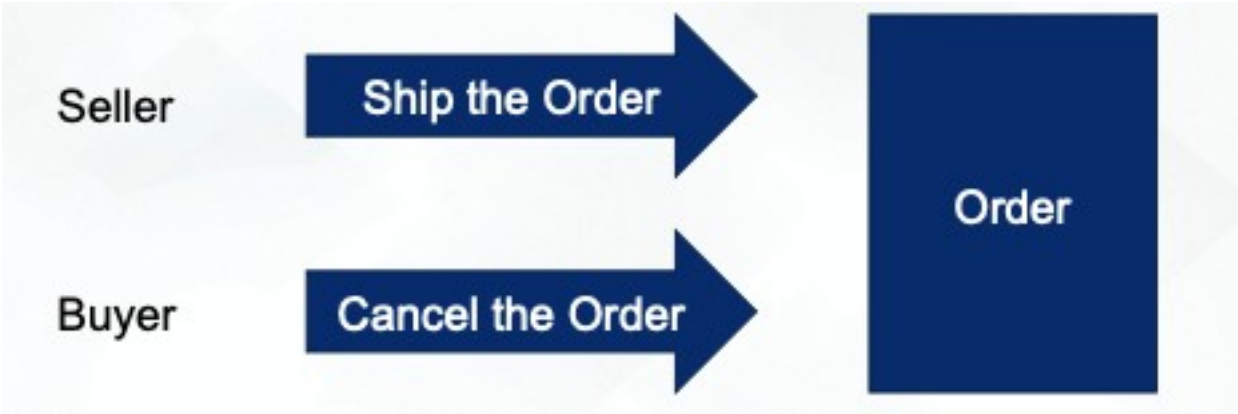
# 分布式锁介绍



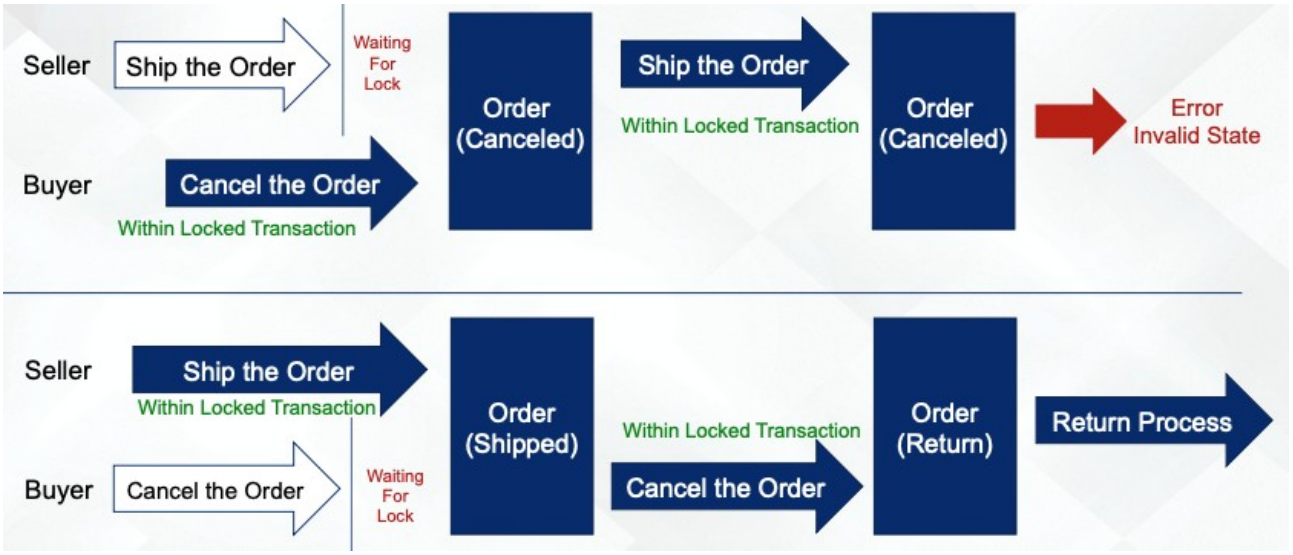
## 背景问题

### 一致性

一个实际的案例，商家要配送订单，顾客要取消订单。如何保证订单状态的一致性？




如果有了锁，我们就能保证订单状态不会错乱。




# 解决方案

## 数据库

### 共享锁

语法：

**SELECT ... LOCK IN SHARE MODE**

特点：

- 允许其它事务也增加共享锁读取
- 不允许其它事物增加排他锁(for update)
- 当事务同时增加共享锁时候，事务的更新必须等待先执行的事务 commit 后才行，如果同时并发太大可能很容易造成死锁
- 共享锁，事务都加，都能读。修改是惟一的，必须等待前一个事务 commit

