一、复制:

1. 主从复制

a,方式一、新增 redis6380.conf, 加入 slaveof 192.168.42.111 6379, 在 6379 启动完后再启 6380，完成配置；

b,方式二、redis-server --slaveof 192.168.42.111 6379

c,查看状态：info replication

d,断开主从复制：在 slave 节点，执行 6380:>slaveof no one

e,断开后再变成主从复制：6380:> slaveof 192.168.42.111 6379

f,数据较重要的节点，主从复制时使用密码验证： requirepass

e,从节点建议用只读模式 slave-read-only=yes, 若从节点修改数据，主从数据不一致

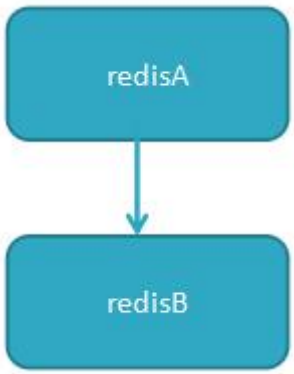
h,传输延迟：主从一般部署在不同机器上，复制时存在网络延时问题，redis 提供 repl-disable-tcp-nodelay 参数决定是否关闭 TCP_NODELAY,默认为关闭

参数关闭时：无论大小都会及时发布到从节点，占带宽，适用于主从网络好的场景，

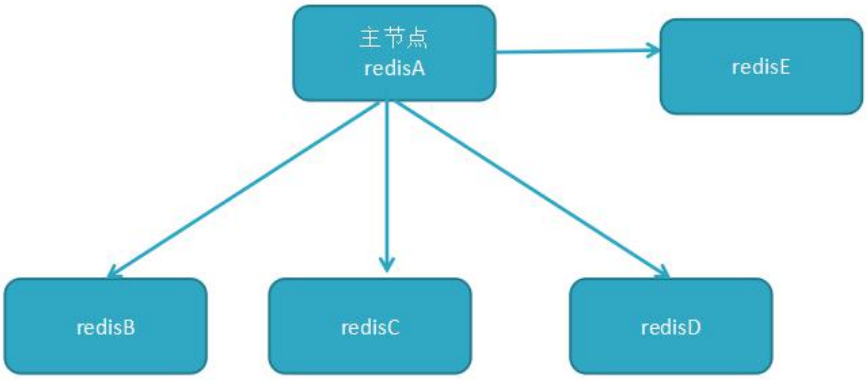
参数启用时：主节点合并所有数据成 TCP 包节省带宽，默认为 40 毫秒发一次，取决于内核，主从的同步延迟 40 毫秒，适用于网络环境复杂或带宽紧张，如跨机房

2，主从拓扑：支持单层或多层

A,一主一从：用于主节点故障转移从节点，当主节点的“写”命令并发高且需要持久化，可以只在从节点开启 AOF（主节点不需要），这样即保证了数据的安全性，也避免持久化对主节点的影响

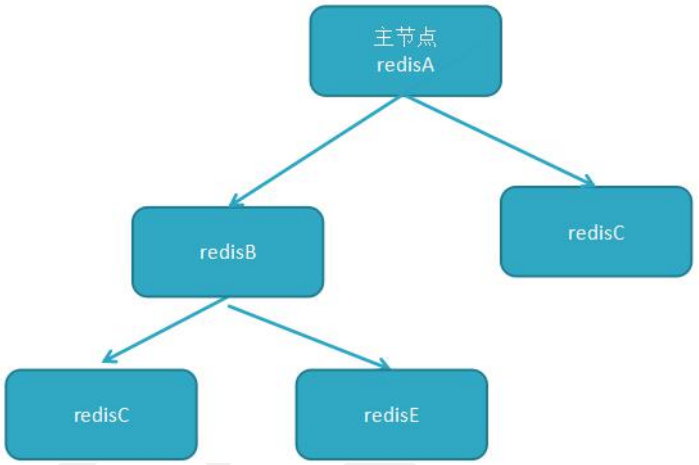


B, 一主多从：针对“读”较多的场景，“读”由多个从节点来分担，但节点越多，主节点同步到多节点的次数也越多，影响带宽，也加重主节点的稳定



C,树状主从：一主多从的缺点（主节点推送次数多压力大）可用些方案解决，

主节点只推送一次数据到从节点 1，再由从节点 2 推送到 11，减轻主节点推送的压力。



2. 复制原理

执行 slave master port 后，

与主节点连接，同步主节点的数据,6380:>info replication：查看主从及同步信息

3，数据同步：

redis 2.8 版本以上使用 psync 命令完成同步，过程分“全量”与“部分”复制

全量复制：一般用于初次复制场景（第一次建立 SLAVE 后全量）

部分复制：网络出现问题，从节点再次连主时，主节点补发缺少的数据，每次数据增加同步

心跳：主从有长连接心跳，主节点默认每 10S 向从节点发 ping 命令，repl-ping-slave-period 控制发送频率

二、哨兵机制:

1，为什么要讲哨兵机制？

A，我们学习了 redis 的主从复制，但如果说主节点出现问题不能提供服务，需要人工重新把从节点设为主节点，还要通知我们的应用程序更新了主节点的地址，这种处理方式不是科学的，耗时费事

B，同时主节点的写能力是单机的，能力能限

C，而且主节点是单机的，存储能力也有限

其中 2，3 的问题在后面 redis 集群课会讲，第 1 个问题我们用哨兵机制来解决

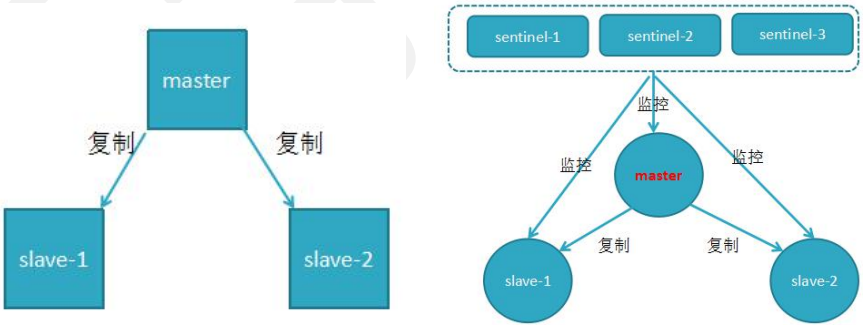
2,主从故障如何故障转移(不满足高可用)：

A，主节点(master)故障，从节点 slave-1 端执行 slaveof no one 后变成新主节点

B，其它的节点成为新主节点的从节点，并从新节点复制数据

3，哨兵机制(sentinel)的高可用：

A，原理：当主节点出现故障时，由 redis sentinel 自动完成故障发现和转移，并通知应用方，实现高可用性。

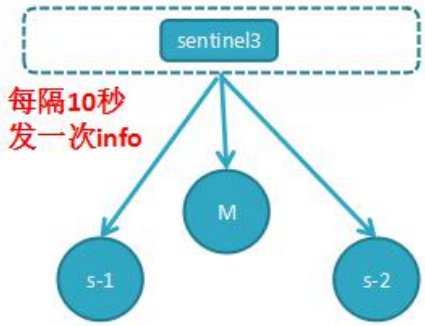


主从复制与 redis sentinel 拓扑结构图

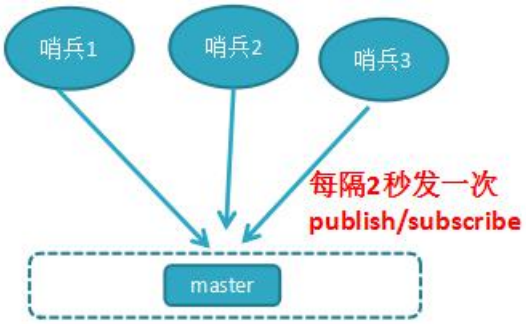
其实整个过程只需要一个哨兵节点来完成，首先使用 Raft 算法（感兴趣的同学可以查一下，其实就是个选举算法）实现选举机制，选出一个哨兵节点来完成转移和通知

