**十一：TILT模式**

根据之前的介绍可知，哨兵的运行，非常依赖于系统时间，但是当系统时间被调整，或者哨兵中的流程因为某种原因（比如负载较高、IO发生阻塞、进程被信号停止等）而被阻塞时，哨兵的行为就会变得不可预知了。

所谓TILT模式，就是一种特殊的保护模式。进入TILT模式后，哨兵只定期发送命令用于收集信息，而不采取实质性的动作，比如不会进行故障转移流程。

当恢复正常30秒后，哨兵就是退出TILT模式。

在哨兵的定时器函数sentinelTimer中，首先就是调用函数sentinelCheckTiltCondition判断哨兵当前是否需要进入TILT模式。该函数的代码如下：

void sentinelCheckTiltCondition**(**void**)** **{**

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

mstime\_t delta **=** now **-** sentinel**.**previous\_time**;**

**if** **(**delta **<** 0 **||** delta **>** SENTINEL\_TILT\_TRIGGER**)** **{**

sentinel**.**tilt **=** 1**;**

sentinel**.**tilt\_start\_time **=** mstime**();**

sentinelEvent**(**REDIS\_WARNING**,**"+tilt"**,NULL,**"#tilt mode entered"**);**

**}**

sentinel**.**previous\_time **=** mstime**();**

**}**

正常情况下，本函数每隔100ms执行一次。每次执行都会更新sentinel.previous\_time属性。如果某次调用本函数时，发现当前时间与sentinel.previous\_time间的差值为负值，或者大于SENTINEL\_TILT\_TRIGGER(2000)，则置sentinel.tilt为1，说明哨兵进入了TILT模式，并且置sentinel.tilt\_start\_time为当前时间。

当进入TILT模式后，在收到其他实例的”INFO”命令回复后的回调函数sentinelRefreshInstanceInfo中，仅将收到的信息保存下来，而后续涉及到主从角色变化、故障转移流程等，都不再处理；而且当收到其他哨兵发来的，用于询问某主节点是否下线的"is-master-down-by-addr"命令时，一律回复“未下线”，因为处于TILT模式下的哨兵的判断，已经不可信了。

在哨兵的“主函数”sentinelHandleRedisInstance中，在调用函数sentinelSendPeriodicCommands发送完周期性的命令之后，有下面的代码：

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

*/\* ========== MONITORING HALF ============ \*/*

*/\* Every kind of instance \*/*

sentinelReconnectInstance**(**ri**);**

sentinelSendPeriodicCommands**(**ri**);**

*/\* ============== ACTING HALF ============= \*/*

*/\* We don't proceed with the acting half if we are in TILT mode.*

*\* TILT happens when we find something odd with the time, like a*

*\* sudden change in the clock. \*/*

**if** **(**sentinel**.**tilt**)** **{**

**if** **(**mstime**()-**sentinel**.**tilt\_start\_time **<** SENTINEL\_TILT\_PERIOD**)** **return;**

sentinel**.**tilt **=** 0**;**

sentinelEvent**(**REDIS\_WARNING**,**"-tilt"**,NULL,**"#tilt mode exited"**);**

**}**

*/\* Every kind of instance \*/*

sentinelCheckSubjectivelyDown**(**ri**);**

**...**

**}**

因此，在TILT模式下，仅仅发送命令收集信息，而不会进行故障转移流程相关的动作。并且，当哨兵处于TILT模式下连续超过SENTINEL\_TILT\_PERIOD（30秒）后，就会退出TILT模式。

**十二：执行脚本**

哨兵支持在发生某种事件，或者是因发生了故障转移而主节点的地址发生变化时，能够执行相应的脚本，以便通知系统管理员事件的发生，或是通知客户端主节点的新地址信息。

目前哨兵支持两种脚本。一种是当发生某种WARNING级别的事件（比如实例主观下线、客观下线等）时，调用脚本以便通过邮件、短信或者其他方式，将事件通知给系统管理员。脚本调用时，会传递两个参数，一是事件的类型，一是事件的描述信息。这种脚本可以通过配置文件中的” notification-script”选项配置：

sentinel notification**-**script mymaster **/**var**/**redis**/**notify**.**sh

另一种是当发生故障转移，导致主节点的地址信息发生了变化时，可以调用脚本通知连接Redis的客户端，使其能够感知到这种配置的变化，以及主节点的新地址信息。这种脚本的参数包括：

<master-name> <role> <state> <from-ip> <from-port> <to-ip> <to-port>

参数<role>，根据故障转移流程是否是当前哨兵为领导节点完成的，要么是”leader”，要么是”observer”；参数<state>，目前只能是”start”；参数<from-ip>和<from-port>，是原来主节点的地址信息；参数<to-ip>和<to-port>，是新主节点的地址信息。

