Redis sentinel原理及实现源码剖析

冯光普 @ 多点Dmall

2020-08

About me

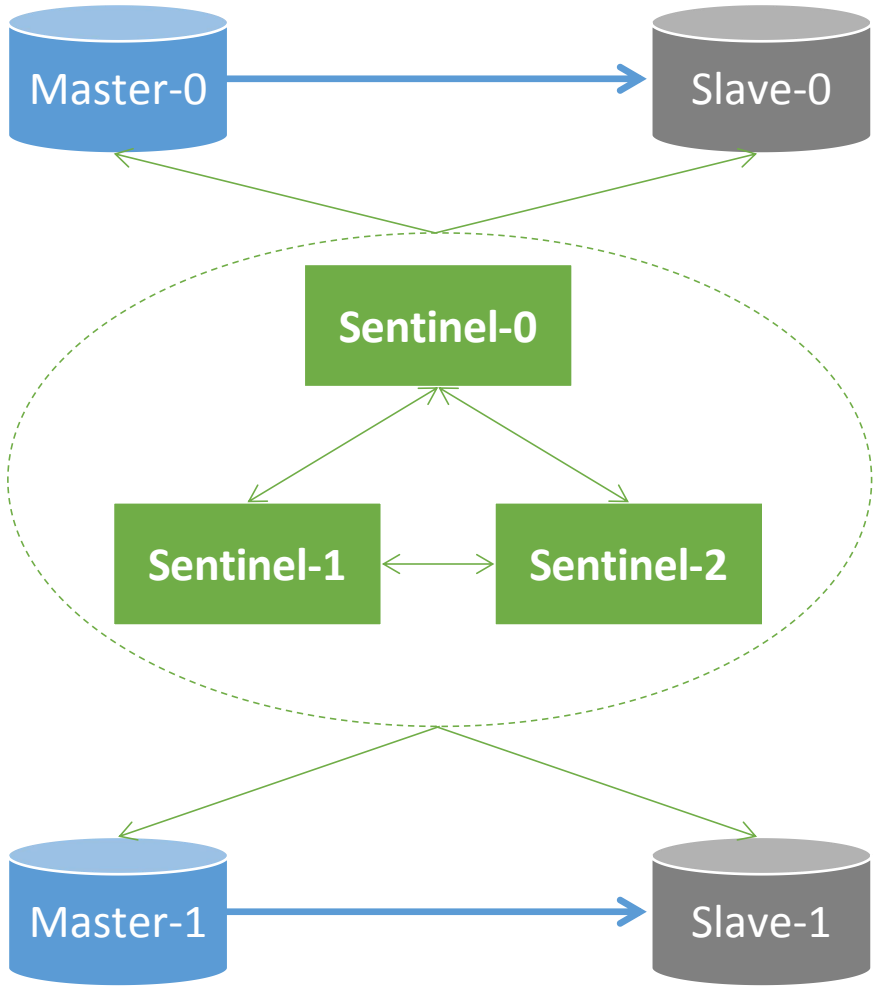
- 多点Dmall数据库负责人
  - MySQL、Redis、MongoDB
  - DB PaaS平台建设
- AliSQL内核研发（2012 ~ 2014）
  - bugfix、特性开发、技术研究

