**Memcache缓存服务器**

**一、Memcache简述**

**1、Memcache定义：**

memcache是一款开源、高性能、分布式内存对象缓存系统，可应用各种需要缓存的场景，其主要目的是通过降低对Database的访问来加速web应用程序。它是一个基于内存的“键值对(Key/Value)”存储，它的工作机制是在内存中开辟一块空间，用于存储数据库调用、API调用或页面引用结果的直接数据，如字符串、图片等，并且将其建立为Hash-Table，由memcached自身管理。

memcached是memcache系统的主程序文件，以守护程序方式运行于一个或多个服务器中，随时接受客户端的连接操作，使用共享内存存取数据。

**2、Memcache出现的背景：**

许多Web应用都将数据保存到RDBMS(关系型数据库管理系统)中，应用服务器从中读取数据并在浏览器中显示。 但随着数据量的增大、访问的集中，就会出现RDBMS的负担加重、数据库响应恶化、 网站显示延迟等重大影响。

在大型系统中，访问同样的数据是很频繁的，memcached可以大大降低数据库压力，通过缓存数据库查询结果，减少对数据库访问次数，来加速Web应用程序的速度、 提高可扩展性。使系统执行效率提升。



**3、Memcache特性：**

Memcached是一款开发工具，它既不是一个代码加速器，也不是数据库中间件。其设计思想主要反映在如下方面：

(1)简单key/value存储：服务器不关心数据本身的意义及结构，只要是可序列化数据即可。存储项由“键、过期时间、可选的标志及数据”四个部分组成；

(2)功能的实现一半依赖于客户端，一半基于服务器端：客户负责发送存储项至服务器端、从服务端获取数据以及无法连接至服务器时采用相应的动作；服务端负责接收、存储数据，并负责数据项的超时过期；

(3) memcached不互相通信的分布式即不在服务器间进行数据同步；

(4)O(1)的执行效率

(5)清理超期数据：默认情况下，Memcached是一个LRU(最近最少使用)缓存，同时，它按事先预订的时长清理超期数据；但事实上，memcached不会删除任何已缓存数据，只是在其过期之后不再为客户所见；而且，memcached也不会真正按期限清理缓存，而仅是当get命令到达时检查其时长；

Memcached的服务器客户端通信并不使用复杂的XML等格式， 而使用简单的基于文本格式和二进制格式。因此，通过telnet 也能在memcached上保存数据、取得数据。

**二、Memcached内存分配机制**

**1、Slab Allocation机制**

memcached默认情况下采用了名为Slab Allocator的机制分配、管理内存。在该机制出现以前，内存的分配是通过对所有记录简单地进行malloc()和free()来进行的。但是，这种方式会导致内存碎片，加重操作系统内存管理器的负担，最坏的情况下，会导致操作系统比memcached进程本身还慢。Slab Allocator就是为解决该问题而诞生的。

Slab Allocator的基本原理是按照预先规定的大小，将分配的内存分割成特定长度的块，以完全解决内存碎片问题。

**2、Slab Allocation的主要术语**

(1)page：分配给slab用于切割的内存空间，默认是1MB。分配给slab之后根据slab的大小切分成chunk

(2)chunk：存放数据的最小单元。用户数据item（key、value等）最终会保存在chunk中。

(3)slab class：特定大小的chunk的组，在memcached中，对数据的管理是以slab为单元进行管理的。

每个slab class对应一个或多个空间大小相同的chunk。参考下图：



**3、在Slab中缓存记录的原理**

memcached对客户端发送的数据选择最适合数据大小的slab并缓存到chunk中，memcached服务器中保存着slab内空闲chunk的列表，根据该列表选择chunk，然后将数据缓存其中。



**4、Slab Allocator还存在的问题：**

Slab Allocator解决了当初的内存碎片问题，但新的机制也给memcached带来了新的问题。这个问题就是，由于分配的是特定长度的内存，因此无法有效利用分配的内存。例如，将35字节的数据缓存到45字节的chunk中，剩余的10字节浪费了，见下图：



