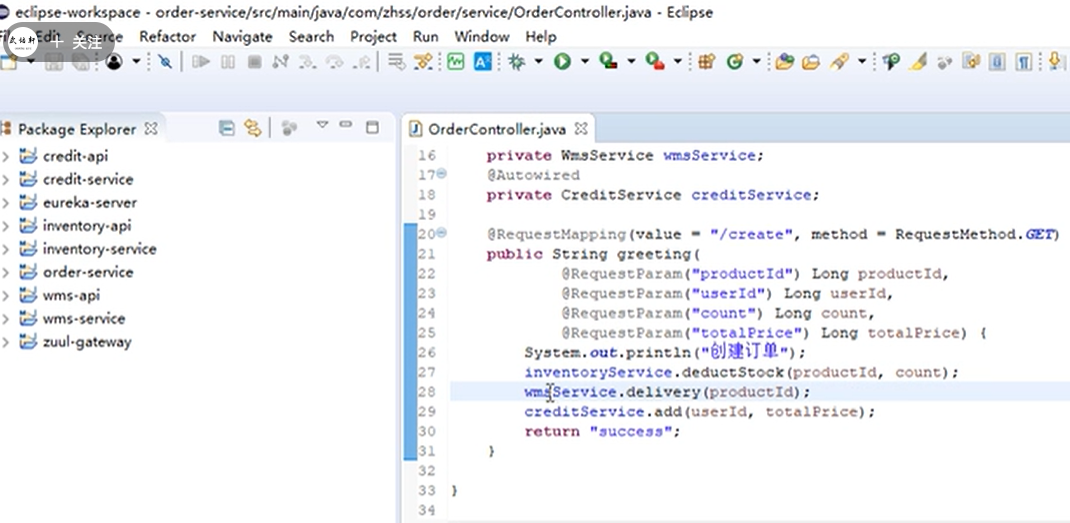
# 分布式事务

## 场景

**订单系统**

* 库存服务：扣减库存
* 积分系统：增加积分
* 物流系统：通知发货

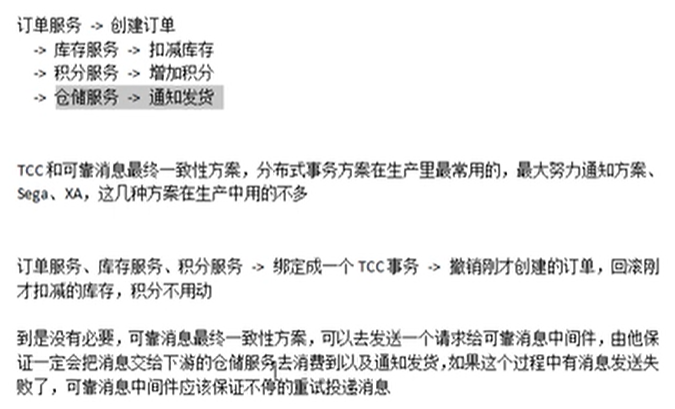


每个系统各自独立的数据库，不管是单机多库还是分布分表

失败场景：

订单服务-》已创建订单；库存服务-》已扣减库存；积分服务-》增加积分失败

需要增加回滚逻辑，需考虑中间步骤回滚失败场景



## 原理

### BASE

解决分布式事务的理论：最终一致性。

## 方案：

### XA

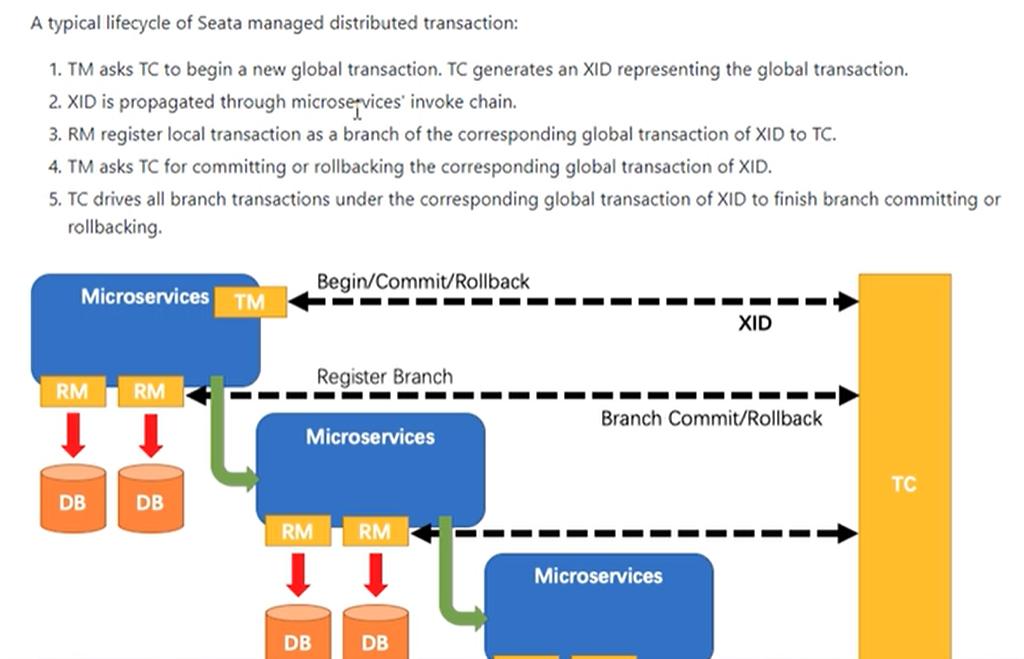