多点 dmall.com

主要内容

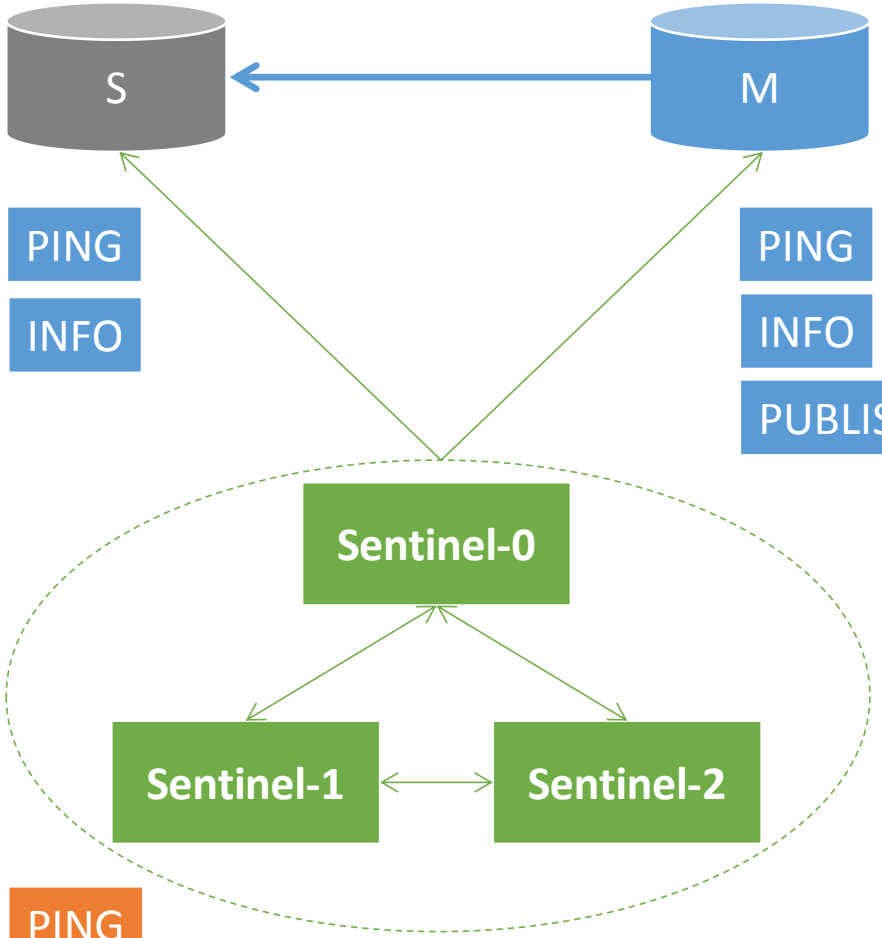
- Sentinel原理简介
  - 特性、监控原理、failover流程
- 源码实现剖析
  - 总体设计、raft选举、failover细节
- 实践建议及讨论
  - 参数配置、自定义脚本、线上案例

Sentinel原理 - 特性



- 官方实现
- 分布式集群
- Raft选举failover
- 自定义脚本调用支持
- 多master监控支持

Sentinel原理 - 监控



Redis/Sentinel实例存活检测，每1s

Redis实例信息：角色、master、slave-list，每10s

Sentinel上报监控的master信息，每2s

__sentinel__:hello
sentinel_ip,sentinel_port,sentinel_runid,current_epoch,
master_name,master_ip,master_port,master_config_epoch

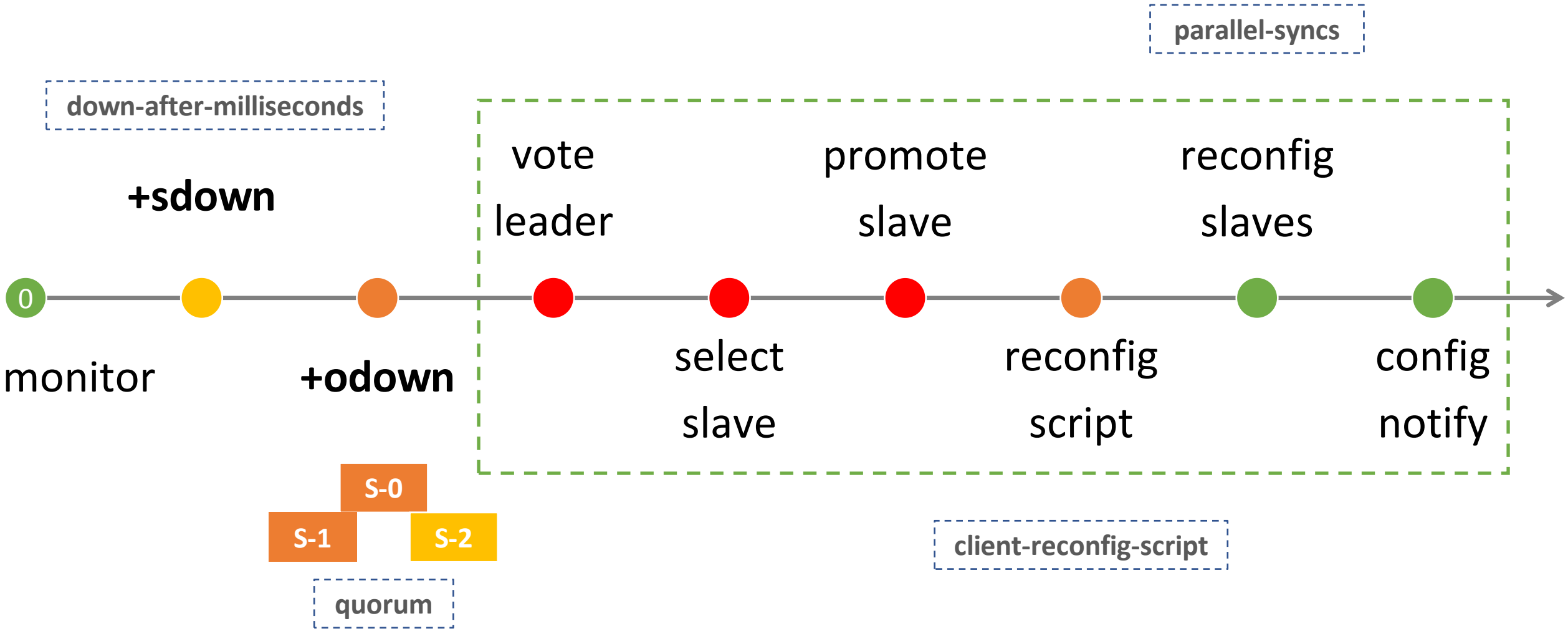
SENTINEL MONITOR *mymaster master_ip master_port quorum*