这种脚本可以通过配置文件中的” client-reconfig-script”选项配置：

sentinel client**-**reconfig**-**script mymaster **/**var**/**redis**/**reconfig**.**sh

这两种脚本的调用规则和错误处理方式是：当脚本的退出码为1，或者因为收到某种信号导致脚本退出，则该脚本后续会被重试执行，最大的重试次数为10；当脚本的退出码为2（或者更高的值）时，脚本不会被重试执行；脚本的最长运行时间为60秒，运行时间超过该阈值的脚本会被KILL掉。

1：事件通知脚本

在哨兵的代码中，每当有事件发生时，就会调用sentinelEvent函数。该函数主要做三件事：将事件信息记录日志；将事件信息发布到某个频道上，订阅该频道的客户端可以接收到这种事件信息；创建用以执行事件通知脚本的任务。

sentinelEvent函数的代码如下：

void sentinelEvent**(**int level**,** char **\***type**,** sentinelRedisInstance **\***ri**,**

const char **\***fmt**,** **...)** **{**

va\_list ap**;**

char msg**[**REDIS\_MAX\_LOGMSG\_LEN**];**

robj **\***channel**,** **\***payload**;**

*/\* Handle %@ \*/*

**if** **(**fmt**[**0**]** **==** '%' **&&** fmt**[**1**]** **==** '@'**)** **{**

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

**NULL** **:** ri**->**master**;**

**if** **(**master**)** **{**

snprintf**(**msg**,** **sizeof(**msg**),** "%s %s %s %d @ %s %s %d"**,**

sentinelRedisInstanceTypeStr**(**ri**),**

ri**->**name**,** ri**->**addr**->**ip**,** ri**->**addr**->**port**,**

master**->**name**,** master**->**addr**->**ip**,** master**->**addr**->**port**);**

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

snprintf**(**msg**,** **sizeof(**msg**),** "%s %s %s %d"**,**

sentinelRedisInstanceTypeStr**(**ri**),**

ri**->**name**,** ri**->**addr**->**ip**,** ri**->**addr**->**port**);**

**}**

fmt **+=** 2**;**

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

msg**[**0**]** **=** '\0'**;**

**}**

*/\* Use vsprintf for the rest of the formatting if any. \*/*

**if** **(**fmt**[**0**]** **!=** '\0'**)** **{**

va\_start**(**ap**,** fmt**);**

vsnprintf**(**msg**+**strlen**(**msg**),** **sizeof(**msg**)-**strlen**(**msg**),** fmt**,** ap**);**

va\_end**(**ap**);**

**}**

*/\* Log the message if the log level allows it to be logged. \*/*

**if** **(**level **>=** server**.**verbosity**)**

redisLog**(**level**,**"%s %s"**,**type**,**msg**);**

*/\* Publish the message via Pub/Sub if it's not a debugging one. \*/*

**if** **(**level **!=** REDIS\_DEBUG**)** **{**

channel **=** createStringObject**(**type**,**strlen**(**type**));**

payload **=** createStringObject**(**msg**,**strlen**(**msg**));**

pubsubPublishMessage**(**channel**,**payload**);**

decrRefCount**(**channel**);**

decrRefCount**(**payload**);**

**}**

*/\* Call the notification script if applicable. \*/*

**if** **(**level **==** REDIS\_WARNING **&&** ri **!=** **NULL)** **{**

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

ri **:** ri**->**master**;**

**if** **(**master**->**notification\_script**)** **{**

sentinelScheduleScriptExecution**(**master**->**notification\_script**,**

type**,**msg**,NULL);**

**}**

**}**

**}**

参数level表示日志级别，还用于控制是否将事件发布到相应频道，以及是否创建任务；参数type表示事件名，比如"+monitor"，"+slave"，"+role-change"等，该参数还是发布频道的频道名；ri表示触发事件的实例；后面的参数表示事件的描述消息；

该函数中，首先处理可变参数，组装事件消息msg；

如果level大于等于server.verbosity，则将type和msg记录到日志中；

如果level不是REDIS\_DEBUG，则将msg发布到以type为名的频道中；

如果level为REDIS\_WARNING，并且ri不为NULL，则先根据ri找到相应的主节点master，如果该master配置了事件通知脚本的话，则调用函数sentinelScheduleScriptExecution创建任务节点，后续该任务会以type和msg为参数执行notification\_script脚本；

