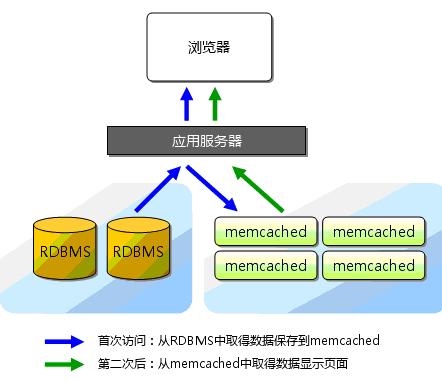
memcached的基本使用

# Memcached的简介

Memcached是一个自由开源的，高性能，分布式内存对象缓存系。Memcached是以LiveJournal旗下Danga Interactive公司的Brad Fitzpatric为首开发的一款软件。现在已成为mixi、hatena、Facebook、Vox、LiveJournal等众多服务中提高Web应用扩展性的重要因素。

Memcached是一种基于内存的key-value存储，用来存储小块的任意数据（字符串、对象）。这些数据可以是数据库调用、API调用或者是页面渲染的结果。Memcached简洁而强大。它的简洁设计便于快速开发，减轻开发难度，解决了大数据量缓存的很多问题。它的API兼容大部分流行的开发语言。本质上，它是一个**简洁的key-value存储系统**。使用目的是：通过缓存数据库查询结果，减少数据库访问次数，以提高动态Web应用的速度、提高可扩展性。



## Memcached的特征

* 协议简单
* 基于libevent的事件处理
* 内置内存存储方式
* memcached不互相通信的分布式

## Memcached的安装

yum install memcached -y

启动选项：

-d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；

-u是运行Memcache的用户；

-l是监听的服务器IP地址，可以有多个地址；

-p是设置Memcache监听的端口，，最好是1024以上的端口；

-c是最大运行的并发连接数，默认是1024；

-P是设置保存Memcache的pid文件。

例如：memcached –d –u root –m 512 –c 1024 -p 11211

## Memcached的连接

我们可以通过 telnet 命令并指定主机ip和端口来连接 Memcached 服务。

格式：telnet HOST PORT

quit //退出

# Memcached存储命令

## Set命令

Memcached set 命令：用于将 **value(数据值)** 存储在指定的 **key(键)** 中。

如果set的key已经存在，该命令可以更新该key所对应的原来的数据，也就是实现更新的作用。

语法：

|  |
| --- |
| set key flags exptime bytes [noreply]  value  参数：   * key：键值 key-value 结构中的 key，用于查找缓存值。 * flags：memcached 基本文本协议,传输的东西是理解成字符串来存储.也就是说不管你往里面存入什么数据,最终都是字符串来存储;所以我们一般把数组等数据,序列化以后存入memcache,到取出来的时候,这个flag就决定是否要序列化; * exptime：在缓存中保存键值对的时间长度（以秒为单位，0 表示永远）或者时间戳 * bytes：在缓存中存储的字节数 * noreply（可选）： 该参数告知服务器不需要返回数据 * value：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）   输出信息说明：   * STORED：保存成功后输出。 * ERROR：在保持失败后输出。   例如：  set runoob 0 900 9  memcached  STORED  get runoob  VALUE runoob 0 9  memcached  END |

## add 命令

Memcached add 命令用于将 **value(数据值)** 存储在指定的 **key(键)** 中。

如果 add 的 key 已经存在，则不会更新数据，之前的值将仍然保持相同，并且您将获得响应 **NOT\_STORED**。

语法：

|  |
| --- |
| add key flags exptime bytes [noreply]  value  参数：   * key：键值 key-value 结构中的 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 。 * exptime：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数 * noreply（可选）： 该参数告知服务器不需要返回数据 * value：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）   输出信息说明：   * STORED：保存成功后输出。 * NOT\_STORED ：在保持失败后输出。   例如：  add new\_key 0 900 10  new\_\_\_date  NOT\_STORED |