自动发现其他Sentinel节点

SENTINEL is-master-down-by-addr <ip> <port> <current-epoch> <runid>

master存活交流、leader选举

Sentinel原理 - 切换



Sentinel源码实现剖析

- 总体设计
- 多Master监控
- Sentinel -> (Master, Slave, Sentinel) 命令交互
- Raft选举
- Failover流程

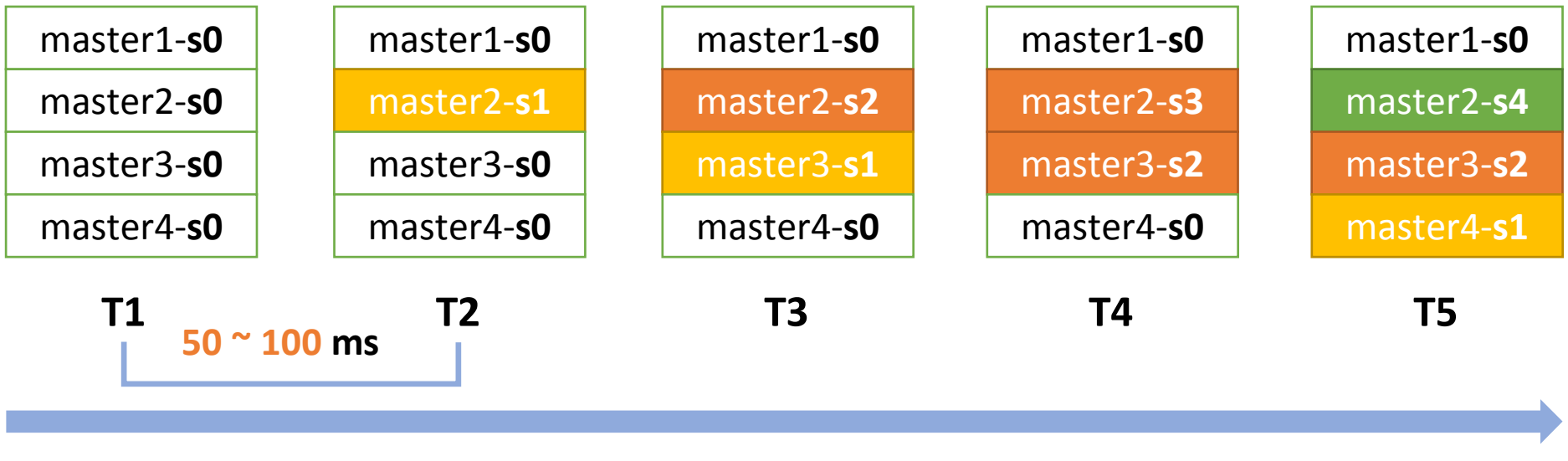
Sentinel源码剖析 - 总体设计

1. **单进程**
   无锁
2. **定时器**
   时间片
3. **异步IO**
   无阻塞
4. **状态机**
   可重入

Sentinel源码剖析 - 定时器/事件循环

- int main - *server.cc*
  - initSentinelConfig()
  - initSentinel()
  - initServer()
    - server.hz = ...
