一、Redis持久化　RDB&AOF

Redis是一种高级key-value 型的NoSQL数据库。

它跟memcached类似是内存型数据库，

不过Redis数据可以做持久化，即内存中的数据可以同步到磁盘进行存储。

而且Redis所支持的数据类型很丰富。有字符串，链表，集合和有序集合等。

Redis 支持在服务器端计算集合的并交和补集difference)等，还支持多种排序功能。

所以Redis也可以被看成是一个数据结构服务器

Redis的所有数据都是保存在内存中，

然后不定期的通过**异步方式**保存到磁盘上(这称为“半持久化模式”RDB);

也可以把每一次数据变化都写入到一个append only fle(AOF)里面(这称为“全持久化模式”)

Redis的数据都存放在内存中,如果没有配置持久化功能，Redis重启后数据就全丢失了，

于是需要开启Redis的持久化功能，将数据保存到磁盘上，

当Redis重启后，可以从磁盘中恢复数据。

Redis 提供两种方式进行持久化，

RDB持久化原理　--类似定时执行mysqldump

是将Redis 在内存中的数据库记录定时dump到磁盘上的RDB持久化)，

AOF (append only file)　---binlog类似

持久化(原理是将Reids 的操作日志以追加的方式写入文件)。

那么这两种持久化方式有什么区别呢，改如何选择呢?

1.RDB和AOF的区别

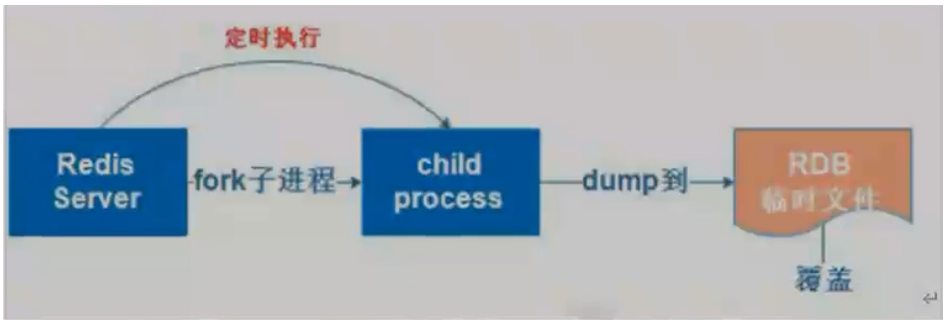
RDB持久化

是指在指定的时间间隔内将内存中的数据集快照写入磁盘，

实际操作过程

是fork一个子进程，

先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储



AOF持久化

以日志的形式记录服务器所处理的每一个写、删除操作

查询操作不会记录（类似于mysql的binlog）

以文本的方式记录，可以打开文件看到详细的操作记录

2.RDB和AOF的优缺点

RDB优势

1) 一旦采用该方式，那么你的整个Redis数据库将只包含一个文件

这对于文件备份而言是非常完美的。

比如，你可能打算每个小时归档一次最近24小时的数据，

同时还要每天归档一次最近30天的数据。

通过这样的备份策略，一 旦系统出现灾难性故障，

我们可以非常容易的进行恢复。

2) 对于灾难恢复而言，RDB是非常不错的选择

因为我们可以非常轻松的将一个单独的文件压缩后再转移到其它存储介质上

3) 性能最大化

对于Redis的服务进程而言，在开始持久化时，

它唯一需要做的只是 fork出子进程，

之后再由子进程完成这些持久化的工作，

这样就可以极大的避免服务进程执行IO操作了

4) 相比于 AOF机制，如果数据集很大，RDB的启动效率会更高

RDB劣势

1） 如果想保证数据的高可用性，即最大限度的避免数据丢失

那么RDB将不是一 个很好的选择

因为系统一旦在定时持久化之前出现宕机现象

此前没有来得及写入磁盘的数据都将丢失

2） 由于 RDB是通过fork 子进程来协助完成数据持久化工作的

因此，如果当数据集较大时，可能会导致整个服务器停止服务几百毫秒，甚至是1秒钟。

AOF优势

1） 该机制可以带来更高的数据安全性，即数据持久性

Redis 中提供了3种同步策略，

即每秒同步、每修改同步和不同步。

事实上，每秒同步也是异步完成的，其效率也是非常高的，所差的是一旦系统出现宕机现象，那么这一秒钟之内修改的数据将会丢失。

而每修改同步，我们可以将其视为同步持久化，

即每次发生的数据变化都会被立即记录到磁盘中。

可以预见，这种方式在效率上是最低的。

2） 由于该机制对日志文件的写入操作采用的是append模式

因此在写入过程中即使出现宕机现象，也不会破坏日志文件中已经存在的内容。

然而如果我们本次操作只是写入了一半数据就出现了系统崩溃问题，

不用担心，在Redis下一次启动之前，

我们可以通过redis-check-aof 工具来帮助我们解决数据一致性的问题。

3） 如果日志过大，Redis 可以自动启用rewrite 机制

即Redis以append模式不断的将修改数据写入到老的磁盘文件中，

同时Redis还会创建一个新的文件用于记录此期间有哪些修改命令被执行。

因此在进行rewrite切换时可以更好的保证数据安全性

4) AOF 包含一个格式清晰、易于理解的日志文件用于记录所有的修改操作

事实上,我们也可以通过该文件完成数据的重建。

AOF的劣势

1） 对于相同数量的数据集而言，AOF文件通常要大于RDB文件

---带给redis的压力更大

RDB 在恢复大数据集时的速度比AOF的恢复速度要快。

2） 根据同步策略的不同，AOF在运行效率上往往会慢于RDB

总之，每秒同步策略的效率是比较高的，同步禁用策略的效率和RDB一样高效

二者选择的标准，就是看系统是愿意牺牲一些性能， 换取更高的缓存一致性(aof)，
还是愿意写操作频繁的时候，不启用备份来换取更高的性能，
待手动运行save的时候，再做备份(rdb)。 rdb 这个就更有些eventually consistent的意思了。

3.Redis持久化配置

1.RDB持久化配置

Redis会将数据集的快照dump到dump.rdb文件中。此外，我们也可以通过配置文件来修改Redis服务器dump快照的频率，在打开6379.conf文件之后，我们搜索save可以看到下面的配置信息：**时间短，更新越频繁，越执行快照**

```
save 900 1      #在900秒(15分钟)之后，如果至少有1个key发生变化，则dump内存快照
save 300 10     #在300秒(5分钟)之后，如果至少有10个key发生变化，则dump内存快照
save 60 10000   #在60秒(1分钟)之后，如果至少有10000个key发生变化，则dump内存快照
[root@localhost ~]# cd /usr/src/
[root@localhost src]# cd redis-4.0.10/
[root@localhost redis-4.0.10]# vim /etc/redis/6379.conf
 219 save 900 1
 220 save 300 10
 221 save 60 10000
 242 rdbcompression yes        # 开启压缩
 254 dbfilename dump.rdb       # rdb存储位置
 264 dir /var/lib/redis/6379     # dir保存的位置
[root@localhost ~]# redis-cli
127.0.0.1:6379> set name sofia
OK
127.0.0.1:6379> set age 18
OK
127.0.0.1:6379> set address beijing
OK
[root@localhost ~]# ls /var/lib/redis/6379
dump.rdb
[root@localhost ~]# /etc/init.d/redis_6379 stop
[root@localhost ~]# rm -rf /var/lib/redis/6379/dump.rdb
[root@localhost ~]# /etc/init.d/redis_6379 start
```

```
Starting Redis server...
[root@localhost ~]# redis-cli
127.0.0.1:6379> keys *
(empty list or set)
```

2.AOF持久化配置

Redis的配置文件中存在三种同步方式，它们分别是:

```
appendfsync always    #每次有 数据修改发生时都会写入AOF文件
appendfsync everysec    #每秒钟同步一次， 该策略为AOF的缺省策略
appendfsync no      #从不同步。高效但是数据不会被持久化
[root@localhost ~]# vim /etc/redis/6379.conf
 673 appendonly yes
 677 appendfilename "appendonly.aof"       # 同步文件名，还是原来的目录
 703 appendfsync everysec          # 每秒钟同步一次
 744 auto-aof-rewrite-percentage 100
 745 auto-aof-rewrite-min-size 64mb
[root@localhost ~]# /etc/init.d/redis_6379 stop
Stopping ...
Redis stopped
[root@localhost ~]# rm -rf /var/lib/redis/6379/dump.rdb
[root@localhost ~]# /etc/init.d/redis_6379 start
Starting Redis server...
127.0.0.1:6379> keys *
(empty list or set)
```

```
127.0.0.1:6379> set name jkjk
OK
127.0.0.1:6379> set age 18
OK
127.0.0.1:6379> set address beijing
OK
[root@localhost ~]# ls /var/lib/redis/6379/appendonly.aof
/var/lib/redis/6379/appendonly.aof
127.0.0.1:6379> save      # 手动执行save就是在进行持久化操作
[root@localhost ~]# ls /var/lib/redis/6379/
appendonly.aof  dump.rdb
[root@localhost ~]# rm -rf /var/lib/redis/6379/dump.rdb
```

```
[root@localhost ~]# ls /var/lib/redis/6379/
appendonly.aof
[root@localhost ~]# cat /var/lib/redis/6379/appendonly.aof
[root@localhost ~]# service redis_6379 restart
Stopping ...
Redis stopped
Starting Redis server...
127.0.0.1:6379> keys *
1) "name"
2) "address"
3) "age"
```

3.AOF持久化缺点:

