1. 前言

讨论Redis同城双机房条件下的部署架构

1. 具体实现方案

### 方案A-基于哨兵模式高可用主从方案（单写多读）

**哨兵作用：**

* + 集群监控：持续监测集群中的master和slave是否可用。
  + 消息通知：通知系统管理员，集群中的部分服务不可用。
  + 故障切换：若master挂掉，哨兵集群会启动故障切换，选出其中一个slave作为新的master。使用redis的应用程序也会被告知使用新的master。
  + 配置中心：哨兵集群为redis客户提供服务发现功能，客户端访问哨兵集群，会被告知当前的master地址。

**哨兵部署：**

* + 建议至少三个节点，一个sentinel要接管failover过程必须得到至少半数以上的sentinel认可，三个sentinel节点挂掉一个集群仍然正常工作；
  + 为了确保机房故障（断电或者网络分区）下仍然能够进行主从切换，则主节点所在的机房sentinel个数应该小于集群的一半，这样当主节点所在的机房故障时，另一个机房中存活有超过半数的sentinel，从而接管failover过程（在另一个机房的slave节点中提升一个当做主节点）。

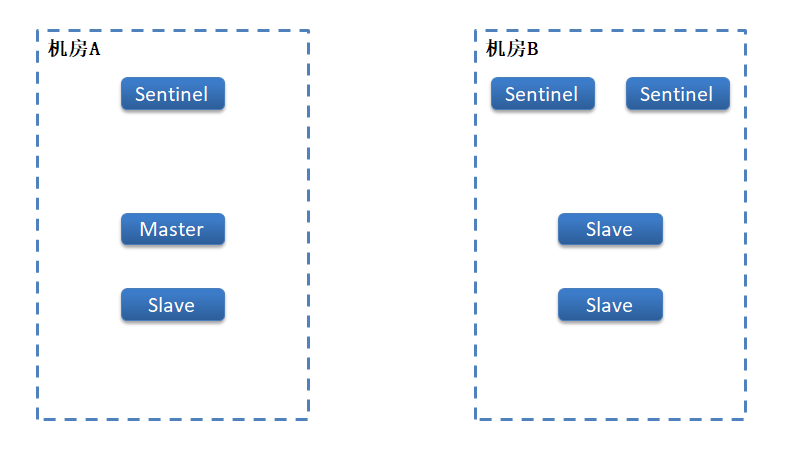


图1 哨兵模式

**数据同步**：借助redis自身主从节点之间的数据同步。

1. 当主节点挂后，哨兵察觉到Master挂掉后，会启动故障自动切换，选出一个slave作为新的主节点，这里尽量优先选择同机房的slave做主从切换（脚本动态切换slave节点的优先级配置，另一种就是在redis节点相互请求信息中带上DC-id作为机房标识），仅当机房故障时才进行跨机房故障转移；

