

# Squid 中文权威指南

## (第 16 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](mailto:yonghua_peng@yahoo.com.cn)

彭勇华

## 目 录

第 16 章 调试和故障处理.....	2
16.1 一些通用问题.....	2
16.1.1 "Failed to make swap directory" .....	2
16.1.2 "Address already in use" .....	2
16.1.3 "Could not determine fully qualified hostname" .....	2
16.1.4 "DNS name lookup tests failed" .....	3
16.1.5 "Illegal character in hostname" .....	3
16.1.6 "Running out of filedescriptors" .....	4
16.1.7 "icmpRecv: Connection refused" .....	4
16.1.8 在运行一段时间后, Squid变慢了.....	4
16.1.9 调试访问控制.....	5
16.2 通过cache.log进行调试.....	5
16.3 Coredump, 断点, 和堆栈跟踪.....	11
16.3.1 不能找到core文件? .....	14
16.4 重现问题.....	15
16.5 报告Bug.....	17
译后序 .....	19

# 第 16 章 调试和故障处理

## 16.1 一些通用问题

在讨论通用 debug 前，我先提起一些经常发生的问题。

### 16.1.1 "Failed to make swap directory"

Failed to make swap directory /var/spool/cache: (13) Permission denied

这点发生在你运行 squid -z, 并且 squid 的用户 ID 没有对/var/spool 目录的写权限的时候。记住假如以 root 来启动 squid, 并且没有增加 cache\_effective\_user 行, 那么 squid 默认以 nobody 用户运行。解决方法很简单:

```
# chown nobody:nobody /var/spool
```

### 16.1.2 "Address already in use"

commBind: Cannot bind socket FD 10 to \*:3128: Address already in use

这个消息出现在 bind() 系统调用失败时, 因为请求端口已经被其他应用程序所打开。通常, 若已有一个 squid 在运行, 而又试图启动第 2 个 squid 实例, 就会发生这种情况。假如你见到这个错误消息, 请使用 ps 来观察是否 squid 已经在运行。

Squid 使用 SO\_REUSEADDR socket 选项, 以便 bind() 调用总能成功, 即使仍有一些残余的 socket 位于 TIME\_WAIT 状态。若该消息出现, 尽管 squid 没有在运行, 但你的操作系统可能在处理这个问题上有 bug。重启操作系统是解决问题的一个方法。

另一个可能性是端口 (例如 3128) 当前已被其他应用程序使用。假如你怀疑这点, 就可使用 lsof 程序来发现哪个应用正在该端口上侦听。FreeBSD 用户能使用 sockstat 代替。

### 16.1.3 "Could not determine fully qualified hostname"

FATAL: Could not determine fully qualified hostname. Please set 'visible\_hostname'

假如 squid 不能识别它自己的完整可验证域名, 就会报这个错。如下是 squid 使用的算法:

1) 假如你将 squid 的 HTTP 端口绑定在指定的接口地址上, squid 试图对该地址执行反向 DNS 查询。假如成功, 查询答案就被用上。

2) Squid 调用 `gethostname()` 函数, 然后使用 `gethostbyname()` 函数, 试着解析其 IP 地址。假如成功, squid 使用后者函数返回的官方主机名串。

假如以上 2 项技术都不能工作, squid 以前面提到的致命错误消息退出。在该情形下, 必须使用 `visible_hostname` 指令来告诉 squid 它的主机名。例如:

```
visible_hostname my.host.name
```

## 16.1.4 "DNS name lookup tests failed"

默认情况下, squid 在启动前执行一些 DNS 查询。这点确保你的 DNS 服务器可到达, 并且运行正确。假如测试失败, 可在 `cache.log` 或 `syslog` 里见到如下消息:

```
FATAL: ipcachel_init: DNS name lookup tests failed
```

假如你在内网里使用 squid, squid 可能不能查询到它的标准主机名列表。可使用 `dns_testnames` 指令来指定你自己的主机名。只要接受到响应, squid 就会认为 DNS 测试成功。

假如你想完全跳过 DNS 测试, 简单的在启动 squid 时, 使用 `-D` 命令行选项:

```
% squid -D ...
```

## 16.1.5 "Illegal character in hostname"

```
urlParse: Illegal character in hostname 'super_bikes.tripod.com'
```

默认情况下, squid 检查 URL 的主机名部分的字符, 假如它发现了非标准的字符, squid 会抱怨。参考 RFC 1034 和 1035, 名字必须由字母 A-Z, 数字 0-9, 以及短横线(-)组成。下划线(\_)是最有问题的字符之一。

