# redis单机安装

## 安装redis需要先安装gcc环境

|  |
| --- |
| yum install gcc |

## 安装redis

|  |
| --- |
| tar -zxvf redis-3.2.9.tar.gz  cd redis-3.2.9.tar.gz  make  cd src && make install |

如下图所示时代表安装成功



### ****创建目录存放redis命令和配置文件****

|  |
| --- |
| mkdir -p /usr/local/redis/bin  mkdir -p /usr/local/redis/etc |

### ****移动文件****

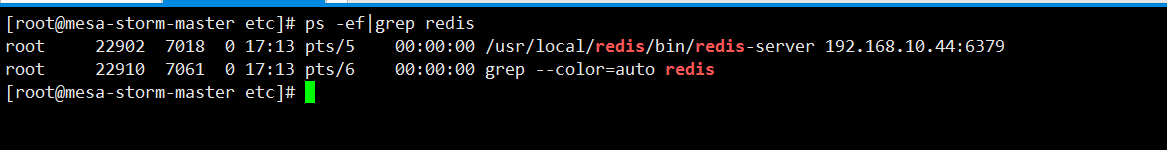
|  |
| --- |
| cd /home/redis/redis-3.2.9  mv redis.conf /usr/local/redis/etc  cd src  mv mkreleasehdr.sh redis-benchmark redis-check-aof redis-check-rdb redis-cli redis-server redis-sentinel redis-trib.rb /usr/local/redis/bin |

## ****启动redis服务****

|  |
| --- |
| /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf |

如上，启动redis服务需要指定配置文件的，后台启动的话需要修改redis.conf文件，daemonize no ---- >daemonize yes。redis服务端默认链接端口是6379，最好也将IP绑定为本机IP。

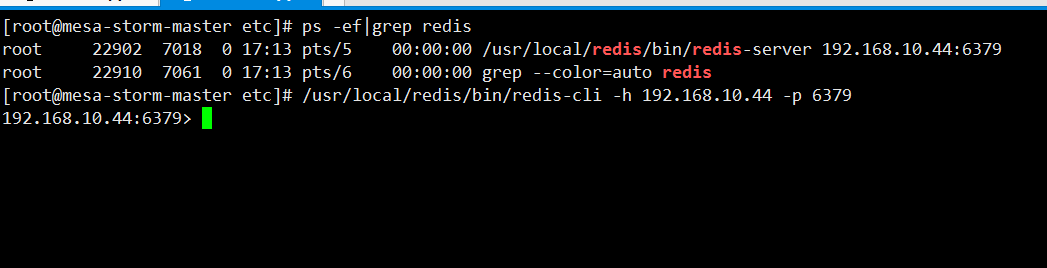
## ****验证是否启动成功****



## ****客户端连接****

/usr/local/redis/bin/redis-cli -h 192.168.2.128 -p 6379

注意这里使用的ip需要与redis.conf里面的bind属性的值一致,默认是127.0.0.1



## ****停止redis服务****

关闭指定主机的redis服务

|  |
| --- |
| /usr/local/redis/bin/redis-cli -h 192.168.10.44 shutdown |

或者登陆进客户端输入shutdown

或者使用pkill redis-server

## redis安全设置

### 设置客户端连接密码

redis安装好后，默认情况下登陆客户端和使用命令操作时不需要密码的。某些情况下，为了安全起见，我们可以设置在客户端连接后进行任何操作之前都要进行密码验证。修改redis.conf进行配置。

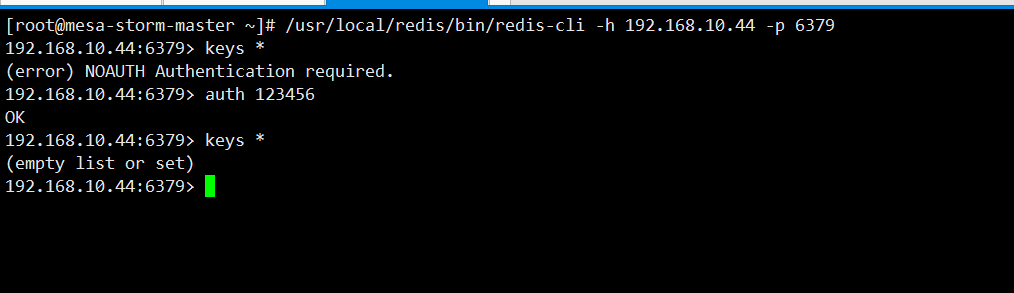
|  |
| --- |
| # Warning: since Redis is pretty fast an outside user can try up to  # 150k passwords per second against a good box. This means that you should  # use a very strong password otherwise it will be very easy to break.  #  # requirepass foobared  requirepass 123456 |

编辑requirepass属性,密码是123456

如上，找到# requirepass foobared这一行，在下面添加“requirepass 密码”一行设置密码。设置好密码后，有两种方式授权客户端进行操作。

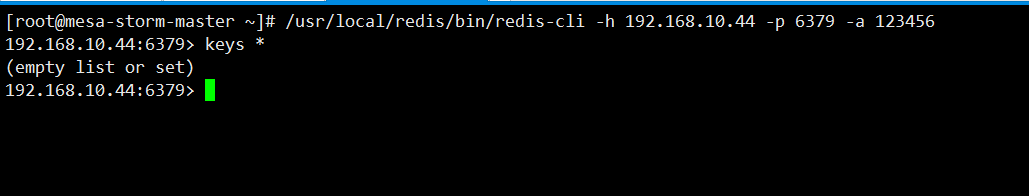
### 客户端登陆方式1, 登录客户端后使用auth命令进行授权，如下

先使用/usr/local/redis/bin/redis-cli -h 192.168.10.44 -p 6379命令登陆后再使用auth 密码方式进行认证



### 客户端登陆方式2, 登录时使用-a参数指定客户端密码，如下

直接使用/usr/local/redis/bin/redis-cli -h 192.168.10.44 -p 6379 -a 123456



## ****主从复制****

主从复制，即主服务器与从服务器之间数据备份的问题。Redis 支持简单且易用的主从复制（master-slave replication）功能， 该功能可以让从服务器(slave server)成为主服务器(master server)的精确复制品。

### ****主从复制的特点****

（1）一个主服务器可以有多个从服务器。

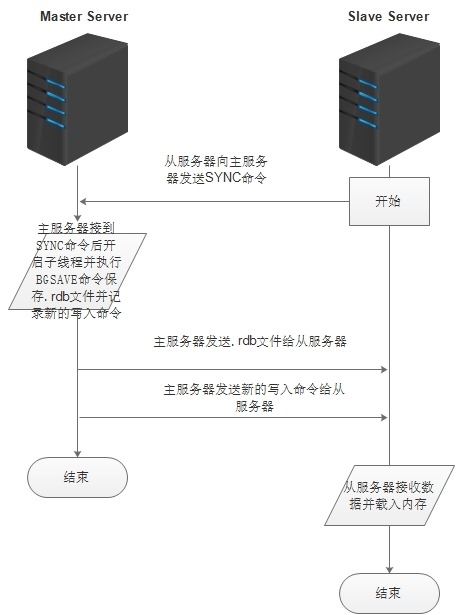
（2）不仅主服务器可以有从服务器， 从服务器也可以有自己的从服务器。

（3）Redis 支持异步复制和部分复制（这两个特性从Redis 2.8开始），主从复制过程不会阻塞主服务器和从服务器。

（4）主从复制功能可以提升系统的伸缩性和功能，如让多个从服务器处理只读命令，使用复制功能来让主服务器免于频繁的执行持久化操作。

### ****主从复制的过程****

下面我们用一个图来讲解redis主从复制的过程。

[](http://images2015.cnblogs.com/blog/783994/201607/783994-20160711160410326-1705321436.jpg)

Redis主从复制过程示意图

从上面的示意图可以看出，主服务器与从服务器建立连接之后，Redis主从复制过程主要有下面几步：

（1）从服务器都将向主服务器发送一个 SYNC 命令。

（2）主服务器接到 SYNC 命令后开启一个后台子进程并开始执行 BGSAVE，并在保存操作执行期间， 将所有新执行的写入命令都保存到一个缓冲区里面。

（3）当 BGSAVE 执行完毕后， 主服务器将执行保存操作所得的 .rdb 文件发送给从服务器， 从服务器接收这个 .rdb 文件， 并将文件中的数据载入到内存中。

（4）主服务器会以 Redis 命令协议的格式， 将写命令缓冲区中积累的所有内容都发送给从服务器。

### ****配置从服务器****

复制redis.conf为redis-repl.conf

redis配置一个从服务器非常简单， 只要在从服务器的配置文件redis-repl.conf中增加主服务器的IP地址和端口号就可以，如果主服务器设置了客户端密码，还需要在从服务器中配置主服务器的密码，如下

|  |
| --- |
| slaveof 192.168.10.44 6379  masterauth 123456 |

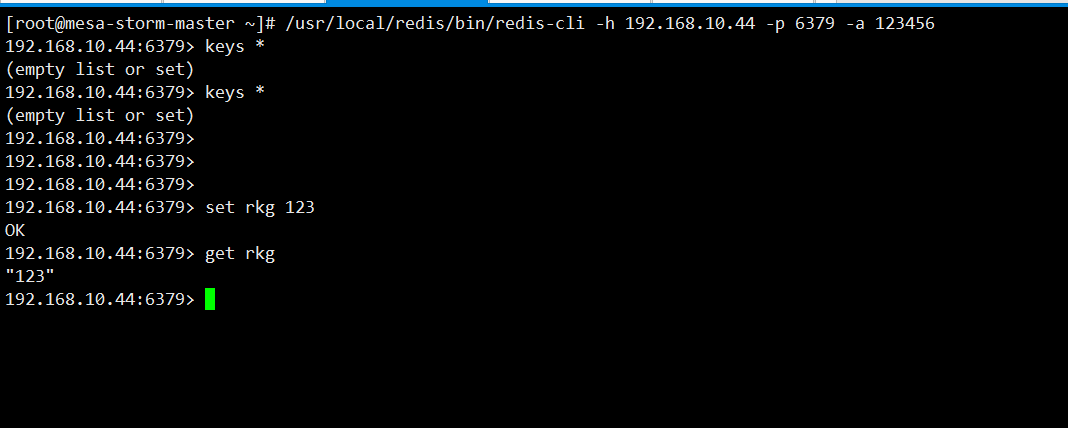
### 验证从服务器是否启动成功

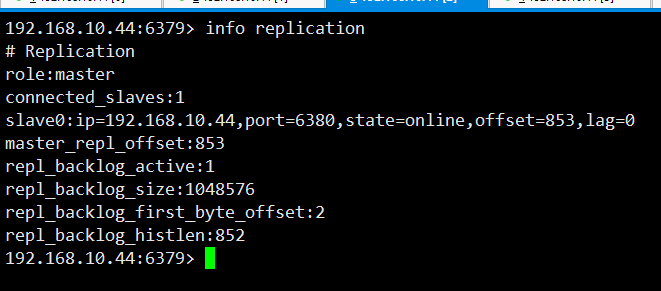
分别启动主从redis服务

/usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf

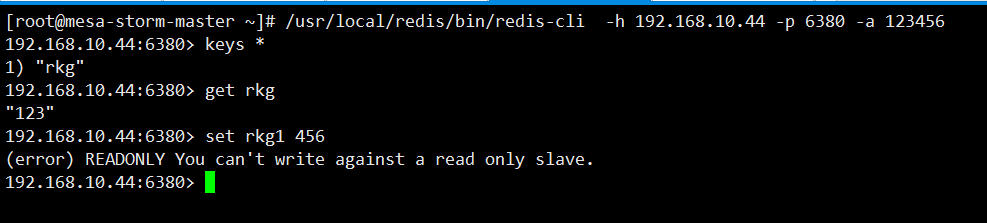
/usr/local/redis/bin/redis-server /usr/local/redis/etc/redis-repl.conf

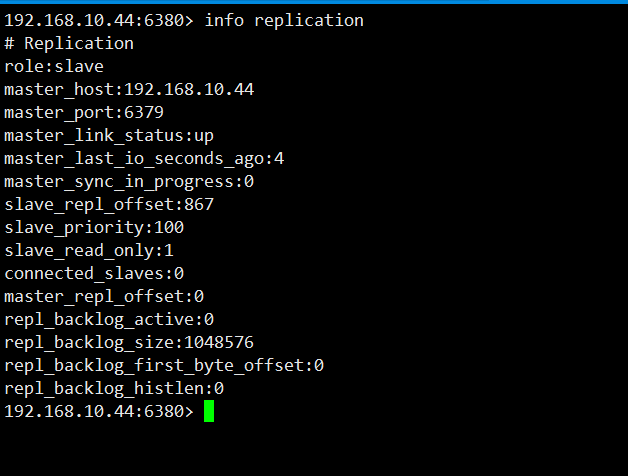
启动主客户端,并且设置值





启动从客户端,发现可以获取上面设置的值,但是在从服务器设置key和value得时候提示只读,不能设置





主从复制的步骤：

1、从服务器开始链接主服务器时，会向主服务器发送一个 SYNC 命令   
2、主服务接收到命令之后，执行 BGSAVE，异步的将写命令保存到一个缓冲区里面   
3、主服务器执行完BGSAVE之后，就.rdb 文件(BOF的持久化文件)发送给从服务器，从服务器从该文件恢复数据到内存中   
4、主服务器还会以 Redis 命令协议的格式， 将写命令缓冲区中积累的所有内容都发送给从服务器，从服务器通过执行命令将数据恢复到内存中