在后续的学习中可知：可以通过设置合适的growth factor（增长系数/因子）选项来调节slab class的大小差别，从而来减少内存资源的浪费

**四、分布式缓存中三种负载均衡的方法**

*memcached的分布式不是在服务器端实现的，而是在客户端的应用中实现的，即通过内置算法制定目标数据的节点*。基于客户端的memcached的分布式架构(见下图)。memcached客户端中不同的内置算法会严重影响缓存的有效性，基于此作者整理了以下三种内置算法。



**1、传统的数据分布方法,将key的hash值对机器数取模**

    Hash，翻译做“散列”，就是把任意长度的输入（又叫做预映射， pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来唯一的确定输入值。简单的描述就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。

对用户的检索关键词分词后取hash值，计算hash(key)/n，n为机器数，得到的值就是该key需要[路由](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=18&is_app=0&jk=65b6cc0cdd74670c&k=%C2%B7%D3%C9&k0=%C2%B7%D3%C9&kdi0=0&luki=10&mcpm=0&n=10&p=baidu&q=65035100_cpr&rb=0&rs=1&seller_id=1&sid=c6774ddcccb665&ssp2=1&stid=0&t=tpclicked3_hc&td=1836545&tu=u1836545&u=http%3A%2F%2Fwww.bubuko.com%2Finfodetail-554591.html&urlid=0)到的服务器编号了。优点很明显，缺点是在在服务器数量发生变化的时候，缓存会大量失效，举例来说，共有4个memcache缓存服务器，当客户端A浏览的数据都缓存在1号服务器，且当1号服务器宕机时，由于缓存服务器个数由4减为1，那么A的数据不仅将会全部丢失，对于后续浏览器的缓存数据也会造成雪崩式的丢失(由于取模结果与之前不同)。

**2、一致性hash 闭环**

考虑到传统数据分布方法对缓存数据具有潜在的灾难性后果，一致性hash则可以解决部分问题，通常可以考虑[0,(2^32-1)]为hash值的取值范围。我们可以把取值范围想象成一个闭环，2^32-1和0相连接。如下图所示。

假设有4个缓存服务器节点(node1~node4)，首先计算这四个节点的hash值，这四个点占据了闭环的四个位置。需要根据key来路由缓存服务器，首先计算hash(key)值（这个hash算法需要保证hash(key)的值落在在区间[0,2^32-1]中，hash(key)对2^32-1取模可保证在区间内）。然后hash（key）值必定位于闭环中的某一个点，然后按照顺时针方向，选择距离最近的node，命中的node就是key需要路由的服务器。如上图，需要找到key2的缓存值所在的服务器，首先计算hash(key2)，落在如图的闭环中，然后顺时针找到离这个hash值最近的node，可以看到是node1。key2对应的缓存值就存在node1中。

当四个节点中有一个节点失效时，影响的缓存数据仅仅为上图中黄色区域(node3~node4)，大大的减少了缓存失效的范围。



如下图所示，为了提高查询效率，当增加了一个节点时：只有hash(key)落在node1~node5之间的key才会有影响，在未增加node5之前落在这个范围的key最终路由到node2,增加node5之后路由到node5了，缓存失效了。除此之外，落在其他范围的hash值就不受影响。如上图，增加node5之后key3和key4原来会路由到node2的，现在路由到node5。而key7不受影响，还是路由到node2。

一致性哈希闭环算法也存在缺点，如下图，当节点机数量少且分布不均衡时，会造成缓存分布不均，当其中缓存两=量较大的节点失效时，也会造成比较大的缓存失效的后果，因此在最初设计时因避免节点分布不均的情况：



针对当节点机数量少造成缓存分布不均的情况，引入一个虚拟节点的概念，即在现有节点的基础上，在每一个节点内部虚拟出几个memcache服务器。这样就可以把压力尽量分配到各个机器上。另外需要维护一张表格，用来记录虚拟节点指向的真正节点。

