables解锁主库。

7）把主库导出的数据恢复到从库。

8）根据主库的show master status查看到的binlog的位置状态，在从库执行change master to...语句。

9）从库开启复制开关，即执行start slave；。

10）从库show slave status\G，检查同步状态，并在主库进行更新测试。

9.8.7　生产场景下轻松部署MySQL主从复制

1.快速配置MySQL主从复制

步骤如下：

1）安装好要配置从库的数据库，配置好log-bin和server-id参数。

2）无需配置主库my.cnf文件，主库的log-bin和server-id参数默认就是配置好的。

3）登录主库，增加从库连接主库同步的账户，例如：rep，并授权replication slave同步的权限。

4）使用曾经在半夜通过mysqldump带-x和--master-data=1的命令及参数定时备份的全备数据文件，把它恢复到从库。

5）在从库执行change master to...语句，无需binlog文件及对应位置点。

6）从库开启同步开关，start slave。

7）从库show slave status\G，检查同步状态，并在主库进行更新测试。

2.无需熬夜，轻松部署MySQL主从复制

实战过程如下。

1）半夜在主库上通过定时任务执行如下命令，备份导出主库数据：

```
mysqldump -uroot -p'oldboy123' -S /data/3306/mysql.sock -A --events -B -x --master-d

date +%F)

.sql.gz
```

--master-data=1参数会在备份数据里增加如下语句：

```
-- Position to start replication or point-in-time recovery from
CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000005',

 MASTER_LOG_POS=107;
```

2）白天找机会在需要做复制的从库上导入全备做从库，命令如下：

```
gzip -d 2015-07-28.sql.gz
mysql -uroot -p'oldboy123' -S /data/3308/mysql.sock <2015-07-28.sql
mysql -uroot -p'oldboy123' -S /data/3308/mysql.sock<< EOF
CHANGE MASTER TO
MASTER_HOST='10.0.0.5',
```

```
MASTER_PORT=3306,

MASTER_USER='rep',

MASTER_PASSWORD='oldboy123';

EOF
```

这里的CHANGE MASTER后面无需指定binlog文件名及具体位置，因为这部分已经在还原数据时提前应用到数据库里了（备份时--master-data=1的功劳）。

```
start slave;

        #<==开启主从复制开关

show slave status\G     #<==查看主从复制状态
```

9.8.8　MySQL主从复制线程状态说明及用途

1.MySQL主从复制主库I/O线程状态说明

（1）登录主数据库查看MySQL线程的同步状态

命令如下：

```
mysql> show processlist\G
*************************** 1. row ***************************
     Id:


 7
   User:


 rep
   Host:


 10.0.0.7:


27306
     db:


 NULL
Command:


 Binlog Dump
   Time:


 538
  State:
```

```
Master has sent all binlog to slave;


waiting for binlog to be updated
  Info:


NULL
```

提示：上述状态的意思是线程已经从binlog日志读取所有更新，并已经发送到了从数据库服务器。线程目前为空闲状态，等待由主服务器上二进制日志中的新事件更新。

表9-4中列出了主服务器的binlog Dump线程中State列的最常见状态。如果你没有在主服务器上看见任何binlog Dump线程，则说明复制没有运行，二进制binlog日志由各种事件组成，事件通常会为更新添加信息。

表9-4　主库I/O线程工作状态

主库 I/O 线程工作状态	解释说明
Sending binlog event to slave	线程已经从二进制 binlog 日志读取了一个事件并且正将它发送到从服务器
Finished reading one binlog; switching to next binlog	线程已经读完二进制 binlog 日志文件，并且正打开下一个要发送到从服务器的 binlog 日志文件
Has sent all binlog to slave; waiting for binlog to be updated	线程已经从 binlog 日志读取所有更新并已经发送到了从数据库服务器。线程目前为空闲状态，等待由主服务器上二进制 binlog 日志中的新事件更新
Waiting to finalize termination	线程停止时发生的一个很简单的状态

（2）登录从数据库查看MySQL线程工作状态

从库有两个线程，即I/O和SQL线程。从库I/O线程的状态如下：

```
mysql> show processlist\G
*************************** 1. row ***************************
     Id:


 1
   User:


 system user
   Host:



     db:


 NULL
Command:


 Connect
   Time:


 493
  State:


 Waiting for master to send event
   Info:


 NULL
```

表9-5列出了从服务器的I/O线程的State列的最常见的状态。该状态也出现在Slave_IO_State列，由SHOW SLAVE STATUS显示。

表9-5　从库I/O线程工作状态

从库 I/O 线程工作状态	解释说明
Connecting to master	线程正试图连接主服务器
Checking master version	同主服务器之间建立连接后临时出现的状态
Registering slave on master	
Requesting binlog dump	建立同主服务器之间的连接后临时出现的状态。线程向主服务器发送一条请求，索取从请求的二进制 binlog 日志文件名和位置开始的二进制 binlog 日志的内容
Waiting to reconnect after a failed binlog dump request	如果二进制 binlog 日志转储请求失败，线程进入睡眠状态，然后定期尝试重新连接。可以使用 --master-connect-retry 选项指定重试之间的间隔
Reconnecting after a failed binlog dump request	线程正尝试重新连接主服务器
Waiting for master to send event	线程已经连接上主服务器，正等待二进制 binlog 日志事件到达

（续）

从库 I/O 线程工作状态	解释说明
Queueing master event to the relay log	线程已经读取一个事件，正将它复制到中继日志供 SQL 线程来处理
Waiting to reconnect after a failed master event read	读取时（由于没有连接）出现错误。线程企图重新连接前将睡眠 master-connect-retry 秒
Reconnecting after a failed master event read	线程正尝试重新连接主服务器。当连接重新建立后，状态变为 Waiting for master to send event

从库SQL线程的状态如下：

```
*************************** 2. row ***************************
     Id:


  2
   User:


 system user
   Host:


      db:
```

```
     NULL
  Command: 


     Connect
       Time: 


     435
      State: 


     Slave has read all relay log; 


     waiting for the slave I/O thread to update it
       Info: 


     NULL
```

表9-6列出了从服务器的SQL线程的State列的最常见状态。

表9-6　从库SQL线程状态

从库 SQL 线程状态	解释说明
Reading event from the relay log	线程已经从中继日志读取一个事件，可以对事件进行处理了
Has read all relay log; waiting for the slave I/O thread to update it	线程已经处理了中继日志文件中的所有事件，现在正等待 I/O 线程将新事件写入中继日志
Waiting for slave mutex on exit	线程停止时发生的一个很简单的状态

有关MySQL主从复制参与线程的状态的更多信息，请参考MySQL官方手册。

2.查看MySQL线程同步状态的用途

通过MySQL线程同步状态可以看到同步是否正常进行，故障的位

置是什么，另外还可查看数据库同步是否完成，可用于主库宕机切换数据库或人工数据库主从切换迁移等。

例如：主库宕机，要选择最快的从库将其提升为主库，就需要查看主从库的线程状态，如果主从复制在正常情况下进行角色切换，也需要查看主从库的线程状态，根据复制状态确定更新是否完成。

9.9 MySQL主从复制更多应用技巧实践

1.工作中MySQL从库停止复制故障案例

模拟重现故障的能力是运维人员最重要的能力。下面就来次模拟操作。先在从库创建一个库，然后去主库创建同名的库来模拟数据冲突，命令如下：

```
show slave status：报错：且

show slave status\G：

Slave_IO_Running：

 Yes
Slave_SQL_Running：

 NO
Seconds_Behind_Master：

 NULL
                  Last_Error：

 Error 'Can't create database 'xiaoliu'；

 database exists' on query. Default database：

 'xiaoliu'. Query：
```

```
 'create database xiaoliu'
```

对于该冲突，解决方法1如下：

```
stop slave;

 #<==临时停止同步开关

set global sql_slave_skip_counter =1 ;

     #<==将同步指针向下移动一个，如果多次不同步，

                                    可以重复操作

start slave;
```

对于普通的互联网业务，上述的移动指针的操作带来的问题不是很大。当然，要在确认不影响公司业务的前提下。

若是在企业场景下，对当前业务来说，解决主从同步比主从不一致更重要，如果主从数据一致也是很重要的，那就再找个时间恢复这个从库。

是主从数据不一致更重要，还是保持主从同步持续状态更重要，要

根据业务选择。

这样Slave就会与Master同步了，主要关键点如下：

```
Slave_IO_Running:

  Yes
Slave_SQL_Running:

 Yes
Seconds_Behind_Master是否为

0,

#0表示已经同步状态
```

 提示：set global sql_slave_skip_counter=n；#n取值>0，忽略执行N个更新。

解决方法2：根据可以忽略的错误号事先在配置文件中配置，跳过指定的不影响业务数据的错误，例如：

```
[root@MySQL ~]# grep slave-skip /data/3306/my.cnf
slave-skip-errors = 1032,

1062,

1007
```

 提示：类似由于入库重复导致的失败可以忽略，其他情况是不是可以忽略需要根据不同公司的具体业务来评估。

其他可能引起复制故障的原因：

·MySQL自身的原因及人为重复插入数据。

·不同的数据库版本会引起不同步，低版本到高版本可以，但是高版本不能往低版本同步。

·MySQL的运行错误或程序bug。

·binlog记录模式，例如：row level模式就比默认的语句模式要好。

2.让MySQL从库记录binlog日志的方法

从库需要记录binlog的应用场景：当前的从库还要作为其他从库的主库，例如级联复制或双主互为主从场景的情况下。下面介绍从库记录binlog日志的方法。

在从库的my.cnf中加入如下参数，然后重启服务生效即可。

```
log-slave-updates       #<==必须要有这个参数


log-bin = /data/3307/mysql-bin
expire_logs_days = 7     #<==相当于
```

```
find /data/3307/ -type f -name " mysql-bin.000*" -mtime +7 |xargs rm -f
```

3.MySQL主从复制集群架构的数据备份策略

有主从复制了，还需要做定时全量加增量备份么？答案是肯定的！

因为，如果主库有语句级误操作（例如：drop database oldboy；），从库也会执行drop database oldboy；，这样MySQL主从库就都删除了该数据。

把从库作为数据库备份服务器时，备份策略如下：

高并发业务场景备份时，可以选择在一台从库上备份（Slave5），把从库作为数据库备份服务器时需要在从库开启binlog功能，其逻辑如图9-12所示。

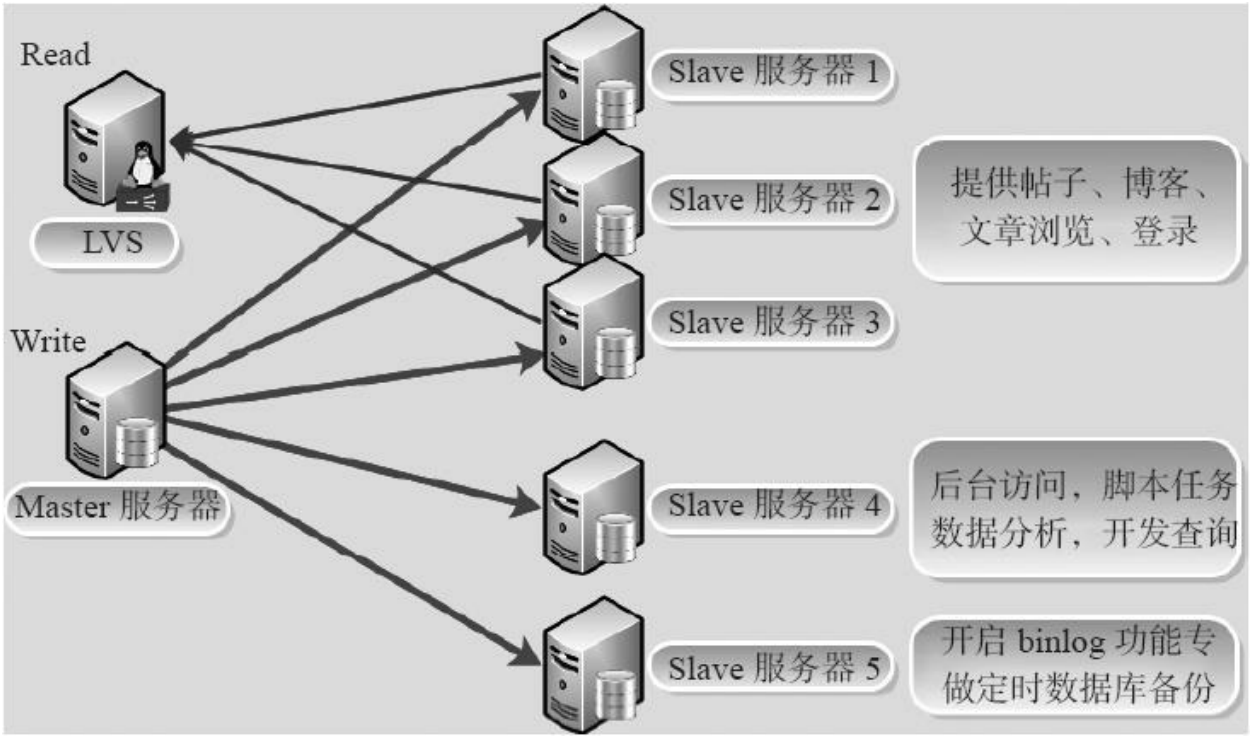


图9-12　MySQL主从复制根据业务重要性拆分从库方案

步骤如下：

1）选择一个不对外提供服务的从库，这样可以确保和主库更新最接近，专门用于做数据备份。

2）开启从库的binlog功能。

备份时可以选择只停止SQL线程，停止应用SQL语句到数据库，I/O线程保留工作状态，执行命令为stop slave sql_thread；，备份方式可以采取mysqldump逻辑备份或直接物理备份，例如：使用cp、tar（针对/data/目录）工具或xtrabackup（第三方的物理备份软件）进行备份，则逻辑备份和物理备份的选择，一般是根据总的备份数据量的多少进行

选择的，数据量低于30G，建议选择mysqldump逻辑备份方法，安全稳定，最后把全备和binlog数据发送到备份服务器上留存。

4.MySQL主从复制延迟问题的原因及解决方案

问题一：主库的从库太多，导致复制延迟。

从库数量以3~5个为宜，要复制的从节点数量过多，会导致复制延迟。

问题二：从库硬件比主库差，导致复制延迟。

查看Master和Slave的系统配置，可能会因为机器配置不当，包括磁盘I/O、CPU、内存等各方面因素造成复制的延迟。这一般发生在高并发大数据量写入场景中。

问题三：慢SQL语句过多。

假如一条SQL语句执行时间是20秒，那么从执行完毕到从库上能查到数据至少需要20秒，这样就延迟20秒了。

一般要把SQL语句的优化作为常规工作，不断地进行监控和优化，如果单个SQL的写入时间长，可以修改后分多次写入。通过查看慢查询日志或show full processlist命令，找出执行时间长的查询语句或大的事务。

问题四：主从复制的设计问题。

例如，主从复制单线程，如果主库写并发太大，来不及传送到从库，就会导致延迟。

更高版本的MySQL可以支持多线程复制，门户网站则会自己开发多线程同步功能。

问题五：主从库之间的网络延迟。

主从库的网卡、网线、连接的交换机等网络设备都可能成为复制的瓶颈，导致复制延迟，另外，跨公网主从复制很容易导致主从复制延迟。

问题六：主库读写压力大，导致复制延迟。

主库硬件要搞好一点，架构的前端要加buffer及缓存层。

5.通过read-only参数让从库只读访问

read-only参数选项可以让从服务器只允许来自从服务器线程或具有SUPER权限的数据库用户进行更新，确保从服务器不接受来自用户端的非法用户更新。

read-only参数允许数据库更新的条件为：

·具有SUPER权限的用户可以更新，不受read-only参数影响，例如：管理员root。

·来自从服务器线程可以更新，不受read-only参数影响，例如：前文的rep用户。

在生产环境中，可以在从库Slave中使用read-only参数，确保从库数据不被非法更新。

read-only参数的配置方法如下。

方法一：直接带--read-only参数启动或重启数据库，使用

```
killall mysqld
```

或

```
mysqladmin -uroot -poldboy123 -S /data/3307/mysql.sock shutdown
mysqld_safe --defaults-file=/data/3307/my.cnf --read-only &
```

方法二：在my.cnf里[mysqld]模块下加read-only参数重启数据库，配置如下：

```
[mysqld]
read-only
```

6.Web用户专业设置方案：MySQL主从复制读写分离集群

专业的运维人员提供给开发人员读写分离的账户设置方法如下：

1）访问主库和从库时使用一套用户密码，例如，用户为web，密码为oldboy123。

2）即使访问IP不同，端口也尽量相同（3306）。例如：写库VIP为10.0.0.7，读库VIP为10.0.0.8。

除了IP没办法修改之外，要尽量为开发人员提供方便，如果数据库前端有DAL层（DBProxy），还可以只给开发人员一套用户、密码、IP、端口，这样就更专业了，剩下的都由运维人员搞定。

下面是授权Web连接用户访问的方案：MySQL主从复制读写分离集群。

方法1：主库和从库使用不同的用户，授予不同的权限。

主库上对web_w用户授权如下：

用户：

web_w 密码：

oldboy123 端口：

3306 主库

VIP：

10.0.0.7权限：

SELECT，

 INSERT，

 UPDATE，

 DELETE命令：

GRANT SELECT，

 INSERT，

 UPDATE，

 DELETE ON `web`.* TO 'web_w'@'10.0.0.%' identified by 'oldboy123';

从库上对web_r用户授权如下：

用户：

web_r 密码：

oldboy123 端口：

3306 从库

VIP：

10.0.0.8权限：

SELECT命令：

```
GRANT SELECT ON `web`.* TO 'web_r'@'10.0.0.%' identified by 'oldboy123';
```

 提示：此法显得不够专业，但是可以满足开发需求。

方法2：主库和从库使用相同的用户，但授予不同的权限。

主库上对web用户授权如下：

用户：

web 密码：

oldboy123 端口：

3306 主库

VIP：

10.0.0.7权限：

SELECT，

INSERT,

UPDATE,

DELETE命令：

GRANT SELECT,

INSERT,

UPDATE,

DELETE ON `web`.* TO 'web'@'10.0.0.%' identified by 'oldboy123';

从库上对web用户授权如下：

用户：

web 密码：

oldboy123 端口：

3306 主库

VIP：

10.0.0.8权限：

```
SELECT提示：由于主库和从库是同步复制的，所以从库上的

web用户会自动和主库保持一致，即无法实现只读

select的授权
```

要实现方法2中的授权方案，有如下两个方法。

一是在主库上创建用户和权限后，从库上revoke收回对应更新权限（insert、update、delete）。

命令为：

```
REVOKE INSERT,

UPDATE,

DELETE ON web.* FROM 'web'@'10.0.0.%';
```

二是忽略授权库MySQL同步，主库的配置参数如下：

```
binlog-ignore-db = mysql
replicate-ignore-db = mysql
```

提示：上面参数等号两边必须有空格。

方法3：在从库上设置read-only参数，让从库只读。

主库从库：主库和从库使用相同的用户，授予相同的权限（非ALL权限）。

用户：

web 密码：

oldboy123 端口：

3306 主库

VIP：

10.0.0.7，从库

VIP：

10.0.0.8权限：

SELECT，

 INSERT，

 UPDATE，

 DELETE命令：

GRANT SELECT，

```
INSERT,

UPDATE,

DELETE ON web.* TO 'web_w'@'10.0.0.%' identified by 'oldboy123';
```

由于从库设置了read-only，非super权限是无法写入的，因此，通过read-only参数就可以很好地控制用户，使其不能非法将数据写入从库。

老男孩生产工作场景的设置方案如下：

1）忽略授权库MySQL同步，主库配置参数如下：

```
binlog-ignore-db = mysql
replicate-ignore-db = mysql
```

 提示：上面参数等号两边必须有空格。

2）主库和从库使用相同的用户，但授予不同的权限。

主库上对web用户授权如下：

```
用户：

web 密码：
```

oldboy123 端口：

3306 主库

VIP：

10.0.0.7权限：

SELECT，

 INSERT，

 UPDATE，

 DELETE命令：

GRANT SELECT，

 INSERT，

 UPDATE，

 DELETE ON web.* TO 'web'@'10.0.0.%' identified by 'oldboy123'；

从库上对web用户授权如下：

用户：

web 密码：

oldboy123 端口：

3306 主库

VIP：

10.0.0.8权限：

SELECT

3）从库设置read-only，增加双保险。

9.10 本章重点回顾

1）MySQL多实例的实现原理及实战部署。

2）MySQL主从复制的原理（面试中经常会问到）。

3）MySQL主从复制的实践。

4）MySQL主从复制故障解决思路。

5）MySQL主从复制延迟原因及解决思路。

6）MySQL主从复制集群，从库备份的思想和思路。

7）MySQL主从复制读写分离授权访问用户方案。

9.11 本章参考资料

·MySQL官方手册5.1及5.5

·MySQL数据库企业级核心知识精品

http://edu.51cto.com/course/course_id-4058.html

·Heartbeat+DRBD+MySQL高可用架构方案与实施过程细节

http://oldboy.blog.51cto.com/2561410/1240412

·MySQL数据库企业级应用实战

http://edu.51cto.com/pack/view/id-214.html

第10章　企业级NFS网络文件共享服务

10.1　NFS介绍

10.1.1　什么是NFS

NFS是Network File System的缩写，中文意思是网络文件系统。它的主要功能是通过网络（一般是局域网）让不同的主机系统之间可以共享文件或目录。NFS客户端（一般为应用服务器，例如Web）可以通过挂载（mount）的方式将NFS服务器端共享的数据目录挂载到NFS客户端本地系统中（就是某一个挂载点下）。从客户端本地看，NFS服务器端共享的目录就好像是客户端自己的磁盘分区或者目录一样，而实际上却是远端的NFS服务器的目录。

NFS网络文件系统很像Windows系统的网络共享、安全功能、网络驱动器映射，这也与Linux系统里的samba服务类似。只不过一般情况下，Windows网络共享服务或samba服务用于办公局域网共享，而互联网中小型网站集群架构后端常用NFS进行数据共享，如果是大型网站，那么有可能还会用到更复杂的分布式文件系统，例如：
Moosefs（mfs）、GlusterFS、FastDFS，这些不在本书讨论内容之列，有兴趣的读者可以阅读老男孩的其他书或者相关教学视频。

10.1.2 NFS的历史介绍

第一个网络文件系统被称为File Access Listener，由Digital Equipment Corporation（DEC）在1976年开发。

NFS是第一个构建于IP协议之上的现代网络文件系统。在20世纪80年代，它首先作为实验的文件系统，由Sun Microsystems在内部完成开发。NFS协议归属于Request for Comments（RFC）标准，并且随后演化为NFSv2。作为一个标准，由于NFS与其他客户端和服务器的互操作能力很好而发展快速。

之后，标准继续演化，成为NFSv3，在RFC1813中有定义。这一新的协议比以前的版本具有更好的可扩展性，支持大文件（超过2GB），异步写入，并且将TCP作为传输协议，为文件系统在更广泛的网络中使用铺平了道路。在2000年，RFC 3010（由RFC 3530修订）将NFS带入企业级应用。此时，Sun引入了具有较高安全性、带有状态协议的NFSv4（NFS之前的版本都是无状态的）。今天，NFS版本的4.1（由RFC 5661定义）增加了对跨越分布式服务器并行访问的支持（称为PNFS extension）。

NFS系统发展的进程，包括记录其特性的特定RFC，可用图10-1来展示。

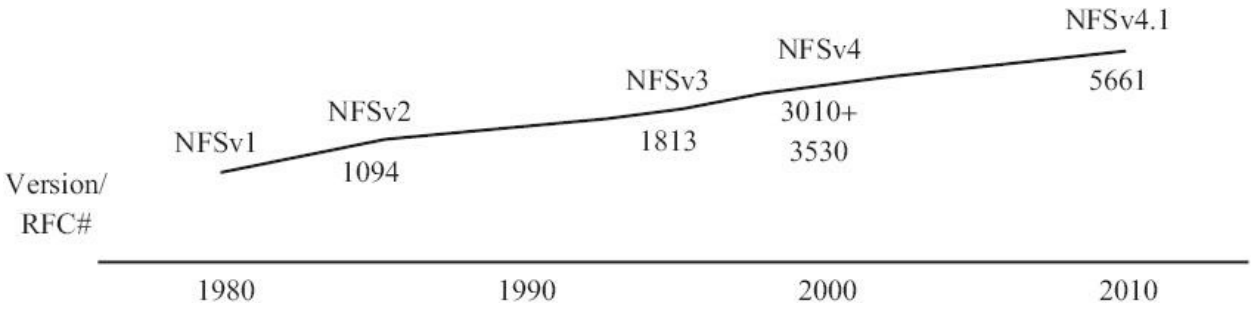


图10-1　NFS协议发展进程

NFS系统已经历了近30年的发展。它代表了一个非常稳定的（及可移植）网络文件系统，具备可扩展、高性能等特性，并达到了企业级应用质量标准。由于网络速度的增加和延迟的降低，NFS系统一直是通过网络提供文件系统服务的有竞争力的选择，特别是在中小型互联网企业中，应用十分广泛。

10.1.3　NFS在企业中的应用场景

在企业集群架构的工作场景中，NFS网络文件系统一般被用来存储共享视频、图片、附件等静态资源文件，通常网站用户上传的文件都会放到NFS共享里，例如：BBS产品的图片、附件、头像（注意网站BBS程序不要放在NFS共享里），然后前端所有的节点访问这些静态资源时都可读取NFS存储上的资源。NFS是当前互联网系统架构中最常用的数据存储服务之一，前面说过，中小型网站公司应用频率更高，大公司或门户除了使用NFS外，还可能会使用更为复杂的分布式文件系统，比如Moosefs（mfs）、GlusterFS、FastDFS等。

在企业生产集群架构中，图10-2右边的实线框里带圆点的就是NFS系统的工作位置，NFS作为所有前端Web服务的共享存储，内容一般包括网站用户上传的图片、附件、头像等，注意，网站的程序代码不要放NFS共享里，因为网站程序是开发运维人员统一发布的，不存在发布延迟问题，直接批量发布到Web节点提供访问比共享到NFS里访问效率更高。

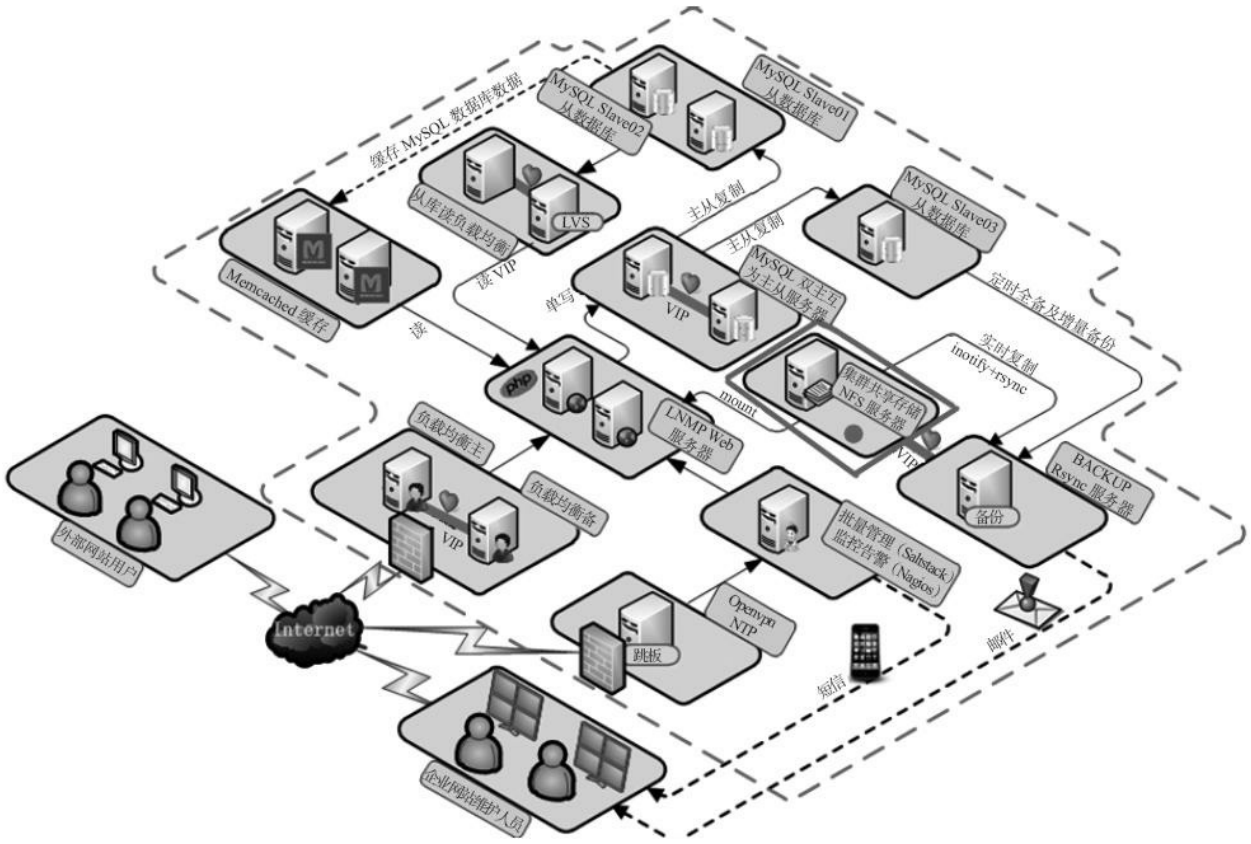


图10-2　NFS服务在企业生产集群架构中的位置

10.1.4　企业生产集群为什么需要共享存储角色

这里通过图解给大家展示一下集群架构需要共享存储服务的理由。例如：A用户传图片到Web1服务器，然后让B用户访问这张图片，结果B用户访问的请求分发到了Web2，因为Web2上没有这张图片，这就导致它无法看到A用户上传的图片，如果此时有一个共享存储，A用户上传图片的请求无论是分发到Web1还是Web2上，最终都会存储到共享存储上，而在B用户访问图片时，无论请求分发到Web1还是Web2上，最终也都会去共享存储上找，这样就可以访问到需要的资源了。这个共享存储的位置可以通过开源软件和商业硬件实现，互联网中小型集群架构会用普通PC服务器配置NFS网络文件系统实现。

当集群中没有NFS共享存储时，用户访问图片的情况如图10-3所示。

如果集群中有NFS共享存储，用户访问图片的情况如图10-4所示。

中小型互联网企业一般不会买硬件存储，因为太贵，大公司如果业务发展很快的话，可能会临时买硬件存储顶一下网站的压力，当网站并发继续加大时，硬件存储的扩展相对就会很费劲，且价格成几何级数增加。例如：淘宝网就曾替换掉了很多硬件设备，比如，用LVS+Haproxy替换了netscaler负载均衡设备，用FastDFS、TFS配合PC服务器替换了

netapp、emc等商业存储设备，去IOE正在成为互联网公司的主流。

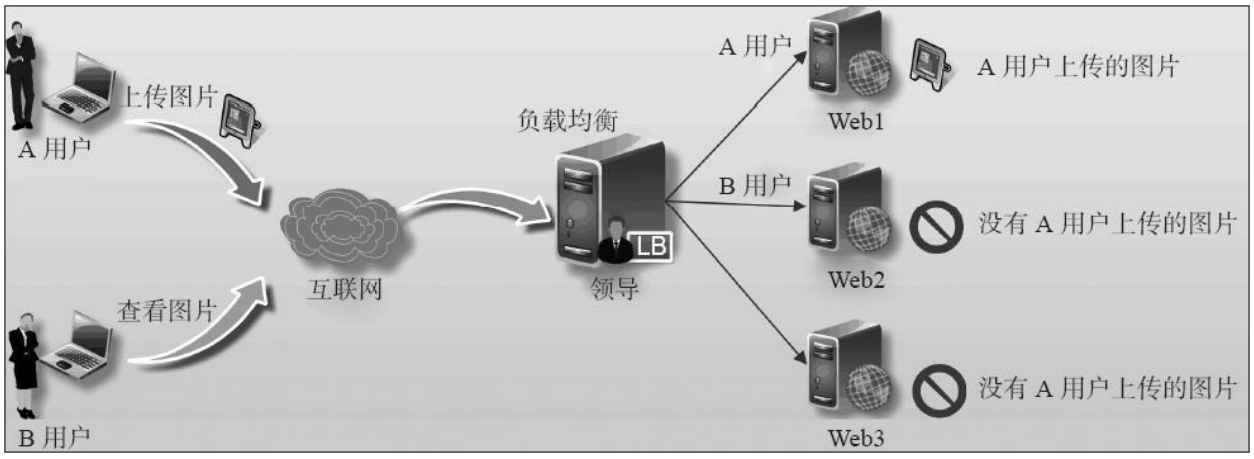


图10-3　企业生产集群没有NFS共享存储访问示意图

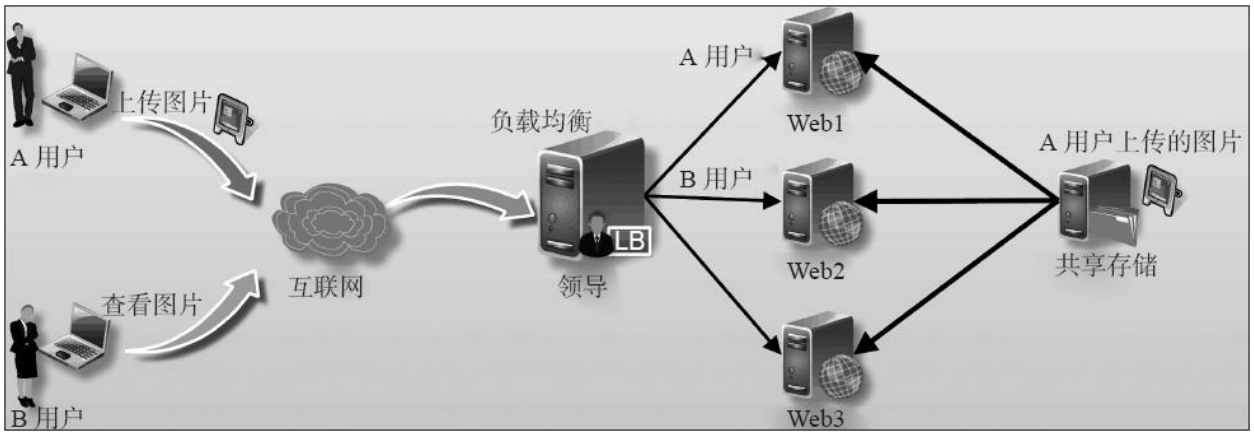


图10-4　企业生产集群有NFS共享存储访问示意图

10.2　NFS系统原理介绍

10.2.1　NFS系统挂载结构图解与介绍

图10-5是企业工作中的NFS服务器与客户端挂载情况结构。

在图10-5中，在NFS服务器端设置好一个共享目录/video后，其他有权限访问NFS服务器端的客户端都可以将这个共享目录/video挂载到客户端本地的某个挂载点（其实就是一个目录，这个挂载点目录可以自己随意指定），图10-5中的两个NFS客户端本地的挂载点分别为/v/video和/video，不同客户端的挂载点可以不相同。

客户端正确挂载完毕后，就可以通过NFS客户端的挂载点所在的/v/video或/video目录查看到NFS服务器端/video共享出来的目录下的所有数据。在客户端上查看时，NFS服务器端的/video目录就相当于客户端本地的磁盘分区或目录，几乎感觉不到使用上的区别，根据NFS服务器端授予的NFS共享权限以及共享目录的本地系统权限，只要在指定的NFS客户端操作挂载/v/video或/video的目录，就可以将数据轻松地存取到NFS服务器端上的/video目录中了。

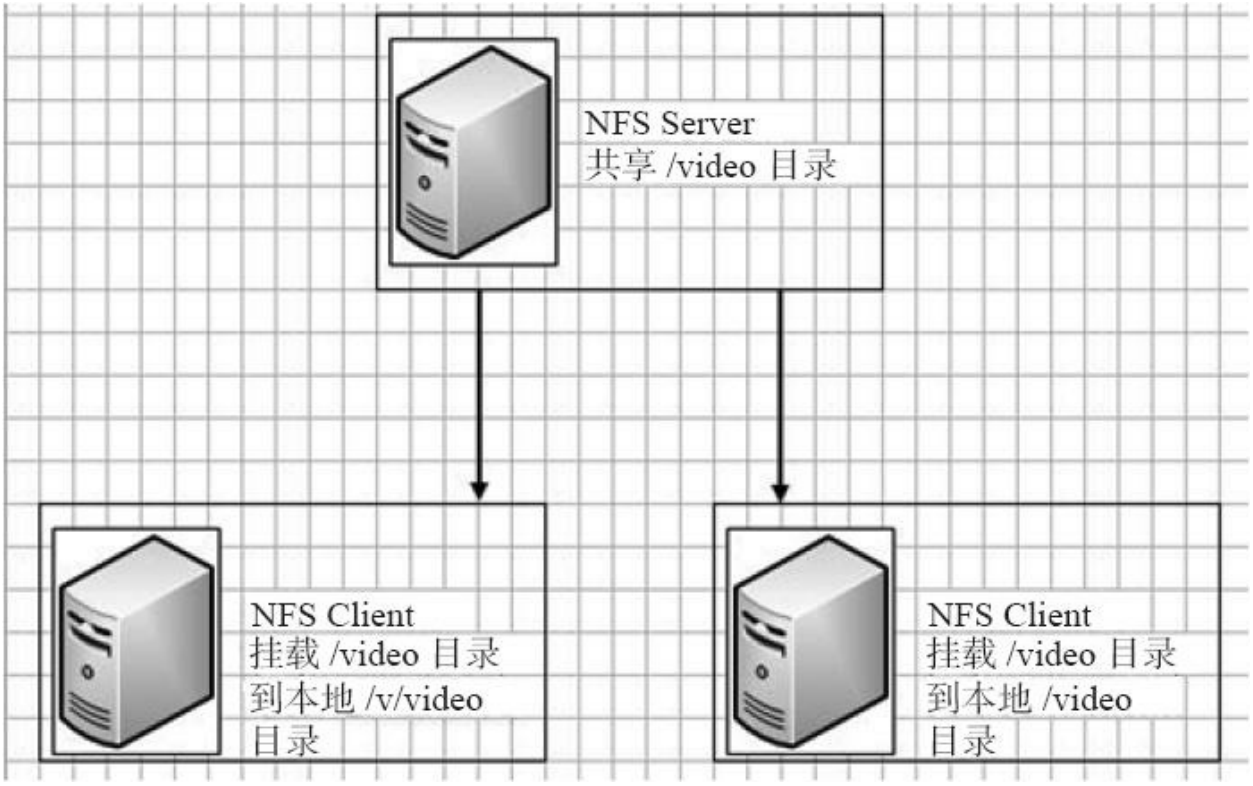


图10-5　NFS服务器共享与客户端挂载结构图

客户端挂载NFS后，本地挂载基本信息显示如下：

```
[root@nfs-client ~]# df -h
Filesystem               Size  Used Avail Use% Mounted on
/dev/sda1               1.1T  467G  544G  47% /
tmpfs                 7.9G     0  7.9G   0% /dev/shm
10.0.0.7:


/video  1002G  59G  892G  7% /video  #<==10.0.0.7为


nfs server的


ip地址


提示：
```

```
mount 源

 目标

    mount 10.0.0.7:

/video /video
```

从挂载信息来看，与本地的磁盘分区几乎没什么差别，只是文件系统对应列的开头是IP地址的形式了。

经过前面的介绍，我们知道NFS系统是通过网络来进行数据传输的（所以叫做网络文件系统），因此，NFS会使用一些端口来传输数据，那么，NFS到底使用哪些端口来进行数据传输呢？图10-6是NFS服务两次重启向RPC服务注册的端口列表结果对比。

由上面的实际测试得知，NFS在传输数据时使用的端口会随机选择。可能有读者会纳闷，既然这样，NFS客户端是怎么知道NFS服务器端使用的是哪个端口呢？

答案：就是通过RPC（中文意思远程过程调用，英文Remote Procedure Call简称RPC）协议/服务来实现，这个RPC服务的应用在门户级的网站有很多，例如：百度。下面就来谈谈什么是RPC协议/服务。

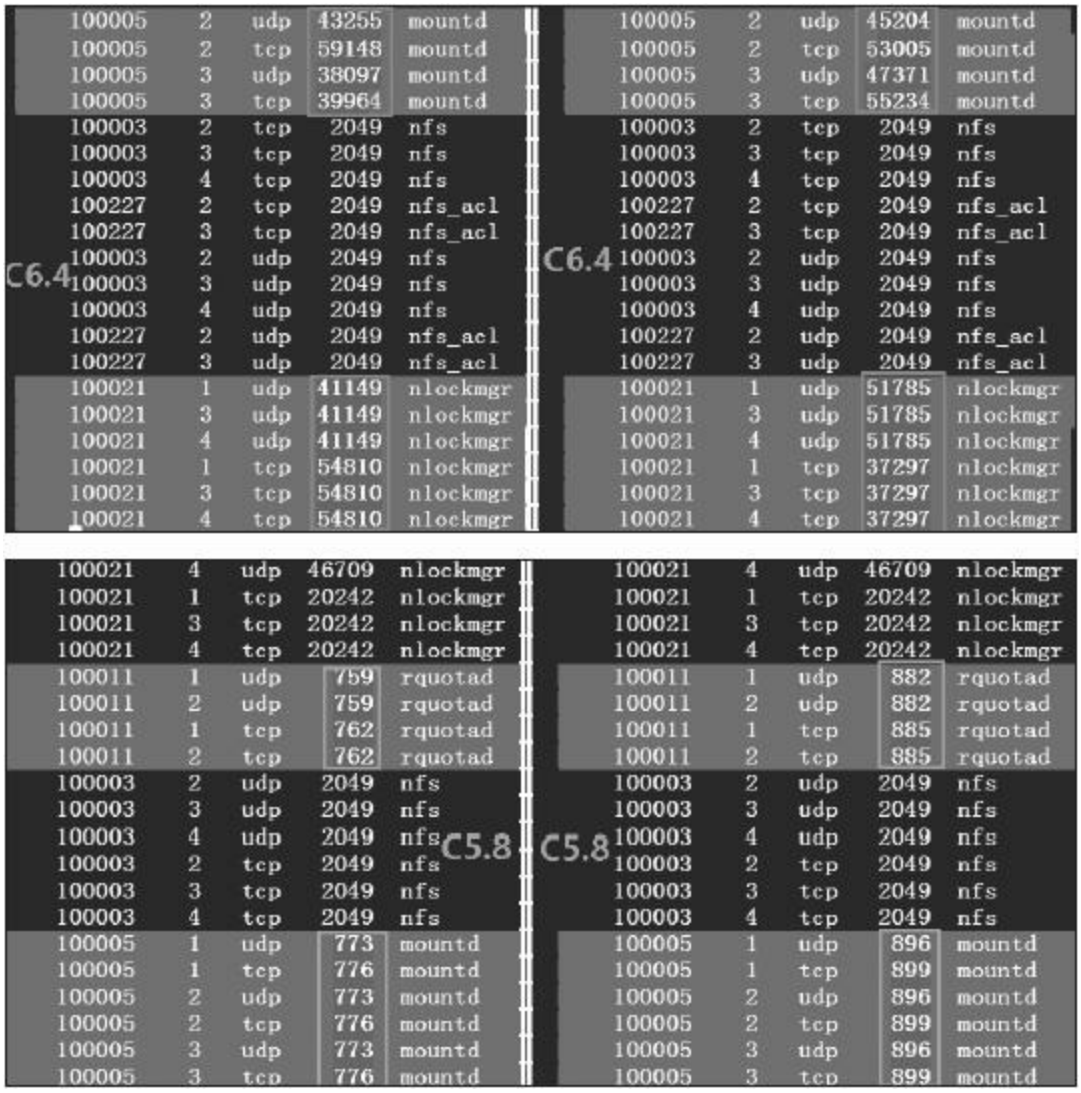


图10-6　NFS服务启动后的端口对比

10.2.2　什么是RPC

因为NFS支持的功能相当多，而不同的功能都会使用不同的程序来启动，每启动一个功能就会启用一些端口来传输数据，因此，NFS的功能所对应的端口无法固定，它会随机取用一些未被使用的端口来作为传输之用，其中CentOS 5.x的随机端口都小于1024，而CentOS 6.x的随机端口都是较大的，见图10-6。

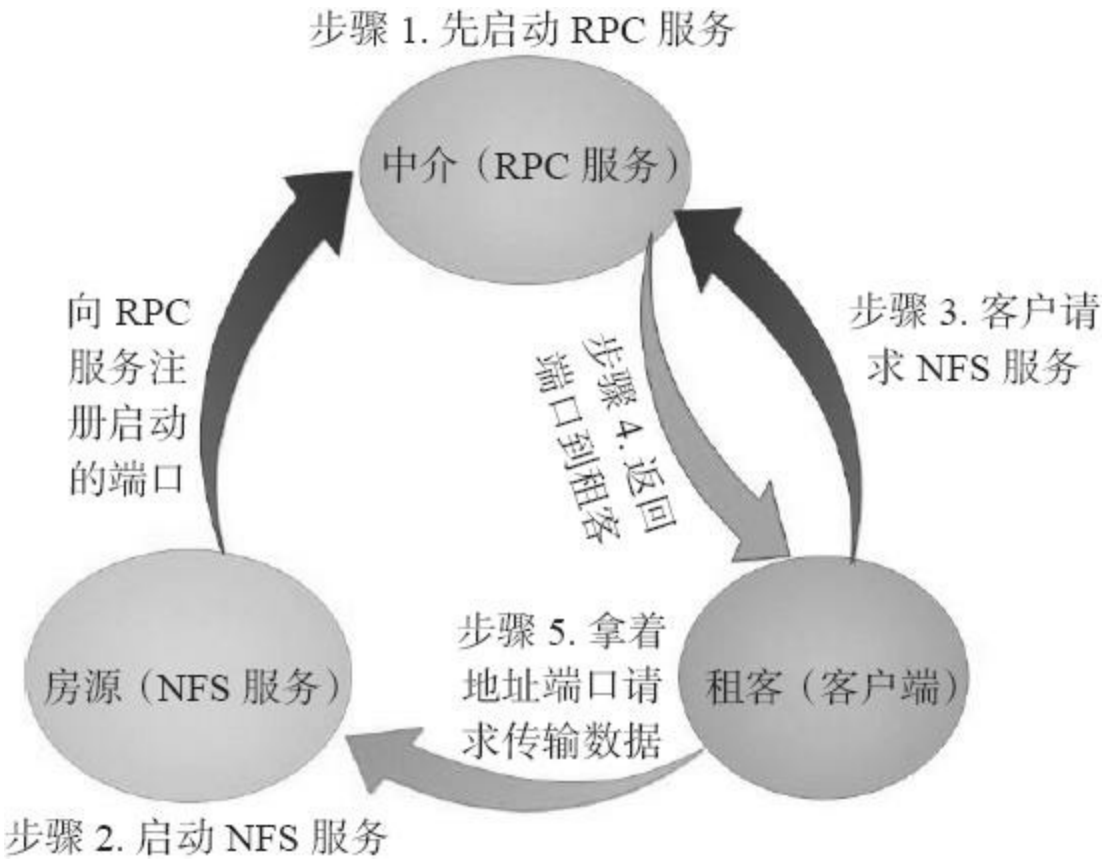


图10-7　NFS工作流程图

因为端口不固定，这样一来就会造成NFS客户端与NFS服务器端的通信障碍，因为NFS客户端必须要知道NFS服务器端的数据传输端口才能进行通信，才能交互数据。

要解决上面的困扰，就需要通过远程过程调用RPC （Remote Procedure Call）服务 来帮忙了，NFS的RPC服务最主要的功能就是记录每个NFS功能所对应的端口号，并且在NFS客户端请求时将该端口和功能对应的信息传递给请求数据的NFS客户端，从而确保客户端可以连接到正确的NFS端口上去，达到实现数据传输交互数据目的。这个RPC服务类似NFS服务器端和NFS客户端之间的一个中介，流程如图10-7所示。

拿房屋中介打个比喻吧：假设我们要找房子，这里的我们就相当于NFS客户端，中介介绍房子，就相当于RPC服务，房子所有者房东就相当于NFS服务，租房的人找房子，就要找中介，中介要预先存有房子主人的信息，才能将房源信息告诉租房的人。

那么RPC服务如何知道每个NFS的端口呢？

当NFS服务器端启动服务时会随机取用若干端口，并主动向RPC服务注册取用的相关端口及功能信息，如此一来，RPC服务就知道NFS每个端口对应的NFS功能了，然后RPC服务使用固定的111端口来监听NFS客户端提交的请求，并将正确的NFS端口信息回复给请求的NFS客户端，这样一来，NFS客户端就可以与NFS服务器端进行数据传输了。

在启动NFS Server之前，首先要启动RPC服务（CentOS 5.8下为portmap服务，CentOS 6.6下为rpcbind服务，下同），否则NFS Server就无法向RPC服务注册了。另外，如果RPC服务重新启动，原来已经注册好的NFS端口数据就会丢失，因此，此时RPC服务管理的NFS程序也需要重新启动以重新向RPC注册。要特别注意的是，一般修改NFS配置文件后，是不需要重启NFS的，直接在命令行执行/etc/init.d/nfs reload或exportfs-rv即可使修改的/etc/exports生效。

10.2.3　NFS的工作流程原理

前文描述的整个启动过程如图10-8所示。

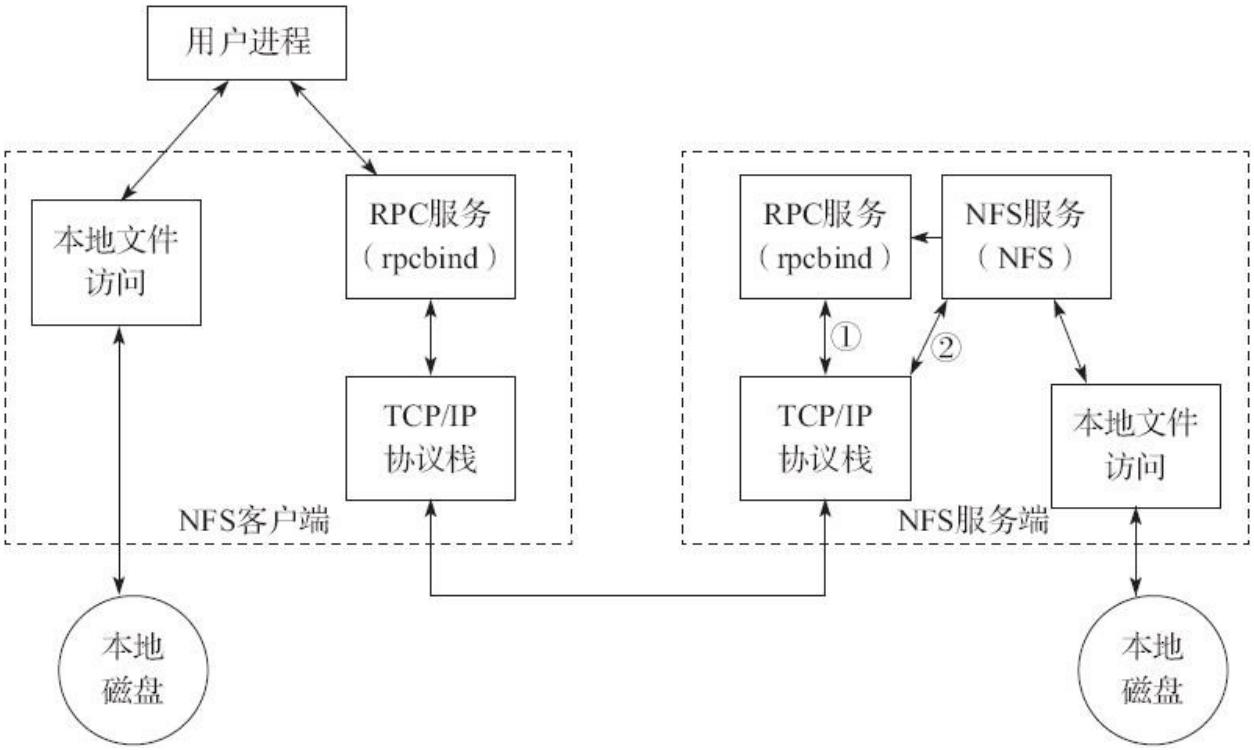


图10-8　NFS工作原理流程简图

当访问程序通过NFS客户端向NFS服务器端存取文件时，其请求数据流程大致如下：

1）首先用户访问网站程序，由程序在NFS客户端上发出存取NFS文件的请求，这时NFS客户端（即执行程序的服务器）的RPC服务（rpcbind服务）就会通过网络向NFS服务器端的RPC服务（rpcbind服

务）的111端口发出NFS文件存取功能的询问请求。

2）NFS服务器端的RPC服务（rpcbind服务）找到对应的已注册的NFS端口后，通知NFS客户端的RPC服务（rpcbind服务）。

3）此时NFS客户端获取到正确的端口，并与NFS daemon联机存取数据。

4）NFS客户端把数据存取成功后，返回给前端访问程序，告知用户存取结果，作为网站用户，就完成了一次存取操作。

因为NFS的各项功能都需要向RPC服务（rpcbind服务）注册，所以只有RPC服务才能获取到NFS服务的各项功能对应的端口号（port number）、PID、NFS在主机所监听的IP等信息，而NFS客户端也只能通过向RPC服务询问才能找到正确的端口。也就是说，NFS需要有RPC服务的协助才能成功对外提供服务。从上面的描述，我们不难推断，无论是NFS客户端还是NFS服务器端，当要使用NFS时，都需要首先启动RPC服务，NFS服务必须在RPC服务启动之后启动，客户端无需启动NFS服务，但需要启动RPC服务。


注意：NFS的RPC服务，在CentOS 5.X下名称为portmap，在CentOS 6.X下名称为rpcbind。

更多的NFS相关信息可以参考：

http://www.tldp.org/HOWTO/NFS-HOWTO/indes.html

http://www.citi.umich.edu/projects/nfsv4/linux/

http://www.vanemery.com/Linux/NFSv4/NFSv4-no-rpcsec.html

http://www.ibm.com/developerworks/cn/linux/l-network-filesystems/

10.3　NFS服务器端部署环境准备

1.NFS服务部署服务器准备

表10-1给出了部署NFS服务的准备信息，包括系统版本、各服务角色、IP地址等。

表10-1　NFS服务部署服务器角色IP列表

服务器系统	角　　色	IP
CentOS 6.6 x86_64	NFS 服务器端（nfs-server）	10.0.0.7
CentOS 6.6 x86_64	NFS 客户端 1（nfs-client1）	10.0.0.8
CentOS 6.6 x86_64	NFS 客户端 2（nfs-client2）	10.0.0.9

2.CentOS 6.6 x86_64模拟环境信息

NFS服务器端操作系统及内核版本信息如下：

```
[root@nfs-server ~]# cat /etc/redhat-release
CentOS release 6.6 (


Final)



[root@nfs-server ~]# uname -r
2.6.32-504.el6.x86_64
[root@nfs-server ~]# uname -m
x86_64
```

NFS客户端操作系统及内核版本信息如下：

```
[root@nfs-client1 ~]# cat /etc/redhat-release
CentOS release 6.6 (


Final)



[root@nfs-client1 ~]# uname -r
2.6.32-504.el6.x86_64
[root@nfs-client1 ~]# uname -m
x86_64
```

提示：经常有朋友和学生抱怨，在网上找到的文档都不好用，照着做都做不出来。我想其中的一个原因可能就是网上的文档大部分未留下当前系统环境，导致读者操作时因环境不一致而出现问题。当然也与所写内容是不是真的在生产环境经受过考验有直接关系，此外，网上的文章大多都是跳跃性的，作者自己已会的就会不自觉地略过，写得不是那么详细。建议读者多考虑有教学经验的作者写的书，一般对于知识的描述会好很多。

10.4　NFS服务器端的设置

10.4.1　NFS软件列表

要部署NFS服务，需要安装下面的软件包：

·nfs-utils：NFS服务的主程序，包括rpc.nfsd、rpc.mountd这两个daemons和相关文档说明，以及执行命令文件等。

·rpcbind：CentOS 6.X下面RPC的主程序。NFS可以视为一个RPC程序，在启动任何一个RPC程序之前，需要做好端口和功能的对应映射工作，这个映射工作就是由rpcbind服务来完成的。因此，在提供NFS服务之前必须先启动rpcbind服务才行。


注意：有关RPC协议知识这里大家不必细究，详细说明可见本章结尾命令部分。

10.4.2　查看NFS软件包

可使用如下命令查看默认情况下CentOS 5.8/6.6里NFS软件的安装情况。

```
[root@nfs-server ~]# rpm -aq nfs-utils rpcbind    #←这个不用管道的命令更有效率。
```

当不知道软件名字时，可以用rpm-aq|grep-E"nfs-|rpcbind"来过滤包含在引号内的字符串。grep-E在这里相当于egrep。grep、egrep这两个命令在运维工作中非常常用并且很好用，需要读者掌握。

CentOS 6.6默认没有安装NFS软件包（CentOS 5默认会安装），可以使用yum install nfs-utils rpcbind-y命令来安装NFS软件。

```
[root@nfs-server ~]# yum install nfs-utils rpcbind -y...忽略软件显示信息...


[root@nfs-server ~]# rpm -aq nfs-utils rpcbind
rpcbind-0.2.0-11.el6.x86_64            #←这个包在


5.8名字为


portmap。


nfs-utils-1.2.3-54.el6.x86_64
```

如果出现rpcbind和nfs-utils开头的两个软件包，表示NFS服务器端软件安装完毕。

10.4.3　启动NFS相关服务

1.启动rpcbind服务

因为NFS及其辅助程序都是基于RPC（Remote Procedure Call）协议的（使用的端口为111），所以首先要确保系统中运行了rpcbind服务。有关RPC协议的介绍请见后文。启动的实际操作如下（是CentOS 6.6环境下的操作）：

```
[root@nfs-server ~]# LANG=en  #<==临时调整系统为英文字符集，便于

grep过滤。

[root@nfs-server ~]# /etc/init.d/rpcbind status  #<==检查

rpcbind 服务状态

rpcbind is stopped
[root@nfs-server ~]# rpcinfo -p localhost       #<==rpcbind 服务未启动检查

rpcinfo信息的报错

rpcinfo:

 can't contact portmapper:
```

RPC：

Remote system error - Connection refused
[root@nfs-server ~]# /etc/init.d/rpcbind start   #<==启动

rpcbind 服务

Starting rpcbind：

[OK]
[root@nfs-server ~]# /etc/init.d/rpcbind status
rpcbind （

pid 27835）

is running...
[root@nfs-server ~]# lsof -i ：

111
COMMAND PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
rpcbind 1276 rpc   6u IPv4 10665   0t0 UDP *：

sunrpc
rpcbind 1276 rpc   8u IPv4 10668   0t0 TCP *：

sunrpc （

LISTEN）

rpcbind 1276 rpc   9u IPv6 10670   0t0 UDP *：

sunrpc
rpcbind 1276 rpc   11u IPv6 10673   0t0 TCP *：

```
sunrpc  (


LISTEN)



[root@nfs-server ~]# netstat -lntup|grep rpcbind
tcp     0     0 0.0.0.0:


111          0.0.0.0:


*     LISTEN     1276/rpcbind
tcp     0     0 : : :


111          : : :


*          LISTEN     1276/rpcbind
udp     0     0 0.0.0.0:


603          0.0.0.0:


*          1276/rpcbind
udp     0     0 0.0.0.0:


111          0.0.0.0:


*          1276/rpcbind
udp     0     0 : : :


603          : : :


*               1276/rpcbind
udp     0     0 : : :
```

111   : : :

*   1276/rpcbind
[root@nfs-server ~]# rpcinfo -p localhost #<==查看

NFS服务向

rpc服务注册的端口信息，因为

NFS服务还没起，因此，没太多注册的端口映射信息。请注意和后面启动

NFS服务后这里的对比。

```
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
#提示：
```

111端口为

rpcbind 服务对外提供服务的主端口。

[root@nfs-server ~]# chkconfig --list rpcbind <==检查

rpcbind开机是否自启动了

rpcbind   0：

off   1：

```
off   2:

on    3:

on    4:

on    5:

on    6:

off
```

我们看到结果显示rpcbind服务已处于运行状态，并且随机器开机自启动。

提示：如果rpcbind服务未启动，执行命令rpcinfo-p localhost检查时，会报如下错误。

```
rpcinfo:

 can't contact portmapper:

 RPC:

 Remote system error - Connection refused
```

中文解释为：rpcinfo：无法同portmapper交互：RPC：远程系统错

误-拒绝连接。

解决办法：执行/etc/init.d/rpcbind start启动rpcbind服务即可。

2.启动NFS服务

启动的命令如下：

```
[root@nfs-server ~]# LANG=en
[root@nfs-server ~]# /etc/init.d/rpcbind status     #<==检查


rpcbind服务状态



rpcbind （


pid  996）


 is running...
[root@nfs-server ~]# /etc/init.d/nfs status     #<==检查


NFS服务状态



rpc.svcgssd 已停



rpc.mountd is stopped
nfsd is stopped
rpc.rquotad is stopped
[root@nfs-server ~]# /etc/init.d/nfs start     #<==启动


NFS 服务
```

```
Starting NFS services:

                                           [  OK  ]
Starting NFS quotas:

                                             [  OK  ]
Starting NFS mountd:

                                             [  OK  ]
Starting NFS daemon:

                                             [  OK  ]正在启动

 RPC idmapd:


[root@nfs-server ~]# /etc/init.d/nfs status     #<==检查

NFS服务状态


rpc.svcgssd 已停


rpc.mountd （

pid 1233）

 is running...
nfsd （

pid 1249 1248 1247 1246 1245 1244 1243 1242）
```

```
 is running...
rpc.rquotad （

pid 1228）

 is running...
[root@nfs-server ~]# rpcinfo -p localhost     #<==查看
```

NFS服务向

rpc服务注册的端口信息，

NFS服务启动后，我们发现信息很多了，有很多端口和功能的对应。

```
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  45557  status
    100024    1   tcp  55422  status
    100011    1   udp    875  rquotad
    100011    2   udp    875  rquotad
    100011    1   tcp    875  rquotad
    100011    2   tcp    875  rquotad
    100005    1   udp  33050  mountd
    100005    1   tcp  33405  mountd
    100005    2   udp  34531  mountd
    100005    2   tcp  60545  mountd
    100005    3   udp  54527  mountd
    100005    3   tcp  48669  mountd
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100227    2   tcp   2049  nfs_acl
    100227    3   tcp   2049  nfs_acl
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    100227    2   udp   2049  nfs_acl
    100227    3   udp   2049  nfs_acl
    100021    1   udp  39036  nlockmgr
    100021    3   udp  39036  nlockmgr
    100021    4   udp  39036  nlockmgr
    100021    1   tcp  38820  nlockmgr
    100021    3   tcp  38820  nlockmgr
```

```
    100021    4   tcp  38820  nlockmgr
```

10.4.4　NFS服务常见进程详解

从上面NFS服务启动的过程可以看出，运行NFS服务默认需要启动的服务或进程至少有：NFS quotas（rpc.rquotad）、NFS daemon（nfsd）、NFS mountd（rpc.mountd）。可以通过执行如下命令查看启动NFS后，系统中运行的NFS相关进程：

```
[root@nfs-server ~]# ps -ef|egrep "rpc|nfs"
rpc       996     1 0 16:


27       00:


00:


00 rpcbind
rpcuser 1016     1 0 16:


27       00:


00:


00 rpc.statd      #      <==检查文件一致性


root     1219     2 0 16:


29       00:
```

```
00:


00 [rpciod/0]
root    1552    1 0 16:


31      00:


00:


00 rpc.rquotad     #     <==磁盘配额进程（


remote quota server）


root    1557    1 0 16:


31      00:


00:


00 rpc.mountd     #     <==权限管理验证等（


NFS mount daemon）


root    1564    2 0 16:


31      00:


00:


00 [nfsd4]
root    1565    2 0 16:
```

```
31      00:


00:


00 [nfsd4_callbacks]
root    1566    2 0 16:


31      00:


00:


00 [nfsd]     #     <==NFS主进程


root    1567    2 0 16:


31      00:


00:


00 [nfsd]     #     <==NFS主进程


root    1568    2 0 16:


31      00:


00:


00 [nfsd]
root    1569    2 0 16:
```

```
31      00:

00:

00 [nfsd]     #     <==NFS主进程，管理登入，

ID身份判别等。

root    1570    2 0 16:

31      00:

00:

00 [nfsd]
root    1571    2 0 16:

31      00:

00:

00 [nfsd]
root    1572    2 0 16:

31      00:

00:

00 [nfsd]
root    1573    2 0 16:
```

```
31      00:

00:

00 [nfsd]     #     <==NFS主进程

root    1600    1 0 16:

31      00:

00:

00 rpc.idmapd     #     <==name mapping daemon
root    1638 1119 0 16:

34 pts/0     00:

00:

00 egrep rpc|nfs
```

NFS服务的主要任务是共享文件系统数据，而文件系统数据的共享离不开权限问题。所以NFS服务器启动时最少需要两个不同的进程，一个是管理NFS客户端是否能够登入的rpc.nfsd主进程，另一个用于管理NFS客户端是否能够取得对应权限的rpc.mountd进程。如果还需要管理磁盘配额，则NFS还要再加载rpc.rquotad进程。表10-2给出了NFS服务启动的进程说明。

表10-2　NFS服务启动的进程说明

服务或进程名	用途说明
nfsd（rpc.nfsd）	rpc.nfsd 的主要功能是管理 NFS 客户端是否能够登入 NFS 服务器端主机，其中还包含登入者的 ID 判别等
mountd（rpc.mountd）	rpc.mountd 的主要功能则是管理 NFS 文件系统。当 NFS 客户端顺利通过 rpc.nfsd 登入 NFS 服务器端主机时，在使用 NFS 服务器提供数据之前，它会去读 NFS 的配置文件 /etc/exports 来比对 NFS 客户端的权限，通过这一关之后，还要经过 NFS 服务器端本地文件系统使用权限（就是 owner、group、other 权限）的认证程序。如果都通过了，NFS 客户端就可以取得使用 NFS 服务器端文件的权限。注意，这个 /etc/exports 文件也是我们用来管理 NFS 共享目录的使用权限与安全设置的地方，特别强调，NFS 本身设置的是网络共享权限，整个共享目录的权限还和目录自身的系统权限有关
rpc.lockd（非必要）	可用来锁定文件，用于多客户端同时写入
rpc.statd（非必要）	检查文件的一致性，与 rpc.lockd 有关。c、d 两个服务需要客户端、服务器端同时开启才可以；rpc.statd 监听来自其他主机重启的通知，并且管理当本地系统重启时主机列表
rpc.idmapd	名字映射后台进程

查看以上进程，均可以执行“man进程名”命令，例如：“man rpc.idmapd”，可查看有关NFS服务进程的英文说明参考：

NFS serving is taken care of by five daemons：rpc.nfsd，which does most of the work；rpc.lockd and rpc.statd，which handle file locking；rpc.mountd，which handles the initial mount requests，and rpc.rquotad，which handles user file quotas on exported volumes.Starting with 2.2.18，lockd is called by nfsd upon demand，so you do not need to worry about starting it yourself.statd will need to be started separately.Most recent Linux distributions will have startup scripts for these daemons.


提示：对于使用yum/rpm包安装的软件，命令service nfs start的

启动方式等同于/etc/init.d/nfs start。在此，老男孩推荐使用/etc/init.d/nfs start，原因是可以使用tab命令补全完整的启动路径，而命令service nfs start则需要完全手敲出来。另外，这里提到的启动方式不仅仅适合nfs命令，同样适合大多数使用yum/rpm包安装的其他软件。

10.4.5　配置NFS服务器端服务开机自启动

配置NFS及rpcbind服务在系统开机或重新启动后自动运行的命令如下：

```
[root@nfs-server ~]# chkconfig rpcbind on
[root@nfs-server ~]# chkconfig nfs on
```

大家也可以使用更细化的命令，如chkconfig--level 3 nfs on，表示只在3级别（即字符界面）上开机启动NFS服务。

下面来查看chkconfig设置开机启动后的结果。不同的机器可能会有中英文显示差别，可以统一设置好英文字符集后查看：

```
[root@nfs-server ~]# LANG=en
[root@nfs-server ~]# chkconfig --list rpcbind
rpcbind         0:


off   1:


off   2:


on    3:


on    4:


on    5:
```

```
on    6:


off
[root@nfs-server ~]# chkconfig --list nfs
nfs             0:


off   1:


off   2:


on    3:


on    4:


on    5:


on    6:


off
```

下面的方法和上面相当，但是效率不如上面的命令。因为要先输出所有的，再过滤需要的，而上面的直接查看对应服务名更精确。

```
[root@nfs-server ~]# chkconfig --list|egrep "nfs\b|rpcbind"
nfs             0:


off   1:


off   2:


on    3:
```

```
on    4:

on    5:

on    6:

off
rpcbind         0:

off   1:

off   2:

on    3:

on    4:

on    5:

on    6:

off
```

 提示：如果“3：”后的部分为“on”或“启用”状态就表示OK。0-6为系统不同的启动级别，3为文本模式即当前运行的模式。

要说明的是，在老男孩曾工作的多家大型企业里，大都是统一按照运维规范将服务的启动命令放到/etc/rc.local文件里的，而不是用

chkconfig管理的。把/etc/rc.local文件作为本机的重要服务档案文件，所有服务的开机自启动都必须放入/etc/rc.local。这样规范的好处是，一旦管理此服务器的人员离职，或者业务迁移都可以通过/etc/rc.local很容易地查看到服务器对应的相关服务，可以方便运维管理。下面是把启动命令放入到/etc/rc.local文件中的配置信息，注意别忘了加上启动服务的注释。

```
[root@nfs-server ~]# tail -3 /etc/rc.local
#start up nfs serivce by oldboy at 20150613
/etc/init.d/rpcbind start
/etc/init.d/nfs start
```

10.5　实战配置NFS服务器端

10.5.1　NFS服务器端配置文件路径

NFS服务的默认配置文件路径为：/etc/exports，并且默认是空的。

```
[root@nfs-server ~]# ls -l /etc/exports
-rw-r--r--. 1 root root 0 Jan 12  2010 /etc/exports
[root@nfs-server ~]# cat /etc/exports
```

提示：NFS默认配置文件/etc/exports其实是存在的，但是没有内容，需要用户自行配置。

10.5.2　exports配置文件格式

/etc/exports文件配置格式为：

NFS共享的目录

　NFS客户端地址

1（参

1，参

2，…

.）

　客户端地址

2（参

1，参

2，…

.）

或

```
NFS共享的目录

  NFS客户端地址（参

1，参

2，...

.）
```

查看exports语法文件格式帮助的方法为：

执行man exports命令，然后切换到文件结尾，可以快速查看如下样例格式：

```
EXAMPLE
    # sample /etc/exports file
    /               master（


rw）


 trusty（


rw，


no_root_squash）


    /projects       proj*.local.domain（
```

```
rw)

    /usr            *.local.domain (

ro)

 @trusted (

rw)

    /home/joe       pc001 (

rw,

all_squash,

anonuid=150,

anongid=100)

    /pub             (

ro,

insecure,

all_squash)
```

上述各个列的参数含义如下：

·NFS共享的目录： 为NFS服务器端要共享的实际目录，要用绝对路径，如（/data）。注意共享目录的本地权限，如果需要读写共享，一定要让本地目录可以被NFS客户端的用户（nfsnobody）读写。

·NFS客户端地址： 为NFS服务器端授权的可访问共享目录的NFS客户端地址，可以为单独的IP地址或主机名、域名等，也可以为整个网段地址，还可以用“*”来匹配所有客户端服务器，这里所谓的客户端一般来说是前端的业务服务器，例如：Web服务。具体说明见表10-3。

·权限参数集： 对授权的NFS客户端的访问权限设置。参数具体说明见后文。

表10-3　指定NFS客户端地址的配置详细说明

客户端地址	具体地址	说　明
授权单一客户端访问 NFS	10.0.0.30	一般情况，生产环境中此配置不多
授权整个网段可访问 NFS	10.0.0.0/24	其中的 24 等同于 255.255.255.0，指定网段为生产环境中最常见的配置。配置简单、维护方便
授权整个网段可访问 NFS	10.0.0.*	指定网段的另外写法（不推荐使用）
授权某个域名客户端访问	nfs.oldboyedu.com	此方法生产环境中一般情况不常用
授权整个域名客户端访问	*.oldboyedu.com	此方法生产环境中一般情况不常用

10.5.3　企业生产场景NFS exports配置实例

下面介绍企业生产环境中常见的NFS配置实例，具体配置见表10-4。

表10-4　/etc/exports文件格式配置实例说明

常用格式说明	要共享的目录　客户端 IP 地址或 IP 段（参 1，参 2，….）
配置例一	/data 10.0.0.0/24（rw，sync）#<== 允许客户端读写，并且数据同步写到服务器端的磁盘里，注意，24 和“(”之间不能有空格
配置例二	/data/blog 10.0.0.0/24（rw, sync, all_squash, anonuid=2000, anongid=2000） #<== 允许客户端读写，并且数据同步写到服务器端的磁盘里，并且指定客户端的用户 UID 和 GID。早期生产环境的一种配置，适合多客户端共享一个 NFS 服务单目录，如果所有服务器的 nfsnobody 账户 UID 都是 65534，则本例没什么必要了。早期 CentOS 5.5 的系统默认情况下 nfsnobody 的 UID 不一定是 65534，此时如果这些服务器共享一个 NFS 目录，就会出现访问权限问题
配置例三	/home/oldboy 10.0.0.0/24（ro）#<== 只读共享 #<== 用途：例如在生产环境中，开发人员有查看生产服务器日志的需求，但又不希望给开发生产服务器的权限，那么就可以给开发提供从某个测试服务器 NFS 客户端上查看某个生产服务器的日志目录（NFS 共享）的权限，当然这不是唯一的方法，例如可以把程序记录的日志发送到测试服务器供开发查看或者通过收集日志等其他方式展现

这里以“配置例一”为例进行说明，如下：

·/data为要共享的NFS服务器端的目录，注意，被共享的目录一定要用绝对路径。

·10.0.0.0/24表示允许NFS客户端访问共享目录的网段范围。24表示255.255.255.0。

·（rw，sync）中的rw表示允许读写，sync表示数据同步写入到NFS

服务器端的硬盘中。

·也可用通配符*替换IP地址，表示允许所有主机，也可以写成10.0.0.*的形式。

10.6　NFS配置参数权限

本节将介绍NFS服务器端的权限设置，即/etc/exports文件配置格式中小括号（）里的参数集，具体见表10-5。

表10-5　NFS配置权限设置常用参数说明

参数名称	参数用途
rw※	Read-write，表示可读写权限
ro	Read-only，表示只读权限
sync※	请求或写入数据时，数据同步写入到 NFS Server 的硬盘后才返回。 优点，数据安全不会丢，缺点，性能比不启用该参数要差
async※	写入时数据会先写到内存缓冲区，直到硬盘有空档才会再写入磁盘，这样可以提升写入效率！风险为若服务器宕机或不正常关机，会损失缓冲区中未写入磁盘的数据（解决办法：服务器主板电池或加 UPS 不间断电源）
no_root_squash	访问 NFS Server 共享目录的用户如果是 root 的话，它对该共享目录具有 root 权限。这个配置原本是为无盘客户端准备的。用户应避免使用
root_squash	如果访问 NFS Server 共享目录的用户是 root，则它的权限将被压缩成匿名用户，同时它的 UID 和 GID 通常会变成 nfsnobody 账号身份
all_squash※	不管访问 NFS Server 共享目录的用户身份如何，它的权限都将被压缩成匿名用户，同时它的 UID 和 GID 都会变成 nfsnobody 账号身份。在早期多个 NFS 客户端同时读写 NFS Server 数据时，这个参数很有用 在生产中配置 NFS 的重要技巧： 1）确保所有客户端服务器对 NFS 共享目录具备相同的用户访问权限 a.all_squash 把所有客户端都压缩成固定的匿名用户（UID 相同）。 b. 就是 anonuid，anongid 指定的 UID 和 GID 的用户。 2）所有的客户端和服务器端都需要有一个相同的 UID 和 GID 的用户，即 nfsnobody（UID 必须相同）
anonuid※	参数以 anon* 开头即指 anonymous 匿名用户，这个用户的 UID 设置值通常为 nfsnobody 的 UID 值，当然也可以自行设置这个 UID 值。但是，UID 必须存在于 /etc/passwd 中。在多 NFS Clients 时，如多台 Web Server 共享一个 NFS 目录，通过这个参数可以使得不同的 NFS Clients 写入的数据对所有 NFS Clients 保持同样的用户权限，即为配置的匿名 UID 对应用户权限，这个参数很有用，一般默认即可
anongid※	同 anonuid，区别就是把 uid（用户 id）换成 gid（组 id）
提示：带 ※ 号的表示常用参数，更多参数请查看 man exports。	

图10-9为NFS配置权限设置常用参数关系的图解。

大家可以通过执行man exports查阅更多exports参数说明，这种man查询的方法是通向高手层次的必经之路。希望读者能够尽早掌握man工具。

配置好NFS服务后，通过cat/var/lib/nfs/etab命令可以看到NFS配置的参数以及默认自带的参数。结果如下：

```
/oldboy 10.0.0.0/24(
ro,
sync,
wdelay,
hide,
nocrossmnt,
secure,
root_squash,
no_all_squash,
no_subtree_check,
```

```
 secure_locks,

acl,

mapping=identity,

anonuid=65534,

anongid=65534)
```

怎么样？很多吧？别怕，其实，一般情况下，大多数参数我们不需要理会。尤其是对于初学者。

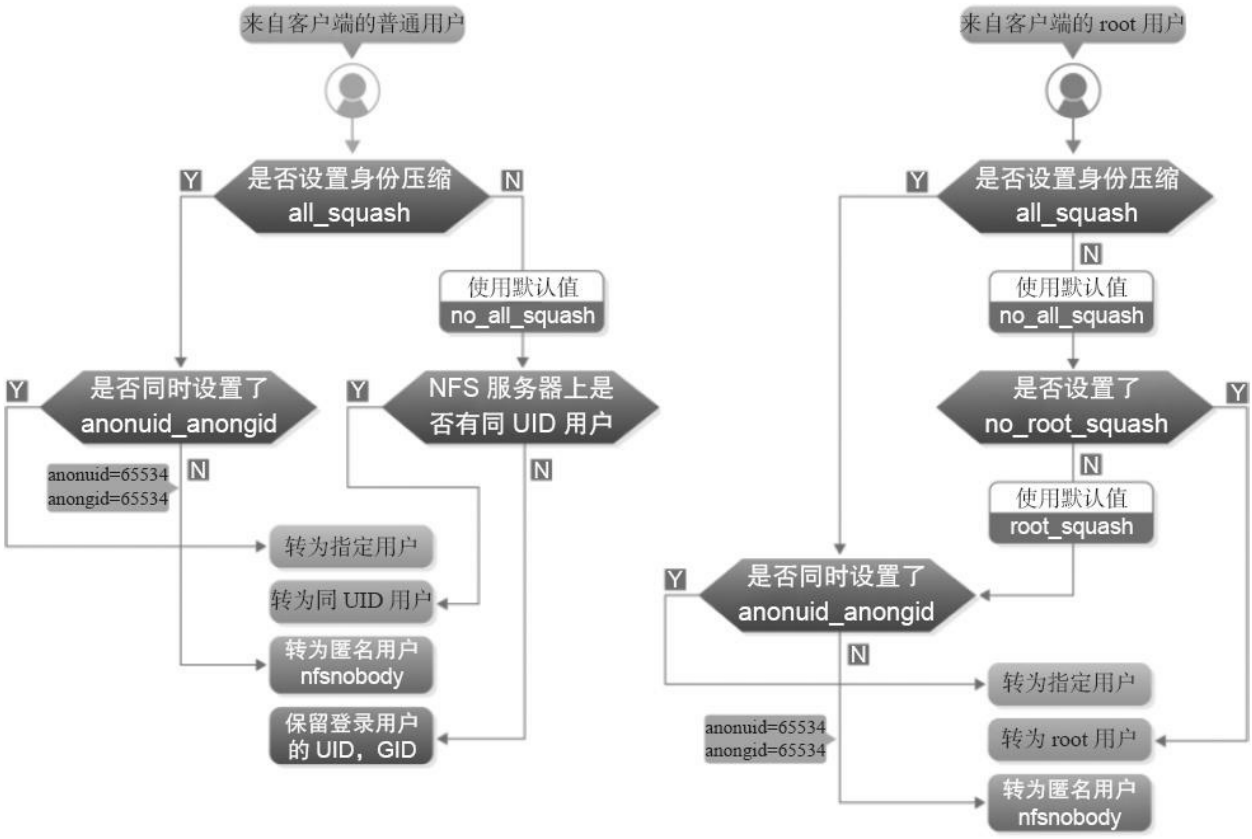


图10-9　NFS配置权限设置常用参数关系图

10.7　NFS服务企业案例配置实践

该例将实现把NFS server上的/data目录共享给10.0.0.0/24整个网段的主机，且可读写。

1.在NFS Server端执行的操作

这里以CentOS 6.6 x86_64系统为例讲解，对于CentOS 5.x x86_64系统，除了将RPC服务由rpcbind换为portmap外，其他差别不大。

在NFS Server端执行如下操作（NFS Server端IP为10.0.0.7）。

1）查看CentOS 6.6 x86_64系统环境：

```
[root@nfs-server ~]# cat /etc/redhat-release
CentOS release 6.6 (


Final)



[root@nfs-server ~]# uname -r
2.6.32-504.el6.x86_64
[root@nfs-server ~]# uname -m
x86_64
```

2）查看并启动rpcbind及NFS服务，然后加入开机自启动：

```
[root@nfs-server ~]# /etc/init.d/rpcbind status
rpcbind (
```

```
pid  996)


 is running...
[root@nfs-server ~]# /etc/init.d/nfs status
rpc.svcgssd 已停



rpc.mountd （


pid 1557)


 is running...
nfsd （


pid 1573 1572 1571 1570 1569 1568 1567 1566)


 is running...
rpc.rquotad （


pid 1552)


 is running...
[root@nfs-server ~]# chkconfig --list rpcbind
rpcbind         0:


off   1:


off   2:


on    3:


on    4:


on    5:
```

```
on    6:


off
[root@nfs-server ~]# chkconfig --list nfs
nfs             0:

off   1:

off   2:

on    3:

on    4:

on    5:

on    6:


off
[root@nfs-server ~]# tail -3 /etc/rc.local
#start up nfs serivce by oldboy at 20150613
/etc/init.d/rpcbind start
/etc/init.d/nfs start
```

chkconfig和/etc/rc.local的配置二选一即可。

3）创建需要共享的目录并授权（工作中这里的/data应该是已存在的目录）：

```
[root@nfs-server ~]# mkdir -p /data     #     <==创建要共享的
```

NFS目录，也可使用已有的目录

[root@nfs-server ~]# touch /data/oldboy.txt   #   <==创建一个测试文件

[root@nfs-server ~]# chown -R nfsnobody.nfsnobody /data
[root@nfs-server ~]# ls -ld /data   #   <==养成操作后检查的习惯

drwxr-xr-x 2 nfsnobody nfsnobody 4096 Jun 13 17:

34 /data
#<==在

NFS服务器端把要共享的

NFS目录赋予

NFS默认用户

nfsnobody用户和用户组权限，如不设置这个权限，会导致

NFS客户端访问时无法通过

NFS本地共享目录权限写入数据。当然也可以给

NFS服务器端本地共享目录

777的权限，但是工作中最好不用，因为

777的权限太大了

```
[root@nfs-server ~]# grep nfsnobody /etc/passwd
nfsnobody:


x:


65534:


65534:


Anonymous NFS User:


/var/lib/nfs:


/sbin/nologin
```

4）配置NFS服务配置文件，并在本地查看挂载信息情况：

```
[root@nfs-server ~]# tail -2 /etc/exports
#shared /data by oldboy for students
/data 10.0.0.0/24（


rw.sync）



#<==相当于使用


vi编辑


NFS配置文件


/etc/exports然后输入
```

/data 10.0.0.0/24（

rw.sync）一样，这里故意把括号里的逗号写成了点号，看看会报错不

[root@nfs-server ~]# exportfs -rv
exportfs：

 /etc/exports：

1：

 unknown keyword "rw.sync"   #   <==加载配置，果然提示错误

[root@nfs-server ~]# sed -i 's#rw.sync#rw，

sync#g' /etc/exports #<==把点号改回逗号

[root@nfs-server ~]# cat /etc/exports   #   <==查看通过生成的最终配置文件结果

#shared /data by oldboy for students
/data 10.0.0.0/24（

rw，

sync）

[root@nfs-server ~]# exportfs -rv   #   <==修改配置文件

/etc/exports后，需要重新加载

NFS配置

exporting 10.0.0.0/24:

/data
[root@nfs-server ~]# showmount -e localhost   #   <==别忘了在

NFS服务器本地查看挂载情况

Export list for localhost:

/data 10.0.0.0/24
[root@nfs-server ~]# cat /var/lib/nfs/etab #<==通过

cat /var/lib/nfs/etab 查看

NFS server配置文件的参数（包括默认加载的参数）

/data
10.0.0.0/24（

rw,

sync,

wdelay,

hide,

```
nocrossmnt,

secure,

root_squash,

no_all_squash,

no_subtree_check,

secure_locks,

acl,

anonuid=65534,

anongid=65534)
```

大多参数大家并不需要深入了解

可以在本地把服务器端同时又作为客户端进行挂载测试，执行的命令和结果如下：

```
[root@nfs-server ~]# mount -t nfs 10.0.0.7:


/data /mnt
[root@nfs-server ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3       7.1G  1.5G  5.3G  22% /
tmpfs           497M     0  497M   0% /dev/shm
```

```
/dev/sda1       190M   27M  153M  15% /boot
10.0.0.7:


/data  7.1G  1.5G  5.3G  22% /mnt
```

根据提示可知挂载是成功的，到此NFS服务器端的配置完毕。

```
[root@nfs-server ~]#cat /var/lib/nfs/rmtab
#<==等客户端挂载了后，通过执行


cat /var/lib/nfs/rmtab来查看哪些客户端主机挂载了


NFS server
10.0.0.7:


/data:


0x00000001 #<==经过测试，目前适合


5.x，不适合


6.x
```

说明：

·上文中的配置/oldboy 10.0.0.0/24（rw，sync），其中“24”和“（”之间不能有空格。

·修改/etc/exports后，需执行/etc/init.d/nfs reload或exportfs-rv重新加载NFS配置，但不需要restart NFS。在生产环境中，此操作要注意。

·使用yum/rpm包安装的软件，service nfs reload启动方式等同于/etc/init.d/nfs reload。

2.在NFS Client端执行的操作

在所有的NFS Client上执行的操作都是相同的，这里以CentOS 6.6 NFS Client1为例说明（NFS Client1端IP为10.0.0.8）。具体操作如下。

1）检查CentOS 6.6 x86_64系统环境：

```
[root@nfs-client1 ~]# cat /etc/redhat-release
CentOS release 6.6 (


Final)



[root@nfs-client1 ~]# uname -r
2.6.32-504.el6.x86_64
[root@nfs-client1 ~]# uname -m
x86_64
```

2）安装客户端软件rpcbind：

```
[root@nfs-client1 ~]# yum install rpcbind -y
[root@nfs-client1 ~]# rpm -qa rpcbind
rpcbind-0.2.0-11.el6.x86_64
```

为了使用showmount等功能，所有客户端最好也要安装NFS软件，但是不启动NFS服务。

```
[root@nfs-client1 ~]# yum install nfs-utils -y
```

```
[root@nfs-client1 ~]# rpm -qa nfs-utils
nfs-utils-1.2.3-54.el6.x86_64
```

3）启动RPC服务（注意，无须启动NFS服务）。

```
[root@nfs-client1 ~]# LANG=en
[root@nfs-client1 ~]# /etc/init.d/rpcbind start
Starting rpcbind:


                                           [  OK  ]
[root@nfs-client1 ~]# /etc/init.d/rpcbind status
rpcbind (


pid  5337)


 is running...
[root@nfs-client1 ~]# showmount -e 10.0.0.7
Export list for 10.0.0.7:



/data 10.0.0.0/24
#<==如果此时出现如下错误，多数原因是因为防火墙没关导致的



[root@nfs-client1 ~]# showmount -e 10.0.0.7     #<==恭喜各位，这里遭遇了故障



clnt_create:


 RPC:


 Port mapper failure - Unable to receive:


 errno 113 (


No route to host)
```

执行如下命令关闭防火墙。

```
[root@nfs-client1 ~]# /etc/init.d/iptables stop
iptables:


 Setting chains to policy ACCEPT:


 filter          [  OK  ]
iptables:


 Flushing firewall rules:


                          [  OK  ]
iptables:


 Unloading modules:


                                [  OK  ]
[root@nfs-client1 ~]# /etc/init.d/iptables status
iptables:


 Firewall is not running.
```

4）挂载NFS共享目录/data，命令如下：

```
[root@nfs-client1 ~]# showmount -e 10.0.0.7     #      <==挂载前首先检查有权限需要挂载的信


Export list for 10.0.0.7:
```

```
/data 10.0.0.0/24     #     <==可以清晰地看到共享了
```

/data目录

```
[root@nfs-client1 ~]# mount -t nfs 10.0.0.7:
```

```
/data /mnt     #<==执行挂载命令挂载
```

```
[root@nfs-client1 ~]# df -h     #<==查看挂载后的结果
```

```
.
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3       7.1G  1.4G  5.4G  21% /
tmpfs           497M     0  497M   0% /dev/shm
/dev/sda1       190M   27M  153M  15% /boot
10.0.0.7:
```

```
/data  7.1G  1.5G  5.3G  22% /mnt     #<==这就是客户端挂载后的结果
```

```
[root@nfs-client1 ~]# mount
/dev/sda3 on / type ext4 (
```

```
rw)
```

```
proc on /proc type proc (
```

```
rw)
```

```
sysfs on /sys type sysfs (
```

```
rw)

devpts on /dev/pts type devpts (
rw,
gid=5,
mode=620)

tmpfs on /dev/shm type tmpfs (
rw)

/dev/sda1 on /boot type ext4 (
rw)

none on /proc/sys/fs/binfmt_misc type binfmt_misc (
rw)

sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (
rw)

10.0.0.7:
```

```
/data on /mnt type nfs （

rw,

vers=4,

addr=10.0.0.7,

clientaddr=10.0.0.8）
```

5）测试读写数据，命令如下：

```
[root@nfs-client1 ~]# ls /mnt/     #<==这是服务器端事先建立的文件，表示可读

oldboy.txt
[root@nfs-client1 ~]# mkdir /mnt/test     #<==表示可写

[root@nfs-client1 ~]# ls -l /mnt/总用量

 4
-rw-r--r-- 1 root      root         0 6月

  13 19:

10 oldboy.txt
drwxr-xr-x 2 nfsnobody nfsnobody 4096 6月

  13 19:
```

```
11 test #<==用户和用户组都是

nfsnobody啊。没错，默认所有的客户端写入文件和目录都会被压缩成默认的

UID为

65534的

nfsnobody用户
```