2：客户端重配置脚本

当发生故障转移流程后，主节点的信息发生变化。哨兵感知到这种变化后，就会调用sentinelCallClientReconfScript函数，该函数会创建执行客户端重配置脚本的任务。

sentinelCallClientReconfScript函数的代码如下：

void sentinelCallClientReconfScript**(**sentinelRedisInstance **\***master**,** int role**,** char **\***state**,** sentinelAddr **\***from**,** sentinelAddr **\***to**)** **{**

char fromport**[**32**],** toport**[**32**];**

**if** **(**master**->**client\_reconfig\_script **==** **NULL)** **return;**

ll2string**(**fromport**,sizeof(**fromport**),**from**->**port**);**

ll2string**(**toport**,sizeof(**toport**),**to**->**port**);**

sentinelScheduleScriptExecution**(**master**->**client\_reconfig\_script**,**

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

**(**role **==** SENTINEL\_LEADER**)** **?** "leader" **:** "observer"**,**

state**,** from**->**ip**,** fromport**,** to**->**ip**,** toport**,** **NULL);**

**}**

该函数只在两个地方调用，一是当前哨兵为领导节点进行故障转移时，选中的从节点在其"INFO"命令回复信息中，表明其已升级为主节点时。这中情况下，参数role为SENTINEL\_LEADER；一是当前哨兵收到其他哨兵发来的HELLO消息，发现其中的主节点信息与当前哨兵记录的主节点信息不一致时。这种情况下，参数role为SENTINEL\_OBSERVER；

本函数用于创建执行脚本master->client\_reconfig\_script的任务，如果master->client\_reconfig\_script属性为NULL，则说明未配置该脚本，因此直接返回；

然后调用sentinelScheduleScriptExecution函数，根据脚本名，及其参数，创建任务节点。

3：创建新任务

脚本都是由任务执行的，任务以节点的形式存放到列表sentinel.scripts\_queue中。创建新任务的函数是sentinelScheduleScriptExecution，代码如下：

void sentinelScheduleScriptExecution**(**char **\***path**,** **...)** **{**

va\_list ap**;**

char **\***argv**[**SENTINEL\_SCRIPT\_MAX\_ARGS**+**1**];**

int argc **=** 1**;**

sentinelScriptJob **\***sj**;**

va\_start**(**ap**,** path**);**

**while(**argc **<** SENTINEL\_SCRIPT\_MAX\_ARGS**)** **{**

argv**[**argc**]** **=** va\_arg**(**ap**,**char**\*);**

**if** **(!**argv**[**argc**])** **break;**

argv**[**argc**]** **=** sdsnew**(**argv**[**argc**]);** */\* Copy the string. \*/*

argc**++;**

**}**

va\_end**(**ap**);**

argv**[**0**]** **=** sdsnew**(**path**);**

sj **=** zmalloc**(sizeof(\***sj**));**

sj**->**flags **=** SENTINEL\_SCRIPT\_NONE**;**

sj**->**retry\_num **=** 0**;**

sj**->**argv **=** zmalloc**(sizeof(**char**\*)\*(**argc**+**1**));**

sj**->**start\_time **=** 0**;**

sj**->**pid **=** 0**;**

memcpy**(**sj**->**argv**,**argv**,sizeof(**char**\*)\*(**argc**+**1**));**

listAddNodeTail**(**sentinel**.**scripts\_queue**,**sj**);**

*/\* Remove the oldest non running script if we already hit the limit. \*/*

**if** **(**listLength**(**sentinel**.**scripts\_queue**)** **>** SENTINEL\_SCRIPT\_MAX\_QUEUE**)** **{**

listNode **\***ln**;**

listIter li**;**

listRewind**(**sentinel**.**scripts\_queue**,&**li**);**

**while** **((**ln **=** listNext**(&**li**))** **!=** **NULL)** **{**

sj **=** ln**->**value**;**

**if** **(**sj**->**flags **&** SENTINEL\_SCRIPT\_RUNNING**)** **continue;**

*/\* The first node is the oldest as we add on tail. \*/*

listDelNode**(**sentinel**.**scripts\_queue**,**ln**);**

sentinelReleaseScriptJob**(**sj**);**

**break;**

**}**

redisAssert**(**listLength**(**sentinel**.**scripts\_queue**)** **<=**

SENTINEL\_SCRIPT\_MAX\_QUEUE**);**

**}**

**}**

参数path为任务要执行的脚本路径，之后的参数就是该脚本执行时的参数。

首先将所有可变参数记录到数组argv中，然后将脚本路径记录到argv[0]中；

然后创建任务结构sj，初始化该结构的属性，并将数组argv复制到sj->argv中；

然后将sj追加到列表sentinel.scripts\_queue的结尾；

如果列表当前长度超过了SENTINEL\_SCRIPT\_MAX\_QUEUE(256)，则需要删除最早添加的任务。因此轮训列表，找到第一个当前未执行的任务，将其从列表中删除；