优点：相比简单的对机器数取模算法，当节点变动的时候只有相对较少的key失效，实现也相对简单。不需要进行数据迁移，每个服务器是独立的。

缺点：还是会有部分的key失效，当访问量非常大的时候，若访问到失效的key，则会直接连接数据库，可能造成数据服务器宕机。

**3、tair负载均衡算法：(详细参考：https://www.lvtao.net/database/tair.html)**

tair是淘宝开发的一个分布式 key/value 存储引擎。tair 分为持久化和非持久化两种使用方式：非持久化的 tair 可以看成是一个分布式缓存；持久化的 tair 将数据存放于磁盘中。为了解决磁盘损坏导致数据丢失，tair可以配置数据的备份数目，tair 自动将数据备份到不同的主机上, 当有主机发生异常, 无法正常提供服务的时候, 其于的备份会继续提供服务。

**(1)tair的总体结构**

tair作为一个分布式系统, 是由一个中心控制节点和一系列的服务节点组成。称中心控制节点为config server，服务节点data server。config server负责管理所有的data server, 维护data server的状态信息。data server 对外提供各种数据服务, 并以心跳的形式将自身状况汇报给config server。config server是控制点, 而且是单点, 目前采用一主一备的形式来保证其可靠性。所有的 data server 地位都是等价的。



**五、memcached 数据过期清理方式：Lazy Expiration + LRU**

**1、Lazy Expiration（惰性过期）**

根据缓存对象的ttl进行清理，memcached内部不会监视记录是否过期，而是在get时查看记录的时间戳，检查记录是否过期。这种技术被称为lazy（惰性）expiration。因此，memcached不会在过期监视上耗费CPU时间。

**2、LRU（最近最少使用算法）**

memcached会优先使用已超时的记录的空间，但即使如此，也会发生追加新记录时空间不足的情况，此时就要使用名为Least Recently Used（LRU）机制来分配空间。当memcached的内存空间不足时（无法从slab class 获取到新的空间时），就从最近未被使用的记录中搜索，并将其空间分配给新的记录。可以支持禁用此功能。

**六、memcached 安装配置**

**1、memcached依赖于libevent库**

memcached 基于libevent的事件处理，libevent是个程序库，它将Linux的epoll、epool、select、BSD类操作系统的kqueue等事件处理功能封装成统一的接口。即使对服务器的连接数增加，也能发挥O(1)的性能。 memcached依赖于libevent API，因此要需首先安装，libevent官方主页：http://libevent.org/

**[root@web ~]#tar zxvf libevent-2.0.22-stable.tar.gz**

**[root@web ~]#cd libevent-2.0.22-stable**

**[root@web ~]#./configure --prefix=/usr/local/libevent**

**[root@web ~]#make && make install**

**##添加配置文件：**

**[root@web ~]#echo "/usr/local/libevent/lib" > /etc/ld.so.conf.d/libevent.conf**

**##将liebevent库文件追击到系统库内，使其生效：**

**[root@web ~]#ldconfig -a|head**

**2、memcached的安装**

**(1)yum安装**

**[root@web ~]#yum install memcached -y**

**[root@web ~]#rpm -ql memcached**

**/etc/rc.d/init.d/memcached ##服务脚本**

**/etc/sysconfig/memcached ##配置文件**

**/usr/bin/memcached ##可执行文件**

**/usr/bin/memcached-tool ##memcached工具**

**......**

**(2)编译安装memcached**

官方主页：https://memcached.org。前期已将libevent安装在/usr/local/libevent目录下，以下是编译安装配置memcached：

