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

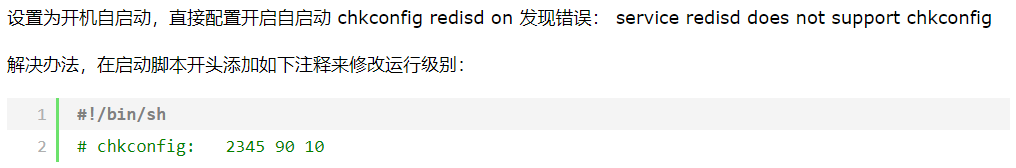
port 6379 设置redis的监听端口号

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



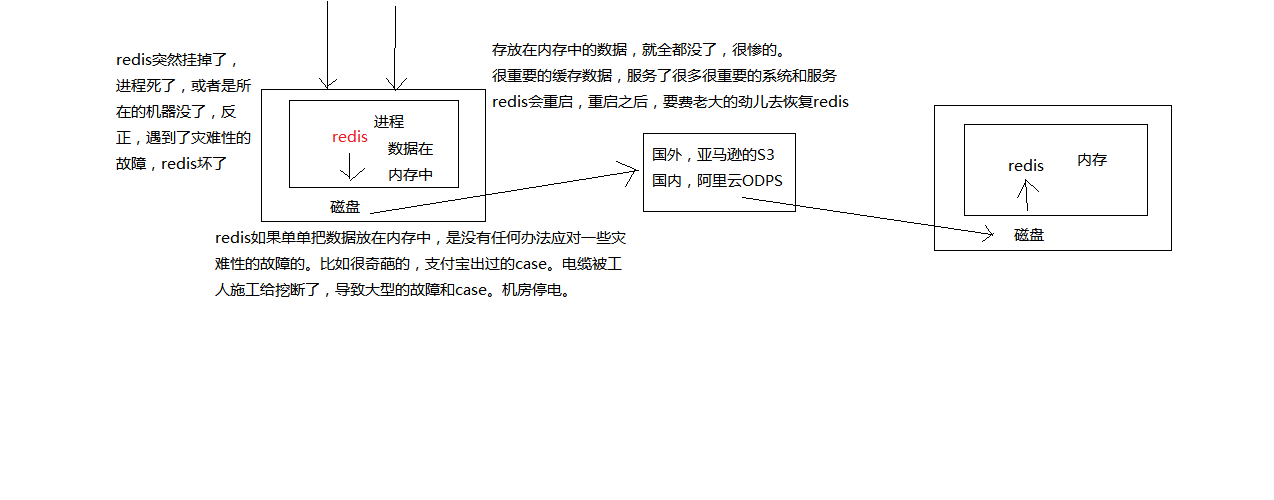
在redis\_6379脚本中，最上面，加入

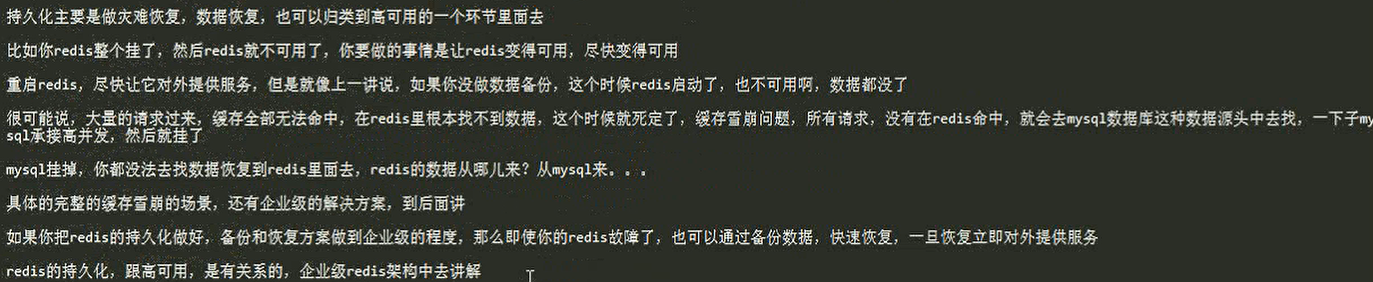
# chkconfig: 2345 90 10

再设置自启动

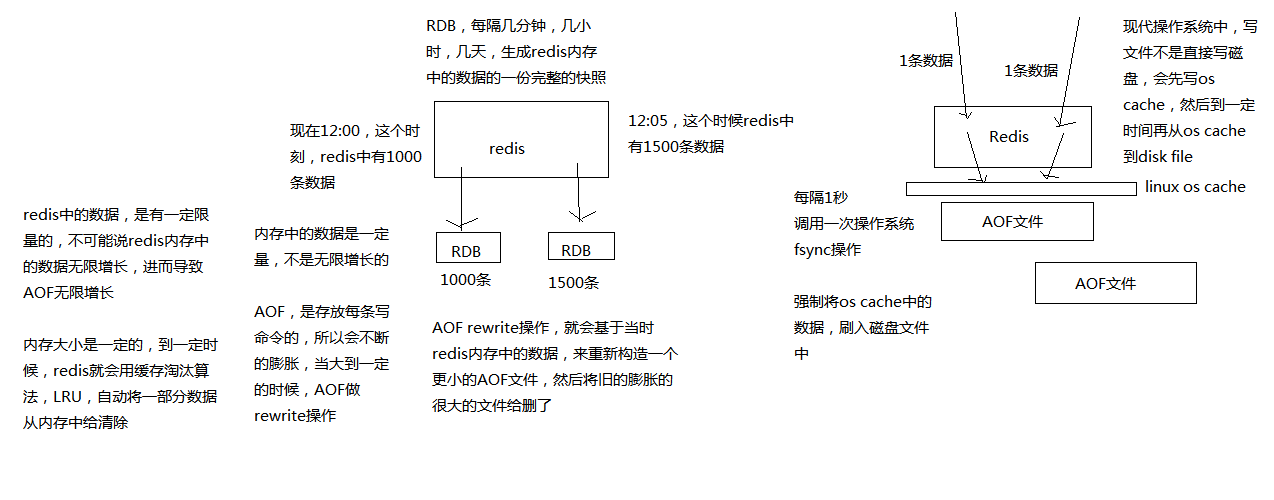
chkconfig redis\_6379 on

