1.全局异常：

实现HanlerExceptionResolver

MappingJacksonJsonView

2.REDIS死锁：

通过代码来控制

未获取到锁后可以继续判断，判断时间戳，看是否可以重置并获取到锁

或者用redission trylock里的waittime参数可以设置成0避免两边同时拿到分布式锁的问题

3.获取属性文件值可以通过@Value("${myProps.arrayProps}")放在属性上面来得到

但是只能得到基本类型，类中要引入@ConfigurationProperties(prefix = "XXX")

1. redis配置分离及随机启动
2. 这里用脚本方式来启动。redis utils目录下，有个redis\_init\_script脚本



这里配置文件有目录约定，我们可以直接按照上面的约定来命名相关文件。

1. 将redis\_init\_script脚本拷贝到linux的/etc/init.d目录中，将redis\_init\_script重命名为redis\_6379（这个名字无所谓，还可以叫做redisd,通常以d结尾表示是后台自启动服务），6379是我们希望这个redis实例监听的端口号

cp redis\_init\_script /etc/init.d/redis\_6379

（3）修改redis\_6379脚本的第6行的REDISPORT，设置为相同的端口号（默认就是6379）

（4）创建两个目录：/etc/redis（存放redis的配置文件），/var/redis/6379（存放redis的持久化文件）

（5）修改redis配置文件（默认在根目录下，redis.conf），拷贝到电脑/etc/redis/6379.conf文件

cp redis.conf /etc/redis/6379.conf

（6）修改redis.conf中的部分配置为生产环境

daemonize yes 让redis以daemon进程运行

pidfile /var/run/redis\_6379.pid 设置redis的pid文件位置

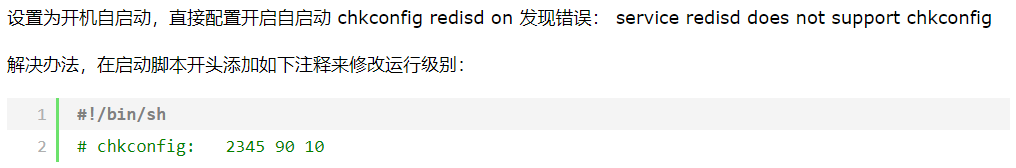
port 6379 设置redis的监听端口号

dir /var/redis/6379 设置持久化文件的存储位置

（7）启动redis，执行cd /etc/init.d, chmod 777 redis\_6379，./redis\_6379 start

（8）确认redis进程是否启动，ps -ef | grep redis

1. 让redis跟随系统启动自动启动



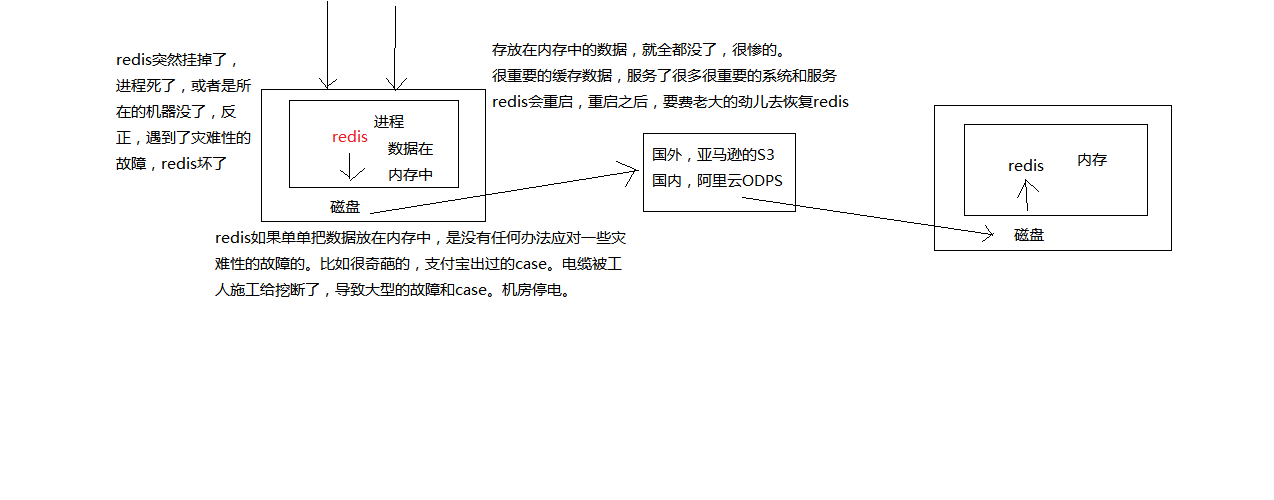
在redis\_6379脚本中，最上面，加入

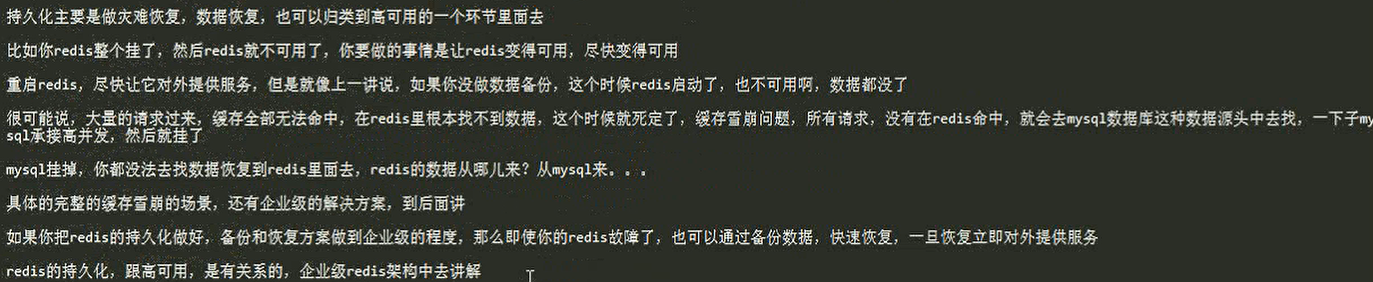
# chkconfig: 2345 90 10

再设置自启动

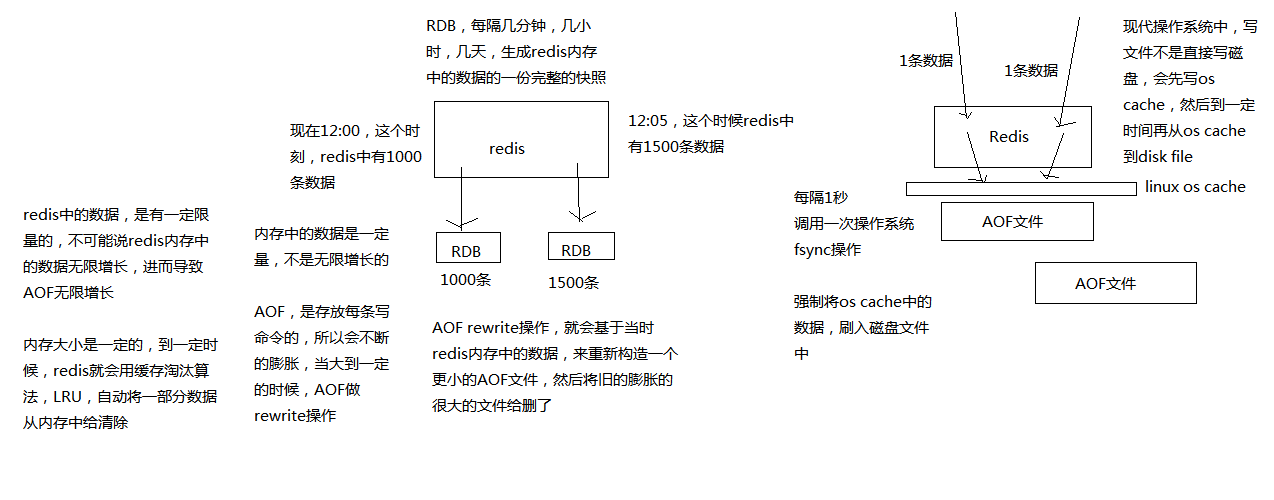
chkconfig redis\_6379 on

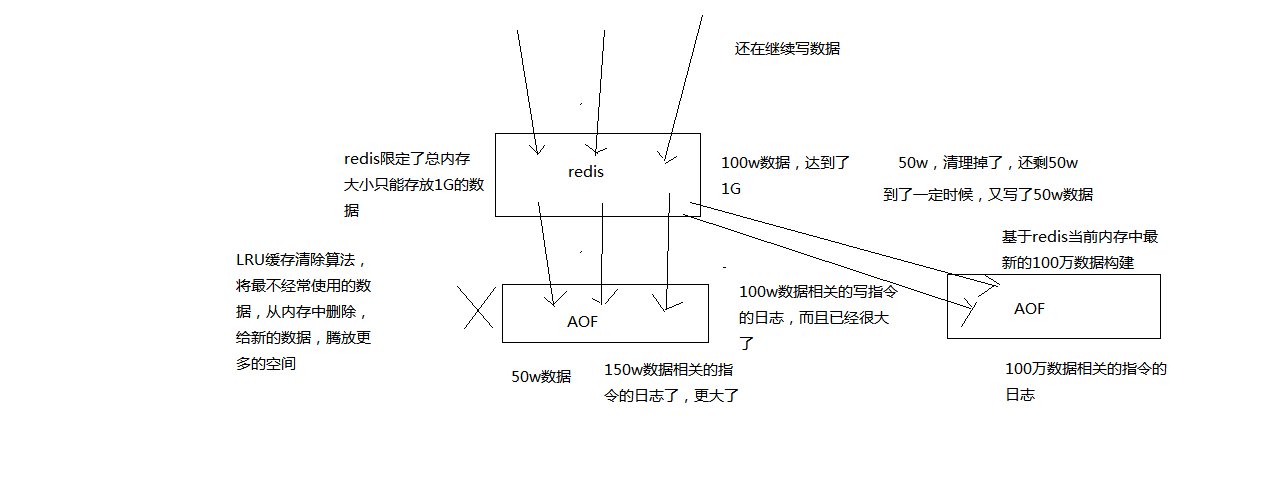
1. 装好redis就直接在/usr/bin下面有命令了，不管在什么目录下，都可以直接redis-cli了。
2. redis持久化
   1. 意义





* 1. AOF和RDB





aof :appendonly 设置成yes

rdb ：加/改save

如果都aof和rdb都打开，那默认他会先去读AOF

AOF损坏怎么办？

用redis-check-aof命令修复

以前AOF发生过bug，就是通过AOF记录的日志，进行数据恢复的时候，没有恢复一模一样的数据出来。所以说，类似AOF这种较为复杂的基于命令日志/merge/回放的方式，比基于RDB每次持久化一份完整的数据快照文件的方式，更加脆弱一些，容易有bug。不过AOF就是为了避免rewrite过程导致的bug，因此每次rewrite并不是基于旧的指令日志进行merge的，而是基于当时内存中的数据进行指令的重新构建，这样健壮性会好很多。

RDB和AOF到底该如何选择

（1）不要仅仅使用RDB，因为那样会导致你丢失很多数据

（2）也不要仅仅使用AOF，因为那样有两个问题，第一，你通过AOF做冷备，没有RDB做冷备，来的恢复速度更快; 第二，RDB每次简单粗暴生成数据快照，更加健壮，可以避免AOF这种复杂的备份和恢复机制的bug

（3）综合使用AOF和RDB两种持久化机制，用AOF来保证数据不丢失，作为数据恢复的第一选择; 用RDB来做不同程度的冷备，在AOF文件都丢失或损坏不可用的时候，还可以使用RDB来进行快速的数据恢复

RDB:只会出现一个dump.rdb，每次生成一个新的快照，都会覆盖之前的老快照

* 1. 基于RDB持久化机制的数据恢复实验

（1）在redis中保存几条数据，立即停掉redis进程，然后重启redis，看看刚才插入的数据还在不在

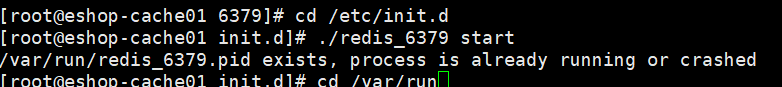
数据还在，为什么？

带出来一个知识点，通过redis-cli SHUTDOWN这种方式去停掉redis，其实是一种安全退出的模式，redis在退出的时候会将内存中的数据立即生成一份完整的rdb快照

/var/redis/6379/dump.rdb

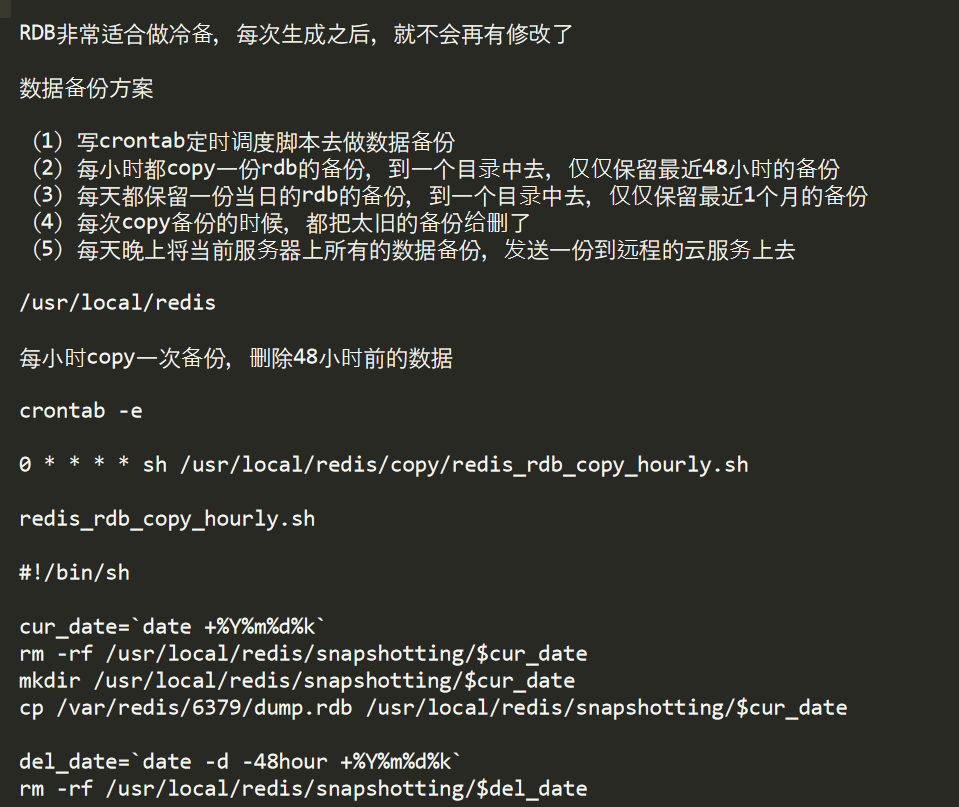
（2）在redis中再保存几条新的数据，用kill -9粗暴杀死redis进程，模拟redis故障异常退出，导致内存数据丢失的场景

