[1] prepare masters and slaves

**sentinel**

**sentinelRedisInstance**

**dict**

**attribute**

[2] config and start three sentinels. Sentinel1, sentinel2, sentinel3

[3] all master sentinelRedisInstance sync connect to master in timer logic

[4] send info command to masters and slaves in sentinelSendPeriodicCommands(timer logic) for getting redis info. The callback sentinelRefreshInstanceInfo will set runid and other infos, also will create new slave sentinelRedisInstance if needed.

[5] all slave sentinelRedisInstance sync connect to master in timer logic

[6] send info command to masters and slaves in sentinelSendPeriodicCommands(timer logic) for getting redis info. The callback sentinelRefreshInstanceInfo will set runid and other infos, also will create new slave sentinelRedisInstance if needed.

[7] sentinel SUBSCRIBE hello channel through pubsub connection from master or slave, and will publish its info in timer logic function sentinelSendHello. They will find each other and make connections in sentinelProcessHelloMessage callback.

[8] sentinels send ping command by function sentinelSendPing in timer logic to check if remote sentinel/master/slave is alive. In sentinelPingReplyCallback, sentinel update some timer attributes for checking remote redis if down in other timer logic.

**ASSUME SUCH SITUATION**:

Sentinel1,2,3 monitors masterA. Then s1, s2 lose connection from masterA while s3 is normal. S1,s2,s3 can communicate with each other.

Sentinels check master DOWN and failover with following steps.

[9] sentinel1 checks masterA S\_DOWN in timer logic sentinelCheckSubjectivelyDown.

Sentinel1,2,3 will check every **sentinelRedisInstance** if it is S\_DOWN.

S1 check masterA S\_DOWN while s2,s3 not.

In s1’s timer logic, it will continuing ask other sentinels about masterA’s state to check masterA O\_DOWN.

Just now, s1 make flags masterA S\_DOWN while s2, s3 was not.

[10] sentinel2 checks masterA S\_DOWN. Both s1 and s2 will ask other sentinels about masterA’s state by calling function sentinelAskMasterStateToOtherSentinels. It use normal connection sending “is-master-down-by-addr“ message to each other and process response in function sentinelReceiveIsMasterDownReply. Then both s1 and s2 flags masterA S\_DOWN and respond state to each other.

[11] Both s1 and s2 will check masterA O\_DOWN state in timer logic function sentinelCheckObjectivelyDown. masterA’s quorum is configured in config file. masterA can be flaged O\_DOWN only when number of S\_DOWN checked by sentinels is greater than masterA’s quorum.

[12] Both s1 s2 check starting failover in sentinelStartFailoverIfNeeded and do starting failover in sentinelStartFailover which will only be called once in one failover stage.

[13] failover state machine: select leader sentinel by function sentinelAskMasterStateToOtherSentinels(masterA, SENTINEL\_ASK\_FORCED). This function will send SENTINEL is-master-down-by-addr message to other sentinels for electing leader sentinel. Each sentinel will receive this message and vote leader by sentinelVoteLeader.

For example:

Sentinel1 send message with params [epoch=2 myid=1] to other sentinels

Sentinel2 send message with params [epoch=2 myid=2] to other sentinels

S1 recv message from s2 and set masterA.leader=s2 and response leader=S2 to s2

S2 recv message from s1 and set masterA.leader=s1 and response leader=S1 to s1

S3 recv message from s1 and set masterA.leader=s1 and response leader=S1 to s1

S3 recv message from s2 and set masterA.leader=s1 and response leader=S1 to s2

S1 receive response from s2 and set sentinel2.leader=s1

S1 receive response from s3 and set sentinel3.leader=s1

S2 receive response from s1 and set sentinel1.leader=s2

S2 receive response from s3 and set sentinel3.leader=s1

[14] In failover state logic sentinelFailoverStateMachine, process elect leader. First in state logic: sentinelFailoverWaitStart. S1 and S2 run sentinelGetLeader to get leader. At last, S1 select s1 as leader and do the next state logic, while s2 and s3 will not. Then S1 will change masterA.failover\_state to SENTINEL\_FAILOVER\_STATE\_SELECT\_SLAVE.

[15] S1 will select a slave to be a master in the future by function sentinelFailoverSelectSlave.

[16] After s1 selects slave1 as new master, it changes slave1’s role by sending ‘slave of none’ in state logic function sentinelFailoverSendSlaveOfNoOne and go to next state: SENTINEL\_FAILOVER\_STATE\_WAIT\_PROMOTION.

[17] In wait\_promotion state, every sentinel changes its data by sending ‘info’ to redis instances.

In ‘info’ logic:

S1, s2, s3 will change slave1’s role\_repported to MASTER while it used to be SLAVE.

[17-1] In s1:

It will find slave1 changing to be master and force hello message by hello channel. Then go to next state SENTINEL\_FAILOVER\_STATE\_RECONF\_SLAVES.

[17-2] In s1:

In this state, s1 call function sentinelFailoverReconfNextSlave to send SLAVE OF <new master address> to all the remaining slaves and add SRI\_RECONF\_SENT flag to slave.flags.

In ‘info’ response, sentinel will check and find slaves reconfig to new master done. Thus, sentinel check failover end by function sentinelFailoverDetectEnd.

At last, sentinel call function sentinelResetMasterAndChangeAddress to change master and slaves infos.

[17-3] In s2 and s3:

While receving hello message from hello channel, they find master’s addr was changed, and use function sentinelResetMasterAndChangeAddress to change master/slave roles.

[others]

<1> sentinel 首次启动通过给master发送info命令获得其他slave的地址，连接后会将此数据写入到自己本地的配置文件 known-slave 属性. 下次重新启动时会从配置文件加载监控的redis的地址。

<2> send ping 和 publish hello 会往所有的instance上发(master, slave, sentinel)

<3> 当master down后，会有多个sentinel检测到，产生S\_DOWN状态，此时会询问其他sentinel关于master的状态。当得到多数回复时S\_DOWN后，会进入O\_DOWN，此时这些sentinel会发出投票消息。每个sentinel在发出投票信息和收到投票信息时会记录 failover\_start\_time，保证只有一个sentinel会成为leader。（多个sentinel同时发起投票后，可能会产生不了leader，此时会无法产生leader，等待超时后中断failover，从O\_DOWN重新开始）

<4> sentinel发送hello消息时，会有两种情况 ，如果是发给master/slave，则在此channel上订阅的sentinel会收到消息。如果直接发给了sentinel，相当于一个client给sentinel发送了命令，sentinel会直接处理，最终调用统一函数处理hello消息内容。

<5> sentinel leader的恢复过程和其他sentinel不一样。Leader会不停发送INFO命令检查各个master/slave是否都已经完成更新了，才会退出 SENTINEL\_FAILOVER\_STATE\_RECONF\_SLAVES 此状态。最后在进入到switch-master状态。

最后所有的sentinel都会重连master/slave。

<6> 原master重新启动后，因为其自身的配置状态还是master，但是在sentinel那边设置的instance中，其对应的角色是一个slave，因此在发现role不匹配时，将此redis重新使用slaveof 命令连接到新的master上，改变其角色。