    - aeCreateTimeEvent(server.el, 1, serverCron...)
      - **sentinelTimer()**
  - aeMain(server.el)
    - aeProcessEvents(eventLoop...) - *ae.c*
      - aeApiPoll(eventLoop, tvp)
      - fe->rfileProc(eventLoop,fd...
      - fe->wfileProc(eventLoop,fd...
      - processTimeEvents(eventLoop)

Sentinel源码剖析 - 多master监控

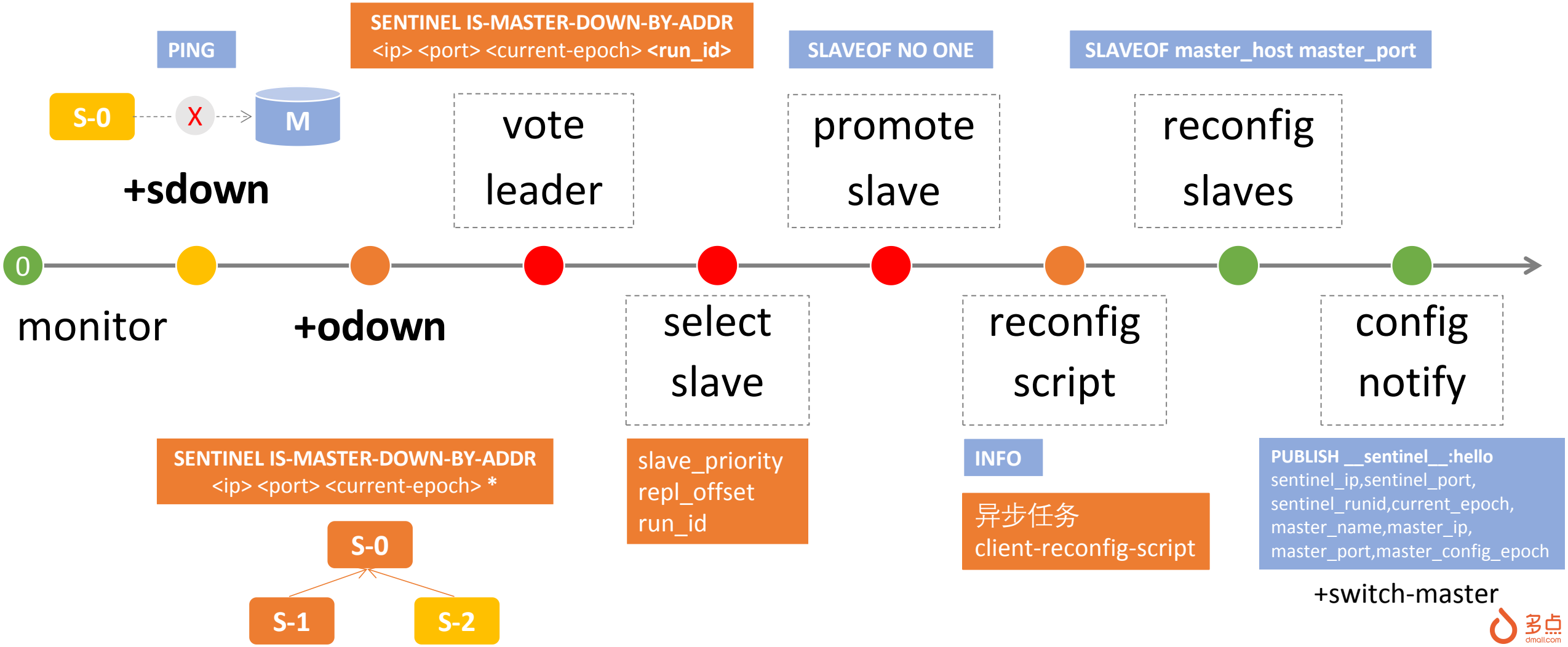



```
void sentinelTimer(void) {
    ....
    sentinelHandleDictOfRedisInstances(sentinel.masters);

    server.hz = CONFIG_DEFAULT_HZ + rand() % CONFIG_DEFAULT_HZ;
}
```

