## 安装

下载redis安装文件 http://redis.io/ redis-4.0.2.tar.gz

tar -zvxf redis-4.0.2.tar.gz

mv redis-4.0.2 /usr/local/redis

cd /usr/local/redis

make

make install

## 启动

mkdir config

cp redis.conf ./config

cd config

cat redis.conf | grep –v “#” | grep –v “^$” > redis-6379.conf

vi redis-6379.conf

daemonize yes

port 6379

dir '/usr/local/redis/data'

logfile '6379.log'

pidfile '6379.pid'

cd ..

mkdir data

./src/redis-server ./confg/redis-6379.conf

启动以后，在data目录下会有6739.log和6379.pid。

## 3.Redis单线程的优点和缺点

优点：

1. 纯内存操作
2. 多路复用IO
3. 不用加锁，不会有死锁
4. 不存在进程切换，竞态消耗
5. 实际上还是有后台线程的，比如：写硬盘、复制等等

缺点：

1. 执行时间长的命令会导致严重阻塞
2. 无法发挥多核的优势，可以开启多个实例

## 4.Redis的发布订阅

发布者、通道、订阅者，群发模式，所有的订阅者都能收到。

publish suhu:tv “helloworld” #返回订阅者数量

subscribe suhu:tv

publish可以立即返回，但是subscribe会一直在等待，直到有消息过来，继续等待新的消息。

Unsubscribe suhu:tv 取消订阅

## 5.Redis持久化

RDB：快照，全量

AOF：binlog，增量

### 5.1 RDB

触发生成rdb的3种方式：

1. 手动调用save：阻塞，会替换老的文件，
2. 手动调用bgsave：非阻塞，fork子进程来做复制，fork会阻塞但是很快，fork还会消耗额外的内存
3. 自动，当达到某些条件以后，自动触发，一般不推荐开启。

默认配置：

save 900 1

save 300 10

save 60 10000

1. 配置参数：

dbfilename dump-6379.rdb#配合dir来使用

stop-writes-on-bgsave-error yes #bgsave出错就不复制了

rdbcompression yes #开启压缩

rdbchecksum yes #开启完整性检查

1. 主从复制的时候，会全量复制主的rdb文件
2. shutdown save，也会生成rdb
3. rdb的缺点：耗时耗性能，不可控丢失数据

### 5.2 AOF

（1）AOF写日志同步到硬盘的3种方式：

always：每条命令到会同步硬盘，一条数据也不丢，效率低

everysec：一秒同步一次硬盘，会丢1秒的数据

no：由操作系统来决定何时同步到硬盘。

（2）AOF重写

会合并执行的语句，比如：set aa 123； set aa 234；set aa 456，可以优化成set aa 456

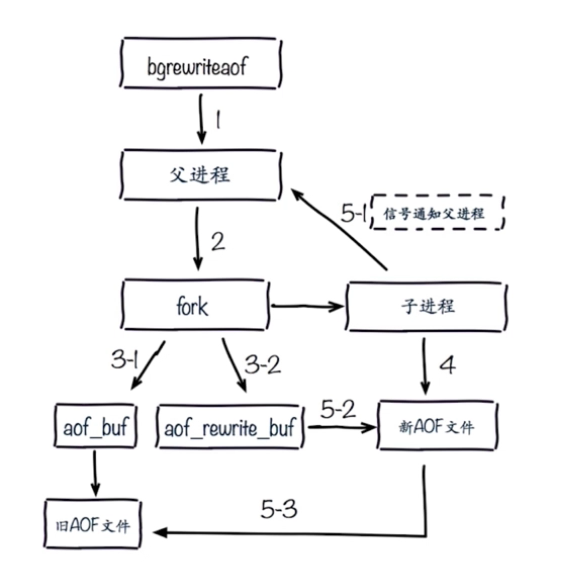
两种实现方式：（1）直接调用bgrewriteaof （2）自动配置

1. bgreweiteaof：fork子进程重写
2. 自动配置：

auto-aof-rewrite-min-size：aof需要重写的最小尺寸

auto-aof-rewrite-precentage:aof文件的增长率

有两个系统参数：aof\_currrent\_size和aof\_base\_size



1. 其他配置参数：

appendonly yes #开启aof

appendfilename “append-6379.aof” #配合dir

appendfsync everysec #每秒刷盘

no-appendfsync-on-rewrite yes#在aof重写的时候不同步硬盘，aof重写会比较耗性能和磁盘，fsync也会耗性能和磁盘。

auto-aof-rewrite-percentage 100 #增长率100%做aof重写

auto-aof-rewrite-min-size 64mb #达到64M才做aof重写

aof-load-truncated yes #是否加载不完整的aof，比如掉电

## 5.3 AOF与RDB

（1）如果同时有aof和rdb，会优先使用aof来恢复数据

（2）rdb的文件比aof小，因此rdb恢复数据快

（3）rdb会丢失数据，aof数据安全性要高得多

（4）adb很耗时耗性能，aof相对要轻量级，不考虑aof重写只是写日志。

建议：

1. 关掉自动生成rdb
2. 可以按天按周对rdb做备份
3. 主从环境下，可以在从节点开启rdb
4. 开启aof
5. aof每秒刷盘
6. 小分片，每个redis服务器设置最大内存，不要太大，防止fork、复制太慢
7. Info：latest\_fork\_usec 上次fork消耗的微妙数

aof\_delayed\_fsync:100 aof重写阻塞的次数

## 6.Redis Replication

slaveof 27.0.0.1 6389：变成6389的从，异步，从上之前的数据会被删除

slaveof no one：不再是从，但是原有的数据不删除，异步

slave-read-only true:从节点只读

sed s/6380/6381/g redis-6380.conf > redis-6381.conf

#### 6380.conf:

bind 0.0.0.0

port 6380

daemonize yes

dir /usr/local/redis/data

pidfile redis\_6380.pid

logfile redis\_6380.log

stop-writes-on-bgsave-error yes

rdbcompression yes

rdbchecksum yes

dbfilename dump\_6380.rdb

appendonly yes

appendfilename "appendonly\_6380.aof"

appendfsync everysec

no-appendfsync-on-rewrite yes

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

aof-load-truncated yes

aof-rewrite-incremental-fsync yes

#### 6381.conf

bind 0.0.0.0

port 6381

daemonize yes

dir /usr/local/redis/data

pidfile redis\_6381.pid

logfile redis\_6381.log

slaveof 127.0.0.1 6380

stop-writes-on-bgsave-error yes

rdbcompression yes

rdbchecksum yes

dbfilename dump\_6381.rdb

appendonly yes

appendfilename "appendonly\_6381.aof"