这次就发现，redis进程异常被杀掉，数据没有进dump文件，几条最新的数据就丢失了

（kill-9后在/var/run下面还有一个pid，还需要把这个pid给删除掉才行。否则

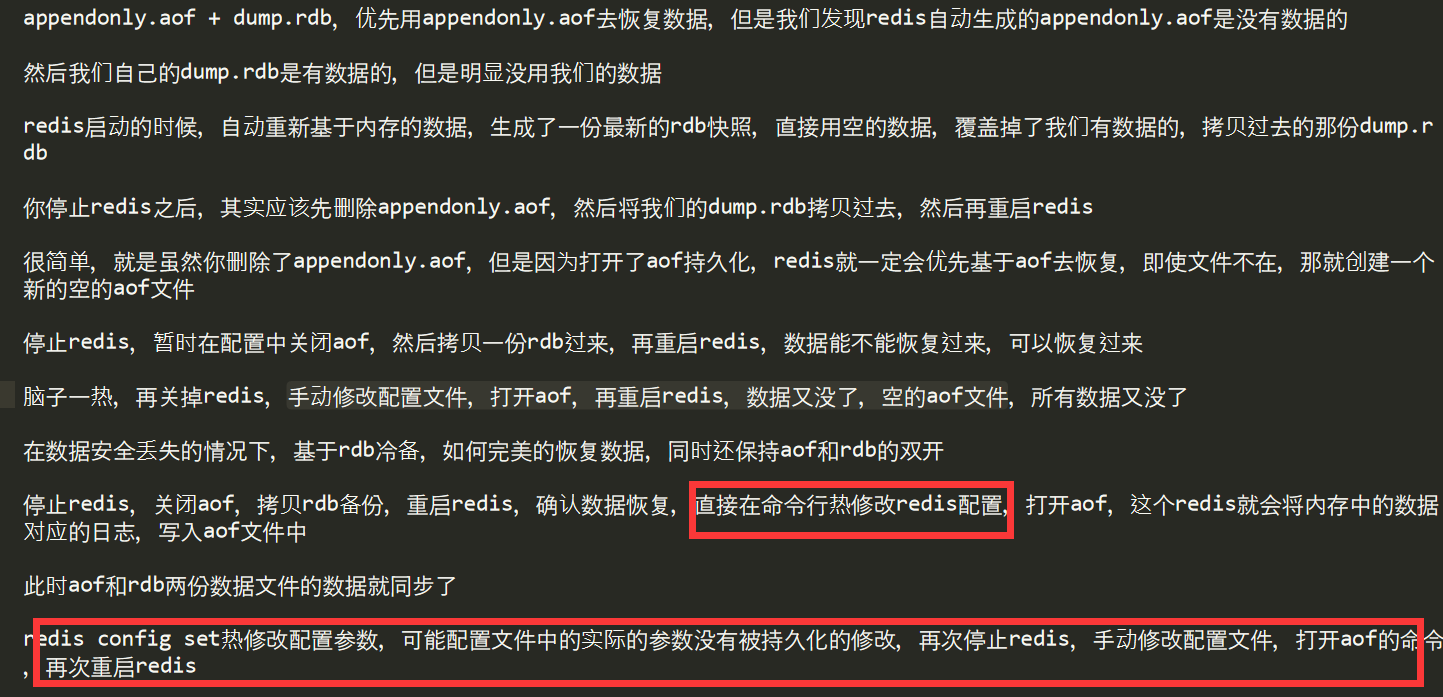
）

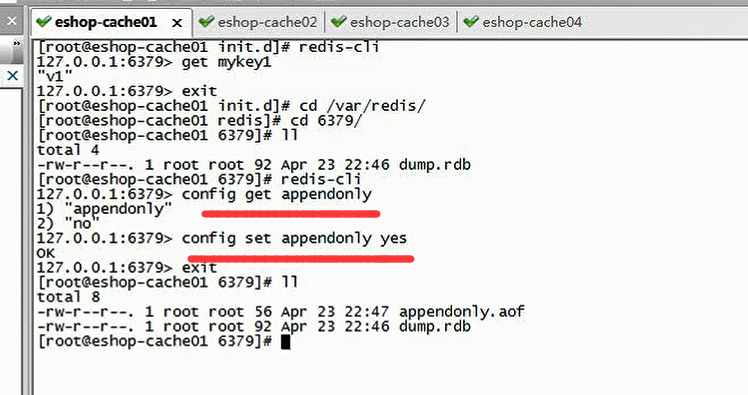
* 1. 手动设置一个save检查点，save 5 1。写入几条数据，等待5秒钟，会发现自动进行了一次dump rdb快照，在dump.rdb中发现了数据。异常停掉redis进程，再重新启动redis，看刚才插入的数据还在
  2. 企业级备份方案
     1. RDB非常适合冷备



* + 1. 如果是redis进程挂掉，那么重启redis进程即可，直接基于AOF日志文件恢复数据
    2. 如果是redis进程所在机器挂掉，那么重启机器后，尝试重启redis进程，尝试直接基于AOF日志文件进行数据恢复。AOF没有破损，也是可以直接基于AOF恢复的。AOF append-only，顺序写入，如果AOF文件破损，那么用redis-check-aof fix
    3. 如果redis当前最新的AOF和RDB文件出现了丢失/损坏，那么可以尝试基于该机器上当前的某个最新的RDB数据副本进行数据恢复

找到RDB最新的一份备份，小时级的备份可以了，小时级的肯定是最新的，copy到redis里面去，就可以恢复到某一个小时的数据





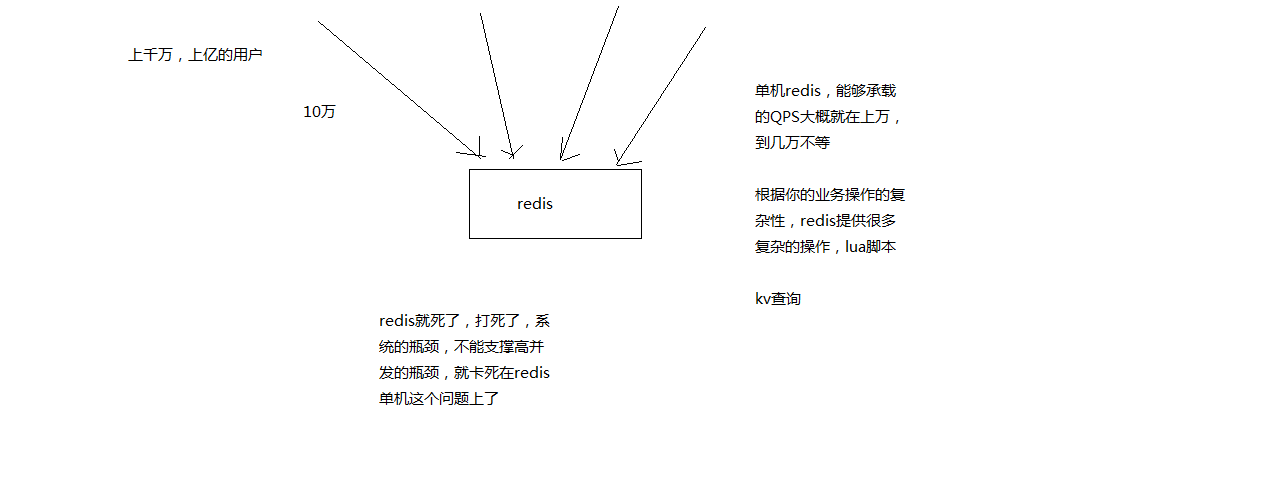
* + 1. 如果当前机器上的所有RDB文件全部损坏，那么从远程的云服务上拉取最新的RDB快照回来恢复数据
    2. 如果是发现有重大的数据错误，比如某个小时上线的程序一下子将数据全部污染了，数据全错了，那么可以选择某个更早的时间点，对数据进行恢复

举个例子，12点上线了代码，发现代码有bug，导致代码生成的所有的缓存数据，写入redis，全部错了

找到一份11点的rdb的冷备，然后按照上面的步骤，去恢复到11点的数据，不就可以了吗

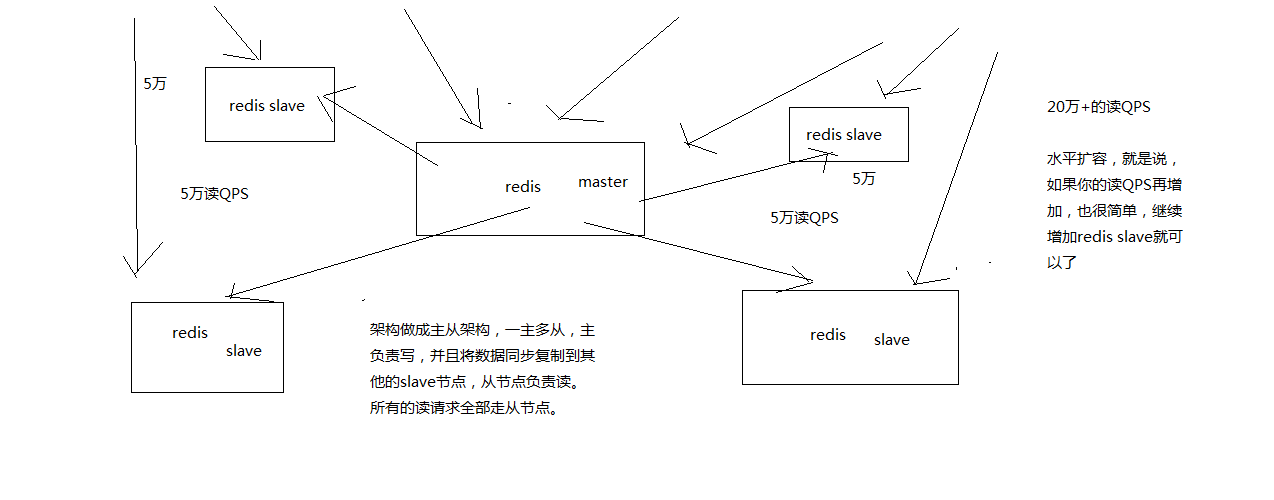
1. 主从
   1. redis不能支撑高并发的瓶颈在哪里？

单机

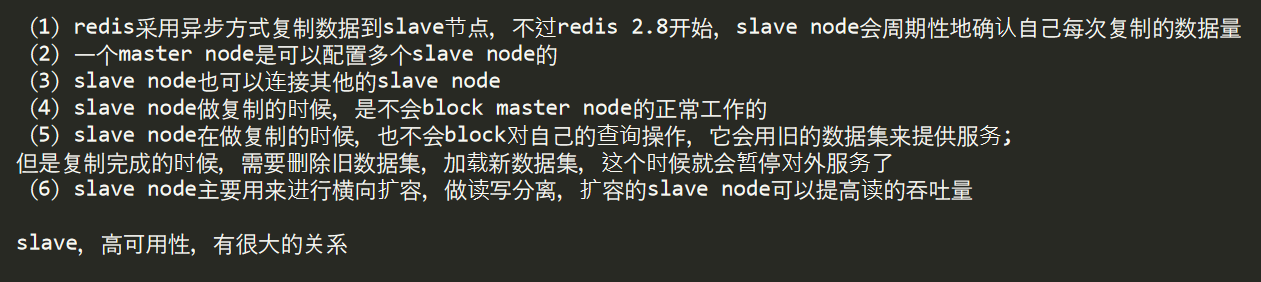


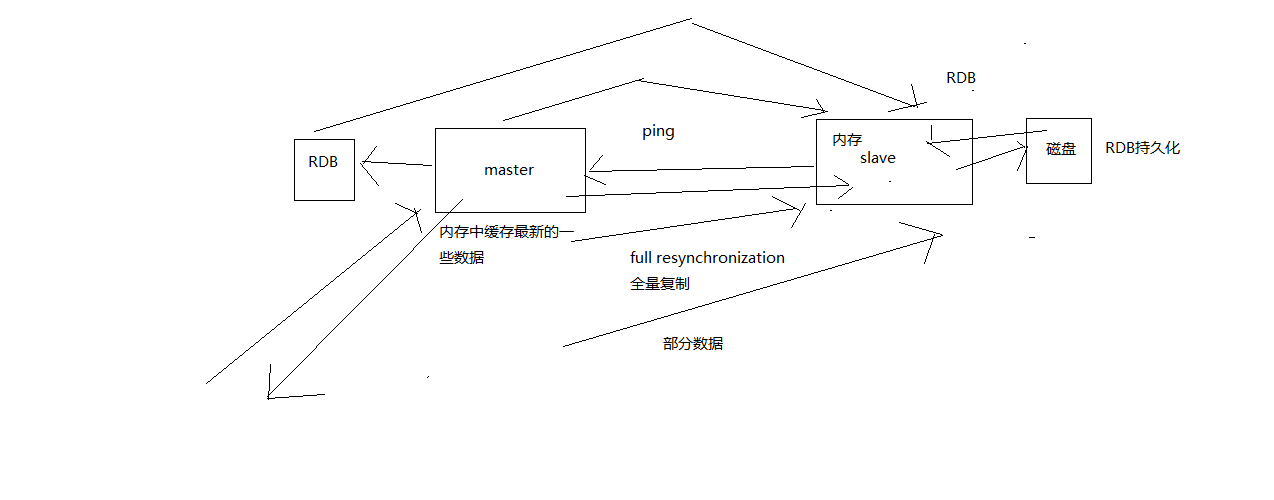
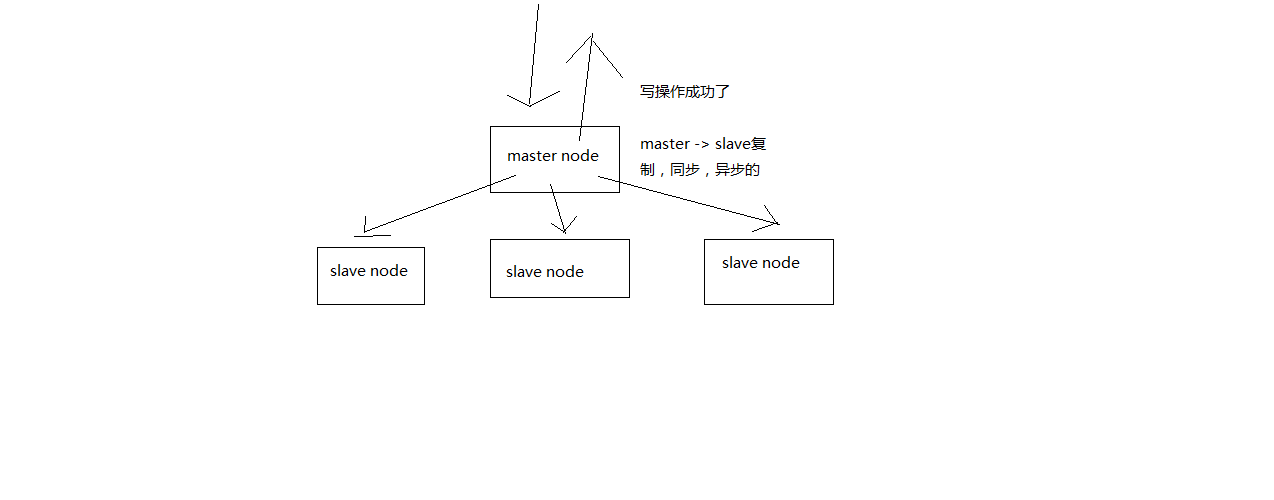
* 1. 如果redis要支撑超过10万+的并发，那应该怎么做？

