Squid 全套使用手册

整理者：哈密瓜

blog:http://hi.baidu.com/gyl4802959

目录：

第 1 章 Squid 的简介性描述

1.1 Web 缓存
1.2 Squid 的简明历史
1.3 硬件和操作系统要求
1.4 squid 是开源的
1.5 Squid 的 Web 主页
1.6 获取帮助
1.7 启动 Squid

第 2 章 获取 Squid

2.1 版本和发布
2.2 使用源代
2.3 预编译的二进制文件
2.4 匿名 CVS
2.5 devel.squid-cache.org

第 3 章 编译和安装

3.1 安装之前
3.2 解开源代码包
3.3 调整内核
3.4 Configure 脚本
3.5 编译
3.6 安装
3.7 打补丁

3.8 重运行 **configure**

第 **4** 章 快速配置向导

4.1 squid.conf 语法
4.2 User IDs
4.3 端口号
4.4 日志文件路径
4.5 访问控制
4.6 可见主机名
4.7 管理联系信息
4.8 下一步

第 **5** 章 运行 **Squid**

5.1 squid 命令行选项
5.2 对配置文件查错
5.3 初始化 **cache** 目录
5.4 在终端窗口里测试 **squid**
5.5 将 **squid** 作为服务进程运行
5.6 启动脚本
5.7 chroot 环境
5.8 停止 **squid**
5.9 重配置运行中的 **squid** 进程
5.10 滚动日志文件

第 **6** 章 访问控制

6.1 访问控制元素
6.2 访问控制规则
6.3 常见用法
6.4 测试访问控制

第 **7** 章 磁盘缓存基础

7.1 cache_dir 指令
7.2 磁盘空间基准
7.3 对象大小限制
7.4 分配对象到缓存目录
7.5 置换策略
7.6 删除缓存对象
7.7 refresh_pattern

第 **8** 章 高级磁盘缓存主题

8.1 是否存在磁盘 **I/O** 瓶颈?
8.2 文件系统调整选项
8.3 可选择的文件系统
8.4 aufs 存储机制

8.5 diskd 存储机制
8.6 coss 存储机制
8.7 null 存储机制
8.8 哪种最适合我？

第 **9** 章 **Cache** 拦截

9.1 它如何工作？
9.2 为何要（或不要）拦截？
9.3 网络设备
9.4 操作系统配置
9.5 配置 **Squid**
9.6 调试问题

第 1 章、Squid 的简介性描述

1.1 Web 缓存

这节里需要明白 3 个概念：

cache 命中 - 在 squid 每次从它的缓存里满足 HTTP 请求时发生。cache 命中率，是所有 HTTP 请求中命中的比例。Web 缓存典型的 cache 命中率在 30%到 60%之间。另一个相似的度量单位叫做字节命中率，描绘了 cache 提供服务的数据容量（字节数）。

cache 丢失 - 在 squid 不能从它的缓存里满足 HTTP 请求时发生。cache 丢失的理由有很多种。最明显的，当 squid 第一次接受到对特殊资源的请求时，就是一个 cache 丢失。类似的情况是，squid 会清除缓存以释放空间给新对象。另外的可能是资源不可到达。原始服务器会指示 cache 怎样处理响应。例如，它会提示数据不能被缓存，或在有限的时间内才被重复使用，等等。

cache 确认 - 保证 squid 不对用户返回过时数据。在重复使用缓存对象时，squid 经常从原始服务器确认它。假如服务器指示 squid 的拷贝仍然有效，数据就发送出去。否则，squid 升级它的缓存拷贝，并且转发给客户。

1.2 Squid 的简明历史

对本节感兴趣的读者请阅读英文原文档。

1.3 硬件和操作系统要求

Squid 运行在所有流行的 Unix 系统上，也可以在 Microsoft Windows 上运行。尽管 squid 的 Windows 支持在不断改进，但也许在 Unix 上容易一些。假如你有一个喜欢的操作系统，我建议你使用那个。否则，假如你找人推荐，我很喜欢 FreeBSD。

squid 对硬件要求不算高。内存是最重要的资源。内存短缺会严重影响性能。磁盘空间也是另一个重要因素。更多的磁盘空间意味着更多的缓存目标和更高的命中率。快速的磁盘和驱动器也是有利的。如果你舍得花钱，SCSI 磁盘比 ATA 的执行性能好。当然快速的 CPU 也是好的，但它并不是提高性能的关键因素。

因为 squid 对每个缓存响应使用少数内存，因此在磁盘空间和内存要求之间有一定联系。基本规则是，每 G 磁盘空间需要 32M 内存。这样，512M 内存的系统，能支持 16G 的磁盘缓存。你的情况当然会不同。内存需求依赖于如下事实：缓存目标大小，CPU 体系（32 位或 64 位），同时在线的用户数量，和你使用的特殊功能。

人们经常问如此问题："我的网络有 X 个用户，需要配备什么样的硬件给 squid？"因为许多理由，这样的问题好难回答。特别的，很难说 X 个用户将产生多少流量。我告诉人们去建立一个有足够磁盘空间，可存储 3-7 天 web 流量数据的系统。例如，假如你的用户每天 8 小时耗费 1M 流量（仅仅 HTTP 和 FTP 传输），那就是每天大约 3.5G。所以，我可以说，每兆 web 传

输你需要 10 到 25G 的磁盘空间。

1.4 squid 是开源的

Squid 是自由软件和合作项目。假如你觉得 squid 有用，请考虑以下面一种或几种方法来回报该项目：
1.参与 squid 用户讨论列表，回答问题和帮助新用户。
2.测试新版本，报告 bug 或其他问题。
3.致力于在线文档和 FAQ。假如你发现错误，将它报告给维护者。
4.将你的局部修改提交给开发者。
5.对开发者提供财政支持。
6.告诉开发者你想要的新功能。
7.告诉你的朋友和同学，Squid 非常 Cool。

Squid 是在 GNU 公用许可证（GPL）下发行的自由软件。关于 GPL 的更多信息请见：http://www.gnu.org/licenses/gpl-faq.html

1.5 Squid 的 Web 主页

Squid 的主页在 http://www.squid-cache.org 你自己阅读该站点吧。

1.6 获取帮助

1.6.1 FAQ

Squid 的 FAQ 文档在 http://www.squid-cache.org/Doc/FAQ/FAQ.html ,是对新用户的好信息资源。

1.6.2 邮件列表

Squid 有三个邮件列表可用。邮件列表主页在：http://www.squid-cache.org/mailing-lists.html

1.6.2.1 Squid 用户

订阅该邮件列表，发邮件到 squid-users-subscribe@squid-cache.org

1.6.2.2 Squid 公告

订阅该邮件列表，发邮件到 squid-announce-subscribe@squid-cache.org

1.6.2.3 Squid 开发

加入该邮件列表有所限制。它的内容发布在 http://www.squid-cache.org/mail-

archive/squid-dev/

1.6.3 职业支持

即付费的支持 - 职业支持服务提供商列表，请见 http://www.squid-cache.org/Support/services.html

1.7 启动 Squid

请按下面的章节一步一步来吧。

第 2 章 获取 Squid

2.1 版本和发布

Squid 开发者定期发布源代码。每一个发布版有一个版本号，例如 2.5.STABLE4。版本号的第三部分以 STABLE 或 DEVEL（短期开发版本）开头。

也许你能猜到，DEVEL 版本倾向于拥有更新，更试验性的功能。但也许它们有更多的 bugs。无经验的用户不应该运行 DEVEL 版本。假如你选择运行一个 DEVEL 版本，并且遇到了问题，请将问题报告给 Squid 维护者。

在一段时间的开发期后，Squid 版本号变为 STABLE。该版本适合于普通用户。当然，即使稳定版可能也有一些 bugs。高的稳定版本（例如 STABLE3,STABLE4）应该 bugs 更少。假如你特别关心稳定性，你应该使用这些最近发布版本中的一个。

2.2 使用源代

为什么你不能 copy 一份预编译的二进制代码到你的系统中，并且期望它运行良好呢？主要理由是 squid 的代码需要知道特定操作系统的参数。实际上，最重要的参数是打开文件描述符的最大数量。Squid 的./configure 脚本在编译之前侦察这些值。假如你获取一个已编译的使用某个参数值的 squid 到另一个使用不同参数值的系统中，可能会遇到问题。

另一个理由是许多 squid 功能在编译时必须被激活。假如你获取一个别人已编译的 squid 文件，它不包含你所需要的功能，那么你又得再编译一遍。最后，共享库的问题可能使得在系统之间共享可执行文件困难。共享库在运行时被装载，如已知的动态链接一样。squid 在编译时会侦察你系统中的 C 库的某些功能（例如它们是否被提供，是否能运行等）。尽管库功能不常改变，但两个不同的系统的 C 库之间可能有明显的区别。如果两个系统差别太大，就会对 Squid 造成问题。

获取 squid 的源代码是非常容易的。请访问 squid 的首页：http://www.squid-cache.org。首页有链接指向不同的稳定版和开发版。假如你不在美国，那么请访问 squid 的众多镜像站点中的一个。镜像站点通常以"wwwN.CC.squid-cache.org"命名，N 是数字，CC 是国家的两位代码。例如，www1.au.squid-cache.org 是澳大利亚的镜像站点，在主页上有链接指向不同的镜像站点。

每一个 squid 发布版分支（例如 Squid-2.5）有它自己的 HTML 页面。该页面有链接指向源代码，以及与其他发布版的差别。假如你从一个发布版升级到下一个，你应该下载这些差别文件，并且打上补丁，请见 3.7 章节中的描述。每个版本的发布页描述新功能和重要的改进，也有链接指向已经修正的 bugs。

如果 web 访问不可行，你能从 ftp://ftp.squid-cache.org 的 FTP 服务器获取源代码，或者使用其他 FTP 镜像。要获取当前版本，请访问 pub/squid-2/DEVEL 或 pub/squid-2/STABLE 目录。FTP 镜像也在许多国家有，你能用同样的国家代码去猜测一些 FTP 镜像站点，例如 ftp1.uk.squid-cache.org。

当前的 Squid 发布版本大约 1M 大小。在下载完压缩的打包文件后，你能继续第 3 章。

2.3 预编译的二进制文件

一些 Unix 发布版可能预包含了 Squid 的编译版。对 Linux 系统，你可以找到 Squid 的 RPM 包。通常 squid RPM 包含在你所买的 Linux 光碟里。Freebsd/Netbsd/OpenBSD 也在它们的 ports 或者 packages 里面包含了 squid。

虽然 RPM 或者预编译的 packages 能节省你一些时间，但它们也有一些弊端。就像我提过的一样，在你开始编译 squid 之前，某些功能必须被激活或禁止。而你安装的预编译的包可能不包含你想要的特定功能。而且，squid 的./configure 脚本侦察你系统中的特定参数，这些在你系统中的参数可能与编译它的机器的参数不同。

最后，假如你想对 squid 打补丁，你必须等某个人编译更新的 RPM 或 packages，或者你还得自己找源代码编译。

我强烈建议你从源代码编译 squid，当然怎样选择由得你。

2.4 匿名 CVS

你能匿名访问 squid 的 CVS 文件（只读）以保持你的源代码同步更新。使用 CVS 的有利面是你能轻易获取当前运行版本的补丁。这样就容易发现近来改变了什么。

将这些补丁打到你所运行的版本中，有效的保持你的源代码和官方版本的同步。

CVS 使用树型索引系统，树干叫做头分支。对 Squid 而言，这里也是所有的新改变和新功能的存放之地。头分支通常包含试验性的，也许不太稳定的代码。稳定的代码通常在其他分支上。

为了有效的使用 squid 的匿名 CVS，你首先应知道版本和分支是怎样被标明不同的。例如， 版本 2.5 分支被命名为 SQUID_2_5 。具体的发布有长的命名， 例如 SQUID_2_5_STABLE4。为了得到 squid 版本 2.5.STABLE4，请使用 SQUID_2_5_STABLE4 标签；使用 SQUID_2_5 得到最近的 2.5 分支的代码。

为了使用 squid 匿名 CVS 服务，你首先必须设置 CVSROOT 环境变量：
csh% setenv CVSROOT:pserver:anoncvs@cvs.squid-cache.org:/squid，
或者，对 Bourne shell 用户：
sh$ CVSROOT=:pserver:anoncvs@cvs.squid-cache.org:/squid
sh$ export CVSROOT

然后你就可以登陆到服务器：

% cvs login
(Logging in to anoncvs@cvs.squid-cache.org)
CVS password:

在提示符下，敲入 anoncvs 作为密码。现在你可以用这个命令检查源代码树：

% cvs checkout -r SQUID_2_5 -d squid-2.5 squid

-r 选项指定获取修订标签。省略-r 选项你将获得头分支。

-d 选项改变存放文件的顶级目录名。假如你省略-d 选项，顶级目录名就与模块名字一样。

最后的命令行参数（squid）是要检查的模块名字。

一旦你检查完 squid 源代码树，你能运行 cvs update 命令去升级你的文件，和保持文件同步。其他命令包括：cvs diff, cvs log, 和 cvs annotate。

想获取更多 CVS 知识，请访问：http://www.cvshome.org

2.5 devel.squid-cache.org

Squid 的开发者维持一个独立的站点，当前运行在 SourceForge，提供了试验性的 squid 功能。请检查它们在 http://devel.squid-cache.org.在这里你能发现许多正在开发的工程，它们还未集成到 squid 的官方源代码里。你能通过 SourceForge 的匿名 CVS 服务来访问这些工程，或者下载与标准版本不同的差别文件。

第 3 章 编译和安装

3.1 安装之前

假如你使用 unix 有一段时间，并且已编译过许多其他软件包，那么只需快速的扫描本章。编译安装 squid 的过程与安装其他软件相似。

为了编译 squid，你需要一个 ANSI C 编译器。不要被 ANSI 字眼吓倒。假如你已经有一个编

译器，它顺从 ANSI 指令，那么也一样。GNU C 编译器（gcc）是很好的选择，它被广泛使用。大部分操作系统在其标准安装中附带了 C 编译器，不过 Solaris 和 HP-UX 除外。假如你使用这样的操作系统，那可能没有安装编译器。

理论上你应该在即将运行 squid 的机器上编译 squid。安装过程侦察你的操作系统以发现特定的参数，例如可用文件描述符的数量。然而，假如你的系统没有 C 编译器存在，你也许会在其他机器上编译 squid，然后把二进制代码 copy 回来。如果操作系统不同，那么 squid 可能会遇到问题。假如操作系统有不同的内核配置，squid 会变得混乱。

除了 C 编译器，你还需要 perl 和 awk。awk 是所有 unix 系统的标准程序，所以你不必担心它。perl 也是相当普及的，但它也许没有默认安装在你的系统上。你需要 gzip 程序来解压源代码发布文件。

对 Solaris 用户，请确认/usr/ccs/bin 包含在你的 PATH 环境变量里，即使你使用 gcc 编译器。为了编译 squid，make 和 ar 程序需要在这个目录找到。

3.2 解开源代码包

在下载完源代码后，你需要在某个目录解开它。具体哪个目录无关紧要。你能解开 squid 在你的家目录或任何其他地方，大概需要 20M 的自由磁盘空间。我个人喜欢用/tmp。使用 tar 命令来展开源代码目录：

```
% cd /tmp
% tar xzvf /some/where/squid-2.5.STABLE4-src.tar.gz
```

一些 tar 程序不支持 z 选项，该选项自动解压 gzip 文件。如果这样，你需要运行如下命令：

```
% gzip -dc /some/where/squid-2.5.STABLE4-src.tar.gz | tar xvf -
```

一旦源代码被展开，下一步通常是配置源代码树。然而，假如这是你第一次编译 squid，你应确认特定的内核资源限制足够高。怎样发现，请继续。

3.3 调整内核

Squid 在高负载下，需要大量的内核资源。特别的，你需要给你的系统配置比正常情况更高的文件描述符和缓存。文件描述符的限制通常很恼人。你最好在开始编译 squid 之前来增加这些限制的大小。

因为这点，你可能为了避免重建内核的麻烦，而倾向于使用预编译的二进制版本。不幸的是，不管如何你必须重建一个新内核。squid 和内核通过数据结构来交换信息，数据结构的大小不能超过设置的文件描述符的限制。squid 在运行时检查这些设置，并且使用最安全的（最小的）值。这样，即使预编译的二进制版本有比你的内核更高的文件描述符，但还是以你系统内核的实际数值为主。

为了改编一些参数，你需要重建新内核。这个过程在不同的操作系统之间不同。假如需要，请参阅 Unix 系统管理员手册（Prentice Hall 出版）或者你的操作系统文档。假如你正使用 Linux，

可能不必重建内核。

3.3.1 文件描述符

文件描述符是一个简单的整数，用以标明每一个被进程所打开的文件和 socket。第一个打开的文件是 0，第二个是 1，依此类推。Unix 操作系统通常给每个进程能打开的文件数量强加一个限制。更甚的是，unix 通常有一个系统级的限制。

因为 squid 的工作方式，文件描述符的限制可能会极大的影响性能。当 squid 用完所有的文件描述符后，它不能接收用户新的连接。也就是说，用完文件描述符导致拒绝服务。直到一部分当前请求完成，相应的文件和 socket 被关闭，squid 不能接收新请求。当 squid 发现文件描述符短缺时，它会发布警告。

在运行./configure 之前，检查你的系统的文件描述符限制是否合适，能给你避免一些麻烦。大多数情况下，1024 个文件描述符足够了。非常忙的 cache 可能需要 4096 或更多。在配置文件描述符限制时，我推荐设置系统级限制的数量为每个进程限制的 2 倍。

通常在你的 Unix shell 中能找到系统的文件描述符限制。所有的 C shell 及其类似的 shell 有内建的 limit 命令。更新的 Bourne shell 及其类似的 shell 有一条叫做 ulimit 的命令。为了发现你的系统的文件描述符限制，试运行如下命令：

```
csh% limit descriptors unlimited
csh% limit descriptors
descriptors 4096
```

或者

```
sh$ ulimit -n unlimited
sh$ ulimit -n
4096
```

在 Freebsd 上，你能使用 sysctl 命令：

```
% sysctl -a | grep maxfiles
kern.maxfiles: 8192
kern.maxfilesperproc: 4096
```

如果你不能确认文件描述符限制，squid 的./configure 脚本能替你做到。当你运行./configure 时，请见 3.4 章节，观察末尾这样的输出：

```
checking Maximum number of file descriptors we can open... 4096
```

假如其他的 limit,ulimit,或者./configure 报告这个值少于 1024，你不得不在编译 squid 之前，花费时间来增加这个限制值的大小。否则，squid 在高负载时执行性能将很低。

增加文件描述符限制的方法因系统不同而不同。下面的章节提供一些方法帮助你开始。

3.3.1.1 Freebsd,NetBSD,OpenBSD

编辑你的内核配置文件，增加如下一行：
options MAXFILES=8192

在 OpenBSD 上，使用 option 代替 options。然后，configure，编译，和安装新内核。最后重启系统以使内核生效。

3.3.1.2 Linux

在 Linux 上配置文件描述符有点复杂。在编译 squid 之前，你必须编辑系统 include 文件中的一个，然后执行一些 shell 命令。请首先编辑/usr/include/bits/types.h 文件，改变 __FD_SETSIZE 的值：
#define _ _FD_SETSIZE 8192

下一步，使用这个命令增加内核文件描述符的限制：
echo 8192 >; /proc/sys/fs/file-max

最后，增加进程文件描述符的限制，在你即将编译 squid 的同一个 shell 里执行：
sh# ulimit -Hn 8192

该命令必须以 root 运行，仅仅运行在 bash shell。不必重启机器。

使用这个技术，你必须在每一次系统启动后执行上述 echo 和 ulimit 命令，或者至少在 squid 启动之前。假如你使用某个 rc.d 脚本来启动 squid，那是一个放置这些命令的好地方。

3.3.1.3 Solaris

增加该行到你的/etc/system 文件：
set rlim_fd_max = 4096

然后，重启机器以使改动生效。

3.3.2 Mbuf Clusters

BSD 基础的网络代码使用一个叫做 mbuf（参阅 W.R.Stevens 的 TCP/IP 描述卷 2）的数据结构。Mbuf 典型的是小块内存（例如 128 字节）。较大的网络包的数据存储在 mbuf clusters 里。内核可能给系统可用的 mbuf clusters 的总数量强加一个最高限制。你能使用 netstat 命令来发现这个限制：
% netstat -m
196/6368/32768 mbufs in use (current/peak/max):
146 mbufs allocated to data
50 mbufs allocated to packet headers
103/6182/8192 mbuf clusters in use (current/peak/max)

13956 Kbytes allocated to network (56% of mb_map in use)
0 requests for memory denied
0 requests for memory delayed
0 calls to protocol drain routines

在这个例子里，有 8192 个 mbuf clusters 可用，但是永远不会同时用到 6182 个。当系统用尽 mbuf clusters 时，I/O 机制例如 read()和 write()返回"无缓存空间可用"的错误信息。

NetBSD 和 OpenBSD 使用 netstat -m 不会显示 mbuf 的输出。代替的，它们在 syslog 里报告："WARNING: mclpool limit reached" 。

为了增加 mbuf clusters 的数量，你必须在内核配置文件里增加一个选项：
options NMBCLUSTERS=16384

3.3.3 临时端口范围

临时端口是 TCP/IP 栈分配给出去连接的本地端口。换句话说，当 squid 发起一条连接到另一台服务器，内核给本地 socket 分配一个端口号。这些本地端口号有特定的范围限制。

例如，在 FreeBSD 上，默认的临时端口范围是 1024-5000。

临时端口号的短缺对非常忙的代理服务器（例如每秒数百个连接）来说，会较大的影响性能。这是因为一些 TCP 连接在它们被关闭时进入 TIME_WAIT 状态。当连接进入 TIME_WATI 状态时，临时端口号不能被重用。

你能使用 netstat 命令来显示有多少个连接进入这个状态：

```
% netstat -n | grep TIME_WAIT
Proto Recv-Q Send-Q Local Address Foreign Address (state)
tcp4 0 0 192.43.244.42.19583 212.67.202.80.80 TIME_WAIT
tcp4 0 0 192.43.244.42.19597 202.158.66.190.80 TIME_WAIT
tcp4 0 0 192.43.244.42.19600 207.99.19.230.80 TIME_WAIT
tcp4 0 0 192.43.244.42.19601 216.131.72.121.80 TIME_WAIT
tcp4 0 0 192.43.244.42.19602 209.61.183.115.80 TIME_WAIT
tcp4 0 0 192.43.244.42.3128 128.109.131.47.25666 TIME_WAIT
tcp4 0 0 192.43.244.42.3128 128.109.131.47.25795 TIME_WAIT
tcp4 0 0 192.43.244.42.3128 128.182.72.190.1488 TIME_WAIT
tcp4 0 0 192.43.244.42.3128 128.182.72.190.2194 TIME_WAIT
```

注意这个例子中既有客户端连接又有服务器端的连接。客户端连接有 3128 作为临时端口号，服务器端连接有 80 作为远程主机的端口号。临时端口号出现在本地地址栏里。在该例子里，它们是 19000 秒。

如果你没有看到数千个临时端口在 TIME_WAIT 状态，那也许不必增加这个端口范围。

在 Freebsd 上，用如下命令增加临时端口范围：
sysctl -w net.inet.ip.portrange.last=30000

在 OpenBSD 上，命令类似，但 sysctl 变量有不同的名字：
sysctl -w net.inet.ip.portlast=49151

在 NetBSD 上，事情稍有不同。默认的值是 49152-65535.为了增加这个范围，需改变最低限制：
sysctl -w net.inet.ip.anonportmin=10000

在 Linux 上，简单的写一对数字到下列指定文件：
echo "1024 40000" >; /proc/sys/net/ipv4/ip_local_port_range

不要忘记将这些命令加到你的系统启动脚本中，以使机器每一次重启后都生效。

3.4 Configure 脚本

象许多其他 Unix 软件一样，squid 在开始编译之前使用./configure 脚本来了解操作系统信息。./configure 脚本由流行的 GNU autoconf 程序产生。当 script 运行时，它用不同的方法来侦察系统，以发现关于库，函数，类型，参数，和有没有功能被提供等。./configure 所做的第一件事情是去找一个 C 编译器。假如 C 编译器没有找到，或者编译一个简单的测试程序失败，./configure 脚本不能继续。

./configure 脚本有大量的选项。最重要的是安装 prefix。在运行./configure 之前，你需要决定 squid 被安装在哪里。prefix 选项指定 squid 日志，二进制文件，和配置文件的默认位置 。你可以在安装之后改变这些文件的位置，但假如你现在决定，事情更容易。

默认的安装位置是/usr/local/squid.squid 将文件放在 prefix 指定目录下面的 7 个子目录：

```
% ls -l /usr/local/squid
total 5
drwxr-x--- 2 wessels wheel 512 Apr 28 20:42 bin
drwxr-x--- 2 wessels wheel 512 Apr 28 20:42 etc
drwxr-x--- 2 wessels wheel 512 Apr 28 20:42 libexec
drwxr-x--- 3 wessels wheel 512 Apr 28 20:43 man
drwxr-x--- 2 wessels wheel 512 Apr 28 20:42 sbin
drwxr-x--- 4 wessels wheel 512 Apr 28 20:42 share
drwxr-x--- 4 wessels wheel 512 Apr 28 20:43 var
```

Squid 使用 bin,etc,libexec,man,sbin,和 share 目录存放一些相对较小的文件（或其他目录），这些文件不经常改变。但 var 目录的文件别有洞天。这里你可以发现 squid 的日志文件，它增长得非常大（数十或数百兆）。var 也是实际磁盘 cache 的默认位置。你也许想将 var 目

录放在磁盘空间足够的位置，这样做较容易的方法是使用--localstatedir 选项：
% ./configure --localstatedir=/bigdisk/var

当配置 squid 时，你不必对这些路径名称担心太多。你以后可以在 squid.conf 文件里改变这些路径名。

3.4.1 configure 选项

./configure 脚本有大量的不同选项，它们以-开始。当你敲入./configure --help 时，能看到选项的完整列表。一些选项对所有 configure 脚本是通用的，还有一些是 squid 专有的。下面是你可能用得到的标准选项：
--perfix =PREFIX

如前面描述的一样，这里设置安装目录。安装目录是所有可执行文件，日志，和配置文件的默认目录。在整本书中，$prefix 指你选择的安装目录。

--localstatedir =DIR
该选项允许你改变 var 目录的安装位置。默认是$prefix/var，但也许你想改变它，以使 squid 的磁盘缓存和日志文件被存储在别的地方。

--sysconfdir =DIR
该选项允许你改变 etc 目录的位置。默认的是$prefix/etc。假如你想使用/usr 作为安装位置，你也许该配置--sysconfdir 为/etc.

以下是 squid 的专有./configure 选项：

--enable-dlmalloc[=LIB]
在一些系统上，内建的内存分配机制（malloc）在使用 squid 时表现不尽人意。使用--enable-dlmalloc 选项将 squid 源代码包中的 dlmalloc 包编译和链接进来。假如你的系统中已安装 dlmalloc，你能使用=LIB 参数指定库的路径。请见 http://g.oswego.edu/dl/html/malloc.html 更多关于 dlmalloc 的信息。

--enable-gnuregex
在访问控制列表和其他配置指令里，squid 使用正则表达式作为匹配机制。GNU 的正则表达式库包含在 squid 的源代码包里；它可以在没有内建正则表达式的操作系统中使用。./configure 脚本侦察你系统中的正则表达式库，假如必要，它可以激活使用 GNU 正则表达式。如果因为某些理由，你想强制使用 GNU 正则表达式，你可以将这个选项加到./configure 命令后。

--enable-carp
Cache 数组路由协议（CARP）用来转发丢失的 cache 到父 cache 的数组或 cluster。在 10.9 章有更多关于 CARP 的细节。

--enable-async-io[=N_THREADS]
同步 I/O 是 squid 技术之一，用以提升存储性能。aufs 模块使用大量的线程来执行磁盘 I/O 操

作。该代码仅仅工作在 linux 和 solaris 系统中。=N_THREADS 参数改变 squid 使用的线程数量。aufs 和同步 I/O 在 8.4 章中被讨论。
请注意--enable-async-io 是打开其他三个./configure 选项的快捷方式，它等同于：
--with-aufs-threads=N_THREADS
--with-pthreads
--enable-storeio=ufs,aufs
--with-pthreads

该选项导致编译过程链接到你系统中的 P 线程库。aufs 存储模块是 squid 中唯一需要使用线程的部分。通常来说，如果你使用--enable-saync-io 选项，那么不必再单独指定该选项，因为它被自动激活了。

--enable-storeio=LIST
Squid 支持大量的不同存储模块。通过使用该选项，你告诉 squid 编译时使用哪个模块。在 squid-2.5 中，支持 ufs,aufs,diskd,和 null 模块。通过查询 src/fs 中的目录，你能得到一个模
块列表。
LIST 是一个以逗号分隔的模块列表，例如：
% ./configure --enable-storeio=afus,diskd,ufs

ufs 模块是默认的，看起来问题最少。不幸的是，它性能有限。其他模块可能在某些操作系统中不必编译。关于 squid 存储模块的完整描述，请见第 8 章。
--with-aufs-threads=N_THREADS

指定 aufs 存储机制使用的线程数量（见 8.4 章）。squid 默认根据缓存目录的数量，自动计算需要使用多少线程。

--enable-heap-replacement
该选项不再使用，但被保留用于向后兼容性。你该使用--enable-removal-policies 来代替。

--enable-removal-policies=LIST
排除策略是 squid 需要腾出空间给新的 cache 目标时，用以排除旧目标的机制。squid-2.5 支持 3 个排除策略：最少近期使用(LRU),贪婪对偶大小(GDS),最少经常使用(LFU)。

然而，因为一些理由，./configure 选项使指定的替代策略和需要执行它们的基本数据结构之间的差别模糊化。LRU 是默认的，它以双链表数据结构执行。GDS 和 LFU 使用堆栈的数据结构。

为了使用 GDS 或 LFU 策略，你指定：
% ./configure --enable-removal-policies=heap

然后你在 squid 的配置文件里选择使用 GDS 或 LFU。假如你想重新使用 LRU,那么指定：
% ./configure --enable-removal-policies=heap,lru
更多的关于替换策略的细节请见 7.5 章。

--enable-icmp
如在 10.5 章中描述的一样，squid 能利用 ICMP 消息来确定回环时间尺寸，非常象 ping 程序。你能使用该选项来激活这些功能。

--enable-delay-pools
延时池是 squid 用于传输形状或带宽限制的技术。该池由大量的客户端 IP 地址组成。当来自这些客户端的请求处于 cache 丢失状态，他们的响应可能被人工延迟。关于延时池的更多细节请见附录 C。

--enable-useragent-log
该选项激活来自客户请求的 HTTP 用户代理头的日志。更多细节请见 13.5 章。

--enable-referer-log
该选项激活来自客户请求的 HTTP referer 日志。更多细节请见 13.4 章。

--disable-wccp
Web cache 协调协议(WCCP)是 CISCO 的专有协议，用于阻止或分发 HTTP 请求到一个或多个 caches。WCCP 默认被激活，假如你愿意，可以使用该选项来禁止该功能。

--enable-snmp
简单网络管理协议(SNMP)是监视网络设备和服务器的流行方法。该选项导致编译过程去编译所有的 SNMP 相关的代码，包括一个裁切版本的 CMU SNMP 库。

--enable-cachemgr -hostname[=hostname]
cachemgr 是一个 CGI 程序，你能使用它来管理查询 squid。默认 cachemgr 的 hostname 值是空的，但你能使用该选项来指定一个默认值。例如：
% ./configure --enable-cachemgr-hostname=mycache.myorg.net

--enable-arp-acl
squid 在一些操作系统中支持 ARP,或者以太地址访问控制列表。该代码使用非标准的函数接口，来执行 ARP 访问控制列表，所以它默认被禁止。假如你在 linux 或 solaris 上使用 squid，你可能用的上这个功能。

--enable-htcp
HTCP 是超文本缓存协议--类似于 ICP 的内部缓存协议。更多细节请见 10.8 章。

--enable-ssl
使用该选项赋予 squid 终止 SSL/TLS 连接的能力。注意这仅仅工作在 web 加速器中用以加速请求。更多细节请见 15.2.2 章节。

--with-openssl[=DIR]
假如必要，你使用该选项来告诉 squid 到哪里找到 OpenSSL 库或头文件。假如它们不在默认位置，在该选项后指定它们的父路径。例如：
% ./configure --enable-ssl --with-ssl=/opt/foo/openssl

在这个例子中，你的编译器将在/opt/foo/openssl/include 目录中找头文件， 在/opt/foo/openssl/lib 中找库文件。

--enable-cache-digests
Cache 消化是 ICP 的另一个替代，但有着截然不同的特性。请见 10.7 章。

--enable-err-languages="lang1 lang2 ..."
squid 支持定制错误消息，错误消息可以用多种语言报告。该选项指定复制到安装目录($prefix/share/errors)的语言。假如你不使用该选项，所有可用语言被安装。想知道何种语言可用，请见源代码包里 errors 目录下的目录列表。如下显示如何激活多种语言：
% ./configure --enable-err-languages="Dutch German French" ...

