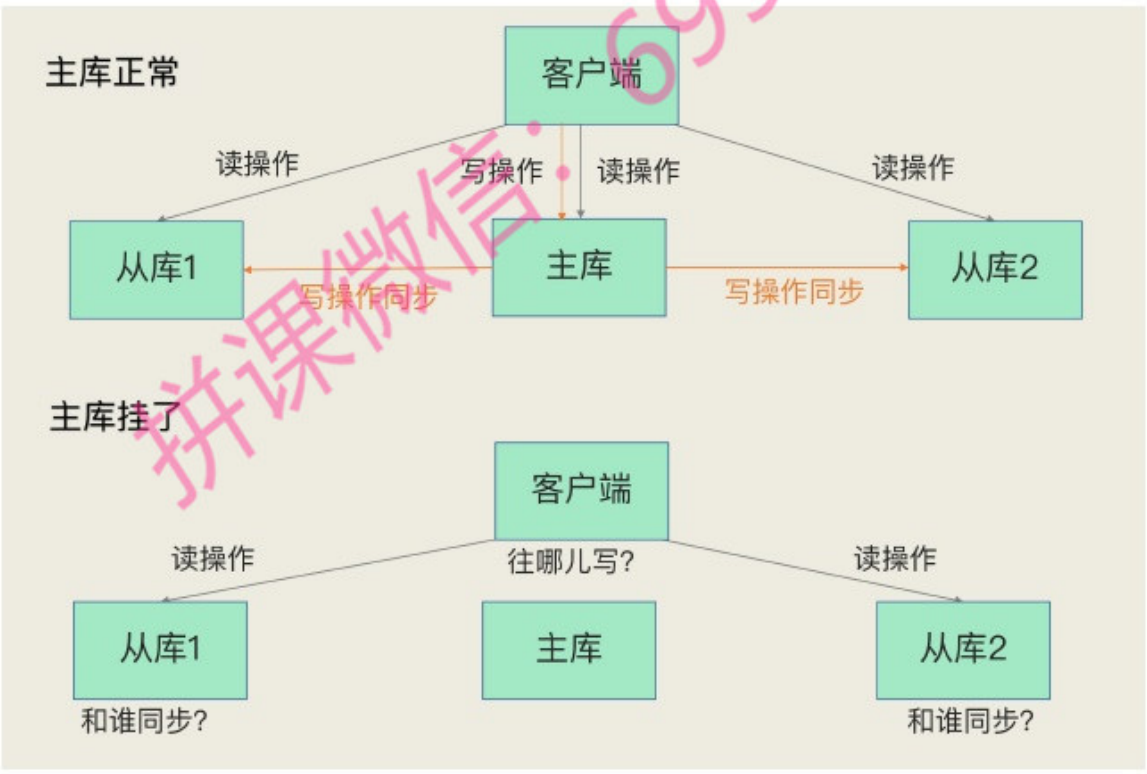
07-哨兵机制：主库挂了，如何不间断服务？

你好，我是蒋德钧。

上节课，我们学习了主从库集群模式。在这个模式下，如果从库发生故障了，客户端可以继续向主库或其他从库发送请求，进行相关的操作，但是如果主库发生故障了，那就直接会影响到从库的同步，因为从库没有相应的主库可以进行数据复制操作了。

而且，如果客户端发送的都是读操作请求，那还可以由从库继续提供服务，这在纯读的业务场景下还能被接受。但是，一旦有写操作请求了，按照主从库模式下的读写分离要求，需要由主库来完成写操作。此时，也没有实例可以来服务客户端的写操作请求了，如下图所示：



无论是写服务中断，还是从库无法进行数据同步，都是不能接受的。所以，如果主库挂了，我们就需要运行一个新主库，比如说把一个从库切换为主库，把它当成主库。这就涉及到三个问题：