appendfsync everysec

no-appendfsync-on-rewrite yes

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

aof-load-truncated yes

aof-rewrite-incremental-fsync yes

分别启动，6381就会成为6380的slave。

#### 6380上：

127.0.0.1:6380> info Replication

# Replication

role:master

connected\_slaves:1

slave0:ip=127.0.0.1,port=6381,state=online,offset=140,lag=1

master\_replid:849c198416bf26201110deb596b0c2d3affcaf70

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:140

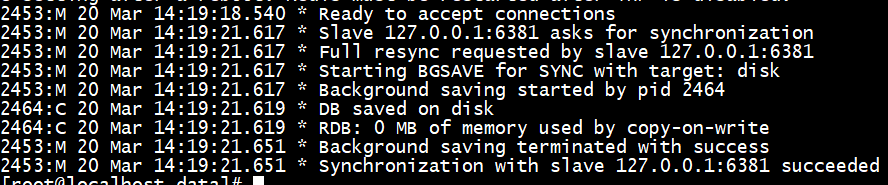
second\_repl\_offset:-1

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:1

repl\_backlog\_histlen:140



#### 6381上：

127.0.0.1:6381> info Replication

# Replication

role:slave

master\_host:127.0.0.1

master\_port:6380

master\_link\_status:up #状态

master\_last\_io\_seconds\_ago:10

master\_sync\_in\_progress:0

slave\_repl\_offset:210

slave\_priority:100

slave\_read\_only:1

connected\_slaves:0

master\_replid:849c198416bf26201110deb596b0c2d3affcaf70

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:210

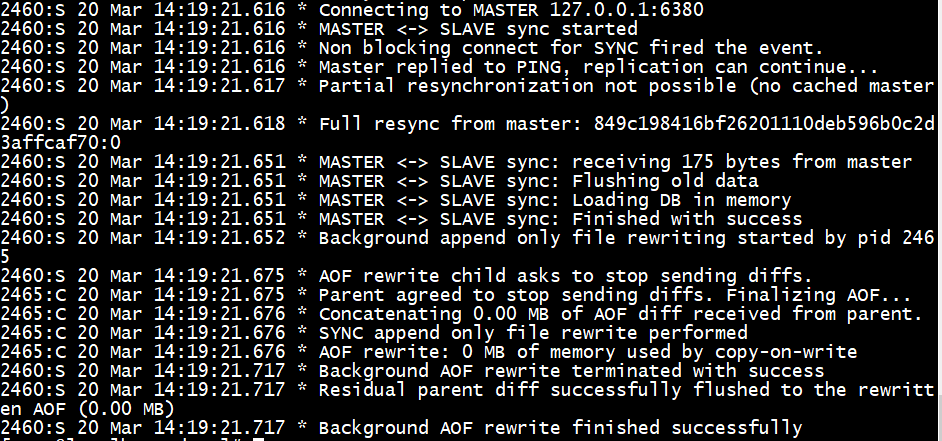
second\_repl\_offset:-1

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:1

repl\_backlog\_histlen:210



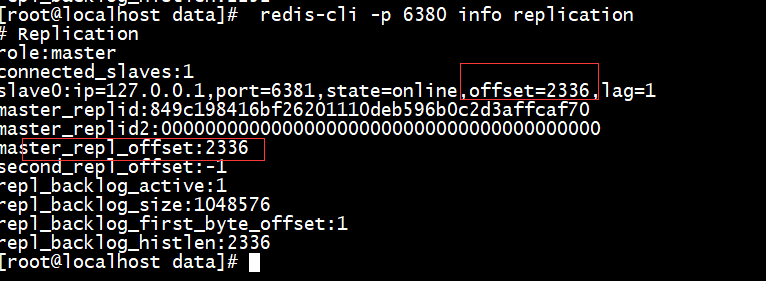
#### run id

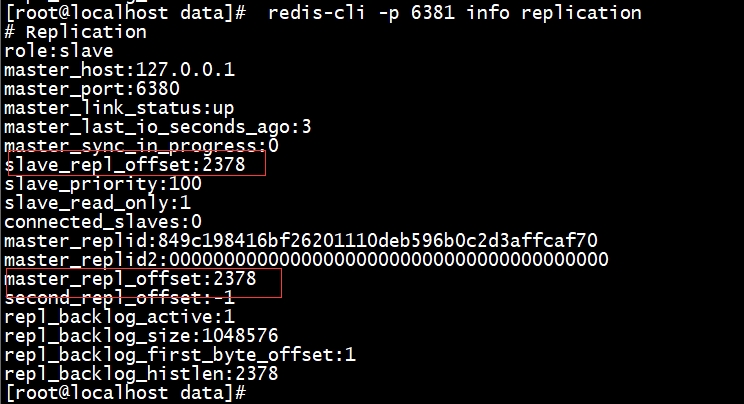
[root@localhost data]# redis-cli -p 6380 info server | grep run\_id