**[root@web ~]#wget http://www.memcached.org/files/memcached-1.4.33.tar.gz**

**[root@web ~]#tar zxvf memcached-1.4.33.tar.gz**

**[root@web ~]#cd memcached-1.4.33**

**[root@web ~]#./configure --prefix=/usr/local/memcached --with-libevent=/usr/local/libevent**

**[root@web ~]#make && make install**

**##环境配置**

**[root@web ~]#cat /etc/profile.d/memcache.sh**

**export PATH=$PATH:/usr/local/memcached/bin/**

**[root@web ~]#source /etc/profile.d/memcache.sh**

**3、memcached常用选项**

* -p TCP监听端口 (default: 11211)，小写
* -U UDP 监听端口 (default: 11211, 0默认关闭)，大写
* -l监听的服务器IP地址 (default: all addresses)
* -d 运行为守护进程，后台运行
* -u 运行memcached用户
* -m #：最大的内存使用 (default: 64 MB)，也不宜过大
* -c 最大并发连接 (default: 1024)
* -v 输出警告和错误信息
* -vv 同时打印客户端请求和返回信息
* -vvv 打印内部状态转换信息
* -P 设置保存pid文件目录, 仅当使用-d选项时
* **-f 指定增长因子|倍数 (default: 1.25)**
* -n key+value+flags最小分配空间(default: 48)
* -L 尝试使用大内存页。增加内存页大小可以减少失误的TLB数量，提高性能。
* -t 使用的线程数量 (default: 4)
* -R 每个事件的最大请求数 (default: 20)
* -o 配置额外选项

**4、memcached的使用**

**(1)启动memcached：**

**[root@web ~]#id memcached ##使用yum安装，自动添加memcached用户**

**uid=495(memcached) gid=493(memcached) groups=493(memcached)**

**[root@web ~]# memcached -u memcached -f 4 -vvv**

**slab class 1: chunk size 96 perslab 10922**

**slab class 2: chunk size 384 perslab 2730**

**slab class 3: chunk size 1536 perslab 682**

**slab class 4: chunk size 6144 perslab 170**

**slab class 5: chunk size 24576 perslab 42**

**slab class 6: chunk size 98304 perslab 10**

**slab class 7: chunk size 1048576 perslab 1**

**<26 server listening (auto-negotiate)**

**<27 server listening (auto-negotiate)**

**<28 send buffer was 229376, now 268435456**

**<29 send buffer was 229376, now 268435456**

**##从输出结果可以看到memcached的内存分配过程：**

**[root@web ~]#service memcached start**

**注：编译安装的memcached后文提供了startup脚本以及自定义的配置文件**

**[root@web ~]#ss -tnl ##监听11211TCP端口**

**State Recv-Q Send-Q Local Address:Port Peer Address:Port**

**LISTEN 0 128 :::11211 :::\***

**LISTEN 0 128 \*:11211 \*:\***

**(2)memcached 基本命令**

memcached提供了几个命令来完成与服务器端的交互，这些命令基于memcached的协议实现。memcached可以利用telnet进行交互式连接，以下的命令均可在Telnet连接下执行：