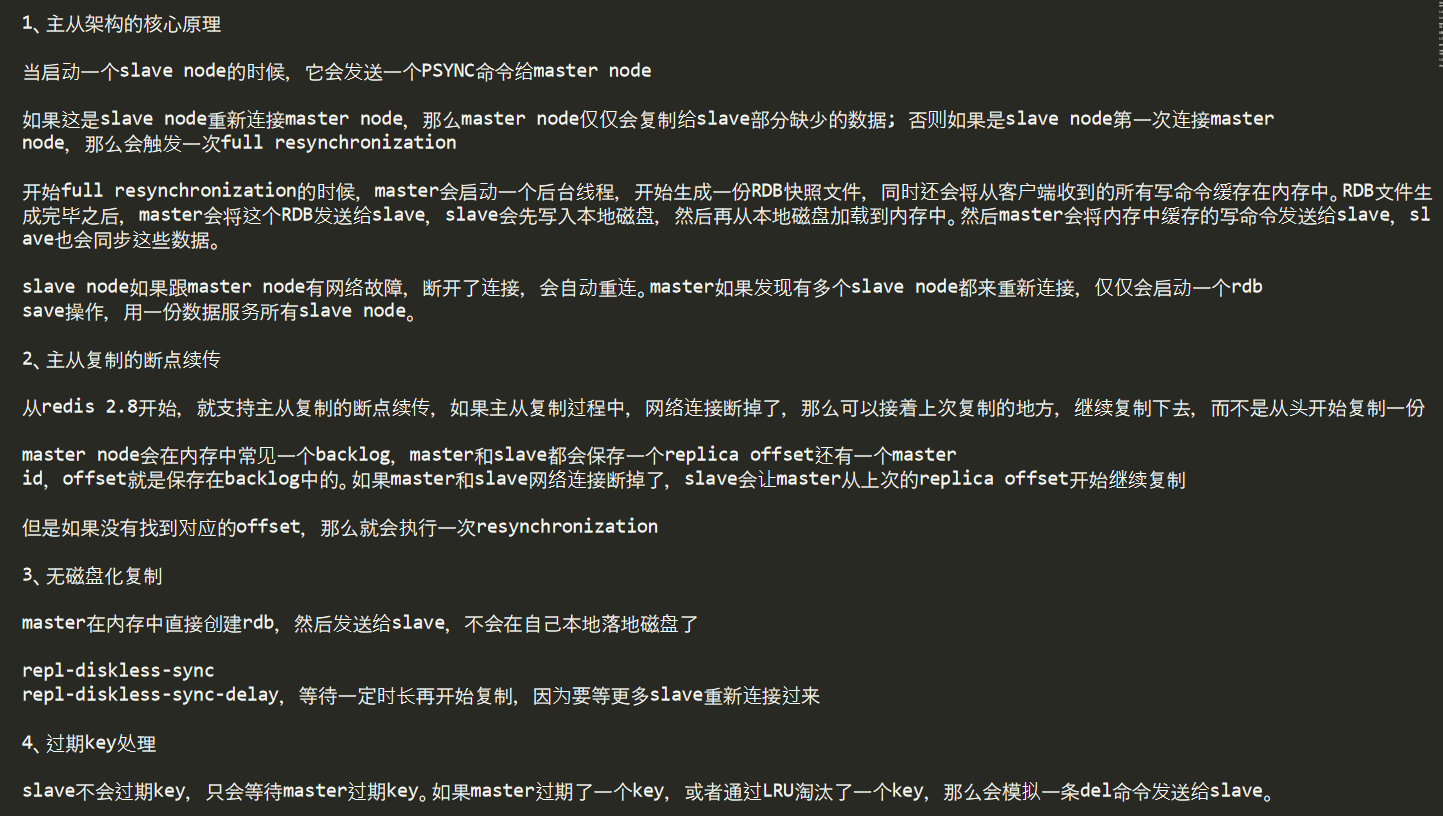
主从架构 -> 读写分离 -> 支撑10万+读QPS的架构

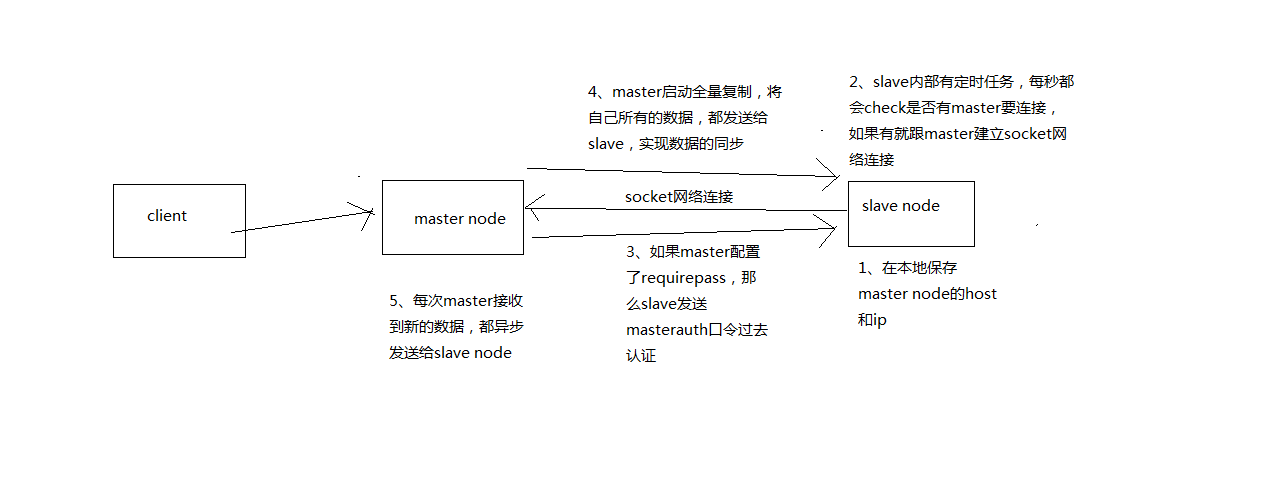


* 1. redis replication的核心机制









1、复制的完整流程

（1）slave node启动，仅仅保存master node的信息，包括master node的host和ip，但是复制流程没开始

master host和ip是从哪儿来的，redis.conf里面的slaveof配置的

（2）slave node内部有个定时任务，每秒检查是否有新的master node要连接和复制，如果发现，就跟master node建立socket网络连接

（3）slave node发送ping命令给master node

（4）口令认证，如果master设置了requirepass，那么salve node必须发送masterauth的口令过去进行认证

（5）master node第一次执行全量复制，将所有数据发给slave node

（6）master node后续持续将写命令，异步复制给slave node

2、数据同步相关的核心机制

指的就是第一次slave连接msater的时候，执行的全量复制，那个过程里面你的一些细节的机制

（1）master和slave都会维护一个offset

master会在自身不断累加offset，slave也会在自身不断累加offset

slave每秒都会上报自己的offset给master，同时master也会保存每个slave的offset

这个倒不是说特定就用在全量复制的，主要是master和slave都要知道各自的数据的offset，才能知道互相之间的数据不一致的情况

（2）backlog

master node有一个backlog，默认是1MB大小

master node给slave node复制数据时，也会将数据在backlog中同步写一份

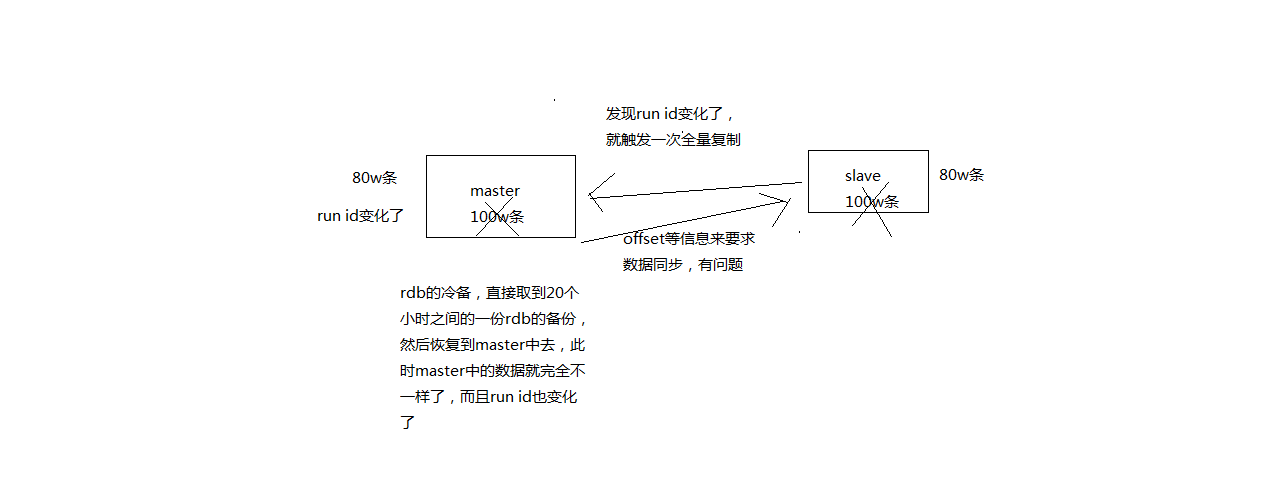
backlog主要是用来做全量复制中断候的增量复制的

（3）master run id

info server，可以看到master run id

如果根据host+ip定位master node，是不靠谱的，如果master node重启或者数据出现了变化，那么slave node应该根据不同的run id区分，run id不同就做全量复制

如果需要不更改run id重启redis，可以使用redis-cli debug reload命令



（4）psync

从节点使用psync从master node进行复制，psync runid offset

master node会根据自身的情况返回响应信息，可能是FULLRESYNC runid offset触发全量复制，可能是CONTINUE触发增量复制

3、全量复制

（1）master执行bgsave，在本地生成一份rdb快照文件

（2）master node将rdb快照文件发送给salve node，如果rdb复制时间超过60秒（repl-timeout），那么slave node就会认为复制失败，可以适当调节大这个参数

（3）对于千兆网卡的机器，一般每秒传输100MB，6G文件，很可能超过60s

（4）master node在生成rdb时，会将所有新的写命令缓存在内存中，在salve node保存了rdb之后，再将新的写命令复制给salve node

（5）client-output-buffer-limit slave 256MB 64MB 60，如果在复制期间，内存缓冲区持续消耗超过64MB，或者一次性超过256MB，那么停止复制，复制失败