run\_id:4d96e9ca7a6fb55062672fc8548c0a8fea89bc42

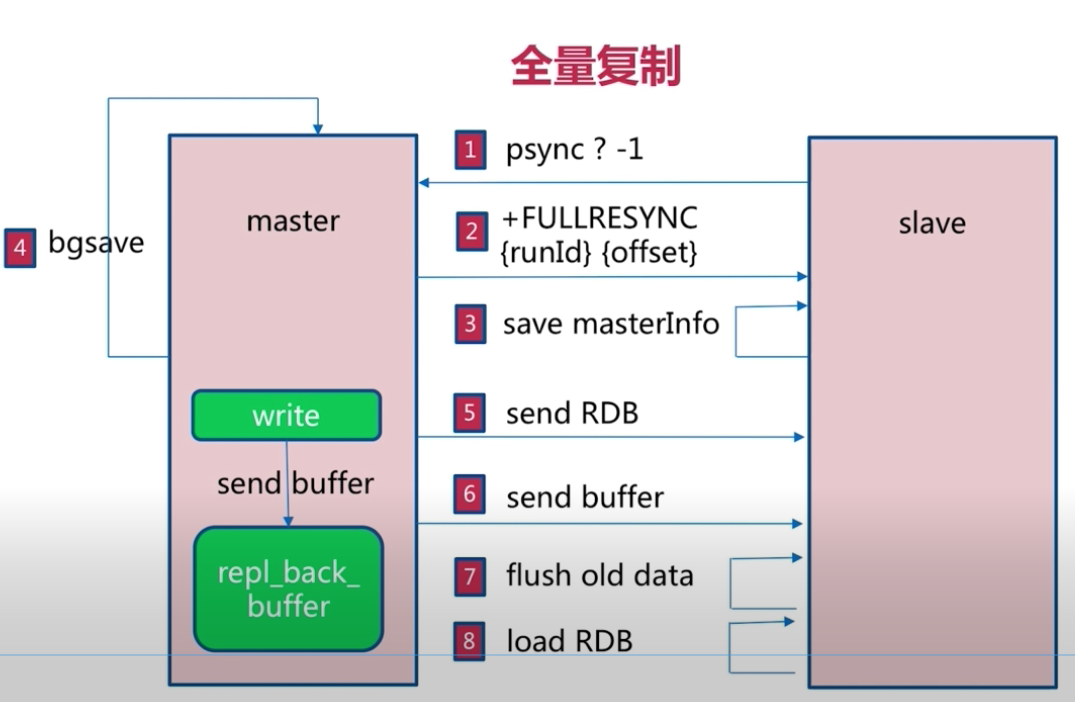
run\_id:每次重启都会改变

#### 偏移量





#### 全量复制过程

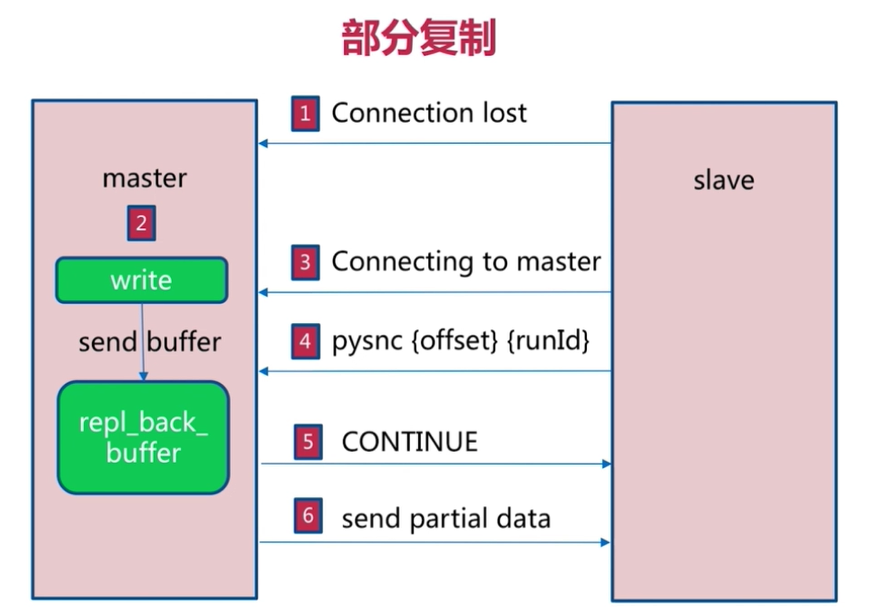


全量复制的开销：

bgsave+rdb文件传输带宽时间+flush old data时间+加载rdb时间+可能的aof重写（如果从上开启了aof）

如果网络抖动，2.8以上提供了部分复制功能。

#### 部分复制过程



前提是，断网的这段时间，buffer里面的数据包含了偏移量，如果这段时间的数据太多，超过了buffer，还是要全量复制。

#### 如何规避全量复制

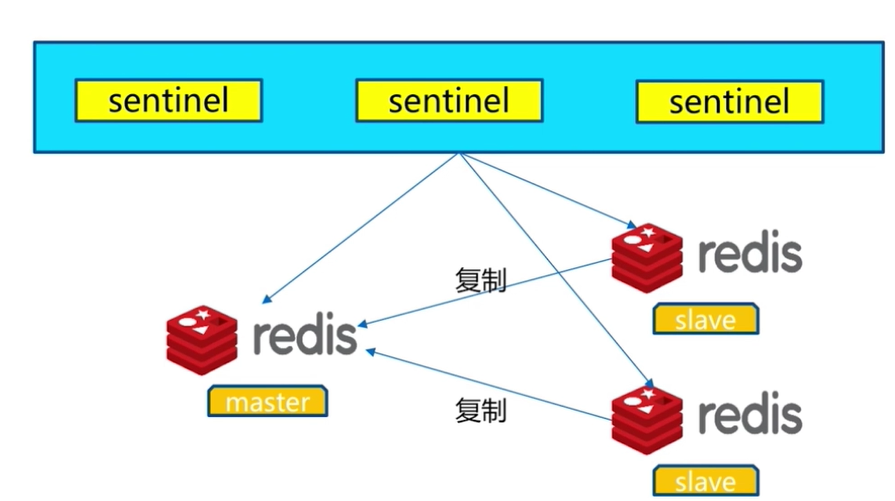
1. 第一次全量不可避免
2. Max memory 不要设置的太大，在业务低峰进行复制
3. 主节点重启导致run id发生变化，也会导致全量复制
4. 使用sentinel或者cluster做故障转移，主节点故障以后，从节点晋升成主节点
5. 复制缓冲区不足也会导致全量复制，默认值是1M，发生了网络抖动暂存数据，可以调大rel\_backlog\_size=10M

#### 如何避免复制风暴

如果主节点下面有多个从节点，如果主节点挂掉，重启以后，因为runid发生了变化，所有的从节点都过来做同步。

可以更改网络拓扑图，一个人主挂一个从，从再挂从。

## 7.Redis Sentinel

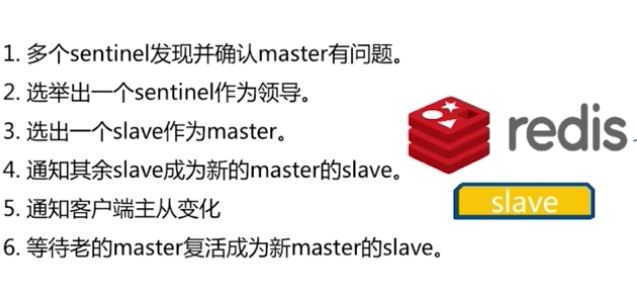


Sentinel可以监控多个主从集群，通过mastername来区分

客户端从Sentinel来获取连接信息。

多个Sentinel会监控redis主从，先选举一个leader sentinel，选一个slave作为master，通知别的slave新的master，通知客户端，等待老的master复活，复活以后变成新的master的slave。

