# Redis特性

## 版本特性

**2.x版本**

2.8-Redis sentinle HA；

2.8-Psync1，主从支持增量复制；

2.8-键空间通知，订阅指定key执行了哪些事件；

**3.x版本**

Redis Cluster高可用+分片集群解决方案；

WAIT命令；

MIGRATE多键操作，性能优化，COPY&REPLACE；

**4.x版本**

Key的惰性删除，LAZY FREEING配置项；

RDB-AOF，支持aof和rdb混合加载模式恢复数据；

MEMORY内存命令；

Psync2：更多的场景支持增量复制；

Slowlog：问题排查；

**5.x版本**

Stream Data Type；

RDB中保存LFU和LRU信息；

Redis-cli支持cluster操作；

Active defregmentation V2；

## 数据结构

|  |  |
| --- | --- |
| **对象类型** | **使用场景** |
| String | 缓存功能 |
| Hash | 读取并修改一个对象的属性时，如果使用String比较麻烦，通常需要将字符串（JSON）取出来，然后进行反序列化，修改某一项的值，在序列化成字符串（JSON）存储回去 |
| List | redis的lpush+brpop命令组合即可实现阻塞队列，生产者客户端是用lupsh从列表左侧插入元素， 多个消费者客户端使用brpop命令阻塞时的“抢”列表尾部的元素，多个客户端保证了消费的负载均衡和高可用性 |
| Set | 在微博应用中，可以将一个用户所有的关注人存在一个集合中，将其所有粉丝存在一个集合。Redis还为集合提供了求交集、并集、差集等操作，可以非常方便的实现如共同关注、共同喜好、二度好友等功能 |
| Zset（Sorted Set） | 例如视频网站需要对用户上传的视频做排行榜，榜单维护可能是多方面： 按照时间、按照播放量、按照获得的赞数等 |

## 主从复制

**复制过程**

slaveof指令执行，获取主节点信息->建立socket连接(定时任务直到成功)->从节点发送ping指令->主节点响应pong指令->密码验证->首次同步（全量复制）->持续数据同步

**异步复制**

数据复制的过程是异步的，主节点在接收到客户端的写命令后直接返回成功，并不等待从节点数据复制完成

**全量复制**

主节点bgsave生成rdb->发送rdb文件给从节点->从节点保存rdb文件->清除旧数据->加载rdb文件->是否AOF持久化

**部分复制**

主从中断导致的数据丢失，当从节点再次连上主节点后，仅补发丢失的数据从而避免了全量复制。复制偏移量mater\_repl\_offset和slave\_offset，可以通过info replication查看。

psync [runId] [offset]：runId传主节点id，offset传-1表示全量，或者是slave\_offset，然后主节点会在backlog缓冲区中查找是否有这个offset后的数据，有则进行部分复制

## 持久化

### RDB持久化

rdb快照存储，将Redis在内存中的数据库状态保存到磁盘中去。

SAVE指令，阻塞redis创建rdb文件；

BGSAVE指令，fork一个子进程，由子进程创建rdb文件；

rdb文件载入是在服务器启动时会自动执行的，如果AOF持久化功能开启，则优先使用AOF文件来还原数据库状态；

自动持久化策略，saveparams数据配置save条件，如：每10s更新500次，会遍历配置的save条件，满足即执行一次rdb持久化

dirty计数器：继上次save或者bgsave后对数据库（所有）进行了多少次修改（写入更新删除）。

lastsave：上次save或者bgsave的时间戳。

会有一个周期性函数去定期（100ms）遍历saveparams条件，判断是否需要bgsave。

**优点：**

**rdb文件数据完整性高，恢复速度快，很适合做灾备版本控制，如每天备份等**

**缺点：**

**当数据集过大的时候fork操作会耗时变长并且会阻塞客户端请求，**

**半持久化，数据的时效性难以得到保证，会丢失上次bgsave到异常期间的所有写入数据。**

### AOF持久化

文件写入和同步：调用write指令写文件，通常会先将内容暂时保存在内存缓冲区中，当缓冲区被塞满或者指定的时间后，才会将数据真正写入磁盘。可能存在数据丢失的问题。

1、命令追加>>缓冲区，当缓冲区满了或者超过指定的时间将数据从缓冲区写入文件。

2、这里就有一个同步策略问题，通过appendfsync配置项设置：

always:每次事件循环都要将aof\_buf缓冲区中的数据同步到磁盘；

everysec:每秒将缓冲区中的数据同步到磁盘中，最多只会丢失1s的数据；

no:完全依赖操作系统决定何时将数据同步到磁盘中，写入速度最快，但是会丢失上次AOF同步之后的所有写数据。

AOF重写：解决AOF文件过大的问题，占用空间大、恢复数据慢。

aof\_rewrite重新读取数据库当前状态，仅保留还原当前数据库状态必须的命令，这个过程也是fork一个子进程来实现，并且在子进程重写AOF文件的过程中，新的写命令也会追加到一个AOF重写缓冲区中。

**优点：持久化写入时只是追加操作日志文件，灵活配置同步策略，如每秒同步一次，性能又好，异常时也仅丢失1s的数据。**

**缺点：文件体积大（但是提供了AOF重写功能），根据同步策略，速度可能会比rdb慢。数据恢复重载AOF文件的速度慢并且可能会出现个别命令导致数据完整性可能会有差异。**

### 持久化策略

* 如果你可以接受一段时间内的数据丢失（几分钟），可以仅适用RDB；
* 如果对数据的一致性要求较高，可以使用AOF，但同时也可以定期给数据做一个完整的快照，便于快速恢复；
* 可以使用主从机制配合持久化，主节点不做持久化而交给从节点来做。

**二者同时存在时，默认AOF恢复的优先级高于RDB，因为他的数据更完整。**

**4.0新特性，支持aof和rdb混合加载模式恢复数据，先执行rdb的全量恢复，再追加aof增量日志数据；详见Redis配置详解持久化配置**

# 部署运维

## Redis安装

### 在线安装

**redis-install-online.sh脚本如下**

readonly redis\_path="/root/redis"

readonly redis\_version\_tar\_gz="redis-4.0.10.tar.gz"

rpm -q gcc

if [[ $? != 0 ]];then

echo "isntall gcc"

yum install -y gcc

fi

mkdir -p ${redis\_path} && cd ${redis\_path}

wget http://download.redis.io/releases/${redis\_version\_tar\_gz} && tar zxvf ${redis\_version\_tar\_gz} --strip-component=1

make && make install

if [[ $? == 0 ]]; then

echo "redis install success"

fi

##可以在任何地方直接执行redis-cli等操作

cd src

cp redis-cli redis-server redis-sentinel /usr/bin/

### 离线安装

Gcc安装

Redis安装

### Redis-trib.rb集群管理工具依赖安装

#### 安装ruby

wget https://cache.ruby-lang.org/pub/ruby/2.5/ruby-2.5.1.tar.gz

tar zxvf ruby-2.5.1.tar.gz

cd ruby-2.5.1

./configure

make && make install

gem install redis

cp redis-trib.rb /usr/bin/

#### 常见问题

##### Ruby版本

ERROR: Error installing redis:

redis requires Ruby version >= 2.2.2.

自动安装的ruby版本可能不够，gem需要2.2.2以上的版本，这里选的是2.5.1

##### zlib依赖

ERROR: Loading command: install (LoadError)

cannot load such file – zlib

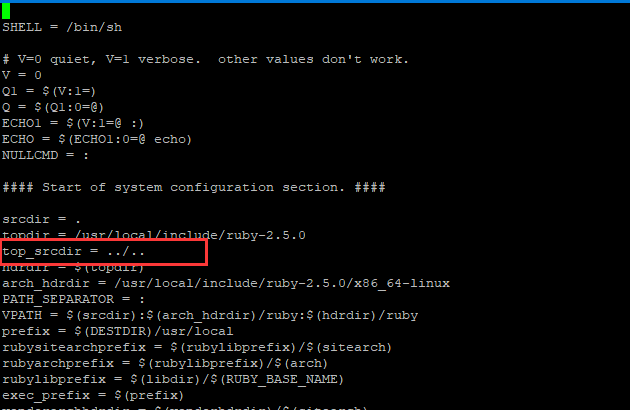
解决方案：

yum install zlib-devel

cd /root/ruby-2.5.1/ext/zlib

ruby extconf.rb

修改Makefile文件，在头部set top\_srcdir ../..



make && make install

##### openssl依赖

ERROR: While executing gem ... (Gem::Exception)

Unable to require openssl, install OpenSSL and rebuild Ruby (preferred) or use non-HTTPS sources

解决方案：

yum install openssl-devel

cd /root/ruby-2.5.1/ext/openssl

ruby extconf.rb

修改Makefile文件，在头部set top\_srcdir ../..

make && make install

### Redis-cli工具获取（5.x版本）

Redis从5.0版本开始将redis-trib.rb的功能集成到redis-cli工具中，这样就移除了对ruby的依赖。

若安装的是低版本的redis，又不想使用ruby版本的redis-trib.rb工具，这里可以考虑下一个5.x版本的redis，然后make一下，获取redis-cli编译文件，就可以直接用5.0版本的工具来操作集群了。

* wget <http://download.redis.io/releases/redis-5.0.3.tar.gz>
* tar zxvf redis-5.0.3.tar.gz
* cd redis-5.0.3
* make
* 在src目录下会生成redis-cli，可以将这个工具拷贝出来重命名redis-cli-5

直接使用这个工具即可以操作集群

## Redis配置详解

### 网络配置

##################### NETWORK #####################

**#绑定网路ip，不设置就监听所有网路接口**

**bind 127.0.0.1**

#保护模式开启，禁止外部客户端不配置密码就直接访问

protected-mode yes

**#端口号**

**port 6379**

#tcp连接监听队列大小

tcp-backlog 511

##配置监听unixsocket通讯，指定文件和权限

##unixsocket /tmp/redis.sock

##unixsocketperm 700

#server端主动关闭客户端空闲连接，设置成0表示永不关闭

timeout 0

#server端每隔一段时间（单位：秒）向客户端发送ACK请求，检测客户端是否下线，若客户端无响应则关闭连接，设置为0则不会进行检测

tcp-keepalive 300

### 通用配置

##################### GENERAL #####################

**#是否以守护进程的方式启动，**

**daemonize yes**

#操作系统层的操作

supervised no

**#若是daemonize方式会指定pidfile文件，启动时生成退出时删除**

**pidfile /var/run/redis\_6379.pid**

#日志级别debug、verbose、notice、warning

loglevel notice

**#日志路径**

**logfile "/root/Redis/redis\_log/redis-6379.log"**

##syslog服务相关用法，一般不配置

##syslog-enabled no

##syslog-ident redis

##syslog-facility local0

#设置数据库数量，默认是0号库，SELECT <dbid>指令可以选择0~databases-1号库

databases 16

#启动日志显示logo

always-show-logo yes

### rdb持久化配置

##################### SNAPSHOTTING #####################

**#执行rdb持久化策略，save <seconds> <changes>，上一次save或者bgsave后900秒更新1次、300秒更新10次、60秒更新10000次，遍历满足任意一个条件就会触发快照**

**save 900 1**

**save 300 10**

**save 60 10000**

#当rdb持久化出现异常时是否继续接收客户端写请求，这里设置成关闭仍可接收客户端写请求，当对数据持久化要求较高时建议还是开启

stop-writes-on-bgsave-error no

#设置是否对rdb文件采用LZF算法进行压缩存储，默认开启，当存储容量和网络带宽很好而CPU资源较紧张时可以关闭

rdbcompression yes

#设置是否对rdb文件采用CRC64算法来进行数据校验，默认开启，但当存储和加载rdb文件时会消耗约10%的性能，若想获取最大性能可以关闭

rdbchecksum yes

**#rdb持久化文件名**

**dbfilename dump-6379.rdb**

**#rdb文件存储路径**

**dir /root/Redis/redis\_data**

### 主从配置

##################### REPLICATION #####################

**#主从模式下使用，将当前节点设置为指定master的从节点**

**slaveof 192.168.80.128 6379**

**#若主节点配置了requirepass，这里就需要配置上对应的密码**

**masterauth 123456**

#当从节点和主节点断开连接或者正在主从复制期间，若该配置项设置为yes表示还可以为客户端的读命令服务，但是可能读不到数据或者读到错误的数据；设置成no则会对除了info和slaveof以外的所有指令直接返回"SYNC with master in progress"错误。