1) Redis会不断地将被执行的命令记录到AOF文件里面

所以随着Redis不断运行，AOF文件的体积也会不断增长。

在极端情况下，体积不断增大的AOF文件甚至可能会用完硬盘的所有可用空间

2) Redis 在重启之后需要通过重新执行AOF文件记录的所有写命令来还原数据集

所以如果AOF文件的体积非常大，那么还原操作执行的时间就可能会非常长

为了解决AOF文件体积不断增大的问题

用户可以向Redis发送BGREWRITEAOF命令，

这个命令会通过移除AOF文件中的冗余命令来重写(rewrite) AOF文件，

使AOF文件的体积变得尽可能地小。

4.AOF重写功能

BGREWRITEAOF的工作原理和BGSAVE创建快照的工作原理非常相似:

Redis会创建一个子进程，

然后由子进程负责对AOF文件进行重写。

因为AOF文件重写也需要用到子进程

所以快照持久化因为创建子进程而导致的性能问题和内存占用问题

在AOF持久化中也同样存在

跟快照持久化可以通过设置save选项来自动执行BGSAVE一样，

AOF持久化也可以通过设置auto-aof-rewrite-percentage选项和auto aof-rewrite-min-size选项来自动执行BGREWRITEAOF

举个例子，假设用户对Redis设置了配置选项auto-aof rewrite percentage 100

和auto-aof-rewrite-min-size 64mb，并且启动了AOF持久化，

那么当AOF文件的体积大于64MB，
并且AOF文件的体积比上一次重写之后的体积大了至少一倍(100%) 的时候，
Redis 将执行BGREWRITEAOF命令。
如果AOF重写**执行得过于频繁**的话，用户可以考虑将auto-aof-rewrite

国内时间同步 ntp服务器地址

二、Redis主从复制

一、环境概述

在分布式集群系统中为了解决服务单点故障问题
通常会把数据复制出多个副本部署到不同的机器中，
满足故障恢复和负载均衡等需求。
Redis也是如此，它为我们提供了复制功能，
实现了相同数据的多个Redis副本。
复制功能是高可用Redis的基础，
Redis 的哨兵和集群(Cluster) 模式都是在主从复制模式的基础上实现的。
复制也是Redis日常运维的常见维护点。
因此深刻理解复制的工作原理与使用技巧对日常的运维非常有帮助

二、Redis主从结构

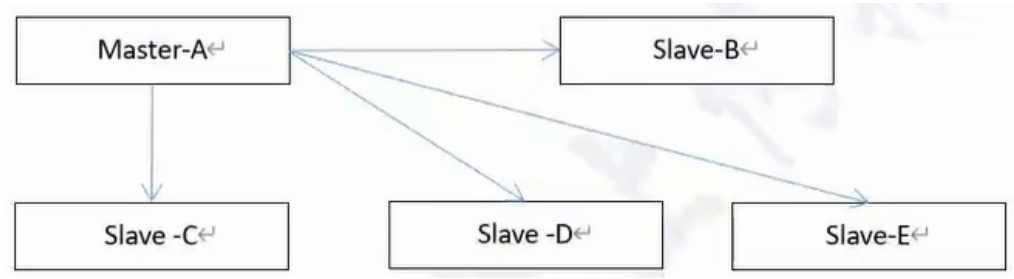
2.1、一主一从结构

一主一从结构是Redis最简单的复制拓扑结构，
用于主节点出现宕机时从节点来提供故障转移支持。
当应用写命令并发量较高且需要持久化时，可以只在从节点上开启AOF,
这样既保证数据安全性同时，也避免了持久化对主节点的性能压力

2.2、一主多从结构

一主多从结构(又称为星型拓扑结构)
使得应用端可以利用多个从节点实现读写分离方案
对于读占比较大的场景，可以把读命令发送到多个从节点来分担主节点压力
同时在日常开发中如果需要执行一些比较耗时的读命令
可以在其中一台从节点上执行，防止慢查询
对主节点造成阻塞从而影响线上服务的稳定性
对于写并发量较高的场景

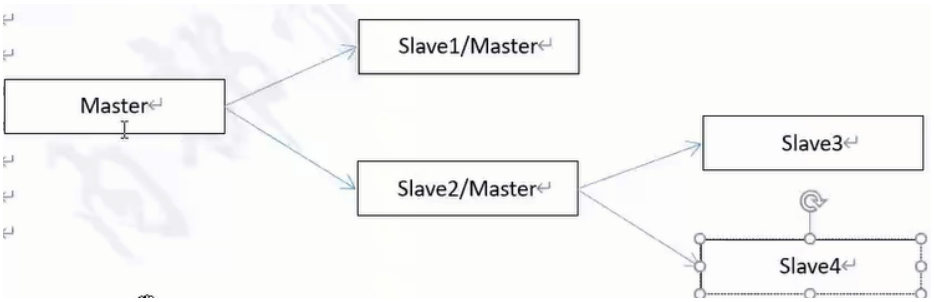
多个从节点会导致主节点写命令的多次发送从而过度消耗网络带宽
同时也加重了主节点的负载影响服务稳定性



再加上读写分离会更优化整个网络环境

2.3、树状主从结构

树状结构(级联结构)使得从节点不但可以复制主节点数据，
同时可以作为其他从节点的主节点继续向下层复制。
通过引入中间复制层，可以有效降低主节点负载和需要传送给从节点的数据量。
当主节点需要挂载多个从节点时为了避免对主节点的性能干扰，
可以采用树状主从结构降低主节点压力



这种方式与一主多从的区别是：master的压力更小

三、主从复制原理

---着重理解一下，无需完整背诵，面试不会问太多

3.1、复制流程 ---面试（mysql主从复制原理）

在从节点执行slaveof命令后，（指定master的IP和master的窗口），复制过程便开始运作，大致分为6个过程:

(1)保存主节点信息

执行slaveof命令后从节点只保存主节点的地址信息便直接返回
这时建立复制流程还没有开始
在从节点执行info replication命令可以看到主节点的相关信息
并在日志中记录复制启动信息

(2)主从建立socket连接

从节点内部通过每秒运行的定时任务维护复制相关逻辑
当定时任务发现存在新的主节点后，会尝试与该节点建立网络连接
从节点会建立一个socket套接字连接，专门用于接收主节点发送的复制命令

如果从节点无法建立连接，定时任务会无限重试

直到连接成功或者执行slaveof no one命令取消复制

(3)发送ping命令

连接建立成功后，从节点发送ping请求进行首次通信，目的在于:

检测主从之间网络 套接字是否可用

检测主节点当前是否可接受处理命令

如果发送ping命令后，从节点没有收到主节点回复pong或者超时未回复

从节点会断开复制连接，等待下次定时任务发起重连

(4)权限验证

如果主节点设置了requirepass 参数，则需要密码验证，

从节点必须配置masterauth参数保证与主节点相同的密码才能通过验证。

如果验证失败复制将终止，从节点重新发起复制流程

(5)同步数据集

主从复制连接正常通信后，对于首次建立复制的场景

主节点会把所有的数据全部发送给从节点

这部分操作是耗时最长的步骤

在同步过程中会分为两种情况:全量同步和部分同步

(6)命令持续复制

当主节点把当前的数据同步给从节点后

便完成了复制建立的流程

接下来主节点会持续地把写命令发送给从节点，保证数据的一致性

3.2、 复制类型

Redis在2.8及以上版本使用psync命令完成主从数据同步，
过程分为:全量复制和部分复制

1.全量复制

一般用于初次复制的场景，Redis 早期支持的复制功能只有全量复制，
他会把主节点全部数据一次性发送给从节点，
当数据量较大时，会对主从节点和网络造成很大的开销

2.部分复制

用于处理在主从复制中因网络闪断等原因造成的数据丢失场景，
当从节点再次连上主节点后，
如果条件允许，主节点会补发丢失数据给从节点
而补发的数据量远远小于全量数据，
可以有效避免全量复制的过高开销部分复制是对老版复制的重大优化，
有效避免了不必要的全量复制操作。
因此当使用复制功能时，尽量采用2.8以上版本的Redis

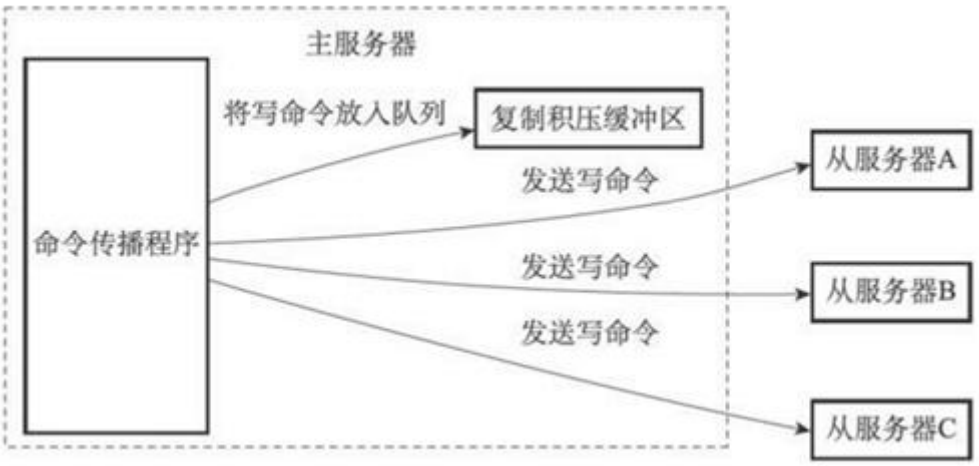
psync命令运行时需要以下组件的支持:

(1)复制偏移量 -----类似页码的感觉，识别差异

参与复制的主从节点都会维护自身复制偏移量
主节点在处理完写入命令后，
会把命令的字节长度做累加记录，
统计信息在info replication中的master repl offset 指标中
从节点每秒钟上报自身的复制偏移量给主节点
因此主节点也会保存从节点复制偏移量
并且从节点在收到主节点发送的命令后
也会累加记录自身的偏移量，记录在自己的统计信息中

(2)复制积压缓冲区

复制积压缓冲区是保存在主节点上的一个固定长度的队列，默认大小为1MB，
当主节点进行主从复刚时，写命令不但会发送给从节点，还会写入复制积压缓冲区中



缓冲区本质上是一个先进先出的定长队列

可以实现保存最近已复制数据的功能，

用于部分复制和复制命令丢失的数据补救

统计信息记录与info replication中

repl_ backlog__active:1　　//开启复制缓冲区

repl_ backlog_ size:1048576　　//缓冲区最大大小

repl_ backlog_ first_ byte_ offset:7479　　//起始偏移量

repl_ backlog_ histlen:1048576　　//已保存数据的有效长度

(3)主节点运行ID

每个Redis节点启动后，都会动态分配一 个运行ID,用来唯一识别一个 Redis节点。

从节点通过保存主节点的运行ID，来判断自己正在向谁进行复制，当主的运行ID改变后，

从节点将进行全量复制

注意: Redis 重启时，运行ID会随之改变

若不希望从节点对主节点进行全量复制，则需要保持主节点ID不变

而当修改配置文件需要重启操作时

可以对主节点使用redis-cli debug reload 命令来进行对配置文件的重新加载

而不用关闭Redis

但是，debug reload命令会阻塞Redis主进程

阻塞期间会生成本地RDB快照并清空数据之后再加载RDB文件

因此对于无法容忍阻塞的场景，谨慎使用

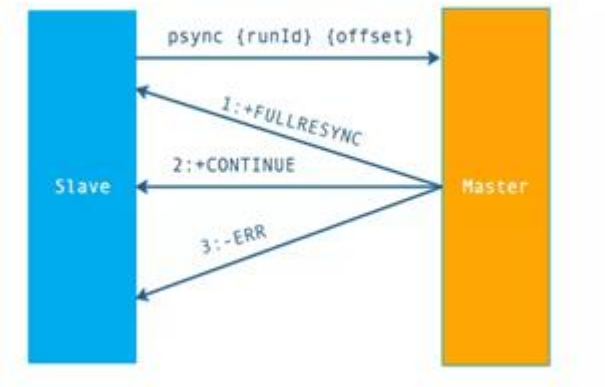
(4) psync 命令

命令格式:

psync.. { runID} { offset }

runID:主节点的运行id

offset:当前从节点已复制的数据偏移量



流程说明:

1、从节点发送psync命令给主节点，如果没有runID，默认值为?，

如果是第一次复.制，则offset默认值为-1

2、主节点根据psync参数和自身数据情况决定响应结果

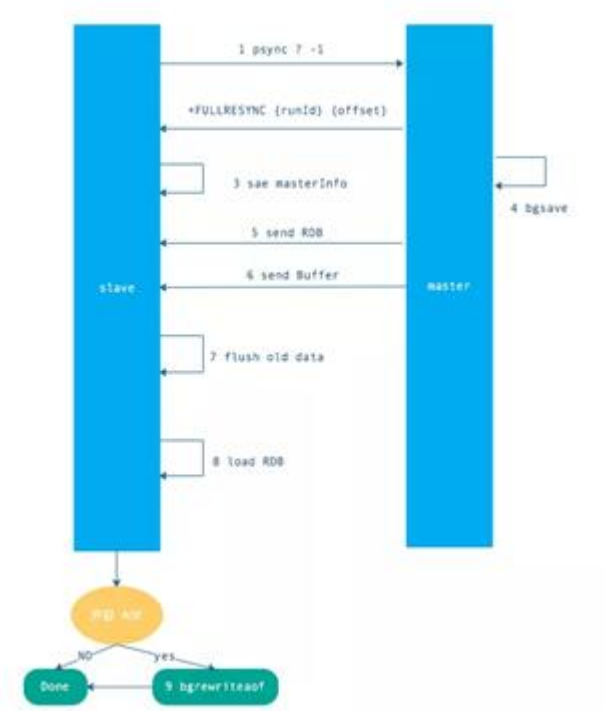
如果回复+FULLRESYNC {runlD} {offset}，从节点开始全量复制

如果回复+CONTINUE，从节点开始部分复制

如果回复+ERR，说明主节点版本低于2.8，不支持psync，

从节点会发送sync命令进行全量复制

3.2.1、 全量复制



全量复制流程说明:

(1)从节点发送psync命令给主节点，psync ?-1

(2)主节点根据命令解析出当前进行全量复制，回复+FULLRESYNC相应。

(3)从节点接收主节点的相应数据保存运行ID和偏移量offset

(4)主节点执行bgsave保存RDB文件(持久化)到本地

(5)主节点发送RDB文件给从节点，从节点把接收的RDB文件保存在本地并直接作为从节点的数据文件

注意:对于数据量较大的主节点，生成的RDB文件超过6GB以上时要格外小心。

如果传输时间超过repl-timeout所配置的值(默认60秒)，

从节点将放弃接受RDB文件并清理已经下载的临时文件，导致全量复制失败

(6)从节点开始接收RDB文件到完成期间，主节点仍然响应读写命令，

所以在RDB文件传输期间，

主节点会将修改操作保存在缓冲区中

当从节点加载完RDB文件后，

主节点再把缓冲区中的数据发送给从节点，保证数据一致

注意:高流量写入场景可能会导致主节点复制客户端缓冲区溢出，

默认配置为client-output- buffer-limit slave 256MB 64MB 60

如果60秒内缓冲区消耗持续大于64MB或者直接超过256MB时

主节点将直接关闭复制客户端连接，造成复制失败

对于主节点，当发送完所有的数据后就认为全量复制完成，但是对于从节点全量复制还有后续步骤要处理

(7)从节点接收完主节点传送来的全部数据后，会清空自身旧数据

(8)加载RDB文件

(9)加载完毕后，如果当前节点开启了AOF持久化功能

会立刻进行bgrewriteaof操作，为了保证全量复制后AOF持久化文件可立刻可用

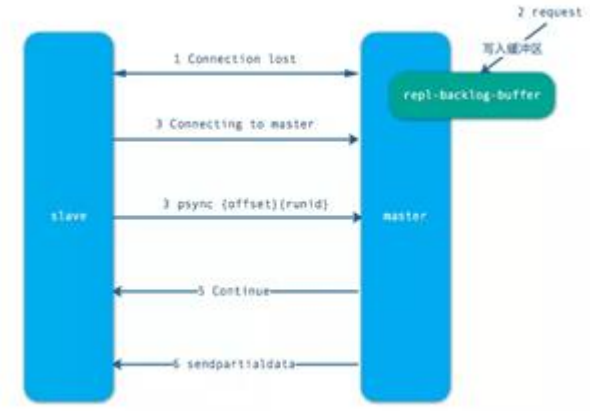
3.2.2、部分复制

当主从复制过程中，如果出现网络闪断或者命令丢失等异常情况时，

从节点会向主节点要求补发丢失的命令数据，

如果主节点的复制积压缓冲区内存在这部分数据则会直接发给从节点。

在减少开销的同时，保证了数据的一致性。流程如下：



部分复制流程说明:

(1)当主从网络断开，超过repl-timeout设置的时间，主会认为从故障而中断数据连接

(2)连接中断期间，主节点依然会响应客户端命令，但无法发送给从节点，

不过主节点内部存在复制加压缓冲区，

所以可以保存最近一段时间的写命令数据，默认最大1MB

(3)当网络恢复，从节点会重新连接到主节点

(4)连接恢复后，

从节点会将之前保存的自己已复制的偏移量和运行ID用psync命令发送给主节点

要求进行部分复制操作

(5)主节点接到psync命令后，核对信息

之后根据offset在自身复制积压缓冲区查找，

如果偏移量之后的数据存在缓冲区中

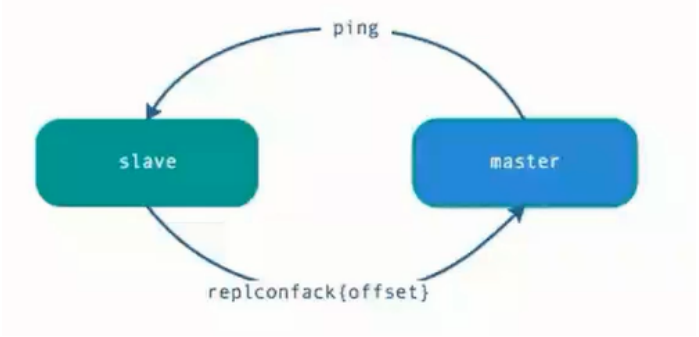
则对从节点发送+CONTINUE响应，表示可以进行部分复制

(6)主节点根据偏移量把复制积压缓冲区里的数据发给从节点

保证主从复制进入正常状态

3.3、心跳

主从节点在建立复制后，他们之间维护着长连接并彼此发送心跳命令心跳的关键机制如下:



1、主从都有心跳检测机制

各自模拟成对方的客户端进行通信

通过clientlist 命令查看复制相关客户端信息

主节点的连接状态为flags=M， 从节点的连接状态是flags=S

2、主节点默认每隔10秒对丛节点发送ping 命令

可修改配置repl-ping-slave-period控制发送频率

3、从节点在主线程每隔一秒发送replconf ack{offset}命令

给主节点上报自身当前的复制偏移量

4、主节点收到replconf 信息后，判断从节点超时时间，如果超过repl-timeout60 秒，则判断节点下线

注意:

为了降低主从延迟，一般把redis主从节点部署在相同的机房/同城机房，

避免网络延迟带来的网络分区造成的心跳中断等情况

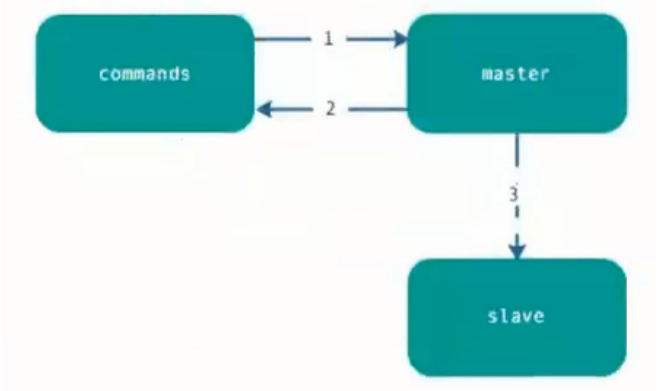
3.4、异步复制

主节点不但负责数据读写，还负责把写命令同步给从节点，

写命令的发送过程是异步完成

也就是说主节点处理完写命令后立即返回客户端，

并不等待从节点复制完成。异步复制的步骤很简单，如下:



1、主节点接受处理命令

2、主节点处理完后返回响应结果

3、对于修改命令，异步发送给从节点，从节点在主线程中执行复制的命令

四、主从配置管理

4.1、 建立复制

主从复制配置方式有三种:

在配置文件中加入slaveof {masterHost} {masterPort}随Redis启动生效

在redis-server启动命令后加入--slaveof {masterHost} {masterPort}

直接使用命令slaveof {masterHost} {masterPort}生效

4.2、断开复制

slaveof命令不但可以建立复制，也可以执行slaveof no one来断开与主节点复制关系

4.3、 安全性

主节点通常会设置

requirepass

参数进行密码验证，这时所有的客户端访问必须使用

auth命令实行校验。因此需要在从节点配置masterauth参数与主节点密码保持一致

4.4、只读(默认)

默认情况下，从节点使用slave-read-only=yes配置为只读模式。由于复制只能从主节点到从节点，对于从节点的任何修改主节点都是无法感知的，修改从节点会造成主从数据不一致。因此建议线上将从节点设置为只读模式

4.5、 传输延迟

主从节点一般在不同机器上

复制时的网络延迟就成为需要考虑的问题

Redis为我们提供了repl-disable-tcp-nodelay参数用于控制是否关闭TCP_ NODELAY

默认为no，为开启状态

设置为no时，表示开启TCP_ NODELAY，允许小包发送，使得网络延迟变小

但是会增加网络带宽消耗设置为yes时，表示关闭TCP_ NODELAY

主节点会将小包合并发送

默认发送时间间隔取决于Linux内核，一般为40ms

节省了带宽，但是增大了延迟

五、部署Redis主从复制

5.1、案例环境

系统	IP地址	主机名	所需软件	redis角色及端口
CentOS 7.7	192.168.200.107	master.redis.com	redis-4.0.10.tar.gz	master: 6379 (6379.conf)
CentOS 7.7	192.168.200.107	slave.redis.com	redis-4.0.10.tar.gz	slave: 6379 (6379.conf) slave: 6380 (6380.conf)

5.2、案例步骤推荐

- 安装并配置master角色的redis服务;
- 安装并配置slave角色的redis服务(双实例);
- 配置master角色的文件;
- 配置slave角色的文件;
- 验证主从复制功能;

5.3、 案例实施

安装并配置master角色的redis服务，从主机进行同样的安装配置

```
[root@master ~]# iptables -F
[root@master ~]# systemctl stop firewalld
[root@master ~]# setenforce 0
setenforce: SELinux is disabled
[root@slave ~]# tar xf redis-4.0.10.tar.gz -C /usr/src/
```

执行make编译安装，从主机进行同样的安装配置

在我们之前的学习中，我们都会执行./configure进行编译从而生成Makefile，

redis 的源码包中则直接提供了Makefile. 文件所以在解压完成之后直接进行make编译即可)

redis其实可以不用执行make install
但是为了后面方便我们执行所以执行make install

在安装过程中，若想更改默认的安装路径，可使用以下命令格式来进行安装操作
make PREFIX=安装路径install

生成启动脚本及配置文件信息
make install只是生成了二进制文件到系统中，并没有启动脚本和配置文件，所以此时使用redis默认提供的install server.sh 脚本进行生成启动脚本和配置文件

```
[root@master ~]# vim /etc/redis/6379.conf
  70 bind 192.168.200.107
  89 protected-mode no
 137 daemonize yes
 501  requirepass 123456
[root@master ~]# /etc/init.d/redis_6379 restart
Stopping ...
Redis stopped
Starting Redis server...
[root@master ~]# netstat -lnpt |grep 6379
tcp        0      0 192.168.200.107:6379    0.0.0.0:*               LISTEN
90165/redis-server
[root@slave ~]# cd /usr/src/redis-4.0.10/utils
[root@slave utils]# ./install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379]
Selecting default: 6379
Please select the redis config file name [/etc/redis/6379.conf]
Selected default - /etc/redis/6379.conf
Please select the redis log file name [/var/log/redis_6379.log]
Selected default - /var/log/redis_6379.log
Please select the data directory for this instance [/var/lib/redis/6379]
Selected default - /var/lib/redis/6379
Please select the redis executable path [/usr/local/bin/redis-server]
```

```
Selected config:
Port           : 6379
Config file    : /etc/redis/6379.conf
Log file       : /var/log/redis_6379.log
Data dir       : /var/lib/redis/6379
Executable     : /usr/local/bin/redis-server
Cli Executable : /usr/local/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/6379.conf => /etc/init.d/redis_6379
Installing service...
Successfully added to chkconfig!
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!
[root@slave utils]# ./install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379] 6380
Please select the redis config file name [/etc/redis/6380.conf]
Selected default - /etc/redis/6380.conf
Please select the redis log file name [/var/log/redis_6380.log]
Selected default - /var/log/redis_6380.log
Please select the data directory for this instance [/var/lib/redis/6380]
Selected default - /var/lib/redis/6380
Please select the redis executable path [/usr/local/bin/redis-server]
Selected config:
Port           : 6380
Config file    : /etc/redis/6380.conf
Log file       : /var/log/redis_6380.log
Data dir       : /var/lib/redis/6380
Executable     : /usr/local/bin/redis-server
Cli Executable : /usr/local/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/6380.conf => /etc/init.d/redis_6380
```

```
Installing service...
Successfully added to chkconfig!
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!

[root@slave utils]# netstat -lnpt | grep redis
tcp        0      0 127.0.0.1:6379          0.0.0.0:*               LISTEN
5152/redis-server
tcp        0      0 127.0.0.1:6380          0.0.0.0:*               LISTEN
5237/redis-server
```

修改 redis_6379配置文件:

```
[root@slave utils]# sed -i '/^bind 127.0.0.1$/s/127.0.0.1/192.168.200.108/g'
/etc/redis/6379.conf
```

关闭redis的保护模式:

```
[root@slave utils]# sed -i '/protected-mode/s/yes/no/g' /etc/redis/6379.conf
```

开启redis的后台守护进程模式:

```
[root@slave utils]# sed -i '/daemonize/s/no/yes/g' /etc/redis/6379.conf
```

设置redis的密码为123456:

```
[root@slave utils]# sed -i '/requirepass/s/foobared/123456/g'
/etc/redis/6379.conf
```

开启redis密码: (去注释)

```
[root@slave utils]# sed -i '/requirepass 123456/s/^#//g' /etc/redis/6379.conf
[root@slave utils]# service redis_6379 restart
Stopping ...
Redis stopped
Starting Redis server...
[root@slave utils]#
[root@slave utils]# netstat -lnpt | grep 6379
tcp        0      0 192.168.200.108:6379    0.0.0.0:*               LISTEN
5400/redis-server 1
```

修改 redis_6380配置文件: 优化

```
[root@slave utils]# sed -i '/^bind 127.0.0.1$/s/127.0.0.1/192.168.200.108/g'
/etc/redis/6380.conf
```

关闭redis的保护模式：

```
[root@slave utils]# sed -i '/protected-mode/s/yes/no/g' /etc/redis/6380.conf
```

开启redis的后台守护进程模式：

```
[root@slave utils]# sed -i '/daemonize/s/no/yes/g' /etc/redis/6380.conf
```

设置redis的密码为123456：

```
[root@slave utils]# sed -i '/requirepass/s/foobared/123456/g'
/etc/redis/6380.conf
```

开启redis密码：（去注释）

```
[root@slave utils]# sed -i '/requirepass 123456/s/^#//g' /etc/redis/6380.conf
[root@slave utils]# service redis_6380 restart
Stopping ...
Redis stopped
Starting Redis server...
[root@slave utils]# netstat -lnpt | grep redis
tcp        0      0 192.168.200.108:6379    0.0.0.0:*               LISTEN
5400/redis-server 1
tcp        0      0 192.168.200.108:6380    0.0.0.0:*               LISTEN
5479/redis-server 1
```

连接测试：

```
[root@slave utils]# redis-cli -h 192.168.200.108 -a 123456 -p 6379
Warning: Using a password with '-a' option on the command line interface may not
be safe.
192.168.200.108:6379> ping
PONG
192.168.200.108:6379> exit
[root@slave utils]# redis-cli -h 192.168.200.108 -a 123456 -p 6380
Warning: Using a password with '-a' option on the command line interface may not
be safe.
192.168.200.108:6380> ping
PONG
192.168.200.108:6380>
```

