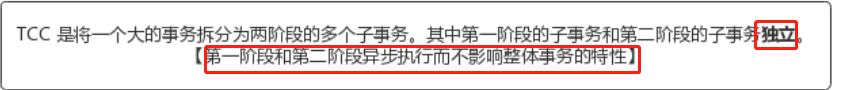
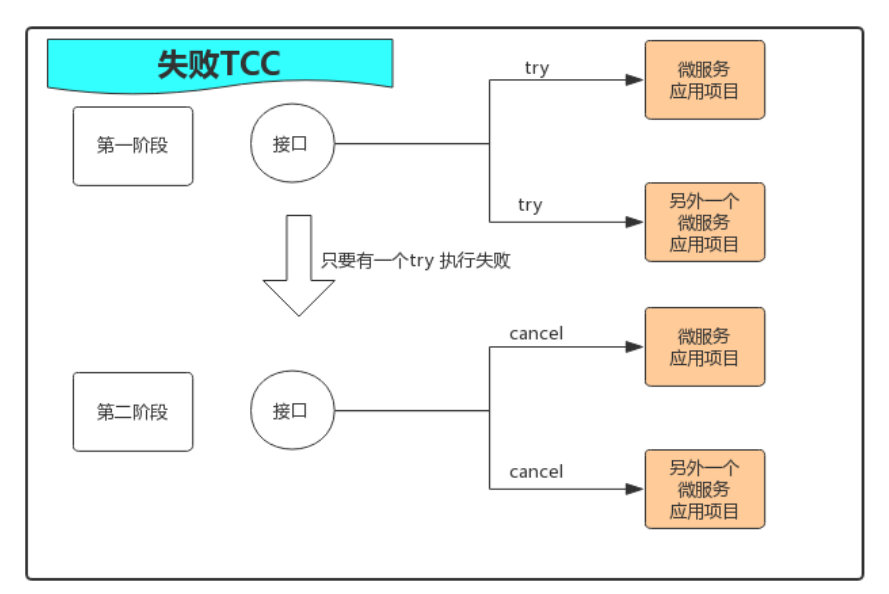
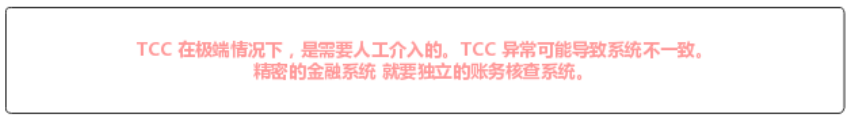