Squid 验证主机名是因为, 在某些情形下, DNS 对畸形字符的解析会很困难。例如:

```
% host super_bikes.tripod.com
super_bikes.tripod.com has address 209.202.196.70

% ping super_bikes.tripod.com
ping: cannot resolve super_bikes.tripod.com: Unknown server error
```

Squid 事先检查主机名, 这好过于以后返回 `Unknown server error` 消息。然后它会告诉用户主机名包含畸形字符。

某些 DNS 解析器确实能处理下划线和其他非标准字符。假如你想让 squid 不检查主机

名，请在运行 `./configure` 时，使用 `--disable-hostname-checks` 选项。假如你允许下划线作为唯一的例外，那么使用 `--enable-underscores` 选项。

## 16.1.6 "Running out of filedescriptors"

**WARNING! Your cache is running out of filedescriptors**

上述消息出现在 `squid` 用完了所有可用文件描述符时。假如这点发生在正常条件下，就有必要增加内核的文件描述符限制，并且重新编译 `squid`。请见 3.3.1 章。

假如 `squid` 成为了拒绝服务攻击的目标，那也会见到这条消息。某些人可能有意或无意的，同时对 `squid` 发送成百上千条请求。在这种情形下，可以增加一条包过滤规则，阻止来自恶意地址的 TCP 进入连接。假如攻击是分布式的，或使用假冒源地址，就很难阻止它们。

转发循环（见 10.2 章）也可能耗尽 `squid` 的所有文件描述符，但仅仅发生在 `squid` 不能检测到死循环时。`Via` 头部包含了某个请求遍历过的所有代理的主机名。`squid` 在头部里查找它自己的主机名，假如发现了，就报告这个循环。假如因为某些理由，`Via` 头部从外出或进入 HTTP 请求里过滤掉了，`squid` 就不能检测到循环。在该情形下，所有文件描述符被循环遍历 `squid` 的同一请求迅速耗完。

## 16.1.7 "icmpRecv: Connection refused"

假如 `pinger` 程序没有正确的安装，可见到下列消息：

```
icmpRecv: recv: (61) Connection refused
```

不过看起来更象是因为没有打开 ICMP socket 的权限，`pinger` 立刻退出了。因为该进程未在运行，当 `squid` 试图与它会话时，会接受到 I/O 错误。为了解决该问题，请到源代码目录以 `root` 运行：

```
# make install-pinger
```

假如成功，你可见到 `pinger` 程序有下列文件属主和许可设置：

```
# ls -l /usr/local/squid/libexec/pinger
```

```
-rws--x--x  1 root  squid  140728 Sep 16 19:58 /usr/local/squid/libexec/pinger
```

## 16.1.8 在运行一段时间后，Squid 变慢了

看起来更象 `squid` 与其他进程，或与它自己，在竞争系统中的内存。当 `squid` 进程的内存不再充足时，操作系统被迫从交换空间进行内存读写。这对 `squid` 的性能有强烈影响。

为了证实这个想法，请使用 `top` 和 `ps` 等工具检查 `squid` 的进程大小。也检查 `squid` 自己的页面错误计数器，见 14.2.1.24 章的描述。一旦你已确认内存耗费是问题所在，请执行下列步骤来减少 `squid` 的内存使用：

1. 减少 `cache_mem` 值，见附录 B。
2. 关掉内存池，用该选项：  
`memory_pools off`
3. 通过降低一个或多个 `cache` 目录的 `size`，减少磁盘 `cache` 大小。

## 16.1.9 调试访问控制

假如访问控制不能正确工作，如下是一些有用帮助。编辑 `squid.conf` 文件，设置 `debug_options` 行如下：

```
debug_options ALL,1 33,2
```

然后，重配置 `squid`：

```
% squid -k reconfigure
```

现在，对每个客户端请求以及每个响应，`squid` 都写一条消息到 `cache.log`。该消息包含了请求方式，URI，是否请求/响应被允许或拒绝，以及与之匹配的最后 ACL 的名字。例如：

```
2003/09/29 20:22:05| The request
GET http://images.slashdot.org:80/topics/topicprivacy.gif is ALLOWED,
because it matched 'localhost'
```

```
2003/09/29 20:22:05| The reply for
GET http://images.slashdot.org/topics/topicprivacy.gif is ALLOWED,
because it matched 'all'
```

