**六：定时发送消息**

哨兵每隔一段时间，会向其所监控的所有实例发送一些命令，用于获取这些实例的状态。这些命令包括：”PING”、”INFO”和”PUBLISH”。

“PING”命令，主要用于哨兵探测实例是否活着。如果对方超过一段时间，还没有回复”PING”命令，则认为其是主观下线了。

“INFO”命令，主要用于哨兵获取实例当前的状态和信息，比如该实例当前是主节点还是从节点；该实例反馈的IP地址和PORT信息，是否与我记录的一样；该实例如果是主节点的话，那它都有哪些从节点；该实例如果是从节点的话，它与主节点是否连通，它的优先级是多少，它的复制偏移量是多少等等，这些信息在故障转移流程中，是判断实例状态的重要信息；

“PUBLISH”命令，主要用于哨兵向实例的HELLO频道发布有关自己以及主节点的信息，也就是所谓的HELLO消息。因为所有哨兵都会订阅主节点和从节点的HELLO频道，因此，每个哨兵都会收到其他哨兵发布的信息。

因此，通过这些命令，尽管在配置文件中只配置了主节点的信息，但是哨兵可以通过主节点的”INFO”回复，得到所有从节点的信息；又可以通过订阅实例的HELLO频道，接收其他哨兵通过”PUBLISH”命令发布的信息，从而得到监控同一主节点的所有其他哨兵的信息。

在“主函数”sentinelHandleRedisInstance中，是通过调用sentinelSendPeriodicCommands来发送这些命令的。注意，以上的命令都有自己的发送周期，在sentinelSendPeriodicCommands函数中，并不是一并发送三个命令，而是发送那些，按照发送周期应该发送的命令。该函数的代码如下：

void sentinelSendPeriodicCommands**(**sentinelRedisInstance **\***ri**)** **{**

mstime\_t now **=** mstime**();**

mstime\_t info\_period**,** ping\_period**;**

int retval**;**

*/\* Return ASAP if we have already a PING or INFO already pending, or*

*\* in the case the instance is not properly connected. \*/*

**if** **(**ri**->**flags **&** SRI\_DISCONNECTED**)** **return;**

*/\* For INFO, PING, PUBLISH that are not critical commands to send we*

*\* also have a limit of SENTINEL\_MAX\_PENDING\_COMMANDS. We don't*

*\* want to use a lot of memory just because a link is not working*

*\* properly (note that anyway there is a redundant protection about this,*

*\* that is, the link will be disconnected and reconnected if a long*

*\* timeout condition is detected. \*/*

**if** **(**ri**->**pending\_commands **>=** SENTINEL\_MAX\_PENDING\_COMMANDS**)** **return;**

*/\* If this is a slave of a master in O\_DOWN condition we start sending*

*\* it INFO every second, instead of the usual SENTINEL\_INFO\_PERIOD*

*\* period. In this state we want to closely monitor slaves in case they*

*\* are turned into masters by another Sentinel, or by the sysadmin. \*/*

**if** **((**ri**->**flags **&** SRI\_SLAVE**)** **&&**

**(**ri**->**master**->**flags **&** **(**SRI\_O\_DOWN**|**SRI\_FAILOVER\_IN\_PROGRESS**)))** **{**

info\_period **=** 1000**;**

**}** **else** **{**

info\_period **=** SENTINEL\_INFO\_PERIOD**;**

**}**

*/\* We ping instances every time the last received pong is older than*

*\* the configured 'down-after-milliseconds' time, but every second*

*\* anyway if 'down-after-milliseconds' is greater than 1 second. \*/*

ping\_period **=** ri**->**down\_after\_period**;**

**if** **(**ping\_period **>** SENTINEL\_PING\_PERIOD**)** ping\_period **=** SENTINEL\_PING\_PERIOD**;**

**if** **((**ri**->**flags **&** SRI\_SENTINEL**)** **==** 0 **&&**

**(**ri**->**info\_refresh **==** 0 **||**

**(**now **-** ri**->**info\_refresh**)** **>** info\_period**))**

**{**

*/\* Send INFO to masters and slaves, not sentinels. \*/*

retval **=** redisAsyncCommand**(**ri**->**cc**,**

sentinelInfoReplyCallback**,** **NULL,** "INFO"**);**

**if** **(**retval **==** REDIS\_OK**)** ri**->**pending\_commands**++;**

**}** **else** **if** **((**now **-** ri**->**last\_pong\_time**)** **>** ping\_period**)** **{**

*/\* Send PING to all the three kinds of instances. \*/*

sentinelSendPing**(**ri**);**

**}** **else** **if** **((**now **-** ri**->**last\_pub\_time**)** **>** SENTINEL\_PUBLISH\_PERIOD**)** **{**

*/\* PUBLISH hello messages to all the three kinds of instances. \*/*

sentinelSendHello**(**ri**);**

**}**

**}**

如果实例标志位中设置了SRI\_DISCONNECTED标记，说明当前实例的异步上下文还没有创建好，因此直接返回；

实例的pending\_commands属性，表示已经向该实例发送的命令中，尚有pending\_commands个命令还没有收到回复。每次调用redisAsyncCommand函数，向实例异步发送一条命令之后，就会增加该属性的值，而每当收到命令回复之后，就会减少该属性的值；

因此，如果该属性的值大于SENTINEL\_MAX\_PENDING\_COMMANDS(100)，说明该实例尚有超过100条命令的回复信息没有收到。这种情况下，说明与实例的连接已经不正常了，为了节约内存，因此直接返回；

接下来计算info\_period和ping\_period，这俩值表示发送"INFO"和"PING"命令的时间周期。如果当前时间距离上次收到"INFO"或"PING"回复的时间已经超过了info\_period或ping\_period，则向实例发送"INFO"或"PING"命令；