哨兵有三个定时监控任务完成对各节点的发现和监控：

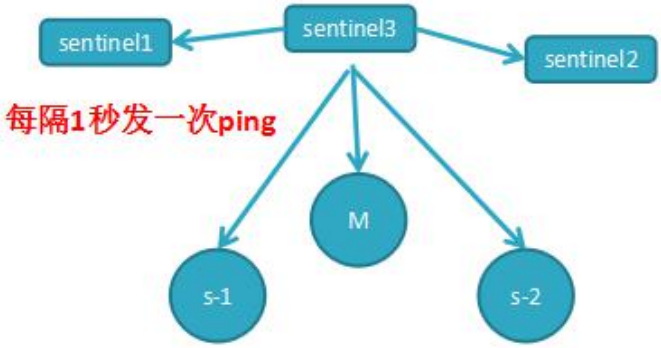
任务 1，每个哨兵节点每 10 秒会向主节点和从节点发送 info 命令获取最拓扑结构图，哨兵配置时只要配置对主节点的监控即可，通过向主节点发送 info，获取从节点的信息，并当有新的从节点加入时可以马上感知到



任务 2，每个哨兵节点每隔 2 秒会向 redis 数据节点的指定频道上发送该哨兵节点对于主节点的判断以及当前哨兵节点的信息，同时每个哨兵节点也会订阅该频道，来了解其它哨兵节点的信息及对主节点的判断，其实就是通过消息 publish 和 subscribe 来完成的；

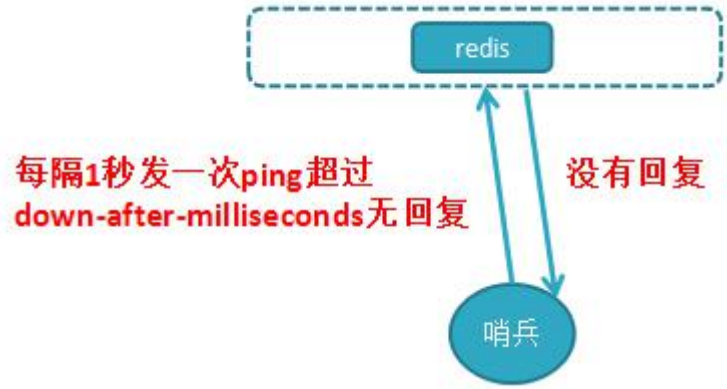


任务 3，每隔 1 秒每个哨兵会向主节点、从节点及其余哨兵节点发送一次 ping 命令做一次心跳检测，这个也是哨兵用来判断节点是否正常的重要依据

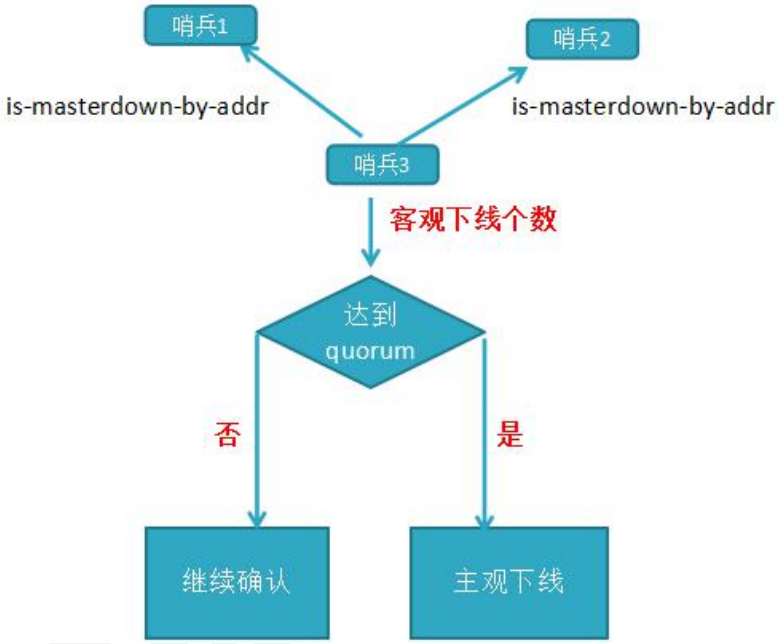


主观下线和客观下线：

主观下线：刚我知道知道哨兵节点每隔 1 秒对主节点和从节点、其它哨兵节点发送 ping 做心跳检测，当这些心跳检测时间超过 down-after-milliseconds 时，哨兵节点则认为该节点错误或下线，这叫主观下线；这可能会存在错误的判断。



客观下线：当主观下线的节点是主节点时，此时该哨兵 3 节点会通过指令 sentinel is-masterdown-by-addr 寻求其它哨兵节点对主节点的判断，当超过 quorum（法定人数）个数，此时哨兵节点则认为该主节点确实有问题，这样就客观下线了，大部分哨兵节点都同意下线操作，也就说是客观下线

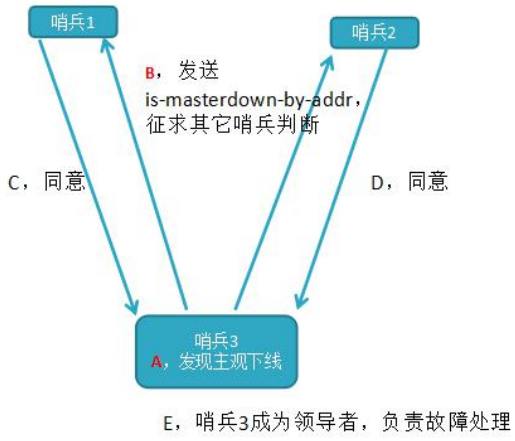


领导者哨兵选举流程：

a,每个在线的哨兵节点都可以成为领导者，当它确认（比如哨兵 3）主节点下线时，会向其它哨兵发 is-master-down-by-addr 命令，征求判断并要求将自己设置为领导者，由领导者处理故障转移；

b,当其它哨兵收到此命令时，可以同意或者拒绝它成为领导者；

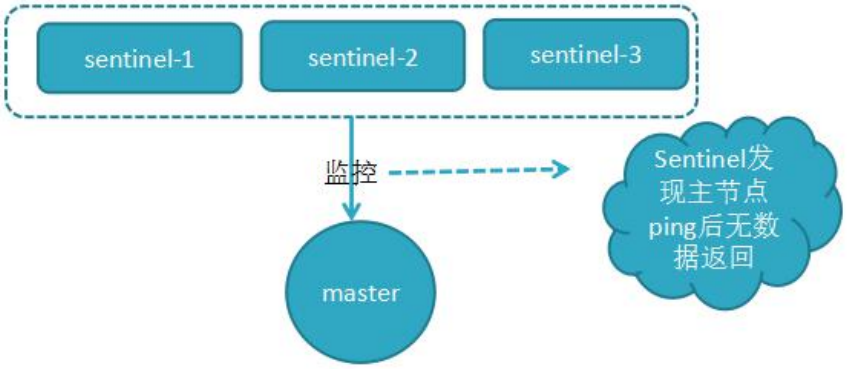
c,如果哨兵 3 发现自己在选举的票数大于等于 num(sentinels)/2+1 时，将成为领导者，如果没有超过，继续选举……