### TCC（常用）

三个阶段：Try-Confirm-Cancel

如下来自Seata的github截图：

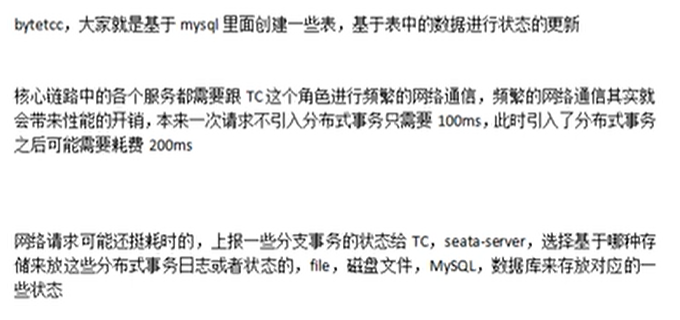
首先申请一个全局的XID，每个子步骤都去注册到TC（事务协调者）。

如果有一个步骤失败，会通知TC。TC会触发全局的回滚。



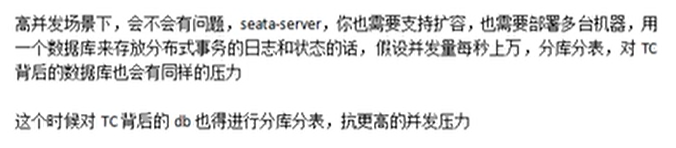
**TCC不足：**

**1、引入分布式事务后会带来的网络开销和性能问题**



**2、高并发场景下的扩容**

**可能也需要解决TC的分库分表**

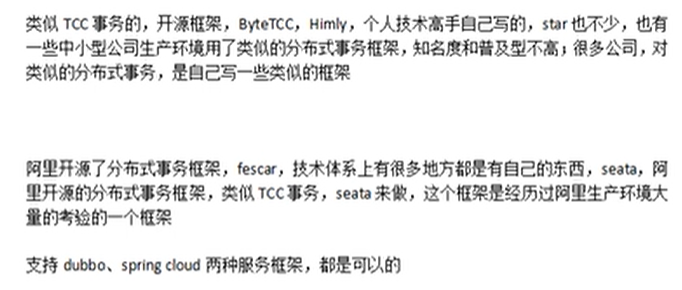


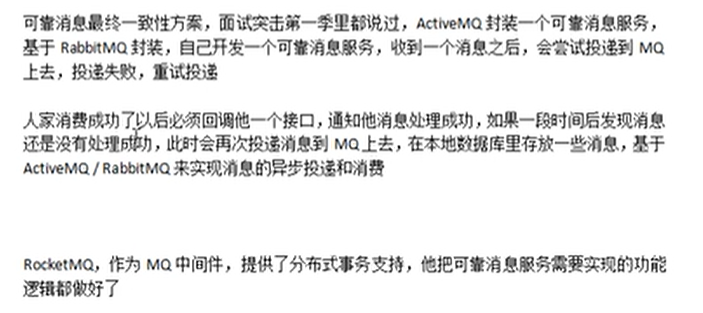