--enable-default-err-language=lang
该选项设置 error_directory 指令的默认值。例如，假如你想使用荷兰语，你能这样指定：
% ./configure --enable-default-err-language=Dutch
你也能在 squid.conf 里指定 error_directory 指令，在附录 A 中有描述。假如你忽略该选项，英语是默认错误语言。

--with-coss-membuf-size=N
循环目录存储系统（coss）是 squid 的试验性存储机制。该选项设置 coss 缓存目录的内存缓冲大小。注意为了使用 coss，你必须在--enable-storeio 选项里指定存储类型。
该参数以字节形式赋值，默认是 1048576 字节或 1M。你能指定 2M 缓冲如下：
% ./configure --with-coss-membuf-size=2097152

--enable-poll
unix 提供两个相似的函数用以在 I/O 事件里扫描开放文件描述符：select() 和 poll()。./configure 脚本通常能非常好的计算出何时使用 poll()来代替 select().假如你想强制使用 poll()，那么指定该选项。
--desable-poll
类似的，如果不使用 poll()，那么指定该选项。

--disable-http-violations
squid 默认可以被配置成违背 HTTP 协议规范。你能使用该选项来删除违背 HTTP 协议的代码。

--enable-ipf-transparent
在第 9 章中，我将描述如何配置 squid 来拦截缓存。一些操作系统使用 IP Filter 包来协助拦截缓存。在这些环境下你应该使用该./configure 选项。如果你使用了该选项，但是编译器提示 src/client_side.c 文件出错，那是因为 IP Filter 包没有或没有正确的安装在你的系统中。

--enable-pf-transparent
你可能需要指定该选项，使用 PF 包过滤器在操作系统中拦截 HTTP。PF 是 OpenBSD 的标准包过滤器，也可能被发布到其他系统中。假如你使用该选项，但是编译器提示 src/client_side.c 文件出错，那是因为 PF 没有实际安装到你的系统中。

--enable-linux-netfilter
Netfilter 是 linux 2.4 系列内核的包过滤器名字。假如你想在 linux2.4 或以后的版本中使用 HTTP 拦截功能，那么激活该选项。

--disable-ident-lookups
ident 是一个简单的协议，允许服务器利用客户端的特殊 TCP 连接来发现用户名。假如你使用该选项，编译器将把执行这些查询的代码排除出去。即使你在编译时保留了这些代码，除非你在 squid.conf 文件里指定，squid 不会执行 ident 查询。

--disable-internal-dns
squid 源代码包含两个不同的 DNS 解决方案，叫做"内部的"和"外部的"。内部查询是默认的，但某些人可能要使用外部技术。该选项禁止内部功能，转向使用旧的方式。
内部查询使用 squid 自己的 DNS 协议执行工具。也就是说，squid 产生未完成的 DNS 查询并且将它们发送到一个解析器。假如超时，它重新发送请求，你能指定任意数量的解析器。该工具的有利处之一是，squid 获得准确无误的 DNS 响应的 TTLs。
外部查询利用 C 库的 gethostbyname()和 gethostbyaddr()函数。squid 使用一个外部进程池来制造并行查询。使用外部 DNS 解析的主要弊端是你需要更多的辅助进程，增加 squid 的负载。另一个麻烦是 C 库函数不在响应里传输 TTLs，这样 squid 使用 postive_dns_ttl 指令提供的一个常量值。

--enable-truncate
truncate()系统调用是 unlink()的替代品。unlink()完全删除 cache 文件，truncate()将文件大小设为零。这样做释放了分配给该文件的磁盘空间，但留下适当的目录接口。该选项存在的理由是，某些人相信（或希望）truncate()比 unlink()性能表现更好。然而，压力测试显示两者有很少的或根本没有区别。

--disable-hostname-checks
默认的，squid 要求 URL 主机名在一定程度上遵守古老的 RFC 1034 规范：
标签必须遵循下列 ARPANET 主机名规则。它们必须以字母开始，以字母或数字结尾，仅仅包含字母，数字和下划线。
这里字母意味着 ASCII 字符，从 A 到 Z。既然国际域名日益流行，你可能希望使用该选项来移除限制。

--enable-underscores
该选项控制 squid 针对主机名里下划线的行为。通用的标准是主机名里不包含下划线字符，尽管有些人不赞成这点。squid 默认会对 URL 主机名里带下划线的请求产生一条错误消息。你能使用该选项，让 squid 信任它们，把它们当作合法的。然而，你的 DNS 解析器也许强迫使用非下划线请求，并且对带下划线的主机名解析失败。

--enable-auth[=LIST]
该选项控制在 squid 的二进制文件里支持哪种验证机制。你能选择下列机制的任意组合：basic,digest,ntlm。假如你忽略该选项，squid 仅仅支持 basic 验证。假如你使用不带参数的--enable-auth 选项，编译进程将增加对所有验证机制的支持。你可以使用以逗号分隔的验证机制列表：

% ./configure --enable-auth=digest,ntlm
我在第六章和第十二章里会谈得更多。

--enable-auth-helpers=LIST
这个旧选项现在已舍弃了， 但为了保持向后兼容性仍保留着。你可以使用--enable-basic-auth-helperes=LIST 来代替。

--enable-basic-auth-helpers=LIST
使用该选项，你能将 helpers/basic_auth 目录的一个或多个 HTTP Basic 验证辅助程序编译进来。请见 12.2 章找到它们的名字和描述。

--enable-ntlm-auth-helpers=LIST
使用该选项，你能将 helpers/ntlm_auth 目录的一个或多个 HTTP NTLM 验证辅助程序编译进来。请见 12.4 章找到它们的名字和描述。

--enable-digest-auth-modules=LIST
使用该选项，你能将 helpers/digest_auth 目录的一个或多个 HTTP Digest 验证辅助程序编译进来。请见 12.3 章找到它们的名字和描述。

--enable-external-acl-helpers=LIST
使用该选项，你能编译一个或多个扩展 ACL 辅助程序，这些在 12.5 章中讨论。例如：
% ./configure --enable-external-acl-helpers=ip_user,ldap_group

--disable-unlinkd
unlinkd 是另一个 squid 的外部辅助进程。它的基本工作是对 cache 文件执行 unlink()或 truncate()系统调用。通过在外部进程里执行文件删除工作，能给 squid 带来明显的性能提升。使用该选项来禁止外部 unlink 进程功能。

--enable-stacktrace
某些系统支持在程序崩溃时，自动产生数据追踪。当你激活该功能后，如果 squid 崩溃，数据追踪信息被写到 cache.log 文件。这些信息对开发和程序 bug 调试有用。

--enable-x-accelerator-vary
该高级功能可能在 squid 被配置成加速器时使用。它建议 squid 在响应请求时，从后台原始服务器中寻找 X-Accelerator-Vary 头。请见 15.5 章。

3.4.2 运行 configure

现在我们准备运行./configure 脚本。进入源代码的顶级目录敲入./configure，后面跟上前面提到过的任意选项，例如：
% cd squid-2.5.STABLE4
% ./configure --enable-icmp --enable-htcp

./configure 的工作就是侦察你的操作系统，以发现什么东西可用，什么不可用。它首先做的事

情之一就是确认你的 C 编译器可用。假如./configure 检测到你的 C 编译器有问题，脚本会退出，返回如下错误：
configure: error: installation or configuration problem: C compiler
cannot create executables.

很可能你从不会看到这个消息。假如看到了，那意味着你的系统中没有 C 编译器存在，或者编译器没有正确安装。请见 config.log 文件找到解决问题的建议。假如你的系统中有多个 C 编译器，你可以在运行./configure 之前设置 CC 环境变量，来告诉./configure 使用哪个：
% setenv CC /usr/local/bin/gcc
% ./configure ...

在./configure 检查完该编译器后，它查找头文件，库文件和函数的长列表。通常你不必担心该部分。在某些实际情况中，./configure 会终止以引起你的注意，某些事情可能有问题，例如没有足够的文件描述符。假如你指定不完整的或不合理的命令行选项，它也会终止。假如有错误发生，请检查 config.log 输出。./configure 的最终任务是创造 Makefiles 和其他文件，这些文件基于 squid 从你系统中了解到的知识。到此为止，你准备做编译工作。

3.5 编译

一旦./configure 完成了它的工作，你简单的敲入 make 开始编译源代码：
%make

正常来说，该过程很顺利，你可以见到大量的滚动行。
你也许见到一些编译器警告。大多数情况下，可以安全的忽略这些。假如这些警告非常多，并且一些看起来非常严重，请将它们报告给开发者，在第 16.5 章中有描述。
假如编译过程没有错误，你可以转移到下一节，描述如何安装你刚才编译的程序。
为了验证编译是否成功，你可以再次运行 make。你将看到如下输出：
% make
Making all in lib...
Making all in scripts...
Making all in src...
Making all in fs...
Making all in repl...
'squid' is up to date.
'client' is up to date.
'unlinkd' is up to date.
'cachemgr.cgi' is up to date.
Making all in icons...
Making all in errors...
Making all in auth_modules...

因为许多理由，编译步骤也许会失败，包括：

源代码 bugs

通常 squid 源代码是完整的调试过的。然而，你也许会遇到某些 bugs 或问题从而阻止你编译。这种问题在新的开发版本中更容易出现，请将它们报告给开发者。

编译器安装问题

不正确安装的 C 编译器不能够编译 squid 或其他软件包。通常编译器随着操作系统预安装，所以你不必担心它。然而，假如你在操作系统安装完后，试图升级编译器，那么可能会犯错误。绝对不要把已经安装好的编译器从一台机器拷贝到另一台，除非你绝对清楚你在做什么。我觉得在每台机上独立的安装编译器总是最好的。

请确认你的编译器的头文件总是与库文件同步。头文件通常在/usr/include 目录，而库文件在/usr/lib 目录。Linux 的流行 RPM 系统允许它去升级其中之一，但并非另一个。假如库文件基于不同的头文件，squid 不能编译。

假如你想在开源 BSD 变种之一中升级编译器，请确认在/usr/src 目录中运行 make world，这好过从/usr/src/lib 或/usr/src/include 中运行。

如下是一些通用的编译器问题和错误消息：

Solaris: make[1]: *** [libmiscutil.a] Error 255
这意味着./configure 不能发现 ar 程序。请确认/usr/ccs/bin 位于你的 PATH 环境变量里。假如你没有安装 Sun 的编译器，那么需要 GNU 的工具。(http://www.gnu.org/directory/binutils.html).

Linux: storage size of 'rl' isn't known
这是因为头文件和库文件不匹配所致，象前面描述的一样。请确认同时升级两者。

Digital Unix: Don't know how to make EXTRA_libmiscutil_a_SOURCES. Stop.
Digital Unix 的 make 程序不能兼容 automake 包产生的 Makefile 文件。例如，lib/Makefile.in 包含如下行：
noinst_LIBRARIES = \
@LIBDLMALLOC@ \
libmiscutil.a \
libntlmauth.a \
@LIBREGEX@
在替换后，当 lib/Makefile 被创建时，它看起来如下：
noinst_LIBRARIES = \
\
libmiscutil.a \
libntlmauth.a \
<TAB>;

象上面显示的一样，最后一行包括一个不可见的 TAB 字符，它阻止了 make。通过安装和使用 GNU make，或者手工编辑 lib/Makefile 如下，来解决这个问题：

```
noinst_LIBRARIES = \
\
libmiscutil.a \
libntlmauth.a
```

假如你在编译 squid 时遇到问题，请先检查 FAQ。你也许该在 Squid 的 web 站点上搜索（使用主页里的搜索栏）。最后，假如你仍有问题，请发邮件到 squid-users@squid-cache.org 列表。

3.6 安装

在编译完后，你需要把程序安装到指定的目录。可能需要超级用户权限来把它们放置到安装目录。所以，请先切换到 root：

```
%su
password:
#make install
```

假如你通过使用--enable-icmp 选项，激活了 squid 的 ICMP 衡量功能，那么必须安装 pinger 程序。pinger 程序必须以超级用户权限安装，因为仅仅允许 root 来发送和接受 ICMP 消息。

下列命令以相应的许可来安装 pinger 程序：

```
#make install-pinger
```

在安装完后，你将在 squid 的安装目录里（默认是/usr/local/squid）见到下列目录和文件：

sbin
sbin 目录的程序正常只能被 root 启动

sbin/squid
Squid 的主程序

bin
bin 目录包含对所有用户可用的程序

bin/RunCache
RunCache 是一个 shell 脚本，你能用它来启动 squid。假如 squid 死掉，该脚本自动重启它，除非它检测到经常的重启。RunCache 是一个时间遗留的产物，那时 Squid 还不是后台服务进程。在最近的版本里，RunCache 很少用到，因为 Squid 自动重启它自身，当你不使用-N 选项时。

bin/RunAccel

RunAccel 与 RunCache 几乎一致，唯一的不同是它增加了一个命令行参数，告诉 squid 在哪里侦听 HTTP 请求。

bin/squidclient
squidclient 是个简单的 HTTP 客户端程序，你能用它来测试 squid。它也有一些特殊功能，用以对运行的 squid 进程发起管理请求。

libexec
libexec 目录传统的包含了辅助程序。有一些命令你不能正常的启动。然而，这些程序通常被其他程序启动。

libexec/unlinkd
unlinkd 是一个辅助程序，它从 cache 目录里删除文件。如你后面看到的一样，文件删除是个性能瓶颈。通过在外部进程里执行删除操作，Squid 提升了一些执行性能。

libexec/cachemgr.cgi
cachemgr.cgi 是 Squid 管理功能的 CGI 接口。为了使用它，你需要拷贝该程序到你的 WEB 服务器的 cgi-bin 目录。在 14.2 章中有更多描述。

libexec/diskd(optional)
假如你指定了--enable-storeio=diskd，你才能看到它。

libexec/pinger(optional)
假如你指定了--enable-icmp，你才能看到它。

etc
etc 目录包含 squid 的配置文件。

etc/squid.conf
这是 squid 的主要配置文件。初始的该文件包含了大量的注释，用以解释每一个选项做什么。在你理解了这些配置指令后，建议你删除这些注释，让配置文件更小和更容易阅读。注意假如该文件存在，安装过程不会覆盖该文件。

etc/squid.conf.default
这是从源代码目录中拷贝过来的默认配置文件。在升级了 squid 安装后，你也许发现有一份当前默认配置文件的拷贝是有用的。可能会增加新的配置指令，一些存在的旧指令可能有所改变。

etc/mime.conf
mime.conf 文件告诉 squid 对从 FTP 和 Gopher 服务器获取的数据使用何种 MIME 类型。该文件是一个关联文件名扩展到 MIME 类型的表。正常而言，你不必编辑该文件。然而，你可能需要增加特殊文件类型的接口，它们在你的组织内使用。

etc/mime.conf.default
这是从源代码目录里拷贝过来的默认 mime.conf 文件。

share
share 目录通常包括 squid 的只读数据文件。

share/mib.txt
这是 squid 的 SNMP 管理信息基础（MIB）文件。squid 自身不使用该文件，然而，你的 SNMP 客户端软件（例如 snmpget 和多路由走向图(MRTG)）需要该文件，用以理解来自 squid 的 SNMP 对象可用。

share/icons
share/icons 目录包含大量的小图标文件，squid 用在 FTP 和 Gopher 目录列举里。正常而言，你不必担心这些文件，但如果需要，你可以改变它们。

share/errors
share/errors 目录包含了 squid 显示给用户看的错误消息模板。这些文件在你安装 squid 时，从源代码目录拷贝而来。如果需要你可以编辑它们。然而，在每次运行 make install 时，安装过程总会覆盖它们。所以假如你想定制错误消息，建议你把它们放在不同的目录。

var
var 目录包含了不是很重要的和经常变化的文件。这些文件你不必正常的备份它们。

var/logs
var/logs 目录是 squid 不同日志文件的默认位置。当你第一次安装 squid 时，它是空的。一旦 squid 开始运行，你能在这里看到名字为 access.log,cache.log 和 store.log 这样的文件。

var/cache
假如你不在 squid.conf 文件里指定，这是默认的缓存目录(cache_dir)。第七章有关于缓存目录的所有细节。

3.7 打补丁

在你运行 squid 一段时间后，你可能发现需要打源代码补丁，用以修正 bug 或者增加试验性的功能。在 squid-cache.org 站点上，对重要的 bug 修正会发布补丁。假如你不想等到下一个官方发布版本，你能下载补丁，并且打到你的源代码中。然后你需要重新编译 squid。

为了打补丁-或者有时候叫差别文件-你需要一个叫做"patch"的程序。你的操作系统必须有该程序。如果没有，你可以从 GNU 工具集里下载(http://www.gnu.org/directory/patch.html).

注意假如你在使用匿名 CVS（见 2.4 节），你不必担心补丁文件。当你升级源代码树时，CVS 系统自动升级了补丁。

为了打补丁，你必须把补丁文件存放在系统中某处。然后进入到 squid 的源代码目录，运行如下命令：

% cd squid-2.5.STABLE4
% patch < /tmp/patch_file

默认的，在 patch 程序运行时，它告诉你它正在做什么。通常输出滚动非常快，除非有问题。你能安全的忽略它输出的 offset NNN lines 警告。假如你不想见到所有这些输出，使用-s 选项选择安静模式。

当补丁更新了源代码后，它创造了原始文件的拷贝。例如，假如你对 src/http.c 打一个补丁，备份文件名就是 src/http.c.orig。这样，假如你在打了补丁后想撤销这个操作，简单的重命名所有的.orig 文件到它们以前的格式。为了成功的使用该技术，建议你在打补丁之前删除所有的.orig 文件。

假如 patch 程序遇到问题，它停止运行并且给出建议。通常问题如下：

在错误的目录运行 patch 程序——解决的方法是，进入到正确的目录，或者使用 patch 的-p 选项。

补丁已打过——patch 会告诉你是否已打过补丁文件。在这样的情况下，它会问你是否撤销这个文件的补丁。

patch 程序不能理解你赋给它的文件——补丁文件通常有三个风格：正常的，context 的和 unified 的。旧版本的 patch 程序可能不理解后两者的差异输出。从 GNU 的 FTP 站点获取最近的版本能解决该问题。

损坏的补丁文件——假如你在下载和存储补丁文件时不小心，它有可能被损坏。有时候人们以 email 消息发送补丁文件，在新的窗口里，它们被简单的剪切和粘贴。

在这样的系统中，剪切和粘贴能将 Tab 字符改变为空格，或者不正确的捆绑长行。这些改变混乱了 patch。-l 选项也许有用，但最好是正确的拷贝和存储补丁文件。

某些时候 patch 不能应用部分或所有的差别文件——在这样的情况下，你能见到类似于 Hunk 3 of 4 failed 的消息。失败的部分被存储在命名为.rej 的文件里。例如，假如在处理 src/http.c 时失败，patch 程序将该差别文件片断存为 src/http.c.rej。在这样的情况下，你也许能手工修正这些问题，但它通常不值得这么做。假如你有大量的"failed hunks"或者.rej 文件，建议你去下载最近源代码版本的完整新拷贝。

在你打完补丁后，你必须重新编译 squid。make 的先进功能之一就是它仅仅编译改变了的文件。但有时候 make 不能理解错综复杂的依赖关系，它没有完整的重编译所需文件。为了安全起见，通常建议你去重编译所有文件。最好的方法是在开始编译之前清除源代码树：

%make clean
%make

3.8 重运行 configure

有时候你可能发现有必要重新运行./configure。例如，假如你调整了内核参数，你必须再次运行./configure 以使它能发现新设置。当你阅读本书时，你也发现你必须使用./configure 选项来激活所需的功能。

以相同的选项重运行./configure，使用如下命令：
%config.status --recheck

另一个技术是`touch config.status`文件，它更新了该文件的时间戳。这导致 make 在编译源代码之前，重新运行./configure 脚本：
% touch config.status
% make

如果增加或删除./configure 选项，你必须重新敲入完整的命令行。假如你记不住以前的选项，请查看 config.status 文件的顶部。例如：
% head config.status
#! /bin/sh
Generated automatically by configure.
Run this file to recreate the current configuration.
Squid 中文权威指南 16
This directory was configured as follows,
on host foo.life-gone-hazy.com:
#
./configure --enable-storeio=ufs,diskd --enable-carp \
--enable-auth-modules=NCSA
Compiler output produced by configure, useful for debugging
configure, is in ./config.log if it exists.

在运行./configure 之后，你必须再次编译和安装 squid。安全起见，建议先运行 make clean：
%make clean
%make

请回想一下，./configure 会缓存它在你系统中发现的东西。在这样的形式下，你可能想清除这些缓存，从头开始编译过程。假如喜欢，你可以简单的删除 config.cache 文件。然后，下一次./configure 运行时，它不会使用以前的数值。你也能恢复 squid 源代码树到它的 configure 之前的状态，使用如下命令：
%make distclean

这将删除所有的目标文件和其他被./configure 和 make 程序产生的文件。

第 4 章 快速配置向导

4.1 squid.conf 语法

Squid 的配置文件相对规范。它与其他许多 unix 程序相似。每行以配置指令开始，后面跟着数字值或关键字。在读取配置文件时，squid 忽略空行和注释掉的行（以#开始）。如下是一些配置行示例：

```
cache_log /squid/var/cache.log
# define the localhost ACL
acl Localhost src 127.0.0.1/32
connect_timeout 2 minutes
log_fqdn on
```

某些指令取唯一值。在这些情形下，重复赋予该指令不同的值，将覆盖前面的值。例如，下面是一个连接超时值。第一行无效，因为第二行覆盖了它：

```
connect_timeout 2 minutes
connect_timeout 1 hour
```

另外，某些指令取列表值。在这些情形下，每一个新增的值都有效。"扩展方式"指令以这种方法工作：

```
extension_methods UNGET
extension_methods UNPUT
extension_methods UNPOST
```

对这些基于列表的指令，你通常能在同一行中赋予多个值：

```
extension_methods UNGET UNPUT UNPOST
```

许多指令有通用类型。例如，连接超时值是一个时间规范，在数字后面跟着时间单元。例如：

```
connect_timeout 3 hours
client_lifetime 4 days
negative_ttl 27 minutes
```

类似的，大量的指令指向文件大小或者内存额度。例如，你可以这样编写大小规范：十进制数字后面跟 bytes,KB,MB 或 GB.例如：

```
minimum_object_size 12 bytes
request_header_max_size 10 KB
maximum_object_size 187 MB
```

另一种值得提起的类型是触发器，它的值是 on 或者 off。许多指令使用该类型。例如：

```
server_persistent_connections on
strip_query_terms off
prefer_direct on
```

通常，配置文件指令能以任何顺序出现。然而，如果某个指令指向的值被其他指令所定义，那么顺序就很重要。访问控制列表是个好的例子。acl 被用在 http_access 规则之前必须被定义：

```
acl Foo src 1.2.3.4
http_access deny Foo
```

squid.conf 文件里的许多东西是大小写敏感的，例如指令名。你不能将 http_port 写成 HTTP_port。

默认的 squid.conf 文件包含了对每个指令的大量注释，以及指令的默认值。例如：

```
# TAG: persistent_request_timeout
# How long to wait for the next HTTP request on a persistent
# connection after the previous request completes.
#
#Default:
# persistent_request_timeout 1 minute
```

每次安装 squid 后，当前默认配置文件存放在$prefix/etc 目录下的 squid.conf.default。既然指令每次都有所改变，你能参考该文档，以获取最近的更新。

该章剩下的部分是关于在开始运行 squid 之前，你必须知道的少数指令。

4.2 User IDs

你可能知道，unix 进程和文件拥有文件和组属主的属性。你必须选择某个用户和组给 squid。该用户和组的组合，必须对大部分 squid 相关的文件和目录有读和写的权限。

我高度推荐创建名为"squid"的用户和组。这避免了某人利用 squid 来读取系统中的其他文件。假如不止一个人拥有对 squid 的管理权限，你可以将他们加到 squid 组里。

unix 进程继承了它们父进程的属主属性。那就是说，假如你以 joe 用户来启动 squid，squid 也以 joe 来运行。假如你不想以 joe 来运行 squid，你需要预先改变你的用户 ID。这是 su 命令的典型功能。例如：

```
joe% su - squid
squid% /usr/local/squid/sbin/squid
```

不幸的是，运行 squid 并非总是如此简单。在某些情况下，你必须以 root 来启动 squid，这依赖于你的配置。例如，仅仅 root 能绑定 TCP 套接字到特权端口上，如 80。假如你必须以 root 来启动 squid，你必须设置 cache_effective_user 指令。它告诉 squid，在执行完需要特别权限的任务后，变成哪个用户。例如：

```
cache_effective_user squid
```

你提供的该名字必须是有效用户（在/etc/passwd 文件里）。请注意仅仅当你以 root 来启动 squid 时，你才需要用到该指令。仅仅 root 有能力来随意改变用户身份。假如你以 joe 来启动 squid，它不能改变到 squid 用户。

你可能尝试不设置 cache_effective_user，直接以 root 来运行 squid。假如你试过，你会发现 squid 拒绝运行。这违背了安全规则。假如外部攻击者有能力危及或利用 squid，他能获取

对系统的全部访问权。尽管我们努力使 squid 安全和少 bug，但还是稳重点好。

假如你没有设置 cache_effective_user,以 root 来启动 squid，squid 使用 nobody 作为默认值。不管你选择什么用户 ID，请确认它有对下面目录的读访问权：$prefix/etc,$prefix/libexec,$prefix/share。该用户 ID 也必须有对日志文件和缓存目录的写访问
权。

squid 也有一个 cache_effective_group 指令，但你也许不必设置它。默认的，squid 使用 cache_effective_user 的默认组（从/etc/passwd 文件读取）。

4.3 端口号

http_port 指令告诉 squid 在哪个端口侦听 HTTP 请求。默认端口是 3128：
http_port 3128

假如你将 squid 作为加速器运行（见 15 章），你也许该将它设为 80。

你能使用附加的 http_port 行，来指示 squid 侦听在多个端口上。假如你必须支持客户组（它们被配置得不一致），这点就经常有用。例如，来自某个部门的浏览器发送请求到 3128，
然而另一个部门使用 80 端口。简单的将两个端口号列举出来：
http_port 3128
http_port 8080

你也能使用 http_port 指令来使 squid 侦听在指定的接口地址上。当 squid 作为防火墙运行时，它有两个网络接口：一个内部的和一个外部的。你可能不想接受来自外部的 http 请求。为了使 squid 仅仅侦听在内部接口上，简单的将 IP 地址放在端口号前面：
http_port 192.168.1.1:3128

4.4 日志文件路径

我将在第 13 章讨论所有 squid 的日志细节。你现在你关注的唯一事情是，squid 将它的日志放在何处。默认的日志目录是 squid 安装位置下的 logs 目录。例如，假如你在./configure 时没有使用--prefix=选项，那么默认的日志文件路径是/usr/local/squid/var/logs。

你必须确认日志文件所存放的磁盘位置空间足够。在 squid 写日志时如果接受到错误，它会退出和重启。该行为的主要理由应引起你的注意。squid 想确认你不会丢失任何重要的日志信息，特别是你的系统被滥用或者被攻击时。

squid 有三个主要的日志文件：cache.log，access.log，store.log.第一个文件即 cache.log，包含状态性的和调试性的消息。当你刚开始运行 squid 时，你应密切的关注该文件。假如 squid 拒绝运行，理由也许会出现在 cache.log 文件的结尾处。在正常条件下，该文件不会变得很大。

也请注意，假如你以-s 选项来运行 squid，重要的 cache.log 消息也可被送到你的 syslog 进程。通过使用 cache_log 指令，你可以改变该日志文件的路径：
cache_log /squid/logs/cache.log

access.log 文件包含了对 squid 发起的每个客户请求的单一行。每行平均约 150 个字节。也就是说，在接受一百万条客户请求后，它的体积约是 150M。请使用 cache_access_log 指令来改变该日志文件的路径：
cache_access_log /squid/logs/access.log

假如因为某些理由，你不想 squid 记录客户端请求日志，你能指定日志文件的路径为/dev/null.

store.log 文件对大多数 cache 管理员来说并非很有用。它包含了进入和离开缓存的每个目标的记录。平均记录大小典型的是 175-200 字节。然而，squid 不在 store.log 里对 cache 点击创建接口，所以它比 access.log 包含少得多的记录。请使用 cache_store_log 指令来改变它的位置：
cache_store_log /squid/logs/store.log

通过指定路径为 none，你能轻易的完全禁止 store.log 日志：
cache_store_log none

假如你不小心，squid 的日志文件增加没有限制。某些操作系统对单个文件强制执行 2G 的大小限制，即使你有充足的磁盘空间。超过该限制会导致写错误，这样 squid 就会退出。为了保证日志文件大小合理，你应创建任务来有规律的重命名和打包日志。squid 有内建功能来使这个容易做到。请见 13.7 章关于日志轮循的解释。

4.5 访问控制

在第 6 章里有更多的关于访问控制的描述。现在，我只讲述少量的访问控制方法，以使热心的读者能快速开始使用 squid。

squid 默认的配置文件拒绝每一个客户请求。在任何人能使用代理之前，你必须在 squid.conf 文件里加入附加的访问控制规则。最简单的方法就是定义一个针对客户 IP 地址的 ACL 和一个访问规则，告诉 squid 允许来自这些地址的 HTTP 请求。squid 有许多不同的 ACL 类型。src 类型匹配客户 IP 地址，squid 会针对客户 HTTP 请求检查 http_access 规则。这样，你需要增加两行：
acl MyNetwork src 192.168.0.0/16
http_access allow MyNetwork

请将这些行放在正确的位置。http_access 的顺序非常重要，但是 acl 行的顺序你不必介意。你也该注意默认的配置文件包含了一些重要的访问控制，你不应该改变或删除它们，除非你完全理解它们的意义。在你第一次编辑 squid.conf 文件时，请看如下注释：
INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS

在该注释之后，以及"http_access deny all"之前插入你自己的新规则。

为了彻底说明，如下是一个合理的初始访问控制配置，包括推荐的默认控制和早先的例子：

```
acl All src 0/0
acl Manager proto cache_object
acl Localhost src 127.0.0.1/32
acl Safe_ports port 80 21 443 563 70 210 280 488 591 777 1025-65535
acl SSL_ports 443 563
acl CONNECT method CONNECT
acl MyNetwork src 192.168.0.0/16
http_access allow Manager Localhost
http_access deny Manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow MyNetwork
http_access deny All
```

4.6 可见主机名

希望你不必担心 visible_hostname 指令。然而，假如 squid 不能发现它所运行的机器的主机名，你就必须设置它。如果发生这样的事，squid 抱怨和拒绝运行：

```
% squid -Nd1
FATAL: Could not determine fully qualified hostname. Please set 'visible_hostname'
```

有大量的理由使 squid 需要知道主机名：

+ 主机名出现在 squid 的错误消息里，这帮助用户验证潜在问题的源头。
+ 主机名出现在 squid 转发的 cache 单元的 HTTP Via 头里。当请求到达原始主机时，Via 头包含了在传输过程中涉及的代理列表。squid 也使用 Via 头来检测转发环路。我将在第 10 章里讨论转发环路。

squid 对特定事务使用内部 URL，例如 FTP 目录列表的图标。当 squid 对 FTP 目录产生 HTML 页面时，它插入小图标用以指明该目录中的文件类型。图标 URL 包含了 cache 的主机名，以便 web 浏览器能直接从 squid 请求它们。

每个从 squid 响应的 HTTP 回复包含了 X-Cache 头。这并非官方 HTTP 头。它是一个扩展头，用以指明该响应是 cache 点击还是 cache 丢失。既然请求和响应可能经过多个 cache，每个 X-Cache 头包含了 cache 报告点击或丢失的名字。如下是一个通过 2 个 cache 的响应示例：

```
HTTP/1.0 200 OK
Date: Mon, 29 Sep 2003 22:57:23 GMT
Content-type: text/html
Content-length: 733
X-Cache: HIT from bo2.us.ircache.net
```

X-Cache: MISS from bo1.us.ircache.net

squid 在启动时试图自动获取主机名。首先它调用 gethostname()函数，这通常能返回正确的主机名。接着，squid 调用 gethostbyname()函数尝试对主机名进行 DNS 查询。该函数典型的返回 IP 地址和系统的规范名。假如 gethostbyname()成功，squid 在错误消息里，Via 头里等地方使用这个规范名。

因为大量的理由，squid 可能不能检测到它的规范主机名，包括：
+ 主机名可能未设置。
+ 主机名可能从 DNS 区域或/etc/hosts 文件里丢失。

squid 系统的 DNS 客户端配置可能不正确或丢失。在 unix 系统上，你该检查 /etc/resolv.conf 和/etc/host.conf 文件。

假如你看到上述的致命错误，你必须修正主机名和 DNS 信息，或者显式的给 squid 指明主机名。在大多数情况下，请确认"hostname"命令返回一个完全规范的主机名，并且在/etc/hosts 文件里增加这个接口。假如这样不成功，请在 squid.conf 里设置可见主机名：
visible_hostname squid.packet-pushers.net

4.7 管理联系信息

你应该设置 cache_mgr 指令作为对用户的帮助。它是一个 email 地址，假如问题发生，用户能写信给它。cache_mgr 地址默认出现在 squid 的错误消息里。例如：
cache_mgr squid@web-cache.net

4.8 下一步

在创建了初步的配置文件后，你多少准备首次运行 squid 了。请遵循下面章节的建议。

当你掌握了启动和停止 squid 后，你该花费一些时间来改善配置文件。你可能想增加更高级的访问控制，这在第 6 章里有描述。既然我在这里没有讨论磁盘 cache，你该花些时间阅读第 7 和第 8 章。

第 5 章 运行 Squid

5.1 squid 命令行选项

在开始其他事情之前，让我们先看一下 squid 的命令行选项。这里的许多选项你从不会使用，另外有些仅仅在调试问题时有用。

-a port

指定新的 http_port 值。该选项覆盖了来自 squid.conf 的值。然而请注意，你能在 squid.conf 里指定多个值。-a 选项仅仅覆盖配置文件里的第一个值。（该选项使用字母 a 是因为在 Harvest cache 里，HTTP 端口被叫做 ASCII 端口）

-d level
让 squid 将它的调试信息写到标准错误（假如配置了，就是 cache.log 和 syslog）。level 参数指定了显示在标准错误里的消息的最大等级。在多数情况下，d1 工作良好。请见 16.2 章关于调试等级的描述。

-f file
指定另一个配置文件。

-h
显示用法。

-k function
指示 squid 执行不同的管理功能。功能参数是下列之一：reconfigure，rotate，shutdown，interrupt，kill，debug，check，or parse。
+ reconfigure 导致运行中的 squid 重新读取配置文件。
+ rotate 导致 squid 滚动它的日志，这包括了关闭日志，重命名，和再次打开它们。
+ shutdown 发送关闭 squid 进程的信号。
+ interrupt 立刻关闭 squid，不必等待活动会话完成。
+ kill 发送 KILL 信号给 squid，这是关闭 squid 的最后保证。
+ debug 将 squid 设置成完全的调试模式，假如你的 cache 很忙，它能迅速的用完你的磁盘空间。
+ check 简单的检查运行中的 squid 进程，返回的值显示 squid 是否在运行。
+ 最后，parse 简单的解析 squid.conf 文件，如果配置文件包含错误，进程返回非零值。

-s
激活将日志记录到 syslog 进程。squid 使用 LOCAL4 syslog 设备。0 级别调试信息以优先级 LOG_WARNING 被记录，1 级别消息以 LOG_NOTICE 被记录。更高级的调试信息不会被发送到 syslogd.你可以在/etc/syslogd.conf 文件里使用如下接口：
local4.warning /var/log/squid.log

-u port
指定另一个 ICP 端口号，覆盖掉 squid.conf 文件里的 icp_port。

-v
打印版本信息。

-z
初始化 cache，或者交换，目录。在首次运行 squid，或者增加新的 cache 目录时，你必须使用该选项。

-C
阻止安装某些信号句柄，它们捕获特定的致命信号例如 SIGBUS 和 SIGSEGV。正常的，这些信号被 squid 捕获，以便它能干净的关闭。然而，捕获这些信号可能让以后调试问题困难。使用该选项，致命的信号导致它们的默认动作，通常是 coredump。

-D
禁止初始化 DNS 测试。正常情况下，squid 直到验证它的 DNS 可用才能启动。该选项阻止了这样的检测。你也能在 squid.conf 文件里改变或删除 dns_testnames 选项。

-F
让 squid 拒绝所有的请求，直到它重新建立起存储元数据。假如你的系统很忙，该选项可以减短重建存储元数据的时间。然而，如果你的 cache 很大，重建过程可能会花费很长的时间。

-N
阻止 squid 变成后台服务进程。

-R
阻止 squid 在绑定 HTTP 端口之前使用 SO_REUSEADDR 选项。

-V
激活虚拟主机加速模式。类似于 squid.conf 文件里的 httpd_accel_host virtual 指令。

-X
强迫完整调试模式，如你在 squid.conf 文件里指定 debug_options ALL,9 一样。

-Y
在重建存储元数据时，返回 ICP_MISS_NOFETCH 代替 ICP_MISS.忙碌的父 cache 在重建时，该选项可以导致最少的负载。请见 10.6.1.2 章。

5.2 对配置文件查错

在开启 squid 之前，你应该谨慎的验证配置文件。这点容易做到，运行如下命令即可：
%squid -k parse

假如你看不到输出，配置文件有效，你能继续后面的步骤。

然而，如果配置文件包含错误，squid 会告诉你：
squid.conf line 62: http_access allow okay2
aclParseAccessLine: ACL name 'okay2' not found.