6）将rpcbind服务和挂载加入开机自启动，如下：

```
[root@nfs-client1 ~]# echo "/etc/init.d/rpcbind start" >>/etc/rc.local
[root@nfs-client1 ~]# echo "/bin/mount -t nfs 10.0.0.7:

/data /mnt" >>/etc/rc.local
[root@nfs-client1 ~]# tail -2 /etc/rc.local
/etc/init.d/rpcbind start
/bin/mount -t nfs 10.0.0.7:

/data /mnt
```

到此，NFS客户端挂载成功。

10.8　NFS服务的重点知识梳理

当多个NFS客户端访问服务器端的读写文件时，需要具有以下几个权限：

·NFS服务器/etc/exports设置需要开放可写入的权限，即服务器端的共享权限。

·NFS服务器实际要共享的NFS目录权限具有可写入w的权限，即服务器端本地目录的安全权限。

·每台机器都对应存在和NFS默认配置UID的相同UID 65534的nfsnobody用户（确保所有客户端的访问权限统一，否则每个机器需要同时建立相同UID的用户，并覆盖NFS的默认用户配置）。

只有满足上述三个条件，多个NFS客户端才能具有查看、修改、删除其他任意NFS客户端上传文件的权限，这在大规模的集群环境中作为集群共享存储时尤为重要。

表10-6列出了常用的重点NFS服务文件或命令。

表10-6　重点NFS服务文件或命令的说明

NFS 常用路径	说　明
/etc/exports	NFS 服务主**配置文件**，配置 NFS 具体共享服务的地点，默认内容为空。以行为单位。 [root@nfs-server ~]# cat /etc/exports #shared data for bbs by oldboy at 20140901 /data 10.0.0.0/24(rw,sync)

（续）

NFS 常用路径	说　明
/usr/sbin/exportfs	NFS 服务的**管理命令**。例如：可以加载 NFS 配置生效，还可以直接配置 NFS 共享目录，即无需配置 /etc/exports 实现共享。 [root@nfs-server ~]# exportfs -rv **← 加载配置生效，等价优雅重启 /etc/init.d/nfs reload** exporting 10.0.0.0/24:/data 这里有一个服务平滑重启的概念，即超市、银行到关门时间了，还会继续提供服务给已经在门里的人，但是新来的人就会被挡在门外了。网站服务平滑重启，是提升用户体验必须要考虑的。 maintain table of exported NFS file systems exportfs 不但可以加载配置生效，也可以通过命令直接共享目录。越过 /etc/exports，但是重启失效
/usr/sbin/showmount	常用来在客户端查看 NFS 配置及挂载结果的命令。 show mount information for an NFS server 配置 nfsserver，分别在服务器端及客户端查看挂载情况
/var/lib/nfs/etab	NFS 配置文件的完整参数设定的文件（有很多没有配置但是默认就有的 NFS 参数）。 /var/lib/nfs/etab　　master table of exports
/var/lib/nfs/xtab	适合 C5.x 记录曾经挂载过的 NFS 客户端的信息，包括 IP 地址等，**CentOS 6.6 没有此文件了**
/proc/mounts	客户端挂载参数 [root@B mnt]# grep mnt /proc/mounts 10.0.0.7:/data/ /mnt nfs4 rw,relatime,vers=4,rsize=131072,wsize=131072,namlen=255,hard,proto=tcp,port=0,timeo=600,retrans=2,sec=sys,clientaddr=10.0.0.8,minorversion=0,local_lock=none,addr=10.0.0.7 0 0
/var/lib/nfs/rmtab	客户端访问服务器 exports 的信息列表

10.9　NFS客户端挂载命令

10.9.1　NFS客户端挂载命令格式

这里先强调下客户端挂载的命令格式，具体见表10-7。

表10-7　客户端挂载的命令格式

挂载命令	挂载的格式类型	NFS 服务器端提供的共享目录	NFS 客户端的挂载点
mount	-t nfs	10.0.0.7:/data	/mnt（必须存在）
完整挂载命令为：mount -t nfs 10.0.0.7:/data /mnt，此命令要在客户端执行			

下面介绍执行挂载的过程。

```
[root@nfs-client1 ~]# showmount -e 10.0.0.7      #      <==挂载前首先检查有权限需要挂载的信

Export list for 10.0.0.7:

/data 10.0.0.0/24      #<==可以清晰地看到共享了

/data目录

[root@nfs-client1 ~]# mount -t nfs 10.0.0.7:

/data /mnt      #<==执行挂载命令挂载
```

```
[root@nfs-client1 ~]# df -h     #<==查看挂载后的结果。


Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3       7.1G  1.4G  5.4G  21% /
tmpfs           497M     0  497M   0% /dev/shm
/dev/sda1       190M   27M  153M  15% /boot
10.0.0.7:

/data  7.1G  1.5G  5.3G  22% /mnt     #<==这就是客户端挂载后的结果


[root@nfs-client1 ~]# mount     #<==查看挂载后的结果


/dev/sda3 on / type ext4 (

rw)


proc on /proc type proc (

rw)


sysfs on /sys type sysfs (

rw)


devpts on /dev/pts type devpts (

rw,

gid=5,
```

```
mode=620)

tmpfs on /dev/shm type tmpfs (

rw)

/dev/sda1 on /boot type ext4 (

rw)

none on /proc/sys/fs/binfmt_misc type binfmt_misc (

rw)

sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (

rw)

10.0.0.7:

/data on /mnt type nfs (

rw,

vers=4,

addr=10.0.0.7,
```

```
clientaddr=10.0.0.8)

[root@nfs-client1 ~]# grep mnt /proc/mounts     #<==查看挂载后的结果

10.0.0.7:

/data/ /mnt nfs4 rw,

relatime,

vers=4,

rsize=131072,

wsize=131072,

namlen=255,

hard,

proto=tcp,

port=0,

timeo=600,

retrans=2,

sec=sys,
```

```
clientaddr=10.0.0.8,

minorversion=0,

local_lock=none,

addr=10.0.0.7 0 0
```

进行到这一步，NFS客户端就已经挂载成功了。可以进入/mnt写入一些实验数据，然后去看看NFS Server端的/data里是否有从NFS客户端/mnt目录传过来的数据。

下面进行读写数据测试。

```
[root@nfs-client1 ~]# ls /mnt/     #<==这是服务器端事先建立的文件，表示可读

oldboy.txt
[root@nfs-client1 ~]# mkdir /mnt/test     #<==表示可写

[root@nfs-client1 ~]# ls -l /mnt/总用量

 4
-rw-r--r-- 1 root      root         0 6月

  13 19:

10 oldboy.txt
drwxr-xr-x 2 nfsnobody nfsnobody 4096 6月
```

13 19:

11 test #<==用户和用户组都是

nfsnobody。没错，默认所有的客户端写入文件和目录都会被压缩成默认的

uid为

65534的

nfsnobody用户

10.9.2 NFS客户端挂载排错思路

当客户端挂载出现故障时，首先要确认NFS服务器端的配置和服务是否OK。确认命令如下：

```
[root@nfs-server ~]# showmount -e localhost
Export list for localhost:



/data 10.0.0.0/24
```

最好在服务器端自己挂自己，看看可以不可以？

```
[root@nfs-server ~]# mount -t nfs 10.0.0.7:



/data /mnt
[root@nfs-server ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3       7.1G  1.5G  5.3G  22% /
tmpfs           497M     0  497M   0% /dev/shm
/dev/sda1       190M   27M  153M  15% /boot
10.0.0.7:


/data  7.1G  1.5G  5.3G  22% /mnt
```

 提示：这步主要检查服务器端的NFS服务是否正常！

然后确认NFS客户端showmount是否OK。正常情况下：

```
[root@nfs-client1 ~]# showmount -e 10.0.0.7
Export list for 10.0.0.7:
```

```
/data 10.0.0.0/24
```

如果环境准备不当，可能会遇到如下两个报错。

第一个报错：

```
[root@nfs-client1 ~]# showmount -e 10.0.0.7
clnt_create:

 RPC:

 Port mapper failure - Unable to receive:

 errno 113 (

No route to host)
```

第二个报错：

```
[root@nfs-client1 ~]# showmount -e 10.0.0.7
clnt_create:

 RPC:

 Program not registered
```

如果检查发现配置或服务有问题，则一般要根据提示解决，通用的

总体思路如下。

1）从客户端ping NFS服务器端IP，目的是看物理链路是否通。

```
[root@nfs-client1 ~]# ping 10.0.0.7
PING 10.0.0.7 (
10.0.0.7)
 56(
84)
 bytes of data.
64 bytes from 10.0.0.7:
 icmp_seq=1 ttl=64 time=9.13 ms
64 bytes from 10.0.0.7:
 icmp_seq=2 ttl=64 time=0.555 ms
```

2）从客户端telnet服务器端IP端口检查，用于测试NFS服务（RPC服务）是否通。

```
[root@nfs-client1 ~]# telnet 10.0.0.7 111
Trying 10.0.0.7...
telnet:
 connect to address 10.0.0.7:
 No route to host #<==防火墙拦截了
```

如果不通！再通过去NFS服务器端执行如下命令看看111端口是不是开了！

```
[root@nfs-server ~]# telnet 10.0.0.7 111
Trying 10.0.0.7...
telnet:


 connect to address 10.0.0.7:


 Connection refused
```

前面报错中的“No route to host”多数是因为服务器端防火墙所致，可执行下面的命令关闭防火墙：

```
[root@nfs-server ~]# /etc/init.d/iptables stop
Flushing firewall rules:


 [  OK  ]
Setting chains to policy ACCEPT:


 filter [  OK  ]
Unloading iptables modules:


 [  OK  ]
```

然后再从客户端检查，如下：

```
[root@nfs-client1 ~]# telnet 10.0.0.7 111
Trying 10.0.0.7...
Connected to 10.0.0.7. #<== Connected表示通了。
```

```
Escape character is '^]'.
^]
telnet> quit
Connection closed.
[root@nfs-client1 ~]# showmount -e 10.0.0.7
Export list for 10.0.0.7:



/data 10.0.0.0/24
```

 提示：注意命令提示符中主机名的变化，nfs-client为NFS客户端操作，nfs-server为NFS服务器端操作。

上述第二个报错：“RPC：Program not registered”提示很清楚了，是服务器端RPC服务和NFS服务存在启动顺序问题。

10.9.3　NFS客户端开机自启动挂载

配置客户端mount挂载命令使挂载开机自动执行，这里有两种方法，如下：

第一种方法，将挂载命令放在/etc/rc.local里。

缺点：偶尔开机挂载不上，工作中除了开机自启动配置，还要对是否挂载做监控。

第二种方法，将挂载命令放在/etc/fstab里。

其实这种配置方法有如下误区。

fstab会优先于网络被Linux系统加载。网络没启动时执行fstab会导致连不上NFS服务器端，无法实现开机挂载。而且，即使是本地的文件系统，也要注意，fstab最后两列要设置0 0。否则有可能导致无法启动服务器的问题，如图10-10所示。

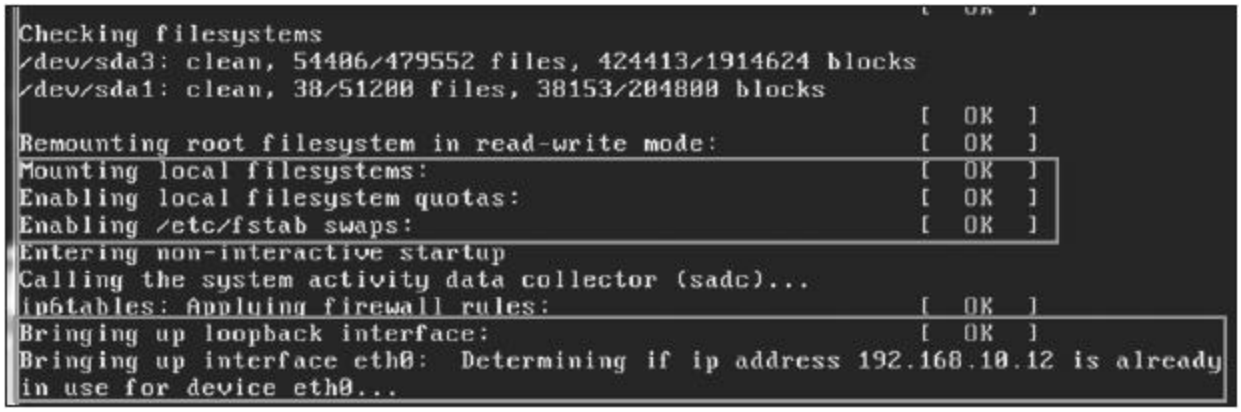


图10-10　开机启动fstab优先网卡启动图

因此，NFS网络文件系统最好不要放到fstab里实现开机挂载。

但是，如果在开机自启动服务里设置并启动了netfs服务，放入fstab里也是可以开机挂载的，如图10-11所示。

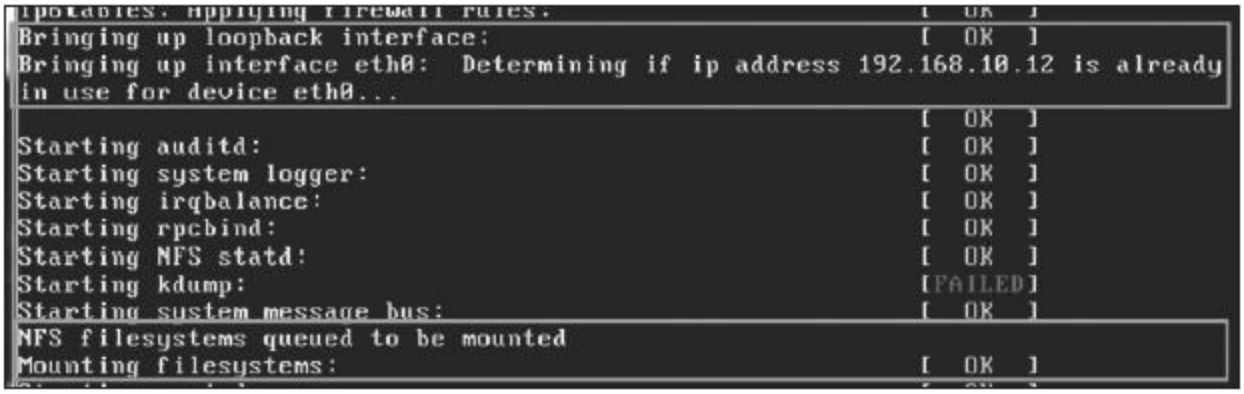


图10-11　开机启动网卡启动后NFS被挂载图

前面的操作是把挂载的命令放入了/etc/rc.local中，以实现开机自动挂载。命令如下：

```
[root@nfs-client1 ~]# echo "/bin/mount -t nfs 10.0.0.7:
```

```
/data /mnt" >>/etc/rc.local
[root@nfs-client1 ~]# echo "#mount by oldboy" >>/etc/rc.local
[root@nfs-client1 ~]# tail -2 /etc/rc.local
#mount by oldboy
/bin/mount -t nfs 10.0.0.7:



/data /mnt
```

10.10　生产环境高级案例配置实战

10.10.1　指定固定UID用户配置NFS共享的实例

指定固定UID用户配置NFS共享在CentOS 5.5 x86_64及以下生产环境中常用，在CentOS 6.x的场景不是必须要用的，这里仅仅作为一个例子讲解。

为什么要讲这个呢？因为，在实际生产环境下，一般为多台NFS客户端服务器同时挂一台NFS Server，这样A客户端写数据后，希望B、C等其他客户端也能同时读，能增删改A服务器写入的文件属性及内容。因此，这就需要NFS服务器端对所有客户端权限统一。而恰恰CentOS 5.5系统的NFS有个bug，就是没有UID为65534的nfsnobody用户，nfsnobody对应的UID为一长串数字，若将nfs-utils更新到1.0.9-60、nfs-utils-libs更新到1.0.8-7.9即可解决此bug。

此时需要在客户端及服务器端建立一个统一的NFS用户：名称、UID和GID均相同。也可使用默认的匿名用户nfsnobody。

下面以CentOS 6.6为例讲解修改共享用户UID的例子。

10.10.2　NFS服务器端的操作步骤

建立用户组zuma并指定GID 888，所有客户端也要用同样的命令建立。

```
[root@nfs-server ~]# groupadd zuma -g 888
```

建立用户zuma指定UID 888并加入zuma组，所有客户端也要用同样的命令建立。

```
[root@nfs-server ~]# useradd zuma -u 888 -g zuma
[root@nfs-server ~]# id zuma #<==查看建立的用户及组


uid=888（

zuma）

 gid=888（

zuma）

 groups=888（

zuma）
```

开始部署NFS Server，命令如下：

```
[root@nfs-server ~]# mkdir /oldboy
[root@nfs-server ~]# chown -R zuma.zuma /oldboy
[root@nfs-server ~]# ls -ld /oldboy
drwxr-xr-x 2 zuma zuma 4096 Jun 13 20:



23 /oldboy
[root@nfs-server ~]# echo '#new example'>>/etc/exports
[root@nfs-server ~]# echo '/oldboy 10.0.0.0/24 (



rw,



sync,



all_squash,



anonuid=888,



anongid=888)



'>>/etc/exports
[root@nfs-server ~]# tail -2 /etc/exports
#new example
/oldboy 10.0.0.0/24 (



rw,



sync,



all_squash,



anonuid=888,
```

```
anongid=888)


#<==nfs配置文件内容为


/oldboy 10.0.0.0/24(


rw,


sync,


all_squash,


anonuid=888,


anongid=888)


'
[root@nfs-server ~]# /etc/init.d/nfs reload
[root@nfs-server ~]# showmount -e localhost
Export list for localhost:



/oldboy 10.0.0.0/24
/data   10.0.0.0/24
```

NFS服务器端部署完成。

10.10.3　NFS客户端的操作步骤

建立用户组zuma并指定GID 888，同服务器端。

```
[root@nfs-client1 ~]# groupadd zuma -g 888
```

建立用户zuma指定UID 888并加入zuma组，同服务器端。

```
[root@nfs-client1 ~]# useradd zuma -u 888 -g zuma
[root@nfs-client1 ~]# id zuma                          #<==查看建立的用户及组



uid=888（

zuma）

 gid=888（

zuma）

 groups=888（

zuma）



[root@nfs-client1 ~]# mkdir /video                     #<==创建挂载点

/video
[root@nfs-client1 ~]# /etc/init.d/rpcbind status
rpcbind （
```

```
pid  2658)


 正在运行


...
[root@nfs-client1 ~]# showmount -e 10.0.0.7
Export list for 10.0.0.7:



/oldboy 10.0.0.0/24
/data   10.0.0.0/24
[root@nfs-client1 /]# mount -t nfs 10.0.0.7:


/oldboy /video  #<==挂在到客户端


video目录



[root@nfs-client1 /]# df -h
Filesystem        Size  Used Avail Use% Mounted on
/dev/sda3         7.1G  1.4G  5.4G  21% /
tmpfs             497M     0  497M   0% /dev/shm
/dev/sda1         190M   27M  153M  15% /boot
10.0.0.7:


/data    7.1G  1.5G  5.3G  22% /mnt
10.0.0.7:


/oldboy  7.1G  1.5G  5.3G  22% /video     #<==这个就是新挂载的



[root@nfs-client1 /]# cd /video
[root@nfs-client1 video]# touch newfile.txt     #<==测试创建文件



[root@nfs-client1 video]# ls -l总用量
```

```
 0
-rw-r--r-- 1 zuma zuma 0 Jun 13 20:


37 newfile.txt      #<==用户和组都是


zuma
[root@nfs-server ~]# ls -ld /oldboy      #<==目录权限仅有


zuma可以写入



drwxr-xr-x 2 zuma zuma 4096 Jun 13 20:


37 /oldboy
[root@nfs-server ~]# ls -l /oldboy      #<==文件用户和组都改成了


zuma
total 0
-rw-r--r-- 1 zuma zuma 0 Jun 13 20:


37 newfile.txt
```

指定NFS共享的用户zuma共享/oldboy目录的配置大功告成。

10.11　NFS客户端挂载深入

10.11.1　NFS客户端挂载参数说明

在NFS服务器端可以通过cat/var/lib/nfs/etab查看NFS服务器端配置参数的细节。在NFS客户端可以通过cat/proc/mounts查看mount的挂载参数细节。

1.mount挂载及fstab文件说明

通过如下命令在NFS客户端测试挂载获取的默认挂载参数：

```
[root@nfs-client1 video]# grep mnt /proc/mounts
10.0.0.7:

/data/ /mnt nfs4 rw,

relatime,

vers=4,

rsize=131072,

wsize=131072,

namlen=255,
```

```
hard,

proto=tcp,

port=0,

timeo=600,

retrans=2,

sec=sys,

clientaddr=10.0.0.8,

minorversion=0,

local_lock=none,

addr=10.0.0.7 0 0
```

表10-8提供了NFS客户端挂载参数说明。

表10-8　NFS客户端挂载参数列表

参　数	参数功能	默认参数
fg bg	当在客户端执行挂载时，可选择是前台（fg）还是在后台（bg）执行。若在前台执行，则 mount 会持续尝试挂载，直到成功或挂载时间超时为止，若在后台执行，则 mount 会在后台持续多次进行 mount，而不会影响到前台的其他程序操作。如果网络联机不稳定，或是服务器常常需要开关机，建议使用 bg 比较妥当	fg
soft hard	当 NFS Client 以 soft 挂载 Server 时，若网络或 Server 出现问题，造成 Client 和 Server 无法传输资料，Client 就会一直尝试，直到 timeout 后显示错误才停止。若使用 soft mount 的话，可能会在 timeout 出现时造成资料丢失，故一般不建议使用。若用 hard 模式挂载硬盘时，刚与 soft 相反，此时 Client 会一直尝试连线到 server，若 server 有回应就继续刚才的操作，若没有回应 NFS Client 会一直尝试，此时无法 umount 或 kill，所以常常会配合 intr 使用。这是默认值	hard

（续）

参　数	参数功能	默认参数
intr	当使用 hard 挂载的资源 timeout 后，若有指定 intr 参数，可以在 timeout 后把它中断掉，这避免出问题时系统整个被 NFS 锁死，建议使用 intr	无
rsize wsize	读出（rsize）与写入（wsize）的区块大小（block size），这个设置值可以影响客户端与服务器端传输数据的缓冲存储量，一般来说，如果在局域网内（LAN），并且客户端与服务器端都具有足够的内存，这个值可以设置大一点，比如说 65535（bytes），提升缓冲区块将提升 NFS 文件系统的传输能力。但设置的值也不要太大，最好以网络能够传输的最大值为限	CentOS 5：默认值 rsize=1024 wsize=1024 CentOS 6：默认值 rsize=131072,wsize=131072
proto=udp	使用 UDP 协定来传输资料，在 LAN 中会有比较好的性能。若要跨越 Internet 的话，使用 proto=tcp 多传输的数据会有比较好的纠错能力	proto=tcp

可通过man nfs查看上述参数信息。如果追求极致，可以用如下参数挂载：

```
mount -t nfs -o fg,

hard,

 intr,
```

```
rsize=13107,

wsize=131072 10.0.0.7:

/data/ /mnt
```

但是如果考虑以简单、易用为原则，直接选择默认值就可以了。

```
mount -t nfs 10.0.0.7:

/data/ /mnt
```

表10-8中的参数了解即可，无需细究。表10-9是mount-o参数对应的选项列表。

表10-9　mount-o参数对应的选项

参数	参数意义	系统默认值
suid nosuid	当挂载的文件系统上有任何 SUID 的程序时，只要使用 nosuid 就能够取消设置 SUID 的功能（SUID，在 Linux 基础学习篇权限管理里面讲过了）	suid
rw ro	可以指定文件系统是只读（ro）或可写（rw）	rw
dev nodev	是否可以保留装置文件的特殊功能？一般来说只有 /dev 才会有特殊的装置，因此可以选择 nodev	dev
exec noexec	是否具有执行文件的权限？如果想要挂载的仅是普通资源数据区（例如：图片、附件），那么可以选择 noexec	exec
user nouser	是否允许用户拥有文件的挂载与卸载功能？如果要保护文件系统，最好不要为用户提供挂载与卸载功能	nouser
auto noauto	这个 auto 指的是“ mount-a”时会不会被挂载的项目，如果不需要这个分区随时被挂载，可以设置为 noauto	auto

2.把man mount后的-o参数中英翻译对比

下面是mount命令的-o选项后面可以接的参数，注意，有些选项只有出现在/etc/fstab里才有效，下面这些选项可以应用在绝大多数文件系统上，但是sync仅适合ext2、ext3、fat、vfat和ufs等文件系统。

·async：涉及文件系统I/O的操作都是异步处理，即不会同步写到磁盘，此参数会提高性能，但会降低数据安全。一般情况，生产环境下不推荐使用。除非对性能要求很高，对数据可靠性不要求的场合。

·sync：该参数与async相反。有I/O操作时，都会同步处理I/O，即把数据同步写入硬盘。此参数会牺牲一点I/O性能，但是，换来的是掉电后数据的安全性。※

·atime：在每一次数据访问时，会同步更新访问文件的inode时间戳，是默认选项，在高并发的情况下，建议通过明确加上noatime，来取消这个默认项，以到达提升I/O性能，优化I/O的目的。

·ro：以只读的方式挂载一个文件系统。

·rw：以可写的方式挂载一个文件系统。※

·auto：能够被自动挂载通过-a选项。

·noauto：不会自动挂载文件系统。

·defaults：这是fstab里的默认值，包括rw、suid、dev、exec、auto、

nouser、async，默认情况大部分都是默认值。

·exec：允许文件系统执行二进制文件，取消这个参数，可以提升系统安全性。

·noexec：在挂载的文件系统中不允许直接执行任何二进制的程序，注意，仅对二进制程序有效，即使设置了noexec、shell，php程序还是可以执行的。*

·noatime：访问文件时不更新文件的inode时间戳，高并发环境下，推荐显式应用该选项，可以提高系统I/O性能。※

·nodiratime：不更新文件系统上的directory inode时间戳，高并发环境，推荐显式应用该选项，可以提高系统I/O性能。※

·nosuid：不允许set-user-identifier or set-group-identifier位生效。*

·suid：允许set-user-identifier or set-group-identifier位生效。

·nouser：禁止一个普通用户挂载该文件系统，这是默认挂载时的默认选项。

·remount：尝试重新挂载一个已经挂载了的文件系统，这通常被用来改变一个文件系统的挂载标志，从而使得一个只读文件系统变的可写，这个动作不会改变设备或者挂载点。当系统故障时进入single或

rescue模式修复系统时，会发现根文件系统经常会变成只读文件系统，不允许修改，此时该命令就派上用场了。具体命令为：mount-o remount，rw/，表示将根文件系统重新挂载使得可写。single或rescue模式修复系统时这个命令十分重要。※

·dirsync：目录更新时同步写入磁盘。*

其中，标有“※”的为性能优化重要选项，标有“*”的为安全优化重要选项。一般情况安全的参数和性能参数是对立的，即越安全性能就越差。

10.11.2　NFS客户端挂载优化

某网友问：在企业生产环境中，NFS客户端挂载有没有必须要加某些参数，比如，加noexec、nosuid、nodev、bg、soft、rsize、wsize等参数，有书上说建议加rsize、wsize这两个参数，是否也建议加呢？老师你在生产环境中是怎么做的？

解答：这个问题属于mount挂载优化内容（有些参数也适合其他文件系统），一般来说要适当加挂载参数，但是，最好是先做好测试，用数据来说话，才能更好地确定到底是挂载还是不挂载。

1.有关系统安全挂载参数选项

在企业工作场景，一般来说，NFS服务器共享的只是普通静态数据（图片、附件、视频），不需要执行suid、exec等权限，挂载的这个文件系统只能作为数据存取之用，无法执行程序，对于客户端来讲增加了安全性，例如：很多木马篡改站点文件都是由上传入口上传的程序到存储目录，然后执行的。

因此在挂载的时候，用下面的命令很有必要：

```
mount -t nfs -o nosuid,

noexec,
```

```
nodev,

rw 10.0.0.7:

/data /mnt
```

通过mount-o指定挂载参数与在/etc/fstab里指定挂载参数的效果是一样的。

网络文件系统和本地的文件系统效果也是一样的。

2.mount挂载性能优化参数选项

下面介绍几个在企业生产环境下，NFS性能优化挂载的例子。

1）禁止更新目录及文件时间戳挂载，命令如下：

```
mount -t nfs -o noatime,

nodiratime 10.0.0.7:

/data
```

2）安全加优化的挂载方式如下：

```
mount -t nfs -o nosuid,
```

```
noexec,

nodev,

noatime,

nodiratime,

intr,

rsize=131072,

wsize=131072 10.0.0.7:

/data /mnt
```

3）默认的挂载方式如下：

```
mount -t nfs 10.0.0.7:

/data /mnt
```

如果是本地文件系统，使用如下命令：

```
[root@nfs-server bbs]# mount /dev/sdb1 /mnt -o defaults,

async,

noatime,
```

```
data=writeback,


barrier=0
[root@nfs-server bbs]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda3             5.2G  3.5G  1.5G  71% /
/dev/sda1             190M   12M  169M   7% /boot
tmpfs                 249M     0  249M   0% /dev/shm
/dev/sdb1             892M   24M  817M   3% /mnt
[root@c58-nfs-server bbs]# grep mnt /proc/mounts
/dev/sdb1 /mnt ext3 rw,


noatime,


data=writeback 0 0
```

本地文件系统挂载时，如果加nodirtime会报错。

```
[root@c58-nfs-server bbs]# mount /dev/sdb1 /mnt -o defaults,


async,


noatime,


nodirtime,


data=writeback,


barrier=0
mount:


 wrong fs type,


 bad option,
```

```
bad superblock on /dev/sdb1,


    missing codepage or other error
    In some cases useful info is found in syslog - try
    dmesg | tail  or so
```

根据前面的测试我们知道，默认情况下NFS Server共享目录的参数默认配置为WSIZE和RSIZE，它们设定了NFS Server和NFS Client之间往来数据块的大小。

WSIZE和RSIZE的大小最好是1024的倍数，对于NFSV2来说，8192是RSIZE和WSIZE的最大数值，如果使用的是NFSV3，则可以尝试设置32768，如果是NFSV4可以到65536或更大。

如果在客户端挂载时使用了这两个参数，可以让客户端在读取和写入数据时，一次性读写更多的数据包，这可以提升访问性能和效率。

除此之外，还有noatime、nodiratime性能优化选项，这两个选项是说在读写磁盘的时候，不更新文件和目录的时间戳（即不更新文件系统中文件对应inode信息），这样就可以减少和磁盘系统的交互，提升读取和写入磁盘的效率，因为磁盘是机械的，每次读写都会消耗磁盘I/O，而更新文件时间戳对于工作数据必要性不大，最大问题是增加了访问磁盘I/O的次数，拖慢系统性能。

以下是NFS网络文件系统优化挂载的参数建议。

在CentOS 6.6 x86_64服务器端和客户端环境下，可使用如下命令参数：

```
mount -t nfs -o noatime,

nodiratime,

nosuid,

noexec,

nodev,

rsize=131072,

wsize=131072  10.0.0.7:

/data  /mnt
```

经过实际测试，CentOS 6.6 x86_64默认的挂载参数性能还是不错的。

```
mount -t nfs 10.0.0.7:

/data  /mnt
```


注意：非性能的参数越多，速度可能会越慢。根据具体的业务需要以及实际测试效果选择挂载参数。

3.NFS内核优化建议

下面是优化选项说明。

·/proc/sys/net/core/rmem_default：该文件指定了接收套接字缓冲区大小的默认值（以字节为单位），默认设置：124928。

·/proc/sys/net/core/rmem_max：该文件指定了接收套接字缓冲区大小的最大值（以字节为单位），默认设置：124928。

·/proc/sys/net/core/wmem_default：该文件指定了发送套接字缓冲区大小的默认值（以字节为单位），默认设置：124928。

·/proc/sys/net/core/wmem_max：该文件指定了发送套接字缓冲区大小的最大值（以字节为单位），默认设置：124928。

上述文件对应的具体内核优化命令如下：

```
cat >>/etc/sysctl.conf<<EOF
net.core.wmem_default = 8388608
net.core.rmem_default = 8388608
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
EOF
sysctl -p
```

4.企业生产场景NFS共享存储优化小结

·硬件：sas/ssd磁盘，买多块，raid0/raid10。网卡吞吐量要大，至少

千兆（多块bond）。

·NFS服务器端配置：/data 10.0.0.0/24（rw，sync，all_squash，anonuid=65534，anongid=65534）

·NFS客户端挂载优化配置命令：

```
mount -t nfs -o nosuid,

noexec,

nodev,

noatime,

nodiratime,

rsize=131072,

wsize=131072 10.0.0.7:

/data/ /mnt ←兼顾安全性能
```

·对NFS服务的所有服务器内核进行优化时，执行如下命令：

```
cat >>/etc/sysctl.conf<<EOF
net.core.wmem_default = 8388608
net.core.rmem_default = 8388608
```

```
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
EOF
```

执行sysctl-p生效。

·如果卸载的时候提示“umount：/mnt：device is busy”，需要退出挂载目录再进行卸载，如果是NFS server宕机了，则需要强制卸载，可执行umount-lf/mnt。

·大型网站NFS网络文件系统的替代软件为分布式文件系统Moosefs（mfs）、GlusterFS、FastDFS。

10.12　NFS系统应用的优缺点说明

NFS服务可以让不同的客户端挂载使用同一个共享目录，也就是将其作为共享存储使用，这样可以保证不同节点客户端数据的一致性，在集群架构环境中经常会用到。如果是Windows和Linux混合环境的集群系统，可以用samba来实现。

优点：

·简单，容易上手，容易掌握。

·NFS文件系统内数据是在文件系统之上的，即数据是能看得见的。

·部署快速，维护简单方便，且可控，满足需求就是最好的。

·可靠，从软件层面上看，数据可靠性高，经久耐用。数据是在文件系统之上的。

·服务非常稳定。

局限：

·存在单点故障，如果NFS Server宕机了，所有客户端都不能访问共享目录。这个在后期会通过负载均衡及高可用方案弥补。

·在大数据高并发的场合，NFS效率、性能有限（2千万/日以下PV的网站不是瓶颈，除非网站架构设计太差）。

·客户端认证是基于IP和主机名的，权限要根据ID识别，安全性一般（用于内网则问题不大）。

·NFS数据是明文的，NFS本身不对数据完整性进行验证。

·多台客户机器挂载一个NFS服务器时，连接管理维护麻烦（耦合度高）。尤其当NFS服务器端出问题后，所有NFS客户端都处于挂掉状态（测试环境可使用autofs自动挂载解决，正式环境可修复NFS服务或强制卸载）。

·涉及了同步（实时等待）和异步（解耦）的概念，NFS服务器端和客户端相对来说就是耦合度有些高。网站程序也是一样，尽量不要耦合度太高，系统及程序 架构师的重要职责就是为程序及架构解耦，让网站的扩展性变得更好。

应用建议：

对于大中小型网站（参考点2000万/日PV以下）线上应用，都有用武之地。门户网站也会有应用，生产场景应该多将数据的访问往前推，即尽量将静态存储里的资源通过CDN或缓存服务器提供服务，如果没有缓存服务或架构不好，存储服务器数量再多也是扛不住压力的，而且用

户体验会很差。

10.13　本章涉及的相关知识

10.13.1　showmount命令说明

showmount命令一般用于从NFS客户端检查NFS服务器端共享目录的情况，其常用的参数说明见表10-10。

表10-10　常用参数说明

短格式	长格式	用途及实例结果
-e	--exports	显示 NFS 服务器输出的目录列表： [root@nfs-client ~]# showmount -e 10.0.0.14 Export list for 10.0.0.14: /oldboy 10.0.0.0/24 [root@nfs-client ~]# showmount --exports 10.0.0.14 Export list for 10.0.0.14: /oldboy 10.0.0.0/24
-d	--directories	显示 NFS 服务器中提供共享的目录： [root@nfs-client ~]# showmount -d 10.0.0.14 Directories on 10.0.0.14: /oldboy
-a	--all	以 ip:/dir 格式显示 NFS 服务器的 IP 地址和可被挂载的目录： [root@nfs-client ~]# showmount -a 10.0.0.14 **All mount points on 10.0.0.14:** **10.0.0.7:/data**
更多命令帮助请执行 showmount --help 或 man showmount		

10.13.2　exportfs命令说明

1.exportfs命令介绍

exportfs-rv命令相当于/etc/init.d/nfs reload，该带参数的命令用于使新加载的配置生效，除此之外，通过exportfs命令，我们还可以管理当前NFS共享的文件系统目录列表。

在启动了NFS服务之后，如果此时修改了/etc/exports，就需要重新启动NFS，使修改的配置生效。这个时候就可以用exportfs命令来完成。下面就看看exportfs的语法，如下：

```
/usr/sbin/exportfs [-avi] [-o options,


..] [client:


/path ..]
/usr/sbin/exportfs -r [-v] 相当于


/etc/init.d/nfs reload
/usr/sbin/exportfs [-av] -u [client:


/path ..]
/usr/sbin/exportfs [-v]
/usr/sbin/exportfs -f
```

或

```
```

```
/usr/sbin/exportfs [ -a ] [ -v ] [ -u ] [ -i ] [ -fFile ] [ -oOption
[ ,


Option ... ] ] [ Directory ]
```

2.exportfs命令参数的使用方法

1）重新加载配置使修改生效，实现这个功能的命令为：exportfs-rv，该命令相当于/etc/init.d/nfs reload。

2）管理当前NFS共享的文件系统目录列表。

下面的示例将不通过/etc/exports文件，而是通过exportfs命令来共享NFS存储目录，具体命令如下。

```
[root@C64-A ~]# exportfs -o rw,


sync 10.0.0.7:


/data
[root@C64-A ~]# showmount -e localhost
Export list for localhost:



/mnt            10.0.0.7
/data 10.0.0.0/24  #<==这就是上述


exportfs命令共享


/data目录的结果
```

```
/data/r_shared 10.0.0.0/24
/data      10.0.0.0/24
```

还可以指定其他参数共享/data目录，例如增加了“all_squash，anonuid=888，anongid=888”3个参数，设置NFS共享的命令为：

```
[root@oldboy data]# exportfs -o rw,


sync,


all_squash,


anonuid=888,


anongid=888 10.0.0.0/24:


/data
[root@oldboy data]# showmount -e localhost|grep data
/data  10.0.0.0/24
```

10.13.3 RPC

1.RPC原理描述

什么是RPC（Remote Procedure Call）？从字面上翻译，意思是远程过程调用，其实就是一些程序（Program）在执行远程联机时，需要用到的程序！简单地说，当我们在使用某些服务来进行远程联机的时候，有些信息（例如主机的IP、服务的port、与对应到的服务PID等），需要管理与之对应！这个管理的工作就是RPC的任务所在！

如果我们将NFS与RPC两者的相关性连接起来，那么应该就可以知道：NFS本身的服务并没有提供数据传递的协议，但是NFS却能让我们访问其提供的文件，这其中的原因，就是NFS使用到了一些其他相关的传输协议！而这些传输的协议，就使用到这个RPC的功能！

2.RPC相关命令

命令如下：

```
[root@nfs-client ~]# rpcinfo用法：

 rpcinfo [ -n 端口号

 ] -u 主机名
```

程序号

[版本号

]
        rpcinfo [-n 端口号

] -t 主机

程序号

[版本号

]
        rpcinfo -p [主机

]
        rpcinfo -b 程序号

版本号

        rpcinfo -d 程序号

版本号

```
[root@nfs-client ~]# rpcinfo -p 10.0.0.14
```
    程序

       版本

  协议

端口

```
100000    2    tcp    111  portmapper
100000    2    udp    111  portmapper
100024    1    udp    743  status
100024    1    tcp    746  status
100011    1    udp    774  rquotad
100011    2    udp    774  rquotad
100011    1    tcp    777  rquotad
100011    2    tcp    777  rquotad
100003    2    udp   2049  nfs
100003    3    udp   2049  nfs
100003    4    udp   2049  nfs
100021    1    udp  33116  nlockmgr
100021    3    udp  33116  nlockmgr
100021    4    udp  33116  nlockmgr
100003    2    tcp   2049  nfs
100003    3    tcp   2049  nfs
100003    4    tcp   2049  nfs
100021    1    tcp  37085  nlockmgr
100021    3    tcp  37085  nlockmgr
100021    4    tcp  37085  nlockmgr
100005    1    udp    801  mountd
100005    1    tcp    804  mountd
100005    2    udp    801  mountd
100005    2    tcp    804  mountd
100005    3    udp    801  mountd
100005    3    tcp    804  mountd
```

一般NFS Client端只启动rpcbind即可完成NFS的挂载服务。

10.13.4　NFS服务器端的防火墙控制

真正企业生产环境的存储服务器都属于内网环境，无需防火墙，因此，可以不对其进行配置，如果要配置的话就有两种方法，任选其一即可。

方法一，仅允许内部IP段访问（最佳）。

```
iptables -A INPUT -s 10.0.0.0/24  -j ACCEPT
```

方法二，允许IP段加端口访问。

```
iptables -A INPUT -i eth1 -p tcp -s 10.0.0.0/24 --dport 111 -j ACCEPT
iptables -A INPUT -i eth1 -p udp -s 10.0.0.0/24 --dport 111 -j ACCEPT
iptables -A INPUT -i eth1 -p udp -s 10.0.0.0/24 --dport 2049 -j ACCEPT
iptables -A INPUT -i eth1 -p udp -s 10.0.0.0/24  -j ACCEPT
```

10.13.5　NFS常见故障排查

示例1：客户端挂载报错“No such file or directory”。

```
[root@nfs-client1 ~]# showmount -e 192.168.0.11
Export list for 192.168.0.11:



/data 192.168.0.0/16
[root@nfs-client1 ~]# mount -t nfs 192.168.0.11:


/data /mnt
mount:


 192.168.0.11:


/data failed,


 reason given by server:


 No such file or directory
```

解答：原因是NFS服务器端没有共享目录/data，创建即可。命令如下：

```
[root@nfs-server~]1# mkdir /data
```

示例2：NFS服务器端启动失败，如下：

```
[root@nfs-server ~]# /etc/init.d/nfs start
Starting NFS services:


  [  OK  ]
Starting NFS quotas:


 Cannot register service:


 RPC:


 Unable to receive:


 errno = Connection refused
rpc.rquotad:


 unable to register  (


RQUOTAPROG,


 RQUOTAVERS,


 udp)


.
[FAILED]
Starting NFS daemon:


 [FAILED]
```

解答：这是因为RPC服务没有在NFS前面启动，需要先启动RPC服务再启动NFS。

解决方法为，按顺序启动rpcbind及NFS，命令如下：

```
[root@nfs-server ~]# /etc/init.d/ rpcbind restart
[root@nfs-server ~]# /etc/init.d/ nfs restart
```

示例3：注册RPC服务失败，出现failed：RPC Error：Program not registered错误。

```
[root@nfs-client ~]# mount -t nfs 192.168.0.11:


/data /mnt
mount:


 mount to NFS server '192.168.0.11' failed:


 RPC Error:


 Program not registered.
```

解答：服务器端的NFS没有启动，客户端没有收到服务器端发来的随机端口信息。

解决方法如下：

```
[root@nfs-server ~]# /etc/init.d/ rpcbind restart
[root@nfs-server ~]# /etc/init.d/ nfs restart
```

示例4：卸载挂载设备时显示device is busy。

```
[root@nfs-client mnt]# umount /mnt
umount:


 /mnt:


 device is busy
umount:


 /mnt:


 device is busy
```

解答：有可能是当前目录就是挂载的NFS目录（/mnt），也有可能是NFS Server挂了。对于第一种情况，解决办法为退出挂载的目录/mnt，再执行umount/mnt卸载。对于第二种情况，NFS Server挂了，NFS Client就会出问题（df-h窗口会死掉）。这时只能强制卸载，方法为：

umount-lf/mnt其中的参数-f为强制卸载，参数-l为懒惰的卸载。

示例5：CentOS 6.6客户端NFS挂载时遇到问题。

```
[root@nfs-client1 sbin]# mount -t nfs 10.0.0.7:


/data/ /mnt
mount:


 wrong fs type,


 bad option,
```

```
 bad superblock on 10.0.0.7:

/data/,

    missing codepage or helper program,

 or other error
     (

for several filesystems  (

e.g. nfs,

 cifs)

 you might
    need a /sbin/mount.<type> helper program)


    In some cases useful info is found in syslog - try
    dmesg | tail  or so
```

排查思路：同样的机器，NFS服务器本地可以挂载，但是客户端挂载报wrong fs type，因此尝试所有客户端安装nfs-utils。CentOS 6.5及以前的NFS没有遇到这个问题。

解决方法：执行yum install nfs-utils-y，客户端安装NFS软件，但不启动服务。

10.14　本章重点回顾

1）NFS服务的访问原理流程（会口述）。

2）NFS作为集群共享存储角色的搭建、部署。

3）NFS作为集群共享存储角色的排障，高级优化（会口述）。

4）mount命令的知识及参数，如-o（noatime，nodirtime，noexec，nosuid，rsize，wsize）等。

5）fstab文件的知识。

6）常用命令showmount，exportfs，umount（-lf），rpcinfo。

7）NFS的优点、缺点，适合的应用场景，替代产品（FastDFS、Moosefs（mfs）、GlusterFS）。

8）了解autofs。

10.15 本章参考资料

·http://nfs.sourceforge.net/nfs-howto/

·http://nfs.sourceforge.net/nfs-howto/ar01s02.html#whatis_nfs

·http://ben.timby.com/p=109

·http://www.faqs.org/rfcs/rfc1094.html

·http://www.tldp.org/HOWTO/NFS-HOWTO/index.html

·http://www.citi.umich.edu/projects/nfsv4/linux/

·http://www.vanemery.com/Linux/NFSv4/NFSv4-no-rpcsec.html

·http://www.ibm.com/developerworks/cn/linux/l-network-filesystems/

第11章　Nginx反向代理与负载均衡应用实践

11.1　集群简介

简单地说，集群就是指一组（若干个）相互独立的计算机，利用高速通信网络组成的一个较大的计算机服务系统，每个集群节点（即集群中的每台计算机）都是运行各自服务的独立服务器。这些服务器之间可以彼此通信，协同向用户提供应用程序、系统资源和数据，并以单一系统的模式加以管理。当用户客户机请求集群系统时，集群给用户的感觉就是一个单一独立的服务器，而实际上用户请求的是一组集群服务器。

打开谷歌、百度的页面，看起来好简单，也许你觉得用几分钟就可以制作出相似的网页，而实际上，这个页面的背后是由成千上万台服务器集群协同工作的结果。而这么多的服务器维护和管理，以及相互协调工作也许就是读者你未来的工作职责了。

若要用一句话描述集群，即一堆服务器合作做同一件事，这些机器可能需要整个技术团队架构、设计和统一协调管理，这些机器可以分布在一个机房，也可以分布在全国全球各个地区的多个机房。图11-1是一个中等规模网站集群架构逻辑图。

11.2　为什么要使用集群

想了解为什么要使用集群，那么就要从集群的特点讲起。

1.高性能（Performance）

一些国家重要的计算密集型应用（如天气预报、核试验模拟等），需要计算机有很强的运算处理能力。以全世界现有的技术，即使是大型机，其计算能力也是有限的，很难单独完成此任务。因为计算时间可能会相当长，也许几天，甚至几年或更久。因此，对于这类复杂的计算业务，便使用了计算机集群技术，集中几十上百台，甚至成千上万台计算机进行计算。

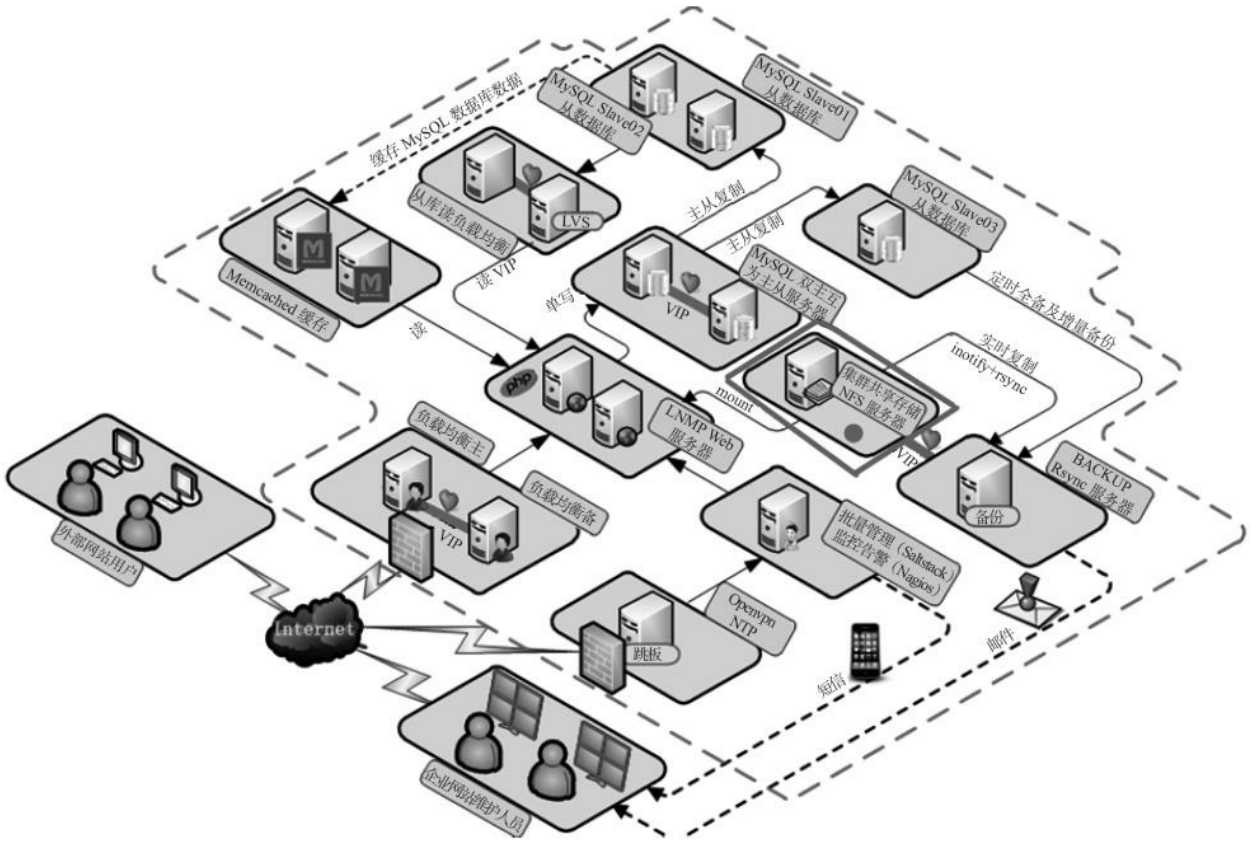
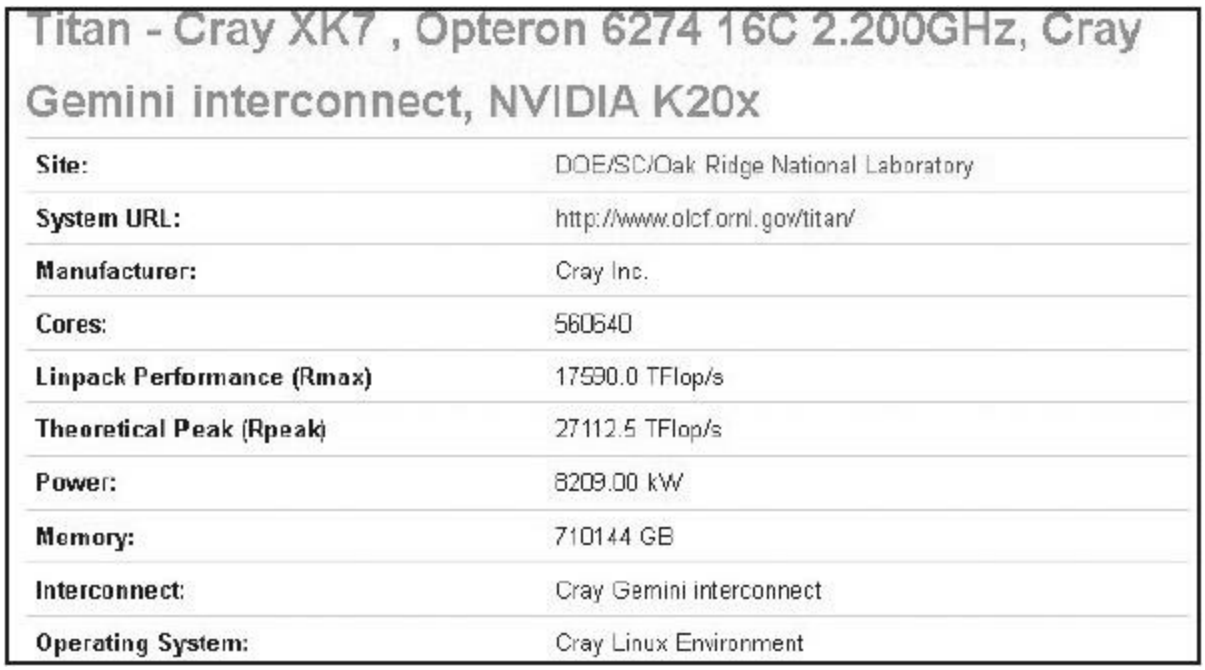


图11-1　中等规模网站集群架构逻辑图

大家耳熟能详的大型网站谷歌、百度、淘宝等，都不是几台大型机可以构建的，都是上万台服务器组成的高性能集群，分布于不同的地点。图11-2中是某大型计算机设备的超强硬件配置。



Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x

Site:	DOE/SC/Oak Ridge National Laboratory
System URL:	http://www.olcf.ornl.gov/titan/
Manufacturer:	Cray Inc.
Cores:	560640
Linpack Performance (Rmax)	17590.0 TFlop/s
Theoretical Peak (Rpeak)	27112.5 TFlop/s
Power:	8209.00 kW
Memory:	710144 GB
Interconnect:	Cray Gemini interconnect
Operating System:	Cray Linux Environment

图11-2　某大型计算机设备的硬件配置

假如你配一个LNMP环境，每次只需要服务10个并发请求，那么单台服务器一定会比多个服务器集群要快。只有当并发或总请求数量超过单台服务器的承受能力时，服务器集群才会体现出优势。

2.价格有效性（Cost-effectiveness）

通常一套系统集群架构，只需要几台或数十台服务器主机即可。与动辄价值上百万元的专用超级计算机相比便宜了很多。在达到同样性能需求的条件下，采用计算机集群架构比采用同等运算能力的大型计算机具有更高的性价比。

早期的淘宝、支付宝的数据库等核心系统就是使用上百万元的小型机服务器。后因使用维护成本太高以及扩展设备费用成几何级数翻倍，

甚至成为扩展瓶颈，人员维护也十分困难，最终使用PC服务器集群替换之，比如，把数据库系统从小机结合Oracle数据库迁移到MySQL开源数据库结合PC服务器上来。不但成本下降了，扩展和维护也更容易了。

3.可伸缩性（Scalability）

当服务负载、压力增长时，针对集群系统进行较简单的扩展即可满足需求，且不会降低服务质量。

通常情况下，硬件设备若想扩展性能，不得不增加新的CPU和存储器设备，如果加不上去了，就不得不购买更高性能的服务器，就拿我们现有的服务器来讲，可以增加的设备总是有限的。如果采用集群技术，则只需要将新的单个服务器加入现有集群架构中即可，从访问的客户角度来看，系统服务无论是连续性还是性能上都几乎没有变化，系统在不知不觉中完成了升级，加大了访问能力，轻松地实现了扩展。集群系统中的节点数目可以增长到几千乃至上万个，其伸缩性远超过单台超级计算机。

4.高可用性（Availability）

单一的计算机系统总会面临设备损毁的问题，如CPU、内存、主板、电源、硬盘等，只要一个部件坏掉，这个计算机系统就可能会宕机，无法正常提供服务。在集群系统中，尽管部分硬件和软件也还是会

发生故障，但整个系统的服务可以是7×24可用的 。

集群架构技术可以使得系统在若干硬件设备故障发生时仍可以继续工作，这样就将系统的停机时间减少到了最小。集群系统在提高系统可靠性的同时，也大大减小了系统故障带来的业务损失，目前几乎100%的互联网网站都要求7×24小时提供服务。

5.透明性（Transparency）

多个独立计算机组成的松耦合集群系统构成一个虚拟服务器。用户或客户端程序访问集群系统时，就像访问一台高性能、高可用的服务器一样，集群中一部分服务器的上线、下线不会中断整个系统服务，这对用户也是透明的。

6.可管理性（Manageability）

整个系统可能在物理上很大，但其实容易管理 ，就像管理一个单一映像系统一样。在理想状况下，软硬件模块的插入能做到即插即用（Plug&Play）。

7.可编程性（Programmability）

在集群系统上，容易开发及修改各类应用程序。

11.3 集群的分类

1.集群的常见分类

计算机集群架构按功能和结构可以分成以下几类：

·负载均衡集群（Load balancing clusters），简称LBC或者LB。

·高可用性集群（High-availability（HA）clusters），简称HAC。

·高性能计算集群（High-performance（HP）clusters），简称HPC。

·网格计算（Grid computing）集群。


提示：负载均衡集群和高可用性集群是互联网行业常用的集群架构模式，也是接下来内容的重点。

2.不同种类的集群介绍

（1）负载均衡集群

负载均衡集群为企业提供了更为实用、性价比更高的系统架构解决方案。负载均衡集群可以把很多客户集中的访问请求负载压力尽可能平均地分摊在计算机集群中处理。客户访问请求负载通常包括应用程序处理负载和网络流量负载。这样的系统非常适合使用同一组应用程序为大

量用户提供服务的模式，每个节点都可以承担一定的访问请求负载压力，并且可以实现访问请求在各节点之间动态分配，以实现负载均衡。

负载均衡集群运行时，一般是通过一个或多个前端负载均衡器将客户访问请求分发到后端的一组服务器上，从而达到整个系统的高性能和高可用性。一般高可用性集群和负载均衡集群会使用类似的技术，或同时具有高可用性与负载均衡的特点。

负载均衡集群的作用为：

·分担 用户访问请求及数据流量（负载均衡）。

·保持业务连续性，即7×24小时服务（高可用性）。

·应用于Web业务及数据库从库等服务器的业务。

负载均衡集群典型的开源软件包括LVS、Nginx、Haproxy等。其架构图如图11-3所示。

（2）高可用性集群

一般是指在集群中任意一个节点失效的情况下，该节点上的所有任务会自动转移到其他正常的节点上。此过程并不影响整个集群的运行。

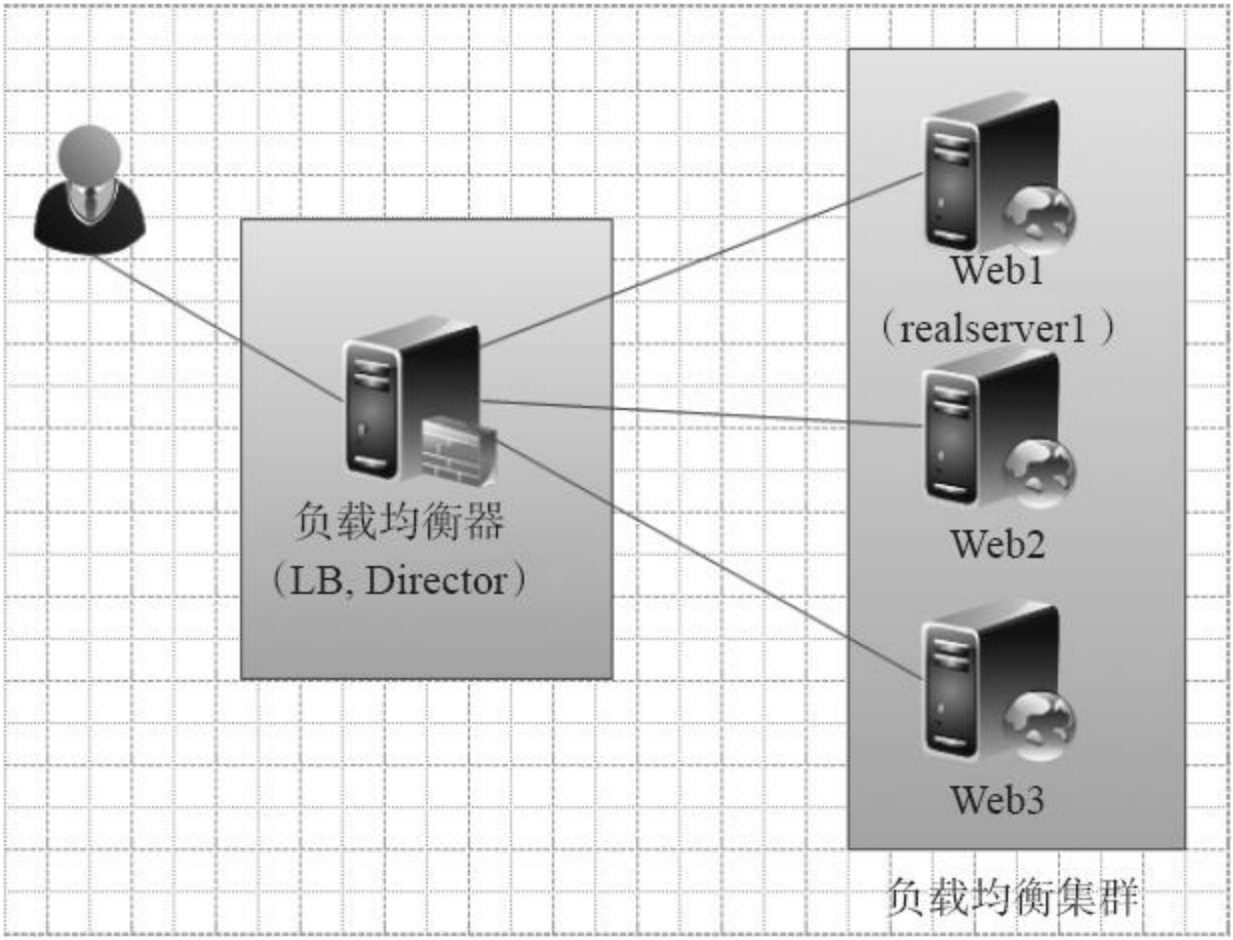


图11-3　服务器节点负载均衡图示

提示：不同的业务会有若干秒的切换时间，DB业务明显长于Web业务切换时间。

当集群中的一个节点系统发生故障时，运行着的集群服务会迅速做出反应，将该系统的服务分配到集群中其他正在工作的系统上运行。考虑到计算机硬件和软件的容错性，高可用性集群的主要目的是使集群的整体服务尽可能可用。如果高可用性集群中的主节点发生了故障，那么这段时间内将由备节点代替它。备节点通常是主节点的镜像。当它代替

主节点时，它可以完全接管主节点（包括IP地址及其他资源）提供服务，因此，使集群系统环境对于用户来说是一致的，即不会影响用户的访问。

高可用性集群使服务器系统的运行速度和响应速度会尽可能的快。它们经常利用在多台机器上运行的冗余节点和服务来相互跟踪。如果某个节点失败，它的替补者将在几秒钟或更短时间内接管它的职责。因此，对于用户而言，集群里的任意一台机器宕机，业务都不会受影响（理论情况下）。

高可用性集群的作用为：

·当一台机器宕机时，另外一台机器接管宕机的机器的IP资源和服务资源，提供服务。

·常用于不易实现负载均衡的应用，比如负载均衡器，主数据库、主存储对之间。

高可用性集群常用的开源软件包括Keepalived、Heartbeat等，其架构图如图11-4所示。

（3）高性能计算集群

高性能计算集群也称并行计算。通常，高性能计算集群涉及为集群开发的并行应用程序，以解决复杂的科学问题（天气预报、石油勘探、

核反应模拟等）。高性能计算集群对外就好像一个超级计算机，这种超级计算机内部由数十至上万个独立服务器组成，并且在公共消息传递层上进行通信以运行并行应用程序。在老男孩的工作中实际是把任务切成蛋糕，然后下发到集群节点计算，计算后返回结果，然后继续领新任务计算，如此往复。

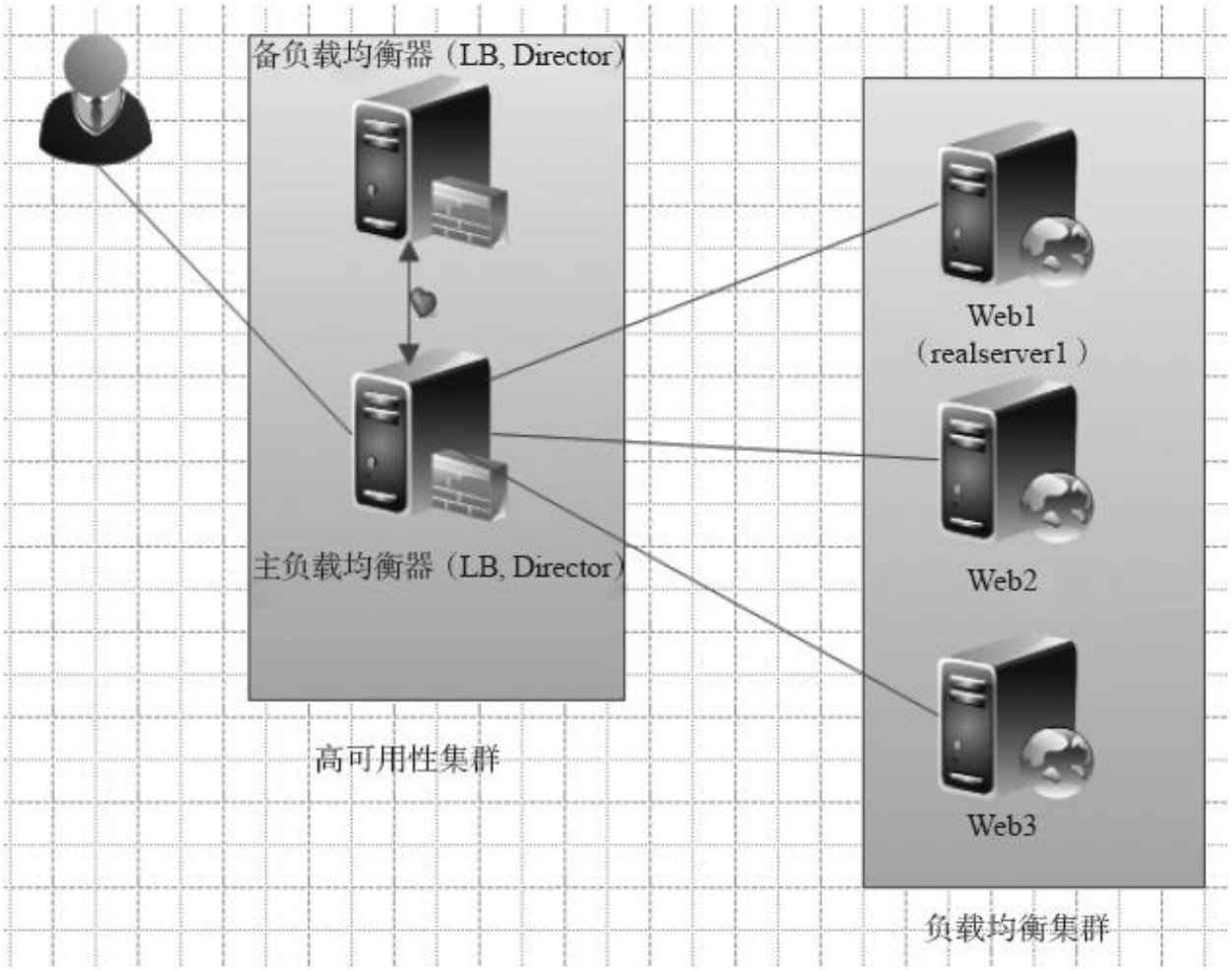


图11-4　服务器负载均衡器高可用图示

（4）网格计算集群

由于很少用到，在此从略。

 特别提示：在互联网网站运维中，比较常用的是负载均衡集群和高可用性集群

11.4　常用的集群软硬件介绍及选型

1.企业运维中常见的集群软硬件产品

互联网企业常用的开源集群软件有：Nginx、LVS、Haproxy、Keepalived、Heartbeat。

互联网企业常用的商业集群硬件有：F5、Netscaler、Radware、A10等，工作模式相当于Haproxy的工作模式。

淘宝、赶集网、新浪等公司曾使用过Netscaler负载均衡产品。集群硬件Netscaler的产品图如图11-5所示，相关指标请大家自行查询相关资料。

集群硬件F5产品图如图11-6所示，相关指标请大家自查相关资料。

2.对于集群软硬件产品如何选型

下面是老男孩的基本选择建议，更多的建议等大家读完负载均衡内容后再细分讲解。

图11-5　Netscaler负载均衡产品

图11-6　F5负载均衡产品图

·当企业业务重要，技术力量又薄弱，并且希望出钱购买产品及获取更好的服务时，可以选择硬件负载均衡产品，如F5、Netscaler、Radware等，此类公司多为传统的大型非互联网企业，如银行、证券、金融业及宝马、奔驰公司等。

·对于门户网站来说，大多会并用软件及硬件产品来分担单一产品的风险，如淘宝、腾讯、新浪等。融资了的企业会购买硬件产品，如赶集等网站。

·中小型互联网企业，由于起步阶段无利润可赚或者利润很低，会希望通过使用开源免费的方案来解决问题，因此会雇佣专门的运维人员进行维护。例如：51CTO等。

相比较而言，商业的负载均衡产品成本高，性能好，更稳定，缺点是不能二次开发，开源的负载均衡软件对运维人员的能力要求较高，如果运维及开发能力强，那么开源的负载均衡软件是不错的选择，目前的互联网行业更倾向于使用开源的负载均衡软件。

3.如何选择开源集群软件产品

中小企业互联网公司网站在并发访问和总访问量不是很大的情况下，建议首选Nginx负载均衡，理由是Nginx负载均衡配置简单、使用方便、安全稳定、社区活跃，使用的人逐渐增多，成为流行趋势，另外一个实现负载均衡的类似产品为Haproxy（支持L4和L7负载，同样优秀，但社区不如Nginx活跃）。

如果要考虑Nginx负载均衡的高可用功能，建议首选Keepalived软件，理由是安装和配置简单，使用方便，安全稳定，与Keepalived服务类似的高可用软件还有Heartbeat（使用比较复杂，不建议初学者使用）。

如果是大型企业互联网公司，负载均衡产品可以使用LVS+Keepalived在前端做四层转发（一般是主备或主主，如果需要扩展可以使用DNS或前端使用OSPF），后端使用Nginx或者Haproxy做7层转发（可以扩展到百台），再后面是应用服务器，如果是数据库与存储的负载均衡和高可用，建议选择LVS+Heartbeat，LVS支持TCP转发且DR模式效率很高，Heartbeat可以配合drbd，不但可以进行VIP的切换，还可以支持块设备级别的数据同步（drbd），以及资源服务的管理。

11.5 Nginx负载均衡集群介绍

11.5.1 搭建负载均衡服务的需求

负载均衡（Load Balance）集群提供了一种廉价、有效、透明的方法，来扩展网络设备和服务器的负载、带宽和吞吐量，同时加强了网络数据处理能力，提高了网络的灵活性和可用性。

搭建负载均衡服务的需求如下：

1）把单台计算机无法承受的大规模并发访问或数据流量分担到多台节点设备上，分别进行处理，减少用户等待响应的时间，提升用户体验。

2）单个重负载的运算分担到多台节点设备上做并行处理，每个节点设备处理结束后，将结果汇总，返回给用户，系统处理能力得到大幅度提高。

3）7×24小时的服务保证，任意一个或多个有限后面节点设备宕机，不能影响业务。

在负载均衡集群中，同组集群的所有计算机节点都应该提供相同的服务。集群负载均衡器会截获所有对该服务的入站请求。然后将这些请

求尽可能地平均地分配在所有集群节点上。

11.5.2　Nginx负载均衡集群介绍

1.反向代理与负载均衡概念简介

严格地说，Nginx仅仅是作为Nginx Proxy反向代理使用的，因为这个反向代理功能表现的效果是负载均衡集群的效果，所以本文称之为Nginx负载均衡。那么，反向代理和负载均衡有什么区别呢？

普通负载均衡软件，例如大名鼎鼎的LVS，其实现的功能只是对请求数据包的转发（也可能会改写数据包）、传递，其中DR模式明显的特征是从负载均衡下面的节点服务器来看，接收到的请求还是来自访问负载均衡器的客户端的真实用户，而反向代理就不一样了，反向代理接收访问用户的请求后，会代理用户重新发起请求代理下的节点服务器，最后把数据返回给客户端用户，在节点服务器看来，访问的节点服务器的客户端用户就是反向代理服务器了，而非真实的网站访问用户。

一句话，LVS等的负载均衡是转发用户请求的数据包，而Nginx反向代理是接收用户的请求然后重新发起请求去请求其后面的节点。

2.实现Nginx负载均衡的组件说明

实现Nginx负载均衡的组件主要有两个，见表11-1。

表11-1　Nginx负载均衡的组件模块

Nginx http 功能模块	模块说明
ngx_http_proxy_module	proxy 代理模块，用于把请求后抛给服务器节点或 upstream 服务器池
ngx_http_upstream_module	负载均衡模块，可以实现网站的负载均衡功能及节点的健康检查

11.6　快速实践Nginx负载均衡环境准备

本节先带大家一起操作实战，让大家对Nginx负载均衡有一个初步的概念，然后再继续深入讲解Nginx负载均衡的核心知识应用。

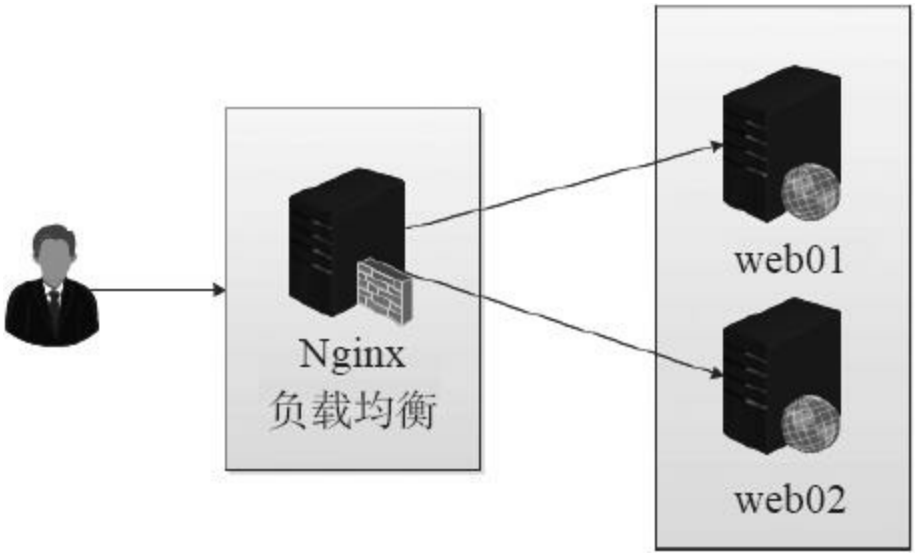


图11-7　快速实践Nginx负载均衡的逻辑架构图

图11-7是快速实践Nginx负载均衡的逻辑架构图。

在图11-7中，所有用户的请求统一发送到Nginx负载均衡器，然后由负载均衡器根据调度算法来请求web01和web02。

11.6.1　软硬件准备

1.硬件准备

准备4台VM虚拟机（有物理服务器更佳），两台做负载均衡，两台做RS，如表11-2所示。

表11-2　Nginx负载均衡实验环境准备

HOSTNAME	IP	说　　明
lb01	10.0.0.7	Nginx 主负载均衡器
lb02	10.0.0.8	Nginx 辅负载均衡器
web01	10.0.0.9	web01 服务器
web02	10.0.0.10	web02 服务器

2.软件准备

系统：CentOS 6.6 x86_64

软件：nginx-1.6.3.tar.gz（http://nginx.org/download/nginx-1.6.3.tar.gz）

11.6.2　安装Nginx软件

下面将在以上4台服务器上安装Nginx。完整的安装过程见本书前面“Nginx应用指南”章节，这里只给出安装的命令部分。

1）安装依赖软件包命令集合。

```
yum install openssl openssl-devel pcre pcre-devel -y
rpm -qa openssl openssl-devel pcre pcre-devel
```

2）安装Nginx软件包命令集合。

```
mkdir -p /home/oldboy/tools
cd /home/oldboy/tools
wget -q http:


// nginx.org/download/nginx-1.6.3.tar.gz
ls -l nginx-1.6.3.tar.gz
useradd nginx -s /sbin/nologin -M
tar xf nginx-1.6.3.tar.gz
cd nginx-1.6.3
./configure  --user=nginx --group=nginx --prefix=/application/nginx-1.6.3 --with-htt
make
make install
ln -s /application/nginx-1.6.3 /application/nginx
```

11.6.3　配置用于测试的Web服务

本小节将在表11-2给出的两台Nginx Web服务器的节点上操作：配置并查看Web服务器的配置结果。

Nginx web01和web02的配置如下：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
```

```
    server {
        listen       80;

        server_name  bbs.etiantian.org;

        location / {
            root   html/bbs;

            index  index.html index.htm;

        }
        access_log  logs/access_bbs.log  main;

    }
    server {
        listen       80;

        server_name  www.etiantian.org;

        location / {
            root   html/www;

            index  index.html index.htm;

        }
        access_log  logs/access_www.log  main;

    }
}
```


提示：这里故意将www虚拟主机放在下面，便于用后面的参数配置测试效果。

配置完成后检查语法，并启动Nginx服务。

```
[root@web01 conf]# mkdir /application/nginx/html/{www,


bbs}
[root@web01 conf]# /application/nginx/sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@web01 conf]# /application/nginx/sbin/nginx
[root@web01 conf]# netstat -lntup|grep nginx
tcp     0     0 0.0.0.0:


80     0.0.0.0:


*     LISTEN     28538/nginx
```

然后填充测试文件数据，如下：

```
[root@web01 conf]# mkdir /application/nginx/html/{www,


bbs}
[root@web01 conf]# for dir in www bbs;


do echo "`ifconfig eth0|grep -o "10.0.0.[109]."` $dir" >/application/nginx/html/$dir
```

```
done
[root@web01 conf]# for dir in www bbs;


do cat /application/nginx/html/$dir/index.html;


done
10.0.0.9  www
10.0.0.9  bbs
```

以上命令需要在web01上操作。

```
[root@web02 conf]# mkdir /application/nginx/html/{www,


bbs}
[root@web02 conf]# for dir in www bbs;


do echo "`ifconfig eth0|grep -o "10.0.0.[109]."` $dir" >/application/nginx/html/$dir


done
[root@web02 conf]# for dir in www bbs;


do cat /application/nginx/html/$dir/index.html;


done
10.0.0.10 www
10.0.0.10 bbs
```

以上命令需要在web02上操作。

 提示：以上配置在两台Web服务器上的操作命令是一样的。

配置解析web01的IP和主机名后，用curl简单测试web01，如下：

```
[root@lb01 conf]# tail -2 /etc/hosts
10.0.0.9 www.etiantian.org
10.0.0.9 bbs.etiantian.org
[root@lb01 conf]# curl www.etiantian.org
10.0.0.9  www
[root@lb01 conf]# curl bbs.etiantian.org
10.0.0.9  bbs
```

配置解析web02的IP和主机名后，用curl简单测试web02，如下：

```
[root@lb01 conf]# tail -2 /etc/hosts
10.0.0.10 www.etiantian.org
10.0.0.10 bbs.etiantian.org
[root@lb01 conf]# curl www.etiantian.org
10.0.0.10 www
[root@lb01 conf]# curl bbs.etiantian.org
10.0.0.10 bbs
```

 提示：

1）不同Web测试节点，返回的结果是不同的，这是为了方便测试演示！

2）通过上面配置就实现了两台Web服务器基于域名的虚拟主机配置。

11.6.4　实现一个简单的负载均衡

本小节将在nginx lb01服务器节点操作（lb02和lb01相同，后文配置负载均衡器高可用时会用到lb02），其准备信息见表11-3。

表11-3　第一台Nginx负载均衡器准备信息

HOSTNAME	IP	说　　明
lb01	10.0.0.7	Nginx 主负载均衡器

下面进行一个简单的Nginx负载均衡配置，代理www.etiantian.org 服务，节点为web01和web02。nginx.conf配置文件内容如下：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;
```

```
    keepalive_timeout  65;

    upstream www_server_pools {     #<==这里是定义
```

Web服务器池，包含了

9，

10两个

Web节点

```
            server 10.0.0.9:
80  weight=1;

             server 10.0.0.10:
80  weight=1;

    }
    server {     #<==这里是定义代理的负载均衡域名虚拟主机

        listen       80;

        server_name  www.etiantian.org;

        location / {
```

```
        proxy_pass http:


// www_server_pools;


 #<==访问


www.etiantian.org，请求发送给


www_server_pools里面的节点



        }
    }
}
```

现在，检查语法并启动。命令如下：

```
[root@lb01 conf]# /application/nginx/sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@lb01 conf]# /application/nginx/sbin/nginx
[root@lb01 conf]# netstat -lntup|grep nginx
tcp     0     0 0.0.0.0:


80     0.0.0.0:


*     LISTEN     2026/nginx
```

然后，检查负载均衡测试结果。Linux作为客户端的测试结果如

下：

```
[root@lb01 conf]# tail -1 /etc/hosts
10.0.0.7 www.etiantian.org #<== 10.0.0.7为负载均衡器的


IP。



[root@lb01 conf]# curl www.etiantian.org
10.0.0.9  bbs
[root@lb01 conf]# curl www.etiantian.org
10.0.0.10 bbs
[root@lb01 conf]# curl www.etiantian.org
10.0.0.9  bbs
[root@lb01 conf]# curl www.etiantian.org
10.0.0.10 bbs
```

再来看看Windows客户端的测试结果。首先要为www.etiantian.org做hosts解析，Windows系统hosts文件的路径为：C:\Windows\System32\drivers\etc\hosts。

解析的内容为：

```
10.0.0.7    www.etiantian.org
```

解析完毕保存后，通过浏览器多次访问www.etiantian.org查看结果如图11-8所示。

图11-8　Web测试节点访问效果

可以看到，无论是Linux客户端还是Windows客户端，结果都是Nginx把请求一比一地分配到了Nginx后面的节点上。

下面宕掉任意一个Web节点，看看测试结果如何，测试如下：

```
[root@lb01 conf]# curl www.etiantian.org
10.0.0.10 bbs
[root@lb01 conf]# curl www.etiantian.org
10.0.0.10 bbs
[root@lb01 conf]# curl www.etiantian.org
10.0.0.10 bbs
```

可以看到，网站业务不受影响，访问请求都定位到了正常的节点上。

现在，宕掉所有Web节点，此时，访问测试结果如下：

```
[root@lb01 conf]# curl www.etiantian.org
<html>
<head><title>502 Bad Gateway</title></head>
<body bgcolor="white">
<center><h1>502 Bad Gateway</h1></center>
<hr><center>nginx/1.6.3</center>
</body>
</html>
```

可以看到，Nginx代理下面没有节点了，因此，Nginx向用户报告了502错误。

如果同时开启所有的Web服务又会怎样？测试结果如下：

```
[root@lb01 conf]# curl www.etiantian.org
10.0.0.10 bbs
[root@lb01 conf]# curl www.etiantian.org
10.0.0.9  bbs
[root@lb01 conf]# curl www.etiantian.org
10.0.0.10 bbs
[root@lb01 conf]# curl www.etiantian.org
10.0.0.9  bbs
```

结果是Nginx又把请求一比一分配到了Nginx后面的节点上。

怎么样，Nginx的负载均衡很简单吧？对的，就是这么简单！

11.7　Nginx负载均衡核心组件介绍

11.7.1　Nginx upstream模块

1.upstream模块介绍

Nginx的负载均衡功能依赖于ngx_http_upstream_module模块，所支持的代理方式包括proxy_pass、fastcgi_pass、memcached_pass等，新版Nginx软件支持的方式所有增加。本文主要讲解proxy_pass代理方式。

ngx_http_upstream_module模块允许Nginx定义一组或多组节点服务器组，使用时可以通过proxy_pass代理方式把网站的请求发送到事先定义好的对应Upstream组的名字上，具体写法为“proxy_pass http://www_server_pools”，其中www_server_pools就是一个Upstream节点服务器组名字。ngx_http_upstream_module模块官方地址为：http://nginx.org/en/docs/http/ngx_http_upstream_module.html

2.upstream模块语法

upstream模块的语法相当简单，这里直接上范例给大家讲解。

范例1：基本的upstream配置案例

```
#blog lb by oldboy at 201303 server pools
```

```
    upstream www_server_pools {
#<== upstream是关键字必须要有，后面的

www_server_pools为一个

Upstream集群组的名字，可以自己起名，调用时就用这个名字

        server 10.0.0.17:

80  weight=5;

#<==server关键字是固定的，后面可以接域名（门户会用）或

IP。如果不指定端口，默认是

80端口。

weight代表权重，数值越大被分配的请求越多，结尾有分号，别忘了

        server 10.0.0.18:

80  weight=10;

        server 10.0.0.19:

82  weight=15;

    }
```

范例2：较完整的upstream配置案例

```
upstream blog_server_pool {
    server   10.0.10.5;


     #<==这一行标签和下一行是等价的


    server   10.0.10.6;


80 weight=1 max_fails=1 fail_timeout=10s;


 #<==这一行标签和上一行是等价的，此行多余的部分就是默认配置，不写也可以


    #the server config above is the same.
    server   10.0.10.7;


80 weight=1 max_fails=2 fail_timeout=20s backup;


#<==server最后面可以加很多参数，具体参数作用看下文表格


    server   10.0.10.8;


80 weight=1 max_fails=2 fail_timeout=20s backup;


}
```

范例3：使用域名及socket的upstream配置案例

```
upstream backend {
    server backend1.example.com weight=5;

    server backend2.example.com:

8080;

     #<==域名加端口。转发到后端的指定端口上

    server unix:

/tmp/backend3;

     #<==指定

socket文件
```

 提示：server后面如果接域名，需要内网有DNS服务器或者在负载均衡器的hosts文件做域名解析。

```
server 192.168.1.2
server 192.168.1.3:

8080
    server backup1.example.com:

8080   backup;
```

```
#<==备份服务器，等上面指定的服务器都不可访问的时候会启用，

backup的用法和

Haproxy中用法一样

    server backup2.example.com:

8080   backup;

}
```

如果是两台Web服务器做高可用，常规方案就需要Keepalived配合，那么这里使用Nginx的backup参数通过负载均衡功能就可以实现Web服务器集群了，对于企业应用来说，能做集群就不做高可用。

3.upstream模块相关说明

upstream模块的内容应放于nginx.conf配置的http{}标签内，其默认调度节点算法是wrr（weighted round-robin，即权重轮询）。表11-4为upstream模块内部server标签部分参数说明。

表11-4　upstream模块内部server标签参数说明

upstream 模块内参数	参数说明
server 10.0.10.8:80	负载均衡后面的 RS 配置，可以是 IP 或域名，如果端口不写，默认是 80 端口。高并发场景下，IP 可换成域名，通过 DNS 做负载均衡
weight=1	代表服务器的权重，默认值是 1。权重数字越大表示接受的请求比例越大
max_fails=1	Nginx 尝试连接后端主机失败的次数，这个数值是配合 proxy_next_upstream、fastcgi_next_upstream 和 memcached_next_upstream 这三个参数来使用的，当 Nginx 接收后端服务器返回这三个参数定义的状态码时，会将这个请求转发给正常工作的后端服务器，例如 404、502、503。Max_fails 的默认值是 1；**企业场景下建议 2 ~ 3 次。如京东 1 次，蓝汛 10 次**，根据业务需求去配置
backup	热备配置（RS 节点的高可用），当前面激活的 RS 都失败后会自动启用热备 RS。这标志着这个服务器作为备份服务器，若主服务器全部宕机了，就会向它转发请求；注意，当负载调度算法为 ip_hash 时，后端服务器在负载均衡调度中的状态不能是 weight 和 backup
fail_timeout=10s	在 max_fails 定义的失败次数后，距离下次检查的间隔时间，默认是 10s；如果 max_fails 是 5，它就检测 5 次，如果 5 次都是 502。那么，它就会根据 fail_timeout 的值，等待 10s 再去检查，还是只检查一次，如果持续 502，在不重新加载 Nginx 配置的情况下，每隔 10s 都只检测一次。常规业务 2 ~ 3 秒比较合理，比如京东 3 秒，蓝汛 3 秒，可根据业务需求去配置
down	这标志着服务器永远不可用，这个参数可配合 ip_hash 使用

提示：以上的参数与专业的Haproxy参数很类似，但不如Haproxy的参数易懂。

来看个示例，如下：

```
upstream backend {
    server backend1.example.com  weight=5;


 <==如果就是单个


Server，没必要设置权重


    server 127.0.0.1:


8080           max_fails=5 fail_timeout=10s;
```

```
<==当检测次数等于

5的时候，

5次连续检测失败后，间隔

10s再重新检测，这个参数和

proxy/fasrcgi/memcached_next_upstream相关；

    server unix：

/tmp/backend3；

    server backup1.example.com：

8080 backup；

  <==热备机器设置

}
```

需要特别说明的是，如果是Nginx代理Cache服务，可能需要使用hash算法，此时若宕机，可通过设置down参数确保客户端用户按照当前的hash算法访问，这一点很重要。示例配置如下：

```
upstream backend {
```

```
    ip_hash;

    server backend1.example.com;

    server backend2.example.com;

    server backend3.example.com down;

    server backend4.example.com;

}
```

下面是Haproxy负载均衡器server标签的配置示例。

```
# 开启对后端服务器的健康检测，通过

GET /test/index.php来判断后端服务器的健康情况

    server php_server_1 10.12.25.68:

80 cookie 1 check inter 2000 rise 3 fall 3 weight 2
    server php_server_2 10.12.25.72:

80 cookie 2 check inter 2000 rise 3 fall 3 weight 1
    server php_server_bak 10.12.25.79:

80 cookie 3 check inter 1500 rise 3 fall 3 backup
```

上述命令的说明如下：

·weight：调节服务器的请求分配权重。

·check：开启对该服务器健康检查。

·inter：设置连续两次的健康检查间隔时间，单位毫秒，默认值2000。

·rise：指定多少次连续成功的健康检查后，即可认定该服务器处于可用状态。

·fall：指定多少次不成功的健康检查后，即认为服务器为宕机状态，默认值3。

·maxconn：指定可被发送到该服务器的最大并发连接数。

更多的Nginx upstream模块参数请参考：http://nginx.org/en/docs/http/ngx_http_ups-tream_module.html 。

4.upstream模块调度算法

调度算法一般分为两类，第一类为静态调度算法，即负载均衡器根据自身设定的规则进行分配，不需要考虑后端节点服务器的情况，例如：rr、wrr、ip_hash等都属于静态调度算法。

第二类为动态调度算法，即负载均衡器会根据后端节点的当前状态来决定是否分发请求，例如：连接数少的优先获得请求，响应时间短的优先获得请求。例如：least_conn、fair等都属于动态调度算法。