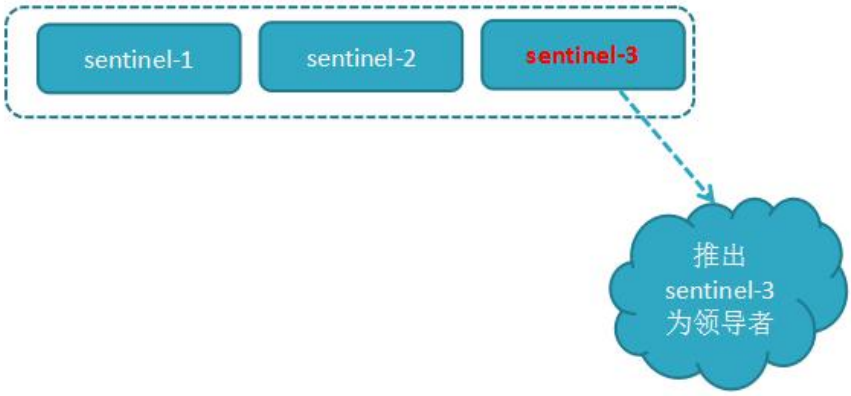
4，故障转移机制

A，由 Sentinel 节点定期监控发现主节点是否出现了故障

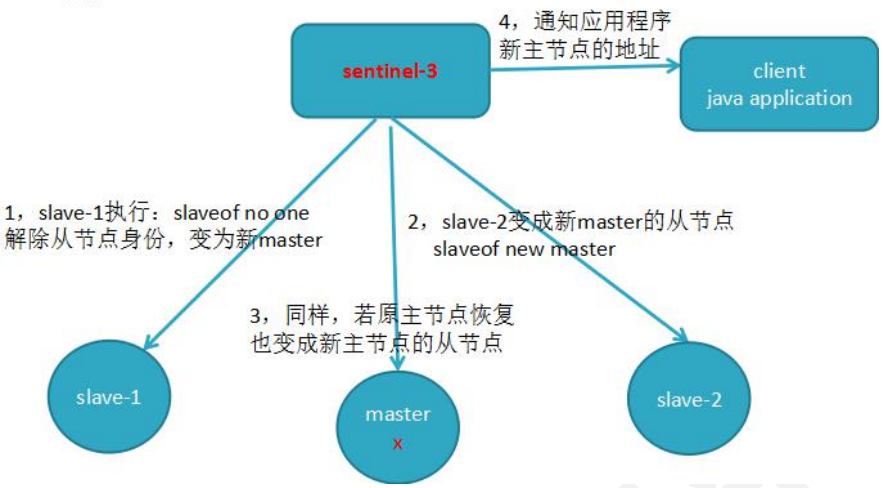
sentinel 会向 master 发送心跳 PING 来确认 master 是否存活，如果 master 在“一定时间范围”内不回应 PONG 或者是回复了一个错误消息，那么这个 sentinel 会主观地(单方面地)认为这个 master 已经不可用了



B，当主节点出现故障，此时 3 个 Sentinel 节点共同选举了 Sentinel3 节点为领导，负载处理主节点的故障转移，



C，由 Sentinel3 领导者节点执行故障转移，过程和主从复制一样，但是自动执行



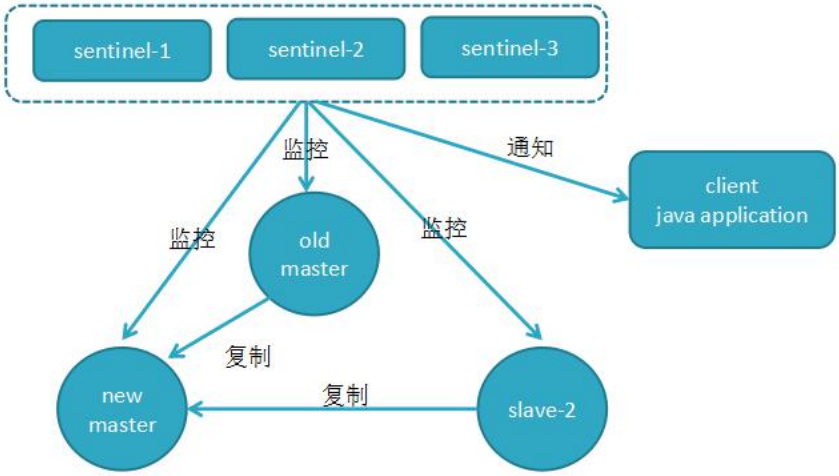
流程： 1，将 slave-1 脱离原从节点，升级主节点，

2，将从节点 slave-2 指向新的主节点

3，通知客户端主节点已更换

4，将原主节点（oldMaster）变成从节点，指向新的主节点

D，故障转移后的 redis sentinel 的拓扑结构图

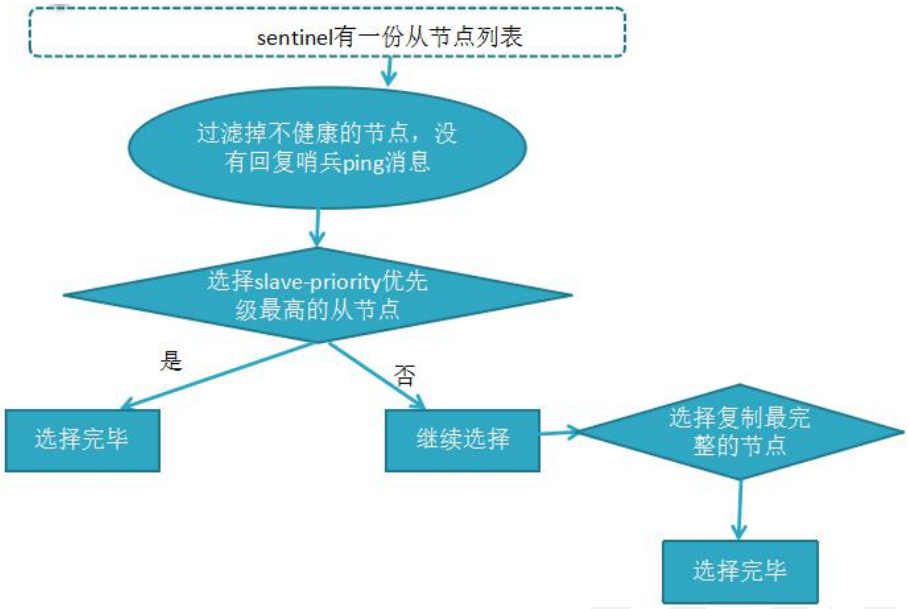


5，哨兵机制－故障转移详细流程

A，过滤掉不健康的（下线或断线），没有回复过哨兵 ping 响应的从节点

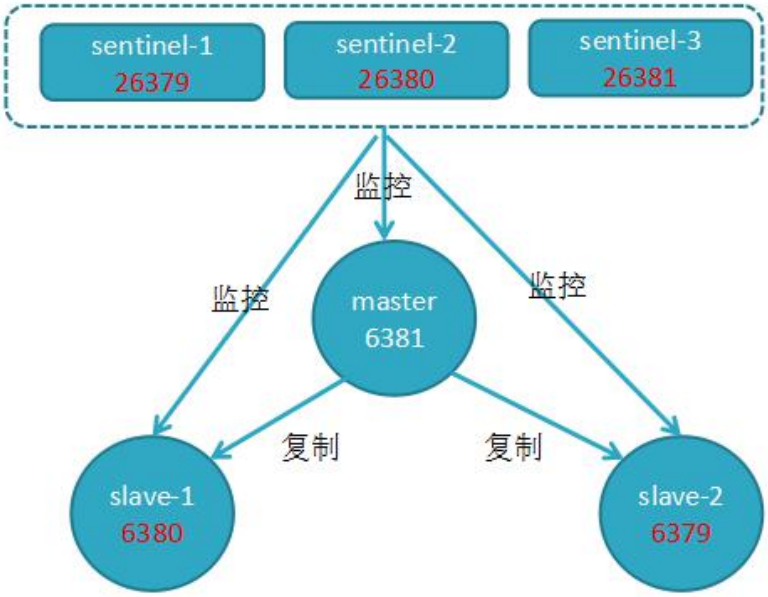
B，选择 slave-priority 从节点优先级最高（redis.conf）

C，选择复制偏移量最大，指复制最完整的从节点



5，如何安装和部署 Reids Sentinel?