## ****事务与锁****

Redis 的事务支持相对简单，MULTI 、 EXEC 、 DISCARD 和 WATCH 这四个命令是 Redis 事务的基础。

### ****事务开启与取消****

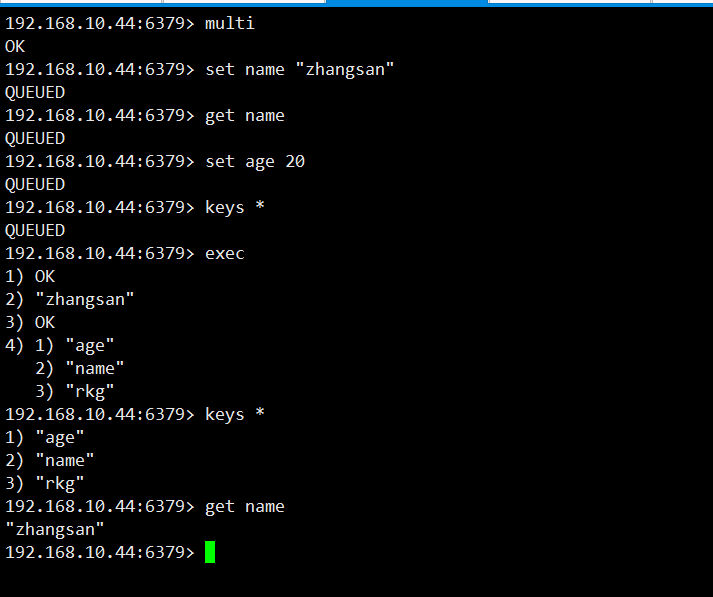
MULTI 开启一个事务。当客户端发出了MULTI 命令时，客户端和服务端的连接就进入了一个事务上下文的状态。MULTI 执行之后， 客户端可以继续向服务器发送任意多条命令， 这些命令不会立即被执行， 而是被放到一个队列中， 当 EXEC 命令被调用时， 所有队列中的命令才会被执行。

EXEC 顺序执行事务队列中的命令。

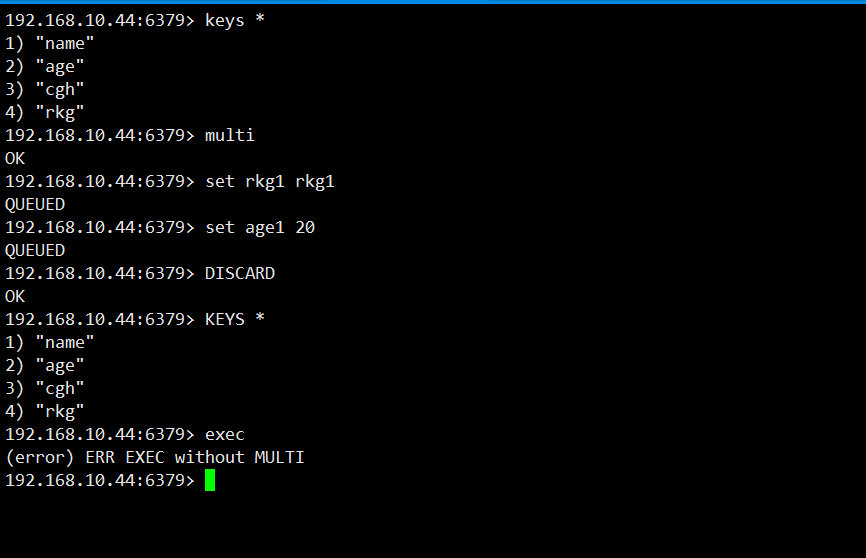
multi:开启事务

之后再执行命令时都被加入了queued队列中,当执行exec时会自动执行队列中的命令

exec提交事务



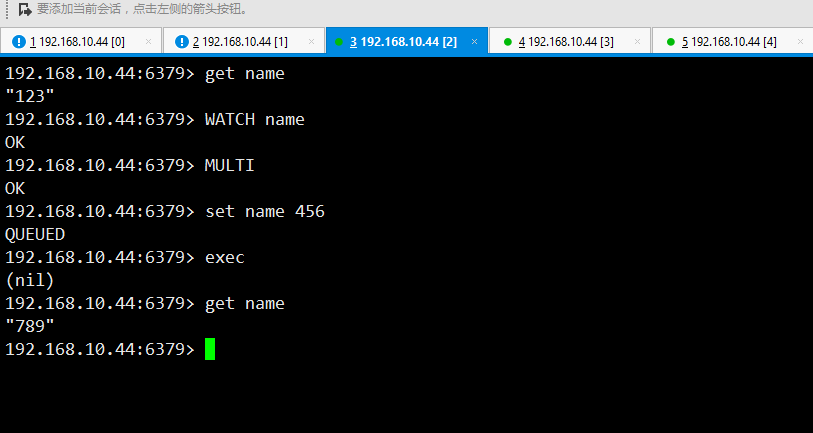
discard取消事务

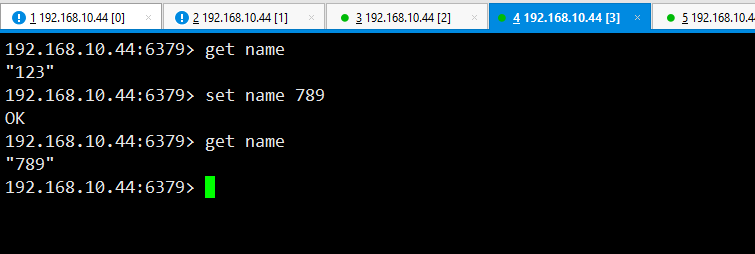


### 乐观锁

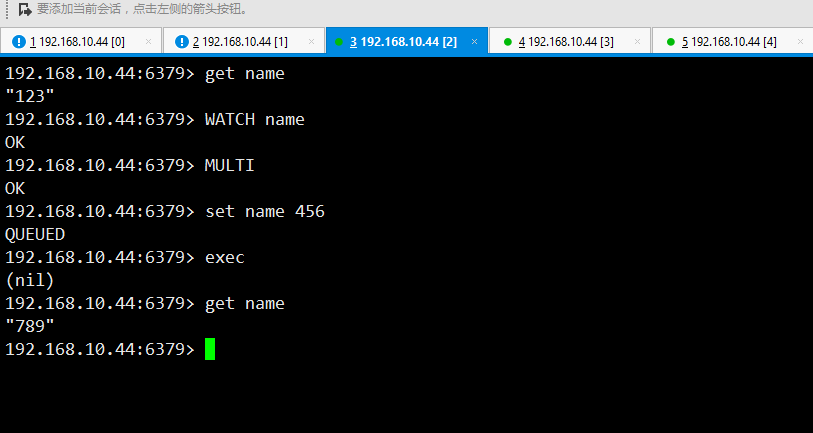
WATCH 对key值进行锁操作。 在 WATCH 执行之后， EXEC 执行之前， 有其他客户端修改了 key 的值， 那么当前客户端的事务就会失败。如下：

Client1开启watch name并在事务中修改name，但是没有执行exec



Client2 修改name为789,此时在client1中执行exec提示nil,再次获取name的值时发现值是789不是456

Client1执行exec



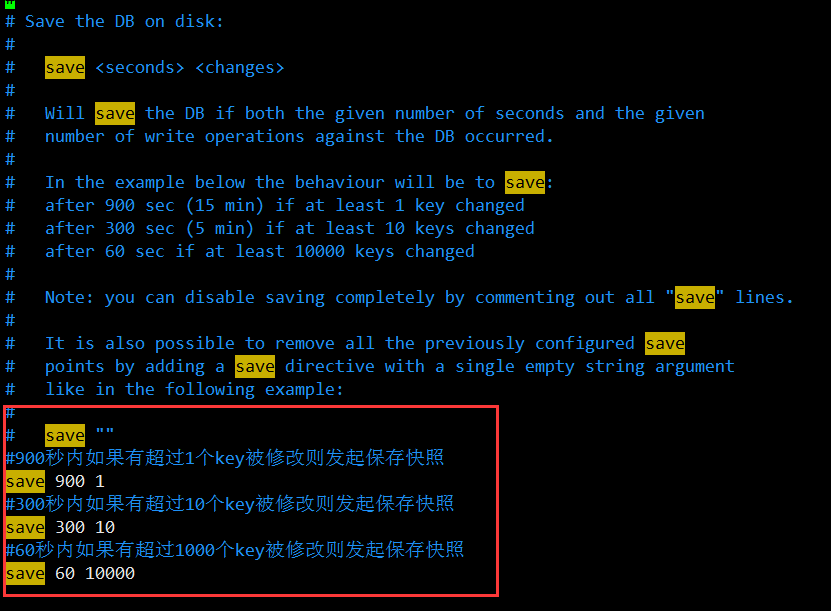
可见，由于被watch的name已经被Client2 修改，所以Client1的事务执行失败，程序需要做的， 就是不断重试这个操作， 直到没有发生碰撞（Crash）为止。对key进行加锁监视的机制类似Java多线程中的锁（synchronized中的监视器对象），被称作乐观锁。乐观是一种非常强大的锁机制。

## ****持久化机制****

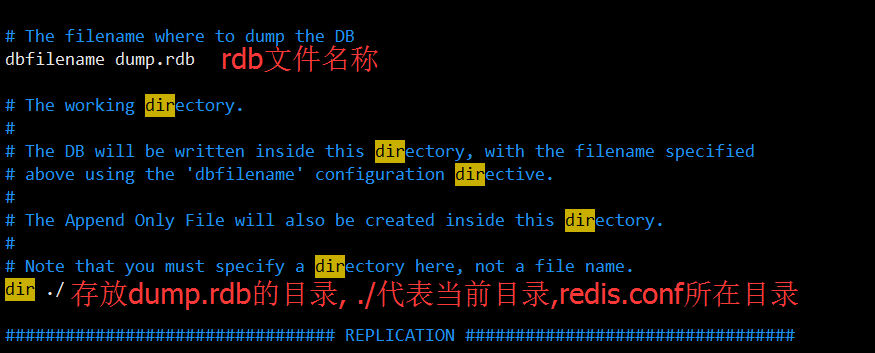
前面我们已经说过，既可以把redis理解为缓存技术，也可以理解为数据库，因为redis支持将内存中的数据周期性的写入磁盘或者把操作追加到记录文件中，这个过程称为redis的持久化。redis支持两种方式的持久化，一种是快照方式（snapshotting）,也称RDB方式；两一种是追加文件方式（append-only file），也称AOF方式。RDB方式是redis默认的持久化方式。

### ****RDB方式****

RDB方式是将内存中的数据的快照以二进制的方式写入名字为 dump.rdb的文件中。我们对 Redis 进行设置， 让它根据设置周期性自动保存数据集。修改redis.conf文件，如下



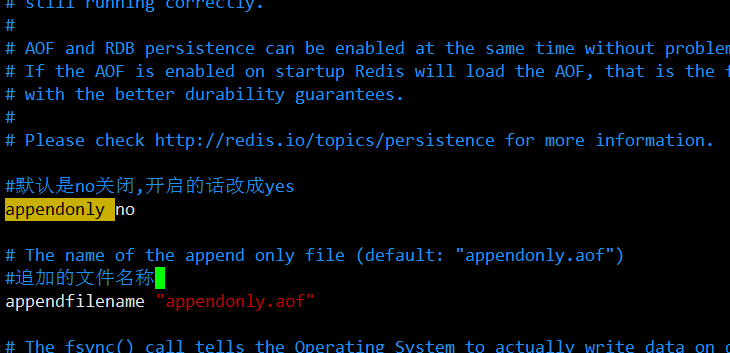
dump.rdb文件默认生成在%REDIS\_HOME%etc目录下（如/usr/local/redis/etc/），可以修改redis.conf文件中的dir指定dump.rdb的保存路径