这里你可以看到，62 行的 http_access 指令指向的 ACL 不存在。有时候错误信息很少：
FATAL: Bungled squid.conf line 76: memory_pools

在这个情形里，我们忘记了在 76 行的 memory_pools 指令后放置 on 或 off。

建议你养成习惯：在每次修改配置文件后，使用 squid -k parse。假如你不愿麻烦，并且你的配置文件有错误，squid 会告诉你关于它们而且拒绝启动。假如你管理着大量的 cache，也许你会编辑脚本来自动启动，停止和重配置 squid。你能在脚本里使用该功能，来确认配置文件是有效的。

5.3 初始化 cache 目录

在初次运行 squid 之前，或者无论何时你增加了新的 cache_dir，你必须初始化 cache 目录。命令很简单：

```
%squid –z
```

对 UFS 相关的存储机制（ufs,aufs,and diskd;见第 8 章），该命令在每个 cache_dir 下面创建了所需的子目录。你不必担心 squid 会破坏你的当前 cache 目录（如果有的话）。

在该阶段属主和许可权是通常遇到的问题。squid 在特定的用户 ID 下运行，这在 squid.conf 文件里的 cache_effective_user 里指定。用户 ID 必须对每个 cache_dir 目录有读和写权限。否则，你将看到如下信息：

```
Creating Swap Directories
FATAL: Failed to make swap directory /usr/local/squid/var/cache/00:
(13) Permission denied
```

在这样的情形下，你该确认/usr/local/squid/var/cache 目录的所有组成都可被 squid.conf 给定的用户 ID 访问。最终的组件--cache 目录--必须对该用户 ID 可写。

cache 目录初始化可能花费一些时间，依赖于 cache 目录的大小和数量，以及磁盘驱动器的速度。假如你想观察这个过程，请使用-X 选项：

```
%squid –zX
```

5.4 在终端窗口里测试 squid

一旦你已经初始化 cache 目录，就可以在终端窗口里运行 squid，将日志记录到标准错误。这样，你能轻易的定位任何错误或问题，并且确认 squid 是否成功启动。使用-N 选项来保持 squid 在前台运行，-d1 选项在标准错误里显示 1 级别的调试信息。

```
%squid -N -d1
```

你将看到类似于以下的输出：

```
2003/09/29 12:57:52| Starting Squid Cache
version 2.5.STABLE4 for i386-unknown-freebsd4.8...
2003/09/29 12:57:52| Process ID 294
2003/09/29 12:57:52| With 1064 file descriptors available
```

```
2003/09/29 12:57:52| DNS Socket created on FD 4
2003/09/29 12:57:52| Adding nameserver 206.107.176.2 from /etc/resolv.conf
2003/09/29 12:57:52| Adding nameserver 205.162.184.2 from /etc/resolv.conf
2003/09/29 12:57:52| Unlinkd pipe opened on FD 9
2003/09/29 12:57:52| Swap maxSize 102400 KB, estimated 7876 objects
2003/09/29 12:57:52| Target number of buckets: 393
2003/09/29 12:57:52| Using 8192 Store buckets
2003/09/29 12:57:52| Max Mem size: 8192 KB
2003/09/29 12:57:52| Max Swap size: 102400 KB
2003/09/29 12:57:52| Rebuilding storage in /usr/local/squid/var/cache (DIRTY)
2003/09/29 12:57:52| Using Least Load store dir selection
2003/09/29 12:57:52| Set Current Directory to /usr/local/squid/var/cache
2003/09/29 12:57:52| Loaded Icons.
2003/09/29 12:57:52| Accepting HTTP connections at 0.0.0.0, port 3128, FD 11.
2003/09/29 12:57:52| Accepting ICP messages at 0.0.0.0, port 3130, FD 12.
2003/09/29 12:57:52| WCCP Disabled.
2003/09/29 12:57:52| Ready to serve reques
```

假如你看到错误消息，你该首先修正它。请检查输出信息的开始几行以发现警告信息。最普通的错误是文件/目录许可问题，和配置文件语法错误。假如你看到一条不引起注意的错误消息，请见 16 章中关于 squid 故障处理的建议和信息。如果还不行，请检查 squid FAQ，或查找邮件列表来获得解释。

一旦你见到"Ready to serve requests"消息，就可用一些 HTTP 请求来测试 squid。配置你的浏览器使用 squid 作为代理，然后打开某个 web 页面。假如 squid 工作正常，页面被迅速载入，就象没使用 squid 一样。另外，你可以使用 squidclient 程序，它随 squid 发布：

```
% squidclient http://www.squid-cache.org/
```

假如它正常工作，squid 的主页 html 文件会在你的终端窗口里滚动。一旦确认 squid 工作正常，你能中断 squid 进程（例如使用 ctrl-c）并且在后台运行 squid。

5.5 将 squid 作为服务进程运行

正常情况下你想将 squid 以后台进程运行（不出现在终端窗口里）。最容易的方法是简单执行如下命令：

```
%squid –s
```

-s 选项导致 squid 将重要的状态和警告信息写到 syslogd。squid 使用 LOCAL4 设备，和 LOG_WARNING 和 LOG_NOTICE 优先权。syslog 进程实际可能会或不会记录 squid 的消息，这依赖于它被如何配置。同样的消息被写进 cache.log 文件，所以假如你愿意，忽略-s 选项也是安全的。

当你不使用-N 选项来启动 squid，squid 自动在后台运行并且创建父/子进程对。子进程做所

有的实际工作。父进程确认子进程总在运行。这样，假如子进程意外终止，父进程启动另外一个子进程以使 squid 正常工作。通过观察 syslog 消息，你能看到父/子进程交互作用。
Jul 31 14:58:35 zapp squid[294]: Squid Parent: child process 296 started

这里显示的父进程 ID 是 294，子进程是 296。当你查看 ps 的输出，你可以看到子进程以(squid)形式出现：
%ps ax | grep squid
294 ?? Is 0:00.01 squid -sD
296 ?? S 0:00.27 (squid) -sD (squid)

假如 squid 进程意外终止，父进程启动另一个。例如：
Jul 31 15:02:53 zapp squid[294]: Squid Parent: child process 296 exited due to signal 6
Jul 31 15:02:56 zapp squid[294]: Squid Parent: child process 359 started

在某些情形下，squid 子进程可能立即终止。为了防止频繁的启动子进程，假如子进程连续 5 次没有运行至少 10 秒钟，父进程会放弃。
Jul 31 15:13:48 zapp squid[455]: Squid Parent: child process 474 exited with status 1
Jul 31 15:13:48 zapp squid[455]: Exiting due to repeated, frequent failures

如果发生这样的事，请检查 syslog 和 squid 的 cache.log 以发现错误。

5.5.1 squid_start 脚本

当 squid 以后台进程运行时，它查找 squid 执行程序目录下的名为 squid_start 的文件。假如发现，该程序在父进程创建子进程之前被执行。你能使用该脚本完成特定的管理任务，例如通知某人 squid 在运行，管理日志文件等。除非 squid_start 程序存在，squid 不会创建子进程 。

squid_start 脚本在你使用绝对或相对路径启动 squid 时才开始工作。换句话说，squid 不使用 PATH 环境变量来定位 squid_start.这样，你应该养成习惯这样启动 squid：
% /usr/local/squid/sbin/squid –sD

而不要这样：
%squid –sD

5.6 启动脚本

通常你希望 squid 在每次计算机重启后自动启动。对不同的操作系统，它们的启动脚本如何工作也很不同。我在这里描述一些通用的环境，但对你自己的特殊操作系统，也许该有特殊的处理方法。

5.6.1 /etc/rc.local

最容易的机制之一是/etc/rc.local 脚本。这是个简单的 shell 脚本，在每次系统启动时以 root 运行。使用该脚本来启动 squid 非常容易，增加一行如下：
/usr/local/squid/sbin/squid –s

当然你的安装位置可能不同，还有你可能要使用其他命令行选项。不要在这里使用 -N 选项。

假如因为某些理由，你没有使用 cache_effective_user 指令，你可以尝试使用 su 来让 squid 以非 root 用户运行：
/usr/bin/su nobody -c '/usr/local/squid/sbin/squid -s'

5.6.2 init.d 和 rc.d

init.d 和 rc.d 机制使用独立的 shell 脚本来启动不同的服务。这些脚本通常在下列目录之中：/sbin/init.d, /etc/init.d, /usr/local/etc/rc.d.脚本通常获取单一命令行参数，是 start 或 stop。某些系统仅仅使用 start 参数。如下是启动 squid 的基本脚本：

```
#!/bin/sh
# this script starts and stops Squid
case "$1" in
start)
/usr/local/squid/sbin/squid -s
echo -n ' Squid'
;;
stop)
/usr/local/squid/sbin/squid -k shutdown
;;
esac
```

Linux 用户可能在启动 squid 之前需要设置文件描述符限制。例如：

```
echo 8192 >; /proc/sys/fs/file-max
limit -HSn 8192
```

为了使用该脚本，先找到脚本存放的目录。给它一个有意义的名字，类似于其他的系统启动脚本。可以是 S98squid 或 squid.sh。通过重启计算机来测试该脚本，而不要假想它会正常工作。

5.6.3 /etc/inittab

某些操作系统支持另一种机制，是/etc/inittab 文件。在这些系统中，init 进程启动和停止基于运行等级的服务。典型的 inittab 接口类似如此：
sq:2345once:/usr/local/squid/sbin/squid –s

使用该接口，init 进程启动 squid 一次并且随后忘记它。squid 确认它驻留在运行状态，象前面描述的一样。或者，你能这样做：

```
sq:2345:respawn:/usr/local/squid/sbin/squid –Ns
```

这里我们使用了 respawn 选项，假如进程不存在 init 会重启 squid。假如使用 respawn，请确认使用-N 选项。

在编辑完 inittab 文件后，使用下面的命令来使 init 重新读取它的配置文件和启动 squid：

```
# init q
```

5.7 chroot 环境

某些人喜欢在 chroot 环境运行 squid。这是 unix 的功能，给予进程新的 root 文件系统目录。在 squid 受安全威胁时，它提供额外等级的安全保护。假如攻击者在某种程度上通过 squid 获取了对操作系统的访问权，她仅仅能访问在 chroot 文件系统中的文件。在 chroot 树之外的系统文件，她不可访问。

最容易在 chroot 环境里运行 squid 的方法是，在 squid.conf 文件里指定新的 root 目录，如下：

```
chroot /new/root/directory
```

chroot()系统调用需要超级用户权限，所以你必须以 root 来启动 squid。

chroot 环境不是为 unix 新手准备的。它有点麻烦，因为你必须在新的 root 目录里重复放置大量的文件。例如，假如默认的配置文件正常在/usr/local/squid/etc/squid.conf，并且你使用 chroot 指令，那么文件必须位于/new/root/directory/usr/local/squid/etc/squid.conf. 你必须将位于$prefix/etc,$prefix/share,$prefix/libexec 下的所有文件拷贝到 chroot 目录。请确认$prefix/var 和 cache 目录在 chroot 目录中存在和可写。

同样的，你的操作系统需要将大量的文件放在 chroot 目录里，例如/etc/resolv.conf 和 /dev/null.假如你使用外部辅助程序，例如重定向器（见 11 章）或者验证器（见 12 章），你也需要来自/usr/lib 的某些共享库。你可以使用 ldd 工具来查找给定的程序需要哪些共享库：

```
% ldd /usr/local/squid/libexec/ncsa_auth
/usr/local/squid/libexec/ncsa_auth:
libcrypt.so.2 =>; /usr/lib/libcrypt.so.2 (0x28067000)
libm.so.2 =>; /usr/lib/libm.so.2 (0x28080000)
libc.so.4 =>; /usr/lib/libc.so.4 (0x28098000)
```

你可以使用 chroot 命令来测试辅助程序：

```
# chroot /new/root/directory /usr/local/squid/libexec/ncsa_auth
/usr/libexec/ld-elf.so.1: Shared object "libcrypt.so.2" not found
```

更多的关于 chroot 的信息，请见你系统中 chroot()的 manpage。

5.8 停止 squid

最安全的停止 squid 的方法是使用 squid -k shutdown 命令：
%squid -k shutdown

该命令发送 TERM 信号到运行中的 squid 进程。在接受到 TERM 信号后，squid 关闭进来的套接字以拒收新请求。然后它等待一段时间，用以完成外出请求。默认时间是 30 秒，你可以在 shutdown_lifetime 指令里更改它。
假如因为某些理由，squid.pid 文件丢失或不可读，squid -k 命令会失败。在此情形下，你可以用 ps 找到 squid 的进程 ID，然后手工杀死 squid。例如：
%ps ax |grep squid

假如你看到不止一个 squid 进程，请杀死以(squid)显示的那个。例如：
% ps ax | grep squid
294 ?? Is 0:00.01 squid -sD
296 ?? S 0:00.27 (squid) -sD (squid)
% kill -TERM 296

在发送 TERM 信号后，你也许想查看日志，以确认 squid 已关闭：
% tail -f logs/cache.log
2003/09/29 21:49:30| Preparing for shutdown after 9316 requests
2003/09/29 21:49:30| Waiting 10 seconds for active connections to finish
2003/09/29 21:49:30| FD 11 Closing HTTP connection
2003/09/29 21:49:31| Shutting down...
2003/09/29 21:49:31| FD 12 Closing ICP connection
2003/09/29 21:49:31| Closing unlinkd pipe on FD 9
2003/09/29 21:49:31| storeDirWriteCleanLogs: Starting...
2003/09/29 21:49:32| Finished. Wrote 253 entries.
2003/09/29 21:49:32| Took 0.1 seconds (1957.6 entries/sec).
2003/09/29 21:49:32| Squid Cache (Version 2.5.STABLE4): Exiting normally.

假如你使用 squid -k interrupt 命令，squid 立即关闭，不用等待完成活动请求。这与在 kill 里发送 INT 信号相同。

5.9 重配置运行中的 squid 进程

在你了解了更多关于 squid 的知识后，你会发现对 squid.conf 文件做了许多改动。为了让新设置生效，你可以关闭和重启 squid,或者在 squid 运行时，重配置它。

重配置运行中的 squid 最好的方法是使用 squid -k reconfigure 命令：

%squid -k reconfigure

当你运行该命令时，HUP 信号被发送到运行中的 squid 进程。然后 squid 读取和解析 squid.conf 文件。假如操作成功，你可以在 cache.log 里看到这些：
2003/09/29 22:02:25| Restarting Squid Cache (version 2.5.STABLE4)...
2003/09/29 22:02:25| FD 12 Closing HTTP connection
2003/09/29 22:02:25| FD 13 Closing ICP connection
2003/09/29 22:02:25| Cache dir '/usr/local/squid/var/cache' size remains unchanged
at 102400 KB
2003/09/29 22:02:25| DNS Socket created on FD 5
2003/09/29 22:02:25| Adding nameserver 10.0.0.1 from /etc/resolv.conf
2003/09/29 22:02:25| Accepting HTTP connections at 0.0.0.0, port 3128, FD 9.
2003/09/29 22:02:25| Accepting ICP messages at 0.0.0.0, port 3130, FD 11.
2003/09/29 22:02:25| WCCP Disabled.
2003/09/29 22:02:25| Loaded Icons.
2003/09/29 22:02:25| Ready to serve requests.

在使用 reconfigure 选项时你须谨慎，因为所做的改变可能会导致致命错误。例如，请注意 squid 关闭和重新打开进来的 HTTP 和 ICP 套接字；假如你将 http_port 改变为 squid 不能打开的端口，它会发生致命错误并退出。

在 squid 运行时，某些指令和和选项不能改变，包括：
+ 删除 cache 目录（cache_dir 指令）
+ 改变 store_log 指令
+ 改变 coss cache_dir 的块大小数值。事实上，无论何时你改变了该值，你必须重新初始化 coss cache_dir。
+ coredump_dir 指令在重配置过程中不被检查。所以，在 squid 已经启动了后，你不能让 squid 改变它的当前目录。

solaris 用户在重配置 squid 过程中可能遇到其他问题。solaris 的 stdio 执行组件里的 fopen()调用要求使用小于 256 的未用文件描述符。FILE 结构以 8 位值存储该文件描述符。正常情况下这不构成问题，因为 squid 使用底层 I/O（例如 open()）来打开 cache 文件。然而，在重配置过程中的某些任务使用 fopen()，这就有可能失败，因为前面的 256 个文件描述符已被分配出去。

5.10 滚动日志文件

除非你在 squid.conf 里禁止，squid 会写大量的日志文件。你必须周期性的滚动日志文件，以阻止它们变得太大。squid 将大量的重要信息写入日志，假如写不进去了，squid 会发生错误并退出。为了合理控制磁盘空间消耗，在 cron 里使用如下命令：
%squid -k rotate

例如，如下任务接口在每天的早上 4 点滚动日志：
0 4 * * * /usr/local/squid/sbin/squid -k rotate

该命令做两件事。首先，它关闭当前打开的日志文件。然后，通过在文件名后加数字扩展名，它重命名 cache.log,store.log,和 access.log。例如，cache.log 变成 cache.log.0,cache.log.0 变成 cache.log.1,如此继续，滚动到 logfile_rotate 选项指定的值。

squid 仅仅保存每个日志文件的最后 logfile_rotate 版本。更老的版本在重命名过程中被删除。假如你想保存更多的拷贝，你需要增加 logfile_rotate 限制，或者编写脚本用于将日志文件移动到其他位置。

请见 13.7 章关于滚动日志的其他信息。

第 6 章. 访问控制

6.1 访问控制元素

ACL 元素是 Squid 的访问控制的基础。这里告诉你如何指定包括 IP 地址，端口号，主机名，和 URL 匹配等变量。每个 ACL 元素有个名字，在编写访问控制规则时需要引用它们。基本的 ACL 元素语法如下：
acl name type value1 value2 ...

例如：
acl Workstations src 10.0.0.0/16

在多数情况下，你能对一个 ACL 元素列举多个值。你也可以有多个 ACL 行使用同一个名字。例如，下列两行配置是等价的：
acl http_ports port 80 8000 8080
acl Http_ports port 80
acl Http_ports port 8000
acl Http_ports port 8080

6.1.1 一些基本的 ACL 类型

Squid 大约有 25 个不同的 ACL 类型，其中的一些有通用基本类型。例如，src 和 dst ACL 使用 IP 地址作为它们的基本类型。为避免冗长，我首先描述基本类型，然后在接下来章节里描述每种 ACL 类型。

6.1.1.1 IP 地址

使用对象：src,dst,myip
squid 在 ACL 里指定 IP 地址时，拥有强有力的语法。你能以子网，地址范围，域名等形式编

写地址。squid 支持标准 IP 地址写法（由"."连接的 4 个小于 256 的数字）和无类域间路由规范。另外，假如你忽略掩码，squid 会自动计算相应的掩码。例如，下例中的每组是相等的：

```
acl Foo src 172.16.44.21/255.255.255.255
acl Foo src 172.16.44.21/32
acl Foo src 172.16.44.21
acl Xyz src 172.16.55.32/255.255.255.248
acl Xyz src 172.16.55.32/28
acl Bar src 172.16.66.0/255.255.255.0
acl Bar src 172.16.66.0/24
acl Bar src 172.16.66.0
```

当你指定掩码时，squid 会检查你的工作。如果你的掩码在 IP 地址的非零位之外，squid 会告警。例如，下列行导致告警：

```
acl Foo src 127.0.0.1/8
aclParseIpData: WARNING: Netmask masks away part of the specified IP in 'Foo'
```

这里的问题是/8 掩码（255.0.0.0）在最后三个字节里都是零值，但是 IP 地址 127.0.0.1 不是这样的。squid 警告你这个问题，以便你消除歧义。正确的写法是：

```
acl Foo src 127.0.0.1/32
or:
acl Foo src 127.0.0.0/8
```

有时候你可能想列举多个相邻子网，在这样的情况下，通过指定地址范围很容易做到。例如：

```
acl Bar src 172.16.10.0-172.16.19.0/24
```

这等价但高效于下面的行：

```
acl Foo src 172.16.10.0/24
acl Foo src 172.16.11.0/24
acl Foo src 172.16.12.0/24
acl Foo src 172.16.13.0/24
acl Foo src 172.16.14.0/24
acl Foo src 172.16.15.0/24
acl Foo src 172.16.16.0/24
acl Foo src 172.16.18.0/24
acl Foo src 172.16.19.0/24
```

注意使用 IP 地址范围，掩码只能取一个。你不能为范围里的地址设置多个不同掩码。

你也能在 IP ACL 里指定主机名，例如：

```
acl Squid dst www.squid-cache.org
```

squid 在启动时，将主机名转换成 IP 地址。一旦启动，squid 不会对主机名的地址发起第二次 DNS 查询。这样，假如在 squid 运行中地址已改变，squid 不会注意到。

假如主机名被解析成多个 IP 地址，squid 将每一个增加到 ACL 里。注意你也可以对主机名使用网络掩码。

在基于地址的 ACL 里使用主机名通常是坏做法。squid 在初始化其他组件之前，先解析配置文件，所以这些 DNS 查询不使用 squid 的非阻塞 IP 缓存接口。代替的，它们使用阻塞机制的 gethostbyname()函数。这样，将 ACL 主机名转换到 IP 地址的过程会延缓 squid 的启动。除非绝对必要，请在 src,dst,和 myip ACL 里避免使用主机名。

squid 以一种叫做 splay tree 的数据结构在内存里存储 IP 地址 ACL（请见 http://www.link.cs.cmu.edu/splay/）。splay tree 有一些有趣的自我调整的特性，其中之一是在查询发生时，列表会自动纠正它自己的位置。当某个匹配元素在列表里发现时，该元素变成新的树根。在该方法中，最近参考的条目会移动到树的顶部，这减少了将来查询的时间。

属于同一 ACL 元素的所有的子网和范围不能重迭。如果有错误，squid 会警告你。例如，如下不被允许：
acl Foo src 1.2.3.0/24
acl Foo src 1.2.3.4/32

它导致 squid 在 cache.log 里打印警告：
WARNING: '1.2.3.4' is a subnetwork of '1.2.3.0/255.255.255.0'
WARNING: because of this '1.2.3.4' is ignored to keep splay tree searching predictable
WARNING: You should probably remove '1.2.3.4' from the ACL named 'Foo'

在该情形下，你需要修正这个问题，可以删除其中一个 ACL 值，或者将它们放置在不同的 ACL 列表中。

6.1.1.2 域名

使用对象：srcdomain,dstdomain,和 cache_host_domain 指令域名简单的就是 DNS 名字或区域。例如，下面是有效的域名：
www.squid-cache.org
squid-cache.org
org

域名 ACL 有点深奥，因为相对于匹配域名和子域有点微妙的差别。当 ACL 域名以"."开头，squid 将它作为通配符，它匹配在该域的任何主机名，甚至域名自身。相反的，如果 ACL 域名不以"."开头，squid 使用精确的字符串比较，主机名同样必须被严格检查。

表 6-1 显示了 squid 的匹配域和主机名的规则。第一列显示了取自 URL 请求的主机名（或者 srcdomain ACL 的客户主机名）。第二列指明是否主机名匹配 lrrr.org。第三列显示是否主机名匹配.lrrr.org ACL。你能看到，唯一的不同在第二个实例里。
Table 6-1. Domain name matching

_____
__
__URL hostname_____Matches ACL lrrr.org? _____Matches ACL .lrrr.org?
__lrrr.org_____Yes_____Yes
__i.am.lrrr.org_____No_____Yes
__iamlrrr.org_____No_____No
_____
__
**说明：为了表现表格形状，“__”仅代表空格分隔符，没有任何实际意义（段誉 注释）。

域名匹配可能让人迷惑，所以请看第二个例子以便你能真正理解它。如下是两个稍微不同的 ACL：
acl A dstdomain foo.com
acl B dstdomain .foo.com

用户对 http://www.foo.com/的请求匹配 ACL B，但不匹配 A。ACL A 要求严格的字符串匹配，然而 ACL B 里领头的点就像通配符。

另外，用户对 http://foo.com/的请求同时匹配 A 和 B。尽管在 URL 主机名里的 foo.com 前面没有字符，但 ACL B 里领头的点仍然导致一个匹配。

squid 使用 splay tree 的数据结构来存储域名 ACL，就像它处理 IP 地址一样。然而，squid 的域名匹配机制给 splay tree 提供了一个有趣的问题。splay tree 技术要求唯一键去匹配任意特定搜索条目。例如，让我们假设搜索条目是 i.am.lrrr.org。该主机名同时匹配.lrrr.org 和.am.lrrr.org。事实上就是两个 ACL 值匹配同一个主机名扰乱了 splay 机制。换句话说，在配置文件里放置如下语句是错误的：
acl Foo dstdomain .lrrr.org .am.lrrr.org

假如你这样做，squid 会产生如下警告信息：
WARNING: '.am.lrrr.org' is a subdomain of '.lrrr.org'
WARNING: because of this '.am.lrrr.org' is ignored to keep splay tree searching predictable
WARNING: You should probably remove '.am.lrrr.org' from the ACL named 'Foo'

在该情况下你应遵循 squid 的建议。删除其中一条相关的域名，以便 squid 明确知道你的意图。注意你能在不同的 ACL 里任意使用这样的域名：
acl Foo dstdomain .lrrr.org
acl Bar dstdomain .am.lrrr.org

这是允许的，因为每个命名 ACL 使用它自己的 splay tree.