我们以 3 个 Sentinel 节点、2 个从节点、1 个主节点为例进行安装部署



1，前提：先搭好一主两从 redis 的主从复制，和之前复制搭建一样，搭建方式如下：

A 主节点 6379 节点（/usr/local/bin/conf/redis6379.conf）：

修改 requirepass 12345678，注释掉#bind 127.0.0.1

B 从节点 redis6380.conf 和 redis6381.conf:

修改 requirepass 12345678 ,注释掉#bind 127.0.0.1,

加上 masterauth 12345678 ,加上 slaveof 127.0.0.1 6379

注意：当主从起来后，主节点可读写，从节点只可读不可写

2，redis sentinel 哨兵机制核心配置(也是 3 个节点)：

/usr/local/bin/conf/sentinel_26379.conf

/usr/local/bin/conf/sentinel_26380.conf

/usr/local/bin/conf/sentinel_26381.conf

将三个文件的端口改成: 26379　26380　26381

然后：sentinel monitor mymaster 190.168.1.111 **6379** 2　//监听主节点 6379

sentinel auth-pass mymaster 12345678　　//连接主节点时的密码

三个配置除端口外，其它一样。

3，哨兵其它的配置：只要修改每个 sentinel.conf 的这段配置即可：

sentinel monitor mymaster 192.168.1.10 6379 2

//监控主节点的 IP 地址端口，sentinel 监控的 master 的名字叫做 mymaster

2 代表，当集群中有 2 个 sentinel 认为 master 死了时，才能真正认为该 master 已经不可用了

sentinel auth-pass mymaster 12345678　//sentinel 连主节点的密码

sentinel config-epoch mymaster 2　//故障转移时最多可以有 2 从节点同时对新主节点进行数据同步

sentinel leader-epoch mymaster 2

sentinel failover-timeout mymasterA **180000** //故障转移超时时间 180s，

a,如果转移超时失败，下次转移时时间为之前的 2 倍；

b,从节点变主节点时，从节点执行 slaveof no one 命令一直失败的话，当时间超过 **180S** 时，则故障转移失败

c,从节点复制新主节点时间超过 **180S** 转移失败

sentinel down-after-milliseconds mymasterA **300000**//sentinel 节点定期向主节点 ping 命令，当超过了 **300S** 时间后没有回复，可能就认定为此主节点出现故障了……

sentinel parallel-syncs mymasterA **1** //故障转移后，**1** 代表每个从节点按顺序排队一个一个复制主节点数据，如果为 3，指 3 个从节点同时并发复制主节点数据，不会影响阻塞，但存在网络和 IO 开销

4，启动 sentinel 服务:

./redis-sentinel conf/sentinel_26379.conf &

./redis-sentinel conf/sentinel_26380.conf &

./redis-sentinel conf/sentinel_26381.conf &

关闭：./redis-cli -h 192.168.42.111 -p 26379 shutdown

5，测试：kill -9 6379　杀掉 6379 的 redis 服务

看日志是分配 6380 还是 6381 做为主节点，当 6379 服务再启动时，已变成从节点

假设 6380 升级为主节点:进入 6380>info replication　　可以看到 role:master

打开 sentinel_26379.conf 等三个配置，sentinel monitor mymaster 127.0.0.1 **6380** 2

打开 redis6379.conf 等三个配置, slaveof 192.168.42.111 6380,也变成了 6380

注意：生产环境建议让 redis Sentinel 部署到不同的物理机上。

重要：sentinel monitor mymaster **192.168.42.111** 6379 2 //**切记将 IP 不要写成 127.0.0.1**

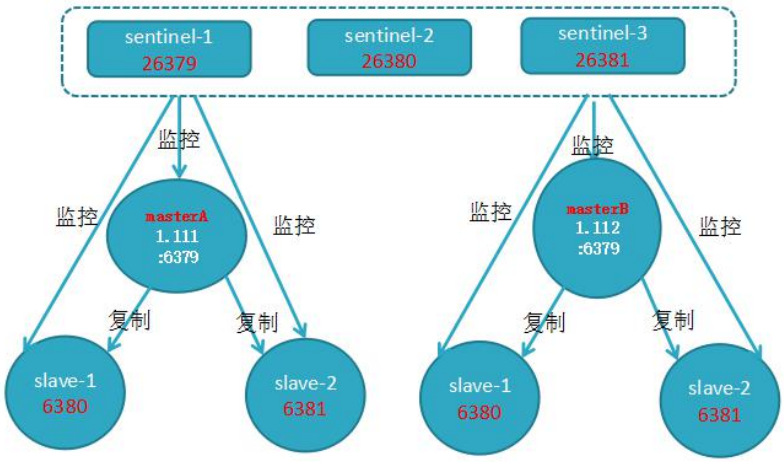
不然使用 JedisSentinelPool 取 jedis 连接的时候会变成取 127.0.0.1 6379 的错误地址

注：我们稍后要启动四个 redis 实例，其中端口为 6379 的 redis 设为 master，其他

两个设为 slave 。所以 mymaster 后跟的是 master 的 ip 和端口，最后一个’2’代表只要有 2 个 sentinel 认为 master 下线，就认为该 master 客观下线，选举产生新的 master。通常最后一个参数不能多于启动的 sentinel 实例数。

哨兵 sentinel 个数为奇数，选举嘛，奇数哨兵个才能选举成功，一般建议 3 个

6，RedisSentinel 如何监控 2 个 redis 主节点呢？



sentinel monitor mymasterB 192.168.1.20 6379 2
……与上面一样…….。

7，部署建议：

a，sentinel 节点应部署在多台物理机（线上环境）

b，至少三个且奇数个 sentinel 节点

c，通过以上我们知道，3 个 sentinel 可同时监控一个主节点或多个主节点

监听 N 个主节点较多时，如果 sentinel 出现异常，会对多个主节点有影响，同时还会造成 sentinel 节点产生过多的网络连接，

一般线上建议还是， 3 个 sentinel 监听一个主节点

8，sentinel 哨兵的 API

命令：redis-cli -p 26379 //进入哨兵的命令模式，使用 redis-cli 进入

26379>sentinel masters 或 sentinel master mymaster //查看 redis 主节点相关信息

26379>sentinel slaves mymaster //查看从节点状态与相关信息

26379>sentinel sentinels mymaster //查 sentinel 节点集合信息(不包括当前 26379)

26379>sentinel failover mymaster //对主节点强制故障转移，没和其它节点协商

9，客户端连接（redis-sentinel 例子工程）

远程客户端连接时，要打开 protected-mode no

./redis-cli -p 26380 shutdown //关闭

在使用工程 redis-sentinel，调用 jedis 查询的流程如下：

1，将三个 sentinel 的 IP 和地址加入 JedisSentinelPool

2，根据 IP 和地址创建 JedisSentinelPool 池对象

3，在这个对象创建完后，此时该对象已把 redis 的主节点（此时 sentinel monitor mymaster 必须写成 **192.168.42.111** 6379 2，不能为 127.0.0.1，不然查询出来的主节点的 IP 在客户端就变成了 127.0.0.1，拿不到连接了）查询出来了，当客户准备发起查询请求时，调用 pool.getResource()借用一个 jedis 对象，内容包括主节点的 IP 和端口；

