HZ BOOKS
华章科技

从Web应用、数据备份与恢复、网络存储应用、运维监控与性能优化、集群高级应用等多个方面深入讲解了如何构建高性能的Linux服务器

實戰

高俊峰 著

Build High Performance Linux Servers: Maintainance, Optimization and Cluster

高性能Linux服务器构建实战

运维监控、性能调优与集群应用

机械工业出版社
China Machine Press

實戰

51CTO.com 技术成就梦想

ITPUB www.itpub.net

联合策划

本书很有可能成为Linux服务器构建与运维领域的经典著作之一，从运维监控、性能优化和集群应用等多方面对如何构建高性能的Linux服务器进行了细致的讲解和全面的解析，蕴含了丰富的运维经验。更为重要的是，本书的内容不受硬件环境的限制，同时包含大量实用性极强的案例。对于广大Linux服务器运维人员来说，真可谓“一书在手，运维不愁”。

—— ITPUB技术论坛（http://www.itpub.net/）

开源赋予了Linux强大的生命力，Linux因为开源而聚集了全球技术精英的智慧。本书围绕“高性能”这个话题，先从实用的角度详细讲解了各种与构建高性能Linux服务器相关的开源软件的配置、使用、管理和维护；然后结合实际生产环境讲解了Web应用、数据备份与恢复、网络存储应用、性能优化与运维监控、集群高级应用等方面的知识，能给Linux运维人员和系统管理人员非常实用的指导。

—— 51CTO（www.51cto.com）

从内容上讲，本书基本涵盖了当前Linux服务器系统运维所需要的主流技术。与大部分运维手册性质的书籍不同的是，本书注重于实践，包含大量来自于实际生产环境中的案例，能帮助我们解决很多实际工作中会遇到的问题，实践指导意义很强。我将这本书推荐本给所有从事Linux服务器运维的同行，希望它能帮助大家提高技能、收获经验，最终实现升职加薪的愿望。

—— 田逸 资深系统架构师/《互联网运营智慧：高可用可扩展网站技术实战》作者

“南非蚂蚁”是我多年的好友，也是我的同行，我深知他的系统运维功底相当深厚，在这个领域积累了丰富的经验。他能将自己的这些经验梳理、总结并以书的方式分享出来，实在是难能可贵。本书从Web应用、数据备份与恢复、网络存储应用、性能优化与运维监控、集群高级应用等方面讲解了构建高性能Linux服务器的方法与最佳实践。此外，本书的叙述通俗易懂、语言幽默风趣，可读性也很好。如果你拿起了这本书，就建议你不要再犹豫了，按照书中的内容去实践吧，相信你一定会大有收获。

—— 杨向勇 51JOB系统运维经理兼首席DBA

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

上架指导：计算机/操作系统
ISBN 978-7-111-36695-9
9 787111 366959

定价：79.00元

實戰

Build High Performance Linux Servers: Maintainance, Optimization and Cluster

高性能Linux服务器构建实战

运维监控、性能调优与集群应用

高俊峰 著

机械工业出版社
China Machine Press

本书以构建高性能 Linux 服务器为核心内容，从 Web 应用、数据备份与恢复、网络存储应用、运维监控与性能优化、集群高级应用等多个方面深入讲解了如何构建高性能的 Linux 服务器。全书以实战性为导向，所有内容都来自于作者多年实践经验的总结，同时从社区中收集了大量 Linux 运维人员遇到的有代表性的疑难问题，并给出了优秀的解决方案，实践指导意义极强。

全书分为 5 个部分。Web 应用篇详细介绍了 Nginx、Varnish 和 Memcached 这三款 Linux 服务器上极为常用的 Web 应用软件的安装、配置、管理、使用方法、工作原理和性能调优技巧。数据备份与恢复篇首先讲述了开源备份软件 bacula 的使用与管理技巧，并通过实例讲解了在 bacula 上进行各种备份与恢复操作的具体方法；其次讲解了开源数据镜像备份工具 rsync 和 unison 的使用，并通过两个企业级案例演示了这两个工具在生产环境中的使用过程；最后讲解了如何利用 ext3grep 工具来恢复误删除的数据文件和 MySQL 数据库的方法。网络存储应用篇首先系统地讲解了网络存储技术 iSCSI 的配置和使用，然后讲解了分布式存储系统 MFS 的使用和维护。运维监控与性能优化篇通过理论与实践相结合的方法讲解了如何利用 Nagios 进行性能监控，以及 Linux 服务器的性能分析原则和优化方法。集群高级应用篇是前面内容的综合，也是本书的核心，主要讲述了如何通过 LVS+heartbeat、piranha、LVS+Keepalived 来构建高可用的负载均衡集群，其次讲解了红帽集群套件 RHCS 的配置、管理、维护和监控，然后讲解了 Oracle 集群解决方案，即 Oracle RAC 数据库的构建、使用和维护。最后以构建一个 MySQL+heartbeat+DRBD+LVS 集群系统的实战案例结束全书，巧妙地将本书的所有核心内容都融合到了一起。

图书在版编目（CIP）数据

高性能 Linux 服务器构建实战：运维监控、性能调优与集群应用 / 高俊峰著 . —北京：机械工业出版社，2011.12

ISBN 978-7-111-36695-9

I. 高…　II. 高…　III. Linux 操作系统　IV. TP316.89

中国版本图书馆 CIP 数据核字（2011）第 251700 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）
责任编辑：姜　影
北京京北印刷有限公司印刷
2012 年 3 月第 1 版第 2 次印刷
186mm×240mm　• 29.5 印张
标准书号：ISBN 978-7-111-36695-9
定价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

前　言

为什么要写这本书

随着企业信息系统的广泛应用和深入发展，用户核心应用数量越来越多，企业对业务系统的性能需求越来越高，高稳定性、高可靠性成为评价业务系统性能的主要指标。在这种趋势下，分布式应用系统构架应运而生，高性能多节点的集群系统日益被广泛接受和使用，企业应用进入了集群和高性能时代。与几个主要硬件厂商（例如 IBM、HP、SGI 等）开始研制并有计划地推出基于 Linux 开放源码的集群产品，集群软件也逐渐从 UNIX 平台的高端应用向基于 Linux 平台发展，Linux 由此进入了企业应用的高端市场，同时，一些老牌的 Linux 厂商更是把 Linux 集群这一高端应用领域作为自己的战略发展方向，不遗余力地加入激烈的市场产品竞争中。

开放源码的迅猛发展为集群的出现提供了良好的技术平台，目前，已经有多种多样可供选择的集群解决方案，这些方案有基于硬件的，也有纯软件的，那么，如何选择这些集群软件和集群方案呢？借助开源软件丰富的技术资源，构建一个优秀的集群系统，是技术人员要解决的首要问题，这也正是写作本书的目的。

目前市场上关于 Linux 系统管理、维护和优化的书籍很多，但是普遍存在模式单一的现象，要么只讲基础理论和系统命令，要么侧重代码示例，要么针对具体的系统版本（Redhat Linux/Ubuntu Linux 等），要么缺少实践应用，很少对 Linux 进行全面、深入、灵活的讲解。

本书针对这种现象，从基础入手，再进行深入研究，同时结合实际的应用案例进行由点到面、由浅入深的讲述，将 Linux 应用的各个方面系统、深入、全面地展现给读者。理论介绍

结合实际应用贯穿全书，通过真实案例使读者可以更深入地了解 Linux 应用的现实环境，从而真正提高实践能力。

本书分为 5 篇，以 Linux 平台下的应用软件为中心，涉及 Linux 运维的各个方面，包括 Web 应用方面、数据备份恢复方面、网络存储应用方面、运维监控与性能优化方面、集群高级应用方面，其中，前四个方面是 Linux 运维的核心内容，是本书的基础，而最后一个方面是前面内容的综合和深入，更是本书介绍的重点。读完本书相信读者一定会有一种豁然开朗的感觉，而这正是我们所期待的。

本书是作者多年实践工作的经验总结，全书贯穿了由点及线、由线及面的学习方法，既可以供初学者参考学习，也可以帮助有一定基础的中高级 Linux 系统管理员进阶学习，使不同层次的读者都能从本书受益。

读者对象

本书适合的阅读对象有：

- 中 / 高级 Linux 系统管理员
- 系统运维工程师
- 系统集成商
- 解决方案构架师
- 所有从事开源的爱好者

如何阅读本书

本书最大的特点是注重实践、理论与实际相结合，在讲述完一个知识点后，一般都附有实例对知识点进行补充，并且每个章节都是一个独立的知识块，读者可以选择从中间阅读，也可以从第 1 章依次阅读。

本书分为 5 篇，共 14 章，基本结构如下。

Web 应用篇（第 1 章至第 3 章）

Web 应用篇介绍了 3 个 Web 应用软件的使用：Nginx、Varnish 和 Memcached。

第 1 章讲述了轻量级 HTTP 服务器 Nginx 的安装、配置、管理和使用技巧，同时也介绍了 Nginx 在性能优化方面的一些经验和技巧，最后通过实例演示了 Nginx 与 PHP 以及 Nginx 与 Java、Perl 整合的过程。

第 2 章介绍了高性能 HTTP 加速器 Varnish 的安装、配置、管理和使用技巧，同时还介绍了 Varnish 常用的指令及 Varnish 的调优技巧和经验，最后通过两个实例介绍了 Varnish 的应用。

第 3 章介绍了 Memcached 的特征和安装过程，同时详细介绍了 Memcached 的工作原理和

性能监控方法，最后介绍了 Memcached 的几种升级产品和 Memcached 的使用经验。

数据备份恢复篇（第 4 章至第 6 章）

数据备份恢复篇介绍了几个常用的备份恢复软件：bacula、rsync、unison 和 ext3grep。

第 4 章讲述了开源备份软件 bacula 的安装、使用与管理技巧，然后通过实例方式介绍了在 bacula 上进行完全备份、增量备份、差异备份、完全恢复、不完全恢复的具体操作步骤。

第 5 章讲述了开源数据镜像备份工具 rsync 和 unison 的使用，主要介绍两个软件的安装和配置，接着通过两个实际的企业案例演示了这两个工具在生产环境中的使用过程。

第 6 章讲述了如何利用 ext3grep 工具恢复误删除的数据文件和 MySQL 数据库的方法。

网络存储应用篇（第 7 章和第 8 章）

网络存储应用篇介绍了网络存储技术 iSCSI 和分布式存储系统 MFS。第 7 章主要讲述 iSCSI 的概念、组成、安装、配置和使用，还介绍了 iSCSI 在安全方面的设置方法。第 8 章介绍了分布式文件存储系统 MFS 的结构、安装配置和基本的管理维护。

运维监控与性能优化篇（第 9 章和第 10 章）

运维监控与性能优化篇介绍了 Nagios 监控软件及 Linux 服务器的性能优化方法。第 9 章主要讲述 Nagios 的安装、配置和使用，以及如何利用外部插件扩展 Nagios 的监控功能。第 10 章通过理论与实践相结合的方法系统地讲述了 Linux 服务器的性能分析原则和优化方法，同时，还通过两个具体的案例一步步演示 Web 应用系统的优化过程。

集群高级应用篇（第 11 章至第 14 章）

集群高级应用篇是本书的重点，主要介绍了高可用集群和负载均衡集群的应用案例。第 11 章讲述了通过 LVS+heartbeat、piranha、LVS+Keepalived 构建高可用的负载均衡集群的方法。第 12 章介绍了红帽集群套件 RHCS 的安装、配置、管理、维护和监控。第 13 章讲解了 Oracle 集群解决方案，即 Oracle RAC 数据库的搭建、使用和维护过程。第 14 章是本书内容的综合体，从实战出发，系统讲述了 MySQL+heartbeat+DRBD+LVS 集群解决方案的搭建过程。

勘误和支持

本书主要由高俊峰编写，其中，第 3 章、第 14 章杨海潮参与撰写，第 8 章陶利军参与撰写。

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不准确的地方，不妥之处恳请读者批评指正。

本书的修订信息会发布在笔者的博客上，地址为 http://ixdba.blog.51cto.com。笔者会在该博客中不定期更新书中的遗漏，当然，也欢迎读者将遇到的疑惑或书中的错误在博客留言中

提出。如果您有更多的宝贵意见，也欢迎发送邮件至笔者的邮箱（m13388@163.com），或加入本书的微群（q.weibo.com/943166），期待能够收到你们的真挚反馈。

致谢

首先要感谢我的爸爸妈妈，感谢你们将我培养成人，并时时刻刻向我传递信心和力量！

感谢好友杨海潮、陶利军、郑辉，他们从技术角度对本书某些章节进行了修改和补充，并提出了很多意见和建议。

感谢我所在公司的杨武先生，感谢我的挚友王超、兰海文，是他们的鼓励和支持让我坚持写完了这本书。

感谢机械工业出版社华章公司的编辑杨福川老师和姜影老师，此书的出版离不开他们的辛苦付出。

感谢 IXPUB 社区每一位充满创意和活力的朋友——IXPUB 管理员齐宝玮（网络忏悔）、资深系统管理员田逸（sery）、IXPUB 版主郭瑞佳（grjboy30）、IXPUB 版主李康（winsky）、CU 网友昌德胜（molecar），以及社区中遇到的其他朋友，感谢你们长期对社区的支持和贡献。感谢华章培训网妙妙老师的引荐，您的努力才促成了这本书的出版。

谨以此书献给我最亲爱的家人以及众多热爱 Linux 的朋友们。

高俊峰（南非蚂蚁）

目 录

第 2 篇　数据备份恢复篇

第 4 章　开源网络备份软件 bacula / 92

第 5 章　数据镜像备份工具 rsync 与 unison / 126

第 3 篇　网络存储应用篇

第 7 章　IP 网络存储 iSCSI / 162

第 8 章　分布式存储系统 MFS / 184

第 12 章　RHCS 集群 / 296

第 13 章　Oracle RAC 集群 / 353

第 1 篇

Web 应用篇

第 1 章　轻量级 HTTP 服务器 Nginx

本章主要介绍 Nginx 的配置管理和使用。作为一个轻量级的 HTTP 服务器，Nginx 与 Apache 相比有以下优势：在性能上，它占用很少的系统资源，能支持更多的并发连接，达到更高的访问效率；在功能上，Nginx 是优秀的代理服务器和负载均衡服务器；在安装配置上，Nginx 安装简单、配置灵活。下面就详细介绍 Nginx 的配置与使用。

1.1　什么是 Nginx

相信很多读者都对 Apache 非常熟悉，Nginx 与 Apache 类似，也是一款高性能的 HTTP 和反向代理服务器软件，还是一个 IMAP/POP3/SMTP 代理服务器。Nginx（发音是“engine x”）由俄罗斯的程序设计师 Igor Sysoev 开发（Igor 将源代码以类 BSD 许可证的形式发布），可以运行在 UNIX、GNU/Linux、BSD、Mac OS X、Solaris 以及 Microsoft Windows 等操作系统中。随着 Nginx 在很多大型网站的广泛使用，其稳定、高效的特性逐渐被越来越多的用户认可。

1.2　为什么要选择 Nginx

1.2.1　Nginx 与 Apache 的异同

Nginx 和 Apache 一样，都是 HTTP 服务器软件，在功能实现上都采用模块化结构设计，都支持通用的语言接口，如 PHP、Perl、Python 等，同时还支持正向和反向代理、虚拟主机、URL 重写、压缩传输、SSL 加密传输等。它们之间最大的差别是 Apache 的处理速度很慢，且占用很多内存资源，而 Nginx 却恰恰相反；在功能实现上，Apache 的所有模块都支持动、静态编译，而 Nginx 模块都是静态编译的，同时，Apache 对 Fcgi 的支持不好，而 Nginx 对 Fcgi 的支持非常好；在处理连接方式上，Nginx 支持 epoll，而 Apache 却不支持；在空间使用上，Nginx 安装包仅仅只有几百 K，和 Nginx 比起来 Apache 绝对是庞然大物。在了解了 Nginx 和 Apache 之间的异同点后基本上就知道了 Nginx 作为 HTTP 服务器的优势所在。

1.2.2　选择 Nginx 的优势所在

通过上面的简单介绍，可以看出，Nginx 作为 HTTP 服务器的优势是显而易见的，它有很多其他 Web 服务器无法比拟的性能和优势：

- 作为 Web 服务器，Nginx 处理静态文件、索引文件，自动索引的效率非常高。
- 作为代理服务器，Nginx 可以实现无缓存的反向代理加速，提高网站运行速度。
- 作为负载均衡服务器，Nginx 既可以在内部直接支持 Rails 和 PHP，也可以支持 HTTP 代理服务器对外进行服务，同时还支持简单的容错和利用算法进行负载均衡。
- 在性能方面，Nginx 是专门为性能优化而开发的，在实现上非常注重效率。它采用内核 Poll 模型，可以支持更多的并发连接，最大可以支持对 50 000 个并发连接数的响应，而且只占用很低的内存资源。
- 在稳定性方面，Nginx 采取了分阶段资源分配技术，使得 CPU 与内存的占用率非常低。Nginx 官方表示，Nginx 保持 10 000 个没有活动的连接，而这些连接只占用 2.5MB 内存，因此，类似 DOS 这样的攻击对 Nginx 来说基本上是没有任何作用的。
- 在高可用性方面，Nginx 支持热部署，启动速度特别迅速，因此可以在不间断服务的情况下，对软件版本或者配置进行升级，即使运行数月也无需重新启动，几乎可以做到 7×24 小时不间断地运行。

1.3　Nginx 的模块与工作原理

Nginx 由内核和模块组成，其中，内核的设计非常微小和简洁，完成的工作也非常简单，仅仅通过查找配置文件将客户端请求映射到一个 location block（location 是 Nginx 配置中的一个指令，用于 URL 匹配），而在这个 location 中所配置的每个指令将会启动不同的模块去完成相应的工作。

Nginx 的模块从结构上分为核心模块、基础模块和第三方模块，HTTP 模块、EVENT 模块和 MAIL 模块等属于核心模块，HTTP Access 模块、HTTP FastCGI 模块、HTTP Proxy 模块和 HTTP Rewrite 模块属于基础模块，而 HTTP Upstream Request Hash 模块、Notice 模块和 HTTP Access Key 模块属于第三方模块，用户根据自己的需要开发的模块都属于第三方模块。正是有了这么多模块的支撑，Nginx 的功能才会如此强大。

Nginx 的模块从功能上分为如下三类。

- Handlers（处理器模块）。此类模块直接处理请求，并进行输出内容和修改 headers 信息等操作。Handlers 处理器模块一般只能有一个。
- Filters（过滤器模块）。此类模块主要对其他处理器模块输出的内容进行修改操作，最后由 Nginx 输出。
- Proxies（代理类模块）。此类模块是 Nginx 的 HTTP Upstream 之类的模块，这些模块主要与后端一些服务比如 FastCGI 等进行交互，实现服务代理和负载均衡等功能。

图 1-1 展示了 Nginx 模块常规的 HTTP 请求和响应的过程。

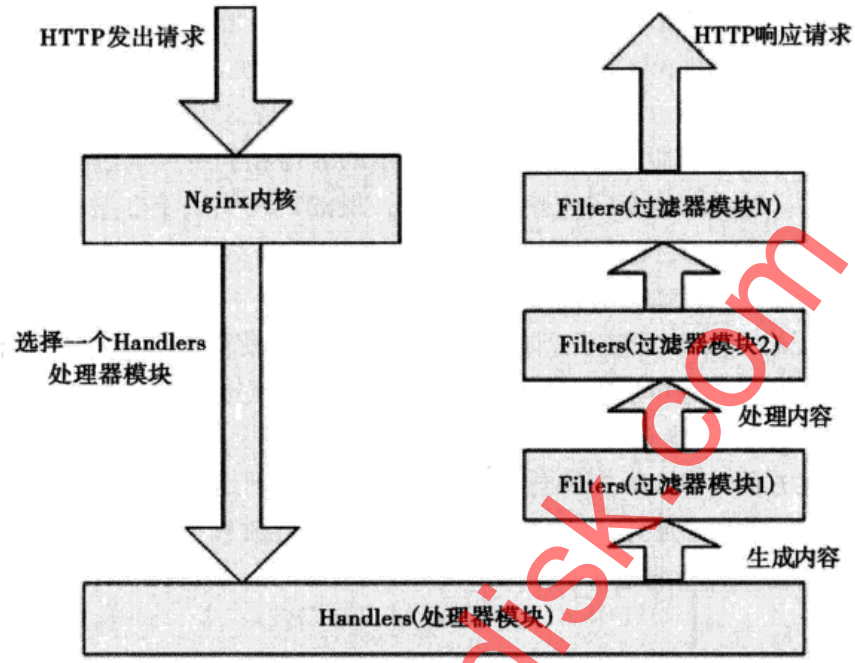

图 1-1 Nginx 模块的 HTTP 请求和响应过程

在工作方式上，Nginx 分为单工作进程和多工作进程两种模式。在单工作进程模式下，除主进程外，还有一个工作进程，工作进程是单线程的；在多工作进程模式下，每个工作进程包含多个线程。Nginx 默认为单工作进程模式。

Nginx 的模块直接被编译进 Nginx，因此属于静态编译方式。启动 Nginx 后，Nginx 的模块被自动加载，不像 Apache，首先将模块编译为一个 so 文件，然后在配置文件中指定是否进行加载。在解析配置文件时，Nginx 的每个模块都有可能去处理某个请求，但是同一个处理请求只能由一个模块来完成。

1.4 Nginx 的安装与配置

1.4.1 下载与安装 Nginx

Nginx 的官方网站是 http://sysoev.ru/nginx/，英文主页为 http://nginx.net，从这里可以获得 Nginx 的最新版本信息。Nginx 有三个版本：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，但同时也可能会出现新的 bug。开发版一旦更新稳定下来，就会被加入稳定版分支中。然而有些新功能不一定会被加到稳定版中去。稳定版更新较慢，但是 bug 较少，可以作为生产环境的首选，因此通常建议使用稳定版。历史稳定版为以往稳定版本的汇总，不包含最新的功能。

这里选择当前的稳定版本 nginx-0.7.65 作为介绍对象，开始介绍编译安装。在安装 Nginx 之前，确保系统已经安装了 gcc、openssl-devel、pcre-devel 和 zlib-devel 软件库。

Linux 开发库是在安装系统时通过手动选择安装的，gcc、openssl-devel、zlib-devel 三个软件库可以通过安装光盘直接选择安装，而 pcre-devel 库默认不在系统光盘中，所以这里重点介绍 pcre-devel 库。

1. 安装 Nginx 所需的 pcre-devel 库

安装 pcre 库是为了使 Nginx 支持 HTTP Rewrite 模块。下面进行安装，过程如下：

```
[root@localhost home]# tar zxvf pcre-8.02.tar.gz
[root@localhost home]# cd pcre-8.02
[root@localhost pcre-8.02]#./configure
[root@localhost pcre-8.02]#make
[root@localhost pcre-8.02]#make install
```

2. 开始安装 Nginx

Nginx 的安装非常简单。在默认情况下，经过编译安装的 Nginx 包含了大部分可用模块。可以通过“./configure --help”选项设置各个模块的使用情况，例如对不需要的 http_ssi 模块，可通过“--without-http_ssi_module”方式关闭此。同理，如果需要“http_perl”模块，那么可以通过“--with-http_perl_module”方式进行安装。下面是安装过程：

```
[root@localhost home]# tar zxvf nginx-0.7.65.tar.gz
[root@localhost home]#cd nginx-0.7.65
[root@localhostnginx-0.7.65]#./configure \
--with-http_stub_status_module --prefix=/opt/nginx
[root@localhost nginx-0.7.65]#make
[root@localhost nginx-0.7.65]#make install
```

在上面的 configure 选项中，“--with-http_stub_status_module”可以用来启用 Nginx 的 NginxStatus 功能，以监控 Nginx 的当前状态。

至此，Nginx 已经安装完成了。

1.4.2　Nginx 配置文件的结构

Nginx 的配置文件是一个纯文本文件，它一般位于 Nginx 安装目录的 conf 目录下，整个配置文件是以 block 的形式组织的。每个 block 一般以一个大括号“{}”来表示，block 可以分为几个层次，整个配置文件中 main 指令位于最高层，在 main 层下面可以有 Events、HTTP 等层级，而在 HTTP 层中又包含有 server 层，即 server block，server block 中又可分为 location 层，并且一个 server block 中可以包含多个 location block。

一个完整的配置文件结构如图 1-2 所示。

在了解完配置文件结构之后，就可以开始配置和调试 Nginx 了。

1.4.3 配置与调试 Nginx

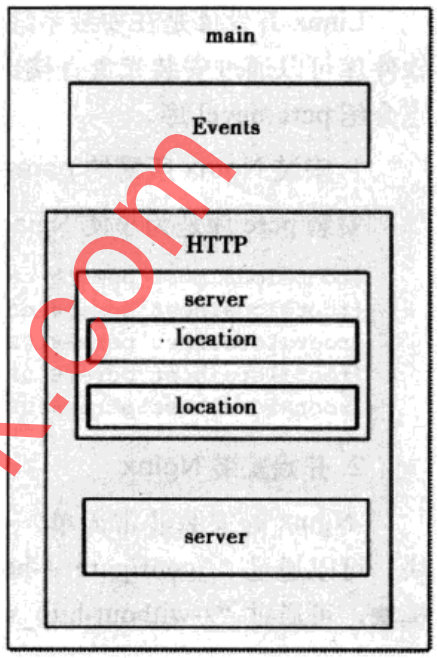

图 1-2 Nginx 配置文件结构

Nginx 安装完毕后，会产生相应的安装目录，根据前面的安装路径，Nginx 的配置文件路径为 /opt/nginx/conf，其中 nginx.conf 为 Nginx 的主配置文件。这里重点介绍 nginx.conf 这个配置文件。

Nginx 配置文件主要分为 4 部分：main（全局设置）、server（主机设置）、upstream（负载均衡服务器设置）和 location（URL 匹配特定位置的设置）。main 部分设置的指令将影响其他所有设置；server 部分的指令主要用于指定主机和端口；upstream 指令主要用于负载均衡，设置一系列的后端服务器；location 部分用于匹配网页位置。这四者之间的关系如下：server 继承 main，location 继承 server，upstream 既不会继承其他设置也不会被继承。

在这 4 个部分当中，每个部分都包含若干指令，这些指令主要包含 Nginx 的主模块指令、事件模块指令、HTTP 核心模块指令。同时每个部分还可以使用其他 HTTP 模块指令，例如 Http SSL 模块、Http Gzip Static 模块和 Http Addition 模块等。

下面通过一个 Nginx 配置实例，详细介绍 nginx.conf 每个指令的含义。为了能更清楚地了解 Nginx 的结构和每个配置选项的含义，这里按照功能点将 Nginx 配置文件分为 7 个部分依次讲解。下面就围绕这 7 个部分进行介绍。

1. Nginx 的全局配置

下面这段内容是对 Nginx 的全局属性配置，代码如下：

```
user  nobody nobody;
worker_processes  4;
error_log  logs/error.log  notice;
pid        logs/nginx.pid;
worker_rlimit_nofile 65535;
events{
        use epoll;
        worker_connections      65536;
      }
```

上面这段代码中每个配置选项的含义解释如下：

- user 是个主模块指令，指定 Nginx Worker 进程运行用户以及用户组，默认由 nobody 账号运行。

- worker_processes 是个主模块指令，指定了 Nginx 要开启的进程数。每个 Nginx 进程平均耗费 10MB ～ 12MB 内存。根据经验，一般指定一个进程足够了，如果是多核 CPU，建议指定和 CPU 的数量一样多的进程数即可。
- error_log 是个主模块指令，用来定义全局错误日志文件。日志输出级别有 debug、info、notice、warn、error、crit 可供选择，其中，debug 输出日志最为最详细，而 crit 输出日志最少。
- pid 是个主模块指令，用来指定进程 id 的存储文件位置。
- worker_rlimit_nofile 用于绑定 worker 进程和 CPU，Linux 内核 2.4 以上可用。
- events 指令用来设定 Nginx 的工作模式及连接数上限。
- use 是个事件模块指令，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，epoll 工作模式是首选。
- 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，在作为反向代理时变为：max_clients = worker_processes * worker_connections/4。

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 -n 65536”后 worker_connections 的设置才能生效。

2. HTTP 服务器配置

接下来开始对 HTTP 服务器进行配置。

下面这段内容是 Nginx 对 HTTP 服务器相关属性的配置，代码如下：

```
http{
include       conf/mime.types;
default_type  application/octet-stream;
log_format main '$remote_addr - $remote_user [$time_local] '
    '"$request" $status $bytes_sent '
    '"$http_referer" "$http_user_agent" '
    '"$gzip_ratio"';
    log_format download '$remote_addr - $remote_user [$time_local] '
    '"$request" $status $bytes_sent '
    '"$http_referer" "$http_user_agent" '
    '"$http_range" "$sent_http_content_range"';
client_max_body_size  20m;
client_header_buffer_size    32k;
large_client_header_buffers  4 32k;
sendfile  on;
tcp_nopush     on;
tcp_nodelay    on;
```

```
keepalive_timeout 60;
client_header_timeout  10;
client_body_timeout    10;
send_timeout           10;
```

下面详细介绍这段代码中每个配置选项的含义。

- include是个主模块指令，实现对配置文件所包含的文件的设定，可以减少主配置文件的复杂度。类似于Apache中的include方法。
- default_type属于HTTP核心模块指令，这里设定默认类型为二进制流，也就是当文件类型未定义时使用这种方式，例如在没有配置PHP环境时，Nginx是不予解析的，此时，用浏览器访问PHP文件就会出现下载窗口。

下面的代码实现对日志格式的设定。

```
log_format main '$remote_addr - $remote_user [$time_local] '
'"$request" $status $bytes_sent '
'"$http_referer" "$http_user_agent" '
'"$gzip_ratio"';
log_format download '$remote_addr - $remote_user [$time_local] '
'"$request" $status $bytes_sent '
'"$http_referer" "$http_user_agent" '
'"$http_range" "$sent_http_content_range"';
```

- log_format是Nginx的HttpLog模块指令，用于指定Nginx日志的输出格式。main为此日志输出格式的名称，可以在下面的access_log指令中引用。
- client_max_body_size用来设置允许客户端请求的最大的单个文件字节数。
- client_header_buffer_size用于指定来自客户端请求头的headerbuffer大小。对于大多数请求，1KB的缓冲区大小已经足够，如果自定义了消息头或有更大的cookie，可以增加缓冲区大小。这里设置为32KB。
- large_client_header_buffers用来指定客户端请求中较大的消息头的缓存最大数量和大小，“4”为个数，“128K”为大小，最大缓存为4个128KB。
- sendfile参数用于开启高效文件传输模式。将tcp_nopush和tcp_nodely两个指令设置为on，用于防止网络阻塞。
- keepalive_timeout用于设置客户端连接保持活动的超时时间。在超过这个时间之后，服务器会关闭该连接。
- client_header_timeout用于设置客户端请求头读取超时时间。如果超过这个时间，客户端还没有发送任何数据，Nginx将返回“Request time out（408）”错误。
- client_body_timeout用于设置客户端请求主体读取超时时间，默认值为60。如果超过这个时间，客户端还没有发送任何数据，Nginx将返回“Request time out（408）”错误。

- send_timeout 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。

3. HttpGzip 模块配置

下面配置 Nginx 的 HttpGzip 模块。这个模块支持在线实时压缩输出数据流。要查看是否安装了此模块，需要使用下面的命令：

```
[root@localhost conf]# /opt/nginx/sbin/nginx  -V
nginx version: nginx/0.7.65
configure arguments: --with-http_stub_status_module --with-http_gzip_static_module
    --prefix=/opt/nginx
```

通过 /opt/nginx/sbin/nginx -V 命令可以查看安装 Nginx 时的编译选项。由输出可知，我们已经安装了 HttpGzip 模块。

下面是 HttpGzip 模块在 Nginx 配置中的相关属性设置：

```
gzip  on;
gzip_min_length  1k;
gzip_buffers     4  16k;
gzip_http_version  1.1;
gzip_comp_level  2;
gzip_types  text/plain application/x-javascript text/css application/xml;
gzip_vary  on;
```

- gzip 用于设置开启或者关闭 gzip 模块，“gzip on”表示开启 gzip 压缩，实时压缩输出数据流。
- gzip_min_length 用于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，不管页面多大都进行压缩。建议设置成大于 1K 的字节数，小于 1K 可能会越压越大。
- gzip_buffers 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 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 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 用来指定压缩的类型，无论是否指定，“text/html”类型总是会被压缩的。
- gzip_vary 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。

4. 负载均衡配置

下面设定负载均衡的服务器列表。

```
upstream ixdba.net{
```

```
ip_hash;
server 192.168.12.133:80;
server 192.168.12.134:80  down;
server 192.168.12.135:8009  max_fails=3  fail_timeout=20s;
server 192.168.12.136:8080;
}
```

upstream 是 Nginx 的 HTTP Upstream 模块，这个模块通过一个简单的调度算法来实现客户端 IP 到后端服务器的负载均衡。在上面的设定中，通过 upstream 指令指定了一个负载均衡器的名称 ixdba.net。这个名称可以任意指定，在后面需要用到的地方直接调用即可。

Nginx 的负载均衡模块目前支持 4 种调度算法，下面进行分别介绍，其中后两项属于第三方调度算法。

- 轮询（默认）。每个请求按时间顺序逐一分配到不同的后端服务器，如果后端某台服务器宕机，故障系统被自动剔除，使用户访问不受影响。
- Weight。指定轮询权值，Weight 值越大，分配到的访问机率越高，主要用于后端每个服务器性能不均的情况下。
- ip_hash。每个请求按访问 IP 的 hash 结果分配，这样来自同一个 IP 的访客固定访问一个后端服务器，有效解决了动态网页存在的 session 共享问题。
- fair。这是比上面两个更加智能的负载均衡算法。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 本身是不支持 fair 的，如果需要使用这种调度算法，必须下载 Nginx 的 upstream_fair 模块。
- url_hash。此方法按访问 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。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 软件包。

在 HTTP Upstream 模块中，可以通过 server 指令指定后端服务器的 IP 地址和端口，同时还可以设定每个后端服务器在负载均衡调度中的状态。常用的状态有：

- down，表示当前的 server 暂时不参与负载均衡。
- backup，预留的备份机器。当其他所有的非 backup 机器出现故障或者忙的时候，才会请求 backup 机器，因此这台机器的压力最轻。
- max_fails，允许请求失败的次数，默认为 1。当超过最大次数时，返回 proxy_next_upstream 模块定义的错误。
- fail_timeout，在经历了 max_fails 次失败后，暂停服务的时间。max_fails 可以和 fail_timeout 一起使用。

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。

5. server 虚拟主机配置

下面介绍对虚拟主机的配置。建议将对虚拟主机进行配置的内容写进另外一个文件，然后通过 include 指令包含进来，这样更便于维护和管理。

```
server{
listen          80;
server_name     192.168.12.188  www.ixdba.net;
index index.html index.htm index.jsp;
root  /web/wwwroot/www.ixdba.net
charset gb2312;
```

server 标志定义虚拟主机开始；listen 用于指定虚拟主机的服务器端口；server_name 用来指定 IP 地址或者域名，多个域名之间用空格分开；index 用于设定访问的默认首页地址；root 指令用于指定虚拟主机的网页根目录，这个目录可以是相对路径，也可以是绝对路径；charset 用于设置网页的默认编码格式。

```
access_log  logs/www.ixdba.net.access.log  main;
```

access_log 用来指定此虚拟主机的访问日志存放路径。最后的 main 用于指定访问日志的输出格式。

6. URL 匹配配置

URL 地址匹配是 Nginx 配置中最灵活的部分。location 支持正则表达式匹配，也支持条件判断匹配，用户可以通过 location 指令实现 Nginx 对动、静态网页的过滤处理。

以下这段设置是通过 location 指令来对网页 URL 进行分析处理，所有扩展名为 .gif、.jpg、.jpeg、.png、.bmp、.swf 的静态文件都交给 Nginx 处理，而 expires 用来指定静态文件的过期时间，这里是 30 天。

```
location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$  {
                    root    /web/wwwroot/www.ixdba.net;
                    expires 30d;
        }
```

以下这段设置是将 upload 和 html 下的所有文件都交给 Nginx 来处理，当然，upload 和 html 目录包含在 /web/wwwroot/www.ixdba.net 目录中。

```
        location ~ ^/(upload|html)/  {
                    root    /web/wwwroot/www.ixdba.net;
                    expires 30d;
        }
```

在最后这段设置中，location 是对此虚拟主机下动态网页的过滤处理，也就是将所有以 .jsp 为后缀的文件都交给本机的 8080 端口处理。

```
        location ~ .*.jsp$ {
                    index index.jsp;
                    proxy_pass http://localhost:8080;
            }
```

7. StubStatus模块配置

StubStatus模块能够获取Nginx自上次启动以来的工作状态，此模块非核心模块，需要在Nginx编译安装时手工指定才能使用。

以下指令指定启用获取Nginx工作状态的功能。

```
location /NginxStatus {
                stub_status            on;
                access_log             logs/NginxStatus.log;
                auth_basic             "NginxStatus";
                auth_basic_user_file   ../htpasswd;
}
```

stub_status为“on”表示启用StubStatus的工作状态统计功能；access_log用来指定StubStatus模块的访问日志文件；auth_basic是Nginx的一种认证机制；auth_basic_user_file用来指定认证的密码文件。由于Nginx的auth_basic认证采用的是与Apache兼容的密码文件，因此需要用Apache的htpasswd命令来生成密码文件。例如要添加一个webadmin用户，可以使用下面的方式生成密码文件：

```
/usr/local/apache/bin/htpasswd -c  /opt/nginx/conf/htpasswd webadmin
```

会得到以下提示信息：

```
New password:
```

输入密码之后，系统会要求再次输入密码，确认之后添加用户成功。

要查看Nginx的运行状态，可以输入http://ip/ NginxStatus，然后输入刚刚创建的用户名和密码就可以看到如下信息：

```
Active connections: 1
server accepts handled requests
 393411 393411 393799
Reading: 0 Writing: 1 Waiting: 0
```

Active connections表示当前活跃的连接数，第三行的3个数字表示Nginx当前总共处理了393411个连接，成功创建了393 411次握手，总共处理了393 799个请求。最后一行的Reading表示Nginx读取到客户端Header信息数；Writing表示Nginx返回给客户端的Header信息数；Waiting表示Nginx已经处理完、正在等候下一次请求指令时的驻留连接数。

在最后这段设置中，设置了虚拟主机的错误信息返回页面，通过error_page指令可以定制各种错误信息的返回页面。在默认情况下，Nginx会在主目录的html目录中查找指定的返回页面。特别需要注意的是，这些错误信息的返回页面大小一定要超过512KB，否则会被IE浏览器替换为IE默认的错误页面。

```
error_page  404              /404.html;
error_page   500 502 503 504  /50x.html;
location = /50x.html {
```

```
            root    html;
        }
    }
    }
```

1.4.4　Nginx 的启动、关闭和平滑重启

在完成对 nginx.conf 文件的配置后，就可以启动服务了。Nginx 自身提供了一些用于日常维护的命令，下面进行详细的介绍。

1. Nginx 基本信息检查

（1）检查 Nginx 配置文件的正确性

Nginx 提供的配置文件调试功能非常有用，可以快速定位配置文件存在的问题。执行如下命令可检测配置文件的正确性：

```
/opt/nginx/sbin/nginx -t 或者
/opt/nginx/sbin/nginx -t -c /opt/nginx/conf/nginx.conf
```

其中，“-t”参数用于检查配置文件是否正确，但并不执行，“-c”参数用于指定配置文件路径，如果不指定配置文件路径，Nginx 默认会在安装时指定的安装目录下查找 conf/nginx.conf 文件。

如果检测结果显示如下信息，说明配置文件正确。

```
the configuration file /opt/nginx/conf/nginx.conf syntax is ok
configuration file /opt/nginx/conf/nginx.conf test is successful
```

（2）显示 Nginx 的版本以及相关编译信息

在命令行执行以下命令可以显示安装 Nginx 的版本信息：

```
/opt/nginx/sbin/nginx -v
```

执行以下命令可显示安装的 Nginx 版本和相关编译信息：

```
/opt/nginx/sbin/nginx -V
```

上述命令不但显示 Nginx 的版本信息，同时显示 Nginx 在编译时指定的相关模块信息。

2. Nginx 的启动、关闭与重启

Nginx 对进程的控制能力非常强大，可以通过信号指令控制进程。常用的信号有：

- QUIT，表示处理完当前请求后，关闭进程。
- HUP，表示重新加载配置，也就是关闭原有的进程，并开启新的工作进程。此操作不会中断用户的访问请求，因此可以通过此信号平滑地重启 Nginx。
- USR1，用于 Nginx 的日志切换，也就是重新打开一个日志文件，例如每天要生成一个新的日志文件时，可以使用这个信号来控制。
- USR2，用于平滑升级可执行程序。

❑ WINCH，从容关闭工作进程。

（1）Nginx 的启动

Nginx 的启动非常简单，只需输入如下命令：

```
/opt/nginx/sbin/nginx
```

即可完成 Nginx 的启动。Nginx 启动后，可以通过如下命令查看 Nginx 的启动进程：

```
[root@localhost logs]# ps -ef|grep nginx
root     16572     1  0 11:14 ?    00:00:00 nginx: master process /opt/nginx/sbin/nginx
nobody   16591 16572  0 11:15 ?    00:00:00 nginx: worker process
nobody   16592 16572  0 11:15 ?    00:00:00 nginx: worker process
nobody   16593 16572  0 11:15 ?    00:00:00 nginx: worker process
nobody   16594 16572  0 11:15 ?    00:00:00 nginx: worker process
```

（2）Nginx 的关闭

如果要关闭 Nginx 进程，可以使用如下命令：

```
kill -XXX pid
```

其中，XXX 就是信号名，pid 是 Nginx 的进程号，可以通过如下两个命令获取：

```
ps -ef | grep "nginx: master process" | grep -v "grep" | awk -F ' ' '{print $2}'
cat /opt/nginx/logs/nginx.pid
```

（3）Nginx 的平滑重启

要不间断服务地重新启动 Nginx，可以使用如下命令：

```
kill -HUP `cat /opt/nginx/logs/nginx.pid `
```

1.5 Nginx 常用配置实例

Nginx 作为一个 HTTP 服务器，在功能实现方面和性能方面都表现得非常卓越，完全可以与 Apache 相媲美，几乎可以实现 Apache 的所有功能。下面就介绍一些 Nginx 常用的配置实例，具体包含虚拟主机配置、负载均衡配置、防盗链配置以及日志管理等。

1.5.1 虚拟主机配置实例

下面在 Nginx 中创建 3 个虚拟主机，需要说明的是，这里仅仅列出了虚拟主机的配置部分。

```
http {
    server {
    listen          80;
    server_name     www.domain1.com;
    access_log      logs/domain1.access.log main;
    location / {
    index index.html;
```

```
    root   /web/www/domain1.com/htdocs;
    }
  }
    server {
    listen          80;
    server_name     www.domain2.com;
    access_log      logs/domain2.access.log main;
    location / {
    index index.html;
    root   /web/www/domain2.com/htdocs;
    }
  }
  include    /opt/nginx/conf/vhosts/www.domain2.com.conf;
}
```

这里用到了 include 指令，其中 /opt/nginx/conf/vhosts/www.domain2.com.conf 的内容如下：

```
server {
    listen          80;
    server_name     www.domain3.com;
    access_log      logs/domain3.access.log main;
    location / {
    index index.html;
    root   /web/www/domain3.com/htdocs;
    }
  }
```

1.5.2　负载均衡配置实例

下面通过 Nginx 的反向代理功能配置一个 Nginx 负载均衡服务器。后端有 3 个服务节点，用于提供 Web 服务，通过 Nginx 的调度实现 3 个节点的负载均衡。

```
http
{
  upstream  myserver {
    server    192.168.12.181:80 weight=3 max_fails=3 fail_timeout=20s;
    server    192.168.12.182:80 weight=1 max_fails=3 fail_timeout=20s;
    server    192.168.12.183:80 weight=4 max_fails=3 fail_timeout=20s;
  }

  server
  {
      listen          80;
      server_name  www.domain.com 192.168.12.189;
      index index.htm index.html;
      root   /ixdba/web/wwwroot;

location / {
      proxy_pass http://myserver;
      proxy_next_upstream http_500 http_502 http_503 error timeout invalid_header;
```

```
        include     /opt/nginx/conf/proxy.conf;
        }
      }
}
```

在上面这个配置实例中，先定义了一个负载均衡组 myserver，然后在 location 部分通过“proxy_pass http://myserver”实现负载调度功能，其中 proxy_pass 指令用来指定代理的后端服务器地址和端口，地址可以是主机名或者 IP 地址，也可以是通过 upstream 指令设定的负载均衡组名称。

proxy_next_upstream 用来定义故障转移策略，当后端服务节点返回 500、502、503、504 和执行超时等错误时，自动将请求转发到 upstream 负载均衡组中的另一台服务器，实现故障转移。最后通过 include 指令包含进来一个 proxy.conf 文件。

其中 /opt/nginx/conf/proxy.conf 的内容如下：

```
proxy_redirect off;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
client_body_buffer_size 128k;
proxy_connect_timeout 90;
proxy_send_timeout 90;
proxy_read_timeout 90;
proxy_buffer_size  4k;
proxy_buffers 4 32k;
proxy_busy_buffers_size 64k;
proxy_temp_file_write_size 64k;
```

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 proxy.conf 文件中每个选项代表的含义。

- proxy_set_header：设置由后端的服务器获取用户的主机名或真实 IP 地址，以及代理者的真实 IP 地址。
- client_body_buffer_size：用于指定客户端请求主体缓冲区大小，可以理解为先保存到本地再传给用户。
- proxy_connect_timeout：表示与后端服务器连接的超时时间，即发起握手等候响应的超时时间。
- proxy_send_timeout：表示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。

1.5.3　防盗链配置实例

Nginx 的防盗链功能也非常强大。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。请看下面的这个实例：

```
location ~* \.(jpg|gif|png|swf|flv|wma|wmv|asf|mp3|mmf|zip|rar)$ {
            valid_referers none blocked *.ixdba1.net ixdba1.net;
            if ($invalid_referer) {
            rewrite ^/ http://www.ixdba.net/img/error.gif;
            #return 403;
                        }
            }
            location /images {
            root /opt/nginx/html;
            valid_referers none blocked *.ixdba1.net ixdba1.net;
            if ($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的设定。

“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。

“*.ixdba1.net ixdba1.net”表示这个请求可以正常访问上面指定的文件资源。

if{} 中的内容的意思是：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。

要做更加复杂的防盗链处理，可以使用 Nginx 的 HttpAccessKeyModule，通过这个模块可以实现功能更强大的防盗链处理。更详细的内容可参考官方文档。

1.5.4　日志分割配置实例

Nginx 没有类似 Apache 的 cronolog 日志分割处理的功能，但是，可以通过 Nginx 的信号控制功能的脚本来实现日志的自动切割。请看下面的一个实例。

Nginx 对日志进行处理的脚本。

```
#/bin/bash
savepath_log='/home/nginx/logs'
nglogs='/opt/nginx/logs'

mkdir -p  $savepath_log/$(date +%Y)/$(date +%m)
mv $nglogs/access.log  $savepath_log/$(date +%Y)/$(date +%m)/access.$(date +%Y%m%d).log
mv  $nglogs/error.log  $savepath_log/$(date +%Y)/$(date +%m)/error.$(date +%Y%m%d).log
kill -USR1 `cat /opt/nginx/logs/nginx.pid`
```

将这段脚本保存后加入到 Linux 的 crontab 守护进程，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了。

其中，变量 savepath_log 指定分割后的日志存放的路径，而变量 nglogs 指定 Nginx 日志文件的存放路径。最后一行，通过 Nginx 的信号“USR1”实现了日志的自动切换功能。

1.6 Nginx 性能优化技巧

1.6.1 编译安装过程优化

1. 减小 Nginx 编译后的文件大小

在编译 Nginx 时，默认以 debug 模式进行，而在 debug 模式下会插入很多跟踪和 ASSERT 之类的信息，编译完成后，一个 Nginx 要有好几兆字节。而在编译前取消 Nginx 的 debug 模式，编译完成后 Nginx 只有几百千字节。因此可以在编译之前，修改相关源码，取消 debug 模式。具体方法如下：

在 Nginx 源码文件被解压后，找到源码目录下的 auto/cc/gcc 文件，在其中找到如下几行：

```
# debug
CFLAGS=" $CFLAGS -g"
```

注释掉或删掉这两行，即可取消 debug 模式。

2. 为特定的 CPU 指定 CPU 类型编译优化

在编译 Nginx 时，默认的 GCC 编译参数是“-O”，要优化 GCC 编译，可以使用以下两个参数：

```
--with-cc-opt='-O3'
--with-cpu-opt=CPU  # 为特定的 CPU 编译，有效的值包括：pentium, pentiumpro, pentium3,
                    # pentium4, athlon, opteron, amd64, sparc32, sparc64, ppc64
```

要确定 CPU 类型，可以通过如下命令：

```
[root@localhost home]#cat /proc/cpuinfo | grep "model name"
```

1.6.2　利用 TCMalloc 优化 Nginx 的性能

TCMalloc 的全称为 Thread-Caching Malloc，是谷歌开发的开源工具 google-perftools 中的一个成员。与标准的 glibc 库的 Malloc 相比，TCMalloc 库在内存分配效率和速度上要高很多，这在很大程度上提高了服务器在高并发情况下的性能，从而降低了系统的负载。下面简单介绍如何为 Nginx 添加 TCMalloc 库支持。

要安装 TCMalloc 库，需要安装 libunwind（32 位操作系统不需要安装）和 google-perftools 两个软件包，libunwind 库为基于 64 位 CPU 和操作系统的程序提供了基本函数调用链和函数调用寄存器功能。下面介绍利用 TCMalloc 优化 Nginx 的具体操作过程。

1. 安装 libunwind 库

可以从 http://download.savannah.gnu.org/releases/libunwind 下载相应的 libunwind 版本，这里下载的是 libunwind-0.99-alpha.tar.gz。安装过程如下：

```
[root@localhost home]#tar zxvf libunwind-0.99-alpha.tar.gz
[root@localhost home]# cd libunwind-0.99-alpha/
[root@localhost libunwind-0.99-alpha]#CFLAGS=-fPIC ./configure
[root@localhost libunwind-0.99-alpha]#make CFLAGS=-fPIC
[root@localhost libunwind-0.99-alpha]#make CFLAGS=-fPIC install
```

2. 安装 google-perftools

可以从 http://google-perftools.googlecode.com 下载相应的 google-perftools 版本，这里下载的是 google-perftools-1.8.tar.gz。安装过程如下：

```
[root@localhost home]#tar zxvf google-perftools-1.8.tar.gz
[root@localhost home]#cd google-perftools-1.8/
[root@localhost google-perftools-1.8]# ./configure
[root@localhost google-perftools-1.8]#make && make install
[root@localhost google-perftools-1.8]#echo "/usr/local/lib" > /etc/ld.so.conf.d/
    usr_local_lib.conf
[root@localhost google-perftools-1.8]# ldconfig
```

至此，google-perftools 安装完成。

3. 重新编译 Nginx

为了使 Nginx 支持 google-perftools，需要在安装过程中添加“–with-google_perftools_module”选项重新编译 Nginx。安装代码如下：

```
[root@localhostnginx-0.7.65]#./configure \
>--with-google_perftools_module --with-http_stub_status_module  --prefix=/opt/nginx
[root@localhost nginx-0.7.65]#make
[root@localhost nginx-0.7.65]#make install
```

到这里 Nginx 安装完成。

4. 为 google-perftools 添加线程目录

创建一个线程目录，这里将文件放在 /tmp/tcmalloc 下。操作如下：

```
[root@localhost home]#mkdir /tmp/tcmalloc
[root@localhost home]#chmod 0777 /tmp/tcmalloc
```

5. 修改 Nginx 主配置文件

修改 nginx.conf 文件，在 pid 这行的下面添加如下代码：

```
#pid        logs/nginx.pid;
google_perftools_profiles /tmp/tcmalloc;
```

接着，重启 Nginx 即可完成 google-perftools 的加载。

6. 验证运行状态

为了验证 google-perftools 已经正常加载，可通过如下命令查看：

```
[root@ localhost home]# lsof -n | grep tcmalloc
nginx      2395 nobody    9w  REG    8,8        0    1599440 /tmp/tcmalloc.2395
nginx      2396 nobody   11w REG    8,8        0    1599443 /tmp/tcmalloc.2396
nginx      2397 nobody   13w REG   8,8         0    1599441  /tmp/tcmalloc.2397
nginx     2398 nobody    15w REG   8,8         0   1599442 /tmp/tcmalloc.2398
```

由于在 Nginx 配置文件中设置 worker_processes 的值为 4，因此开启了 4 个 Nginx 线程，每个线程会有一行记录。每个线程文件后面的数字值就是启动的 Nginx 的 pid 值。

至此，利用 TCMalloc 优化 Nginx 的操作完成。

1.6.3 Nginx 内核参数优化

内核参数的优化，主要是在 Linux 系统中针对 Nginx 应用而进行的系统内核参数优化。下面给出一个优化实例以供参考。

```
net.ipv4.tcp_max_tw_buckets = 6000
net.ipv4.ip_local_port_range = 1024 65000
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_syncookies = 1
net.core.somaxconn = 262144
net.core.netdev_max_backlog = 262144
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_synack_retries = 1
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_fin_timeout = 1
net.ipv4.tcp_keepalive_time = 30
```

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@ localhost home]#/sbin/sysctl -p
```

下面对实例中选项的含义进行介绍：

- net.ipv4.tcp_max_tw_buckets 选项用来设定 timewait 的数量，默认是 180 000，这里设为 6000。
- net.ipv4.ip_local_port_range 选项用来设定允许系统打开的端口范围。
- net.ipv4.tcp_tw_recycle 选项用于设置启用 timewait 快速回收。
- net.ipv4.tcp_tw_reuse 选项用于设置开启重用，允许将 TIME-WAIT sockets 重新用于新的 TCP 连接。
- net.ipv4.tcp_syncookies 选项用于设置开启 SYN Cookies，当出现 SYN 等待队列溢出时，启用 cookies 进行处理。
- net.core.somaxconn 选项的默认值是 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者重传，因此，需要结合并发请求数来调节此值。
- net.core.netdev_max_backlog 选项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的最大数目。
- net.ipv4.tcp_max_orphans 选项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。如果超过这个数字，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，更多的情况下应该增加这个值。
- net.ipv4.tcp_max_syn_backlog 选项用于记录那些尚未收到客户端确认信息的连接请求的最大值。对于有 128MB 内存的系统而言，此参数的默认值是 1024，对小内存的系统则是 128。
- 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。
- net.ipv4.tcp_syn_retries 选项表示在内核放弃建立连接之前发送 SYN 包的数量。
- net.ipv4.tcp_fin_timeout 选项决定了套接字保持在 FIN-WAIT-2 状态的时间。默认值是 60 秒。正确设置这个值非常重要，有时即使一个负载很小的 Web 服务器，也会出现大量的死套接字而产生内存溢出的风险。
- net.ipv4.tcp_syn_retries 选项表示在内核放弃建立连接之前发送 SYN 包的数量。

如果发送端要求关闭套接字，net.ipv4.tcp_fin_timeout 选项决定了套接字保持在 FIN-WAIT-2 状态的时间。接收端可以出错并永远不关闭连接，甚至意外宕机。

- net.ipv4.tcp_fin_timeout 的默认值是 60 秒。需要注意的是，即使一个负载很小的 Web 服务器，也会出现因为大量的死套接字而产生内存溢出的风险。FIN-WAIT-2 的危险性比 FIN-WAIT-1 要小，因为它最多只能消耗 1.5KB 的内存，但是其生存期长些。
- net.ipv4.tcp_keepalive_time 选项表示当 keepalive 启用的时候，TCP 发送 keepalive 消息的频度。默认值是 2（单位是小时）。

1.7 实战 Nginx 与 PHP（FastCGI）的安装、配置与优化

1.7.1 什么是 FastCGI

FastCGI 是一个可伸缩地、高速地在 HTTP server 和动态脚本语言间通信的接口。多数流行的 HTTP server 都支持 FastCGI，包括 Apache、Nginx 和 lighttpd 等。同时，FastCGI 也被许多脚本语言支持，其中就有 PHP。

FastCGI 是从 CGI 发展改进而来的。传统 CGI 接口方式的主要缺点是性能很差，因为每次 HTTP 服务器遇到动态程序时都需要重新启动脚本解析器来执行解析，然后将结果返回给 HTTP 服务器。这在处理高并发访问时几乎是不可用的。另外传统的 CGI 接口方式安全性也很差，现在已经很少使用了。

FastCGI 接口方式采用 C/S 结构，可以将 HTTP 服务器和脚本解析服务器分开，同时在脚本解析服务器上启动一个或者多个脚本解析守护进程。当 HTTP 服务器每次遇到动态程序时，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。

1.7.2 Nginx+FastCGI 运行原理

Nginx 不支持对外部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（这个 socket 可以是文件 socket，也可以是 ip socket）。为了调用 CGI 程序，还需要一个 FastCGI 的 wrapper（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 socket 上，如端口或者文件 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者外部程序处理脚本并读取返回数据；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，如图 1-3 所示。

1.7.3 spawn-fcgi 与 PHP-FPM

前面介绍过，FastCGI 接口方式在脚本解析服务器上启动一个或者多个守护进程对动态脚本进行解析，这些进程就是 FastCGI 进程管理器，或者称为 FastCGI 引擎。 spawn-fcgi 与 PHP-FPM 就是支持 PHP 的两个 FastCGI 进程管理器。

下面简单介绍 spawn-fcgi 与 PHP-FPM 的异同。

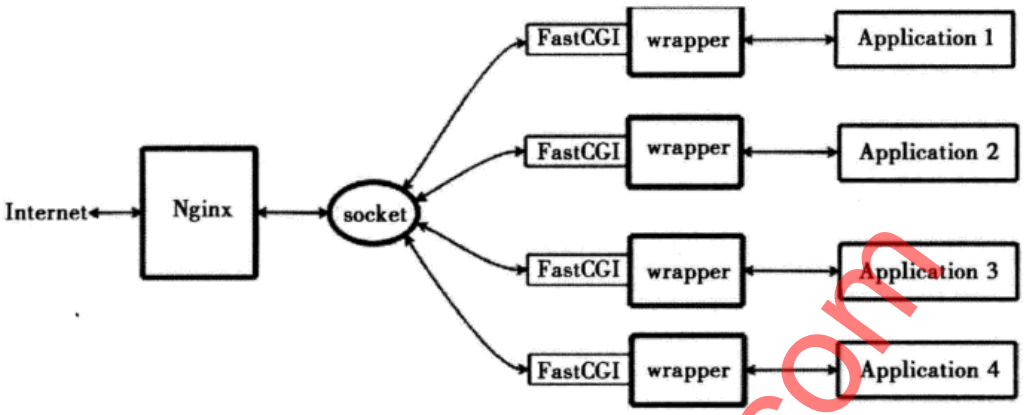

图 1-3　Nginx+FastCGI 运行过程

spawn-fcgi 是 HTTP 服务器 lighttpd 的一部分，目前已经独立成为一个项目，一般与 lighttpd 配合使用来支持 PHP。但是 ligttpd 的 spwan-fcgi 在高并发访问的时候，会出现内存泄漏甚至自动重启 FastCGI 的问题。

Nginx 是个轻量级的 HTTP server，必须借助第三方的 FastCGI 处理器才可以对 PHP 进行解析，因此 Nginx+spawn-fcgi 的组合也可以实现对 PHP 的解析，这里不过多讲述。

PHP-FPM 也是一个第三方的 FastCGI 进程管理器，它是作为 PHP 的一个补丁来开发的，在安装的时候也需要和 PHP 源码一起编译，也就是说 PHP-FPM 被编译到 PHP 内核中，因此在处理性能方面更加优秀。同时 PHP-FPM 在处理高并发方面也比 spawn-fcgi 引擎好很多，因此，推荐使用 Nginx+PHP/PHP-FPM 这个组合对 PHP 进行解析。

FastCGI 的主要优点是把动态语言和 HTTP Server 分离开来，所以 Nginx 与 PHP/PHP-FPM 经常被部署在不同的服务器上，以分担前端 Nginx 服务器的压力，使 Nginx 专一处理静态请求和转发动态请求，而 PHP/PHP-FPM 服务器专一解析 PHP 动态请求。

1.7.4　PHP 与 PHP-FPM 的安装及优化

1. 下载安装包

从 www.php.net 官方网站下载 PHP 源码包，这里下载的是稳定版 php-5.2.13.tar.gz。

从 http://php-fpm.org/downloads/ 下载对应的 PHP-FPM 源码包，这里下载的是 php-5.2.13-fpm-0.5.13.diff.gz。

需要注意，在下载软件包版本时，尽量使 PHP 和 PHP-FPM 版本一致，如果版本之间相差太大，可能会出现兼容的问题。

2. 配置安装环境

安装 PHP 需要下面软件包的支持，如果没有安装，请自行安装。

```
gcc gcc-c++ libxml2 libxml2-devel autoconf libjpeg libjpeg-devel libpng libpng-
```

```
devel freetype freetype-devel  zlib zlib-devel glibc glibc-devel glib2 glib2-
devel
```

由于各个 Linux 系统版本有不确定性，读者也可以在安装 PHP 过程中，根据错误提示信息，安装对应的软件库。

3. 开始编译安装 PHP 和 PHP-FPM

编译安装 PHP 和 PHP-FPM 很简单，下面是安装过程：

```
[root@localhost local]#tar zxvf php-5.2.13.tar.gz
[root@localhost local]#gzip -cd php-5.2.13-fpm-0.5.13.diff.gz | patch -d php-
    5.2.13 -p1
[root@localhost local]#cd php-5.2.13
[root@localhost php-5.2.13]#./configure  --prefix=/usr/local/php --enable-fastcgi
    --enable-fpm
[root@localhost php-5.2.13]#make
[root@localhost php-5.2.13]#make install
[root@localhost php-5.2.13]cp php.ini-dist /usr/local/php/lib/php.ini
```

其中，第二步将 PHP-FPM 作为补丁加入 PHP 源码中。

在“./configure”编译选项中，指定将 PHP 安装到 /usr/local 下；“--enable-fastcgi”是启用对 PHP 的 FastCGI 支持；“--enable-fpm”是激活对 FastCGI 模式的 fpm 支持。

在编译 PHP 时可以加入很多编译选项，但是这里为了介绍 PHP 的 FastCGI 功能没有加入更多的编译选项。

4. 配置与优化 PHP-FPM

PHP 的全局配置文件是 php.ini，在上面的步骤中，已经将此文件复制到了 /usr/local/php/lib/php.ini 下。可以根据每个应用需求的不同，对 php.ini 进行相应的配置。

下面重点介绍 PHP-FPM 引擎的配置文件。

根据上面指定的安装路径，PHP-FPM 的默认配置文件为 /usr/local/php/etc/php-fpm.conf。

php-fpm.conf 是一个 XML 格式的纯文本文件，其内容很容易看明白。这里重点介绍几个重要的配置标签。

标签 listen_address 是配置 FastCGI 进程监听的 IP 地址以及端口，默认是 127.0.0.1:9000。

```
<value name="listen_address">127.0.0.1:9000</value>
```

标签 display_errors 用来设置是否显示 PHP 错误信息，默认是 0，不显示错误信息，设置为 1 可以显示 PHP 错误信息。

```
<value name="display_errors">0</value>
```

标签 user 和 group 用于设置运行 FastCGI 进程的用户和用户组。需要注意的是，这里指定的用户和用户组要和 Nginx 配置文件中指定的用户和用户组一致。

```
<value name="user">nobody</value>
<value name="group">nobody</value>
```

标签 max_children 用于设置 FastCGI 的进程数。根据官方建议，小于 2GB 内存的服务器，可以只开启 64 个进程，4GB 以上内存的服务器可以开启 200 个进程。

```
<value name="max_children">5</value>
```

标签 request_terminate_timeout 用于设置 FastCGI 执行脚本的时间。默认是 0 秒，也就是无限地执行下去，可以根据情况对其进行修改。

```
<value name="request_terminate_timeout">0s</value>
```

标签 rlimit_files 用于设置 PHP-FPM 对打开文件描述符的限制，默认值为 1024。这个标签的值必须和 Linux 内核打开文件数关联起来，例如，要将此值设置为 65 535，就必须在 Linux 命令行执行“ulimit -HSn 65536”。

```
<value name="rlimit_files">1024</value>
```

标签 max_requests 指明了每个 children 最多处理多少个请求后便会被关闭，默认的设置是 500。

```
<value name="max_requests">500</value>
```

标签 allowed_clients 用于设置允许访问 FastCGI 进程解析器的 IP 地址。如果不在这里指定 IP 地址，将无法接受 Nginx 转发过来的 PHP 解析请求。

```
<value name="allowed_clients">127.0.0.1</value>
```

5. 管理 FastCGI 进程

在配置完 PHP_FPM 后，就可以启动 FastCGI 进程了。启动 FastCGI 进程有以下两种方式：

```
/usr/local/php/bin/php-cgi --fpm
```

或者

```
/usr/local/php/sbin/php-fpm start
```

建议采用第二种方式启动 FastCGI 进程。

/usr/local/php/sbin/php-fpm 还有其他参数，具体为 start|stop|quit|restart|reload|logrotate。

每个启动参数的含义如下：

- start，启动 PHP 的 FastCGI 进程。
- stop，强制终止 PHP 的 FastCGI 进程。
- quit，平滑终止 PHP 的 FastCGI 进程。
- restart，重启 PHP 的 FastCGI 进程。
- reload，重新加载 PHP 的 php.ini。
- logrotate，重新启用 log 文件。

reload 是个很重要的参数，它可以在 PHP 的 FastCGI 进程不中断的情况下重新加载改动过的 php.ini，因此通过 PHP_FPM 可以平滑地变更 FastCGI 模式下的 PHP 设置。

在 FastCGI 进程启动后，其监听的 IP 地址和端口也随即启动，可以通过 ps 和 netstat 查看相关信息。

```
[root@localhost php]# netstat -antl|grep 9000
tcp        0      0 127.0.0.1:9000              0.0.0.0:*                   LISTEN
[root@localhost php]# ps -ef|grep php-cgi
root       3567      1  0  17:06 ?                        00:00:00 /usr/local/
    php/bin/php-cgi --fpm --fpm-config /usr/local/php/etc/php-fpm.conf
nobody     3568   3567  0  17:06 ?        00:00:00 /usr/local/php/bin/php-cgi --fpm
    --fpm-config /usr/local/php/etc/php-fpm.conf
nobody     3569   3567  0  17:06 ?        00:00:00 /usr/local/php/bin/php-cgi --fpm
    --fpm-config /usr/local/php/etc/php-fpm.conf
nobody     3570   3567  0  17:06 ?        00:00:00 /usr/local/php/bin/php-cgi --fpm
    --fpm-config /usr/local/php/etc/php-fpm.conf
nobody     3571   3567  0  17:06 ?        00:00:00 /usr/local/php/bin/php-cgi --fpm
    --fpm-config /usr/local/php/etc/php-fpm.conf
nobody     3572   3567  0  17:06 ?        00:00:00 /usr/local/php/bin/php-cgi --fpm
    --fpm-config /usr/local/php/etc/php-fpm.conf
root       3583   3524  0 17:09 pts/1    00:00:00 grep php-cgi
```

1.7.5 配置 Nginx 来支持 PHP

Nginx 的安装特别简单，前面已经对此进行了详细介绍，这里不再进行讲述。下面重点介绍 Nginx 如何通过 PHP_FPM 的 FastCGI 进程对 PHP 进行解析处理。

由于 Nginx 本身不会对 PHP 进行解析，因此要实现 Nginx 对 PHP 的支持，将对 PHP 页面的请求交给 FastCGI 进程监听的 IP 地址及端口。如果把 PHP_FPM 当做动态应用服务器，那么 Nginx 其实就是一个反向代理服务器。Nginx 通过反向代理功能实现对 PHP 的解析，这就是 Nginx 实现 PHP 动态解析的原理。

这里假定 Nginx 的安装目录为 /usr/local，则 Nginx 配置文件的路径为 /usr/local/nginx/conf/nginx.conf。下面是在 Nginx 下支持 PHP 解析的一个虚拟主机配置实例。

```
        server {
    include port.conf;
    server_name www.ixdba.net ixdba.net;

            location / {
    index index.html index.php;
    root /web/www/www.ixdba.net;
    }

    location ~ \.php$ {
                root           html;
                fastcgi_pass   127.0.0.1:9000;
                fastcgi_index  index.php;
                fastcgi_param  SCRIPT_FILENAME  html$fastcgi_script_name;
                include        fastcgi_params;
        }
}
```

通过 location 指令，将所有以 php 为后缀的文件都交给 127.0.0.1:9000 来处理，而这里的 IP 地址和端口就是 FastCGI 进程监听的 IP 地址和端口。

fastcgi_param 指令指定放置 PHP 动态程序的主目录，也就是 $fastcgi_script_name 前面指定的路径，这里是 /usr/local/nginx/html 目录。建议将这个目录与 Nginx 虚拟主机指定的根目录保持一致，当然也可以不一致。

fastcgi_params 文件是 FastCGI 进程的一个参数配置文件，在安装 Nginx 后，会默认生成一个这样的文件。这里通过 include 指令将 FastCGI 参数配置文件包含了进来。

接下来，启动 Nginx 服务。

```
/usr/local/nginx/sbin/nginx
```

到此为止，Nginx+PHP 已经配置完成。

1.7.6　测试 Nginx 对 PHP 的解析功能

这里在 /usr/local/nginx/html 目录下创建一个 phpinfo.php 文件，内容如下：

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

然后通过浏览器访问 http://www.ixdba.net/index.html，默认会在浏览器显示“Welcome to Nginx!”表示 Nginx 正常运行。

接着在浏览器中访问 http://www.ixdba.net/phpinfo.php，如果 PHP 能够正常解析，会出现 PHP 安装配置以及功能列表统计信息。

1.7.7　优化 Nginx 中 FastCGI 参数的实例

在配置完成 Nginx+FastCGI 之后，为了保证 Nginx 下 PHP 环境的高速稳定运行，需要添加一些 FastCGI 优化指令。下面给出一个优化实例，将下面代码添加到 Nginx 主配置文件中的 HTTP 层级。

```
fastcgi_cache_path /usr/local/nginx/fastcgi_cache levels=1:2 keys_zone=TEST:10m
    inactive=5m;
fastcgi_connect_timeout 300;
fastcgi_send_timeout 300;
fastcgi_read_timeout 300;
fastcgi_buffer_size 64k;
fastcgi_buffers 4 64k;
fastcgi_busy_buffers_size 128k;
fastcgi_temp_file_write_size 128k;
fastcgi_cache TEST;
fastcgi_cache_valid 200 302 1h;
fastcgi_cache_valid 301 1d;
fastcgi_cache_valid any 1m;
```

下面对上述代码的含义进行介绍。

第一行代码是为FastCGI缓存指定一个文件路径、目录结构等级、关键字区域存储时间和非活动删除时间。

fastcgi_connect_timeout指定连接到后端FastCGI的超时时间。

fastcgi_send_timeout指定向FastCGI传送请求的超时时间，这个值是已经完成两次握手后向FastCGI传送请求的超时时间。

fastcgi_read_timeout指定接收FastCGI应答的超时时间，这个值是已经完成两次握手后接收FastCGI应答的超时时间。

fastcgi_buffer_size用于指定读取FastCGI应答第一部分需要多大的缓冲区，这个值表示将使用1个64KB的缓冲区读取应答的第一部分（应答头），可以设置为fastcgi_buffers选项指定的缓冲区大小。

fastcgi_buffers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。

fastcgi_busy_buffers_size的默认值是fastcgi_buffers的两倍。

fastcgi_temp_file_write_size表示在写入缓存文件时使用多大的数据块，默认值是fastcgi_buffers的两倍。

fastcgi_cache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。但是开启缓存也会引起很多问题，要视具体情况而定。

fastcgi_cache_valid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，将301应答缓存1天，其他应答均缓存1分钟。

1.8 实战Nginx与Perl、Java的安装与配置

通过前面的介绍，可以对Nginx有了一个比较全面的认识：Nginx本身是一个静态的HTTP服务器和反向代理服务器，它不支持动态页面，所谓的Nginx对动态程序的支持都是通过反向代理功能实现的。下面要讲述的Nginx对Perl和JSP的支持，就是通过Nginx的反向代理功能来完成的。Nginx对Perl和JSP的支持在实现细节上可能有一定差别，但是实现原理是完全一样的。

Nginx的安装这里不再讲述，假定Nginx的安装路径为/usr/local/nginx。

1.8.1　Perl（FastCGI）的安装

1. 获取 wrapper 程序

读者可以从 http://www.nginx.eu/nginx-fcgi/ 上下载 nginx-fcgi.txt 文件，然后将其命名为 nginx-fcgi.pl，并放到 /usr/local/nginx 目录下。nginx-fcgi.pl 是一个用 Perl 脚本写的 wrapper 实例，所以，操作系统必须要安装 Perl 程序以及相关模块。

2. 安装相关的系统支持模块

可以从 http://search.cpan.org 下载所需的相应模块，然后进行安装。

（1）安装 FCGI 模块

```
[root@localhost opt]# tar zxvf FCGI-0.71.tar.gz
[root@localhost opt]# cd FCGI-0.71
[root@localhost FCGI-0.71]# perl Makefile.PL
[root@localhost FCGI-0.71]# make
[root@localhost FCGI-0.71]# make install
```

（2）安装 IO 模块

```
[root@localhost opt]# tar -xvzf IO-1.25.tar.gz
[root@localhost opt]# cd IO-1.25
[root@localhost IO-1.25]# perl Makefile.PL
[root@localhost IO-1.25]# make
[root@localhost IO-1.25]# make install
```

（3）安装 IO-ALL 模块

```
[root@localhost opt]# tar -xvzf IO-All-0.39.tar.gz
[root@localhost opt]# cd IO-ALL-0.39
[root@localhost IO-ALL-0.39]# perl Makefile.PL
[root@localhost IO-ALL-0.39]#make
[root@localhost IO-ALL-0.39]#make install
```

3. 编写 nginx-fcgi 启动脚本

仅仅有 wrapper 文件是不够的，还需要一个脚本来创建 socket、启动 wrapper 以及将 wrapper 和 socket 绑定。下面通过一个 shell 脚本来完成这一系列工作。

```
[root@localhost root]# more nginx-fcgi
#!/bin/bash
nginxroot=/usr/local/nginx

start ()
{
chown nobody.root $nginxroot/logs
echo "$nginxroot/nginx-fcgi.pl -l $nginxroot/logs/nginx-fcgi.log -pid $nginxroot/
    logs/nginx-fcgi.pid  -S  $nginxroot/logs/nginx-fcgi.sock" >>$nginxroot/nginx_
    fcgi.sh
chown nobody.nobody $nginxroot/nginx_fcgi.sh
```

```
chmod 755 $nginxroot/nginx_fcgi.sh
sudo -u nobody $nginxroot/nginx_fcgi.sh
echo "start nginx-fcgi done"
}

stop ()
{
kill $(cat $nginxroot/logs/nginx-fcgi.pid)
rm $nginxroot/logs/nginx-fcgi.pid 2>/dev/null
rm $nginxroot/logs/nginx-fcgi.sock 2>/dev/null
rm $nginxroot/nginx_fcgi.sh 2>/dev/null
echo "stop nginx-fcgi done"
}

case $1 in
stop)
stop
;;

start)
start
;;

restart)
stop
start
;;

*)
echo $"Usage: perl-cgi {start|stop|restart}"
        exit 1
esac
```

在nginx-fcgi中，变量nginxroot用于指定Nginx的安装目录，nginx-fcgi.sock是生成的文件sock，nobody为运行nginx_fcgi进程的用户，这个用户要和运行Nginx的用户一致。

配置完脚本后，将此文件放到/usr/local/nginx目录下，接着通过如下方式管理nginx-fcgi进程：

```
[root@localhost root]#chmod 755 /usr/local/nginx/nginx-fcgi.pl
[root@localhost root]#chmod 755 /usr/local/nginx/nginx-fcgi
[root@localhost root]#/usr/local/nginx/nginx-fcgi start|stop|restart
```

1.8.2 为Nginx添加FCGI支持

修改Nginx配置文件，在server虚拟主机中添加如下配置：

```
location ~ \.cgi$ {
    root      html;
    fastcgi_pass    unix:/usr/local/nginx/logs/nginx-fcgi.sock;
```

```
        fastcgi_index   index.cgi;
        fastcgi_param   SCRIPT_FILENAME  $document_root$fastcgi_script_name;
        include         fastcgi_params;
    }
```

在这个 location 配置中，Nginx 与 FastCGI 的通信方式为 Unix Socket。根据经验，IP Socket 在高并发访问下比 Unix Socket 稳定，但 Unix Socket 速度要比 IP Socket 快。“$document_root”是虚拟主机的根目录，在这里是 /usr/local/nginx/html 目录。

1.8.3　测试 Nginx +Perl(FastCGI)

所有配置工作完成后，即可启动服务了。首先启动 nginx-fcgi 进程，操作如下：

```
/usr/local/nginx/nginx-fcgi start
```

然后启动 nginx 服务。

```
/usr/local/nginx/sbin/nginx
```

下面在 /usr/local/nginx/html 目录下创建一个 test.cgi 的文件。

```
# disable filename globbing
set -f
echo "Content-type: text/plain; charset=iso-8859-1"
echo

echo CGI/1.0 test script report:
echo

echo argc is $#. argv is "$*".
echo

echo SERVER_SOFTWARE = $SERVER_SOFTWARE
echo SERVER_NAME = $SERVER_NAME
echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE
echo SERVER_PROTOCOL = $SERVER_PROTOCOL
echo SERVER_PORT = $SERVER_PORT
echo REQUEST_METHOD = $REQUEST_METHOD
echo REMOTE_ADDR = $REMOTE_ADDR
```

接着通过浏览器访问 test.cgi 文件，如果显示与下面类似的信息，表明 Nginx+Perl 环境搭建成功。

```
CGI/1.0 test script report:
argc is 1. argv is .
SERVER_SOFTWARE = nginx/0.7.65
SERVER_NAME = localhost
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.1
SERVER_PORT = 8000
REQUEST_METHOD = GET
REMOTE_ADDR = 125.76.159.197
```

1.8.4 搭建 Nginx+Java 环境

Apache 对 Java 的支持很灵活，它们的结合度也很高，例如 Apache+Tomcat 和 Apache+resin 等都可以实现对 Java 应用的支持。Apache 一般采用一个内置模块来和 Java 应用服务器打交道。与 Apache 相比，Nginx 在配合 Java 应用服务器方面，耦合度很低，它只能通过自身的反向代理功能来实现与 Java 应用服务器的支持。但这恰恰是 Nginx 的一个优点，耦合度的降低，可以使 Nginx 与 Java 服务器的相互影响降到最低。

接下来通过 Nginx+Tomcat 的实例来讲解 Nginx 对 Java 的支持。Tomcat 在高并发环境下处理动态请求时性能很低，而在处理静态页面更加脆弱。虽然 Tomcat 的最新版本支持 epoll，但是通过 Nginx 来处理静态页面要比通过 Tomcat 处理在性能方面好很多。

Nginx 可以通过以下两种方式来实现与 Tomcat 的耦合：

- 将静态页面请求交给 Nginx，动态请求交给后端 Tomcat 处理。
- 将所有请求都交给后端的 Tomcat 服务器处理，同时利用 Nginx 自身的负载均衡功能进行多台 Tomcat 服务器的负载均衡。

下面通过两个配置实例分别讲述这两种实现 Nginx 与 Tomcat 耦合的方式。

1. 动态页面与静态页面分离的实例

这里假定 Tomcat 服务器的 IP 地址为 192.168.12.130，同时 Tomcat 服务器开放的服务器端口为 8080。Nginx 相关配置代码如下：

```
server {
    listen 80;
    server_name www.ixdba.net;
    root /web/www/html;

location /img/ {
    alias /web/www/html/img/;
}

location ~ (\.jsp)|(\.do)$ {
        proxy_pass http://192.168.12.130:8080;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        client_max_body_size 10m;
        client_body_buffer_size 128k;
        proxy_connect_timeout 90;
        proxy_send_timeout 90;
        proxy_read_timeout 90;
        proxy_buffer_size 4k;
        proxy_buffers 4 32k;
        proxy_busy_buffers_size 64k;
```

```
            proxy_temp_file_write_size 64k;
    }

    }
```

在这个实例中，首先定义了一个虚拟主机 www.ixdba.net，然后通过 location 指令将 /web/www/html/img/ 目录下的静态文件交给 Nginx 来完成。最后一个 location 指令将所有以 .jsp、.do 结尾的文件都交给 Tomcat 服务器的 8080 端口来处理，即 http://192.168.12.130:8080。

需要特别注意的是，在 location 指令中使用正则表达式后，proxy_pass 后面的代理路径不能含有地址链接，也就是不能写成 http://192.168.12.130:8080/，或者类似 http://192.168.12.130:8080/jsp 的形式。在 location 指令不使用正则表达式时，没有此限制。

2. 多个 Tomcat 负载均衡的实例

这里假定有 3 台 Tomcat 服务器，分别开放不同的端口，地址如下：

```
192.168.12.131:8000
192.168.12.132:8080
192.168.12.133:8090
```

Nginx 的相关配置代码如下：

```
upstream mytomcats {
     server 192.168.12.131:8000;
     server 192.168.12.132:8080;
     server 192.168.12.133:8090;
}

server {
     listen 80;
     server_name www.ixdba.net;

location ~* \.(jpg|gif|png|swf|flv|wma|wmv|asf|mp3|mmf|zip|rar)$ {
      root /web/www/html/;
}

location / {
           proxy_pass http://mytomcats;
           proxy_redirect off;
           proxy_set_header Host $host;
           proxy_set_header X-Real-IP $remote_addr;
           proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
           client_max_body_size 10m;
           client_body_buffer_size 128k;
           proxy_connect_timeout 90;
           proxy_send_timeout 90;
           proxy_read_timeout 90;
           proxy_buffer_size 4k;
           proxy_buffers 4 32k;
           proxy_busy_buffers_size 64k;
```

```
            proxy_temp_file_write_size 64k;
    }

    }
```

在这个实例中，先通过 upstream 定义一个负载均衡组，组名为 mytomcats，组的成员就是上面指定的 3 台 Tomcat 服务器；接着通过 server 指令定义一个 www.ixdba.net 的虚拟主机；然后通过 location 指令以正则表达式的方式将指定类型的文件全部交给 Nginx 去处理；最后将其他所有请求全部交给负载均衡组来处理。

这里还有一点需要注意，如果在 location 指令使用正则表达式后再用 alias 指令，Nginx 是不支持的。

1.9 本章小结

本章主要介绍了对高性能 HTTP 服务器 Nginx 的安装、配置、管理和使用，以及 Nginx 在性能优化方面的一些经验和技巧，并通过实例分别演示了 Nginx 与 PHP 整合，Nginx 和 Java、Perl 整合的过程。通过本章的学习，读者能够对 Nginx 有一个清晰的认识，并且可以熟练地配置和管理 Nginx 服务器。

随着 Nginx 知识的普及，相信 Nginx 会越来越受欢迎。如果你还没有使用 Nginx 来搭建 Web 应用系统，不妨现在尝试一下。

第 2 章　高性能 HTTP 加速器 Varnish

本章主要介绍 Varnish 的配置管理和使用技巧。Varnish 是一个开源的反向代理软件和 HTTP 加速器，与传统的 Squid 相比，Varnish 具有性能更高、速度更快、管理更方便等诸多优点，很多大型的运营网站都开始尝试用 Varnish 来替换 Squid，这些都促使 Varnish 迅速发展起来。本章将详细介绍 Varnish 的安装、配置、管理和性能优化等几个方面，并将理论与实践经验贯穿其中。相信阅读完本章，读者就能够熟练使用 Varnish 了。

2.1　初识 Varnish

2.1.1　Varnish 概述

Varnish 是一款高性能且开源的反向代理服务器和 HTTP 加速器，它的开发者 Poul-Henning Kamp 是 FreeBSD 核心的开发人员之一。Varnish 采用全新的软件体系机构，和现在的硬件体系配合紧密。在 1975 年时，储存媒介只有两种：内存与硬盘。而现在计算机系统的内存除了主存外，还包括 CPU 内的 L1、L2，有的还包括 L3 快取，硬盘上也有自己的快取装置，因此 Squid Cache 自行处理数据替换的架构不可能得知这些情况而做到最佳化，但操作系统可以得知这些情况，所以这部分工作应该交给操作系统处理，这就是 Varnish Cache 设计架构。

挪威最大的在线报纸 Verdens Gang(vg.no) 使用 3 台 Varnish 代替了原来的 12 台 Squid，性能比以前更好，这是 Varnish 最成功的应用案例。目前，Varnish 可以在 FreeBSD6.0/7.0、Solaris 和 Linux 2.6 内核上运行。本章主要介绍 Varnish 在 Linux 上的应用。

2.1.2　Varnish 的结构与特点

Varnish 是一个轻量级的 Cache 和反向代理软件。先进的设计理念和成熟的设计框架是 Varnish 的主要特点。现在的 Varnish 总共代码量不大，虽然功能在不断改进，但是还需要继续丰富和加强。下面是 Varnish 的一些特点。

- 基于内存进行缓存，重启后数据将消失。
- 利用虚拟内存方式，I/O 性能好。
- 支持设置 0~60 秒的精确缓存时间。

- VCL 配置管理比较灵活。
- 32 位机器上缓存文件大小为最大 2GB。
- 具有强大的管理功能，例如 top、stat、admin、list 等。
- 状态机设计巧妙，结构清晰。
- 利用二叉堆管理缓存文件，可达到积极删除目的。

2.1.3 Varnish 与 Squid 的对比

说到 Varnish，就不能不提 Squid。Squid 是一个高性能的代理缓存服务器，它和 Varnish 相比较有诸多的异同点，下面进行分析。

下面是 Varnish 与 Squid 之间的相同点。

- 都是一个反向代理服务器。
- 都是开源软件。

下面是它们的不同点，也是 Varnish 的优点。

- Varnish 的稳定性很高。两者在完成相同负荷的工作时，Squid 服务器发生故障的几率要高于 Varnish，因为 Squid 需要经常重启。
- Varnish 访问速度更快。Varnish 采用了“Visual Page Cache”技术，所有缓存数据都直接从内存读取，而 Squid 是从硬盘读取缓存数据，因此 Varnish 在访问速度方面会更快。
- Varnish 可以支持更多的并发连接。因为 Varnish 的 TCP 连接释放要比 Squid 快，所以在高并发连接情况下可以支持更多的 TCP 连接。
- Varnish 可以通过管理端口，使用正则表达式批量清除部分缓存，而 Squid 做不到。

当然，与传统的 Squid 相比，Varnish 也有缺点。

- Varnish 在高并发状态下 CPU、I/O 和内存等资源开销都高于 Squid。
- Varnish 进程一旦挂起、崩溃或者重启，缓存数据都会从内存中完全释放，此时所有请求都会被发送到后端服务器，在高并发情况下，这会给后端服务器造成很大压力。

2.2 开始安装 Varnish

Varnish 的安装非常简单，下面逐步介绍。

2.2.1 安装前的准备

Varnish 安装环境如表 2-1 所示。

表 2-1　Varnish 安装环境

主机名	操作系统	IP 地址
Varnish-server	CentOS release 5.4	192.168.12.246
Web-server	CentOS release 5.4	192.168.12.26

接着，建立 Varnish 用户以及用户组，并且创建 Varnish 缓存目录和日志目录。

```
[root@varnish-server ~]#useradd  -s /sbin/nologin varnish
[root@varnish-server ~]#mkdir /data/varnish/cache
[root@varnish-server ~]#mkdir /data/varnish/log
[root@varnish-server ~]#chown -R varnish:varnish  /data/varnish/cache
[root@varnish-server ~]#chown -R varnish:varnish  /data/varnish/log
```

2.2.2　获取 Varnish 软件

Varnish 的官方网址为 http://varnish-cache.org，这里面有 Varnish 的最新说明文档及版本升级记录，在此网站中可以找到 Varnish 在 SourceForge 中的下载链接。目前，Varnish 的最新版本是 Varnish 2.1.2，下载完成后的包名为 varnish-2.1.2.tar.gz，这里以此版本为例，进行安装配置。

2.2.3　安装 pcre

如果没有安装 pcre，在编译 varnish 2.0 以上版本时，会提示找不到 pcre 库，而 pcre 库是为了兼容正则表达式，所以必须先安装 pcre 库。下面是 pcre 的安装过程。

```
[root@varnish-server ~]#tar zxvf pcre-7.9.tar.gz
[root@varnish-server ~]#cd pcre-7.9/
[root@varnish-server ~]#./configure --prefix=/usr/local/pcre/
[root@varnish-server ~]#make && make install
```

2.2.4　安装 Varnish

这里将 Varnish 安装到 /usr/local/ 目录下，操作如下：

```
[root@varnish-server ~]#tar -zxvf varnish-2.1.2.tar.gz
[root@varnish-server ~]#cd varnish-2.1.2
[root@varnish-server ~]#export PKG_CONFIG_PATH=/usr/local/pcre/lib/pkgconfig
[root@varnish-server ~]#./configure --prefix=/usr/local/varnish \
 >--enable-dependency-trackin
 >--enable-debugging-symbols
 >--enable-developer-warnings
[root@varnish-server ~]#make
[root@varnish-server ~]#make install
[root@varnish-server ~]#cp redhat/varnish.initrc  /etc/init.d/varnish
[root@varnish-server ~]#cp redhat/varnish.sysconfig  /etc/sysconfig/varnish
```

其中，“PKG_CONFIG_PATH”是指定 Varnish 查找 pcre 库的路径。如果 pcre 安装在了其他路径下，在这里指定相应的路径即可，Varnish 默认查找 pcre 库的路径为 /usr/local/lib/pkgconfig。最后两步操作是复制一些 Varnish 守护进程的初始化脚本文件，这些脚本用于 Varnish 的启动、关闭等方面，在 2.4 节中会进行详细讲解。

至此，Varnish 安装完毕。

2.3 配置 Varnish

2.3.1 VCL 使用说明

VCL，即为 Varnish Configuation Language，用来定义 Varnish 的存取策略。VCL 语法比较简单，跟 C 和 Perl 比较相似，可以使用指定运算符“=”、比较运算符“==”、逻辑运算符“!,&&,!!”等形式；还支持正则表达式和用“~”进行 ACL 匹配运算；还可以使用“set”这样的关键字来指定变量。

需要注意的是，“\”字符在 VCL 里没有特别的含义，这点与其他语言略有不同。另外，VCL 只是配置语言，并不是真正的编程语言，没有循环，也没有自定义变量。

在讲述 Varnish 配置之前，首先需要了解 Varnish 的配置语法，即 VCL。下面对 VCL 常用的一些内置函数和公用变量进行详细介绍。

1. VCL 内置函数

（1）vcl_recv 函数

用于接收和处理请求。当请求到达并被成功接收后被调用，通过判断请求的数据来决定如何处理请求。

此函数一般以如下几个关键字结束。

- pass：表示进入 pass 模式，把请求控制权交给 vcl_pass 函数。
- pipe：表示进入 pipe 模式，把请求控制权交给 vcl_pipe 函数。
- error code [reason]：表示返回“code”给客户端，并放弃处理该请求。“code”是错误标识，例如 200 和 405 等。“reason”是错误提示信息。

（2）vcl_pipe 函数

此函数在进入 pipe 模式时被调用，用于将请求直接传递至后端主机，在请求和返回的内容没有改变的情况下，将不变的内容返回给客户端，直到这个连接被关闭。

此函数一般以如下几个关键字结束。

- error code [reason]。
- pipe。

（3）vcl_pass 函数

此函数在进入 pass 模式时被调用，用于将请求直接传递至后端主机。后端主机在应答数据后将应答数据发送给客户端，但不进行任何缓存，在当前连接下每次都返回最新的内容。

此函数一般以如下几个关键字结束。

- error code [reason]。
- pass。

（4）lookup

表示在缓存中查找被请求的对象，并且根据查找的结果把控制权交给函数 vcl_hit 或函数 vcl_miss。

（5）vcl_hit 函数

在执行 lookup 指令后，在缓存中找到请求的内容后将自动调用该函数。

此函数一般以如下几个关键字结束。

- deliver：表示将找到的内容发送给客户端，并把控制权交给函数 vcl_deliver。
- error code [reason] 。
- pass。

（6）vcl_miss 函数

在执行 lookup 指令后，在缓存中没有找到请求的内容时自动调用该方法。此函数可用于判断是否需要从后端服务器获取内容。

此函数一般以如下几个关键字结束。

- fetch：表示从后端获取请求的内容，并把控制权交给 vcl_fetch 函数。
- error code [reason] 。
- pass。

（7）vcl_fetch 函数

在后端主机更新缓存并且获取内容后调用该方法，接着，通过判断获取的内容来决定是将内容放入缓存，还是直接返回给客户端。

此函数一般以如下几个关键字结束。

- error code [reason]。
- pass。
- deliver。

（8）vcl_deliver 函数

将在缓存中找到请求的内容发送给客户端前调用此方法。

此函数一般以如下几个关键字结束。

- error code [reason]。
- deliver。

（9）vcl_timeout 函数

在缓存内容到期前调用此函数。

此函数一般以如下几个关键字结束。

- discard：表示从缓存中清除该内容。
- fetch。

（10）vcl_discard 函数

在缓存内容到期后或缓存空间不够时，自动调用该函数。

此函数一般以如下几个关键字结束。

- keep：表示将内容继续保留在缓存中。
- discard。

2.VCL 处理流程图

通过上面对 VCL 函数的介绍，读者能够对各个函数实现的功能有个简单的了解。其实每个函数之间都是相互关联的，图 2-1 所示为 Varnish 处理 HTTP 请求的运行流程图。

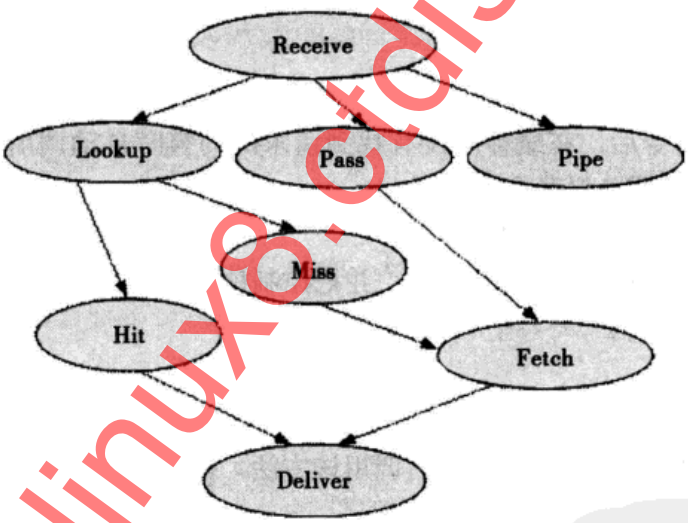

图 2-1 Varnish 处理 HTTP 请求的运行流程图

Varnish 处理 HTTP 请求的过程大致分为如下几个步骤。

（1）Receive 状态。也就是请求处理的入口状态，根据 VCL 规则判断该请求应该 Pass 或 Pipe，还是进入 Lookup（本地查询）。

（2）Lookup 状态。进入此状态后，会在 hash 表中查找数据，若找到，则进入 Hit 状态，否则进入 Miss 状态。

（3）Pass 状态。在此状态下，会进入后端请求，即进入 Fetch 状态。

（4）Fetch 状态。在 Fetch 状态下，对请求进行后端获取，发送请求，获得数据，并进行本地存储。

（5）Deliver 状态。 将获取到的数据发送给客户端，然后完成本次请求。

3. 内置公用变量

VCL 内置的公用变量可以用在不同的 VCL 函数中。下面根据这些公用变量使用的不同阶段依次进行介绍。

当请求到达后，可以使用的公用变量如表 2-2 所示。

表 2-2　请求到达后可以使用的 VCL 内置的公用变量

公用变量名称	含　义
req.backend	指定对应的后端主机
server.ip	表示服务器端 IP
client.ip	表示客户端 IP
req.request	指定请求的类型，例如 GET、HEAD 和 POST 等
req.url	指定请求的地址
req.proto	表示客户端发起请求的 HTTP 协议版本
req.http.header	表示对应请求中的 HTTP 头部信息
req. restarts	表示请求重启的次数，默认最大值为 4

Varnish 在向后端主机请求时，可以使用的公用变量如表 2-3 所示。

表 2-3　向后端主机请求时可以使用的 VCL 内置的公用变量

公用变量名称	含　义
beresp.request	指定请求的类型，例如 GET 合 HEAD 等
beresp.url	指定请求的地址
beresp .proto	表示客户端发起请求的 HTTP 协议版本
beresp .http.header	表示对应请求中的 HTTP 头部信息
beresp .ttl	表示缓存的生存周期，也就是 cache 保留多长时间，单位是秒

从 cache 或后端主机获取内容后，可以使用的公用变量如表 2-4 所示。

表 2-4　从 cache 或后端主机获取内容后可以使用的 VCL 内置的公用变量

公用变量名称	含　义
obj.status	表示返回内容的请求状态代码，例如 200、302 和 504 等
obj.cacheable	表示返回的内容是否可以缓存，也就是说，如果 HTTP 返回的是 200、203、300、301、302、404 或 410 等，并且有非 0 的生存期，则可以缓存
obj.valid	表示是否是有效的 HTTP 应答
obj.response	表示返回内容的请求状态信息
obj.proto	表示返回内容的 HTTP 协议版本
obj.ttl	表示返回内容的生存周期，也就是缓存时间，单位是秒
obj.lastuse	表示返回上一次请求到现在的间隔时间，单位是秒

对客户端应答时，可以使用的公用变量如表 2-5 所示。

表 2-5 对客户端应答时可以使用的公用变量

公用变量名称	含 义
resp.status	表示返回给客户端的 HTTP 状态代码
resp.proto	表示返回给客户端的 HTTP 协议版本
resp.http.header	表示返回给客户端的 HTTP 头部信息
resp.response	表示返回给客户端的 HTTP 状态信息

在上面的讲述中，只介绍了常用的 VCL 内置公用变量，如果需要了解和使用更多的公用变量信息，请登录 Varnish 官方网站查阅。

2.3.2 配置一个简单的 Varnish 实例

由于版本不同，Varnish 配置文件的写法也存在一定差异，Varnish 的 2.x 版本不但在配置文件写法上和 1.x 版本不同，而且还增加了很多新功能，并且去除了很多应用 bug。这里讲述的版本是 Varnish 2.1.2，配置文件写法以 Varnish 2.x 版本为基准。

Varnish 安装完成后，默认的配置文件为 /usr/local/varnish/etc/varnish/default.vcl，此文件内容默认全部被注释掉。这里以这个文件为模板，创建一个新的文件 vcl.conf，并且将其放到 /usr/local/varnish/etc 目录下。配置完成的 vcl.conf 文件如下：

```
            # 通过 backend 定义一个名称为 webserver 的后端主机，".host" 指定后端主机的 IP 地址或者域
            # 名，".port" 指定后端主机的服务器端口。其中，"192.168.12.26" 就是后端的一个 Web 服务器
backend webserver {
     .host = "192.168.12.26";
     .port = "80";
 }

# 开始调用 vcl_recv
sub vcl_recv {
        if (req.http.x-forwarded-for) {
                set req.http.X-Forwarded-For =
                req.http.X-Forwarded-For ", " client.ip;
        } else {
                set req.http.X-Forwarded-For = client.ip;
        }
           # 如果请求的类型不是 GET、HEAD、PUT、POST、TRACE、OPTIONS 或 DELETE 时，则进入
           # pipe 模式。注意这里是 "&&" 关系
        if (req.request != "GET" &&
           req.request != "HEAD" &&
           req.request != "PUT" &&
           req.request != "POST" &&
           req.request != "TRACE" &&
           req.request != "OPTIONS" &&
           req.request != "DELETE") {
           return (pipe);
        }
```

```
                # 如果请求的类型不是 GET 或 HEAD，则进入 pass 模式
    if (req.request != "GET" && req.request != "HEAD") {
        return (pass);
    }

                # 对 ixdba.net 或者 ixdba.cn 两个域名进行缓存加速。这是个泛域名的概念，也就
                # 是将所有以 ixdba.net 或者 ixdba.cn 结尾的域名都进行缓存
    if (req.http.host ~ "^(.*).ixdba.net" || req.http.host ~ "^(.*).ixdba.cn") {
        set req.backend = webserver;
    }

                # 对以 .jsp 和 .do 结尾以及带有 ? 的 URL，直接从后端服务器读取内容
    if (req.url ~ "\.(jsp|do)($|\?)") {
        return (pass);
    } else {
        return (lookup);
    }
}

sub vcl_pipe {
    return (pipe);
}

sub vcl_pass {
    return (pass);
}

sub vcl_hash {
    set req.hash += req.url;
    if (req.http.host) {
        set req.hash += req.http.host;
    } else {
        set req.hash += server.ip;
    }
    return (hash);
}

sub vcl_hit {
    if (!obj.cacheable) {
        return (pass);
    }
    return (deliver);
}

sub vcl_miss {
    return (fetch);
}

sub vcl_fetch {
    if (!beresp.cacheable) {
```

```
        return (pass);
    }
    if (beresp.http.Set-Cookie) {
        return (pass);
    }

        # 当 url 中包含 servlet 时，不进行缓存
    if (req.url ~ "^/servlet/") {
        return (pass);
    }

        # 当 url 中包含 services 时，不进行缓存
    if (req.url ~ "^/services/") {
        return (pass);
    }

        # 如果请求类型是 GET，并且请求的 URL 中包含 upload，那么就进行缓存，缓存的时间是
        #300 秒，即 5 分钟
    if (req.request == "GET" && req.url ~ "^/upload(.*)$") {
        set beresp.ttl = 300s;
    }
        # 当请求类型是 GET，并且请求的 URL 以 png、xsl、xml、gif、css、js 等结尾时，进行缓存，
        # 缓存时间为 600 秒
    if (req.request == "GET" && req.url ~ "\.(png|xsl|xml|pdf|ppt|doc|docx|chm|rar|
       zip|bmp|jpeg|swf|ico|mp3|mp4|rmvb|ogg|mov|avi|wmv|swf|txt|png|gif|jpg|css|
       js|html|htm)$") {
       set beresp.ttl = 600s;
    }
    return (deliver);
}
        # 下面添加一个 Header 标识，以判断缓存是否命中
sub vcl_deliver {
    if (obj.hits > 0) {
       set resp.http.X-Cache = "HIT from www.ixdba.net";
    } else {
      set resp.http.X-Cache = "MISS from www.ixdba.net";
    }
    return (deliver);
}
```

2.3.3 Varnish 对应多台 Web 服务器的配置实例

VCL 语法非常灵活，功能强大。下面是一个 Varnish 对应多台 Web 主机的应用实例，具有负载分担和健康检测机制。配置完成的 vcl.conf 文件如下：

```
# 下面定义了 4 台后端 Web 服务器
backend webserver1 {
  .host = "192.168.12.12";
  .port = "80";
}
backend webserver2 {
```

```
  .host = "192.168.12.13";
  .port = "80";
}
backend webserver3 {
  .host = "192.168.12.14";
  .port = "80";
}
backend webserver4 {
  .host = "192.168.12.15";
  .port = "80";
}

# 定义一个名为 webserver 的 director，也就是由 webserver1 和 webserver2 两台后端服务器随机分担
# 请求。".weight" 用来指定两台后端服务器的权值。权值高的处理请求的几率就高些
director webserver random {
  {.backend = webserver1; .weight = 5;  }
  {.backend = webserver2; .weight = 8;  }
}
# 这里设定清理缓存的规则，Varnish 允许 localhost、127.0.0.1 和 192.168.12.*** 三个来源 IP 通过
# PURGE 方法清除缓存
acl purge {
  "localhost";
  "127.0.0.1";
  "192.168.12.0"/26;
}
sub vcl_recv {

# 这里设定，当发送 PURGE 请求的客户端不是在 acl 中设定的地址时，将返回 405 状态代码，提示
# "Not allowed"。当请求的 URL 是以 .php 和 .cgi 结尾时，则交给后端服务器去处理
  if (req.request == "PURGE") {
      if (!client.ip ~ purge) {
      error 405 "Not allowed.";
    }
      elseif(req.url ~ "\.(php|cgi)($|\?)") {
      return (pass);
    }
      else {
      return (lookup);
    }
  }

# 下面设定域名访问策略，其实也是设定对后端主机健康状态检测的一个机制。如果访问 www.ixdba.net
# 或者 bbs.ixdba.net，并且请求重启次数为 0，则将请求交给 webserver 来处理。如果请求重启次数
# 为 1，则将请求交给 webserver3 处理。如果访问 img.ixdba.net 或者 images.ixdba.net，则将
# 请求交给 webserver4 来处理
  if((req.http.host ~"^(www.|bbs.)?ixdba.net")&&(req.restarts == 0))  {
      set req.backend = webserver;
  } elseif(req.restarts == 1) {
      set req.backend = webserver3;
  }
```

```
  if(req.http.host ~"^(img.|images.)?ixdba.net") {
      set req.backend = webserver4;
  }
# 下面定义缓存的策略。当请求以 .cgi 和 .php 结尾及带有 ? 的 URL 时，不进行缓存，直接从后端服务器
# 读取内容。其他请求都进入 lookup 模式，也就是进入 cache 中通过 hash 表寻找被请求的数据
  if (req.request != "GET" && req.request != "HEAD")
  {
    return (pipe);
  }
  elseif (req.url ~ "\.(cgi|php)($|\?)")
  {
   return (pass);
  }
  elseif (req.http.Authenticate || req.http.Authorization) {
    return (pass);
  }
  return (lookup);
}

# 如果请求的类型是 PURGE 方法，Varnishd 会将此请求的缓存周期设置为 0，也就是使这个 URL 的缓存失效，
# 从而达到刷新 Varnish 缓存的目的
sub vcl_hit
{
  if (req.request == "PURGE") {
    set obj.ttl = 0s;
    error 200 "Purged.";
  }

  if (!obj.cacheable)
  {
    return (pass);
  }

  if (obj.http.Vary)
  {
    unset obj.http.Vary;
  }
}

sub vcl_miss
{
  if (req.request == "PURGE") {
    error 404 "Not in cache.";
  }

}
# 定义 hash 的值，并且处理压缩内容
sub vcl_hash {
  set req.hash += req.url;
  if (req.http.host) {
```

```
    set req.hash += req.http.host;
  } else {
    set req.hash += server.ip;
  }
  if ( req.http.Accept-Encoding ){
    if (req.url ~ "\.(jpg|jpeg|png|gif|rar|zip|gz|tgz|bz2|tbz|mp3|ogg|swf|exe|flv
       |avi|rmvb|rm|mpg|mpeg|pdf)$") {
    } else {
        set req.hash += req.http.Accept-Encoding;
    }
  }
  return (hash);
}

sub vcl_fetch
{
     if (!beresp.cacheable) {
           return (pass);
     }

     if (beresp.http.Set-Cookie) {
           return (pass);
     }

# 定义在什么状态下进入 restart 模式
     if (beresp.status == 500 || beresp.status == 501 || beresp.status == 502 ||
        beresp.status == 503 || beresp.status == 504 || beresp.status == 404)
     {
        return (restart);
        }

# 下面定义不缓存含有哪些 HTTP 头的请求
      if (beresp.http.Pragma ~ "no-cache" || beresp.http.Cache-Control ~ "no-
         cache" || beresp.http.Cache-Control ~ "private") {
         return (pass);
  }

# 定义不同内容的缓存时间
      if (req.request == "GET" && req.url ~ "\.(css|js|html|htm)$") {
               set beresp.ttl = 300s;
    }
   if (req.request == "GET" && req.url ~ "\.(gif|jpg|jpeg|bmp|png|tiff|tif|ico|im
      g|bmp|wmf)$") {
        set beresp.ttl = 3600s;
        }
   if (req.request == "GET" && req.url ~ "\.(svg|swf|ico|mp3|mp4|m4a|wav|rmvb|avi
      |wmv)$") {
        set beresp.ttl = 10d;
        }
return (deliver);

}
```

```
sub vcl_deliver {
    if (obj.hits > 0) {
       set resp.http.X-Cache = "HIT from www.ixdba.net";
    } else {
      set resp.http.X-Cache = "MISS from www.ixdba.net";
    }
    return (deliver);
}
```

2.4 运行 Varnish

2.4.1 varnishd 指令

Varnish 启动的命令是 /usr/local/varnish/sbin/varnishd。此命令参数较多，用法比较复杂，在命令行执行“/usr/local/varnish/sbin/varnishd –h”即可得到 varnishd 的详细用法。表 2-6 列出了 varnishd 常用参数的使用方法和含义。

表 2-6 varnishd 常用参数的使用方法和含义

参 数	含 义
-a address:port	表示 Varnish 对 httpd 的监听地址及端口
-b address:port	表示后端服务器地址及端口
-d	表示使用 debug 调试模式
-f file	指定 Varnish 服务器的配置文件
-p param=value	指定服务器参数，用来优化 Varnish 性能
-P file	Varnish 进程 pid 文件存放路径
-n dir	指定 Varnish 的工作目录
-s kind[,storageoptions]	指定 Varnish 缓存内容的存放方式，常用的方式有：-s file，<dir_or_file>，<size> 其中 <dir_or_file> 用于指定缓存文件的存放路径，“<size>”用于指定缓存文件的大小
-t	指定默认的 TTL 值
-T address:port	设定 varnish 的 telnet 管理地址及端口
-w int[,int[,int]]	设定 Varnish 的工作线程数，常用的方式有： -w min,max -w min,max,timeout 如 -w5，51200，30 这里需要说明下，在 Varnish 2.0 版本以后，不能将最小启动的线程数设定过大，如果设定过大，会导致 Varnish 运行异常缓慢
-v	显示 Varnish 版本号和版权信息

2.4.2 配置 Varnish 运行脚本

在安装 Varnish 时，已经将 Varnish 的管理脚本复制到相应的目录下，这里稍作修改即

可。首先修改 /etc/sysconfig/varnish 文件，根据这里的要求，配置好的文件如下：

```
NFILES=131072
MEMLOCK=82000
DAEMON_OPTS="-a 192.168.12.246:80 \
             -T 127.0.0.1:3500 \
             -f /usr/local/varnish/etc/vcl.conf \
             -u varnish -g varnish \
             -w 2,51200,10 \
             -n /data/varnish/cache \
             -s file, /data/varnish/cache/varnish_cache.data,4G"
```

这里需要说明的是，在 32 位操作系统下，最大只能支持 2GB 的缓存文件 varnish_cache.data，如果需要更大的缓存文件，则需要安装 64 位的 Linux 操作系统。

接下来要修改的文件是 /etc/init.d/varnish，找到如下几行，修改相应的路径即可。

```
exec="/usr/local/varnish/sbin/varnishd"
prog="varnishd"
config="/etc/sysconfig/varnish"
lockfile="/var/lock/subsys/varnish"
```

其中，exec 用于指定 varnishd 的路径，只需修改为 Varnish 安装路径下对应的 varnishd 文件即可；config 用于指定 Varnish 守护进程配置文件路径。

两个文件修改完毕，就可以授权、运行 /etc/init.d/varnish 脚本了。执行过程如下：

```
[root@varnish-server ~]#chmod 755 /etc/init.d/varnish
[root@varnish-server ~]#/etc/init.d/varnish
Usage:/etc/init.d/varnish
{start|stop|status|restart|condrestart|try-restart|reload|force-reload}
```

从后两行的输出可知，此脚本功能强大，可以对 Varnish 进行启动、关闭、查看状态、重启等操作。最后，启动 Varnish。过程如下：

```
[root@varnish-server ~]# /etc/init.d/varnish  start
Starting varnish HTTP accelerator:                         [  OK  ]
```

2.4.3　管理 Varnish 运行日志

Varnish 是通过内存共享的方式提供日志的，它提供了两种日志输出形式，分别是：

- 通过自带的 varnishlog 指令获得 Varnish 详细的系统运行日志。

例如：

```
[root@varnish-server ~]#/usr/local/varnish/bin/varnishlog -n /data/varnish/cache
0 CLI          - Rd ping
0 CLI          - Wr 200 PONG 1279032175 1.0
0 CLI          - Rd ping
0 CLI          - Wr 200 PONG 1279032178 1.0
```

❑ 通过自带的 varnishncsa 指令得到类似 Apache 的 combined 输出格式的日志。

例如：

```
[root@varnish-server ~]#/usr/local/varnish/bin/varnishncsa  -n  /data/varnish/cache
```

也可以将日志输出到一个文件中，通过“-w”参数指定即可。

```
[root@varnish-server ~]#/usr/local/varnish/bin/varnishncsa -n /data/varnish/cache \
>-w /data/varnish/log/varnish.log
```

在 Varnish 的两种日志输出形式中，第一种在大多数情况下不是必须的，这里重点介绍第二种日志输出形式的配置方式。

下面编写一个名为 varnishncsa 的 shell 脚本，并把此文件放到 /etc/init.d 目录下。varnishncsa 脚本的完整内容如下所示：

```
#!/bin/sh

if [ "$1" = "start" ];then
/usr/local/varnish/bin/varnishncsa -n /data/varnish/cache  -f |/usr/sbin/
    rotatelogs /data/varnish/log/varnish.%Y.%m.%d.%H.log 3600 480 &

elif [ "$1" = "stop" ];then
      killall varnishncsa
else
                echo $0 "{start|stop}"
fi
```

在这个脚本中，通过管道方式把日志导入 rotatelogs 中，而 rotatelogs 是一个文件分割工具，它可以根据指定时间或者大小等方式来分割日志文件，这样就避免了日志文件过大而造成的性能问题。

其中，“3600”表示一个小时，也就是每个小时生成一个日志文件，“480”表示一个时区参数，中国是第八时区，相对于 UTC 相差 480 分钟。如果不设置 480 这个参数，将导致日志记录时间和服务器时间相差 8 小时。

通过对 Varnish 日志的监控，可以知道 Varnish 的运行状态和情况。

接着对此脚本进行授权。

```
[root@varnish-server ~]#chmod 755 /etc/init.d/varnishncsa
```

最后可以通过如下方式，对日志进行启动和关闭等操作。

```
[root@varnish-server ~]#/etc/init.d/varnishncsa  {start|stop }
```

2.5　管理 Varnish

2.5.1　查看 Varnish 进程

通过上一节的设置，Varnish 已经可以启动起来了。执行如下命令可以查看 Varnish 是否正常启动。

```
[root@varnish-server ~]# ps -ef|grep varnish
root 29615 1 0 00:20 pts/1 00:00:00 /usr/local/varnish/bin/varnishncsa -n /data/
    varnish/cache -f
root 29616 1 0 00:20 pts/1 00:00:00 /usr/sbin/rotatelogs /data/varnish/log/
    varnish.%Y.%m.%d.%H.log 3600 480
root 29646 1 0 00:21 ?  00:00:00 /usr/local/varnish/sbin/varnishd -P /var/run/
    varnish.pid -a 192.168.12.246:80 -T 127.0.0.1:3500 -f /usr/local/varnish/etc/
    vcl.conf -u varnish -g varnish -w 2,51200,10 -n /data/varnish/cache -s file,/
    data/varnish/cache/varnish_cache.data,4G
varnish 29647 29646 0 00:21 ? 00:00:00 /usr/local/varnish/sbin/varnishd -P /var/
    run/varnish.pid -a 192.168.12.246:80 -T 127.0.0.1:3500 -f /usr/local/varnish/
    etc/vcl.conf -u varnish -g varnish -w 2,51200,10 -n /data/varnish/cache -s
    file,/data/varnish/cache/varnish_cache.data,4G
```

从命令执行结果可知，PID 为 29615 和 29616 的进程是 Varnish 的日志输出进程，而 PID 为 29646 的进程是 varnishd 的主进程，并且派生出一个 PID 为 29647 的子进程。这点跟 Apache 类似。

如果 Varnish 正常启动，80 端口和 3500 端口应该处于监听状态，这一点通过如下命令可以查看。

```
[root@varnish-server ~]# netstat -antl|grep 3500
tcp        0      0 127.0.0.1:3500        0.0.0.0:*       LISTEN
[root@varnish-server ~]#netstat -antl|grep 80
tcp        0      0 192.168.12.246:80     0.0.0.0:*       LISTEN
tcp        1      0 192.168.12.246:41782  192.168.12.26:80        CLOSE_WAIT
```

其中，80 端口为 Varnish 的代理端口，3500 为 Varnish 的管理端口。

2.5.2　查看 Varnish 缓存效果与状态

可以通过浏览器访问对应的网页来查看 Varnish 缓存的效果。如果 Varnish 缓存成功，第二次打开网页的速度会明显比第一次快，但是这种方式并不能够充分说明问题。下面用命令行方式，通过查看网页头来查看命中情况。

```
[root@varnish-server ~]# curl -I http://www.ixdba.net/a/mz/2010/0421/11.html
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Unix) PHP/5.3.1 mod_perl/2.0.4 Perl/v5.10.1
Last-Modified: Sat, 10 Jul 2010 11:25:15 GMT
```

```
ETag: "5e850b-616d-48b06c6031cc0"
Content-Type: text/html
Content-Length: 24941
Date: Fri, 09 Jul 2010 08:29:16 GMT
X-Varnish: 1364285597
Age: 0
Via: 1.1 varnish
Connection: keep-alive
X-Cache: MISS from www.ixdba.net  #这里的“MISS”表示此次访问没有从缓存读取
```

再次打开这个页面，查看网页的头信息。

```
[root@varnish-server ~]# curl -I http://www.ixdba.net/a/mz/2010/0421/11.html
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Unix) PHP/5.3.1 mod_perl/2.0.4 Perl/v5.10.1
Last-Modified: Sat, 10 Jul 2010 11:25:15 GMT
ETag: "5e850b-616d-48b06c6031cc0"
Content-Type: text/html
Content-Length: 24941
Date: Fri, 09 Jul 2010 08:30:35 GMT
X-Varnish: 1364398612 1364285597
Age: 79
Via: 1.1 varnish
Connection: keep-alive
X-Cache: HIT from www.ixdba.net  #由“HIT”可知，第二次访问此页面时，从缓存中读取内容，也
    就是缓存命中
```

缓存命中率的高低直接说明了 Varnish 的运行状态和效果，较高的缓存命中率说明 Varnish 运行状态良好，Web 服务器的性能也会提高很多；反之，过低的缓存命中率说明 Varnish 的配置可能存在问题，需要进行调整。因此，从整体上了解 Varnish 的命中率和缓存状态，对于优化和调整 Varnish 至关重要。

Varnish 提供了一个 varnishstat 命令，通过它可以获得很多重要的信息。

下面是一个 Varnish 系统的缓存状态：

```
[root@varnish-server ~]#/usr/local/varnish/bin/varnishstat  -n /data/varnish/cache
Hitrate ratio:       10      100      113
Hitrate avg:     0.9999   0.9964   0.9964

      9990        68.92        49.70 Client connections accepted
    121820       953.84       606.07 Client requests received
    112801       919.88       561.20 Cache hits
        68         0.00         0.34 Cache misses
      2688        33.96        13.37 Backend conn. success
      6336         1.00        31.52 Backend conn. reuses
      2642        33.96        13.14 Backend conn. was closed
      8978        29.96        44.67 Backend conn. recycles
      6389         1.00        31.79 Fetch with Length
      2630        32.96        13.08 Fetch chunked
       444                           N struct sess_mem
```

```
    23          .           .   N struct sess
    64          .           .   N struct object
    78          .           .   N struct objectcore
    78          .           .   N struct objecthead
   132          .           .   N struct smf
     2          .           .   N small free smf
     3          .           .   N large free smf
     6          .           .   N struct vbe_conn
    14          .           .   N worker threads
    68       1.00        0.34 N worker threads created
     0       0.00        0.00 N queued work requests
  1201      11.99        5.98 N overflowed work requests
     1          .           .   N backends
     4          .           .   N expired objects
  3701          .           .   N LRU moved objects
118109     942.85      587.61 Objects sent with write
  9985      71.91       49.68 Total Sessions
121820     953.84      606.07 Total Requests
```

这里需要注意以下几点：

- “Client connections accepted”表示客户端向反向代理服务器成功发送 HTTP 请求的总数量。
- “Client requests received”表示到现在为止，浏览器向反向代理服务器发送 HTTP 请求的累计次数。由于可能会使用长连接，因此这个值一般会大于“Client connections accepted”的值。
- “Cache hits”表示反向代理服务器在缓存区中查找并且命中缓存的次数。
- “Cache misses”表示直接访问后端主机的请求数量，也就是非命中数。
- “N struct object”表示当前被缓存的数量。
- “N expired objects”表示过期的缓存内容数量。
- “N LRU moved objects”表示被淘汰的缓存内容个数。

2.5.3　通过端口管理 Varnish

Varnish 提供了基于端口的管理方式，用户可以通过 telnet 方式登录到管理端口，对 Varnish 子进程进行启动、关闭、查看状态和清除缓存等操作。具体的使用方式如下：

```
[root@varnish-server ~]#telnet 192.168.12.246 3500
Trying 192.168.12.246...
Connected to localhost.localdomain (192.168.12.246).
Escape character is '^]'.
200 154
-----------------------------
Varnish HTTP accelerator CLI.
-----------------------------
Type 'help' for command list.
```

```
Type 'quit' to close CLI session.

help                                    # 在这里输入“help”即可得到如下帮助信息
200 377
help [command]
ping [timestamp]
auth response
quit
banner
status                                  # 显示服务运行状态
start                                   # 启动 Varnish 的子服务
stop                                    # 关闭 Varnish 的子服务
stats                                   # 显示服务的全部状态
vcl.load <configname> <filename>        # 操作 VCL 配置文件的相关操作，如果要修改 VCL 文件，首先
                                        # 要通过 vcl.load 载入一个配置，然后执行 vcl.use 配置
vcl.inline <configname> <quoted_VCLstring>
vcl.use <configname>                    # 载入指定的配置文件
vcl.discard <configname>                # 丢弃指定的 VCL 配置文件
vcl.list                                # 显示当前载入的 VCL 配置文件信息
vcl.show <configname>                   # 可以显示某个 VCL 文件的内容
param.show [-l] [<param>]               # 用于显示程序的运行参数
param.set <param> <value>               # 用于动态更改某个运行参数
purge.url <regexp>                      # 用来清除指定规则的 url 缓存
purge <field> <operator> <arg> [&& <field> <oper> <arg>]...
purge.list                              # 列出执行过的规则列表
```

这里列举一个简单的操作示例，执行如下：

```
[root@varnish-server ~]#telnet 192.168.12.246 3500
Trying 192.168.12.246...
Connected to localhost.localdomain (192.168.12.246).
Escape character is '^]'.
200 154
-----------------------------
Varnish HTTP accelerator CLI.
-----------------------------
Type 'help' for command list.
Type 'quit' to close CLI session.

stop                                    # 这里表示关闭 varnishd 的子进程
200 0

Start                                   # 这里表示重新启动 varnishd 的子进程
200 0

Status                                  # 查看 varnishd 的运行状态
200 22
Child in state running
```

2.5.4　管理 Varnish 缓存内容

Varnish 的一个显著优点是可以灵活管理缓存内容。而管理缓存的主要工作是迅速有效地控制和清除指定的缓存内容。Varnish 清除缓存的操作相对比较复杂，不过幸运的是，可以通过 Varnish 的管理端口发送 purge 指令来清除不需要的缓存。

清除缓存内容的命令格式如下：

```
/usr/local/varnish/bin/varnishadm -T 192.168.12.246:3500 purge.url  <regexp>
```

列出最近清除的详细 URL 列表的命令如下：

```
/usr/local/varnish/bin/varnishadm -T 192.168.12.246:3500  purge.list
```

下面举例介绍如何管理 Varnish 缓存内容。

- 如果要清除 http://www.example.com/a/2010.html 的 URL 地址，可执行如下命令：

```
/usr/local/varnish/bin/varnishadm -T 192.168.12.246:3500  purge.url /a/2010.html
```

- 批量清除类似 http://www.example.com/a/b/s*.html 的 URL 地址，可执行如下命令：

```
/usr/local/varnish/bin/varnishadm -T 192.168.12.246:3500  purge.url ^/a/b/s.*$
```

- 批量清除类似 http://www.example.com/a/b/*.html 的 URL 地址，可执行如下命令：

```
/usr/local/varnish/bin/varnishadm -T 192.168.12.246:3500  purge.url^/a/b.*$
```

- 如果要清除所有缓存，可执行如下命令：

```
/usr/local/varnish/bin/varnishadm -T 192.168.12.246:3500 purge.url^.*$
```

- 查看最近清除的详细 URL 列表，可执行如下命令：

```
[root@varnish-server ~]# /usr/local/varnish/bin/varnishadm -T 192.168.12.246:3500
    purge.list
0x2dc310c0 1278674980.497631     0      req.url ~ /zm/a/web/2010/0423/64.html
0x2dc31100 1278674964.851327     1      req.url ~ ^/zm/a/d.*$
```

除了可以通过 Linux 命令行方式清理 Varnish 缓存外，还可以通过 telnet 到管理端口的方式来清理缓存页面，例如：

```
[root@varnish-server ~]#telnet 192.168.12.246 3500
Trying 192.168.12.246...
Connected to localhost.localdomain (192.168.12.246).
Escape character is '^]'.
200 154
-----------------------------
Varnish HTTP accelerator CLI.
-----------------------------
Type 'help' for command list.
Type 'quit' to close CLI session.

purge.url  /a/mz/2010/0421/11.html  #这里清除这个页面缓存
200 0
```

```
purge.url   ^/zm/a/d.*$    #这里清除/zm/a/目录下所有以字母d开头的缓存页面
200 0
```

对于系统管理人员或运维人员来说，时刻了解Varnish命中率是非常重要的。虽然Varnish给出了很详细的统计数据，但是这些统计数据不是很直观，并且必须登录到Varnish服务器才能查看。下面给出一个PHP程序，可以随时随地清晰地了解Varnish系统的命中率的相关情况。

```
<?php

$adminHost = "127.0.0.1"; //Varnish服务器的IP地址
$adminPort = "3500"; // Varnish服务器管理端口

returns the results, or an error on failure
function pollServer($command) {
   global $adminHost, $adminPort;

   $socket = socket_create(AF_INET, SOCK_STREAM, getprotobyname("tcp"));
   if ((!socket_set_option($socket, SOL_SOCKET, SO_RCVTIMEO, Array("sec" => "5",
      "usec" => "0"))) OR (!socket_set_option($socket, SOL_SOCKET, SO_SNDTIMEO,
      Array("sec" => "5", "usec" => "0")))) {
         die("Unable to set socket timeout");
   }

   if (@socket_connect($socket, $adminHost, $adminPort)) {
         $data = "";

         if (!$socket) {
               die("Unable to open socket to " . $server . ":" . $port . "\n");
         }

         socket_write($socket, $command . "\n");
         socket_recv($socket, $buffer, 65536, 0);
         $data .= $buffer;

         socket_close($socket);

         return $data;
   }
   else {
         return "Unable to connect: " . socket_strerror(socket_last_error()) . "\n";
   }
}

function byteReduce($bytes) {
   if ($bytes > 1099511627776) {
         return round($bytes / 1099511627776) . "TB";
   }
```

```
    else if ($bytes > 1073741824) {
            return round($bytes / 1073741824) . "GB";
    }
    else if ($bytes > 1048576) {
            return round($bytes / 1048576) . "MB";
    }
    else if ($bytes > 1024) {
            return round($bytes / 1024) . "KB";
    }
    else {
            return $bytes . "B";
    }
}

echo "<div class=\"inner\"><br />Statistics since last reset:<ul>";
$stats = pollServer("stats");
if (substr($stats, 0, 3) == "200") {
      $stats = preg_replace("/ {2,}/", "|", $stats);
      $stats = preg_replace("/\n\|/", "\n", $stats);
      $statsArray = explode("\n", $stats);

    array_shift($statsArray);
    $statistics = array();
    foreach ($statsArray as $stat) {
            @$statVal = explode("|", $stat);
            @$statistics[$statVal[1]] = $statVal[0];
    }
    unset($stats, $statsArray, $stat, $statVal);

    echo "<li>" . $statistics["Client connections accepted"] . " clients served
        over " . $statistics["Client requests received"] . " requests";
    echo "<li>" . round(($statistics["Cache hits"] / $statistics["Client requests
        received"]) * 100) . "% of requests served from cache";
    echo "<li>" . byteReduce($statistics["Total header bytes"] + $statistics["Total
        body bytes"]) . " served (" . byteReduce($statistics["Total header bytes"])
        . " headers, " . byteReduce($statistics["Total body bytes"]) . " content)";

    // echo "<li>" . byteReduce($statistics["bytes allocated"]) . " out of " .
       byteReduce($statistics["bytes allocated"] + $statistics["bytes free"]) .
       " used (" . round(($statistics["bytes allocated"] / ($statistics["bytes
       allocated"] + $statistics["bytes free"])) * 100) . "% usage)";
}
else {
    echo "Unable to get stats, error was: \"" . $stats;
}

echo "</ul></div>";

?>
```

将此PHP程序放到Varnish服务器上，即可统计出当前Varnish的命中率及缓存状态。统计结果类似于如下的一个输出：

```
Statistics since last reset:
❑ 63543 clients served over 584435 requests
❑ 98% of requests served from cache
❑ 4GB served (246MB headers, 4GB content)
```

在这个输出中，清晰地列出了浏览器的请求总数、缓存命中率、缓存区中所有缓存内容的HTTP头信息长度和缓存内容的正文长度。根据这个结果判断，Varnish的缓存效果还是很不错的，命中率很高。

2.6 Varnish优化

Varnish是否能稳定、快速地运行，与Linux本身的优化及Varnish自身参数的设置有很大关系。在安装配置完Varnish后，还必须从操作系统和Varnish配置参数两个方面对Varnish服务器进行性能优化，从而最大限度地发挥Varnish的性能优势。

2.6.1 优化Linux内核参数

内核参数是用户和系统内核之间交互的一个接口，通过这个接口，用户可以在系统运行的同时动态更新内核配置，而这些内核参数是通过Linux Proc文件系统存在的。因此，可以通过调整Proc文件系统达到优化Linux性能的目的。

以下参数是官方给出的一个配置：

```
net.ipv4.ip_local_port_range = 1024 65536
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
net.ipv4.tcp_fin_timeout = 30
net.core.netdev_max_backlog = 30000
net.ipv4.tcp_no_metrics_save=1
net.core.somaxconn = 262144
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syn_retries = 2
```

上面每个参数的含义如下：

❑ net.ipv4.ip_local_port_range：用来指定外部连接的端口范围，默认是32 768到61 000，这里设置为1024到65 536。

❑ net.core.rmem_max：指定接收套接字缓冲区大小的最大值，单位是字节。

- net.core.wmem_max：指定发送套接字缓冲区大小的最大值，单位是字节。
- net.ipv4.tcp_rmem：此参数与 net.ipv4.tcp_wmem 都是用来优化 TCP 接收 / 发送缓冲区的，包含 3 个整数值，分别是 min、default、max。

对于 tcp_rmem，min 表示为 TCP socket 预留的用于接收缓存的最小内存数量，default 表示为 TCP socket 预留的用于接收缓存的默认的内存值，max 表示用于 TCP socket 接收缓存的内存最大值。

对于 tcp_wmem，min 表示为 TCP socket 预留的用于发送缓存的内存最小值，default 表示为 TCP socket 预留的用于发送缓存的默认的内存值，max 表示用于 TCP socket 发送缓存的内存最大值。

- net.ipv4.tcp_fin_timeout：此参数用于减少处于 FIN-WAIT-2 连接状态的时间，使系统可以处理更多的连接。此参数值为整数，单位为秒。

  例如，在一个 tcp 会话过程中，在会话结束时，A 首先向 B 发送一个 fin 包，在获得 B 的 ack 确认包后，A 就进入 FIN-WAIT-2 状态等待 B 的 fin 包，然后给 B 发 ack 确认包。net.ipv4.tcp_fin_timeout 参数用来设置 A 进入 FIN-WAIT-2 状态等待对方 fin 包的超时时间。如果时间到了仍未收到对方的 fin 包就主动释放该会话。
- net.core.netdev_max_backlog：该参数表示当在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的最大数量。
- net.ipv4.tcp_syncookie：表示是否打开 SYN Cookie。tcp_syncookies 是一个开关，该参数的功能有助于保护服务器免受 SyncFlood 攻击。默认值为 0，这里设置为 1。
- net.ipv4.tcp_max_orphans：表示系统中最多有多少 TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这里设置的数字，连接就会复位并输出警告信息。这个限制仅仅是为了防止简单的 DoS 攻击。此值不能太小。这里设置为 262 144。
- net.ipv4.tcp_max_syn_backlog：表示 SYN 队列的长度，预设为 1024，这里设置队列长度为 262 144，以容纳更多的等待连接。
- net.ipv4.tcp_synack_retries：这个参数用于设置内核放弃连接之前发送 SYN+ACK 包的数量。
- net.ipv4.tcp_syn_retries：此参数表示在内核放弃建立连接之前发送 SYN 包的数量。

将以上内容添加到 /etc/sysctl.conf 文件中，然后执行如下命令，使设置生效。

```
[root@varnish-server ~]#sysctl  -p
```

2.6.2　优化系统资源

假设有这样一种情况，一台 Linux 主机上同时登录了 10 个用户，在没有限制系统资源的情况下，这 10 个用户同时打开了 500 个文档，而每个文档的大小为 10MB，这时系统的内存资源就会受到巨大的挑战。如果没有内存方面的限制，势必造成系统资源利用的混乱。而实际的应用环境要比这种假设复杂得多。这时，ulimit 就派上用场了。ulimit 是一种简单并且有效的实现资源限制的方式。

ulimit 可以限制系统的各个方面，它通过限制 shell 启动进程所占用的资源，来完成对系统资源的合理利用和分配。ulimit 支持对以下内容进行限制：所创建的内核文件的大小、内存锁住的大小、常驻内存集的大小、进程数据块的大小、打开文件描述符的数量、shell 进程所能使用的最大虚拟内存、shell 进程创建文件的大小、分配堆栈的最大值、单个用户的最大线程数和 CPU 时间等。同时，它还支持对硬资源和软资源的限制。

ulimit 有临时限制和永久限制两种实现方式。临时限制可以限制通过命令行登录的 shell 会话，并在会话终止时结束限制，而不影响与其他 shell 会话。对于永久限制，ulimit 命令可以将 ulimit 命令添加到有登录 shell 的配置文件中，这样就实现了对 shell 启动进程所占用的资源的永久限制。

ulimit 使用格式如下：

```
ulimit [options] [value]
```

options 中可设置的选项的含义以及简单示例如表 2-7 所示。

表 2-7 options 中可设置的选项的含义及示例

选项	含义	示例
-H	设置硬资源限制，设置后不能增加	ulimit -Hs 64； 限制硬资源，线程栈大小为 64KB
-S	设置软资源限制，设置后可以增加，但是不能超过硬资源限制	ulimit -Sn 32； 限制软资源，32 个文件描述符
-a	显示当前所有的资源限制	ulimit -a；
-c	最大的 core 文件的大小，以 block 为单位	ulimit -c unlimited； 对生成的 core 文件的大小不进行限制
-d	进程最大的数据段的大小，以 KB 为单位	ulimit -d unlimited； 对进程的数据段大小不进行限制
-f	进程可以创建文件的最大值，以 block 为单位	ulimit -f 2048； 限制进程可以创建的最大文件大小为 2048 blocks
-l	最大可加锁内存大小，以 KB 为单位	ulimit -l 32； 限制最大可加锁内存大小为 32 KB
-m	最大内存大小，以 KB 为单位	ulimit -m unlimited； 对最大内存不进行限制
-n	可以打开最大文件描述符的数量	ulimit -n 256； 限制最大可以使用 256 个文件描述符
-p	管道缓冲区的大小，以 KB 为单位	ulimit -p 512； 限制管道缓冲区的大小为 512 KB
-s	线程栈大小，以 KB 为单位	ulimit – s 512； 限制线程栈的大小为 512 KB
-t	最大的 CPU 占用时间，以秒为单位	ulimit -t unlimited； 对最大的 CPU 占用时间不进行限制
-u	用户最大可用的进程数	ulimit -u 64； 限制用户最多可以使用 64 个进程
-v	进程最大可用的虚拟内存，以 KB 为单位	ulimit -v 200000； 限制最大可用的虚拟内存为 200 000 KB

在了解了 ulimit 的含义和用法以后，接下来就可以针对 Varnish 系统进行相关的设定。这里的参数设定值如下（此值不能一概而论，需要根据应用环境的不同，选择适合的值）。

```
ulimit -HSn 131072
ulimit -HSc unlimited
```

为了保证这个限制永久生效，最好将 ulimit 设置放到 Varnish 的启动脚本中。

2.6.3　优化 Varnish 参数

telnet 到 Varnish 的 3500 管理端口，然后执行 param.show 命令即可看到 Varnish 运行中的所有参数。当然也可以通过这种方式更改相关参数，由于 Varnish 运行中用到的参数有很多，因此这里仅选取对 Varnish 性能影响比较大的几个参数进行介绍，更多介绍请查阅 varnish 官方文档。

首先查看以下 4 个参数：

```
thread_pools                    4 [pools]
thread_pool_min                 50 [threads]
thread_pool_max                 5120 [threads]
thread_pool_timeout             10 [seconds]
```