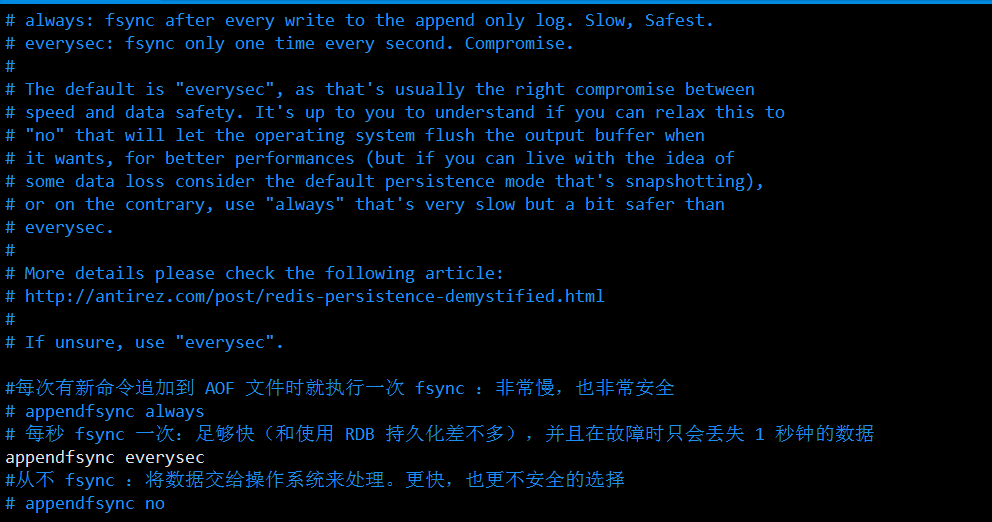


### ****AOF方式****

RDB方式是周期性的持久化数据， 如果未到持久化时间点，Redis 因为某些原因而造成故障停机， 那么服务器将丢失最近写入、且仍未保存到快照中的那些数据。所以从redis 1.1开始引入了AOF方式，AOF 持久化记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。 AOF 文件中的命令全部以 Redis 协议的格式来保存，新命令会被追加到文件的末尾。

AOF方式仍然有丢失数据的可能，因为收到写命令后可能并不会马上将写命令写入磁盘，因此我们可以修改redis.conf，配置redis调用write函数写入命令到文件中的时机。如下





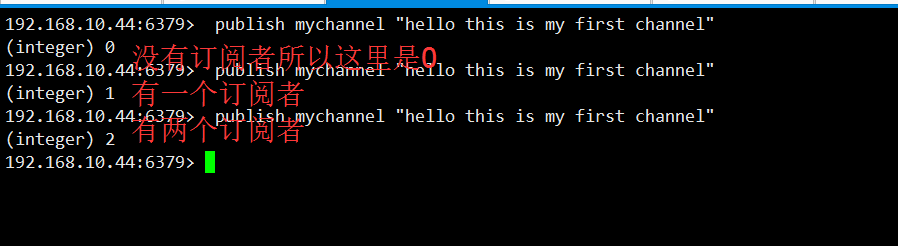
从上面三种AOF持久化时机来看，为了保证不丢失数据，appendfsync always是最安全的。

## ****发布以及订阅消息****

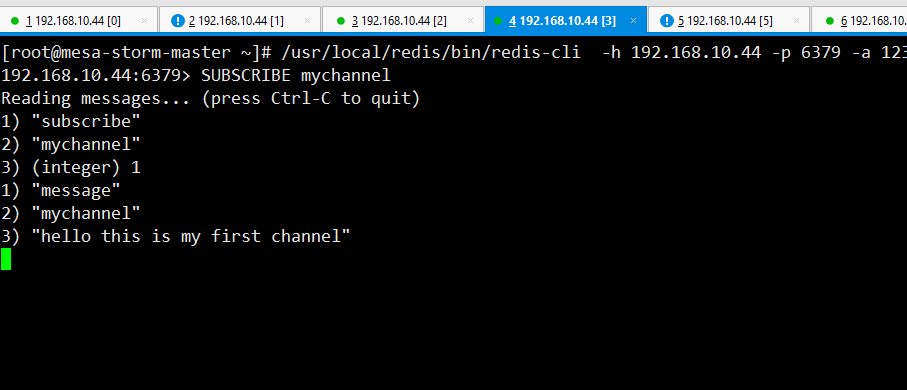
Redis的发布以及订阅有点类似于聊天，是一种消息通信模式。在这个模式中，发送者（发送信息的客户端）不是将信息直接发送给特定的接收者（接收信息的客户端）， 而是将信息发送给频道（channel）， 然后由频道将信息转发给所有对这个频道感兴趣的订阅者。SUBSCRIBE 、 UNSUBSCRIBE 和 PUBLISH 三个命令实现了消息的发布与订阅。如下

Client1发布频道mychannel与消息

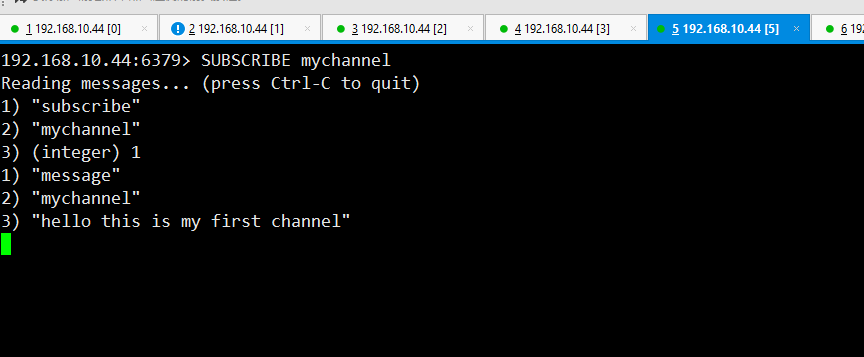
发布者



订阅者1



订阅者2



# Redis-Sentinel(redis哨兵)

g

# redis-cluster(redis集群)

**Redis集群简介**

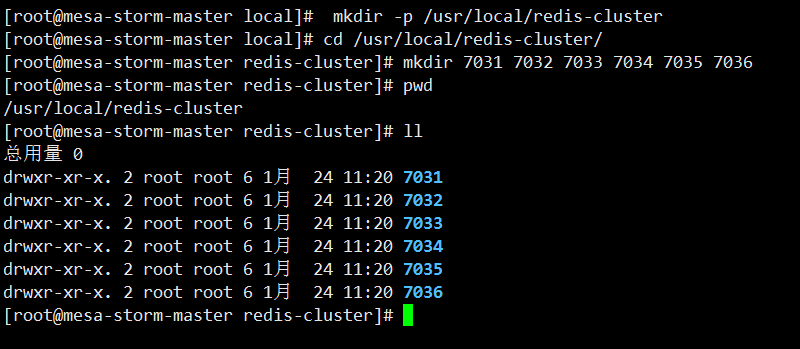
Redis 集群是3.0之后才引入的，在3.0之前，使用哨兵（sentinel）机制来监控各个节点之间的状态。Redis 集群是一组能进行数据共享的Redis 实例（服务或者节点）的设施，集群可以使用的功能是普通单机 Redis 所能使用的功能的一个子集；Redis 集群通常具有高可用、可扩展性、分布式、容错等特性。

## ****Redis集群搭建****

### ****创建集群文件夹****

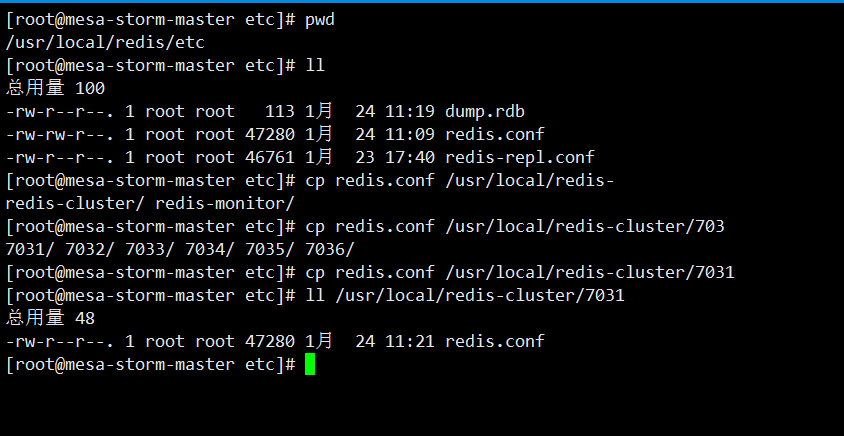
在/usr/local/下新建redis-cluster目录并在redis-cluster下新建7031~7036共6个文件夹，这6个文件夹代表创建redis集群的6个节点。如下

|  |
| --- |
| mkdir -p /usr/local/redis-cluster  cd /usr/local/redis-cluster/  mkdir 7031 7032 7033 7034 7035 7036 |



### ****拷贝修改配置文件****

将已有的/usr/local/redis/etc/下的redis.conf拷贝到新创建的7031目录中



修改项如下：

（1）绑定端口，port 7031

（2）绑定IP，bind 192.168.10.44

（3）指定数据存放路径，dir /usr/local/redis-cluster/7031

（4）启动集群模式，cluster-enabled yes

（5）指定集群节点配置文件，cluster-config-file nodes-7031.conf

（6）后台启动，daemonize yes

（7）指定集群节点超时时间，cluster-node-timeout 5000

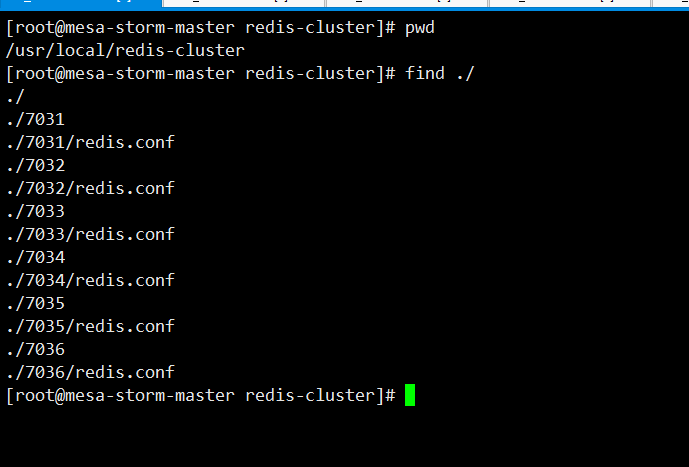
（8）指定持久化方式，appendonly yes

上面红色项目最好全部设置，不然会出意想不到的错误，703x最好与节点文件夹保持一致。

将7031的redis.conf改完后再拷贝到剩下的5个目录中，然后只要全局替换redis.conf中的7031为相应的节点即可。

(9)指定logfile,服务端日志存储位置

logfile /usr/local/redis-cluster/7036/redis.log

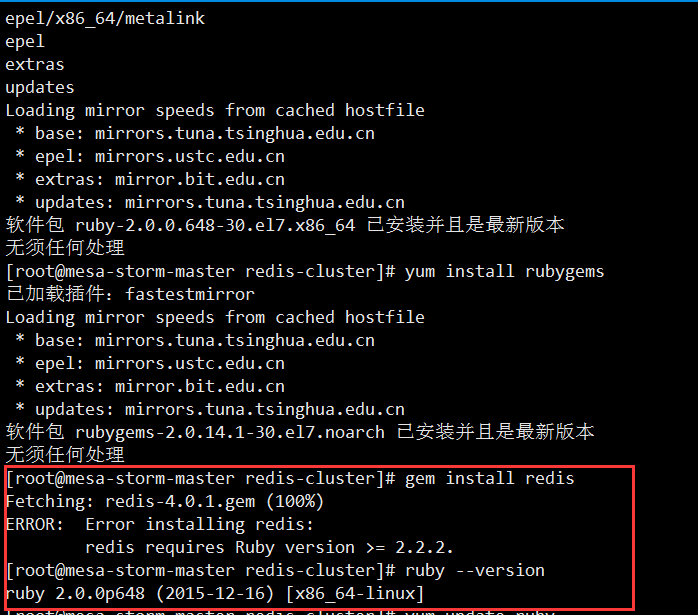


### ****安装ruby****

由于Redis 集群客户端实现很少，redis集群的启动需要用到ruby实现的redis-trib.rb，所以我们需要先安装ruby。

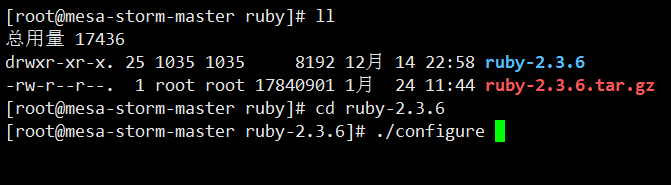
|  |
| --- |
| yum install ruby  yum install rubygems  gem install redis |