6.1.1.3 用户名

使用对象：ident，proxy_auth

该类型的 ACL 被设计成匹配用户名。squid 可能通过 RFC 1413 ident 协议或者通过 HTTP 验证头来获取用户名。用户名必须被严格匹配。例如，bob 不匹配 bobby。squid 也有相关的 ACL 对用户名使用正则表达式匹配（ident_regex 和 proxy_auth_regex）。

你可以使用单词"REQUIRED"作为特殊值去匹配任意用户名。假如 squid 不能查明用户名，ACL 不匹配。当使用基于用户名的访问控制时，squid 通常这样配置。

6.1.1.4 正则表达式

使用对象：srcdom_regex，dstdom_regex，url_regex，urlpath_regex，browser，referer_regex，ident_regex，proxy_auth_regex，req_mime_type，rep_mime_type

大量的 ACL 使用正则表达式来匹配字符串（完整的正则表达式参考，请见 O'Reilly 的 Mastering Regular Expressions 一书）。对 squid 来说，最常使用的正则表达式功能用以匹配字符串的开头或结尾。例如，^字符是特殊元字符，它匹配行或字符串的开头：
^http://

该正则表达式匹配任意以 http://开头的 URL。$也是特殊的元字符，因为它匹配行或字符串的结尾：
.jpg$

实际上，该示例也有些错误，因为.字符也是特殊元字符。它是匹配任意单个字符的通配符。我们实际想要的应该是：
\.jpg$

反斜杠对这个"."进行转义。该正则表达式匹配以.jpg 结尾的任意字符串。假如你不使用^或$字符，正则表达式的行为就象标准子串搜索。它们匹配在字符串里任何位置出现的单词或词组。

对所有的 squid 正则表达式类，你可以使用大小写敏感的选项。匹配是默认大小写敏感的。为了大小写不敏感，在 ACL 类型后面使用-i 选项。例如：
acl Foo url_regex -i ^http://www

6.1.1.5 TCP 端口号

使用对象：port，myport

该类型是相对的。值是个别的端口号或端口范围。回想一下 TCP 端口号是 16 位值，这样它的值必须大于 0 和小于 65536。如下是一些示例：
acl Foo port 123
acl Bar port 1-1024

6.1.1.6 自主系统号
使用对象：src_as，dst_as
Internet 路由器使用自主系统(AS)号来创建路由表。基本上，某个 AS 号指向被同一组织管理的 IP 网络范围。例如，我的 ISP 分配了如下网络块：134.116.0.0/16, 137.41.0.0/16, 206.168.0.0/16,和其他更多。在 Internet 路由表里，这些网络被公布为属于 AS 3404。当路由器转发包时，它们典型的选择经过最少 AS 的路径。假如这些对你不重要，请不必关注它们。AS 基础的 ACL 仅仅被网络 gurus 使用。

如下是基于 AS 的类型如何工作的：当 squid 首先启动时，它发送一条特殊的查询到某个 whois 服务器。查询语句基本是："告诉我哪个 IP 网络属于该 AS 号"。这样的信息被 RADB 收集和管理。一旦 Squid 接受到 IP 网络列表，它相似的将它们作为 IP 基础的 ACL 对待。

基于 AS 的类型仅仅在 ISP 将他们的 RADB 信息保持与日更新时才工作良好。某些 ISP 更新 RADB 比其他人做得更好；而许多根本不更新它。请注意 squid 仅仅在启动或者 reconfigure 时才将 AS 号转换为网络地址。假如 ISP 更新了它的 RADB 接口，除非你重启或者重配置 squid，squid 不会知道这个改变。

另外的情况是，在你的 squid 启动时，RADB 可能不可到达。假如 Squid 不能联系上 RADB 服务器，它从访问控制配置里删除 AS 接口。默认的 whois 服务器是 whois.ra.net，对许多用户来说太遥远了而不可信赖。

6.1.2 ACL 类型

现在我们能把焦点放在 ACL 类型自身上。我在这里按照重要性的降序来列举它们。

6.1.2.1 src

IP 地址在访问控制元素里是最普遍使用的。大部分站点使用 IP 地址来控制客户允许或不允许访问 Squid。src 类型指客户源 IP 地址。也就是说，当 src ACL 出现在访问控制列表里时，squid 将它与发布请求的客户 IP 地址进行比较。

正常情况下你允许来自内网中主机的请求，并阻塞其他的。例如，假如你的单位使用 192.168.0.0 子网，你可以这样指定 ACL：
acl MyNetwork src 192.168.0.0

假如你有许多子网，你能在同一个 acl 行里面列举它们：
acl MyNetwork src 192.168.0.0 10.0.1.0/24 10.0.5.0/24 172.16.0.0/12

squid 有许多其他 ACL 类型用以检查客户地址。srcdomain 类型比较客户的完整可验证域名。它要求反向 DNS 查询，这可能会延缓处理该请求。srcdom_regex ACL 是类似的，但它允许

你使用正则表达式来匹配域名。最后，src_as 类型比较客户的 AS 号。

6.1.2.2 dst

dst 类型指向原始服务器（目标）IP 地址。在某些情况下，你能使用该类型来阻止你的用户访问特定 web 站点。然而，在使用 dst ACL 时你须谨慎。大部分 squid 接受到的请求有原始服务器主机名。例如：
GET http://www.web-cache.com/ HTTP/1.0

这里，www.web-cache.com 是主机名。当访问列表规则包含了 dst 元素时，squid 必须找到该主机名的 IP 地址。假如 squid 的 IP 缓存包含了该主机名的有效接口，这条 ACL 被立即检测。否则，在 DNS 查询忙碌时，squid 会延缓处理该请求。这对某些请求来说会造成延时。

为了避免延时，你该尽可能的使用 dstdomain ACL 类型来代替 dst。

如下是简单的 dst ACL 示例：
acl AdServers dst 1.2.3.0/24

请注意，dst ACL 存在的问题是，你试图允许或拒绝访问的原始服务器可能会改变它的 IP 地址。假如你不关心这样的改变，那就不必麻烦去升级 squid.conf。你可以在 acl 行里放上主机名，但那样会延缓启动速度。假如你的 ACL 需要许多主机名，你也许该预处理配置文件，将主机名转换成 IP 地址。

6.1.2.3 myip

myip 类型指 Squid 的 IP 地址，它被客户连接。当你在 squid 机上运行 netstat -n 时，你见到它们位于本地地址列。大部分 squid 安装不使用该类型。通常所有的客户连接到同一个 IP 地址，所以该 ACL 元素仅仅当系统有多个 IP 地址时才有用。

为了理解 myip 为何有用，考虑某个有两个子网的公司网络。在子网 1 的用户是程序员和工程师。子网 2 包括会计，市场和其他管理部门。这样情况下的 squid 有三个网络接口：一个连接子网 1，一个连接子网 2，第三个连接到外部因特网。

当正确的配置时，所有在子网 1 的用户连接到 squid 位于该子网的 IP 地址，类似的，子网 2 的用户连接到 squid 的第二个 IP 地址。这样你就可以给予子网 1 的技术部员工完全的访问权，然而限制管理部门的员工仅仅能访问工作相关的站点。

ACL 可能如下：
acl Eng myip 172.16.1.5
acl Admin myip 172.16.2.5

然而请注意，使用该机制你必须特别小心，阻止来自某个子网的用户连接 squid 位于另一子网

的 IP 地址。否则，在会计和市场子网的聪明的用户，能够通过技术部子网进行连接，从而绕过你的限制。

6.1.2.4 dstdomain

在某些情况下，你发现基于名字的访问控制非常有用。你可以使用它们去阻塞对某些站点的访问，去控制 squid 如何转发请求，以及让某些响应不可缓存。dstdomain 之所以非常有用，是因为它检查请求 url 里的主机名。

然而首先我想申明如下两行的不同：
acl A dst www.squid-cache.org
acl B dstdomain www.squid-cache.org

A 实际上是 IP 地址 ACL。当 Squid 解析配置文件时，它查询 www.squid-cache.org 的 IP 地址，并将它们存在内存里。它不保存名字。假如在 squid 运行时 IP 地址改变了，squid 会继续使用旧的地址。

然而 dstdomain ACL 以域名形式存储，并非 IP 地址。当 squid 检查 ACL B 时，它对 URL 的主机名部分使用字符串比较功能。在该情形下，它并不真正关心是否 www.squid-cache.org 的 IP 地址改变了。

使用 dstdomain ACL 的主要问题是某些 URL 使用 IP 地址代替主机名。假如你的目标是使用 dstdomain ACL 来阻塞对某些站点的访问，聪明的用户能手工查询站点的 IP 地址，然后将它们放在 URL 里。例如，下面的 2 行 URL 带来同样的页面：
http://www.squid-cache.org/docs/FAQ/
http://206.168.0.9/docs/FAQ/

第一行能被 dstdomain ACL 轻易匹配，但第二行不能。这样，假如你依靠 dstdomain ACL，你也该同样阻塞所有使用 IP 地址代替主机名的请求。请见 6.3.8 章节。

6.1.2.5 srcdomain

srcdomain ACL 也有点麻烦。它要求对每个客户 IP 地址进行所谓的反向 DNS 查询。技术上，squid 请求对该地址的 DNS PTR 记录。DNS 的响应--完整可验证域名(FQDN)--是 squid 匹配 ACL 值的东西。（请参考 O'Reilly's DNS and BIND 找到更多关于 DNS PTR 记录的信息）使用 dst ACL,FQDN 查询会导致延时。请求会被延缓处理直到 FQDN 响应返回。FQDN 响应被缓存下来，所以 srcdomain 查询通常仅在客户首次请求时延时。

不幸的是，srcdomain 查询有时不能工作。许多组织并没有保持他们的反向查询数据库与日更新。假如某地址没有 PTR 记录，ACL 检查失败。在该情形下，请求可能会延时非常长时间（例如 2 分钟）直到 DNS 查询超时。假如你使用 srcdomain ACL，请确认你自己的 DNS in-addr.arpa 区域配置正确并且在工作中。假如这样，你可以使用如下的 ACL：

acl LocalHosts srcdomain .users.example.com

6.1.2.6 port

你很可能想使用 port ACL 来限制对某些原始服务器端口号的访问。就像我即将讲到的，squid 其实不连接到某些服务，例如 email 和 IRC 服务。port ACL 允许你定义单独的端口或端口范围。例如：
acl HTTPports port 80 8000-8010 8080

HTTP 在设计上与其他协议类似，例如 SMTP。这意味着聪明的用户通过转发 email 消息到 SMTP 服务器能欺骗 squid。Email 转发是垃圾邮件的主要原因之一，我们必须处理它们。历史上，垃圾邮件有真正的邮件服务器。然而近来，越来越多的垃圾邮件制造者使用开放 HTTP 代理来隐藏他们的踪迹。你肯定不想 Squid 被当成垃圾邮件转发器。假如是这样，你的 IP 地址很可能被许多邮件转发黑名单冻结（MAPS,ORDB,spamhaus 等）。除 email 之外，还有其他许多 TCP/IP 服务是 squid 不与其通信的。这些包括 IRC,Telnet,POP,和 NNTP。你的针对端口的策略必须被配置成拒绝已知危险端口，并允许剩下的；或者允许已知安全端口，并拒绝剩下的。

我的态度比较保守，仅仅允许安全的端口。默认的 squid.conf 包含了下面的安全端口 ACL：
acl Safe_ports port 80 # http
acl Safe_ports port 21 # ftp
acl Safe_ports port 443 563 # https, snews
acl Safe_ports port 70 # gopher
acl Safe_ports port 210 # wais
acl Safe_ports port 1025-65535 # unregistered ports
acl Safe_ports port 280 # http-mgmt
acl Safe_ports port 488 # gss-http
acl Safe_ports port 591 # filemaker
acl Safe_ports port 777 # multiling http
http_access deny !Safe_ports

这是个较明智的配置。它允许用户连接到任何非特权端口（1025－65535），但仅仅指定的特权端口可以被连接。假如你的用户试图访问某个 URL 如下：http://www.lrrr.org:123/，squid 会返回访问拒绝错误消息。在某些情形下，为了让你的用户满意，你可能需要增加另外的端口号 。

宽松的做法是，拒绝对特别危险的端口的访问。Squid FAQ 包括了如下示例：
acl Dangerous_ports 7 9 19 22 23 25 53 109 110 119
http_access deny Dangerous_ports

使用 Dangerous_ports 的弊端是 squid 对几乎每个请求都要搜索整个列表。这对 CPU 造成了额外的负担。大多数情况下，99%到达 squid 的请求是对 80 端口的，它不出现在危险端口列表里。所有请求对该表的搜索不会导致匹配。当然，整数比较是快速的操作，不会显然影响性能。

（译者注：这里的意思是，两者都要对列表进行搜索和匹配。在第一种情况下，它搜索安全端口列表并匹配 80，显然第一个元素就匹配成功了。而第二种情况中，会搜索危险端口列表并试图匹配 80，当然危险端口不会包括 80，所以每次对 80 的请求都要搜索完整个列表，这样就会影响性能。）

6.1.2.7 myport

squid 也有 myport ACL。port ACL 指向原始服务器的端口号，myport 指向 squid 自己的端口号，用以接受客户请求。假如你在 http_port 指令里指定不止一个端口号，那么 squid 就可以在不同的端口上侦听。

假如你将 squid 作为站点 HTTP 加速器和用户代理服务器，那么 myport ACL 特别有用。你可以在 80 上接受加速请求，在 3128 上接受代理请求。你可能想让所有人访问加速器，但仅仅你自己的用户能以代理形式访问 squid。你的 ACL 可能如下：

```
acl AccelPort myport 80
acl ProxyPort myport 3128
acl MyNet src 172.16.0.0/22
http_access allow AccelPort # anyone
http_access allow ProxyPort MyNet # only my users
http_access deny ProxyPort # deny others
```

6.1.2.8 method

method ACL 指 HTTP 请求方法。GET 是典型的最常用方法，接下来是 POST,PUT，和其他 。下例说明如何使用 method ACL：

```
acl Uploads method PUT POST
```

Squid 知道下列标准 HTTP 方法：GET, POST, PUT, HEAD, CONNECT, TRACE,OPTIONS 和 DELETE。另外，squid 了解下列来自 WEBDAV 规范，RFC 2518 的方法：PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK。某些 Microsoft 产品使用非标准的 WEBDAV 方法，所以 squid 也了解它们：BMOVE, BDELETE, BPROPFIND。最后，你可以在 extension_methods 指令里配置 squid 去理解其他的请求方法。请见附录 A。

注意 CONNECT 方法非常特殊。它是用于通过 HTTP 代理来封装某种请求的方法（请见 RFC 2817:Upgrading to TLS Within HTTP/1.1）。在处理 CONNECT 方法和远程服务器的端口号时应特别谨慎。就像前面章节讲过的一样，你不希望 squid 连接到某些远程服务。你该限制 CONNECT 方法仅仅能连接到 HTTPS/SSL 或 NNTPS 端口（443 和 563）。默认的 squid.conf 这样做：

```
acl CONNECT method CONNECT
acl SSL_ports 443 563
http_access allow CONNECT SSL_ports
```

http_access deny CONNECT

在该配置里，squid 仅仅允许加密请求到端口 443（HTTPS/SSL）和 563（NNTPS）。CONNECT 方法对其他端口的请求都被拒绝。

PURGE 是另一个特殊的请求方法。它是 Squid 的专有方法，没有在任何 RFC 里定义。它让管理员能强制删除缓存对象。既然该方法有些危险，squid 默认拒绝 PURGE 请求，除非你定义了 ACL 引用了该方法。否则，任何能访问 cache 者也许能够删除任意缓存对象。我推荐仅仅允许来自 localhost 的 PURGE:

```
acl Purge method PURGE
acl Localhost src 127.0.0.1
http_access allow Purge Localhost
http_access deny Purge
```

关于从 squid 的缓存里删除对象，请见 7.6 章。

6.1.2.9 proto

该类型指 URI 访问（或传输）协议。如下是有效值：http, https (same as HTTP/TLS), ftp,gopher, urn, whois, 和 cache_object。也就是说，这些是被 squid 支持的 URL 机制名字。例如，假如你想拒绝所有的 FTP 请求，你可以使用下列指令：

```
acl FTP proto FTP
http_access deny FTP
```

cache_object 机制是 squid 的特性。它用于访问 squid 的缓存管理接口，我将在 14.2 章讨论它。不幸的是，它并非好名字，可能会被改变。

默认的 squid.conf 文件有许多行限制缓存管理访问：

```
acl Manager proto cache_object
acl Localhost src 127.0.0.1
http_access allow Manager Localhost
http_access deny Manager
```

这些配置行仅允许来自本机地址的缓存管理请求，所有其他的缓存管理请求被拒绝。这意味着在 squid 机器上有帐号的人，能访问到潜在的敏感缓存管理信息。你也许想修改缓存管理访问控制，或对某些页面使用密码保护。我将在 14.2.2 章里谈论到。

6.1.2.10 time

time ACL 允许你控制基于时间的访问，时间为每天中的具体时间，和每周中的每天。日期以单字母来表示，见如下表。时间以 24 小时制来表示。开始时间必须小于结束时间，这样在编写跨越 0 点的 time ACL 时可能有点麻烦。

Code____Day

S_____Sunday
M_____Monday
T_____Tuesday
W_____Wednesday
H_____Thursday
F_____Friday
A_____Saturday
D_____All weekdays (M-F)

日期和时间由 localtime()函数来产生。请确认你的计算机位于正确的时区，你也该让你的时钟与标准时间同步。

为了编写 time ACL 来匹配你的工作时间，你可以这样写：
acl Working_hours MTWHF 08:00-17:00
or:
acl Working_hours D 08:00-17:00

让我们看一个麻烦的例子。也许你是某个 ISP，在下午 8 点到早上 4 点这段不忙的时间内放松访问。既然该时间跨越子夜，你不能编写"20:00-04:00"。代替的，你需要把它们分成两个 ACL 来写，或者使用否定机制来定义非忙时。例如：
acl Offpeak1 20:00-23:59
acl Offpeak2 00:00-04:00
http_access allow Offpeak1 ...
http_access allow Offpeak2 ...

另外，你可以这样写：
acl Peak 04:00-20:00
http_access allow !Peak ...

尽管 squid 允许，你也不应该在同一个 time ACL 里放置多个日期和时间范围列表。对这些 ACL 的解析不一定是你想象的那样。例如，假如你输入：
acl Blah time M 08:00-10:00 W 09:00-11:00

实际能做到的是：
acl Blah time MW 09:00-11:00

解析仅仅使用最后一个时间范围。正确的写法是，将它们写进两行：
acl Blah time M 08:00-10:00
acl Blah time W 09:00-11:00

6.1.2.11 ident

ident ACL 匹配被 ident 协议返回的用户名。这是个简单的协议，文档是 RFC 1413。它工作过程如下：

1.用户代理（客户端）对 squid 建立 TCP 连接。

2.squid 连接到客户系统的 ident 端口（113）。

3.squid 发送一个包括两个 TCP 端口号的行。squid 端的端口号可能是 3128（或者你在 squid.conf 里配置的端口号），客户端的端口号是随机的。

4.客户端的 ident 服务器返回打开第一个连接的进程的用户名。

5.squid 记录下用户名用于访问控制目的，并且记录到 access.log。

当 squid 遇到对特殊请求的 ident ACL 时，该请求被延时，直到 ident 查询完成。这样，ident ACL 可以对你的用户请求造成延时。

我们推荐仅仅在本地局域网中，并且大部分客户工作站运行 ident 服务时，才使用 ident ACL。假如 squid 和客户工作站连在一个局域网里，ident ACL 工作良好。跨广域网使用 ident 难以成功。

ident 协议并非很安全。恶意的用户能替换他们的正常 ident 服务为假冒服务，并返回任意的他们选择的用户名。例如，假如我知道从 administrator 用户的连接总是被允许，那么我可以写个简单的程序，在回答每个 ident 请求时都返回这个用户名。

你可以使用 ident ACL 拦截 cache（请见第 9 章）。当 squid 被配置成拦截 cache 时，操作系统假设它自己是原始服务器。这意味着用于拦截 TCP 连接的本地 socket 地址有原始服务器的 IP 地址。假如你在 squid 上运行 netstat -n 时，你可以看到大量的外部 IP 地址出现在本地地址栏里。当 squid 发起一个 ident 查询时，它创建一个新的 TCP 套接字，并绑定本地终点到同一个 IP 地址上，作为客户 TCP 连接的本地终点。既然本地地址并非真正是本地的（它可能与原始服务器 IP 地址相距遥远），bind()系统调用失败。squid 将这个作为失败的 ident 查询来处理。

注意 squid 也有个特性，对客户端执行懒惰 ident 查询。在该情形下，在等待 ident 查询时，请求不会延时。在 HTTP 请求完成时，squid 记录 ident 信息，假如它可用。你能使用 ident_lookup_access 指令来激活该特性，我将在本章后面讨论。

6.1.2.12 proxy_auth

squid 有一套有力的，在某种程度上有点混乱的特性，用以支持 HTTP 代理验证功能。使用代理验证，客户的包括头部的 http 请求包含了验证信用选项。通常，这简单的是用户名和密码。

squid 解密信用选项，并调用外部验证程序以发现该信用选项是否有效。

squid 当前支持三种技术以接受用户验证：HTTP 基本协议，数字认证协议，和 NTLM。基本认证已经发展了相当长时间。按今天的标准，它是非常不安全的技术。用户名和密码以明文同时发送。数字认证更安全，但也更复杂。基本和数字认证在 RFC 2617 文档里被描述。NTLM 也比基本认证更安全。然而，它是 Microsoft 发展的专有协议。少数 squid 开发者已经基本完成了对它的反向工程。

为了使用代理验证，你必须配置 squid 使用大量的外部辅助程序。squid 源代码里包含了一些程序，用于对许多标准数据库包括 LDAP,NTLM,NCSA 类型的密码文件，和标准 Unix 密码数据库进行认证。auth_param 指令控制对所有辅助程序的配置。我将在 12 章里讨论这些细节。

auth_param 指令和 proxy_auth ACL 是少数在配置文件里顺序重要的实例。你必须在 proxy_auth ACL 之前定义至少一个验证辅助程序（使用 auth_param）。假如你没有这样做，squid 打印出错误消息，并且忽略 proxy_auth ACL。这并非致命错误，所以 squid 可以启动，但所有你的用户的请求可能被拒绝。

proxy_auth ACL 取用户名作为值。然而，大部分安装里简单的使用特殊值 REQUIRED：
```
auth_param ...
acl Auth1 proxy_auth REQUIRED
```

在该情况中，任何具有有效信用选项的请求会匹配该 ACL。假如你需要细化控制，你可以指定独立的用户名：
```
auth_param ...
acl Auth1 proxy_auth allan bob charlie
acl Auth2 proxy_auth dave eric frank
```

代理验证不支持 HTTP 拦截，因为用户代理不知道它在与代理服务器，而非原始服务器通信。用户代理不知道在请求里发送 Proxy-Authorization 头部。见 9.2 章更多细节。

6.1.2.13 src_as

该类型检查客户源 IP 地址所属的具体 AS 号（见 6.1.1.6 关于 squid 如何将 AS 号映射到 IP 地址的信息）。作为示例，我们虚构某 ISP 使用 AS 64222 并且通告使用 10.0.0.0/8,172.16.0.0/12,192.168.0.0/16 网络。你可以编写这样的 ACL，它允许来自该 ISP 地址空间的任何主机请求：
```
acl TheISP src 10.0.0.0/8
acl TheISP src 172.16.0.0/12
acl TheISP src 192.168.0.0/16
http_access allow TheISP
```

当然，你还可以这样写：
```
acl TheISP src_as 64222
```

```
http_access allow TheISP
```

第二种写法不但更短，而且假如 ISP 增加了新的网络，你不必更新 ACL 配置。

6.1.2.14 dst_as

dst_as ACL 经常与 cache_peer_access 指令一起使用。在该方法中，squid 使用与 IP 路由一致的方式转发 cache 丢失。考虑某 ISP，它比其他 ISP 更频繁的更换路由。每个 ISP 处理他们自己的 cache 代理，这些代理能转发请求到其他代理。理论上，ISP A 将 ISP B 网络里主机的 cache 丢失转发到 ISP B 的 cache 代理。使用 AS ACL 和 cache_peer_access 指令容易做到这点：

```
acl ISP-B-AS dst_as 64222
acl ISP-C-AS dst_as 64333
cache_peer proxy.isp-b.net parent 3128 3130
cache_peer proxy.isp-c.net parent 3128 3130
cache_peer_access proxy.isb-b.net allow ISP-B-AS
cache_peer_access proxy.isb-c.net allow ISP-C-AS
```

我将在第 10 章里讨论更多关于 cache 协作。

6.1.2.15 snmp_community

snmp_community ACL 对 SNMP 查询才有意义，后者被 snmp_access 指令控制。例如，你可以这样写：

```
acl OurCommunityName snmp_community hIgHsEcUrItY
acl All src 0/0
snmp_access allow OurCommunityName
snmp_access deny All
```

在该情况中，假如 community 名字设置为 hIgHsEcUrItY，SNMP 查询才被允许。

6.1.2.16 maxconn

maxconn ACL 指来自客户 IP 地址的大量同时连接。某些 squid 管理员发现这是个有用的方法，用以阻止用户滥用代理或者消耗过多资源。

maxconn ACL 在请求超过指定的数量时，会匹配这个请求。因为这个理由，你应该仅仅在 deny 规则里使用 maxconn。考虑如下例子：

```
acl OverConnLimit maxconn 4
http_access deny OverConnLimit
```

在该情况中，squid 允许来自每个 IP 地址的同时连接数最大为 4 个。当某个客户发起第五个连接时，OverConnLimit ACL 被匹配，http_access 规则拒绝该请求。

6.1.2.17 arp

arp ACL 用于检测 cache 客户端的 MAC 地址（以太网卡的物理地址）。地址解析协议（ARP）是主机查找对应于 IP 地址的 MAC 地址的方法。某些大学学生发现，在 Microsoft Windows 下，他们可以改变系统的 IP 地址到任意值，然后欺骗 squid 的基于地址的控制。这时 arp 功能就派上用场了，聪明的系统管理员会配置 squid 检查客户的以太网地址。

不幸的是，该特性使用非移植性代码。假如你运行 Solaris 或 Linux，你能使用 arp ACL。其他系统不行。当你运行./configure 时增加--enable-arp-acl 选项，就可以激活该功能。

arp ACL 有另一个重要限制。ARP 是数据链路层协议，假如客户主机和 squid 在同一子网，它才能工作。你不容易发现不同子网主机的 MAC 地址。假如在 squid 和你的用户之间有路由器存在，你可能不能使用 arp ACL。

现在你知道何时去使用它们，让我们看看 arp ACL 实际上是怎样的。它的值是以太网地址，当使用 ifconfig 和 arp 时你能看到以太网地址。例如：

```
acl WinBoxes arp 00:00:21:55:ed:22
acl WinBoxes arp 00:00:21:ff:55:38
```

6.1.2.18 srcdom_regex

srcdom_regex ACL 允许你使用正则表达式匹配客户域名。这与 srcdomain ACL 相似，它使用改进的的子串匹配。相同的限制是：某些客户地址不能反向解析到域名。作为示例，下面的 ACL 匹配以 dhcp 开头的主机名：

```
acl DHCPUser srcdom_regex -i ^dhcp
```

因为领头的^ 符号， 该 ACL 匹配主机名 dhcp12.example.com ， 但不匹配 host12.dhcp.example.com。

6.1.2.19 dstdom_regex

dstdom_regex ACL 也与 dstdomain 相似。下面的例子匹配以 www 开头的主机名：

```
acl WebSite dstdom_regex -i ^www\.
```

如下是另一个有用的正则表达式，用以匹配在 URL 主机名里出现的 IP 地址：

```
acl IPaddr dstdom_regex [0-9]$
```

这样可以工作，因为 squid 要求 URL 主机名完全可验证。既然全局顶级域名中没有以数字结

尾的，该 ACL 仅仅匹配 IP 地址，它以数字结尾。

6.1.2.20 url_regex

url_regex ACL 用于匹配请求 URL 的任何部分，包括传输协议和原始服务器主机名。例如，如下 ACL 匹配从 FTP 服务器的 MP3 文件请求：
acl FTPMP3 url_regex -i ^ftp://.*\.mp3$

6.1.2.21 urlpath_regex

urlpath_regex 与 url_regex 非常相似，不过传输协议和主机名不包含在匹配条件里。这让某些类型的检测非常容易。例如，假设你必须拒绝 URL 里的"sex"，但仍允许在主机名里含有"sex"的请求，那么这样做：
acl Sex urlpath_regex sex

另一个例子，假如你想特殊处理 cgi-bin 请求，你能这样捕获它们：
acl CGI1 urlpath_regex ^/cgi-bin

当然，CGI 程序并非总在/cgi-bin/目录下，这样你应该编写其他的 ACL 来捕获它们。

6.1.2.22 browser

大部分 HTTP 请求包含了 User-Agent 头部。该头部的值典型如下：
Mozilla/4.51 [en] (X11; I; Linux 2.2.5-15 i686)

browser ACL 对 user-agent 头执行正则表达式匹配。例如，拒绝不是来自 Mozilla 浏览器的请求，可以这样写：
acl Mozilla browser Mozilla
http_access deny !Mozilla

在使用 browser ACL 之前，请确认你完全理解 cache 接受到的 User-Agent 字符串。某些 user-agent 与它们的来源相关。甚至 squid 可以重写它转发的请求的 User-Agent 头部。某些浏览器例如 Opera 和 KDE 的 Konqueror，用户可以对不同的原始服务器发送不同的 user-agent 字串，或者干脆忽略它们。

6.1.2.23 req_mime_type

req_mime_type ACL 指客户 HTTP 请求里的 Content-Type 头部。该类型头部通常仅仅出现在请求消息主体里。POST 和 PUT 请求可能包含该头部，但 GET 从不。你能使用该类型 ACL 来检测某些文件上传，和某些类型的 HTTP 隧道请求。

req_mime_type ACL 值是正则表达式。你可以这样编写 ACL 去捕获音频文件类型：
acl AuidoFileUploads req_mime_type -i ^audio/

6.1.2.24 rep_mime_type

该类型 ACL 指原始服务器的 HTTP 响应里的 Content-Type 头部。它仅在使用 http_reply_access 规则时才有用。所有的其他访问控制形式是基于客户端请求的。该 ACL 基于服务器响应。

假如你想使用 squid 阻塞 Java 代码，你可以这样写：
acl JavaDownload rep_mime_type application/x-java
http_reply_access deny JavaDownload

6.1.2.25 ident_regex

在本节早些时讲过 ident ACL。ident_regex 允许你使用正则表达式，代替严格的字符串匹配，这些匹配是对 ident 协议返回的用户名进行。例如，如下 ACL 匹配包含数字的用户名：
acl NumberInName ident_regex [0-9]

6.1.2.26 proxy_auth_regex

该 ACL 允许对代理认证用户名使用正则表达式。例如， 如下 ACL 匹配 admin,administrator 和 administrators:
acl Admins proxy_auth_regex -i ^admin

6.1.3 外部 ACL

Squid 2.5 版本介绍了一个新特性：外部 ACL。你可以指示 squid 发送某些信息片断到外部进程，然后外部的辅助程序告诉 squid，数据匹配或不匹配。

squid 附带着大量的外部 ACL 辅助程序；大部分用于确定命名用户是不是某个特殊组的成员。请见 12.5 章关于这些程序的描述，以及关于如何编写你自己的程序的信息。现在，我解释如何定义和使用外部 ACL 类型。

external_acl_type 指令定义新的外部 ACL 类型。如下是通用语法：
external_acl_type type-name [options] format helper-command

type-name 是用户定义的字串。你也可以在 acl 行里引用它。

Squid 当前支持如下选项(options)：

ttl=n
时间数量，单位是秒，用以缓存匹配值的时间长短。默认是 3600 秒，或 1 小时。

negative_ttl=n
时间数量，单位是秒，用以缓存不匹配值的时间长短。默认是 3600 秒，或 1 小时。

concurrency=n
衍生的辅助程序的数量，默认是 5。

cache=n
缓存结果的最大数量。默认是 0，即不限制 cache 大小。

格式是以%字符开始的一个或多个关键字。squid 当前支持如下格式：

%LOGIN
从代理验证信用选项里获取的用户名。

%IDENT
从 RFC 1413 ident 获取的用户名。

%SRC
客户端 IP 地址。

%DST
原始服务器 IP 地址。

%PROTO
传输协议（例如 HTTP,FTP 等）

%PORT
原始服务器的 TCP 端口。

%METHOD
HTTP 请求方法。

%{Header}
HTTP 请求头部的值；例如，%{User-Agent}导致 squid 发送这样的字串到验证器："Mozilla/4.0 (compatible; MSIE 6.0; Win32)"

%{Hdr:member}
选择某些数量的基于列表的 HTTP 头部，例如 Caceh-Control；例如，给出如下 HTTP 头部：X-Some-Header: foo=xyzzy, bar=plugh, foo=zoinks

对%{X-Some-Header:foo}的取值，squid 发送这样的字串到外部 ACL 进程：
foo=xyzzy, foo=zoinks

%{Hdr:;member}
与%{Hdr:member}相同，除了";"是列表分隔符外。你能使用任何非字母数字的字符作为分隔符。

辅助命令是 squid 为辅助程序衍生的命令。你也可以在这里包含命令参数。例如，整条命令可能类似如此：
/usr/local/squid/libexec/my-acl-prog.pl -X -5 /usr/local/squid/etc/datafile

将这些放在一个长行里。squid 不支持如下通过反斜杠分隔长行的技术，所以请记住所有这些必须放在单行里：
external_acl_type MyAclType cache=100 %LOGIN %{User-Agent} \
/usr/local/squid/libexec/my-acl-prog.pl -X -5 \
/usr/local/squid/share/usernames \
/usr/local/squid/share/useragents

现在你知道如何定义外部 ACL，下一步是编写引用它的 acl 行。这相对容易，语法如下：
acl acl-name external type-name [args ...]

如下是个简单示例：
acl MyAcl external MyAclType

squid 接受在 type-name 后面的任意数量的参数。这些在每个请求里被发送到辅助程序。请见 12.5.3 章，我描述了 unix_group 辅助程序，作为该功能的示例。

6.1.4 处理长 ACL 列表

ACL 列表某些时候非常长。这样的列表在 squid.conf 文件里难以维护。你也可能想从其他资源里自动产生 squid ACL 列表。在如此情况下，你可以从外部文件里包含 ACL 列表。语法如下：
acl name "filename"

这里的双引号指示 squid 打开 filename，并且将它里面的内容分配给 ACL。例如，如下的 ACL 太长了：
acl Foo BadClients 1.2.3.4 1.2.3.5 1.2.3.6 1.2.3.7 1.2.3.9 ...

你可以这样做：
acl Foo BadClients "/usr/local/squid/etc/BadClients"

将 IP 地址放在 BadClients 文件里：

1.2.3.4
1.2.3.5
1.2.3.6
1.2.3.7
1.2.3.9
...