1. 主库真的挂了吗？
2. 该选择哪个从库作为主库？
3. 怎么把新主库的相关信息通知给从库和客户端呢？

这就要提到哨兵机制了。在Redis主从集群中，哨兵机制是实现主从库自动切换的关键机制，它有效地解决了主从复制模式下故障转移的这三个问题。

接下来，我们就一起学习下哨兵机制。

哨兵机制的基本流程

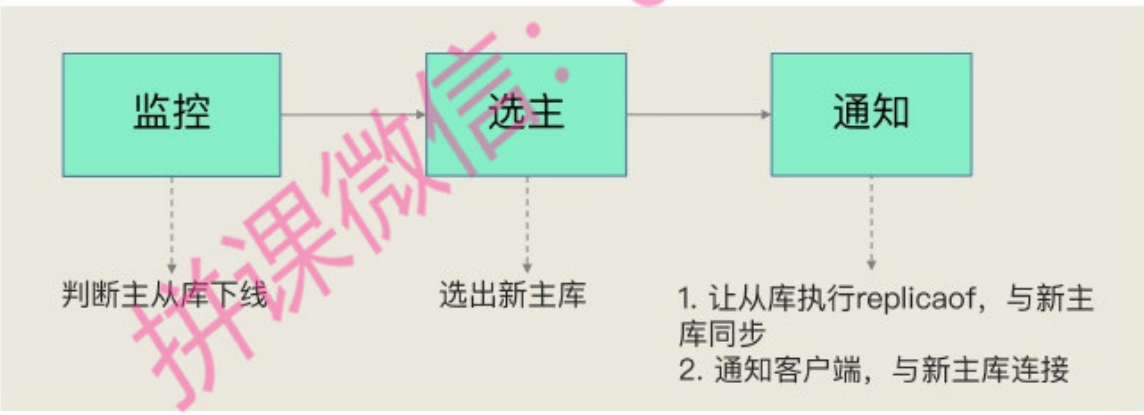
哨兵其实就是一个运行在特殊模式下的Redis进程，主从库实例运行的同时，它也在运行。哨兵主要负责的

我们先看监控。监控是指哨兵进程在运行时，周期性地给所有的主从库发送PING命令，检测它们是否仍然在线运行。如果从库没有在规定时间内响应哨兵的PING命令，哨兵就会把它标记为“下线状态”；同样，如果主库也没有在规定时间内响应哨兵的PING命令，哨兵就会判定主库下线，然后开始**自动切换主库**的流程。

这个流程首先是执行哨兵的第二个任务，选主。主库挂了以后，哨兵就需要从很多个从库里，按照一定的规则选择一个从库实例，把它作为新的主库。这一步完成后，现在的集群里就有了新主库。

然后，哨兵会执行最后一个任务：通知。在执行通知任务时，哨兵会把新主库的连接信息发给其他从库，让它们执行replicaof命令，和新主库建立连接，并进行数据复制。同时，哨兵会把新主库的连接信息通知给客户端，让它们把请求操作发到新主库上。

我画了一张图片，展示了这三个任务以及它们各自的目标。



在这三个任务中，通知任务相对来说比较简单，哨兵只需要把新主库信息发给从库和客户端，让它们和新主库建立连接就行，并不涉及决策的逻辑。但是，在监控和选主这两个任务中，哨兵需要做出两个决策：

- 在监控任务中，哨兵需要判断主库是否处于下线状态；
- 在选主任务中，哨兵也要决定选择哪个从库实例作为主库。

接下来，我们就先说说如何判断主库的下线状态。

你首先要知道的是，哨兵对主库的下线判断有“主观下线”和“客观下线”两种。那么，为什么会存在两种判断呢？它们的区别和联系是什么呢？

主观下线和客观下线

我先解释下什么是“主观下线”。

哨兵进程会使用PING命令检测它自己和主、从库的网络连接情况，用来判断实例的状态。如果哨兵发现主库或从库对PING命令的响应超时了，那么，哨兵就会先把它标记为“主观下线”。

