Redis集群的安装

2017年12月13日　　16:27

集群环境安装

安装ruby(如果使用虚拟机的empty文件无需安装ruby)

因为集群命令文件需要ruby语言的支持

wget包后解压

文件中有2个地址,需要使用源站地址

在linux中调用wget命令 后面使用双引号"" 复制粘贴源站地址在里面可以获取tar包

```
[root@10-9-62-65 home]# mkdir software
[root@10-9-62-65 home]# wget "http://jt-xxw.cn-bj.ufileos.com/ruby-2.3.1.tar.gz?UCloudPublicKey=ucl
oudzhaodong%40tarena.com.cn14442715760002074868321&Signature=T%2FJDeLcF5JHA5h8OaR3G40Y30NY%3D&Expir
es=1513143170"
--2017-12-13 13:03:53--  http://jt-xxw.cn-bj.ufileos.com/ruby-2.3.1.tar.gz?UCloudPublicKey=ucloudzh
aodong%40tarena.com.cn14442715760002074868321&Signature=T%2FJDeLcF5JHA5h8OaR3G40Y30NY%3D&Expires=15
13143170
Resolving jt-xxw.cn-bj.ufileos.com... 106.38.254.244, 106.38.254.242, 106.38.254.229, ...
Connecting to jt-xxw.cn-bj.ufileos.com|106.38.254.244|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 17797997 (17M) [application/gzip]
```

到解压目录执行编译

```
[root@10-9-62-65 home]# tar -xf ruby-2.3.1.tar.gz\?UCloudPublicKey\=ucloudzhaodong\@tarena.com.cn14
4427157600020748683 21\&Signature\=T%2FJDeLcF5JHA5h8OaR3G40Y30NY\=\&Expires\=1513143170
```

ls查看发现ruby解压成功

```
[root@10-9-62-65 home]# ls
ruby-2.3.1
ruby-2.3.1.tar.gz?UCloudPublicKey=ucloudzhaodong@tarena.com.cn14442715760002074868321&Signature=T%2
FJDeLcF5JHA5h8OaR3G40Y30NY=&Expires=1513143170
software
[root@10-9-62-65 home]#
```

进入ruby目录

```
[root@10-9-62-65 home]# cd ruby-2.3.1
[root@10-9-62-65 ruby-2.3.1]#
```

#./configure 在当前根目录执行编译;

相当于c语言有各种打包,压缩过程,可以将打包压缩的内容解压

```
[root@10-9-39-219 software]# cd ruby-2.3.1/
[root@10-9-39-219 ruby-2.3.1]# ./configure
```

编译安装#make && make install(时间较长,可以在下课前说明一下然后执行)

```
checking pthread.h presence... yes
checking for pthread.h... yes
checking if make is GNU make... yes
checking for nroff... /usr/bin/nroff
.ext/include/x86_64-linux/ruby/config.h updated
configure: ruby library version = 2.3.0
configure: creating ./config.status
config.status: creating GNUmakefile
config.status: creating Makefile
config.status: creating ruby-2.3.pc
[root@10-9-62-65 ruby-2.3.1]# make && make install
```

编译的详细信息

C语言的编译安装

```
                              minitest 5.8.3
                              did_you_mean 1.0.0
                              net-telnet 0.1.1
                              rake 10.4.2
installing rdoc:              /usr/local/share/ri/2.3.0/system
installing capi-docs:         /usr/local/share/doc/ruby
[root@10-9-62-65 ruby-2.3.1]#
```

编译安装完成

yum安装gems

#yum -y install rubygems //rubygems是什么?

```
installing rdoc:              /usr/local/share/ri/2.3.0/system
installing capi-docs:         /usr/local/share/doc/ruby
[root@10-9-62-65 ruby-2.3.1]# yum install rubygems
```

RubyGems（简称 gems）是一个用于对 Ruby组件进行打包的 Ruby 打包系统。 它提供一个分发 Ruby 程序和库的标准格式，还提供一个管理程序包安装的工具。

简单理解就是ruby运行时,需要的各种插件都在gems里;

是一种技术支持;

过程中需要同意操作

```
================================================================
Install       6 Package(s)

Total download size: 3.2 M
Installed size: 11 M
Is this ok [y/N]:
```

需要输入Y同意

#gem install redis//安装redis接口包

下载支持redis的插件内容;

```
  ruby-rdoc.x86_64 0:1.8.7.374-5.el6

Complete!
[root@10-9-62-65 ruby-2.3.1]# gem install redis
```

```
Complete!
[root@10-9-62-65 ruby-2.3.1]# gem install redis
Fetching: redis-4.0.1.gem (100%)
Successfully installed redis-4.0.1
Parsing documentation for redis-4.0.1
Installing ri documentation for redis-4.0.1
Done installing documentation for redis after 0 seconds
1 gem installed
[root@10-9-62-65 ruby-2.3.1]#
```

