Memcache 集群部署和使用

Name： 曲中岭

Email:zlingqu@126.com

Q Q :441869115

第一章 简介

1.1 是什么

Memcached 是一个自由开源的，高性能，分布式内存对象缓存系统。

Memcached 是一种基于内存的 key-value 存储，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。

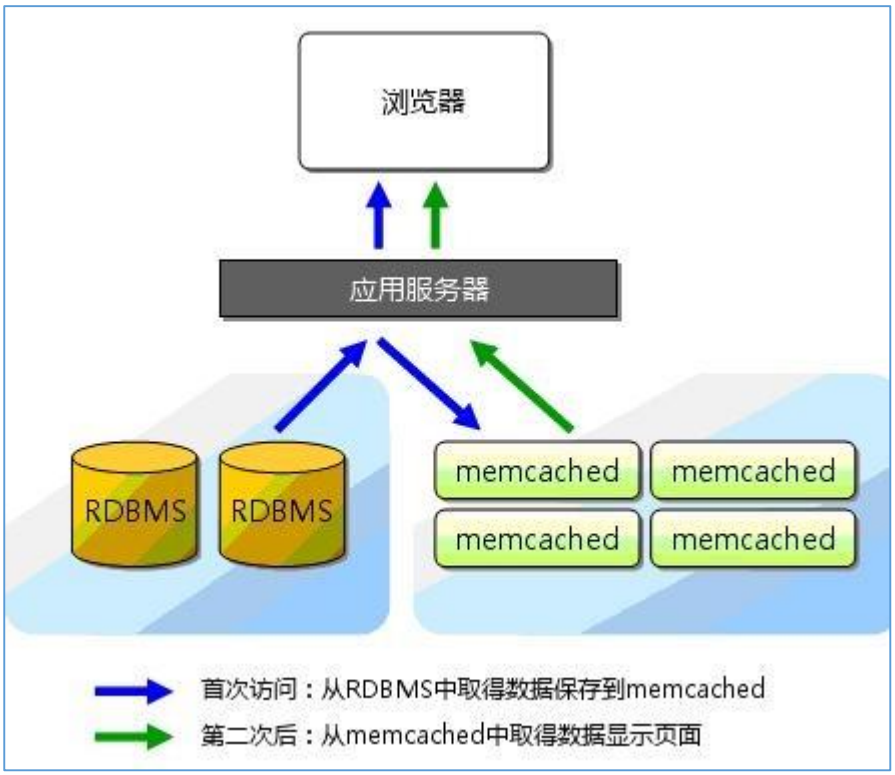
GitHub 地址：https://github.com/memcached/memcached

官网：http://memcached.org/

1.2 干什么

一般的使用目的是，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。

1.3 工作结构图



1.4 特征

memcached 作为高速运行的分布式缓存服务器，具有以下的特点。

- 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

1.5 支持的语言

许多语言都实现了连接 memcached 的客户端，仅仅 memcached 网站上列出的有：

- Perl
- PHP

- Python
- Ruby
- C#
- C/C++
- Lua
- 等等

第二章 单机部署

2.1 规划

OS：CentOS 6.6 x64
IP：10.1.5.201
Memcache：1.5.10
安装路径：/usr/local/memcached
PID 文件：/tmp/memcached.pid
PORT：8831
监听地址：0.0.0.0

2.2 安装依赖

yum install libevent libevent-devel

2.3 安装和部署

到下面
https://github.com/memcached/memcached/wiki/ReleaseNotes
选择合适版本

```
wget http://www.memcached.org/files/memcached-1.5.10.tar.gz
tar xf memcached-1.5.10.tar.gz
cd memcached-1.5.10
make
make install
```

2.4 启动

```
cd /usr/local/memcached/
./bin/memcached -u hadoop -p 8831 -m 4096 -c 256 -P /tmp/memcached.pid –d
```

其他启动参数：

```
-p <num> 设置 TCP 端口号(默认设置为: 11211)
-U <num> UDP 监听端口(默认: 11211, 0 时关闭，若不指定同 TCP-Port 相同)
-l <ip_addr> 绑定地址(默认是 0.0.0.0，即所绑定所有 IP
-c <num>       max simultaneous connections (default: 1024)最大并发连接数
-d 以 daemon 方式运行
-u <username> 绑定使用指定用户运行进程
-m <num> 允许最大内存用量，单位 M (默认: 64 MB)
```