下面介绍一下常见的调度算法。

（1）rr轮询（默认调度算法，静态调度算法）

按客户端请求顺序把客户端的请求逐一分配到不同的后端节点服务器，这相当于LVS中的rr算法，如果后端节点服务器宕机（默认情况下Nginx只检测80端口），宕机的服务器会被自动从节点服务器池中剔除，以使客户端的用户访问不受影响。新的请求会分配给正常的服务器。

（2）wrr（权重轮询，静态调度算法）

在rr轮询算法的基础上加上权重，即为权重轮询算法，当使用该算法时，权重和用户访问成正比，权重值越大，被转发的请求也就越多。可以根据服务器的配置和性能指定权重值大小，有效解决新旧服务器性能不均带来的请求分配问题。

举个例子帮助大家加深理解。

后端服务器192.168.1.2的配置为：E5520*2 CPU，8GB内存。

后端服务器192.168.1.3的配置为：Xeon（TM）2.80GHz*2，4GB内存。

假设希望在有30个请求到达前端时，其中20个请求交给192.168.1.3处理，剩余10个请求交给192.168.1.2处理，就可做如下配置；

```
upstream oldboy_lb {
server 192.168.1.2 weight=1;



server 192.168.1.3 weight=2;


}
```

权重测试的例子见前文快速实现一个简单的负载均衡内容部分。

（3）ip_hash（静态调度算法）

每个请求按客户端IP的hash结果分配，当新的请求到达时，先将其客户端IP通过哈希算法哈希出一个值，在随后的客户端请求中，客户IP的哈希值只要相同，就会被分配至同一台服务器，该调度算法可以解决动态网页的session共享问题，但有时会导致请求分配不均，即无法保证1：1的负载均衡，因为在国内大多数公司都是NAT上网模式，多个客户端会对应一个外部IP，所以，这些客户端都会被分配到同一节点服务器，从而导致请求分配不均。LVS负载均衡的-p参数、Keepalived配置

里的persistence_timeout 50参数都类似这个Nginx里的ip_hash参数，其功能都可以解决动态网页的session共享问题。

同样也来看一个示例，如下：

```
upstream oldboy_lb {
ip_hash;



server 192.168.1.2:



80;



server 192.168.1.3:



8080;



}
upstream backend {
    ip_hash;



    server backend1.example.com;



    server backend2.example.com;



    server backend3.example.com down;



    server backend4.example.com;
```

```
}
```

注意： 当负载调度算法为ip_hash时，后端服务器在负载均衡调度中的状态不能有weight和backup，即使有也不会生效。

（4）fair（动态调度算法）

此算法会根据后端节点服务器的响应时间来分配请求，响应时间短的优先分配。这是更加智能的调度算法。此种算法可以依据页面大小和加载时间长短智能地进行负载均衡，也就是根据后端服务器的响应时间来分配请求，响应时间短的优先分配。Nginx本身不支持fair调度算法，如果需要使用这种调度算法，必须下载Nginx的相关模块upstream_fair。

示例如下：

```
upstream oldboy_lb {
server 192.168.1.2;


server 192.168.1.3;


fair;


}
```

（5）least_conn

least_conn算法会根据后端节点的连接数来决定分配情况，哪个机器连接数少就分发。

除了上面介绍的这些算法外，还有一些第三方调度算法，例如：url_hash、一致性hash算法等，介绍如下。

（6）url_hash算法

与ip_hash类似，这里是根据访问URL的hash结果来分配请求的，让每个URL定向到同一个后端服务器，后端服务器为缓存服务器时效果显著。在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash_method使用的是hash算法。

url_hash按访问URL的hash结果来分配请求，使每个URL定向到同一个后端服务器，可以进一步提高后端缓存服务器的效率命中率。Nginx本身是不支持url_hash的，如果需要使用这种调度算法，必须安装Nginx的hash模块软件包。

url_hash（web缓存节点）和ip_hash（会话保持）类似。示例配置如下：

```
upstream oldboy_lb {
server squid1:


3128;
```

```
    server squid2;

    3128;

    hash $request_uri;

    hash_method crc32;

    }
```

（7）一致性hash算法

一致性hash算法一般用于代理后端业务为缓存服务（如Squid、Memcached）的场景，通过将用户请求的URI或者指定字符串进行计算，然后调度到后端的服务器上，此后任何用户查找同一个URI或者指定字符串都会被调度到这一台服务器上，因此后端的每个节点缓存的内容都是不同的，一致性hash算法可以解决后端某个或几个节点宕机后，缓存的数据动荡最小，一致性hash算法知识比较复杂，详细内容可以参考后文或相关资料，这里仅仅给出配置示例：

```
    http {
        upstream test {
            consistent_hash $request_uri;

            server 127.0.0.1;
```

```
    9001 id=1001 weight=3;

            server 127.0.0.1:

    9002 id=1002 weight=10;

            server 127.0.0.1:

    9003 id=1003 weight=20;

        }
    }
```

虽然Nginx本身不支持一致性hash算法，但Nginx的分支Tengine支持。详细可见

http://tengine.taobao.org/document_cn/http_upstream_consistent_hash_cn.htn

11.7.2 http_proxy_module模块

1.proxy_pass指令介绍

proxy_pass指令属于ngx_http_proxy_module模块，此模块可以将请求转发到另一台服务器，在实际的反向代理工作中，会通过location功能匹配指定的URI，然后把接收到的符合匹配URI的请求通过proxy_pass抛给定义好的upstream节点池。该指令官方地址见：http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass。

下面proxy_pass的使用案例。

1）将匹配URI为name的请求抛给http://127.0.0.1/remote/。

```
location /name/ {
    proxy_pass http:


// 127.0.0.1/remote/;


}
```

2）将匹配URI为some/path的请求抛给http://127.0.0.1。

```
location /some/path/ {
    proxy_pass http:
```

```
// 127.0.0.1;


}
```

3）将匹配URI为name的请求应用指定的rewrite规则，然后抛给http://127.0.0.1。

```
location /name/ {
    rewrite    /name/ (


[^/]+)


 /usersname=$1 break;


    proxy_pass http:


// 127.0.0.1;


}
```

2.http proxy模块参数

Nginx的代理功能是通过http proxy模块来实现的。默认在安装Nginx时已经安装了http proxy模块，因此可直接使用http proxy模块。下面详细解释模块中每个选项代表的含义，见表11-5。

表11-5　http proxy模块相关参数

http proxy 模块相关参数	说　　明
proxy_set_header	设置 http 请求 header 项传给后端服务器节点，例如：可实现让代理后端的服务器节点获取访问客户端用户的真实 IP 地址
client_body_buffer_size	用于指定客户端请求主体缓冲区大小，此处如了解前文的 http 请求包的原理就好理解了
proxy_connect_timeout	表示反向代理与后端节点服务器连接的超时时间，即发起握手等候响应的超时时间

（续）

http proxy 模块相关参数	说　　明
proxy_send_timeout	表示代理后端服务器的数据回传时间，即在规定时间之内后端服务器必须传完所有的数据，否则，Nginx 将断开这个连接
proxy_read_timeout	设置 Nginx 从代理的后端服务器获取信息的时间，表示连接建立成功后，Nginx 等待后端服务器的响应时间，其实是 Nginx 已经进入后端的排队之中等候处理的时间
proxy_buffer_size	设置缓冲区大小，默认该缓冲区大小等于指令 proxy_buffers 设置的大小
proxy_buffers	设置缓冲区的数量和大小。Nginx 从代理的后端服务器获取的响应信息，会放置到缓冲区
proxy_busy_buffers_size	用于设置系统很忙时可以使用的 proxy_buffers 大小，官方推荐的大小为 proxy_buffers*2
proxy_temp_file_write_size	指定 proxy 缓存临时文件的大小

11.8　Nginx负载均衡配置实战

11.8.1　配置基于域名虚拟主机的Web节点

注意，本小节将在表11-6所示的Nginx Web服务器节点上操作。

表11-6　Nginx Web服务器节点信息

主机名	IP 地址	角色说明
web01	10.0.0.9	nginx web01 服务器
web02	10.0.0.10	nginx web02 服务器

以下操作以web01 10.0.0.9为例讲解，web02 10.0.0.10做同样的操作，其中Nginx的安装过程省略了，有需要的读者可以参考前文。

1）Nginx的配置文件如下：

```
[root@web01 ~]# cat /application/nginx/conf/nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;
```

```
    sendfile        on;

    keepalive_timeout  65;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    server {
        listen       80;

        server_name  bbs.etiantian.org;

        location / {
            root   html/bbs;

            index  index.html index.htm;

        }
        access_log  logs/access_bbs.log  main;

    }
    server {
        listen       80;

        server_name  www.etiantian.org;

        location / {
```

```
            root   html/www;

            index  index.html index.htm;

        }
        access_log  logs/access_www.log  main;

    }
}
```

提示：上面配置了bbs.etiantian.org和www.etiantian.org两个虚拟主机。

2）创建站点目录及对应测试文件，命令如下：

```
[root@web01 html]# for n in bbs www;

do cd /application/nginx/html/;

mkdir -p $n;

echo "$n.etiantian.org9" >$n/index.html;

done
[root@web01 html]# cat bbs/index.html
bbs.etiantian.org9
[root@web01 html]# cat www/index.html
www.etiantian.org9
```

提示：结尾加个数字9来区分对应的Web节点机器。

3）检查语法并重启Nginx服务，命令如下：

```
[root@web01 html]# /application/nginx/sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@web01 html]# /application/nginx/sbin/nginx -s reload
[root@web01 html]# netstat -lntup|grep 80
tcp     0     0 0.0.0.0:


80     0.0.0.0:


*     LISTEN     4998/nginx
```

4）把域名加入hosts解析，本机进行访问测试。

```
#提示：这里的


9为


IP地址，简写。



[root@web01 html]# echo "10.0.0.9 www.etiantian.org" >>/etc/hosts
[root@web01 html]# echo "10.0.0.9 bbs.etiantian.org" >>/etc/hosts
[root@web01 html]# tail -2 /etc/hosts
10.0.0.9 www.etiantian.org
10.0.0.9 bbs.etiantian.org
```

5）检查虚拟主机配置结果。

10.0.0.9 web01上的测试检查结果如下：

```
[root@web01 html]# curl www.etiantian.org
www.etiantian.org9
[root@web01 html]# curl bbs.etiantian.org
bbs.etiantian.org9
```

10.0.0.10 web02上的测试检查结果如下：

```
[root@web02 ~]# curl bbs.etiantian.org
bbs.etiantian.org10
[root@web02 ~]# curl www.etiantian.org
www.etiantian.org10
```

到此，搭建配置虚拟主机就成功了。

 提示：以上web01、web02的虚拟主机配置仅仅作为负载均衡的节点使用，因此，简单搭建能访问出结果即可。

11.8.2　Nginx负载均衡反向代理实践

1.小试牛刀实现为www服务代理

（1）利用upstream定义一组WWW服务器池

先定义一个名字为“www_server_pools”的服务器池，里面有两台Web服务器，具体内容如下：

```
 upstream www_server_pools {
        server 10.0.0.9:

80  weight=1;

        server 10.0.0.10:

80  weight=1;

}
```

提示：

·默认调度算法是weighted round-robin，即权重轮询算法。

·upstream仅仅是定义服务器池，并不会直接处理用户的请求，必须

要有其他方式将请求转给这个服务器池才行。

·虽然定义的是WWW服务器池，但是这个服务器池也可以作为BBS等业务的服务器池。因为节点服务器的虚拟主机都是根据访问的主机头字段区分的。

（2）配置www服务的虚拟主机Server负载代理

下面是Nginx反向代理WWW服务的虚拟主机配置：

```
      server {
          listen       80;


          server_name  www.etiantian.org;


          location / {
          proxy_pass http:

   // www_server_pools;


   #<==通过

   proxy_pass功能把用户的请求交由上面反向代理

   upstream 定义的

   www_server_pools服务器池处理
```

```
        }
        access_log off;


    }
```

（3）实际配置并测试效果

Nginx的实际配置如下：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    upstream www_server_pools {
        server 10.0.0.9;


80  weight=1;
```

```
        server 10.0.0.10:


80  weight=1;



    }
    server {
        listen       80;



        server_name  www.etiantian.org;



        location / {
        proxy_pass http:


// www_server_pools;



        }
    }
}
```

现在配置hosts解析到代理的IP或VIP上，然后重新加载服务，访问测试：

```
[root@lb01 ~]# tail -2 /etc/hosts
10.0.0.7 www.etiantian.org
10.0.0.7 bbs.etiantian.org
[root@lb01 ~]# /application/nginx/sbin/nginx -s reload
[root@lb01 ~]# curl www.etiantian.org
bbs.etiantian.org10
[root@lb01 ~]# curl www.etiantian.org
bbs.etiantian.org9
```

从测试结果可以看出，已经实现了反向代理、负载均衡功能，但是

有一个特殊问题，出来的结果并不是带有www.etiantian.org 的字符串，而是BBS的信息，根据访问结果，我们推测是访问了Web节点下BBS的虚拟主机，明明代理的是WWW虚拟主机，为什么结果是访问了后端的BBS虚拟主机了呢？问题又该如何解决呢？请继续往下看。

2.反向代理多虚拟主机节点服务器企业案例

上一节代理的结果不对，究其原因是当用户访问域名时确实是携带了www.etiantian.org 主机头请求Nginx反向代理服务器，但是反向代理向下面节点重新发起请求时，默认并没有在请求头里告诉节点服务器要找哪台虚拟主机，所以，Web节点服务器接收到请求后发现没有主机头信息，因此，就把节点服务器的第一个虚拟主机发给了反向代理了（而节点上的第一个虚拟主机放置的是故意这样放置的BBS）。解决这个问题的办法，就是当反向代理向后重新发起请求时，要携带主机头信息，以明确告诉节点服务器要找哪个虚拟主机。具体的配置很简单，就是在Nginx代理WWW服务虚拟主机配置里增加如下一行配置即可：

```
proxy_set_header Host  $host;
```

在代理向后端服务器发送的http请求头中加入host字段信息后，若后端服务器配置有多个虚拟主机，它就可以识别代理的是哪个虚拟主机。这是节点服务器多虚拟主机时的关键配置。整个Nginx代理配置

为：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    upstream www_server_pools {
        server 10.0.0.9:


80  weight=1;



        server 10.0.0.10:


80  weight=1;



    }
    server {
        listen       80;
```

```
        server_name  www.etiantian.org;


        location / {
        proxy_pass http:


// www_server_pools;


        proxy_set_header Host  $host;


#<==在代理向后端服务器发送的


http请求头中加入


host字段信息，用于当后端服务器配置有多个虚拟主机时，可以识别代理的是哪个虚拟主机。这是节点服务器多虚


        }
    }
}
```

此时，再重新加载Nginx服务，并用curl测试检查，结果如下：

```
[root@lb01 conf]# /application/nginx/sbin/nginx -s reload
[root@lb01 conf]# curl www.etiantian.org
www.etiantian.org
[root@lb01 conf]# curl www.etiantian.org
www.etiantian.org10
```

可以看到这次访问的结果和访问的域名就完全对应上了，这样代理多虚拟主机的节点服务器就不会出问题了。

3.经过反向代理后的节点服务器记录用户IP企业案例

完成了反向代理WWW服务后，自然很开心，但是，不久后你用其他客户端作为客户端测试时，就会发现一个问题，节点服务器对应的WWW虚拟主机的访问日志的第一个字段记录的并不是客户端的IP，而是反向代理服务器的IP，最后一个字段也是“-”！

例如：使用Windows客户端计算机（IP为10.0.0.2）访问已经解析好代理IP的www.etiantian.org 后，去节点服务器WWW服务的日志查看，会发现如下日志：

```
[root@web01 ~]# tail -2 /application/nginx/logs/access_www.log
10.0.0.7 - - [31/May/2015:


20:


19:


27 +0800] "GET / HTTP/1.0" 200 19 "-" "curl/7.19.7 (


x86_64-redhat-linux-gnu)


 libcurl/7.19.7 NSS/3.15.3 zlib/1.2.3 libidn/1.18 libssh2/1.4.2" "-"
10.0.0.7 - - [31/May/2015:


20:


19:
```

```
28 +0800] "GET / HTTP/1.0" 200 19 "-" "curl/7.19.7 （

x86_64-redhat-linux-gnu）

 libcurl/7.19.7 NSS/3.15.3 zlib/1.2.3 libidn/1.18 libssh2/1.4.2" "-"
```

web01节点服务器对应的WWW虚拟主机的访问日志的第一个字段记录的并不是客户端的IP（10.0.0.2），而是反向代理服务器本身的IP（10.0.0.7），最后一个字段也是一个"-"，那么，如何解决这个问题呢？其实也很简单，同样是增加如下一行参数：

```
proxy_set_header X-Forwarded-For $remote_addr;

#<==这是反向代理时，节点服务器获取用户真实

IP的必要功能配置
```

在反向代理请求后端节点服务器的请求头中增加获取的客户端IP的字段信息，然后节点后端可以通过程序或者相关的配置接收X-Forwarded-For传过来的用户真实IP的信息。

解决上述问题的整个Nginx代理配置为：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;
```

```
events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    upstream www_server_pools {
        server 10.0.0.9:


80  weight=1;



        server 10.0.0.10:


80  weight=1;



    }
    server {
        listen       80;



        server_name  www.etiantian.org;
```

```
        location / {
        proxy_pass http:

// www_server_pools;

        proxy_set_header Host  $host;

proxy_set_header X-Forwarded-For $remote_addr;

#<==在代理向后端服务器发送的

http请求头中加入

X-Forwarded-For字段信息，用于后端服务器程序、日志等接收记录真实用户的

IP，而不是代理服务器的

IP
        }
    }
}
```

重新加载Nginx反向代理服务：

```
[root@lb01 conf]# /application/nginx/sbin/nginx -s reload
```

特别注意，虽然反向代理这块已经配好了，但是节点服务器需要的访问日志如果要记录用户的真实IP，还必须进行日志格式配置，这样才能把代理传过来的X-Forwarded-For头信息记录下来，具体配置为：

```
 [root@web01 ~]# cat /application/nginx/conf/nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';



#<==就是这里的



"$http_x_forwarded_for"'参数，如果希望在第一行显示，可以替换掉第一行的



$remote_addr变量，这个日志格式前面章节已经详细讲解过这里就不多提了



    server {
        listen       80;
```

```
        server_name  bbs.etiantian.org;



        location / {
            root   html/bbs;



            index  index.html index.htm;



        }
        access_log  logs/access_bbs.log  main;



    }
    server {
        listen       80;



        server_name  www.etiantian.org;



        location / {
            root   html/www;



            index  index.html index.htm;



        }
        access_log  logs/access_www.log  main;



    }
}
```

注意：这里是客户端web01的配置。

完成web01、web02节点服务器的日志配置后，就可以检查了，注意，不要用curl从反向代理上检查，最好换一个客户端检查，这样才能看到效果。这里使用Windows客户端计算机（IP为10.0.0.2）访问已经解析好代理IP的www.etiantian.org ，如图11-9所示。

图11-9　访问网站效果

此时，再去节点服务器WWW服务的访问日志里查看，会发现日志的结尾已经变化了：

```
[root@web01 ~]# tail -2 /application/nginx/logs/access_www.log
10.0.0.7 - - [31/May/2015:

20:

47:

09 +0800] "GET / HTTP/1.0" 200 19 "-" "Mozilla/5.0 (

Windows NT 6.1;

 WOW64)

 AppleWebKit/537.36 (
```

```
KHTML,

 like Gecko)

 Chrome/42.0.2311.90 Safari/537.36" "10.0.0.2"
10.0.0.7 - - [31/May/2015:

20:

47:

10 +0800] "GET / HTTP/1.0" 200 19 "-" "Mozilla/5.0 (

Windows NT 6.1;

 WOW64)

 AppleWebKit/537.36 (

KHTML,

 like Gecko)

 Chrome/42.0.2311.90 Safari/537.36" "10.0.0.2"
```

其中，日志里的10.0.0.7为反向代理的IP，对应Nginx日志格式里的$remote_addr变量，而日志结尾的10.0.0.2对应的是日志格式里的"$http_x_forwarded_for"变量，即接收了前面反向代理配置

中“proxy_set_header X-Forwarded-For$remote_addr；”参数X-Forwarded-For的IP了。

关于X-Forwarded-For的详细说明，可见 http://en.wikipedia.org/wiki/X-Forwarded-For 。表11-7是Nginx反向代理相关重要基础参数的总结，供读者参考。

表11-7　相关重要参数说明

Nginx 反向代理重要参数	解释说明
proxy_pass http://server_pools；	通过 proxy_pass 功能把用户的请求转向到反向代理定义的 upstream 服务器池
proxy_set_header Host $host；	在代理向后端服务器发送的 http 请求头中加入 host 字段信息，用于当后端服务器配置有多个虚拟主机时，可以识别代理的是哪个虚拟主机。这是节点服务器多虚拟主机时的关键配置
proxy_set_header X-Forwarded-For $remote_addr；	在代理向后端服务器发送的 http 请求头中加入 X-Forwarded-For 字段信息，用于后端服务器程序、日志等接收记录真实用户的 IP，而不是代理服务器的 IP。 这是反向代理时，节点服务器获取用户真实 IP 的必要功能配置

4.与反向代理配置相关的更多参数说明

除了具有多虚拟主机代理以及节点服务器记录真实用户IP的功能外，Nginx软件还提供了相当多的作为反向代理和后端节点服务器对话的相关控制参数，具体见前面在讲解proxy模块时提供的表11-6。

看过表11-6中的说明后，相信读者已经对这些参数有了一定的了解，由于参数众多，最好把这些参数放到一个配置文件里，然后用include方式包含到虚拟主机配置里，效果如下：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    upstream www_server_pools {
        server 10.0.0.9:


80  weight=1;



        server 10.0.0.10:


80  weight=1;



    }
    server {
        listen       80;



        server_name  www.etiantian.org;
```

```
        location / {
        proxy_pass http:


// www_server_pools;


        include proxy.conf;


     #<==这就是包含的配置，具体配置内容见下文


        }
    }
}
[root@lb01 conf]# cat proxy.conf     #     <==把参数写成一个文件，使用


include包含，看起来更简洁规范


        proxy_set_header Host  $host;


        proxy_set_header X-Forwarded-For $remote_addr;


        proxy_connect_timeout 60;


        proxy_send_timeout 60;


        proxy_read_timeout 60;


        proxy_buffer_size 4k;
```

```
        proxy_buffers 4 32k;

        proxy_busy_buffers_size 64k;

        proxy_temp_file_write_size 64k;
```

更多Nginx反向代理参数说明：http://nginx.org/en/docs/http/ngx_http_proxy_module.html 。

11.8.3　根据URL中的目录地址实现代理转发

1.根据URL中的目录地址实现代理转发的说明

为了让大家能学以致用，还是通过真实的企业的案例模拟来给大家讲解这部分的知识。

案例背景：通过Nginx实现动静分离，即通过Nginx反向代理配置规则实现让动态资源和静态资源及其他业务分别由不同的服务器解析，以解决网站性能、安全、用户体验等重要问题。

图11-10为企业常见的动静分离集群架构图，此架构图适合网站前端只使用同一个域名提供服务的场景，例如，用户访问的域名是www.etiantian.org，然后，当用户请求www.etiantian.org/upload/xx 地址的时候，代理会分配请求到上传服务器池处理数据；当用户请求www.etiantian.org/static/xx 地址的时候，代理会分配请求到静态服务器池请求数据；当用户请求www.etiantian.org/xx 地址的时候，即不包含上述指定的目录地址路径时，代理会分配请求到默认的动态服务器池请求数据（注意：上面的xx表示任意路径）。

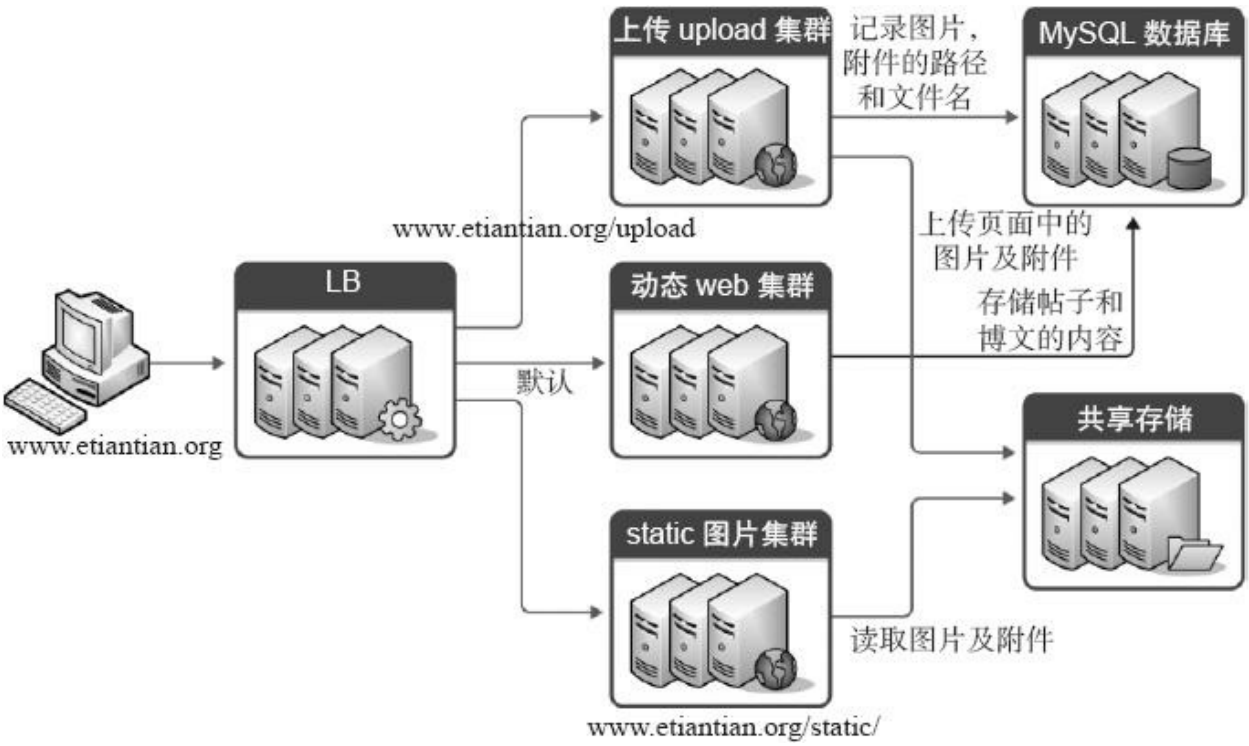


图11-10　动静分离网站集群架构

2.准备：案例配置实战

先进行企业案例需求梳理：

·当用户请求www.etiantian.org/upload/xx 地址时，实现由upload上传服务器池处理请求。

·当用户请求www.etiantian.org/static/xx 地址时，实现由静态服务器池处理请求。

·除此以外，对于其他访问请求，全都由默认的动态服务器池处理请求。

了解了需求后，就可以进行upstream模块服务器池的配置了。static_pools为静态服务器池，有一个服务器，地址为10.0.0.9，端口为80。

```
upstream static_pools {
        server 10.0.0.9:


80  weight=1;



    }
```

upload_pools为上传服务器池，有一个服务器，地址为10.0.0.10，端口为80。

```
upstream upload_pools {
        server 10.0.0.10:


80  weight=1;



    }
```

default_pools为默认的服务器池，即动态服务器池，有一个服务器，地址为10.0.0.11，端口为80。

```
upstream default_pools {
        server 10.0.0.11:


80  weight=1;
```

```
    }
```

提示：需要增加一台测试Web节点web03（ip是10.0.0.11），其配置与web01、web02一样。

下面利用location或if语句把不同的URI（路径）请求，分给不同的服务器池处理，具体配置如下。

方案1：以location方案实现。

将符合static的请求交给静态服务器池static_pools，配置如下：

```
location /static/ {
    proxy_pass http:


// static_pools;


    include proxy.conf;


}
```

将符合upload的请求交给上传服务器池upload_pools，配置如下：

```
location /upload/ {
    proxy_pass http:
```

```
// upload_pools;

    include proxy.conf;

}
```

不符合上述规则的请求，默认全部交给动态服务器池default_pools，配置如下：

```
location / {
    proxy_pass http:

// default_pools;

    include proxy.conf;

}
```

方案2：以if语句实现。

```
if  (

$request_uri   ~*   "^/static/ (

.*)

$")
```

```
{
        proxy_pass http:


// static_pools/$1;



}
if  (


$request_uri   ~*   "^/upload/ (


.*)


$")



{
        proxy_pass http:


// upload_pools/$1;



}
location / {
    proxy_pass http:


// default_pools;



    include proxy.conf;



}
```

下面以方案1为例进行讲解，Nginx反向代理的实际配置如下：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



        upstream static_pools {
                server 10.0.0.9:



80  weight=1;



        }
        upstream upload_pools {
                server 10.0.0.10:



80  weight=1;



        }
```

```
        upstream default_pools {
                server 10.0.0.11:

80  weight=1;

        }
    server {
        listen       80;

        server_name  www.etiantian.org;

        location / {
                proxy_pass http:

// default_pools;

                include proxy.conf;

        }
        location /static/ {
                proxy_pass http:

// static_pools;

                include proxy.conf;

        }
        location /upload/ {
                proxy_pass http:

// upload_pools;
```

```
                include proxy.conf;


        }
    }
}
```

重新加载配置生效，如下：

```
[root@lb01 conf]# /application/nginx/sbin/nginx -t
nginx:


 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:


 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@lb01 conf]# /application/nginx/sbin/nginx -s reload
```

暂时不要立刻测试成果，为了实现上述代理的测试，还需要在web01和web02上做节点的测试配置，才能更好地展示测试效果。

以web01作为static静态服务，地址端口为：10.0.0.9：80，需事先配置一个用于测试静态的地址页面，并测试访问，确定它会返回正确结果。操作步骤如下：

```
[root@web01 ~]# cd /application/nginx/html/www/
[root@web01 www]# mkdir static
[root@web01 www]# echo static_pools >static/index.html
[root@web01 www]# curl http:


// www.etiantian.org/static/index.html <==这里的
```

```
www.etiantian.org是解析过

web01本地

IP的。

static_pools
```

提示：测试的静态地址为http://www.etiantian.org/static/index.html，注意，是带static路径的地址。

以web02作为upload上传服务，地址端口为10.0.0.10：80，需事先配置一个用于测试上传服务的地址页面，并测试访问，确定它会返回正确结果。操作步骤如下：

```
[root@web02 ~]# cd /application/nginx/html/www/
[root@web02 www]# mkdir upload
[root@web02 www]# echo upload_pools >upload/index.html
[root@web02 www]# curl http:

// www.etiantian.org/upload/index.html  #<==这里的

www.etiantian.org是解析过

web01本地

IP的。
```

```
upload_pools
```

提示：测试的上传地址为http://www.etiantian.org/upload/index.html，注意，是带upload路径的地址。

以web03作为动态服务节点，地址端口为10.0.0.11：80，同样需事先配置一个默认的地址页面，并测试访问，确定它会返回正确结果。操作步骤如下：

```
[root@web03 ~]# cd /application/nginx/html/www/
[root@web03 www]# echo default_pools >index.html
[root@web03 www]# curl http:


// www.etiantian.org  #<==这里的


www.etiantian.org是解析过


web01本地


IP的。


default_pools
```

以上准备了三台Web节点服务器，分别加入到了upstream定义的不同服务器池，代表三组不同的业务集群组，从本机通过hosts解析各自的

域名，然后测试访问，其地址与实际访问的内容输出请对照表11-8。

表11-8　测试节点信息及访问效果

节点	IP 及端口	测试地址	字符串为代表业务
web01	10.0.0.9:80	http://www.etiantian.org/static/index.html	static_pools
web02	10.0.0.10:80	http://www.etiantian.org/upload/index.html	upload_pools
web03	10.0.0.11:80	http://www.etiantian.org/index.html http://www.etiantian.org	default_pools

使用客户端计算机访问测试时，最好选用集群以外的机器，这里先在浏览器客户端的hosts文件里把www.etiantian.org 解析到Nginx反向代理服务器的IP，然后访问上述URL，看代理是不是把请求正确地转发到了指定的服务器上。如果可以得到与表11-8对应的内容，表示配置的Nginx代理分发的完全正确（如图11-11所示），因为如果分发请求到错误的机器上就没有对应的URL页面内容，输出会是404错误。

图11-11　实现动静分离的测访问试效果

3.根据URL目录地址转发的应用场景

根据HTTP的URL进行转发的应用情况，被称为第7层（应用层）的负载均衡，而LVS的负载均衡一般用于TCP等的转发，因此被称为第4层（传输层）的负载均衡。

在企业中，有时希望只用一个域名对外提供服务，不希望使用多个域名对应同一个产品业务，此时就需要在代理服务器上通过配置规则，使得匹配不同规则的请求会交给不同的服务器池处理。这类业务有：

·业务的域名没有拆分或者不希望拆分，但希望实现动静分离、多业务分离，这在前面已经讲解过案例了。

·不同的客户端设备（例如：手机和PC端）使用同一个域名访问同一个业务网站，就需要根据规则将不同设备的用户请求交给后端不同的服务器处理，以便得到最佳用户体验。这也是非常重要的，接下来就讲解相关案例。

11.8.4　根据客户端的设备（user_agent）转发实践

1.根据客户端的设备（user_agent）转发实践需求

企业中，为了让不同的客户端设备用户访问有更好的体验，需要在后端架设不同服务器来满足不同的客户端访问，例如：移动客户端访问网站，就需要部署单独的移动服务器及程序，体验才能更好，而且移动端还分苹果、安卓、Ipad等，在传统的情况下，一般用下面的办法解决这个问题。

（1）常规4层负载均衡解决方案架构

在常规4层负载均衡架构下，可以使用不同的域名来实现这个需求，例如，人为分配好让移动端用户访问wap.etiantian.org，PC客户端用户访问www.etiantian.org，通过不同域名来引导用户到指定的后端服务器，该解决方案的架构如图11-12所示。

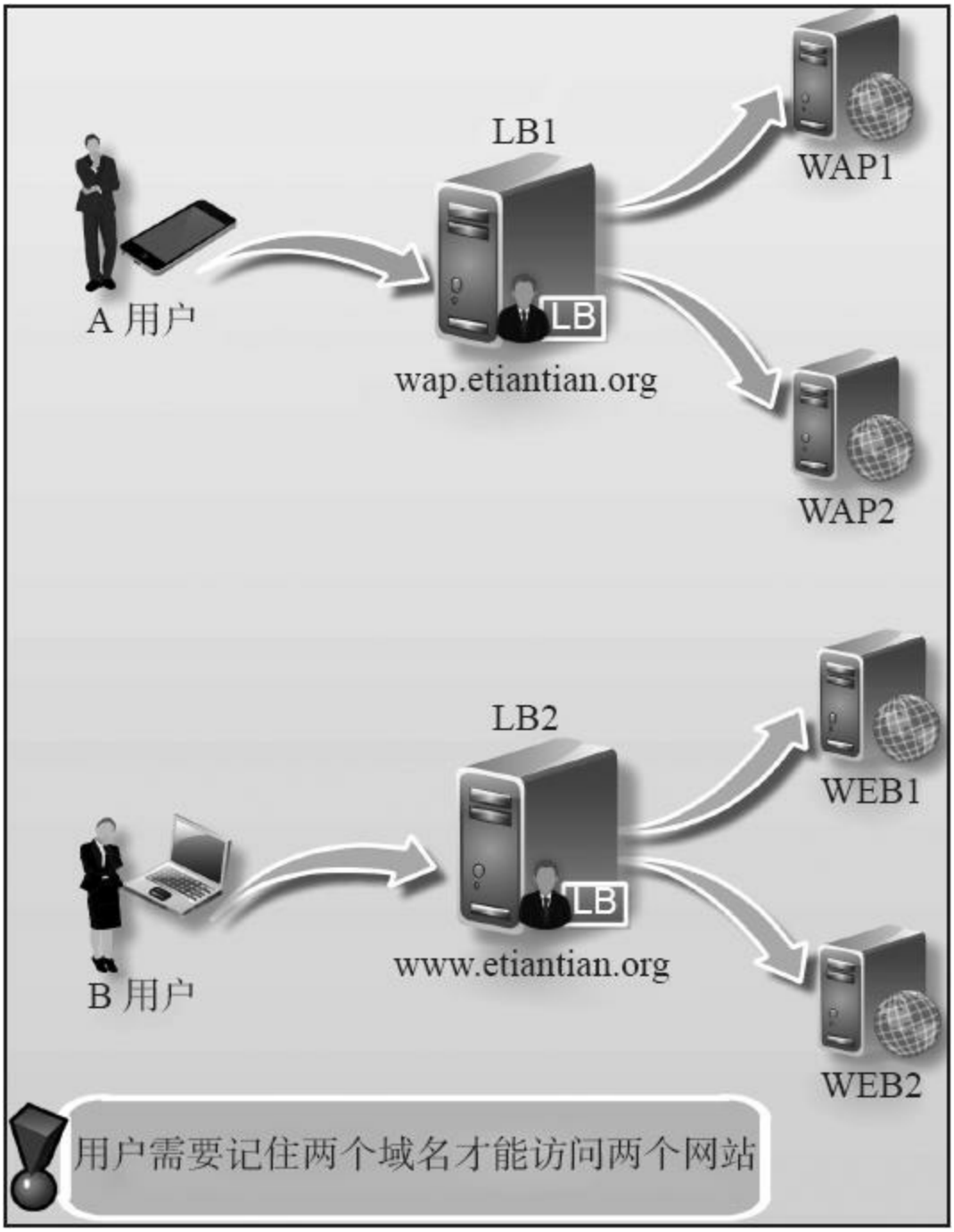


图11-12　通过域名来引导用户架构示意图

此解决方案的最大问题就是不同客户端的用户要记住对应的域名！而绝大多数用户只会记住www.etiantian.org，不会记住wap.etiantian.org

，这样一来就会导致用户体验不是很好。有没有办法让所有客户端用户只访问一个统一的www.etiantian.org 这个地址，还能让不同客户端设备都能有更好的访问体验呢？当然有！那就是下面的第7层负载均衡解决方案。

（2）第7层负载均衡解决方案

在第7层负载均衡架构下，就可以不需要人为拆分域名了，对外只需要用一个域名，例如www.etiantian.org ，然后通过获取用户请求中的设备信息（利用$http_user_agent获取），根据这些信息转给后端合适的服务器处理，这个方案最大好处就是不需要让用户记忆多个域名了，用户只需要记住主网站地址www.etiantian.org ，剩下的由网站服务器处理，这样的思路大大地提升了用户访问体验，这是当前企业网站非常常用的解决方案。

下面我们就来讲解此方案，图11-13描述了上述解决方案相应的架构逻辑图。

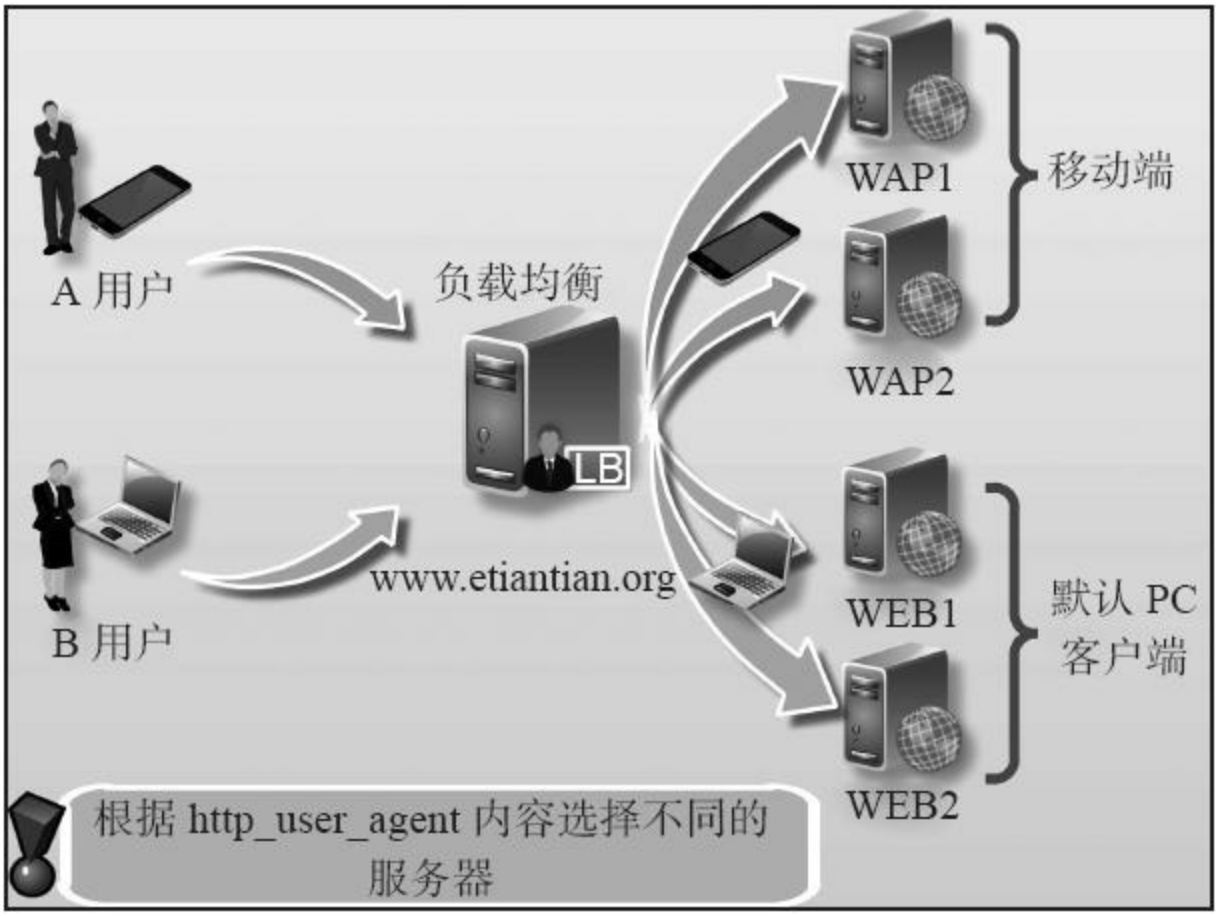


图11-13　通过识别user_agent分发请求架构示意图

2.根据客户端设备（user_agent）转发请求实践

这里还是使用static_pools、upload_pools作为本次实验的后端服务器池。下面先根据计算机客户端浏览器的不同设置对应的匹配规则。

```
        location / {
            if （

    $http_user_agent ~* "MSIE"）

    #如果请求的浏览器为微软
```

IE浏览器（

MSIE），则让请求由

static_pools池处理

```
          {
            proxy_pass http:

// static_pools;

          }
          if （

$http_user_agent ~* "Chrome"）

#如果请求的浏览器为谷歌浏览器（

Chrome），则让请求由

upload_pools池处理

          {
            proxy_pass http:

// upload_pools;

          }
        proxy_pass http:
```

```
// default_pools;

#其他客户端，由

default_pools处理

        include proxy.conf;

    }
```

实践中完整的配置文件内容如下：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;

events {
    worker_connections  1024;

}
http {
    include       mime.types;

    default_type  application/octet-stream;

    sendfile        on;

    keepalive_timeout  65;
```

```
        upstream static_pools {
                server 10.0.0.9:

80  weight=1;

        }
        upstream upload_pools {
                server 10.0.0.10:

80  weight=1;

        }
        upstream default_pools {
                server 10.0.0.11:

80  weight=1;

        }
    server {
        listen       80;

        server_name  www.etiantian.org;

        location / {
          if  (

$http_user_agent ~* "MSIE")

          {
            proxy_pass http:
```

```
// static_pools;



          }
          if  (


$http_user_agent ~* "Chrome")



#if  (


$http_user_agent ~* "Firefox")



          {
            proxy_pass http:


// upload_pools;



            }
        proxy_pass http:


// default_pools;



        include proxy.conf;



        }
        access_log off;



    }
}
```

下面来看看最终的测试效果。

使用微软IE浏览器访问的效果如图11-14所示。

图11-14　微软IE浏览器访问效果

访问http://www.etiantian.org/static/ 返回正常，可以确认转发到了后端的static_pools池。而访问http://www.etiantian.org/upload/ 则返回404，因为没有匹配到任何规则，就会交给默认的default_pools池服务器处理了，此时default_pools服务器池没有upload这个目录及内容页，所以，就返回404错误了。

使用谷歌浏览器访问效果如图11-15所示。

图11-15　谷歌浏览器访问效果图

访问http://www.etiantian.org/upload/ 返回正常，可以确认转发到了

后端upload_pools池了。而访问http://www.etiantian.org/static/ 时返回了404，因为没有匹配到任何规则，就会交给默认的default_pools池服务器处理，此default_pools服务器池没有static这个目录及内容页，所以就返回404错误了。

需要特别说明的是，当你无法知道客户端设备字符串名字（例如Chrome、MSIE、Firefox这样的字符串）时，可以先用这些浏览器访问对应的节点服务器，然后根据访问日志中的$http_user_agent格式记录的日志确认，例如：

```
[root@web01 conf]# tail -2 ../logs/access_www.log
10.0.0.102 - - [13/Jul/2015:

18:

12:

08 +0800] "GET /upload/index.html HTTP/1.1" 404 168 "-" "Mozilla/5.0 (

Windows NT 6.1;

 WOW64;

 rv:

29.0)

 Gecko/20100101 Firefox/29.0" "-"
```

```
10.0.0.102 - - [13/Jul/2015:
18:
13:
33 +0800] "GET /favicon.ico HTTP/1.1" 404 570 "http:
// www.etiantian.org/upload/" "Mozilla/5.0 (
Windows NT 6.1;
 WOW64)
 AppleWebKit/537.36 (
KHTML,
 like Gecko)
 Chrome/42.0.2311.90 Safari/537.36" "-"
```

除了针对浏览器外，上述“$http_user_agent”变量也可针对移动端，比如安卓、苹果、Ipad设备进行匹配，去请求指定的服务器，具体配置如下：

```
location / {
 if (
$http_user_agent ~* "android")
```

```
    {
    proxy_pass http:

// android_pools;

 #<==这是

android服务器池，需要提前定义

upstream。

    }
 if （

$http_user_agent ~* "iphone"）

    {
    proxy_pass http:

// iphone_pools;

 #<==这是

iphone服务器池，需要提前定义

upstream。

    }
 proxy_pass http:
```

```
// pc_pools;

 include extra/proxy.conf;

}
```

这部分测试可以通过局域网的WiFi功能来实现，用手机等连接到WiFi，然后访问服务器的IP测试就可以了。测试时，请用节点的第一个虚拟主机请求测试，这样就不需要本地hosts域名解析了，因为手机端测试做hosts解析也不容易，当然有公网的域名和服务器测试最佳，这部分的配合和测试与浏览器设备实践几乎一样，因此，这里的测试就留给读者了，看看能不能达到你想的测试效果？

此外，查找移动设备的user_agent对应的具体名称时，还是先用对应的设备通过IP地址访问节点服务器，然后看访问日志，注意IP访问只找第一个虚拟主机的网站。

```
[root@web01 conf]# tail -2 ../logs/access_bbs.log
10.0.0.103 - - [13/Jul/2015:

18:

28:

45 +0800] "GET /favicon.ico HTTP/1.1" 404 570 "-" "Mozilla/5.0 (
```

```
Linux;

 Android 5.0.1;

 SAMSUNG-SM-N9109W Build/LRX22C)

 AppleWebKit/537.36 (

KHTML,

 like Gecko)

 SamsungBrowser/2.1 Chrome/34.0.1847.76 Mobile Safari/537.36" "-"     #<==SANSUNG NO

10.0.0.104 - - [13/Jul/2015:

18:

30:

31 +0800] "GET / HTTP/1.1" 200 19 "-" "Mozilla/5.0 (

iPhone;

 CPU iPhone OS 8_4 like Mac OS X)

 AppleWebKit/600.1.4 (

KHTML,
```

like Gecko)

Mobile/12H143 QQ/5.6.0.438 NetType/WIFI Mem/35" "-"
    #<==苹果

iPhone6手机设备访问的日志。

11.8.5　根据文件扩展名实现代理转发

除了根据URI路径及user_agent转发外，还可以实现根据文件扩展名进行转发。

1.相关server配置

先看看location方法的匹配规则，如下：

```
location ~ .*.(
gif|jpg|jpeg|png|bmp|swf|css|js)
$ {
    proxy_pass http:
// static_pools;
    include proxy.conf;
}
```

下面是if语句方法的匹配规则：

```
if (
```

```
    $request_uri   ~*   ".*\.(
    php|php5)
    $")

    {
        proxy_pass http:
    // php_server_pools;

    }
    if  (
    $request_uri   ~*   ".*\.(
    jsp|jsp*|do|do*)
    $")

    {
        proxy_pass http:
    // java_server_pools;

    }
```

此部分实践方法和前面的根据URI路径以及user_agent转发是相同的，因此，这里不再赘述了，读者可以当作作业自行练习完成。

2.根据扩展名转发的应用场景

可根据扩展名实现资源动静分离访问，如图片、视频等请求静态服务器池，PHP、JSP等请求动态服务器池。示例代码如下：

```
location ~ .*. (

gif|jpg|jpeg|png|bmp|swf|css|js)

$ {
    proxy_pass http:

// static_pools;

    include proxy.conf;

}
location ~ .*. (

php|php3|php5)

$ {
    proxy_pass http:

// dynamic_pools;

    include proxy.conf;

}
```

在开发无法通过程序实现动静分离的时候，运维可以根据资源实体进行动静分离，而不依赖于开发，具体实现策略是先把后端的服务器分成不同的组。注意，每组服务器的程序都是相同的，因为开发没有把程序拆开，分组后，在前端代理服务器上通过讲解过的路径、扩展名进行规则匹配，从而实现请求的动静分离。

11.9 Nginx负载均衡监测节点状态

淘宝技术团队开发了一个Tengine（Nginx的分支）模块nginx_upstream_check_module，用于提供主动式后端服务器健康检查。通过它可以检测后端realserver的健康状态，如果后端realserver不可用，则所有的请求就不会转发到该节点上。

Tengine原生支持这个模块，而Nginx则需要通过打补丁的方式将该模块添加到Nginx中。补丁下载地址：https://github.com/yaoweibin/nginx_upstream_check_module 。下面介绍如何使用这个模块。

1）安装nginx_upstream_check_module模块。

```
# 系统已经安装了


nginx-1.6.3软件



[root@lb01 ~]# /application/nginx/sbin/nginx -V
nginx version:



 nginx/1.6.3
[root@lb01 ~]# cd /home/oldboy/tools/
# 下载补丁包



[root@lb01 tools]# wget https:
```

```
// codeload.github.com/yaoweibin/nginx_upstream_check_module/zip/master
[root@lb01 tools]# unzip master
# 因为是对源程序打补丁，所以还需要
```

Nginx软件包

```
[root@lb01 tools]# cd nginx-1.6.3
[root@lb01 nginx-1.6.3]# patch -p1 < ../nginx_upstream_check_module-master/check_1.5
# 编译参数要和以前一致，最后加上
```

```
--add-module=../nginx_upstream_check_module-master/
[root@lb01 nginx-1.6.3]# ./configure --prefix=/application/nginx-1.6.3 --user=nginx
[root@lb01 nginx-1.6.3]# make
# 如果是新装的
```

Nginx则继续执行下面

make install一步，如果给已经安装的

nginx系统打监控补丁就不用执行

make install了，本文忽略执行

make install。

make的作用就是重新生成

Nginx二进制启动命令而已。

操作前备份

```
[root@lb01 nginx-1.6.3]# mv /application/nginx/sbin/nginx{,
.ori}
# 将打过补丁的
Nginx二进制程序复制到
/application/nginx/sbin/目录下

[root@lb01 nginx-1.6.3]# cp ./objs/nginx /application/nginx/sbin/
# 检查是否正常

[root@lb01 nginx-1.6.3]# /application/nginx/sbin/nginx -t
nginx:
 the configuration file /application/nginx-1.6.3/conf/nginx.conf syntax is ok
nginx:
 configuration file /application/nginx-1.6.3/conf/nginx.conf test is successful
[root@lb01 nginx-1.6.3]# /application/nginx/sbin/nginx -V
nginx version:
 nginx/1.6.3
built by gcc 4.4.7 20120313 (
Red Hat 4.4.7-11)
  (
GCC)

TLS SNI support enabled
configure arguments:
```

```
 --prefix=/application/nginx-1.6.3 --user=nginx --group=nginx --with-http_ssl_module
```

2）配置Nginx健康检查，如下：

```
[root@lb01 nginx-1.6.3]# vi /application/nginx/conf/nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



upstream static_pools {
    server 10.0.0.9:


80 weight=1;



    server 10.0.0.10:


80 weight=1;
```

```
    check interval=3000 rise=2 fall=5 timeout=1000 type=http;



}
upstream default_pools {
    server 10.0.0.11:


80 weight=1;



}
    server {
        listen       80;



        server_name   www.etiantian.org;



        location / {
            root   html;



            index  index.html index.htm;



            proxy_pass  http:


// default_pools;



            include proxy.conf;



        }
        location ~ .*. (
```

```
gif|jpg|jpeg|png|bmp|swf|css|js)


$ {
            proxy_pass  http:


// static_pools;



            include proxy.conf;



         }
        location /status {
            check_status;



            access_log off;



        }
    }
}
[root@lb01 nginx-1.6.3]# /application/nginx/sbin/nginx -s stop
[root@lb01 nginx-1.6.3]# /application/nginx/sbin/nginx
#注意此处必须重启


Nginx，不能重新加载。



# check interval=3000 rise=2 fall=5 timeout=1000 type=http;
```

上面配置的意思是，对static_pools这个负载均衡条目中的所有节点，每隔3秒检测一次，请求2次正常则标记realserver状态为up，如果检测5次都失败，则标记realserver的状态为down，超时时间为1秒，检查的

协议是HTTP。

详细用法见官网http://tengine.taobao.org/document_cn/http_upstream_check_cn.html 。

访问http://10.0.0.7/status 页面时，显示如图11-16所示。

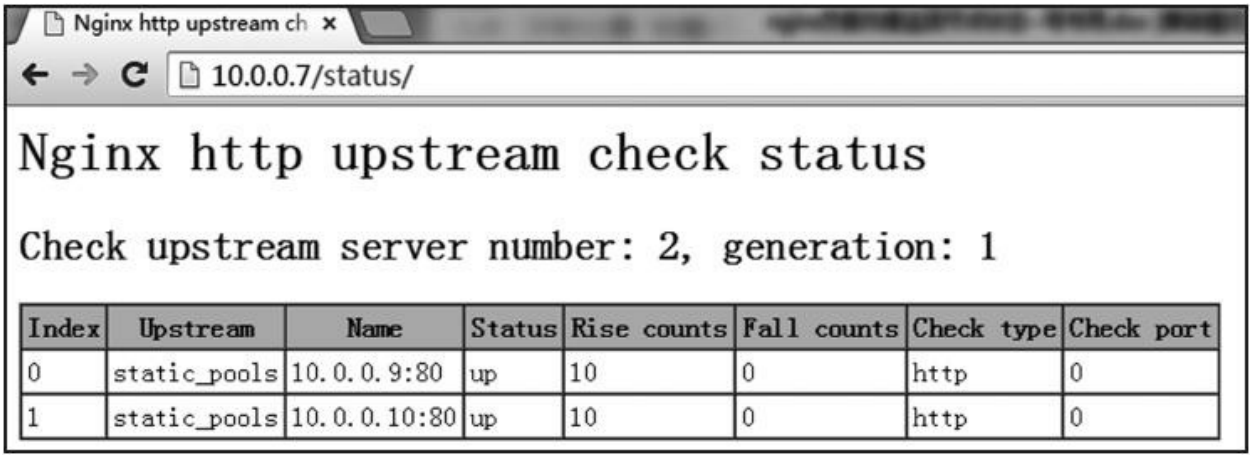


图11-16　正常状态下Nginx节点健康检查状态

测试节点故障的命令如下：

```
[root@web01 ~]# /application/nginx/sbin/nginx -s stop
```

访问http://10.0.0.7/status 页面时，显示如图11-17所示。

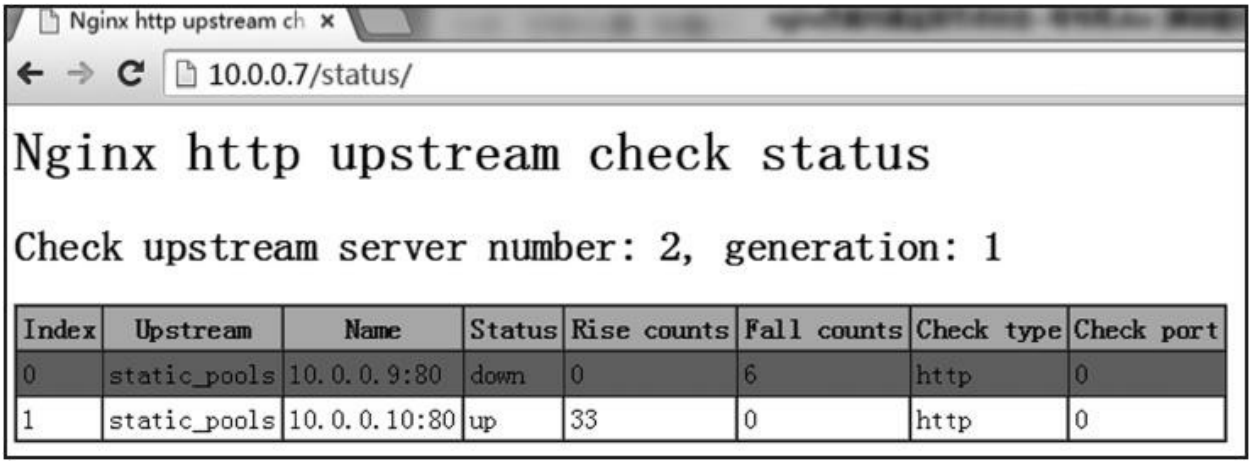


图11-17　非正常状态下Nginx节点健康检查状态

特别说明：除了此种插件的方法外，也可以自己开发Shell脚本监控达到上述效果，具体代码可见：http://oldboy.blog.51cto.com/2561410/1656844 。视频讲解见 http://edu.51cto.com/index.phpdo=lesson&id=70879 。

11.10　proxy_next_upstream参数补充

当Nginx接收后端服务器返回proxy_next_upstream参数定义的状态码时，会将这个请求转发给正常工作的后端服务器，例如500、502、503、504，此参数可以提升用户的访问体验，具体配置如下：

```
server {
    listen       80;


    server_name  www.etiantian.org;


    location / {
        proxy_pass http:


// static_pools;


        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503


        include proxy.conf;


    }
```

11.11　本章重点回顾

1）集群的概念、分类、软硬件知识。

2）upstream负载均衡模块知识。

3）http_proxy代理模块知识及相关参数。

4）基于URI路径、user_agent、扩展名规则知识及实践案例。

5）监测Nginx代理下面节点的健康状态。

第12章　Keepalived高可用集群应用实践

12.1　Keepalived高可用软件

12.1.1　Keepalived介绍

Keepalived软件起初是专为LVS负载均衡软件设计的，用来管理并监控LVS集群系统中各个服务节点的状态，后来又加入了可以实现高可用的VRRP功能。因此，Keepalived除了能够管理LVS软件外，还可以作为其他服务（例如：Nginx、Haproxy、MySQL等）的高可用解决方案软件。

Keepalived软件主要是通过VRRP协议实现高可用功能的。VRRP是Virtual Router Redundancy Protocol（虚拟路由器冗余协议）的缩写，VRRP出现的目的就是为了解决静态路由单点故障问题的，它能够保证当个别节点宕机时，整个网络可以不间断地运行。所以，Keepalived一方面具有配置管理LVS的功能，同时还具有对LVS下面节点进行健康检查的功能，另一方面也可实现系统网络服务的高可用功能。

Keepalived软件的官方站点是http://www.keepalived.org 。

12.1.2　Keepalived服务的三个重要功能

1.管理LVS负载均衡软件

早期的LVS软件，需要通过命令行或脚本实现管理，并且没有针对LVS节点的健康检查功能。为了解决LVS的这些使用不便的问题，Keepalived就诞生了，可以说，Keepalived软件起初是专为解决LVS的问题而诞生的。因此，Keepalived和LVS的感情很深，它们的关系如同夫妻一样，可以紧密地结合，愉快地工作。Keepalived可以通过读取自身的配置文件，实现通过更底层的接口直接管理LVS的配置以及控制服务的启动、停止等功能，这使得LVS的应用更加简单方便了。LVS和Keepalived的组合应用不是本章的内容范围，此部分内容可以参考老男孩教育提供的Linux运维就业班视频或参考网上文章。

2.实现对LVS集群节点健康检查功能（healthcheck）

前文已讲过，Keepalived可以通过在自身的keepalived.conf文件里配置LVS的节点IP和相关参数实现对LVS的直接管理；除此之外，当LVS集群中的某一个甚至是几个节点服务器同时发生故障无法提供服务时，Keepalived服务会自动将失效的节点服务器从LVS的正常转发队列中清除出去，并将请求调度到别的正常节点服务器上，从而保证最终用户的访问不受影响；当故障的节点服务器被修复以后，Keepalived服务又会

自动地把它们加入到正常转发队列中，对客户提供服务。

3.作为系统网络服务的高可用功能（failover）

Keepalived可以实现任意两台主机之间，例如Master和Backup主机之间的故障转移和自动切换，这个主机可以是普通的不能停机的业务服务器，也可以是LVS负载均衡、Nginx反向代理这样的服务器。

Keepalived高可用功能实现的简单原理为，两台主机同时安装好Keepalived软件并启动服务，开始正常工作时，由角色为Master的主机获得所有资源并对用户提供服务，角色为Backup的主机作为Master主机的热备；当角色为Master的主机失效或出现故障时，角色为Backup的主机将自动接管Master主机的所有工作，包括接管VIP资源及相应资源服务；而当角色为Master的主机故障修复后，又会自动接管回它原来处理的工作，角色为Backup的主机则同时释放Master主机失效时它接管的工作，此时，两台主机将恢复到最初启动时各自的原始角色及工作状态。


说明：Keepalived的高可用功能是本章的重点，后面除了讲解Keepalived高可用的功能外，还会讲解Keepalived配合Nginx反向代理负载均衡的高可用的实战案例。

12.1.3　Keepalived高可用故障切换转移原理

Keepalived高可用服务对之间的故障切换转移，是通过VRRP（Virtual Router Redundancy Protocol，虚拟路由器冗余协议）来实现的。

在Keepalived服务正常工作时，主Master节点会不断地向备节点发送（多播的方式）心跳消息，用以告诉备Backup节点自己还活着，当主Master节点发生故障时，就无法发送心跳消息，备节点也就因此无法继续检测到来自主Master节点的心跳了，于是调用自身的接管程序，接管主Master节点的IP资源及服务。而当主Master节点恢复时，备Backup节点又会释放主节点故障时自身接管的IP资源及服务，恢复到原来的备用角色。

那么，什么是VRRP呢？

VRRP，全称Virtual Router Redundancy Protocol，中文名为虚拟路由冗余协议，VRRP的出现就是为了解决静态路由的单点故障问题，VRRP是通过一种竞选机制来将路由的任务交给某台VRRP路由器的。

VRRP早期是用来解决交换机、路由器等设备单点故障的，下面是交换、路由的Master和Backup切换原理描述，同样适用于Keepalived的

工作原理。

在一组VRRP路由器集群中，有多台物理VRRP路由器，但是这多台物理的机器并不是同时工作的，而是由一台称为Master的机器负责路由工作，其他的机器都是Backup。Master角色并非一成不变的，VRRP会让每个VRRP路由参与竞选，最终获胜的就是Master。获胜的Master有一些特权，比如拥有虚拟路由器的IP地址等，拥有系统资源的Master负责转发发送给网关地址的包和响应ARP请求。

VRRP通过竞选机制来实现虚拟路由器的功能，所有的协议报文都是通过IP多播（Multicast）包（默认的多播地址224.0.0.18）形式发送的。虚拟路由器由VRID（范围0-255）和一组IP地址组成，对外表现为一个周知的MAC地址：00-00-5E-00-01-{VRID}。所以，在一个虚拟路由器中，不管谁是Master，对外都是相同的MAC和IP（称之为VIP）。客户端主机并不需要因Master的改变而修改自己的路由配置。对它们来说，这种切换是透明的。

在一组虚拟路由器中，只有作为Master的VRRP路由器会一直发送VRRP广播包（VRRP Advertisement messages），此时Backup不会抢占Master。当Master不可用时，Backup就收不到来自Master的广播包了，此时多台Backup中优先级最高的路由器会抢占为Master。这种抢占是非常快速的（可能只有1秒甚至更少），以保证服务的连续性。出于安全性考虑，VRRP数据包使用了加密协议进行了加密。

如果你在面试时，要你解答Keepalived的工作原理，建议用自己的话回答如下内容，以下为对面试官的表述：

Keepalived高可用对之间是通过VRRP通信的，因此，我从VRRP开始给您讲起：

1）VRRP，全称Virtual Router Redundancy Protocol，中文名为虚拟路由冗余协议，VRRP的出现是为了解决静态路由的单点故障。

2）VRRP是通过一种竞选协议机制来将路由任务交给某台VRRP路由器的。

3）VRRP用IP多播的方式（默认多播地址（224.0.0.18））实现高可用对之间通信。

4）工作时主节点发包，备节点接包，当备节点接收不到主节点发的数据包的时候，就启动接管程序接管主节点的资源。备节点可以有多个，通过优先级竞选，但一般Keepalived系统运维工作中都是一对。

5）VRRP使用了加密协议加密数据，但Keepalived官方目前还是推荐用明文的方式配置认证类型和密码。

介绍完了VRRP，接下来我再介绍一下Keepalived服务的工作原理：

Keepalived高可用对之间是通过VRRP进行通信的，VRRP是通过竞

选机制来确定主备的，主的优先级高于备，因此，工作时主会优先获得所有的资源，备节点处于等待状态，当主挂了的时候，备节点就会接管主节点的资源，然后顶替主节点对外提供服务。

在Keepalived服务对之间，只有作为主的服务器会一直发送VRRP广播包，告诉备它还活着，此时备不会抢占主，当主不可用时，即备监听不到主发送的广播包时，就会启动相关服务接管资源，保证业务的连续性。接管速度最快可以小于1秒。

12.2 Keepalived高可用服务搭建准备

经过了前面对Keepalived[1]的介绍和原理讲解，相信读者已经初步了解了Keepalived这个高可用软件，下面开始实战之旅。

1.安装Keepalived环境说明

这里建议大家使用第11章介绍的Nginx负载均衡的系统环境来安装Keepalived服务，因为本章后面的实战案例是实现Nginx负载均衡的高可用案例。安装Keepalived的基础准备环境如下。

（1）硬件环境准备

准备4台物理服务器或4台VM虚拟机，两台用来做Keepalived服务，两台做测试的Web节点（如表12-1所示）。

表12-1 Keepalived高可用服务实验环境准备

HOSTNAME	IP	说 明
lb01	10.0.0.7	Keepalived 主服务器（Nginx 主负载均衡器）
lb02	10.0.0.8	Keepalived 备服务器（Nginx 备负载均衡器）
web01	10.0.0.9	web01 服务器（第 11 章搭建好的）
web02	10.0.0.10	web02 服务器（第 11 章搭建好的）

（2）CentOS系统及Nginx代理环境

下面是CentOS系统及Nginx代理环境：

```
[root@lb01 ~]# cat /etc/redhat-release
CentOS release 6.6 （


Final）



[root@lb01 ~]# uname -r
2.6.32-504.el6.x86_64
[root@lb01 ~]# uname -m
x86_64
[root@lb01 ~]# ls -l /application/nginx/总用量


 44
drwx------ 2 nginx root 4096 5月


  30 11：


40 client_body_temp
drwxr-xr-x 2 root  root 4096 7月


  14 14：


31 conf
drwx------ 2 nginx root 4096 5月


  30 11：


40 fastcgi_temp
drwxr-xr-x 4 root  root 4096 5月


  30 11：


45 html
drwxr-xr-x 2 root  root 4096 7月


  14 14：
```

```
32 logs
drwx------ 2 nginx root 4096 5月


  30 11:


40 proxy_temp
drwxr-xr-x 2 root  root 4096 7月


  14 14:


25 sbin
drwx------ 2 nginx root 4096 5月


  30 11:


40 scgi_temp
drwx------ 2 nginx root 4096 5月


  30 11:


40 uwsgi_temp
```

2.开始安装Keepalived软件

 说明：下面有关Keepalived安装、启动服务的操作都是同时处理lb01、lb02两台机器。

可以通过官方地址获取Keepalived源码软件包编译安装，也可以使用yum的安装方式直接安装，这里选择更为简便的后者——yum安装方

式，下面以lb01为例，介绍整个安装步骤，如下：

```
[root@lb01 ~]# yum install keepalived -y
[root@lb01 ~]# rpm -qa keepalived
keepalived-1.2.13-5.el6_6.x86_64
```

 提示：

1）上述安装过程需要在lb01和lb02两台服务器上同时安装。

2）Keepalived版本为2.13版。

3.启动Keepalived服务并检查

启动及检查Keepalived服务的命令如下：

```
[root@lb01 ~]# /etc/init.d/keepalived start正在启动


 keepalived:


                                          [确定


]
[root@lb01 ~]# ps -ef|grep keep|grep -v grep
root      7263     1  0 17:


40         00:


00:


00 /usr/sbin/keepalived -D
```

```
root      7265  7263  0 17:

40         00:

00:

00 /usr/sbin/keepalived -D
root      7266  7263  0 17:

40         00:

00:

00 /usr/sbin/keepalived -D
# 提示：启动后有

3个

Keepalived进程表示安装正确

[root@lb01 ~]# ip add|grep 192.168
    inet 192.168.200.16/32 scope global eth0
    inet 192.168.200.17/32 scope global eth0
    inet 192.168.200.18/32 scope global eth0
# 提示：默认情况会启动三个

VIP地址

[root@lb01 ~]# /etc/init.d/keepalived stop停止

 keepalived:

                                                [确定
```

```
]
# 提示：测试完毕后关闭服务，上述测试需同时在

lb01和

lb02两台服务器上进行
```

4.Keepalived配置文件说明

和其他使用yum安装的软件一样，Keepalived软件的配置文件默认路径及配置文件名为：

```
[root@lb01 ~]# ls -l /etc/keepalived/keepalived.conf
-rw-r--r-- 1 root root 3562 3月

  19 18:

21 /etc/keepalived/keepalived.conf
```

前文已经说过，Keepalived软件有3个主要功能，而本章仅讲解其高可用部分的功能，因此，下面的讲解将会默认去掉非高可用功能的相关参数，有关Keepalived服务其他功能参数的讲解，请参看老男孩教育的LVS视频或者网上文章。

这里的具备高可用功能的keepalived.conf配置文件包含了两个重要

区块，下面会分别说明。

（1）全局定义（Global Definitions）部分

这部分主要用来设置Keepalived的故障通知机制和Router ID标识。示例代码如下：

```
[root@lb01 ~]# head -13 /etc/keepalived/keepalived.conf|cat -n
     1  !


     Configuration File for keepalived
     2
     3  global_defs {
     4     notification_email {
     5       acassen@firewall.loc
     6       failover@firewall.loc
     7       sysadmin@firewall.loc
     8     }
     9     notification_email_from Alexandre.Cassen@firewall.loc
    10     smtp_server 192.168.200.1
    11     smtp_connect_timeout 30
    12     router_id LVS_DEVEL
    13  }
```

基础参数说明：

第1行是注释，！开头和#号开头一样，都是注释。

第2行是空行。

第3~8行是定义服务故障报警的Email地址。作用是当服务发生切换或RS节点等有故障时，发报警邮件。这几行是可选配置，notification_email指定在keepalived发生事件时，需要发送的Email地址，可以有多个，每行一个。

第9行是指定发送邮件的发送人，即发件人地址，也是可选的配置。

第10行smtp_server指定发送邮件的smtp服务器，如果本机开启了sendmail或postfix，就可以使用上面默认配置实现邮件发送，也是可选配置。

第11行smtp_connect_timeout是连接smtp的超时时间，也是可选配置。


注意：第4~11行所有和邮件报警相关的参数均可以不配，在实际工作中会将监控的任务交给更加擅长监控报警的Nagios或Zabbix软件。

第12行是Keepalived服务器的路由标识（router_id）。在一个局域网内，这个标识（router_id）应该是唯一的。

大括号“{}”。用来分隔区块，要成对出现。如果漏写了半个大括号，Keepalived运行时，不会报错，但也不会得到预期的结果。另外，由于区块间存在多层嵌套关系，因此很容易遗漏区块结尾处的大括号，要特别注意。

更多参数信息请执行man keepalived.conf获得。

（2）VRRP实例定义区块（VRRP instance（s））部分

这部分主要用来定义具体服务的实例配置，包括Keepalived主备状态、接口、优先级、认证方式和IP信息等。示例代码如下：

```
15  vrrp_instance VI_1 {
16      state MASTER
17      interface eth0
18      virtual_router_id 51
19      priority 100
20      advert_int 1
21      authentication {
22          auth_type PASS
23          auth_pass 1111
24      }
25      virtual_ipaddress {
26          192.168.200.16
27          192.168.200.17
28          192.168.200.18
29      }
30  }
```

参数说明：

第15行表示定义一个vrrp_instance实例，名字是VI_1，每个vrrp_instance实例可以认为是Keepalived服务的一个实例或者作为一个业务服务，在Keepalived服务配置中，这样的vrrp_instance实例可以有多个。注意，存在于主节点中的vrrp_instance实例在备节点中也要存在，这样才能实现故障切换接管。

第16行state MASTER表示当前实例VI_1的角色状态，当前角色为MASTER，这个状态只能有MASTER和BACKUP两种状态，并且需要大写这些字符。其中MASTER为正式工作的状态，BACKUP为备用的状

态。当MASTER所在的服务器故障或失效时，BACKUP所在的服务器会接管故障的MASTER继续提供服务。

第17行interface为网络通信接口。为对外提供服务的网络接口，如eth0、eth1。当前主流的服务器都有2~4个网络接口，在选择服务接口时，要搞清楚了。

第18行virtual_router_id为虚拟路由ID标识，这个标识最好是一个数字，并且要在一个keepalived.conf配置中是唯一的。但是MASTER和BACKUP配置中相同实例的virtual_router_id又必须是一致的，否则将出现脑裂问题。

第19行priority为优先级，其后面的数值也是一个数字，数字越大，表示实例优先级越高。在同一个vrrp_instance实例里，MASTER的优先级配置要高于BACKUP的。若MASTER的priority值为150，那么BACKUP的priority必须小于150，一般建议间隔50以上为佳，例如：设置BACKUP的priority为100或更小的数值。

第20行advert_int为同步通知间隔。MASTER与BACKUP之间通信检查的时间间隔，单位为秒，默认为1。

第21~24行authentication为权限认证配置。包含认证类型（auth_type）和认证密码（auth_pass）。认证类型有PASS（Simple Passwd（suggested））、AH（IPSEC（not recommended））两种，官

方推荐使用的类型为PASS。验证密码为明文方式，最好长度不要超过8个字符，建议用4位的数字，同一vrrp实例的MASTER与BACKUP使用相同的密码才能正常通信。

第25~29行virtual_ipaddress为虚拟IP地址。可以配置多个IP地址，每个地址占一行，配置时最好明确指定子网掩码以及虚拟IP绑定的网络接口。否则，子网掩码默认是32位，绑定的接口和前面的interface参数配置的一致。注意，这里的虚拟IP就是在工作中需要和域名绑定的IP，即和配置的高可用服务监听的IP要保持一致！

[1] Keepalived软件的官方文档地址 http://www.keepalived.org/documentation.html。

12.3 Keepalived高可用服务单实例实战

12.3.1 配置Keepalived实现单实例单IP自动漂移接管

事实上，网络服务的高可用功能基本原理都很简单，就是把手动的操作自动化运行而已。当没有配置高可用服务时，如果服务器宕机了怎么解决呢？无非就是找一个新服务器，配好域名解析的那个原IP，然后搭好相应的网络服务罢了，只不过手工去实现这个过程会比较漫长，相比而言，自动化切换效率更高，效果更好，而且还可以有更多的功能，例如：发送ARP广播，触发执行相关脚本动作等。

实际上也可以将高可用对的两台机器应用服务同时开启，但是只让有VIP一端的服务器提供服务，若主的服务器宕机，VIP会自动漂移到备用服务器上，此时用户的请求直接发送到备用服务器上，而无需临时启动对应服务（事先开启应用服务）。下面来讲解VIP自动漂移的实战案例。

1.实战配置Keepalived主服务器lb01 MASTER

首先，配置lb01 MASTER的keepalived.conf配置文件，操作步骤如

下：

```
[root@lb01 ~]# cd /etc/keepalived/
[root@lb01 keepalived]# vim keepalived.conf
```

删掉已有的所有默认配置，加入经过老男孩老师修改好的如下配置：

```
!

 Configuration File for keepalived
global_defs {
    notification_email {
    49000448-@qq.com
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id lb01     #<==id为


lb01，不同的


keepalived.conf此


ID要唯一



}
vrrp_instance VI_1 {     #<==实例名字为


VI_1，相同实例的备节点名字要和这个相同


    state MASTER     #<==状态为
```

MASTER，备节点状态需要为

BACKUP
    interface eth0   #<==通信接口为

eth0，此参数备节点设置和主节点相同

    virtual_router_id 55   #<==实例

ID为

55，

keepalived.conf里唯一

    priority 150   #<==优先级为

150，备节点的优先级必须比此数字低

    advert_int 1   #<==通信检查间隔时间

1秒

    authentication {
        auth_type PASS   #<==PASS认证类型，此参数备节点设置和主节点相同

        auth_pass 1111   #<==密码是

1111，此参数备节点设置和主节点相同

```
    }
    virtual_ipaddress {
        10.0.0.12/24 dev eth0 label eth0:


1
#<==虚拟


IP，即


VIP为


10.0.0.12，子网掩码为


24位，绑定接口为


eth0，别名为


eth0:


1，此参数备节点设置和主节点相同



    }
}提示：此处设置的虚拟


IP为


10.0.0.12
```

配置完毕后，启动Keepalived服务，如下：

```
[root@lb01 keepalived]# /etc/init.d/keepalived start正在启动


 keepalived:


                                                    [确定


]
```

然后检查配置结果，查看是否有虚拟IP 10.0.0.12。

```
[root@lb01 keepalived]# ip addr|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

出现上述带有vip：10.0.0.12行的结果表示lb01的Keepalived服务单实例配置成功。

2.实战配置Keepalived备服务器lb02 BACKUP

首先，配置lb02 BACKUP的keepalived.conf配置文件，操作步骤如下：

```
[root@lb02 ~]# cd /etc/keepalived/
[root@lb02 keepalived]# vim keepalived.conf
```

删掉已有的默认配置，加入经过老男孩修改好的如下配置（注意和lb01的不同）：

```
!