安装完成

检查版本

#ruby -v

完成ruby环境的安装,就可以利用ruby的环境配置

执行.rb的文件

```
[root@10-9-62-65 home]# ruby -v
ruby 2.3.1p112 (2016-04-26 revision 54768) [x86_64-linux]
[root@10-9-62-65 home]#
```

安装redis(已经安装好了)(使用虚拟机操作开始位置)

上传redis包,必须是3.0以上版本

解压编译安装

#make make install

c语言编写内容需要编译后,进行编译安装,

```
-bash: redis-server: command not found
[root@10-9-62-65 redis-3.2.11]# make && make install
cd src && make all
make[1]: Entering directory `/home/redis-3.2.11/src'
```

```
make[1]: Entering directory `/home/redis-3.2.11/src'

Hint: It's a good idea to run 'make test' ;)

    INSTALL install
    INSTALL install
    INSTALL install
    INSTALL install
    INSTALL install
make[1]: Leaving directory `/home/redis-3.2.11/src'
[root@10-9-62-65 redis-3.2.11]#
```

rediscluster的具体测试

创建节点

创建集群节点,不同的节点创建对应的目录管理配置文件

例如: 占用8000端口的redis服务,对应redis.conf放到8000的文件夹;

这里我们以节点端口名称为名称,本次创建6个节点的集群目录

mkdir 8000 8001 8002 8003 8004 8005

```
[root@10-9-17-153 redis-3.2.11]# mkdir 8000 8001 8002 8003 8004 8005
[root@10-9-17-153 redis-3.2.11]#
```

ll

查看当前所在目录下的所有内容的详细信息,与ls类似

```
[root@10-9-17-153 redis-3.2.11]# ll
total 732
-rw-rw-r--  1 root root 92766 Sep 21  2017 00-RELEASENOTES
drwxr-xr-x  2 root root  4096 Jul 13 10:25 8000
drwxr-xr-x  2 root root  4096 Jul 13 10:25 8001
drwxr-xr-x  2 root root  4096 Jul 13 10:25 8002
drwxr-xr-x  2 root root  4096 Jul 13 10:25 8003
drwxr-xr-x  2 root root  4096 Jul 13 10:25 8004
drwxr-xr-x  2 root root  4096 Jul 13 10:25 8005
```

准备8000-8005启动的redis配置文件,各自的配置文件放到对应的文件夹下

配置文件修改并且上传到指定目录

P61 bind 127.0.0.1//默认ip为127.0.0.1改为其他节点机器可访问的ip

注释掉bind;可以监听连接当前服务的所有-h 后的ip;例如;绑定了127.0.0.1,但是登录时使用-h 10.9.17.153

P80 protected-mode no //yes修改为no

yes表示开启,保护模式下,连接redis服务需要用到密码;no表示不开启,redis-cli登录时,不需要提供安全信息,例如密码;

P84 port 8000 //端口8000-8005

P128 daemonize yes //后台运行

P150 pidfile /var/run/redis_8000.pid //pidfile文件对应7000

当大量的进程启动后,很难通过ps -ef|grep redis直接寻找你需要操作的进程的pid

P163 logfile 8000/redis.log //相对路径，启动时在redis的根目录

log文件记录的内容,就是在启动时,控制台打印的内容,和后续操作redis时所有的日志信息;

P224 dump dump8000.rdb //指定当前服务加载的持久化文件

P593 appendonly yes //默认是rdb方式持久化要改成AOF模式;

rdb持久化;和aof持久化模式对比

rdb:需要客户端必须调用命令 save完成数据从内存保存到磁盘的操作;没有来得及save的数据一旦丢失,数据出现未命中;

aof:二进制的日志,实时记录 redis客户端操作的所有写命令;没有来得及save的数据不保存在dump里,但是命令内容,保存在了aof文件;恢复时,只需要将每save的所有内容的命令调出来,

小量数据的性能对比: 开启aof模式,比rdb单独使用,多消耗2时间

海量数据:同时开启aof和rdb,单独开启rdb 消耗近似;

P721　cluster-enabled yes　//加载当前配置文件的redis服务一旦启动需要开启集群模式,否则无法使用集群的各种策略和逻辑

P729　cluster-config-file nodes_8000.conf　//集群配置启动后,将会创建这个nodes文件,记录当前服务器上唯一的一个集群状态;例如,搭建了8000-8005这样一个集群,操作失误下,集群失效,重新搭建集群,当前服务器读取这个配置文件会发现,槽道已经被一个集群分配了,新的集群搭建失败;