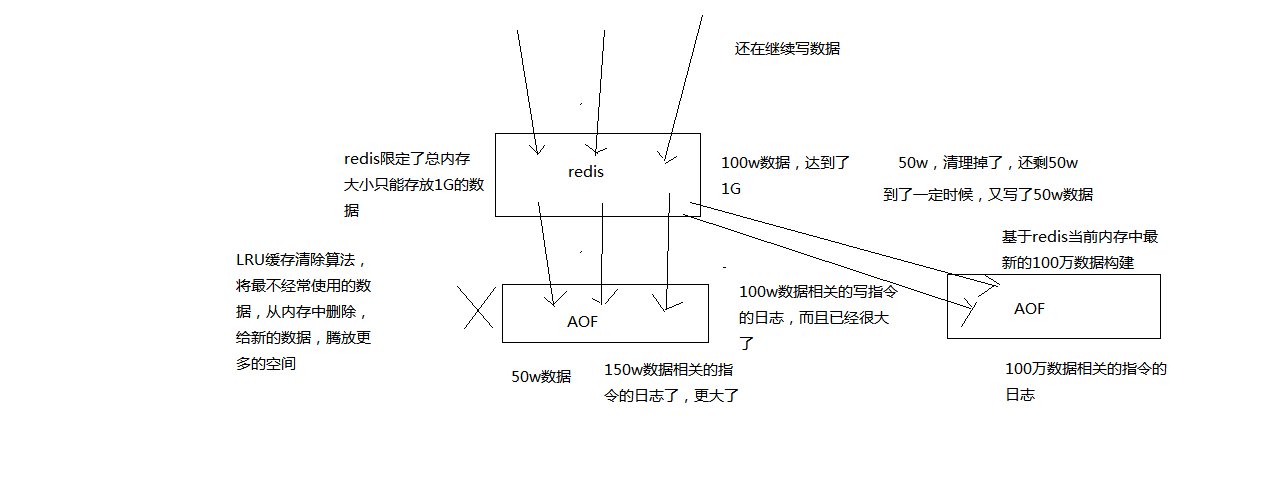
1. 装好redis就直接在/usr/bin下面有命令了，不管在什么目录下，都可以直接redis-cli了。
2. redis持久化
   1. 意义





* 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

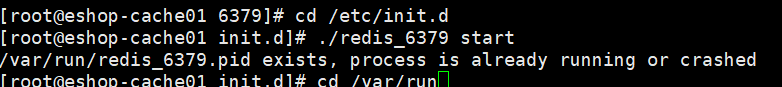
数据还在，为什么？

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

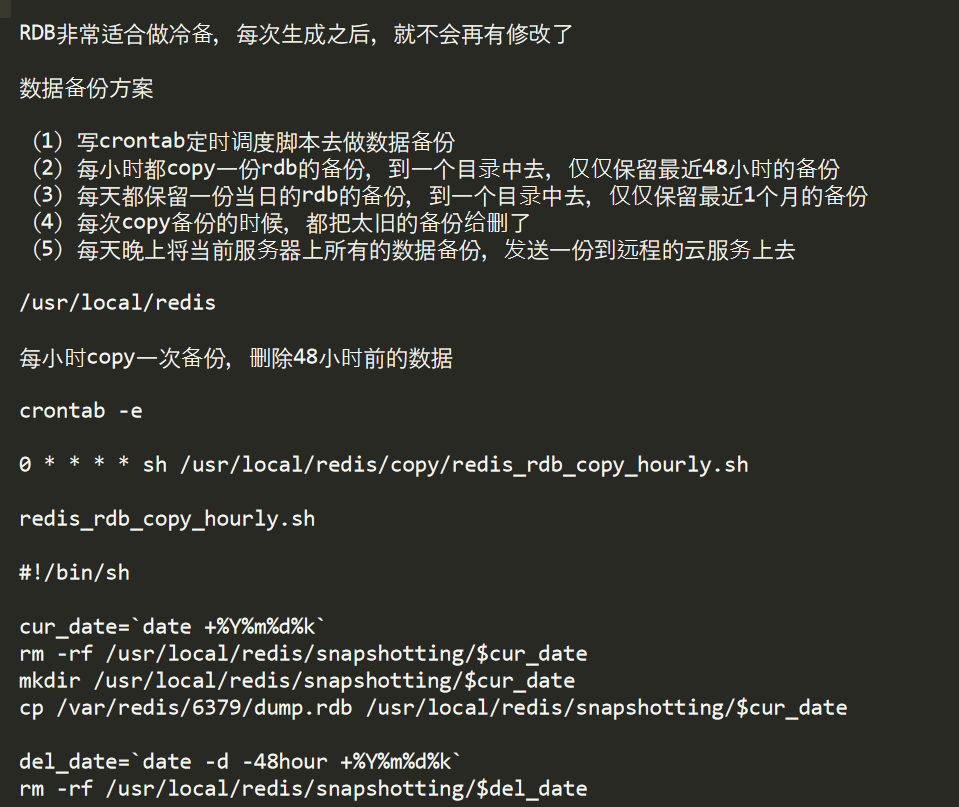
（2）在redis中再保存几条新的数据，用kill -9粗暴杀死redis进程，模拟redis故障异常退出，导致内存数据丢失的场景