（6）slave node接收到rdb之后，清空自己的旧数据，然后重新加载rdb到自己的内存中，同时基于旧的数据版本对外提供服务

（7）如果slave node开启了AOF，那么会立即执行BGREWRITEAOF，重写AOF

rdb生成、rdb通过网络拷贝、slave旧数据的清理、slave aof rewrite，很耗费时间

如果复制的数据量在4G~6G之间，那么很可能全量复制时间消耗到1分半到2分钟

4、增量复制

（1）如果全量复制过程中，master-slave网络连接断掉，那么salve重新连接master时，会触发增量复制

（2）master直接从自己的backlog中获取部分丢失的数据，发送给slave node，默认backlog就是1MB

（3）msater就是根据slave发送的psync中的offset来从backlog中获取数据的

5、heartbeat

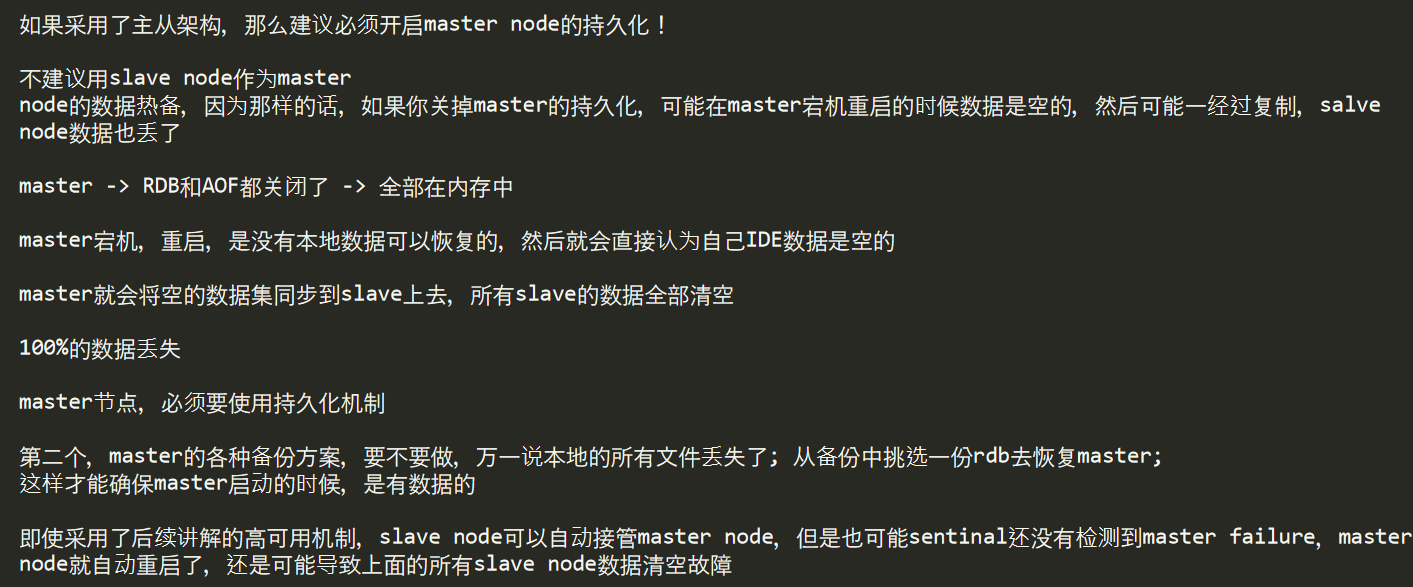
主从节点互相都会发送heartbeat信息

master默认每隔10秒发送一次heartbeat，salve node每隔1秒发送一个heartbeat

6、异步复制

master每次接收到写命令之后，现在内部写入数据，然后异步发送给slave node

* 1. master持久化对于主从架构的安全保障的意义



* 1. 配置主从

redis.conf里在slave node上配置：slaveof 主机IP 6379，即可

还要把所有机器上的bind 127.0.0.1配置改成自己的IP地址。

防火墙端口也放开下

在客户端可以用info replication查看信息

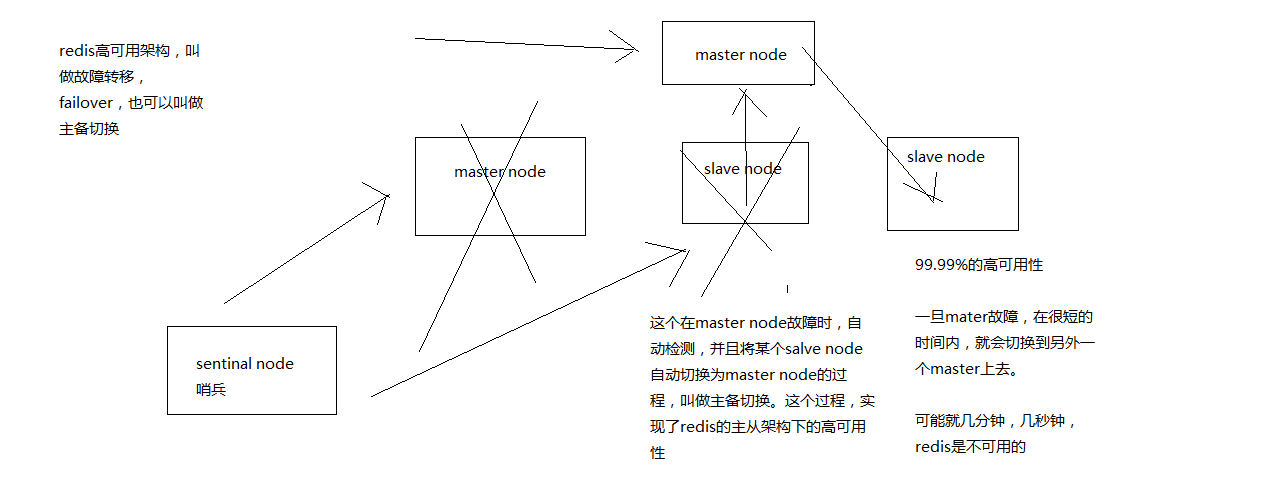
* 1. 压测

redis自己提供的redis-benchmark压测工具，是最快捷最方便的，当然啦，这个工具比较简单，用一些简单的操作和场景去压测

cd redis-3.2.8/src

./redis-benchmark -h 192.168.0.108

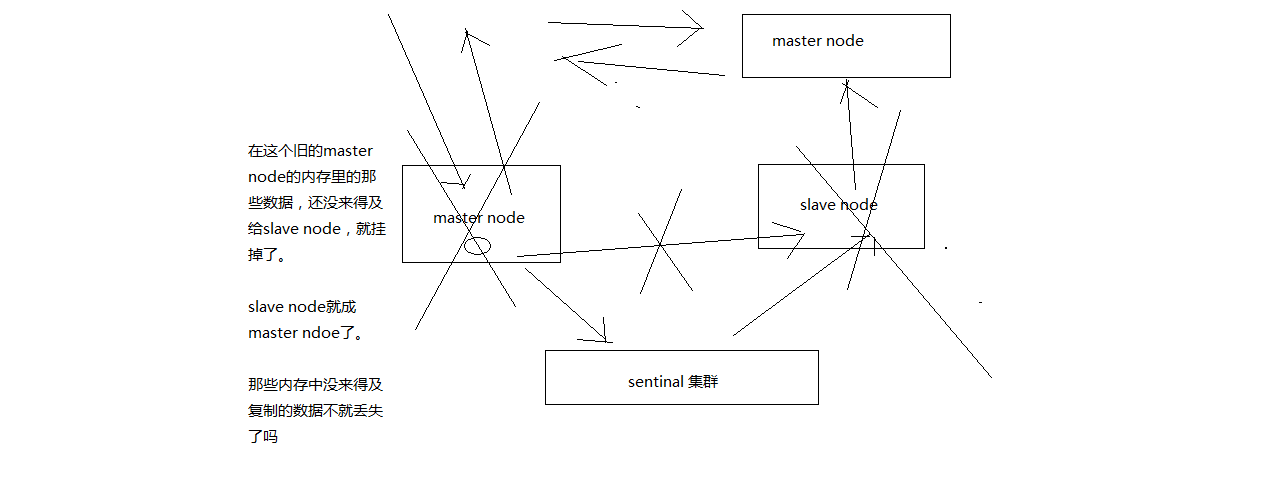
* 1. redis基于哨兵的高可用性





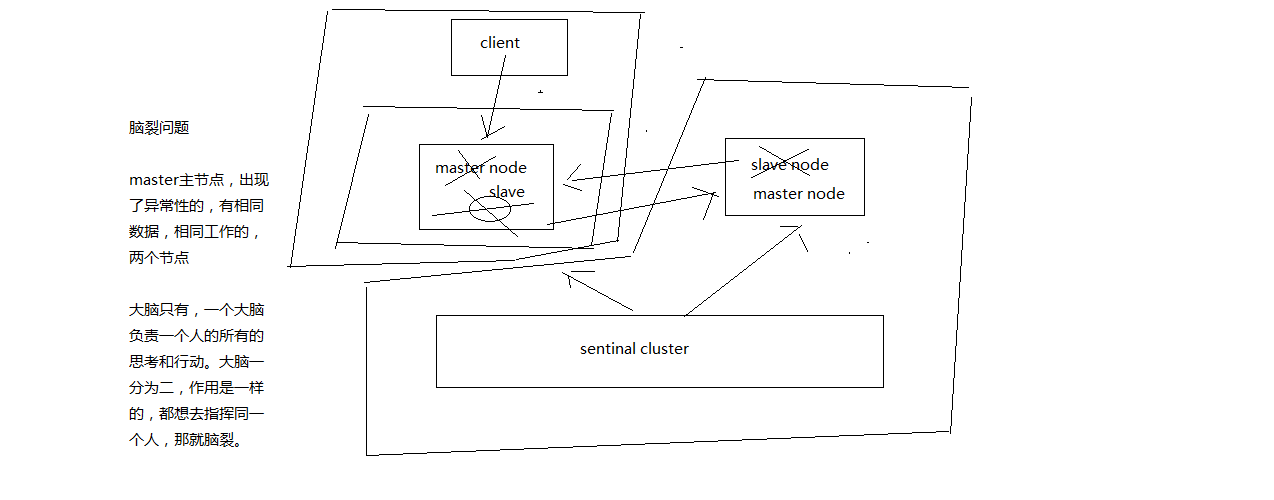
（接上）同时3个哨兵的majority是2，所以还剩下的2个哨兵运行着，就可以允许执行故障转移

* 1. 异步复制



* 1. 集群脑裂

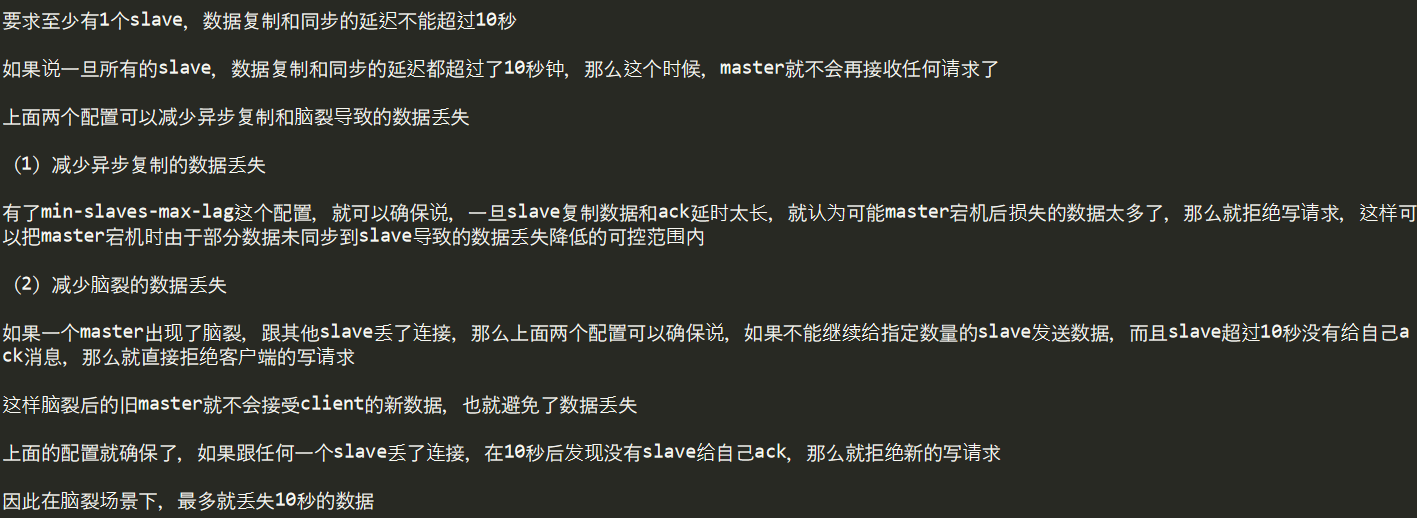
由于网络故障，出现了网络分区，右边的认为左边的已经不能访问了，然后slave就被提拔成master了，这时候就出现了两个master了。本来client是访问左边的，然后出现了脑裂，这时网络故障已经修复，左边的master变成slave 了。这时候右边的会复制数据给左边。从开始脑裂到恢复网络并正常运行这段期间的数据丢失了。

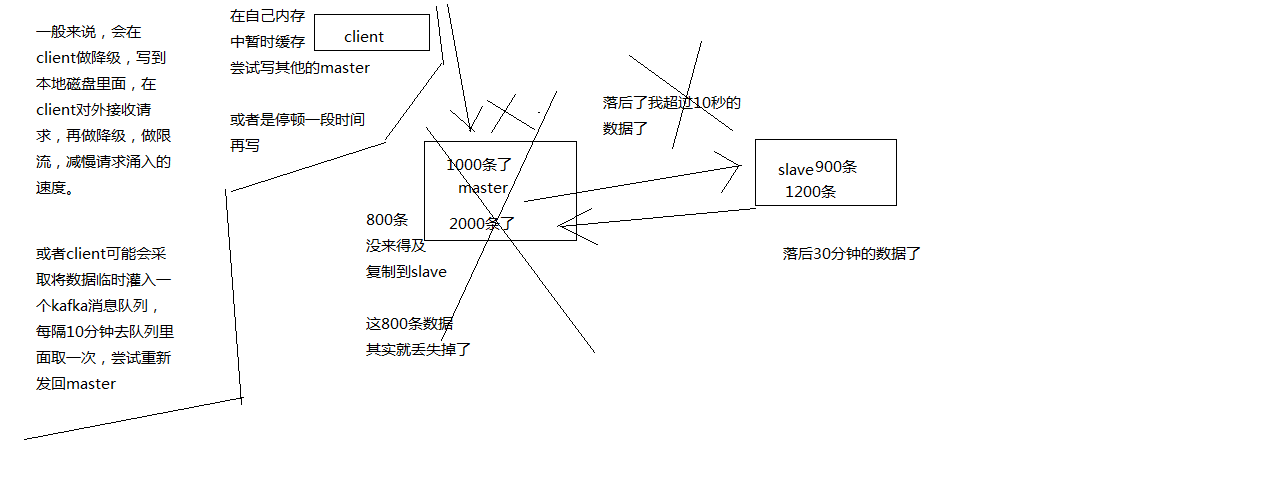


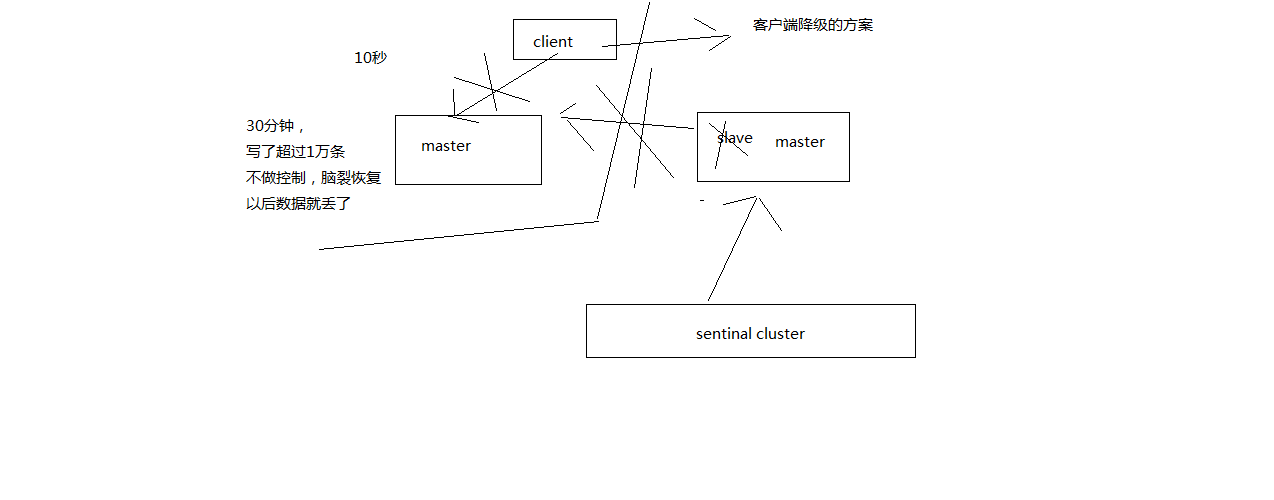
* 1. 降低异步复制和集群脑裂数据损失

min-slaves-to-write 1

min-slaves-max-lag 10







* 1. [哨兵原理](redis哨兵的多个核心底层原理的深入解析（包含slave选举算法）.txt)
  2. 哨兵的配置文件

哨兵默认用26379端口，默认不能跟其他机器在指定端口连通，只能在本地访问

mkdir /etc/sentinal

mkdir -p /var/sentinal/5000

vim /etc/sentinel/5000.conf

daemonize yes

port 5000

bind 192.168.0.108

dir /var/sentinal/5000

sentinel monitor mymaster 192.168.0.108 6379 2 #这里的2是quorum

sentinel down-after-milliseconds mymaster 30000#这里是说超过30s就认为master down了

sentinel failover-timeout mymaster 60000

sentinel parallel-syncs mymaster 1

如果需要日志的话

logfile /var/log/sentinal/5000

mkdir -p /var/log/sentinal/5000

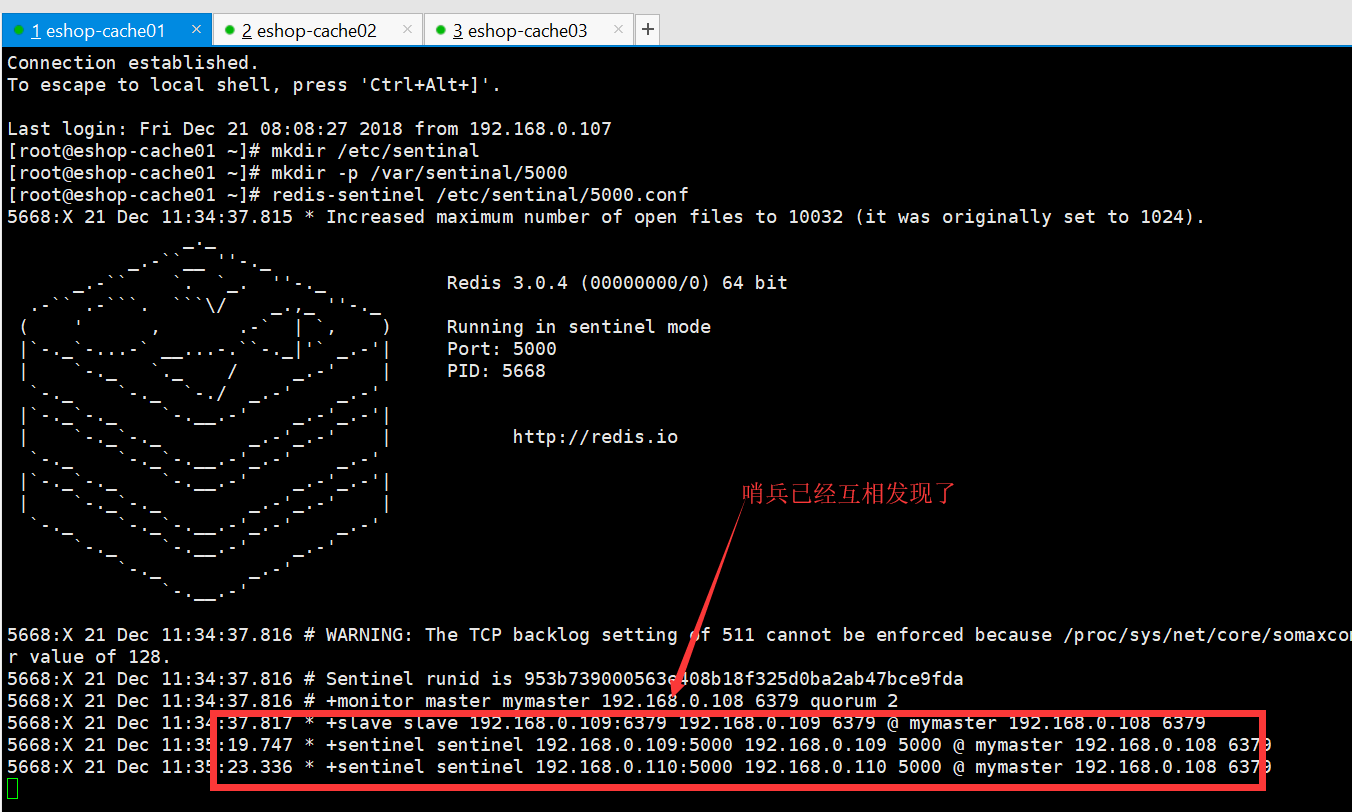
另外 两台也这样配置，只要改bind就行了。

启动哨兵进程

在eshop-cache01、eshop-cache02、eshop-cache03三台机器上，分别启动三个哨兵进程，组成一个集群，观察一下日志的输出

redis-sentinel /etc/sentinal/5000.conf

哨兵启动成功后



日志里会显示出来，每个哨兵都能去监控到对应的redis master，并能够自动发现对应的slave。哨兵之间，互相会自动进行发现，用的就是之前说的pub/sub，消息发布和订阅channel消息系统和机制