XA适合一个方法中设计多个数据库，

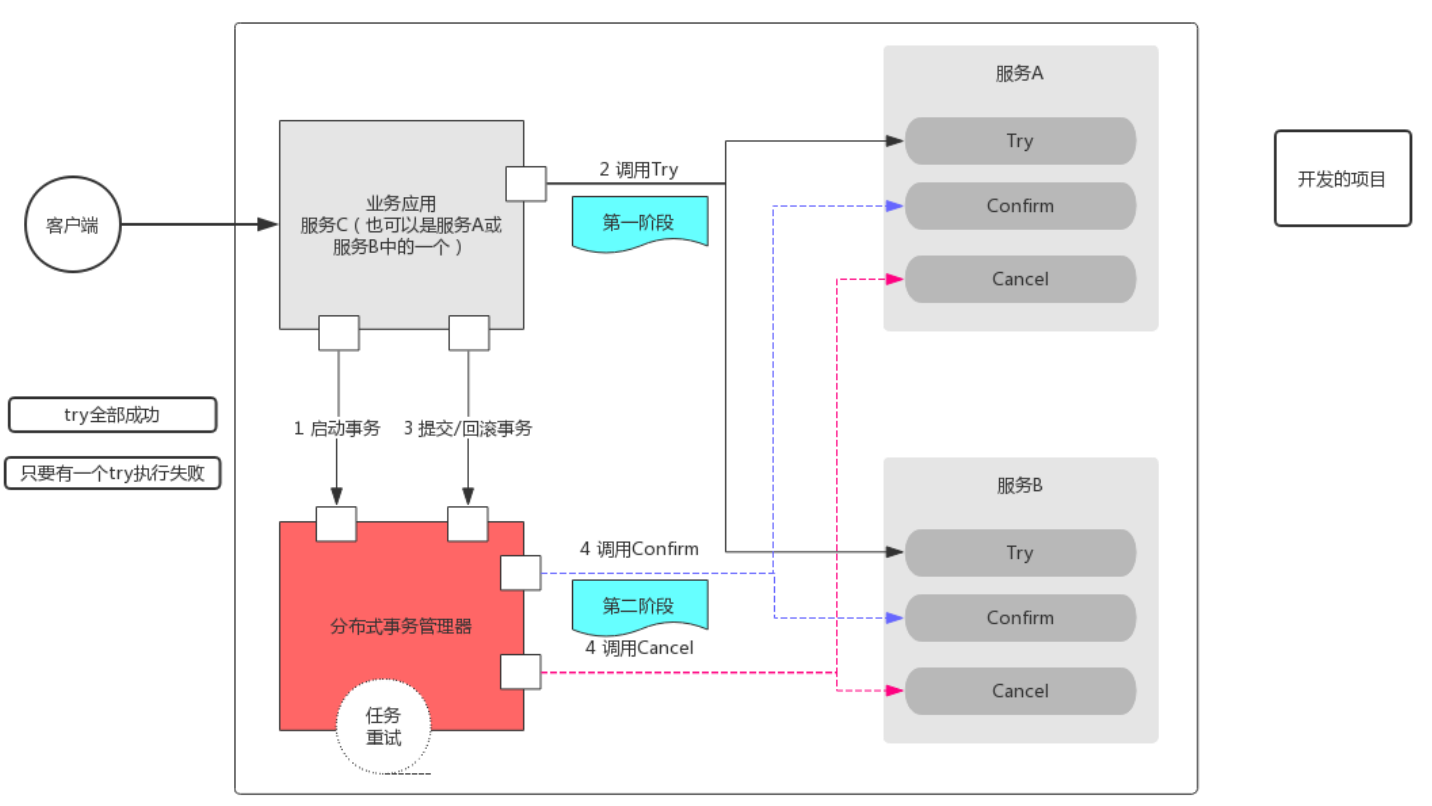
不同服务之间的事务需要TCC和mq最终一致







第二阶段一直confirm不成功，或者超过重试次数



Himly就是一种事务管理器

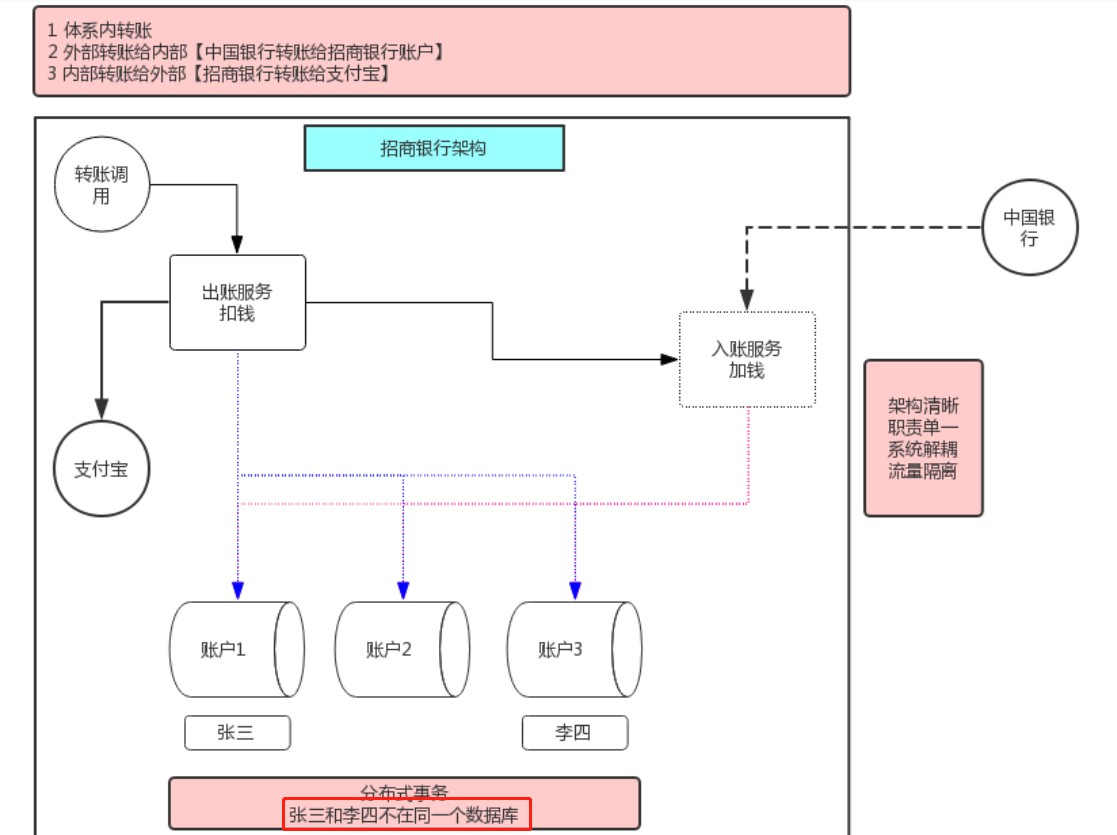
服务C（可以是A或B） 去TM启动事务