### 排他锁

语法：

**SELECT ... FOR UPDATE**

特点：

- 事务之间不允许其它排他锁或共享锁读取，修改更不可能
- 一次只能有一个排他锁执行 commit 之后，其它事务才可执行
- 不允许其它事务增加共享或排他锁读取。修改是惟一的，必须等待前一个事务 commit

### 乐观锁

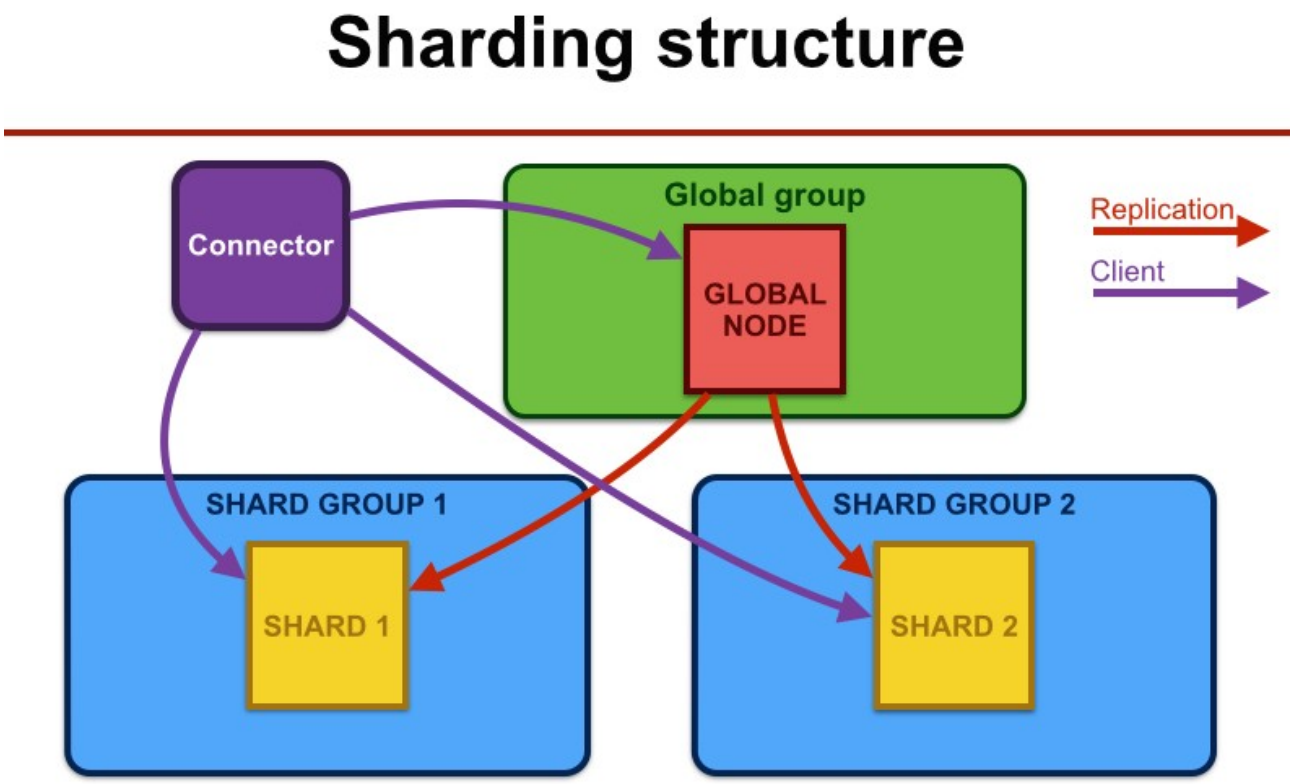
语法：

1. **SELECT id, data, version FROM table WHERE id = $ID**
2. **{ Business Code }**

3. **UPDATE table SET data = $data, version = version + 1 WHERE id = $ID AND version=$VER**
4. **if {UPDATE} Failed, go to step 1 else Done!**