#### 故障转移过程



#### 搭建Sentinel环境

Sentinel1：127.0.0.1 27000

Sentinel2：127.0.0.1 27001

Sentinel3：127.0.0.1 27002

master： 127.0.0.1 7000

slave1: 127.0.0.1 7001

slave2: 127.0.0.1 7002

##### master配置：

[root@localhost redis]# cat ./config/redis-7000.conf

bind 0.0.0.0

port 7000

daemonize yes

dir /usr/local/redis/data

pidfile redis\_7000.pid

logfile redis\_7000.log

dbfilename dump\_7000.rdb

##### slave1配置：

[root@localhost redis]# cat ./config/redis-7001.conf

bind 0.0.0.0

port 7001

daemonize yes

dir /usr/local/redis/data

pidfile redis\_7001.pid

logfile redis\_7001.log

dbfilename dump\_7001.rdb

slaveof 127.0.0.1 7000

##### slave2配置：

[root@localhost redis]# cat ./config/redis-7002.conf

bind 0.0.0.0

port 7002

daemonize yes

dir /usr/local/redis/data

pidfile redis\_7002.pid

logfile redis\_7002.log

dbfilename dump\_7002.rdb

slaveof 127.0.0.1 7000

##### 检查主从

[root@localhost redis]# redis-cli -p 7000 info replication

# Replication

role:master

connected\_slaves:2

slave0:ip=127.0.0.1,port=7001,state=online,offset=126,lag=0

slave1:ip=127.0.0.1,port=7002,state=online,offset=126,lag=1

master\_replid:8023fa9400c3b29f316fd659f1e91c37ca680e8d

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:126

second\_repl\_offset:-1

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:1

repl\_backlog\_histlen:126

##### Sentinel配置：

[root@localhost config]# cat sentinel-27000.conf

port 27000

daemonize yes

dir /usr/local/redis/data

sentinel monitor mymaster 127.0.0.1 7000 2 # mymaster集群的名字，127.0.0.1 7000主的IP端口，2个sentinel认为master挂掉就做故障转移

sentinel down-after-milliseconds mymaster 30000#超过30秒就会认为挂掉

sentinel parallel-syncs mymaster 1 #新主每次只会向1个master同步数据

sentinel failover-timeout mymaster 180000 #

启动：./src/redis-sentinel ./config/sentinel-27000.conf

[root@localhost redis]# redis-cli -p 27000 info sentinel

# Sentinel

sentinel\_masters:1

sentinel\_tilt:0

sentinel\_running\_scripts:0

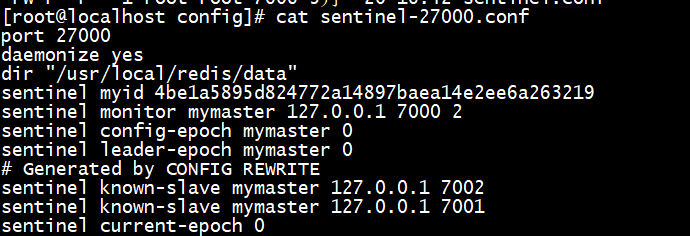
sentinel\_scripts\_queue\_length:0

sentinel\_simulate\_failure\_flags:0

master0:name=mymaster,status=ok,address=127.0.0.1:7000,slaves=2,sentinels=1

[root@localhost redis]#

配置发生了变化，自动发现了slave：



sed s/27000/27001/g sentinel-27000.conf > sentinel-27001.conf

删掉myid

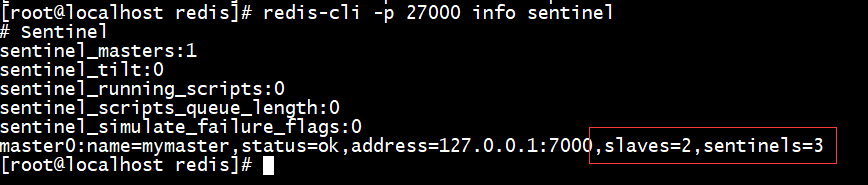
sed s/27000/27002/g sentinel-27000.conf > sentinel-27002.conf

删掉myid

启动

./src/redis-sentinel ./config/sentinel-27001.conf

./src/redis-sentinel ./config/sentinel-27002.conf



##### Sentinel内部的3个定时任务：

1. 每10秒每个sentinel对master和slave执行info

可以获取slave节点

确定主从关系

1. 每2秒，每个sentinel通过master节点的channel交换信息（pub/sub）

通过\_sentinel\_:hello频道交互

交互对节点的“看法”和自身信息

1. 每一秒。每个sentinel节点会对其他的sentinel和redis执行ping，做心跳，做失败检测

#### 主管下线和客观下线

sentinel monitor mymaster 127.0.0.1 7000 2 # mymaster集群的名字，127.0.0.1 7000主的IP端口，2个sentinel认为master挂掉就做故障转移

sentinel down-after-milliseconds mymaster 30000#每秒ping，超过30秒没有收到回复，主观下线

sentinel parallel-syncs mymaster 1 #新主每次只会向1个master同步数据

sentinel failover-timeout mymaster 180000 #

先做主观下线，然后发送 sentinel is-master-down-by-addr询问别的sentinel是否认为这个节点已经下线，当达成共识以后，做一个客观下线，认为这个master真的挂掉了。

#### 领导者选举

只需要一个sentinel做故障转移就够了

发送sentinel is-master-down-by-addr来选举master，表示自己想成为leader。

收到命令的sentinel节点如果还没同意其他的sentinel节点发送的命令就会同意该请求，否则拒绝

如果该sentinel发现自己的票数已经超过了集合半数且超过quorum，那么他会变成leader

如果多个sentinel变成了leader会过一段时间重新选举

#### 故障转移

从slave中选举一个合适的节点作为master

对选中的slave节点执行slaveof no one让其成为新的master

向剩余的slave发送命令，让他们成为新的master的slave，根据parallel-syncs参数复制rdb文件。

更新原来的master为slave，并保持关注，当它复活以后会变成新的master的slave。

#### 手动故障转移

Sentinel failover masterName# 在sentinel节点上执行，就不需要再主观下线、客观下线、领导者选举，只需要做故障转移就ok了

对于从节点：临时下线还是永久下线

## 8.Redis Cluster

服务端分片slot：0-16383总共16384个槽，添加节点对老的数据没有任何影响，不会出现老的key对应的数据丢失问题。

#### 手动搭建集群

（1）开启配置节点

（2）meet

（3）指派槽

（4）主从

##### 配置节点

[root@localhost cluster]# cat 7000.conf

port 7000

daemonize yes

dir /usr/local/redis/data

logfile '7000.log'