4：执行任务

在哨兵的定时器函数sentinelTimer中，会调用sentinelRunPendingScripts函数，依次执行列表sentinel.scripts\_queue中的任务。该函数的代码如下：

void sentinelRunPendingScripts**(**void**)** **{**

listNode **\***ln**;**

listIter li**;**

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

*/\* Find jobs that are not running and run them, from the top to the*

*\* tail of the queue, so we run older jobs first. \*/*

listRewind**(**sentinel**.**scripts\_queue**,&**li**);**

**while** **(**sentinel**.**running\_scripts **<** SENTINEL\_SCRIPT\_MAX\_RUNNING **&&**

**(**ln **=** listNext**(&**li**))** **!=** **NULL)**

**{**

sentinelScriptJob **\***sj **=** ln**->**value**;**

pid\_t pid**;**

*/\* Skip if already running. \*/*

**if** **(**sj**->**flags **&** SENTINEL\_SCRIPT\_RUNNING**)** **continue;**

*/\* Skip if it's a retry, but not enough time has elapsed. \*/*

**if** **(**sj**->**start\_time **&&** sj**->**start\_time **>** now**)** **continue;**

sj**->**flags **|=** SENTINEL\_SCRIPT\_RUNNING**;**

sj**->**start\_time **=** mstime**();**

sj**->**retry\_num**++;**

pid **=** fork**();**

**if** **(**pid **==** **-**1**)** **{**

*/\* Parent (fork error).*

*\* We report fork errors as signal 99, in order to unify the*

*\* reporting with other kind of errors. \*/*

sentinelEvent**(**REDIS\_WARNING**,**"-script-error"**,NULL,**

"%s %d %d"**,** sj**->**argv**[**0**],** 99**,** 0**);**

sj**->**flags **&=** **~**SENTINEL\_SCRIPT\_RUNNING**;**

sj**->**pid **=** 0**;**

**}** **else** **if** **(**pid **==** 0**)** **{**

*/\* Child \*/*

execve**(**sj**->**argv**[**0**],**sj**->**argv**,**environ**);**

*/\* If we are here an error occurred. \*/*

\_exit**(**2**);** */\* Don't retry execution. \*/*

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

sentinel**.**running\_scripts**++;**

sj**->**pid **=** pid**;**

sentinelEvent**(**REDIS\_DEBUG**,**"+script-child"**,NULL,**"%ld"**,(**long**)**pid**);**

**}**

**}**

**}**

sentinel.running\_scripts表示当前正在运行的子进程数，也就是正在运行的任务数。如果该值小于SENTINEL\_SCRIPT\_MAX\_RUNNING(16)，则轮训列表sentinel.scripts\_queue中的每个任务节点：

如果该任务节点的标志位中设置了SENTINEL\_SCRIPT\_RUNNING，说明该任务正在运行，因此直接忽略该任务节点；

创建任务节点时，其start\_time属性置为0，当运行该任务时，就会将start\_time置为当时时间。如果任务运行失败，且需要重试时，则将其置为下次运行该任务的时间。因此如果该属性不为0，且其值大于当前时间，说明该任务还不到运行的时候，因此直接忽略该任务节点；

接下来就可以运行该任务节点了。首先将SENTINEL\_SCRIPT\_RUNNING标记增加到其标志位中；然后设置任务的start\_time属性为当前时间；增加任务的retry\_num值，该属性表示任务重试次数；

然后就是调用fork创建子进程。创建子进程失败，则将SENTINEL\_SCRIPT\_RUNNING标记从任务标志位中清除，这样下次调用本函数时，会重新运行该任务；创建子任务成功，则在子进程中调用execve执行脚本；在父进程中，将子进程pid记录到任务的pid属性中，并增加sentinel.running\_scripts的值。

5：收集任务执行状态

在哨兵的定时器函数sentinelTimer中，会调用sentinelCollectTerminatedScripts函数，收集终止任务的结束状态，主要是判断任务是否需要重试执行。该函数的代码如下：

void sentinelCollectTerminatedScripts**(**void**)** **{**

int statloc**;**

pid\_t pid**;**

**while** **((**pid **=** wait3**(&**statloc**,**WNOHANG**,NULL))** **>** 0**)** **{**

int exitcode **=** WEXITSTATUS**(**statloc**);**

int bysignal **=** 0**;**

listNode **\***ln**;**

sentinelScriptJob **\***sj**;**

**if** **(**WIFSIGNALED**(**statloc**))** bysignal **=** WTERMSIG**(**statloc**);**

sentinelEvent**(**REDIS\_DEBUG**,**"-script-child"**,NULL,**"%ld %d %d"**,**

**(**long**)**pid**,** exitcode**,** bysignal**);**

ln **=** sentinelGetScriptListNodeByPid**(**pid**);**

**if** **(**ln **==** **NULL)** **{**

redisLog**(**REDIS\_WARNING**,**"wait3() returned a pid (%ld) we can't find in our scripts execution queue!"**,** **(**long**)**pid**);**

