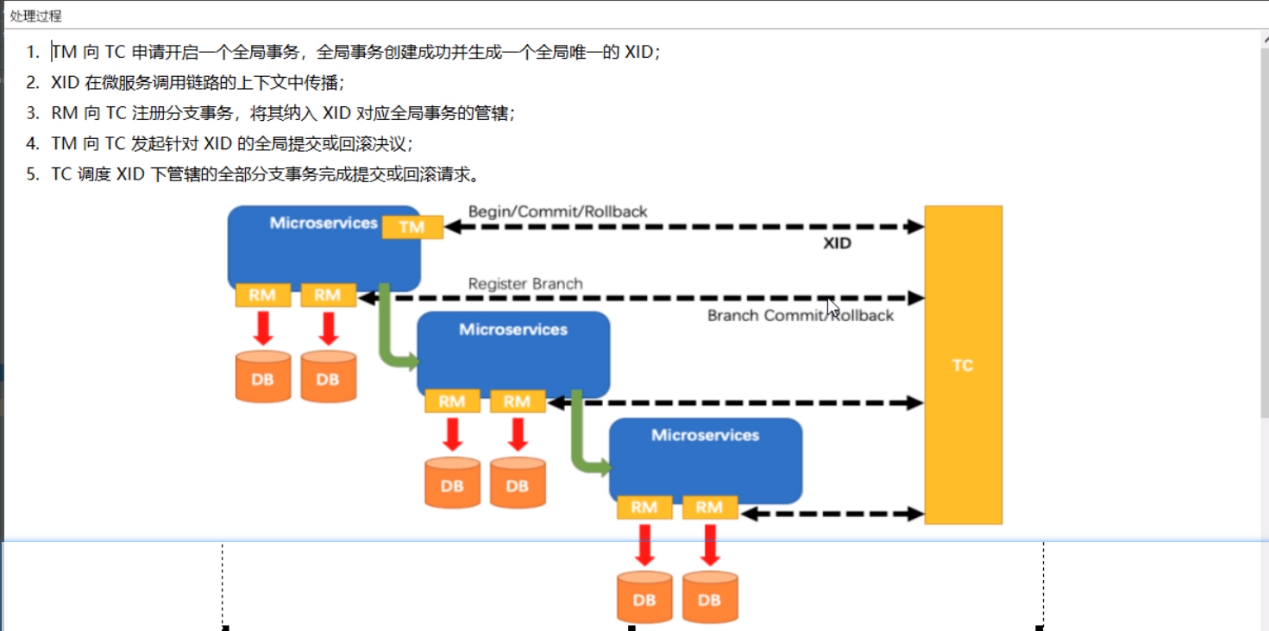
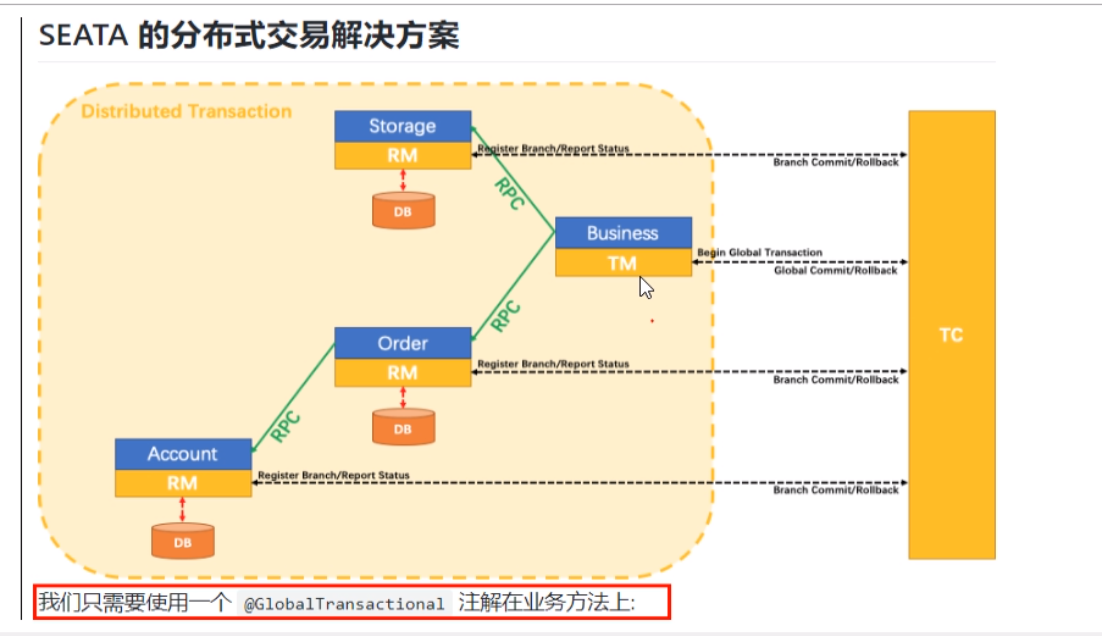
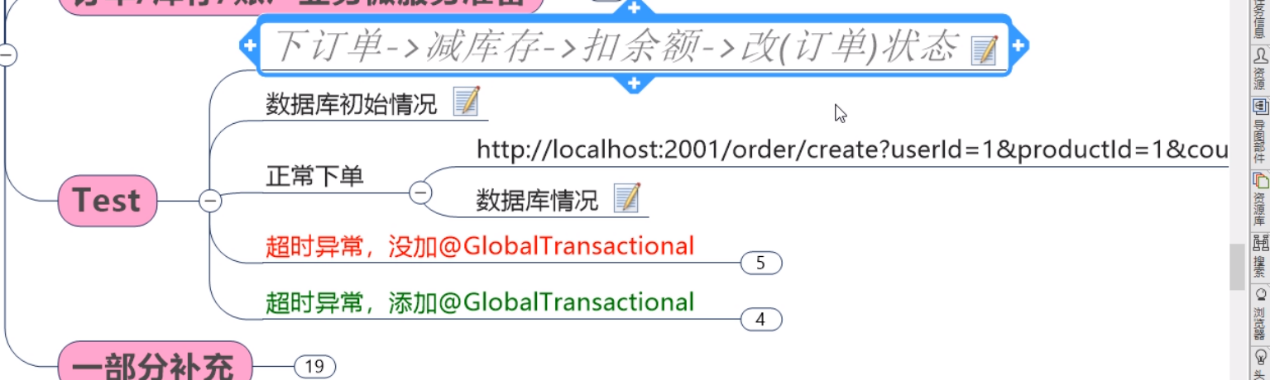
# SpringCloud Alibaba 相关学习-Seata