- thread_pools：用来设置线程池的数量。一般认为这个值和系统 CPU 的数目相同最好。设置多一些的 pool，Varnish 的并发处理能力会更强，但是也会消耗更多的 CPU 和内存。
- thread_pool_min：用来设置每个 pool 的最小 thread 数。pool 接收到可用的请求后，就会将请求分配给空闲的 thread 来处理。
- thread_pool_max：表示所有 pool 对应的 thread 数总和的最大值。此值不能太大，设置为系统峰值的 90% 左右即可，设置过大会导致进程被挂起。
- thread_pool_timeout：表示 thread 的超时过期时间。当 thread 数大于 thread_pool_min 设定值时，如果 thread 空闲超过 thread_pool_timeout 设定的时间，thread 就会被释放掉。

Varnish 运行中还有以下两个参数：

```
lru_interval                    20 [seconds]
listen_depth                    1024 [connections]
```

- lru_interval：这是个时间参数，表示在内存中如果某一个对象超过了此参数设定的时间后还没有被重用时，就把这个对象从 LRU（Least Recently Used）队列中移除。这是缓存系统的一个常用算法。合理地设置这个时间，可以提高系统的运行效率。
- listen_depth：这个参数用于设置 TCP 连接队列的长度，将其设置得大一些可以提高并发处理的能力。

对于这些优化参数，最好的方式是将它们加到 Varnish 的启动脚本中，然后通过 varnishd 的“-p”参数调用即可。

2.7 Varnish 的常见应用实例

Varnish 可应用在多方面，并且随着版本的升级，其功能性应用也在不断增加，这里仅介绍几个应用广泛、功能强大的应用实例。由于 Varnish 的配置格式在前面已经详细介绍过，所以这里只给出相应模块的配置代码，并附上相关的说明。

2.7.1 利用 Varnish 实现图片防盗链

图片防盗链功能对于大型网站的运维非常重要，各种 Web 服务器，如 Apache 和 Nginx 都可以很容易地实现图片防盗链功能。利用 Varnish 实现这个功能也非常简单，只需在配置文件的 vcl_recv 函数内增加如下配置即可。

```
if (req.http.referer ~ "http://.*") {
        if (  !(req.http.referer ~ "http://.*ixdba\.net"
            || req.http.referer ~ "http://.*google\.com"
            || req.http.referer ~ "http://.*yahoo\.cn"et
            || req.http.referer ~ "http://.*google\.cn"
            )) {
                set req.http.host = "www.ixdba.net";
                set req.url = "/templets/default/images/logo.gif";
        }
                    return (lookup);
        }
```

在这段配置中，用了一个内置变量 req.http.referer，防盗链就是通过 referer 来实现的。其实，referer 是 http header 的一部分，当浏览器向 Web 服务器发送请求的时候，一般会带上一个 referer 标识，用来告诉服务器请求是从哪个页面链接过来的服务器根据这个标识就可以获取信息来源，进而进行相应的处理。

这段配置的含义为：Varnish 服务器对接收或发送的请求进行判断，如果 referer 标识存在，且 referer 标识不匹配下面域名列表中的任意一个，就将请求重定向到 www.ixdba.net 域名下的 /templets/default/images/logo.gif 图片，而对找到匹配域名的请求执行 lookup 操作。

2.7.2 利用 Varnish 实现静态文件压缩处理

网页压缩技术是一种最为简便的提高网络速度的方法。通过压缩技术，可以减少服务器发送网页的大小，从而降低用户下载的时间，最大效率地利用带宽，提高网站的性能。Varnish 本身并不提供压缩的功能，但是可以将要压缩的工作交给后端的服务器去完成，从而变相实现了网页压缩。

首先在 Varnish 配置文件的“vcl_recv”函数中加入如下配置：

```
if (req.http.Accept-Encoding) {
if (req.url ~ "\.(jpg|png|gif|gz|tgz|bz2|tbz|mp3|ogg)$") {
```

```
                # No point in compressing these
                remove req.http.Accept-Encoding;
                } else if (req.http.Accept-Encoding ~ "gzip") {
                                set req.http.Accept-Encoding = "gzip";
                } else if (req.http.Accept-Encoding ~ "deflate") {
                                set req.http.Accept-Encoding = "deflate";
                } else {
                                remove req.http.Accept-Encoding;
                }
        }
```

然后修改“vcl_hash”函数为如下配置：

```
sub vcl_hash {
    set req.hash += req.url;
          if (req.http.Accept-Encoding ~ "gzip") {
                        set req.hash += "gzip";
        }
        else if (req.http.Accept-Encoding ~ "deflate") {
                        set req.hash += "deflate";
        }

    return (hash);
}
```

这样就完成了 Varnish 的压缩配置，将需要压缩的内容都交给了后端服务器去处理。

下面通过一个实例测试来验证压缩的效果。

首先用 curl 命令请求未压缩的内容。

```
[root@varnish-server ~]#curl -I  http://www.ixdba.com/article/3e/1557.html
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Unix) PHP/5.3.1 mod_perl/2.0.4 Perl/v5.10.1
Last-Modified: Mon, 28 Jul 2008 00:48:20 GMT
ETag: "7102d5-819f-4530ae1357d00"
Vary: Accept-Encoding,User-Agent
Content-Type: text/html
Content-Length: 33183
Date: Fri, 16 Jul 2010 06:34:35 GMT
X-Varnish: 1515651004 1515651001
Age: 17
Via: 1.1 varnish
Connection: keep-alive
X-Cache: HIT from ixdba.net
```

然后用 curl 命令请求压缩的内容。

```
[root@varnish-server~]#curl http://www.ixdba.com/article/3e/1557.html \
>-H Accept-Encoding:gzip,defalte -I
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Unix) PHP/5.3.1 mod_perl/2.0.4 Perl/v5.10.1
Last-Modified: Fri, 16 Jul 2010 05:50:54 GMT
```

```
ETag: "748b19-8197-48b7acd54cb80"
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Type: text/html
Content-Length: 8538
Date: Fri, 16 Jul 2010 06:34:27 GMT
X-Varnish: 1515651003 1515651002
Age: 5
Via: 1.1 varnish
Connection: keep-alive
X-Cache: HIT from ixdba.net
```

通过前后两个输出结果可以清楚地看到，压缩已经生效，说明配置成功。

2.8 本章小结

本章主要介绍了高性能HTTP加速器Varnish的安装、配置、管理和使用技巧，同时还介绍了Varnish常用的指令以及Varnish的调优技巧和经验，最后通过两个实例介绍了Varnish的常见应用。通过本章的学习，读者可以对Varnish的结构和特点有一个清晰的认识，并且能够熟练安装、搭建和配置一套Varnish系统。

Varnish是一个开源的反向代理软件和HTTP加速器，与传统的Squid相比，Varnish具有性能更高、速度更快、管理方便等诸多优点，很多大型的运营网站都开始尝试用Varnish来替换Squid，这些都促使Varnish迅速发展起来。本章通过理论与实践经验相结合的方法，力求使读者迅速掌握Varnish的使用。

第 3 章　Memcached 应用实战

本章主要介绍 Memcached 的特征、运行原理和使用经验。Memcached 是一套分布式内存对象缓存系统，用于在动态系统中减少数据库负载，进而提升系统性能。Memcached 在很多时候都作为数据库的前端 cache 使用，因为它比数据库少了很多 SQL 解析、磁盘操作等开销，而且使用内存来管理数据，所以它可以提供比直接读取数据库更好的性能。另外，Memcached 也经常作为服务器之间数据共享的存储媒介。学习完本章，相信读者能够对 Memcached 有一个全面的了解，使构建一套属于自己的分布式内存对象缓存系统变成很简单的事情。

3.1　Memcached 基础

3.1.1　什么是 Memcached

Memcached 是一个免费开源的、高性能的、具有分布式内存对象的缓存系统，它通过减轻数据库负载加速动态 Web 应用。最初版本由 LiveJournal 的 Brad Fitzpatrick 在 2003 年开发完成。目前全世界很多用户都在使用它来构建自己的大负载网站或提高自己的高访问网站的响应速度。Memcache 是这个项目的名称，而 Memcached 是服务器端的主程序文件名。

缓存一般用来保存一些经常存取的对象或数据（例如，浏览器会把经常访问的网页缓存起来），通过缓存来存取对象或数据要比磁盘存取快很多。Memcache 是一种内存缓存，把经常存取的对象或数据缓存在内存中，内存中缓存的这些数据通过 API 的方式被存取，数据就像一张大的 HASH 表，以 key-value 对的方式存在。Memcache 通过缓存经常被存取的对象或数据，来减轻数据库的压力，提高网站的响应速度，构建速度更快的可扩展的 Web 应用。图 3-1 展示了 Memcache 和数据库协作的流程。

这个流程的具体逻辑如下：

1）检查客户端请求的数据是否在 Memcache 中存在，如果存在，直接把请求的数据返回，不再对数据进行任何操作。

2）如果请求的数据不在 Memcache 中，就去查询数据库，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到 Memcache 中。

3）每次更新数据库（如更新、删除数据库的数据）的同时更新 Memcache 中的数据，保证 Memcache 中的数据和数据库中的数据一致。

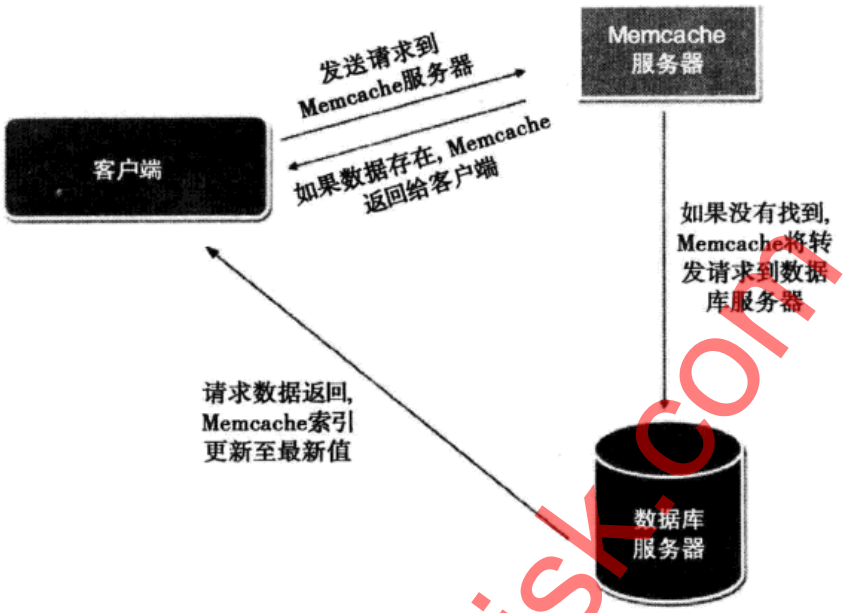

图 3-1 Memcache 和数据库协作工作流程

4）当分配给 Memcache 内存空间用完之后，会使用 LRU（Least Recently Used，最近最少使用）策略加到期失效策略，失效的数据首先被替换掉，然后再替换掉最近未使用的数据。

3.1.2 Memcached 的特征

Memcached 作为高性能的缓存服务器，具有如下特征：

- 协议简单。
- 基于 libevent 的事件处理。
- 内置的内存管理方式。
- 互不通信的 Memcached 之间具有分布特征。

下面分别进行简单介绍。

1. 协议简单

Memcached 的协议实现比较简单，使用的是基于文本行的协议，能直接通过 telnet 在 Memcached 服务器上存取数据。

2. 基于 libevent 的事件处理

了解 libevent 的用户都知道，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 利用这个库进行异步事件处理。关于这个库的详细内容，有兴趣的读者可以查看相关文档。

3. 内置的内存管理方式

Memcached 有一套自己管理内存的方式，这套管理方式非常高效，所有的数据都保存在 Memcached 内置的内存中，当存入的数据占满空间时，使用 LRU 算法自动删除不使用的缓存，即重用过期数据的内存空间。Memcached 是为缓存系统设计的，没有考虑数据的容灾问题，和机器的内存一样，重启机器数据将会丢失。

4. 互不通信的 Memcached 之间具有分布特征

各个 Memcached 服务器之间互相不通信，都是独立的存取数据，不共享任何信息。通过对客户端的设计，让 Memcached 具有分布式，能支持海量缓存和大规模应用。

3.1.3　Memcached 的安装

Memcached 的安装比较简单，这里稍加说明。很多平台支持 Memcached，常见的有：Linux、FreeBSD、Solaris、Mac OS X。

也可以将 Memcached 安装在 Windows 系统上。这里以 CentOS 为例进行说明。

1. 安装 libevent

安装 Memcached 前需要先安装 libevent，首先用 wget 下载 libevent。

```
[root@web181 ~]# wget -b http://www.monkey.org/~provos/libevent-1.4.13-stable.
    tar.gz
Continuing in background, pid 8752.
Output will be written to `wget-log'.
[root@web181 ~]# tail -5 wget-log | sed '/^$/d'
   450K .......... .......... ..........        100%  139K=8.3s #说明下载完成
2010-09-29 23:20:03 (58.6 KB/s) - `libevent-1.4.13-stable.tar.gz' saved
    [499603/499603]
```

下面开始安装。

```
[root@web181 ~]# tar zxf libevent-1.4.13-stable.tar.gz
[root@web181 ~]# cd libevent-1.4.13-stable
[root@web181 libevent-1.4.13-stable]# ./configure
[root@web181 libevent-1.4.13-stable]# make && make install
```

CentOS 系统也可以通过 yum 直接安装 libevent，不过所安装的版本可能比较低。

2. 安装 Memcached

安装 Memcached 的过程如下：

```
[root@web181 ~]# wget -b http://memcached.googlecode.com/files/memcached-1.4.5.tar.gz
Continuing in background, pid 8659.
Output will be written to `wget-log'.
[root@web181 ~]# tail -5 wget-log | sed '/^$/d'
   250K .......... .......... .......... .......... .....     100%  145K=2.8s
2010-09-29 23:18:03 (105 KB/s) - `memcached-1.4.5.tar.gz' saved [302516/302516]
```

```
[root@web181 ~]# tar zxf memcached-1.4.5.tar.gz
[root@web181 memcached-1.4.5]# ./configure
[root@web181 memcached-1.4.5]# make && make install
```

安装完成后，Memcached 的默认目录为 /usr/local/bin/memcached。

3. 启动 Memcached

Memcached 启动的过程如下：

```
[root@web181~]# /usr/local/bin/memcached -m 32m -p 11211 -d -u root -P /var/run/
    memcached.pid -c 256
```

这里需要说明的是，启动时可能出现如下错误：

```
[root@web181~]# /usr/local/bin/memcached -m 32m -p 11211 -d -u root -P /var/run/
    memcached.pid -c 256 -vv
/usr/local/bin/memcached: error while loading shared libraries: libevent-1.4.so.2:
    cannot open shared object file: No such file or directory
```

找不到 libevent-1.4.so.2 文件的解决的办法是，把 /usr/local/lib 加入到 /etc/ld.so.conf 中，过程如下：

```
[root@web181 ~]# echo "/usr/local/lib" >> /etc/ld.so.conf
[root@web181 ~]# ldconfig
```

启动过程中所用选项说明如下：

- -p，使用的 TCP 端口。默认为 11 211。
- -m，最大内存大小。默认为 64MB。
- -vv，以 very vrebose 模式启动，将调试信息和错误输出到控制台。
- -d，作为守护进程在后台运行。
- -c，最大运行的并发连接数，默认是 1024，按照服务器的负载量来设定。
- -P，设置保存 Memcache 的 pid 文件。
- -l，监听的服务器 IP 地址，如果有多个地址。
- -u，运行 Memcached 的用户，默认不能由 root 用户启动，所以当前用户为 root 用户时，需要利用 -u 参数来指定。

还有很多其他选项，通过 /usr/local/bin/memcached –h 命令可以显示所有可用选项。其中很多选项能改变 Memcached 的各种行为，推荐读者阅读相关资料。

4. 测试启动是否连接正常

下面通过一个示例简单测试连接是否正常，每个步骤的操作含义均在语句后进行注释。

```
[root@web181 ~]# telnet localhost 11211
Trying 127.0.0.1...
<30 new auto-negotiating client connection
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
set test 0 0 10          # 向 test 中存入数据
```

```
30: Client using the ascii protocol
<30 set test 0 0 10
test_value              #输入的 key 为 test 存入的数据
>30 STORED
STORED                 #返回 set 结果
get test                  #获取的数据
<30 get test
>30 sending key test
>30 END
VALUE test 0 10
test_value              #取得的 key 为 test 中的数据
END
quit
<30 quit
<30 connection closed.
Connection closed by foreign host.
```

5. 关闭 Memcached

关闭 Memcached 的命令如下：

```
[root@web181 ~]# kill `cat /var/run/memcached.pid`
```

6. 安装 Memcache 的 PHP 扩展

1）在 http://pecl.php.net/package/memcache 中选择想要下载的 Memcache 版本。

2）这里以 memcache-2.2.5 版本为例来安装 Memcache 的 PHP 扩展。安装代码如下：

```
[root@web181 ~]# wget -b http://pecl.php.net/get/memcache-2.2.5.tgz
Continuing in background, pid 20072.
Output will be written to `wget-log'.
[root@web181~]# tail -5 wget-log | sed '/^$/d'
0K .......... .......... .......... .....                  100% 24.9K=1.4s
2010-09-30 13:09:27 (24.9 KB/s) - `memcache-2.2.5.tgz' saved [35981/35981]
[root@web181 ~]# tar zxf memcache-2.2.5.tgz
[root@web181 ~]# cd memcache-2.2.5
[root@web181 memcache-2.2.5]#/usr/local/php/bin/phpize
Configuring for:
PHP Api Version:         20041225
Zend Module Api No:      20060613
Zend Extension Api No:   220060519
[root@web181 memcache-2.2.5]#./configure \
--enable-memcache \
--with-php-config=/usr/local/php/bin/php-config
[root@web181 memcache-2.2.5]# make && make install
```

3）完成上述安装后会有类似以下的提示：

```
Installing shared extensions:
/usr/local/php-cgi/lib/php/extensions/no-debug-zts-20060613/
```

4）修改 php.ini 文件，把 php.ini 中的 extension_dir =" ./" 修改为 extension_dir = "/usr/local/php/lib/php/extensions/no-debug-zts-20060613/"。

5）添加如下一行代码来载入 Memcache 扩展：

```
extension=memcache.so
```

7. 测试 Memcache 的 PHP 扩展是否安装成功

运行下面的 PHP 代码，如果输出“Hello world!”，就表示环境搭建成功。

```
<?php
$mem = new Memcache;
$mem->connect('127.0.0.1', 11211);
$mem->set('test', 'Hello world!', 0, 12);
$val = $mem->get('test');
echo $val;
?>
```

3.1.4 Memcached 的简单使用过程

前面已经提到过，许多语言都实现了 Memcached 的客户端，目前应用最多的是通过 Perl、PHP 实现对 Memcached 的操作。这里以 Perl 的 Cache::Memcached 为例简要说明使用 Memcached 的过程。Perl 的 Cache::Memcached 在 CPAN 上对应的地址为：http://search.cpan.org/dist/CacheMemcached/。

通过 Cache::Memcached 连接 Memcached 的示例代码如下：

```
#!/usr/bin/perl

use Cache::Memcached;

my $key = "test";
my $value = "HelloWorld!";
my $expires = 120;
my $memcached = Cache::Memcached->new({
servers => ["127.0.0.1:11211"],
compress_threshold => 100
});
$memcached->add($key, $value, $expires);
my $return = $memcached->get($key);
print "$return\n";
```

Cache::Memcached 的常用选项如下：

- servers，用数组指定连接 Memcached 服务器的 IP 地址和端口号。
- compress_threshold，启用数据压缩时使用的值。
- namespace，指定添加到键的前缀。

另外，利用 Cache::Memcached 可以将 Perl 的复杂数据序列化之后再保存，所以能直接将散列、数组、对象都保存到 Memcached 的缓存中。

1. 向 Memcached 存数据的方法

向 Memcached 存数据的方法有 set、add 和 replace，这些方法的使用方式相同。

```
my $add = $memcached->add( '键', '值', '过期时间' );
my $replace = $memcached->replace( '键', '值', '过期时间' );
my $set = $memcached->set( '键', '值', '过期时间' );
```

向 Memcached 存数据时可以指定过期时间（秒），若不指定过期时间，Memcached 则按照 LRU 算法保存存入的数据。但这三个方法有一些区别：

- set，无论在什么情况下都保存写入的数据。
- add，仅当存储空间中不存在 key 相同的数据时才保存。
- replace，仅当存储空间中存在 key 相同的数据时才保存。

2. 获取数据的方法

可以使用 get 和 get_multi 方法来获取数据，使用方法如下：

```
my $value = $memcached->get( '键');
my $value = $memcached->get_multi( '键1', '键2', '键3');
```

如果想一次获取多条数据，可以使用 get_multi 方法。利用 get_multi 方法可以非同步地同时取得多个键值，其速度要比循环调用 get 方法快很多倍。

3. 删除数据方法

删除数据可以使用 delete 方法。

```
$memcached->delete('键', '阻塞时间（秒）');
```

第一个参数指定要删除的数据的键，第二个参数指定一个时间值，以禁止使用同样的键保存新数据，这个功能可以防止缓存数据的不完整。但是 set 方法忽视该阻塞，会继续保存数据。

4. 加一和减一方法

可以把 Memcached 上某一个特定的键值作为计数器使用。

```
my $return = $memcached->incr('键');
$memcached->add('键', 0) unless defined $return;
```

加一和减一都属于原子操作，进行这两个操作时若未对键值设置初始值，则不会自动将初始值赋为 0。因此，应当进行错误检查，必要时加入初始化操作。

3.2　剖析 Memcached 的工作原理

3.2.1　Memcached 的工作过程

Memcached 是一种 C/S 模式，在服务器端启动服务守护进程，此时可以指定监听的 IP 地址、端口号以及使用多少内存来处理客户端的请求等几个关键参数。服务器端的服务启动后就一直处于等待处理客户端的连接状态。Memcached 是由 C 语言来实现的，采用的是异步 I/O，其实现方式是基于事件的单进程和单线程的。使用 libevent 作为事件通知机制，多个服

务器端可以协同工作，但这些服务器端之间没有任何通信关系，每个服务器端只对自己的数据进行管理。客户端通过指定服务器的IP地址和端口进行通信。

需要被缓存的对象或数据以key/value对的形式保存在服务器端，每个被缓存的对象或数据都有唯一的标识符key，存取操作通过这个key进行。保存到Memcached中的对象或数据放置在内存中，并不会作为文件存储在磁盘上，所以存取速度非常快。由于没有对这些对象进行持久性存储，因此在服务器端的服务重启之后存储在内存中的这些数据就会消失。而且当存储的容量达到启动时设定的值时，就自动使用LRU算法删除不用的缓存。Memcached是为缓存而设计的服务器，因此在设计之初并没有过多考虑数据的永久性问题。Memcached支持各种语言编写的客户端API，目前包括Perl、PHP、Python、Ruby、Java、C#和C等。

3.2.2 Slab Allocation的工作机制

Memcached利用Slab Allocation机制来分配和管理内存。传统的内存管理方式是：使用完通过malloc分配的内存后通过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。Slab Allocation机制不存在这样的问题，它按照预先规定的大小，将分配的内存分割成特定长度的内存块，再把尺寸相同的内存块分成组，这些内存块不会释放，可以重复利用。

Memcached服务器端保存着一个空闲的内存块列表，当有数据存入时根据接收到的数据大小，分配一个能存下这个数据的最小内存块。这种方式有时会造成内存浪费，例如：将一个200字节的数据存入一个300字节的内存块中，会有100字节内存被浪费掉，不能使用。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，这样就可以减少对内存的浪费。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，使用默认值1.25进行部署。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。

3.2.3 Memcached的删除机制

上一节已经介绍过，Memcached不会释放已分配的内存空间，在数据过期后，客户端不能通过key取出它的值，其存储空间被重新利用。

Memcached使用的是一种Lazy Expiration策略，自己不会监控存入的key/value对是否过期，而是在获取key值时查看记录的时间戳，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。

Memcached在分配空间时，优先使用已经过期的key/value对空间，当空间占满时，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，将其空间分配给新的key/value对。在某些情况下，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。

3.2.4　Memcached 的分布式算法

前面已经介绍过，Memcached 的分布式是通过客户端的程序库来实现的。下面举例描述其工作过程。

假设有 node1、node2、node2 三台 Memcached 服务器，应用程序要实现保存名为“tokyo”、“ tokyo1”、“ tokyo2”、“ tokyo3”的数据，如图 3-2 所示。

向 Memcached 中存入“tokyo”，将“tokyo”传给客户端程序后，客户端实现的算法就会根据这个“键”来决定保存数据的 Memcached 服务器，选定服务器后，就命令该服务器保存“tokyo”及其值，如图 3-3 所示。

同样，存入“tokyo1”、“ tokyo2”和“ tokyo3”的过程都是先通过客户端的算法选择服务器再保存数据。

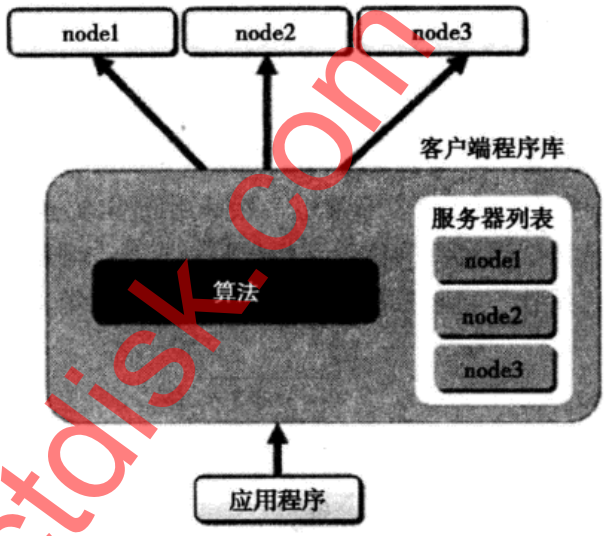

图 3-2　向 Memcached 中存入数据的初始状态

接下来获取保存的数据。获取保存的 key/value 对时也要将要获取的键“tokyo”传递给函数库。函数库通过与存取数据操作相同的算法，根据“键”来选择服务器。只要使用的算法相同，就能确定存入在哪一台服务器上，然后发送 get 命令。只要数据没有因为某些原因被删除，就能获得保存的值，如图 3-4 所示。

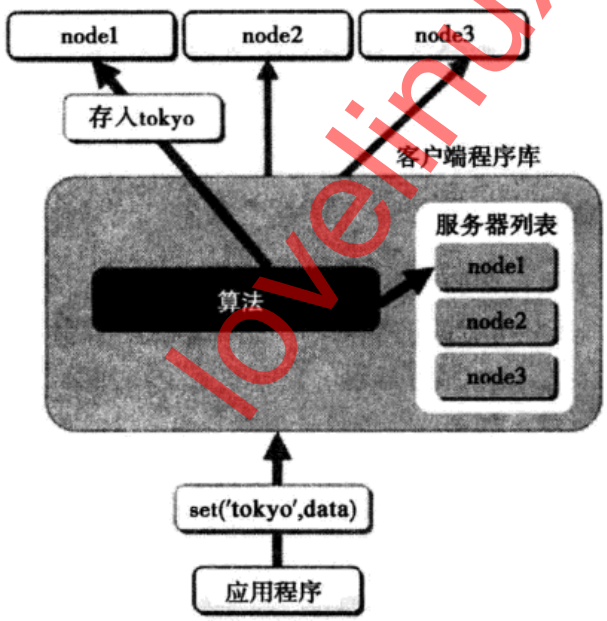

图 3-3　向 Memcached 中存入“tokyo”

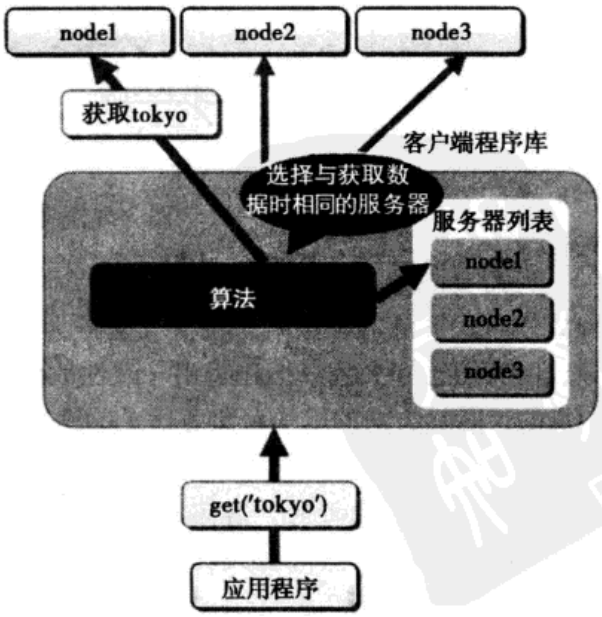

图 3-4　向 Memcached 获取“tokyo”的过程

将不同的键保存到不同的服务器上，就实现了 Memcached 的分布式算法。部署多台 Memcached 服务器时，将键分散保存到这些服务器上，当某一台 Memcached 服务器发生故障无法连接时，只有分散到这台服务器上的 key/values 对不能访问，其他 key/value 对不受影响。

目前有两种分布式算法使用得最多，一种是根据余数来计算分布，另一种是根据一致性散列算法来计算分布。根据余数分布式算法先求得键的整数散列值，再除以服务器台数，根据余数来选择将键存放到哪一台服务器上。这种方法虽然计算简单，效率很高，但在服务器增加或减少时，会导致几乎所有的缓存失效，所以在大规模部署中，很少使用这种方法。一致性散列的原理如图 3-5 所示，先算出 Memcached 服务器（节点）的散列值，并将其分散到 0 到 2 的 32 次方的圆上，然后用同样的方法算出存储数据的键的散列值并映射到圆上，最后从数据映射到的位置开始顺时针查找，将数据保存到查找到的第一个服务器上。如果超过 2^{32} 仍然找不到服务器，就会将数据保存到第一台 Memcached 服务器上。

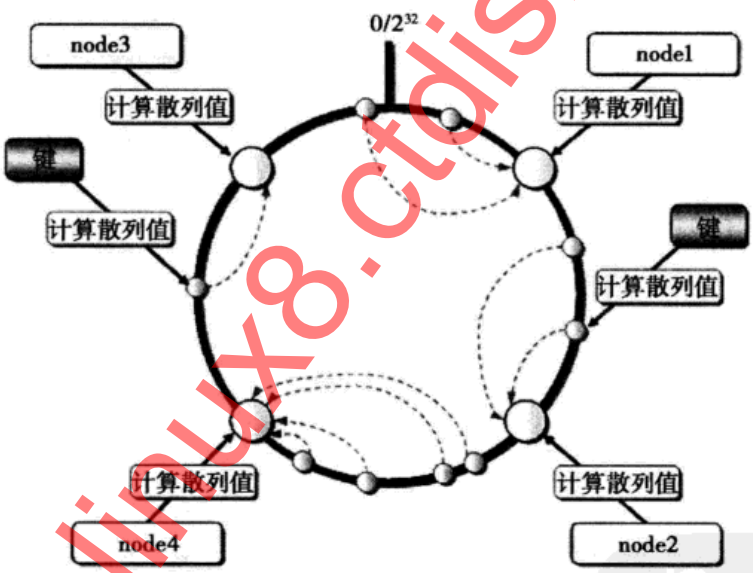

图 3-5 一致性散列算法的原理

当需要添加一台 Memcached 服务器时，由于保存键的服务器的个数发生了变化，因此余数分布式算法的余数结果也会发生巨大变化，几乎所有的键都找不到之前存入的服务器，导致所有的缓存失效。但在采用一致性散列算法时，添加服务器后，只有在圆上增加服务器地点的逆时针方向的第一台服务器上的键会受到影响，如图 3-6 所示。

一致性散列算法对数据的存储是不均匀的，但可以最大限度地减少缓存的失效量。在大规模部署 Memcached 时，容灾和扩容一定要使用一致性散列算法，以确保在出现故障或容量问题时减小对数据库的影响。

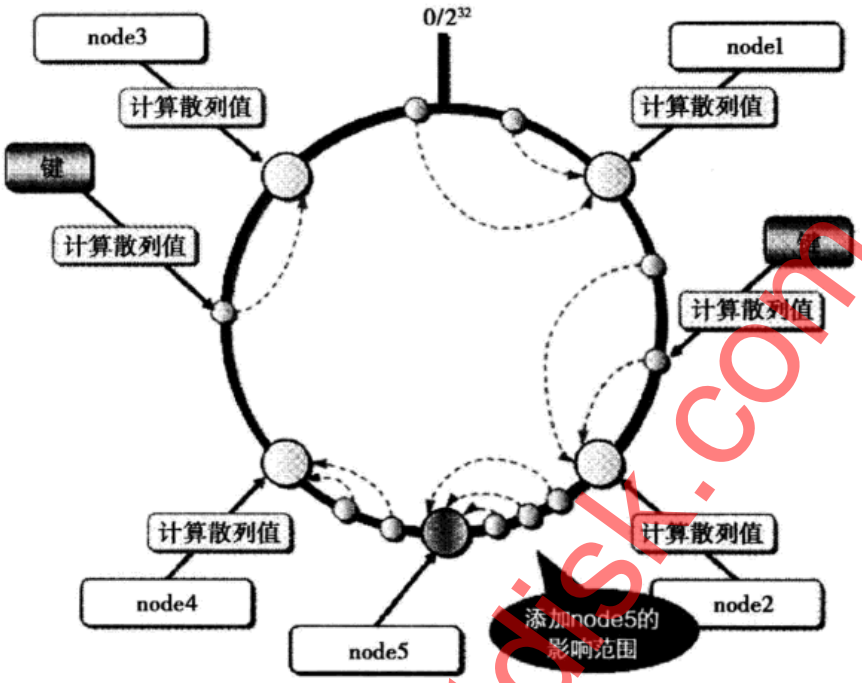

图 3-6　在使用一致性散列算法时增加一台新服务器

3.3　Memcached 的管理与性能监控

3.3.1　如何管理 Memcached

1. 通过 Memcached 的监听端口进行管理

Memcached 的管理相对比较容易，通过命令行登录到 Memcached 的监听端口，然后执行一些命令，通过这些命令的输入即可查看 Memcached 的运行状态。

管理 Memcached 的命令如下：

- stats，统计 Memcached 的各种信息。
- stats reset，重新统计数据。
- stats slabs，显示 slabs 信息。通过这个命令能获取每个 slabs 的 chunksize 长度，从而确定数据到底保存在哪个 slab。
- stats items，显示 slab 中的 item 数目。
- set | get, gets，前面已经介绍过，用来保存或获取数据。

例如，要查看 Memcached 的统计信息，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@web181 ~]# telnet 192.168.1.181 11211
Trying 192.168.1.181...
Connected to web181 (192.168.1.181).
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
```

通过以上信息可以看到 Memcached 的状态、连接的次数、当前的并发连接数。通过这些信息可以分析出，当前 Memcached 的换入换出是否比较厉害，容量是否足够。

2. 利用 memcached-tool 管理 Memcached

memcached-tool 是 Brad Fitzpatrick 利用 Perl 编写的一个 Memcached 管理脚本。这个脚

本通过将之前的命令行进行封装，使输入的值更加规整，进而更便于分析查看。可以通过下面的网址下载 memcached-tool：http://code.sixapart.com/svn/memcached/trunk/server/scripts/memcached-tool。

下面是 memcached-tool 的基本用法：

```
[root@web181 ~]# perl memcached-tool
Usage: memcached-tool <host[:port]> [mode]
       memcached-tool 10.0.0.5:11211 display
       memcached-tool 10.0.0.5:11211
       memcached-tool 10.0.0.5:11211 stats
       memcached-tool 10.0.0.5:11211 move 7 9
```

memcached-tool 的执行示例如下：

```
[root@web181 ~]# perl memcached-tool 127.0.0.1:11211
  1       96 B       278 s        1        2       no
[root@web181 ~]# perl memcached-tool 127.0.0.1:11211 display
  1       96 B       278 s        1        2       no
```

以上过程格式化输出的 stats 命令的信息非常规整，如图 3-7 所示。

总的来说，对单个或少量的 Memcached 服务器的维护相对容易，如果 Memcached 量非常大（这里说的量是指由单个 Memcached 运行 100GB 的数据容量或者有多台大容量的 Memcached 运行服务时），如何组织这些资源，以及在某一个 Memcached 服务器宕机时，如何能保证后端的数据库压力不会产生瓶颈，这是需要考虑的问题。在进行大规模部署时，需要根据业务分配不同的 Memcached 服务，同一类型的业务使用相同的资源，并且记录这些业务使用的端口资源，定期分析这些资源是否足够，以及利用一致性散列算法来确保在扩容 Memcached 的情况下，尽量减少对数据库的影响。

```
$ ./memcached_tool 127.0.0.1:11211 stats
#127.0.0.1:11211 Field        Value
                   bytes          135
              bytes_read         5964
           bytes_written        51394
                 cmd_get            3
                 cmd_set           54
     connection_structures            4
          curr_connections            3
               curr_items            1
                evictions            0
                 get_hits            0
               get_misses            3
             limit_maxbytes   2147483648
                      pid         7186
             pointer_size           64
             rusage_system     0.002999
               rusage_user     0.003999
                  threads            1
                     time   1238401521
      total_connections          112
               total_items           54
                   uptime         1872
                  version        1.2.6
```

图 3-7　格式化输出的 stats 命令的信息

3.3.2　Memcached 的监控

部署好 Memcached 之后，需要对 Memcached 的使用情况进行跟踪：监控 Memcached 的状态信息、内存使用情况、hit/miss 的值。通过

对Memcached的监控不仅能立刻了解M emcached出现问题，还能够利用状态信息来分析业务数据的增长并为未来的容量规划提供依据。

监控Memcached的工具很多，目前常用的监控工具有：memcache.php、Nagios和cacti。下面分别对这三种监控方法进行介绍。

1. 利用memcache.php对单台Memcached进行监控

利用memcache.php进行监控是最简单的监控方法，只要机器支持PHP环境即可。这种监控的执行过程是：把这个文件放到可以访问的目录中，然后对这个memcache.php文件进行修改。下载此文件的网址如下：

```
http://livebookmark.net/memcachephp/memcachephp.zip
```

在使用memcache.php文件之前需要先修改用于访问的用户名和密码。

```
vi memcache.php
.....
define('ADMIN_USERNAME','memcache');     // 定义用户名
define('ADMIN_PASSWORD','password');     // 定义密码
.....
$MEMCACHE_SERVERS[] = 'mymemcache-server:11211'; // 定义要查看的IP和端口
$MEMCACHE_SERVERS[] = 'mymemcache-server2:11212'; // 可添加多个
```

修改自己定义的用户名和密码之后，就可以通过URL访问这个php文件了。输入用户名和密码后Memcached的状态图如图3-8所示。

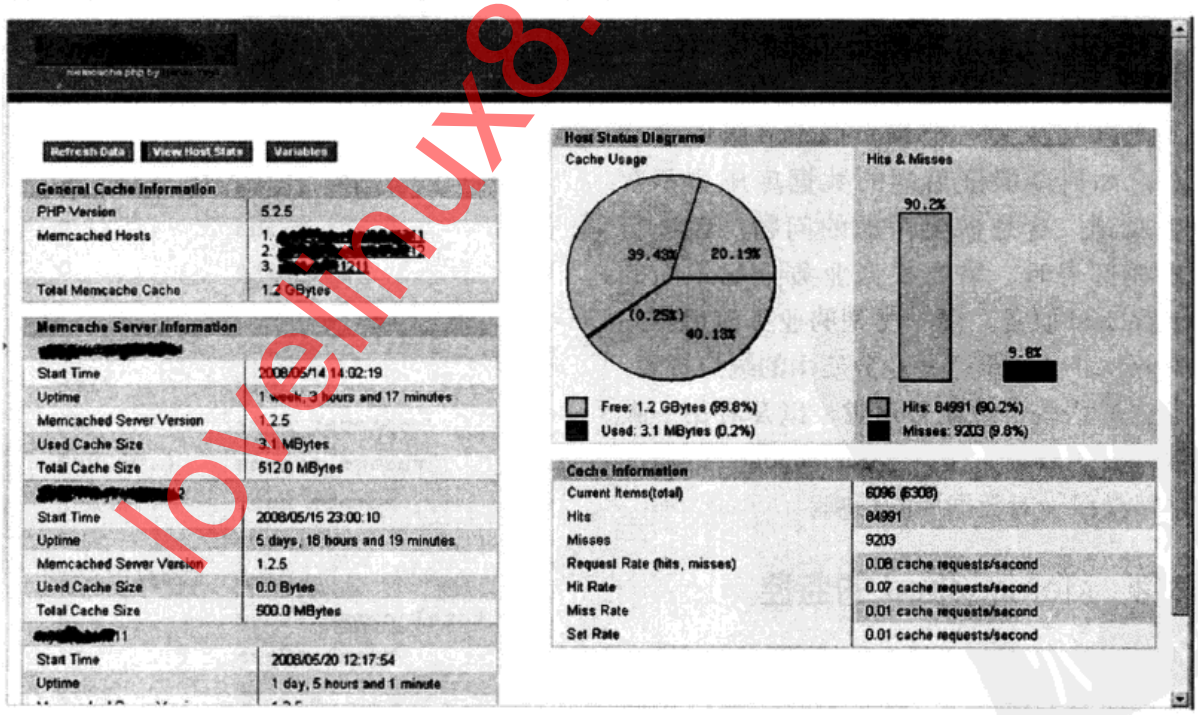

图3-8 利用memcache.php监控Memcached的状态图

图 3-8 很直观地反映了当前 Memcached 的使用情况、命中和不命中的比例等各种状态。

2. 利用 Nagios 监控 Memcached

利用 Nagios 来监控 Memcached 的方式有两种：一种方式是直接使用 check_tcp(mixi) 的方法来监控，代码如下：

```
define command {
command_name check_memcached
command_line $USER1$/check_tcp H$HOSTADDRESS$ -p 11211 -t 5 -E
-s 'stats\r\nquit\r\n' -e 'uptime' -M crit
}
```

这种方式很简单，可以直接使用，不需要安装其他插件。

另一种方式是通过 Nagios 的 Memcached 监控插件可以详细地监视 Memcached 的 get 和 add 等动作。例如，可以通过 stats 命令来确认 Memcached 的运行状态。下载 Nagios 的 Memcached 监控插件的网址如下：

```
http://search.cpan.org/CPAN/authors/id/Z/ZI/ZIGOROU/Nagios-Plugins-Memcached-
    0.02.tar.gz
```

编译安装 Nagios 的 Memcached 监控插件后，通过 Nagios 的 check_memcached 直接访问 Memcached 服务器的监听端口。当然，也可以在将此插件安装在 Memcached 服务器后利用 check_nrpe 来获取 Memcached 服务器的状态信息，过程如下：

```
define command {
        command_name check_memcached_11211
        command_line $USER1$/check_memcached -H 192.168.1.221:11211 --size-warning
            80 --size-critical 90
         }
```

配置完成后，在 Memcached 的界面会看到如图 3-9 所示的信息。

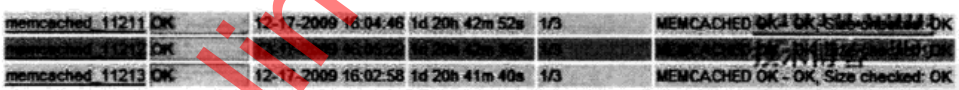

图 3-9　Memcached 在 Nagios 下的监控状态

也可以将 stats 目录的结果通过 rrdtool 转化成图形，以进行性能监视，并将每天的内存使用量做成报表，通过邮件发送出去。

3. 利用 cacti 监控 Memcached

在 cacti 中采用一套监控 Memcached 的模板，可以图形化显示 Memcached 当前的状态信息。这个模板可以从以下网址下载：http://dealnews.com/developers/cacti/memcached.htmls

这里不介绍配置过程。配置完成后 cacti 监控 Memcached 的效果图如图 3-10、图 3-11、图 3-12 和图 3-13 所示。

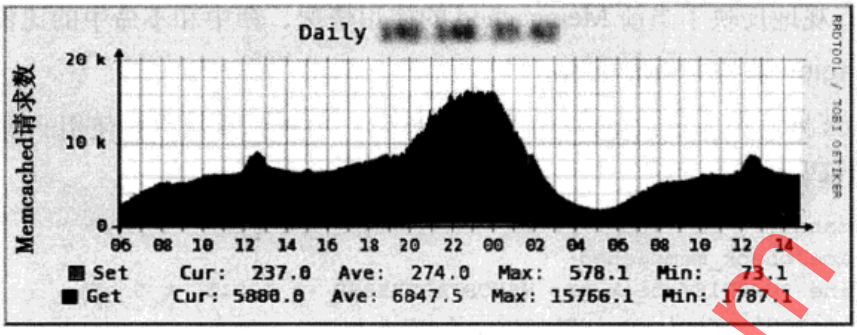

图 3-10 监控 Memcached 的请求数

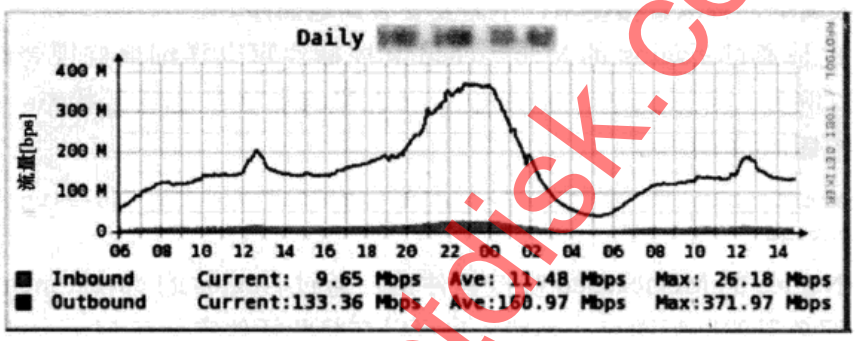

图 3-11 监控 Memcached 的流量状态

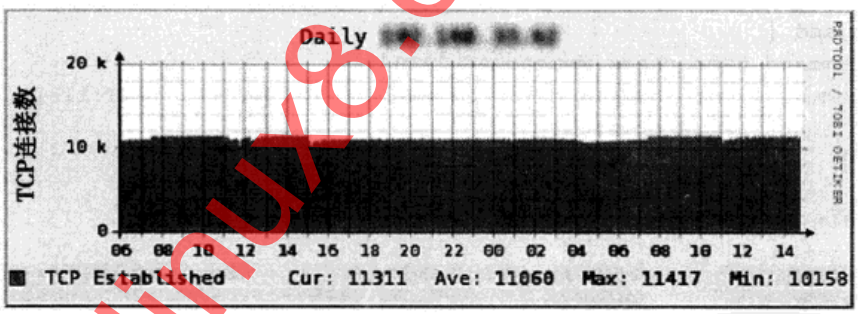

图 3-12 监控 Memcached 的 TCP 连接数

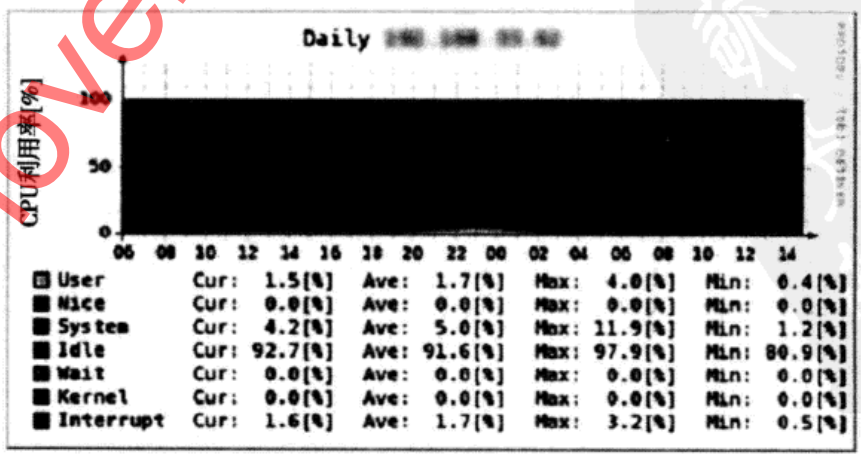

图 3-13 监控 Memcached 的 CPU 的利用率

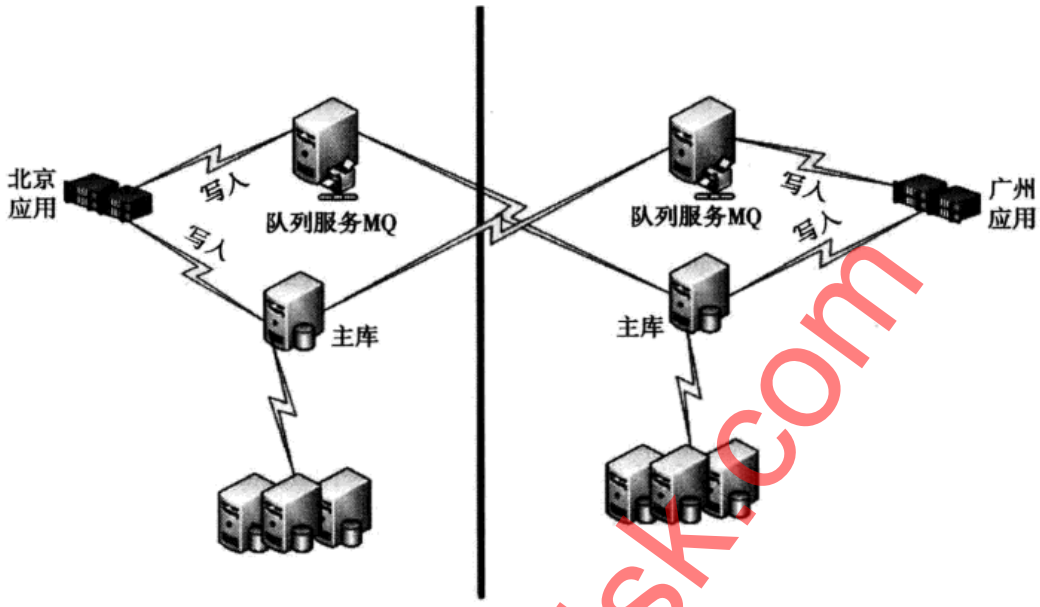

图 3-14 memcacheQ 在数据库的多机房分布式部署环境中的结构

消息队列服务还能使一个有波峰的业务转化成一条直线，这对利用资源非常有好处，只需要准备直线的资源，不需要准备到波峰的资源。Twitter 之前通过 RabbitMQ 来实现消息队列服务，现在改为通过 Kestrel 来实现消息队列服务，类似的消息队列服务产品还有 ActiveMQ 和 ZeroMQ 等。

3.4 通过 UDFs 实现 Memcached 与 MySQL 的自动更新

3.4.1 UDFs 使用简介

UDFs 是 User Defined Functions 的缩写，表示 MySQL 的用户定义函数，应用程序可以利用这些函数从 MySQL 5.0 以上版本的数据库中访问 Memcached 写入或者获取的数据。此外，MySQL 从 5.1 版本开始支持触发器，从而可以在触发器中使用 UDFs 直接更新 Memcached 的内容，这种方式降低了应用程序设计和编写的复杂性。下面简单介绍 UDFs 的安装和使用。

安装 UDFs 需要在数据库服务器上安装两个包，分别是 libmemcached 和 memcached_functions_mysql，这两个包都可以从 http://download.tangent.org/ 下载。安装过程如下。

1）需要的软件有 memcached-1.2.6、libevent-1.4.4-stable、libmemcached-0.30，这些软件的安装非常简单，因此不做说明。

2）安装 mysql5.1，也不做说明。

3）安装 memcached_functions_mysql。基本步骤如下：

```
| memc_stats                    | libmemcached_functions_mysql.so |
| memc_stat_get_keys            | libmemcached_functions_mysql.so |
| memc_stat_get_value           | libmemcached_functions_mysql.so |
+-------------------------------+---------------------------------+
32 rows in set (0.00 sec)
```

3.4.2 memcached_functions_mysql 应用实例

下面通过一个具体的实例来演示 memcached_functions_mysql 的使用方法。

1. 创建两张表

新建两张表：urls 和 results，更新 urls 表中的内容，使系统自动更新 Memcached 的内容。results 用来记录更新 Memcached 失败的记录。

SQL 代码如下：

```
use tests;
drop table if exists urls;
CREATE TABLE `urls` (
  `id` int(10) NOT NULL,
  `url` varchar(255) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
);

drop table if exists results;
CREATE TABLE `results` (
  `id` int(10) NOT NULL,
  `result` varchar(255) NOT NULL DEFAULT 'error',
  `time` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
);
```

2. 建立 3 个 trigger

当向 urls 表中插入数据时，对 Memcached 执行 set 操作。trigger 的代码如下：

```
DELIMITER //
DROP TRIGGER IF EXISTS url_mem_insert;
CREATE TRIGGER url_mem_insert
BEFORE INSERT ON urls
FOR EACH ROW BEGIN
    set @mm = memc_set(NEW.id, NEW.url);
    if @mm <> 0 then
    insert into results(id) values(NEW.id);
    end if;

END //
DELIMITER ;
```

当对 urls 表中的数据进行更新时，对 Memcached 执行 replace 操作。trigger 代码如下：

```
DELIMITER //
DROP TRIGGER IF EXISTS url_mem_update;
CREATE TRIGGER url_mem_update
BEFORE UPDATE ON urls
FOR EACH ROW BEGIN
    set @mm = memc_replace(OLD.id,NEW.url);
    if @mm <> 0 then
    insert into results(id) values(OLD.id);
    end if;

END //
DELIMITER ;
```

当对 urls 表中的数据进行删除操作时，对 Memcached 执行 delete 操作。trigger 代码如下：

```
DELIMITER //
DROP TRIGGER IF EXISTS url_mem_delete;
CREATE TRIGGER url_mem_delete
BEFORE DELETE ON urls
FOR EACH ROW BEGIN
    set @mm = memc_delete(OLD.ID);
    if @mm <> 0 then
    insert into results(id) values(OLD.id);
    end if;

END //
DELIMITER ;
```

3. 设置 Memcached 相关参数

设置 UDFs 操作 Memcaced 服务器的 IP 地址和端口。

```
mysql>SELECT memc_servers_set('192.168.1.184:11900');
+--------------------------------------+
| memc_servers_set('192.168.1.184:11900') |
+--------------------------------------+
|                                    0 |
+--------------------------------------+
1 row in set (0.00 sec)

mysql>select memc_server_count();
+----------------------+
| memc_server_count() |
+----------------------+
|                    1 |
+----------------------+
1 row in set (0.00 sec)
```

在 MySQL 命令行中列出可以修改 Memcached 参数的行为，执行的命令和输出结果如下：

```
mysql>select memc_list_behaviors()\G
*************************** 1. row ***************************
```

```
memc_list_behaviors():
MEMCACHED SERVER BEHAVIORS
MEMCACHED_BEHAVIOR_SUPPORT_CAS
MEMCACHED_BEHAVIOR_NO_BLOCK
MEMCACHED_BEHAVIOR_TCP_NODELAY
MEMCACHED_BEHAVIOR_HASH
MEMCACHED_BEHAVIOR_CACHE_LOOKUPS
MEMCACHED_BEHAVIOR_SOCKET_SEND_SIZE
MEMCACHED_BEHAVIOR_SOCKET_RECV_SIZE
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS
MEMCACHED_BEHAVIOR_KETAMA
MEMCACHED_BEHAVIOR_POLL_TIMEOUT
MEMCACHED_BEHAVIOR_RETRY_TIMEOUT
MEMCACHED_BEHAVIOR_DISTRIBUTION
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS
MEMCACHED_BEHAVIOR_USER_DATA
MEMCACHED_BEHAVIOR_SORT_HOSTS
MEMCACHED_BEHAVIOR_VERIFY_KEY
MEMCACHED_BEHAVIOR_CONNECT_TIMEOUT
MEMCACHED_BEHAVIOR_KETAMA_WEIGHTED
MEMCACHED_BEHAVIOR_KETAMA_HASH
MEMCACHED_BEHAVIOR_BINARY_PROTOCOL
MEMCACHED_BEHAVIOR_SND_TIMEOUT
MEMCACHED_BEHAVIOR_RCV_TIMEOUT
MEMCACHED_BEHAVIOR_SERVER_FAILURE_LIMIT
MEMCACHED_BEHAVIOR_IO_MSG_WATERMARK
MEMCACHED_BEHAVIOR_IO_BYTES_WATERMARK

1 row in set (0.00 sec)
```

设置 MEMCACHED_BEHAVIOR_NO_BLOCK 为打开状态，这样在 Memcached 出现问题（不能连接）时，数据可以继续插入 MySQL 中，但有报错提示；如果不设置此值，那么 Memcached 失败时，数据需要等到 Memcached 失败超时后才可以插入到表中。

通过下面的设置，可以避免这种情况的发生。

```
mysql>select memc_servers_behavior_set('MEMCACHED_BEHAVIOR_NO_BLOCK','1');
+-----------------------------------------------------------------+
| memc_servers_behavior_set('MEMCACHED_BEHAVIOR_NO_BLOCK','1') |
+-----------------------------------------------------------------+
|                                                               0 |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)

mysql>select memc_servers_behavior_set('MEMCACHED_BEHAVIOR_TCP_NODELAY','1');
+-------------------------------------------------------------------+
| memc_servers_behavior_set('MEMCACHED_BEHAVIOR_TCP_NODELAY','1') |
+-------------------------------------------------------------------+
|                                                                 0 |
+-------------------------------------------------------------------+
1 row in set (0.00 sec)
```

3.4.3　对 memcached_functions_mysql 的简单功能进行测试

1）向表 urls 中插入数据，然后查看 Memcached 是否对数据执行 set 操作。

```
mysql>insert into urls (id,url) values (1, 'http://www.test.com.cn');
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> select memc_get('1');
+------------------------+
| memc_get('1')          |
+------------------------+
| http://www.test.com |
+------------------------+
1 row in set (0.00 sec)


1>telnet 192.168.1.184 11900
Trying 192.168.1.184...
Connected to 192.168.1.184 (192.168.1.184).
Escape character is '^]'.
get 1
VALUE 1 0 22
http://www.test.com.cn
END
```

2）更新表 urls 里面的数据，然后查询 Memcached 中是否也进行了更新。

```
mysql>update test.urls set url='http://blog.test.com.cn' where id=1;
Query OK, 1 row affected, 1 warning (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select memc_replace('1','http://blog.test.com.cn');
+---------------------------------------------+
| memc_replace('1','http://blog.test.com.cn') |
+---------------------------------------------+
|                                           0 |
+---------------------------------------------+
1 row in set (0.00 sec)

mysql> select memc_get('1');
+------------------------+
| memc_get('1')          |
+------------------------+
| http://blog.test.com.cn |
+------------------------+
1 row in set (0.00 sec)

1>telnet 192.168.1.184 11900
Trying 192.168.1.184...
Connected to 192.168.1.184 (192.168.1.184).
```

```
Escape character is '^]'.
get 1
VALUE 1 0 23
http://blog.test.com.cn
END
```

3）删除表 urls 中的数据，然后查看 Memcached 是否也将该数据删除了。

```
mysql>delete from test.urls where id=1;
Query OK, 1 row affected, 1 warning (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select memc_get('1');
+---------------+
| memc_get('1') |
+---------------+
| NULL          |
+---------------+
1 row in set (0.00 sec)

1>telnet 192.168.1.184 11900
Trying 192.168.1.184...
Connected to 192.168.1.184 (192.168.1.184).
Escape character is '^]'.
get 1
END
```

3.4.4 使用 memcached_functions_mysql 的经验与技巧

memcached_functions_mysql 使用起来比较简单，但是由于环境的差别，在实践过程中可能会遇到诸多的问题。下面总结了一些在使用 memcached_functions_mysq 过程中可能出现的问题和注意事项。

1. MySQL 重启问题

如果 MySQL 服务器出现重启，需要重新设置连接 Memcached 关系（SELECT memc_servers_set('192.168.1.184:11900')）。

2. 程序 BUG 问题

memcached_functions_mysql 的源程序有可能存在 bug，并且会导致 MySQL 的失败。针对这个问题，读者要尽量选择源程序的稳定版本。

3. 网络因素

网络因素是指 MySQL 和 Memcached 是否处在同一个 IDC、它们之间的网络性能是否很好。网络性能越好，则速度越快。使用本机的 Memcached 可以适当减少网络开销。

4. 插入的数据量

插入数据量的大小包含两个方面：向 MySQL 插入每条记录的大小，向 Memcached 中更

新数据的大小。更新 MySQL、Memcached 的数据越大，更新的速度越慢。因此，要做好前期规划。

5. 延时问题

如果 MySQL 所在的机器使用的资源比较大，会导致更新 Memcached 过于缓慢，即出现类似 m/s 的延时问题。

6. 容灾问题

如果 MySQL 和 Memcached 中有宕机情况出现时，需要考虑怎么恢复，根据前一小节的测试可以这样考虑：建一张错误表，如果在更新 mc 时出现问题，自动把更新错误的记录插到这张表中，通过查询这张表，可以知道哪些数据在什么时间出现过更新错误。

如果 memcached_functions_mysql 应用于生产环境，需要考虑监控和出现问题时的恢复工作（写好脚本以完善这个工作）。

7. MySQL 自身因素

如执行的 MySQL 语句的效率以及连接 MySQL 的 client 程序（php）的连接开销等，这些问题都需要考虑。

3.5　本章小结

本章先介绍 Memcached 的特征，然后介绍了 Memcached 安装过程，接着详细介绍了 Memcached 的工作原理和性能监控方法，最后介绍了 Memcached 的几种升级产品和 Memcached 的使用经验。通过本章的学习，读者能够熟练掌握 Memcached 的安装和使用方法，并且了解 Memcached 的特征及工作机制。

Memcached 是一套分布式内存对象缓存系统，用于在动态系统中减少数据库负载，进而提升性能。Memcached 在很多时候是作为数据库的前端 cache 来使用的，因为它比数据库少了很多 SQL 解析和磁盘操作等方面的开销，而且使用内存来管理数据，所以它可以提供比直接读取数据更好的性能。另外，Memcached 也经常作为服务器之间数据共享的存储媒介。

学习完本章，相信读者已经对 Memcached 有了一个全面的了解，为构建一套属于自己的分布式内存对象缓存系统做好了准备。

第 2 篇

数据备份恢复篇

第 4 章　开源网络备份软件 bacula

本章主要介绍开源备份软件 bacula 的使用与管理技巧。先介绍 bacula 的功能特点和应用范围，接着介绍 bacula 的工作原理及运行机制，然后详细介绍 bacula 的安装和具体配置，同时还讲解 bacula 的基本维护技巧。最后通过实例介绍 bacula 在完全备份、增量备份、差异备份、完全恢复、不完全恢复时的具体操作步骤。学习完本章内容，读者可以熟练使用 bacula。

4.1　bacula 总体概述

4.1.1　bacula 是什么

bacula 是一款开源的跨平台网络备份工具，它提供了基于企业级的客户端 / 服务器的备份恢复解决方案。通过它，系统管理人员可以对数据进行备份、恢复，以及完整性验证等操作。同时，它还提供了许多高级存储管理功能，使系统管理人员能够很容易地发现并恢复丢失的或已经损坏的文件。bacula 既有 Windows 版本的，也有 Linux 和 UNIX 版本的。

4.1.2　bacula 适合哪些用户

如果业务系统数据量巨大，每天都在迅速增长，还需要以 tar 打包方式进行低级备份，并且没有相应的异地容灾策略时，那么就应该考虑使用 bacula。bacula 拥有一个完美的增量备份功能，同时还支持远程容灾备份。通过 bacula，可以将数据备份到任意一个远程主机上，用户只需要对 bacula 进行简单的设置即可自动完成数据备份。

如果用户已经拥有一套存储设备，如磁盘阵列、磁带或带库，只是需要将业务数据从服务器自动备份到这些存储设备上，bacula 无疑也是最佳选择。因为 bacula 具有介质管理功能，利用它可以轻松地实现将服务器数据保存到一个或者多个已经挂载的磁带或带库中。虽然商业的备份软件也能完成将数据自动备份到存储设备上，但代价昂贵。

对于正在使用一个商业的备份软件如 legato 和 Veritas 等的用户，更应该尝试一下 bacula，因为 bacula 完全可以和这些商业软件相媲美，更重要的是，bacula 是开源软件，如果某些关键功能无法实现，可以选择修改开源软件代码的方式来实现。通过对开源软件进行简单的修改来满足特殊需求，将能够大大地简化用户的工作。

4.1.3　bacula 的功能特点

1. 支持多种备份方式

（1）完全备份

完全备份就是完整地备份业务数据。例如，星期一用一盘磁带对整个业务系统进行备份，星期二用另一盘磁带对整个业务系统进行备份，依此类推。

这种备份策略的优点是：当发生数据丢失时，只要用一盘磁带（即灾难发生前一天的备份磁带）就可以恢复丢失的数据。当然，它也有不足之处。首先，由于每天都对整个系统进行完全备份，难免造成备份数据大量重复。这些重复的数据占用了大量的磁带空间，这对用户来说就意味着增加成本。其次，如果备份的数据量很大，那么备份所需的时间也就较长。对于一些业务繁忙、备份时间有限的企业来说，选择这种备份策略是不明智的。

（2）增量备份

增量备份是以上次备份为基准的备份方式，也就是只对每天新增的或被修改过的数据进行备份，例如，星期天进行一次完全备份，星期一只备份从星期天到星期一期间增加的数据，星期二备份从星期一到星期二期间增加的数据，依此类推。

这种备份策略的优点是：因为只备份当天更新或者增加的数据，因而数据量小，节省了磁带空间，缩短了备份时间。当然，它也是有缺点的。当灾难发生时，数据的恢复过程比较麻烦。如果系统在星期五的早晨发生故障，丢失了大量的数据，那么就要将系统恢复到星期四时的状态。这时系统管理员首先要找到星期天的完全备份进行系统恢复，然后找到星期一的备份来恢复星期一的数据，接着找到星期二的备份来恢复星期二的数据。按照这种方式，一直恢复星期四的数据为止。很明显，这种方式很繁琐，备份的可靠性也很差。在这种备份方式下，各个备份间的关系就像一个链子，环环相扣，其中任何一个备份出了问题都会导致整条链子脱节。在上例中，若星期三的备份出了故障，那么管理员最多只能将系统数据恢复到星期二时的状态。

（3）差异备份

差异备份是以完全备份为基准的一种备份方式。例如，系统管理员在星期天对系统进行一次完全备份，在星期一备份星期天到星期一期间的数据，在星期二备份星期天到星期二期间的数据，依此类推。也就是备份当天所有与星期天不同的数据（新的或修改过的）。

差异备份方式避免了上面两种备份策略的缺陷，同时，又具有以上两种备份方式的所有优点。首先，它无需每天都对系统做完全备份，因此备份数据量小，备份所需时间短，并节省空间；其次，它在灾难恢复时也很方便，只需要两个备份（即完全备份与灾难发生前一天的备份）就可以将系统恢复。

其实每种备份方式都不是孤立存在的，在实际的备份应用中，通常采用以上三种方式相结合的备份策略。例如每星期一至星期六进行一次增量备份或差异备份，每星期日进行完全备份，每月底进行一次完全备份，每年底进行一次完全备份。

通过对三种备份方式的介绍，可以知道每种备份的数据量是不同的：完全备份 > 差异备份 > 增量备份。因而，在进行数据恢复时，使用的数据也不尽相同。如果使用完全备份的方式，只需要利用上次的完全备份就可以恢复所有数据；如果使用完全备份 + 增量备份的方式，则需要利用上次的完全备份 + 上次完全备份后的所有增量备份才能恢复所有数据；如果使用完全备份 + 差异备份的方式，则只需要利用上次的完全备份 + 最近的一个差异备份就可以恢复所有数据。

2. 支持多种恢复方式

- ❑ 可以恢复某个目录、文件到指定的位置，恢复时自动恢复数据的原始结构。
- ❑ 可以恢复所有数据到指定位置，恢复时自动恢复数据的原始结构。
- ❑ 可以保存恢复文件或目录的权限、属主、访问时间等属性。
- ❑ 可以恢复某个时间点的备份到指定位置，恢复时自动恢复数据的原始结构。

3. 支持多种文件系统下的备份与恢复

bacula 支持的文件系统有：ext3、ext2、reiserfs、xfs、jfs、smbfs、iso9660 和 ntfs 等。

4. 支持各种备份介质

- ❑ 支持把备份写到磁盘。
- ❑ 支持把备份写到磁带。
- ❑ 支持把备份写到磁盘阵列。
- ❑ 支持把备份写到光盘。

5. 支持多种操作系统

- ❑ Linux（RHEL/SUSE/CentOS）。
- ❑ UNIX。
- ❑ Mac。
- ❑ Windows（Windows 98、Windows Me、Windows NT、Windows XP、Windows 2000 和 Windows 2003）。

6. 强大的内部功能

- ❑ 支持定时备份，无需人工干预。
- ❑ 支持终端命令控制，更加灵活。
- ❑ 支持正则表达式，可以对备份文件进行更严格的匹配。
- ❑ 支持 MD5 和 SHA1 两种签名校验。
- ❑ 支持压缩备份，备份效率更高，传输更快。
- ❑ 支持报表自动绘制功能，可以自动生成备份报表和恢复报表。

4.1.4　bacula 的工作原理

1. bacula 基本组成

一个完整的 bacula 备份系统，由下面 5 个部分组成。

- Director Daemon：负责监听所有的备份、恢复、验证、存档事务，以及定制备份和恢复文件计划等，并将整个系统运行状况记录在一个数据库中。支持 Director Daemon 的数据库有 MySQL、PostgreSQL 和 SQLite，推荐使用 MySQL。其配置文件为 bacula-dir.conf。
- Storage Daemon（SD）：在备份数据时，用来指定备份和恢复数据的存储介质（存储介质可以是本地磁盘、光纤磁盘阵列、磁带和 DVD 等），主要负责将数据备份到存储介质上；而在数据恢复时，负责将数据从存储介质中传送出去。其配置文件为 bacula-sd.conf。
- File Daemon（FD）：是一个安装在需要备份数据的机器上的守护进程，在备份数据时，它负责把文件传出；在恢复数据时，它负责接收数据并执行恢复操作。其配置文件为 bacula-fd.conf。
- Console：是一个管理控制台，用户可以通过这个控制台连接到 Director Daemon 进行管理备份与恢复操作。有三种管理方式：基于文本的控制台界面、GNOME 的界面和 wxWidgets 的图形界面。其配置文件是 bconsole.conf。
- Monitor：是一个进程监控端，负责监控 Director Daemon、Storage Daemon 和 File Daemon 的守护进程。

bacula 备份系统的组成如图 4-1 所示。

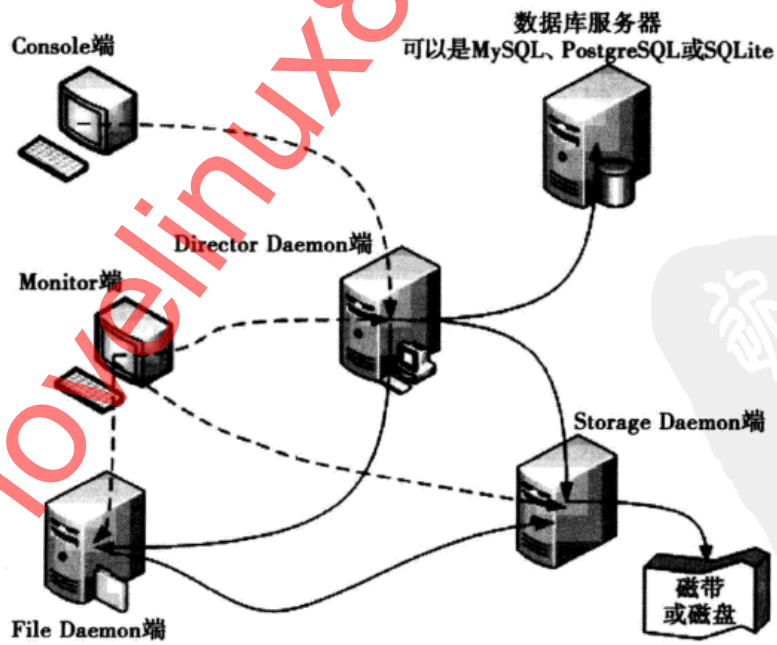

图 4-1　bacula 备份系统的组成

从图 4-1 中可以看出，bacula 的备份恢复流程如下：

1）通过 Console 连接到 Director 端，备份恢复操作开始。

2）Director 端从自己的数据库中调出记录信息，对存储端 SD 与客户端 FD 的任务进行协调。

3）客户端 FD 负责验证 Director 的操作许可，如果验证通过，则允许连接存储端 SD。

4）客户端 FD 根据 Director 发出的请求去连接 SD，将 FD 端的数据备份到存 SD 指定的存储介质上，或者将 SD 端存储介质中的数据传回到客户端 FD 指定的位置上，完成备份恢复过程。

需要注意的是，在 bacula 的整个备份恢复系统中，客户端 FD 和 SD 要保证网络连接畅通，为了保证备份以及恢复数据的速度和效率，最好让客户端 FD 和 SD 处在一个网段中。

2. bacula 各个组成部分的关联性

在 baclua 的 5 个组成部分中，3 个主要配置文件是相互关联的，修改任何一个配置文件，另外两个文件都要进行相应的改动。为了使读者对这 3 个配置文件有更清晰的认识，图 4-2 列出了这 3 个文件之间的相互关系。

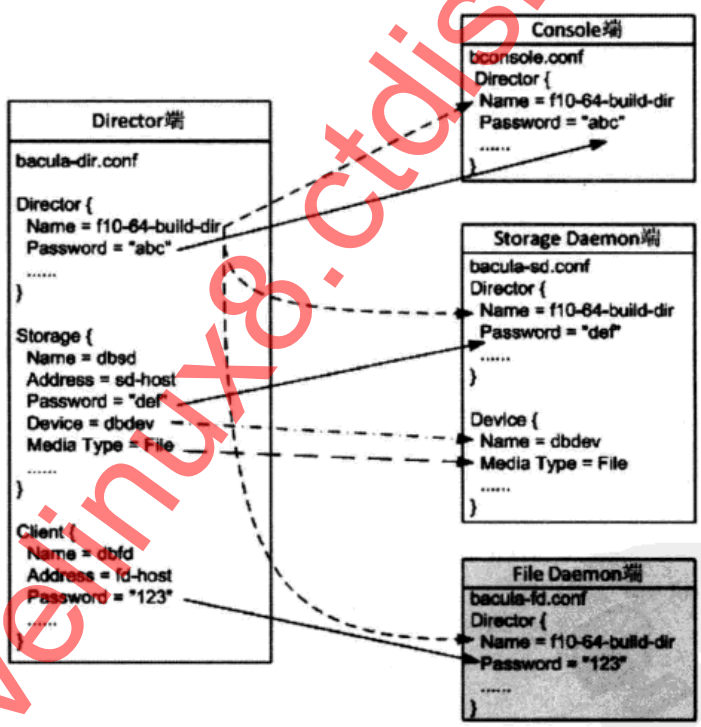

图 4-2 bacula 配置文件之间的关系

4.2 安装 bacula

4.2.1 bacula 的几种网络备份拓扑

前面介绍了 bacula 有 5 个组成部分，在实际的应用中，没有必要将 5 个部分分别放在不

同的服务器上，它们之间的某些部分是可以合并的。常见的 bacula 部署结构有如下几种：

- Director 与 SD 以及 Console 在一台机器上，而客户端 FD 在另外一台机器上，当然客户端 FD 可以在一台或者多台机器上。
- Director 与 Console 在一台机器上，SD 在一台机器上，客户端 FD 在一台或者多台机器上。
- Director 与客户端 FD、SD 以及 Console 端都在一台机器上，也就是服务器自己备份自己，数据保存在本机。

4.2.2　编译与安装 bacula

这里对上一节的第一种 bacula 部署结构进行介绍。环境如表 4-1 所示。

表 4-1　一个 bacula 部署结构的环境

主机名	IP 地址	操作系统	应用角色
baculaServer	192.168.12.188	CentOS release 5.4	Director、SD、Console
baculaClient	192.168.12.189	CentOS release 5.4	FD

整个拓扑结构如图 4-3 所示。

图 4-3　bacula 实例的拓扑结构

1. 在 bacula 服务器端安装 bacula

首先在 http://www.bacula.org 下载相应的源码，这里下载的是 bacula-5.0.1.tar.gz。接着进行编译安装。安装过程如下：

```
[root@baculaServer opt]# tar zxvf bacula-5.0.1.tar.gz
[root@baculaServeropt]# cd bacula-5.0.1
[root@baculaServer bacula-5.0.1]# ./configure --prefix=/opt/bacula --with-mysql=/
    opt/mysql
[root@baculaServer bacula-5.0.1]#make
[root@baculaServer bacula-5.0.1]#make install
```

bacula 需要数据库的支持，这里采用 MySQL 数据库，并假定 MySQL 已经在 bacula 服

务器端安装好了，且MySQL安装路径为/opt/mysql（bacula在编译时通过“--with-mysql”选项指定了MySQL数据库的安装路径）。

bacula安装完成后，所有配置文件默认放在/opt/bacula/etc/目录下。

2. 在bacula客户端安装bacula

由于bacula客户端只是需要备份的客户端，因而只需安装相应的客户端组件即可。过程如下：

```
[root@baculaClient opt]# tar zxvf bacula-5.0.1.tar.gz
[root@baculaClient opt]# cd bacula-5.0.1
[root@baculaClient bacula-5.0.1]# ./configure --prefix=/opt/bacula --enable-client-only
[root@baculaClient bacula-5.0.1]#make
[root@baculaClient bacula-5.0.1]#make install
```

4.2.3 初始化MySQL数据库

在baculaServer上安装完bacula后，还需要创建bacula对应的MySQL数据库以及访问数据库的授权。bacula已经为用户准备好了这样的脚本，只要在bacula服务器端上执行如下脚本即可。

```
[root@localhost bacula-5.0.1]#cd /opt/bacula/etc
[root@localhost etc]# ./grant_mysql_privileges
[root@localhost etc]# ./create_mysql_database
Creation of bacula database succeeded.
[root@localhost etc]# ./make_mysql_tables
Creation of Bacula MySQL tables succeeded.
```

接下来可以登录MySQL数据库，查看bacula的数据库和数据表是否已经建立。在执行上面三行MySQL初始代码时，默认由空密码的root用户执行，因此要确保MySQL数据库root密码为空。

4.3 配置一个bacula备份系统

配置bacula备份系统，其实就是对Director端配置文件bacula-dir.conf、SD配置文件bacula-sd.conf、客户端FD配置文件bacula-fd.conf以及Console端配置文件bconsole.conf进行配置的过程。

根据上面的安装部署，将Director端、SD、Console端集中在一台服务器baculaServer（即192.168.12.188）上，而将客户端FD部署在baculaClient（即192.168.12.189）服务器上。下面详细讲述配置过程。

4.3.1 配置bacula的Console端

Console端的配置文件是bconsole.conf，这个配置文件很简单。配置完的文件如下：

```
Director {
  Name = f10-64-build-dir          #控制端名称，在下面的 bacula-dir.conf 和 bacula-sd.conf
                                   #文件中会被引用
  DIRport = 9101                   #控制端服务器端口
  address = 192.168.12.188         #控制端服务器 IP 地址
  Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"
                                   #控制端密码文件
}
```

4.3.2　配置 bacula 的 Director 端

bacula-dir.conf 是 Director 端的配置文件，也是 bacula 的核心配置文件，这个文件非常复杂，共分为 10 个逻辑段，分别是：

- Director，定义全局设置。
- Catalog，定义后台数据库。
- Jobdefs，定义默认执行任务。
- Job，自定义一个备份或者恢复任务。
- Fileset，定义备份哪些数据，不备份哪些数据。
- Schedule，定义备份时间策略。
- Pool，定义供 Job 使用的池属性。
- Client，定义要备份的主机地址。
- Storage，定义数据的存储方式。
- Messages，定义发送日志报告和记录日志的位置。

下面是一个已经配置好的文件。

```
Director {                         #定义 bacula 的全局配置
  Name = f10-64-build-dir
  DIRport = 9101                   #定义 Director 的监听端口
  QueryFile = "/opt/bacula/etc/query.sql"
  WorkingDirectory = "/opt/bacula/var/bacula/working"
  PidDirectory = "/var/run"
  Maximum Concurrent Jobs = 1      #定义一次能处理的最大并发数

 #验证密码，这个密码必须与 bconsole.conf 文件中对应的 Director 逻辑段密码相同
  Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"

 #定义日志输出方式，“Daemon”在下面的 Messages 逻辑段中进行了定义
  Messages = Daemon
}

Job {                              #自定义一个备份任务
  Name = "Client1"                 #备份任务名称
  Client = dbfd                    #指定要备份的客户端主机，“dbfd”在后面 Client 逻辑段中
                                   #进行定义
```

```
  Level = Incremental          # 定义备份的级别,Incremental 为增量备份。Level 的取值可为 Full(完
                               # 全备份)、Incremental (增量备份) 和 Differential (差异备份), 如
                               # 果第一次没做完全备份, 则先进行完全备份后再执行 Incremental
  Type = Backup                # 定义 Job 的类型, "backup" 为备份任务, 可选
                               # 的类型还有 restore 和 verify 等
  FileSet = dbfs               # 指定要备份的客户端数据, "dbfs" 在后面 FileSet
                               # 逻辑段中进行定义
  Schedule = dbscd             # 指定这个备份任务的执行时间策略, "dbscd" 在
                               # 后面的 Schedule 逻辑段中进行了定义
  Storage = dbsd               # 指定备份数据的存储路径与介质, "dbsd" 在后
                               # 面的 Storage 逻辑段中进行定义
  Messages = Standard
  Pool = dbpool                # 指定备份使用的 pool 属性, "dbpool" 在后面的
                               #  Pool 逻辑段中进行定义
  Write Bootstrap = "/opt/bacula/var/bacula/working/Client2.bsr" # 指定备份的引导信息路径
}

Job {                          # 定义一个名为 Client 的差异备份的任务
  Name = "Client"
  Type = Backup
  FileSet = dbfs
  Schedule = dbscd
  Storage = dbsd
  Messages = Standard
  Pool = dbpool
  Client = dbfd
  Level = Differential         # 指定备份级别为差异备份
  Write Bootstrap = "/opt/bacula/var/bacula/working/Client1.bsr"
}


Job {                          # 定义一个名为 BackupCatalog 的完全备份任务
  Name = "BackupCatalog"
  Type = Backup
  Level = Full                 # 指定备份级别为完全备份
  Client = dbfd
  FileSet="dbfs"
  Schedule = "dbscd"
  Pool = dbpool
  Storage = dbsd
  Messages = Standard
  RunBeforeJob = "/opt/bacula/etc/make_catalog_backup bacula bacula"
  RunAfterJob  = "/opt/bacula/etc/delete_catalog_backup"
  Write Bootstrap = "/opt/var/bacula/working/BackupCatalog.bsr"
}


Job {                          # 定义一个还原任务
  Name = "RestoreFiles"
  Type = Restore               # 定义 Job 的类型为 "Restore ", 即恢复数据
```

```
  Client=dbfd
  FileSet=dbfs
  Storage = dbsd
  Pool = dbpool
  Messages = Standard
  Where = /tmp/bacula-restores                    # 指定默认恢复数据到这个路径
}

FileSet {                                         # 定义一个名为 dbfs 的备份资源，也就是指定需要备
                                                  # 份哪些数据，需要排除哪些数据等，可以指定多个
                                                  # FileSet
  Name = dbfs
  Include {
    Options {
     signature = MD5; Compression=GZIP; }         # 表示使用 MD5 签名并压缩
    File = /cws3                                  # 指定客户端 FD 需要备份的文件目录
  }

  Exclude {                                       # 通过 Exclude 排除不需要备份的文件或者目录，可根
                                                  # 据具体情况修改
    File = /opt/bacula/var/bacula/working
    File = /tmp
    File = /proc
    File = /tmp
    File = /.journal
    File = /.fsck
  }
}

Schedule {                                        # 定义一个名为 dbscd 的备份任务调度策略
  Name = dbscd
  Run = Full 1st sun at 23:05                     # 第一周的星期日晚 23:05 分进行完全备份
  Run = Differential 2nd-5th sun at 23:05         # 第 2~5 周的星期日晚 23:05 进行差异备份
  Run = Incremental mon-sat at 23:05              # 所有星期一至星期六晚 23:05 分进行增量备份
}

FileSet {
  Name = "Catalog"
  Include {
    Options {
     signature = MD5
    }
    File = /opt/bacula/var/bacula/working/bacula.sql
  }
}

Client {                                          #Client 用来定义备份哪个客户端 FD 的数据
  Name = dbfd                                     #Clinet 的名称，可以在前面的 Job 中调用
  Address = 192.168.12.189                        # 要备份的客户端 FD 主机的 IP 地址
```

```
  FDPort = 9102                  # 与客户端 FD 通信的端口
  Catalog = MyCatalog            # 使用哪个数据库存储信息，“MyCatalog” 在后面
                                 # 的 MyCatalog 逻辑段中进行定义
  Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"
                                 # Director 端与客户端 FD 的验证密码，这个值必须与客户端 FD
                                 # 配置文件 bacula-fd.conf 中密码相同
  File Retention = 30 days       # 指定保存在数据库中的记录多久循环一次，这里是 30 天，只
                                 # 影响数据库中的记录，不影响备份的文件
  Job Retention = 6 months       # 指定 Job 的保持周期，应该大于 File Retention 指定的值
  AutoPrune = yes                # 当达到指定的保持周期时，是否自动删除数据库中的记录，
                                 #yes 表示自动清除过期的 Job
}

Client {
  Name = dbfd1
  Address = 192.168.12.188
  FDPort = 9102
  Catalog = MyCatalog
  Password = "Wr8lj3q51PgZ21U2FSaTXICYhLmQkT1XhHbm8a6/j8Bz"
  File Retention = 30 days
  Job Retention = 6 months
  AutoPrune = yes
}


Storage {                        # Storage 用来定义将客户端的数据备份到哪个存储设备上
  Name = dbsd
  Address = 192.168.12.188       # 指定存储端 SD 的 IP 地址
  SDPort = 9103                  # 指定存储端 SD 通信的端口
  Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"
                                 # Director 端与存储端 SD 的验证密码，这个值必须与存储端 SD
                                 # 配置文件 bacula-sd.conf 中 Director 逻辑段密码相同指
  Device = dbdev                 # 定数据备份的存储介质，必须与存储端（这里是 192.168.12.188）
                                 # 的 bacula-sd.conf 配置文件中的 “Device” 逻辑段的 “Name”
                                 # 项名称相同
  Media Type = File              # 指定存储介质的类别，必须与存储端 SD（这里是 192.168.12.188）
                                 # 的 bacula-sd.conf 配置文件中的 “Device” 逻辑段的 “Media
                                 # Type” 项名称相同

}

Catalog {                        # Catalog 逻辑段用来定义关于日志和数据库设定
  Name = MyCatalog
  dbname = "bacula"; dbuser = "bacula"; dbpassword = ""
                                 # 指定库名、用户名和密码
}

Messages {                       # Messages 逻辑段用来设定 Director 端如何保存日志，以及
                                 # 日志的保存格式，可以将日志信息发送到管理员邮箱，前提是必
                                 # 须开启 sendmail 服务

  Name = Standard
```

```
      mailcommand = "/usr/sbin/bsmtp -h localhost -f \"\(Bacula\) \<%r\>\" -s \"Bacula:
%t %e of %c %l\" %r"
      operatorcommand = "/usr/sbin/bsmtp -h localhost -f \"\(Bacula\) \<%r\>\" -s
\"Bacula: Intervention needed for %j\" %r"
      mail = dba.gao@gmail.com = all, !skipped
      operator = exitgogo@126.com = mount
      console = all, !skipped, !saved
      append = "/opt/bacula/log/bacula.log" = all, !skipped
                        # 定义 bacula 的运行日志
      append ="/opt/bacula/log/bacula.err.log" = error,warning, fatal
                        # 定义 bacula 的错误日志
      catalog = all
    }

    Messages {          # 定义了一个名为 Daemon 的 Messages 逻辑段，“Daemon” 已经在前面进行了引用
      Name = Daemon
      mailcommand = "/usr/sbin/bsmtp -h localhost -f \"\(Bacula\) \<%r\>\" -s \"Bacula
daemon message\" %r"
      mail = exitgogo@126.com = all, !skipped
      console = all, !skipped, !saved
      append = "/opt/bacula/log/bacula_demo.log" = all, !skipped
    }


    Pool {                              # 定义供 Job 任务使用的池属性信息，例如，设定备份文件过期时间、
                                        # 是否覆盖过期的备份数据、是否自动清除过期备份等
      Name = dbpool
      Pool Type = Backup
      Recycle = yes                     # 重复使用
      AutoPrune = yes                   # 表示自动清除过期备份文件
      Volume Retention = 7 days         # 指定备份文件保留的时间
      Label Format ="db-${Year}-${Month:p/2/0/r}-${Day:p/2/0/r}-id${JobId}"
                                        # 设定备份文件的命名格式，这个设定格式所产生的命名文件为：
                                        # db-2010-04-18-id139
      Maximum Volumes = 7               # 设置最多保存多少个备份文件
      Recycle Current Volume = yes      # 表示可以使用最近过期的备份文件来存储新备份
      Maximum Volume Jobs = 1           # 表示每次执行备份任务创建一个备份文件
    }

    Console {                           # 限定 Console 利用 tray-monitor 获得 Director 的状态信息
      Name = f10-64-build-mon
      Password = "RSQy3sRjak3ktZ8Hr07gc728VkZHBr0QCjOC5x3pXEap"
      CommandACL = status, .status
    }
```

4.3.3　配置 bacula 的 SD

SD 可以是一台单独的服务器，也可以和 Director 在一台机器上。本例就将 SD 和 Director 端放在一起进行配置。SD 的配置文件是 bacula-sd.conf，下面是一个已经配置好的 bacula-sd.conf 文件。

```
Storage {                    # 定义存储，本例中是 f10-64-build-sd
  Name = f10-64-build-sd     # 定义存储名称
  SDPort = 9103              # 监听端口
  WorkingDirectory = "/opt/bacula/var/bacula/working"
  Pid Directory = "/var/run"
  Maximum Concurrent Jobs = 20
}

Director {                             # 定义一个控制 StorageDaemon 的 Director
  Name = f10-64-build-dir              # 这里的“Name”值必须和 Director 端配置文件
                                       #bacula-dir.conf 中 Director 逻辑段名称相同
  Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"    # 这里的“Password”值
   # 必须和 Director 端配置文件 bacula-dir.conf 中 Storage 逻辑段密码相同
}

Director {                             # 定义一个监控端的 Director
  Name = f10-64-build-mon              # 这里的“Name”值必须和 Director 端配置文件
                                       #bacula-dir.conf 中 Console 逻辑段名称相同
  Password = "RSQy3sRjak3ktZ8Hr07gc728VkZHBr0QCjOC5x3pXEap"     # 这里的“Password”
   # 值必须和 Director 端配置文件 bacula-dir.conf 中 Console 逻辑段密码相同
  Monitor = yes
}

Device {                               # 定义 Device
  Name = dbdev                         # 定义 Device 的名称，这个名称在 Director 端配置文件
                                       #bacula-dir.conf 中的 Storage 逻辑段 Device 项中被引用
  Media Type = File                    # 指定存储介质的类型，File 表示使用文件系统存储
  Archive Device = /webdata            #Archive Device 用来指定备份存储的介质，可以
          # 是 cd、dvd、tap 等，这里是将备份的文件保存在 /webdata 目录下
  LabelMedia = yes;                    # 通过 Label 命令来建立卷文件
  Random Access = yes;                 # 设置是否采用随机访问存储介质，这里选择 yes
  AutomaticMount = yes;                # 表示当存储设备打开时，是否自动使用它，这里选择 yes
  RemovableMedia = no;                 # 是否支持可移动的设备，如 tap 或 cd，这里选择 no
  AlwaysOpen = no;                     # 是否确保 tap 设备总是可用，这里没有使用 tap 设备，
                                       # 因此设置为 no
}

Messages {                             # 为存储端 SD 定义一个日志或消息处理机制
  Name = Standard
  director = f10-64-build-dir = all
}
```

4.3.4 配置 bacula 的 FD 端

客户端FD运行在一台独立的服务器上，在本例中是baculaclient主机（即192.168.12.189），它的配置文件是bacula-fd.conf。配置好的文件如下：

```
Director {                              # 定义一个允许连接 FD 的控制端
  Name = f10-64-build-dir               # 这里的“Name”值必须和 Director 端配置文件
                                        #bacula-dir.conf 中 Director 逻辑段名称相同
  Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"
     # 这里的“Password”值必须和 Director 端配置文件 bacula-dir.conf 中 Client 逻辑段密码相同
}

Director {                              # 定义一个允许连接 FD 的监控端
  Name = f10-64-build-mon
  Password = "RSQy3sRjak3ktZ8Hr07gc728VkZHBr0QCjOC5x3pXEap"
  Monitor = yes
}

FileDaemon {                            # 定义一个 FD 端
  Name = localhost.localdomain-fd
  FDport = 9102                         # 监控端口
  WorkingDirectory = /opt/bacula/var/bacula/working
  Pid Directory = /var/run
  Maximum Concurrent Jobs = 20          # 定义一次能处理的并发作业数
}

Messages {                              # 定义一个用于 FD 端的 Messages
  Name = Standard
  director = localhost.localdomain-dir = all, !skipped, !restored
}
```

4.4　启动与关闭 bacula

4.4.1　启动 bacula 的 Director daemon 与 Storage daemon

完成上面的配置后，就可以启动或关闭 bacula 了。在 baculaserver 上启动或关闭控制端的所有服务有如下两种方式。

第一种方式如下：

```
[root@baculaserver  etc]# /opt/bacula/sbin/bacula
 {start|stop|restart|status}
```

第二种方式是通过分别管理 bacula 各个配置端的方式，依次启动或者关闭每个服务。

```
[root@baculaserver  etc]# /opt/bacula/etc/bacula-ctl-dir  {start|stop|restart|status}
[root@baculaserver  etc]# /opt/bacula/etc/bacula-ctl-sd  {start|stop|restart|status}
[root@baculaserver  etc]# /opt/bacula/etc/bacula-ctl-fd  {start|stop|restart|status}
```

由于将客户端 FD 配置到了另一个主机 baculaclient 上，因此无需在 baculaserver 上启动 File daemon 服务。

启动 bacula 的所有服务后，可以通过 netstat 命令，观察启动端口情况。

```
[root@localhost etc]# netstat -antl |grep 91
tcp  0      0 0.0.0.0:9101                0.0.0.0:*                   LISTEN
tcp  0      0 0.0.0.0:9102                0.0.0.0:*                   LISTEN
tcp  0      0 0.0.0.0:9103                0.0.0.0:*                   LISTEN
```

其中，9101 代表 Director daemon；9102 代表 File daemon；9103 代表 Storage daemon。

注意在启动 bacula 的所有服务前，必须先启动 MySQL 数据库，如果 MySQL 数据库没有启动，连接 bacula 的控制端时会报错。

```
 [root@baculaserver opt]# /opt/bacula/sbin/bconsole
Connecting to Director 192.168.12.188:9101
19-04 月  09:45 bconsole JobId 0: Fatal error: bsock.c:135 Unable to connect to
Director daemon on 192.168.12.188:9101. ERR=拒绝连接
```

此时，执行 netstat 命令可以发现，9101 端口根本没有启动。

4.4.2 在客户端 FD 启动 File daemon

最后，在客户端 FD（即 baculaclient）上启动 File daemon 服务。操作如下：

```
[root@baculaclient etc]# /opt/bacula/sbin/bacula start
Starting the Bacula File daemon
```

管理客户端 FD 的服务，也可以通过以下方式完成：

```
[root@baculaclient etc]# /opt/bacula/sbin/bacula {start|stop|restart|status}
[root@ baculaclient  etc]# /opt/bacula/etc/bacula-ctl-fd  {start|stop|restart|status}
```

4.5 实战 bacula 备份恢复过程

下面对 bacula 的实战操作进行介绍，通过实例详细演示 bacula 的完全备份、增量备份、差异备份、完全恢复和不完全恢复的过程。

4.5.1 实例演示 bacula 的完全备份功能

1. 创建卷组

执行如下命令，连接到 bacula 控制端，执行备份恢复操作。

```
[root@baculaserver opt]#/opt/bacula/sbin/bconsole
Connecting to Director 192.168.12.188:9101
1000 OK: f10-64-build-dir Version: 3.0.2 (18 July 2009)
Enter a period to cancel a command
*label
Automatically selected Storage: dbsd
Enter new Volume name: cicro4      # 卷组名称，可随意指定，指定完毕后会在
                                   #bacula-dir.conf 文件的指定位置生成一个 cicro4 文件
Defined Pools:
```

```
     1: dbpool
     2: Scratch
Select the Pool (1-2): 1  #指定卷组的放置位置，即为 SD 名称
                          # Dbpool 在 bacula-dir.conf 中定义
Connecting to Storage daemon dbsd at 192.168.12.188:9103 ...
                          # 由于指定 SD 在 192.168.12.188 主机上，因此将卷 cicro4 创建到此主机上
Sending label command for Volume "cicro4" Slot 0 ...
3000 OK label. VolBytes=191 DVD=0 Volume="cicro4" Device="dbdev" (/cicro/backup2)
Catalog record for Volume "cicro4", Slot 0  successfully created.
Requesting to mount dbdev ...
3906 File device "dbdev" (/cicro/backup2) is always mounted.
```

2. 利用 run 命令执行备份操作

下面继续在 bacula 控制端执行备份操作。

```
*run
A job name must be specified.
The defined Job resources are:
      1: Client1          #这里的 Client1 是在 bacula-dir.conf 中定义的一个 Job 执行任务
      2: Client           #这里的 Client 也是在 bacula-dir.conf 中定义的一个 Job 执行任务
      3: BackupCatalog
      4: RestoreFiles
Select Job resource (1-2): 1
Run Backup job
JobName:  Client1
Level:    Incremental
Client:   dbfd
FileSet:  dbfs
Pool:     dbpool (From Job resource)
Storage:  dbsd (From Job resource)
When:     2009-08-21 13:40:13
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=67
```

到此为止，可以开始执行备份，在 bacula-dir.conf 文件中定义的 Client1 是一个增量备份，因此这个备份只是个增量操作。由于这是第一个备份，因此默认 Client1 会做一个完全备份，第二次备份时，才执行增量备份。

3. 查看备份状态

在 bacula 控制端利用“status”可以查看 bacula 的各种状态。这里查看备份时 Director 端的一个状态信息。

```
*status
Status available for:
      1: Director
      2: Storage
      3: Client
      4: All
```

```
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 13:22, 0 Jobs run since started.
 Heap: heap=241,664 smbytes=82,242 max_bytes=82,498 bufs=245 max_bufs=250

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4
Running Jobs:
Console connected at 21-Aug-09 13:37
 JobId Level   Name                       Status
======================================================================
   67 Full    Client1.2009-08-21_13.40.16_07 is running
```

由此可知，备份正在进行。备份完成再次查看备份信息，输出如下：

```
*status
Status available for:
      1: Director
      2: Storage
      3: Client
      4: All
Select daemon type for status (1-4): 1
```

由于这里 bacula 将备份的存储端 (SD) 和控制端（DIR）设置在一台服务器上，因此可以通过选项 1 “Director” 来查看 SD 的状态。如果 SD 端和 DIR 端不在一台服务器上，要查看选项 2，即 “Storage”。这里选择选项 1 后输出内容如下：

```
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 13:22, 1 Job run since started.
 Heap: heap=241,664 smbytes=73,891 max_bytes=84,825 bufs=212 max_bufs=250

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4

Running Jobs:
Console connected at 21-Aug-09 13:42
No Jobs running.
Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    58  Incr         2    46.85 M   OK       19-Aug-09 16:51 Client1
    59  Incr         2    3.908 M   OK       19-Aug-09 16:51 Client1
    60  Incr         2    8.377 K   OK       19-Aug-09 16:52 Client1
    61  Incr         1          0   OK       19-Aug-09 16:53 Client1
    62  Incr         2    8.344 K   OK       19-Aug-09 16:53 Client1
    63  Incr         2    8.377 K   OK       19-Aug-09 16:54 Client1
```

```
64                3     77.44 K  OK       19-Aug-09 16:57 RestoreFiles
65                1          0  OK       19-Aug-09 16:58 RestoreFiles
66                2     38.77 K  OK       19-Aug-09 16:59 RestoreFiles
67  Full         25     145.5 M  OK       21-Aug-09 13:41 Client1
```

这里可以看到，JobId 为 67 的备份是一个完全备份，备份数据的文件数为 25 个，备份压缩后的大小为 145.5MB，备份完成时间为“21-Aug-09 13:41”。最后的 Client1 是备份资源的名称。

4.5.2　实例演示 bacula 的增量备份功能

在上面的操作中，设定的备份资源 Job 本身就是一个增量备份。下面执行的备份操作与上面完全相同，不同的是，这里是第二次备份，bacula 会按照设定执行增量备份，操作如下。

```
* run
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
A job name must be specified.
The defined Job resources are:
     1: Client1
     2: Client
     3: BackupCatalog
     4: RestoreFiles
Select Job resource (1-2): 1
Run Backup job
JobName:  Client1
Level:    Incremental
Client:   dbfd
FileSet:  dbfs
Pool:     dbpool (From Job resource)
Storage:  dbsd (From Job resource)
When:     2009-08-21 14:20:24
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=68
*status
Status available for:
       1: Director
       2: Storage
       3: Client
       4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 13:22, 2 Jobs run since started.
 Heap: heap=241,664 smbytes=84,576 max_bytes=97,749 bufs=218 max_bufs=252

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
```

```
================================================================================
Incremental     Backup     10  21-Aug-09 23:05     Client1            cicro4

Running Jobs:
Console connected at 21-Aug-09 14:19
No Jobs running.

Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    59  Incr           2    3.908 M  OK       19-Aug-09 16:51 Client1
    60  Incr           2    8.377 K  OK       19-Aug-09 16:52 Client1
    61  Incr           1        0    OK       19-Aug-09 16:53 Client1
    62  Incr           2    8.344 K  OK       19-Aug-09 16:53 Client1
    63  Incr           2    8.377 K  OK       19-Aug-09 16:54 Client1
    64                 3    77.44 K  OK       19-Aug-09 16:57 RestoreFiles
    65                 1        0    OK       19-Aug-09 16:58 RestoreFiles
    66                 2    38.77 K  OK       19-Aug-09 16:59 RestoreFiles
    67  Full          25    145.5 M  OK       21-Aug-09 13:41 Client1
68  Incr               2    3.908 M  OK       21-Aug-09 14:20 Client1
```

从最后的备份列表可以看出，系统进行了增量备份，“3.908 M”就是上次备份与这次备份之间的数据增加量。而“JobId=68”这个备份的 Level 级别为 Incr 也说明了此次备份为增量备份。

4.5.3 实例演示 bacula 的差异备份功能

差异备份与增量备份的原理前面已经介绍过了，这里只给出具体的操作步骤。

1. 开始执行备份

```
*run
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
A job name must be specified.
The defined Job resources are:
       1: Client1
       2: Client
       3: BackupCatalog
       4: RestoreFiles
Select Job resource (1-4): 2      # 这里有两个 Job 任务可选，名为 Client1 的为增量
                                  # 备份操作，名为 Client 的为差异备份操作

Run Backup job
JobName:  Client
Level:    Differential
Client:   dbfd
FileSet:  dbfs
Pool:     dbpool (From Job resource)
Storage:  dbsd (From Job resource)
```

```
When:     2009-08-21 14:31:04
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=69
You have messages.
```

至此，开始执行差异备份，如果备份的数据量较大，可能需要一段时间才能完成备份。下面可以用“status”指令查看备份的状态。

2. 查看备份状态

```
*status
Status available for:
       1: Director
       2: Storage
       3: Client
       4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 14:30, 0 Jobs run since started.
 Heap: heap=253,952 smbytes=67,810 max_bytes=68,066 bufs=267 max_bufs=272

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4
Incremental    Backup    10  21-Aug-09 23:05    Client             cicro4
Incremental    Backup    10  21-Aug-09 23:05    BackupCatalog      cicro4

Running Jobs:
Console connected at 21-Aug-09 14:30
 JobId Level   Name                       Status
======================================================================
    69 Full    Client.2009-08-21_14.31.10_03 is running
Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    59  Incr         2    3.908 M  OK       19-Aug-09 16:51 Client1
    60  Incr         2    8.377 K  OK       19-Aug-09 16:52 Client1
    61  Incr         1          0  OK       19-Aug-09 16:53 Client1
    62  Incr         2    8.344 K  OK       19-Aug-09 16:53 Client1
    63  Incr         2    8.377 K  OK       19-Aug-09 16:54 Client1
    64               3    77.44 K  OK       19-Aug-09 16:57 RestoreFiles
    65               1          0  OK       19-Aug-09 16:58 RestoreFiles
    66               2    38.77 K  OK       19-Aug-09 16:59 RestoreFiles
    67  Full        25    145.5 M  OK       21-Aug-09 13:41 Client1
    68  Incr         2    3.908 M  OK       21-Aug-09 14:20 Client1
```

从上面可以看到，备份还没有完成，备份状态为“running”。

等待几分钟，再次查看备份状态如下：

```
* status
Status available for:
       1: Director
       2: Storage
       3: Client
       4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 14:30, 1 Job run since started.
 Heap: heap=253,952 smbytes=65,680 max_bytes=78,853 bufs=240 max_bufs=274

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4
Incremental    Backup    10  21-Aug-09 23:05    Client             cicro4
Incremental    Backup    10  21-Aug-09 23:05    BackupCatalog      cicro4

Running Jobs:
Console connected at 21-Aug-09 14:30
No Jobs running.

Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    60  Incr         2    8.377 K  OK       19-Aug-09 16:52 Client1
    61  Incr         1        0    OK       19-Aug-09 16:53 Client1
    62  Incr         2    8.344 K  OK       19-Aug-09 16:53 Client1
    63  Incr         2    8.377 K  OK       19-Aug-09 16:54 Client1
    64               3    77.44 K  OK       19-Aug-09 16:57 RestoreFiles
    65               1        0    OK       19-Aug-09 16:58 RestoreFiles
    66               2    38.77 K  OK       19-Aug-09 16:59 RestoreFiles
    67  Full        25    145.5 M  OK       21-Aug-09 13:41 Client1
    68  Incr         2    3.908 M  OK       21-Aug-09 14:20 Client1
    69  Full        26    149.4 M  OK       21-Aug-09 14:32 Client
```

从这里可以看到，JobId 为“69”的备份已经完成，并且是一个完全备份，备份文件大小为 149.4MB，而完成此完全备份的 Job 任务名称是 Client。

3. 继续执行备份操作

在执行下面的操作前，首先在客户端 FD 需要备份的文件夹下（根据前面的配置可知，应该是 /cws3）增加一个大小为 3.8MB 左右的文件。

```
* run
A job name must be specified.
The defined Job resources are:
      1: Client1
      2: Client
      3: BackupCatalog
```

```
      4: RestoreFiles
Select Job resource (1-4): 2
Run Backup job
JobName:  Client
Level:    Differential
Client:   dbfd
FileSet:  dbfs
Pool:     dbpool (From Job resource)
Storage:  dbsd (From Job resource)
When:     2009-08-21 14:34:25
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=70
*status
Status available for:
       1: Director
       2: Storage
       3: Client
       4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 14:30, 2 Jobs run since started.
 Heap: heap=253,952 smbytes=67,989 max_bytes=81,162 bufs=240 max_bufs=274

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4
Incremental    Backup    10  21-Aug-09 23:05    Client             cicro4
Incremental    Backup    10  21-Aug-09 23:05    BackupCatalog      cicro4

Running Jobs:
Console connected at 21-Aug-09 14:30
No Jobs running.

Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    61  Incr         1          0   OK       19-Aug-09 16:53 Client1
    62  Incr         2      8.344 K  OK       19-Aug-09 16:53 Client1
    63  Incr         2      8.377 K  OK       19-Aug-09 16:54 Client1
    64               3      77.44 K  OK       19-Aug-09 16:57 RestoreFiles
    65               1          0   OK       19-Aug-09 16:58 RestoreFiles
    66               2      38.77 K  OK       19-Aug-09 16:59 RestoreFiles
    67  Full        25      145.5 M  OK       21-Aug-09 13:41 Client1
    68  Incr         2      3.908 M  OK       21-Aug-09 14:20 Client1
    69  Full        26      149.4 M  OK       21-Aug-09 14:32 Client
    70  Diff         2      3.908 M  OK       21-Aug-09 14:34 Client
```

从 JobId 为“70”的备份信息可以看出，此备份为差异备份，此次进行差异备份的数据

量大小为 3.908 MB，与前面增加的文件大小基本相同。

为了证明前面执行的是差异备份，在 FD 端需要备份的文件夹下再次增加一个大小为 3.8MB 左右的一个文件，并继续执行以下备份操作：

```
*run
A job name must be specified.
The defined Job resources are:
      1: Client1
      2: Client
      3: BackupCatalog
      4: RestoreFiles
Select Job resource (1-4): 2
Run Backup job
JobName:  Client
Level:    Differential
Client:   dbfd
FileSet:  dbfs
Pool:     dbpool (From Job resource)
Storage:  dbsd (From Job resource)
When:     2009-08-21 14:34:59
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=71
*status
Status available for:
       1: Director
       2: Storage
       3: Client
       4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 14:30, 3 Jobs run since started.
 Heap: heap=253,952 smbytes=69,087 max_bytes=82,260 bufs=240 max_bufs=274

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4
Incremental    Backup    10  21-Aug-09 23:05    Client             cicro4
Incremental    Backup    10  21-Aug-09 23:05    BackupCatalog      cicro4

Running Jobs:
Console connected at 21-Aug-09 14:30
No Jobs running.

Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    62  Incr         2    8.344 K  OK       19-Aug-09 16:53 Client1
```

```
 63  Incr          2    8.377 K  OK       19-Aug-09 16:54 Client1
 64                3    77.44 K  OK       19-Aug-09 16:57 RestoreFiles
 65                1        0    OK       19-Aug-09 16:58 RestoreFiles
 66                2    38.77 K  OK       19-Aug-09 16:59 RestoreFiles
 67  Full         25    145.5 M  OK       21-Aug-09 13:41 Client1
 68  Incr          2    3.908 M  OK       21-Aug-09 14:20 Client1
 69  Full         26    149.4 M  OK       21-Aug-09 14:32 Client
 70  Diff          2    3.908 M  OK       21-Aug-09 14:34 Client
 71  Diff          3    7.817 M  OK       21-Aug-09 14:35 Client
```

由 JobId 为 71 的备份可以看出，此次备份的大小为 7.817 M，刚好是上次差异增量备份的大小与此次增加文件的大小之和，由此证明确实为差异备份。

第三次在 FD 端需要备份的文件夹下增加一个大小为 3.8M 左右的一个文件，继续执行备份操作。

```
*run
A job name must be specified.
The defined Job resources are:
      1: Client1
      2: Client
      3: BackupCatalog
      4: RestoreFiles
Select Job resource (1-4): 2
Run Backup job
JobName:  Client
Level:    Differential
Client:   dbfd
FileSet:  dbfs
Pool:     dbpool (From Job resource)
Storage:  dbsd (From Job resource)
When:     2009-08-21 14:35:32
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=72
*status
Status available for:
       1: Director
       2: Storage
       3: Client
       4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 14:30, 4 Jobs run since started.
 Heap: heap=253,952 smbytes=69,087 max_bytes=82,260 bufs=240 max_bufs=274

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4
```

```
Incremental     Backup     10  21-Aug-09 23:05     Client              cicro4
Incremental     Backup     10  21-Aug-09 23:05     BackupCatalog       cicro4

Running Jobs:
Console connected at 21-Aug-09 14:30
No Jobs running.

Terminated Jobs:
 JobId  Level    Files       Bytes   Status   Finished        Name
====================================================================
    63  Incr         2    8.377 K  OK       19-Aug-09 16:54 Client1
    64               3    77.44 K  OK       19-Aug-09 16:57 RestoreFiles
    65               1         0   OK       19-Aug-09 16:58 RestoreFiles
    66               2    38.77 K  OK       19-Aug-09 16:59 RestoreFiles
    67  Full        25    145.5 M  OK       21-Aug-09 13:41 Client1
    68  Incr         2    3.908 M  OK       21-Aug-09 14:20 Client1
    69  Full        26    149.4 M  OK       21-Aug-09 14:32 Client
    70  Diff         2    3.908 M  OK       21-Aug-09 14:34 Client
    71  Diff         3    7.817 M  OK       21-Aug-09 14:35 Client
    72  Diff         4    11.72 M  OK       21-Aug-09 14:35 Client
```

从 JobId 为“72”的备份可以很清楚地看出差异备份与增量备份的差别。

4.5.4 实例演示 bacula 的完全恢复功能

1. 通过差异备份进行完全恢复

操作步骤如下：

```
*status
Status available for:
       1: Director
       2: Storage
       3: Client
       4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 15:01, 1 Job run since started.
 Heap: heap=253,952 smbytes=84,808 max_bytes=85,641 bufs=266 max_bufs=279

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================
Incremental     Backup     10  21-Aug-09 23:05     Client1            cicro4
Incremental     Backup     10  21-Aug-09 23:05     Client             cicro4
Incremental     Backup     10  21-Aug-09 23:05     BackupCatalog      cicro4

Running Jobs:
Console connected at 21-Aug-09 16:32
```

```
Console connected at 21-Aug-09 16:33
No Jobs running.

Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    64               3    77.44 K  OK       19-Aug-09 16:57 RestoreFiles
    65               1          0  OK       19-Aug-09 16:58 RestoreFiles
    66               2    38.77 K  OK       19-Aug-09 16:59 RestoreFiles
    67  Full        25    145.5 M  OK       21-Aug-09 13:41 Client1
    68  Incr         2    3.908 M  OK       21-Aug-09 14:20 Client1
    69  Full        26    149.4 M  OK       21-Aug-09 14:32 Client
    70  Diff         2    3.908 M  OK       21-Aug-09 14:34 Client
    71  Diff         3    7.817 M  OK       21-Aug-09 14:35 Client
    72  Diff         4    11.72 M  OK       21-Aug-09 14:35 Client
    73  Incr         4    11.72 M  OK       21-Aug-09 15:34 Client1
```

从这里的备份状态可知，JobId 为“69”的任务是一个完全备份，而 JobId 为“70”、“71”、“72”的 3 个任务是差异备份。下面进行恢复操作：

```
* restore
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
     10: Find the JobIds for a backup for a client before a specified time
     11: Enter a list of directories to restore for found JobIds
     12: Select full restore to a specified JobId
     13: Cancel
Select item:  (1-13): 3
Enter JobId(s), comma separated, to restore: 69,72
You have selected the following JobIds: 69,72
#这里仅仅指定了差异备份的第一个完全备份和差异备份的最后一个备份的 JobId，即可完全恢复的数据
Building directory tree for JobId(s) 69,72 ...
26 files inserted into the tree.
You are now entering file selection mode where you add (mark) and
```

```
    remove (unmark) files to be restored. No files are initially added, unless
    you used the "all" keyword on the command line.
    Enter "done" to leave this mode.

    cwd is: /
    $ mark cicro
    29 files marked.
    $ done
Bootstrap records written to /opt/bacula/var/bacula/working/f10-64-build-dir.
restore.1.bsr
    The job will require the following
       Volume(s)                 Storage(s)                SD Device(s)
    ===========================================================================
        cicro4                    dbsd                      dbdev
    Volumes marked with "*" are online.
    29 files selected to be restored.
    Defined Clients:  #指定恢复到哪个FD上去
          1: dbfd
          2: dbfd1
    Select the Client (1-2): 1
    Run Restore job
    JobName:          RestoreFiles
    Bootstrap:        /opt/bacula/var/bacula/working/f10-64-build-dir.restore.1.bsr
    Where:            /tmp/bacula-restores
    Replace:          always
    FileSet:          dbfs
    Backup Client:    dbfd
    Restore Client:   dbfd
    Storage:          dbsd
    When:             2009-08-21 16:35:07
    Catalog:          MyCatalog
    Priority:         10
    Plugin Options:   *None*
    OK to run? (yes/mod/no): mod
    #选择要将数据恢复到的路径，bacula默认恢复到前面指定的/tmp/bacula-restores目录下，
    #在这里，直接将数据恢复到/cicro目录下，因此选择“mod”
    Parameters to modify:
        1: Level
        2: Storage
        3: Job
        4: FileSet
        5: Restore Client
        6: When
        7: Priority
        8: Bootstrap
        9: Where
       10: File Relocation
       11: Replace
       12: JobId
       13: Plugin Options
```

```
Select parameter to modify (1-13): 9
Please enter path prefix for restore (/ for none): /
#这里指定 / 即可，因为前面已经设置了一个 /cicro
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/var/bacula/working/f10-64-build-dir.restore.1.bsr
Where:
Replace:         always
FileSet:         dbfs
Backup Client:   dbfd
Restore Client:  dbfd
Storage:         dbsd
When:            2009-08-21 16:35:07
Catalog:         MyCatalog
Priority:        10
Plugin Options:  *None*
OK to run? (yes/mod/no): yes
Job queued. JobId=74
```

这样，一个完全的恢复就通过差异备份方式完成了。可以在远程备份机器（即 FD 端）上看到，数据已经恢复了。

2. 通过增量备份进行完全恢复

执行下面操作前需要先删除 FD 端的备份目录。

```
*status
Status available for:
      1: Director
      2: Storage
      3: Client
      4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 15:01, 2 Jobs run since started.
 Heap: heap=253,952 smbytes=100,693 max_bytes=223,457 bufs=280 max_bufs=312

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4
Incremental    Backup    10  21-Aug-09 23:05    Client             cicro4
Incremental    Backup    10  21-Aug-09 23:05    BackupCatalog      cicro4

Running Jobs:
Console connected at 21-Aug-09 16:32
Console connected at 21-Aug-09 16:50
No Jobs running.

Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
```

```
============================================================================
    65                  1          0   OK       19-Aug-09 16:58 RestoreFiles
    66                  2     38.77 K  OK       19-Aug-09 16:59 RestoreFiles
    67  Full           25     145.5 M  OK       21-Aug-09 13:41 Client1
    68  Incr            2     3.908 M  OK       21-Aug-09 14:20 Client1
    69  Full           26     149.4 M  OK       21-Aug-09 14:32 Client
    70  Diff            2     3.908 M  OK       21-Aug-09 14:34 Client
    71  Diff            3     7.817 M  OK       21-Aug-09 14:35 Client
    72  Diff            4     11.72 M  OK       21-Aug-09 14:35 Client
    73  Incr            4     11.72 M  OK       21-Aug-09 15:34 Client1
    74                 29     4.092 G  OK       21-Aug-09 16:35 RestoreFiles
*restore
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
    10: Find the JobIds for a backup for a client before a specified time
    11: Enter a list of directories to restore for found JobIds
    12: Select full restore to a specified JobId
    13: Cancel
Select item:  (1-13): 3
Enter JobId(s), comma separated, to restore: 67,68,73
You have selected the following JobIds: 67,68,73
# 在这里，JobId 为“67”的备份是一个完全备份，而 JobId 为“68”的备份为第一个增量备份，
# JobId 为“73”的备份为第二个增量备份，如果还有其他的增量备份，都要在这里指定，才能完全恢复
Building directory tree for JobId(s) 67,68,73 ...
26 files inserted into the tree.

You are now entering file selection mode where you add (mark) and
remove (unmark) files to be restored. No files are initially added, unless
you used the "all" keyword on the command line.
Enter "done" to leave this mode.
cwd is: /
$ mark cicro    # 标记恢复
29 files marked.
$ done          # 确认操作
```

```
    Bootstrap records written to /opt/bacula/var/bacula/working/f10-64-build-dir.
restore.2.bsr
    The job will require the following
       Volume(s)                 Storage(s)                SD Device(s)
    ===========================================================================
        cicro4                    dbsd                      dbdev

    Volumes marked with "*" are online.
    29 files selected to be restored.

    Defined Clients:
          1: dbfd
          2: dbfd1
    Select the Client (1-2): 1
    Run Restore job
    JobName:          RestoreFiles
    Bootstrap:        /opt/bacula/var/bacula/working/f10-64-build-dir.restore.2.bsr
    Where:            /tmp/bacula-restores
    Replace:          always
    FileSet:          dbfs
    Backup Client:    dbfd
    Restore Client:   dbfd
    Storage:          dbsd
    When:             2009-08-21 16:52:04
    Catalog:          MyCatalog
    Priority:         10
    Plugin Options:   *None*
    OK to run? (yes/mod/no): mod
    Parameters to modify:
        1: Level
        2: Storage
        3: Job
        4: FileSet
        5: Restore Client
        6: When
        7: Priority
        8: Bootstrap
        9: Where
        10: File Relocation
        11: Replace
        12: JobId
        13: Plugin Options
    Select parameter to modify (1-13): 9
    Please enter path prefix for restore (/ for none): /
    Run Restore job
    JobName:          RestoreFiles
    Bootstrap:        /opt/bacula/var/bacula/working/f10-64-build-dir.restore.2.bsr
    Where:
    Replace:          always
    FileSet:          dbfs
```

```
    Backup Client:   dbfd
    Restore Client:  dbfd
    Storage:         dbsd
    When:            2009-08-21 16:52:04
    Catalog:         MyCatalog
    Priority:        10
    Plugin Options:  *None*
    OK to run? (yes/mod/no): yes
Job queued. JobId=75
```

至此，增量备份的完全恢复也完成了。由此可以看出增量备份的恢复与差异备份的恢复之间的差异。

4.5.5 实例演示 bacula 的不完全恢复功能

不完全恢复的操作过程与完全恢复基本一致，不同的是，在指定恢复 JobId 时，不完全恢复只需恢复到指定的某个 JobId 即可用增量备份和差异备份来实现不完全恢复。这里以差异备份为例，操作过程如下：

```
*status
Status available for:
      1: Director
      2: Storage
      3: Client
      4: All
Select daemon type for status (1-4): 1
f10-64-build-dir Version: 3.0.2 (18 July 2009) x86_64-unknown-linux-gnu redhat
Daemon started 21-Aug-09 15:01, 3 Jobs run since started.
 Heap: heap=442,368 smbytes=105,435 max_bytes=237,737 bufs=280 max_bufs=312

Scheduled Jobs:
Level          Type     Pri  Scheduled          Name               Volume
===================================================================================
Incremental    Backup    10  21-Aug-09 23:05    Client1            cicro4
Incremental    Backup    10  21-Aug-09 23:05    Client             cicro4
Incremental    Backup    10  21-Aug-09 23:05    BackupCatalog      cicro4

Running Jobs:
Console connected at 21-Aug-09 16:32
Console connected at 21-Aug-09 17:19
No Jobs running.

Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    66                2    38.77 K  OK       19-Aug-09 16:59 RestoreFiles
    67  Full         25    145.5 M  OK       21-Aug-09 13:41 Client1
    68  Incr          2    3.908 M  OK       21-Aug-09 14:20 Client1
```

```
69   Full        26    149.4 M   OK       21-Aug-09 14:32 Client
70   Diff         2    3.908 M   OK       21-Aug-09 14:34 Client
71   Diff         3    7.817 M   OK       21-Aug-09 14:35 Client
72   Diff         4    11.72 M   OK       21-Aug-09 14:35 Client
73   Incr         4    11.72 M   OK       21-Aug-09 15:34 Client1
74               29    4.092 G   OK       21-Aug-09 16:35 RestoreFiles
75               29    4.092 G   OK       21-Aug-09 16:52 RestoreFiles
```

从这里可以看出，JobId 为“69”到 JobId 为“72”的备份都属于差异备份。在进行不完全恢复时，若希望将数据恢复到 JobId 为“71”的状态下，可以执行如下操作：

```
*restore
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
     10: Find the JobIds for a backup for a client before a specified time
     11: Enter a list of directories to restore for found JobIds
     12: Select full restore to a specified JobId
     13: Cancel
Select item:  (1-13): 3
Enter JobId(s), comma separated, to restore: 69,71   #这里指定数据恢复到什么程度
You have selected the following JobIds: 69,71

Building directory tree for JobId(s) 69,71 ...
25 files inserted into the tree.

You are now entering file selection mode where you add (mark) and
remove (unmark) files to be restored. No files are initially added, unless
you used the "all" keyword on the command line.
Enter "done" to leave this mode.

cwd is: /
$ mark cicro
28 files marked.
$ done
```

```
    Bootstrap records written to /opt/bacula/var/bacula/working/f10-64-build-dir.
restore.3.bsr
    The job will require the following
       Volume(s)                 Storage(s)                SD Device(s)
    ===========================================================================
        cicro4                    dbsd                      dbdev
    Volumes marked with "*" are online.
    28 files selected to be restored.