**continue;**

**}**

sj **=** ln**->**value**;**

*/\* If the script was terminated by a signal or returns an*

*\* exit code of "1" (that means: please retry), we reschedule it*

*\* if the max number of retries is not already reached. \*/*

**if** **((**bysignal **||** exitcode **==** 1**)** **&&**

sj**->**retry\_num **!=** SENTINEL\_SCRIPT\_MAX\_RETRY**)**

**{**

sj**->**flags **&=** **~**SENTINEL\_SCRIPT\_RUNNING**;**

sj**->**pid **=** 0**;**

sj**->**start\_time **=** mstime**()** **+**

sentinelScriptRetryDelay**(**sj**->**retry\_num**);**

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

*/\* Otherwise let's remove the script, but log the event if the*

*\* execution did not terminated in the best of the ways. \*/*

**if** **(**bysignal **||** exitcode **!=** 0**)** **{**

sentinelEvent**(**REDIS\_WARNING**,**"-script-error"**,NULL,**

"%s %d %d"**,** sj**->**argv**[**0**],** bysignal**,** exitcode**);**

**}**

listDelNode**(**sentinel**.**scripts\_queue**,**ln**);**

sentinelReleaseScriptJob**(**sj**);**

sentinel**.**running\_scripts**--;**

**}**

**}**

**}**

本函数就是以参数WNOHANG循环调用wait3，只要当前已经有终止子进程了，则wait3返回该子进程的pid，否则返回负值，直接退出循环。在循环中：

首先取得子进程的退出状态；

如果子进程是因为接收到信号后而终止的，则取得该信号值bysignal；

然后调用函数sentinelGetScriptListNodeByPid，根据子进程的pid，找到任务列表sentinel.scripts\_queue中对应的任务节点sj；

如果子进程是由信号终止的，或者子进程的退出状态为"1"，并且任务的重试次数不等于SENTINEL\_SCRIPT\_MAX\_RETRY(10)，则该任务可以重新执行。因此先将SENTINEL\_SCRIPT\_RUNNING标记从任务标志位中清除，然后置任务pid为0，然后调用函数sentinelScriptRetryDelay，得到该任务下一次执行的时间，记录到任务的start\_time属性中；

其他情况下，要么任务执行成功了，要么任务退出码不是1，则都需要将该任务节点从列表sentinel.scripts\_queue中删除，并且减少sentinel.running\_scripts的值；

ps：这里感觉有BUG，当任务需要重试时，也需要减少sentinel.running\_scripts的值；

6：杀死执行超时的任务

在哨兵的定时器函数sentinelTimer中，会调用sentinelKillTimedoutScripts函数，杀死那些执行时间超过60秒的任务。该函数的代码如下：

void sentinelKillTimedoutScripts**(**void**)** **{**

listNode **\***ln**;**

listIter li**;**

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

listRewind**(**sentinel**.**scripts\_queue**,&**li**);**

**while** **((**ln **=** listNext**(&**li**))** **!=** **NULL)** **{**

sentinelScriptJob **\***sj **=** ln**->**value**;**

**if** **(**sj**->**flags **&** SENTINEL\_SCRIPT\_RUNNING **&&**

**(**now **-** sj**->**start\_time**)** **>** SENTINEL\_SCRIPT\_MAX\_RUNTIME**)**

**{**

sentinelEvent**(**REDIS\_WARNING**,**"-script-timeout"**,NULL,**"%s %ld"**,**

sj**->**argv**[**0**],** **(**long**)**sj**->**pid**);**

kill**(**sj**->**pid**,**SIGKILL**);**

**}**

**}**

**}**

该函数很简单，就是轮训列表sentinel.scripts\_queue，针对其中的每个任务，如果该任务正在执行，并且执行时间已经超过了60秒，则调用kill，向该任务发送SIGKILL信号，杀死该子进程。

PS:

关于哨兵，就暂时到这里了，呵呵…

更多关于函数的注释，参考：

https://github.com/gqtc/redis-3.0.5/blob/master/redis-3.0.5/src/sentinel.c