存储类命令：set, add, replace, append（在一个缓存后附加内容）, prepend（在一个缓存之前附加内容）

获取数据类命令：get, delete, incr/decr（自加1/自减1）

统计类命令：stats, stats items, stats slabs, stats sizes

清理命令：flush\_all

**##命令详解：**

**##set： add keyname flag（修饰符）  timeout  datasize ##添加或者更新key**

**##add： add keyname flag（修饰符）  timeout  datasize ##key不存在时添加一个key**

**##示例： add mykey 0（#表示没有修饰符） 10（#生成周期） 12（#数据大小）**

**Hello World(#键值(value))**

**##get: get keyname ##获取键所对应的值，gets 命令比get返回的值多一个数字，用来判断数据是否发生过改变。**

**##replace：replace keyname flag（修饰符）  timeout  datasize ##数据存在时替换，不存在时返回错误**

**##delete： delete keyname ##删除key及其对应的值**

**##stats： stats ##状态命令，显示memcached当前状态**

**[root@web ~]# telnet 192.168.88.66 11211**

**Trying 192.168.88.66...**

**Connected to 192.168. 88.66.**

**Escape character is '^]'.**

**stats**

**STAT pid 8279 #进程ID**

**STAT uptime 8000 #服务器运行秒数**

**STAT time 1378284623 #服务器当前unix时间戳,秒**

**STAT version 1.4.15 #服务器版本**

**STAT libevent 2.0.21-stable #libevent版本号**

**STAT pointer\_size 64 #操作系统指针大小(这台服务器是64位的)**

**STAT rusage\_user 0.000999 #计用户时间**

**STAT rusage\_system 0.003999 #进程累计系统时间**

**STAT curr\_connections 10 #当前打开连接数**

**STAT total\_connections 11 #曾打开的连接总数**

**STAT connection\_structures 11 #服务器分配的连接结构数**

**STAT reserved\_fds 20 #内部使用的FD数**

**STAT cmd\_get 0 #执行get命令总数**

**STAT cmd\_set 0 #执行set命令总数**

**STAT cmd\_flush 0 #执行flush命令总数**

**STAT cmd\_touch 0 #执行touch命令总数**

**STAT get\_hits 0 #get命中次数**

**STAT get\_misses 0 #get未命中次数**

**STAT delete\_misses 0 #delete未命中次数**

**STAT delete\_hits 0 #delete命中次数**

**STAT incr\_misses 0 #incr未命中次数**

**STAT incr\_hits 0 #incr命中次数**

**STAT decr\_misses 0 #decr未命中次数**

**STAT decr\_hits 0 #decr命中次数**

**STAT cas\_misses 0 #cas未命中次数**

**STAT cas\_hits 0 #cas命中次数**

**STAT cas\_badval 0 #使用擦拭次数**

**STAT touch\_hits 0 #touch命中次数**

**STAT touch\_misses 0 #touch未命中次数**

**STAT auth\_cmds 0 #认证处理的次数**

**STAT auth\_errors 0 #认证失败次数**

**STAT bytes\_read 7 #读取字节总数**

**STAT bytes\_written 0 #写入字节总数**

**STAT limit\_maxbytes 134217728 #现在的内存大小为128M**

**STAT accepting\_conns 1 #目前接受的新接数**

**STAT listen\_disabled\_num 0 #失效的监听数**

**STAT threads 4 #当前线程数**

**STAT conn\_yields 0 #连接操作主支放弃数目**

**STAT hash\_power\_level 16 #hash等级**

**STAT hash\_bytes 524288 #当前hash表等级**

**STAT hash\_is\_expanding 0 #hash表扩展大小**

**STAT bytes 0 #当前存储占用的字节数**

**STAT curr\_items 0 #当前存储数据总数**

**STAT total\_items 0 #启动以来存储的数据总数**

**STAT expired\_unfetched 0 #已过期但未获取的对象数目**

**STAT evicted\_unfetched 0 #已驱逐但未获取的对象数目**

**STAT evictions 0 #LRU释放的对象数目**

**STAT reclaimed 0 #用已过期的数据条目来存储新数据的数目**

**END**

**##flush\_all ##清空所有选项，flush\_all 实际上没有立即释放项目所占用的内存，而是在随后陆续有新的项目被储存时执行（这是由memcached的懒惰检测和删除机制决定的）。**

**##append和prepend ##在Value后续追加或在其前面添加**

**5、memcached图形工具(memcachephp)**

**##下载memcachephp**

**[root@web ~]#wget https://codeload.github.com/lagged/memcache.php/zip/master**

**[root@web ~]#unzip memcache.php-master.zip**

**[root@web ~]#mv memcahch.php-master /var/www/html/memadmin**

**[root@web www]# ls memadmin/**

**apps config.php images include index.php langs LICENSE.txt README.txt views**

**##config.php文件定义了管理员默认登录用户和密码，并设置要连接memcached服务器**

火狐或者谷歌浏览器中输入：http://192.168.88.66/memadmin即可看到如下图形化的web界面：



**6、使用libmemcached的客户端工具:**

访问memcached的传统方法是使用基于perl语言开发的Cache::memcached模块，这个模块在大多数perl代码中都能良好的工作，但也有着性能方面的问题。libMemcached则是基于C语言开发的开源的C/C++代码，它还提供了数个可以远程使用的memcached管理工具，如memcat, memping，memstat，memslap等。