### 可靠消息最终一致性方案（常用）

### 最大努力通知方案

### Sega

## 技术选型



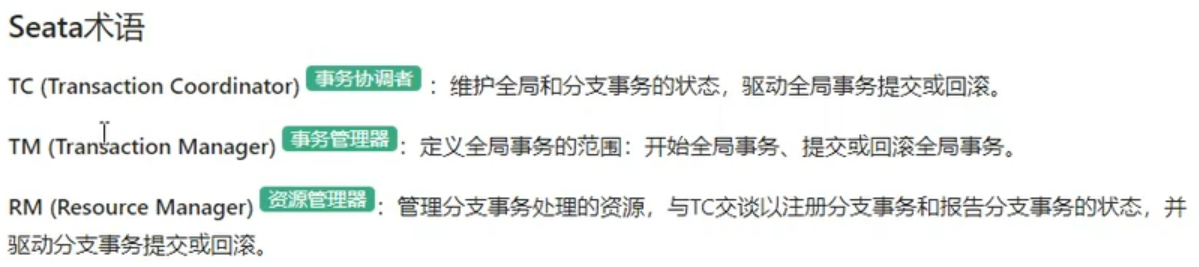


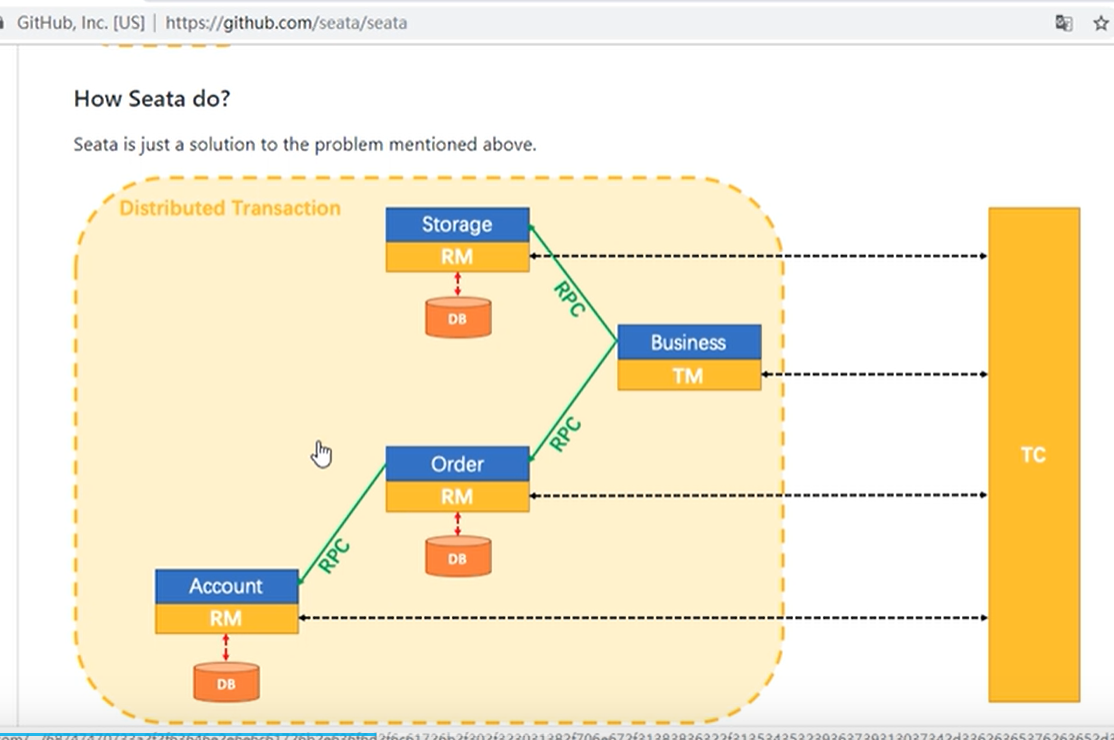
### 开源Seata

seata有四种事务模式：AT、TCC、Sage、XA，其中AT模式是最常用的模式

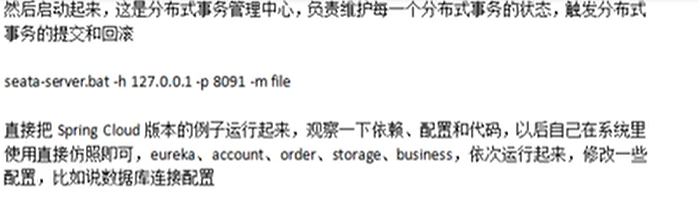
参考：github 上seata-samples

**几种角色：**





安装一个专门的seata-server（对应上图TC）



