Redis lock

当多个实例节点出现资源竞争时，通常需要使用分布式锁来解决。 目前常用的分布式锁实现方案主要有三种

1. 基于数据库（唯一索引）
2. 基于缓存(Redis、memcached、tair)
3. 基于Zookeeper

此文记录最近学习的几种Redis lock的实现方式的比较。主要内容来源于[Redis.io](https://redis.io/topics/distlock)中对Red lock的介绍

**单个Redis节点**

1. **常见的一种的方式- setnx**

**加锁方式**：SET resource\_name my\_random\_value NX PX 30000

          仅当竞争资源key不存在时才能成功获取锁(NX 选项)，并在30000ms后超时(PX 选项)。为了避免锁被其他client错误释放，my\_random\_value必须能唯一标识此次acquire lock的请求和发出请求的client。

**释放锁方式：**通过lua脚本释放，当且仅当key存在且对应的value是当前client期望的值时才能被释放，这是为了避免锁被其他client错误释放。例如，当一个client A成功获取到锁后，由于某些原因被阻塞，当时间超过key expire time时，锁已被超时释放并可能被另一个客户端client B成功获取，此时client A从阻塞中恢复并完成业务逻辑后进行锁释放，则会将client B的锁错误释放。 解锁lua脚本如下：

if redis.call("get",KEYS[1]) == ARGV[1] then

return redis.call("del",KEYS[1])

else

return 0

end

**不足之处：**

1. 由于使用单个redis节点，可能造成系统中的单点failover。
2. 若增加slave节点，由于redis的同步策略是异步的，会造成race condition。例如， client A从redis master中获取到锁；在redis master将该锁的key同步到slave之前，master节点crash；slave节点被提升为当前master节点；client B此时可获取到此锁，显然违反了锁的定义。(??? redis的replicate log策略， 当client A能够从当前redis master中获取到锁时， 是否意味着redis已经被同步到slave了。 联想到redis 配置的aof的同步策略如每秒同步 -》 似乎是会造成此条讨论中的问题。)

**多个Redis节点**

1. **Redlock Algorithm**

一、什么是RedLock？

    redis.io官网上使用redis实现的一种distributed lock。

二、RedLock的实现简述

    假设有5个相互独立的redis master节点， 为了获取资源锁，client应当做以下操作。

1. client获取当前时间戳(ms)
2. 按顺序在5个节点上获取锁(使用常见的redis lua加锁脚本，在所有的节点都使用同样的key与unique random value)。
3. client计算获取锁的时间： currentTime减去step1的时间戳。 当且仅当在超过半数的节点上获取到锁并且获取锁的时间小于锁的过期时长时，才认为成功获取到了该锁。
4. 如果成功获取到锁， 锁的剩余有效时长为锁的初始超期时长减去step3的获取锁花费的时间。
5. 如果竞争锁失败，释放所有已经获取到的锁。

三、RedLock是否会失效？

一、假设系统有5个redis节点a,b,c,d,e与两个客户端client1, client2。如果其中某一个redis节点的系统时钟异常跳前了，会发生什么情况？

1. client 1 在a,b,c上获得锁，并且由于系统网络原因，client1无法触达d,e。
2. c节点的系统时钟跳前，导致c节点上的锁超时(失效)。
3. client2 在节点c,d,e上获得锁，并且由于网络原因， client2无法触达a,b。
4. 此时， client1与client2同时持有锁， 违反了锁的定义。

二、一个更为实际一点的例子：由客户端应用本身短期无法正常服务导致失效

1. client1 在a,b,c,d,e获取到锁。
2. 给client1的返回值还在网络传输中时，client1发生stop-the-world gc。
3. 锁在所有redis节点超时失效
4. client2 在所有节点获取到锁
5. client1完成gc并接受redis servers的返回，认为成功获取到锁
6. 此时，client1与client2都认为自己持有该锁， 违反了锁的定义

与情景二同理， 长时间的网络延迟也可以造成上述结果。还有其他情形，不一一列举， 有兴趣可以去原文查看 ，google “martin kleppmann how to do distributed locking”。

四、使用RedLock的前提

1. bounded network delay:保障网络请求可以在固定时限内到达
2. bound process pauses:对应例子中的stop-the-world gc，可类比其他情形。
3. bound clock error:统一系统时钟服务，并且NTP(Network Time Protocol) server不失效。

五、个人总结

    反正我信了， 我不用RedLock，有需求使用其他替代分布式锁。