如果检测的是从库，那么，哨兵简单地把它标记为“主观下线”就行了，因为从库的下线影响一般不太大，集群的对外服务不会间断。

能存在这么一个情况：那就是哨兵误判了，其实主库并没有故障。可是，一旦启动了主从切换，后续的选主和通知操作都会带来额外的计算和通信开销。

为了避免这些不必要的开销，我们要特别注意误判的情况。

首先，我们要知道啥叫误判。很简单，就是主库实际并没有下线，但是哨兵误以为它下线了。误判一般会发生在集群网络压力较大、网络拥塞，或者是主库本身压力较大的情况下。

一旦哨兵判断主库下线了，就会开始选择新主库，并让从库和新主库进行数据同步，这个过程本身就会有开销，例如，哨兵要花时间选出新主库，从库也需要花时间和新主库同步。而在误判的情况下，主库本身根本就不需要进行切换的，所以这个过程的开销是没有价值的。正因为这样，我们需要判断是否有误判，以及减少误判。

那怎么减少误判呢？在日常生活中，当我们要对一些重要的事情做判断的时候，经常会和家人或朋友一起商量一下，然后再做决定。

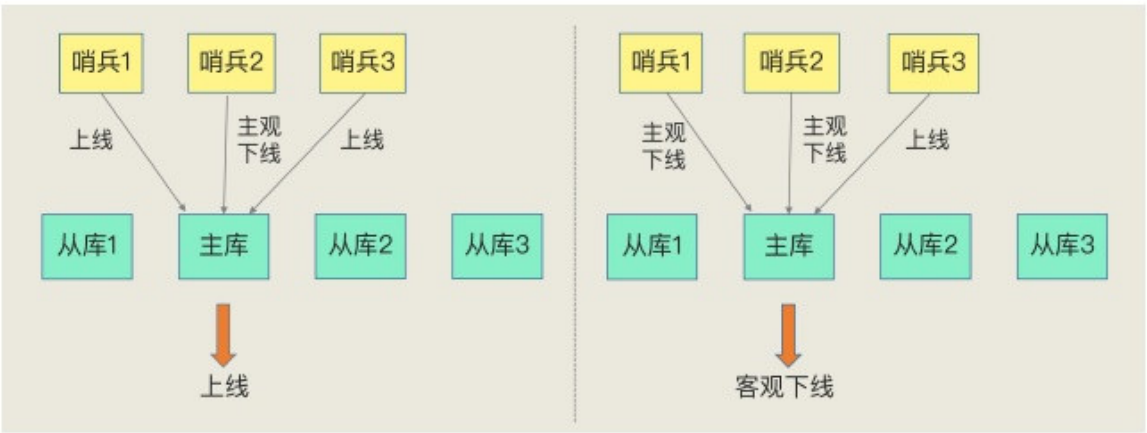
哨兵机制也是类似的，它**通常会采用多实例组成的集群模式进行部署，这也被称为哨兵集群**。引入多个哨兵实例一起来判断，就可以避免单个哨兵因为自身网络状况不好，而误判主库下线的情况。同时，多个哨兵的网络同时不稳定的概率较小，由它们一起做决策，误判率也能降低。

这节课，你只需要先理解哨兵集群在减少误判方面的作用，就行了。至于具体的运行机制，下节课我们再重点学习。

在判断主库是否下线时，不能由一个哨兵说了算，只有大多数的哨兵实例，都判断主库已经“主观下线”了，主库才会被标记为“客观下线”，这个叫法也是表明主库下线成为一个客观事实了。这个判断原则就是：少数服从多数。同时，这会进一步触发哨兵开始主从切换流程。

为了方便你理解，我再画一张图展示一下这里的逻辑。

如下图所示，Redis主从集群有一个主库、三个从库，还有三个哨兵实例。在图片的左边，哨兵2判断主库为“主观下线”，但哨兵1和3却判定主库是上线状态，此时，主库仍然被判断为处于上线状态。在图片的右边，哨兵1和2都判断主库为“主观下线”，此时，即使哨兵3仍然判断主库为上线状态，主库也被标记为“客观下线”了。