-P <file> 将 PID 写入文件<file>，这样可以使得后边进行快速进程终止，需要与-d 一起使用

-n <num> 指定最小的 slab chunk 空间,默认 48B

-f <factor> 指定增长因子，默认 1.25

-S 启用 sasl 认证，注意，编译安装时必须支持 sasl 才可以

-t 线程数量，默认是 4，memcache 效率很高，大多数时候不需要更改

-V 查看版本

-h 查看帮助

启动后查看：

第三章 集群部署

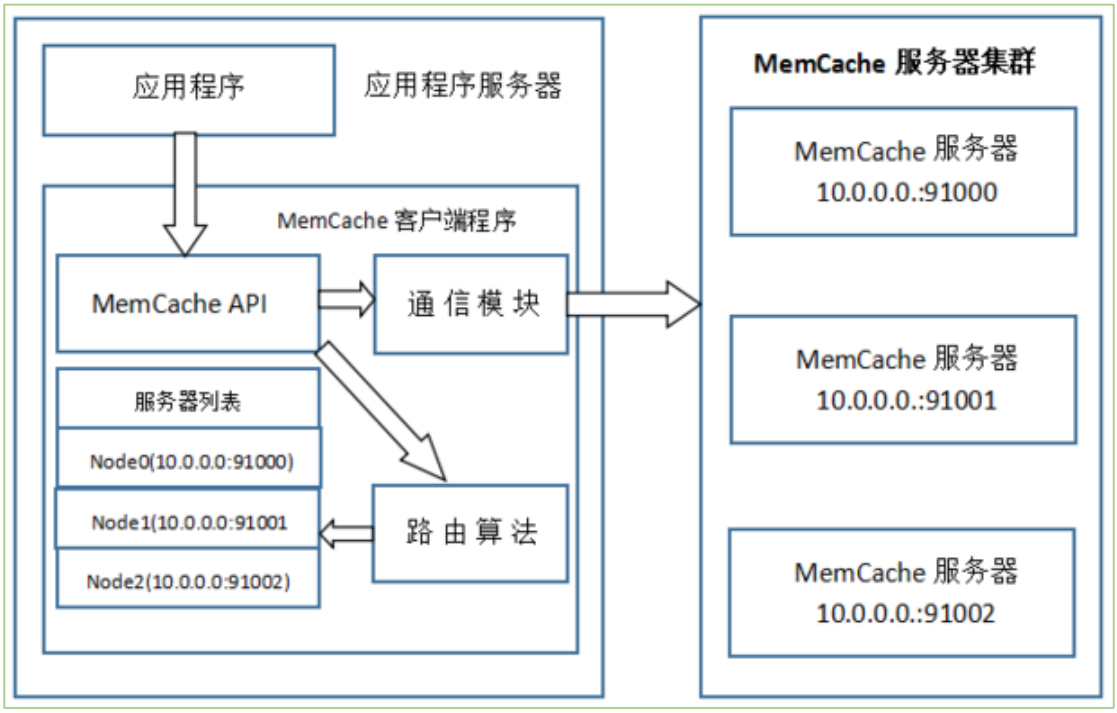
3.1 背景

Memcache 各 server 之间没有任何通讯，也不进行任何数据复制备份，所以说 memcache 自身并没有实现集群功能。其高可用完全是通过前端实现，比如可以借助第三方软件或者自己设计编程实现。

3.2 客户端实现 HA

这里的客户端指：memcached client for java 、spymemcached 、xmemcached 等，就是各语言用来连接 memcache 的通用组件。其功能和 java 的 AbstractRoutingDataSource 组件有异曲同工之妙。

客户端实现 HA 的工作原理图如下：



其中路由算法有取余算法、一致性 hash 算法等，可参考
https://blog.csdn.net/fdipzone/article/details/7170045
这里不再详细介绍。

