Squid 中文权威指南

（第 3 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

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。

3.3.2Mbuf 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]

异步 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 和异步 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.
```

```
# 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。