执行gem install redis时提示ruby版本太老,可以发现centos7默认的ruby版本是2.0.0



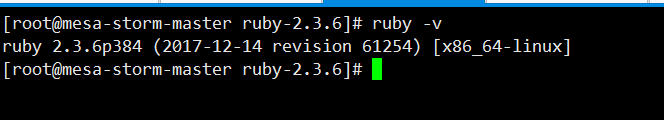
#### 下面开始更新ruby

解压ruby源码包并执行./configure



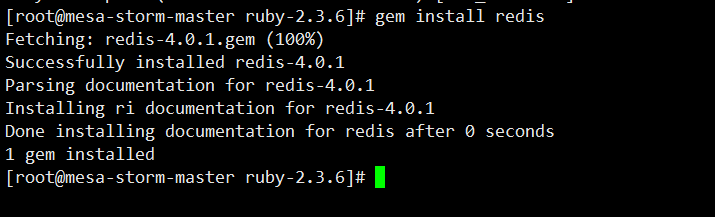
接着执行make 和make install

再次查看版本号发现更新ruby成功



再次执行

|  |
| --- |
| gem install redis |



### ****启动redis实例****

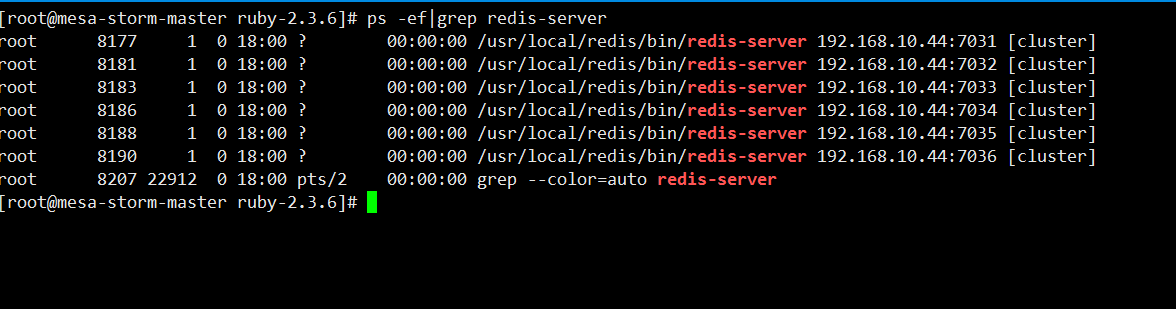
/usr/local/redis/bin/redis-server /usr/local/redis-cluster/7031/redis.conf

分别启动6个redis实例。也可以用脚本循环启动，这样更方便省时

#### 批量启动

|  |
| --- |
| for((i=1;i<=6;i++)); do /usr/local/redis/bin/redis-server /usr/local/redis-cluster/703$i/redis.conf; done |

查看是否创建成功



#### 批量关闭

|  |
| --- |
| for((i=1;i<=6;i++)); do /usr/local/redis/bin/redis-cli -c -h 192.168.10.44 -p 703$i -a 123456 shutdown; done |

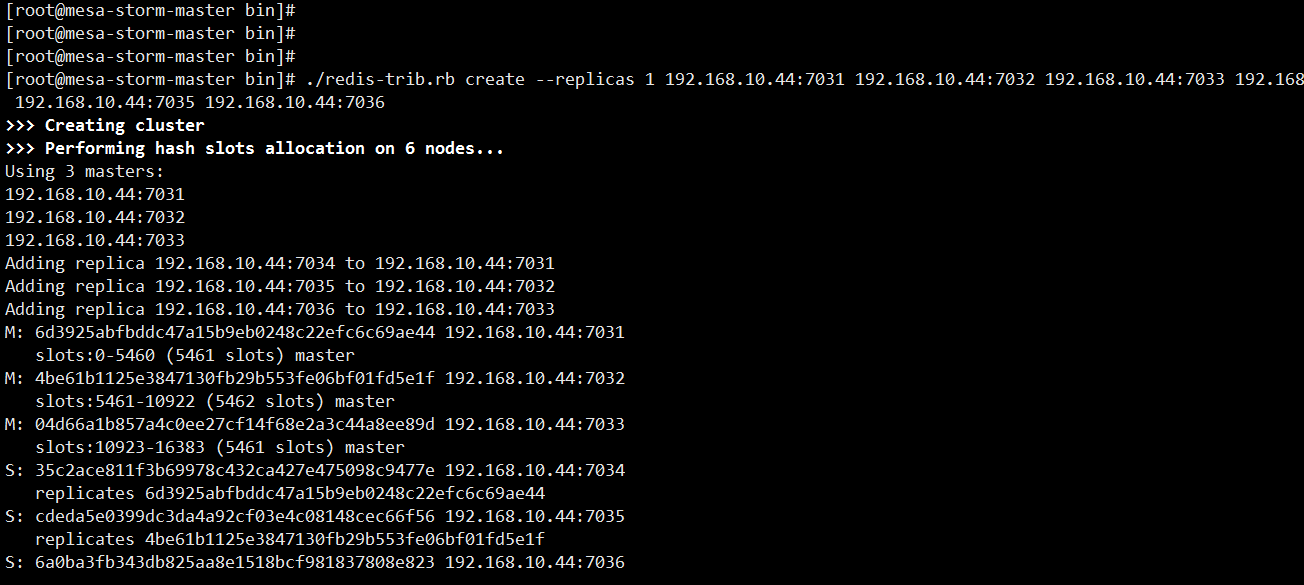
### ****创建并启动集群****

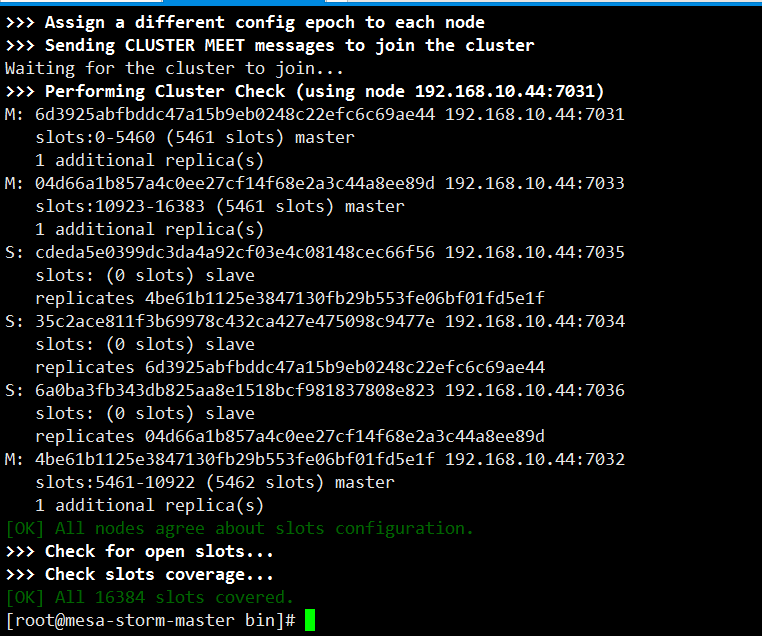
进入redis安装目录的bin目录下

|  |
| --- |
| /usr/local/redis/bin/redis-trib.rb create --replicas 1 192.168.10.44:7031 192.168.10.44:7032 192.168.10.44:7033 192.168.10.44:7034 192.168.10.44:7035 192.168.10.44:7036 |

命令的意义如下：

给定 redis-trib.rb 程序的命令是 create，表示创建一个新的集群。选项 --replicas 1 表示为集群中的每个主节点创建一个从节点。之后跟着的其他参数则是实例的地址列表， 指定使用这些地址所指示的实例来创建新集群。





“All 16384 slots covered.”表示集群中的 16384 个槽都有至少一个主节点在处理， 集群运作正常。从打印出来的信息也可以看出，7031，7032，7033是主节点，其它三个是从节点。

### ****客户端连接集群****

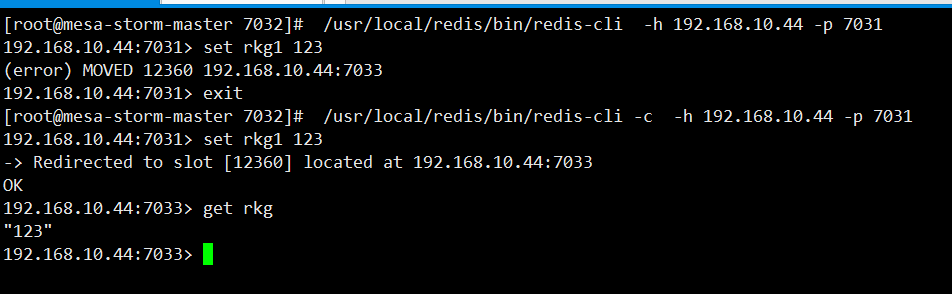
集群启动成功后，我们就可以用任意一个客户端连接集群了，如下

|  |
| --- |
| /usr/local/redis/bin/redis-cli -c -h 192.168.10.44 -p 7031 |

注意在客户端连接时一定要加参数c否则会报如下错误

(error) MOVED 12360

可以查看redis-cli --help 会发现-c参数是集群模式启动的意思

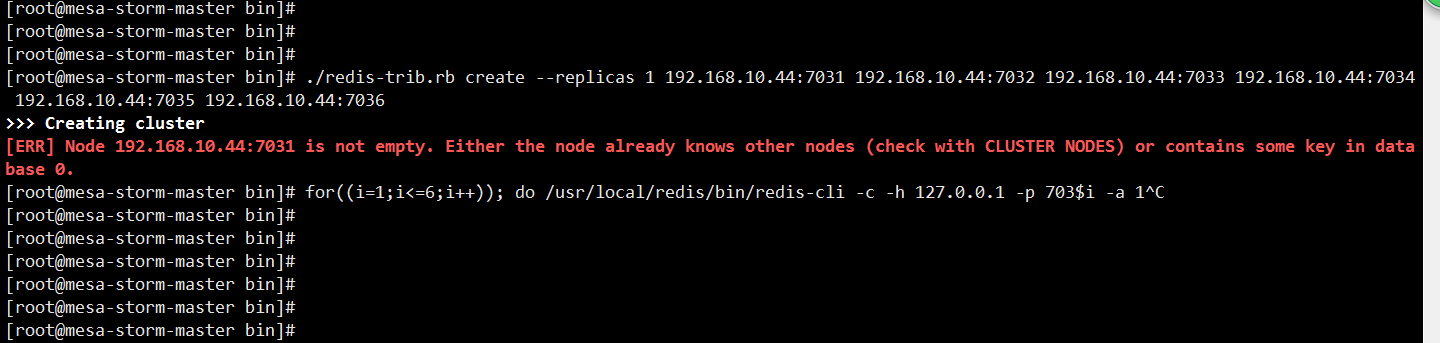


### ****集群关闭****

关闭集群需要逐个关闭

|  |
| --- |
| for((i=1;i<=6;i++)); do /usr/local/redis/bin/redis-cli -c -h 192.168.10.44 -p 703$i -a 123456 shutdown; done |

如果重新启动集群报以下错误

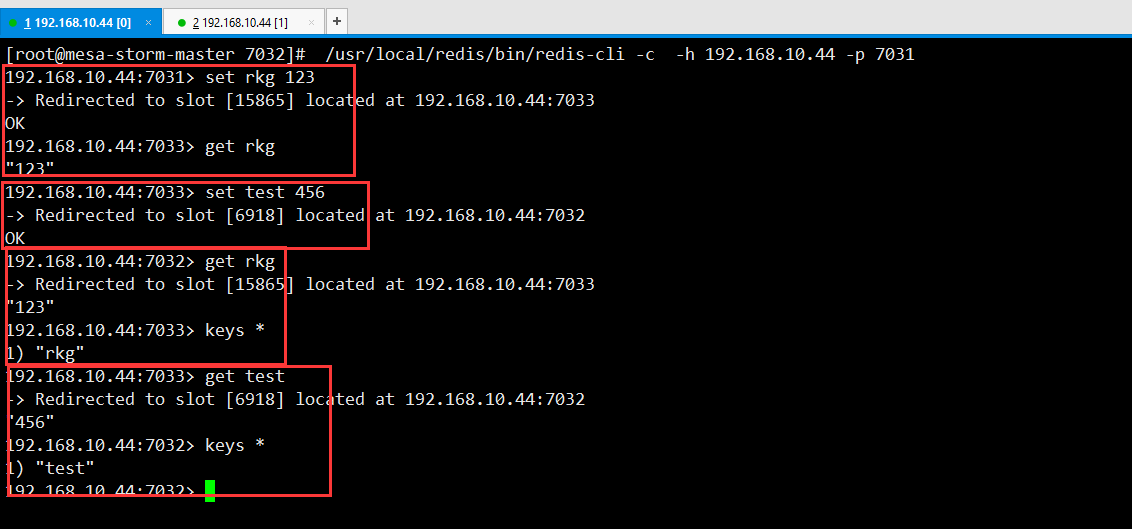


因为已经创建过集群了,所以再次启动集群会报错

需要清除杀掉redis实例，然后删除每个节点下的临时数据文件appendonly.aof，dump.rdb，nodes-703x.conf，然后再重新启动redis实例即可启动集群。

|  |
| --- |
| for((i=1;i<=6;i++)); do cd 703$i; rm -rf appendonly.aof; rm -rf dump.rdb; rm -rf nodes-703$i.conf; cd ..; done |