pidfile '7000.pid'

dbfilename 'dump-7000.rdb'

cluster-enabled yes

cluster-config-file nodes-7000.conf

cluster-require-full-coverage no

[root@localhost cluster]# sed s/7000/7001/g 7000.conf > 7001.conf

[root@localhost cluster]# sed s/7000/7002/g 7000.conf > 7002.conf

[root@localhost cluster]# sed s/7000/7003/g 7000.conf > 7003.conf

[root@localhost cluster]# sed s/7000/7004/g 7000.conf > 7004.conf

[root@localhost cluster]# sed s/7000/7005/g 7000.conf > 7005.conf

启动：

[root@localhost redis]# ./src/redis-server ./config/cluster/7000.conf

[root@localhost redis]# ./src/redis-server ./config/cluster/7001.conf

[root@localhost redis]# ./src/redis-server ./config/cluster/7002.conf

[root@localhost redis]# ./src/redis-server ./config/cluster/7003.conf

[root@localhost redis]# ./src/redis-server ./config/cluster/7004.conf

[root@localhost redis]# ./src/redis-server ./config/cluster/7005.conf



[root@localhost data]# cat nodes-7000.conf

c900315652b59b5abbc43f8bf723ea747a3a34eb :0@0 myself,master - 0 0 0 connected

vars currentEpoch 0 lastVoteEpoch 0

[root@localhost data]# redis-cli -p 7000 cluster nodes

c900315652b59b5abbc43f8bf723ea747a3a34eb :7000@17000 myself,master - 0 0 0 connected

[root@localhost data]# redis-cli -p 7000 cluster info

cluster\_state:fail #fail

cluster\_slots\_assigned:0

cluster\_slots\_ok:0

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:1

cluster\_size:0

cluster\_current\_epoch:0

cluster\_my\_epoch:0

cluster\_stats\_messages\_sent:0

cluster\_stats\_messages\_received:0

##### meet：gossip协议

[root@localhost data]# redis-cli -p 7000 cluster meet 127.0.0.1 7001

OK

[root@localhost data]# redis-cli -p 7000 cluster nodes

c900315652b59b5abbc43f8bf723ea747a3a34eb 127.0.0.1:7000@17000 myself,master - 0 0 0 connected

070d9bec78156c612e545f73951330ca8864a915 127.0.0.1:7001@17001 master - 0 1521873839529 1 connected

[root@localhost data]# redis-cli -p 7000 cluster meet 127.0.0.1 7002

OK

[root@localhost data]# redis-cli -p 7000 cluster meet 127.0.0.1 7003

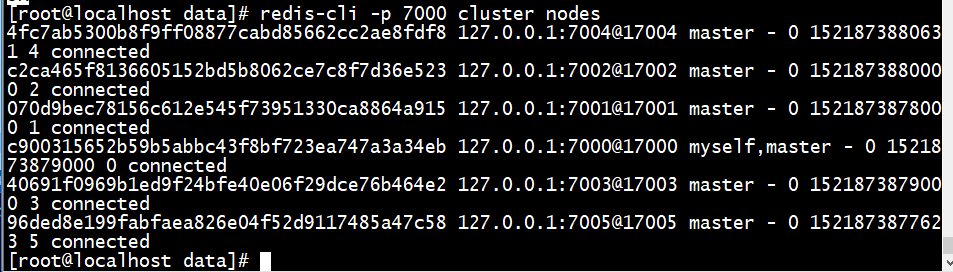
OK

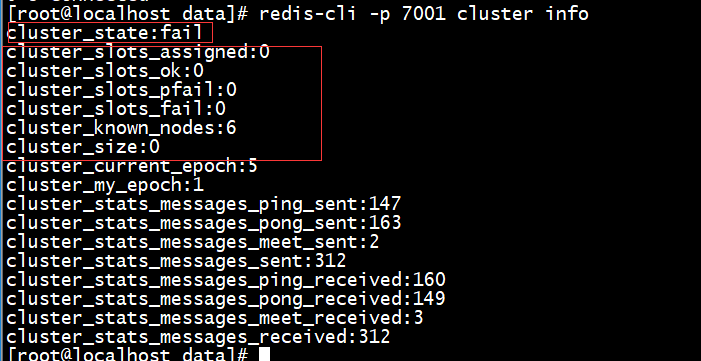
[root@localhost data]# redis-cli -p 7000 cluster meet 127.0.0.1 7004

OK

[root@localhost data]# redis-cli -p 7000 cluster meet 127.0.0.1 7005

OK





##### 分配槽

[root@localhost cluster]# cat slot.sh

start=$1

end=$2

port=$3

for slot in `seq ${start} ${end}`

do

echo ${slot}

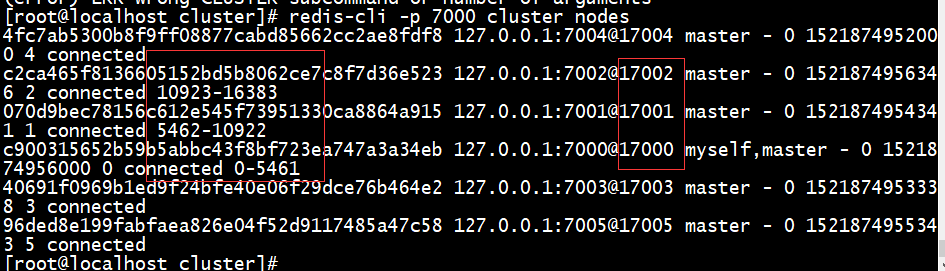
redis-cli -p ${port} cluster addslots ${slot}

done

[root@localhost cluster]# ./slot.sh 0 5461 7000

[root@localhost cluster]# ./slot.sh 5462 10922 7001

[root@localhost cluster]# ./slot.sh 10923 16383 7002





##### 设置主从关系

[root@localhost cluster]# redis-cli -p 7003 cluster replicate c900315652b59b5abbc43f8bf723ea747a3a34eb