如果当前实例为从节点，并且该从节点对应的主节点已经客观下线了，则置info\_period为1000，否则的话置为SENTINEL\_INFO\_PERIOD(10000)。之所以在主节点客观下线后更频繁的向从节点发送"INFO"命令，是因为从节点可能会被置为新的主节点，因此需要更加实时的获取其状态；

将ping\_period置为ri->down\_after\_period的值，该属性的值是根据配置文件中down-after-milliseconds选项得到的，如果该属性值大于SENTINEL\_PING\_PERIOD(1000)，则将ping\_period置为SENTINEL\_PING\_PERIOD；

接下来开始发送命令：如果当前实例不是哨兵实例，并且距离上次收到"INFO"命令回复已经超过了info\_period，则向该实例异步发送"INFO"命令。

否则，如果距离上次收到"PING"命令回复已经超过了ping\_period，则调用函数sentinelSendPing向该实例异步发送"PING"命令；

否则，如果距离上次收到"PUBLISH"命令的回复已经超过了SENTINEL\_PUBLISH\_PERIOD(2000)，则调用函数sentinelSendHello向该实例异步发送"PUBLISH"命令；

因此，"PING"用于探测实例是否活着，可以发送给所有类型的实例；而"INFO"命令用于获取实例的信息，只需发送给主节点和从节点实例；而"PUBLISH"用于向HELLO频道发布哨兵本身和主节点的信息，除了发送给主节点和从节点之外，哨兵本身也实现了"PUBLISH"命令的处理函数，因此该命令也会发送给哨兵实例。

1：PING消息

函数sentinelSendPing用于向实例发送”PING”命令，因为该命令用于探测实例是否主观下线，因此等到后面讲解主观下线是在分析。

2：HELLO消息

函数sentinelSendHello用于发布HELLO消息，它的代码如下：

int sentinelSendHello**(**sentinelRedisInstance **\***ri**)** **{**

char ip**[**REDIS\_IP\_STR\_LEN**];**

char payload**[**REDIS\_IP\_STR\_LEN**+**1024**];**

int retval**;**

char **\***announce\_ip**;**

int announce\_port**;**

sentinelRedisInstance **\***master **=** **(**ri**->**flags **&** SRI\_MASTER**)** **?** ri **:** ri**->**master**;**

sentinelAddr **\***master\_addr **=** sentinelGetCurrentMasterAddress**(**master**);**

**if** **(**ri**->**flags **&** SRI\_DISCONNECTED**)** **return** REDIS\_ERR**;**

*/\* Use the specified announce address if specified, otherwise try to*

*\* obtain our own IP address. \*/*

**if** **(**sentinel**.**announce\_ip**)** **{**

announce\_ip **=** sentinel**.**announce\_ip**;**

**}** **else** **{**

**if** **(**anetSockName**(**ri**->**cc**->**c**.**fd**,**ip**,sizeof(**ip**),NULL)** **==** **-**1**)**

**return** REDIS\_ERR**;**

announce\_ip **=** ip**;**

**}**

announce\_port **=** sentinel**.**announce\_port **?**

sentinel**.**announce\_port **:** server**.**port**;**

*/\* Format and send the Hello message. \*/*

snprintf**(**payload**,sizeof(**payload**),**

"%s,%d,%s,%llu," */\* Info about this sentinel. \*/*

"%s,%s,%d,%llu"**,** */\* Info about current master. \*/*

announce\_ip**,** announce\_port**,** server**.**runid**,**

**(**unsigned long long**)** sentinel**.**current\_epoch**,**

*/\* --- \*/*

master**->**name**,**master\_addr**->**ip**,**master\_addr**->**port**,**

**(**unsigned long long**)** master**->**config\_epoch**);**

retval **=** redisAsyncCommand**(**ri**->**cc**,**

sentinelPublishReplyCallback**,** **NULL,** "PUBLISH %s %s"**,**

SENTINEL\_HELLO\_CHANNEL**,**payload**);**

**if** **(**retval **!=** REDIS\_OK**)** **return** REDIS\_ERR**;**

ri**->**pending\_commands**++;**

**return** REDIS\_OK**;**

**}**

首先得到实例ri所属的主节点实例master；然后调用sentinelGetCurrentMasterAddress函数得到master的地址信息；

如果实例ri的标志位中具有SRI\_DISCONNECTED标记的话，直接返回；

如果当前哨兵配置了sentinel.announce\_ip的话，则使用该ip信息作为自己的ip地址，否则，调用anetSockName函数，根据socket描述符得到当前哨兵的ip地址；

如果当前哨兵配置了sentinel.announce\_port的话，则使用该port信息作为自己的端口信息，否则，使用server.port作为当前哨兵的端口信息；

接下来组装要发布的HELLO信息，HELLO信息的格式是："sentinel\_ip,sentinel\_port,sentinel\_runid,current\_epoch,master\_name,master\_ip,master\_port,master\_config\_epoch"

接下来，向ri异步发送"PUBLISH \_\_sentinel\_\_:hello <HELLO>"命令，设置命令回调函数为sentinelPublishReplyCallback；

