分布式锁的实现

基于 Redis

- 基本原理
  - 简易的 Redis 分布式锁通常都是基于 SETNX + Expire Time 所实现的
  - **SET key value [EX seconds] [NX|XX]**
    - NX 表示 key 不存在时才设置 key，XX 表示当 key 存在时才设置 key，相当于更新操作
- 基本流程

```
SET key value 10 NX;  // 获取分布式锁
// 业务逻辑
DEL key;              // 释放锁
```

- Question
  - 为什么需要使用 NX 标志位?
    - 保证当分布式锁不存在时，只有一个 client 能写入 key 成功，并获取到分布式锁
    - ① 这是分布式锁最为重要的一点，即互斥性。同一个时间只允许一个 client 持有锁，不能有多个 client 持有
  - 为什么需要设置过期时间?
    - 因为如果应用程序 Crash 或者是 Redis 出现了网络分区后无法和该 Redis 进行通信，那么这个锁将一直得不到释放，其它 client 将永远无法获取分布式锁，导致死锁
    - ② 这是分布式锁的第二个核心要素，活性。在实现分布式锁的时候一定要考虑所依赖的其它网络组件出现网络分区的情况，并且能够主动的释放掉分布式锁
  - SET key value 和设置过期时间可以使用 2 个命令吗?
    - 不可以，因为 完全有可能出现 SET key value 成功后，Redis 出现网络分区，使得我们无法为其设置过期时间，最终出现死锁
    - ③ 这是分布式锁的第三核心要素，原子性。我们需要原子性的申请分布式锁以及设置锁的过期时间
- 茅台超卖事件
  - https://juejin.cn/post/6854573212831842311
  - 这位老哥就是使用上面的方式来实现减库存的操作，但是最终还是导致了超卖

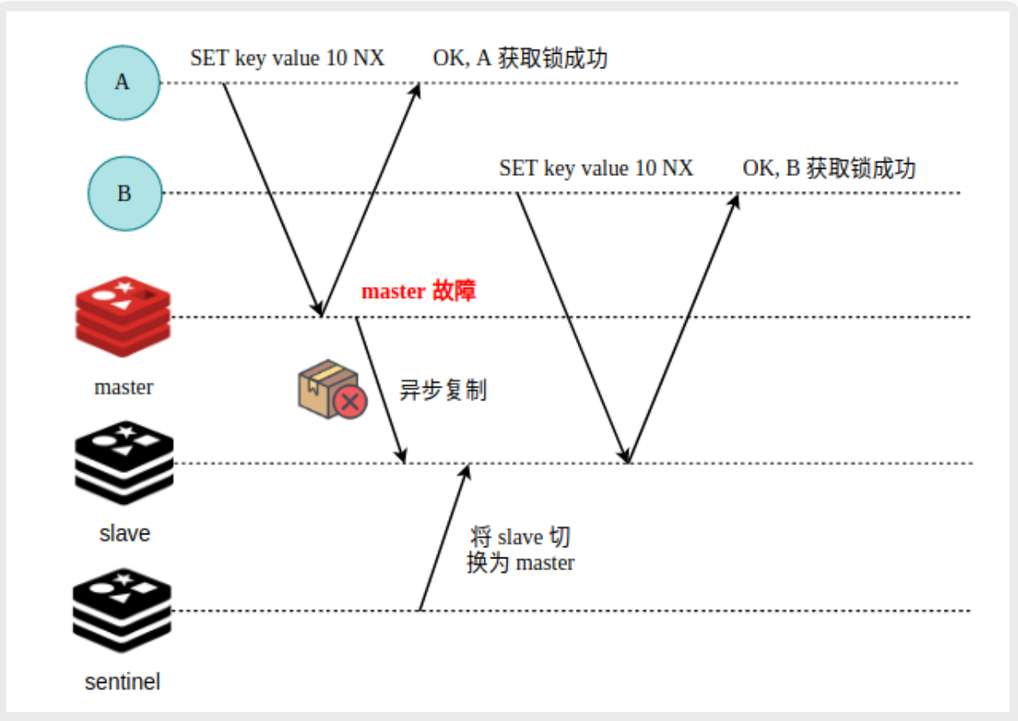
```
SET key value 10 NX;  // 获取分布式锁
// 业务逻辑 1，校验用户身份
// 业务逻辑 2，查询并判断库存是否大于 0
// 业务逻辑 3，若库存 > 0，生成茅台订单
DEL key;              // 释放锁
```

  - 其原因就在于秒杀活动开始以后，加锁逻辑中的校验用户身份因为高负载而出现了长时间的阻塞（30秒），然后锁在 10 秒后就过期了，导致其它 client 能够获取到锁。并且更为严重的是，当阻塞请求执行完毕以后，又会把其它 client 所持有的锁释放掉了

因此，使用 SET key value 10 NX 所实现的分布式锁有一个非常大的问题: client A 可以释放掉 client B 所持有的锁，并且这个情况完全有可能出现

- 这里再次鞭尸 Golang

- Redis 分布式锁的问题
  - 高可用情况下的主备不一致
    - 在绝大多数情况下，Master 和 Slave 都是使用异步复制的，那么这中间就会有一小段时间出现主备不一致的情况
    - 如果这个时候 Master 节点挂了，并且 Sentinel 执行了切主操作，那么此时在 Master 节点设置的分布式锁将"消失"，其它 client 也能够获取到锁



  - 高可用情况下发生脑裂
    - 另外一个问题就是当 Redis 集群出现网络分区导致脑裂时，会有两个 master 节点，那么也会出现多个 client 同时获取到分布式锁的情况
  - RedLock
    - Redis 作者为了解决 SET key value [EX] 10 [NX] 命令实现分布式锁不安全的问题，提出了RedLock 算法。它是基于多个独立的 master 节点的一种实现，有点儿类似于 Quorum NWR，client 依次向各个节点申请锁，若能从多数个节点中申请锁成功并满足一些条件限制，那么 client 就能获取锁成功
    - 但是 RedLock 对 Redis 集群的数量有要求，通常为 3 个或者是 5 个，并且依赖系统时间，当时钟产生跳跃时，就会出现安全性问题

基于 etcd

- etcd 基于 Raft 共识算法所实现，专门用于解决分布式系统中数据一致性问题，因此不会出现 Leader 节点和 Follower 节点数据不一致的问题，并且，不会出现脑裂问题。但是，强一致性的代价就是 etcd 的吞吐量远远赶不上 Redis 的吞吐量
- etcd 底层使用 boltdb 进行数据存储，支持事务。并且使用基于 B-Tree 的 TreeIndex 实现了MVCC。同时，使用 Lease 来优化 TTL 的运行效率，也就是 etcd 中 key 支持设置过期时间
- 这里我们不再对分布式锁的具体实现进行梳理，其原理和 Redis 近乎相同，当 key 不存在时创建表示获取到锁，并原子性地设置过期时间
- 与 RedLock 的对比
  - 基础设施问题
    - 如果存在 k8s 集群，那么必然存在 3 个或者 5 个 etcd 节点，我们可以直接拿来用。相反地，在大多数情况下，并不一定有 5 个 Redis Master 节点
  - 性能问题
    - RedLock 和 etcd 一样，都需要多数节点通过后才能真正的获取分布式锁，但 Redis 由于是纯内存操作，所以性能要高于 etcd