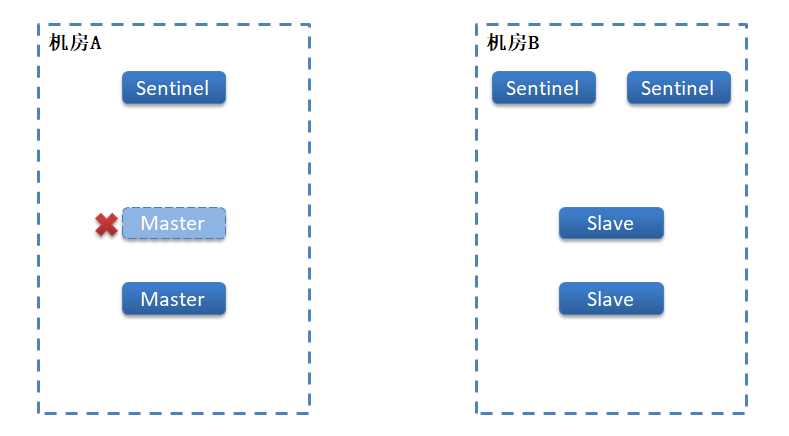


图2 主节点切换

1. 当故障的主节点再次启动加入主从模式中，它将作为新主节点的从节点，同步当前主节点的数据。

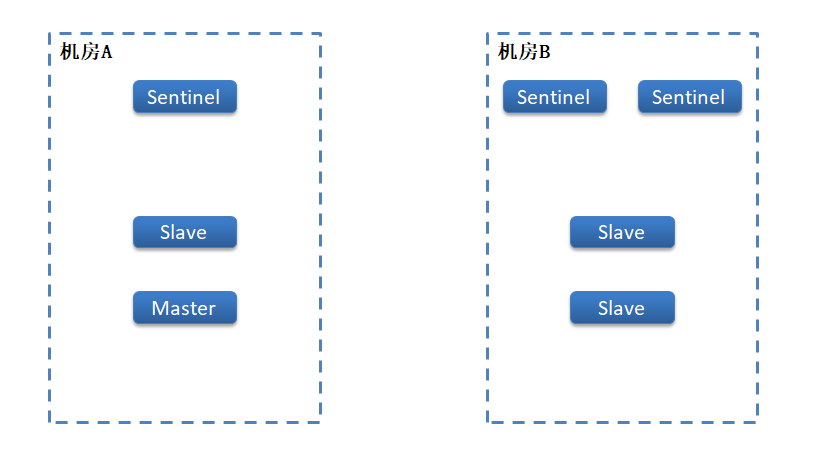


图3 主变从

1. 接下来讨论机房故障的情况。

* 机房B故障：A的sentinel发现其他Sentinel和B的Slave节点都失效，此时虽然哨兵的failover功能无法使用，但是A的Master和Slave节点仍然能够正常读写工作，此时只要等待B机房恢复即可，恢复重启后sentinel重新监测A机房master节点，slave节点会从master节点同步数据（redis自身确保，可能是全量同步也可能是增量同步）；
* 机房A故障：B的哨兵发起failover过程，将其中的一个slave节点提升为master节点，此时B机房可以正常提供读写操作。

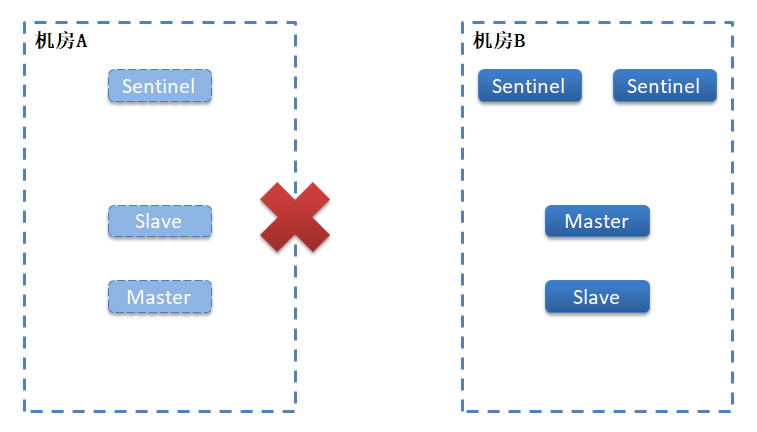


图4机房A故障，B接管

* 此时机房A恢复，sentinel更新自己和redis节点的配置将B的master节点确认为新的主节点（这个是由sentinel自身机制确保，B中的sentinel配置版本号较新，A会以B为准），A机房的redis节点变为slave并从B的master节点进行数据同步；

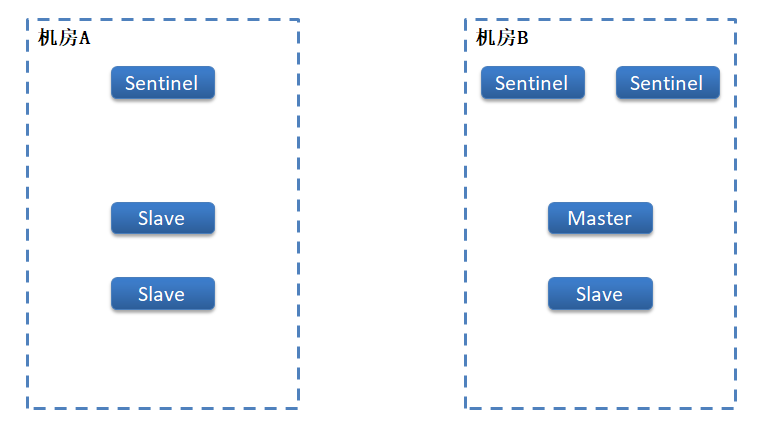


图5机房A恢复后，最新的主从结构

* 到这里redis主从切换都没有问题，单节点的故障也不会影响整个主从结构。
* 现在的问题是Master节点和超过半数的Sentinel节点都在B机房，这个时候如果机房B故障（宕机或者网络分区），双机房都无法正常使用（无master节点可用也无法进行failover）。需要人工介入或者依赖第三方监控服务去调整当前节点部署结构。将机房B的sentinel减少到半数以下，使机房A中有超过半数的sentinel节点。调整后如图六所示，参照图一，相当于机房A和机房B进行了一次对换。