避免Vote共振

Sentinel源码剖析 - 命令交互



Sentinel源码剖析 - raft选举

Raft选举关键步骤

Follower接收Leader心跳RPC超时，发起选举

- term += 1
- 启动定时器（随机因子）
- 给自己投票
- RequestVote，请求其它节点投票
- 选票超过大多数，成为新Leader

split vote：candidate在定时器超时后，发起下一轮选举

约定：对其他节点term更大vote请求，无条件投票，同一term内，投票后结果不变

Sentinel选举步骤

确认+odown后，发起failover选举

- epoch += 1
- => failover_start_time（随机因子），异步请求投票
- 首轮，给自己投票
- 第二轮，获取其它节点投票
- 选票超过max(majority, quorum)，成为failover Leader

split vote：candidate在2*failover_timeout时间后，发起下一轮选举

约定：对其他节点epoch更大vote请求，无条件投票，同一epoch内，投票后结果不变，成为Follower后，在2*failover_timeout 时间内，不再发起failover选举

Sentinel源码剖析 - failover入口

```
sentinelTimer
    sentinelHandleDictOfRedisInstances(sentinel.masters);
    sentinelRunPendingScripts();
```

遍历，处理所有监控的集群

fork，执行reconfig，notify脚本

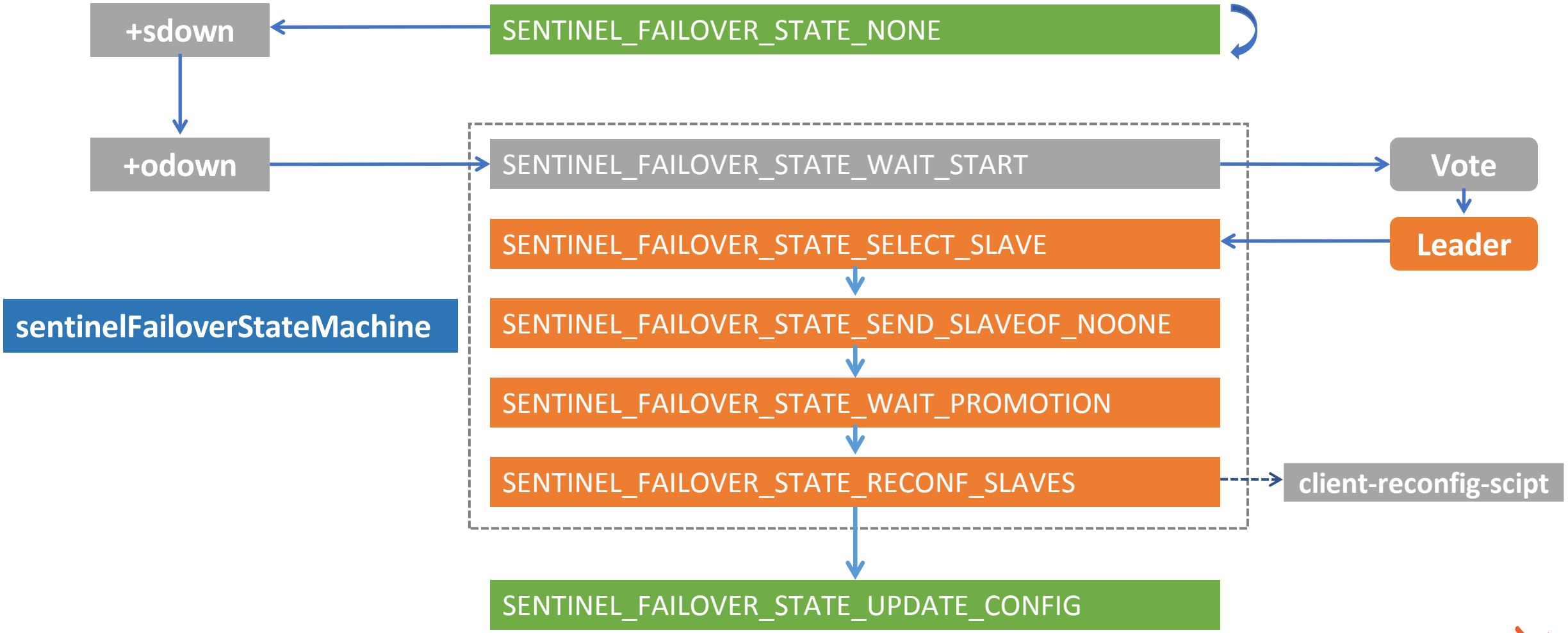
```
sentinelHandleDictOfRedisInstances > sentinelHandleRedisInstance
    ...
1   sentinelCheckSubjectivelyDown(ri);          +sdown
    ...
    /* Only masters */
    if (ri->flags & SRI_MASTER) {
3       sentinelCheckObjectivelyDown(ri);       +odown
4       if (sentinelStartFailoverIfNeeded(ri))  启动failover
5           sentinelAskMasterStateToOtherSentinels(ri,SENTINEL_ASK_FORCED);
6 7 8 ..    sentinelFailoverStateMachine(ri);   failover状态机
2       sentinelAskMasterStateToOtherSentinels(ri,SENTINEL_NO_FLAGS);
    }
```