**不足：参考TCC目录下总结（比如性能损失、高并发场景处理）**

### 开源bytetcc

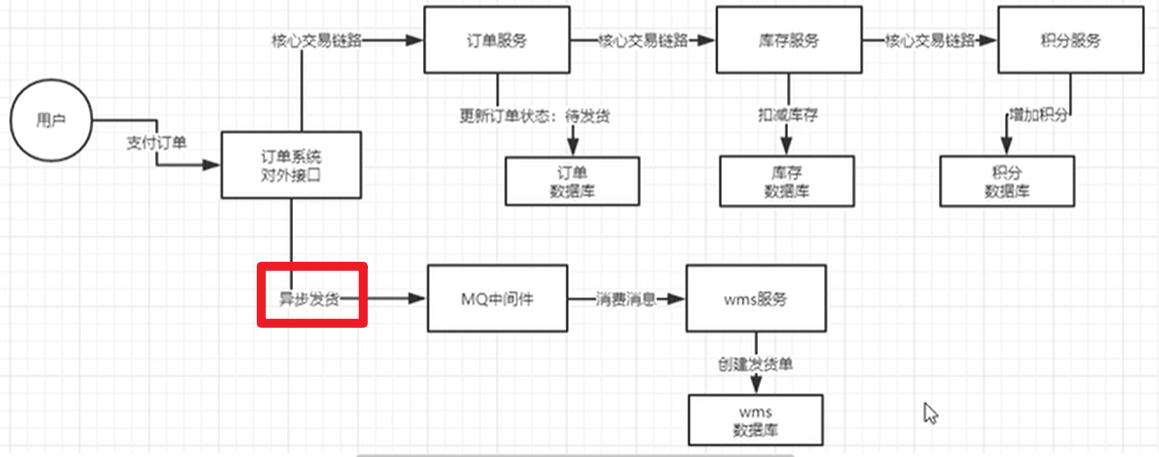
用的少，需要单独实现try、confirm、cancel的方法，供开源件来调用

<https://github.com/liuyangming/ByteTCC>

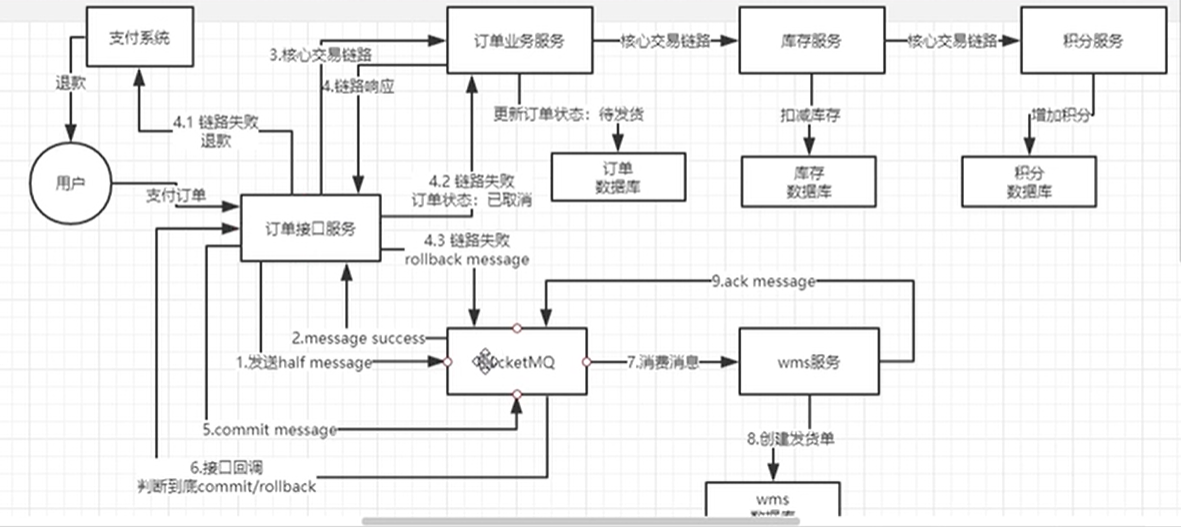
## 基于RocketMQ的可靠消息最终一致性方案

**核心链路走TCC**

**针对发货通过异步方案实现，为了提升并发，没有设置成REST接口**

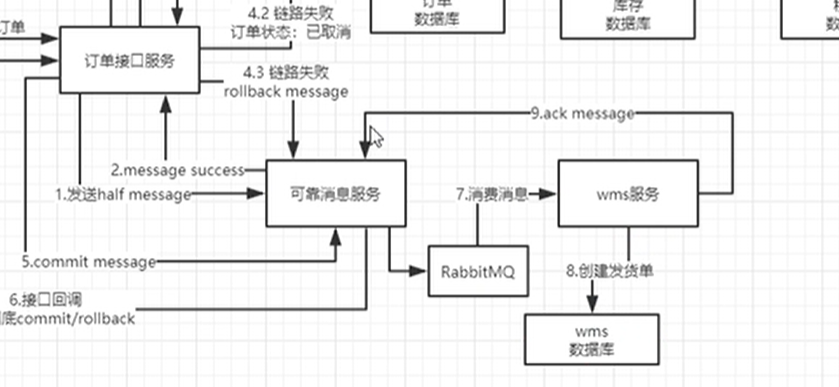


**完整的业务链路**

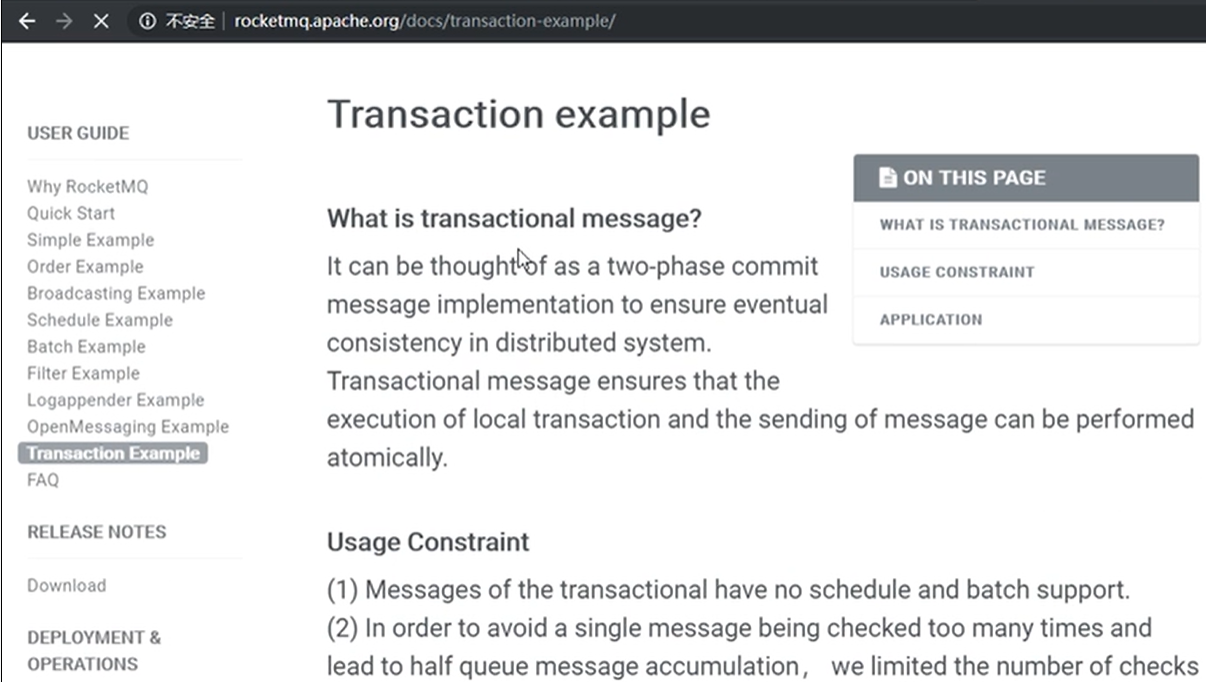