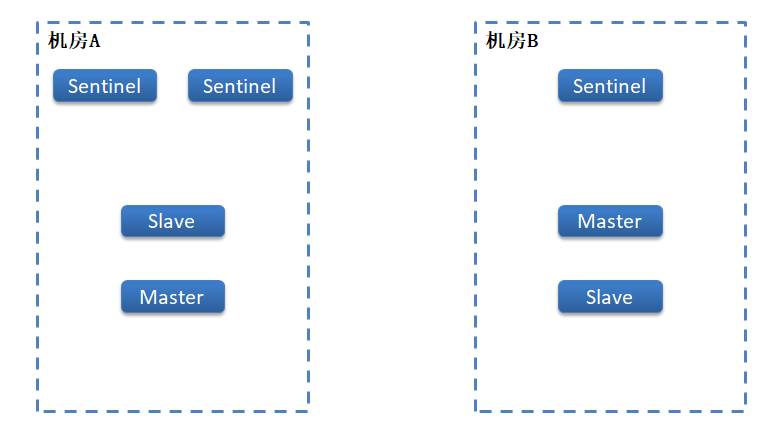


图6调整后的部署结构图

### 方案B-基于代理+哨兵的高可用分片方案（多写多读）

**Twemproxy代理：**

* 支持redis和memcache协议，提供多种hash算法和分布式算法、权重配置等；
* 后端Redis节点宕机，twemproxy会自动摘除，恢复后，能自动识别并重新加入redis分片集群中继续使用；
* 节点数量发生变化时需要进行数据重新分布，需要人工介入、脚本变更配置，重启后生效；
* 单台twemproxy性能有一定的损耗，约10%-20%左右，多twemproxy和多redis会有一定的提升；

**Sentinel哨兵：**

* 这里不同与方案A的是各机房独立部署sentinel group，仅负责机房内部的主从切换，当然如果有第三方仲裁机房，sentinel group可以只部署一个，同时监听多个master-slave主从结构。

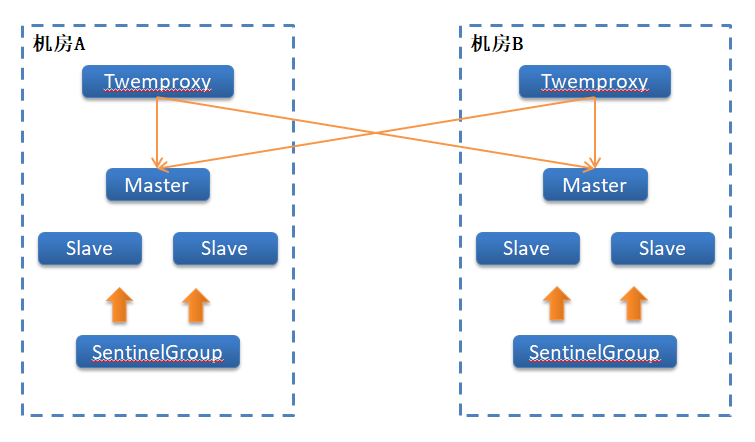


图7 代理+哨兵水平分片结构

**Twemproxy配置**

redis\_15454:

listen: 192.168.80.129:15454 #代理监听的IP端口

hash: fnv1a\_64 #键hash算法

distribution: ketama #键分布算法，决定键落到哪个服务器

auto\_eject\_hosts: true #

redis: true #代理redis命令请求，默认是memcache

# server\_retry\_timeout: -1 #连接重试超时时间

server\_failure\_limit: 1 #连接失败重试次数

servers:

- 192.168.80.128:6579:1 #redis节点

- 192.168.80.130:6581:1 #redis节点

redis\_auth: 123456 #redis密码

**第一种情况：机房内部主节点宕掉**

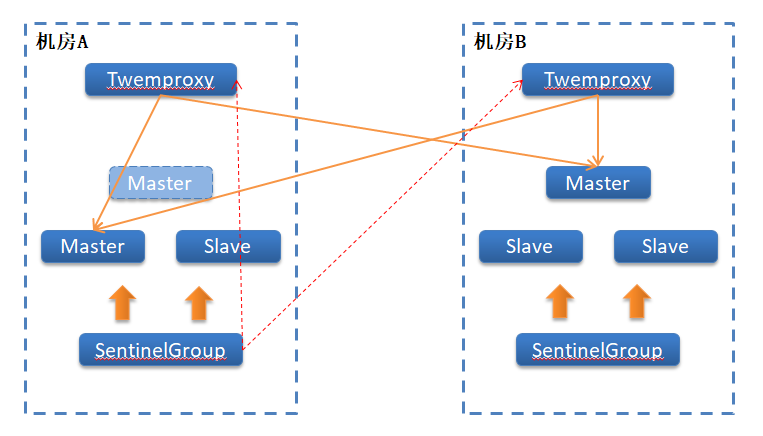


图8 主节点故障转移

* Master节点宕掉，SentinelGroup负责redis的故障转移，并且在sentinel的配置中支持事件通知脚本，当发现主从切换时执行相应的脚本操作。
* sentinel client-reconfig-script mymaster /var/redis/reconfig.sh
* 脚本中去更新twemproxy的server节点指向新的master节点，并且重启twemproxy使配置生效。
* 这时数据基本不会丢失，分片集群恢复正常。

**这里可能会出现的问题：**

主从切换期间客户端访问数据会不会触发twemproxy的自动重分片规则，如果在切换期间触发了代理的重分片，数据可能都会落到机房B，并且在故障转移完成reload代理后又会回到机房A，那么这段时间写入的数据就会丢失，并且两边数据不一致。

**第二种情况：机房A挂掉**

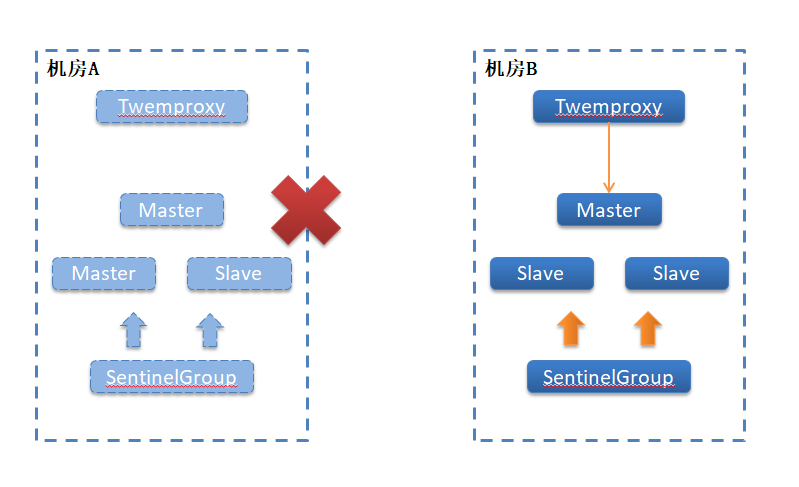


图9机房故障

此时上层负载的所有请求都会落到B机房的twemproxy节点，自动剔除失效节点，所有读写请求都到B机房，当A机房恢复时，又由twemproxy自动恢复分片。

**这里的问题也是一样：**

机房的切换故障恢复期间如果有更新缓存操作就会导致两边缓存数据不一致，如果有写入或者删除操作则会丢失操作。如果A机房仅做数据库缓存，那么恢复时可以直接清空数据重新加载缓存，避免数据不一致性。（注意：另外B机房在这段期间写入的数据该如何清理，否则如果A机房再挂掉，B中的数据也是有问题的）

1. 两种方案总结

### 方案A

总结：采用redis官方HA方案，基于sentinel部署的哨兵主从双机房结构，redis节点的故障转移由sentinel负责，能够支持单节点或者机房故障时的failover场景。但是要确保大多数sentinel节点在故障时存活，这里需要人工介入或者自动监控服务去进行sentinel节点动态调整。

优点：最简单的主从实现方案，单写使双机房数据一致性较高（注意，这里并不能保证强一致性，因为redis主从间的数据同步操作是异步的，主从复制期间发生故障可能会导致主节点部分数据丢失）。

缺点：单写节点有瓶颈，后期无法扩容；依赖redis同步机制，如果仅当做数据库缓存使用问题不大。如果要求强一致性的话需要客户端使用wait命令来确保写操作落地到slave节点，但是该操作会阻塞master节点；机房间的故障恢复需要人工介入或者第三方服务监控调整。

### 方案B

总结：twemproxy+sentinel高可用分片方案提供了强大的扩展能力，并且结合脚本和通知事件可以实现自动的故障切换过程。

优点：扩展性强，自动切换方便；

缺点：数据一致性问题分片切换的过程很难保证数据的一致性，在数据库状态变更操作过多的场景下需要提供数据一致性的解决方案，要么就不能开启twemproxy的自动故障切换功能，拒绝不属于当前分片的写操作，需要人工介入进行分片节点的恢复。最好能够支持配置在线 reload以及Redis Master-Slave 读写分离，需要后续开发。