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

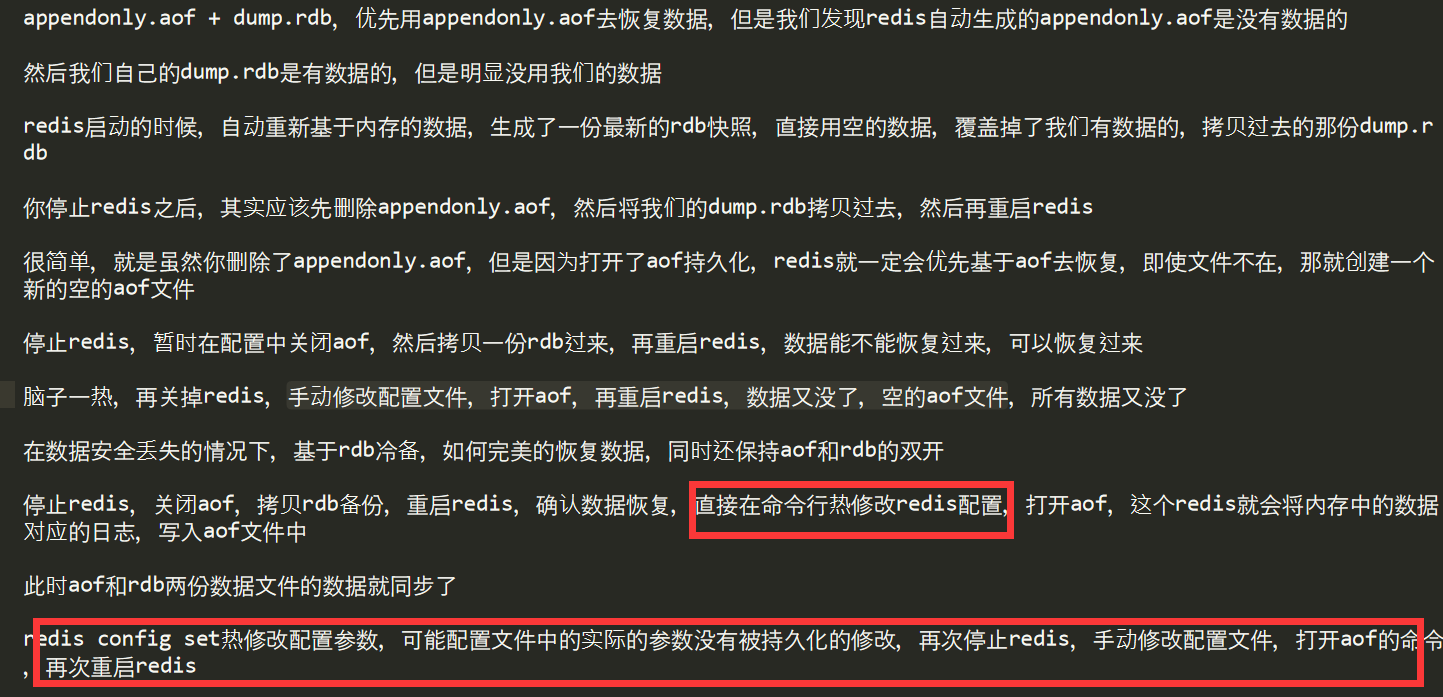
）

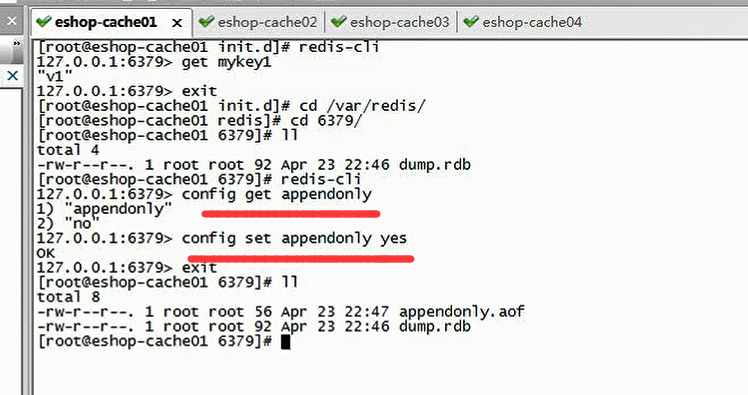
* 1. 手动设置一个save检查点，save 5 1。写入几条数据，等待5秒钟，会发现自动进行了一次dump rdb快照，在dump.rdb中发现了数据。异常停掉redis进程，再重新启动redis，看刚才插入的数据还在
  2. 企业级备份方案
     1. RDB非常适合冷备