问题：如果没有本地RocketMQ，如何实现？

参考下图：引入一个消息服务，入自己数据库。如果长时间没得到业务的消费确认，则重发消息



**基于RocketMQ实现实例：官网事务例子**



# 分布式配置中心

## Eureka、Nacos

# 分布式协调 Zookeeper

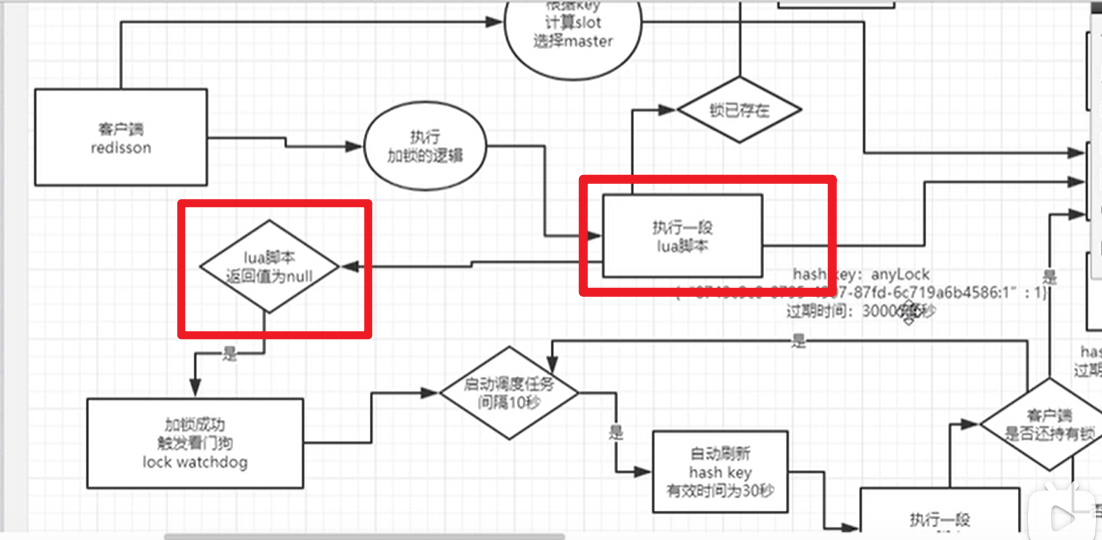
# 分布式限流Sentinel

# 分布式链路追踪 skywalking+zipkin

# 分布式锁

## Redis

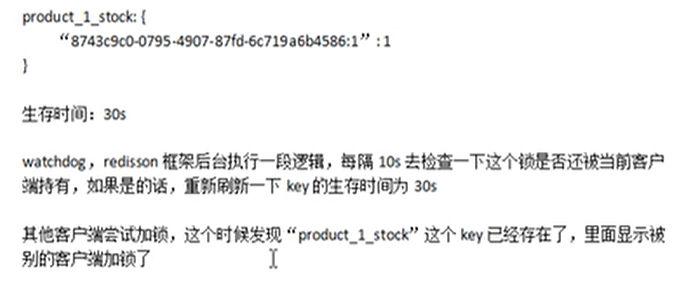
客户端：Redisssion，后台基于Lua