slave-serve-stale-data yes

**#设置为yes默认从节点只读，一旦修改会导致主从数据不一致。**

**slave-read-only yes**

#当磁盘速度较慢而网络带宽又很充足的情况下可以使用无盘复制模式，直接通过网络传输rdb文件，默认关闭

repl-diskless-sync no

#开启无盘复制模式时，设置一个等待延迟，让更多的slave到达并建立socke后开始传输rdb文件

repl-diskless-sync-delay 5

**#从节点每隔十秒会向主节点发送ping指令，心跳检测**

**repl-ping-slave-period 10**

**#设置复制过程总耗时的超时时间，首次全量复制传输的rdb文件过大时，很可能因为超时导致同步失败。需要根据数据量和带宽进行计算，配置合适的值**

**repl-timeout 60**

**#设置主从数据同步是否开启TCP\_NODELAY模式，当设置为yes时主节点会合并小的TCP包进行传输从而节省带宽，但是因为合并包和tcp的ACK延迟确认机制会有一个约40ms的数据同步延迟；当设置为no时，主节点会立即发送同步数据，带宽消耗较大一些。当跨机房部署或者主从间的网络环境复杂时建议设置成yes**

**repl-disable-tcp-nodelay no**

**#设置复制数据缓冲区大小，用于部分复制，当从节点与主节点间断开，主节点在复制时会将数据信息储存在缓冲区中，待从节点恢复连接后通过保存的offset直接进行部分复制即可，而不需要使用全量复制**

**repl-backlog-size 1mb**

#当最后一个从节点断开开始，主节点将在这个ttl时间后释放缓冲区，而从节点则不会释放

repl-backlog-ttl 3600

**#配置slave节点优先级，数字越小优先级越高，当故障转移选举slave节点的时候，可以根据优先级进行选举，设置成0则永远不会被选举为master**

**slave-priority 100**

**#设置一组条件来决定master节点是否接受外部写请求，有num个slave的连接延迟大于10秒的情况下master节点将不再接受写请求。**

**min-slaves-to-write 0**

**min-slaves-max-lag 10**

### 安全配置

##################### SECURITY #####################

**#密码**

**requirepass 123456**

#配置命令重命名或禁用，防止客户端使用一些配置指令，给指令加密或者直接设置成""禁用

#rename-command CONFIG "b840fc02d524045429941cc15f59e41cb7be6c52"

### 客户端配置

##################### CLIENTS #####################

**#设置最大客户端连接数，客户端实际最大连接数再减去32，用户redis内部句柄连接**

**maxclients 10000**

### 内存管理配置

##################### MEMORY MANAGEMENT #####################

**#最大工作内存，满了后会根据一定的策略清理内存，若不清理则直接返回错误**

**#maxmemory <bytes>**

**#配置内存策略，lru，lfu，random，ttl，noeviction(不移除)**

**maxmemory-policy noeviction**

#为lru和ttl算法提供参考样例值，默认5已经够用，10更准确但消耗cpu更多，3更快但不太准确

maxmemory-samples 5

### 惰性删除配置（4.0新特性）

##################### LAZY FREEING #####################

#4.0新特性，实现big key的惰性删除，防止删除大键数据时阻塞redis服务，会另起一个线程执行删除操作

**#内存回收时是否开启惰性删除**

**lazyfree-lazy-eviction no**

**#过期键的清理是否开启惰性删除**

**lazyfree-lazy-expire no**

#rename等命令导致的删除是否开启惰性删除

lazyfree-lazy-server-del no

#从节点全量同步flush本地数据是否开启惰性删除

slave-lazy-flush no

### AOF持久化配置

##################### APPEND ONLY MODE #####################

**#开启AOF持久化，追加命令写入缓冲区，再由缓冲区同步到磁盘**

**appendonly no**

**#文件名称，同样也会产生在上面配置的dir目录下**

**appendfilename "appendonly-6379.aof"**

**#aof文件同步策略，默认每秒同步一次，always每次时间都要同步，no由操作系统决定何时同步**

**appendfsync everysec**

#当重写aof时是否允许appendfsync同步，设置成no可能出现磁盘IO竞争而阻塞，但不会丢失数据，设置成yes将可能丢失数据（相当于上面的sppendfsync在重写期间被置为no）

no-appendfsync-on-rewrite no

#自动重写策略，超过64mb后，若当前aof大小超过base大小（每次重写后能够记录当前数据库状态的基础大小）的100%，且这个base大小大于开启重写策略的最小值64mb

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

#当aof文件加载恢复数据时最后一条日志可能不完整（断电或其他原因导致），yes表示直接忽略，no则会恢复失败后再加载

aof-load-truncated yes

#4.0新特性，支持aof和rdb混合加载模式恢复数据，先执行rdb的全量恢复，再追加aof增量日志数据

aof-use-rdb-preamble no

### Lua脚本配置

##################### LUA SCRIPTING #####################

#lua脚本最长执行时间，毫秒

lua-time-limit 5000

### 集群配置

##################### REDIS CLUSTER #####################

**#集群模式开关**

**#cluster-enabled no**

**#集群节点信息配置文件名称，这个文件不是手动配置而是由redis节点自动创建/更新的**

**#cluster-config-file nodes-6379.conf**

**#集群节点间通信（ping）超时时间，毫秒，超过则被认为可能主观下线了**

**#cluster-node-timeout 15000**

#当master失效进行failover过程时，与主节点断开过久的slave节点不会当选，计算公式(node-timeout \* slave-validity-factor) + repl-ping-slave-period，这里就是15\*10+10=160秒

#cluster-slave-validity-factor 10

#当一个master下有大于该值个slave节点值，才会尝试迁移slave给孤立的master

#cluster-migration-barrier 1

#默认yes时，所有的slots必须被覆盖集群才是正常状态，任何一个slot没有分配整个集群都无法使用

#cluster-require-full-coverage yes

**#当master节点失败时不允许slave执行failover完成故障转移，但是可以在master节点手动转移。默认不打开，当master和slave是不同数据中心时可能禁止另一方接管数据写入时打开为yes**

**#cluster-slave-no-failover no**

##################### CLUSTER DOCKER/NAT support #####################

#默认情况下，Redis会自动检测自己的IP和从配置中获取绑定的PORT，告诉客户端或者是其他节点。而在Docker环境中，如果使用的不是host网络模式，在容器内部的IP和PORT都是隔离的，那么客户端和其他节点无法通过节点公布的IP和PORT建立连接。如果配置了以后，Redis节点会将配置中的这些IP和PORT告知客户端或其他节点。而这些IP和PORT是通过Docker转发到容器内的临时IP和PORT的

# cluster-announce-ip 10.1.1.5

# cluster-announce-port 6379

# cluster-announce-bus-port 6380

### 监控配置

##################### SLOW LOG #####################

**#记录执行时间大于10000微秒的查询命令**

**slowlog-log-slower-than 10000**

**#因为会消耗内存所以可以限定日志条数，可以通过 SLOWLOG RESET 来回收内存**

**slowlog-max-len 128**

##################### LATENCY MONITOR #####################

#redis默认情况下，延迟监视是禁用的，因为如果不存在延迟问题，则通常不需要延迟监视，而且收集数据会对性能产生影响，虽然这种影响很小，但可以在大负载下进行测量。对超过多少毫秒的操作进行监控，并通过LATENCY命令打印图表和报告信息

latency-monitor-threshold 0

##################### EVENT NOTIFICATION #####################

#数据库通知事件：1、键空间通知-某个键执行了什么命令2、键时间通知-某个命令被什么键执行了，默认关闭

notify-keyspace-events ""

### 高级配置

##################### ADVANCED CONFIG #####################

**hash-max-ziplist-entries 512**

**hash-max-ziplist-value 64**

**list-max-ziplist-size -2**

**list-compress-depth 0**

**set-max-intset-entries 512**

**zset-max-ziplist-entries 128**

**zset-max-ziplist-value 64**

**hll-sparse-max-bytes 3000**

**activerehashing yes**

**#客户端缓冲区大小，硬限制（超过512m则直接关闭连接），软限制（连续60s超过128m），设成0关闭保护，normal、slave、pubsub表示客户端类型：普通客户端，从节点或monitor，pubsub**

**client-output-buffer-limit normal 0 0 0**

**client-output-buffer-limit slave 256mb 64mb 60**

**client-output-buffer-limit pubsub 32mb 8mb 60**

# client-query-buffer-limit 1gb

# proto-max-bulk-len 512mb

#redis定时任务执行频率(关闭客户端连接，删除过期key等)，默认值是10，过大会消耗cpu

hz 10

#rewriteAOF文件时，默认yes，文件每生成32MB数据便进行fsync同步。增量地将文件写入磁盘避免较大的延迟高峰。

aof-rewrite-incremental-fsync yes

# lfu-log-factor 10

# lfu-decay-time 1

# Enabled active defragmentation

# activedefrag yes

# Minimum amount of fragmentation waste to start active defrag

# active-defrag-ignore-bytes 100mb

# Minimum percentage of fragmentation to start active defrag

# active-defrag-threshold-lower 10

# Maximum percentage of fragmentation at which we use maximum effort

# active-defrag-threshold-upper 100

# Minimal effort for defrag in CPU percentage

# active-defrag-cycle-min 25

# Maximal effort for defrag in CPU percentage

# active-defrag-cycle-max 75

## Setinel配置详解

#sentinel使用的端口

**port 26379**

#sentinel后台运行。这行配置是添加的

**daemonize yes**

#log文件地址，这行配置是添加的

**logfile "/root/Redis/redis\_log/sentinel-xxxx.log"**

#指定master。后面的数字表示，当有几个节点认为主节点down时才认为主节点进入ODOWN状态，就是真正挂了。

**sentinel monitor mymaster 172.16.203.10 6379 1**

#当多久，连接不上节点时，认为被连接节点进入S\_DOWN（主观认为它down了），单位：毫秒；

**sentinel down-after-milliseconds mymaster 5000**

#这个配置有很多作用。1、重新执行failover的时间是该值的2倍；2、取消一个没更改配置的failover3、failover中等待所有slave更改新的配置的最大时间，单位：毫秒。

**sentinel failover-timeout mymaster 15000**

#设置校验的密码。如果redis设置了密码，这个一定要设置

**sentinel auth-pass mymaster 123456**

#通知型脚本:当sentinel有任何警告级别的事件发生时（比如说redis实例的主观失效和客观失效等等），将会去调用这个脚本，这时这个脚本应该通过邮件，SMS等方式去通知系统管理员关于系统不正常运行的信息。调用该脚本时，将传给脚本两个参数，一个是事件的类型，一个是事件的描述。如果sentinel.conf配置文件中配置了这个脚本路径，那么必须保证这个脚本存在于这个路径，并且是可执行的，否则sentinel无法正常启动成功。