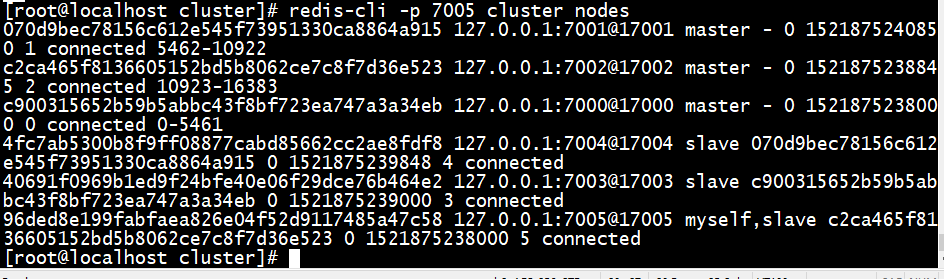
OK

[root@localhost cluster]# redis-cli -p 7004 cluster replicate 070d9bec78156c612e545f73951330ca8864a915

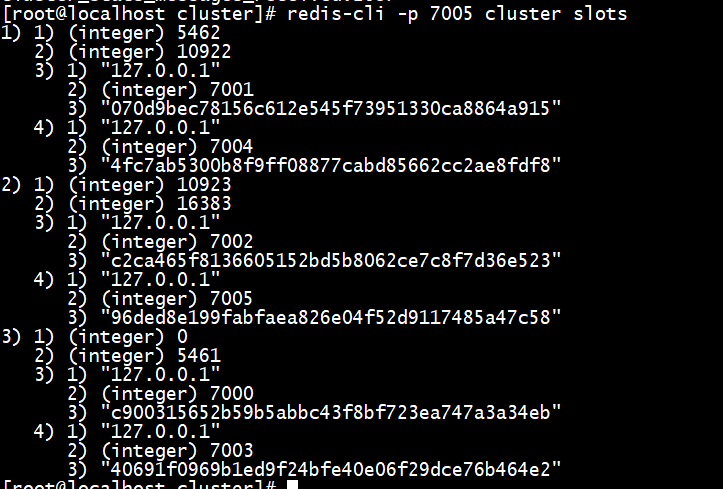
OK

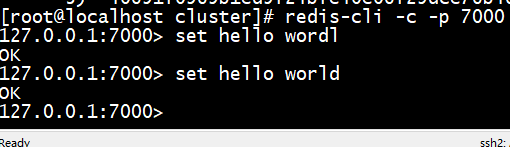
[root@localhost cluster]# redis-cli -p 7005 cluster replicate c2ca465f8136605152bd5b8062ce7c8f7d36e523

OK









#### 官方ruby工具安装

wget https://cache.ruby-china.org/pub/ruby/ruby-2.3.1.tar.gz

tar xzvf ruby-2.3.1.tar.gz

cd ruby-2.3.1

./configure -prefix=/usr/local/ruby

make && make install

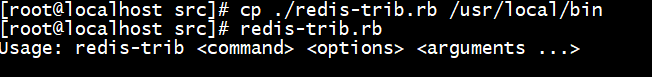
wget <http://rubygems.org/downloads/redis-3.3.0.gem>

gem install –l redis-3.3.0.gem

gem list -- check redis gem # 查看



把redis-trib.rb拷贝到系统目录：



启动6个实例：

ps -ef | grep cluster | grep -v grep |awk '{print $2}' | xargs kill -9



[root@localhost redis]# redis-trib.rb create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005

>>> Creating cluster

>>> Performing hash slots allocation on 6 nodes...

Create：创建集群

--replicats 1：每个主节点有一个从节点

Host：port：集群中的所有节点

Using 3 masters:

127.0.0.1:7000

127.0.0.1:7001

127.0.0.1:7002

Adding replica 127.0.0.1:7003 to 127.0.0.1:7000

Adding replica 127.0.0.1:7004 to 127.0.0.1:7001

Adding replica 127.0.0.1:7005 to 127.0.0.1:7002

M: c3dbaa1d7af1ebfda170018a1c5027c0d8f34a7a 127.0.0.1:7000

slots:0-5460 (5461 slots) master

M: d193402d8dca30607b48e78356fe28298e91ae18 127.0.0.1:7001

slots:5461-10922 (5462 slots) master

M: 1471c8bd503d856ddc7769a34532086b19761030 127.0.0.1:7002

slots:10923-16383 (5461 slots) master

S: e206bdd9fa64de61fb55d3adcac0adb33d1c0507 127.0.0.1:7003

replicates c3dbaa1d7af1ebfda170018a1c5027c0d8f34a7a

S: 825e4c0e224e3c26148110eabc1a4116cd13f006 127.0.0.1:7004

replicates d193402d8dca30607b48e78356fe28298e91ae18

S: e27d4b682dddf770b08f1326a24f22afab8e61a7 127.0.0.1:7005

replicates 1471c8bd503d856ddc7769a34532086b19761030

Can I set the above configuration? (type 'yes' to accept): yes

>>> Nodes configuration updated

>>> Assign a different config epoch to each node

>>> Sending CLUSTER MEET messages to join the cluster

Waiting for the cluster to join....

>>> Performing Cluster Check (using node 127.0.0.1:7000)

M: c3dbaa1d7af1ebfda170018a1c5027c0d8f34a7a 127.0.0.1:7000

slots:0-5460 (5461 slots) master

1 additional replica(s)

S: e27d4b682dddf770b08f1326a24f22afab8e61a7 127.0.0.1:7005

slots: (0 slots) slave

replicates 1471c8bd503d856ddc7769a34532086b19761030

M: 1471c8bd503d856ddc7769a34532086b19761030 127.0.0.1:7002

slots:10923-16383 (5461 slots) master

1 additional replica(s)

M: d193402d8dca30607b48e78356fe28298e91ae18 127.0.0.1:7001

slots:5461-10922 (5462 slots) master

1 additional replica(s)

S: e206bdd9fa64de61fb55d3adcac0adb33d1c0507 127.0.0.1:7003

slots: (0 slots) slave

replicates c3dbaa1d7af1ebfda170018a1c5027c0d8f34a7a

S: 825e4c0e224e3c26148110eabc1a4116cd13f006 127.0.0.1:7004

slots: (0 slots) slave

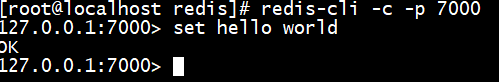
replicates d193402d8dca30607b48e78356fe28298e91ae18

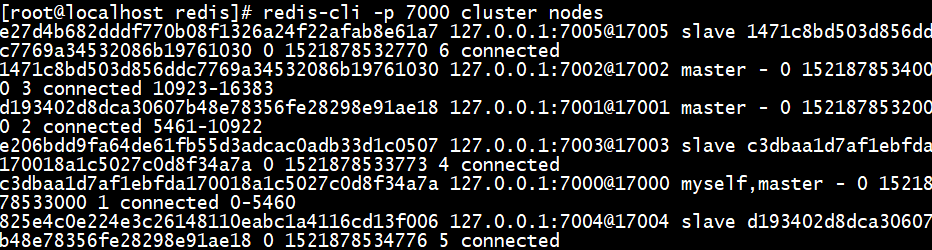
[OK] All nodes agree about slots configuration.