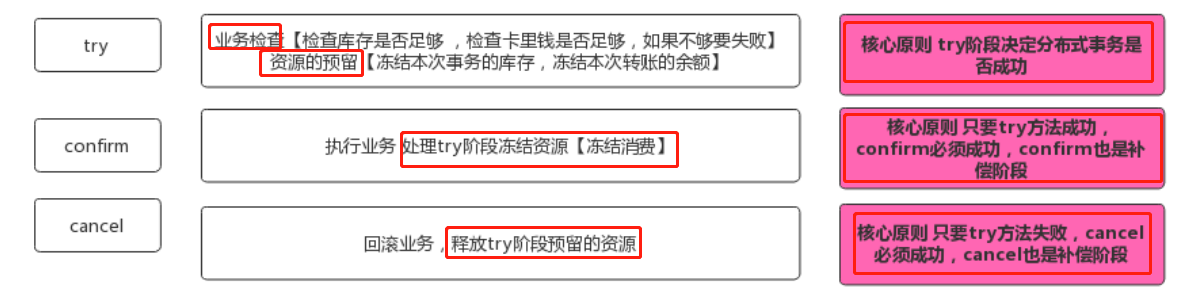
调用两个服务的try

去TM执行提交或回滚（有重试机制）

调用两个服务的confirm或cancel

客户端访问业务应用（可以是A或B），业务应用将会去访问事务管理器启动事务，并调用A，B两个服务的try方法。当执行结束后，业务应用将会更加try方法的结果更新事务管理器的状态，然后事务管理器将会去A，B服务执行对应的confirm方法或cancel方法