    Defined Clients:
         1: dbfd
         2: dbfd1
    Select the Client (1-2): 1
    Run Restore job
    JobName:          RestoreFiles
    Bootstrap:        /opt/bacula/var/bacula/working/f10-64-build-dir.restore.3.bsr
    Where:            /tmp/bacula-restores
    Replace:          always
    FileSet:          dbfs
    Backup Client:    dbfd
    Restore Client:   dbfd
    Storage:          dbsd
    When:             2009-08-21 17:21:16
    Catalog:          MyCatalog
    Priority:         10
    Plugin Options:   *None*
    OK to run? (yes/mod/no): mod
    Parameters to modify:
         1: Level
         2: Storage
         3: Job
         4: FileSet
         5: Restore Client
         6: When
         7: Priority
         8: Bootstrap
         9: Where
         10: File Relocation
         11: Replace
         12: JobId
         13: Plugin Options
    Select parameter to modify (1-13): 9
    Please enter path prefix for restore (/ for none): /
    Run Restore job
    JobName:          RestoreFiles
    Bootstrap:        /opt/bacula/var/bacula/working/f10-64-build-dir.restore.3.bsr
    Where:
    Replace:          always
    FileSet:          dbfs
    Backup Client:    dbfd
```

```
Restore Client:   dbfd
Storage:          dbsd
When:             2009-08-21 17:21:16
Catalog:          MyCatalog
Priority:         10
Plugin Options:   *None*
OK to run? (yes/mod/no): yes
Job queued. JobId=76
```

最后，查看恢复的路径可以发现，已经将数据恢复到了指定的时间段内，不完全恢复工作完成。

至此，bacula 的安装、配置和使用已经介绍完毕。

4.6　本章小结

本章主要介绍了开源备份软件 bacula 的安装、使用与管理技巧。开篇先介绍了 bacula 的功能特点和应用范围，接着介绍 bacula 的工作原理及运行机制，然后开始详细介绍 bacula 的安装和具体配置，同时还讲解了 bacula 的基本维护技巧，最后通过实例的方式介绍了在 bacula 上进行完全备份、增量备份、差异备份、完全恢复、不完全恢复的具体操作步骤。学习完本章内容，相信读者能够熟练使用 bacula 了。

根据 SourceForge 的统计，bacula 已经成为最流行的企业开源应用软件，作为一个备份工具，它可以在网络上的各种不同系统之间实现文件的备份、恢复和验证等功能。它是基于传统的 C/S 模式的网络备份程序，被誉为开源平台下最优秀的网络备份工具之一。功能强大的 bacula 完全可以和商用备份软件相媲美。相信在不久的将来 bacula 一定会在企业中得到广泛应用。

第 5 章　数据镜像备份工具 rsync 与 unison

随着 Linux 系统的迅速发展和普及，很多中小企业用户都选择 Linux 作为应用平台。Linux 作为服务器，其稳定、高效的特性得到了很多用户的肯定，同时也带来了一些问题，如需要实现数据的本地和异地备份，以保证数据安全。虽然有很多的商业备份软件可供选择，但是这些产品的价格往往过于昂贵，很多用户无法承受。因此，如何利用开源软件高效地实现数据的镜像备份和异地备份就成为一个热门话题。本章重点介绍两个高效的数据镜像备份工具 rsync 和 unison，通过它们基本可以满足一般的备份需求。

5.1　rsync 简介

传统的数据备份方式有 cp 命令或者 wget 命令。cp 命令的源文件和目标文件都在本地，该命令仅实现对文件的一种完整复制，如果要复制的数据量巨大，那么备份的时间就会变得很长；wget 命令通过网络进行备份，它不支持增量备份，每次都需要将所有数据重新在网络上传输一遍，而不考虑哪些文件是更新过的，因此该命令的效率也非常低。这里介绍一个小巧而实用的工具 rsync，借助于这个工具能轻松实现数据的本地镜像和远程备份。

5.1.1　什么是 rsync

rsync 是 Linux/UNIX 系统下的文件同步和数据传输工具，它采用“rsync 算法”使一个客户机和远程文件服务器之间的文件同步。通过 rsync 可以将同一个服务器的数据从一个分区备份到另一个分区，也可以将本地系统的数据通过网络传输方式备份到任何一个远程主机上；rsync 可以在中断之后恢复传输；rsync 只传输源文件和目标文件之间不一致的部分；rsync 可以执行完整备份或增量备份。

5.1.2　rsync 的功能特性

rsync 即 remote sync，从软件名称上就可以看出它所实现的功能。rsync 有如下特性：

- 可以镜像保存整个目录树和文件系统。
- 可以增量同步数据，文件传输效率高，因而同步时间很短。
- 可以保持原有文件的权限、时间等属性。
- 加密传输数据，保证了数据的安全性。

- 可以使用 rcp、ssh 等方式来传输文件，当然也可以直接通过 Socket 连接传输文件。
- 支持匿名传输。

5.1.3　下载与安装 rsync 软件

rysnc 的主页地址为：http://rsync.samba.org/，这里下载的版本为 rsync-3.0.4。下面进行编译安装，过程如下：

```
[root@web ~]#tar zxvf rsync-3.0.4.tar.gz
[root@web ~]#cd rsync-3.0.4
[root@web rsync-3.0.4]# ./configure
[root@web rsync-3.0.4]# make
[root@web rsync-3.0.4]# make install
```

这样就完成了 rsync 的安装。

5.2　利用 rsync 搭建数据镜像备份系统

5.2.1　rsync 的应用模式

rsync 有 4 种应用模式，第一种是 shell 应用模式，也称为本地模式；第二种是远程 shell 模式，它利用 SSH 执行底层连接和加密传输；第三种是查询（也叫列表）模式，与 ls 命令实现的功能类似；最后一种是服务器模式，平时所说的架设 rsync 服务器就是指这种模式。下面分别对这 4 种模式进行详细介绍。

1. 本地 shell 模式

本地 shell 模式主要用于复制指定目录到另一个目录，例如：

```
[root@localhost server]# rsync  -av license  /tmp
building file list ... done
license/
license/license.data
license/ixdba/
license/ixdba/license.data

sent 793 bytes  received 76 bytes  1738.00 bytes/sec
total size is 566  speedup is 0.65

[root@localhost server]# rsync  -av license/  /tmp
building file list ... done
./
license.data
ixdba/
ixdba/license.data
```

```
sent 783 bytes  received 76 bytes  1718.00 bytes/sec
total size is 566  speedup is 0.66
```

看到这两个命令的差异了吗？明显的差异是源参数末尾的斜杠。如果源参数的末尾没有斜杠，就将指定的源目录复制到指定的目的目录；如果源参数末尾有斜杠，就会复制指定源目录中的内容到目的目录中，而不复制目录本身。目标参数末尾的斜杠没有任何作用。

在以上代码中，“-a”即为“--archive”（归档模式），表示以递归方式传输文件，并保持所有文件属性；“-v”即为“--verbose”，表示输出详细模式信息。

2. 远程 shell 模式

通过远程 shell 模式，rsync 可以把指定的本地目录复制到另一个系统中。例如：

```
[root@localhost server]# rsync  -av license 192.168.12.251:test
root@192.168.12.251's password:
building file list ... done
license/
license/license.data
license/ixdba/
license/ixdba/license.data

sent 793 bytes  received 76 bytes  34.08 bytes/sec
total size is 566  speedup is 0.65
```

如果以 root 用户身份执行此命令，rsync 会提示输入远程主机 192.168.12.251 的 root 密码。完成密码验证后，会在远程主机的 root 用户根目录下创建 test 目录，然后将 license 目录及该目录下的内容复制过来。在默认情况下，rsync 使用 Secure Shell（SSH）作为传输机制。

3. rsync 列表模式

在这个模式下，rsync 与 ls 命令有相似的功能。例如：以 root 用户身份查看远程主机 192.168.12.251 的 test 目录下的内容，可以使用如下组合：

```
[root@localhost server]# rsync  -a  192.168.12.251:test
root@192.168.12.251's password:
drwxr-xr-x       4096 2010/04/13 10:33:15 test
drwxr-xr-x       4096 2010/04/13 10:37:07 test/license
-rw-r--r--        283 2010/03/24 11:11:21 test/license/license.data
drwxr-xr-x       4096 2010/04/13 10:37:04 test/license/ixdba
-rw-r--r--        283 2010/04/13 10:37:04 test/license/ixdba/license.data
```

查看本地系统中 /tmp/license 目录的内容，可以使用如下组合：

```
[root@localhost server]# rsync  -a /tmp/license
drwxr-xr-x       4096 2010/04/13 10:37:07 license
-rw-r--r--        283 2010/03/24 11:11:21 license/license.data
drwxr-xr-x       4096 2010/04/13 10:37:04 license/ixdba
-rw-r--r--        283 2010/04/13 10:37:04 license/ixdba/license.data
```

4. 服务器模式

这种模式是基于 C/S 模式的，在这种模式下，rsync 在后台启用了一个守护进程，这个守护进程在 rsync 服务器端永久运行，用于接收文件传输请求，因此，客户端既可以把文件发送给守护进程，也可以向守护进程请求文件。rsync 的服务器模式非常适合作为异地的中心备份服务器或数据异地存储库来使用。关于服务器模式的使用方法，在下节进行深入介绍。

5.2.2　企业案例：搭建远程容灾备份系统

为了更清楚地介绍 rsync 服务器模式的使用方法，这里通过一个企业案例，介绍如何搭建一个远程容灾备份系统。

案例描述

某电子商务企业有一个门户网站，Web 服务器的操作系统是 Linux，网站数据每天都会增加。为保证数据安全，需要建立一个远程容灾系统，将网站数据在每天凌晨 3 点 30 分备份到远程的容灾服务器上。由于数据量很大，每天只能进行增量备份，即仅仅备份当天增加的数据，当网站出现故障后，可以通过备份最大程度地恢复数据。

解决方案

这里假定有 A、B 两个 Linux 系统，A 系统作为网站服务器，B 系统作为 A 的远程容灾备份机，因此 A 系统就是 rsync 的服务器端，B 系统就是 rsync 的客户端。为了完成数据的容灾备份，需要在 A、B 两个系统上都安装 rsync 软件，这样，在 A 系统上运行 rsync 守护进程，而在 B 系统上可以通过系统守护进程 crontab 来定时备份由 A 系统指定的数据，从而实现数据的远程容灾。

系统环境

操作系统：Red Hat Enterprise Linux Server release 5

内核版本：Linux web 2.6.18-8.el5

A 系统 IP 地址：192.168.12.253

B 系统 IP 地址：192.168.12.231

由于 rsync 的安装非常简单，下面直接进入 rsync 配置。

1. 在 A 系统上配置 rsync

rsync 的配置文件为 /etc/rsyncd.conf，在安装完 rsync 时，默认没有这个文件，手动建立一个即可。rsyncd.conf 文件由一个或多个模块结构组成，相应地，包括全局参数和模块参数，一个模块定义从方括弧中的模块名开始，直到下个模块的定义开始。配置完毕的内容如下：

```
uid = nobody
gid = nobody
use chroot = no
max connections = 10
strict modes = yes
```

```
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsync.lock
log file = /var/log/rsyncd.log

[ixdba]
path = /webdata
comment = ixdba file
ignore errors
read only = no
write only = no
hosts allow = *
hosts deny = 192.168.12.131
list = false
uid = root
gid = root
auth users = backup
secrets file = /etc/server.pass
```

其中，/etc/server.pass 中的内容如下：

```
[root@localhost webdata]# more /etc/server.pass
backup:ixdba123
[root@localhost webdata]# chmod 600 /etc/server.pass
```

上面每个选项的含义如下：

- uid，此选项指定当该模块传输文件时守护进程应该具有的用户 ID，默认值是“nobody”。
- gid，此选项指定当该模块传输文件时守护进程应该具有的用户组 ID，默认值为“nobody”。
- max connections，此选项指定模块的最大并发连接数量，以保护服务器。超过限制的连接请求将被暂时限制。默认值是 0，即没有限制。
- strict modes，此选项指定是否检查口令文件的权限，yes 为检查口令文件权限，反之为 no。如果设置为 yes，密码文件的权限必须为 root 用户权限。
- pid file，此选项用来指定 rsync 守护进程对应的 PID 文件路径。
- lock file，此选项用来指定支持 max connections 的锁文件，默认值是 /var/run/rsyncd.lock。
- log file，此选项指定了 rsync 的日志输出文件路径。
- [ixdba]，表示定义一个模块的开始，ixdba 就是对应的模块名称。
- path，此选项用来指定需要备份的文件或目录，是必须设置的项。这里指定的目录为 /webdata。
- ignore errors，表示可以忽略一些无关的 I/O 错误。
- read only，设置为 no 表示客户端可以上传文件，设置为 yes 表示只读。
- write only，设置为 no 表示客户端可下载文件，设置为 yes 表示不能下载。

- hosts allow，设置可以连接 rsync 服务器的主机，“*”表示允许连接任何主机。
- hosts deny，设置禁止连接 rsync 服务器的主机地址。
- list，此选项用于设定当客户请求可以使用的模块列表时，该模块是否被列出。默认值是 true，如果需要建立隐藏的模块，可以设置为 false。
- vauth users，此选项用来定义可以连接该模块的用户名，多个用户以空格或逗号分隔开。需要注意的是，这里的用户和 Linux 系统用户没有任何关系。这里指定的用户是 backup。
- secrets file，此选项用于指定一个包含“用户名:密码”格式的文件，用户名就是“auth users”选项定义的用户，密码可以随便指定，这里设定为 ixdba123，只要和客户端的 secrets file 对应起来即可。只有在 auth users 被定义时，该文件才起作用。系统默认没有这个文件，自己手动创建一个即可。

2. 在 A 系统上启动 rsync 守护进程

执行如下指令启动 rsync 守护进程：

```
[root@web ~]# /usr/local/bin/rsync --daemon
[root@web ]# ps -ef|grep rsync
root     20278     1  0 16:29 ?        00:00:00 /usr/local/bin/rsync --daemon
```

3. 在 B 系统上配置 rsync

在备份机上不用做任何设置，只需执行 rsync 同步操作即可。为了在同步过程中不用输入密码，需要在 B 系统上创建一个 secrets file，此文件的内容为 A 系统 rsyncd.conf 文件中“auth users”选项指定用户的密码，而这个文件的名称及路径可以随意指定，只要在执行 rsync 同步时指定即可。

接下来执行同步操作，具体指令如下：

```
[root@localhost]# /usr/local/bin/rsync -vzrtopg  --delete  --progress --exclude
    "*access*" --exclude "debug*" backup@192.168.12.253::ixdba /ixdba.net
    --password-file=/etc/server.pass
```

其中，B 服务器（客户端系统）中 /etc/server.pass 的内容如下：

```
[root@localhost ~]# more /etc/server.pass
ixdba123
[root@localhost ~]# chmod 600 /etc/server.pass
```

这条指令中每个参数的含义如下：

- 在“-vzrtopg”选项中，v 是“--verbose”，即详细模式输出；z 表示“--compress”，即在传输时对备份的文件进行压缩处理；r 表示“--recursive”，也就是对子目录以递归模式处理；t 即“--times”，用来保持文件时间信息；o 即“--owner”，用来保持文件属主信息；p 即“--perms”，用来保持文件权限；g 即“--group”，用来保持文件的属组信息。

- “--delete”选项指定以 rsync 服务器端为基准进行数据镜像同步，也就是要保持 rsync 服务器端目录与客户端目录的完全一致。在这里以 A 服务器为基准进行同步。
- “--progress”选项用于显示数据镜像同步的过程。
- “--exclude”选项用于排除不需要传输的文件类型。
- “backup@192.168.12.253::ixdba”表示对服务器 192.168.12.253 中的 ixdba 模块进行备份，也就是指定备份的模块。backup 表示使用“backup”这个用户对该模块进行备份。
- “/ixdba.net”用于指定备份文件在客户端机器上的存放路径，也就是将备份的文件存放在备份机的 /ixdba.net 目录下。
- “--password-file=/etc/server.pass”用来指定客户机上存放密码文件的位置，这样在客户端执行同步命令时就无需输入交互密码。注意，这个密码文件的名称和位置可以随意指定，但是在客户端主机上必须存在此文件，文件的内容仅仅为备份用户的密码，这里指的是 backup 用户的密码。

其实，rsync 作为客户端工具，还提供了很多其他的选项和参数，这里仅仅介绍了常用的一部分，更详细的信息请执行“man rsync”命令查看。

如果配置没有错误，接下来 rsync 自动将服务器端（即 A 系统）需要备份的数据同步到客户端（即 B 系统）。rsync 指令在客户端执行完数据的同步后，将自动停止，以后如果服务器端目录下有新增数据，客户端不会自动将数据同步，此时，还需要再次执行 rsync 命令组合进行数据同步，因此，rsync 方式的数据备份需要触发同步指令才能完成。

4. 设置定时备份策略

触发同步指令的方式有很多种，例如，可以将同步指令放入客户端系统的 crontab 守护进程，设定同步时间，然后让 Linux 系统触发同步指令，自动完成数据备份。这种数据备份方式可以用于对数据安全性要求不高的业务系统中。

根据案例的要求，还需要设定客户端 rsync 在每天的凌晨 3 点 30 分执行镜像备份操作，在 B 服务器执行“crontab -e”，然后添加如下信息即可。

```
30 3 * * *  /usr/local/bin/rsync -vzrtopg --delete --progress --exclude "*access*"
    --exclude "debug*"  backup@192.168.12.253::ixdba /ixdba.net  --password-file=/
    etc/server.pass
```

通过以上 4 步操作，一个远程容灾系统已经搭建完成。细心的读者可能发现了，这并不是一个完美的容灾方案，由于 rsync 需要通过触发才能将服务器端数据同步，而触发操作是通过 crontab 守护进程完成的，因此在两次触发同步操作的时间间隔内，服务器端和客户端（即远程容灾系统）的数据可能出现不一致。如果在这个时间间隔内，网站系统出现问题，就意味着数据会丢失。因此，通过这种方式搭建的容灾系统在网站出现故障时是不能完全恢复数据的。

那么，对于对数据安全性要求极高的业务系统，如何才能做到服务器端和客户端数据实时同步呢？很幸运，Linux 2.6.13 以后的内核提供了 inotify 文件系统监控机制，通过 rsync

与 inotify 的组合，完全可以实现 rsync 服务器端和客户端数据的实时同步。

5.3　通过 rsync+inotify 实现数据的实时备份

5.3.1　rsync 的优点与不足

与传统的 cp、tar 备份方式相比，rsync 具有安全性高、备份迅速、支持增量备份等优点，通过 rsync 可以解决对实时性要求不高的数据备份需求，例如，定期地备份文件服务器数据到远端服务器，对本地磁盘定期进行数据镜像等。

随着应用系统规模的不断扩大，对数据的安全性和可靠性提出了更高的要求，rsync 在高端业务系统中也逐渐暴露出了很多不足。首先，rsync 同步数据时，需要扫描所有文件后进行比对，然后进行差量传输。如果文件数量达到了百万甚至千万量级，扫描所有文件将是非常耗时的，而且发生变化的往往是其中很少的一部分，因此 rsync 是非常低效的方式。其次，rsync 不能实时监测、同步数据，虽然它可以通过 Linux 守护进程的方式触发同步，但是两次触发动作一定会有时间差，可能导致服务器端和客户端数据出现不一致，无法在出现应用故障时完全恢复数据。基于以上原因，rsync+inotify 组合出现了！

5.3.2　初识 inotify

inotify 是一种强大的、细粒度的、异步的文件系统事件监控机制，Linux 内核从 2.6.13 版本起，加入了对 inotify 的支持。通过 inotify 可以监控文件系统中添加、删除、修改、移动等各种细微事件，利用这个内核接口，第三方软件可以监控文件系统下文件的各种变化情况，inotify-tools 就是这样的一个第三方软件。

在上一节中讲到，rsync 可以实现触发式的文件同步，但是通过 crontab 守护进程方式进行触发，同步的数据和实际数据会有差异，而 inotify 可以监控文件系统的各种变化，当文件有任何变动时，会触发 rsync 同步，这样刚好解决了同步数据的实时性问题。

5.3.3　安装 inotify 工具 inotify-tools

由于 inotify 的特性需要 Linux 内核的支持，在安装 inotify-tools 前要先确认 Linux 系统内核是否是 2.6.13 版本以上，如果 Linux 内核低于 2.6.13 版本，就需要重新编译内核加入对 inotify 的支持。可以用如下方法判断内核是否支持 inotify：

```
[root@localhost webdata]# uname -r
2.6.18-164.11.1.el5PAE
[root@localhost webdata]# ll /proc/sys/fs/inotify
总计 0
```

```
-rw-r--r-- 1 root root 0 04-13 19:56 max_queued_events
-rw-r--r-- 1 root root 0 04-13 19:56 max_user_instances
-rw-r--r-- 1 root root 0 04-13 19:56 max_user_watches
```

如果有上面3项输出，就表示系统默认支持inotify，可以开始安装inotify-tools了。

可以到http://inotify-tools.sourceforge.net/下载相应版本的inotify-tools，然后开始编译安装。过程如下：

```
[root@localhost  ~]# tar zxvf inotify-tools-3.14.tar.gz
root@localhost  ~]# cd inotify-tools-3.14
[root@localhost inotify-tools-3.14]# ./configure
[root@localhost inotify-tools-3.14]# make
[root@localhost inotify-tools-3.14]# make install
[root@localhost inotify-tools-3.14]# ll /usr/local/bin/inotifywa*
-rwxr-xr-x 1 root root 37264 04-14 13:42 /usr/local/bin/inotifywait
-rwxr-xr-x 1 root root 35438 04-14 13:42 /usr/local/bin/inotifywatch
```

安装完inotify-tools后，会生成inotifywait和inotifywatch两个指令。其中，inotifywait用于等待文件或文件集上的一个特定事件，可以监控任何文件和目录设置，并且可以递归地监控整个目录树；inotifywatch用于收集被监控的文件系统统计数据，包括每个inotify事件发生多少次等信息。

5.3.4 inotify相关参数

inotify定义了一些接口参数，可以用来限制inotify消耗kernel memory的大小。由于这些参数都是内存参数，因此，可以根据应用需求，实时调节其大小。下面对inotify相关参数进行简单介绍。

- “/proc/sys/fs/inotify/max_queued_events”表示调用inotify_init时分配到inotify instance中可排队的event数的最大值，超出这个值的事件被丢弃，但会触发IN_Q_OVERFLOW事件。
- “/proc/sys/fs/inotify/max_user_instances”表示每一个real user ID可创建的inotify instatnces数量的上限。
- “/proc/sys/fs/inotify/max_user_watches”表示每个inotify实例相关联的watches的上限，也就是每个inotify实例可监控的最大目录数量。如果监控的文件数目巨大，需要根据情况适当增加此值的大小。例如：

```
echo 30000000 > /proc/sys/fs/inotify/max_user_watches
```

5.3.5 inotifywait相关参数

inotifywait是一个监控等待事件，可以配合shell脚本使用它。下面介绍一下常用的inotifywait参数：

- -m，即“—monitor”，表示始终保持事件监听状态。
- -r，即“—recursive”，表示递归查询目录。
- -q，即“—quiet”，表示打印出监控事件。
- -e，即“—event”，通过此参数可以指定要监控的事件，常见的事件有 modify、delete、create 和 attrib 等。

更详细的介绍请参看 man inotifywait。

5.3.6　企业应用案例：利用 rsync+inotify 搭建实时同步系统

案例描述

这是一个 CMS 内容发布系统，后端采用负载均衡集群部署方案，由一个负载调度节点、3 个服务节点及一个内容发布节点构成。内容发布节点负责将用户发布的数据生成静态页面，同时将静态网页传输给 3 个服务节点，而负载调度节点负责将用户请求根据负载算法调度到相应的服务节点上，实现用户访问。用户要求在前端访问到的网页数据始终是最新的、一致的。

解决方案

为了保证用户访问到的数据的一致性和实时性，必须保证 3 个服务节点上的数据与内容发布节点上的数据始终是一致的，这就需要通过文件同步工具来实现，这里采用 rsync。同时又要保证数据是实时的，这就需要 inotify，即利用 inotify 监视内容发布节点文件的变化，如果文件有变动，那么就启动 rsync，将文件实时同步到 3 个服务节点上。

系统环境

这里所有服务器均采用 Linux 操作系统，系统内核版本与节点信息如表 5-1 所示。

表 5-1　Linux 操作系统的内核版本与节点信息

节点名称	内核版本	用途	IP 地址	网页数据路径
Web1	2.6.18-164.el5PAE	服务节点 1	192.168.12.131	/web1/wwwroot
Web2	2.6.18-164.el5PAE	服务节点 2	192.168.12.132	/web2/wwwroot
Web3	2.6.18-164.el5PAE	服务节点 3	192.168.12.133	/web3/wwwroot
Server	2.6.18-164.el5PAE	内容发布节点	192.168.12.134	/web/wwwroot

1. 安装 rsync 与 inotify-tools

inotify-tools 是用来监控文件系统变化的工具，因此必须安装在内容发布节点上，服务节点上无需安装 inotify-tools。另外需要在 Web1、Web2、Web3 和 Server 节点上安装 rsync，安装非常简单，这里不再讲述。

在这个案例中，内容发布节点（即 Server）充当了 rsync 客户端的角色，而 3 个服务节点充当了 rsync 服务器端的角色，整个数据同步的过程其实就是一个从客户端向服务器端发

送数据的过程。这一点与前面讲述的案例（5.2.2 节）刚好相反。

2. 在 3 个服务节点上配置 rsync

这里给出 3 个服务节点的 rsync 配置文件以供参考，读者可根据实际情况自行修改。

Web1 节点的 rsyncd.conf 配置如下：

```
uid = nobody
gid = nobody
use chroot = no
max connections = 10
strict modes = yes
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsync.lock
log file = /var/log/rsyncd.log

[web1]
path = /web1/wwwroot/
comment = web1 file
ignore errors
read only = no
write only = no
hosts allow = 192.168.12.134
hosts deny = *
list = false
uid = root
gid = root
auth users = web1user
secrets file = /etc/web1.pass
```

Web2 节点的 rsyncd.conf 配置如下：

```
uid = nobody
gid = nobody
use chroot = no
max connections = 10
strict modes = yes
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsync.lock
log file = /var/log/rsyncd.log

[web2]
path = /web2/wwwroot/
comment = web2 file
ignore errors
read only = no
write only = no
hosts allow = 192.168.12.134
hosts deny = *
list = false
uid = root
```

```
gid = root
auth users = web2user
secrets file = /etc/web2.pass
```

Web3 节点的 rsyncd.conf 配置如下：

```
uid = nobody
gid = nobody
use chroot = no
max connections = 10
strict modes = yes
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsync.lock
log file = /var/log/rsyncd.log

[web3]
path = /web3/wwwroot/
comment = web3 file
ignore errors
read only = no
write only = no
hosts allow = 192.168.12.134
hosts deny = *
list = false
uid = root
gid = root
auth users = web3user
secrets file = /etc/web3.pass
```

在配置完 3 台服务节点的 rsyncd.conf 文件后，依次启动 rsync 守护进程。接着将 rsync 服务加入到自启动文件中。

```
echo "/usr/local/bin/rsync --daemon" >>/etc/rc.local
```

到此为止，3 个 Web 服务节点已经配置完成。

3. 配置内容发布节点

配置内容发布节点的主要工作是将生成的静态网页实时同步到集群中 3 个服务节点上，这个过程可以通过一个 shell 脚本来完成。脚本内容大致如下：

```
#!/bin/bash
host1=192.168.12.131
host2=192.168.12.132
host3=192.168.12.133

src=/web/wwwroot/
dst1=web1
dst2=web2
dst3=web3
user1=web1user
user2=web2user
```

```
user3=web3user

/usr/local/bin/inotifywait -mrq --timefmt '%d/%m/%y %H:%M' --format '%T %w%f%e'
    -e modify,delete,create,attrib  $src \
| while read files
          do
          /usr/bin/rsync -vzrtopg --delete --progress --password-file=/etc/server.
              pass $src $user1@$host1::$dst1
          /usr/bin/rsync -vzrtopg --delete --progress --password-file=/etc/server.
              pass $src $user2@$host2::$dst2
          /usr/bin/rsync -vzrtopg --delete --progress --password-file=/etc/server.
              pass $src $user3@$host3::$dst3
                  echo "${files} was rsynced" >>/tmp/rsync.log 2>&1
          done
```

脚本中相关参数如下：

- --timefmt：指定时间的输出格式。
- --format：指定变化文件的详细信息。

这两个参数一般配合使用，通过指定输出格式输出类似以下的内容：

```
15/04/10 00:29 /web/wwwroot/ixdba.shDELETE ISDIR was rsynced
15/04/10 00:30 /web/wwwroot/index.htmlMODIFY was rsynced
15/04/10 00:31 /web/wwwroot/pcre-8.02.tar.gzCREATE was rsynced
```

这个脚本的作用就是通过 inotify 监控文件目录的变化，进而触发 rsync 进行同步操作。由于这个过程是一个主动触发操作的过程，是通过系统内核完成的，所以，比那些遍历整个目录的扫描方式效率要高很多。

有时会遇到这样的情况：向 inotify 监控的目录（这里是 /web/wwwroot/）中写入一个很大的文件，当写入这个大文件需要一段时间时，inotify 会持续不停地输出该文件被更新的信息，这样就会持续不断地触发 rsync 执行同步操作，占用大量系统资源。针对这种情况，最理想的做法是等待文件写完后再触发 rsync 同步。在这种情况下，可以修改 inotify 的监控事件，即“-e close_write,delete,create,attrib”。

接着，将这个脚本命名为 inotifyrsync.sh 后放到 /web/wwwroot 目录下，然后为其指定可执行权限，放到后台运行。过程如下：

```
chmod 755 /web/wwwroot/inotifyrsync.sh
/web/wwwroot/inotifyrsync.sh &
```

最后，将此脚本加入系统自启动文件。过程如下：

```
echo  “/web/wwwroot/inotifyrsync.sh &” >>/etc/rc.local
```

这样就完成了内容发布节点上的所有配置工作。

4. 测试 rsync+inotify 实时同步功能

完成所有配置后，可以在网页发布节点的 /web/wwwroot 目录下添加、删除或修改某个

文件，然后到 3 个服务节点对应的目录中查看文件是否随网页发布节点中 /web/wwwroot 目录下文件的变化而变化，如果看到 3 个服务节点对应的目录文件与内容发布节点目录文件同步变化了，那么这个业务系统就配置成功了。

5.4　unison 简介

通过前面的介绍可知，rsync 数据镜像方式是单向同步的，也就是说，客户端只保持与服务器端同步，而在客户端增加或者删除一些文件时，并不会更新到服务器端。但是，有些时候，用户希望保持客户端和服务器端双向同步，即任何一端发生数据变化，都会更新到另一端。此时不妨试试 unison 这个工具。

unison 是一个双向同步镜像工具，在 Windows 和 UNIX 平台下都可以使用，支持跨平台同步。unison 可以使本地磁盘的两个文件夹保持内容一致，同时也支持网络数据同步。在内部实现上，unison 使用 OCaml 语言进行开发，通过基于 rsync 的算法对两端文件进行比较，将这两端的文件更新到一致的状态。unison 的几个显著特点如下：

- 跨平台使用。
- 对内核和用户权限没有特别要求。
- unison 是双向的，它能自动更新两份副本中没有冲突的部分，有冲突的部分将会显示出来由用户选择更新策略。
- 只要是能连通的两台主机，就可以运行 unison。unison 提供两种方法进行远程通信，一种是远程 shell 方式（Remote shell method），另一种是 socket 方式（socket method）。远程 shell 方式可以由工具 ssh 来完成，这种方式简单且安全，对带宽要求也不高，可以使用类似 rsync 的压缩传输协议。socket 方式要求通过发送 tcp 包在两地之间进行通信，而且，这种方式不能保证数据传输安全，不建议使用。
- 支持增量同步，每次同步完成后会将文件状态记录下来，在下次同步时，以上次记录的状态为起点开始同步。

unison 有字符界面和基于 GTK+ 的图形界面，这里只介绍如何在字符界面下使用 unison。

5.5　安装 unison

unison 的官方网站是 http://www.seas.upenn.edu/~bcpierce/unison/，在这个地址可以下载各种平台、各种版本的 unison，如基于源码安装的或基于二进制的。二进制版本的 unison 虽然可以直接使用，但版本一般都比较老，更新很慢。为了使用最新的 unison 版本，这里主要介绍一下 unison 的源码安装方式。

在 Linux 下通过源码包安装 unison 时，首先需要安装一个名为 Objective Caml compiler

的工具，ocaml 的版本至少为 3.0.7 或更高，可以从 http://caml.inria.fr/ 下载相应的 ocaml 版本。

ocaml 的安装过程如下：

```
[root@filedata unison]# tar -zxf ocaml-3.10.2.tar.gz
[root@filedata unison]#cd ocaml-3.10.2
[root@filedata ocaml-3.10.2]#./configure
[root@filedata ocaml-3.10.2]#make world opt
[root@filedata ocaml-3.10.2]#make install
```

安装完 ocaml 后，就可以安装 unison 了。这里选择 unison 的最新稳定版本，即 unison-2.32.52.tar.gz。安装过程如下：

```
[root@filedata unison]#tar -zxvf unison-2.32.52.tar.gz
[root@filedata unison]#cd unison-2.32.52
[root@filedata unison-2.32.52]#make UISTYLE=text THREADS=true STATIC=true
```

“UISTYLE=text THREADS=true STATIC=true”表示：使用命令行方式，加入线程支持，以静态模式编译。

在执行完上面命令后，会在当前目录下生成可执行文件 unison，将其复制到系统的 PATH 路径即可。

```
[root@filedata unison-2.32.52]#cp unison /usr/local/bin
```

5.6 配置双机 ssh 信任

由于 unison 在同步远程文件夹时要登录到远程服务器，因此要配置两机互相信任，这里进行如下假定：

本地机：192.168.12.235（Linux 操作系统），主机名：filedata1

远程机：192.168.12.237（Solaris 操作系统），主机名：filedata2

5.6.1 在两台机器上创建 RSA 密钥

以下操作要在本地机和远程机上都执行一遍，这里以 filedata1 为例进行介绍。

1）以 root 用户登录。

2）在 root 用户的主目录内创建 .ssh 目录并设置正确的权限。

```
[root@filedata1 ~]# mkdir ~/.ssh
[root@filedata1 ~]# chmod 700 ~/.ssh
```

3）使用 ssh-keygen 命令生成第 2 版的 SSH 协议的 RSA 密钥。

```
[root@filedata1 ~]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

```
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
17:e4:7c:79:8d:a0:00:3b:d9:f7:7a:56:f3:ac:54:4d root@filedata1
```

在提示保存私钥（key）和公钥（public key）的位置时，使用默认值。 如果需要私钥密码（passphrase），则输入一个私钥密码（如果使用私钥密码，在利用 ssh 执行远程命令时需要输入私钥密码，本案例未使用私钥密码，因此，直接按回车键即可）。

5.6.2　添加密钥到授权密钥文件中

1）以 root 用户登录。

2）在本地机上执行，过程如下：

```
[root@filedata1 ~]# cd ~/.ssh
[root@filedata1 .ssh]#ssh 192.168.12.235 cat /root/.ssh/id_rsa.pub >> authorized_keys
[root@filedata1 .ssh]#ssh 192.168.12.237 cat /root/.ssh/id_rsa.pub >> authorized_keys
[root@filedata1 .ssh]#scp authorized_keys 192.168.12.237:/root/.ssh/
[root@filedata1 .ssh]# chmod 600  /root/.ssh/authorized_keys
```

3）在远程机 192.168.12.237 上执行如下操作：

```
bash-2.05# chmod 600  /root/.ssh/authorized_keys
```

4）分别在两台机器上执行如下测试：

```
[root@filedata1 ~]# ssh 192.168.12.235  date
[root@filedata1 ~]# ssh 192.168.12.237  date
bash-2.05#ssh 192.168.12.237 date
bash-2.05#ssh 192.168.12.235 date
```

在第一次执行时，会要求输入密码信息，再次执行时，不需要输入密码就能显示系统日期，这说明 ssh 互相信任配置成功。

5.7　unison 的使用

unison 的用法非常灵活和简单，可以通过如下三种方式调用 unison。

第一种方式：“unison profile_name [options] ”

unison 默认会读取 ~/.unison 目录下的配置文件“profile_name.prf”。

注意，在这种方式下，命令行中并没有指出要进行文件同步的两个地址，所以，此种调用 unison 的方式必须在配置文件 profile_name.prf 中通过相关的 root 指令设置同步的路径和同步参数，如：

```
# Roots of the synchronization
root = /ixdba/webdata
root = ssh://root@192.168.12.237//ixdba/webdata
```

```
#force =/ixdba/webdata
ignore = Path upload/*
#prefer = ssh://root@192.168.12.237//ixdba/webdata
batch = true
```

此文件中涉及的相关参数将在下面进行详细讲述。

第二种方式："unison profile root1 root2 [options] "

root1、root2 分别表示要执行同步的两个路径。这两个路径可以是本地目录路径，也可以是远程服务器的路径，如 ssh://username@remotehost//home/ixdba/files。由于同步的路径已经在命令行指定了，所以这时无需在 profile.prf 配置文件中进行 root 指令的相关设置。

第三种方式："unison root1 root2 [options] "。

这种方式相当于执行 "unison default root1 root2" 命令，即 unison 默认读取 "default.prf" 的配置。

了解了 unison 的几种调用方式后，接下来通过几个示例来详细讲解 unison 的使用方法。uison 可以在一台主机上使用，同步两个文件夹；也可以在网络上使用，同步两个网络文件夹。下面分别进行介绍。

5.7.1 本地使用 unison

先看一个简单的例子：对主机 filedata1 上的文件夹 /test1 和 /test2 进行数据镜像同步。执行的操作如下：

```
[root@filedata1  /]# unison /test1  /test2
Contacting server...
Connected [//filedata1//test1 -> //filedata1//test2]
Looking for changes
Reconciling changes
test1           test2
new file ---->          httpd.conf   [f] f
Proceed with propagating updates? [] y
Propagating updates
UNISON 2.32.52 started propagating changes at 23:58:51 on 27 Sep 2010
[BGN] Copying httpd.conf from /test1 to /test2
[END] Copying httpd.conf
UNISON 2.32.52 finished propagating changes at 23:58:51 on 27 Sep 2010
Saving synchronizer state
Synchronization complete at 23:58:51  (1 item transferred, 0 skipped, 0 failed)
```

从输出可以看到，unison 在检测完两个文件夹后，如果检测到文件夹内容存在不同，会提示选择相应的操作。重点查看代码中的斜体加粗部分：表示左边 test1 的文件夹中有新的文件；确认是否同步到右边的 test2 文件夹中，默认参数是 f（表示 force），即强制同步。输入 "f" 参数后，会继续提示是否确认同步，输入 "y" 将确认同步，然后开始进行更新；如果输入 "?" 会有更详细的介绍。

5.7.2　远程使用 unison

基本使用方法为：

```
unison < 本地目录 > ssh://remotehostname(IP)/< 远程目录的绝对路径 >
```

例如：

```
unison /home/AAA ssh://username@remotehostname(ip)//DB/path/BBB
```

表示将本机的目录 /home/AAA 和远端主机的 /DB/path/BBB 同步。由于是通过 ssh 方式建立的连接，因此同步的两台主机必须开启 ssh 连接服务。

需要特别注意的是：在远程主机和目录之间多加了一个“/”。下面是一个操作示例：

```
[root@filedata1 /]# unison /test1 ssh://root@192.168.12.237//mnt
Contacting server...
root@192.168.12.237's password:
Connected [//filedata2//mnt -> //filedata1//test1]
Looking for changes
Warning: No archive files were found for these roots, whose canonical names are:
         /test1
         //filedata2//mnt
Waiting for changes from server
Reconciling changes

local          filedata2
file     ---->            httpd.conf  [f]
    <---- file       kernel-2.6.18-194.11.1.el5.i686.rpm  [f] f
    <---- file       kernel-PAE-2.6.18-194.11.1.el5.i686.rpm  [f] f
    <---- file       kernel-PAE-devel-2.6.18-194.11.1.el5.i686.rpm  [f] f
    <---- file       kernel-debug-2.6.18-194.11.1.el5.i686.rpm  [f] f
Proceed with propagating updates? [] y
Propagating updates

UNISON 2.32.52 started propagating changes at 00:18:00 on 28 Sep 2010
[BGN] Copying httpd.conf from /test1 to //filedata2//mnt
[BGN] Copying kernel-2.6.18-194.11.1.el5.i686.rpm from //filedata2//mnt to /test1
[BGN] Copying kernel-PAE-2.6.18-194.11.1.el5.i686.rpm from //filedata2//mnt to /test1
[BGN] Copying kernel-PAE-devel-2.6.18-194.11.1.el5.i686.rpm from //filedata2//mnt to /test1
[BGN] Copying kernel-debug-2.6.18-194.11.1.el5.i686.rpm from //filedata2//mnt to /test1
[END] Copying httpd.conf
[END] Copying kernel-2.6.18-194.11.1.el5.i686.rpm
[END] Copying kernel-PAE-2.6.18-194.11.1.el5.i686.rpm
[END] Copying kernel-PAE-devel-2.6.18-194.11.1.el5.i686.rpm
[BGN] Copying kernel-debug-2.6.18-194.11.1.el5.i686.rpm
Saving synchronizer state
Synchronization complete at 00:18:11  (5 items transferred, 0 skipped, 0 failed)
```

这个同步操作是将本地主机下的 /test1 目录与远程主机 192.168.12.237 下的 /mnt 目录进行同步，同步完成后，两个目录数据就完全一致了。特别注意上面输出中斜体加粗部分的内容。

5.7.3 unison 参数说明

unison 有很多参数，这里只介绍常用的几个。

1. testserver 参数

这个参数用来测试连通性，连接到服务器后就退出。例如：

```
[root@filedata1 /]# unison / ssh://root@192.168.12.237/ -testserver
Contacting server...
root@192.168.12.237's password:
Connected [//drbd1//root -> //drbd2//]
```

这个输出表示主机 filedata1 与 192.168.12.237 可以连通。成功连接后自动退出，不会执行目录的比较等后续操作。

有时候可能会出现这样的问题，如果远程主机中的 unison 可执行文件不在系统的默认路径下，在做连通测试时就会报错。例如：

```
[root@filedata1 /]# unison / ssh://root@192.168.12.237/ -testserver
Contacting server...
root@192.168.12.237's password:
bash: unison: command not found
Fatal error: Lost connection with the server
```

出错提示是“bash: unison: command not found”，其实 unison 命令是存在的，只是不在系统默认的路径下，此时就需要使用另一个参数“- servercmd”来告诉 unison 命令的位置。

2. servercmd 参数

这个参数用来告诉 unison 服务器端的 unison 命令是什么。例如：

```
[root@drbd2 /]# unison / ssh://root@192.168.12.237/ -testserver -servercmd=/mnt/unison
Contacting server...
root@192.168.12.237's password:
Connected [//drbd1//root -> //drbd2//]
```

3. auto 参数

使用规则：-auto

这个参数表示接受默认的动作，然后等待用户确认是否执行。

4. batch 参数

使用规则：-batch

这个参数表示全自动模式，接受默认动作并自动执行，无需人为干预。

5. ignore

使用规则：ignore xxx

这个参数表示同步时可以忽略的目录或者路径，增加 xxx 到忽略列表中。

6. ignorecase 参数

使用规则：ignorecase [true|false|default]

这个参数表示是否忽略文件名大小写，可选项有 3 个。

7. follow 参数

使用规则：follw xxx

这个参数表示是否支持对符号链接指向的内容同步，如果在同步时配置此参数，那么 unison 将在同步时跟踪符号链接指向的实际文件，然后进行同步。

8. path 参数

使用规则：path xxx

这个参数表示只同步指定的子目录及文件，而非整个目录。“-path”在配置中可以多次出现，例如：

```
unison /home/username ssh://remotehost//home/username \
-path shared \
-path classes \
-path .ssh/auto.fsck
```

9. 其他参数

- owner = true，表示保持同步的文件属主信息。
- group = true，表示在同步过程中保持同步的文件组信息。
- perms = −1，表示在同步过程中保持同步文件的读写权限。
- repeat = 1，表示间隔 1 秒后开始一次新的同步检查。
- retry = 3，指定失败重试次数。
- sshargs = −C，表示使用 ssh 的压缩传输方式。
- xferbycopying，这是个优化传输参数，默认值为 true。
- immutable xxx，指定不变化的目录，扫描时可以忽略。
- silent，除了错误，不打印任何信息。
- times，表示同步修改时间。
- maxthreads n，指定文件同步的最大线程数。
- rsync，默认值是 true，用于激活 rsync 传输模式。
- log，表示记录 unison 运行日志，默认值是 true。
- logfile，将 unison 日志信息输出到指定文件中。

5.7.4　通过配置文件来使用 unison

尽管完全可以通过命令行的方式来指定 unison 运行所需要的参数，但还是推荐使用配置文件来使用 unison，原因很简单，配置文件比命令行容易理解，而且可管理性更强。

1. .unison 目录

.unison 目录用于保存 unison 的配置文件以及 Archive 文件。在 Linux 系统中，默认的配置文件位于 ~currentuser/.unison 下，也就是当前用户的 home 目录下，如果是 root 用户，则位于 /root/.unison 目录下；在 Windows 系统中，配置文件则位于 C:\Documents and Settings\currentuser\.unison 下，如果不指定配置文件名称，则默认的配置文件名是 default.prf。在 .unison 目录下可以有多个配置文件，配置文件的扩展名为 .prf。

Archive 文件也可以有多个，这个文件记录了每次完成同步后每个文件的状态，可以在下次的更新动作中更快速地判断文件是否应该更新，减少了完全扫描所有文件消耗的时间，使 unison 同步速度加快。

2. *.prf 配置文件

运行如下命令：

```
[root@filedata1 /]#unison ixdba
```

unison 将默认读取 ~currentuser/.unison/ixdba.prf 文件中的配置信息。在 root 用户下，此配置文件的路径是 /root/.unison/ ixdba.prf，因此可以将上一节介绍的参数配置信息写入一个 .prf 的文件中。

下面将在主机 filedata1 和 filedata2 之间搭建一个远程文件镜像服务，前提是需要配置两个主机之间互信，这在前面已经进行了详细的介绍，这里不再多说。下面显示两台主机进行远程数据同步的配置文件中的内容。

```
root = /ixdba/webdata
root = ssh://root@192.168.12.237//ixdba/webdata
#force =/ixdba/webdata
path= www
path= upload
ignore = Path WEB-INF/tmp
#prefer = ssh://root@192.168.12.237//ixdba/webdata
batch = true
maxthreads = 300
#repeat = 1
#retry = 3
owner = true
group = true
perms = -1
fastcheck=false
rsync =false
#debug=verbose
sshargs = -C
xferbycopying = true
confirmbigdel= false
log = true
logfile = /root/.unison/ixdba_10.10.log
```

下面对这段配置进行解释。

- 两个 root 表示需要同步的文件夹。
- force 表示以本地的 /ixdba/webdata 文件夹为标准，将该目录同步到远端。注意，如果指定了 force 参数，那么 unison 就变成单项同步了，也就是说以 force 指定的文件夹为准进行同步。
- unison 本身是可以双向同步的，但是要做到双向同步，就不要设置 force 参数，如果设置了 force 参数，就成单项同步了，此时 unison 类似于 rsync。
- 两个 path 指定了需要同步的两个目录分别是 /ixdba/webdata/www 和 /ixdba/webdata/upload。
- ignore = Path 表示忽略 /ixdba/webdata 下面的 WEB-INF/tmp 目录，即同步时不同步它。注意，这里是“Path”，而不是“path”。
- batch = true 表示全自动模式，接受并执行默认动作。
- maxthreads 指定了同步时的最大线程数，unison 的新版本支持多线程同步。
- fastcheck 参数有 true 和 false 两个选项，true 表示同步时通过文件的创建时间来比较两地文件；如果这个选项为 false，unison 则将比较两地文件的内容。建议设置为 true。
- confirmbigdel 参数的默认值是 true，表示当需要同步的两个目录有一个为空时，unison 将停止。这里建议设置为 false，可以保证当需要同步的某个目录为空时，unison 不会停止运转。
- log = true 表示在终端输出运行信息。
- logfile 指定了同时将输出写入 log 文件。

unison 双向同步的基本原理是：假如有 A、B 两个文件夹，A 文件夹将自己的改动同步到 B，B 文件夹也把自己的改动同步到 A，最后 A、B 两个文件夹的内容相同，都是原来 A 和 B 文件夹的合集。

unison 双向同步的一个缺点是，一个文件在两个同步文件夹中都被修改时，unison 不会进行同步，因为 unison 无法判断以哪个为准，需要人工指定。

5.8　本章小结

本章主要介绍了两个开源数据镜像备份工具 rsync 和 unison 的使用。先介绍了两个软件的安装和配置，然后通过两个实际的企业案例演示了这两个工具在生产环境中的使用过程。

rsync 可以实现数据的单项同步，通过触发机制实现本地和异地数据的定时镜像备份。而 unison 是一个双向同步工具，可以实现本地或异地数据的互相同步，并且可以实现增量备份。

Linux 下的数据备份工具有很多，但多数是商业性的软件，价格比较昂贵，对很多中小企业来说，可能无法承受，rsync 和 unison 就成为他们备份软件时的首选。通过合理的整合和配置，rsync 和 unison 的高效性完全可以与商业备份软件相媲美。

第 6 章　ext3 文件系统反删除利器 ext3grep

Linux 作为企业级服务器，数据的安全性至关重要，任何数据的丢失和误删除都是不可容忍的。做为系统管理员，一定要有数据保护意识，不但要对服务器数据进行定期备份，而且还要具有在误删除数据后将其快速恢复的技能。本章重点讲述 Linux 下的 ext3 文件系统中用于数据恢复的开源软件 ext3grep。通过这个软件，可以快速、准确地恢复误删除的数据。最后，通过两个实例具体介绍利用 ext3grep 恢复数据的详细过程。

6.1　“rm –rf”带来的困惑

国外一份非常著名的 Linux 系统管理员守则中有这么一条：“慎用 rm –rf 命令，除非你知道此命令将带来什么后果”。可见，这个命令对系统管理员的重要性。在实际的工作中，由此命令带来的误删除数据的案例屡见不鲜，很多系统管理员都遇到过或者犯过这样的错误。由于开发人员对命令不熟悉，或者粗心大意、疏于管理，执行了此命令，数据在一瞬间就被清空了。Linux 不具备类似回收站的功能，这就意味着数据会丢失。虽然 Linux 自身提供了恢复数据的机制，但是这个功能基本没用，要恢复数据，通过常规手段是无法完成的，此时，只有找专业的数据恢复公司来恢复数据，这样无疑要付出很大的成本，造成无法估量的损失。

可见，作为系统管理员，一定要有数据安全意识，数据保护意识，严格遵守相关维护守则，将这种失误带来的损失降低到最低。幸运的是，Linux 下提供了一款恢复误删数据的开源软件，利用这个 ext3 文件系统数据恢复工具 ext3grep 可以恢复误删除的数据。

6.2　ext3grep 的安装与使用

ext3grep 是一个开源的 ext3 文件系统反删除工具。在 ext3grep 出现之前，数据被删除后，通过常规手段恢复基本上是不可能的，虽然 debugfs 命令可以对 ext2 文件系统做一些恢复，但是对 ext3 文件系统就无能为力了。ext3 是一个日志型文件系统，ext3grep 正是通过分析 ext3 文件系统的日志信息来恢复被删除的文件和数据的。

6.2.1　ext3grep 的恢复原理

在介绍 ext3grep 恢复原理之前，先介绍一下文件系统中 inode 的一些相关知识。inode

是文件系统组成的最基本单元，是文件系统连接任何子目录、任何文件的桥梁。它包括了文件系统中文件基本属性和存放数据的位置等相关信息。每个文件由两部分组成：一部分是 inode，另一部分是 block。block 用来存储数据，inode 用来存储数据索引信息，包括文件大小、读写权限、属主、归属的用户组等。操作系统根据用户指令，通过 inode 值就能很快找到对应的文件。

打个比方，存储设备或磁盘分区就相当于一本书，block 相当于书中的每一页，inode 相当于这本书的目录。一本书有很多内容，要查找某部分的内容，先查找目录，然后就能很快定位到想要查看的内容。

在 Linux 下可以通过“ls –id”命令来查看某个文件或者目录的 inode 值。例如查看根目录的 inode 值，可以输入：

```
[root@localhost /]# ls -id  /
2 /
```

由此可知，根目录的 inode 值为 2。

利用 ext3grep 恢复文件时并不依赖特定文件格式。首先 ext3grep 通过文件系统的 root inode（根目录的 inode 一般为 2）来获得当前文件系统下所有文件的信息，包括存在的和已经删除的文件，这些信息包括文件名和 inode。然后利用 inode 信息结合日志去查询该 inode 所在的 block 位置，包括直接块、间接块等信息。最后利用 dd 命令将这些信息备份出来，从而恢复数据文件。

6.2.2　ext3grep 的安装过程

操作系统环境：CentOS release 5.4。

ext3grep 版本：ext3grep-0.10.1。

ext3grep 官方网站：http://code.google.com/p/ext3grep/，可以从这里下载最新的 ext3grep 版本。这里下载的是 ext3grep-0.10.1.tar.gz。

所需的系统相关包：

- [root@localhost ~]# rpm -qa |grep e2fsprogs
- e2fsprogs-libs-1.39-8.el5
- e2fsprogs-1.39-8.el5
- e2fsprogs-devel-1.39-8.el5

系统必须安装 e2fsprogs-libs，不然后面 ext3grep 的安装会出现问题。

下面进入编译安装阶段，过程如下：

```
[root@localhost /opt]# tar zxvf ext3grep-0.10.1.tar.gz
[root@localhost ext3grep-0.10.1]# ./configure
[root@localhost ext3grep-0.10.1]# make
[root@localhost ext3grep-0.10.1]# make install
```

```
[root@localhost ext3grep-0.10.1]# ext3grep  -v
Running ext3grep version 0.10.1
```

这样，ext3grep 就安装完成了，默认的 ext3grep 命令放在 /usr/local/bin 目录下。ext3grep 的使用非常简单，这里不做介绍，可以通过“ext3grep --help”获取详细的使用帮助。

6.3 通过 ext3grep 恢复误删除的文件与目录

6.3.1 数据恢复准则

当发现某个分区的数据被误删除后，要做的第一件事是立刻卸载被误删除文件所在的分区，或者重新以只读方式挂载此分区。

这么做的原因其实很简单：删除一个文件，就是将文件 inode 节点中的扇区指针清除，同时，释放这些数据对应的数据块，而真实的文件还存留在磁盘分区中。但是这些被删除的文件不一定会一直存留在磁盘中，当这些释放的数据块被操作系统重新分配时，那些被删除的数据就会被覆盖。因此，在数据误删除后，马上卸载文件所在分区可以降低数据块中数据被覆盖的风险，进而提高成功恢复数据的机率。

6.3.2 实战 ext3grep 恢复文件

1. 模拟数据误删除环境

下面通过一个模拟环境，详细介绍利用 ext3grep 恢复数据文件的过程。

```
[root@localhost /]# mkdir /disk                                    # 建立一个挂载点
[root@localhost /]# cd /mydata
[root@localhost mydata]# dd if=/dev/zero  of=/mydata/disk1  count=102400
                                                                   # 模拟磁盘分区，创建一个空设备
102400+0 records in
102400+0 records out
52428800 bytes (52 MB) copied, 1.20597 seconds, 43.5 MB/s
[root@localhost mydata]# mkfs.ext3 /mydata/disk1                   # 将空设备格式化为 ext3 格式
[root@localhost mydata]# mount -o loop /mydata/disk1 /disk  # 挂载设备到 /disk 目录下
[root@localhost mydata]# cd /disk/
[root@localhost disk]# cp /etc/profile  /disk                      # 复制文件到模拟磁盘分区
[root@localhost disk]# cp /boot/initrd-2.6.18-164.11.1.el5xen.img  /disk
[root@localhost disk]# echo "ext3grep  test">ext3grep.txt
[root@localhost disk]# mkdir  /disk/ext3grep
[root@localhost disk]# cp /etc/hosts  /disk/ext3grep
[root@localhost disk]# pwd
/disk
[root@localhost disk]# ls -al
总计 2512
```

```
drwxr-xr-x  4 root root    4096 04-07 16:46 .
drwxr-xr-x 31 root root    4096 04-07 16:45 ..
drwxr-xr-x  2 root root    4096 04-07 16:46 ext3grep
-rw-r--r--  1 root root      14 04-07 16:31 ext3grep.txt
-rw-------  1 root root 2535991 04-07 16:30 initrd-2.6.18-164.11.1.el5xen.img
drwx------  2 root root    4096 04-07 16:33 lost+found
-rw-r--r--  1 root root    1029 04-07 16:30 profile
[root@localhost disk]# md5sum  profile    # 获取文件校验码
a6e82d979bb95919082d9aceddf56c39 profile
[root@localhost disk]# md5sum initrd-2.6.18-164.11.1.el5xen.img
031226080e00d7f312b1f95454e5a1ff  initrd-2.6.18-164.11.1.el5xen.img
[root@localhost disk]# md5sum  ext3grep.txt
5afe55495cdb666daad667e1cd797dcb  ext3grep.txt

[root@localhost disk]# rm -rf /disk/*    # 模拟误删除数据操作
[root@localhost disk]# ls
```

2. 卸载磁盘分区

执行以下命令卸载磁盘分区：

```
[root@localhost disk]# cd /opt            # 切换到 /opt 目录下
[root@localhost /opt]# umount /disk       # 卸载模拟磁盘分区
```

3. 查询恢复数据信息

执行如下命令，查询需要恢复的数据信息：

```
[root@localhost /opt]# ext3grep  /mydata/disk1  --ls --inode 2
```

执行该命令后，ext3grep 就开始搜索可以恢复的数据文件信息，输出如图 6-1 所示。

```
Writing analysis so far to "disk1.ext3grep.stage2". Delete that file if you want to do this stage again.
The first block of the directory is 433.
Inode 2 is directory "".
Directory block 433:
          .-- File type in dir_entry (r=regular file, d=directory, l=symlink)
          |       .-- D: Deleted ; R: Reallocated
Indx Next |  Inode    | Deletion time                    Mode        File name
==========+==========+---------------data-from-inode------+-----------+==========
   0    1 d       2                                      drwxr-xr-x  .
   1  end d       2                                      drwxr-xr-x  ..
   2    3 d      11  D 1270629192 Wed Apr  7 16:33:12 2010  drwx------  lost+found
   3  end r      12  D 1270629192 Wed Apr  7 16:33:12 2010  rrw-r--r--  profile
   4  end r      13  D 1270629192 Wed Apr  7 16:33:12 2010  rrw-------  initrd-2.6.18-164.11.1.el5xen.img
   5  end r      14  D 1270629192 Wed Apr  7 16:33:12 2010  rrw-r--r--  ext3grep.txt
   6  end d    1833  D 1270629192 Wed Apr  7 16:33:12 2010  drwxr-xr-x  ext3grep
```

图 6-1　通过 ext3grep 查询可恢复的数据信息

“ext3grep /mydata/disk1 --ls --inode 2”主要用于扫描当前文件系统下所有文件的信息，包括存在的和已经删除的文件，其中含有 D 标识的就是已被删除的文件，如果不记得被删除的文件的名称，可以通过这种方式来获取要恢复的文件的名称。

通过下面的方式可以获取文件要恢复的路径信息。

```
[root@localhost /opt]# ext3grep  /mydata/disk1  --dump-names
Running ext3grep version 0.10.1
Number of groups: 7
Minimum / maximum journal block: 447 / 4561
```

定的形式。

通过“--restore-inode”参数，只需指定文件对应的 inode 值即可恢复文件。操作如下（其中 inode 值为 12 的是 profile 文件）：

```
[root@localhost RESTORED_FILES]# ext3grep  /mydata/disk1   --restore-inode 12
Running ext3grep version 0.10.1
Number of groups: 7
Minimum / maximum journal block: 447 / 4561
Loading journal descriptors... sorting... done
The oldest inode block that is still in the journal, appears to be from 1270629014
   = Wed Apr  7 16:30:14 2010
Number of descriptors in journal: 63; min / max sequence numbers: 2 / 10
Writing output to directory RESTORED_FILES/
Restoring inode.12
```

下面进入 RESTORED_FILES 目录，验证文件是否成功恢复。

```
[root@localhost /opt]# cd RESTORED_FILES
[root@localhost RESTORED_FILES]# ls
ext3grep  ext3grep.txt  inode.12
[root@localhost RESTORED_FILES]# md5sum ext3grep.txt
5afe55495cdb666daad667e1cd797dcb  ext3grep.txt
[root@localhost RESTORED_FILES]# md5sum inode.12
a6e82d979bb95919082d9aceddf56c39  inode.12
```

根据校验结果可知，这个校验码与文件被删除之前的校验码完全一致，因此，通过这个方式恢复出来的文件是完整的。

5. 恢复所有已删除数据

当需要恢复的文件较少时，通过前面介绍的指定文件的方式进行逐个恢复是可行的。但是如果要恢复很多个文件，如 1000 个以上，还采取逐个指定的方式，效率是非常低下的，此时就要利用 ext3grep 命令的“--restore-all”参数了。具体操作如下：

```
[root@localhost /opt]# ext3grep  /mydata/disk1  --restore-all
Running ext3grep version 0.10.1
Number of groups: 7
Minimum / maximum journal block: 447 / 4561
Loading journal descriptors... sorting... done
The oldest inode block that is still in the journal, appears to be from 1270629014
   = Wed Apr  7 16:30:14 2010
Number of descriptors in journal: 63; min / max sequence numbers: 2 / 10
Loading disk1.ext3grep.stage2... done
Restoring ext3grep.txt
Restoring ext3grep/hosts
Restoring initrd-2.6.18-164.11.1.el5xen.img
Restoring profile
[root@localhost /opt]# cd RESTORED_FILES/
[root@localhost RESTORED_FILES]# ls -al
总计 2512
```

```
drwxr-xr-x  4 root root    4096 04-07 16:46 .
drwxr-xr-x 31 root root    4096 04-07 16:45 ..
drwxr-xr-x  2 root root    4096 04-07 16:46 ext3grep
-rw-r--r--  1 root root      14 04-07 16:31 ext3grep.txt
-rw-------  1 root root 2535991 04-07 16:30 initrd-2.6.18-164.11.1.el5xen.img
drwx------  2 root root    4096 04-07 16:33 lost+found
-rw-r--r--  1 root root    1029 04-07 16:30 profile
```

根据这个输出可知，“--restore-all”参数将指定存储设备中可以恢复的文件都恢复出来并放到了 RESTORED_FILES 目录中。“--restore-all”参数对恢复大量数据文件是非常有用的。

6.4 通过 ext3grep 恢复误删除的 MySQL 表

6.4.1 MySQL 存储引擎介绍

MySQL 数据库在存储管理方面，有多个存储引擎可以选择，MyISAM 是默认的存储引擎，它具有高速存储和检索及全文搜索能力，但不支持事务处理和故障恢复，因此，在误删除一个表后，如果没有备份，那么数据就永远丢失了。另一个常用的存储引擎是 InnoDB，它具有对事务的回滚和崩溃恢复能力，但是仅仅限于事务管理方面，在对恢复库或表的误删除操作方面有很大的局限性。

MySQL 采用 MyISAM 存储引擎时，每张表分别对应 3 个文件，分别是以 MYI 为后缀的索引文件、以 frm 为后缀的结构文件、以 MYD 为后缀的数据文件。恢复 MySQL 表的过程，其实就是恢复这 3 个文件的过程。

6.4.2 模拟 MySQL 表被误删除的环境

下面介绍在采用的是 MyISAM 存储引擎的 MySQL 中模拟表被误删除后的恢复过程。

这里设定：MySQL 所在的磁盘分区为 /dev/sda6，挂载到 /data 目录下，而 MySQL 的安装目录为 /data/mysql。下面进入实例介绍。

1. 查看 MySQL 数据库表信息

首先登录 MySQL 数据库查看 cicro 库中相关的表信息，操作如图 6-2 所示。

接着查看 t_manager 表的内容及数据结构，操作如图 6-3 所示。

2. 模拟误删除操作，删除表 t_manager

```
mysql>drop table t_manager;
Query OK, 0 rows affected (0.00 sec)
mysql>exit
```

```
[root@localhost mysql]# bin/mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.30 MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use cicro
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> show tables;
+----------------+
| Tables_in_cicro |
+----------------+
| t_log_opt       |
| t_manager       |
| t_manager_org   |
| t_organization  |
| t_usertable     |
+----------------+
5 rows in set (0.00 sec)
```

图 6-2　查询 cicro 库中所有表

```
mysql> select * from t_manager;
+--------+----------+-----------------+---------------+
| MNG_ID | MNG_NAME | MNG_PASSWORD    | MNG_NOTE      |
+--------+----------+-----------------+---------------+
|      1 | _orga    | =#=mCX1K3g1Ers= | administrator |
+--------+----------+-----------------+---------------+
1 row in set (0.00 sec)

mysql> desc t_manager;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| MNG_ID       | decimal(9,0) | NO   | PRI | NULL    |       |
| MNG_NAME     | varchar(20)  | YES  |     | NULL    |       |
| MNG_PASSWORD | varchar(30)  | YES  |     | NULL    |       |
| MNG_NOTE     | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

图 6-3　查看 t_manager 表的内容和结构

3. 停止 MySQL 数据库，卸载 MySQL 所在分区

```
[root@localhost mysql]# ./mysqld stop
[root@localhost /]# umount /dev/sda6
```

6.4.3　通过 ext3grep 分析数据、恢复数据

1. 对 MySQL 执行分区数据扫描

通过 ext3grep 分析 MySQL 数据所在的分区信息，进而判断是否有可恢复的信息，操作如图 6-4 所示。

```
[root@localhost /]# ext3grep  /dev/sda6  --ls --inode 2
Running ext3grep version 0.10.1
Number of groups: 63
Loading group metadata... done
Minimum / maximum journal block: 530 / 9271
Loading journal descriptors... sorting... done
The oldest inode block that is still in the journal,
appears to be from 1270697526 = Thu Apr  8 11:32:06 2010
Number of descriptors in journal: 3796; min / max sequence numbers: 2 / 22
Inode is Allocated
Loading vm-disk.ext3grep.stage2.... done
The first block of the directory is 516.
Inode 2 is directory "".
Directory block 516:
          .-- File type in dir_entry (r=regular file, d=directory, l=symlink)
          |          .-- D: Deleted ; R: Reallocated
Indx Next |  Inode   | Deletion time                          Mode          File name
==========+==========+---------------data-from-inode------+----------+=========
   0    1 d       2                                        drwxr-xr-x  .
   1    2 d       2                                        drwxr-xr-x  ..
   2    3 d      11                                        drwx------  lost+found
   3  end d   34545                                        drwxr-xr-x  mysql
```

图 6-4 通过 ext3grep 查询 /data 分区可恢复的数据信息

通过图 6-4 可知，MySQL 目录中有可恢复的数据信息。根据查询到的恢复信息，可知 MySQL 目录的 Inode 号是 34545。接着继续扫描 MySQL 目录的 Inode 信息，如图 6-5 所示。

```
[root@localhost /]# ext3grep  /dev/sda6  --ls --inode 34545
Running ext3grep version 0.10.1
Number of groups: 63
Minimum / maximum journal block: 530 / 9271
Loading journal descriptors... sorting... done
The oldest inode block that is still in the journal,
appears to be from 1270697526 = Thu Apr  8 11:32:06 2010
Number of descriptors in journal: 3796; min / max sequence numbers: 2 / 22
Inode is Allocated
Loading vm-disk.ext3grep.stage2.... done
The first block of the directory is 139777.
Inode 34545 is directory "mysql".
Directory block 139777:
          .-- File type in dir_entry (r=regular file, d=directory, l=symlink)
          |          .-- D: Deleted ; R: Reallocated
Indx Next |  Inode   | Deletion time                          Mode          File name
==========+==========+---------------data-from-inode------+----------+=========
   0    1 d   34545                                        drwxr-xr-x  .
   1    2 d       2                                        drwxr-xr-x  ..
   2    3 r   34546                                        rrwxr-xr-x  mysqld
   3    4 r   34547                                        rrw-r--r--  README
   4    5 d   34548                                        drwxr-xr-x  sql-bench
   5    6 d   36577                                        drwxr-x---  data
   6    7 d   42677                                        drwxr-xr-x  docs
   7    8 d   34643                                        drwxr-xr-x  scripts
   8    9 d   34645                                        drwxr-xr-x  mysql-test
   9   10 d   56970                                        drwxr-xr-x  bin
  10   11 d   49200                                        drwxr-xr-x  support-files
  11   12 r   35468                                        rrw-r--r--  EXCEPTIONS-CLIENT
  12   13 r   35469                                        rrw-r--r--  COPYING
  13   14 r   35470                                        rrw-r--r--  INSTALL-BINARY
  14   15 d   49215                                        drwxr-xr-x  share
  15   16 d   63042                                        drwxr-xr-x  lib
  16   17 d   36663                                        drwxr-xr-x  man
  17  end d   36666                                        drwxr-xr-x  include
```

图 6-5 扫描 MySQL 目录下可恢复的数据信息

在上面的操作中，首先通过“--ls --inode 2”参数扫描了整个分区信息，查找到 MySQL 目录对应的 inode 为 34545，接着查找 inode 为 34545 下面的文件信息。通过对 inode 为 34545 的 MySQL 目录进行扫描，查找到了此目录下所有文件和目录的 inode 信息。根据图 6-5 输出可知，MySQL 目录的 Directory block 为 139777，因此，也可以通过下面的命令查看 MySQL 目录下的 inode 信息。

```
[root@localhost /]# ext3grep  /dev/sda6  --ls --block 139777
```

由于 MySQL 数据文件存放在 data 目录下，因此可以通过 ext3grep 继续查看 inode 为 36577，即 data 目录下的文件信息，过程如图 6-6 所示。

```
[root@localhost /]# ext3grep  /dev/sda6  --ls --inode 36577
Running ext3grep version 0.10.1
Number of groups: 63
Minimum / maximum journal block: 530 / 9271
Loading journal descriptors... sorting... done
The oldest inode block that is still in the journal,
appears to be from 1270697526 = Thu Apr  8 11:32:06 2010
Number of descriptors in journal: 3796; min / max sequence numbers: 2 / 22
Inode is Allocated
Loading vm-disk.ext3grep.stage2.... done
The first block of the directory is 147969.
Inode 36577 is directory "mysql/data".
Directory block 147969:
          .-- File type in dir_entry (r=regular file, d=directory, l=symlink)
          |          .-- D: Deleted ; R: Reallocated
Indx Next |  Inode   | Deletion time                          Mode        File name
=========+=========+----------------data-from-inode------+----------+=========
   0    1 d   36577                                       drwxr-x---  .
   1    2 d   34545                                       drwxr-xr-x  ..
   2    3 d   36578                                       drwx------  cforms
   3    4 d   36589                                       drwxr-xr-x  mysql
   4    5 r   36659                                       rrw-r-----  ibdata1
   5    6 r   36660                                       rrw-r-----  ib_logfile1
   6    7 d   40641                                       drwxr-xr-x  cicro
   7    8 r   36661                                       rrw-r-----  ib_logfile0
   8  end d   42673                                       drwxr-x---  test
   9  end r   36662  D 1270697765 Thu Apr  8 11:36:05 2010  rrw-r-----  localhost.pid
  10  end r   38687  D 1270697574 Thu Apr  8 11:32:54 2010  rrw-rw----  localhost.lower-test
```

图 6-6　查看 Inode 为 36577 目录下的可恢复的数据信息

mysql 表文件存放在 cicro 目录下，因此，可以继续查看 inode 为 40641 的目录信息，过程如图 6-7 所示。

到这里为止，根据 D 标识可以看到被删除的 3 个文件，这 3 个文件正是被删除的表 t_manager 对应的文件。

2. 恢复 mysql 数据文件

接下来，通过 ext3grep 的“--restore-inode”参数恢复这 3 个文件。过程如下：

```
[root@localhost /]# ext3grep  /dev/sda6 --restore-inode 40650
Running ext3grep version 0.10.1
Number of groups: 63
Minimum / maximum journal block: 530 / 9271
Loading journal descriptors... sorting... done
```

```
The oldest inode block that is still in the journal, appears to be from 1270697526
   = Thu Apr  8 11:32:06 2010
Number of descriptors in journal: 3796; min / max sequence numbers: 2 / 22
Restoring inode.40650
[root@localhost /]# ext3grep  /dev/sda6  --restore-inode 40653
Running ext3grep version 0.10.1
Number of groups: 63
Minimum / maximum journal block: 530 / 9271
Loading journal descriptors... sorting... done
The oldest inode block that is still in the journal, appears to be from 1270697526
   = Thu Apr  8 11:32:06 2010
Number of descriptors in journal: 3796; min / max sequence numbers: 2 / 22
Restoring inode.40653
[root@localhost /]# ext3grep  /dev/sda6  --restore-inode 40655
Running ext3grep version 0.10.1
Number of groups: 63
Minimum / maximum journal block: 530 / 9271
Loading journal descriptors... sorting... done
The oldest inode block that is still in the journal, appears to be from 1270697526
   = Thu Apr  8 11:32:06 2010
Number of descriptors in journal: 3796; min / max sequence numbers: 2 / 22
Restoring inode.40655
```

```
[root@localhost /]# ext3grep  /dev/sda6  --ls --inode 40641
Running ext3grep version 0.10.1
Number of groups: 63
Minimum / maximum journal block: 530 / 9271
Loading journal descriptors... sorting... done
The oldest inode block that is still in the journal,
appears to be from 1270697526 = Thu Apr  8 11:32:06 2010
Number of descriptors in journal: 3796; min / max sequence numbers: 2 / 22
Inode is Allocated
Loading vm-disk.ext3grep.stage2.... done
The first block of the directory is 164823.
Inode 40641 is directory "mysql/data/cicro".
Directory block 164823:
          .-- File type in dir_entry (r=regular file, d=directory, l=symlink)
          |          .-- D: Deleted ; R: Reallocated
Indx Next |  Inode   | Deletion time                        Mode        File name
=========+=========+-----------data-from-inode------+----------+=========
   0    1 d   40641                                    drwxr-xr-x  .
   1    2 d   36577                                    drwxr-x---  ..
   2    3 r   40642                                    rrw-r-----  t_organization.frm
   3    4 r   40643                                    rrw-r-----  t_log_opt.frm
   4    5 r   40644                                    rrw-r-----  t_log_opt.MYD
   5    6 r   40645                                    rrw-r-----  t_manager_org.MYI
   6    7 r   40646                                    rrw-r-----  t_usertable.MYI
   7    8 r   40647                                    rrw-r-----  t_organization.MYI
   8    9 r   40648                                    rrw-r-----  t_manager_org.MYD
   9   11 r   40649                                    rrw-r-----  t_usertable.MYD
  10   11 r   40650  D 1270697750 Thu Apr  8 11:35:50 2010  rrw-r-----  t_manager.frm
  11   12 r   40651                                    rrw-r-----  t_log_opt.MYI
  12   14 r   40652                                    rrw-r-----  t_organization.MYD
  13   14 r   40653  D 1270697750 Thu Apr  8 11:35:50 2010  rrw-r-----  t_manager.MYD
  14   16 r   40654                                    rrw-r-----  db.opt
  15   16 r   40655  D 1270697750 Thu Apr  8 11:35:50 2010  rrw-r-----  t_manager.MYI
  16   17 r   40656                                    rrw-r-----  t_manager_org.frm
  17  end r   40657                                    rrw-r-----  t_usertable.frm
```

图 6-7 查看 Inode 为 40641 目录下的可恢复的数据信息

接着，查看文件是否已经恢复到了 RESTORED_FILES 目录下。过程如下：

```
[root@localhost /]# cd RESTORED_FILES/
[root@localhost RESTORED_FILES]# ls
inode.40650  inode.40655 inode.40653
```

可以看到，3 个文件已经恢复了。下面将 3 个文件的名称改为原始名称：

```
[root@localhost RESTORED_FILES]# mv inode.40650  t_manager.frm
[root@localhost RESTORED_FILES]# mv inode.40653  t_manager.MYD
[root@localhost RESTORED_FILES]# mv inode.40655  t_manager.MYI
[root@localhost RESTORED_FILES]# ls
t_manager.frm  t_manager.MYD  t_manager.MYI
```

接着授予这 3 个文件 MySQL 用户和组的权限，然后将文件复制到 MySQL 的数据文件目录下。过程如下：

```
[root@localhost RESTORED_FILES]# chown -R mysql:mysql ./*
[root@localhost RESTORED_FILES]# cp t_manager.* /data/mysql/data/cicro/
```

3. 验证已恢复的 MySQL 表

下面重新启动 MYSQL 数据库，验证被删除的数据表是否已经正确恢复，如图 6-8 所示。

```
[root@localhost RESTORED_FILES]# /data/mysql/mysqld start

[root@localhost RESTORED_FILES]# /data/mysql/bin/mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.30 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use cicro
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from t_manager;
+--------+----------+----------------+---------------+
| MNG_ID | MNG_NAME | MNG_PASSWORD   | MNG_NOTE      |
+--------+----------+----------------+---------------+
|      1 | orga     | =#=mCX1K3glErs= | administrator |
+--------+----------+----------------+---------------+
1 row in set (0.00 sec)

mysql>
```

图 6-8　验证恢复的数据表

可以看到，表 t_manager 已经被完整地恢复了。

6.5　本章小结

本章重点讲述了利用 ext3grep 工具恢复数据文件和 MySQL 数据库的方法。首先模拟了

一个误删除数据文件的环境，然后详细介绍了利用 ext3grep 工具恢复数据文件的方法，接着又通过实例介绍了如何通过 ext3grep 恢复 Mysql 数据库下某个表的具体操作过程。其实恢复表的过程和恢复文件的过程基本一致，这里介绍的恢复表的方法只是提供了一种思路：当表被误删除后，如果没有备份，通过这个方法可以恢复数据以挽回不必要的损失。

作为一名系统管理人员，每天都要和数据打交道，误删除数据也是难免的，但恢复数据不是我们的“本意”，备份数据才是唯一的真理，正所谓：“备份不是万能的，但是没有备份是万万不能的”。

第 3 篇

网络存储应用篇

第 7 章　IP 网络存储 iSCSI

本章主要介绍基于 IP SAN 的网络存储 iSCSI。iSCSI 技术以其低廉的构建成本和优秀的存储性能，博得了很多 CIO 和存储管理员的喜爱，目前陆续进入企业应用领域，推动了企业的存储环境向集中式转变。虽然，目前对于 iSCSI 应该在什么样的环境中使用还存在着诸多争议，但是 iSCSI 的前途是光明的，在未来的存储世界中，iSCSI 一定会占据重要的席位。本章重点介绍 iSCSI 在 Windows 和 Linux 环境下的配置和使用。

7.1　存储的概念与术语

在存储的世界里，有各种各样的名词和术语，常见的有 SCSI、FC、DAS、NAS、SAN 等。本节重点介绍与存储相关的术语和知识。

7.1.1　SCSI 介绍

SCSI 是小型计算机系统接口（Small Computer System Interface）的简称，SCSI 作为输入 / 输出接口，主要用于硬盘、光盘、磁带机、扫描仪、打印机等设备。

7.1.2　FC 介绍

FC 是光纤通道（Fibre Channel）的简称，是一种适合于千兆数据传输的、成熟而安全的解决方案。与传统的 SCSI 技术相比，FC 提供更高的数据传输速率，更远的传输距离，更多的设备连接支持，更稳定的性能，更简易的安装。

7.1.3　DAS 介绍

DAS 是直连式存储（Direct-Attached Storage）的简称，是指将存储设备通过 SCSI 接口或光纤通道直接连接到一台计算机上。当服务器在地理上比较分散，很难通过远程进行互连时，DAS 是比较好的解决方案。但是这种式存储只能通过与之连接的主机进行访问，不能实现数据与其他主机的共享，同时，DAS 会占用服务器操作系统资源，例如 CPU 资源、IO 资源等，并且数据量越大，占用操作系统资源就越严重。

7.1.4　NAS 介绍

网络接入存储（Network-Attached Storage）简称 NAS，它通过网络交换机连接存储系统和服务器，建立专门用于数据存储的私有网络，用户通过 TCP/IP 协议访问数据，采用业界标准的文件共享协议如 NFS、HTTP、CIFS 来实现基于文件级的数据共享。NAS 存储使文件共享访问变得更方便和快捷，并且能很容易地增加存储容量。通过专业化的文件服务器与存储技术相结合，NAS 为那些需要共享大量文件数据的企业提供了一个高效的、高可靠的、高性价比的解决方案。但是 NAS 也有一定的局限性，它会受到网络带宽和网络拥堵的影响，在一定程度上限制了 NAS 的网络传输能力。

7.1.5　SAN 介绍

存储区域网络（Storage Area Network）简称 SAN，它是一种通过光纤交换机、光纤路由器、光纤集线器等设备将磁盘阵列、磁带等存储设备与相关服务器连接起来的高速专用子网。

SAN 由 3 个部分组成，分别是连接设备（如路由器、光纤交换机和 Hub）、接口（如 SCSI、FC）、通信协议（如 IP 和 SCSI）。这 3 个部分再加上存储设备和服务器就构成了一个 SAN 系统。SAN 提供了一个灵活的、高性能的和高扩展性的存储网络环境，它可以更加有效地传输海量的数据块。由于采用了光纤接口，因此 SAN 还具有更高的带宽，同时，SAN 也使统一管理和集中控制实现简化。现在 SAN 已经广泛应用于 ISP 和银行等，随着用户业务量的增大，SAN 的应用前景将越来越光明。

7.2　iSCSI 的概念

iSCSI，即 internet SCSI，是 IETF 制订的一项标准，用于将 SCSI 数据块映射为以太网数据包。从根本上说，它是一种基于 IP Storage 理论的新型存储技术，该技术将存储行业广泛应用的 SCSI 接口技术与 IP 网络技术相结合，可以在 IP 网络上构建 SAN。简单地说，iSCSI 就是在 IP 网络上运行 SCSI 协议的一种网络存储技术。iSCSI 技术最初由 Cisco 和 IBM 两家开发，并且得到了广大 IP 存储技术爱好者的大力支持，这几年得到迅速的发展壮大。

对于中小企业的存储网络来说，iSCSI 是个非常好的选择。首先，从技术实现上来讲，iSCSI 是基于 IP 协议的技术标准，它允许网络在 TCP/IP 协议上传输 SCSI 命令，实现 SCSI 和 TCP/IP 协议的连接，这样用户就可以通过 TCP/IP 网络来构建 SAN，只需要不多的投资，就可以方便、快捷地对信息和数据进行交互式传输和管理。但是，在 iSCSI 出现之前，构建 SAN 的唯一技术是利用光纤通道，这要花费很大的建设成本，一般中小企业无法承担。其次，iSCSI 技术解决了传输效率、存储容量、兼容性、开放性、安全性等方面的诸多问题，在使用性能上绝对不输给商业的存储系统或光纤存储网络。

iSCSI 的优势主要表现为：首先，iSCSI 沿用 TCP/IP 协议，而 TCP/IP 是在网络方面最通用、最成熟的协议，且 IP 网络的基础建设非常完善，同时，SCSI 技术是被磁盘和磁带等设备广泛采用的存储标准，这两点使 iSCSI 的建设费用和维护成本非常低廉；其次，iSCSI 支持一般的以太网交换机而不是特殊的光纤通道交换机，从而减少了异构网络带来的麻烦；还有，iSCSI 是通过 IP 封包传输存储命令，因此可以在整个 Internet 上传输数据，没有距离的限制。

7.3 FC SAN 与 IP SAN

在 iSCSI 技术出现后，通过 IP 技术搭建的存储网络也应运而生，SAN 技术也就出现了两种不同的实现方式，即 FC SAN 与 IP SAN。简单来说，以光纤搭建的存储网络就是 FC SAN，以 iSCSI 技术搭建的存储网络叫做 IP SAN。

作为 SAN 的两种实现方式，FC SAN 与 IP SAN 各有优劣，下面从几个方面分别阐述。

- 在数据传输方式上，FC SAN 与 IP SAN 都采用块协议方式来完成。这是它们的相同点。
- 在传输速度上，就目前的传输速率而言，FC SAN(2Gbit/s) 最快，iSCSI(1Gbit/s) 次之。
- 在传输距离上，FC SAN 理论上可以达到 100 公里，而事实上，传输超过 50 公里后，就会出现瓶颈。而通过 IP 网络的 iSCSI 技术在理论上没有距离的限制，即 iSCSI 可以进行没有距离限制的数据传输。
- 在管理及维护成本上，架设 FC SAN 网络需要投入很多硬件成本，并且需要特定的工具软件进行操作管理，而 IP SAN 构建成本低廉，由于 iSCSI 是通过 IP 网络来传输数据和分配存储资源的，因此只要在现有的网络上进行管理和使用即可，这样就可以省下大笔的管理费用及培训成本。

其实 IP SAN 也面临着一些不可回避的困扰：首先，基于 IP SAN 的网络存储还没有得到用户的充分肯定；其次，IP SAN 存储需要专门的驱动和设备，幸运的是，一些传统的光纤适配器厂商都发布了 iSCSI HBA 设备，同时 Inter 也推出了专用的 IP 存储适配器，而 Microsoft、HP、Novell、SUN、AIX、Linux 也具有 iSCSI Initiator 软件，并且免费供用户使用；还有，在安全方面，IP SAN 虽然有一套规范的安全机制，但是尚未得到用户的认可。

这些问题和困扰虽然会妨碍 iSCSI 的发展，但是相信在未来的网络存储世界里，IP SAN 绝对会拥有一席之地。

7.4 iSCSI 的组成

一个简单的 iSCSI 系统大致由以下部分组成：

- iSCSI Initiator 或者 iSCSI HBA

- iSCSI Target
- 以太网交换机
- 一台或者多台服务器

一个完整的 iSCSI 系统的拓扑结构如图 7-1 所示。

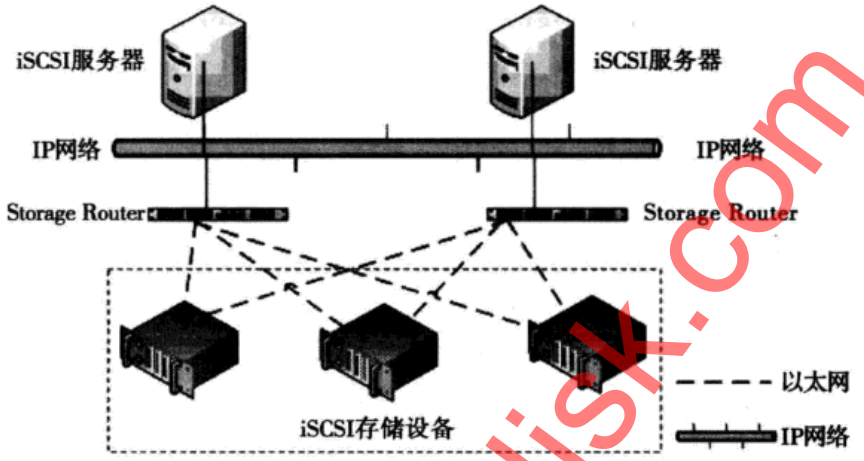

图 7-1　完整的 iSCSI 系统拓扑结构

在图 7-1 中，iSCSI 服务器用来安装 iSCSI 驱动程序，即安装 iSCSI Initiator；Storage Router 可以是以太网交换机或者路由器；iSCSI 存储设备可以是 iSCSI 磁盘阵列，也可以是具有存储功能的 PC 服务器。下面详细介绍一下 iSCSI Initiator 与 iSCSI Target 的含义。

7.4.1　iSCSI Initiator

iSCSI Initiator 是一个安装在计算机上的软件或硬件设备，它负责与 iSCSI 存储设备进行通信。

iSCSI 服务器与 iSCSI 存储设备之间的连接方式有两种：第一种是基于软件的方式，即 iSCSI Initiator 软件。在 iSCSI 服务器上安装 Initiator 后，Initiator 软件可以将以太网卡虚拟为 iSCSI 卡，进而接受和发送 iSCSI 数据报文，从而实现主机和 iSCSI 存储设备之间的 iSCSI 协议和 TCP/IP 协议传输功能。这种方式只需以太网卡和以太网交换机，无需其他设备，因此成本是最低的。但是 iSCSI 报文和 TCP/IP 报文转换需要消耗 iSCSI 服务器的一部分 CPU 资源，只有在低 I/O 和低带宽性能要求的应用环境中才能使用这种方式。

第二种是硬件 iSCSI HBA（Host Bus Adapter）卡方式，即 iSCSI Initiator 硬件。这种方式需要先购买 iSCSI HBA 卡，然后将其安装在 iSCSI 服务器上，从而实现 iSCSI 服务器与交换机之间、iSCSI 服务器与存储设备之间的高效数据传输。与第一种方式相比，硬件 iSCSI HBA 卡方式不需要消耗 iSCSI 服务器的 CPU 资源，同时硬件设备是专用的，所以基于硬件的 iSCSI Initiator 可以提供更好的数据传输和存储性能。但是，iSCSI HBA 卡的价格比较昂贵，因此用户要在性能和成本之间进行权衡。

iSCSI Initiator 软件一般都是免费的，Centos 和 RHEL 对 iSCSI Initiator 的支持都非常不错，现在的 Linux 发行版本都默认自带了 iSCSI Initiator。

7.4.2 iSCSI Target

一个可以用于存储数据的 iSCSI 磁盘阵列或者具有 iSCSI 功能的设备都可以被称为“iSCSI Target”，因为大多数操作系统都可以利用一些软件将系统转变为一个“iSCSI Target”。本章重点讲述如何构建一个 PC 构架的 iSCSI 存储系统。所谓 PC 构架就是选择一个普通的、性能优良的、可支持多块磁盘的 PC（一般为 PC 服务器），再选择一款相对成熟稳定的 iSCSI Target 软件，将 iSCSI Target 软件安装在 PC 服务器上，使普通的 PC 服务器转变成一台 iSCSI 存储设备，并通过 PC 服务器的以太网卡对外提供 iSCSI 数据传输服务。

目前大多数 iSCSI Target 软件都是收费的，例如 DataCore Software 的 SANmelody，FalconStor Software 的 iSCSI Server for Windows 等，这些都是 Windows 平台支持的。不过，也有一些 Linux 平台的开源 iSCSI Target 软件，例如 iSCSI Enterprise Target，后面的内容会重点介绍这个软件。

利用 iSCSI Target 软件，可以将服务器的存储空间分配给客户机使用，客户机可以像使用本地硬盘一样使用 iSCSI 磁盘，包括对其进行分区、格式化及读写等。而且每个客户端都可以向 iSCSI 磁盘写数据，互不干扰，并且不会破坏存储到服务器中的数据。同时，iSCSI Target 软件对用户权限控制非常灵活，支持配置文件。

我们知道，iSCSI 是使用 TCP/IP 协议进行通信的，因此，将 iSCSI 两端连接起来，仅仅需要一个以太网络就可以了。由此可知，iSCSI 的存储性能和这个以太网络有直接关系，所以最好在 iSCSI 网络中使用千兆以太网交换机，劣质的网络设备会严重影响存储系统的性能，也就是说，要为每个服务器配备高质量的千兆以太网交换机，并提供两个连接。对于 iSCSI Target，应该为每个独立阵列中的两个独立端口配备交换机，最后将交换机连接起来，采用这种配置方式，即使两个交换机中的一个出现了故障，整个 iSCSI 存储系统仍然能够正常工作，这保证了存储系统的不间断运行。

7.5 iSCSI 的工作原理

要理解 iSCSI 的工作原理，就必须知道 iSCSI 的层次结构。根据 OSI 模型，iSCSI 的协议自顶向下一共可以分为三层，如图 7-2 所示。

下面对每个分层进行简单介绍。

- SCSI 层：根据客户端发出的请求建立 SCSI CDB（命令描述块），并传给 iSCSI 层。同时接收来自 iSCSI 层的 CDB，并向应用返回数据。

- iSCSI 层：对 SCSI CDB 进行封装，以便能够在基于 TCP/IP 协议的网络上进行传输，完成 SCSI 到 TCP/IP 的协议映射。这一层是 iSCSI 协议的核心层。本章也主要针对这一层的配置和管理进行介绍。
- TCP/IP 层：对 IP 报文进行路由和转发，并且提供端到端的透明可靠的传输。

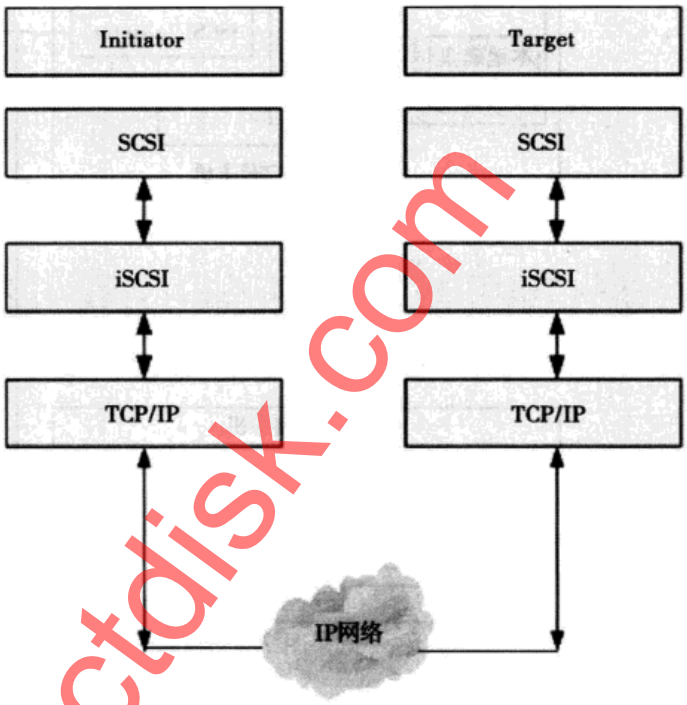

图 7-2　iSCSI 的协议结构

iSCSI 协议定义了在 TCP/IP 网络发送、接收数据块存储数据的规则和方式。先发送端将 SCSI 命令和数据封装到 TCP/IP 包中，然后通过 IP 网络转发，接收端收到 TCP/IP 包之后，将其还原为 SCSI 命令和数据并执行，执行完成后，将返回的 SCSI 命令和数据再封装到 TCP/IP 包中，之后再传回发送端。这样就完成了数据传输的整个过程。

iSCSI 的整个数据传输过程在用户看来是完全透明的，用户使用远端的存储设备就像使用本地的硬盘设备一样。不过，这只是理论状态，实际上 iSCSI 的数据传输速率并不能完全达到本地硬盘的数据传输速率，但差别并不明显。而且这种网络存储模式还有一个优点是安全性高，这对于数据集中存储的 iSCSI 来说显然非常重要。

7.6　搭建基于 IP SAN 的 iSCSI 存储系统

在本节中，重点介绍如何构建一个 PC 构架的 iSCSI 存储系统。这里们选择一个普通的、性能优良的、可支持多块磁盘的 PC 服务器作为 iSCSITarget，并且选择一个成熟稳定的 iSCSI Target 软件 iscsitarget。基本配置环境如表 7-1 所示。

表 7-1　iSCSI 存储系统示例的基本配置环境

名称	操作系统	IP 地址	安装软件
主机	CentOS release 5.3	192.168.12.246	iSCSI Enterprise Target
Initiator 主机	CentOS release 5.3	192.168.12.26	iscsi-initiator-utils
Initiator 主机	Windows XP	192.168.12.136	Initiator-2.08

这里将主机的第三块硬盘（硬盘标识为 /dev/sdc）作为 iSCSI 共享磁盘，硬盘大小为

7.6.2　配置一个简单的 iSCSI Target

iSCSI Target 的主配置文件为 /etc/iet/ietd.conf，此文件中的选项默认全部被注释掉，需要用到这些选项时将注释去掉即可。

打开 ietd.conf 文件，首先找到类似如下的代码：

```
#Target iqn.2001-04.com.example:storage.disk2.sys1.xyz
```

先将前面的“#”号去掉，此选项表示该 iSCSI Target 的名称。Target 的名称在同一子网内应该是唯一的，标准命名方式如下：

```
iqn.yyyy-mm.<reversed domain name>[:identifier]
```

其中参数含义如下：

- iqn 表示“iSCSI Qualified Name”，简称 iqn。
- yyyy-mm 表示“年份 - 月份”。这里是 2001-04。
- reversed domain name 表示倒过来的域名，这里是 com.example。
- identifier 表示识别代码，这里是 storage.disk2.sys1.xyz。

接下来就要设定 LUN（Logical Unit Number，逻辑单元号），找到类似如下的代码：

```
#Lun 0 Path=/dev/sdb,Type=fileio,ScsiId=xyz,ScsiSN=xyz
```

将前面的“#”号去掉，“Lun 0 Path=/dev/sdb”表示块设备号为 0，映射的磁盘为 /dev/sdb。“Type”值（fileio 默认值是 fileio）表示 iSCSI Target 可以用于磁盘、file 和 LVM，这里设定的是“fileio”，表示 iSCSI Target 主要用来对一个磁盘进行存储共享。读者可以根据自己的情况将 Path 改为需要共享的存储分区的设备标识，这里假定共享的设备标识为 /dev/sdb。

至此，一个简单的 iSCSI Target 已经配置完毕，最后启动 iscsi-target 服务。

```
[root@iscsi-target iscsi]# service iscsi-target start
```

7.6.3　在 Windows 上配置 iSCSI Initiator

下面的操作是在 Initiator 的 Windows 主机即 IP 为 192.168.12.136 主机上进行的。

微软对 iSCSI initiator 的支持相当完备，读者可以免费从微软网站上下载 iSCSI initiator 软件，网址是 http://www.microsoft.com/WindowsServer2003/technologies/storage/iscsi/default.mspx，本章示例所采用的版本是 Initiator-2.08-build3825-x86fre.exe。接下来开始说明如何让 Windows 连接 iSCSI Target。

安装完成 iSCSI Initiator 后，在桌面上会发现启动图标。启动 Microsoft iSCSI Initiator 后，选择第二个标签“Discovery”，然后在“Target Portals”部分单击“Add”按钮，弹出“Add Target Portal”对话框，如图 7-4 所示。

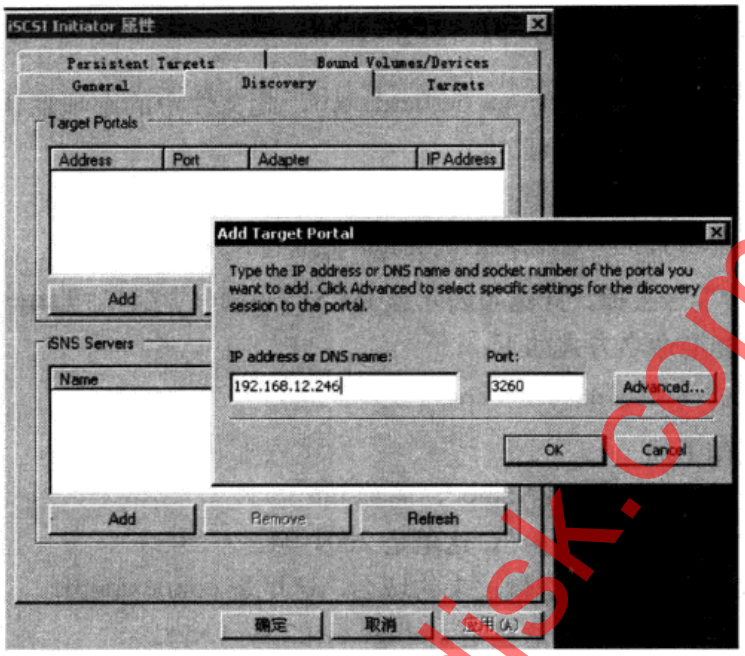

图 7-4 添加一个 iSCSI Target

在此对话框中填写 iSCSI Target 的 IP 地址为上面设定的 Target 主机的地址，如果没有特殊设定，那么 iSCSI Target 的端口默认是 3260。填写完成后单击“OK”按钮。

接下来，选择第三个标签“Targets”，如图 7-5 所示。可以看到，iSCSI Initiator 已经检测到了 iSCSI Target 的名称，但是，此时的 iSCSI Target 还处于“inactive”状态，单击下方的“Log On”按钮后弹出“Log On to Target”对话框，接着单击“OK”按钮来激活 Target。此时 iSCSI Target 已经从“inactive”状态变为“Connected”状态，如图 7-6 和图 7-7 所示。

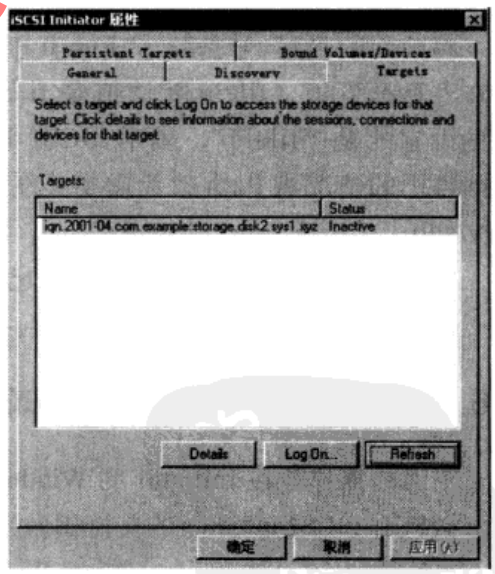

图 7-5 添加 iSCSI Target 后的状态

到此为止，Windows 系统已经识别了 iSCSI Target 提供的共享磁盘分区，通过 Windows 的磁盘管理器可以查看到新增加的磁盘分区，如图 7-8 所示。

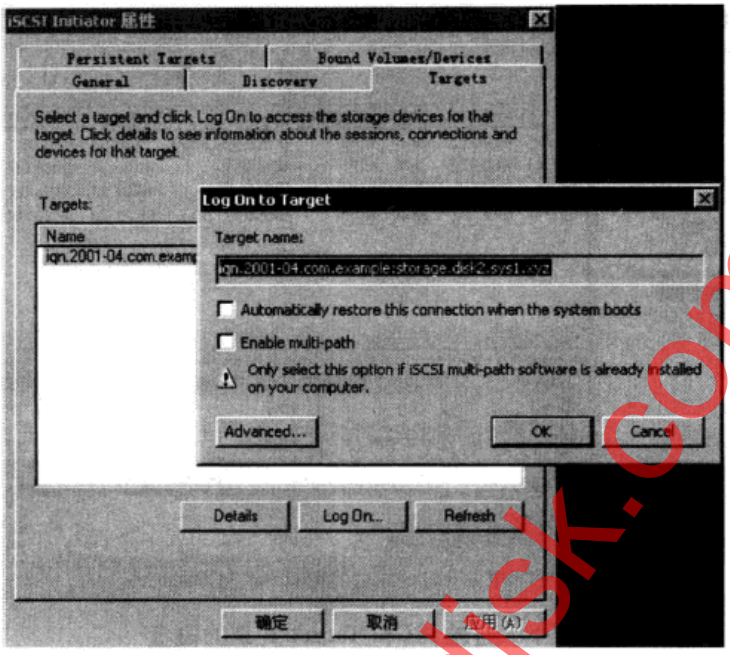

图 7-6　激活 iSCSI Target

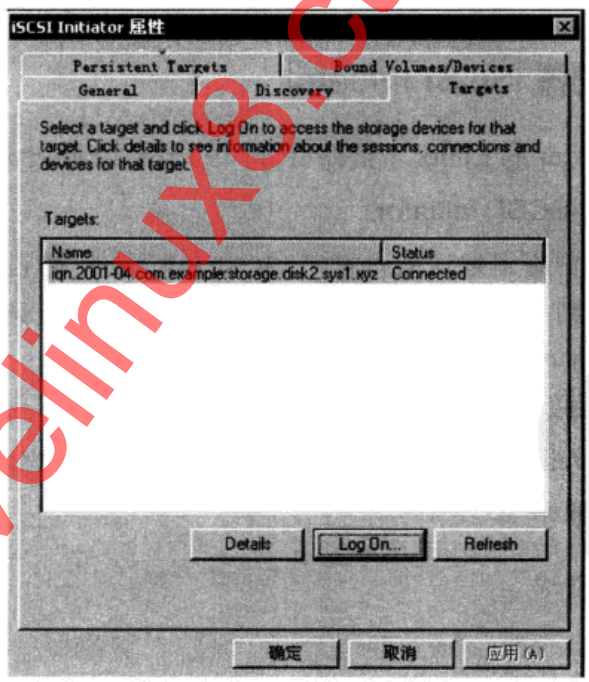

图 7-7　激活后的 iSCSI Target 状态

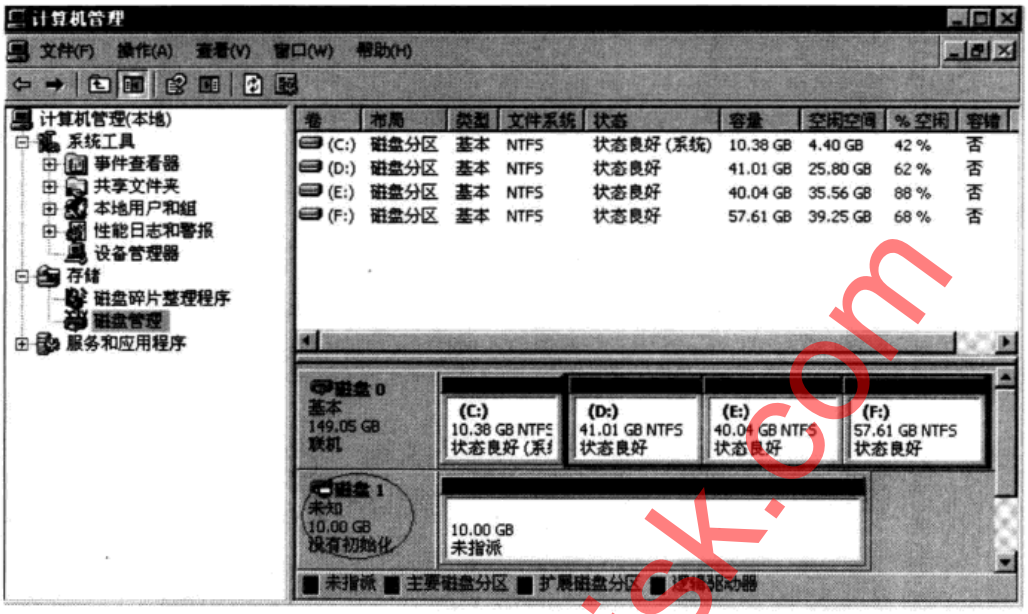

图 7-8 通过 iSCSI Target 共享的磁盘

现在就可以使用 Windows 的磁盘管理功能对这块共享磁盘进行分区、格式化及挂载等操作了。

7.6.4 在 Linux 上配置 iSCSI Initiator

下面的操作是在 Initiator 的 Linux 主机即 IP 为 192.168.12.26 主机上进行的。

1. 安装 Linux 版本的 iSCSI Initiator

现在的主流 Linux 发行版本默认都自带了 iSCSI Initiator，即 Open-iSCSI，如果系统没有安装，只需通过光盘找到 iscsi-initiator-utils-6.2.0.871-0.16.el5.i386.rpm 包，以 rpm 方式安装即可。当然也可以通过 yum 方式进行自动安装，操作如下：

```
[root@ Initiator iscsi]#yum install iscsi*
```

安装完成后，会生成 /etc/iscsi 主程序配置目录。其他相关文件的安装位置如下：

```
/etc/iscsi/iscsid.conf
/etc/rc.d/init.d/iscsi
/etc/rc.d/init.d/iscsid
/sbin/iscsi-iname
/sbin/iscsiadm
/sbin/iscsid
/sbin/iscsistart
/var/lib/iscsi
/var/lib/iscsi/ifaces
/var/lib/iscsi/isns
/var/lib/iscsi/nodes
```

```
/var/lib/iscsi/send_targets
/var/lib/iscsi/slp
/var/lib/iscsi/static
/var/lock/iscsi
```

接下来需要启动 Initiator 服务，操作如下：

```
[root@ Initiator iscsi]# service iscsi start
```

2.iSCSI Initiator 目录的功能介绍

- /sbin/iscsiadm 命令。在安装完 iSCSI Initiator 后，会生成 /sbin/iscsiadm 命令，此命令是用来管理（更新、删除、插入、查询）iSCSI 配置数据库文件的命令行工具，用户能够用它对 iSCSI nodes、sessions、connections 和 discovery records 进行一系列的操作。
- /var/lib/iscsi/send_targets 目录。在此目录下，会生成一个或多个以 iSCSI 存储服务器的 IP 地址和端口号命名的文件夹，文件名为“iSCSI Target IP，端口号”（例如“192.168.12.246,3260”）。
- /var/lib/iscsi/nodes 目录。在此目录下，会生成一个或多个以 iSCSI 存储服务器上的 Target 名命名的文件夹，在该文件夹下有一个文件名为“iSCSI portal IP，端口号”（例如“192.168.12.246,3260”）的配置参数文件，该文件是 iSCSI Initiator 登录 iSCSI Target 时要使用的参数，而这些参数的设置是从 /etc/iscsi/iscsi.conf 中的参数继承而来的，可以通过 iscsiadm 命令对某一个参数文件进行更改。

3. 在 Linux 上执行 iSCSI Target 发现

可以使用如下指令查询 iSCSI Target 主机划分了哪些 lun。

```
iscsiadm -m discovery --type sendtargets --portal IP
```

或者

```
iscsiadm -m discovery -t sendtargets -p IP
```

例如：

```
[root@ Initiator iscsi ]# iscsiadm -m discovery -t sendtargets -p 192.168.12.246:3260
192.168.12.246:3260,1 iqn.2001-04.com.example:storage.disk2.sys1.xyz
```

可以看出，“iqn.2001-04.com.example:storage.disk2.sys1.xyz”就是 iSCSI Target 的名称。由于在配置 iSCSI Target 时，没有做任何限制，因此允许所有的客户端主机连接 iSCSI Target 共享的磁盘。

需要说明的是，当成功执行一次 Target 发现后，iSCSI Initiator 就会将查询记录写到 /var/lib/iscsi/send_targets 对应的目录下。因此，Target 发现只执行一次即可。

接着通过 iscsiadm 指令与 iSCSI Target 主机建立连接，也就是登录到 iSCSI Target。具体指令如下：

```
iscsiadm -m node -T <target-name> -p <ip-address>:<port> --login
```

或者

```
iscsiadm -m node -T [target-name] -p [ip-address] -l
```

这里的“-T”后面跟的是 Target 名称，“ip-address”是 Target 主机的 IP 地址，“port”是 Target 主机的端口号，默认是 3260。

例如：

```
[root@ Initiator iscsi ]#iscsiadm -m node -T iqn.2001-04.com.example:storage.
    disk2.sys1.xyz -p 192.168.12.246 -l
Logging in to [iface: default, target: iqn.2001-04.com.example:storage.disk2.sys1.
    xyz, portal: 192.168.12.246,3260]
Login to [iface: default, target: iqn.2001-04.com.example:storage.disk2.sys1.xyz,
    portal: 192.168.12.246,3260]: successful
```

如果有多个 Target 主机时，可以通过如下命令一次登录到所有的 Target 主机：

```
[root@ Initiator iscsi ]#iscsiadm -m node --loginall=all
```

这里需要说明的是，执行 Target 发现操作，其实已经与 iSCSI Target 主机建立了连接，此时如果再次执行 iscsiadm 命令与 Target 主机建立连接，会提示“iscsiadm: initiator reported error (15 - already exists)”错误，所以需要先断开与 iSCSI Target 主机的连接。执行如下指令可断开 Initiator 与 iSCSI Target 主机的连接。

```
iscsiadm -m node -T [target-name] -p [ip-address] -u
```

例如：

```
[root@ Initiator iscsi ]#iscsiadm -m node -T iqn.2001-04.com.example:storage.
    disk2.sys1.xyz -p 192.168.12.246 -u
Logging out of session [sid: 2, target: iqn.2001-04.com.example:storage.disk2.
    sys1.xyz, portal: 192.168.12.246,3260]
Logout of [sid: 2, target: iqn.2001-04.com.example:storage.disk2.sys1.xyz, portal:
    192.168.12.246,3260]: successful
```

当 iSCSI Initiator 与 iSCSI Target 连接成功后，还可以通过如下命令查看 iSCSI session 信息。

```
[root@ Initiator iscsi ]#iscsiadm -m session -i
```

例如：

```
[root@ Initiator iscsi ]#iscsiadm -m session -i
iSCSI Transport Class version 2.0-871
version 2.0-871
Target: iqn.2001-04.com.example:storage.disk2.sys1.xyz
   Current Portal: 192.168.12.246:3260,1
   Persistent Portal: 192.168.12.246:3260,1
           **********
           Interface:
           **********
           Iface Name: default
```

```
Iface Transport: tcp
Iface Initiatorname: iqn.1994-05.com.redhat:fd37f211e3a
Iface IPaddress: 192.168.12.26
Iface HWaddress: <empty>
Iface Netdev: <empty>
SID: 1
iSCSI Connection State: LOGGED IN
iSCSI Session State: LOGGED_IN
Internal iscsid Session State: NO CHANGE
************************
Negotiated iSCSI params:
************************
HeaderDigest: None
DataDigest: None
MaxRecvDataSegmentLength: 262144
MaxXmitDataSegmentLength: 8192
FirstBurstLength: 65536
MaxBurstLength: 262144
ImmediateData: Yes
InitialR2T: Yes
MaxOutstandingR2T: 1
************************
Attached SCSI devices:
************************
Host Number: 32 State: running
scsi32 Channel 00 Id 0 Lun: 0
        Attached scsi disk sdb          State: running
```

4. 管理共享磁盘

可以通过 fdisk 命令查看共享的磁盘标识，也可以通过 dmesg 命令查看系统是否识别到了共享的 iSCSI 磁盘。操作如下：

```
[root@ Initiator iscsi ]#fdisk -l
Disk /dev/sda: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14       38913   312464250   8e  Linux LVM

Disk /dev/sdb: 10.7 GB, 10737418240 bytes
255 heads, 63 sectors/track, 1305 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1         609     4891761   83  Linux
/dev/sdb2             610        1305     5590620   83  Linux
```

从 fdisk 输出可知，iSCSI 共享磁盘标识为 /dev/sdb，大小为 10.7GB。接下来就可以通

过 fdisk 命令对这个磁盘进行重新分区、格式化、创建文件系统等操作了。

7.7 iSCSI 在安全方面的相关设定

在上一节中介绍了如何搭建一个简单的 iSCSI 网络存储系统。iSCSI Initiator 的客户端主机可以任意连接和使用 iSCSI Target 共享的所有磁盘和分区，而在很多时候，通过授权认证连接共享磁盘或分区是必须的，例如，只允许客户端主机 A 连接 Target 共享出来的磁盘分区一，而客户端主机 B 只允许连接 Target 共享的磁盘分区二等。在这种情况下，就需要在 iSCSI target 主机上进行授权设定。

iSCSI 在授权访问和安全管理方面很有优势，它能够以主机为基础，也就是以 IP 地址为基础来设定允许或拒绝存取；也可以通过账户密码认证来设定允许或拒绝存取。

下面通过一个应用实例来介绍 iSCSI 授权获取磁盘资源的方法。

一个 PC 构架的 iSCSI Target 服务器，共享的硬盘标识为 /dev/sdc，大小为 10GB。将此硬盘划分了两个分区 /dev/sdc1 和 /dev/sdc2，将 /dev/sdc1 共享给一个 IP 地址为 192.168.12.136 的 Windows 客户端主机，将 /dev/sdc2 共享给一个 IP 地址为 192.168.12.26 的 Linux 客户端主机，iSCSI Target 服务器的 IP 地址为 192.168.12.246。接下来通过 IP 认证和账户密码认证两种方式来介绍如何实现这种需求。

7.7.1 Initiator 主机以 IP 认证方式获取 iSCSI Target 资源

此种方式配置非常简单，只需在 iSCSI Target 服务器上修改两个文件即可。先在 iscsitarget 主目录 /etc/iet 下找到 ietd.conf 文件，然后添加如下内容：

```
Target iqn.2000-04.net.ixdba:sdc1
Lun 0 Path=/dev/sdc1,Type=fileio
Target iqn.2002-04.net.ixdba:sdc2
Lun 0 Path=/dev/sdc2,Type=fileio
```

在 ietd.conf 文件中，定义了两个 Target，为每个 Target 分别添加了对应的磁盘分区。接着修改 /etc/iet/initiators.allow 文件，这个文件定义了 Initiator 主机对 Target 服务器的访问规则，作用类似于 Linux 操作系统中的 /etc/hosts.allow 文件。修改后的 initiators.allow 文件内容如下：

```
iqn.2000-04.net.ixdba:sdc1 192.168.12.136
iqn.2002-04.net.ixdba:sdc2 192.168.12.26
```

修改完成，重启 iscsi-target 服务，操作如下：

```
[root@iscsi-target iet]# service iscsi-target restart
Stopping iSCSI Target:                                     [  OK  ]
Starting iSCSI Target:                                     [  OK  ]
```

接着，在 IP 地址为 192.168.12.26 的 Linux Initiator 主机上执行如下操作：

```
[root@ Initiator iscsi]# /etc/init.d/iscsi restart
[root@ Initiator iscsi]#iscsiadm -m discovery -t sendtargets -p 192.168.12.246
192.168.12.246:3260,1 iqn.2002-04.net.ixdba:sdc2
[root@ Initiator iscsi]#fdisk -l
Disk /dev/sda: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14       38913   312464250   8e  Linux LVM

Disk /dev/sdb: 5724 MB, 5724794880 bytes
177 heads, 62 sectors/track, 1018 cylinders
Units = cylinders of 10974 * 512 = 5618688 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        1018     5585735   83  Linux
```

通过重启 iscsi-target 服务，重新执行 Target 发现，Linux 系统已经识别了 Target 共享的磁盘分区，其中“/dev/sdb: 5724 MB”就是 iSCSI 共享磁盘。接下来就可以在 Linux 上管理和使用这个共享磁盘了。

最后，登录 windows 系统，打开 Microsoft iSCSI Initiator，添加 iSCSI 共享磁盘即可。

7.7.2　Initiator 主机以密码认证方式获取 iSCSI Target 资源

iSCSI Target 以账号密码方式认证分为两阶段：

第一阶段是 discovery 查询认证所使用的账号和密码（即 SendTargets 用的）。

第二阶段是登录 Target / iqn / Lun 时所使用的账号密码（即 Login 登录时用的）。

此种方式在配置方面稍微复杂一些，需要在 Initiator 主机和 iSCSI Target 服务器上进行简单配置。下面分步介绍。

1. 配置 iSCSI Target

首先修改 /etc/iet/initiators.allow 文件，打开所有权限。修改后的内容如下：

```
#iqn.2000-04.net.ixdba:sdc1 192.168.12.136
#iqn.2002-04.net.ixdba:sdc2 192.168.12.26
ALL ALL
```

接着修改 /etc/iet/ietd.conf 文件。修改后的内容如下：

```
IncomingUser  discovery.auth  discoverysecret

Target iqn.2000-04.net.ixdba:sdc1
IncomingUser  login.windows.auth  windowssecret
```

```
Lun 0 Path=/dev/sdc1,Type=fileio

Target iqn.2002-04.net.ixdba:sdc2
IncomingUser  login.linux.auth linuxsecret
Lun 0 Path=/dev/sdc2,Type=fileio
```

其中，第一个“IncomingUser”是个全局参数，用来指定discovery查询认证所使用的账号和密码，必须与Initiator主机中设定的用户名和密码一致。第二个和第三个“IncomingUser”选项包含在对应的Target中，用来指定Windows和Linux客户端主机登录Target/iqn/Lun时所使用的账号密码，也必须与initiator主机中设定的用户名和密码一致。

所有配置完毕以后，重启iscsi-target服务。

2. 配置Linux Initiator主机

修改/etc/iscsi/iscsid.conf文件，添加如下选项：

```
# 以下三个是针对 login 的
node.session.auth.authmethod = CHAP
          # 表示在登录时启用 CHAP 验证
node.session.auth.username = login.linux.auth
          # 验证用户名称，可以是任意字符，但必须与 Target 端 IncomingUser 配置的名字一致
node.session.auth.password = linuxsecret
          # 验证密码，必须与 Target 端对应的 IncomingUser 选项设置的密码一致

# 以下三个是针对 discovery 的
discovery.sendtargets.auth.authmethod = CHAP
          # 表示 discovery 时启用 CHAP 验证
discovery.sendtargets.auth.username = discovery.auth
          # 验证用户名称，可以是任意字符，但必须与 Target 端 IncomingUser 配置的名称一致
discovery.sendtargets.auth.password = discoverysecret
          # 验证密码，必须与 Target 端对应的 IncomingUser 选项设置的密码一致
```

配置完毕后，重启Initiator，重新执行discovery查询，操作如下：

```
[root@ Initiator iscsi ]#/etc/init.d/iscsi restart
[root@ Initiator iscsi ]# iscsiadm -m discovery -t sendtargets -p 192.168.12.246
192.168.12.246:3260,1 iqn.2000-04.net.ixdba:sdc1
192.168.12.246:3260,1 iqn.2002-04.net.ixdba:sdc2
```

从查询结果可知，Initiator查询到了两个Target，最后执行fdisk操作。过程如下：

```
[root@ Initiator iscsi ]# fdisk -l
Disk /dev/sda: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
```

```
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14       38913   312464250   8e  Linux LVM

Disk /dev/sdb: 5724 MB, 5724794880 bytes
177 heads, 62 sectors/track, 1018 cylinders
Units = cylinders of 10974 * 512 = 5618688 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        1018     5585735   83  Linux
```

从 fdisk 的输出结果可知，Linux Initiator 已经成功地连接了 iSCSI 共享磁盘，而“/dev/sdb: 5724 MB”就是识别的硬盘标识和大小。

3. 配置 Windows Initiator 主机

配置 Windows Initiator 主机的方法在前面已经介绍过，这里只讲述不同的地方。先打开 Microsoft iSCSI Initiator，选择第二个标签“Discovery”，然后在“Target Portals”部分单击“Add”按钮，弹出“Add Target Portal”对话框，在此对话框中填写 iSCSI Target 的 IP 地址和端口，填写完毕后单击“Advanced”按钮，如图 7-9 所示。

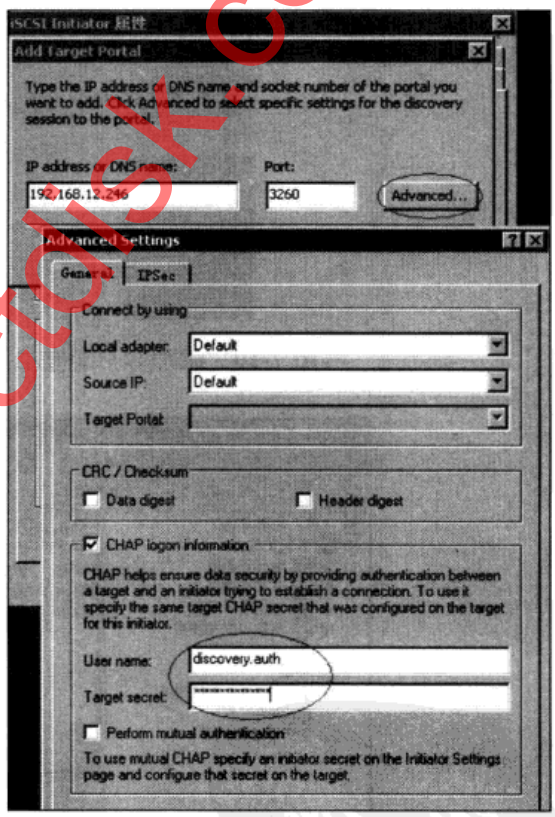

图 7-9　添加 Discovery 查询认证的账号密码

在此对话框中，选中“CHAP logon information”复选框，然后填写 discovery 查询认证所使用的账号和密码。填写完毕后单击“确定”按钮。

接着选择第三个标签“Targets”，此时 Initiator 已经从 iSCSI Target 端查询到了两个 Target，选中第一个名为“Target iqn.2000-04.net.ixdba:sdc1”的 Target，单击“Log On”按钮，然后在弹出的“Log On to Target”对话框中单击“Advanced”按钮，如图 7-10 所示。

在此对话框中，选中“CHAP logon information”复选框，然后填写客户端登录 iSCSI Target / iqn / Lun 时所使用的账号密码。填写完毕单击“确定”按钮。

此时，名为“Target iqn.2000-04.net.ixdba:sdc1 ”的 Target 已经处于“Connected”状态，即 Microsoft iSCSI Initiator 已经连接上了 iSCSI Target 服务器共享的磁盘分区。最后，查看 Windows 磁盘管理器，可以看到共享硬盘分区，如图 7-11 所示。

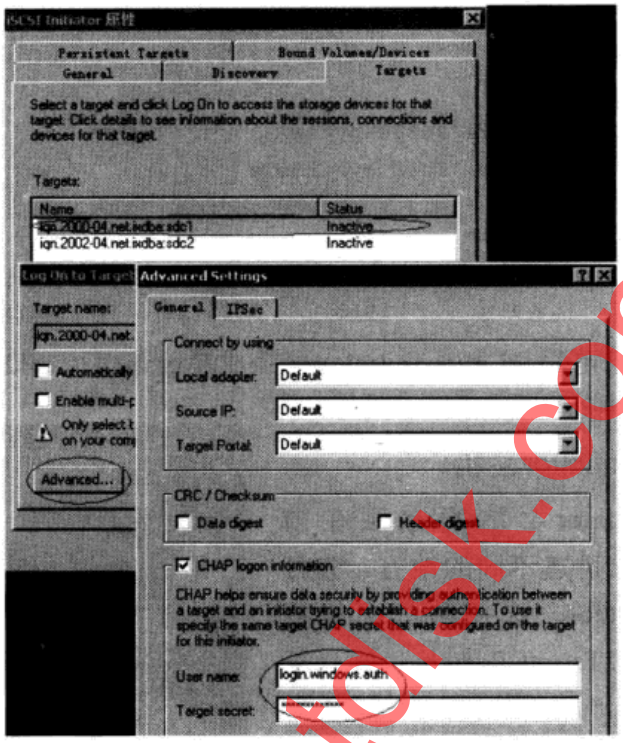

图 7-10 添加客户端登录 iSCSI Target 时所使用的账号和密码

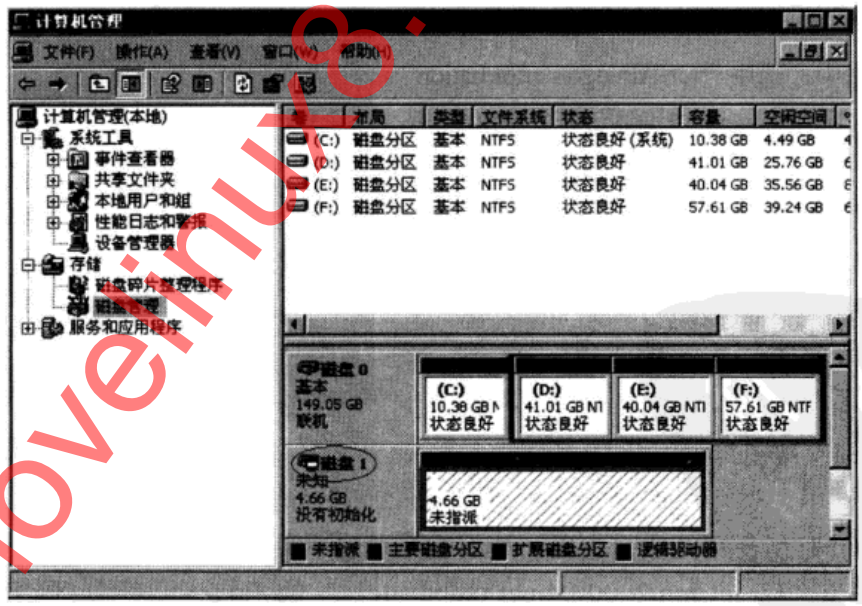

图 7-11 通过 iSCSI 共享给 Windows 的磁盘

到这里为止，Windows 可以对这个 iSCSI 磁盘进行分区、格式化等操作了。

7.8　iSCSI 性能优化方案

7.8.1　iSCSI 性能瓶颈

iSCSI 协议建立在传统的 TCP/IP 协议之上，在进行实际数据传输时，其传输系统性能受限于 TCP/IP 协议栈负载及以太网最大带宽，另外 iSCSI 协议层也会额外增加一些负载开销，在实际应用中，iSCSI 数据传输性能仍然存在瓶颈。以发送端的写操作为例，一次完整的 iSCSI 协议数据传输包括数据封装、数据拷贝、数据传输三个步骤。

1. 数据封装

iSCSI 协议层首先将接收到的命令封装成 iSCSI 协议层的 PDU 数据包，此过程还需要对 PDU 数据包进行 CRC 校验计算，以便接收端接收到数据后进行正确性检验。数据封装过程还必须进行 CRC 循环冗余校验计算，由于校验生成与检测属于高密度计算型操作，需要消耗大量 CPU 资源，因此会严重影响 iSCSI 系统的性能。

2. 数据拷贝

封装完毕后的 PDU 数据包以数据拷贝的方式传递给操作系统的 TCP/IP 协议层缓冲区，再由该缓冲区发送给数据链路层进行网络传输，一个 PDU 数据包可能对应多个 TCP 连接。数据拷贝过程也会占用发送端或接收端的大量 CPU 资源及内存带宽，从而影响系统性能。

3. 数据传输

TCP/IP 层接收到数据包后通过 PCI DMA 的方式将数据传递给网卡，经物理链路传输到目标服务器。读操作过程的操作顺序正好相反，目标服务器端操作也与启动服务器端类似。数据传输过程是利用 TCP/IP 协议进行网络传输的，必然受限于 TCP/IP 协议栈本身的开销负载以及实际的网络带宽，因此进行数据传输也会占用系统 CPU 资源。

7.8.2　iSCSI 性能优化

通过前面的介绍，已经了解了影响 iSCSI 性能的瓶颈，因此如何优化现有的 iSCSI 系统的问题就变得显而易见了。下面根据 iSCSI 协议在数据传输、数据拷贝、数据校验三个方面存在的性能瓶颈问题，分别介绍对 iSCSI 网络存储进行性能优化的几种方法。

1. 优化网络传输

iSCSI 协议利用通用的 TCP/IP 协议进行海量存储数据传输，传统的 TCP/IP 协议用于在不可靠的数据链路上实现可靠的数据传输，其性能必然受限于 TCP/IP 本身的传输性能，基于通用 TCP/IP 协议的网络存储面临的一个很重要的性能问题就是 TCP/IP 协议栈开销。目前已经出现了一些新的网络技术用于改善 TCP/IP 协议栈开销。

1）有些新型网卡可以分担过去由主机 CPU 完成的 CRC 校验和计算功能，利用这些网卡，可进一步释放主机 CPU 资源，进而降低 CPU 利用率。

2）一些新型网卡可以实现过去由主机 TCP/IP 软件协议栈完成的 TCP 包拆分功能，这样也可缓解主机 CPU 的压力。

3）iSCSI SAN 网络绝不能与一般的以太网用户混合。如果混合不仅会削弱 SAN 的性能，还会使 LAN 的存储数据受到影响。正确的做法是将 iSCSI SAN 网络与日常的用户网络分开。最常见的分离方法是采用虚拟局域网（VLAN），限制 iSCSI 网络通向虚拟局域网，保持正常的网络通道。

2. 避免数据拷贝

iSCSI 协议层建立在传统的 TCP/IP 协议层之上，其 PDU 数据包必须通过内存拷贝的方式传递给 TCP/IP 协议栈的缓冲区进行网络传输。而内存拷贝要进行内存读写操作各一次，内存数据拷贝又必须依赖于 CPU 执行指令来完成，因此会消耗 CPU 资源并降低内存性能，最终影响 iSCSI 系统性能。为了避免数据拷贝操作，目前主要的方法是采用 TOE 硬件方案：TOE 方案是将过去由操作系统完成的 TCP/IP 协议栈直接转移到 TOE 网卡上由硬件实现。这个方案有效地避免了 TCP/IP 协议栈对系统 CPU 资源的占用，优化了 iSCSI 系统性能。TOE 卡可以从一些供应商处获得，如 Alacritech、LeWiz Communications、QLogic 等。

3. 数据校验的优化

CRC 校验数据的生成或检测均是计算密集型操作，会消耗大量的 CPU 资源，因此如何优化数据校验性能成为优化 iSCSI 性能的关键。目前常用的优化方案有下面几种。

（1）iSCSI HBA 硬件方案

为了进一步消除 CRC 校验运算对系统 CPU 资源的影响，工业界将 iSCSI 协议及 TCP/IP 协议全部集成到网卡上构成 iSCSI 的 HBA 适配卡，这种方式中的 iSCSI HBA 适配卡对主机来说就是一块 SCSI 卡，既不存在数据拷贝问题，也不存在数据校验问题。采用硬件方案虽然能进一步释放 CPU 资源，但 iSCSI HBA 硬件处理能力有限制，无法分享 CPU 日益提升的性能。

（2）优化数据校验

通过优化 CRC 校验算法来加快校验过程也是提高 iSCSI 性能的一个途径。

（3）直接关闭校验

CRC 校验主要保证 PDU 数据在传输中的安全性和可靠性，避免数据被篡改。如果不强调安全性，CRC 校验意义就不大，所以在一些对性能要求很高而对数据安全性要求不高的场合，可以通过关闭 CRC 校验的方法来优化 iSCSI 性能，而数据的正确性可以依靠 TCP/IP 层的 CRC 校验数据来保证。

7.9　本章小结

本章主要讲述了网络存储 iSCSI 技术的概念、组成、安装、配置和使用，还介绍了 iSCSI 在安全方面的设置方法。通过本章的学习，读者能够对 iSCSI 有一个基本的了解和认识，同时也能够迅速地搭建出一套 iSCSI 存储系统。

利用 iSCSI 技术构建的基于 IP 的存储网络系统，无需昂贵的专用存储网络，易于管理维护，并可以有效减少用户投资，节约成本。同时，iSCSI 还具有良好的扩展性，方便进行异地容灾。而对 iSCSI 存储系统的优化，主要集中在对 iSCSI 数据校验、数据拷贝、数据传输三个方面进行改善。随着 iSCSI 技术的日益成熟，采用 iSCSI 技术构建的 IP 存储网络系统将拥有广阔的应用前景。

第 8 章　分布式存储系统 MFS

本章讲述的是 Linux 下的开源存储系统 MFS，它是由波兰人开发的。MFS 文件系统能够实现 RAID 的功能，不但能够更节约存储成本，而且不逊色于专业的存储系统，更重要的是它能够实现在线扩展。读者必须明白的一点是，MFS 是一种半分布式文件系统。

8.1　MFS 概论

MFS 的官方网站为 http://www.moosefs.org/，在这里可以获取更为详细的帮助。官方 MFS 的网络组成及运行原理如图 8-1 所示。

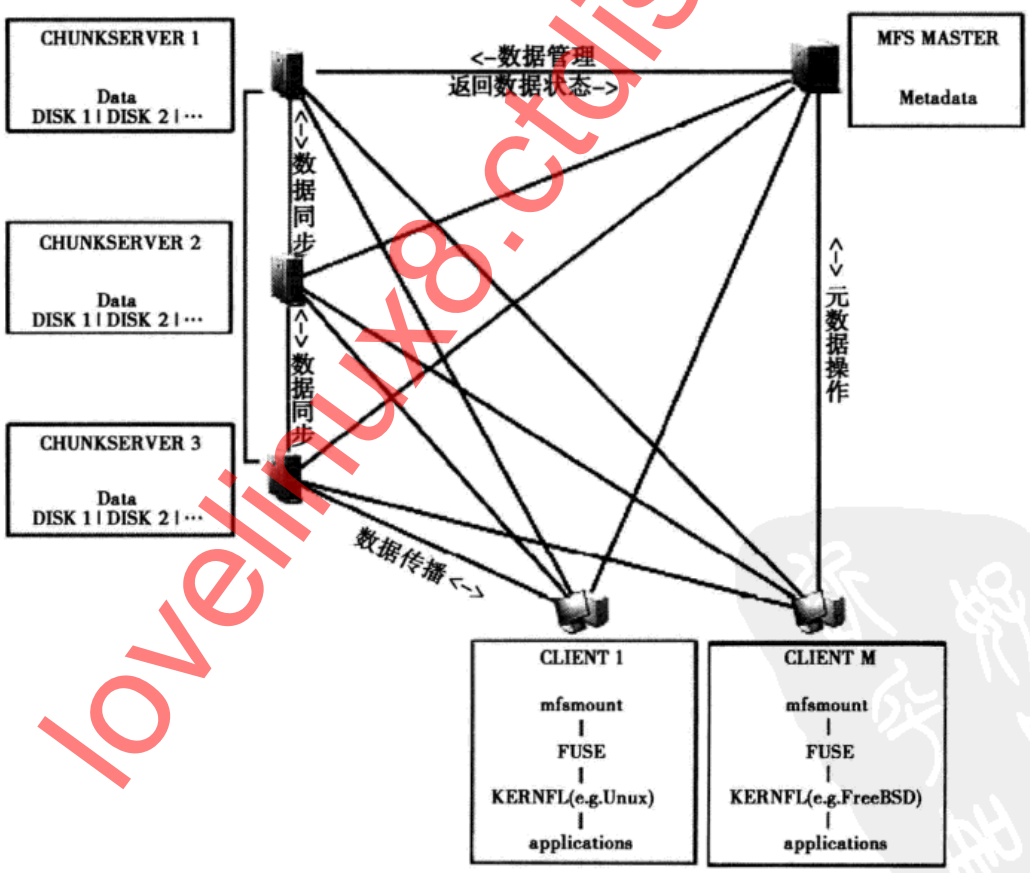

图 8-1　MFS 组成及运行原理图

MFS 的网络组成有三部分：MASTER SERVER、CHUNK SERVER 和 CLIENT，其中 MASTER SERVER 只有一个，而 CHUNK SERVER 和 CLIENT 可以有多个。MFS 读进程和

写进程的工作机制分别如图 8-2 和图 8-3 所示。

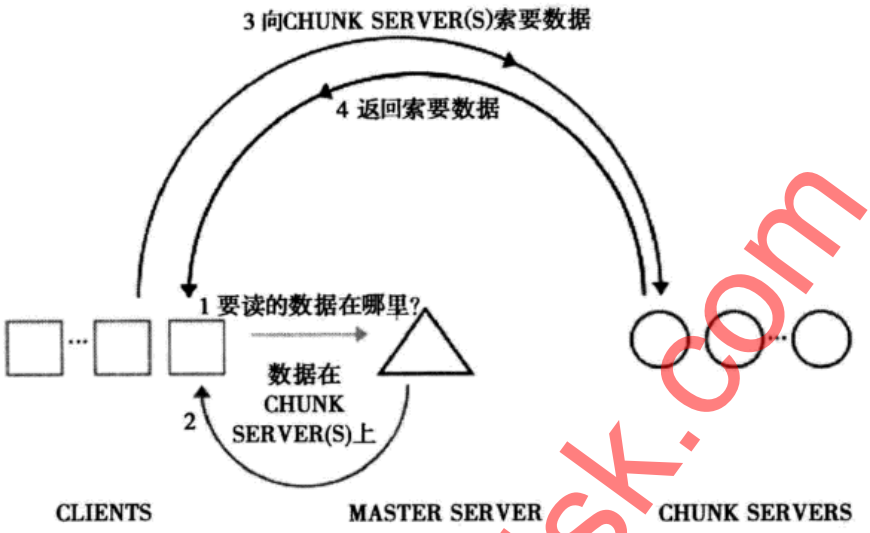

图 8-2　MFS 读进程的工作机制

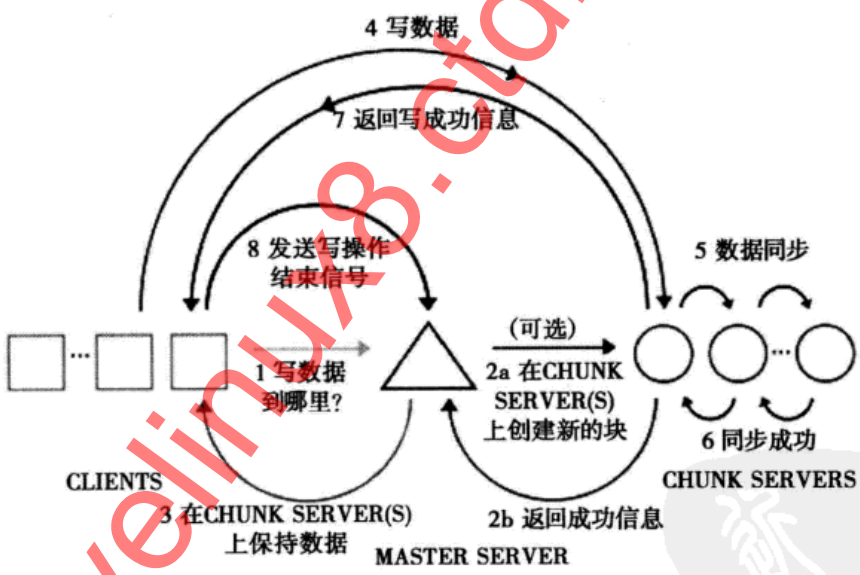

图 8-3　MFS 写进程的工作机制

从图 8-3 中可以清楚地看出 MFS 在写入数据时的内部运作过程。

8.2　MFS 文件系统

8.2.1　MFS 文件系统结构

MFS 文件系统结构参见图 8-1，整个文件系统包含 4 种角色，分别是：

- 管理服务器——MASTER SERVER
- 元数据日志服务器——Metalogger
- 数据存储服务器——CHUNK SERVER
- 客户端——CLIENT

4种角色作用如下：

- 管理服务器，有时也称为元数据服务器，负责管理各个数据存储服务器，调度文件读写，回收文件空间以及恢复多节点拷贝。
- 元数据日志服务器负责备份管理服务器的变化日志文件，文件类型为changelog_ml.*.mfs，以便于在管理服务器出问题时接替其进行工作。元数据日志服务器是mfs1.6以后版本新增的服务，可以把元数据日志保留在管理服务器中，也可以单独存储在一台服务器中。为保证数据的安全性和可靠性，建议单独用一台服务器来存放元数据日志。需要注意的是，元数据日志守护进程跟管理服务器在同一个服务器上，备份元数据日志服务器作为它的客户端，从管理服务器取得日志文件进行备份。
- 数据存储服务器是真正存储用户数据的服务器。在存储文件时，首先把文件分成块，然后将这些块在数据存储服务器之间互相复制。同时，数据存储服务器还负责连接管理服务器，听从管理服务器调度，并为客户提供数据传输。数据存储服务器可以有多个，并且数量越多，可靠性越高，MFS可用的磁盘空间也越大。
- 客户端通过fuse内核接口挂接远程管理服务器上所管理的数据存储服务器，使共享的文件系统和使用本地Linux文件系统的效果看起来是一样的。

8.2.2 MFS的编译与安装实例

这里假定元数据服务的IP地址为192.168.3.34，3个MFS客户端的IP地址分别为192.168.3.98、192.168.3.138和192.168.3.139。下面介绍MFS的搭建过程。

1. 安装和配置元数据服务

（1）下载源码

```
[root@nas ~]#wget \
>http://ncu.dl.sourceforge.net/project/moosefs/moosefs/1.6.11/mfs-1.6.11.tar.gz
```

（2）创建用户

```
[root@nas ~]#useradd mfs -s /sbin/nologin
```

（3）解压源码

```
[root@nas ~]#tar zxvf mfs-1.6.11.tar.gz
[root@nas ~]#cd mfs-1.6.11
```

（4）脚本配置

```
[root@nas mfs-1.6.11]#./configure --prefix=/usr/local/mfs \
```

```
>--with-default-user=mfs  --with-default-group=mfs
```

（5）编译安装

```
[root@nas mfs-1.6.11]#make; make install
```

（6）配置文件

配置文件位于安装目录 /usr/local/mfs/etc 下，需要用到的配置文件有两个：mfsmaster.cfg 和 mfsexports.cfg。mfsmaster.cfg 是主配置文件；mfsexports.cfg 对被挂接目录及其权限进行设置。

1）mfsmaster.cfg 的配置。

mfsmaster.cfg 文件的内容如下：

```
[root@nas etc]# cp mfsmaster.cfg.dist mfsmaster.cfg
[root@nas etc]# vi mfsmaster.cfg
# WORKING_USER = mfs
# WORKING_GROUP = mfs
# SYSLOG_IDENT = mfsmaster
# LOCK_MEMORY = 0
# NICE_LEVEL = -19
# EXPORTS_FILENAME = /usr/local/mfs/etc/mfsexports.cfg
# DATA_PATH = /usr/local/mfs/var/mfs
# BACK_LOGS = 50
# REPLICATIONS_DELAY_INIT = 300
# REPLICATIONS_DELAY_DISCONNECT = 3600
# MATOML_LISTEN_HOST = *
# MATOML_LISTEN_PORT = 9419
# MATOCS_LISTEN_HOST = *
# MATOCS_LISTEN_PORT = 9420
# MATOCU_LISTEN_HOST = *
# MATOCU_LISTEN_PORT = 9421
# CHUNKS_LOOP_TIME = 300
# CHUNKS_DEL_LIMIT = 100
# CHUNKS_WRITE_REP_LIMIT = 1
# CHUNKS_READ_REP_LIMIT = 5
# REJECT_OLD_CLIENTS = 0
```

下面解释这些变量的含义。需要注意的是，凡是用 # 注释掉的变量均使用默认值。

- WORKING_USER 和 WORKING_GROUP：运行 MASTER SERVER 的用户和组。
- SYSLOG_IDENT：是 MASTER SERVER 在 syslog 中的标识，说明这是由 MASTER SERVER 产生的。
- LOCK_MEMORY：是否执行 mlockall() 以避免 mfsmaster 进程溢出（默认为 0）。
- NICE_LEVEL：运行的优先级（默认是 -19。注意，进程必须是由 root 启动的）。
- EXPORTS_FILENAME：被挂接目录及其权限控制文件的存放位置。
- DATA_PATH：数据存放路径，此目录下大致有三类文件，changelog、sessions 和 stats。
- BACK_LOGS：元数据的改变日志文件数量（默认是 50）。
- REPLICATIONS_DELAY_INIT：延迟复制的时间（默认是 300 秒）。

- REPLICATIONS_DELAY_DISCONNECT：CHUNK SERVER 断开复制的延迟（默认是 3600 秒）。
- MATOML_LISTEN_HOST：元数据日志服务器监听的 IP 地址（默认是 *，代表任何 IP）。
- MATOML_LISTEN_PORT：元数据日志服务器监听的端口地址（默认是 9419）。
- MATOCS_LISTEN_HOST：用于 CHUNK SERVER 连接的 IP 地址（默认是 *，代表任何 IP）。
- MATOCS_LISTEN_PORT：用于 CHUNK SERVER 连接的端口地址（默认是 9420）。
- MATOCU_LISTEN_HOST：用于客户端挂接连接的 IP 地址（默认是 *，代表任何 IP）。
- MATOCU_LISTEN_PORT：用于客户端挂接连接的端口地址（默认是 9421）。
- CHUNKS_LOOP_TIME:chunks 的回环频率（默认是 300 秒）。
- CHUNKS_DEL_LIMIT：表示在一个 loop 设备中可以删除 chunks 的最大数（默认是 100）。
- CHUNKS_WRITE_REP_LIMIT：在一个循环里复制到一个 CHUNK SERVER 的最大 chunks 数目（默认是 1）。
- CHUNKS_READ_REP_LIMIT：在一个循环里从一个 CHUNK SERVER 中复制的最大 chunks 数目（默认是 5）。
- REJECT_OLD_CLIENTS：弹出低于 1.6.0 的客户端挂接（0 或 1，默认是 0）。

注意 mfsexports 访问控制对那些老客户是没用的。

以上是对 MASTER SERVER 的 mfsmaster.cfg 配置文件的解释，这个文件不需要任何修改就可以工作。

2）mfsexports.cfg 的配置。

mfsexports.cfg 文件中的内容如下：

```
[root@nas etc]# cp mfsexports.cfg.dist mfsexports.cfg
[root@nas etc]# vi mfsexports.cfg
#*                      /       ro
#192.168.1.0/24         /       rw
#192.168.1.0/24         /       rw,alldirs,maproot=0,password=passcode
#10.0.0.0-10.0.0.5      /test   rw,maproot=nobody,password=test
*                       .       rw
#*                      /       rw,alldirs,maproot=0

192.168.3.98            /tt     rw,alldirs,maproot=0
192.168.3.139           /       rw,alldirs,maproot=0
192.168.3.138           /       rw,alldirs,maproot=0,password=111111
```

该文件每一个条目分为三部分，第一部分表示客户端的 IP 地址，第二部分表示被挂接的目录，第三部分表示客户端拥有的权限。下面对三个部分进行解释。

- 客户端 IP 地址

可以将 IP 地址指定为以下几种表现形式：

- * 所有的 IP 地址
- n.n.n.n 单个 IP 地址
- n.n.n.n/b IP 网络地址 / 位数掩码
- n.n.n.n/m.m.m.m IP 网络地址 / 子网掩码
- f.f.f.f-t.t.t.t IP 段

❑ 被挂载的目录

目录部分需要注意以下两点：

- / 表示 MooseFS 根。
- . 表示 MFSMETA 文件系统 。

❑ 客户端拥有的权限

权限部分选项含义如下：

- ro，只读模式共享。
- rw，读写方式共享。
- alldirs，允许挂载任何指定的子目录。
- maproot，映射为 root 用户还是指定的用户。
- password，指定客户端密码。

3）复制文件。

```
[root@nas etc]# cp /usr/local/mfs/var/mfs/metadata.mfs.empty \
>/usr/local/mfs/var/mfs/metadata.mfs
```

默认的元数据文件为 metadata.mfs.empty，这是 mfs-1.6.x 新增的一个选项。要启动 MASTER SERVER，需要将 metadata.mfs.empty 改名为 metadata.mfs。

（7）启动 MASTER SERVER

启动 MASTER SERVER 的命令为 mfsmaster，其参数的使用方法如表 8-1 所示。

表 8-1　mfsmaster 命令的参数及其作用

参　数	作　用
-h	显示帮助信息并退出
-v	显示版本号并退出
-d	运行在前台
-u	记录未定义的配置变量
-t locktimeout	等待 lockfile 的时长
-c cfgfile	使用指定的配置文件
start	启动 mfsmaster
stop	停止 mfsmaster
restart	重新启动 mfsmaster
reload	重新载入 mfsmaster

下面启动 MASTER SERVER。执行过程如下：