P735　cluster-node-timeout 15000　//请求超时 默认15秒，可自行设置

手动修改一个模板文件,后续5个可以利用fillzilla,或者其他的ftp工具抓下模板文件,window编辑器编辑;

这里为了方便使用上传文件的工具fileZilla

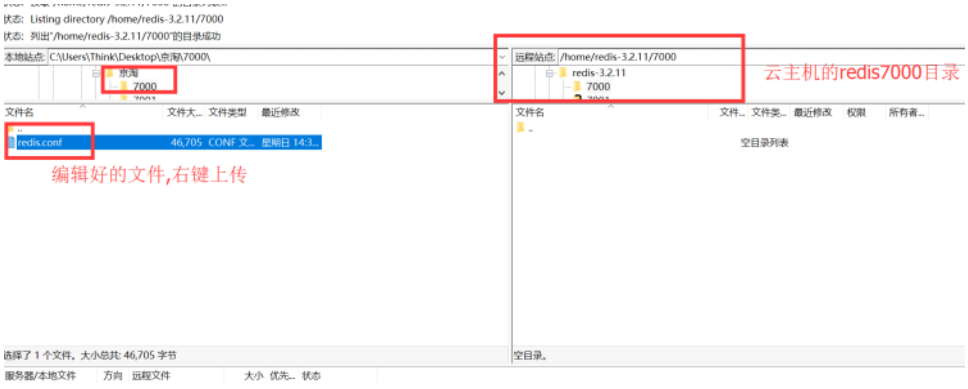
在windows中修改后上传到指定文件夹

将模板文件复制到8000的文件夹下

[redis-3.2.11]#cp redis.conf 8000

```
drwxrwxr-x  7 root root  4096 Sep 21  2017 utils
[root@10-9-17-153 redis-3.2.11]# vim redis.conf
[root@10-9-17-153 redis-3.2.11]# cp redis.conf 8000
```

fillzilla获取模板文件



这里如果不使用上传工具也可以调用vim命令直接修改文件

编辑完成第一个7000端口对应的节点文件

直接编辑7001 利用编辑软件的替换,将4个7000 端口换成7001,然后保存上传

完成之后调用ls检查目录中是否都存在redis.conf文件

```
[root@10-9-17-153 redis-3.2.11]# cp redis.conf 8000
[root@10-9-17-153 redis-3.2.11]# ls 800*
8000:
redis.conf

8001:
redis.conf

8002:
redis.conf

8003:
redis.conf

8004:
redis.conf

8005:
redis.conf
```

分别启动节点实例,一共启动集群配置的6个进程

```
drwxrwxr-x  7 root root  4096 Sep 21 22:20 utils
[root@10-9-62-65 redis-3.2.11]# ps -ef|grep redis
root     15347  1166  0 13:34 pts/0    00:00:00 grep redis
[root@10-9-62-65 redis-3.2.11]#
```

redis-server 8000/redis.conf

redis-server 8001/redis.conf

redis-server 8002/redis.conf

redis-server 8003/redis.conf

redis-server 8004/redis.conf

redis-server 8005/redis.conf

```
redis.conf
[root@10-9-17-153 redis-3.2.11]# redis-server 8000/redis.conf
[root@10-9-17-153 redis-3.2.11]# redis-server 8001/redis.conf
[root@10-9-17-153 redis-3.2.11]# redis-server 8003/redis.conf
[root@10-9-17-153 redis-3.2.11]# redis-server 8002/redis.conf
[root@10-9-17-153 redis-3.2.11]# redis-server 8004/redis.conf
[root@10-9-17-153 redis-3.2.11]# redis-server 8005/redis.conf
[root@10-9-17-153 redis-3.2.11]#
```

检查是否启动成功

#ps -ef|grep redis

搜索关键字redis有关的启动进程信息

```
[root@10-9-17-153 redis-3.2.11]# ps -ef|grep redis
root      8548     1  0 10:54 ?        00:00:00 redis-server *:8000
[cluster]
root      8554     1  0 10:54 ?        00:00:00 redis-server *:8001
[cluster]
root      8559     1  0 10:54 ?        00:00:00 redis-server *:8003
[cluster]
root      8566     1  0 10:54 ?        00:00:00 redis-server *:8002
[cluster]
root      8572     1  0 10:54 ?        00:00:00 redis-server *:8004
[cluster]
root      8578     1  0 10:54 ?        00:00:00 redis-server *:8005
[cluster]
root      8608  8013  0 10:55 pts/5    00:00:00 grep redis
```