* + 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





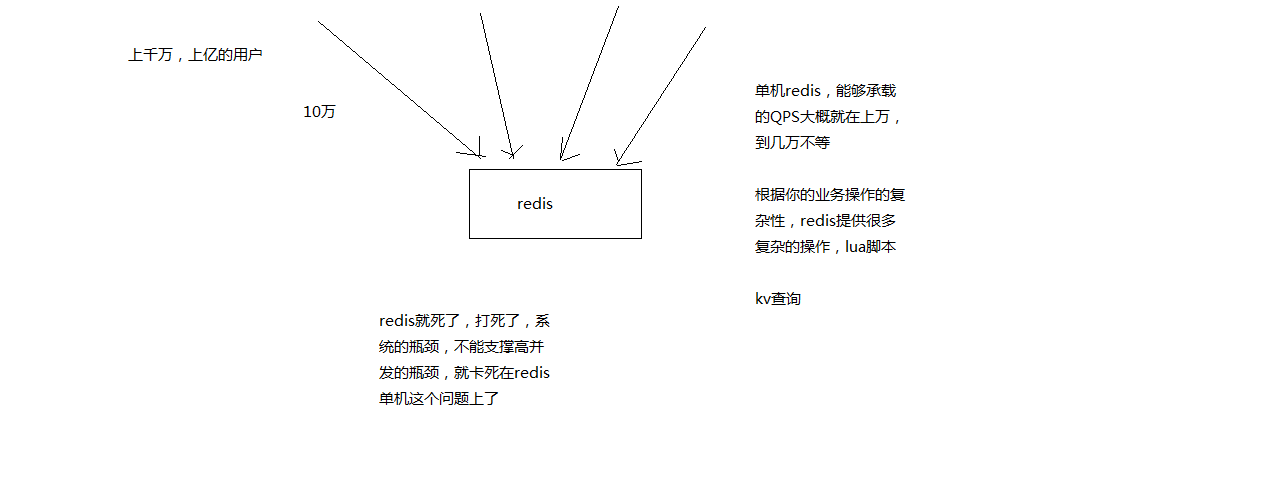
* + 1. 如果当前机器上的所有RDB文件全部损坏，那么从远程的云服务上拉取最新的RDB快照回来恢复数据
    2. 如果是发现有重大的数据错误，比如某个小时上线的程序一下子将数据全部污染了，数据全错了，那么可以选择某个更早的时间点，对数据进行恢复