**申请并持有锁**

类似注册信息：锁的KEY， 客户端的ID，当前已被加锁1次。

后台有watchdog，会自动延长锁的持有



**释放锁：**

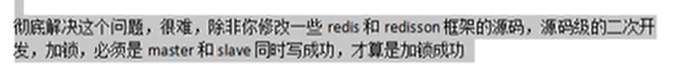
1. 正常场景：主动释放锁
2. 异常：客户端down机，到期后会自动释放

**问题：如果reids集群部署方式，集群故障时，分布式锁可能存在的问题**

集群场景下，当客户端申请的锁未从master同步到slave时。Master挂掉。

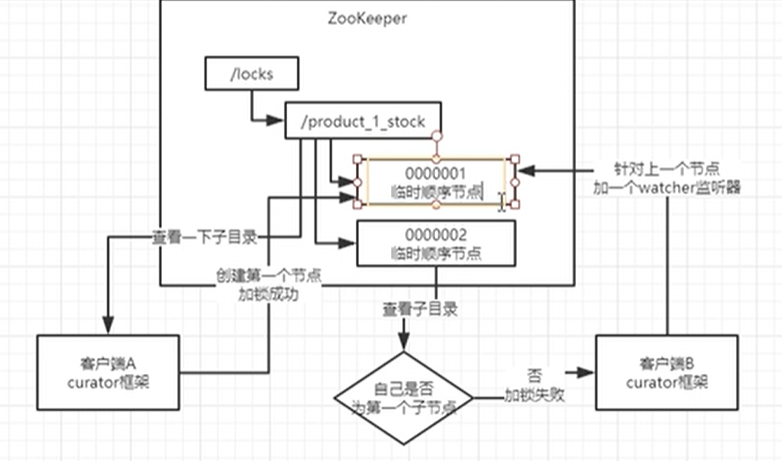
此时新的客户端就可以申请成功，从而出现多个客户端同时申请锁场景。

--- 必须自行改造开源代码，从而必须master和slave同时写成功，才算枷锁成功。解决上述问题。