#登录集群的客户端命令

#redis-cli -c -p 8000

-p 表示端口连接, -c 以集群状态登录节点(本质上是对单个节点的命令封装);但是这个时候所有的节点并不是集群状态,我们单独登录节点一样无法实现集群的操作

#redis-cli -p 8

```
root      8608  8013  0 10:55 pts/5    00:00:00 grep redis
[root@10-9-17-153 redis-3.2.11]# redis-cli -p 8000
127.0.0.1:8000> set name xiao
(error) CLUSTERDOWN Hash slot not served
127.0.0.1:8000> exit
[root@10-9-17-153 redis-3.2.11]#
```

创建集群

<查看>启动文件

在redis根目录src目录下有个文件redis-trib.rb,这个就是用ruby写的命令文件,它是一个语言编写的快速操作集群的命令文件,整合了底层客户端的各种各样的命令,

执行创建集群命令(redis集群最少需要3个master才能正常运行)

./redis-trib.rb 如果我使用的是./说明当前的/etc/profile中没有对PATH进行 ./的配置,所有不在环境变量中的命令文件需要到对应目录调用./启动

```
  create          host1:port1 ... hostN:portN
                  --replicas <arg>
```

./redis-trib.rb create 10.9.17.153:8000 10.9.17.153:8001 10.9.17.153:8002

这里的8003 8004 8005暂不添加,后续添加节点使用

解释下， --replicas 1 表示 自动为每一个master节点分配一个slave节点 上面有6个节点，程序会按照一定规则生成 3个master（主）3个slave(从)

```
[root@10-9-17-153 src]# ./redis-trib.rb create 10.9.17.153:8000 10.9.17.153:800
1 10.9.17.153:8002
>>> Creating cluster
>>> Performing hash slots allocation on 3 nodes...
Using 3 masters:
10.9.17.153:8000
10.9.17.153:8001
10.9.17.153:8002
M: 81044f938fb7889096e848c5758b84e5b60e03d5 10.9.17.153:8000
   slots:0-5460 (5461 slots) master
M: 5c90a71b4ace2ed64c28e686072cd40c535980fe 10.9.17.153:8001
   slots:5461-10922 (5462 slots) master
M: 1f24289bf6383d941fad86e6ec8ce1a4ca5728c5 10.9.17.153:8002
   slots:10923-16383 (5461 slots) master
Can I set the above configuration? (type 'yes' to accept):
```

ISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

提问是否同意以上内容的配置结果,yes表示同意

```
M: 1f24289bf6383d941fad86e6ec8ce1a4ca5728c5 10.9.17.15
   slots:10923-16383 (5461 slots) master
   0 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@10-9-17-153 src]#
```

表示集群搭建成功,槽道分配完毕;

后期的连接需要端口开发,防火墙直接关闭;

#service iptables stop //临时关闭防火墙,重启之后防火墙依然开启

#chkconfig iptables off //永久关闭防火墙配置,两个命令配合,

登陆

#redis-cli -c -p 8000

这里注意，如果使用原有的命令没有-c则表示单独登陆节点，没有集群效果

查看集群状态

8000>cluster info

```
127.0.0.1:8000> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:3
cluster_size:3
cluster_current_epoch:3
cluster_my_epoch:1
cluster_stats_messages_sent:412
cluster_stats_messages_received:412
127.0.0.1:8000>
```

都比较好理解，唯独epoch，epoch是个逻辑计算时间，与节点的所有变化都有关,与集群事件有关;

cluster_current_epoch:6 代表当前集群的最新逻辑计算时间，数字越大，表示操作或者配置越新，整个集群的这个值是保持一致的

cluster_my_epoch:1代表当前节点的逻辑计算时间，

查看集群节点

8000>cluster nodes

查看集群所有节点的信息

```
127.0.0.1:8000> cluster nodes
5c90a71b4ace2ed64c28e686072cd40c535980fe 10.9.17.153:8001 master - 0 1531452575
095 2 connected 5461-10922
1f24289bf6383d941fad86e6ec8ce1a4ca5728c5 10.9.17.153:8002 master - 0 1531452576
098 3 connected 10923-16383
81044f938fb7889096e848c5758b84e5b60e03d5 10.9.17.153:8000 myself,master - 0 0 1
 connected 0-5460
127.0.0.1:8000>
```

这里的内容比较杂乱

节点 ID ：例如 3fc783611028b1707fd65345e763befb36454d73 。

ip:port ：节点的 IP 地址和端口号， 例如 127.0.0.1:7000 ，

flags ：节点的角色（例如 master 、 slave 、 myself 如果标识myself说明当前客户端登录的端口）以及状态（例如 fail ，等等）。

如果节点是一个从节点的话， 那么跟在 flags 之后的将是主节点的节点 ID ： 例如 127.0.0.1:7002 的主节点的节点 ID 就是 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e 。

master节点最后有一个值的范围，就是hash槽，0~16383（2^14），平均的分配到各master节点。

测试set和get数据体会槽道的原理

登录8000 set name hanlaoshi

集群底层计算判断逻辑:name的哈希取模,5798, 是否归8000管理;