知道 ACL 的名字，并非总能知道相应的 `http_access` 行，但也相当接近了。假如必要，可以复制 `acl` 行，并给予它们唯一的名字，以便给定的 ACL 名字仅仅出现在一个 `http_access` 规则里。

## 16.2 通过 `cache.log` 进行调试

从 13.1 章已了解到，`cache.log` 包含了不同的操作消息，`squid` 认为这些消息足够重要，从而告诉了你。我们也将这些作为 `debug` 消息考虑。可以使用 `debug_options` 指令来控制出现在 `cache.log` 里的消息的冗长度。通过增加 `debug` 等级，可以见到更详细的消息，有助于

理解 squid 正在做什么。例如：

debug\_options ALL,1 11,3 20,3

在 squid 源代码里的每个 debug 消息有 2 个数字特征：1 个节和 1 个等级。节范围从 0 到 100，等级范围从 0 到 10。通常来说，节号对应着源代码的组成成分。换句话说，在单一源文件里的所有消息，有相同的节号。在某些情形下，多个文件使用同一 debug 节，这意味着某个源文件变得太大，从而被拆分成多个小块。

每个源文件的顶部有一行，用于指示 debug 节。它看起来如此：

\* DEBUG: section 9      File Transfer Protocol (FTP)

我不指望你通过查看源文件来查找节号，所有相关信息定义在表 16-1 里。

Table 16-1. Debugging section numbers for the debug_options directive		
Number	Description	Source file(s)
0	Client Database	<i>client_db.c</i>
1	Startup and Main Loop	<i>main.c</i>
2	Unlink Daemon	<i>unlinkd.c</i>
3	Configuration File Parsing	<i>cache_cf.c</i>
4	Error Generation	<i>errorpage.c</i>
5	Socket Functions	<i>comm.c</i>
5	Socket Functions	<i>comm_select.c</i>
6	Disk I/O Routines	<i>disk.c</i>
7	Multicast	<i>multicast.c</i>
8	Swap File Bitmap	<i>filemap.c</i>
9	File Transfer Protocol (FTP)	<i>ftp.c</i>
10	Gopher	<i>gopher.c</i>
11	Hypertext Transfer Protocol (HTTP)	<i>http.c</i>
12	Internet Cache Protocol	<i>icp_v2.c</i>
12	Internet Cache Protocol	<i>icp_v3.c</i>
13	High Level Memory Pool Management	<i>mem.c</i>

Table 16-1. Debugging section numbers for the debug_options directive		
Number	Description	Source file(s)
14	IP Cache	<i>ipcache.c</i>
15	Neighbor Routines	<i>neighbors.c</i>
16	Cache Manager Objects	<i>cache_manager.c</i>
17	Request Forwarding	<i>forward.c</i>
18	Cache Manager Statistics	<i>stat.c</i>
19	Store Memory Primitives	<i>stmem.c</i>
20	Storage Manager	<i>store.c</i>
20	Storage Manager Client-Side Interface	<i>store_client.c</i>
20	Storage Manager Heap-Based Replacement	<i>repl/heap/store_heap_replacement.c</i>
20	Storage Manager Logging Functions	<i>store_log.c</i>
20	Storage Manager MD5 Cache Keys	<i>store_key_md5.c</i>
20	Storage Manager Swapfile Metadata	<i>store_swapmeta.c</i>
20	Storage Manager Swapin Functions	<i>store_swapin.c</i>
20	Storage Manager Swapout Functions	<i>store_swapout.c</i>
20	Store Rebuild Routines	<i>store_rebuild.c</i>
21	Misc Functions	<i>tools.c</i>
22	Refresh Calculation	<i>refresh.c</i>
23	URL Parsing	<i>url.c</i>
24	WAIS Relay	<i>wais.c</i>
25	MIME Parsing	<i>mime.c</i>
26	Secure Sockets Layer Proxy	<i>ssl.c</i>
27	Cache Announcer	<i>send-announce.c</i>
28	Access Control	<i>acl.c</i>