当哨兵收到实例对于该”PUBLISH”命令的回复之后，会调用回调函数sentinelPublishReplyCallback，该函数只用于更新属性ri->last\_pub\_time，对回复内容无需关心：

void sentinelPublishReplyCallback**(**redisAsyncContext **\***c**,** void **\***reply**,** void **\***privdata**)** **{**

sentinelRedisInstance **\***ri **=** c**->**data**;**

redisReply **\***r**;**

REDIS\_NOTUSED**(**privdata**);**

**if** **(**ri**)** ri**->**pending\_commands**--;**

**if** **(!**reply **||** **!**ri**)** **return;**

r **=** reply**;**

*/\* Only update pub\_time if we actually published our message. Otherwise*

*\* we'll retry again in 100 milliseconds. \*/*

**if** **(**r**->**type **!=** REDIS\_REPLY\_ERROR**)**

ri**->**last\_pub\_time **=** mstime**();**

**}**

之前在介绍sentinelReconnectInstance函数时讲过，当哨兵向主节点或从节点实例建立订阅连接时，向实例发送” SUBSCRIBE \_\_sentinel\_\_:hello"命令，订阅HELLO频道时，设置该命令的回调函数为sentinelReceiveHelloMessages。因此，当收到该频道上发布的消息时，就会调用函数sentinelReceiveHelloMessages。

该频道上的消息，是监控同一实例的其他哨兵节点发来的HELLO消息，当前哨兵通过HELLO消息，来发现其他哨兵，并且相互之间交互最新的主节点信息。sentinelReceiveHelloMessages函数的代码如下：

void sentinelReceiveHelloMessages**(**redisAsyncContext **\***c**,** void **\***reply**,** void **\***privdata**)** **{**

sentinelRedisInstance **\***ri **=** c**->**data**;**

redisReply **\***r**;**

REDIS\_NOTUSED**(**privdata**);**

**if** **(!**reply **||** **!**ri**)** **return;**

r **=** reply**;**

*/\* Update the last activity in the pubsub channel. Note that since we*

*\* receive our messages as well this timestamp can be used to detect*

*\* if the link is probably disconnected even if it seems otherwise. \*/*

ri**->**pc\_last\_activity **=** mstime**();**

*/\* Sanity check in the reply we expect, so that the code that follows*

*\* can avoid to check for details. \*/*

**if** **(**r**->**type **!=** REDIS\_REPLY\_ARRAY **||**

r**->**elements **!=** 3 **||**

r**->**element**[**0**]->**type **!=** REDIS\_REPLY\_STRING **||**

r**->**element**[**1**]->**type **!=** REDIS\_REPLY\_STRING **||**

r**->**element**[**2**]->**type **!=** REDIS\_REPLY\_STRING **||**

strcmp**(**r**->**element**[**0**]->**str**,**"message"**)** **!=** 0**)** **return;**

*/\* We are not interested in meeting ourselves \*/*

**if** **(**strstr**(**r**->**element**[**2**]->**str**,**server**.**runid**)** **!=** **NULL)** **return;**

sentinelProcessHelloMessage**(**r**->**element**[**2**]->**str**,** r**->**element**[**2**]->**len**);**

**}**

该函数中，首先更新ri->pc\_last\_activity为当前时间；

然后判断是否处理接收到的消息，注意，只处理"message"消息，也就是说不会处理"subscribe"消息；

注意，如果收到的"message"消息中，包含了自身的runid，说明这是本哨兵自己发送的消息，因此无需处理，直接返回；

最后，调用sentinelProcessHelloMessage函数处理收到的HELLO消息；

注意：在测试时发现会收到从节点重复的HELLO消息，也就是同一时间，同一个哨兵发布的两条一模一样的消息。这是因为哨兵向主节点发送的”PUBLISH”命令，会因为主从复制的原因，而同步到从节点；而同时该哨兵也向从节点发送”PUBLISH”命令，因此，从节点就会在同一时间，收到两条一模一样的HELLO消息，并将它们发布到频道上。

另外，一旦哨兵发现了其他哨兵之后，可以直接向其发送"PUBLISH \_\_sentinel\_\_:hello <HELLO>"命令。哨兵自己实现了”PUBLISH”的处理函数sentinelPublishCommand，当收到其他哨兵直接发来的HELLO消息时，就会调用该函数处理。该函数的代码如下：

void sentinelPublishCommand**(**redisClient **\***c**)** **{**

**if** **(**strcmp**(**c**->**argv**[**1**]->**ptr**,**SENTINEL\_HELLO\_CHANNEL**))** **{**

addReplyError**(**c**,** "Only HELLO messages are accepted by Sentinel instances."**);**

**return;**

**}**

sentinelProcessHelloMessage**(**c**->**argv**[**2**]->**ptr**,**sdslen**(**c**->**argv**[**2**]->**ptr**));**

addReplyLongLong**(**c**,**1**);**

**}**

因此，不管是从真正的订阅频道中收到HELLO消息，还是直接收到其他哨兵发来的”PUBLISH”命令，最终都是通过sentinelProcessHelloMessage函数对HELLO消息进行处理的。该函数的代码如下：

void sentinelProcessHelloMessage**(**char **\***hello**,** int hello\_len**)** **{**

*/\* Format is composed of 8 tokens:*

*\* 0=ip,1=port,2=runid,3=current\_epoch,4=master\_name,*

*\* 5=master\_ip,6=master\_port,7=master\_config\_epoch. \*/*

int numtokens**,** port**,** removed**,** master\_port**;**

uint64\_t current\_epoch**,** master\_config\_epoch**;**

char **\*\***token **=** sdssplitlen**(**hello**,** hello\_len**,** ","**,** 1**,** **&**numtokens**);**