检查哨兵状态

redis-cli -h 192.168.0.108 -p 5000

sentinel master mymaster

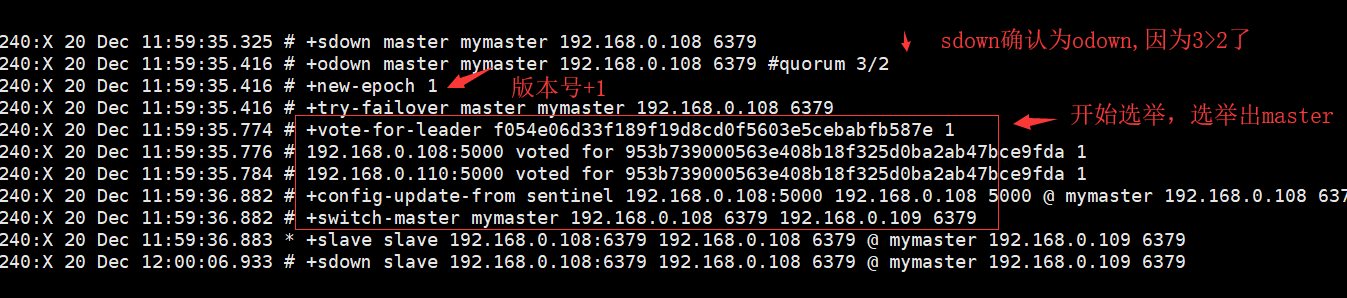
SENTINEL slaves mymaster

SENTINEL sentinels mymaster

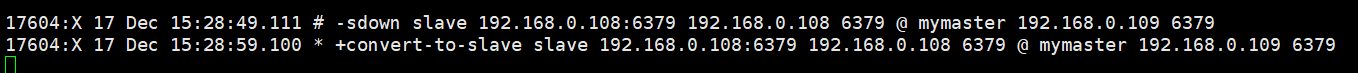
SENTINEL get-master-addr-by-name mymaster

* 1. 哨兵容灾演练

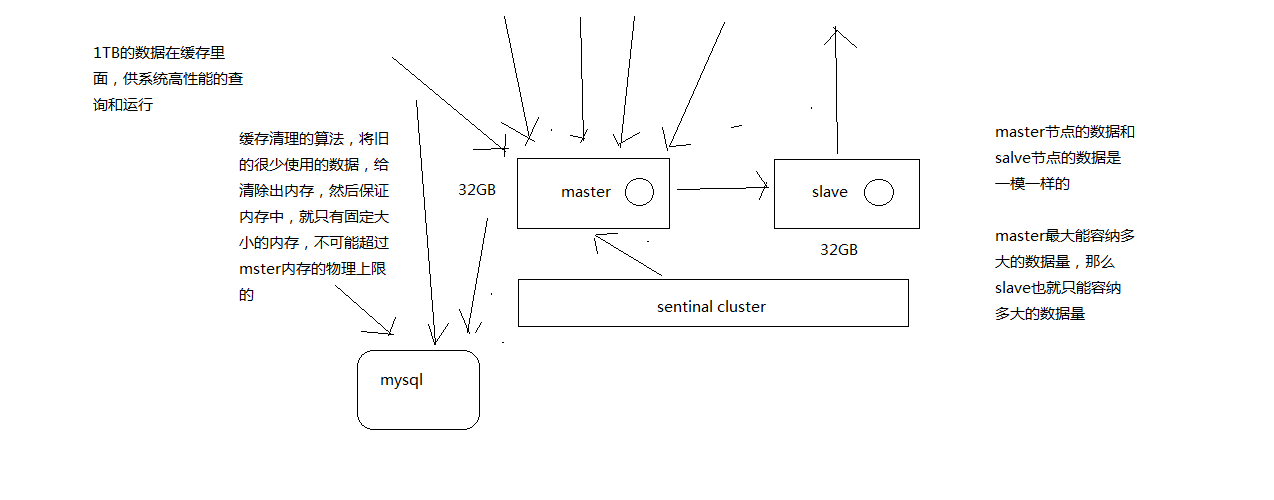
先停掉master,这时候主备切换

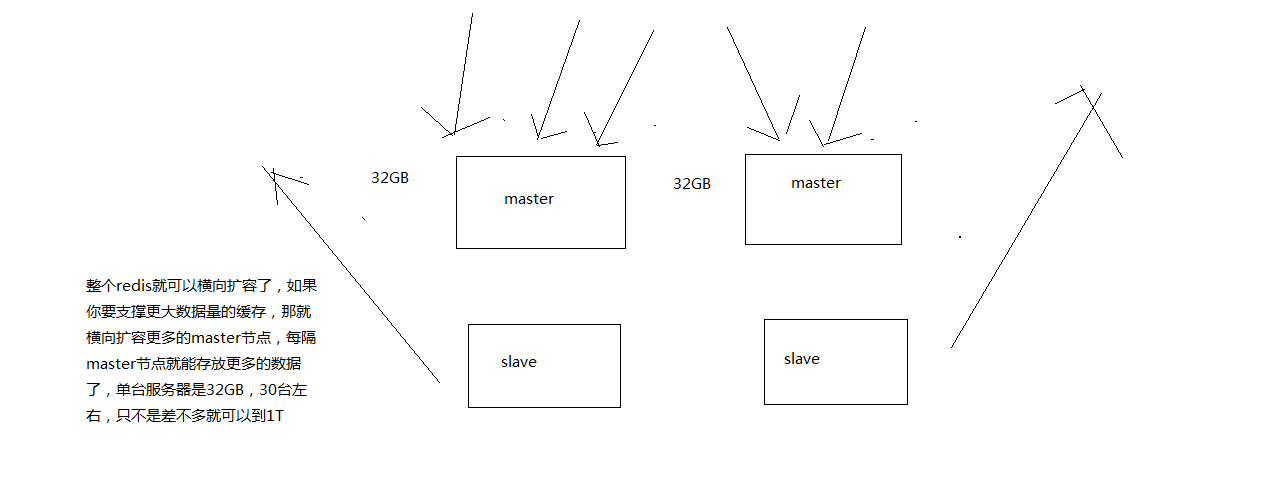


然后停掉的那台恢复回来。



1. 集群
   1. 为什么要集群





redis cluster（多master + 读写分离 + 高可用）

我们只要基于redis cluster去搭建redis集群即可，不需要手工去搭建replication复制+主从架构+读写分离+哨兵集群+高可用

**redis cluster vs. replication + sentinal**

**如果你的数据量很少，主要是承载高并发高性能的场景，比如你的缓存一般就几个G，单机足够了**

**replication，一个mater，多个slave，要几个slave跟你的要求的读吞吐量有关系，然后自己搭建一个sentinal集群，去保证redis主从架构的高可用性，就可以了**

**redis cluster，主要是针对海量数据+高并发+高可用的场景，海量数据，如果你的数据量很大，那么建议就用redis cluster**

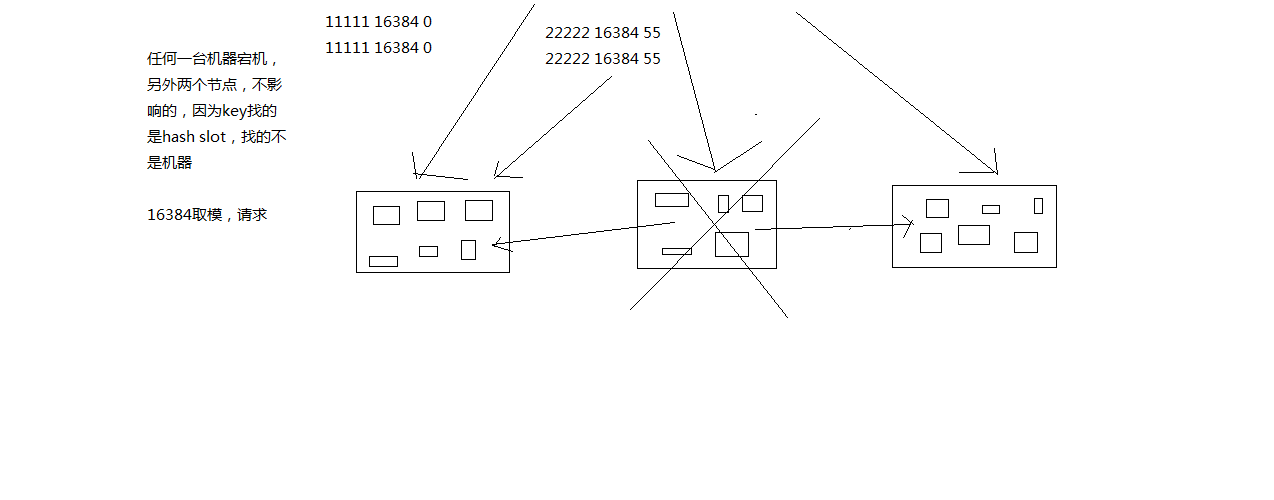
* 1. hash slot算法

redis cluster有固定的16384个hash slot，对每个key计算CRC16值，然后对16384取模，可以获取key对应的hash slot

redis cluster中每个master都会持有部分slot，比如有3个master，那么可能每个master持有5000多个hash slot

hash slot让node的增加和移除很简单，增加一个master，就将其他master的hash slot移动部分过去，减少一个master，就将它的hash slot移动到其他master上去

移动hash slot的成本是非常低的

客户端的api，可以对指定的数据，让他们走同一个hash slot，通过hash tag来实现

* 1. 配置
     1. mkdir -p /etc/redis-cluster mkdir -p /var/log/redis mkdir -p /etc/redis
     2. mkdir -p /var/redis/7001 放持久化文件
     3. 配置文件修改

port 7001

cluster-enabled yes

cluster-config-file /etc/redis-cluster/node-7001.conf

cluster-node-timeout 15000

daemonize yes

pidfile /var/run/redis\_7001.pid

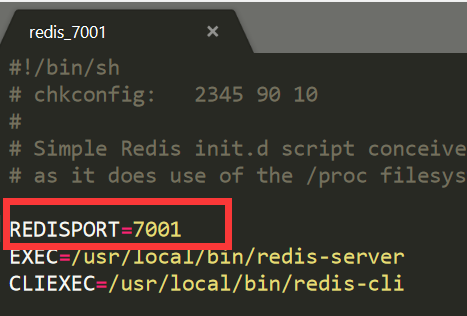
dir /var/redis/7001

logfile /var/log/redis/7001.log

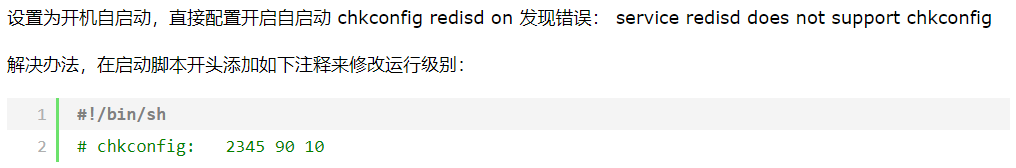
bind 192.168.31.187

appendonly yes

将上面的配置文件，在/etc/redis下放6个，分别为: 7001.conf，7002.conf，7003.conf，7004.conf，7005.conf，7006.conf，端口都改掉



* + 1. 在/etc/init.d下，放6个启动脚本，分别为: redis\_7001, redis\_7002, redis\_7003, redis\_7004, redis\_7005, redis\_7006 每个启动脚本内，都修改对应的端口号
    2. 让redis跟随系统启动自动启动



在redis\_700X脚本中，最上面，加入

# chkconfig: 2345 90 10

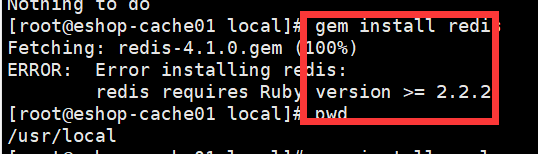
再设置自启动

chkconfig redis\_6379 on

* + 1. 分别在3台机器上，启动6个redis实例
    2. 在一台机器上配置集群即可。用redis-trib.rb命令来配置集群，因为他依赖ruby,所以需要安装ruby
    3. yum install -y ruby yum install -y rubygems gem install redis

-------------------

执行gem命令时候出错，解决方法：



 redis requires Ruby version >= 2.2.2的报错，查了资料发现是Centos默认支持ruby到2.0.0，可gem 安装**[redis](http://lib.csdn.net/base/redis" \o "Redis知识库" \t "https://www.cnblogs.com/carryping/p/_blank)**需要最低是2.2.2

解决办法是 先安装rvm，再把ruby版本提升至2.3.3

1.安装curl

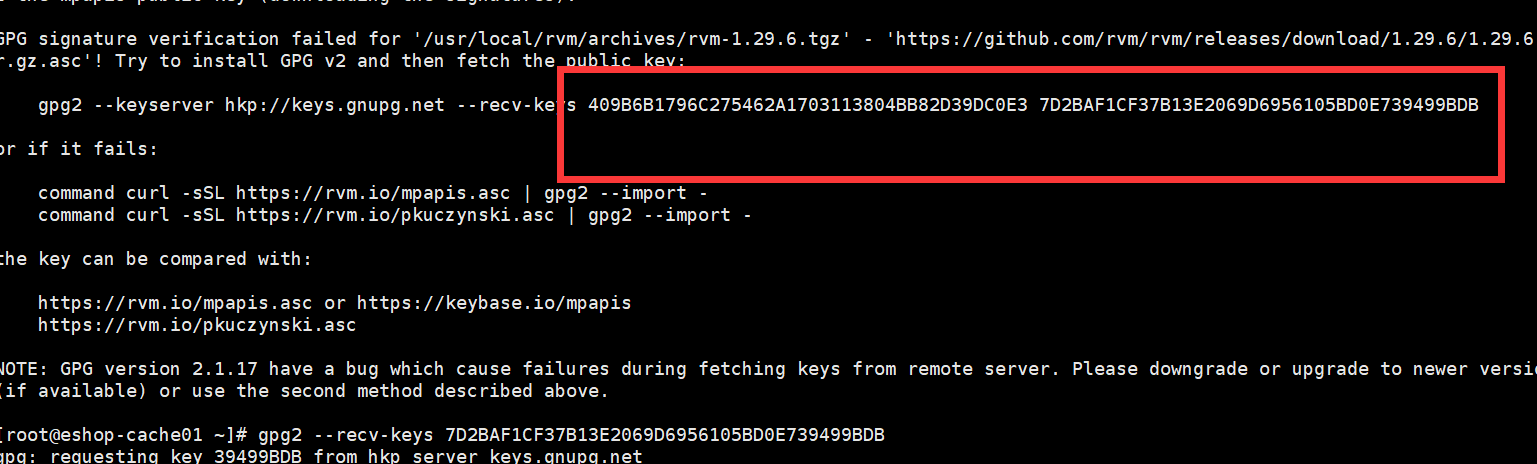
yum install curl

1. 安装RVM

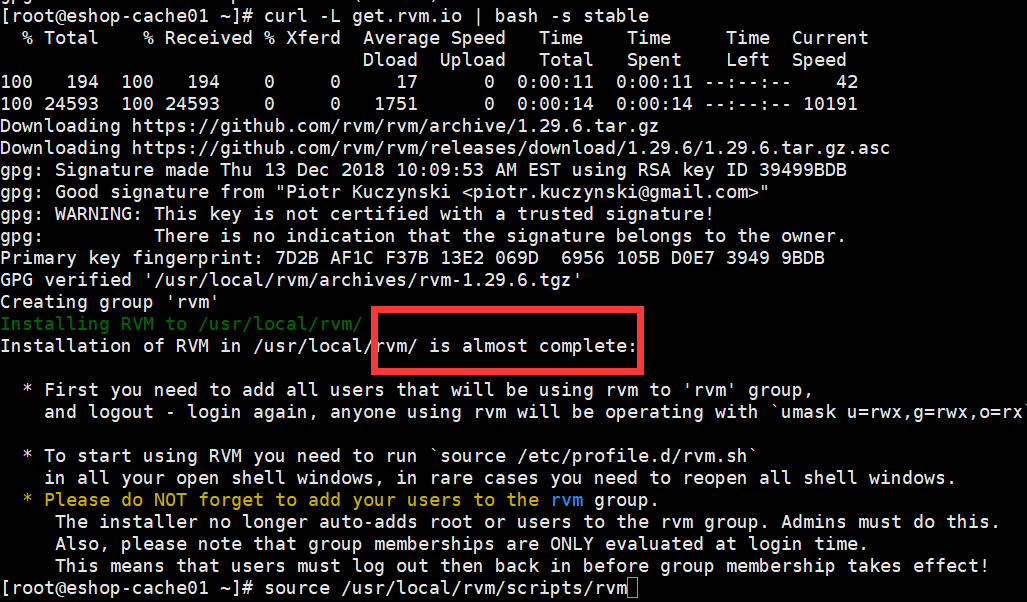
curl -sSL https://rvm.io/mpapis.asc | gpg2 --import -

gpg2 --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3

（注意完成有几个KEY就需要recover几个，比如这里有两个key,则需要再次recv 7D...这个key）



再次执行curl -L get.rvm.io | bash -s stable



这样rvm才算装好

3.