优点：

1） 数据分散存储，降低单台压力，提高性能

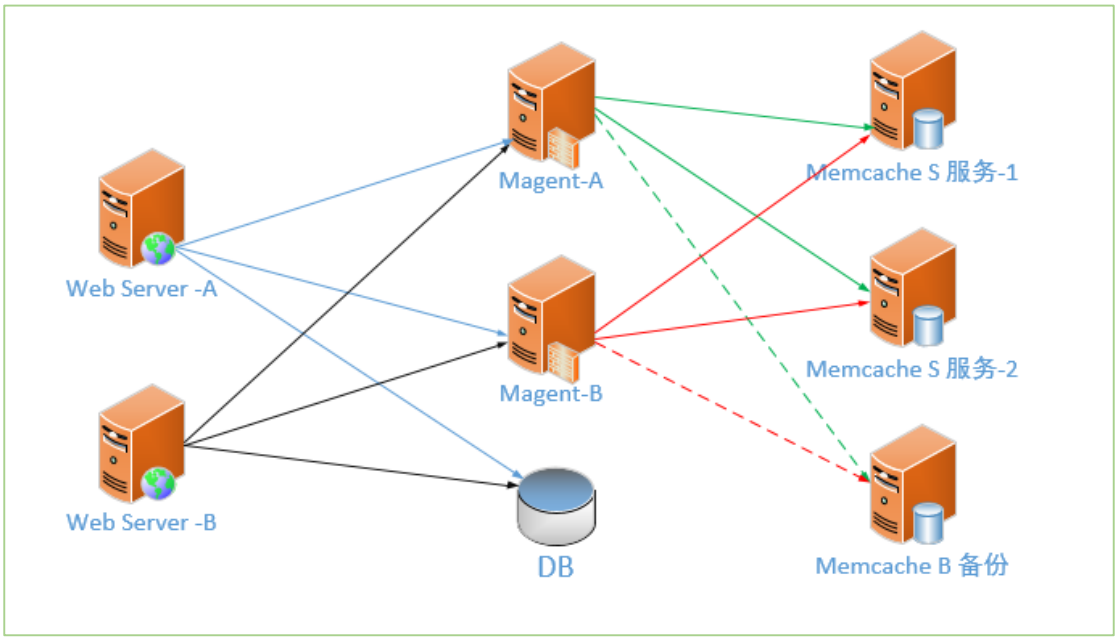
缺点：

1） 单点失效后，需要重新计算，会使大多数原有数据失效

2） 增加新节点，也要重新计算，同样会使大多数原有数据失效

3.3 第三方软件实现 HA

这里说一种比较常见的第三软件，即 Memagent(简写 magent)，是 google 开源出来的，项目地址 http://code.google.com/p/memagent/ 。其作用类似于 mysql-proxy 对于 mysql 的角色，即在程序和数据之间再加一层。
magent 工作原理图如下:



优点：
1） 由 magent 实现数据分散存储，程序只负责读写即可
2） magent 还可以设置备份节点，实现自动故障转移，这对程序是透明的
缺点:
1） 故障节点恢复后，无数据，也不会自动从 backup 节点同步数据过来，会造成这个节点的数据失效
2） 每次写入数据，都会同步写到备份节点，增大开销。备份节点保留有工作节点的所有数据

3.3.1 规划

Magent-A : 10.1.5.201:8830
Magent-B : 10.1.5.202:8830
Memcache S-1: 10.1.5.201:8831
Memcache S-2: 10.1.5.202:8831
Memcache B: 10.1.5.203:8831

3.3.2 memcache 部署

按照规划需要在 10.1.5.201-203 三台机器上部署，部署过程参考第二章。

3.3.3 magent 部署

按照规划需要在 10.1.5.201-202 两台机器上部署。

wget -c http://memagent.googlecode.com/files/magent-0.6.tar.gz
如果访问不了，就到 csdn 上下载，有很多资源。

```
mkdir /usr/local/magent
tar xf magent-0.6.tar.gz -C /usr/local/magent
cd /usr/local/magent
make
```

报错 1:

```
[root@hadoop02 ~]# mkdir /usr/local/magent
[root@hadoop02 ~]# tar xf magent.tar.gz -C /usr/local/magent
[root@hadoop02 ~]# cd !$
cd /usr/local/magent
[root@hadoop02 magent]# ls
ketama.c  ketama.h  magent.c  Makefile
[root@hadoop02 magent]# make
gcc -Wall -g -O2 -I/usr/local/include -m64 -c -o magent.o magent.c
magent.c: 在函数‘writev_list’中:
magent.c:729: 错误：‘SSIZE_MAX’未声明(在此函数内第一次使用)
magent.c:729: 错误：(即使在一个函数内多次出现，每个未声明的标识符在其
magent.c:729: 错误：所在的函数内也只报告一次。)
make: *** [magent.o] 错误 1
[root@hadoop02 magent]#
[root@hadoop02 magent]#
```