**#sentinel notification-script mymaster /var/redis/notify.sh**

##sentinel当故障转移切换master成功后触发执行此脚本时，会默认传递几个参数过来

##<master-name> <role> <state> <from-ip> <from-port> <to-ip> <to-port>

##通过这些事件参数，可以做一些脚本的处理，如更新VIP下的redis实例地址更新

**#sentinel client-reconfig-script mymaster /var/redis/reconfig.sh**

## Redis常用命令

### 动态配置命令

CONFIG SET parameter value动态设置参数，如缓冲区大小、最大内存限制、maxclients等。

CONFIG GET \* 可以查看所有支持动态修改的参数。

### 键命令

#### 过期时间设置

EXPIRE key seconds 设置多久后过期，单位s

PEXPIRE key milliseconds 设置多久后过期，单位s

EXPIREAT key timestamp 设置过期时间点，timestamp

PEXPIREAT key milliseconds-timestamp

#### 过期时间查看

TTL key 查看key还剩多久过期

### SHUTDOWN命令，停止当前redis实例。

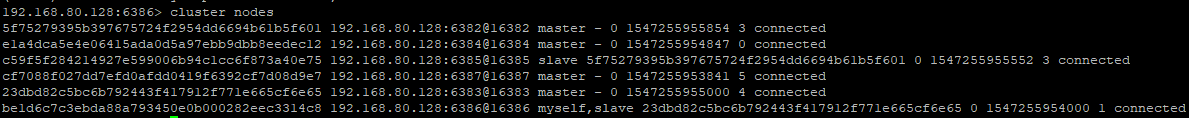
SHUTDOWN

### Cluster常用命令

#### ClUSTER NODES

查看集群中所有节点信息，格式如下：

**<id> <ip:port> <flags> <master> <ping-sent> <pong-recv> <config-epoch> <link-state> <slot> <slot> ... <slot>**

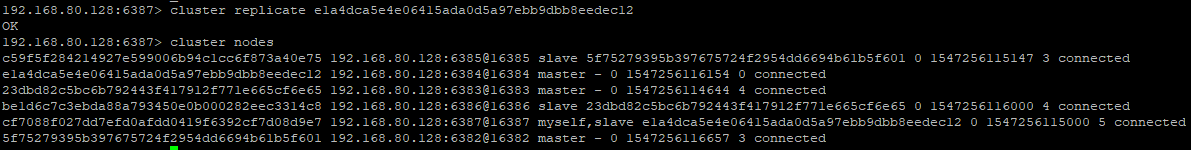


#### CLUSTER MEET ip port

在一个已经启动的节点上执行meet命令，将指定的新实例加入集群；

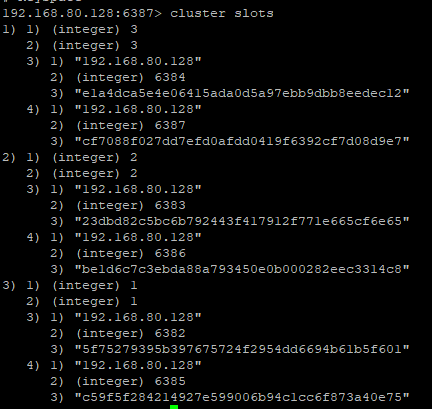
#### CLUSTER REPLICATE node-id

将当前节点设置为指定nodeid节点的slave；



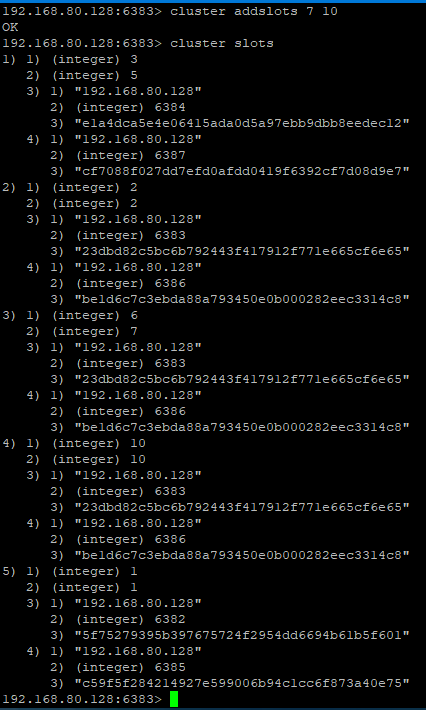
#### CLUSTER SLOTS

显示集群的slot分布情况；



#### CLUSTER ADDSLOTS slot [slot ...]

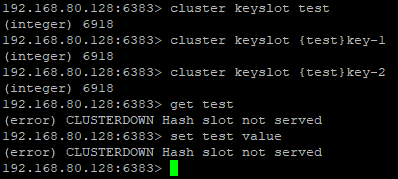
给当前连接的节点增加slot，取值范围0-16383；



这里注意一下，显示是按slot值区间来的，比如分配了6，7,10三个slot，这里会分开展示6-7,10-10两个区间对应的ip、port和node-id。

#### ClUSTER KEYSLOT key

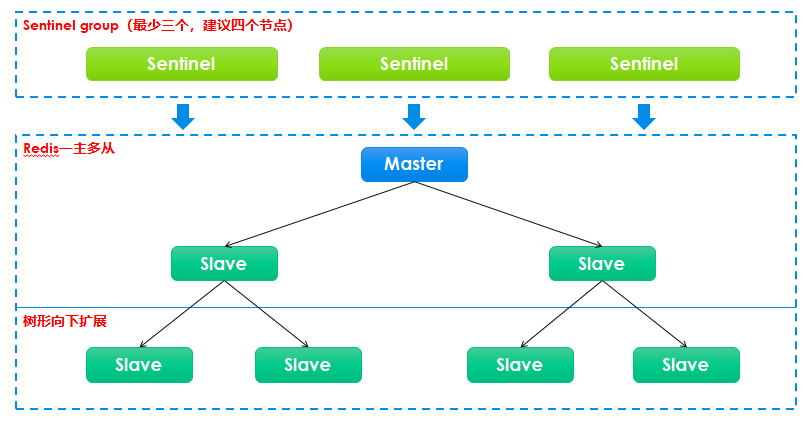
用来测试key会被分配到哪个slot，若key对应的slot未分配，则无法操作可以，返回如下：



## 官方高可用方案——Redis Sentinel

### Sentinel HA特性

#### 拓扑结构



#### 使用场景

至少需要三个节点来部署Sentinel集群，Redis实例采用一主多从方式部署。

**扩容方案：**

* 通过设置实例的maxmemory进行垂直扩容。
* 水平扩容——根据系统对读性能的要求增加slave节点，但是单机master写瓶颈无法解决。

**读写方案：**

* 主节点可读写，从节点仅读；

**负载方案：**

* 主节点负责写，从节点负责读，可以进行读负载，在读节点上加一层nginx或者HAProxy，客户端创建读连接池，指向代理redis地址端口即可；

**所以Redis Sentinel主要是redis的高可用实现，适用于读操作远大于写操作，并且单机足够支撑应用所有的写操作的场景。**

#### 故障转移

* 发现主服务器已经进入客观下线状态。
* 对我们的当前纪元进行自增（详情请参考 Raft leader election）， 并尝试在这个纪元中当选。
* 如果当选失败， 那么在设定的故障迁移超时时间的两倍之后， 重新尝试当选。 如果当选成功， 那么执行以下步骤。
* 选出一个从服务器，并将它升级为主服务器。
* 向被选中的从服务器发送 SLAVEOF NO ONE 命令，让它转变为主服务器。
* 通过发布与订阅功能， 将更新后的配置传播给所有其他 Sentinel ， 其他 Sentinel 对它们自己的配置进行更新。
* 向已下线主服务器的从服务器发送 [SLAVEOF](http://www.redis.cn/commands/slaveof.html) 命令， 让它们去复制新的主服务器。
* 当所有从服务器都已经开始复制新的主服务器时， 领头 Sentinel 终止这次故障迁移操作

**参见附录一 主从故障切换场景测试**

### 部署配置

#### 准备工作

--root

--Redis 总目录

--redis\_{version} redis安装目录

--redis\_conf 配置文件目录

--redis\_log 日志文件目录

--redis\_data 数据文件目录

* **Redis安装目录有redis.conf以及sentinel.conf模板文件**，通常直接拷贝出来根据需要修改即可，**注意尽量不要直接修改模板文件**，以便后续部署需要；
* 在redis\_conf目录中复制三个哨兵配置sentinel-xxxx.conf，参考**2.3.4.4章节sentinel.conf配置修改**；
* 在redis\_conf目录中复制三个redis-xxxx.conf，参考**2.2 Redis配置详解**以及**2.3.4.2最小化配置**进行修改根据实际需要进行修改，一个master两个slave，修改各自实例的端口号，若是从节点，则配置slaveof为主节点的ip和port；
* **这里注意日志文件、rdb文件、aof文件、配置文件等名称必须带上端口号，因为一台机器上可能部署多个redis实例。**

#### 主节点redis.conf最小化配置修改

这里注意几个配置项：

daemonize yes ——设置redis后台运行；

port 6379 ——设置端口；

logfile “/root/Redis/redis\_log/redis-xxxx.log” ——配置日志文件位置；

requirepass 123#qwe ——配置redis连接密码；

dbfilename dump-6379.rdb ——rdb持久化文件名

dir “/root/Redis/redis\_data” ——工作目录保存rdb、aof、node文件等；

#### 从节点redis.conf最小化配置修改

其他配置参考master

slaveof 192.168.18.131 6379 配置其master

masterauth 123#qwe ——配置redis作为Master时，其他Slave的连接密码；

#### sentinel.conf配置修改

#sentinel使用的端口

port 26379

#sentinel后台运行。这行配置是添加的

daemonize yes

#log文件地址，这行配置是添加的

logfile "/root/Redis/redis\_log/sentinel-xxxx.log"

#指定master。后面的数字表示，当有几个节点认为主节点down时才认为主节点进入ODOWN状态，就是真正挂了。

sentinel monitor mymaster 172.16.203.10 6379 1

#当多久，连接不上节点时，认为被连接节点进入S\_DOWN（主观认为它down了）；

sentinel down-after-milliseconds mymaster 5000

#这个配置有很多作用。1、重新执行failover的时间是该值的2倍；2、取消一个没更改配置的failover3、failover中等待所有slave更改新的配置的最大时间。

sentinel failover-timeout mymaster 15000

#设置校验的密码。如果redis设置了密码，这个一定要设置

sentinel auth-pass mymaster 123456

这里有个小坑，sentinel monitor mymaster 172.16.203.10 6379 1这一项必须在其他几个mymaster的前面，动态生成配置文件时候顺序可能会有问题，报错

\*\*\* FATAL CONFIG FILE ERROR \*\*\*

Reading the configuration file, at line 1

>>> 'sentinel failover-timeout mymaster 15000'

No such master with specified name.