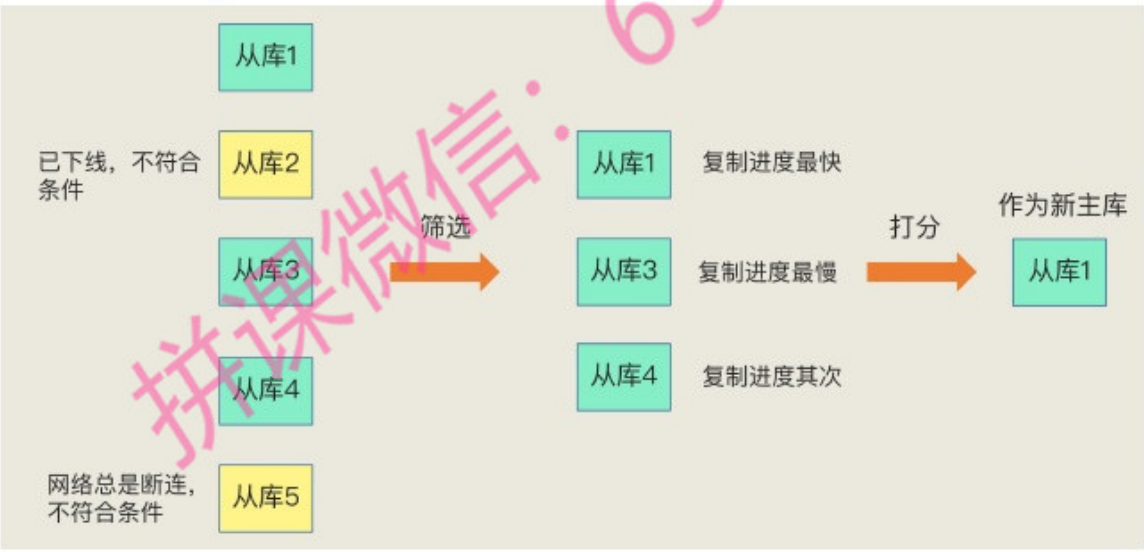
简单来说，“客观下线”的标准就是，当有N个哨兵实例时，最好要有N/2 + 1个实例判断主库为“主观下

线”，才能最终判定主库为“客观下线”。这样一来，就可以减少误判的概率，也能避免误判带来的无谓的主从库切换。（当然，有多少个实例做出“主观下线”的判断才可以，可以由Redis管理员自行设定）。

好了，到这里，你可以看到，借助于多个哨兵实例的共同判断机制，我们就可以更准确地判断出主库是否处于下线状态。如果主库的确下线了，哨兵就要开始下一个决策过程了，即从许多从库中，选出一个从库来做新主库。

如何选定新主库？

一般来说，我把哨兵选择新主库的过程称为“筛选+打分”。简单来说，我们在多个从库中，先按照**一定的筛选条件**，把不符合条件的从库去掉。然后，我们再按照**一定的规则**，给剩下的从库逐个打分，将得分最高的从库选为新主库，如下图所示：



在刚刚的这段话里，需要注意的是两个“一定”，现在，我们要考虑这里的“一定”具体是指什么。

首先来看筛选的条件。

一般情况下，我们肯定要先保证所选的从库仍然在线运行。不过，在选主时从库正常在线，这只能表示从库的现状良好，并不代表它就是最适合做主库的。

设想一下，如果在选主时，一个从库正常运行，我们把它选为新主库开始使用了。可是，很快它的网络出了故障，此时，我们就得重新选主了。这显然不是我们期望的结果。

所以，在选主时，**除了要检查从库的当前在线状态，还要判断它之前的网络连接状态**。如果从库总是和主库断连，而且断连次数超出了一定的阈值，我们就有理由相信，这个从库的网络状况并不是太好，就可以把这个从库筛掉了。

