|  |
| --- |
| Neusoft |
| Memcached |
| CentOS 6.4 Memcached安装配置与使用 |

|  |
| --- |
| mattdamon1987@sina.com  2014/2/28 |

## 前言

libevent是一个事件触发的网络库，是安装memcached唯一前提条件，libevent适用于windows、linux、bsd等多种平台，内部使用select、epoll、kqueue等系统调用管理事件机制。著名分布式缓存[软件](http://baike.baidu.com/view/37.htm)memcached也是libevent based，而且libevent在使用上可以做到跨平台，而且根据libevent官方网站上公布的数据统计，似乎也有着非凡的性能。

官网地址： <http://libevent.org/>

Memcached 官网地址： <http://memcached.org/>

文档选用版本：

Libevent : 2.0.21

Memcached : v1.4.20

Linuxe : CentOS 6.4

安装目录：

Libevent: /usr/local/ libevent-2.0.21-stable

Memcached : /usr/local/memcached-1.4.20

## 生成和安装 libevent

Yum update

|  |
| --- |
| # tar zxvf libevent-2.0.21-stable.tar.gz # cd libevent-2.0.21-stable  # ./configure --prefix=/usr/local/libevent-2.0.21-stable  # make # make install  # ln -s /usr/local/libevent-2.0.21-stable /usr/local/libevent |

测试是否安装

|  |
| --- |
| # ls -al /usr/local/libevent-2.0.21-stable/lib | grep libevent  # vi /etc/ld.so.conf 加入/usr/local/libevent-2.0.21-stable/lib  运行 ldconfig 否则运行memcached的时候，会提示找不到so文件  还有一种方法是直接在lib64目录中新增一个link链接  # ln -s /usr/local/lib/libevent-2.0.so.5 /usr/lib64/libevent-2.0.so.5  注意：安装时指定prefix 省略此步骤 |

## 安装memcached，同时需要安装中指定libevent的安装位置：

|  |
| --- |
| # tar zxvf memcached-1.4.20.tar.gz  # cd memcached-1.4.20  #64位安装方法  # ./configure --prefix=/usr/local/memcached-1.4.20 --with-libevent=/usr/local/libevent-2.0.21-stable --enable-64bit --enable-threads  # make  # make install  #创建软链  # ln -s /usr/local/memcached-1.4.20 /usr/local/memcached |

如果中间出现报错，请仔细检查错误信息，按照错误信息来配置或者增加相应的库或者路径。  
安装完成后会把memcached放到 /usr/local/memcached/bin/memcached

测试是否安装成功

|  |
| --- |
| # ls -al /usr/local/memcached/bin/mem\* |

## 3、安装常见问题：配置memcached 连接libevent

|  |
| --- |
| 如果启动时出现“memcached: error while loading shared libraries:libevent-2.0.so.5: cannot  open shared object file: No such file or directory”之类的信息，表示memcached 找不到  libevent 的位置，所以，请先使用whereis libevent 得到位置，然后连接到memcached 所寻找的路径  首先查看，libevent 在哪里  # whereis libevent  libevent: /usr/local/lib/libevent.la /usr/local/lib/libevent.so /usr/local/lib/libevent.a  然后，再看memcached 从哪里找它  # LD\_DEBUG=libs memcached -v 2>&1 > /dev/null | less  可以看到：是  /usr/lib/libevent-2.0.so.5，  所以，创建软链  [root@localhost tools]# ln -s /usr/local/lib/libevent-2.0.so.5 /usr/lib/libevent-2.0.so.5  再次启动  注意： 64位系统 memcached 查找路径为： /usr/lib64/libevent-2.0.so.5  软链 ln -s /usr/local/lib/libevent-2.0.so.5 /usr/lib64/libevent-2.0.so.5 |

4、编写memcached 启动和停止脚本

目录：/usr/local/memcached-1.4.20/bin

启动 startMemcached.sh

|  |
| --- |
| ./memcached -d -m 1024 -u root -l 192.168.24.60 -p 11211 |

停止 stopMemcached.sh

|  |
| --- |
| pkill memcached |

## 清单 1. 连接到 memcached

|  |
| --- |
| telnet localhost 11211 |

如果一切正常，则应该得到一个 telnet 响应，它会指示 **Connected to localhost（已经连接到 localhost）**。如果未获得此响应，则应该返回之前的步骤并确保 libevent 和 memcached 的源文件都已成功生成。

您现现已经登录到 memcached 服务器。此后，您将能够通过一系列简单的命令来与 memcached 通信。9 个 memcached 客户端命令可以分为三类：