# 数据分片

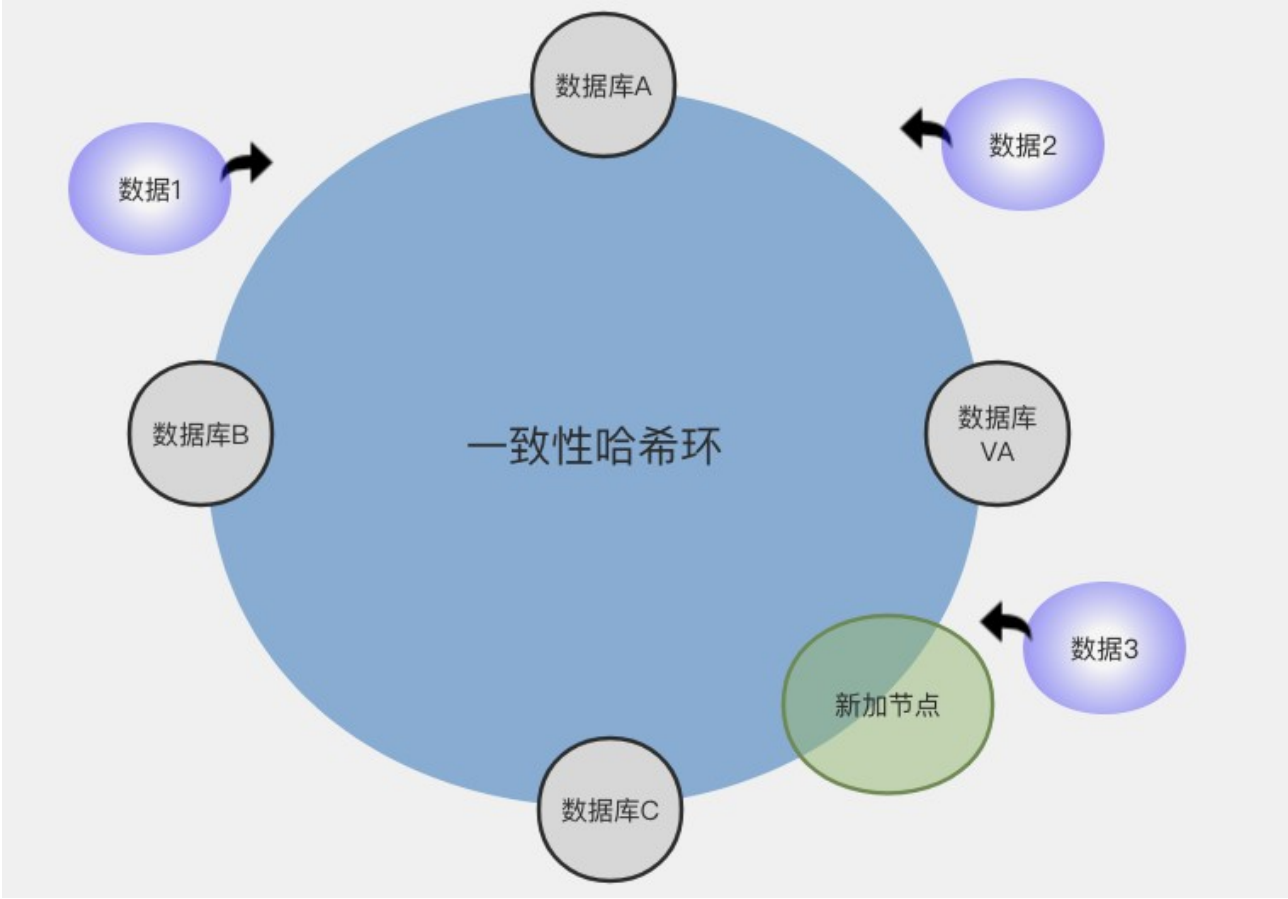
同一个数据的请求转发到同一个机器上
单台机器一致性容易保证




一致性 **hash** https://dgryski.medium.com/consistent-hashing-algorithmic-tradeoffs-ef6b8e2fcae8

**gossip** 做元数据同步 https://cloud.tencent.com/developer/article/1662426

参考 ：**Uber Ringpop** **https://eng.uber.com/ringpop-open-source-nodejs-library/**