## replace 命令

memcached replace 命令用于替换已存在的 **key(键)** 的 **value(数据值)**。

如果 key 不存在，则替换失败，并且您将获得响应 **NOT\_STORED**。

语法：

|  |
| --- |
| replace key flags exptime bytes [noreply]  value  参数说明：   * key：键值 key-value 结构中的 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 。 * exptime：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数 * noreply（可选）： 该参数告知服务器不需要返回数据 * value：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）   输出信息：   * STORED：保存成功后输出。 * NOT\_STORED：执行替换失败后输出。   例如：  add mykey 0 900 5  hello  STORED  get mykey  VALUE mykey 0 5  hello  END  replace mykey 0 900 4  join  STORED  get mykey  VALUE mykey 0 4  join  END |

## append 命令

Memcached append 命令用于向已存在 key(键) 的 value(数据值) 后面追加数据 。

语法：

|  |
| --- |
| append key flags exptime bytes [noreply]  value  参数说明：   * key：键值 key-value 结构中的 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 。 * exptime：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数 * noreply（可选）： 该参数告知服务器不需要返回数据 * value：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）   输出信息：   * **STORED**：保存成功后输出。 * **NOT\_STORED**：该键在 Memcached 上不存在。 * **CLIENT\_ERROR**：执行错误。   例如：  set runoob 0 900 9  memcached  STORED  get runoob  VALUE runoob 0 9  memcached  END  append runoob 0 900 5  redis  STORED  get runoob  VALUE runoob 0 14  memcachedredis  END |

## prepend 命令

Memcached prepend 命令用于向已存在 key(键) 的 value(数据值) 前面追加数据 。

语法：

|  |
| --- |
| prepend key flags exptime bytes [noreply]  value  参数说明：   * key：键值 key-value 结构中的 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 。 * exptime：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数 * noreply（可选）： 该参数告知服务器不需要返回数据 * value：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）   输出信息：   * STORED：保存成功后输出。 * NOT\_STORED：该键在 Memcached 上不存在。 * CLIENT\_ERROR：执行错误   例如：  set runoob 0 900 9  memcached  STORED  get runoob  VALUE runoob 0 9  memcached  END  prepend runoob 0 900 5  redis  STORED  get runoob  VALUE runoob 0 14  redismemcached  END |

## CAS命令

Memcached CAS（Check-And-Set 或 Compare-And-Swap） 命令用于执行一个"检查并设置"的操作。它仅在当前客户端最后一次取值后，该key 对应的值没有被其他客户端修改的情况下， 才能够将值写入。检查是通过cas\_token参数进行的， 这个参数是Memcach指定给已经存在的元素的一个唯一的64位值。

语法：

|  |
| --- |
| cas key flags exptime bytes unique\_cas\_token [noreply]  value  参数说明：   * key：键值 key-value 结构中的 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 。 * exptime：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数 * unique\_cas\_token通过 gets 命令获取的一个唯一的64位值。 * noreply（可选）： 该参数告知服务器不需要返回数据 * value：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）   输出信息：   * STORED：保存成功后输出。 * ERROR：保存出错或语法错误。 * EXISTS：在最后一次取值后另外一个用户也在更新该数据。 * NOT\_FOUND：Memcached 服务上不存在该键值。   例如：  cas tp 0 900 9  ERROR <− 缺少 token  cas tp 0 900 9 2  memcached  NOT\_FOUND <− 键 tp 不存在  set tp 0 900 9  memcached  STORED  gets tp  VALUE tp 0 9 1  memcached  END  cas tp 0 900 5 1  redis  STORED  get tp  VALUE tp 0 5  redis  END |