### ****集群测试****



通过设置key和获取key可以发现客户端连接加-c选项的时候，存储和提取key的时候不断在7033和7032之间跳转，这个称为客户端重定向。之所以发生客户端重定向，是因为Redis Cluster中的每个Master节点都会负责一部分的槽(slot)，存取的时候都会进行键值空间计算定位key映射在哪个槽(slot)上，如果映射的槽(slot)正好是当前Master节点负责则直接存取，否则就跳转到其他Master节点负的槽(slot)中存取，这个过程对客户端是透明的。

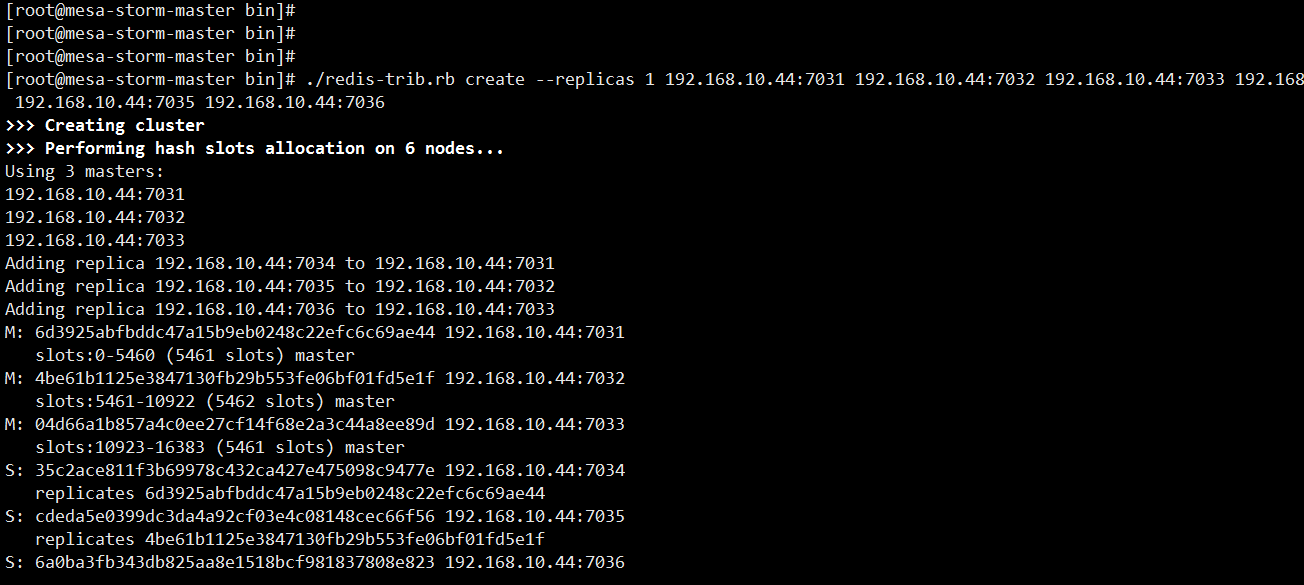
## ****Redis集群分区原理****

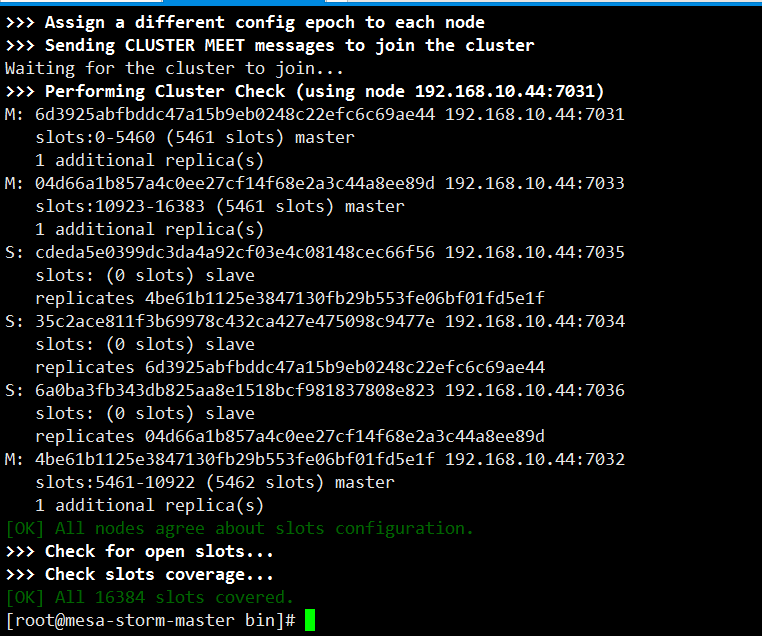
### ****槽(slot)的基本概念****

从上面集群的简单操作中，我们已经知道redis存取key的时候，都要定位相应的槽(slot)。

Redis 集群键分布算法使用数据分片（sharding）而非一致性哈希（consistency hashing）来实现： 一个 Redis 集群包含 16384 个哈希槽（hash slot）， 它们的编号为0、1、2、3……16382、16383，这个槽是一个逻辑意义上的槽，实际上并不存在。redis中的每个key都属于这 16384 个哈希槽的其中一个，存取key时都要进行key->slot的映射计算。

下面我们来看看启动集群时候打印的信息:





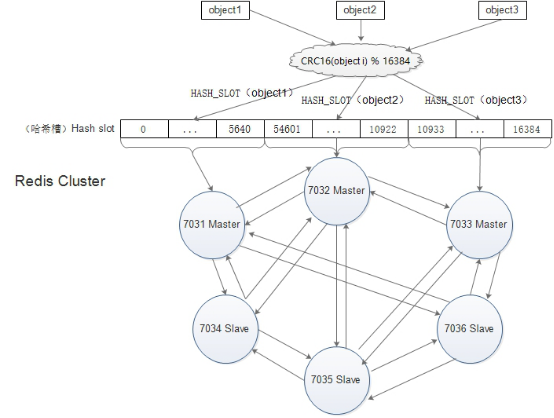
从上面信息可以看出，创建集群的时候，哈希槽被分配到了三个主节点上，从节点是没有哈希槽的。7031负责编号为0-5460 共5461个 slots，7032负责编号为 5461-10922共5462 个 slots，7033负责编号为10923-16383 共5461个 slots。

### ****键-槽映射算法****

和memcached一样，redis也采用一定的算法进行键-槽（key->slot）之间的映射。memcached采用一致性哈希（consistency hashing）算法进行键-节点（key-node）之间的映射，而redis集群使用集群公式来计算键 key 属于哪个槽：

|  |
| --- |
| HASH\_SLOT（key）= CRC16(key) % 16384 |

其中 CRC16(key) 语句用于计算键 key 的 CRC16 校验和 。key经过公式计算后得到所对应的哈希槽，而哈希槽被某个主节点管理，从而确定key在哪个主节点上存取，这也是redis将数据均匀分布到各个节点上的基础。



键-槽-节点（key->slot->node）映射示意图

### ****集群分区好处****

无论是memcached的一致性哈希算法，还是redis的集群分区，最主要的目的都是在移除、添加一个节点时对已经存在的缓存数据的定位影响尽可能的降到最小。redis将哈希槽分布到不同节点的做法使得用户可以很容易地向集群中添加或者删除节点， 比如说：

如果用户将新节点 D 添加到集群中， 那么集群只需要将节点 A 、B 、 C 中的某些槽移动到节点 D 就可以了。

与此类似， 如果用户要从集群中移除节点 A ， 那么集群只需要将节点 A 中的所有哈希槽移动到节点 B 和节点 C ， 然后再移除空白（不包含任何哈希槽）的节点 A 就可以了。

因为将一个哈希槽从一个节点移动到另一个节点不会造成节点阻塞， 所以无论是添加新节点还是移除已存在节点， 又或者改变某个节点包含的哈希槽数量， 都不会造成集群下线，从而保证集群的可用性。

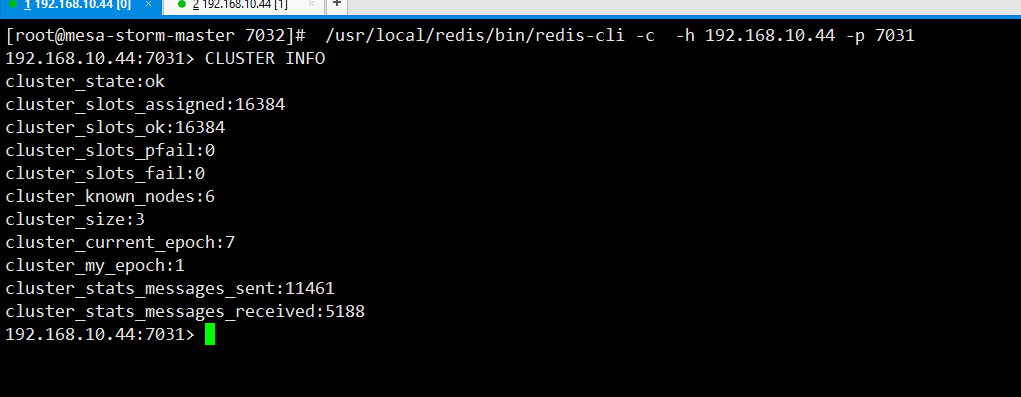
## ****集群操作****

集群操作包括查看集群信息，查看集群节点信息，向集群中增加节点、删除节点，重新分配槽等操作。

### ****查看集群信息****

cluster info 查看集群状态，槽分配，集群大小等，cluster nodes也可查看主从节点。

|  |
| --- |
| cluster info |



### ****新增节点****

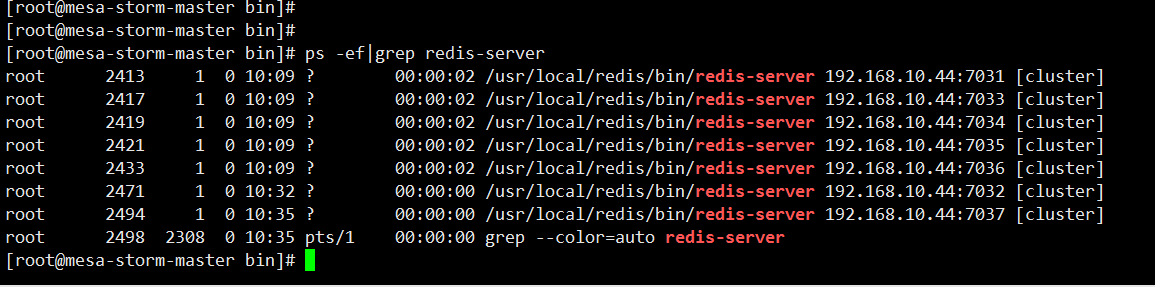
#### 新增节点配置文件

执行下面的脚本创建脚本配置文件

|  |
| --- |
| mkdir /usr/local/redis-cluster/7037 && cp /usr/local/redis-cluster/7031/redis.conf /usr/local/redis-cluster/7037/redis.conf && sed -i "s/7031/7037/g" /usr/local/redis-cluster/7037/redis.conf |

#### 启动新增节点

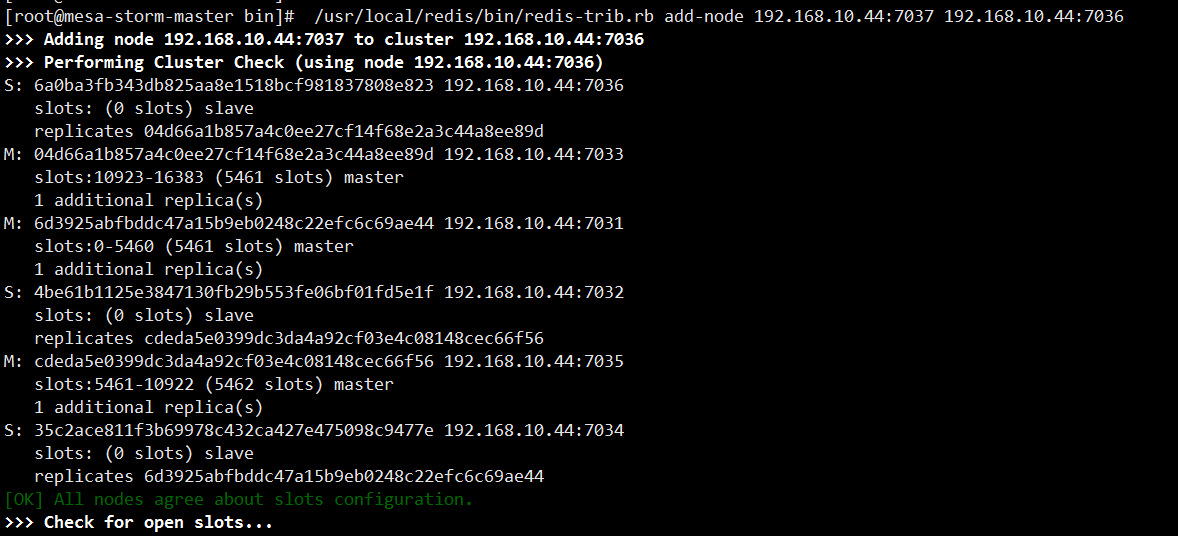
|  |
| --- |
| /usr/local/redis/bin/redis-server /usr/local/redis-cluster/7037/redis.conf |