source /usr/local/rvm/scripts/rvm

4. 查看rvm库中已知的ruby版本

rvm list known

5. 安装一个ruby版本

rvm install 2.4.1

6. 使用一个ruby版本

rvm use 2.4.1

7. 设置默认版本

rvm use 2.4.1 --default

9. 再安装redis就可以了

gem install redis

ruby gem安装的redis库，版本不能使用最新的4.0，否则redis-trib.rb reshard 192.168.2.106:8002 重新分片时会报错误。

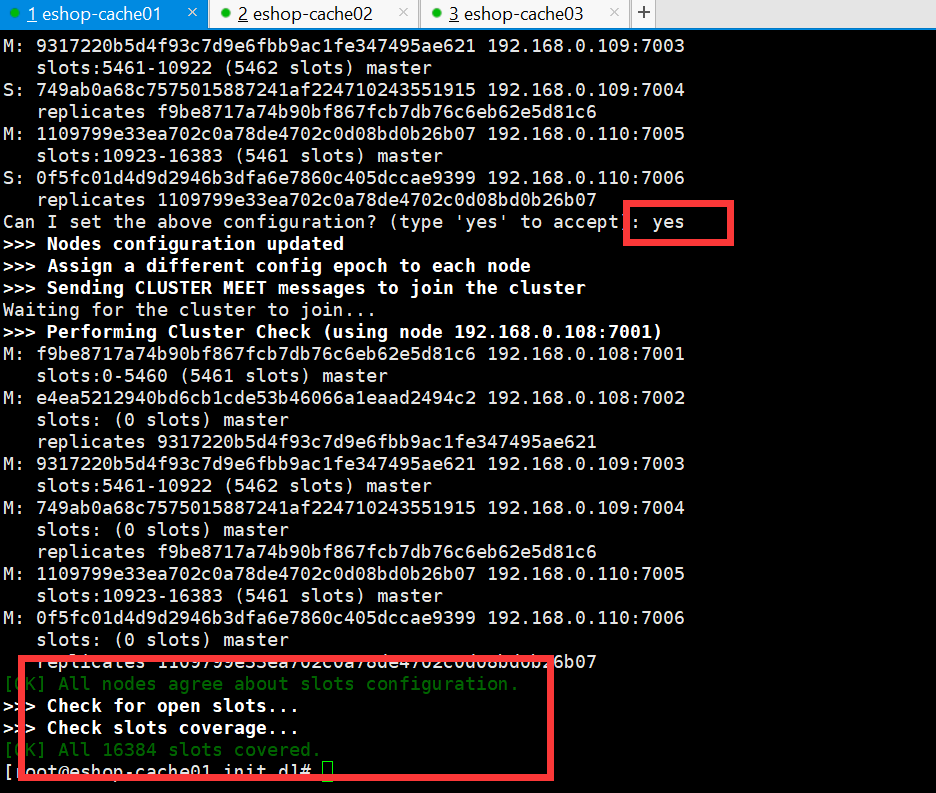
1、卸载最新redis库，gem uninstall redis

2、安装3.x版本，gem install redis -v 3.3.5 测试3.2.1到3.3.5都可以，4.x以上的分片报错

cp /usr/local/redis-3.0.4/src/redis-trib.rb /usr/local/bin

redis-trib.rb create --replicas 1 192.168.0.108:7001 192.168.0.108:7002 192.168.0.109:7003 192.168.0.109:7004 192.168.0.110:7005 192.168.0.110:7006

需要打下yes



成功

可以用check命令查看redis-trib.rb check 192.168.0.108:7001

redis-cli -c启动，就会自动进行各种底层的重定向的操作

如 redis-cli -h 192.168.0.108 -p 7001 -c 会自动重定向。

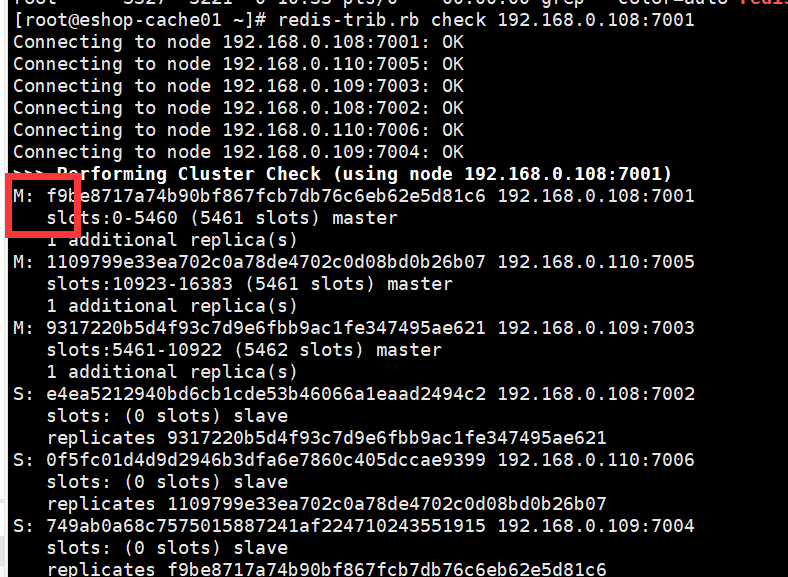
* 1. 集群高可用

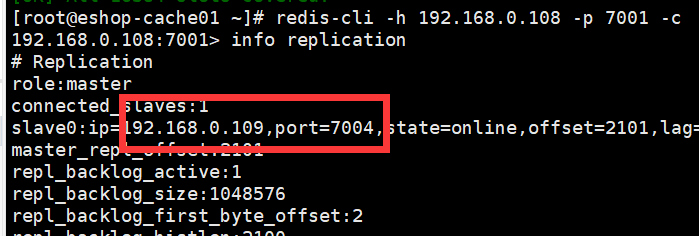
cluster的核心的理念，主要是用slave做高可用的，每个master挂一两个slave，主要是做数据的热备，还有master故障时的主备切换，实现高可用的

redis cluster默认是不支持slave节点读或者写的，跟我们手动基于replication搭建的主从架构不一样的

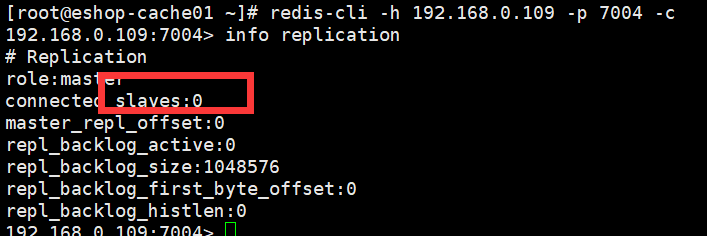
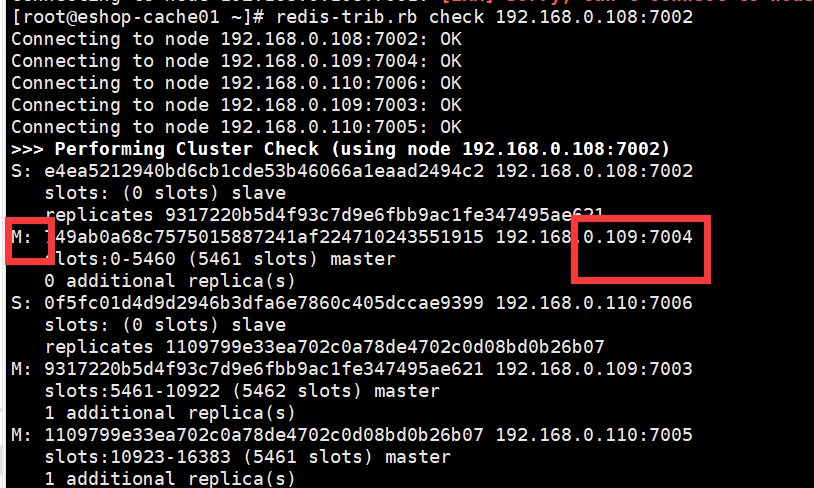
检验高可用：

开始的状态：7001是master挂载着7004



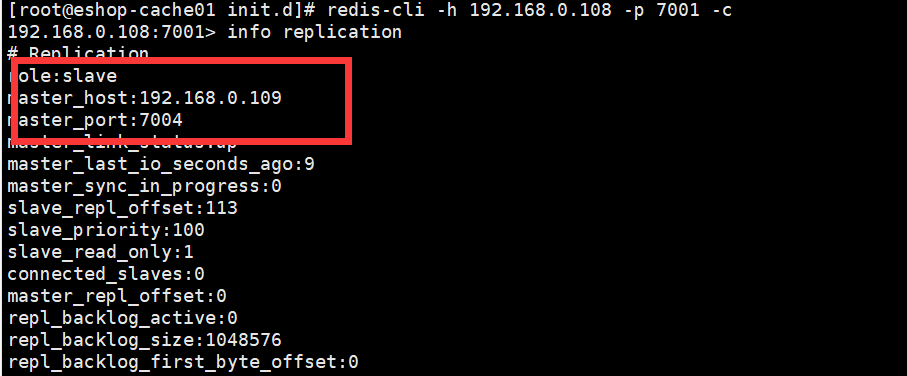
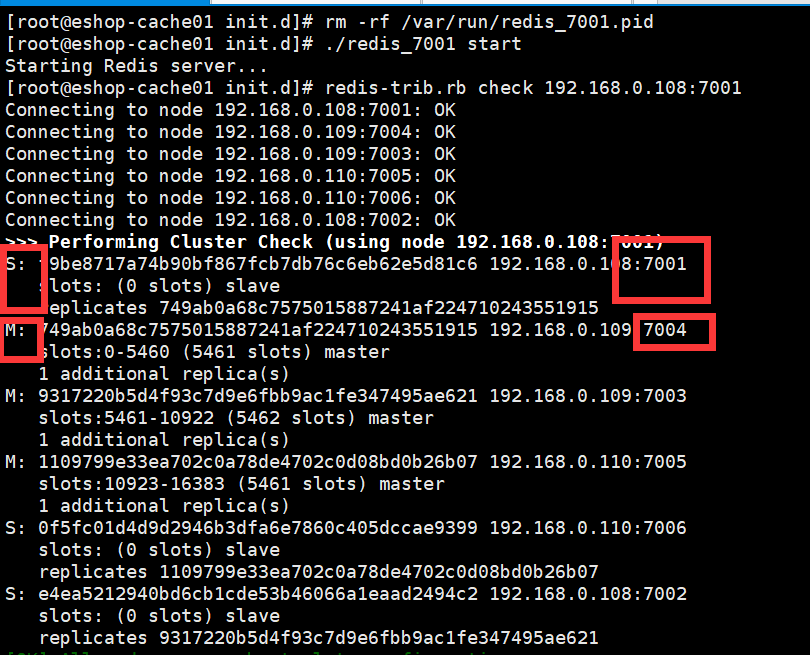