修改master的配置文件：

```
[root@master ~]# vim /etc/redis/6379.conf
 451  min-slaves-to-write 2
#     设置slave节点的数量，
    如果slave节点数量少于此值
    那么master节点将停止客户端的一切写请求
 452  min-slaves-max-lag 10
[root@master ~]# killall -9 redis-server
[root@master ~]# rm -rf /var/run/redis_6379.pid
[root@master ~]# service redis_6379 start
Starting Redis server...
```

修改从下的配置文件：

```
[root@slave utils]# vim /etc/redis/6379.conf
 283  slaveof 192.168.200.107 6379
 290  masterauth 123456
[root@slave utils]# vim /etc/redis/6380.conf
 282  slaveof 192.168.200.107 6379
 289  masterauth 123456
[root@slave utils]# killall -9 redis-server
[root@slave utils]# rm -rf /var/run/redis_6379.pid
[root@slave utils]# service redis_6379 start
Starting Redis server...
[root@slave utils]# rm -rf /var/run/redis_6380.pid
[root@slave utils]# service redis_6380 start
Starting Redis server...
```

如果不考虑优化（安全性，超时时间）的话，步骤会更简单：
只要三个实例都出来，在其配置文件修改下面两个即可

```
 283  slaveof 192.168.200.107 6379
 290  masterauth 123456
```

验证测试：

```
[root@master ~]# rm -rf  /var/lib/redis/6379/appendonly.aof
[root@master ~]# rm -rf  /var/lib/redis/6379/dump.rdb
[root@master ~]# killall -9 redis-server
[root@master ~]# rm -rf /var/run/redis_6379.pid
```

```
[root@master ~]# service redis_6379 start
Starting Redis server...
[root@master ~]# redis-cli -h 192.168.200.107 -a 123456 -p 6379
192.168.200.107:6379> set name pink
OK
192.168.200.107:6379> get name
"pink"
192.168.200.107:6379> keys *
1) "name"
192.168.200.107:6379> info replication
# Replication
role:master
connected_slaves:2
min_slaves_good_slaves:2
slave0:ip=192.168.200.108,port=6379,state=online,offset=4620,lag=0
slave1:ip=192.168.200.108,port=6380,state=online,offset=4620,lag=0
master_replid:cc9ab193403aedd850351aee127dae677e0b7eff
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:4620
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:4620
```

从服务器:

```
[root@slave ~]# redis-cli -h 192.168.200.108 -p 6379 -a 123456
Warning: Using a password with '-a' option on the command line interface may not be safe.
192.168.200.108:6379> keys *
1) "name"
192.168.200.108:6379> get name
"pink"
192.168.200.108:6379> set age 18
(error) READONLY You can't write against a read only slave.
```

```
192.168.200.108:6379> info replication
# Replication
role:slave
master_host:192.168.200.107
master_port:6379
master_link_status:up
master_last_io_seconds_ago:1
master_sync_in_progress:0
slave_repl_offset:5194
slave_priority:100
slave_read_only:1
connected_slaves:0
master_replid:cc9ab193403aedd850351aee127dae677e0b7eff
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:5194
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:5194
[root@slave ~]# redis-cli -h 192.168.200.108 -p 6380 -a 123456
Warning: Using a password with '-a' option on the command line interface may not be safe.
192.168.200.108:6380> keys *
1) "name"
192.168.200.108:6380> get naem
(nil)
192.168.200.108:6380> get name
"pink"
192.168.200.108:6380> exit
```

redis集群方案——redis sentinel哨兵集群

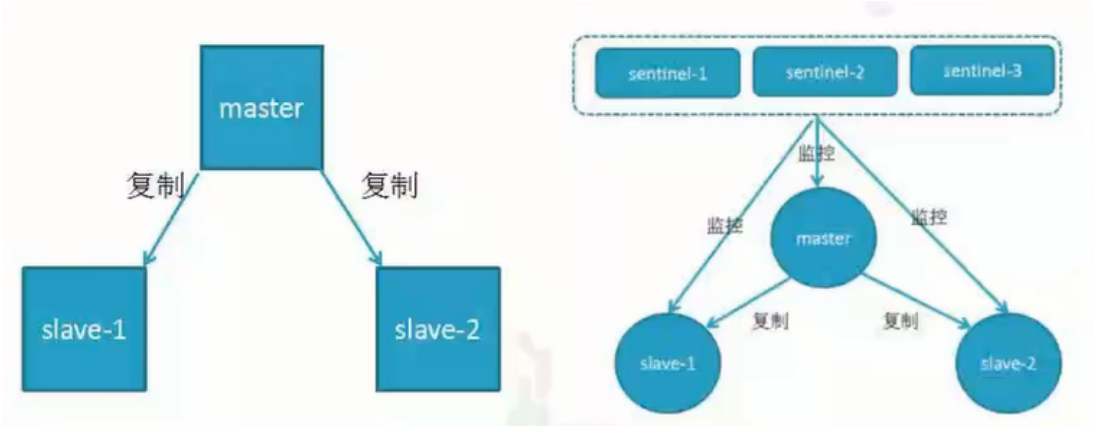
这只是一个数据同步技术（方案），把主的数据同步到从的机器上面，并不能当成企业的架构方案

一、Redis主从复制中的问题

Redis主从复制集群可以将主节点的数据改变同步给从节点，
这样从节点就可以起到两个作用:
第一:
作为主节点的一个备份，一旦主节点出了故障不能继续对外提供服务时，
从节点可以作为后备“顶”上来，并且保证数据尽量不丢失　　**---比较麻烦，需要手动设置**
第二,
从节点可以扩主节点的读能力,通过实现读写分离结构，
可以大大减轻主节点在进行高并发读写操作时的访问压力
但是主从同步也带了一些问题:
一旦主节点故障，需要手动将一个从节点晋升为主节点
需要修改客户端或者应用程序的主节点地址　　**---比较麻烦，当连接redis的时候，地址需要变**
如果是一主多从结构，还需将其他从节点调整，让其从新的主节点进行复制
而以上整个过程都需要人工干预

二、Redis高可用方案

类似于mysql的MHA的角色很像
Redis Sentinel是Redis的高可用实现方案，在实际的生产环境中，
对提高整个系统的高可用性是非常有帮助的。
哨兵是一个分布式架构，
其中包含若干个**Sentinel节点（哨兵节点）**和**Redis数据节点**
每个Sentinel节点都会对数据节点和其他Sentinel节点进行监控
当他发现节点不可达时，会对节点做下线标识
如果被标识的是主节点，他还会和其他Sentinel节点进行“协商”
当半数以上Sentinel节点都认为主节点不可达时，它们会选举出一个Sentinel 节点
来完成自动故障转移的工作，同时会将这个变化实时通知给Redis的应用方
整个过程是完全自动的，不需要人工来介入，所以这套方案很有效的解决了Redis的高可用问题



来一波通俗点的解释吧~

看上图，三个哨兵节点，它们会相互监控，当1号哨兵节点发现master有问题时，它就会赶紧找其他的哨兵节点，进行讨论，说我发现master是有问题的，你们觉得呢？然后三个哨兵节点会进行投票，当半数以上的人都认为这个master是有问题的时候，那么这个哨兵小组就会选举出一个代表负责人，一般来说，第一个发现问题的人会先推荐自己说，这个问题是我发现的，所以本次的故障切换操作由我来做，如果其他人也同意，那么它就是本次的负责人。

这里面，隐含了一个技术参数，哨兵节点一般来说，是奇数；如果是偶数有可能会出现平票的现象

整个故障转移的处理逻辑基本上可分为4步:

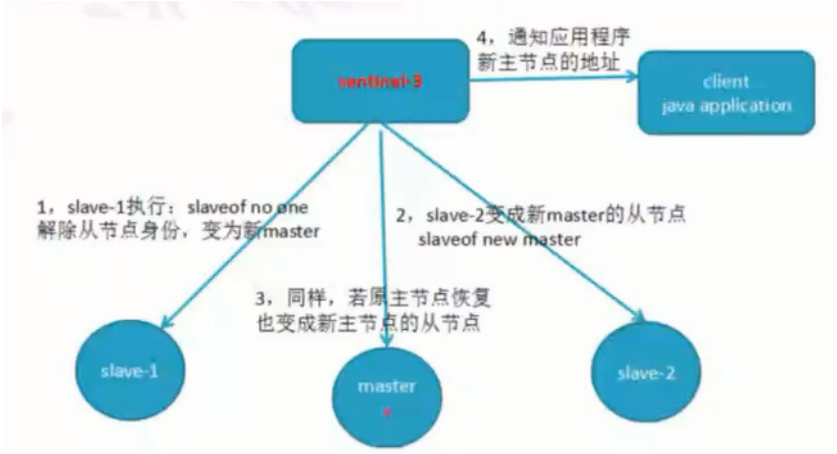
(1) 主节点出现故障，此时两个从节点与主节点失去连接（主从之间有心跳），主从复制失败