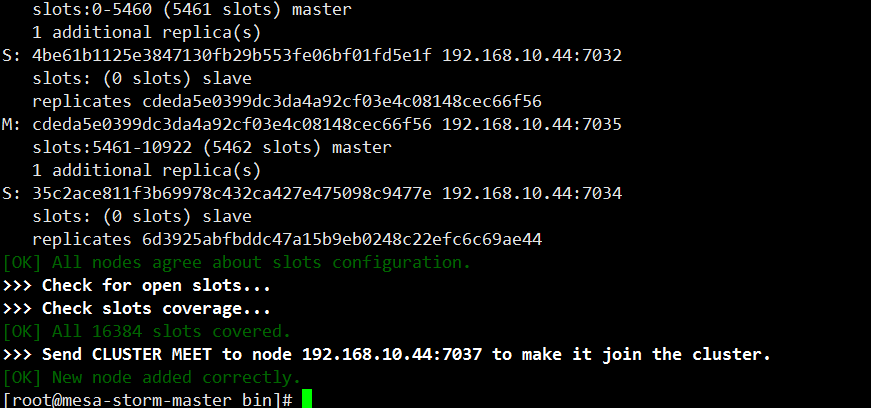


#### 添加节点到集群

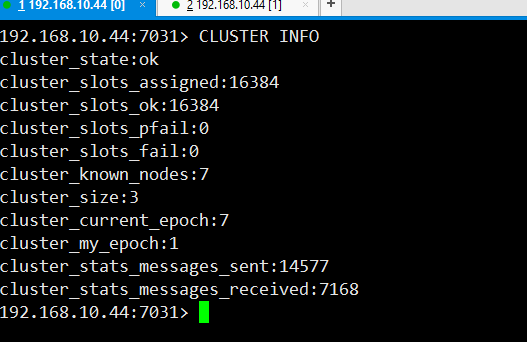
现在已经添加了新增一个节点所需的配置文件，但是这个这点还没有添加到集群中，现在让它成为集群中的一个主节点

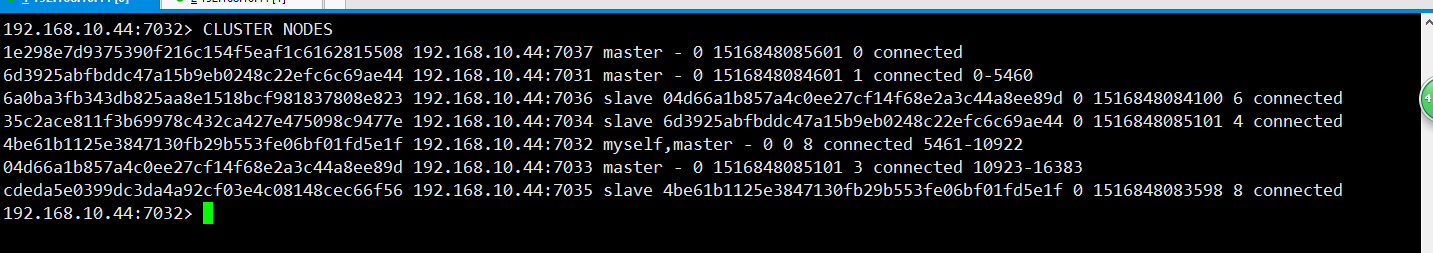
|  |
| --- |
| /usr/local/redis/bin/redis-trib.rb add-node 192.168.10.44:7037 192.168.10.44:7036 |





./redis-trib.rb add-node 命令中，7037 是新增的主节点，7036 是集群中已有的从节点。再来看看集群信息

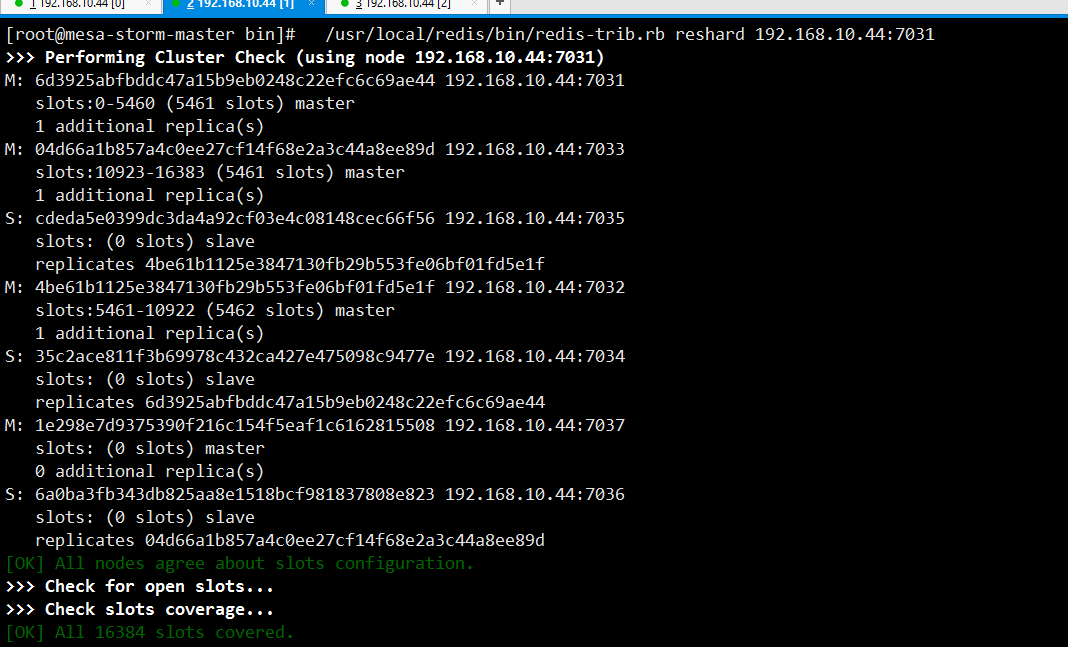




#### 分配槽

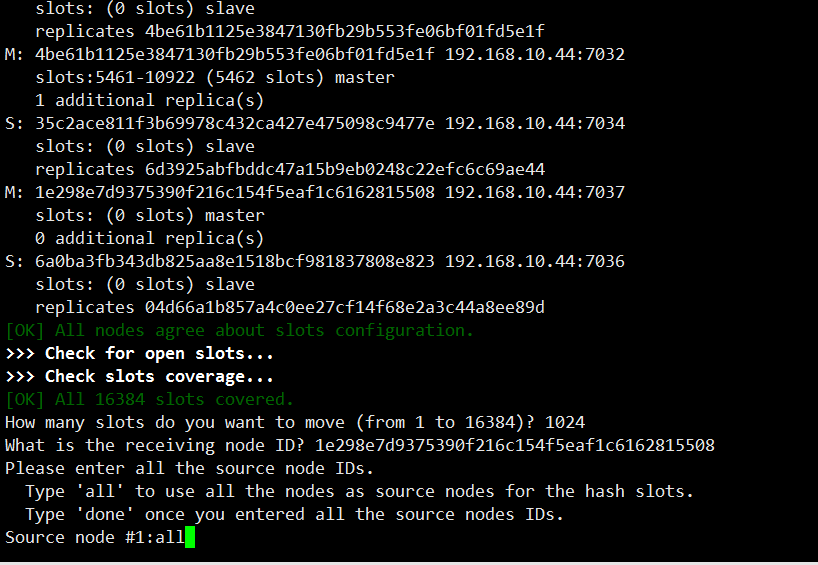
从添加主节点输出信息和查看集群信息中可以看出，我们已经成功的向集群中添加了一个主节点，但是这个主节还没有成为真正的主节点，因为还没有分配槽（slot），也没有从节点，现在要给它分配槽（slot）

|  |
| --- |
| /usr/local/redis/bin/redis-trib.rb reshard 192.168.10.44:7031 |

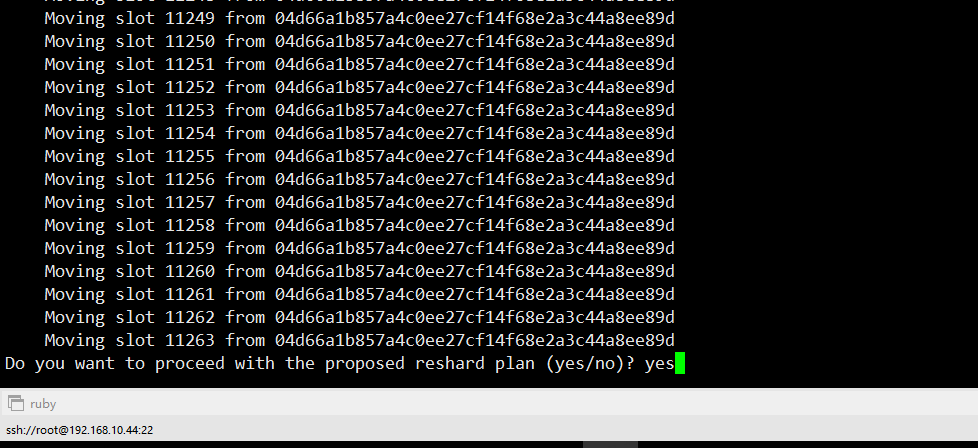


系统提示要移动多少个配槽（slot）,并且配槽（slot）要移动到哪个节点，任意输入一个数，如1024，再输入新增节点的ID 1e298e7d9375390f216c154f5eaf1c6162815508

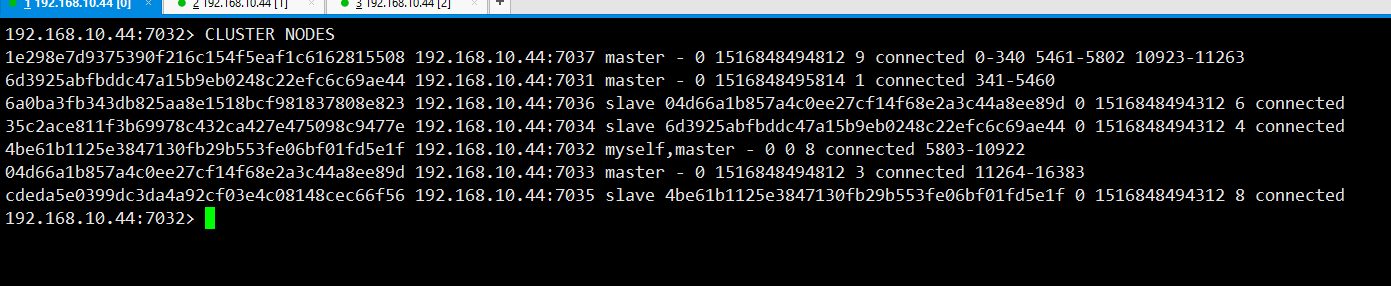
 然后提示要从哪几个节点中移除1024个槽（slot），这里输入‘all’表示从所有的主节点中随机转移，done-> 数据要提取slot的master节点id，最后用done结束 ,凑够1024个哈希槽，然后就开始从新分配槽（slot）了。从新分配完后再次查看集群节点信息



是否确定重新分配,确定yes



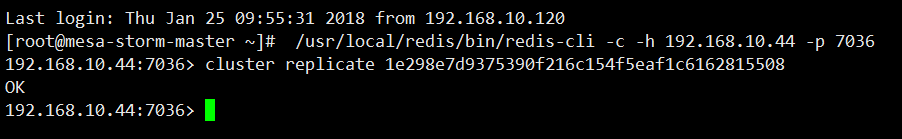
此时再次查看各节点信息可见，0-340 5461-5802 10923-11263的槽（slot）被分配给了新增加的节点。三个加起来刚好1024个槽（slot）。