解决方法，在 ketama.h 文件的前面加入以下三行

```
#ifndef SSIZE_MAX
#define SSIZE_MAX      32767
#endif
```

```
[root@hadoop02 magent]#
[root@hadoop02 magent]# cat ketama.h
#ifndef _KETAMA_H
#define _KETAMA_H

#ifndef SSIZE_MAX

#define SSIZE_MAX       32767

#endif

struct dot {
        unsigned int point;
        int srvid;
};

struct ketama {
        unsigned int numpoints;
        struct dot *dot;

        int count;
        char **name;
        int *weight;
        int totalweight;
};

int create_ketama(struct ketama *, int);
void free_ketama(struct ketama *);
int get_server(struct ketama *, const char *);
#endif
[root@hadoop02 magent]#
[root@hadoop02 magent]#
```

再次 make，报错 2，gcc: /usr/lib64/libm.a：没有那个文件或目录：

```
[root@hadoop02 magent]# make
gcc -Wall -g -O2 -I/usr/local/include -m64 -c -o magent.o magent.c
gcc -Wall -g -O2 -I/usr/local/include -m64 -c -o ketama.o ketama.c
gcc -Wall -g -O2 -I/usr/local/include -m64 -o magent magent.o ketama.o /usr/lib64/libevent.a /usr/lib64/libm.a
gcc: /usr/lib64/libm.a：没有那个文件或目录
make: *** [magent] 错误 1
```

解决方法：

```
cp /usr/lib64/libm.so /usr/lib64/libm.a
```

再次 make，报错 3，undefined reference to `clock_gettime'

```
[root@hadoop02 magent]# cp /usr/lib64/libm.so /usr/lib64/libm.a
[root@hadoop02 magent]# make
gcc -Wall -g -O2 -I/usr/local/include -m64 -o magent magent.o ketama.o /usr/lib64/libevent.a /usr/lib64/libm.a
/usr/lib64/libevent.a(event.o): In function `gettime':
(.text+0x439): undefined reference to `clock_gettime'
/usr/lib64/libevent.a(event.o): In function `event_base_new':
(.text+0x6fa): undefined reference to `clock_gettime'
collect2: ld 返回 1
make: *** [magent] 错误 1
```

输入以下语句解决：

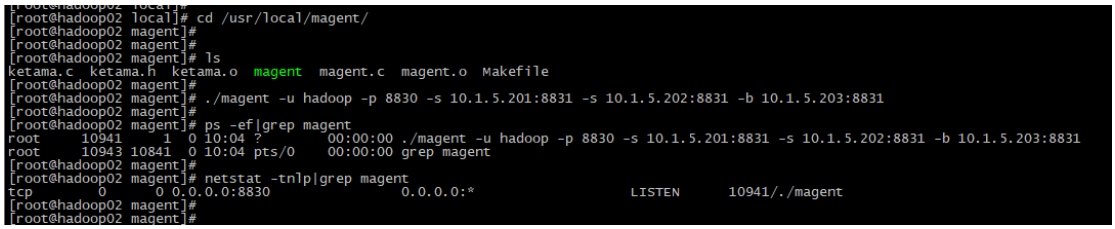
```
sed -i 's/-Wall/-lrt -Wall/g' Makefile
```

再次 make，看到生成可执行文件 magent，即表明编译成功了。

```
[root@hadoop02 magent]# ls
ketama.c  ketama.h  ketama.o  magent.c  magent.o  Makefile
[root@hadoop02 magent]#
[root@hadoop02 magent]# sed -i 's/-Wall/-lrt -Wall/g' Makefile
[root@hadoop02 magent]#
[root@hadoop02 magent]# make
gcc -lrt -Wall -g -O2 -I/usr/local/include -m64 -o magent magent.o ketama.o /usr/lib64/libevent.a /usr/lib64/libm.a
[root@hadoop02 magent]# ls
ketama.c  ketama.h  ketama.o  magent  magent.c  magent.o  Makefile
[root@hadoop02 magent]#
```

启动 magent

```
cd /usr/local/magent
./magent -u hadoop -p 8830 -s 10.1.5.201:8831 -s 10.1.5.202:8831 -b 10.1.5.203:8831
```



使用./magent –h 可以查看该命令的帮助信息:

-h this message
-u uid
-g gid
-p port, default is 11211. (0 to disable tcp support)
-s ip:port, set memcached server ip and port
-b ip:port, set backup memcached server ip and port
-l ip, local bind ip address, default is 0.0.0.0
-n number, set max connections, default is 4096
-D don't go to background
-k use ketama key allocation algorithm
-f file, unix socket path to listen on. default is off
-i number, set max keep alive connections for one memcached server, default is 20
-v verbose

第四章 集群测试

4.1 查看集群状态

echo stats | nc 10.1.5.201 8830
echo stats | nc 10.1.5.202 8830

```
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo stats | nc 10.1.5.201 8830
memcached agent v0.6
matrix 1 -> 10.1.5.201:8831, pool size 1
matrix 2 -> 10.1.5.202:8831, pool size 0
END
[root@hadoop01 ~]# echo stats | nc 10.1.5.202 8830
memcached agent v0.6
matrix 1 -> 10.1.5.201:8831, pool size 0
matrix 2 -> 10.1.5.202:8831, pool size 0
END
[root@hadoop01 ~]#
```

使用 stats 查看 magent 可以看到有两个主节点信息，虽然未显示备份节点，但其实是存在的，下面继续测试。

4.2 负载均衡和备份节点同步

```
[root@hadoop01 ~]# echo stats| nc 10.1.5.201 8831|grep curr_items
STAT curr_items 0
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo stats| nc 10.1.5.202 8831|grep curr_items
STAT curr_items 0
[root@hadoop01 ~]# echo stats| nc 10.1.5.203 8831|grep curr_items
STAT curr_items 0
[root@hadoop01 ~]#
[root@hadoop01 ~]# telnet 10.1.5.201 8830
Trying 10.1.5.201...
Connected to 10.1.5.201.
Escape character is '^]'.
set name1 0 0 5
name1
STORED
set name2 0 0 5
name2
STORED
quit
Connection closed by foreign host.
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo stats| nc 10.1.5.201 8831|grep curr_items
STAT curr_items 1
[root@hadoop01 ~]# echo stats| nc 10.1.5.202 8831|grep curr_items
STAT curr_items 1
[root@hadoop01 ~]# echo stats| nc 10.1.5.203 8831|grep curr_items
STAT curr_items 2
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo "gets name1"| nc 10.1.5.203 8831
VALUE name1 0 5 6
name1
END
[root@hadoop01 ~]# echo "gets name2"| nc 10.1.5.203 8831
VALUE name2 0 5 7
name2
END
[root@hadoop01 ~]# echo "gets name2"| nc 10.1.5.201 8831
VALUE name2 0 5 30
name2
END
[root@hadoop01 ~]# echo "gets name1"| nc 10.1.5.202 8831
VALUE name1 0 5 3
name1
END
```

```
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo "gets name1"| nc 10.1.5.201 8831
END
[root@hadoop01 ~]# echo "gets name2"| nc 10.1.5.202 8831
END
[root@hadoop01 ~]#
```

```
telnet 10.1.5.201 8830
Trying 10.1.5.201...
Connected to 10.1.5.201.
Escape character is '^]'.
gets name1
VALUE name1 0 5 3
name1
END
gets name2
VALUE name2 0 5 30
name2
END
quit
Connection closed by foreign host.
[root@hadoop01 ~]#
[root@hadoop01 ~]# telnet 10.1.5.202 8830
Trying 10.1.5.202...
Connected to 10.1.5.202.
Escape character is '^]'.
gets name1
VALUE name1 0 5 3
name1
END
gets name2
VALUE name2 0 5 30
name2
END
```

测试过程如上图，测试流程是

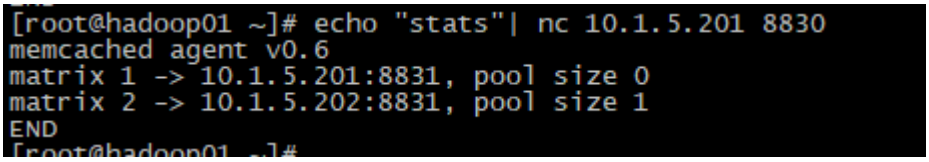
1） 初始状态，三个 memcache 都没有 key
2） 通过 magent 添加两个 key，name1、name2
3） 再次查看三个 memcache，发现 name1 被存到了 202、name2 被存到了 201，证明该集群有负载均衡在多用
4） 203 同时存有 name1、name2，证明备份节点是有效的
5） 通过 201、202 的 magent，也都可以查到 name1、name2，证明 magent 代理节点是有效的