4，将得到 jedis 对象后，可执行 jedis.get("age")指令了……。

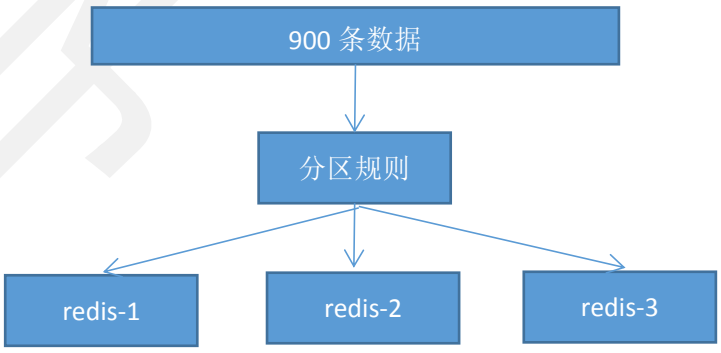
三、redis 集群:

RedisCluster 是 redis 的分布式解决方案，在 3.0 版本后推出的方案，有效地解决了 Redis 分布式的需求，当遇到单机内存、并发等瓶颈时，可使用此方案来解决这些问题

8.1 分布式数据库概念：

1，分布式数据库把整个数据按分区规则映射到多个节点，即把数据划分到多个节点上，每个节点负责整体数据的一个子集

比如我们库有 900 条用户数据，有 3 个 redis 节点，将 900 条分成 3 份，分别存入到 3 个 redis 节点



2，分区规则：

常见的分区规则哈希分区和顺序分区，redis 集群使用了哈希分区，顺序分区暂用不到，不做具体说明；

rediscluster 采用了哈希分区的“虚拟槽分区”方式（哈希分区分节点取余、一

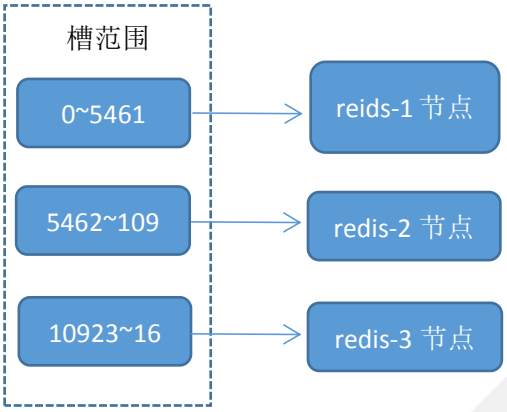
致性哈希分区和虚拟槽分区），其它两种也不做介绍，有兴趣可以百度了解一下。

3，虚拟槽分区(槽：slot)

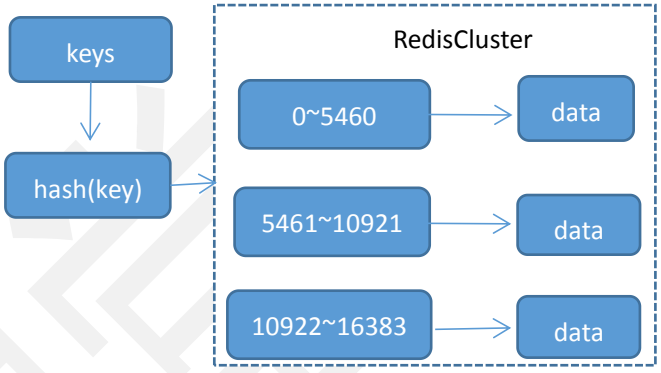
RedisCluster 采用此分区，所有的键根据哈希函数(CRC16[key]&16383)映射到 0－16383 槽内，共 16384 个槽位，每个节点维护部分槽及槽所映射的键值数据

哈希函数: Hash()=CRC16[key]&16383 按位与

槽与节点的关系如下



槽与节点的关系



用 hash 函数将键映射到槽，再由槽指向数据

redis 用虚拟槽分区原因：1，解耦数据与节点关系，节点自身维护槽映射关系，分布式存储

4，redisCluster 的缺陷：

a，键的批量操作支持有限，比如 mset, mget，如果多个键映射在不同的槽，就不支持了

b，键事务支持有限，当多个 key 分布在不同节点时无法使用事务，同一节点是支持事务

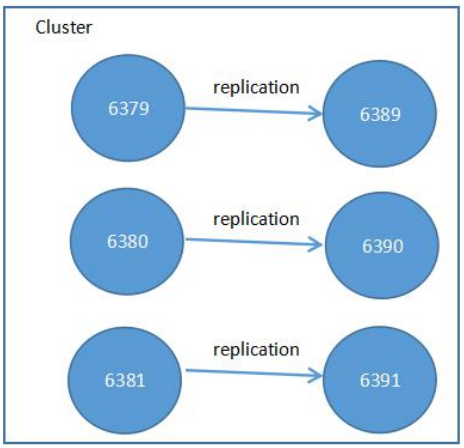
c，键是数据分区的最小粒度，不能将一个很大的键值对映射到不同的节点

d，不支持多数据库，只有 0，select 0

e，复制结构只支持单层结构，不支持树型结构。

8.2 集群环境搭建：

1，在/usr/local/bin/clusterconf 目录，



6389 为 6379 的从节点，6390 为 6380 的从节点，6391 为 6381 的从节点

2，分别修改 6379、 6380、 7381、 6389、 6390、 6391 配置文件

port 6379 //节点端口

cluster-enabled yes //开启集群模式

cluster-node-timeout 15000 //节点超时时间(接收 pong 消息回复的时间)

cluster-config-file /usrlocalbin/cluster/data/nodes-6379.conf 集群内部配置文件

其它节点的配置和这个一致，改端口即可

3，配置完后，**启动 6 个 redis 服务**

4，**自动安装模式：**

在/usr/local 新建目录：ruby

```
[root@myhost ruby]# pwd
/usr/local/ruby
```

下载链接：https://pan.baidu.com/s/1kWsf3Rh 密码：n3pc

从这个链接下载 ruby-2.3.1.tar.gz 和 redis-3.3.0.gem

tar -zxvf ruby-2.3.1.tar.gz

```
[root@myhost rubysoft]# pwd
/usr/local/soft/rubysoft
[root@myhost rubysoft]# ll
总用量 17488
-rw-r--r--.  1 root root    92160 1月  12 22:59 redis-3.3.0.gem
drwxr-xr-x. 26 1044 1044     8192 1月  13 06:47 ruby-2.3.1
-rw-r--r--.  1 root root 17797997 4月  26 2016 ruby-2.3.1.tar.gz
```

a, cd ruby-2.3.1

b, ./configure -prefix=/usr/local/ruby

c, make && make install //过程会有点慢，大概 5－10 分钟

d, 然后 gem install -l redis-3.3.0.gem //没有 gem 需要安装 yum install gem

e, 准备好 6 个节点，（注意不要设置 requirepass），将/usr/local/bin/clusterconf/data 的 config-file 删除；依次启动 6 个节点：./redis-server clusterconf/redis6379.conf

如果之前 redis 有数据存在，flushall 清空；(**坑:不需要 cluster meet ..**)

f, 进入 cd /usr/local/bin, 执行以下：1 代表从节点的个数

./redis-trib.rb create --replicas 1 192.168.0.111:6379 192.168.0.111:6380 192.168.0.111:6381 192.168.0.111:6389 192.168.0.111:6390 192.168.0.111:6391