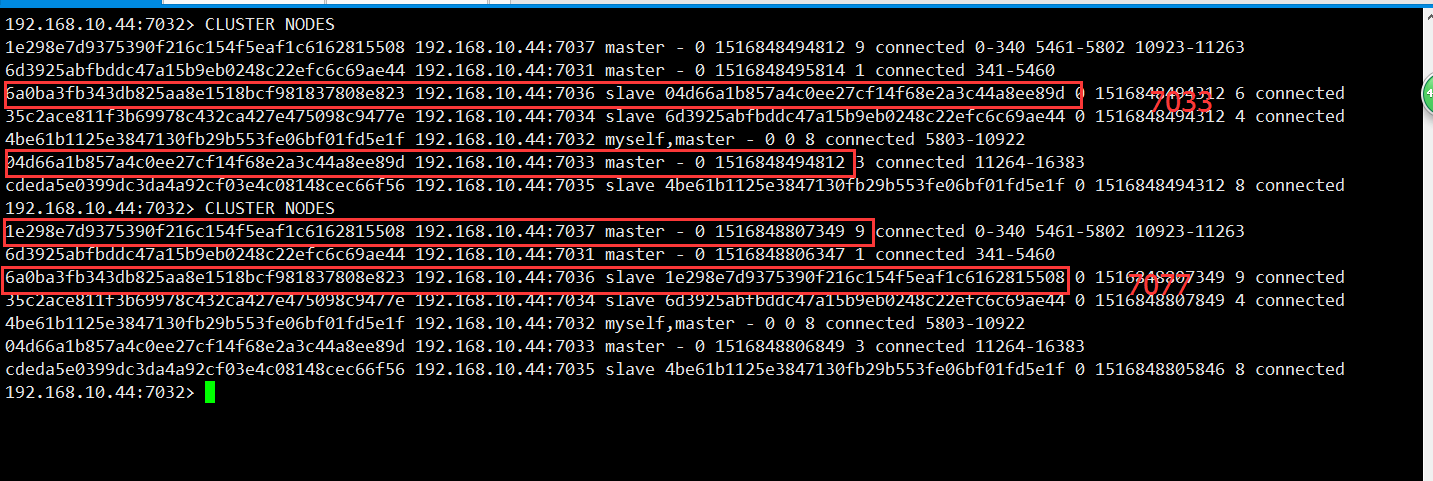
#### 指定从节点

现在从节点7036的主节点是7033，现在我们要把他变为新增加节点（7037）的从节点，需要登录7036的客户端

|  |
| --- |
| /usr/local/redis/bin/redis-cli -c -h 192.168.10.44 -p 7036 |
| cluster replicate 1e298e7d9375390f216c154f5eaf1c6162815508 |

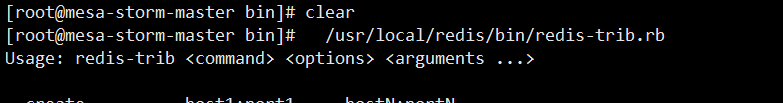


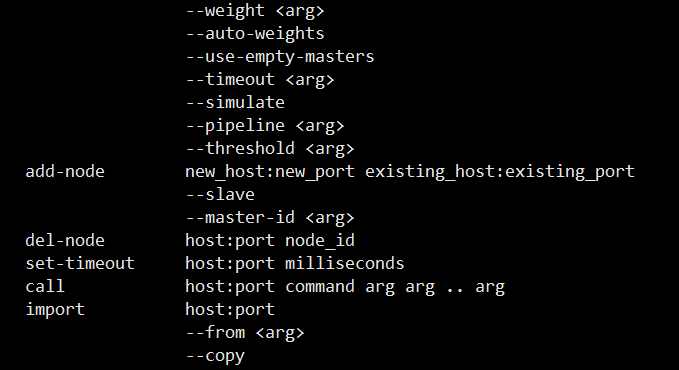
再来查看集群节点信息,发现7036从之前7033的从节点变更为7037的从节点



#### ****删除节点****

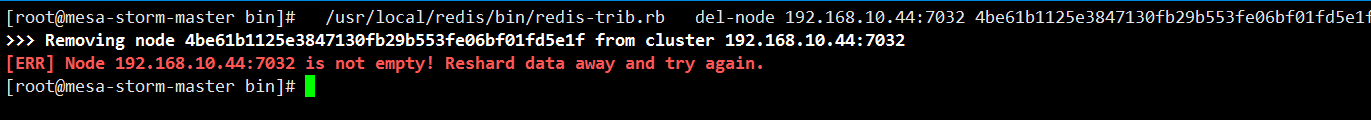
查看帮助,发现删除节点可以使用del-node host:port node\_id

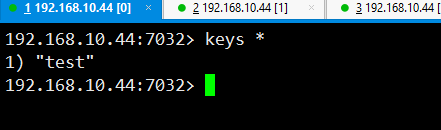




删除7032节点

|  |
| --- |
| /usr/local/redis/bin/redis-trib.rb del-node 192.168.10.44:7032 4be61b1125e3847130fb29b553fe06bf01fd5e1f |





移除节点前，要先保证它是空的即先要Reshard将哈希槽转移。

|  |
| --- |
| /usr/local/redis/bin/redis-trib.rb reshard 192.168.10.44:7032 |

How many slots do you want to move (from 1 to 16384)?7032中所拥有的槽即可,这里输入16384

因为我们要删除7032节点,  这里指定接收这些slots的目标节点7032节点6d3925abfbddc47a15b9eb0248c22efc6c69ae44

// 选择指定slot的来源

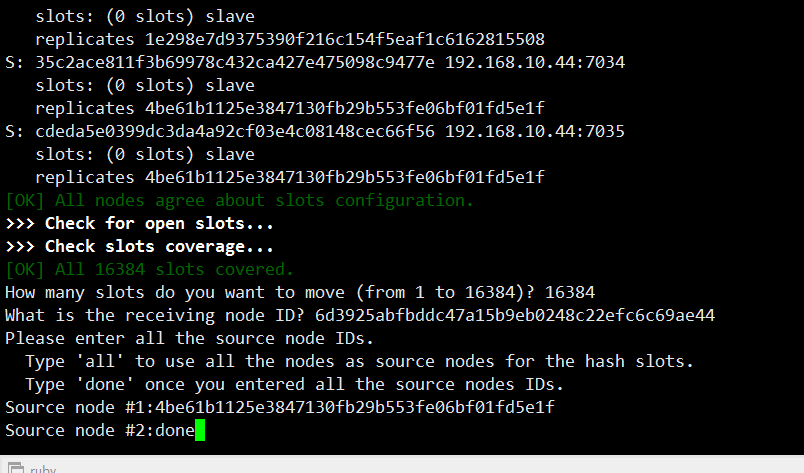
// all->代表从所有的master中重新分配

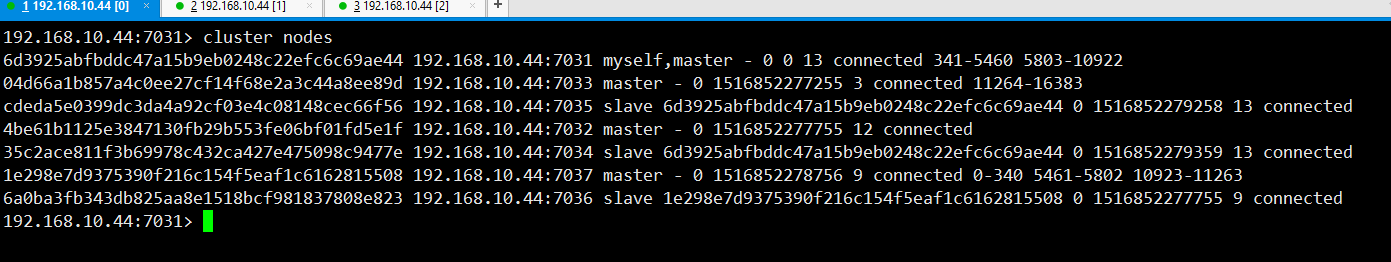
// done-> 数据要提取slot的master节点id，最后用done结束

 因为我们要删除7032这个节点,所以源节点的id就是7032的节点ID

4be61b1125e3847130fb29b553fe06bf01fd5e1f

//输入done,回车,就会开始从7032这个节点迁移16384个slot(没有这么多就迁移拥有的全部)到7031节点中去.





可以发现7032中以没有槽了

### ****集群操作小结****

从上面过程可以看出，添加节点、分配槽、删除节点的过程，不用停止集群，不阻塞集群的其他操作。

|  |
| --- |
| #向集群中添加节点，7037是新增节点，7036是集群中已有的节点  ./redis-trib.rb add-node 192.168.10.44:7037 192.168.10.44:7036  #重新分配槽  ./redis-trib.rb reshard 192.168.10.44:7031  #指定当前节点的主节点  cluster replicate cf48228259def4e51e7e74448e05b7a6c8f5713f  #删除节点  ./redis-trib.rb del-node 192.168.10.44:7037 'a56461a171334560f16652408c2a45e629d268f6' |

# 主从模式、sentinel模式、cluster模式小结

## 单节点实例

单节点实例是最简单的，平时做个测试，写个小程序如果需要用到缓存的话，启动一个Redis还是很轻松的，做为一个key/value数据库也是可以胜任的。参考上文：[安装redis](#_安装redis)

## 主从模式

主从模式的简介和配置，此处不再说，参照上文：[主从复制](#_主从复制)

主从模式介绍

1. 主从模式的一个作用是备份数据，这样当一个节点损坏（指不可恢复的硬件损坏）时，数据因为有备份，可以方便恢复。
2. 另一个作用是负载均衡，所有客户端都访问一个节点肯定会影响Redis工作效率，有了主从以后，查询操作就可以通过查询从节点来完成。
3. 一个Master可以有多个Slaves
4. 默认配置下，master节点可以进行读和写，slave节点只能进行读操作，写操作被禁止
5. 不要修改配置让slave节点支持写操作，没有意义，原因一，写入的数据不会被同步到其他节点；原因二，当master节点修改同一条数据后，slave节点的数据会被覆盖掉
6. slave节点挂了不影响其他slave节点的读和master节点的读和写，重新启动后会将数据从master节点同步过来
7. master节点挂了以后，不影响slave节点的读，Redis将不再提供写服务，master节点启动后Redis将重新对外提供写服务。
8. master节点挂了以后，不会slave节点重新选一个master
9. 对有密码的情况说明一下，当master节点设置密码时：

* 客户端访问master需要密码
* 启动slave需要密码，在配置中进行配置即可
* 客户端访问slave不需要密码

1. 主从模式的缺点

* master节点挂了以后，redis就不能对外提供写服务了，因为剩下的slave不能成为master
* 这个缺点影响是很大的，尤其是对生产环境来说，是一刻都不能停止服务的，所以一般的生产坏境是不会单单只有主从模式的。所以有了下面的sentinel模式。

## sentinel模式

sentinel模式的配置参照上文:[Redis-Sentinel(redis哨兵)](#_Redis-Sentinel(redis哨兵))

sentinel的中文含义是哨兵、守卫。也就是说既然主从模式中，当master节点挂了以后，slave节点不能主动选举一个master节点出来，那么我就安排一个或多个sentinel来做这件事，当sentinel发现master节点挂了以后，sentinel就会从slave中重新选举一个master。

对sentinel模式的理解：

* sentinel模式是建立在主从模式的基础上，如果只有一个Redis节点，sentinel就没有任何意义
* 当master节点挂了以后，sentinel会在slave中选择一个做为master，并修改它们的配置文件，其他slave的配置文件也会被修改，比如slaveof属性会指向新的master
* 当master节点重新启动后，它将不再是master而是做为slave接收新的master节点的同步数据
* sentinel因为也是一个进程有挂掉的可能，所以sentinel也会启动多个形成一个sentinel集群
* 当主从模式配置密码时，sentinel也会同步将配置信息修改到配置文件中，不许要担心。
* 一个sentinel或sentinel集群可以管理多个主从Redis。
* sentinel最好不要和Redis部署在同一台机器，不然Redis的服务器挂了以后，sentinel也挂了
* sentinel监控的Redis集群都会定义一个master名字，这个名字代表Redis集群的master Redis。

 当使用sentinel模式的时候，客户端就不要直接连接Redis，而是连接sentinel的ip和port，由sentinel来提供具体的可提供服务的Redis实现，这样当master节点挂掉以后，sentinel就会感知并将新的master节点提供给使用者。

　　sentinel模式基本可以满足一般生产的需求，具备高可用性。但是当数据量过大到一台服务器存放不下的情况时，主从模式或sentinel模式就不能满足需求了，这个时候需要对存储的数据进行分片，将数据存储到多个Redis实例中，就是下面要讲的。

## cluster模式

cluster模式的参照上文：[redis-cluster(redis集群)](#_redis-cluster(redis集群))

　 cluster的出现是为了解决单机Redis容量有限的问题，将Redis的数据根据一定的规则分配到多台机器。对cluster的一些理解：

* cluster可以说是sentinel和主从模式的结合体，通过cluster可以实现主从和master重选功能，所以如果配置两个副本三个分片的话，就需要六个Redis实例。
* 因为Redis的数据是根据一定规则分配到cluster的不同机器的，当数据量过大时，可以新增机器进行扩容