sentinelRedisInstance **\***si**,** **\***master**;**

**if** **(**numtokens **==** 8**)** **{**

*/\* Obtain a reference to the master this hello message is about \*/*

master **=** sentinelGetMasterByName**(**token**[**4**]);**

**if** **(!**master**)** **goto** cleanup**;** */\* Unknown master, skip the message. \*/*

*/\* First, try to see if we already have this sentinel. \*/*

port **=** atoi**(**token**[**1**]);**

master\_port **=** atoi**(**token**[**6**]);**

si **=** getSentinelRedisInstanceByAddrAndRunID**(**

master**->**sentinels**,**token**[**0**],**port**,**token**[**2**]);**

current\_epoch **=** strtoull**(**token**[**3**],NULL,**10**);**

master\_config\_epoch **=** strtoull**(**token**[**7**],NULL,**10**);**

**if** **(!**si**)** **{**

*/\* If not, remove all the sentinels that have the same runid*

*\* OR the same ip/port, because it's either a restart or a*

*\* network topology change. \*/*

removed **=** removeMatchingSentinelsFromMaster**(**master**,**token**[**0**],**port**,**

token**[**2**]);**

**if** **(**removed**)** **{**

sentinelEvent**(**REDIS\_NOTICE**,**"-dup-sentinel"**,**master**,**

"%@ #duplicate of %s:%d or %s"**,**

token**[**0**],**port**,**token**[**2**]);**

**}**

*/\* Add the new sentinel. \*/*

si **=** createSentinelRedisInstance**(NULL,**SRI\_SENTINEL**,**

token**[**0**],**port**,**master**->**quorum**,**master**);**

**if** **(**si**)** **{**

sentinelEvent**(**REDIS\_NOTICE**,**"+sentinel"**,**si**,**"%@"**);**

*/\* The runid is NULL after a new instance creation and*

*\* for Sentinels we don't have a later chance to fill it,*

*\* so do it now. \*/*

si**->**runid **=** sdsnew**(**token**[**2**]);**

sentinelFlushConfig**();**

**}**

**}**

*/\* Update local current\_epoch if received current\_epoch is greater.\*/*

**if** **(**current\_epoch **>** sentinel**.**current\_epoch**)** **{**

sentinel**.**current\_epoch **=** current\_epoch**;**

sentinelFlushConfig**();**

sentinelEvent**(**REDIS\_WARNING**,**"+new-epoch"**,**master**,**"%llu"**,**

**(**unsigned long long**)** sentinel**.**current\_epoch**);**

**}**

*/\* Update master info if received configuration is newer. \*/*

**if** **(**master**->**config\_epoch **<** master\_config\_epoch**)** **{**

master**->**config\_epoch **=** master\_config\_epoch**;**

**if** **(**master\_port **!=** master**->**addr**->**port **||**

strcmp**(**master**->**addr**->**ip**,** token**[**5**]))**

**{**

sentinelAddr **\***old\_addr**;**

sentinelEvent**(**REDIS\_WARNING**,**"+config-update-from"**,**si**,**"%@"**);**

sentinelEvent**(**REDIS\_WARNING**,**"+switch-master"**,**

master**,**"%s %s %d %s %d"**,**

master**->**name**,**

master**->**addr**->**ip**,** master**->**addr**->**port**,**

token**[**5**],** master\_port**);**

old\_addr **=** dupSentinelAddr**(**master**->**addr**);**

sentinelResetMasterAndChangeAddress**(**master**,** token**[**5**],** master\_port**);**

sentinelCallClientReconfScript**(**master**,**

SENTINEL\_OBSERVER**,**"start"**,**

old\_addr**,**master**->**addr**);**

releaseSentinelAddr**(**old\_addr**);**

**}**

**}**

*/\* Update the state of the Sentinel. \*/*

**if** **(**si**)** si**->**last\_hello\_time **=** mstime**();**

**}**

cleanup**:**

sdsfreesplitres**(**token**,**numtokens**);**

**}**

首先，根据消息中的master\_name，调用函数sentinelGetMasterByName，在字典sentinel.masters中寻找相应的主节点实例master，如果找不到，则直接退出；

然后，调用getSentinelRedisInstanceByAddrAndRunID函数，根据消息中的sentinel\_ip,sentinel\_port和sentinel\_runid信息，在字典master->sentinels中，找到runid，ip和port都匹配的哨兵实例。

如果没有找到匹配的哨兵实例，要么这是一个新发现的哨兵，要么是某个哨兵的信息发生了变化（比如有可能某个哨兵实例重启了，导致runid发生了变化；或者网络拓扑发生了变化，导致ip或port发生了变化）。

这种情况下，首先调用函数removeMatchingSentinelsFromMaster，删除字典master->sentinels中，具有相同runid，或者具有相同ip和port的哨兵实例；然后根据HELLO消息中的ip和port信息，重新创建一个新的哨兵实例，添加到字典master->sentinels中，这样下次调用sentinelReconnectInstance时，就会向该哨兵实例进行建链了。；

如果找到了匹配的哨兵实例，并且HELLO消息中的sentinel\_current\_epoch，大于本实例当前的current\_epoch，则更新本实例的current\_epoch属性；

如果HELLO消息中的master\_config\_epoch，大于本实例记录的master的config\_epoch，则更新本实例记录的master的config\_epoch。并且如果HELLO消息中的master\_ip或master\_port，与本实例记录的主节点的ip或port信息不匹配的话，则说明可能发生了故障转移，某个从节点升级成为了新的主节点，因此调用sentinelResetMasterAndChangeAddress函数，重置主节点，及其从节点实例的信息；

最后，更新si->last\_hello\_time属性为当前时间；

3：”INFO”命令

“INFO”命令，主要用于哨兵获取主从节点实例当前的状态和信息，比如该实例当前是主节点还是从节点；该实例反馈的IP地址和PORT信息，是否与本哨兵记录的一样；该实例如果是主节点的话，那它都有哪些从节点；该实例如果是从节点的话，它与主节点是否连通，它的优先级是多少，它的复制偏移量是多少等等，这些信息在故障转移流程中，是判断实例状态的重要信息；