**问题**

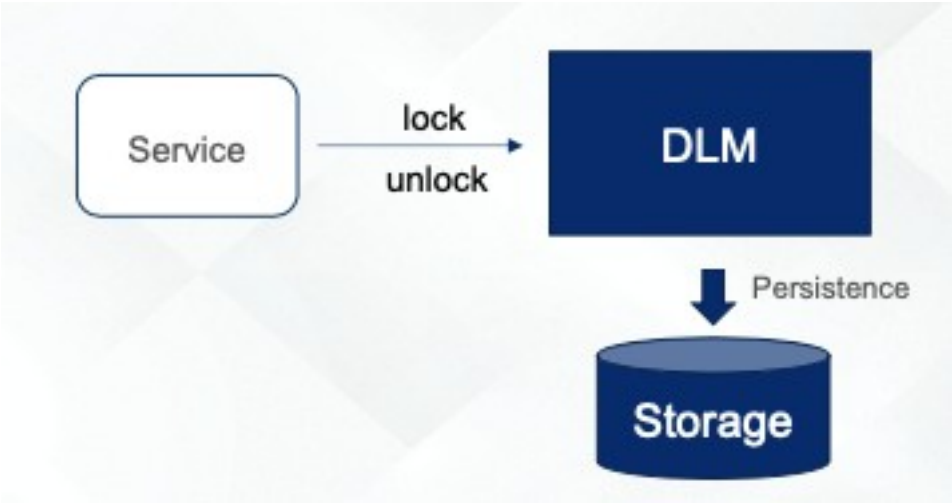
数据热点问题，负载不均衡

操作多条数据问题，节点之间一致性不好解决

数据节点失败，数据分片会导致更多的一致性问题，R+W > N ==> CAP 定理

# 分布式锁服务

参考文档：https://en.wikipedia.org/wiki/Distributed_lock_manager



| Mode | Requesting Process | Other Processes |
|---|---|---|
| Null (NL) | No access | Read or write access |
| Concurrent Read (CR) | Read access only | Read or write access |
| Concurrent Write (CW) | Read or write access | Read or write access |
| Protected Read (PR) | Read access only | Read access only |
| Protected Write (PW) | Read or write access | Read access only |
| Exclusive (EX) | Read or write access | No access |

| Mode | NL | CR | CW | PR | PW | EX |
|---|---|---|---|---|---|---|
| NL | Yes | Yes | Yes | Yes | Yes | Yes |
| CR | Yes | Yes | Yes | Yes | Yes | No |
| CW | Yes | Yes | Yes | No | No | No |
| PR | Yes | Yes | No | Yes | No | No |
| PW | Yes | Yes | No | No | No | No |
| EX | Yes | No | No | No | No | No |