(1) 编译安装libmemcached，访问官网：http://libmemcached.org/libMemcached.html，下载相应版本的软件包

**##下载libmemcached**

**[root@web ~]#wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz**

**[root@web ~]#** **tar xf libmemcached-1.0.18.tar.gz**

**[root@web ~]#cd libmemcached-1.0.18**

**[root@web ~]#./configure**

**[root@web ~]#make && make install**

**[root@web ~]#ldconfig -v**

(2) 客户端工具

**[root@web ~]#** **memcat --servers=127.0.0.1:11211 mykey**

**[root@web ~]#memping**

**[root@web ~]#memslap --servers=127.0.0.1**

**[root@web ~]#memstat --servers=127.0.0.1**

**7、memcached整合其他软件的运用：**

**(1)Memcached的PHP扩展**

官网https://pecl.php.net/package/memcache

**[root@web ~]#wget https://pecl.php.net/get/memcache-3.0.0.tgz**

**[root@web ~]#tar xvf memcache-3.0.0.tgz**

**[root@web ~]#cd memcache-3.0.0**

**[root@web ~]#/usr/local/php/phpize #采用php的命令生成configure文件**

**[root@web ~]#./configure --with-php-config=/usr/local/php/bin/php-config --enable-memcach**

**[root@web ~]#make && make install ##在安装的最后会有如下提示信息输出**

**.............................**

**Installing shared extensions:/usr/local/php/lib/php/extensions/no-debug-non-zts-20131226/memcached.so**

**##将上述信息添加到下述文件中，使memcache加载进php中：**

**[root@web ~]#cat /etc/php.d/php.ini**

**extension=/usr/local/php/lib/php/extensions/no-debug-non-zts-20131226/memcached.so**

**[root@web ~]#service php-fpm restart**

**##而后对memcached功能进行测试，在网站目录中建立测试页面test.php，添加如下内容：**

**[root@web ~]# vim /var/www/html/index.php**

**<?php**

**$mem = new Memcache;**

**$mem->connect("192.168.88.11", 11211) or die("Could not connect");**

**$mem->set('hellokey', 'Hello World', 0, 600) or die("Failed to save data at the memcac**

**hed server");**

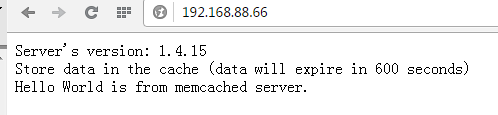
**echo "Store data in the cache (data will expire in 600 seconds)<br/>\n";**

**$get\_result = $mem->get('hellokey');**

**echo "$get\_result is from memcached server.";**

**?>**

浏览器中输出测试页面：



**(2)Nginx整合memcached:**

nginx的memcached\_module模块可以直接从memcached服务器中读取内容后输出，后续的请求不再经过应用程序(如php-fpm、django)处理,大大的提升动态页面的速度。nginx只负责从memcached服务器中读取数据，要往memcached写入数据还得需要后台的应用程序来完成，主动的将要缓存的页面缓存到memcached中，可以通过404重定向到后端去处理的。



memcached的key可以通过memcached\_key变量来设置，如以$uri。如果命中，那么直接输出内容，没有命中就意味着nginx需要从应用程序请求页面。同时，还希望该应用程序将键值对写入到memcached服务器中，以便下一个请求可以直接从memcached获取。如果键值不存在，nginx将报告not found错误(404)。最好的方法是使用error\_page(指定和location请求处理。同时包含”Bad Gateway” 502)错误和”Gateway Timeout”(504)错误，如：error\_page 404 502 504 = @app;。注意：需要设置default\_type，否则可能会显示不正常。

1)Nginx整合memcached的配置文件(/etc/nginx/nginx.conf)修改：

**[root@web ~]# vim /etc/nginx/nginx.conf**

**...........................**

**server {**

**location / {**

**set $memcached\_key "$uri?$args";**

**memcached\_pass 192.168.88.11:11211;**

**# default\_type text/html;**

**error\_page 404 @fallback; #调用一个叫fallback的location**

**}**

**location @fallback { #定义一个叫fallback的location**

**proxy\_pass http://127.0.0.1;**

**}**

**}**

**.........................**

从配置文件可知，一个请求到达后，将uri作为key去Memcached服务器192.168.88.11：11211上查找value，如果没有命中，则返回404。这时通过error\_page将404接收转到@fallback，返回给Web服务去处理请求。