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

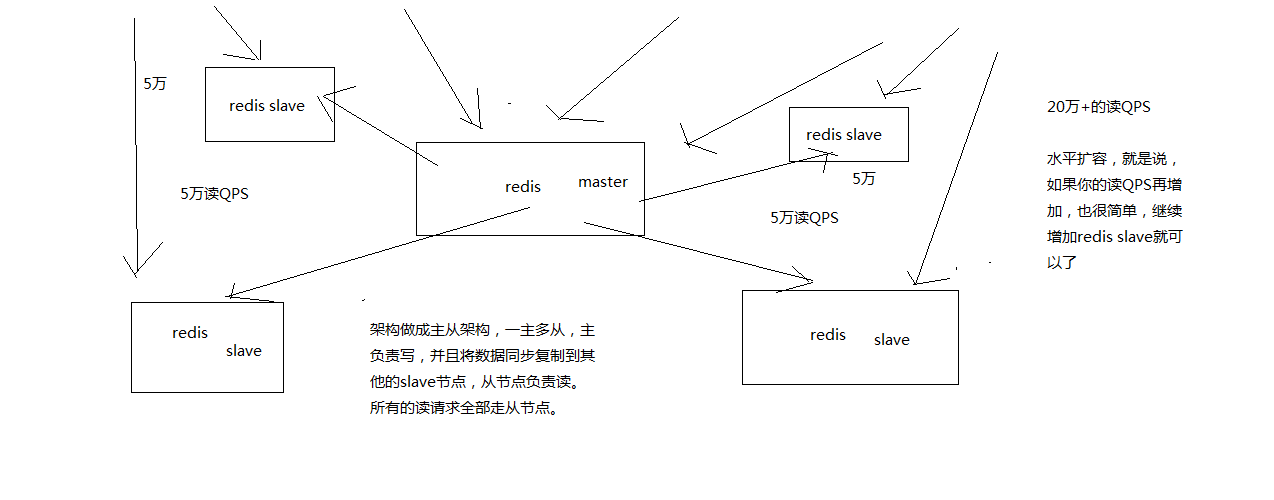
1. 主从
   1. redis不能支撑高并发的瓶颈在哪里？

单机

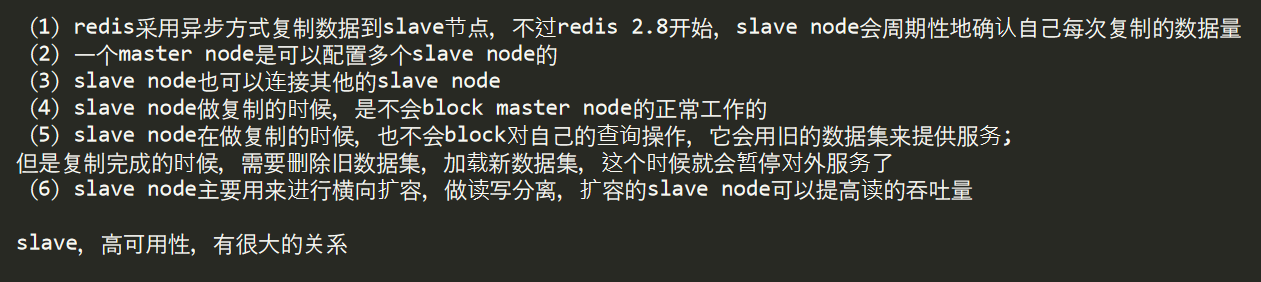


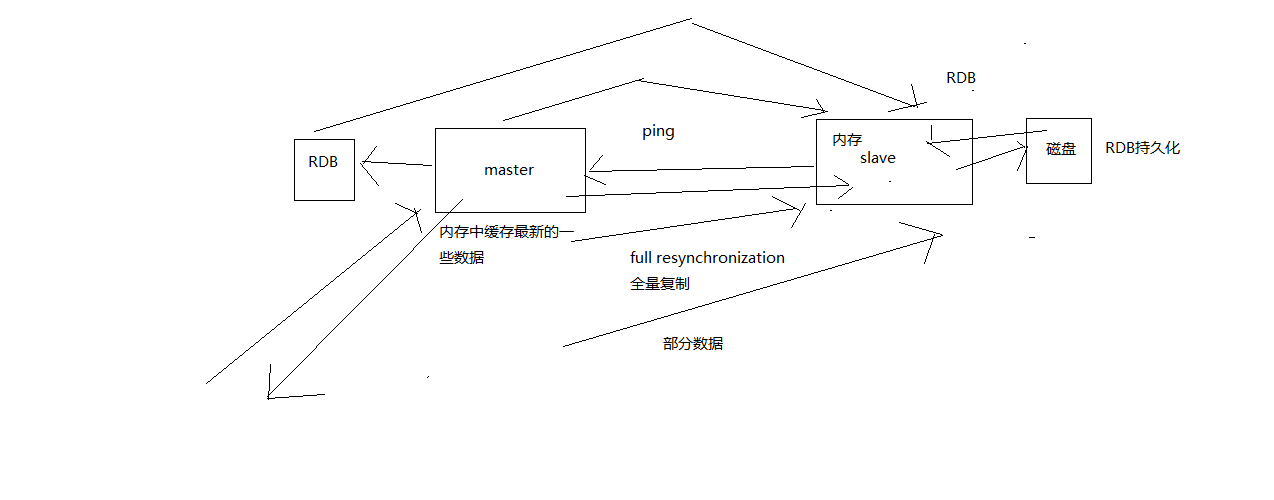
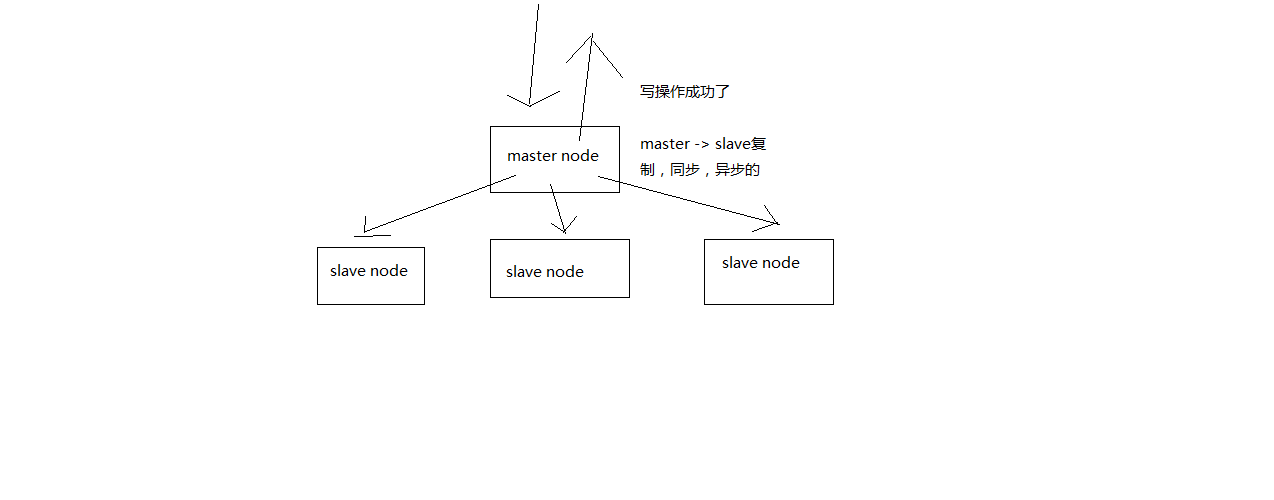
* 1. 如果redis要支撑超过10万+的并发，那应该怎么做？