文件可以包含以#开头的注释。注意在该文件里的每个 IP 地址必须是一个单独的行。acl 行里的任何地方，以空格来分隔值，新行是包含 ACL 值的文件的分界。

6.1.5 Squid 如何匹配访问控制元素

理解 squid 如何搜索 ACL 元素去匹配是很重要的。当 ACL 元素有多个值时，任何单个值能导致匹配。换句话说，squid 在检查 ACL 元素值时使用 OR 逻辑。当 squid 找到第一个值匹配时，它停止搜索。这意味着把最可能匹配的值放在列表开头处，能减少延时。

让我们看一个特殊的例子，考虑如下 ACL 定义：
acl Simpsons ident Maggie Lisa Bart Marge Homer

当 squid 在访问列表里遇到 Simpsons ACL 时，它执行 ident 查询。让我们看一下，当用户 ident 服务返回 Marge 时，会发生什么呢？squid 的 ACL 代码在成功匹配 Marge 前，会先后将这个值与 Maggie,Lisa,和 Bart 对比。当搜索完成时，我们认为 Simpsons ACL 匹配了这个请求。

实际上，这有点欺骗。ident ACL 值并非存储在无序列表里。它们存储在 splay tree 中。这意味着，在非匹配事件中，squid 不会搜索完所有的名字。对一个 splay tree 搜索 N 个条目需要记录 N 个比较。许多其他的 ACL 类型也使用 splay tree。然而，基于正则表达式的类型不使用。

既然正则表达式不能这样存储，它们以链表形式存储。这使得在大链表里它们特别低效，特别是不匹配链表里任何正则表达式的请求。为了改进这个形式，当匹配发生时，squid 将正则表达式移到列表的顶部。实际上，因为 ACL 匹配代码的天然特性，squid 将匹配的条目移到列表的第二个位置。这样，普通的匹配值自然移到 ACL 列表的顶部，这样会减少比较数量。

让我们看另一个简单示例：
acl Schmever port 80-90 101 103 107 1 2 3 9999

该 ACL 匹配到原始服务器 80-90 端口，和其他独立端口的请求。对 80 端口的请求，squid 通过查看第一个值就匹配了该 ACL。对 9999 端口，其他每个值都先被检查。对某个不在列表里的端口，squid 要检查所有值才宣布它不匹配。就像我已经讲过的，将最常用的值放在第一位能优化 ACL 匹配。

6.2 访问控制规则

前面提过，ACL 元素是建立访问控制的第一步。第二步是访问控制规则，用来允许或拒绝某些动作。在早先的例子里，你已见过 http_access 规则。squid 有大量其他的访问控制列表：

http_access
这是最重要的访问控制列表。它决定哪些客户 HTTP 请求被允许，和哪些被拒绝。假如 http_access 配置错误，squid cache 容易遭受攻击或被不当利用。

http_reply_access
http_reply_access 与 http_access 类似。不同之处是前者在 squid 接受到来自原始服务器或上级代理的响应时，才会被检测。大部分访问控制基于客户请求的方式，对这些使用 http_access 就够了。然而，某些人喜欢基于响应内容类型来允许或拒绝请求。更多信息请见 6.3.9 章。

icp_access
假如你的 squid 被配置来服务 ICP 响应（见 10.6 章），那么该使用 icp_access 列表。大部分情况下，你该仅仅允许来自邻居 cache 的 ICP 请求。

no_cache
你能使用 no_cache 访问列表来指示 squid，它不必存储某些响应（在磁盘或内存里）。该列表典型的与 dst,dstdomain,url_regex ACL 结合使用。
对 no_cache 使用"否"条件，这样的双重否定会导致某些混乱。被 no_cache 列表拒绝的请求不被缓存。换句话说，no_cache deny...是让目标不被缓存。见 6.3.10 章的示例。

miss_access
miss_access 列表主要用于 squid 的邻居 cache。它决定 squid 怎样处理 cache 丢失的请求。如果 squid 使用集群技术，那么该功能必需。见 6.3.7 的示例。

redirector_access
该访问列表决定哪个请求被发送到重定向进程（见 11 章）。默认情况下，假如你使用重定向器，那么所有的请求都通过重定向器。你可以使用 redirector_access 列表来阻止某些请求被重写。这点特别有用，因为这样的访问列表，使重定向器相对于访问控制系统，接受的请求信息要少一些。

ident_lookup_access
ident_lookup_access 列表与 redirector_access 类似。它允许你对某些请求执行懒惰 ident 查询。squid 默认不发布 ident 查询。假如请求被 ident_lookup_access 规则（或 ident ACL）允许，那么 squid 才会进行 ident 查询。

always_direct
该访问列表影响 squid 怎样处理与邻居 cache 转发 cache 丢失。通常 squid 试图转发 cache 丢失到父 cache，和/或 squid 使用 ICP 来查找临近 cache 响应。然而，当请求匹配 always_direct 规则时，squid 直接转发请求到原始服务器。

使用该规则，对"allow"规则的匹配导致 squid 直接转发请求，见 10.4.4 章的更多细节和示例。

never_direct
never_direct 与 always_direct 相反。匹配该列表的 cache 丢失请求必须发送到邻居 cache。这点对在防火墙之后的代理特别有用。
使用该列表，对"allow"规则的匹配导致 squid 转发请求到邻居 cache。见 10.4.3 章的更多细节和示例。

snmp_access
该访问列表应用到发送给 squid 的 SNMP 端口的查询。你能配合该列表使用的 ACL 是 snmp_community 和 src。假如你确实想使用它，那也能使用 srcdomain,srcdom_regex 和 src_as。见 14.3 章的示例。

broken_posts
该访问列表影响 squid 处理某些 POST 请求的方法。某些老的用户代理在请求主体的结尾处发送一个特别的回车换行符。那就是说，消息主体比 content-length 头部指示的长度要多 2 个字节。更糟糕的是，某些老的 HTTP 服务器实际上依赖于这种不正确的行为。当请求匹配该访问列表时，squid 模拟这种客户端并且发送特殊的回车换行符。
Squid 有大量的使用 ACL 元素的其他配置指令。它们中的某些过去是全局配置，后被修改来使用 ACL 以提供更灵活的控制。

cache_peer_access
该访问列表控制发送到邻居 cache 的 HTTP 请求和 ICP/HTCP 查询。见 10.4.1 章的更多信息和示例。

reply_body_max_size
该访问列表限制对 HTTP 响应主体的最大可接受 size。见附录 A 的更多信息。

delay_access
该访问规则列表控制是否延时池被应用到某个请求的 cache 丢失响应。见附录 C。

tcp_outgoing_address
该访问列表绑定服务端 TCP 连接到指定的本地 IP 地址。见附录 A。

tcp_outgoing_tos
该访问列表能设置到原始服务器和邻居 cache 的 TCP 连接的不同 TOS/Diffserv 值，见附录 A。

header_access
使用该指令，你能配置 squid 从它转发的请求里删除某些 HTTP 头部。例如，你也许想 Squid 过滤掉发送到某些原始服务器的请求里的 Cookie 头部。见附录 A。

header_replace
该指令允许你替换，而不是删除，HTTP 头部的内容。例如，你能设置 user-agent 头部为假

值，满足某些原始服务器的要求，但仍保护你的隐私。见附录 A。

6.2.1 访问规则语法

访问控制规则的语法如下：
access_list allow|deny [!]ACLname ...

例如：
http_access allow MyClients
http_access deny !Safe_Ports
http_access allow GameSites AfterHours

当读取配置文件时，squid 仅仅扫描一遍访问控制行。这样，在访问列表里引用 ACL 元素之前，你必须在 acl 行里定义它们。甚至，访问列表规则的顺序也非常重要。你以怎样的顺序编写访问列表，那么 squid 就按怎样的顺序来检查它们。将最常用的 ACL 放在列表的开始位置，可以减少 squid 的 CPU 负载。

对大部分访问列表，deny 和 allow 的意义明显。然而，它们中的某些，却并非如此含义清楚。请谨慎的编写 always_direct,never_direct,和 no_cache 规则。在 always_direct 中，allow 规则意味着匹配的请求直接转发到原始服务器。always_direct deny 规则意味着匹配的请求不强迫发送到原始服务器，但假如邻居 cache 不可到达，那可能还是会这么做。no_cache 规则也有点麻烦。这里，你必须对不必被 cache 的请求使用 deny。

6.2.2 Squid 如何匹配访问规则

回想一下 squid 在搜索 ACL 元素时使用的“或”逻辑。在 acl 里的任何单值都可以导致匹配。

然而，访问规则恰好相反。对 http_access 和其他规则设置，squid 使用“与”逻辑。考虑如下示例：
access_list allow ACL1 ACL2 ACL3

对该匹配规则来说，请求必须匹配 ACL1,ACL2,ACL3 中的任何一个。假如这些 ACL 中的任何一个不匹配请求，squid 停止搜索该规则，并继续处理下一条。对某个规则来说，将最少匹配的 ACL 放在首位，能使效率最佳。考虑如下示例：
acl A method http
acl B port 8080
http_access deny A B

该 http_access 规则有点低效，因为 A ACL 看起来比 B ACL 更容易匹配。反转顺序应该更好，以便 squid 仅仅检查一个 ACL，而不是两个：
http_access deny B A

人们易犯的典型错误是编写永不正确的规则。例如：

```
acl A src 1.2.3.4
acl B src 5.6.7.8
http_access allow A B
```

该规则永不正确，因为某个源 IP 地址不可能同时等同于 1.2.3.4 和 5.6.7.8。这条规则的真正意图是：

```
acl A src 1.2.3.4 5.6.7.8
http_access allow A
```

对某个 ACL 值的匹配算法是，squid 在访问列表里找到匹配规则时，搜索终止。假如没有访问规则导致匹配，默认动作是列表里最后一条规则的取反。例如，考虑如下简单访问配置：

```
acl Bob ident bob
http_access allow Bob
```

假如用户 Mary 发起请求，她会被拒绝。列表里最后的（唯一的）规则是 allow 规则，它不匹配用户名 mary。这样，默认的动作是 allow 的取反，故请求被拒绝。类似的，假如最后的规则是 deny 规则，默认动作是允许请求。在访问列表的最后加上一条，明确允许或拒绝所有请求，是好的实际做法。为清楚起见，以前的示例应该如此写：

```
acl All src 0/0
acl Bob ident bob
http_access allow Bob
http_access deny All
```

src 0/0 ACL 表示匹配每一个和任意类型的请求。

6.2.3 访问列表风格

squid 的访问控制语法非常强大。大多数情况下，你可以使用两种或多种方法来完成同样的事。通常，你该将更具体的和受限制的访问列表放在首位。例如，如下语句并非很好：

```
acl All src 0/0
acl Net1 src 1.2.3.0/24
acl Net2 src 1.2.4.0/24
acl Net3 src 1.2.5.0/24
acl Net4 src 1.2.6.0/24
acl WorkingHours time 08:00-17:00
http_access allow Net1 WorkingHours
http_access allow Net2 WorkingHours
http_access allow Net3 WorkingHours
http_access allow Net4
http_access deny All
```

假如你这样写，访问控制列表会更容易维护和理解：

```
http_access allow Net4
http_access deny !WorkingHours
http_access allow Net1
http_access allow Net2
http_access allow Net3
http_access deny All
```

无论何时，你编写了一个带两个或更多 ACL 元素的规则，建议你在其后紧跟一条相反的，更广泛的规则。例如，默认的 squid 配置拒绝非来自本机 IP 地址的 cache 管理请求，你也许试图这样写：

```
acl CacheManager proto cache_object
acl Localhost src 127.0.0.1
http_access deny CacheManager !Localhost
```

然而，这里的问题是，你没有允许确实来自本机的 cache 管理请求。随后的规则可能导致请求被拒绝。如下规则就产生了问题：

```
acl CacheManager proto cache_object
acl Localhost src 127.0.0.1
acl MyNet 10.0.0.0/24
acl All src 0/0
http_access deny CacheManager !Localhost
http_access allow MyNet
http_access deny All
```

既然来自本机的请求不匹配 MyNet，它被拒绝。编写本规则的更好方法是：

```
http_access allow CacheManager localhost
http_access deny CacheManager
http_access allow MyNet
http_access deny All
```

6.2.4 延时检查

某些 ACL 不能在一个过程里被检查，因为必要的信息不可用。ident,dst,srcdomain 和 proxy_auth 类型属于该范畴。当 squid 遇到某个 ACL 不能被检查时，它延迟决定并且发布对必要信息的查询（IP 地址，域名，用户名等）。当信息可用时，squid 再次在列表的开头位置检查这些规则。它不会从前次检查剩下的位置继续。假如可能，你应该将这些最可能被延时的 ACL 放在规则的顶部，以避免不必要的，重复的检查。

因为延时的代价太大，squid 会尽可能缓存查询获取的信息。ident 查询在每个连接里发生，而不是在每个请求里。这意味着，当你使用 ident 查询时，持续 HTTP 连接切实对你有利。DNS 响应的主机名和 IP 地址也被缓存，除非你使用早期的外部 dnsserver 进程。代理验证信息被缓存，请见 6.1.2.12 章节的描述。

6.2.5 减缓和加速规则检查

Squid 内部考虑某些访问规则被快速检查，其他的被减缓检查。区别是 squid 是否延迟它的决定，以等待附加信息。换句话说，在 squid 查询附加信息时，某个减缓检查会被延时，例如：
+ 反向 DNS 查询：客户 IP 地址的主机名
+ RFC 1413 ident 查询：客户 TCP 连接的用户名
+ 验证器：验证用户信用
+ DNS 转发查询：原始服务器的 IP 地址
+ 用户定义的外部 ACL

某些访问规则使用快速检查。例如，icp_access 规则被快速检查。为了快速响应 ICP 查询，它必须被快速检查。甚至，某些 ACL 类型例如 proxy_auth，对 ICP 查询来说无意义。下列访问规则被快速检查：
header_access
reply_body_max_size
reply_access
ident_lookup
delay_access
miss_access
broken_posts
icp_access
cache_peer_access
redirector_access
snmp_access

下列 ACL 类型可能需要来自外部数据源（DNS，验证器等）的信息，这样与快速的访问规则不兼容：
srcdomain, dstdomain, srcdom_regex, dstdom_regex
dst, dst_as
proxy_auth
ident
external_acl_type

这意味着，例如，不能在 header_access 规则里使用 ident ACL。

6.3 常见用法

因为访问控制可能很复杂，本节包含一些示例。它们描述了一些访问控制的普通用法。你可以在实际中调整它们。

6.3.1 仅仅允许本地客户

几乎每个 squid 安装后，都限制基于客户 IP 地址的访问。这是保护你的系统不被滥用的最好的方法之一。做到这点最容易的方法是，编写包含 IP 地址空间的 ACL，然后允许该 ACL 的 HTTP 请求，并拒绝其他的。

```
acl All src 0/0
acl MyNetwork src 172.16.5.0/24 172.16.6.0/24
http_access allow MyNetwork
http_access deny All
```

也许该访问控制配置过于简单，所以你要增加更多行。记住 http_access 的顺序至关重要。不要在 deny all 后面增加任何语句。假如必要，应该在 allow MyNetwork 之前或之后增加新规则。

6.3.2 阻止恶意客户

因为某种理由，你也许有必要拒绝特定客户 IP 地址的访问。这种情况可能发生，例如，假如某个雇员或学生发起一个异常耗费网络带宽或其他资源的 web 连接，在根本解决这个问题前，你可以配置 squid 来阻止这个请求：

```
acl All src 0/0
acl MyNetwork src 172.16.5.0/24 172.16.6.0/24
acl ProblemHost src 172.16.5.9
http_access deny ProblemHost
http_access allow MyNetwork
http_access deny All
```

6.3.3 内容过滤

阻塞对特定内容的访问是棘手的问题。通常，使用 squid 进行内容过滤最难的部分，是被阻塞的站点列表。你也许想自己维护一个这样的列表，或从其他地方获取一个。squid FAQ 的“访问控制”章节有链接指向免费的可用列表。

使用这样的列表的 ACL 语法依赖于它的内容。假如列表包含正则表达式，你可能要这样写：

```
acl PornSites url_regex "/usr/local/squid/etc/pornlist"
http_access deny PornSites
```

另一方面，假如列表包含原始服务器主机名，那么简单的更改 url_regex 为 dstdomain。

6.3.4 在工作时间的受限使用

某些公司喜欢在工作时间限制 web 使用，为了节省带宽，或者是公司政策禁止员工在工作时做某些事情。关于这个最难的部分是，所谓合适的和不合适的 internet 使用之间的区别是什么。不幸的是，我不能对这个问题作出回答。在该例子里，假设你已收集了一份 web 站点域名列表，

它包含已知的不适合于你的站点名，那么这样配置 squid：
```
acl NotWorkRelated dstdomain "/usr/local/squid/etc/not-work-related-sites"
acl WorkingHours time D 08:00-17:30
http_access deny !WorkingHours NotWorkRelated
```

请注意在该规则里首先放置!WorkingHours ACL。相对于字符串或列表，dstdomain ACL 产生的性能代价较大，但 time ACL 检查却很简单。

下面的例子，进一步理解如何结合如下方法和前面描述的源地址控制，来控制访问。
```
acl All src 0/0
acl MyNetwork src 172.16.5.0/24 172.16.6.0/24
acl NotWorkRelated dstdomain "/usr/local/squid/etc/not-work-related-sites"
acl WorkingHours time D 08:00-17:30
http_access deny !WorkingHours NotWorkRelated
http_access allow MyNetwork
http_access deny All
```

上面的方法可行，因为它实现了我们的目标，在工作时间内拒绝某些请求，并允许来自你自己网络的请求。然而，它也许有点低效。注意 NotWorkRelated ACL 在所有请求里被搜索，而不管源 IP 地址。假如那个列表非常长，在列表里对外部网络请求的搜索，纯粹是浪费 CPU 资源。所以，你该这样改变规则：
```
http_access deny !MyNetwork
http_access deny !WorkingHours NotWorkRelated
http_access Allow All
```

这里，将代价较大的检查放在最后。试图滥用 squid 的外部用户不会再浪费你的 CPU 资源。

6.3.5 阻止 squid 与非 HTTP 服务器会话

你必须尽可能不让 squid 与某些类型的 TCP/IP 服务器通信。例如，永不能够使用 squid 缓存来转发 SMTP 传输。我在前面介绍 port ACL 时提到过这点。然而，它是至关重要的，所以再强调一下。

首先，你必须关注 CONNECT 请求方法。使用该方法的用户代理，通过 HTTP 代理来封装 TCP 连接。它被创造用于 HTTP/TLS 请求，这是 CONNECT 方法的主要用途。某些用户代理也可以通过防火墙代理来封装 NNTP/TLS 传输。所有其他的用法应该被拒绝。所以，
你的访问列表，应该仅仅允许到 HTTP/TLS 和 NNTP/TLS 端口的 CONNECT 请求。

第二，你应该阻止 squid 连接到某些服务，例如 SMTP。你也可以开放安全端口和拒绝危险端口。我对这两种技术给出示例。

让我们看看默认的 squid.conf 文件提供的规则：
```
acl Safe_ports port 80 # http
```

```
acl Safe_ports port 21 # ftp
acl Safe_ports port 443 563 # https, snews
acl Safe_ports port 70 # gopher
acl Safe_ports port 210 # wais
acl Safe_ports port 280 # http-mgmt
acl Safe_ports port 488 # gss-http
acl Safe_ports port 591 # filemaker
acl Safe_ports port 777 # multiling http
acl Safe_ports port 1025-65535 # unregistered ports
acl SSL_ports port 443 563
acl CONNECT method CONNECT
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
<additional http_access lines as necessary...>;
```

Safe_ports ACL 列举了所有的 squid 有合法响应的特权端口（小于 1024）。它也列举了所有非特权端口范围。注意 Safe_ports ACL 也包括了安全 HTTP 和 NNTP 端口（443 和 563），即使它们也出现在 SSL_ports ACL 里。这是因为 Safe_ports 在规则里首先被检查。假如你交换了两个 http_access 行的顺序，你也许能从 Safe_ports 列表里删除 443 和 563，但没必要这么麻烦。

与此相似的其他方法是，列举已知不安全的特权端口：

```
acl Dangerous_ports 7 9 19 22 23 25 53 109 110 119
acl SSL_ports port 443 563
acl CONNECT method CONNECT
http_access deny Dangerous_ports
http_access deny CONNECT !SSL_ports
<additional http_access lines as necessary...>;
```

假如你不熟悉这些奇特的端口号，也不要担心。你可以阅读 unix 系统的/etc/services 文件，或者阅读 IANA 的注册 TCP/UDP 端口号列表：
http://www.iana.org/assignments/port-numbers

6.3.6 授予某些用户特殊的访问

使用基于用户名进行访问控制的组织，通常需要授予某些用户特殊的权限。在该简单示例里，有三个元素：所有授权用户，管理员用户名，限制访问的 web 站点列表。正常的用户不允许访问受限站点，但管理员有维护这个列表的任务。他们必须连接到所有服务器，去验证某个特殊站点是否该放到受限站点列表里。如下显示如何完成这个任务：

```
auth_param basic program /usr/local/squid/libexec/ncsa_auth
/usr/local/squid/etc/passwd
acl Authenticated proxy_auth REQUIRED
acl Admins proxy_auth Pat Jean Chris
```

```
acl Porn dstdomain "/usr/local/squid/etc/porn.domains"
acl All src 0/0
http_access allow Admins
http_access deny Porn
http_access allow Authenticated
http_access deny All
```

首先，有三个 ACL 定义。Authenticated ACL 匹配任何有效的代理验证信用。Admins ACL 匹配来自用户 Pat,Jean,和 Chris 的有效信用。Porn ACL 匹配某些原始服务器主机名，它们在 porn.domains 文件里找到。

该示例有四个访问控制规则。第一个仅仅检查 Admins ACL，允许所有来自 Pat,Jean,和 Chris 的请求。对其他用户，squid 转移到下一条规则。对第二条规则，假如原始主机名位于 porn.domains 文件，那么该请求被拒绝。对不匹配 Porn ACL 的请求，squid 转移到第三条规则。第三条规则里，假如请求包含有效的验证信用，那么该请求被允许。外部验证器（这里的 ncsa_auth）决定是否信用有效。假如它们无效，最后的规则出现，该请求被拒绝。

注意 ncsa_auth 验证器并非必需。你可以使用 12 章里描述的任何验证辅助程序。

6.3.7 阻止邻近 cache 的滥用

假如你使用了 cache 集群，你必须付出多余的小心。cache 通常使用 ICP 来发现哪些对象被缓存在它们的邻居机器上。你仅该接受来自已知授权的邻居 cache 的 ICP 查询。

更进一步，通过使用 miss_access 规则列表，你能配置 squid 强制限制邻近关系。squid 仅仅在 cache 丢失，没有 cache 命中时才检查这些规则。这样，在 miss_access 列表生效前，所有请求必须首先通过 http_access 规则。

在本示例里，有三个独立的 ACL。一个是直接连接到 cache 的本地用户；另一个是子 cache，它被允许来转发 cache 丢失的请求；第三个是邻近 cache，它必须从不转发导致 cache 丢失的请求。如下是它们如何工作：

```
alc All src 0/0
acl OurUsers src 172.16.5.0/24
acl ChildCache src 192.168.1.1
acl SiblingCache src 192.168.3.3
http_access allow OurUsers
http_access allow ChildCache
http_access allow SiblingCache
http_access deny All
miss_access deny SiblingCache
icp_access allow ChildCache
icp_access allow SiblingCache
icp_access deny All
```

6.3.8 使用 IP 地址拒绝请求

我在 6.1.2.4 章节里提过，dstdomain 类型是阻塞对指定原始主机访问的好选择。然而，聪明的用户通过替换 URL 主机名成 IP 地址，能够绕过这样的规则。假如你想彻底阻止这样的请求，你可能得阻塞所有包含 IP 地址的请求。你可以使用重定向器，或者使用 dstdom_regex ACL 来完成。例如：

```
acl IPForHostname dstdom_regex ^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+$
http_access deny IPForHostname
```

6.3.9 http_reply_access 示例

回想一下，当 squid 检查 http_reply_access 规则时，响应的内容类型是唯一的可用新信息。这样，你能保持 http_reply_access 规则简单化。你只需检查 rep_mime_type ACL。例如，如下示例告诉你如何拒绝某些内容类型的响应：

```
acl All src 0/0
acl Movies rep_mime_type video/mpeg
acl MP3s rep_mime_type audio/mpeg
http_reply_access deny Movies
http_reply_access deny MP3s
http_reply_access allow All
```

你不必在 http_reply_access 列表里重复 http_access 规则。这里的 allow ALL 规则不意味着所有对 squid 的请求被允许。任何被 http_access 拒绝的请求，从来不会再被 http_reply_access 检查。

6.3.10 阻止对本地站点的 cache 命中

假如你有许多原始服务器在本地网络中，你也许想配置 squid，以便它们的响应永不被缓存。因为服务器就在附近，它们不会从 cache 命中里获益很多。另外，它释放存储空间给其他远程原始主机。

第一步是定义本地服务器的 ACL。你可能使用基于地址的 ACL，例如：

```
acl LocalServers dst 172.17.1.0/24
```

假如服务器不位于单一的子网，你也许该创建 dstdomain ACL:

```
acl LocalServers dstdomain .example.com
```

接下来，你简单的使用 no_cache access 规则，拒绝这些服务器的 cache：

```
no_cache deny LocalServers
```

no_cache 规则不会阻止客户发送请求到 squid。没有办法配置 squid 阻止这样的请求进来。代替的，你必须配置用户代理自身。

假如你在 squid 运行一段时间后增加 no_cache 规则，cache 可能包含一些匹配新规则的对象。在 squid2.5 之前的版本，这些以前缓存的对象可能以 cache 命中返回。然而现在，squid 清除掉所有匹配 no_cache 规则的缓存响应。

6.4 测试访问控制

访问控制配置越长，它就越复杂。强烈建议你在将它们用于产品环境之前，先测试访问控制。当然，首先做的事是确认 squid 能正确的解析配置文件。使用-k parse 功能：

```
% squid -k parse
```

为了进一步测试访问控制，你需要安装一个用于测试的 squid。容易做到的方法是，编译另一份 squid 到其他$prefix 位置。例如：

```
% tar xzvf squid-2.5.STABLE4.tar.gz
% cd squid-2.5.STABLE4
% ./configure --prefix=/tmp/squid ...
% make && make install
```

在安装完后，你必须编辑新的 squid.conf 文件，更改一些指令。假如 squid 已经运行在默认端口，那么请改变 http_port。为了执行简单的测试，创建单一的小目录：

```
cache_dir ufs /tmp/squid/cache 100 4 4
```

假如你不想重编译 squid，你也能创建一份新的配置文件。该方法的弊端是你必须设置所有的日志文件路径为临时目录，以便不会覆盖真正的文件。

你可以使用 squidclient 程序来轻松的测试某些访问控制。例如，假如你有一条规则，它依赖于原始服务器主机名（dstdomain ACL），或者某些 URL 部分（url_regex 或 urlpath_regex），简单的输入你期望被允许或拒绝的 URI:

```
% squidclient -p 4128 http://blocked.host.name/blah/blah
or:
% squidclient -p 4128 http://some.host.name/blocked.ext
```

某些类型的请求难以控制。假如你有 src ACL，它们阻止来自外部网络的请求，你也许需要从外部主机测试它们。测试 time ACL 也很困难，除非你能改变系统时钟，或者等待足够长时间。你能使用 squidclient 的-H 选项来设置任意请求头。例如，假如你需要测试 browser ACL，那么这样做：

```
% squidclient -p 4128 http://www.host.name/blah \
-H 'User-Agent: Mozilla/5.0 (compatible; Konqueror/3)\r\n'
```

更多的复杂请求，包括多个头部，请参考 16.4 章中描述的技术。

你也许考虑制订一项 cron，定期检查 ACL，以发现期望的行为，并报告任何异常。如下是可以

起步的示例 shell 脚本：

```
#!/bin/sh
set -e
TESTHOST="www.squid-cache.org"
# make sure Squid is not proxying dangerous ports
#
ST=`squidclient 'http://$TESTHOST:25/' | head -1 | awk '{print $2}'`
if test "$ST" != 403 ; then
echo "Squid did not block HTTP request to port 25"
fi
# make sure Squid requires user authentication
#
ST=`squidclient 'http://$TESTHOST/' | head -1 | awk '{print $2}'`
if test "$ST" != 407 ; then
echo "Squid allowed request without proxy authentication"
fi
# make sure Squid denies requests from foreign IP addresses
# elsewhere we already created an alias 192.168.1.1 on one of
# the system interfaces
#
EXT_ADDR=192.168.1.1
ST=`squidclient -l $EXT_ADDR 'http://$TESTHOST/' | head -1 | awk '{print $2}'`
if test "$ST" != 403 ; then
echo "Squid allowed request from external address $EXT_ADDR"
fi
exit 0
```

第 7 章．磁盘缓存基础

7.1 cache_dir 指令

cache_dir 指令是 squid.conf 配置文件里最重要的指令之一。它告诉 squid 以何种方式存储 cache 文件到磁盘的什么位置。cache_dir 指令取如下参数：

```
cache_dir scheme directory size L1 L2 [options]
```

7.1.1 参数：Scheme

Squid 支持许多不同的存储机制。默认的（原始的）是 ufs。依赖于操作系统的不同，你可以选择不同的存储机制。在./configure 时，你必须使用--enable-storeio=LIST 选项来编译其他存储机制的附加代码。我将在 8.7 章讨论 aufs,diskd,coss 和 null。现在，我仅仅讨论 ufs 机

制，它与 aufs 和 diskd 一致。

7.1.2 参数：Directory

该参数是文件系统目录，squid 将 cache 对象文件存放在这个目录下。正常的，cache_dir 使用整个文件系统或磁盘分区。它通常不介意是否在单个文件系统分区里放置了多个 cache 目录。然而，我推荐在每个物理磁盘中，仅仅设置一个 cache 目录。例如，假如你有 2 个无用磁盘，你可以这样做：

```
# newfs /dev/da1d
# newfs /dev/da2d
# mount /dev/da1d /cache0
# mount /dev/da2d /cache1
```

然后在 squid.conf 里增加如下行：

```
cache_dir ufs /cache0 7000 16 256
cache_dir ufs /cache1 7000 16 256
```

假如你没有空闲硬盘，当然你也能使用已经存在的文件系统分区。选择有大量空闲空间的分区，例如/usr 或/var，然后在下面创建一个新目录。例如：

```
# mkdir /var/squidcache
```

然后在 squid.conf 里增加如下一行：

```
cache_dir ufs /var/squidcache 7000 16 256
```

7.1.3 参数：Size

该参数指定了 cache 目录的大小。这是 squid 能使用的 cache_dir 目录的空间上限。计算出合理的值也许有点难。你必须给临时文件和 swap.state 日志，留出足够的自由空间（见 13.6 章）。我推荐挂载空文件系统，可以运行 df：

```
% df -k
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/da1d     3037766        8  2794737     0%    /cache0
/dev/da2d     3037766        8  2794737     0%    /cache1
```

这里你可以看到文件系统有大约 2790M 的可用空间。记住，UFS 保留了部分最小自由空间，这里约是 8%，这就是 squid 为什么不能使用全部 3040M 空间的原因。

你也许试图分配 2790M 给 cache_dir。如果 cache 不很繁忙，并且你经常轮转日志，那么这样做也许可行。然而，为安全起见，我推荐保留 10%的空间。这些额外的空间用于存放 squid 的 swap.state 文件和临时文件。

注意 cache_swap_low 指令也影响了 squid 使用多少空间。我将在 7.2 章里讨论它的上限和下限。

底线是，你在初始时应保守的估计 cache_dir 的大小。将 cache_dir 设为较小的值，并允许写满 cache。在 squid 运行一段时间后，cache 目录会填满，这样你可以重新评估 cache_dir 的大小设置。假如你有大量的自由空间，就可以轻松的增加 cache 目录的大小了。

7.1.3.1 Inodes

Inodes（i 节点）是 unix 文件系统的基本结构。它们包含磁盘文件的信息，例如许可，属主，大小，和时间戳。假如你的文件系统运行超出了 i 节点限制，就不能创造新文件，即使还有空间可用。超出 i 节点的系统运行非常糟糕，所以在运行 squid 之前，你应该确认有足够的 i 节点。

创建新文件系统的程序（例如,newfs 或 mkfs）基于总空间的大小，保留了一定数量的 i 节点。这些程序通常允许你设置磁盘空间的 i 节点比率。例如，请阅读 newfs 和 mkfs 手册的-i 选项。磁盘空间对 i 节点的比率，决定了文件系统能实际支持的文件大小。大部分 unix 系统每 4KB 创建一个 i 节点，这对 squid 通常是足够的。研究显示，对大部分 cache 代理，实际文件大小大约是 10KB。你也许能以每 i 节点 8KB 开始，但这有风险。
你能使用 df -i 命令来监视系统的 i 节点，例如：

```
% df -ik
Filesystem  1K-blocks     Used    Avail Capacity iused   ifree  %iused  Mounted on
/dev/ad0s1a    197951    57114   125001    31%    1413   52345     3%   /
/dev/ad0s1f   5004533  2352120  2252051    51%  129175 1084263    11%   /usr
/dev/ad0s1e    396895     6786   358358     2%     205   99633     0%   /var
/dev/da0d     8533292  7222148   628481    92%  430894  539184    44%   /cache1
/dev/da1d     8533292  7181645   668984    91%  430272  539806    44%   /cache2
/dev/da2d     8533292  7198600   652029    92%  434726  535352    45%   /cache3
/dev/da3d     8533292  7208948   641681    92%  427866  542212    44%   /cache4
```

如果 i 节点的使用（%iused）少于空间使用（Capacity），那就很好。不幸的是，你不能对已经存在的文件系统增加更多 i 节点。假如你发现运行超出了 i 节点，那就必须停止 squid，并且重新创建文件系统。假如你不愿意这样做，那么请削减 cache_dir 的大小。

7.1.3.2 在磁盘空间和进程大小之间的联系

Squid 的磁盘空间使用也直接影响了它的内存使用。每个在磁盘中存在的对象，要求少量的内存。squid 使用内存来索引磁盘数据。假如你增加了新的 cache 目录，或者增加了磁盘 cache 大小，请确认你已有足够的自由内存。假如 squid 的进程大小达到或超过了系统的物理内存容

量，squid 的性能下降得非常块。

Squid 的 cache 目录里的每个对象消耗 76 或 112 字节的内存，这依赖于你的系统。内存以 StoreEntry, MD5 Digest, 和 LRU policy node 结构来分配。小指令（例如，32 位）系统，象那些基于 Intel Pentium 的，取 76 字节。使用 64 位指令 CPU 的系统，每个目标取 112 字节。通过阅读 cache 管理的内存管理文档，你能发现这些结构在你的系统中耗费多少内存（请见 14.2.1.2 章）。

不幸的是，难以精确预测对于给定数量的磁盘空间，需要使用多少附加内存。它依赖于实际响应大小，而这个大小基于时间波动。另外，Squid 还为其他数据结构和目的分配内存。不要假设你的估计正确。你该经常监视 squid 的进程大小，假如必要，考虑削减 cache 大小。

7.1.4 参数：L1 和 L2

对 ufs,aufs,和 diskd 机制，squid 在 cache 目录下创建二级目录树。L1 和 L2 参数指定了第一级和第二级目录的数量。默认的是 16 和 256。图 7-1 显示文件系统结构。

Figure 7-1. 基于 ufs 存储机制的 cache 目录结构

（略图）

某些人认为 squid 依赖于 L1 和 L2 的特殊值，会执行得更好或更差。这点听起来有关系，即小目录比大目录被检索得更快。这样，L1 和 L2 也许该足够大，以便 L2 目录的文件更少。

例如，假设你的 cache 目录存储了 7000M，假设实际文件大小是 10KB，你能在这个 cache_dir 里存储 700,000 个文件。使用 16 个 L1 和 256 个 L2 目录，总共有 4096 个二级目录。700,000/4096 的结果是，每个二级目录大约有 170 个文件。

如果 L1 和 L2 的值比较小，那么使用 squid -z 创建交换目录的过程，会执行更快。这样，假如你的 cache 文件确实小，你也许该减少 L1 和 L2 目录的数量。