Table 16-1. Debugging section numbers for the debug_options directive		
Number	Description	Source file(s)
29	Authenticator	<i>auth/basic/auth_basic.c</i>
29	Authenticator	<i>auth/digest/auth_digest.c</i>
29	Authenticator	<i>authenticate.c</i>
29	NTLM Authenticator	<i>auth/ntlm/auth_ntlm.c</i>
30	Ident (RFC 1413)	<i>ident.c</i>
31	Hypertext Caching Protocol	<i>htcp.c</i>
32	Asynchronous Disk I/O	<i>fs/aufs/async_io.c</i>
33	Client-Side Routines	<i>client_side.c</i>
34	Dnsserver Interface	<i>dns.c</i>
35	FQDN Cache	<i>fqdnocache.c</i>
37	ICMP Routines	<i>icmp.c</i>
38	Network Measurement Database	<i>net_db.c</i>
39	Cache Array Routing Protocol	<i>carp.c</i>
40	Referer Logging	<i>referer.c</i>
40	User-Agent Logging	<i>useragent.c</i>
41	Event Processing	<i>event.c</i>
42	ICMP Pinger Program	<i>pinger.c</i>
43	AIOPS	<i>fs/aufs/aiops.c</i>
44	Peer Selection Algorithm	<i>peer_select.c</i>
45	Callback Data Registry	<i>cbdata.c</i>
45	Callback Data Registry	<i>leakfinder.c</i>
46	Access Log	<i>access_log.c</i>
47	Store COSS Directory Routines	<i>fs/coss/store_dir_coss.c</i>
47	Store Directory Routines	<i>fs/aufs/store_dir_aufs.c</i>
47	Store Directory Routines	<i>fs/diskd/store_dir_diskd.c</i>
47	Store Directory Routines	<i>fs/null/store_null.c</i>

Table 16-1. Debugging section numbers for the debug_options directive		
Number	Description	Source file(s)
47	Store Directory Routines	<i>fs/ufs/store_dir_ufs.c</i>
47	Store Directory Routines	<i>store_dir.c</i>
48	Persistent Connections	<i>pconn.c</i>
49	SNMP Interface	<i>snmp_agent.c</i>
49	SNMP Support	<i>snmp_core.c</i>
50	Log File Handling	<i>logfile.c</i>
51	File Descriptor Functions	<i>fd.c</i>
52	URN Parsing	<i>urn.c</i>
53	AS Number Handling	<i>asn.c</i>
54	Interprocess Communication	<i>ipc.c</i>
55	HTTP Header	<i>HTTPHeader.c</i>
56	HTTP Message Body	<i>HttpBody.c</i>
57	HTTP Status-Line	<i>HttpStatusLine.c</i>
58	HTTP Reply (Response)	<i>HttpReply.c</i>
59	Auto-Growing Memory Buffer with printf	<i>MemBuf.c</i>
60	Packer: A Uniform Interface to Store Like Modules	<i>Packer.c</i>
61	Redirector	<i>redirect.c</i>
62	Generic Histogram	<i>StatHist.c</i>
63	Low Level Memory Pool Management	<i>MemPool.c</i>
64	HTTP Range Header	<i>HttpHdrRange.c</i>
65	HTTP Cache Control Header	<i>HttpHdrCc.c</i>
66	HTTP Header Tools	<i>HTTPHeaderTools.c</i>
67	String	<i>String.c</i>
68	HTTP Content-Range Header	<i>HttpHdrContRange.c</i>
69	HTTP Header: Extension Field	<i>HttpHdrExtField.c</i>
70	Cache Digest	<i>CacheDigest.c</i>

Table 16-1. Debugging section numbers for the debug_options directive		
Number	Description	Source file(s)
71	Store Digest Manager	<i>store_digest.c</i>
72	Peer Digest Routines	<i>peer_digest.c</i>
73	HTTP Request	<i>HttpRequest.c</i>
74	HTTP Message	<i>HttpMsg.c</i>
75	WHOIS Protocol	<i>whois.c</i>
76	Internal Squid Object handling	<i>internal.c</i>
77	Delay Pools	<i>delay_pools.c</i>
78	DNS Lookups; interacts with lib/rfc1035.c	<i>dns_internal.c</i>
79	Squid-Side DISKD I/O Functions	<i>fs/diskd/store_io_diskd.c</i>
79	Storage Manager COSS Interface	<i>fs/coss/store_io_coss.c</i>
79	Storage Manager UFS Interface	<i>fs/ufs/store_io_ufs.c</i>
80	WCCP Support	<i>wccp.c</i>
82	External ACL	<i>external_acl.c</i>
83	SSL Accelerator Support	<i>ssl_support.c</i>
84	Helper Process Maintenance	<i>helper.c</i>