这种模式适合数据量巨大的缓存要求，当数据量不是很大使用sentinel即可。

# 安装redis监控工具redislive

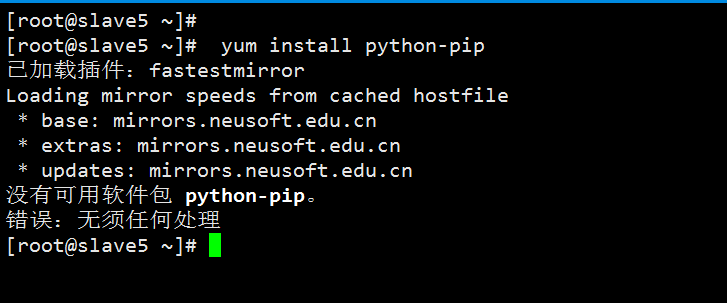
redislive依赖于python,需要先安装pip(python包管理工具)

## pip安装步骤

|  |
| --- |
| 可参考:  <http://blog.csdn.net/yulei_qq/article/details/52984334> |
| <https://www.cnblogs.com/madyina/p/4060556.html> |

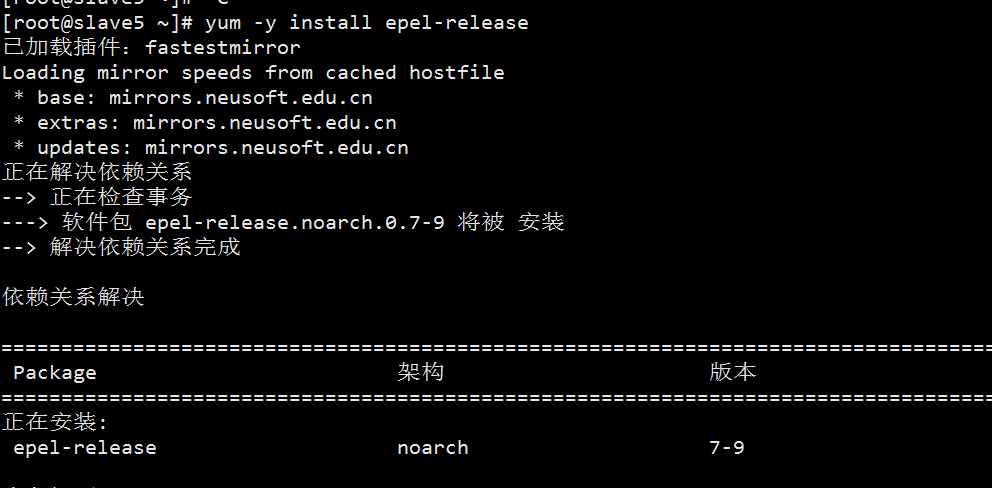
首先检查linux有没有安装python-pip包，直接执行

|  |
| --- |
| yum install python-pip |



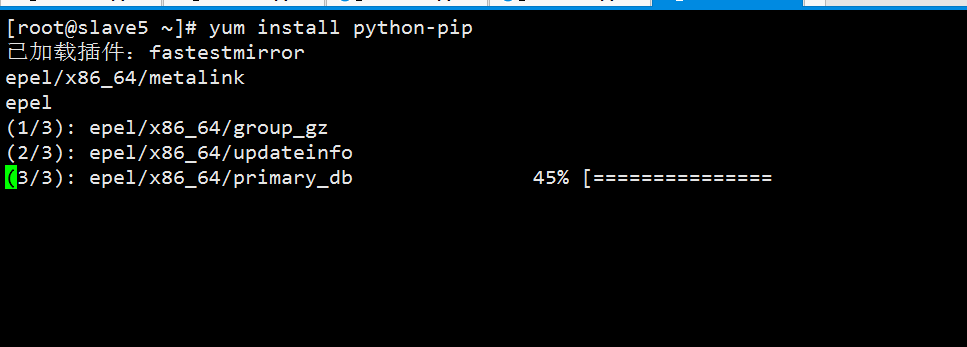
没有python-pip包就执行命令

|  |
| --- |
| yum -y install epel-release |



执行成功之后，再次执行

|  |
| --- |
| yum install python-pip |

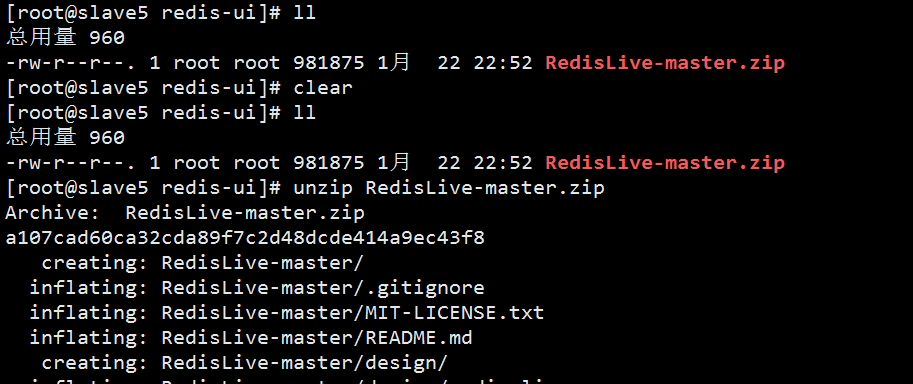


对安装好的pip进行升级

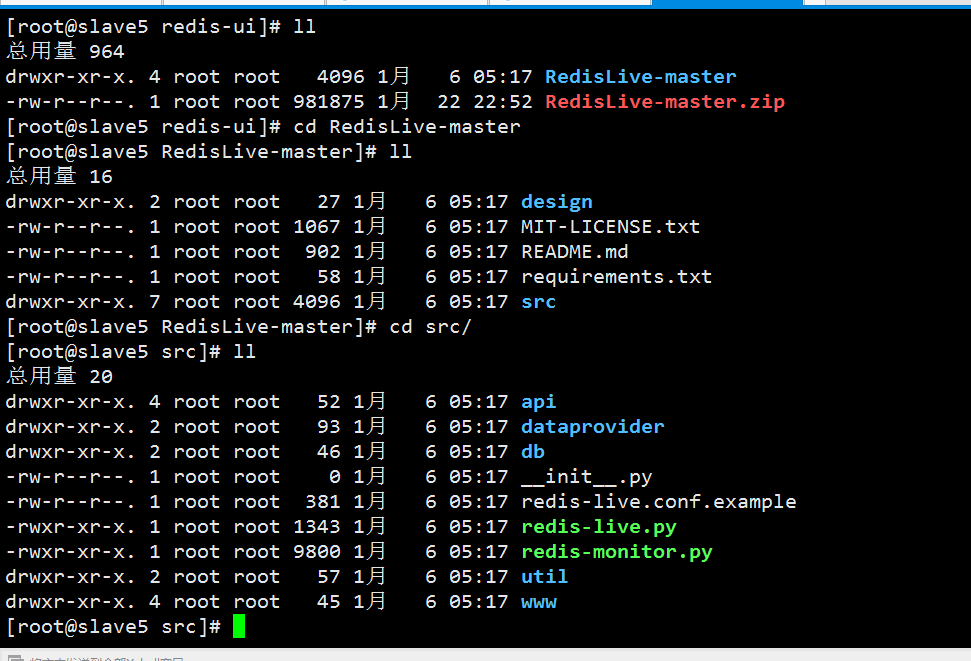
|  |
| --- |
| pip install --upgrade pip |

## 安装redislive

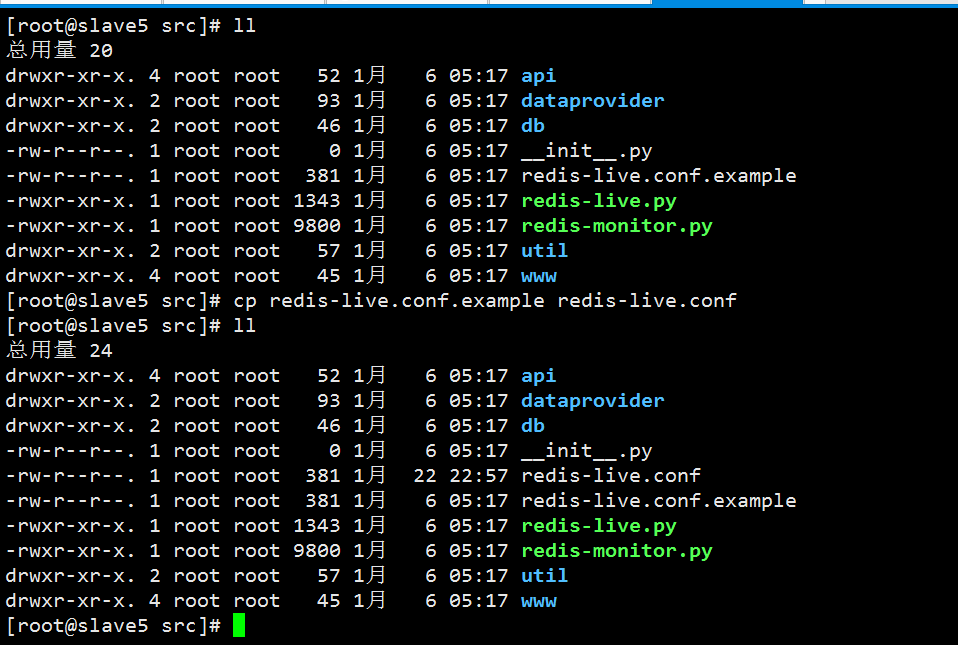
下载并解压redislive



进入src目录



复制redis-live.conf.example为redis-live.conf并修改内容如下



redis-live.conf

|  |
| --- |
| {  "RedisServers":[  {  "server":"192.168.10.44",  "port":6379  },  {  "server":"localhost",  "port":6380  }  ],  "DataStoreType":"redis",  "RedisStatsServer":{  "server":"192.168.10.44",  "port":6379  },  "SqliteStatsStore":{  "path":"to your sql lite file"  }  } |

开启监控脚本：

|  |
| --- |
| ./redis-monitor.py --duration 120 |

开启webserver：

|  |
| --- |
| ./redis-live.py |

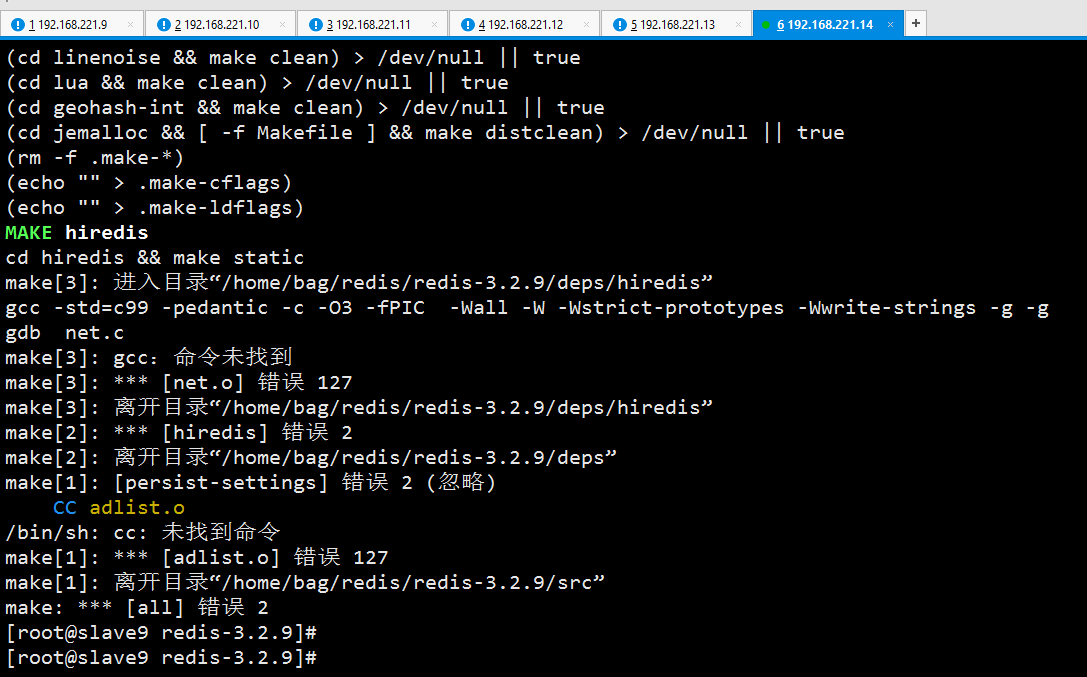
浏览器打开即可,默认端口8888,注意一定要加后面的index.html,否则403错误

http://192.168.10.44:8888/index.html

# 常见错误

## 缺少gcc

出现如下图所示错误时安装gcc即可



## make错误

出现下图错误时使用make MALLOC=libc命令即可