 Configuration File for keepalived
global_defs {
    notification_email {
    49000448-@qq.com
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id lb02
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 55
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.12/24 dev eth0 label eth0:



1
    }
}
```

配置完毕后，启动Keepalived服务，如下：

```
[root@lb02 keepalived]# /etc/init.d/keepalived start正在启动


 keepalived:


                                                [确定


]
```

然后检查配置结果，查看是否有虚拟IP 10.0.0.12。

```
[root@lb02 keepalived]# ip addr|grep 10.0.0.12
#<==这里没有返回任何结果就对了，因为


lb02为


BACKUP，当主节点活着的时候，它不会接管


VIP 10.0.0.12。
```

出现上述无任何结果的现象，表示lb02的Keepalived服务单实例配置成功。如果读者的配置过滤后有10.0.0.12的IP，则表示Keepalived工作不正常，同一个IP地址同一时刻应该只能出现一台服务器。

如果查看BACKUP备节点VIP有如下信息，说明高可用裂脑了，裂脑是两台服务器争抢同一资源导致的，例如：两边都配置了同一个VIP地址。

```
[root@lb02 keepalived]# ip addr|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

出现上述两台服务器争抢同一IP资源问题，一般要先考虑排查两个地方：

·主备两台服务器之间是否通信正常，如果不正常是否有iptables防

火墙阻挡？

·主备两台服务器对应的keepalived.conf配置文件是否有错误？例如，是否同一实例的virtual_router_id配置不一致。

3.进行高可用主备服务器切换实验

停掉主服务器上的Keealived服务或关闭主服务器，操作及检查步骤如下：

```
[root@lb01 keepalived]# ip addr|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
[root@lb01 keepalived]# /etc/init.d/keepalived stop停止


 keepalived:

                                                [确定


]
[root@lb01 keepalived]# ip addr|grep 10.0.0.12  #<==查看


VIP消失了
```

可以看到VIP 10.0.0.12消失了，此时查看BACKUP备服务器，看是否会有VIP 10.0.0.12出现，操作及检查步骤如下：

```
[root@lb02 keepalived]# ip addr|grep 10.0.0.12
[root@lb02 keepalived]# ip addr|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

可以看到备节点lb02已经接管绑定了10.0.0.12这个VIP，这期间备节点还会发送ARP广播，让所有的客户端更新本地的ARP表，以便客户端访问新接管VIP服务的节点。

此时如果再启动主服务器的Keealived服务，主服务器就会接管回VIP 10.0.0.12，启动后可以观察下主备的IP漂移情况，备服务器是否释放了IP？主服务器是否又接管了IP？

主节点启动Keepalived服务后，发现很快就又接管了VIP10.0.0.12，操作及检查步骤如下：

```
[root@lb01 keepalived]# /etc/init.d/keepalived start正在启动


 keepalived:


                                                [确定


]
[root@lb01 keepalived]# ip addr|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

与此同时，备节点上的VIP 10.0.0.12则被释放了，如下：

```
[root@lb02 keepalived]# ip addr|grep 10.0.0.12
```

这样就实现了单实例Keepalived服务IP自动漂移接管了，VIP漂移到了新机器新服务上，用户的访问请求自然就会找新机器新服务了。

 说明：这里仅实现了VIP的自动漂移切换，因此，仅适合两台服务器提供的服务均保持开启的应用场景，这也是工作中常用的高可用解决方案。

12.3.2　单实例主备模式Keepalived配置文件对比

表12-2为单实例主备模式Keepalived配置文件内容的对比，从表中可以看到主备配置的细微差别。

表12-2　单实例主备模式Keepalived配置文件内容对比

MASTER keepalived.conf

```
[root@lb01 ~]# cat /etc/keepalived/keepalived.conf -n
 1  ! Configuration File for keepalived
 2
 3  global_defs {
 4      notification_email {
 5      49000448-@qq.com
 6      }
 7      notification_email_from Alexandre.Cassen@
        firewall.loc
 8      smtp_server 127.0.0.1
 9      smtp_connect_timeout 30
10      router_id lb01
11  }
12
13  vrrp_instance VI_1 {
14      state MASTER
15      interface eth0
16      virtual_router_id 55
17      priority 150
18      advert_int 1
19      authentication {
20          auth_type PASS
21          auth_pass 1111
22      }
23      virtual_ipaddress {
24          10.0.0.12/24 dev eth0 label eth0:1
25      }
26  }
```

BACKUP keepalived.conf

```
[root@lb02 ~]# cat -n /etc/keepalived/keepalived.conf
 1  ! Configuration File for keepalived
 2
 3  global_defs {
 4      notification_email {
 5      49000448-@qq.com
 6      }
 7      notification_email_from Alexandre.Cassen@
        firewall.loc
 8      smtp_server 127.0.0.1
 9      smtp_connect_timeout 30
10      router_id lb02 #<== 此参数和 MASTER 有区别
11  }
12
13  vrrp_instance VI_1 {
14      state BACKUP #<== 此参数和 MASTER 有区别
15      interface eth0
16      virtual_router_id 55
17      priority 100 #<== 此参数和 MASTER 有区别
18      advert_int 1
19      authentication {
20          auth_type PASS
21          auth_pass 1111
22      }
23      virtual_ipaddress {
24          10.0.0.12/24 dev eth0 label eth0:1
25      }
26  }
```

表12-3为上述Keepalived单实例MASTER和BACKUP节点的配置差别项，只有3项是不同的。

表12-3　主备单实例keepalived.conf配置差别项

Keepalived 配置参数	MASTER 节点特殊参数	BACKUP 节点特殊参数
router_id（唯一标识）	router_id lb01	router_id lb02
state（角色状态）	state MASTER	state BACKUP
priority（竞选优先级）	priority 150	priority 100

12.4 Keepalived高可用服务器的“裂脑”问题

12.4.1 什么是裂脑

由于某些原因，导致两台高可用服务器对在指定时间内，无法检测到对方的心跳消息，各自取得资源及服务的所有权，而此时的两台高可用服务器对都还活着并在正常运行，这样就会导致同一个IP或服务在两端同时存在而发生冲突，最严重的是两台主机占用同一个VIP地址，当用户写入数据时可能会分别写入到两端，这可能会导致服务器两端的数据不一致或造成数据丢失，这种情况就被称为裂脑。

12.4.2　导致裂脑发生的原因

一般来说，裂脑的发生，有以下几种原因：

·高可用服务器对之间心跳线链路发生故障，导致无法正常通信。

·心跳线坏了（包括断了，老化）。

·网卡及相关驱动坏了，IP配置及冲突问题（网卡直连）。

·心跳线间连接的设备故障（网卡及交换机）。

·仲裁的机器出问题（采用仲裁的方案）。

·高可用服务器上开启了iptables防火墙阻挡了心跳消息传输。

·高可用服务器上心跳网卡地址等信息配置不正确，导致发送心跳失败。

·其他服务配置不当等原因，如心跳方式不同，心跳广播冲突、软件Bug等。


提示：Keepalived配置里同一VRRP实例如果virtual_router_id两端参数配置不一致，也会导致裂脑问题发生。

12.4.3 解决裂脑的常见方案

在实际生产环境中，我们可以从以下几个方面来防止裂脑问题的发生：

·同时使用串行电缆和以太网电缆连接，同时用两条心跳线路，这样一条线路坏了，另一个还是好的，依然能传送心跳消息。

·当检测到裂脑时强行关闭一个心跳节点（这个功能需特殊设备支持，如Stonith、fence）。相当于备节点接收不到心跳消息，通过单独的线路发送关机命令关闭主节点的电源。

·做好对裂脑的监控报警（如邮件及手机短信等或值班），在问题发生时人为第一时间介入仲裁，降低损失。例如，百度的监控报警短信就有上行和下行的区别。报警信息发送到管理员手机上，管理员可以通过手机回复对应数字或简单的字符串操作返回给服务器，让服务器根据指令自动处理相应故障，这样解决故障的时间更短。

当然，在实施高可用方案时，要根据业务实际需求确定是否能容忍这样的损失。对于一般的网站常规业务，这个损失是可容忍的。

12.4.4 解决Keepalived裂脑的常见方案

作为互联网应用服务器的高可用，特别是前端Web负载均衡器的高可用，裂脑的问题对普通业务的影响是可以忍受的，如果是数据库或者存储的业务，一般出现裂脑问题就非常严重了。因此，可以通过增加冗余心跳线路来避免裂脑问题的发生，同时加强对系统的监控，以便裂脑发生时人为快速介入解决问题。

·如果开启防火墙，一定要让心跳消息通过，一般通过允许IP段的形式解决。

·可以拉一条以太网网线或者串口线作为主被节点心跳线路的冗余。

·开发监测程序通过监控软件（例如Nagios）监测裂脑。

下面是生产场景检测裂脑故障的一些思路：

1）简单判断的思想：只要备节点出现VIP就报警，这个报警有两种情况，一是主机宕机了备机接管了；二是主机没宕，裂脑了。不管属于哪个情况，都进行报警，然后由人工查看判断及解决。

2）比较严谨的判断：备节点出现对应VIP，并且主节点及对应服务

（如果能远程连接主节点看是否有VIP就更好了）还活着，就说明发生裂脑了。

具体监测系统裂脑的脚本见本章结尾“开发监测Keepalived裂脑的脚本”一节。

12.5 Keepalived双实例双主模式配置

12.5.1 Keepalived双实例双主模式配置实战

前面给出的是Keepalived单实例主备模式的高可用演示，Keepalived还支持多实例多业务双向主备模式，即A业务在lb01上是主模式，在lb02上是备模式，而B业务在lb01上是备模式，在lb02上是主模式，下面就以双实例为例讲解不同业务实现双主的配置。表12-4为Keepalived双实例双主模式IP及VIP规划表。

表12-4 Keepalived双实例双主模式的IP及VIP规划表

HOSTNAME	IP	说明
lb01	10.0.0.7	VIP:10.0.0.12（用于绑定 A 服务 www.etiantian.org 域名）
lb02	10.0.0.8	VIP:10.0.0.13（用于绑定 B 服务 bbs.etiantian.org 域名）

首先，配置lb01 10.0.0.7的keepalived.conf，在单实例的基础上增加一个vrrp_instance VI_2实例，步骤及内容如下：

```
[root@lb01 ~]# cd /etc/keepalived/
[root@lb01 keepalived]# cat keepalived.conf !


 Configuration File for keepalived
global_defs {
    notification_email {
    49000448-@qq.com
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id lb01
```

```
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 55
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.12/24 dev eth0 label eth0:



1
    }
}
vrrp_instance VI_2 {
    state BACKUP
    interface eth0
    virtual_router_id 56
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.13/24 dev eth0 label eth0:



2
    }
}
```

提示：以vrrp_instance VI_1在lb01 10.0.0.7服务器上的角色为主，vrrp_instance VI_2在lb01 10.0.0.7服务器上的角色为备，加粗带底纹部分为增加的配置。

然后配置lb02 10.0.0.8的keepalived.conf，在单实例的基础上增加vrrp_instance VI_2实例，步骤及内容如下：

```
[root@lb02 keepalived]# cat keepalived.conf !
```

```
 Configuration File for keepalived
global_defs {
    notification_email {
    49000448-@qq.com
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id lb02
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 55
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.12/24 dev eth0 label eth0:



1
    }
}
vrrp_instance VI_2 {
    state MASTER
    interface eth0
    virtual_router_id 56
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.13/24 dev eth0 label eth0:



2
    }
}
```

提示：以vrrp_instance VI_1在lb02 10.0.0.8服务器上的角色为备，vrrp_instance VI_2在lb02 10.0.0.8服务器上的角色为主，加粗带底纹部分为增加的配置。

接着，在lb01、lb02上分别重启Keepalived服务，观察初始VIP设置情况。

lb01操作的结果如下：

```
[root@lb01 keepalived]# /etc/init.d/keepalived restart     停止

 keepalived:

                                                    [确定

]正在启动

 keepalived:

                                                    [确定


]
[root@lb01 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
[root@lb01 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

提示：启动lb01后，初始状态已经启动了10.0.0.12 VIP，即lb01由VI_1实例配置的VIP对外提供服务，例如可以把www.etiantian.org 解析到10.0.0.12上。

lb02操作的结果如下：

```
[root@lb02 keepalived]# /etc/init.d/keepalived restart停止

 keepalived:

                                                          [确定

]正在启动

 keepalived:

                                                          [确定

]
[root@lb02 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
    inet 10.0.0.13/24 scope global secondary eth0:

2
```

提示：启动lb02后，初始状态已经启动了10.0.0.13 VIP，即lb02由VI_2实例配置的VIP对外提供服务。例如可以把bbs.etiantian.org解析到10.0.0.13上。

下面停掉任意一端服务器或者Keepalived服务，然后看看VIP是不是会漂移到另一端。停止lb01的Keepalived服务：

```
[root@lb01 keepalived]# /etc/init.d/keepalived stop停止

 keepalived:
```

```
                                                    [确定

]
[root@lb01 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
#<==此处无显示信息
```

可以看到，停掉后，VIP 10.0.0.12即被释放。

现在，检查lb02服务器上VIP的接管情况：

```
[root@lb02 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
    inet 10.0.0.13/24 scope global secondary eth0:


2
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

可以看到，已经接管了lb01的VIP 10.0.0.12。

再次启动lb01的Keepalived服务：

```
[root@lb01 keepalived]# /etc/init.d/keepalived start正在启动


 keepalived:

                                                    [确定


]
[root@lb01 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
    inet 10.0.0.12/24 scope global secondary eth0:
```

```
1
```

在lb01上启动Keepalived服务后，很快它就接管回了自己的VIP。

此时，检查lb02服务器上此时IP的设置情况：

```
[root@lb02 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
    inet 10.0.0.13/24 scope global secondary eth0:


2
```

可以看到，已经释放了lb01的VIP。

若是停止lb02的Keepalived服务，VIP 10.0.0.13也立即被释放。

```
[root@lb02 keepalived]# /etc/init.d/keepalived stop停止


 keepalived:

                                                [确定


]
[root@lb02 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
#<==此处无显示信息
```

然后检查lb01服务器上IP的接管情况：

```
[root@lb01 keepalived]# ip add|egrep "10.0.0.12|10.0.0.13"
    inet 10.0.0.12/24 scope global secondary eth0:
```

```
1
    inet 10.0.0.13/24 scope global secondary eth0:

2
```

可以看到，它也接管了lb02的VIP 10.0.0.13。

到此为止，我们发现lb01、lb02主备节点已经实现了初始配置的VIP服务状态，当任意一端宕机，VIP可以实现互相切换接管。在实际工作中，可以把www.etiantian.org 解析到10.0.0.12提供服务，把bbs.etiantian.org解析到10.0.0.13提供服务，当然了，lb01、lb02也要配置相应服务，例如：Nginx反向代理服务等。

12.5.2　双实例双主模式的配置文件对比

Keepalived双实例双主模式在企业工作场景也是比较常用的，表12-5为大家展示了这种情况下主备配置文件的差别。

表12-5　双实例双主模式的配置文件对比

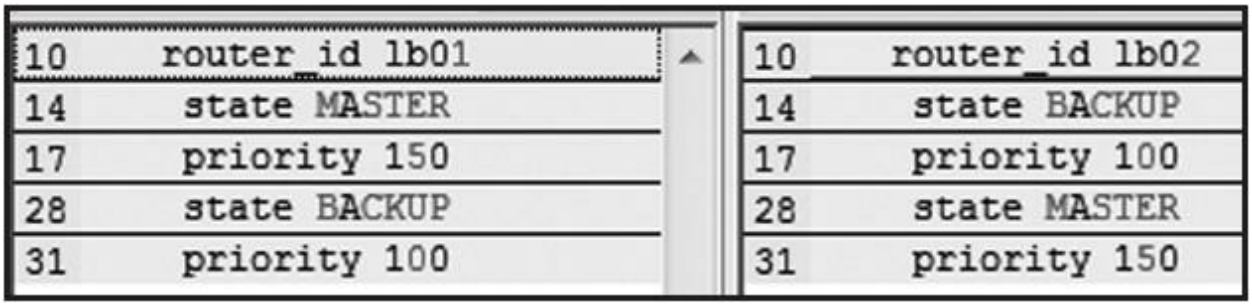
lb01 MASTER keepalived.conf

```
[root@lb01 keepalived]# cat -n keepalived.conf
 1  ! Configuration File for keepalived
 2
 3  global_defs {
 4      notification_email {
 5      49000448-@qq.com
 6      }
 7      notification_email_from Alexandre.
        Cassen@firewall.loc
 8      smtp_server 127.0.0.1
 9      smtp_connect_timeout 30
10      router_id lb01
11  }
12
13  vrrp_instance VI_1 {
14      state MASTER
15      interface eth0
16      virtual_router_id 55
17      priority 150
18      advert_int 1
19      authentication {
20          auth_type PASS
21          auth_pass 1111
22      }
23      virtual_ipaddress {
24          10.0.0.12/24 dev eth0 label eth0:1
25      }
26  }
27  vrrp_instance VI_2 {
28      state BACKUP
29      interface eth0
30      virtual_router_id 56
31      priority 100
32      advert_int 1
33      authentication {
34          auth_type PASS
35          auth_pass 1111
36      }
37      virtual_ipaddress {
38          10.0.0.13/24 dev eth0 label eth0:2
39      }
40  }
```

lb02 BACKUP keepalived.conf

```
[root@lb02 keepalived]# cat -n keepalived.conf
 1  ! Configuration File for keepalived
 2
 3  global_defs {
 4      notification_email {
 5      49000448-@qq.com
 6      }
 7      notification_email_from Alexandre.
        Cassen@firewall.loc
 8      smtp_server 127.0.0.1
 9      smtp_connect_timeout 30
10      router_id lb02
11  }
12
13  vrrp_instance VI_1 {
14      state BACKUP
15      interface eth0
16      virtual_router_id 55
17      priority 100
18      advert_int 1
19      authentication {
20          auth_type PASS
21          auth_pass 1111
22      }
23      virtual_ipaddress {
24          10.0.0.12/24 dev eth0 label eth0:1
25      }
26  }
27  vrrp_instance VI_2 {
28      state MASTER
29      interface eth0
30      virtual_router_id 56
31      priority 150
32      advert_int 1
33      authentication {
34          auth_type PASS
35          auth_pass 1111
36      }
37      virtual_ipaddress {
38          10.0.0.13/24 dev eth0 label eth0:2
39      }
40  }
```

图12-1为Keepalived双实例双主模式配置文件的区别。



10	router_id lb01	10	router_id lb02
14	state MASTER	14	state BACKUP
17	priority 150	17	priority 100
28	state BACKUP	28	state MASTER
31	priority 100	31	priority 150

图12-1　Keepalived双实例双主模式配置文件的区别

可以看出主备节点在增加的实例方面就两项区别：

·state（状态）

·priority（竞选优先级）

其中优先级决定VIP在哪个机器上初始运行。

12.6　Nginx负载均衡配合Keepalived服务案例实战

12.6.1　在lb01和lb02上配置Nginx负载均衡

结合第11章介绍的Nginx负载均衡的环境，根据图12-2调整好主负载均衡器lb01、备用负载均衡器lb02服务器上Nginx负载均衡环境，两台服务器的安装基础环境一模一样。

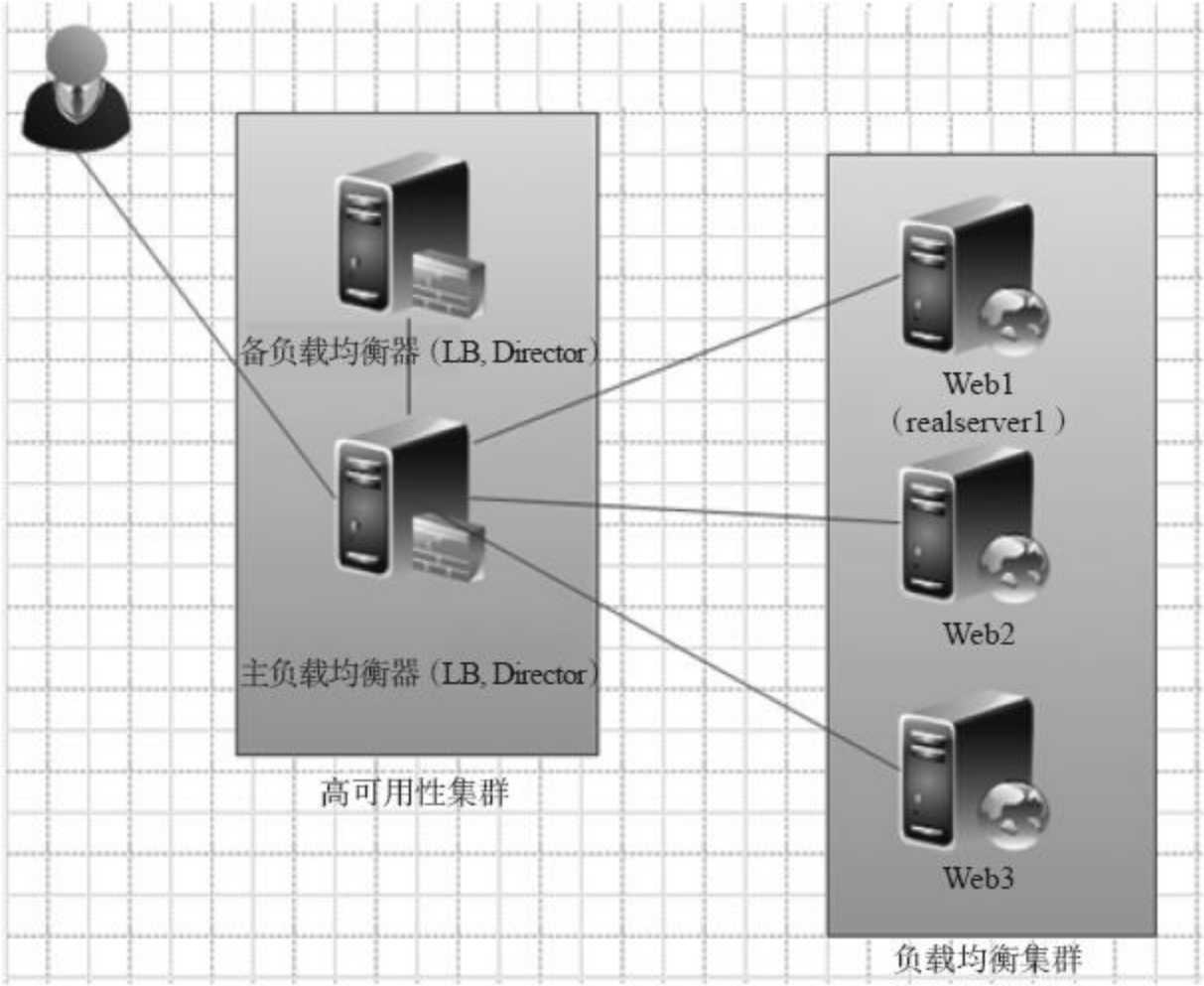


图12-2　Nginx负载均衡器高可用逻辑图

这里使用的Nginx负载均衡的配置如下：

```
[root@lb01 conf]# cat nginx.conf
worker_processes  1;



events {
    worker_connections  1024;



}
http {
    include       mime.types;



    default_type  application/octet-stream;



    sendfile        on;



    keepalive_timeout  65;



    upstream www_server_pools {
        server 10.0.0.9:


80  weight=1;



        server 10.0.0.10:


80  weight=1;
```

```
    }
    server {
        listen       10.0.0.12:


80;



        server_name  www.etiantian.org;



        location / {
        proxy_pass http:


// www_server_pools;



        proxy_set_header Host  $host;



        proxy_set_header X-Forwarded-For $remote_addr;



        }
    }
}
```

提示：此配置仅代理了www.etiantian.org 域名。

12.6.2　在lb01和lb02上配置Keepalived服务


说明：此处使用单实例为例进行配置说明。

lb01上Keepalived服务单实例主节点的配置如下：

```
[root@lb01 keepalived]# cat keepalived.conf!


 Configuration File for keepalived
global_defs {
    notification_email {
    49000448-@qq.com
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id lb01
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 55
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.12/24 dev eth0 label eth0:


1
    }
}
```


提示：VIP为10.0.0.12，即工作时需要把Nginx负载均衡代理的www.etiantian.org 解析到这个VIP。

lb02上Keepalived服务单实例备节点的配置如下：

```
[root@lb02 keepalived]# cat keepalived.conf!


 Configuration File for keepalived
global_defs {
    notification_email {
    49000448-@qq.com
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id lb02
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 55
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.12/24 dev eth0 label eth0:



1
    }
}
```

12.6.3　用户访问准备及模拟实际访问

准备工作如下。

1）在客户端hosts文件里把www.etiantian.org 域名解析到VIP 10.0.0.12上，正式场景需通过DNS解析。

```
10.0.0.12    www.etiantian.org
```

2）两台服务器配好Nginx负载均衡服务，并且确保后面代理的Web节点可以测试访问。

```
[root@lb01 keepalived]# lsof -i :


80
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nginx   6027  root    6u  IPv4  24194      0t0  TCP *:


http  (


LISTEN)


nginx   6028 nginx    6u  IPv4  24194      0t0  TCP *:


http  (


LISTEN)
```

```
[root@lb01 keepalived]# ip add|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

下面模拟实际的访问过程：

1）通过在客户端浏览器输入www.etiantian.org 测试访问，按Ctrl+F5键刷新几次，正常应该可以出现如图12-3所示的两种访问结果。

图12-3　客户端访问负载均衡测试结果

2）此时停止lb01服务器或停掉Keepalived服务，观察业务是否正常：

```
[root@lb01 conf]# /etc/init.d/keepalived stop停止

 keepalived:

                                                [确定

]
[root@lb01 conf]# ip add|grep 10.0.0.12
```

提示：严格来讲应该关掉服务器来模拟才是最佳的。

3）观察lb02备节点是否接管VIP10.0.0.12。

```
[root@lb02 keepalived]# ip add|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

再次在客户端浏览器输入www.etiantian.org 测试访问，按Ctrl+F5键刷新几次，正常应该可以出现和切换lb02前相同的访问结果（如图12-4所示）。

图12-4　客户端访问负载均衡测试结果

4）开启lb01的Keepalived服务。

```
[root@lb01 conf]# ip add|grep 10.0.0.12
[root@lb01 conf]# /etc/init.d/keepalived start正在启动


 keepalived:


                                                [确定
```

```
]
[root@lb01 conf]# ip add|grep 10.0.0.12
[root@lb01 conf]# ip add|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

可以看到，VIP很快就接管回来了，此时浏览器访问结果依然正常。

12.7 解决服务监听的网卡上不存在IP地址问题

如果配置使用“listen 10.0.0.12：80；”的方式指定IP监听服务，而本地的网卡上没有10.0.0.12这个IP，Nginx就会报错，如下：

```
[root@lb01 server]# /application/nginx/sbin/nginx
nginx:

 [emerg] bind ()

 to 10.0.0.12:

80 failed  (

99:

 Cannot assign requested address)
```

如果要实施双主即主备同时跑不同的业务，配置文件里指定了IP监听，备节点则会因为网卡实际不存在VIP也报错。

出现上面问题的原因就是在物理网卡上没有与配置文件里监听的IP相对应的IP，解决办法是在/etc/sysctl.conf中加入如下内核参数配置：

```
net.ipv4.ip_nonlocal_bind = 1
```

也可通过如下命令快速追加：

```
echo 'net.ipv4.ip_nonlocal_bind = 1' >> /etc/sysctl.conf注

*：

4.net.ipv4.ip_nonlocal_bind = 1 #此项表示启动

nginx而忽略配置中监听的

VIP是否存在，它同样适合

Haproxy。
```

最后执行sysctl-p使上述修改生效。

12.8 解决高可用服务只针对物理服务器的问题

默认情况下Keepalived软件仅仅在对方机器宕机或Keepalived停掉的时候才会接管业务。但在实际工作中，有业务服务停止而Keepalived服务还在工作的情况，这就会导致用户访问的VIP无法找到对应的服务，那么，如何解决业务服务宕机可以将IP漂移到备节点使之接管提供服务呢？

第一个方法：可以写守护进程脚本来处理。当Nginx业务有问题时，就停掉本地的Keepalived服务，实现IP漂移到对端继续提供服务。实际工作中部署及开发的示例脚本如下：

```
[root@lb01 script]# cat check_nginx.sh
#!



/bin/sh
while true
do
 if [ `netstat -lntup|grep nginx|wc -l` -ne 1 ];



then
    /etc/init.d/keepalived stop
 fi
  sleep 5
done
```

此脚本的基本思想是若没有80端口存在，就停掉Keepalived服务实现释放本地的VIP。

在后台执行上述脚本并检查：

```
[root@lb01 script]# sh check_nginx.sh &
[1] 10231
[root@lb01 script]# ps -ef|grep check|grep -v grep
root     10231  8382  0 22:


36 pts/0    00:


00:


00 sh check_nginx.sh
```

确认Nginx以及Keepalived服务是正常的。

```
[root@lb01 script]# netstat -lntup|grep nginx
tcp     0     0 10.0.0.12:


80     0.0.0.0:


*     LISTEN     9244/nginx
[root@lb01 script]# /etc/init.d/keepalived status
keepalived (


pid  10205)


 正在运行


...
```

然后模拟Nginx服务挂掉，看IP是否发生切换。

```
[root@lb01 script]# /application/nginx/sbin/nginx -s stop
[root@lb01 script]# 停止


 keepalived:


  [确定


] #<==这是停掉


nginx后系统的提示输出。



[root@lb01 script]# /etc/init.d/keepalived status
keepalived 已停



[root@lb01 script]# netstat -lntup|grep nginx
```

此时，备节点已接管：

```
[root@lb02 conf]# ip add|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
```

第二个方法：可以使用Keepalived的配置文件参数触发写好的监测服务脚本。

首先要开发监测服务脚本，注意这个脚本与上一个脚本的不同。

```
[root@lb01 scripts]# cat chk_nginx_proxy.sh
#!
```

```
/bin/sh
if [ `netstat -lntup|grep nginx|wc -l` -ne 1 ];


then
    /etc/init.d/keepalived stop
fi
[root@lb01 scripts]# chmod +x chk_nginx_proxy.sh
[root@lb01 scripts]# ls -l chk_nginx_proxy.sh
-rwxr-xr-x 1 root root 97 7月


  17 22:


50 chk_nginx_proxy.sh
```

此时，Keepalived服务的完整配置为：

```
[root@lb01 script]# cat /etc/keepalived/keepalived.conf!


 Configuration File for keepalived
global_defs {
    notification_email {
    49000448-@qq.com
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id lb01
}
vrrp_script chk_nginx_proxy {     #     <==定义


vrrp脚本，检测


HTTP端口



script "/server/scripts/chk_nginx_proxy.sh"     #     <==执行脚本，当


Nginx服务有问题，就停掉
```

Keepalived服务

interval 2   #   <==间隔

2秒

weight 2
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 55
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.12/24 dev eth0 label eth0:

1
    }
    track_script {
    chk_nginx_proxy   #   <==触发检查

    }
}

下面测试接管结果。

[root@lb01 server]# netstat -lntup|grep nginx
tcp   0   0 10.0.0.12:

80   0.0.0.0:

```
*     LISTEN     12161/nginx
[root@lb01 server]# /etc/init.d/keepalived status
keepalived （


pid  12177）


 正在运行


...
[root@lb01 server]# ip add|grep 10.0.0.12
    inet 10.0.0.12/24 scope global secondary eth0:


1
[root@lb01 server]# /application/nginx/sbin/nginx -s stop
[root@lb01 server]# ip add|grep 10.0.0.12
[root@lb01 server]# /etc/init.d/keepalived status
keepalived 已停
```

当停掉Nginx的时候，Keepalived 2秒钟内会被自动停掉，VIP被释放，由对端接管，这样就实现了即使服务宕机也会进行IP漂移，业务切换。

12.9　解决多组Keepalived服务器在一个局域网的冲突问题

当在同一个局域网内部署了多组Keepalived服务器对，而又未使用专门的心跳线通信时，可能会发生高可用接管的严重故障问题。前文已经讲解过Keepalived高可用功能是通过VRRP协议实现的，VRRP协议默认通过IP多播的形式实现高可用对之间的通信，如果同一个局域网内存在多组Keepalived服务器对，就会造成IP多播地址冲突问题，导致接管错乱，不同组的Keepalived都会使用默认的224.0.0.18作为多播地址。此时的解决办法是，在同组的Keepalived服务器所有的配置文件里指定独一无二的多播地址，配置如下：

```
global_defs {
    router_id LVS_19
    vrrp_mcast_group4 224.0.0.19  #<==这个就是指定多播地址的配置



}
```

提示：

1）不同实例的通信认证密码也最好不同，以确保接管正常。

2）另一款高可用软件Heartbeat，如果采用多播方式实现主备通

信，同样会有多播地址冲突问题。

12.10　配置指定文件接收Keepalived服务日志

默认情况下Keepalived服务日志会输出到系统日志/var/log/messages，和其他日志信息混合在一起，很不方便，可以将其调整成由独立的文件记录Keepalived服务日志。操作步骤如下：

1）编辑配置文件/etc/sysconfig/keepalived，将第14行的“KEEPALIVED_OPTIONS="-D""修改为“KEEPALIVED_OPTIONS="-D-d-S 0""，快速修改方法为：

```
[root@lb01 server]# sed -i '14 s#KEEPALIVED_OPTIONS="-D"#KEEPALIVED_OPTIONS="-D -d -
[root@lb01 server]# sed -n '14p' /etc/sysconfig/keepalived
KEEPALIVED_OPTIONS="-D -d -S 0"说明：可以查看


/etc/sysconfig/keepalived里注释获得上述参数的说明


# --dump-conf          -d      导出备份配置数据


# --log-detail          -D      详细日志


# --log-facility          -S      设置本地的


syslog设备，编号


0-7（
```

```
default=LOG_DAEMON)

# -S 0 表示指定为

local0设备
```

2）修改rsyslog的配置文件vi/etc/rsyslog.conf，在结尾处加入如下2行内容：

```
#keepalived
local0.*                                                /var/log/keepalived.log
```

上述配置表示来自local0设备的所有日志信息都记录到/var/log/keepalived.log文件。

然后在约第42行如下信息的第一列结尾加入“；local0.none”：

```
*.info;

mail.none;

authpriv.none;

cron.none;

local0.none   /var/log/messages
```

上述配置表示来自local0设备的所有日志信息不再记录于/var/log/messages里。

3）配置完成后，重启rsyslog服务。

```
[root@lb01 server]# /etc/init.d/rsyslog restart关闭系统日志记录器：

                                                           [确定

]启动系统日志记录器：

                                                           [确定

]
```

4）测试Keepalived日志记录结果。在重启Keepalived服务后，就会把日志信息输出到rsyslog定义的/var/log/keepalived.log文件，如下：

```
[root@lb01 server]# tail /var/log/keepalived.log
Jul 17 23：


22：


44 lb01 Keepalived_healthcheckers[12373]：


  Smtp server connection timeout = 30
Jul 17 23：


22：
```

44 lb01 Keepalived_healthcheckers[12373]:

  Email notification from = Alexandre.Cassen@firewall.loc
Jul 17 23:

22:

44 lb01 Keepalived_healthcheckers[12373]:

  Email notification = 49000448-@qq.com
Jul 17 23:

22:

44 lb01 Keepalived_healthcheckers[12373]:

  VRRP IPv4 mcast group = 224.0.0.18
Jul 17 23:

22:

44 lb01 Keepalived_healthcheckers[12373]:

  VRRP IPv6 mcast group = 224.0.0.18
Jul 17 23:

22:

44 lb01 Keepalived_healthcheckers[12373]:

  SNMP Trap disabled
Jul 17 23:

22:

44 lb01 Keepalived_healthcheckers[12373]:

 ------< SSL definitions >------
Jul 17 23:

22:

44 lb01 Keepalived_healthcheckers[12373]:

  Using autogen SSL context
Jul 17 23:

22:

44 lb01 Keepalived_healthcheckers[12373]:

 Using LinkWatch kernel netlink reflector...
Jul 17 23:

22:

44 lb01 Keepalived[12371]:

 Stopping Keepalived v1.2.13 (

03/19,

2015)

如果要求更高，还可以在/var/log/messages上设置对/var/log/keepalived.log进行轮询，以防单个日志文件变得太大。

```
[root@lb01 server]# >/var/log/messages #<==清空所有日志，然后查看是否还有记录到

messages。

[root@lb01 server]# tail /var/log/messages
```

12.11　开发监测Keepalived裂脑的脚本

检测思路：在备节点上执行脚本，如果可以ping通主节点并且备节点有VIP就报警，让人员介入检查是否裂脑。

1）在lb02备节点开发脚本并执行。

```
[root@lb02 scripts]# cat check_split_brain.sh
#!



/bin/sh
lb01_vip=10.0.0.12
lb01_ip=10.0.0.7
while true
do
ping -c 2 -W 3 $lb01_ip &>/dev/null
 if [ $ -eq 0 -a `ip add|grep "$lb01_vip"|wc -l` -eq 1 ]
  then
    echo "ha is split brain.warning."
else
    echo "ha is ok"
fi
sleep 5
done
[root@lb02 scripts]# sh check_split_brain.sh
ha is ok
ha is ok
```

正常情况下，主节点活着，VIP 10.0.0.12在主节点，因此不会报警，提示"ha is ok"。

2）停止Keepalived服务看lb02脚本执行情况。

lb01上：

```
[root@lb01 scripts]# /etc/init.d/keepalived stop停止


 keepalived:


                                                    [确定


]
[root@lb01 scripts]# ip add|grep 10.0.0.12
```

在lb02上观察即可，此前脚本已经执行。

```
[root@lb02 scripts]# sh check_split_brain.sh
ha is ok
ha is ok
ha is split brain.warning.
ha is split brain.warning.
ha is split brain.warning.
```

3）关掉lb01服务器，然后再观察lb02脚本的输出。

```
[root@lb02 scripts]# sh check_split_brain.sh
ha is ok
ha is ok
ha is split brain.warning.
ha is split brain.warning.
ha is split brain.warning.
ha is split brain.warning.
ha is split brain.warning.
ha is split brain.warning.
ha is ok
ha is ok
ha is ok
ha is ok
```

裂脑报警恢复了。

4）可以将此脚本整合到Nagios或Zabbix监控服务里，进行监控报警。

12.12　本章重点回顾

1）Keepalived及VRRP协议原理。

2）Keepalived配置文件重要参数。

3）Keepalived单多实例VIP漂移的配置及实战。

4）裂脑的概念及如何规避解决裂脑问题。

5）Keepalived单多实例结合Nginx代理配置实战。

6）服务监听网卡上不存在IP地址问题解决方案。

7）高可用服务只针对物理服务器问题解决方案。

8）多组Keepalived服务器对通信冲突问题解决方案。

9）配置指定文件接收Keepalived服务日志。

10）开发监测Keepalived裂脑的思路及脚本。

第13章　企业级Memcached服务应用实践

本章主要介绍企业级Memcached服务的应用实践，包括软件工作运行原理和应用场景以及生产中的部署、优化等经验。

Memcached是一套仅利用系统内存进行数据缓存的软件，常用于在动态Web集群系统后端、数据库前端等处，可临时缓存Web系统查询过的数据库数据，当用户请求查询数据时，由Memcached优先提供服务，从而减少Web系统直接请求数据库的次数，这极大地降低了后端数据库的压力，也因此提升了网站系统的性能；Memcached服务是通过预分配指定的内存空间来存取数据的，因此它比MySQL这样的数据库要快很多（因为MySQL数据库需要频繁操作磁盘）。另外，Memcached也经常用于共享存储集群架构后端所有Web节点服务器的session会话数据。学习完本章，读者可以掌握Memcached服务的企业级技术实战，轻松构建以及维护大型网站中后端的缓存系统架构。

13.1 Memcached介绍

13.1.1 Memcached与常见同类软件对比

1.Memcached是什么？

Memcached是一个开源的、支持高性能、高并发的分布式内存缓存系统，由C语言编写，总共2000多行代码。从软件名称上看，前3个字符“Mem”就是内存的意思，而接下来的后面5个字符“cache”就是缓存的意思，最后一个字符d，是daemon的意思，代表是服务器端守护进程模式服务。

Memcached服务分为服务器端和客户端两部分，其中，服务器端软件的名字形如Memcached-1.4.24.tar.gz，客户端软件的名字形如Memcache-2.25.tar.gz。

Memcached软件诞生于2003年，最初由LiveJournal的Brad Fitzpatrick开发完成。Memcache是整个项目的名称，而Memcached是服务器端的主程序名，因其协议简单，应用部署方便、且支持高并发，因此被互联网企业广泛使用，直到现在仍然如此。其官方网站地址：http://memcached.org/ 。

2.Memcached的作用

传统场景中，多数Web应用都将数据保存到关系型数据库中（例如：MySQL），Web服务器从中读取数据并在浏览器中显示。但随着数据量的增大、访问的集中，关系型数据库的负担就会出现加重、响应缓慢、导致网站打开延迟等问题，影响用户体验。

这时就需要Memcached软件出马了。使用Memcached的主要目的是，通过在自身内存中缓存关系型数据库的查询结果，减少数据库被访问的次数，以提高动态Web应用的速度、提高网站架构的并发能力和可扩展性。

Memcached服务的运行原理是通过在事先规划好的系统内存空间中临时缓存数据库中的各类数据，以达到减少前端业务服务对数据库的直接高并发访问，从而提升大规模网站集群中动态服务的并发访问能力。

生产场景的Memcached服务一般被用来保存网站中经常被读取的对象或数据，就像我们的客户端浏览器也会把经常访问的网页缓存起来一样，通过内存缓存来存取对象或数据要比磁盘存取快很多，因为磁盘是机械的，因此，在当今的IT企业中，Memcached的应用范围很广泛。

13.1.2 互联网常见内存缓存服务软件

表13-1为互联网企业场景常见内存缓存服务软件相关对比信息。

表13-1 互联网企业常见内存缓存服务软件的相关信息对比

软　件	类　型	主要作用	缓存的数据
Memcached	纯内存型	常用于缓存网站后端的各类数据，例如数据库中的数据	主要缓存用户重复请求的动态内容，例如：blog 的博文、BBS 的帖子等内容以及用户的 session 会话信息
Redis、Memcachedb	可持久化存储，即使用内存，也会使用磁盘存储	• 缓存后端数据库的查询数据 • 作为关系数据库的重要补充	作为缓存时，主要缓存用户重复请求的动态内容，例如：blog 的博文、BBS 的帖子等内容 作为数据库的有效补充时，例如：好友关注、粉丝统计、业务统计等功能可以用持久化存储

（续）

软　件	类　型	主要作用	缓存的数据
Squid、Nginx	内存或内存加磁盘缓存	主要用于缓存 Web 前端的服务内容	主要用于静态数据缓存，例如：图片、附件（压缩包）及 JS、CSS、html 静态代码等，此部分功能大多数企业会选择专业的 CDN 公司，如：网宿、蓝讯

13.2 Memcached的用途与应用场景

13.2.1 Memcached常见用途工作流程

Memcached是一种内存缓存软件，在工作中经常用来缓存数据库的查询数据，数据被缓存在事先预分配的Memcached管理的内存中，可以通过API或命令的方式存取内存中缓存的这些数据，Memcached服务内存中缓存的数据就像一张巨大的hash表，每条数据都是以key-value对的形式存在。

1.网站读取Memcached数据时工作流程

从逻辑上来说，当程序访问后端数据库获取数据时会优先访问Memcached缓存，如果缓存中有数据就直接返回给客户端用户，如果没有合适的数据（没有命中），再去后端的数据库读取数据，读取到需要的数据后，就会把数据返回给客户端，同时还会把读取到的数据缓存到Memcached内存中，这样客户端用户再次请求相同的数据时就会直接读取Memcached缓存的数据了，这就大大地减轻了后端数据库的压力，并提高了整个网站的响应速度，提升了用户体验。

图13-1展示了Memcached缓存系统和后端数据库系统的协作流程。

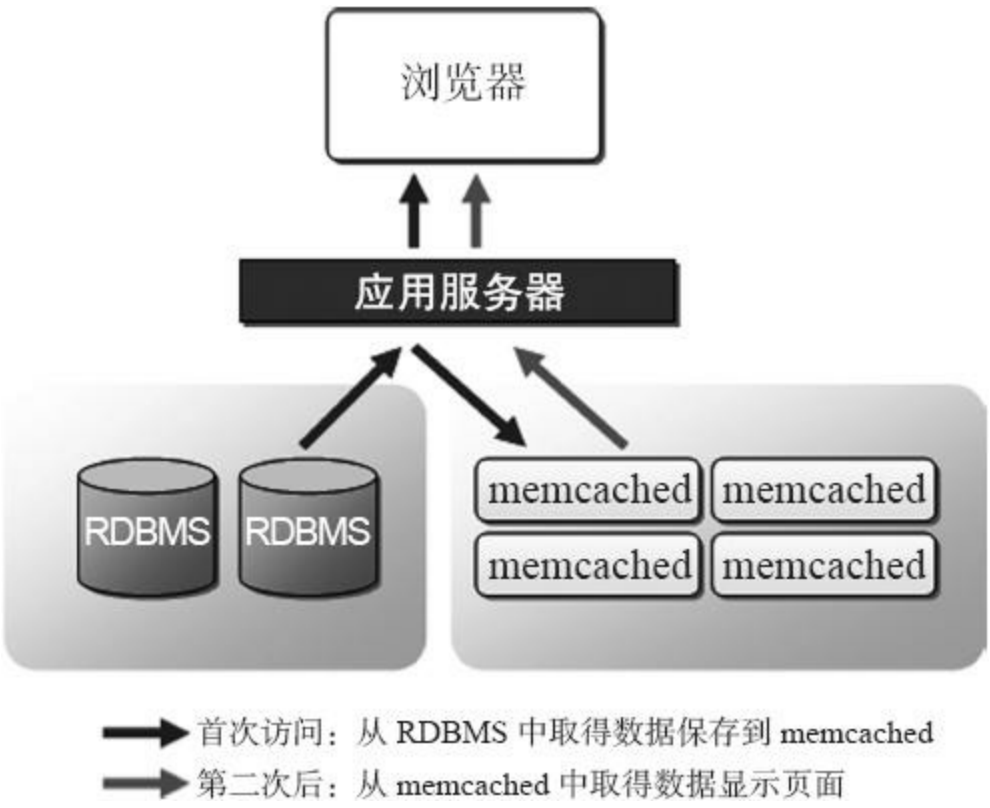


图13-1　Memcached作为网站缓存应用读数据流程图

在图13-1中，使用Memcached缓存查询的数据来减少数据库压力的具体工作流程如下：

1）Web程序首先检查客户端请求的数据是否在Memcached缓存中存在，如果存在，直接把请求的数据返回给客户端，此时不再请求后端数据库。

2）如果请求的数据在Memcached缓存中不存在，则程序会去请求数据库服务，把从数据库中取到的数据返回给客户端，同时把新取到的数据缓存一份到Memcached缓存中。

2.网站更新Memcached数据时的工作流程

具体流程如下：

1）当程序更新或删除数据时，会首先处理后端数据库中的数据。

2）在处理后端数据库中数据的同时，也会通知Memcached，告诉它对应的旧数据失效，从而保证Memcached中缓存的数据始终和数据库中一致，这个数据一致性非常重要，也是大型网站分布式缓存集群最头疼的问题所在。

3）如果是在高并发读写场合，除了要程序通知Memcached过期的缓存失效外，还可能要通过相关机制，例如在数据库上部署相关程序（如在数据库中设置触发器使用UDFs），实现当数据库有更新时就把数据更新到Memcached服务中，这样一来，客户端在访问新数据时，因预先把更新过的数据库数据复制到Memcached中缓存起来了，所以可以减少第一次查询数据库带来的访问压力，提升Memcached中缓存的命中率，甚至新浪门户还会把持久化存储Redis做成MySQL数据库的从库，实现真正的主从复制。

Memcached网站作为缓存应用更新数据流程见图13-2。

Memcached服务作为缓存应用通过相关软件更新数据流程见图13-3。

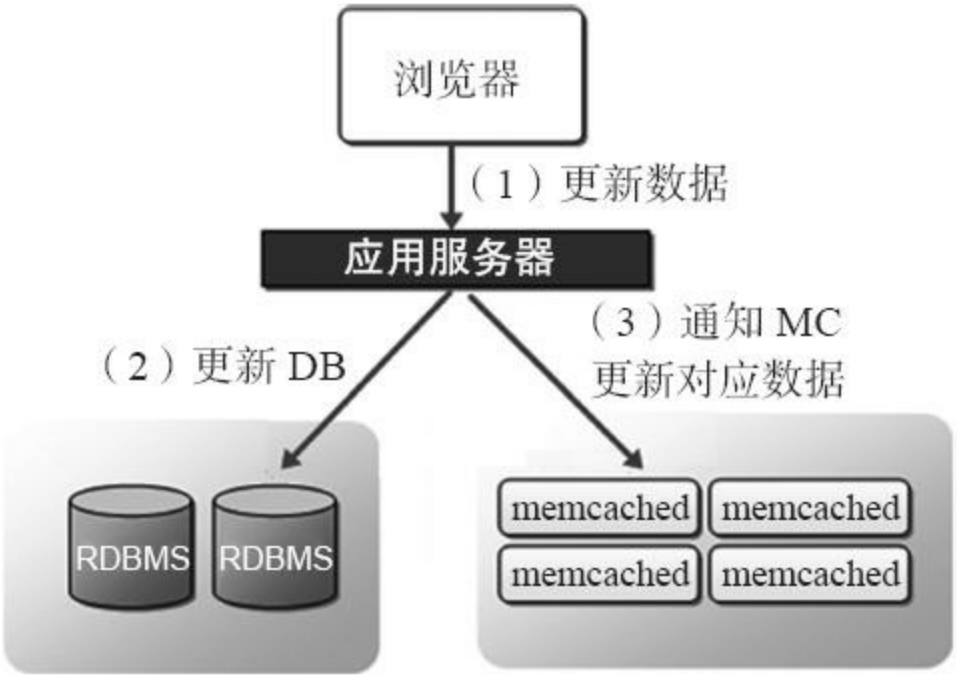


图13-2　Memcached网站作为缓存应用更新数据流程图

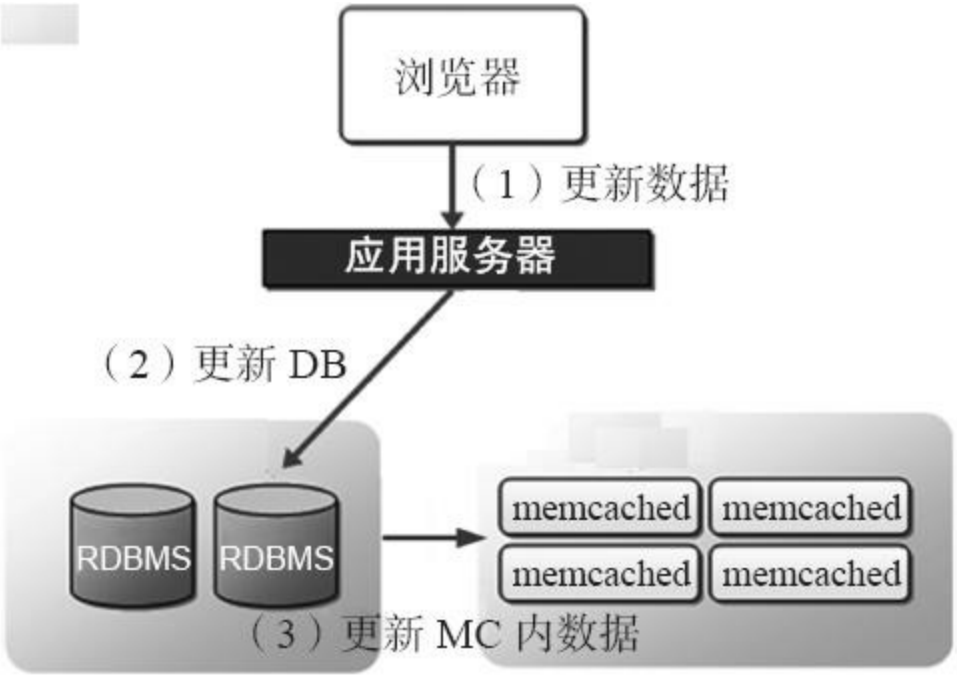


图13-3　Memcached作为缓存应用通过软件更新数据流程

在生产工作中，图13-2比图13-3的方案更为常用，老男孩也推荐采

用图13-2的架构方案，即由网站程序负责更新Memcached缓存。

13.2.2 Memcached在企业中的应用场景

1.作为数据库的查询数据缓存

（1）完整数据缓存

例如：电商的商品分类功能不是经常变动的，因此可以事先放到Memcached里，然后再对外提供数据访问。这个过程被称之为“数据预热”。

此时只需读取缓存，无需读取数据库就能得到Memcached缓存里的所有商品分类数据了，所以数据库的访问压力就会大大降低。

为什么商品分类数据可以事先放在缓存里呢？

因为，商品分类几乎都是由内部人员管理的，如果需要更新数据，更新数据库后，就可以把数据同时更新到Memcached里。

如果把商品分类数据做成静态化文件，然后，通过在前端Web缓存或者使用CDN加速效果更好。

（2）热点数据缓存

热点数据缓存一般是用于由用户更新的商品，例如淘宝的卖家，在

卖家新增商品后，网站程序就会把商品写入后端数据库，同时把这部分数据，放入Memcached内存中，下一次访问这个商品的请求就直接从Memcached内存中取走了。这种方法用来缓存网站热点的数据，即利用Memcached缓存经常被访问的数据。

 提示：这个过程可以通过程序实现，也可以在数据库上安装相关软件进行设置，直接由数据库把内容更新到Memcached中，就相当于Memcached是MySQL的从库一样。

如果碰到电商双11、秒杀高并发的业务场景，必须要事先预热各种缓存，包括前端的Web缓存和后端的数据库缓存。

也就是先把数据放入内存预热，然后逐步动态更新。此时，会先读取缓存，如果缓存里没有对应的数据，再去读取数据库，然后把读到的数据放入缓存。如果数据库里的数据更新，需要同时触发缓存更新，防止给用户过期的数据，当然对于百万级别并发还有很多其他的工作要做。

绝大多数的网站动态数据都是保存在数据库当中的，每次频繁地存取数据库，会导致数据库性能急剧下降，无法同时服务更多的用户（比如MySQL特别频繁的锁表就存在此问题），那么，就可以让Memcached来分担数据库的压力。增加Memcached服务的好处除了可以分担数据库的压力以外，还包括无须改动整个网站架构，只须简单地修改下程序逻

辑，让程序先读取Memcached缓存查询数据即可，当然别忘了，更新数据时也要更新Memcached缓存。

2.作为集群节点的session会话共享存储

即把客户端用户请求多个前端应用服务集群产生的session会话信息，统一存储到一个Memcached缓存中。由于session会话数据是存储在内存中的，所以速度很快。

图13-4为Memcached服务在企业集群架构中的常见工作位置。

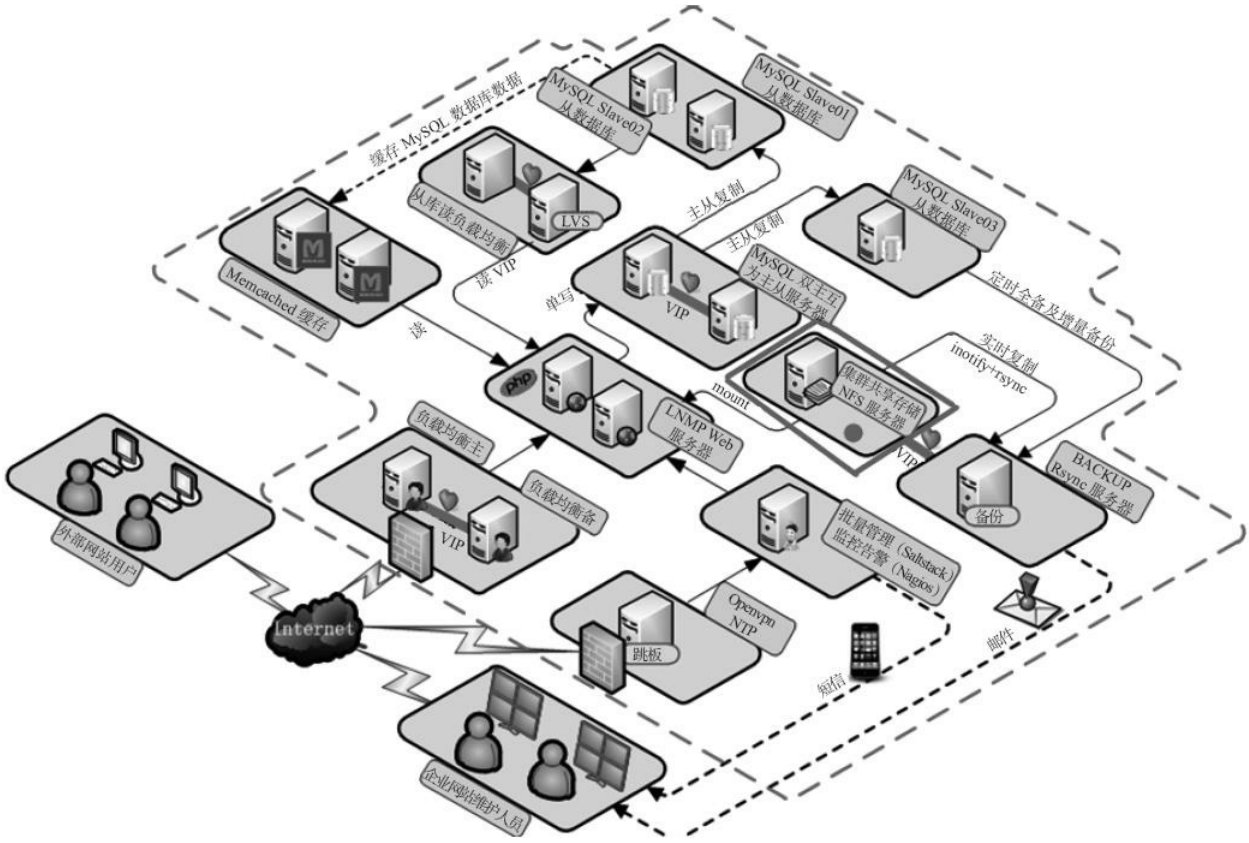


图13-4　Memcached服务在企业集群架构中的常见位置

13.3 Memcached的特点与工作机制

13.3.1 Memcached的特点

Memcached作为高并发、高性能的缓存服务，具有如下特点：

·协议简单。Memcached的协议实现很简单，采用的是基于文本行的协议，能通过telnet/nc等命令直接操作Memcached服务存取数据。

·支持epoll/kqueue异步I/O模型，使用libevent作为事件处理通知机制。

简单的说，libevent是一套利用C开发的程序库，它将BSD系统的kqueue、Linux系统的epoll等事件处理功能封装成一个接口，确保即使服务器端的连接数增加也能发挥很好的性能。Memcached就是利用这个libevent库进行异步事件处理的。

·采用key/value键值数据类型。

被缓存的数据以key/value键值形式存在，例如：

oldboy→36，key=oldboy，value=36

oldgirl→28，key=oldgirl，value=28

通过oldboy key可以获取到36值，同理通过oldgirl key可以获取28值。

·全内存缓存，效率高。Memcached管理内存的方式非常高效，即全部的数据都存放于Memcached服务事先分配好的内存中，无持久化存储的设计，和系统的物理内存一样，当重启系统或Memcached服务时，Memcached内存中的数据就会丢失。

如果希望重启后，数据依然能保留，那么就可以采用Redis这样的持久性内存缓存系统，更多的开源软件见 http://oldboy.blog.51cto.com/2561410/775056 。

当内存中缓存的数据容量达到服务启动时设定的内存值时，就会自动使用LRU算法删除过期的缓存数据。也可以在存放数据时对存储的数据设置过期时间，这样过期后数据就自动被清除，Memcached服务本身不会监控数据过期，而是在访问的时候查看key的时间戳判断是否过期。

·可支持分布式集群。

Memcahced没有像MySQL那样的主从复制方式，分布式Memcahced集群的不同服务器之间是互不通信的，每一个节点都独立存取数据，并且数据内容也不一样。通过对Web应用端的程序设计或者通过支持hash算法的负载均衡软件，可以让Memcached支持大规模海量分布式缓存集

群应用。

下面是利用Web端程序程序实现Memcahced分布式的简单代码：

```
"memcached_servers" => array(

        '10.4.4.4:
11211',

        '10.4.4.5:
11211',

        '10.4.4.6:
11211',

    )
```

下面使用Tengine反向代理负载均衡的一致性哈希算法实现分布式Memcahced的配置。

```
http {
    upstream test {
        consistent_hash $request_uri;
```

```
        server 127.0.0.1:
11211 id=1001 weight=3;

        server 127.0.0.1:
11212 id=1002 weight=10;

        server 127.0.0.1:
11213 id=1003 weight=20;

    }
}
```

 提示：Tengine是淘宝网开源的Nginx的分支，上述代码来自：

http://tengine.taobao.org/document_cn/http_upstream_consistent_hash_c

。

13.3.2 Memcached工作原理与机制

1.Memcached工作原理

Memcached是一套类似C/S模式架构的软件，在服务器端启动Memcached服务守护进程，可以指定监听本地的IP地址、端口号、并发访问连接数，以及分配了多少内存来处理客户端请求。

2.Socket事件处理机制

Memcached软件是由C语言来实现的，全部代码仅有2000多行，采用的是异步epoll/kqueue非阻塞I/O网络模型，其实现方式是基于异步的libevent事件单进程、单线程模式。使用libevent作为事件通知机制，应用程序端通过指定服务器的IP地址及端口，就可以连接Memcached服务进行通信。

3.数据存储机制

需要被缓存的数据以key/value键值对的形式保存在服务器端预分配的内存区中，每个被缓存的数据都有唯一的标识key，操作Memcached中的数据就是通过这个唯一标识的key进行的。缓存到Memcached中的数据仅放置在Memcached服务预分配的内存中，而非存储在Memcached服务器所在的磁盘上，因此存取速度非常快。

由于Memcached服务自身没有对缓存的数据进行持久化存储的设计，因此，在服务器端的Memcached服务进程重启之后，存储在内存中的这些数据就会丢失。且当内存中缓存的数据容量达到启动时设定的内存值时，也会自动使用LRU算法删除过期的数据。

开发Memcached的初衷仅是通过内存缓存提升访问效率，并没有过多考虑数据的永久存储问题。因此，如果使用Memcached作为缓存数据服务，要考虑数据丢失后带来的问题，例如：是否可以重新生成数据，还有，在高并发场合下缓存宕机或重启会不会导致大量请求直接到数据库，导致数据库无法承受，最终导致网站架构雪崩等。

4.内存管理机制

Memcached采用了如下机制：

·采用slab内存分配机制。

·采用LRU对象清除机制。

·采用hash机制快速检索item。

5.多线程处理机制

多线程处理时采用的是pthread（POSIX）线程模式。

若要激活多线程，可在编译时指定：./configure--enable-threads。

锁机制不够完善。

负载过重时，可以开启多线程（-t线程数为CPU核数）。

13.3.3 Memcached预热理念及集群节点的正确重启方法

1.Memcached预热理念

当需要大面积重启Memcached时，首先要在前端控制网站入口的访问流量，然后，重启Memcached集群并进行数据预热，所有数据都预热完毕之后，再逐步放开前端网站入口的流量。

为了满足Memcached服务数据可以持久化存储的需求，在较早时期，新浪网基于Memcached服务开发了一款NoSQL软件，名字为MemcacheDB，实现了在缓存的基础上增加了持久存储的特性，不过目前逐步被更优秀的Redis软件取代了。

2.如何正确开启网站集群服务器

如果由于机房断电或者搬迁服务器集群到新机房，那么启动集群服务器时，一定要从网站集群的后端依次往前端开启，特别是开启Memcached缓存服务器时要提前预热。

13.4 Memcached内存管理

13.4.1 Memcached内存管理机制深入剖析

1.Malloc内存管理机制

在讲解Memcached内存管理机制前，先来了解malloc。

malloc的全称是memory allocation，中文名称动态内存分配，当无法知道内存具体位置的时候，想要绑定真正的内存空间，就需要用到动态分配内存。

早期的Memcached内存管理是通过malloc分配的内存实现的，使用完后通过free来回收内存。这种方式容易产生内存碎片并降低操作系统对内存的管理效率。因此，也会加重操作系统内存管理器的负担，最坏的情况下，会导致操作系统比Memcached进程本身还慢，为了解决上述问题，Slab Allocator内存分配机制就诞生了。

2.Slab内存管理机制

现在的Memcached是利用Slab Allocation机制来分配和管理内存的，过程如下。

1）提前将大内存分配大小为1MB的若干个slab，然后针对每个slab再进行小对象填充，这个小对象称为chunk，避免大量重复的初始化和清理，减轻了内存管理器的负担。

Slab Allocation内存分配的原理是按照预先规定的大小，将分配给Memcached服务的内存预先分割成特定长度的内存块（chunk），再把尺寸相同的内存块（chunk）分成组（chunks slab class），这些内存块不会释放，可以重复利用（如图13-5所示）。

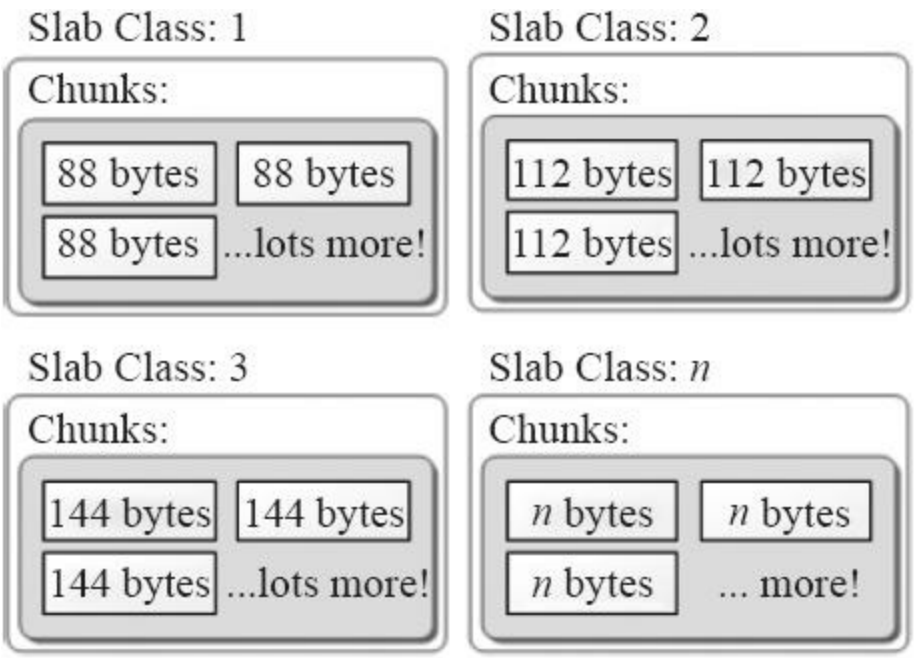


图13-5　Slab和内部的chunk图

2）新增数据对象存储时。因Memcached服务器中保存着slab内空闲chunk的列表，它会根据该列表选择chunk，然后将数据缓存于其中。当有数据存入时，Memcached根据接收到的数据大小，选择最适合数据大

小的slab（见图13-6）分配一个能存下这个数据的最小内存块（chunk）。例如：有100字节的一个数据，就会被分配存入下面112字节的一个内存块中，这样会有12字节被浪费，这部分空间就不能被使用了，这也是Slab Allocator机制的一个缺点。

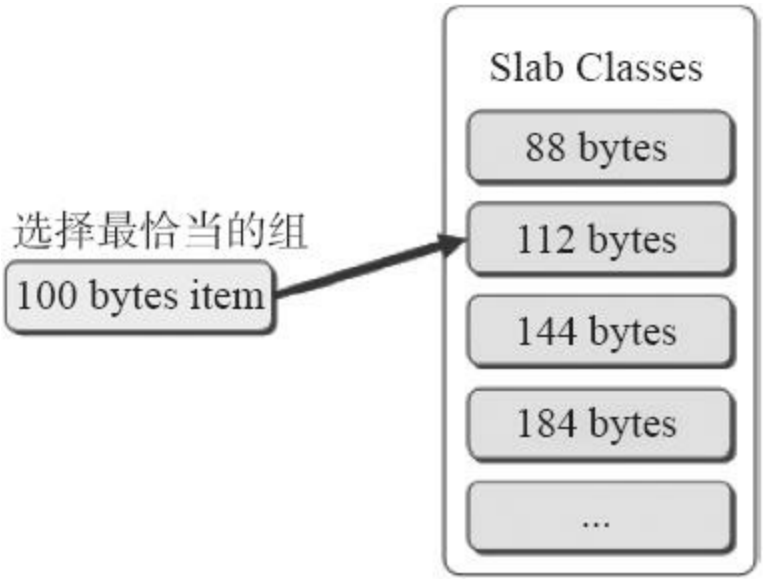


图13-6　存放数据选择chunk图

Slab Allocator还可重复使用已分配的内存，即分配到的内存不释放，而是重复利用。

3.Slab Allocation的主要术语

主要术语说明见表13-2。

表13-2　Slab Allocation的主要术语说明

Slab Allocation 主要术语	注解说明
slab class	内存区类别（48byte-1MB）
slab	动态创建的实际内存区，即分配给 Slab 的内存空间，默认是 1MB。分配给 Slab 之后根据 slab 的大小切分成 chunk。slab 默认大小为 1048576byte（1MB），大于 1MB 的数据会忽略
slab classid	Slab class 的 ID
chunk	数据区块，固定大小，chunk 初始大小，1.4 版本中是 48bytes
item	实际存储在 chunk 中的数据项

4.Slab内存管理机制特点

·提前分配大内存Slab 1MB，再进行小对象填充chunk。

·避免大量重复的初始化和清理，减轻内存管理器负担。

·避免频繁malloc/free内存分配导致的碎片。

下面对Mc的内存管理机制进行一个小结。

·Mc的早期内存管理机制为malloc（动态内存分配）。

·malloc（动态内存分配）产生内存碎片，导致操作系统性能急剧下降。

·Slab内存分配机制可以解决内存碎片的问题。

·Memcached服务的内存预先分割成特定长度的内存块，称为chunk，用于缓存数据的内存空间或内存块，相当于磁盘的block，只不过磁盘的每一个block都是相等的，而chunk只有在同一个Slab Class内才是相

等的。

·Slab Class指特定大小（1MB）的包含多个chunk的集合或组，一个Memcached包含多个Slab Class，每个Slab Class包含多个相同大小的chunk。

·Slab机制也有缺点，例如，Chunk的空间会有浪费等。

13.4.2　Memcached Slab Allocator内存管理机制的缺点

1.chunk存储item浪费空间

Slab Allocator解决了当初的内存碎片问题，但新的机制也给Memcached带来了新的问题。这个问题就是，由于分配的是特定长度的内存，因此无法有效利用分配的内存。例如，将100字节的数据缓存到128字节的chunk中，剩余的28字节就浪费了（如图13-7所示）。

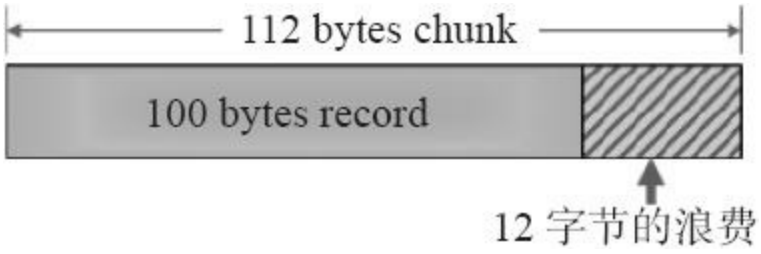


图13-7　内存存储到chunk浪费情况图解

避免浪费内存的办法是，预先计算出应用存入的数据大小，或把同一业务类型的数据存入一个Memcached服务器中，确保存入的数据大小相对均匀，这样就可以减少内存的浪费。

还有一种办法是，在启动时指定“-f”参数，能在某种程度上控制内存组之间的大小差异 。在应用中使用Memcached时，通常可以不重新设置这个参数，即使用默认值1.25进行部署即可。如果想优化Memcached

对内存的使用，可以考虑重新计算数据的预期平均长度，调整这个参数来获得合适的设置值，命令如下：

```
-f <factor>   chunk size growth factor  (

default:

 1.25) !
```

2.Slab尾部剩余空间

假设在classid=40中，两个chunk占用了1009384byte，那么就有1048576-1009384=39192byte会被浪费掉。解决办法：规划slab大小=chunk大小×n整数倍。

13.4.3 使用Growth Factor对Slab Allocator内存管理机制调优

在启动Memcached时指定Growth Factor因子（通过-f选项），就可以在某种程度上控制每组Slab之间的差异。默认值为1.25。但是，在该选项出现之前，这个因子曾经被固定为2，称为“powers of 2”策略。让我们用以前的设置，以verbose模式启动Memcached试试看：

```
# memcached -f 2 vv
```

下面是启动后的verbose输出：

```
slab class 1:


 chunk size 128 perslab 8192
slab class 2:


 chunk size 256 perslab 4096
slab class 3:


 chunk size 512 perslab 2048
slab class 4:


 chunk size 1024 perslab 1024
slab class 5:


 chunk size 2048 perslab 512
slab class 6:
```

```
 chunk size 4096 perslab 256
slab class 7:


 chunk size 8192 perslab 128
slab class 8:


 chunk size 16384 perslab 64
slab class 9:


 chunk size 32768 perslab 32
slab class 10:


 chunk size 65536 perslab 16
slab class 11:


 chunk size 131072 perslab 8
slab class 12:


 chunk size 262144 perslab 4
slab class 13:


 chunk size 524288 perslab 2
```

可见，从128字节的组开始，组的大小依次增大为原来的2倍。这样设置的问题是，Slab之间的差别比较大，有些情况下就相当浪费内存。因此，为尽量减少内存浪费，两年前追加了growth factor这个选项。

来看看现在的默认设置（f=1.25）时的输出：

```
slab class 1:
```

```
 chunk size 88 perslab 11915←


88*11915=1048520
slab class 2:


 chunk size 112 perslab 9362
slab class 3:


 chunk size 144 perslab 7281
slab class 4:


 chunk size 184 perslab 5698
slab class 5:


 chunk size 232 perslab 4519
slab class 6:


 chunk size 296 perslab 3542
slab class 7:


 chunk size 376 perslab 2788
slab class 8:


 chunk size 472 perslab 2221
slab class 9:


 chunk size 592 perslab 1771
slab class 10:


 chunk size 744 perslab 1409←


744*1409=1048520
```

此时每个Slab的大小是一样的，即1048520，1MB。组间的差距比

因子为2时小得多，可见，这个值越小，Slab中chunk size的差距就越小，内存浪费也就越小。可见，默认值1.25更适合缓存几百字节的对象。从上面的输出结果来看，可能会觉得有些计算误差，这些误差是为了保持字节数的对齐而故意设置的。

当使用Memcached或是直接使用默认值进行部署时，最好是重新计算一下数据的预期平均长度，调整growth factor，以获得最恰当的设置。内存是珍贵的资源，浪费就太可惜了。

13.4.4 Memcached的检测过期与删除机制

1.Memcached懒惰检测对象过期机制

首先要知道，Memcached不会主动检测item对象是否过期，而是在进行get操作时检查item对象是否过期以及是否应该删除！

因为不会主动检测item对象是否过期，自然也就不会释放已分配给对象的内存空间了，除非为添加的数据设定过期时间或内存缓存满了，在数据过期后，客户端不能通过key取出它的值，其存储空间将被重新利用。

Memcached使用的这种策略为懒惰检测对象过期策略，即自己不监控存入的key/value对是否过期，而是在获取key值时查看记录的时间戳（sed key flag exptime bytes），从而检查key/value对空间是否过期。这种策略不会在过期检测上浪费CPU资源。

2.Memcached懒惰删除对象机制

当删除item对象时，一般不会释放内存空间，而是做删除标记，将指针放入slot回收插槽，下次分配的时候直接使用。

Memcached在分配空间时，会优先使用已经过期的key/value对空

间；若分配的内存空间占满，Memcached就会使用LRU算法来分配空间，删除最近最少使用的key/value对，从而将其空间分配给新的key/value对。在某些情况下（完整缓存），如果不想使用LRU算法，那么可以通过“-M”参数来启动Memcached，这样，Memcached在内存耗尽时，会返回一个报错信息，如下：

```
-M      return error on memory exhausted (

rather than removing items)
```

下面针对Memcached删除机制进行一个小结。

·不主动检测item对象是否过期，而是在get时才会检查item对象是否过期以及是否应该删除。

·当删除item对象时，一般不释放内存空间，而是做删除标记，将指针放入slot回收插槽，下次分配的时候直接使用。

·当内存空间满的时候，将会根据LRU算法把最近最少使用的item对象删除。

·数据存入可以设定过期时间，但是数据过期后不会被立即删除，而是在get时检查item对象是否过期以及是否应该删除。

·如果不希望系统使用LRU算法清除数据，可以用使用-M参数。

13.5 Memcached服务安装

Memcached的安装比较简单，支持Memcached的平台常见的有Linux、FreeBSD、Solaris、Windows。这里以Centos 6.6为例进行讲解。

1.安装libevent及连接Memcached工具nc

系统安装环境如下：

```
[root@cache01 ~]# cat /etc/redhat-release
CentOS release 6.6 (


Final)



[root@cache01 ~]# uname -m
x86_64
[root@cache01 ~]# uname -r
2.6.32-504.el6.x86_64
```

安装Memcached前需要先安装libevent，有关libevent的内容在前文已经介绍，此处用yum命令安装libevent。操作命令如下：

```
yum install libevent libevent-devel nc -y
rpm -qa libevent libevent-devel nc
```

操作过程如下：

```
[root@cache01 ~]# rpm -qa libevent libevent-devel nc
libevent-1.4.13-4.el6.x86_64
```

```
[root@cache01 ~]# yum install libevent libevent-devel nc -y
[root@cache01 ~]# rpm -qa libevent libevent-devel nc
libevent-devel-1.4.13-4.el6.x86_64
nc-1.84-24.el6.x86_64
libevent-1.4.13-4.el6.x86_64
```

2.安装Memcached

操作命令如下：

```
yum install memcached -y
rpm -qa memcached
```

需要说明的是，yum安装的Memcached版本略低，但是不影响使用，如果想安装更高版本的则需要编译安装，命令集合如下：

```
wgethttp:


// memcached.org/files/memcached-1.4.24.tar.gz
tar zxf memcached-1.4.24.tar.gz
cd memcached-1.4.24
./configure
make
make install
cd ../
```

操作过程如下：

```
[root@cache01 ~]# yum install memcached -y...省略...


[root@cache01 ~]# rpm -qa memcached
memcached-1.4.4-3.el6.x86_64
```

 提示：

·建议使用yum或rpm包方式安装，它简单、易用。

·形如“memcache-2.2.7.tgz”文件名的软件为客户端源代码软件，而形如“memcached-1.4.24.tar.gz”的文件为服务器端的源代码软件。

13.6　Memcached服务的基本管理

13.6.1　启动Memcached

启动Memcached的命令如下：

```
[root@cache01 ~]# which memcached      #<==查看

Memcached命令路径

/usr/bin/memcached
[root@cache01 ~]# memcached -m 16m -p 11211 -d -u root -c 8192
                              #<==启动第一个

Memcached实例
```

现在来检查启动结果：

```
[root@cache01 ~]# lsof -i :

11211      #<==查看启动情况

COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
memcached 7224 root   26u  IPv4  70973      0t0  TCP *:

memcache （
```

```
LISTEN)


memcached 7224 root   27u  IPv6  70974      0t0  TCP *:


memcache  (


LISTEN)


memcached 7224 root   28u  IPv4  70977      0t0  UDP *:


memcache
memcached 7224 root   29u  IPv6  70978      0t0  UDP *:


memcache
[root@cache01 ~]# ps -ef|grep memcached|grep -v grep
                             #<==查看


Memcached进程


root   7224   1  0 18:


20    00:


00:


00 memcached -m 16m -p 11211 -d -u root -c 8192
```

也可同时启动多个Memcached实例，例如：再启动一个11212实例

的命令如下。

```
[root@cache01 ~]# memcached -m 16m -p 11212 -d -u root -c 8192
                                #<==启动第二个


Memcached实例



[root@cache01 ~]# ps -ef|grep memcache|grep -v grep
root   1405   1  0 20:


55    00:


00:


00 memcached -m 16m -p 11212 -d -u root -c 8192
root   1415   1  0 20:


55    00:


00:


00 memcached -m 16m -p 11211 -d -u root -c 8192
```

可把上述两个实例的启动命令放入/etc/rc.local，以便下次开机可以自启动。

```
[root@cache01 ~]# tail -2 /etc/rc.local
/usr/bin/memcached -m 16m -p 11212 -d -u root -c 8192
/usr/bin/memcached -m 16m -p 11211 -d -u root -c 8192
```

13.6.2 Memcached启动命令相关参数说明

表13-3为Memcached启动命令相关参数的说明。

表13-3 Memcached启动命令相关参数

命令参数	说　明
进程与连接设置	
-d	以守护进程（daemon）方式运行服务
-u	指定运行 Memcached 的用户，如果当前用户为 root，需要使用此参数指定用户
-l	指定 Memcached 进程监听的服务器 IP 地址，可以不设置此参数
-p（小写）	指定 Memcached 服务监听 TCP 端口号。默认为 11211
-P（大写）	设置保存 Memcached 的 pid 文件（$$），保存 PID 到指定文件
内存相关设置	
-m	指定 Memcached 服务可以缓存数据的最大内存，默认为 64MB
-M	Memcached 服务内存不够时禁止 LRU，如果内存满了会报错

（续）

命令参数	说　明
-n	为 key+value+flags 分配的最小内存空间，默认为 48 字节
-f	chunk size 增长因子，默认为 1.25
-L	启用大内存页，可以降低内存浪费，改进性能
并发连接设置	
-c	最大的并发连接数，默认是 1024
-t	线程数，默认 4。由于 Memcached 采用的是 NIO，所以太多线程作用不大
-R	每个 event 最大请求数，默认是 20
-C	禁用 CAS（可以禁止版本计数，减少开销）
调试参数	
-v	打印较少的 errors/warnings
-vv	打印非常多调试信息和错误输出到控制台，也打印客户端命令及响应
-vvv	打印极多的调试信息和错误输出，也打印内部状态转变

其他选项可通过“memcached-h”命令来显示。请读者自行阅读相关

资料。

13.6.3　向Memcached中写入数据并检查

1.Memcached中的数据形式及与MySQL相关语句对比

向Memcached中添加数据时，注意添加的数据一般为键值对的形式，例如：

key1→values1，key2→values2

这里把Memcached添加、查询、删除等的命令和MySQL数据库做一个基本类比，见表13-4。

表13-4　管理MySQL和Memcached的常见命令类比

MySQL 数据库管理	Memcached 管理
MySQL 的 insert 语句	Memcached 的 set 命令
MySQL 的 select 语句	Memcached 的 get 命令
MySQL 的 delete 语句	Memcached 的 delete 命令

2.向Memcached中写入数据实践

通过printf配合nc向Memcached中写入数据，命令如下：

```
[root@cache01 ~]# printf "set key1 0 0 6\r\noldboy\r\n"|nc 127.0.0.1 11211
STORED #<==出现


STORED表示成功添加
```

```
key1及对应的数据
```

如果set命令的字节是6，那么后面就要6个字符（字节）。否则插入数据就会不成功。示例如下：

```
[root@cache01 ~]# printf "set key2 0 0 5\r\noldboy\r\n"|nc 127.0.0.1 11211
CLIENT_ERROR bad data chunk
ERROR
```

通过printf配合nc从Memcached中读取数据，命令如下：

```
[root@cache01 ~]# printf "get key1\r\n"|nc 127.0.0.1 11211
VALUE key1 0 6
oldboy          #<==这就是读取到的


key1对应额值。
```

通过printf配合nc从Memcached中删除数据，命令如下：

```
[root@cache01 ~]# printf "delete key1\r\n"|nc 127.0.0.1 11211
DELETED          #<==出现


DELETED表示成功删除


key1及对应的数据



[root@cache01 ~]# printf "get key1\r\n"|nc 127.0.0.1 11211
END
```

 提示：推荐使用上述方法测试操作Memcached。

通过telnet命令写入数据时，具体步骤如下。

1）安装telnet工具。

```
[root@cache01 ~]# yum install telnet -y
[root@cache01 ~]# rpm -qa telnet
telnet-0.17-48.el6.x86_64
```

2）通过telnet向Memcached中写入数据。

```
[root@cache01 ~]# telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set user01 0 0 7     #<==写入数据



oldgirl
STORED
get user01           #<==浏览数据



VALUE user01 0 7
oldgirl
END
delete user01     #<==删除数据



DELETED
get user01           #<==再浏览数据，数据被删除了



END
quit
Connection closed by foreign host.
```

 提示：telnet连接后如果输入字符错了，可以通过ctrl+Backspace删除。

13.6.4　操作Memcached相关命令的语法

以下为操作Memcached的相关命令基本语法：

```
set             key1   0      0         6
<command name> <key> <flags> <exptime> <bytes>\r\n
<datablock>\r\n
<status>\r\n
```

表13-5为操作Memcached相关命令的详细说明。

表13-5　Memcached相关命令的详细说明

命　令	说　明
command name	set 无论如何都进行写入数据，会覆盖老数据 add 只有对应数据不存在时才添加数据 repalce 只有数据存在时进行替换数据 append 往后追加：append <key> datablock <status>? prepend 往前追加：prepend <key> datablock <status> cas 按版本号更改
key	普通字符串，要求小于 250 个字符，不包含空格和控制字符
flags	客户端用来标识数据格式的数值，如 json、xml、压缩等
exptime	存活时间 s，0 为永远，小于 30 天，60 × 60 × 24 × 30 为秒数，大于 30 天为 unixtime
bytes	byte 字节数，不包含 \r\n，根据长度截取存 / 取的字符串，可以是 0，即存空串
datablock	文本行，以 \r\n 结尾，当然可以包含 \r 或 \n
status	STORED/NOT_STORED/EXISTS/NOT_FOUND ERROR/CLIENT_ERROR/SERVER_ERROR 服务器端会关闭连接以修复

13.6.5 关闭Memcached

单实例关闭Memcached的方法如下：

```
[root@cache01 ~]# ps -ef|grep memcache|grep -v grep
root   1415   1  0 20:


55    00:


00:


00 memcached -m 16m -p 11211 -d -u root -c 8192
[root@cache01 ~]# killall memcached或


pkill memcached
[root@cache01 ~]# netstat -lnt|grep 11211
```

若启动了多个实例Memcached，使用killall或pkill方式就会同时关闭这些实例！因此最好在启动时增加-P参数指定固定的pid文件，这样便于管理不同的实例。示例如下：

```
[root@cache01 ~]# memcached -m 16m -p 11211 -d -u root -c 8192 -P /var/run/11211.pic
[root@cache01 ~]# memcached -m 16m -p 11212 -d -u root -c 8192 -P /var/run/11212.pic
[root@cache01 ~]# ps -ef|grep memcache|grep -v grep
root  1685  1  0 21:


54   00:


00:
```

```
00 memcached -m 16m -p 11211 -d -u root -c 8192 -P /var/run/11211.pid
root  1692  1  0 21:

54   00:

00:

00 memcached -m 16m -p 11212 -d -u root -c 8192 -P /var/run/11212.pid
```

此时，即可通过kill命令关闭Memcached。

```
[root@cache01 ~]# kill `cat /var/run/11211.pid`
[root@cache01 ~]# lsof -i :

11211
[root@cache01 ~]# lsof -i :

11212
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
memcached 1692 root   26u  IPv4  15468      0t0  TCP *:

11212 (

LISTEN)

memcached 1692 root   27u  IPv6  15469      0t0  TCP *:

11212 (

LISTEN)
```

```
memcached 1692 root   28u  IPv4  15472      0t0  UDP *:


11212
memcached 1692 root   29u  IPv6  15473      0t0  UDP *:


11212
```

关闭Memcached的方法小结如下：

```
ps -ef|grep memcached|grep -v grep|awk '{print $2}'|xargs kill
kill `cat /var/run/11211.pid`
pkill memcached
killall memcached
```

13.6.6　企业工作场景中如何配置Memcached

在企业实际工作中，一般是开发人员提出需求，说要部署一个Memcached数据缓存。运维人员在接到这个不确定的需求后，需要和开发人员深入沟通，进而确定要将内存指定为多大，或者和开发人员商量如何根据具体业务来指定内存缓存的大小。此外，还要确定业务的重要性，进而决定是否采取负载均衡、分布式缓存集群等架构，最后确定使用多大的并发连接数等。

对于运维人员，部署Memcached一般就是安装Memcached服务器端，把服务启动起来，做好监控，配好开机自启动，基本就OK了，客户端的PHP程序环境一般在安装LNMP环境时都会提前安装Memcached客户端插件，Java程序环境下，开发人员会用第三方的JAR包直接连接Memcached服务。

13.7 安装Memcached客户端

本节的内容在第7章已经详细讲解过了，为了本章节的完整性，这里简略给出。

1.LNMP PHP环境准备

这里以PHP程序为例，即为本书讲过的LNMP环境安装Memcache客户端，以blog服务虚拟机为例测试安装后的情况，首先要在LNMP环境下能出来phpinfo信息页面，只有这样才能继续操作，具体如图13-8所示。

图13-8　LNMP环境phpinfo界面

2.Memcached缓存PHP扩展插件安装

前文已经提过，Memcached分为服务器端软件和客户端插件两部分，本文是Memcached客户端PHP的扩展插件（memcache-2.2.7.tgz）在PHP环境中的安装，用于访问Memcached服务器端数据。

PHP的Memcached扩展插件下载地址为：http://pecl.php.net/package/memcache 。

PHP的Memcached客户端扩展插件安装命令集如下：

```
cd /home/oldboy/tools/
wget -q http:


// pecl.php.net/get/memcache-2.2.7.tgz
tar zxf memcache-2.2.7.tgz
cd memcache-2.2.7
/application/php/bin/phpize
./configure  --enable-memcache  --with-php-config=/application/php/bin/php-config
make
make install
cd ../
```

如果安装的是memcache-2.2.4.tgz，可能会报如下错误：

```
make:


 *** [memcache.lo] Error 1
```

解决方法为使用如下命令：

```
cp memcache.loT memcache.lo
```

整个操作过程如下：

```
[root@www ~]# cd /home/oldboy/tools/
[root@www tools]# wget -q http:



// pecl.php.net/get/memcache-2.2.7.tgz
[root@www tools]# ls -l memcache-2.2.7.tgz
-rw-r--r-- 1 root root 36459 Mar 30 15:



05 memcache-2.2.7.tgz
[root@www tools]# tar zxf memcache-2.2.7.tgz
[root@www tools]# cd memcache-2.2.7
[root@www memcache-2.2.7]# /application/php/bin/phpize
Configuring for:



PHP Api Version:



         20090626
Zend Module Api No:



      20090626
Zend Extension Api No:



   220090626
[root@www memcache-2.2.7]# ./configure  --enable-memcache  --with-php-config=/applic
[root@www memcache-2.2.7]# make
...skip...
Build complete.
Don't forget to run 'make test'.
[root@www memcache-2.2.7]# make install
Installing shared extensions:



 /application/php5.3.27/lib/php/extensions/no-debug-non-zts-20090626/
[root@www memcache-2.2.7]# cd ..
```

```
[root@www tools]# ls -l /application/php5.3.27/lib/php/extensions/no-debug-non-zts-2
-rwxr-xr-x 1 root root 246728 May  1 15:


51 memcache.so
#<==最后生成了


memcache.so模块就表示


memcache扩展插件成功安装
```

3.配置Memcache客户端，使其生效

修改PHP的配置文件php.ini，加入Memcache客户端的配置，命令如下。

```
[root@www memcache-2.2.7]# cd /application/php/lib/
[root@www lib]# vi php.ini
```

把如下两行内容增加到php.ini文件结尾：

```
extension_dir = "/application/php5.3.27/lib/php/extensions/no-debug-non-zts-2009062(
extension=memcache.so
 [root@www lib]# tail -2 php.ini
extension_dir = "/application/php5.3.27/lib/php/extensions/no-debug-non-zts-2009062(
extension = memcache.so
```

提示：这部分内容在本书第7章已经详细讲解过了，此处不再重复！

4.重启php fpm服务使PHP的配置修改生效

1）检查php-fpm语法：

```
[root@www lib]# /application/php/sbin/php-fpm -t
[26-Aug-2015 11:

09:

30] NOTICE:

 configuration file /application/php5.3.27/etc/php-fpm.conf test is successful
```

2）重启fpm，命令如下：

```
[root@www lib]# pkill php-fpm
[root@www lib]# ps -ef|grep php-fpm|grep -v grep
[root@www lib]# /application/php/sbin/php-fpm
[root@www lib]# ps -ef|grep php-fpm|grep -v grep
root 6791 1  0 11:

13    00:

00:

00 php-fpm:

 master process (

/application/php5.3.27/etc/php-fpm.conf)

nginx     6792  6791  0 11:

13         00:
```

```
00:

00 php-fpm:

 pool www
```

3）打开浏览器访问phpinfo页面，若出现如图13-9所示的内容，则表示Memcache客户端安装成功。

5.编写测试Memcached服务的PHP脚本

下面为简单PHP程序连接Memcached测试脚本。

```
[root@www blog]# cat -n op_mem.php
    1  <php
    2      $memcache = new Memcache;


    3      $memcache->connect (

'10.0.6.31',

 11211)

 or die  (

"Could not connect Mc server") ;


    4      $memcache->set (
```

```
'key',
 'oldboy book');
    5       $get= $memcache->get (
'key');
    6       echo $get;
    7   >
```

图13-9　Memcache客户端安装配置成功

说明：

第一行：

PHP开始标识

第二行：创建一个

Memcache对象

第三行：连接

Memcached服务器

第四行：设置一个变量到内存中，名称是

key值是

oldboy book第五

-六行：从内存中取出

key值

第七行：

PHP结束标识

测试结果如下：

```
[root@www blog]# /application/php/bin/php op_mem.php
oldboy book
```

出现上面的提示就表示LNMP环境连接Memcached服务成功。

13.8 Memcached应用管理

13.8.1 通过命令管理Memcached

运维人员一般可通过在命令行执行“telnet ip port”的方式登录到Memcached，然后执行一些管理的命令，除了telnet外，nc命令也是一个不错的管理Memcached服务的命令！

有关“telnet ip port”命令前文已经介绍过了，下面将重点讲解通过nc管理及监控Memcached的一些常见操作。

以下通过脚本模拟用户插入及删除数据来监控Memcached服务是否正常的示例。

（1）Memcached服务是否异常的监控脚本

脚本内容如下：

```
[root@cache01 ~]# cat mon_mc.sh
#!