```
[root@nas etc]#/usr/local/mfs/sbin/mfsmaster start
```

MASTER SERVER 可以单独启动，即使没有任何数据存储服务器（CHUNK SERVER）也是能正常工作的。安装配置完 MFS 后，即可启动 MASTER SERVER。如果没有意外，MASTER SERVER 应该作为一个守护进程运行起来。MASTER SERVER 是否启动，可以通过如下命令进行检查。

```
[root@nas etc]# ps -ef|grep mfs
mfs       12327     1  0 08:38 ?        00:00:00 /usr/local/mfs/sbin/mfsmaster start
```

（8）停止 MASTER SERVER

```
[root@nas etc]#/usr/local/mfs/sbin/mfsmaster -s
```

安全停止 MASTER SERVER 是非常必要的，最好不要用 kill，应该利用“mfsmaster -s”来安全停止 MASTER SERVER。使用了 kill 也有解决方法，后面会进行介绍。

（9）查看系统日志

查看系统日志的命令为 tail -f /var/log/messages。

2. 安装和配置元数据日志服务器

（1）下载源码

```
[root@mail ~]#wget \
>http://ncu.dl.sourceforge.net/project/moosefs/moosefs/1.6.11/mfs-1.6.11.tar.gz
```

（2）创建 MFS 用户

```
[root@mail ~]#useradd mfs -s /sbin/nologin
```

（3）解压源码

```
[root@mail ~]#tar zxvf mfs-1.6.11.tar.gz
[root@mail ~]#cd mfs-1.6.11
```

（4）脚本配置

```
[root@mail mfs-1.6.11]#./configure \
>--prefix=/usr/local/mfs --with-default-user=mfs  --with-default-group=mfs
```

（5）编译安装

```
[root@mail mfs-1.6.11]#make ; make install
```

（6）配置文件介绍

MFS 安装完成后，默认的配置文件位于 /usr/local/mfs/etc 目录下，该服务只有一个配置文件，即 mfsmetalogger.cfg。该文件内容如下：

```
[root@mail etc]# cp mfsmetalogger.cfg.dist mfsmetalogger.cfg
[root@mail etc]# vi mfsmetalogger.cfg
# WORKING_USER = mfs
```

```
# WORKING_GROUP = mfs
# SYSLOG_IDENT = mfsmetalogger
# LOCK_MEMORY = 0
# NICE_LEVEL = -19
# DATA_PATH = /usr/local/mfs/var/mfs
# BACK_LOGS = 50
# META_DOWNLOAD_FREQ = 24
# MASTER_RECONNECTION_DELAY = 5
MASTER_HOST = 192.168.3.34
# MASTER_PORT = 9419
# MASTER_TIMEOUT = 60
# deprecated, to be removed in MooseFS 1.7
# LOCK_FILE = /var/run/mfs/mfsmetalogger.lock
```

该文件中的多数变量不难理解，与 mfsmaster.cfg 中的变量类似，其中两个介绍如下：

- META_DOWNLOAD_FREQ，元数据备份文件下载请求频率。默认为 24 小时，即每隔一天从元数据服务器上下载一个 metadata.mfs.back 文件。当元数据服务器关闭或出故障时，matedata.mfs.back 文件将消失，此时要恢复整个 MFS，需从元数据日志服务器中取得该文件。注意，这个文件与日志文件共同使用才能够恢复整个被损坏的分布式文件系统。
- MASTER_HOST，这个文件中需要修改的是 MASTER_HOST 变量，这个变量的值是 MASTER SERVER 的 IP 地址。

（7）启动元数据日志服务器

启动元数据日志服务器的命令为 mfsmetalogger，其参数的作用如表 8-2 所示。

表 8-2　命令 mfsmetalogger 的参数的作用

参　数	作　用
-h	显示帮助信息并退出
-v	显示版本号并退出
-d	运行在前台
-u	记录未定义的配置变量
-t locktimeout	等待 lockfile 的时长
-c cfgfile	使用给定的配置文件
start	启动 mfsmetalogger
stop	停止 mfsmetalogger
restart	重新启动 mfsmetalogger
reload	重新载入 mfsmetalogger

启动元数据日志服务器的执行过程如下：

```
[root@mail sbin]#/usr/local/mfs/sbin/mfsmetalogger start
working directory: /usr/local/mfs/var/mfs
lockfile created and locked
```

```
initializing mfsmetalogger modules ...
mfsmetalogger daemon initialized properly
```

这说明元数据日志服务器正常启动了。可利用如下命令检查 MFS 进程信息：

```
[root@mail sbin]# ps -ef |grep mfs
mfs       12254     1  0 15:25 ?        00:00:00  /usr/local/mfs/sbin/mfsmetalogger start
```

查看 MFS 通信端口是否打开的命令如下：

```
[root@mail sbin]# lsof -i:9419
COMMAND     PID USER   FD   TYPE  DEVICE SIZE NODE NAME
mfsmetalo 12292  mfs   7u IPv4 1395372  TCP mail:52456->192.168.3.34:9419 (ESTABLISHED)
```

查看元数据日志服务器的工作目录的命令如下：

```
[root@mail mfs]# pwd
/usr/local/mfs/var/mfs
[root@mail mfs]# ll
total 8
-rw-r----- 1 mfs mfs 249 Jan 13 15:39 changelog_ml.1.mfs
-rw-r----- 1 mfs mfs 519 Jan 13 15:40 sessions_ml.mfs
```

（8）停止元数据日志服务器

```
[root@mail sbin]# /usr/local/mfs/sbin/mfsmetalogger -s
working directory: /usr/local/mfs/var/mfs
sending SIGTERM to lock owner (pid:12254)
waiting for termination ... terminated
```

3. 安装配置数据存储服务器

（1）下载源码

```
[root@chunkserver ~]#wget \
>http://ncu.dl.sourceforge.net/project/moosefs/moosefs/1.6.11/mfs-1.6.11.tar.gz
```

（2）创建用户

```
[root@chunkserver ~]#useradd mfs -s /sbin/nologin
```

（3）解压源码

```
[root@chunkserver ~]#tar zxvf mfs-1.6.11.tar.gz
[root@chunkserver ~]#cd mfs-1.6.11
```

（4）脚本配置

```
[root@chunkserver mfs-1.6.11]#./configure \
>--prefix=/usr/local/mfs --with-default-user=mfs --with-default-group=mfs
```

（5）编译安装

```
[root@chunkserver mfs-1.6.11]#make ; make install
```

（6）配置文件

配置文件位于安装目录 /usr/local/mfs/etc 下。需要用到的配置文件有两个：mfschunkserver.

cfg 和 mfshdd.cfg。mfschunkserver.cfg 是主配置文件；mfshdd.cfg 配置文件用来指定服务器分配给 MFS 使用的空间，这个空间最好是一个单独的硬盘或者一个 raid 卷，至少是一个磁盘分区。

mfschunkserver.cfg 中的配置内容如下：

```
[root@chunkserver etc]# cp mfschunkserver.cfg.dist mfschunkserver.cfg
[root@chunkserver etc]# vi mfschunkserver.cfg
# WORKING_USER = mfs
# WORKING_GROUP = mfs
# DATA_PATH = /usr/local/mfs/var/mfs
# LOCK_FILE = /var/run/mfs/mfschunkserver.pid
# SYSLOG_IDENT = mfschunkserver
# BACK_LOGS = 50
# MASTER_RECONNECTION_DELAY = 30
 MASTER_HOST = 192.168.3.34
 MASTER_PORT = 9420
# MASTER_TIMEOUT = 60
# CSSERV_LISTEN_HOST = *
# CSSERV_LISTEN_PORT = 9422
# CSSERV_TIMEOUT = 60
# CSTOCS_TIMEOUT = 60
# HDD_CONF_FILENAME = /usr/local/mfs/etc/mfshdd.cfg
```

该文件中的多数变量不难理解，类似于 mfsmaster.cfg 中的变量，其中三个介绍如下：

- MASTER_HOST，元数据服务器的名称或地址，可以是主机名，也可以是 IP 地址。
- CSSERV_LISTEN_PORT，这个监听端口用于与其他数据存储服务器间的连接，通常是数据复制。
- HDD_CONF_FILENAME，分配给 MFS 使用的磁盘空间配置文件的位置。

mfshdd.cfg 的配置如下：

```
[root@chunkserver etc]# cp mfshdd.cfg.dist mfshdd.cfg
[root@chunkserver etc]# more mfshdd.cfg
/data
```

在这里 /data 是一个 MFS 的分区，但在本机上是一个独立的磁盘挂载分区。执行如下命令，可将此分区的属主改为 MFS。

```
[root@chunkserver etc]#chown -R mfs:mfs /data
```

（7）启动数据存储服务器

使用命令 mfschunkserver 即可启动 CHUNK SERVER。mfschunkserver 命令的用法与 mfsmaster 和 mfsmetalogger 完全相同，这里不再过多解释。执行如下命令启动数据存储服务器：

```
[root@chunkserver etc]#/usr/local/mfs/sbin/mfschunkserver start
```

如果没有意外，数据存储服务器应该作为一个守护进程运行起来。数据存储服务器是否启动，可以通过如下命令进行检查：

```
[root@chunkserver etc]# ps -ef|grep mfs
mfs      12327     1  0 08:38 ?        00:00:00 /usr/local/mfs/sbin/mfschunkserver start
```

（8）停止数据存储服务器

要停止数据存储服务器，最安全的方式是执行“mfschunkserver -s”命令。

```
[root@chunkserver etc]#/usr/local/mfs/sbin/mfschunkserver -s
```

4. MFS 客户端的安装及配置

由于 MFS 客户端依赖于 fuse，所以要先安装 fuse。

（1）fuse 的安装

1）下载源码。

```
[root@www ~]#wget \
>http://cdnetworks-kr-1.dl.sourceforge.net/project/fuse/fuse-2.X/2.8.1/fuse-
    2.8.1.tar.gz
```

2）解压源码。

```
[root@www ~]#tar zxvf fuse-2.8.1.tar.gz
[root@www ~]#cd fuse-2.8.1
```

3）脚本配置。

```
[root@www fuse-2.8.1]#./configure
```

4）编译安装。

```
[root@www fuse-2.8.1]#make ; make install
```

如果所在的系统已经安装了 fuse，则跳过这个步骤，高版本的 Linux 内核已经支持了。

（2）安装 MFS 客户端

1）下载源码。

```
[root@www ~]#wget \
>http://ncu.dl.sourceforge.net/project/moosefs/moosefs/1.6.11/mfs-1.6.11.tar.gz
```

2）创建 MFS 用户。

```
[root@www ~]#useradd mfs -s /sbin/nologin
```

3）解压源码。

```
[root@www ~]#tar zxvf mfs-1.6.11.tar.gz
[root@www ~]#cd mfs-1.6.11
```

4）脚本配置。

```
[root@www mfs-1.6.11]#./configure --prefix=/usr/local/mfs --with-default-user=mfs  \
>--with-default-group=mfs --enable-mfsmount
```

在这个过程中，当执行到“-enable-mfsmount”时可能出现下面的错误：

```
“checking for FUSE… no  configure: error: mfsmount build was forced, but fuse
    development package is not installed”
```

这样的错误导致不能正确安装 MFS 客户端程序，这是因为没有设置环境变量。通过下面两种方法可以解决这个问题：

- 编辑 /etc/profile，在此文件中加入如下内容：

```
export  PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH
```

然后再利用 source 命令 /etc/profile 使修改生效，即执行 source /etc/profile。

- 直接在命令行中执行以下命令：

```
export  PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH
```

5）编译安装。

```
[root@www mfs-1.6.11]#make ; make install
```

（3）挂接 MFS 文件系统

1）创建挂接点。

```
[root@www ~]#mkdir /mnt/mfs
```

2）加载 fuse 模块到内核。

```
[root@www ~]#modprobe fuse
```

3）挂接 MFS。

```
[root@www ~]#/usr/local/mfs/bin/mfsmount  /mnt/mfs -H 192.168.3.34 -p
```

然后输入密码就可以了。特别需要注意的是，所有的 MFS 都挂接的是同一个元数据服务器的 IP，而不是其他数据存储服务器的 IP。

（4）挂接 MFSMETA 文件系统

1）创建挂接点。

```
[root@www ~]#mkdir /mnt/mfsmeta
```

2）挂接 MFSMETA。

```
[root@www ~]#/usr/local/mfs/bin/mfsmount -m /mnt/mfsmeta/  -H 192.168.3.34
```

3）查看目录内容。

```
[root@www ~]# ls -R /mnt/mfsmeta
/mnt/meta/:
reserved  trash

/mnt/meta/reserved:

/mnt/meta/trash:
undel

/mnt/meta/trash/undel:
```

（5）查看挂载情况

通过 df 命令查看磁盘使用情况以检查是否挂接成功。

```
[root@www ~]# df -h
Filesystem            Size  Used  Avail  Use%  Mounted on
/dev/mapper/VolGroup00-LogVol00
                      73G   25G   45G    36%   /
/dev/sda1             99M   13M   82M    13%   /boot
none                  247M  0     247M   0%    /dev/shm
MFS                   5G    204M  45G    1%    /mnt/mfs
MFSMETA               72K   72K   0      100%  /mnt/mfsmeta
```

利用 mount 命令查看 MFS 挂接信息。

```
[root@www ~]# mount
mfsmeta#192.168.3.34:9421 on /mnt/mfsmeta type fuse (rw,nosuid,nodev,allow_
    other,default_permissions)
mfs#192.168.3.34:9421 on /mnt/mfs type fuse (rw,nosuid,nodev,allow_other,default_
    permissions
```

（6）卸载已挂接的文件系统

利用 Linux 系统的 umount 命令就可以卸载已挂接的文件系统，例如：

```
[root@www ~]# umount /mnt/mfs
```

执行后出现下列情况：

```
[root@www ~]# umount /mnt/mfs
umount: /mnt/mfs: device is busy
umount: /mnt/mfs: device is busy
```

这说明客户端本机正在使用此文件系统，利用“fuser”、“lsof –p”等命令，可查明是什么进程在使用，然后关闭相应的进程就可以成功卸载此文件系统了。最好不要强制退出。

5. 通过 mfscgiserv 监控客户端连接状态

mfscgiserv 是用 Python 编写的一个 Web 服务器，它的监听端口是 9425。

命令 mfscgiserv 中的参数的作用如表 8-3 所示。

表 8-3 mfscgiserv 中的参数的作用

参 数	作 用
-h	-help，显示帮助信息
[-H bind_host]	绑定 IP 地址，默认值为 any
[-P bind_port]	绑定端口号，默认值为 9425
[-R rootpath]	mfscgi 的 root 路径，默认值为 /usr/local/mfs/share/mfscgi
[-f [-v]]	运行 HTTP 服务器 (-f 表示运行在前台；-v 表示请求的日志发往标准的错误设备)

可以利用如下命令在管理服务器上启动 mfscgiserv：

```
[root@nas ~]#/usr/local/mfs/sbin/mfscgiserv
```

用户利用浏览器就可全面监控所有客户挂接、CHUNK SERVER、MASTER SERVER 及客户端的各种操作等信息，绝对是个好工具。

访问方法很简单，在浏览器中输入“http://masterserver IP: 端口号”即可。例如，在浏览器中输入“http://192.168.3.34:9425”。从浏览器中可以看到这个图形工具由 8 个部分组成，分别是：Info、Servers、Disks、Exports、Mounts、Operations、Master Charts 和 Server Charts。

下面介绍一下各部分的情况。

- Info 部分：这个部分显示了 MFS 的基本信息，如图 8-4 所示。

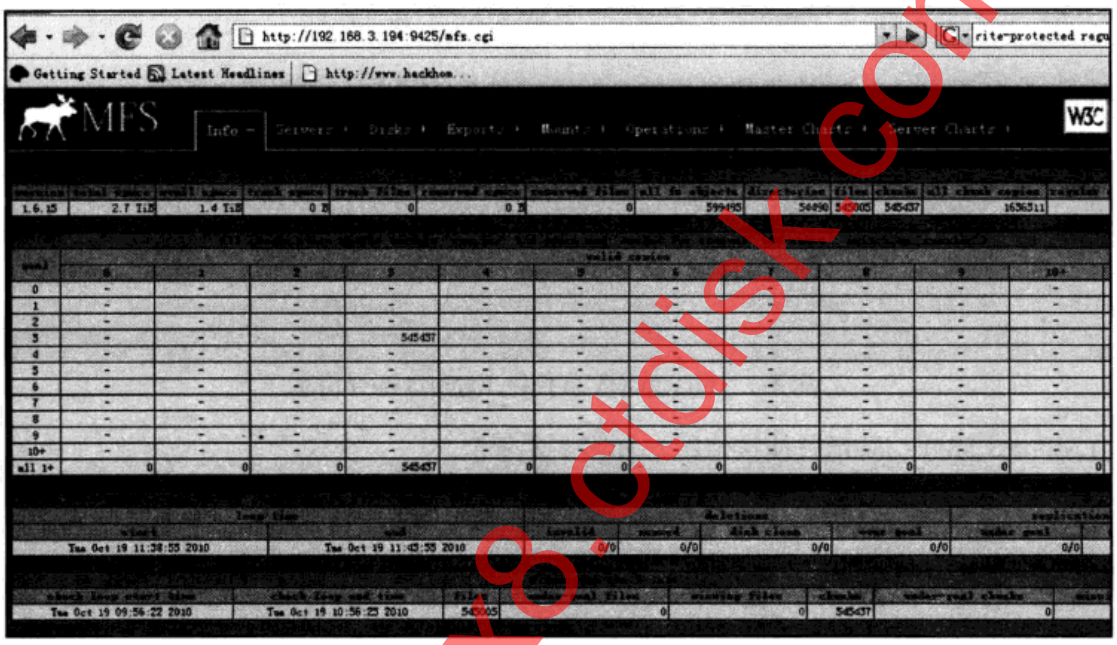

图 8-4　mfscgiserv 监控之 Info 部分信息

- Servers 部分：列出现有的 CHUNK SERVER，如图 8-5 所示。

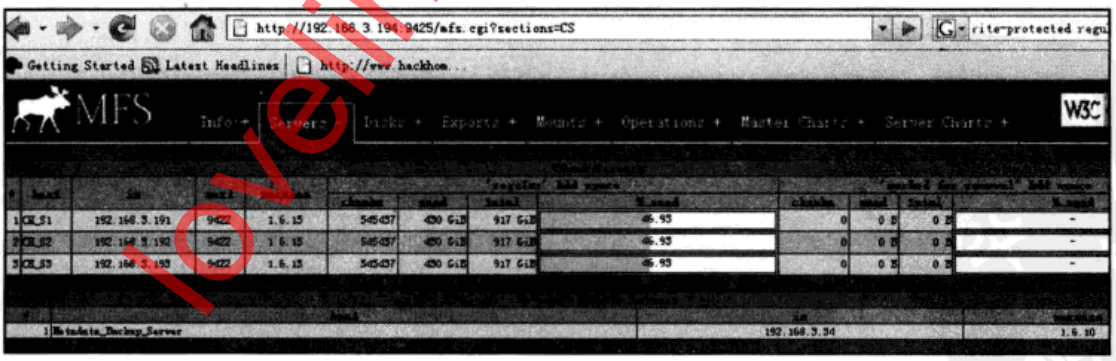

图 8-5　mfscgiserv 监控之 Servers 部分信息

- Disks 部分：列出每一台 CHUNK SERVER 的磁盘目录及使用量，如图 8-6 所示。

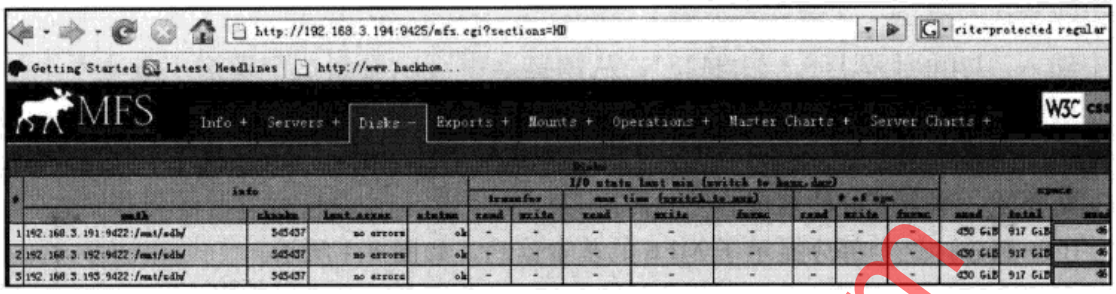

图 8-6 mfscgiserv 监控之 Disks 部分信息

- Exports 部分：列出被共享的目录，即可被挂接的目录，如图 8-7 所示。

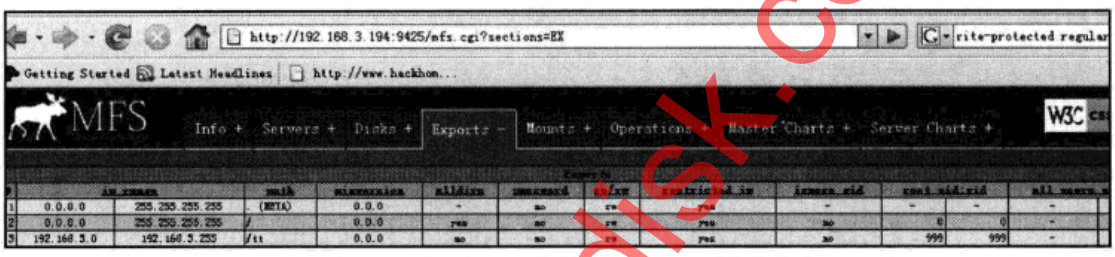

图 8-7 mfscgiserv 监控之 Exports 部分信息

- Mounts 部分：显示被挂接情况，如图 8-8 所示。

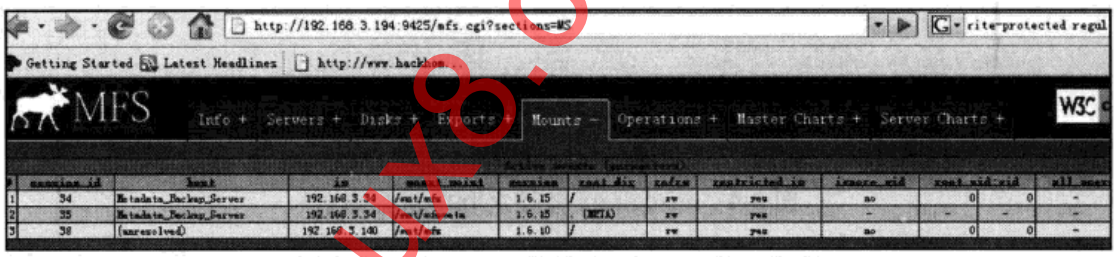

图 8-8 mfscgiserv 监控之 Mounts 部分信息

- Operations 部分：显示正在执行的操作，如图 8-9 所示。

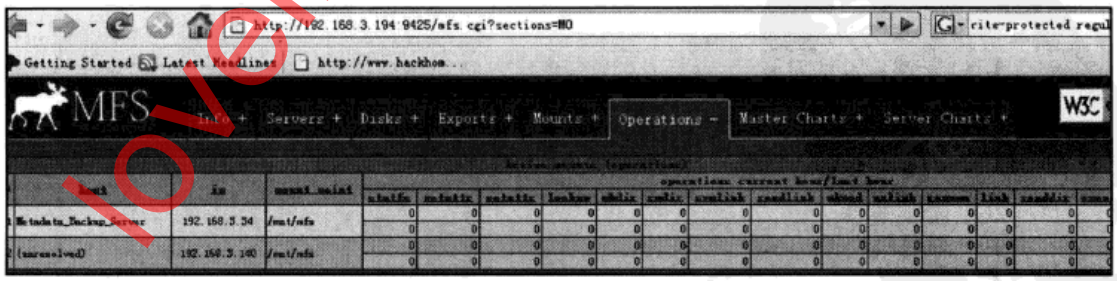

图 8-9 mfscgiserv 监控之 Operations 部分信息

- Master Charts 部分：显示 MASTER SERVER 的操作情况，包括读取、写入、创建目录、删除目录等信息，由于版面有限，这里只显示其中的一小部分信息，如图 8-10 所示。

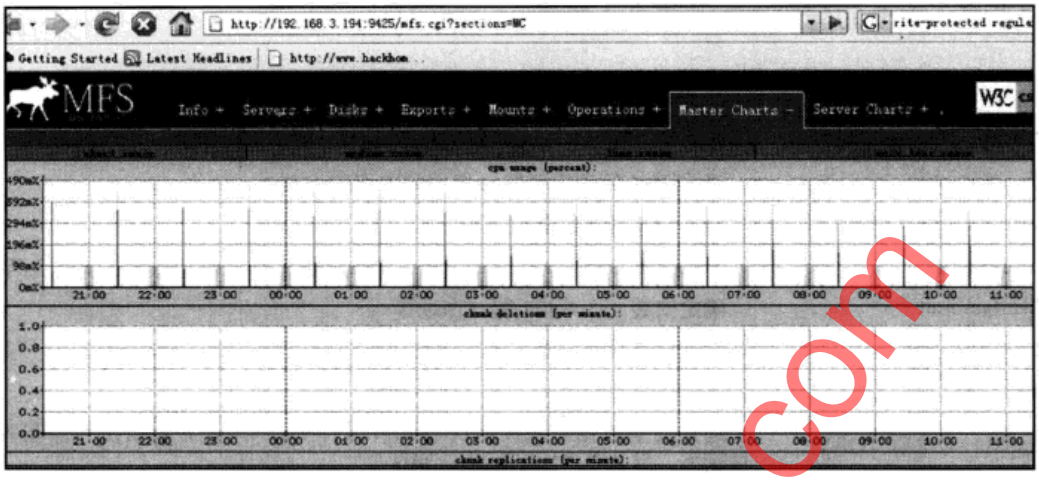

图 8-10　mfscgiserv 监控之 Master Charts 部分信息

- Server Charts 部分：显示 CHUNK SERVER 的操作情况、数据传输率及系统状态等信息，如图 8-11 所示。

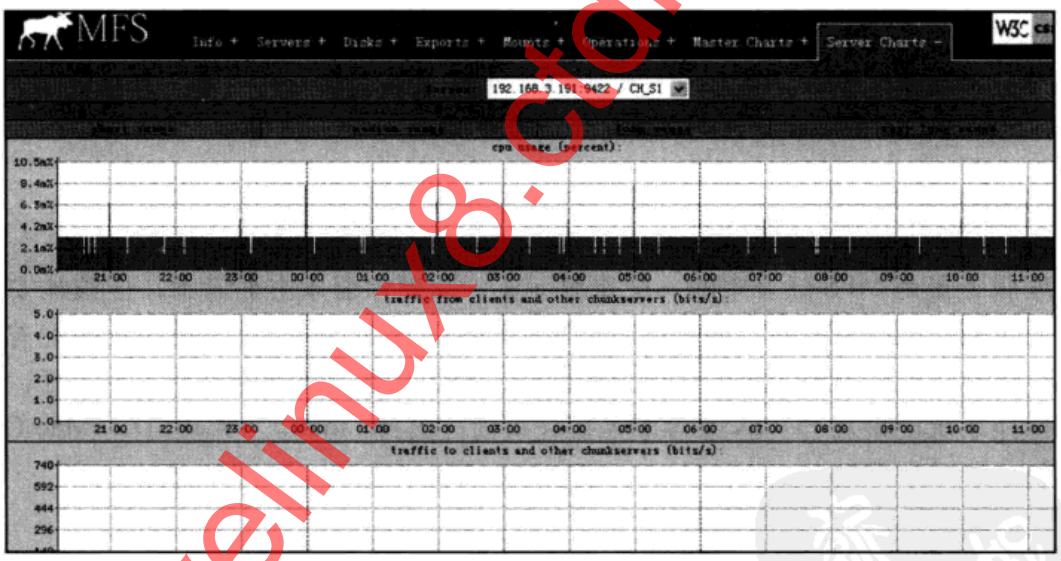

图 8-11　mfscgiserv 监控之 Server Charts 部分信息

由于版面有限，这里只显示了其中的一小部分信息。另外，注意上面的 Server 下拉列表框，在这里选择不同的 Server 后可以看到不同 CHUNK SERVER 的情况。

8.3　编译与使用 MFS 的经验总结

8.3.1　安装选项说明

部署 MFS 的首选方法是从源代码安装。源代码安装支持标准 ./configure → make → make

install 的步骤，重要的配置选项如表 8-4 所示。

表 8-4 MFS 的配置选项说明

选 项	作 用
--disable-mfsmaster	不创建成管理服务器（用于纯节点的安装）
--disable-mfschunkserver	不创建成数据存储服务器
--disable-mfsmount	不创建 mfsmount 和 mfstools（如果用开发包安装，会默认创建这两者）
--enable-mfsmount	确定安装 mfsmount 和 mfstools
--prefix=DIRECTORY	锁定安装目录（默认是 /usr/local）
--sysconfdir=DIRECTORY	选择配置文件目录（默认是 ${prefix}/etc）
--localstatedir=DIRECTORY	选择变量数据目录（默认是 ${prefix}/var，MFS 元数据被存储在 MFS 的子目录下，默认是 ${prefix}/var/mfs ）
--with-default-user	运行守护进程的用户，如果配置文件中没有设定用户，默认为 nobody 用户
--with-default-group=GROUP	运行守护进程的用户组，如果配置文件中没有设定用户组，默认为 nogroup 用户组

例如，利用 FHS（文件系统层次标准）的兼容路径在 Linux 上的安装如下：

```
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib
```

编译安装遵守标准的 DESTDIR= variable，允许安装包在临时目录（如已创建的二进制包）下，已经存在的配置或元数据文件将会被覆盖掉。

8.3.2 管理服务器

管理服务器是 MFS 部署中的一个重要元素，从硬件要求方面考虑，应该被安装在一台能够保证高可靠性和能胜任整个系统存取要求的机器上，一个明智的做法是配有冗余电源、ECC 内存、磁盘阵列，如 RAID1/RAID5/RAID10；从操作系统要求方面考虑，管理服务器的操作系统应该是 POSIX 兼容的系统（目前，支持 Linux。 FreeBSD。 Mac OS X 和 OpenSolaris）。

安装管理服务器的过程大致如下：

1）安装 mfs-master。

2）如果是从源码安装，在设置 configure 选项时不要加 --disable-mfsmaster 选项。

3）创建运行管理服务器的用户（如果这样的用户不存在）。

4）确定存放元数据文件的目录是否存在，且能够被运行管理服务器的用户写入。可以通过 configure 的选项来设置运行管理服务器的用户和元数据存储的路径，make install 命令的执行必须是 root 用户。

5）配置管理服务器是通过配置文件 mfsmaster.cfg 来进行的，要特别注意的是 TCP 端口的使用。

6）最好添加或创建（依赖于操作系统和 MFS 发布版本）一组启动 mfsmaster 进程的脚本。

安装完管理服务器后，便可以利用 mfsmaster 命令来启动它。如果是 root 用户执行 mfsmaster 命令，则要在管理服务器启动后转为由 mfsmaster.cfg 中指定的用户来运行，否则将由执行 mfsmaster 命令的用户来运行管理服务器。

8.3.3　元数据日志服务器

元数据日志的守护进程是在安装管理服务器时一同安装的，也就是说元数据日志守护进程是运行在元数据服务器上的，但大小不要比管理服务器本身大。元数据日志服务器可以运行在任何服务器上（例如任意一台 CHUNK SERVER），但是最好放置在 MFS 管理服务器之外的一台独立备份机上，它用来备份管理服务器变化的日志文件，文件的类型为 changelog_ml.*.mfs。这是因为主要的管理服务器一旦失效，可能就会取代这台元数据日志服务器而作为管理服务器。

安装元数据日志服务器的过程大致如下：

1）从源代码安装 mfs-master，在执行 configure 时不要带有 --disable-mfsmaste 选项。

2）创建有运行 mfsmetalogger 服务权限的用户（如果这样的用户不存在）。

3）确定存放元数据文件的目录是否存在，且能够被运行元数据日志服务器的用户写入，通过 configure 的选项来设置运行元数据日志服务器的用户和元数据的存储路径，make install 命令的执行必须是 root 用户。

4）通过 mfsmetalogger.cfg 文件来配置元数据日志服务器，要特别注意的是 TCP 端口，在这个文件中指定使用的 MASTER_PORT 必须和 mfsmaster.cfg 文件中的 MATOML_LISTEN_PORT 一致。

5）最好添加或创建（依赖于操作系统和 MFS 发布版本）一组启动元数据日志服务器进程的脚本。

安装完管理服务器后，便可以用 mfsmetalogger 命令来启动元数据日志服务器。如果是由 root 用户执行 mfsmetalogger 命令，则在启动后元数据日志服务器进程将转为由 mfsmetalogger.cfg 文件中指定的用户来运行，否则将由执行 mfsmetalogger 命令的用户来运行。

8.3.4　数据存储服务器

安装完元数据日志服务器后，将安装数据存储服务器。运行数据存储服务器的机器的磁盘上要有适当的剩余空间，而且操作系统要遵循 POSIX 标准。数据存储服务器在一个普通的文件系统上储存数据块 / 碎片（chunks/fragments）作为文件。

在独立的磁盘上创建一个普通文件系统，作为数据存储服务器的存储空间。在 Linux、FreeBSD 和 MacOS X 系统上的创建方式如下：

- Linux

```
creating:
```

```
dd if=/dev/zero of=file bs=100m seek=400 count=0
mkfs -t ext3 file
mounting:
mount -o loop file mount-point
```

❑ FreeBSD

```
creating and mounting:
dd if=/dev/zero of=file bs=100m count=400
mdconfig -a -t vnode -f file -u X
newfs -m0 -O2 /dev/mdX
mount /dev/mdX mount-point
mounting a previously created file system:
mdconfig -a -t vnode -f file -u X
mount /dev/mdX mount-point
```

❑ Mac OS X

```
Start "Disk Utility" from "/Applications/Utilities"
Select from menu "Images->New->Blank Image ..."
```

注意，每一个数据存储服务器的磁盘都要为增长中的块分区保留些磁盘空间，以便创建新的块分区。只有磁盘都超过 256MB 并且数据存储服务器报告自由空间超过 1GB 总量才允许新的数据访问，至少应该保留几个 GB 的存储空间。

安装数据存储服务器的过程大致如下：

1）把预先隔离的磁盘空间作为一个单独的文件系统，挂接在一个本地目录下（如 /mnt/hd1、/mnt/hd2 等）。

2）安装 mfs-chunkserver，在执行 configure 时不要加“--disable-mfschunkserver”选项。

3）创建运行 chunkserver 服务的系统用户（如果这样的用户不存在的话），并给予这个用户对整个 MooseFS 文件系统写的权限。

4）利用 mfschunkserver.cfg 文件配置数据存储服务器服务，要特别注意 TCP 端口。另外，mfschunkserver.cfg 文件的“MASTER_PORT”变量的值要和 mfsmaster.cfg 文件中的“MATOCS_LISTEN_PORT”变量的值一样。

5）在 mfshdd.conf 文件中列出要用于 MoFS 数据存储分区的挂载点。

6）添加或创建（依赖于操作系统和 MFS 发布版本）一组启动数据存储服务器进程的脚本。

需要特别注意的是，数据存储服务器的本地 IP 很重要，数据存储服务器利用此 IP 和管理服务器进行连接，管理服务器通过此 IP 和 MFS 客户端连接，而且其他数据存储服务器之间的通信也通过这个 IP 进行，因此这个 IP 必须是远程可访问的。必须将管理服务器的本地 IP 地址（MASTER_HOST）设置成和数据存储服务器一样，以便于正确连接。通常的做法是使管理服务器、数据存储服务器和 MFS 客户端在同一网段。一般的回环地址（即 localhost，127.0.0.1）不能用于 MASTER_HOST，因为它将使数据存储服务器无法被其他主机访问。

安装完数据存储服务器后，便可以用 mfschunkserver 命令来启动数据存储服务器服务，如果由 root 用户执行 mfschunkserver 命令，则在启动后转为由 mfschunkserver.cfg 中指定的

用户来运行，否则将由执行 mfschunkserver 命令的用户来运行数据存储服务器服务。

8.3.5　客户端挂载

客户端挂载需要 fuse 才可以正常工作，fuse 支持多种操作系统：Linux、FreeBSD、OpenSolaris 和 Mac OS X。

Linux 一个内核模块的 API 版本至少是 7.8，这可以通过 dmesg 命令来检测，在载入内核模块后，应该能看到一行有 fuse init (API version 7.8) 的内容。一些可用的 fuse 版本是 Linux kernel 2.6.20（Linux 内核从 2.6.20 版本后加入了 fuse）以上。由于存在一些小 bug，因此推荐使用比较新的模块，如 fuse 2.7.2 及 Linux 2.6.24（尽管 fuse 2.7.x 没有包含 getattr/write race condition fix）。在 FreeBSD 系统上，fusefs-kmod 版本要在 0.3.9 版本以上才可以，在 Mac OS X Mac 上 fuse 要为 10.5 版本。

安装 MooseFS 客户端的过程大致如下：

1）安装 mfs-client，从源代码安装，在进行 configure 时不要加“--disable-mfsmount”选项。

2）建立被 MFS 挂载的挂载点目录，例如 /mnt/mfs。

3）利用 mfsmount 命令挂载 MFS 文件系统。

8.4　管理与使用 MFS

8.4.1　在客户端挂载文件系统

启动管理服务器和数据存储服务器（数据存储服务器是必需的，但推荐至少启动两个）后，客户机便可以利用 mfsmount 挂接 MFS 文件系统。

mfsmount 的用法如下：

```
mfsmount [-H master] [-P port] [-S path] mountpoint
```

参数含义如表 8-5 所示。

表 8-5　mfsmount 命令的参数的作用

参　数	作　用
-H master	管理服务器的 IP 地址
-P port	管理服务器的端口号，要按照 mfsmaster.cfg 配置文件中的变量 MATOCU_LISTEN_POR 的值填写。如果管理服务器使用的是默认端口号，则不用指出
-S path	指出挂载 MFS 目录的子目录，默认是 / 目录，即挂载整个 MFS 目录
mountpoint	是指先前创建的用来挂接 MFS 的目录

在启动管理服务器进程时，用了一个“-m”或“-o mfsmeta”的选项，这样可以挂载一个辅助的文件系统 mfsmeta。辅助文件系统可以在如下两个方面恢复丢失的数据：

- 从 MFS 卷上误删除了文件，而此文件又过了垃圾文件存放期。
- 为了释放磁盘空间而删除或移动文件，当需要恢复这些文件时，文件又过了垃圾文件存放期。

要使用 MFS 辅助文件系统，可以执行如下命令：

```
mfsmount -m /mnt/mfsmeta
```

需要注意的是，如果决定挂载 mfsmeta，那么一定要在 mfsmaster 的 mfsexports.cfg 文件中加入如下条目：

```
*                    .          rw
```

原文件中有此条目，只要将其前面的 # 去掉即可。

挂载文件系统后就可以执行所有标准的文件操作了，如创建、复制、删除、重命名文件等。MFS 是一个网络文件系统，因此操作进度可能比本地系统要慢。

对 MFS 卷的剩余空间的检查可以采用和本地文件系统同样的方法。例如执行 df 命令如下：

```
[root@www ~]# df -h | grep mfs
mfsmaster:9421          85T   80T  4.9T  95% /mnt/mfs
mfsmaster:9321         394G  244G  151G  62% /mnt/mfs-test
```

注意，每一个文件可以被储存为多个副本，在这种情况下，每一个文件所占用的空间要比其文件本身大得多。此外，被删除且在有效期内（trashtime）的文件都放在一个“垃圾箱”中，所以它们也占用空间，其大小也依赖文件的份数。就像其他 Linux/UNIX 的文件系统一样，为防止删除被其他进程打开的文件，数据将一直被存储，直到文件被关闭。

8.4.2 MFS 常用操作

MFS 在客户端安装完毕后，会生成 /usr/local/mfs/bin 目录，在这个目录下有很多命令是用户所需要的，其中核心命令是 mfstools。为了能让系统识别到这些命令，可以将 /usr/local/mfs/bin 路径加到用户的环境变量中，也可以将这些命令复制到系统默认的命令路径下。

1. mfsgetgoal 与 mfssetgoal 命令

目标（goal）是指文件被复制的份数，设定了复制的份数后就可以通过 mfsgetgoal 命令来证实，也可以通过 mfssetgoal 来改变设定。例如：

```
[root@www ~]# mfsgetgoal /mnt/mfs-test/test1
/mnt/mfs-test/test1: 2
[root@www ~]# mfssetgoal 3 /mnt/mfs-test/test1
/mnt/mfs-test/test1: 3
[root@www ~]# mfsgetgoal /mnt/mfs-test/test1
/mnt/mfs-test/test1: 3
```

利用 mfsgetgoal –r 和 mfssetgoal –r 操作可以对整个树形目录进行递归操作。

```
[root@www ~]# mfsgetgoal -r /mnt/mfs-test/test2
```

```
/mnt/mfs-test/test2:
 files with goal        2 :                    36
 directories with goal  2 :                     1
[root@www ~]#  mfssetgoal -r 3 /mnt/mfs-test/test2
/mnt/mfs-test/test2:
 inodes with goal changed:                         37
 inodes with goal not changed:                      0
 inodes with permission denied:                     0
[root@www ~]# mfsgetgoal -r /mnt/mfs-test/test2
/mnt/mfs-test/test2:
 files with goal        3 :                    36
 directories with goal  3 :                     1
```

2. mfscheckfile 与 mfsfileinfo 命令

实际的副本份数可以通过 mfscheckfile 和 mfsfileinfo 命令来证实。例如：

```
[root@www ~]# mfscheckfile /mnt/mfs-test/test1
/mnt/mfs-test/test1:
3 copies: 1 chunks
[root@www ~]# mfsfileinfo /mnt/mfs-test/test1
/mnt/mfs-test/test1:
   chunk 0: 00000000000520DF_00000001 / (id:336095 ver:1)
           copy 1: 192.168.0.12:9622
           copy 2: 192.168.0.52:9622
           copy 3: 192.168.0.54:9622
```

注意，一个不包含数据的零长度的文件，虽然没有被设置为非零目标（the non-zero "goal"），但是查询后将返回一个空的结果。例如：

```
[root@www ~]#  touch    /mnt/mfs/mmm
[root@www ~]#  mfsfileinfo  /mnt/mfs/mmm
/mnt/mfs/mmm:
```

如果对此文件进行编辑，可执行如下命令：

```
[root@www ~]#  echo "1234"> /mnt/mfs/mmm
```

可以看到如下结果：

```
[root@www ~]#  mfsfileinfo  /mnt/mfs/mmm
/mnt/mfs/mmm:
   chunk 0: 0000000000000040_00000001 / (id:64 ver:1)
           copy 1: 192.168.3.31:9422
           copy 2: 192.168.3.96:9422
           copy 3: 192.168.3.139:9422
```

此时将文件清空。

```
[root@www ~]#  echo ""> /mnt/mfs/mmm
```

又会看到如下结果：

```
[root@www ~]# mfsfileinfo  /mnt/mfs/mmm
```

```
/mnt/mfs/mmm:
   chunk 0: 0000000000000041_00000001 / (id:65 ver:1)
            copy 1: 192.168.3.31:9422
            copy 2: 192.168.3.96:9422
            copy 3: 192.168.3.139:9422
```

可以看到，副本依然存在。

如果改变一个已经存在的文件的副本份数，那么文件的副本份数将会被扩大或删除，这个过程会有延时。可以通过上面的命令来证实。

对一个目录设定“目标”，此目录下新创建的文件和子目录均会继承此目录的设定，但不会改变已经存在的文件及目录的副本份数。例如：

```
[root@www f]# touch 1
[root@www f]# echo "11" > 1
[root@www f]# /usr/local/mfs/bin/mfsfileinfo 1
1:
   chunk 0: 0000000000000043_00000001 / (id:67 ver:1)
            copy 1: 192.168.3.31:9422
            copy 2: 192.168.3.96:9422
            copy 3: 192.168.3.139:9422
[root@www f]# cd ..
[root@www mfs]# /usr/local/mfs/bin/mfssetgoal 2 f
f: 2
[root@www mfs]# cd f/
[root@www f]# ls
1
[root@www f]# touch 2
[root@www f]# echo "222" > 2
[root@www f]# /usr/local/mfs/bin/mfsfileinfo 1
1:
   chunk 0: 0000000000000043_00000001 / (id:67 ver:1)
            copy 1: 192.168.3.31:9422
            copy 2: 192.168.3.96:9422
            copy 3: 192.168.3.139:9422
[root@www f]# /usr/local/mfs/bin/mfsfileinfo 2
2:
   chunk 0: 0000000000000044_00000001 / (id:68 ver:1)
            copy 1: 192.168.3.31:9422
            copy 2: 192.168.3.96:9422
```

3. mfsdirinfo 命令

整个目录树的内容摘要可以通过一个功能增强的、等同于“du –s”的命令 mfsdirinfo 来显示。mfsdirinfo 可以显示 MFS 的具体信息。

例如：

```
[root@www mfs]# mfsdirinfo /mnt/mfs-test/test/
 inodes:                       15
  directories:                  4
```

```
 files:                          8
chunks:                          6
length:                     270604
size:                       620544
realsize:                  1170432
```

上述内容摘要显示了目录、文件、分区的数目，以及整个目录占用磁盘空间的情况，其中：

- length，表示文件大小的总和。
- size，表示块长度总和。
- realsize，表示磁盘空间的使用，包括所有的副本。

8.4.3　为垃圾箱设定隔离时间

删除的文件存放在“垃圾箱（trash bin）”的时间就是隔离时间（quarantine time），这个时间可以用 mfsgettrashtime 命令来验证，也可以用 mfssettrashtime 命令来设置。例如：

```
[root@www mfs]# mfsgettrashtime /mnt/mfs-test/test1
/mnt/mfs-test/test1: 604800
[root@www mfs]# mfssettrashtime 0 /mnt/mfs-test/test1
/mnt/mfs-test/test1: 0
[root@www mfs]# mfsgettrashtime /mnt/mfs-test/test1
/mnt/mfs-test/test1: 0
```

这个命令工具中的递归选项“-r”，可以对整个目录树操作。例如：

```
[root@www mfs]# mfsgettrashtime -r /mnt/mfs-test/test2
/mnt/mfs-test/test2:
 files with trashtime                0 :                    36
 directories with trashtime     604800 :                     1
[root@www mfs]# mfssettrashtime -r 1209600 /mnt/mfs-test/test2
/mnt/mfs-test/test2:
 inodes with trashtime changed:                        37
 inodes with trashtime not changed:                     0
 inodes with permission denied:                         0
[root@www mfs]# mfsgettrashtime -r /mnt/mfs-test/test2
/mnt/mfs-test/test2:
 files with trashtime          1209600 :                    36
 directories with trashtime    1209600 :                     1
```

以上代码中的时间单位是秒。与文件被存储的份数一样，为一个目录设定存放时间后，在此目录下新创建的文件和目录就可以继承这个设置了。数字 0 意味着一个文件被删除后，会立即彻底删除，不可能再恢复了。

删除的文件可以通过一个单独安装的 mfsmeta 辅助文件系统来恢复。这个文件系统包含了目录 trash（含有仍然可以被还原的删除文件的信息）和目录 trash/undel（用于获取文件）。只有管理员有权限访问 mfsmeta 辅助文件系统（管理员用户的系统 uid 为 0，通常是 root 用户）。

下面这段代码演示了一个文件被删除后，在 mfsmeta 辅助文件系统中仍然可以找到。

```
[root@www mfs]# mfssettrashtime 3600 /mnt/mfs-test/test1
/mnt/mfs-test/test1: 3600
[root@www mfs]# rm /mnt/mfs-test/test1
[root@www mfs]# ls  /mnt/mfs-test/test1
ls: /mnt/mfs-test/test1: No such file or directory
[root@www mfs]# ls -l /mnt/mfs-test-meta/trash/*test1
-rw-r--r-- 1 user  users  1 2007-08-09 15:23 /mnt/mfs-test-meta/trash/00013BC7|test1
```

被删文件的文件名在“垃圾箱”目录里还是可见的，文件名由一个 8 位十六进制数的 i-node 和被删文件的文件名组成，在文件名和 i-node 之间不用“/”，而是以“|”替代。如果一个文件名的长度超过操作系统的限制（通常是 255 字符），那么超出部分将被删除。从挂载点起全路径的文件名被删除的文件仍然可以被读写。

需要注意的是：被删除的文件在使用全路径文件名（注意文件名是两部分）时，一定要用单引号引起来。例如：

```
[root@www mfs]# cat '/mnt/mfs-test-meta/trash/00013BC7|test1'
test1
[root@www mfs]# echo 'test/test2' > '/mnt/mfs-test-meta/trash/00013BC7|test1'
[root@www mfs]# cat '/mnt/mfs-test-meta/trash/00013BC7|test1'
test/test2
```

移动这个文件到 trash/undel 子目录下，将会使原始的文件恢复到正确的 MFS 文件系统原来的路径下（如果路径没有改变）。例如：

```
[root@www ~]# cd /mnt/mfs
[root@www mfs]# ll dgg
-rw-r--r--  1 root root 8 Jan 13 08:45 dgg
[root@www mfs]# rm -f dgg
[root@www mfs]# ll dgg
ls: dgg: No such file or directory
[root@www mfs]#cd /mnt/mfsmeta/trash
[root@www trash]# ls
0000000B|dgg    00000047|f1    undel
[root@www trash]# mv '/mnt/mfsmeta/trash/0000000B|dgg' ./undel/
[root@www trash]# ls
 undel 00000047|f1
[root@www trash]# cd /mnt/mfs
[root@www mfs]# ll dgg
-rw-r--r--  1 root root 8 Jan 13 08:45 dgg
```

注意，如果在同一路径下有个新的同名文件，那么恢复是不会成功的。

从“垃圾箱”中删除文件的结果是释放之前被它占用的空间（删除有延迟，数据被异步删除）。在从“垃圾箱”中删除后，该文件就不可能恢复了。

可以通过 mfssetgoal 工具来改变文件的副本数，也可以通过 mfssettrashtime 工具来改变文件存储在“垃圾箱”中的时间。

在 mfsmeta 目录里，除了 trash 和 trash/undel 两个目录外，还有第三个目录 reserved，该目录内有已经被删除的文件，这些文件又一直打开着。在用户关闭了这些被打开的文件后，reserved 目录中的文件将被彻底删除，文件中的数据也立即被删除。在 reserved 目录中，文件的命名方法与 trash 目录中的一样，但是不能进行查看或恢复文件的操作。

8.4.4　快照

MFS 系统的另一个特征是利用 mfsmakesnapshot 工具给文件或者目录树做快照（snapshot）。例如：

```
[root@www ~]# mfsmakesnapshot source ... destination
```

其中，source 是源文件路径或者目录，destination 是快照文件路径或者目录。需要注意的是，destination 路径必须在 MFS 文件系统下，即 source 与 destination 路径必须都在 MFS 体系下，不能将快照放到 MFS 文件系统之外的其他文件系统下。

mfsmakesnapshot 是在一次执行中整合了一个或一组文件的副本，而且对这些文件的源文件进行任何修改都不会影响源文件的快照，就是说任何对源文件的操作，如写入源文件，将不会修改副本（反之亦然）。

文件快照可以利用 mfsappendchunks 实现，在 MooseFS 1.5 版本中也可以通过 mfssnapshot 实现，作为选择，二者都可以用。例如：

```
[root@www ~]# mfsappendchunks destination-file source-file ...
```

当有多个源文件时，它们的快照会被加入同一个目标文件中。通过对快照的测试，可以发现快照的本质。

- 对一个 MFS 系统下的文件做快照后，查看两个文件的块信息，它们是同一个块。接着，把原文件删除，删除原文件后（最初可以在回收站中看到该文件，但一段时间后，回收站中的文件也被彻底删除了），快照文件仍然存在，并且可以访问。使用 mfsfileinfo 查看，发现仍然是原来的块。
- 对一个文件做快照后，查看两个文件的块信息，发现是同一个块。把原文件修改后，发现原文件使用的块信息变了，即使用了一个新块。而快照文件仍然使用原来的块，保持文件内容不变。

8.4.5　MFS 的其他命令

文件或目录的额外属性（noowner、noattrcache 和 noentrycache），可以通过 MFS 提供的命令（如 mfsgeteattr、mfsseteattr、mfsdeleattr 等）检查、设置和删除，其行为类似于 mfsgetgoal/mfssetgoal 或者是 mfsgettrashtime/mfssettrashtime。

8.5 维护 MFS

维护 MFS，最重要的是维护元数据服务器，而元数据服务器最重要的目录为 /usr/local/mfs/var/mfs，MFS 数据的存储、修改、更新等操作变化都会记录在这个目录的某个文件中，因此只要保证这个目录的数据安全，就能保证整个 MFS 文件系统的安全性和可靠性。

/usr/local/mfs/var/mfs 目录下的数据由两部分组成：一部分是元数据服务器的改变日志文件，文件名称类似于 changelog.*.mfs；另一部分是元数据文件 metadata.mfs，运行 mfsmaster 时该文件会被命名为 metadata.mfs.back。只要保证了这两部分数据的安全，即使元数据服务器遭到致命的破坏，也可以通过备份的元数据文件重新部署一套元数据服务器。

8.5.1 启动 MFS 集群

最安全的启动 MFS 集群（避免任何读或写的错误数据或类似的问题）的步骤如下：

1）启动 mfsmaster 进程。

2）启动所有的 mfschunkserver 进程。

3）启动 mfsmetalogger 进程（如果配置了 mfsmetalogger）。

当所有的数据存储服务器连接到 MFS 管理服务器后，任何数目的客户端都可以利用 mfsmount 去挂接共享出来的文件系统（可以通过检查管理服务器的日志或 CGI 监视器来查看所有的数据存储服务器是否被连接）。

8.5.2 停止 MFS 集群

要安全地停止 MFS 集群，按照如下步骤进行操作：

1）在所有的客户端卸载 MFS 文件系统（利用 umount 命令或者其他等效的命令）。

2）利用“mfschunkserver –s”命令停止数据存储服务器进程。

3）利用“mfsmetalogger –s”命令停止元数据日志服务器进程。

4）利用“mfsmaster –s”命令停止管理服务器进程。

8.5.3 MFS 数据存储服务器的维护

假如每个文件的 goal（目标）都不小于 2，并且没有 under-goal 文件（这些可以通过“mfsgetgoal –r”和 mfsdirinfo 命令来检查），那么一个数据存储服务器在任何时刻都可以停止或重新启动。以后每当需要停止或者重新启动另一个数据存储服务器的时候，要确定之前的数据存储服务器被连接，而且没有 under-goal chunks。

8.5.4　MFS 元数据的备份

通常元数据由两部分数据组成：

- 主要元数据文件 metadata.mfs，在 MFS 的管理服务器 master 运行时会被命名为 metadata.mfs.back。
- 元数据改变日志 changelog.*.mfs，存储过去 N 小时内的文件改变（N 的数值是由 BACK_LOGS 参数设置的，参数的设置在 mfschunkserver.cfg 配置文件中进行）。

主要的元数据文件需要定期备份，备份的频率取决于多少小时改变日志的储存。元数据改变日志应该实时地自动复制。自从 MooseFS 1.6.5 开始，这两项任务都是由元数据日志服务器守护进程完成的。

8.5.5　MFS 管理服务器的恢复

一旦管理服务器崩溃（例如由主机或电源失败导致的），需要最后一个元数据改变日志 changelog 和主要元数据文件 metadata.mfs，这个操作可以通过 mfsmetarestore 工具来完成。最简单的方法如下：

```
mfsmetarestore -a
```

执行此命令后，默认会在 /usr/local/mfs/var/mfs 目录中自动寻找需要的改变日志文件和主要元数据文件。注意，mfsmetarestore 命令在恢复时自动查找的是 metadata.mfs.back 文件，而不是 metadata.mfs 文件，如果找不到 metadata.mfs.back 文件，会继续查找是否存在 metadata_ml.mfs.back 文件，如果没有，将提示恢复错误。

如果管理服务器的数据被存储在 MFS 编译指定地点外的路径，则要利用“-d”参数指定使用。如：

```
mfsmetarestore -a -d /storage/mfsmaster
```

8.5.6　从备份恢复 MFS 管理服务器

为了从备份中恢复一个管理服务器，需要按以下步骤进行：

1）安装一个管理服务器。

2）利用同样的配置来配置这台管理服务器（利用备份找回 mfsmaster.cfg），可见配置文件也是需要备份的。

3）找回 metadata.mfs.back 文件，可以从备份服务器中找，也可以从元数据日志服务器中找（如果启动了元数据日志服务），然后把 metadata.mfs.back 放入 data 目录中，一般为 ${prefix}/var/mfs。

4）从在管理服务器宕机之前的任何运行元数据日志服务的服务器上复制最后一个 changelog.*.mfs 文件，放入管理服务器的数据目录。

5）利用 mfsmetarestore 命令合并元数据改变日志。这时可以采用自动恢复模式，命令如下：

```
mfsmetarestore -a
```

也可以利用非自动化恢复模式，在 ${prefix}/var/mfs 目录下执行如下命令：

```
mfsmetarestore -m metadata.mfs.back -o metadata.mfs changelog_ml.*.mfs
```

8.6 通过冗余实现失败防护的解决方案

对一个没有内置失效功能的管理服务器实现冗余功能是正确的选择。这个话题（subject）对应用业务系统来说非常关键，因为冗余在生产系统中是非常重要的，一个没有冗余的 MFS 应用是不能用在生产系统中的，这是起码的要求。

使用通用地址冗余协议（CARP）就可以解决失败防护的需求，CARP 允许同一个 LAN 中的两台机器用同一个 IP，一个作为管理服务器，另一个是备份服务器。因此可以在 CARP 的网卡接口上设置管理服务器的 IP，并且配置为 MFS 的主要（main）管理服务器。在备份服务器上也要安装管理服务器，但是不要运行它。

MooseFS 1.6.5 及其以上版本中包含了一个新的程序——mfsmetalogger，这个程序可以运行在任何机器上。这个程序每隔数小时（默认为 24 小时）从管理服务器上获得一次完整的元数据，并且在现有的基础上形成一个彻底改变的日志。

如果运行的是一个比 1.6.5 早的版本，那么就需要设置几个简单的脚本并且依据系统的计划任务运行（如每小时运行一次）。脚本的主要内容是从主管理服务器下“PREFIX/var/mfs/metadata.mfs.back”备份元数据文件。

另外，还需要持续地运行一个额外的脚本来检测 CARP 网卡接口状态，如果该接口在一个 MASTER 模式进行了改变，那么将会从任何一台数据存储服务器（仅仅通过使用 SCP）上获得两个或三个最新的改变日志文件，之后就可以运行“mfsmetarestore”命令并最终切换到新的管理服务器上。这个切换时间大约有几秒钟，随着切换的进行，各台数据存储服务器将重新连接到新的管理服务器，新的管理服务器将会在一分钟内恢复全部功能（读和写）。

在 MFS 新的版本中，开发者计划加一个让管理服务器运行在只读模式下的选项，这对运行备份服务器将是十分有好处的，它可以确保使系统潜在的两个管理服务器同步，并且合并发生在备份服务器的变化到主管理服务器上。

8.7 本章小结

本章主要讲述了分布式文件存储系统 MFS 的结构、安装配置和基本的管理维护。在安装配置方面，不但详细介绍了配置方法，而且讲述了安装配置过程中容易出现的问题和经验

技巧；在管理维护方面，详细介绍了 MFS 如何启动和关闭，以及如何进行备份和恢复，还简单说明了 MFS 常用的几个文件系统命令；最后还介绍了通过冗余实现 MFS 失败防护的解决方案，并分享了 MFS 在生产环境中的使用经验和技巧。

作为一个开源的分布式文件存储系统，MFS 完成的功能决不逊色于专业的存储系统，而且还可以实现在线扩容。随着 MFS 版本和功能的升级，相信 MFS 的应用会越来越广泛！

第 4 篇

运维监控与性能优化篇

第 9 章　运维监控利器 Nagios

本章主要介绍开源监控软件 Nagios 的安装配置和使用技巧。Nagios 是系统管理人员和运维监控人员必需的工具之一，利用 Nagios 可以监控本地或远程主机资源，如磁盘空间、系统负载等情况，也可以监控各种应用服务，例如 HTTP 服务、FTP 服务等。当主机或服务出现故障时，Nagios 还可以通过邮件、手机短信等形式在第一时间通知系统维护人员。作为一名系统管理人员，一定不要错过这个功能强大的开源监控软件。

9.1　Nagios 综述

作为一名运维人员或系统管理员，难免会遇到主机或服务异常的情况。遭遇故障并不可怕，可怕的是在出现故障后，系统管理人员并不知道。由于没有及时发现故障，不但解决问题时存在困难，而且可能带来很大的损失。因此，一个能完成对主机或服务进行检测的自动化工具对于运维人员或系统管理员来说非常重要。Nagios 就是一个这样的开源管理软件，通过 Nagios 可以轻松实现对远程主机、服务以及网络的全面监控。

9.1.1　什么是 Nagios

Nagios 是一款 Linux 上成熟的监视系统运行状态和网络信息的开源 IT 基础设施监视系统。Nagios 能监视所指定的本地或远程主机及服务，同时提供异常通知、事件处理等功能。与商业 IT 管理系统，如 IBM Tivoli、HP OpenView/Operations 等相比，Nagios 具有成本低廉、结构简单、可维护性强等诸多优点，越来越受 IT 运维人员和系统管理员的青睐。

Nagios 可运行在 Linux 和 UNIX 平台上，同时提供一个可选的基于浏览器的 Web 界面，以方便系统管理人员查看系统的运行状态、网络状态、各种系统问题及日志异常等。

9.1.2　Nagios 的结构与特点

从结构上讲，Nagios 可分为核心和插件两个部分。Nagios 的核心部分只提供了很少的监控功能，因此要搭建一个完善的 IT 监控管理系统，用户还需要为 Nagios 安装相应的插件，这些插件可以从 Nagios 官方网站下载，也可以根据实际要求编写。

Nagios 的主要功能特点如下：

- 监视本地或者远程主机资源（内存、进程、磁盘等）。

- 监视网络服务资源（HTTP、PING、FTP、SMTP、POP3 等）。
- 允许用户编写自己的插件来监控特定的服务。
- 当被监控对象出现异常时，可以通过邮件、短信等方式通知管理人员。
- 可以事先定义事件处理程序，当主机或者服务出现故障时自动调用指定的处理程序。
- 可以通过 Web 界面来监控各个主机或服务的运行状态。

9.2　Nagios 的安装与配置

9.2.1　安装 Nagios

1. 安装前的准备

（1）创建 Nagios 用户和用户组

将 Nagios 进程的运行用户和组设置为 nagios，并且将 nagios 的主程序目录设置为 nagios，以保证系统的安全（当然设置为 root 用户也是可以的，但是不建议这么做）。基本操作如下：

```
[root@localhost ~]#useradd -s /sbin/nologin nagios
[root@localhost ~]#mkdir /usr/local/nagios
[root@localhost ~]#chown -R  nagios.nagios /usr/local/nagios
```

（2）开启系统的 sendmail 服务

在 Nagios 监控服务器上开启 sendmail 服务的主要作用是让 Nagios 在检测到故障时可以发送报警邮件。目前几乎所有的 Linux 发行版本都默认自带了 sendmail 服务，所以，在安装系统时只需开启 sendmail 服务即可，并不需要在 sendmail 上进行任何配置。

2. 编译安装 Nagios

所有准备工作完成后，开始编译安装 Nagios。过程如下：

```
[root@localhost ~]# tar -zxvf nagios-3.2.0.tar.gz
[root@localhost ~]# cd nagios-3.2.0
[root@localhost nagios-3.2.0]#./configure --prefix=/usr/local/nagios
          #指定 Nagios 的安装目录，这里指定将 Nagios 安装到 /usr/local/nagios 目录下
[root@localhost nagios-3.2.0]#make all
[root@localhost nagios-3.2.0]#make install
          #通过 make install 命令来安装 Nagios 主程序的 CGI 和 HTML 文件
[root@localhost nagios-3.2.0]# make install-init
          #通过 make install-init 命令可以在 /etc/rc.d/init.d 目录下创建 Nagios 启动脚本
[root@localhost nagios-3.2.0]# make install-commandmode
          #通过 make install-commandmode 命令来配置目录权限
[root@localhost nagios-3.2.0]# make install-config
          #make install-cofig 命令用来安装 Nagios 示例配置文件，这里的安装路径是 /usr/
          #local/nagios/etc
```

设置开机自启动。

```
[root@localhost etc]# chkconfig --add nagios
[root@localhost etc]# chkconfig --level 35 nagios on
[root@localhost etc]# chkconfig --list nagios
nagios          0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

Nagios 各个目录名称及用途说明如表 9-1 所示。

表 9-1 Nagios 安装目录的名称及用途

目录名称	用 途
bin	Nagios 可执行程序所在目录
etc	Nagios 配置文件所在目录
sbin	Nagios CGI 文件所在目录，也就是执行外部命令所需文件所在的目录
Share	Nagios 网页文件所在的目录
libexec	Nagios 外部插件所在目录
var	Nagios 日志文件、lock 等文件所在的目录
var/archives	Nagios 日志自动归档目录
var/rw	用来存放外部命令文件的目录

3. 安装 Nagios 插件

Nagios 提供的各种监控功能基本是通过插件来完成的，而 Nagios 核心只提供了很少的监控功能，因此安装插件是必须的。Nagios 的插件可以在 www.nagios.org 下载到，这里下载的是 nagios-plugins-1.4.14。其实插件版本与 Nagios 版本的关联并不大，不一定非要用 nagios-plugins-1.4.14 这个版本。接着上传对应的 nagios-plugins-1.4.14.tar.gz 包到服务器，解压缩并且安装。过程如下：

```
[root@localhost nagios]#tar -zxvf  nagios-plugins-1.4.14.tar.gz
[root@localhost nagios]#cd  nagios-plugins-1.4.14
[root@localhost nagios-plugins-1.4.14]#./configure --prefix=/usr/local/nagios
[root@localhost nagios-plugins-1.4.14]# make
[root@localhost nagios-plugins-1.4.14]# make install
```

这样，安装就完成了。这里需要说明的是，插件的安装路径最好和 Nagios 安装路径一致，这样安装完插件后会在 Nagios 主程序目录（即 /usr/local/nagios 下的 libexec 目录）下生成很多可执行文件，这些就是 Nagios 所需要的插件。

4. 安装 Nagios 汉化插件

对于英文水平不高的用户，还可以为 Nagios 安装汉化插件，可以从 http://sourceforge.net/projects/nagios-cn/files/ 下载对应 Nagios 版本的汉化插件。这里下载的是 nagios-cn-3.2.0.tar.bz2。接着开始安装编译 Nagios 插件，过程如下：

```
[root@localhost ~]#tar jxvf nagios-cn-3.2.0.tar.bz2
[root@localhost nagios-cn-3.2.0]#cd nagios-cn-3.2.0
```

```
[root@localhost nagios-cn-3.2.0]#./configure
[root@localhost nagios-cn-3.2.0]#make all
[root@localhost nagios-cn-3.2.0]#make install
```

5. 安装与配置 Apache

Apache 不是安装 Nagios 所必需的，但是 Nagios 提供了 Web 监控界面，通过 Web 监控界面可以清晰地看到被监控的主机和资源的运行状态，因此，安装一个 Web 服务是很必要的。可选的 Web 服务器有 Apache、Nginx 等，这里选择 Apache，选取的版本为 apache2.0.63。

Apache 的安装非常简单，需要注意的是，Nagios 在 nagios 3.1.x 版本以后，配置 Web 监控界面时需要 PHP 的支持。这里下载的 Nagios 版本为 nagios 3.2.0，因此在编译安装完 Apache 后，还需要编译 PHP 模块，这里选取的 PHP 版本为 php 5.3.2。操作过程如下。

（1）安装 Apache 与 PHP

首先安装 Apache，步骤如下：

```
[root@nagiosserver ~]# tar zxvf httpd-2.0.63.tar.gz
[root@nagiosserver ~]#cd httpd-2.0.63
[root@nagiosserver ~]#./configure  --prefix=/usr/local/apache2
[root@nagiosserver ~]#make
[root@nagiosserver ~]#make install
```

接着安装 PHP，步骤如下：

```
[root@nagiosserver ~]# tar zxvf php-5.3.2.tar.gz
[root@nagiosserver ~]#cd php-5.3.2
[root@nagiosserver ~]#./configure  --prefix=/usr/local/php \
>--with-apxs2=/usr/local/apache2/bin/apxs
[root@nagiosserver ~]#make
[root@nagiosserver ~]#make install
```

从安装步骤可知，Apache 的安装路径为 /usr/local/apache2，而 PHP 的安装路径为 /usr/local/php。

（2）配置 Apache

首先在 Apache 配置文件 /usr/local/apache2/conf/httpd.conf 中修改 Apache 进程的启动用户为 Nagios，即找到：

```
User nobody
Group #-1
```

修改为：

```
User nagios
Group nagios
```

然后找到：

```
DirectoryIndex index.html index.html.var
```

修改为：

```
DirectoryIndex index.html index.php
```

接着增加如下内容：

```
AddType application/x-httpd-php .php
```

安全起见，一般要求必须经过授权才能访问 Nagios 的 Web 监控界面，因此需要增加验证配置，即在 httpd.conf 文件的最后添加如下信息：

```
#setting for nagios
ScriptAlias /nagios/cgi-bin "/usr/local/nagios/sbin"
<Directory "/usr/local/nagios/sbin">
    AuthType Basic
    Options ExecCGI
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthUserFile /usr/local/nagios/etc/htpasswd
    Require valid-user
</Directory>

Alias /nagios "/usr/local/nagios/share"
<Directory "/usr/local/nagios/share">
    AuthType Basic
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "nagios Access"
    AuthUserFile /usr/local/nagios/etc/htpasswd
    Require valid-user
</Directory>
```

（3）创建 Apache 目录验证文件

在上面的配置中，指定了目录验证文件 htpasswd，下面创建这样一个文件：

```
[root@localhost nagios]#/usr/local/apache2/bin/htpasswd \
>-c /usr/local/nagios/etc/htpasswd ixdba
New password:（输入密码）
Re-type new password:（再输入一次密码）
Adding password for user ixdba
```

这样就在 /usr/local/nagios/etc 目录下创建了一个 htpasswd 验证文件，对应的用户为 ixdba。当通过 http://ip/nagios/ 访问 Web 监控界面时就需要输入用户名和密码了。

（4）启动 Apache 服务

通过以下命令启动 Apache 服务：

```
[root@ nagiosserver ~]#/usr/local/apache2/bin/apachectl  start
```

启动 Apache 后，可以看到 Nagios 的默认 Web 监控界面。如果安装的是 Nagios 的中文包，看到的应该是中文界面。

9.2.2　配置 Nagios

Nagios 主要用于监控一台或者多台本地主机及远程主机的各种信息，包括本机资源及对外的服务等。默认的 Nagios 配置没有任何监控内容，仅是一些模板文件。下面通过理论与实践相结合的方式详细介绍如何搭建一个完善的 Nagios 监控系统。

1. 默认配置文件介绍

Nagios 安装完毕后，默认的配置文件在 /usr/local/nagios/etc 目录下，每个文件或目录的用途如表 9-2 所示。

表 9-2　Nagios 配置文件及用途

文件名或目录名	用　途
cgi.cfg	控制 CGI 访问的配置文件
nagios.cfg	Nagios 主配置文件
resource.cfg	变量定义文件，又称为资源文件，在此文件中定义变量，以便由其他配置文件引用，如 $USER1$
objects	objects 是一个目录，在此目录下有很多配置文件模板，用于定义 Nagios 对象
objects/commands.cfg	命令定义配置文件，其中定义的命令可以被其他配置文件引用
objects/contacts.cfg	定义联系人和联系人组的配置文件
objects/localhost.cfg	定义监控本地主机的配置文件
objects/printer.cfg	定义监控打印机的一个配置文件模板，默认没有启用此文件
objects/switch.cfg	监控路由器的一个配置文件模板，默认没有启用此文件
objects/templates.cfg	定义主机和服务的一个模板配置文件，可以在其他配置文件中引用
objects/timeperiods.cfg	定义 Nagios 监控时间段的配置文件
objects/windows.cfg	监控 Windows 主机的一个配置文件模板，默认没有启用此文件

Nagios 在配置方面非常灵活，默认的配置文件并不是必需的。可以使用这些默认的配置文件，也可以创建自己的配置文件，然后在主配置文件 nagios.cfg 中引用即可。

2. 配置文件之间的关系

Nagios 的配置过程涉及的几个定义有：主机、主机组、服务、服务组、联系人、联系人组、监控时间和监控命令等，从这些定义可以看出，Nagios 的各个配置文件之间是互为关联、彼此引用的。成功配置一台 Nagios 监控系统，必须弄清楚每个配置文件之间依赖与被依赖的关系，最重要的 4 点是：第一要定义监控哪些主机、主机组、服务和服务组，第二要定义这个监控要通过什么命令实现，第三要定义监控的时间段，最后要定义主机或服务出现问题时要通知的联系人和联系人组。

清楚了 Nagios 的配置重点和各个配置文件之间的依赖关系后，配置 Nagios 就变得非常容易了。下面开始详细介绍如何配置 Nagios。

3. 配置 Nagios

为了能更清楚地说明问题，同时为了方便维护，建议为 Nagios 各个定义对象创建独立的配置文件：创建 hosts.cfg 文件定义主机和主机组，创建 services.cfg 文件定义服务，用默认的 contacts.cfg 文件定义联系人和联系人组，用默认的 commands.cfg 文件定义命令，用默认的 timeperiods.cfg 文件定义监控时间段，将默认的 templates.cfg 文件作为资源引用文件。下面分别介绍如下。

（1）templates.cfg 文件

Nagios 主要用于监控主机资源及服务（在 Nagios 配置中被称为对象）。为了不重复定义一些监控对象，Nagios 引入了一个模板配置文件，将一些共性的属性定义成模板，以便多次引用，这就是 templates.cfg 的作用。templates.cfg 文件的内容如下：

```
define contact{
    name    generic-contact
            # 联系人名称
    service_notification_period     24x7
            # 当服务出现异常时，# 发送通知的时间段，这个时间段“24x7”在 timeperiods.
            # cfg 文件中定义
    host_notification_period        24x7
            # 当主机出现异常时，发送通知的时间段，这个时间段“24x7" 在 timeperiods.cfg 文件中定义
service_notification_options    w,u,c,r
            # 定义的是“通知可以被发出的情况”，w 即 warn，表示警告状态；u 即 unknown，表示不明状态；
            # c 即 criticle，表示紧急状态；r 即 recover，表示恢复状态。也就是在服务出现警告状态、
            # 未知状态、紧急状态和重新恢复状态时都发送通知给使用者

    host_notification_options       d,u,r
            # 定义主机在什么状态下需要发送通知给使用者，d 即 down，表示宕机状态；u 即 unreachable，
            # 表示不可到达状态；r 即 recovery，表示重新恢复状态
    service_notification_commands   notify-service-by-email
            # #服务故障时，发送通知的方式，可以是邮件和短信，这里的发送方式是邮件，其中“notify-
            # service-by-email”在 commands.cfg 文件中定义
    host_notification_commands      notify-host-by-email
            # 主机故障时，发送通知的方式，可以是邮件和短信，这里发送的方式是邮件，其中“notify-
            # host-by-email”在 commands.cfg 文件中定义
    register                        0
    }

define host{
    name                            generic-host
            # 主机名称，这里的主机名并不是直接对应真正机器的主机名，乃是对应在主机配置文件中设定的主机名
    notifications_enabled           1
    event_handler_enabled           1
    flap_detection_enabled          1
    failure_prediction_enabled      1
    process_perf_data               1
            # 其值可以为 0 或 1，其作用为是否启用 Nagios 的数据输出功能。如果将此项赋值为 1，那
```

```
        # 么 Nagios 就会将收集的数据写入某个文件中，以备提取
    retain_status_information       1
    retain_nonstatus_information    1
    notification_period             24x7
        # 指定“发送通知”的时间段，也就是可以在什么时候发送通知给使用者
    register                        0
    }

define host{
    name                            linux-server
        # 主机名称
    use                             generic-host
        # use 表示引用，也就是将主机 generic- host 的所有属性引用到 linux-server 中。
        # 在配置 Nagios 过程中，很多情况下会用到引用
    check_period                    24x7
        # 这里的 check_period 告诉 Nagios 检查主机的时间段
    check_interval                  5
        #  Nagios 对主机的检查时间间隔，这里是 5 分钟
    retry_interval                  1
        # 重试检查时间间隔，单位是分钟
    max_check_attempts              10
        # Nagios 对主机的最大检查次数，也就是 Nagios 在检查过程中发现某主机异常时，并不马上判断
        # 为异常状况，而是多试几次，因为可能只是一时网络太拥挤，或是一些其他原因，使主机受到了一
        # 点影响，这里的 10 就是至少试 10 次的意思
    check_command                   check-host-alive
        # 指定检查主机状态的命令，其中“check-host-alive”在 commands.cfg 文件中定义
    notification_period             workhours
        # 主机故障时，发送通知的时间范围，其中“workhours”在 timeperiods.cfg 中进行了定义

    notification_interval           120
        # 在主机出现异常后，故障一直没有解决，Nagios 再次对使用者发出通知的时间。单位是分钟。如果
        # 觉得所有的事件只需要一次通知就够了，可以把这个选项设为 0
    notification_options            d,u,r
        # 定义主机在什么状态下可以发送通知给使用者，d 即 down，表示宕机状态；u 即 unreachable，
        # 表示不可到达状态；r 即 recovery，表示重新恢复状态
    contact_groups          admins
        # 指定联系人组，这个“admins”在 contacts.cfg 文件中定义
    register                        0
    }

define service{
    name                            generic-service
        # 定义一个服务名称
    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
            # 这里的 check_period 告诉 Nagios 检查服务的时间段
    max_check_attempts              3
            # Nagios 对服务的最大检查次数
    normal_check_interval           10
            # 此选项用来设置服务检查时间间隔，也就是说，Nagios 这一次检查与下一次检查所隔的时间，
            # 这里是 10 分钟
    retry_check_interval            2
            # 重试检查时间间隔，单位是分钟
    contact_groups                  admins
            # 指定联系人组
    notification_options            w,u,c,r
            # 定义“通知可以被发出的情况”
    notification_interval           60
            # 在服务出现异常后，故障一直没有解决，Nagios 再次向使用者发出通知的时间。单位是分钟
    notification_period             24x7
            # 指定“发送通知”的时间段，也就是可以在什么时候发送通知给使用者
     register                        0
    }
```

（2）resource.cfg 文件

resource.cfg 是 Nagios 的变量定义文件，文件内容只有一行：

```
$USER1$=/usr/local/nagios/libexec
```

其中，变量 $USER1$ 指定了安装 Nagios 插件的路径，如果把插件安装在其他路径，只需在这里进行修改即可。需要注意的是，变量必须先定义，然后才能在其他配置文件中进行引用。

（3）commands.cfg 文件

此文件在默认情况下是存在的，无需修改即可使用。当然，如果有新的命令需要加入，在此文件中进行添加即可。这里并未列出文件的所有内容，仅介绍了配置中用到的一些命令。

❑ notify-host-by-email 命令的定义如下：

```
define command{
    command_name    notify-host-by-email
            # 命令名称，即定义了一个主机异常时发送邮件的命令
    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 "** $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is $HOSTSTATE$ **"
        $CONTACTEMAIL$
            # 命令的具体执行方式，“-H $HOSTADDRESS$”是定义目标主机的地址，这个地址在 hosts.
            # cfg 文件中定义
    }
            # 下面是 notify-host-by-email 命令的定义
define command{
    command_name    notify-service-by-email  # 命令名称，即定义了一个服务异常时发送邮件的命令
    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 "** $NOTIFICATIONTYPE$
    Service Alert: $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **"
    $CONTACTEMAIL$
  }
```

❑ check-host-alive 命令的定义如下：

```
define command{
    command_name     check-host-alive        # 命令名称，用来检测主机状态
   command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w 3000.0,80% -c 5000.0,100% -p 5
# 这里的变量 $USER1$ 在 resource.cfg 文件中进行定义，即 $USER1$=/usr/local/nagios/libexec,
# 因此 check_ping 的完整路径为 /usr/local/nagios/libexec/check_ping "-w 3000.0,80%" 中
# "-w" 表明后面的一对值对应的是 "WARNING" 状态，"80%" 是其临界值 "-c 5000.0,100%" 中 "-c"
# 表明后面的一对值对应的是 "CRITICAL"，"100%" 是其临界值 "-p 1" 说明每次探测发送一个包
    }
```

❑ check-ftp 命令的定义如下：

```
define command{
    command_name     check_ftp
    command_line     $USER1$/check_ftp -H $HOSTADDRESS$ $ARG1$
    #$ARG1$ 是指在调用这个命令的时候，命令后面的第一个参数
    }
```

❑ check_http 命令的定义如下：

```
define command{
    command_name     check_http
    command_line     $USER1$/check_http -I $HOSTADDRESS$ $ARG1$
    }
```

❑ check_ssh 命令的定义如下：

```
define command{
    command_name     check_ssh
    command_line     $USER1$/check_ssh $ARG1$ $HOSTADDRESS$
    }
```

❑ check_ping 命令的定义如下：

```
define command{
    command_name     check_ping
    command_line     $USER1$/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$ -p 5
    }
```

❑ check_tcp 命令的定义如下：

```
define command{
    command_name     check_tcp
    command_line     $USER1$/check_tcp -H $HOSTADDRESS$ -p $ARG1$ $ARG2$
    }
```

（4）hosts.cfg 文件

此文件在默认情况下不存在，需要手动创建。hosts.cfg 主要用来指定被监控的主机地址

及相关属性信息。一个配置好的示例如下：

```
define host{
    use                     linux-server        # 引用主机 linux-server 的属性信息，linux-server
                                                # 主机在 templates.cfg 文件中进行定义
    host_name               web                 # 主机名
    alias                   ixdba-web           # 主机别名
    address                 192.168.12.251      # 被监控的主机地址，这个地址可以
                                                # 是 IP，也可以是域名
    }

define host{
    use                     linux-server
    host_name               mysql
    alias                   ixdba-mysql
    address                 192.168.12.26
    }

define hostgroup{                               # 定义一个主机组
hostgroup_name  sa-servers                      # 主机组名称，可以随意指定
alias           sa servers                      # 主机组别名
members         web,mysql                       # 主机组成员，其中“web”、“mysql”就是
                                                # 上面定义的两个主机
}
```

此文件创建了 192.168.12.251 和 192.168.12.26 两个远程主机和一个主机组。如果要创建更多被远程监控的主机和主机组，按照相同的格式分别手动创建即可。

（5）services.cfg 文件

此文件在默认情况下也不存在，需要手动创建。services.cfg 文件主要用于定义监控的服务和主机资源，例如监控 HTTP 服务、FTP 服务、主机磁盘空间、主机系统负载等。

一个配置好的示例如下：

```
##################  ixdba   Web  ######################
define service{
    use                             local-service
            # 引用 local-service 服务的属性值，local-service 在 templates.cfg 文件中进行了定义
    host_name                       web
            # 指定要监控哪个主机上的服务，“web”在 hosts.cfg 文件中进行了定义
    service_description             PING
            # 对监控服务内容的描述，以供维护人员参考
    check_command                   check_ping!100.0,20%!500.0,60%
            # 指定检查的命令，check_ping 命令在 commands.cfg 中定义，后跟两个参数，命令与参数
            # 间用 ! 分割
    }

define service{
    use                             local-service
    host_name                       web
    service_description                  SSH
    check_command                  check_ssh
            # check_ssh 命令也在 commands.cfg 中定义
    }
```

```
define service{
    use                             local-service
    host_name                       web
    service_description             SSHD
    check_command                   check_tcp!22
    }

define service{
    use                             local-service
    host_name                       web
    service_description             http
    check_command                   check_http
    }

################################## MySQL ######################

define service{
    use                             local-service
    host_name                       mysql
    service_description             PING
    check_command                   check_ping!100.0,20%!500.0,60%
    }

define service{
    use                             local-service
    host_name                       mysql
    service_description             SSH
    check_command                   check_ssh
    }

define service{
    use                             local-service
    host_name                       mysql
    service_description             ftp
    check_command                   check_ftp
    }

define service{
    use                             local-service
    host_name                       mysql
    service_description             mysqlport
    check_command                   check_tcp!3306
    }
```

这里对 Web 和 MySQL 主机设置了 4 个监控服务，分别是 check_ping、check_ssh、check_ftp 和 check_http。如果要监控远程主机更多的服务，按照相同格式增加即可。

在 service.cfg 文件中，有很多对命令的定义，例如监控 SSH 服务的 check_ssh、监控 FTP 服务的 check_ftp、监控 HTTP 服务的 check_http 等，这些命令均在 commands.cfg 中进行定义。结合 commands.cfg 和 resource.cfg 文件，不难看出，这些命令对应的真实路径是 /usr/local/nagios/libexec，也就是说，这些命令其实就是安装 Nagios 插件后生成的可执行文件。

在 Nagios 中，插件命令和参数的组合格式为：命令！参数！参数。如果有更多参数，依次通过叹号分割即可。下面列举几个命名组合进行介绍。

```
check_ping!100.0,20%!500.0,60%
```

此命令组合从左到右依次为：命令！告警时延，丢包率！严重告警时延，丢包率。

```
check_http!0.0020!0.0050!10
```

此命令组合从左到右依次为：命令！告警时延！严重告警时延！连接超时时间。

```
check_tcp!23!0.0020!0.0050!10
```

此命令组合从左到右依次为：命令！端口！告警时延！严重告警时延！连接超时时间。

```
check_ssh!22!10
```

此命令组合从左到右依次为：命令！端口！连接超时时间。

```
check_smtp!0.0020!0.0050!10
```

此命令组合从左到右依次为：命令！告警时延！严重告警时延！连接超时时间。

另外，在监控服务器端口时，很多命令都可以使用 check_tcp 来代替，例如：

```
check_ssh=check_tcp!22
check_imap=check_tcp!143
check_ftp=check_tcp!21
check_nntp=check_tcp!119
check_pop=check_udp!110
check_udp=check_tcp
check_telnet=check_tcp!23
```

（6）contacts.cfg 文件

contacts.cfg 是一个定义联系人和联系人组的配置文件，当监控的主机或者服务出现故障时，Nagios 会通过指定的通知方式（邮件或短信）将信息发给这里指定的联系人或使用者。下面是一个配置好的示例。

```
define contact{
    contact_name        sasystem          # 联系人名称
    use     generic-contact               # 引用 generic-contact 的属性信息，
                                          # 其中“generic-contact”在 templates.cfg 文件中进行定义
    alias   sa-system                     # 联系人别名
    email   ixdba@126.com                 # 联系人的邮件地址
    }
define contactgroup {
 contactgroup_name     admins                                  # 联系人组名称
 alias                 system administrator group              # 联系人组描述
 members               sasystem                                # 联系人组成员，其中“sasystem”
                                                               # 就是上面定义的联系人
}
```

（7）timeperiods.cfg 文件

此文件只用于定义监控的时间段。下面是一个配置好的示例：

```
                                    # 下面定义一个名为 24x7 的时间段，即监控所有时间段
define timeperiod{
   timeperiod_name 24x7
   alias           24 Hours A Day, 7 Days A Week
   sunday          00:00-24:00
   monday          00:00-24:00
   tuesday         00:00-24:00
   wednesday       00:00-24:00
   thursday        00:00-24:00
   friday          00:00-24:00
   saturday        00:00-24:00
   }
                                    # 下面定义一个名为 workhours 的时间段，即工作时间段
define timeperiod{
   timeperiod_name workhours
   alias           Normal Work Hours
   monday          09:00-17:00
   tuesday         09:00-17:00
   wednesday       09:00-17:00
   thursday        09:00-17:00
   friday          09:00-17:00
   }
```

（8）cgi.cfg 文件

此文件用来控制相关 CGI 脚本，如果想在 Nagios 的 Web 监控界面执行 CGI 脚本，例如重启 Nagios 进程、关闭 Nagios 通知、停止 Nagios 主机检测等，这时就需要配置 cgi.cfg 文件了。由于 Nagios 的 Web 监控界面验证用户为 ixdba，因此只需在 cgi.cfg 文件中添加此用户的执行权限就可以了。需要修改的配置信息如下：

```
default_user_name=ixdba
authorized_for_system_information=nagiosadmin,ixdba
authorized_for_configuration_information=nagiosadmin,ixdba
authorized_for_system_commands=ixdba
authorized_for_all_services=nagiosadmin,ixdba
authorized_for_all_hosts=nagiosadmin,ixdba
authorized_for_all_service_commands=nagiosadmin,ixdba
authorized_for_all_host_commands=nagiosadmin,ixdba
```

到这里为止，Nagios 的所有对象配置文件已经修改完了。将创建好的 8 个文件全部放到 /usr/local/nagios/etc 目录下，接着开始配置 nagios.cfg 文件。

（9）nagios.cfg 文件

nagios.cfg 文件默认的路径为 /usr/local/nagios/etc/nagios.cfg，是 Nagios 的核心配置文件，所有的对象配置文件都必须在这个文件中进行定义才能发挥其作用，这里只需引用对象配置文件即可。下面分别对 Nagios.cfg 文件中的内容进行介绍。

```
log_file=/usr/local/nagios/var/nagios.log
```

“log_file”变量用来定义 Nagios 日志文件的路径。

```
cfg_file=/usr/local/nagios/etc/hosts.cfg
```

```
cfg_file=/usr/local/nagios/etc/services.cfg
cfg_file=/usr/local/nagios/etc/commands.cfg
cfg_file=/usr/local/nagios/etc/contacts.cfg
cfg_file=/usr/local/nagios/etc/timeperiods.cfg
cfg_file=/usr/local/nagios/etc/templates.cfg
```

cfg_file 变量用来引用对象配置文件，如果有更多的对象配置文件，那么在这里依次添加即可。但在有些时候，监控的对象多达几千个，如果都在这里进行引用，那将会非常费力，而且还不利于日后的维护，此时，就需要另一个变量了，那就是 cfg_dir。该变量用于指定一个目录，所有在这个目录下的以 .cfg 为扩展名的文件都将作为对象配置文件来处理。另外，Nagios 将会递归该目录下的子目录并处理其子目录下的全部配置文件。

```
object_cache_file=/usr/local/nagios/var/objects.cache
```

object_cache_file 变量用于指定一个“所有对象配置文件”的副本文件，又称为对象缓冲文件，这个设定可以加快 CGI 的配置缓冲，并且在编辑对象配置文件时，可以让正在运行的 Nagios 不影响 CGI 的显示输出。

```
resource_file=/usr/local/nagios/etc/resource.cfg
```

resource_file 变量用于指定 Nagios 资源文件的路径，可以在 Nagios.cfg 中定义多个资源文件。

```
status_file=/usr/local/nagios/var/status.dat
```

status_file 变量用于定义一个状态文件，此文件用于保存 Nagios 的当前状态、注释和宕机信息等。

```
status_update_interval=10
```

status_update_interval 变量用于定义状态文件（即 status.dat）的更新时间间隔，单位是秒，最小更新间隔是 1 秒。

```
nagios_user=nagios
```

nagios_user 变量指定了 Nagios 进程由哪个用户运行。

```
nagios_group=nagios
```

nagios_group 变量用于指定 Nagios 由哪个用户组运行。

```
check_external_commands=1
```

check_external_commands 变量用于设置是否允许 Nagios 在 Web 监控界面上运行 CGI 命令，也就是是否允许 Nagios 在 Web 界面下执行重启 Nagios、停止主机 / 服务检查等操作，“1”为运行，“0”为不允许运行。

```
command_check_interval=2
```

command_check_interval 变量用于设置 Nagios 对外部命令检测的时间间隔，如果指定了一个数字加一个“s”（如 10s），那么外部检测命令的间隔是这个数值以秒为单位的时间间隔。如果没有“s”，那么外部检测命令的间隔是以这个数值为“时间单位”的时间间隔。

```
interval_length=60
```

interval_length 变量指定了 Nagios 的时间单位，默认值是 60 秒，也就是 1 分钟，即在 Nagios 配置中所有的时间单位都是分钟。

9.3　Nagios 的运行和维护

9.3.1　验证 Nagios 配置文件的正确性

在上节中，已经配置完成了一个基本的 Nagios 监控系统。那么如何知道配置文件的正确性呢？ Nagios 在这个方面做得非常到位，只需通过如下一个命令即可完成：

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

得到的检测结果如下：

```
Nagios Core 3.2.0
Copyright (c) 2009 Nagios Core Development Team and Community Contributors
Copyright (c) 1999-2009 Ethan Galstad
Last Modified: 08-12-2009
License: GPL
Website: http://www.nagios.org
Reading configuration data...
   Read main config file okay...
Processing object config file '/usr/local/nagios/etc/templates.cfg'...
Processing object config file '/usr/local/nagios/etc/services.cfg'...
Processing object config file '/usr/local/nagios/etc/hosts.cfg'...
Processing object config file '/usr/local/nagios/etc/timeperiods.cfg'...
Processing object config file '/usr/local/nagios/etc/contacts.cfg'...
Processing object config file '/usr/local/nagios/etc/commands.cfg'...
Error: Could not find any host matching 'web1' (config file '/usr/local/nagios/etc/
monitor/hosts.cfg', starting on line 14)
Error: Could not expand members specified in hostgroup (config file '/usr/local/
nagios/etc/monitor/hosts.cfg', starting on line 14)
   Error processing object config files!
```

通过 Nagios 给出的错误提示，可以非常迅速地定位错误根源，由此可以知道，Nagios 无法找到在 hosts.cfg 文件的第 14 行中定义的 web1 主机，因为在 hosts.cfg 中定义的主机是 web，所以修改主机组中的 web1 为 web 即可解决问题。

Nagios 提供的这个验证功能非常有用，在错误信息中通常会显示出错的配置文件及在文件中的哪一行，这使修改 Nagios 的配置变得非常容易。检测结果中的报警信息通常是可以忽略的，因为一般只是建议性的。

9.3.2　启动与停止 Nagios

有多种启动、停止和重启 Nagios 的方法，读者可以根据自己的需要，任选其一。

1. 启动 Nagios

（1）通过初始化脚本启动 Nagios

```
/etc/init.d/nagios start
```

或者

```
Service nagios start
```

（2）手工方式启动 Nagios

通过 nagios 命令的“-d”参数来启动 nagios 守护进程。

```
/usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg
```

2. 关闭 nagios

（1）通过初始化脚本关闭 Nagios 服务

```
/etc/init.d/nagios stop
```

或者

```
Service nagios stop
```

（2）通过 kill 方式关闭 Nagios

```
kill <nagios_pid>
```

3. 重启 Nagios

（1）通过初始化脚本来重启 Nagios

```
/etc/rc.d/init.d/nagios reload
/etc/rc.d/init.d/nagios restart
```

（2）重启 Nagios

通过 Web 监控页重启 Nagios，如图 9-1 所示。

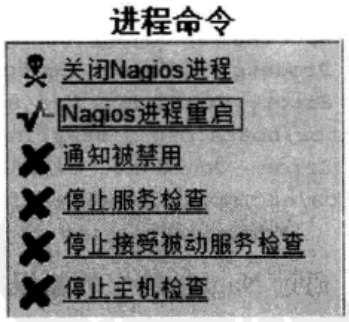

图 9-1 通过 Web 监控页重启 Nagios

（3）手工方式平滑重启

```
kill -HUP <nagios_pid>
```

到此为止，Nagios 监控系统已经成功搭建并且运行起来了。图 9-2 是 Nagios 的一个 Web 监控界面截图。

9.3.3 Nagios 故障报警

Nagios 的故障报警功能非常强大，可以支持邮件、短信、MSN、QQ 等多种方式，每种报警方式都有自己的优缺点：邮件报警是最普通、最简单的方式，无需任何插件，只需要添加报警的邮件地址即可，但是实时性不好；短信报警实时性最好，可以直接将监控结果发送到指定的手机上，但是短信报警方式需要硬件或第三方插件支持，成本略高；MSN 和 QQ 报警方式实时性介于邮件和短信之间，但需要编写自定义脚本，因此复杂性稍高。读者根据自己的环境和需要选择合适的报警方式即可。

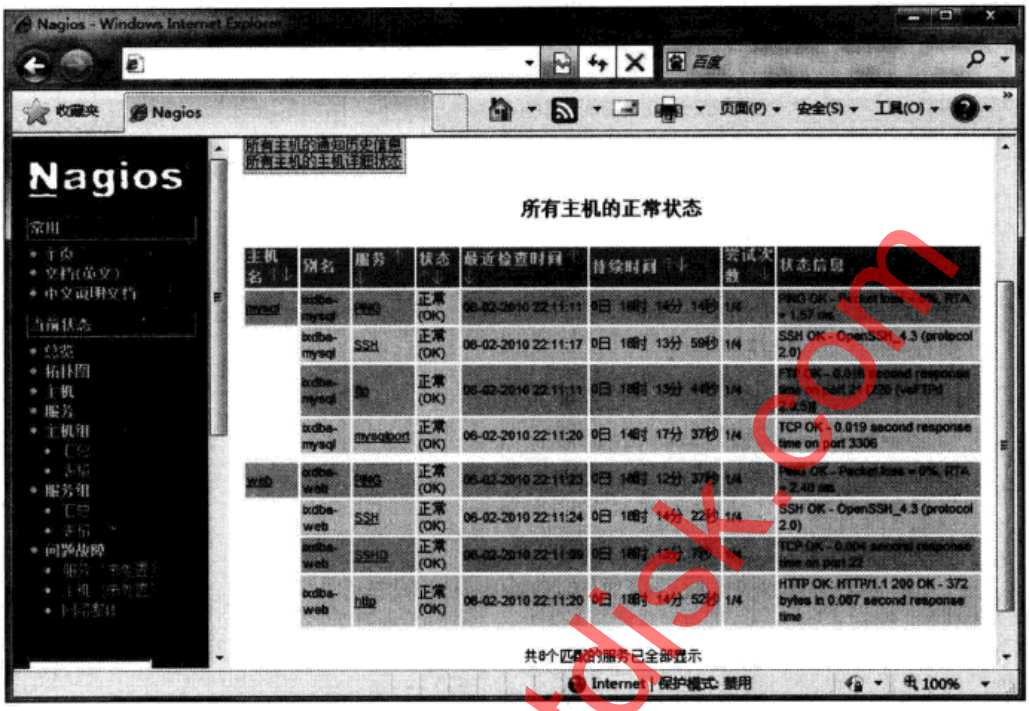

图 9-2　Nagios 监控系统截图

下面是两个邮件报警通知的截图，图 9-3 是服务异常时 Nagios 发送的服务故障邮件报警，而图 9-4 是服务恢复正常后 Nagios 再次发送的服务正常邮件报警。

图 9-3　服务故障时 Nagios 发送的报警邮件

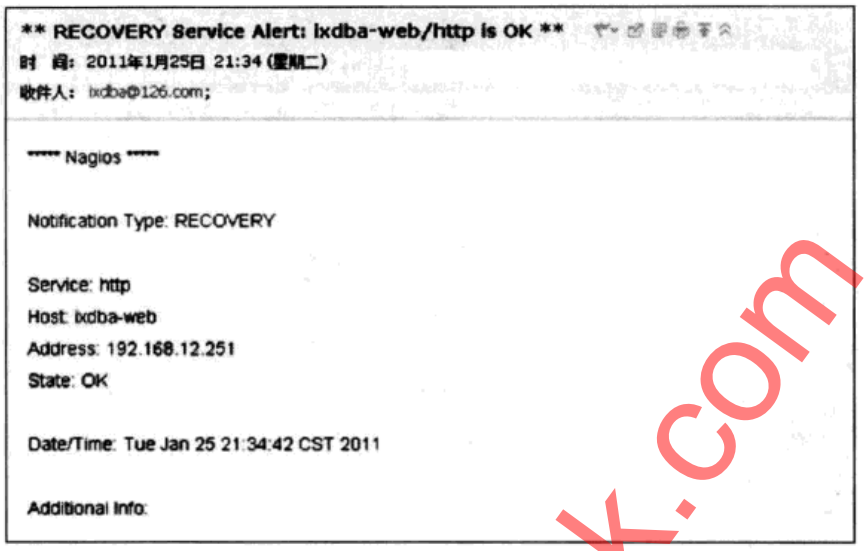

图 9-4 服务恢复时 Nagios 发送的服务正常邮件

9.4 Nagios 性能分析图表的实现

9.4.1 Nagios 性能分析图表的作用

Nagios 对服务或主机监控的是一个瞬时状态，有时候系统管理员需要了解主机在一段时间内的性能及服务的响应状态，并且形成图表，这就需要通过查看日志数据来分析。但是这种方式不但烦琐，而且抽象。不过幸运的是，PNP 可以帮助我们来完成这个工作。

9.4.2 PNP 的概念与安装环境

PNP 是一个小巧的开源软件包，它是基于 PHP 和 Perl 的。PNP 可以利用 rrdtool 工具将 Nagios 采集的数据绘制成相关的图表，然后显示主机或者服务在一段时间内的运行状况。

如果要安装 PNP，首先需要安装如下环境：

- 整合后的 Apache 和 PHP 环境，需支持 GD\zlib\jpeg。
- 安装 RDDtool 工具。
- 安装 Perl。

9.4.3 安装 PNP

RRDtool 是一个图表生成工具，可以从 http://www.mrtg.org/rrdtool/ 获得信息。这里下载的版本是 rrdtool-1.4.5.tar.gz。安装过程如下：

```
[root@nagios rrdtool]# tar zxvf rrdtool-1.4.5.tar.gz
[root@nagios rrdtool]# cd rrdtool-1.4.5
[root@nagios rrdtool-1.4.5]# ./configure --prefix=/usr/local/rrdtool
[root@nagios rrdtool-1.4.5]# make
[root@nagios rrdtool-1.4.5]# make install
```

接着安装 PNP，这里下载的版本是 pnp-0.4.13.tar.gz。安装过程如下：

```
[root@nagios pnp]#tar -xvzf pnp-0.4.13.tar.gz
[root@nagios pnp]#cd pnp-0.4.13
[root@nagios pnp-0.4.13]#./configure --with-nagios-user=nagios \
>--with-nagios-group-nagios \
>--with-rrdtool=/usr/local/rrdtool/bin/rrdtool \
>--with-perfdata-dir=/usr/local/nagios/share/perfdata
[root@nagios pnp-0.4.13]#make all
[root@nagios pnp-0.4.13]#make install
[root@nagios pnp-0.4.13]#make install-config
[root@nagios pnp-0.4.13]#make install-init
```

安装完成。PNP 默认文件的放置情况如下：

```
General Options:
-----------------------------        --------------------
  Nagios user/group:                 nagios nagios
  Install directory:                 /usr/local/nagios
  HTML Dir:                          /usr/local/nagios/share/pnp
  Config Dir:                       /usr/local/nagios/etc/pnp
  Path to rrdtool:                   /usr/local/bin/rrdtool (Version 1.4.5)
  RRDs Perl Modules:                  *** NOT FOUND ***
  RRD Files stored in:               /usr/local/nagios/share/perfdata
  process_perfdata.pl Logfile:       /usr/local/nagios/var/perfdata.log
  Perfdata files (NPCD) stored in:   /usr/local/nagios/var/spool/perfdata/
-------------------------------------------------------------------------
```

9.4.4　配置 PNP

1. 创建默认配置文件

在 PNP 安装完成后，默认安装目录下自带了模板配置文件，因此，只需将模板文件复制一份作为 PNP 配置文件即可。操作如下：

```
[root@nagios etc]#cd  /usr/local/nagios/etc/pnp/
[root@nagios pnp]#cp process_perfdata.cfg-sample  process_perfdata.cfg
[root@nagios pnp]#cp npcd.cfg-sample  npcd.cfg
[root@nagios pnp]#cp rra.cfg-sample  rra.cfg
[root@nagios pnp]#chown -R nagios:nagios /usr/local/nagios/etc/pnp
```

2. 修改 process_perfdata.cfg 文件

打开 Nagios 下的 process_perfdata.cfg 文件，修改相关内容。操作如下：

```
[root@nagios pnp]#vi /usr/local/nagios/etc/pnp/process_perfdata.cfg
```

```
LOG_FILE = /usr/local/nagios/var/perfdata.log
# Loglevel 0=silent 1=normal 2=debug
LOG_LEVEL = 2
```

这里将日志级别改为 2，即 debug 模式。

9.4.5 修改 Nagios 配置文件

1. 增加小太阳图标

修改 templates.cfg，增加一个定义 PNP 的 host 和 service。修改后的内容如下：

```
define host {
   name        hosts-pnp
   register    0
   action_url /nagios/pnp/index.php?host=$HOSTNAME$
   process_perf_data                1

}
define service {
   name        services-pnp
   register    0
   action_url /nagios/pnp/index.php?host=$HOSTNAME$&srv=$SERVICEDESC$
   process_perf_data                1

}
```

2. 修改 nagios.cfg

Nagios 监控系统提供的数据接口可供第三方插件使用，而 PNP 刚好就是调用 Nagios 的数据来生成图表的。在前面介绍 templates.cfg 文件时提到“process_perf_data”选项，这个选项就是用来定义是否开启 Nagios 的数据输出功能的，这个选项的值可以是 0 或 1，设置为 1 表示开启 Nagios 的数据输出功能。因此，如果想让 Nagios 将数据输出，首先要修改 Nagios 的主配置文件 nagios.cfg，找到如下几项，去掉注释。修改后的信息如下：

```
process_performance_data=1
host_perfdata_command=process-host-perfdata
service_perfdata_command=process-service-perfdata
```

其中，“process-host-perfdata”和“process-service-perfdata”指令是新启用的，这两个指令默认已经在 commands.cfg 文件中进行定义了。

3. 修改 commands.cfg

process-host-perfdata 和 process-service-perfdata 指令声明了 Nagios 输出哪些值到输出文件中。不过这些定义相对简单，而 PNP 提供了一个 Perl 脚本，非常详细地定义了一个输出数据的方法，process_perfdata.pl 就是 PNP 自带的一个脚本，这个脚本在 PNP 安装完成后会自动生成。因此，可以将 process-host-perfdata 和 process-service-perfdata 指令中对应的执行命令的内容替换成此脚本。修改后内容如下：

```
# '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
}
```

4. 修改 hosts.cfg 文件和 services.cfg 文件

将 hosts-pnp 和 services-pnp 引用到 hosts.cfg 和 services.cfg 中，修改后的 hosts.cfg 内容如下：

```
define host{
    use                     linux-server,hosts-pnp
    host_name               web
    alias                   ixdba-web
    address                 192.168.12.251
    }

define host{
    use                     linux-server,hosts-pnp
    host_name               mysql
    alias                   ixdba-mysql
    address                 192.168.12.237
    }
```

修改后的 services.cfg 内容如下：

```
define service{
    use                             local-service,services-pnp
    host_name                       mysql
    service_description             SSH
    check_command                   check_ssh
    }
define service{
    use                             local-service,services-pnp
    host_name                       web
    service_description             http
    check_command                   check_http
    }
```

9.4.6　测试 PNP 功能

完成所有配置之后，重新检查 Nagios 配置文件是否正确，然后重启 Nagios。执行的命

令如下：

```
[root@nagios web]#/etc/init.d/nagios restart
```

如果配置正确，此时就会生成相应主机的 PNP 文件。

```
[root@nagios web]# pwd
/usr/local/nagios/share/perfdata/web
[root@nagios web]# ls
http.rrd  http.xml  PING.rrd  PING.xml  SSHD.rrd  SSHD.xml
```

最后打开网页 http://IP/nagios，选择主机选项，然后单击主机旁边的小太阳图标，可以看到主机监控状态图表，如图 9-5 所示。也可以单击服务旁边的小太阳图标，进入服务监控状态图表，如图 9-6 所示。或者访问 http://ip/nagios/pnp 也可以直接访问图表信息。

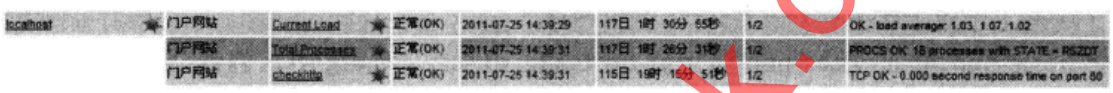

图 9-5 带有小太阳图标的主机状态

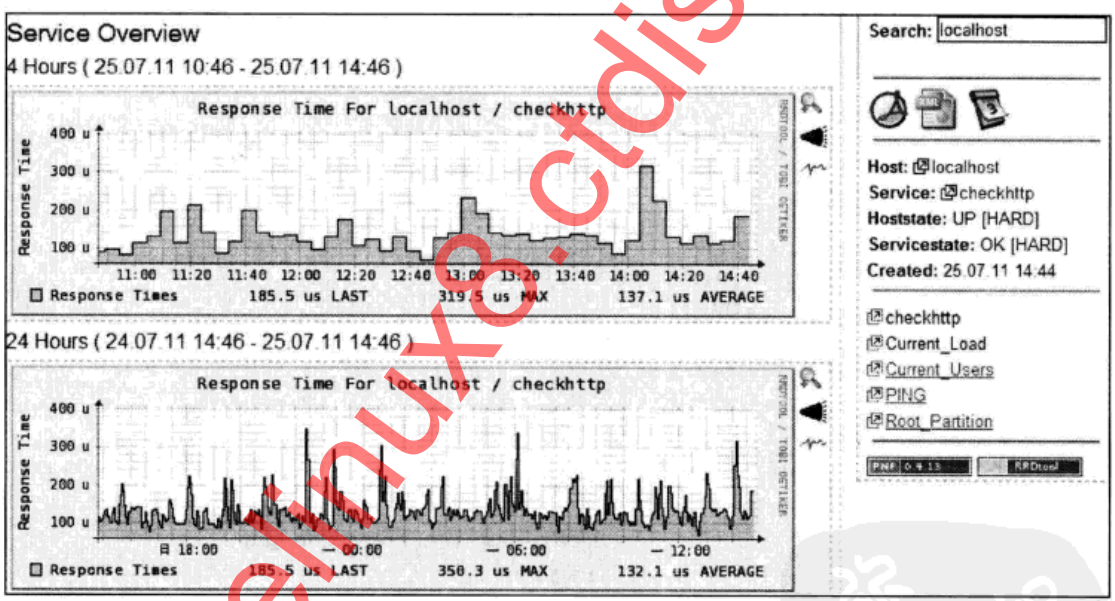

图 9-6 Nagios 监控服务运行状态生成的图表

9.5 利用插件扩展 Nagios 的监控功能

9.5.1 利用 NRPE 外部构件监控远程主机

Nagios 监控系统对远程主机上服务状态的获取，可以通过一些相应的服务检测命令，例如 check_http、check_ftp 等来完成。那么，如果要获取远程主机上的本地资源或者属性，例

如要监控远程主机上的磁盘利用率、CPU 利用率、系统负载时，该如何实现呢？虽然插件中有 check_disk、check_load、check_swap 之类的工具，但是这些工具仅能获取主机自身信息，而无法获取远程主机的信息，此时就需要借助一个外部构件来完成，这个构件就是 NRPE。

NRPE 是 Nagios 的一个功能扩展，它可在远程 Linux 和 UNIX 主机上执行插件程序。通过在远程服务器上安装 NRPE 构件及 Nagios 插件程序来向 Nagios 监控平台提供该服务器的一些本地情况，例如，CPU 负载、内存使用、硬盘使用等。为了方便理解，这里将 Nagios 监控平台称为 Nagios 服务器端，而将远程被监控的服务器称为 Nagios 的客户端。

其实在 Nagios 的插件中，有一个名为 check_ssh 的插件，它也可以用于实现对远程服务器中本地信息的监控。但是，与 NRPE 相比，check_ssh 会占用很高的系统负荷，在监控少量的服务时可能不会察觉，但是在监控成百个主机中的上千个服务时，差距就非常明显了。还有一点要说明的是，虽然 NRPE 也使用 SSL 安全通道，但是 check_ssh 的安全性要高于 NRPE。从这里可以看出，安全性总是和实用性成反比的。

图 9-7 表示 NRPE 构件监控远程主机本地信息的运行原理。

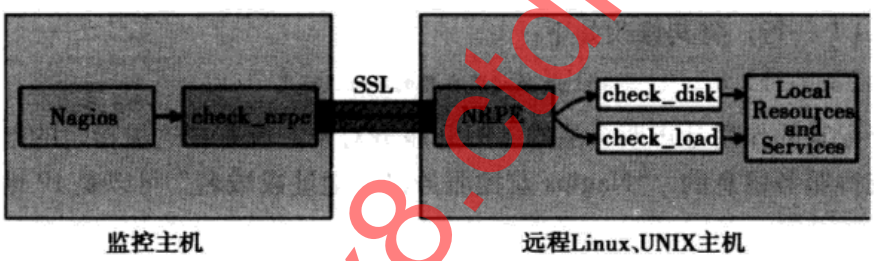

图 9-7　NRPE 监控远程主机的运行原理

下面通过在 Nagios 服务器端和客户端安装 NRPE 来搭建一个更加完善的 Nagios 监控系统。

1. 配置 Nagios 客户端（即远程主机）

在 Nagios 客户端主机上安装 NRPE 和 Nagios 插件，NRPE 插件可以从 Nagios 官方网站下载，这里从 http://www.nagios.org/download/addons 下载最新稳定版本 nrpe-2.12.tar.gz，然后开始安装和配置。基本操作如下：

1）安装 Nagios 插件。

```
[root@nagios-client ~]#useradd -s /sbin/nologin nagios
[root@nagios-client ~]#tar zxvf nagios-plugins-1.4.14.tar.gz
[root@nagios-client ~]#cd nagios-plugins-1.4.14
[root@nagios-client ~]#./configure
[root@nagios-client ~]#make
[root@nagios-client ~]#make install
```

执行如下命令设置插件目录权限：

```
[root@nagios-client ~]#chown nagios.nagios /usr/local/nagios
```

```
[root@nagios-client ~]#chown -R nagios.nagios /usr/local/nagios/libexec
```

2）安装 NRPE 插件。

在客户端安装 NRPE 插件的过程要比在服务器端安装复杂，因为 NRPE 在客户端是作为一个守护进程运行的。操作如下：

```
[root@nagios-client ~]#tar zxvf nrpe-2.12.tar.gz
[root@nagios-client ~]#cd nrpe-2.12
[root@nagios-client ~]#./configure
[root@nagios-client ~]#make all
[root@nagios-client ~]#make install-plugin
[root@nagios-client ~]#make install-daemon
[root@nagios-client ~]#make install-daemon-config
```

这样，NRPE 插件就安装完成了。可以看到，在 /usr/local/nagios/libexec 下已经生成了一个 check_nrpe 指令，这就是监控远程主机必需的命令。

3）配置 NRPE。

NRPE 的配置文件为 /usr/local/nagios/etc/nrpe.cof。在该文件中找到“allowed_hosts=127.0.0.1”一行，将其修改如下：

```
allowed_hosts=127.0.0.1, Nagios 监控服务器的地址或域名
```

修改这个配置的作用是声明合法的 NRPE 服务对象，没有在这里指定的地址是无法从本机的 NRPE 获得服务信息的。“Nagios 监控服务器的地址或域名”可以是 IP 地址，也可以是域名，根据情况设定。

4）启动 NRPE 守护进程。

启动 NRPE 很简单，只需执行如下操作：

```
/usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
```

建议将此命令加入到 /etc/rc.local 文件中，这样就可以在开机时自动运行 NRPE 守护进程了。

NRPE 守护进程的默认端口为 5666，通过如下命令可以检测端口是否启动：

```
[root@nagios-client ~]# netstat -antl|grep 5666
tcp        0      0 0.0.0.0:5666                0.0.0.0:*                   LISTEN
```

可以看到，NRPE 守护进程端口 5666 已经启动了。

5）测试 NRPE 功能。

首先在 Nagios 客户端本机上测试，执行如下命令：

```
/usr/local/nagios/libexec/check_nrpe -H 127.0.0.1
```

如果正常，应该出现如下信息：

```
[root@nagios-client ~]# /usr/local/nagios/libexec/check_nrpe  -H  127.0.0.1
NRPE v2.12
```

正常的返回值为被监控服务器上安装的 NRPE 的版本信息，如果能看到这些，表示 NRPE 已经正常工作了。

通过前面的介绍，可以得出一个结论：只要被监控服务器上有的插件（/usr/local/nagios/libexec 中的所有插件），都可以在 Nagios 服务器端通过 NRPE 插件来获取远程主机信息，也可以说，要监控客户端什么信息，只要客户端有对应的插件，就可以实现。

6）定义监控服务器内容。

要监控一个远程服务器下的某些信息，首先要在远程服务器中定义监控的内容，例如，要监控一台远程服务器的当前用户数、CPU 负载、磁盘利用率、交换空间使用情况，需要在 nrpe.conf 中定义如下监控内容：

```
command[check_users_1]=/usr/local/nagios/libexec/check_users -w 5 -c 10
command[check_load_1]=/usr/local/nagios/libexec/check_load -w 15,10,5 -c 30,25,20
command[check_sda5_1]=/usr/local/nagios/libexec/check_disk -w 20% -c 10% -p /dev/sda5
command[check_zombie_procs_1]=/usr/local/nagios/libexec/check_procs -w 5 -c 10 -s Z
command[check_total_procs_1]=/usr/local/nagios/libexec/check_procs -w 150 -c 200
command[check_swap_1]=/usr/local/nagios/libexec/check_swap -w 20 -c 10
```

其中，command 后面中括号中的内容就是定义的变量，变量名可以随意指定，只要在 Nagios 服务器端配置文件引用时保持统一即可。从变量指向的路径可知，最终是指向了 Nagios 插件。

2. 在服务器端安装 NRPE 和配置 Nagios 服务

（1）安装 NRPE 插件

通过图 9-7 可知，Nagios 服务器端是通过 NRPE 插件来和客户端 NRPE 守护进程进行通信的，因此在 Nagios 服务器端也需要安装 NRPE 插件。在服务器端安装 NRPE 很简单，操作如下：

```
[root@nagiosserver ~]#tar zxvf nrpe-2.12.tar.gz
[root@ nagiosserver ~]#cd nrpe-2.12
[root@ nagiosserver ~]#./configure
[root@ nagiosserver ~]#make all
[root@ nagiosserver ~]#make install-plugin
```

通过 make install-plugin 命令默认将 check_nrpe 插件安装到 /usr/local/nagios/libexec 目录下。

（2）测试插件与客户端是否能正常通信

在 Nagios 服务器端（即 Nagios 监控平台）执行如下指令：

```
/usr/local/nagios/libexec/check_nrpe -H 客户端主机地址
```

例如：

```
[root@nagiosserver  ~]# /usr/local/nagios/libexec/check_nrpe  -H 192.168.12.251
NRPE v2.12
```

如果能显示如上的输出信息，表明 NRPE 可以与客户端正常通信。

（3）定义一个 check_nrpe 监控命令

修改 /usr/local/nagios/etc/commands.cfg 文件，添加如下内容：

```
define command{
command_name check_nrpe
command_line $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
}
```

（4）添加远程主机监控

紧接 9.2.2 章节的配置示例，修改 /usr/local/nagios/etc/service.cfg，添加如下监控内容：

```
define service{
    use                             local-service
    host_name                       mysql
    service_description             users
    check_command                   check_nrpe!check_users_1
    }
define service{
    use                             local-service
    host_name                       mysql
    service_description             load
    check_command                   check_nrpe!check_load_1
    }

define service{
    use                             local-service
    host_name                       mysql
    service_description             disk
    check_command                   check_nrpe!check_sda5_1
    }
define service{
    use                             local-service
    host_name                       mysql
    service_description             swap
    check_command                   check_nrpe!check_swap_1
    }
define servicegroup{                            # 定义一个服务组
    servicegroup_name       servergroup         # 服务组名称，可以随意指定
    alias                   server-group        # 服务组别名
    members
web,PING,web,SSH,web,SSHD,web,http,mysql,users,mysql,load,mysql,disk,mysql,swap
                                                # 服务组成员，格式为“主机名，主机对应的服务描述”
    }
```

增加的这段配置是为了对远程 MySQL 主机的当前用户数、系统负载、磁盘空间利用率、swap 内存使用 4 个方面进行监控。从这段配置中可以看出，监控远程主机的命令方式为：check_nrpe! 远程主机下定义的监控变量，而 check_users_1、check_load_1 等变量已经在 Nagios 客户端 nrpe.cfg 文件中进行了定义，这里仅作为 check_nrpe 的参数进行引用而已。

（5）测试和启动 Nagios 服务

```
[root@nagiosserver ~]#/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

```
[root@nagiosserver  ~]#/etc/init.d/nagios restart
```

最后给出一个通过扩展插件 NRPE 搭建完成的 Nagios 监控系统，如图 9-8 所示。

图 9-8　通过扩展插件 NRPE 搭建完成的 Nagios 监控系统

9.5.2　利用飞信实现 Nagios 短信报警功能

对于一个完善的 Nagios 监控系统，故障报警的准确性和及时性显得尤为重要。报警的方式有很多种，可以通过邮件报警、手机短信报警、QQ 或 MSN 报警等。这些方式各有优缺点，通过邮件、QQ 或 MSN 进行报警最简单实用，但是及时性不好；通过手机短信方式最方便，而且及时性很高，但是短信报警需要使用短信猫或短信网关，这些设备不是每个企业都有的，并且还要支付短信费用。那么，有没有一种既免费又及时快捷的报警方式呢？当然有！飞信业务就提供了一个很好的思路。

要使用飞信功能，首先手机要开通中国移动的飞信业务，目前中国移动的官方网站仅仅提供了飞信的 Windows 客户端，但飞信爱好者为了满足自己的使用，开发出了飞信的 Linux 版本，这对于我们是一种福音。

1. 下载 linux 版本的飞信程序

可以在 http://www.it-adv.net/fetion/downng/fetion20091117-linux.tar.gz（不保证此链接永久有效）上下载飞信的 Linux 客户端程序。安装飞信的 Linux 客户端需要 libace 的 glibc 库的

支持，不过，这个包中带有匹配的 Library 库文件，所以不需要单独下载。

2. 安装与配置飞信

（1）安装飞信

这里假定飞信的安装目录为 /usr/local/fetion。操作如下：

```
[root@nagiosserver  ~]#tar zxvf fetion20091117-linux.tar.gz
[root@nagiosserver  ~]# cp fx/* /usr/local/fetion
```

执行完毕后，/usr/local/fetion/fetion 就是我们需要的飞信客户端程序。

（2）配置飞信

配置飞信所需的动态链接库。

```
[root@nagiosserver  ~]# vi /etc/ld.so.conf
include ld.so.conf.d/*.conf
/usr/local/fetion
[root@localhost src]# ldconfig
```

测试飞信能否正常运行。

```
[root@nagiosserver  ~]#ldd /usr/local/fetion/fetion
[root@nagiosserver  ~]#/usr/local/fetion/fetion
Usage:
    --mobile=[mobile]
    --sid=[sid]
    --pwd=[pwd]
    --config=[config file] *format:index mobile password
    --index=[index no in config file,refer to sample.conf]
    --debug  *debug mode on
    --hide  *login fetion in hidden state
    --to=[mobile/sid]
    --add=[uri]
    --command-path=[command file path]
    --robotmode
    --daemon(linux only)
    --proxy-ip(http proxy ip)
    --proxy-port(http proxy port)
    --msg-gb=[gb2312/gbk message]
    --msg-utf8=[utf8 message]
    --msg-type=[0/1/2/3 sms longsms smartmsg mms]
    --file-gb=[gb2312/gbk file]
    --file-utf8=[utf8 file]
    --query-cmcc-no
    --testaccount
    --auto-retry
    --get-web-session
    --action=getpiccode --mobile=13910000000 --pwd=[pwd] (--piccode-url=[url])
      --action=appsubscribe  --mobile=[mobile]  --pwd=[pwd]  --pic-certificate-
```

```
id=[certid] --pic-cert-code=[certcode] --apply-sub-service-url=[url]
        --action=subscribe --mobile=[mobile] --pwd=[pwd] --sms-code=[smscode]
            --subscribe-url=[url]
        --action=updatepwd --mobile=[mobile] --pwd=[pwd] --sms-code=[smscode]
            --update-pwd-url=[url]
```

如果出现以上的帮助信息，表示飞信已经安装成功。

（3）飞信的使用说明

以下参数提供登录用的账号和密码，如表 9-3 所示。

表 9-3　提供登录账号和密码的参数及含义

参数名称	表示含义
--mobile=[手机号]	以手机号登录飞信
--sid=[飞信号]	以飞信号登录飞信
--pwd=[密码]	登录飞信的密码
--config=[文件名]	手机号、密码的存储文件
--index=[索引号]	索引

以下参数用来定义接收者的属性，如表 9-4 所示。

表 9-4　定义接收者属性的参数及含义

参数名称	表示含义
--to=[手机号 / 飞信号 /URI]	接收消息的手机号 / 飞信号 /URI，支持多个号码，中间用逗号分隔。如果知道对方的 URI，则只需自己在对方好友列表，无需对方在自己好友列表就能发送
--msg-utf8=[信息]	指定发送消息格式采用 UTF8 编码
--msg-gb=[信息]	指定发送消息格式采用 GBK 编码
--file-utf8=[文件 utf8 格式]	以文件形式指定发送消息的内容，文件格式必须是 UTF8 编码格式
--file-gb=[文件 gb 格式]	以文件形式指定发送消息的内容，文件格式必须是 GBK 编码格式
--msg-type=[0/1/2]	发送消息类型：“0”表示普通消息，“1”表示长消息，“2”表示智能短信
--query-cmcc-no	查询移动公司手机段

以下参数为可选项，如表 9-5 所示。

表 9-5　可选参数及含义

参数名称	表示含义
--debug	显示调试信息
--hide	以隐身方式登录飞信
--proxy-ip=http	以代理 IP 方式登录
--proxy-port=http	以代理端口方式登录

（4）举例

```
[root@nagiosserver ~]#/usr/local/fetion/fetion --mobile=13466xxxxxx
    --pwd=chengxc123 --to 13866xxxxxx --msg-utf8=”test fetion”
```

这个例子是测试飞信能否成功发送短信的。注意，发送对象必须是自己的飞信好友或自己。其中，“13466xxxxxx”是发送人的手机号码，“13866xxxxxx”是接收人的手机号码。

3. 测试飞信功能

执行以下命令：

```
[root@nagiosserver ~]#/usr/local/fetion/fetion --mobile=135xxxxxxxx
    --pwd=123456789 --to=136xxxxxxxx --msg-gb="ThisI is test for fetion" --debug
```

如果短信发送成功，应该能看到以下返回信息：

```
SIP-C/2.0 280 Send SMS OK
```

飞信测试成功后，就可将飞信整合到 Nagios 中了。

4. Nagios 配置

（1）编辑 /usr/local/nagios/etc/command.cfg 文件

添加如下内容：

```
define command{
   command_name notify-service-by-sms      #定义一个服务发生故障时发送报警
                                           #短信的指令
   command_line /usr/local/fetion/fetion --mobile=xxxxxxxxxxx --pwd=xxxxxx
       --to=$CONTACTPAGER$ --msg-utf8="$HOSTADDRESS$' $HOSTALIAS$/$SERVICEDESC$
       is $SERVICESTATE$"
        }

define command{
   command_name notify-host-by-sms        #定义一个主机发生故障时发送报警短信的指令
   command_line /usr/local/fetion/fetion --mobile=xxxxxxxxxxx --pwd=xxxxxx
       --to=$CONTACTPAGER$ --msg-utf8="Host $HOSTSTATE$ alert for $HOSTNAME$! on
       '$DATETIME$'"
}
```

（2）修改 /usr/local/nagios/etc/templates.cfg 文件

找到联系人为 generic-contact 的定义，修改后的内容如下：

```
define contact{
   name                            generic-contact
   service_notification_period     24x7
   host_notification_period        24x7
   service_notification_options    w,u,c,r
heduled downtime events
   host_notification_options       d,u,r
uled downtime events
   service_notification_commands   notify-service-by-email,notify-service-by-sms
   host_notification_commands      notify-host-by-email,notify-host-by-sms
   register                         0
   }
```

其中，加粗字体是新增的内容，也就是在 command.cfg 文件中新定义的两个指令。

（3）修改 /usr/local/nagios/etc/contacts.cfg 文件

修改联系人为 sasystem 的定义，修改后的内容如下：