### Sentinel模式启动

为了在任何地方直接执行redis-cli、redis-server、redis-sentinel等操作

cd src

cp redis-cli redis-server redis-sentinel /usr/bin/

#### 启动sentinel哨兵

redis-sentinel sentinel-26379.conf

redis-sentinel sentinel-26380.conf

redis-sentinel sentinel-26380.conf

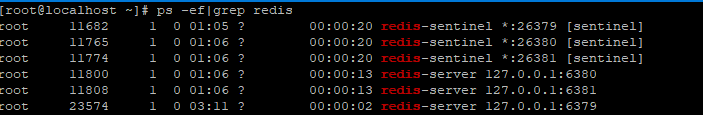
#### 启动redis实例

redis-server redis-6379.conf

redis-server redis-6380.conf

redis-server redis-6381.conf

这里是以单机开启6个端口为例，实际看机器情况而定。启动成功如下所示：



### Redis Sentinel主从模式运维

#### Redis-cli使用

Redis-cli –h 127.0.0.1 –p 6379 –a 123456连接客户端工具；

127.0.0.1:6379>这里可以直接执行redis命令，如info、keys \*、flushdb等等

#### 动态扩容

##### 垂直扩容

CONFIG SET maxmemory 2gb，所有主节点和从节点都需要设置一下。

这里可以动态修改redis相关配置，直接生效。

##### 水平扩容/收缩

当读压力增大时，可以增加slave节点以提升读性能。不支持master节点扩容，存在单机写瓶颈。

新部署一个redis节点，注意需要设置masterauth；

配置中可以直接指向slaveof 主节点，也可以通过客户端命令直接执行slaveof ip host命令；

会自动进行数据同步；

若收回多余的从节点，redis-cli执行slaveof no one即可；

#### 节点下线

客户端连接后使用Shutdown命令



当然也可以直接杀进程。

**主节点下线：**会引发failover故障转移。

**从节点下线：**sentinel会监控到+sdown主观下线状态，但是不会进行任何操作。

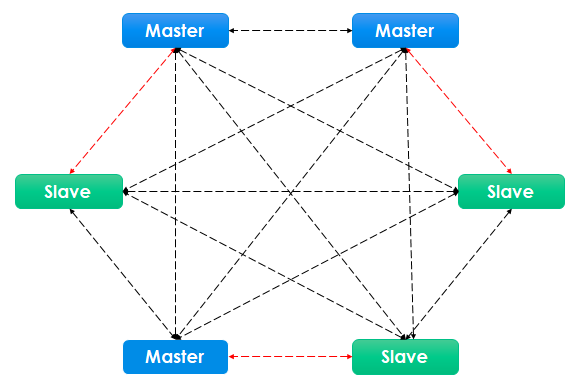
#### AOF迁移

redis-cli -h 192.168.80.128 -p 6582 -a 123456 --pipe < appendonly-6579.aof

## 官方集群方案——RedisCluster

### 集群特性

#### 无中心结构



#### 基本概念

**普通Redis TCP端口：**用于redis和客户端间的交互；

**集群总线端口：**节点间的失败检测、配置更新、故障转移授权等，通常在普通端口的基础上+10000 ；

注意：两种类型的网络权限都需要开通，保证客户端和redis，以及redis集群节点间的ip端口权限都已开通。

**CLUSTERMSG事件类型**

#define CLUSTERMSG\_TYPE\_PING 0 /\* Ping \*/心跳检测

#define CLUSTERMSG\_TYPE\_PONG 1 /\* Pong (reply to Ping) \*/心跳响应

#define CLUSTERMSG\_TYPE\_MEET 2 /\* Meet “let‘s join” message \*/加入集群

#define CLUSTERMSG\_TYPE\_FAIL 3 /\* Mark node xxx as failing \*/标记接待为Fail状态

#define CLUSTERMSG\_TYPE\_FAILOVER\_AUTH\_REQUEST 5 /\* May I failover? \*/请求接管failover

#define CLUSTERMSG\_TYPE\_FAILOVER\_AUTH\_ACK 6 /\* Yes, you have my vote \*/得到master投票

**节点数**

Master不能少于3个，高可用至少为每个master准备一个slave，所以通常一个cluster最少6个节点

**节点状态**

**PFAIL ：**发现节点超过nodetimeout未响应则置为PFAIL状态；

**FAIL：**超过半数的主节点标记目标节点为PFAIL状态，将目标节点由PFAIL状态置为FAIL状态；

**GOSSIP协议**

无中心化结构，每秒10次定时任务clusterCron()>>clusterProcessGossipSection()处理各节点的状态

#### 一致性

不能保证强一致性：

**场景一**

Your client writes to the master B.

The master B replies OK to your client.

The master B propagates the write to its slaves B1, B2 and B3

因为这里使用的是异步复制，如上描述，在B还没完成写入他的slave时宕掉了，任意一个没有接收到写入的slave被提升为主节点，那么这部分写入的数据就丢失了。

Redis集群在绝对需要时支持同步写入，通过WAIT命令实现，这使得丢失写操作的可能性大大降低，但是请注意，即使使用同步复制，Redis集群也不会实现很强的一致性：在更复杂的故障场景中，无法接收写入的从节点被选为主服务器，这是可能的。

**场景二**

还有一种典型场景是网络分区导致的一致性问题，当客户端和少数节点（至少包含一个主节点）的网络分区被隔离时：

Take as an example our 6 nodes cluster composed of A, B, C, A1, B1, C1, with 3 masters and 3 slaves. There is also a client, that we will call Z1.After a partition occurs, it is possible that in one side of the partition we have A, C, A1, B1, C1, and in the other side we have B and Z1.

Z1还会继续向B中写数据，如果这个网络分区的时间较短，集群正常。如果B1被选举为主节点，那么Z1向B中写的数据将会丢失。

这里会有一个很重要的配置node timeout，当主节点在这个时间后还无法感知到其他大多数主节点，那么他将进入错误状态并且不再接收写操作。

#### 数据分片

Redis Cluster的分片不使用一致性hash，而是一种不同的分片策略，每个键在概念上都是我们所称的hash slot（散列槽）的一部分。一共有16384个槽位，Redis Cluster通常对key使用CRC16算法后模除16384来计算一个key对应的槽位。A节点0~5460，B节点5461~10922，C节点10923~16383

当向集群中添加一个新的节点D时，我们需要从ABC中迁移一些hash slot给D，如右图（执行了一次rebalance操作）。同样的当删除集群中的节点A时，需要将A的hash slot迁移给B和C。并且hash slot的迁移并不需要停机，仍然可以支持增加或删除节点。

另外，用户还可以通过hash tag的方式强制多个键成为同一个哈希槽的一部分，如：this{foo}key和another{foo}key会被分配到同一个hash slot中。

#### 故障转移

**心跳检测——needed\_quorum = (server.cluster->size / 2) + 1;**

节点间通过ping、pong进行心跳检测，当A节点ping节点B超过NODE\_TIMEOUT仍无响应时，会将其标记为PFAIL状态（主从节点都可以操作）。A节点会通过gossip消息收集到集群中其他节点标记B的状态信息，当大部分主节点都标记B为PFAIL状态时，A会把B标记为FAIL状态，并向所有可达节点发送给一个FAIL消息强制所有节点将B标记FAIL状态。

**从节点选举(FAIL\_OVER\_AUTH) ——needed\_quorum = (server.cluster->size / 2) + 1;**

从节点通过广播一个 FAILOVER\_AUTH\_REQUEST 数据包给集群里的每个主节点来请求选票。然后等待回复（最多等 NODE\_TIMEOUT 这么长时间）。一旦一个主节点给这个从节点投票，会回复一个 FAILOVER\_AUTH\_ACK，并且在 NODE\_TIMEOUT \* 2 这段时间内不能再给同个主节点的其他从节点投票，一旦某个从节点收到了大多数主节点的回应，那么它就赢得了选举。

**故障转移(FAIL\_OVER)**

clusterFailoverReplaceYourMaster(){

1) Turn this node into a master.

2) Claim all the slots assigned to our master.

3) Update state and save config

4) Pong all the other nodes so that they can update the state accordingly and detect that we.

5) If there was a manual failover in progress, clear the state.

}

### Cluster配置

#######################REDIS CLUSTER#######################

#集群模式开关

**cluster-enabled yes**

#集群节点信息配置文件名称，这个文件不是手动配置而是由redis节点自动创建/更新的

**cluster-config-file nodes-6379.conf**

#集群节点间通信（ping）超时时间，毫秒，超过则被认为可能主观下线了

**cluster-node-timeout 15000**

#当master失效进行failover过程时，与主节点断开过久的slave节点不会当选，计算公式(node-timeout \* slave-validity-factor) + repl-ping-slave-period，这里就是15\*10+10=160秒

**cluster-slave-validity-factor 10**

#当一个master下有大于该值个slave节点值，才会尝试迁移slave给孤立的master

**cluster-migration-barrier 1**

#默认yes时，所有的slots必须被覆盖集群才是正常状态，任何一个slot没有分配整个集群都无法使用

**cluster-require-full-coverage no**

#当master节点失败时不允许slave执行failover完成故障转移，但是可以在master节点手动转移。默认不打开，当master和slave是不同数据中心时可能禁止另一方接管数据写入时打开为yes

**cluster-slave-no-failover no**

### Cluster启动

#### 分别启动所有实例

* 在/root/redis\_conf/cluster目录下复制6份redis.conf模板，分别为redis-xxxx.conf；
* 参照**2.2.4.3章节（主节点redis.conf最小化配置修改）**，修改主节点基础配置；
* 这里注意一下，启动节点时都是主节点，先不考虑分配主从角色；
* 需要预先设置**masterauth** password ——配置redis作为Master时，其他Slave的连接密码；
* 参照**2.3.6章节（Cluster配置）**，打开并修改集群配置；
* 依次启动6个实例redis-server redis-xxxx.conf；
* redis-cli –h ip –p port连接任意一个实例，执行CLUSTER MEET ip:port，依次将指定实例加入集群；
* 此时6个实例全部是master，通过redis-cli –h ip –p port选择一个希望设置为slave的实例， 执行CLUSTER REPLICATE node-id，node-id为集群中节点的唯一标识id，可以通过CLUSTER NODES进行查看，依次设置好三个从节点；
* 接下来为master节点分配slots，必须在master实例里执行CLUSTER ADDSLOTS 1 2 3 4…为节点分配slots，只有当所有的slots被分配完成cluster状态才是ok

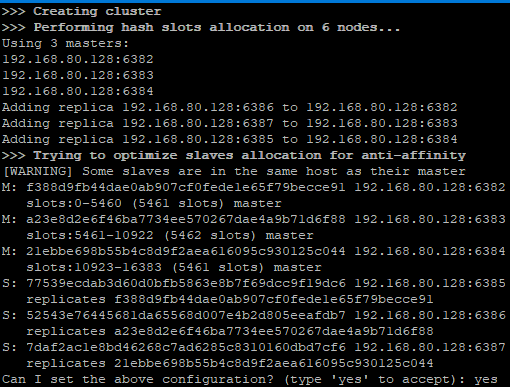
**至此，一个纯手动的redis-cluster启动成功，但是这种方式十分麻烦，而且CLUSTER SLOT相关的一些指令操作时可能出现bug，推荐还是使用已经开发好的cluster工具更方便，例如redis-trib.rb。**

#### redis-trib.rb执行启动(At least 6 nodes are required.)

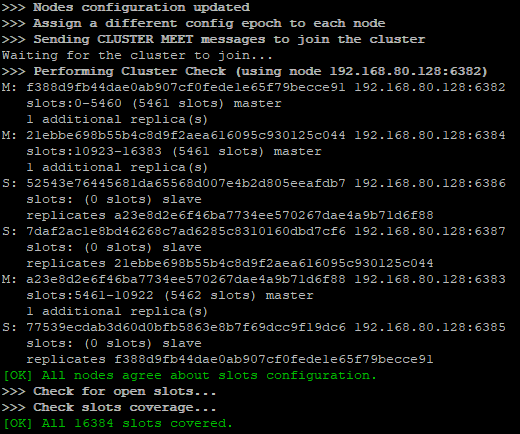
当启动好6个redis实例后，通过工具执行

redis-trib.rb create --replicas 1 192.168.80.128:6382 192.168.80.128:6383 192.168.80.128:6384 192.168.80.128:6385 192.168.80.128:6386 192.168.80.128:6387