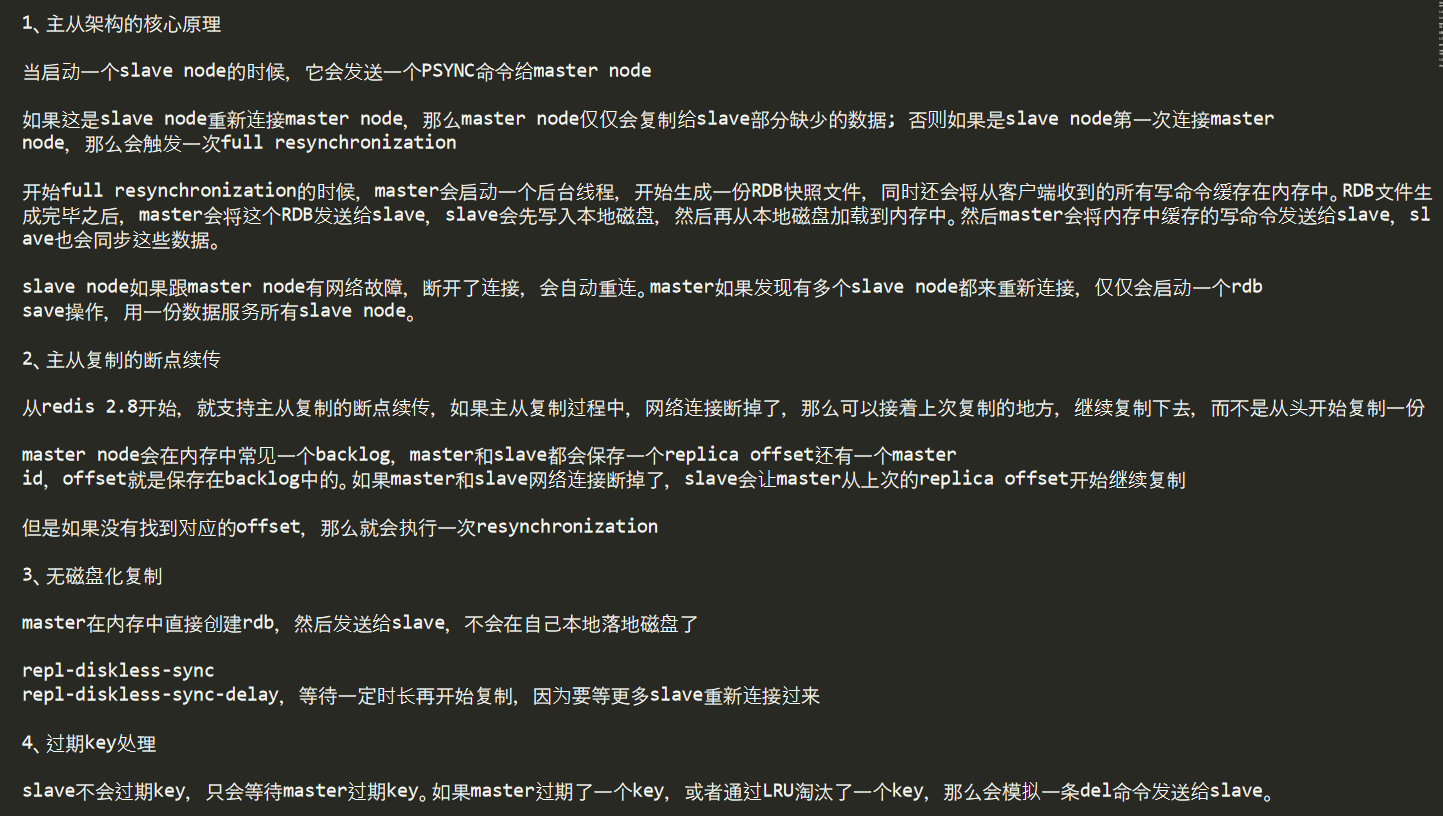
主从架构 -> 读写分离 -> 支撑10万+读QPS的架构

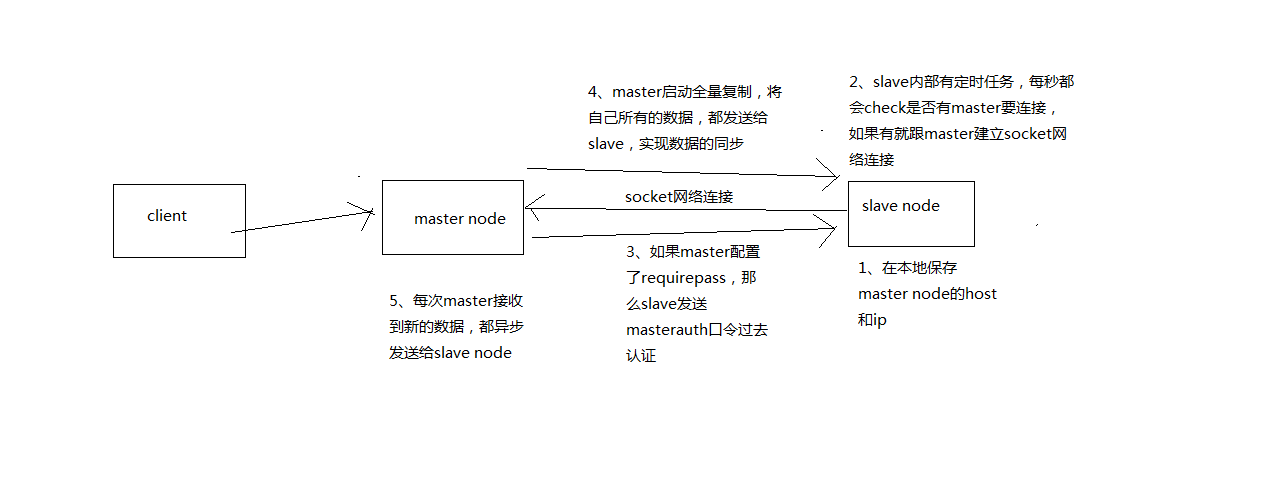


* 1. redis replication的核心机制









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

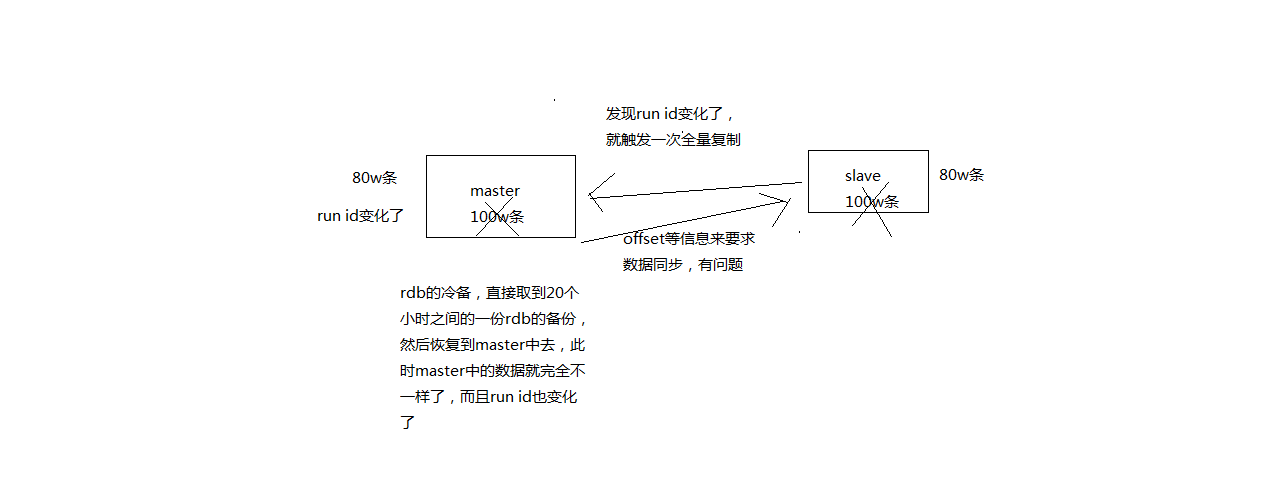