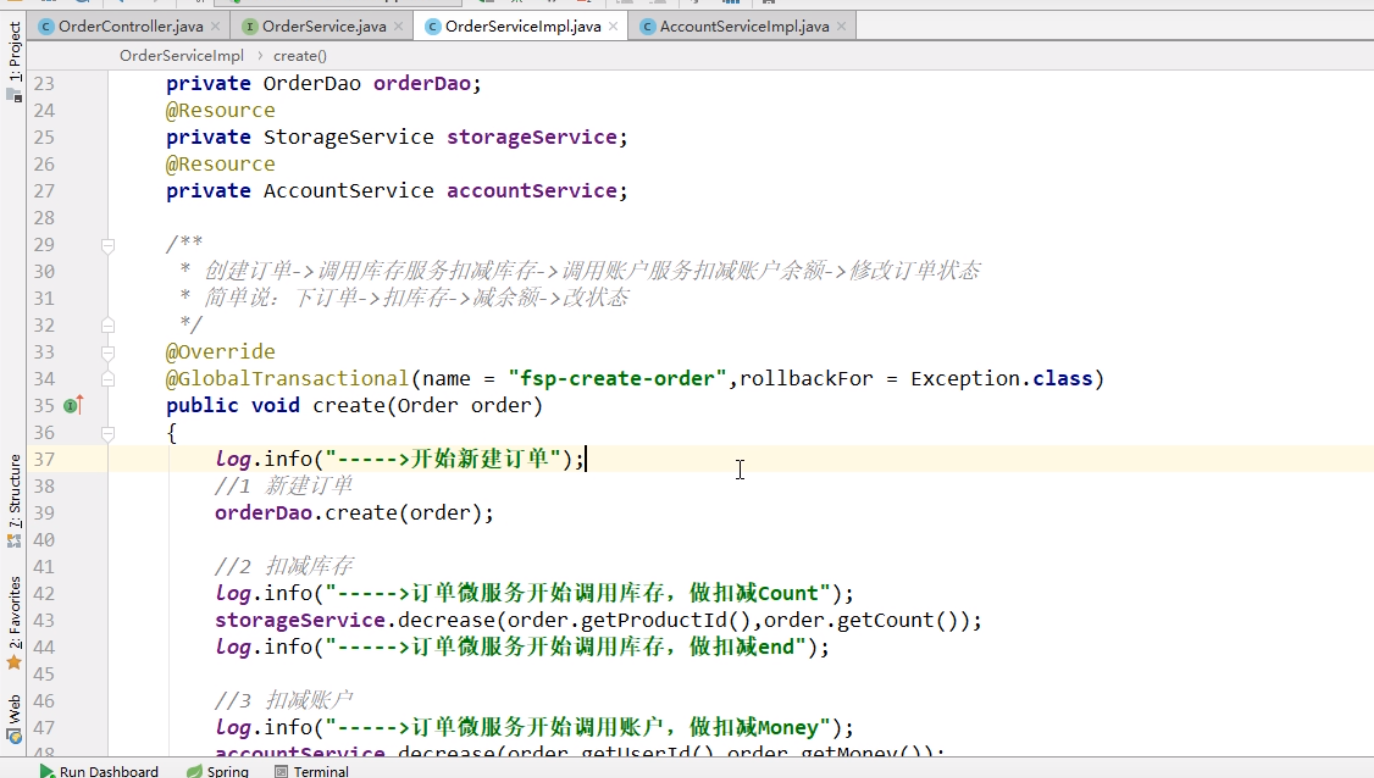
1、分布式事务问题的由来：

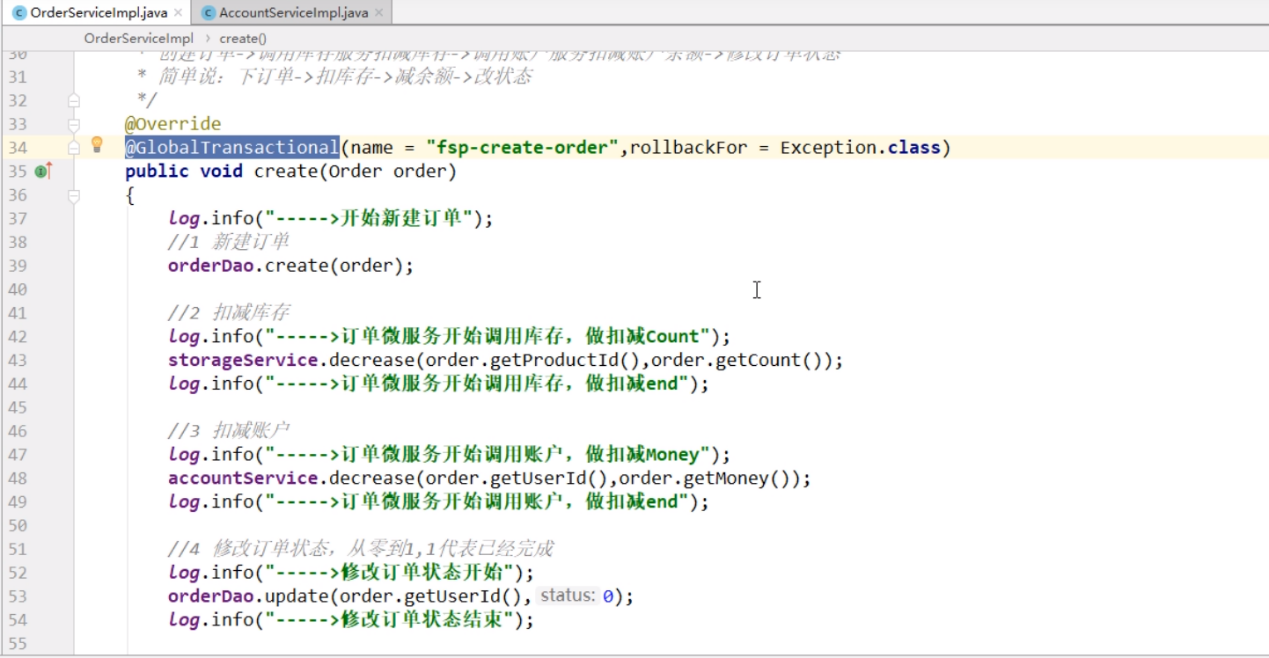


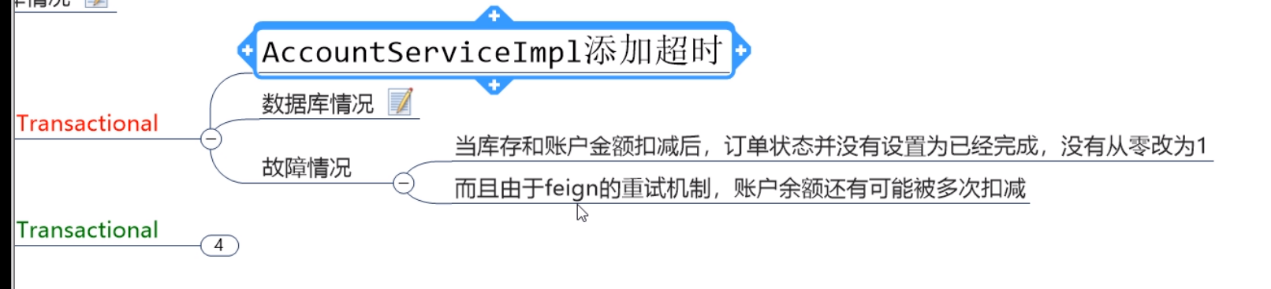


相关业务逻辑代码：

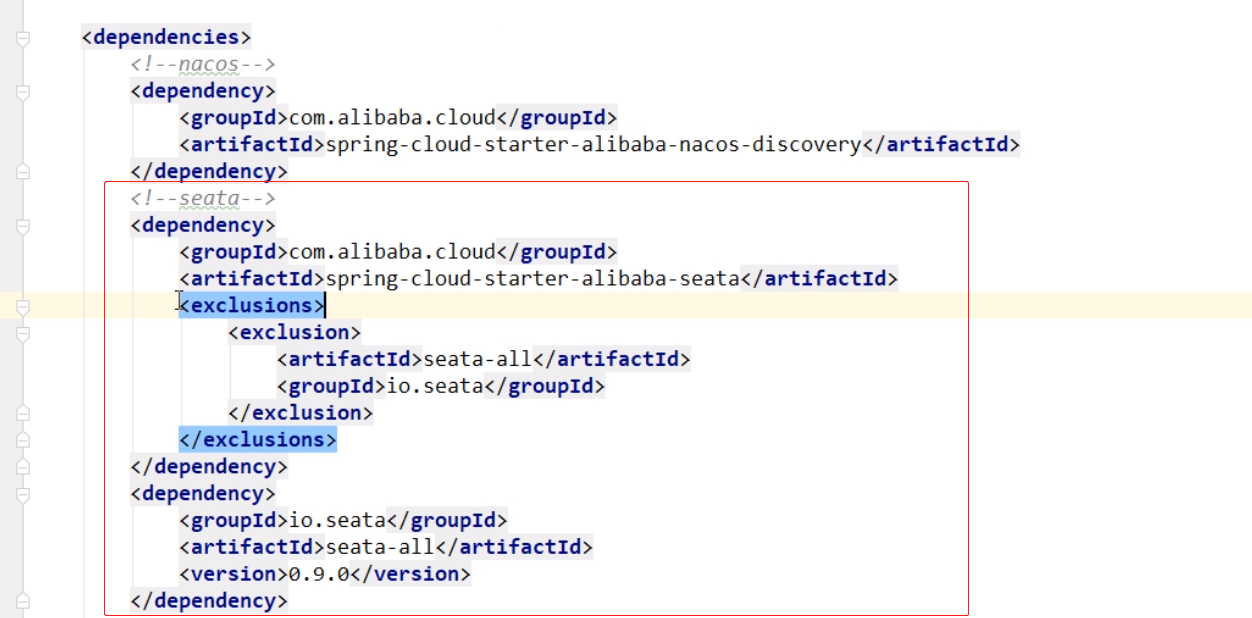