在sentinelSendPeriodicCommands函数中，设置的”INFO”命令的回调函数是sentinelInfoReplyCallback。该函数的代码很简单，主要是调用sentinelRefreshInstanceInfo函数对回复进行处理。因此，主要看一下sentinelRefreshInstanceInfo函数的代码：

void sentinelRefreshInstanceInfo**(**sentinelRedisInstance **\***ri**,** const char **\***info**)** **{**

sds **\***lines**;**

int numlines**,** j**;**

int role **=** 0**;**

*/\* The following fields must be reset to a given value in the case they*

*\* are not found at all in the INFO output. \*/*

ri**->**master\_link\_down\_time **=** 0**;**

*/\* Process line by line. \*/*

lines **=** sdssplitlen**(**info**,**strlen**(**info**),**"\r\n"**,**2**,&**numlines**);**

**for** **(**j **=** 0**;** j **<** numlines**;** j**++)** **{**

sentinelRedisInstance **\***slave**;**

sds l **=** lines**[**j**];**

*/\* run\_id:<40 hex chars>\*/*

**if** **(**sdslen**(**l**)** **>=** 47 **&&** **!**memcmp**(**l**,**"run\_id:"**,**7**))** **{**

**if** **(**ri**->**runid **==** **NULL)** **{**

ri**->**runid **=** sdsnewlen**(**l**+**7**,**40**);**

**}** **else** **{**

**if** **(**strncmp**(**ri**->**runid**,**l**+**7**,**40**)** **!=** 0**)** **{**

sentinelEvent**(**REDIS\_NOTICE**,**"+reboot"**,**ri**,**"%@"**);**

sdsfree**(**ri**->**runid**);**

ri**->**runid **=** sdsnewlen**(**l**+**7**,**40**);**

**}**

**}**

**}**

*/\* old versions: slave0:<ip>,<port>,<state>*

*\* new versions: slave0:ip=127.0.0.1,port=9999,... \*/*

**if** **((**ri**->**flags **&** SRI\_MASTER**)** **&&**

sdslen**(**l**)** **>=** 7 **&&**

**!**memcmp**(**l**,**"slave"**,**5**)** **&&** isdigit**(**l**[**5**]))**

**{**

char **\***ip**,** **\***port**,** **\***end**;**

**if** **(**strstr**(**l**,**"ip="**)** **==** **NULL)** **{**

*/\* Old format. \*/*

ip **=** strchr**(**l**,**':'**);** **if** **(!**ip**)** **continue;**

ip**++;** */\* Now ip points to start of ip address. \*/*

port **=** strchr**(**ip**,**','**);** **if** **(!**port**)** **continue;**

**\***port **=** '\0'**;** */\* nul term for easy access. \*/*

port**++;** */\* Now port points to start of port number. \*/*

end **=** strchr**(**port**,**','**);** **if** **(!**end**)** **continue;**

**\***end **=** '\0'**;** */\* nul term for easy access. \*/*

**}** **else** **{**

*/\* New format. \*/*

ip **=** strstr**(**l**,**"ip="**);** **if** **(!**ip**)** **continue;**

ip **+=** 3**;** */\* Now ip points to start of ip address. \*/*

port **=** strstr**(**l**,**"port="**);** **if** **(!**port**)** **continue;**

port **+=** 5**;** */\* Now port points to start of port number. \*/*

*/\* Nul term both fields for easy access. \*/*

end **=** strchr**(**ip**,**','**);** **if** **(**end**)** **\***end **=** '\0'**;**

end **=** strchr**(**port**,**','**);** **if** **(**end**)** **\***end **=** '\0'**;**

**}**

*/\* Check if we already have this slave into our table,*

*\* otherwise add it. \*/*

**if** **(**sentinelRedisInstanceLookupSlave**(**ri**,**ip**,**atoi**(**port**))** **==** **NULL)** **{**

**if** **((**slave **=** createSentinelRedisInstance**(NULL,**SRI\_SLAVE**,**ip**,**

atoi**(**port**),** ri**->**quorum**,** ri**))** **!=** **NULL)**

**{**

sentinelEvent**(**REDIS\_NOTICE**,**"+slave"**,**slave**,**"%@"**);**

sentinelFlushConfig**();**

**}**

**}**

**}**

*/\* master\_link\_down\_since\_seconds:<seconds> \*/*

**if** **(**sdslen**(**l**)** **>=** 32 **&&**

**!**memcmp**(**l**,**"master\_link\_down\_since\_seconds"**,**30**))**

**{**

ri**->**master\_link\_down\_time **=** strtoll**(**l**+**31**,NULL,**10**)\***1000**;**