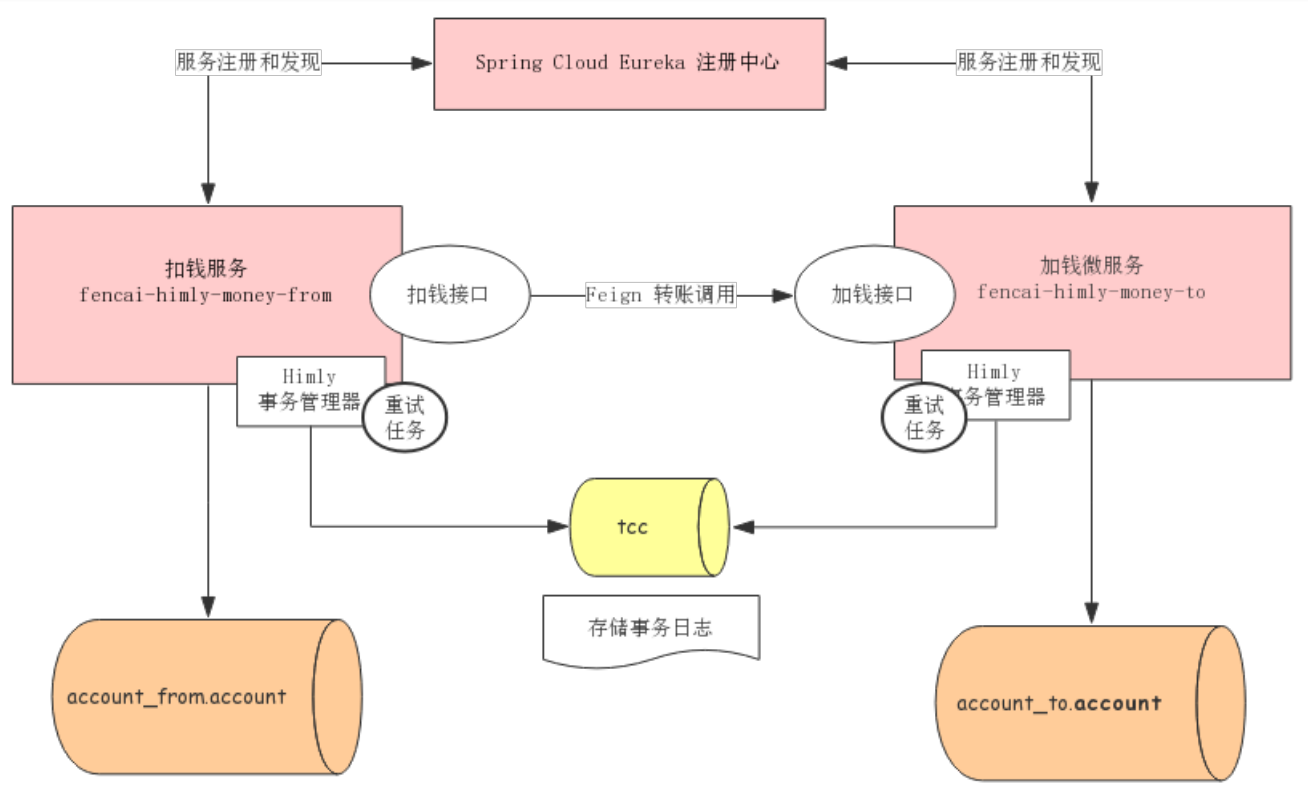


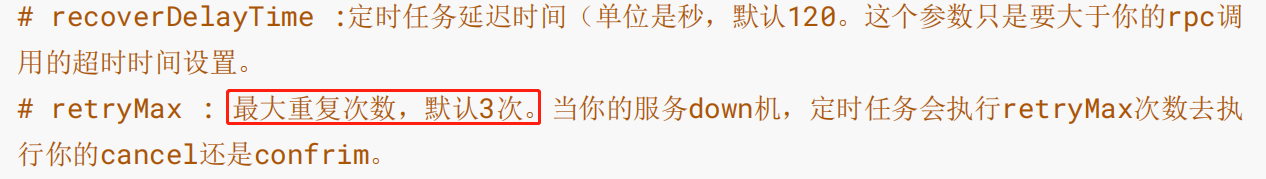
Try很重要，而且只能少钱后面补偿回去，不能加钱

业务检查，看事务能不能超过，相当于XA是prepare；资源冻结（XA是锁全部，prepare是锁一部分）

每个服务都有自己的try confirm cancel

Try（业务检查和资源占用很重要）

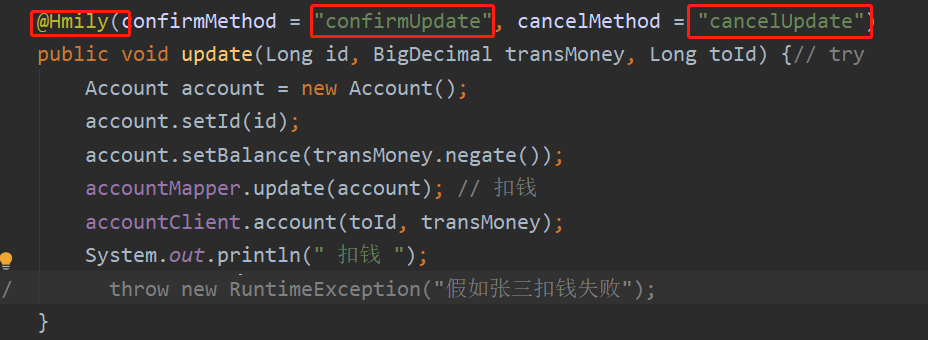




Himly生成者事务通过Feign调用消费端的事务

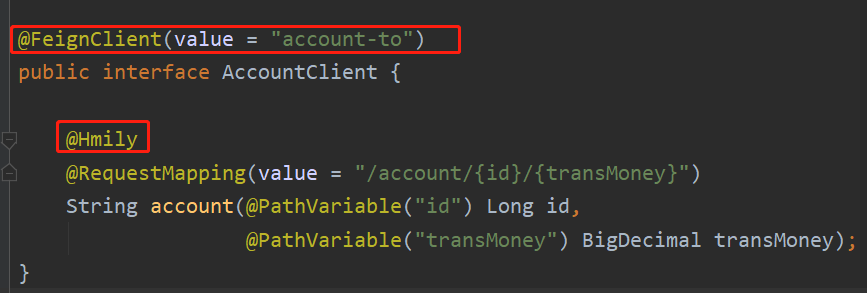