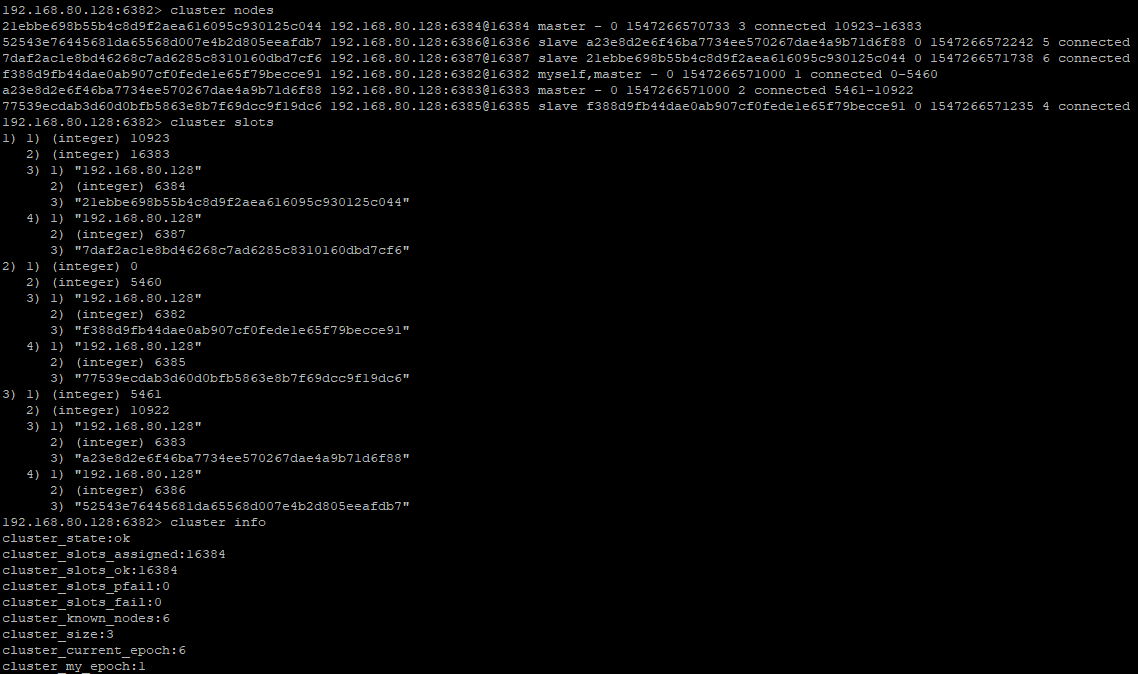
默认最少要6个节点， 并且会自动设置成3主3从，每个master节点一个slave，并且会给master自动平均分配好slot区间0-5460、5461-10922、10922-16383。



手动确认yes后，完成集群配置并且自动为master分配slots



查看集群、节点、slot信息，全部成功，如下所示



### Cluster运维

#### redis-trib.rb用法

参考文章：

<http://weizijun.cn/2016/01/08/redis%20cluster%E7%AE%A1%E7%90%86%E5%B7%A5%E5%85%B7redis-trib-rb%E8%AF%A6%E8%A7%A3/>

命令包括原理讲解的很详细。

* 主要使用的redis-trib.rb reshard来完成数据迁移。
* redis-3.0.6版本以前migrate操作是单个key逐一操作。从redis-3.0.6开始，支持单次迁移多个key。
* redis集群内部最多只允许一个slot处于迁移状态，不能并发的迁移slots。
* redis-trib.rb reshard如果执行中断，用redis-trib.rb fix修复集群状态。
* redis-trib.rb支持resharding/rebalance，分配权重。
* redis-trib.rb支持从单个redis迁移数据到cluster集群中。

#### redis-cli（5.0版本）

##### 连接集群

redis-cli -h 192.168.80.128 -p 6479 -a 111111，任意连接集群中的一个节点，然后使用cluster 相关命令即可，如CLUSTER NODES查看节点信息，CLUSTER SLOTS查看hash槽分配情况等等。

##### 集群创建

redis-cli --cluster create 127.0.0.1:6379 127.0.0.1:6380 127.0.0.1:6381 127.0.0.1:6382 127.0.0.1:6383 127.0.0.1:6384 --cluster-replicas 1

##### 自动分配slot

redis-cli --cluster --rebalance ip host –a 111111

##### 集群恢复

redis-cli --cluster –fix ip host –a 111111

##### 操作节点

###### 新增主节点

redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000

第一个ip:port对应要新增的节点，第二个ip:port可以是任意一个集群内已经存在的节点。新增主节点需要分配slot，可以执行--reblance命令自动分配

###### 新增从节点

redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000 --cluster-slave

redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000 --cluster-slave --cluster-master-id 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e

第一条没有指定新增的从节点对应主节点id，则会自动分配到集群中副本较少的master节点下面，第二条则直接成为指定id的master节点的从节点。

###### 删除节点

redis-cli --cluster del-node 127.0.0.1:7000 `<node-id>`

从节点可以直接删除，主节点必须保证为空后才可以执行删除命令，所以要先将主节点的重新分片数据迁移到其他的主节点上。

移除主节点的另一种方法是对其一个从节点执行手动故障转移，并在节点变为新主节点的从节点后移除该节点。显然，当您想要减少集群中的主服务器的实际数量时，这是没有帮助的，在这种情况下，需要Resharding。

## 其他集群方案（Proxy分片）

### Twitter开源的Twemproxy

Github地址：https://github.com/twitter/twemproxy

参考文章：https://www.wengbi.com/thread\_93142\_1.html

这里知乎在实际运用中提到了两种策略，一种当做存储（HA的要求会高一些），一种当做缓存（直接动态剔除节点，不做主从复制），也会使用自研的工具来实现平滑的扩容和单核模型的性能瓶颈问题。

单核模型造成性能瓶颈；

传统扩容模式仅支持停机扩容；

无运维控制台；

Twemproxy+Sentinel+Redis部署方案：<https://my.oschina.net/dchuang/blog/666827>

#### 安装

yum install autoconf

yum install automake

yum install libtool

wget https://github.com/twitter/twemproxy/archive/master.zip

unzip master.zip

cd twemproxy-master

## 在twemproxy源码目录执行autoreconf 生成 configure文件等

aclocal

autoreconf -f -i -Wall,no-obsolete

## 然后编译安装

./configure --prefix=/usr/local/twemproxy/

make && make install

wget -O m4-1.4.9.tar.gz http://ftp.gnu.org/gnu/m4/m4-1.4.9.tar.gz

tar -zvxf m4-1.4.9.tar.gz

cd m4-1.4.9

./configure

make

make install

wget http://www.cpan.org/src/5.0/perl-5.16.1.tar.gz

tar -xzf perl-5.16.1.tar.gz

cd perl-5.16.1

./Configure -des -Dprefix=/usr/local/perl

make

make test

make install

perl -v

#### 配置模板

redis\_15454:

listen: 192.168.80.129:15454

hash: fnv1a\_64

distribution: ketama

auto\_eject\_hosts: true

redis: true

server\_retry\_timeout: 2000

server\_failure\_limit: 1

servers:

- 192.168.80.128:6579:1

- 192.168.80.130:6579:1

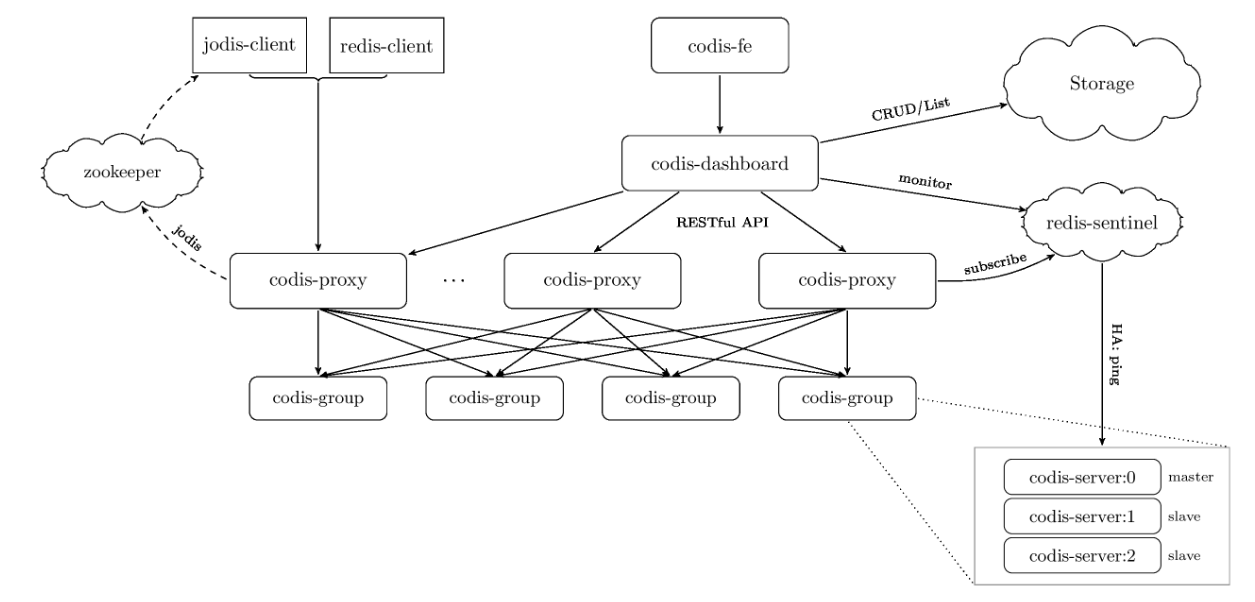
redis\_auth: 123456

#### 启动命令

/usr/local/twemproxy/sbin/nutcracker -c /usr/local/twemproxy/conf/redis\_15454.yml -p /usr/local/twemproxy/conf/redis\_15454.pid -o /usr/local/twemproxy/logs/redis\_15454.log -v 11 -d

### 豌豆荚开源的Codis

Github地址：https://github.com/CodisLabs/codis



优点：

* 自带控制台；
* 支持多键操作，不停机平滑扩容，新版本resharding支持异步数据迁移；
* 支持Pipeline

缺点：

* 通过代理降低性能；
* 一些redis操作不支持；
* 不负责主从复制，不支持跨机房架构
* 部署较复杂

## Redis数据迁移

**github链接：**[**https://github.com/vipshop/redis-migrate-tool**](https://github.com/vipshop/redis-migrate-tool)

* 快速。
* 多线程。
* 基于redis复制。
* 实时迁移。
* 迁移过程中，源集群不影响对外提供服务。
* 异构迁移。
* 支持Twemproxy集群，redis cluster集群，rdb文件 和 aof文件。
* 过滤功能。
* 当目标集群是Twemproxy，数据会跳过Twemproxy直接导入到后端的redis。
* 迁移状态显示。
* 完善的数据抽样校验。

迁移工具的来源可以是：单独的redis实例，twemproxy集群，redis cluster，rdb文件，aof文件。

迁移工具的目标可以是：单独的redis实例，twemproxy集群，redis cluster，rdb文件。

软件编译安装：

$ cd redis-migrate-tool

$ autoreconf -fvi

$ ./configure

$ make

$ src/redis-migrate-tool -h

软件运行：

src/redis-migrate-tool -c rmt.conf -o log -d

配置文件示例：从redis cluster集群迁移数据到twemproxy集群

[source]

type: redis cluster

servers:

- 127.0.0.1:6379

[target]

type: twemproxy

hash: fnv1a\_64

hash\_tag: "{}"

distribution: ketama

servers:

- 127.0.0.1:6380:1 server1

- 127.0.0.1:6381:1 server2

- 127.0.0.1:6382:1 server3

- 127.0.0.1:6383:1 server4

[common]

listen: 0.0.0.0:34345

threads: 8

step: 1

mbuf\_size: 512

source\_safe: true

配置文件示例：从redis cluster集群迁移数据到另外一个cluster集群

[source]

type: redis cluster

servers:

- 127.0.0.1:8379

[target]

type: redis cluster

servers:

- 127.0.0.1:7379

[common]

listen: 0.0.0.0:8888

配置文件示例：从rdb文件恢复数据到redis cluster集群

[source]

type: rdb file

servers:

- /data/redis/dump1.rdb

- /data/redis/dump2.rdb

- /data/redis/dump3.rdb

[target]

type: redis cluster

servers:

- 127.0.0.1:7379

[common]

listen: 0.0.0.0:8888

状态查看：通过redis-cli连接redis-migrate-tool监控的端口，运行info命令

$redis-cli -h 127.0.0.1 -p 8888

127.0.0.1:8888> info

# Server

version:0.1.0

os:Linux 2.6.32-573.12.1.el6.x86\_64 x86\_64

multiplexing\_api:epoll

gcc\_version:4.4.7

process\_id:9199

tcp\_port:8888

uptime\_in\_seconds:1662

uptime\_in\_days:0

config\_file:/ect/rmt.conf

# Clients

connected\_clients:1

max\_clients\_limit:100

total\_connections\_received:3

# Memory

mem\_allocator:jemalloc-4.0.4

# Group

source\_nodes\_count:32

target\_nodes\_count:48

# Stats

all\_rdb\_received:1

all\_rdb\_parsed:1

rdb\_received\_count:32

rdb\_parsed\_count:32

total\_msgs\_recv:7753587

total\_msgs\_sent:7753587

total\_net\_input\_bytes:234636318

total\_net\_output\_bytes:255384129

total\_net\_input\_bytes\_human:223.77M

total\_net\_output\_bytes\_human:243.55M

total\_mbufs\_inqueue:0

total\_msgs\_outqueue:0

127.0.0.1:8888>

数据校验：

$src/redis-migrate-tool -c rmt.conf -o log -C redis\_check

Check job is running...

Checked keys: 1000

Inconsistent value keys: 0

Inconsistent expire keys : 0

Other check error keys: 0

Checked OK keys: 1000

All keys checked OK!

Check job finished, used 1.041s

# 监控指标

## 单节点监控

**INFO命令（client、memory、 cpu、 persistence、stats、replication、cluster、key）**

### # Server服务端信息

redis\_version:4.0.10

redis\_git\_sha1:00000000

redis\_git\_dirty:0

redis\_build\_id:5e919e5ad2fcc2cb

redis\_mode:standalone

os:Linux 3.10.0-862.el7.x86\_64 x86\_64

arch\_bits:64

multiplexing\_api:epoll

atomicvar\_api:atomic-builtin

gcc\_version:4.8.5

process\_id:43623

run\_id:2a378d6e1d518a357fc65162a6b5a3101077d1d9

tcp\_port:6579

uptime\_in\_seconds:335111

uptime\_in\_days:3

hz:10

lru\_clock:7332957

executable:/root/Redis/redis-4.0.10/src/redis-server

config\_file:/root/Redis/redis\_conf/redis-sentinel-6579.conf

### # Clients客户端信息

connected\_clients:7

client\_longest\_output\_list:0

client\_biggest\_input\_buf:0

blocked\_clients:0

### # Memory内存信息

used\_memory:46227848

used\_memory\_human:44.09M

used\_memory\_rss:55529472

used\_memory\_rss\_human:52.96M

used\_memory\_peak:86136480

used\_memory\_peak\_human:82.15M内存峰值

used\_memory\_peak\_perc:53.67%

used\_memory\_overhead:22906704

used\_memory\_startup:786704

used\_memory\_dataset:23321144

used\_memory\_dataset\_perc:51.32%

total\_system\_memory:1919254528

total\_system\_memory\_human:1.79G

used\_memory\_lua:37888

used\_memory\_lua\_human:37.00K

maxmemory:134217728

maxmemory\_human:128.00M 最大内存限制

maxmemory\_policy:noeviction

mem\_fragmentation\_ratio:1.20

mem\_allocator:jemalloc-4.0.3

active\_defrag\_running:0

lazyfree\_pending\_objects:0

### # Persistence

loading:0

rdb\_changes\_since\_last\_save:0

rdb\_bgsave\_in\_progress:0

rdb\_last\_save\_time:1550831347

rdb\_last\_bgsave\_status:ok

rdb\_last\_bgsave\_time\_sec:1

rdb\_current\_bgsave\_time\_sec:-1

rdb\_last\_cow\_size:8810496

aof\_enabled:0

aof\_rewrite\_in\_progress:0

aof\_rewrite\_scheduled:0

aof\_last\_rewrite\_time\_sec:-1

aof\_current\_rewrite\_time\_sec:-1

aof\_last\_bgrewrite\_status:ok

aof\_last\_write\_status:ok

aof\_last\_cow\_size:0

### # Stats状态统计

total\_connections\_received:207

total\_commands\_processed:3303916

instantaneous\_ops\_per\_sec:9

total\_net\_input\_bytes:247263116

total\_net\_output\_bytes:957317776

instantaneous\_input\_kbps:0.51

instantaneous\_output\_kbps:8.24

rejected\_connections:0

sync\_full:5

sync\_partial\_ok:1

sync\_partial\_err:3

expired\_keys:0

expired\_stale\_perc:0.00

expired\_time\_cap\_reached\_count:0

evicted\_keys:0

keyspace\_hits:0 命中次数

keyspace\_misses:0 丢失次数

pubsub\_channels:1

pubsub\_patterns:0

latest\_fork\_usec:1005 上次fork耗时，微秒，如果不能接受阻塞时间过长，限制容量

migrate\_cached\_sockets:0

slave\_expires\_tracked\_keys:0

active\_defrag\_hits:0

active\_defrag\_misses:0

active\_defrag\_key\_hits:0

active\_defrag\_key\_misses:0

### # Replication主从信息

role:master

connected\_slaves:5

slave0:ip=192.168.80.128,port=6580,state=online,offset=99640689,lag=1

slave1:ip=192.168.80.128,port=6581,state=online,offset=99640832,lag=1

slave2:ip=192.168.80.128,port=6582,state=online,offset=99640832,lag=1

slave3:ip=192.168.80.128,port=6583,state=online,offset=99640689,lag=1

slave4:ip=192.168.80.128,port=6379,state=online,offset=99640832,lag=1

master\_replid:88caea015d49ce5b73bb9abb7162b43aa974d454

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:99640832

second\_repl\_offset:-1

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:98592257

repl\_backlog\_histlen:1048576

### # CPU信息

used\_cpu\_sys:358.83

used\_cpu\_user:61.47

used\_cpu\_sys\_children:1.05

used\_cpu\_user\_children:1.28

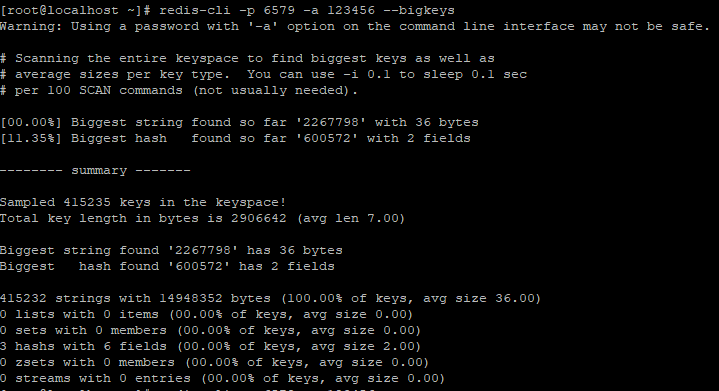
### # Cluster

cluster\_enabled:0

### # Keyspace键空间信息

db0:keys=415232,expires=0,avg\_ttl=0

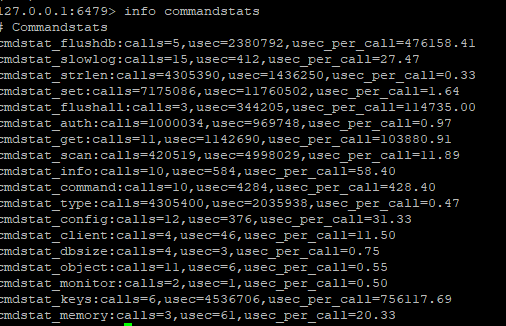
## BigKey查询



可以统计出每种数据结构类型key的数量，占用内存最大的key以及平均大小

## 命令统计

**INFO commandstats统计命令调用次数、耗费CPU时间、每个命令平均耗费CPU**



命令统计可以通过定时任务，定时采集，比如一分钟采集一次，可以计算出每分钟命令调用次数，cpu时间等。从而可以定制化一些曲线图。

## 慢查询分析

### 慢查询配置

#记录执行时间大于10000微秒的查询命令  
**slowlog-log-slower-than 10000**  
#因为会消耗内存所以可以限定日志条数，可以通过 **SLOWLOG RESET** 来回收内存  
**slowlog-max-len 128**

### 测试方案

**向缓存中插入100w条数据：key为1~1000000，value为一个uuid**

执行keys \*命令大概需要2974ms

**向缓存中插入一条大key数据：key为testBigKey，value为1|2|3|….|10000000**

执行GET “testBigKey”命令耗时2177ms ，cpu增至20%~30%

执行MEMORY USAGE “testBigKey”命令查看该键值占用内存为79445399

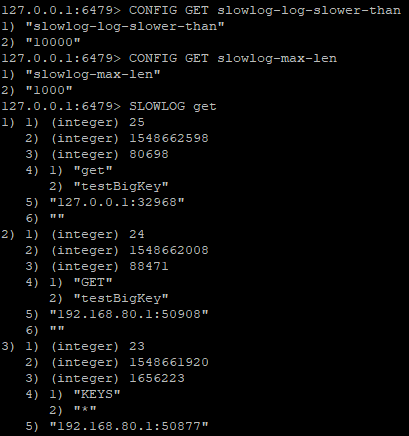
**向缓存中插入一条大key数据：key为testBigKey，value为1|2|3|….|30000000**

执行GET “testBigKey”命令耗时7206ms，cpu增至40%~50%

执行MEMORY USAGE “testBigKey”命令查看该键值占用内存为259850479

### 查询结果

然后可以通过SLOWLOG命令查看执行较慢的日志，这里根据上面的配置，最多查看128条执行时间超过10ms的指令，超过128条会展示最新的，底层是一个FIFO队列



# 客户端

## Jedis

轻量级的客户端，不支持读写分离、异步api等高级特性，非线程安全。

## Lettuce

### 高级特性

Spring Boot 2.0中 Redis 客户端驱动现在由 Jedis变为了 Lettuce。 Lettuce 的确有很多优秀的特性，例如：

* 基于 netty，支持事件模型
* 支持 同步、异步、响应式 的方式
* 可以方便的连接 Redis Sentinel
* 完全支持 Redis Cluster
* SSL 连接
* Streaming API
* CDI 和 Spring 的集成
* 兼容 Java 8 和 9

## Tedis

Tedis的目标是打造一个可在生产环境直接使用的高可用Redis解决方案

### Feature

* 高可用，Tedis使用多写随机读做HA确保redis的高可用
* 高性能，使用特殊的线程模型，使redis的性能不限制在客户端
* 多种使用方式，如果你只有一个redis实例，并不需要tedis的HA功能，可以直接使用tedis-atomic；使用tedis的高可用功能需要部署多个redis实例使用tedis-group
* 两种API，包括针对byte的底层api和面向object的高层api
* 多种方便使用redis的工具集合，包括mysql数据同步到redis工具，利用redis做搜索工具等