backlog主要是用来做全量复制中断候的增量复制的

（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

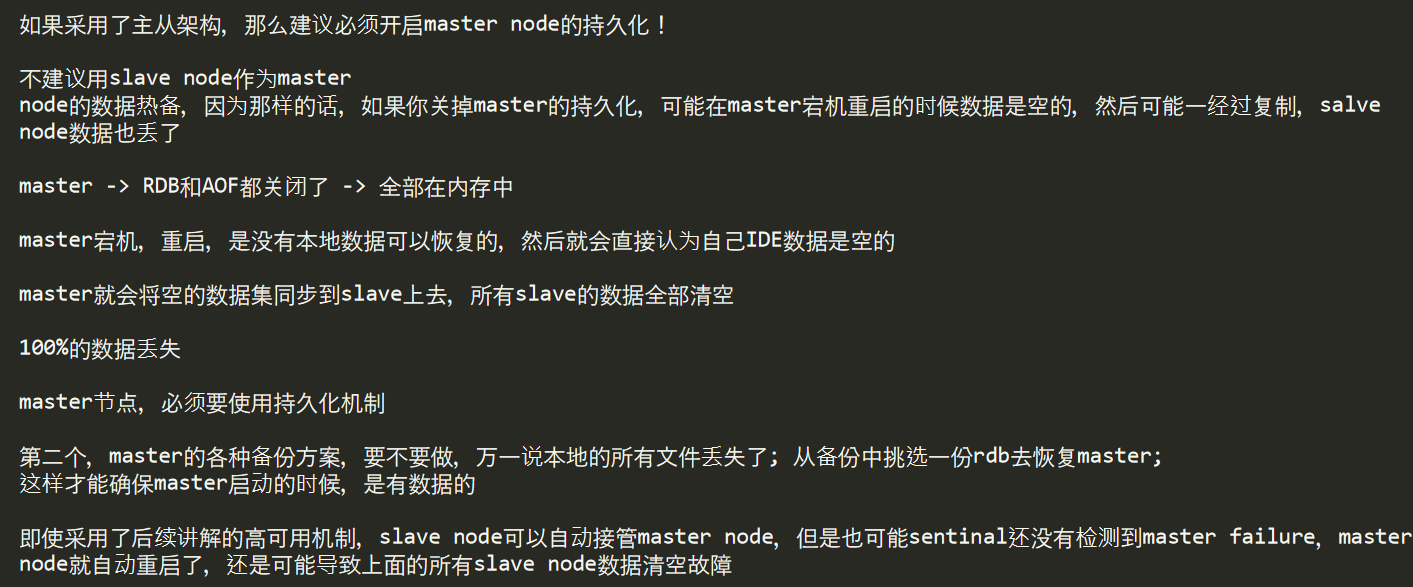
主从节点互相都会发送heartbeat信息

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

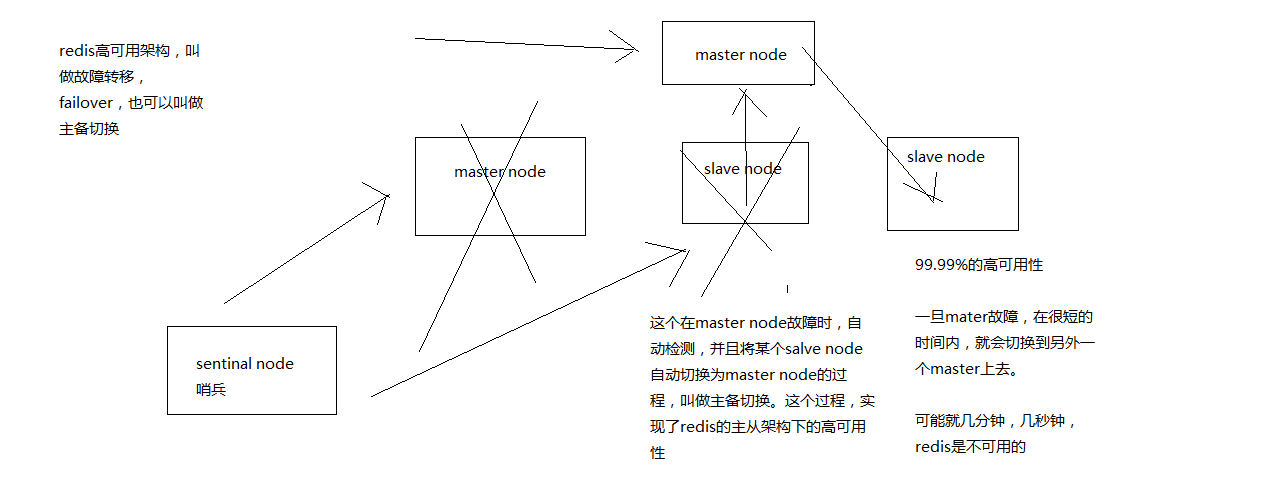
* 1. 压测