/bin/sh
export MemcachedIp=$1
export MemcachedPort=$2
export NcCmd="nc $MemcachedIp $MemcachedPort"
export MD5=3fe396c01f03425cb5e2da8186eb090d
USAGE ()
```

```
{
        echo "$0 MemcachedIp MemcachedPort"
        exit 3
}
[ $# -ne 2 ] && USAGE
printf "set $MD5 0 0 6\r\noldboy\r\n"|$NcCmd >/dev/null 2>&1
if [ $ -eq 0 ];



then
    if [ `printf "get $MD5\r\n"|$NcCmd|grep oldboy|wc -l` -eq 1 ];



then
        echo "Memcached status is ok"
        printf "delete $MD5\r\n"|$NcCmd >/dev/null 2>&1
        exit 0
    else
        echo "Memcached status is error1"
        exit 2
    fi
else
        echo "Could not connect Mc server"
        exit 2
fi
```

Memcached服务正常的情况下，测试检验脚本：

```
[root@cache01 ~]# sh mon_mc.sh 127.0.0.1 11211
Memcached status is ok
```

然后关闭Memcached服务，测试检验脚本：

```
[root@cache01 ~]# pkill memcached
[root@cache01 ~]# lsof -i :



11211
[root@cache01 ~]# sh mon_mc.sh 127.0.0.1 11211
Could not connect Mc server
```

最后开启Memcached服务，测试检验脚本：

```
[root@cache01 ~]# memcached -m 16m -p 11211 -d -u root -c 8192 -P /var/run/11211.pid
[root@cache01 ~]# sh mon_mc.sh 127.0.0.1 11211
Memcached status is ok
```

通过上面的测试，我们发现使用监控脚本监控Memcached服务是否异常的运行是正确的。

（2）通过nc命令查看Memcached服务的运行状态信息

查看Memcached服务运行状态信息的nc命令如下：

```
[root@cache01 ~]# printf "stats\r\n"|nc 127.0.0.1 11211
STAT pid 28123
STAT uptime 20063728
STAT time 1440577412
STAT version 1.4.4
STAT libevent 1.4.13-stable
STAT pointer_size 64
STAT rusage_user 645314.961214
STAT rusage_system 1134305.468357
STAT curr_connections 1361
STAT total_connections 18299935
STAT connection_structures 12455
STAT reserved_fds 20
STAT cmd_get 60424570825
STAT cmd_set 854196259
STAT cmd_flush 0
STAT cmd_touch 0
STAT get_hits 58105159197
STAT get_misses 2319411628
STAT delete_misses 258
STAT delete_hits 8803
STAT incr_misses 0
STAT incr_hits 0
STAT decr_misses 0
STAT decr_hits 0
STAT cas_misses 0
STAT cas_hits 0
STAT cas_badval 0
STAT touch_hits 0
STAT touch_misses 0
STAT auth_cmds 0
STAT auth_errors 0
STAT bytes_read 5969739043455
STAT bytes_written 140330591248204
STAT limit_maxbytes 8489271296
STAT accepting_conns 1
STAT listen_disabled_num 0
STAT threads 4
STAT conn_yields 0
```

```
STAT hash_power_level 25
STAT hash_bytes 268435456
STAT hash_is_expanding 0
STAT malloc_fails 0
STAT bytes 3680105052
STAT curr_items 24628253
STAT total_items 854196259
STAT expired_unfetched 271346565
STAT evicted_unfetched 7503
STAT evictions 7509
STAT reclaimed 336538393
STAT crawler_reclaimed 0
END
```

除了上述输出的全部状态以外，Memcached还支持通过以下命令的输入来查看Memcached的部分运行状态信息。目前管理Memcached的命令见表13-6。

表13-6　Memcached状态命令的说明

Memcached 状态命令	说　　明
stats	统计 Memcached 的各种信息
stats settings	查看一些 memcached 的设置信息，例如：线程数
stats slabs	查看 slabs 相关情况，例如：chunksize 长度
stats items	查看 items 相关情况
stats sizes	查看 items 个数和大小
stats reset	清理统计数据

例如，要查看Memcached的统计信息，可先执行“telnet ip监听端口”命令，登录成功之后执行stats命令，具体过程如下：

```
[root@nginx ~]# telnet 10.0.0.17 11211
Trying 10.0.0.17...
Connected to web181 (


10.0.0.17)


. Escape character is '^]'.
```

```
stats
STAT pid 19900                #Memcached 启动的进程


 id
STAT uptime 2115676           #到目前为止启动了多少秒



STAT time 1287937993
STAT version 1.4.5           #Memcached 的版本信息



STAT pointer_size 64
STAT rusage_user 0.003999
STAT rusage_system 0.001999
STAT curr_connections 10           #当前的并发连接数



STAT total_connections 16     #总的连接数



STAT connection_structures 11
STAT cmd_get 1                #执行的


 get 命令的次数



STAT cmd_set 4                #执行的


 set 命令的次数



STAT cmd_flush 0               #执行


 flush 命令的次数



STAT get_hits 1               #get 的命中数
```

```
STAT get_misses 0                #get 的非命中数



STAT delete_misses 0
STAT delete_hits 0
STAT incr_misses 0
STAT incr_hits 0
STAT decr_misses 0
STAT decr_hits 0
STAT cas_misses 0
STAT cas_hits 0
STAT cas_badval 0
STAT auth_cmds 0
STAT auth_errors 0
STAT bytes_read 157
STAT bytes_written 129
STAT limit_maxbytes 33554432      #允许使用的最大内存容量



STAT accepting_conns 1
STAT listen_disabled_num 0
STAT threads 4
STAT conn_yields 0
STAT bytes 149
STAT curr_items 2
STAT total_items 2
STAT evictions 0
STAT reclaimed 0
END
```

使用printf及nc命令获取状态信息更佳，因为无需交互，可以使用以下脚本批量操作。

```
[root@cache01 ~]# printf "stats items\r\n"|nc 127.0.0.1 11211
STAT items:



1:



number 2
STAT items:



1:
```

```
age 4439
STAT items:


1:


evicted 0
STAT items:


1:


evicted_nonzero 0
STAT items:


1:


evicted_time 0
STAT items:


1:


outofmemory 0
STAT items:


1:


tailrepairs 0
END
```

13.8.2 Memcached状态信息详细说明

表13-7为Memcached状态信息的详细说明，读者可以通过该表来了解相关信息。

表13-7 Memcached状态信息的详细说明

参 数	值	描 述	实际作用
pid	28123	Memcached 服务进程 ID	查看服务信息
uptime	20063728	服务已运行秒数	
time	1440577412	服务当前 UNIX 时间戳	
version	1.4.4	Memcached 版本	
libevent	1.4.13-stable	libevent 版本	
pointer_size	64	操作系统指针大小	

（续）

参　数	值	描　述	实际作用
rusage_user	645314.961214	进程累计用户时间	分析占用 CPU 的情况
rusage_system	1134305.468357	进程累计系统时间	
curr_connections	1361	当前连接数	分析连接数情况
total_connections	18299935	Memcached 运行以来连接总数	
connection_structures	12455	Memcached 分配的连接结构数量	
reserved_fds	20	内部使用的 FD 数	
cmd_get	60424570825	get 命令请求次数	分析命令率情况
get_hits	58105159197	get 命令命中次数	
get_misses	2319411628	get 命令未命中次数	
cmd_set	854196259	set 命令请求次数	
cmd_flush	0	flush 命令请求次数	
cmd_touch	0	touch 命令请求次数	
delete_misses	258	delete 命令未命中次数	
delete_hits	8803	delete 命令命中次数	
incr_misses	0	incr 命令未命中次数	
incr_hits	0	incr 命令命中次数	
decr_misses	0	decr 命令未命中次数	
decr_hits	0	decr 命令命中次数	
cas_misses	0	cas 命令未命中次数	
cas_hits	0	cas 命令命中次数	
cas_badval	0	使用擦拭次数	
touch_hits	0	touch 命令命中次数	
touch_misses	0	touch 命令未命中次数	
auth_cmds	0	认证命令处理的次数	
auth_errors	0	认证失败数目	
bytes	3680105052	已过期但未获取的对象数目	分析字节数情况
bytes_read	5969739043455	读取总字节数	
bytes_written	140330591248204	发送总字节数	
limit_maxbytes	8489271296	分配的内存总大小（字节）	
accepting_conns	1	接受新的连接	
listen_disabled_num	0	失效的监听数	
threads	4	当前线程数	
conn_yields	0	连接操作主动放弃数目	
hash_power_level	25	hash 表等级	

（续）

参　数	值	描　述	实际作用
hash_bytes	268435456	当前 hash 表大小	
hash_is_expanding	0	hash 表正在扩展	
malloc_fails	0	分配内存失败数目	
curr_items	24628253	当前的对象数目	分析对象数 LRU 频率
total_items	854196259	当前存储占用的字节数	
evictions	7509	LRU 释放的对象数目	
expired_unfetched	271346565	当前存储的数据总数	
evicted_unfetched	7503	启动以来存储的数据总数	
reclaimed	336538393	已过期回收的数据条目	
crawler_reclaimed	0	爬虫回收的数目	

13.8.3　通过memadmin php工具展示Memcached状态信息

运维人员除了通过命令行管理Memcached以外，还可以使用很多的第三方开源管理工具，例如：通过memadmin工具管理及监控Memcached状态，软件的名称为“memadmin-1.0.12.tar.gz”，这个软件的部署十分简单，功能非常强大，但是依赖于PHP环境，推荐初学者使用。

1.部署memadmin php工具

因为这个软件是基于PHP程序的，因此，需要有PHP的环境才行，本章已经准备好了LNMP的环境，因此，可以简单地将上述“memadmin-1.0.12.tar.gz”解压到虚拟主机站点目录下，本文还是以blog虚拟主机为例进行讲解。

部署的关键命令集合如下：

```
[root@www tools]# pwd
/home/oldboy/tools
[root@www tools]# ls -l memadmin-1.0.12.tar.gz
-rw-r--r-- 1 root root 196734 12月


 23 2014 memadmin-1.0.12.tar.gz
[root@www tools]# tar xf memadmin-1.0.12.tar.gz
[root@www tools]# mv memadmin /application/nginx/html/blog/
```

2.采用IP或者解析好的域名进行访问

假设具体的浏览地址为：http://blog.etiantian.org/memadmin/ ，已将解析记录“10.0.0.8 www.etiantian.org bbs.etiantian.org blog.etiantian.org”放入C:\Windows\System32\drivers\etc\hosts里，其中10.0.0.8为LNMP服务器的IP。现在来看看memadmin是如何管理Memcached服务的。

首先，打开登录前的管理页面，如图13-10所示。

登录之后的页面如图13-11所示。

图13-10　memadmin登录前的界面

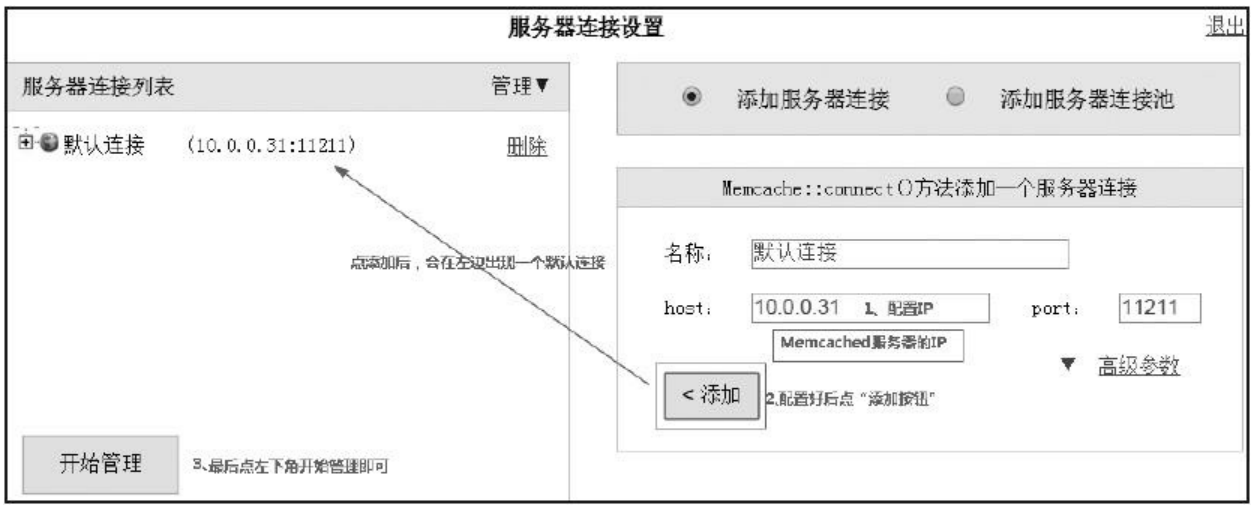


图13-11　memadmin登录后的界面

然后单击“开始管理”，进入如图13-12~图13-14所示的界面。

图13-12　memadmin基本管理界面

图13-13　memadmin数据项统计界面

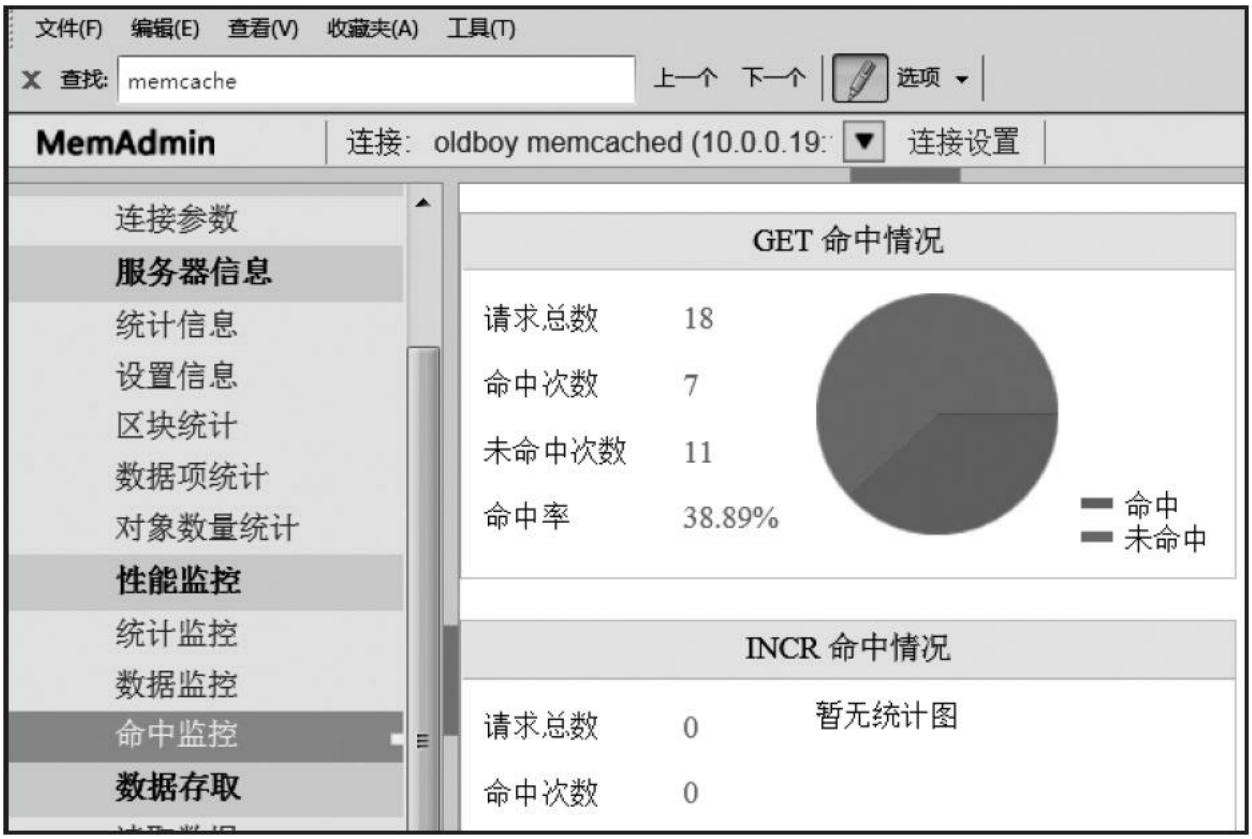


图13-14　memadmin命中率监控界面

memadmin对单个或少量的Memcached服务器数量的监控、维护管理还是很直观容易的。

13.9 Memcached服务应用的优化

13.9.1 Memcached服务应用优化案例

下面给大家介绍一个Memcached服务应用优化的企业案例。用户访问网站打开页面很慢，经运维人员排查后，发现MySQL数据库负载很高，load值大概为20~30，如下：

```
[root@oldboy ~]# uptime
 10:

41:

01 up 15:

02,

  4 users,

  load average:

 20,

 15,

 10
```

登录数据库后，使用“show full processlist；”查看，或者在Linux命令行使用下面的命令查看：

```
mysql -uroot -p'1111' -e "show full processlist"|grep -vi sleep
```

发现数据库中像“LIKE'%杜冷丁%'”这样的SQL语句特别多，导致数据库负载很高，我们都知道像LIKE'%杜冷丁%'这样的SQL语句对于MySQL数据库来说，利用索引没有太大的优化余地。

打开该网站首页查看，发现该首页有一个搜索框，根据前面查看的SQL语句形式，可以推测上述“LIKE'%杜冷丁%'”应该是搜索框的语句带来的结果（如图13-15所示）。事后从这个公司的数据库维护人员处得到了证实，请问如果是你，你该如何优化解决当前的问题呢？

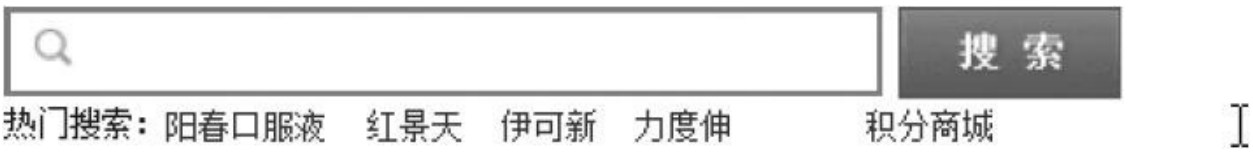


图13-15　网站首页搜索框

当时老男孩给这家公司的优化方案思路如下：

1）看是否可以从业务上整改，例如，只有在用户后登录后才可以进行搜索，通过这种改进来减少搜索的次数，达到减轻数据库服务压力的目的。

2）如果有大量频繁的搜索SQL语句，很有可能是有网络爬虫在爬我们的网站，可以通过分析Web日志或者网络连接状态，封掉这些非正常的搜索请求。

3）为主库配置多个从库，然后实现数据的读写分离，让“LIKE'%杜冷丁%'”这样的查询去多个从库查，从而减轻主库的读写压力。

4）在数据库前端加Memcached缓存服务器，这个效果在所有的方法里是最好的。

5）在数据库里使用“LIKE'%杜冷丁%'”实现搜索，并非明智的选择，可以通过Sphinx等搜索服务实现用户搜索。

6）还可以利用C、Ruby等开发语言开发程序，部署计算服务器实现每日读数据库计算全量搜索索引，然后保存在提供搜索的Web服务器上，除了设定每日计算全量搜索索引外，可以每分钟读单独的1~2个从库做增量计算索引。这是大公司针对站内搜索采取的基本解决方案之一。

其中1）、2）、3）是短期的方案，简单，易实施。4）、5）、6）是长期的方案目标。

这个案例的最终问题是，网站前端有爬虫爬站，通过分析Web日志获取到爬站的IP，临时封掉后，MySQL数据库负载立刻下降了很多，网

站服务恢复正常。

图13-16是针对方案6给出的大型网站搜索集群架构逻辑案例图。

13.9.2 Memcached服务优化策略

Memcached服务的安装配置简单，几乎不需要优化就可以跑得很好，在特殊的大并发高访问量的场合才需要进行一些优化。

1.提高Memcached访问命中率是优化最关键的指标

例如：每次新增数据到数据库的同时，就将数据写入或者复制一份到Memcached里，然后从业务逻辑上让程序优先读缓存，没有数据再查数据库。

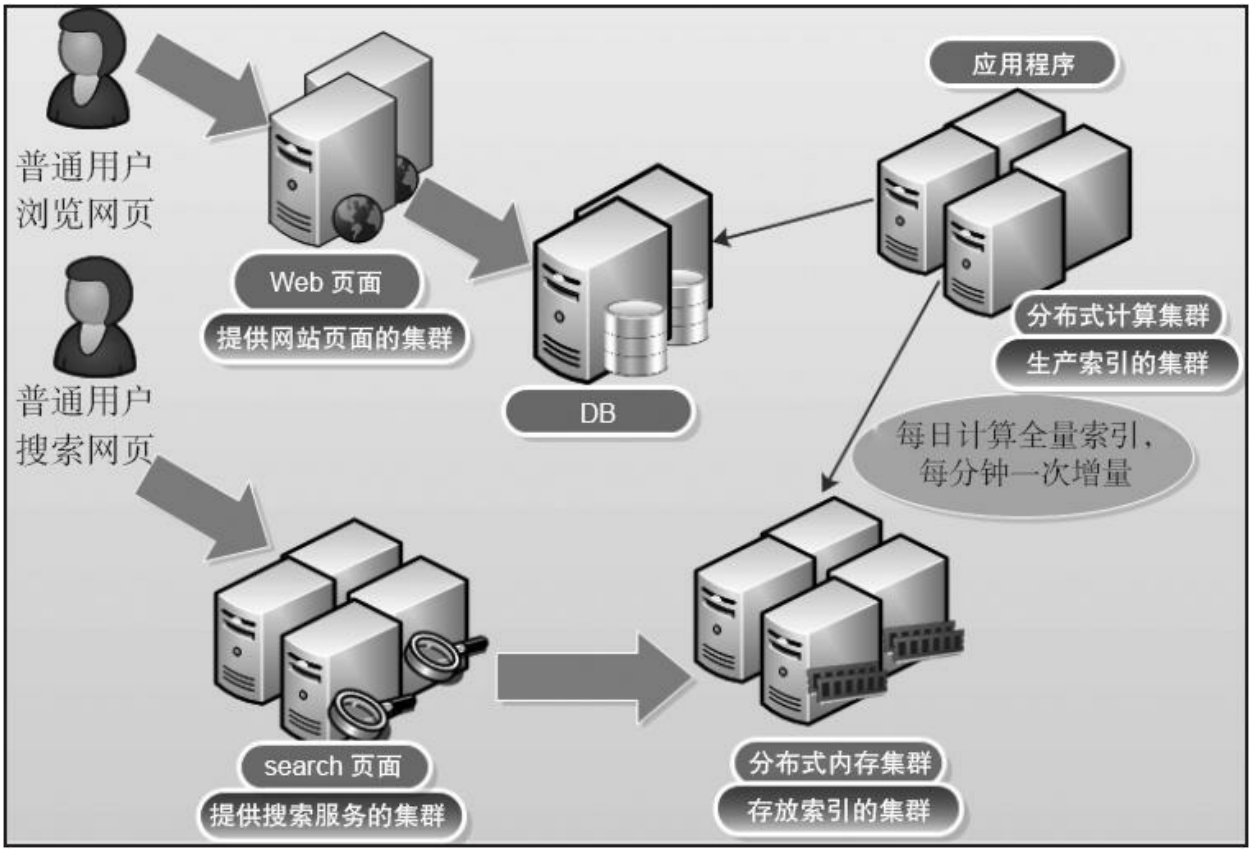


图13-16　大型网站搜索集群架构逻辑案例图

2.提高内存利用率，减少内存浪费

·减少chunk内存空间浪费的调优方法为，根据业务数据的大小，利用-n参数设定chunk的初始值，及通过-f参数factor增长因子设置chunk的大小尽可能接近业务数据的大小。

·减少slab的浪费，设定slab的大小为chunk整数倍。

·采用一致性哈希分布式缓存集群架构。

当网站后端的数据库数据量很大时，单台Memcached就无法存放绝大部分的数据库内数据，导致Memcached服务的命中率很低，此时，可以采用一致性哈希分布式缓存集群架构，提升网站的命中率，一致性哈希可以由程序实现或者支持一致性哈希算法的负载均衡器实现。

13.9.3　Memcached服务在大型站点中的架构优化

1.大型网站的架构设计原则

当访问量增大时，在整个网站集群架构中最先出现瓶颈的几乎都是后端的数据库或用于存储的服务器，在企业生产工作中应尽量把用户的访问请求往整个网站架构的最前面推，即当用户请求数据时，越是在靠近用户端返回数据，效果就越好。具体见老男孩的千万级PV/IP规模高性能高并发网站架构的博

文：http://oldboy.blog.51cto.com/2561410/736710 。

2.大型网站的数据库架构常见设计

为了缓解应用对数据库的高并发访问压力，大型网站都会在数据库层配置数据读写分离，并提供读的数据库做负载均衡，逻辑图示例如图13-17所示，其中是以Amoeba软件作为读写分离说明的。

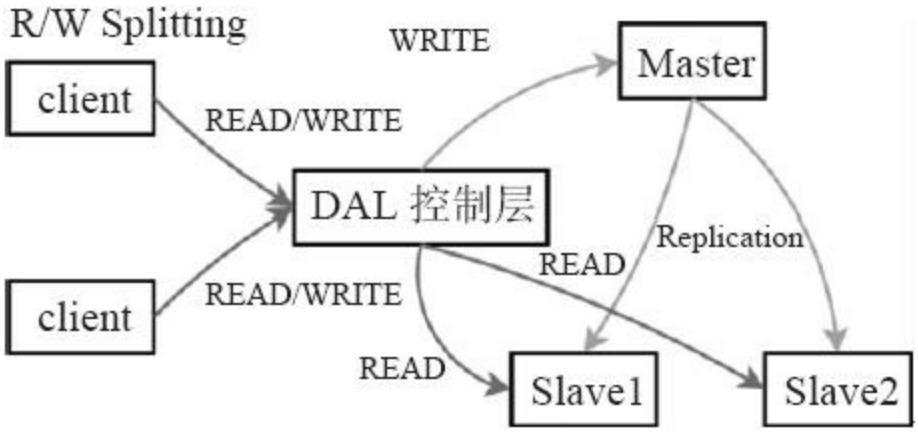


图13-17　数据库架构设计逻辑简图

由于访问数据库读写的是磁盘，所以，对于高并发的业务场景，上述读写分离的架构还是远远解决不了问题的，根据离访问用户越近效率越高以及用内存代替磁盘的架构思想，可以在数据库架构的前端部署Memcached服务作为缓存区，最大限度地把对数据库查询信息保存在Memcached服务的内存中，这样前端的应用服务就能够迅速地从Memcached中读取到原本在数据库中才能读取到的数据，从而加快了网站的访问速度。

3.分布式Memcached缓存服务架构

由于单台Memcached服务器的内存容量是有限的，并且单台也存在单点故障，因此，大型网站往往会将多个Memcached服务器组合成集群提供服务，那么怎么组合效率才更高呢？

（1）缓存服务器使用常规负载均衡模式的问题

在常规负载均衡模式中，集群节点上的程序和数据都是一样的，例如Web集群。而缓存集群设计下的所有节点缓存的数据就不能都一样，因为这样一来，访问数据命中率会非常低下，低下的原因主要有两个：

·当客户端访问缓存A无数据时，就会去查后端数据库，然后把查到的数据放入缓存A中一份，当下次客户端访问相同的数据时，可能被分

配到的是缓存B，结果B中还是无数据，这样客户端又会去查后端数据库，这样就会给数据库造成了访问压力，同时造成缓存服务器的命中率很低。

·在集群运行一段时间后，所有缓存的数据可能交叉并极其接近，如果数据库数据量远大于单台Memcached服务器的内存总量，Memcached的命中率也会非常低下。理想的情况是所有缓存数据之和接近数据库总的数据容量，这样缓存的效率才会高。

解决上述问题的方案最常见的就是使用哈希算法或一致性哈希算法将需要同样数据的请求始终调度到同一台缓存服务器，提升访问命中率。其次，所有缓存服务器缓存的数据都是不同的，所有服务器缓存的数据之和接近数据库总的数据容量，使得缓存集群无需换入换出，达到更高的命中率。

下面的图13-18和图13-19是缓存服务器使用常规负载均衡模式的问题图解。

可以看到，后端节点的缓存服务器越多，缓存的命中率越低，数据库的压力就越大。

（2）分布式缓存集群的优劣势

Memcached支持分布式集群，其中的方法之一就是在客户端应用程

序上进行改造。例如：可以根据key适当进行有规律的封装。比如以用户为主的网站来说，每个用户都有UserID，那么可以按照固定的ID来进行提取和存取，假设以1开头的用户保存在第一台Memcached服务器上，以2开头的用户的数据保存在第二台Memcached服务器上，那么存取数据时都会按照User ID来进行相应的转换和存取。

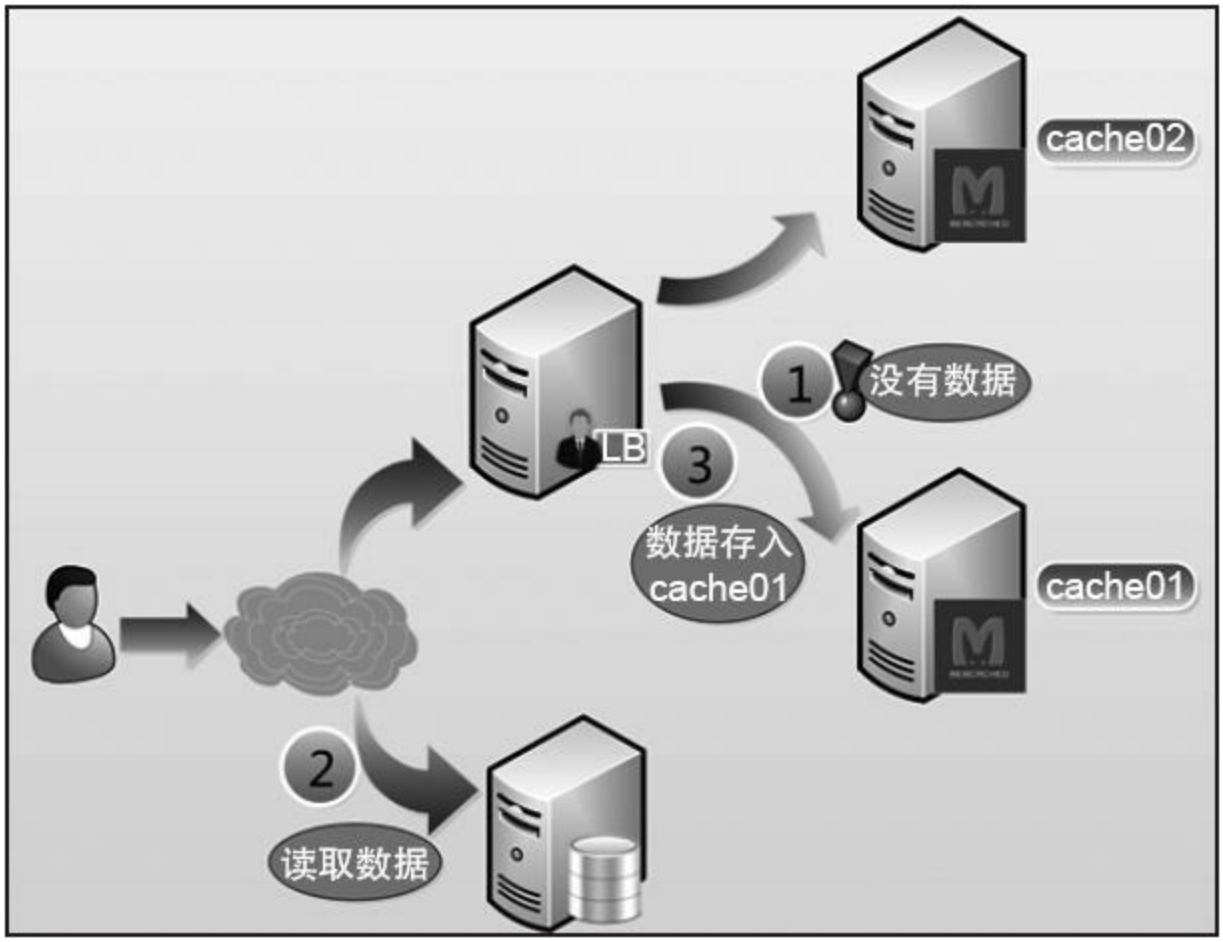


图13-18　缓存使用常规负载均衡模式问题第一次访问

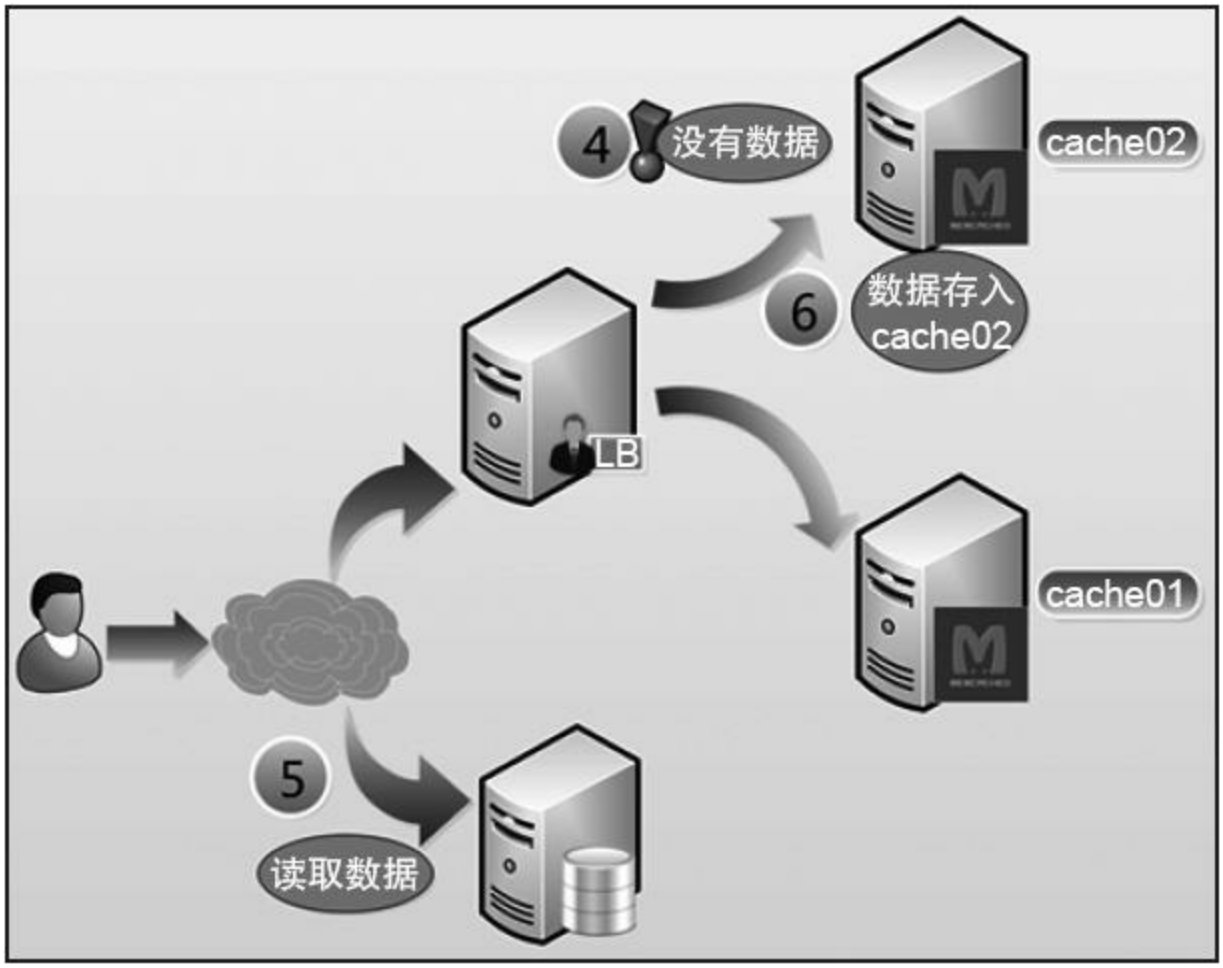


图13-19　缓存使用常规负载均衡模式问题第二次访问

但是这样也有缺点，就是需要对User ID进行判断，如果业务不一致，或者是其他类型的应用，可能不是那么合适，此时可以根据自己的实际业务来进行考虑，或者去想更合适的方法。

此外，还可以在应用服务器上通过程序用哈希算法或一致性哈希算法调度Memcached服务器，所有Memcached服务器的地址池可以简单地配在每个程序的配置文件里。并且可在负载均衡器上使用哈希算法或一致性哈希算法，不过此时需要负载均衡器的支持。

4.分布式缓存集群设计思想

1）每一台Memcached服务器的内容都是不一样的。这些Memcached服务器缓存的内容加起来接近整个数据库的数据容量。

2）通过在客户端程序或者Memcached的负载均衡器上用hash算法，让同一数据内容都分配到一个Memcached服务器。

3）普通的hash算法对于节点宕机会带来大量的缓存数据流动（失效），可能会引起雪崩效应。

4）一致性哈希算法（还可以带虚拟节点）可以让缓存节点宕机对节点的数据流动（失效）降到最低。

5.分布式Memcached缓存集群的调度算法

（1）取模计算hash

优点：简单、分散性优秀

缺点：添加/移除服务器时，缓存重组代价巨大，影响命中率。

例：将26个字母缓存到了3个节点，此时新增加一个节点，访问命中率将下降23%（如图13-20所示）。

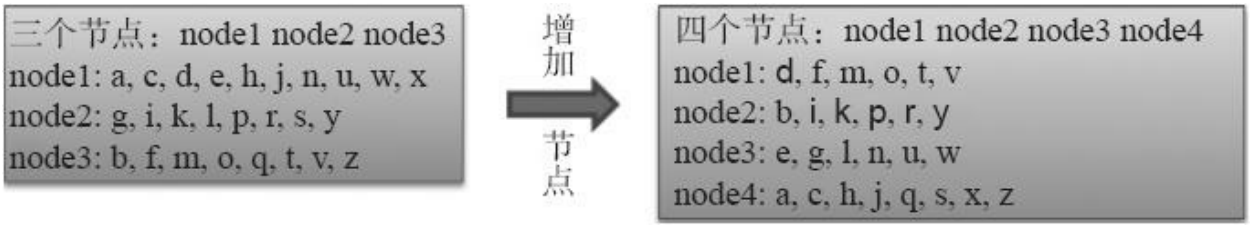


图13-20　增加节点后缓存服务器内容的变化情况

（2）Consistent hash（一致性哈希算法）

一致性哈希算法（consistent hash）是一种特殊的hash算法，简单地说，在移除以及添加一个cache节点时，它能够尽可能小地改变已存在key的映射关系，让缓存服务器缓存的内容受到的影响最小。

consistent hash算法原理如图13-21所示，首先想象一个0~2^{32} -1次方的数值空间，将这个空间想象成一个首（0）尾（2^{32} -1）相接的圆环，然后算出不同Memcached服务器节点的哈希值（0~（2^{32} -1）之间），将这些值分散到上述的圆环上，接着用同样的方法算出存储不同数据的键的哈希值并映射到相同的圆环上，最后从数据映射到的位置开始顺时针查找，将键对应的数据保存到查找到的第一个服务器上，如图13-21所示。

添加服务器node5时如图13-22所示。

可以看到添加node5服务器后，缓存的数据只影响node2到node5之间的数据范围，即node4的一部分数据，缓存到了node5，其他的缓存服务器没有受到影响，当移除服务器时，原理和添加服务器时一样，不再

赘述。

这就是一致性哈希算法的作用，可以最大限度地减少键的重新分布。该算法的实现方法还可以采用更好的虚拟节点的策略思路，一致性哈希算法可以通过在前端使用程序实现或者通过负载均衡器实现。

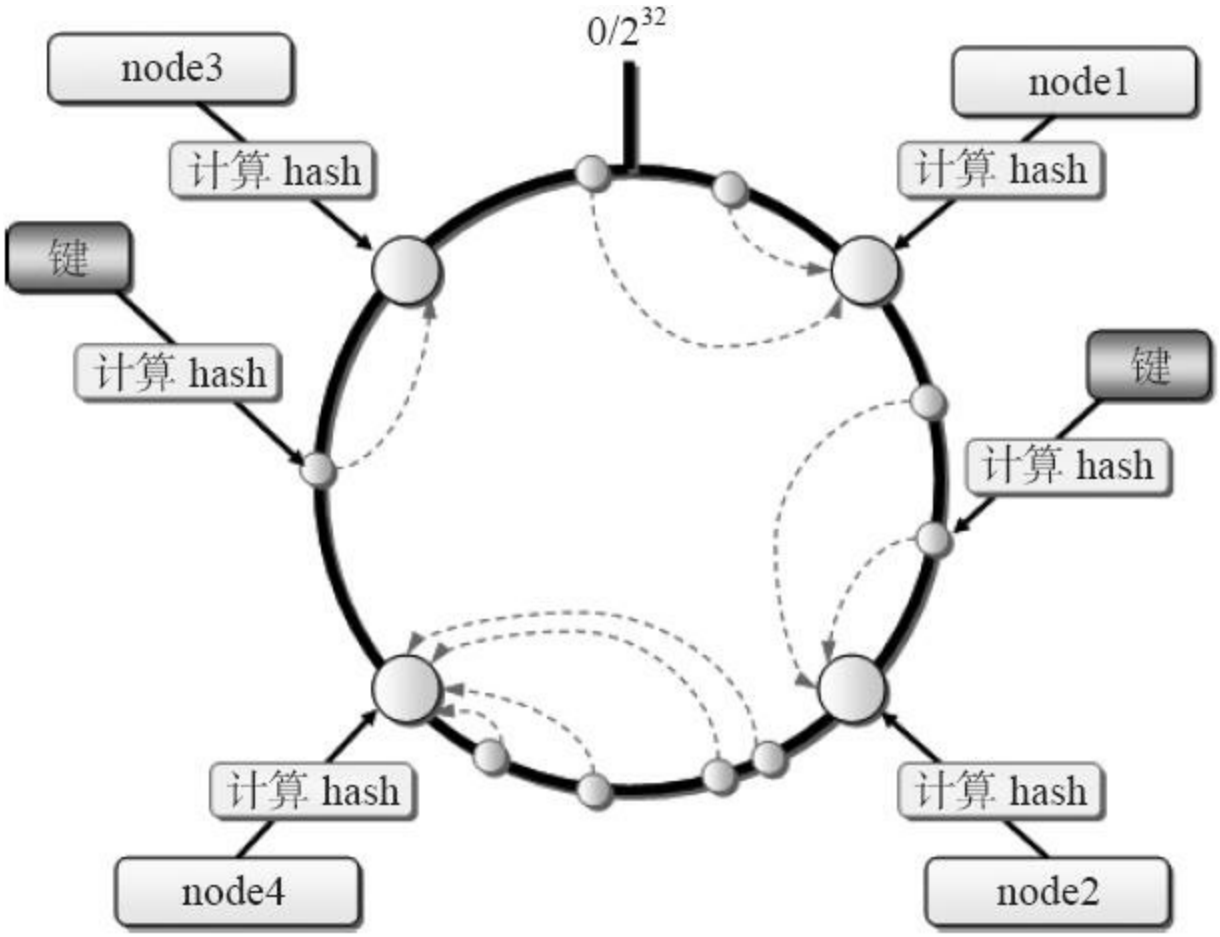


图13-21　环形哈希空间想象图

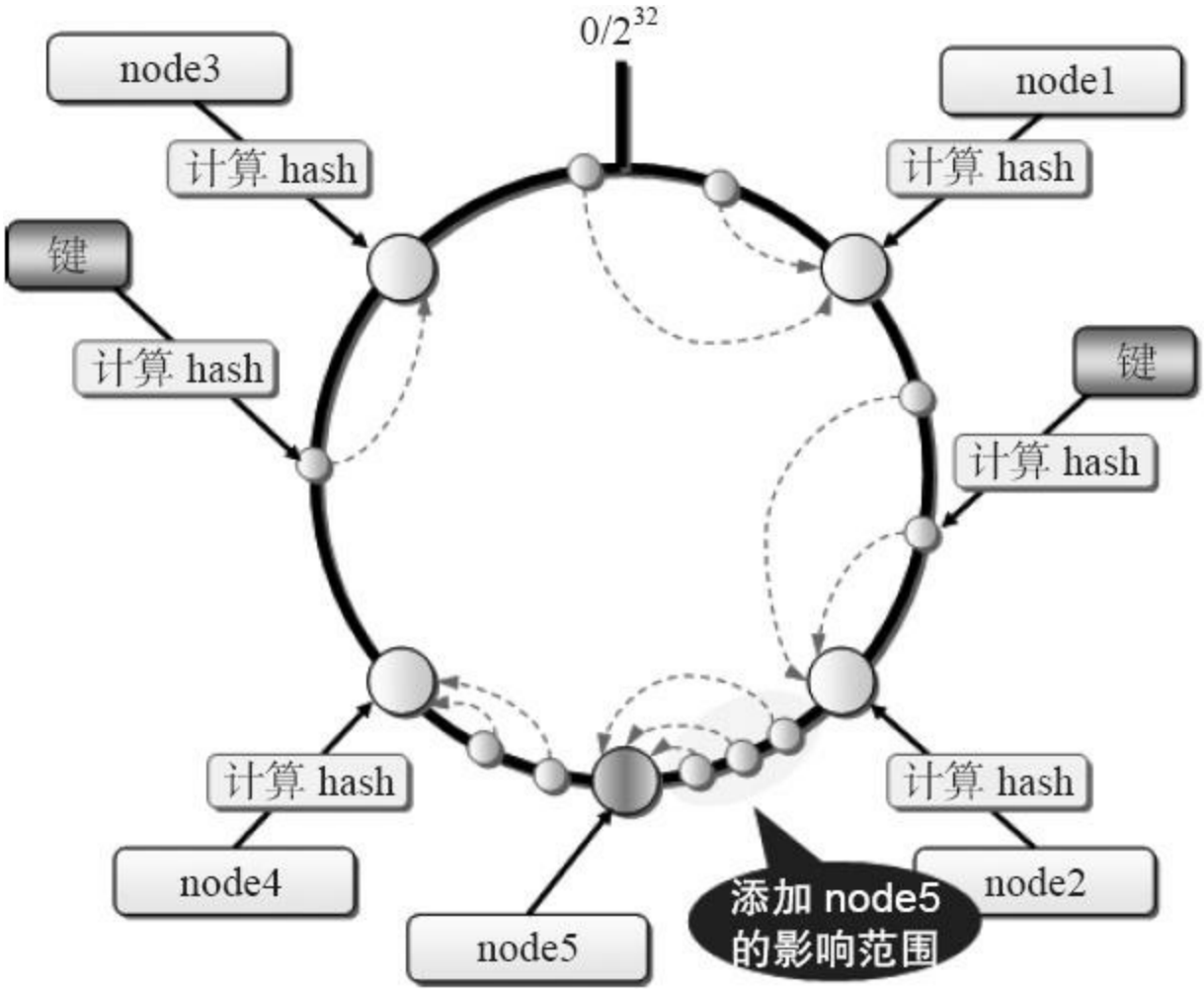


图13-22　添加node5后的环形哈希空间想象图

以下是一致性哈希算法的PHP测试代码片段：

```
<php
header (


'Content-type:


 text/html;


 charset=utf8') ;
```

```
# 抽象接口



interface hash{
    public function _hash（


$str）；



}
interface distribution {
    public function lookup（


$key）；



}
# hash算法实例



class Consistent implements hash，


distribution {
    protected $point_num = 64；



    protected $posi = array（）；


    protected $server；


# 计算一个


hash值
```

```
    public function _hash(
$str)
{
        return sprintf(
'%u',
crc32(
$str));
    }
    # 计算
key分布到的服务器
    public function lookup(
$key)
{
        foreach(
$this->posi as $k=>$v)
{
            if(
```

```
 $this->_hash(
$key)
 <= $k )
{
                $this->server = $v;
                break;
            }
        }
        return $this->server;
    }
    # 添加服务器节点
    public function addServer(
$server)
{
        for(
$i=1;
$i<=$this->point_num;
$i++)
```

```
{
            $this->posi[$this->_hash(
$server.'_'.$i)
] = $server;

        }
        $this->sortPosi();

    }
    # 排序定位点

    public function sortPosi()
{
        ksort(
$this->posi);

    }
 # 仅仅用来测试

=====================================
    # 打印定位点

    public function printPosi()
{
        echo '<pre>';
```

```
        print_r (

$this->posi) ;



    }
}
$hash = new Consistent () ;



$hash->addServer (

'a') ;



$hash->addServer (

'b') ;



$hash->addServer (

'c') ;



# test hash
$key = 'abc';



$server = $hash->lookup (

$key) ;



echo $key.'对应的服务器是:
```

```
'.$server.'  

  对应的

hash值是:

'.$hash->_hash（

$key）;

echo '<hr />';

$hash->printPosi（）;
```

Java程序哈希算法的配置如下：

```
<bean id="docViewRemoteCache" class="com.Etian.core.common.cache.men.impl.EtianMemCa
    <property name="servers" value="cache-1-2.wlwx.etiantian.org:

12010,

cache-1-3.wlwx.etiantian.org:

12010,

cache-1-4.wlwx.etiantian.org:

12010"/>
```

```
    <property name="weights" value="10,


10,


10" />
    <property name="poolName" value="docViewRemoteCache" />
</bean>
```

13.10　Memcached在集群中session共享案例

13.10.1　Memcached在集群中的session共享存储实战

以下是PHP Web环境集群的session共享存储设置。

默认php.ini中session的类型和配置路径为：

```
;

session.save_handler = files;

session.save_path = "/tmp"
```

修改成如下配置：

```
session.save_handler = memcache
session.save_path = "tcp:

// 10.0.0.19:

11211"
```

说明：

·10.0.0.19：11211为Memcached数据库缓存的IP及端口。

·上述配置适合LNMP/LAMP环境。

·Memcached服务器也可以是多台，并且可通过hash算法调度。

配置完毕通过phpinfo页面查看效果如图13-23所示。

session.save_handler	memcache	memcache
session.save_path	tcp://10.0.0.19:11211	tcp://10.0.0.19:11211

图13-23　效果图

13.10.2 Memcached在集群中的session共享存储的优缺点

优点：

1）读写速度上会比普通files速度快很多。

2）可以解决多个服务器共用session的难题。

缺点：

1）session数据都保存在memory中，持久化方面有所欠缺，但对session数据来说不是问题。

2）一般是单台，如果部署多台，多台之间无法数据同步。通过hash算法分配依然有session丢失的问题。

对于上面的缺点，解决思路如下：

1）可以用其他的持久化系统存储sessions，例如：Redis、ttserver来替代Memcached。

2）高性能高并发场景，cookies效率比session要好很多，因此，大网站都会用cookies解决会话共享问题。

3）有初级运维网友通过牺牲LB的负载均衡的策略实现，例如：lvs-p、nginx ip_hash等，但这些不是好的方法。

更多内容见：

http://oldboy.blog.51cto.com/2561410/1323468

http://oldboy.blog.51cto.com/2561410/1331316

13.11 Memcached兼容持久化工具介绍

13.11.1 MemcacheDB（key-value）

Memcached用于数据库内存缓存时存在一个问题，即进程退出时，数据全丢。这样就算缓存启动了，内存里也没有数据，而这会造成所有用户同时访问数据库，从而导致数据库撑不住。要解决上面的问题，可通过脚本或者程序，从数据库里把数据读出来存到Memcached缓存里，然后前端才能开启对外访问。

MemcacheDB是新浪网基于Memcached开发的一个开源项目。通过增加Berkeley DB的持久化存储机制和异步主辅复制机制，使Memcached具备了事务恢复能力、持久化能力和分布式复制能力，非常适合需要超高性能读写速度、持久化保存的应用场景。如果对Memcached有持久化需求，可以考虑使用MemcacheDB。

MemcacheDB持久化的缓存系统，不但可以像Memcached一样提供内存缓存，还可以把内存的数据放到磁盘。数据量的多少和NOSQL软件的机制决定了重新把数据加载到内存需要多久。

MemcacheDB的特点：

·基于key/value对象的高性能读写数据库。

·支持永久存储具备事务恢复能力。

·支持主从复制负载均衡高可用。

·兼容Memcache协议。

Memcached和MemcacheDB的工作原理如图13-24所示。

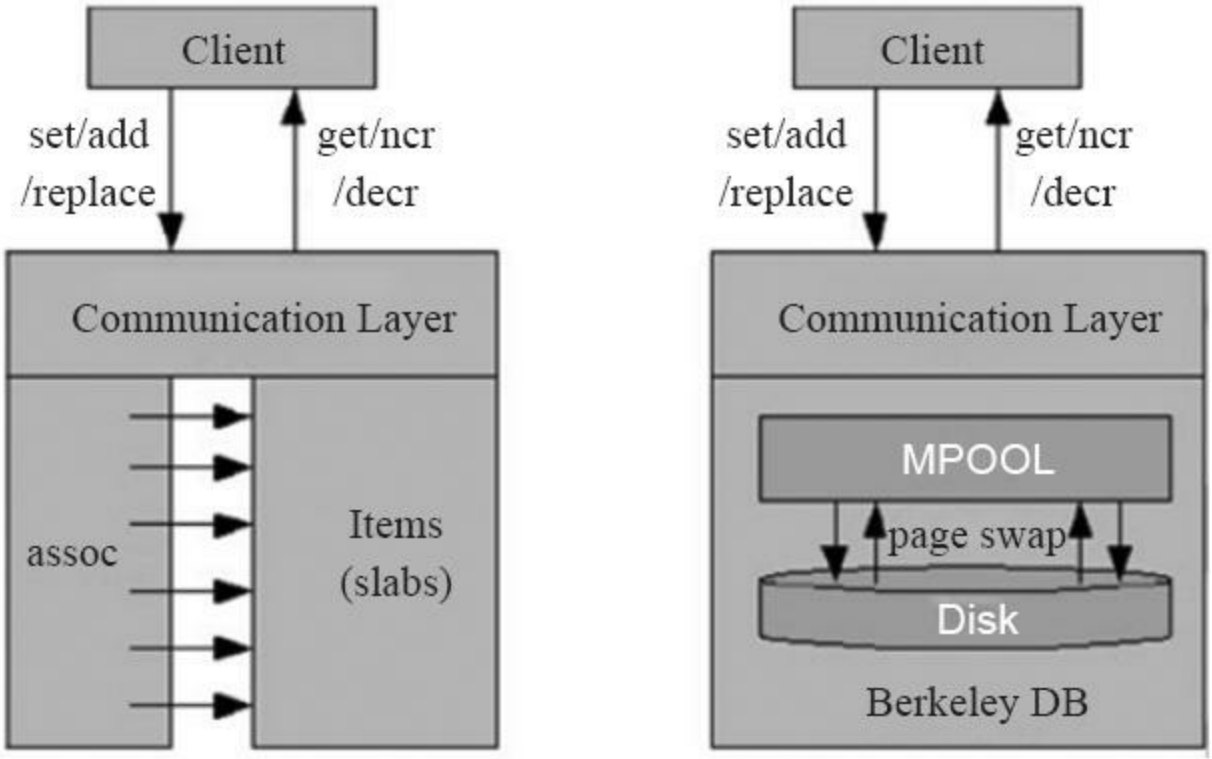


图13-24　Memcached和MemcacheDB的工作原理简单图解

MemcacheDB官方网址为：http://memcachedb.org/ 。

13.11.2 Tokyo Tyrant（key-value）

1.Tokyo Cabinet介绍

Tokyo Cabinet是日本人Mikio Hirabayashi（平林干雄）开发的一款DBM数据库（注：大名鼎鼎的DBM数据库qdbm就是他开发的），该数据库读写非常快。从insert：0.4sec/1000000 recordes（2500000qps）可以看到，写入100万数据只需要0.4秒。从search：0.33sec/1000000 recordes（3000000 qps）可以看到，读取100万数据只需要0.33秒。图13-25为各种key-value数据库读写数据的性能测试，可以看出Tokyo Cabinet的速度是非常快的，是Berkeley DB等DBM的很多倍。

Benchmark Test of DBM Brothers

This benchmark test is to calculate processing time (real time) and file size of database.
Writing test is to store 1,000,000 records. Reading test is to fetch all of its records.
Both of the key and the value of each record are such 8-byte strings as '00000001', '00000002', '00000003'...
Tuning parameters of each DBM are set to display its best performance.
Platform: Linux 2.6.16 kernel, EXT3 file system (writeback), Intel Xeon quad core 2.3GHz CPU, 8GB RAM
Compilation: gcc 4.2.3 (using -O3), glibc 2.7

Result

NAME	DESCRIPTION	WRITE TIME	READ TIME	FILE SIZE
TC	Tokyo Cabinet 1.2.9	0.643	0.773	42,583,208
QDBM	Quick Database Manager 1.8.77	2.813	0.959	56,582,932
NDBM	New Database Manager 5.1	5.183	3.538	834,003,968
SDBM	Substitute Database Manager 1.0.2	6.202	0.000	621,281,280
GDBM	GNU Database Manager 1.8.3	18.878	3.119	88,137,728
TDB	Trivial Database 1.0.6	7.023	0.789	52,523,008
CDB	Tiny Constant Database 0.75	0.352	0.370	40,002,048
BDB	Berkeley DB 4.6.21	9.665	3.118	41,938,944
TC-BT-ASC	B+ tree API of TC (ascending order)	1.039	0.787	32,209,795
TC-BT-RND	B+ tree API of TC (at random)	4.075	3.114	12,466,925
QDBM-BT-ASC	B+ tree API of QDBM (ascending order)	1.401	0.955	40,620,715
QDBM-BT-RND	B+ tree API of QDBM (at random)	6.324	3.837	15,731,675
BDB-BT-ASC	B+ tree API of BDB (ascending order)	1.367	1.449	57,999,360
BDB-BT-RND	B+ tree API of BDB (at random)	3.659	2.316	29,818,880
TC-FIXED	Fixed-length API of TC	0.240	0.037	9,000,256

Unit of time is seconds. Unit of size is bytes.
Read time of SDBM can not be calculated because its database is broken when more than 100000 records.

图13-25　测试结果

2.Tokyo Tyrant介绍

Tokyo Tyrant是提供DBM数据库Tokyo Cabinet的网络接口。它使用简单的基于TCP/IP的简单二进制协议进行通信，同时它还拥有Memcached兼容协议，并且可以用HTTP/1.1协议进行数据交换，所以实现了跨平台、跨语言使用Tokyo Tyrant。此外，Tokyo Tyrant采用热备份、更新日志记录、复制（replication）来实现高可用性和高可靠性。到目前为止，Tokyo Tyrant可以运行在Linux、FreeBSD、Mac OS X、Solaris等系统上。

Tokyo Tyrant加上Tokyo Cabinet，构成了一款支持高并发的分布式持久化存储系统，虽然可以将Tokyo Tyrant Server看成是一个Memcached，但是，它的数据是可以持久存储的，这一点和MemcacheDB一样。

官网：http://fallabs.com/tokyotyrant/

3.Tokyo Tyrant优势与缺点

相比于Memcached和MemcacheDB，Tokyo Tyrant具有以下优势：

·不但支持内存缓存，而且还可以持久化存储。

·故障转移方面，支持主从模式，也支持双机互为主辅模式，主辅库均可读写，而MemcacheDB目前支持类似MySQL主辅库同步的方式实现读写分离，支持“主服务器可读写、辅助服务器只读”模式。

·对于3千万条数据级别内的访问，速度相当快。

Tokyo Tyrant在中小企业的应用很广，其缺点是当数据存储超过3000万条（实际测试机应用得出的参考）时，数据访问性能会急剧下降，但是作为Memcached的协议兼容产品，解决Memcached持久化问题还是不二之选，如果换了Redis等新软件，前端的程序也要有大面积更改。

13.12　本章重点回顾

1）Memcached在企业中的应用场景案例。

2）Memcached企业中常见用途读写工作流程。

3）Memcached特性与工作原理机制。

4）Memcached内存管理核心机制。

5）Memcached检测过期与删除工作原理机制。

6）Memcached服务器端安装与应用实践。

7）Memcache客户端插件安装及配置。

8）Memcached的运行状态信息监控。

9）Memcached服务应用优化策略及案例。

10）Memcached在集群后端作为session共享案例。

11）持久化工具Tokyo Tyrant与MemcacheDB介绍。

第14章　企业级监控Nagios实践

14.1　Nagios监控简介

生活中大家应该对监控已司空见惯了，例如：餐馆门前的监控探头，小区里的视频监控，城市道路高速监控探头等，这些监控的目的大家都清楚，无须多说。那么，企业工作中为什么要部署监控系统呢？

我们都知道军队里，哨兵的角色很重要，我们在杀敌前，基本都要先把敌人站岗的哨兵给解决了，这样敌人就相当于眼睛瞎了，耳朵聋了，然后再进攻就能轻松搞定这些敌人。在互联网企业大型网站架构里，服务器、业务系统相当多，可达上万甚至十万级别，而且还很复杂，如果没有监控系统这个“哨兵”，网络业务出现什么问题，我们就很难知晓，因此，大型网站中监控系统的重要性就不言而喻了。

监控系统需要监控的数据有哪些呢？

·系统本地资源：负载（uptime）、CPU（top、sar）、磁盘（df-hi）、内存（free）、I/O（iostat）、Raid内磁盘故障、CPU温度、passwd文件的变化、本地所有文件改动。

·网络服务：端口、Web（URL）、DB、ping包、进程、IDC带宽网络流量。

·其他设备：路由器、交换机（端口、光衰、日志）、打印机、Windows等。

·业务数据：用户登录失败次数，用户登录网站次数，输入验证码失败的次数，某个API接口流量并发、网络连接数、IP、PV数，电商网站订单，支付交易的数量等。

如何获取到这么多的数据呢？

Nagios监控软件本身仅仅是一个监控平台，理论上想监控的具体内容只要在服务器中执行命令行就可以获取，并纳入Nagios监控体系里。

14.2 Nagios监控工具及原理介绍

14.2.1 Nagios介绍

Nagios是一款开源的网络及服务的监控工具，其功能强大，灵活性强。能有效监控Windows、Linux和UNIX等系统的主机各种状态信息，以及交换机、路由器等网络设备和主机端口及URL服务等，它会根据不同业务故障级别发出告警信息（邮件、微信、短信、语音报警、飞信），当故障恢复时也会发出对应的恢复消息给管理员。

Nagios服务器端可以在Linux系统和类UNIX系统上运行，但目前无法在Windows上运行。Windows可以作为被监控的主机运行Nagios客户端软件。

Nagios监控软件本身仅仅是一个监控平台，我们想监控的具体内容，理论上只要通过服务器命令就可以获取并纳入Nagios监控体系里，所以，可以说Nagios强大到了无所不能的地步。

Nagios官方网站地址为http://www.nagios.org/ 。

14.2.2 Nagios的特点

Nagios可以支持非常多的功能特性，这里把一些常见的功能特性简单介绍如下：

·监控网络服务（HTTP、TCP、PING、SMTP、POP3等）；

·监控主机资源（CPU、负载、I/O状况，虚拟及正式内存及磁盘利用率等）；

·简单的插件设计模式使得用户可以很方便地定制符合自己服务的检测方法；

·并行服务检查机制；

·具备定义网络分层结构的能力，用“parent”主机定义来表达网络主机间的关系，这种关系可被用来发现和明晰主机宕机或不可达的状态；

·当服务或主机问题产生与解决后将及时通报联系人（mail/im/sms/sound/语音）；

·具备定义事件句柄功能，它可以在主机或服务的事件发生时获取更多问题定位；

·自动的日志回滚；

·可以支持并实现对主机的冗余监控（支持分布式监控）；

·可选的Web界面用于查看当前的网络状态、通知和故障历史、日志文件等。

14.2.3 Nagios监控系统家族成员的构成

Nagios监控一般由一个主程序（Nagios）、一个插件程序（Nagios-plugins）和一些可选的附加程序（NRPE、NSClient++、NSCA和NDOUtils）等组成。

Nagios本身只是一个监控的平台而已，其具体的监控工作都是通过各类插件（例如：Nagios-plugins）来实现的，也可以自己编写插件。因此，Nagios主程序和Nagios-plugins插件都是Nagios服务器端必须要安装的程序组件，不过，一般Nagios-plugins也要安装于被监控端，用来获取相应的数据。Nagios可选的附加组件描述如下。

1.NRPE组件

存在的位置： 工作于被监控端，操作系统为Linux/UNIX系统。

作用： 用于在被监控的远程Linux/UNIX主机上执行脚本插件，获取数据回传给服务器端，以实现对这些主机资源和服务的监控。

存在形式： 守护进程（agent）模式，开启的端口为5666。

监控的资源： 主要用于监控本地资源，包括负载（uptime）、CPU（top、sar）、磁盘（df-hi）、内存（free）、I/O（iostat）、Raid

内磁盘故障、CPU温度、passwd文件的变化，以及本地所有文件的指纹识别监控，当然也可以监控进程、端口、URL等。

图14-1为NRPE组件的运行原理图。

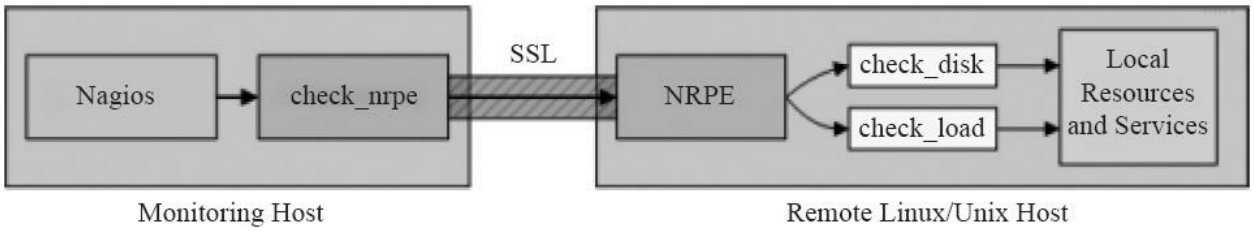


图14-1　NRPE组件的运行原理图

工作原理：通常由Nagios服务器端发起获取数据请求，由check_nrpe插件携带要获取的命令，传给被监控端的nrpe守护进程（默认5666端口），nrpe进程读取nrpe.cfg里对应服务器端发送的命令信息，调用本地插件获取数据，然后返回给Nagios服务器端check_nrpe，进而传给Nagios展示到Web界面中，严格讲可以称之为半被动工作模式，本文主要讲解这个NRPE组件的功能，后文将称其为被动模式。

2.NSClient++组件

NSClient++组件用于被监控端为Windows系统的服务器。

作用：相当于Linux下的nrpe，监控安装在Windows主机上的agent组件。

监控的资源：主要监控Windows系统的本地资源，比如CPU、磁

盘、内存、I/O等。

工作形式： 通常由服务器端发起获取数据请求，由check_nt传给被监控端的NSClient++进程，获取数据后返回给服务器端。

图14-2为NSClient++运行原理图。

工作原理： NSClient++的工作原理和NRPE基本相同，只不过适合于Windows被监控端的监控，且服务器端的插件为check_nt，客户端的进程为NSClient++。通常由服务器端发起获取数据请求，由check_nt传给被监控端的NSClient++进程，获取数据后返回给服务器端，本书不会涉及Windows被监控端，有需要的读者可查看相关书籍或者学习老男孩教育的深入课程。

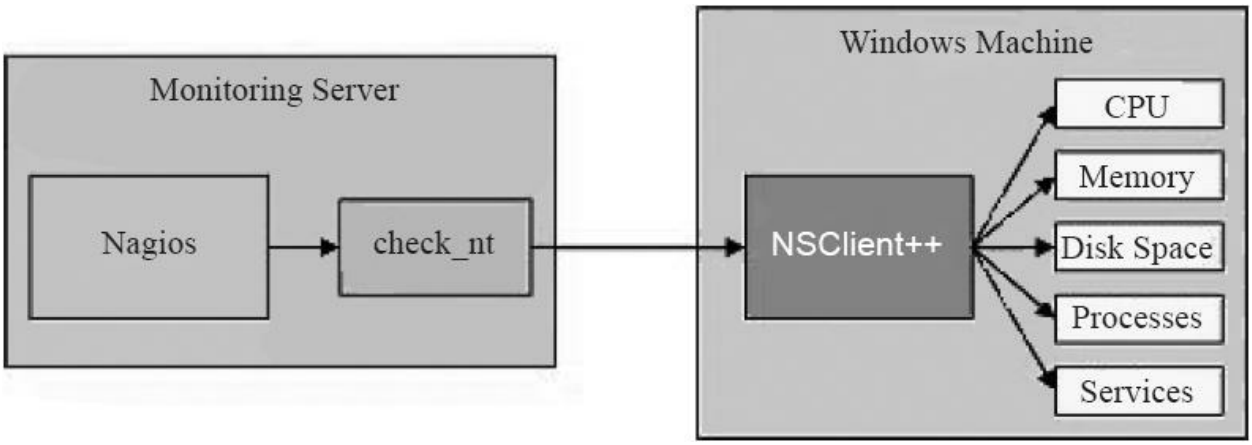


图14-2　NSClient++组件的运行原理图

3.NDOUtils组件（不推荐用）

NDOUtils组件工作于Nagios服务器端。

作用： 将Nagios的配置信息和各event产生的数据存入数据库，以实现对这些数据的检索和处理，对于中小企业，不推荐使用NDOUtils，直接使用文件记录数据就很好，本章后面也不会过多提及此内容。

图14-3为NDOUtils的运行原理简图。

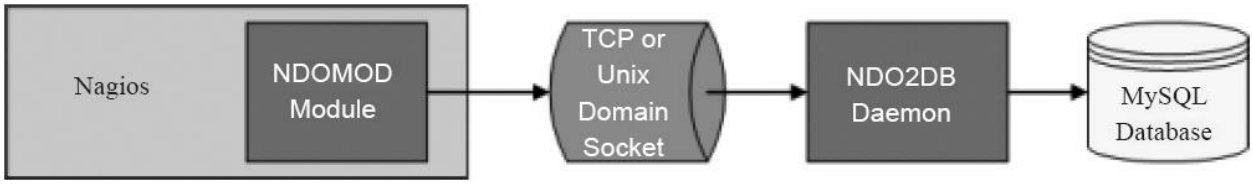


图14-3　NDOUtils组件运行原理简图

4.NSCA组件

相对于nrpe，这个可以说是纯被动模式的监控组件，目前被应用的场景较少，这里只是作为知识点进行介绍，对于中小企业，老男孩不推荐使用。

NSCA需要同时安装在Nagios的服务器端和客户端（被监控端）。

作用： 让被监控的远程Linux/UNIX主机主动将监控到的信息发送给Nagios服务器，可以用在大规模分布式监控集群模式中，中小企业无需使用。

分布式监控NSCA外部构件简介：为完成从远程主机主动提交强制

检测结果，于是就开发了NSCA外部构件。该外部构件包括两部分，第一部分是客户端程序（send_nsca），运行于远程主机上，并负责将强制检测结果发送到指定的Nagios服务器端，另一部分是NSCA守护进程（nsca），它既可以作为守护进程独立运行，也可以注册到inetd里作为一个inetd客户程序来提供监听。从客户端收到服务检测结果信息之后，守护进程将结果提交给在中心服务器的Nagios，方式是在外部命令文件里插入一条PROCESS_SVC_CHECK_RESULT命令，之后跟上检测结果。在Nagios服务器端下一次处理外部命令时将会找到这条由分布式服务器送来的强制检测信息并处理它。

图14-4为NSCA分布式监控运行原理简图。

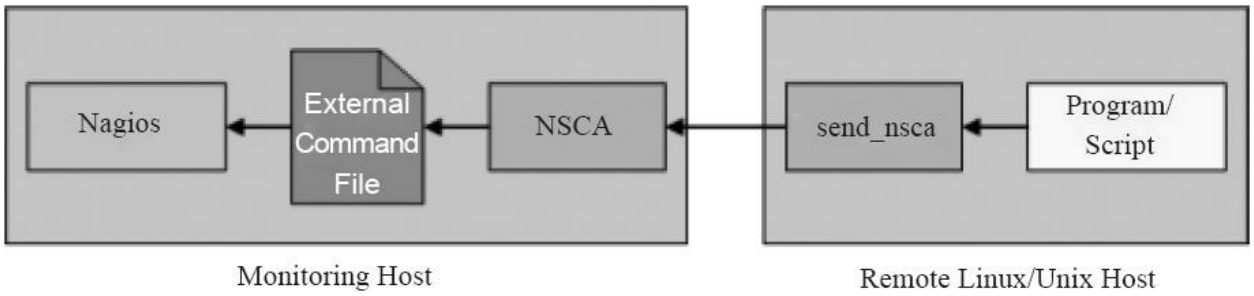


图14-4　NSCA分布式监控运行原理简图

14.2.4　Nagios监控系统完整图解

图14-5为Nagios监控工具综合系统简单原理图。该系统在中小企业用不到，也没必要搞得这么复杂，这里会对将要讲解的Nagios系统做了一个剪裁，只讲解企业最常见的功能应用，对一般的企业已经足够了。

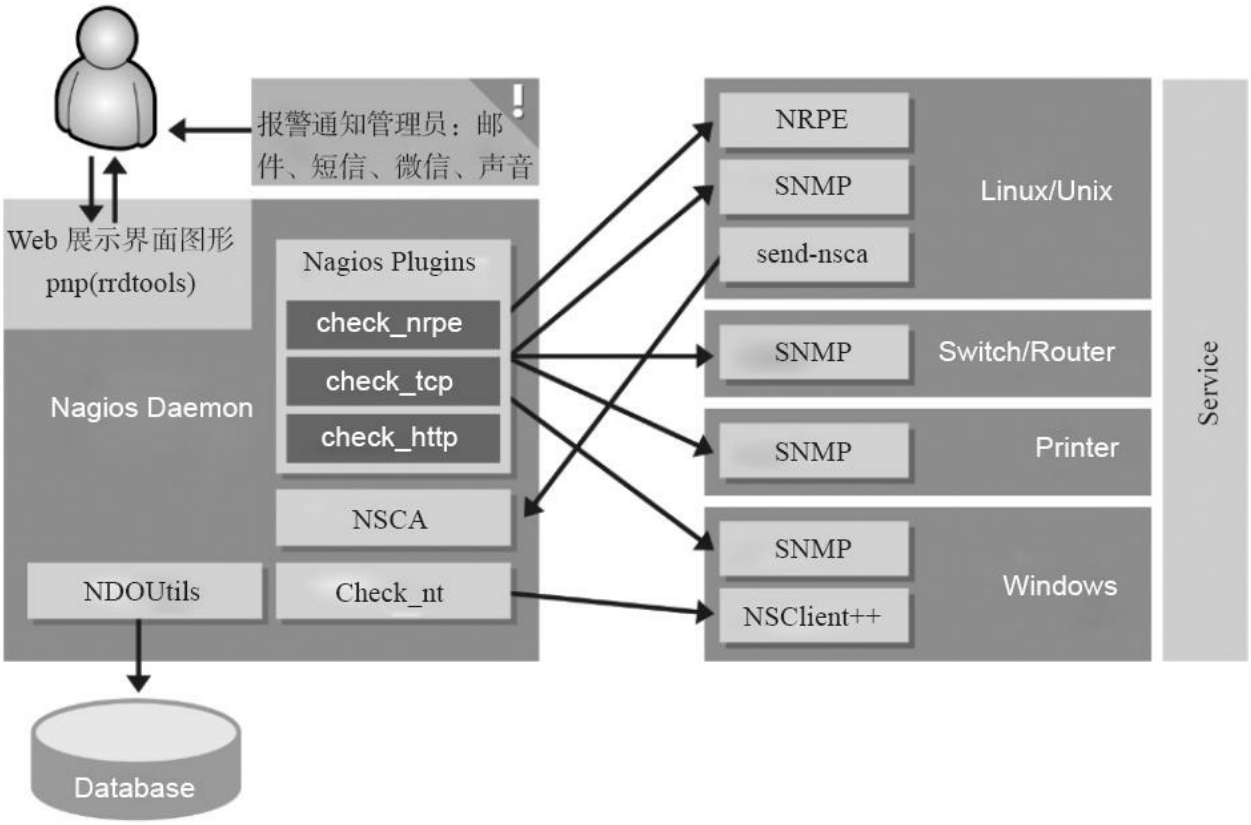


图14-5　Nagios监控工具框架原理总图

运维思想：化繁为简是优秀的运维工程师必须掌握的重要技术思想。

图14-6是接下来要详细讲解的Nagios监控工具实际原理简图。这张原理图符合生产运维的原则：简单、易用、高效。

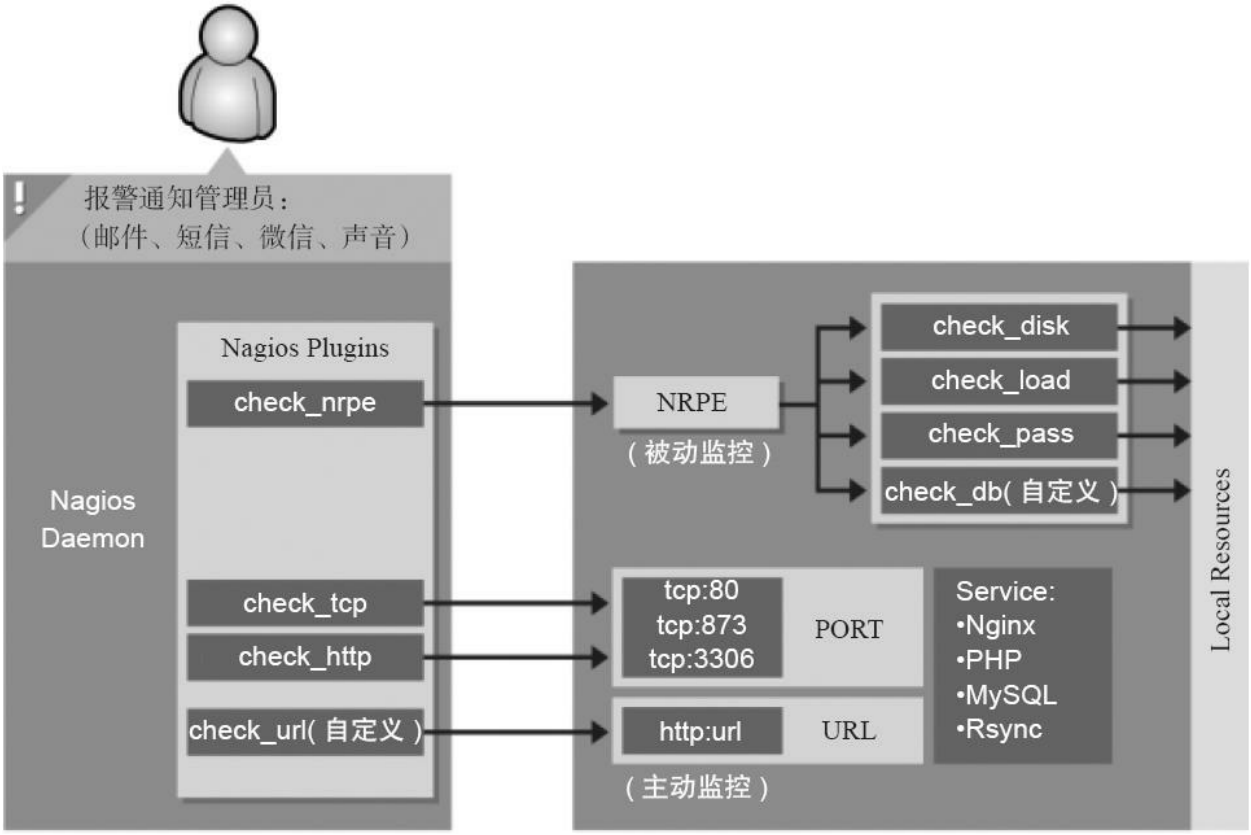


图14-6　Nagios监控本文实战逻辑图

14.3 Nagios服务器端安装

14.3.1 Nagios安装准备

1.准备3台服务器或VM虚拟机

表14-1为Nagios服务器及客户端服务器列表。

表14-1 Nagios服务器及客户端服务器

管理 IP 地址	角　　色	备　　注
10.0.0.7	nagios-server	Nagios 服务器端
10.0.0.8	web01	被监控的客户端服务器
10.0.0.9	web02	被监控的客户端服务器

2.设置yum安装源

默认情况执行yum会从国外的站点下载，速度慢。因此要换成国内的提供yum源的站点，这样安装软件时更快。yum是一个非常方便的RPM软件包安装命令，一般安装基础的软件都会用到。命令语法为：yum install软件包名或关键字-y。

具体安装方法如下：先在命令行执行ping -c 1 baidu.com确保服务器可以上网，然后执行如下命令！

```
cd /etc/yum.repos.d/
/bin/mv CentOS-Base.repo CentOS-Base.repo.oldboy.ori
```

```
wget -O /etc/yum.repos.d/CentOS-Base.repo http:

// mirrors.aliyun.com/repo/Centos-6.repo
```

以上命令可批量执行，需要确保可以上网。具体配置的更改内容，可以通过比较工具来比较已配置好的和原始的内容，从而知晓。会配即可，细节暂时可以不用了解太多。

除了配置网络yum源外，还可以配置光盘yum源等，配置网络yum源的命令具体执行过程如下：

```
[root@nagios-server ~]# ping -c1 baidu.com     #<==检查是否可以上网

PING baidu.com （

123.125.114.144）

 56（

84）

 bytes of data.
64 bytes from 123.125.114.144:

 icmp_seq=1 ttl=46 time=109 ms
[root@nagios-server ~]# cd /etc/yum.repos.d/
[root@nagios-server yum.repos.d]# /bin/mv CentOS-Base.repo CentOS-Base.repo.oldboy.c

[root@nagios-server yum.repos.d]# wget -q -O /etc/yum.repos.d/CentOS-Base.repo http:
```

```
// mirrors.aliyun.com/repo/Centos-6.repo          #<==更新为阿里云提供的

                                                  yum源文件及地址

[root@nagios-server yum.repos.d]# cd ~
```

这样yum安装源就配好了。

3.解决Perl软件编译问题

在安装好Nagios监控服务后，还要安装Perl插件程序，因此要提前设置相关环境变量，批量执行命令如下：

```
echo 'export LC_ALL=C'>> /etc/profile
tail -1 /etc/profile
source /etc/profile
echo $LC_ALL
cd ~
```

执行过程如下：

```
[root@nagios-server ~]# echo 'export LC_ALL=C'>> /etc/profile
#<==设置环境变量，解决后面

Perl程序插件的编译问题。符号“

>>”表示向文件追加内容

[root@nagios-server ~]# tail -1 /etc/profile     #<==查看是否正确追加了
```

```
export LC_
                                        ALL=C环境配置



export LC_ALL=C
[root@nagios-server ~]# source /etc/profile      #<==使增加的环境变量配置生效



echo $LC_ALL
cd ~
[root@nagios-server ~]# echo $LC_ALL           #<==查看设置变量结果



C
[root@nagios-server ~]# cd ~
```

4.关闭Nagios server端防火墙及SELinux

在测试环境下为了调试方便，最好关掉iptables防火墙及SELinux，如果是生产环境中，因为有外部IP，所以在调试完毕后需要开启防火墙。一般允许服务通过的方法是整个局域网IP段都通过，SELinux是一个可关可开的软件，企业可以根据需求选择，大部分企业还是会选择关闭SELinux，使用其他保护措施，因此，这里也关闭SELinux。

关闭防火墙的命令集合如下：

```
/etc/init.d/iptables stop
/etc/init.d/iptables status
chkconfig iptables off
chkconfig --list iptables
```

执行过程如下：

```
[root@nagios-server ~]# /etc/init.d/iptables stop
#<==这是关闭


iptables的命令，等同于


service iptables stop
[root@nagios-server ~]# /etc/init.d/iptables status     #<==查看


iptables是否关闭了



Firewall is stopped.
#<==提示防火墙已关闭



[root@nagios-server ~]# chkconfig iptables off
#<==关闭文本模式下


iptables防火墙的自启动



#<==提示：测试环境下为了调试方便而关掉


iptables，生产环境在调试完毕后需要开启防火墙，允许


Nagios服务通过就可以了，不需要关闭



[root@nagios-server ~]# chkconfig --list iptables
iptables          0：


off   1：


off   2：
```

```
off  3:

off  4:

off  5:

off   6:

off
```

下面的命令用于关闭SELinux对系统的控制（关闭原因见第3章Linux优化部分）。

```
[root@nagios-server ~]# sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/
<==修改配置文件则永久生效，但是必须要重启系统



[root@nagios-server ~]# grep SELINUX=disabled /etc/selinux/config
SELINUX=disabled
[root@nagios-server ~]# setenforce 0     <==临时生效的命令



[root@nagios-server ~]# getenforce     <==查看


SELinux当前状态



Permissive                          ←会警告，但不阻止。
```

提示：处理SELinux问题有三种方法。

1）临时关闭服务器开启的SELinux可直接输入如下命令：

```
[root@nagios-server nagios]# setenforce 0
```

这个命令输入后会直接生效。

2）永久关闭服务器开启的SELinux可直接输入如下命令：

```
[root@nagios-server nagios]#sed -i 's#SELINUX=enforcing#SELINUX=disabled#' /etc/seli
```

这个sed替换相当于执行vi/etc/selinux/config命令，然后修改SELINUX项为disabled。

要使得服务器的SELinux关闭的配置信息立刻生效，需要重启服务器才可以。

3）不关闭SELinux的解决办法如下：

```
[root@nagios-server nagios]# chcon -R -t httpd_sys_content_t /usr/local/nagios/sbin/
[root@nagios-server nagios]# chcon -R -t httpd_sys_content_t /usr/local/nagios/share
```

推荐你1、2同时执行，这样的好处就是当前生效了，以后重启服务时也生效。有关SELinux的内容本书不涉及，请参考相关文章。

5.解决系统时间同步问题

如果不解决服务器的时间同步问题，很可能会导致Nagios整个服务

配置异常甚至失败，这个是很多初学者不太注意的地方，Nagios一度被称为“难够死”，意思是太难配置了，不过跟随老男孩来实践，相信你会感觉很轻松。

批量执行的命令如下：

```
/usr/sbin/ntpdate pool.ntp.org
echo '#time sync by oldboy at 2015-06-26'>>/var/spool/cron/root
echo '*/5 * * * * /usr/sbin/ntpdate pool.ntp.org >/dev/null 2>&1'>>/var/spool/cron/r
crontab -l
```

执行过程如下：

```
[root@nagios-server ~]# /usr/sbin/ntpdate pool.ntp.org
26 Jun 21:



26:



08 ntpdate[26368]:



 adjust time server 202.112.31.197 offset 0.030437 sec
#<==让



Nagios server的系统时间和当前的标准时间保持一致。虚拟机要能上网才行



[root@nagios-server ~]# echo '#time sync by oldboy at 2015-06-26'>>/var/spool/cron/r
[root@nagios-server ~]#echo '*/5 * * * * /usr/sbin/ntpdate pool.ntp.org >/dev/null 2
[root@nagios-server ~]# crontab -l
#time sync by oldboy at 2015-06-26
*/5 * * * * /usr/sbin/ntpdate pool.ntp.org >/dev/null 2>&1
#<==执行



 echo '*/5 * * * * /usr/sbin/ntpdate pool.ntp.org >/dev/null 2>&1'>>/var/spool/cron/
```

```
crontab -l查看。有关定时任务

crontab请参考老男孩其他课程或相关书籍
```

6.安装Nagios服务器端所需软件包

Nagios服务器端需要有Web界面展示监控效果，界面的展示主要使用PHP程序，因此，需要LAMP环境。

 特别强调：有些网友总想安装LNMP环境，这完全是自找麻烦，yum安装的LAMP环境是配合Nagios服务器端展示界面的最佳环境。

批量执行命令如下：

```
yum install gcc glibc glibc-common -y      #<==编译软件升级

yum install gd gd-devel -y           #<==用于后面

PNP出图的包

yum install mysql-server -y           #<==非必须，如果有监控数据库，那么需要先安装

                                    MySQL，否则，
```

```
MySQL的相关插件不会被安装


yum install httpd php php-gd -y          #<==Apache、

PHP环境
```

 提示：

1）通过yum工具安装上述所有软件包，且这些环境一般不需要在Nagios客户端安装。

2）上述软件包装好后的版本为：Apache2.2.15、PHP5.3.3、MySQL 5.1.73。

下面通过yum安装所需的基础软件包。

```
[root@nagios-server ~]# yum install gcc glibc glibc-common -y  #<==GCC compiler
Loaded plugins:


 fastestmirror
Determining fastest mirrors
addons                                  |  951 B     00:


00
base                                    |  2.1 kB    00:


00      ...省略大部分...
```

```
[root@nagios-server ~]#yum install gd gd-devel -y...省略全部...


[root@nagios-server ~]yum install httpd php php-gd -y
#此处还是推荐使用


yum install httpd php* -y...省略全部...


[root@nagios-server ~]# rpm -qa mysql httpd php #<==检查


LAMP环境的版本


mysql-5.1.73-5.el6_6.x86_64
httpd-2.2.15-39.el6.centos.x86_64
php-5.3.3-40.el6_6.x86_64
```

7.创建Nagios服务器端需要的用户及组

批量执行命令如下：

```
/usr/sbin/useradd nagios      #<==这个地方最好创建家目录，否则，启动


Nagios会提醒没家目录


/usr/sbin/groupadd nagcmd
/usr/sbin/usermod -a -G nagcmd nagios
/usr/sbin/usermod -a -G nagcmd apache
id -n -G nagios
id -n -G apache
groups nagios
groups apache
```

执行过程如下：

```
[root@nagios-server ~]# /usr/sbin/useradd nagios
#<==添加用户
```

nagios。

```
[root@nagios-server ~]# /usr/sbin/useradd apache -M -s /sbin/nologin
#<==添加用户
```

apache，

-s /sbin/nologin表示禁止其登录。如果已经存在

apache用户会有如下提示

useradd：

user apache exists（一般使用

yum安装

httpd时，

Apache用户会同步创建好。）

```
[root@nagios-server ~]# /usr/sbin/groupadd nagcmd
#<==添加用户组
```

nagcmd

```
[root@nagios-server ~]# /usr/sbin/usermod -a -G nagcmd nagios
#<==usermod -a -G表示把用户
```

nagios加入到

nagcmd组

```
[root@nagios-server ~]# /usr/sbin/usermod -a -G nagcmd apache
#<==把用户
```

apache加入到

nagcmd组

```
[root@nagios-server ~]# id -n -G nagios     #<==检查
```

nagios用户所属的组

-n等同于

--
name，显示组名

```
nagios nagcmd
[root@nagios-server ~]# id -n -G apache     #<==检查
```

apache用户所属的组

-n等同于

--
name，显示组名

```
apache nagcmd
#提示：如果使用软件包编译


Apche，默认


Apache用户为


daemon，此时需执行


/usr/sbin/usermod -a -G nagcmd daemon，也可用


groups加用户名来查，比如：



[root@nagios-server ~]# groups nagios
nagios ：


 nagios nagcmd
[root@nagios-server ~]# groups apache
apache ：


 apache nagcmd
```

8.上传软件包到指定目录或通过URL下载

操作命令如下：

```
[root@nagios-server ~]# mkdir -p /home/oldboy/tools/nagios
[root@nagios-server ~]# cd /home/oldboy/tools/nagios
[root@nagios-server nagios]# rz -y
#<==本文选择从本地上传软件包集合


oldboy_training_nagios_soft.zip，输入
```

```
rz -y命令，然后通过弹出的窗口上传软件包集合


oldboy_training_nagios_soft.zip
rz waiting to receive.正在开始


 zmodem 传输。


 按


 Ctrl+C 取消。


正在传输


 oldboy_training_nagios_soft.zip...
100%    6322 KB 6322 KB/s 00:


00:


01  0 错误
```

或者可以自行去下载后面需要的软件包：

地址：http://sourceforge.net/projects/nagios/files/

https://www.nagios-plugins.org/download/nagios-plugins-1.4.16.tar.gz

 技巧：rz是一个很好的从本地上传文件的工具，-y参数表示覆盖存在的文件，可用yum install lrzsz-y命令安装此工具。

为了方便同学们学习安装Nagios，老男孩已搜集好相关软件包一并打包提供给大家，内容如下：

```
[root@nagios-server tools]# unzip oldboy_training_nagios_soft.zip
[root@server tools]# tree nagios/
nagios/
|-- nagios-3.5.1.tar.gz              ←

 Nagios主程序

|-- nagios-plugins-1.4.16.tar.gz     ←

Nagios插件包

|-- nrpe-2.12.tar.gz                ←客户端的守护进程软件，

agent。

|-- Class-Accessor-0.31.tar.gz          ←

iostat插件需要它

|-- Config-Tiny-2.12.tar.gz          ←

iostat插件需要它

|-- Math-Calc-Units-1.07.tar.gz          ←

iostat插件需要它
```

|-- Params-Validate-0.91.tar.gz ←

iostat插件需要它

|-- Regexp-Common-2010010201.tar.gz ←

iostat插件需要它

|-- Nagios-Plugin-0.34.tar.gz ←

iostat插件需要它

|-- check_iostat ←这就是上面提到

iostat插件程序

|-- check_memory.pl ←检测内存的插件程序

|-- check_mysql ←检测

MySQL插件程序，这里可不用

|-- libart_lgpl-2.3.17.tar.gz ←绘图相关依赖库

|-- pnp-0.4.14.tar.gz ←

绘图的

web界面软件

`-- rrdtool-1.2.14.tar.gz ←实际绘图软件，被

PNP调用

0 directories,

15 files

现在启动LAMP环境的HTTP服务：

```
[root@nagios-server nagios]# /etc/init.d/httpd start
Starting httpd:

 httpd:

 apr_sockaddr_info_get()

 failed for nagios-server
                                ←可忽略或者看后文解决方案

httpd:

 Could not reliably determine the server's fully qualified domain name,

 using 127.0.0.1 for ServerName          ←可忽略或者看后文解决方案
```

```
                                                [  OK  ]
  [root@nagios-server nagios]# /etc/init.d/httpd restart
  Stopping httpd:


                                      [  OK  ]
  Starting httpd:


                                      [  OK  ]
  [root@nagios-server nagios]# lsof -i :


80
  COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
httpd     26666   root     4u  IPv6  41346     0t0  TCP *:


http  (


LISTEN)



httpd     26668 apache     4u  IPv6  41346     0t0  TCP *:


http  (


LISTEN)



httpd     26669 apache     4u  IPv6  41346     0t0  TCP *:


http  (


LISTEN)
```

```
httpd     26670 apache     4u  IPv6  41346     0t0  TCP *:
http  (
LISTEN)

httpd     26671 apache     4u  IPv6  41346     0t0  TCP *:
http  (
LISTEN)

httpd     26672 apache     4u  IPv6  41346     0t0  TCP *:
http  (
LISTEN)

httpd     26673 apache     4u  IPv6  41346     0t0  TCP *:
http  (
LISTEN)

httpd     26674 apache     4u  IPv6  41346     0t0  TCP *:
http  (
LISTEN)
```

```
httpd     26675 apache     4u  IPv6  41346     0t0  TCP *:

http  (

LISTEN)
```

上面的结果表明Nagios服务器端的LAMP环境是正常的。下面针对HTTP启动时的提示进行说明。

提示一：“Starting httpd：httpd：apr_sockaddr_info_get（）failed for nagios-server”，表示这是hosts解析问题，在/etc/hosts中配好主机名和IP的解析就好了，配置结果如下：

```
[root@nagios-server nagios]# grep nagios-server /etc/hosts
10.0.0.7 nagios-server
```

提示二：“httpd：Could not reliably determine the server's fully qualified domain name，using 127.0.0.1 for ServerName”，这表示httpd.conf中缺少ServerName配置，可以在/etc/httpd/conf/httpd.conf中加入ServerName 127.0.0.1：80。

14.3.2　安装Nagios服务器端

官方安装文档：

http：

// nagios.sourceforge.net/docs/3_0/quickstart-fedora.html。

```
[root@nagios-server ~]# cd /home/oldboy/tools/nagios
[root@nagios-server nagios]# ls -l     #<==执行操作命令后，检查是个好习惯
```

```
total 6324
-rw-r--r-- 1 root root 6474375 Dec  9  2013 oldboy_training_nagios_soft.zip
[root@nagios-server nagios]# unzip -q oldboy_training_nagios_soft.zip
#<==*.zip格式的文件
```

可用

unzip来解压，不能用

tar来解压，这点要注意

```
[root@nagios-server nagios]# cd /home/oldboy/tools/nagios
[root@nagios-server nagios]# ls -l nagios-3.5.1.tar.gz
-rw-r--r-- 1 root root 1763584 Aug 31  2013 nagios-3.5.1.tar.gz
[root@nagios-server nagios]# tar xf nagios-3.5.1.tar.gz
[root@nagios-server nagios]# cd nagios     #<==这里不是传统的
```

nagios-3.5.1目录

```
[root@nagios-server nagios]# ./configure --with-command-group=nagcmd...省略开头...


Review the options above for accuracy.  If they look okay,


type 'make all' to compile the main program and CGIs.
#<==如


./configure 命令执行后出现如上两行结尾提示，表示正常


  [root@nagios-server nagios]# make all...省略开头...