发起Leader投票，下轮获取结果

Sentinel之间询问，确认+down

SENTINEL IS-MASTER-DOWN-BY-ADDR <ip> <port> <current-epoch> *

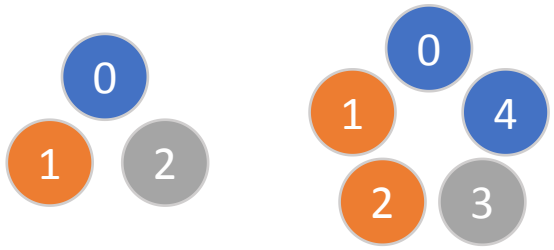
Sentinel源码剖析 - failover状态机



Sentinel实践 - 参数配置

集群规模及quorum

- 3节点？5节点？7节点？



failover-timeout（default: 180s）

- 发生split vote后，2*failover-timeout后才能发起下一次选举
- SLAVEOF NO ONE最大等待时间，否则放弃failover
- 超过failover-timeout后，配置新拓扑，忽略parallel-syncs参数
- 建议配置：？

Sentinel实践 - 自定义脚本

自定义脚本

- **client-reconfig-script**: VIP/DNS/配置中心
  - 通过INFO观察到新master up后，异步调用
- notification-script: WARNING级日志事件

1 监控集群不宜多，16个以内安全

2 自定义脚本需快速结束，< 60s

3 脚本exit code不为1，否则重试

任务队列

- SENTINEL_SCRIPT_**MAX_RUNNING** 16
- SENTINEL_SCRIPT_**MAX_QUEUE** 256
- SENTINEL_SCRIPT_**MAX_RUNTIME** 60000
- SENTINEL_SCRIPT_MAX_RETRY 10
- SENTINEL_SCRIPT_RETRY_DELAY 30000

```
# Sentinel
sentinel_masters:208
sentinel_tilt:0
sentinel_running_scripts:16
sentinel_scripts_queue_length:256
```

Sentinel实践 - 线上案例1

sentinelScheduleScriptExecution

```
638
639     listAddNodeTail(sentinel.scripts_queue,sj);
640
641     /* Remove the oldest non running script if we already hit the limit. */
642     if (listLength(sentinel.scripts_queue) > SENTINEL_SCRIPT_MAX_QUEUE) {
643         listNode *ln;
644         listIter li;
645
646         listRewind(sentinel.scripts_queue,&li);
647         while ((ln = listNext(&li)) != NULL) {
648             sj = ln->value;
649
650             if (sj->flags & SENTINEL_SCRIPT_RUNNING) continue;
651             /* The first node is the oldest as we add on tail. */
652             listDelNode(sentinel.scripts_queue,ln);
653             sentinelReleaseScriptJob(sj);
654             break;
655         }
656         redisAssert(listLength(sentinel.scripts_queue) <=
657                     SENTINEL_SCRIPT_MAX_QUEUE);
658     }
659 }
```