## 基于zk

### 基于curator的实现



在指定锁目录下，每次都生成新的顺序节点。临时顺序节点，选序号最小

每个获取不到锁的仅对上一个顺序节点加监听，监听锁的释放情况；原有锁释放后再尝试获取获取锁

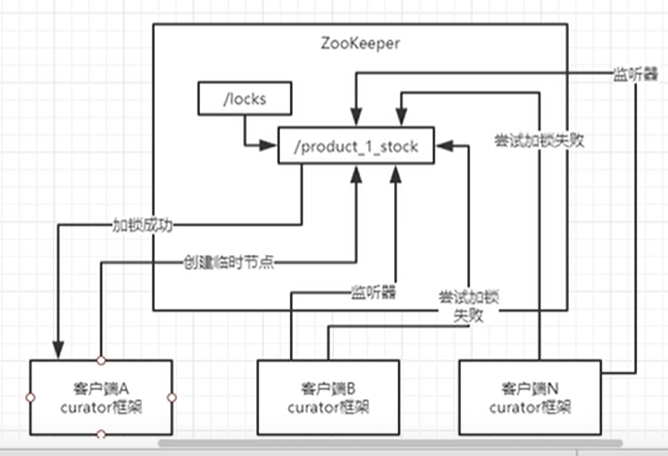
释放锁场景：

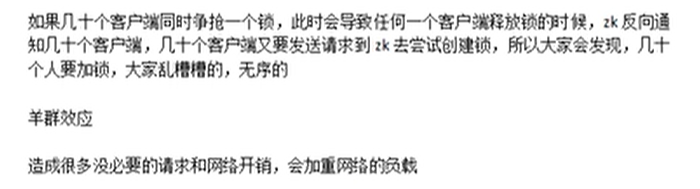
1. 主动释放锁
2. 宕机后，zk会自动检测并释放锁

**--- 上面为curator的实现方式**

### 异常：羊群效应问题

非正确使用，所有客户端都尝试创建下图stock节点，而不是在其下面申请子顺序节点。如果原有锁被释放，则一对排队客户端都来尝试申请加锁。