1. 把7001给停掉



7004已经成功取代7001变成master了，这时他是没slave的

1. 把7001恢复



成功挂载到7004这个master上了

* 1. 扩容

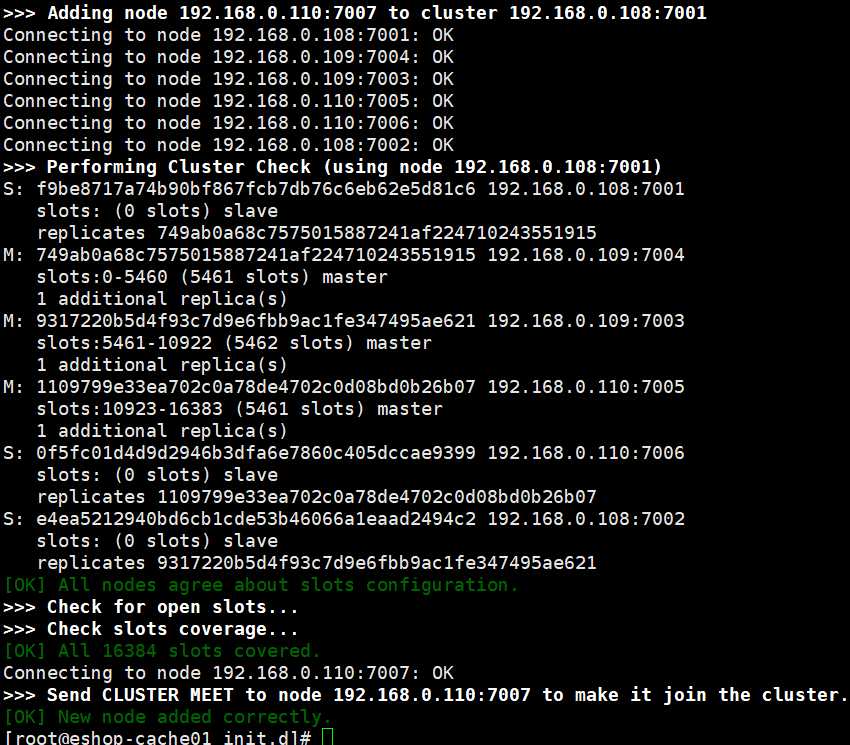
1、加入新master

mkdir -p /var/redis/7007

搞一个7007.conf，再搞一个redis\_7007启动脚本

手动启动一个新的redis实例，在7007端口上

redis-trib.rb add-node 192.168.0.110:7007 192.168.0.108:7001



redis-trib.rb check 192.168.0.108:7001

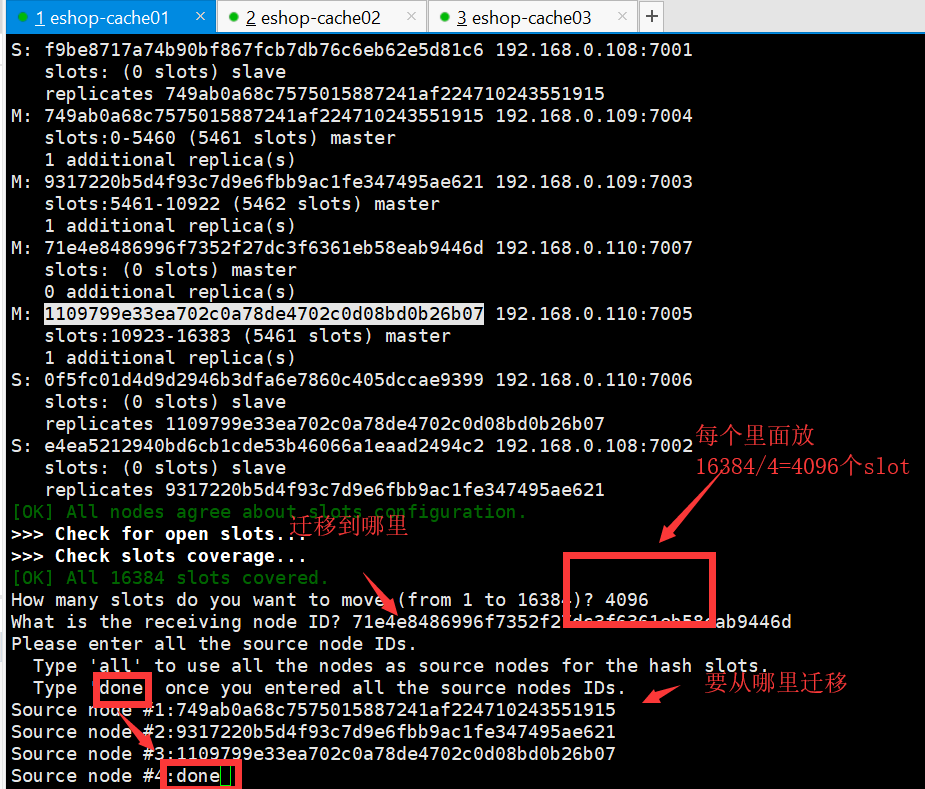
连接到新的redis实例上，cluster nodes，确认自己是否加入了集群，作为了一个新的master

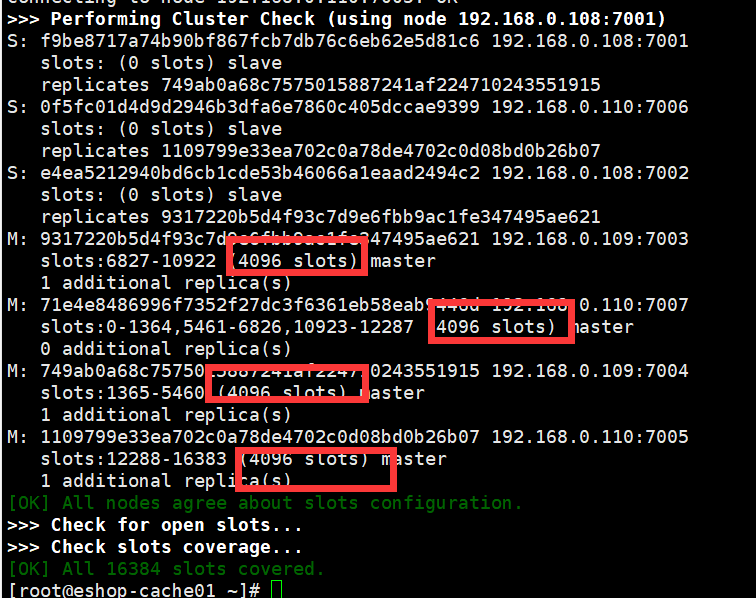
2、reshard一些数据过去

resharding的意思就是把一部分hash slot从一些node上迁移到另外一些node上

redis-trib.rb reshard 192.168.0.108:7001

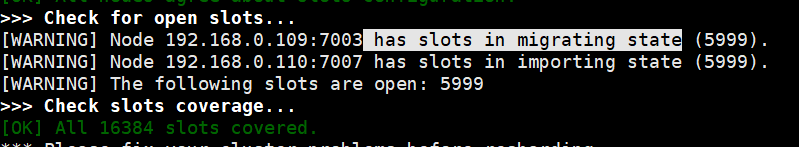
要把之前3个master上，总共4096个hashslot迁移到新的第四个master上去



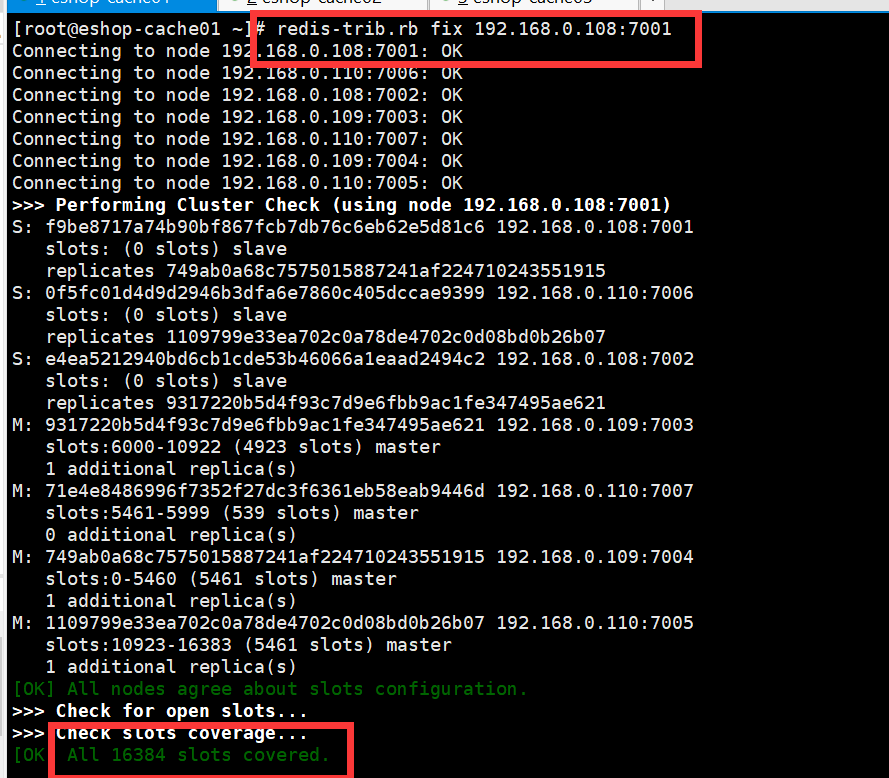


完成。

当分片发生错误时



可以考虑使用命令“redis-trib.rb fix 192.168.0.108:7001”尝试修复，后面根的IP是你resharding时用的IP



3、添加node作为slave

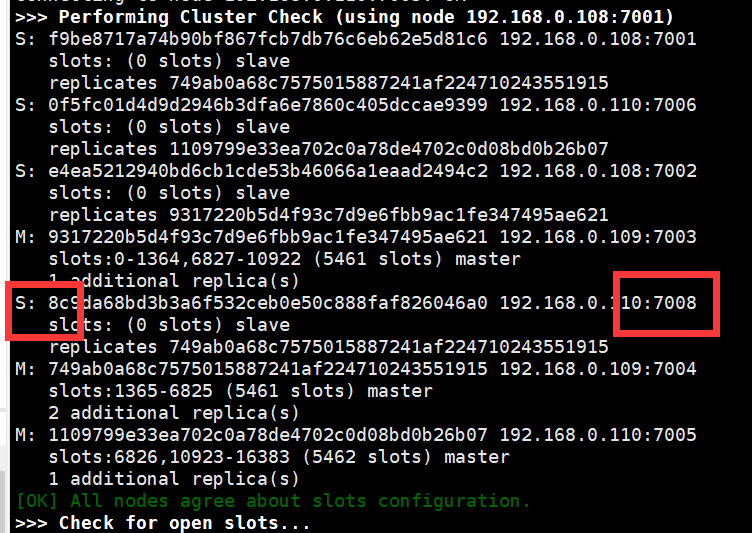
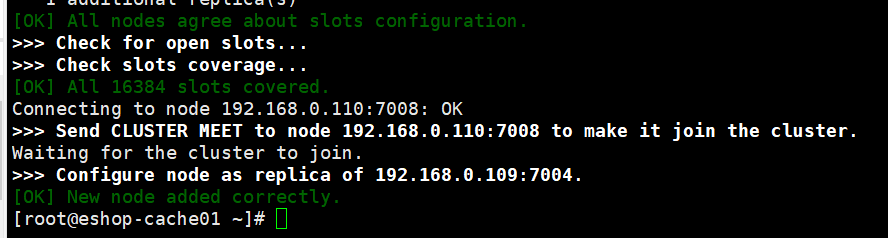
mkdir -p /var/redis/7008

搞一个7008.conf，再搞一个redis\_7008启动脚本

手动启动一个新的redis实例，在7008端口上

redis-trib.rb add-node --slave --master-id f9be8717a74b90bf867fcb7db76c6eb62e5d81c6 192.168.0.110:7008 192.168.0.108:7001

28927912ea0d59f6b790a50cf606602a5ee48108是要挂载到的masterid，必须是master的ID，不能是slave,不然报错,7008是slave地址，7001是说明连到这台机器上去操作



****[ERR] Node 172.168.63.202:7001 is not empty. Either the nodealready knows other nodes (check with CLUSTER NODES) or contains some**** 错误解决方案：

1)、将需要新增的节点下aof、rdb等本地备份文件删除；

2)、同时将新Node的集群配置文件删除,即：删除你redis.conf里面cluster-config-file所在的文件；

3)、再次添加新节点如果还是报错，则登录新Node,./redis-cli–h x –p对数据库进行清除：

172.168.63.201:7001>  flushdb      #清空当前数据库

4、删除node

先用resharding将数据都移除到其他节点，（redis-trib.rb reshard 192.168.0.108:7001

）确保node为空之后，才能执行remove操作

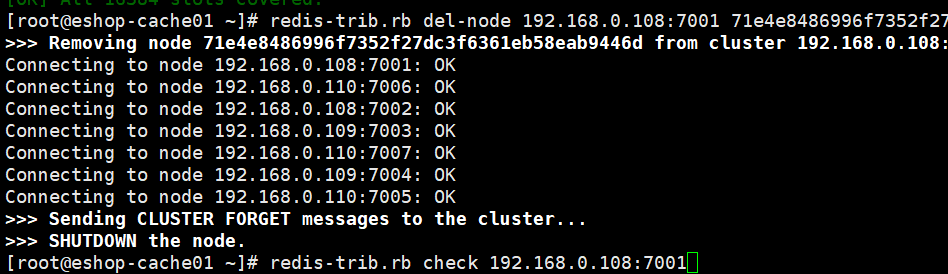
redis-trib.rb del-node 192.168.0.108:7001

bd5a40a6ddccbd46a0f4a2208eb25d2453c2a8db

2个是1365，1个是1366

当你清空了一个master的hashslot时，redis cluster就会自动将其slave挂载到其他master上去

这个时候就只要删除掉master就可以了



* 1. slave冗余

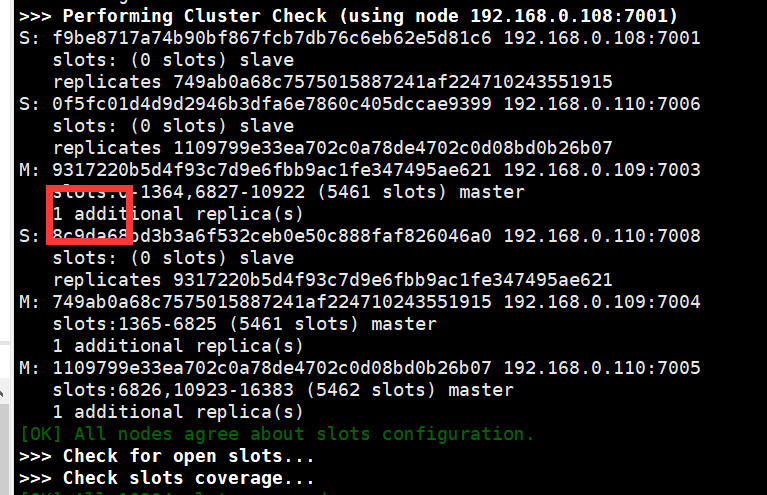
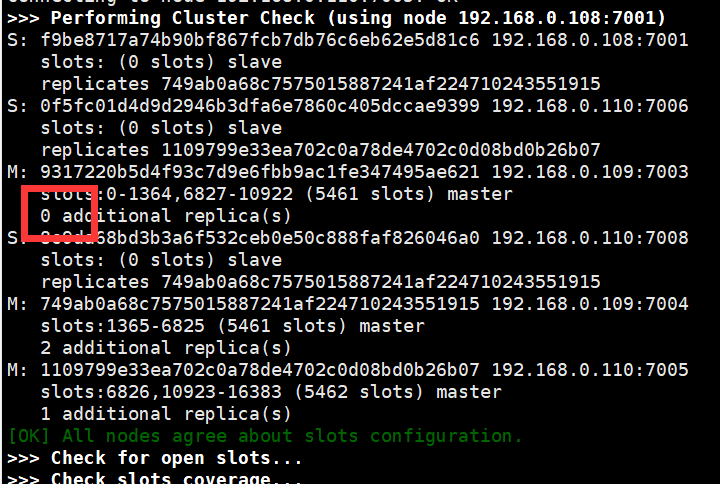
如果你每个master只有一个slave，万一说一个slave死了，然后很快，master也死了，那可用性还是降低了

但是如果你给整个集群挂载了一些冗余slave，那么某个master的slave死了，冗余的slave会被自动迁移过去，作为master的新slave，此时即使那个master也死了