```
define contact{
    contact_name                    sasystem
    use                             generic-contact
    alias                           sa-system
    email                           ixdba@126.com
    pager                           139xxxxxxxx
    }
```

其中，加粗字体部分为新增内容，“pager”用来指定接收报警短信的手机号码，如果有多个手机号码，每个号码之间用逗号分隔即可。

至此，通过飞信发送报警信息的 Nagios 监控系统配置完毕了。

9.6　本章小结

本章重点讲述了 Nagios 的安装、配置和使用。首先讲述了 Nagios 的基本结构，然后详细介绍了 Nagios 的配置和管理，最后讲述了如何利用外部插件扩展 Nagios 的监控功能。

Nagios 在系统运维、网络监控方面的运用非常广泛，越来越多的大型应用系统开始利用 Nagios 进行管理和监控，这与它自身强大的管理和监控功能是分不开的。Nagios 具有很强的扩展性，用户可以通过第三方插件或者自己的定制插件轻松完成特殊监控需求。

第 10 章　基于 Linux 服务器的性能分析与优化

作为一名 Linux 系统管理员，最主要的工作是优化系统配置，使应用在系统上以最优的状态运行。但硬件问题、软件问题、网络环境等的复杂性和多变性，使得对系统的优化变得异常复杂，如何定位性能问题出在哪个方面，是性能优化的一大难题。本章从系统入手，重点讲述由于系统软、硬件配置不当造成的性能问题，并且给出了检测系统故障和优化性能的一般方法和流程。

10.1　系统性能分析的目的

10.1.1　找到系统性能的瓶颈

系统的性能是指操作系统完成任务的有效性、稳定性和响应速度。Linux 系统管理员可能经常会遇到系统不稳定、响应速度慢等问题，例如在 Linux 上搭建了一个 Web 服务，经常出现网页无法打开、打开速度慢等现象。遇到这些问题，就会有人抱怨 Linux 系统不好，其实这些都是表面现象。操作系统完成一个任务是与系统自身设置、网络拓扑结构、路由设备、路由策略、接入设备、物理线路等多个方面密切相关的，任何一个环节出现问题，都会影响整个系统的性能。因此，当 Linux 应用出现问题时，应当从应用程序、操作系统、服务器硬件、网络环境等方面综合排查，定位问题出现在哪个部分，然后集中解决。

10.1.2　提供性能优化方案

查找系统性能瓶颈是个复杂而耗时的过程，需要在应用程序、操作系统、服务器硬件、网络环境等方面进行查找和定位，影响性能最大的是应用程序和操作系统两个方面，因为这两个方面出现的问题不易察觉，隐蔽性很强。而硬件、网络方面出现的问题，一般都能马上定位。一旦找到了系统性能问题，解决起来就非常迅速和容易。例如发现系统硬件存在问题，如果是物理故障，那么更换硬件就可以了，如果是硬件性能不能满足需求，升级硬件就可以了；如果发现是网络问题，比如带宽不够、网络不稳定，只需优化和升级网络即可；如果发现是应用程序问题，修改或优化软件系统即可；而如果是操作系统配置问题，修改系统参数、修改系统配置即可。

可见，只要找到了性能瓶颈，就可以提供性能优化方案，有标准、有目的地进行系统优化。

10.1.3　使系统硬件和软件资源的使用达到平衡

Linux 操作系统是一个开源产品，也是一个开源软件的实践和应用平台，在这个平台下有无数的开源软件支撑，常见的有 Apache、Tomcat、MySQL、PHP 等。开源软件的最大理念是自由、开放，那么 Linux 作为一个开源平台，最终要实现的是通过这些开源软件的支持，以最低廉的成本，达到应用性能的最优化。但是，系统的性能问题并非是孤立的，解决了一个性能瓶颈。可能会出现另一个性能瓶颈。所以说性能优化的最终目的是：在一定范围内使系统的各项资源使用趋于合理并保持一定的平衡，即系统运行良好的时候恰恰就是系统资源达到了一个平衡状态的时候。而在操作系统中，任何一项资源的过度使用都会破坏这种平衡状态，从而导致系统响应缓慢或者负载过高。例如，CPU 资源的过度使用会造成系统中出现大量的等待进程，导致应用程序响应缓慢，而进程的大量增加又会导致系统内存资源的增加，当物理内存耗尽时，系统就会使用虚拟内存，而虚拟内存的使用又会造成磁盘 I/O 的增加并加大 CPU 的开销。因此，系统性能的优化就是在硬件、操作系统、应用软件之间找到一个平衡点。

10.2　分析系统性能涉及的人员

10.2.1　Linux 系统管理人员

在进行性能优化的过程中，系统管理人员承担着很重要的任务，首先，系统管理人员要了解和掌握操作系统的当前运行状态，例如系统负载、内存状态、进程状态、CPU 负荷等，这些信息是检测和判断系统性能的基础和依据；其次，系统管理人员还要掌握系统的硬件信息，例如磁盘 I/O、CPU 型号、内存大小、网卡带宽等参数信息，然后根据这些信息综合评估系统资源的使用情况；第三，作为一名系统管理人员，还要掌握应用程序对系统资源的使用情况，更深入的一点就是要了解应用程序的运行效率，例如是否有程序 bug、内存溢出等问题。通过对系统资源的监控，就能发现应用程序是否存在异常，如果确实是应用程序存在问题，需要把问题立刻反映给程序开发人员，进而改进或升级程序。

性能优化本身就是一个复杂和繁琐的过程，系统管理人员只有了解了系统硬件信息、网络信息、操作系统配置信息和应用程序信息才能有针对性地展开对服务器的性能优化，这就要求系统管理员有充足的理论知识、丰富的实战经验以及缜密分析问题的头脑。

10.2.2　系统架构设计人员

系统性能优化涉及的第二类人员就是应用程序的架构设计人员。如果系统管理人员在经过综合判断后，发现影响性能的是应用程序的执行效率，那么程序架构设计人员就要及时介

入，深入了解程序运行状态。首先，系统架构设计人员要跟踪了解程序的执行效率，如果执行效率存在问题，要找出哪里出现了问题；其次，如果真的是架构设计出现了问题，那么就要马上优化或改进系统架构，设计更好的应用系统架构。

10.2.3 软件开发人员

系统性能优化最后一个环节涉及的是程序或软件开发人员。在系统管理员或架构设计人员找到程序或结构瓶颈后，程序开发人员要马上介入进行相应的程序修改。修改程序要以程序的执行效率为基准，改进程序的逻辑，有针对性地进行代码优化。例如，系统管理人员在系统中发现有条 SQL 语句耗费大量的系统资源，抓取这条执行的 SQL 语句后，发现此 SQL 语句的执行效率太差，是开发人员编写的代码执行效率低造成的，这就需要把这个信息反馈给开发人员。开发人员在了解到这个问题后，可以有针对性地进行 SQL 优化，进而实现程序代码的优化。

从上面这个过程可以看出，系统性能优化一般遵循的流程是：首先系统管理人员查看系统的整体状况，主要从系统硬件、网络设备、操作系统配置、应用程序架构和程序代码 5 个方面进行综合判断。如果发现是系统硬件、网络设备或者操作系统配置问题，系统管理员可以根据情况自主解决；如果发现是程序结构问题，就需要提交给程序架构设计人员；如果发现是程序代码执行问题，就交给开发人员进行代码优化。这样就完成了一个系统性能优化的过程。

10.3 影响 Linux 性能的各种因素

10.3.1 系统硬件资源

1.CPU

CPU 是操作系统稳定运行的根本，CPU 的速度与性能在很大程度上决定了系统整体的性能，因此，CPU 数量越多，主频越高，服务器性能也就相对越好。但事实并非完全如此。

目前大部分 CPU 在同一时间内只能运行一个线程，超线程的处理器可以在同一时间运行多个线程，因此，可以利用处理器的超线程特性提高系统性能。在 Linux 系统下，只有运行 SMP 内核才能支持超线程，但是，安装的 CPU 数量越多，从超线程获得的性能方面的提高就越少。另外，Linux 内核会把多核的处理器当做多个单独的 CPU 来识别，例如两个 4 核的 CPU，在 Linux 系统下会被当做 8 个单核 CPU。但是从性能角度来讲，两个 4 核的 CPU 和 8 个单核的 CPU 并不完全等价。根据权威部门得出的测试结论，前者的整体性能要比后者低 25% ~30%。

可能出现 CPU 瓶颈的应用有邮件服务器、动态 Web 服务器等。对于这类应用，要把 CPU 的配置和性能放在主要位置。

2. 内存

内存的大小也是影响 Linux 性能的一个重要的因素。内存太小，系统进程将被阻塞，应用也将变得缓慢，甚至失去响应；内存太大，导致资源浪费。Linux 系统采用了物理内存和虚拟内存两种方式，虚拟内存虽然可以缓解物理内存的不足，但是占用过多的虚拟内存，应用程序的性能将明显下降。要保证应用程序的高性能运行，物理内存一定要足够大；但是过大的物理内存，会造成内存资源浪费，例如，在一个 32 位处理器的 Linux 操作系统上，超过 8GB 的物理内存都将被浪费。因此，要使用更大的内存，建议安装 64 位的操作系统，同时开启 Linux 的大内存内核支持。

由于处理器寻址范围的限制，在 32 位 Linux 操作系统上，应用程序单个进程最大只能使用 2GB 的内存，这样一来，即使系统有再大的内存，应用程序也无法“享”用。解决的办法就是使用 64 位处理器，安装 64 位操作系统。在 64 位操作系统下，可以满足所有应用程序对内存的使用需求，几乎没有限制。

可能出现内存性能瓶颈的应用有打印服务器、数据库服务器、静态 Web 服务器等，对于这类应用要把内存大小放在主要位置。

3. 磁盘 I/O 性能

磁盘的 I/O 性能直接影响应用程序的性能，在一个有频繁读写操作的应用中，如果磁盘 I/O 性能得不到满足，就会导致应用停滞。好在如今的磁盘采用了很多方法来提高 I/O 性能，比如常见的磁盘 RAID 技术。

RAID 的英文全称为：Redundant Array of Independent Disk，即独立磁盘冗余阵列，简称磁盘阵列。RAID 通过将多块独立的磁盘（物理硬盘）按不同方式组合起来形成一个磁盘组（逻辑硬盘），从而提供比单个硬盘更高的 I/O 性能和数据冗余。

通过 RAID 技术组成的磁盘组，就相当于一个大硬盘，用户可以对它进行分区格式化、建立文件系统等操作，跟单个物理硬盘一模一样，唯一不同的是 RAID 磁盘组的 I/O 性能比单个硬盘要高很多，同时在数据的安全性方面也有很大提升。

根据磁盘组合方式的不同，RAID 可以分为 RAID0、RAID1、RAID2、RAID3、RAID4、RAID5、RAID6、RAID7、RAID0+1、RAID10 等级别，常用的 RAID 级别有 RAID0、RAID1、RAID5、RAID0+1，这里进行简单介绍。

RAID 0：通过把多块硬盘粘合成一个容量更大的硬盘组，提高了磁盘的性能和吞吐量。这种方式成本低，要求至少两块磁盘，但是没有容错和数据修复功能，因而只能用在对数据安全性要求不高的环境中。

RAID 1：也就是磁盘镜像，通过把一个磁盘的数据镜像到另一个磁盘上，最大限度地保证磁盘数据的可靠性和可修复性，具有很高的数据冗余能力，但磁盘利用率只有 50%，因

而，成本最高，多用在保存重要数据的场合。

RAID5：采用了磁盘分段加奇偶校验技术，从而提高了系统的可靠性。RAID5 读出效率很高，写入效率一般，至少需要 3 块盘。允许一块磁盘故障，而不影响数据的可用性。

RAID0+1：把 RAID0 和 RAID1 技术结合起来就成了 RAID0+1，至少需要 4 块硬盘。此种方式的数据除分布在多个盘上外，每个盘都有其镜像盘，提供全冗余能力，同时允许一个磁盘故障，而不影响数据可用性，并具有快速读 / 写能力。

通过了解各个 RAID 级别的性能，可以根据应用的不同特性，选择适合自身的 RAID 级别，从而保证应用程序在磁盘方面达到最优性能。

4. 网络宽带

Linux 下的各种应用，一般都是基于网络的，因此网络带宽也是影响性能的一个重要因素，低速的、不稳定的网络将导致网络应用程序的访问阻塞，而稳定、高速的网络带宽，可以保证应用程序在网络上畅通无阻的运行。幸运的是，现在的网络一般都是千兆带宽或光纤网络，带宽问题对应用程序性能造成的影响正在逐步降低。

10.3.2 操作系统相关资源

基于操作系统的性能优化也是多方面的，可以从系统安装、系统内核参数、网络参数、文件系统等几个方面进行衡量，下面依次进行简单介绍。

1. 系统安装优化

系统优化可以从安装操作系统开始，当安装 Linux 系统时，磁盘的划分、SWAP 内存的分配都直接影响以后系统的运行性能，例如，磁盘分配可以遵循应用的需求：对于读写操作频繁而对数据安全性要求不高的应用，可以把磁盘做成 RAID0；而对于对数据安全性较高，对读写没有特别要求的应用，可以把磁盘做成 RAID1；对于对读操作要求较高，而对写操作无特殊要求，并要保证数据安全性的应用，可以选择 RAID5；对于对读写要求都很高，并且对数据安全性要求也很高的应用，可以选择 RAID0+1。这样通过不同的应用需求设置不同的 RAID 级别，在磁盘底层对系统进行优化操作。

随着内存价格的降低和内存容量的日益增大，对虚拟内存 SWAP 的设定，现在已经没有了所谓虚拟内存是物理内存两倍的要求，但是 SWAP 的设定还是不能忽略，根据经验，如果内存较小（物理内存小于 4GB），一般设置 SWAP 交换分区大小为内存的 2 倍；如果物理内存大于 4GB 小于 16GB，可以设置 SWAP 大小等于或略小于物理内存即可；如果内存大小在 16GB 以上，原则上可以设置 SWAP 为 0，但并不建议这么做，因为设置一定大小的 SWAP 还是有一定作用的。

2. 内核参数优化

系统安装完成后，优化工作并没有结束，接下来还可以对系统内核参数进行优化，不

过内核参数的优化要和系统中部署的应用结合起来整体考虑。例如，如果系统部署的是 Oracle 数据库应用，那么就需要对系统共享内存段（kernel.shmmax、kernel.shmmni、kernel.shmall）、系统信号量（kernel.sem）、文件句柄（fs.file-max）等参数进行优化设置；如果部署的是 Web 应用，那么就需要根据 Web 应用特性进行网络参数的优化，例如修改 net.ipv4.ip_local_port_range、net.ipv4.tcp_tw_reuse、net.core.somaxconn 等网络内核参数。

3. 文件系统优化

文件系统的优化也是系统资源优化的一个重点，在 Linux 下可选的文件系统有 ext2、ext3、xfs、ReiserFS。可根据不同的应用，选择不同的文件系统。

Linux 标准文件系统是从 VFS 开始的，然后是 ext，接着就是 ext2，应该说，ext2 是 Linux 上标准的文件系统，ext3 是在 ext2 基础上增加日志形成的。从 VFS 到 ext3，其设计思想没有太大变化，都是早期 UNIX 家族基于超级块和 inode 的设计理念设计而成的。

XFS 文件系统是 SGI 开发的一个高级日志文件系统，后来移植到了 Linux 系统下。XFS 通过分布处理磁盘请求、定位数据、保持 Cache 的一致性来提供对文件系统数据的低延迟、高带宽的访问。因此，XFS 极具伸缩性，非常健壮，具有优秀的日志记录功能、可扩展性强、快速写入性能等优点。

ReiserFS 是在 Hans Reiser 领导下开发出来的一款高性能的日志文件系统，它通过完全平衡树结构来管理数据，包括文件数据、文件名及日志支持等。与 ext2/ext3 相比，其最大的优点是访问性能和安全性大幅提升。ReiserFS 具有高效、合理利用磁盘空间，先进的日志管理机制，特有的搜寻方式，海量磁盘存储等优点。

10.3.3　应用程序软件资源

应用程序的优化其实是整个优化工程的核心，如果一个应用程序存在 bug，那么即使其他方面都达到了最优状态，整个应用系统还是性能低下的。所以，对应用程序的优化是性能优化过程的重中之重，这就对程序架构设计人员和程序开发人员提出了更高的要求。

10.4　系统性能分析标准和优化原则

性能调优的主要目的是使系统能够有效地利用各种资源，最大可能地发挥应用程序和系统之间的性能融合，使应用高效、稳定地运行。但是，衡量系统资源利用率好坏的标准没有一个严格的定义，针对不同的系统和应用也没有一个统一的说法，因此，这里提供的标准其实是一个经验值，如表 10-1 所示。

表 10-1 系统性能分析标准

影响性能因素	评判标准		
	好	坏	糟糕
CPU	user% + sys%< 70%	user% + sys%= 85%	user% + sys% >=90%
内存	Swap In（si）= 0 Swap Out（so）= 0	Per CPU with 10 page/s	More Swap In & Swap Out
磁盘	iowait % < 20%	iowait % =35%	iowait % >= 50%

其中：

- user% 表示 CPU 处在用户模式下的时间百分比。
- sys% 表示 CPU 处在系统模式下的时间百分比。
- iowait% 表示 CPU 等待输入输出完成时间的百分比。
- Swap In 即 si，表示虚拟内存的页导入，即从 SWAP DISK 交换到 RAM。
- Swap Out 即 so，表示虚拟内存的页导出，即从 RAM 交换到 SWAP DISK。

10.5 几种典型应用对系统资源使用的特点

10.5.1 以静态内容为主的 Web 应用

这类应用的一个主要特点是小文件居多，并且读操作频繁，Web 服务器一般为 Apache 或 Nginx，因为这两个 HTTP 服务器对静态资源的处理非常迅速和高效。在 Web 访问量不大时，可以直接对外提供服务，但是在有很大并发请求时，单一的 Web 服务无法支撑大量的客户端访问，此时就需要由多台 Web 服务器组成负载集群系统。为了实现更高效的访问，在最前端还可以搭建 Cache 服务器，也就是将静态资源文件缓存到操作系统内存中直接进行读操作，因为直接从内存读取数据要比从硬盘读数据效率高很多，所以在 Web 前端搭建 Cache 服务器可以大大提高并发访问性能。常用的 Cache 软件有 Squid、Varinsh 等。

Cache 服务器虽然可以提高访问性能，但要求服务器有很大的内存，当系统内存充足时，可以缓解磁盘随机读的压力；当内存过小或者内存不足时，系统就会使用虚拟内存，而虚拟内存的使用会引起磁盘 I/O 的增大，当磁盘 I/O 增大时，CPU 的开销也随之增加。

在高并发访问时，还存在另外一个问题，就是网络带宽瓶颈，如果客户端访问量很大且带宽不够，就会阻塞网络，影响访问。因此，在构建基于 Web 的网络应用时，网络带宽也是必须考虑的一个因素。

10.5.2 以动态内容为主的 Web 应用

这类应用的一个特点是频繁地执行写操作，例如 Java、PHP、Perl、CGI 等，会导致

CPU 资源消耗严重。因为动态程序的执行要进行编译、读取数据库等操作，而这些操作都会消耗 CPU 资源。因此，一个基于动态程序的 Web 应用，应该选择多个性能较高的 CPU，这将对系统整体性能的提高有很大帮助。

基于动态内容的 Web 应用在高并发访问时，系统执行的进程数会很多，因此要注意负载的分配。由于过多的进程会消耗大量系统内存，如果内存不足，就会使用虚拟内存，而虚拟内存的增加会导致磁盘写操作频繁，进而消耗 CPU 资源。因此要寻求一个硬件资源和软件资源的平衡点，例如配置较大的内存和高性能的 CPU，而在软件方面可通过如 Memcached 之类的软件加快程序与数据库之间的访问效率。

10.5.3　数据库应用

数据库应用的一个主要特点是消耗内存和磁盘 I/O，而对 CPU 的消耗并不是很大，因此最基本的做法就是为数据库服务器配置较大的内存和读写较快的磁盘阵列。例如，可以为数据库服务器的磁盘选择 RAID5、RAID0+1 等 RAID 级别。将 Web Server 与 DB Server 分离也是优化数据库应用的一个常用做法。如果客户端用户对数据库的请求过大，还可以考虑采取数据库的负载均衡方案，通过软件负载均衡或硬件负载均衡的方式提高数据库访问性能。

对于数据库中过大的表，可以考虑进行拆分，也就是将一个大表拆分成多个小表，再通过索引进行关联处理，这样可以避免查询大表造成的性能问题，因为表太大时，查询遍历全表会造成磁盘读操作激增，进而出现读操作等待的情况。同时，数据库中查询语句复杂，大量的 where 子句，order by、group by 排序语句等，容易使 CPU 出现瓶颈。最后，当数据更新时，数据更新量大或更新频繁，也会造成磁盘写操作激增，出现写操作的瓶颈。这些也应该在程序代码中避免。

在日常应用中，还有一种方法可以显著提高数据库服务器的性能，那就是读写分离。同时对数据库进行读和写的操作，是效率极低的一种访问方式，较好的做法是根据读、写的压力和需求，分别建立两台结构完全相同的数据库服务器，将负责写的服务器上的数据，定时复制给负责读的服务器，通过读写的协作提高系统整体性能。

通过缓存方式也可以提高数据库的性能，缓存是数据库或对象在内存中的临时容器，使用缓存可大幅减少数据库的读取操作，改由内存来提供数据。比如可以在 Web Server 和 DB Server 之间增加一层数据缓存层，在系统内存中建立被频繁请求对象的副本，这样一来，不访问数据库也可为程序提供数据。现在应用很广泛的 Memcached 就是基于这个原理。

10.5.4　软件下载应用

静态资源下载服务器的主要特点是带宽消耗严重，同时对存储性能要求也很高。在下载量极高时，可以采用多台、多点服务器分流的形式分担下载负荷。在 HTTP 服务器方面，从高性能和减少服务器部署的角度考虑，推荐采用 Lighttpd HTTP 服务器，而不是采用传统的

Apache 服务器，原因是 Apache 使用阻塞模式的 I/O 操作，性能相对较差，并发能力有限，而 Lighttpd 使用异步 I/O 方式，处理资源下载的并发能力远远超过 Apache。

10.5.5 流媒体服务应用

流媒体主要应用在视频会议、视频点播、远程教育、在线直播等方面，这类应用主要的性能瓶颈是网络带宽和存储系统带宽（主要是读操作）。面对海量用户，如何保障用户接收到高清晰的、流畅的画面，如何最大限度地节省网络带宽，这些都是流媒体应用要解决的首要问题。

对于流媒体服务器的优化，可以从存储策略、传输策略、调度策略、代理服务器缓存策略及流媒体服务器的体系结构设计等几个方面进行考虑。在存储方面，需要对视频的编码格式进行优化，进而节省空间，优化存储性能；在传输方面，可以采用智能流技术控制发送的速率，最大程度地保障用户观看视频的流畅性；在调度方面，可以采用静态调度和动态调度结合的方法；在代理服务器方面，可以采用分段缓存、动态缓存等管理策略；在流媒体体系结构方面，可以采用内存池和线程池技术改善内存消耗和线程过多对性能造成的影响。

10.6 Linux 下常见的性能分析工具

10.6.1 vmstat 命令

vmstat 是 Virtual Meomory Statistics（虚拟内存统计）的缩写，很多 Linux 发行版本都默认安装了此命令工具。利用 vmstat 命令可以对操作系统的内存信息、进程状态、CPU 活动等进行监视，不足之处是无法对某个进程进行深入分析。

vmstat 命令的语法如下：

```
vmstat [-V] [-n] [delay [count]]
```

各个选项及参数含义如下：

- -V，表示打印出版本信息，是可选参数。
- -n，表示在周期性循环输出时，输出的头部信息仅显示一次。
- delay，表示两次输出之间的间隔时间。
- count，表示按照“delay”指定的时间间隔统计的次数，默认为 1。

例如：

```
vmstat 3
```

表示每 3 秒更新一次输出信息，循环输出，按 Ctrl+c 停止输出。

```
vmstat 3 5
```

表示每 3 秒更新一次输出信息，统计 5 次后停止输出。

下面是 vmstat 命令在某个系统中的输出结果：

```
[root@node1 ~]# vmstat 2 3
procs -------memory----------  ---swap--  -----io----  --system--  -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs  us sy  id  wa st
 0  0      0 162240   8304  67032    0    0    13    21 1007   23   0  1  98   0  0
 0  0      0 162240   8304  67032    0    0     1     0 1010   20   0  1 100   0  0
 0  0      0 162240   8304  67032    0    0     1     1 1009   18   0  1  99   0  0
```

上面每项输出的解释如下：

- procs
  - r 列表示运行和等待 CPU 时间片的进程数，这个值如果长期大于系统 CPU 的个数，说明 CPU 不足，需要增加 CPU。
  - b 列表示在等待资源的进程数，比如正在等待 I/O 或者内存交换等。
- memory
  - swpd 列表示切换到内存交换区的内存大小（以 KB 为单位）。如果 swpd 的值不为 0，或者比较大，只要 si、so 的值长期为 0，这种情况一般不用担心，不会影响系统性能。
  - free 列表示当前空闲的物理内存数量（以 KB 为单位）。
  - buff 列表示 buffers cache 的内存数量，一般对块设备的读写才需要缓冲。
  - cache 列表示 page cached 的内存数量，一般作为文件系统进行缓存，频繁访问的文件都会被缓存。如果 cache 值较大，说明缓存的文件数较多，如果此时 io 中的 bi 比较小，说明文件系统效率比较好。
- swap
  - si 列表示由磁盘调入内存，也就是由内存进入内存交换区的内存大小。
  - so 列表示由内存调入磁盘，也就是由内存交换区进入内存的内存大小。

在一般情况下，si、so 的值都为 0，如果 si、so 的值长期不为 0，则表示系统内存不足，需要增加系统内存。

- io 项显示磁盘读写状况
  - bi 列表示从块设备读入数据的总量（即读磁盘）（kb/s）。
  - bo 列表示写到块设备的数据总量（即写磁盘）（kb/s）。

这里设置的 bi+bo 参考值为 1000，如果超过 1000，而且 wa 值较大，则表示系统磁盘 I/O 有问题，应该考虑提高磁盘的读写性能。

- system 显示采集间隔内发生的中断数。
  - in 列表示在某一时间间隔内观测到的每秒设备中断数。
  - cs 列表示每秒产生的上下文切换次数。

上面这两个值越大，由内核消耗的 CPU 时间越多。

- cpu 项显示了 CPU 的使用状态，此列是关注的重点。
  - us 列显示了用户进程消耗的 CPU 时间百分比。us 的值比较高时，说明用户进程消耗的 CPU 时间多，但是如果长期大于 50%，就需要考虑优化程序或算法。
  - sy 列显示了内核进程消耗的 CPU 时间百分比。sy 的值较高时，说明内核消耗的 CPU 资源很多。

根据经验，us+sy 的参考值为 80%，如果 us+sy 大于 80%，说明可能存在 CPU 资源不足。

  - id 列显示了 CPU 处在空闲状态的时间百分比。
  - wa 列显示了 IO 等待所占用的 CPU 时间百分比。wa 值越高，说明 I/O 等待越严重。根据经验，wa 的参考值为 20%，如果 wa 超过 20%，说明 I/O 等待严重。引起 I/O 等待的原因可能是磁盘大量随机读写造成的，也可能是磁盘或者磁盘控制器的带宽瓶颈（主要是块操作）造成的。

综上所述，在对 CPU 的评估中，需要重点注意 procs 项中 r 列的值和 CPU 项中 us、sy 和 id 列的值。

10.6.2 sar 命令

sar 命令很强大，是分析系统性能的重要工具之一。通过 sar 指令，可以全面获取系统的 CPU、运行队列、磁盘 I/O、分页（交换区）、内存、CPU 中断、网络等性能数据。

sar 命令的语法如下：

```
sar [options]  [-o filename] [interval [count] ]
```

各个选项及参数的含义如下：

- options，命令行选项，sar 命令的命令行选项很多，下面只列出常用选项。
  - -A，显示系统所有资源设备（CPU、内存、磁盘）的运行状况。
  - -u，显示系统所有 CPU 在采样时间内的负载状态。
  - -P，显示当前系统中指定 CPU 的使用情况。
  - -d，显示系统所有硬盘设备在采样时间内的使用状况。
  - -r，显示系统内存在采样时间内的使用状况。
  - -b，显示缓冲区在采样时间内的使用情况。
  - -v，显示进程、文件、节点和锁表状态。
  - -n，显示网络运行状态。参数后面可跟 DEV、EDEV、SOCK 和 FULL。DEV 显示网络接口信息，EDEV 显示网络错误的统计数据，SOCK 显示套接字信息，FULL 显示前三参数的所有信息。它们可以单独使用或者一起使用。
  - -q，显示了运行队列的大小，它与系统当时的平均负载相同。
  - -R，显示进程在采样时间内的活动情况。
  - -y，显示终端设备在采样时间内的活动情况。

- -w，显示系统交换活动在采样时间内的状态。
- -o filename，表示将命令结果以二进制格式存放在文件中，filename 是文件名。
- interval，表示采样间隔时间，是必须有的参数。
- count，表示采样次数，是可选参数，默认值是 1。

例如，要查看系统 CPU 的整体负载状况，每 3 秒统计一次，统计 5 次，可以使用以下组合：

```
sar -u 3 5
```

系统的 CPU 计数是从 0 开始的，如果要查看第二个 CPU 的运行负载，使用下面组合：

```
sar -P 1 3 5
```

要查看系统磁盘的读写性能，使用以下组合：

```
sar -d 3 5
```

同理，查看系统内存使用情况、网络运行状态，可以分别使用下面的命令：

```
sar -r 5 2
sar -n DEV 5 3
```

下面是 sar 命令对某个系统的 CPU 统计输出：

```
[root@webserver ~]# sar -u 3 5
Linux 2.6.9-42.ELsmp (webserver)        11/28/2008      _i686_  (8 CPU)

11:41:24 AM      CPU     %user     %nice   %system   %iowait    %steal     %idle
11:41:27 AM      all      0.88      0.00      0.29      0.00      0.00     98.83
11:41:30 AM      all      0.13      0.00      0.17      0.21      0.00     99.50
11:41:33 AM      all      0.04      0.00      0.04      0.00      0.00     99.92
11:41:36 AM      all      0.29      0.00      0.13      0.00      0.00     99.58
11:41:39 AM      all      0.38      0.00      0.17      0.04      0.00     99.41
Average:         all      0.34      0.00      0.16      0.05      0.00     99.45
```

上面每项输出的解释如下：

- %user 列显示了用户进程消耗的 CPU 时间百分比。
- %nice 列显示了运行正常进程所消耗的 CPU 时间百分比。
- %system 列显示了系统进程消耗的 CPU 时间百分比。
- %iowait 列显示了 I/O 等待所占用的 CPU 时间百分比。
- %steal 列显示了在内存相对紧张的环境下 pagein 强制对不同的页面进行的 steal 操作。
- %idle 列显示了 CPU 处在空闲状态的时间百分比。

这个输出是对系统整体 CPU 使用状况的统计，每项的输出都非常直观，并且最后一行 Average 是个汇总行，是上面统计信息的平均值。

需要注意的一点是，第一行的统计信息中包含了 sar 本身的统计消耗，所以 %user 列的值会偏高一点，不过，这不会对统计结果产生太大影响。

在一个多 CPU 的系统中，如果程序使用了单线程，会出现这么一个现象，CPU 的整体使用率不高，但是系统应用却响应缓慢。单线程只使用一个 CPU，导致这个 CPU 占用率为 100%，无法处理其他请求，而其他的 CPU 却闲置，这就导致了整体 CPU 使用率不高而应用缓慢现象的发生。

针对这个问题，可以分开查询系统的每个 CPU，统计每个 CPU 的使用情况。

```
[root@webserver ~]# sar -P 0 3 5
Linux 2.6.9-42.ELsmp (webserver)        11/29/2008      _i686_    (8 CPU)

06:29:33 PM       CPU      %user      %nice    %system    %iowait     %steal      %idle
06:29:36 PM         0       3.00       0.00       0.33       0.00       0.00      96.67
06:29:39 PM         0       0.67       0.00       0.33       0.00       0.00      99.00
06:29:42 PM         0       0.00       0.00       0.33       0.00       0.00      99.67
06:29:45 PM         0       0.67       0.00       0.33       0.00       0.00      99.00
06:29:48 PM         0       1.00       0.00       0.33       0.33       0.00      98.34
Average:            0       1.07       0.00       0.33       0.07       0.00      98.53
```

这个输出是对系统的第一个 CPU 的信息统计，需要注意的是，sar 中对 CPU 的计数是从 0 开始的，因此，“sar -P 0 3 5”表示对系统的第一个 CPU 进行信息统计，“sar -P 4 3 5”则表示对系统的第 5 个 CPU 进行统计。依此类推，可以看出，上面的系统有 8 个 CPU。

10.6.3 iostat 命令

iostat 是 I/O statistics（输入 / 输出统计）的缩写，主要的功能是对系统的磁盘 I/O 操作进行监视。它的输出主要显示磁盘读写操作的统计信息，同时给出 CPU 的使用情况。同 vmstat 一样，iostat 也不能对某个进程进行深入分析，仅对系统的整体情况进行分析。

iostat 一般不随系统安装，要使用 iostat 工具，需要在系统上安装一个 Sysstat 的工具包。Sysstat 是一个开源软件，官方网址为 http://pagesperso-orange.fr/sebastien.godard，可以选择源代码包或 rpm 包的方式安装。安装完毕，系统会多出 3 个命令：iostat、sar 和 mpstat，然后就可以直接在系统下运行 iostat 命令了。

iostat 命令的语法如下：

```
iostat [ -c | -d ] [ -k ] [ -t ] [ -x [ device ] ] [ interval  [ count ] ]
```

各个选项及参数含义如下：

- -c，显示 CPU 的使用情况。
- -d，显示磁盘的使用情况。
- -k，每秒以 KB 为单位显示数据。
- -t，打印出统计信息开始执行的时间。
- -x device，指定要统计的磁盘设备名称，默认为所有的磁盘设备。
- interval，指定两次统计间隔的时间。

- count，按照“interval”指定的时间间隔统计的次数。

看下面的一个输出：

```
[root@webserver ~]# iostat  -c
Linux 2.6.9-42.ELsmp (webserver)        11/29/2008      _i686_  (8 CPU)

avg-cpu:  %user   %nice    %system  %iowait  %steal   %idle
      2.52    0.00     0.30      0.24      0.00    96.96
```

在这里，使用了“-c”参数，只显示系统 CPU 的统计信息，输出中每项代表的含义与 sar 命令的输出项完全相同。

下面通过“iostat –d”命令组合来查看系统磁盘的使用状况：

```
[root@webserver ~]#   iostat -d 2 3
Linux 2.6.9-42.ELsmp (webserver)        12/01/2008      _i686_  (8 CPU)

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               1.87         2.58       114.12    6479462  286537372

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.00         0.00          0          0

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               1.00         0.00        12.00          0         24
```

上面每项输出的解释如下：

- Blk_read/s 表示每秒读取的数据块数。
- Blk_wrtn/s 表示每秒写入的数据块数。
- Blk_read 表示读取的所有块数。
- Blk_wrtn 表示写入的所有块数。

这里需要注意的一点是，上面输出的第一项是系统从启动到统计时的所有传输信息，第二次输出的数据才代表在检测的时间段内系统的传输值。

可以通过 Blk_read/s 和 Blk_wrtn/s 的值对磁盘的读写性能有一个基本的了解：如果 Blk_wrtn/s 值很大，表示磁盘的写操作很频繁，可以考虑优化磁盘或优化程序；如果 Blk_read/s 值很大，表示磁盘的读操作很多，可以将读取的数据放入内存中进行操作。这两个选项的值没有一个固定的大小，根据系统应用的不同，会有不同的值。但是有一个规则还是可以遵循的：长期的、超大的数据读写，肯定是不正常的，这种情况一定会影响系统性能。

“iostat –x”组合还提供了对每个磁盘的单独统计，如果不指定磁盘，默认是对所有磁盘进行统计。例如下面这个输出：

```
[root@webserver ~]#   iostat -x /dev/sda  2 3
Linux 2.6.9-42.ELsmp (webserver)        12/01/2008      _i686_  (8 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
      2.45    0.00     0.30     0.24     0.00    97.03
```

```
Device: rrqm/s  wrqm/s  r/s  w/s  rsec/s  wsec/s avgrq-sz avgqu-sz   await  svctm  %util
sda   0.01     12.48    0.10  1.78  2.58   114.03    62.33    0.07    38.39   1.30   0.24

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3.97    0.00    1.83    8.19    0.00   86.14

Device:rrqm/s wrqm/s   r/s  w/s   rsec/s  wsec/s avgrq-sz avgqu-sz   await  svctm  %util
sda     0.00   195.00  0.00 18.00  0.00  1704.00    94.67     0.04    2.50   0.11   0.20

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           4.04    0.00    1.83    8.01    0.00   86.18

Device: rrqm/s  wrqm/s  r/s  w/s   rsec/s   wsec/s avgrq-sz avgqu-sz   await  svctm  %util
sda    0.00     4.50    0.00   7.00   0.00    92.00    13.14     0.01    0.79   0.14   0.10
```

这个输出基本与“sar –d”相同。需要说明以下几个选项的含义：

- rrqm/s 表示每秒进行合并的读操作数目。
- wrqm/s 表示每秒进行合并的写操作数目。
- r/s 表示每秒完成读 I/O 设备的次数。
- w/s 表示每秒完成写 I/O 设备的次数。
- rsec/s 表示每秒读取的扇区数。
- wsec/s 表示每秒写入的扇区数。

10.6.4 free 命令

free 是监控 Linux 内存使用状况最常用的指令。看下面这一个输出：

```
[root@webserver ~]# free -m
             total       used       free     shared    buffers     cached
Mem:          8111       7185        925          0        243       6299
-/+ buffers/cache:        643       7468
Swap:         8189          0       8189
```

“free –m”表示查看以 M 为单位的内存使用情况，在这个输出中，重点关注的应该是 free 列与 cached 列的输出值。由输出可知，此系统共有 8GB 内存，系统空闲内存还有 925MB，其中，Buffer Cache 占用了 243MB，Page Cache 占用了 6299MB。由此可知，系统缓存了很多的文件和目录，而对于应用程序来说，可以使用的内存还有 7468MB，当然这 7468MB 包含了 Buffer Cache 和 Page Cache 的值。从 swap 项可以看出，交换分区还未使用，所以从应用的角度来说，此系统的内存资源还非常充足。

一般有这样一个经验公式：当应用程序可用内存 / 系统物理内存 >70% 时，表示系统内存资源非常充足，不影响系统性能；当应用程序可用内存 / 系统物理内存 <20% 时，表示系统内存资源紧缺，需要增加系统内存；当 20%< 应用程序可用内存 / 系统物理内存 <70% 时，

表示系统内存资源基本能满足应用需求，暂时不影响系统性能。

free 命令还可以适时监控内存的使用状况，使用“-s”参数可以在指定的时间段内不间断地监控内存的使用情况。例如下面这个输出：

```
[root@webserver ~]# free -b -s 5
             total       used       free     shared    buffers     cached
Mem:    8505901056 7528706048  977195008          0  260112384 6601158656
-/+ buffers/cache:  667435008 7838466048
Swap:   8587149312     163840 8586985472

             total       used       free     shared    buffers     cached
Mem:    8505901056 7526936576  978964480          0  260128768 6601142272
-/+ buffers/cache:  665665536 7840235520
Swap:   8587149312     163840 8586985472

             total       used       free     shared    buffers     cached
Mem:    8505901056 7523987456  981913600          0  260141056 6601129984
-/+ buffers/cache:  662716416 7843184640
Swap:   8587149312     163840 8586985472
```

其中，“-b”表示以千字节（也就是 1024 字节）为单位来显示内存使用情况。

10.6.5　uptime 命令

uptime 是监控系统性能最常用的一个命令，主要用来统计系统当前的运行状况。输出的信息依次为：系统现在的时间，系统从上次开机到现在运行了多长时间，系统目前有多少登录用户，系统在 1 分钟内、5 分钟内、15 分钟内的平均负载。看下面的一个输出：

```
[root@webserver ~]# uptime
18:52:11 up 27 days, 19:44,  2 users,  load average: 0.12, 0.08, 0.08
```

这里需要注意 load average 这个输出值，它的 3 个值的大小一般不能大于系统 CPU 的个数。例如，本输出中系统有 8 个 CPU，如果 load average 的 3 个值长期大于 8 时，说明 CPU 很繁忙，负载很高，可能会影响系统性能；如果偶尔大于 8，倒不用担心，一般不会影响系统性能。相反，如果 load average 的输出值小于 CPU 的个数，则表示 CPU 还有空闲的时间片，比如本例的输出显示，CPU 是非常空闲的。

10.6.6　netstat 命令

netstat 命令用于显示本机网络连接、运行端口、路由表等信息。其语法如下：

```
netstat [选项]
```

该命令的选项说明如表 10-2 所示。

为 0，如果这几个选项的值不为 0，并且很大，那么网络质量肯定有问题，网络传输性能也一定会下降。

当网络传输存在问题时，可以检测网卡设备是否存在故障，如果可能的话，应升级为千兆网卡或光纤网络。还可以检查网络部署环境是否合理。

在网络不通或网络异常时，首先想到的就是检查系统的路由表信息。“netstat -r”的输出结果与 route 命令的输出完全相同。看下面的一个实例：

```
[root@webserver ~]#  netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.10.1.0       *               255.255.255.0   U         0 0          0 eth0
192.168.200.0   *               255.255.255.0   U         0 0          0 eth1
169.254.0.0     *               255.255.0.0     U         0 0          0 eth1
default         10.10.1.254     0.0.0.0         UG        0 0          0 eth0
```

关于输出中每项的具体含义显示得很明确，这里不再多讲。这里我们重点关注的是 default 行对应的值，表示系统的默认路由，对应的网络接口为 eth0。

10.6.7　top 命令

top 命令提供了实时对系统处理器状态的监控，能够实时显示系统中各个进程的资源占用状况。该命令可以按照 CPU 的使用、内存的使用和执行时间对系统任务进程进行排序显示。同时 top 命令还可以通过交互式命令进行设定显示。通过 top 命令可以查看即时活跃的进程，类似于 Windows 的任务管理器。

top 命令的语法如下：

```
top [选项]
```

top 的选项很多，其常用的选项如表 10-3 所示。

表 10-3　top 的常用选项说明

选项	含　义
-d	指定每两次屏幕信息刷新之间的时间间隔
-i	不显示闲置或僵死的进程信息
-c	显示进程的整个命令路径，而不是只显示命令名称
-s	使 top 命令在安全模式下运行，此时 top 的交互式指令被取消，避免潜在危险
-b	分屏显示输出信息，结合“-n”选项可以将屏幕信息输出到文档
-n	top 输出信息更新的次数，完成后将退出 top 命令

top 命令中除了以上这些选项外，还有很多交互式命令，这些交互式命令就是在 top 命令执行过程中使用的一些命令。这些命令都是单个字母，从应用角度来讲，熟悉这些交互式命令至关重要。

表 10-4 就是交互式命令及其代表的具体含义。

表 10-4 交互式命令及其代表的具体含义

交互命令	表 示 含 义
h 或 ?	显示帮助信息，给出交互式命令的一些总结或说明总结
k	终止一个进程，系统将提示用户输入一个需要终止进程的 PID
i	忽略闲置进程和僵死进程，这是一个开关式命令
s	改变 top 输出信息两次刷新之间的时间，系统将提示输入新的时间，单位是秒，如果是小数，就换算成毫秒。如果输入 0，系统输出将不断被刷新，默认刷新时间是 5 秒。需要注意的是，如果设置的时间太小，可能会引起系统不断地刷新，无法看清输出的显示情况，而且系统负载也会加大
o 或者 O	改变 top 输出信息中显示项目的顺序。按小写的 a~z 键可以将相应的列向右移动，而按大写的 A~Z 键可以将相应的列向左移动，最后按回车键确定
f 或者 F	从当前显示列表中添加或删除项目。按 f 键之后会显示列的列表，按 a~z 之间的任意键即可显示或隐藏对应的列，最后按回车键确定
m	切换显示内存信息
t	切换显示进程和 CPU 状态信息
r	重新设置一个进程的优先级，系统提示用户输入需要改变的进程 PID 以及需要设置的进程优先级值。输入一个正值将使优先级降低，反之则使该进程拥有更高的优先权。默认值是 10
l	切换显示平均负载和启动时间信息
q	退出 top 显示
c	切换显示完整命令行和命令名称信息
M	根据驻留内存大小进行排序输出
P	根据 CPU 使用百分比大小进行排序输出
T	根据时间 / 累计时间进行排序输出
S	切换到累计模式
W	将当前 top 设置写入 ~/.toprc 文件中

下面通过一个具体的例子来解释 top 命令中每个选项的含义。

查看当前系统活动的进程，如图 10-1 所示。

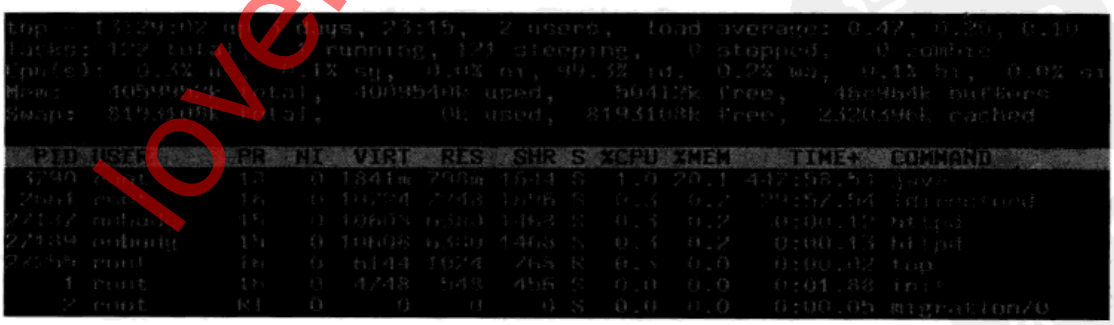

图 10-1 查看当前系统活动的进程

从图 10-1 中可以看到，top 的输出分为两个部分：统计信息区和进程信息区，即前 5 行

显示为统计信息区，后面几行的为进程信息区。下面分别介绍。

（1）统计信息区

第一行为任务队列信息，含义如下：

- 13:29:02，表示当前系统时间。
- up 3 days，23:15，表示系统已经启动了 3 天 23 小时 15 分钟。
- 2 users，当前登录系统的用户数。
- load average: 0.47，0.20，0.10，表示系统平均负载，3 个数值分别表示 1 分钟、5 分钟、15 分钟前到现在的系统平均负载值。

第二、三两行为进程和 CPU 信息，具体含义如下：

- Tasks: 122 total，进程的总数。
- 1 running，正在运行的进程数。
- 121 sleeping，处于休眠的进程数。
- 0 stopped，停止的进程数。
- 0 zombie，僵死的进程数。
- Cpu(s): 0.3% us，表示用户进程占用 CPU 的百分比。
- 0.1% sy，系统进程占用 CPU 的百分比。
- 0.0% ni，用户进程空间内改变过优先级的进程占用 CPU 的百分比。
- 99.3% id，空闲 CPU 占用的百分比。
- 0.2% wa：等待输入输出的进程占用 CPU 的百分比。

最后两行输出的是内存信息，具体含义如下：

- Mem: 4059952k total，系统的物理内存大小。
- 4009540k used，已经使用的物理内存大小。
- 50412k free，目前空余内存大小。
- 468964k buffers，用作内核缓冲区的内存大小。
- Swap: 8193108k total，交换分区的内存大小。
- 0k used，已经使用的交换分区大小。
- 8193108k free，空闲的交换分区大小。
- 2320396k cached，高速缓存的大小。

（2）进程信息区

进程信息区显示了每个进程的运行状态。先来看一下每列输出的含义：

- PID，进程的 id。
- USER，进程所有者的用户名。
- PR，进程优先级。
- NI：nice 值，负值表示高优先级，正值表示低优先级。
- VIRT，进程使用的虚拟内存总量，单位为 KB。VIRT=SWAP+RES。

- RES，进程使用的、未被换出的物理内存大小，单位为 KB。RES=CODE+DATA。
- SHR，共享内存大小，单位为 KB。
- S，进程状态，D 表示不可中断的睡眠状态，R 表示运行状态，S 表示睡眠状态，T 表示跟踪 / 停止，Z 表示僵死进程。
- %CPU，上次更新到现在的 CPU 时间占用百分比。
- %MEM，进程占用的物理内存百分比。
- TIME+，进程使用的 CPU 时间总计，单位为 1/100 秒。
- COMMAND，正在运行进程的命令名或命令路径。

10.7 基于 Web 应用的性能分析及优化案例

10.7.1 基于动态内容为主的网站优化案例

1. 网站运行环境说明

硬件环境：1 台 IBM x3850 服务器，单个双核 Xeon 3.0G CPU，2GB 内存，3 块 72GB SCSI 磁盘。

操作系统：CentOS 5.4。

网站架构：Web 应用是基于 LAMP 架构，所有服务都在一台服务器上部署。

2. 性能问题现象及处理措施

（1）现象描述

网站在上午 10 点左右和下午 3 点左右访问高峰时，网页无法打开，重启服务后，网站在一段时间内能正常服务，但过一会又变得响应缓慢，最后网页彻底无法打开。

（2）检查配置

首先检查系统资源状态，发现服务出现故障时系统负载极高，内存基本耗尽。接着检查 Apache 配置文件 httpd.conf，发现“MaxClients”选项值被设置为 2000，并且打开了 Apache 的 KeepAlive 特性。

（3）处理措施

根据上面的检查，初步判断是 Apache 的“MaxClients”选项配置不当引起的，因为系统内存仅有 2GB 大小，而“MaxClients”选项被配置为 2000，过多的用户访问进程耗尽了系统内存。然后，修改 httpd.conf 配置文件的“MaxClients”选项，将此值由 2000 降到 1500。继续观察发现，网站还是频繁宕机，于是将“MaxClients”选项值降到 1024。观察一段时间发现，网站服务宕机时间间隔加长了，不像以前那么频繁了，但是系统负载还是很高，网页访问速度极慢。

3. 第一次分析优化

既然是由系统资源耗尽导致的网站服务失去响应，那么就深入分析系统资源的使用情况。通过 uptime、vmstat、top、ps 等命令的联合使用，得出如下结论：

（1）结论描述

系统平均负载很高，通过 uptime 输出的系统“load average”值都在 10 以上，而 CPU 资源也消耗严重，这是造成网站响应缓慢或长时间没有响应的主要原因，而导致系统资源消耗过高的主要因素是用户进程消耗资源严重。

（2）原因分析

通过 top 命令发现，每个 Apache 子进程消耗将近 6 ～ 8MB 左右内存，这是不正常的。根据经验，在正常情况下每个 Apache 子进程消耗的内存在 1MB 左右。结合 Apache 输出日志发现，网站首页访问频率最高，也就是说首页程序代码可能存在问题。于是检查首页的 PHP 代码，发现首页的页面非常大，图片很多，并且由全动态的程序组成，这样每次用户访问首页都要多次查询数据库，而查询数据库是个非常耗费 CPU 资源的过程，并且首页 PHP 代码也没有相应的缓存机制，每个用户请求都要重新进行数据库查询操作，数据库查询负荷有多高可想而知。

（3）处理措施

修改首页 PHP 代码，缩减页面大小，并且对访问频繁的操作增加缓存机制，尽量减少程序对数据库的访问。

4. 第二次分析优化

通过前面简单优化，系统服务宕机现象出现次数减少很多，但是在访问高峰时网站偶尔还会无法正常访问。这次仍然从分析系统资源使用状况入手，发现系统内存资源消耗过大，并且磁盘 I/O 有等待问题，于是得出如下结论：

（1）原因分析

内存消耗过大，肯定是用户访问进程数过多导致的，在没有优化 PHP 代码之前，每个 Apache 子进程消耗 6 ～ 8MB 内存，如果设置 Apache 的最大用户数为 1024，那么内存耗尽是必然的。当物理内存耗尽时，虚拟内存就会启用，频繁地使用虚拟内存，肯定会出现磁盘 I/O 等待问题，最终导致 CPU 资源耗尽。

（2）处理措施

通过上面对 PHP 代码的优化，每个 Apache 子进程消耗的内存资源基本维持在 1 ～ 2MB 左右，因此修改 Apache 配置文件 httpd.conf 中的“MaxClients”选项值为“600”，同时把 Apache 配置中的“KeepAlive”特性关闭，这样 Apache 进程数大量减少，基本维持在 500 ～ 600 之间，虽然偶尔也会使用虚拟内存，但是 Web 服务正常了，服务宕机问题也很少出现了。

5. 第三次分析优化

经过前两次的优化，网站基本运行正常，但是在访问高峰时偶尔还会出现站点无法访问

的现象。继续进行问题分析，通过命令查看系统资源，发现仍是 CPU 资源耗尽导致的，但是与前两次又有所不同。

（1）原因分析

通过观察后台日志，发现 PHP 程序有频繁访问数据库的操作，大量的 SQL 语句中有 where、order by 等子句，同时，数据库查询过多，大部分都是复杂查询，一般都需要遍历全表，而大量的表没有建立索引，这样的程序代码导致 MySQL 数据库负荷过高，而 MySQL 数据库和 Apache 部署在同一台服务器上，这也是导致服务器消耗 CPU 资源过高的原因。

（2）处理措施

优化程序中的 SQL 语句，增加 where 子句上的匹配条件，减少遍历全部的查询。同时在 where 和 order by 子句的字段上建立索引，并且增加程序缓存机制。通过这次优化，网站运行基本处于正常状态，再也没有出现宕机的现象。

6. 第四次优化分析

通过前面三次优化以后，网站在程序代码、操作系统、Apache 等方面的优化空间越来越小。要避免出现服务器宕机现象，并且保证网站稳定、高效、快速的运行，可以从网站结构上进行优化，也就是将 Web 和数据库分离部署。可以增加一台专用的数据库服务器，单独部署 MySQL 数据库。随着访问量的增加，如果前端无法满足访问请求，还可以增加多台 Web 服务器，Web 服务器之间进行负载均衡部署，解决前端性能瓶颈。如果在数据库端还存在读写压力，还可以继续增加一台 MySQL 服务器，将 MySQL 进行读写分离部署。这样，一套高性能、高可靠的网站系统就构建起来了。

10.7.2 基于动态、静态内容结合的网站优化案例

1. 网站运行环境说明

硬件环境：两台 IBM x3850 服务器，单个双核 Xeon 3.0G CPU，4GB 内存，3 块 72GB SCSI 磁盘。

操作系统：CentOS 5.4。

网站架构：Web 应用是基于 J2EE 架构的电子商务应用，Web 端应用服务器是 Tomcat，采用 MySQL 数据库，Web 和数据库独立部署在两台服务器上。

2. 性能问题现象以及处理措施

（1）现象描述

网站访问高峰时，网页无法打开，重启 Java 服务后，网站可以正常运行一段时间，但过一会又变得响应缓慢，最后网页彻底无法打开。

（2）检查配置

首先检查系统资源状态，发现服务出现故障时系统负载极高，CPU 满负荷运行，Java 进

程占用了系统 99% 的 CPU 资源，但内存资源占用不大。接着检查应用服务器信息，发现只有一个 Tomcat 在运行 Java 程序。再查看 Tomcat 配置文件 server.xml，发现 server.xml 文件中的参数都是默认配置，没有进行任何优化。

（3）处理措施

server.xml 文件的默认参数需要根据应用的特性进行适当的修改，例如可以修改“connectionTimeout“、“maxKeepAliveRequests”、“maxProcessors” 等几个 Tomcat 配置文件的参数，适当加大这几个参数值。修改参数值后，继续观察发现，网站服务宕机时间间隔加长了，不像以前那么频繁，但是 Java 进程消耗 CPU 资源还是很严重，网页访问速度极慢。

3. 第一次分析优化

既然 Java 进程消耗 CPU 资源严重，那么需要查看到底是什么导致 Java 消耗资源严重。通过 lsof、netstat 命令发现有大量的 Java 请求等待信息，然后查看 Tomcat 日志，发现有大量报错信息、日志提示和数据库连接超时，最终无法连接到数据库。同时，访问网站静态资源，也无法访问，于是得出如下结论：

（1）原因分析

Tomcat 本身就是一个 Java 容器，是使用连接 / 线程模型处理业务请求的，主要用于处理 Jsp、servlet 等动态应用，虽然它也能当作 HTTP 服务器，但是处理静态资源的效率很低，远远比不上 Apache 或 Nginx。从前面观察到的现象分析，可以初步判断是 Tomcat 无法及时响应客户端的请求，进而导致请求队列越来越多，直到 Tomcat 彻底崩溃。对于一个正常的访问请求来说，服务器接收到请求后，会把请求交给 Tomcat 去处理，Tomcat 接着执行编译、访问数据库等操作，然后把信息返回给客户端。客户端接收到信息后，Tomcat 就关闭这个请求链接，这样一个完整的访问过程就结束了。而在高并发访问状态下，很多的请求瞬间都交给 Tomcat 处理，这样 Tomcat 还没有完成第一个请求，第二个请求就来了，接着是第三个，等等。这样越积越多，Tomcat 最终失去响应，Java 进程就会处于僵死状态，资源无法释放，这就是根本原因。

（2）处理措施

要优化 Tomcat 性能，需要从结构上进行重构。首先，加入 Apache 支持，由 Apache 处理静态资源，由 Tomcat 处理动态请求，Apache 服务器和 Tomcat 服务器之间使用 Mod_JK 模块进行通信。使用 Mod_JK 模块的好处是：它可以定义详细的资源处理规则，根据动态、静态网站的特点，将静态资源文件全部交给 Apache 处理，而动态请求通过 Mod_JK 模块传给 Tomcat 去处理。通过 Apache+JK+Tomcat 的整合，可以大幅度提高 Tomcat 应用的性能。

4. 第二次分析优化

经过前面的优化措施，Java 资源偶尔会增高，但是一段时间后又会自动降低，这属于正常状态。而在高并发访问情况下，Java 进程有时还会出现资源上升无法下降的情况，通过查看 Tomcat 日志，综合分析得出如下结论：

要获得更高、更稳定的性能，单一的 Tomcat 应用服务器有时会无法满足需求，因此要结合 Mod_JK 模块运行基于 Tomcat 的负载均衡系统，这样前端由 Apache 负责用户请求的调度，后端又多个 Tomcat 负责动态应用的解析操作。通过将负载均分配给多个 Tomcat 服务器，网站的整体性能会有一个质的提升。

10.8 本章小结

本章通过理论与实践相结合的方法，系统地讲述了 Linux 服务器的性能分析原则和优化方法。首先介绍了系统性能分析的目的和意义，接着讲解了影响 Linux 性能的各种因素，然后分析了常见的几种 Web 应用系统使用系统资源的特点。接下来详细介绍了 Linux 下几个常见的性能优化分析工具的使用，最后通过两个具体的案例从整体上一步一步演示了 Web 应用系统的优化过程。

系统优化是个非常大的话题，涉及的知识面广，分析过程也较复杂，这对系统优化人员提出了非常高的要求。但是万变不离其宗，只要掌握了分析问题的步骤和思考问题的方法，优化系统的过程也将变得十分轻松。

第 5 篇

集群高级应用篇

第 11 章　构建高可用的 LVS 负载均衡集群

本章主要介绍高可用 LVS 负载均衡集群系统的搭建，首先介绍 LVS 的组成和特点，然后介绍高可用 LVS 集群系统的拓扑结构，接着通过 3 个实例详细介绍如何通过 heartbeat、Keepalived 及 piranha 来构建高可用 LVS 集群，最后，总结通过这 3 种方式构建高可用 LVS 集群的优缺点。本章实践性强，实例都是真实环境的模拟。学完本章，读者能够对 LVS 集群有更深的理解和认识。

11.1　LVS 集群的组成与特点

Linux 虚拟服务器（Linux Virtual Server，LVS），是一个由章文嵩开发的一款自由软件。利用 LVS 可以实现高可用的、可伸缩的 Web、Mail、Cache 和 Media 等网络服务。并在此基础上开发支持庞大用户数的、可伸缩的、高可用的电子商务应用。LVS 自 1998 年发展到现在，已经变得比较成熟，目前广泛应用在各种网络服务和电子商务应用中。

LVS 具有很好的可伸缩性、可靠性和可管理性，通过 LVS 要实现的最终目标是：利用 Linux 操作系统和 LVS 集群软件实现一个高可用、高性能、低成本的服务器应用集群。

11.1.1　LVS 集群的组成

利用 LVS 架设的服务器集群系统由 3 个部分组成：最前端的是负载均衡层（这里用 Load Balancer 表示），中间是服务器群组层（用 Server Array 表示），底端是数据共享存储层（用 Shared Storage 表示）。在用户看来，整个 LVS 集群系统的所有内部应用结构都是透明的，最终用户只是在使用一个虚拟服务器提供的高性能服务。

LVS 体系结构如图 11-1 所示。

下面对 LVS 的各个组成部分进行详细介绍。

- 负载均衡层：位于整个集群系统的最前端，由一台或多台负载调度器（Director Server）组成。LVS 核心模板 IPVS 就安装在 Director Server 上，而 Director 的主要作用类似于一个路由器，它含有为完成 LVS 功能所设定的路由表，通过这些路由表把用户的请求分发给服务器群组层的应用服务器（Real Server）。同时，在 Director Server 上还要安装对 Real Server 的监控模块 Ldirectord，此模块用于监测各个 Real Server 服务的健康状况。在 Real Server 不可用时可以将其从 LVS 路由表中剔除，在恢复时重新加入。

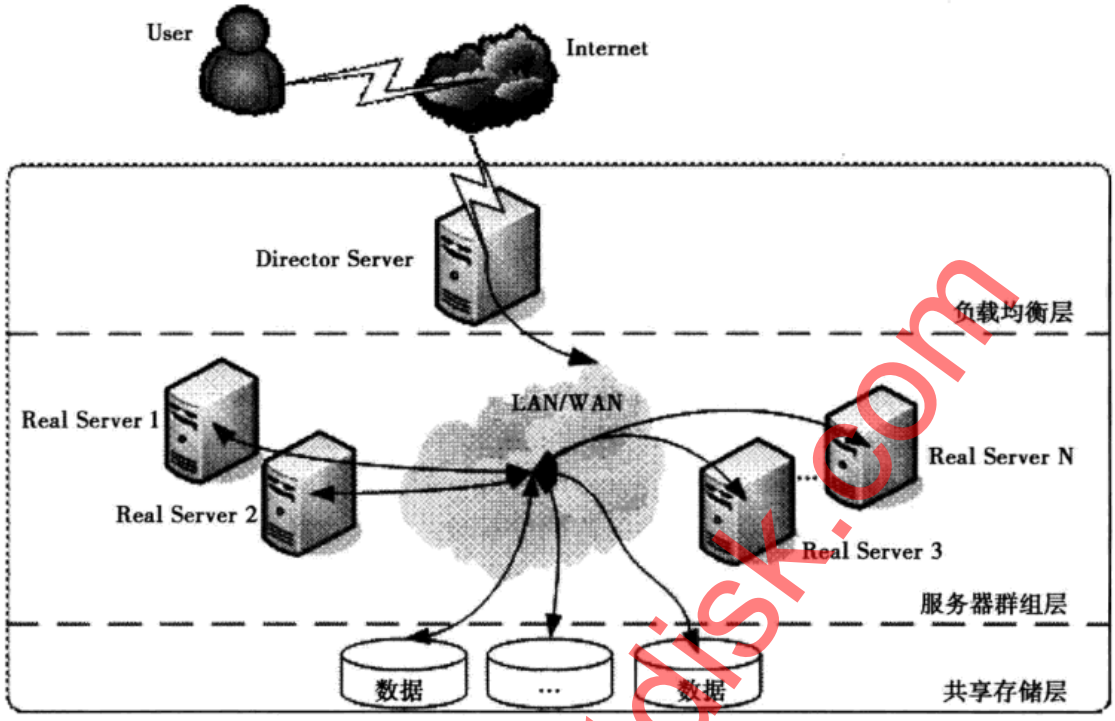

图 11-1　LVS 体系结构

- 服务器群组层：由一组实际运行应用服务的机器组成，Real Server 可以是 Web 服务器、Mail 服务器、FTP 服务器、DNS 服务器、视频服务器中的一个或多个，每个 Real Server 之间通过高速的 LAN 或分布在各地的 WAN 相连接。在实际的应用中，Director Server 也可以同时兼任 Real Server 的角色。
- 共享存储层：是为所有 Real Server 提供共享存储空间和内容一致性的存储区域，一般由磁盘阵列设备组成。为了提供内容的一致性，一般可以通过 NFS 网络文件系统共享数据，但是 NFS 在繁忙的业务系统中，性能并不是很好，此时可以采用集群文件系统，例如 Red Hat 的 GFS 文件系统，Oracle 提供的 OCFS2 文件系统等。

从整个 LVS 结构可以看出，Director Server 是整个 LVS 的核心。目前，用于 Director Server 的操作系统只有 Linux 和 FreeBSD，Linux 2.6 内核完全内置了 LVS 的各个模块，不用任何设置就可以支持 LVS 功能。

对于 Real Server，几乎所有的系统平台，如 Linux、Windows、Solaris、AIX、BSD 系列等都能很好地支持它。

11.1.2　LVS 集群的特点

1. IP 负载均衡与负载调度算法

（1）IP 负载均衡技术

负载均衡技术有很多实现方案，有基于 DNS 域名轮流解析的方法，有基于客户端调度

访问的方法，有基于应用层系统负载的调度方法，还有基于IP地址的调度方法。在这些负载调度算法中，执行效率最高的是IP负载均衡技术。

LVS的IP负载均衡技术是通过IPVS模块来实现的。IPVS是LVS集群系统的核心软件，它的主要作用是：安装在Director Server上，同时在Director Server上虚拟出一个IP地址，用户必须通过这个虚拟的IP地址访问服务器。这个虚拟IP一般称为LVS的VIP，即Virtual IP。访问的请求首先经过VIP到达负载调度器，然后由负载调度器从Real Server列表中选取一个服务节点响应用户的请求。

在用户的请求到达负载调度器后，调度器如何将请求发送到提供服务的Real Server节点，而Real Server节点如何返回数据给用户，是IPVS实现的重点技术。IPVS实现负载均衡的方式有3种，分别是NAT、TUN和DR。下面进行详细介绍。

- VS/NAT：即Virtual Server via Network Address Translation，也就是网络地址翻译技术实现虚拟服务器。当用户请求到达调度器时，调度器将请求报文的目标地址（即虚拟IP地址）改写成选定的Real Server地址，同时将报文的目标端口也改成选定的Real Server的相应端口，最后将报文请求发送到选定的Real Server。在服务器端得到数据后，Real Server将数据返回给用户时，需要再次经过负载调度器将报文的源地址和源端口改成虚拟IP地址和相应端口，然后把数据发送给用户，完成整个负载调度过程。

可以看出，在NAT方式下，用户请求和响应报文都必须经过Director Server地址重写，当用户请求越来越多时，调度器的处理能力将成为瓶颈。

- VS/TUN：即Virtual Server via IP Tunneling，也就是通过IP隧道技术实现虚拟服务器。这种方式的连接调度和管理与VS/NAT方式一样，只是报文转发方法不同。在VS/TUN方式中，调度器采用IP隧道技术将用户请求转发到某个Real Server，而这个Real Server将直接响应用户的请求，不再经过前端调度器。此外，对Real Server的地域位置没有要求，可以和Director Server位于同一个网段，也可以在独立的一个网络中。因此，在TUN方式中，调度器将只处理用户的报文请求，从而使集群系统的吞吐量大大提高。
- VS/DR：即Virtual Server via Direct Routing，也就是用直接路由技术实现虚拟服务器。这种方式的连接调度和管理与前两种一样，但它的报文转发方法又有所不同，VS/DR通过改写请求报文的MAC地址，将请求发送到Real Server，而Real Server将响应直接返回给客户，免去了VS/TUN中的IP隧道开销。这种方式是3种负载调度方式中性能最好的，但是要求Director Server与Real Server必须由一块网卡连在同一物理网段上。

（2）负载调度算法

前面介绍过，负载调度器是根据各个服务器的负载情况，动态地选择一台Real Server响应用户请求。那么动态选择是如何实现呢？其实就是通过这里要说的负载调度算法。根据不

同的网络服务需求和服务器配置，IPVS 实现了 8 种负载调度算法。这里详细讲述最常用的 4 种调度算法。

❑ 轮叫调度（Round Robin）。

“轮叫”调度也叫 1:1 调度，调度器通过“轮叫”调度算法将外部用户请求按顺序 1:1 地分配到集群中每个 Real Server 上。这种算法平等地对待每一台 Real Server，而不管服务器上实际的负载状况和连接状态。

❑ 加权轮叫调度（Weighted Round Robin）。

“加权轮叫”调度算法根据 Real Server 的不同处理能力来调度访问请求。可以对每台 Real Server 设置不同的调度权值，对性能相对较好的 Real Server 可以设置较高的权值，而对处理能力较弱的 Real Server，可以设置较低的权值。这样保证了处理能力强的服务器处理更多的访问流量，充分合理地利用了服务器资源。同时，调度器还可以自动查询 Real Server 的负载情况，并动态地调整其权值。

❑ 最少连接调度（Least Connections）。

“最少连接”调度算法动态地将网络请求调度到已建立的连接数最少的服务器上。如果集群系统的真实服务器具有相近的系统性能，采用“最小连接”调度算法可以较好地均衡负载。

❑ 加权最少连接调度（Weighted Least Connections）。

“加权最少连接调度”是“最少连接调度”的超集。每个服务节点可以用相应的权值表示其处理能力，而系统管理员可以动态地设置相应的权值，默认权值为 1。加权最小连接调度在分配新连接请求时尽可能使服务节点的已建立连接数和其权值成正比。

其他 4 种调度算法分别为：基于局部性的最少连接（Locality-Based Least Connections）、带复制的基于局部性最少连接（Locality-Based Least Connections with Replication）、目标地址散列（Destination Hashing）和源地址散列（Source Hashing）。如果想深入了解这 4 种调度策略，可以登录 LVS 中文站点 zh.linuxvirtualserver.org，查阅更详细的信息。

2. 高可用性

LVS 是一个基于内核级别的应用软件，因此具有很高的处理性能。由 LVS 构建的负载均衡集群系统具有优秀的处理能力，每个服务节点的故障不会影响整个系统的正常使用，又能够实现负载的合理均衡，使应用具有超高负荷的服务能力，可支持上百万个并发连接请求。如配置百兆网卡，采用 VS/TUN 或 VS/DR 调度技术，整个集群系统的吞吐量可高达 1Gbit/s；又如配置千兆网卡，系统的最大吞吐量可接近 10Gbit/s。

3. 高可靠性

LVS 负载均衡集群软件已经在企业和学校中得到了很好的普及，国内外很多大型的、关键性的 Web 站点也都采用了 LVS 集群软件，所以它的可靠性在实践中得到了很好印证。有很多由 LVS 构成的负载均衡系统，运行很长时间，从未进行过重新启动。这些都说明了 LVS 的高稳定性和高可靠性。

4. 适用环境

目前LVS仅支持Linux和FreeBSD系统作为前端Director Server，但是支持大多数的TCP和UDP协议。支持TCP协议的应用有：HTTP、HTTPS、FTP、SMTP、POP3、IMAP4、PROXY、LDAP和SSMTP等；支持UDP协议的应用有：DNS、NTP、ICP、视频和音频流播放协议等。

LVS对Real Server的操作系统没有任何限制，Real Server可运行在任何支持TCP/IP的操作系统上，包括Linux，各种UNIX（如FreeBSD、Sun Solaris、HP Unix等），Mac OS和Windows等。

5. 开源软件

LVS集群软件是按GPL（GNU Public License）许可证发行的自由软件，因此，使用者可以得到软件的源代码，并且可以根据自己的需要进行各种修改，但是修改必须以GPL方式发行。

11.1.3 LVS集群系统的优缺点

LVS是一款自由软件，任何人都可以免费获取并使用它，而且Linux也是一个开源操作系统，这二者的组合大大节约了企业的应用成本。同时LVS具有高稳定性和高可靠性，在高并发和高吞吐量下，具有高负荷处理能力，当某个服务节点出现故障时，并不影响整个系统服务的正常运行。这些优点使LVS已经广泛应用在企业、教育行业以及很多知名网站。

细心的读者可能已经发现了，LVS在具有上述优点的同时，还存在一个致命的缺点，从图11-1可以清楚地看出，所有的用户请求都经过Director Server将任务分发到各个服务器节点，那么，当只有一台Director Server时，将会出现单点故障点，如果这个Director Server出现故障，整个LVS系统将陷入瘫痪状态。

虽然Director Server仅完成用户请求的分发处理，负载并不是很大，但是对于一个健壮的集群系统来说，单点故障是绝对不允许的。要避免这种单点故障，最实用、最简单的办法就是对Director Server进行高可用集群，常见的方案就是为Director Server做一个双机热备：正常状态下主Director Server工作，备用Director Server监控主Director Server的状态，当主Director Server出现异常或者故障时，备用Director Server马上接过主Director Server的工作，负责对用户请求进行分发处理。这样就避免了一台Director Server的单点故障问题，保证了负载均衡端持续地提供服务。

11.2 高可用LVS负载均衡集群体系结构

单一的Director Server可能会造成真个LVS集群系统的单点故障，为了解决这个问题，就需要保证Director Server的高可用性，最常用的方法就是在负载均衡层构建Director

Server 双机热备系统。

高可用的 LVS 负载均衡集群体系结构如图 11-2 所示。

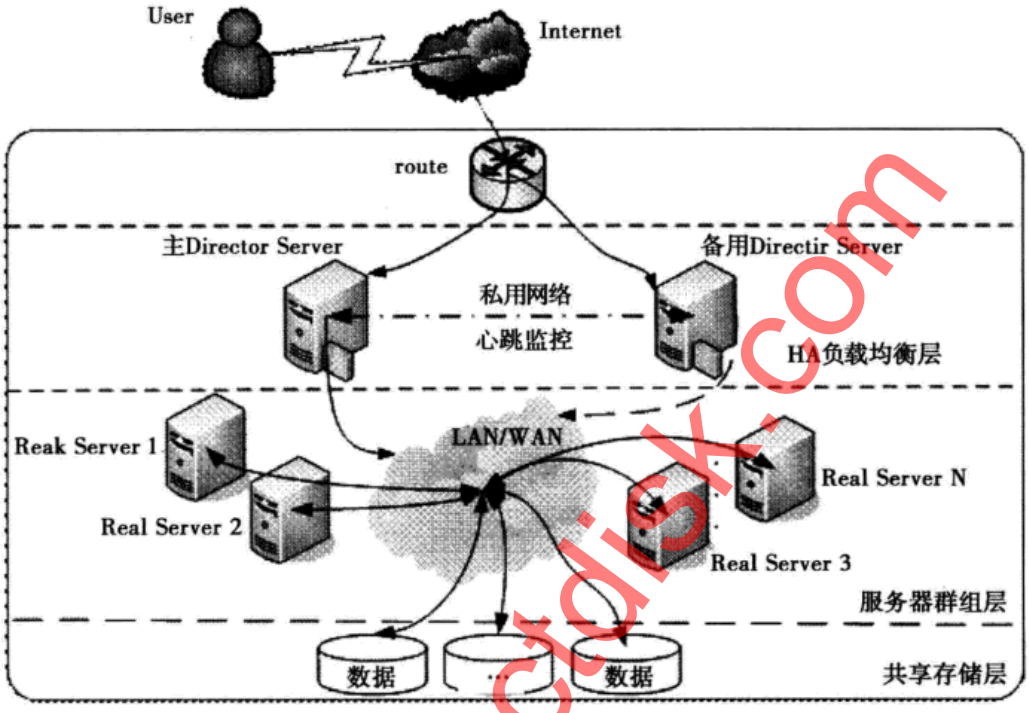

图 11-2　高可用的 LVS 负载均衡集群体系结构

从图 11-2 可以看出，整个体系结构仍然分为三层，在 HA 负载均衡层由主、备两台 Director Server 构成双机热备系统，双机之间通过心跳线连接。在正常状态下主 Director Server 使用虚拟 IP 接收用户请求，并根据设定好的策略和算法将请求分发给各个服务节点，备用 Director Server 负责监控主 Director Server 的运行状态。当主 Director Server 发生异常或出现故障时，备用 Director Server 负责接管主 Director Server 的虚拟 IP 和服务并继续接收用户请求和分发处理。通过这种相互监控策略，任意一方主机出故障时，另一方都能够将 IP 和服务接管，这就保证了负载均衡层业务请求的不间断运行。

服务器群组层和共享存储层实现的功能与图 11-1 完全相同，不再讲述。

11.3　高可用性软件 Heartbeat 与 Keepalived

11.3.1　开源 HA 软件 Heartbeat 的介绍

heartbeat 是 Linux-HA 项目中的一个组件，也是目前开源 HA 项目中最成功的一个例子，它提供了所有 HA 软件需要的基本功能，比如心跳检测和资源接管，监测集群中的系统服务，

在群集的节点间转移共享 IP 地址的所有者等。自 1999 年开始到现在，heartbeat 在行业内得到了广泛应用，也发行了很多的版本。可以从 Linux-HA 的官方网站 www.linux-ha.org 下载到 heartbeat 的最新版本。

11.3.2 安装 heartbeat

这里下载的软件包是 heartbeat-2.1.3.tar.gz，可通过源码进行安装。

同时还需要安装一个 libnet 工具包。libnet 是一个高层次的 API 工具，可以从 http://sourceforge.net/projects/libnet-dev/ 下载到，这里下载的是 libnet-1.1.4.tar.gz。

heartbeat 的安装非常简单，基本操作步骤如下：

（1）安装 libnet。

```
[root@ DR1 ~]#tar -zxvf libnet-1.1.4.tar.gz
[root@ DR1 ~]#cd libnet-1.1.4
[root@ DR1 ~/libnet]#./configure
[root@ DR1 ~/libnet]#make
[root@ DR1 ~/libnet]#make install
```

（2）安装 heartbeat。

```
[root@ DR1 ~]#tar zxvf heartbeat-2.1.3.tar.gz
[root@ DR1 ~]#cd heartbeat-2.1.3
[root@ DR1 ~/ heartbeat-2.1.3]#./ConfigureMe configure \
>--disable-swig --disable-snmp-subagent
[root@DR1 ~/ heartbeat-2.1.3]#make
[root@ DR1 ~/ heartbeat-2.1.3]#make install
[root@ DR1 ~/ heartbeat-2.1.3]#cp doc/ha.cf doc/haresources doc/authkeys /etc/ha.d/
[root@ DR1 ~/ heartbeat-2.1.3]#cp ldirectord/ldirectord.cf /etc/ha.d/
[root@ DR1 ~/ heartbeat-2.1.3]#groupadd -g 694 haclient
[root@ DR1 ~/ heartbeat-2.1.3]#useradd -u 694 -g haclient hacluster
```

heartbeat 的安装包中包含了一个 ldirectord 插件，这个插件以后会用到。在 heartbeat 安装完毕后，此插件默认已经安装。但是为了保证 ldirectord 可用，还需要一个 perl-MailTools 的 rpm 包，这个 rpm 从系统盘中找到后安装即可。

11.3.3 开源 HA 软件 Keepalived 的介绍

Keepalived 起初是为 LVS 设计的，专门用来监控集群系统中各个服务节点的状态。它根据 layer3, 4 & 5 交换机制检测每个服务节点的状态，如果某个服务节点出现异常，或工作出现故障，Keepalived 将检测到，并将出现故障的服务节点从集群系统中剔除，而当故障节点恢复正常后，Keepalived 又可以自动将此服务节点重新加入到服务器集群中。这些工作全部自动完成，不需要人工干涉，需要人工完成的只是修复出现故障的服务节点。

Keepalived 后来又加入了 VRRP 的功能。VRRP 是 Virtual Router Redundancy Protocol(虚

拟路由器冗余协议）的缩写，它的作用是解决静态路由出现的单点故障问题，它能够保证网络不间断地、稳定地运行。综上所述，Keepalived 一方面具有服务器运行检测功能，另一方面也具有 HA cluster 功能。因此通过 Keepalived 可以搭建一个高可用的 LVS 负载均衡集群系统。

11.3.4　安装 Keepalived

Keepalived 的官方网址是 http://www.keepalived.org，可以在这里下载到各种版本的 Keepalived，这里下载的是 keepalived-1.1.19.tar.gz。安装步骤如下：

```
[root@DR1 ~]#tar zxvf keepalived-1.1.19.tar.gz
[root@DR1 ~]#cd keepalived-1.1.19
[root@DR1 keepalived-1.1.19]#./configure    --sysconf=/etc \
>--with-kernel-dir=/usr/src/kernels/2.6.18-8.el5-i686
[root@DR1 keepalived-1.1.19]#make
[root@DR1 keepalived-1.1.19]#make install
[root@DR1 keepalived-1.1.19]#ln -s /usr/local/sbin/keepalived  /sbin/
```

在编译选项中，“--sysconf”指定了 Keepalived 配置文件的安装路径，即路径为 /etc/Keepalived/Keepalived.conf。“--with-kernel-dir”是个很重要的参数，但这个参数并不是要把 Keepalived 编译进内核，而是指定使用内核源码中的头文件，即 include 目录。只有使用 LVS 时，才需要用到此参数，其他时候是不需要的。

安装完成，执行如下操作：

```
[root@dr1 ~]# keepalived --help
Keepalived v1.1.19 (05/05,2010)
Usage:
  keepalived
  keepalived -n
  keepalived -f keepalived.conf
  keepalived -d
  keepalived -h
  keepalived -v

Commands:
Either long or short options are allowed.
  keepalived --vrrp               -P    Only run with VRRP subsystem.
  keepalived --check              -C    Only run with Health-checker subsystem.
  keepalived --dont-release-vrrp  -V    Dont remove VRRP VIPs & VROUTEs on daemon stop.
  keepalived --dont-release-ipvs  -I    Dont remove IPVS topology on daemon stop.
  keepalived --dont-fork          -n    Dont fork the daemon process.
  keepalived --use-file           -f    Use the specified configuration file.
                                 Default is /etc/keepalived/keepalived.conf.
  keepalived --dump-conf          -d    Dump the configuration data.
  keepalived --log-console        -l    Log message to local console.
  keepalived --log-detail         -D    Detailed log messages.
```

```
keepalived --log-facility       -S    0-7 Set syslog facility to LOG_LOCAL[0-7].
    (default=LOG_DAEMON)
keepalived --help               -h    Display this short inlined help screen.
keepalived --version            -v    Display the version number
keepalived --pid                -p    pidfile
keepalived --checkers_pid       -c    checkers pidfile
keepalived --vrrp_pid           -r    vrrp pidfile
```

这里列出了 Keepalived 的各种用法，同时也表明 Keepalived 已安装成功。

11.4 安装 LVS 软件

LVS 是通过 IPVS 模块来实现的。IPVS 是 LVS 集群系统的核心软件，主要用于完成用户的请求到达负载调度器后，如何将请求发送到每个 Real Server 节点、Real Server 节点如何返回数据给用户等。

11.4.1 配置与检查安装环境

这里设定，操作系统采用 CentOS 5.4，该版本内核默认支持 LVS 功能。为了方便编译安装 IPVS 管理软件，在安装操作系统时，建议选择如下这些安装包：

- 桌面环境：xwindows system、GNOME desktop environment。
- 开发工具：development tools、x software development、gnome software、development、kde software development。

系统安装完毕后，可以通过如下命令检查 kernel 是否已经支持 LVS 的 IPVS 模块。

```
[root@DR1 ~]#modprobe -l |grep ipvs
/lib/modules/2.6.18-164.11.1.el5PAE/kernel/net/ipv4/ipvs/ip_vs.ko
/lib/modules/2.6.18-164.11.1.el5PAE/kernel/net/ipv4/ipvs/ip_vs_dh.ko
/lib/modules/2.6.18-164.11.1.el5PAE/kernel/net/ipv4/ipvs/ip_vs_ftp.ko
/lib/modules/2.6.18-164.11.1.el5PAE/kernel/net/ipv4/ipvs/ip_vs_lblc.ko
/lib/modules/2.6.18-164.11.1.el5PAE/kernel/net/ipv4/ipvs/ip_vs_lblcr.ko
/lib/modules/2.6.18-164.11.1.el5PAE/kernel/net/ipv4/ipvs/ip_vs_lc.ko
/lib/modules/2.6.18-164.11.1.el5PAE/kernel/net/ipv4/ipvs/ip_vs_nq.ko
/lib/modules/2.6.18-164.11.1.el5PAE/kernel/net/ipv4/ipvs/ip_vs_rr.ko
```

如果有类似上面的输出，表明系统内核默认支持 IPVS 模块。接下来就可以安装 IPVS 管理软件了。

11.4.2 在 Director Server 上安装 IPVS 管理软件

IPVS 提供的软件包有源码方式的也有 rpm 方式的，这里介绍如何通过源码方式安装 IPVS。首先从 http://www.linuxvirtualserver.org/software/ipvs.html 下载对应版本的 ipvs 源码，由于这里采用的操作系统为 CentOS 5.4 版本，因此，下载对应的 ipvsadm-1.24 版本，接着进

行安装。

```
[root@DR1 ~]#tar zxvf ipvsadm-1.24.tar.gz
[root@ DR1 ~]#cd ipvsadm-1.24
[root@ DR1 ~]#make
[root@ DR1 ~]#make install
```

注意，在执行 make 时可能会出现类似如下的错误编译信息：

```
libipvs.h:14:23: error: net/ip_vs.h: No such file or directory
```

这是由于编译程序找不到对应内核造成的。按照如下操作就可以正常编译：

```
[root@ DR1 ~]# ln -s /usr/src/kernels/2.6.18-164.el5-i686 /usr/src/linux
```

也可以通过 yum 方式在线安装：

```
[root@ DR1 ~]# yum install ipvsadm
```

然后执行：

```
[root@ DR1 ~]# ipvsadm --help
```

如果看到帮助提示，表明 IPVS 已经成功安装。

11.5　搭建高可用 LVS 集群

LVS 集群有 DR、TUN、NAT 三种配置模式，可以对 WWW 服务、FTP 服务、MAIL 服务等进行负载均衡。下面通过 3 个实例详细讲述如何搭建 WWW 服务的高可用 LVS 集群系统，以及基于 DR 模式的 LVS 集群配置。在进行实例介绍之前进行约定：操作系统采用 CentOS 5.4，地址规划如表 11-1 所示。

表 11-1　地址规划情况

节点类型	IP 地址规划	主机名	类型
主 Director Server	eth0：192.168.12.130	DR1	Public IP
	eth1：10.10.10.1	priv1	private IP
	eth0:0：192.168.12.200	无	Virtual IP
备用 Director Server	eth0：192.168.12.131	DR2	Public IP
	eth1：10.10.10.2	priv1	private IP
Real server 1	eth0：192.168.12.132	rs1	Public IP
	lo:0：192.168.12.200	无	Virtual IP
Real server 2	eth0：192.168.12.133	rs2	Public IP
	lo:0：192.168.12.200	无	Virtual IP

整个高可用 LVS 集群系统的拓扑结构如图 11-3 所示。

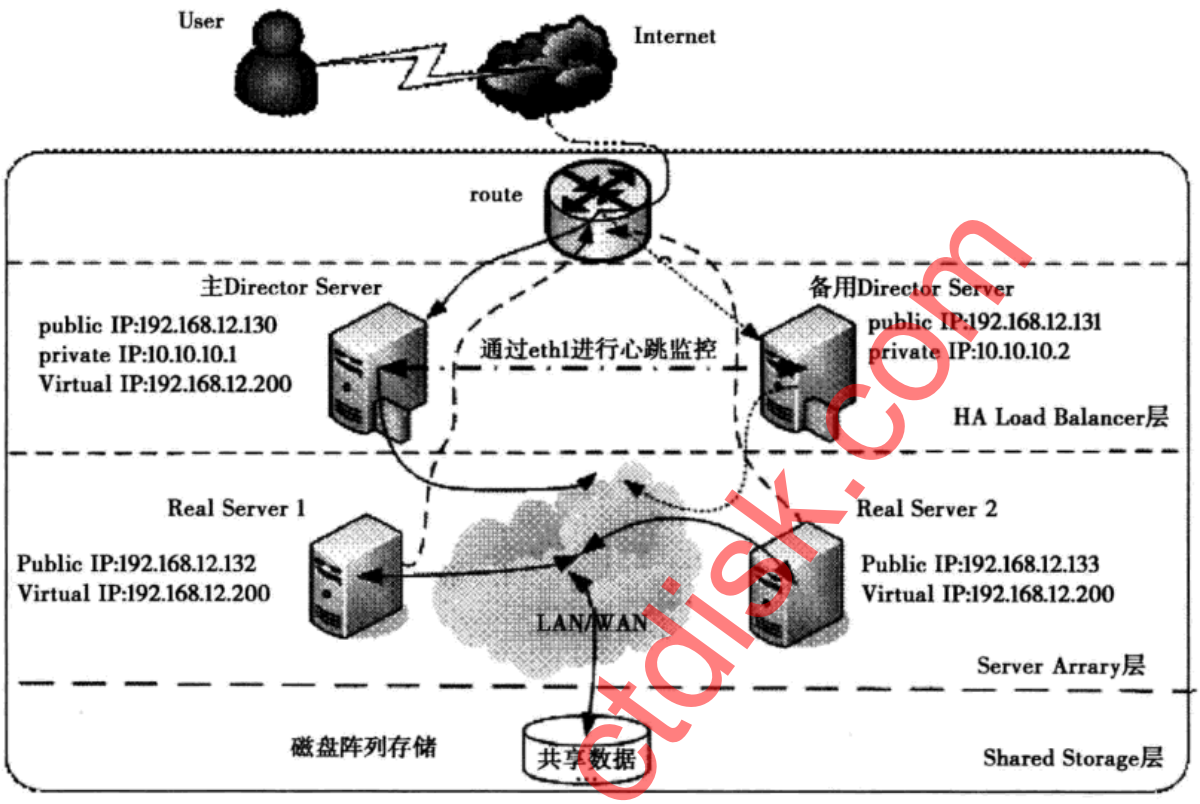

图 11-3 高可用的 LVS 集群拓扑结构

11.5.1 通过 heartbeat 搭建 LVS 高可用性集群

1. 配置 LVS 集群

配置 LVS 的方法有很多，可以通过 LVS 提供的 ipvsadm 命令进行配置，也可以通过第三方插件或工具来进行配置，例如通过 Ldirectord 来配置 LVS，或者通过 Redhat 提供的界面工具 piranha 来配置等。这里选择通过 Ldirectord 来配置 LVS。

（1）通过 Ldirectord 在主、备 Director Server 上配置 LVS

Ldirectord 是 heartbeat 的一个插件，在安装 heartbeat 时，默认已经安装了此插件。Ldirectord 主要用于监控集群系统中每个服务节点的运行状态，当某个节点的服务出现异常或主机出现故障时，将此节点从集群系统中剔除，并且在节点恢复正常后，重新将此节点加入集群系统。

除了监控服务节点外，Ldirectord 的另一个功能是配置 LVS，只需设置好 Ldirectord 的配置文件，启动服务即可，Ldirectord 会自动调用 ipvsadm 命令创建 LVS 路由表信息。Ldirectord 配置文件的默认路径为 /etc/ha.d/ldirectord.cf。这里详细介绍一下这个文件中每个选项的含义。

下面是需要配置的选项。

```
checktimeout=20                 # 判定 Real Server 出错的时间间隔
checkinterval=10                # 指定 Ldirectord 在两次检查之间的间隔时间
fallback=127.0.0.1:80           # 当所有的 Real Server 节点不能工作时，Web 服务重定向的地址
autoreload=yes                  # 是否自动重载配置文件，选 yes 时，配置文件发生变化，自动载入配置信息
logfile="/var/log/ldirectord.log"
                                # 设定 Ldirectord 日志输出文件路径
quiescent=no                    # 当选择 no 时，如果一个节点在 checktimeout 设置的时间周期内没有响应，
                                #ldirectord 将会从 LVS 的路由表中直接移除 Real Server，此时，将中断
                                # 现有的客户端连接，并使 LVS 丢掉所有的连接跟踪记录和持续连接模板；
                                # 如果选择 yes，当某个 Real Server 失效时，Ldirectord 将失效节点的
                                # 权值设置为 0，新的连接将不能到达，但是并不会从 LVS 路由表中清除此
                                # 节点，同时，连接跟踪记录和程序连接模板仍然保留在 Director 上
```

注意，以上几行为 ldirectord.cf 文件的全局设置，它们可以应用到多个虚拟主机。下面是每个虚拟主机的配置。

```
virtual=192.168.12.200:80       # 指定虚拟的 IP 地址和端口号，注意，在 virtual 这行后面的行
                                # 必须缩进 4 个空格或以一个 tab 字符进行标记
real=192.168.12.132:80 gate     # 指定 Real Server 服务器地址和端口，同时设定 LVS 工作模式，
                                # 用 gate 表示 DR 模式，ipip 表示 TUNL 模式，masq 表示 NAT 模式
real=192.168.60.133:80 gate
fallback=127.0.0.1:80 gate
service=http                    # 指定服务的类型，这里是对 HTTP 服务进行负载均衡
request="index.html"            # Ldirectord 将根据指定的 Real Server 地址，结合该选项给
                                # 出的请求页面，发送访问请求，检查 Real Server 上的服务是
                                # 否正常运行，必须确保这里给出的页面地址是可访问的，不然
                                # Ldirectord 会误认为此节点已经失效，发生错误监控现象
receive="Test Page"             # 指定请求和应答字串，也就是 index.html 的内容
scheduler=rr                    # 指定调度算法，这里是 rr（轮询）算法
protocol=tcp                    # 指定协议的类型，LVS 支持 TCP 和 UDP 协议
checktype=negotiate             # 指定 Ldirectord 的检测类型，checktype 可以是 connect、
                                # external、negotiate、off、on、ping 和 checktimeout
                                # 这几个，默认为 negotiate，通过页面交互来判断服务器节点
                                # 是否正常
checkport=80                    # 指定监控的端口号
virtualhost=www.ixdba.net       # 虚拟服务器的名称，可任意指定
```

配置完毕后就可以执行如下命令启动或关闭 Ldirectord 服务：

```
/etc/init.d/ldirectord {start|stop}
```

（2）Real server 的配置

在 LVS 的 DR 和 TUN 模式下，用户的访问请求到达 Real Server 后，是直接返回给用户的，不再经过前端的 Director Server，因此，需要在每个 Real server 节点上增加虚拟的 VIP 地址，这样数据才能直接返回给用户。增加 VIP 地址的操作可以通过创建脚本的方式来实现。创建文件 /etc/init.d/lvsrs，脚本内容如下：

```
[root@rs1 ~]#more /etc/init.d/lvsrs
#!/bin/bash
```

```
#description : Start Real Server
VIP=192.168.12.200
./etc/rc.d/init.d/functions
case "$1" in
    start)
   echo " Start LVS  of  Real Server "
 /sbin/ifconfig lo:0 $VIP broadcast $VIP netmask 255.255.255.255 up
   echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
   echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
   echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
   echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
   ;;
    stop)
   /sbin/ifconfig lo:0 down
   echo "close LVS Director server"
   echo "0" >/proc/sys/net/ipv4/conf/lo/arp_ignore
   echo "0" >/proc/sys/net/ipv4/conf/lo/arp_announce
   echo "0" >/proc/sys/net/ipv4/conf/all/arp_ignore
   echo "0" >/proc/sys/net/ipv4/conf/all/arp_announce
   ;;
    *)
   echo "Usage: $0 {start|stop}"
   exit 1
esac
```

然后修改 lvsrs 使其具有可执行权限。

```
[root@rs1 ~]#chomd 755 /etc/init.d/lvsrs
```

最后，可以通过下面的命令启动或关闭 lvsrs：

```
service lvsrs {start|stop}
```

2. 在主、备 Director Server 上配置 heartbeat

在搭建 Director Server 的双机热备系统之前，首先需要在两台主机上安装 heartbeat 软件。heartbeat 的安装已经在前面介绍过，这里不再讲述，直接进入 heartbeat 的配置。

（1）配置 heartbeat 的主配置文件 (/etc/ha.d/ha.cf)

下面对 ha.cf 文件的每个选项进行详细介绍。

```
#debugfile /var/log/ha-debug
logfile /var/log/ha-log      # 指定 heartbeat 的日志存放位置
#crm yes                     # 是否开启 ClusterResourceManager（集群资源管理）功能
bcast eth1                   # 指定心跳使用以太网广播方式，并且在 eth1 接口上进行广播
logfacility local0
keepalive 2                  # 指定心跳间隔时间为 2 秒（即每 2 秒在 eth1 上发送一次广播）
deadtime 30                  # 如果指定备用节点在 30 秒内没有收到主节点的心跳信号，则
                             # 立即接管主节点的服务资源
warntime 10                  # 指定心跳延迟的时间为 10 秒。当 10 秒内备用机不能接收到主节点的
                             # 心跳信号时，就会在日志中写入一个警告信息，但此时不会切换服务
initdead 120                 # 在某些系统上，系统启动或重启之后需要经过一段时间网络才能
```

```
                                # 正常工作，该选项用于设置这种情况产生的时间间隔，取值至少为
                                #deadtime 的两倍
udpport 694                     # 设置广播通信使用的端口，694 为默认使用的端口号
baud 19200                      # 设置串行通信的比特率
serial /dev/ttyS0               # 选择串行通信设备，用于双机使用串口线连接的情况。如果双机
                                # 通过以太网连接，则应该关闭该选项
#ucast eth0 192.168.1.2         # 采用网卡 eth0 的 UDP 单播来组织心跳，后面跟的 IP 地址应为
                                # 双机中对方的 IP 地址
#mcast eth0 225.0.0.1 694 1 0   # 采用网卡 eth0 的 UDP 多播来组织心跳，一般在备用机不止
                                # 一台时使用。bcast、ucast 和 mcast 分别代表广播、单播和
                                # 多播，是组织心跳的 3 种方式，任选其一即可
auto_failback on                # 用来定义当主节点恢复后，是否将服务自动切回，heartbeat 的两
                                # 台主机分别为主节点和备用节点。主节点在正常情况下占用资源
                                # 并运行所有的服务，遇到故障时把资源交给备用节点并由备用
                                # 节点运行服务。在将该选项设为 on 的情况下，一旦主节点恢复运行，
                                # 则自动获取资源并取代备用节点；如果该选项设置为 off，那么，
                                # 主节点恢复后，将变为备用节点，而原来的备用节点成为主节点
#stonith baytech /etc/ha.d/conf/stonith.baytech
                                #stonith 的主要作用是使出现问题的节点从集群环境中脱离，进而释放集群
                                # 资源，避免两个节点争用一个资源的情况发生。保证共享数据的安全性和完整性
#watchdog /dev/watchdog         # 该选项是可选配置，通过 heartbeat 来监控系统的运行状态。使用该
                                # 特性，需要在内核中载入“softdog”内核模块，用来生成实际的设备
                                # 文件，如果系统中没有这个内核模块，就需要指定此模块，重新编译内核。
                                # 编译完成后输入“insmod softdog”加载该模块，然后输入“grep
                                # misc /proc/devices”（应为 10），输入“cat /proc/misc |
                                # grep watchdog”（应为 130）。最后，生成设备文件“mknod/dev/
                                #watchdog c 10 130”即可使用此功能
node DR1                        # 主节点主机名，可以通过命令“uanme -n”查看
node DR2                        # 备用节点主机名
ping 192.168.12.1               # 选择 ping 的节点。ping 节点选择得越好，HA 集群就越强壮。可以选择
                                # 固定的路由器作为 ping 节点，但是最好不要选择集群中的成员作为 ping
                                # 节点，ping 节点仅用来测试网络连接
ping node 192.168.12.188 192.168.12.100
                                # 指定 ping node。ping node 并不是双机中的两个节点，仅仅用来测试
                                # 网络的连通性
respawn hacluster /usr/lib/heartbeat/ipfail
                                # 该选项是可选配置，列出与 heartbeat 一起启动和关闭的进程，该进程
                                # 一般是和 heartbeat 集成的插件，这些进程遇到故障可以自动重新启动。
                                # 最常用的进程是 ipfail，此进程用于检测和处理网络故障，需要配合 ping
                                # 语句指定的 ping node 来检测网络的连通性。其中 hacluster 表示启动
                                #ipfail 进程的身份
```

（2）配置 heartbeat 的资源文件 (/etc/ha.d/haresources)

haresources 文件用于指定双机系统的主节点、集群 IP、子网掩码、广播地址以及启动的服务等集群资源。文件每一行可以包含一个或多个资源脚本名，资源脚本名之间用空格隔开，参数之间使用两个冒号隔开。

```
DR1 IPaddr::192.168.12.200/24/eth0  ldirectord  # 设置 DR1 为主节点，集群服务器
                                                # 的 IP 地址为 192.168.12.200，
```

```
#netmask 为255.255.255.0,
# 同时指定此IP使用的网络接口为
#eth0, heartbeat 托管的服务
# 为 ldirectord
```

注意，这里的 Ldirectord 对应的文件为 /etc/init.d/ldirectord，即 Ldirectord 服务的启动文件，也就是将 Ldirectord 的启动与关闭交给 heartbeat 来管理。另外，LVS 主节点和备份节点中的资源文件 haresources 要完全一致，当指定 DR1 是主节点后，另一个节点 DR2 就是备份节点。

（3）配置 heartbeat 的认证文件 (/etc/ha.d/authkeys)

authkeys 文件用于设定 heartbeat 的认证方式，该文件中有 3 种可用的认证方式：crc、sha1 和 md5，这里使用 crc 认证方式。设置如下：

```
auth 1
1 crc
#2 sha1 sha1_any_password
#3 md5 md5_any_password
```

需要说明的一点是，无论“auth”后面指定的是什么数字，在下一行必须作为关键字再次出现，例如指定了“auth 6”，下面一定要有一行“6 认证类型”。

最后确保这个文件的权限是 600（即 -rw-------）。

3. 启动 heartbeat+LVS 集群系统

（1）启动 heartbeat 服务

所有配置完成后，就可以在主、备 Director Server 上启动 heartbeat 服务了。可以通过如下方式管理 heartbeat 服务：

```
[root@DR1~]#/etc/init.d/heartbeat \
>{start|stop|status|restart|reload|force-reload}
```

由于 heartbeat 托管了主、备 Director Server 上的 Ldirectord 服务，因此只需在主、备两台机器上启动 heartbeat 服务即可，这样 Ldirectord 服务就在主机上启动起来了。

（2）启动 Real Server 节点服务

分别在两个 Real Server 节点上执行如下脚本：

```
[root@rs1~]#/etc/init.d/lvsrs start
```

至此，通过 heartbeat 构建的高可用 LVS 集群系统已经配置完成并运行起来了。

11.5.2 通过 Keepalived 搭建 LVS 高可用性集群系统

1. 配置 Keepalived

Keepalived 的配置非常简单，仅需要一个配置文件即可完成对 HA cluster 和 LVS 服务节

点监控。Keepalived 的安装已经在前面介绍过，在通过 Keepalived 搭建高可用的 LVS 集群实例中，主、备 Director Server 都需要安装 Keepalived 软件，安装成功后，默认的配置文件路径为 /etc/Keepalived/Keepalived.conf。一个完整的 keepalived 配置文件由 3 个部分组成，分别是全局定义部分、vrrp 实例定义部分以及虚拟服务器定义部分。下面详细介绍这个配置文件中每个选项的详细含义和用法。

```
! Configuration File for keepalived
# 全局定义部分
global_defs {
   notification_email {
     dba.gao@gmail.com      # 设置报警邮件地址，可以设置多个，
                            # 每行一个。注意，如果要开启邮件报警，需要开启本机的 sendmail 服务
     ixdba@163.com
   }
   notification_email_from Keepalived@localhost   # 设置邮件的发送地址
   smtp_server 192.168.200.1 # 设置 smtp server 地址
   smtp_connect_timeout 30   # 设置连接 smtp server 的超时时间
   router_id LVS_DEVEL       # 表示运行 Keepalived 服务器的一个标识。发邮件时显示在邮件主题中的信息
}
                            #vrrp 实例定义部分

vrrp_instance VI_1 {
state MASTER                # 指定 Keepalived 的角色，MASTER 表示此主机是主服务器，
                            #BACKUP 表示此主机是备用服务器
    interface eth0          # 指定 HA 监测网络的接口
    virtual_router_id 51    # 虚拟路由标识，这个标识是一个数字，同一个 vrrp 实例使用唯一的标识，
                            # 即同一个 vrrp_instance 下，MASTER 和 BACKUP 必须是一致的
priority 100                # 定义优先级，数字越大，优先级越高。在一个 vrrp_instance 下，
                            #MASTER 的优先级必须大于 BACKUP 的优先级
    advert_int 1            # 设定 MASTER 与 BACKUP 负载均衡器之间同步检查的时间间隔，单位是秒
    authentication {        # 设定验证类型和密码
   auth_type PASS           # 设置验证类型，主要有 PASS 和 AH 两种
   auth_pass 1111           # 设置验证密码，在一个 vrrp_instance 下，
                            #MASTER 与 BACKUP 必须使用相同的密码才能正常通信
    }
    virtual_ipaddress {     # 设置虚拟 IP 地址，可以设置多个虚拟 IP 地址，每行一个
   192.168.12.200
    }
}
                            # 虚拟服务器定义部分
virtual_server 192.168.12.200 80 {
                            # 设置虚拟服务器，需要指定虚拟 IP 地址和服务端口，IP 与端口之间用空格隔开
    delay_loop 6            # 设置运行情况检查时间，单位是秒
    lb_algo rr              # 设置负载调度算法，这里设置为 rr，即轮询算法
    lb_kind DR              # 设置 LVS 实现负载均衡的机制，有 NAT、TUN 和 DR 三个模式可选
persistence_timeout 50      # 会话保持时间，单位是秒。这个选项对动态
                            # 网页是非常有用的，为集群系统中的 session 共享
                            # 提供了一个很好的解决方案。有了这个会话保持功能，用户的请求会被
```

```
                                # 一直分发到某个服务节点，直到超过这个会话的保持时间。需要注意的是，
                                # 这个会话保持时间是最大无响应超时时间，也就是说，用户在操作动态页
                                # 面时，如果在 50 秒内没有执行任何操作，那么接下来的操作会被分发到
                                # 另外的节点，但是如果用户一直在操作动态页面，则不受 50 秒的时间限制
    protocol TCP                # 指定转发协议类型，有 TCP 和 UDP 两种
    real_server 192.168.12.132 80 {
                                # 配置服务节点 1，需要指定 real server 的真实 IP 地址和端口，
                                #IP 与端口之间用空格隔开
weight 3                        # 配置服务节点的权值，权值大小用数字表示，数字越大，权值越高，设置
                                # 权值的大小可以为不同性能的服务器分配不同的负载，可以为性能高的
                                # 服务器设置较高的权值，而为性能较低的服务器设置相对较低的权值，
                                # 这样才能合理地利用和分配系统资源
   TCP_CHECK {                  #realserve 的状态检测设置部分，单位是秒
       connect_timeout 3        # 表示 3 秒无响应超时
       nb_get_retry 3           # 表示重试次数
      delay_before_retry 3      # 表示重试间隔
   }
    }

    real_server 192.168.12.133 80 {
                                # 配置服务节点 2
   weight 1
   TCP_CHECK {
       connect_timeout 3
       nb_get_retry 3
       delay_before_retry 3
   }
    }
}
```

在配置 Keepalived.conf 时，需要特别注意配置文件的语法格式，因为 Keepalived 在启动时并不检测配置文件的正确性，即使没有配置文件，Keepalived 也照样能够启动，所以一定要保证配置文件正确。

在默认情况下，Keepalived 在启动时会查找 /etc/Keepalived/Keepalived.conf 配置文件，如果配置文件放在了其他路径下，可以通过“Keepalived -f”参数指定配置文件的路径即可。

Keepalived.conf 配置完毕后，将此文件复制到备用 Director Server 对应的路径下，然后进行以下两个简单的修改即可：

- 将“state MASTER”更改为“state BACKUP”。
- 将“priority 100”更改为一个较小的值，这里改为“priority 80”。

2. 配置 Real server 节点

与 heartbeat+LVS 类似，Keepalived+LVS 也需要为 Real server 节点配置相关的脚本，以达到与 Director Server 相互通信的目的。脚本的内容已经在前面介绍过，这里不再讲述。

3. 启动 Keepalived+LVS 集群系统

在主、备 Director Server 上分别启动 Keepalived 服务，可以执行如下操作：

```
[root@DR1 ~]#/etc/init.d/Keepalived  start
```

接着在两个 Real server 上执行如下脚本：

```
[root@rs1~]#/etc/init.d/lvsrs start
```

至此，Keepalived+LVS 高可用的 LVS 集群系统已经运行起来了。

11.5.3　通过 piranha 搭建 LVS 高可用性集群

piranha 是 REDHAT 提供的一个基于 Web 的 LVS 配置软件，通过 piranha 可以省去手工配置 LVS 的繁琐工作。同时，piranha 也可以单独提供集群功能，例如，可以通过 piranha 激活 Director Server 的备用主机。这里利用 piranha 来配置 Director Server 的双机热备功能。

1. 安装与配置 piranha

piranha 工具的安装非常简单，下载 piranha 的 rpm 包，在主、备 Director Server 上进行安装即可。过程如下：

```
[root@DR1 ~]#rpm -ivh piranha-0.8.2-1.i386.rpm
```

也可以通过 yum 命令直接在线安装。过程如下：

```
[root@DR2 ~]# yum install piranha
```

piranha 安装完毕后，会产生 /etc/sysconfig/ha/lvs.cf 配置文件。默认此文件是空的，可以通过 piranha 提供的 Web 界面配置此文件，也可以直接手动编辑此文件。编辑好的 lvs.cf 文件内容大致如下。

```
[root@DR1 ~]# more /etc/sysconfig/ha/lvs.cf
serial_no = 18                  #序号
primary = 192.168.60.130  #指定主 Director Server 的真实 IP 地址
service = lvs                   #指定双机的服务名
backup_active = 1               #是否激活备用 Director Server。“0”表示不激活，“1”表示激活。
                                #这里选择激活
backup = 192.168.12.131   #这里指定备用 Director Server 的真实 IP 地址，如果没有备用
                                #Director Server，可以用“0.0.0.0”代替
heartbeat = 1                   #是否开启心跳，1 表示开启，0 表示不开启
heartbeat_port = 539            #指定心跳的 UDP 通信端口。
keepalive = 5                   #心跳间隔时间，单位是秒
deadtime = 10                   #如果主 Director Server 在 deadtime（秒）后没有响应，那么备用
                                #Director Server 就会接管主 Director Server 的服务
network = direct                #指定 LVS 的工作模式，direct 表示 DR 模式，nat 表示 NAT 模式，tunnel
                                #表示 TUN 模式
debug_level = NONE              #定义 debug 调试信息级别
virtual www.ixdba.net{    #指定虚拟服务的名称
active = 1                      #是否激活此服务
address = 192.168.12.200 eth0:0   #虚拟服务绑定的虚拟 IP 及网络设备名
port = 80                       #虚拟服务的端口
send = "GET / HTTP/1.0\r\n\r\n"   #向 real server 发送的验证字符串
```

```
expect = "HTTP"              # 服务器正常运行时应该返回的文本应答信息，用来判断 Real Server
                             # 是否工作正常
use_regex = 0                # expect 选项中是否使用正则表达式，0 表示不使用，1 表示使用
load_monitor = none          #LVS 中的 Director Server 能够使用 rup 或 ruptime 来监视各个
                             #Real Server 的负载状态。该选项有 3 个可选值，rup、ruptime 和
                             #none，如果选择 rup，每个 Real Server 就必须运行 rstatd 服务。
                             # 如果选择了 ruptime，每个 Real Server 就必须运行 rwhod 服务
scheduler = rr               # 指定 LVS 的调度算法
protocol = tcp               # 虚拟服务使用的协议类型
timeout = 6                  #Real Server 失效后从 LVS 路由列表中移除失效 Real Server 所必须持续
                             # 的时间，以秒为单位
reentry = 15                 # 某个 Real Server 被移除后，重新加入 LVS 路由列表中必须持续的时间，
                             # 以秒为单位
quiesce_server = 0           # 如果此选项为 1，那么当某个新的节点加入集群时，最少连接数会被重设
                             # 为零，因此 LVS 会发送大量请求到此服务节点，造成新的节点服务阻塞，
                             # 建议设置为 0
server RS1 {                 # 指定 Real Server 服务名
address = 192.168.12.132     # 指定 Real Server 的 IP 地址
active = 1                   # 是否激活此 Real Server 服务
weight = 1                   # 指定此 Real Server 的权值，是整数值。权值是相对于所有 Real Server
                             # 节点而言的，权值高的 Real Server 处理负载的性能相对较强
}
server RS2 {
address = 192.168.12.133
active = 1
weight = 1
}
}
```

接着，还需要对两个 Real Server 节点进行配置，也就是创建 /etc/init.d/lvsrs 脚本，这个脚本在前面已经讲述，这里不再介绍。

2. 启动通过 piranha 配置的 LVS 集群系统

将编辑好的 lvs.cf 从 Director Server 的主节点复制到备用节点，然后在主、备节点上分别启动 pulse 服务，即启动 LVS 服务。过程如下：

```
[root@DR1 ~]#service pulse start
```

接下来，还要在主、备节点上启用系统的包转发功能（其实只有在 NAT 模式下需要）。命令如下：

```
[root@DR1 ~]#echo "1" >/proc/sys/net/ipv4/ip_forward
```

最后，在两个 Real server 节点上执行 lvsrs 脚本。命令如下：

```
[root@rs1 ~]#/etc/init.d/lvsrs start
```

到此为止，利用 piranha 工具搭建的高可用 LVS 集群系统已经运行起来了。

11.6　测试高可用 LVS 负载均衡集群系统

高可用的 LVS 负载均衡系统能够实现 LVS 的高可用性、负载均衡特性和故障自动切换特性，因此，对其进行的测试也针对这 3 个方面进行。这里只对 Keepalived+LVS 实例进行测试，其他实例也有类似的效果，不一一讲述。

11.6.1　高可用性功能测试

高可用性是通过 LVS 的两个 Director Server 完成的。为了模拟故障，先将主 Director Server 上面的 Keepalived 服务停止，然后观察备用 Director Server 上 Keepalived 的运行日志。信息如下：

```
May  4 16:50:04 DR2 Keepalived_vrrp: VRRP_Instance(VI_1) Transition to MASTER STATE
May  4 16:50:05 DR2 Keepalived_vrrp: VRRP_Instance(VI_1) Entering MASTER STATE
May  4 16:50:05 DR2 Keepalived_vrrp: VRRP_Instance(VI_1) setting protocol VIPs.
May  4 16:50:05 DR2 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on
   eth0 for 192.168.12.200
May  4 16:50:05 DR2 Keepalived_vrrp: Netlink reflector reports IP 192.168.12.200
   added
May  4 16:50:05 DR2 Keepalived_healthcheckers: Netlink reflector reports IP
   192.168.12.200 added
May  4 16:50:05 DR2 avahi-daemon[2551]: Registering new address record for
   192.168.12.200 on eth0.
May  4 16:50:10 DR2 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on
   eth0 for 192.168.12.200
```

从日志中可以看出，主机出现故障后，备用机立刻检测到，此时备用机变为 MASTER 角色，并且接管了主机的虚拟 IP 资源，最后将虚拟 IP 绑定在 eth0 设备上。

接着，重新启动主 Director Server 上的 Keepalived 服务，继续观察备用 Director Server 的日志状态。

```
May  4 16:51:30 DR2 Keepalived_vrrp: VRRP_Instance(VI_1) Received higher prio advert
May  4 16:51:30 DR2 Keepalived_vrrp: VRRP_Instance(VI_1) Entering BACKUP STATE
May  4 16:51:30 DR2 Keepalived_vrrp: VRRP_Instance(VI_1) removing protocol VIPs.
May  4 16:51:30 DR2 Keepalived_vrrp: Netlink reflector reports IP 192.168.12.200
   removed
May  4 16:51:30 DR2 Keepalived_healthcheckers: Netlink reflector reports IP
   192.168.12.200 removed
May  4 16:51:30 DR2 avahi-daemon[2551]: Withdrawing address record for 192.168.12.200
   on eth0.
```

从日志可知，备用机在检测到主机重新恢复正常后，重新返回 BACKUP 角色，并且释放了虚拟 IP 资源。

11.6.2 负载均衡测试

这里假定两个 Real Server 节点上配置 WWW 服务的网页文件的根目录均为 /webdata/www 目录，然后分别执行如下操作：

在 real server1 执行：

```
echo "This is real server1"  /webdata/www/index.html
```

在 real server2 执行：

```
echo "This is real server2" /webdata/www/index.html
```

接着打开浏览器，访问 http://192.168.12.200 这个地址，然后不断刷新此页面。如果能分别看到“This is real server1”和“This is real server2”就表明 LVS 已经在进行负载均衡了。

11.6.3 故障切换测试

故障切换是测试在某个节点出现故障后，Keepalived 监控模块是否能及时发现，然后屏蔽故障节点，同时将服务转移到正常节点上执行。

这里将 real server 1 节点服务停掉，假定这个节点出现故障，然后查看主、备机日志信息。相关日志如下：

```
May   4  17:01:51  DR1  Keepalived_healthcheckers:  TCP  connection  to
   [192.168.12.132:80] failed !!!
May  4 17:01:51 DR1 Keepalived_healthcheckers: Removing service [192.168.12.132:80]
   from VS [192.168.12.200:80]
May   4 17:01:51 DR1 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25]
   connected.
May  4 17:02:02 DR1 Keepalived_healthcheckers: SMTP alert successfully sent.
```

通过日志可以看出，Keepalived 监控模块检测到 192.168.12.132 这台主机出现故障后，将此节点从集群系统中剔除掉了。

此时访问 http://192.168.12.200 这个地址，应该只能看到“This is real server2”了。这是因为节点 1 出现故障，Keepalived 监控模块将节点 1 从集群系统中剔除了。

下面重新启动 real server 1 节点的服务，可以看到 Keepalived 日志信息如下：

```
May   4  17:07:57  DR1  Keepalived_healthcheckers:  TCP  connection  to
   [192.168.12.132:80] success.
May  4 17:07:57 DR1 Keepalived_healthcheckers: Adding service [192.168.12.132:80]
   to VS [192.168.12.200:80]
May   4 17:07:57 DR1 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25]
   connected.
May  4 17:07:58 DR1 Keepalived_healthcheckers: SMTP alert successfully sent.
```

从日志可知，Keepalived 监控模块检测到 192.168.12.132 这台主机恢复正常后，又将此

节点加入到集群系统中。

此时再次访问 http://192.168.12.200 这个地址，然后不断刷新此页面，应该又能分别看到“This is real server1”和“This is real server2”页面了，这说明在 real server 1 节点恢复正常后，Keepalived 监控模块将此节点加入到集群系统中。

11.7　本章小结

这一章详细讲述了通过 LVS+heartbeat+Ldirectord、LVS+Keepalived 及 piranha 三种方式来配置高可用 LVS 集群系统的过程。这三种方式各有优缺点，读者可以根据自己的需要选择最适合自己的方式。下面是对这三种方式的总结：

- LVS+heartbeat+Ldirectord 方式：

  优点：安装简单，并且无需单独为 ipvsadm 编写脚本。同时，Ldirectord 支持端口和页面方式进行服务节点监控，配置灵活，可以根据需要进行选择。

  缺点：配置比较复杂，需要对 heartbeat 和 Ldirectord 分别进行配置。

- LVS+Keepalived 方式

  优点：安装简单、配置简单，仅需要一个配置文件即可完成所有配置，同时无需单独为 ipvsadm 编写脚本。Keepalived 对后端服务节点的检测是基于底层网络通信协议的，因此检测效率很高，故障切换速度最快。

- piranha 方式

  优点：安装简单，配置简单，只需一个 lvs.cf 文件即可完成所有配置，也无需为 ipvsadm 编写脚本。

  缺点：在 HA cluster 双机切换过程中，没有主、备机之分，也就是说先启动的是主机，最后启动的是备用机，并且没有类似 heartbet 中的 auto_failback 功能。

第 12 章　RHCS 集群

本章主要介绍 RHCS 的安装、配置、管理和维护技巧。RHCS 是一套综合的软件应用套件，可以在部署时采用不同的配置，以满足对高可用性、负载均衡、存储集群、可扩展性、文件共享和节约成本的需要。高可用性集群通过消除单点故障点和节点故障转移功能来提供高可用性；而存储集群可以在一个集群中为服务提供一个一致的文件系统映像，并且允许服务同时去读写一个单一的共享文件系统；通过负载均衡集群可以将客户端请求调度到集群中的多个节点上。由此可知，RHCS 提供了一个集群系统从前端负载到后端数据存储的完整解决方案，是企业级应用的首选集群软件。

12.1　RHCS 集群概述

RHCS 是 Red Hat Cluster Suite 的缩写，即红帽子集群套件。RHCS 是一个能够提供高可用性、高可靠性、负载均衡、存储共享且经济实用的集群工具集合，它将集群系统中的三大集群架构融合为一体，可以为 Web 应用、数据库应用等提供安全、稳定的运行环境。更确切地说，RHCS 是一个功能完备的集群应用解决方案，从应用的前端访问到后端的数据存储都提供了一个行之有效的集群架构实现方案。通过 RHCS 提供的这种解决方案，不但能保证前端应用持久、稳定地提供服务，同时也保证了后端数据存储的安全。

高可用集群是 RHCS 的核心功能。当应用程序出现故障，或者系统硬件或网络出现故障时，应用可以通过 RHCS 提供的高可用性服务管理组件自动、快速地从一个节点切换到另一个节点。节点故障转移功能对客户端来说是透明的，从而保证应用持续、不间断地对外提供服务，这就是 RHCS 高可用集群实现的功能。

RHCS 通过 LVS 来提供负载均衡集群，而 LVS 是一个开源的、功能强大的基于 IP 的负载均衡技术。LVS 由负载调度器和服务访问节点组成，通过 LVS 的负载调度功能，可以将客户端请求平均分配到各个服务节点上，同时还可以定义多种负载分配策略。当一个请求进来时，集群系统根据调度算法来判断应该将请求分配到哪个服务节点上，然后，由分配到的节点响应客户端请求。LVS 还提供了服务节点故障转移功能，也就是当某个服务节点不能提供服务时，LVS 会自动屏蔽这个故障节点，接着将失败节点从集群中剔除，同时将新分配到此节点的请求平滑转移到其他正常节点上；而在此故障节点恢复正常后，LVS 又会自动将此节点加入到集群中去。这一系列切换动作，对用户来说都是透明的。通过故障转移功能，保证了服务不间断、稳定地运行。

RHCS 通过 GFS 文件系统来提供存储集群功能。GFS 是 Global File System 的缩写，它允许多个服务同时读写一个单一的共享文件系统，存储集群通过将共享数据放到一个共享文件系统中来消除在应用程序间同步数据的麻烦。GFS 是一个分布式文件系统，它通过锁管理机制来协调和管理多个服务节点对同一个文件系统的读写操作。

12.2 RHCS 集群的组成与结构

12.2.1 RHCS 集群的组成

RHCS 是一个集群工具的集合，主要由下面几大部分组成：

- 集群构架管理器

这是 RHCS 集群的一个基础套件，提供一个集群的基本功能，使各个节点组成集群一起工作，具体包括分布式集群管理器（CMAN）、成员关系管理、锁管理（DLM）、配置文件管理（CCS）和栅设备（FENCE）。

- 高可用服务管理器

提供节点服务监控和服务故障转移功能。当一个节点服务出现故障时，将服务转移到另一个健康节点。

- 集群配置管理工具

RHCS 最新版本通过 LUCI 来配置和管理 RHCS 集群。LUCI 是一个基于 Web 的集群配置方式，通过 LUCI 可以轻松搭建一个功能强大的集群系统。

- LVS

LVS 是一个开源的负载均衡软件，利用 LVS 可以将客户端的请求根据指定的负载策略和算法合理地分配到各个服务节点，实时地、动态地、智能地负载分担。

RHCS 除了上面的几个核心组成部分，还可以通过下面这些组件来补充 RHCS 集群功能。

- GFS（Global File System）

GFS 是 Red Hat 公司开发的一款集群文件系统，目前的最新版本是 GFS2。GFS 文件系统允许多个服务同时读写一个磁盘分区，通过 GFS 可以实现数据的集中管理，免去了数据同步和复制的麻烦，但 GFS 并不能孤立存在，安装 GFS 需要 RHCS 的底层组件支持。

- CLVM（Cluster Logical Volume Manager）

集群逻辑卷管理，即 CLVM，是 LVM 的扩展，这种扩展允许集群中的机器使用 LVM 来管理共享存储。

- iSCSI

即 internet SCSI，是 IETF 制订的一项标准，用于将 SCSI 数据块映射为以太网数据包。从根本上说，它是一种基于 IP Storage 理论的新型存储技术。RHCS 可以通过 iSCSI 技术来

导出和分配共享存储的使用。

❑ GNBD（Global Network Block Device）

全局网络模块，是 GFS 的一个补充组件，用于 RHCS 分配和管理共享存储。GNBD 分为客户端和服务器端，在服务器端，GNBD 允许导出多个块设备或 GNBD 文件，而 GNBD 客户端通过导入这些导出的块设备或文件，就可以把它们当做本地块设备使用。现在 GNBD 已经停止了开发，因此使用 GNBD 得越来越少。

12.2.2 RHCS 集群结构

RHCS 集群从整体上分为 3 大部分：负载均衡、高可用性和存储集群。典型的 RHCS 集群拓扑结构如图 12-1 所示。

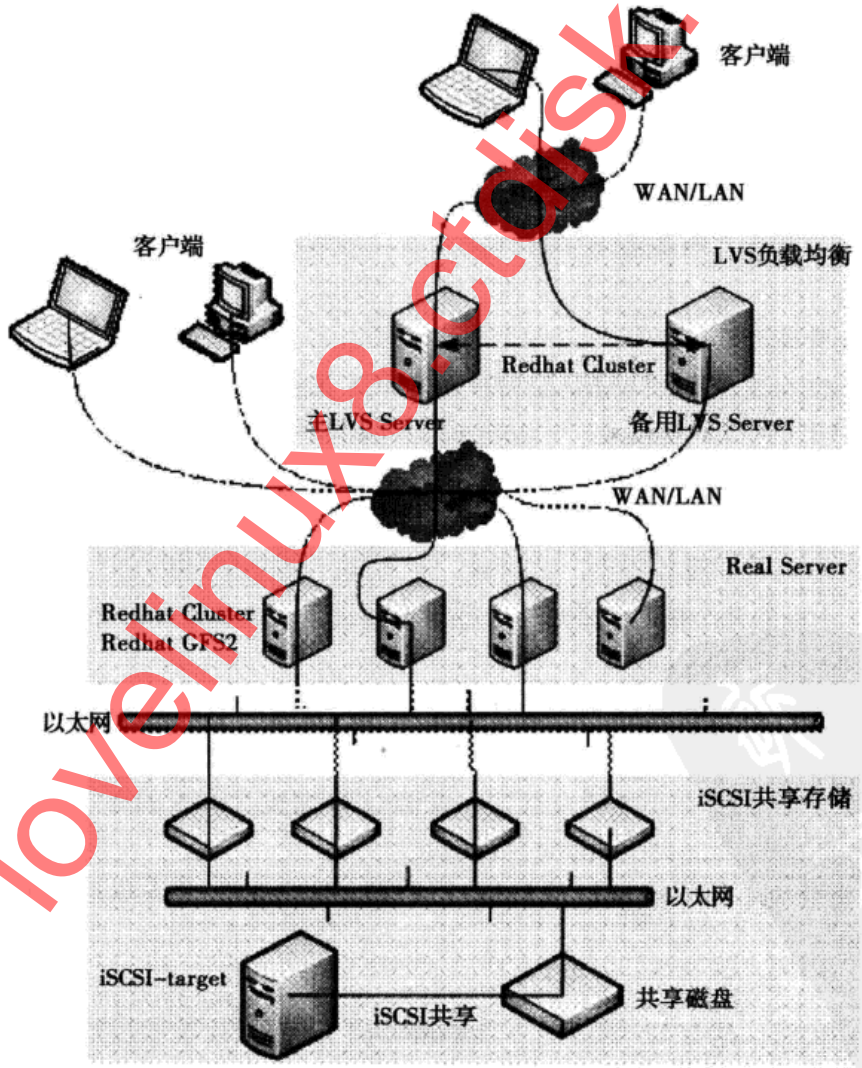

图 12-1 RHCS 集群的拓扑结构

在图 12-1 中，整个拓扑结构分为三层。最上层是 LVS 负载均衡层，通过 LVS 进行客户端请求的负载分配。由于 LVS 服务器是整个集群系统的入口，如果只有一台 LVS 服务器，就会形成整个集群的单点故障，所以用两台 LVS 服务器组成 LVS 高可用集群，当主 LVS 出现故障时，备用 LVS 将自动接管主 LVS 的服务，通过这种机制，保证了 LVS 调度服务持久、稳定地运行。

中间一层是 Real Server 层，也就是服务节点部分，客户端的访问最终都是调度这些节点来响应请求的，服务节点可以有多个，并且可以动态地加入或去除。为了保证客户端访问的一致性，RHCS 集群一般通过 GFS 文件系统为集群提供一个一致的数据镜像。由于 GFS 是一个分布式文件系统，读写操作都由专一的分布式锁进程进行管理，因此，需要在每个服务节点上安装 RHCS 集群基础套件。

最后一层是共享存储层，主要用于为 GFS 文件系统提供共享存储空间。提供共享存储的方式有很多种，可以通过 Red Hat 提供的 GNBD 来实现，也可以通过 iSCSI 技术实现。图 12-1 中的拓扑图就是通过 iSCSI 将共享存储导入到以太网上供每个服务节点使用的。

12.3　RHCS 集群的运行原理及功能

要熟练应用 RHCS 集群，必须了解 RHCS 各个组成部分所实现的详细功能，下面将依次介绍 RHCS 每个集群套件所实现的功能和运行原理。

12.3.1　分布式集群管理器（CMAN）

Cluster Manager，简称 CMAN，是一个分布式集群管理工具，运行在集群的各个节点上，为 RHCS 提供集群管理任务。

CMAN 用于管理集群成员、消息和通知。它通过监控每个节点的运行状态来了解节点成员之间的关系。当集群中某个节点出现故障时，节点成员关系将发生改变，CMAN 及时将这种改变通知底层，进而做出相应的调整。

CMAN 根据每个节点的运行状态，统计出一个法定节点数，作为集群是否存活的依据。当整个集群中有多于一半的节点处于激活状态时，表示达到了法定节点数，此集群可以正常运行；当集群中有一半或者少于一半的节点处于激活状态时，表示没有达到法定的节点数，此时整个集群系统将变得不可用。

图 12-2 显示了集群管理器 CMAN 的运行原理。

12.3.2　锁管理（DLM）

Distributed Lock Manager，简称 DLM，是一个分布式锁管理器，它是 RHCS 的一个底层基础构件，同时也为集群提供了一个公用的锁运行机制。在 RHCS 集群系统中，DLM 运

行在集群的每个节点上，GFS 通过锁管理器的锁机制来同步访问文件系统的元数据。CLVM 通过锁管理器来同步更新数据到 LVM 卷和卷组。

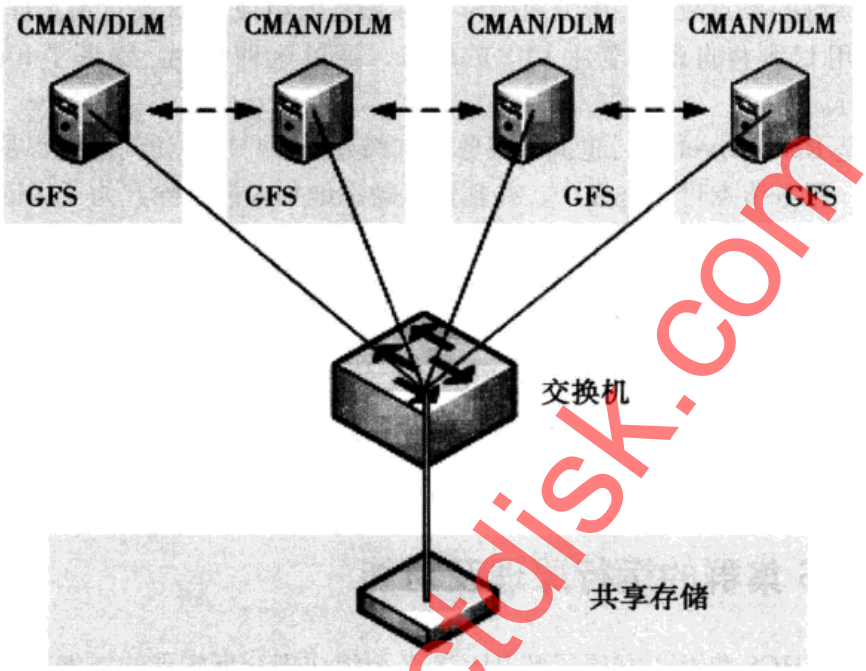

图 12-2 集群管理器 CMAN 的运行原理

DLM 不需要设定锁管理服务器，它采用对等的锁管理方式，大大提高了处理性能。同时，DLM 避免了单个节点失败需要整体恢复的性能瓶颈。另外，DLM 的请求都是本地的，不需要网络请求，因此请求会立即生效。最后，DLM 通过分层机制，可以实现多个锁空间的并行锁模式。

12.3.3 配置文件管理（CCS）

Cluster Configuration System，简称 CCS，主要用于集群配置文件管理和配置文件在节点之间的同步。CCS 运行在集群的每个节点上，监控每个集群节点上的单一配置文件 /etc/cluster/cluster.conf 的状态。当这个文件发生任何变化时，都将此变化更新到集群中的每个节点上，时刻保持每个节点的配置文件同步。例如，管理员在节点 A 上更新了集群配置文件，CCS 发现 A 节点的配置文件发生变化后，马上将此变化传播到其他节点上去。

cluster.conf 是一个 XML 文件，其中包含集群名称、集群节点信息、集群资源和服务信息、fence 设备等，这个会在后面进行详细讲解。

CCS 在 RHCS 集群节点中的内部运行原理如图 12-3 所示。

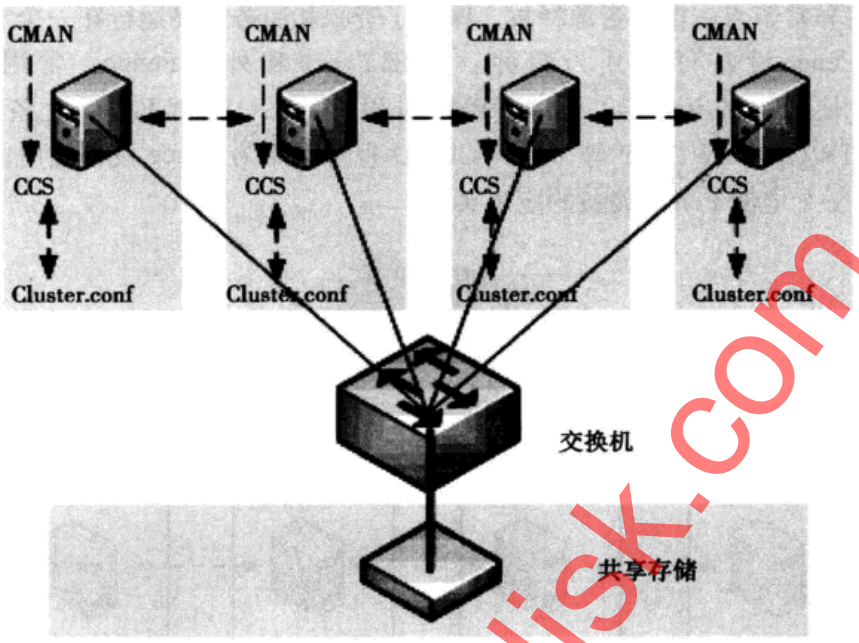

图 12-3　CCS 的内部运行原理

从图 12-3 可知，CMAN 依赖于 CCS，并且 CMAN 通过 CCS 从配置文件访问集群配置信息。

12.3.4　栅设备（Fence）

Fence 设备是 RHCS 集群中必不可少的一个组成部分，通过 Fence 设备可以避免因出现不可预知的情况而造成的“脑裂”现象。所谓“脑裂”是指如下情况：当两个节点之间心跳线中断时，两台主机无法获取对方信息，也无法互相发送监控指令，此时两台主机都会认为自己是主节点，在这种情况下就会发生两个节点对集群资源的争用情况。又如，集群系统中主服务器在某时刻非常繁忙，它可以接收到备用服务器发送的节点监控指令，但是无法向备用服务器发回运行情况的确认信息，这样备用服务器就误认为主服务器已经发生崩溃，接着执行资源切换操作。而此时主服务器并不认为自己出了故障，仍然占据着集群资源，在这个时候，集群中的资源，如虚拟 IP、共享磁盘分区、服务等都运行在两台主机上。这种情况可能导致集群共享数据被破坏，集群服务资源不可用。

Fence 设备的出现，就是为了解决类似以上的问题。Fence 设备主要通过服务器或存储本身的硬件管理接口，或者外部电源管理设备来对服务器或存储直接发出硬件管理指令，将服务器重启或关机，或者与网络断开连接。

Fence 的工作原理是：当意外原因导致主机异常或宕机时，备用机会首先调用 Fence 设备，然后通过 Fence 设备将异常主机重启或从网络上隔离。当隔离操作成功执行后，返回信息给备用机，备用机在接到隔离操作成功的信息后，开始接管主机的服务和资源。这样通过

Fence 设备，将异常节点占据的资源释放，保证了资源和服务始终运行在一个节点上。

RHCS 的 Fence 设备可以分为两种：内部 Fence 和外部 Fence。常用的内部 Fence 有 IBM RSAII 卡、HP 的 iLO 卡，以及 IPMI 的设备等；外部 Fence 设备有 UPS、SAN SWITCH、NETWORK SWITCH 等。另外 GNBD 也可以作为 Fence 设备使用。

图 12-4 是一个电源 Fence 设备的应用实例。

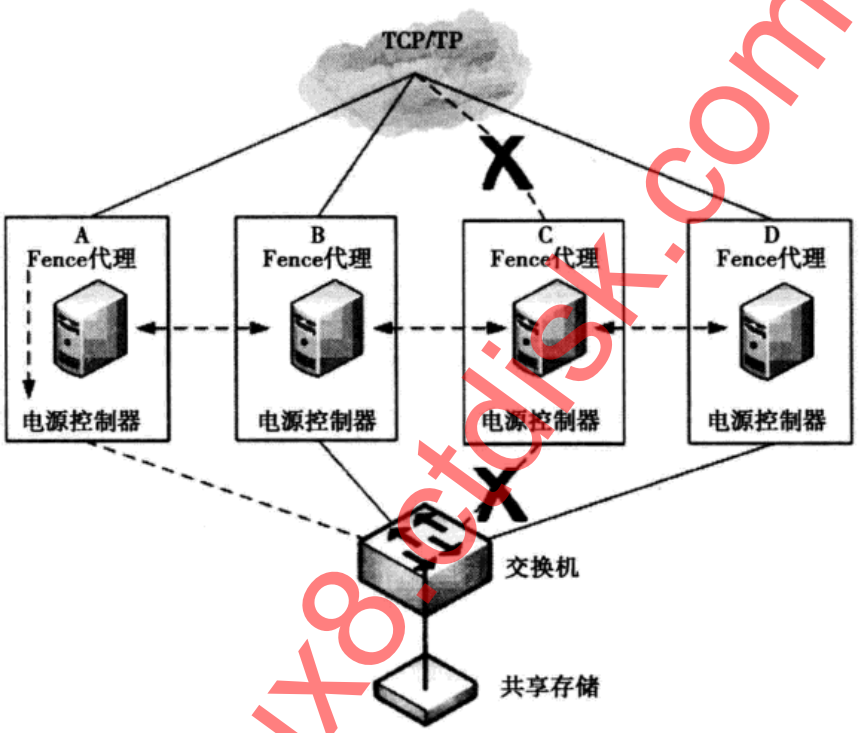

图 12-4 电源 Fence 设备的应用

在图 12-4 中，A、B、C、D 4 个节点组成一个集群，A 节点是主服务器，各个节点之间通过 CMAN 进程互相监控。当 A 节点发现 C 节点出现异常后，由 A 节点上的 Fence 代理进程通知电源控制器将 C 节点从集群中隔离。

同理，当 A 节点发生异常时，由 B、C、D 3 个节点中权值较高的节点来调用 Fence 代理进程，然后通过电源控制器将 A 节点隔离。Fence 代理执行成功后，权值较高的节点开始接管 A 节点的资源和服务，完成故障转移。

12.3.5 高可用性服务管理器

高可用性服务管理主要用来监督、启动和停止集群的应用、服务和资源。它提供了一种对集群服务的管理能力。当一个节点的服务失败时，高可用性集群服务管理进程可以将服务从这个失败节点转移到其他健康节点上。这种服务转移能力是自动的、透明的。

RHCS 通过 rgmanager 来管理集群服务，rgmanager 运行在每个集群节点上，在服务器上

对应的进程为 clurgmgrd。

在一个 RHCS 集群中，高可用性服务包括集群服务和集群资源两个方面。集群服务其实就是应用服务，例如 Apache、MySQL 等。集群资源有很多种，例如一个 IP 地址、一个运行脚本、ext3/GFS 文件系统等。

在 RHCS 集群中，高可用性服务是和一个失败转移域结合在一起的。所谓失败转移域是一个运行特定服务的集群节点的集合。在失败转移域中，可以为每个节点设置相应的优先级，通过优先级的高低来决定节点失败时服务转移的先后顺序，如果没有为节点指定优先级，那么集群高可用服务将在任意节点间转移。因此，通过创建失败转移域不但可以设定服务在节点间转移的顺序，而且可以限制某个服务仅在失败转移域指定的节点内进行切换。

RHCS 失败转移域的实现原理和运行结构如图 12-5 所示。

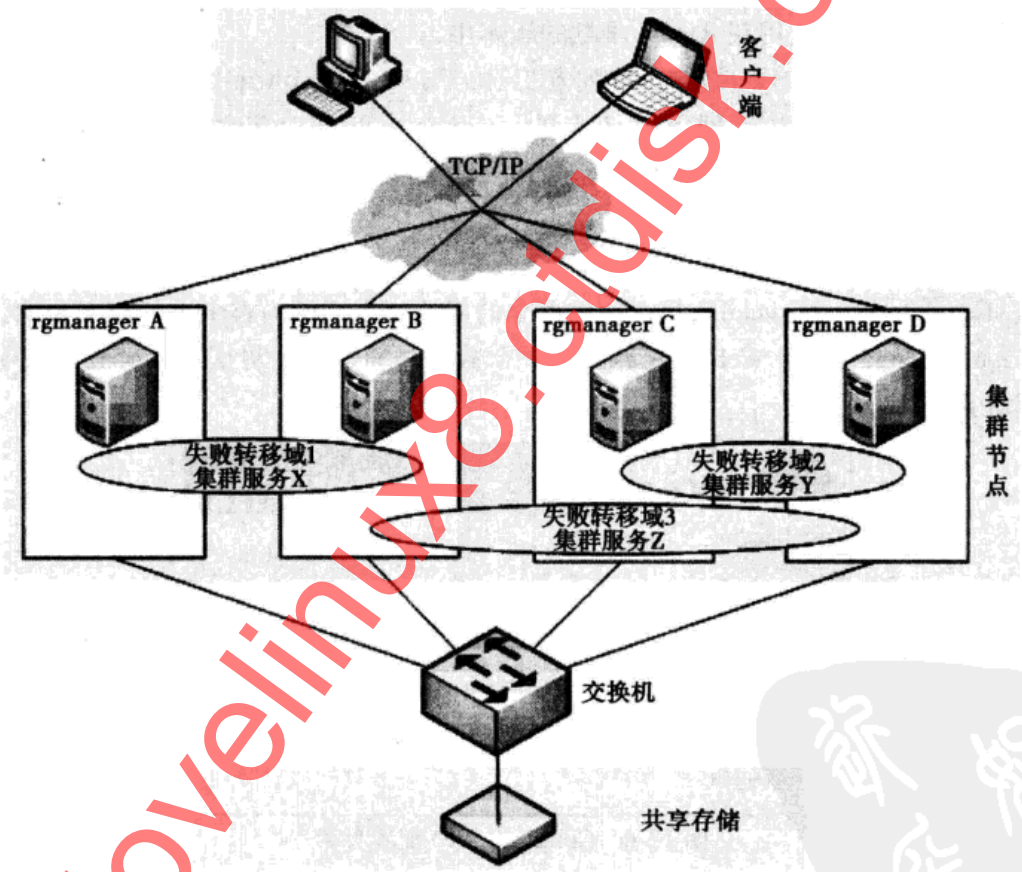

图 12-5　RHCS 失败转移域

在图 12-5 中，创建了 3 个失败转移域，失败转移域 1 由节点 A 和 B 组成，A 的优先级为 1，B 的优先级为 2，在正常情况下，A 为主节点，集群服务 X 运行在 A 节点上，当节点 A 失败时，集群服务 X 自动转移到 B 节点上，也只能转移到 B 节点上；失败转移域 2 由节点 C 和 D 组成，C 的优先级为 3，D 的优先级为 4，在正常情况下，C 节点上为主节点运行

集群服务 Y，当 C 节点出现异常时，D 节点进行服务接管，这和失败转移域 1 类似；失败转移域 3 由节点 B、C 和 D 组成，由于 B 的优先级高于其他两个节点，因此，在正常情况下，集群服务 Z 将运行在 B 节点上，如果 B 节点出现异常，服务 Z 将试图转移到 C 节点上，如果不能成功转移到 C 节点上，则向 D 节点转移。

12.3.6 集群配置和管理工具

RHCS 提供了多种集群配置和管理工具，常用的有基于 GUI 的 system-config-cluster、Conga 等，还提供了基于命令行的管理工具。

system-config-cluster 是一个用于创建集群和配置集群节点的图形化管理工具，它由集群节点配置和集群管理两个部分组成，分别用于创建集群节点配置文件和维护节点运行状态。system-config-cluster 一般用在 RHCS 早期的版本中。

Conga 是一种新的基于网络的集群配置工具。与 system-config-cluster 不同的是，Conga 是通过 Web 方式来配置和管理集群节点的。Conga 由两部分组成，分别是 Luci 和 ricci。Luci 安装在一台独立的计算机上，用于配置和管理集群，ricci 安装在每个集群节点上，Luci 通过 ricci 和集群中的每个节点进行通信。

RHCS 也提供了一些功能强大的集群命令行管理工具，常用的有 clustat、cman_tool、ccs_tool、fence_tool、clusvcadm 等，这些命令的用法将在后面内容中进行深入介绍。

图 12-6 是 Conga 基于 Web 配置界面的一个截图，通过这个界面可以轻松地配置和维护 RHCS 集群系统。

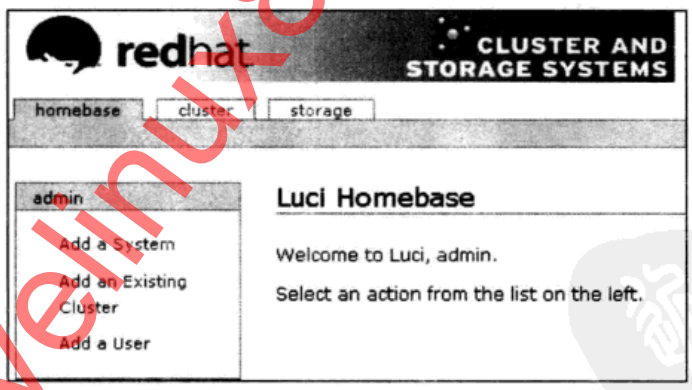

图 12-6 Conga 的 Web 配置界面

12.3.7 Redhat GFS

GFS 是 RHCS 为集群系统提供的一个存储解决方案，它允许集群的多个节点在块级别上共享存储，每个节点通过共享一个存储空间，保证了访问数据的一致性。更确切地说，GFS 是 RHCS 提供的一个集群文件系统，多个节点同时挂载一个文件系统分区，而使文件系统数

据不受破坏，这是单一的文件系统。如 ext3、ext2 所不能做到的。

为了实现多个节点对一个文件系统同时进行读写操作，GFS 使用锁管理器来管理 I/O 操作：当一个写进程操作一个文件时，这个文件就被锁定，此时不允许其他进程进行读写操作，直到这个写进程正常完成才释放锁，只有锁被释放了，其他读写进程才能对这个文件进行操作。另外，一个节点在 GFS 文件系统上修改数据后，这种修改操作会通过 RHCS 底层通信机制立即在其他节点上可见。

在搭建 RHCS 集群时，GFS 一般作为共享存储运行在每个节点上，并且可以通过 RHCS 管理工具对 GFS 进行配置和管理。这里需要说明的是，RHCS 和 GFS 之间的关系，一般初学者很容易混淆这两者之间的关系：运行 RHCS，GFS 不是必需的，只有在需要共享存储时，才需要 GFS 支持；而搭建 GFS 集群文件系统，必须要有 RHCS 的底层支持，所以安装 GFS 文件系统的节点，必须安装 RHCS 组件。

共享存储 GFS 的运行机制和构建结构如图 12-7 所示。

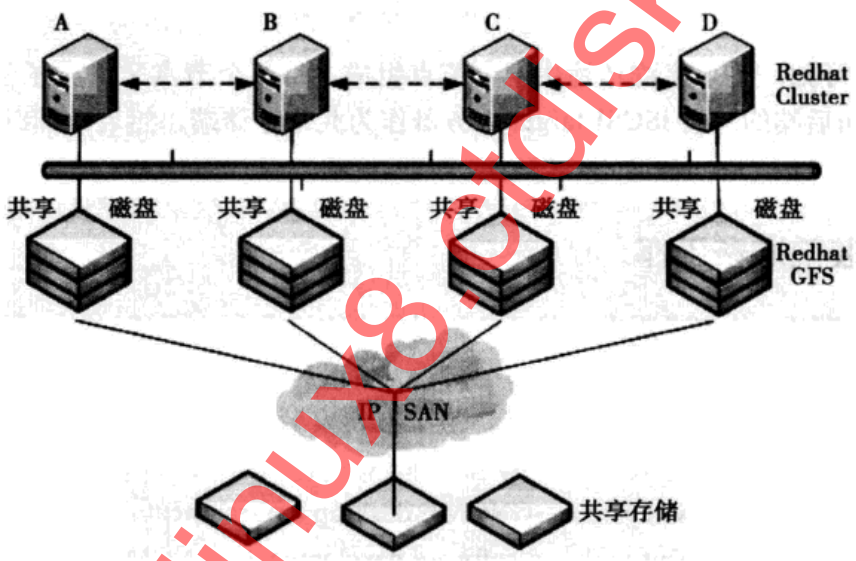

图 12-7　共享存储 GFS 的运行机制和构建结构

12.4　安装 RHCS

这个里要介绍的是 Web+MySQL 集群的构建。整个 RHCS 集群共由 4 台服务器组成，分别是由两台主机搭建的 Web 集群，由两台主机搭建的 MySQL 集群。在这种集群构架下，任何一台 Web 服务器故障，都由另一台 Web 服务器进行服务接管，同时，任何一台 MySQL 服务器故障，也由另一台 MySQL 服务器去接管服务，保证了整个应用系统服务的不间断运行。更详细的信息如图 12-8 所示。

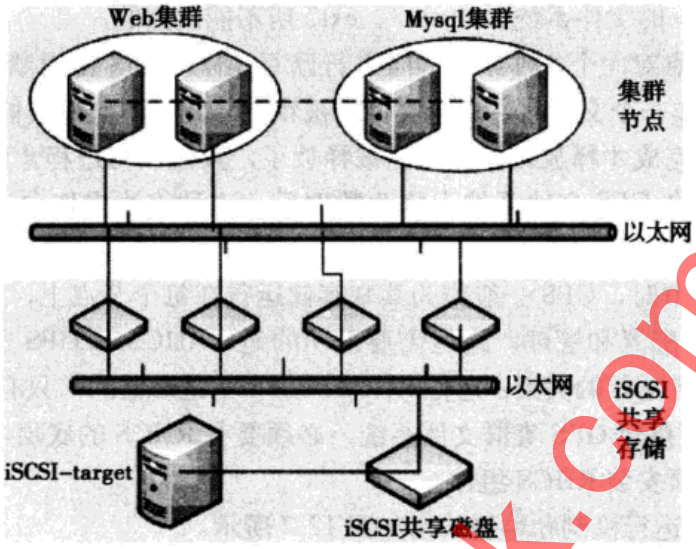

图 12-8 Web+MySQL 集群

从图 12-8 可知，整个集群环境由 4 个节点组成，每两个节点又组成了 Web 和 MySQL 的 HA 集群，而后端的一台 iSCSI-target 服务器作为共享存储端，然后通过以太网将数据共享给每个集群节点。

12.4.1 安装前准备工作

CentOS 是 RHEL 的克隆版本，并且免费提供 RHCS 的所有功能组件。因此下面的讲述以 CentOS 为准。

操作系统统一采用 CentOS 5.3 版本。为了方便安装 RHCS 套件，在安装操作系统时，建议选择如下这些安装包：

- 桌面环境：X Windows system、GNOME desktop environment。
- 开发工具：development tools、x software development、GNOME software development、kde software development。

地址规划如表 12-1 所示。

表 12-1 RHCS 的地址规划

主机名	IP 地址	主机用途	虚拟 IP
storgae-server	192.168.12.246	iSCSI 存储端 /rhcs 管理端	无
Mysql1	192.168.12.231	MySQL 主服务器	192.168.12.234
Mysql2	192.168.12.232	MySQL 备用服务器	
web1	192.168.12.230	Web 主服务器	192.168.12.233
web2	192.168.12.240	Web 备用服务器	

12.4.2 配置共享存储和 RHCS 管理端 Luci

本节以下的所有操作均在 storgae-server 主机，也就是 192.168.12.246 服务器上完成。

1. 配置 iSCSItarget

iSCSI Target 的安装与使用已经在第 7 章进行了深入讲述，这里不再详细说明，仅给出相关配置信息。

在 storgae-server 上安装完 iSCSI Target 后，开始进行配置。编辑 /etc/iet/ietd.conf 文件，增加如下配置信息：