(2) 每个Sentinel节点通过定期监控发现主节点出现了故障

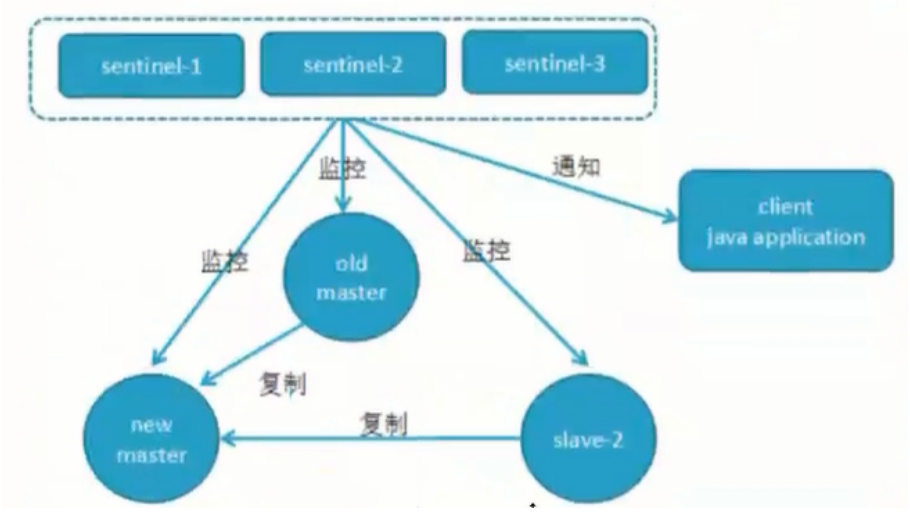
(3) 多个Sentinel节点对主节点的故障达成一致，

选举出其中一个Sentinel节点作为领导者负责本次故障转移工作。

(4) Sentinel 领导者节点执行了故障转移，如下图所示



看上图，如果是哨兵3发现了问题，它会通知slave1：嘿，你别当小弟了，当大哥吧（master），然后哨兵会通知其他从（小弟）以后从新的大哥（master）身上进行同步数据吧，原来的故障master不要了，如果故障master修好了，也得变为新主的从；并且哨兵3会通知其他应用程序，这个集群地址在发生变化，这就是故障切换，转移完的效果，如下图：



三、Sentinel实现原理

3.1、三个定时监控任务

Redis Sentinel通过三个定时监控任务完成对各个节点发现和监控:

(1) 每隔10秒

每个Sentinel节点会向主节点和从节点发送info命令

获取最新的拓扑结构，这个定时任务的作用具体可以表现在三个方面:

通过向主节点执行info命令，获取从节点的信息，

这也是为什么Sentinel节点不需要显示配置监控从节点

当有新的从节点加入时都可以立刻感知出来

节点不可达或者故障转移后，

可以通过info命令实时更新节点拓扑信息

(2) 每隔2秒

每个Sentinel节点会向Redis数据节点的_sentinel _:hello 频道上发送该

Sentinel节点对于主节点的判断以及当前Sentinel节点的信息，

(哨兵小组内有一个群，每个人都订阅了，这个群的消息，每隔两秒每个人都要汇报一下自己的状态以及它们所观察到的master的状态)

同时每个Sentinel节点也会订阅该频道，

来了解其他Sentinel节点以及它们对主节点的判断，

所以这个定时任务可以完成以下两个工作:

发现新的 Sentinel节点:

通过订阅主节点的_sentinel_:hello了解其他的Sentinel节点信息，
如果是新加入的Sentinel节点,将该Sentinel节点信息保存起来，
并与该Sentinel节点创建连接
Sentinel节点之间交换主节点的状态，作为后面客观下线以及领导者选举的依据。

(3) 每隔1秒

每个Sentinel节点会向主节点、从节点、其余Sentinel节点发送一条ping命令做一次心跳检测
来确认这些节点当前是否可达

3.2、 主观下线

每个Sentinel 节点会每隔1秒对主节点、从节点、其他Sentinel节点发送ping命令做心跳检测
当这些节点超过down-after-milliseconds时间没有进行有效回复
Sentinel 节点就会对该节点做失败判定，这个行为叫做主观下线
从字面意思也可以很容易看出主观下线
是当前Sentinel节点的一家之言，存在误判断的可能

3.3、客观下线

当Sentinel主观下线的节点是主节点时，
该Sentinel节点会通过Sentinel is-master-down-by-addr命令向其他Sentinel节点询问对主节点的判断，当超过<quorum> 个数（半数），Sentinel 节点都认为主节点确实有问题
这时该Sentinel节点会做出客观下线的决定，这样客观下线的含义是比较明显了
也就是大部分Sentinel节点都对主节点的下线做了同意的判定，
那么这个判定就是客观的

3.4、 领导者选举

故障转移的工作只需要一个Sentinel节点来完成即可，
所以Sentinel节点之间会做一个领导者选举的工作，
选出一个Sentinel节点作为领导者进行故障转移的工作。
Redis 使用了Raft算法实现领导者选举，
大致思路如下:
(1) 每个在线的Sentinel节点都有资格成为领导者（临时，一次性的）
当他确认主节点主观下线时
会向其他Sentinel节点发送sentinel is-master-down-by-addr.命令
要求将自己设置为领导者
(2) 收到命令的节点，如果没有同意过其他节点的请求，则会同意该请求，否则拒绝
(3) 如果某一个节点的票数已经大于等于max (quorum, num (sentinels) /2+1),

那么它将成为领导者

(4) 如果此过程没有选举出领导者，将进入下一次选举

3.5、故障转移

领导者选举出的Sentinel节点负责故障转移，具体步骤如下:

(1) 在从节点列表中选出一个节点作为新的主节点方法如下:

- 过滤: “不健康”(主观下线、断线)
- 5秒内没有回复过Sentinel节点ping响应
- 与主节点失联超过down-after- milliseconds*10秒
- 选择slave-priority (从优先级)最高的从节点列表，如果存在则返回，不存在则继续
- 选择复制偏移量最大的从节点
- 选择runid最小的从节点

(2) 领导者节点会对第一步选出来的从节点执行slaveofnoone命令，使其成为新主

(3) 领导者节点会向剩余的从节点发送命令，让他们成为新主的从节点

(4) Sentinel节点集合会将原来的主节点更新为从节点并保持这对其关注，

当其恢复后命令它去复制新的主节点

四、Sentinel 集群部署

4.1、 案例环境

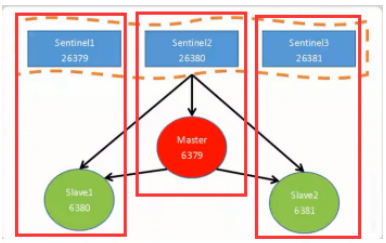
这里只是为了方便， 都放在一台机器上了，但在实际生产环境中，最好分散开，出于安全性考虑

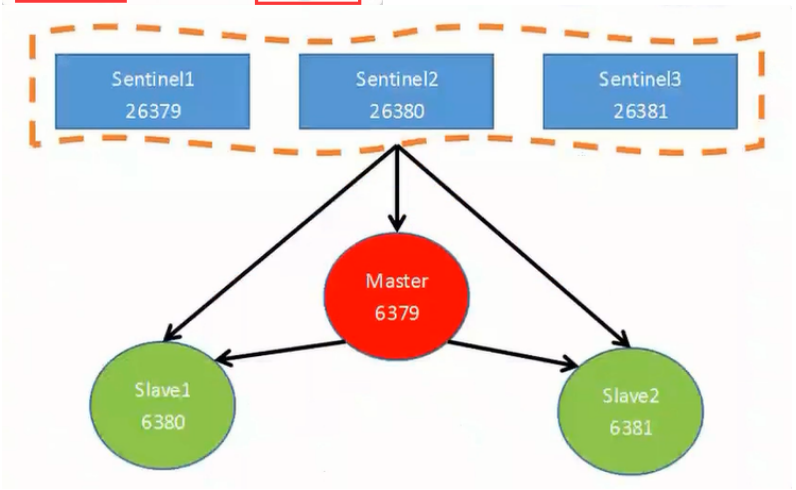
系统	IP地址	主机名	端口	redis角色
CentOS7.6	192.168.200.107	redis	6379	master
CentOS7.6	192.168.200.107	redis	6380	slave1
CentOS7.6	192.168.200.107	redis	6381	slave2
CentOS7.6	192.168.200.107	redis	26379	Sentinel
CentOS7.6	192.168.200.107	redis	26380	Sentinel
CentOS7.6	192.168.200.107	redis	26381	Sentinel

---

redis是单进程模式，对于CPU的利用率不高，建议开启redis多实例，这样可以保证多个进程同时使用CPU的资源;

如果是在真实的工作场景中，建议将这个集群拆分一下，如果都在一台机器上，风险高，出故障，损失大。





详细安装请见4-28

安装并配置 redis 服务及实例

```
[root@localhost ~]# hostname redis
[root@localhost ~]# bash
[root@redis ~]# iptables -F
[root@redis ~]# systemctl stop firewalld
[root@redis ~]# setenforce 0
setenforce: SELinux is disabled
```

下载 redis 源码包，解压并进入解压路径，从主机进行同样的安装配置

```
[root@redis ~]# wget http://download.redis.io/releases/redis-4.0.10.tar.gz
[root@redis ~]# tar xf redis-4.0.10.tar.gz -C /usr/src/

[root@redis ~]# cd /usr/src/redis-4.0.10/
```