2)Nginx中关于memcache的模块说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 模块名称 | 语法 | 模块含义 | 默认值 |
| memcached\_bind | memcached\_bind address | off; | 指定从哪个IP来连接memcached服务器 | none |
| **memcached\_pass** | **memcached\_pass address:port or socket；** | **memcached服务器地址:端口号** | **none** |
| memcached\_buffer\_size | memcached\_buffer\_size size; | 从memcached服务器接收到响应的缓冲大小 | 4k|8k; |
| memcached\_connect\_timeout | memcached\_connect\_timeout time; | 与memcached服务器建立连接的超时时间 | 60s |
| memcached\_read\_timeout | memcached\_read\_timeout time; | 从memcached服务器读取响应超时时间 | 60s |
| memcached\_send\_timeout | memcached\_send\_timeout time； | 发送请求到memcached服务器的超时时间 | 60s |

模块名称：memcached\_next\_upstream

语法: memcached\_next\_upstream error | timeout | invalid\_response | not\_found | off …;

默认值： error timeout;

模块含义：哪些状态下请求将转发到另外的负载均衡服务器上( memcached\_pass有多个适用)。

注：以上各个模块在配置文件中的位置：http->server->location

**附件：**

**##编译安装memcached的启动启动脚本**

**[root@web ~]# vim /etc/rc.d/init.d/memcached**

**#!/bin/bash**

**#**

**# chkconfig: 35 99 05**

###35: runlevel下启动，”-”代表关闭；

###99：在rc3.d和rc5.d下产生S99fmemcached （越小优先权越高）

###05: 在rc0.d、rc1.d、rc2.d、rc4.d、rc6.d下产生K05fmemcached（越小优先权越高）

####S:代表Start

####K:代表Kill

**# description: The memcached daemon is a network memory cache service.**

**# processname: memcached**

**# config: /etc/sysconfig/memcached**

**# pidfile: /var/run/memcached/memcached.pid**

**. /etc/init.d/functions**

**PORT=11211**

**USER=memcached**

**MAXCONN=1024**

**CACHESIZE=64**

**OPTIONS=""**

**[ -f /etc/sysconfig/memcached ]&&. /etc/sysconfig/memcached**

**. /etc/sysconfig/network**

**if [ "$NETWORKING" = "no" ]**

**then**

**exit 0**

**fi**

**RETVAL=0**

**prog="/usr/local/memcache/bin/memcached"**

**pidfile=${PIDFILE-/var/ruan/memcached/memcached.pid}**

**lockfile=${LOCKFILE-/var/lock/subsys/memcached}**

**start () {**

**echo -n $"Starting $prog: "**

**# Ensure that /var/run/memcached has proper permissions**

**if [ "`stat -c %U /var/run/memcached`" != "$USER" ]; then**

**chown $USER /var/run/memcached**

**fi**

**daemon --pidfile ${pidfile} memcached -d -p $PORT -u $USER -m $CACHESIZE -c $MAXCONN -P ${pidfile} $OPTIONS**

**RETVAL=$?**

**echo**

**[ $RETVAL -eq 0 ] && touch ${lockfile}**

**}**

**stop () {**

**echo -n $"Stopping $prog: "**

**killproc -p ${pidfile} /usr/bin/memcached**

**RETVAL=$?**

**echo**

**if [ $RETVAL -eq 0 ] ; then**

**rm -f ${lockfile} ${pidfile}**

**fi**

**}**

**restart () {**

**stop**

**start**

**}**

**case "$1" in**

**start)**

**start**

**;;**

**stop)**

**stop**

**;;**

**status)**

**status -p ${pidfile} memcached**

**RETVAL=$?**

**;;**

**restart|reload|force-reload)**

**restart**

**;;**

**condrestart|try-restart)**

**[ -f ${lockfile} ] && restart || :**

**;;**

**\*)**

**echo $"Usage: $0 {start|stop|status|restart|reload|force-reload|condrestart|try-restart}"**

**RETVAL=2**

**;;**

**esac**

**exit $RETVAL**

**##以下是memcached启动时的配置信息，可手动编辑：**

**[root@web ~]#cat /etc/sysconfig/memcached**

**PORT="11211"**

**USER="memcached"**

**MAXCONN="1024"**

**CACHESIZE="64"**

**OPTIONS=""**

**[root@web ~]#chmod +x /etc/rc.d/init.d/memcached**

**[root@web ~]#chkcon --add memcached**

**[root@web ~]#service memcached start**

参考网址：http://www.ha97.com/4575.html#

https://www.lvtao.net/database/tair.html