```
Target iqn.2002-04.net.ixdba:sdc
Lun 0 Path=/dev/sdc,Type=fileio
```

其中，/dev/sdc 是 storgae-server 主机共享给集群节点使用的共享空间。

接着编辑 /etc/iet/initiators.allow 文件，修改后的文件内容如下：

```
iqn.2002-04.net.ixdba:sdc 192.168.12.231,192.168.12.232,192.168.12.240,192.168.12.230
```

这个设置是赋予集群的 4 个节点对存储的访问权限。

修改完两个文件后，重启 iSCSI Target 服务即可。过程如下：

```
[root@storgae-server ~]# /etc/init.d/iscsi-target  start
```

至此，iSCSI Target 存储端的设置已经完成了。

2. 安装 Luci

Luci 是 RHCS 基于 Web 的集群配置管理工具，可以从系统光盘找到对应的 Luci 安装包。安装过程如下：

```
[root@storgae-server ~]#rpm -ivh luci-0.12.2-12.el5.centos.1.i386.rpm
```

安装完成，执行 Luci 初始化操作如下：

```
[root@storgae-server ~]#luci_admin init
Initializing the Luci server
Creating the 'admin' user
Enter password:
Confirm password:

Please wait...
 The admin password has been successfully set.
Generating SSL certificates...
Luci server has been successfully initialized
```

输入两次密码后，就创建了一个默认登录 Luci 的用户 admin。

最后，执行如下命令启动 Luci 服务即可：

```
[root@storgae-server ~]# /etc/init.d/luci  start
```

成功启动 Luci 服务后，就可以通过 https://ip:8084 访问 Luci 了。

为了使 Luci 能够访问集群其他节点，还需要在 /etc/hosts 中增加如下内容：

```
192.168.12.231  Mysql1
192.168.12.232  Mysql2
192.168.12.230  web1
192.168.12.240  web2
```

到这里为止，在 storgae-server 主机上的设置完成。

12.4.3 在集群节点上安装 RHCS 软件包

为了保证集群的每个节点间可以互相通信，需要将每个节点的主机名信息加入到 /etc/hosts 文件中。修改完成的 /etc/hosts 文件内容如下：

```
127.0.0.1       localhost
192.168.12.230  web1
192.168.12.240  web2
192.168.12.231  Mysql1
192.168.12.232  Mysql2
```

将此文件依次复制到集群每个节点的 /etc/hosts 文件中。

RHCS 软件包的安装有两种方式：可以通过 Luci 管理界面，在创建集群时，通过在线下载方式自动安装；也可以直接从操作系统光盘中找到所需软件包进行手动安装。由于在线安装方式受网络和速度的影响，不建议采用，这里通过手动方式来安装 RHCS 软件包。

安装 RHCS，主要安装的组件包有 cman、gfs2 和 rgmanager。在安装这些软件包时可能需要其他依赖的系统包，只需按照提示进行安装即可。

下面是一个安装 RHCS 软件包的脚本文件，也是安装 RHCS 的一个清单。

```
#install cman

rpm -ivh perl-XML-NamespaceSupport-1.09-1.2.1.noarch.rpm
rpm -ivh perl-XML-SAX-0.14-8.noarch.rpm
rpm -ivh perl-XML-LibXML-Common-0.13-8.2.2.i386.rpm
rpm -ivh perl-XML-LibXML-1.58-6.i386.rpm
rpm -ivh perl-Net-Telnet-3.03-5.noarch.rpm
rpm -ivh expect-2.3-3.el5.noarch.rpm
rpm -ivh openais-0.80.6-16.el5_5.2.i386.rpm
rpm -ivh cman-2.0.115-34.el5.i386.rpm

#install ricci

rpm -ivh modcluster-0.12.1-2.el5.centos.i386.rpm
rpm -ivh ricci-0.12.2-12.el5.centos.1.i386.rpm

#install gfs2
```

```
rpm -ivh   gfs2-utils-0.1.62-20.el5.i386.rpm

#install rgmanager
rpm -ivh  rgmanager-2.0.52-6.el5.centos.i386.rpm
```

在这个脚本中，ricci 是 Conga 的一部分，用于和管理端 Luci 进行通信。

将此脚本分别在集群的 4 个节点上执行，如果没有报错，那么证明 RHCS 底层软件包已经成功安装了。

12.4.4 在集群节点上安装和配置 iSCSI 客户端

安装 iSCSI 客户端是为了和 iSCSI Target 服务器端进行通信，进而将共享磁盘导入到各个集群节点。这里以集群节点 web1 为例，介绍如何安装和配置 iSCSI，其他节点的安装和配置方式与 web1 节点完全相同，不再一一讲述。

iSCSI 客户端的安装和配置非常简单，只需如下几个步骤即可完成：

```
[root@web1 rhcs]# rpm -ivh iscsi-initiator-utils-6.2.0.871-0.16.el5.i386.rpm
[root@web1 rhcs]# /etc/init.d/iscsi restart
[root@web1 rhcs]# iscsiadm -m discovery -t sendtargets -p 192.168.12.246
[root@web1 rhcs]# /etc/init.d/iscsi restart
[root@web1 rhcs]# fdisk -l
Disk /dev/sdb: 10.7 GB, 10737418240 bytes
64 heads, 32 sectors/track, 10240 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes
Disk /dev/sdb doesn't contain a valid partition table
```

通过 fdisk 的输出可知，/dev/sdb 就是从 iSCSI Target 共享过来的磁盘分区。

至此，安装工作全部结束。

12.5 配置 RHCS 高可用集群

配置 RHCS，其核心就是配置 /etc/cluster/cluster.conf 文件。下面通过 Web 管理界面和命令行操作两种方法介绍如何配置 cluster.conf 文件。

在 storgae-server 主机上启动 Luci 服务，然后通过浏览器访问 https://192.168.12.246:8084/，就可以打开 Luci 登录界面，如图 12-9 所示。

成功登录后，Luci 有三个配置选项，分别是“homebase”、“cluster”和“storage”，其中，“cluster”主要用于创建和配置集群系统，“storage”用于创建和管理共享存储，而“homebase”主要用于添加、更新、删除 cluster 系统和 storage 设置，同时也可以创建和删除 Luci 登录用户。CentOS 下的 Luci 主界面如图 12-10 所示。

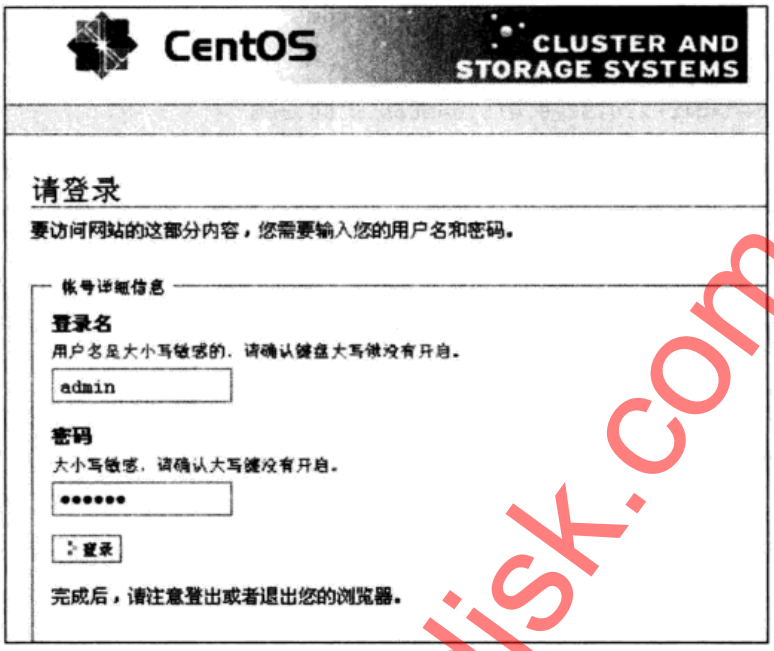

图 12-9 Luci 登录界面

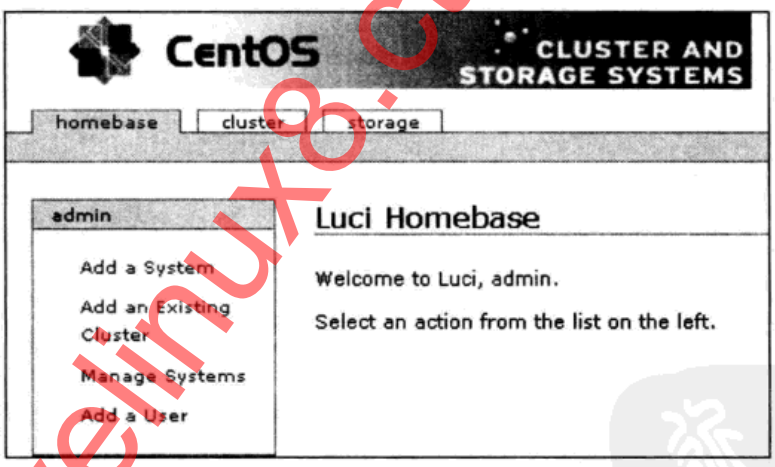

图 12-10 CentOS 下的 Luci 主界面

12.5.1 创建一个 cluster

登录 Luci 后，切换到“cluster”选项，然后单击左边的“clusters”选择框中的“Create a New Cluster”，增加一个 cluster，如图 12-11 所示。

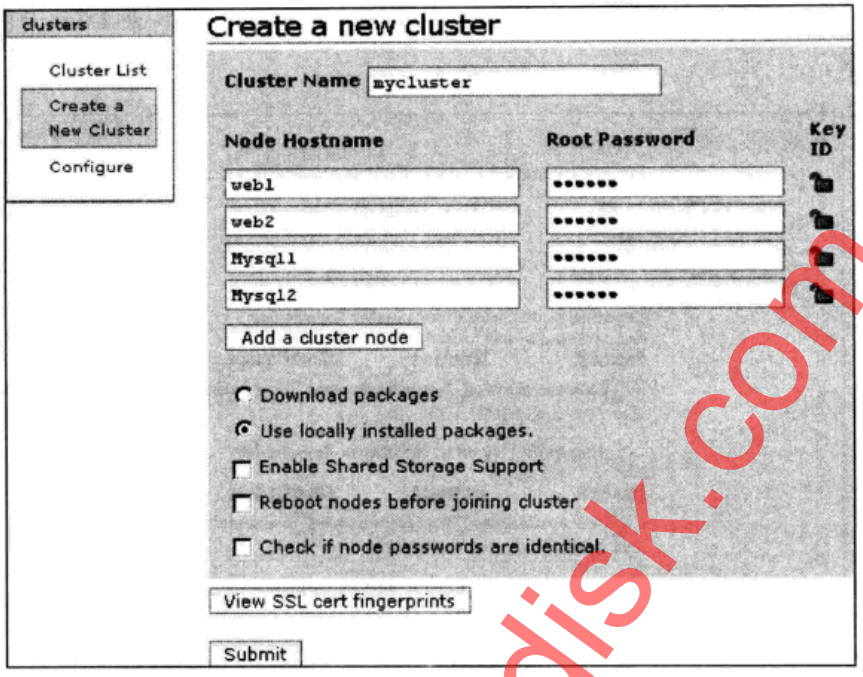

图 12-11 创建 cluster

在图 12-11 中，创建的 cluster 名称为“mycluster”。“Node Hostname”表示每个节点的主机名称，“Root Password”表示每个节点的 root 用户密码。每个节点的 root 密码可以相同，也可以不同。

在图 12-11 中的下面 5 个选项中，“Download packages”表示在线下载并自动安装 RHCS 软件包，而“Use locally installed packages”表示利用本地安装包进行安装，由于 RHCS 组件包在前面已经手动安装，所以这里选择本地安装即可。其余的 3 个复选框分别是启用共享存储支持（“Enable Shared Storage Support”）、节点加入集群时重启系统（“Reboot nodes before joining cluster”）和检查节点密码的一致性（“Check if node passwords are identical”），这些创建 cluster 的设置，可选可不选，这里不做任何选择。

“View SSL cert fingerprints”用于验证集群各个节点与 Luci 通信是否正常，并检测每个节点的配置是否可以创建集群。如果检测失败，会给出相应的错误提示信息；如果验证成功，会输出成功信息，如图 12-12 所示。

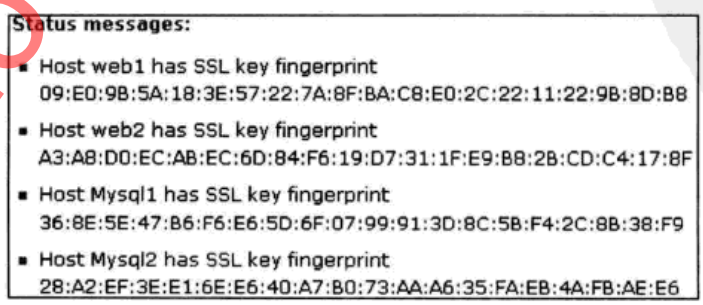

图 12-12 SSL cert 验证成功信息

所有选项填写完成后，单击“Submit”按钮进行提交。接下来 Luci 开始创建 cluster，如图 12-13 所示。

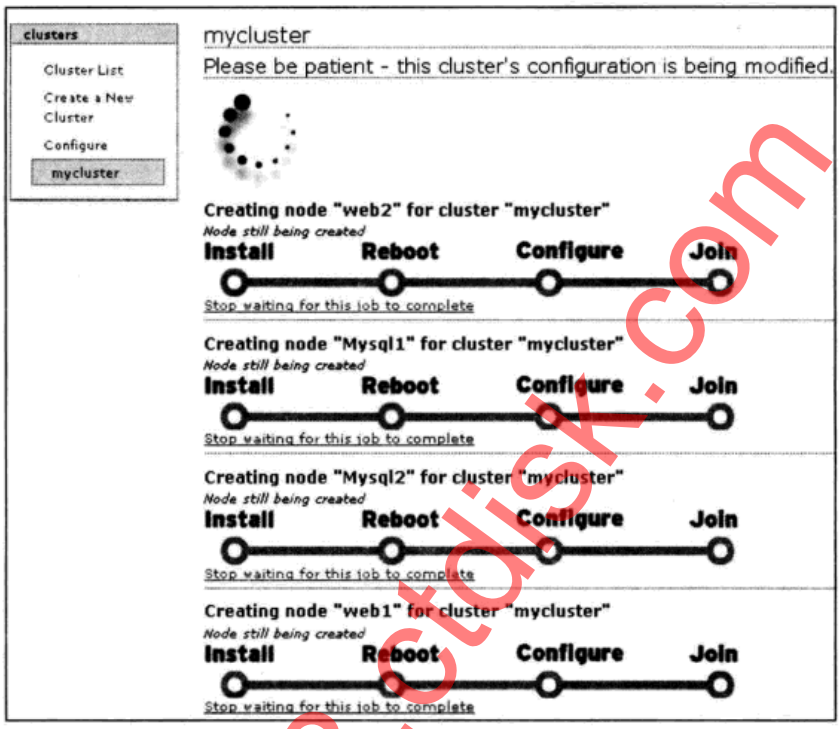

图 12-13 Luci 正在创建 cluster

在经过 Install——Reboot——Configure——Join 4 个过程后，如果没有报错，“mycluster”就创建完成了。其实创建 cluster 的过程，就是 Luci 将设定的集群信息写入到每个集群节点配置文件中的过程。集群创建成功后，默认显示“mycluster”的集群全局属性列表，如图 12-14 所示。

依次单击“cluster → Cluster List”来查看创建的 mycluster 的状态，如图 12-15 所示。

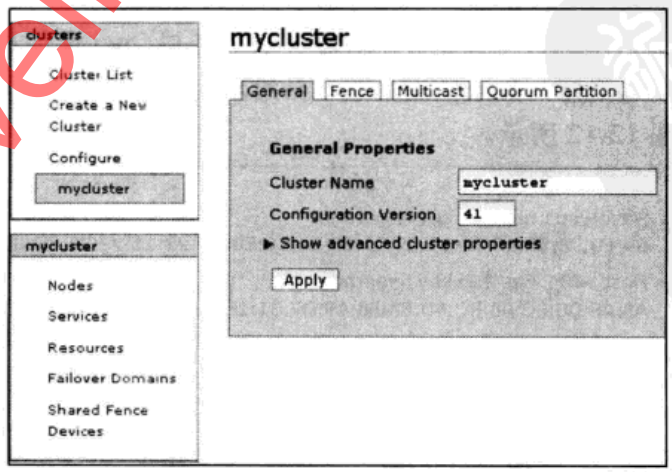

图 12-14 mycluster 的全局属性列表

在图 12-14 中的选项含义如下：

- “General”，用于显示集群的名称、集群配置文件的版本号和集群高级属性配置。
- “Fence”，提供配置 Fence daemon 属性的参数接口。
- “Multicast”，用于配置集群多播属性，也就是配置集群的心跳。配置选项有：“集群选择默认多播地址”、“手工指定多播地址”，默认选择“集群选择默认多播地址”。
- “Quorum Partition”，用于配置集群表决磁盘及相关属性，在后面会进行详细讲述。

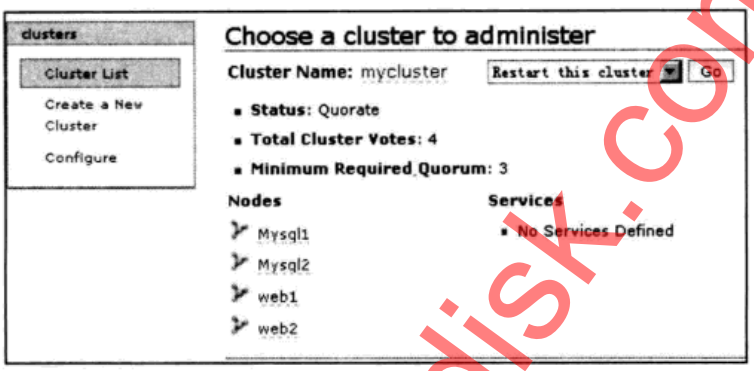

图 12-15 mycluster 的运行状态

从图 12-15 可知，mycluster 集群下有 4 个节点，在正常状态下，节点 Nodes 名称和 Cluster Name 均显示为绿色，如果出现异常，将显示为红色。

单击 Nodes 下的任意一个节点名称，可以查看此节点的运行状态，如图 12-16 所示。

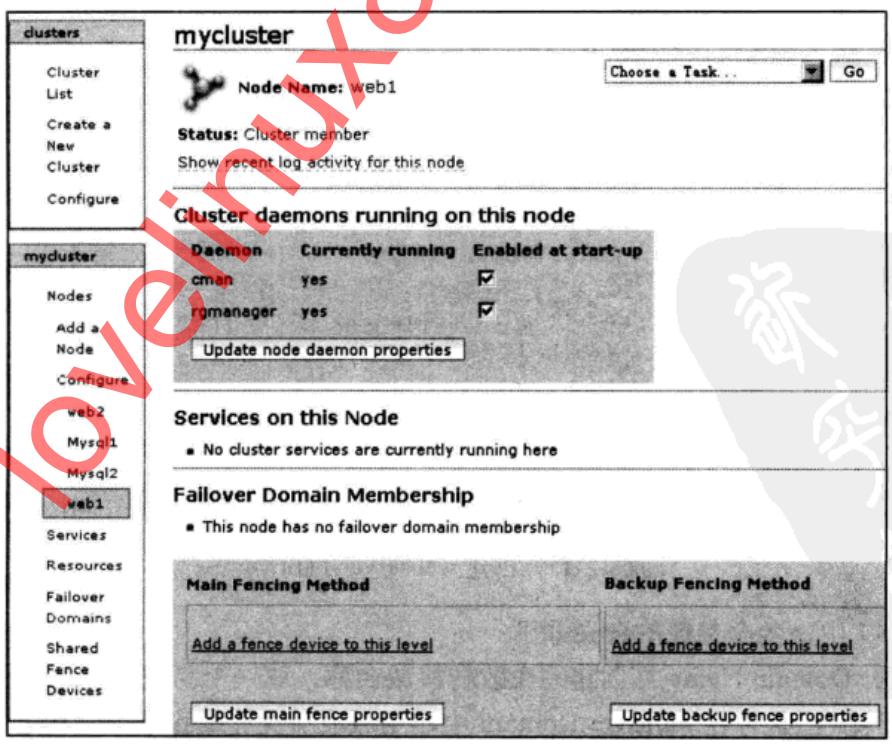

图 12-16 web1 节点运行状态

从图 12-16 可以看出，cman 和 rgmanager 服务运行在每个节点上，并且这两个服务需要开机自动启动，它们是 RHCS 的核心守护进程。如果这两个服务在某个节点上没有启动，可以通过命令行方式手工启动。命令如下：

```
/etc/init.d/cman start
/etc/init.d/rgmanager start
```

服务启动成功后，单击图 12-16 中的“Update node daemon properties”按钮，更新节点的状态。

通过上面的操作，一个简单的 cluster 就创建完成了。但是这个 cluster 目前还不能工作，还需要为这个 cluster 创建 Failover Domain、Resources、Service、Shared Fence Device 等。下面依次进行介绍。

12.5.2 创建 Failover Domain

Failover Domain 是配置集群的失败转移域，通过失败转移域可以将服务和资源的切换限制在指定的节点间。下面的操作将创建两个失败转移域，分别是 webserver-Failover 和 mysql-failover。

单击“cluster”，然后在“Cluster List”中单击“mycluster”，接着，在左下端的“mycluster”栏中依次单击“Failover Domains”→“Add a Failover Domain”，增加一个 Failover Domain，如图 12-17 所示。

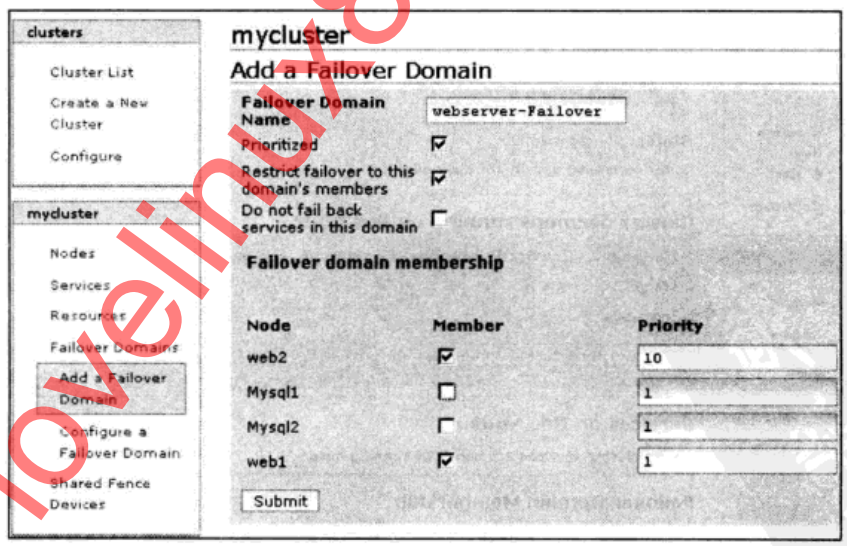

图 12-17 创建 webserver-Failover

在图 12-17 中，各个参数的含义如下：

- Failover Domain Name：创建的失败转移域名称，取一个易记的名字即可。
- Prioritized：是否在 Failover Domain 中启用域成员优先级设置，这里选择启用。

- Restrict failover to this domain's members：表示是否在失败转移域成员中启用服务故障切换限制，这里选择启用。
- Do not fail back services in this domain：表示在这个域中使用故障切回功能，也就是说，主节点故障时，备用节点会自动接管主节点服务和资源，当主节点恢复正常时，集群的服务和资源会从备用节点自动切换到主节点。

接下来，在 Failover domain membership 的“Member”复选框中选择加入此域的节点，这里选择的是 web1 和 web2 节点。然后，在“Priority”处将 web1 的优先级设置为 1，web2 的优先级设置为 10。需要说明的是，“Priority”设置为 1 的节点，优先级是最高的，随着数值的增大，节点的优先级也依次降低。

所有设置完成，单击“Submit”按钮，开始创建 Failover Domain。

按照前面的介绍，继续添加第二个失败转移域 mysql-failover。在 Failover domain membership 的 Member 复选框中，选择加入此域的节点，这里选择 Mysql1 和 Mysql2 节点。然后，在“priority”处将 Mysql1 的优先级设置为 2，Mysql2 的优先级设置为 8，如图 12-18 所示。

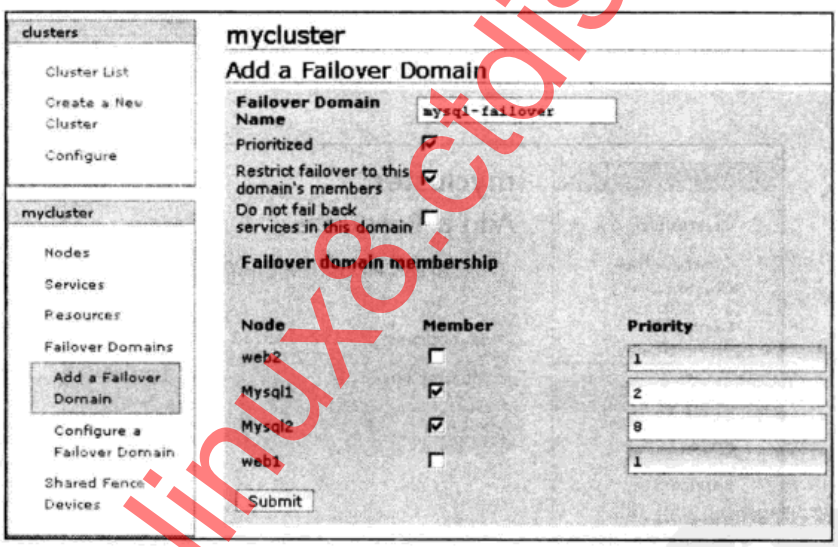

图 12-18 创建 mysql-failover

12.5.3 创建 Resources

Resources 是集群的核心，主要包含服务脚本、IP 地址、文件系统等。RHCS 提供的资源如图 12-19 所示。

在图 12-9 左下端 mycluster 选择框中，依次单击“Resources”→“Add a Resource”，首先添加一个虚拟 IP 地址，如图 12-20 所示。

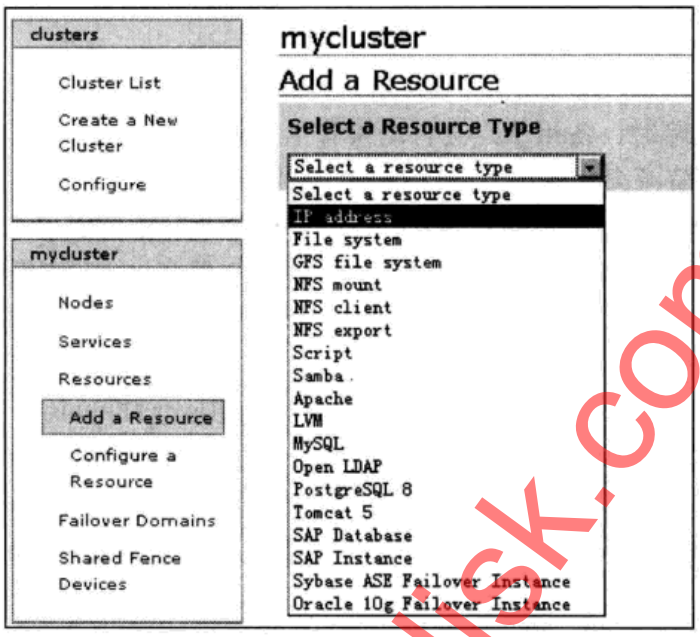

图 12-19 RHCS 资源一览

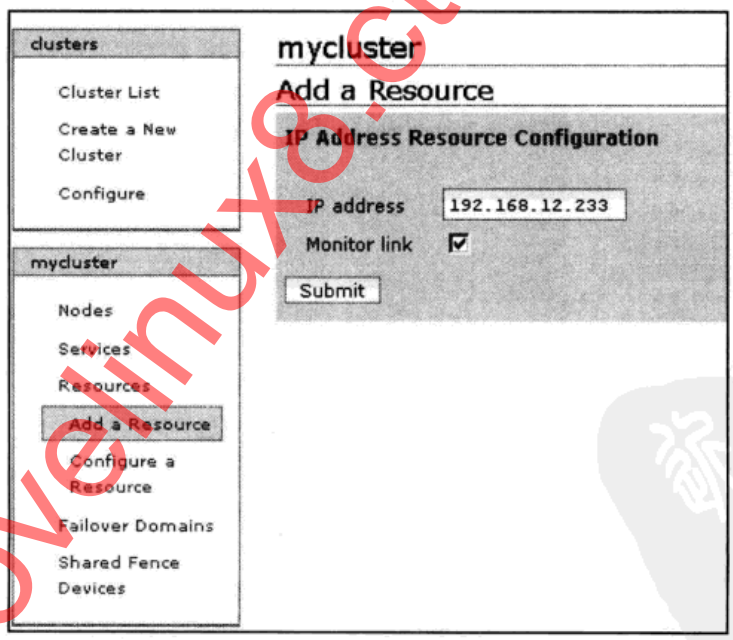

图 12-20 添加 IP 地址

接着，再添加一个 HTTP 服务脚本资源，如图 12-21 所示。

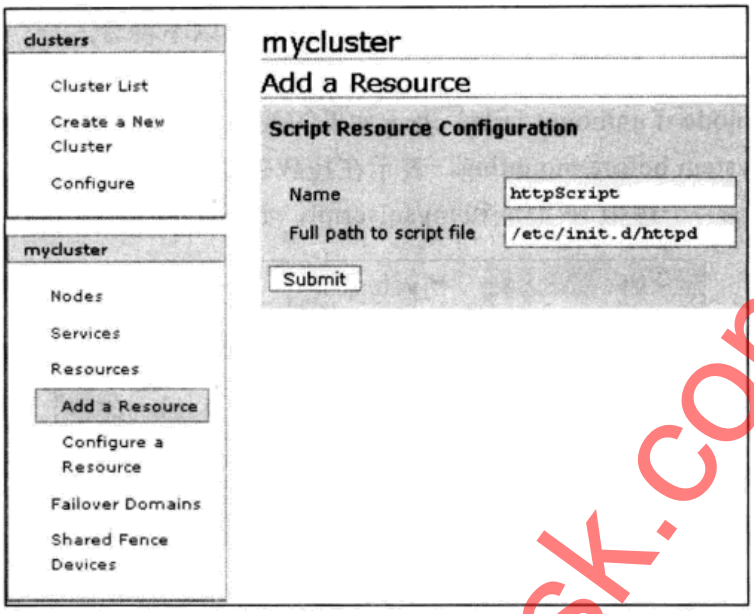

图 12-21　添加 httpScript 资源

然后添加一个 ext3 文件系统资源，如图 12-22 所示。

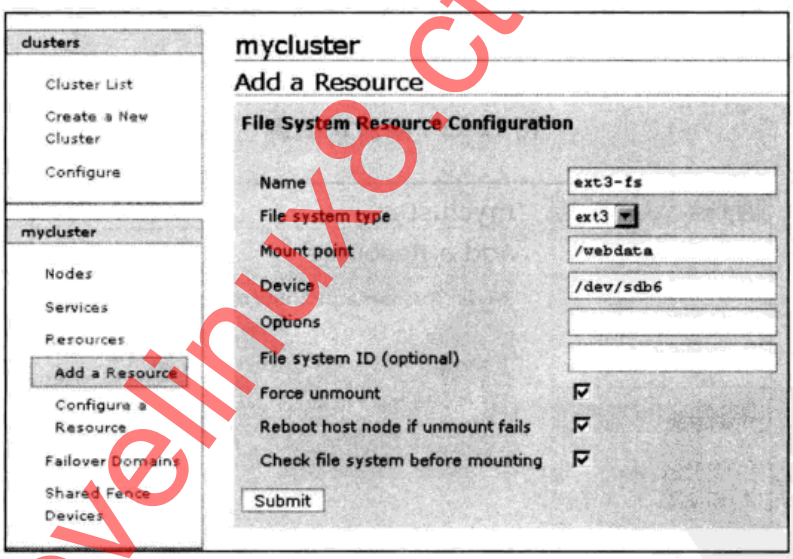

图 12-22　添加文件系统资源

图 12-22 中的各个选项的含义如下：

- Name，表示添加文件系统资源的名称，任意指定一个即可。
- File system type，指定文件系统的类型，默认的选项有 ext2、ext3。
- Mount point，指定文件系统的挂载点，这个挂载点要在服务器上存在。
- Device，指定需要挂载共享磁盘的设备标识。

- Options，指定挂载文件系统时的一些挂载参数，可以不填写。
- Force unmount，表示执行强制卸载文件系统操作。
- Reboot host node if unmount fails，表示如果卸载分区失败就重启系统。
- Check file system before mounting，表示在挂载磁盘前，首先检测文件系统。

最后，再次添加一个虚拟 IP 地址和 mysqlscript，如图 12-23、图 12-24 所示。

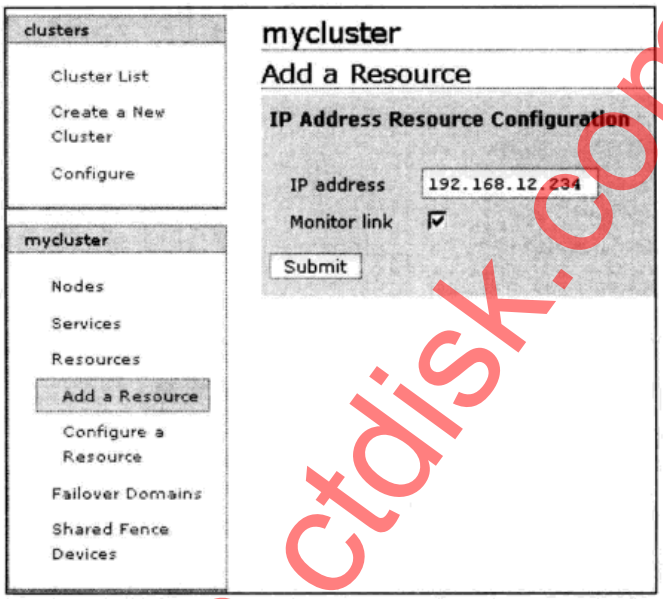

图 12-23 添加 IP 资源

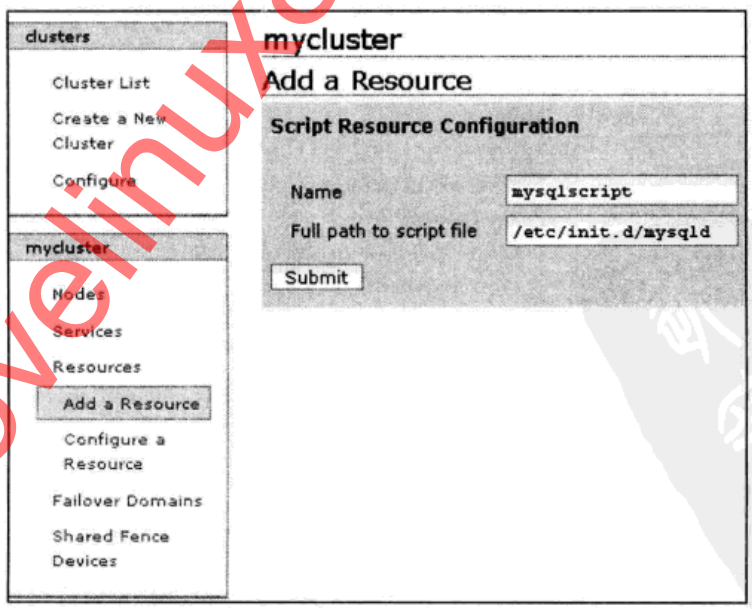

图 12-24 添加 mysqlscript

所有资源添加完成后，单击“mycluster”选择框中的“Resources”选项，显示所有已添加的资源，如图 12-25 所示。

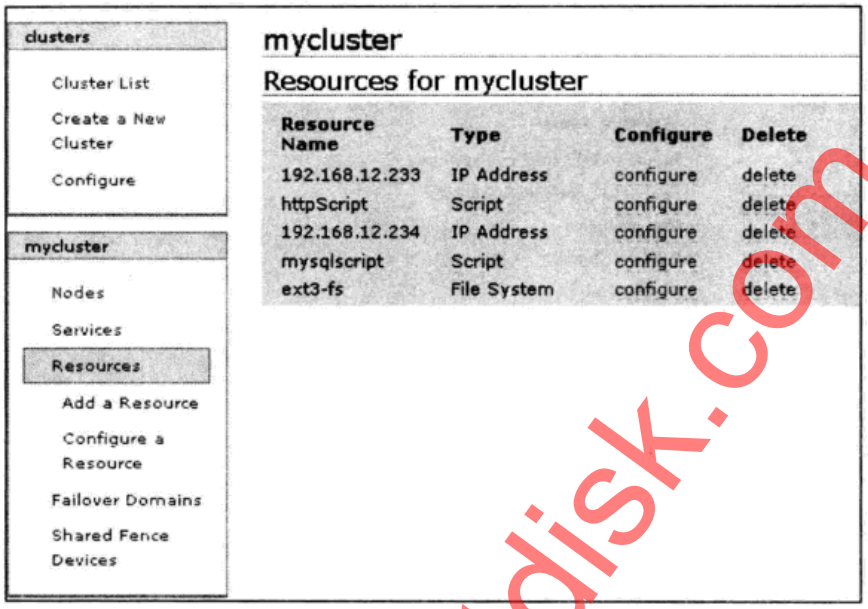

图 12-25 已添加资源的列表

12.5.4 创建 Service

在 RHCS 高可用性集群中，服务（Service）指的是一系列资源（Resource）的集合，而不仅仅是 /etc/init.d/httpd 之类的应用程序。在一个集群系统中，如果主节点宕机，备用节点仅接管主节点的应用程序服务是不够的，还必须要接管集群服务 IP 地址、共享磁盘分区等。在故障切换时，集群服务的转移是有先后顺序的，如果集群中有共享磁盘，那么主节点会先卸载共享磁盘，接着释放集群 IP 地址，最后停止集群应用程序服务；而备用节点刚好相反，在接到故障节点成功卸载共享磁盘的通知后，开始执行挂载磁盘，接着启用集群 IP，最后，启动应用程序服务。这样就完成了整个集群系统高可用服务的转移。

单击“cluster”，然后在“Cluster List”中单击“mycluster”，接着，在左下端的“mycluster”栏中单击“Services”→“Add a Service”，在集群中添加一个服务，如图 12-26 所示。

在图 12-26 中，创建了一个名为 webserver 的服务，设定服务自动启动，然后将 webserver-Failover 加入到此服务中，并且设定服务的恢复策略为“Relocate”。“Relocate”的含义是：当服务在一个节点上不可用时，立刻将服务转移到其他节点。在恢复策略中还可以选择“Restart”，也就是服务不可用时，尝试重启服务，而不是切换到其他节点。

单击“Add a resource to this service”按钮，增加一个资源到这个服务中，选择“Use an existing global resource”，将已经存在的资源加入到新建的服务中，这里依次加入集群 IP 地

址 192.168.12.233 和 httpScript。所有资源添加完成后，单击“Submit”按钮完成一个服务的创建。

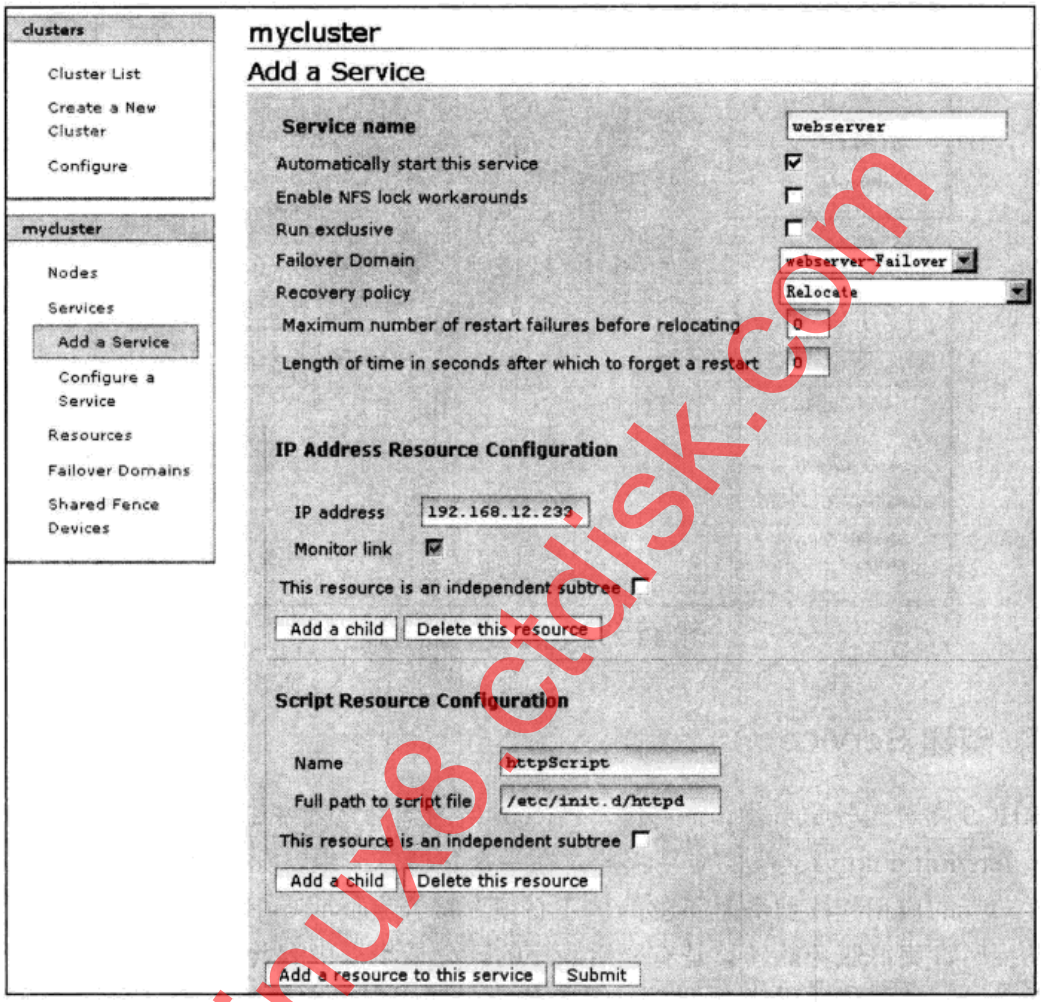

图 12-26 添加一个 webserver 服务

接着，继续创建第二个名为 mysqlserver 的服务，设置服务自动启动，并将 mysql-Failover 加入到此服务中，设定恢复策略为“Relocate”，最后将集群另一个服务 IP 地址 192.168.12.234 和 mysqlScript 加入到此服务中，如图 12-27 所示。

所有服务添加完成后，如果应用程序设置正确，服务将自动启动，单击“cluster”，然后在 Cluster List 中可以看到两个服务的启动状态，在正常情况下，均显示为绿色，如图 12-28 所示。

RHCS 也提供了根据命令行查看集群状态的方式，通过 clustat 命令可以更清晰地了解集群每个节点以及服务的运行状态。操作如下：

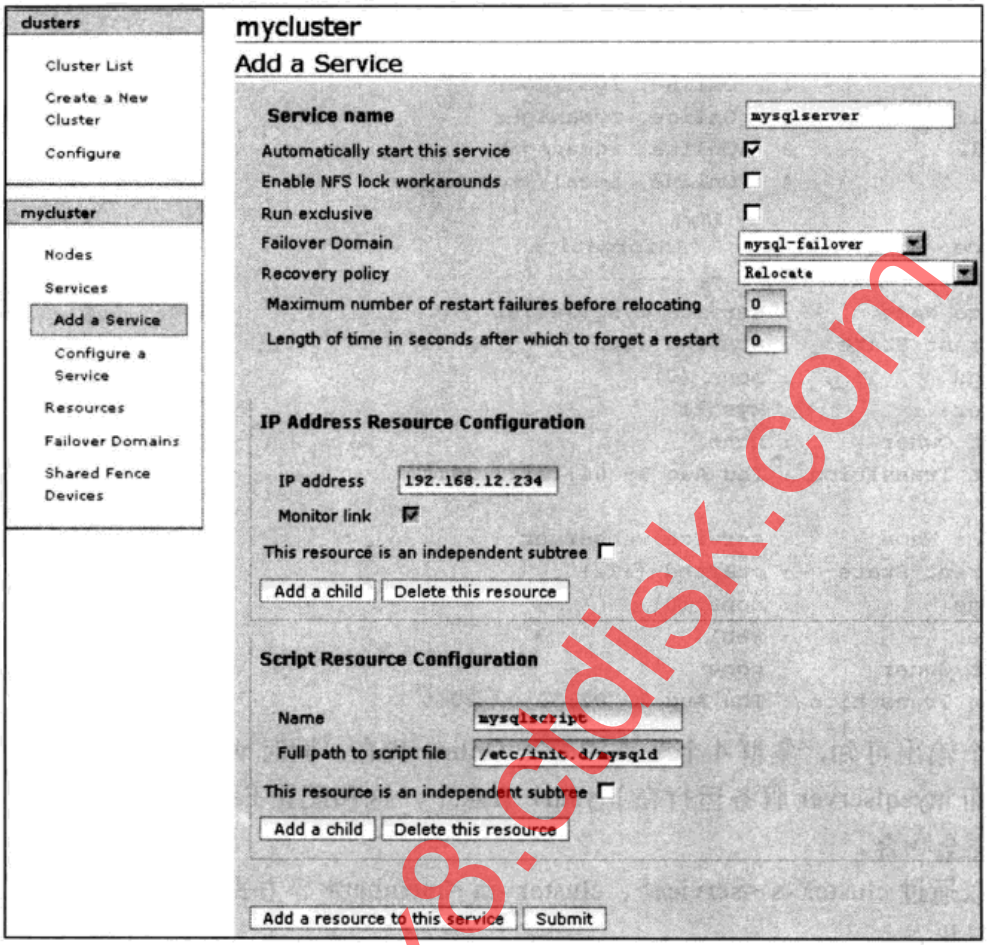

图 12-27 添加一个 mysqlserver 服务

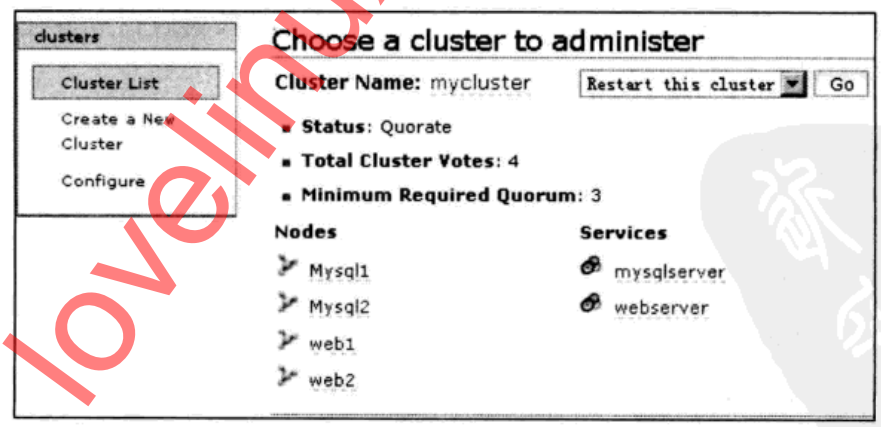

图 12-28 mycluster 运行状态

```
[root@web1 ~]#  clustat -l
Cluster Status for mycluster @ Thu Aug 19 16:39:46 2010
Member Status: Quorate
```

```
 Member Name   ID   Status
 ------ ----        ----  ------
 web2              1   Online, rgmanager
 Mysql1            2   Online, rgmanager
 Mysql2            3   Online, rgmanager
 web1              4   Online, Local, rgmanager

Service                    Information
-------                    -----------
Service Name      : service:mysqlserver
  Current State   : started (112)
  Flags           : none (0)
  Owner           : Mysql1
  Last Owner      : none
  Last Transition : Thu Aug 19 04:12:13 2010

Service Name      : service:webserver
  Current State   : started (112)
  Flags           : none (0)
  Owner           : web1
  Last Owner      : none
  Last Transition : Thu Aug 19 02:59:07 2010
```

由这个输出可知，集群4个节点均处于Online状态，同时webserver服务运行在web1节点上，而mysqlserver服务运行在Mysql1节点上，这和配置Failover Domain时指定的节点优先级完全吻合。

还可以通过cluster -s <service>、cluster -m <member>等方式单独查看一个服务或一个节点的运行状态。

了解RHCS系统中每个节点的运行状态，对集群的维护和问题的排查至关重要。Luci提供的Web界面虽然比较直观，但目前还不是很稳定，这里建议通过命令方式来查看和管理RHCS集群系统，Web界面可只作为配置cluster.conf文件所用。

12.5.5 配置存储集群GFS

在前面的内容中，已经通过storgae-server主机将一个磁盘分区共享给了集群系统的4个节点，接下来将进行磁盘分区、格式化、创建文件系统等操作。

1. 对磁盘进行分区

可以在集群系统任意节点上对共享磁盘分区进行磁盘的分区和格式化，这里选择在节点web1上进行。首先对共享磁盘进行分区，操作如下：

```
[root@web1 ~]# fdisk /dev/sdb
Command (m for help): n
Command action
   e   extended
```

```
   p   primary partition (1-4)
e
Partition number (1-4): 1
First cylinder (1-10240, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-10240, default 10240):
Using default value 10240

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (1-10240, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-10240, default 10240): +5G

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (4770-10240, default 4770):
Using default value 4770
Last cylinder or +size or +sizeM or +sizeK (4770-10240, default 10240): +4G

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (8586-10240, default 8586):
Using default value 8586
Last cylinder or +size or +sizeM or +sizeK (8586-10240, default 10240):
Using default value 10240

Command (m for help): p

Disk /dev/sdb: 10.7 GB, 10737418240 bytes
64 heads, 32 sectors/track, 10240 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1       10240    10485744    5  Extended
/dev/sdb5               1        4769     4883424   83  Linux
/dev/sdb6            4770        8585     3907568   83  Linux
/dev/sdb7            8586       10240     1694704   83  Linux
Command (m for help):  w
The partition table has been altered!
```

```
Calling ioctl() to re-read partition table.
Syncing disks.
```

这里将共享磁盘分为 3 个有效分区，分别将 /dev/sdb5 用于 GFS 文件系统，将 /dev/sdb6 用于 ext3 文件系统，而将 /dev/sdb7 用于表决磁盘。表决磁盘将在后面进行讲述。

2. 格式化磁盘

接下来，在 web1 节点上将磁盘分区分别格式化为 ext3 和 gfs2 文件系统。操作如下：

```
[root@web1 ~]# mkfs.ext3 /dev/sdb6
[root@web1 ~]# mkfs.gfs2 -p lock_dlm -t mycluster:my-gfs2 -j 4 /dev/sdb5
This will destroy any data on /dev/sdb5.
  It appears to contain a gfs2 filesystem.

Are you sure you want to proceed? [y/n] y

   Device:                    /dev/sdb5
Blocksize:                    4096
Device Size                   4.66 GB (1220856 blocks)
Filesystem Size:              4.66 GB (1220854 blocks)
Journals:                     4
Resource Groups:              19
Locking Protocol:             "lock_dlm"
Lock Table:                   "mycluster:my-gfs2"
UUID:                         F5207D5D-88AE-6E6D-DB44-438BCF204353
```

其中参数含义如下：

- -p lock_dlm，定义为 DLM 锁方式，如果没有此参数，当在两个系统中同时挂载此分区时就会像 ext3 格式一样，两个系统的信息不能同步。
- -t mycluster:my-gfs2，指定 DLM 锁所在的表名称，mycluster 就是 RHCS 集群的名称，必须与 cluster.conf 文件中 Cluster 标签的 name 值相同。
- -j 4，设定 GFS2 文件系统最多支持多少个节点同时挂载，这个值可以通 gfs2_jadd 命令在使用中动态调整。
- /dev/sdb5，指定要格式化的分区设备标识。

所有操作完成后，重启集群所有节点，保证划分的磁盘分区能够被所有节点识别。

3. 挂载磁盘

所有节点重新启动后，就可以挂载文件系统了，依次在集群的每个节点上执行如下操作，将共享文件系统挂载到 /gfs2 目录下。

```
[root@web1 ~]#mount -t gfs2 /dev/sdb5 /gfs2 -v
/sbin/mount.gfs2: mount /dev/sdb5 /gfs2
/sbin/mount.gfs2: parse_opts: opts = "rw"
/sbin/mount.gfs2:   clear flag 1 for "rw", flags = 0
/sbin/mount.gfs2: parse_opts: flags = 0
/sbin/mount.gfs2: parse_opts: extra = ""
```

```
/sbin/mount.gfs2: parse_opts: hostdata = ""
/sbin/mount.gfs2: parse_opts: lockproto = ""
/sbin/mount.gfs2: parse_opts: locktable = ""
/sbin/mount.gfs2: message to gfs_controld: asking to join mountgroup:
/sbin/mount.gfs2: write "join /gfs2 gfs2 lock_dlm mycluster:my-gfs2 rw /dev/sdb5"
/sbin/mount.gfs2: message from gfs_controld: response to join request:
/sbin/mount.gfs2: lock_dlm_join: read "0"
/sbin/mount.gfs2: message from gfs_controld: mount options:
/sbin/mount.gfs2: lock_dlm_join: read "hostdata=jid=3:id=65540:first=0"
/sbin/mount.gfs2: lock_dlm_join: hostdata: "hostdata=jid=3:id=65540:first=0"
/sbin/mount.gfs2: lock_dlm_join: extra_plus: "hostdata=jid=3:id=65540:first=0"
/sbin/mount.gfs2: mount(2) ok
/sbin/mount.gfs2: lock_dlm_mount_result: write "mount_result /gfs2 gfs2 0"
/sbin/mount.gfs2: read_proc_mounts: device = "/dev/sdb5"
/sbin/mount.gfs2: read_proc_mounts: opts = "rw,hostdata=jid=3:id=65540:first=0"
```

通过“-v”参数可以输出挂载 gfs2 文件系统的过程，有助于理解 gfs2 文件系统和问题排查。

为了能让共享文件系统开机自动挂载磁盘，将下面内容添加到每个集群节点的 /etc/fstab 文件中。

```
#GFS   MOUNT POINTS
/dev/sdb5      /gfs2            gfs2     defaults        1 1
```

12.5.6 配置表决磁盘

1. 使用表决磁盘的必要性

在一个多节点的 RHCS 集群系统中，一个节点失败后，集群的服务和资源可以自动转移到其他节点上。但是这种转移是有条件的，例如，在一个 4 节点的集群中，一旦有两个节点发生故障，整个集群系统将会挂起，集群服务也随之停止。而如果配置了存储集群 GFS 文件系统，那么只要有一个节点发生故障，所有节点挂载的 GFS 文件系统将挂起，此时共享存储将无法使用。这种情况的出现，对高可用的集群系统来说是绝对不允许的。解决这种问题就要通过表决磁盘来实现。

2. 表决磁盘的运行机制

集群系统是基于成员的，它具有民主特征并可以通过投票来决定集群的行为。因此，在一个完全民主的集群环境中，只有超过半数的投票才能决定集群的行为。例如，在一个 6 节点的集群中，如果失败节点超过两个，集群投票数将无法大于集群节点数的一半，那么集群就无法激活。针对这种情况，可采取一种方法：当某个节点失败后，将此节点从集群系统中剔除，由于节点脱离了集群系统，因此也就失去了投票的权利，然后继续由剩余的健康节点进行投票，通过这种方式可以始终保证集群投票数过半，即使集群系统中只存在一个健康节点，整个集群系统也能正常运行。表决磁盘就是为完成此功能而存在的。

表决磁盘，即 Quorum Disk，在 RHCS 里简称 qdisk，是基于磁盘的集群仲裁服务程序。为了解决小规模集群中的投票问题，RHCS 引入了 Quorum 机制，Quorum 表示集群法定的节点数。和 Quorum 对应的是 Quorate，Quorate 是一种状态，表示达到法定节点数。在正常状态下，Quorum 的值是每个节点投票值加上 qdisk 分区的投票值之和。

qdisk 是一个小于 10MB 的共享磁盘分区。qdiskd 进程运行在集群的所有节点上，通过 qdiskd 进程，集群节点定期评估自身的健康情况，并且把自身的状态信息写到指定的共享磁盘分区中。同时 qdiskd 还可以查看其他节点的状态信息，并传递信息给其他节点。例如，在某个节点 A 上，如果 qdiskd 进程经过多次尝试都不能访问自己的共享磁盘分区，那么运行在另一个节点 B 的 qdiskd 进程会请求节点 A 从集群系统中隔离，只有当节点 A 重新启动后才允许其再次加入集群中。

3. RHCS 中表决磁盘的概念

和 qdisk 相关的几个工具有 mkdisk、Heuristics。

mkdisk 是一个集群仲裁磁盘工具集，可以用来创建一个 qdisk 共享磁盘，也可以查看共享磁盘的状态信息。mkqdisk 操作只能创建 16 个节点的投票空间，因此目前 qdisk 最多可以支持 16 个节点的 RHCS 高可用集群。

有时候仅靠检测 qdisk 分区来判断节点状态还是不够的，还可以通过应用程序来扩展对节点状态检测的精度，Heuristics 就是这么一个扩充选项，它允许通过第三方应用程序来辅助定位节点状态，常用的有 ping 网关或路由，或者通过脚本程序等。如果试探失败，qdiskd 会认为此节点失败，进而试图重启此节点，以使节点进入正常状态。

4. 创建一个表决磁盘

在前面内容中，已经划分了多个共享磁盘分区，这里将共享磁盘分区 /dev/sdb7 作为 qdisk 分区。下面创建一个 qdisk 分区：

```
[root@web1 ~]# mkqdisk -c /dev/sdb7 -l myqdisk
mkqdisk v0.6.0
Writing new quorum disk label ' myqdisk ' to /dev/sdb7.
WARNING: About to destroy all data on /dev/sdb7; proceed [N/y] ? y
Initializing status block for node 1...
Initializing status block for node 2...
: : :
Initializing status block for node 16...
[root@web1 ~]#  mkqdisk -L
mkqdisk v0.6.0
/dev/sdb7:
        Magic:                eb7a62c2
        Label:                myqdisk
        Created:              Thu Aug 19 23:27:04 2010
        Host:                 web1
        Kernel Sector Size:   512
        Recorded Sector Size: 512
```

5. 配置 qdisk

这里通过 Conga 的 Web 界面来配置 qdisk。首先登录 Luci，然后单击“Cluster”，在“ClusterList”中单击“mycluster”，接着选择“Use a Quorum Partition”一项，如图 12-2 所示。

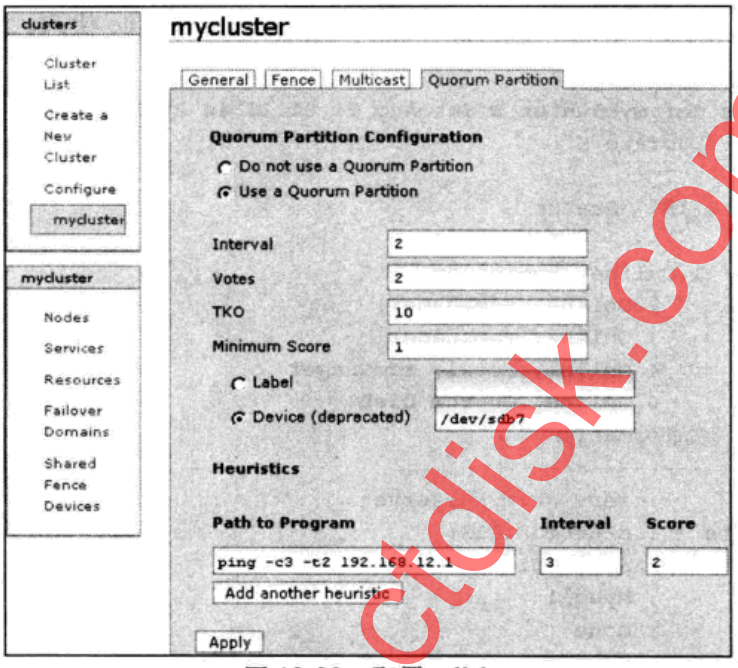

图 12-29　配置 qdisk

图 12-29 中每个选项的含义如下：

- Interval：表示间隔多长时间执行一次检查评估，单位是秒。
- Votes：指定 qdisk 分区投票值是多少。
- TKO：表示允许检查失败的次数。一个节点在 TKO×Interval 时间内如果还连接不上 qdisk 分区，那么就认为此节点失败，会从集群中被隔离。
- Minimum Score：指定最小投票值是多少。
- Label：qdisk 分区对应的卷标名，也就是在创建 qdisk 时指定的“myqdisk”。这里建议用卷标名，因为设备名有可能会在系统重启后发生变化，但卷标名称是不会发生改变的。
- Device：指定共享存储在节点中的设备名是什么。
- Path to Program：配置第三方应用程序来扩展对节点状态检测的精度。这里配置的是 ping 命令。
- Score：设定 ping 命令的投票值。
- Interval：设定多长时间执行 ping 命令一次。

这里将 qdisk 的投票值设定为 2，qdisk 分区设置为共享磁盘 /dev/sdb7，并且通过 ping 网关的方式来扩展对节点状态的检测，单击“Apply”按钮，完成对 qdisk 的设置。

6. 启动 qdiskd 服务

在集群每个节点上执行如下命令，启动 qdiskd 服务：

```
[root@web1 ~]# /etc/init.d/qdiskd  start
```

qdiskd 启动后，如果配置正确，qdisk 磁盘将自动进入 Online 状态。信息如下：

```
[root@web1 ~]# clustat  -l
Cluster Status for mycluster @ Sat Aug 21 01:25:40 2010
Member Status: Quorate

 Member Name     ID   Status
 ------ ----     ---- ------
 Web             1 Online, rgmanager
 Mysql1             2 Online, rgmanager
 Mysql2             3 Online, rgmanager
 web1               4 Online, Local, rgmanager
 /dev/sdb7          0 Online, Quorum Disk
Service         Information
-------         -----------
Service Name       : service:mysqlserver
  Current State    : started (112)
  Flags            : none (0)
  Owner            : Mysql1
  Last Owner       : none
  Last Transition  : Fri Aug 20 21:36:00 2010

Service Name       : service:webserver
  Current State    : started (112)
  Flags            : none (0)
  Owner            : web1
  Last Owner       : none
  Last Transition  : Fri Aug 20 20:31:03 2010
```

至此，qdisk 已经运行起来了。

12.5.7 配置 Fence 设备

配置 Fence 设备是 RHCS 集群系统中必不可少的一个环节，通过 Fence 设备可以防止集群资源（例如文件系统）同时被多个节点占有，保护了共享数据的安全性和一致性，同时也可以防止节点间“脑裂”的发生。

GFS 基于集群底层架构来传递锁信息，或者说是基于 RHCS 的一种集群文件系统，因此使用 GFS 文件系统必须要有 Fence 设备。

RHCS 提供的 Fence 设备有两种。一种是内部 fence 设备，常见的如下：

- IBM 服务器提供的 RSAII 卡。
- HP 服务器提供的 iLO 卡。

- DELL 服务器提供的 DRAC 卡。
- 智能平台管理接口 IPMI。

另一种是外部 Fence 设备，常见的如下：

- UPS。
- SAN SWITCH。
- NETWORK SWITCH。

另外，如果共享存储是通过 GNBD Server 实现的，那么还可以使用 GNBD 的 Fence 功能。

单击“Cluster”，然后单击“ClusterList”中的“mycluster”，在左下角的 mycluster 选择框中依次选择“Shared Fence Devices”→“Add a Fence Device”，如图 12-30 所示。

图 12-30 列出了 RHCS 支持的所有类型的 Fence Device，可以根据自己的 Fence 设备选择对应的 Fence 类型。

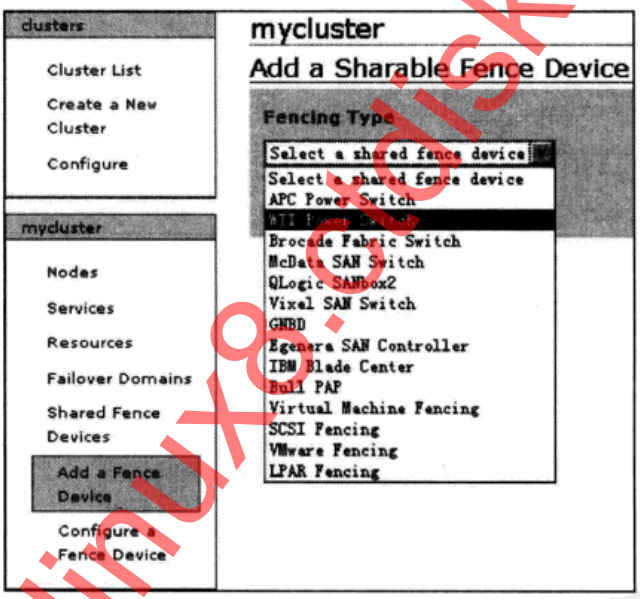

图 12-30 RHCS 提供的 Fence Device 列表

这里选择的 Fence 设备为“WTI Power Switch”，Fence 的名称为“WTI-Fence”。然后依次输入 IP Address 和 Password，如图 12-31 所示。

最后，将添加的 Fence 设备绑定到集群的每个节点上。选择“Cluster”，然后在“Cluster List”中单击“mycluster”中的 web1 节点，如图 12-32 所示。

在“Main Fencing Method”区域中，单击“Add a fence device to this level”，选择一个已经存在的 Fence 设备，这里选择刚才创建好的 Fence 设备，即 WTI-Fence，如图 12-33 所示，最后单击“Update main fence properties”按钮，Fence 设备在节点 web1 上绑定完成。

按照这个操作方法，依次将 Fence 设备绑定到节点 web2、Mysql1、Mysql2 中。

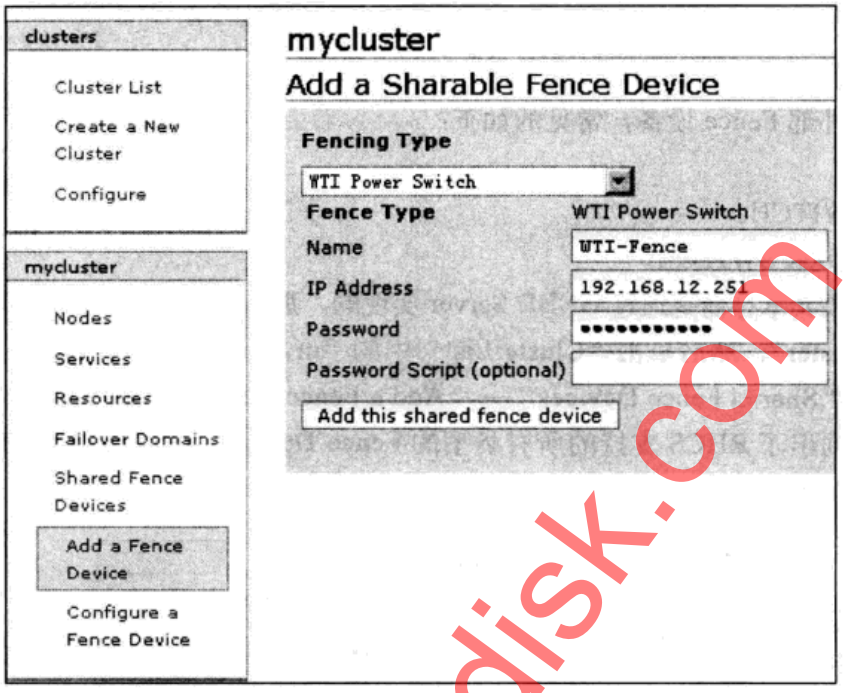

图 12-31 添加一个 Fence 设备

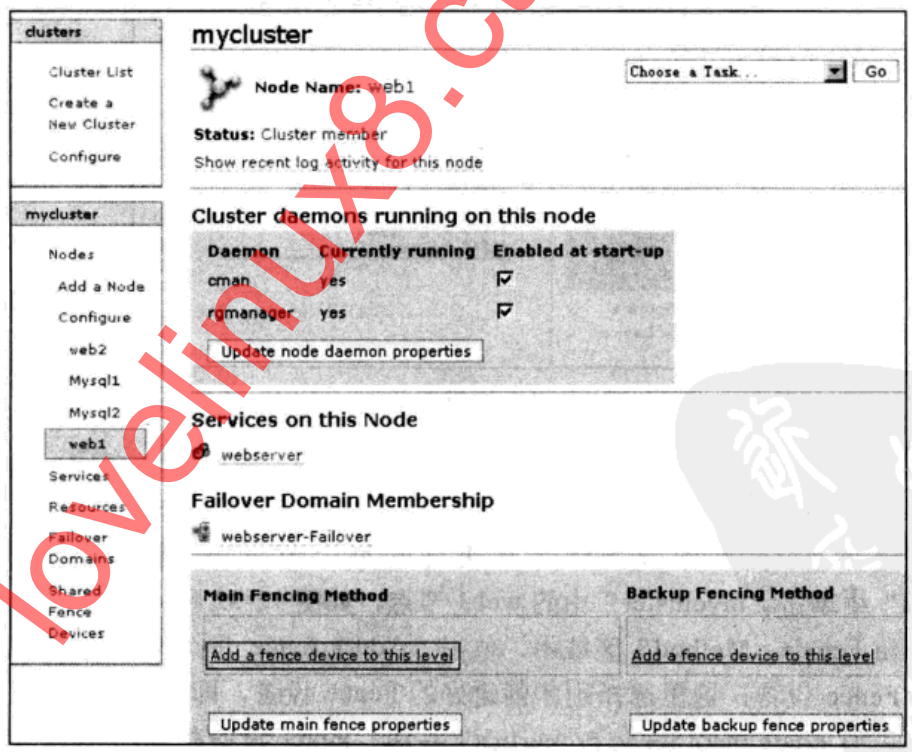

图 12-32 绑定 Fence 设备到每个节点

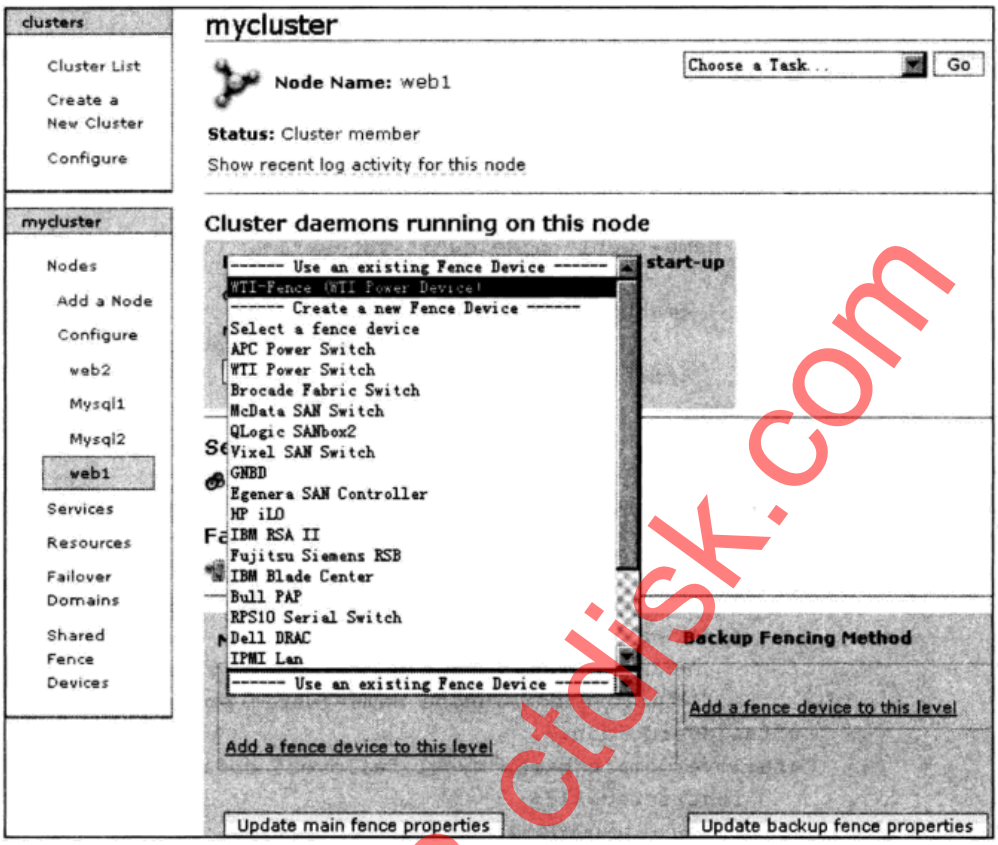

图 12-33　添加 WTI-Fence 到节点 web1 上

到这里为止，一个完整的 RHCS 集群系统已经全部配置完成了。配置完成的 cluster.conf 文件内容如下：

```
<?xml version="1.0"?>
<cluster alias="mycluster" config_version="35" name="mycluster">
    <fence_daemon clean_start="0" post_fail_delay="0" post_join_delay="3"/>
    <clusternodes>
        <clusternode name="web2" nodeid="1" votes="1">
            <fence>
                <method name="1">
                    <device name="WTI-Fence" port="620"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="Mysql1" nodeid="2" votes="1">
            <fence>
                <method name="1">
                    <device name="WTI-Fence" port="620"/>
                </method>
            </fence>
        </clusternode>
```

```
        <clusternode name="Mysql2" nodeid="3" votes="1">
                <fence>
                        <method name="1">
                                <device name="WTI-Fence" port="620"/>
                        </method>
                </fence>
        </clusternode>
        <clusternode name="web1" nodeid="4" votes="1">
                <fence>
                        <method name="1">
                                <device name="WTI-Fence" port="620"/>
                        </method>
                </fence>
        </clusternode>
</clusternodes>
<cman expected_votes="6"/>
<rm>
        <failoverdomains>
                <failoverdomain name="webserver-Failover" nofailback="0"
                    ordered="1" restricted="1">
                        <failoverdomainnode name="web2" priority="10"/>
                        <failoverdomainnode name="web1" priority="1"/>
                </failoverdomain>
                <failoverdomain name="mysql-failover" nofailback="0" ordered="1"
                    restricted="1">
                        <failoverdomainnode name="Mysql1" priority="2"/>
                        <failoverdomainnode name="Mysql2" priority="8"/>
                </failoverdomain>
        </failoverdomains>
        <resources>
                <ip address="192.168.12.233" monitor_link="1"/>
                <script file="/etc/init.d/httpd" name="httpScript"/>
                <ip address="192.168.12.234" monitor_link="1"/>
                <script file="/etc/init.d/mysqld" name="mysqlscript"/>
                <fs device="/dev/sdb6" force_fsck="1" force_unmount="1"
                    fsid="7108" fstype="ext3" mountpoint="/webdata" name="ext3-
                    fs" self_fence="1"/>
        </resources>
        <service autostart="1" domain="webserver-Failover" exclusive="0"
            name="webserver" recovery="relocate">
                <ip ref="192.168.12.233"/>
                <script ref="httpScript"/>
        </service>
        <service autostart="1" domain="mysql-failover" exclusive="0"
            name="mysqlserver" recovery="relocate">
                <ip ref="192.168.12.234"/>
                <script ref="mysqlscript"/>
        </service>
</rm>
<fencedevices>
```

```
        <fencedevice agent="fence_wti" ipaddr="192.168.12.251" name="WTI-Fence"
            passwd="fencepasswd"/>
    </fencedevices>
    <quorumd device="/dev/sdb7" interval="2" min_score="1" tko="10" votes="2">
        <heuristic interval="3" program="ping -c3 -t2 192.168.12.1" score="2"/>
    </quorumd>
</cluster>
```

12.6　管理和维护 RHCS 集群

管理和维护 RHCS 集群是一个非常复杂和繁琐的工作，要维护好一个 RHCS 集群，必须熟悉 RHCS 的基本运行原理，所以前面花费了大量章节讲述 RHCS 集群的构成和运行原理。在集群管理方面，RHCS 提供了两种方式：即 Luci 图形界面方式和命令行方式。图形方式的使用非常简单，这里不再详述，这里重点讲述在命令行下如何管理 RHCS 集群。

12.6.1　启动 RHCS 集群

RHCS 集群的核心进程有 cman 和 rgmanager。要启动集群，依次在集群的每个节点上执行如下命令即可：

```
service cman start
service rgmanager start
```

需要注意的是，执行这两个命令是有先后顺序的，要先启动 cman，然后再启动 rgmanager。在集群所有节点成功启动 cman 服务后，继续依次在每个节点启动 rgmanager 服务。

例如，在主机 web1 上启动集群服务，操作如下：

```
[root@web1 ~]# service cman start
Starting cluster:
   Loading modules... done
   Mounting configfs... done
   Starting ccsd... done
   Starting cman... done
   Starting qdiskd... done
   Starting daemons... done
   Starting fencing... done
[  OK  ]
```

等 cman 在其他节点成功启动后，开始启动 rgmanager 服务。

```
[root@web1 ~]# service rgmanager start
Starting Cluster Service Manager: [  OK  ]
```

从这个输出可知，cman 其实是一个集群服务的集合，在启动 cman 时，同时也启动了 ccsd、qdisk、daemons、fenceing 等服务，并且将相关集群模块载入系统内核。

12.6.2 关闭 RHCS 集群

与启动集群服务刚好相反，关闭 RHCS 集群的命令如下：

```
service rgmanager stop
service cman stop
```

首先在集群的每个节点上依次关闭 rgmanager 服务，待所有节点的 rgmanager 服务成功关闭后，再依次关闭每个节点上的 cman 服务即可完成整个集群服务的关闭。

例如，关闭 Mysql2 主机上的集群服务，执行如下操作：

```
[root@Mysql2 ~]# /etc/init.d/rgmanager  stop
Shutting down Cluster Service Manager...
Waiting for services to stop: [  OK  ]
[root@Mysql2 ~]# /etc/init.d/cman  stop
Stopping cluster:
   Stopping fencing... done
   Stopping cman... done
   Stopping ccsd... done
   Unmounting configfs... done
[  OK  ]
```

有时在关闭 cman 服务时，可能会提示关闭失败，此时可以检查本机的共享存储 GFS2 文件系统是否已经卸载，也可以检查其他节点的 rgmanager 服务是否都已经正常关闭。

12.6.3 管理应用服务

集群系统启动后，默认自动启动应用服务，但是，如果某个应用服务没有自动启动，就需要通过手工方式来启动。管理应用服务的命令是 clusvcadm，通过这个命令可以启动、关闭、重启、切换集群中的应用服务。

1. 启动某个应用服务

可以通过如下方式启动某个节点上的应用服务：

```
clusvcadm -e <Service> -m <Node>
```

其中：

- Service，表示集群中创建的应用服务名称。
- Node，表示集群节点名称。

例如，要启动节点 web1 上的 webserver 服务，操作如下：

```
[root@web1 ~]# clusvcadm -e webserver -m web1
Member web1 trying to enable service:webserver...Success
service:webserver is now running on web1
```

可以通过 /var/log/messages 文件查看启动应用服务的详细信息。当 webserver 启动后，与服务相关的集群资源，如虚拟 IP、应用程序服务脚本也随之启动，可以通过如下命令查看

集群资源是否已经正常加载。

```
[root@web1 ~]# ip addr show |grep eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    inet 192.168.12.230/24 brd 192.168.12.255 scope global eth0
    inet 192.168.12.233/24 scope global secondary eth0
[root@web1 ~]# ps -ef |grep httpd
root      22869     1  0 Aug22 ?        00:00:00 /usr/sbin/httpd
apache    22870 22869  0 Aug22 ?        00:00:00 /usr/sbin/httpd
apache    22871 22869  0 Aug22 ?        00:00:00 /usr/sbin/httpd
apache    22872 22869  0 Aug22 ?        00:00:00 /usr/sbin/httpd
apache    22873 22869  0 Aug22 ?        00:00:00 /usr/sbin/httpd
apache    22874 22869  0 Aug22 ?        00:00:00 /usr/sbin/httpd
apache    22876 22869  0 Aug22 ?        00:00:00 /usr/sbin/httpd
apache    22877 22869  0 Aug22 ?        00:00:00 /usr/sbin/httpd
apache    22878 22869  0 Aug22 ?        00:00:00 /usr/sbin/httpd
root      26069  3428  0 00:08 pts/1    00:00:00 grep httpd
```

从输出可知，虚拟 IP 地址 192.168.12.233 已经加载，httpd 服务也自动启动。

2. 关闭某个应用服务

可以通过如下方式关闭某个节点的应用服务：

```
clusvcadm -s <Service> -m <Node>
```

例如，要关闭节点 Mysql1 上的 mysqlserver 服务，操作如下：

```
[root@Mysql1 ~]# clusvcadm -s mysqlserver -m Mysql1
Member Mysql1 stopping service:mysqlserver...Success
```

可以通过 /var/log/messages 文件查看关闭应用服务的详细信息。当 mysqlserver 关闭后，与服务相关的集群资源，如虚拟 IP、应用程序服务脚本也随之释放。

3. 重启某个应用服务

可以通过如下方式重启某个节点上的应用服务：

```
clusvcadm -R <Service> -m <Node>
```

例如，要重启节点 web1 上的 webserver 服务，操作如下：

```
[root@web2 ~]# clusvcadm -R webserver -m web1
Member web1 trying to restart service:webserver...Success
```

这个命令是在 web2 节点上执行的，也能将 web1 节点上的 webserver 进行重启。由此可知，clusvcadm 命令在集群任意节点上执行都是可以的。

4. 切换某个服务

可以通过如下方式将一个应用服务从一个节点切换到另一个节点：

```
clusvcadm -r <Service> -m <Node>
```

例如，要将节点 web1 的服务切换到节点 web2 上，操作如下：

```
[root@web1 ~]# clusvcadm -r webserver -m web2
Trying to relocate service:webserver to web2...Success
service:webserver is now running on web2
```

12.6.4 监控 RHCS 集群状态

通过对 RHCS 的监控，有助于了解集群每个节点的健康状况，发现问题并及时解决问题。RHCS 集群提供了丰富的状态查看命令，这里主要介绍 cman_tool、clustat、ccs_tool 的使用方法。

1. cman_tool 命令

cman_tool 的参数比较多，但是用法比较简单，基本语法格式如下：

```
cman_tool <join|leave|kill|expected|votes|version|wait|status|nodes|services|debug>
    [options]
```

下面列举几个简单的使用例子。

```
[root@web1 ~]# cman_tool  nodes -a
Node  Sts   Inc   Joined               Name
   0   M      0   2010-08-23 01:24:00  /dev/sdb7
   1   M   2492   2010-08-23 01:22:43  web2
       Addresses: 192.168.12.240
   2   M   2492   2010-08-23 01:22:43  Mysql1
       Addresses: 192.168.12.231
   3   M   2492   2010-08-23 01:22:43  Mysql2
       Addresses: 192.168.12.232
   4   M   2488   2010-08-23 01:22:43  web1
       Addresses: 192.168.12.230
```

此命令显示了节点名称、对应的节点 IP 地址和加入集群的时间。

要了解更多集群节点信息，可以通过如下命令：

```
[root@web1 ~]# cman_tool   status
Version: 6.2.0
Config Version: 35                    # 集群配置文件版本号
Cluster Name: mycluster               # 集群名称
Cluster Id: 56756
Cluster Member: Yes
Cluster Generation: 2764
Membership state: Cluster-Member
Nodes: 4                              # 集群节点数
Expected votes: 6                     # 期望的投票数
Quorum device votes: 2                # 表决磁盘投票值
Total votes: 6                        # 集群中所有投票值大小
Quorum: 4                             # 集群法定投票值，低于这个值，集群将停止服务
Active subsystems: 9
Flags: Dirty
Ports Bound: 0 177
```

```
Node name: web1
Node ID: 4                                          # 本节点在集群中的 ID 号
Multicast addresses: 239.192.221.146                # 集群广播地址
Node addresses: 192.168.12.230                      # 本节点对应的 IP 地址
```

2. clustat 命令

clustat 命令的使用非常简单，详细的使用方法可以通过“clustat -h”获取帮助信息，这里仅列举几个例子。

```
[root@web1 ~]#clustat  -i 3
Cluster Status for mycluster @ Mon Aug 23 18:54:15 2010
Member Status: Quorate
Member Name                ID              Status
 ------ ----               ----            ------
 web2                      1               Online, rgmanager
 Mysql1                    2               Online, rgmanager
 Mysql2                    3               Online, rgmanager
 web1                      4               Online, Local, rgmanager
 /dev/sdb7                 0               Online, Quorum Disk

 Service Name              Owner (Last)              State
 ------- ----              ----- ------              -------
 service:mysqlserver       Mysql1                    started
 service:webserver         web1                      started
```

clustat 的“-i”参数可以实时显示集群系统中每个节点及服务的运行状态，“-i 3”表示每 3 秒刷新一次集群状态。

在这个输出中可以看到，每个节点都处于“Online”状态，表明每个节点都运行正常。如果某个节点退出了集群，对应的状态应该是“Offline”。同时还可以看到，集群的两个服务也处于“started”状态，分别运行在 Mysql1 节点和 web1 节点上。

另外，通过“ID”一列可以知道集群节点的对应关系，例如，web2 在此集群中对应的是“Node 1”节点，同理，web1 对应的是“Node 4”节点。了解集群节点顺序有助于对集群日志的解读。

3. ccs_tool 命令

ccs_tool 主要用来管理集群配置文件 cluster.conf，通过 ccs_tool 可以在集群中进行增加 / 删除节点、增加 / 删除 Fence 设备、更新集群配置文件等操作。

下面是 ccs_tool 的几个应用实例。

在一个节点中修改完配置文件后，可以执行“ccs_tool update”指令将配置文件在所有节点中进行更新。例如：

```
[root@web1 cluster]# ccs_tool  update /etc/cluster/cluster.conf
Proposed updated config file does not have greater version number.
  Current config_version :: 35
  Proposed config_version:: 35
```

```
Failed to update config file.
```

ccs_tool 是根据 cluster.conf 中的“config_version”值来决定是否进行更新的，因此在修改完 cluster.conf 文件后，一定要将 cluster.conf 的 config_version 值进行更新，这样执行 ccs_tool 时才能更新配置文件。例如：

```
[root@web1 cluster]# ccs_tool  update /etc/cluster/cluster.conf
Config file updated from version 35 to 36

Update complete.
```

此外，通过 ccs_tool 也可以创建一个集群配置文件。相关命令如下：

```
ccs_tool create MyCluster
ccs_tool addfence apc fence_apc ipaddr=apc.domain.net user=apc password=apc
ccs_tool addnode node1 -n 1 -f apc port=1
ccs_tool addnode node2 -n 2 -f apc port=2
ccs_tool addnode node3 -n 3 -f apc port=3
ccs_tool addnode node4 -n 4 -f apc port=4
```

12.6.5 管理和维护 GFS2 文件系统

GFS2 文件系统提供了很多管理和维护工具，常用的有 gfs2_fsck、gfs2_tool、gfs2_jadd、gfs2_quota 和 gfs2_convert 等。这里重点介绍前三个命令的用法。

1. gfs2_fsck 命令

类似于 ext3 文件系统下的 fsck.ext3 命令，主要用于检测和修复文件系统错误。其实 GFS2 还有一个 fsck.gfs2 命令，此命令与 gfs2_fsck 命令完全一致。

gfs2_fsck 的用法如下：

```
gfs2_fsck [-afhnpqvVy] <device>
```

下面列举几个使用的例子：

```
[root@Mysql1 ~]# gfs2_fsck  -y /dev/sdb5
Initializing fsck
Validating Resource Group index.
Level 1 RG check.
(level 1 passed)
Starting pass1
Pass1 complete
Starting pass1b
Pass1b complete
Starting pass1c
Pass1c complete
.....
Pass5 complete
gfs2_fsck complete
```

2. gfs2_tool 命令

gfs2_tool 命令的参数较多，但使用并不复杂，该命令主要用来查看、修改 GFS2 文件系统的相关参数信息。

下面列举几个使用的例子。

（1）查看 GFS2 文件系统挂载信息

```
[root@web1 ~]# gfs2_tool df  /gfs2
/gfs2:
  SB lock proto = "lock_dlm"
  SB lock table = "mycluster:my-gfs2"
  SB ondisk format = 1801
  SB multihost format = 1900
  Block size = 4096
  Journals = 4
  Resource Groups = 19
  Mounted lock proto = "lock_dlm"
  Mounted lock table = "mycluster:my-gfs2"
  Mounted host data = "jid=2:id=65539:first=0"
  Journal number = 2
  Lock module flags = 0
  Local flocks = FALSE
  Local caching = FALSE

  Type            Total Blocks   Used Blocks    Free Blocks    use%
  --------        ------------   -----------    -----------    -----
  data            1220724        136578         1084146        11%
  inodes          1084263        117            1084146        0%
```

从这个输出可以知道 GFS2 文件系统中的锁类型、锁标识、数据块大小、可挂载节点数等信息。需要注意的是，“gfs2_tool df”后面跟的是 GFS2 文件系统对应的挂载点，也就是说，GFS2 文件系统必须处于挂载状态下才能查看其相关信息。

（2）锁定与解锁 GFS2 文件系统

```
[root@node1 gfs2]# gfs2_tool freeze  /gfs2
[root@node1 gfs2]# gfs2_tool unfreeze  /gfs2
```

GFS2 文件系统被锁定后，无法进行读写操作，直到被解锁。

（3）查询 GFS2 可挂载的节点数

```
[root@web1 ~]# gfs2_tool journals /gfs2
journal2 - 128MB
journal3 - 128MB
journal1 - 128MB
journal0 - 128MB
4 journal(s) found.
```

这里显示了可挂载节点数为 4，并且每个 journal 的大小为 128MB。

（4）显示 GFS2 的版本信息

```
[root@web1 ~]# gfs2_tool  version
gfs2_tool 0.1.62 (built Mar 31 2010 07:34:25)
Copyright (C) Red Hat, Inc.  2004-2006  All rights reserved.
```

3.gfs2-jadd 命令

gfs2-jadd 主要用来配置 GFS2 的 Journals 数量和大小，用法非常简单。

```
gfs2_jadd [-cDhJjqV] /path/to/filesystem
```

下面列举几个例子。

设置 Journals 的大小为 64MB。

```
[root@Mysql1 ~]# gfs2_jadd -J 64M
```

将 GFS2 可同时挂载的节点数目增加到 5 个。

```
[root@Mysql1 ~]# gfs2_jadd  -j 5 /gfs2
```

另外，gfs2_quota 可用于 GFS2 文件系统磁盘配额管理。gfs2_convert 是一个数据转换应用程序，它可以对 GFS 文件系统的元数据进行更新，把它转换为一个 GFS2 文件系统。

12.7 RHCS 集群功能测试

完成集群配置后，如何知道集群已经配置成功了呢？下面就分情况测试 RHCS 提供的高可用集群和存储集群功能。

12.7.1 高可用集群测试

前面配置了 4 个节点的集群系统，每个节点的主机名分别是 web1、web2、Mysql1、Mysql2。4 个节点之间的关系是：web1 和 web2 组成 Web 集群，运行 webserver 服务，其中 web1 是主节点，在正常状态下，服务运行在此节点，web2 是备用节点；Mysql1 和 Mysql2 组成 MySQL 集群，运行 mysqlserver 服务，其中，Mysql1 是主节点，在正常状态下，服务运行在此节点，Mysql2 为备用节点。

下面分 4 种情况来介绍当节点发生宕机时，集群是如何进行切换和工作的。

1. 节点 web2 宕机时

宕机分为正常关机和异常宕机两种情况，下面分别说明。

（1）节点 web2 正常关机

在节点 web2 上执行正常关机命令。

```
[root@web2 ~]#init 0
```

然后在 web1 节点上查看 /var/log/messages 日志。输出信息如下：

```
Aug 24 00:57:09 web1 clurgmgrd[3321]: <notice> Member 1 shutting down
```

```
Aug 24 00:57:17 web1 qdiskd[2778]: <info> Node 1 shutdown
Aug 24 00:57:29 web1 openais[2755]: [TOTEM] The token was lost in the OPERATIONAL
   state.
Aug 24 00:57:29 web1 openais[2755]: [TOTEM] Receive multicast socket recv buffer
   size (320000 bytes).
Aug 24 00:57:29 web1 openais[2755]: [TOTEM] Transmit multicast socket send buffer
   size (219136 bytes).
Aug 24 00:57:29 web1 openais[2755]: [TOTEM] entering GATHER state from 2.
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] entering GATHER state from 0.
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] Creating commit token because I am the
   rep.
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] Saving state aru 73 high seq received 73
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] Storing new sequence id for ring bc8
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] entering COMMIT state.
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] entering RECOVERY state.
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] position [0] member 192.168.12.230:
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] previous ring seq 3012 rep 192.168.12.230
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] aru 73 high delivered 73 received flag 1
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] position [1] member 192.168.12.231:
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] previous ring seq 3012 rep 192.168.12.230
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] aru 73 high delivered 73 received flag 1
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] position [2] member 192.168.12.232:
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] previous ring seq 3012 rep 192.168.12.230
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] aru 73 high delivered 73 received flag 1
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] Did not need to originate any messages
   in recovery.
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] Sending initial ORF token
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] New Configuration:
Aug 24 00:57:49 web1 openais[2755]: [CLM  ]     r(0) ip(192.168.12.230)
Aug 24 00:57:49 web1 openais[2755]: [CLM  ]     r(0) ip(192.168.12.231)
Aug 24 00:57:49 web1 openais[2755]: [CLM  ]     r(0) ip(192.168.12.232)
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] Members Left:
Aug 24 00:57:49 web1 openais[2755]: [CLM  ]     r(0) ip(192.168.12.240)
Aug 24 00:57:49 web1 kernel: dlm: closing connection to node 1
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] Members Joined:
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] New Configuration:
Aug 24 00:57:49 web1 openais[2755]: [CLM  ]     r(0) ip(192.168.12.230)
Aug 24 00:57:49 web1 openais[2755]: [CLM  ]     r(0) ip(192.168.12.231)
Aug 24 00:57:49 web1 openais[2755]: [CLM  ]     r(0) ip(192.168.12.232)
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] Members Left:
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] Members Joined:
Aug 24 00:57:49 web1 openais[2755]: [SYNC ] This node is within the primary
   component and will provide service.
Aug 24 00:57:49 web1 openais[2755]: [TOTEM] entering OPERATIONAL state.
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] got nodejoin message 192.168.12.230
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] got nodejoin message 192.168.12.231
Aug 24 00:57:49 web1 openais[2755]: [CLM  ] got nodejoin message 192.168.12.232
Aug 24 00:57:49 web1 openais[2755]: [CPG  ] got joinlist message from node 3
```

```
Aug 24 00:57:49 web1 openais[2755]: [CPG   ] got joinlist message from node 4
Aug 24 00:57:49 web1 openais[2755]: [CPG   ] got joinlist message from node 2
```

从输出日志可以看出，当 web2 节点正常关机后，qdiskd 进程立刻检测到 web2 节点已经关闭，然后 dlm 锁进程正常关闭了从 web2 的连接。由于是正常关闭节点 web2，所以 RHCS 认为整个集群系统没有发生异常，仅把节点 web2 从集群中隔离而已（重点看日志中斜体部分）。

重新启动 web2 节点，然后在 web1 上继续观察日志信息。

```
Aug 24 01:10:50 web1 openais[2755]: [TOTEM] entering GATHER state from 11.
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] Creating commit token because I am the
   rep.
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] Saving state aru 2b high seq received 2b
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] Storing new sequence id for ring bcc
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] entering COMMIT state.
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] entering RECOVERY state.
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] position [0] member 192.168.12.230:
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] previous ring seq 3016 rep 192.168.12.230
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] aru 2b high delivered 2b received flag 1
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] position [1] member 192.168.12.231:
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] previous ring seq 3016 rep 192.168.12.230
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] aru 2b high delivered 2b received flag 1
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] position [2] member 192.168.12.232:
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] previous ring seq 3016 rep 192.168.12.230
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] aru 2b high delivered 2b received flag 1
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] position [3] member 192.168.12.240:
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] previous ring seq 3016 rep 192.168.12.240
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] aru c high delivered c received flag 1
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] Did not need to originate any messages
   in recovery.
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] Sending initial ORF token
Aug 24 01:10:51 web1 openais[2755]: [CLM   ] CLM CONFIGURATION CHANGE
Aug 24 01:10:51 web1 openais[2755]: [CLM   ] New Configuration:
Aug 24 01:10:51 web1 openais[2755]: [CLM   ]      r(0) ip(192.168.12.230)
Aug 24 01:10:51 web1 openais[2755]: [CLM   ]      r(0) ip(192.168.12.231)
Aug 24 01:10:51 web1 openais[2755]: [CLM   ]      r(0) ip(192.168.12.232)
Aug 24 01:10:51 web1 openais[2755]: [CLM   ] Members Left:
Aug 24 01:10:51 web1 openais[2755]: [CLM   ] Members Joined:
Aug 24 01:10:51 web1 openais[2755]: [CLM   ] CLM CONFIGURATION CHANGE
Aug 24 01:10:51 web1 openais[2755]: [CLM   ] New Configuration:
Aug 24 01:10:51 web1 openais[2755]: [CLM   ]      r(0) ip(192.168.12.230)
Aug 24 01:10:51 web1 openais[2755]: [CLM   ]      r(0) ip(192.168.12.231)
Aug 24 01:10:51 web1 openais[2755]: [CLM   ]      r(0) ip(192.168.12.232)
Aug 24 01:10:51 web1 openais[2755]: [CLM   ]      r(0) ip(192.168.12.240)
Aug 24 01:10:51 web1 openais[2755]: [CLM   ] Members Left:
Aug 24 01:10:51 web1 openais[2755]: [CLM   ] Members Joined:
Aug 24 01:10:51 web1 openais[2755]: [CLM   ]      r(0) ip(192.168.12.240)
Aug 24 01:10:51 web1 openais[2755]: [SYNC ] This node is within the primary component
   and will provide service.
Aug 24 01:10:51 web1 openais[2755]: [TOTEM] entering OPERATIONAL state.
```

```
Aug 24 01:10:51 web1 openais[2755]: [CLM  ] got nodejoin message 192.168.12.230
Aug 24 01:10:51 web1 openais[2755]: [CLM  ] got nodejoin message 192.168.12.231
Aug 24 01:10:51 web1 openais[2755]: [CLM  ] got nodejoin message 192.168.12.232
Aug 24 01:10:51 web1 openais[2755]: [CLM  ] got nodejoin message 192.168.12.240
Aug 24 01:10:51 web1 openais[2755]: [CPG  ] got joinlist message from node 3
Aug 24 01:10:51 web1 openais[2755]: [CPG  ] got joinlist message from node 4
Aug 24 01:10:51 web1 openais[2755]: [CPG  ] got joinlist message from node 2
Aug 24 01:10:55 web1 kernel: dlm: connecting to 1
```

从输出可知，重新启动节点 web2 后，openais 底层通信进程检测到 web2 节点已经激活，并且将 web2 重新加入集群中（重点看日志中斜体部分）。

（2）节点 web2 异常宕机

在节点 web2 上执行如下命令，使内核崩溃：

```
[root@web2 ~]#echo c>/proc/sysrq-trigger
```

然后在节点 Mysql1 上查看 /var/log/messages 日志。信息如下：

```
Aug 24 02:26:16 Mysql1 openais[2649]: [TOTEM] entering GATHER state from 12.
Aug 24 02:26:28 Mysql1 qdiskd[2672]: <notice> Node 1 evicted
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] entering GATHER state from 11.
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] Saving state aru 78 high seq received 78
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] Storing new sequence id for ring bd0
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] entering COMMIT state.
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] entering RECOVERY state.
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] position [0] member 192.168.12.230:
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] previous ring seq 3020 rep 192.168.12.230
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] aru 78 high delivered 78 received flag 1
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] position [1] member 192.168.12.231:
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] previous ring seq 3020 rep 192.168.12.230
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] aru 78 high delivered 78 received flag 1
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] position [2] member 192.168.12.232:
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] previous ring seq 3020 rep 192.168.12.230
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] aru 78 high delivered 78 received flag 1
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] Did not need to originate any messages in
   recovery.
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] New Configuration:
Aug 24 02:26:36 Mysql1 kernel: dlm: closing connection to node 1
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ]   r(0) ip(192.168.12.230)
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ]   r(0) ip(192.168.12.231)
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ]   r(0) ip(192.168.12.232)
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] Members Left:
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ]   r(0) ip(192.168.12.240)
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] Members Joined:
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] New Configuration:
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ]   r(0) ip(192.168.12.230)
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ]   r(0) ip(192.168.12.231)
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ]   r(0) ip(192.168.12.232)
```

```
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] Members Left:
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] Members Joined:
Aug 24 02:26:36 Mysql1 openais[2649]: [SYNC ] This node is within the primary component
   and will provide service.
Aug 24 02:26:36 Mysql1 fenced[2688]: web2 not a cluster member after 0 sec post_fail_
   delay
Aug 24 02:26:36 Mysql1 openais[2649]: [TOTEM] entering OPERATIONAL state.
Aug 24 02:26:36 Mysql1 fenced[2688]: fencing node "web2"
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] got nodejoin message 192.168.12.230
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] got nodejoin message 192.168.12.231
Aug 24 02:26:36 Mysql1 openais[2649]: [CLM  ] got nodejoin message 192.168.12.232
Aug 24 02:26:36 Mysql1 openais[2649]: [CPG  ] got joinlist message from node 3
Aug 24 02:26:36 Mysql1 openais[2649]: [CPG  ] got joinlist message from node 4
Aug 24 02:26:36 Mysql1 openais[2649]: [CPG  ] got joinlist message from node 2
Aug 24 02:26:45 Mysql1 fenced[2688]: fence "web2" success
Aug 24 02:26:45 Mysql1 kernel: GFS2: fsid=mycluster:my-gfs2.2: jid=3: Trying to acquire
   journal lock...
Aug 24 02:26:45 Mysql1 kernel: GFS2: fsid=mycluster:my-gfs2.2: jid=3: Looking at
   journal...
Aug 24 02:26:45 Mysql1 kernel: GFS2: fsid=mycluster:my-gfs2.2: jid=3: Done
```

从输出信息可知，qdiskd 首先检测到 web2 出现异常，然后将它从集群中隔离。由于是异常宕机，所以 RHCS 为了保证集群资源的唯一性，必须重置 web2 节点，于是 fenced 进程启动。 在 fenced 进程没有返回成功信息之前，所有节点挂载的 GFS2 共享分区将无法使用，处于挂起的状态，直到 fenced 进程成功。

此时，在节点 web1 上查看 web2 的集群状态如下：

```
[root@web1 ~]#clustat  -m web2
 Member Name        ID      Status
 ------ ----        ----    ------
 web2               1       Offline
```

由输出可知，web2 已经处于 Offline 状态了。

2. 节点 Mysql2 宕机时

节点 Mysql2 在正常关机和异常宕机时，RHCS 的切换状态与上面讲述的 web2 节点情况一模一样，这里不再重复讲述。

3. 节点 web1 宕机时

（1）节点 web1 正常关机

在节点 web1 上执行正常关机命令。

```
[root@web1 ~]# init 0
```

然后在节点 web2 上查看 /var/log/messages 日志。信息如下：

```
Aug 24 02:06:13 web2 last message repeated 3 times
Aug 24 02:14:58 web2 clurgmgrd[3239]: <notice> Member 4 shutting down
Aug 24 02:15:03 web2 clurgmgrd[3239]: <notice> Starting stopped service
```

```
   service:webserver
Aug 24 02:15:05 web2 avahi-daemon[3110]: Registering new address record for
   192.168.12.233 on eth0.
Aug 24 02:15:06 web2 in.rdiscd[4451]: setsockopt (IP_ADD_MEMBERSHIP): Address
   already in use
Aug 24 02:15:06 web2 in.rdiscd[4451]: Failed joining addresses
Aug 24 02:15:07 web2 clurgmgrd[3239]: <notice> Service service:webserver started
Aug 24 02:15:08 web2 qdiskd[2712]: <info> Node 4 shutdown
Aug 24 02:15:21 web2 openais[2689]: [TOTEM] entering GATHER state from 12.
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] entering GATHER state from 11.
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] Saving state aru b7 high seq received b7
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] Storing new sequence id for ring bd8
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] entering COMMIT state.
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] entering RECOVERY state.
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] position [0] member 192.168.12.231:
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] previous ring seq 3028 rep 192.168.12.230
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] aru b7 high delivered b7 received flag 1
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] position [1] member 192.168.12.232:
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] previous ring seq 3028 rep 192.168.12.230
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] aru b7 high delivered b7 received flag 1
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] position [2] member 192.168.12.240:
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] previous ring seq 3028 rep 192.168.12.230
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] aru b7 high delivered b7 received flag 1
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] Did not need to originate any messages
   in recovery.
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] New Configuration:
Aug 24 02:15:41 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.231)
Aug 24 02:15:41 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.232)
Aug 24 02:15:41 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.240)
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] Members Left:
Aug 24 02:15:41 web2 kernel: dlm: closing connection to node 4
Aug 24 02:15:41 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.230)
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] Members Joined:
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] New Configuration:
Aug 24 02:15:41 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.231)
Aug 24 02:15:41 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.232)
Aug 24 02:15:41 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.240)
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] Members Left:
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] Members Joined:
Aug 24 02:15:41 web2 openais[2689]: [SYNC ] This node is within the primary component
   and will provide service.
Aug 24 02:15:41 web2 openais[2689]: [TOTEM] entering OPERATIONAL state.
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.231
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.232
Aug 24 02:15:41 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.240
Aug 24 02:15:41 web2 openais[2689]: [CPG  ] got joinlist message from node 2
Aug 24 02:15:41 web2 openais[2689]: [CPG  ] got joinlist message from node 3
Aug 24 02:15:41 web2 openais[2689]: [CPG  ] got joinlist message from node 1
```

从输出日志可知，节点 web1 正常关机后，节点 web1 的服务和 IP 资源自动切换到了节点 web2 上，然后由 qdiskd 进程将节点 web1 从集群系统中隔离。由于 web1 节点是正常关闭的，所以集群中 GFS2 共享文件系统可以正常读写，不受 web1 关闭的影响。

此时，在 web2 上查看节点 web1 的状态如下：

```
[root@web2 ~]# clustat    -m web1
 Member Name                ID      Status
 ------ ----                ----    ------
 web1                       4       Offline
```

从输出可知，web1 节点已经处于 Offline 状态了。

接着，登录到节点 web2，查看集群服务和 IP 资源是否正常切换。操作如下：

```
 [root@web2 ~]# clustat  -s webserver
 Service Name      Owner (Last)          State
------- ----            ----- ------    -----
service:webserver    web2              started
[root@web2 ~]# ip addr show|grep eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    inet 192.168.12.240/24 brd 192.168.12.255 scope global eth0
    inet 192.168.12.233/24 scope global secondary eth0
```

从输出可知，集群服务和 IP 地址已经成功切换到 web2 节点上。

最后，重新启动节点 web1，然后在节点 web2 上查看 /var/log/messages 日志。信息如下：

```
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] entering GATHER state from 11.
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] Saving state aru 2b high seq received 2b
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] Storing new sequence id for ring bdc
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] entering COMMIT state.
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] entering RECOVERY state.
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] position [0] member 192.168.12.230:
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] previous ring seq 3028 rep 192.168.12.230
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] aru 0 high delivered 0 received flag 1
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] position [1] member 192.168.12.231:
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] previous ring seq 3032 rep 192.168.12.231
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] aru 2b high delivered 2b received flag 1
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] position [2] member 192.168.12.232:
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] previous ring seq 3032 rep 192.168.12.231
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] aru 2b high delivered 2b received flag 1
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] position [3] member 192.168.12.240:
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] previous ring seq 3032 rep 192.168.12.231
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] aru 2b high delivered 2b received flag 1
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] Did not need to originate any messages
   in recovery.
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] New Configuration:
Aug 24 02:42:36 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.231)
Aug 24 02:42:36 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.232)
Aug 24 02:42:36 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.240)
```

```
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] Members Left:
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] Members Joined:
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] New Configuration:
Aug 24 02:42:36 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.230)
Aug 24 02:42:36 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.231)
Aug 24 02:42:36 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.232)
Aug 24 02:42:36 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.240)
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] Members Left:
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] Members Joined:
Aug 24 02:42:36 web2 openais[2689]: [CLM  ]     r(0) ip(192.168.12.230)
Aug 24 02:42:36 web2 openais[2689]: [SYNC ] This node is within the primary component
   and will provide service.
Aug 24 02:42:36 web2 openais[2689]: [TOTEM] entering OPERATIONAL state.
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.230
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.231
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.232
Aug 24 02:42:36 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.240
Aug 24 02:42:36 web2 openais[2689]: [CPG  ] got joinlist message from node 3
Aug 24 02:42:36 web2 openais[2689]: [CPG  ] got joinlist message from node 1
Aug 24 02:42:36 web2 openais[2689]: [CPG  ] got joinlist message from node 2
Aug 24 02:42:40 web2 kernel: dlm: got connection from 4
Aug 24 02:43:06 web2 clurgmgrd[3239]: <notice> Relocating service:webserver to
   better node web1
Aug 24 02:43:06 web2 clurgmgrd[3239]: <notice> Stopping service service:webserver
Aug 24 02:43:07 web2 avahi-daemon[3110]: Withdrawing address record for
   192.168.12.233 on eth0.
Aug 24 02:43:17 web2 clurgmgrd[3239]: <notice> Service service:webserver is stopped
```

从输出可知，节点 web1 在重新启动后，再次被加入到集群系统中，同时停止自身的服务并释放 IP 资源。这个切换方式跟集群设置的 Failover Domain 策略有关，在创建的失败转移域 webserver-Failover 中，没有加入“Do not fail back services in this domain”一项功能，也就是主节点在重新启动后，自动将服务切换回来。

此时在节点 web1 上查看 /var/log/messages 日志。信息如下：

```
Aug 24 02:43:19 web1 clurgmgrd[3252]: <notice> stop on script "mysqlscript"
   returned 5 (program not installed)
Aug 24 02:43:35 web1 clurgmgrd[3252]: <notice> Starting stopped service
   service:webserver
Aug 24 02:43:37 web1 avahi-daemon[3126]: Registering new address record for
   192.168.12.233 on eth0.
Aug 24 02:43:38 web1 in.rdiscd[4075]: setsockopt (IP_ADD_MEMBERSHIP): Address
   already in use
Aug 24 02:43:38 web1 in.rdiscd[4075]: Failed joining addresses
Aug 24 02:43:39 web1 clurgmgrd[3252]: <notice> Service service:webserver started
```

这个输出表明，web1 在重启后自动将集群服务和 IP 资源切换回来。

（2）节点 web1 异常宕机

在节点 web1 上执行如下命令，使系统内核崩溃：

```
[root@web2 ~]#echo c>/proc/sysrq-trigger
```

然后在节点 web2 上查看 /var/log/messages 日志。信息如下：

```
Aug 24 02:59:57 web2 openais[2689]: [TOTEM] entering GATHER state from 12.
Aug 24 03:00:10 web2 qdiskd[2712]: <notice> Node 4 evicted
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] entering GATHER state from 11.
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] Saving state aru 92 high seq received 92
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] Storing new sequence id for ring be0
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] entering COMMIT state.
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] entering RECOVERY state.
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] position [0] member 192.168.12.231:
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] previous ring seq 3036 rep 192.168.12.230
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] aru 92 high delivered 92 received flag 1
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] position [1] member 192.168.12.232:
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] previous ring seq 3036 rep 192.168.12.230
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] aru 92 high delivered 92 received flag 1
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] position [2] member 192.168.12.240:
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] previous ring seq 3036 rep 192.168.12.230
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] aru 92 high delivered 92 received flag 1
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] Did not need to originate any messages
   in recovery.
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] New Configuration:
Aug 24 03:00:17 web2 openais[2689]: [CLM  ]      r(0) ip(192.168.12.231)
Aug 24 03:00:17 web2 openais[2689]: [CLM  ]      r(0) ip(192.168.12.232)
Aug 24 03:00:17 web2 openais[2689]: [CLM  ]      r(0) ip(192.168.12.240)
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] Members Left:
Aug 24 03:00:17 web2 openais[2689]: [CLM  ]      r(0) ip(192.168.12.230)
Aug 24 03:00:17 web2 kernel: dlm: closing connection to node 4
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] Members Joined:
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] New Configuration:
Aug 24 03:00:17 web2 openais[2689]: [CLM  ]      r(0) ip(192.168.12.231)
Aug 24 03:00:17 web2 openais[2689]: [CLM  ]      r(0) ip(192.168.12.232)
Aug 24 03:00:17 web2 openais[2689]: [CLM  ]      r(0) ip(192.168.12.240)
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] Members Left:
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] Members Joined:
Aug 24 03:00:17 web2 openais[2689]: [SYNC ] This node is within the primary component
   and will provide service.
Aug 24 03:00:17 web2 openais[2689]: [TOTEM] entering OPERATIONAL state.
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.231
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.232
Aug 24 03:00:17 web2 openais[2689]: [CLM  ] got nodejoin message 192.168.12.240
Aug 24 03:00:17 web2 openais[2689]: [CPG  ] got joinlist message from node 2
Aug 24 03:00:17 web2 fenced[2728]: web1 not a cluster member after 0 sec post_fail_
   delay
Aug 24 03:00:17 web2 openais[2689]: [CPG  ] got joinlist message from node 3
Aug 24 03:00:17 web2 fenced[2728]: fencing node "web1"
Aug 24 03:00:17 web2 openais[2689]: [CPG  ] got joinlist message from node 1
```

```
Aug 24 03:00:55 web2 fenced[2728]: fence "web1" success
Aug 24 03:00:55 web2 kernel: GFS2: fsid=mycluster:my-gfs2.3: jid=0: Trying to
   acquire journal lock...
Aug 24 03:00:55 web2 kernel: GFS2: fsid=mycluster:my-gfs2.3: jid=0: Looking at
   journal...
Aug 24 03:00:55 web2 kernel: GFS2: fsid=mycluster:my-gfs2.3: jid=0: Done
Aug 24 03:00:55 web2 clurgmgrd[3239]: <notice> Taking over service service:webserver
   from down member web1
Aug 24 03:00:55 web2 avahi-daemon[3110]: Registering new address record for
   192.168.12.233 on eth0.
Aug 24 03:00:55 web2 clurgmgrd[3239]: <notice> Service service:webserver started
```

从输出日志可以看出，web1 在异常宕机后，首先由 qdiskd 进程将失败节点从集群中隔离，然后 qdiskd 进程将结果返回给 cman 进程。cman 进程接着调用 Fence 进程，最后 Fence 进程根据事先设置好的 Fence agent 调用 Fence 设备将 web1 成功隔离。在 clurgmgrd 进程接收到成功隔离的信息后，web2 开始接管 web1 的服务和 IP 资源，同时释放 dlm 锁，GFS2 文件系统可以正常读写。

4. 节点 Mysql1 宕机时

节点 Mysql1 在正常关机和异常宕机时，RHCS 的切换状态与上面讲述的 web1 节点宕机情况一模一样，这里不再重复介绍。

5. 4 个节点中的任意 3 个宕机

这里将 web1、Mysql2、Mysql1 依次异常宕机，然后在 web2 节点上查看 RHCS 是如何进行切换动作的。

首先停止 web1 节点，查看 /var/log/messages 日志。信息如下：

```
Aug 24 18:57:55 web2 openais[2691]: [TOTEM] entering GATHER state from 12.
Aug 24 18:58:14 web2 qdiskd[2714]: <notice> Writing eviction notice for node 4
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] entering GATHER state from 0.
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] Saving state aru 8e high seq received 8e
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] Storing new sequence id for ring c14
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] entering COMMIT state.
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] entering RECOVERY state.
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] position [0] member 192.168.12.231:
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] previous ring seq 3088 rep 192.168.12.230
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] aru 8e high delivered 8e received flag 1
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] position [1] member 192.168.12.232:
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] previous ring seq 3088 rep 192.168.12.230
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] aru 8e high delivered 8e received flag 1
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] position [2] member 192.168.12.240:
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] previous ring seq 3088 rep 192.168.12.230
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] aru 8e high delivered 8e received flag 1
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] Did not need to originate any messages
   in recovery.
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] New Configuration:
```

```
Aug 24 18:58:15 web2 openais[2691]: [CLM  ]     r(0) ip(192.168.12.231)
Aug 24 18:58:15 web2 openais[2691]: [CLM  ]     r(0) ip(192.168.12.232)
Aug 24 18:58:15 web2 openais[2691]: [CLM  ]     r(0) ip(192.168.12.240)
Aug 24 18:58:15 web2 kernel: dlm: closing connection to node 4
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] Members Left:
Aug 24 18:58:15 web2 openais[2691]: [CLM  ]     r(0) ip(192.168.12.230)
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] Members Joined:
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] CLM CONFIGURATION CHANGE
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] New Configuration:
Aug 24 18:58:15 web2 openais[2691]: [CLM  ]     r(0) ip(192.168.12.231)
Aug 24 18:58:15 web2 openais[2691]: [CLM  ]     r(0) ip(192.168.12.232)
Aug 24 18:58:15 web2 openais[2691]: [CLM  ]     r(0) ip(192.168.12.240)
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] Members Left:
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] Members Joined:
Aug 24 18:58:15 web2 openais[2691]: [SYNC ] This node is within the primary component
   and will provide service.
Aug 24 18:58:15 web2 openais[2691]: [TOTEM] entering OPERATIONAL state.
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] got nodejoin message 192.168.12.231
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] got nodejoin message 192.168.12.232
Aug 24 18:58:15 web2 openais[2691]: [CLM  ] got nodejoin message 192.168.12.240
Aug 24 18:58:15 web2 fenced[2730]: web1 not a cluster member after 0 sec post_fail_
   delay
Aug 24 18:58:15 web2 openais[2691]: [CPG  ] got joinlist message from node 3
Aug 24 18:58:15 web2 fenced[2730]: fencing node "web1"
Aug 24 18:58:15 web2 openais[2691]: [CPG  ] got joinlist message from node 1
Aug 24 18:58:15 web2 openais[2691]: [CPG  ] got joinlist message from node 2
Aug 24 18:58:17 web2 qdiskd[2714]: <notice> Node 4 evicted
Aug 24 18:58:29 web2 fenced[2730]: fence "web1" success
Aug 24 18:58:29 web2 kernel: GFS2: fsid=mycluster:my-gfs2.1: jid=3: Trying to
   acquire journal lock...
Aug 24 18:58:29 web2 kernel: GFS2: fsid=mycluster:my-gfs2.1: jid=3: Looking at
   journal...
Aug 24 18:58:29 web2 kernel: GFS2: fsid=mycluster:my-gfs2.1: jid=3: Done
Aug 24 18:58:30 web2 clurgmgrd[3300]: <notice> Taking over service service:webserver
   from down member web1
Aug 24 18:58:32 web2 avahi-daemon[3174]: Registering new address record for
   192.168.12.233 on eth0.
Aug 24 18:58:33 web2 clurgmgrd[3300]: <notice> Service service:webserver started
```

通过观察日志可以发现，qdiskd 进程会首先将节点 web1 从集群隔离，然后由 fenced 进程将 web1 成功隔离掉，最后，web2 接管了 web1 的服务和 IP 资源。这里有个先后问题，也就是说，只有隔离成功，集群资源切换才会进行。

如果 fenced 进程没有成功地将 web1 节点隔离掉，那么 RHCS 会进入等待状态，此时集群系统 GFS2 共享存储也将变得不可读写，直到 fenced 进程返回成功信息，集群才开始进行资源切换，同时 GFS2 文件系统也将恢复读写。

此时，在 web2 节点通过 cman_tool 查看集群状态。信息如下：

```
[root@web2 ~]# cman_tool  status
Version: 6.2.0
Config Version: 40
Cluster Name: mycluster
Cluster Id: 56756
Cluster Member: Yes
Cluster Generation: 3092
Membership state: Cluster-Member
Nodes: 3
Expected votes: 6
Quorum device votes: 2
Total votes: 5
Quorum: 3
Active subsystems: 9
Flags: Dirty
Ports Bound: 0 177
Node name: web2
Node ID: 1
Multicast addresses: 239.192.221.146
Node addresses: 192.168.12.240
```

从输出可知，集群节点数变为 3，同时 Quorum 值也由原来的 4 变为 3。Quorum 值是通过 N/2 取整加 1 得到的，其中 N 为所有集群节点的投票数加上表决磁盘投票值，也就是这里的“Total votes”值。

接着陆续将 Mysql2、Mysql1 异常宕机，宕机完成后，在 web2 节点上通过 cman_tool 查看集群状态。信息如下：

```
[root@web2 ~]# cman_tool  status
Version: 6.2.0
Config Version: 40
Cluster Name: mycluster
Cluster Id: 56756
Cluster Member: Yes
Cluster Generation: 3100
Membership state: Cluster-Member
Nodes: 1
Expected votes: 6
Quorum device votes: 2
Total votes: 3
Quorum: 2
Active subsystems: 9
Flags: Dirty
Ports Bound: 0 177
Node name: web2
Node ID: 1
Multicast addresses: 239.192.221.146
Node addresses: 192.168.12.240
```

从输出可知，整个集群系统节点数已经变为一个，但是集群系统仍然可用，GFS2 仍然

可以正常读写，这些都是 qdisk 的功能。在一个 RHCS 集群中，如果没有表决磁盘，仅剩一个节点的集群是不能工作的。

12.7.2 存储集群测试

GFS2 文件系统是 RHCS 集群中的共享存储，在集群各个节点挂载 GFS2 共享文件系统，然后在任意节点对共享分区进行数据的读写操作。例如，在节点 web2 上创建一个文件 ixdba 如下：

```
[root@web2 gfs2]# echo "This is GFS2 Files test" >/gfs2/ixdba
```

然后在节点 web1 上查看此文件。