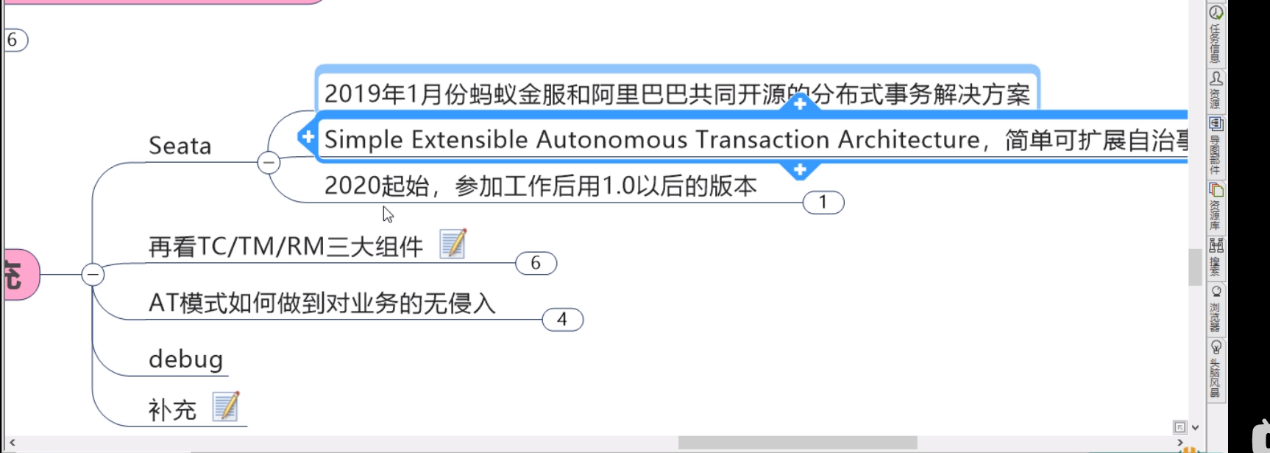




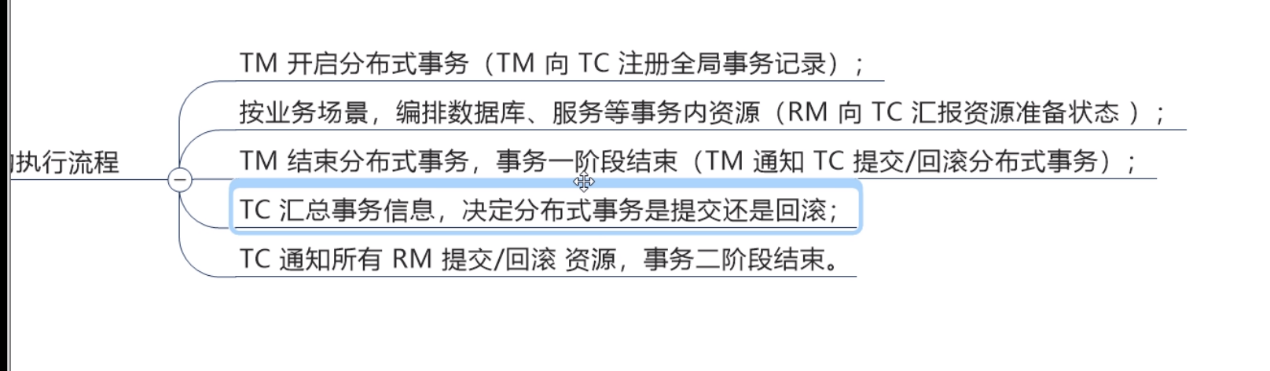
Seata依赖：

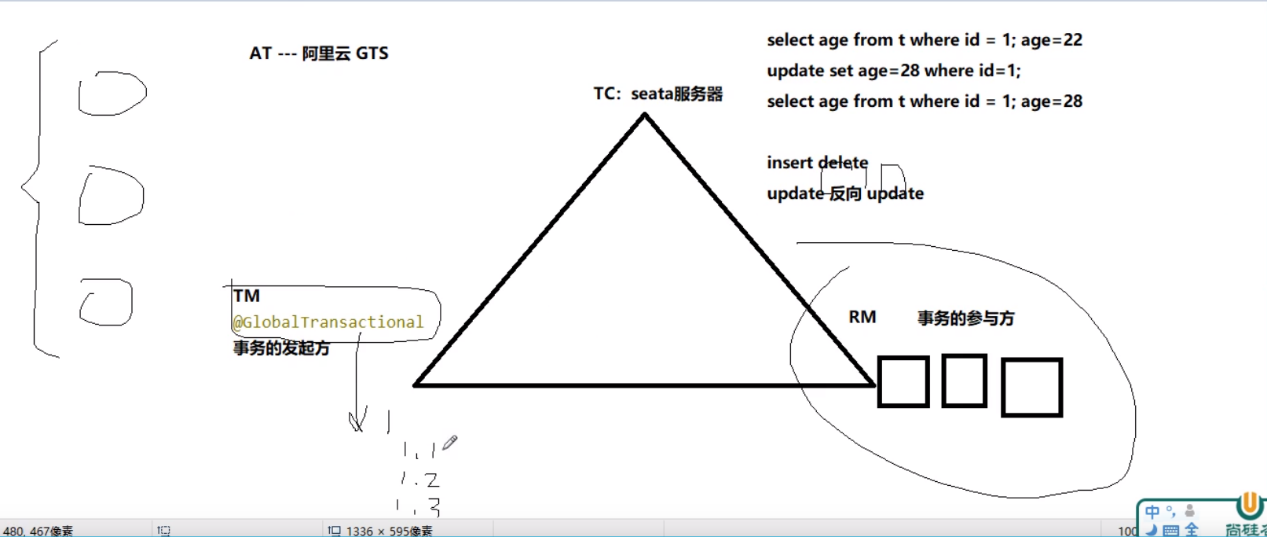


Seata 补充：

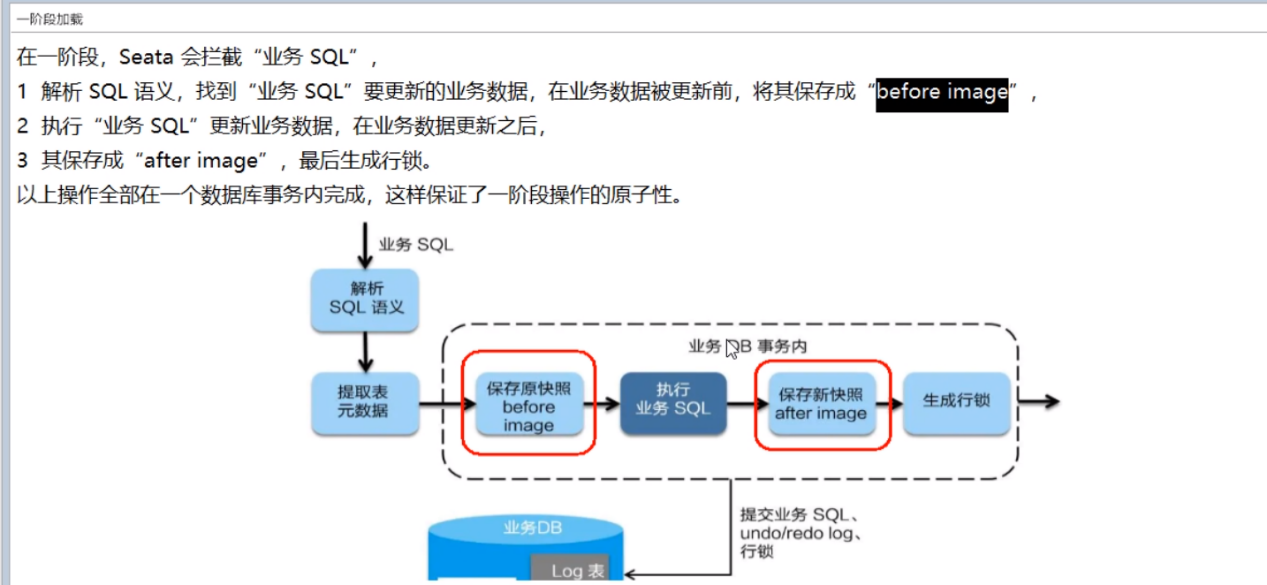


分布式事务的执行流程：



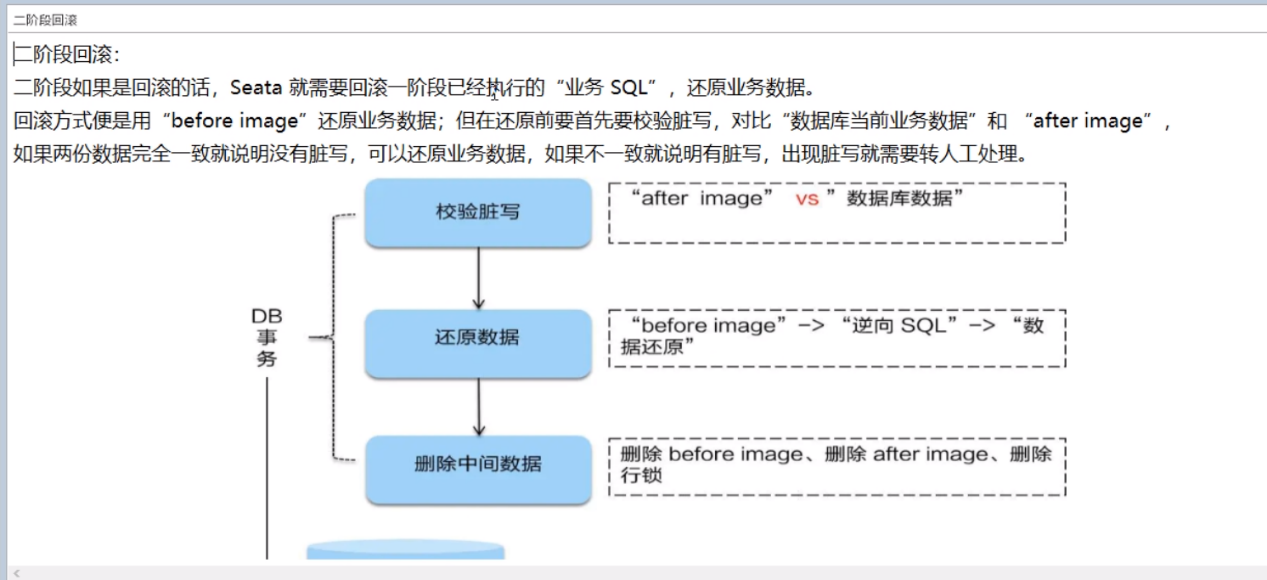