具体怎么判断呢？你使用配置项down-after-milliseconds * 10。其中，down-after-milliseconds是我们认定主从库断连的最大连接超时时间。如果在down-after-milliseconds毫秒内，主从节点都没有通过网络联系上，我们就可以认为主从节点断连了。如果发生断连的次数超过了10次，就说明这个从库的网络状况不好，不适合作为新主库。

接下来就要给剩余的从库打分了。我们可以分别按照三个规则依次进行三轮打分，这三个规则分别是**从库优先级、从库复制进度以及从库ID号**。只要在某一轮中，有从库得分最高，那么它就是主库了，选主过程到此结束。如果没有出现得分最高的从库，那么就继续进行下一轮。

第一轮：优先级最高的从库得分高。

用户可以通过slave-priority配置项，给不同的从库设置不同优先级。比如，你有两个从库，它们的内存大小不一样，你可以手动给内存大的实例设置一个高优先级。在选主时，哨兵会给优先级高的从库打高分，如果有一个从库优先级最高，那么它就是新主库了。如果从库的优先级都一样，那么哨兵开始第二轮打分。

第二轮：和旧主库同步程度最接近的从库得分高。

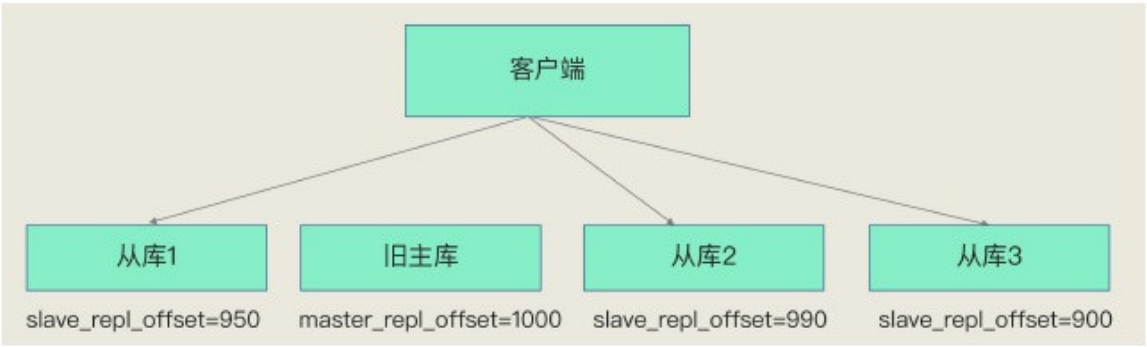
这个规则的依据是，如果选择和旧主库同步最接近的那个从库作为主库，那么，这个新主库上就有最新的数据。

如何判断从库和旧主库间的同步进度呢？

上节课我向你介绍过，主从库同步时有个命令传播的过程。在这个过程中，主库会用master_repl_offset记录当前的最新写操作在repl_backlog_buffer中的位置，而从库会用slave_repl_offset这个值记录当前的复制进度。

此时，我们想要找的从库，它的slave_repl_offset需要最接近master_repl_offset。如果在所有从库中，有从库的slave_repl_offset最接近master_repl_offset，那么它的得分就最高，可以作为新主库。

就像下图所示，旧主库的master_repl_offset是1000，从库1、2和3的slave_repl_offset分别是950、990和900，那么，从库2就应该被选为新主库。



当然，如果有两个从库的slave_repl_offset值大小是一样的（例如，从库1和从库2的slave_repl_offset值都是990），我们就需要给它们进行第三轮打分了。

第三轮：ID号小的从库得分高。

每个实例都会有一个ID，这个ID就类似于这里的从库的编号。目前，Redis在选主库时，有一个默认的规定：**在优先级和复制进度都相同的情况下，ID号最小的从库得分最高，会被选为新主库**。

到这里，新主库就被选出来了，“选主”这个过程就完成了。

我们再回顾下这个流程。首先，哨兵会按照在线状态、网络状态，筛选过滤掉一部分不符合要求的从库，然后，依次按照优先级、复制进度、ID号大小再对剩余的从库进行打分，只要有得分最高的从库出现，就把它选为新主库。