转向了8001,存储

```
[root@10-9-17-153 src]# redis-cli -c -p 8002
127.0.0.1:8002> get name
-> Redirected to slot [5798] located at 10.9.17.153:8001
"hanlaoshi"
10.9.17.153:8001> get name
"hanlaoshi"
10.9.17.153:8001> quit
[root@10-9-17-153 src]# redis-cli -c -p 8000
127.0.0.1:8000> get name
-> Redirected to slot [5798] located at 10.9.17.153:8001
"hanlaoshi"
10.9.17.153:8001>
```

动态添加节点

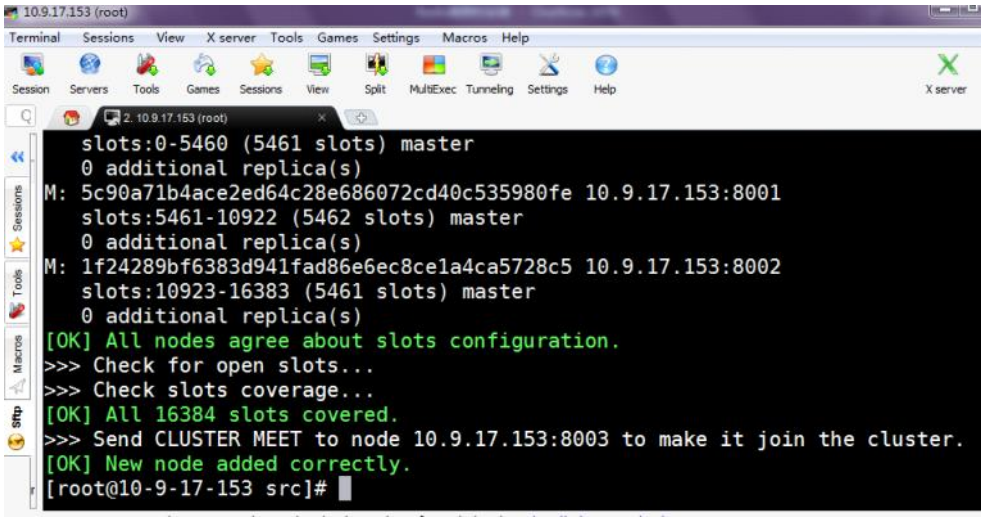
启动新节点8004 8003 8005

```
                --threshold <arg>
add-node        new_host:new_port existing_host:existing_port
                --slave
                --master-id <arg>
 del-node       host:port node_id
```

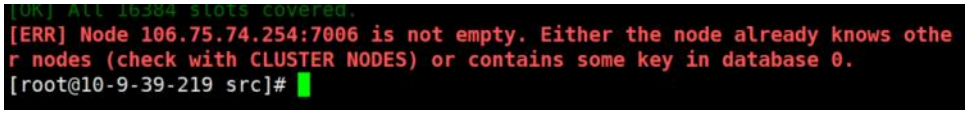
执行添加节点命令(将8003添加到当前集群中)

./redis-trib.rb add-node 10.9.17.153:8003 10.9.17.153:8001

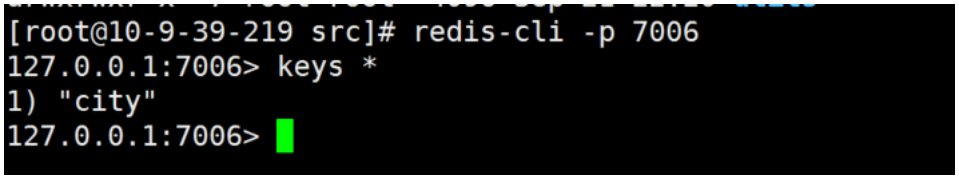
前面是新节点,后面是存在的任意一个节点



如果出现如下问题



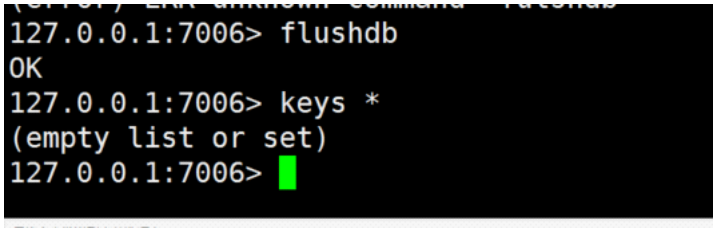
查看当前节点的数据是否有key存在

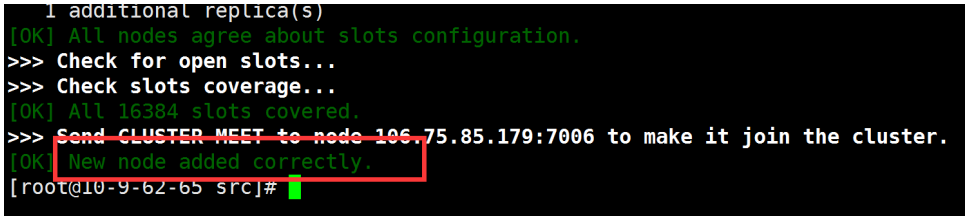


注意:每一个集群中的节点单独维护一个dump文件;以上错误原因,多个节点共享同一个dump文件导致的新节点启动加载就数据;