**}**

*/\* role:<role> \*/*

**if** **(!**memcmp**(**l**,**"role:master"**,**11**))** role **=** SRI\_MASTER**;**

**else** **if** **(!**memcmp**(**l**,**"role:slave"**,**10**))** role **=** SRI\_SLAVE**;**

**if** **(**role **==** SRI\_SLAVE**)** **{**

*/\* master\_host:<host> \*/*

**if** **(**sdslen**(**l**)** **>=** 12 **&&** **!**memcmp**(**l**,**"master\_host:"**,**12**))** **{**

**if** **(**ri**->**slave\_master\_host **==** **NULL** **||**

strcasecmp**(**l**+**12**,**ri**->**slave\_master\_host**))**

**{**

sdsfree**(**ri**->**slave\_master\_host**);**

ri**->**slave\_master\_host **=** sdsnew**(**l**+**12**);**

ri**->**slave\_conf\_change\_time **=** mstime**();**

**}**

**}**

*/\* master\_port:<port> \*/*

**if** **(**sdslen**(**l**)** **>=** 12 **&&** **!**memcmp**(**l**,**"master\_port:"**,**12**))** **{**

int slave\_master\_port **=** atoi**(**l**+**12**);**

**if** **(**ri**->**slave\_master\_port **!=** slave\_master\_port**)** **{**

ri**->**slave\_master\_port **=** slave\_master\_port**;**

ri**->**slave\_conf\_change\_time **=** mstime**();**

**}**

**}**

*/\* master\_link\_status:<status> \*/*

**if** **(**sdslen**(**l**)** **>=** 19 **&&** **!**memcmp**(**l**,**"master\_link\_status:"**,**19**))** **{**

ri**->**slave\_master\_link\_status **=**

**(**strcasecmp**(**l**+**19**,**"up"**)** **==** 0**)** **?**

SENTINEL\_MASTER\_LINK\_STATUS\_UP **:**

SENTINEL\_MASTER\_LINK\_STATUS\_DOWN**;**

**}**

*/\* slave\_priority:<priority> \*/*

**if** **(**sdslen**(**l**)** **>=** 15 **&&** **!**memcmp**(**l**,**"slave\_priority:"**,**15**))**

ri**->**slave\_priority **=** atoi**(**l**+**15**);**

*/\* slave\_repl\_offset:<offset> \*/*

**if** **(**sdslen**(**l**)** **>=** 18 **&&** **!**memcmp**(**l**,**"slave\_repl\_offset:"**,**18**))**

ri**->**slave\_repl\_offset **=** strtoull**(**l**+**18**,NULL,**10**);**

**}**

**}**

ri**->**info\_refresh **=** mstime**();**

sdsfreesplitres**(**lines**,**numlines**);**

*/\* ---------------------------- Acting half -----------------------------*

*\* Some things will not happen if sentinel.tilt is true, but some will*

*\* still be processed. \*/*

*/\* Remember when the role changed. \*/*

**if** **(**role **!=** ri**->**role\_reported**)** **{**

ri**->**role\_reported\_time **=** mstime**();**

ri**->**role\_reported **=** role**;**

**if** **(**role **==** SRI\_SLAVE**)** ri**->**slave\_conf\_change\_time **=** mstime**();**

*/\* Log the event with +role-change if the new role is coherent or*

*\* with -role-change if there is a mismatch with the current config. \*/*

sentinelEvent**(**REDIS\_VERBOSE**,**

**((**ri**->**flags **&** **(**SRI\_MASTER**|**SRI\_SLAVE**))** **==** role**)** **?**

"+role-change" **:** "-role-change"**,**

ri**,** "%@ new reported role is %s"**,**

role **==** SRI\_MASTER **?** "master" **:** "slave"**,**

ri**->**flags **&** SRI\_MASTER **?** "master" **:** "slave"**);**

**}**

*/\* None of the following conditions are processed when in tilt mode, so*

*\* return asap. \*/*

**if** **(**sentinel**.**tilt**)** **return;**

*/\* Handle master -> slave role switch. \*/*

**if** **((**ri**->**flags **&** SRI\_MASTER**)** **&&** role **==** SRI\_SLAVE**)** **{**

*/\* Nothing to do, but masters claiming to be slaves are*

*\* considered to be unreachable by Sentinel, so eventually*

*\* a failover will be triggered. \*/*

**}**

**...**

**}**

该函数首先在for循环中解析"INFO"回复信息：

首先解析出"run\_id"之后的信息，保存在ri->runid中。如果该实例的runid发生了变化，还需要记录日志，向"+reboot"频道发布消息；

如果实例为主节点，则解析"slave"后的从节点信息，取出其中的ip和port信息，然后根据ip和port，调用sentinelRedisInstanceLookupSlave函数，在字典ri->slaves中寻找是否已经保存了该从节点的信息。如果没有，则调用createSentinelRedisInstance创建从节点实例，并插入到ri->slaves中，也就是发现了主节点属下的从节点，下次调用函数sentinelReconnectInstance时，就会向该从节点建链了；

解析"master\_link\_down\_since\_seconds"信息，该信息表示从节点与主节点的断链时间。将其转换成整数后，记录到ri->master\_link\_down\_time中；

解析"role"信息，如果包含"role:master"，则置role为SRI\_MASTER，说明该实例报告自己为主节点；如果包含"role:slave"，则置role为SRI\_SLAVE，说明该实例报告自己为从节点；

如果role为SRI\_SLAVE，找到回复信息中的"master\_host:"信息，记录到ri->slave\_master\_host中；找到回复信息中的"master\_port:"信息，记录到ri->slave\_master\_port中；找到回复信息中的"master\_link\_status:"信息，根据其值是否为"up"，记录到ri->slave\_master\_link\_status中；找到回复信息中的"slave\_priority:"信息，记录到ri->slave\_priority中；找到回复信息中的"slave\_repl\_offset:"信息，记录到ri->slave\_repl\_offset中；

解析完所有"INFO"回复信息之后，更新ri->info\_refresh为当前时间；

接下来根据实例的角色信息执行一些动作：

ri->role\_reported的初始值是根据ri->flags得到的，如果收到"INFO"回复后，解析得到的role与ri->role\_reported不同，说明该实例的角色发生了变化，比如从主节点变成了从节点，或者相反。只要role与ri->role\_reported不同，就首先更新ri->role\_reported\_time为当前时间，并且将ri->role\_reported置为role；如果role为SRI\_SLAVE，还需要更新ri->slave\_conf\_change\_time的值为当前时间；最后，还根据ri->flags中的角色是否与role，来记录日志，发布信息；