## 客户端需要解决的问题

* **读写分离**
* **故障切换过程**

合理配置客户端的超时时间和重试策略，因为在主从故障期间的写入可能会失败

* **拓扑结构更新**

哨兵模式：通过订阅sentinel事件来更新

集群模式：通过cluster nodes命令来更新，定时更新或者是发生MOVE重定向时更新

* **脑裂问题**

哨兵模式：当拓扑更新或者执行命令时发现集群中有两个主节点，直接抛异常重试或者更新为最新的主节点（需要查询节点的current-epoch或者config-epoch来确认）

* **监控/异常信息上报**

定义一些需要监控的客户端异常，如move/ask次数过多、连接超时等等，如何上报跟see打通

* **多写方案**

跨机房多活的场景下，需要提供多写随机读的功能，参考淘宝开源的Tedis

# 附录（测试场景模拟）

## 附录一 Sentinel主从故障切换场景测试

Redis部署：127.0.0.1 6379主节点 6380 6381从节点

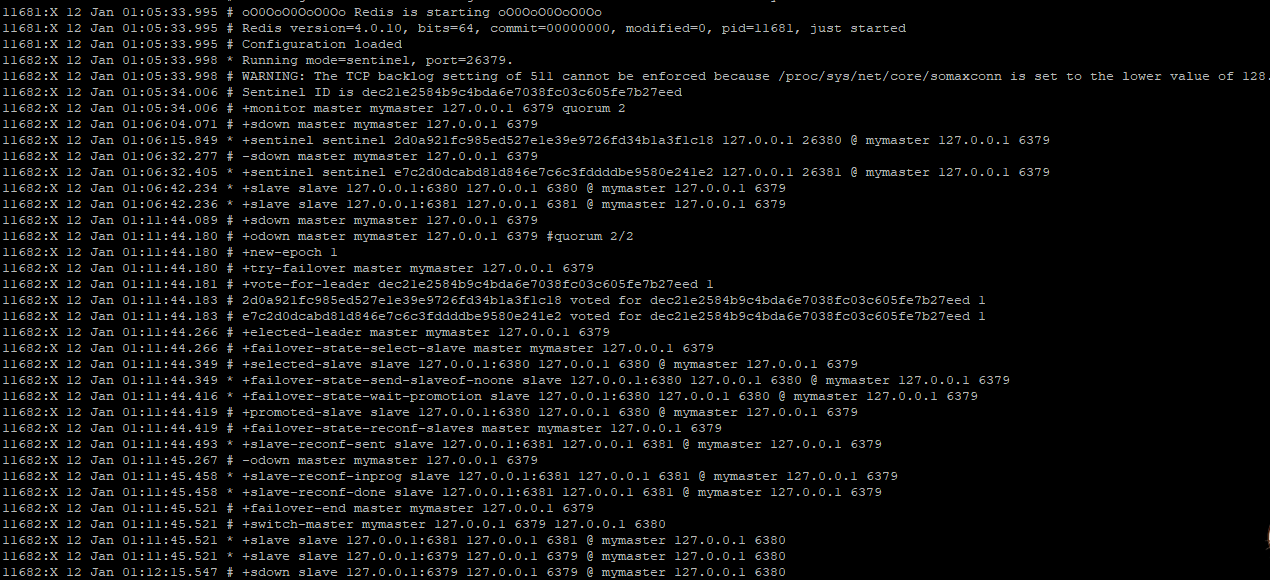
Sentinel部署：127.0.0.1端口分别为26379、26380、26381

手动kill掉6379的redis主节点进程，sentinel会选举出新的主节点为6380，并且修改redis和sentinel的配置文件，将主节点信息全部指向端口号为6380的实例。

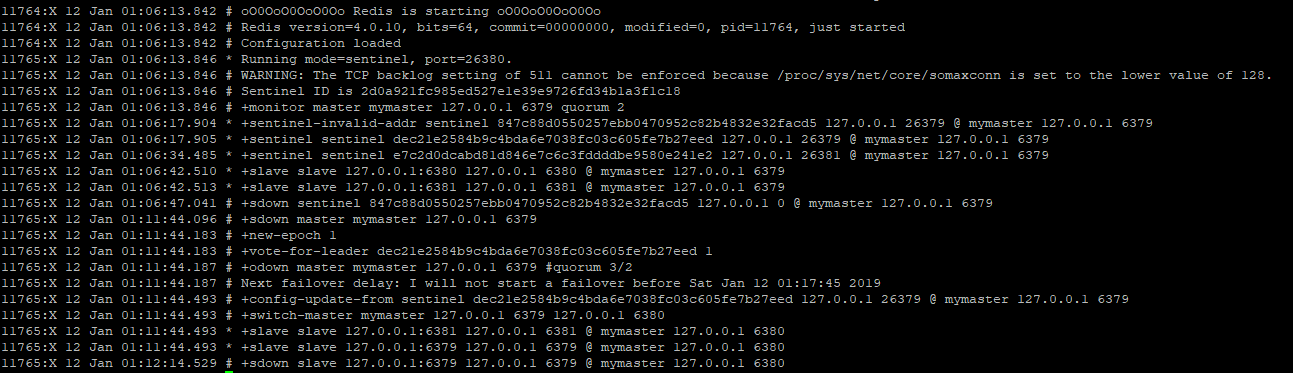
整个过程可以很清楚的在sentinel日志中体现，仔细分析一下sentinel的日志

* 启动监听master
* 自动发现其他sentinel和slave，+sentinel +slave …
* 发现master节点主观下线，+sdown master …
* 确认master节点客观下线(超过quorum且半数以上sentinel确认)，+odown master … #quorum 2/2
* 新增一个配置版本，尝试接管故障转移过程，发起投票，new epoch… try failover…vote-for-leader
* 当选本次故障转移elected-leader
* 选举新的slave节点来晋升主节点，failover-state-select-slave … selected-slave …
* 提升从节点，failover-state-send-slaveof-noone … failover-state-wait-promotion … promoted-slave
* 重新配置其他从节点，failover-state-reconf-slaves … slave-reconf-sent … slave-reconf-inprog … slave-reconf-done …
* 故障转移过程结束failover-end
* 其他非leader sentinel需要更新配置，config-update-from sentinel leader\_id …
* 切换sentinel监听的主节点，switch-master
* 自动发现新的从节点

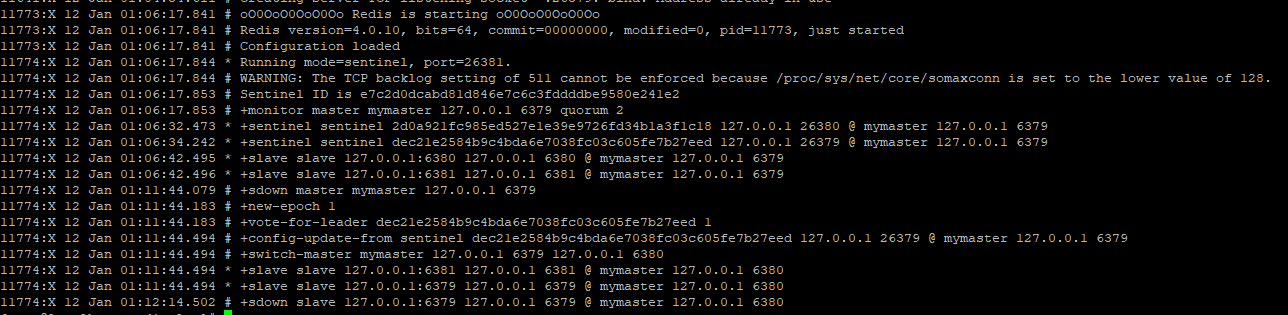
具体的日志，如下三图所示，sentinel-26379.log



sentinel-26380.log



sentinel-26381.log



这里一次故障切换过程已经完成，结尾会有一个+sdown slave 6379。

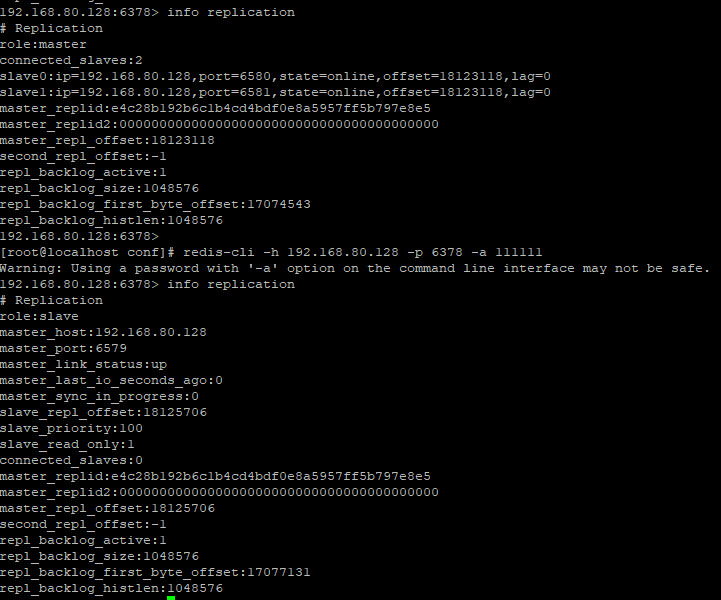
这个时候再重启原先的6379实例，会发现：-sdown slave 6379。

用info或者CONFIG GET slaveof查看发现，6379已经从节点的身份重新加入高可用方案中来，他的master指向新的主节点127.0.0.1 6379



并且在redis-6379.conf的配置文件中也会自动修改





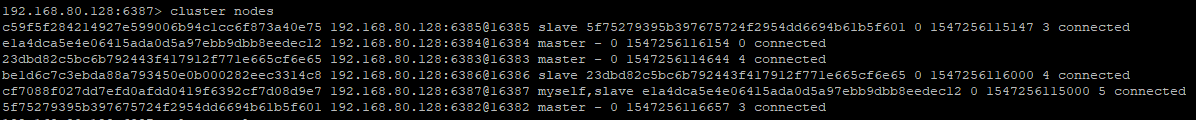
## 附录二 RedisCluster故障转移场景测试

初始部署情况

192.168.80.128:6382、192.168.80.128:6383、192.168.80.128:6384三台主节点

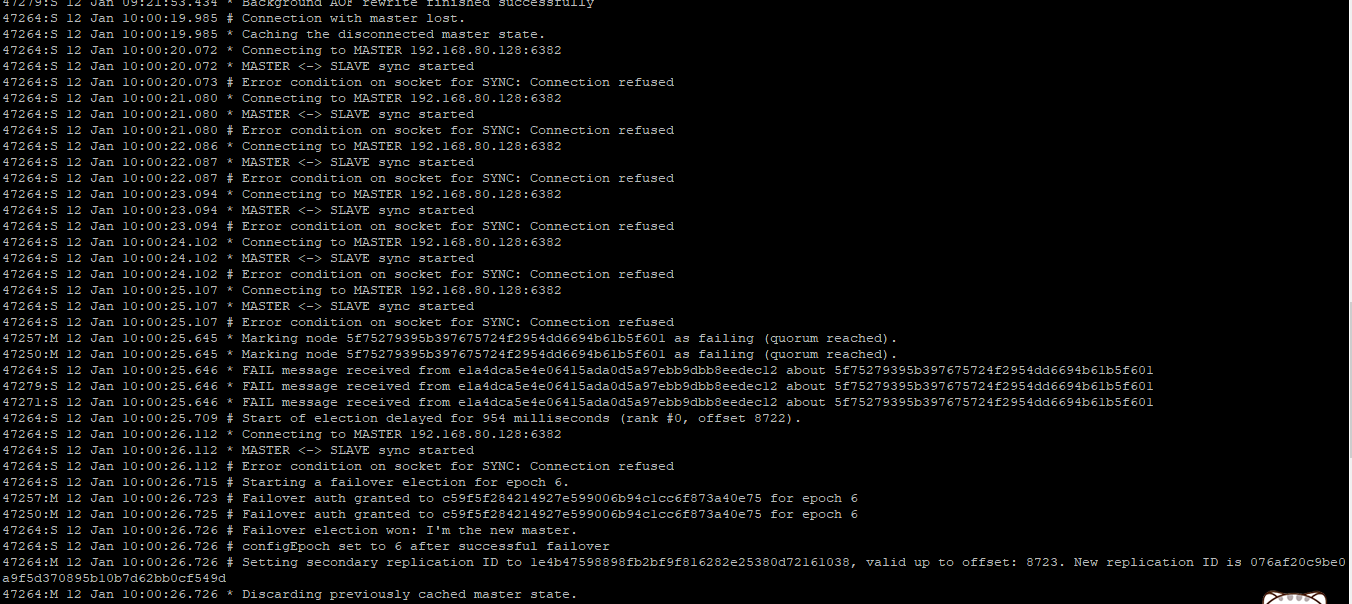
192.168.80.128:6385、192.168.80.128:6386、192.168.80.128:6387三台从节点