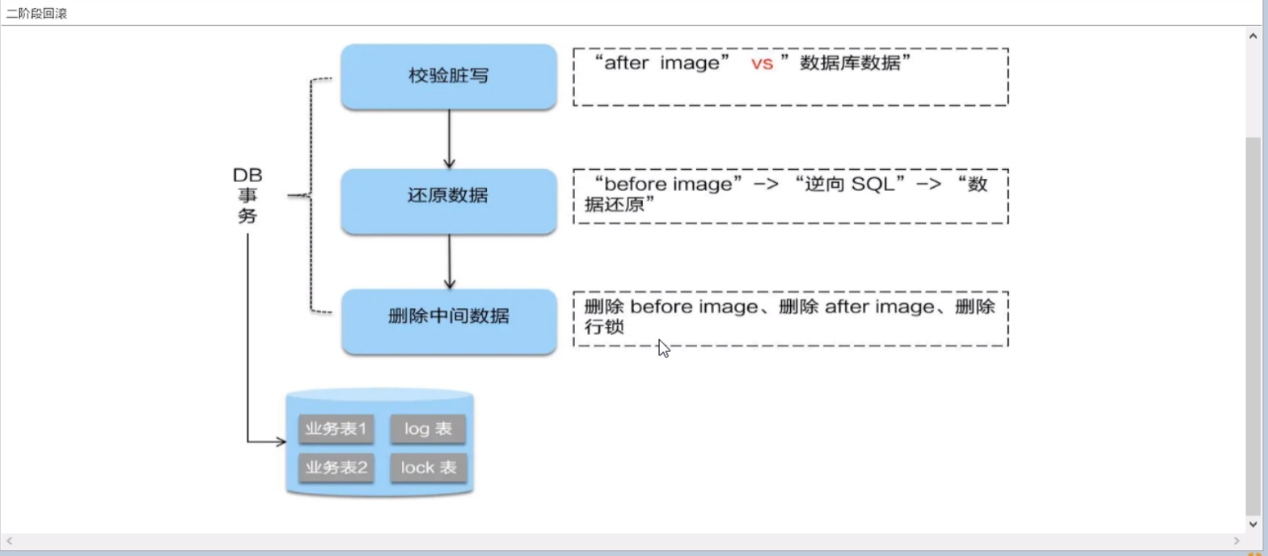


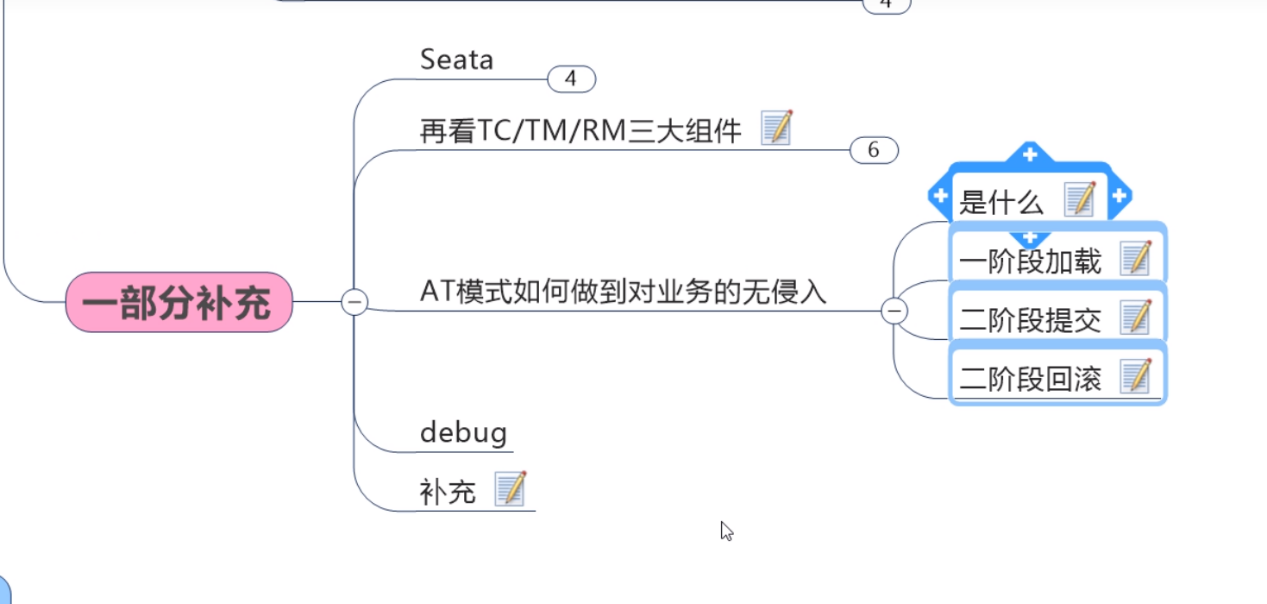


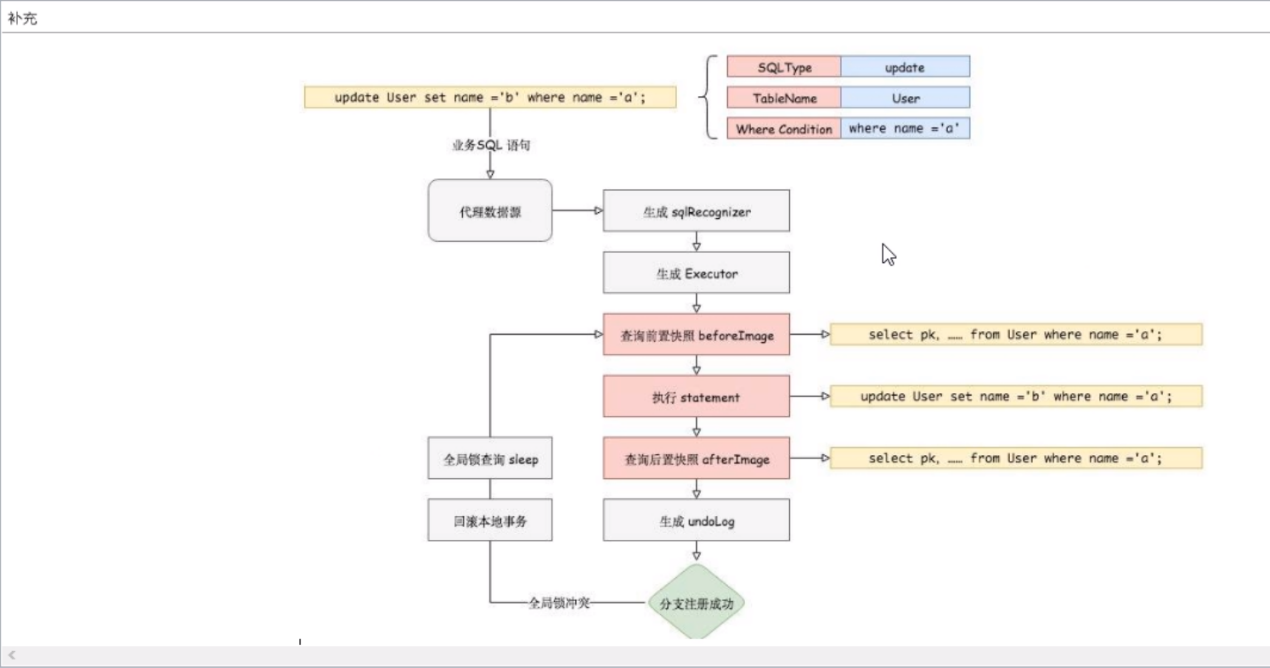


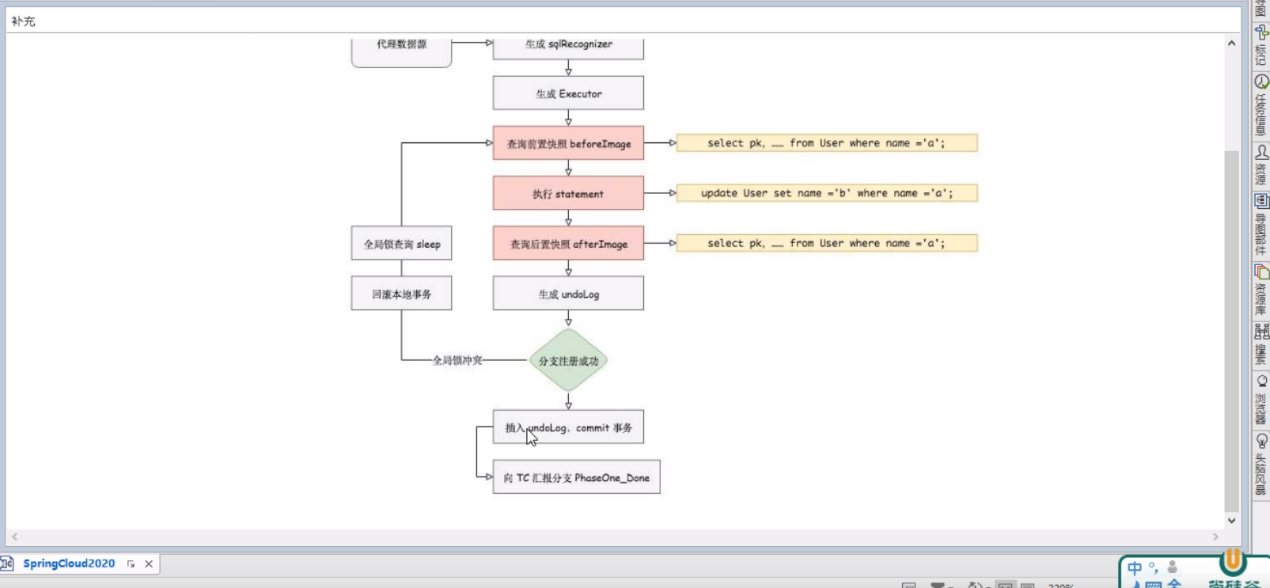
二阶段回滚：











分布式事务TCC：