4.3 故障转移

```
[root@hadoop01 ~]#
[root@hadoop01 ~]# ps -ef|grep memcache
hadoop   25209     1  0 Sep28 ?        00:00:35 /usr/local/memc
root     32107 31691  0 10:52 pts/0    00:00:00 grep memcache
[root@hadoop01 ~]#
[root@hadoop01 ~]# kill 25209
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo "gets name1"| nc 10.1.5.203 8831
VALUE name1 0 5 6
name1
END
[root@hadoop01 ~]# echo "gets name2"| nc 10.1.5.203 8831
VALUE name2 0 5 7
name2
END
[root@hadoop01 ~]# echo "gets name2"| nc 10.1.5.202 8831
END
[root@hadoop01 ~]# echo "gets name1"| nc 10.1.5.202 8831
VALUE name1 0 5 3
name1
END
[root@hadoop01 ~]# echo "gets name1"| nc 10.1.5.201 8831
[root@hadoop01 ~]# echo "gets name2"| nc 10.1.5.201 8831
```

```
[root@hadoop01 ~]# telnet 10.1.5.201 8830
Trying 10.1.5.201...
Connected to 10.1.5.201.
Escape character is '^]'.
gets name1
VALUE name1 0 5 3
name1
END
gets name2
VALUE name2 0 5 7
name2
END
quit
Connection closed by foreign host.
[root@hadoop01 ~]# telnet 10.1.5.202 8830
Trying 10.1.5.202...
Connected to 10.1.5.202.
Escape character is '^]'.
gets name1
VALUE name1 0 5 3
name1
END
gets name2
VALUE name2 0 5 7
name2
END
```

```
[root@hadoop01 ~]# telnet 10.1.5.201 8830
Trying 10.1.5.201...
Connected to 10.1.5.201.
Escape character is '^]'.
set name3 0 0 5
name3
STORED
quit
Connection closed by foreign host.
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo "gets name3"| nc 10.1.5.202 8831
VALUE name3 0 5 4
name3
END
[root@hadoop01 ~]# telnet 10.1.5.201 8830
Trying 10.1.5.201...
Connected to 10.1.5.201.
Escape character is '^]'.
set name4 0 0 5
name4
STORED
quit
Connection closed by foreign host.
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo "gets name3"| nc 10.1.5.202 8831
VALUE name3 0 5 4
name3
END
[root@hadoop01 ~]# echo "gets name4"| nc 10.1.5.202 8831
END
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo "gets name3"| nc 10.1.5.203 8831
VALUE name3 0 5 8
name3
END
[root@hadoop01 ~]# echo "gets name4"| nc 10.1.5.203 8831
VALUE name4 0 5 10
name4
END
```



测试过程如上图，测试流程是

1） 停止 201 的 8831，即 memcache
2） 203 的 8831，可以查到 name1、name2
3） 202 的 8831 可以查到 name1，查不到 name2（与停止前一致）
4） 201 的 8831 停止了，什么都查不到
5） 201、202 的 8830（magent）可以查到 name1、name2
6） 通过 201 的 8830 存入 name3、name4 两个 key，202 的 8831 存入了 name3，备份的 203 同时存入了 name3、name4，即 203 仍然是备份节点的作用
7） 通过 stats 命令查看，201、202 仍然是主节点，再次证明 203 仍然是充当备份节点的作用

经过以上步骤，可以看出在 201 节点挂掉后，整个集群仍然是可用的。因为由于备份节点保留有主节点的完成数据信息，所以实现了故障转移。

进一步测试:

