Squid 中文权威指南

（第 7 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

7．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。