- 提供一个 带租约的锁 协议
- 调用方服务 无状态
- 锁服务 **HA** ，无单点故障，数据可以持久化
- 死锁检查，支持锁过期，心跳机制

# 分布式锁实现

## ⚙ RedLock

文档：https://redis.io/topics/distlock

示例代码：https://hazelcast.com/blog/long-live-distributed-locks/

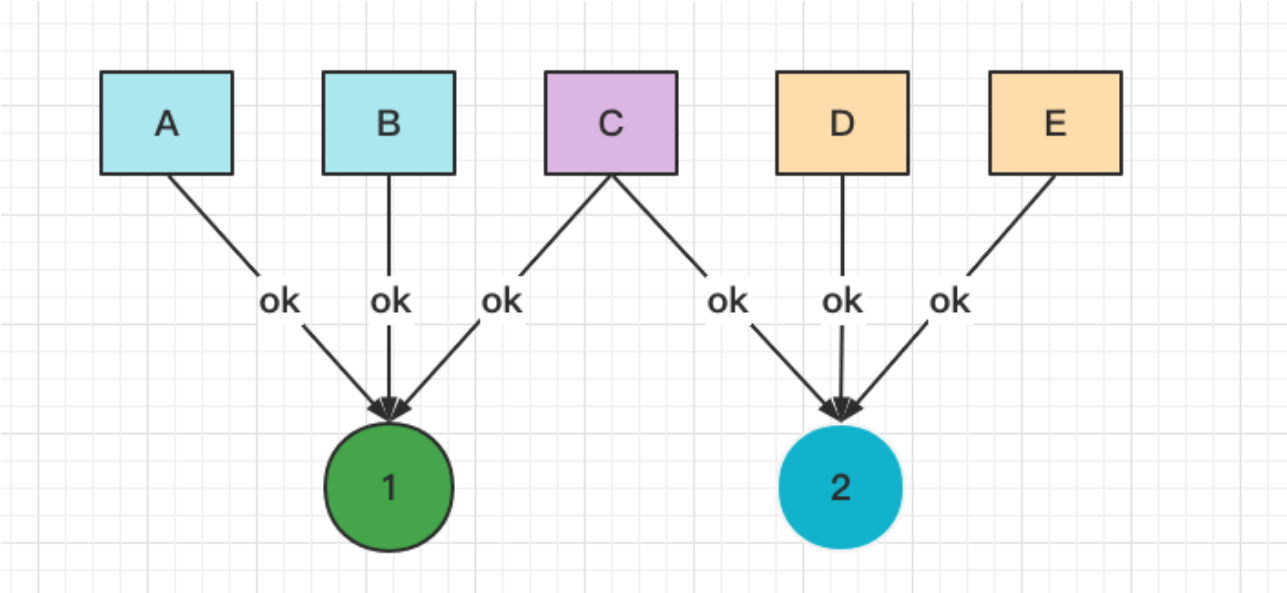
算法：

```
//加锁
SET resource_name my_random_value NX PX 30000


//解锁
if redis.call("get",KEYS[1]) == ARGV[1] then
    return redis.call("del",KEYS[1])
else
    return 0
end
```

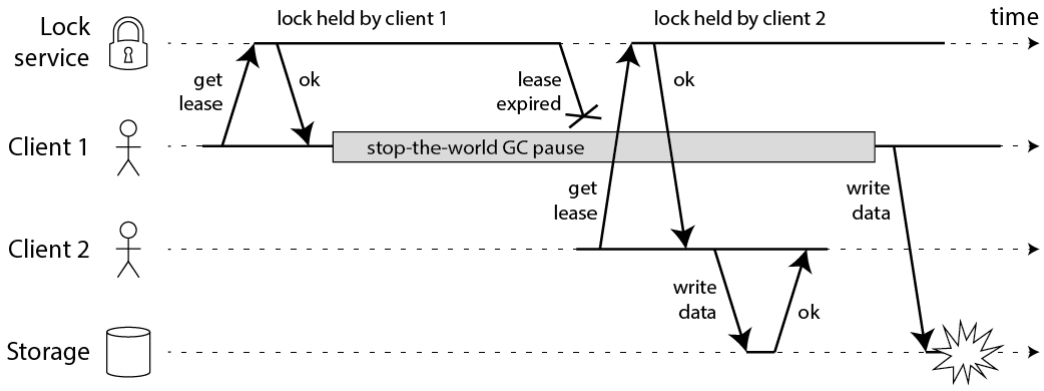
争议：https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html

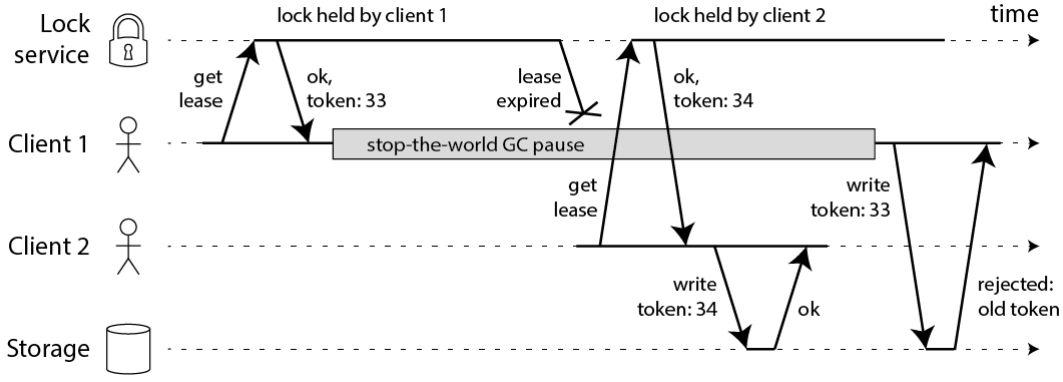
1. 时序问题




- 节点 C 时间跳了，导致锁过期
- 客户端 1 响应 节点 C 的响应，锁过期的时候才有响应 ， 比如 GC 或者 其他原因
- 客户端 1 与 节点 C 有 非常长的网络延迟