解释链接：<https://www.cnblogs.com/jajian/p/10014145.html>

所以说，我们有必要使用 TCC 分布式事务机制来保证各个服务形成一个整体性的事务。

[1] 订单服务-修改订单状态，

[2] 库存服务-扣减库存，

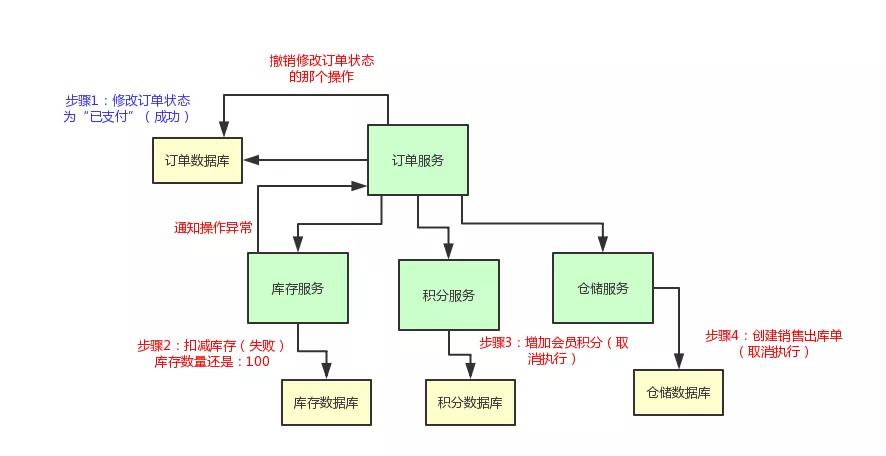
[3] 积分服务-增加积分，

[4] 仓储服务-创建销售出库单。