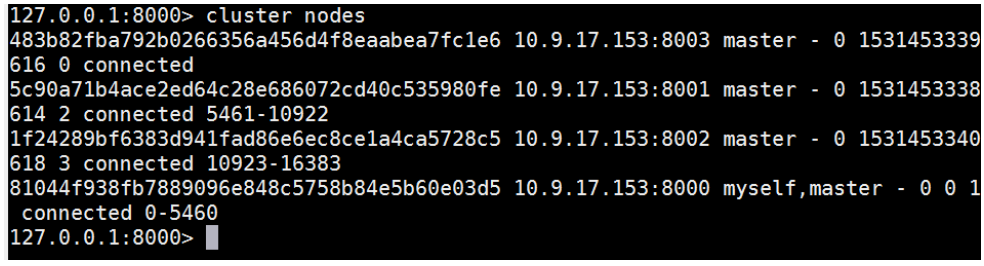
根据提示信息7006中有存留的数据,为了防止数据冲突不允许带有数据的节点动态添加到当前集群

将节点中的数据清空





进入集群查看节点状态(默认情况是master,不管理槽道)



动态添加从节点

```
add-node        new_host:new_port existing_host:existing_port
                --slave
                --master-id <arg>
```

--slave 和--master-id 必须同时配置就可以指定给一个master添加从节点,新增节点就是从节点的角色

添加从节点(8004添加到主节点8001)

上传编辑好的7007文件夹和内部的redis.conf文件

执行添加节点成为某一台主节点的从节点命令

--slave 指定当前节点以从节点角色添加到集群中

--master-id 后面跟随主节点的masterid值,cluster nodes可以查看

[root@10-9-17-153 src]# ./redis-trib.rb add-node --slave --master-id 5c90a71b4ace2ed64c28e686072cd40c535980fe 10.9.17.153:8004 10.9.17.153:8000

--master-id后面必须跟着主节点的id(可以使用cluster nodes查看)

```
[root@10-9-17-153 src]# ./redis-trib.rb add-node --slave --master-id 5c90a71b4a
ce2ed64c28e686072cd40c535980fe 10.9.17.153:8004 10.9.17.153:8000
>>> Adding node 10.9.17.153:8004 to cluster 10.9.17.153:8000
>>> Performing Cluster Check (using node 10.9.17.153:8000)
M: 81044f938fb7889096e848c5758b84e5b60e03d5 10.9.17.153:8000
   slots:0-5460 (5461 slots) master
   0 additional replica(s)
M: 483b82fba792b0266356a456d4f8eaabea7fc1e6 10.9.17.153:8003
   slots: (0 slots) master
   0 additional replica(s)
M: 5c90a71b4ace2ed64c28e686072cd40c535980fe 10.9.17.153:8001
   slots:5461-10922 (5462 slots) master
   0 additional replica(s)
M: 1f24289bf6383d941fad86e6ec8ce1a4ca5728c5 10.9.17.153:8002
   slots:10923-16383 (5461 slots) master
   0 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

检查添加是否成功

8000>cluster nodes 观察结果

新的master要起作用需要保存缓存数据

数据与槽道挂钩,所以需要迁移槽道来

为新的节点分配槽道

./redis-trib.rb reshard 10.9.17.153:8004

表示将集群中的槽道重新分配,后面的节点信息,已经在集群中存在的任意节点信息,