生产者事务



Controller到达update即为try，不跑错通过feign调用消费端事务



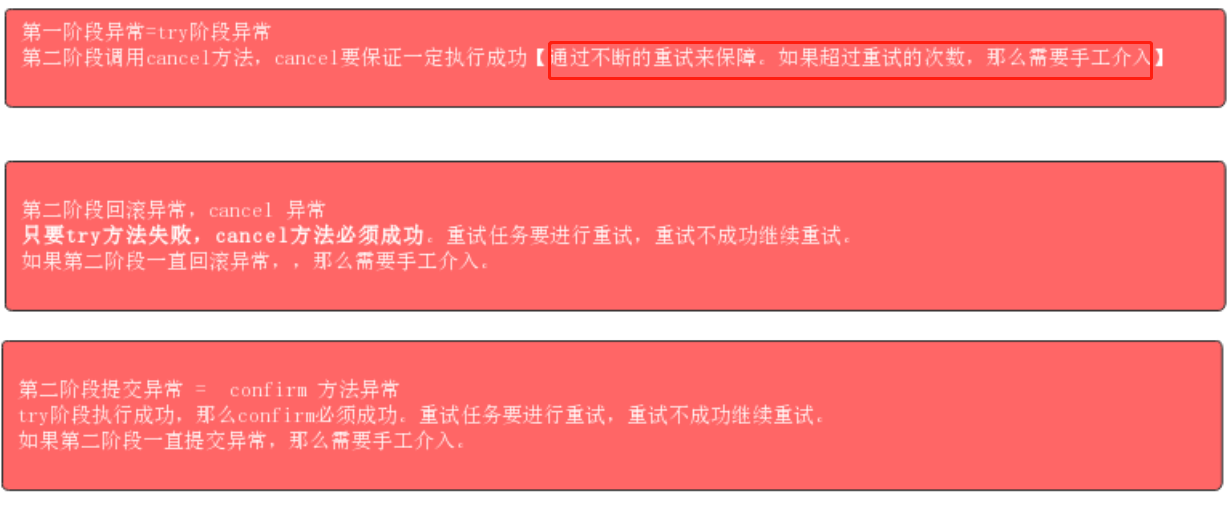


消费端事务



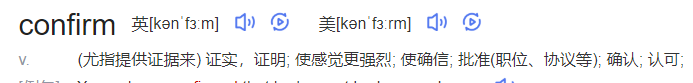
yml的重要属性

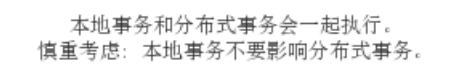


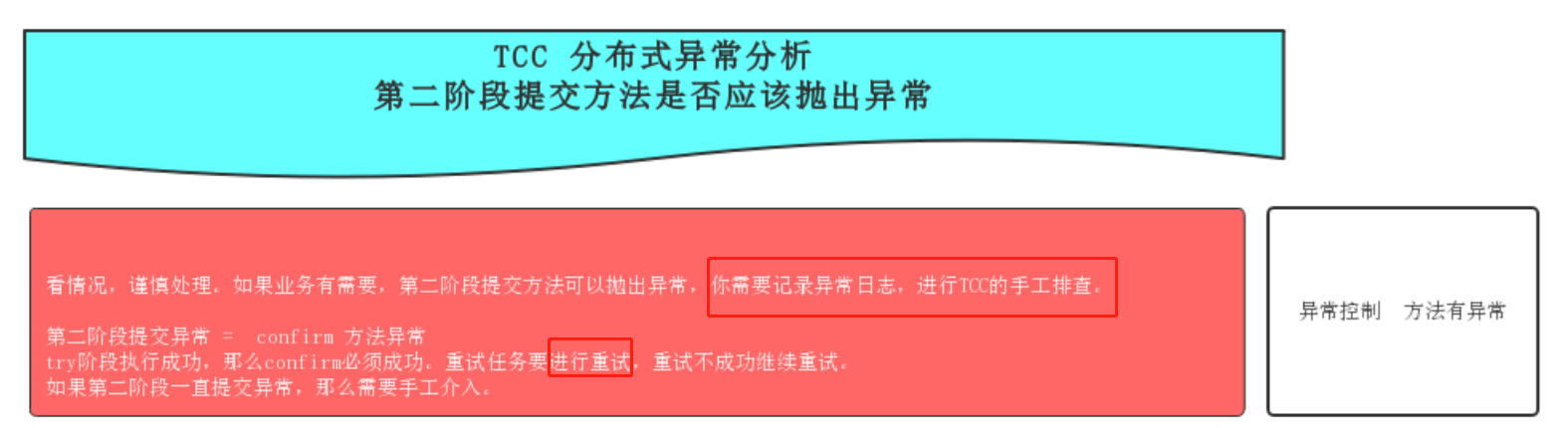


try成功，confirm一定执行成功；try失败，cancel一定执行成功