## 2. 锁冲突问题




redis 作者的解释：http://antirez.com/news/101

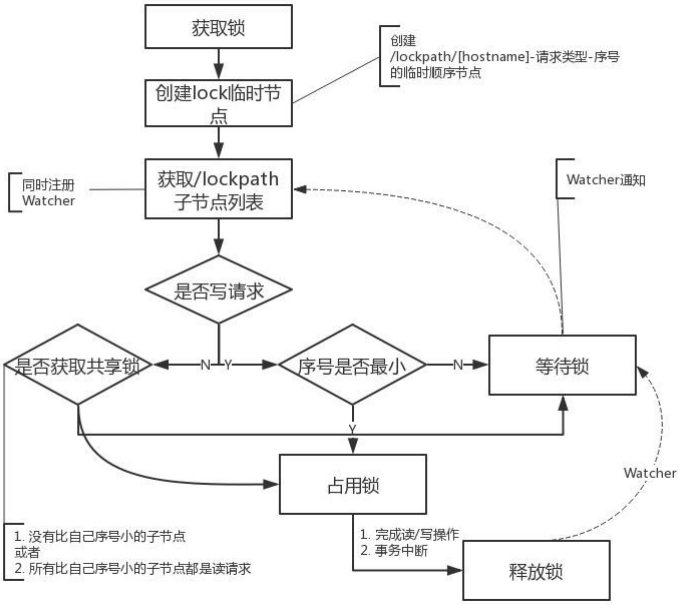
1. 使用 Token 是可以的 （Fencing），但是那样也可以不使用分布式锁，类似数据库乐观锁
2. 时间不一致 是一个大问题，需要运维人员保证机器时间一致

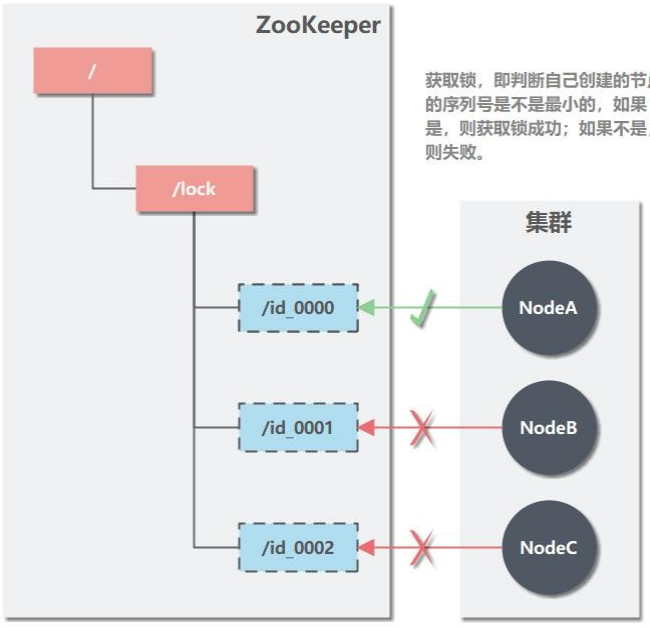
3.网络延时和客户端 Hang ，问题不大，可以正常 Work

## ⚙ Zookeeper Lock

文档：https://zookeeper.apache.org/doc/r3.1.2/recipes.html

实现机制：




## 🚓 Etcd Lock

文档：https://etcd.io/docs/v3.5/dev-guide/api_concurrency_reference_v3/

示例代码：https://medium.com/@felipedutratine/distributed-lock-with-etcd-in-go-d21e7df145bc

简介： https://juejin.cn/post/6844904162791014407

- 包含 Master & Client SDK

- 高可用集群，一般 5 个节点
- 可靠：采用 raft 算法，实现分布式系统数据的可用性和一致性
- 快速：根据官方提供的 benchmark 数据，单实例支持每秒 2k+ 读操作

# 总结思考

1. 并发的事务操作需要做一些同步操作
    0. 数据库锁可以满足一些需求，乐观锁性能会更好
    a. 数据分片不能解决所有的问题
2. 分布式锁服务需要以下特性
    0. 高可用
        0. Data Replicas – 一致性协议 Paxos，Raft，Zab
        i. Master Failover – 选主
    a. 死锁检查
        0. 心跳检测
        i. 过期机制