```
[root@10-9-17-153 redis-3.2.11]# cd src
[root@10-9-17-153 src]# ./redis-trib.rb reshard 10.9.17.153:8000
```

根据提示操作

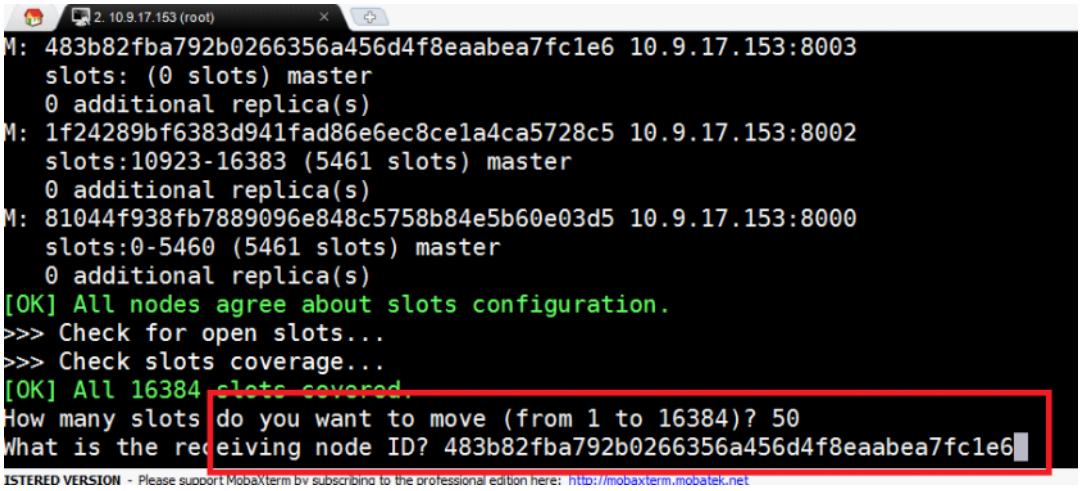
1 移动多少槽道

How many slots do you want to move (from 1 to 16384)? 50

将集群中若干个槽道数量从新分配

2 接收移动槽道的节点id是什么

What is the receiving node ID? 指定一个接收分配槽道的节点id8003id



3 源数据节点都有哪些

Please enter all the source node IDs.

Type 'all' to use all the nodes as source nodes for the hash slots.

Type 'done' once you entered all the source nodes IDs.

如果从固定几个节点获取数据,直接粘贴对应id,必须是主节点

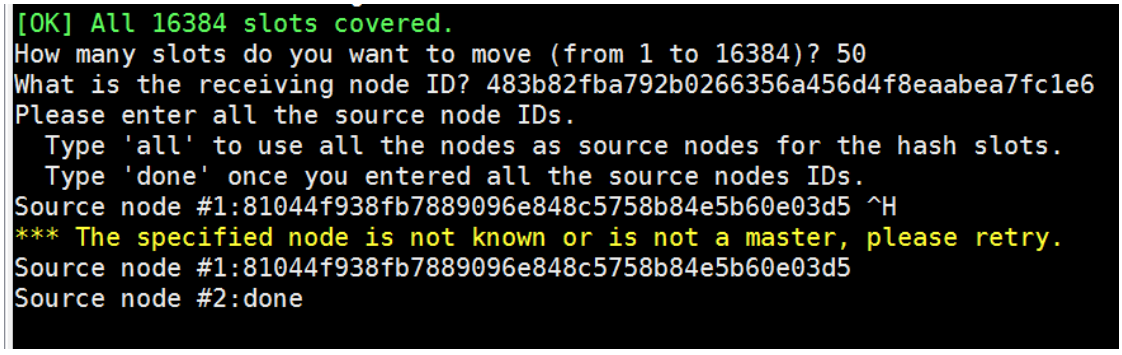
如果从所有id平均分配槽道,all

输入all或者done时,分配计算结束;

Source node #1:all

这里我们就用all

如果使用all将会自动计算从所有主节点平均获取槽道移动



4 确定输入yes

Do you want to proceed with the proposed reshard plan (yes/no)?

```
    Moving slot 11072 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11073 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11074 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11075 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11076 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11077 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11078 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11079 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11080 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11081 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11082 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11083 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11084 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11085 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11086 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11087 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
    Moving slot 11088 from 2dfc3052db7e91cfa0af30aa0264a0763546b908
Do you want to proceed with the proposed reshard plan (yes/no)? yes
```

5 屏幕滚动移动过程

6 查看cluster nodes 发现8003 已经配有的从各个源数据节点上移动过来的槽道

```
966 3 connected 10923-16383
483b82fba792b0266356a456d4f8eaabea7fc1e6 10.9.17.153:8003 myself,master -
 connected 0-49
5c90a71b4ace2ed64c28e686072cd40c535980fe 10.9.17.153:8001 master - 0 15314
962 2 connected 5461-10922
81044f938fb7889096e848c5758b84e5b60e03d5 10.9.17.153:8000 master - 0 15314
963 1 connected 50-5460
```

重新分配槽道,不能重新操作有数据的槽道,空槽道可以reshard,有数据的需要手动迁移数据后在迁移槽道(在补充操作中练习)

高可用验证

停掉一个主的进程

kill掉主节点8001

#ps -ef|grep redis