小结

这节课，我们一起学习了哨兵机制，它是实现Redis不间断服务的重要保证。具体来说，主从集群的数据同步，是数据可靠的基础保证；而在主库发生故障时，自动的主从切换是服务不间断的关键支撑。

Redis的哨兵机制自动完成了以下三大功能，从而实现了主从库的自动切换，可以降低Redis集群的运维开销：

- 监控主库运行状态，并判断主库是否客观下线；
- 在主库客观下线后，选取新主库；
- 选出新主库后，通知从库和客户端。

为了降低误判率，在实际应用时，哨兵机制通常采用多实例的方式进行部署，多个哨兵实例通过“少数服从多数”的原则，来判断主库是否客观下线。一般来说，我们可以部署三个哨兵，如果有两个哨兵认定主库“主观下线”，就可以开始切换过程。当然，如果你希望进一步提升判断准确率，也可以再适当增加哨兵个数，比如说使用五个哨兵。

但是，使用多个哨兵实例来降低误判率，其实相当于组成了一个哨兵集群，我们会因此面临着一些新的挑战，例如：

- 哨兵集群中有实例挂了，怎么办，会影响主库状态判断和选主吗？
- 哨兵集群多数实例达成共识，判断出主库“客观下线”后，由哪个实例来执行主从切换呢？

要搞懂这些问题，就不得不提哨兵集群了，下节课，我们来具体聊聊哨兵集群的机制和问题。

每课一问

按照惯例，我给你提个小问题。这节课，我提到，通过哨兵机制，可以实现主从库的自动切换，这是实现服务不间断的关键支撑，同时，我也提到了主从库切换是需要一定时间的。所以，请你考虑下，在这个切换过程中，客户端能否正常地进行请求操作呢？如果想要应用程序不感知服务的中断，还需要哨兵或需要客户端再做些什么吗？

欢迎你在留言区跟我交流讨论，也欢迎你能帮我把今天的内容分享给更多人，帮助他们一起解决问题。我们下节课见。

精选留言：

- Kaito 2020-08-19 10:54:47
  哨兵在操作主从切换的过程中，客户端能否正常地进行请求操作？

  如果客户端使用了读写分离，那么读请求可以在从库上正常执行，不会受到影响。但是由于此时主库已经挂了，而且哨兵还没有选出新的主库，所以在这期间写请求会失败，失败持续的时间 = 哨兵切换主从的时间 + 客户端感知到新主库 的时间。

如果不想让业务感知到异常，客户端只能把写失败的请求先缓存起来或写入消息队列中间件中，等哨兵切换完主从后，再把这些写请求发给新的主库，但这种场景只适合对写入请求返回值不敏感的业务，而且还需要业务层做适配，另外主从切换时间过长，也会导致客户端或消息队列中间件缓存写请求过多，切换完成之后重放这些请求的时间变长。

哨兵检测主库多久没有响应就提升从库为新的主库，这个时间是可以配置的（down-after-milliseconds参数）。配置的时间越短，哨兵越敏感，哨兵集群认为主库在短时间内连不上就会发起主从切换，这种配置很可能因为网络拥塞但主库正常而发生不必要的切换，当然，当主库真正故障时，因为切换得及时，对业务的影响最小。如果配置的时间比较长，哨兵越保守，这种情况可以减少哨兵误判的概率，但是主库故障发生时，业务写失败的时间也会比较久，缓存写请求数据量越多。

应用程序不感知服务的中断，还需要哨兵和客户端做些什么？当哨兵完成主从切换后，客户端需要及时感知到主库发生了变更，然后把缓存的写请求写入到新库中，保证后续写请求不会再受到影响，具体做法如下：