### 异常：脑裂场景

因为网络问题，ZK错误将当前持有锁客户端认为宕机而主动释放锁，下一个客户端也获取到锁成功。但该客户端认为自己仍持有锁。

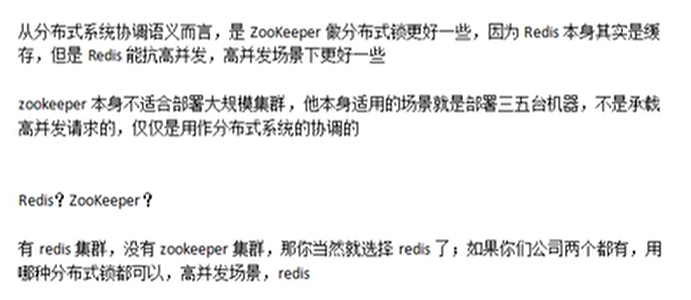
// 导致多个客户端都获取锁

彻底解决：修改curator方案

## 如何选型

Redis抗高并发

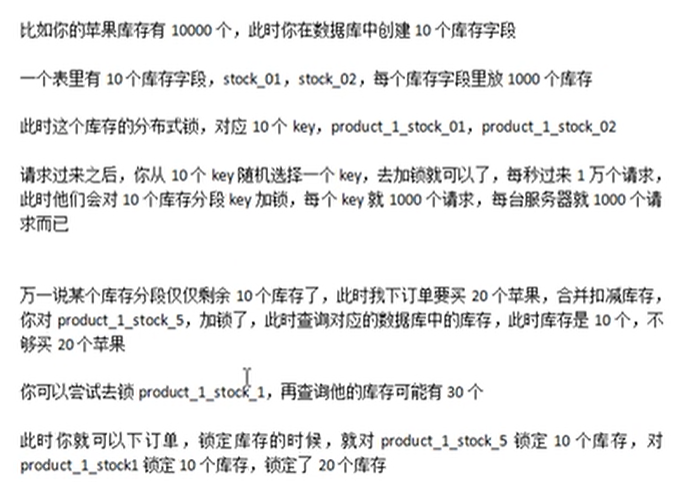
zk不适合部署大规模集群



## 分布式锁高并发优化

思路：分段加锁 + 合并扣减

针对单个段不满足情况，则尝试对多个段同时加锁减库存



## 大厂不基于分布式锁的实现

在缓存中直接扣减，直到扣减后负数（回滚最近一次操作）。

然后通过MQ异步更新数据库。