如果当前哨兵已经进入了TILT模式，则直接返回；

如果ri->flags中为主节点，但是role为从节点，这种情况无需采取动作，因为这种情况会被视为主节点不可达，最终会引发故障迁移流程；

本函数剩下的动作，与故障转移流程有关，后续在介绍。

**七：判断实例是否主观下线**

首先解释一下主观下线和客观下线的区别。

所谓主观下线，就是从“我”（当前实例）的角度来看，某个实例已经下线了。但是单个哨兵的视角可能是盲目的，仅从“我”的角度，就决定一个实例下线是武断的。因此，“我”还会通过命令询问其他哨兵节点，看它们是否也认为该实例已经下线了，如果超过quorum个（包括“我”）哨兵反馈认为该实例已经下线了，则“我”就会认为该实例确实已经下线了，也就是所谓的客观下线了。

判断某个实例主观下线，主要是根据其是否能及时回复”PING”命令决定的。因此，首先看一下发送”PING”命令的函数sentinelSendPing的实现：

int sentinelSendPing**(**sentinelRedisInstance **\***ri**)** **{**

int retval **=** redisAsyncCommand**(**ri**->**cc**,**

sentinelPingReplyCallback**,** **NULL,** "PING"**);**

**if** **(**retval **==** REDIS\_OK**)** **{**

ri**->**pending\_commands**++;**

*/\* We update the ping time only if we received the pong for*

*\* the previous ping, otherwise we are technically waiting*

*\* since the first ping that did not received a reply. \*/*

**if** **(**ri**->**last\_ping\_time **==** 0**)** ri**->**last\_ping\_time **=** mstime**();**

**return** 1**;**

**}** **else** **{**

**return** 0**;**

**}**

**}**

在该函数中，设置收到”PING”命令回复后的回调函数为sentinelPingReplyCallback。

需要注意的是，如果ri->last\_ping\_time值为0，则更新ri->last\_ping\_time为当前时间。而只有在收到"PING"命令的正常回复之后，ri->last\_ping\_time的值才会被置为0。

下面是回调函数sentinelPingReplyCallback的代码：

void sentinelPingReplyCallback**(**redisAsyncContext **\***c**,** void **\***reply**,** void **\***privdata**)** **{**

sentinelRedisInstance **\***ri **=** c**->**data**;**

redisReply **\***r**;**

REDIS\_NOTUSED**(**privdata**);**

**if** **(**ri**)** ri**->**pending\_commands**--;**

**if** **(!**reply **||** **!**ri**)** **return;**

r **=** reply**;**

**if** **(**r**->**type **==** REDIS\_REPLY\_STATUS **||**

r**->**type **==** REDIS\_REPLY\_ERROR**)** **{**

*/\* Update the "instance available" field only if this is an*

*\* acceptable reply. \*/*

**if** **(**strncmp**(**r**->**str**,**"PONG"**,**4**)** **==** 0 **||**

strncmp**(**r**->**str**,**"LOADING"**,**7**)** **==** 0 **||**

strncmp**(**r**->**str**,**"MASTERDOWN"**,**10**)** **==** 0**)**

**{**

ri**->**last\_avail\_time **=** mstime**();**

ri**->**last\_ping\_time **=** 0**;** */\* Flag the pong as received. \*/*

**}** **else** **{**

*/\* Send a SCRIPT KILL command if the instance appears to be*

*\* down because of a busy script. \*/*

**if** **(**strncmp**(**r**->**str**,**"BUSY"**,**4**)** **==** 0 **&&**

**(**ri**->**flags **&** SRI\_S\_DOWN**)** **&&**

**!(**ri**->**flags **&** SRI\_SCRIPT\_KILL\_SENT**))**

**{**

**if** **(**redisAsyncCommand**(**ri**->**cc**,**

sentinelDiscardReplyCallback**,** **NULL,**

"SCRIPT KILL"**)** **==** REDIS\_OK**)**

ri**->**pending\_commands**++;**

ri**->**flags **|=** SRI\_SCRIPT\_KILL\_SENT**;**

**}**

**}**

**}**

ri**->**last\_pong\_time **=** mstime**();**

**}**

如果回复信息为"PONG"，"LOADING"或"MASTERDOWN"，表示正常回复，因此置该实例的属性ri->last\_avail\_time为当前时间，并且置ri->last\_ping\_time为0，这样下次发送"PING"命令时就会更新ri->last\_ping\_time的值了；

如果回复信息以"BUSY"开头，并且该实例已经被置为主观下线，并且还没有向该实例发送过"SCRIPT KILL"命令，则向该实例发送"SCRIPT KILL"命令；

最后，不管回复信息是什么，更新ri->last\_pong\_time为当前时间。

因此，有关”PING”命令的时间属性总结如下：

ri->last\_ping\_time：上一次正常发送”PING”命令的时间。需要注意的是，只有当收到"PING"命令的正常回复后，下次发送"PING"命令时才会更新该属性为当时时间戳。如果发送”PING”命令后，没有收到任何回复，或者没有收到正常回复，则下次发送”PING”命令时，就不会更新该属性。如果该属性值为0，说明已经收到了上一个"PING"命令的正常回复，但是还没有开始发送下一个"PING"命令。检测实例是否主观下线，主要就是根据该属性判断的。

ri->last\_pong\_time：每当收到"PING"命令的回复后，不管是否是正常恢复，都会更新该属性为当时时间戳；

在哨兵的“主函数”sentinelHandleRedisInstance中，调用sentinelCheckSubjectivelyDown函数检测实例是否主观下线，该函数同时还会检测TCP连接是否正常。该函数的代码如下：