Squid 给每个 cache 目标分配一个唯一的文件号。这是个 32 位的整数，它唯一标明磁盘中的文件。squid 使用相对简单的算法，将文件号转换位路径名。该算法使用 L1 和 L2 作为参数。这样，假如你改变了 L1 和 L2，你改变了从文件号到路径名的映射关系。对非空的 cache_dir 改变这些参数，导致存在的文件不可访问。在 cache 目录激活后，你永不要改变 L1 和 L2 值。

Squid 在 cache 目录顺序中分配文件号。文件号到路径名的算法（例如，storeUfsDirFullPath( )），用以将每组 L2 文件映射到同样的二级目录。Squid 使用了参考位置来做到这点。该算法让 HTML 文件和它内嵌的图片更可能的保存在同一个二级目录中。某些人希望 squid 均匀的将 cache 文件放在每个二级目录中。然而，当 cache 初始写入时，你可以发现仅仅开头的少数目录包含了一些文件，例如：

```
% cd /cache0; du -k
2164    ./00/00
2146    ./00/01
2689    ./00/02
1974    ./00/03
2201    ./00/04
2463    ./00/05
2724    ./00/06
3174    ./00/07
1144    ./00/08
```

1 ./00/09
1 ./00/0A
1 ./00/0B

这是完全正常的，不必担心。

7.1.5 参数：Options

Squid 有 2 个依赖于不同存储机制的 cache_dir 选项：read-only 标签和 max-size 值。

7.1.5.1 read-only

read-only 选项指示 Squid 继续从 cache_dir 读取文件，但不往里面写新目标。它在 squid.conf 文件里看起来如下：

```
cache_dir ufs /cache0 7000 16 256 read-only
```

假如你想把 cache 文件从一个磁盘迁移到另一个磁盘，那么可使用该选项。如果你简单的增加一个 cache_dir，并且删除另一个，squid 的命中率会显著下降。在旧目录是 read-only 时，你仍能从那里获取 cache 命中。在一段时间后，就可以从配置文件里删除 read-only 缓存目录。

7.1.5.2 max-size

使用该选项，你可以指定存储在 cache 目录里的最大目标大小。例如：

```
cache_dir ufs /cache0 7000 16 256 max-size=1048576
```

注意值是以字节为单位的。在大多数情况下，你不必增加该选项。假如你做了，请尽力将所有 cache_dir 行以 max-size 大小顺序来存放（从小到大）。

7.2 磁盘空间基准

cache_swap_low 和 cache_swap_high 指令控制了存储在磁盘上的对象的置换。它们的值是最大 cache 体积的百分比，这个最大 cache 体积来自于所有 cache_dir 大小的总和。例如：

```
cache_swap_low 90
cache_swap_high 95
```

如果总共磁盘使用低于 cache_swap_low，squid 不会删除 cache 目标。如果 cache 体积增加，squid 会逐渐删除目标。在稳定状态下，你发现磁盘使用总是相对接近 cache_swap_low 值。你可以通过请求 cache 管理器的 storedir 页面来查看当前磁盘使用状况（见 14.2.1.39 章）。

请注意，改变 cache_swap_high 也许不会对 squid 的磁盘使用有太大效果。在 squid 的早期版本里，该参数有重要作用；然而现在，它不是这样了。

7.3 对象大小限制

你可以控制缓存对象的最大和最小体积。比 maximum_object_size 更大的响应不会被缓存在磁盘。然而，它们仍然是代理方式的。在该指令后的逻辑是，你不想某个非常大的响应来浪费空间，这些空间能被许多小响应更好的利用。该语法如下：

```
maximum_object_size size-specification
```

如下是一些示例：

```
maximum_object_size 100 KB
```

maximum_object_size 1 MB
maximum_object_size 12382 bytes
maximum_object_size 2 GB

Squid 以两个不同的方法来检查响应大小。假如响应包含了 Content-Length 头部，squid 将这个值与 maximum_object_size 值进行比较。假如前者大于后者，该对象立刻不可缓存，并且不会消耗任何磁盘空间。

不幸的是，并非每个响应都有 Content-Length 头部。在这样的情形下，squid 将响应写往磁盘，把它当作来自原始服务器的数据。在响应完成后，squid 再检查对象大小。这样，假如对象的大小达到 maximum_object_size 限制，它继续消耗磁盘空间。仅仅当 squid 在做读取响应的动作时，总共 cache 大小才会增大。

换句话说，活动的，或者传输中的目标，不会对 squid 内在的 cache 大小值有影响。这点有好处，因为它意味着 squid 不会删除 cache 里的其他目标，除非目标不可缓存，并对总共 cache 大小有影响。然而，这点也有坏处，假如响应非常大，squid 可能运行超出了磁盘自由空间。为了减少发生这种情况的机会，你应该使用 reply_body_max_size 指令。某个达到 reply_body_max_size 限制的响应立即被删除。

Squid 也有一个 minimum_object_size 指令。它允许你对缓存对象的大小设置最低限制。比这个值更小的响应不会被缓存在磁盘或内存里。注意这个大小是与响应的内容长度（例如，响应 body 大小）进行比较，后者包含在 HTTP 头部里。

7.4 分配对象到缓存目录

当 squid 想将某个可缓存的响应存储到磁盘时，它调用一个函数，用以选择 cache 目录。然后它在选择的目录里打开一个磁盘文件用于写。假如因为某些理由，open()调用失败，响应不会被存储。在这样的情况下，squid 不会试图在其他 cache 目录里打开另一个磁盘文件。

Squid 有 2 个 cache_dir 选择算法。默认的算法叫做 lease-load；替代的算法是 round-robin。

least-load 算法，就如其名字的意义一样，它选择当前工作负载最小的 cache 目录。负载概念依赖于存储机制。对 aufs,coss 和 diskd 机制来说，负载与挂起操作的数量有关。对 ufs 来说，负载是不变的。在 cache_dir 负载相等的情况下，该算法使用自由空间和最大目标大小作为附加选择条件。

该选择算法也取决于 max-size 和 read-only 选项。假如 squid 知道目标大小超出了限制，它会跳过这个 cache 目录。它也会跳过任何只读目录。

round-robin 算法也使用负载作为衡量标准。它选择某个负载小于 100%的 cache 目录，当然，该目录里的存储目标没有超出大小限制，并且不是只读的。

在某些情况下，squid 可能选择 cache 目录失败。假如所有的 cache_dir 是满负载，或者所有

目录的实际目标大小超出了 max-size 限制，那么这种情况可能发生。这时，squid 不会将目标写往磁盘。你可以使用 cache 管理器来跟踪 squid 选择 cache 目录失败的次数。请见 store_io 页（14.2.1.41 章），找到 create.select_fail 行。

7.5 置换策略

cache_replacement_policy 指令控制了 squid 的磁盘 cache 的置换策略。Squid2.5 版本提供了三种不同的置换策略：最少近来使用（LRU），贪婪对偶大小次数（GDSF），和动态衰老最少经常使用（LFUDA)。

LRU 是默认的策略，并非对 squid，对其他大部分 cache 产品都是这样。LRU 是流行的选择，因为它容易执行，并提供了非常好的性能。在 32 位系统上，LRU 相对于其他使用更少的内存（每目标 12 对 16 字节）。在 64 位系统上，所有的策略每目标使用 24 字节。

在过去，许多研究者已经提议选择 LRU。其他策略典型的被设计来改善 cache 的其他特性，例如响应时间，命中率，或字节命中率。然而研究者的改进结果也可能在误导人。某些研究使用并不现实的小 cache 目标；其他研究显示当 cache 大小增加时，置换策略的选择变得不那么重要 。

假如你想使用 GDSF 或 LFUDA 策略，你必须在 ./configure 时使用 --enable-removal-policies 选项（见 3.4.1 章）。Martin Arlitt 和 HP 实验室的 John Dilley 为 squid 写了 GDSF 和 LFUDA 算法。你可以在线阅读他们的文档：
http://www.hpl.hp.com/techreports/1999/HPL-1999-69.html

我在 O'Reilly 出版的书"Web Caching"，也讨论了这些算法。

cache_replacement_policy 指令的值是唯一的，这点很重要。不象 squid.conf 里的大部分其他指令，这个指令的位置很重要。cache_replacement_policy 指令的值在 squid 解析 cache_dir 指令时，被实际用到。通过预先设置替换策略，你可以改变 cache_dir 的替换策略。例如：

```
cache_replacement_policy lru
cache_dir ufs /cache0 2000 16 32
cache_dir ufs /cache1 2000 16 32

cache_replacement_policy heap GDSF
cache_dir ufs /cache2 2000 16 32
cache_dir ufs /cache3 2000 16 32
```

在该情形中，头 2 个 cache 目录使用 LRU 置换策略，接下来 2 个 cache 目录使用 GDSF。请记住，假如你已决定使用 cache 管理器的 config 选项（见 14.2.1.7 章），这个置换策略指令的特性就非常重要。cache 管理器仅仅输出最后一个置换策略的值，将它置于所有的 cache 目录之前。例如，你可能在 squid.conf 里有如下行：

```
cache_replacement_policy heap GDSF
cache_dir ufs /tmp/cache1 10 4 4
```

cache_replacement_policy lru
cache_dir ufs /tmp/cache2 10 4 4

但当你从 cache 管理器选择 config 时，你得到：
cache_replacement_policy lru
cache_dir ufs /tmp/cache1 10 4 4
cache_dir ufs /tmp/cache2 10 4 4

就象你看到的一样，对头 2 个 cache 目录的 heap GDSF 设置被丢失了。

7.6 删除缓存对象

在某些情况下，你必须从 squid 的 cache 里手工删除一个或多个对象。这些情况可能包括：
+ 你的用户抱怨总接收到过时的数据；
+ 你的 cache 因为某个响应而"中毒"；
+ Squid 的 cache 索引在经历磁盘 I/O 错误或频繁的 crash 和重启后，变得有问题；
+ 你想删除一些大目标来释放空间给新的数据；
+ Squid 总从本地服务器中 cache 响应，现在你不想它这样做。

上述问题中的一些可以通过强迫 web 浏览器 reload 来解决。然而，这并非总是可靠。例如，一些浏览器通过载入另外的程序，从而显示某些类容类型；那个程序可能没有 reload 按钮，或甚至它了解 cache 的情况。

假如必要，你总可以使用 squidclient 程序来 reload 缓存目标。简单的在 uri 前面使用-r 选项:
% squidclient -r http://www.lrrr.org/junk >;/tmp/foo

假如你碰巧在 refresh_pattern 指令里设置了 ignore-reload 选项，你和你的用户将不能强迫缓存响应更新。在这样的情形下，你最好清除这些有错误的缓存对象。

7.6.1 删除个别对象

Squid 接受一种客户请求方式，用于删除 cache 对象。PURGE 方式并非官方 HTTP 请求方式之一。它与 DELETE 不同，对后者，squid 将其转发到原始服务器。PURGE 请求要求 squid 删除在 uri 里提交的目标。squid 返回 200（OK）或 404（Not Found）。

PURGE 方式某种程度上有点危险，因为它删除了 cache 目标。除非你定义了相应的 ACL，否则 squid 禁止 PURGE 方式。正常的，你仅仅允许来自本机和少数可信任主机的 PURGE 请求。配置看起来如下：
acl AdminBoxes src 127.0.0.1 172.16.0.1 192.168.0.1
acl Purge method PURGE

http_access allow AdminBoxes Purge

http_access deny Purge

squidclient 程序提供了产生 PURGE 请求的容易方法，如下：
% squidclient -m PURGE http://www.lrrr.org/junk

代替的，你可以使用其他工具（例如 perl 脚本）来产生你自己的 HTTP 请求。它非常简单：
PURGE http://www.lrrr.org/junk HTTP/1.0
Accept: */*

注意某个单独的 URI 不唯一标明一个缓存响应。Squid 也在 cache 关键字里使用原始请求方式。假如响应包含了不同的头部，它也可以使用其他请求头。当你发布 PURGE 请求时，Squid 使用 GET 和 HEAD 的原始请求方式来查找缓存目标。而且，Squid 会删除响应里的所有 variants，除非你在 PURGE 请求的相应头部里指定了要删除的 variants。Squid 仅仅删除 GET 和 HEAD 请求的 variants。

7.6.2 删除一组对象

不幸的是，Squid 没有提供一个好的机制，用以立刻删除一组对象。这种要求通常出现在某人想删除所有属于同一台原始服务器的对象时。

因为很多理由，squid 不提供这种功能。首先，squid 必须遍历所有缓存对象，执行线性搜索，这很耗费 CPU，并且耗时较长。当 squid 在搜索时，用户会面临性能下降问题。第二，squid 在内存里对 URI 保持 MD5 算法，MD5 是单向哈希，这意味着，例如，你不能确认是否某个给定的 MD5 哈希是由包含"www.example.com"字符串的 URI 产生而来。唯一的方法是从原始 URI 重新计算 MD5 值，并且看它们是否匹配。因为 squid 没有保持原始的 URI，它不能执行这个重计算。

那么该怎么办呢？

你可以使用 access.log 里的数据来获取 URI 列表，它们可能位于 cache 里。然后，将它们用于 squidclient 或其他工具来产生 PURGE 请求，例如：

```
% awk '{print $7}' /usr/local/squid/var/logs/access.log \
        | grep www.example.com \
        | xargs -n 1 squidclient -m PURGE
```

7.6.3 删除所有对象

在极度情形下，你可能需要删除整个 cache，或至少某个 cache 目录。首先，你必须确认 squid 没有在运行。

让 squid 忘记所有缓存对象的最容易的方法之一，是覆盖 swap.state 文件。注意你不能简单的删除 swap.state 文件，因为 squid 接着要扫描 cache 目录和打开所有的目标文件。你也不

能简单的截断 swap.state 为 0 大小。代替的，你该放置一个单字节在里面，例如：
echo '' >; /usr/local/squid/var/cache/swap.state

当 squid 读取 swap.state 文件时，它获取到了错误，因为在这里的记录太短了。下一行读取就到了文件结尾，squid 完成重建过程，没有装载任何目标元数据。

注意该技术不会从磁盘里删除 cache 文件。你仅仅使 squid 认为它的 cache 是空的。当 squid 运行时，它增加新文件到 cache 里，并且可能覆盖旧文件。在某些情形下，这可能导致你的磁盘使用超出了自由空间。假如这样的事发生，你必须在再次重启 squid 前删除旧文件。

删除 cache 文件的方法之一是使用 rm。然而，它通常花费很长的时间来删除所有被 squid 创建的文件。为了让 squid 快速启动，你可以重命名旧 cache 目录，创建一个新目录，启动 squid，然后同时删除旧目录。例如：
squid -k shutdown
cd /usr/local/squid/var
mv cache oldcache
mkdir cache
chown nobody:nobody cache
squid -z
squid -s
rm -rf oldcache &

另一种技术是简单的在 cache 文件系统上运行 newfs（或 mkfs）。这点仅在你的 cache_dir 使用整个磁盘分区时才可以运行。

7.7 refresh_pattern

refresh_pattern 指令间接的控制磁盘缓存。它帮助 squid 决定，是否某个给定请求是 cache 命中，或作为 cache 丢失对待。宽松的设置增加了你的 cache 命中率，但也增加了用户接收过时响应的机会。另一方面，保守的设置，降低了 cache 命中率和过时响应。

refresh_pattern 规则仅仅应用到没有明确过时期限的响应。原始服务器能使用 Expires 头部，或者 Cache-Control:max-age 指令来指定过时期限。

你可以在配置文件里放置任意数量的 refresh_pattern 行。squid 按顺序查找它们以匹配正则表达式。当 squid 找到一个匹配时，它使用相应的值来决定，某个缓存响应是存活还是过期。
refresh_pattern 语法如下：
refresh_pattern [-i] regexp min percent max [options]

例如：
refresh_pattern -i \.jpg$ 30 50% 4320 reload-into-ims
refresh_pattern -i \.png$ 30 50% 4320 reload-into-ims
refresh_pattern -i \.htm$ 0 20% 1440

refresh_pattern -i \.html$ 0 20% 1440
refresh_pattern -i . 5 25% 2880

regexp 参数是大小写敏感的正则表达式。你可以使用-i 选项来使它们大小写不敏感。squid 按顺序来检查 refresh_pattern 行；当正则表达式之一匹配 URI 时，它停止搜索。

min 参数是分钟数量。它是过时响应的最低时间限制。如果某个响应驻留在 cache 里的时间没有超过这个最低限制，那么它不会过期。类似的，max 参数是存活响应的最高时间限制。如果某个响应驻留在 cache 里的时间高于这个最高限制，那么它必须被刷新。

在最低和最高时间限制之间的响应，会面对 squid 的最后修改系数 (LM-factor)算法。对这样的响应，squid 计算响应的年龄和最后修改系数，然后将它作为百分比值进行比较。响应年龄简单的就是从原始服务器产生，或最后一次验证响应后，经历的时间数量。源年龄在 Last-Modified 和 Date 头部之间是不同的。LM-factor 是响应年龄与源年龄的比率。

图 7-2 论证了 LM-factor 算法。squid 缓存了某个目标 3 个小时（基于 Date 和 Last-Modified 头部）。LM-factor 的值是 50%，响应在接下来的 1.5 个小时里是存活的，在这之后，目标会过期并被当作过时处理。假如用户在存活期间请求 cache 目标，squid 返回没有确认的 cache 命中。若在过时期间发生请求，squid 转发确认请求到原始服务器。

图 7-2 基于 LM-factor 计算过期时间

（略图）

理解 squid 检查不同值的顺序非常重要。如下是 squid 的 refresh_pattern 算法的简单描述：
+ 假如响应年龄超过 refresh_pattern 的 max 值，该响应过期；
+ 假如 LM-factor 少于 refresh_pattern 百分比值，该响应存活；
+ 假如响应年龄少于 refresh_pattern 的 min 值，该响应存活；
+ 其他情况下，响应过期。

refresh_pattern 指令也有少数选项导致 squid 违背 HTTP 协议规范。它们如下：
override-expire
该选项导致 squid 在检查 Expires 头部之前，先检查 min 值。这样，一个非零的 min 时间让 squid 返回一个未确认的 cache 命中，即使该响应准备过期。

override-lastmod
改选项导致 squid 在检查 LM-factor 百分比之前先检查 min 值。

reload-into-ims
该选项让 squid 在确认请求里，以 no-cache 指令传送一个请求。换句话说，squid 在转发请求之前，对该请求增加一个 If-Modified-Since 头部。注意这点仅仅在目标有 Last-Modified 时间戳时才能工作。外面进来的请求保留 no-cache 指令，以便它到达原始服务器。

ignore-reload

该选项导致 squid 忽略请求里的任何 no-cache 指令。

第 8 章 高级磁盘缓存主题

8.1 是否存在磁盘 I/O 瓶颈?

Web 缓存器例如 squid，通常在磁盘 I/O 变成瓶颈时，不会正确的体现和告知你。代替的是，随着负载的增加，响应时间和/或命中率会更低效。当然，响应时间和命中率可能因为其他原因而改变，例如网络延时和客户请求方式的改变。

也许探测 cache 性能瓶颈的最好方式是做压力测试，例如 Web Polygraph。压力测试的前提是你能完全控制环境，消除未知因素。你可以用不同的 cache 配置来重复相同的测试。不幸的是，压力测试通常需要大量的时间，并要求有空闲的系统（也许它们正在使用中）。

假如你有资源执行 squid 压力测试，请以标准的 cache 工作负载开始。当你增加负载时，在某些点上你能看到明显的响应延时和/或命中率下降。一旦你观察到这样的性能降低，就禁止掉磁盘缓存，再测试一次。你可以配置 squid 从来不缓存任何响应（使用 null 存储机制，见 8.7 章）。代替的，你能配置工作负载到 100%不可 cache 响应。假如不使用 cache 时，平均响应时间明显更好，那么可以确认磁盘 I/O 是该水平吞吐量的瓶颈。
假如你没有时间或没有资源来执行 squid 压力测试，那么可检查 squid 的运行时统计来查找磁盘 I/O 瓶颈。cache 管理器的 General Runtime Information 页面（见 14 章）会显示出 cache 命中和 cache 丢失的中值响应时间。

```
Median Service Times (seconds)  5 min     60 min:
        HTTP Requests (All):   0.39928  0.35832
        Cache Misses:          0.42149  0.39928
        Cache Hits:            0.12783  0.11465
        Near Hits:             0.37825  0.39928
        Not-Modified Replies:  0.07825  0.07409
```

对健壮的 squid 缓存来说，命中显然快于丢失。中值命中响应时间典型的少于 0.5 秒或更少。我强烈建议你使用 SNMP 或其他的网络监视工具来从 squid 缓存采集定期测量值。如果平均命中响应时间增加得太明显，意味着系统有磁盘 I/0 瓶颈。

假如你认为产品 cache 面临此类问题，可以用前面提到的同样的技术来验证你的推测。配置 squid 不 cache 任何响应，这样就避开了所有磁盘 I/O。然后仔细观察 cache 丢失响应时间。假如它降下去，那么你的推测该是正确的。

一旦你确认了磁盘吞吐能力是 squid 的性能瓶颈，那么可做许多事来改进它。其中一些方法要求重编译 squid，然而另一些相对较简单，只需调整 Unix 文件系统。

8.2 文件系统调整选项

首先，从来不在 squid 的缓存目录中使用 RAID。以我的经验看，RAID 总是降低 squid 使用的文件系统的性能。最好有许多独立的文件系统，每个文件系统使用单独的磁盘驱动器。

我发现 4 个简单的方法来改进 squid 的 UFS 性能。其中某些特指某种类型的操作系统例如 BSD 和 Linux，也许对你的平台不太合适：

1.某些 UFS 支持一个 noatime 的 mount 选项。使用 noatime 选项来 mount 的文件系统，不会在读取时，更新相应的 i 节点访问时间。使用该选项的最容易的方法是在/etc/fstab 里增加如下行：

```
# Device            Mountpoint    FStype  Options       Dump    Pass#
/dev/ad1s1c         /cache0       ufs     rw,noatime    0       0
```

2.检查 mount(😎的 manpage 里的 async 选项。设置了该选项，特定的 I/O 操作（例如更新目录）会异步执行。某些系统的文档会标明这是个危险的标签。某天你的系统崩溃，你也许会丢失整个文件系统。对许多 squid 安装来说，执行性能的提高值得冒此风险。假如你不介意丢失整个 cache 内容，那么可以使用该选项。假如 cache 数据非常有价值，async 选项也许不适合你。

3.BSD 有一个功能叫做软更新。软更新是 BSD 用于 Journaling 文件系统的代替品。在 FreeBSD 上，你可以在没有 mount 的文件系统中，使用 tunefs 命令来激活该选项：

```
# umount /cache0
# tunefs -n enable /cache0
# mount /cache0
```

4.你对每个文件系统运行一次 tunefs 命令就可以了。在系统重启时，软更新自动在文件系统中激活了。

在 OpenBSD 和 NetBSD 中，可使用 softdep mount 选项：

```
# Device            Mountpoint    FStype  Options       Dump    Pass#
/dev/sd0f           /usr          ffs     rw,softdep    1       2
```

假如你象我一样，你可能想知道在 async 选项和软更新选项之间有何不同。一个重要的区别是，软更新代码被设计成在系统崩溃事件中，保持文件系统的一致性，而 async 选项不是这样的。这也许让你推断 async 执行性能好于软更新。然而，如我在附录 D 中指出的，事实相反。

以前我提到过，UFS 性能特别是写性能，依赖于空闲磁盘的数量。对空文件系统的磁盘写操作，要比满文件系统快得多。这是 UFS 的最小自由空间参数，和空间/时间优化权衡参数背后的理由之一。假如 cache 磁盘满了，squid 执行性能看起来很糟，那么试着减少 cache_dir 的容量值，以便更多的自由空间可用。当然，减少 cache 大小也会降低命中率，但响应时间的改进也许值得这么做。假如你给 squid 缓存配置新的设备，请考虑使用超过你需要的更大磁盘，并且仅仅使用空间的一半。

8.3 可选择的文件系统

某些操作系统支持不同于 UFS（或 ext2fs）的文件系统。Journaling 文件系统是较普遍的选择。在 UFS 和 Journaling 文件系统之间的主要不同在于它们处理更新的方式。在 UFS 下，更新是实时的。例如，当你改变了某个文件并且将它存储到磁盘，新数据就替换了旧数据。当你删除文件时，UFS 直接更新了目录。

Journaling 文件系统与之相反，它将更新写往独立的记帐系统，或日志文件。典型的你能选择是否记录文件改变或元数据改变，或两者兼备。某个后台进程在空闲时刻读取记帐，并且执行实际的改变操作。Journaling 文件系统典型的在系统崩溃后比 UFS 恢复更快。在系统崩溃后，Journaling 文件系统简单的读取记帐，并且提交所有显著的改变。

Journaling 文件系统的主要弊端在于它们需要额外的磁盘写操作。改变首先写往日志文件，然后才写往实际的文件或目录。这对 web 缓存影响尤其明显，因为首先 web 缓存倾向于更多的磁盘写操作。

Journaling 文件系统对许多操作系统可用。在 Linux 上，你能选择 ext3fs,reiserfs, XFS,和其他的。XFS 也可用在 SGI/IRIX，它原始是在这里开发的。Solaris 用户能使用 Veritas 文件系统产品。TRU64（以前的 Digital Unix）高级文件系统（advfs）支持 Journaling。
你可以不改变 squid 的任何配置而使用 Journaling 文件系统。简单的创建和挂载在操作系统文档里描述的文件系统，而不必改变 squid.cf 配置文件里的 cache_dir 行。

用类似如下命令在 Linux 中制作 reiserfs 文件系统：
```
# /sbin/mkreiserfs /dev/sda2
```

对 XFS，使用：
```
# mkfs -t xfs -f /dev/sda2
```

注意 ext3fs 其实简单的就是激活了记帐的 ext2fs。当创建该文件系统时，对 mke2fs 使用-j 选项：
```
# /sbin/mke2fs -j /dev/sda2
```

请参考其他操作系统的相关文档。

8.4 aufs 存储机制

aufs 存储机制已经发展到超出了改进 squid 磁盘 I/O 响应时间的最初尝试。"a"代表着异步 I/O。默认的 ufs 和 aufs 之间的唯一区别，在于 I/O 是否被 squid 主进程执行。数据格式都是一样的，所以你能在两者之间轻松选择，而不用丢失任何 cache 数据。

aufs 使用大量线程进行磁盘 I/O 操作。每次 squid 需要读写，打开关闭，或删除 cache 文件时，I/O 请求被分派到这些线程之一。当线程完成了 I/O 后，它给 squid 主进程发送信号，并且返回一个状态码。实际上在 squid2.5 中，某些文件操作默认不是异步执行的。最明显的，磁盘写总是同步执行。你可以修改 src/fs/aufs/store_asyncufs.h 文件，将 ASYNC_WRITE 设

为 1，并且重编译 squid。

aufs 代码需要 pthreads 库。这是 POSIX 定义的标准线程接口。尽管许多 Unix 系统支持 pthreads 库，但我经常遇到兼容性问题。aufs 存储系统看起来仅仅在 Linux 和 Solaris 上运行良好。在其他操作系统上，尽管代码能编译，但也许会面临严重的问题。

为了使用 aufs，可以在./configure 时增加一个选项：
% ./configure --enable-storeio=aufs,ufs

严格讲，你不必在 storeio 模块列表中指定 ufs。然而，假如你以后不喜欢 aufs，那么就需要指定 ufs，以便能重新使用稳定的 ufs 存储机制。
假如愿意，你也能使用—with-aio-threads=N 选项。假如你忽略它，squid 基于 aufs cache_dir 的数量，自动计算可使用的线程数量。表 8-1 显示了 1-6 个 cache 目录的默认线程数量。
Table 8-1. Default number of threads for up to six cache directories

cache_dirs	Threads
1	16
2	26
3	32
4	36
5	40
6	44

将 aufs 支持编译进 squid 后，你能在 squid.conf 文件里的 cache_dir 行后指定它：
cache_dir aufs /cache0 4096 16 256

在激活了 aufs 并启动 squid 后，请确认每件事仍能工作正常。可以运行 tail -f store.log 一会儿，以确认缓存目标被交换到磁盘。也可以运行 tail -f cache.log 并且观察任何新的错误或警告。

8.4.1 aufs 如何工作

Squid 通过调用 pthread_create()来创建大量的线程。所有线程在任何磁盘活动之上创建。这样，即使 squid 空闲，你也能见到所有的线程。
无论何时，squid 想执行某些磁盘 I/O 操作（例如打开文件读），它分配一对数据结构，并将 I/O 请求放进队列中。线程循环读取队列，取得 I/O 请求并执行它们。因为请求队列共享给所有线程，squid 使用独享锁来保证仅仅一个线程能在给定时间内更新队列。

I/O 操作阻塞线程直到它们被完成。然后，将操作状态放进一个完成队列里。作为完整的操作，squid 主进程周期性的检查完成队列。请求磁盘 I/O 的模块被通知操作已完成，并获取结果。

你可能已猜想到，aufs 在多 CPU 系统上优势更明显。唯一的锁操作发生在请求和结果队列。然而，所有其他的函数执行都是独立的。当主进程在一个 CPU 上执行时，其他的 CPU 处理实际

的 I/O 系统调用。

8.4.2 aufs 发行

线程的有趣特性是所有线程共享相同的资源，包括内存和文件描述符。例如，某个线程打开一个文件，文件描述符为 27，所有其他线程能以相同的文件描述符来访问该文件。可能你已知道，在初次管理 squid 时，文件描述符短缺是较普遍问题。Unix 内核典型的有两种文件描述符限制：进程级的限制和系统级的限制。你也许认为每个进程拥有 256 个文件描述符足够了（因为使用线程），然而并非如此。在这样的情况下，所有线程共享少量的文件描述符。请确认增加系统的进程文件描述符限制到 4096 或更高，特别在使用 aufs 时。

调整线程数量有点棘手。在某些情况下，可在 cache.log 里见到如下警告：

2003/09/29 13:42:47| squidaio_queue_request: WARNING - Disk I/O overloading

这意味着 squid 有大量的 I/O 操作请求充满队列，等待着可用的线程。你首先会想到增加线程数量，然而我建议，你该减少线程数量。

增加线程数量也会增加队列的大小。超过一定数量，它不会改进 aufs 的负载能力。它仅仅意味着更多的操作变成队列。太长的队列导致响应时间变长，这绝不是你想要的。

减少线程数量和队列大小，意味着 squid 检测负载条件更快。当某个 cache_dir 超载，它会从选择算法里移除掉（见 7.4 章）。然后，squid 选择其他的 cache_dir 或简单的不存储响应到磁盘。这可能是较好的解决方法。尽管命中率下降，响应时间却保持相对较低。

8.4.3 监视 aufs 操作

Cache 管理器菜单里的 Async IO Counters 选项，可以显示涉及到 aufs 的统计信息。它显示打开，关闭，读写，stat，和删除接受到的请求的数量。例如：

```
% squidclient mgr:squidaio_counts
...
ASYNC IO Counters:
Operation        # Requests
open             15318822
close            15318813
cancel           15318813
write                   0
read             19237139
stat                    0
unlink            2484325
check_callback  311678364
queue                   0
```

取消(cancel)计数器正常情况下等同于关闭(close)计数器。这是因为 close 函数总是调用 cancel 函数，以确认任何未决的 I/O 操作被忽略。
写(write)计数器为 0，因为该版本的 squid 执行同步写操作，即使是 aufs。

check_callbak 计数器显示 squid 主进程对完成队列检查了多少次。

queue 值显示当前请求队列的长度。正常情况下，队列长度少于线程数量的 5 倍。假如你持续观察到队列长度大于这个值，说明 squid 配得有问题。增加更多的线程也许有帮助，但仅仅在特定范围内。

8.5 diskd 存储机制

diskd（disk 守护进程的短称）类似于 aufs，磁盘 I/O 被外部进程来执行。不同于 aufs 的是，diskd 不使用线程。代替的，它通过消息队列和共享内存来实现内部进程间通信。

消息队列是现代 Unix 操作系统的标准功能。许多年以前在 AT&T 的 Unix System V 的版本 1 上实现了它们。进程间的队列消息以较少的字节传递：32-40 字节。每个 diskd 进程使用一个队列来接受来自 squid 的请求，并使用另一个队列来传回请求。

8.5.1 diskd 如何工作

Squid 对每个 cache_dir 创建一个 diskd 进程。这不同于 aufs，aufs 对所有的 cache_dir 使用一个大的线程池。对每个 I/O 操作，squid 发送消息到相应的 diskd 进程。当该操作完成后，diskd 进程返回一个状态消息给 squid。squid 和 diskd 进程维护队列里的消息的顺序。这样，不必担心 I/O 会无序执行。

对读和写操作，squid 和 diskd 进程使用共享内存区域。两个进程能对同一内存区域进行读和写。例如，当 squid 产生读请求时，它告诉 diskd 进程在内存中何处放置数据。diskd 将内存位置传递给 read()系统调用，并且通过发送队列消息，通知 squid 该过程完成了。然后 squid 从共享内存区域访问最近的可读数据。

diskd 与 aufs 本质上都支持 squid 的无阻塞磁盘 I/O。当 diskd 进程在 I/O 操作上阻塞时，squid 有空去处理其他任务。在 diskd 进程能跟上负载情况下，这点确实工作良好。因为 squid 主进程现在能够去做更多工作，当然它有可能会加大 diskd 的负载。diskd 有两个功能来帮助解决这个问题。
首先，squid 等待 diskd 进程捕获是否队列超出了某种极限。默认值是 64 个排队消息。假如 diskd 进程获取的数值远大于此，squid 会休眠片刻，并等待 diskd 完成一些未决操作。这本质上让 squid 进入阻塞 I/O 模式。它也让更多的 CPU 时间对 diskd 进程可用。通过指定 cache_dir 行的 Q2 参数的值，你可以配置这个极限值：
cache_dir diskd /cache0 7000 16 256 Q2=50