```
[root@10-9-17-153 src]# ps -ef|grep redis
root      8548     1  0 10:54 ?        00:00:10 redis-server *:8000 [cluster]
root      8554     1  0 10:54 ?        00:00:11 redis-server *:8001 [cluster]
root      8559     1  0 10:54 ?        00:00:09 redis-server *:8003 [cluster]
root      8566     1  0 10:54 ?        00:00:10 redis-server *:8002 [cluster]
root      8572     1  0 10:54 ?        00:00:10 redis-server *:8004 [cluster]
root      8578     1  0 10:54 ?        00:00:08 redis-server *:8005 [cluster]
root      9318  8013  0 14:02 pts/5    00:00:00 grep redis
[root@10-9-17-153 src]#
```

#kill 8554(执行登录到8001节点 shutdown的命令也可以)

然后登陆集群查看集群节点

8000>cluster nodes

```
root      9318  8013  0 14:02 pts/5    00:00:00 grep redis
[root@10-9-17-153 src]# kill 8554
[root@10-9-17-153 src]# redis-cli -c -p 8000
127.0.0.1:8000> cluster nodes
3b73deb56256bcecea7f96f1965c08d95bd4dd4e 10.9.17.153:8004 slave 5c90a71b4ace2e
64c28e686072cd40c535980fe 0 1531461857997 2 connected
5c90a71b4ace2ed64c28e686072cd40c535980fe 10.9.17.153:8001 master - 15314618509
9 1531461843964 2 disconnected 5461-10922
483b82fba792b0266356a456d4f8eaabea7fc1e6 10.9.17.153:8003 master - 0 153146185
999 5 connected 0-49
81044f938fb7889096e848c5758b84e5b60e03d5 10.9.17.153:8000 myself,master - 0 0
 connected 50-5460
1f24289bf6383d941fad86e6ec8ce1a4ca5728c5 10.9.17.153:8002 master - 0 153146186
006 3 connected 10923-16383
```

正在选举

启动停掉的节点会自动拼接成slave加入

等待一定时间,

8000>cluster nodes

发现8001的从节点8004重新选举成新的主节点

```
127.0.0.1:8000> cluster nodes
3b73deb56256bcecea7f96f1965c08d95bd4dd4e 10.9.17.153:8004 master - 0 1531461887
069 6 connected 5461-10922
5c90a71b4ace2ed64c28e686072cd40c535980fe 10.9.17.153:8001 master,fail - 1531461
850979 1531461843964 2 disconnected
483b82fba792b0266356a456d4f8eaabea7fc1e6 10.9.17.153:8003 master - 0 1531461886
068 5 connected 0-49
81044f938fb7889096e848c5758b84e5b60e03d5 10.9.17.153:8000 myself,master - 0 0 1
 connected 50-5460
1f24289bf6383d941fad86e6ec8ce1a4ca5728c5 10.9.17.153:8002 master - 0 1531461884
062 3 connected 10923-16383
```

删除节点

```
                --master-id <arg>
  del-node      host:port node_id
  set-timeout   host:port milliseconds
```

主节点不能直接删除,需要reshard将槽道全部移出

./redis-trib.rb del-node 10.9.17.153:8000

09a25f98f963e3f90193440be5850bd351eb228b

参数意义: 节点信息是已经存在与集群的节点,id表示删除的节点id

删除从节点(无数据直接删除)

将8001重启之后,称为从节点加入集群,可以测试删除

```
[root@10-9-17-153 src]# cd ..
[root@10-9-17-153 redis-3.2.11]# redis-server 8001/redis.conf
[root@10-9-17-153 redis-3.2.11]#
```

[root@10-9-17-153 src]# ./redis-trib.rb del-node 10.9.17.153:8000 5c90a71b4ace2ed64c28e686072cd40c535980fe

```
[root@10-9-17-153 redis-3.2.11]# cd src
[root@10-9-17-153 src]# ./redis-trib.rb del-node 10.9.17.153:8000 5c90a71b4ace
ed64c28e686072cd40c535980fe
>>> Removing node 5c90a71b4ace2ed64c28e686072cd40c535980fe from cluster 10.9.1
53:8000
>>> Sending CLUSTER FORGET messages to the cluster...
>>> SHUTDOWN the node.
[root@10-9-17-153 src]# ps -ef|grep redis
root      8548     1  0 10:54 ?        00:00:11 redis-server *:8000 [cluster]
root      8559     1  0 10:54 ?        00:00:10 redis-server *:8003 [cluster]
root      8566     1  0 10:54 ?        00:00:10 redis-server *:8002 [cluster]
root      8572     1  0 10:54 ?        00:00:10 redis-server *:8004 [cluster]
root      8578     1  0 10:54 ?        00:00:08 redis-server *:8005 [cluster]
root      9348  8013  0 14:12 pts/5    00:00:00 grep redis
[root@10-9-17-153 src]#
```