执行 make 编译安装，从主机进行同样的安装配置

在我们之前的学习中，我们都会执行./configure 进行编译从而生成 Makefile，redis 的源码包中则直接提供了 Makefile 文件所以在解压完成之后直接进行 make 编译即可）

```
[root@redis redis-4.0.10]# ls | grep --color Makefile          #可以看到解压路径下自带的
Makefile
[root@redis redis-4.0.10]# make
Hint: It's a good idea to run 'make test' ;)
make[1]: 离开目录“/usr/src/redis-4.0.10/src”     #显示如下信息表示编译安装成功
[root@redis redis-4.0.10]# make install
```

redis 其实可以不用执行 make install 但是为了后面方便我们执行所以执行 make install

在安装过程中，若想更改默认的安装路径，可使用以下命令格式来进行安装操作。

make PREFIX=安装路径 install

生成启动脚本及配置文件信息

make install 只是生成了二进制文件到系统中，并没有启动脚本和配置文件，所以此时使用 redis 默认提供的 install_server.sh 脚本进行生成启动脚本和配置文件。

```
[root@redis redis-4.0.10]# cd utils/
[root@redis utils]#        #在解压路径下的 utils 中

配置 master 实例
[root@redis utils]# ./install_server.sh      #执行安装脚本
Welcome to the redis service installer
This script will help you easily set up a running redis server
#一路回车即可
Please select the redis port for this instance: [6379]                    #设置服务端口号
Selecting default: 6379
Please select the redis config file name [/etc/redis/6379.conf]       #设置主配置文件存放位置
Selected default - /etc/redis/6379.conf
Please select the redis log file name [/var/log/redis_6379.log]       #设置 reids 日志存放位置
Selected default - /var/log/redis_6379.log
Please select the data directory for this instance [/var/lib/redis/6379]        #设置数据目录
Selected default - /var/lib/redis/6379
Please select the redis executable path [/usr/local/bin/redis-server]          #设置执行命令
Selected config:
Port               : 6379                          //服务端口
Config file        : /etc/redis/6379.conf          //配置文件
Log file           : /var/log/redis_6379.log       //日志文件
Data dir           : /var/lib/redis/6379           //数据目录
Executable         : /usr/local/bin/redis-server   //服务端命令
Cli Executable : /usr/local/bin/redis-cli          //客户端命令
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/6379.conf => /etc/init.d/redis_6379
```

```
[root@master ~]# killall -9 redis-server
[root@master ~]# rm -rf /var/run/redis_6379.pid
[root@master ~]# rm -rf /var/lib/redis/6379/
appendonly.aof  dump.rdb
[root@master ~]# rm -rf /var/lib/redis/6379/appendonly.aof
[root@master ~]# rm -rf /var/lib/redis/6379/dump.rdb
[root@master ~]# ls /var/lib/redis/6379/
[root@master ~]# cd
```

```
[root@master ~]# /etc/init.d/redis_6379 start
Starting Redis server...
```

配置slave2实例：

```
[root@master ~]# cd /usr/src/redis-4.0.10/utils/
[root@master utils]# ls
build-static-symbols.tcl  graphs             redis-copy.rb          speed-
regression.tcl
cluster_fail_time.tcl     hashtable          redis_init_script
whatisdoing.sh
corrupt_rdb.c             hyperloglog        redis_init_script.tpl
create-cluster            install_server.sh  redis-sha1.rb
generate-command-help.rb  lru                releasetools
[root@master utils]# ./install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379] 6380
Please select the redis config file name [/etc/redis/6380.conf]
Selected default - /etc/redis/6380.conf
Please select the redis log file name [/var/log/redis_6380.log]
Selected default - /var/log/redis_6380.log
Please select the data directory for this instance [/var/lib/redis/6380]
Selected default - /var/lib/redis/6380
Please select the redis executable path [/usr/local/bin/redis-server]
Selected config:
Port           : 6380
Config file    : /etc/redis/6380.conf
Log file       : /var/log/redis_6380.log
Data dir       : /var/lib/redis/6380
Executable     : /usr/local/bin/redis-server
Cli Executable : /usr/local/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/6380.conf => /etc/init.d/redis_6380
Installing service...
Successfully added to chkconfig!
```

```
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!
[root@master utils]# ./install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379] 6381
Please select the redis config file name [/etc/redis/6381.conf]
Selected default - /etc/redis/6381.conf
Please select the redis log file name [/var/log/redis_6381.log]
Selected default - /var/log/redis_6381.log
Please select the data directory for this instance [/var/lib/redis/6381]
Selected default - /var/lib/redis/6381
Please select the redis executable path [/usr/local/bin/redis-server]
Selected config:
Port           : 6381
Config file    : /etc/redis/6381.conf
Log file       : /var/log/redis_6381.log
Data dir       : /var/lib/redis/6381
Executable     : /usr/local/bin/redis-server
Cli Executable : /usr/local/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/6381.conf => /etc/init.d/redis_6381
Installing service...
Successfully added to chkconfig!
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!
[root@master utils]# netstat -lnpt |grep redis
tcp        0      0 192.168.200.107:6379    0.0.0.0:*               LISTEN
95174/redis-server
tcp        0      0 127.0.0.1:6380          0.0.0.0:*               LISTEN
95409/redis-server
```

```
tcp        0      0 127.0.0.1:6381          0.0.0.0:*               LISTEN
95486/redis-server
```

对6380和6381进行优化

```
[root@master utils]#  sed -i '/^bind 127.0.0.1$/s/127.0.0.1/192.168.200.107/g'
/etc/redis/6380.conf
[root@master utils]# sed -i '/protected-mode/s/yes/no/g' /etc/redis/6380.conf
[root@master utils]# sed -i '/daemonize/s/no/yes/g' /etc/redis/6380.conf
[root@master utils]# sed -i '/requirepass/s/foobared/123456/g'
/etc/redis/6380.conf
[root@master utils]# service redis_6380 restart
Stopping ...
Redis stopped
Starting Redis server...
[root@master utils]# sed -i '/^bind 127.0.0.1$/s/127.0.0.1/192.168.200.108/g'
/etc/redis/6381.conf
[root@master utils]# sed -i '/^bind 127.0.0.1$/s/127.0.0.1/192.168.200.107/g'
/etc/redis/6381.conf
[root@master utils]# sed -i '/protected-mode/s/yes/no/g' /etc/redis/6381.conf
[root@master utils]# sed -i '/daemonize/s/no/yes/g' /etc/redis/6381.conf
[root@master utils]# sed -i '/requirepass/s/foobared/123456/g'
/etc/redis/6381.conf
[root@master utils]# service redis_6381 restart
Stopping ...
Redis stopped
Starting Redis server...
[root@redis ~]# !net
netstat -lnpt | grep redis
tcp        0      0 192.168.200.107:6379    0.0.0.0:*               LISTEN
6635/redis-server 1
tcp        0      0 192.168.200.107:6380    0.0.0.0:*               LISTEN
6667/redis-server 1
tcp        0      0 192.168.200.107:6381    0.0.0.0:*               LISTEN
6701/redis-server 1
```

验证redis实例运行状态：免交互

```
[root@redis ~]# redis-cli -h 192.168.200.107 -a 123456 -p 6379 ping
```

```
Warning: Using a password with '-a' option on the command line interface may not be safe.
PONG
[root@redis ~]# redis-cli -h 192.168.200.107 -a 123456 -p 6380 ping
Warning: Using a password with '-a' option on the command line interface may not be safe.
PONG
[root@redis ~]# redis-cli -h 192.168.200.107 -a 123456 -p 6381 ping
Warning: Using a password with '-a' option on the command line interface may not be safe.
PONG
[root@redis ~]# vim /etc/redis/6379.conf
  70 bind 192.168.200.107
  89 protected-mode no
 137 daemonize yes
 501  requirepass 123456
 451 min-slaves-to-write 2
 452 min-slaves-max-lag 10
[root@redis ~]# vim /etc/redis/6380.conf
 282  slaveof 192.168.200.107 6379
 289  masterauth 123456
 451 min-slaves-to-write 2
 452 min-slaves-max-lag 10
[root@redis ~]# vim /etc/redis/6381.conf
 282  slaveof 192.168.200.107 6379
 289  masterauth 123456
 451 min-slaves-to-write 2
 452 min-slaves-max-lag 10
[root@redis ~]# killall -9 redis-server
[root@redis ~]# rm -rf /var/run/redis_6379.pid
[root@redis ~]# rm -rf /var/run/redis_6380.pid
[root@redis ~]# rm -rf /var/run/redis_6381.pid
[root@redis ~]# /etc/init.d/redis_6379 start
Starting Redis server...
[root@redis ~]# /etc/init.d/redis_6380 start
```

```
Starting Redis server...
[root@redis ~]# /etc/init.d/redis_6381 start
Starting Redis server...
[root@redis ~]# redis-cli -h 192.168.200.107 -a 123456 -p 6379
Warning: Using a password with '-a' option on the command line interface may not be safe.
192.168.200.107:6379> set name vee
OK
192.168.200.107:6379> get name
"vee"
192.168.200.107:6379> keys *
1) "name"
192.168.200.107:6379> info replication
# Replication
role:master
connected_slaves:2
min_slaves_good_slaves:2
slave0:ip=192.168.200.107,port=6380,state=online,offset=1049,lag=1
slave1:ip=192.168.200.107,port=6381,state=online,offset=1049,lag=0
master_replid:09fb2474247d8fd5aa30167c5ba77244b93955f6
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:1049
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:1049
```

登录slave节点验证键值同步情况，并测试无法写入

```
[root@redis ~]# redis-cli -h 192.168.200.107 -a 123456 -p 6380
192.168.200.107:6380> keys *
1) "name"
192.168.200.107:6380> get name
"vee"
192.168.200.107:6380> info replication
[root@redis ~]# redis-cli -h 192.168.200.107 -a 123456 -p 6381
```

```
192.168.200.107:6381> keys *
1) "name"
192.168.200.107:6381> get name
"vee"
192.168.200.107:6381> info replication
```