Enjoy.
#<==如


make all命令执行后出现如上结尾


Enjoy.提示，表示正常


[root@nagios-server nagios]# make install...省略开头...


  make install-init
    - This installs the init script in /etc/rc.d/init.d
  make install-commandmode
    - This installs and configures permissions on the
    directory for holding the external command file
  make install-config
    - This installs sample config files in /usr/local/nagios/etc
make[1]:


 Leaving directory `/home/oldboy/tools/nagios/nagios'
#<==make install 命令执行后的正确结尾
```

提示：如果是编译安装httpd，可以执行./configure--with-command-group=nagcmd--with-httpd-conf=/usr/local/apache2/conf/extra增加一个编译参数，即指定编译Nagios Web配置的生成路径。

下面根据make install命令执行后的提示继续操作。

```
[root@nagios-server nagios]# make install-init      #<==安装初始化脚本到


                                    /etc/rc.d/init.d
[root@nagios-server nagios]# make install-config#<==生成


Nagios模板配置


                                    文件到


/usr/local/nagios/etc
[root@nagios-server nagios]# make install-commandmode
                                    #<==安装配置目录许可外部命令文件
```

1.安装Nagios Web配置文件及创建登录用户

接着来安装Nagios Web配置文件（生成Nagios对应于Apache里的配置文件）。

```
[root@nagios-server nagios]# make install-webconf      #<==生成


/etc/httpd/conf.d/
```

nagios.conf配置文件

```
/usr/bin/install -c -m 644 sample-config/httpd.conf /etc/httpd/conf.d/nagios.conf
*** Nagios/Apache conf file installed ***
[root@nagios-server nagios]# cd ..
```

创建Nagios Web监控界面后，登入时会需要用户名及密码，这里分别为oldboy和123456。

```
[root@nagios-server nagios]# htpasswd -bc /usr/local/nagios/etc/htpasswd.users oldbo
Adding password for user oldboy
[root@nagios-server nagios]# cat /usr/local/nagios/etc/htpasswd.users
oldboy:


kQr1C/cKFPSGQ
```

提示：为什么要把密码生成到/usr/local/nagios/etc/htpasswd.users里呢？不能改路径么？

这是因为/etc/httpd/conf.d/nagios.conf配置文件已经指定了htpasswd.user路径和文件名，如果想改可以先修改nagios.conf配置，但是没必要改，查看方式如下：

```
[root@nagios-server nagios]# grep AuthUserFile /etc/httpd/conf.d/nagios.conf
    AuthUserFile /usr/local/nagios/etc/htpasswd.users
    AuthUserFile /usr/local/nagios/etc/htpasswd.users
```

重新加载Apache服务：

```
[root@nagios-server nagios]# /etc/init.d/httpd reload
Reloading httpd:
```

2.添加监控报警信息接收的Email地址

快速修改方法如下：

```
  [root@nagios-server nagios]# sed -i 's#nagios@localhost#31333741@qq.com#g' /usr/lo
  [root@nagios-server nagios]# sed -n '35p' /usr/local/nagios/etc/objects/contacts.c
email             31333741-@qq.com ;


    #<==这里是你的个人


QQ啊，不是老师的


QQ
```

或者利用编辑器修改，输入如下命令：

```
[root@nagios-server nagios]# vi /usr/local/nagios/etc/objects/contacts.cfg +35
```

修改如下行：

```
email                   nagios@localhost
```

改为：

```
email                   31333741-@qq.com
```

保存，退出。

常见的发送邮件方法有两种，一种是启动本机的邮件服务postfix。另外一种是使用网上第三方邮件服务商提供的服务，例如：QQ邮件服务或者网易邮件服务，如果是简单测试，推荐使用网易的126或163的smtp邮件发送服务地址配置。

第一种方法：监控服务器本地开启邮件服务。此功能依赖本机的服务。

```
[root@nagios-server nagios]# /etc/init.d/postfix start
Starting postfix:


                                   [  OK  ]
[root@nagios-server nagios]# lsof -i :


25
COMMAND   PID USER     FD    TYPE DEVICE SIZE/OFF NODE NAME
master    11556 root   12u   IPv4  61647   0t0  TCP localhost:


smtp  (


LISTEN)



master    11556 root   13u   IPv6  61649   0t0  TCP localhost:


smtp  (


LISTEN)
```

```
[root@nagios-server nagios]# chkconfig postfix on
[root@nagios-server nagios]# chkconfig --list postfix
postfix         0:

off     1:

off     2:

on      3:

on      4:

on      5:

on      6:

off
```

如果postfix启动比较慢，可以修改/etc/hosts做好本机IP和主机名的映射。

```
[root@nagios-server nagios]# grep nagios-server /etc/hosts
10.0.0.7 nagios-server
```

发送邮件测试（如图14-7所示）：

图14-7　接收Linux mail命令发送的QQ邮件信息

```
[root@nagios-server nagios]# mail -s "test" 31333741@qq.com </etc/hosts
[root@nagios-server nagios]# mailq
-Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-------
59FF412FC*    628 Sat Jun 27 12:


18:



09     root@nagios-server.localdomain
                                31333741@qq.com
-- 0 Kbytes in 1 Request.
[root@nagios-server nagios]# mailq
Mail queue is empty
```

提示：此环境下由于没有外网IP，并且邮件服务没有做MX记录及反向解析，因此，所发的邮件经常会收不到或者被当做垃圾邮件。

第二种方法：监控服务器本地开启邮件服务。

使用网上第三方邮件服务商提供的邮箱时，只需修改/etc/mail.rc在最后增加如下两行内容即可：

```
set from=70271111@qq.com
smtp=smtp.qq.comset smtp-auth-user=70271111 smtp-auth-password=123456 smtp-auto=logi
```

这个第三方服务选择QQ邮件服务或者网易邮件服务均可，这里选择了QQ邮件服务。

注意这里的70271111@qq.com是作为报警的发件人的，相当于使用70271111用户和123456密码登录QQ信箱，然后给人发信，收件人就是在contact.cfg里定义的。

3.配置启动Apache服务

下面启动Apache服务并加入系统开机自启动：

```
[root@nagios-server nagios]# /etc/init.d/httpd start
Starting httpd:



[root@nagios-server nagios]# /etc/init.d/httpd restart
Stopping httpd:


                           [  OK  ]
Starting httpd:


                           [  OK  ]
[root@nagios-server nagios]# chkconfig httpd on
[root@nagios-server nagios]# netstat -lntup|grep httpd
tcp   0   0  : : :


80  : : :


*  LISTEN  11627/httpd     #<==80端口存在，即表示


Apache
                                    已经启动了。
```

此时，打开客户端计算机上的浏览器访问http://server-ip/nagios （这里是http://10.0.0.7/nagios ）会出现用户名和密码提示窗口（如图14-8所示）。

图14-8　访问Nagios Web界面用户名和密码提示窗口

输入前面生成好的用户名和密码即oldboy和123456。

出现图14-9所示的界面表示Nagios及Nagios的Web环境已经正常了。

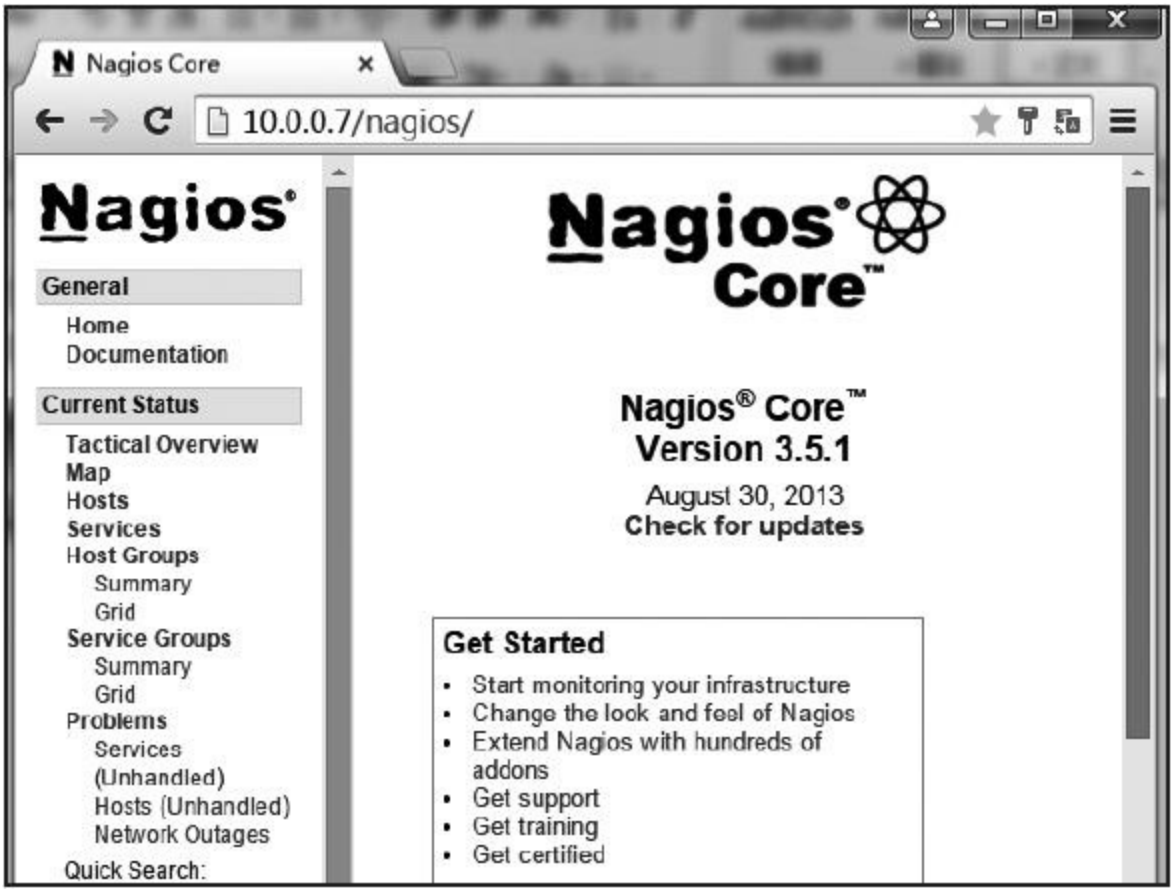


图14-9　登录后的Nagios Web界面窗口

4.安装Nagios插件软件包

Nagios插件软件包就是一些实现获取数据信息的命令或程序，通过这些命令或程序，Nagios可以获取到需要的数据，然后进行报警和展示。具体安装过程如下。

首先安装基础依赖包：

```
[root@nagios-server ~]# yum install perl-devel openssl-devel -y
```

然后安装Nagios plugins插件包：

```
[root@nagios-server ~]# cd /home/oldboy/tools/nagios
[root@nagios-server nagios]# ls nagios-plugins-1.4.16.tar.gz
nagios-plugins-1.4.16.tar.gz
[root@nagios-server nagios]# tar xf nagios-plugins-1.4.16.tar.gz
[root@nagios-server nagios]# cd nagios-plugins-1.4.16
[root@nagios-server nagios-plugins-1.4.16]# ./configure --with-nagios-user=nagios --
#<==如果一行放不下，可以通过反斜线换行输入。



[root@nagios-server nagios-plugins-1.4.16]# make
```

需要说明的是，make过程容易报各种错误！

错误1：

```
check_http.c:

 In function 'process_arguments':

check_http.c:

312:

 error:

 'ssl_version' undeclared (

first use in this function)

check_http.c:
```

```
312:

 error:

  (

Each undeclared identifier is reported only once
check_http.c:

312:

 error:

 for each function it appears in.)

make[2]:

 *** [check_http.o] Error 1
make[2]:

 Leaving directory `/home/oldboy/tools/nagios/nagios-plugins-1.4.16/plugins'
make[1]:

 *** [all-recursive] Error 1
make[1]:

 Leaving directory `/home/oldboy/tools/nagios/nagios-plugins-1.4.16'
make:

 *** [all] Error 2
```

解决办法：

首先执行yum install perl-devel openssl-devel-y，然后删除配置的目录，重新配置编译，别忘了指定--with-mysql。

错误2：

```
Can't run perl Makefile.PL at ../tools/build_perl_modules line 68.
```

解决方法：

```
yum install perl-devel perl-CPAN –


y
make
make clean
```

解决make执行错误后，执行如下操作完成安装。

```
[root@nagios-server nagios-plugins-1.4.16]# make install
[root@nagios-server nagios-plugins-1.4.16]# cd ..
```

此时检查插件个数：

```
[root@nagios-server nagios-plugins-1.4.16]# ls /usr/local/nagios/libexec/|wc -l
59 #<==多一个少一个都是正常的，不要太在意。
```

5.安装nrpe软件

可能有读者会问，nrpe是客户端需要安装的软件，为什么还要安装在Nagios服务器端？这是因为：

·Nagios服务器端需要check_nrpe插件做被动检查，如果服务器端不装nrpe软件，就没有check_nrpe这样的检查插件。

·Nagios服务器端本地的资源也需要被监控，因此，Nagios服务器端也会被作为客户端。

nrpe软件的安装非常简单，就不多费笔墨了。安装nrpe软件的命令集如下：

```
ls /usr/local/nagios/libexec/check_nrpe
tar zxvf nrpe-2.12.tar.gz
cd nrpe-2.12
./configure
make all
make install-plugin
make install-daemon
make install-daemon-config
cd ..
ls /usr/local/nagios/libexec/check_nrpe
```

执行过程如下：

```
[root@nagios-server nagios]# ls /usr/local/nagios/libexec/check_nrpe
ls:


 cannot access /usr/local/nagios/libexec/check_nrpe:


 No such file or directory
[root@nagios-server nagios]# ls -l nrpe-2.12.tar.gz
-rw-r--r-- 1 root root 405725 Oct 10  2009 nrpe-2.12.tar.gz
[root@nagios-server nagios]# tar xf nrpe-2.12.tar.gz
[root@nagios-server nagios]# cd nrpe-2.12
```

```
[root@nagios-server nrpe-2.12]# ./configure
[root@nagios-server nrpe-2.12]# make all
[root@nagios-server nrpe-2.12]# make install-plugin
[root@nagios-server nrpe-2.12]# make install-daemon
[root@nagios-server nrpe-2.12]# make install-daemon-config
[root@nagios-server nrpe-2.12]# cd ../
```

检查check_nrpe插件：

```
[root@nagios-server nagios]# ls /usr/local/nagios/libexec/check_nrpe
/usr/local/nagios/libexec/check_nrpe
[root@nagios-server nagios]# ls /usr/local/nagios/libexec/|wc -l
60
```

 提示：生成nrpe的配置文件为/usr/local/nagios/etc/nrpe.cfg。

到此为止，Nagios服务器端的软件安装部分就配置完了。

6.配置并启动Nagios服务

首先添加Nagios服务到开机自启动：

```
[root@nagios-server nagios]# chkconfig nagios on
[root@nagios-server nagios]# chkconfig --list nagios
nagios          0:


off     1:


off     2:


on      3:


on      4:
```

```
on     5:

on     6:

off
```

更好的设置自动开机的方法是：

```
[root@nagios-server nagios]# echo "/etc/init.d/nagios start" >>/etc/rc.local
[root@nagios-server nagios]# tail -1 /etc/rc.local
/etc/init.d/nagios start
```

然后验证Nagios配置文件（检查语法）。

```
[root@nagios-server nagios]# /etc/init.d/nagios checkconfig
#<==此命令为检查语法的命令，但是只能报告对错无法给出错误的信息



Running configuration check... OK.
```

可以使用命令行命令检查报错，并输出信息。

```
[root@nagios-server nagios]# /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/r



Total Warnings:


 0
Total Errors:


   0
Things look okay - No serious problems were detected during the pre-flight check
#<== Total Warnings警告和
```

Total Errors错误

都为

0，表示配置正常：

#<==Total Errors：

如果不为

0 必须解决。否则

Nagios无法启动。

#<==Total Warnings：这个如果不为

0可以启动

Nagios，可以不理睬。

可以修改/etc/init.d/nagios实现上述命令行检查语法的详细输出，如下：

```
[root@nagios-server nagios]# /etc/init.d/nagios checkconfig
Running configuration check... OK.
#<==此命令默认情况只能判断错误或正确，并不显示错误的详细信息，如果想显示详细错误信息，需要做下简单的
```

```
[root@nagios-server nagios]# grep 'checkconfig)

' -n -A 2 /etc/init.d/nagios
181:

     checkconfig)

182-          printf "Running configuration check..."
183-          $NagiosBin -v $NagiosCfgFile > /dev/null 2>&1;

                              #<==删除定向到空的内容，可让报错输出详细信息
```

删除上述脚本中的>/dev/null 2>&1；，即可检查语法：

```
[root@server nagios]# /etc/init.d/nagios checkconfig
Running configuration check...
Nagios Core 3.2.0
Processing object config file '/usr/local/nagios/etc/objects/commands.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/contacts.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/timeperiods.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/templates.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/localhost.cfg'...
...skip...
Checking for circular paths between hosts...
Checking for circular host and service dependencies...
Checking global event handlers...
Checking obsessive compulsive processor commands...
Checking misc settings...
Total Warnings:

 0
Total Errors:

   0
Things look okay - No serious problems were detected during the pre-flight check
  OK.
```

之后，启动Nagios服务，命令如下：

```
[root@nagios-server nagios]# /etc/init.d/nagios start
Starting nagios:


 done.
```

检查Nagios服务器端进程及端口：

```
[root@nagios-server nagios]# ps -ef|grep nagios|grep -v grep
#<==grep -v grep表示排除自身的这个带


grep命令的进程。出现下文中的提示表示


Nagios已经启动



nagios 22442  1  0 21:


46    00:


00:


00 /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg
[root@nagios-server nagios]# netstat -lntup|grep nagios#<==没发现


Nagios服


                                                务器端有端口
```

最后浏览Nagios Web界面检查。打开浏览器访问：http://服务器端ip/nagios会出现如图14-10所示提示。

图14-10　访问Nagios Web界面用户名和密码提示窗口

输入前面生成好的用户名和密码（oldboy和123456）。如果可以登录，并打开Nagios界面就可以了（界面窗口见图14-8）。不过，在登录Nagios Web界面的过程中，也可能会存在一些出错情况，下面就针对报错故障进行分析。

图14-11　登录后的Nagios Web ISE报错

错误1：出现如图4-11所示的错误提示。

解答：这可能是由于没有关闭SELinux导致。关闭SELinux即可，关闭方法见前文准备部分。

错误2：出现如下报错信息。

```
[root@lss1 nagios]# htpasswd -bc /usr/local/nagios/etc/htpasswd.users olboy  123456
htpasswd:


 cannot create file /usr/local/nagios/etc/htpasswd.users
```

解答：安装Nagios服务器端时缺少了如下命令：make install-daemon-config。

错误3：出现如图14-12所示的Forbidden 403错误。

图14-12　登录后的Nagios Web 403报错

解答：Forbidden 403一般是由服务器端权限引起的，解决方法为执行yum install php-y命令即可。有关Apache和Nginx的403问题，可以参考

老男孩的博客文章“Nginx Forbidden 403多种原因及故障模拟重现”（http://oldboy.blog.51cto.com/2561410/1633952 ）和“apache Forbidden 403问题精彩总结”（http://oldboy.blog.51cto.com/2561410/581383 ）。

到此，Nagios服务器端安装及配置就告一段落。

14.4 Nagios客户端安装

14.4.1 Nagios客户端安装说明

Nagios客户端无需安装LAMP环境，即无需执行下列命令集：

```
yum install gd gd-devel -y
yum install mysql -y
yum install httpd php php-gd -y
```

也无需安装Nagios服务器端软件包，即nagios-3.5.1.tar.gz。

与服务器端相比，Nagios客户端有一些额外的软件包需要安装，如下：

·Class-Accessor-0.31.tar.gz

·Config-Tiny-2.12.tar.gz

·Math-Calc-Units-1.07.tar.gz

·Nagios-Plugin-0.34.tar.gz

·Params-Validate-0.91.tar.gz

·Regexp-Common-2010010201.tar.gz

总体来说，客户端需要安装的命令如下：

```
yum install gcc glibc glibc-common -y
yum install mysql -y
```

软件包如下：

·nagios-plugins-1.4.16.tar.gz

·nrpe-2.12.tar.gz

·Class-Accessor-0.31.tar.gz

·Config-Tiny-2.12.tar.gz

·Math-Calc-Units-1.07.tar.gz

·Nagios-Plugin-0.34.tar.gz

·Params-Validate-0.91.tar.gz

·Regexp-Common-2010010201.tar.gz

·check_iostat

·check_memory.pl

14.4.2　Nagios客户端安装准备

1.准备2台服务器或VM虚拟机

表14-2为Nagios客户端服务器列表。

安装软件环境和安装准备可在两台服务器同时操作。

表14-2　Nagios客户端服务器

管理 IP 地址	角　　色	备　　注
10.0.0.8	web01	被监控的客户端服务器
10.0.0.9	web02	被监控的客户端服务器

2.环境准备与服务器端步骤相同

客户端的yum源安装、关闭Nagios Server端iptables防火墙及SELinux的方法\解决系统时间同步问题的方法均与服务器端一样，这里不再详述，请参考前面的内容。

14.4.3　在Nagios客户端安装软件

1.安装基础系统软件

命令如下：

```
yum install gcc glibc glibc-common -y      #<==升级基础工具包


yum install mysql-server -y           #<==安装这个目的是为了安装

Nagios plugins
                                   生成

check_mysql插件
```

2.上传Nagios相关软件

首先是上传软件包集合：

```
[root@web01 ~]# mkdir -p /home/oldboy/tools/nagios
[root@web01 ~]# cd /home/oldboy/tools/nagios
[root@web01 nagios]# rz -y
#<==本文选择从本地上传软件包集合


oldboy_training_nagios_soft.zip，输入


rz -y命令，然后通过弹出的窗口上传软件包集合
```

oldboy_training_nagios_soft.zip，读者也可以自行下载相关软件，或者加入开篇的

QQ交流群获取软件包

rz waiting to receive.正在开始

 zmodem 传输。

 按

 Ctrl+C 取消。

正在传输

```
 oldboy_training_nagios_soft.zip...
  100%    6322 KB 3161 KB/s 00:
```

00:

02   0 错误

#<==或者你可以自行去下载后面需要的软件包：

地址：

http:

```
// sourceforge.net/projects/nagios/files/
wget http:
```

```
// osdn.dl.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.16.tar.gz
wget http:


// prdownloads.sourceforge.net/sourceforge/nagios/nagios-3.5.1.
tar.gz
```

 技巧：rz是一个很好的从本地上传文件的工具，参数-y表示覆盖和上传文件同名的内容。

为了方便同学们安装，相关包已搜集好一并提供如下：

```
[root@web01 nagios]# unzip -q oldboy_training_nagios_soft.zip
```

在去掉无用的Nagios服务器端软件后，Nagios客户端需要安装的软件如下：

```
[root@web01 nagios]# ls
check_iostat               Math-Calc-Units-1.07.tar.gz
check_memory.pl               Nagios-Plugin-0.34.tar.gz
check_mysql               nagios-plugins-1.4.16.tar.gz
Class-Accessor-0.31.tar.gz     nrpe-2.12.tar.gz
Config-Tiny-2.12.tar.gz          Params-Validate-0.91.tar.gz
libart_lgpl-2.3.17.tar.gz     Regexp-Common-2010010201.tar.gz
```

3.添加Nagios用户

添加方法如下：

```
[root@web01 nagios]# useradd nagios -M -s /sbin/nologin
[root@web01 nagios]# id nagios
uid=500(


nagios)
```

```
 gid=500(
nagios)
 组
=500(
nagios)
```

4.安装nagios-plugins插件

完整的操作命令集如下：

```
yum install perl-devel perl-CPAN openssl-devel -y
tar xf nagios-plugins-1.4.16.tar.gz
cd nagios-plugins-1.4.16
./configure --with-nagios-user=nagios --with-nagios-group=nagios --enable-perl-modul
make
make install
cd ../
```

操作过程如下：

```
[root@web01 nagios-plugins-1.4.16]# yum install perl-devel perl-CPAN openssl-devel -
[root@web01 nagios]# tar xf nagios-plugins-1.4.16.tar.gz
[root@web01 nagios]# cd nagios-plugins-1.4.16
[root@web01 nagios-plugins-1.4.16]# ./configure --with-nagios-user=nagios --with-nag
[root@web01 nagios-plugins-1.4.16]# make
[root@web01 nagios-plugins-1.4.16]# make install
[root@web01 nagios-plugins-1.4.16]# cd ..
```

若出现任何错误，请参考前文Nagios服务器端同一软件的报错总结。

此时检查插件个数：

```
[root@lamp-1 nagios]# ls /usr/local/nagios/libexec/|wc -l
59
```

5.安装Nagios客户端nrpe软件

命令如下：

```
#---------------Dear，我是分隔符


-----------------
tar xf nrpe-2.12.tar.gz
cd nrpe-2.12
./configure
make all
make install-plugin
make install-daemon
#生成


nrpe.cfg
make install-daemon-config
cd ..
```

6.安装其他相关的插件

以下是check_iostat插件需要的依赖包，非必须安装，如果不使用软件包里提供的check_iostat可以不装，虽然命令集很多，但是安装很简单，因此可以批量执行这些命令。

```
#---------------Dear，我是分隔符



-----------------
tar zxf Params-Validate-0.91.tar.gz
cd Params-Validate-0.91
perl Makefile.PL
make
make install
cd ..
#---------------Dear，我是分隔符



-----------------
tar zxf Class-Accessor-0.31.tar.gz
cd Class-Accessor-0.31
perl Makefile.PL
make
make install
cd ..
#---------------Dear，我是分隔符



-----------------
tar zxf Config-Tiny-2.12.tar.gz
cd Config-Tiny-2.12
perl Makefile.PL
make
make install
cd ..
#---------------Dear，我是分隔符



-----------------
tar zxf Math-Calc-Units-1.07.tar.gz
cd Math-Calc-Units-1.07
perl Makefile.PL
make
make install
cd ..
#---------------Dear，我是分隔符



-----------------
tar zxf Regexp-Common-2010010201.tar.gz
cd Regexp-Common-2010010201
perl Makefile.PL
make
make install
cd ..
#---------------Dear，我是分隔符



-----------------
```

```
tar zxf Nagios-Plugin-0.34.tar.gz
cd Nagios-Plugin-0.34
perl Makefile.PL
make
make install
cd ..
#---------------Dear，我是分隔符


-----------------
#for monitor iostat
yum install sysstat -y
#<==这个命令是安装系统性能分析的工具，监控系统性能时程序或脚本会调用这些工具。
```

 提示：上述的安装包完全可以写一个for循环执行，减少相应的代码。

sysstat工具包中包含两类工具，分别为即时查看工具（iostat、mpstat、sar）和累计统计工具（sar）。可以看到，这两类工具中都有sar，也就是说，sar具有这两种功能。可见，sar是sysstat中的核心工具。为了实现sar的累计统计功能，系统必须周期性地记录当时的信息，这是通过调用/usr/lib/sa/中的三个工具实现的：

·sa1：收集并存储每天的系统动态信息到一个二进制的文件中，用作sadc的前端程序。

·sa2：收集每天的系统活跃信息写入总结性的报告，用作sar的前端程序。

·sadc：系统动态数据收集工具，收集的数据被写入一个二进制的文

件中，用作sar工具的后端程序。

这里针对监视物理组件的高级

Linux命令小结如下。

内存：

top free、

vmstat、

mpstat、

iostat、

sar
CPU：

top vmstat、

mpstat、

iostat、

sar
I/O：

vmstat、

mpstat、

iostat、

sar进程：

ipcs、

ipcrm负载：

uptime

7.配置监控内存、磁盘I/O脚本插件

批量部署命令如下：

```
yum install dos2UNIX -y
/bin/cp /home/oldboy/tools/nagios/check_memory.pl /usr/local/nagios/libexec
/bin/cp /home/oldboy/tools/nagios/check_iostat /usr/local/nagios/libexec
chmod 755 /usr/local/nagios/libexec/check_memory.pl
chmod 755 /usr/local/nagios/libexec/check_iostat
dos2UNIX /usr/local/nagios/libexec/check_memory.pl
dos2UNIX /usr/local/nagios/libexec/check_iostat
```

执行部署的过程：

```
[root@web01 nagios]# /bin/cp /home/oldboy/tools/nagios/check_memory.pl /usr/local/na
[root@web01 nagios]# /bin/cp /home/oldboy/tools/nagios/check_iostat /usr/local/nagio
#<==以上两命令是将事先写好的脚本（
```

check_memory.pl和

check_iostat）放到

Nagios脚本目录下。

```
[root@web01 nagios]# chmod 755 /usr/local/nagios/libexec/check_memory.pl
[root@web01 nagios]# chmod 755 /usr/local/nagios/libexec/check_iostat
#<==以上两命令是授权使脚本可执行



[root@web01 nagios]# dos2UNIX /usr/local/nagios/libexec/check_memory.pl
dos2UNIX:


 converting file /usr/local/nagios/libexec/check_memory.pl to UNIX format ...
[root@web01 nagios]# dos2UNIX /usr/local/nagios/libexec/check_iostat
dos2UNIX:


 converting file /usr/local/nagios/libexec/check_iostat to UNIX format ...
#<==使用


dos2UNIX使之格式为


UNIX的脚本格式，否则执行可能出错。
```

注意：当前路径为软件包及脚本所在的路径，即/home/oldboy/tools/nagios。

14.4.4　配置Nagios客户端nrpe服务

可通过以下命令配置客户端的nrpe.cfg。

```
[root@web01 nagios]# cd /usr/local/nagios/etc
[root@web01 etc]# sed -n '79p' nrpe.cfg
allowed_hosts=127.0.0.1
[root@web01 etc]# sed -i 's#allowed_hosts=127.0.0.1#allowed_hosts=127.0.0.1,



10.0.0.7#g' nrpe.cfg
[root@web01 etc]# sed -n '79p' nrpe.cfg
allowed_hosts=127.0.0.1,



10.0.0.7
```

加入可以监控当前client的Nagios Server端的IP，如下：

```
allowed_hosts=127.0.0.1,



10.0.0.7
#<==再次提醒，



allowed_hosts=后面的



ip为



Nagios服务器端的



ip，不是客户端的。
```

```
#<==如果机器大于

500台做集群或分布式监控时，可以使用多个

Nagios Server ，

IP要用逗号隔开。
```

然后在命令模式下执行shift+g命令到结尾（shift+g为vi里切到结尾的快速操作命令请牢记类似的命令），并进行如下修改。

第一步，注释掉或者干脆删除199~203行，即下面几行：

```
#command[check_users]=/usr/local/nagios// libexec/check_users -w 5 -c 10
#command[check_load]=/usr/local/nagios// libexec/check_load -w 15,

10,

5 -c 30,

25,

20
#command[check_hda1]=/usr/local/nagios// libexec/check_disk -w 20% -c 10% -p /dev/hd
#command[check_zombie_procs]=/usr/local/nagios// libexec/check_procs -w 5 -c 10 -s Z
#command[check_total_procs]=/usr/local/nagios// libexec/check_procs -w 150 -c 200
```

第二步，在下面新添加要监控的内容：

```
command[check_load]=/usr/local/nagios/libexec/check_load -w 15,
```

```
10,


5 -c 30,


25,


20
command[check_mem]=/usr/local/nagios/libexec/check_memory.pl -w 10% -c 3%
command[check_disk]=/usr/local/nagios/libexec/check_disk -w 15% -c 7% -p /
command[check_swap]=/usr/local/nagios/libexec/check_swap -w 20% -c 10%
command[check_iostat]=/usr/local/nagios/libexec/check_iostat -w 6 -c 10
```

上面依次为对负载、内存、硬盘、虚拟内存、磁盘I/O进行监控，这些都是本地的服务（我们一般通过nrpe去客户端执行脚本插件获取信息，这样的模式称为被动监控，与nsca的客户端主动提交结果不冲突），由Nagios服务器端通过nrpe插件定时在client的nrpe服务上获取信息。

之后，启动Nagios client nrpe守护进程。

```
[root@web01 etc]# /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
```

现在，检查启动结果。

```
[root@web01 etc]# netstat -lntup|grep nrpe
tcp        0      0 0.0.0.0:


5666          .0.0.0:
```

```
*     LISTEN      1609/nrpe
[root@web01 etc]# ps -ef|grep nrpe|grep -v grep
nagios 1609  1  0 21:


21   00:


00:


00 /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
```

技巧：重启client端Nagios nrpe服务的组合命令如下：

```
[root@web01 etc]# pkill nrpe
[root@web01 etc]# /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
```

这时，可以将nrpe加入开机自启动了，命令如下：

```
[root@web01 etc]# echo  "#nagios nrpe process cmd by oldboy 2015-6-7" >> /etc/rc.loc
[root@web01 etc]# echo  "/usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.c1
[root@web01 etc]# tail -2 /etc/rc.local
#nagios nrpe process cmd by oldboy 2015-6-7
/usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
```

如果你够勤快，也可以写一个简单的客户端启动脚本来停止和关闭nrpe服务。

重要提醒：客户端的nrpe.cfg配置文件，最好保留一份到计算机上，这样以后在其他的机器上装nrpe时，直接上传即可，就省得费劲修改了。

关于/usr/local/nagios/etc/nrpe.cfg配置文件的详细说明见14.9节。

到此，web01、web02客户端的安装配置部分全部结束。在此过程中，也会存在一些安装报错情况，一起来分析。

错误1：安装nrpe时执行make all命令报错。

```
checking for SSL headers... configure:

 error:

 Cannot find ssl headers
```

解答：提示缺少ssl，通过如下命令可解决。

```
yum -y install openssl openssl -devel
```

错误2：安装nagios-plugins时执行make命令报错。

```
check_http.c:

 In function 'process_arguments':


check_http.c:

312:

 error:

 'ssl_version' undeclared (
```

```
first use in this function)

check_http.c:

312:

 error:

  (

Each undeclared identifier is reported only once
check_http.c:

312:

 error:

 for each function it appears in.)

make[2]:

 *** [check_http.o] Error 1
make[2]:

 Leaving directory `/root/tools/nagios-plugins-1.4.16/plugins'
make[1]:

 *** [all-recursive] Error 1
make[1]:

 Leaving directory `/root/tools/nagios-plugins-1.4.16'
```

```
make:

 *** [all] Error 2
```

问题解决：

根据报错提示执行yum install openssl openssl-devel perl-devel-y命令，删除已经解压的nagios-plugins-1.4.16目录，重新解压编译安装。

14.5 Nagios服务器端监控

14.5.1 Nagios服务器端监控基础介绍

1.Nagios服务器端目录结构

Nagios服务器端安装后的目录结构如下：

```
[root@nagios-server ~]# ls -l /usr/local/nagios/
total 32
drwxrwxr-x          2 nagios nagios 4096 Jun 27 17:


48 bin
drwxrwxr-x          3 nagios nagios 4096 Jun 27 17:


48 etc
drwxr-xr-x          2 root   root   4096 Jun 27 17:


34 include
drwxrwxr-x          2 nagios nagios 4096 Jun 27 17:


48 libexec
drwxr-xr-x          5 root   root   4096 Jun 27 17:


34 perl
drwxrwxr-x          2 nagios nagios 4096 Jun 27 08:


59 sbin
drwxrwxr-x          11 nagios nagios 4096 Jun 27 17:


34 share
drwxrwxr-x          5 nagios nagios 4096 Jun 27 21:
```

为方便大家查看和理解，下面采用表格的形式来说明，见表14-3。

表14-3　Nagios服务器端目录说明

目录名称	说　明
bin	bin 下为 Nagios 相关命令 [root@nagios-server nagios] # tree bin/ bin/ |-- nagios |-- nagiostats `-- nrpe 0 directories, 3 files
etc*****	etc 下 Nagios 的配置文件及目录信息 [root@nagios-server nagios] # tree etc/ etc/ |-- cgi.cfg |-- htpasswd.users |-- nagios.cfg #==> nagios 主配置，相当于 httpd.conf/nginx.conf。 |-- nrpe.cfg #==> 客户端的配置文件 |-- objects #==> 相当于 Apache 的 extra 包含目录。 | |-- commands.cfg | |-- contacts.cfg | |-- localhost.cfg | |-- printer.cfg | |-- switch.cfg | |-- templates.cfg | |-- timeperiods.cfg | `-- windows.cfg `-- resource.cfg 1 directory, 13 files
libexec	libexec 为所有插件的目录路径 [root@nagios-server nagios] # tree libexec/ libexec/ |-- check_apt |-- check_breeze |-- check_by_ssh |-- check_clamd -> check_tcp |-- check_cluster

（续）

目录名称	说　明
libexec	|-- check_dhcp |-- check_dig |-- check_disk ... 省略部分 ...
sbin	#sbin [root@nagios-server nagios] # tree sbin/ sbin/ |-- avail.cgi |-- cmd.cgi |-- config.cgi |-- extinfo.cgi |-- histogram.cgi |-- history.cgi |-- notifications.cgi |-- outages.cgi |-- showlog.cgi |-- status.cgi |-- statusmap.cgi |-- statuswml.cgi |-- statuswrl.cgi |-- summary.cgi |-- tac.cgi `-- trends.cgi 0 directories, 16 files
share	share 为 Nagios 界面展示的 PHP 程序等内容的目录 [root@nagios-server nagios] # tree -L 1 share share |-- config.inc.php |-- contexthelp |-- docs |-- images |-- includes |-- index.php |-- locale |-- main.php |-- media |-- robots.txt |-- side.php |-- ssi `-- stylesheets 8 directories, 5 files

（续）

目录名称	说　明
var	var为Nagios数据及日志的目录 [root@nagios-server nagios]# tree var var |-- archives |-- nagios.lock |-- nagios.log |-- objects.cache |-- retention.dat |-- rw |　　`-- nagios.cmd |-- spool |　　`-- checkresults |　　　　`-- checkS6D3wR `-- status.dat 4 directories, 7 files

所有客户端本地服务的监控都是通过执行libexec目录下的插件来实现的，当然，如果开启了snmp，Nagios服务器端也可主动抓取。

2.Nagios服务器端核心配置文件

Nagios主配置文件为nagios.cfg，默认在/usr/local/nagios/etc目录下，/usr/local/nagios/etc目录下有个objects（类似Apache的extra）目录，里面放的是主配置文件nagios.cfg包含的其他Nagios配置文件。查看命令如下：

```
[root@nagiosserver etc]# tree
|-- cgi.cfg
|-- nagios.cfg
|-- nrpe.cfg
|-- objects
|   |-- commands.cfg
|   |-- contacts.cfg
|   |-- localhost.cfg
|   |-- printer.cfg
|   |-- switch.cfg
```

```
|   |-- templates.cfg
|   |-- timeperiods.cfg
|   `-- windows.cfg
`-- resource.cfg
[root@nagiosserver etc]# tail -10 resource.cfg
# Sets $USER1$ to be the path to the plugins
$USER1$=/usr/local/nagios/libexec
# Sets $USER2$ to be the path to event handlers
#$USER2$=/usr/local/nagios/libexec/eventhandlers
# Store some usernames and passwords (


hidden from the CGIs)



#$USER3$=someuser
#$USER4$=somepassword
[root@nagios-server nagios]# tree etc/objects/
etc/objects/
|-- commands.cfg
|-- contacts.cfg
|-- localhost.cfg
|-- printer.cfg
|-- switch.cfg
|-- templates.cfg
|-- timeperiods.cfg
`-- windows.cfg
0 directories,


 8 files
```

在nagios.cfg中既可以指定单独包含一个cfg文件，也可以指定包含一个目录，即该目录下所有的cfg文件都会包含进来。

为了让目录结构看起来更清晰，以及批量部署服务的需要，我们把主配置文件包含的配置文件修改为表14-4所示的形式。

表14-4　Nagios配置文件说明

配置文件名称	说　　明
commands.cfg	存放 Nagios 命令相关配置（也可指定 commands 目录），这里的命令不是系统命令，而是把 Nagios 里定义的命令和 Linux 系统里的插件命令相关联的一个文件
services.cfg	存放具体被监控的服务相关配置内容（上百台以上可指定 services 目录)(默认不存在)
hosts.cfg	存放具体被监控的主机相关配置内容（上百台以上可指定 hosts 目录)(默认不存在)
contacts.cfg	存放报警联系人相关配置的文件
timeperiods.cfg	存放报警周期时间等相关配置内容
templates.cfg	模版配置文件，模板的存在是为了方便地配置服务配置。类似 Shell 里的函数功能

3.配置主配置文件nagios.cfg

首先，在nagios.cfg文件中找到cfg_file部分，进行如下设置：

```
[root@nagios-server ~]# vi /usr/local/nagios/etc/nagios.cfg +34
#<==增加如下主机和服务的配置文件



cfg_file=/usr/local/nagios/etc/objects/hosts.cfg
cfg_file=/usr/local/nagios/etc/objects/services.cfg
cfg_dir=/usr/local/nagios/etc/objects/services
#这是为备用增加的一个



service目录，初学者可以忽略此行，使用目录的优点很多，在目录下的文件只要符合



*.cfg就可以被



Nagios加载。使用脚本批量部署时可非常方便地随机命名配置文件
```

然后注释掉如下一行：

```
# Definitions for monitoring the local （
```

```
Linux)


 host
#cfg_file=/usr/local/nagios/etc/objects/localhost.cfg
#<==localhost.cfg这个配置为监控


Nagios服务器端本地服务的配置文件，注释掉它，然后统一监控
```

操作完毕，保存nagios.cfg。接着，根据已有数据生成hosts.cfg主机文件，命令集如下：

```
cd /usr/local/nagios/etc/objects/
head -51 localhost.cfg >hosts.cfg
chown nagios.nagios /usr/local/nagios/etc/objects/hosts.cfg
```

操作过程如下：

```
[root@nagios-server ~]# cd /usr/local/nagios/etc/objects/
[root@nagios-server objects]# head -51 localhost.cfg >hosts.cfg
[root@nagios-server objects]# chown nagios.nagios /usr/local/nagios/etc/objects/host
```

然后生成新的空的services.cfg服务文件，命令集如下：

```
touch services.cfg #<==暂时留空


chown -R nagios.nagios services.cfg
```

操作过程如下：

```
[root@nagios-server objects]# touch services.cfg
[root@nagios-server objects]# chown -R nagios.nagios services.cfg
```

最后，生成服务的配置文件目录，所有放到此目录下的配置（*.cfg）都会自动被包含到主配置文件中生效。命令集如下：

```
mkdir services
chown -R nagios.nagios services
```

操作过程如下：

```
[root@nagios-server objects]# mkdir services
[root@nagios-server objects]# chown -R nagios.nagios services
```

现在，检查生成结果：

```
[root@nagios-server objects]# ls -lrt
total 56
-rw-rw-r-- 1 nagios nagios 10812 Jun 27 09:


01 templates.cfg
-rw-rw-r-- 1 nagios nagios  4019 Jun 27 09:


01 windows.cfg
-rw-rw-r-- 1 nagios nagios  3208 Jun 27 09:


01 timeperiods.cfg
-rw-rw-r-- 1 nagios nagios  3124 Jun 27 09:


01 printer.cfg
-rw-rw-r-- 1 nagios nagios  5403 Jun 27 09:


01 localhost.cfg
-rw-rw-r-- 1 nagios nagios  7716 Jun 27 09:
```

```
01 commands.cfg
-rw-rw-r-- 1 nagios nagios  3293 Jun 27 09:


01 switch.cfg
-rw-rw-r-- 1 nagios nagios  2165 Jun 27 09:


21 contacts.cfg
-rw-r--r-- 1 nagios nagios  1870 Jun 27 21:


53 hosts.cfg
-rw-r--r-- 1 nagios nagios     0 Jun 27 21:


54 services.cfg
drwxr-xr-x 2 nagios nagios  4096 Jun 27 21:


55 services
```

4.Nagios监控模式定义及监控模式选择

根据监控的行为，将Nagios的监控分为主动监控和被动监控（即nrpe半被动和nsca全被动），nsca全被动暂不详述，下面先来看看什么是主动监控和半被动监控。

·主动监控：把像URL监控一样由Nagios服务器端发出请求的主动探测监控方式，定义为主动监控方式，也就是说不需要在客户端安装任何插件。当然，主动监控模式也可以配置成被动模式。

·（半）被动监控：由Nagios服务器端通过nrpe插件定时去连接client的nrpe服务获取信息，并发回到Nagios服务器端的监控称之为半被动监

控，这类监控通常是针对本地资源的，比如负载、内存、硬盘、虚拟内存、磁盘I/O、温度、风扇转速等，而非系统对外提供的服务，只要安装了类似nrpe的插件方式的监控，都认为是半被动监控。

如何选择主动监控模式和（半）被动监控模式？

1）对于本地的资源性能，一般用被动监控模式（NRPE）。例如，对负载、内存、硬盘、虚拟内存、磁盘I/O、温度、风扇等的监控（我们也可以通过snmp实现监控部分系统资源）。

2）对于Web服务、数据库服务这种能对外提供服务的，一般用主动模式，例如：监控httpd、sshd、mysqld、rsyncd等服务。

主动模式和被动模式是相对的，并且是可以互相转换的，即主动模式的服务，可以改成被动模式的，被动模式的服务有时也可以改为主动模式的。


提示：本书下文把nrpe的监控统称为被动监控模式。

14.5.2　配置Nagios服务器端监控项

1.定义要监控的Nagios客户端主机

hosts.cfg一般用来存放Nagios要监控的主机相关配置，下面是hosts.cfg中的主机定义部分的配置参数详解。

```
define host {                           #<==define host为关键字，意思是

                                        定义主机，主机内容用一对大括号括起来

        use                 linux-server   #<==定义主机使用的模版，具体参见

                                        templates.cfg
        host_name               08-web01      #<==直接定义主机名称，根据服务功

                                        能可随意定义

        alias               08-web01      #<==直接定义主机别名，同上

        address               10.0.0.8      #<==直接定义被监控服务器的

IP
        check_command          check-host-alive      #<==检测主机存活命令，来自

com-
```

mands.cfg
max_check_attempts 3 #<==故障后，最大尝试检测次数

normal_check_interval 2 #<==正常的检查间隔，默认单位分钟

retry_check_interval 2 #<==故障后重试的检查间隔，默认单位分钟

check_period 24x7 #<==检查周期

24x7，具体参见

timeperiods.cfg
notification_interval 300 #<==故障后，两次报警的通知间隔，默认单位分钟

notification_period 24x7 #<==一天之内通知的周期。比如全天还是半天等，

具体参见

timeperiods.cfg
notification_options d，

u，

r #<==主机状态通知选项

d-down，

u-unreache-
able，

```
r-recovery
        contact_groups          admins      #<==报警到


admins 用户组。在


contacts.
                                cfg里定义


}                                #<==这个括号不能丢
```

主机也可以只配置关键选项，多数选项可采取linux-server模版的默认值，有能力的同学还可以先调整linux-server模版，然后让所有机器统一采用这种默认值，这看起来更加简单方便。示例如下：

```
# web01
define host {
        use                 linux-server
        host_name                 08-web01
        alias               08-web01
        address               10.0.0.8
}
```

下面将采用上述简化版的配置，服务器多时可批量部署，当然也可以安装图形配置界面Centreon、NagiosQL，但不推荐。

现在开始配置hosts.cfg，增加web01、web02主机。此时，需要在hosts.cfg里添加所有需要监控的客户端主机和主机组（HOST GROUP）。示例如下：

```
[root@nagios-server ~]# cd /usr/local/nagios/etc/objects/
```

#<==进入到辅助配置文件的目录

[root@nagios-server objects]# cat -n hosts.cfg #<==默认情况下

hosts.cfg是不存

在的，需要手工创建

```
1# Define a host for the local machine
2define host{
3     use          linux-server          #<==主机模板，在
```

templates.cfg

里定义的，很多此处没有的参数就都

在

templates.cfg里

```
4     host_name     web01               #<==第一个
```

Nagios客户端主机名

```
5     alias          web01               #<==第一个
```

Nagios客户端别名

```
6     address          10.0.0.8          #<==第一个
```

Nagios客户端

IP

```
     7     }
     8define host{
     9     use          linux-server
    10     host_name     web02
    11     alias          web02
    12     address          10.0.0.9
    13     }
```

#提示

#<==host_name web01 不同的主机名这行需要改改

#<==alias web01 不同的主机别名这行需要改改

#<==address 10.0.0.8 客户端服务器地址

```
    14
    15#################################################
    16
    17# Define an optional hostgroup for Linux machines
    18
    19define hostgroup{
    20          hostgroup_name  linux-servers
    21          alias            Linux Servers
    22          members            web01,
```

web02 #<==把前面定义的每一个

Nagios客

户端主机名在这里用逗号隔开列出来

```
    23          }
```

2.配置services.cfg，定义要监控的资源服务

services.cfg文件是配置监控服务的，是Nagios最重要的配置文件之一，如果服务器数量比较少（50台以内），则需要监控的大部分服务配置都可以在这里面添加。这个配置文件默认是不存在的，需要人为定义。

先来看看services.cfg配置文件的service配置参数，详细说明如下：

```
define service {                        #<==define service为关键字，意思是定义

                                        一个服务，服务内容用一对大括号括起来

        use         generic-service             #<==定义该服务使用的模版，具体参见

templates.
                                cfg
        host_name   08-web01            #<==被监控的主机名，来自

hosts.cfg，可在

                                        hosts.cfg中自定义

        service_description   Current Load      #<==报警服务描述，根据内容取有意义的名称

        check_command     check_nrpe!
```

check_load #<==检查服务的命令，这个很关键，注意被动

服务的监控均由

check_nrpe调用

max_check_attempts 2 #<==尝试检查的最大次数

normal_check_interval 4 #<==正常状态检查时间间隔，每

4分钟去检查

一次是否正常

retry_check_interval 4 #<==重试检查时间间隔，默认单位是分

check_period 24x7 #<==检查的周期，

24x7仅仅是个字符串而已

notification_interval 1440 #<==通知的间隔，即

1440分通知一次

notification_period 24x7 #<==通知的周期，这个参数来自

```
timeperiods.
                                cfg中的配置，例如可以定义半夜不报警到短信手机

        notification_options w,

u,

c,

r      #<==要通知的服务状态选项

 w-warning,

u-
                               unknown,

c-critical,

r-recovery。

        contact_groups           admins      #<==要通知的用户组，其定义来自于

contacts.
                               cfg
        process_perf_data        1      #<==PNP出图记录数据相关。

}                                 #<==define service结尾的大括号
```

下面来看看配置监控客户端本地资源的示例。

磁盘分区监控（被动监控）如下：

```
define service {
        use                 generic-service
        host_name                08-web01  #<==这里可以指定多台机器，通过逗号隔开



        service_description     Disk Partition
        check_command           check_nrpe!


check_disk
}
```

swap监控（被动监控）如下：

```
define service {
        use                     generic-service
        host_name                     08-web01
        service_description           Swap Useage
        check_command                check_nrpe!


check_swap
}
```

内存监控（被动监控）如下：

```
define service {
        use                     generic-service
        host_name                     08-web01
        service_description           MEM Useage
        check_command                check_nrpe!


check_mem
}
```

通过使用模板templates.cfg，可以把上面的配置简写为如下几行。

系统负载监控（被动监控）：

```
define service {
        use                     generic-service
        host_name                       08-web01
        service_description             Current Load
        check_command               check_nrpe!


check_load
}
```

磁盘I/O监控（被动监控）：

```
define service {
        use                     generic-service
        host_name                       08-web01
        service_description             Disk Iostat
        check_command               check_nrpe!


check_iostat!


5!


11
}
```

ping监控（主动监控）：

```
define service {
        use                     generic-service
        host_name                       08-web01
        service_description             PING
        check_command               check_ping!


100.0,
```

```
20%!


500.0,


60%
}
```

以下是客户端本地资源监控的实际配置。

```
define service {
        use                     generic-service
        host_name                       08-web01
        service_description             Disk Partition
        check_command                 check_nrpe!


check_disk
}
define service {
        use                     generic-service
        host_name                       08-web01
        service_description             Swap Useage
        check_command                 check_nrpe!


check_swap
}
define service {
        use                     generic-service
        host_name                       08-web01
        service_description             MEM Useage
        check_command                 check_nrpe!


check_mem
}
define service {
        use                     generic-service
        host_name                       08-web01
        service_description             Current Load
        check_command                 check_nrpe!


check_load
}
define service {
        use                     generic-service
        host_name                       08-web01
```

```
        service_description          Disk Iostat
        check_command               check_nrpe!


check_iostat!


5!


11
}
define service {
        use                     generic-service
        host_name                       08-web01
        service_description             PING
        check_command               check_ping!


100.0,


20%!


500.0,


60%
}
```

 提示：

1）上述配置中的check_nrpe是服务器端的插件（是commands.cfg里预先定义的命令名），负责和客户端的nrpe进程交流并执行check_nrpe叹号后的插件，所以，check_nrpe！check_load配置中的check_load是客户端的插件名，是在与客户端的nrpe进程对应的配置nrpe.cfg里定义的命令名。

2）以上services.cfg中添加了对磁盘分区、Load、Mem、Swap、磁盘IO、ping的监控。

Nagios软件默认没有提供客户端的内存和I/O插件，但本文在配置时已经复制进去了，因此，只需在commands.cfg里配置即可，详细请看下文的commands.cfg说明。

3.调试hosts.cfg和service.cfg的所有配置

若在调试前执行如下检查Nagios语法的命令：

```
[root@nagios-server objects]# /etc/init.d/nagios checkconfig #<==修改了调试输出配置


Running configuration check...
Nagios Core 3.5.1
Copyright (


c)


 2009-2011 Nagios Core Development Team and Community Contributors
Copyright (


c)


 1999-2009 Ethan Galstad
Last Modified:


 08-30-2013
License:
```

```
 GPL
Website:


 http:


// www.nagios.org
Reading configuration data...
  Read main config file okay...
Processing object config file '/usr/local/nagios/etc/objects/commands.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/contacts.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/timeperiods.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/templates.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/hosts.cfg'...
Processing object config file '/usr/local/nagios/etc/objects/services.cfg'...
Processing object config directory '/usr/local/nagios/etc/objects/services'...
  Read object config files okay...
Running pre-flight check on configuration data...
Checking services...
Error:


 Service check command 'check_nrpe' specified in service 'Current Load' for host 'we



Error:


 Service check command 'check_nrpe' specified in service 'Disk Iostat' for host 'web



Error:


 Service check command 'check_nrpe' specified in service 'Disk Partition' for host '



Error:


 Service check command 'check_nrpe' specified in service 'MEM Useage' for host 'webC



Error:
```

```
 Service check command 'check_nrpe' specified in service 'Swap Useage' for host 'web

Error:

 Service check command 'check_nrpe' specified in service 'Current Load' for host 'we

Error:

 Service check command 'check_nrpe' specified in service 'Disk Iostat' for host 'web

Error:

 Service check command 'check_nrpe' specified in service 'Disk Partition' for host '

Error:

 Service check command 'check_nrpe' specified in service 'MEM Useage' for host 'web0

Error:

 Service check command 'check_nrpe' specified in service 'Swap Useage' for host 'web



        Checked 12 services.
Checking hosts...
        Checked 2 hosts.
Checking host groups...
        Checked 1 host groups.
Checking service groups...
        Checked 0 service groups.
Checking contacts...
        Checked 1 contacts.
Checking contact groups...
```

```
        Checked 1 contact groups.
Checking service escalations...
        Checked 0 service escalations.
Checking service dependencies...
        Checked 0 service dependencies.
Checking host escalations...
        Checked 0 host escalations.
Checking host dependencies...
        Checked 0 host dependencies.
Checking commands...
        Checked 24 commands.
Checking time periods...
        Checked 5 time periods.
Checking for circular paths between hosts...
Checking for circular host and service dependencies...
Checking global event handlers...
Checking obsessive compulsive processor commands...
Checking misc settings...
Total Warnings:



 0
Total Errors:



   10
***> One or more problems was encountered while running the pre-flight check...
    Check your configuration file (



s)



 to ensure that they contain valid
    directives and data defintions.  If you are upgrading from a previous
    version of Nagios,



 you should be aware that some variables/definitions
    may have been removed or modified in this version.  Make sure to read
    the HTML documentation regarding the config files,



 as well as the
    'Whats New' section to find out what has changed.
CONFIG ERROR!



  Check your Nagios configuration.
```

根据错误提示，我们可以知道，是check_nrpe插件没有定义导致

的。来看一下解决方法。

首先，需要在commands.cfg中加入check_nrpe的插件配置。

```
[root@nagios-server objects]# vi commands.cfg     #<==进入后按

shift+g键切到结尾

                                加入下面内容

# 'check_nrpe' command definition
define command{
        command_name     check_nrpe
        command_line     $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
        }
```

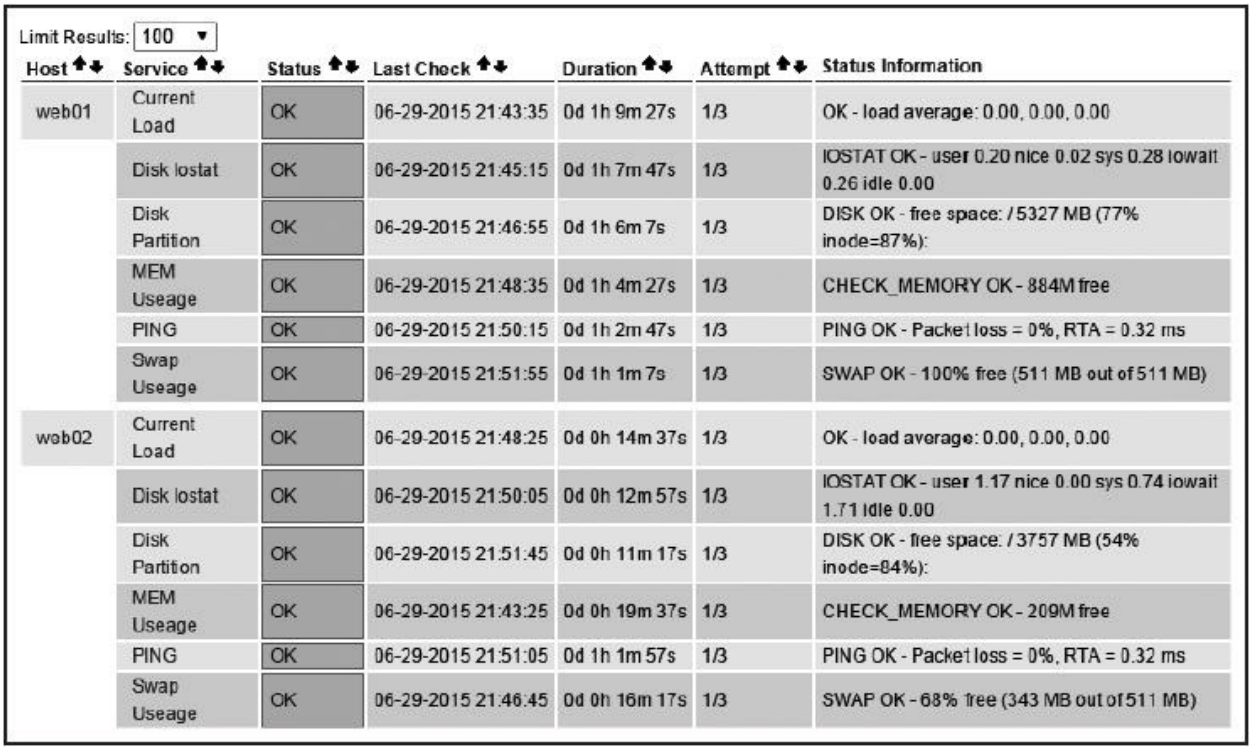
此时重新执行检查语法的命令：

```
[root@nagios-server objects]# /etc/init.d/nagios checkconfig
#<==这是

Nagios启动脚本里检查语法的命令，默认情况只能检查是否有错，但不显示错误的详细信息

Total Warnings:

     0
Total Errors:

     0
```

此时，警告和错误都为0，这表示已经OK了，一般来讲警告可以忽略，错误必须要解决掉，否则，无法继续，调试完成后启动Nagios服务。

```
[root@nagios-server objects]# /etc/init.d/nagios start
Starting nagios:


 done.
#<==如果


Nagios已经启动了，就执行


/etc/init.d/nagios reload，修改配置文件可以不用


restart
```

此时可以看到自己配置的本地各系统状态的监控成果了，如图14-13所示。



Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
web01	Current Load	OK	06-29-2015 21:43:35	0d 1h 9m 27s	1/3	OK - load average: 0.00, 0.00, 0.00
	Disk Iostat	OK	06-29-2015 21:45:15	0d 1h 7m 47s	1/3	IOSTAT OK - user 0.20 nice 0.02 sys 0.28 iowait 0.26 idle 0.00
	Disk Partition	OK	06-29-2015 21:46:55	0d 1h 6m 7s	1/3	DISK OK - free space: / 5327 MB (77% inode=87%):
	MEM Useage	OK	06-29-2015 21:48:35	0d 1h 4m 27s	1/3	CHECK_MEMORY OK - 884M free
	PING	OK	06-29-2015 21:50:15	0d 1h 2m 47s	1/3	PING OK - Packet loss = 0%, RTA = 0.32 ms
	Swap Useage	OK	06-29-2015 21:51:55	0d 1h 1m 7s	1/3	SWAP OK - 100% free (511 MB out of 511 MB)
web02	Current Load	OK	06-29-2015 21:48:25	0d 0h 14m 37s	1/3	OK - load average: 0.00, 0.00, 0.00
	Disk Iostat	OK	06-29-2015 21:50:05	0d 0h 12m 57s	1/3	IOSTAT OK - user 1.17 nice 0.00 sys 0.74 iowait 1.71 idle 0.00
	Disk Partition	OK	06-29-2015 21:51:45	0d 0h 11m 17s	1/3	DISK OK - free space: / 3757 MB (54% inode=84%):
	MEM Useage	OK	06-29-2015 21:43:25	0d 0h 19m 37s	1/3	CHECK_MEMORY OK - 209M free
	PING	OK	06-29-2015 21:51:05	0d 1h 1m 57s	1/3	PING OK - Packet loss = 0%, RTA = 0.32 ms
	Swap Useage	OK	06-29-2015 21:46:45	0d 0h 16m 17s	1/3	SWAP OK - 68% free (343 MB out of 511 MB)

图14-13　监控主机针对本地各系统状态监控的成果

事实上，很可能有不少读者配置到此处时会报错，一起来看看都会碰到哪些错误。

当打开浏览器，查看左边的Hosts和Services时，如果无服务内容，而是出现下面英文错误提示：

It appears as though you do not have permission to view information for any of the services you requested...

If you believe this is an error，check the HTTP server authentication requirements for accessing this CGIand check the authorization options in your CGI configuration file.

则表示登录Web端的用户oldboy没有被许可查看这些服务资源，可按照如下方法解决上面的问题。

```
[root@nagios-server objects]# cd /usr/local/nagios/etc
[root@nagios-server etc]# grep ^"authorized_for" cgi.cfg
authorized_for_system_information=nagiosadmin
authorized_for_configuration_information=nagiosadmin
authorized_for_system_commands=nagiosadmin
authorized_for_all_services=nagiosadmin
authorized_for_all_hosts=nagiosadmin
authorized_for_all_service_commands=nagiosadmin
authorized_for_all_host_commands=nagiosadmin
```

上述authorized开头的行对应的都是不同权限，可以看到，在结尾的许可处用户都是nagiosadmin。此时，把在前文中建立的Nagios Web登录用户oldboy加到每一个许可项的后面，注意两者用逗号隔开。

```
[root@nagios-server etc]# sed -i 's#nagiosadmin#oldboy#g' cgi.cfg
[root@nagios-server etc]# grep ^"authorized_for" cgi.cfg  authorized_for_system_infc
authorized_for_configuration_information=oldboy
authorized_for_system_commands=oldboy
authorized_for_all_services=oldboy
authorized_for_all_hosts=oldboy
authorized_for_all_service_commands=oldboy
authorized_for_all_host_commands=oldboy
```

 提示：

1）理想情况最好换掉默认管理员用户nagiosadmin，替换成自己的oldboy等，然后根据需求加适合自己的权限，比如，给别人看只给浏览权限就可以了。具体见cgi.cfg。

2）遇到调试问题注意查看/usr/local/nagios/var/nagios.log，这点很重

要。高手都这么做的。

3）vi命令行替换。使用：g/nagiosadmin/s//oldboy/g或：%s/oldboy/test/g或sed替换。

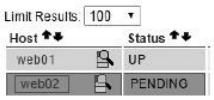
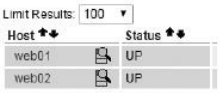
4）reload nagios命令为：/etc/init.d/nagios reload，此处为cgi.cfg的修改，也可以不执行reload。

现在，重新加载配置使得修改生效。

```
[root@nagios-server etc]# /etc/init.d/nagios reload
Running configuration check...done.
Reloading nagios configuration...done
```

正常监控后的效果图见表14-5。

表14-5　主机监控效果图

刚配置完的主机监控图	完全正常的主机监控图
 “PENDING”意思是服务器还没确定其状态。	 “UP”意思是服务器正常。

下面看看服务监控效果图。刚配置完的主机监控图如图14-14所示。

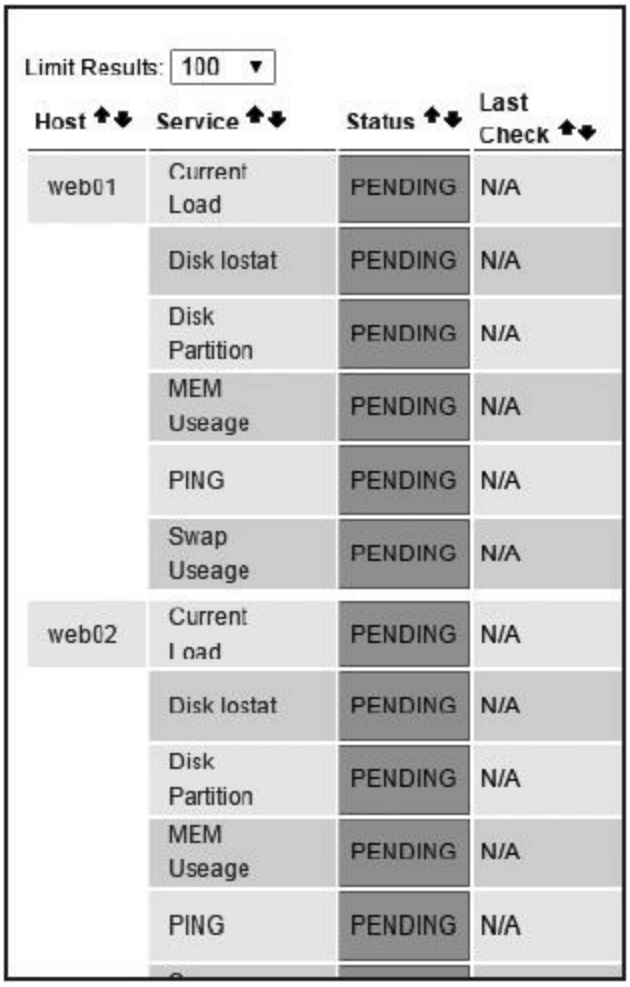


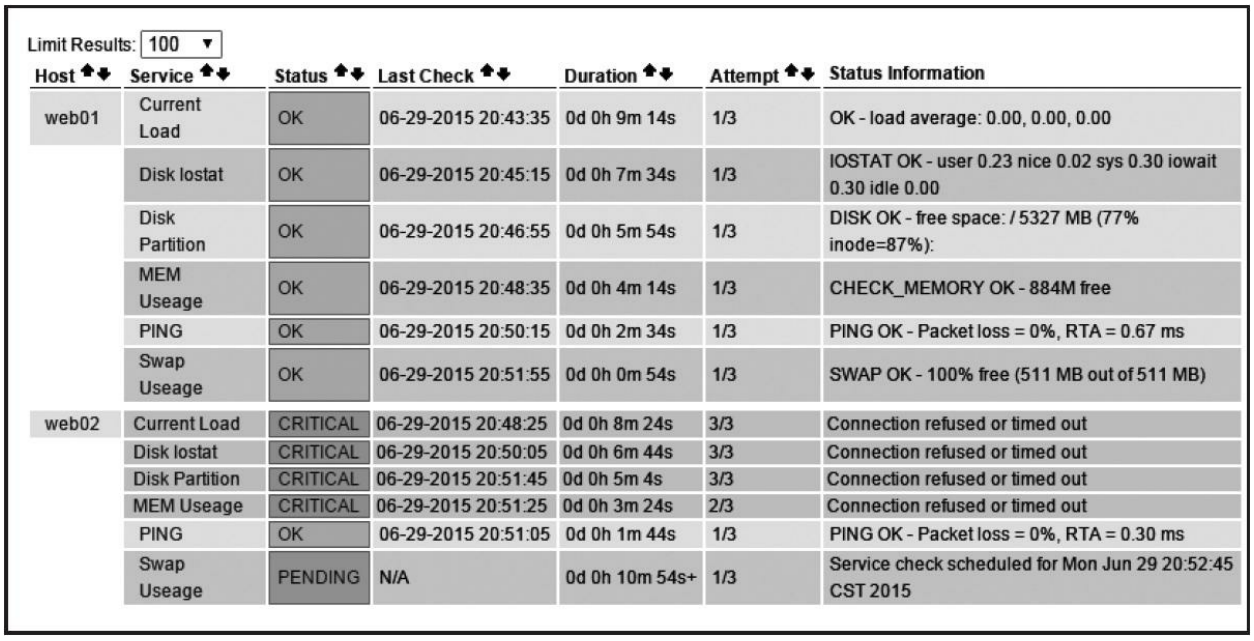
图14-14　新增主机重启未确定时的显示图

这里“PENDING”的意思是服务器还没确定其状态，过一会再刷新查看，状态如图14-15所示。

其中绿色的表示一切正常，红色背景的表示有故障，在Web02还没开启nrpe服务的情况下，开启之后整体效果如图14-16所示。

如果出现如图14-17所示的报错，对于第①个问题“NRPE：Unable to read output”，解决办法见下文。第②个问题“NRPE：

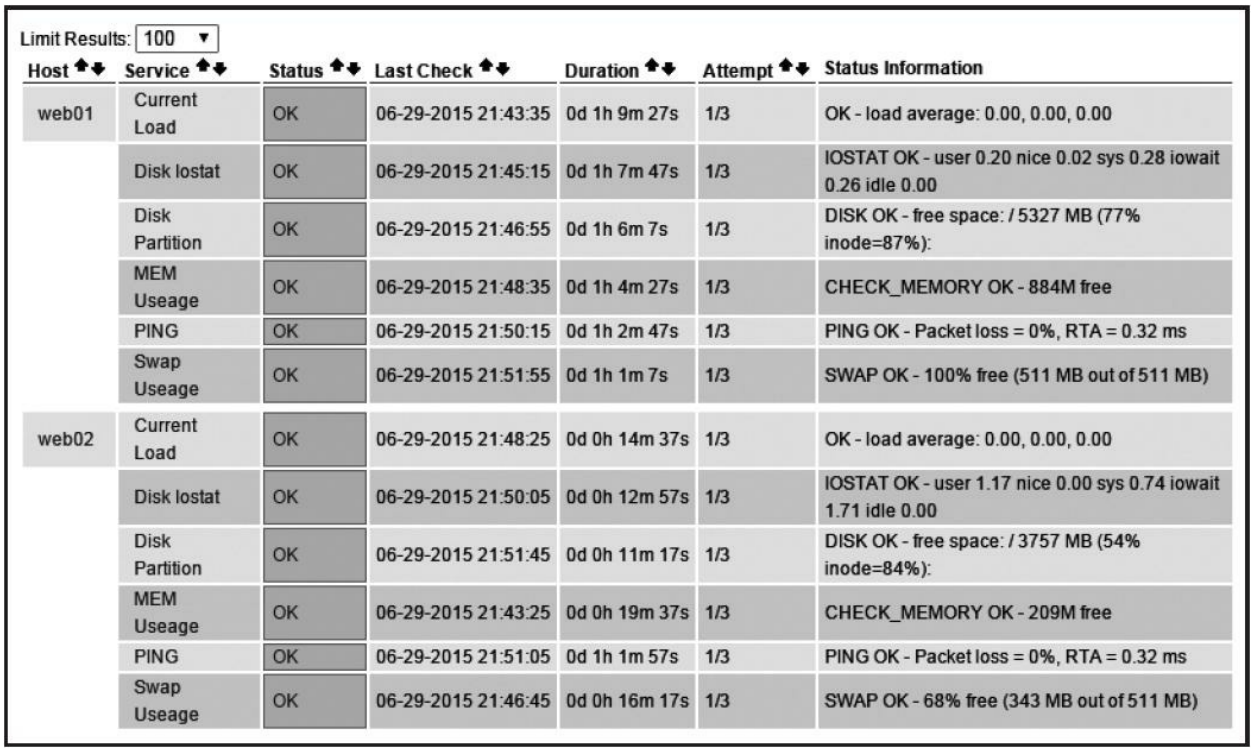
Command'check_disk'not defined”，一共三个类似的错误，提示NRPE命令check_disk等未定义，解决办法也在下文中。



Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
web01	Current Load	OK	06-29-2015 20:43:35	0d 0h 9m 14s	1/3	OK - load average: 0.00, 0.00, 0.00
	Disk Iostat	OK	06-29-2015 20:45:15	0d 0h 7m 34s	1/3	IOSTAT OK - user 0.23 nice 0.02 sys 0.30 iowait 0.30 idle 0.00
	Disk Partition	OK	06-29-2015 20:46:55	0d 0h 5m 54s	1/3	DISK OK - free space: / 5327 MB (77% inode=87%):
	MEM Useage	OK	06-29-2015 20:48:35	0d 0h 4m 14s	1/3	CHECK_MEMORY OK - 884M free
	PING	OK	06-29-2015 20:50:15	0d 0h 2m 34s	1/3	PING OK - Packet loss = 0%, RTA = 0.67 ms
	Swap Useage	OK	06-29-2015 20:51:55	0d 0h 0m 54s	1/3	SWAP OK - 100% free (511 MB out of 511 MB)
web02	Current Load	CRITICAL	06-29-2015 20:48:25	0d 0h 8m 24s	3/3	Connection refused or timed out
	Disk Iostat	CRITICAL	06-29-2015 20:50:05	0d 0h 6m 44s	3/3	Connection refused or timed out
	Disk Partition	CRITICAL	06-29-2015 20:51:45	0d 0h 5m 4s	3/3	Connection refused or timed out
	MEM Useage	CRITICAL	06-29-2015 20:51:25	0d 0h 3m 24s	2/3	Connection refused or timed out
	PING	OK	06-29-2015 20:51:05	0d 0h 1m 44s	1/3	PING OK - Packet loss = 0%, RTA = 0.30 ms
	Swap Useage	PENDING	N/A	0d 0h 10m 54s+	1/3	Service check scheduled for Mon Jun 29 20:52:45 CST 2015

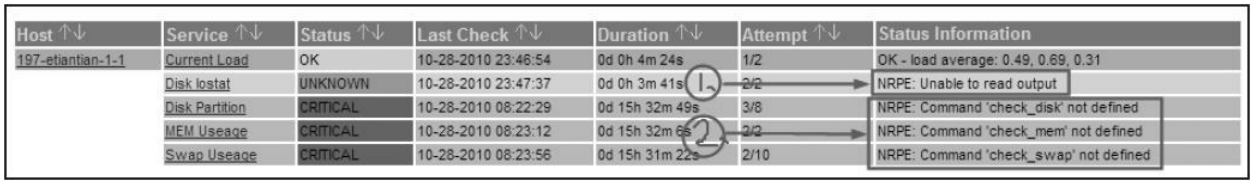
图14-15　新增主机重启后监控显示图



Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
web01	Current Load	OK	06-29-2015 21:43:35	0d 1h 9m 27s	1/3	OK - load average: 0.00, 0.00, 0.00
	Disk Iostat	OK	06-29-2015 21:45:15	0d 1h 7m 47s	1/3	IOSTAT OK - user 0.20 nice 0.02 sys 0.28 iowait 0.26 idle 0.00
	Disk Partition	OK	06-29-2015 21:46:55	0d 1h 6m 7s	1/3	DISK OK - free space: / 5327 MB (77% inode=87%):
	MEM Useage	OK	06-29-2015 21:48:35	0d 1h 4m 27s	1/3	CHECK_MEMORY OK - 884M free
	PING	OK	06-29-2015 21:50:15	0d 1h 2m 47s	1/3	PING OK - Packet loss = 0%, RTA = 0.32 ms
	Swap Useage	OK	06-29-2015 21:51:55	0d 1h 1m 7s	1/3	SWAP OK - 100% free (511 MB out of 511 MB)
web02	Current Load	OK	06-29-2015 21:48:25	0d 0h 14m 37s	1/3	OK - load average: 0.00, 0.00, 0.00
	Disk Iostat	OK	06-29-2015 21:50:05	0d 0h 12m 57s	1/3	IOSTAT OK - user 1.17 nice 0.00 sys 0.74 iowait 1.71 idle 0.00
	Disk Partition	OK	06-29-2015 21:51:45	0d 0h 11m 17s	1/3	DISK OK - free space: / 3757 MB (54% inode=84%):
	MEM Useage	OK	06-29-2015 21:43:25	0d 0h 19m 37s	1/3	CHECK_MEMORY OK - 209M free
	PING	OK	06-29-2015 21:51:05	0d 1h 1m 57s	1/3	PING OK - Packet loss = 0%, RTA = 0.32 ms
	Swap Useage	OK	06-29-2015 21:46:45	0d 0h 16m 17s	1/3	SWAP OK - 68% free (343 MB out of 511 MB)

图14-16　新增主机重启后正常后监控显示图



Host	Service	Status	Last Check	Duration	Attempt	Status Information
197-etiantian-1-1	Current Load	OK	10-28-2010 23:46:54	0d 0h 4m 24s	1/2	OK - load average: 0.49, 0.69, 0.31
	Disk Iostat	UNKNOWN	10-28-2010 23:47:37	0d 0h 3m 41s	2/2	NRPE: Unable to read output
	Disk Partition	CRITICAL	10-28-2010 08:22:29	0d 15h 32m 49s	3/8	NRPE: Command 'check_disk' not defined
	MEM Useage	CRITICAL	10-28-2010 08:23:12	0d 15h 32m 6s	2/2	NRPE: Command 'check_mem' not defined
	Swap Useage	CRITICAL	10-28-2010 08:23:56	0d 15h 31m 22s	2/10	NRPE: Command 'check_swap' not defined

图14-17　常见监控报错显示图

4.被动模式下基于Nagios监控原理排错的案例

还记得前文的nrpe原理图么？来回顾一下，见图14-18。

根据Nagios被动模式的监控原理进行错误排查的方法为：查看check_nrpe插件帮助。

Nagios被动模式的监控原理离不开check_nrpe这个插件，下面我们就看看check_nrpe插件帮助。

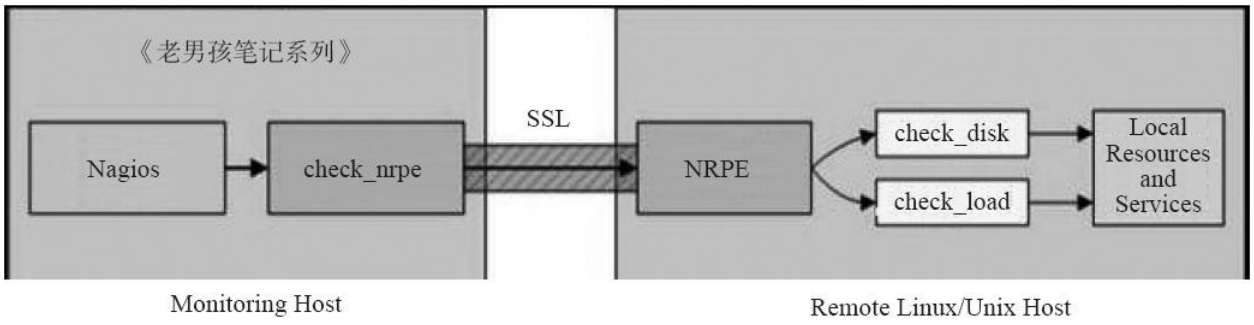


图14-18　回顾nrpe原理图

```
[root@nagios-server libexec]# ./check_nrpe --help
NRPE Plugin for Nagios.....


Usage:
```

```
 check_nrpe -H <host> [-n] [-u] [-p <port>] [-t <timeout>] [-c <command>] [-a <argli
Options:



-n          = Do no use SSL
-u          = Make socket timeouts return an UNKNOWN state instead of CRITICAL
<host>     = The address of the host running the NRPE daemon
#==>客户端的


IP地址



[port]     = The port on which the daemon is running  (


default=5666)



[timeout]     = Number of seconds before connection times out  (


default=10)



[command]     = The name of the command that the remote daemon should run
#==>客户端


nrpe配置文件里的命令名



[arglist]     = Optional arguments that should be passed to the command.  Multiple
            arguments should be separated by a space.  If provided,


 this must be
            the last option supplied on the command line.....
```

Nagios被动模式的监控原理其实就是利用下面这个命令工作的：

```
/usr/local/nagios/libexec/check_nrpe -H 10.0.0.8 -c check_mem
```

下面来看两个模拟Nagios配置错误的案例。

第一个案例模拟：取消check_memory.pl脚本的执行权限，模拟前面的错误。

```
[root@web01 ~]# cd /usr/local/nagios/libexec/
[root@web01 libexec]# chmod a-x check_memory.pl
[root@web01 libexec]# ls -l check_memory.pl
```

重启Nagios后，此时页面内存服务监控行出现出错提示。

```
NRPE:


 Unable to read output
```

第二个案例模拟：把nrpe.cfg中的命令名写错，如下：

```
command[check_disk1]=/usr/local/nagios/libexec/check_disk -w 15% -c 7% -p /
#==>check_disk1多个数字


1。
```

重启Nagios后，此时页面磁盘服务监控行出现如下出错提示。

```
NRPE:

 Command 'check_disk' not defined
```

下面就针对上面两个案例进行故障排查。

案例1：NRPE：Unable to read output排查

1）在Nagios服务器端执行如下命令：

```
[root@nagios-server libexec]# /usr/local/nagios/libexec/check_nrpe -H 10.0.0.8 -c ch
NRPE:

 Unable to read output
#==可以看到这里和页面报错是一样的
```

2）在客户端本地执行命令脚本检查（就是command[check_mem]=后面对应的脚本）。

```
[root@web01 libexec]# /usr/local/nagios/libexec/check_memory.pl -w 10% -c 3%
#==>这个命令最好是复制

nrpe.cfg里的配置，不要手敲，防止出错

-bash:

 /usr/local/nagios/libexec/check_memory.pl:
```

```
 Permission denied
#==>提示拒绝，所以，原因就找到了。因为


check_memory.pl脚本无执行权限导致的


NRPE:


 Unable to read output错误



[root@web01 libexec]# chmod a+x /usr/local/nagios/libexec/check_memory.pl
[root@web01 libexec]# /usr/local/nagios/libexec/check_memory.pl -w 10% -c 3%
CHECK_MEMORY OK - 95M free | free=100634624b;


12019302.4: ;


3605790.72:
```

3）此时在Nagios服务器端执行如下命令：

```
[root@nagios-server libexec]# /usr/local/nagios/libexec/check_nrpe -H 10.0.0.8 -c ch
CHECK_MEMORY OK - 95M free | free=99807232b;


12019302.4: ;


3605790.72:


#==>提示


OK了，那么页面应该也是正常的了
```

特别说明：如果步骤2正常但是步骤3依然不正常，此时可以在客户端本地修改nrpc.cfg的allows_hosts，增加一个本机IP地址10.0.0.8，然后在本机执行：

```
/usr/local/nagios/libexec/check_nrpe-H 10.0.0.8-c check_mem
```

来检查，从而排除是不是nrpe进程及nrpe配置文件的问题。

如果上面的可以，但从服务器端依然不行，一般就是网络原因或者软件版本等问题了，这类问题发生的几率很小。

案例2：NRPE：Command'check_disk'not defined排查

该案例涉及的是磁盘监控故障，排查思路如下：

在Nagios服务器端执行如下命令：

```
[root@nagios-server libexec]# /usr/local/nagios/libexec/check_nrpe -H 10.0.0.8 -c ch
NRPE:


 Command 'check_disk' not defined
```

如果大家理解了Nagios被动监控的原理，很容易判断上面问题的故障在于nrpe.cfg中check_disk相关命令写错了，或者没配。

根据Nagios被动模式监控原理排查问题的思路如下。

1）在服务器端执行如下命令看是否返回数据。

```
/usr/local/nagios/libexec/check_nrpe -H 10.0.0.8 -c check_mem
#<==10.0.0.8为客户端


IP地址。
```

2）如果前面无法正确返回数据，可在客户端10.0.0.8本地执行如下命令看是否返回数据。

```
/usr/local/nagios/libexec/check_nrpe -H 127.0.0.1 -c check_mem
```

3）如果前面还是无法正确返回数据，则执行nrpe.cfg里下面配置的内容中等号后面的命令。

```
command[check_mem]=/usr/local/nagios/libexec/check_memory.pl -w 10% -c 3%
```

即执行

```
/usr/local/nagios/libexec/check_memory.pl -w 10% -c 3%
```

如果以上3步都OK，那么就找服务器端原因，一般是配置文件以及命令插件问题。

5.添加http服务的URL地址及端口监控

现在增加从Nagios服务器端发起的监控，如URL地址、端口监控等。此类服务一般都开启了对外提供服务的业务。这样的业务，一般采用主动监控的方式，当然了，也可以写脚本通过被动的方式来监控，但一般不这么做。

URL监控的实质是通过命令行理解HTTP的监控原理，如下：

```
[root@nagios-server service]# /usr/local/nagios/libexec/check_http -H 10.0.0.8
HTTP OK HTTP/1.1 200 OK - 266 bytes in 0.006 seconds |time=0.006011s; ; ;


0.000000 size=266B; ; ;


0
[root@nagios-server service]# /usr/local/nagios/libexec/check_http -H 10.0.0.8 -u /
index.html
HTTP OK HTTP/1.1 200 OK - 266 bytes in 0.001 seconds |time=0.001147s; ; ;


0.000000 size=266B; ; ;


0
```

下面是对域名URL地址http://blog.etiantian.org 进行监控的配置，将要监控的服务配置到services.cfg中即可。

```
#url examples http:


// blog.etiantian.org
define service{
        use                 generic-service
```

```
        host_name               web01
        service_description     blog_url
        check_command          check_weburl!


-H blog.etiantian.org
        }
```

当然也可以不用自己定义check_weburl，而使用配置自带的check_http。

```
define service {
        use               generic-service
        host_name               web01
        service_description     http_url
        check_command          check_http
}
```

若是要对复杂URL地址进行监控（例如：http://blog.etiantian.org/oldboy/test.html ），可采用如下方式：

```
define service{
        use               generic-service
        host_name               web01
        service_description     blog_oldboy_url
        check_command          check_weburl!


-H blog.etiantian.org -u /oldboy/test.html
#<==check_weburl是检查命令，在


command.cfg里定义的，后面就是真正检查的


URL地址了，即


http:


// blog.etiantian.org/oldboy/test.html，在该命令中，
```

```
-u后加域名后面的地址。



#<==[root@nagios-server objects]# /usr/local/nagios/libexec/check_http --help
#<==-u,



 --url=PATH
#<==   URL to GET or POST  (


default:


 /)



        }
```

若是集群中有多个URL，可采用如下方式监控。

```
define service {
        use               generic-service
        host_name               web01
        service_description     http_url_oldboy
        check_command          check_http!


 -u /oldboy/test.html
}
```

对特殊的带传参的URL地址进行监控时，方式如下：

```
例如:


http:
```

```
// blog.etiantian.org/article/index.phpm=article&a=list&id=670
define service{
        use                 generic-service
        host_name                 web01
        service_description     blog_oldboy_url
        check_command          check_weburl!


-H blog.etiantian.org -u "/article/index.phpm=article&a=list&id=670"
}
```

6.配置好URL后检查Nagios语法

若在配置好URL后，就执行如下命令检查Nagios语法：

```
[root@nagios-server objects]#/etc/init.d/nagios checkconfig
```

会发现报错了：

```
Checking services...
Error:


 Service check command 'check_weburl' specified in service 'blog_oldboy_url' for hos


Error:


 Service check command 'check_weburl' specified in service 'blog_url' for host '08-w


...省略若干...


Total Warnings:


 0
```

```
Total Errors:

   2
```

根据报错信息可以知道，是因为在command.cfg配置文件中没有定义check_weburl插件导致的。因此，需要在commands.cfg中加入check_weburl的插件配置。

```
# 'check_weburl' command definition
define command{
        command_name     check_weburl
        command_line     $USER1$/check_http $ARG1$  -w 10 -c 30
        }
# 'check_http' command definition
define command{
        command_name     check_http
        command_line     $USER1$/check_http -I $HOSTADDRESS$ $ARG1$
        }
#==>实际就是利用


check_http插件，查看帮助的方法如下：



[root@web01 ~]# cd /usr/local/nagios/libexec/
[root@web01 libexec]# ./check_http -help
```

如果是测试域名URL监控，注意在Nagios服务器端的/etc/hosts下加如下域名解析：

```
10.0.0.8 blog.etiantian.org
#10.0.0.8 为要监控的


URL所在的服务器


IP地址
```

```
#如果你在客户端没有配也可以不加，在笔者的公网博客已经给出了测试地址。
```

 特别说明：为了防止生产服务器频繁误报，在生产环境有时也会将域名解析成对应DNS A记录的IP并放在/etc/hosts里保存，这样可以防止因监控服务器本身的DNS问题导致误报。但这也有一个小问题，就是不能监控到DNS导致的域名解析故障了。

此外，利用别名可实现对集群下面同样节点的URL监控。

```
web1     blog.etiantian.org,


blog1.etiantian.org
web2     blog.etiantian.org,


blog2.etiantian.org
```

此时执行检查语法的命令为：

```
[root@nagios-server objects]#/etc/init.d/nagios checkconfig
Total Warnings:


     0
Total Errors:


     0
```

使配置文件生效的命令为：

```
/etc/init.d/nagios reload
```

到这里，可以看到自己配置的URL监控成果了，如图14-19所示。

Host	Service	Status	Last Check	Duration	Attempt	Status Information
180-lnmp-1	Current Load	OK	06-08-2012 07:23:37	0d 8h 33m 31s	1/1	OK - load average: 0.00, 0.00, 0.00
	Disk Iostat	OK	06-08-2012 07:23:37	0d 8h 33m 4s	1/2	IOSTAT OK - user 0.10 nice 0.00 sys 1.07 iowait 0.05 idle 0.00
	Disk Partition	OK	06-08-2012 07:23:37	0d 8h 33m 4s	1/8	DISK OK - free space: / 3927 MB (57% inode=93%):
	MEM Useage	OK	06-08-2012 07:23:37	0d 8h 33m 4s	1/2	CHECK_MEMORY OK - 89M free
	Swap Useage	OK	06-08-2012 07:23:37	0d 8h 33m 4s	1/10	SWAP OK - 100% free (509 MB out of 509 MB)
	blog_url	OK	06-08-2012 07:23:37	0d 0h 16m 18s	1/3	HTTP OK HTTP/1.1 200 OK - 263 bytes in 0.132 seconds
	blog_url_1	OK	06-08-2012 07:23:37	0d 0h 5m 3s	1/3	HTTP OK HTTP/1.1 200 OK - 261 bytes in 0.103 seconds
182-lamp-1	Current Load	OK	06-08-2012 07:23:37	0d 1h 47m 7s	1/1	OK - load average: 0.00, 0.01, 0.00
	Disk Iostat	OK	06-08-2012 07:23:37	0d 1h 45m 55s	1/2	IOSTAT OK - user 0.04 nice 0.00 sys 0.60 iowait 0.14 idle 0.00
	Disk Partition	OK	06-08-2012 07:23:37	0d 1h 47m 6s	1/8	DISK OK - free space: / 3896 MB (56% inode=93%):
	MEM Useage	OK	06-08-2012 07:23:37	0d 1h 45m 55s	1/2	CHECK_MEMORY OK - 89M free
	Swap Useage	OK	06-08-2012 07:23:37	0d 1h 47m 6s	1/10	SWAP OK - 100% free (509 MB out of 509 MB)

图14-19　新增监控URL服务正常后结果显示图

说明：未生效前为灰色PENDING状态，需要等待。

7.监控任意TCP端口举例

端口监控的实质就是执行如下命令去监控：

```
[root@nagios-server service]# /usr/local/nagios/libexec/check_tcp -H 10.0.0.8 -p 80
TCP OK - 0.000 second response time on port 80|time=0.000461s; ; ;


0.000000;


10.000000
[root@nagios-server objects]#vi services.cfg
define service{
        use                             generic-service
        host_name                       web01
        service_description                 ssh_22
```

```
        check_command                   check_tcp!


22
        }
define service{
        use                             generic-service
        host_name                       web01
        service_description                 http_80
        check_command                       check_tcp!