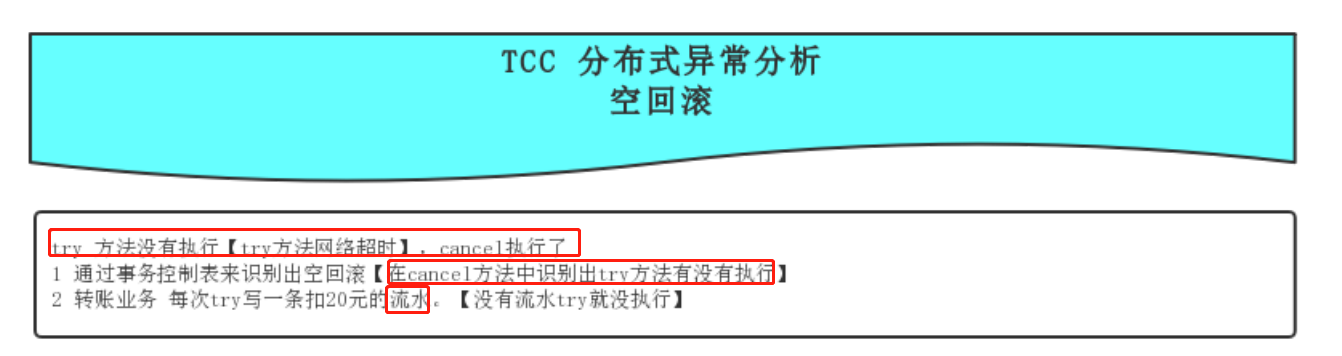
如果confirm和cancel失败，重试后手工介入



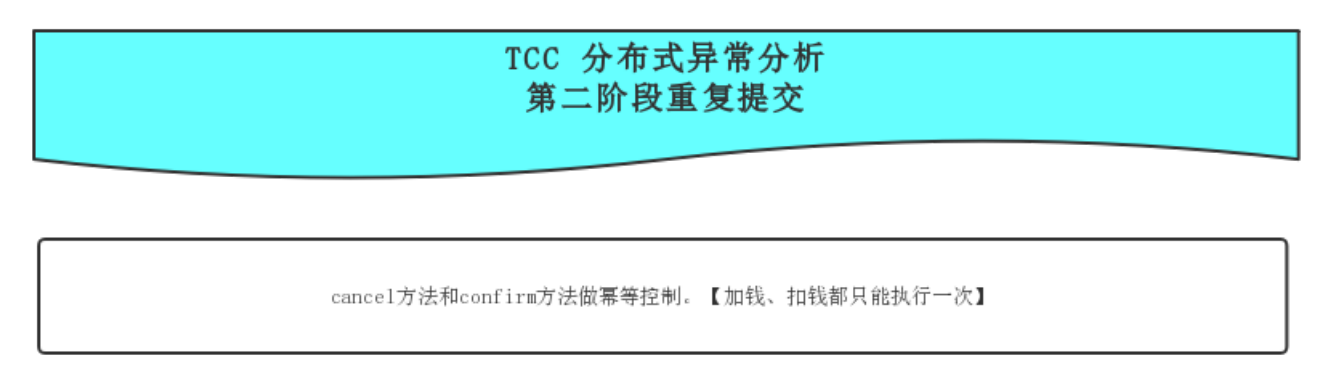




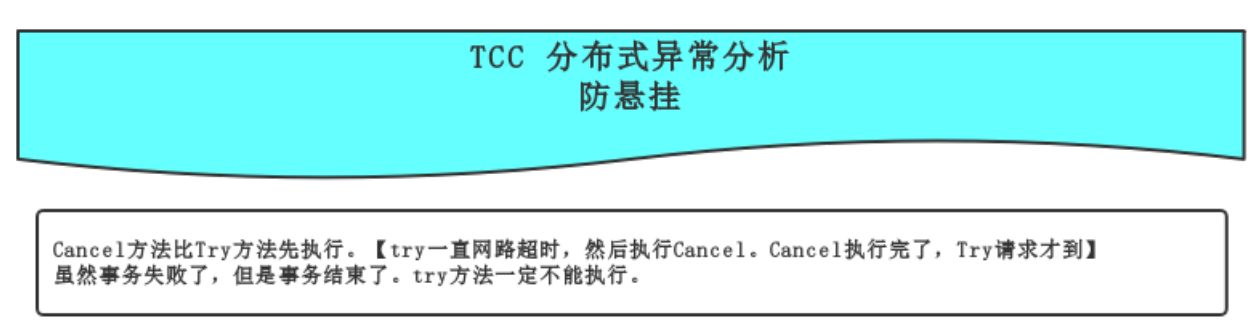
第二阶段抛出异常会不断重试，一定次数人工介入



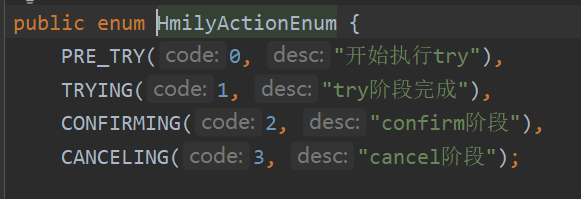
空回滚检查

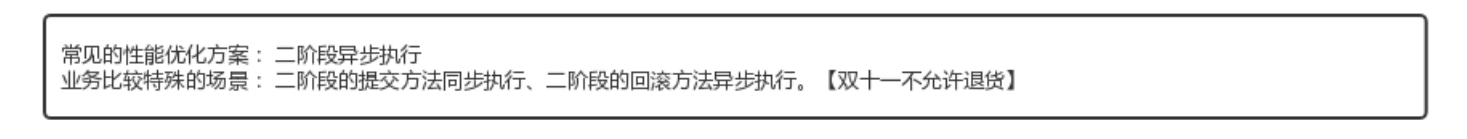


幂等性处理（网络异常，执行结束后没告诉TM我执行成功了，触发重试）

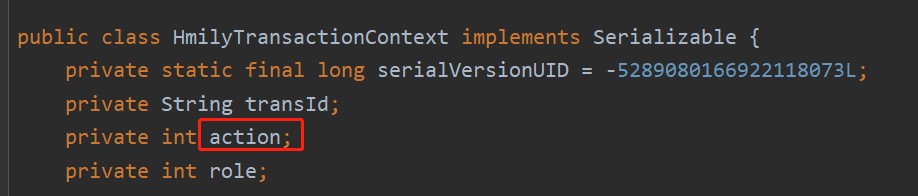


需要在try方法检查事务是否结束





hmilyTransactionContext = HmilyTransactionContextLocal.getInstance().get();



1. try方法执行成功，cancel和confirm方法一定要执行成功，允许非业务逻辑的出错，可以通过重试解决。业务逻辑的出错，需要人工介入
2. 防止空回滚，try没执行，cancel执行了，cancel需要做try判断
3. 第二阶段需要做幂等性处理
4. 防止悬挂，cancel比try先执行，此时需要在try方法判断事务是否已经结束