**debug** 等级这样分配：重要消息有较低值，非重要消息有较高值。0 等级是非常重要的消息，10 等级是相对不紧要的消息。另外，关于等级其实并没有严格的向导或要求。开发者通常自由选择适应的 **debug** 等级。

**debug\_options** 指令决定哪个消息出现在 `cache.log`，它的语法是：

**debug\_options** section,level section,level ...

默认设置是 **ALL,1**，这意味着 **squid** 会将所有等级是 0 或 1 的 **debug** 消息打印出来。假如希望 `cache.log` 里出现更少的 **debug** 消息，可设置 **debug\_options** 为 **ALL,0**。

假如想观察某个组件的其他 **debug** 信息，简单的将相应的节号和等级增加到 **debug\_options** 列表的末端。例如，如下行对 **FTP** 服务端代码增加了等级 5 的 **debug**：

```
debug_options ALL,1 9,5
```

如同其他配置指令一样，可以改变 `debug_options`，然后给 squid 发送重置信号：

```
% squid -k reconfigure
```

注意 `debug_options` 参数是按顺序处理的，后来的值会覆盖先前的值。假如使用 `ALL` 关键字，这点尤其要注意。考虑如下示例：

```
debug_options 9,5 20,9 4,2 ALL,1
```

在该情形下，最后的值覆盖了所有先前的设置，因为 `ALL,1` 对所有节设置了 `debug` 等级为 1。

选择合适的 `debug` 节号和等级有时非常困难，尤其是对 squid 新手而言。许多更详细的 `debug` 消息仅对 squid 开发者和熟悉源代码的用户有意义。无经验的 squid 用户会发现许多 `debug` 消息无意义和不可理解。进一步的说，假如 squid 相对忙的话，你可能对某个特殊请求或事件进行独立 `debug` 有困难。假如你能一次用一个请求来测试 squid，那么高的 `debug` 等级通常更有用。

若以高 `debug` 等级来运行 squid 较长时间，需要特别谨慎。假如 squid 繁忙，`cache.log` 增长非常快，并可能最终耗尽它的分区的剩余空间。假如这点发生，squid 以致命消息退出。另一个关注点是性能可能下降明显。因为有大量的 `debug` 消息，squid 要耗费许多 CPU 资源来格式化和打印字符串。将所有 `debug` 消息写往 `cache.log`，也浪费了大量的磁盘带宽。

## 16.3 Coredump，断点，和堆栈跟踪

假如不幸，squid 可能在运行时遭遇致命错误。这类型的错误来自 3 个风格：断点，总线错误，和异常分片。

断点是源代码里的正常检测。它是一个工具，被开发者用来确认在处理某事情前，相应的条件总为真。假如条件为假，程序退出并创建一个 `core` 文件，以便开发者能分析形势。如下是个典型的示例：

```
int some_array[100];

void
some_func(int idx)
{
    ...
    assert(idx < 100);
    some_array[idx]++;
    ...
}
```

```
}
```

这里，断点确保数组索引的值位于数组范围内。假如去访问大于或等于 100 的数组元素，就会遇到错误。假如不知何故，idx 的值不小于 100，程序运行时会打印如下消息：

```
assertion failed: filename.c:123: "idx < 100"
```

假如这点发生在 squid 上，就可在 cache.log 里见到"assertion failed"消息。另外，操作系统会创建一个 core 文件，这对事后分析有用。在本节结尾，我会解释如何去处理 core 文件。

总线错误是：由于处理器检测到其总线上的异常条件，会引发机器语言指令执行时致命失败。当处理器试图操作非连续的内存地址时，通常会发生这种错误。在 64 位处理器系统上可能更容易见到总线错误，例如 Alpha 和某些 SPARC CPU。幸运的是，它们容易修复。