主从分配，6379 是 6389 的从节点

```
127.0.0.1:6379> cluster nodes
6f1e6e80fe1b07ac2e16a7c969dfef41dd355736 127.0.0.1:6381@16381 master - 0 1515799937925 3 connected 10923-16383
9e16a0ce0676f02a4a272c4f5cb376e5478129ab 127.0.0.1:6389@16389 master - 0 1515799934907 7 connected 0-5460
d47a5f138689951af14375ecb329033a3b0a5d9e 127.0.0.1:6379@16379 myself,slave 9e16a0ce0676f02a4a272c4f5cb376e5478129ab 0
b165e20920c35657bcbe3a6289ff7909980cfa61 127.0.0.1:6390@16390 slave 348011ada5826982ea6b2f43526f87bd17db394e 0 151579
348011ada5826982ea6b2f43526f87bd17db394e 127.0.0.1:6380@16380 master - 0 1515799935000 2 connected 5461-10922
bee0eaccec8eadc200d4225cd8d275958c91aeb2 127.0.0.1:6391@16391 slave 6f1e6e80fe1b07ac2e16a7c969dfef41dd355736 0 151579
127.0.0.1:6379> set ddd 33
```

貌似只有主节点可读写，从节点不可以

主节点死后，从节点变成主节点

e,集群健康检测：

redis-trib.rb check 192.168.42.111:6379　(注：redis 先去注释掉 requirepass，不然连不上)

```
[ERR] Nodes don't agree about configuration!
>>> Check for open slots...
[WARNING] Node 127.0.0.1:6379 has slots in importing state (5798).
[WARNING] Node 127.0.0.1:6380 has slots in importing state (1180,2998,11212,11756).
[WARNING] Node 127.0.0.1:6381 has slots in importing state (1180,2998,5798).
[WARNING] The following slots are open: 5798,1180,2998,11212,11756
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@myhost bin]# redis-trib.rb check 127.0.0.1:6379
```

如此出现了这个问题，6379 的 5798 槽位号被打开了

解决如下：

```
[root@myhost bin]# ./redis-cli -c -p 6379
127.0.0.1:6379> cluster setslot 5798 stable
```

6379，6380，6381 的有部分槽位被打开了，分别进入这几个节点，执行

6380:>cluster setslot 1180 stable

cluster setslot 2998 stable

cluster setslot 11212 stable

其它也一样，分别执行修复完后：

```
[root@myhost bin]# redis-trib.rb check 127.0.0.1:6381
>>> Performing Cluster Check (using node 127.0.0.1:6381)
M: 6f1e6e80fe1b07ac2e16a7c969dfef41dd355736 127.0.0.1:6381
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
S: bee0eaccec8eadc200d4225cd8d275958c91aeb2 127.0.0.1:6391
   slots: (0 slots) slave
   replicates 6f1e6e80fe1b07ac2e16a7c969dfef41dd355736
M: d47a5f138689951af14375ecb329033a3b0a5d9e 127.0.0.1:6379
   slots:0-5460 (5461 slots) master
   1 additional replica(s)
M: 348011ada5826982ea6b2f43526f87bd17db394e 127.0.0.1:6380
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
S: b165e20920c35657bcbe3a6289ff7909980cfa61 127.0.0.1:6390
   slots: (0 slots) slave
   replicates 348011ada5826982ea6b2f43526f87bd17db394e
S: 9e16a0ce0676f02a4a272c4f5cb376e5478129ab 127.0.0.1:6389
   slots: (0 slots) slave
   replicates d47a5f138689951af14375ecb329033a3b0a5d9e
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

此时修复后的健康正常；

当停掉 6379 后，过会 6389 变成主节点

注意：使用客户端工具查询时要加-c

./redis-cli -h 192.168.42.111 -p 6379 **-c**

mset aa bb cc dd,批设置对应在不同的 solt 上，缺点

```
192.168.1.111:6380> mset aa dd bb dd
(error) CROSSSLOT Keys in request don't hash to the same slot
```

14,集群正常启动后，在每个 redis.conf 里加上

masterauth “12345678”

requiredpass “12345678”

当主节点下线时，从节点会变成主节点，用户和密码是很有必要的，设置成一致

15,这上面是一主一从，那能不能一主多从呢？