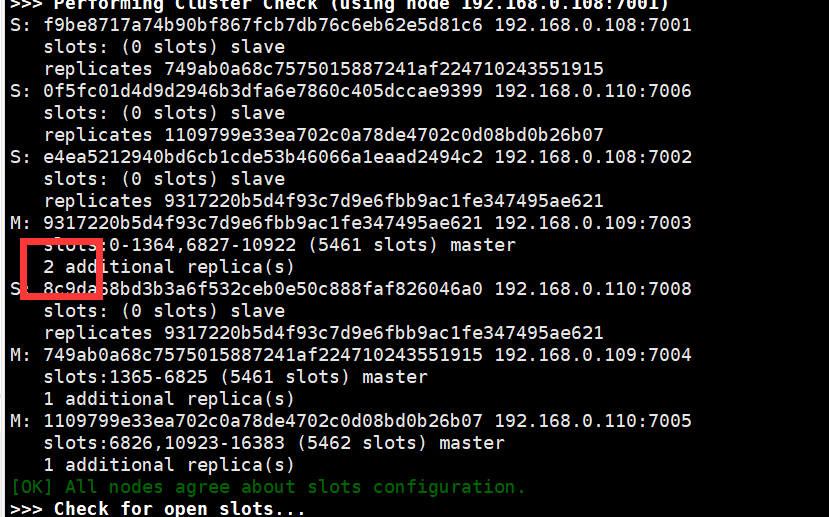
还是有一个slave会切换成master的

之前有一个master是有冗余slave的，直接让其他master其中的一个slave死掉，然后看有冗余slave会不会自动挂载到那个master

演示：当一台slave死了，过一会儿会自动迁移



然后那个slave回来了

又有了2个slave了

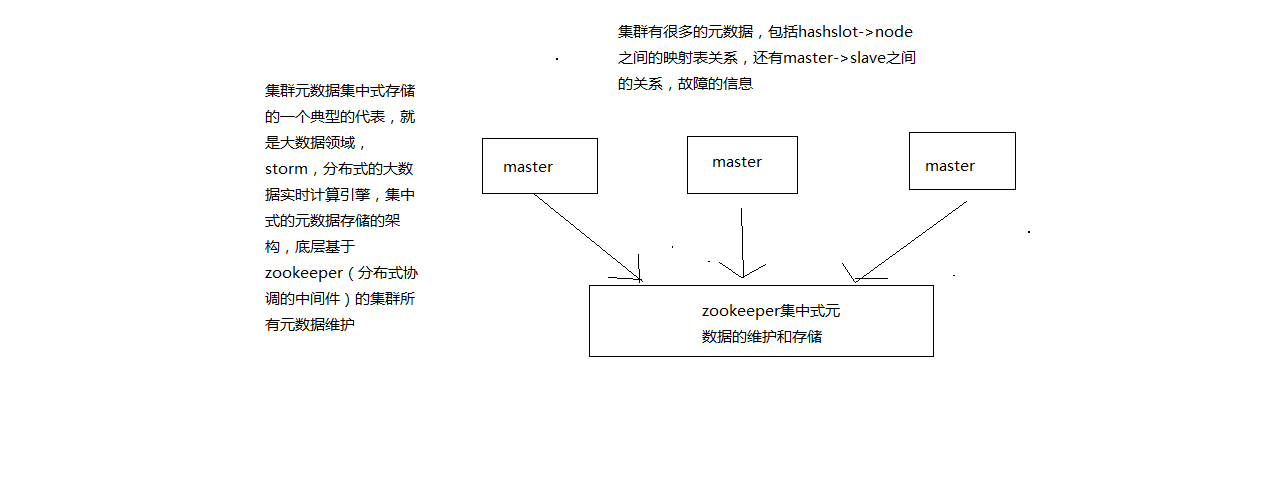
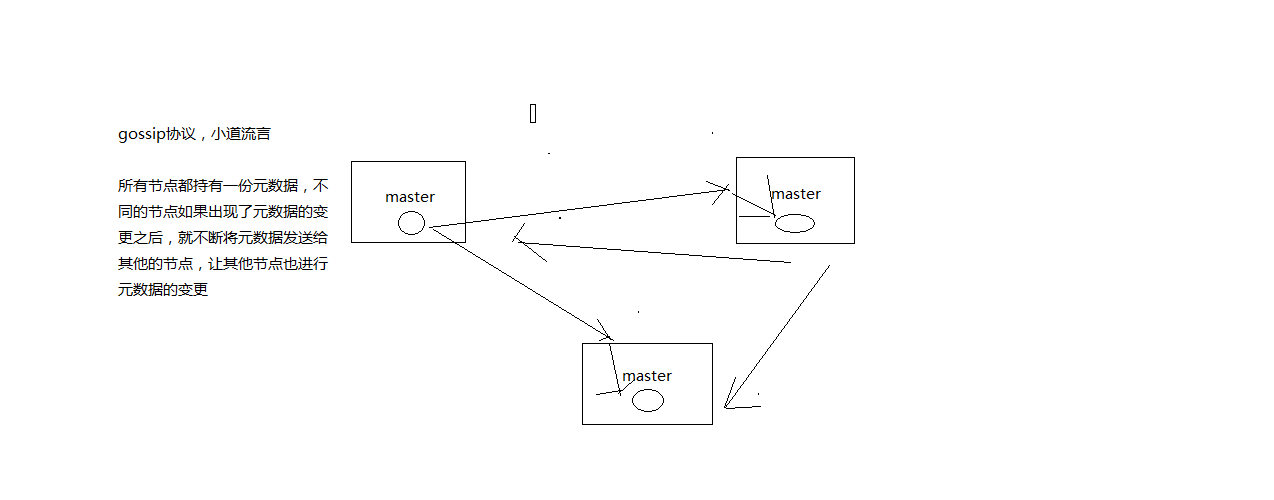
1. redis cluster的核心原理分析：gossip通信、jedis smart定位、主备切换
   1. redis cluster节点间采取gossip协议进行通信

跟集中式不同，不是将集群元数据（节点信息，故障，等等）集中存储在某个节点上，而是互相之间不断通信，保持整个集群所有节点的数据是完整的

维护集群的元数据用得，集中式，一种叫做gossip

集中式：好处在于，元数据的更新和读取，时效性非常好，一旦元数据出现了变更，立即就更新到集中式的存储中，其他节点读取的时候立即就可以感知到; 不好在于，所有的元数据的跟新压力全部集中在一个地方，可能会导致元数据的存储有压力

gossip：好处在于，元数据的更新比较分散，不是集中在一个地方，更新请求会陆陆续续，打到所有节点上去更新，有一定的延时，降低了压力; 缺点，元数据更新有延时，可能导致集群的一些操作会有一些滞后



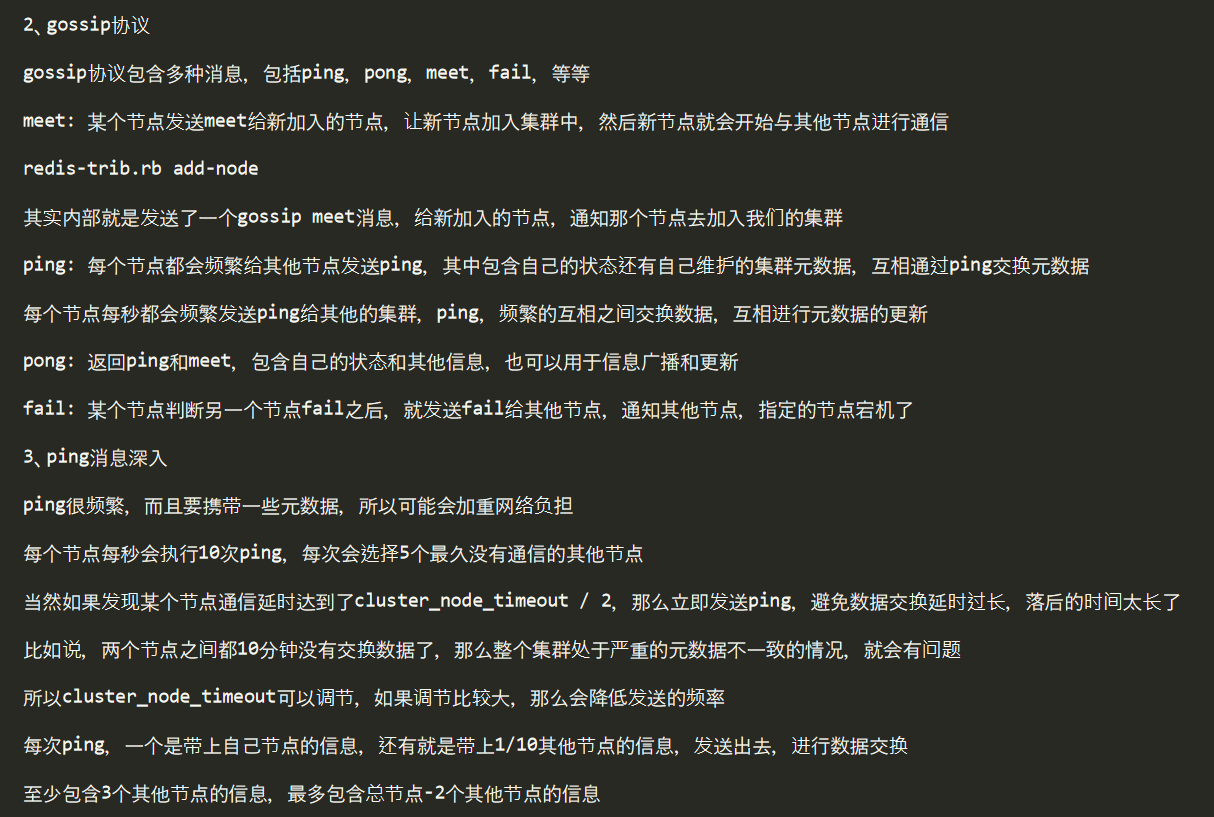
* 1. 10000端口

每个节点都有一个专门用于节点间通信的端口，就是自己提供服务的端口号+10000，比如7001，那么用于节点间通信的就是17001端口

每隔节点每隔一段时间都会往另外几个节点发送ping消息，同时其他几点接收到ping之后返回pong

* 1. 交换的信息

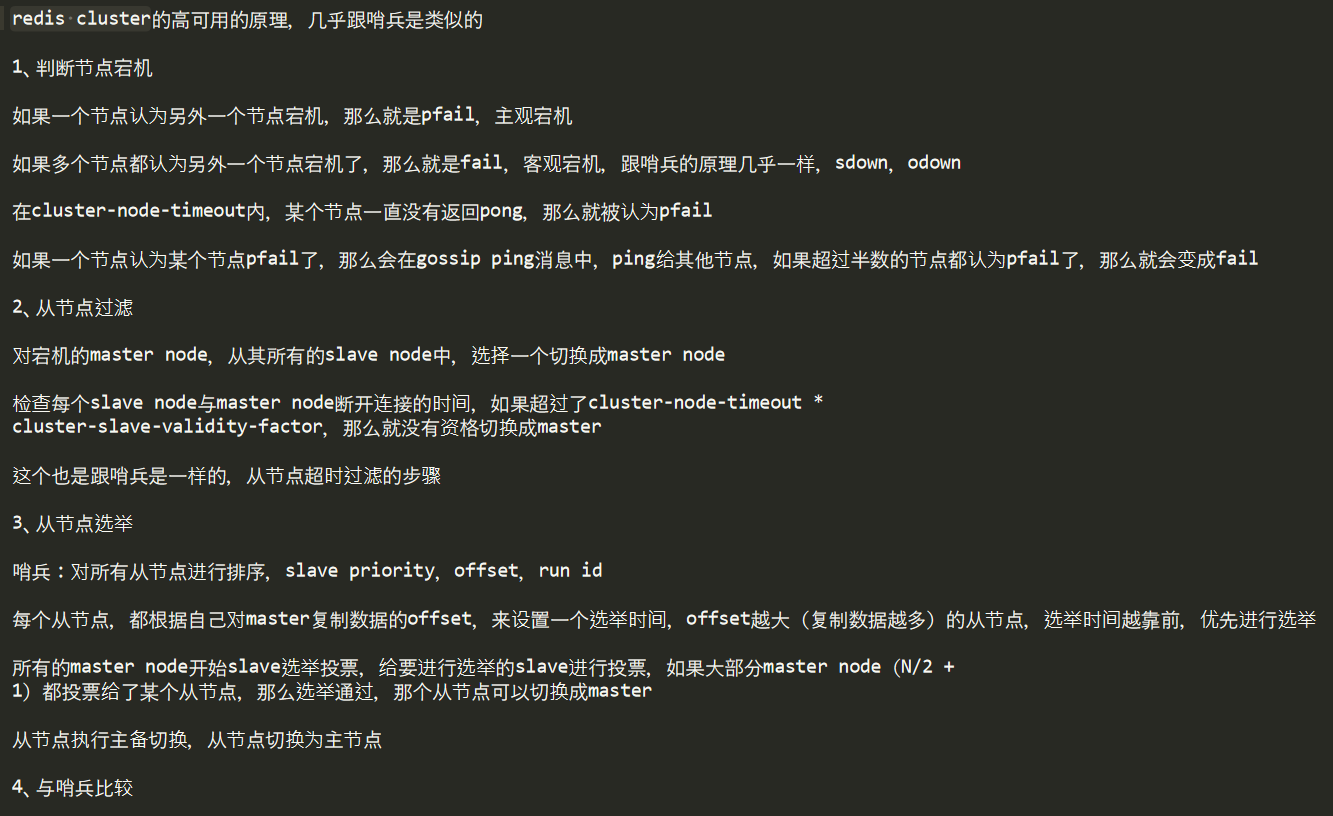
故障信息，节点的增加和移除，hash slot信息，等等



1. smart jedis

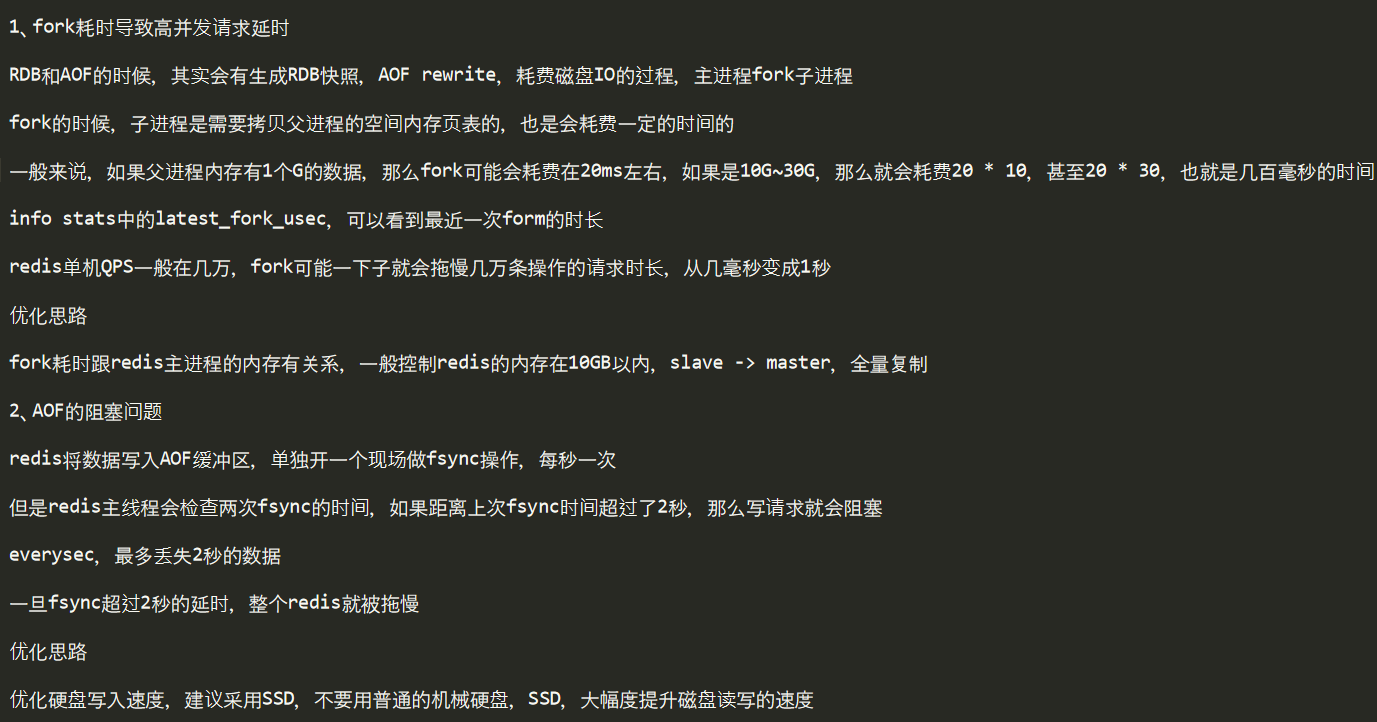


1. redis cluster高可用性与主备切换原理



IMG_256

1. .redis在实践中的一些常见问题以及优化思路（包含linux内核参数优化）







1. [总结](redis总结.docx)