异常分片错误不幸的更常见，且有时难以修复。SEGV 通常发生在进程试图访问无效内存区域时（可能是个 NULL 指针，或超出进程空间之外的内存地址）。当 bug 原因和 SEGV 影响在不同时间呈现时，它们特别难于捕获到。

Squid 默认捕获总线错误和异常分片，当它们发生时，squid 试图执行一个 clean shutdown（清理关闭）。可在 cache.log 里见到类似的语句：

```
FATAL: Received Bus Error...dying.  
2003/09/29 23:18:01| storeDirWriteCleanLogs: Starting...
```

大多数情形下，squid 能够写 swap.state 文件的 clean 版本。在退出前，squid 调用 abort() 函数来创建 core 文件。core 文件可以帮助你或其他开发者来捕获和修复 bug。

在错误发生时马上创建 core 文件，而不是先调用 clean shutdown 过程，这样更利于调试。使用 -C 命令行选项，可以告诉 squid 不去捕获总线错误和异常分片：

```
% squid -C ...
```

注意某些操作系统使用文件名 core，而另外一些优先考虑进程名（例如 squid.core）。一旦找到 core 文件，请使用调试器来进行堆栈跟踪。gdb 是 GNU 调试器--GNU C 编译器的配套工具。假如没有 gdb，可试着运行 dbx 或 adb 代替。如下显示如何使用 gdb 来进行堆栈跟踪：

```
% gdb /usr/local/squid/sbin/squid /path/to/squid.core  
  
...  
Core was generated by 'squid'.  
Program terminated with signal 6, Abort trap.  
...
```

然后，敲入 `where` 来打印堆栈轨迹：

```
(gdb) where
```

```
#0  0x28168b54 in kill ( ) from /usr/lib/libc.so.4

#1  0x281aa0ce in abort ( ) from /usr/lib/libc.so.4

#2  0x80a2316 in death (sig=10) at tools.c:301

#3  0xbfbfffac in ?? ( )

#4  0x80abe0a in storeDiskdSend (mtype=4, sd=0x82101e0, id=1214000,
    sio=0x9e90a10, size=4096, offset=-1, shm_offset=0)
    at diskd/store_io_diskd.c:485

#5  0x80ab726 in storeDiskdWrite (SD=0x82101e0, sio=0x9e90a10,
    buf=0x13e94000 "...", size=4096, offset=-1, free_func=0)
    at diskd/store_io_diskd.c:251

#6  0x809d2fb in storeWrite (sio=0x9e90a10, buf=0x13e94000 "...",
    size=4096, offset=-1, free_func=0) at store_io.c:89

#7  0x80a1c2d in storeSwapOut (e=0xc5a7800) at store_swapout.c:259

#8  0x809b667 in storeAppend (e=0xc5a7800, buf=0x810f9a0 "...", len=57344)
    at store.c:533

#9  0x807873b in httpReadReply (fd=134, data=0xc343590) at http.c:642

#10 0x806492f in comm_poll (msec=10) at comm_select.c:445

#11 0x8084404 in main (argc=2, argv=0xbfbffa8c) at main.c:742

#12 0x804a465 in _start ( )
```

你可见到，堆栈轨迹打印了每个函数的名字，它的参数，以及源代码文件名和行数。当捕获 `bug` 时，这些信息特别有用。然而在某些情形下，这些还不够。可能要求你在调试器里

执行其他命令，例如打印来自某个函数的变量的值：

```
(gdb) frame 4
```

```
#4 0x80abe0a in storeDiskdSend (mtype=4, sd=0x82101e0, id=1214000,
    sio=0x9e90a10, size=4096, offset=-1, shm_offset=0)
    at diskd/store_io_diskd.c:485
485          x = msgsnd(diskdinfo->smsgid, &M,
    msg_snd_rcv_sz, IPC_NOWAIT);
```

```
(gdb) set print pretty
```

```
(gdb) print M
```

```
$2 = {
  mtype = 4,
  id = 1214000,
  seq_no = 7203103,
  callback_data = 0x9e90a10,
  size = 4096,
  offset = -1,
  status = -1,
  shm_offset = 0
}
```