第二，假如排队操作的数量抵达了另一个极限，squid 会停止要求 diskd 进程打开文件。这里

的默认值是 72 个消息。假如 squid 想打开一个磁盘文件读或写，但选中的 cache_dir 有太多的未完成操作，那么打开请求会失败。当打开文件读时，会导致 cache 丢失。当打开文件写时，会阻碍 squid 存储 cache 响应。这两种情况下用户仍能接受到有效响应。唯一实际的影响是 squid 的命中率下降。这个极限用 Q1 参数来配置：

```
cache_dir diskd /cache0 7000 16 256 Q1=60 Q2=50
```

注意在某些版本的 squid 中，Q1 和 Q2 参数混杂在默认的配置文件里。最佳选择是，Q1 应该大于 Q2。

8.5.2 编译和配置 diskd

为了使用 diskd，你必须在运行./configure 时，在--enable-storeio 列表后增加一项：

```
% ./configure --enable-storeio=ufs,diskd
```

diskd 看起来是可移植的，既然共享内存和消息队列在现代 Unix 系统上被广泛支持。然而，你可能需要调整与这两者相关的内核限制。内核典型的有如下可用参数：

MSGMNB
每个消息队列的最大字节限制。对 diskd 的实际限制是每个队列大约 100 个排队消息。squid 传送的消息是 32－40 字节，依赖于你的 CPU 体系。这样，MSGMNB 应该是 4000 或更多。为安全起见，我推荐设置到 8192。

MSGMNI
整个系统的最大数量的消息队列。squid 对每个 cache_dir 使用两个队列。假如你有 10 个磁盘，那就有 20 个队列。你也许该增加更多，因为其他应用程序也要使用消息队列。我推荐的值是 40。

MSGGSZ
消息片断的大小（字节）。大于该值的消息被分割成多个片断。我通常将这个值设为 64，以使 diskd 消息不被分割成多个片断。

MSGSEG
在单个队列里能存在的最大数量的消息片断。squid 正常情况下，限制队列的长度为 100 个排队消息。记住，在 64 位系统中，假如你没有增加 MSGSSZ 的值到 64，那么每个消息就会被分割成不止 1 个片断。为了安全起见，我推荐设置该值到 512。

MSGTQL
整个系统的最大数量的消息。至少是 cache_dir 数量的 100 倍。在 10 个 cache 目录情况下，我推荐设置到 2048。

MSGMAX
单个消息的最大 size。对 Squid 来说，64 字节足够了。然而，你系统中的其他应用程序可能要用到更大的消息。在某些操作系统例如 BSD 中，你不必设置这个。BSD 自动设置它为 MSGSSZ * MSGSEG。其他操作系统中，你也许需要改变这个参数的默认值，你可以设置它与 MSGMNB

相同。

SHMSEG
每个进程的最大数量的共享内存片断。squid 对每个 cache_dir 使用 1 个共享内存标签。我推荐设置到 16 或更高。

SHMMNI
共享内存片断数量的系统级的限制。大多数情况下，值为 40 足够了。

SHMMAX
单个共享内存片断的最大 size。默认的，squid 对每个片断使用大约 409600 字节。
为安全起见，我推荐设置到 2MB，或 2097152。

SHMALL
可分配的共享内存数量的系统级限制。在某些系统上，SHMALL 可能表示成页数量，而不是字节数量。在 10 个 cache_dir 的系统上，设置该值到 16MB（4096 页）足够了，并有足够的保留给其他应用程序。

在 BSD 上配置消息队列，增加下列选项到内核配置文件里：

```
# System V message queues and tunable parameters
options         SYSVMSG         # include support for message queues
options         MSGMNB=8192      # max characters per message queue
options         MSGMNI=40       # max number of message queue identifiers
options         MSGSEG=512       # max number of message segments per queue
options         MSGSSZ=64        # size of a message segment MUST be power of
2
options         MSGTQL=2048      # max number of messages in the system
options         SYSVSHM
options         SHMSEG=16       # max shared mem segments per process
options         SHMMNI=32       # max shared mem segments in the system
options         SHMMAX=2097152  # max size of a shared mem segment
options         SHMALL=4096      # max size of all shared memory (pages)
```

在 Linux 上配置消息队列，增加下列行到/etc/sysctl.conf：

```
kernel.msgmnb=8192
kernel.msgmni=40
kernel.msgmax=8192
kernel.shmall=2097152
kernel.shmmni=32
kernel.shmmax=16777216
```

另外，假如你需要更多的控制，可以手工编辑内核资源文件中的 include/linux/msg.h 和

include/linux/shm.h。

在 Solaris 上，增加下列行到/etc/system，然后重启：

```
set msgsys:msginfo_msgmax=8192
set msgsys:msginfo_msgmnb=8192
set msgsys:msginfo_msgmni=40
set msgsys:msginfo_msgssz=64
set msgsys:msginfo_msgtql=2048
set shmsys:shminfo_shmmax=2097152
set shmsys:shminfo_shmmni=32
set shmsys:shminfo_shmseg=16
```

在 Digital Unix(TRU64)上，可以增加相应行到 BSD 风格的内核配置文件中，见前面所叙。另外，你可使用 sysconfig 命令。首先，创建如下的 ipc.stanza 文件：

```
ipc:
        msg-max = 2048
        msg-mni = 40
        msg-tql = 2048
        msg-mnb = 8192
        shm-seg = 16
        shm-mni = 32
        shm-max = 2097152
        shm-max = 4096
```

然后，运行这个命令并重启：

```
# sysconfigdb -a -f ipc.stanza
```

一旦你在操作系统中配置了消息队列和共享内存，就可以在 squid.conf 里增加如下的 cache_dir 行：

```
cache_dir diskd /cache0 7000 16 256 Q1=72 Q2=64
cache_dir diskd /cache1 7000 16 256 Q1=72 Q2=64
...
```

8.5.3 监视 diskd

监视 diskd 运行的最好方法是使用 cache 管理器。请求 diskd 页面，例如：

```
% squidclient mgr:diskd
...
sent_count: 755627
recv_count: 755627
```

```
max_away: 14
max_shmuse: 14
open_fail_queue_len: 0
block_queue_len: 0

            OPS SUCCESS     FAIL
   open    51534   51530        4
create    67232   67232        0
  close  118762  118762        0
unlink    56527   56526        1
   read    98157   98153        0
  write  363415  363415        0
```

请见 14.2.1.6 章关于该输出的详细描述。

8.6 coss 存储机制

循环目标存储机制（Cyclic Object Storage Scheme,coss）尝试为 squid 定制一个新的文件系统。在 ufs 基础的机制下，主要的性能瓶颈来自频繁的 open()和 unlink()系统调用。因为每个 cache 响应都存储在独立的磁盘文件里，squid 总是在打开，关闭，和删除文件。

与之相反的是，coss 使用 1 个大文件来存储所有响应。在这种情形下，它是特定供 squid 使用的，小的定制文件系统。coss 实现许多底层文件系统的正常功能，例如给新数据分配空间，记忆何处有自由空间等。

不幸的是，coss 仍没开发完善。coss 的开发在过去数年里进展缓慢。虽然如此，基于有人喜欢冒险的事实，我还是在这里描述它。

8.6.1 coss 如何工作

在磁盘上，每个 coss cache_dir 是一个大文件。文件大小一直增加，直到抵达它的大小上限。这样，squid 从文件的开头处开始，覆盖掉任何存储在这里的数据。然后，新的目标总是存储在该文件的末尾处。

squid 实际上并不立刻写新的目标数据到磁盘上。代替的，数据被拷贝进 1MB 的内存缓冲区，叫做 stripe。在 stripe 变满后，它被写往磁盘。coss 使用异步写操作，以便 squid 主进程不会在磁盘 I/O 上阻塞。

象其他文件系统一样，coss 也使用块大小概念。在 7.1.4 章里，我谈到了文件号码。每个 cache 目标有一个文件号码，以便 squid 用于定位磁盘中的数据。对 coss 来说，文件号码与块号码一样。例如，某个 cache 目标，其交换文件号码等于 112，那它在 coss 文件系统中就从第 112 块开始。因此 coss 不分配文件号码。某些文件号码不可用，因为 cache 目标通常在 coss 文件

里占用了不止一个块。

coss 块大小在 cache_dir 选项中配置。因为 squid 的文件号码仅仅 24 位，块大小决定了 coss 缓存目录的最大 size：size = 块大小 x (2 的 24 次方)。例如，对 512 字节的块大小，你能在 coss cache_dir 中存储 8GB 数据。

coss 不执行任何 squid 正常的 cache 置换算法（见 7.5 章）。代替的，cache 命中被"移动"到循环文件的末尾。这本质上是 LRU 算法。不幸的是，它确实意味着 cache 命中导致磁盘写操作，虽然是间接的。

在 coss 中，没必要去删除 cache 目标。squid 简单的忘记无用目标所分配的空间。当循环文件的终点再次抵达该空间时，它就被重新利用。

8.6.2 编译和配置 coss

为了使用 coss，你必须在运行./configure 时，在--enable-storeio 列表里增加它：

```
% ./configure --enable-storeio=ufs,coss ...
```

coss 缓存目录要求 max-size 选项。它的值必须少于 stripe 大小（默认 1MB，但可以用--enable-coss-membuf-size 选项来配置）。也请注意你必须忽略 L1 和 L2 的值，它们被 ufs 基础的文件系统使用。如下是示例：

```
cache_dir coss /cache0/coss 7000 max-size=1000000
cache_dir coss /cache1/coss 7000 max-size=1000000
cache_dir coss /cache2/coss 7000 max-size=1000000
cache_dir coss /cache3/coss 7000 max-size=1000000
cache_dir coss /cache4/coss 7000 max-size=1000000
```

甚至，你可以使用 block-size 选项来改变默认的 coss 块大小。

```
cache_dir coss /cache0/coss 30000 max-size=1000000 block-size=2048
```

关于 coss 的棘手的事情是，cache_dir 目录参数（例如/cache0/coss）实际上不是目录，它是 squid 打开或创建的常规文件。所以你可以用裸设备作为 coss 文件。假如你错误的创建 coss 文件作为目录，你可以在 squid 启动时见到如下错误：

```
2003/09/29 18:51:42|  /usr/local/squid/var/cache: (21) Is a directory
FATAL: storeCossDirInit: Failed to open a coss file.
```

因为 cache_dir 参数不是目录，你必须使用 cache_swap_log 指令（见 13.6 章）。否则 squid 试图在 cache_dir 目录中创建 swap.state 文件。在该情形下，你可以见到这样的错误：

```
2003/09/29 18:53:38| /usr/local/squid/var/cache/coss/swap.state:
        (2) No such file or directory
```

FATAL: storeCossDirOpenSwapLog: Failed to open swap log.

coss 使用异步 I/O 以实现更好的性能。实际上，它使用 aio_read()和 aio_write()系统调用。这点也许并非在所有操作系统中可用。当前它们可用在 FreeBSD,Solaris,和 Linux 中。假如 coss 代码看起来编译正常，但你得到"Function not implemented"错误消息，那就必须在内核里激活这些系统调用。在 FreeBSD 上，必须在内核配置文件中有如下选项：
options VFS_AIO

8.6.3 coss 发行

coss 还是实验性的功能。没有充分证实源代码在日常使用中的稳定性。假如你想试验一下，请做好存储在 coss cache_dir 中的资料丢失的准备。 从另一面说，coss 的初步性能测试表现非常优秀。示例请见附录 D。

coss 没有很好的支持从磁盘重建 cache 数据。当你重启 squid 时，你也许会发现从 swap.state 文件读取数据失败，这样就丢失了所有的缓存数据。甚至，squid 在重启后，不能记忆它在循环文件里的位置。它总是从头开始。

coss 对目标置换采用非标准的方式。相对其他存储机制来说，这可能导致命中率更低。

某些操作系统在单个文件大于 2GB 时，会有问题。假如这样的事发生，你可以创建更多小的 coss 区域。例如：
cache_dir coss /cache0/coss0 1900 max-size=1000000 block-size=128
cache_dir coss /cache0/coss1 1900 max-size=1000000 block-size=128
cache_dir coss /cache0/coss2 1900 max-size=1000000 block-size=128
cache_dir coss /cache0/coss3 1900 max-size=1000000 block-size=128

使用裸磁盘设备（例如/dev/da0s1c）也不会工作得很好。理由之一是磁盘设备通常要求 I/O 发生在 512 个字节的块边界（译者注：也就是块设备访问）。另外直接的磁盘访问绕过了系统高速缓存，可能会降低性能。然而，今天的许多磁盘驱动器，已经内建了高速缓存。

8.7 null 存储机制

Squid 有第 5 种存储机制叫做 null。就像名字暗示的一样，这是最不健壮的机制。写往 null cache_dir 的文件实际上不被写往磁盘。
大多数人没有任何理由要使用 null 存储系统。当你想完全禁止 squid 的磁盘缓存时，null 才有用。你不能简单的从 squid.conf 文件里删除所有 cache_dir 行，因为这样的话 squid 会增加默认的 ufs cache_dir。null 存储系统有些时候在测试 squid，和压力测试时有用。既然文件系统是典型的性能瓶颈，使用 null 存储机制能获取基于当前硬件的 squid 的性能上限。

为了使用该机制，在运行./configure 时，你必须首先在--enable-storeio 列表里指定它：
% ./configure --enable-storeio=ufs,null ...

然后在 squid.conf 里创建 cache_dir 类型为 null:
cache_dir /tmp null

也许看起来有点奇怪，你必须指定目录给 null 存储机制。squid 使用目录名字作为 cache_dir 标识符。例如，你能在 cache 管理器的输出里见到它。

8.8 哪种最适合我？

Squid 的存储机制选择看起来有点混乱和迷惑。aufs 比 diskd 更好？我的系统支持 aufs 或 coss 吗？假如我使用新的机制，会丢失数据吗？可否混合使用存储机制？

首先，假如 Squid 轻度使用（就是说每秒的请求数少于 5 个），默认的 ufs 存储机制足够了。在这样的低请求率中，使用其他存储机制，你不会观察到明显的性能改进。

假如你想决定何种机制值得一试，那你的操作系统可能是个决定因素。例如，aufs 在 Linux 和 Solaris 上运行良好，但看起来在其他系统中有问题。另外，coss 代码所用到的函数，当前不可用在某些操作系统中（例如 NetBSD）。

从我的观点看来，高性能的存储机制在系统崩溃事件中，更易受数据丢失的影响。这就是追求最好性能的权衡点。然而对大多数人来说，cache 数据相对价值较低。假如 squid 的缓存因为系统崩溃而破坏掉，你会发现这很容易，只需简单的 newfs 磁盘分区，让 cache 重新填满即可。如果你觉得替换 Squid 的缓存内容较困难或代价很大，你就应该使用低速的，但可信的文件系统和存储机制。

近期的 Squid 允许你对每个 cache_dir 使用不同的文件系统和存储机制。然而实际上，这种做法是少见的。假如所有的 cache_dir 使用相同的 size 和相同的存储机制，可能冲突更少。

第 9 章 Cache 拦截

Cache 拦截是让传输流向 Squid 的流行技术，它不用配置任何客户端。你可以配置路由器或交换机将 HTTP 连接转发到 squid 运行的主机。squid 运行的操作系统被配置成接受外部数据包，并将其递交给 squid 进程。为了让 HTTP 拦截生效，你必须配置 3 个独立的因素：网络设备，squid 运行的操作系统，和 squid 自身。
<ww-01>;

（译者注：Cache 拦截实际上指的是 Squid 的透明代理）

9.1 它如何工作？

Cache 拦截包含了某些网络欺骗，它对理解在客户端和 Squid 之间的会话有用。我使用图 9-1 和如下的 tcpdump 示例输出，来解释当数据包通过网络时，如何被拦截。

1.用户代理(user-agent)想请求某个资源，它对原始服务器发起 index.html 请求，例如：www.oreilly.com。它需要原始服务器的 IP 地址，所以先发起一个 DNS 请求：

Packet 1
TIME: 19:54:41.317310
UDP: 206.168.0.3.2459 ->; 206.168.0.2.53
DATA: .d..........www.oreilly.com.....

Packet 2
TIME: 19:54:41.317707 (0.000397)
UDP: 206.168.0.2.53 ->; 206.168.0.3.2459
DATA: .d..........www.oreilly.com.............PR.....%........PR.
....$........PR...ns1.sonic.net.........PR...ns2.Q........PR
...ns...M..............h.............!.z.......b......

2.现在有了 IP 地址，用户代理初始化到原始服务器 80 端口的 TCP 连接：

Packet 3
TIME: 19:54:41.320652 (0.002945)
TCP: 206.168.0.3.3897 ->; 208.201.239.37.80 Syn
DATA: <No data>;

3.路由器或交换机注意到目的地址是 80 端口的 TCP SYN 包。下一步会发生什么依赖于特定的拦截技术。在 4 层交换和路由策略上，网络设备简单的将 TCP 包转发到 Squid 的数据链路地址。当 squid 直接挂在网络设备上时，就这样工作。对 WCCP 来说，路由器封装 TCP 包为 GRE 包。因为 GRE 包有它自己的 IP 地址，它可能被通过多个子网进行路由。换句话说，WCCP 不要求 squid 直接挂在路由器上。

4.Squid 主机的操作系统接受到拦截包。对 4 层交换来说，TCP/IP 包并没有改变。

假如包使用了 GRE 封装，主机会剥离外部的 IP 和 GRE 头部，并将原始的 TCP/IP 包放在输入队列里。

注意 squid 主机接受到的包是针对外部地址的（原始服务器的）。正常情况下，这个包不匹配任何本地地址，它会被丢弃。为了让主机接受外部数据包，你必须在大多数操作系统上激活 IP 转发。

5.客户端的 TCP/IP 包被包过滤代码处理。数据包必须匹配某个规则，该规则指示内核转交这个包给 squid。如果没有这样的规则，内核简单的将包按照它自己的方式转发给原始服务器，这不是你想要的。

注意 SYN 包的目的端口是 80，但 squid 可能侦听在不同的端口，例如 3128。包过滤规则允

许你改变端口号。你不必让 squid 侦听在 80 端口。通过 tcpdump，你能见到这步，因为转发的包不会再次通过网络接口代码。

即使 squid 侦听在 80 端口，包过滤器的重定向规则仍是必要的。可以让 squid 不在这些端口上接受拦截包。重定向规则有点神奇，它转交外部数据包给 squid。

6.Squid 接受到新连接的通知，它接受这个连接。内核发送 SYN/ACK 包返回给客户端：

```
Packet 4
TIME:    19:54:41.320735 (0.000083)
TCP:     208.201.239.37.80 ->; 206.168.0.3.3897 SynAck
DATA:    <No data>;
```

就象你见到的一样，源地址是原始服务器，尽管这个包不会抵达原始服务器。操作系统只是简单的将源地址和目的地址交换一下，并将它放进响应数据包里。

7.用户代理接受到 SYN/ACK 包，建立起完整的 TCP 连接。用户代理现在相信它是连接到原始服务器，所以它发送 HTTP 请求：

```
Packet 5
TIME:    19:54:41.323080 (0.002345)
TCP:     206.168.0.3.3897 ->; 208.201.239.37.80 Ack
DATA:    <No data>;
------------------------------------------------------------------------
Packet 6
TIME:    19:54:41.323482 (0.000402)
TCP:     206.168.0.3.3897 ->; 208.201.239.37.80 AckPsh
DATA:    GET / HTTP/1.0
         User-Agent: Wget/1.8.2
         Host: www.oreilly.com
         Accept: */*
         Connection: Keep-Alive
```

8.Squid 接受 HTTP 请求。它使用 HTTP Host 头部来转换局部 URL 为完整的 URL。在这种情形下，可在 access.log 文件里见到 http://www.oreilly.com。

9.从这点开始，squid 正常的处理请求。一般 cache 命中会立刻返回。cache 丢失会转发到原始服务器。

10.最后，是 squid 从原始服务器接受到的响应：

```
Packet 8
TIME:    19:54:41.448391 (0.030030)
TCP:     208.201.239.37.80 ->; 206.168.0.3.3897 AckPsh
```

```
DATA:   HTTP/1.0 200 OK
        Date: Mon, 29 Sep 2003 01:54:41 GMT
        Server: Apache/1.3.26 (Unix) PHP/4.2.1 mod_gzip/1.3.19.1a mo
        d_perl/1.27
        P3P: policyref="http://www.oreillynet.com/w3c/p3p.xml",CP="C
        AO DSP COR CURa ADMa DEVa TAIa PSAa PSDa IVAa IVDa CONo OUR
        DELa PUBi OTRa IND PHY ONL UNI PUR COM NAV INT DEM CNT STA P
        RE"
        Last-Modified: Sun, 28 Sep 2003 23:54:44 GMT
        ETag: "1b76bf-b910-3ede86c4"
        Accept-Ranges: bytes
        Content-Length: 47376
        Content-Type: text/html
        X-Cache: MISS from www.oreilly.com
        X-Cache: MISS from 10.0.0.1
        Connection: keep-alive
```

不应该让交换机或路由器来拦截 squid 到原始服务器的连接。假如这种情况发生，squid 结束与自己的会话，并且不能满足任何 cache 丢失。防止这类转发死循环的最好方法是，确认用户和 squid 连接到交换机或路由器的独立接口。无论何时，应该在指定接口上应用拦截规则。最明显的，不该在 squid 使用的接口上激活拦截。

9.2 为何要（或不要）拦截？

许多单位发现，cache 拦截很有用，因为他们不能，或不愿意配置所有用户的 web 浏览器。相对于配置成百上千台工作站来说，在单个交换机或路由器上做一点网络欺骗更容易。从我们面临的许多选择来看，cache 拦截确实有好也有坏。它可能让你的生活更容易，但也许会更难。

Cache 拦截的最明显的贡献是，所有 HTTP 请求通过 squid 自动离开你的网络。你不必担心配置任何浏览器，用户可能在浏览器上禁止他们的代理设置。cache 拦截让网络管理员完全控制 HTTP 会话。你可以改变，增加，或删除 squid 的缓存，而不会显著影响你的用户上网冲浪。

关于 HTTP 拦截的主要不利点就是该技术违背了 TCP/IP 的标准。这些协议要求路由器或交换机转发 TCP/IP 包到目的 IP 地址里指定的主机。然而转发包到 cache 代理破坏了这些规则。代理伪装身份接受转交过来的连接。用户代理被欺骗了，以为它们在与真正的 web 服务器会话。

这样的混乱导致在老版本的 Microsoft IE 浏览器中产生严重问题。浏览器的 Reload 按钮是刷新 HTML 页面的最容易的方法。当浏览器被配置成使用 cache 代理时，reload 请求包含了一个 Cache-Control:no-cache 头部，它强迫产生 cache 丢失（或 cache 确认），并确保响应是最近更新的。假如没有明确配置使用代理，浏览器会忽略该头部。当使用 cache 拦截时，浏览器认为它在连接到原始服务器，因此没必要发送该头部。在这种情形下，squid 不会告知用户的 Reload 按钮，也许不会验证 cache 响应。squid 的 ie_refresh 提供了解决此 bug 的局部解决方法（见附录 A）。Microsoft 已经在其 IE 5.5 SP1 中解决了这个问题。

因为类似的理由，你不能结合 cache 拦截使用 HTTP 代理验证。因为客户端不知道这个代理，它不会发送必要的 Proxy-Authorization 头部。另外，407（代理验证请求）响应代码也不恰当，因为响应看起来象来自原始服务器，原始服务器从来不会发送如此响应。

也不能在 cache 拦截中使用 RFC 1413 ident 查询（见 6.1.2.11 章节）。Squid 不能对必要的 IP 地址建立新的 TCP Socket 连接。操作系统在转发拦截连接到 squid 时，它执行欺骗。然后，当 squid 希望 bind 新的 TCP Socket 到外部 IP 地址时，它不能执行欺骗。它想 bind 的地址实际上并非真正本地的，所以 bind 系统调用失败。

cache 拦截也与设计成阻止地址欺骗的 IP 过滤冲突（见 RFC 2267:Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source AddressSpoofing）。考虑如图 9-2 显示的网络。路由器有 2 个 LAN 接口:lan0 和 lan1。网络管理员在路由器上使用包过滤器，以确保没有内部主机传送假冒源地址的数据包。路由器只会转发源地址对应相连网络的数据包。包过滤规则也许看起来如下：

```
# lan0
allow ip from 172.16.1.0/24 to any via lan0
deny ip from any to any via lan0

# lan1
allow ip from 10.0.0.0/16 to any via lan1
deny ip from any to any via lan1
```

现在看看，当路由器和 lan1 中的 squid 主机配置成拦截来自 lan0 中的 HTTP 连接后，会发生什么。Squid 装扮成原始服务器，这意味着从 squid 到用户的响应 TCP 包欺骗了源地址。lan0 过滤规则导致路由器拒绝这些包。为了让 cache 拦截生效，网络管理员须移除 lan0 规则。这样就让网络有漏洞，从而易遭拒绝服务攻击。

我在先前的章节里描述过，客户端在打开连接之前必须先进行 DNS 查询。在某些防火墙环境中，这样做可能有问题。你想进行 HTTP 拦截的主机必须能够查询 DNS。如果客户端了解自己正使用代理（因为手工配置或代理自动配置），它通常就不去解析主机名。代替的，它简单的将完整 URL 转发给 squid，由 squid 来查询原始服务器的 IP 地址。

另一个小问题是，squid 接受任意目的 IP 地址的连接。例如，某个 web 站点当机了，但它仍然有 DNS 记录存在。squid 伪装这个站点接受 TCP 连接。客户端会认为该站点仍然在运行，因为连接有效。当 squid 连接到原始服务器失败时，它强迫返回错误消息。

万一形势不清，HTTP 拦截在初次使用时有些棘手或困难。许多不同的组件必须组合工作，并且要配置正确。甚至，从内存中恢复整个配置也很困难。我强烈建议你在将其应用于生产环境之前，先建立测试环境。一旦你让它正常运行，请记录每一步细节。

9.3 网络设备

现在你了解了 cache 拦截的相关细节，让我们看看如何实际让它工作。我们先配置网络设备，它们用来拦截 HTTP 连接。

9.3.1 内置 Squid

在该配置中，你无需交换或网络路由设备来拦截 HTTP 连接。代替的，squid 运行的 Unix 系统，也就是路由器（或网桥），请见图 9-2。
<ww-02>;

该配置本质上跳过了 9.1 章的头三步。squid 主机充当网络路由器，它接受 HTTP 连接包。假如你采用此方法，请直接跳到 9.4 章。

9.3.2 四层交换

许多单位使用四层交换机来支持 HTTP 拦截。这些产品提供更多的功能，例如健壮性检测和负载均衡。我在这里仅仅讲讲拦截。关于健壮性检测和负载均衡的信息，请见 O'Reilly's Server Load Balancing and Load Balancing Servers, Firewalls, and Caches (John Wiley & Sons). 下面的章节包含了许多产品和技术的示例配置。

9.3.2.1 Alteon/Nortel

下面的配置来自 ACEswitch 180 和 Alteon's WebOS 8.0.21。网络设置请见图 9-4。
<ww-03>;

客户端连接到端口 1，通过端口 2 连接到因特网，squid 运行在端口 3。下面的行是交换机的 /cfg/dump 命令的输出。你无须敲入所有这些行。甚至，在 Alteon 的新版软件里，某些命令可能改变了。注意 Alteon 把这个功能叫做 Web Cache 重定向（WCR）。如下是处理步骤：

1.首先，你必须分配给 Alteon 交换机一个 IP 地址。这是必要的，以便交换机能检查 squid 的存活状态。

```
/cfg/ip/if 1
        ena
        addr 172.16.102.1
        mask 255.255.255.0
        broad 172.16.102.255
```

2.Alteon 的 WCR 属于服务负载均衡(SLB)配置。所以，必须使用如下命令在交换机上激活 SLB 功能：

```
/cfg/slb
        on
```

3. 现在，用 squid 的 IP 地址定义 real server:

```
/cfg/slb/real 1
        ena
        rip 172.16.102.66
```

4. 必须定义一个组，并分配给 real server 一个组号：

```
/cfg/slb/group 1
        health tcp
        add 1
```

5. 下一步定义 2 个过滤规则。第 1 条规则匹配 HTTP 连接（目的端口是 80 的 TCP 包），并重定向它们到组 1 里的 server。第 2 条规则匹配所有其他数据包，并正常转发它们。

```
/cfg/slb/filt 1
        ena
        action redir
        sip any
        smask 0.0.0.0
        dip any
        dmask 0.0.0.0
        proto tcp
        sport any
        dport http
        group 1
        rport 0

/cfg/slb/filt 224
        ena
        action allow
        sip any
        smask 0.0.0.0
        dip any
        dmask 0.0.0.0
        proto any
```

6. 最后一步是给 SLB 配置指定的交换端口。在端口 1 上，处理客户端连接（这也是客户端连接的端口），并增加 2 条过滤规则。在端口 2 上，仅须配置它正常服务（例如，向上连接到 Internet）：

```
cfg/slb/port 1
        client ena
        filt ena
        add 1
        add 224

/cfg/slb/port 2
        server ena
```

为了验证 HTTP 拦截配置正确并工作良好，你可以使用/stats/slb 和/info/slb 菜单里的命令。/info/slb/dump 是快速有效的查看整个 SLB 配置的方法：

```
>;>; Main# /info/slb/dump

Real server state:
  1: 172.16.102.66, 00:c0:4f:23:d7:05, vlan 1, port 3, health 3, up

Virtual server state:

Redirect filter state:
  1: dport http, rport 0, group 1, health tcp, backup none
     real servers:
       1: 172.16.102.66, backup none, up

Port state:
  1: 0.0.0.0, client
     filt  enabled, filters: 1 224
  2: 0.0.0.0, server
     filt disabled, filters: empty
  3: 0.0.0.0
     filt disabled, filters: empty
```

在该输出里，注意到交换机显示 Squid 在端口 3 上可到达，并且运行正常。你也能见到过滤规则 1 应用到端口 1。在端口状态节里，端口 1 定义为客户端连接端口，端口 2 简单的标记为服务端口。

/stats/slb/real 命令显示 real server(squid)的有用统计：

```
>;>; Main# /stats/slb/real 1
------------------------------------------------------------------

Real server 1 stats:
Health check failures:                0
Current sessions:                    41
```

```
Total sessions:                  760
Highest sessions:                 55
Octets:                           0
```

大部分统计与任务（例如 TCP 连接）数量相关。假如再次运行该命令，总共的任务计数会增加。

最后，/stats/slb/group 命令显示几乎同样的信息：

```
>;>; Main# /stats/slb/group 1
-------------------------------------------------------------------

Real server group 1 stats:

                     Current       Total  Highest
Real IP address       Sessions    Sessions Sessions           Octets
---- -------------- -------- ---------- --------  ---------------
   1 172.16.102.66         65        2004        90                0
---- -------------- -------- ---------- --------  ---------------
                         65        2004        90                0
```

假如不止 1 个 real server 在组里，该输出会更有趣。

9.3.2.2 Foundry

下面的配置示例来自 ServerIron XL，运行的软件版本是 07.0.07T12。跟前面一样，客户端在端口 1，Internet 连接在端口 2，squid 运行在端口 3。然而，这样的配置少了点东西，因为这里可以激活 HTTP 全局拦截。Foundry 的 cache 拦截的名字叫做 Transparent Cache Switching（TCS）。请参考图 9-4。
<ww-04>;

首先请给交换机分配 1 个 IP 地址，以便执行健壮性检测：
ip address 172.16.102.1 255.255.255.0

Foundry 允许你在特定端口上激活或禁用 TCS。然而简单起见，这里全局激活它：
ip policy 1 cache tcp http global

在该行里，cache 是针对 TCS 功能的关键字。下 1 行定义 web cache，我定义其名字为 squid1，并且告诉交换机它的 IP 地址：
server cache-name squid1 172.16.102.66

最后的步骤是将 web cache 加进 cache 组里：
server cache-group 1
cache-name squid1

假如在转发连接时有问题，请参阅 show cache-group 命令的输出：

ServerIron#show cache-group

Cache-group 1 has 1 members Admin-status = Enabled Active = 0

Hash_info: Dest_mask = 255.255.255.0 Src_mask = 0.0.0.0

Cache Server Name Admin-status Hash-distribution
squid1 6 3

HTTP Traffic From <->; to Web-Caches

Name: squid1 IP: 172.16.102.66 State: 6 Groups = 1

		Host->;Web-cache			Web-cache->;Host	
	State	CurConn	TotConn	Packets	Octets	Packets	Octets
Client	active	441	12390	188871	15976623	156962	154750098
Web-Server	active	193	11664	150722	151828731	175796	15853612
Total		634	24054	339593	167805354	332758	170603710