任务队列，尾部添加

256

丢弃任务，没有日志

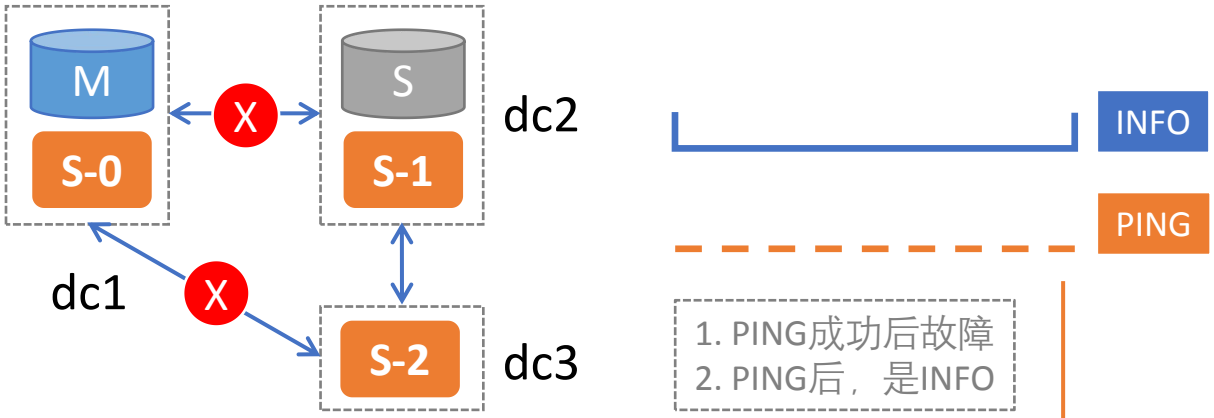
最多256个pending任务

修复脚本耗时bug解决

拓扑切换成功，
流量切换失败

```
# Sentinel
sentinel_masters:208
sentinel_tilt:0
sentinel_running_scripts:16
sentinel_scripts_queue_length:256
```

Sentinel实践 - 线上案例2



TCP重传超时后
才检测到+sdown

版本升级至 5.0.5解决

https://github.com/antirez/redis/issues/2819

sentinelSendPeriodicCommands

```
2367    if ((ri->flags & SRI_SENTINEL) == 0 &&
2368        (ri->info_refresh == 0 ||
2369        (now - ri->info_refresh) > info_period))
2370    {
2371        /* Send INFO to masters and slaves, not sentinels. */
2372        retval = redisAsyncCommand(ri->cc,
2373            sentinelInfoReplyCallback, NULL, "INFO");
2374        if (retval == REDIS_OK) ri->pending_commands++;
2375    } else if ((now - ri->last_pong_time) > ping_period) {
2376        /* Send PING to all the three kinds of instances. */
2377        sentinelSendPing(ri);
2378    } else if ((now - ri->last_pub_time) > SENTINEL_PUBLISH_PERIOD) {
2379        /* PUBLISH hello messages to all the three kinds of instances.
2380        sentinelSendHello(ri);
```

sentinelCheckSubjectivelyDown

```
2975  /* Is this instance down from our point of view? */
2976  void sentinelCheckSubjectivelyDown(sentinelRedisInstance *ri) {
2977      mstime_t elapsed = 0;
2978
2979      if (ri->last_ping_time)
2980          elapsed = mstime() - ri->last_ping_time;
```

sentinelPingReplyCallback

```
2060      if (r->type == REDIS_REPLY_STATUS ||
2061          r->type == REDIS_REPLY_ERROR) {
2062          /* Update the "instance available" field only if this is an
2063           * acceptable reply. */
2064          if (strncmp(r->str,"PONG",4) == 0 ||
2065              strncmp(r->str,"LOADING",7) == 0 ||
2066              strncmp(r->str,"MASTERDOWN",10) == 0)
2067          {
2068              ri->last_avail_time = mstime();
2069              ri->last_ping_time = 0; /* Flag the pong as received. */
```

Q & A

Thanks!

Reference

- https://redis.io/topics/sentinel
- https://github.com/antirez/redis/issues/2819
- https://github.com/antirez/redis/commit/8631e6477904f3d8f87662fd93a1ba294615654a