哨兵提升一个从库为新主库后，哨兵会把新主库的地址写入自己实例的pubsub（switch-master）中。客户端需要订阅这个pubsub，当这个pubsub有数据时，客户端就能感知到主库发生变更，同时可以拿到最新的主库地址，然后把写请求写到这个新主库即可，这种机制属于哨兵主动通知客户端。

如果客户端因为某些原因错过了哨兵的通知，或者哨兵通知后客户端处理失败了，安全起见，客户端也需要支持主动去获取最新主从的地址进行访问。

所以，客户端需要访问主从库时，不能直接写死主从库的地址了，而是需要从哨兵集群中获取最新的地址（sentinel get-master-addr-by-name命令），这样当实例异常时，哨兵切换后或者客户端断开重连，都可以从哨兵集群中拿到最新的实例地址。

一般Redis的SDK都提供了通过哨兵拿到实例地址，再访问实例的方式，我们直接使用即可，不需要自己实现这些逻辑。当然，对于只有主从实例的情况，客户端需要和哨兵配合使用，而在分片集群模式下，这些逻辑都可以做在proxy层，这样客户端也不需要关心这些逻辑了，Codis就是这么做的。

另外再简单回答下哨兵相关的问题：

1、哨兵集群中有实例挂了，怎么办，会影响主库状态判断和选主吗？

这个属于分布式系统领域的问题了，指的是在分布式系统中，如果存在故障节点，整个集群是否还可以提供服务？而且提供的服务是正确的？

这是一个分布式系统容错问题，这方面最著名的就是分布式领域中的“拜占庭将军”问题了，“拜占庭将军问题”不仅解决了容错问题，还可以解决错误节点的问题，虽然比较复杂，但还是值得研究的，有兴趣的同学可以去了解下。

简单说结论：存在故障节点时，只要集群中大多数节点状态正常，集群依旧可以对外提供服务。具体推导过程细节很多，大家去查前面的资料了解就好。

2、哨兵集群多数实例达成共识，判断出主库“客观下线”后，由哪个实例来执行主从切换呢？

哨兵集群判断出主库“主观下线”后，会选出一个“哨兵领导者”，之后整个过程由它来完成主从切换。

但是如何选出“哨兵领导者”？这个问题也是一个分布式系统中的问题，就是我们经常听说的共识算法，

指的是集群中多个节点如何就一个问题达成共识。共识算法有很多种，例如Paxos、Raft，这里哨兵集群采用的类似于Raft的共识算法。

简单来说就是每个哨兵设置一个随机超时时间，超时后每个哨兵会请求其他哨兵为自己投票，其他哨兵节点对收到的第一个请求进行投票确认，一轮投票下来后，首先达到多数选票的哨兵节点成为“哨兵领导者”，如果没有达到多数选票的哨兵节点，那么会重新选举，直到能够成功选出“哨兵领导者”。 [17赞]

- Darren 2020-08-19 10:39:04
肯定会中断的，但是这么让客户端无感知，说说可能不成熟的想法，请老师和大家指点：
1、如果是读请求，可以直接读取从库，客户端无影响；
2、如果是写请求，可以先把命令缓存到哨兵中（比如说哨兵内部维护一个队列等），等选主成功后，在新的主库进行执行即可。 [3赞]

- Monday 2020-08-19 10:09:08
1、master_repl_offset是存储在主库的，但主库已经挂了，怎么获取的这个值？
可否这样理解，master_repl_offset如事物id一样单调递增，这样的话，就只要不叫从库的slave_repl_offset就行。
至于master_repl_offset真实位置可以对master_repl_offset取模就行。 [3赞]

- 吕 2020-08-19 08:00:39
关于第二步，根据master_repl_offset和slave_repl_offset来比较，但此时master已经挂掉了，哨兵如何知道master_repl_offset的，难道哨兵也会存一份主的master_repl_offset？根据之前的学习，salve是不存储master_repl_offset的 [1赞]