在报告了某个 bug 后，请保留 core 文件一些天，可能还需要从它获取其他信息。

## 16.3.1 不能找到 core 文件？

core 文件写在进程的当前目录。squid 在启动时默认不改变其当前目录。这样你的 core 文件（如果有的话），会写在启动 squid 的目录。假如文件系统没有足够的自由空间，或进程属主没有对该目录的写权限，就无法产生 core 文件。可以使用 `coredump_dir` 指令来让 squid 使用指定的 `coredump` 目录--位于其他地方的有充足自由空间和完全权限的目录。

进程资源限制也会阻止产生 core 文件。进程限制参数之一是 `coredump` 文件的大小。大部分操作系统默认设置这个值为“无限”。在当前 shell 里使用 `limits` 或 `ulimit` 命令，可以检查当前限制。然而请注意，你的 shell 的限制可能不同于 squid 的进程限制，特别是当 squid 随系统启动而自动启动时。假如怀疑进程限制阻止了 core 文件的产生，试试这样：

```
csh% limit coredumpsize unlimited
csh% squid -NCd1
```

在 FreeBSD 上，某个 `sysctl` 参数控制了操作系统对调用了 `setuid()` 或 `setgid()` 函数的进程，是否产生 core 文件。假如以 root 启动，squid 会用到这些函数。这样为了得到 `coredump`，必须告诉内核创建 core 文件，用这个命令：

```
# sysctl kern.sugid_coredump=1
```

请见 `sysctl.conf` 的 `manpage`，关于在系统启动时如何自动设置变量的信息。

## 16.4 重现问题

有时候可能遇到这样的问题：某个请求，或原始服务器看起来不能与 `squid` 协调工作。可以使用下面的技术来确定问题在于 `squid`，客户端，或原始服务器。技巧就是捕获 HTTP 请求，然后用不同的方法响应它，直到你验证了问题。

捕获 HTTP 请求意味着获取除了 URL 外的更多信息，包括请求方式，HTTP 版本号，和所有请求头部。捕获请求的一个方法是，短期激活 `squid` 的完整 `debug` 模式。在 `squid` 主机上，敲入：

```
% squid -kdebug
```

然后，到 web 浏览器上发布请求。`squid` 几乎会立刻接受到请求。在若干秒后，回到 `squid` 主机，并发布同样的命令：

```
% squid -kdebug
```

现在 `cache.log` 文件包含了上述客户端的请求。假如 `squid` 繁忙，`cache.log` 会包含大量的请求，所以你必须查找它。它看起来如下：

```
2003/09/29 10:37:40| parseHttpRequest: Method is 'GET'
2003/09/29 10:37:40| parseHttpRequest: URI is 'http://squidbook.org/'
2003/09/29 10:37:40| parseHttpRequest: Client HTTP version 1.1.
2003/09/29 10:37:40| parseHttpRequest: req_hdr = {
User-Agent: Mozilla/5.0 (compatible; Konqueror/3)
Pragma: no-cache
Cache-control: no-cache
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, gzip, identity
Accept-Charset: iso-8859-1, utf-8;q=0.5, */q=0.5
Accept-Language: en
Host: squidbook.org
```

注意 `squid` 把首行元素分开打印，必须手工组合它们如下：

```
GET http://squidbook.org/ HTTP/1.1
```

捕获完整请求的另一个方法是使用工具例如 `netcat` 或 `socket` (<http://www.jnickelsen.de/socket/>)。启动 `socket` 程序侦听在某个端口，然后配置浏览器使用该



端口作为代理地址。当再次发起请求时，socket打印出HTTP请求：

```
% socket -s 8080

GET http://squidbook.org/ HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3)
Pragma: no-cache
Cache-control: no-cache
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, gzip, identity
Accept-Charset: iso-8859-1, utf-8;q=0.5, */q=0.5
Accept-Language: en
Host: squidbook.org
```