# Memcached查找命令

## get命令

Memcached get 命令获取存储在 key(键) 中的 value(数据值) ，如果 key 不存在，则返回空。

语法：

|  |
| --- |
| get key  get key1 key2 key3  参数说明：  key：键值 key-value 结构中的 key，用于查找缓存值。  例如：  set runoob 0 900 9  memcached  STORED  get runoob  VALUE runoob 0 9  memcached  END |

## gets 命令

Memcached gets 命令获取带有 CAS 令牌存 的 **value(数据值)** ，如果 key 不存在，则返回空。

语法：

|  |
| --- |
| gets key  gets key1 key2 key3  参数说明：  key：键值 key-value 结构中的 key，用于查找缓存值。  例如：  set runoob 0 900 9  memcached  STORED  gets runoob  VALUE runoob 0 9 1  memcached  END |

## delete 命令

Memcached delete 命令用于删除已存在的 key(键)。

语法：

|  |
| --- |
| delete key [noreply]  gets key1 key2 key3  参数说明：   * key：键值 key-value 结构中的 key，用于查找缓存值。 * noreply（可选）： 该参数告知服务器不需要返回数据   输出信息：   * DELETED：删除成功。 * ERROR：语法错误或删除失败。 * NOT\_FOUND：key 不存在。   例如：  set runoob 0 900 9  memcached  STORED  get runoob  VALUE runoob 0 9  memcached  END  delete runoob  DELETED  get runoob  END |

## incr 与 decr 命令

Memcached incr 与 decr 命令用于对已存在的 key(键) 的数字值进行自增或自减操作。incr 与 decr 命令操作的数据必须是十进制的32位无符号整数。

如果 key 不存在返回 **NOT\_FOUND**，如果键的值不为数字，则返回 **CLIENT\_ERROR**，其他错误返回 **ERROR**。

语法：

|  |
| --- |
| incr key increment\_value  decr key decrement\_value  参数说明：   * key：键值 key-value 结构中的 key，用于查找缓存值。 * increment\_value： 增加的数值。 * decrement\_value： 减少的数值。   输出信息：   * NOT\_FOUND：key 不存在。 * CLIENT\_ERROR：自增值不是对象。 * ERROR其他错误，如语法错误等   例如：  set visitors 0 900 2  10  STORED  get visitors  VALUE visitors 0 2  10  END  incr visitors 5  15  get visitors  VALUE visitors 0 2  15  END |

# Memcached统计命令

## stats 命令

Memcached stats 命令用于返回统计信息例如 PID(进程号)、版本号、连接数等。

语法：stats

状态信息：

|  |
| --- |
| pid： memcache服务器进程ID  uptime：服务器已运行秒数  time：服务器当前Unix时间戳  version：memcache版本  pointer\_size：操作系统指针大小  rusage\_user：进程累计用户时间  rusage\_system：进程累计系统时间  curr\_connections：当前连接数量  total\_connections：Memcached运行以来连接总数  connection\_structures：Memcached分配的连接结构数量  cmd\_get：get命令请求次数  cmd\_set：set命令请求次数  cmd\_flush：flush命令请求次数  get\_hits：get命令命中次数  get\_misses：get命令未命中次数  delete\_misses：delete命令未命中次数  delete\_hits：delete命令命中次数  incr\_misses：incr命令未命中次数  incr\_hits：incr命令命中次数  decr\_misses：decr命令未命中次数  decr\_hits：decr命令命中次数  cas\_misses：cas命令未命中次数  cas\_hits：cas命令命中次数  cas\_badval：使用擦拭次数  auth\_cmds：认证命令处理的次数  auth\_errors：认证失败数目  bytes\_read：读取总字节数  bytes\_written：发送总字节数  limit\_maxbytes：分配的内存总大小（字节）  accepting\_conns：服务器是否达到过最大连接（0/1）  listen\_disabled\_num：失效的监听数  threads：当前线程数  conn\_yields：连接操作主动放弃数目  bytes：当前存储占用的字节数  curr\_items：当前存储的数据总数  total\_items：启动以来存储的数据总数  evictions：LRU释放的对象数目  reclaimed：已过期的数据条目来存储新数据的数目 |