* 基本
* 高级
* 管理

**基本 memcached 客户机命令**

您将使用五种基本 memcached 命令执行最简单的操作。这些命令和操作包括：

* set
* add
* replace
* get
* delete

前三个命令是用于操作存储在 memcached 中的键值对的标准修改命令。它们都非常简单易用，且都使用清单 5 所示的语法：

## 清单 2. 修改命令语法

|  |
| --- |
| command <key> <flags> <expiration time> <bytes>  <value> |

表 1 定义了 memcached 修改命令的参数和用法。

**表 1. memcached 修改命令参数**

|  |  |
| --- | --- |
| **参数** | **用法** |
| key | key 用于查找缓存值 |
| flags | 可以包括键值对的整型参数，客户机使用它存储关于键值对的额外信息 |
| expiration time | 在缓存中保存键值对的时间长度（以秒为单位，0 表示永远） |
| bytes | 在缓存中存储的字节点 |
| value | 存储的值（始终位于第二行） |

现在，我们来看看这些命令的实际使用。

**set**   
set 命令用于向缓存添加新的键值对。如果键已经存在，则之前的值将被替换。

注意以下交互，它使用了 set 命令：

|  |
| --- |
| set userId 0 0 5  12345  STORED |

如果使用 set 命令正确设定了键值对，服务器将使用单词 **STORED** 进行响应。本示例向缓存中添加了一个键值对，其键为 userId，其值为 12345。并将过期时间设置为 0，这将向 memcached 通知您希望将此值存储在缓存中直到删除它为止。

**add**   
仅当缓存中不存在键时，add 命令才会向缓存中添加一个键值对。如果缓存中已经存在键，则之前的值将仍然保持相同，并且您将获得响应 **NOT\_STORED**。

下面是使用 add 命令的标准交互：

|  |
| --- |
| set userId 0 0 5  12345  STORED  add userId 0 0 5  55555  NOT\_STORED  add companyId 0 0 3  564  STORED |

**replace**   
仅当键已经存在时，replace 命令才会替换缓存中的键。如果缓存中不存在键，那么您将从 memcached 服务器接受到一条 **NOT\_STORED** 响应。

下面是使用 replace 命令的标准交互：

|  |
| --- |
| replace accountId 0 0 5  67890  NOT\_STORED  set accountId 0 0 5  67890  STORED  replace accountId 0 0 5  55555  STORED |

最后两个基本命令是 get 和 delete。这些命令相当容易理解，并且使用了类似的语法，如下所示：

|  |
| --- |
| command <key> |

接下来看这些命令的应用。

**get**   
get 命令用于检索与之前添加的键值对相关的值。您将使用 get 执行大多数检索操作。

下面是使用 get 命令的典型交互：

|  |
| --- |
| set userId 0 0 5  12345  STORED  get userId  VALUE userId 0 5  12345  END  get bob  END |

如您所见，get 命令相当简单。您使用一个键来调用 get，如果这个键存在于缓存中，则返回相应的值。如果不存在，则不返回任何内容。

**delete**   
最后一个基本命令是 delete。delete 命令用于删除 memcached 中的任何现有值。您将使用一个键调用 delete，如果该键存在于缓存中，则删除该值。如果不存在，则返回一条 **NOT\_FOUND** 消息。

下面是使用 delete 命令的客户机服务器交互：

|  |
| --- |
| set userId 0 0 5  98765  STORED  delete bob  NOT\_FOUND  delete userId  DELETED  get userId  END |

**高级 memcached 客户机命令**

可以在 memcached 中使用的两个高级命令是 gets 和 cas。gets 和 cas 命令需要结合使用。您将使用这两个命令来确保不会将现有的名称/值对设置为新值（如果该值已经更新过）。我们来分别看看这些命令。

**gets**   
gets 命令的功能类似于基本的 get 命令。两个命令之间的差异在于，gets 返回的信息稍微多一些：64 位的整型值非常像名称/值对的 “版本” 标识符。

下面是使用 gets 命令的客户机服务器交互：

|  |
| --- |
| set userId 0 0 5  12345  STORED  get userId  VALUE userId 0 5  12345  END  gets userId  VALUE userId 0 5 **4**  12345  END |

考虑 get 和 gets 命令之间的差异。gets 命令将返回一个额外的值 — 在本例中是整型值 4，用于标识名称/值对。如果对此名称/值对执行另一个 set 命令，则 gets 返回的额外值将会发生更改，以表明名称/值对已经被更新。清单 6 显示了一个例子：

**清单 6. set 更新版本指示符**

|  |
| --- |
| set userId 0 0 5  33333  STORED  gets userId  VALUE userId 0 5 **5**  33333  END |

您看到 gets 返回的值了吗？它已经更新为 5。您每次修改名称/值对时，该值都会发生更改。

**cas**   
cas（check 和 set）是一个非常便捷的 memcached 命令，用于设置名称/值对的值（如果该名称/值对在您上次执行 gets 后没有更新过）。它使用与 set 命令相类似的语法，但包括一个额外的值：gets 返回的额外值。

注意以下使用 cas 命令的交互：

|  |
| --- |
| set userId 0 0 5  55555  STORED  gets userId  VALUE userId 0 5 **6**  55555  END  cas userId 0 0 5 **6**  33333  STORED |