停止 202 的 8831，可得到与上面类似的结果，即是说，只要备份节点存在，整个集群就可以认为是可用的，这是一种比较特殊的集群。

4.4 故障恢复

```
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo stats| nc 10.1.5.201 8831|grep curr_items
STAT curr_items 0
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo stats| nc 10.1.5.202 8831|grep curr_items
STAT curr_items 0
[root@hadoop01 ~]#
[root@hadoop01 ~]# telnet 10.1.5.201 8830
Trying 10.1.5.201...
Connected to 10.1.5.201.
Escape character is '^]'.
gets name1
END
gets name2
END
gets name3
END
gets name4
END
quit
Connection closed by foreign host.
[root@hadoop01 ~]#
[root@hadoop01 ~]# echo "gets name1" |nc 10.1.5.203 8831
VALUE name1 0 5 6
name1
END
[root@hadoop01 ~]# echo "gets name2" |nc 10.1.5.203 8831
VALUE name2 0 5 7
name2
END
[root@hadoop01 ~]# echo "gets name3" |nc 10.1.5.203 8831
VALUE name3 0 5 8
name3
END
[root@hadoop01 ~]# echo "gets name4" |nc 10.1.5.203 8831
VALUE name4 0 5 10
name4
END
```

```
[root@hadoop01 ~]# !tel
telnet 10.1.5.201 8830
Trying 10.1.5.201...
Connected to 10.1.5.201.
Escape character is '^]'.
get name1
END
set name1 0 0 5
namen
STORED
get name1
VALUE name1 0 5
namen
END
quit
Connection closed by foreign host.
[root@hadoop01 ~]# echo "get name1"|nc 10.1.5.203 8831
VALUE name1 0 5
namen
END
[root@hadoop01 ~]# echo "get name1"|nc 10.1.5.201 8831
END
[root@hadoop01 ~]# echo "get name1"|nc 10.1.5.202 8831
VALUE name1 0 5
namen
END
```

测试过程如上图，测试流程是

1） 重新启动 201、202 的 8831
2） 发现 201、202 里面没有任何 key，什么信息都查不到，这与 memcache 内存存储特性有关
3） 203 里面由于保留有完整的数据信息，所以都可以查到
4） 主节点都恢复了，通过 magent 查询时，会从主节点获取数据，所以原有数据都查不到了，这也是这种集群的一个重大缺陷
5） 重新添加 name1，发现可以查到，且备份节点也获得了更新
6）从新添加的 name1 被存到了 202，这与之前的分配方法是一样的了，即同样的 key 被分配到哪个 memcache，仅仅是 magent 启动时-s 参数的顺序决定的。

4.5 注意事项

1） magent 在分配 key 到 memcached 上时只是简单的使用散列余数算法
2） 如果有多个 magent，其启动命令中-s 参数的顺序应该是一致的，否则将导致混乱

附录 常用命令

5.1 连接和退出

对于 java、php、python 等连接 memcache，使用对应的 API 即可，这里直接使用 telnet 命令行进行演示 set 和 get 命令。

```
[root@hadoop01 memcached]#
[root@hadoop01 memcached]# telnet 127.0.0.1 8831
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set name 0 30 4
quzl
STORED
get name
VALUE name 0 4
quzl
END
get name
VALUE name 0 4
quzl
END

ERROR
get name
VALUE name 0 4
quzl
END

ERROR

ERROR
get name
END
```

quit 退出

5.2 存储命令

5.2.1 set

set 命令用于将 value(数据值) 存储在指定的 key(键) 中，如果已经存在，则更新 value。

语法：

```
set key flags exptime bytes [noreply]
value
```

- **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）

实例：

```
set name 0 90 4
quzl
STORED
```

5.2.2 add

同 set 命令类似，唯一的区别是，key 必须存在且未过期，则不会更新 value

5.2.3 replace

同 set 命令类似，唯一的区别是，只有当 key 存在且未过期才更新，否则返回失败

5.2.4 append

同 set 命令类型，在 value 的尾部追加数据，前提是 key 必须存在且未过期

5.2.5 prepend

同 append 命令类似，唯一的区别是在 value 的前面追加数据

5.2.6 cas

语法：

```
cas key flags exptime bytes unique_cas_token [noreply]
value
```

cas 命令对比 set 命令，最后面多了一个 unique_cas_token 参数，

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。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，改值可理解为 "版本" 标识符。

如下，通过 gets 得到的最后一位 15 就是这个值，这个值是全局唯一的，比如新建了另

一个 key 占用了 16，如果再修改 name，则该值会变成 17。

```
gets name
VALUE name 0 4 15
hell
END
```

cas 使用方法如下：

```
gets name
VALUE name 0 4 15
hell
END
cas name 0 0 5 14
quzll
EXISTS
cas name 0 0 5 15
quzll
STORED
gets name
VALUE name 0 5 16
quzll
END
```

cas 命令设计的目的在于多客户端操作同一个 key 的冲突，即使同一个客户端，由于 memcache 是多线程的，也会有这个问题。比如一个 key 是：name:A，客户端 1 操作尾部加 B，期望得到 name:AB，客户端 2 操作尾部加 C，期望得到 name:AC，实际的结果应该却是 ABC 或者 ACB。cas 就是为了解决这样的类似问题的。

cas 直接操作时直接指定 unique_cas_token（版本号），如果期间有变化，则返回失败。

5.3 查找命令

5.3.1 get

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。可同时查多个。

```
get key1 [key2] [key3]
```