```
[root@web1 gfs2]# more /gfs2/ixdba
This is GFS2 Files test
```

可以看到，写操作正常。

接着测试同时读写文件。首先在节点 web1 上编辑文件 ixdba，然后同时在节点 web2 上编辑 ixdba，此时会提示该文件被锁。

到这里为止，RHCS 所有功能点都已经测试完成了。在测试集群功能时，观察日志信息是很有必要的。通过观察日志输出可以知道 RHCS 更加详细的切换过程，如果切换失败，也会在日志中输出，然后根据错误信息可以很容易地定位问题。

12.8 本章小结

本章主要讲述了 RHCS 的使用，具体包括 RHCS 的结构与组成、RHCS 的运行原理、如何安装和配置一套 RHCS 系统，以及如何管理、维护和监控 RHCS 的正常运行。其中，安装和配置 RHCS 是本章的重点。读者阅读完本章后，要能够熟练地安装和配置 RHCS 集群套件，而管理与维护 RHCS 要在日常工作中慢慢积累起来。至于 RHCS 的概念和运行机制了解即可。

目前，RHCS 已经广泛应用在企业的各个领域。与其他 HA 软件相比，RHCS 显得比较复杂，管理和维护起来比较困难，但是 RHCS 的可靠性和稳定性是毋庸置疑的。随着人们对业务实时性和稳定性需求的不断提升，RHCS 的应用必将越来越广泛。

第 13 章　Oracle RAC 集群

本章主要介绍 Oracle RAC 集群系统的搭建、使用和维护。Oracle RAC 是 Oracle 公司推出的一套完整的数据库集群解决方案，RAC 数据库集群从高可用性和负载均衡两个方面提高 Oracle 的性能。为了解决数据读写的瓶颈，Oracle 在 RAC 集群的共享存储方面推出了独创的 ASM 存储机制，通过 ASM 管理数据，平衡了 I/O 消耗，同时提高了整体性能。目前 RAC 集群构架方案已经广泛应用于各个行业，它不但满足了高端业务的需求，同时也降低了管理和投入成本。如果您的数据库面临性能问题，不妨尝试一下 Oracle 的 RAC 数据库。

13.1　Oracle 集群体系结构

Oracle 数据库是当今非常流行的数据库之一，它占据了大部分高端市场，广泛应用在电信、证券、保险、能源、政府、航天、制造业、交通等行业。这些高端应用对业务的可靠性和稳定性要求非常高，例如 7×24 小时不间断服务、高并发访问等。为了解决 Oracle 数据库的高可用性和负载均衡问题，Oracle 公司推出了数据库集群系统，使 Oracle 数据库能够提供优质的服务，并达到高可用性和可伸缩性等级。通过该集群系统，不但满足了高端业务的应用需求，也显著降低了管理成本和管理的灵活性。

Oracle RAC，全称是 Oracle Real Application Cluster，即真正的应用集群，是 Oracle 提供的一个并行集群系统，整个集群系统由 Oracle Clusterware（集群就绪软件）和 Real Application Clusters（RAC）两大部分组成。Oracle RAC 的实质是：位于不同操作系统的 Oracle 实例节点同时访问同一个 Oracle 数据库，每个节点间通过私有网络进行通信，互相监控节点的运行状态，Oracle 数据库所有的数据文件、联机日志文件、控制文件等均放在集群的共享存储设备上，而共享存储设备可以是 RAW、ASM、OCFS2 等，所有集群节点可以同时读写共享存储。Oracle RAC 的基本拓扑结构如图 13-1 所示。

从图 13-1 可知，一个 Oracle RAC 数据库由多个服务器节点组成，每个服务器节点上都有自己独立的 OS、ClusterWare、Oracle RAC 数据库程序等，并且每个节点上都有自己的网络监听器（Oracle Rac Listener）。ClusterWare 是一个集群软件，主要用于集群系统管理。Oracle RAC 数据库程序用于提供 Oracle 实例进程，以供客户端访问集群系统。监听器主要用于监控自己的网络端口信息。所有的服务和程序通过操作系统去访问一个共享存储，最终完成数据的读写。共享存储的实现方式有很多种，可以通过使用自动存储管理（ASM）、Oracle 集群文件系统（OCFS）、裸设备（Raw）、网络区域存储（NAS）等来保证整个集群系统数据的一致性。

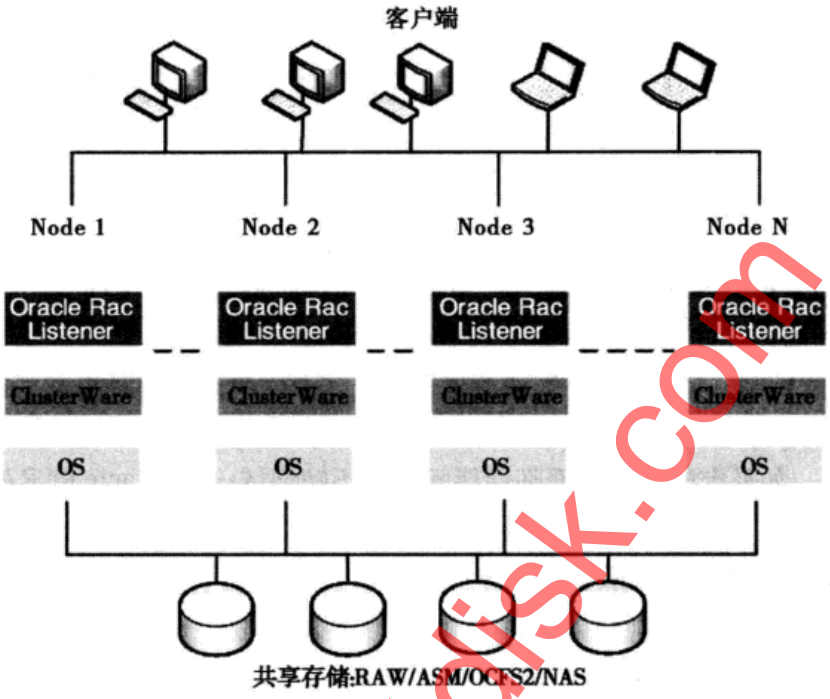

图 13-1 Oracle RAC 集群的结构

从 Oracle 10g 起，Oracle 提供了自己的集群软件，即 Oracle ClusterWare，它通过 CRS（即 Cluster Ready Services）来实现。这个软件是安装 Oracle RAC 的前提，也是 RAC 环境稳定运行的基础。在 Oracle 10g 之前的版本，安装 RAC 时必须借助于第三方的集群软件，而在 Oracle 10g 以后，安装 Oracle RAC 时，可以利用 Oracle 自带的集群软件，也可以用经过 RAC 认证的第三方集群软件来代替。

从 Oracle 的运行机制来说，集群中每台服务器就是一个 Oracle 实例，多个 Oracle 实例对应同一个 Oracle 数据库，组成了 Oracle 数据库集群。图 13-2 显示了这种关系。

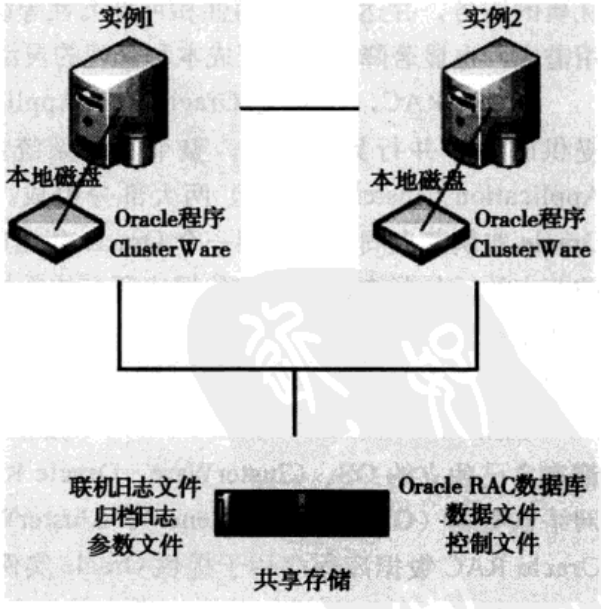

图 13-2 两个实例对应一个数据库

从图 13-2 可以看出，运行在两个节点上的数据库实例访问同一个 RAC 数据库，并且两个节点的本地磁盘仅用来存放 Oracle 安装程序和 ClusterWare 软件，而在共享存储上，存放了 Oracle 的数据文件、控制文件、联机日志文件、归档日志文件等，这是安装 Oracle RAC 时的一种数据存储分配方式。其实，

RAC 提供了多种数据存储方式，这个将在下面一节进行介绍。

13.2　Oracle ClusterWare 体系结构与进程介绍

13.2.1　Oracle ClusterWare 简介

Cluster Ready Services，简称 CRS，是 Oracle 开发的一个集群软件。与其他集群软件类似，CRS 主要完成集群成员管理、心跳监控、故障切换等功能。CRS 要求每个集群节点的操作系统必须相同，这样，通过 CRS 将多个节点的操作系统绑定到了一起，客户端对集群的访问，就像访问一台服务器一样。

CRS 主要由两个集群套件组成，分别是 voting disk 和 Oracle Cluster Registry。

- voting disk，即为表决磁盘，集群中每个节点定期评估自身的运行情况，然后会把它的状态信息放入到表决磁盘上，并且节点间也会互相查看其运行状态，并把信息传递给其他节点进而写入表决磁盘。当集群节点发生故障时，还可以通过表决磁盘进行投票仲裁等。因此，表决磁盘必须放在共享存储设备上，以保证每个节点都能访问到。表决磁盘可以是一个裸磁盘分区，也可以是一个独立的文件。由于表决磁盘仅记录节点运行信息，磁盘大小一般在 10 ～ 20MB 左右即可。
- Oracle Cluster Registry，简称 OCR，即集群注册服务，OCR 主要用于记录 RAC 中集群和数据库的配置信息。这些信息包括了集群节点的列表、集群数据库实例到节点的映射以及 CRS 应用程序资源信息。

CRS 使用两种心跳设备来验证节点成员的状态，保证集群的完整性：一种心跳设备是表决磁盘，集群同步服务进程每隔几秒钟就会向表决磁盘写入一条心跳信息，集群通过表决磁盘即可验证节点的状态，如果某个节点在指定的最大时间段内没有向表决磁盘写入信息，集群就认为此节点失效了，进而执行故障切换。另一种是节点间私有以太网的心跳，通过这个心跳可以判断节点间是否出现了网络故障。两种心跳机制的结合，有效地增加了集群的可靠性。

另外，CRS 建议：用于内部通信的私有以太网心跳必须与用于 RAC 节点间通信的网络分开，也就是不能在同一网络中。如果 RAC 节点间通信的网络与私有以太网心跳在同一个网络内，那么，必须保证该网络不能被非集群系统的节点访问到。

13.2.2　Oracle ClusterWare 进程介绍

Oracle ClusterWare 通过 CRS 来完成集群功能。CRS 包含了一组相互协作的后台进程，下面详细介绍 CRS 中几个很重要的后台进程。

（1）Cluster Synchronization Services

简称 CSS，用于管理与协调集群中各节点的关系，并用于节点间通信。当节点在加入或离开集群时，都由 CSS 通知集群。CSS 在集群中对应的后台进程为 CSSD，该进程由 Oracle 用户运行和管理。当节点发生故障时，CSSD 会自动重启操作系统。

（2）Cluster Ready Services

简称 CRS，是管理群集内高可用操作的主要程序，在集群中 CRS 管理所有资源，包括数据库、服务、实例、vip 地址、监听器、应用进程等。CRS 在集群中对应的后台进程为 CRSD，该进程可以对集群资源进行启动、停止、监视和容错等操作。在正常状态下，CRSD 监控节点的各种资源，当某个资源发生异常时，自动重启或者切换该资源。

（3）Process Monitor Daemon

简称 OPROCD，此进程被锁定在内存中，用于监控集群及提供 I/O 防护（I/O fencing）。OPROCD 运行在每个节点上，且定期执行运行情况检测，如果在超过它所希望的间隔内，仍然不能和某个节点通信，那么，OPROCD 将会重置处理器并重启节点。一个 OPROCD 故障也将导致 ClusterWare 重启节点。

（4）Oracle Notification Service

简称 ONS，即 Oracle 通告服务，主要用于发布和订阅 Fast Application Notification 事件。

（5）Event Management

简称 EVM，是一个事件检测的后台进程，由 Oracle 用户运行和管理。

13.3 RAC 数据库体系结构与进程

13.3.1 RAC 简介

RAC 是一个具有共享缓存体系结构的集群数据库，它克服了传统的不共享和共享磁盘方法的限制，为所有业务应用程序提供了一种具有可伸缩性和可用性的数据库解决方案，它一般与 Oracle ClusterWare 或第三方集群软件共同组成 Oracle 集群系统。因此可以说，通过集群系统访问的数据库就是 RAC 数据库。

RAC 是一个全共享式的体系架构，它的所有数据文件、控制文件、联机日志文件、参数文件等都必须存放在共享磁盘中，只有这样，集群所有节点才能被访问到。针对共享数据在磁盘的组织形式，RAC 支持多种存储方式，可以使用下面几种方式中的任意一种。

（1）裸设备（Raw devices）

即不经过文件系统，将数据直接写入磁盘中，这种方式的好处是磁盘 I/O 性能很高，适合写操作频繁的业务系统。这种方式的缺点也很明显：数据维护和备份不方便，备份只能通

过 dd 命令或者基于块级别的备份设备来完成，这无疑增加了维护成本。

（2）集群文件系统

为了支持共享存储，Oracle 开发出了集群文件系统 OCFS，这个文件系统可用于 Windows、Linux 和 Solaris，现在已经发展到了 OCFS2。通过 OCFS2 文件系统，多个集群节点可以同时挂载一个共享磁盘分区，并且可以同时读写磁盘而不破坏数据。有了集群文件系统，RAC 数据库的管理和维护变得非常直观和方便，但对于大量读写的业务系统，集群文件系统的性能不是很高。另外，Oracle RAC 也支持第三方的集群文件系统，例如 Red Hat 的 GFS 等。

（3）网络文件系统（NFS）

（4）Automated Storage Management

Automated Storage Management，简称 ASM，是 Oracle 推荐的共享数据存储方式，是 Oracle 10g 包含的一个特性。ASM 其实就是通过 RAW 方式存储数据，但是加入了数据管理功能，它通过将数据直接写入磁盘，避免了经过文件系统产生的 I/O 消耗。因此，使用 ASM 可以很方便地管理共享数据，并提供异步 I/O 性能。ASM 还可以通过分配 I/O 负载来优化性能，免除了手动调整 I/O 的麻烦。

13.3.2　Oracle RAC 的特点

通过 RAC 数据库，可以构建一个高性能、高可靠的数据库集群系统。RAC 的优势如下：

1）可以实现多个节点间的负载均衡。

RAC 数据库集群可以根据设定的调度策略，在集群节点间实现负载均衡，因此，RAC 数据库每个节点都是工作的，并处于互相监控状态，当某个节点出现故障时，RAC 集群自动将失败节点从集群隔离，并将失败节点的请求自动转移到其他正常运行的节点上，从而实现服务的透明切换。

2）可以提供高可用服务。

这个是 Oracle ClusterWare 实现的功能，通过 CRS 可以实现节点状态监控，故障透明转移，这保证了 Oracle 数据库可以对外不间断地提供服务。

3）通过横向扩展提高并发连接数。

RAC 的这个优点使其非常适合应用于大型的联机事务系统中。

4）通过并行执行技术提高了事务响应时间。

这个是 RAC 集群的一大优势，通常体现在数据分享系统中。

5）具有很好的扩展性。

在集群系统不能满足繁忙的业务系统时，RAC 数据库可以很方便地添加集群节点，并且可以在线完成节点的添加，并自动将其加入集群系统，不存在宕机时间。在不需要某个集群节点时，删除节点也非常简单。

当然，RAC 数据库也有一些缺点：

1）与单机数据库相比，管理维护更复杂，对维护人员要求更高。

2）底层规划设计不好时，系统整体性能会较差，甚至不如单机系统的性能。如果对 RAC 数据库不是很了解，建议不要马上在生产环境中使用。

3）由于 RAC 集群系统需要多个节点，因此需要购买多台服务器。同时还需要 Oracle 企业级版本数据库，这也增加了软硬件成本。

13.3.3 RAC 进程管理

RAC 数据库是由多个节点构成的，每个节点就是一个数据库实例，而每个实例都有自己的后台进程和内存结构。并且在 RAC 集群中，每个实例的后台进程和内存结构都是相同的，从整体上看起来，就像是一个单一数据库的镜像。但是，RAC 数据库在结构上与单实例库也有不同之处：

1）RAC 数据库的每个实例至少拥有一个额外的重做线程（redo thread）。

2）RAC 数据库的每个实例都拥有自己的撤销表空间（undo tablespace）。

很显然，这种机制是每个实例独立使用自己的重做线程和撤销表空间，各自锁定自己修改的数据。RAC 的这种设计方式，把多个实例的操作相对独立地分开。那么 RAC 数据库如何实现节点数据的一致性呢？其实每个 RAC 实例的 SGA 内有一个 buffer cache（缓冲区），通过 Cache Fusion（缓存融合）技术，RAC 在各个节点之间同步 SGA 中的缓存信息，从而保证了节点数据的一致性，同时也提高了集群的访问速度。

RAC 数据库最大的特点是共享。那么多个节点如何有条不紊地实现数据共享呢？就是通过 RAC 的两个进程：Global Cache Service（GCS）和 the Global Enqueue Service（GES）。

全局缓存服务（GCS）和全局队列服务（GES）是最基本的 RAC 进程，主要用于协调对共享数据库和数据库内的共享资源的同时访问。同时，GES 和 GCS 通过全局资源目录（Global Resource Directory，GRD）来记录和维护每个数据文件的状态信息，而 GRD 保存在内存中，内容分布存储在所有实例上，每个实例都管理部分内容。

RAC 通过几个特别的进程与 GRD 相结合，使 RAC 可以使用缓存融合技术。实现这些功能的不同进程在不同版本中名称不同，以下是一些比较重要的进程：

- Global Cache Service Processes（LMS）

LMS，即全局缓冲服务进程，以前称为 Lock Manager Services，此进程主要用来管理集群内数据块的访问，并在不同实例的 buffer cache 中传输块镜像。

- Global Enqueue Service Monitor（LMON）

LMON，即锁监控进程，以前称为 Lock Monitor，主要监视群集内的全局资源和集群间的资源交互，并负责管理实例、处理异常，以及集群队列的恢复操作。

- Global Enqueue Service Daemon（LMD）

LMD，即全局资源服务进程，以前称为 Lock Manager Daemon，此进程主要管理对全局队列和全局资源的访问，并更新相应队列的状态，处理来自其他实例的资源请求。

- Lock Processes（LCK）

LCK 进程主要用来管理实例间资源请求和跨实例调用操作，并管理除 Cache Fusion 以外的资源请求，比如 library 和 row cache 的请求等。

- Diagnosability Daemon（DIAG）

DIAG 进程主要用来捕获实例中失败进程的诊断信息，并生成相应的 TRACE 文件。

13.3.4　RAC 数据库存储规划

安装 RAC 数据库时涉及的软件有 Oracle ClusterWare、Oracle RAC 数据库软件，同时还涉及 voting disk、OCR 等，每个软件需要占用的磁盘空间大小如表 13-1 所示。

表 13-1　RAC 数据库各个软件的用途及占用的磁盘空间

项目名称	用　途	需要磁盘空间
Oracle ClusterWare 软件	Oracle 集群软件	500~800MB
Oracle RAC 数据库软件	Oracle RAC 运行程序	2~3GB
RAC 数据库	RAC 共享数据库文件	5~8GB
voting disk（表决磁盘）	用于记录集群节点信息	20~50MB
OCR（集群注册服务）	用于存储集群配置信息	100~200MB
Flash Recovery Area	用于快速恢复数据	2GB

在了解了 RAC 每部分所需的磁盘空间大小后，就可以根据每部分的用途来规划数据存储了。RAC 广泛支持各种数据存储方式，例如单一日志文件系统 ext2/ext3、集群文件系统 OCFS2/GFS、网络文件系统 NFS、裸设备 RAW、自动存储管理 ASM 等。表 13-2 列出了 RAC 数据库各部分可以使用的存储类型。

表 13-2　RAC 数据库各部分可以使用的存储类型及存储位置

项目名称	可用的存储类型	存储位置
Oracle Clusterware 软件	单一文件系统 ext2/ext3、网络文件系统 NFS	本地磁盘或网络磁盘（NFS）
Oracle RAC 数据库程序	单一文件系统 ext2/ext3、网络文件系统 NFS、集群文件系统 OCFS2	本地磁盘、网络磁盘（NFS）或共享磁盘（OCFS2）
RAC 数据库	NFS、OCFS2、RAW、ASM	共享磁盘或网络磁盘
voting disk（表决磁盘）	NFS、OCFS2、RAW	共享磁盘或网络磁盘
OCR（集群注册服务）	NFS、OCFS2、RAW	共享磁盘或网络磁盘
Flash Recovery Area	NFS、OCFS2、ASM	共享磁盘或网络磁盘

从表 13-2 可以看出，RAC 所有部分的内容都可以用 NFS 来存储，但是 NFS 方式需要 NAS 设备的支持，如果没有 NAS 设备，除了 Oracle ClusterWare 软件和 Oracle RAC 数据库

程序，其他数据必须都存储在共享磁盘上。

具体使用哪种存储策略，要根据安装 RAC 环境的不同而不同。这里推荐 3 种常用的存储方式，读者可以采用其中之一。

- 将 Oracle ClusterWare 集群软件和 RAC 数据库软件安装在本地磁盘，也就是安装到每个集群节点上，并且使用裸设备来存储表决磁盘和 OCR，最后用 ASM 来存储和管理 RAC 共享数据库和快速恢复文件。详细信息如表 13-3 所示。

表 13-3 构建 Oracle RAC 集群的存储策略一

项目名称	使用的存储类型	存储位置
Oracle ClusterWare 软件	ext2/ext3	本地磁盘
Oracle RAC 数据库程序	ext2/ext3	本地磁盘
RAC 数据库	ASM	共享磁盘
voting disk（表决磁盘）	RAW	共享磁盘
OCR（集群注册服务）	RAW	共享磁盘
Flash Recovery Area	ASM	共享磁盘

- 将 Oracle ClusterWare 集群软件安装在本地磁盘，而将其他所有部分安装在共享磁盘，如表 13-4 所示。

表 13-4 构建 Oracle RAC 集群的存储策略二

项目名称	使用存储类型	存储位置
Oracle ClusterWare 软件	ext2/ext3	本地磁盘
Oracle RAC 数据库程序	OCFS2	共享磁盘
RAC 数据库	ASM	共享磁盘
Voting disk（表决磁盘）	OCFS2	共享磁盘
OCR（集群注册服务）	OCFS2	共享磁盘
Flash Recovery Area	ASM	共享磁盘

- 将 RAC 数据库的所有部分全部存放在集群文件系统 OCFS2 中，仅将 Oracle ClusterWare 集群软件安装在本地磁盘，如表 13-5 所示。

表 13-5 构建 Oracle RAC 集群的存储策略三

项目名称	使用的存储类型	存储位置
Oracle Clusterware 软件	ext2/ext3	本地磁盘
Oracle RAC 数据库程序	OCFS2	共享磁盘
RAC 数据库	OCFS2	共享磁盘
voting disk（表决磁盘）	OCFS2	共享磁盘
OCR（集群注册服务）	OCFS2	共享磁盘
Flash Recovery Area	OCFS2	共享磁盘

3 种数据存储方式各有优劣，第一种存储策略性能最好，但是复杂度也最高，管理和维护都相对复杂。第三种存储策略最简单，安装、配置和维护都相对容易，但是性能不是很好，对于 I/O 操作频繁的业务系统，会存在性能瓶颈。第二种存储策略介于第一种和第三种之间，是集群文件系统 OCFS2 和 ASM 混合的存储管理方式。

至于使用哪种存储方式，需要了解业务需求，并根据自己的实际环境整体考虑。本章采用第一种存储策略的实例来讲述 Oracle RAC 的安装、配置、管理和维护。

13.4 安装 Oracle RAC 数据库

Oracle RAC 数据库的安装相对单机数据库复杂一些，因为整个安装过程涉及网络、操作系统、存储等方面，一个环节设置不当，都可能导致安装失败。本节将详细的讲述 RAC 数据库的安装过程。

13.4.1 安装前的系统配置需求

安装 Oracle RAC 数据库需要的组件分为软、硬件两部分，一个推荐的软硬件配置情况如表 13-6 所示。

表 13-6 一个推荐的软硬件配置情况

RAC 组件	用途及最低需求
服务器	最少两台，也可以三台或更多
操作系统	推荐使用 Oracle 认证的系统，版本不要太老，稳定版本最好
CPU/ 内存	内存至少 1GB，建议 4~8GB
磁盘大小	每台服务器磁盘空间大于 20GB
网卡	每台服务器至少两个网卡，推荐每台主机用 4 个以上千兆网卡进行网卡冗余绑定
私有以太网络	私有网络建议单独连接在千兆交换机上
共享存储设备	推荐 SAN 存储设备，通过光纤连接到每台主机
HBA 卡	推荐使用两个 HBA 卡进行冗余

根据这个配置建议，本节要讲述的安装环境如下：

- 操作系统：CentOS 5.3。
- 服务器：两台 PC Server，即两个集群节点。
- 内存：每个服务器内存 2GB。
- 网卡：两块，一块用于公用网络通信，另一块用于私有网络心跳监控。
- 共享磁盘：两块共享磁盘，大小分别为 10GB 和 30GB。

更详细的拓扑结构信息如图 13-3 所示。

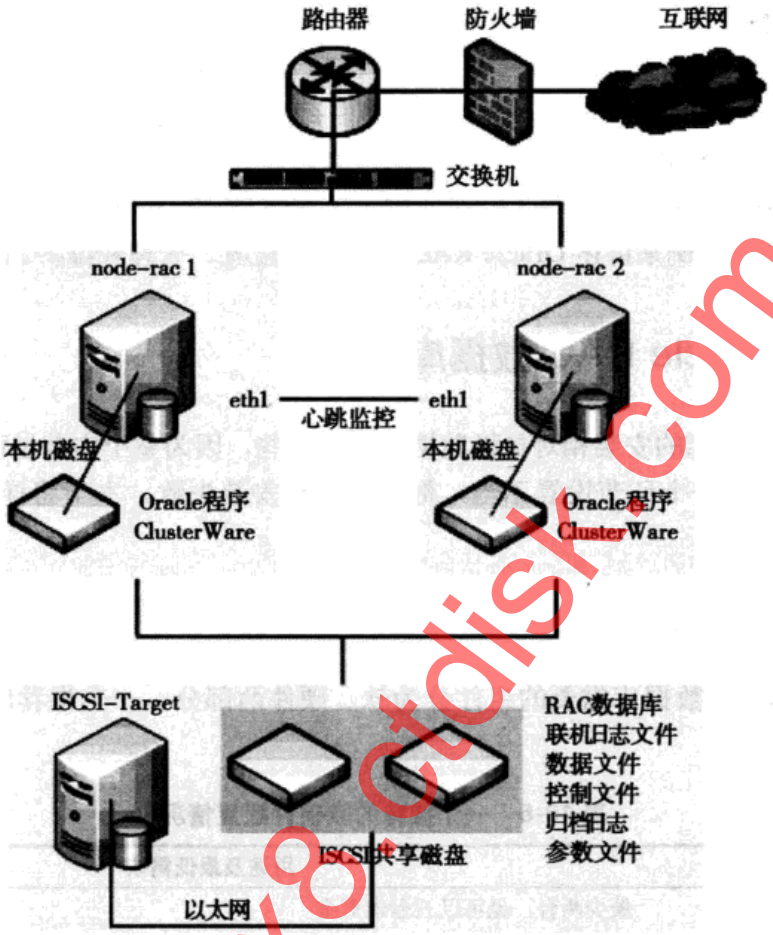

图 13-3 Oracle RAC 安装拓扑图

为了方便安装 RAC 数据库，在安装操作系统时，建议选择如下系统包：

- 桌面环境：XWindows system、GNOME desktop environment。
- 开发工具：development tools、x software development、gnome software development、kde software development。

Oracle RAC 数据库涉及公用网络和私有网络，因此要进行网络划分和 IP 地址规划。表 13-7 列出了要安装的 RAC 数据库对应的 IP 地址、主机名及网络连接类型。

表 13-7 安装的 RAC 数据库对应的 IP 地址、主机名及网络连接类型

IP 地址	主机名	网络类型	解析方式
192.168.12.231	node-rac1	公用 IP	/etc/hosts
192.168.12.232	node-rac2	公用 IP	/etc/hosts
192.168.12.230	node-vip1	虚拟 IP	/etc/hosts
192.168.12.240	node-vip2	虚拟 IP	/etc/hosts
10.10.10.1	node-priv1	私有 IP	/etc/hosts
10.10.10.2	node-priv2	私有 IP	/etc/hosts
192.168.12.246	iscsi-target	公用 IP	无

根据业务系统的实际需求，合理规划 RAC 数据库的存储策略非常重要。这里依据表 13-3 给定的数据存储方式进行讲述。

13.4.2　设置数据库安装资源

安装 RAC 数据库需要的软件包有 3 个部分，分别是 Oracle RAC 安装程序包、Oracle ASMLib 工具包及系统补丁包。这里 Oracle 的安装版本为 Oracle 11g，下面详细介绍软件包信息。

（1）Oracle 11g Release 1 (11.1.0.6.0) 软件包

下载地址：

http://www.oracle.com/technetwork/database/enterprise-edition/downloads/111060-linuxsoft-085130.html

软件包名称：

linux_11gR1_database_1013.zip

linux_x86_11gR1_clusterware.zip

软件包说明：总共需要下载两个安装程序，一个是 Oracle RAC 安装程序包，另一个是 Oracle ClusterWare 安装程序包。

（2）Oracle ASMLib 工具包

下载地址：http://www.oracle.com/technetwork/topics/linux/downloads/rhel5-084877.html

软件包名称：

oracleasmlib-2.0.4-1.el5.i386.rpm

oracleasm-2.6.18-194.11.1.el5-2.0.5-1.el5.i686.rpm

oracleasm-support-2.1.3-1.el5.i386.rpm

软件包说明：这 3 个软件包是使用 ASM 存储管理方式必需的驱动工具包。

（3）系统补丁包

下载地址：

http://www.idevelopment.info/data/Oracle/DBA_tips/Oracle11gRAC/Install11gR1RACOnCentOS51/RPMS/redhat-release-5-1.0.el5.centos.1.i386.rpm

软件包名称：

redhat-release-5-1.0.el5.centos.1.i386.rpm

软件包说明：由于 CentOS 不在 Oracle 支持平台之列，因此安装检测时无法通过。通过安装这个软件包，可以使安装检测顺利通过。

系统补丁软件包的安装非常简单，这里不再讲述。分别在两个节点上安装这些系统补丁包，安装成功后才可以进入下一步。

13.4.3　配置主机解析文件

为了使每个主机间可以正常通信，需要在每个节点上修改本地解析文件，即 /etc/hosts

文件。在两个节点上添加如下配置信息：

```
192.168.12.231          node-rac1
192.168.12.232          node-rac2
192.168.12.230          node-vip1
192.168.12.240          node-vip2
10.10.10.1              node-priv1
10.10.10.2              node-priv2
```

13.4.4 检查所需软件包

在每个节点上执行相同的操作，命令如下：

```
rpm -q make binutils  libaio-devel libaio elfutils-libelf-devel compat-libstdc++-
   33 libgcc  gcc gcc-c++ glibc sysstat libstdc++ libstdc++-devel  unixODBC-devel
   unixODBC
```

如果出现某个软件包没有安装，要安装该软件包。

13.4.5 配置系统内核参数

Linux 的内核参数信息都存在于内存中，可以通过命令直接修改，并且修改后直接生效。但是，在系统重新启动后，原来设置的参数值就会丢失，而系统每次启动时都会自动到 /etc/sysctl.conf 文件中读取内核参数，因此将内核的参数配置写入这个文件中是一个比较好的选择。

Oracle 对 Linux 系统内核参数有严格的要求，如果设置不当，就会导致安装失败。编辑 /etc/sysctl.conf 文件，修改后的参数配置如下：

```
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
kernel.msgmnb = 65536
kernel.msgmax = 65536
net.core.rmem_default = 4194304
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 262144
kernel.shmmax = 1073741823
kernel.sem = 250 32000 100 128
fs.file-max = 65536
net.ipv4.ip_local_port_range = 1024 65000
```

下面简单讲述常用的几个内核参数的含义：

- kernel.shmmax，表示单个共享内存段的最大值，以字节为单位，此值一般为物理内存

的一半，不过大一点也没关系，这里设定为 1GB。

- kernel.shmmni，表示单个共享内存段的最小值，一般为 4KB，即 4096bits。
- kernel.shmall，表示可用共享内存的总量，单位是页，在 32 位系统上一页等于 4K，也就是 4096 字节。
- fs.file-max，表示文件句柄的最大数量。文件句柄表示在 Linux 系统中可以打开的文件数量。
- net.ipv4.ip_local_port_range，表示端口的范围为指定的内容。
- kernel.sem，用来设置 Linux 的信号量，可以通过如下命令查看：

```
[root@node-rac1 rac]# ipcs -ls
------ Semaphore Limits --------
max number of arrays = 128
max semaphores per array = 250
max semaphores system wide = 32000
max ops per semop call = 32
semaphore max value = 32767
```

也可以使用以下命令：

```
[root@node-rac1 rac]#cat /proc/sys/kernel/sem
250 32000 32 128
```

这 4 个输出值的含义如下：

- SEMMSL，此参数用于控制每个信号集的最大信号数，Oracle 建议将 SEMMNI 设置为不小于 100。
- SEMMNS，此参数用于控制整个 Linux 系统中信号（而不是信号集）的最大数量。
- SEMOPM，此参数用于控制每个 semop 系统调用可以执行的信号操作数，Oracle 建议将 SEMOPM 的值设置为不少于 100。
- SEMMNI，此内核参数用于控制整个 Linux 系统中信号集的最大数量，Oracle 建议将 SEMMNI 设置为不小于 100。

- net.core.rmem_default：表示接收套接字缓冲区大小的默认值（以字节为单位）。
- net.core.rmem_max：表示接收套接字缓冲区大小的最大值（以字节为单位）。
- net.core.wmem_default：表示发送套接字缓冲区大小的默认值（以字节为单位）。
- net.core.wmem_max：表示发送套接字缓冲区大小的最大值（以字节为单位）。

13.4.6 设置 Shell 对 Oracle 用户的限制

以 root 用户身份，在每个节点上执行相同的操作。

首先，修改 /etc/security/limits.conf，在文件最后添加如下内容：

```
oracle soft nproc 2047
oracle hard nproc 16384
```

```
oracle soft nofile 1024
oracle hard nofile 65536
```

接着，修改 /etc/pam.d/login，在文件最后添加如下内容：

```
session required /lib/security/pam_limits.so
```

最后，修改 /etc/profile，在文件最后添加如下内容：

```
if [ $USER = "oracle" ]; then
           if [ $SHELL = "/bin/ksh" ]; then
                    ulimit -p 16384
                    ulimit -n 65536
                else
               ulimit -u 16384 -n 65536
                fi
fi
```

所有修改完毕后重启所有 Linux 系统。

13.4.7 配置 hangcheck-timer 内核模块

以 root 用户身份在所有节点上做如下配置。

查看模块是否存在：

```
[root@node-rac1 ~]#find /lib/modules -name "hangcheck-timer.ko"
```

接着，编辑 /etc/modprobe.conf 文件：

```
[root@node-rac1 ~]# vi /etc/modprobe.conf
```

在文件的末尾加入一行：

```
options hangcheck-timer hangcheck_tick=30 hangcheck_margin=180
```

然后，将 hangcheck-timer 模块配置为自动启动：

```
[root@node-rac1 ~]#vi /etc/rc.d/rc.local
```

在文件的末尾加入一行：

```
/sbin/modprobe hangcheck_timer
```

接着，启动 hangcheck：

```
[root@node-rac1 ~]# /sbin/modprobe hangcheck_timer
```

最后，检查 hangcheck 是否成功启动：

```
[root@node-rac1 ~]#grep hangcheck /var/log/messages | tail -2
Aug 26 19:08:17 Mysql1 kernel: Hangcheck: starting hangcheck timer 0.9.0 (tick is
   30 seconds, margin is 180 seconds).
```

如果显示上面的输出信息，说明已经成功启动 hangcheck。

13.4.8 配置系统安全设置

由于在安装 RAC 数据库时，安装进程需要在每个节点间传送数据，这就要求每个节点间是相互信任的，因此，最简单的方式就是关闭系统的安全限制。常用的方式是关闭 Linux 系统的 iptables 及 selinux，在每个节点上执行相同的操作。

关闭 iptables 很简单，可以直接执行如下命令：

```
[root@node-rac1 ~]# iptables -F
[root@node-rac1 ~]# /etc/init.d/iptables  save
```

最后，将 selinux 禁用即可，也就是修改 /etc/selinux/config 文件。修改后的内容为：

```
SELINUX=disabled
SELINUXTYPE=targeted
```

13.4.9 创建 Oracle 用户和组

在安装之前，需要创建两个用户组和一个用户，分别用于 Oracle 安装和管理。在两个节点执行相同的操作，操作如下：

```
[root@node-rac1 ~]#groupadd -g 1001 dba
[root@node-rac1 ~]#groupadd -g 1002 oinstall
[root@node-rac1 ~]#useradd -u 1001 -g oinstall -G dba oracle
```

然后，为 Oracle 用户设置密码：

```
[root@node-rac1 ~]# passwd oracle
```

最后，确认匿名用户 nobody 是否存在于系统中，因为在安装完成后 nobody 用户需要执行一些扩展任务。

```
[root@node-rac1 ~]# id nobody
uid=99(nobody) gid=99(nobody) groups=99(nobody)
```

这样，用户和组就创建完毕了。

13.4.10 设置 Oracle 用户环境变量

用文本编辑器 vi 编辑 /home/oracle/.bash_profile 文件，在文件最后添加如下内容，这里以 node-rac1 为例。同理，需要在节点 node-rac2 执行相同的操作。

```
export ORACLE_BASE=/u01/oracle
export ORACLE_HOME=$ORACLE_BASE/product/11.0.6/rac_db
export ORA_CRS_HOME=/app/crs/product/11.0.6/crs
export ORACLE_PATH=$ORACLE_BASE/common/oracle/sql:.:$ORACLE_HOME/rdbms/admin
export ORACLE_SID=racdb1
export NLS_LANG=AMERICAN_AMERICA.zhs16gbk
export NLS_DATE_FORMAT="YYYY-MM-DD HH24:MI:SS"
export PATH=.:${PATH}:$HOME/bin:$ORACLE_HOME/bin:$ORA_CRS_HOME/bin
```

```
export PATH=${PATH}:/usr/bin:/bin:/usr/bin/X11:/usr/local/bin
export PATH=${PATH}:$ORACLE_BASE/common/oracle/bin
export ORACLE_TERM=xterm
export TNS_ADMIN=$ORACLE_HOME/network/admin
export ORA_NLS10=$ORACLE_HOME/nls/data
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:$ORACLE_HOME/oracm/lib
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/lib:/usr/lib:/usr/local/lib
export CLASSPATH=$ORACLE_HOME/JRE
export CLASSPATH=${CLASSPATH}:$ORACLE_HOME/jlib
export CLASSPATH=${CLASSPATH}:$ORACLE_HOME/rdbms/jlib
export CLASSPATH=${CLASSPATH}:$ORACLE_HOME/network/jlib
export THREADS_FLAG=native
export TEMP=/tmp
export TMPDIR=/tmp
```

根据 Oracle 官方的建议，这里将 Oracle RAC 数据库程序和 Oracle ClusterWare 软件安装在了不同的目录。而“ORACLE_SID”在节点 node-rac2 上应该设置为“export ORACLE_SID=racdb2”。

设置完毕 Oracle 用户环境变量后，还需要在两个节点上创建环境变量中指定的安装目录。基本操作如下：

```
[root@node-rac1 ~]# mkdir -p /u01/oracle/product/11.0.6/rac_db
[root@node-rac1 ~]# mkdir -p /app/oracrs/product/11.0.6/crs
[root@node-rac1 ~]# chown -R oracle:oinstall /u01/oracle
[root@node-rac1 ~]#chown -R oracle:oinstall /app
```

13.4.11 配置节点间 SSH 信任

在安装 RAC 过程中，OUI 程序会使用 ssh 和 scp 命令来执行远程复制操作，将文件从安装节点复制到其他节点上。如果节点间不相互信任，那么传输过程就需要输入密码，从而导致安装失败，因此必须在所有的节点上为 Oracle 用户配置节点间的互信。

1. 在每个节点上创建 RSA 密钥和公钥

1）以 Oracle 用户登录。

2）在 Oracle 用户的根目录内创建 .ssh 目录并设置读取权限。

```
[oracle@node-rac1 ~]$ mkdir ~/.ssh
[oracle@node-rac1 ~]$ chmod 700 ~/.ssh
```

3）使用 ssh-keygen 命令生成基于 SSH 协议的 RSA 密钥。

```
[oracle@node-rac1 ~]$ cd ~/.ssh
[oracle@node-rac1 .ssh]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/oracle/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

```
Your identification has been saved in /home/oracle/.ssh/id_rsa.
Your public key has been saved in /home/oracle/.ssh/id_rsa.pub.
The key fingerprint is:
dd:69:5a:aa:e6:85:88:a4:07:72:ab:15:7b:3b:4a:77 oracle@node-rac1
```

在提示保存私钥（key）和公钥（public key）的位置时，选择默认值，然后按回车键即可。

2. 整合公钥文件

1）以 Oracle 用户登录。

2）在要执行 Oracle 安装程序的节点 node-rac1 上执行如下操作：

```
[oracle@node-rac1 ~] $ cd ~/.ssh
[oracle@node-rac1 .ssh]$ ssh node-rac1 cat /home/oracle/.ssh/id_rsa.pub >>
    authorized_keys
[oracle@node-rac1 .ssh]$ ssh node-rac2 cat /home/oracle/.ssh/id_rsa.pub >>
    authorized_keys
[oracle@node-rac1 .ssh]$ chmod 600 ~/.ssh/authorized_keys
[oracle@node-rac1 .ssh]$scp authorized_keys  node-rac2:/home/oracle/.ssh/
```

这个操作过程是将两个节点生成的公钥文件整合为一个 authorized_keys 文件，然后进行授权，并将 authorized_keys 复制到另一个节点上。

3）测试 SSH 互信。

首先在 node-rac1 节点上执行如下命令：

```
[oracle@node-rac1 ~]$ ssh node-rac1 date
[oracle@node-rac1 ~]$ ssh node-rac2 date
```

然后在 node-rac2 节点上执行如下命令：

```
[oracle@node-rac2 ~]$ ssh node-rac1 date
[oracle@node-rac2 ~]$ ssh node-rac2 date
```

如果不需要输入密码就出现系统当前日期，这说明 SSH 互信已经配置成功了。

13.4.12 配置共享存储系统

在这里这个环境中，共享存储由一台 iSCSI Target 主机来提供，通过以太网，假定将两块本地磁盘 /dev/sdb 和 /dev/sdc 共享给 RAC 数据库的两个节点。由于 iSCSI 的安装已经在前面进行了详细讲述，这里仅简单介绍一下共享磁盘在 iSCSI target 主机上的配置。

1. 在 iSCSI Target 上配置 iSCSI 共享磁盘

首先修改 /etc/iet/ietd.conf 文件，增加如下内容：

```
Target iqn.2002-04.net.ixdba:sdb
Lun 0 Path=/dev/sdb,Type=fileio

Target iqn.2002-04.net.ixdba:sdc
Lun 0 Path=/dev/sdc,Type=fileio
```

然后修改 /etc/iet/initiators.allow，修改后的内容如下：

```
iqn.2002-04.net.ixdba:sdb 192.168.12.231,192.168.12.232
iqn.2002-04.net.ixdba:sdc 192.168.12.231,192.168.12.232
```

最后，重启 iscsi-target 服务，执行的命令如下：

```
/etc/init.d/iscsi-target  restart
```

2. 在 node-rac1 和 node-rac2 客户端配置 iSCSI

首先安装 iSCSI 客户端软件，即 iscsi-initiator：

```
[root@node-rac1 ~]#yum install iscsi-initiator*
```

然后执行 iSCSI 发现操作：

```
[root@node-rac1 ~]# /etc/init.d/iscsi start
[root@node-rac1 ~]# iscsiadm -m discovery -t sendtargets -p 192.168.12.246
192.168.12.246:3260,1 iqn.2002-04.net.ixdba:sdb
192.168.12.246:3260,1 iqn.2002-04.net.ixdba:sdc
[root@node-rac1 ~]# /etc/init.d/iscsi restart
```

同理在 node-rac2 上执行相同的操作。执行完毕后，登录 node-rac1 节点，然后执行 fdisk 命令应该可以看到如下共享磁盘信息：

```
[root@node-rac1 ~]# fdisk -l
Disk /dev/sda: 21.4 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14         140     1020127+  82  Linux swap / Solaris
/dev/sda3             141        2610    19840275   83  Linux

Disk /dev/sdb: 10.7 GB, 10737418240 bytes
64 heads, 32 sectors/track, 10240 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System

Disk /dev/sdc: 32.2 GB, 32212254720 bytes
64 heads, 32 sectors/track, 30720 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

Disk /dev/sdc doesn't contain a valid partition table
```

从输出可以看到，节点已经识别了 iSCSI Target 共享过来的两块磁盘分区，大小分别是 10.7GB 和 32.2GB，注意代码中的斜体字部分。

接着在节点 node-rac1 上对共享磁盘进行分区操作。分区完成后的磁盘信息如下：

```
[root@node-rac1 /]# fdisk -l
Disk /dev/sda: 21.4 GB, 21474836480 bytes
```

```
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14         140     1020127+  82  Linux swap / Solaris
/dev/sda3             141        2610    19840275   83  Linux

Disk /dev/sdb: 10.7 GB, 10737418240 bytes
64 heads, 32 sectors/track, 10240 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1       10240    10485744    5  Extended
/dev/sdb5               1        3907     4000736   83  Linux
/dev/sdb6            3908        7814     4000752   83  Linux
/dev/sdb7            7815        8292      489456   83  Linux
/dev/sdb8            8293        8770      489456   83  Linux
/dev/sdb9            8771       10240     1505264   83  Linux

Disk /dev/sdc: 32.2 GB, 32212254720 bytes
64 heads, 32 sectors/track, 30720 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1               1       30720    31457264    5  Extended
/dev/sdc5               1        2099     2149344   83  Linux
/dev/sdc6            2100        6010     4004848   83  Linux
/dev/sdc7            6011        9921     4004848   83  Linux
/dev/sdc8            9922       19688    10001392   83  Linux
/dev/sdc9           19689       29455    10001392   83  Linux
```

在这个操作中，分别对两块共享磁盘划分了 5 个分区，每个分区的用途如表 13-8 所示。

表 13-8　共享磁盘分区的用途

磁盘分区标识	磁盘大小	磁盘用途
/dev/sdb5	4096MB	OCR 磁盘，即集群注册磁盘，用于存储集群配置信息，Oracle 要求磁盘最小空间为 256MB
/dev/sdb6	4096MB	OCR 镜像磁盘
/dev/sdb7	500MB	voting disk（表决磁盘），用于记录集群节点信息，Oracle 要求磁盘最小空间为 256MB
/dev/sdb8	500MB	表决磁盘镜像磁盘
/dev/sdb9	500MB	表决磁盘镜像磁盘
/dev/sdc5	2048MB	Oracle 闪回数据存放分区
/dev/sdc6	4096MB	Oracle 归档日志存储分区
/dev/sdc7	4096MB	Oracle 归档日志存储镜像分区
/dev/sdc8	10240MB	RAC 数据库数据存放分区
/dev/sdc9	10240MB	RAC 数据库数据存放镜像分区

全部操作执行完毕后重启RAC数据库的两个节点。

3. 建立和配置RAW设备

关于RAW设备的使用，从CentOS 4版本到CentOS 5发生了很大变化，在CentOS 4以前版本中，可以通过/etc/sysconfig/rawdevices和/etc/init.d/rawdevices文件来创建和配置RAW设备，而在CentOS 5以后，RAW设备必须通过udev来管理，并且raw命令的位置从/usr/bin/raw变为/bin/raw，这使CentOS在安全方面有了不少的改进，但是仍然兼容之前的配置方式。因此在CentOS 5版本中配置RAW的方法有以下两种：

- 手动建立/etc/sysconfig/rawdevices文件，然后从其他操作系统上复制/etc/init.d/rawdevices文件到本机，修改/etc/init.d/rawdevices文件中raw命令的路径，然后就可以通过/etc/init.d/rawdevices来启动和关闭RAW文件了。
- 通过udev来管理RAW设备，添加RAW设备对应的配置文件为/etc/udev/rules.d/60-raw.rules。

这里采用第二种方式来建立和配置RAW设备。首先修改/etc/udev/rules.d/60-raw.rules文件，修改后的内容如下：

```
ACTION=="add",KERNEL=="sdb5",RUN+="/bin/raw /dev/raw/raw1 %N"
ACTION=="add",KERNEL=="sdb6",RUN+="/bin/raw /dev/raw/raw2 %N"
ACTION=="add",KERNEL=="sdb7",RUN+="/bin/raw /dev/raw/raw3 %N"
ACTION=="add",KERNEL=="sdb8",RUN+="/bin/raw /dev/raw/raw4 %N"
ACTION=="add",KERNEL=="sdb9",RUN+="/bin/raw /dev/raw/raw5 %N"
ACTION=="add",KERNEL=="sdc5",RUN+="/bin/raw /dev/raw/raw6 %N"
ACTION=="add",KERNEL=="sdc6",RUN+="/bin/raw /dev/raw/raw7 %N"
ACTION=="add",KERNEL=="sdc7",RUN+="/bin/raw /dev/raw/raw8 %N"
ACTION=="add",KERNEL=="sdc8",RUN+="/bin/raw /dev/raw/raw9 %N"
ACTION=="add",KERNEL=="sdc9",RUN+="/bin/raw /dev/raw/raw10 %N"
KERNEL=="raw1", OWNER="oracle", GROUP="oinstall", MODE="644"
KERNEL=="raw2", OWNER="oracle", GROUP="oinstall", MODE="644"
KERNEL=="raw3", OWNER="oracle", GROUP="oinstall", MODE="660"
KERNEL=="raw4", OWNER="oracle", GROUP="oinstall", MODE="660"
KERNEL=="raw5", OWNER="oracle", GROUP="oinstall", MODE="660"
KERNEL=="raw6", OWNER="oracle", GROUP="oinstall", MODE="660"
KERNEL=="raw7", OWNER="oracle", GROUP="oinstall", MODE="660"
KERNEL=="raw8", OWNER="oracle", GROUP="oinstall", MODE="660"
KERNEL=="raw9", OWNER="oracle", GROUP="oinstall", MODE="660"
KERNEL=="raw10", OWNER="oracle", GROUP="oinstall", MODE="660"
```

然后启动udev服务生成RAW设备：

```
[root@node-rac1 /]# start_udev
Starting udev:                                             [  OK  ]
```

接着验证一下RAW设备是否生成：

```
[root@node-rac1 /]#  ll /dev/raw/raw*
crw-r--r-- 1 oracle oinstall 162,  1 Aug 27 00:13 /dev/raw/raw1
```

```
crw-rw---- 1 oracle oinstall 162, 10 Aug 27 00:13 /dev/raw/raw10
crw-r--r-- 1 oracle oinstall 162,  2 Aug 27 00:13 /dev/raw/raw2
crw-rw---- 1 oracle oinstall 162,  3 Aug 27 00:13 /dev/raw/raw3
crw-rw---- 1 oracle oinstall 162,  4 Aug 27 00:13 /dev/raw/raw4
crw-rw---- 1 oracle oinstall 162,  5 Aug 27 00:13 /dev/raw/raw5
crw-rw---- 1 oracle oinstall 162,  6 Aug 27 00:13 /dev/raw/raw6
crw-rw---- 1 oracle oinstall 162,  7 Aug 27 00:13 /dev/raw/raw7
crw-rw---- 1 oracle oinstall 162,  8 Aug 27 00:13 /dev/raw/raw8
crw-rw---- 1 oracle oinstall 162,  9 Aug 27 00:13 /dev/raw/raw9
```

从输出可以看出，RAW 设备已经生成，并且相关权限也自动加载。

13.4.13　安装 Oracle Clusterware

（1）解压软件包

这里假定数据库所有软件放在 /rac 目录下，首先需要解压 Oracle 的两个软件包。操作如下：

```
[root@node-rac1 rac]#ls
linux_11gR1_database_1013.zip    linux_x86_11gR1_clusterware.zip
[root@node-rac1 rac]#unzip linux_x86_11gR1_clusterware.zip
[root@node-rac1 rac]#unzip linux_11gR1_database_1013.zip
```

（2）安装补丁包

解压完成，安装补丁包。操作如下：

```
[root@node-rac1 rac]#/rac/clusterware/rpm
[root@node-rac1 rac]#rpm -Uvh cvuqdisk-1.0.1-1.rpm
```

在节点 node-rac1 安装完成后，继续在 node-rac2 进行补丁安装。

（3）验证安装环境

在开始安装之前，可以使用 Oracle 自己的一个检测脚本验证系统环境是否可以进行安装。以 oracle 身份登录系统，执行如下命令：

```
[oracle@node-rac1 ~]$/rac/clusterware/runcluvfy.sh stage -pre crsinst -n node-
    rac1,node-rac2 -verbose
```

（4）开始安装

以 oracle 用户登录 Linux 图形界面，然后执行安装脚本：

```
[oracle@node-rac1 ~]$ cd /rac/clusterware/
[oracle@node-rac1 clusterware]$ ./runInstaller
```

接着就会弹出图形安装向导界面，如图 13-4 所示。

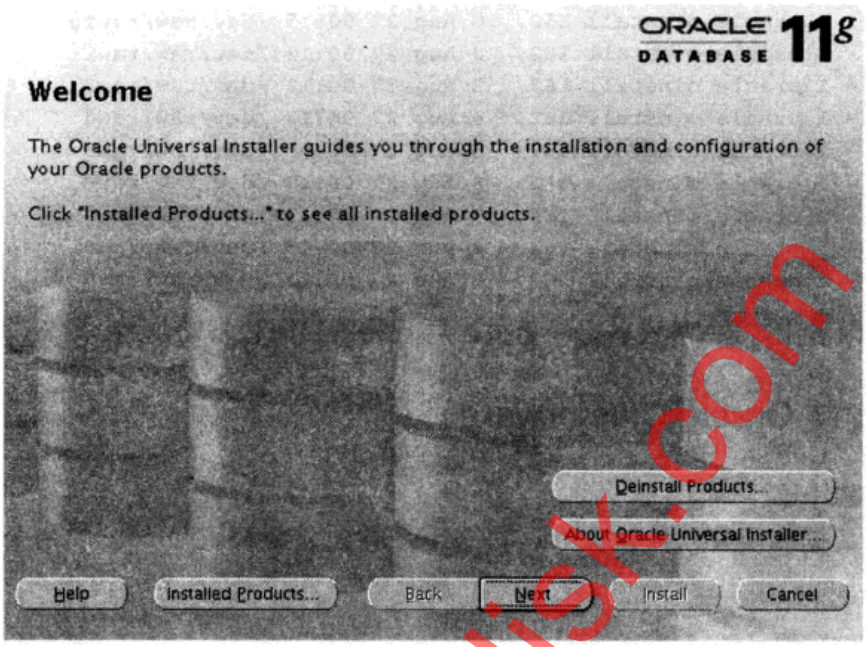

图 13-4 Oracle 安装向导

单击“Next”按钮进入下一步，如图 13-5 所示。

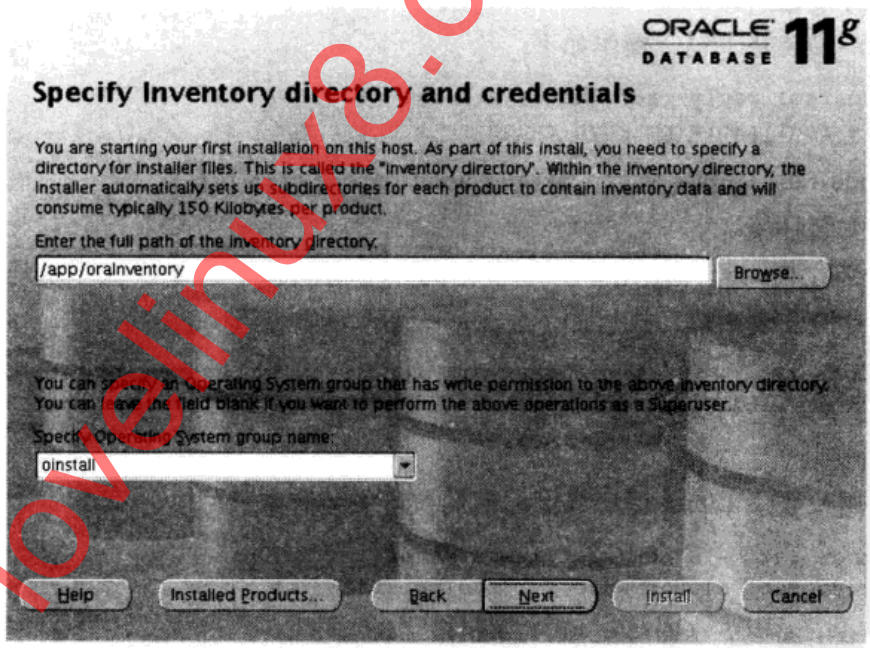

图 13-5 设置安装清单目录

这里将安装清单目录指定到 /app 下，务必确认 oracle 用户拥有对 /app 目录的读取和写入权限。单击“Next”按钮进入下一步，如图 13-6 所示。

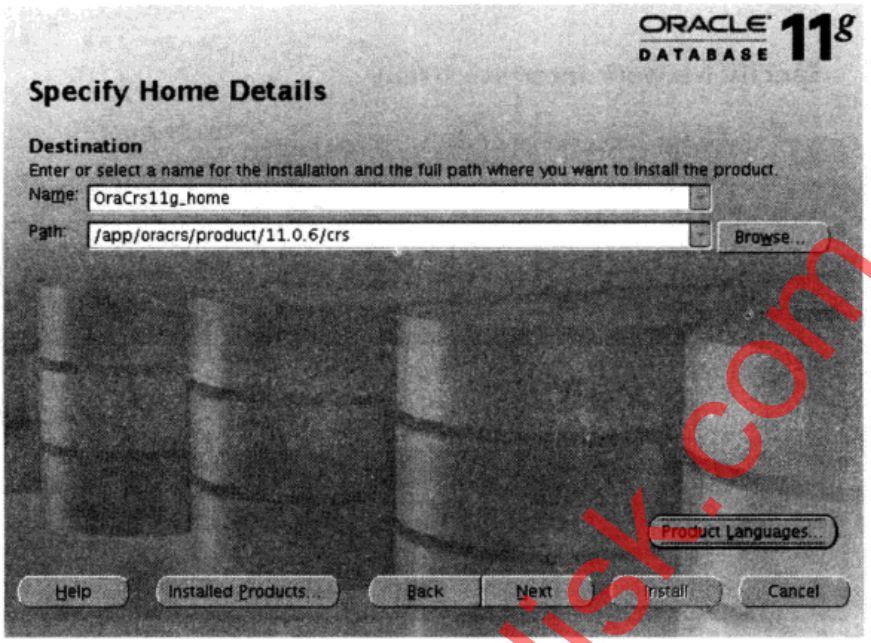

图 13-6　指定安装主目录信息

输入 CRS 的名称和安装路径，单击“Next”按钮进入下一步，如图 13-7 所示。

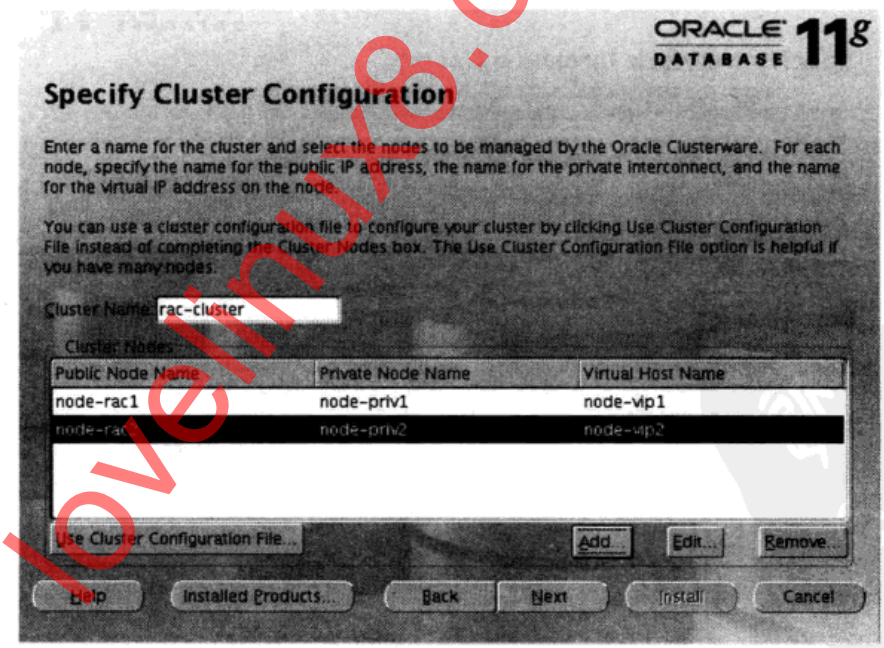

图 13-7　修改集群节点信息

按照配置的节点信息，将集群的两个节点添加进来，填写相应节点信息即可。填写完成后，单击“Next”按钮进入下一步，如图 13-8 所示。

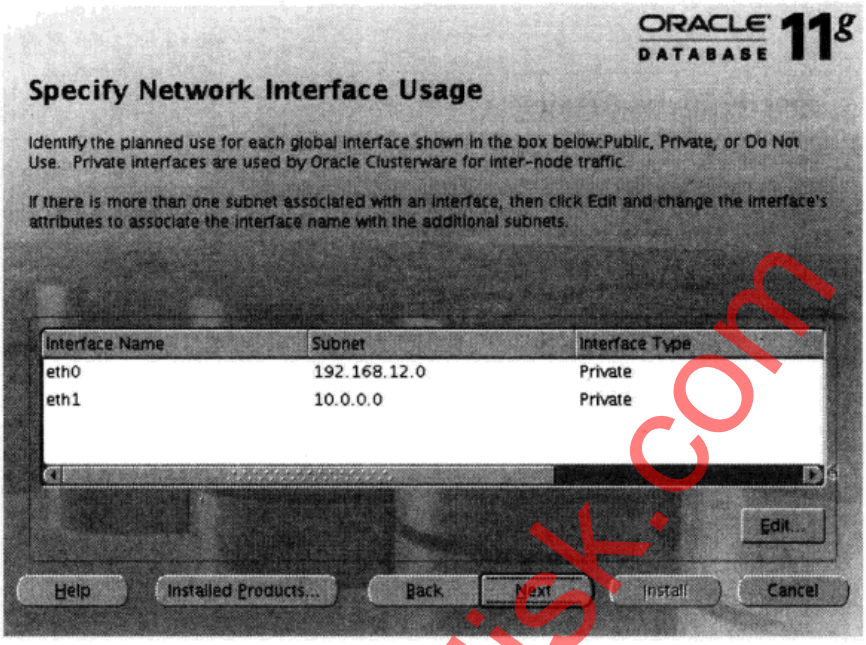

图 13-8 指定网络接口类型

单击“Edit”按钮，将公网 eth0 网段设为“Public”，然后单击“OK”按钮，如图 13-9 所示。

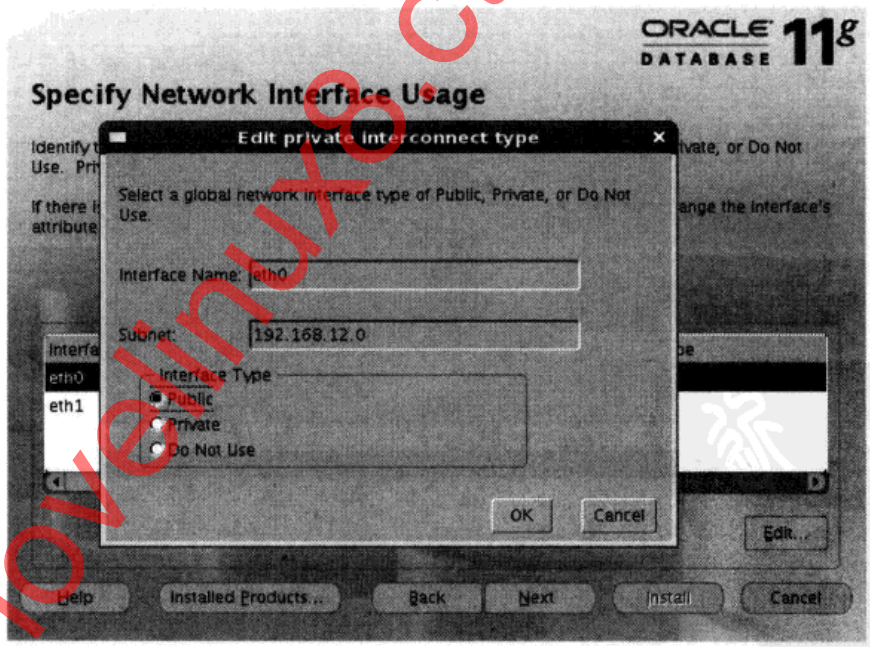

图 13-9 指定网络接口 eth0 为公用网络

按照预先设定好的存储方式，指定 OCR 的存储位置，如图 13-10 所示。

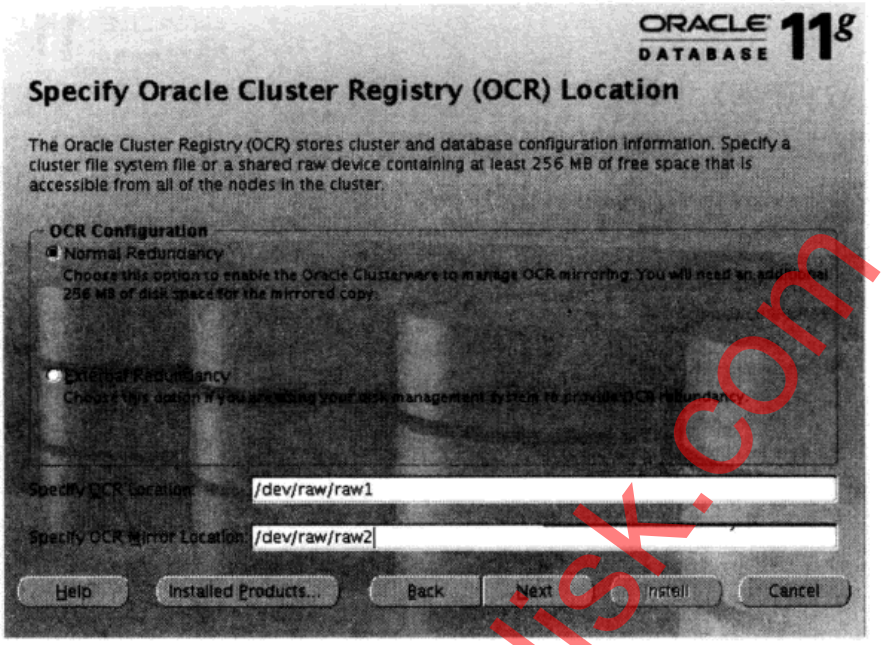

图 13-10　指定 OCR 的位置

设置表决磁盘使用 RAW 设备，并添加对应的设备标识，如图 13-11 所示。

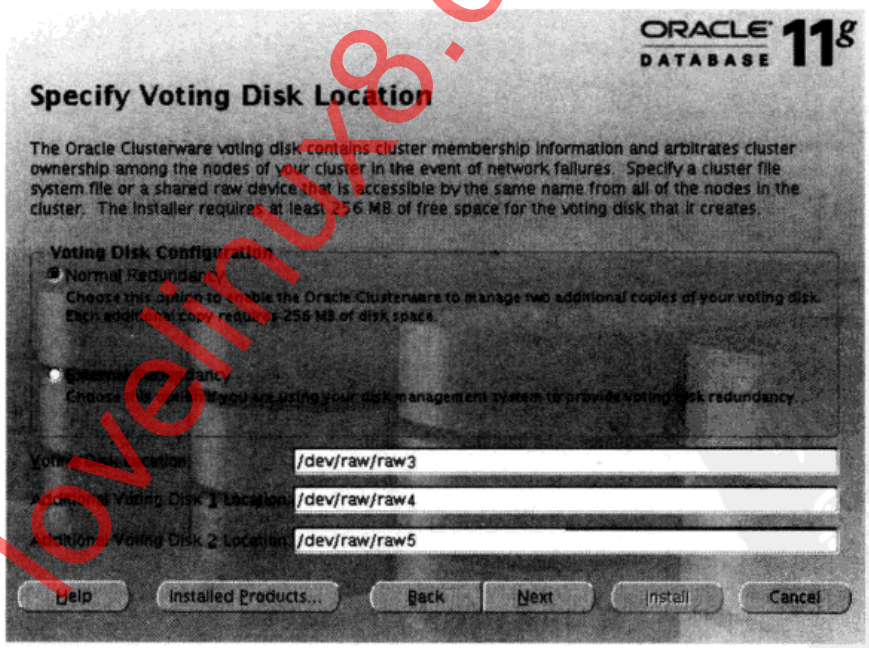

图 13-11　指定表决磁盘的位置

图 13-12 是安装信息预览，单击“install”按钮开始安装 ClusterWare，如图 13-13 所示。

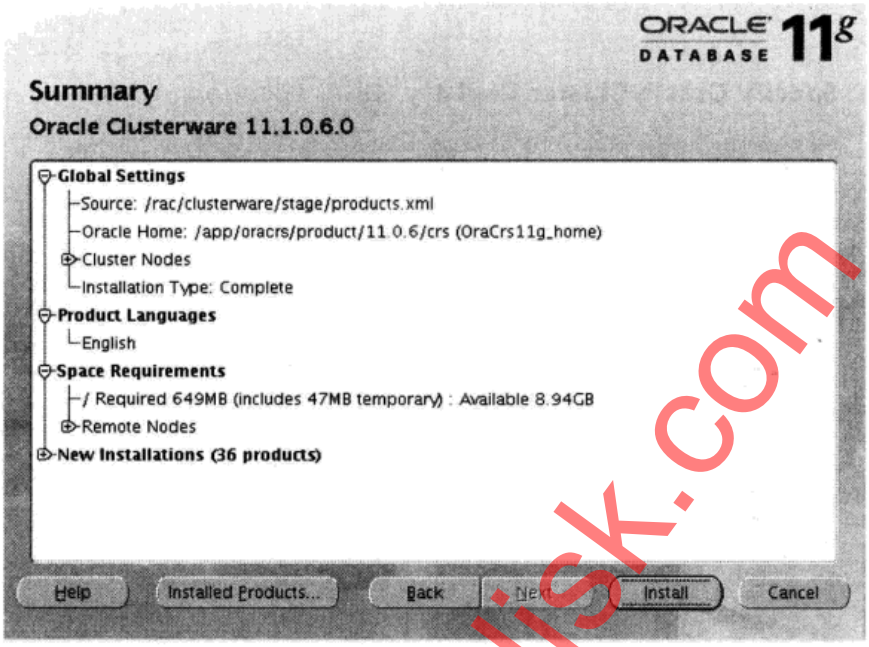

图 13-12 安装信息预览

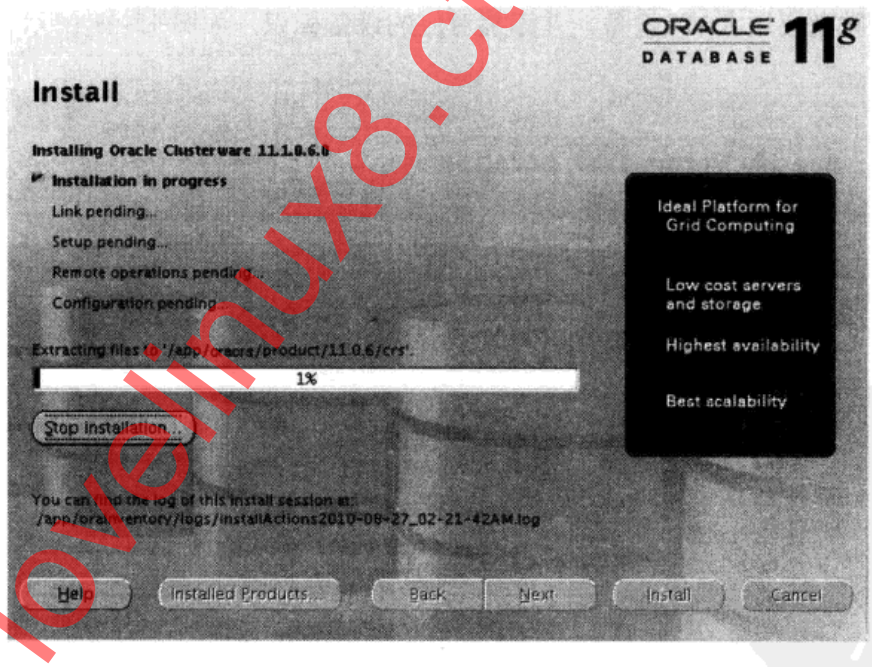

图 13-13 开始安装 ClusterWare

ClusterWare 的安装非常快，在等待了几分钟后，安装就完成了，如图 13-14 所示。

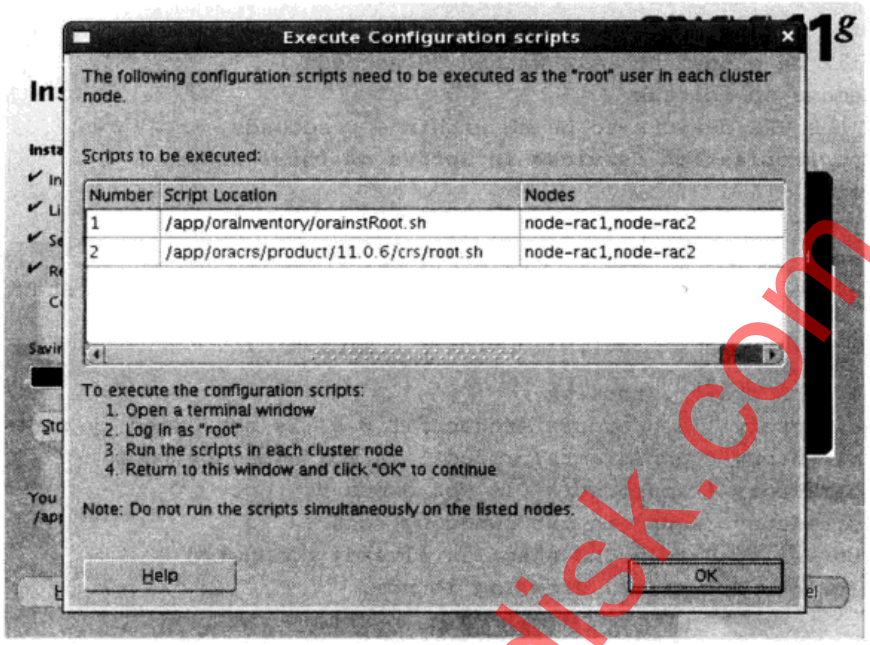

图 13-14　安装完成

根据安装向导提示，以 root 用户身份在所有节点上分别执行图 13-14 中的两个脚本，要一个节点一个节点地执行。在 node-rac1 上执行 root.sh 时的信息如下：

```
[root@node-rac1 crs]# ./root.sh
WARNING: directory '/app/oracrs/product/11.0.6' is not owned by root
WARNING: directory '/app/oracrs/product' is not owned by root
WARNING: directory '/app/oracrs' is not owned by root
WARNING: directory '/app' is not owned by root
Checking to see if Oracle CRS stack is already configured
/etc/oracle does not exist. Creating it now.

Setting the permissions on OCR backup directory
Setting up Network socket directories
Oracle Cluster Registry configuration upgraded successfully
The directory '/app/oracrs/product/11.0.6' is not owned by root. Changing owner to root
The directory '/app/oracrs/product' is not owned by root. Changing owner to root
The directory '/app/oracrs' is not owned by root. Changing owner to root
The directory '/app' is not owned by root. Changing owner to root
Successfully accumulated necessary OCR keys.
Using ports: CSS=49895 CRS=49896 EVMC=49898 and EVMR=49897.
node <nodenumber>: <nodename> <private interconnect name> <hostname>
node 1: node-rac1 node-priv1 node-rac1
node 2: node-rac2 node-priv2 node-rac2
Creating OCR keys for user 'root', privgrp 'root'..
Operation successful.
Now formatting voting device: /dev/raw/raw3
 Now formatting voting device: /dev/raw/raw4
```

```
Now formatting voting device: /dev/raw/raw5
 Format of 3 voting devices complete.
Startup will be queued to init within 30 seconds.
Adding daemons to inittab
Expecting the CRS daemons to be up within 600 seconds.
Cluster Synchronization Services is active on these nodes.
   node-rac1
Cluster Synchronization Services is inactive on these nodes.
   node-rac2
Local node checking complete. Run root.sh on remaining nodes to start CRS daemons.
```

然后，继续在 node-rac2 上执行 root.sh 脚本。输出如下：

```
[root@node-rac2 crs]# ./root.sh
WARNING: directory '/app/oracrs/product/11.0.6' is not owned by root
WARNING: directory '/app/oracrs/product' is not owned by root
WARNING: directory '/app/oracrs' is not owned by root
WARNING: directory '/app' is not owned by root
Checking to see if Oracle CRS stack is already configured
/etc/oracle does not exist. Creating it now.

Setting the permissions on OCR backup directory
Setting up Network socket directories
Oracle Cluster Registry configuration upgraded successfully
The directory '/app/oracrs/product/11.0.6' is not owned by root. Changing owner to root
The directory '/app/oracrs/product' is not owned by root. Changing owner to root
The directory '/app/oracrs' is not owned by root. Changing owner to root
The directory '/app' is not owned by root. Changing owner to root
clscfg: EXISTING configuration version 4 detected.
clscfg: version 4 is 11 Release 1.
Successfully accumulated necessary OCR keys.
Using ports: CSS=49895 CRS=49896 EVMC=49898 and EVMR=49897.
node <nodenumber>: <nodename> <private interconnect name> <hostname>
node 1: node-rac1 node-priv1 node-rac1
node 2: node-rac2 node-priv2 node-rac2
clscfg: Arguments check out successfully.

NO KEYS WERE WRITTEN. Supply -force parameter to override.
-force is destructive and will destroy any previous cluster
configuration.
Oracle Cluster Registry for cluster has already been initialized
Startup will be queued to init within 30 seconds.
Adding daemons to inittab
 Expecting the CRS daemons to be up within 600 seconds.
 Cluster Synchronization Services is active on these nodes.
   node-rac1
   node-rac2
Cluster Synchronization Services is active on all the nodes.
Waiting for the Oracle CRSD and EVMD to start
Oracle CRS stack installed and running under init(1M)
```

```
Running vipca(silent) for configuring nodeapps

Creating VIP application resource on (2) nodes . ..
Creating GSD application resource on (2) nodes ...
Creating ONS application resource on (2) nodes . ..
Starting VIP application resource on (2) nodes ...
Starting GSD application resource on (2) nodes...
Starting ONS application resource on (2) nodes ...
Done.
```

注意上面的输出中斜体字部分的内容。通过观察日志输出，可以很清楚地了解 Oracle ClusterWare 的内部运作机制。

在两个节点上执行完毕配置脚本后，单击“OK”按钮，进入 Oracle 组件配置和启动界面，如图 13-15 所示。

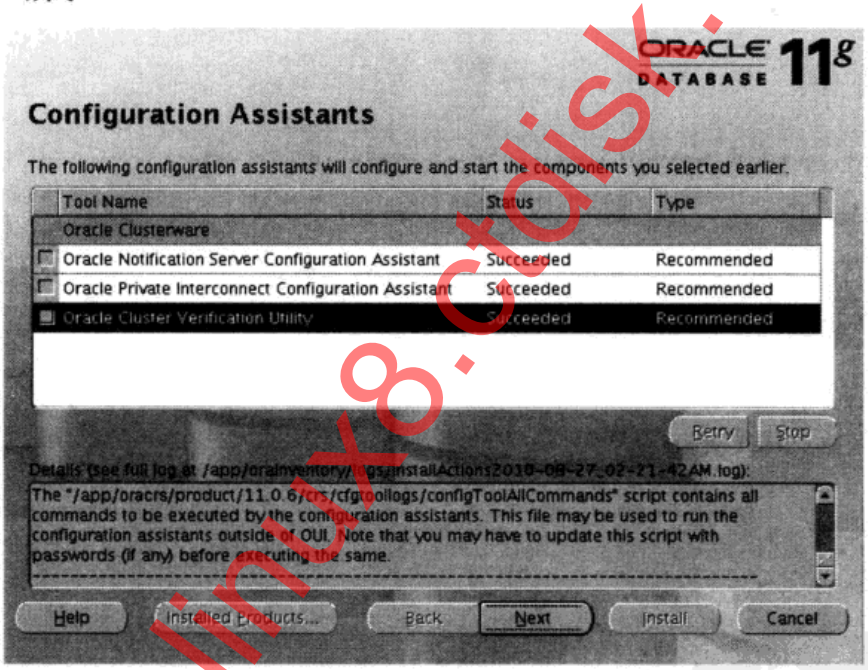

图 13-15　配置并启动 Oracle 组件

所有组件成功配置并启动后，Oracle ClusterWare 安装结束。

13.4.14　安装 Oracle 数据库

在 Oracle ClusterWare 安装成功后，开始进入 Oracle 数据库的安装。以 oracle 用户身份登录到任意一个集群节点，执行如下命令开始安装：

```
[oracle@node-rac2 rac]$ /rac/database/runInstaller
```

接着会弹出图形安装向导界面，如图 13-4 所示。单击“Next”按钮进入下一步，如图 13-16 所示。

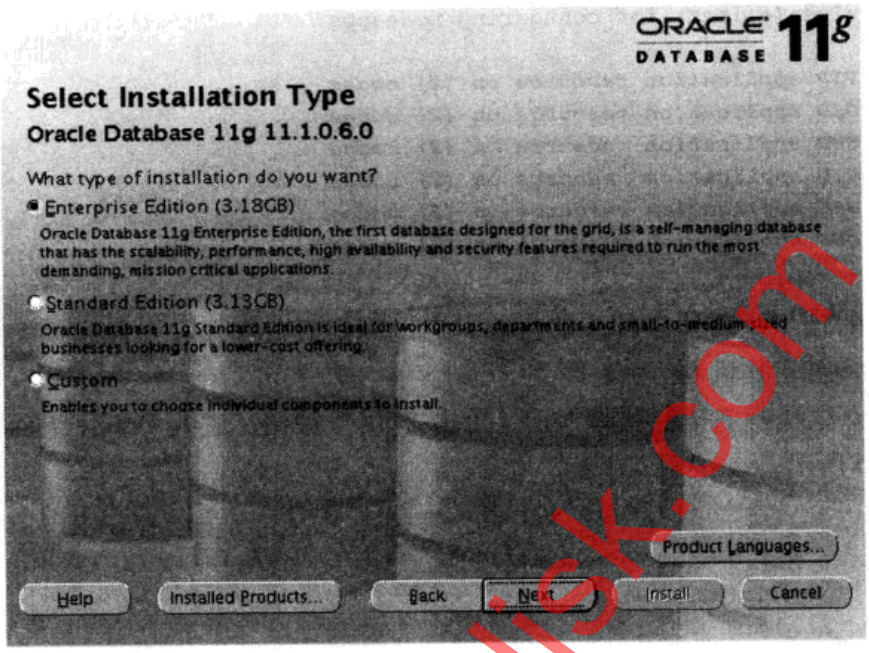

图 13-16 选择 Oracle 安装类型

这里选择“Enterprise Edition”，即企业版本。单击“Next”按钮进入下一步，如图 13-17 所示。

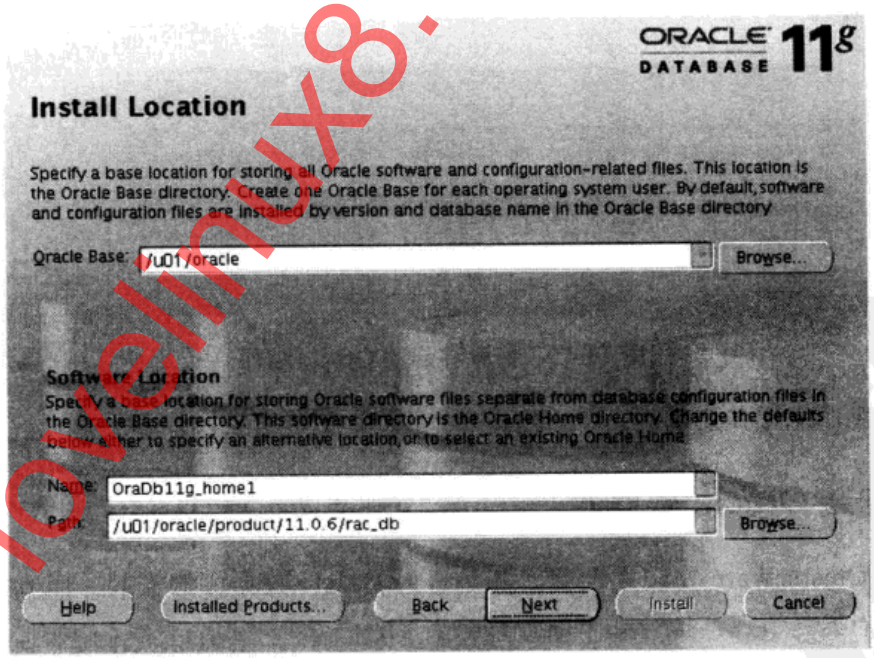

图 13-17 指定 Oracle 安装主目录

在指定 Oracle 主目录和程序的安装路径后单击“Next”按钮进入下一步，如图 13-18 所示。

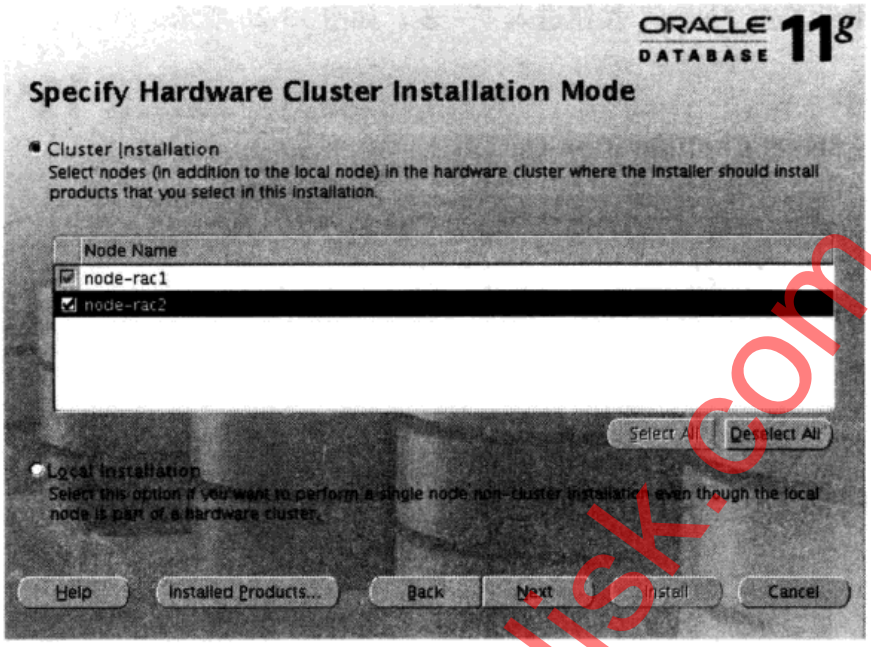

图 13-18　指定集群安装模式

这里选用集群安装方式，选择集群所有节点。单击“Next”按钮进入下一步，如图 13-19 所示。

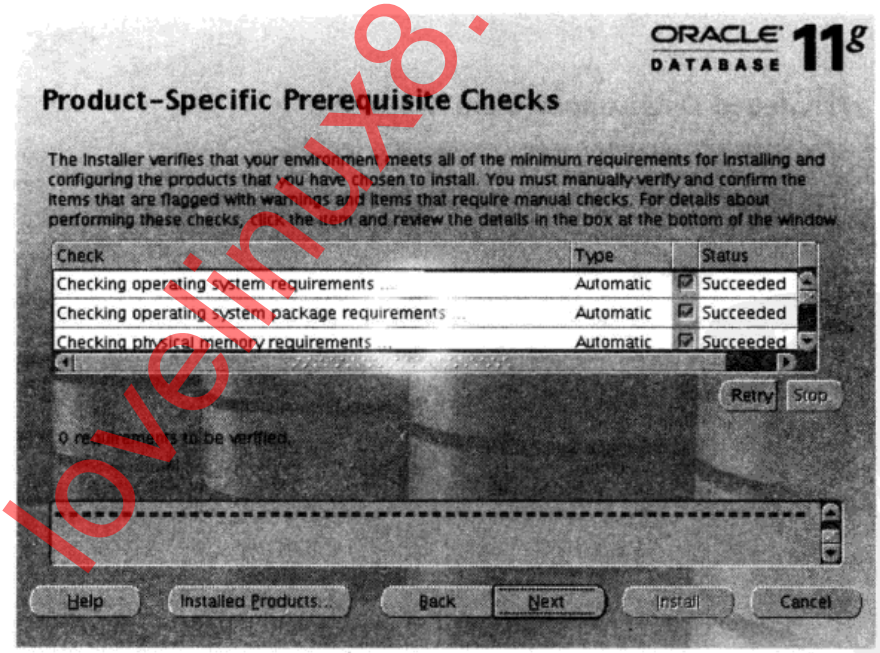

图 13-19　检查 Oracle 安装环境

这里检查系统环境是否满足 Oracle 的安装需求，如果检测没有通过，根据相关提示进

行修改即可。继续单击“Next”按钮进入下一步，如图 13-20 所示。

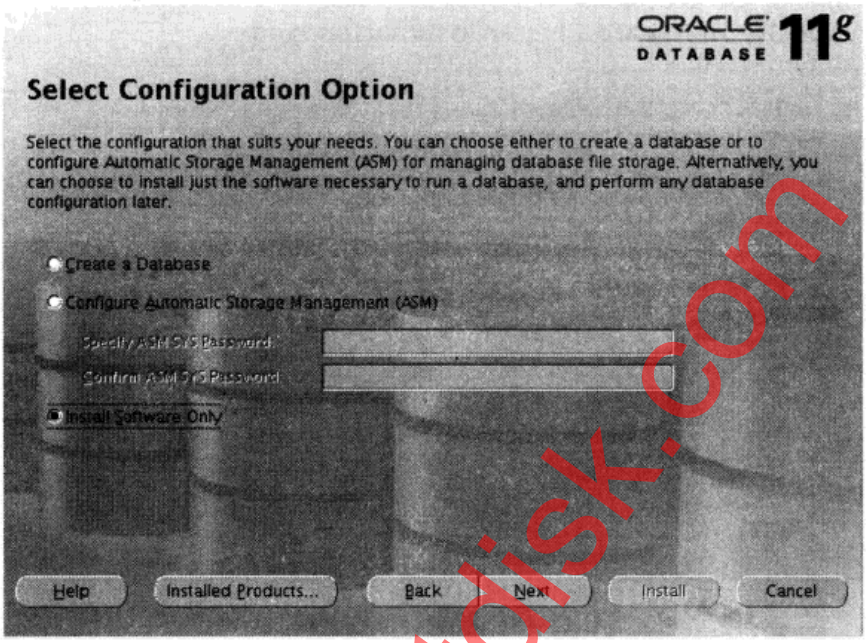

图 13-20 选择配置选项

这里选择“Install Software Only”。单击“Next”按钮进入下一步，如图 13-21 所示。

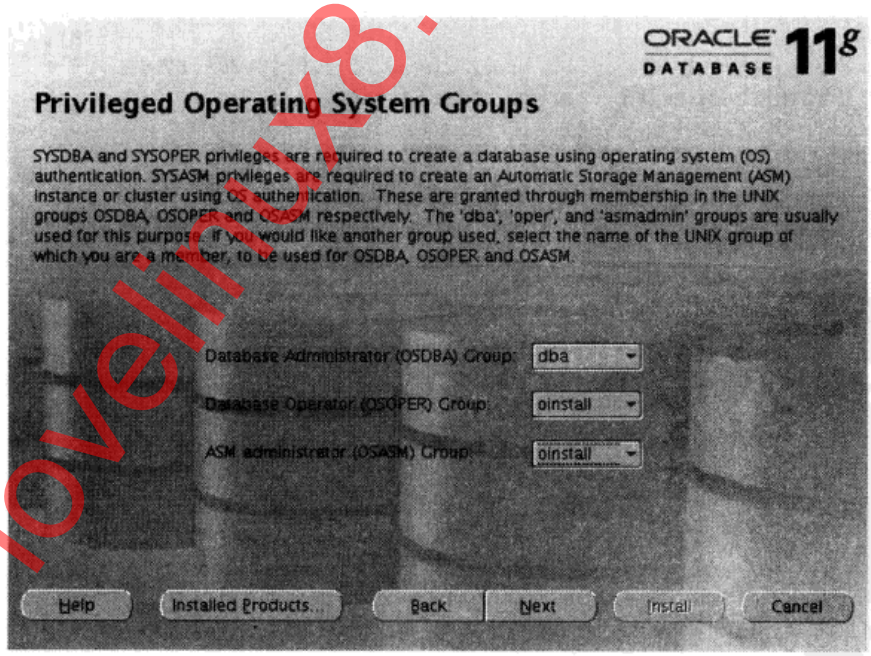

图 13-21 选择安装的系统组

这里将 OSDBA 组设置为“dba”，将 OSOPOER 和 OSASM 组都设置为“oinstall”。单

击“Next”按钮进入下一步，如图 13-22 所示。

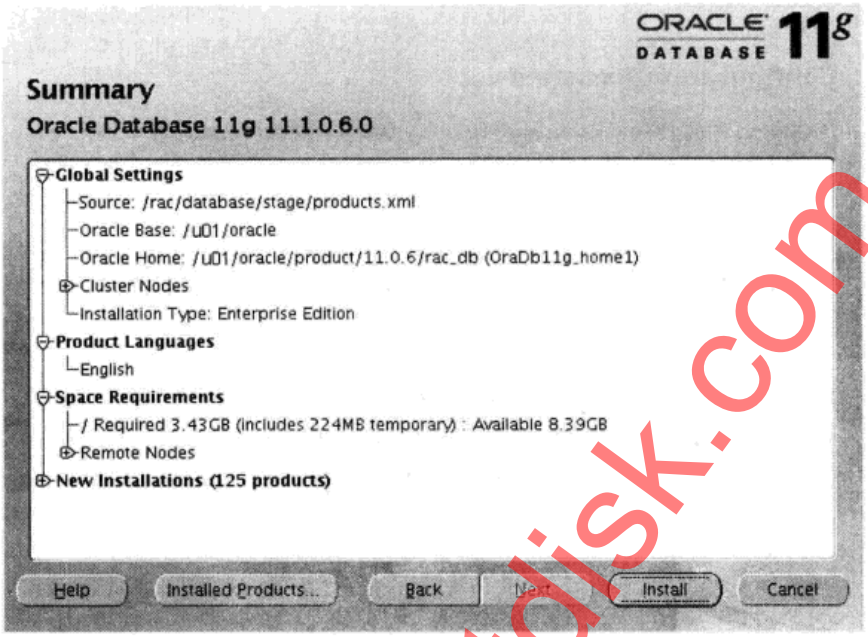

图 13-22　安装概要一览

图 13-22 显示了安装 Oracle 数据库的相关信息，单击“Install”按钮开始安装，如图 13-23 所示。

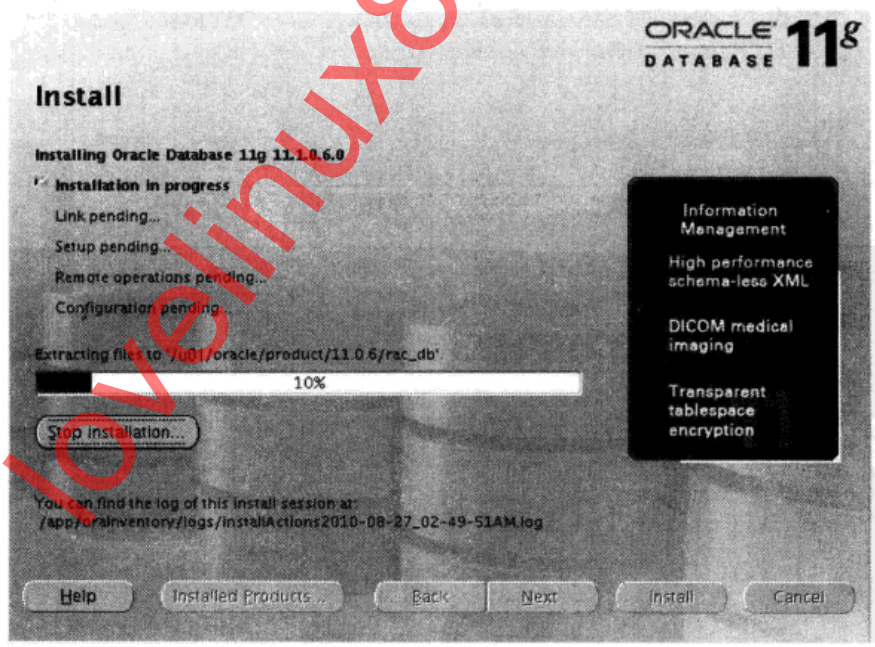

图 13-23　安装 Oracle

安装完成后自动进入 Oracle 组件配置和启动界面，如图 13-24 所示。

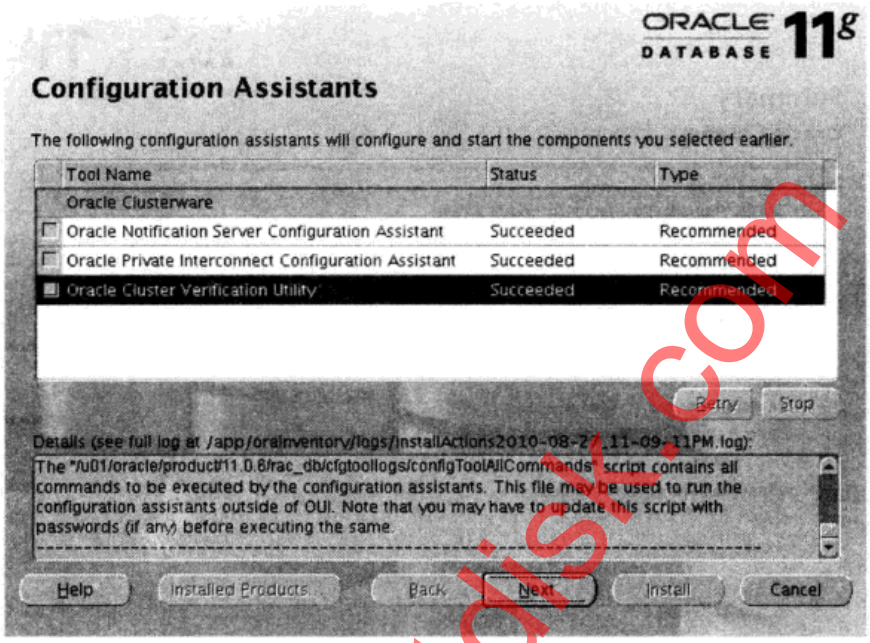

图 13-24 配置和启动 Oracle 组件

所有组件成功启动后，自动进入如图 13-25 所示界面。

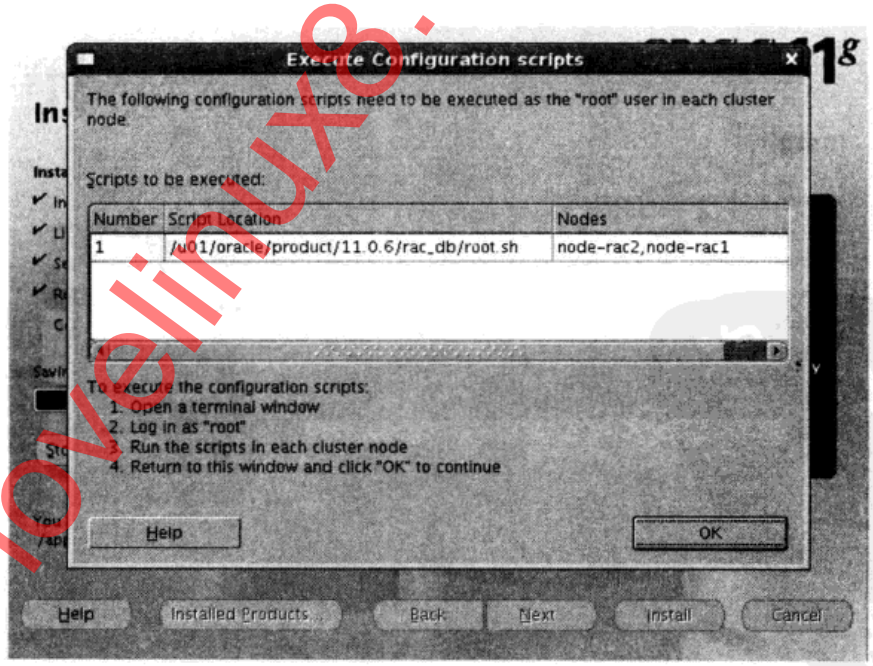

图 13-25 执行配置脚本

以 root 用户身份依次在两个节点上执行 root.sh 配置脚本，执行完毕，单击“OK”按钮

完成 Oracle 数据库软件的安装。

13.4.15　配置 Oracle Net

以 oracle 用户身份登录到集群任意节点的图形界面，然后执行 netca 命令配置 Oracle Net，如图 13-26 所示。

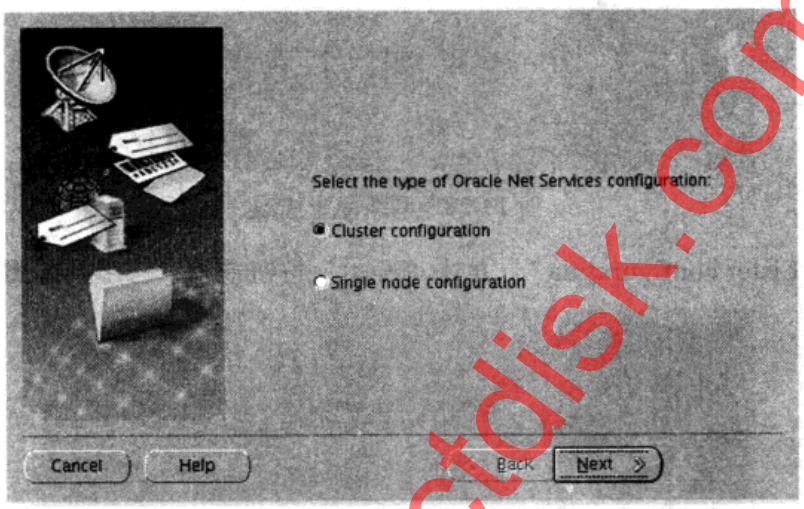

图 13-26　选择 Net Services 配置类型

这里选择“Cluster configuration”。单击“Next”按钮进入下一步，如图 13-27 所示。

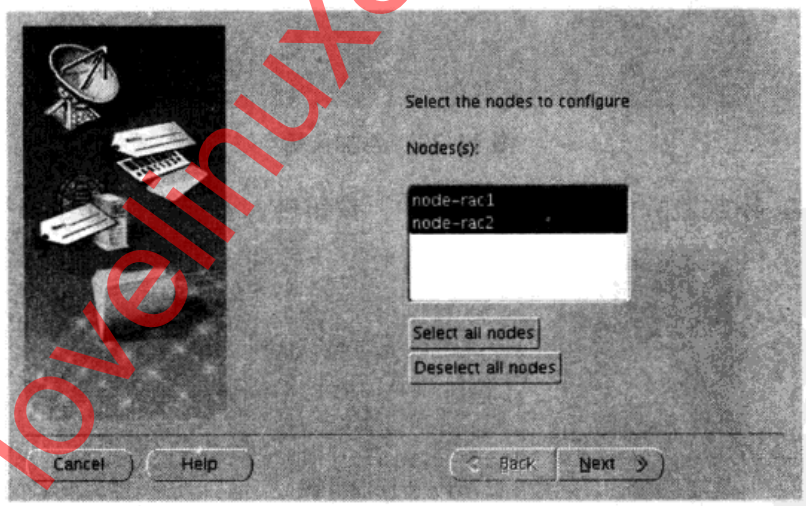

图 13-27　选择要配置的集群节点

这里选择集群所有节点。单击“Next”按钮进入下一步，如图 13-28 所示。

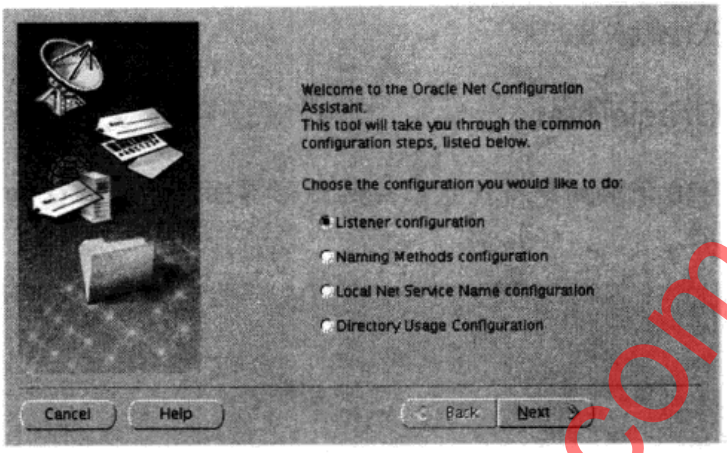

图 13-28 网络配置选项

这里选择“Lister configuration”。单击“Next”按钮进入下一步，如图 13-29 所示。

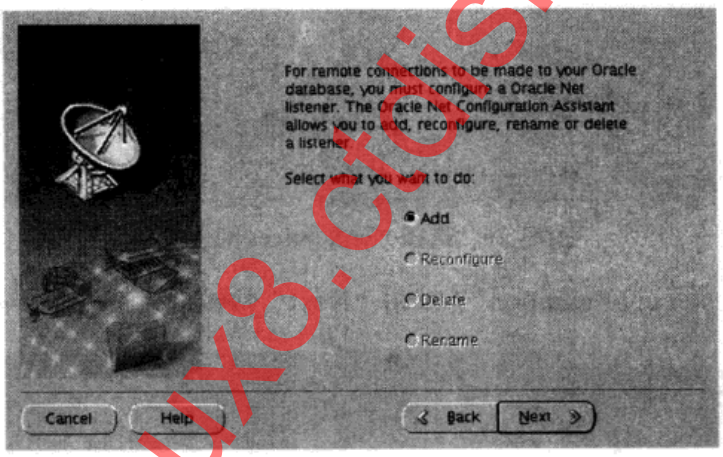

图 13-29 添加监听

选择“Add”。添加监听设置。单击“Next”按钮进入下一步，如图 13-30 所示。

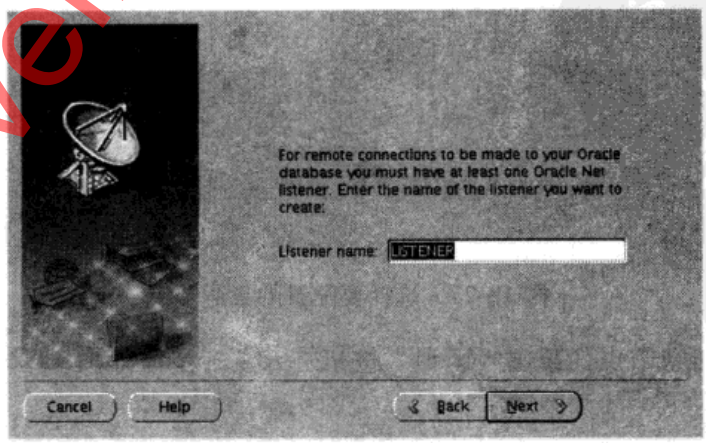

图 13-30 设置监听名称

这里选择默认的监听名称“LISTENER”。单击“Next”按钮进入下一步，如图 13-31 所示。

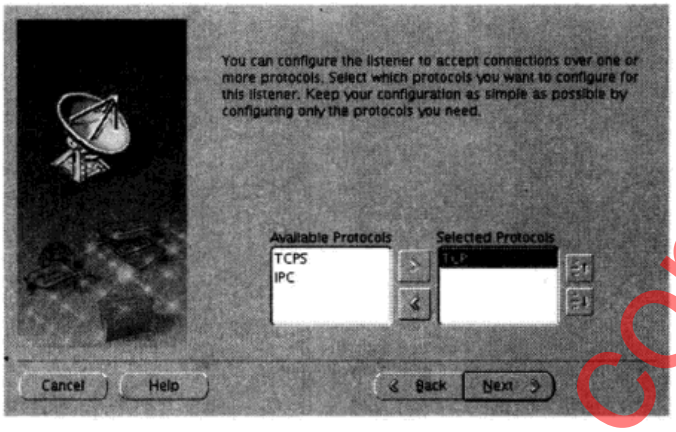

图 13-31　选择可用协议

这里选择 TCP 协议。单击“Next”按钮进入下一步，如图 13-32 所示。

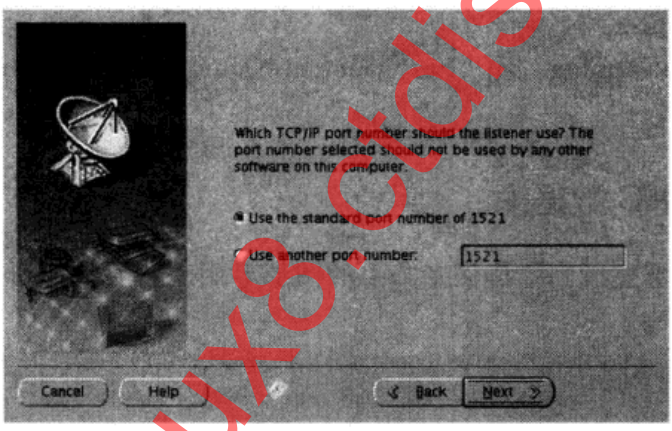

图 13-32　选择监听端口

这里选择默认的 1521 端口。单击“Next”按钮进入下一步，进而完成监听的创建。接着再配置一个命名方法，如图 13-33 所示。

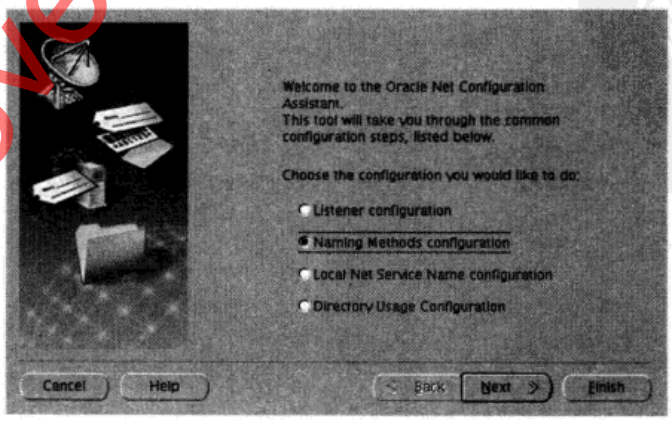

图 13-33　配置命名方法

单击“Next”按钮进入下一步，如图 13-34 所示。

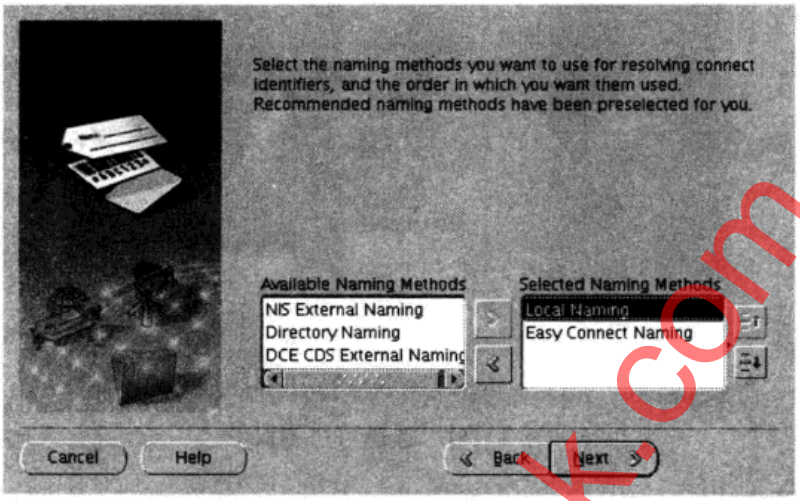

图 13-34 选择可用命名方法

这里选择“Local Naming”和“Easy Content Naming”。单击“Next”按钮完成命名方法的配置。

13.4.16 创建 RAC 数据库

以 oracle 用户身份登录到集群任意节点的图形界面，然后执行 dbca 命令，会弹出如图 13-35 所示图形界面。

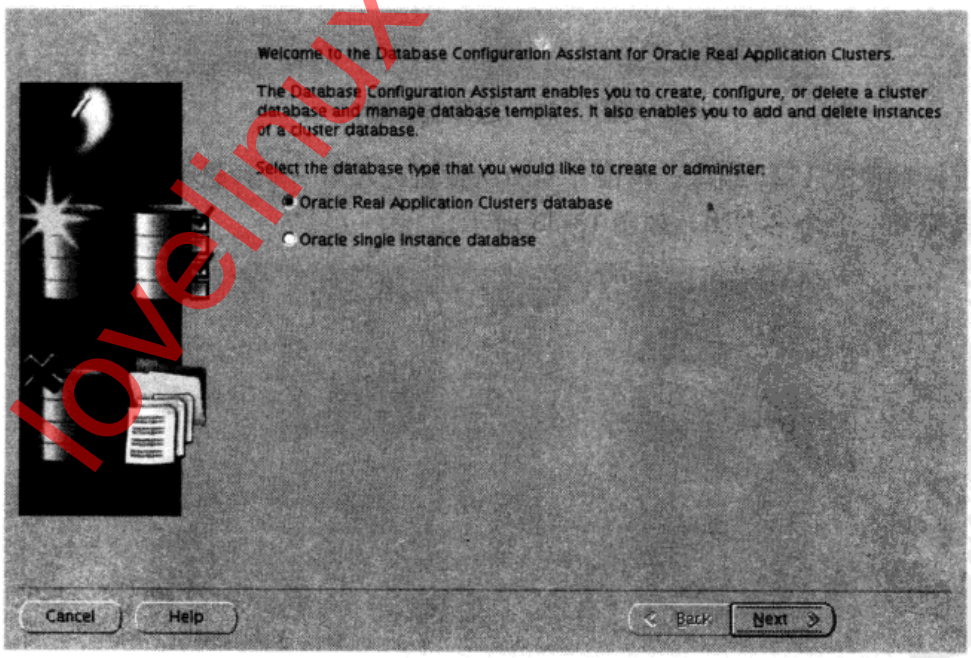

图 13-35 选择创建数据库类型

这里选择“Oracle Real Application Clusters database”，单击“Next”按钮进入下一步，如图 13-36 所示。

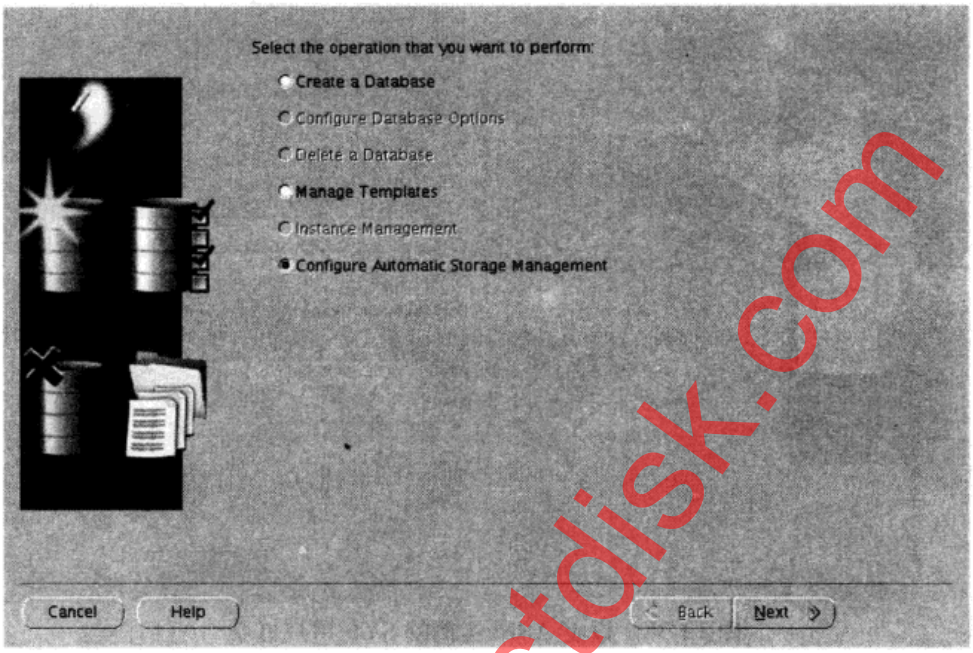

图 13-36　选择操作类型

这里选择“Configure Automatic Storage Management”，即设置 ASM。单击“Next”按钮进入下一步，如图 13-37 所示。

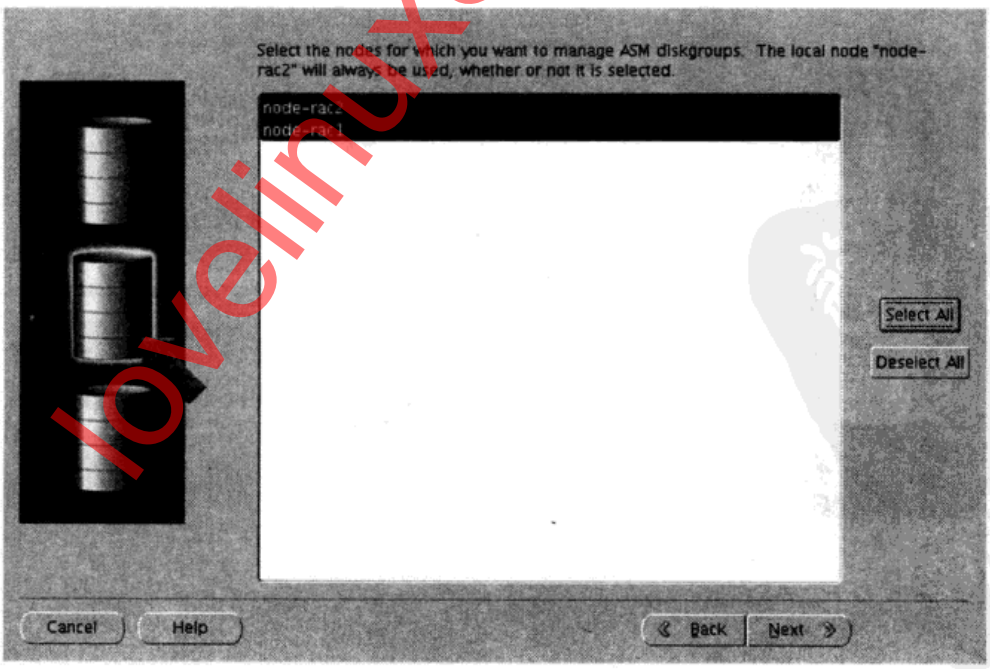

图 13-37　选择集群节点

这里选择集群所有节点。单击“Next”按钮进入下一步，如图 13-38 所示。

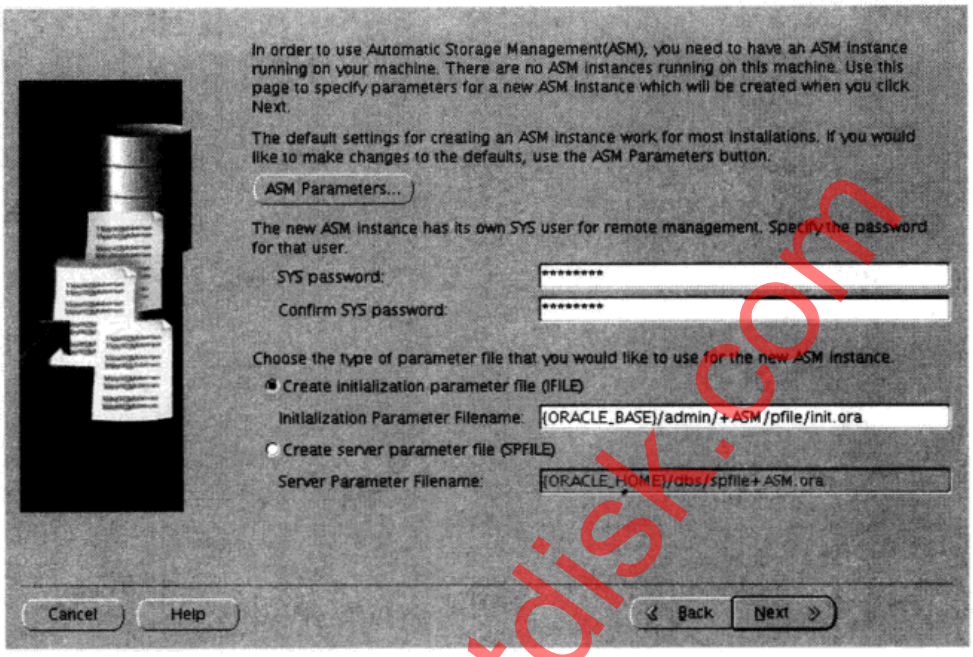

图 13-38 设定 ASM 实例的 SYS 用户口令

这里设置一个 ASM 实例的 SYS 用户口令，然后指定初始化参数文件的类型。单击“Next”按钮进入下一步，如图 13-39 所示。

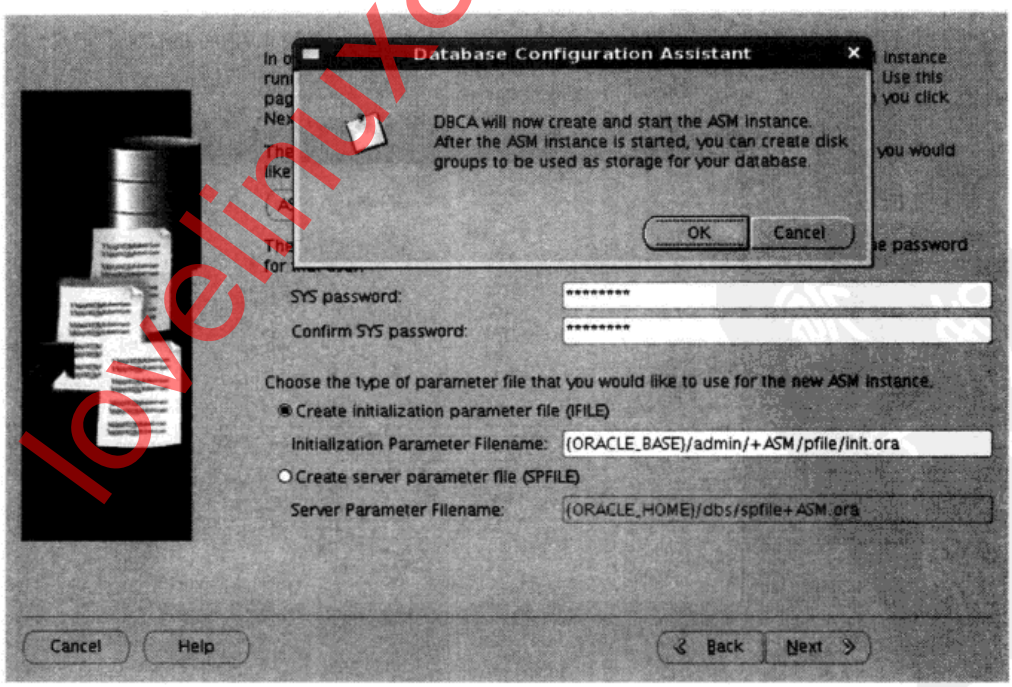

图 13-39 准备创建 ASM 实例

此时，安装向导将提示开始创建并启动 ASM 实例。单击“OK”按钮，开始创建实例，如图 13-40 所示。

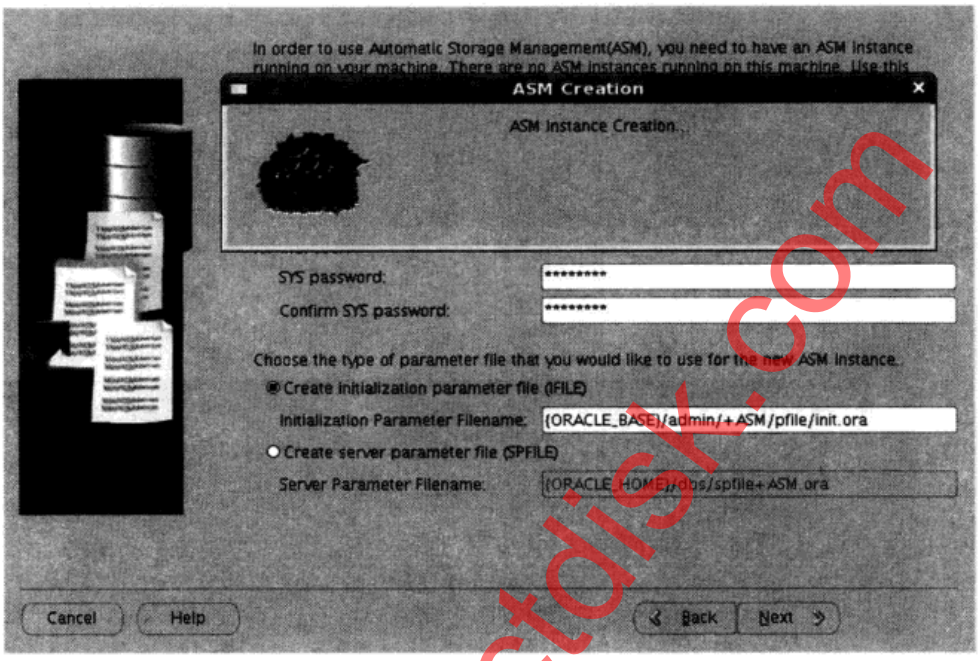

图 13-40　开始创建 ASM 实例

实例创建完毕后自动进入创建 ASM 磁盘组界面，如图 13-41 所示。

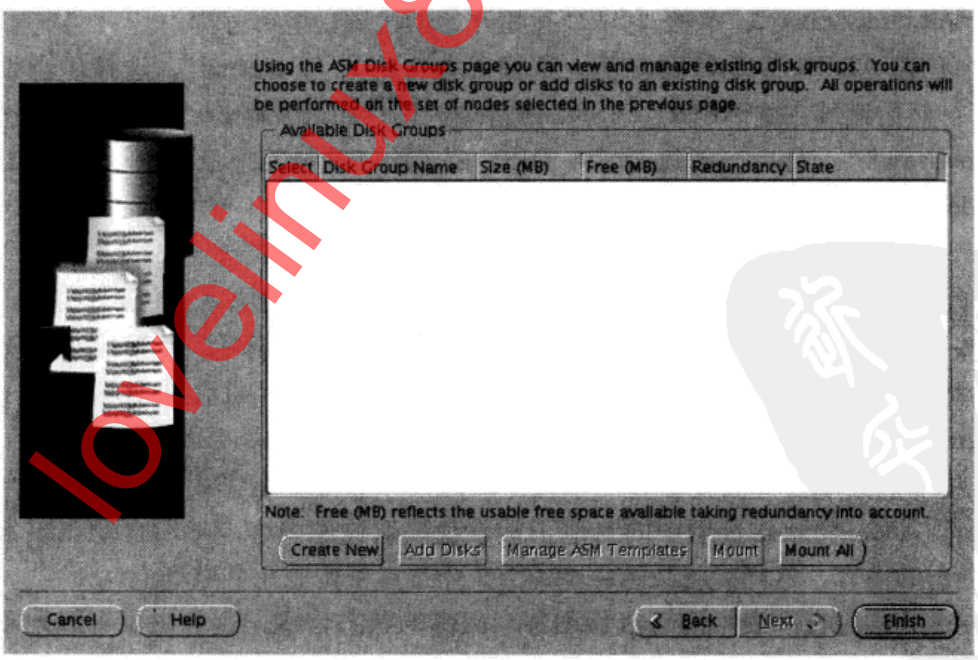

图 13-41　创建 ASM 磁盘组

由于还没有创建 ASM 磁盘组，因此这里是空的。单击“Create New”按钮，创建 ASM 磁盘组，如图 13-42 所示。

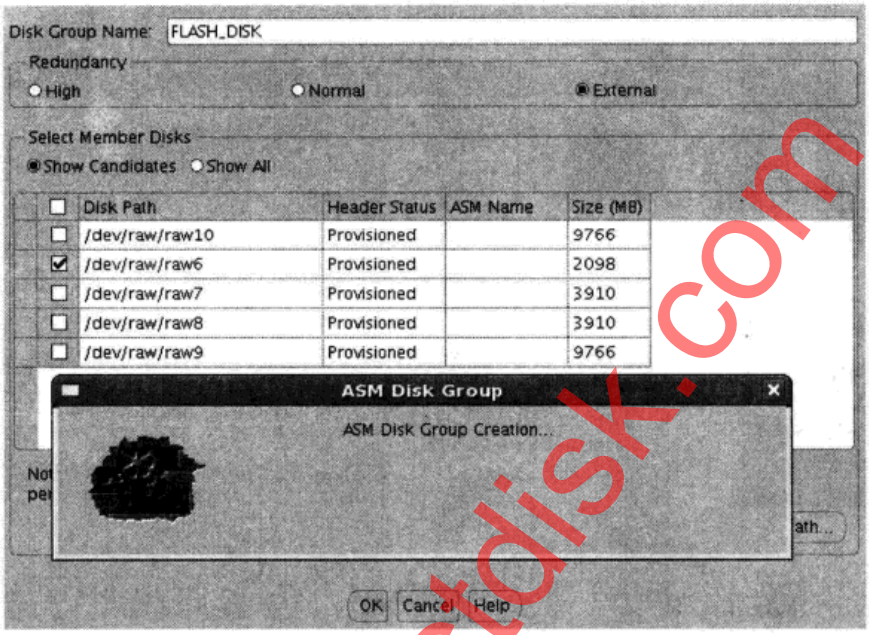

图 13-42 创建 FLASH_DISK 磁盘组

这里输入磁盘组名称为“FLASH_DISK”，然后选择冗余策略为“External”，最后选择磁盘设备“/dev/raw/raw6”，单击“OK”按钮开始创建磁盘组。

接着，创建 ARCH_DISK1 磁盘组，选择冗余策略为“External”，选择磁盘设备“/dev/raw/raw7”；然后继续创建 ARCH_DISK2 磁盘组，选择冗余策略为“External”，选择磁盘设备“/dev/raw/raw8”；最后，创建 DATA_DISK 磁盘组，选择冗余策略为“Normal”，选择磁盘设备“/dev/raw/raw9”和“/dev/raw/raw10”。

从图 13-42 中可知，ASM 提供了采用 3 种不同冗余策略的磁盘组。

- External，即外部冗余，不提供数据镜像功能，可以由单块 ASM 磁盘组成。如果使用硬件镜像，或者允许由于磁盘故障而导致的数据丢失，则可以使用外部冗余磁盘组。
- Normal，即正常冗余，支持双向镜像，最少由两块 ASM 磁盘组成。
- High，即高冗余，可以提供三向镜像，也就是最少由 3 块 ASM 磁盘组成。

ASM 并不镜像磁盘，而镜像分配单元，因此，只有磁盘组需要备用容量。当某一磁盘发生故障时，AMS 将从磁盘组中的其他正常磁盘读取镜像内容，然后自动在正常的磁盘中重建故障磁盘的内容，这样就将故障磁盘的 I/O 分布到了其他几个磁盘上。

从图 13-42 的磁盘列表中还可以看出每个磁盘的状态信息，这些磁盘可以处于以下某一种状态：

- Candidate，表示该磁盘以前从未分配给任何一个 ASM 磁盘组。

- Former，表示该磁盘以前曾分配给某 ASM 磁盘组，但目前处于未分配状态。
- Provisioned，表示正在使用 ASMLib，并且此磁盘尚未分配给任何磁盘组。

所有的磁盘组创建完毕后，状态如图 13-43 所示。

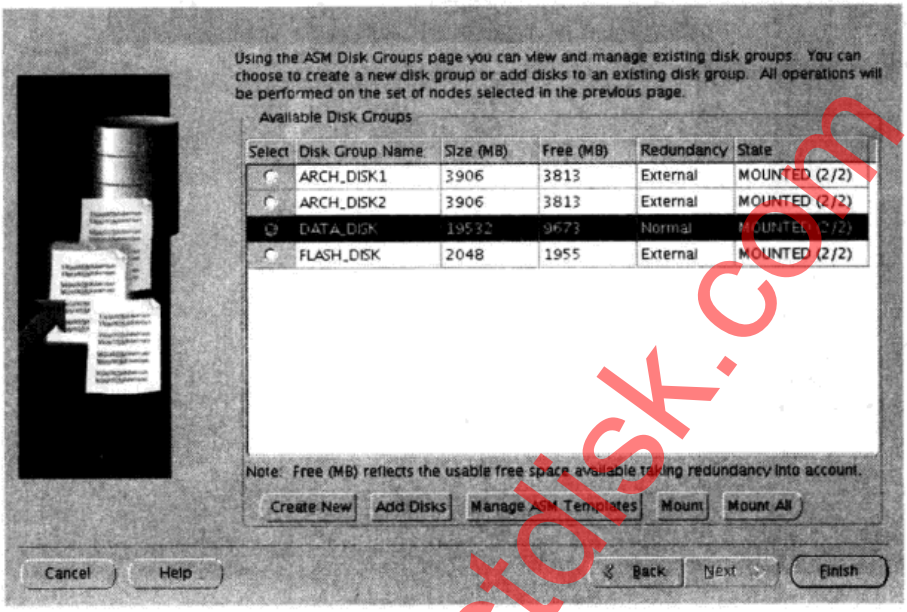

图 13-43　磁盘组状态列表

磁盘组创建完毕后，自动进入 mount 状态，单击“Finish”按钮完成 ASM 的设置。接着开始创建数据库，如图 13-44 所示。

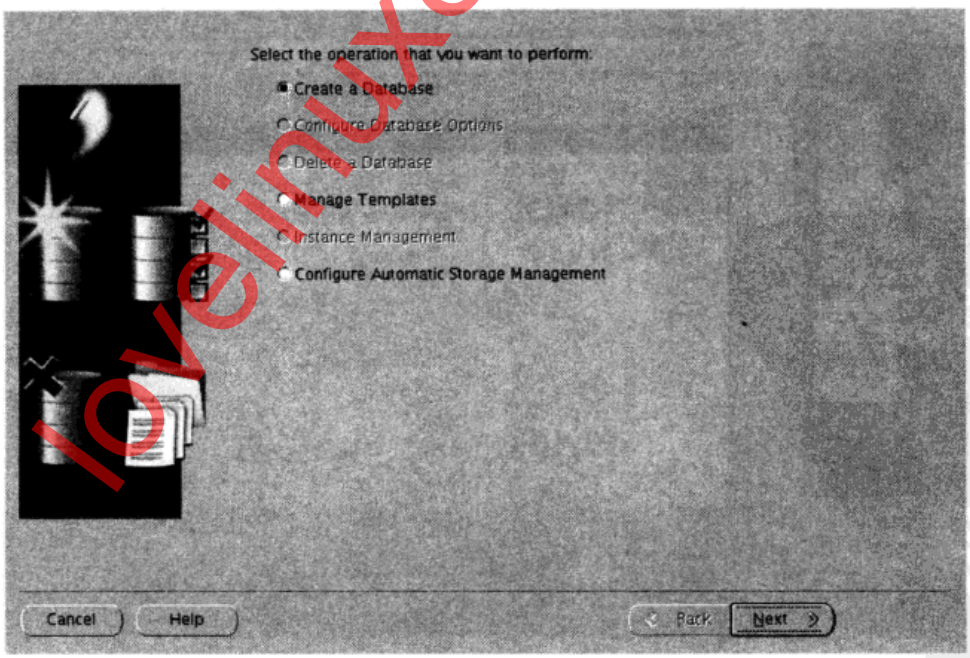

图 13-44　选择创建数据库

这里选择“Create a Database”。单击“Next”按钮进入下一步，如图 13-45 所示。

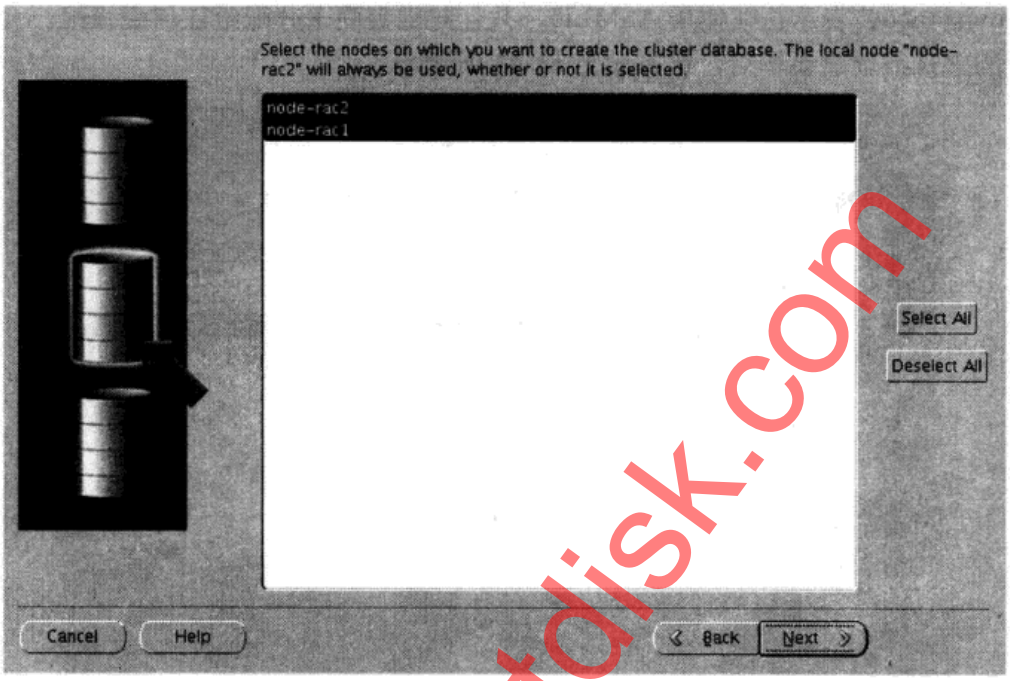

图 13-45 选择集群节点

这里选择在集群所有节点上创建数据库。单击“Next”按钮进入下一步，如图 13-46 所示。

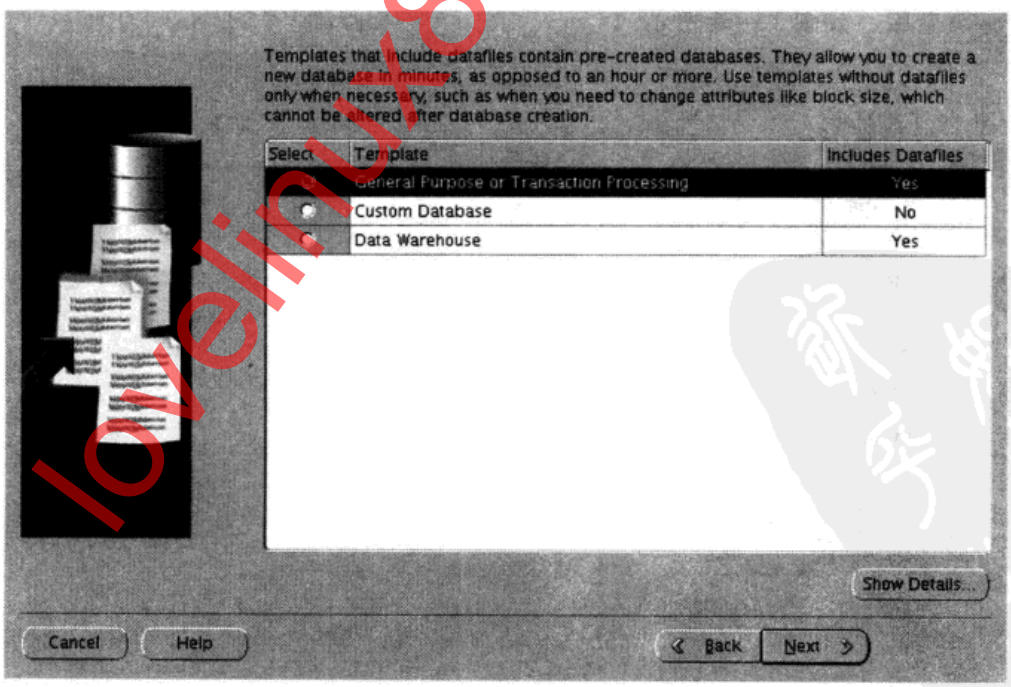

图 13-46 选择创建数据库类型

这里选择“General Purpose or Transation Processing”，即一般用途的数据库模板。单击“Next”按钮进入下一步，如图 13-47 所示。