最后，还可以使用网络包分析工具例如 tcpdump 或 ethereal。使用 tcpdump 捕获到一些包后，可以使用 tcpshow 来查看它们：

```
# tcpdump -w tcpdump.log -c 10 -s 1500 port 80

# tcpshow -noHostNames -noPortNames < tcpdump.log | less

...

Packet 4

TIME: 08:39:29.593051 (0.000627)

LINK: 00:90:27:16:AA:75 -> 00:00:24:C0:0D:25 type=IP

IP: 10.0.0.21 -> 206.168.0.6 hlen=20 TOS=00 dgramlen=304 id=4B29
MF/DF=0/1 frag=0 TTL=64 proto=TCP cksum=15DC

TCP: port 2074 -> 80 seq=0481728885 ack=4107144217
hlen=32 (data=252) UAPRSF=011000 wnd=57920 cksum=EB38 urg=0

DATA: GET / HTTP/1.0.
Host: www.ircache.net.
Accept: text/html, text/plain, application/pdf, application/
postscript, text/sgml, */*;q=0.01.
Accept-Encoding: gzip, compress.
Accept-Language: en.
Negotiate: trans.
User-Agent: Lynx/2.8.1rel.2 libwww-FM/2.14.
.
```

注意 `tcpshow` 按数据里的新行字符为周期来进行打印。

一旦捕获到了某个请求，就将它存到文件。然后可以使用 `netcat` 或 `socket` 来让它重新通过 `squid`：

```
% socket squidhost 3128 < request | less
```

假如响应看起来正常，问题可能在于用户代理。否则，可以改变不同事情来孤立问题。例如，假如你看到一些古怪的 `HTTP` 头部，那就从请求里删除它们，然后再试一次。让请求直接到达原始服务器，而不是经过 `squid`，这样做也可调试。方法就是从请求里删除 `http://host.name/`，并将请求发送到原始服务器：

```
% cat request
```

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3)
Pragma: no-cache
Cache-control: no-cache
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, gzip, identity
Accept-Charset: iso-8859-1, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: squidbook.org
```

```
% socket squidbook.org 80 < request | less
```

以这种方式使用 `HTTP` 时，请参考 RFC 2616 和 O'Reilly 的 `HTTP: The Definitive Guide` 这本书。

## 16.5 报告 Bug

假如你的 `squid` 版本已经有几个月未更新了，在报告 `bug` 前你应该更新它。因为其他人可能也注意了同样的 `bug`，并且它已被修复。

假如你发现了 `squid` 的合理 `bug`，请将它填入到 `squid` 的 `bug` 跟踪数据库：<http://www.squid-cache.org/bugs/>。它当前是个“`bugzilla`”数据库，需要你创建一个帐号。当 `bug` 被 `squid` 开发者处理了时，你会接到更新通知。

假如你对报告 `bug` 很陌生，请先花时间阅读 Simon Tatham 写的 “How to Report Bugs Effectively” (<http://www.chiark.greenend.org.uk/~sgtatham/bugs.html>)。

当报告 `bug` 时，确认包含下列信息：

- 1) Squid 版本号。假如 bug 发生在不止一个版本上，就也要写上其他版本号。
- 2) 操作系统名字和版本。
- 3) bug 每次都发生，还是偶尔发生。
- 4) 所发生事情的精确描述。类似于"它不能工作"，"请求失败"之类的语句，本质上对 bug 修复者无用。记得要非常详细。
- 5) 对断点，总线错误，或异常分片的堆栈跟踪。

记住 squid 开发者通常是无报酬的义务劳动，所以要有耐心。严重 bug 比小问题享有更高的解决优先级。

# 译后序

当译完本书最后一章时，心头袭来深深的寂寞。

在计算机领域，国内外技术水平差之甚远，部分原因归咎于语言的差异。

某种技术在国外流行若干年后，才有相应的中文文档出现。

没有文档，技术人员无法起步；而不规范的发行文档，更是误导了一批又一批的初学者。

本书的作者 Duane Wessels 是位大师级的人物，除了精湛的技术外，他写的本书文笔通畅，脉络清晰，丝毫不晦涩。

若对研究 Squid 抱着严肃的态度，那么请认真的拜读原著。

而我所能做的，只是按照我自己的理解，把原著译成中文。

好的软件只是解决了某一方面的问题，而真正可贵的，是开源的精神。

在开源的世界里，可以体会到现实中所没有的无私与奉献。

想起陈玉莲演的小龙女，和金轮法王有过这么一段对话：

金轮法王：你能接住十招，我就放过你们。

小龙女：试试看咯。

金轮法王：若接不住呢？

小龙女：接不住就接不住咯。

多年来让我记住的，是陈玉莲这种不食人间烟火，与世无争的神韵。

彭勇华

[yonghua\\_peng@yahoo.com.cn](mailto:yonghua_peng@yahoo.com.cn)

2005 年 10 月于广州