- 三木子 2020-08-19 18:36:54
为什么“旧主库同步程度最接近的从库得分高为第一个优先条件“呢？这样可以保证数据最接近原主库

- Oracleblog 2020-08-19 17:17:56
主从切换选出新的主后，新的从库同步是需要做一次全量同步吗？

- 杨逸林 2020-08-19 16:52:56
老师讲得应该是让大家都能懂，了解个大概的。
有些细节没说，我查了下，还翻了下书《Redis 设计与实现》第 16 章。
启动哨兵需要配置 sentinel.conf，里面有些重要的配置，
SENTINEL MONITOR <name> <ip> <port> <quorum>
Sentine监听的maste地址，第一个参数是给master起的名字，第二个参数为master IP，第三个为master端口，第四个为当该master挂了的时候，若想将该master判为失效，
更多的东西，我就不做复读机了，https://www.cnblogs.com/kevingrace/p/9004460.html 可以看这个补全有关哨兵的内容，还有书上的第 16 章。

- 不负青春不负己ᵕ̈ 2020-08-19 10:59:08
1 sentinel 集群 一般建议是3个节点 还是，多个节点， 怎么保证 sentinal 集群的高可用， 以及集群节点过多 ，会不会 导致选举时间过长， sentinel 选举 类似于 变体raft 协议
2 能不能创建一个微信 或者QQ 群， 一些简单的问题 可以互相交流，

- 那时刻 2020-08-19 10:37:07
在主从切换的时候，由哨兵把新请求倒流到新的主节点？切换完成之后，需要客户端切换到新的主节点操作

- MClink 2020-08-19 10:09:42
我是这么香的，可以基于从库的读快照+写缓冲区的命令，类似MySQL 的change_buff 机制。这样就可以短暂恢复业务，然后切换成功后，再把对应的写缓存命令往主库发也顺便维持这个机制，主库执行完命令并且和缓存机制的数据达到一致后就可以正常无缝切换了

- kyon 2020-08-19 10:08:27
请教下，切换新主时，筛选从节点的过程中，主从节点之间是否段开连接及断开时间 down-after-milliseconds，也是由哨兵判断的吗？我理解哨兵判断的是，哨兵和各个节点之间是否联通。

- 游弋云端 2020-08-19 09:20:51
写业务会中断，但是客户端可以做缓存，等待主恢复或者重新选主后，再下发请求，同时缓冲区大小有限，如果长时间没有产生新主，缓冲区满了后还是会返回失败或者超时。

- test 2020-08-19 08:52:17
主从切换会造成业务中断

- 徐鹏 2020-08-19 08:43:46
有两个问题想请教哈
1.每一个哨兵实例都有整个redis集群的信息，会和每一个redis实例通信吗？
2.在选主过程中，比较从库的salve_repl_offset，是把每个从库salve_repl_offset相互比较还是和master_repl_offset比较？原来的主库不是已经挂了，master_repl_offset 是如何获取到的呢？

- zhou 2020-08-19 08:40:14
判断主库存活，如果一个哨兵的误判率为百分之一，那么三个哨兵的误判率为百万分之一，大大降低了误判的可能。

- tt 2020-08-19 08:18:30
老师，有两个问题。

1、当主节点宕机后，哨兵是从哪里知道master_repl_offset的值的，是之前主节点在命令传播的过程中传递给哨兵的么？如果是的话，是使用TCP单播逐个通知的还是利用组播机制通知的？无论如何，主节点都要维护集群中哨兵的信息，这个信息是由谁维护和更新的？

2、客户端是在配置信息里配置的是哨兵的链接信息，然后通过哨兵知道当前主节点的地址的么？如果是这样，那客户端发现主节点没有响应时，会再查询哨兵么？

那如果哨兵充当了数据节点的元信息的保存这个角色后，客户端如何配置哨兵的连接信息呢？因为哨兵集群貌似是一个对等的网络，是连接到任意一个coordinator上就可以了么？

- Geek_c37e49 2020-08-19 00:24:53
主从切换的时候应该是没办法响应写请求的，不过可以把请求缓存记录下来
读请求应该是可以服务的吧