上面那几个步骤，要么全部成功，如果任何一个服务的操作失败了，就全部一起回滚，撤销已经完成的操作。

比如说库存服务要是扣减库存失败了，那么订单服务就得撤销那个修改订单状态的操作，然后得停止执行增加积分和通知出库两个操作。

说了那么多，老规矩，给大家上一张图，大伙儿顺着图来直观的感受一下：



消息异步解决：

上面咱们聊了聊 TCC 分布式事务，对于常见的微服务系统，大部分接口调用是同步的，也就是一个服务直接调用另外一个服务的接口。

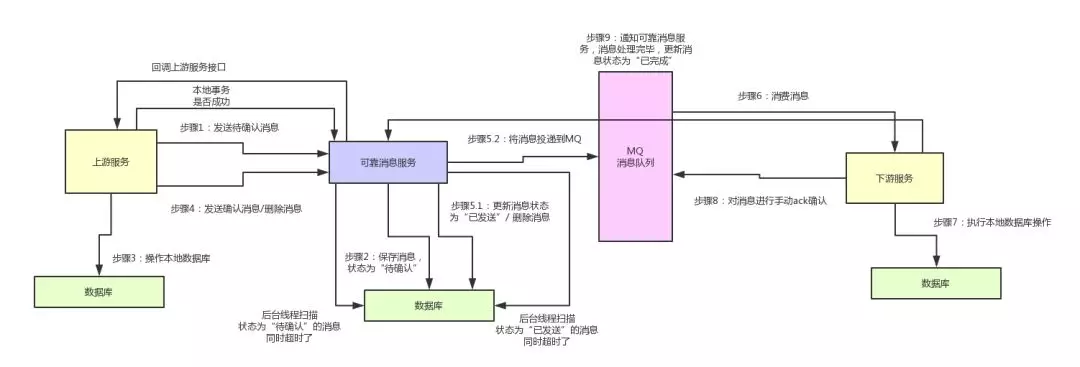
这个时候，用 TCC 分布式事务方案来保证各个接口的调用，要么一起成功，要么一起回滚，是比较合适的。

但是在实际系统的开发过程中，可能服务间的调用是异步的。也就是说，一个服务发送一个消息给 MQ，即消息中间件，比如 RocketMQ、RabbitMQ、Kafka、ActiveMQ 等等。

然后，另外一个服务从 MQ 消费到一条消息后进行处理。这就成了基于 MQ 的异步调用了。

那么针对这种基于 MQ 的异步调用，如何保证各个服务间的分布式事务呢？也就是说，我希望的是基于 MQ 实现异步调用的多个服务的业务逻辑，要么一起成功，要么一起失败。

这个时候，就要用上可靠消息最终一致性方案，来实现分布式事务。



大家看上图，如果不考虑各种高并发、高可用等技术挑战的话，单从“可靠消息”以及“最终一致性”两个角度来考虑，这种分布式事务方案还是比较简单的。