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

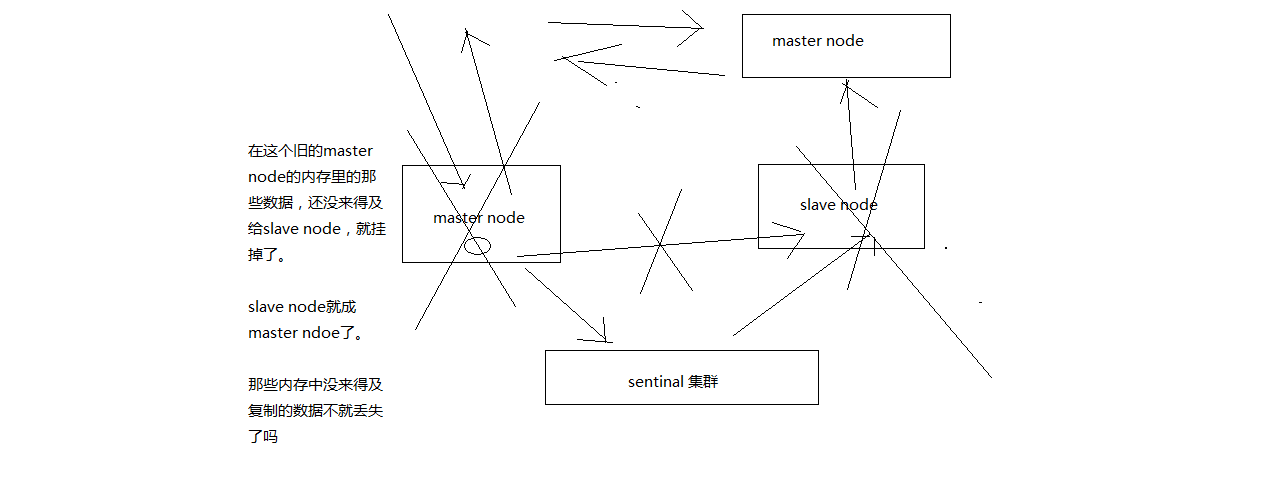
* 1. redis基于哨兵的高可用性





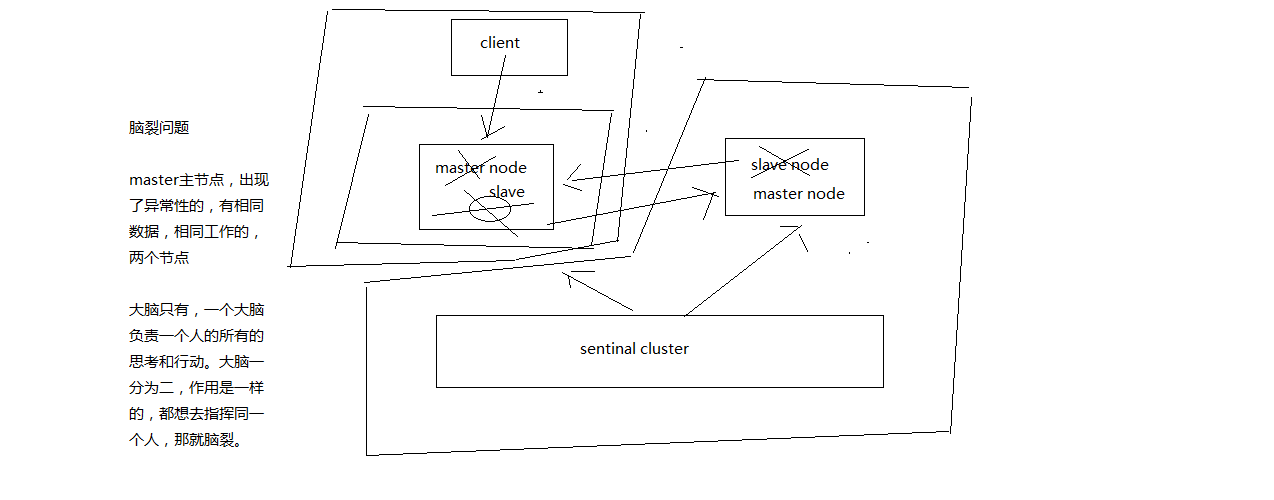
（接上）同时3个哨兵的majority是2，所以还剩下的2个哨兵运行着，就可以允许执行故障转移

* 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