## stats items 命令

memcached stats items 命令用于显示各个 slab 中 item 的数目和存储时长(最后一次访问距离现在的秒数)。

语法：stats items

## stats slabs 命令

Memcached stats slabs 命令用于显示各个slab的信息，包括chunk的大小、数目、使用情况等。

语法：stats slabs

## stats sizes 命令

memcached stats sizes 命令用于显示所有item的大小和个数。

该信息返回两列，第一列是 item 的大小，第二列是 item 的个数。

语法：stats sizes

## flush\_all 命令

Memcached flush\_all 命令用于用于清理缓存中的所有 **key=>value(键=>值)** 对。该命令提供了一个可选参数 **time**，用于在制定的时间后执行清理缓存操作。

语法：

|  |
| --- |
| flush\_all [time] [noreply]  例如：  set runoob 0 900 9  memcached  STORED  get runoob  VALUE runoob 0 9  memcached  END  flush\_all  OK  get runoob  END |

# 在java中使用Memcached

package test;

import java.net.InetSocketAddress;

import java.util.concurrent.Future;

import net.spy.memcached.CASResponse;

import net.spy.memcached.CASValue;

import net.spy.memcached.MemcachedClient;

public class MemcachedJava {

public static void main(String[] args) {

try {

// 连接服务器

MemcachedClient mcc = new MemcachedClient(new InetSocketAddress(

"172.16.128.218", 11211));

System.out.println("Memcached connection successful!");

// 清空所有缓存

mcc.flush();

// set存储数据

Future fo = mcc.set("runoob", 900, "runoob");

// 存储状态

System.out.println(fo.get());

// get输出数据

System.out.println("set runoob:" + mcc.get("runoob"));

// 使用add添加数据

fo = mcc.add("runoob", 900, "memcached");

System.out.println("add runoob:" + mcc.get("runoob"));

fo = mcc.add("runoob1", 900, "runoob1");

System.out.println("add new runoob:" + mcc.get("runoob1"));

// 使用replace替换

fo = mcc.replace("runoob", 900, "reset runoob");

System.out.println("replace runoob:" + mcc.get("runoob"));

// append追加内容到值后面

fo = mcc.append("runoob1", " appended");

System.out.println("append runoob1:" + mcc.get("runoob1"));

// prepend追加内容在值前面

fo = mcc.set("runoob1", 900, "runoob1");

fo = mcc.prepend("runoob1", "prepending something in front of ");

System.out.println("prepend runoob1:" + mcc.get("runoob1"));

System.out.println();

// cas:check and set

fo = mcc.set("runoob", 900, "runoob");

// 通过 gets 方法获取 CAS token（令牌）

CASValue casValue = mcc.gets("runoob");

// 输出 CAS token（令牌） 值

System.out.println("CAS token:" + casValue);

// 尝试使用cas方法来更新数据

CASResponse casresp = mcc.cas("runoob", casValue.getCas(), 900,

"Largest Tutorials-Library");

// 输出 CAS 响应信息

System.out.println("CAS Response:" + casresp);

// 输出值

System.out.println("runoob value in cache - " + mcc.get("runoob"));

System.out.println();

//删除操作delete

fo = mcc.set("runoob", 900, "runoob");

System.out.println("runoob:"+mcc.get("runoob"));

System.out.print("执行删除,");

mcc.delete("runoob");

System.out.println("runoob:"+mcc.get("runoob"));

//自增自减运算Incr/Decr

fo = mcc.set("num", 900, "1000");

System.out.println("num="+mcc.get("num"));

System.out.println("after incr 111,num="+mcc.incr("num", 111));

System.out.println("after decr 112,num="+mcc.decr("num", 112));

mcc.shutdown();

} catch (Exception e) {

}

}

}