```
6379(Master) → 6479(Slave of 6379) → 6579(Slave of 6379)
6380(Master) → 6480(Slave of 6380) → 6580(Slave of 6380)
6381(Master) → 6481(Slave of 6381) → 6581(Slave of 6381)
```

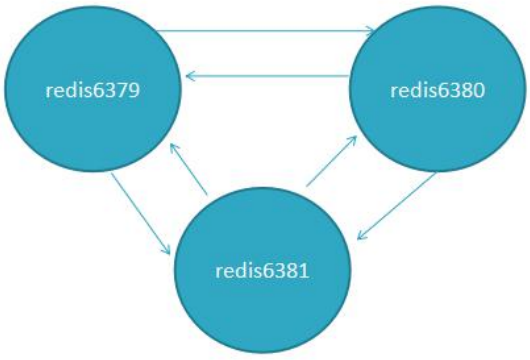
./redis-trib.rb create --replicas 2
192.168.42.111:6379 192.168.42.111:6380 192.168.42.111:6381

192.168.42.111:6479 192.168.42.111:6480 192.168.42.111:6481

192.168.42.111:6579 192.168.42.111:6580 192.168.42.111:6581

8.3 节点之间的通信

1，节点之间采用 Gossip 协议进行通信，Gossip 协议就是指节点彼此之间不断通信交换信息



当主从角色变化或新增节点，彼此通过 ping/pong 进行通信知道全部节点的最新状态并达到集群同步

2,Gossip 协议

Gossip 协议的主要职责就是信息交换，信息交换的载体就是节点之间彼此发送的 Gossip 消息，常用的 Gossip 消息有 ping 消息、pong 消息、meet 消息、fail 消息

meet 消息：用于通知新节点加入，消息发送者通知接收者加入到当前集群，meet 消息通信完后，接收节点会加入到集群中，并进行周期性 ping pong 交换

ping 消息：集群内交换最频繁的消息，集群内每个节点每秒向其它节点发 ping 消息，用于检测节点是在在线和状态信息，ping 消息发送封装自身节点和其他节点的状态数据；

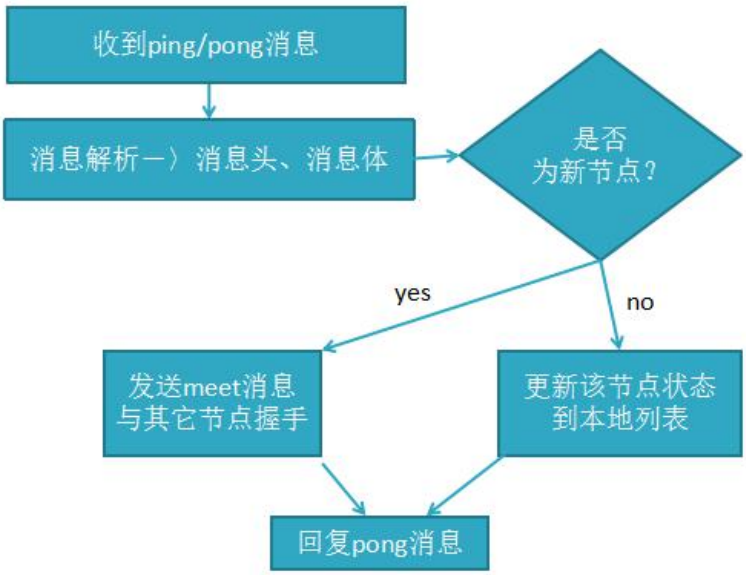
pong 消息，当接收到 ping meet 消息时，作为响应消息返回给发送方，用来确认正常通信，pong 消息也封闭了自身状态数据；

fail 消息：当节点判定集群内的另一节点下线时，会向集群内广播一个 fail 消息，后面会讲到。……

3,消息解析流程

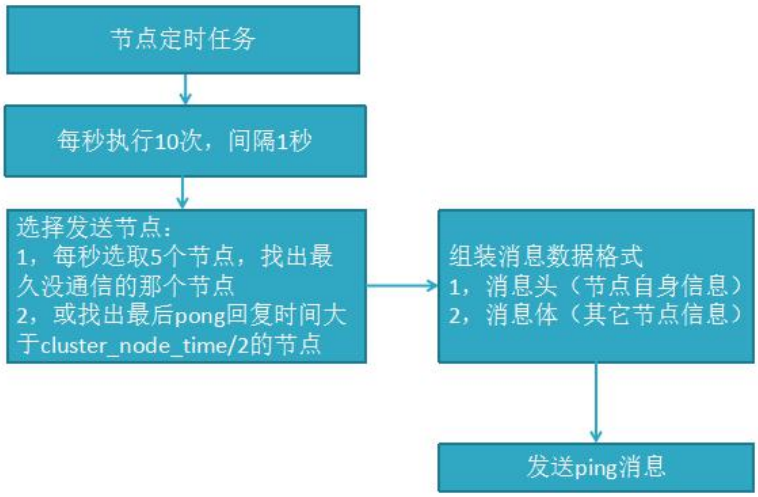
所有消息格式为：消息头、消息体，消息头包含发送节点自身状态数据（比如节点 ID、槽映射、节点角色、是否下线等），接收节点根据消息头可以获取到发送节点的相关数据。

消息解析流程：



4,选择节点并发送 ping 消息：

Gossip 协议信息的交换机制具有天然的分布式特性，但 ping pong 发送的频率很高，可以实时得到其它节点的状态数据，但频率高会加重带宽和计算能力，因此每次都会有目的性地选择一些节点； 但是节点选择过少又会影响故障判断的速度，redis 集群的 Gossip 协议兼顾了这两者的优缺点，看下图：



不难看出：节点选择的流程可以看出消息交换成本主要体现在发送消息的节点数量和每个消息携带的数据量

流程说明：

A，选择发送消息的节点数量：集群内每个节点维护定时任务默认为每秒执行 10 次，每秒会随机选取 5 个节点，找出最久没有通信的节点发送 ping 消息，用来保证信息交换的随机性，每 100 毫秒都会扫描本地节点列表，如果发现节点最近一次接受 pong 消息的时间大于 cluster-node-timeout/2 则立刻发送 ping 消息，这样做目的是防止该节点信息太长时间没更新，当我们宽带资源紧张时，在可 redis.conf 将 cluster-node-timeout 15000 改成 30 秒，但不能过度加大

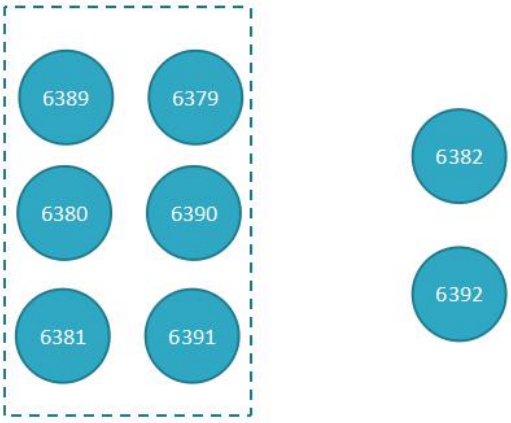
B，消息数据：节点自身信息和其他节点信息

5，集群扩容

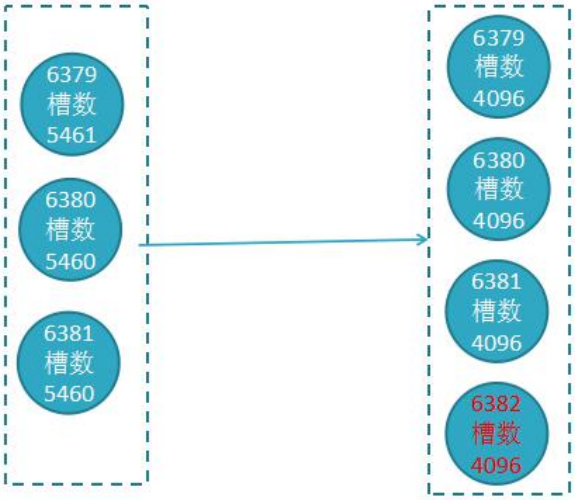
这也是分布式存储最常见的需求，当我们存储不够用时，要考虑扩容

扩容步骤如下：

A，准备好新节点



B，加入集群,迁移槽和数据



1),同目录下新增 redis6382.conf、redis6392.conf 两

启动两个新 redis 节点

./redis-server clusterconf/redis6382.conf &　（新主节点）

./redis-server clusterconf/redis6392.conf &　（新从节点）

2),新增主节点

./redis-trib.rb add-node 192.168.42.111:6382 192.168.42.111:6379

6379 是原存在的主节点，6382 是新的主节点

3),添加从节点

redis-trib.rb add-node --slave --master-id 03ccad2ba5dd1e062464bc7590400441fafb63f2 192.168.42.111:6392 192.168.42.111:6379

--slave，表示添加的是从节点