CLUSTER NODES指令查看集群所有节点情况。

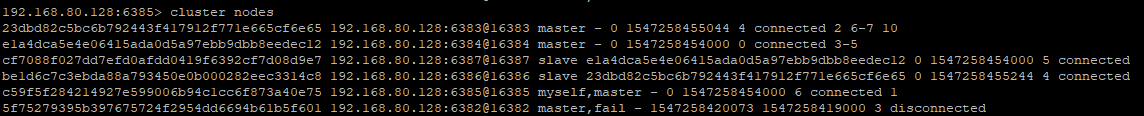


打开查看集群日志日志（目前单机所有端口的logfile都写进一个日志文件中，所以会有多条重复的消息），将6382端口的redis主服务进程kill掉，可以从日志中看到整个故障转移的过程：

* 发现6382端口master连接失败；
* 另外两个主节点将6382标记为失败；
* 向集群所有可达节点发送FAIL消息确认6382端口实例下线；
* 开始选举新的master；
* 选举成功，6385当选新的master节点；



再次查看集群节点状态，发现6382端口的master服务状态为fail，6385端口的slave服务已经被提升为master。

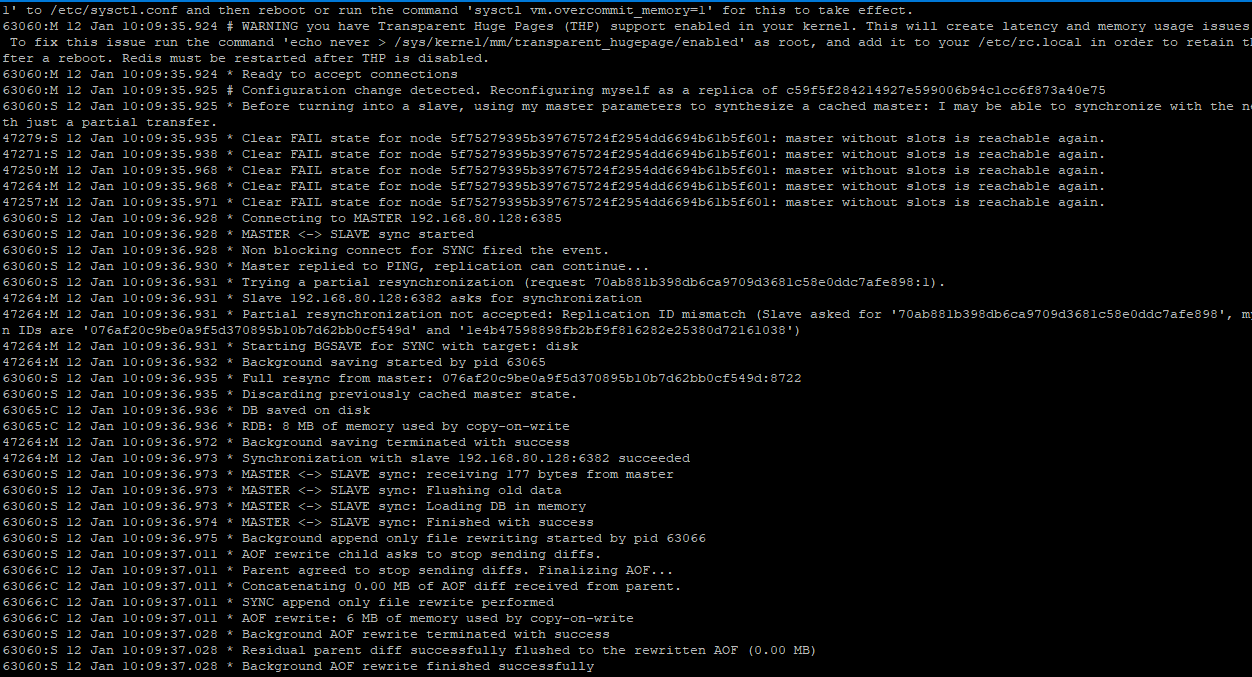


此时再次启动6382端口的redis服务，从日志中可以查看启动恢复的过程：

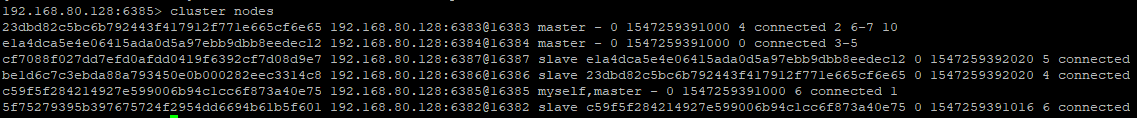
* 清除FAIL标记；
* 连接6385新的master；
* 开始主从同步；

这里还可以看到一个主从同步流程的完整日志：

* master响应slave的ping；
* 执行bgsave生成rdb文件；
* slave收到master发送的rdb文件；
* flush旧数据；
* 加载rdb文件，完成主从同步；



再次查看cluster nodes集群节点状态发现6382端口成功被切换为slave节点

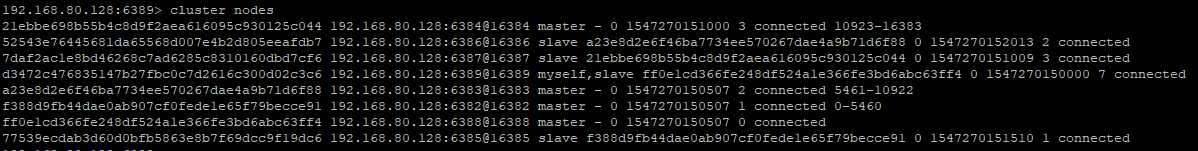


## 附录三 集群扩容新增节点&重新分片场景测试

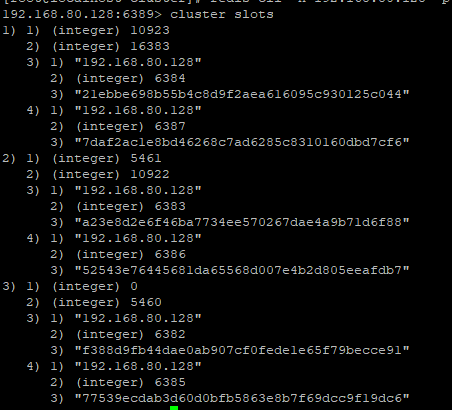
cluster meet 192.168.80.128 6388

cluster meet 192.168.80.128 6389

192.168.80.128:6389> cluster replicate ff0e1cd366fe248df524a1e366fe3bd6abc63ff4

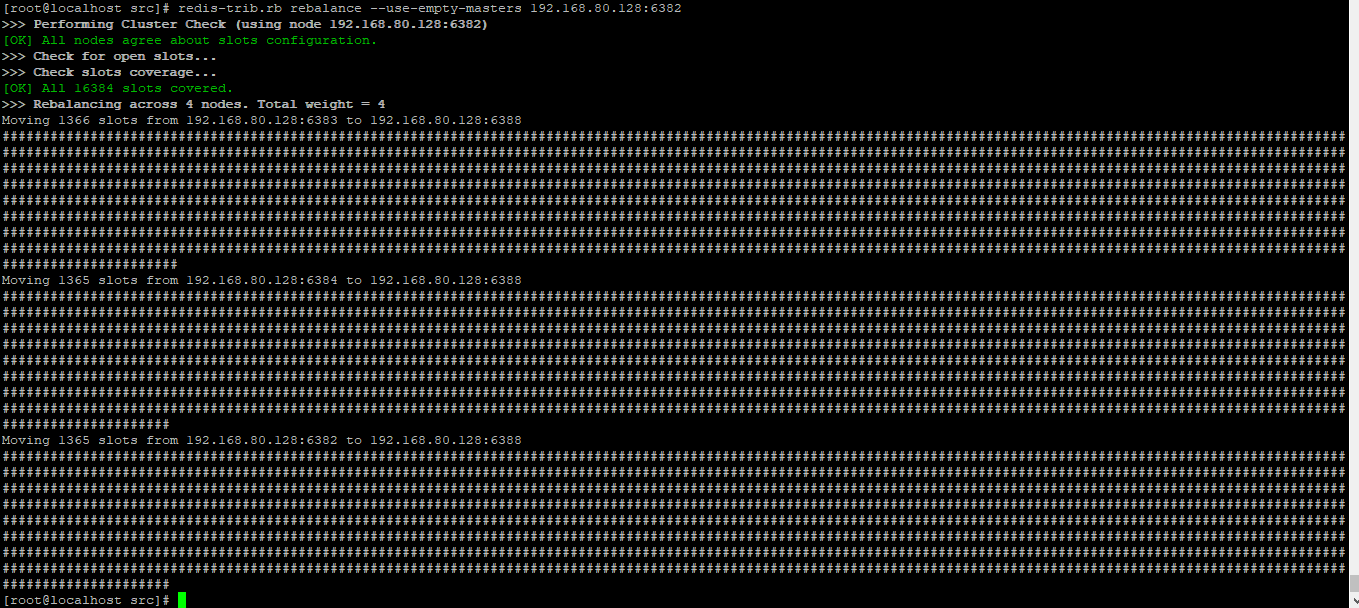


此时集群已经变成4主4从，新加入的节点6388和6389已经meet到集群，但是还没有分配slots。

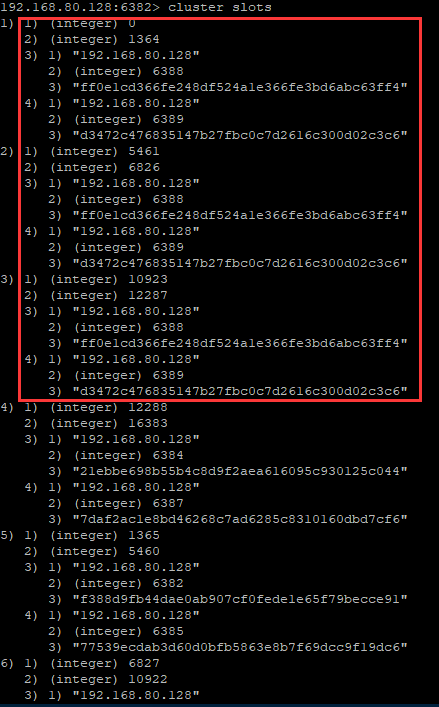


执行工具rebalance重新分配slot，过程如下：

redis-trib.rb rebalance –use-empty-masters 192.168.80.128:6382



可以看到分别从原先的三个master分别迁移过来1365或1366个slots，此时在查看集群节点的slots分布信息，如下图：



可以看到slots已经rebalance成，因为这里没有配置权重和指定迁移的节点，所以默认的算法是从所有的master节点平均分配slots，新的节点占1/(1+3)=1/4。