80
        }
```

这里的check_tcp为Nagios plugin默认插件，commands.cfg会自动配置进去，不需要添加。此外，注意多端口同时监控的写法。

说明：从多年的监控经验看，端口检查也是很不错的辅助监控方式！对于要求高的业务，一定要模拟真正用户的访问行为监控才好。

监控结果如图14-20所示。

Host	Service	Status	Last Check	Duration	Attempt	Status Information
180-lnmp-1	Current Load	OK	06-08-2012 07:45:35	0d 8h 56m 1s	1/1	OK - load average: 0.00, 0.00, 0.00
	Swap Useage	OK	06-08-2012 07:43:36	0d 8h 55m 34s	1/10	SWAP OK - 100% free (509 MB out of 509 MB)
	blog_url	OK	06-08-2012 07:45:36	0d 0h 38m 48s	1/3	HTTP OK HTTP/1.1 200 OK - 263 bytes in 0.007 seconds
	blog_url_1	OK	06-08-2012 07:45:36	0d 0h 27m 33s	1/3	HTTP OK HTTP/1.1 200 OK - 261 bytes in 0.025 seconds
	http_80	OK	06-08-2012 07:45:36	0d 0h 18m 47s	1/3	TCP OK - 0.010 second response time on port 80
	http_80_ssh_22	OK	06-08-2012 07:43:36	0d 0h 18m 25s	1/5	TCP OK - 0.007 second response time on port 80
	ssh_22	OK	06-08-2012 07:45:36	0d 0h 18m 3s	1/3	TCP OK - 0.029 second response time on port 22
182-lamp-1	Current Load	OK	06-08-2012 07:45:35	0d 2h 9m 37s	1/1	OK - load average: 0.00, 0.00, 0.00
	Disk Iostat	OK	06-08-2012 07:43:36	0d 2h 8m 25s	1/2	IOSTAT OK - user 0.05 nice 0.00 sys 0.60 iowait 0.14 idle 0.00
	Disk Partition	OK	06-08-2012 07:43:36	0d 2h 9m 36s	1/8	DISK OK - free space: / 3896 MB (56% inode=93%):
	MEM Useage	OK	06-08-2012 07:43:35	0d 2h 8m 25s	1/2	CHECK_MEMORY OK - 93M free
	Swap Useage	OK	06-08-2012 07:43:36	0d 2h 9m 36s	1/10	SWAP OK - 100% free (509 MB out of 509 MB)

图14-20　监控端口效果图

如果想对端口做精细控制，则须在commands.cfg里做配置。下面以

对Memcached服务进行监控为例，首先在commands.cfg里添加如下监控Memcached的命令：

```
define command {
command_name check_memcached_11111
command_line $USER1$/check_tcp -H $HOSTADDRESS$ -p 11111 -t 5 -E -s 'stats\r\nquit\r
        }
```

然后在services.cfg里添加如下命令：

```
######memcached check start ####
# memcached验证码缓存


create by oldboy 20080201
define service {
        use                     generic-service
        host_name                       web01
        service_description             Memcache_11111
        check_command                 check_memcached_11111
}
```

别忘了检查语法，reload看结果（如图14-21所示）。

Host	Service	Status	Last Check	Duration	Attempt	Status Information
031-etiantian-1-1	Current Load	OK	07-03-2010 07:32:56	0d 0h 26m 5s	1/2	OK - load average: 0.00, 0.04, 0.04
	Disk Iostat	OK	07-03-2010 07:33:35	0d 0h 25m 26s	1/2	IOSTAT OK - user 1.87 nice 0.00 sys 0.86 iowait 0.40 idle 0
	Disk Partition	OK	07-03-2010 07:33:35	0d 0h 28m 46s	1/8	DISK OK - free space: / 33425 MB (90% inode=98%):
	MEM Useage	OK	07-03-2010 07:30:58	0d 0h 28m 3s	1/2	CHECK_MEMORY OK - 208M free
	Memcache_11111	OK	07-03-2010 07:34:39	0d 0h 0m 22s	1/3	TCP OK - 0.001 second response time on port 11111 [STAT
	Swap Useage	OK	07-03-2010 07:34:42	0d 0h 27m 19s	1/10	SWAP OK - 100% free (799 MB out of 799 MB)
	blog_ryan_url	OK	07-03-2010 07:34:32	0d 0h 16m 29s	1/3	HTTP OK HTTP/1.1 200 OK - 189 bytes in 0.002 seconds
	blog_url	OK	07-03-2010 07:32:59	0d 0h 16m 2s	1/3	HTTP OK HTTP/1.1 200 OK - 189 bytes in 0.001 seconds
	http_80	OK	07-03-2010 07:34:05	0d 0h 12m 56s	1/3	TCP OK - 0.001 second response time on port 80
	http_80_ssh_22	OK	07-03-2010 07:34:22	0d 0h 12m 39s	1/5	TCP OK - 0.001 second response time on port 80
	ssh_22	OK	07-03-2010 07:34:40	0d 0h 12m 21s	1/3	TCP OK - 0.000 second response time on port 22

图14-21　配置监控Memcached效果图

Memcached服务监控很管用，在生产环境下，即使是Memcached连接数不够用也都能报警。当然Memcached监控还有更精细的监控方法，

下面会给出一定的样例，有问题可加开篇的QQ群交流。

样例如下（需要单独安装Memcached插件才可以按如下配置）：

```
### check response time (


msec)


 for memcached
define command {
       command_name     check_memcached_response
       command_line     $USER1$/check_memcached -H $HOSTADDRESS$ -w 300 -c 500
       }
### check cache size ratio (


bytes/limit_maxbytes[%])


 for memcached
define command {
       command_name     check_memcached_size
       command_line     $USER1$/check_memcached -H $HOSTADDRESS$ --size-warning 90 -
       }
### check cache hit ratio (


get_hits/cmd_get[%])


 for memcached
define command {
       command_name     check_memcached_hit
       command_line     $USER1$/check_memcached -H $HOSTADDRESS$ --hit-warning 40 --
       }
```

第13章中监控Memcached服务的脚本，大家还记得么？可酌情使用。

实例暂时就讲这些，关于如何让不同服务在不同时间报警给不同用

户，如何配置nfs、rsync、drbd、MySQL、Oracle等特殊服务的监控，如何分组监控，如何让多个运维值班协调解决问题等内容，见后文的手工开发插件部分。Nagios很复杂，不仅可以做上面的事，还可以写插件监控业务层的问题，甚至可以监控到服务器的温度及硬件信息。而且，Nagios给你扩展开发的机会很多，请大家多去摸索。这里针对上面的内容小结一下：

一般客户端对外开启的服务，都会采用主动模式监控，例如：port、URL。

主动模式的监控配置过程如下：

1）在服务器端的命令行把要监控的命令先调试好。

2）在commands.cfg里定义Nagios命令，同时调用命令行的插件。

3）在服务的配置文件里定义要监控的服务，调用commands.cfg里定义Nagios的监控命令！

课后作业：

·请用主动及被动模式分别监控MySQL主从同步（check_mysql）！

·根据不同管理员显示不同的主机和服务，根据用户分类显示主机和服务。

14.5.3 Nagios的调试

1.检查Nagios语法并优化配置Nagios启动脚本

字符串检查的语法如下：

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

优化配置Nagios启动脚本中检查语法的部分。

```
[root@nagios-server ~]#vi /etc/init.d/nagios +178
    176          checkconfig)



    177              printf "Running configuration check..."
    178              #$NagiosBin -v $NagiosCfgFile > /dev/null 2>&1;



    179              $NagiosBin -v $NagiosCfgFile;
```

这里要注释掉第178行的内容，添加第179行的内容，然后保存，其实也就是去掉了第178行的“>/dev/null 2>&1”，让错误在屏幕上打印出来。测试如下：

```
[root@nagios-server ~]# /etc/init.d/nagios checkconfig
Running configuration check.....省略大部分...
```

```
...这里是详细的检查结果...


Total Warnings:

 0
Total Errors:

  1.....


 CONFIG ERROR!

 Check your Nagios configuration.
```

2.检查语法及加载配置

配置完服务后，就是该检查结果的时候了。

检查语法的命令如下：

```
[root@nagios-server objects]#/etc/init.d/nagios checkconfig
```

如果语法错误，需要根据错误信息调试好，然后继续执行检查语法。

使配置文件生效的命令为/etc/init.d/nagios reload，一般不需要用restart。

```
[root@nagios-server ~]# /etc/init.d/nagios reload
Running configuration check...done.
```

```
Reloading nagios configuration...done
```

3.通过日志排查问题

在配置监控出问题后，也可以看看Nagios的日志，有可能会发现一些故障所在。

```
[root@web01 ~]# tail /usr/local/nagios/var/nagios.log
[1337334508] SERVICE ALERT:


 179-oldboy;


blog_url;


CRITICAL;


HARD;


1;


Name or service not known
[1337335957] Auto-save of retention data completed successfully.
[1337336087] SERVICE ALERT:


 179-oldboy;


Current Load;


OK;


HARD;
```

1;

OK - load average:

 1.79,

 1.03,

 0.39
[1337336087] SERVICE ALERT:

 179-oldboy;

179-check_22;

OK;

HARD;

1;

10.0.0.8 22 is ok.
[1337336087] HOST ALERT:

 179-oldboy;

UP;

HARD;

1;

PING OK - Packet loss = 0%,

 RTA = 2.77 ms
[1337336207] SERVICE ALERT:

 179-oldboy;

blog_url;

CRITICAL;

SOFT;

1;

Name or service not known
[1337336267] SERVICE ALERT:

 179-oldboy;

blog_url;

CRITICAL;

SOFT;

2;

Name or service not known
[1337336327] SERVICE NOTIFICATION:

nagiosadmin;

179-oldboy;

blog_url;

CRITICAL;

notify-service-by-email;

Name or service not known
[1337337846] Caught SIGTERM,

shutting down...
[1337337846] Successfully shutdown... (

PID=1779)

[root@web01 ~]# tail /var/log/messages
Jun 7 07:

44:

59 lvs2 nrpe[25333]:

Listening for connections on port 5666
Jun 7 07:

44:

59 lvs2 nrpe[25333]:

```
 Allowing connections from:


 10.0.0.8
Jun     7 08:


01:


05 lvs2 nrpe[25333]:


 Caught SIGTERM - shutting down...
Jun     7 08:


01:


05 lvs2 nrpe[25333]:


 Cannot remove pidfile '/var/run/nrpe.pid' - check your privileges.
Jun     7 08:


01:


05 lvs2 nrpe[25333]:


 Daemon shutdown
Jun     7 08:


01:


09 lvs2 nrpe[25438]:


 Starting up daemon
Jun     7 08:
```

01:

09 lvs2 nrpe[25438]:

 Listening for connections on port 5666
Jun   7 08:

01:

09 lvs2 nrpe[25438]:

 Allowing connections from:

 10.0.0.8
Jun   7 08:

01:

34 lvs2 nrpe[25440]:

 Error:

 Could not complete SSL handshake. 5
Jun   7 08:

19:

07 lvs2 nrpe[25558]:

 Error:

 Could not complete SSL handshake. 5

上述日志中的Error部分就是监控报警的故障。

14.6　服务器端Nagios图形监控显示和管理

前面搭建的Nagios服务虽然能显示信息，能报警。但是在企业工作中还会需要一个历史趋势图，跟踪每一个业务的长期趋势，并且能以图形的方式展示，例如：根据磁盘的剩余趋势，确定是否需要提前购买磁盘。

14.6.1 服务器端安装PNP生成图形监控曲线

PNP是一款配合Nagios出图的软件，官方站点为：http://www.pnp4nagios.org 。

1.PNP出图基础依赖软件安装

先通过下面的命令安装PNP软件需要的基础包，如果有部分是重复安装，再执行一下也不会有问题。

```
[root@nagios-server ~]# yum install cairo pango zlib zlib-devel freetype freetype-de
[root@nagios-server ~]# rpm -qa cairo pango zlib zlib-devel freetype freetype-devel
zlib-devel-1.2.3-29.el6.x86_64
freetype-devel-2.3.11-15.el6_6.1.x86_64
gd-2.0.35-11.el6.x86_64
gd-devel-2.0.35-11.el6.x86_64
freetype-2.3.11-15.el6_6.1.x86_64
zlib-1.2.3-29.el6.x86_64
pango-1.28.1-10.el6.x86_64
cairo-1.8.8-6.el6_6.x86_64
```

然后安装rrdtool依赖的libart_lgpl相关软件包，这个软件包要优先于rrdtool安装。

```
[root@nagios-server ~]# yum install libart_lgpl libart_lgpl-devel -y
[root@nagios-server ~]# rpm -qa libart_lgpl libart_lgpl-devel
libart_lgpl-2.3.20-5.1.el6.x86_64
libart_lgpl-devel-2.3.20-5.1.el6.x86_64
```

PNP工具最终是通过rrdtool实现的画图，因此需要提前安装rrdtool。

```
[root@nagios-server ~]# yum install rrdtool rrdtool-devel -y
[root@nagios-server ~]# rpm -qa rrdtool rrdtool-devel
rrdtool-1.3.8-7.el6.x86_64
rrdtool-devel-1.3.8-7.el6.x86_64
[root@nagios-server ~]# which rrdtool
/usr/bin/rrdtool
```

2.安装出图Web界面展示软件PNP

此处选择0.4.14的PNP版本，如果选择高版本在出图方面可能会有坑，正常情况下，选0.4版已经足够了，因此，如果没有特殊需求，建议最好完全按照本书测试步骤，在弄清楚之后再变通版本。

PNP软件无法yum安装，可通过编译的方式进行安装，编辑过程的命令集如下：

```
wget -O pnp-0.4.14.tar.gz http:


// downloads.sourceforge.net/project/pnp4nagios/PNP/pnp-0.4.14/pnp-0.4.14.tar.gzuse_
yum install perl-Time-HiRes -y
tar zxf pnp-0.4.14.tar.gz
cd pnp-0.4.14
./configure \
--with-rrdtool                                  #<==配置出图的


rrdtool
--with-perfdata-dir=/usr/local/nagios/share/perfdata/      #<==出图所用的数据路径



make all
make install
make install-config
make install-init
ll /usr/local/nagios/libexec/ |grep process
```

如果configure后出现如下警告信息，请忽略：

```
##################
#  WARNING:


 The RRDs Perl Modules are not found on your System
#           Using RRDs will speedup things in larger Installtions.
######################
```

安装Nagios PNP时可能会有报错，如下：

```
configure:


 error:


 Perl Module Time: :


HiRes not available
```

解决方案为：

```
yum install perl-Time-HiRes -y
```

PNP提供了一个获取数据出图的Perl脚本，可以用如下命令查到：

```
[root@nagios-server pnp-0.4.14]# ll /usr/local/nagios/libexec/ |grep process
-rwxr-xr-x 1 nagios nagios  31813 Jul  1 18:


50 process_perfdata.pl
```

此时打开浏览器访问"http://10.0.0.7/nagios/pnp/ "，应该会出现如图

14-22所示的图形界面，但是没有业务数据显示。

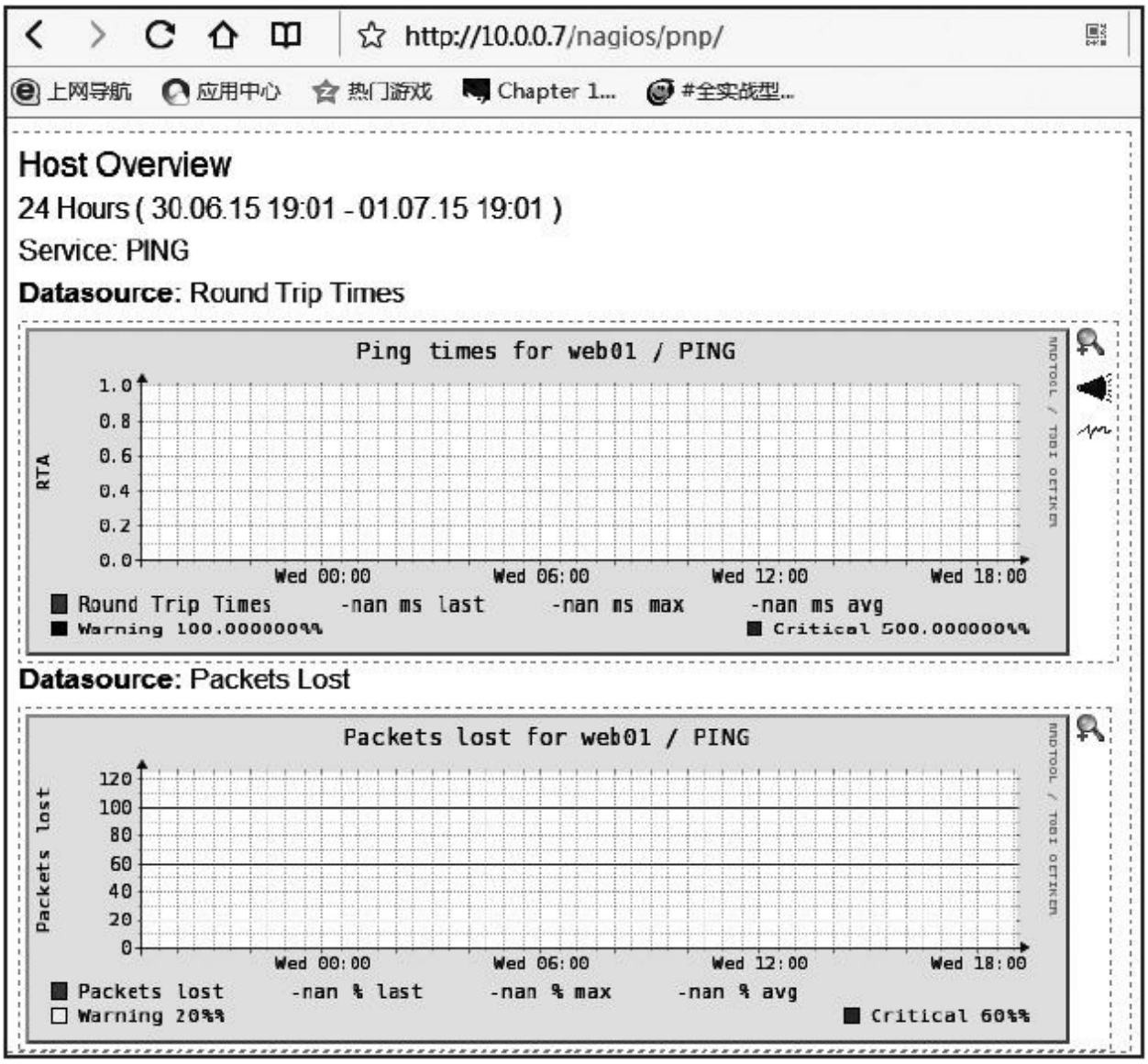


图14-22　刚部署完无数据时PNP图形

如果出现如图14-23所示的问题图形显示，先别着急，可能过一会重新访问上述地址就会恢复正常。

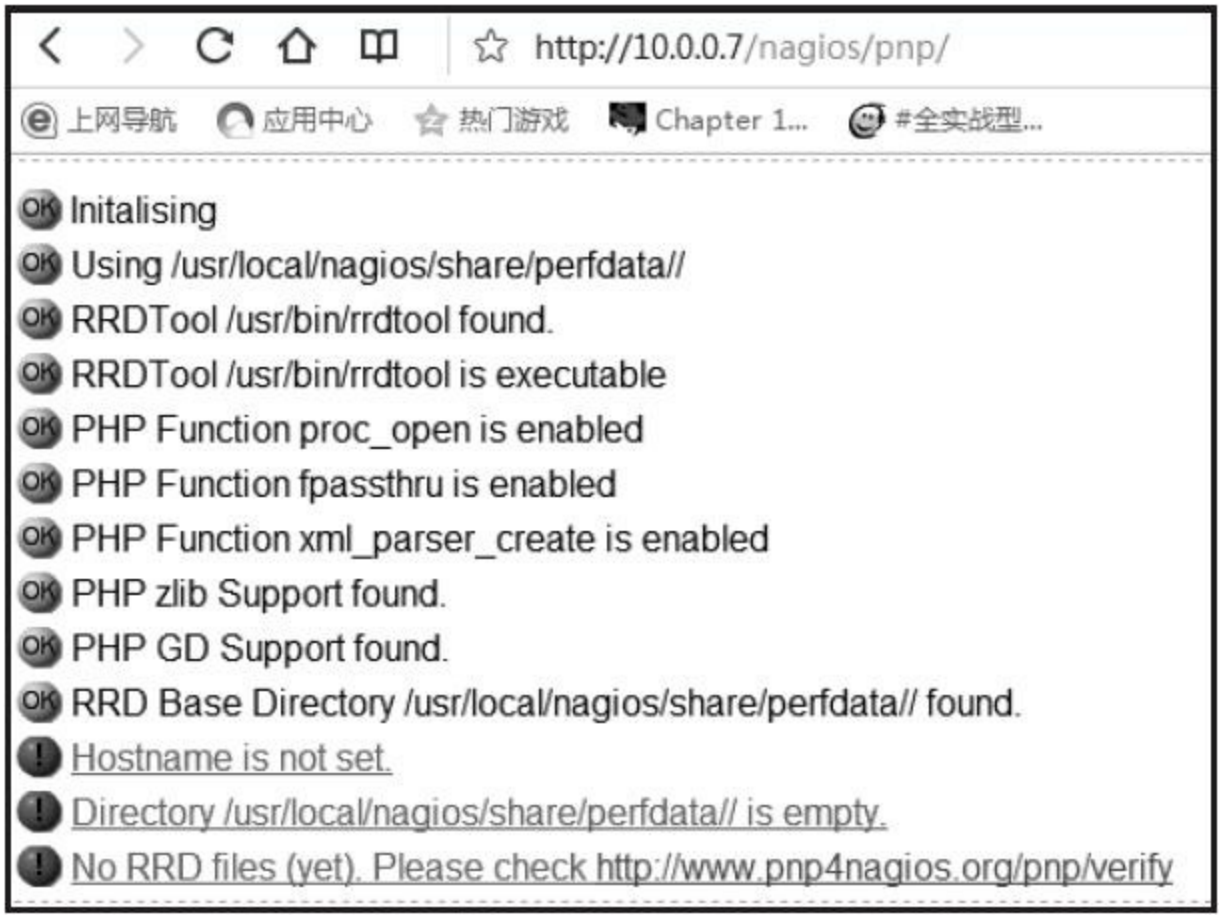


图14-23　刚部署完PNP时可能出现的问题图

如果过了很长时间重新访问上述地址还不正常，可以执行如下命令看看，或者继续后面的操作，然后在访问试试：

```
yum install php-gd gd gd-devel -y
```

提示：如果是按照本书步骤安装的Nagios，应该不会出现上面图14-23无法显示图形的问题了。上述问题已经在本文的开始部分提前处理掉了。

3.Nagios出图相关配置

1）执行编辑命令“vi/usr/local/nagios/etc/nagios.cfg+835”，修改nagios.cfg主配置文件第835行，将如下参数对应的值从0改为1，表示记录数据。

```
process_performance_data=1      #<==默认值为

0，改为

1。
```

然后继续向下从大概第848行开始，找到如下两项，取消参数开头的注释，修改后的最终结果如下：

```
host_perfdata_command=process-host-perfdata
service_perfdata_command=process-service-perfdata
```

2）执行编辑命令“vi/usr/local/nagios/etc/objects/commands.cfg+227”，修改commands.cfg Nagios命令配置文件，定义出图获取数据的命令。

要修改的是commands.cfg配置文件的第227~238行，默认的配置如下：

```
[root@nagios-server pnp-0.4.14]# sed -n '227,


238p' /usr/local/nagios/etc/objects/commands.cfg
# 'process-host-perfdata' command definition
define command{
        command_name    process-host-perfdata
        command_line          /usr/bin/printf "%b" "$LASTHOSTCHECK$\t$HOSTNAME$\t$HO
        }
# 'process-service-perfdata' command definition
define command{
        command_name    process-service-perfdata
        command_line          /usr/bin/printf "%b" "$LASTSERVICECHECK$\t$HOSTNAME$\t
        }
```

现在删除上述的默认配置，然后将其改为如下的配置内容：

```
# 'process-host-perfdata' command definition
define command{
        command_name    process-host-perfdata
        command_line    /usr/local/nagios/libexec/process_perfdata.pl
        }
# 'process-service-perfdata' command definition
define command{
        command_name    process-service-perfdata
        command_line    /usr/local/nagios/libexec/process_perfdata.pl
        }
```

也可以用Nagios变量$USER1$替代/usr/local/nagios/libexec/路径。特别要说明的是，上面的和下面的配置是等价的。

```
# 'process-host-perfdata' command definition
define command{
        command_name    process-host-perfdata
        command_line    $USER1$/process_perfdata.pl
        }
# 'process-service-perfdata' command definition
define command{
        command_name    process-service-perfdata
        command_line    $USER1$/process_perfdata.pl
        }
```

3）执行检查语法命令。

```
[root@nagios-server objects]# /etc/init.d/nagios reload.....


Total Warnings:


 0
Total Errors:


   0.....
```

从提示来看，警告和错误数都为0个，表示配置通过。

4）执行命令使Nagios配置文件生效。

```
[root@nagios-server objects]# /etc/init.d/nagios reload
Running configuration check...done.
Reloading nagios configuration...done
```

5）此时在浏览器输入“http://10.0.0.7/nagios/pnp/index.php ”打开页面，正确的PNP界面如图14-24所示。

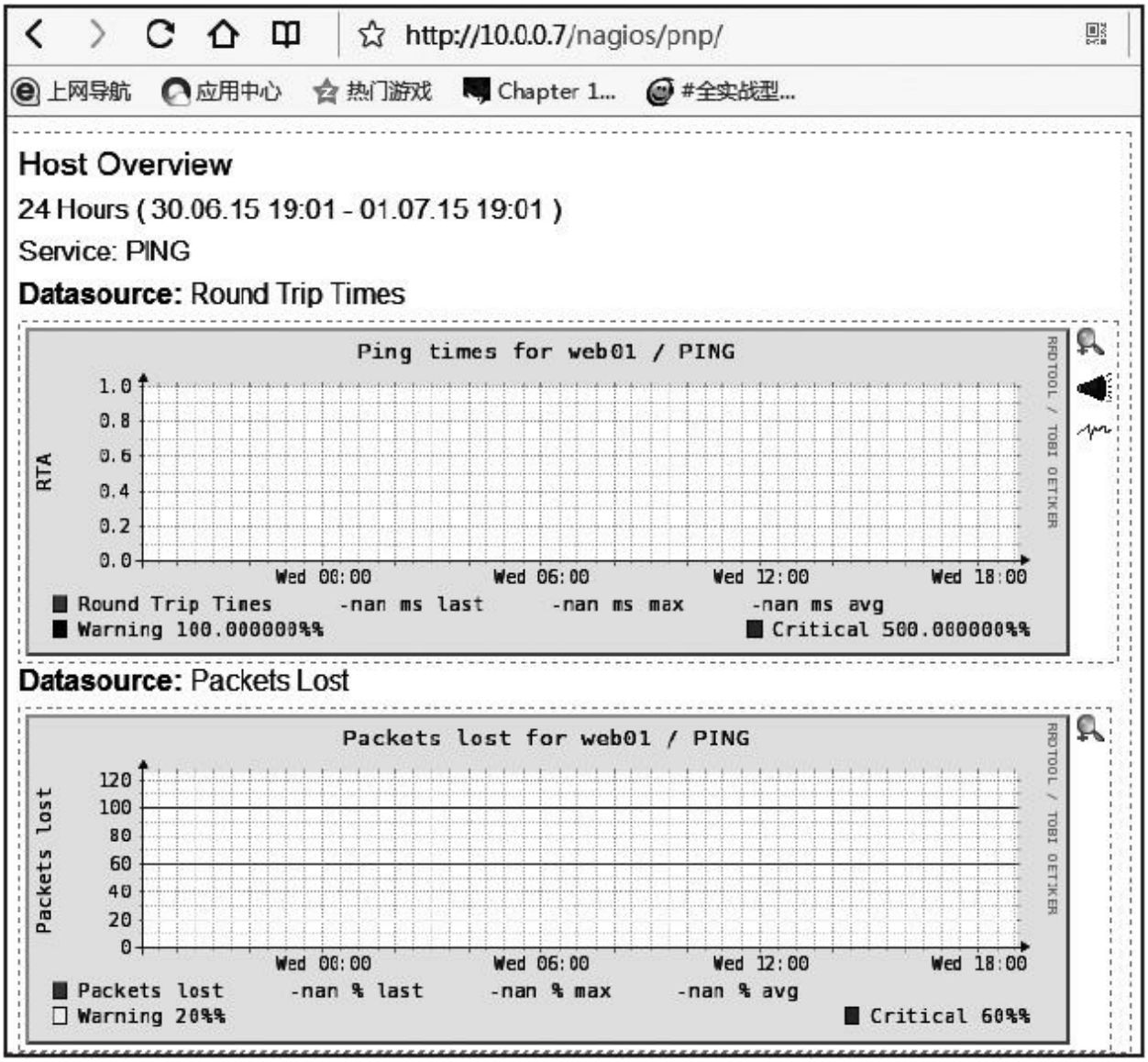


图14-24　正确的PNP界面

到这里为止，PNP软件的出图就OK了，但是还没有业务数据的图形趋势，因为还没有配置呢，接下去就来配置。

14.6.2　配置主机及服务获取状态数据出图

14.6.1节结尾的图形是没有具体的业务数据图形趋势的，因为那时还没有为Nagios的各个主机和具体要监控的服务配置获取数据信息，下面是让各个主机或服务获取数据的配置。

1.设置让被监控的主机记录数据

如果要让所有的主机获取数据并出趋势图，则须编辑Nagios的主机hosts.cfg文件，不过，只要在每一个被监控主机的配置下面增加同一个参数项“process_perf_data 1”即可。操作步骤如下：

```
[root@nagios-server ~]# cd /usr/local/nagios/etc/objects/
[root@nagios-server objects]# vi hosts.cfg
# Define a host for the local machine
define host{
        use                     linux-server
        host_name                       web01
        alias                   web01
        address                   10.0.0.8
        process_perf_data           1     #<==为


web01增加此行，表示将记录


web01
                                  主机的状态数据。



        }
define host{
        use                     linux-server
        host_name                       web02
        alias                   web02
        address                   10.0.0.9
```

```
        process_perf_data           1      #<==为

web02增加此行，表示将记录

web02
                                    主机的状态数据。

        }
```

上述代码中的“process_perf_data 1”就是获取对应主机及服务数据的参数配置。

2.设置让被监控主机对应的服务记录数据

如果需要所有的主机对应的服务获取数据并出趋势图，则要编辑Nagios的服务配置文件services.cfg，当然，也只需要在每一个对应服务下面增加同一个参数项即可，即“process_perf_data 1”，配置步骤如下：

```
[root@nagios-server objects]# vi services.cfg
define service {
        use                 generic-service
        host_name                 web01,


web02
        service_description      Disk Partition
        check_command           check_nrpe!


check_disk
        process_perf_data   1      #<==为


web01,
```

web02增加此行，表示将记录

web01,

web02主机的剩余磁盘空间状态数据，下同

```
}
define service {
        use                 generic-service
        host_name                web01,

web02
        service_description     Swap Useage
        check_command          check_nrpe!

check_swap
        process_perf_data   1
}
define service {
        use                 generic-service
        host_name                web01,

web02
        service_description     MEM Useage
        check_command          check_nrpe!

check_mem
        process_perf_data   1
}
define service {
        use                 generic-service
        host_name                web01,

web02
        service_description     Current Load
        check_command          check_nrpe!

check_load
        process_perf_data   1
}
```

```
define service {
        use                 generic-service
        host_name               web01,


web02
        service_description     Disk Iostat
        check_command          check_nrpe!


check_iostat!


5!


11
        process_perf_data   1
}
define service {
        use                 generic-service
        host_name               web01,


web02
        service_description     PING
        check_command          check_ping!


100.0,


20%!


500.0,


60%
        process_perf_data    1
}
```

由于每个主机对应的服务内容太多了，因此可以采取在所有服务对应的统一模板里添加配置参数的方式，这样可使所有的服务都可以生效。这里每个服务使用的模板就是由服务里的“use generic-service”这个

选项确定的，查看与模板文件里服务模板generic-service名对应的服务参数：

```
[root@nagios-server objects]# sed -n '154,


177p' templates.cfg
        name                    generic-service
        active_checks_enabled           1
        passive_checks_enabled           1
        parallelize_check          1
        obsess_over_service          1
        check_freshness               0
        notifications_enabled          1
        event_handler_enabled          1
        flap_detection_enabled           1
        failure_prediction_enabled     1
        process_perf_data          1
        retain_status_information     1
        retain_nonstatus_information     1
       is_volatile                0
        check_period                24x7
        max_check_attempts           3
        normal_check_interval           10
        retry_check_interval          2
        contact_groups               admins
        notification_options           w,


u,


c,


r
        notification_interval           60
        notification_period          24x7
         register                    0
        }
```

提示：为了看得清晰，这里去掉了所有的注释，服务的模板里默认已经配置了“process_perf_data 1”，即凡是使用templates.cfg模板文件里名字为generic-service的模板，均作为服务的模板，这样，就相当于

所有的服务都执行generic-service模板里的配置了。

配置完毕重启Nagios服务：

```
[root@nagios-server objects]# /etc/init.d/nagios reload
Running configuration check...done.
Reloading nagios configuration...done
```

到此，如果等一段时间，然后查看PNP URL就可以发现生成了图形数据，有些数据需要压测或者真实环境才能看的，例如主机负载，这里看的是磁盘剩余空间监控，其趋势图见图14-25下面方框里的标记。

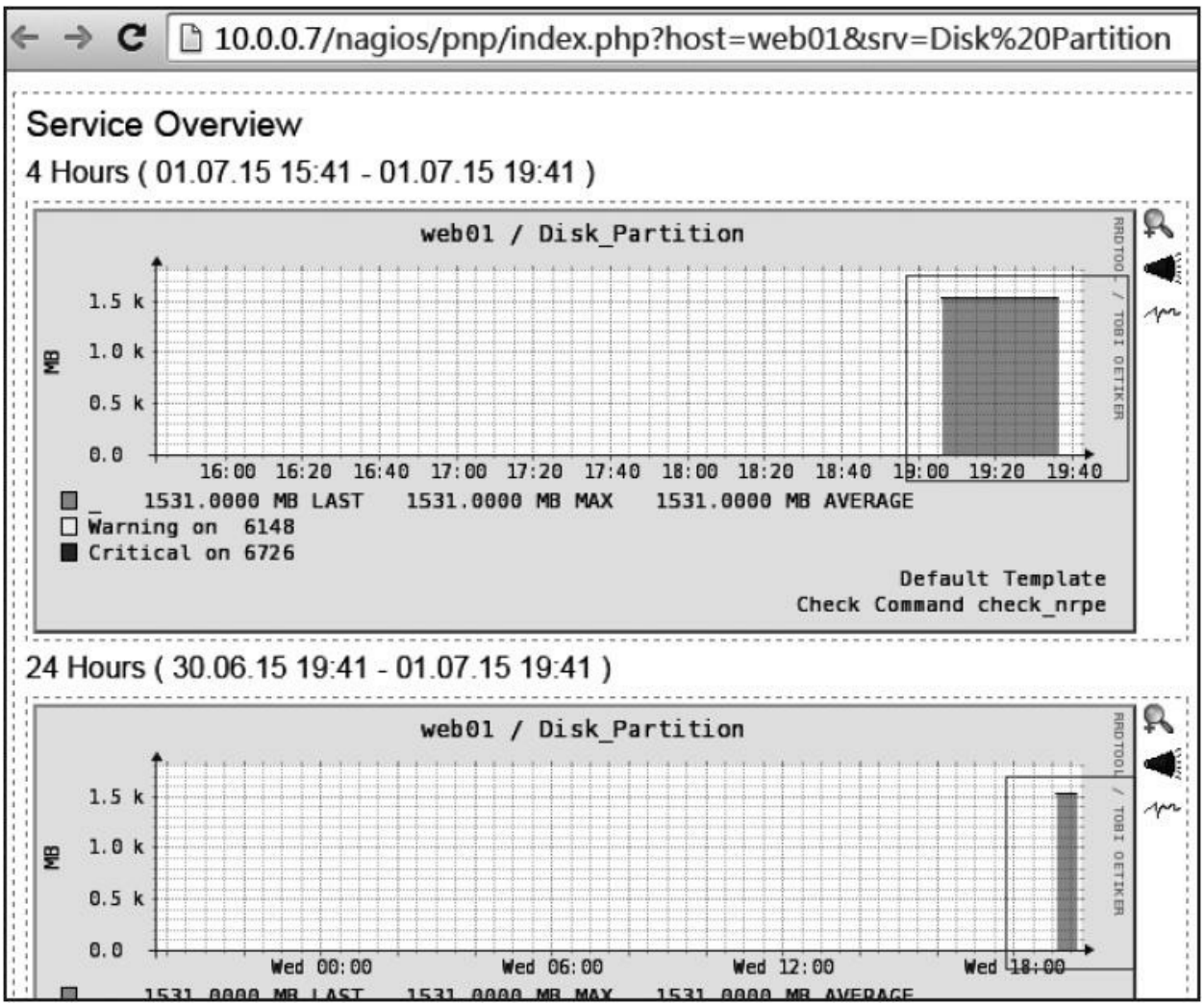


图14-25　带数据的PNP趋势图

14.6.3　整合PNP URL超链接到Nagios Web界面

在整合PNP URL超链接到Nagios Web界面后，会在所有的主机或主机对应服务的前面，出现一个闪电样的超链接图标，单击超链接，就可以查看到对应的主机或服务实际的监控状态趋势图。

1.给被监控的所有主机添加超链接图标

默认情况PNP的URL为http://10.0.0.7/nagios/pnp/index.php 和Nagios不在一个界面里，所以查看主机或服务对应的趋势图很费劲。那么有办法完善么？

最简单的方法就是直接在host.cfg里在希望出图的主机里配置如下一行参数：

```
action_url              /nagios/pnp/index.phphost=$HOSTNAME$提示：有开发经验的读者一眼就看出

URL传个主机的参数。
```

然后编辑host.cfg，增加上述配置。配置结果如下：

```
# Define a host for the local machine
define host{
        use                     linux-server
        host_name                    web01
        alias                  web01
```

```
        address                 10.0.0.8
        process_perf_data       1
        action_url            /nagios/pnp/index.phphost=$HOSTNAME$
                            #<==添加超链接图标



        }
define host{
        use                 linux-server
        host_name                 web02
        alias                 web02
        address                 10.0.0.9
        process_perf_data       1
        action_url            /nagios/pnp/index.phphost=$HOSTNAME$
                            #<==添加超链接图标



        }
```

接着，检查语法并重新加载Nagios。

```
[root@nagios-server objects]# /etc/init.d/nagios reload
Running configuration check...done.
Reloading nagios configuration...done
```

如果配置过程都正确，打开浏览器访问Nagios界面，最终可以看到如图14-26所示的图形。图中，右边方框里标记的白色方格里，中间带波浪线的就是超链接图标。单击进去即可看到一个主机所有的服务图。

2.给被监控主机指定的服务添加超链接图标

和上述主机添加超链接图标的配置几乎一样，执行“vi/usr/local/nagios/etc/objects/services.cfg”，添加如下内容：

```
action_url /nagios/pnp/index.phphost=$HOSTNAME$&srv=$SERVICEDESC$
```

有开发经验的读者一眼就能看出，这里实际就是给URL传了一个主机的参数和一个主机对应服务的参数。

给具体服务增加超链接配置方法是，直接在define service{}大括号中增加参数即可，具体配置的内容如下“action_url参数部分 ”：

```
define service {
        use                     generic-service
        host_name                       web01,


web02
        service_description             Current Load
        check_command                check_nrpe!


check_load
        action_url /nagios/pnp/index.phphost=$HOSTNAME$&srv=$SERVICEDESC$



}
```

指定的单个服务出图的效果如图14-27所示。

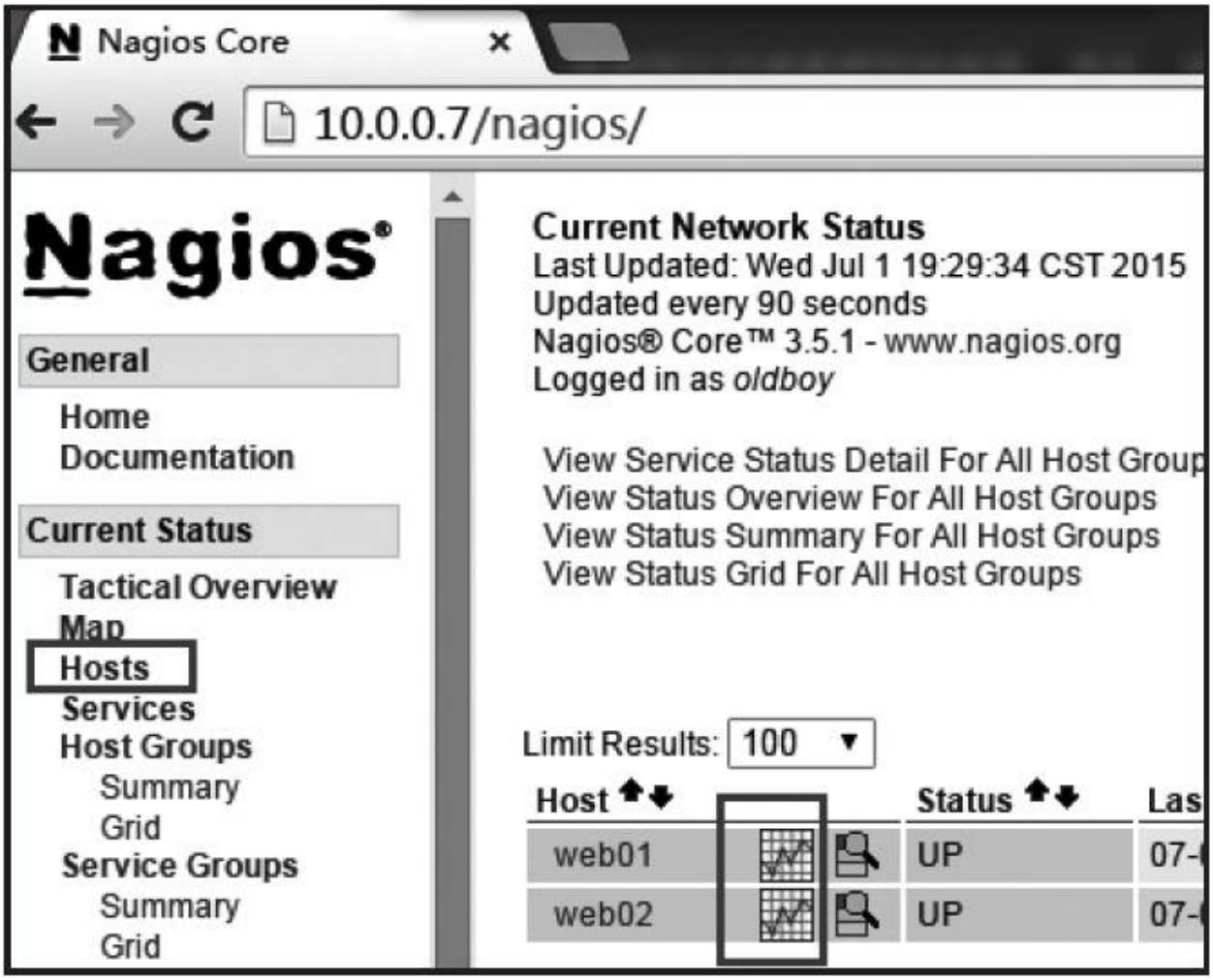


图14-26　配置超链接查看对应项趋势图

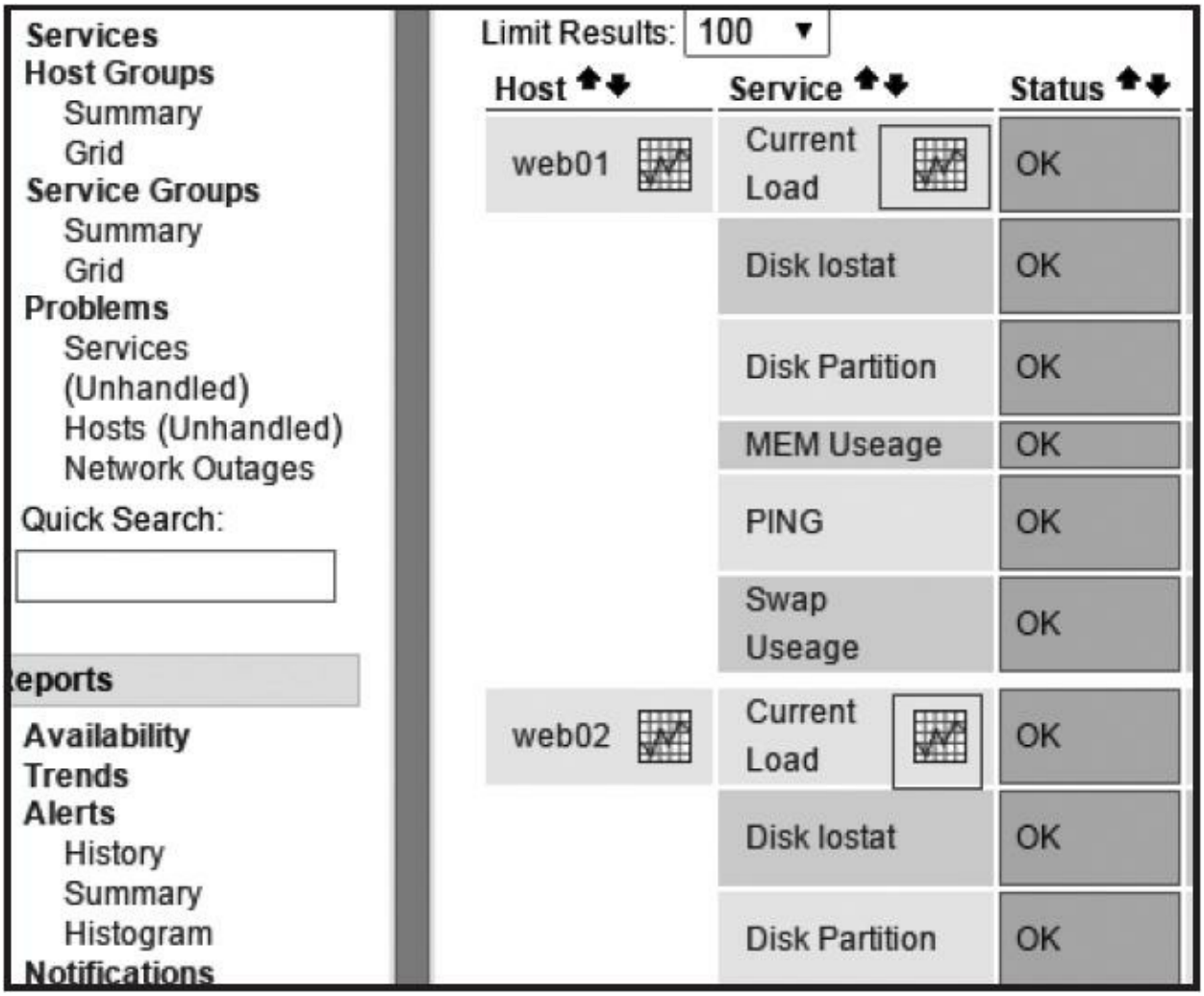


图14-27　配置指定的单个服务设置超链接出图

也可以快速设置让全部的服务出图，找到templates.cfg模板文件，找到默认的服务名generic-service，在这个服务名大括号的内部结尾增加“action_url/nagios/pnp/index.phphost=$HOSTNAME$&srv=$SERVICED行即可。

```
[root@nagios-server objects]# sed -n '154,

177p' templates.cfg
        name                    generic-service
```

```
        active_checks_enabled          1
        passive_checks_enabled          1
        parallelize_check          1
        obsess_over_service          1
        check_freshness               0
        notifications_enabled          1
        event_handler_enabled          1
        flap_detection_enabled          1
        failure_prediction_enabled     1
        process_perf_data          1
        retain_status_information     1
        retain_nonstatus_information     1
       is_volatile               0
        check_period               24x7
        max_check_attempts          3
        normal_check_interval          10
        retry_check_interval          2
        contact_groups               admins
        notification_options          w,


u,


c,


r
        notification_interval          60
        notification_period          24x7
        register                    0
        action_url /nagios/pnp/index.phphost=$HOSTNAME$&srv=$SERVICEDESC$
        }
```

这样所有主机的所有服务都将增加出图的超链接图标了。

现在，仍要检查语法并重新加载Nagios。

```
[root@nagios-server objects]# /etc/init.d/nagios reload
Running configuration check...done.
Reloading nagios configuration...done
```

全部主机和服务的监控图最终结果如图14-28所示。

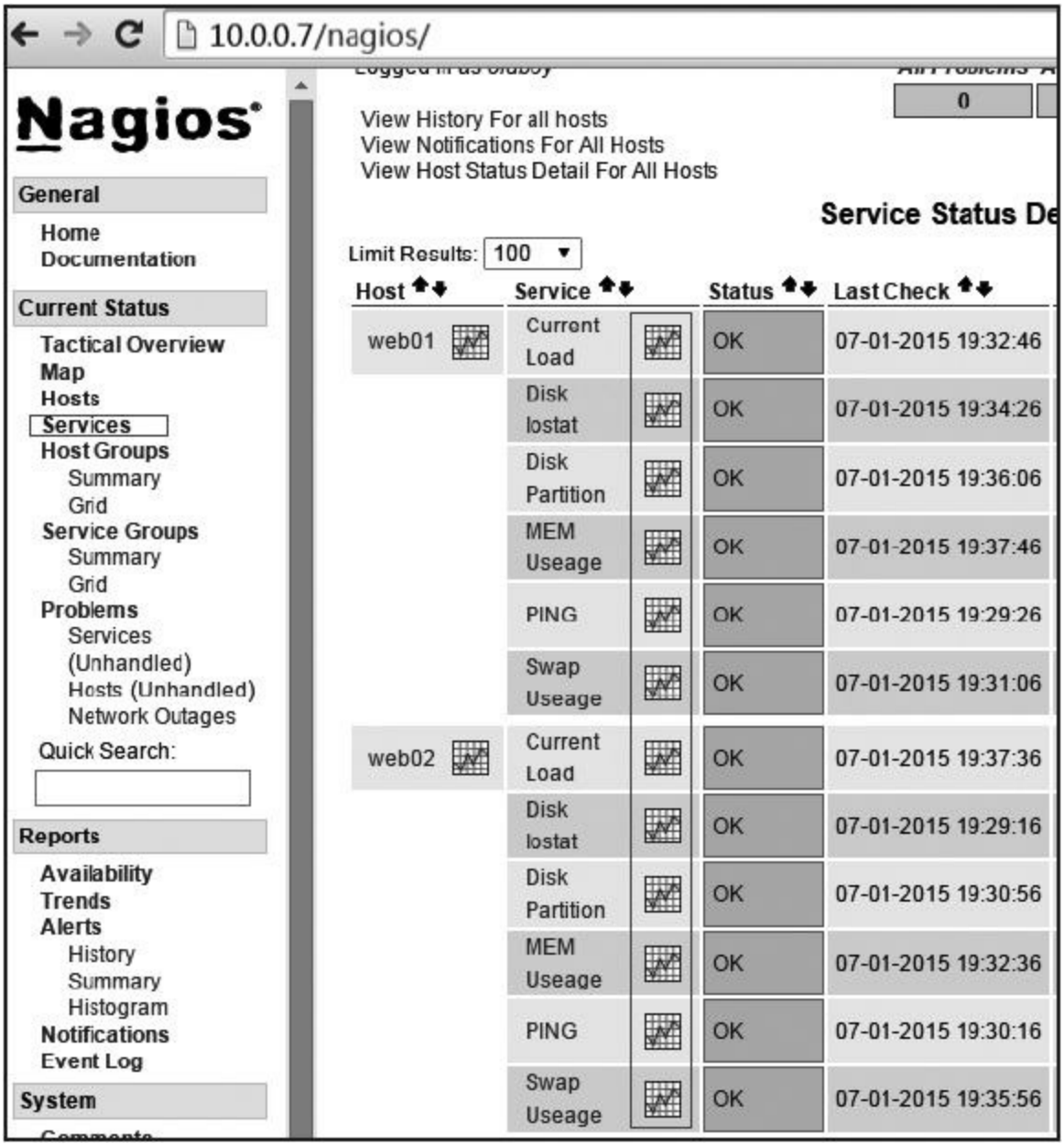


图14-28　全部主机和服务的超链接设置最终结果

此时，单击任意一个超链接图标，就可以查看对应的主机或服务的业务趋势图了，到此，Nagios的主机和服务出图的配置就完成了，是不是很简单？

3.Nagios出图获取的数据存放路径

想真正绘制出业务的趋势图全靠下面命令生成的数据。这些历史数据要备份好。

```
[root@nagios-server objects]# ll /usr/local/nagios/share/perfdata/
total 8
drwxr-xr-x 2 nagios nagios 4096 Jul  1 19:



37 web01
drwxr-xr-x 2 nagios nagios 4096 Jul  1 19:



39 web02
[root@nagios-server objects]# tree /usr/local/nagios/share/perfdata/
/usr/local/nagios/share/perfdata/
|-- web01
|   |-- Current_Load.rrd
|   |-- Current_Load.xml
|   |-- Disk_Iostat.rrd
|   |-- Disk_Iostat.xml
|   |-- Disk_Partition.rrd
|   |-- Disk_Partition.xml
|   |-- MEM_Useage.rrd
|   |-- MEM_Useage.xml
|   |-- PING.rrd
|   |-- PING.xml
|   |-- Swap_Useage.rrd
|   `-- Swap_Useage.xml
`-- web02
    |-- Current_Load.rrd
    |-- Current_Load.xml
    |-- Disk_Iostat.rrd
    |-- Disk_Iostat.xml
    |-- Disk_Partition.rrd
    |-- Disk_Partition.xml
    |-- MEM_Useage.rrd
    |-- MEM_Useage.xml
    |-- PING.rrd
    |-- PING.xml
    |-- Swap_Useage.rrd
    `-- Swap_Useage.xml
2 directories,


 24 files
```

14.7　实现将Nagios故障报警给管理员

要将Nagios故障报警给管理员时，常用的方式包括邮件报警和手机报警，下面分别介绍。

1.邮件报警

普通邮件报警就是在故障发生或恢复时，将报警信息发到系统管理员或相关维护人员的信箱中，一般来说最好使用公司内部信箱作为报警信箱。读者线下学习测试时如果用QQ、126等信箱可能会有收不到邮件的情况或者被当做垃圾邮件了，可以采用第三方SMTP服务测试，其测试报警的效果会好很多，前文已经讲过，不再赘述。

一般白天上班时，邮件报警还算比较及时，但是如果人不在计算机旁（开会、休息时），邮件报警就不行了，因此，邮件报警只适合不是特别重要的业务，或者作为发送大量报警信息中的一个辅助方式，如硬盘、内存、及日志相关等不需要及时解决的服务报警。故而，在生产环境中，邮件报警一般会结合其他报警方式一起使用。

那么，下面就来看一下邮件报警的基本配置方法。

首先，添加监控报警的接收Email地址。

```
[root@nagios-server nagios]# vi /usr/local/nagios/etc/objects/contacts.cfg +35 修改如
```

```
email                   nagios@localhost改为：


email                   31333741@qq.com
#2.快速修改方法：


sed -i 's#nagios@localhost#31333741@qq.com#' /usr/local/nagios/etc/objects/contacts.
```

实现发邮件功能常见有两种方案。

第一种方案：依赖本机的sendmail或postfix服务，可执行/etc/init.d/postfix start开启。

```
[root@nagios-server ~]# /etc/init.d/postfix restart
Shutting down postfix：


                            [  OK  ]
Starting postfix：


                                [  OK  ]
[root@nagios-server ~]# lsof -i ：


25
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
master  1399 root   12u  IPv4  10507      0t0  TCP localhost：


smtp （


LISTEN）
```

```
master  1399 root   13u  IPv6  10509      0t0  TCP localhost:

smtp (

LISTEN)
```

如果postfix启动比较慢，可以修改/etc/hosts，如下：

```
[root@nagios-server ~]# cat /etc/hosts
127.0.0.1     localhost localhost.localdomain localhost4 localhost4.localdomain4: :

1     localhost localhost.localdomain localhost6 localhost6.localdomain6
10.0.0.7 nagios-server                    #<==增加主机名和

IP的解析
```

第二种方案：使用第三方的smtp服务（网易、QQ等），即在网易等注册好邮箱，然后通过SMTP服务发送邮件。

在command.cfg里，Nagios默认的报警配置信息如下：

```
[root@nagios-server ~]# cd /usr/local/nagios/etc/objects/
[root@nagios-server objects]# sed -n '27,

37p' commands.cfg
# 'notify-host-by-email' command definition      #<==主机报警的命令配置
```

```
define command{
        command_name    notify-host-by-email
        command_line          /usr/bin/printf "%b" "***** Nagios *****\n\nNotificati
```

```
 $NOTIFICATIONTYPE$\nHost:
```

```
 $HOSTNAME$\nState:
```

```
 $HOSTSTATE$\nAddress:
```

```
 $HOSTADDRESS$\nInfo:
```

```
 $HOSTOUTPUT$\n\nDate/Time:
```

```
 $LONGDATETIME$\n" | /bin/mail -s "** $NOTIFICATIONTYPE$ Host Alert:
```

```
 $HOSTNAME$ is $HOSTSTATE$ **" $CONTACTEMAIL$                    #<==这里是具体的调用
```

```
linux
```

下

```
mail命令发送的命令
```

```
        }
# 'notify-service-by-email' command definition
define command{
        command_name    notify-service-by-email     #<==服务报警的命令配置
```

```
        command_line          /usr/bin/printf "%b" "***** Nagios *****\n\nNotificati
```

```
 $NOTIFICATIONTYPE$\n\nService:
```

```
 $SERVICEDESC$\nHost:

 $HOSTALIAS$\nAddress:

 $HOSTADDRESS$\nState:

 $SERVICESTATE$\n\nDate/Time:

 $LONGDATETIME$\n\nAdditional Info:

\n\n$SERVICEOUTPUT$\n" | /bin/mail -s "** $NOTIFICATIONTYPE$ Service Alert:

 $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **" $CONTACTEMAIL$
#<==这里是具体的调用

linux下

mail命令发送的命令


        }
```

上述notify-host-by-email和notify-service-by-email在templates.cfg的联系人模板下默认已配置（如果还需要配置短信、飞信、微信等需要在后面追加命令）。

```
[root@nagios-server objects]# sed -n '28,

37p' templates.cfg|sed -r 's#(
```

```
.*) ;


.*$#\1#g'
define contact{
        name                    generic-contact
        service_notification_period     24x7
        host_notification_period     24x7
        service_notification_options     w,


u,


c,


r,


f,


s
        host_notification_options     d,


u,


r,


f,


s
        service_notification_commands     notify-service-by-email
        host_notification_commands     notify-host-by-email
        register                    0
        }
```

只要报警联系人使用了generic-contact作为配置模板（这也是默认的配置），那么就可以进行主机和服务报警了。

由于报警信息不仅仅是要通过邮件发送，可能还涉及短信，而短信一般都有字符数限制（70字以内），因此在默认的报警内容比较多的情况下，要优化报警的内容，只报关键的信息，详细的信息让其查看邮件正文。调整如下：

```
# 'notify-host-by-email' command definition
define command{
    command_name     notify-host-by-email
    command_line     /usr/bin/printf "%b" "***** Nagios *****\n\nNotification Type:



$NOTIFICATIONTYPE$\nHost:


 $HOSTNAME$\nState:



$HOSTSTATE$\nAddress:


 $HOSTADDRESS$\nInfo:



$HOSTOUTPUT$\n\nDate/Time:


 $LONGDATETIME$\n" | /bin/mail -s"Host $HOSTSTATE$ alert for $HOSTNAME$!


"
$CONTACTEMAIL$
}
```

经过上述修改后，报警邮件的标题就简化了，例如：某主机的报警“Host DOWN alert for idc-11-200”，意思是idc-11-200主机宕机了，清

晰明了。

```
# 'notify-service-by-email' command definition
define command{
    command_name     notify-service-by-email
    command_line     /usr/bin/printf "%b" "***** Nagios *****\n\nNotification Type:


$NOTIFICATIONTYPE$\n\nService:


 $SERVICEDESC$\nHost:


 $HOSTALIAS$\nAddress:


 $HOSTADDRESS$\nState:


$SERVICESTATE$\n\nDate/Time:


 $LONGDATETIME$\n\nAdditional Info:


\n\n$SERVICEOUTPUT$" | /bin/mail -s
"$HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ " $CONTACTEMAIL$
```

同样，若服务报警的邮件标题为“10-web01/80_url is CRITICAL”，则意思是10-web01主机80_url有严重故障了。

 说明：工作中的报警需要遵循一个原则，“该报的一定要报出来，不该报的就一定不要报出来”。这句话的要求可能太完美，难以做到，但是我们可以无限接近。

2.手机短信报警

手机短信报警可采用139、126、189等信箱，或者飞信、SMS网关。一般用于紧急的业务报警。

·飞信报警：装个飞信客户端，把对方手机加为好友（需要对方确认），然后就可以给对方发短信了。

·邮件转短信报警：实现信箱邮件通知手机的一个功能。

·http短信网关：此为收费项目。

购买后的http短信网关地址类似如下：

```
http:

// s.ccme.cc/qxt/send.jspcircle=oldboy&pwd=oldboy123&mobile=$CONTACT&service=abcd54(
```

·购买短信猫：这是类似手机终端一样的客户端硬件设备，监控服务器调用短信猫实现发短信报警，图14-29就是短信猫设备。

 说明：其实开发任何程序对于Nagios来说都只是一个插件而已，因此，监控内存，短信报警，以及其他业务类插件的开发和部署方法是一样的。仅仅是功能和名字不同而已。

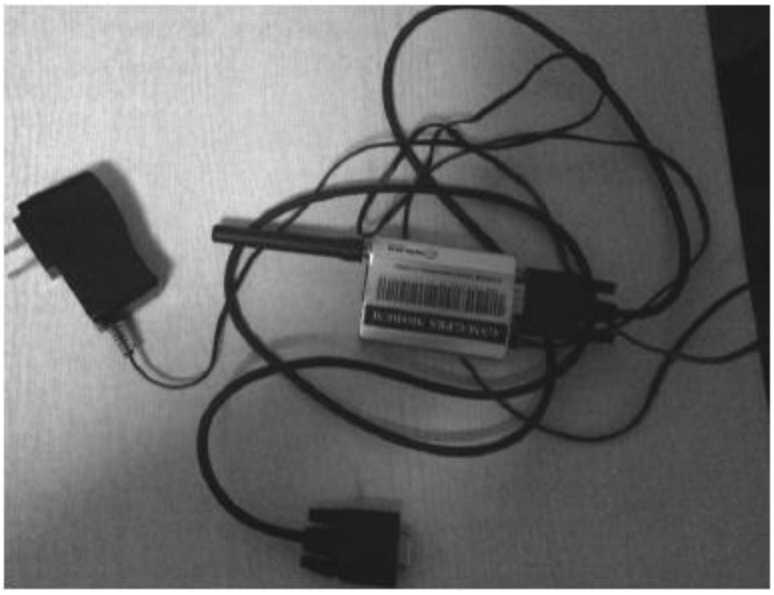


图14-29　短信猫设备图

经过多年的实践，对Nagios报警的使用，老男孩推荐采用HTTP短信网关接口方式，有专门的公司提供直接发送信息到手机的短信网关，常用的报警命令就是一个URL地址携带传参信息的形式，报警执行的命令展示如下。

```
curl -d  cdkey=3ADK-DFY-3430-MADQK -d password=52324235 -d phone=$CONTACT -d message

// sdkhttp.eucp.b2m.cn/sdkproxy/sendsms.action
```

这里的curl-d用于把参数传给后面的URL，URL是花钱购买的短信发送接口。

在赋予执行权限后，手工执行“/usr/local/nagios/libexec/sms_send内容加手机号”来进行测试。

推荐原因：

1）收费合理，报警比较及时，服务有保证。

2）设置简单，一个简单脚本搞定。

部分公司为了省钱，使用139和飞信、微信等短信报警方式，老男孩认为除非业务宕机确实没关系，最好还是不要采用这种方式。适当的花费，很好的解决报警问题才是最好的运维策略，不花钱的报警，服务上和收费的相比还是差距不小的，作为辅助方式可以。下面针对HTTP短信网关的报警进行介绍。

在购买短信服务后，除了有界面可以发送报警（一般市场人员用）外，还会有一个URL地址，地址后面携带账号、密码、及报警信息。URL地址如下面的形式：

```
http:

// s.ccme.cc/qxt/send.jspcircle=oldboy&pwd=oldboy123&
mobile=$CONTACT&service=abcd546-eee6-gg69-gg40-3gg0524c1f88d&msgid=23224&message=$T
```

在这个地址中的信息如表14-6所示。

表14-6　HTTP短信网关携带的信息列表

用户名	验证密码	目标手机	消息内容
oldboy	oldboy123	$CONTACT，脚本变量	$TITLE[${alert_date}sa] 消息变量及时间变量

实际发送命令演示如下：

```
#curl方式



curl   -d  cdkey=3RTY-EMY-0980-MTUQ2 -d password=189162 -d phone=$CONTACT -d message



// sdkhttp.eucp.b2m.cn/sdkproxy/sendsms.action
#wget --quiet "http:



// s.ccme.cc/qxt/send.jspcircle=159net_131&pwd=oldboy123&mobile=18911718229&service=
```

下面介绍一下实现短信网关设备报警的细节。首先，开发报警脚本sms_send放在libexec目录下面，授权755。

```
[root@nagios-server ~]# cd /usr/local/nagios/libexec/
[root@nagios-server libexec]# cat -n sms_send
     1#!



/bin/sh
     2PROGNAME=`basename $0`
     3PROGPATH=`dirname  $0`
     4print_usage()



 {
     5  echo "Usage:



 /bin/sh  $PROGNAME title contact"
     6  exit 1
     7}
     8if [ $# -ne 2 ];



then
     9  print_usage
    10fi
    11alert_date=$(
```

```
date +%y-%m-%d" "%H:

%M)

    12TITLE=$1                #<==发送主题，即短信内容

    13CONTACT=$2           #<==发送地址，手机号

    14#FORMAT "Host $HOSTSTATE$ alert for $HOSTNAME$"
    15curl -d cdkey=3RTY-EMY-0980-MTUQ2 -d password=189162 -d phone=$CONTACT -d mess

// sdkhttp.eucp.b2m.cn/sdkproxy/sendsms.action
[root@nagios-server libexec]# chmod +x sms_send
```

然后在command.cfg中定义报警脚本。

```
#command.cfg
# 'notify-host-by-pager' command definition
define command{
        command_name    notify-host-by-pager
        command_line      $USER1$/sms_send "Host $HOSTSTATE$ alert for $HOSTNAME$"
        #$USER1$/sms_send就是调用上面

/usr/local/nagios/libexec/sms_send脚本

        #"Host $HOSTSTATE$ alert for $HOSTNAME$" 是主机报警标题，即短信内容

        #$CONTACTPAGER$是报警联系人，即手机号

        }
```

```
# 'notify-service-by-pager' command definition
define command{
        command_name    notify-service-by-pager
        command_line    $USER1$/sms_send "$HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE
        #$USER1$/sms_send就是调用上面


/usr/local/nagios/libexec/sms_send脚本


        #"$HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$" 是服务报警标题，即短信内容


        #$CONTACTPAGER$是报警联系人，即手机号


        }
```

接着在模板templates.cfg中增加如下配置，即在command.cfg中定义报警脚本里短信报警的命令名，编辑templates.cfg里的定义内容为：

```
define contact{
        name                    generic-contact
        service_notification_period     24x7
        host_notification_period     24x7
        service_notification_options     w,


u,


c,


r,


f,


s
        host_notification_options     d,
```

```
u,


r,


f,


s
        service_notification_commands     notify-service-by-email,


notify-service-by-pager
        host_notification_commands     notify-host-by-email,


notify-service-by-pager
        register                    0
}
```

最后在联系人contact.cfg里做如下联系人定义。

```
define contact{
        contact_name                    oldboy-pager
        use                             generic-contact
        alias                            Nagios users
        pager                           18911111111
}
define contactgroup{
        contactgroup_name         admins
        alias                     Nagios Administrators
        members                   nagiosadmin,


oldboy-pager
        }
```

这样就可以实现通过HTTP短信网关报警了。

14.8　Nagios插件开发

14.8.1　概述

1.什么是Nagios插件

前文在部署Nagios服务时已安装了nagios-plugins-1.4.16.tar.gz，这个软件包就是Nagios的插件安装包，安装后，执行ls-l/usr/local/nagios/libexec可以看到如下插件内容：

```
[root@nagios-server ~]# ls -l /usr/local/nagios/libexec/|egrep "nrpe|tcp|http|disk|s
lrwxrwxrwx 1 root   root          9 Jun  1 08:


16 check_clamd -> check_tcp
-rwxr-xr-x 1 nagios nagios 118205 Jun  1 08:


16 check_disk
-rwxr-xr-x 1 nagios nagios   8163 Jun  1 08:


16 check_disk_smb
lrwxrwxrwx 1 root   root          9 Jun  1 08:


16 check_ftp -> check_tcp
-rwxr-xr-x 1 nagios nagios 190806 Jun  1 08:


16 check_http
lrwxrwxrwx 1 root   root          9 Jun  1 08:


16 check_imap -> check_tcp
lrwxrwxrwx 1 root   root          9 Jun  1 08:
```

```
16 check_jabber -> check_tcp
-rwxr-xr-x 1 nagios nagios  73784 Jun  1 08:


16 check_load
-rwxrwxr-x 1 nagios nagios  62381 Jun  1 08:


24 check_nrpe
-rwxr-xr-x 1 nagios nagios  78020 Jun  1 08:


16 check_swap
-rwxr-xr-x 1 nagios nagios 105775 Jun  1 08:


16 check_tcp
lrwxrwxrwx 1 root   root          9 Jun  1 08:


16 check_udp -> check_tcp
...省略部分


...
```

提示：默认安装后大概有60个左右的插件，数量比较多，这里只介绍几个常见的。

以上结果内容都是Nagios的插件，现在大家应该对Nagios插件有一个基本的了解了。其实，Nagios软件本身仅仅是一个监控的平台，如果要监控具体的主机及服务的状态和数据信息，还必须配置或调用插件或程序文件才能完成任务，因此，如果没有Nagios插件，Nagios就是一个空壳，啥都做不了。

2.为什么要开发Nagios插件

既然已经安装了Nagios的插件软件包，为什么还要开发Nagios插件呢？

首先想说明的是，在生产场景中常用的大部分服务都是不需要编写插件就可以完成监控的，check_http、check_tcp、check_nrpe等这些自带的插件已经很强大了。但是，仍然有部分我们想要监控的服务，是Nagios未自带插件的，如：监控LVS RS的lo网卡的VIP，监控Nfs的状态，又或者监控iostat、mem、sar系统指标及相关APP应用（MQ队列）等。这个时候我们有两个选择，一个是去网上搜索，看看有没有别人写过的脚本，拿来使用或修改后使用；另外就是自己开发编写脚本。这里建议大家学会手工编写插件，如果开始不会写，可以把网上别人分享的插件拿来改，改着改着就会写了。

如果要开发插件，最好掌握一门开发语言，例如：Shell、Python，这部分开发内容需要读者自行学习，或者参加老男孩老师开办的培训班学习获得。

14.8.2　编写Nagios插件的规则

1.编写Nagios插件说明

Nagios插件是Nagios提供的一种通过可扩展的方式部署的程序组件，该插件可通过Shell、Java、C\C++、PHP等多种语言开发，运维或系统架构人员只要通过修改Nagios配置文件和相应参数，就能很方便地将该插件集成到Nagios中，实现对目标系统的监控。

Nagios服务为插件程序 提供了两个返回值接口和插件交互：一个是插件执行后的退出状态码 ，另一个是插件执行过程中在控制台打印的第一行数据 。退出状态码可以被Nagios主程序作为判断被监控系统服务状态的依据，控制台打印的第一行数据可以被Nagios主程序作为被监控系统服务状态的补充说明，会显示在Web管理页面里，图14-30中方框为显示的内容。

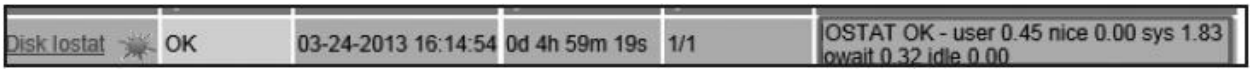


图14-30　Nagios界面报警显示的内容

为了管理Nagios插件，Nagios每查询一个服务的状态时，就会产生一个子进程，并使用来自该命令的输出和退出代码来确定其具体的状态。Nagios主程序可识别的插件的推出状态码和说明如下：

·OK：退出代码，0表示服务工作正常。

·WARNING：退出代码，1表示服务处于警告状态。

·CRITICAL：退出代码，2表示服务处于紧急，严重状态。

·UNKNOWN：退出代码，3表示服务处于未知状态。

 注意：此处数字代码的含义曾经有公司面试时考过。

最后一种状态通常表示该插件无法确定服务的状态。例如，可能出现了网络或内部错误。相关状态可以从如下文件中看到：

```
[root@nagios-server libexec]# head -7 utils.sh
#!


 /bin/sh
STATE_OK=0
STATE_WARNING=1
STATE_CRITICAL=2
STATE_UNKNOWN=3
STATE_DEPENDENT=4
```

 提示：结尾处比前文列举的还多了个状态，但不常用。

2.Nagios插件开发原理

Nagios插件程序中需要调用监控服务规定的操作序列，并根据预先定义的规则，对返回结果进行分析，判断服务的当前状态，然后以指定的状态码退出程序，同时将对该状态的说明不换行输出到控制台。

不同语言的系统退出函数的示例如下：

```
Java     System.exit (

int status)

php     exit (

status)

python     sys.exit (

int status)

c/c++     return int status
bash     exit int status
```

不同语言的控制台打印函数的示例如下：

```
Java     System.out.println (

String msg)

php     echo msg
python     print msg
c/c++     printf (

"%s",
```

```
msg)

bash     echo msg (

printf)
```

3.Nagios插件开发语言

Nagios的插件开发不限制任何开发语言，只要该插件能被Nagios调用，并获取到相应业务数据就OK，如能在命令行执行输出结果也可以，常用的插件语言有Shell、Perl、Python、PHP、C/C++。

14.8.3　使用Shell开发Nagios插件

1.编写检查WebURL地址的插件

以下脚本只是针对访问客户端10.0.0.8的IP的。

```
[root@nagios-server ~]# cd /usr/local/nagios/libexec/
[root@nagios-server libexec]# cat check_url.sh
#!


/bin/sh
wget -T 10 --spider 10.0.0.8 >/dev/null 2>&1     #<==用


wget检查


10.0.0.8是不是可以访问


if [ $ -eq 0 ]                          #<==判断上述


wget命令指令的返回值，


0为


                                成功，非


0为失败


    then
```

```
    echo "URL 10.0.0.8 OK"
    exit 0
else
    echo "URL 10.0.0.8 CRITICAL"
    exit 2
fi
```

下面利用传参把脚本改进为通用的WebURL插件。

```
[root@nagios-server libexec]# cat check_url.sh
#!


/bin/sh
PROGNAME=`basename $0`                #<==取脚本名


PROGPATH=`dirname  $0`                 #<==去脚本路径


usage（）


 {                             #<==打印帮助


    echo "Usage：


 /bin/sh  $PROGNAME url"
    exit 1
}
[ $# -ne 1 ]&&usage                 #<==参数个数不为


1，打印帮助


wget -T 10 --spider $1 >/dev/null 2>&1      #<==URL地址改成传参


if [ $ -eq 0 ]
```

```
    then
        echo "URL $1 OK"
        exit 0                    #<==返回


0值表示成功



else
        echo "URL $1 CRITICAL"
        exit 2
fi
```

以下是监控WebURL的插件脚本改进（专业规范的脚本）。

```
[root@nagios-server libexec]# cat check_url.sh
#!



/bin/sh
PROGNAME=`basename $0`
PROGPATH=`dirname  $0`
usage（）



 {
    echo "Usage:


 /bin/sh  $PROGNAME url"
    exit 1
}
[ $# -ne 1 ]&&usage
. $PROGPATH/utils.sh
if wget  -T 20 --spider $1 >/dev/null 2>&1 ;



then
        echo 'url $1 OK'
        exit $STATE_OK
else
        echo 'url $1 NO'
        exit $STATE_CRITICAL
fi
```

最后，手工测试一下改进的WebURL插件脚本。

```
[root@nagios-server libexec]# sh /usr/local/nagios/libexec/check_url
Usage:


 /bin/sh check_url url
[root@nagios-server libexec]# sh /usr/local/nagios/libexec/check_url www.etiantian.c
URL www.etiantian.org OK
[root@nagios-server libexec]# sh /usr/local/nagios/libexec/check_url oldboy.blog.51c
URL oldboy.blog.51cto.com OK
```

2.WebURL插件脚本的部署过程（主动监控方式）

Nagios主动模式监控和Nagios客户端的nrpe进程没有关系，这里停掉nrpe进程进行下面的实验。主动模式的所有操作完全在Nagios主服务器上进行。部署步骤如下。

1）开发check_url.sh，放到/usr/local/nagios/libexec中，授权为可执行。

```
[root@nagios-server ~]# cd /usr/local/nagios/libexec/
[root@nagios-server libexec]# chmod +x check_url.sh
[root@nagios-server libexec]# cat check_url.sh
#!


/bin/sh
PROGNAME=`basename $0`
PROGPATH=`dirname  $0`
usage ()


 {
    echo "Usage:


 /bin/sh $PROGNAME url"
    exit 1
}
[ $# -ne 1 ]&&usage
wget -T 10 --spider $1 >/dev/null 2>&1
if [ $ -eq 0 ]
```

```
    then
        echo "URL $1 OK"
        exit 0
else
        echo "URL $1 CRITICAL"
        exit 2
fi
```

2）在command.cfg建立check_url命令：

```
[root@nagios-server ~]# cd /usr/local/nagios/etc/objects/
[root@nagios-server objects]# vi commands.cfg          #<==到结尾增加如下内容



# 'check_url' command definition by oldboy
define command{
        command_name    check_url
        command_line    $USER1$/check_url.sh 10.0.0.8#<==加载


libexec下的脚本，

                                                并传参


10.0.0.8
        }
```

3）在services.cfg里添加监控上述URL地址的服务。

```
[root@nagios-server objects]# pwd
/usr/local/nagios/etc/objects
[root@nagios-server objects]# cd services     #<==这个是被


nagios.cfg包含的

                                                目录，所有放到目录的配置文件（以
```

.cfg结尾）都会生效

```
[root@nagios-server services]# vi check_url.cfg     #<==创建
```

check_url.cfg添加如下

对

web01，

http服务的监控

```
define service{
        use          generic-service
        host_name          web01
        service_descriptionhttp_zhudong_url     #<==http服务主动监控的名字

        check_command     check_url          #<==监控的命令，来自于
```

command.

cfg中定义好的

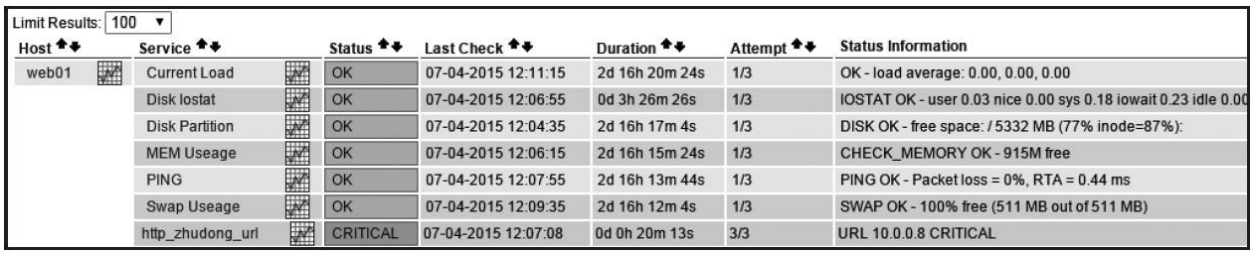
```
        }
```

4）重新加载Nagios，查看结果。

```
[root@nagios-server objects]# /etc/init.d/nagios checkconfig
#====》这个启动脚本是改过的，可以省去敲一堆字符检查语法的麻烦
```

```
Running configuration check...
...省略部分内容


...
Checking misc settings...
Total Warnings:


     0          #==>警告不为


0，可以忽略



Total Errors:


     0          #==>这里为


0就


OK。



Things look okay - No serious problems were detected during the pre-flight check
OK.
[root@nagios-server objects]# /etc/init.d/nagios reload #==>reload好于


restart
Running configuration check...done.
Reloading nagios configuration...done
```

5）查看Nagios服务页面监控结果，如图14-31所示。



Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
web01	Current Load	OK	07-04-2015 12:11:15	2d 16h 20m 24s	1/3	OK - load average: 0.00, 0.00, 0.00
	Disk Iostat	OK	07-04-2015 12:06:55	0d 3h 26m 26s	1/3	IOSTAT OK - user 0.03 nice 0.00 sys 0.18 iowait 0.23 idle 0.00
	Disk Partition	OK	07-04-2015 12:04:35	2d 16h 17m 4s	1/3	DISK OK - free space: / 5332 MB (77% inode=87%):
	MEM Useage	OK	07-04-2015 12:06:15	2d 16h 15m 24s	1/3	CHECK_MEMORY OK - 915M free
	PING	OK	07-04-2015 12:07:55	2d 16h 13m 44s	1/3	PING OK - Packet loss = 0%, RTA = 0.44 ms
	Swap Useage	OK	07-04-2015 12:09:35	2d 16h 12m 4s	1/3	SWAP OK - 100% free (511 MB out of 511 MB)
	http_zhudong_url	CRITICAL	07-04-2015 12:07:08	0d 0h 20m 13s	3/3	URL 10.0.0.8 CRITICAL

图14-31　Nagios服务页面监控结果

此时，发现设置的内容是红色报警状态，原来10.0.0.8没有Web服务，检查结果如下：

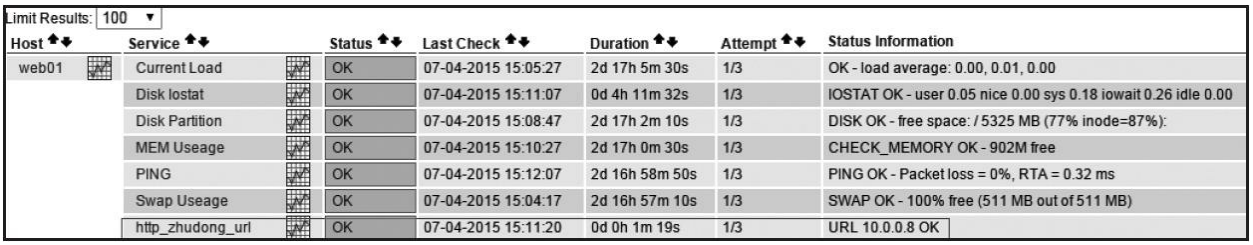
```
[root@nagios-server ~]# telnet 10.0.0.8 80
Trying 10.0.0.8...
telnet:


 connect to address 10.0.0.8:


 Connection refused
```

6）在Nagios客户端web01上安装HTTP服务进行测试。

```
[root@web01 ~]# yum install httpd -y
[root@web01 ~]# /etc/init.d/httpd start正在启动


 httpd:


httpd:


 apr_sockaddr_info_get()


 failed for web01
httpd:
```

```
 Could not reliably determine the server's fully qualified domain name,


 using 127.0.0.1 for ServerName
                                                    [确定


]
[root@web01 ~]# echo ok >/var/www/html/index.html
[root@web01 ~]# curl 10.0.0.8/index.html
ok
```

7）重新查看Nagios服务页面的监控结果，如图14-32所示。



Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
web01	Current Load	OK	07-04-2015 15:05:27	2d 17h 5m 30s	1/3	OK - load average: 0.00, 0.01, 0.00
	Disk Iostat	OK	07-04-2015 15:11:07	0d 4h 11m 32s	1/3	IOSTAT OK - user 0.05 nice 0.00 sys 0.18 iowait 0.26 idle 0.00
	Disk Partition	OK	07-04-2015 15:08:47	2d 17h 2m 10s	1/3	DISK OK - free space: / 5325 MB (77% inode=87%):
	MEM Useage	OK	07-04-2015 15:10:27	2d 17h 0m 30s	1/3	CHECK_MEMORY OK - 902M free
	PING	OK	07-04-2015 15:12:07	2d 16h 58m 50s	1/3	PING OK - Packet loss = 0%, RTA = 0.32 ms
	Swap Useage	OK	07-04-2015 15:04:17	2d 16h 57m 10s	1/3	SWAP OK - 100% free (511 MB out of 511 MB)
	http_zhudong_url	OK	07-04-2015 15:11:20	0d 0h 1m 19s	1/3	URL 10.0.0.8 OK

图14-32　重新查看Nagios服务页面的监控结果

3.利用被动模式的nrpe方式监控/etc/passwd文件是否变化

Nagios被动模式下的所有插件都需要部署在被监控的Nagios客户端。部署步骤如下。

1）在Nagios客户端web01上取/etc/passwd的文件指纹，即md5值。

```
[root@nagios-server ~]# md5sum /etc/passwd >/opt/ps.md5
[root@nagios-server ~]# cat /opt/ps.md5
c70166c5379d57f45e2ac10ed166b2ba  /etc/passwd      #<==记录好前面的一堆字符串。
```

2）在Nagios客户端web01上开发插件脚本，并测试。

```
[root@nagios-server ~]# cd /usr/local/nagios/libexec/
[root@web01 libexec]# cat check_passwd
#!


/bin/sh
OriMd5="f90679e1ff07a90181edbbf82cb10a01"     #<==/etc/passwd正确的


MD5指纹


                                        字符串


CurrMd5=`md5sum /etc/passwd|cut -c 1-32`     #<==每次执行脚本重新获取


MD5指纹


                                        和正确的指纹比较


if [ "$OriMd5" == "$CurrMd5" ]
    then
        echo "/etc/passwd:


ok"
        exit 0
else
        echo "/etc/passwd:


FAILED"
        exit 2
fi
[root@nagios-server libexec]# sh check_passwd
/etc/passwd:
```

```
ok
[root@nagios-server libexec]# chmod + x check_passwd      #<==需要具备执行权限
```

 提示：还可以用md5sum-c/opt/ps.md5的方法比较。

3）在Nagios客户端web01上编辑nrpe.cfg，插入如下的内容后保存：

```
[root@nagios-server libexec]# cd /usr/local/nagios/etc/
[root@nagios-server etc]# vim nrpe.cfg  #<==切到文件结尾增加如下内容


command[check_passwd]=/usr/local/nagios/libexec/check_passwd
```

4）在Nagios客户端web01上重启nrpe，并检查是否重启成功（check_nrpe检验）。

```
[root@web01 etc]# ps -ef|grep nrpe|grep -v grep
nagios    2398     1  0 15:


30           00:


00:


00 /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
[root@web01 etc]# pkill nrpe
[root@web01 etc]# ps -ef|grep nrpe|grep -v grep
[root@web01 etc]# /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
[root@web01 etc]# ps -ef|grep nrpe|grep -v grep
nagios    2414     1  0 15:


30           00:
```

```
00:


00 /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
```

5）在Nagios服务器端nagios-server上进入service目录，创建配置文件check_passwd_web01.cfg。

```
[root@nagios-server ~]# cd /usr/local/nagios/etc/objects/services
[root@nagios-server services]# cat check_passwd_web01.cfg
define service {
        use                     generic-service
        host_name                       web01
        service_description             check_passwd
        check_command                  check_nrpe!


check_passwd
#<==这里的


check_passwd就是


Nagios客户端


nrpe.cfg里


command[check_passwd]=/usr/local/nagios/libexec/check_passwd配置的中括号命令名


check_passwd
}
```

6）在Nagios服务器端检查语法。

```
[root@nagios-server services]# /etc/init.d/nagios checkconfig
```

```
Running configuration check...
Nagios Core 3.5.1.....