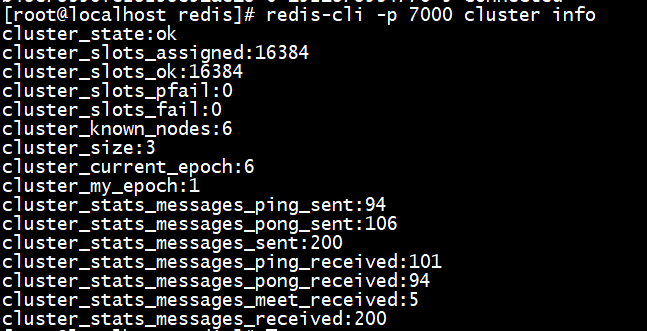
>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.



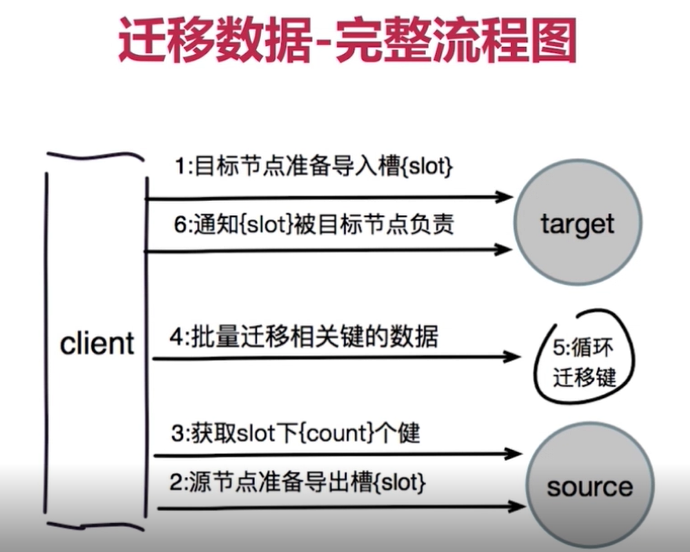




#### 集群伸缩

##### 加入节点

三步：启动节点，加入集群，迁移槽





启动节点

[root@localhost redis]# ./src/redis-server ./config/cluster/7006.conf

[root@localhost redis]# ./src/redis-server ./config/cluster/7007.conf

[root@localhost redis]#

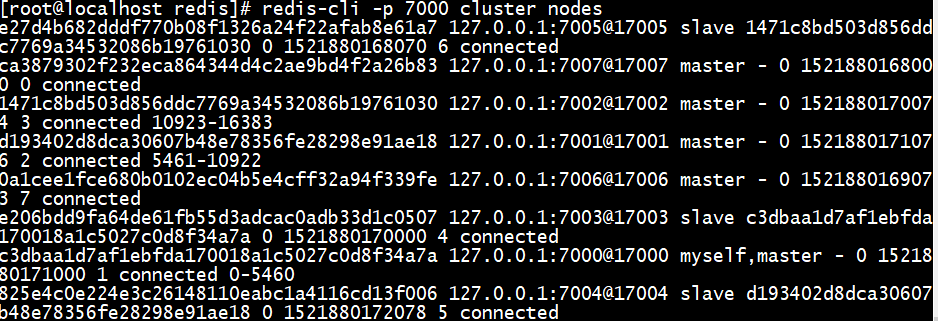
加入集群

[root@localhost redis]# redis-cli -p 7000 cluster meet 127.0.0.1 7006

OK

[root@localhost redis]# redis-cli -p 7000 cluster meet 127.0.0.1 7007

OK



设置主从：

[root@localhost redis]# redis-cli -p 7007 cluster replicate 0a1cee1fce680b0102ec04b5e4cff32a94f339fe

OK

分配槽：

[root@localhost redis]# redis-trib.rb reshard 127.0.0.1:7000

>>> Performing Cluster Check (using node 127.0.0.1:7000)

M: c3dbaa1d7af1ebfda170018a1c5027c0d8f34a7a 127.0.0.1:7000

slots:0-5460 (5461 slots) master

1 additional replica(s)

S: e27d4b682dddf770b08f1326a24f22afab8e61a7 127.0.0.1:70

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

How many slots do you want to move (from 1 to 16384)? 4096

What is the receiving node ID? 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942

Please enter all the source node IDs.

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

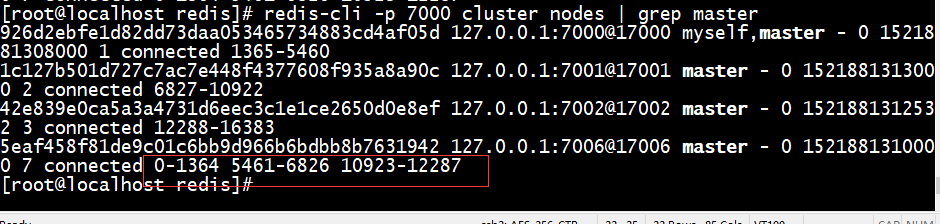
Type 'done' once you entered all the source nodes IDs.