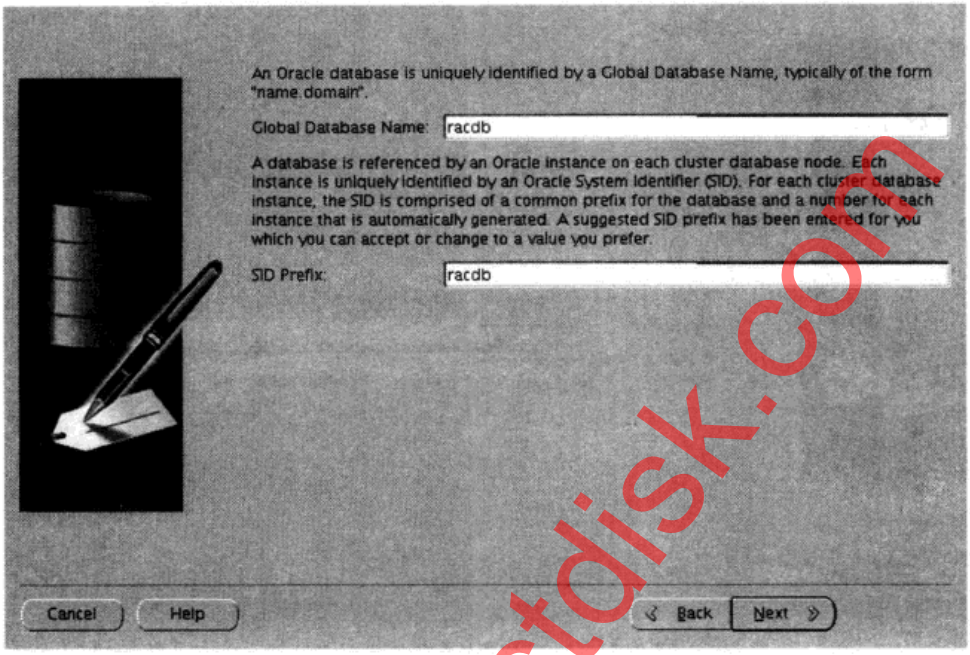

图 13-47　指定数据库标识

这里输入全局数据库名及 SID 前缀均为 racdb。单击“Next”按钮进入下一步，如图 13-48 所示。

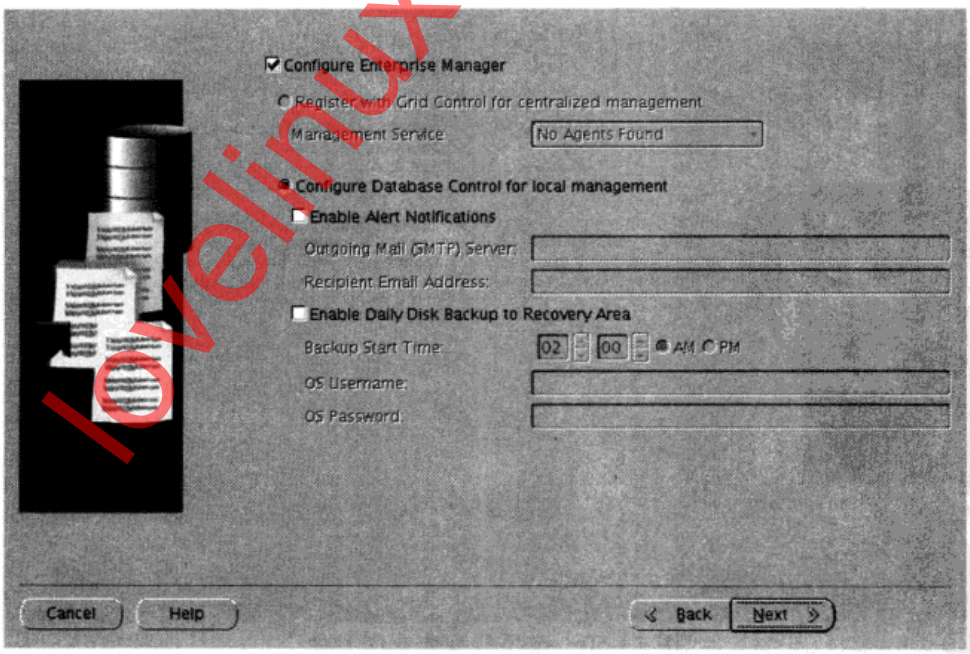

图 13-48　指定管理选项

这里按照默认配置即可。单击“Next”按钮进入下一步，如图 13-49 所示。

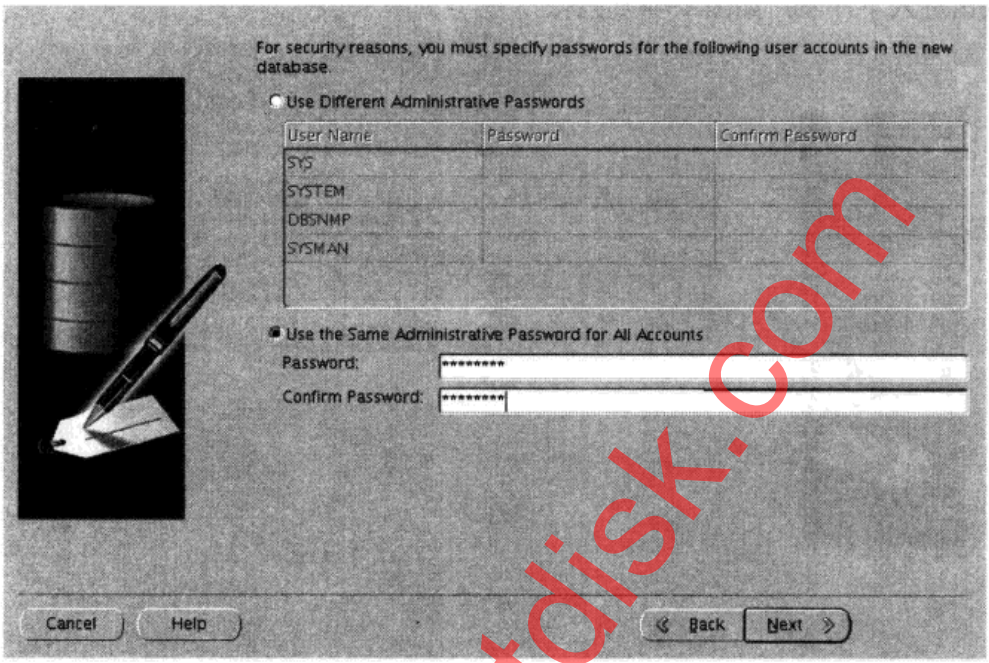

图 13-49 配置数据库身份认证

这里选择所有账户使用同一个口令，然后输入口令即可。单击“Next”按钮进入下一步，如图 13-50 所示。

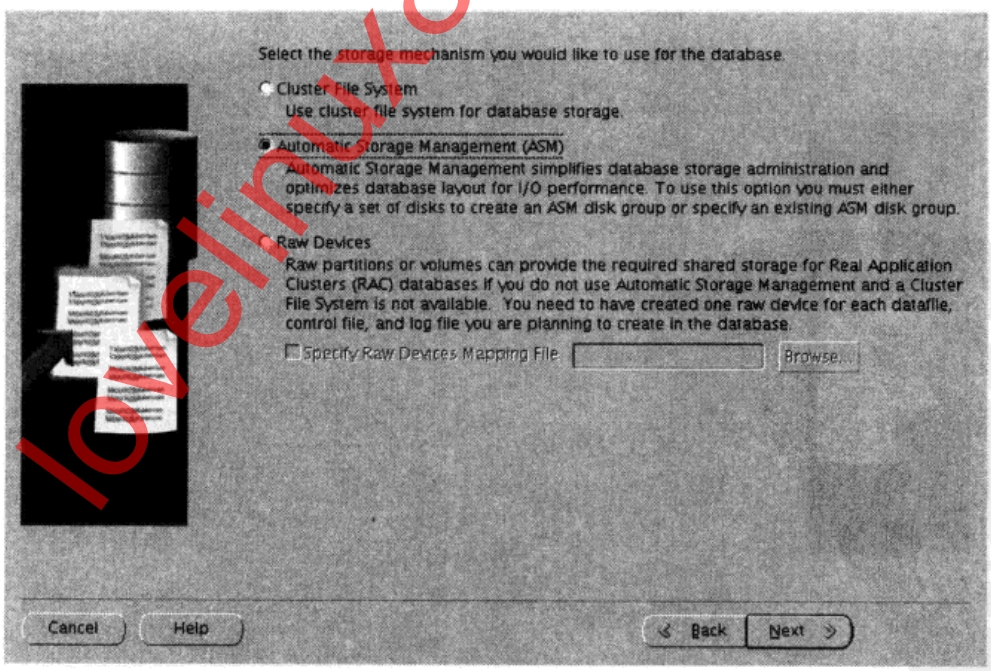

图 13-50 选择数据库存储方式

这里选择“Automatic Storage Management”，即自动存储管理 ASM。单击“Next”按钮进入下一步，如图 13-51 所示。

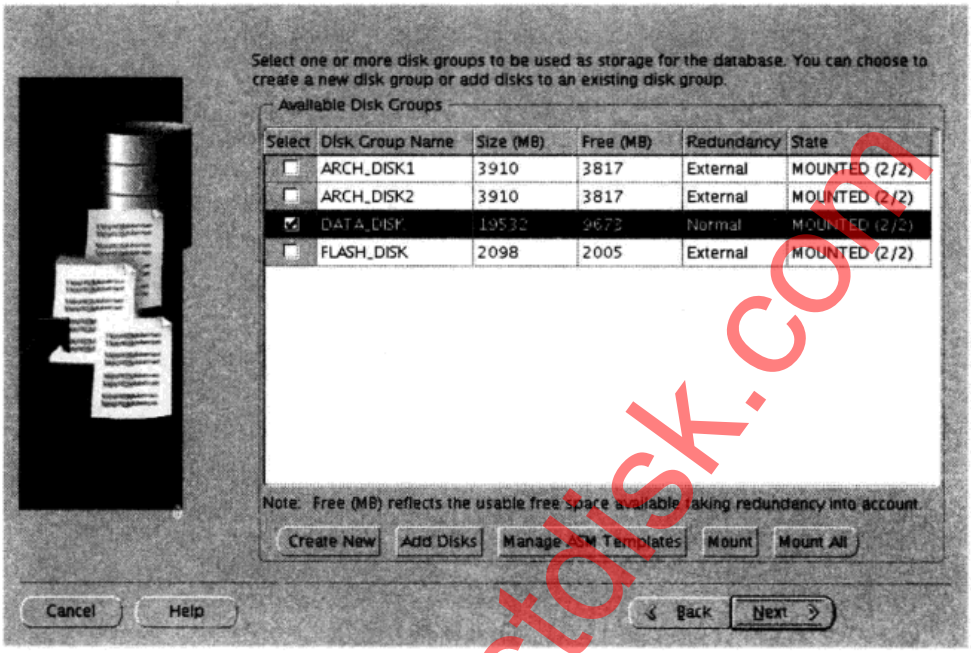

图 13-51　可用的 ASM 磁盘组

在这里可以看到刚创建好的 ASM 磁盘组，选中“DATA_DISK”磁盘组。单击“Next”按钮进入下一步，如图 13-52 所示。

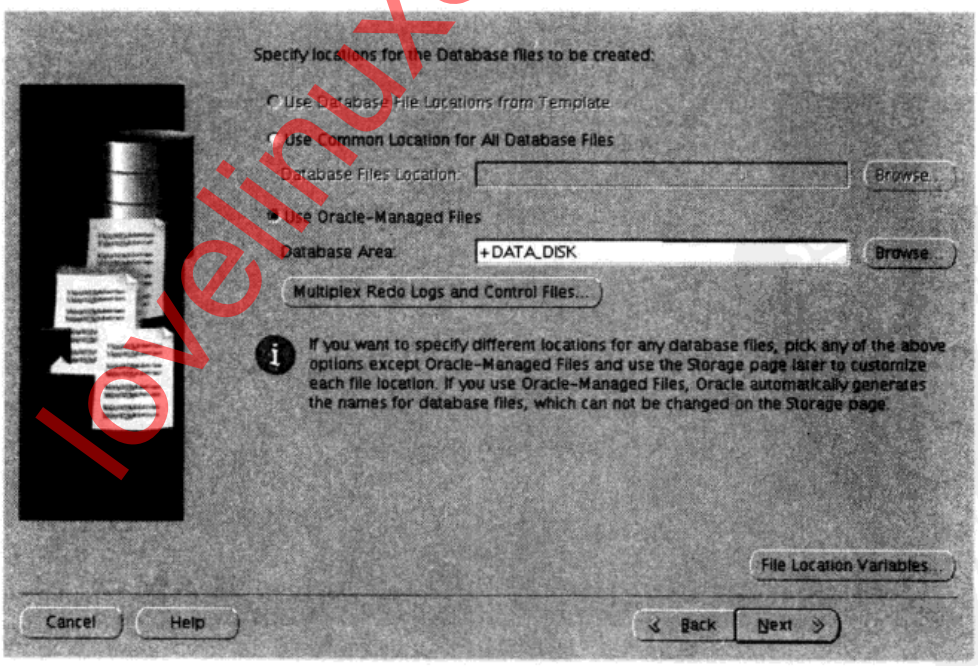

图 13-52　指定 Oracle 数据文件的存储位置

这里将 Oracle 数据文件存储到“+DATA_DISK”磁盘组。单击“Next”按钮进入下一步，如图 13-53 所示。

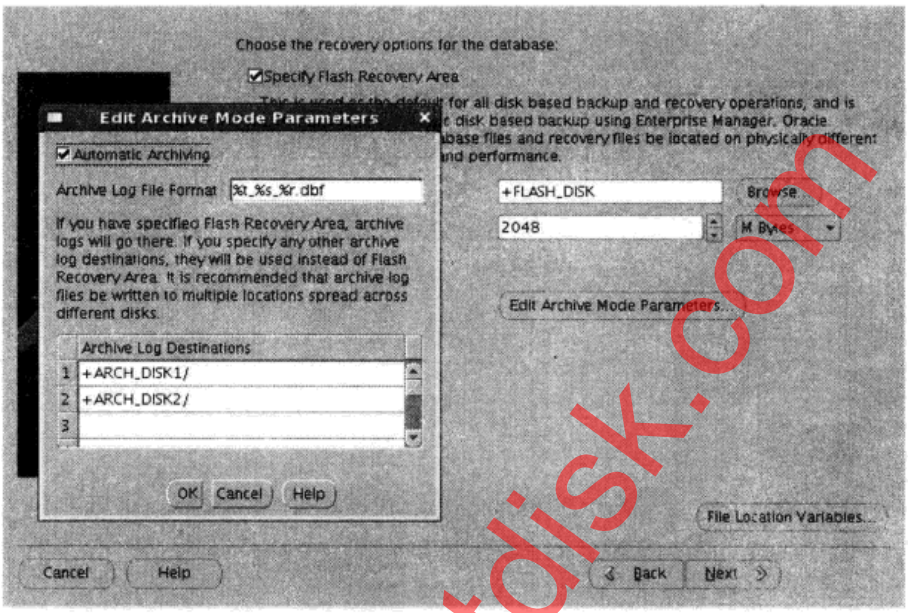

图 13-53 Oracle 恢复设置

这里将快速恢复数据的存储路径指定为“+FLASH_DISK”磁盘组，同时启用数据库的日志归档功能，并将归档日志路径分别设置到“+ARCH_DISK1”和“+ARCH_DISK2”磁盘组。单击“Next”按钮进入下一步，如图 13-54 所示。

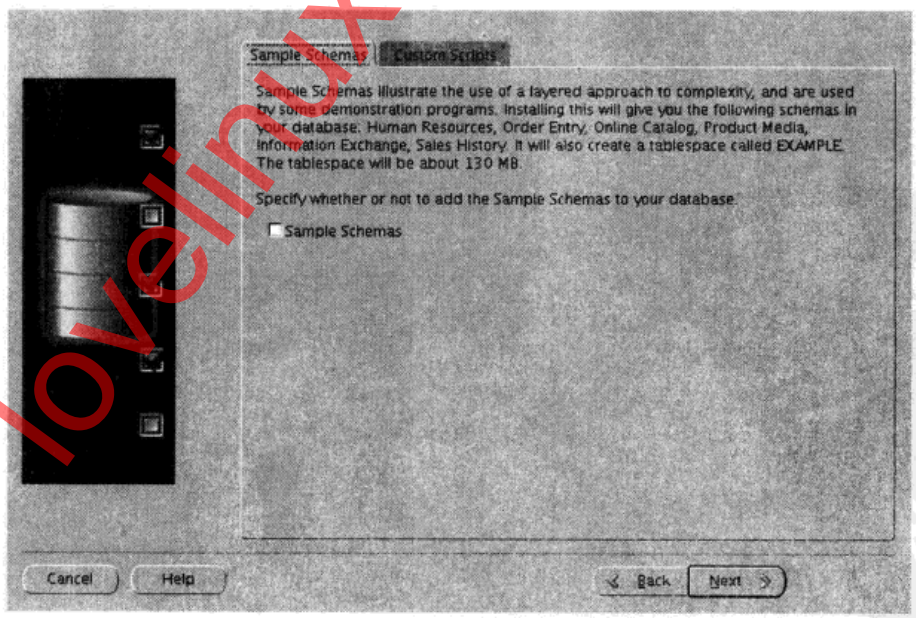

图 13-54 选择示例方案和定制脚本

这里不做任何选择。单击“Next”按钮进入下一步，如图 13-55 所示。

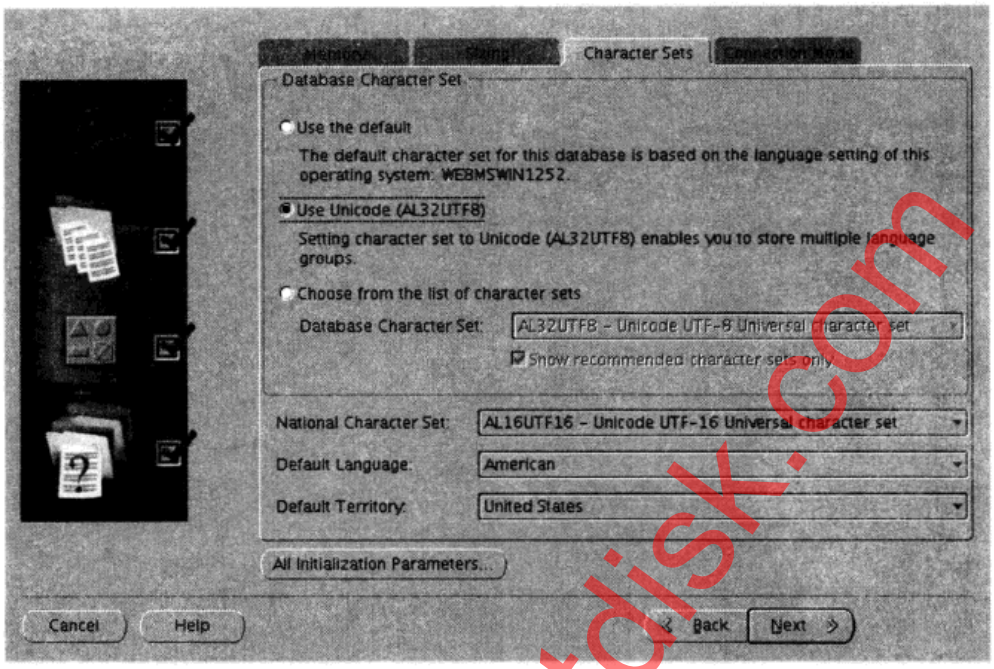

图 13-55　设置 Oracle 初始化参数

在这里指定 SGA 大小、数据库字符集、语言、连接进程数、连接模式等。设置完毕，单击“Next”按钮进入下一步，如图 13-56 所示。

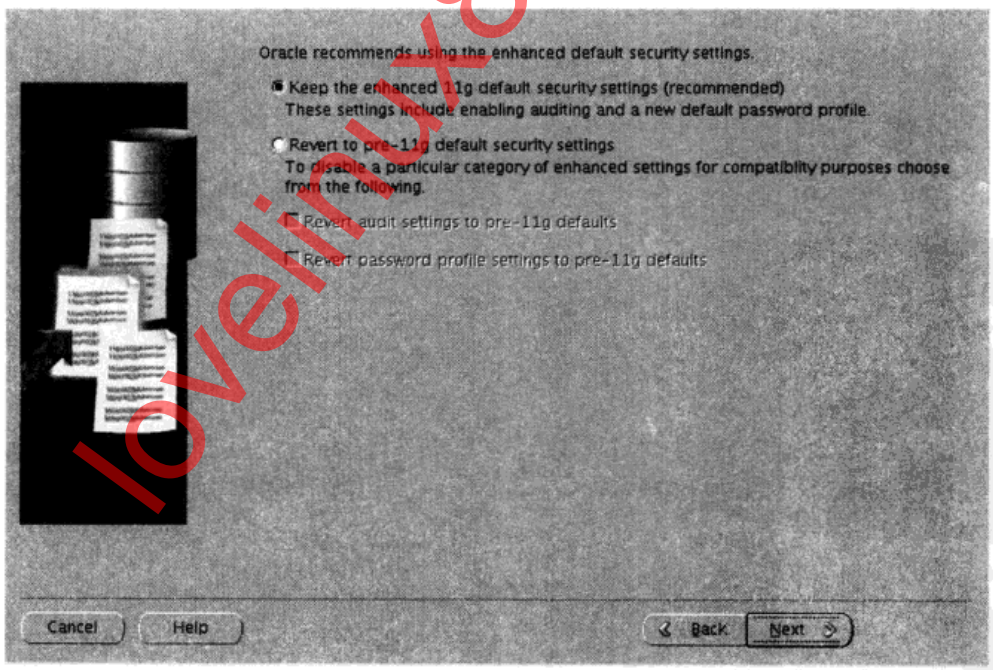

图 13-56　Oracle 安全设置

这里选择默认选项，也就是 Oracle 建议的安全设置。设置完毕，单击“Next”按钮进入下一步，如图 13-57 所示。

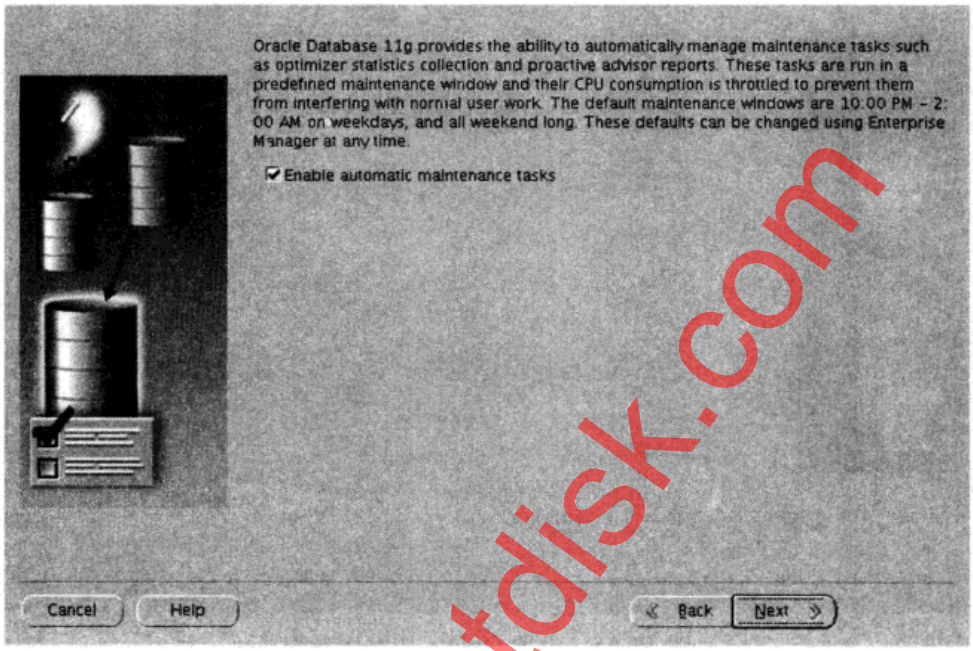

图 13-57 Oracle 自动维护任务

这是 Oracle 11g 新增的一个功能，选择启用即可，也就是启用 Oracle 的自动管理维护功能。接着连续两次单击“Next”按钮进入图 13-58 所示的界面。

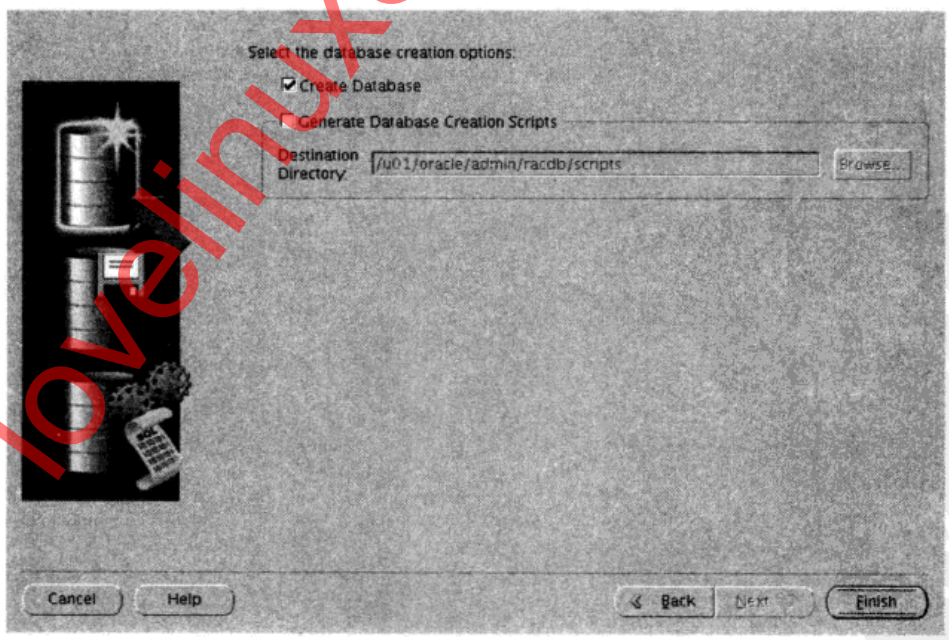

图 13-58 Oracle 安装确认

这里可以选择创建安装脚本，然后单击“Finish”开始安装，如图 13-59 所示。

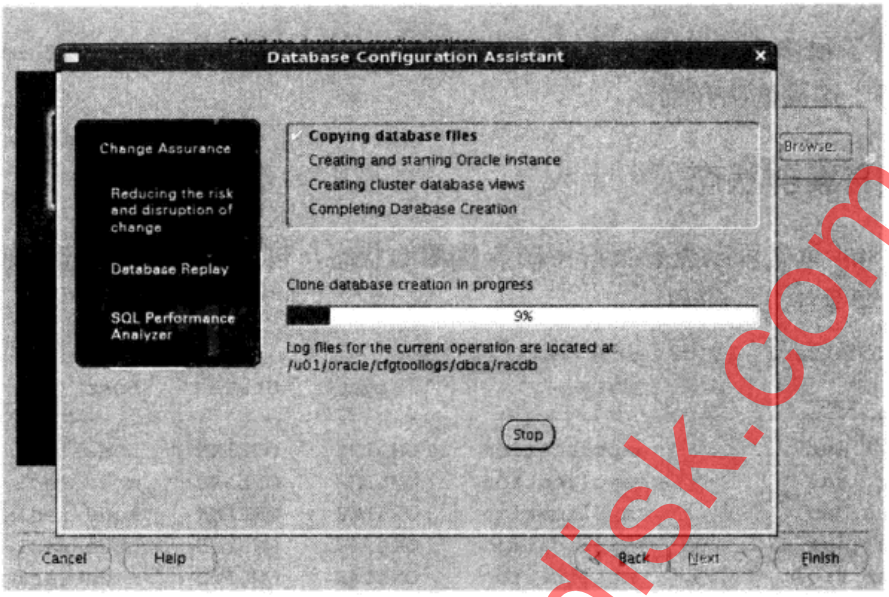

图 13-59　安装 Oracle

根据硬件环境的不同，安装过程可能需要 5 ～ 20 分钟。安装完成后，自动进入如图 13-60 所示的界面。

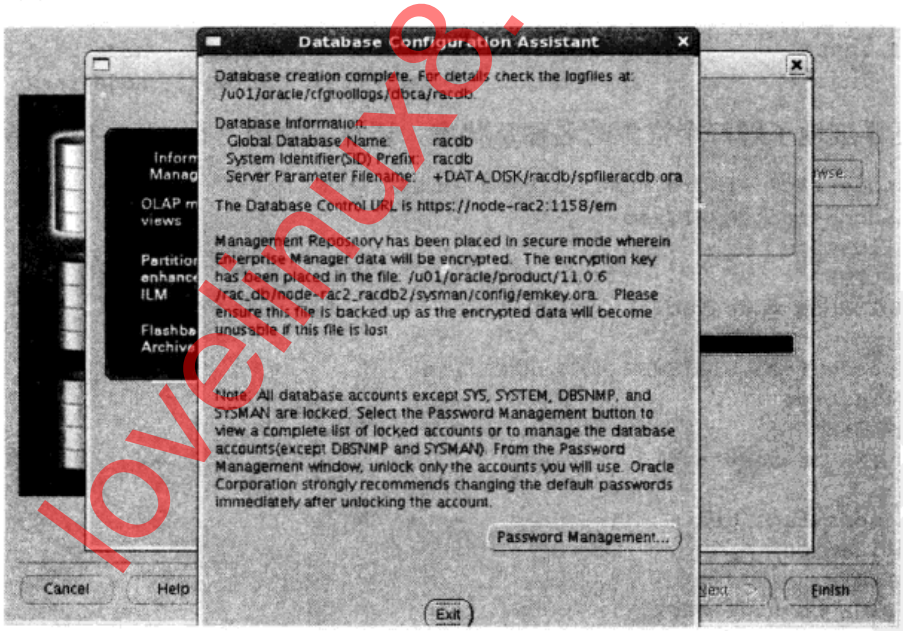

图 13-60　安装 Oracle 结束

数据库安装完成后，安装向导给出了相关确认信息，此时还可以进行密码修改等操作。单击“Exit”按钮结束安装。

13.5 Oracle CRS 的管理与维护

CRS 提供了很多命令来管理和查看集群服务状态，常用的有 crs_stat、crs_start、crs_stop、crsctl 等，这里依次介绍。

13.5.1 查看集群状态

通过 crs_stat 命令可以查看集群中所有资源的状态，包括资源状态、资源运行在哪个节点上、资源类型等信息。例如：

```
[oracle@node-rac1 ~]$ crs_stat -t
Name                    Type          Target    State     Host
------------------      ----------    -------   --------   ------
ora....SM1.asm          application   ONLINE    ONLINE    node-rac1
ora....C1.lsnr          application   ONLINE    ONLINE    node-rac1
ora....ac1.gsd          application   ONLINE    ONLINE    node-rac1
ora....ac1.ons          application   ONLINE    ONLINE    node-rac1
ora....ac1.vip          application   ONLINE    ONLINE    node-rac1
ora....SM2.asm          application   ONLINE    ONLINE    node-rac2
ora....C2.lsnr          application   ONLINE    ONLINE    node-rac2
ora....ac2.gsd          application   ONLINE    ONLINE    node-rac2
ora....ac2.ons          application   ONLINE    ONLINE    node-rac2
ora....ac2.vip          application   ONLINE    ONLINE    node-rac2
ora.racdb.db            application   ONLINE    ONLINE    node-rac2
ora....b1.inst          application   ONLINE    ONLINE    node-rac1
ora....b2.inst          application   ONLINE    ONLINE    node-rac2
```

如果要更详细地了解每个资源的名称及状态，还可以使用“crs_stat -l”命令。例如：

```
[oracle@node-rac2 ~]$crs_stat -l|head -n 15
NAME=ora.node-rac1.ASM1.asm
TYPE=application
TARGET=ONLINE
STATE=ONLINE on node-rac1

NAME=ora.node-rac1.LISTENER_NODE-RAC1.lsnr
TYPE=application
TARGET=ONLINE
STATE=ONLINE on node-rac1

NAME=ora.node-rac1.gsd
TYPE=application
TARGET=ONLINE
STATE=ONLINE on node-rac1
```

可以看到，这个输出中包含了每个服务完整的名称和运行状态。了解了节点运行状态，有助于管理和维护 RAC 集群。

还可以使用 crs_stat -p <resource_name> 来查看资源的属性情况，包括依赖关系等。例如：

```
[oracle@node-rac1 ~]$crs_stat -p  ora.node-rac2.LISTENER_NODE-RAC2.lsnr
```

13.5.2 启动与关闭集群服务资源

1. crs_stop 与 crs_start 命令

通过 crs_stop 命令可以方便地关闭某个服务资源。例如：

```
[oracle@node-rac1 admin]$ crs_stop  ora.node-rac1.LISTENER_NODE-RAC1.lsnr
Attempting to stop `ora.node-rac1.LISTENER_NODE-RAC1.lsnr` on member `node-rac1`
Stop of `ora.node-rac1.LISTENER_NODE-RAC1.lsnr` on member `node-rac1` succeeded.
```

此时，查看 node-rac1 节点的 LISTENER 服务状态如下：

```
[oracle@node-rac1 ~]$ crs_stat -t|grep  lsnr
ora....C1.lsnr application    OFFLINE   OFFLINE
ora....C2.lsnr application    ONLINE    ONLINE    node-rac2
```

从输出可知，node-rac1 节点的 LISTENER 服务已经处于 OFFLINE 状态了。

接着启动 node-rac1 节点的 LISTENER 服务。

```
[oracle@node-rac1 ~]$crs_start  ora.node-rac1.LISTENER_NODE-RAC1.lsnr
Attempting to start `ora.node-rac1.LISTENER_NODE-RAC1.lsnr` on member `node-rac1`
Start of `ora.node-rac1.LISTENER_NODE-RAC1.lsnr` on member `node-rac1` succeeded.
```

其实，RAC 数据库的监听还可以通过如下方式启动和关闭：

```
[oracle@node-rac1 ~]$lsnrctl start LISTENER_NODE-RAC1
[oracle@node-rac1 ~]$lsnrctl stop LISTENER_NODE-RAC1
```

在需要将集群资源全部关闭时，可以通过如下命令完成：

```
[oracle@node-rac1 ~]$ crs_stop -all
```

此时查看集群资源状态，信息如下：

```
[oracle@node-rac1 ~]$ crs_stat -t
Name                 Type          Target    State     Host
------------         ----------    -------   -------   ------
ora....SM1.asm       application   OFFLINE   OFFLINE
ora....C1.lsnr       application   OFFLINE   OFFLINE
ora....ac1.gsd       application   OFFLINE   OFFLINE
ora....ac1.ons       application   OFFLINE   OFFLINE
ora....ac1.vip       application   OFFLINE   OFFLINE
ora....SM2.asm       application   OFFLINE   OFFLINE
ora....C2.lsnr       application   OFFLINE   OFFLINE
ora....ac2.gsd       application   OFFLINE   OFFLINE
ora....ac2.ons       application   OFFLINE   OFFLINE
ora....ac2.vip       application   OFFLINE   OFFLINE
ora.racdb.db         application   OFFLINE   OFFLINE
ora....b1.inst       application   OFFLINE   OFFLINE
ora....b2.inst       application   OFFLINE   OFFLINE
```

可以看到，集群所有资源都处于 OFFLINE 状态。

也可以通过一个命令将集群所有资源全部启动，操作如下：

```
[oracle@node-rac2 ~]$ crs_start  -all
```

2. crsctl 命令

crsctl 命令的功能非常强大，它可以检查 CRS 后台进程运行状态、添加 / 删除表决磁盘、启动 / 关闭集群所有资源、启动 / 关闭 CRS 服务等。下面简单介绍此命令的几个常用参数组合。

检测 node-rac1 节点的状态：

```
[oracle@node-rac2 ~]$crsctl check cluster -node node-rac1
node-rac1 is ONLINE
```

检查 cssd 服务的运行状态：

```
[oracle@node-rac2 ~]$crsctl check cssd
Cluster Synchronization Services appears healthy
```

检查 CRS 的运行版本：

```
[oracle@node-rac2 ~]$crsctl query crs activeversion
Oracle Clusterware active version on the cluster is [11.1.0.6.0]
```

查看 css 加载的模块：

```
[oracle@node-rac2 ~]$ crsctl  lsmodules css
The following are the Cluster Synchronization Services modules::
    CSSD
    COMMCRS
    COMMNS
```

停止本节点上所有 CRS 资源：

```
[oracle@node-rac1 ~]$crsctl stop resources
Stopping resources.
This could take several minutes.
Successfully stopped Oracle Clusterware resources
```

启动本节点上所有 CRS 资源：

```
[oracle@node-rac1 ~]$crsctl start resources
Starting resources.
Successfully started CRS resources
```

在本节点上关闭 CRS 服务（此操作需要 root 用户身份）：

```
[root@node-rac1 ~]#/app/oracrs/product/11.0.6/crs/bin/crsctl stop crs
Stopping resources.
This could take several minutes.
Successfully stopped Oracle Clusterware resources
Stopping Cluster Synchronization Services.
Shutting down the Cluster Synchronization Services daemon.
Shutdown request successfully issued.
```

13.5.3 启动与关闭 CRS

在 RAC 数据库中，CRS 接管了数据库的启动和关闭等操作，集群节点的实例随着 CRS

服务的启动而自动启动，但是 CRS 也可以手工启动和关闭。

管理 CRS 服务的命令如下：

```
[root@node-rac1 ~]# /etc/init.d/init.crs {stop|start|enable|disable}
```

例如，要关闭某个节点的 CRS 服务，可以执行如下操作：

```
[root@node-rac1 ~]# /etc/init.d/init.crs  stop
Shutting down Oracle Cluster Ready Services (CRS):
Sep 08 10:57:14.806 | INF | daemon shutting down
Stopping resources.
This could take several minutes.
 Successfully stopped Oracle Clusterware resources
Stopping Cluster Synchronization Services.
Shutting down the Cluster Synchronization Services daemon.
Shutdown request successfully issued.
Shutdown has begun. The daemons should exit soon.
```

CRS 服务关闭后，与此节点相关的集群实例也随之停止，同时此节点的 VIP 地址也将转移到另一个节点上。

启动 CRS 服务，执行如下操作：

```
[root@node-rac1 ~]# /etc/init.d/init.crs  start
Startup will be queued to init within 30 seconds.
```

CRS 启动后，主要有以下 4 个后台进程：

```
[oracle@node-rac1 ~]$ ps -ef|grep d.bin
root      5166  4186  0 Sep07 ? 00:02:33 /app/oracrs/product/11.0.6/crs/bin/crsd.bin reboot
oracle    5176  5170  0 Sep07 ? 00:00:05 /app/oracrs/product/11.0.6/crs/bin/evmd.bin
oracle    5840  5309  0 Sep07 ? 00:01:04 /app/oracrs/product/11.0.6/crs/bin/ocssd.bin
oracle    6306     1  0 Sep07 ? 00:00:00 /app/oracrs/product/11.0.6/crs/bin/oclskd.bin
oracle   30233 30185  0 10:01 pts/1    00:00:00 grep d.bin
```

下面简单介绍每个进程的含义：

- ocssd，用于管理与协调集群中各节点的关系，并用于节点间通信。该进程非常重要，如果这个进程异常中止，会导致系统自动重启。在某些极端情况下，如果 ocssd 无法正常启动，会导致操作系统循环重启。
- crsd，监控节点各种资源，当某个资源发生异常时，自动重启或者切换该资源。
- evmd，是一个基于后台的事件检测程序。
- oclskd，该守护进程是 Oracle 11g（11.1.0.6）新增的一个后台进程，主要用于监控 RAC 数据库节点实例，当某个实例挂起时，就重启该节点。

13.5.4　管理 voting disk

voting disk（表决磁盘）主要用于记录节点成员信息，例如包含哪些节点成员、节点添加删除信息的记录等。

（1）查看 Voting disk

要查看 voting disk 信息，可执行如下命令：

```
crsctl query css votedisk
```

例如：

```
[oracle@node-rac1 ~]$ crsctl query css votedisk
 0.     0    /dev/raw/raw3
 1.     0    /dev/raw/raw4
 2.     0    /dev/raw/raw5
Located 3 voting disk(s).
```

（2）备份 voting disk

voting disk 的信息很重要，需要定期备份 votedisk。例如：

```
[oracle@node-rac1 ~]$dd if=/dev/raw/raw3 of=/tmp/votedisk.bak
```

这样就把 voting disk 的信息备份到了 /tmp 目录下。

（3）恢复 voting disk

例如：

```
[oracle@node-rac1 ~]$dd if=/tmp/votedisk.bak of=/dev/raw/raw3
```

（4）删除 voting disk

如果要删除一块 voting disk，可以执行如下命令：

```
crsctl delete css votedisk /dev/raw/raw5 -force
[root@node-rac1 ~]#/app/oracrs/product/11.0.6/crs/bin/crsctl \
>delete css votedisk  /dev/raw/raw5
Successful deletion of voting disk /dev/raw/raw5.
[oracle@node-rac1 ~]$ crsctl query css votedisk
 0.     0    /dev/raw/raw3
 1.     0    /dev/raw/raw4
```

（5）添加 voting disk

添加一块表决磁盘时，执行如下命令：

```
crsctl add css votedisk /dev/raw/raw5 -force
```

这个操作需要以 root 用户身份执行，例如：

```
[root@node-rac1 ~]#/app/oracrs/product/11.0.6/crs/bin/crsctl add css votedisk /
    dev/raw/raw5 Now formatting voting disk: /dev/raw/raw5.
Successful addition of voting disk /dev/raw/raw5.
[oracle@node-rac1 ~]$ crsctl query css votedisk
crsctl query css votedisk
 0.     0    /dev/raw/raw3
 1.     0    /dev/raw/raw4
 2.     0    /dev/raw/raw5
Located 3 voting disk(s).
```

13.5.5 管理 OCR

OCR 磁盘主要用于记录节点成员的配置信息，如数据库、服务、实例、VIP 地址、监听

器、应用进程等 CRS 资源配置信息，OCR 可以存储在裸设备或群集文件系统上。

1. 检查 OCR 设置

可以通过如下命令查看当前的 OCR 设置：

```
[oracle@node-rac1 ~]$ ocrcheck
Status of Oracle Cluster Registry is as follows :
Version:                          2
Total space (kbytes):             4000280
Used space (kbytes):              3840
Available space (kbytes) :        3996440
ID:                               1806962807
Device/File Name:                         /dev/raw/raw1
                                Device/File integrity check succeeded
Device/File Name:                         /dev/raw/raw2
                                Device/File integrity check succeeded
Cluster registry integrity check succeeded
```

2. OCR 的备份与恢复

在默认情况下，Oracle 每 4 个小时自动备份一次 OCR，并保存 3 个有效版本，但是仅保存在集群的某一个节点上。

要查看自动备份 OCR 文件的路径，执行如下命令：

```
[oracle@node-rac1 ~]$ocrconfig -showbackup
node-rac2   2010/09/08 15:33:44     /app/oracrs/product/11.0.6/crs/cdata/rac-cluster/
    backup00.ocr
node-rac2   2010/09/08 11:33:43     /app/oracrs/product/11.0.6/crs/cdata/rac-cluster/
    backup01.ocr
node-rac2   2010/09/08 07:33:42     /app/oracrs/product/11.0.6/crs/cdata/rac-cluster/
    backup02.ocr
node-rac2   2010/09/07 23:33:38     /app/oracrs/product/11.0.6/crs/cdata/rac-cluster/
    day.ocr
node-rac2   2010/08/28 00:41:30     /app/oracrs/product/11.0.6/crs/cdata/rac-cluster/
    week.ocr
```

恢复 OCR 的方法也很简单，例如：

```
[oracle@node-rac1 ~]$ocrconfig -restore \
> /app/oracrs/product/11.0.6/crs/cdata/rac-cluster/backup01.ocr
```

其实 OCR 也可以通过手动的方式导出、导入，方法如下：

1）手动导出：ocrconfig -export /tmp/ocr_bak。

2）手动导入：ocrconfig -import /tmp/ocr_bak。

3. 添加 OCR 镜像盘

向磁盘组中添加 OCR 磁盘的基本步骤如下：

1）执行命令 crsctl stop crs 停掉 CRS 服务。

2）创建用于镜像 OCR 的 RAW 设备，例如 :/dev/raw/raw11。

3）执行“ocrconfig –export”命令导出OCR的信息。

4）修改/etc/oracle/ocr.loc文件，添加新增的RAW设备，例如：

```
[oracle@node-rac1 ~]$more /etc/oracle/ocr.loc
ocrconfig_loc=/dev/raw/raw1
ocrmirrorconfig_loc=/dev/raw/raw2
ocrmirrorconfig_loc=/dev/raw/raw11
local_only=FALSE
```

5）执行“ocrconfig -import”命令将OCR备份信息导入磁盘。

6）检查OCR设置信息，看是否增加成功。

7）用“crsctl start crs”命令启动CRS服务。

13.5.6 快速卸载CRS

安装CRS非常简单，但是如果CRS出现问题，需要重新安装时，卸载CRS并不是一件轻松的事情，下面就重点讲述如何在Linux下卸载CRS程序。

1）在卸载CRS之前，必须停止CRS服务，即执行：

```
/etc/init.d/init.crs stop
```

如果无法停止CRS服务，那么就将CRS服务禁用，即执行：

```
/etc/init.d/init.crs disable
```

然后重启系统。

2）执行删除脚本。首先在集群中的所有节点上执行$ORA_CRS_HOME/install/rootdelete.sh，这里以节点node-rac1为例。下面是操作过程：

```
[root@node-rac1 install]# /app/oracrs/product/11.0.6/crs/install/rootdeinstall.sh
Verifying existence of ocr.loc file
Removing contents from OCR mirror device
2560+0 records in
2560+0 records out
10485760 bytes (10 MB) copied, 3.90308 seconds, 2.7 MB/s
Removing contents from OCR device
2560+0 records in
2560+0 records out
10485760 bytes (10 MB) copied, 3.84383 seconds, 2.7 MB/s
```

接着，在集群中的任意一个节点上执行$ORA_CRS_HOME/install/rootdeinstall.sh。

3）通过图形界面OUI卸载CRS。首先在CRS安装包目录下启动CRS安装欢迎界面，如图13-4所示。然后在欢迎界面上单击“Installed Products”按钮，会弹出图13-61所示的界面。

在这个界面中可以看到已经安装的CRS信息，单击右下角的“Remove”按钮，开始卸载CRS。

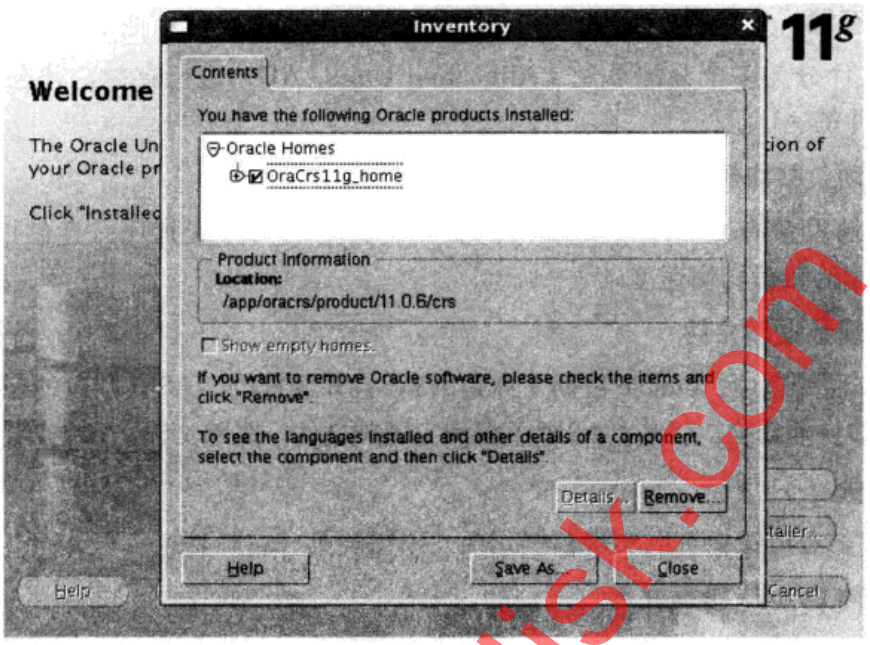

图 13-61　卸载 CRS

4）CRS 卸载程序可以删除大部分安装程序，但是并不能完全卸载安装程序，所以需要手动删除未卸载的一些文件。在集群的所有节点上执行如下删除命令：

```
rm -rf /app/oracrs/product/11.0.6/crs/
/bin/rm -rf /var/tmp/.oracle
/bin/rm -f /etc/oraInst.loc
rm -rf /etc/oracle/
rm -f /etc/oratab
```

所有操作执行完毕后，重启集群的每个节点，重新安装 CRS 即可。

13.6　ASM 基本操作维护

ASM（自动存储管理）是一个专门为 Oracle 数据库服务的数据文件存储机制，通过 ASM 管理数据文件，DBA 不用再担心 I/O 性能问题，也不需要知道文件的名称，同时 ASM 也提供了文件系统到卷管理器的集成。

13.6.1　ASM 的特点

1. 自动调整 I/O 负载

ASM 可以在所有可用的磁盘中自动调整 I/O 负载，不但避免了人工调整 I/O 的难度，而且也优化了性能。同时，利用 ASM 可以在线增加数据库的大小，而无需关闭数据库。

2. 条带化存储

ASM 将文件分为多个分配单元（Allocation Units，AU）进行存储，并在所有磁盘间平均分配每个文件的 AU。

3. 在线自动负载均衡

当共享存储设备有变化时，ASM 中的数据会自动均匀分配到现有存储设备中。同时，还可以调节数据的负载均衡速度。

4. 自动管理数据库文件

在 ASM 存储管理中，Oracle 数据文件是 ASM 自动管理的。ASM 创建的任何文件一旦不再需要，就会被自动删除。但是，ASM 不管理二进制文件、跟踪文件、预警日志和口令文件。

5. 数据冗余

ASM 通过磁盘组镜像可以实现数据冗余，不需要第三方工具。

6. 支持各种 Oracle 数据文件

ASM 存储支持 Oracle 数据文件、日志文件、控制文件、归档日志、RMAN 备份集等。

13.6.2 ASM 的体系结构与后台进程

图 13-62 显示了 ASM 的物理构成。

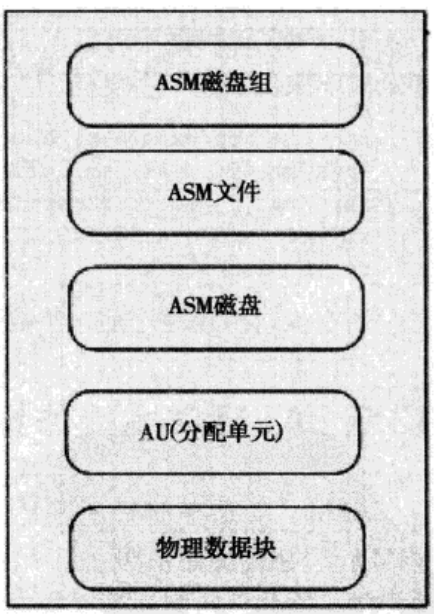

图 13-62 ASM 的物理构成

从图 13-62 可以看出，在顶层是 ASM 磁盘组，ASM 实例和数据库实例可以直接访问这些磁盘组；然后是 ASM 文件，每个 ASM 文件只能包含在一个磁盘组中，不过，一个磁盘组中可以包含属于多个数据库的多个 ASM 文件，并且单个数据库可以使用来自多个磁盘组的存储空间；第三部分是 ASM 磁盘，多个 ASM 磁盘组成了 ASM 磁盘组，但每个 ASM 磁盘只能属于一个磁盘组；接着是 AU（分配单元），AU 是 ASM 磁盘组分配的最小连续磁盘空间，ASM 磁盘按照 AU 进行分区，每个 AU 的大小为 1MB；这个结构的底层是 Oracle 数据块，由于 AU 是 ASM 分配的最小连续磁盘空间，因此，ASM 是不允许跨分配单元拆分一个 Oracle 数据块的。

要使用 ASM，需要在启动数据库实例之前，先启动一个名为“+ASM”的实例，ASM 实例不会装载数据库，启动它的目的是为了管理磁盘组和保护其中的数据。同时，ASM 实例还可以向数据库实例传递有关文件布局的信息。通过这种方式，数据库实例就可以直接访问磁盘组中存储的文件。图 13-63 显示了 ASM 的一般体系结构。

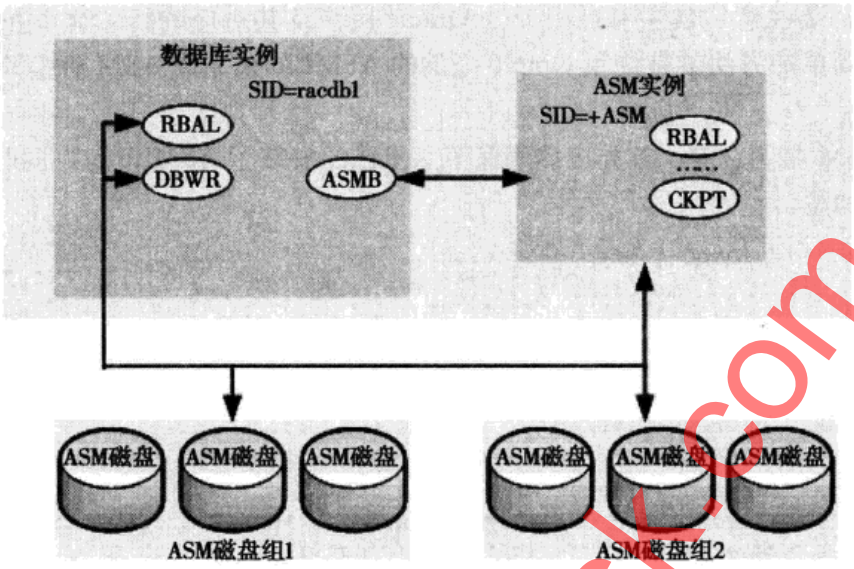

图 13-63　ASM 的一般体系结构

从图 13-63 可以看出，ASM 实例与数据库实例进行通信的桥梁是 ASMB 进程，此进程运行在每个数据库实例上，是两个实例间信息交换的通道。ASMB 进程先利用磁盘组名称通过 CSS 获得管理该磁盘组的 ASM 实例连接串，然后建立一个到 ASM 的持久连接，这样两个实例之间就可以通过这条连接定期交换信息，同时这也是一种心跳监控机制。

另外，在 ASM 实例中还存在另外一个新的进程，即 RBAL，此进程负责规划和协调磁盘组的重新平衡活动。除此之外，ASM 实例还有一些与数据库实例中的进程相同的后台进程，例如 LGWR、SMON、PMON、DBWR、CKPT 等。

如果一个数据库实例使用 ASM 作为存储，那么它将多出两个后台进程，即 RBAL 和 ASMB。RBAL 负责打开磁盘组中所有磁盘和数据，而 ASMB 负责和 ASM 实例进程通信。

13.6.3　管理 ASM 实例

在使用 ASM 作为数据存储时，ASM 实例管理显得非常重要，Oracle 提供了丰富的管理功能，对 ASM 实例进行管理需要具备 SYSDBA 权限，在 Oracle 11g 中可以使用一个新角色，即 SYSASM，此角色只用于管理 ASM 实例。

1. 创建 ASM 实例

创建 ASM 实例有两种方法，第一种是利用 dbca 创建，这种方法只需运行 Database Configuration Assistant（DBCA），然后根据提示即可创建一个 ASM 实例，此种方式在前面已有讲述，这里不再多说。第二种方法是用命令行方式创建 ASM 实例，下面进行简单介绍。

（1）创建 ASM 磁盘

可以使用 RAID 划分的 LUN、分区和裸设备等来创建 ASM 磁盘，但是在使用 LUN、分

区或裸设备时，要注意将属主和属组改为 Oracle 用户及其对应的组，这个在前面已经讲述过。另外一种简单的方法就是使用 Oracle 提供的 ASMLib 来完成 ASM 磁盘的创建，下面将讲述这种方法。

在创建 ASM 实例之前，首先应该确保节点上已经安装了 ASMlib 包，同时确认 ASMLib 是否已经自动加载。

```
[root@node1 ~]# lsmod | grep oracleasm
oracleasm              46356  1
```

然后，通过 ASMlib 提供的 oracleasm 将已经划分好的磁盘分区转化为 ASM 磁盘。例如：

```
[root@node1 ~]#/etc/init.d/oracleasm createdisk ASMDISK1 /dev/sdc5
Marking disk "/dev/sdc5" as an ASM disk [  OK  ]
[root@node1 ~]#/etc/init.d/oracleasm createdisk ASMDISK2 /dev/sdc6
Marking disk "/dev/sdc6" as an ASM disk [  OK  ]
[root@node1 ~]#/etc/init.d/oracleasm createdisk ASMDISK3 /dev/sdc7
Marking disk "/dev/sdc7" as an ASM disk [  OK  ]
[root@node1 ~]#/etc/init.d/oracleasm createdisk ASMDISK4 /dev/sdc8
Marking disk "/dev/sdc8" as an ASM disk [  OK  ]
[root@node1 ~]#/etc/init.d/oracleasm createdisk ASMdisk5 /dev/sdc9
Marking disk "/dev/sdc9" as an ASM disk [  OK  ]
```

创建完 ASM 磁盘后，可以查看系统的 /dev/oracleasm/disks/ 目录下是否已经生成磁盘设备，可以采用的命令如下：

```
[root@node1 ~]# ll /dev/oracleasm/disks/ASMDISK*
brw-rw---- 1 oracle oinstall 8, 21 Sep 10 23:40 /dev/oracleasm/disks/ASMDISK1
brw-rw---- 1 oracle oinstall 8, 22 Sep 10 23:40 /dev/oracleasm/disks/ASMDISK2
brw-rw---- 1 oracle oinstall 8, 23 Sep 10 23:36 /dev/oracleasm/disks/ASMDISK3
brw-rw---- 1 oracle oinstall 8, 24 Sep 10 23:40 /dev/oracleasm/disks/ASMDISK4
brw-rw---- 1 oracle oinstall 8, 25 Sep 10 23:40 /dev/oracleasm/disks/ASMDISK5
```

也可以通过如下方式查看：

```
[root@node1 ~]#service oracleasm listdisks
ASMDISK1
ASMDISK2
ASMDISK3
ASMDISK4
ASMDISK5
```

如果要删除 ASM 磁盘可通过以下命令：

```
[root@node1 ~]#/etc/init.d/oracleasm deletedisk ASMDISK5
Removing ASM disk "ASMdisk5" [  OK  ]
```

在 RAC 环境中，要注意另外一个节点是否能够发现对应的 ASM 磁盘，执行如下命令，让另外一个节点来获取这种变化。

```
[root@node2 ~]#/etc/init.d/oracleasm scandisks
```

到此为止，ASM 磁盘已经创建完毕了。

（2）初始化参数

启动 ASM 实例只需要如下几个参数即可，利用这些参数可以实现 ASM 实例的内存的自动分配和自动管理。

下面介绍 ASM 实例初始化参数。

```
instance_type=asm
cluster_database=true
DB_UNIQUE_NAME=+ASM
ASM_POWER_LIMIT=1
large_pool_size=60M
asm_diskgroups='FLASH_DISK','ARCH_DISK','DATA_DISK'
asm_diskstring='/dev/oracleasm/disks/*'
```

每个参数的含义如下：

- instance_type，指定实例的类型，对于 ASM 实例，应设置为 ASM。
- cluster_database，指定是否是数据库集群，true 表示是 ASM 集群。
- DB_UNIQUE_NAME，指定 ASM 实例的名称，默认是 +ASM。
- ASM_POWER_LIMIT，该参数用来控制 ASM 中数据的负载均衡速度。
- large_pool_size，设置池的大小，由于 ASM 文件的分配单元映射是从 large_pool 分配的，因此 large_pool_size 至少要 8MB，建议越大越好。
- asm_diskgroups，指定实例启动时可用的 ASM 磁盘组，ASM 实例将在启动时自动挂载这些磁盘组。
- asm_diskstring，用于限制 ASM 实例可用于创建磁盘组的磁盘设备。如果该值为 NULL，则 ASM 实例可见的所有磁盘都可以成为创建磁盘组的可选磁盘。

（3）创建密码文件

```
[oracle@node1 ~]$su - oracle
[oracle@node1 ~]$ cd $ORACLE_HOME/dbs
[oracle@node1 ~]$orapwd file=orapw+ASM password=oracle
```

（4）创建目录结构

```
[oracle@node1 ~]  $su - oracle
[oracle@node1 ~]  $cd $ORACLE_HOME/dbs
[oracle@node1 ~]  $mkdir -p $ORALCE_BASE/admin/+ASM/udump
[oracle@node1 ~]  $mkdir -p $ORALCE_BASE/admin/+ASM/bdump
[oracle@node1 ~]  $mkdir -p $ORALCE_BASE/admin/+ASM/adump
[oracle@node1 ~]  $mkdir -p $ORALCE_BASE/admin/+ASM/cdump
```

2. 启动 ASM 实例

无论在 RAC 环境中还是单实例环境，ASM 实例都需要用到 CSS 进程，在 RAC 环境中，启动 CRS 后 CSS 已经运行，而在单实例环境下，需要以 root 用户运行脚本，初始化 CSS 服务，否则，在启动 ASM 实例时会报如下错误：

```
ORA-29701：unable to connect to Cluster Manager
```

执行初始化脚本的过程如下：

```
[root@node1 ~]#$ORACLE_HOME/bin/localconfig add
/etc/oracle does not exist. Creating it now.
Successfully accumulated necessary OCR keys.
Creating OCR keys for user 'root', privgrp 'root'…
Operation successful.
Configuration for local CSS has been initialized

Cleaning up Network socket directories
Setting up Network socket directories
 Adding to inittab
Startup will be queued to init within 30 seconds.
Checking the status of new Oracle init process...
Expecting the CRS daemons to be up within 600 seconds.
 Cluster Synchronization Services is active on these nodes.
   node1
Cluster Synchronization Services is active on all the nodes.
Oracle CSS service is installed and running under init(1M)
```

然后启动 ASM 实例。

```
[oracle@node1 ~]$export ORACLE_SID=+ASM
[oracle@node1 ~]$sqlplus / as sysdba
SQL> startup
ASM instance started
Total System Global Area          134217728 bytes
Fixed Size                          1218124 bytes
Variable Size                     107833780 bytes
ASM Cache                          25165824 bytes
ORA-15110: no diskgroups mounted
```

因为首次启动 ASM 实例并没有创建 ASM 磁盘组，所以显示 15110 错误是正常的。

3. 管理 ASM 磁盘组

ASM 磁盘组是作为逻辑单元进行统一管理的一组磁盘，在 ASM 实例中，可以创建和添加新的磁盘组，可以修改现有的磁盘组，在其中添加一个磁盘或者删除一个磁盘，也可以删除现有的磁盘组。

（1）添加磁盘组

```
SQL> create diskgroup FLASH_DISK external redundancy disk '/dev/oracleasm/disks/
    ASMDISK1' name flashdisk;
Diskgroup created.
SQL> create diskgroup ARCH_DISK external redundancy disk '/dev/oracleasm/disks/
    ASMDISK2' name archdisk1;
Diskgroup created.
SQL> create diskgroup DATA_DISK normal redundancy disk '/dev/oracleasm/disks/
    ASMDISK4' name datadisk1, '/dev/oracleasm/disks/ASMDISK5' name datadisk2;
Diskgroup created.
```

（2）查看磁盘组状态

```
SQL> select name,state from v$asm_diskgroup;
NAME                             STATE
-------------------              --------------
FLASH_DISK                       MOUNTED
ARCH_DISK                        MOUNTED
DATA_DISK                        MOUNTED
```

（3）卸载 FLASH_DISK 磁盘组

```
SQL> alter diskgroup FLASH_DISK dismount;
Diskgroup altered.
SQL> select name,state from v$asm_diskgroup;
NAME                STATE
----------------  -----------
FLASH_DISK          DISMOUNTED
ARCH_DISK           MOUNTED
DATA_DISK           MOUNTED
```

（4）挂载 FLASH_DISK 磁盘组

```
SQL> alter diskgroup  FLASH_DISK mount;
Diskgroup altered.
SQL> select name,state from v$asm_diskgroup;
NAME                STATE
----------------  ------------
FLASH_DISK          MOUNTED
ARCH_DISK           MOUNTED
DATA_DISK           MOUNTED
```

（5）查看磁盘名与裸设备对应关系

```
SQL>  select name,path from v$asm_disk_stat;
NAME                             PATH
-------------------              ---------------------------------------
                                 /dev/oracleasm/disks/ASMDISK3
DATADISK2                        /dev/oracleasm/disks/ASMDISK5
DATADISK1                        /dev/oracleasm/disks/ASMDISK4
ARCHDISK1                        /dev/oracleasm/disks/ASMDISK2
FLASHDISK                        /dev/oracleasm/disks/ASMDISK1
```

（6）查看每个磁盘组的可用大小

```
SQL> select name,allocation_unit_size,total_mb from v$asm_diskgroup;
NAME                ALLOCATION_UNIT_SIZE            TOTAL_MB
--------------    ------------------------          ----------
FLASH_DISK          1048576                         3815
ARCH_DISK           1048576                         3815
DATA_DISK           1048576                         954
```

（7）向磁盘组中增加一个磁盘

```
SQL> ALTER DISKGROUP ARCH_DISK ADD DISK '/dev/oracleasm/disks/ASMDISK3' name
```

```
    ARCHDISK2;
Diskgroup altered.
```

查看每个磁盘组的可用大小。

```
SQL> select name,allocation_unit_size,total_mb from v$asm_diskgroup;
NAME              ALLOCATION_UNIT_SIZE   TOTAL_MB
---------------   --------------------   ----------
FLASH_DISK        1048576                3815
ARCH_DISK         1048576                4292
DATA_DISK         1048576                954
SQL> select name,path from v$asm_disk_stat;
NAME                              PATH
-----------                       --------------------------------
DATADISK2                         /dev/oracleasm/disks/ASMDISK5
DATADISK1                         /dev/oracleasm/disks/ASMDISK4
ARCHDISK2                         /dev/oracleasm/disks/ASMDISK3
ARCHDISK1                         /dev/oracleasm/disks/ASMDISK2
FLASHDISK                         /dev/oracleasm/disks/ASMDISK1
```

可以看出，磁盘组 ARCH_DISK 的大小发生变化了，表明添加磁盘成功。

（8）从磁盘组中删除一个磁盘：

```
SQL> ALTER DISKGROUP ARCH_DISK DROP DISK ARCHDISK2;
Diskgroup altered.
```

（9）删除一个磁盘组：

```
SQL> drop diskgroup  FLASH_DISK;
Diskgroup dropped.
```

当有数据库使用 ASM 的磁盘组时，是无法卸载和删除这个磁盘组的。ASM 实例如果宕掉，那么使用 ASM 的数据库实例也会宕掉。在 RAC 环境中，在删除一个磁盘组之前，其他节点的 ASM 实例必须将这个要删除的磁盘组卸载。

4. 关闭 ASM 实例

关闭 ASM 实例的命令和关闭数据库实例的命令相同，但只有在没有任何数据库实例连接到该 ASM 实例的情况下，才能正常关闭 ASM 实例，如果至少有一个数据库实例与之连接，会提示以下错误：

```
ORA-15097: cannot SHUTDOWN ASM instance with connected RDBMS instance
```

此时，如果对该 ASM 实例强制执行 SHUTDOWN ABORT 命令，那么 ASM 实例将被关闭，任何与之连接的数据库实例最终也将自动关闭，同时报以下错误：

```
ORA-15064: communication failure with ASM instance
```

ASM 实例被强制关闭后，在下次启动时，会要求进行恢复。

5. ASMCMD 命令

Oracle 在 10g 版本中提供了 ASMCMD 命令，通过这个命令可以管理存储在 ASM 磁盘

中的数据。下面简单介绍 ASMCMD 命令的使用方法。

在使用 ASMCMD 命令时必须启动 ASM 实例，然后指定 ORACLE_HOME 和 ORACLE_SID。例如：

```
[oracle@node-rac1 ~]$ export ORACLE_SID=+ASM1
[oracle@node-rac1 ~]$ asmcmd
ASMCMD>
```

还可以使用“asmcmd -p”，加上“-p”参数可以显示当前路径。

下面是 ASMCMD 提供的一些可用命令，其中 oracle 11g 新增的命令有 cp、md_backup、md_restore。

```
ASMCMD> ?
   commands:
   --------
   help

   cd
   cp
   du
   find
   ls
   lsct
   lsdg
   mkalias
   mkdir
   pwd
   rm
   rmalias

   md_backup
   md_restore

   lsdsk
   remap
```

（1）切换目录

```
ASMCMD> cd +DATA_DISK/RACDB
```

（2）列出目录信息

```
ASMCMD> ls
CONTROLFILE/
DATAFILE/
ONLINELOG/
PARAMETERFILE/
TEMPFILE/
spfileracdb.ora
```

（3）查看磁盘空间信息

执行 ASMCMD 的“du DATAFILE”命令查看磁盘空间信息，如图 13-64 所示。

```
ASMCMD> du DATAFILE
Used_MB      Mirror_used_MB
   1682                3368

ASMCMD> lsdg
State    Type    Rebal  Sector Block   AU       Total_MB  Free_MB  Req_mir_free_MB  Usable_file_MB  Offline_disks  Name
MOUNTED  EXTERN  N         512  4096  1048576      3910     2252                0            2252              0  ARCH_DISK1/
MOUNTED  EXTERN  N         512  4096  1048576      3910     2252                0            2252              0  ARCH_DISK2/
MOUNTED  NORMAL  N         512  4096  1048576     19532    15414                0            7707              0  DATA_DISK/
MOUNTED  EXTERN  N         512  4096  1048576      2098     1756                0            1756              0  FLASH_DISK/
```

图 13- 64 查看磁盘空间信息

（4）显示 ASM 和数据库实例连接情况

执行 ASMCMD 的“lsct”命令显示 ASM 和数据库实例连接情况，如图 13-65 所示。

```
ASMCMD> lsct
DB_Name  Status     Software_Version  Compatible_version  Instance_Name  Disk_Group
racdb    CONNECTED        11.1.0.6.0          11.1.0.0.0  racdb2         DATA_DISK
racdb    CONNECTED        11.1.0.6.0          11.1.0.0.0  racdb2         FLASH_DISK
```

图 13-65 显示 ASM 和数据库实例连接情况

（5）创建一个目录

```
ASMCMD> mkdir test
ASMCMD> ls
CONTROLFILE/
DATAFILE/
ONLINELOG/
PARAMETERFILE/
TEMPFILE/
spfileracdb.ora
test/
```

（6）复制磁盘文件

将 ASM 磁盘文件 spfileracdb.or 复制到 test 目录下：

```
ASMCMD> cp spfileracdb.ora test
source +DATA_DISK/RACDB/spfileracdb.ora
target +DATA_DISK/RACDB/test/spfileracdb.ora
copying file(s)...
file, +DATA_DISK/racdb/test/spfileracdb.ora, copy committed.
ASMCMD> cd test
ASMCMD> ls
spfileracdb.ora
```

将 ASM 磁盘文件 UNDOTBS1.258.728340289 文件复制到操作系统某目录下：

```
ASMCMD> cp UNDOTBS1.258.728340289 UNDOTBS1.dbf
source +DATA_DISK/RACDB/DATAFILE/UNDOTBS1.258.728340289
target UNDOTBS1.dbf
copying file(s)...
copying file(s)...
copying file(s)...
```

```
copying file(s)...
copying file(s)...
copying file(s)...
file, /u01/oracle/product/11.0.6/rac_db/dbs/UNDOTBS1.dbf, copy committed.
```

（7）备份 ASM 的 metadata

```
ASMCMD> md_backup -b /u01/oracle/datadisk.bak -g data_disk
Disk group to be backed up: DATA_DISK
```

这样就把 ASM 的 metadata 备份到了文件系统上，通过查看 datadisk.bak 文件可以得知 metadata 的组织信息。

（8）为 ASM 磁盘文件设置别名

执行如图 13-66 所示的命令为 ASM 磁盘文件设置别名。

```
ASMCMD> mkalias SYSAUX.257.728340287 SYSAUX.dbf
ASMCMD> mkalias USERS.259.728340289 USERS.dbf
ASMCMD> mkalias UNDOTBS1.258.728340289 UNDOTBS1.dbf
ASMCMD> mkalias UNDOTBS2.264.728340891 UNDOTBS2.dbf
ASMCMD> mkalias SYSTEM.256.728340285 SYSTEM.dbf

ASMCMD> ls -al
Type      Redund  Sys  Name
DATAFILE  MIRROR  Y  +DATA_DISK/RACDB/DATAFILE/SYSAUX.dbf => SYSAUX.257.728340287
                  N  SYSAUX.dbf => +DATA_DISK/RACDB/DATAFILE/SYSAUX.257.728340287
DATAFILE  MIRROR  Y  +DATA_DISK/RACDB/DATAFILE/SYSTEM.dbf => SYSTEM.256.728340285
                  N  SYSTEM.dbf => +DATA_DISK/RACDB/DATAFILE/SYSTEM.256.728340285
DATAFILE  MIRROR  Y  +DATA_DISK/RACDB/DATAFILE/UNDOTBS1.dbf => UNDOTBS1.258.728340289
                  N  UNDOTBS1.dbf => +DATA_DISK/RACDB/DATAFILE/UNDOTBS1.258.728340289
DATAFILE  MIRROR  Y  +DATA_DISK/RACDB/DATAFILE/UNDOTBS2.dbf => UNDOTBS2.264.728340891
                  N  UNDOTBS2.dbf => +DATA_DISK/RACDB/DATAFILE/UNDOTBS2.264.728340891
DATAFILE  MIRROR  Y  +DATA_DISK/RACDB/DATAFILE/USERS.dbf => USERS.259.728340289
                  N  USERS.dbf => +DATA_DISK/RACDB/DATAFILE/USERS.259.728340289
```

图 13-66　为 ASM 磁盘文件设置别名

（9）在磁盘组查找文件

```
ASMCMD> find +DATA_DISK sys*
+DATA_DISK/RACDB/DATAFILE/SYSAUX.257.728340287
+DATA_DISK/RACDB/DATAFILE/SYSAUX.dbf
+DATA_DISK/RACDB/DATAFILE/SYSTEM.256.728340285
+DATA_DISK/RACDB/DATAFILE/SYSTEM.dbf
```

（10）在磁盘组删除文件

```
ASMCMD> rm -rf test
```

13.7　利用 srvctl 管理 RAC 数据库

srvctl 即 Server Control，是 Oracle 提供的一个命令行工具，用于管理 Oracle 的 RAC 环境。srvctl 在 Oracle 9i 中被引入，Oracle10g、11g 对其功能进行了很大的增强和改进。下面介绍此命令的简单用法。

13.7.1 查看实例状态（srvctl status）

查询所有实例和服务的状态：

```
[oracle@node-rac1 ~]$ srvctl status database -d racdb
Instance racdb2 is running on node node-rac2
Instance racdb1 is running on node node-rac1
```

查询实例 racdb1 的状态：

```
[oracle@node-rac1 ~]$ srvctl status instance -d racdb -i racdb1
Instance racdb1 is running on node node-rac1
```

查询实例 racdb2 的状态：

```
[oracle@node-rac1 ~]$ srvctl status instance -d racdb -i racdb2
Instance racdb2 is running on node node-rac2
```

查询特定节点上应用程序的状态：

```
[oracle@node-rac1 ~]$ srvctl status nodeapps -n node-rac2
VIP is running on node: node-rac2
GSD is running on node: node-rac2
Listener is running on node: node-rac2
ONS daemon is running on node: node-rac2
```

查询特定节点上 ASM 实例的状态：

```
[oracle@node-rac1 ~]$ srvctl status asm -n node-rac2
ASM instance +ASM2 is running on node node-rac2.
```

在上面的命令行操作中，都用到的参数是：

- -d，即 database name，表示数据库名称。
- -n，即 node name，表示节点名称。
- -i，即 instance name，表示实例名称。

13.7.2 查看 RAC 数据库配置信息（srvctl config）

显示 RAC 数据库的配置：

```
[oracle@node-rac1 ~]$ srvctl config database -d racdb
node-rac2 racdb2 /u01/oracle/product/11.0.6/rac_db
node-rac1 racdb1 /u01/oracle/product/11.0.6/rac_db
```

列出配置的所有数据库：

```
[oracle@node-rac1 ~]$ srvctl config database
racdb
```

显示指定节点的应用程序配置：

```
[oracle@node-rac1 ~]$ srvctl config nodeapps -n node-rac2
VIP exists.: /node-vip2/192.168.12.240/255.255.255.0/eth0
```

```
GSD exists.
ONS daemon exists.
Listener exists.
```

显示指定节点的 ASM 实例配置：

```
[oracle@node-rac1 ~]$ srvctl config asm -n node-rac2
+ASM2 /u01/oracle/product/11.0.6/rac_db
```

13.7.3 启动 / 关闭实例（srvctl start/stop）

停止 Oracle RAC 所有服务：

```
[oracle@node-rac1 ~]$ emctl stop dbconsole
[oracle@node-rac1 ~]$ srvctl stop instance -d racdb -i racdb1
[oracle@node-rac1 ~]$ srvctl stop asm -n node-rac1
[oracle@node-rac1 ~]$ srvctl stop nodeapps -n node-rac1
```

也可以通过一条命令停止所有实例及其启用的服务：

```
[oracle@node-rac1 ~]$srvctl stop database -d racdb
```

启动 Oracle RAC 所有服务：

```
[oracle@node-rac1 ~]$ srvctl start nodeapps -n node-rac1
[oracle@node-rac1 ~]$ srvctl start asm -n node-rac1
[oracle@node-rac1 ~]$ srvctl start instance -d racdb -i racdb1
[oracle@node-rac1 ~]$ emctl start dbconsole
```

也可以通过一条命令启动所有实例及其启用的服务：

```
[oracle@node-rac1 ~]$srvctl start database -d racdb
```

13.7.4 增加 / 删除 / 修改实例（srvctl add/remove/modify）

增加一个服务，然后在节点间切换此服务：

```
[oracle@node-rac1 ~]$ srvctl add service -d racdb -s test -r racdb1 -a racdb2 -P BASIC
```

其中参数的含义如下：

- -r，表示首选实例。
- -a，表示可用的实例。
- -P，表示故障切换策略，有 none、BASIC、preconnect 三个可选项。

在集群节点之间切换集群服务：

```
[oracle@node-rac1 ~]$ srvctl start service -d racdb -s test -i racdb1
[oracle@node-rac1 ~]$ srvctl status service -d racdb -s test
Service test is running on instance(s) racdb1
[oracle@node-rac1 ~]$ srvctl stop  service  -d racdb -s test -i racdb1
[oracle@node-rac1 ~]$ srvctl start  service  -d racdb -s test -i racdb2
[oracle@node-rac1 ~]$ srvctl status  service  -d  racdb -s test
```

```
Service test is running on instance(s) racdb2
```

从某个实例节点移除一个服务：

```
[oracle@node-rac1 ~]$ srvctl remove service -d racdb -s test -i racdb2
test PREF: racdb1 racdb2 AVAIL:
Remove service test from the instance racdb2? (y/[n]) y
```

使数据库服务对某个实例可用：

```
[oracle@node-rac1 ~]$ srvctl add service -d racdb -s test -u -a racdb2
[oracle@node-rac1 ~]$ srvctl start service -d racdb -s test
[oracle@node-rac1 ~]$ srvctl modify service -d racdb -s test -i racdb2 -r
```

13.8 测试 RAC 数据库集群的功能

Oracle RAC 是一个集群数据库，可以实现负载均衡和故障无缝切换。如何知道 RAC 数据库已经实现了这些功能呢？下面就对此进行功能测试。

13.8.1 负载均衡测试

RAC 数据库的负载均衡是指对数据库连接的负载均衡，当一个新的会话连接到 RAC 数据库时，通过指定的分配算法将请求分配到集群的任一节点上，这就是 RAC 数据库完成的功能。负载均衡在 RAC 中分为两种：一种是基于客户端连接的负载均衡；另一种是基于服务器端的负载均衡。

1. RAC 客户端负载均衡

客户端连接的负载均衡配置起来非常简单，与 RAC 数据库的实例负载和监听没有任何关系，因此也就不需要在集群节点进行任何设置，只要在客户端机器上的 tnsnames.ora 文件中添加负载均衡策略配置即可。这里以 Linux 客户端为例进行介绍。

（1）修改 /etc/hosts 文件

编辑 /etc/hosts 文件，将 RAC 数据库相关的 IP 地址信息添加进去。例如：

```
192.168.12.231          node-rac1
192.168.12.232          node-rac2
192.168.12.230          node-vip1
192.168.12.240          node-vip2
```

（2）查看 RAC 数据库的 service_names

```
[oracle@node-rac1 ~]$ sqlplus "/as sysdba"
SQL*Plus: Release 11.1.0.6.0 - Production on Sun Sep 12 22:05:53 2010
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
```

```
and Real Application Testing options
NAME              TYPE          VALUE
--------------    -----------   --------
service_names     string        racdb
```

这里需要说明的是，在配置 RAC 负载均衡时，客户端连接的是 RAC 数据库的服务名，而不是实例名，也就是 SERVICE_NAME 必须设置为“SERVICE_NAME = racdb”。

（3）修改 Oracle 客户端的配置文件 tnsnames.ora

```
RACDB=
  (DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = node-vip2)(PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = node-vip1)(PORT = 1521))
    (LOAD_BALANCE = yes)
    )
    (CONNECT_DATA =
        (SERVER = DEDICATED)
        (SERVICE_NAME = racdb)
    )
)
```

这个配置文件的说明如下：

- LOAD_BALANCE = yes，表示启用连接负载均衡。在默认情况下“LOAD_BALANCE = no”，因此如果要配置负载均衡，必须添加设置“LOAD_BALANCE =yes”。启用负载均衡后，SQLNet 会随机选择 ADDRESS_LIST 列表中的任意一个监听，然后将请求分发到此监听上，通过这种方式完成负载均衡。如果“LOAD_BALANCE = no”，那么 SQLNet 会按照 ADDRESS_LIST 列表中的顺序选择监听，只要这个监听正常就一直使用该监听。
- SERVICE_NAME = racdb，这个“racdb”是 RAC 数据库的服务名，而非实例名。

（4）在客户端测试负载均衡

在客户端开启一个 sqlplus 连接，执行如下操作：

```
[oracle@client ~]$ sqlplus system/xxxxxx@racdb
SQL*Plus: Release 11.1.0.7.0 - Production on Sun Sep 12 21:24:55 2010
Copyright (c) 1982, 2008, Oracle.  All rights reserved.
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options
SQL> show parameter instance_name
NAME              TYPE         VALUE
-------------     --------     ---------
instance_name     string       racdb1
```

继续开启第二个 sqlplus 连接，执行如下操作：

```
[oracle@client ~]$ sqlplus system/xxxxxx@racdb
SQL*Plus: Release 11.1.0.7.0 - Production on Sun Sep 12 21:31:53 2010
Copyright (c) 1982, 2008, Oracle.  All rights reserved.
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options
SQL> show parameter instance_name

NAME                 TYPE        VALUE
--------------       ---------   ---------
instance_name        string      racdb2
```

按照这种方法，陆续打开多个 sqlplus 连接，可以看到，每次连接到的实例都在 racdb1 和 racdb2 之间变化，这样就实现了 RAC 数据库连接的负载均衡。

2. 服务器端的负载均衡

客户端的负载均衡解决了连接数据库的负载问题，但是由于连接是由客户端发起的，它并不知道 RAC 数据库集群中各个节点的繁忙状态和连接信息，因此负荷较大的节点仍然会增加新的连接，这样就可能导致 RAC 节点无法真正做到负载均衡。不过幸运的是，从 Oracle 10g 开始，服务器端负载均衡就可以根据 RAC 中各节点的负荷及连接数情况，将新的请求分配到集群中负载较低、连接数较少的节点上来，这样就从根本上实现了数据库的负载均衡，并且使客户端连接的负载均衡与服务器端的负载均衡可以配合使用，互不影响。

每个集群节点的负载情况是由 PMON 进程来定期更新的。PMON 进程每 3 秒会将集群中每个节点的负载信息及连接数写入 service_register 中，当节点的负载发生变化时，将会立刻通知监听程序，最后由监听程序来决定将新的客户端连接分配到哪个节点上，通过这种方式，RAC 数据库实现了真正的负载均衡。

服务器端负载均衡配置也非常简单，只需在各节点的 tnsnames.ora 文件中添加一个对连接到各个节点进行监听的配置，然后在初始化参数中设置 remote_listener 即可。

（1）修改服务器端的 tnsnames.ora

只需添加如下内容即可：

```
LISTENERS_RACDB =
  (ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP)(HOST = node-vip2)(PORT = 1521))
(ADDRESS = (PROTOCOL = TCP)(HOST = node-vip1)(PORT = 1521))
  )
```

（2）修改参数 remote_listener

查看 RAC 数据库的参数 remote_listener：

```
SQL> show parameter remote_listener
NAME              TYPE        VALUE
------------      -------     ---------
```

```
remote_listener    string    LISTENERS_RACDB
```

可以看到，remote_listener 已经设置为“LISTENERS_RACDB”了。

如果 remote_listener 的值为空，可以通过如下命令修改每个实例的 remote_listener 参数：

```
SQL> alter system set remote_listener='LISTENERS_RACDB' sid='node-rac1';
SQL> alter system set remote_listener='LISTENERS_RACDB' sid='node-rac2';
```

这样，服务器端的负载均衡就配置完成了。

13.8.2 透明应用失败切换测试

透明应用失败切换（Transparent Application Failover，TAF），这是客户端的一种功能。TAF 包含两层意思：失败切换是指客户端连接到某个实例，如果连接失败，可以连接到另外一个实例；透明应用是指客户端应用程序在连接失败后可以自动重新连接到另一个数据库实例，而这个过程对应用程序是不可见的。

要使用 TAF 功能，只需修改客户端的 tnsnames.ora 文件中的设置即可。结合前面介绍的客户端负载均衡功能，一个包含负载均衡和 TAF 功能的客户端设置如下：

```
RACDB =
  (DESCRIPTION =
  (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = TCP)(HOST = node-vip2)(PORT = 1521))
     (ADDRESS = (PROTOCOL = TCP)(HOST = node-vip1)(PORT = 1521))
     (LOAD_BALANCE = yes)
     )
     (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = racdb)
                   (FAILOVER_MODE =
                   (TYPE=SELECT)
                   (MODE=BASIC)
                   (RETRY=3)
                   (DEALY=5)
                   )
            )
  )
```

其中的几个参数的含义如下：

- TYPE，用于指定 FAILOVER_MODE 的类型，有 3 种类型可选，分别是 SESSION、SELECT 和 NONE。
  - SESSION，表示当一个正在连接的会话实例发生故障时，系统可以自动将会话切换到其他可用的实例，而应用程序无需再次发起连接请求，但是实例故障时正在执行的 SQL 需要重新执行。
  - SELECT，表示如果正在连接的实例发生故障，将使用游标和之前的快照继续执行

SELECT 操作，其他操作必须要重新执行。

- NONE，这个是客户端默认值，表示禁止 SQL 接管功能。

- MODE，表示连接模式，有两种类型，分别是 BASIC 和 PRECONNECT。
  - BASIC 表示在建立初始连接时仅连接到一个节点，并且只有在发生节点故障时才连接到备用节点。
  - PRECONNECT 表示在建立初始连接时就连接到主节点和备用节点。
- RETRY：表示当前节点失败后，失败切换功能尝试连接备用节点的次数。
- DELAY：表示两次尝试之间等待的秒数。

设置完客户端监听后，重启客户端服务，然后执行下面的操作：

```
[oracle@client ~]$sqlplus system/xxxxxx@racdb
SQL*Plus: Release 11.1.0.7.0 - Production on Sun Sep 12 23:23:15 2010
Copyright (c) 1982, 2008, Oracle.  All rights reserved.
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options
SQL> COLUMN instance_name FORMAT a10
SQL> COLUMN host_name FORMAT a10
SQL> COLUMN failover_method FORMAT a15
SQL> COLUMN failed_over FORMAT a10
SQL> SELECT instance_name, host_name, NULL AS failover_type, NULL AS failover_
    method, NULL AS failed_over FROM v$instance UNION SELECT NULL, NULL, failover_
    type , failover_method, failed_over FROM v$session WHERE username = 'SYSTEM';

INSTANCE_ NAME  HOST_NAME   FAILOVER_TYPE   FAILOVER_METHOD   FAILED_OVER
----------      ----------  ----------      -------------     ------------
racdb2          node-rac2   SELECT          BASIC             NO
```

此时，不断开此连接，然后在 RAC 数据库的任意一个节点上执行如下语句：

```
[oracle@node-rac2 ~]$ srvctl stop instance -d racdb -i racdb2
```

关闭 node-rac2 节点的 racdb2 实例后，继续执行与前面那个 SQL 命令相同的语句，结果如下：

```
INSTANCE_ NAME  HOST_NAME   FAILOVER_TYPE   FAILOVER_METHOD   FAILED_OVER
----------      --------    ----------      -----------       -------------
racdb1          node-rac1   SELECT          BASIC              YES
```

从输出可以看到，上面的 SQL 会话已经切换到了 node-rac1 的实例 racdb1 上，也就是实现了故障自动切换功能。

至此，关于 RAC 数据库的功能测试已经验证完毕了。

13.9 本章小结

本章主要讲述了 Oracle RAC 集群系统的搭建、使用和维护过程。首先介绍了 RAC 数据

库的体系结构和进程；然后深入介绍了安装 Oracle RAC 数据库的过程，具体包括安装前期的系统设置，安装 Oracle ClusterWare，安装与创建 Oracle RAC 数据库；接着介绍了 Oracle 高可用软件 CRS 的管理与维护；接下来介绍了 ASM 的基本操作与维护；最后介绍了如何使用 srvctl 管理和维护 RAC 数据库。

纵观全章，安装 Oracle RAC 数据库是本章学习的重点，另外，CRS 软件的维护、ASM 的基本操作也是要掌握的基本内容，而 RAC 数据库的体系结构与特点只需了解即可。

目前 Oracle RAC 集群构架方案已经广泛应用于各个行业，通过这个方案不仅可以满足高端业务的需求，还能够降低管理和投入成本。如果您的数据库在性能方面存在问题，不妨尝试一下 Oracle 的 RAC 数据库。

第 14 章　构建 MySQL+heartbeat+DRBD+LVS 集群应用系统

本章主要介绍 MySQL 高可用集群环境的搭建和应用。在企业应用中，MySQL+heartbeat+DRBD+LVS 是一套成熟的集群解决方案，通过 heartbeat+DRBD 完成 MySQL 的主节点写操作的高可用性，而通过 MySQL+LVS 实现 MySQL 数据库的主从复制和 MySQL 读操作的负载均衡。整个方案在读写方面进行了分离，融合了写操作的高可用和读操作的负载均衡，是一个完美而又廉价的企业应用解决方案。各大门户网站（如新浪等）都在利用此方案搭建 MySQL 集群环境。

14.1　MySQL 高可用集群概述

数据库作为最基础的数据存储服务之一，在整个系统中有着非常重要的地位，需要具备高可用性是无可厚非的。有很多解决方案能实现不同的 SLA（服务水平协定），这些方案可以保证数据库服务器在硬件或软件出现故障时服务继续可用。

目前比较流行的高可用解决方案有如下几种：

- MySQL 的复制功能是通过在建立复制关系的两台或多台机器环境中，一台机器出现故障就切换到另一台机器上来保证一定程度的可用性，可以实现 90.000% 的 SLA。
- MySQL 的复制功能加一些集群软件可以实现 95.000% 的 SLA。
- MySQL+heartbeat+DRBD 的复制功能可以实现 99.900% 的 SLA。
- 共享存储 + MySQL 的复制功能可以实现 99.990% 的 SLA。
- MySQL Cluster 的标准版和电信版可以达到 99.999% 的 SLA。

这几种方案与 SLA 的关系如图 14-1 所示。

图 14-1　MySQL 几种高可用解决方案和 SLA 的关系

在企业级应用中，对于 MySQL 来说，使用共享存储的相对较少，使用最多的方案是 heartbeat+DRBD 和 MySQL Cluster 的方案。对于 Oracle 的 RAC 来说，使用的是共享存储的结构方式，同时将 heartbeat+DRBD 作为 Oracle 公司为客户提供解决方案的服务之一。

14.2　heartbeat + DRBD 高可用性方案的实现原理

DRBD 的英文全称为 Distributed Replicated Block Device（分布式块设备复制），是 Linux 内核的存储层中的一个分布式存储系统，可利用 DRBD 在两台 Linux 服务器之间共享块设备、文件系统和数据，类似于一个网络 RAID1 的功能。DRBD 的架构如图 14-2 所示。

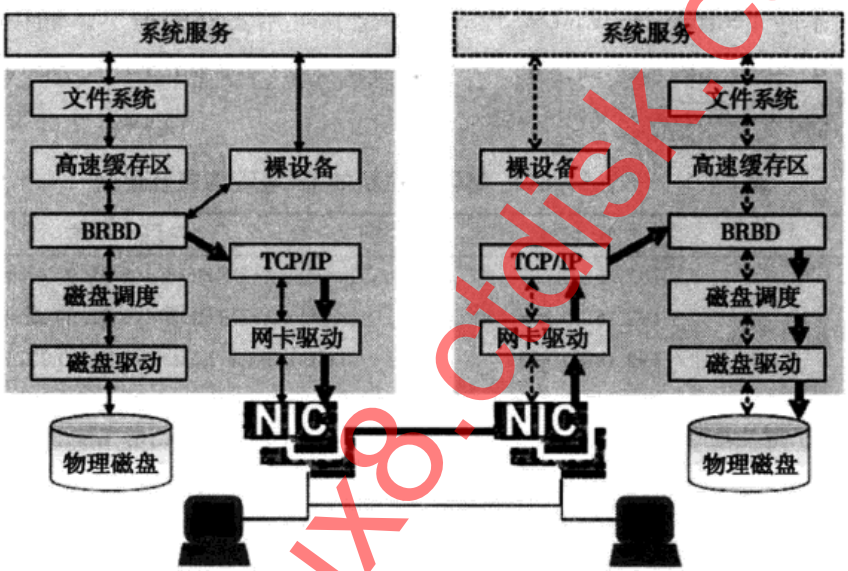

图 14-2　DRBD 的架构图

当将数据写入到本地主节点的文件系统时，这些数据会通过网络发送到另一台主节点上。本地主节点和远程主节点数据通过 TCP/IP 协议保持同步，主节点故障时，远程节点保存着相同的数据，可以接替主节点继续提供数据。两个节点之间使用 heartbeat 来检测对方是否存活。

同步过程如下：

1）在 NODE1 上写操作被提交，然后通过内核传给 DRBD 模块。

2）DRBD 发送写操作到 NODE2。

3）在 NODE2 上的 DRBD 发送写操作给本地磁盘。

4）在 NODE2 上的 DRBD 向 NODE1 发确认信息，确认已经接收到写操作并发送给本地磁盘。

5）在 NODE1 上的 DRBD 发送写操作给本地磁盘。

6）NODE1 的内核回应写操作完成。

此同步过程还依赖于 DRBD 的 3 种同步协议：

- Protocol A，写 I/O 到达本地磁盘和本地的 TCP 发送缓存区之后，返回操作成功。
- Protocol B，写 I/O 到达本地磁盘和远程节点的缓存区之后，返回操作成功。
- Protocol C，写 I/O 到达本地磁盘和远程节点的磁盘之后，返回操作成功。

14.3 部署 MySQL 高可用高扩展集群

企业级 MySQL 集群具备高可用、可扩展、易管理、低成本的特点，通常采用 MySQL 读写分离的办法，而读写之间的数据同步采用 MySQL 的单向或双向复制技术实现。MySQL 写操作一般采用基于 heartbeat+DRBD+MySQL 搭建高可用集群的方案，而读操作普遍采用基于 LVS+Keepalived 搭建高可用高扩展集群的方案。部署结构如图 14-3 所示，主机的 IP 地址信息及用途如表 14-1 所示。

表 14-1 一个 MySQL 集群中主机的 IP 地址信息及用途

主机名	IP 地址	用 途
dbm157	192.168.0.157	heartbeat+DRBD+MySQL 的 primary 节点
dbm158	192.168.0.158	heartbeat+DRBD+MySQL 的 secondary 节点
dbs159	192.168.0.159	slave(master_host=192.168.0.222)
dbs160	192.168.0.160	slave(master_host=192.168.0.222)
dbs161	192.168.0.161	slave(master_host=192.168.0.222)
LVS1	192.168.0.146	LVS+Keepalived
LVS2	192.168.0.147	LVS+Keepalived

heartbeat+DRBD 方案中的 IP 信息如表 14-2 所示。

表 14-2 heartbeat+DRBD 方案中的 IP 信息

heartbeat 使用的 VIP	heartbeat 主、备节点
192.168.0.222	192.168.0.157(primary) 192.168.0.158(secondary)

LVS+Keepalived 上 VIP 和 realserver 的 IP 信息如表 14-3 所示。

表 14-3 LVS+Keepalived 上 VIP 和 realserver 的 IP 信息

VIP	realserver
192.168.0.223	192.168.0.159 192.168.0.160 192.168.0.161

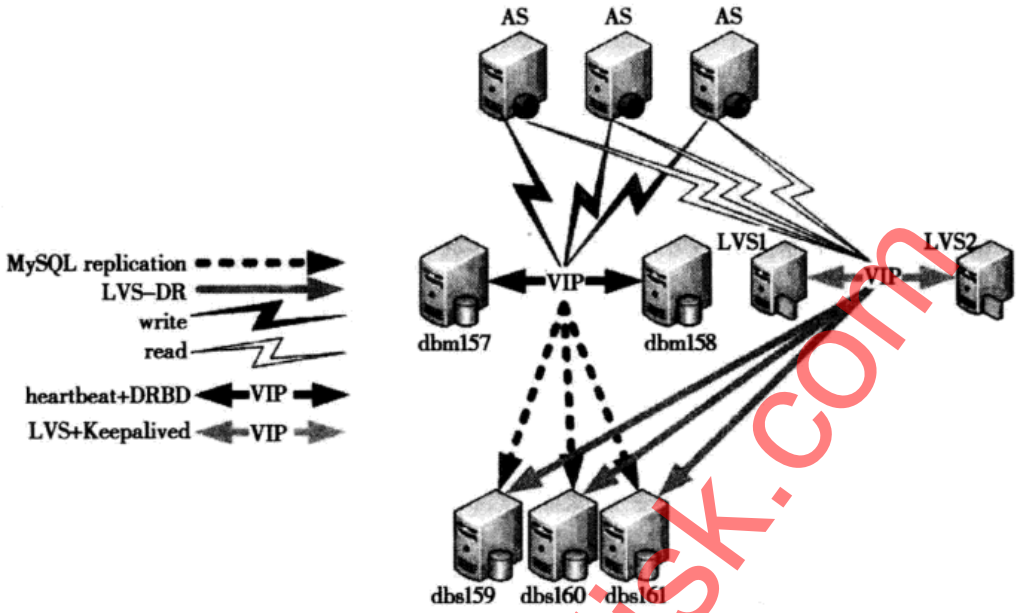

图 14-3　MySQL 集群的总体结构

14.3.1　配置之前的准备

1. 设置 hostname 及解析

首先要给需要配置 DRBD 的两台主机配置 hostname，设置之后通过“uname –n”检查设置是否成功。建议不要将 hostname 设置为全称域名（例如，dbm157.example.com、dbm158.example.com），设置短标识比较好（例如，dbm157、dbm158），因为这个 hostname 在以后的配置中会用到，较短的标识可以减少后期配置的复杂度，并避免出现问题。同时要求这两台主机的 hostname 能分别解析到两台机器的内网 IP（即两台主机互相通信时使用的网络地址）。

编辑 /etc/hosts 文件，添加如下内容：

```
192.168.0.157   dbm157
192.168.0.158   dbm158
```

2. 磁盘分区规划

根据当前 DB 文件的大小及后期的增长划分出一个分区，用于存放所有可变数据文件，例如，整个 MySQL 的 datadir 目录、binlog 文件、relay log 文件、my.cnf 文件，还包括所有表的 ibd 文件（单独表空间）、ibdata 文件和 ib_logfile 文件。两台主机节点分区最好大小一样，至少要保证 secondary 节点分区尺寸大于 primary 节点分区尺寸，以避免后期因空间不足而出现难以预见的问题。

3. 熟悉网络环境

确定利用 DRBD 进行同步的网络状况是否良好。DRBD 同步操作对网络环境要求很高，特别是在写入数据量特别大、需要同步的数据很多时尤为重要。网络环境越好，同步的速度相对越快。可以考虑把 DB 对外提供服务的网络和 DRBD 同步网络分开，这样可以使业务对 DB 的访问请求和 DRBD 同步互不影响，但也会带来一些成本的增加。规划两台主机之间的心跳线为两根以上，这样能保证不会因为某一线路故障而产生切换工作。在高可用方案中，一般要求用三根以上心跳线进行心跳检测，以此来减少误切换和“脑裂”问题，同时要确认上层交换机是否禁止 ARP 广播。

14.3.2 DRBD 的部署

从官方网站下载源码包来编译或直接使用 yum 来安装，这里以 CentOS 为例说明安装过程，其他系统类似。

```
[root@dbm157 ~]# uname -a
Linux dbm157 2.6.18-194.8.1.el5 #1 SMP Thu Jul 1 19:04:48 EDT 2010 x86_64 x86_64
    x86_64 GNU/Linux
```

通过 yum 安装 DRBD 服务：

```
[root@dbm157 ~]# yum -y install kmod-drbd83 drbd83
```

检查 DRBD 是否安装成功：

```
[root@dbm157 ~]# modprobe -l | grep -i drbd
/lib/modules/2.6.18-194.8.1.el5/weak-updates/drbd83/drbd.ko
[root@dbm157 ~]# lsmod | grep -i drbd
drbd                  277272  4
```

安装成功之后，在 /sbin 目录下面有 drbdadm、drbdmeta、drbdsetup 命令，以及 /etc/init.d/drbd 启动脚本。

14.3.3 DRBD 的配置

1. DRBD 使用的硬盘分区

前面已经说过，最好使用相同尺寸的单独分区，同时考虑 DB 的大小和未来的增长量。也可以使用 LVM 进行分区。只有保证两台机器型号或性能一样好，才能保证在切换后 secondary 节点能完成原来 primary 节点承担的业务负载。这里分配大小为 145GB 的 /database 分区给数据库使用，两台机器完全一样。过程如下：

```
[root@dbm157 ~]# df -h | grep database
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2             145G  3.1G  134G   3% /database
```

```
[root@dbm158 ~]# df -h | grep database
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2             145G  3.1G  134G   3% /database
```

2. drbd.conf 配置文件

DRBD 运行需要读取 /etc/drbd.conf 配置文件，可以通过如下命令重建这个配置文件，该文件中描述了 DRBD 设备与硬盘分区的映射关系和 DRBD 的一些配置参数。

```
[root@dbm158 ~]# cp /usr/share/doc/drbd83-8.3.8/drbd.conf /etc/drbd.conf
```

两台主机节点的相关信息如表 14-4 所示。

表 14-4　两台主机节点的相关信息

节点角色	主机名	IP 地址	DRBD 分区
primary 节点	dbm157	192.168.0.157	/dev/sda2
secondary 节点	dbm158	192.168.0.158	/dev/sda2

下面是两台主机节点上 drbd.conf 文件的简单示例：

```
[root@dbm158 ~]# cat /etc/drbd.conf
#
# drbd.conf
#
# create by jackbillow@gmail.com at 2010-08-12
global {
    # minor-count 64;
    # dialog-refresh 5; # 5 seconds
    # disable-ip-verification;
    usage-count no;
    # 是否参加 DRBD 使用者统计，默认为 yes
}

common {
  syncer { rate 200M; }
   # 设置主、备节点同步时的网络速率最大值，单位是字节
}

resource r0 {
   # 资源名为 r0
  protocol C;
   # 使用 DRBD 的第三种同步协议，表示收到远程主机的写入确认后认为写入完成
  handlers {
    pri-on-incon-degr "echo o > /proc/sysrq-trigger ; halt -f";
    pri-lost-after-sb "echo o > /proc/sysrq-trigger ; halt -f";
    local-io-error "echo o > /proc/sysrq-trigger ; halt -f";
    fence-peer "/usr/lib64/heartbeat/drbd-peer-outdater -t 5";
    pri-lost "echo pri-lost. Have a look at the log files. | mail -s 'DRBD Alert' root";
    split-brain "/usr/lib/drbd/notify-split-brain.sh root";
    out-of-sync "/usr/lib/drbd/notify-out-of-sync.sh root";
```

```
}

net {
    # timeout            60;
    # connect-int        10;
    # ping-int           10;
    # max-buffers      2048;
    # max-epoch-size   2048;
    cram-hmac-alg "sha1";
    shared-secret "MySQL-HA";
    # DRBD同步时使用的验证方式和密码信息
}

disk {
    on-io-error detach;
    fencing resource-only;
    #使用dopd (drbd outdate-peer daemon) 功能保证在数据不同步时不进行切换
}

startup {
    wfc-timeout 120;
    degr-wfc-timeout 120;
}

device        /dev/drbd0;

on dbm157 {
    #每个主机的说明以on开头，后面是hostname (uname -n)，其后的{}中是这个主机的配置
    disk          /dev/sda2;
    #/dev/drbd0使用的磁盘分区是/dev/sda2
    address       192.168.0.157:7788;
    #设置DRBD的监听端口，用于与另一台主机通信
    meta-disk     internal;
}
on dbm158 {
    disk          /dev/sda2;
    #/dev/drbd0使用的磁盘分区是/dev/sda2
    address       192.168.0.158:7788;
    #设置DRBD的监听端口，用于与另一台主机通信
    meta-disk     internal;   #drbd的元数据存放方式
}

}
```

将上面的drbd.conf文件分别复制到两台主机的/etc目录下。drbd.conf的配置参数很多，有兴趣的读者可以使用man drbd.conf来了解更多的参数说明。

3. DRBD的启动

启动DRBD服务之前，首先分别在两台主机的/dev/sda2分区上创建DRBD元数据库信

息。执行的命令如下：

```
[root@dbm157 ~]# drbdadm create-md all
Writing meta data...
initializing activity log
NOT initialized bitmap
New drbd meta data block successfully created.
[root@dbm158 ~]# drbdadm create-md all
Writing meta data...
initializing activity log
NOT initialized bitmap
New drbd meta data block successfully created.
```

这里也可以用 drbdadm create-md r0 代替 drbdadm create-md all，r0 是在 drbd.conf 中定义的资源名称。现在我们可以启动 DRBD 了，分别在两台主机上执行启动操作。这一步操作有可能出现创建不成功的情况。

```
[root@dbm157 ~]# drbdadm create-md all
md_offset 151182448
al_offset 151182949
bm_offset 151081421

Found ext3 filesystem which uses 151182448 kB
current configuration leaves usable 151081223 kB

Device size would be truncated, which
would corrupt data and result in
'access beyond end of device' errors.
You need to either
   * use external meta data (recommended)
   * shrink that filesystem first
   * zero out the device (destroy the filesystem)
Operation refused.

Command 'drbdmeta 0 v08 /dev/sda2 internal create-md' terminated with exit code 40
drbdadm create-md data0: exited with code 40
```

这时需要使用如下命令覆盖文件系统中的设备块信息，操作时确认此分区上的数据已经备份过。

```
dd if=/dev/zero of=/dev/sda2 bs=1M count=128
```

执行完“dd”命令后，再次执行“drbdadm create-md all”命令。

启动 DRBD 服务，设置主节点后格式化主节点的 DRBD 分区。

```
[root@dbm157 ~]# /etc/init.d/drbd start
Starting DRBD resources: [ d(data0) s(data0) n(data0) ].
```

可以通过 dmesg 命令查看 DRBD 的启动过程。

```
[root@dbm157 ~]# dmesg | grep drbd
drbd: initialized. Version: 8.3.8 (api:88/proto:86-94)
```

```
drbd: GIT-hash: d78846e52224fd00562f7c225bcc25b2d422321d build by mockbuild@
    builder10.centos.org, 2010-06-04 08:04:09
drbd: registered as block device major 147
drbd: minor_table @ 0xffff81042f8f8bc0
block drbd0: Starting worker thread (from cqueue/3 [306])
block drbd0: disk( Diskless -> Attaching )
block drbd0: Found 4 transactions (192 active extents) in activity log.
block drbd0: Method to ensure write ordering: barrier
block drbd0: max_segment_size ( = BIO size ) = 32768
block drbd0: drbd_bm_resize called with capacity == 307185480
block drbd0: resync bitmap: bits=38398185 words=599972
block drbd0: size = 146 GB (153592740 KB)
block drbd0: recounting of set bits took additional 6 jiffies
block drbd0: 508 MB (130048 bits) marked out-of-sync by on disk bit-map.
block drbd0: disk( Attaching -> UpToDate ) pdsk( DUnknown -> Outdated )
block drbd0: conn( StandAlone -> Unconnected )
block drbd0: Starting receiver thread (from drbd0_worker [2978])
block drbd0: receiver (re)started
block drbd0: conn( Unconnected -> WFConnection )
block drbd0: Handshake successful: Agreed network protocol version 94
block drbd0: Peer authenticated using 20 bytes of 'sha1' HMAC
block drbd0: conn( WFConnection -> WFReportParams )
block drbd0: Starting asender thread (from drbd0_receiver [2986])
block drbd0: data-integrity-alg: <not-used>
block drbd0: drbd_sync_handshake:
block drbd0: self F4E0D8AA07CA1018:38574988F95CFEB3:791FC3B0E66556FC:3C3B5430B17B
    0028 bits:130048 flags:0
block drbd0: peer 38574988F95CFEB2:0000000000000000:791FC3B0E66556FD:3C3B5430B17B
    0028 bits:0 flags:0
block drbd0: uuid_compare()=1 by rule 70
block drbd0: peer( Unknown -> Secondary ) conn( WFReportParams -> WFBitMapS )
    pdsk( Outdated -> UpToDate )
block drbd0: conn( WFBitMapS -> SyncSource ) pdsk( UpToDate -> Inconsistent )
block drbd0: Began resync as SyncSource (will sync 520192 KB [130048 bits set]).
block drbd0: Resync done (total 49 sec; paused 0 sec; 10616 K/sec)
block drbd0: conn( SyncSource -> Connected ) pdsk( Inconsistent -> UpToDate )
```

然后执行如下命令：

```
[root@dbm157 ~]# drbdadm primary all
[root@dbm157 ~]# drbdadm -- --overwrite-data-of-peer primary all   #如果上一步执行不成功
#则执行此命令把此节点设置为 primary 节点，从头开始同步
[root@dbm157 ~]# mkfs.ext3 /dev/drbd0
```

接着把dbm157机器上的drbd.conf文件通过scp命令传送到dbm158机器上并启动DRBD服务。执行过程如下：

```
[root@dbm157 ~]# scp /etc/drbd.conf root@192.168.0.158:/etc/drbd.conf
[root@dbm158 ~]# /etc/init.d/drbd start
Starting DRBD resources: [ d(data0) s(data0) n(data0) ].
```

如果当前是 Secondary 状态，可以通过命令“drbdadm primary all”把当前主机更改为 Primary 状态。drbdadm 命令是 DRBD 的管理命令，它的很多参数可以用来管理 DRBD 同步或停止、网络断开或连接等各种状态的转化。

1. 挂载 DRBD 分区到 /database 目录

```
[root@dbm157 ~]# mount /dev/drbd0 /database
[root@dbm157 ~]# df -h | grep database
Filesystem            Size  Used Avail Use% Mounted on
/dev/drbd0            145G  3.1G  134G   3% /database
```

2. DRBD 设备角色切换

DRBD 设备在进行角色切换操作前，需要先在主节点上执行 umount 命令，去掉对 DRBD 设备的挂载，然后在另一台主机上把 DRBD 角色修改为 Primary，最后再执行挂载。操作如下：

```
[root@dbm157 ~]# df -h | grep database
Filesystem            Size  Used Avail Use% Mounted on
/dev/drbd0            145G  3.1G  134G   3% /database
[root@dbm157 ~]# umount /database
[root@dbm157 ~]#drbdadm secondary all
```

接着在 dbm158 主机上执行如下操作：

```
[root@dbm158 ~]#drbdadm primary all
[root@dbm158 ~]# mount /dev/drbd0 /database
```

还有一种切换策略，先停止 dbm157 主机的 drbd 服务：

```
[root@dbm157 ~]#/etc/init.d/drbd stop
```

然后在节点 dbm158 上执行如下操作：

```
[root@dbm158 ~]# drbdadm ----overwrite-data-of-peer primary all
[root@dbm158 ~]# mount /dev/drbd0 /database
```

14.3.5 DRBD 的性能优化

可以考虑从以下几个方面优化 DRBD 性能。

（1）网络环境

能使用千兆网卡的不要使用百兆网卡。当前主流机器都使用千兆网卡，交换机也不例外。同时，DRBD 的数据同步使用的网络最好和提供服务的网络分开，尽量独立出来。例如，在两块网卡上直接连接一个网线，用做 DRBD 的数据同步。

（2）用做 DRBD 分区的磁盘的性能

用做 DRBD 分区的磁盘的性能要尽量好，例如可以考虑使用不少于 6 块 15KB 的 SAS 盘作为 RAID10 或 RAID0（最好用 BBU），以提供 I/O 性能。在网络环境很好的情况下，DRBD 分区可能会由于 I/O 的写性能而成为瓶颈。

（3）更新系统

尽量把系统更新成最新的内核以及 64 位的系统，同时使用最新版本的 DRBD。目前，kernel2.6.13 已经准备把 DRBD 作为 Linux 内核的主干分支。

（4）注意 syncer 参数设置

syncer 主要用来设置同步相关参数。可以设置“重新”同步（re-synchronization）的速率（rate），当节点间出现不一致的 block 时，DRBD 就需要执行 re-synchronization 动作，而 syncer 中的参数 rate 就是用来设置同步的速率的，rate 的设置与网络和磁盘 I/O 能力密切相关。

千兆网络的同步速率大约是 125Mbit/s，百兆网络的同步速率大约是 11Mbit/s，但笔者测试到的同步速率最大能到 218Mbit/s。用这个同步速率和磁盘写入速率（hdparm -Tt /dev/drbd0 测试结果）中最小者的 30% 带宽来设置 re-synchronization 是比较合适的，这也是官方给出的建议。

例如，同步速率为 125Mbit/s，磁盘写入速度为 110Mbit/s，应该设置 rate 为不能超过 33Mbit/s。这样设置的原因是：DRBD 同步由两个不同的进程来负责：一个 replication 进程用来同步 block 的更新，这个值受限于参数设置；一个 synchronization 进程用来同步元数据信息，这个值不受参数设置限制。如果写入量非常大，设置的参数超过磁盘的写入速率，元数据的同步速率就会受干扰，传输速度变慢，导致机器负载非常高，性能下降得非常厉害，所以这个值应该根据实际环境来进行设置，如果设置得太大，就会把所有的带宽占满了，导致 replication 进程没有可用带宽，最终可能会导致 I/O 停止，出现同步不正常的现象。

（5）注意 al-extents 参数设置

al-extents 控制着一次向磁盘写入多少个 4MB 的数据块。增大这个参数的值有以下几个好处：

- 可以减少更新元数据到 DRBD 设备的频率。
- 降低同步数据时对 I/O 流的中断数量。
- 提高修改 DRBD 设备的速度。

但同时也存在一个风险：当主节点出现宕机时，所有活动的数据（al-extends 的值 x 4M 的数据块）需要在同步连接建立后重新同步，即在主节点出现宕机时，备用节点出现数据不一致（outdate）的情况。因此，不建议在 HA 部署上调整这个参数，可以在某些情况下调整这个参数来提高性能。

总的来说，以上 5 个方面需要特别注意，调整其他参数影响则较小。

14.3.6　MySQL 的部署

1. MySQL 的安装与配置

安装 MySQL 有多种方法，这里仅以利用 rpm 安装 MySQL 为例进行说明。

```
[root@dbm157~]#yum -y install mysql-server mysql-devel mysql mysql-bench mysql-test
```

安装完成后，使用如下命令启动 MySQL 服务：

```
[root@dbm157 ~]# /etc/init.d/mysqld start
```

将数据文件放到 DRBD 分区上：

```
[root@dbm157 ~]# cp -R /var/lib/mysql /database/mysql
[root@dbm157 ~]# chown -R mysql:mysql /database/mysql
```

修改 /etc/my.cnf 文件，在 [mysqld] 组增加如下配置：

```
datadir = /database/mysql
```

在 heartbeat 资源脚本目录中建立 MySQL 启动脚本的软连接：

```
[root@dbm157 ~]# ln -s /etc/init.d/mysqld /etc/ha.d/resource.d/mysqld
```

重启 MySQL 服务：

```
[root@dbm157 ~]# /etc/init.d/mysqld restart
Shutting down MySQL...                                     [  OK  ]
Starting MySQL.                                            [  OK  ]
```

2.MySQL 主从复制的配置

MySQL 的复制（replication）是异步复制，即从一个 MySQL 实例或端口（称之为 Master）复制到另一个 MySQL 实例或端口（称之 Slave）。复制操作由 3 个进程完成，其中两个进程（SQL 进程和 I/O 进程）在 Slave 上，另外一个进程在 Master（binlog dump）上。

要实现复制，必须打开 Master 端的二进制日志（log-bin）功能。这是因为整个复制过程实际上就是 Slave 从 Master 端获取该更新操作的日志，将其传输到本地并写到本地文件中，然后再读取本地文件内容执行日志中所记录的更新操作，如图 14-4 所示。

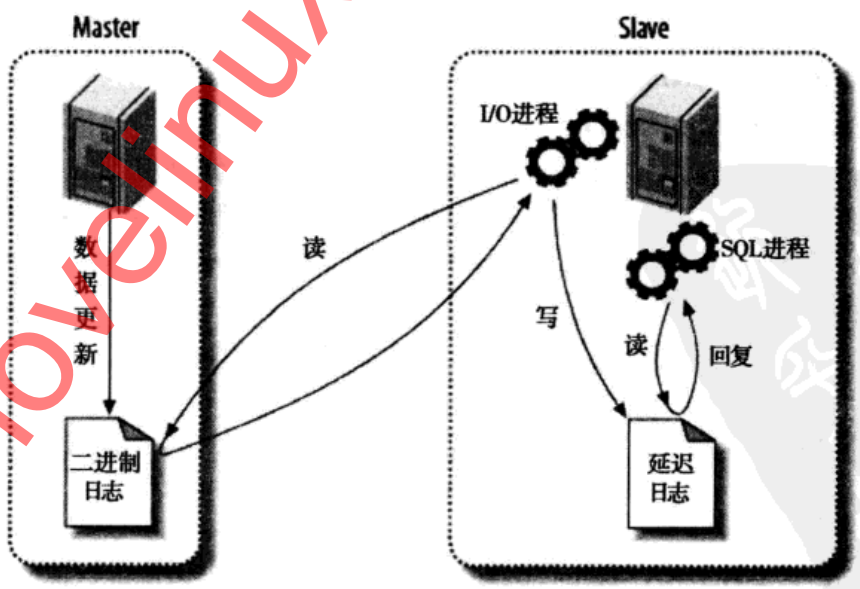

图 14-4 MySQL 主从复制的原理图

不同版本的 MySQL 二进制日志在某些语句上有些差别，因此最好是 Master 和 Slave 的 MySQL 版本相同，或者 Master 的版本不高于 Slave 的版本。

这里以 CentOS 为例演示配置过程。

（1）在 Master 上开启二进制日志同时建立同步需要的账号

每个同步服务器都必须设定一个唯一的编号，修改 my.cnf，增加或修改如下两行：

```
server-id = 1   # 相同端口必须唯一
log-bin         # 开启记录二进制日志功能
```

在 Master（这里为 dbm157 机器）上增加一个用于复制的账号：

```
mysql>GRANT REPLICATION SLAVE ON *.* TO 'repl_user'@'slave_ip' IDENTIFIED BY
    'repl_password';
```

备份 Master 上的数据，首先执行如下 SQL 语句：

```
mysql>FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.00 sec)
mysql> reset master;    # 把 Master 的 position 设置为 98（MySQL 5.1 版本是 106）
Query OK, 0 rows affected (0.00 sec)
```

不要退出终端，否则这个锁就失效了。在不退出终端的情况下，再开启一个终端直接打包压缩数据文件或使用 mysqldump 工具来导出数据。

```
[root@dbm157 ~]# cd /var/lib/     # 进入 MySQL 的数据目录，根据自己的情况而定
[root@dbm157 lib]# tar zcvf mysql.tar.gz mysql
[root@dbm157 lib]# scp mysql.tar.gz  192.168.0.158:/var/lib/
```

用 scp 命令把打包的数据传输到其他几台 Slave 机器上。

数据传输完成后，在执行 FLUSH TABLES WITH READ LOCK 命令的终端上执行如下命令：

```
mysql>UNLOCK TABLES;
```

（2）设置 Slave 主机

修改 my.cnf 的 server-id，内容如下：

```
server-id = 2
```

其他 Slave 上的修改以此类推，保证 server-id 全局唯一。

（3）开启 Master 与 Slave 的同步

在 Slave 上执行如下命令：

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.0.157',
-> MASTER_USER='repl_user',
-> MASTER_PASSWORD='repl_password',
-> MASTER_LOG_FILE='mysql-bin.000001',
-> MASTER_LOG_POS=98;
```

之后执行：

```
mysql> slave start;
Query OK, 0 rows affected (0.00 sec)
mysql> show slave status\G
*************************** 1. row ***************************
      Slave_IO_State: Waiting for master to send event
         Master_Host: 192.168.0.157
         Master_User: repl_user
         Master_Port: 3306
       Connect_Retry: 60
     Master_Log_File: mysql-bin.000001
 Read_Master_Log_Pos: 98
      Relay_Log_File: mysql-relay-bin.000001
       Relay_Log_Pos: 225
   Relay_Master_Log_File: mysql-bin.000001
    Slave_IO_Running: Yes
   Slave_SQL_Running: Yes
     Replicate_Do_DB:
 Replicate_Ignore_DB:
  Replicate_Do_Table:
  Replicate_Ignore_Table:
 Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
          Last_Errno: 0
          Last_Error:
        Skip_Counter: 0
 Exec_Master_Log_Pos: 98
     Relay_Log_Space: 225
     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:
1 row in set (0.00 sec)
```

在其他几台 SLave 机器上也执行上面的命令。

从输出中可以看到：Slave_IO_Running 和 Slave_SQL_Running 都为 Yes 时，表示配置成功。

很多人喜欢把 master_host、master_user、master_password、master_port 写到 Slave 主机的 my.cnf 里面，笔者不建议这么做，因为在 Master 出现故障进行切换之后，可能会忘记曾

修改过这些信息，虽然这些信息没有 master.info 的优先级高，用处不大，但是会让人产生误解，所以建议多使用 master.info 文件内容。

3. 需要注意的几个问题

1）如果在 my.cnf 中定义了 log-bin、relay-log 参数，那么要保证这些定义与主机名无关，因为如果这两类 log 的文件名与主机名有关，切换过程会导致 Slave 主机不能继续同步。例如可以如下设置：

```
log-bin = mysql-bin
relay-log = mysql-relay-bin
```

保证在两台主机上两个文件的名字一样。

2）最好把 my.cnf 文件也放入 DRBD 分区的数据目录中，这样在进行配置变更时，另一台也保持同步，避免由于修改文件导致切换后配置不一样。要把 my.cnf 文件放入 DRBD 分区的数据目录中，需要修改 /etc/init.d/mysqld 启动脚本中 my.cnf 文件的路径。

3）如果不是通过 rpm 安装 MySQL，要保证 MySQL 启动脚本能接收 start、top、status 三个参数。默认 heartbeat 采用的是 LSB（Linux Standard Base）风格，返回值包含 OK 或 running 则表示资源正常，返回值包含 stopped 或 No 则表示资源不正常。其他资源脚本的情况类似。

4）不要设置 mysqld 在机器重启动时自动启动，mysqld 服务作为 heartbeat 的一项资源会统一管理。

14.3.7　heartbeat 的部署

通过 yum 安装 heartbeat 服务：

```
[root@dbm157~]#yum -y install heartbeat heartbeat-devel heartbeat-stonith
    heartbeat-pils
```

创建 heartbeat 的以下配置文件：

- ha.cf 是 heartbeat 的主配置文件。

```
[root@dbm157 ~]# cp /usr/share/doc/heartbeat-2.1.3/ha.cf /etc/ha.d/ha.cf
```

- heartbeat 资源信息定义文件。

```
[root@dbm157 ~]# cp /usr/share/doc/heartbeat-2.1.3/haresources /etc/ha.d/
    haresources
```

- heartbeat 心跳检测使用的认证文件，需要将此文件权限设为 600。

```
[root@dbm157 ~]# cp /usr/share/doc/heartbeat-2.1.3/authkeys /etc/ha.d/authkeys
[root@dbm157 ~]#chmod 600 /etc/ha.d/authkeys
```

根据当前的情况，修改 ha.cf 文件。示例如下：

```
[root@dbm157 ~]# cat /etc/ha.d/ha.cf
```

```
logfile /var/log/ha-log      #指定 heartbeat 日志文件的位置
keepalive 1                  #心跳发送时间间隔
deadtime 15                  #备用节点 15s 内没有检测到主机心跳，确认对方故障
warntime 5                   #警告 5 次
initdead 30                  #守护进程启动 30s 后，启动服务资源

#另一台主机节点 eth1 和 eth0 的 IP 地址，通过两个不同网络来保证心跳的可用性，也可以加上串口的检测或
#直接在机器之间连线
ucast eth1 192.168.1.158
ucast eth0 192.168.0.158

#当 primary 节点切换到 sencondary 节点之后，primary 节点恢复正常，不进行切回操作，因为切换一次
# MySQL master 的成本很高
auto_failback off

#定义两个节点的主机名，一行写一个
node dbm157
node dbm158
respawn hacluster /usr/lib64/heartbeat/ipfail

# 开启 dopd 功能
respawn hacluster /usr/lib64/heartbeat/dopd
apiauth ipfail gid=haclient uid=hacluster
apiauth dopd gid=haclient uid=hacluster
```

authkeys 配置文件的示例如下：

```
[root@dbm157 ~]# cat /etc/ha.d/authkeys
auth 1
1 sha1 HA_JACKBILLOW    #使用 sha1 验证，密码为：HA_JACKBILLOW
```

资源说明文件的示例如下：

```
[root@dbm157 ~]# cat /etc/ha.d/haresources
dbm157  drbddisk::r0  Filesystem::/dev/drbd0::/database  mysqld
    IPaddr::192.168.0.222/24/eth0
```

其中：

- 192.168.0.222 是 VIP，在两台主机之间漂移。
- drbddisk 是一个管理 DRBD 的脚本，heartbeat 默认提供这个脚本文件，可以在 /etc/ha.d/resource.d 目录下找到。r0 是一个启动资源，在 drbd 配置文件中定义，通过“drbddisk::r0”可以切换 drbd 主机为 primary 节点或 secondary 节点，只有状态为 Primary 的主机才能挂载 DRBD 分区。drbddisk 脚本相当于执行“drbdadm primary r0”或“drbdadm secondary r0”操作，表示把 DRBD 资源的角色进行变更。
- Filesystem::/dev/drbd0::/database 表示把 /dev/drbd0 设备挂载到 /database 分区下。

在 dbm157 主机设置完成后，把 ha.cf、authkey、haresources 复制一份到另一台主机 dbm158 上。注意修改 ha.cf 中两个 ucast 后面的 IP 地址（对方的 IP 地址）。

启动 heartbeat 服务：

```
[root@dbm157 ~]# /etc/init.d/heartbeat start
Starting High-Availability services:  [  OK  ]
[root@dbm158 ~]# /etc/init.d/heartbeat start
Starting High-Availability services:  [  OK  ]
```

配置 heartbeat 在启动级别 3 中自动启动：

```
[root@dbm157 ~]# chkconfig --level 3 heartbeat on
[root@dbm158 ~]# chkconfig --level 3 heartbeat on
```

观察 heartbeat 启动日志是否正常：

```
[root@dbm157 ~]# tail -f /var/log/ha-log
heartbeat[30098]: 2010/08/12_17:58:28 WARN: Logging daemon is disabled --enabling
    logging daemon is recommended
heartbeat[30098]: 2010/08/12_17:58:28 info: **************************
heartbeat[30098]: 2010/08/12_17:58:28 info: Configuration validated. Starting
    heartbeat 2.1.3
heartbeat[30099]: 2010/08/12_17:58:28 info: heartbeat: version 2.1.3
heartbeat[30099]: 2010/08/12_17:58:28 info: Heartbeat generation: 1281332434
heartbeat[30099]: 2010/08/12_17:58:28 info: glib: ucast: write socket priority
    set to IPTOS_LOWDELAY on eth1
heartbeat[30099]: 2010/08/12_17:58:28 info: glib: ucast: bound send socket to
    device: eth1
heartbeat[30099]: 2010/08/12_17:58:28 info: glib: ucast: bound receive socket to
    device: eth1
heartbeat[30099]: 2010/08/12_17:58:28 info: glib: ucast: started on port 694
    interface eth1 to 192.168.1.157
heartbeat[30099]: 2010/08/12_17:58:28 info: glib: ucast: write socket priority
    set to IPTOS_LOWDELAY on eth0
heartbeat[30099]: 2010/08/12_17:58:28 info: glib: ucast: bound send socket to
    device: eth0
heartbeat[30099]: 2010/08/12_17:58:28 info: glib: ucast: bound receive socket to
    device: eth0
heartbeat[30099]: 2010/08/12_17:58:28 info: glib: ucast: started on port 694
    interface eth0 to 192.168.0.157
heartbeat[30099]: 2010/08/12_17:58:28 info: G_main_add_TriggerHandler: Added
    signal manual handler
heartbeat[30099]: 2010/08/12_17:58:28 info: G_main_add_TriggerHandler: Added
    signal manual handler
heartbeat[30099]: 2010/08/12_17:58:28 info: G_main_add_SignalHandler: Added signal
    handler for signal 17
heartbeat[30099]: 2010/08/12_17:58:28 info: Local status now set to: 'up'
heartbeat[30099]: 2010/08/12_17:58:29 info: Link dbm157:eth1 up.
heartbeat[30099]: 2010/08/12_17:58:29 info: Status update for node dbm157: status
    active
heartbeat[30099]: 2010/08/12_17:58:29 info: Link dbm157:eth0 up.
harc[30108]:    2010/08/12_17:58:29 info: Running /etc/ha.d/rc.d/status status
heartbeat[30099]: 2010/08/12_17:58:30 info: Comm_now_up(): updating status to active
```

```
heartbeat[30099]: 2010/08/12_17:58:30 info: Local status now set to: 'active'
heartbeat[30099]: 2010/08/12_17:58:30 info: Starting child client "/usr/lib64/
    heartbeat/ipfail" (498,496)
heartbeat[30099]: 2010/08/12_17:58:30 info: Starting child client "/usr/lib64/
    heartbeat/dopd" (498,496)
heartbeat[30124]: 2010/08/12_17:58:30 info: Starting "/usr/lib64/heartbeat/ipfail"
    as uid 498  gid 496 (pid 30124)
heartbeat[30125]: 2010/08/12_17:58:30 info: Starting "/usr/lib64/heartbeat/dopd"
    as uid 498  gid 496 (pid 30125)
heartbeat[30099]: 2010/08/12_17:58:30 info: remote resource transition completed.
heartbeat[30099]: 2010/08/12_17:58:30 info: remote resource transition completed.
heartbeat[30099]: 2010/08/12_17:58:30 info: Local Resource acquisition completed. (none)
heartbeat[30099]: 2010/08/12_17:58:30 info: Initial resource acquisition complete
    (T_RESOURCES(them))
ipfail[30124]: 2010/08/12_17:58:35 info: Ping node count is balanced.
```

14.4 搭建 Slave 集群

14.4.1 为什么要搭建 Slave 集群

heartbeat+DRBD 解决了 MySQL 的 Master 的高可用性问题，在 Master 出现故障时能达到快速切换，Slave 也可以使用这种方案来实现高可用，但是这样部署的成本会比较高。在读操作多而写操作比较少的互联网应用中，一台机器很难承受不断增长的读操作，此时，就需要进行读写分离，一台 Master 承担写操作，多台 Slave 提供读操作。如果仍采用 heartbeat+DRBD 来实现 Slave 的高可用性，搭建成本将无法承受。在大多数公司中，都是一组 5×8 的工作人员维护着一个 365×7×24 的在线业务，要求一台 Slave 出现故障不会影响业务的正常运行，同时可以不及时处理故障。这就要求 Slave 也具有比较高的可用性，也就是说，当某一台 Slave 出现故障时，可以自动关闭对外提供服务，这个过程不需要人为干预。

LVS+Keepalived 能很好地实现上述功能，它可以完成负载均衡功能，同时通过自定义检测脚本，保证在一台 Slave 出现故障时，自动从 LVS 的 realserver 列表中剔除故障节点，不再对外提供服务，保证了 Slave 的高可用性，同时成本也很低。

当前很多公司使用 DNS 解析的轮询去实现多台 Slave 的负载均衡，这种方式存在两个问题：第一，不能实现很好的负载均衡；第二，需要第三方监控来更新 DNS 以实现 Slave 的故障切换，因为更新解析的 TTL 问题不能达到很高的可用性。

下面详细介绍 LVS+keepalived 的配置过程。

14.4.2 利用 LVS+Keepalived 搭建高可用 MySQL Slave 集群

在两台机器上安装 LVS+Keepalived，安装过程这里不再介绍，前面已经有过详细介绍。这里给出 keepalived.conf 的示例文件：

```
[root@router01 ~]#cat /etc/keepalived/keepalived.conf
# db slave
virtual_server 192.168.0.223 3306 {
     delay_loop 30
     lb_algo rr
     lb_kind DR
     persistence_timeout 120
     protocol TCP

     real_server 192.168.0.159 3306 {
          MISC_CHECK {
               misc_path "/etc/keepalived/check_slave.pl 192.168.0.159"
               misc_dynamic
   }
}

real_server 192.168.0.160 3306 {
   MISC_CHECK {
        misc_path "/etc/keepalived/check_slave.pl 192.168.0.160"
        misc_dynamic
   }
}

real_server 192.168.0.161 3306 {
   MISC_CHECK {
        misc_path "/etc/keepalived/check_slave.pl 192.168.0.161"
        misc_dynamic
   }
}

}
```

check_slave.pl 是用 Perl 写的一个检测脚本，定时在 Slave 机器上执行 show slave status\G 命令，检查 Slave_IO_Running、Slave_SQL_Running、Seconds_Behind_Master 三个值。Slave_IO_Running 和 Slave_SQL_Running 有一个值为 No 就自动从 LVS 的 realserver 列表中去掉，不再对外提供服务；如果这两个值均为 Yes，检查 Seconds_Behind_Master 大于设定的值也会自动从对外服务机器列表中去掉。三个值同时满足时又会被加入到服务列表中，对外提供服务。

check_slave.pl 的内容如下：

```
[root@router01 ~]# cat /etc/keepalived/check_slave.pl
#!/usr/bin/perl -w
use DBI;
use DBD::mysql;

# CONFIG VARIABLES
$SBM = 120;
$db = "test";
```

```
$host = $ARGV[0];
$port = 3306;
$user = "mon";
$pw = "mon_password";

# SQL query
$query = "show slave status";

$dbh = DBI->connect("DBI:mysql:$db:$host:$port", $user, $pw, { RaiseError =>
    0,PrintError => 0 });

if (!defined($dbh)) {
     exit 1;
}

$sqlQuery = $dbh->prepare($query);
$sqlQuery->execute;

$Slave_IO_Running =  "";
$Slave_SQL_Running = "";
$Seconds_Behind_Master = "";

while (my $ref = $sqlQuery->fetchrow_hashref()) {
      $Slave_IO_Running = $ref->{'Slave_IO_Running'};
      $Slave_SQL_Running = $ref->{'Slave_SQL_Running'};
      $Seconds_Behind_Master = $ref->{'Seconds_Behind_Master'};
}

$sqlQuery->finish;
$dbh->disconnect();

if ( $Slave_IO_Running eq "No" || $Slave_SQL_Running eq "No" ) {
      exit 1;
} else {
      if ( $Seconds_Behind_Master > $SBM ) {
         exit 1;
} else {
         exit 0;
}
}
```

部署完成后在LVS机器上执行如下命令：

```
[root@lvs1 ~]# ipvsadm  -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.0.223:3306 rr persistent 120
  -> 192.168.0.159:3306           Route   1      0          0
  -> 192.168.0.160:3306           Route   1      0          0
```

```
  -> 192.168.0.161:3306           Route   1      0          0
```

使用 LVS 的 DR 模式，在 3 台 Slave 上执行如下命令，把 VIP 绑定到“lo:1”上。

```
[root@dbs159 ~]#vi /etc/sysctl.conf
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_announce = 2
net.ipv4.conf.lo.arp_ignore = 1
net.ipv4.conf.lo.arp_announce = 2
[root@dbs159 ~]#sysctl -p
[root@dbs159 ~]#ifconfig lo:1 192.168.0.223 netmask 255.255.255.255 broadcast
    192.168.0.223
[root@dbs159 ~]# route add -host 192.168.0.223 dev lo:1
```

至此，Slave 集群的搭建完成，业务可以通过访问 VIP（192.168.0.223）进行查询操作。

14.4.3　高可用 Slave 集群的一些注意点

- LVS 有多种负载均衡算法，采用不同的算法使后端的 Slave 主机达到更好的负载均衡效果。
- 当 Slave 增加很多时，超过 10 台以上建议通过垂直拆分来解决压力问题，因为检测脚本是自定义的，性能问题会导致 LVS 机器负载过高。
- 目前 DR 模式在一个 IDC 中部署，不能实现多 IDC 容灾问题，IDC 容灾问题需要另行考虑。

14.5　部署 MySQL 集群要考虑的问题

MySQL 集群满足了企业级的需求，但也并非完美的解决方案。每一个方案都有优点也有缺点，此方案也不例外，在部署的过程中需要考虑以下几个问题。

- heartbeat+DRBD+MySQL 这个方案本身不能达到毫秒级的切换速度，它的切换速度主要受两个因素影响：文件系统和表的恢复需要的时间。这里需要说明的是，MyISAM 引擎不适合 HA，因为 MyISAM 类型的表在宕机后需要很长的修复时间，这违背了 HA 的初衷，所以把除系统表之外的所有表类型都修改为 innodb 引擎。
- 如果对可靠性要求比较高，写入的并发量非常大，建议在 my.cnf 中修改“innodb_flush_log_at_trx_commit = 1”，以保证事务的安全性。但这会对 I/O 提出挑战，如何抉择，最终需要根据具体情况作出权衡。
- 如果写入量不大，可以考虑在 my.cnf 中加入“sync_binlog = 0”来避免在某种特殊情况下 Master 突然宕机，出现 Slave 上关于 Master 的 binlog 位置（master_log_pos）点超于 Master 宕机时写入的 binlog 点的情况。这对 I/O 很有挑战性。
- 在 Slave 上变更主库连接信息时最好指定“master_connect_retry”的值为一个合适的

数值。在出现 VIP 漂移时，Slave 可以更快地去重新连接 VIP，减少因为切换造成的同步延时问题。

- 要为 my.cnf 中的“innodb_log_file_size”和“innodb_log_buffer_size”参数设置合适的值，设置太大会导致恢复时间比较长，降低故障切换的速度。
- 建议在 ha.cf 中使用“auto_failback off”选项，如果使用“on”选项会导致在主、备机之间来回切换，增加成本。在必须执行主、备切换的情况下，无故障时执行即可。
- 使用 dopd 保证在数据不一致时不进行切换，需要人工干预，同时要对 drbd 的同步进行监控，不同步时报警通知 DBA。
- 如果 OS 是 64 位的，建议使用 /usr/lib64/heartbeat 目录下的 ipfail 和 dopd。
- 在使用非 3306 端口或多个端口运行一台物理机时，修改 /etc/init.d/mysqld 脚本使其支持 status、start 和 stop 参数，这在之前已经提到过。虽然可以在一台主机上部署多个 DRBD 分区，但建议一台主机部署一个 DRBD 分区、一个端口。这样做对资源有一定的浪费，但管理成本很低。不建议在一台主机上部署多个 DRBD 分区和搭建多个 MySQL，例如 /dev/drbd0 (parimary/secondary)、/dev/drbd1(secondary/primary)，这种模式减少了机器成本，但增加了管理成本及恢复的复杂度，同时违背了这个方案的初衷。
- HA 也有自己的适用场合，不能利用它解决所有问题（因为 HA 并不能监控 MySQL 的服务状态，当 MySQL 主节点的连接端口出现宕机时，HA 默认监控不到），这时需要考虑通过 heartbeat 的 crm 模式来实现对 MySQL 端口或服务的监控。
- MySQL 部署完成后，定期检测系统是否运行正常是很有必要的。小概率事件是导致大故障的根源。

14.6 本章小结

本章主要讲述了 MySQL 高可用集群在企业中的搭建和应用，MySQL 通过复制功能实现多台 MySQL 的数据同步，heartbeat 实现了 MySQL 写操作的高可用，DRBD 实现了 MySQL 主节点数据块的实时同步，而 LVS 实现了 MySQL 读操作的负载均衡。

目前，MySQL+heartbeat+DRBD+LVS 组合已经是一套成熟的集群解决方案。首先，通过 heartbeat+DRBD 完成了 MySQL 的主节点写操作的高可用性，然后，通过 MySQL+LVS 组合实现了 MySQL 数据库的主从复制，同时实现了和 MySQL 读操作的负载均衡。从应用方面来看，这个方案实现了读写的分离，并且融合了写操作的高可用和读操作的负载均衡，因此，作为企业应用平台，这样完美的方案绝对是首选！