--master-id 03ccad2ba5dd1e062464bc7590400441fafb63f2 表示主节点 6382 的 master_id

192.168.42.111:6392,新从节点

192.168.42.111:6379 集群原存在的旧节点

4)，redis-trib.rb reshard 192.168.42.111:6382　//为新主节点重新分配 solt

How many slots do you want to move (from 1 to 16384)? 1000 //设置 slot 数 1000

What is the receiving node ID? 464bc7590400441fafb63f2 //新节点 node id

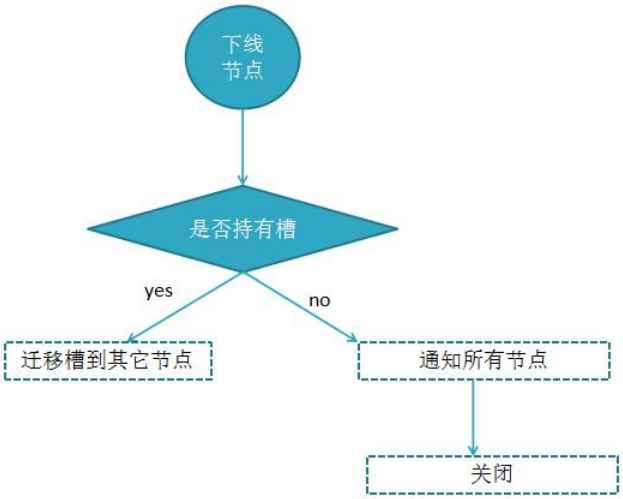
Source node #1:all //表示全部节点重新洗牌

新增完毕！

3，集群减缩节点：

集群同时也支持节点下线掉

下线的流程如下：



流程说明：

A，确定下线节点是否存在槽 slot,如果有，需要先把槽迁移到其他节点，保证整个集群槽节点映射的完整性；

B，当下线的节点没有槽或本身是从节点时，就可以通知集群内其它节点（或者叫忘记节点），当下线节点被忘记后正常关闭。

删除节点也分两种：

一种是主节点 6382，一种是从节点 6392。

在从节点 6392 中，没有分配哈希槽，执行

./redis-trib.rb del-node 192.168.42.111:6392 7668541151b4c37d2d9 有两个参数 ip：port 和节点的 id。 从节点 6392 从集群中删除了。

主节点 6382 删除步骤：

1，./redis-trib.rb reshard 192.168.42.111:6382

问我们有多少个哈希槽要移走，因为我们这个节点上刚分配了 1000 个所以我们这里输入 1000

2，最后

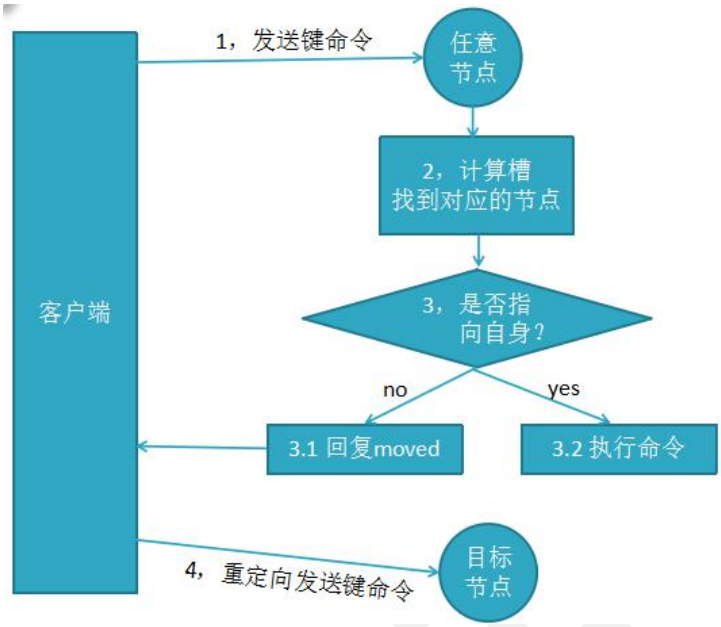
./redis-trib.rb del-node 192.168.42.111:6382 3e50c6398c75e0088a41f908071c2c2eda1dc900

此时节点下线完成……

请求路由重定向

我们知道，在 redis 集群模式下，redis 接收的任何键相关命令首先是计算这个键 CRC

值，通过 CRC 找到对应的槽位，再根据槽找到所对应的 redis 节点，如果该节点是本身，则直接处理键命令；如果不是，则回复键重定向到其它节点，这个过程叫做 MOVED 重定向

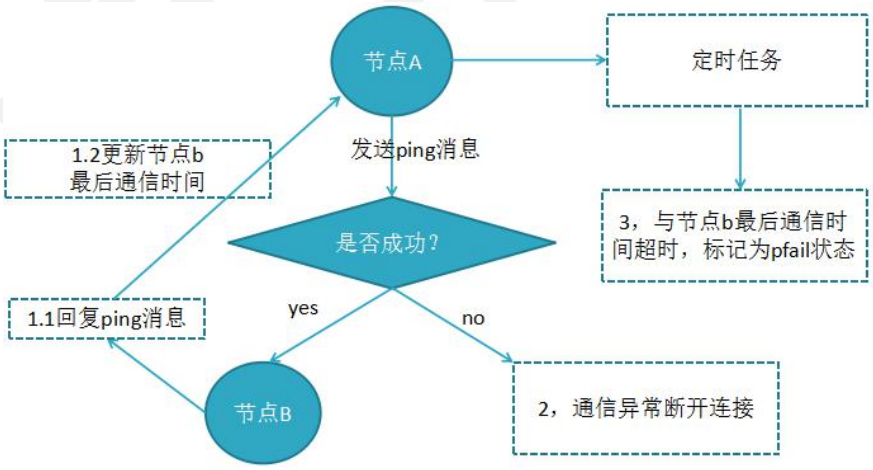


故障转移：

redis 集群实现了高可用，当集群内少量节点出现故障时，通过故障转移可以保证集群正常对外提供服务。

当集群里某个节点出现了问题，redis 集群内的节点通过 ping pong 消息发现节点是否健康，是否有故障，其实主要环节也包括了 主观下线和客观下线；

主观下线：指某个节点认为另一个节点不可用，即下线状态，当然这个状态不是最终的故障判定，只能代表这个节点自身的意见，也有可能存在误判；



下线流程：

A，节点 a 发送 ping 消息给节点 b ,如果通信正常将接收到 pong 消息，节点 a 更新最近一次与节点 b 的通信时间；

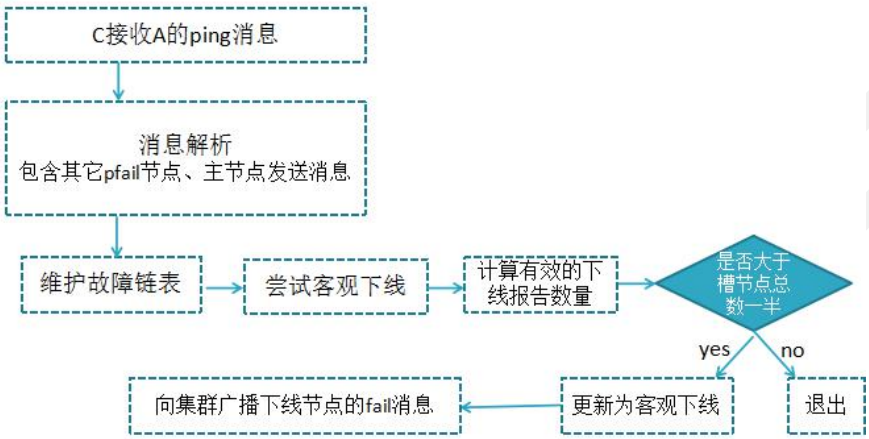
B，如果节点 a 与节点 b 通信出现问题则断开连接，下次会进行重连，如果一直通信失败，则它们的最后通信时间将无法更新；

C，节点 a 内的定时任务检测到与节点 b 最后通信时间超过 cluster_note-timeout 时，更新本地对节点 b 的状态为主观下线（pfail）

客观下线：指真正的下线，集群内多个节点都认为该节点不可用，达成共识，将它下线，如果下线的节点为主节点，还要对它进行故障转移

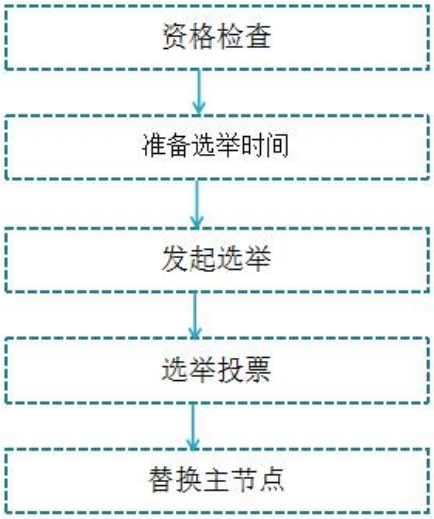
假如节点 a 标记节点 b 为主观下线，一段时间后节点 a 通过消息把节点 b 的状态发到其它节点，当节点 c 接受到消息并解析出消息体时，会发现节点 b 的 pfail 状态时，会触发客观下线流程；

当下线为主节点时，此时 redis 集群为统计持有槽的主节点投票数是否达到一半，当下线报告统计数大于一半时，被标记为客观下线状态。



故障恢复：

故障主节点下线后，如果下线节点的是主节点，则需要在它的从节点中选一个替换它，保证集群的高可用；转移过程如下：



1，资格检查：检查该从节点是否有资格替换故障主节点，如果此从节点与主节点

断开过通信，那么当前从节点不具体故障转移；

2，准备选举时间：当从节点符合故障转移资格后，更新触发故障选举时间，只有到达该时间后才能执行后续流程；

3，发起选举：当到达故障选举时间时，进行选举；

4，选举投票：只有持有槽的主节点才有票，会处理故障选举消息，投票过程其实是一个领导者选举（选举**从节点为领导者**）的过程，每个主节点只能投一张票给从节点，当从节点收集到足够的选票（大于 N/2+1）后，触发替换主节点操作，撤销原故障主节点的槽，委派给自己，并广播自己的委派消息，通知集群内所有节点。