Total Warnings:


 0
Total Errors:


   0.....
```

7）在Nagios服务器端加载Nagios配置，然后打开Nagios页面查看。

```
[root@nagios-server services]# /etc/init.d/nagios reload
Running configuration check...done.
Reloading nagios configuration...done
```

8）如果出问题就进行排查。

首先，进入/usr/local/nagios/libexec/目录，然后，在Nagios服务器端检查语法，如下：

```
[root@nagios-server libexec]# ./check_nrpe -H 10.0.0.8 -c check_passwd
/etc/passwd:


ok
```

接着，在Nagios客户端检查语法，如下：

```
[root@web01 libexec]# ./check_nrpe -H 127.0.0.1 -c check_passwd
/etc/passwd:
```

```
ok
```

最后，在Nagios客户端执行nrpe里命令名后面的插件命令，即/usr/local/nagios/libexec/check_passwd看结果。

```
[root@web01 libexec]# /usr/local/nagios/libexec/check_passwd
/etc/passwd:


ok
```

14.9 常见故障问题总结

问题1：PNP出现PHP GD Support no found错误。

描述：浏览器输入http://10.0.0.7/nagios/pnp/index.php 打开页面提示如下：“PHP GD Support no found”。

原因及解决办法：php-gd模块没装，执行yum install php-gd-y安装即可。

问题2：出现报错信息“NRPE：Unable to read output”。

描述：Nagios界面报“NRPE：Unable to read output”错误。

原因及解决：这是因为客户端对应插件命令不存在或者无执行权限等原因导致的。

问题3：出现报错信息“NRPE：Command'check_passwd'not defined”

原因：这个错误提示，可能是因为服务器端的服务里配置的命令名与客户端的nrpe.cfg里配置的命令名不匹配导致的。

例如：服务器端的配置为“check_command check_nrpe！check_passwd”，其中的check_passwd部分与客户端nrpe.cfg里的命令“command[check_passwd]=/usr/local/nagios/libexec/check_passwd”里的

check_passwd不匹配。

14.10　本章重点回顾

1）Nagios监控系统家族成员功能介绍。

2）Nagios监控系统完整框架图解说明。

3）Nagios服务器端及客户端安装、配置细节。

4）Nagios服务器端核心配置文件之间的关系原理。

5）Nagios利用check_nrpe插件进行监控的原理及排错案例。

6）Nagios图形监控显示和数据管理。

7）Nagios报警方式选择及报警实施细节。

8）Nagios自定义插件开发原理及开发实践。

9）Nagios插件主动和被动方式工作原理及实施部署细节。

第15章　企业级网站集群搭建综合解决方案

本章是全书的最后一章，主要是综合前面所有的章节，把Linux Web集群运维技术由点穿成线，由线组成面，最终为用户呈现一个企业级中小规模网站集群的架构解决方案全貌。整套架构从企业的实际需求出发，以企业生产实战标准为准绳，从分析企业网站需求开始，到硬件选择、IP地址规划、主机名规划，再到部署服务的目录结构规划等，并对每一个集群节点要部署的应用做了细致的设计（甚至画了逻辑图），以使读者对企业生产场景的集群全貌有了一个深入的了解，确保能够根据所学独立实现整个集群架构的搭建。

那么，完成企业中小型网站集群有什么意义呢？

·可以检验自己学习本书前14章的效果。

·把学习的Linux运维技术由点穿成线，由线组成面。

·可以深入了解未来运维工作的实际场景和岗位技术方向。

·可了解并掌握互联网中型企业的网站集群架构细节。

此外，如果你在做集群架构时遇到问题，请加入开篇提供的交流群进行交流，老男孩老师也会在其中答疑。

15.1　企业级中小规模网站集群项目规划

为什么本章要重点讲解企业级中小规模网站集群架构而不是大规模门户网站呢？因为第一，国内99%的企业网站都是中小规模的，例如大家熟知的51CTO、CSDN等网络平台都是中小规模网站集群，且一般成立1~2年的公司网站其规模大多都还是比较小的。第二，中小规模集群技术容易落地、实用，对新手来讲摸得着看得见，学会了就直接上手用于企业工作中，算是最直接最有效的雪中送炭型帮助。第三，所有的大规模网站集群都是从中小规模集群扩展起来的，整个核心还是中小规模的架构框架，就像成人一样，不管是身高2.26米的球星姚明，还是身高1.75米的老男孩，人体的骨架核心都是一样的。

15.1.1　企业级中小规模网站集群架构逻辑图及说明

图15-1为多数中小企业级网站采用的集群架构逻辑图全貌，下文会一一为大家讲解每个区域的设计思路。

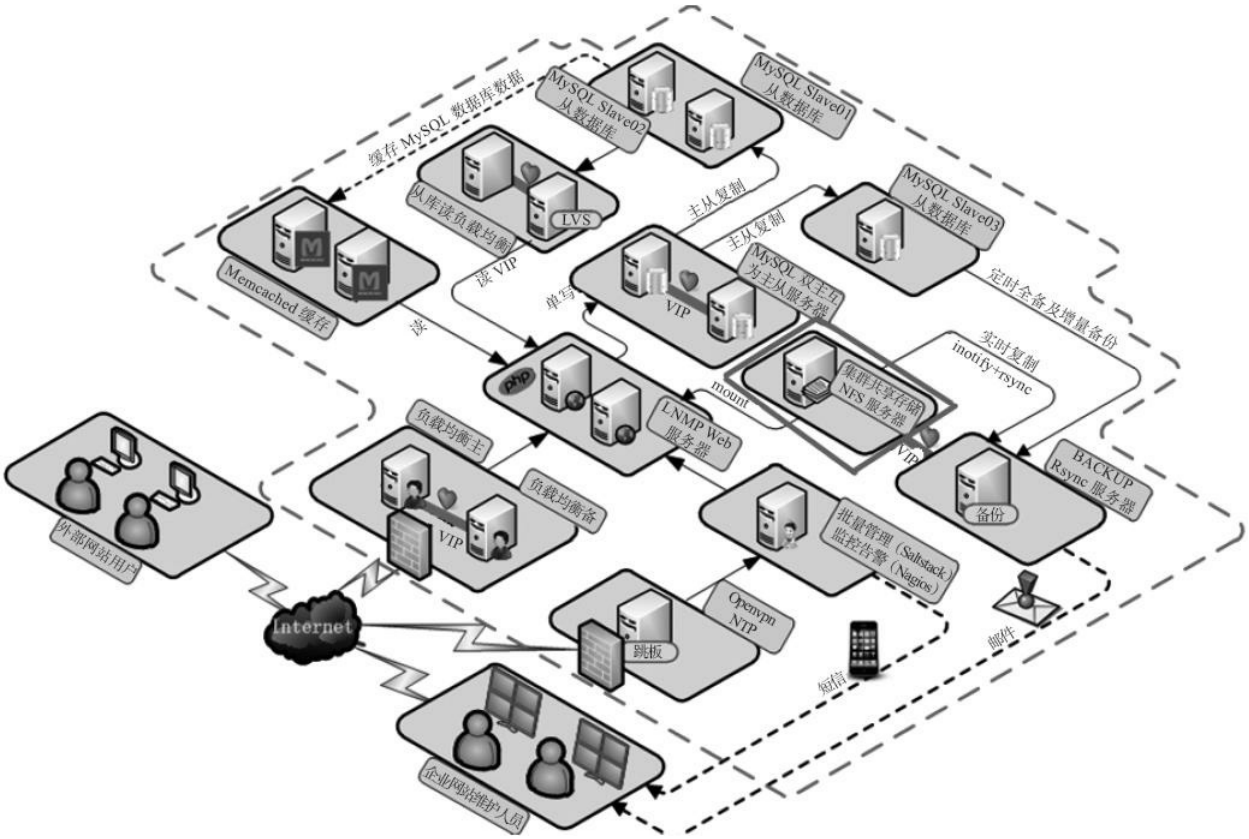


图15-1　企业常见中小规模网站集群架构逻辑图

图15-1对应的企业网站各项指标参考如下：

·服务器数量可以从50台到150台（含生产线应用的云主机及虚拟

机）。

·可以支撑的最高网站流量大小为日PV2000万左右，可支撑高峰期并发过万的连接数访问，具体能承受的最大日PV和并发数取决于具体的业务及用户访问是否集中，甚至包括程序及架构是否做了更多的优化，例如：动态数据静态化、采用CDN加速服务等。

·运维部的人员数量一般为3~5人，以5个为例，可以为1个运维总监，1个企业网管，1个初级运维，1个中高级运维，1个高级运维兼运维开发。

·正常情况下，这个规模的网站集群消耗的IDC带宽总大小为100MB独享，具体带宽大小取决于不同的网站业务类型（例如：在线视频、电商、网络平台、移动APP），以及网站是否实现动态数据静态化，是否把静态资源通过CDN加速服务提供给用户访问等。

 提示：以上各项数字指标为老男孩根据多年工作及教育经验推论的结果，会有误差，仅供读者参考，目的是为了让网站架构有一个清晰可以量化的数值指标，帮助读者更好地理解生产场景的集群架构规模与服务器、维护人力、用户流量、带宽流量的关系。

如果读者能够独立搭建完本章的集群架构，基本可以胜任诸如51CTO、CSDN等中小网站的独立运维工作了，也可以胜任诸如赶集、58同城等中型网站的初中级运维工作，更可以胜任各类移动APP互联网

公司及各类游戏公司的运维工作。

15.1.2　集群服务器硬件及操作系统规划

如果有条件，可以准备一台物理服务器，比如DELL R710，或者干脆使用高配笔记本（台式机计算机也可），然后在宿主机计算机上虚拟出10台VM虚拟机，当然，能有多台物理服务器环境更佳，可全部采用CentOS 6.6 x86_64操作系统来完成整个集群架构的搭建，如果对Docker容器技术了解，使用Docker容器完成集群实验效果更佳。

搭建本集群架构可选的第一套参考环境为：DELL R710服务器1台，具体配置参考表15-1和表15-2。

表15-1　采用单台硬件服务器搭建集群的配置参考

设备名称	配置描述
CPU：	两颗四核至强处理器 Intel(R) Xeon(R) CPU E5606 @ 2.13GHz
MEM：	16GB (4x8GB) 667MHz
Raid：	SAS Raid 0 SAS6I Raid 卡（支持 1/0/5/6/10）
DISK：	300GB × 3 3.5 英寸 SAS 硬盘 15k
网卡：	集成 4 个 Broadcom 千兆网卡支持 TOE 功能

表15-2　软件配置参考

软件名称	配置描述
CentOS 6 操作系统	以 64 位 CentOS 系统作为 10 个 VM 的宿主机
KVM：	采用开源软件 KVM，虚拟 10 个 VM


提示：本书内容是Windows下安装VMware虚拟软件，然后搭建虚拟机完成的，并全部通过企业级物理服务器硬件环境进行了二次测

试。在这里特别感谢中网志腾及实利通和设备提供商为笔者写书提供了共4台物理服务器测试环境。

测试本集群架构可选的第二套参考环境为：笔记本计算机1台，具体配置见表15-3和表15-4。

表15-3　采用硬件笔记本或台式机测试搭建集群的配置参考

设备名称	配置描述
CPU：	1 颗四核 Intel(R) I5-3470 CPU @3.20GHz 3.60GHz
MEM	16GB
DISK	SSD 512GB

表15-4　软件配置参考

软件名称	配置描述
Window7 操作系统	以 64 位 Window7 系统作为 10 个 VM 的宿主机
VMware workstation	采用 VMware workstation8，虚拟 10 个 VM


提示：考虑到大部分读者没有物理服务器的环境，因此，本书的核心就是通过台式机、笔记本等安装Windows，然后通过VMware workstation搭建的简单虚拟化环境来实现集群方案，这套环境每个学习者都可以搞定。

读者可以根据自己的硬件环境选择实现本章的集群架构，可以将多个服务部署在一台VM虚拟机上，减少虚拟机的浪费，还可以使用Docker容器技术来完成本书的集群架构方案。

15.1.3　集群节点的IP地址及主机名规划

1.IP地址及主机名规划表

集群节点IP地址及主机名规划见表15-5和表15-6。

表15-5　集群节点IP地址及主机名规划

服务器说明	外网 eth0	VIP	内网 eth1	名称规划	域名规划
反向代理服务 01	10.0.0.1	10.0.0.3	172.16.1.1	LB-nginx-01	lb01.etiantian.org
反向代理服务 02	10.0.0.2		172.16.1.2	LB-nginx-02	lb02.etiantian.org
nginx web01			172.16.1.10	WEB-LNMP-01	web01.etiantian.org
nginx web02			172.16.1.11	WEB-LNMP-02	web02.etiantian.org
NFS 存储服务		172.16.1.32	172.16.1.30	STORE-NFS-01	nfs.etiantian.org
rsync 备份服务			172.16.1.31	STORE-Rsync-01	rsync.etiantian.org
MySQL 主数据库		172.16.1.52	172.16.1.50	MySQL-master-01	dbmaster01.etiantian.org
MySQL 主热备数据库			172.16.1.51	MySQL-master-02	dbmaster02.etiantian.org
MySQL 从数据库			172.16.1.53	MySQL-slave-01	dbslave01.etiantian.org

（续）

服务器说明	外网 eth0	VIP	内网 eth1	名称规划	域名规划
Memcached 缓存及 session 共享 1		172.16.1.63	172.16.1.61	Cache-Mc-01	cache01.etiantian.org
Memcached 缓存及 session 共享 2			172.16.1.62	Cache-Mc-02	cache02.etiantian.org
监控管理服务			172.16.1.200	MAGE-monitor-01	monitor.etiantian.org
跳板机管理服务	10.0.0.201		172.16.1.201	MAGE-jump-01	jump.etiantian.org

提示：表15-5中所有IP的子网掩码均为24位，即255.255.255.0。

表15-6　集群节点VIP地址及对应域名解析规划

服务器 VIP 说明	VIP	对应域名解析
反向代理服务 VIP	10.0.0.3	www.etiantian.org blog.etiantian.org bbs.etiantian.org 企业场景：外网 IP 通过 DNS 解析
MySQL 主数据库 VIP	172.16.1.52	dbmaster_vip.etiantian.org
NFS 存储服务 VIP	172.16.1.32	nfs_vip.etiantian.org
Memcached 缓存服务 VIP	172.16.1.63	mc_vip.etiantian.org

2.IP地址及主机名规划思路说明

根据网站整体集群架构图所示的功能，本章将该架构分为5个功能区域，并结合项目的整体规划，简化了IP地址的规划，即采用两个C类地址段，一个C类地址段（10.0.0.0/24）作为外网用户所使用IP地址段；另一个C类地址段（172.16.1.0/24）作为内网用户所使用IP地址段。5个功能区域IP地址段具体的规划如下。

（1）前端负载均衡代理区域

采用的外网地址范围为10.0.0.1~10.0.0.3，内网地址范围为172.16.1.1~172.16.1.9。

根据架构图可知，要有两台设备作为反向代理服务器，因此，可规划10.0.0.1、10.0.0.2为两台服务器外网eth0网卡的实际地址，10.0.0.3为虚拟地址VIP，将来网站域名就需要解析到这个VIP上，内网地址172.16.1.1、172.16.1.2分别作为两台服务器的实际内网eth1网卡地址。

（2）Web服务应用区域

为Web服务器的内网eth1网卡预留的地址范围为172.16.1.10~172.16.1.29，根据架构图，当前只有两台设备作为Web服务器，因此，规划172.16.1.10、172.16.1.11为两台Web服务器内网eth1的实际地址，由于采用了反向代理的模式，因此，Web服务器无需外网IP地址。

（3）存储服务器应用区域

为存储服务器的内网eth1网卡预留的地址范围为172.16.1.30~172.16.1.49，根据架构图，有两台设备作为存储服务器，其中一台为NFS服务器，规划172.16.1.30作为NFS服务器的IP；另一台为rsync备份服务器，规划172.16.1.31作为rsync备份服务器IP。

为保证NFS服务的高可用性，在NFS存储服务器与rsync备份服务器之间架设Keepalived高可用服务，设置172.16.1.32作为虚拟地址，即VIP地址。

（4）数据库及缓存服务器应用区域

为MySQL服务器的内网eth1网卡预留的地址范围为172.16.1.50~172.16.1.59，目前有三台MySQL服务器，分别作为MySQL主数据库服务器、主数据库热备服务器、从数据库服务器，规划的地址

分别为172.16.1.50、172.16.1.51、172.16.1.52。

为保证MySQL主数据库服务的高可用性，在MySQL主服务器之间架设了Keepalived服务，采用172.16.1.53作为主数据库虚拟VIP地址，如果有多台从库还可以为多台从库配置LVS负载均衡，使用额外的地址作为读数据库的VIP地址，此内容留给读者去拓展。

为Memcached缓存服务器的内网eth1网卡预留的地址范围为172.16.1.60~172.16.1.69，目前有2台服务器作为数据库的前端分布式缓存服务集群（分布式调度算法使用一致性哈希算法），在每个服务器上均配置2个实例分别作为缓存服务和session会话共享服务。

（5）管理服务器应用区域

内外IP地址在200以后的均作为监控管理服务器的地址。目前架构了两台服务器作为管理服务器，监控管理服务器内网eth1地址为172.16.1.200，无外网地址，配置的服务为Nagios监控服务和批量管理服务（ssh key或saltstack），跳板机服务器外网eth0地址为10.0.0.201，内网eth1地址为172.16.1.201，配置的服务为跳板机及VPN服务。


提示：所有服务器的地址都是由小到大进行规划的，有利于避免地址空余的浪费，内外网地址最后8位尽量相同，有利于运维人员对IP的管理，并且可根据功能区域进行地址规划，便于新增和对现有设备的管理。以上规划和设计是以老男孩工作环境为主的，读者在实际部署

集群时可以参考上述设计规划，但在企业工作场景下，外网eth0对应的10.0.0.0/24地址段应该用真正的外部IP地址段替代。

3.主机名规划说明

根据业务功能对所有集群节点的主机名进行了详细规划，并采用英文缩写方式命名，采用的命名格式见表15-7。

表15-7　主机名规划命名参考格式

功能区域名称	服务器应用名称	应用功能或角色名称	服务器编码
DB	MySQL	master	01
WEB	LNMP	node	01


说明：在表15-7中，各字段名称之间用"-"进行连接。如果觉得名称太长，可以自行缩短。

下面分别来看看各名称的含义。

·功能区域名称：目前架构分为5个功能区域，分别命名为LB（负载）、WEB（网页访问）、STORE（存储）、DB（数据库）及Cache（缓存）、MAGE（管理）。

·服务器应用名称：根据服务器所装的核心应用服务命名。例如：Web服务器安装Nginx、PHP、MySQL软件一体的环境，可命名为LNMP。

·应用功能或角色名称：根据服务器的业务不同，会出现相同的应用软件实现不同的功能的情况，并且相同的功能软件，也会划分主从等不同角色等，因此也就有了根据功能或角色命名的方式，例如：MySQL Master数据库。

·服务器编号：根据相同应用功能的服务器的排序或分组进行命名。例如，两台Nginx服务器的命名规范为：WEB-LNMP-01、WEB-LNMP-02（node为通用节点名，可省略）。


提示：上述仅仅是单机房简单的规划思路，如果是跨机房，分组还得规划出来地区以及机房的标识。例如：阿里集团的做法为主机名的命名规范就是“appname+IP地址+环境标识+机房标识”，但是自动化出来的机器名大多只做到了“appname+IP地址+机房标识”。

4.架构hosts文件的规划

hosts文件解析的内容采用服务器名称结合域名的方式进行命名，并与相对应的IP地址进行解析，具体内容如下。

```
[root@STORE-Rsync-01 base]# cat /etc/hosts
127.0.0.1          localhost localhost.localdomain localhost4 localhost4.localdomain



1          localhost localhost.localdomain localhost6 localhost6.localdomain6
172.16.1.1              LB-nginx-01          lb01.etiantian.org
172.16.1.2              LB-nginx-02          lb02.etiantian.org
172.16.1.10          WEB-LNMP-01          web01.etiantian.org
172.16.1.11          WEB-LNMP-02          web02.etiantian.org
172.16.1.30          STORE-NFS-01          nfs.etiantian.org
172.16.1.31          STORE-Rsync-01          rsync.etiantian.org
172.16.1.32          nfs_vip.etiantian.org
```

```
172.16.1.50          MySQL-master-01     dbmaster01.etiantian.org
172.16.1.51          MySQL-master-02     dbmaster02.etiantian.org
172.16.1.52          dbmaster_vip.etiantian.org
172.16.1.53          MySQL-slave-01          dbslave01.etiantian.org
172.16.1.61          Cache-Mc-01          cache01.etiantian.org
172.16.1.62          Cache-Mc-02          cache02.etiantian.org
172.16.1.63          mc_vip.etiantian.org
172.16.1.200          MAGE-monitor-01 monitor.etiantian.org
172.16.1.201          MAGE-jump-01          jump.etiantian.org
10.0.0.3               www.etiantian.org blog.etiantian.org bbs.etiantian.org
```

15.1.4 集群节点网络服务规划

表15-8是各集群节点对应的网络服务规划表。

表15-8 网站集群节点服务规划表

服务器说明	安装服务说明
rsync 备份服务器	1）安装 rsync 服务软件，提供网络数据远程备份功能，设定备份目录为 /backup； 2）安装 Keepalived 服务软件，为 NFS 服务宕机提供热备切换功能； 3）开发检查所有服务器数据备份完整性的脚本，并每天早晨 8 点发邮件给管理员
NFS 存储服务器	1）安装 NFS 服务软件，提供网络文件共享功能，设定共享目录为 /data； 2）安装 Keepalived 服务软件，为 NFS 服务提供热备切换功能； 3）安装 inotify 服务软件，实时同步 NFS 服务资源到 rsync 备份服务器； 4）开发通用脚本，并通过定时任务把 NFS 本地基础配置数据备份到 rsync 备份服务器
MySQL 主数据库	1）安装 MySQL 服务软件，提供主数据库服务功能； 2）安装 Keepalived 服务软件，为 MySQL 服务提供热备切换功能； 3）配置 MySQL 主数据库和 MySQL 主热备数据库互为主从复制（采用 MySQL 自身复制功能）； 4）开发通用脚本，并通过定时任务把基础配置数据备份到 rsync 备份服务器
MySQL 主热备数据库	1）安装 MySQL 服务软件，提供主数据库服务功能； 2）安装 Keepalived 服务软件，为 MySQL 服务提供热备切换功能； 3）配置 MySQL 主热备数据库和 MySQL 主数据库互为主从复制的功能，主库宕机时接管主库提供服务； 4）开发通用脚本，并通过定时任务把基础配置数据备份到 rsync 备份服务器
MySQL 从数据库	1）安装 MySQL 服务软件，提供从数据库功能，用于读数据； 2）配置 MySQL 多实例（总共 5 个实例），工作中一台服务器可配置 2~4 个实例； 3）配置 LVS 服务，以便对前三个从数据库实例做负载均衡，从而提供读服务。理想情况这个 LVS 服务的主备最好配置到两台前端反向代理服务器上，并通过 Keepalived 服务实现高可用切换； 4）人为对多个 MySQL 实例分工，第 5 个实例作为数据库备份使用； 5）开发通用脚本，并通过定时任务把数据库全量备份及增量数据定时备份到 rsync 备份服务器； 6）开发通用脚本，并通过定时任务把基础配置数据备份到 rsync 备份服务器

（续）

服务器说明	安装服务说明
Memcached 缓存及 session 共享 1	1）安装 Memcached 服务软件； 2）通过不同的端口启用 2 个实例，分别用于数据库缓存及 session 会话共享； 3）通过程序实现一致性哈希算法调度访问 Memcached 缓存服务集群； 4）如果需要集群可采用 MemcacheDB 或 Redis
Memcached 缓存及 session 共享 2	1）安装 Memcached 服务软件； 2）通过不同的端口启用 2 个实例，分别用于数据库缓存及 session 会话共享； 3）通过程序实现一致性哈希算法调度访问 Memcached 缓存服务集群； 4）如果需要集群可采用 MemcacheDB 或 Redis
nginx web01	1）安装 Nginx 服务软件，作为 Web 服务器使用； 2）安装 PHP 服务软件，作为动态 Web 服务器使用，负责处理 PHP 动态页面请求； 3）部署 bbs、cms、blog 三个开源 PHP 网站产品，模拟搭建类似 51CTO 的 IT 资讯平台部分内容； 4）开发通用脚本，并通过定时任务把访问日志、网站程序备份到 rsync 备份服务器； 5）开发通用脚本，并通过定时任务把基础配置数据备份到 rsync 备份服务器； 6）把网站产品的所有站点内用户上传的资源目录挂载到后端的 NFS 存储服务器对应的目录上，从而让用户上传的资源文件直接放到 NFS 共享服务器上； 7）安装 amoeba 服务软件（测试可以用 amoeba，工作中多为开发改造程序实现读写分离），提供 Web 程序对数据库的读写分离功能； 8）配置 PHP 的配置文件 php.ini，将所有集群节点的 session 会话数据都保存到同一个 Memcached 服务中
nginx web02	所有配置项完全同 nginx web01
反向代理及负载均衡服务 01	1）安装 Nginx 服务软件，为后端的 Web 服务提供负载及反向代理功能，可以配置成和反向代理服务 02 互为备份的形式，只不过各自为不同的业务提供对外服务； 2）安装 Keepalived 服务软件，为两台负载及反向代理服务器提供高可用功能； 3）配置通用备份脚本，并通过定时任务把基础重要数据备份到 rsync 备份服务器
反向代理及负载均衡服务 02	1）安装 Nginx 服务软件，为后端的 Web 服务提供负载及反向代理功能，可以配置成和反向代理服务 01 互为备份的形式，只不过各自为不同的业务提供对外服务； 2）安装 Keepalived 服务软件，为两台负载及反向代理服务器提供高可用功能； 3）配置通用备份脚本，并通过定时任务把基础重要数据备份到 rsync 备份服务器
监控管理服务	安装 Nagios 服务，监控所有服务器及服务器上的服务，并实现画趋势图及邮件短信报警
跳板机管理服务	1）配置 VPN 服务，作为网站集群远程管理的入口； 2）对跳板机 SSH 监听本地内网卡 IP，只要不是通过 VPN 拨号的一律禁止采用 SSH 登录服务器，加强服务器安全； 3）配置跳板机脚本，实现跳板机管理功能，所有人员通过跳板机管理其他服务器，所有操作都会被审计记录，跳板机和审计的功能可以通过开源软件实现，或者自己利用 Shell 或 Python 开发； 4）所有集群服务器不提供密码验证登录，只能从跳板机通过密钥登录

15.1.5　集群节点服务应用的目录结构规划

表15-9是集群节点的关键目录结构规划，需要注意的是，这里给出的并非项目全部目录信息，而只是关键目录规划信息。

表15-9　集群节点关键目录结构规划

服务器说明	目录结构	目录结构说明
通用初始化目录结构	/server/scripts /server/tools /application /backup	• 存放脚本的目录 • 存放编译软件源码目录 • 软件程序的安装目录 • 服务器本地数据临时备份目录
web 服务器	/data0/www/ /app/logs/	• Web 服务站点目录，下面包含 www、bbs、blog 目录 • Web 服务的访问日志存放目录
rsync 备份服务器	/backup/upload/ /backup	• NFS 服务器实时无差异同步数据目录 • 所有数据备份目录，不同服务器上传的备份数据目录以 IP 地址或该服务器的主机名命名
NFS 存储服务器	/data	• NFS 服务器共享存储大目录

15.2　集群服务搭建详细规划设计说明

15.2.1　集群服务搭建最佳部署顺序

根据老男孩的经验，整个集群架构的部署最好从后往前进行，即从集群的后端备份、存储、数据库开始部署，部署的最佳顺序见表15-10。

表15-10　网站集群服务最佳部署顺序

安装最佳排序	服务器
01	跳板机管理服务（自动化安装先装，非自动化无所谓）
02	监控及批量管理服务（批量安装，加入监控）
03	rsync 存储及备份服务
04	NFS 存储服务
05	MySQL 主数据库
06	MySQL 主热备数据库
07	MySQL 从数据库
08	Nginx web01
09	Nginx web02
10	反向代理服务 01
11	反向代理服务 02

说明：本章不会带领读者从0开始搭建整个集群架构，因为对于服务的部署及优化，在前面的章节都已经详细讲解过了，限于篇幅，本章只会为读者规划一个完整搭建企业生产场景标准的集群架构思路，具体的搭建部署过程将作为综合作业留给读者自行完成。做任何事情都

需要规划，规划好了，实施起来就会顺风顺水。因此，老男孩也希望读者先看懂本章的规划思路，然后自行设计架构方案，再去搭建，这样效率会更高，收获会更大。

15.2.2　集群架构服务搭建规划设计

1.集群所有服务器节点的基础优化设置

首先要按照前文的规划对所有服务器设置好IP地址，然后通过SSH远程登录把如下初始化文件和脚本文件拷贝到每一台集群节点系统中的对应位置上，并执行optimization.sh脚本对服务器进行初始化设置。

```
[root@oldboy scripts]# pwd
/server/scripts
[root@STORE-Rsync-01 base]# cat/etc/hosts
127.0.0.1          localhost localhost.localdomain localhost4 localhost4.localdomain



1          localhost localhost.localdomain localhost6 localhost6.localdomain6
172.16.1.1          LB-nginx-01          lb01.etiantian.org
172.16.1.2          LB-nginx-02          lb02.etiantian.org
172.16.1.10     WEB-LNMP-01          web01.etiantian.org
172.16.1.11     WEB-LNMP-02          web02.etiantian.org
172.16.1.30     STORE-NFS-01          nfs.etiantian.org
172.16.1.31     STORE-Rsync-01          rsync.etiantian.org
172.16.1.32     nfs_vip.etiantian.org
172.16.1.50     MySQL-master-01     dbmaster01.etiantian.org
172.16.1.51     MySQL-master-02     dbmaster02.etiantian.org
172.16.1.52     dbmaster_vip.etiantian.org
172.16.1.53     MySQL-slave-01          dbslave01.etiantian.org
172.16.1.61     Cache-Mc-01          cache01.etiantian.org
172.16.1.62     Cache-Mc-02          cache02.etiantian.org
172.16.1.63     mc_vip.etiantian.org
172.16.1.200     MAGE-monitor-01 monitor.etiantian.org
172.16.1.201     MAGE-jump-01          jump.etiantian.org
10.0.0.3          www.etiantian.org blog.etiantian.org bbs.etiantian.org
```

hosts文件为规划好的所有服务器的统一模板，每新增服务器就要增加一条对应主机名和IP的hosts解析，并且要全部重新分发到所有机器上（通过SSH及saltstack等软件可实现），这里的hosts文件会被系统初始

化优化脚本optimization.sh调用。

下面的sysctl.conf文件为设置优化好的内核文件模板，此文件也会被优化脚本optimization.sh调用。

```
[root@oldboy scripts]# cat/etc/sysctl.conf
# Kernel sysctl configuration file for Red Hat Linux
#
# For binary values,


 0 is disabled,


 1 is enabled.  See sysctl (


8)


 and
# sysctl.conf (


5)


 for more details.
# Controls IP packet forwarding
net.ipv4.ip_forward = 0
# Controls source route verification
net.ipv4.conf.default.rp_filter = 1
# Do not accept source routing
net.ipv4.conf.default.accept_source_route = 0
# Controls the System Request debugging functionality of the kernel
kernel.sysrq = 0
# Controls whether core dumps will append the PID to the core filename.
# Useful for debugging multi-threaded applications.
kernel.core_uses_pid = 1
# Controls the use of TCP syncookies
net.ipv4.tcp_syncookies = 1
# Disable netfilter on bridges.
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
# Controls the default maxmimum size of a mesage queue
kernel.msgmnb = 65536
# Controls the maximum size of a message,
```

```
 in bytes
kernel.msgmax = 65536
# Controls the maximum shared segment size,



 in bytes
kernel.shmmax = 68719476736
# Controls the maximum number of shared memory segments,



 in pages
kernel.shmall = 4294967296
############by oldboy#########################################
net.ipv4.tcp_fin_timeout = 2
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_keepalive_time = 600
net.ipv4.ip_local_port_range = 4000    65000
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.route.gc_timeout = 100
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_synack_retries = 1
net.core.somaxconn = 16384
net.core.netdev_max_backlog = 16384
net.ipv4.tcp_max_orphans = 16384
#以下参数是对



iptables防火墙的优化，防火墙不开会提示，可以忽略不理



net.nf_conntrack_max = 25000000
net.netfilter.nf_conntrack_max = 25000000
net.netfilter.nf_conntrack_tcp_timeout_established = 180
net.netfilter.nf_conntrack_tcp_timeout_time_wait = 120
net.netfilter.nf_conntrack_tcp_timeout_close_wait = 60
net.netfilter.nf_conntrack_tcp_timeout_fin_wait = 120
net.core.wmem_default = 8388608
net.core.rmem_default = 8388608
net.core.wmem_max = 16777216
net.core.rmem_max = 16777216
```

下面的optimization.sh为Linux系统优化基础脚本，需要在每一台机器上执行该脚本配置优化系统。

```
[root@oldboy scripts]# cat optimization.sh
#!



/bin/bash
#############################################################
# File Name   :



 linux system config
# description :



 This script is used to set linux system
# Author      :



 oldboy
# Mail        :



 service@oldboyedu.com
#############################################################
. /etc/init.d/functions
IP=`/sbin/ifconfig|awk -F '[ :



]+' 'NR==2{print $4}'`
# Defined result function
function Msg ()



{
        if [ $ -eq 0 ];



then
            action "$1" /bin/true
        else
            action "$1" /bin/false
        fi
}
# Defined Close selinux Functions
function selinux ()



{
        [ -f "/etc/selinux/config" ] && {
            sed -i 's#SELINUX=enforcing#SELINUX=disabled#g' /etc/selinux/config
            setenforce 0
            Msg "Close selinux"
        }
```

```
}
# Defined add Ordinary users Functions
function AddUser ()



{
        id oldboy &>/dev/null
        if [ $ -ne 0 ];



then
        useradd oldboy &>/dev/null
        echo "123456"|passwd --stdin oldboy &>/dev/null &&\
        sed -ir '98a oldboy    ALL= (



ALL)



       NOPASSWD:



ALL'  /etc/sudoers &&\
        visudo -c &>/dev/null
        Msg "AddUser oldboy"
        else
            echo "oldboy user is exist."
        fi
}
# Defined Hide the system version number Functions
function HideVersion ()



{
        [ -f "/etc/issue" ] && >/etc/issue
        [ -f "/etc/issue.net" ] && > /etc/issue.net
        Msg "Hide sys info."
}
# Defined SSHD config Functions
function sshd ()



{
    sshd_file=/etc/ssh/sshd_config
    if [ `grep "52113" $sshd_file|wc -l` -eq 0 ];



then
    sed -ir "13 iPort 52113\nPermitRootLogin no\nPermitEmptyPasswords no\nUseDNS no\
    sed -i 's@#ListenAddress 0.0.0.0@ListenAddress '${IP}':
```

```
52113@g' $sshd_file
    /etc/init.d/sshd restart >/dev/null 2>&1
    Msg "sshd config"
    fi
}
# Defined OPEN FILES Functions
function openfiles ()



{
        if [ `grep "nofile  65535" /etc/security/limits.conf|wc -l` -eq 0 ];



then
            echo '*  -  nofile  65535' >> /etc/security/limits.conf
            ulimit -SHn 65535
            Msg "open files"
        fi
}
function hosts ()



{
        if [ !



 -f /server/scripts/hosts ];



then
            echo "/server/scripts/hosts is not exist,



pls solve this question"
            sleep 300
            exit 1
        fi
        /bin/cp /server/scripts/hosts /etc/hosts
}
# Defined System Startup Services Functions
function boot ()



{
        export LANG=en
        for oldboy in `chkconfig --list|grep "3:



on"|awk '{print $1}'|grep -vE "crond|network|rsyslog|sshd|sysstat"`
            do
            chkconfig $oldboy off
        done
        Msg "BOOT config"
```

```
}
# Defined Time Synchronization Functions
function Time ()



{
        grep "time.nist.gov" /var/spool/cron/root > /dev/null 2>&1
        if [ $ -ne 0 ];



then
        echo "#time sync by oldboy at $ (



date +%F)



" >>/var/spool/cron/root
        echo '*/5 * * * * /usr/sbin/ntpdate time.nist.gov &>/dev/null' >>/var/spool/
        fi
        Msg "Time Synchronization"
}
# Defined Kernel parameters Functions
function kernel ()



{
    /bin/cp /etc/sysctl.conf /etc/sysctl.conf.$RANDOM
    /bin/cp /server/scripts/sysctl.conf /etc/
    Msg "kernel"
}
function iptables ()



{
    /etc/init.d/iptables stop
    /etc/init.d/iptables stop
    Msg "iptables"
}
function hostname ()



{
    ip=`/sbin/ifconfig eth1|awk -F "[ :



]+" 'NR==2{print $4}'`
    name=`grep -w "$ip" /etc/hosts |awk '{print $2}'`
    sed -i 's/HOSTNAME=*/HOSTNAME='"$name"'/g' /etc/sysconfig/network
    /bin/hostname $name
    Msg "hostname"
}
# Defined main Functions
```

```
function main ()


{
        AddUser
        HideVersion
        sshd
        openfiles
        hosts
        boot
        Time
        kernel
        iptables
        hostname
}
main
```

执行时的输出信息如图15-2所示。

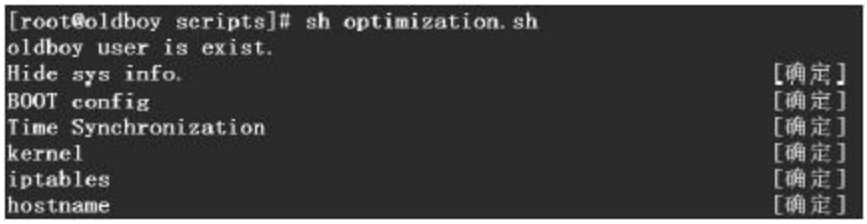


图15-2　优化系统脚本的执行后输出信息

执行后的具体检查结果就不多描述了，读者也可以写一个检查的脚本。关于优化部分内容请参考第3章“CentOS 6.6连接管理及优化”。

下面即将介绍的backup_to_rsync.sh脚本是所有服务器通用的重要数据备份脚本，此脚本需要rsync服务的支持，因此，需要先部署完rsync备份服务器，然后才可正确运行，通过执行设定脚本set_backup_script.sh即可将backup_to_rsync.sh设置为定时任务，即每天晚上0点执行backup_to_rsync.sh，实现所有节点异地数据的集中备份，备份服务器为rsync备份服务器。

在介绍backup_to_rsync.sh脚本之前，先介绍一下backuplist，它为要备份的数据文件名称或目录，需要以全路径形式列出来，可以是文件或目录，读者可以根据业务服务器的需要在文件内添加需要备份的数据。这里列举要备份的基本内容，如下：

```
[root@oldboy scripts]# cat /server/scripts/backuplist
/server/scripts
/etc
/var/spool/cron
```

backup_to_rsync.sh为通用常规数据备份脚本，功能是先在本地打包备份数据并记录数据包文件的md5值，然后推送到rsync备份服务器上，最后在本地留存7天内的数据副本。

```
[root@oldboy scripts]# cat backup_to_rsync.sh
#!



/bin/bash
# Source function library
. /etc/init.d/functions
rsync_host=rsync.etiantian.org
# Defined variables
IP=$ (



ifconfig eth1|awk -F '[ :



]+' 'NR==2 {print $4}')



Path="/backup/$IP"
TIME=`/bin/date +%F`
BackupFile=/server/scripts/backuplist
# Judged the existence of variables
[ !
```

```
 -d $Path ] && mkdir -p $Path
[ !



 -f $BackupFile ] && {
    echo "Please give me $BackupFile"
    exit 1
}
# Defined result function
function Msg ()



{
        if [ $ -eq 0 ];



then
            action "$*" /bin/true
        else
            action "$*" /bin/false
        fi
}
# Backup config files
tar zcfh $Path/conf_${TIME}.tar.gz `cat $BackupFile` &>/dev/null
Msg 'Backup config files'
# Make a flag for backup
find $Path -type f -name "*${TIME}.tar.gz"|xargs md5sum >$Path/flag_$TIME 2>/dev/nul
Msg 'Make a flag for backup'
# Send backup to backup server
rsync -az $Path rsync_backup@${rsync_host}: :



backup --password-file=/etc/rsync.password &>/dev/null
Msg 'Send backup to backup server'
# Delete backup a week ago
find ${Path-/tmp} -type f -name "*.tar.gz" -mtime +7|xargs rm -f &>/dev/null
Msg 'Delete backup a week ago'
```

通过set_backup_script.sh脚本可实现为备份数据脚本设置为系统定时任务，每天0点备份推送数据，并清除7天前的数据副本。

```
[root@oldboy scripts]# cat set_backup_script.sh
#!



/bin/sh
function crond_backup ()
```

```
{
        [ `crontab -l|grep "backup data"|wc -l` -eq 0 ] &&{
        echo -e "#backup data\n00 00 * * * /bin/sh /server/scripts/backup_to_rsync.s
        crontab -l
        sleep 2
        }||{
            echo "backup cron is exist,


no config."
        }
}
crond_backup
```

2.监控管理服务器的安装部署规划

（1）监控管理服务器信息说明

监控管理服务器在集群中的作用非常关键，它可用来监控所有集群服务器节点的工作是否正常，因此这个服务器安排到集群节点第一或仅次于rsync备份服务的位置部署，详细说明如下。

主机名及IP为：MAGE-monitor-01（eth1：172.16.1.200）。监控管理服务器在集群中非常关键，监控的内容除了确定所有集群服务器节点工作是否正常以外，还包括服务器是否宕机、各类服务是否异常（例如：Web、数据库、存储等），因此，优先部署此服务对整个集群服务的部署很有利，这个监控服务器相当于军队里哨兵的角色。

表15-11为监控管理服务器的Nagios信息规划明细。

表15-11　监控管理服务器Nagios信息规划

服务器说明	外网 IP eth0	内部 IP eth1	名称规划	域名规划
监控管理服务器 MAGE-monitor-01	无	172.16.1.200/24	MAGE-monitor-01	monitor.etiantian.org

（2）监控管理服务器部署说明

有关监控管理服务器部署过程请参考第14章“企业级监控Nagios服务应用”，部署完基础监控平台后，以后每新增一个服务器，都要部署一个nrpe客户端到新增服务器上，并在监控服务器上添加对应的服务器及服务监控，nrpe等客户端也可随优化脚本自动安装好。监控管理服务器的逻辑图如图15-3所示。

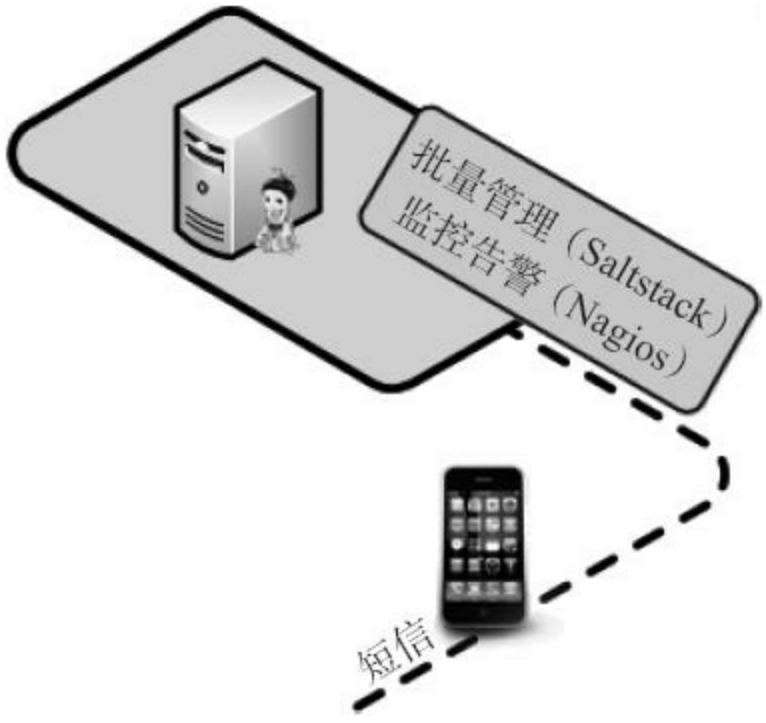


图15-3　监控管理服务器架构逻辑图

（3）兼任批量部署、管理服务

由于机器有限，可以让监控服务器同时兼任Saltstack批量部署和管理的服务器，如果机器数量少也可以使用SSH key的方案，此内容的学习和搭建，请参考老男孩的其他相关课程。

3.rsync备份服务器的安装部署规划

（1）rsync备份服务器信息说明

rsync备份服务器在集群中的作用非常关键，它是用来存放所有集群服务器节点的重要数据备份的，这个服务器安排到第二位来部署，详细说明如下。

主机名及IP为：STORE-Rsync-01（eth1：172.16.1.31），rsync服务器主要是用作备份服务器，用来存储所有集群服务器节点的重要文件，例如：/etc/rc.local[开机自启动文件]、/data0[web站点目录]、/app/logs[web日志目录]、/server/scripts[脚本目录]、/var/spool/cron[定时任务目录]等数据的备份；并与NFS存储服务器实现文件、图片等资源异机实时复制，及NFS服务故障的接管等，表15-12为rsync备份服务器信息规划。

表15-12　rsync备份服务器信息规划

服务器说明	外网 IP eth0	内部 IP eth1	名称规划	域名规划
rsync 备份服务器	无	172.16.1.31/24	STORE-Rsync-01	rsync.etiantian.org

下面是该rsync备份服务器部署的主要应用：

·安装及配置rsync服务，见脚本rsync.sh。

·开发rsync服务启动脚本，见脚本rsyncd。

·检查rsync服务器上的备份数据是否完整并将结果通过定时任务，于每天早晨8点发送至管理员信箱，见脚本check_bak.sh。

·安装Keepalived软件，具体方法请参考第12章“Keepalived高可用集群应用实践”。

（2）部署的主要应用说明

安装配置rsync服务，操作过程见rsync.sh脚本，如下：

```
[root@STORE-Rsync-01 rsync]# cat/server/scripts/rsync.sh
#!


/bin/bash
#############################################################################
# File Name     :


 rsync.sh
# description     :


 install and config rsync server
# Author          :


 oldboy
# Mail          :


 service@oldboyedu.com
#############################################################################
[ -f /etc/init.d/functions ] && . /etc/init.d/functions
```

```
function Msg ()



{
    if [ $ -eq 0 ];



then
    action "$*" /bin/true
    else
    action "$*" /bin/false
    fi
}
function rsyncd_conf ()



{
        cat>/etc/rsyncd.conf<<-EOF
        sync server
        #created by oldboy 15:



01 2015-6-18
        ##rsyncd.conf start##
        uid = rsync
        gid = rsync
        use chroot = no
        max connections = 2000
        timeout = 600
        pid file = /var/run/rsyncd.pid
        lock file = /var/run/rsync.lock
        log file = /var/log/rsyncd.log
        ignore errors
        read only = false
        list = false
        hosts allow = 172.16.1.0/24
        hosts deny = 0.0.0.0/24
        auth users = rsync_backup
        secrets file = /etc/rsync.password
        #####################################
        [backup]
        comment = back server by oldboy 14:



18 2015-6-18
        path = /backup
        EOF
        Msg "rsyncd_conf"
}
function rsync ()



{
        useradd -u 666 rsync -M -s /sbin/nologin
        [ !
```

```
 -d /backup ] && mkdir -p /backup
        [ !



 -d /server/tools ] && mkdir -p /server/tools
        [ !



 -d /server/seripts ] && mkdir -p /server/scripts
        chown -R rsync.rsync /backup
        echo "rsync_backup:



oldboy123" > /etc/rsync.password &&\
        chmod 600 /etc/rsync.password
        #判断



rsync是否启动




        [ !



 -f /server/scripts/rsyncd ] && {
        echo "/server/scripts/rsyncd is not exist,



error."
        exit 2
        }
        \cp /server/scripts/rsyncd /etc/init.d/
        chmod 755 /etc/init.d/rsyncd
        netstat -nultp |grep "873"
        if [ $ -eq 0 ];




        then
                echo "rsync is running" >/dev/null
        else
                /etc/init.d/rsyncd restart
        fi
        /sbin/chkconfig --add rsyncd
        /sbin/chkconfig rsyncd on
        Msg "rsync"
        sleep 1
}
###set mail###
```

```
function postfix ()



{
        if [ `grep "oldboyedu" /etc/mail.rc|wc -l` -eq 0 ];



then
        /bin/cp /etc/mail.rc /etc/mail.rc.$RANDOM
        echo -e "set from=oldboyedu@163.com smtp=smtp.163.com\nset smtp-auth-user=ol
        /etc/init.d/postfix restart
        else
            echo "postfix is not need to config it."
        fi
        sleep 1
}
function crond_check ()



{
        [ `crontab -l|grep "check.sh"|wc -l` -eq 0 ] &&{
        echo -e "#check backup\n00 08 * * * /bin/sh /server/scripts/check.sh >/dev/r
        crontab -l
        sleep 2
        }||{
            echo "check.sh cron is exist,



no config."
        }
}
function main ()



{
        rsyncd_conf
        rsync
        postfix
        crond_check
}
main
```

开发rsync服务的启动脚本，以下为rsyncd脚本内容。

```
[root@oldboy rsync]# cat rsyncd
#!



/bin/bash
#start the rsync
```

```
#chkconfig:



 2345 56 24
#description:



 This is about rysnc restart shell
. /etc/init.d/functions
RETVAL=0
prog="rsync"
#check if rsync is already running
start ()



{
        if [ !



 -f /var/run/rsyncd.pid ];



then
            /usr/bin/rsync --daemon --config=/etc/rsyncd.conf
        fi
        RETVAL=$
        sleep 1
        if [ $RETVAL -eq 0 ];



then
                action "starting $prog ......" /bin/true
        else
                action "starting $prog ......" /bin/false
        fi
        return $RETVAL
}
stop ()



{
        killproc /usr/sbin/rsync
        RETVAL=$
        sleep 1
        if [ $RETVAL -eq 0 ];



then
            rm -f /var/run/rsyncd.pid
            action "stopped $prog ......" /bin/true
        else
            action "stopped $prog ......" /bin/false
        fi
```

```
}
restart ()



{
        $0 stop
        $0 start
}
case "$1" in
start)



        start; ;



stop)



        stop; ;



restart|reload)



        restart; ;



*)



        echo "Usage:


$0 {start|stop|restart}"
        exit 1
esac
```

检查备份数据是否完整，并将结果通过定时任务于每天早晨8点发送管理员信箱，具体见脚本check_bak.sh，如下。

```
#!


/bin/bash
DIR=/backup
TIME=`/bin/date +%F`
log=/tmp/$TIME-check.log
[ -d $DIR ] &&{
    find $DIR -type f -name "flag_$TIME"|xargs md5sum -c >$log  2>/dev/null
    mail -s "$(


date +%F_%T)


 backup check result" 12345678@qq.com <$log
}
```

提示：

1）邮箱服务器的配置见rsync.sh脚本内容。

2）利用指纹比对文件时，要求节点服务器和备份服务器的路径相同。

安装Keepalived高可用服务软件，可在NFS存储服务宕机时，让rsync热备服务器及时接管NFS存储服务，Keepalived高可用服务内容请参考第12章“Keepalived高可用集群应用实践”。

部署实现后的rsync备份服务器的逻辑图如图15-4所示。

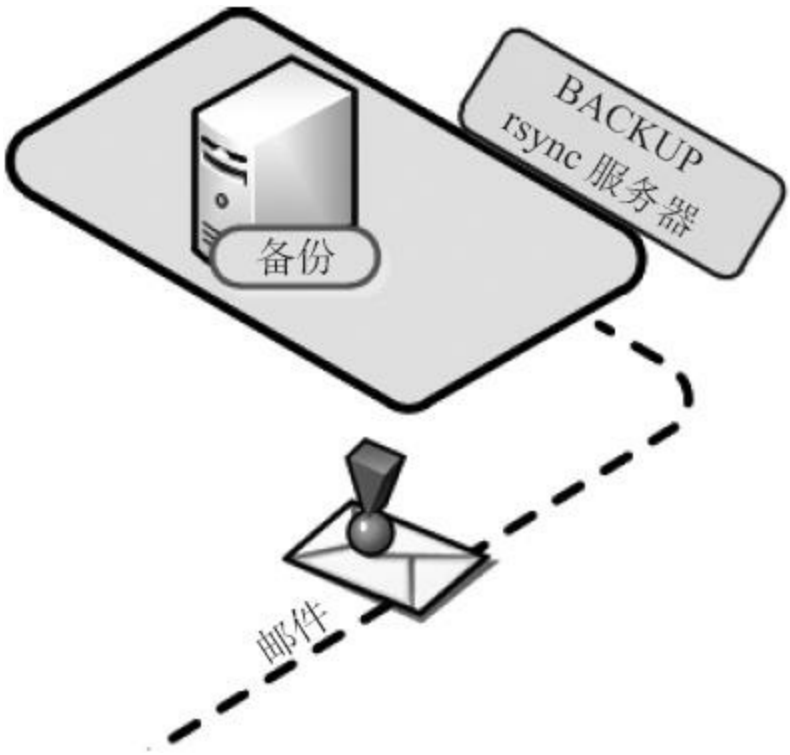


图15-4　rsync备份服务器逻辑图

4.NFS服务器安装部署规划设计

（1）NFS服务器相关说明

NFS服务被用作所有相同业务网站集群服务节点的后端共享存储，例如：图片、附件、头像、视频等的共享存储，由于NFS服务为单点，因此，要通过inotify（也可以通过drbd或程序双写等实时同步方案备份NFS数据）实现把NFS数据实时同步到rsync备份服务器，并且配置Keepalived服务，实现NFS服务的高可用宕机接管。

以下是NFS服务器（STORE-NFS-01（VIP：172.16.1.32）（eth1：172.16.1.30））在集群中的作用说明：NFS是整个集群服务的后端数据

共享服务器，主要用于集群节点的所有Web服务器的挂载，以及图片、附件等资源的存放，并通过inotify把NFS数据实时同步到rsync备份服务器，此外，它还配置了Keepalived服务，来实现NFS服务的高可用宕机接管。表15-13为NFS服务器信息规划。

表15-13　NFS服务器信息规划

服务器说明	外网 IP eth0	内部 IP eth1	名称规划	域名规划
NFS 存储服务	无	172.16.1.30/24	STORE-NFS-01	nfs.etiantian.org

NFS服务器安装部署的主要应用如下：

·安装了NFS共享存储服务。

·安装了Keepalived高可用服务。

·安装了inotify实时监控，并编写了脚本启动inotify服务，见inotify.sh脚本。

·对基础重要数据定时备份（见“集群所有服务器系统的基础优化”相关内容）。

（2）安装的应用说明

·安装的NFS共享存储服务，用于存放所有集群Web服务器上传的图片、附件、头像等。此部分内容请参考第10章“企业级NFS网络文件共享服务”。

·安装的Keepalived高可用服务，用于NFS服务宕机时，可让rsync服务器热备NFS服务接管NFS服务，此部分内容请参考第12章“Keepalived高可用集群应用实践”，NFS服务器为MASTER，VIP为172.16.1.32，前端所有集群Web服务器节点都挂载到NFS的这个VIP地址上。

install_inotify.sh为安装inotify实时监控服务的脚本，如下。

```
[root@STORE-NFS-01 nfs]# cat install_inotify.sh
#!



/bin/bash
[ !



 -f /etc/yum.repos.d/epel.repo ] &&\
wget -O /etc/yum.repos.d/epel.repo http:



// mirrors.aliyun.com/repo/epel-6.repo
if [ $ -eq 0 -a `rpm -qa inotify-tools|wc -l` -ne 1 ];



then
    yum -y install inotify-tools
fi
```

inotifyd.sh实时监控服务的脚本，如下。

```
[root@STORE-Rsync-01 nfs]# cat inotifyd.sh
#!



/bin/bash
[ !



 -f /etc/yum.repos.d/epel.repo ] &&\
wget -O /etc/yum.repos.d/epel.repo http:
```

```
// mirrors.aliyun.com/repo/epel-6.repo
if [ $ -eq 0 -a `rpm -qa inotify-tools|wc -l` -ne 1 ];


then
  yum -y install inotify-tools
fi
inotify=/usr/bin/inotifywait
rsync_host=rsync.etiantian.org
[ !


 -d /data0 ]&& mkdir /data0
$inotify -mrq  --format '%w%f' -e create,


close_write,


delete /data0 \
|while read file
do
  cd / &&
  rsync -az /data0 --delete rsync_backup@${rsync_host}: :


backup/ --password-file=/etc/rsync.password
done
```

提示：此脚本依赖于rsync备份服务器

NFS服务器基础重要数据的定时备份见“集群所有服务器系统的基础优化设置”相关内容。

最终部署完成的NFS共享存储服务器逻辑图如图15-5所示，见虚线部分。

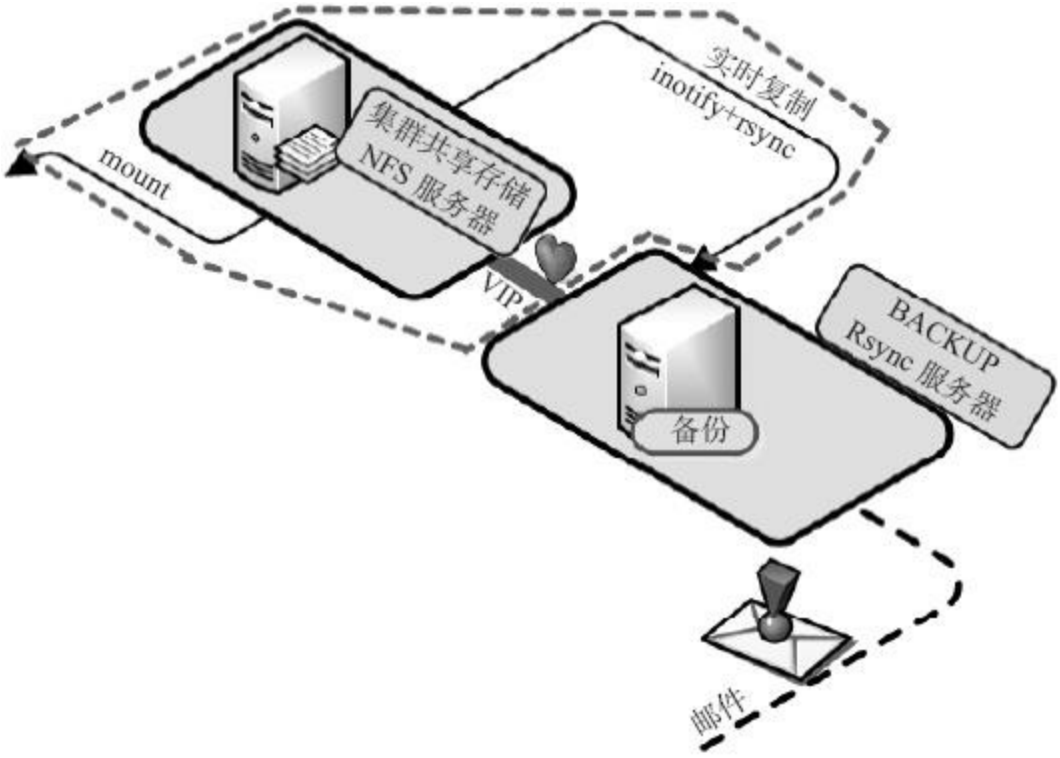


图15-5　NFS共享存储服务逻辑图

5.MySQL数据库服务器安装配置规划设计

本文采用企业典型的MySQL双主多从的数据库架构，所有从库都和主库对外提供服务的VIP进行复制。在部署网站产品数据库数据时，需要将前面章节搭建好的bbs、cms、blog库备份出来，还原到MySQL主库上。表15-14为MySQL服务器在集群中的架构及作用说明，表15-15为MySQL数据库服务器信息规划情况。

表15-14　MySQL服务器在集群中的架构及作用说明

服务器	功能及作用
MySQL 主数据库（eth1:172.16.1.50）（VIP:172.16.1.52）	MySQL 数据库是互联网公司绝大多数集群后端的数据存储，主要用于存储 Web 集群服务器提交的文本内容，并提供各种数据查询及更改服务，例如：用户注册登录，发布帖子、博文等，并通过主从复制技术将在主库修改的数据复制到从数据库上，从而实现数据的备份及热备功能
MySQL 主数据库（热备）（eth1:172.16.1.51）	作为主数据库的热备服务器，可以配置为和主数据库互为热备，通过 Keepalived 服务实现主数据库宕机热备切换，以便继续提供服务
MySQL 从数据库（eth1:172.16.1.52）	所有主库修改的数据都会复制到从数据库上，一般可以配置 3~5 台从数据库，例如：让 3 台从通过 LVS 做负载均衡为用户提供读服务，第 4 台从数据库作为内部开发、数据分析的设备，最后 1 台从开启 binlog 日志，作为数据备份服务器。数据库的读写分离（主库写从库读），多数场景下是在前端 Web 等应用服务器上实现的，开源的实现软件有 mysql-proxy、amoeba 等

表15-15　MySQL数据库服务器信息规划

服务器说明	VIP	内网 IP eth1	名称规划	域名规划
MySQL 主数据库	172.16.1.52	172.16.1.50	MySQL-master-01	dbmaster01.etiantian.org
MySQL 主数据库（热备）		172.16.1.51	MySQL-master-02	dbmaster02.etiantian.org
MySQL 从数据库		172.16.1.52	MySQL-slave-01	dbslave01.etiantian.org

特别说明：MySQL主数据库VIP为172.16.1.52 dbmaster_vip.etiantian.org

图15-6为MySQL数据库服务集群架构的逻辑图，见虚线部分。

一主多从的实现部分请参考第9章“MySQL数据库企业级应用实践”，有关LVS的部署请参考其他文章或者老男孩发布的其他集群课程，这里作为架构框架进行说明。

6.Memcached缓存服务及架构规划设计

Memcached缓存服务器相关说明

表15-16为Memcached缓存服务在集群中的架构及作用说明，表15-17为Memcached缓存服务器信息规划情况。

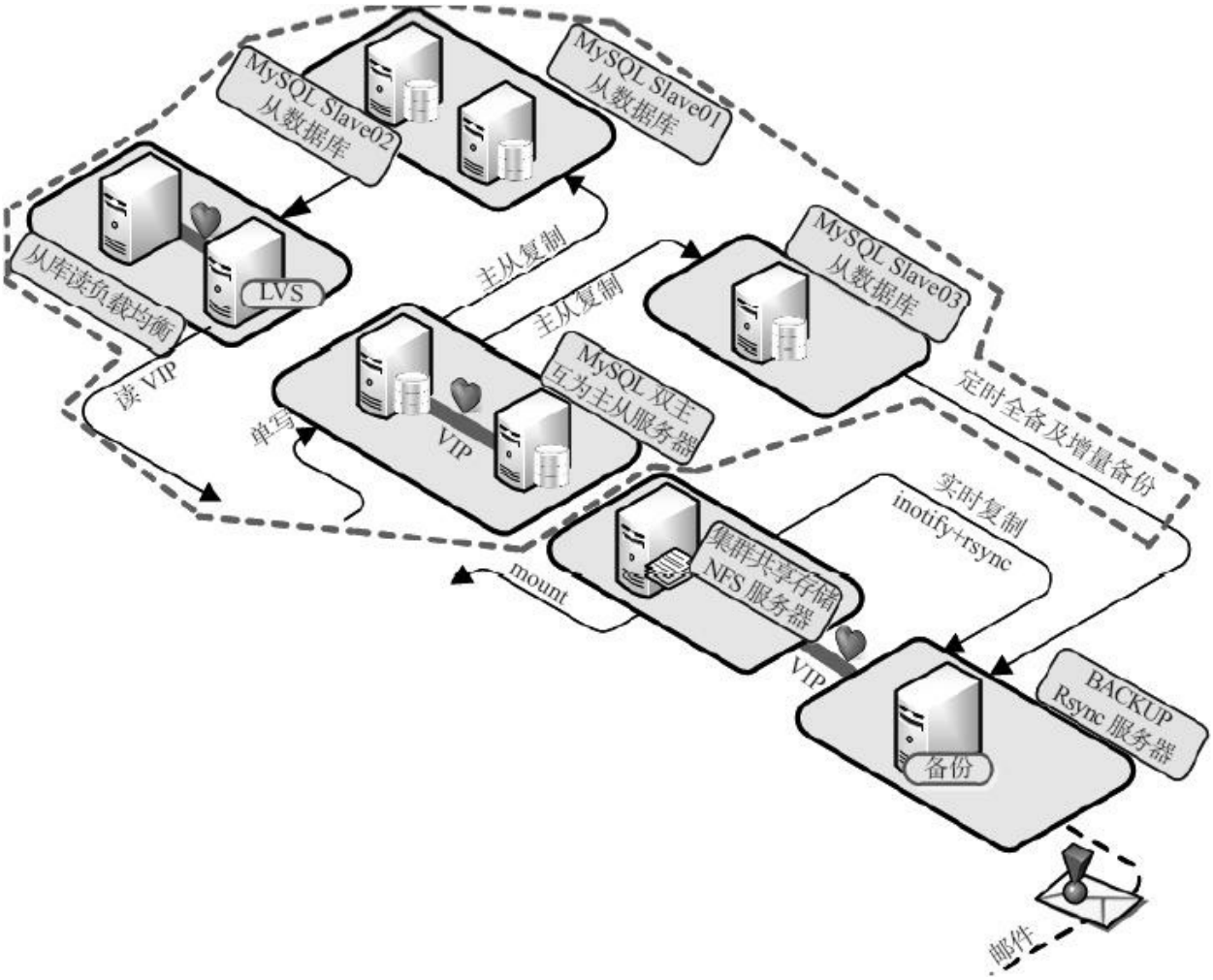


图15-6　MySQL数据库集群架构图

表15-16　Memcached缓存服务在集群中的架构及作用说明

服务器	功能及作用
Memcached 缓存及 session 共享 1（eth1:172.16.1.61）	• 安装 Memcached 服务 • 通过不同的端口使用不同的实例，分别用于缓存及 session 共享 • 实现一致性哈希算法访问 Memcached 缓存服务（开发程序实现） • 如果需要集群可采用 MemcacheDB 或 Redis
Memcached 缓存及 session 共享 2（eth1:172.16.1.62）	• 安装 Memcached 服务 • 通过不同的端口使用不同的实例，分别用于缓存及 session 共享 • 实现一致性散列算法访问 Memcached 缓存服务（开发程序实现） • 如果需要集群可采用 MemcacheDB 或 Redis

表15-17　Memcached缓存服务器信息规划

服务器说明	VIP	内网 IP eth1	名称规划	域名规划
Memcached 缓存及 session 共享 1	172.16.1.63	172.16.1.61	Cache-Mc-01	cache01.etiantian.org
Memcached 缓存及 session 共享 2		172.16.1.62	Cache-Mc-02	cache02.etiantian.org

 特别说明：Memcached缓存服务器的VIP为172.16.1.63 mc_vip.etiantian.org。

关于Memcached缓存服务的安装请参考第13章“Memcached数据缓存服务应用”图15-7为Memcached数据缓存服务架构逻辑图，见虚线部分。

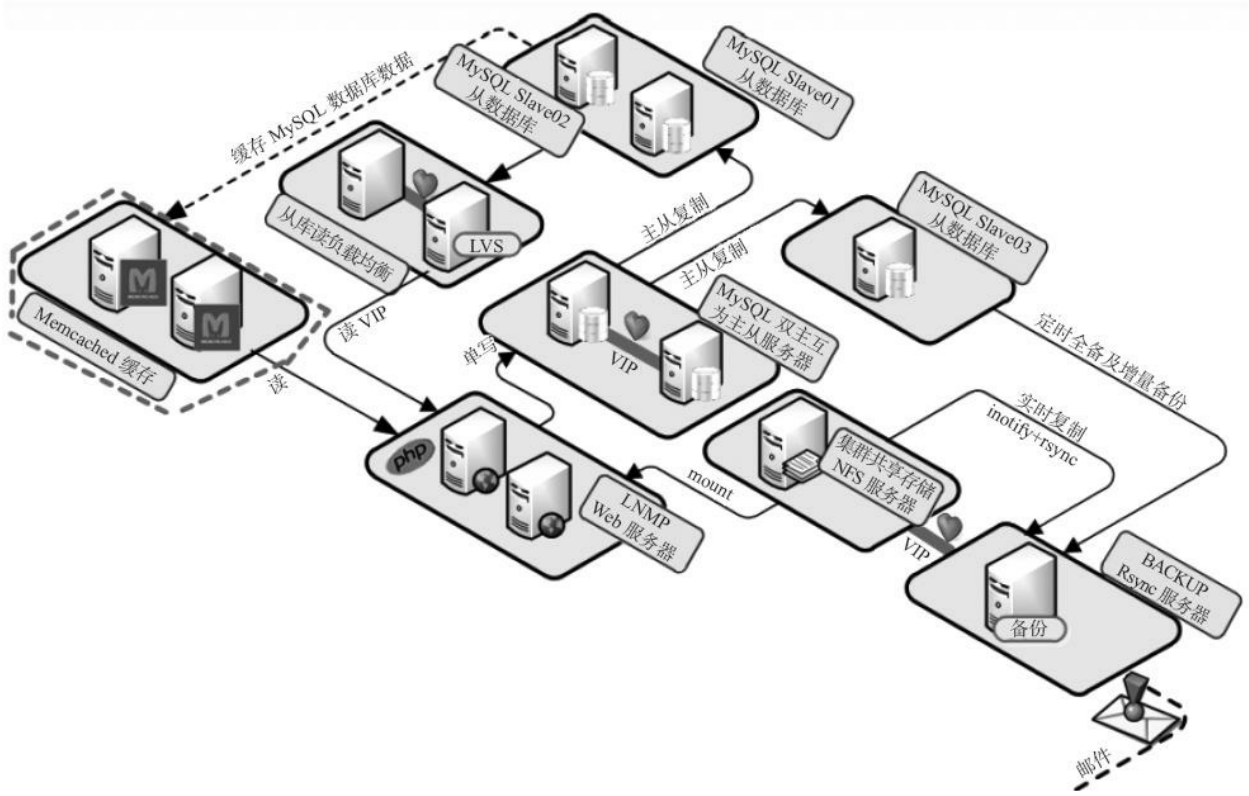


图15-7　Memcached数据缓存服务架构逻辑图

7.LNMP Web服务及架构规划设计

LNMP Web服务器的主要服务为Nginx静态服务，以及结合Nginx的PHP FastcGI动态PHP服务。这里设计搭建一个类51CTO网站的IT资讯平台，具体的搭建说明请参看前面的Web应用章节，有一点特别要注意，在部署bbs、cms、blog三个开源PHP网站产品时，如果之前已经装好了上述产品，这里就不需要从Web界面安装网站产品了，只需打包部署好的网站程序到多个LNMP服务器的对应目录上即可，网站产品数据库的数据也是导出来就可以还原到主数据库服务器上的。表15-18为LNMP Web服务器在集群中的架构及作用说明，表15-19为LNMP Web服务器信

息规划情况。

表15-18　LNMP Web服务器在集群中的架构及作用说明

服务器说明	需要安装的服务功能及作用说明
nginx web01 WEB-LNMP-01 （eth1:172.16.1.10）	• 安装 Nginx 服务软件，用作 Web 服务器 • 安装 PHP 服务软件，用作动态 Web 服务器，负责处理 PHP 动态页面请求 • 部署 bbs、cms、blog 三个开源 PHP 网站产品，模拟搭建类似 51CTO 的 IT 资讯平台部分内容 • 开发通用脚本，并通过定时任务把访问日志、网站程序备份到 rsync 备份服务器 • 开发通用脚本，并通过定时任务把基础配置数据备份到 rsync 备份服务器 • 把网站产品的所有站点内用户上传的资源目录挂载到后端的 NFS 存储服务器的对应目录上，让用户上传资源文件直接放到 NFS 共享服务器上

（续）

服务器说明	需要安装的服务功能及作用说明
nginx web01 WEB-LNMP-01 （eth1:172.16.1.10）	• 安装 amoeba 服务软件（测试可以用 amoeba，工作中多为开发改造程序实现读写分离），提供 Web 程序对数据库的读写分离功能 • 配置 PHP 的配置文件 php.ini，将所有集群节点的 session 会话数据都保存到同一个 Memcached 服务中
nginx web02 WEB-LNMP-02 （eth1:172.16.1.11）	所有配置同 nginx web01

表15-19　LNMP Web服务器信息规划

服务器说明	VIP	内网 IP eth1	名称规划	域名规划
WEB-LNMP-01	10.0.0.3（在前端主负载均衡 eht0 上配置）	172.16.1.61	Cache-Mc-01	cache01.etiantian.org
WEB-LNMP-02		172.16.1.62	Cache-Mc-02	cache02.etiantian.org

特别说明：Memcached缓存服务器VIP为10.0.0.3 lb01_vip.etiantian.org。

图15-8为LNMP Web服务器架构逻辑图，见虚线部分。

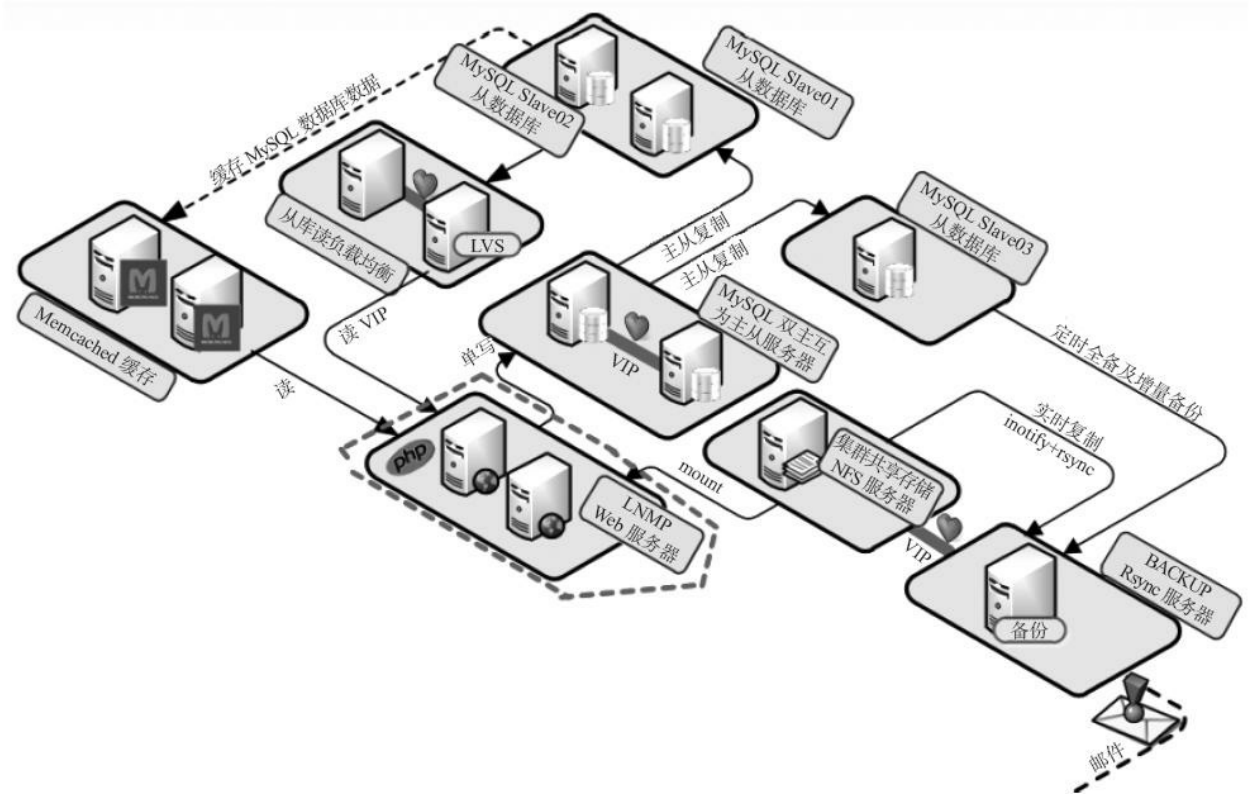


图15-8　LNMP Web服务器架构逻辑图

下面是LNMP Web服务架构图说明。

·LNMP Web服务负责接收用户的请求，将用户上传的资源文件（图片、附件、视频、头像等），放入NFS共享存储服务器中。

·LNMP Web服务将用户发布的帖子博文等内容写入到MySQL主数据库中。

·LNMP Web服务读取用户要请求的数据时，会先读取Memcached缓存服务器，如果有用户请求的数据，则直接返回数据给用户。

·如果Memcached缓存服务器没有用户请求的数据，LNMP Web服务会去请求从负载均衡数据库集群上获取数据。获取数据返回用户后，又把数据缓存到Memca-ched服务内存中，以便下次可以直接从缓存获得数据，从而减轻数据库的压力。

8.Nginx反向代理及负载均衡服务及架构规划设计

表15-20为Nginx反向代理及负载均衡服务器相关说明，表15-21为Nginx反向代理及负载均衡服务器信息规划。

表15-20　Nginx反向代理及负载均衡服务器相关说明

服务器说明	需要安装的服务功能及作用说明
反向代理及负载均衡服务 01	• 安装 Nginx 服务软件，为后端的 Web 服务提供负载及反向代理功能，可以配置成和反向代理服务 02 互为备份，各自为不同的业务提供对外服务 • 安装 Keepalived 服务软件，为两台负载及反向代理服务器提供高可用功能 • 配置通用备份脚本并通过定时任务把基础重要数据备份到 rsync 备份服务器
反向代理及负载均衡服务 02	• 安装 Nginx 服务软件，为后端的 Web 服务提供负载及反向代理功能，可以配置成和反向代理服务 01 互为备份，各自为不同的业务提供对外服务 • 安装 Keepalived 服务软件，为两台负载及反向代理服务器提供高可用功能 • 配置通用备份脚本并通过定时任务把基础重要数据备份到 rsync 备份服务器

表15-21　Nginx反向代理及负载均衡服务器信息规划

服务器说明	外网 eth0	VIP	内网 eth1	名称规划	域名规划
反向代理服务 01	10.0.0.1	10.0.0.3	172.16.1.1	LB-nginx-01	lb01.etiantian.org
反向代理服务 02	10.0.0.2		172.16.1.2	LB-nginx-02	lb02.etiantian.org
反向代理服务 VIP 10.0.0.3 lb_vip.etiantian.org					

图15-9为Nginx反向代理及负载均衡服务架构逻辑图，见虚线部分。

在大并发大流量的时候，企业也会在Nginx反向代理的前面加LVS等四层负载均衡，将Nginx反向代理用于7层应用层的负载均衡，此部分内容请参考老男孩的其他课程。

9.跳板机管理服务器及架构规划设计

表15-22为跳板机管理功能服务器相关说明，表15-23为跳板机管理服务器信息规划。

图15-10为跳板机管理服务架构逻辑图，见虚线部分。

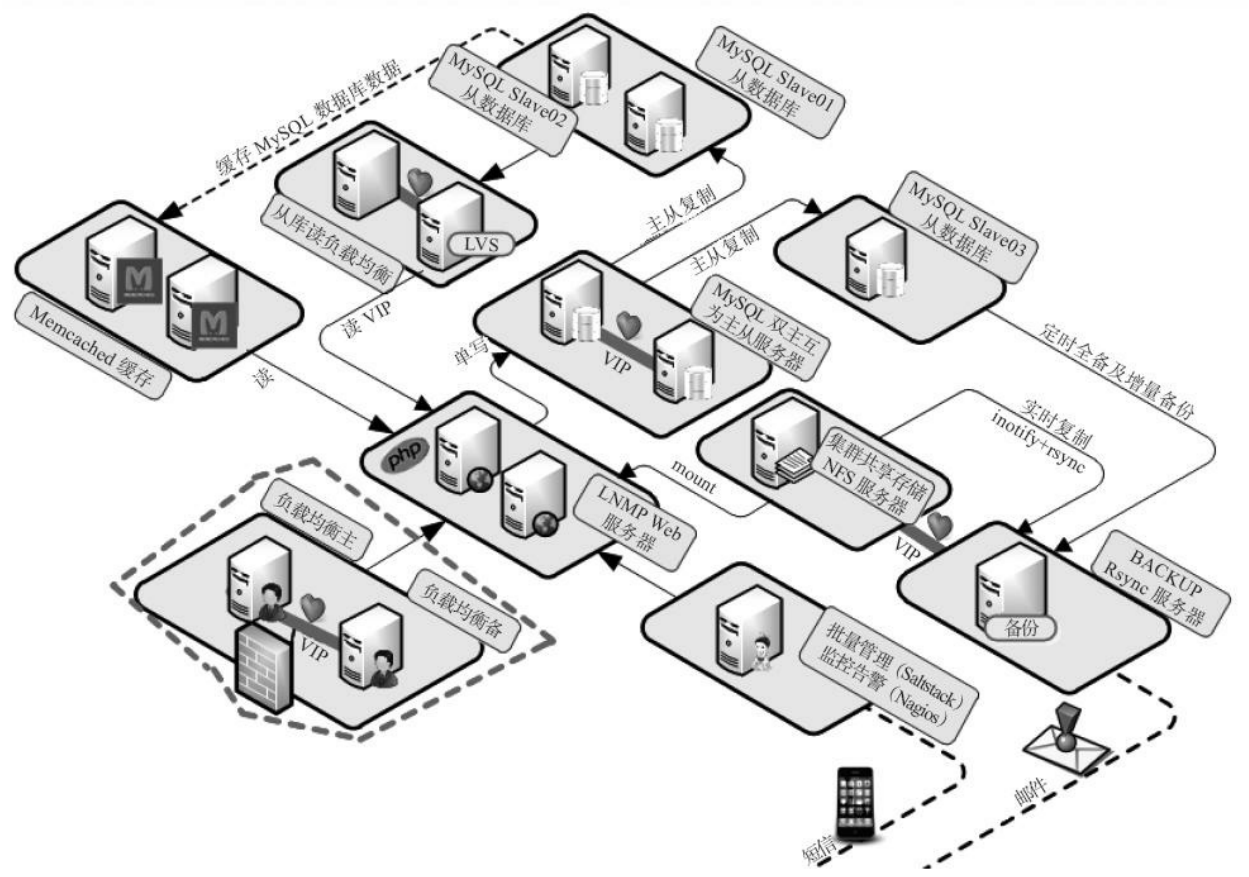


图15-9　Nginx反向代理及负载均衡服务架构逻辑图

表15-22　集群管理功能服务器相关说明

服务器说明	安装服务说明
跳板机管理服务	• 配置 VPN 服务，作为网站集群远程管理的入口 • 对跳板机 SSH 监听本地内网卡 IP，只要不是通过 VPN 拨号的一律禁止采用 SSH 登录服务器，加强服务器安全 • 配置跳板机脚本，实现跳板机管理功能，所有人员通过跳板机管理其他服务器，所有操作都会被审计记录，跳板机和审计的功能可以通过开源软件实现，或者自己利用 Shell 或 Python 开发 • 所有集群服务器不提供密码验证登录，只能从跳板机通过密钥登录

表15-23　跳板机管理服务器信息规划

服务器说明	外网 eth0	VIP	内网 eth1	名称规划	域名规划
跳板机管理服务	10.0.0.201		172.16.1.201	MAGE-jump-01	jump.etiantian.org

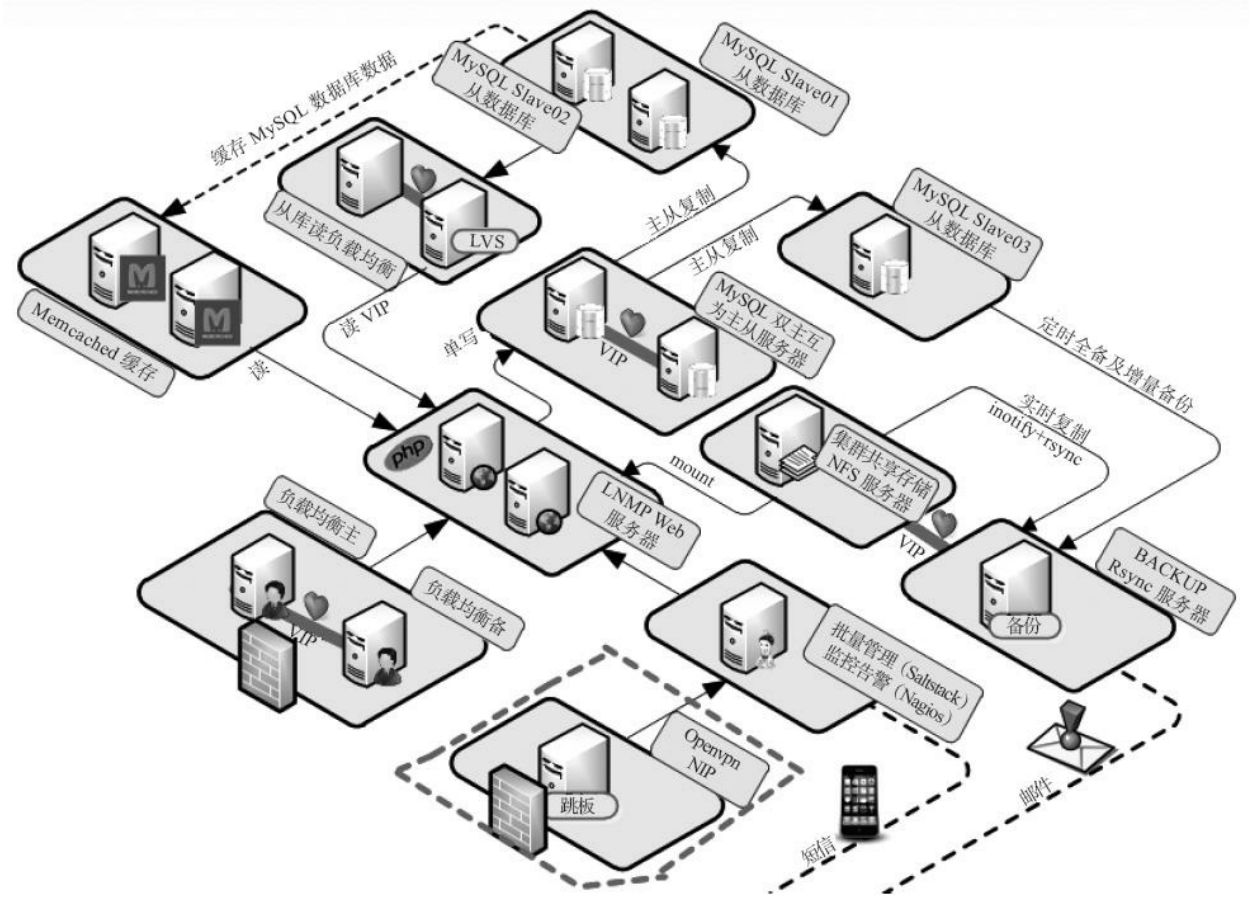


图15-10　集群管理功能服务架构逻辑图

15.3　中小规模网站集群架构综合说明

如果读者能顺利完成了上述设计的架构搭建，那么图15-11就是搭建后的架构图全貌了。

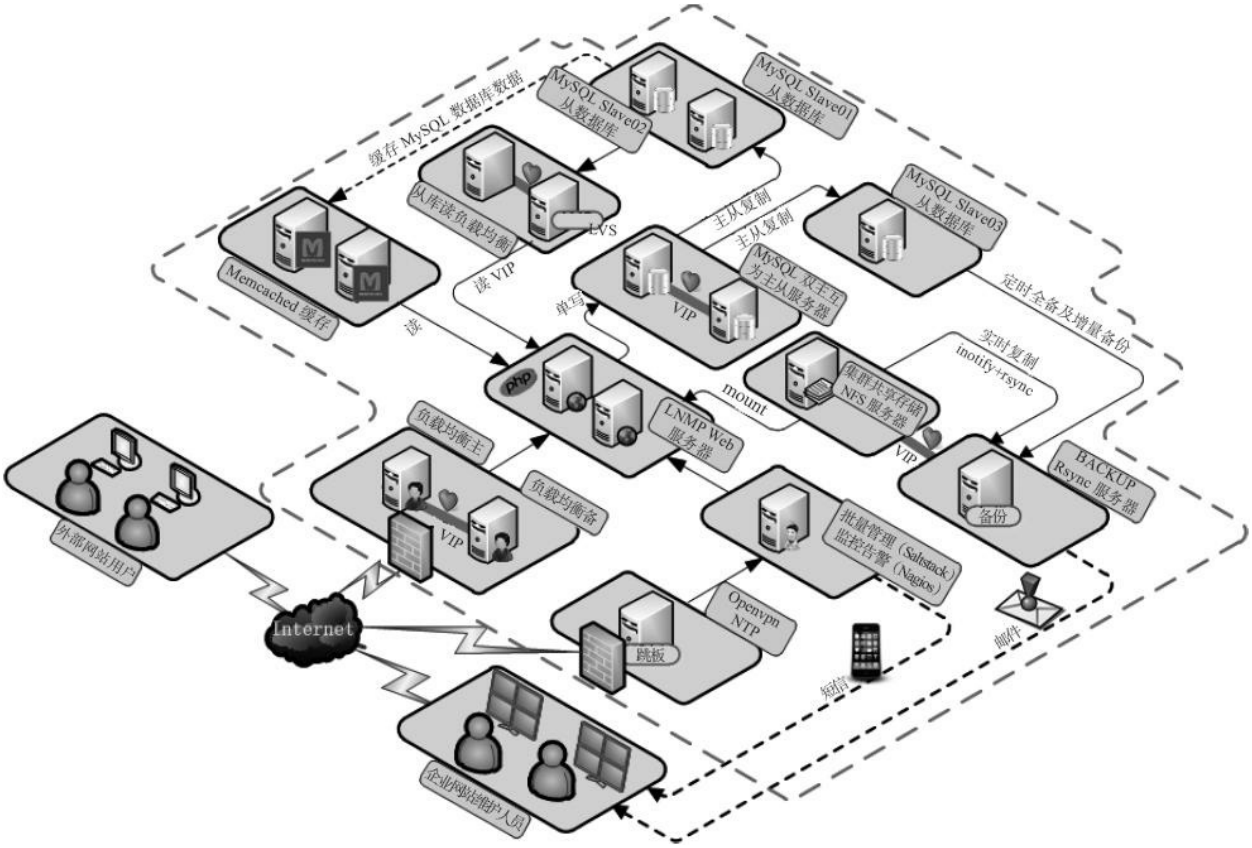


图15-11　企业中小规模网站集群总架构逻辑图

15.3.1 概述

在本书的结尾，为读者梳理一下整个网站的访问流程，希望读者对整个网站架构有一个更好的理解和把握，祝愿大家成为一个合格的Linux网站运维工程师。

·当外部网站用户请求企业域名www.etiantian.org 、blog.etiantian.org、bbs.etiantian.org时，由etiantian.org域名授权的DNS服务器负责将域名解析为IDC网站集群的反向代理负载均衡器上的VIP地址，这个VIP可以对应三个域名，也可以为每个域名使用一个VIP，这些外部的VIP地址，是购买带宽时机房分配给企业的。

·当请求到达负载均衡服务器时，由主负载均衡服务器接收并处理请求，并根据配置规则和调度算法请求后面的Web应用服务器，不同的业务VIP可以由不同的负载均衡服务器处理。例如：www.etiantian.org对应的请求可以由主负载均衡服务器接收并处理，blog.etiantian.org对应的请求可以由备用负载均衡服务器接收并处理，域名解析到不同的VIP时，主备分别使用不同的VIP配置服务。如果门户或大型网站的流量巨大，还可在负载均衡器前再加LVS等处理4层请求的负载均衡器，Nginx负载均衡器只负责处理7层HTTP应用层请求的调度，4层的处理效率比7层的效率要高10倍以上，如果流量再大，可以再加上OSPF调度或DNS

调度以承受更大的并发访问。

·请求到达Web应用服务器后，将根据用户的请求来决定如何操作，如果是向数据库写入数据，则会连接主库将数据库写入到主库；如果是读请求则会先读取数据库缓存Memcached分布式集群，如果缓存中没有对应的数据，Web程序则会通过读写分离软件或LVS负载均衡软件，读取被指定的多个MySQL从库，在将数据返回给用户的同时会将本次读取的数据缓存至Memcached分布式集群；如果是用户登录行为，还会把登录的会话信息写入到Memcached Session缓存中；如果是用户上传图片及附件等资源，Web程序会请求NFS共享存储服务器，将数据存入NFS服务器，数据会经由Inotify+Rsync实时同步到Rsync备份服务器。

·大型网站的Web服务器会根据业务进行拆分，例如：上传服务器集群、静态服务器集群、动态浏览写入Web集群等，电商等网站还会有订单、支付等更细化的Web业务拆分。

·第三台MySQL从库作为数据库的全备及增量备份服务器，全备及增量备份会被定时备份到Rsync备份服务器，当数据库有人为执行的错误SQL语句时，可以使用此处的备份进行恢复。

·跳板机上的SSH仅监听内网卡请求，并安装OpenVPN或pptp VPN服务。需要拨号连接到集群内部，所有服务器不提供密码验证登录，全部要通过跳板机来连接管理，跳板机可以使用Shell或者Python开发，也

可以使用商业版例如齐治堡垒机等产品。

·监控服务器用于监控所有业务的服务器硬软件的好坏、资源使用率及具体业务服务的流量、接口的访问量等，及时向管理员报告相关故障。

·尽量避免服务器的单点故障，负载均衡器、Web服务器、数据库主、数据库从、NFS存储等都尽可能有高可用数据复制及业务接管的功能。

·Rsync负责所有机器的各类数据备份，当业务机器故障或新增服务器时，就可以从备份服务器上进行数据恢复或还原，确保数据安全。

15.3.2　运维人员的两大核心工作主题

网站数据安全和7×24不宕机是运维人员的两大核心工作主题！

网站数据安全是指数据不能丢，不论何时出现何种故障，网站的数据都可以随时快速恢复到最新最完整的状态。

网站7×24不宕机的范围相对更宽一些，它不单是指物理服务器故障及业务服务故障快速恢复的能力，还有当网站的流量成爆发式增长时，网站依然可以高效率地为用户服务，这里就涉及集群架构改造，自动化装系统配服务，自动化监控网站状态，自动化将资源加入集群提供服务等，流量低时还要能收缩集群，将服务器从集群中撤出随时备用。

限于篇幅，本章仅针对企业的网站集群解决方案做了一个详细的规划，并未手把手带读者去从头实现整个搭建过程，这部分内容将会在老男孩的即将出版的另一本新书《全自动化构建Linux企业级网站集群架构实战》里得到更好的体现，新书将通过自动化软件及Shell脚本开发，展现远远超过企业标准的全自动化构建集群的实战全过程，敬请期待。