Squid 中文权威指南

（第 6 章）

译者序：

本人在工作中维护着数台 Squid 服务器，多次参阅 Duane Wessels（他也是 Squid 的创始人）的这本书，原书名是"Squid: The Definitive Guide"，由 O'Reilly 出版。我在业余时间把它翻译成中文，希望对中文 Squid 用户有所帮助。对普通的单位上网用户，Squid 可充当代理服务器；而对 Sina,NetEase 这样的大型站点，Squid 又充当 WEB 加速器。这两个角色它都扮演得异常优秀。窗外繁星点点，开源的世界亦如这星空般美丽，而 Squid 是其中耀眼的一颗星。

对本译版有任何问题，请跟我联系，我的Email是：yonghua_peng@yahoo.com.cn

彭勇华

目录

6．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

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

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
```