Source node #1:all

Moving slot 12286 from 42e839e0ca5a3a4731d6eec3c1e1ce2650d0e8ef

Moving slot 12287 from 42e839e0ca5a3a4731d6eec3c1e1ce2650d0e8ef

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



##### 收缩集群

也是3步（1）迁移槽（2）忘记节点（3）关闭节点

迁移槽：

[root@localhost redis]# redis-trib.rb reshard --from 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942 --to 926d2ebfe1d82dd73daa053465734883cd4af05d --slots 1365 127.0.0.1:7006

Moving slot 1362 from 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942

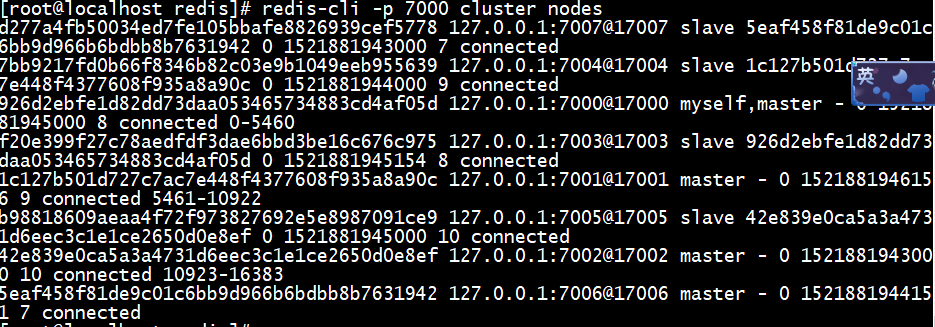
Moving slot 1363 from 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942

Moving slot 1364 from 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942

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

[root@localhost redis]# redis-trib.rb reshard --from 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942 --to 1c127b501d727c7ac7e448f4377608f935a8a90c --slots 1366 127.0.0.1:7006

[root@localhost redis]# redis-trib.rb reshard --from 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942 --to 42e839e0ca5a3a4731d6eec3c1e1ce2650d0e8ef --slots 1365 127.0.0.1:7006



下线节点，先下从节点，再下主节点：

[root@localhost redis]# redis-trib.rb del-node 127.0.0.1:7000 d277a4fb50034ed7fe105bbafe8826939cef5778

>>> Removing node d277a4fb50034ed7fe105bbafe8826939cef5778 from cluster 127.0.0.1:7000

>>> Sending CLUSTER FORGET messages to the cluster...

>>> SHUTDOWN the node.

[root@localhost redis]# redis-trib.rb del-node 127.0.0.1:7000 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942

>>> Removing node 5eaf458f81de9c01c6bb9d966b6bdbb8b7631942 from cluster 127.0.0.1:7000

>>> Sending CLUSTER FORGET messages to the cluster...

>>> SHUTDOWN the node.

[root@localhost redis]# ps -ef | grep 7006

root 2789 15366 0 17:03 pts/1 00:00:00 grep --color=auto 7006

[root@localhost redis]# ps -ef | grep 7007

root 2791 15366 0 17:03 pts/1 00:00:00 grep --color=auto 7007

#### 客户端路由

1. moved异常（2）ask异常（3）smart客户端

（1）moved异常

Redis-cli –c –p 7000 如果槽不在7000 会自动重定向到7001

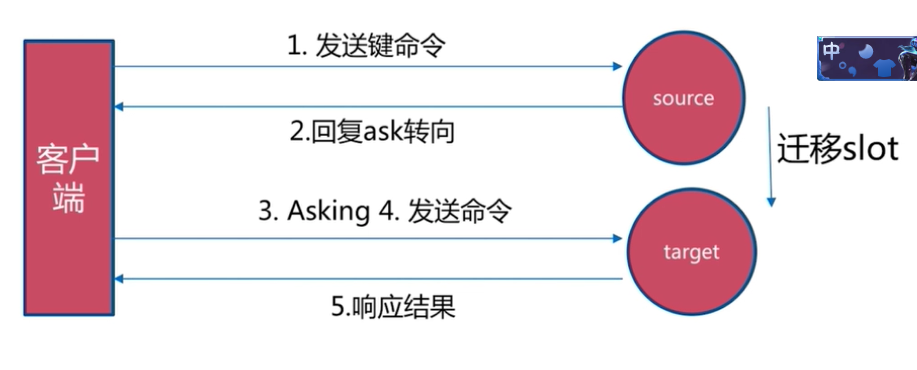
Redis-cli –p 7000如果槽不在7000 会返回moved异常

cluster keyslot hello 可以算出来key的slot

[root@localhost redis]# redis-cli -p 7000 cluster keyslot php

(integer) 9244

1. ask异常



在数据迁移过程中发生的

1. smart客户端

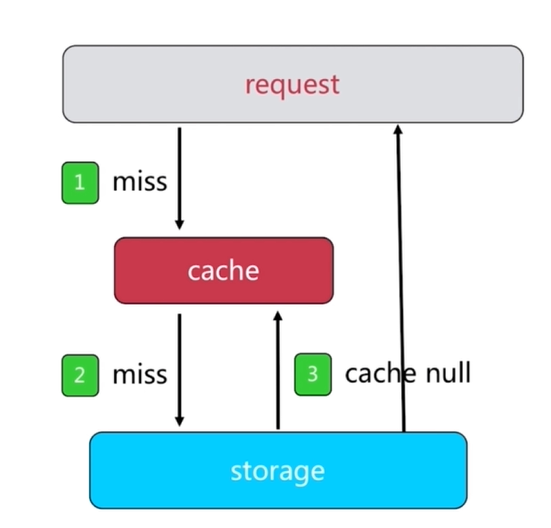
1.从集群中选一个可运行的节点，使用cluster slots初始化槽和节点映射。

2.讲cluster slots的结果映射到本地，为每一个节点创建JedisPool

3.准备执行命令

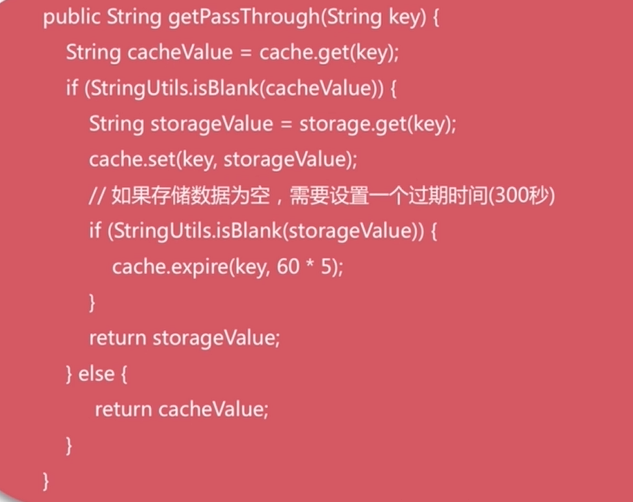
## 9.缓存的问题

#### 缓存穿透-大量请求不命中，同时DB中没有查询出结果来



解决办法：（1）缓存空对象，set nx（2）布隆过滤器

缺点：（1）需要更多的key（2）cache与db数据不一致



#### 缓存雪崩-在同一个时间数据同时失效

设置随机的过期时间

#### 缓存无底洞

优化io性能：

1. 命令本身的优化，不适用慢查询keys hgetall bugkey
2. 减少网络通信次数
3. 客户端使用连接池 nio