void sentinelCheckSubjectivelyDown**(**sentinelRedisInstance **\***ri**)** **{**

mstime\_t elapsed **=** 0**;**

**if** **(**ri**->**last\_ping\_time**)**

elapsed **=** mstime**()** **-** ri**->**last\_ping\_time**;**

*/\* Check if we are in need for a reconnection of one of the*

*\* links, because we are detecting low activity.*

*\**

*\* 1) Check if the command link seems connected, was connected not less*

*\* than SENTINEL\_MIN\_LINK\_RECONNECT\_PERIOD, but still we have a*

*\* pending ping for more than half the timeout. \*/*

**if** **(**ri**->**cc **&&**

**(**mstime**()** **-** ri**->**cc\_conn\_time**)** **>** SENTINEL\_MIN\_LINK\_RECONNECT\_PERIOD **&&**

ri**->**last\_ping\_time **!=** 0 **&&** */\* Ther is a pending ping... \*/*

*/\* The pending ping is delayed, and we did not received*

*\* error replies as well. \*/*

**(**mstime**()** **-** ri**->**last\_ping\_time**)** **>** **(**ri**->**down\_after\_period**/**2**)** **&&**

**(**mstime**()** **-** ri**->**last\_pong\_time**)** **>** **(**ri**->**down\_after\_period**/**2**))**

**{**

sentinelKillLink**(**ri**,**ri**->**cc**);**

**}**

*/\* 2) Check if the pubsub link seems connected, was connected not less*

*\* than SENTINEL\_MIN\_LINK\_RECONNECT\_PERIOD, but still we have no*

*\* activity in the Pub/Sub channel for more than*

*\* SENTINEL\_PUBLISH\_PERIOD \* 3.*

*\*/*

**if** **(**ri**->**pc **&&**

**(**mstime**()** **-** ri**->**pc\_conn\_time**)** **>** SENTINEL\_MIN\_LINK\_RECONNECT\_PERIOD **&&**

**(**mstime**()** **-** ri**->**pc\_last\_activity**)** **>** **(**SENTINEL\_PUBLISH\_PERIOD**\***3**))**

**{**

sentinelKillLink**(**ri**,**ri**->**pc**);**

**}**

*/\* Update the SDOWN flag. We believe the instance is SDOWN if:*

*\**

*\* 1) It is not replying.*

*\* 2) We believe it is a master, it reports to be a slave for enough time*

*\* to meet the down\_after\_period, plus enough time to get two times*

*\* INFO report from the instance. \*/*

**if** **(**elapsed **>** ri**->**down\_after\_period **||**

**(**ri**->**flags **&** SRI\_MASTER **&&**

ri**->**role\_reported **==** SRI\_SLAVE **&&**

mstime**()** **-** ri**->**role\_reported\_time **>**

**(**ri**->**down\_after\_period**+**SENTINEL\_INFO\_PERIOD**\***2**)))**

**{**

*/\* Is subjectively down \*/*

**if** **((**ri**->**flags **&** SRI\_S\_DOWN**)** **==** 0**)** **{**

sentinelEvent**(**REDIS\_WARNING**,**"+sdown"**,**ri**,**"%@"**);**

ri**->**s\_down\_since\_time **=** mstime**();**

ri**->**flags **|=** SRI\_S\_DOWN**;**

**}**

**}** **else** **{**

*/\* Is subjectively up \*/*

**if** **(**ri**->**flags **&** SRI\_S\_DOWN**)** **{**

sentinelEvent**(**REDIS\_WARNING**,**"-sdown"**,**ri**,**"%@"**);**

ri**->**flags **&=** **~(**SRI\_S\_DOWN**|**SRI\_SCRIPT\_KILL\_SENT**);**

**}**

**}**

**}**

ri->cc\_conn\_time属性表示上一次向该实例发起命令类型的TCP建链的时间；ri->pc\_conn\_time属性表示上一次向该实例发起订阅类型的TCP建链的时间；

首先计算elapsed的值，该值表示是当前时间与ri->last\_ping\_time之间的时间差；

然后判断命令类型的TCP连接是否正常，不正常的条件是：距离上次建链时已经超过了SENTINEL\_MIN\_LINK\_RECONNECT\_PERIOD，并且上次发送"PING"后还没有收到正常回复，且当前时间与ri->last\_ping\_time之间的时间差已经超过了ri->down\_after\_period/2，并且距离上次收到任何"PING"回复的时间，已经超过了ri->down\_after\_period/2；

如果命令类型的连接不正常了，则直接调用sentinelKillLink断开连接，释放异步上下文；

然后判断订阅类型的TCP连接是否正常，不正常的条件是：距离上次建链时已经超过了SENTINEL\_MIN\_LINK\_RECONNECT\_PERIOD，并且距离上次收到订阅频道发来的任何消息的时间，已经超过了SENTINEL\_PUBLISH\_PERIOD\*3；

如果订阅类型的连接不正常了，则直接调用sentinelKillLink断开连接，释放异步上下文；

如果elapsed的值大于ri->down\_after\_period，或者：当前实例我认为它是主节点，但是它的"INFO"回复中却报告自己是从节点，并且距离上次收到它在"INFO"回复中报告自己是从节点的时间，已经超过了ri->down\_after\_period+SENTINEL\_INFO\_PERIOD\*2；

满足以上任意一个条件，都认为该实例是主观下线了。因此：只要该实例还没有标志为主观下线，则将SRI\_S\_DOWN标记增加到实例标志位中，表示该实例主观下线；

如果不满足以上条件，但是该实例之前已经被标记为主观下线了，则认为该实例主观上线了，去掉其标志位中的SRI\_S\_DOWN和SRI\_SCRIPT\_KILL\_SENT标记；