4.4、配置Redis sentinel集群

Sentinel节点本身就是独立的Redis节点，只不过它们有一些特殊，

们不存储数据只支持部分命令

准备sentinel日志目录

```
[root@redis ~]# mkdir -p /var/log/redis/              # 保存日志
```

配置Sentinel1节点：

```
[root@redis ~]# vim /etc/redis/redis-sentinel-26379.conf
port 26379
dir "/var/log/redis"
logfile "26379.log"
daemonize yes
protected-mode no
sentinel monitor mymaster 192.168.200.107 6379 2
sentinel down-after-milliseconds mymaster 30000
sentinel parallel-syncs mymaster 1
sentinel failover-timeout mymaster 180000
sentinel auth-pass mymaster 123456
[root@redis ~]# redis-sentinel /etc/redis/redis-sentinel-26379.conf
# 启动
```

一个哨兵搞定~

```
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26379 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=192.168.200.107:6379,slaves=2,sentinels=1
[root@redis ~]# vim /etc/redis/redis-sentinel-26380.conf
port 26380
```

```
dir "/var/log/redis"
logfile "26380.log"
daemonize yes
protected-mode no
sentinel monitor mymaster 192.168.200.107 6379 2
sentinel down-after-milliseconds mymaster 30000
sentinel parallel-syncs mymaster 1
sentinel failover-timeout mymaster 180000
sentinel auth-pass mymaster 123456
[root@redis ~]# redis-sentinel /etc/redis/redis-sentinel-26380.conf
[root@redis ~]# redis-sentinel /etc/redis/redis-sentinel-26381.conf
[root@redis ~]# !net
netstat -lnpt | grep redis
tcp        0      0 0.0.0.0:26379           0.0.0.0:*
LISTEN      7683/redis-sentinel
tcp        0      0 192.168.200.107:6379    0.0.0.0:*               LISTEN
7141/redis-server 1
tcp        0      0 0.0.0.0:26380           0.0.0.0:*
LISTEN      7770/redis-sentinel
tcp        0      0 192.168.200.107:6380    0.0.0.0:*               LISTEN
7155/redis-server 1
tcp        0      0 0.0.0.0:26381           0.0.0.0:*
LISTEN      7783/redis-sentinel
tcp        0      0 192.168.200.107:6381    0.0.0.0:*               LISTEN
7162/redis-server 1
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26379 info Sentinel
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26380 info Sentinel
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26381 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=192.168.200.107:6379,slaves=2,sentinels=3
```

```
[root@redis ~]# vim /etc/redis/redis-sentinel-26381.conf
port 26381
dir "/var/log/redis"
logfile "26381.log"
daemonize yes
protected-mode no
sentinel myid a14c5c291c7936a7883080ffcfec9226c0ae88b6
sentinel monitor mymaster 192.168.200.107 6379 2
sentinel auth-pass mymaster 123456
sentinel config-epoch mymaster 0
sentinel leader-epoch mymaster 0
# Generated by CONFIG REWRITE
sentinel known-slave mymaster 192.168.200.107 6381
sentinel known-slave mymaster 192.168.200.107 6380
sentinel known-sentinel mymaster 192.168.200.107 26380
1f144907f4ce4a900f746d1d8f975e123e664dc8
sentinel known-sentinel mymaster 192.168.200.107 26379
c672aa23911073d3f746b819e98a33a206b53c87
sentinel current-epoch 0
```

观察发现，后面多了两行，主要体现在三个方面：

1.sentinel节点自动发现从节点及其他的sentinel节点，

2.而且会去掉文件原来的配置，会自动的对文件自动的变化

emmmmmm 第三个还没说

配置文件参数的含义：

```
daemonize yes   #以守护进程模式启动;
protected-mode no  #在配置多个sentinel时需在每个sentinel配置文件中必须添加，
                    否则主的redis挂掉时，所有的sentinel丕会选举出新的master
sentinel monitor mymaster 192.168.200.111 6379 2
   #sentinel去监听地址为192.168.200.107 6379的master;
     mymaster 是集群名称，名称可以自定义;
     quorum 是一一个数字，指明当有多少个sentinel认为一个master失效时，master 才算真正失效;
     master-name只能包含英文字母，数字，和"“_”这三个字符。
```

需要注意的是master-ip要写真实的ip地址而不要用回环地址(127.0.0.1)。

sentinel down-after- milliseconds mymaster 3000

这个配置项指定需要多少失效时间，

一个master才会被这个sentinel主观地认为是不可用的。

单位是毫秒， 默认为30秒

sentinel parallel-syncs mymaster 1

#这个配置项指定了在发生failover 主备切换时最多可以有多少个slave同时对新的master进行同步

这个数字越小，完成failover所需的时间就越长

但是如果这个数字越大，就意味着越多的slave 因为replication而不可用

可以通过将这个值设为1来保证每次只有一个slave处于不能处理命令请求的状态

sentinel failover -timeout mymaster 180000 #故障转移超时时间为180000

sentinel auth-pass mymaster 123456

#设置连接 master 和slave 时的密码，

注意的是sentinel不能分别为master和slave设置不同的密码

因此master和slave的密码应该设置相同

4.6、 Sentinel 集群测试

查看slaves信息：

[root@redis ~]# redis-cli -h 192.168.200.107 -p 6380 -a 123456 info replication

Warning: Using a password with '-a' option on the command line interface may not be safe.

Replication

role:slave

master_host:192.168.200.107

master_port:6379

master_link_status:up

master_last_io_seconds_ago:0

master_sync_in_progress:0

slave_repl_offset:382721

slave_priority:100

slave_read_only:1

connected_slaves:0

min_slaves_good_slaves:0

master_replid:09fb2474247d8fd5aa30167c5ba77244b93955f6

master_replid2:0000000000000000000000000000000000000000

```
master_repl_offset:382721
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:382721
[root@redis ~]# redis-cli -h 192.168.200.107 -p 6381 -a 123456 info replication
Warning: Using a password with '-a' option on the command line interface may not be safe.
# Replication
role:slave
master_host:192.168.200.107
master_port:6379
master_link_status:up
master_last_io_seconds_ago:0
master_sync_in_progress:0
slave_repl_offset:386519
slave_priority:100
slave_read_only:1
connected_slaves:0
min_slaves_good_slaves:0
master_replid:09fb2474247d8fd5aa30167c5ba77244b93955f6
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:386519
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:386519
```

模拟master出现故障:

```
[root@redis ~]# netstat -lnpt | grep 6379
tcp        0      0 0.0.0.0:26379           0.0.0.0:*               LISTEN      7683/redis-sentinel
tcp        0      0 192.168.200.107:6379    0.0.0.0:*               LISTEN      7141/redis-server 1
```

```
tcp6       0      0 :::26379                :::*                    LISTEN
7683/redis-sentinel
[root@redis ~]# kill -9 7141
[root@redis ~]# rm -rf /var/run/redis_6379.pid
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26381 info Sentinel
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26380 info Sentinel
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26389 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=192.168.200.107:6381,slaves=2,sentinels=3
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26381 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=192.168.200.107:6381,slaves=2,sentinels=3
[root@redis ~]# redis-cli -h 192.168.200.107 -p 6381 -a 123456 info
replication
Warning: Using a password with '-a' option on the command line interface may not
be safe.
# Replication
role:master
connected_slaves:0
min_slaves_good_slaves:0
master_replid:bce6e5f875bf51d552931e6a78ec88f1cdac983a
master_replid2:09fb2474247d8fd5aa30167c5ba77244b93955f6
master_repl_offset:466011
second_repl_offset:410744
repl_backlog_active:1
```

repl_backlog_size:1048576

repl_backlog_first_byte_offset:1

repl_backlog_histlen:466011

[root@redis ~]# redis-cli -h 192.168.200.107 -p 6380 -a 123456 info replication

Warning: Using a password with '-a' option on the command line interface may not be safe.

Replication

role:slave

master_host:192.168.200.107

master_port:6381

master_link_status:down

master_last_io_seconds_ago:-1

master_sync_in_progress:0

slave_repl_offset:410743

master_link_down_since_seconds:1588544862

slave_priority:100

slave_read_only:1

connected_slaves:0

min_slaves_good_slaves:0

master_replid:09fb2474247d8fd5aa30167c5ba77244b93955f6

master_replid2:0000000000000000000000000000000000000000

master_repl_offset:410743

second_repl_offset:-1

repl_backlog_active:1

repl_backlog_size:1048576

repl_backlog_first_byte_offset:1

repl_backlog_histlen:410743

故障修复:

[root@redis ~]# /etc/init.d/redis_6379 start

Starting Redis server...

[root@redis ~]# redis-cli -h 192.168.200.107 -p 6379 -a 123456 info replication

Warning: Using a password with '-a' option on the command line interface may not be safe.

```
# Replication
role:slave
master_host:192.168.200.107
master_port:6381
master_link_status:up
master_last_io_seconds_ago:1
master_sync_in_progress:0
slave_repl_offset:507836
slave_priority:100
slave_read_only:1
connected_slaves:0
min_slaves_good_slaves:0
master_replid:bce6e5f875bf51d552931e6a78ec88f1cdac983a
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:507836
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:502117
repl_backlog_histlen:5720
[root@redis ~]# redis-cli -h 192.168.200.107 -p 26379 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=192.168.200.107:6381,slaves=2,sentinels=3
```

Sentinel集群总结

- 对于 master节点故障判断是由多个Sentinel节点共同完成，这样可以有效地防止误判
- Sentinel集群可由若干个（奇数）Sentinel 节点组成的，这样即使个别Sentinel节点不可用，整个Sentinel节点集合依然是健壮的

redis哨兵集群高可用