5.3.2 gets

同 get 命令类似，唯一的区别是会同时返回 value 的 CAS 令牌。

5.3.3 delete

删除已经存在的 key

```
delete key [noreply]
```

5.3.4 incr 与 decr

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，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|decr key increment_value
```

- **key**：键值 key-value 结构中的 key，用于查找缓存值。
- **increment_value**： 增加/减少的数值。

5.4 统计命令

5.4.1stats

stats 命令用于返回统计信息例如 PID(进程号)、版本号、连接数等。直接输入 stats 命令，可以查看到很多参数，在 1.5.10 版本中多了很多参数，这些参数的意思还未搞清楚。

参数	值	描述
pid	25209	memcache 服务器进程 ID
uptime	65835	服务器已运行秒数
time	1538184423	服务器当前 Unix 时间戳
version	1.5.10	memcache 版本
libevent	1.4.13-stable	libevent 版本
pointer_size	64	操作系统指针大小
rusage_user	11.406265	进程累计用户时间
rusage_system	4.256352	进程累计系统时间
max_connections	256	最大同时连接数
curr_connections	2	当前连接数量
total_connections	14	Memcached 运行以来连接总数
rejected_connections	0	拒绝的连接总数
connection_structures	4	Memcached 分配的连接结构数量
reserved_fds	20	
cmd_get	44	get 命令请求次数
cmd_set	30	set 命令请求次数
cmd_flush	0	flush 命令请求次数
cmd_touch	0	
get_hits	35	get 命令命中次数
get_misses	9	get 命令未命中次数
get_expired	1	
get_flushed	0	
delete_misses	1	delete 命令未命中次数
delete_hits	2	delete 命令命中次数
incr_misses	0	incr 命令未命中次数
incr_hits	1	incr 命令命中次数
decr_misses	0	decr 命令未命中次数
decr_hits	0	decr 命令命中次数
cas_misses	0	cas 命令未命中次数
cas_hits	3	cas 命令命中次数
cas_badval	2	使用擦拭次数
touch_hits	0	
touch_misses	0	
auth_cmds	0	认证命令处理的次数
auth_errors	0	认证失败数目
bytes_read	1506	读取总字节数
bytes_written	7379	发送总字节数
limit_maxbytes	4294967296	分配的内存总大小（字节）
accepting_conns	1	服务器是否达到过最大连接（0/1）
listen_disabled_num	0	失效的监听数
time_in_listen_disabled_us	0	

threads	4	当前线程数
conn_yields	0	连接操作主动放弃数目
hash_power_level	16	
hash_bytes	524288	
hash_is_expanding	0	
slab_reassign_rescues	0	
slab_reassign_chunk_rescues	0	
slab_reassign_evictions_nomem	0	
slab_reassign_inline_reclaim	0	
slab_reassign_busy_items	0	
slab_reassign_busy_deletes	0	
slab_reassign_running	0	
slabs_moved	0	
lru_crawler_running	0	
lru_crawler_starts	11985	
lru_maintainer_juggles	128427	
malloc_fails	0	
log_worker_dropped	0	
log_worker_written	0	
log_watcher_skipped	0	
log_watcher_sent	0	
bytes	199	当前存储占用的字节数
curr_items	3	当前存储的数据总数
total_items	22	启动以来存储的数据总数
slab_global_page_pool	0	
expired_unfetched	3	
evicted_unfetched	0	
evicted_active	0	
evictions	0	LRU 释放的对象数目
reclaimed	8	已过期的数据条目来存储新数据的数目
crawler_reclaimed	0	
crawler_items_checked	34	
lrutail_reflocked	34	
moves_to_cold	37	
moves_to_warm	16	
moves_within_lru	0	
direct_reclaims	0	
lru_bumps_dropped	0	

5.4.2 stats items

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)。

5.4.3 stats slabs

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。

5.4.4 flush_all

flush_all 命令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]
```