某些输出有些模糊，但通过重复该命令，并且观察计数器的增长，你能了解拦截是否在进行。

show server real 提供几乎同样的信息：

ServerIron#show server real squid1

Real Servers Info

Name : squid1 Mac-addr: 00c0.4f23.d705

IP:172.16.102.66 Range:1 State:Active Wt:1 Max-conn:1000000

Src-nat (cfg☹p)☹off☹ff) Dest-nat (cfg☹p)☹off☹ff)

squid1 is a TRANSPARENT CACHE in groups 1

Remote server : No Dynamic : No Server-resets:0

Mem:server: 02009eae Mem:mac: 045a3714

Port	State	Ms	CurConn	TotConn	Rx-pkts	Tx-pkts	Rx-octet	Tx-octet	Reas
----	-----	--	-------	-------	-------	-------	--------	--------	----

```
http     active    0  855     29557    379793    471713    373508204  39425322    0

default active    0  627     28335    425106    366016    38408994    368496301  0

Server   Total       1482     57892    804899    837729    411917198  407921623  0
```

最后，使用 show logging 命令来观察交换机是否显示 squid 正常或异常：

```
ServerIron#show logging
...

00d00h11m51s:N:L4 server 172.16.102.66 squid1 port 80 is up

00d00h11m49s:N:L4 server 172.16.102.66 squid1 port 80 is down

00d00h10m21s:N:L4 server 172.16.102.66 squid1 port 80 is up

00d00h10m21s:N:L4 server 172.16.102.66 squid1 is up
```

注意 ServerIron 认为服务运行在 80 端口。以后你会见到 squid 运行在 3128 端口的示例。包过滤规则实际上将包的目的地址从 80 改变为 3128。这导致一些与状态检测有关的有趣结果，我在 9.3.2.5 节里会讲到。

9.3.2.3 Extreme Networks

在该示例里，硬件是 Summit1i，软件版本是 6.1.3b11。再次将客户端分配在端口 1，Internet 在端口 2，squid 在端口 3。网络配置见图 9-5。
<ww-05>;

Extreme 交换机仅仅对在不同子网间进行路由的数据包进行 HTTP 连接的拦截。换句话说，如果你配置 Extreme 交换机使用二层模式（单一 VLAN 里），就不能将包转发给 squid。为了让 HTTP 拦截正常工作，必须给用户，Squid，和 Internet 配置不同的 VLAN。

```
configure Default delete port 1-8

create vlan Users
configure Users ip 172.16.102.1 255.255.255.192
configure Users add port 1

create vlan Internet
configure Internet ip 172.16.102.129 255.255.255.192
configure Internet add port 2
```

```
create vlan Squid
configure Squid ip 172.16.102.65 255.255.255.192
configure Squid add port 3
```

下一步是激活和配置交换机的路由：

```
enable ipforwarding
configure iproute add default 172.16.102.130
```

最后，配置交换机重定向 HTTP 连接到 Squid：

```
create flow-redirect http tcp destination any ip-port 80 source any
configure http add next-hop 172.16.102.66
```

9.3.2.4 Cisco Arrowpoint

下类配置基于我以前的测试笔记。然而，最近我没有使用这类型的交换机，不能确保如下命令仍然正确：

```
circuit VLAN1
  ip address 172.16.102.1 255.255.255.0

service pxy1
  type transparent-cache
  ip address 172.16.102.66
  port 80
  protocol tcp
  active

owner foo
  content bar
     add service pxy1
     protocol tcp
     port 80
     active
```

9.3.2.5 关于 HTTP 服务和健壮性检测的评论

在上面的示例里，路由器/交换机都直接转发包，不会改变目的 TCP 端口。在 9.4 章里用到的包过滤规则改变了目的端口。如果试图在同一主机上运行 HTTP 服务和 squid，那么就产生了一个有趣的问题。

为了在 3128 端口运行 Squid 的同时，还要在 80 端口运行 HTTP，包过滤配置必须有 1 条特殊的规则，它接受到本机 HTTP 服务的 TCP 连接。否则，连接会直接转交给 Squid。该规则易于建立。假如目的端口是 80，并且目的地址是服务器的，那么主机正常接受这个包。然而所有

的拦截包有外部目的地址，所以它们不会匹配该规则。

然而，当路由器/交换机进行 HTTP 健壮性检测时，它连接到服务器的 IP 地址。这样，健壮性检测的数据包匹配了上述规则，它不会转交给 Squid。路由器/交换机检测了错误的服务。假如 HTTP 服务 down 掉了，而 squid 还在运行，那健壮性检测就产生错误结果。

解决这个问题的一些选择是：
1.不要在 Squid 主机上运行 HTTP 服务；
2.增加 1 条特殊的包过滤规则，将来自路由器/交换机的状态检测的包转交给 squid；
3.配置路由器/交换机，改变目的端口为 3128；
4.禁止 4 层状态检测。

9.3.3 Cisco 策略路由

策略路由与 4 层交换并非不同。它在 Cisco 和其他公司的路由产品上执行。主要的区别是策略路由不包括任何健壮性检测。这样，假如 squid 超载或完全不可响应，路由器还会继续转发包到 squid，而不是将包路由到原始服务器。策略路由要求 squid 位于路由器直接相连的子网中。

在本示例里，使用了 Cisco 7204 路由器，运行 IOS Version 12.0(5)T。网络配置与前面的一样，见图 9-5。
<ww-05>;

首先的配置步骤是定义访问列表，匹配来自客户端的到 80 端口的数据包。必须确保 Squid 发起的到 80 端口的数据包不会被再次拦截。做到这点的方法之一是，定义 1 个特殊规则，拒绝来自 squid 的数据包，紧跟 1 条规则允许所有其他的数据包：

```
access-list 110 deny tcp host 172.16.102.66 any eq www
access-list 110 permit tcp any any eq www
```

另外，如果 Squid 和用户位于不同的子网，你可以仅仅允许来自用户所在网络的数据包：
```
access-list 110 permit tcp 10.102.0.0 0.0.255.255 any eq www
```

下一步是定义路由映射。在这里你告诉路由器转发拦截包到何处去：

```
route-map proxy-redirect permit 10
match ip address 110
set ip next-hop 172.16.102.66
```

这些命令表明，“假如 IP 地址匹配访问列表 110，就转发该包到 172.16.102.66”。在 route-map 行的数字 10 是一个序列号，假如你有多个路由映射的话。最后一步是应用路由映射到客户端连接的接口：

```
interface Ethernet0/0
```

ip policy route-map proxy-redirect

IOS 不对策略路由提供很多调试方法。然而，show route-map 命令足够可用：

```
router#show route-map proxy-redirect
route-map proxy-redirect, permit, sequence 10

  Match clauses:
    ip address (access-lists): 110
  Set clauses:
    ip next-hop 172.16.102.66
  Policy routing matches: 730 packets, 64649 bytes
```

9.3.4 Web Cache Coordination 协议

Cisco 对 4 层交换技术的响应叫做 Web Cache Coordination Protocol(WCCP).WCCP 在许多方面与典型的 4 层拦截不同。

首先，拦截包被封装在 GRE（路由封装类）里。这点允许数据包跨子网传输，也就意味着 squid 不必直接连在路由器上。因为数据包是封装的，squid 主机必须对其进行解包。并非所有的 Unix 系统有解开 GRE 包的代码。

第二个不同是，路由器如何决定将负载分摊到多个 cache 上。事实上，路由器不做这个决定，由 cache 来做。当路由器有一组支持 WCCP 的 cache 时，其中一个 cache 主动成为组的领导 。由领导 cache 来决定如何分摊负载和通知路由器。在路由器重定向然任何连接之前，这是一个额外的必要步骤。

因为 WCCP 使用 GRE，路由器可能强迫要求将来自 HTTP 请求的大 TCP 包分割成片断。幸运的是，这点不会经常发生，因为大部分 HTTP 请求比以太网 MTU size（1500 字节）要小。默认的 TCP 和 IP 包头部是 20 字节，这意味着以太网帧能实际携带 1460 字节的数据。GRE 封装在 GRE 头部增加了 20 字节，另外在第二个 IP 头部增加了 20 字节。这样来自客户端的正常的 1500 字节的 TCP/IP 包，在封装后变成了 1540 字节。这样数据包就太大而不能在单个以太网帧里传输，所以路由器将原始包分割成两个片断。

9.3.4.1 WCCPv1

该节的配置示例在运行 IOS Version 12.0(5)T 的 Cisco 7204 路由器上测试。网络配置跟图 9-5 同。

首先，在 IOS 配置中输入如下两行，激活路由器的 WCCP:

```
ip wccp version 1
ip wccp web-cache
```

接着，必须在单独的路由器接口上激活 WCCP。在 HTTP 包离开路由器的接口上激活 WCCP，也就是路由器连接到外部原始服务器或 Internet 网关的接口：

```
interface Ethernet0/1
ip address 172.16.102.129 255.255.255.192
ip wccp web-cache redirect out
```

请确认保存了配置改变。

你也许想使用访问列表来阻止某些 web 站点的拦截。可以使用访问列表来防止循环转发。例如：

```
! don't re-intercept connections coming from Squid:
access-list 112 deny   tcp host 172.16.102.66 any eq www

! don't intercept this broken web site
access-list 112 deny   tcp any 192.16.8.7 255.255.255.255 eq www

! allow other HTTP traffic
access-list 110 permit tcp any any eq www

ip wccp web-cache redirect-list 112
```

路由器不发送任何数据到 squid，直到 squid 宣称它自己是路由器。我在 9.5.1 节里解释如何配置 squid 的 WCCP。

9.3.4.2 WCCPv2

当前标准的 Squid 发布仅支持 WCCPv1。然而，可在 http://devel.squid-cache.org/找到 WCCPv2 的补丁。该代码仍是实验性的。

注意从路由器发送到 Squid 的 GRE 包，包含了额外的 4 个字节。WCCPv2 在 GRE 头部和封装的 IP 包之间，插入了一个重定向头部。也许需要修改内核代码来计算这个额外的头部。

9.3.4.3 调试

IOS 提供许多命令来监视和调试 WCCP。show ip wccp web-cache 命令提供一些基本的信息：

```
router#show ip wccp web-cache

Global WCCP information:
    Router information:
```

```
        Router Identifier:                  172.16.102.129
        Protocol Version:                   1.0
    Service Identifier: web-cache
        Number of Cache Engines:              1
        Number of routers:                    1
        Total Packets Redirected:             1424
        Redirect access-list:                 -none-
        Total Packets Denied Redirect:        0
        Total Packets Unassigned:             0
        Group access-list:                    -none-
        Total Messages Denied to Group:       0
        Total Authentication failures:        0
```

欲了解更多细节，在前叙命令后加一个 detail 单词：

```
router#show ip wccp web-cache detail

WCCP Cache-Engine information:

        IP Address:             172.16.102.66
        Protocol Version:       0.4
        State:                  Usable
        Initial Hash Info:      00000000000000000000000000000000
                                00000000000000000000000000000000
        Assigned Hash Info:     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
                                FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        Hash Allotment:         256 (100.00%)
        Packets Redirected:     1424
        Connect Time:           00:17:40
```

这里可以看到 squid 的 IP 地址和状态。假如不止一个 cache 与路由器会话，那么 hash 分配信息看起来不同。大多数情况下，每个 cache 接受到相等的 hash 值。

注意第二条命令输出的协议版本值，与第一条命令的不一样。不幸的是，赋予了版本号太多的意义。show ip wccp web-cache 命令看起来报告 WCCP 协议的主版本号（例如 1 或 2），然而 show ip wccp web-cache detail 的版本号看起来匹配 Squid 的 wccp_version 指令的值。

9.4 操作系统配置

为了让 cache 拦截正常工作，必须在操作系统中激活某些网络功能。首先，必须激活 IP 包转发 。这就允许操作系统接受目的地址是外部 IP 的数据包。接着，必须激活和配置内核中的相关代码，以重定向外部包到 Squid。

9.4.1 Linux

本节的指导适合 2.4 系列 Linux 内核。我使用 RedHat Linux7.2（内核是 2.4.7-10）。假如你使用的版本不同，那可能不能运行。建议搜索下 Squid 的 FAQ 或其他地方，找到关于内核的更新的或历史的信息。

在我测试 iptables 过程中，不必激活 IP 转发。然而，你也可以试试在一开始就激活它，并在一切运行良好后，看看能否禁掉它。激活包转发的最好的方法是在/etc/sysctl.conf 文件里增加如下行：

```
net.ipv4.ip_forward = 1
```

一般来说，在 HTTP 拦截生效前，不必编译新内核。假如你不知道如何配置和创建新 Linux 内核，请参阅 O'Reilly's Running Linux by Matt Welsh, Matthias Kalle Dalheimer, and Lar Kaufman。当你配置内核时，请确认如下选项被激活：

```
o  General setup
     Networking support (CONFIG_NET=y)
     Sysctl support (CONFIG_SYSCTL=y)

o  Networking options
     Network packet filtering (CONFIG_NETFILTER=y)
     TCP/IP networking (CONFIG_INET=y)
     Netfilter Configuration
       Connection tracking (CONFIG_IP_NF_CONNTRACK=y)
       IP tables support (CONFIG_IP_NF_IPTABLES=y)
       Full NAT (CONFIG_IP_NF_NAT=y)
       REDIRECT target support (CONFIG_IP_NF_TARGET_REDIRECT=y)

o  File systems
     /proc filesystem support (CONFIG_PROC_FS=y)
```

另外，请确认该选项没被激活：

```
o Networking options

     Fast switching (CONFIG_NET_FASTROUTE=n)
```

重定向外部数据包到 squid 的代码是 Netfilter 软件的一部分。如下是发送 HTTP 拦截连接到 squid 的规则：

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port
3128
```

Linux 内核维护许多不同的 tables。-t nat 选项表明我们正在修改 NAT（网络地址转换）表。本质上，我们使用 iptables 将原始服务器的 TCP/IP 地址转换为 squid 的本地 TCP/IP 地址。

每个 iptables 表有许多链。-A PREROUTING 表明我们增加了一条规则到内建的链叫做 PREROUTING。PREROUTING 链仅对从外部网络进入系统的数据包有效。

接下来的三个选项决定哪个包匹配该规则。-i eth0 选项限制规则仅对 eth0 接口上接受的数据包有效。-p tcp 选项指定 TCP 包，--dport 80 指定包的目的端口是 80。假如这三个条件都是 true，那么包匹配该规则。

-j REDIRECT 选项表明对匹配规则的包，采取何种动作。REDIRECT 是内建的动作名，它导致 iptables 改变包的目的地址为 127.0.0.1。--to-port 3128 选项也指示 iptables 改变目的 TCP 端口为 3128。

假如你在 squid 主机上运行 HTTP 服务（例如 Apache），就必须增加另外的 iptables 规则。该必要规则允许连接到 HTTP 服务。否则，REDIRECT 规则导致 iptables 转发连接到 squid 的 3128 端口。可以使用-I 选项在列表顶部插入一条新规则：

iptables -t nat -I PREROUTING -i eth0 -p tcp -d 172.16.102.66 --dport 80 -j ACCEPT

一旦确认所有 iptables 规则工作正确，记得运行如下命令来保存配置：

/sbin/service iptables save

将当前规则保存到/etc/sysconfig/iptables，当系统重启时，这些规则自动载入。

9.4.1.1 Linux 和 WCCP

2.4 版本的 Linux 内核自带 1 个 GRE 伪装接口。然而，它不能解码 WCCP 任务里封装的 GRE 包。问题看起来在于路由器设置了 WCCP/GRE 包的协议类型域为 0x883E。Linux 的 GRE 驱动不知道如何处理这类型包，因为它不了解 0x883E 类型的协议。

可以试试给 Linux 打 GRE 模块的补丁，以便它能在 WCCP 下工作。Squid FAQ 包含了对这个补丁的链接。然而，假如使用 WCCP 指定的 Linux 模块，事情会容易些。可以在这里找到它：http://www.squid-cache.org/WCCP-support/Linux/ip_wccp.c

必须编译 ip_wccp.c 文件为可装载内核模块。有点棘手的是，依赖于内核版本的不同，编译选项可能不同。你可以进入内核源代码目录，敲入 make modules 并观察编译器命令的滚动。然后拷贝这些命令中的一个，然后改变最后一个参数为 ip_wccp.c。如下是我在 2.4.7-10 Linux 内核中使用的命令：

% gcc -Wall -D_ _KERNEL_ _ -I/usr/src/linux-2.4.7-10/include \

```
  -DMODULE -DMODVERSIONS -DEXPORT_SYMBAB \
  -include /usr/src/linux-2.4.7-10/include/linux/modversions.h \
  -O2 -c ip_wccp.c
```

gcc 命令在当前目录会生成 ip_wccp.o 文件。下一步使用 insmod 命令，装载模块到内核中：

```
# insmod ip_wccp.o
```

注意 ip_wccp 模块接受来自任何源地址的 GRE/WCCP 包。换句话说，恶意用户可能发送数据到 squid cache。假如使用该模块，应该安装一条 iptables 规则，拒绝外部的 GRE 包。例如:

```
# iptables -A INPUT -p gre -s 172.16.102.65 -j ACCEPT
# iptables -A INPUT -p gre -j DROP
```

不要忘记敲入/sbin/service iptables save 命令来保存配置。

9.4.2 FreeBSD

本节的例子基于 FreeBSD-4.8,并可以在任何 FreeBSD-4 和 5 系列的后续版本上运行。

要激活 IP 包转发，在/etc/sysctl.conf 中增加如下行：

```
net.inet.ip.forwarding=1
```

需要在内核中激活 2 个特殊选项。假如你不知道如何编译内核，参见 FreeBSD Handbook 第9章(http://www.freebsd.org/handbook/index.html). 编辑内核配置文件，确保有如下行：

```
options         IPFIREWALL
options         IPFIREWALL_FORWARD
```

假如 squid 主机位于无人照看的机房中,我也推荐使用 IPFIREWALL_DEFAULT_TO_ACCEPT 选项。假如你被防火墙的规则困扰，仍然可以登陆系统中。

ipfw 命令告诉内核重定向拦截连接到 squid:

```
/sbin/ipfw add allow tcp from 172.16.102.66 to any out
/sbin/ipfw add allow tcp from any 80 to any out
/sbin/ipfw add fwd 127.0.0.1,3128 tcp from any to any 80 in
/sbin/ipfw add allow tcp from any 80 to 172.16.102.66 in
```

第一条规则匹配 squid 主机发起的数据包。它确保外出的 TCP 连接不会被重新定向回 squid。第二条规则匹配 squid 响应客户端的数据包。我在这里列出它，因为随后会有另外的 ipfw 规则，这些规则会拒绝这些包。第三条规则实际重定向进来的连接到 squid。第四条规则匹配从原始服务器返回 squid 的数据包。这里又一次假设随后会有相应的拒绝规则。

假如在 squid 主机上运行 HTTP 服务，就必须增加另外的规则，它放过，而不是重定向，目的地址是原始服务器的 TCP 包。下列规则在 fwd 规则之前：

/sbin/ipfw add allow tcp from any to 172.16.102.66 80 in

FreeBSD 典型的将 ipfw 规则存储在/etc/rc.firewall 里。一旦你确认规则设置正确，记得保存它们。将如下行加入/etc/rc.local 文件，让 FreeBSD 在启动时自动运行/etc/rc.firewall 脚本 。

firewall_enable="YES"

9.4.2.1 FreeBSD 和 WCCP

FreeBSD 版本 4.8 和后续版本内建了对 GRE 和 WCCP 的支持。早期的版本需要补丁，你可以在这里找到：http://www.squid-cache.org/WCCP-support/FreeBSD/ ． 内建代码的性能非常好，它是真正的内核组织编写的。可能也需要编译支持 GRE 的新内核。将如下行加入内核配置文件里：

pseudo-device　gre

对 Freebsd-5，使用 device 代替了 pseudo-device。当然，你也需要前面章节里提到的 FIREWALL 选项。

在安装和重启了新内核后，必须配置 GRE 通道来接受来自路由器的 GRE 包。例如：

```
# ifconfig gre0 create
# ifconfig gre0 172.16.102.66 172.16.102.65 netmask 255.255.255.255 up
# ifconfig gre0 tunnel 172.16.102.66 172.16.102.65
# route delete 172.16.102.65
```

ifconfig 命令在 gre0 接口上，增加了一个到路由器（172.16.102.65）的路由表入口。我发现必须删除该路由，以便 squid 能与其他路由器会话。

你也许想或必须对来自路由器的 GRE 包，增加一条 ipfw 规则：

/sbin/ipfw add allow gre from 172.16.102.65 to 172.16.102.66

9.4.3 OpenBSD

本节的示例基于 OpenBSD 3.3

为了激活包转发，在/etc/sysctl.conf 文件里增加该行：

net.inet.ip.forwarding=1

现在，在/etc/pf.conf 文件里增加如下类似行，配置包过滤规则：

```
rdr inet proto tcp from any to any port = www ->; 127.0.0.1 port 3128
pass out proto tcp from 172.16.102.66 to any
pass out proto tcp from any port = 80 to any
pass in proto tcp from any port = 80 to 172.16.102.66
```

假如你没有使用 OpenBSD 的包过滤器，需要在/etc/rc.conf.local 文件里增加一行来激活它：

```
pf=YES
```

9.4.3.1 OpenBSD 和 WCCP

首先，增加如下行到/etc/sysctl.conf 文件，告诉系统接受和处理 GRE 和 WCCP 包：

```
net.inet.gre.allow=1
net.inet.gre.wccp=1
```

然后，用如下命令配置 GRE 接口：

```
# ifconfig gre0 172.16.102.66 172.16.102.65 netmask 255.255.255.255 up
# ifconfig gre0 tunnel 172.16.102.66 172.16.102.65
# route delete 172.16.102.65
```

跟 Freebsd 一样，我发现必须删除 ifconfig 自动产生的路由。最后，依赖于包过滤器的配置，必须增加一条规则以允许 GRE 包：

```
pass in proto gre from 172.16.102.65 to 172.16.102.66
```

9.4.4 在 NetBSD 和其他系统上的 IPFilter

本节的示例基于 NetBSD 1.6.1。它们也能运行在 Solaris,HP-UX,IRIX,和 Tru64 上，既然这些系统本身就配备了 IPFilter.

激活 NetBSD 的包转发，将如下行加进/etc/sysctl.conf:

```
net.inet.ip.forwarding=1
```

然后，将如下行插入 NAT 配置文件/etc/ipnat.conf 中：

```
rdr fxp0 0/0 port 80 ->; 172.16.102.66 port 3128 tcp
```

你的接口名可能与本例的 fxp0 不同。

9.4.4.1 NetBSD 和 WCCP

我不能在 NetBSD 上运行 WCCP，即使打了 GRE 补丁来接受 WCCP 包。该问题看起来根源在

IPFilter rdr 规则阻塞了特定的端口。来自路由器的包通过 NetBSD 的 gre0 接口（在这里它们被解包）。然而，包回到路由器时，走另一条通道，未被封装并且不走同一网络接口。这样，IPFilter 代码没有将 squid 的本地 IP 地址转换回原始服务器的地址。

9.5 配置 Squid

假如你使用 Linux 2.4 和 iptables，在运行./configure 时，可使用--enable-linux-netfilter 选项。它激活某些 Linux 的特殊代码，以发现发起请求的原始服务器的 IP 地址。Squid 正常情况下从 Host 头部，得到原始服务器的名字（和/或地址）。--enable-linux-netfilter 功能仅对没有 Host 头部的请求来说是必要的。统计显示几乎所有的请求有 Host 头部，所以实际中可以不使用--enable-linux-netfilter 选项。

假如正在使用 IPFilter 包（NetBSD,Solaris,或其他），因为同样的理由，你应该使用--enable-ipf-transparent 选项。在 OpenBSD 上，请使用--enable-pf-transparent 选项。每次运行./configure 时，必须重编译 squid,见 3.8 章的描述。

在运行了./configure 和重编译了 squid 后，可以编辑 squid.conf 文件。作为起点，请确认下列指令定义了给定的值：

```
httpd_accel_host virtual
httpd_accel_port 80
httpd_accel_uses_host_header on
httpd_accel_with_proxy on
httpd_accel_single_host off
```

http_accel_host 指令是关键。它指示 squid 接受包含局部 URI 的 HTTP 请求。httpd_accel_uses_host_header 被激活，允许 squid 使用 Host 头部来重新构建完整 URI。virtual 关键字指示 squid 在缺乏 Host 头部时，将原始服务器的 IP 地址放进 URL 里。

httpd_accel_with_proxy 指令控制 squid 是否既接受 HTTP 服务（部分 URI）请求，又接受代理（完整 URI）请求。在 cache 拦截里，它应该被激活。如果没有用户明确的配置使用 squid 做代理，那即使 httpd_accel_with_proxy 没被激活，squid 也能工作。

httpd_acces_single_host 指令正常情况下被禁止，在早期版本的 squid 里，它可能被默认激活。在 cache 拦截里，它应明确被禁止。

假如拦截不止针对 80 端口，你也许该将 httpd_accel_port 设为 0。见附录 A 的更多信息。

假如你没有使用 WCCP，就该准备开始发起拦截会话到 squid 了。通过使用浏览器来上网冲浪，或者使用 squidclient 发起测试请求，就可以放手一试。假如你使用 WCCP，那么还有许多步骤要完成。

9.5.1 配置 WCCPv1

路由器不发送任何会话到 squid，直到 squid 宣称它自己是路由器。为了让 squid 那样做，在

squid.conf 中增加如下行：

```
wccp_router 172.16.102.65
wccp_version 4
```

路由器有多个接口。请确认使用与 squid 相连的接口的 IP 地址。这点是必要的，因为来自路由器的 WCCP 消息，将源 IP 地址设置为外出接口的地址。假如源地址不匹配 wccp_router 值，squid 会拒绝 WCCP 消息。

WCCPv1 文档规定 4 作为协议版本号。然而，某些用户报告，Cisco IOS 11.2 仅支持协议版本 3。假如你使用该版本的 IOS，请在 squid.conf 里改变版本号：

```
wccp_version 3
```

9.6 调试问题

HTTP 拦截比较复杂，因为许多不同设备必须组合正确工作。为了帮助你跟踪问题，如下是一个问题解答检查列表：

λ 客户端数据包正在通过路由器/交换机吗？

在简单网络里，这点显而易见。你可以 trace 线缆并观察指示灯的活动闪烁。然而在大而复杂的网络，数据包可能走不同的路线。假如你的组织够大，并有网络 sniffer 设备，就可以观察线路中 web 客户端的请求数据包。低技术的方法是，断开有问题的线路，并观察是否影响客户端 的 web 浏览。

λ 路由器/交换机配置是否正确？

你也许要再次检查路由器/交换机配置。假如你已配置了某个接口，那能否确保它正确呢？
是否新的配置真正在设备上运行？也许在你保存配置之前，路由器/交换机已重启了。在改变生效前，你或许需要 reboot 设备。

λ 交换机/路由器能与 squid 主机会话吗？

能从路由器/交换机上 ping 通 squid 吗？大部分 4 层拦截配置要求网络设备和 squid 在同一子网里。登陆路由器/交换机，确认能 ping 通 squid 的 IP 地址。

λ 交换机/路由器相信 squid 在运行吗？

许多传输拦截设备不会发送会话到 squid，除非它们知道 squid 是健壮的。使用调试命令来预览 squid 的健壮性状态。也许会发现三层健壮性检测（例如 ICMP ping）比四层检测（例如 HTTP）更容易，它使网络设备更容易将 squid 标记为存活状态。

λ Squid 实际在运行吗?

请再次确认 squid 真正在运行，特别是在系统近期重启过的情况下。

λ 数据包正在抵达 squid 主机吗？

使用 tcpdump 能见到拦截的 TCP 连接。如下是示例：

```
# tcpdump -n -i eth0 port 80
```

假如使用 WCCP，请检查来自路由器的 GRE 包：

```
# tcpdump -n -i eth0 ip proto gre
```

假如没有看到 tcpdump 的任何输出，则路由器/交换机可能没有发送任何数据。在这种情况下，返回到以前的建议。

注意，假如设备正使用四层健壮性检测，你可以在 tcpdump 的输出里见到这些。健壮性检测来自路由器/交换机的 IP 地址，所以它们容易被认出。假如你见到健壮性检测，但没有其他数据，那可能意味着路由器/交换机正把 squid 的响应理解为不健壮。例如，设备可能想见到 200(OK) 响应，但 squid 返回一个错误，例如 401（未授权）或 404（未发现）。请对 access.log 运行 tail -f 命令。

λ 激活了 IP 转发吗？

请再次确认 squid 运行的操作系统配置了 IP 包转发。假如没有，主机可能会丢弃拦截数据包，因为目的 IP 地址并非本地。

λ 配置包过滤了吗？

请确认包过滤器（例如 ipfw,iptables,pf 等）配置正确。当每件事都运行良好时，你能定期运行命令来显示过滤规则，并观察计数器增长。例如：

```
# ipfw show 300 ; sleep 3; ipfw show 300
00300  86216 8480458 fwd 127.0.0.1,3128 tcp from any to any 80 in
00300  86241 8482240 fwd 127.0.0.1,3128 tcp from any to any 80 in
```

注意该示例在 FreeBSD 上，包和字节计数器（第二和第三列）正在增长。

λ 环路接口起来和配置了吗？

假如有一条规则转发/重定向包到 127.0.0.1，请确认环路接口（例如 lo0,lo 等）起来了，并配置过它。假如没有，内核简单的跳过这条转发/重定向规则。

λ WCCP/GRE 包被正确解开了吗？

假如使用 WCCP，请确认 GRE 包被正确解开。假如因为某些理由，系统不知道该如何处理 GRE 包，那就在 netstat -s 的输出里会见到"unknown/unsupported protocol"计数器在增长。

```
# netstat -s | grep unknown

    46 packets for unknown/unsupported protocol
```

假如 OS 有 GRE 接口，请频繁运行 netstat -i 命令，观察不断增长的包数量：

```
# netstat -in | grep ^gre0

Name    Mtu Network       Address              Ipkts Ierrs     Opkts Oerrs  Coll
gre0   1476 <Link#4>;
304452     0         0     4     0
```

另外，在 GRE 接口上运行 tcpdump：

```
# tcpdump -n -i gre0
```

λ Squid 能响应客户端吗？

有可能路由器/交换机能发送包到 squid，但 squid 不能将包发送回客户端。这种情况可能发生在：防火墙过滤规则拒绝外出数据包，或 squid 没有到客户端 IP 地址的路由。为了检查这种情况，请运行 netstat -n 并观察 SYN_RCVD 状态的 sockets:

```
% netstat -n

Active Internet connections

Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)

tcp4       0      0  10.102.129.246.80      10.102.0.1.36260       SYN_RCVD
tcp4       0      0  10.102.129.226.80      10.102.0.1.36259       SYN_RCVD
tcp4       0      0  10.102.128.147.80      10.102.0.1.36258       SYN_RCVD
tcp4       0      0  10.102.129.26.80       10.102.0.2.36257       SYN_RCVD
tcp4       0      0  10.102.129.29.80       10.102.0.2.36255       SYN_RCVD
tcp4       0      0  10.102.129.226.80      10.102.0.1.36254       SYN_RCVD
tcp4       0      0  10.102.128.117.80      10.102.0.1.36253       SYN_RCVD
tcp4       0      0  10.102.128.149.80      10.102.0.1.36252       SYN_RCVD
```

假如你看到这些，请使用 ping 和 traceroute 来确认 squid 能与客户端双向通信。

λ Squid 能与原始服务器会话吗？

假如 squid 不能连接到原始服务器，拦截 HTTP 连接会无法进行。如果这点发生，netstat 会显示许多连接在 SYN_SENT 状态：

```
% netstat -n

Active Internet connections

Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)

tcp4       0      0  172.16.102.66.5217     10.102.129.145.80      SYN_SENT
tcp4       0      0  172.16.102.66.5216     10.102.129.224.80      SYN_SENT
```

tcp4	0	0	172.16.102.66.5215	10.102.128.71.80	SYN_SENT
tcp4	0	0	172.16.102.66.5214	10.102.129.209.80	SYN_SENT
tcp4	0	0	172.16.102.66.5213	10.102.129.62.80	SYN_SENT
tcp4	0	0	172.16.102.66.5212	10.102.129.160.80	SYN_SENT
tcp4	0	0	172.16.102.66.5211	10.102.128.129.80	SYN_SENT
tcp4	0	0	172.16.102.66.5210	10.102.129.44.80	SYN_SENT
tcp4	0	0	172.16.102.66.5209	10.102.128.73.80	SYN_SENT
tcp4	0	0	172.16.102.66.5208	10.102.128.43.80	SYN_SENT

再次用 ping 和 traceroute 来确认 squid 能与原始服务器会话。

λ 外出连接正被拦截吗？

假如 squid 能 ping 通原始服务器，并且仍然见到大量的连接在 SYN_SENT 状态，那么路由器/交换机可能正在拦截 squid 的外出 TCP 连接。在某些情况下，squid 能检测到这种转发循环，并写警告日志到 cache.log。如此的转发死循环能迅速耗光 squid 的所有文件描述符，这样也会在 cache.log 里产生警告。

假如你怀疑这个问题，请使用 squidclient 程序来发起一些简单的 HTTP 请求。例如，该命令发起一条直接到原始服务器的 HTTP 请求：

% /usr/local/squid/bin/squidclient -p 80 -h slashdot.org /

假如该命令成功，你可以在屏幕上见到来自 Slashdot 站点的许多难看的 HTML。然后通过 squid 来试试同样的请求：

% /usr/local/squid/bin/squidclient -r -p 3128 -h 127.0.0.1 http://slashdot.org/

假如没有在 cache.log 里看到错误消息，就可以再次在屏幕上看到一些 HTML。假如看到转发循环错误，就必须重新配置交换机/路由器，以便它允许 squid 的外出连接正常通过，而不被拦截。