如您所见，我使用额外的整型值 6 来调用 gets 命令，并且操作运行非常顺序。现在，我们来看看清单 7 中的一系列命令：

**清单 7. 使用旧版本指示符的 cas 命令**

|  |
| --- |
| set userId 0 0 5  55555  STORED  gets userId  VALUE userId 0 5 **8**  55555  END  cas userId 0 0 5 **6**  33333  EXISTS |

注意，我并未使用 gets 最近返回的整型值，并且 cas 命令返回 EXISTS 值以示失败。从本质上说，同时使用 gets 和 cas 命令可以防止您使用自上次读取后经过更新的名称/值对。

**缓存管理命令**

最后两个 memcached 命令用于监控和清理 memcached 实例。它们是 stats 和 flush\_all 命令。

**stats**   
stats 命令的功能正如其名：转储所连接的 memcached 实例的当前统计数据。在下例中，执行 stats 命令显示了关于当前 memcached 实例的信息：

|  |
| --- |
| stats  STAT pid 63  STAT uptime 101758  STAT time 1248643186  STAT version 1.4.11  STAT pointer\_size 32  STAT rusage\_user 1.177192  STAT rusage\_system 2.365370  STAT curr\_items 2  STAT total\_items 8  STAT bytes 119  STAT curr\_connections 6  STAT total\_connections 7  STAT connection\_structures 7  STAT cmd\_get 12  STAT cmd\_set 12  STAT get\_hits 12  STAT get\_misses 0  STAT evictions 0  STAT bytes\_read 471  STAT bytes\_written 535  STAT limit\_maxbytes 67108864  STAT threads 4  END |

此处的大多数输出都非常容易理解。稍后在讨论缓存性能时，我还将详细解释这些值的含义。至于目前，我们先来看看输出，然后再使用新的键来运行一些 set 命令，并再次运行 stats 命令，注意发生了哪些变化。

**flush\_all**   
flush\_all 是最后一个要介绍的命令。这个最简单的命令仅用于清理缓存中的所有名称/值对。如果您需要将缓存重置到干净的状态，则 flush\_all 能提供很大的用处。下面是一个使用 flush\_all 的例子：

|  |
| --- |
| set userId 0 0 5  55555  STORED  get userId  VALUE userId 0 5  55555  END  flush\_all  OK  get userId  END |

缓存性能

在本文的最后，我将讨论如何使用高级 memcached 命令来确定缓存的性能。stats 命令用于调优缓存的使用。需要注意的两个最重要的统计数据是 et\_hits 和 get\_misses。这两个值分别指示找到名称/值对的次数（get\_hits）和未找到名称/值对的次数（get\_misses）。

结合这些值，我们可以确定缓存的利用率如何。初次启动缓存时，可以看到 get\_misses 会自然地增加，但在经过一定的使用量之后，这些 get\_misses 值应该会逐渐趋于平稳 — 这表示缓存主要用于常见的读取操作。如果您看到 get\_misses 继续快速增加，而 get\_hits 逐渐趋于平稳，则需要确定一下所缓存的内容是什么。您可能缓存了错误的内容。

确定缓存效率的另一种方法是查看缓存的命中率（hit ratio）。缓存命中率表示执行 get 的次数与错过 get 的次数的百分比。要确定这个百分比，需要再次运行 stats 命令，如清单 8 所示：

**清单 8. 计算缓存命中率**

|  |
| --- |
| stats  STAT pid 6825  STAT uptime 540692  STAT time 1249252262  STAT version 1.2.6  STAT pointer\_size 32  STAT rusage\_user 0.056003  STAT rusage\_system 0.180011  STAT curr\_items 595  STAT total\_items 961  STAT bytes 4587415  STAT curr\_connections 3  STAT total\_connections 22  STAT connection\_structures 4  STAT cmd\_get 2688  STAT cmd\_set 961  STAT get\_hits 1908  STAT get\_misses 780  STAT evictions 0  STAT bytes\_read 5770762  STAT bytes\_written 7421373  STAT limit\_maxbytes 536870912  STAT threads 1  END |

现在，用 get\_hits 的数值除以 cmd\_gets。在本例中，您的命中率大约是 71%。在理想情况下，您可能希望得到更高的百分比 — 比率越高越好。查看统计数据并不时测量它们可以很好地判定缓存策略的效率。

常有命令如下：

启动/结束  
memcached -d -m 10 -u root -l 192.168.0.122 -p 11200 -c 256 -P /tmp/memcached.pid  
-d 选项是启动一个守护进程，   
-m 是分配给Memcache使用的内存数量，单位是MB，这里是10MB  
-u 是运行Memcache的用户，这里是root  
-l 是监听的服务器IP地址，如果有多个地址的话，这里指定了服务器的IP地址192.168.0.122   
-p 是设置Memcache监听的端口，这里设置了12000，最好是1024以上的端口  
-c 选项是最大运行的并发连接数，默认是1024，这里设置了256，按照你服务器的负载量来设定  
-P 是设置保存Memcache的pid文件  
kill `cat /tmp/memcached.pid`

获取运行状态  
echo stats | nc 192.168.1.123 11200  
watch "echo stats | nc 192.168.1.123 11200" (实时状态)