**一：TCC**（原理、优缺点、最终一致性）

1. TCC：分别对应Try、Confirm和Cancel三种操作。

（1）Try：预留业务资源。（DML）

（2）Confirm：确认执行业务操作。（Commit）

（3）Cancel：取消执行业务操作。（Rollback）

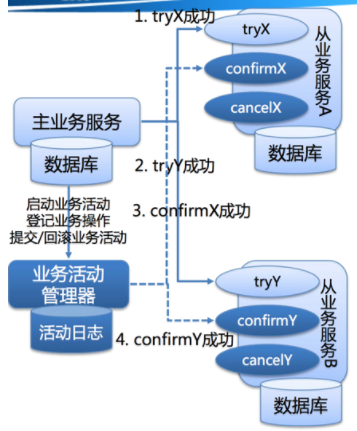


在一个跨应用的业务操作中，Try操作是先把多个应用中的业务资源预留和锁定住，为后续的确认打下基础，类似的，DML操作要锁定数据库记录行，持有数据库资源。

Confirm操作是在Try操作中涉及的所有应用均成功之后进行确认，使用预留的业务资源，和Commit类似。

而Cancel则是当Try操作中涉及的所有应用没有全部成功，需要将已成功的应用进行取消(即Rollback回滚)。其中Confirm和Cancel操作是一对反向业务操作。

2、TCC原理图



（1）主业务服务（order）：主业务服务为整个业务活动的发起方。从业务服务（account、inventory）：从业务服务负责提供TCC业务操作，是整个业务活动的操作方。从业务服务必须实现Try、Confirm和Cancel三个接口，供主业务服务调用。由于Confirm和Cancel操作可能被重复调用，故要求Confirm和Cancel两个接口必须是幂等的。

（2）业务活动管理器：业务活动管理器管理控制整个业务活动，包括记录维护TCC全局事务的事务状态和每个从业务服务的子事务状态，并在业务活动提交时确认所有的TCC型操作的confirm操作，在业务活动取消时调用所有TCC型操作的cancel操作。

3、TCC优点和缺点

优点：解决了跨应用业务操作的原子性问题，在诸如组合支付、账务拆分场景非常实用。TCC实际上把数据库层的二阶段提交上提到了应用层来实现，对于数据库来说是一阶段提交，规避了数据库层的2PC性能低下问题。

缺点：TCC的Try、Confirm和Cancel操作功能需业务提供，开发成本高。

4、TCC案例：

订单业务场景如下，有三个服务订单服务、库存服务和账户服务。进行扣除库存30、账户减少金额50，同时记录订单信息的操作。

**Try：尝试执行业务，完成所有业务检查(一致性)。**

（1）检查订单服务、库存服务和账户服务是否正常。库存中的数量是否不少于30，账户的余额是否不少于50。

（2）预留必须业务资源(准隔离性)：库存的冻结数量增加30，账户的冻结金额增加50元，这样就保证不会出现其他并发进程扣减了库存和账户金额而导致在后续的真正扣库存、扣金额的操作过程中，库存和账户余额不够的情况。

**Confirm：确认执行业务。**

（1）如果Try阶段帐户订单服务、库存服务、帐户服务状态正常，且库存、账户够用，则执行库存扣除30、帐户减少50元的操作。

（2）该阶段不做任何业务检查： Try阶段已经完成了业务检查。只使用Try阶段预留的业务资源：只需要使用Try阶段库存和帐户冻结的库存和金额即可。

**Cancel：取消执行业务释放Try阶段预留的业务资源**。

（1）如果Try阶段部分成功，比如库存够用，且冻结相应库存成功，帐户的余额不够而冻结失败，则需要对库存做Cancel操作，将库存被冻结的金额解冻掉。

5、问题分析：

问题：Try阶段成功；Confirm阶段出现问题；此时回滚事务，同时通过事务补偿的方式进行事务回滚。

解决方案：以订单、库存、金额为例，库存减30、金额减50。

（1）订单表中增加订单状态的字段，分为四种状态：未支付、支付中、支付成功和支付失败。

（2）在订单形成的过程中，检查订单、库存、金额三个服务的状态是否正常，库存是否不少于30，金额是否不少于50。

a、库存表中有总数量字段和冻结数量字段。

b、资金表中有总金额和冻结金额字段。

c、订单表中保存新形成的订单信息。包括的字段有：订单号、订单状态、消费的库存数量、消费的金额数量。

（3）当try阶段没有问题时，confirm阶段出现问题，此时订单的状态在数据库中记录成失败；库存数量进行了回滚，但是金额没有进行回滚，部分金额还是处于锁定状态。

（4）通过db、redis或其它方式将出现问题的为服务进行记录，采用补偿的方式进行金额事务回滚。以transID作为要进行回滚事务的唯一标识，采用该种方法最终达到数据的一致性。（补偿的方式可以采用人工操作+定时任务的方式进行）。

6、要不要使用TCC:

（1）是否真正有保证跨应用业务操作的原子性需求。

（2）能否投入资源开发相对应的TCC接口。

（3）需要开发一个稳定的、高可用的、扩展性强的TCC事务管理器。

**二、事务强一致性**（Atomikos框架）

1、java平台提供的开源事务管理器。分布式事务一般情况下需要一个事务管理器和多个资源管理器组成。

2、Atomikos主要功能

（1）兼容标准的JTA API。

（2）支持嵌套事务。

（3）Atomikos TransactionsEssentials是一个非常快速的嵌入式事务管理器，不需要另外启动一个单独的事务管理器进程。该事务管理器需要从数据库获得XAConnection , XAResource等对象。

（4）Atomikos中资源管理器通过XA协议提供，资源管理器包括数据库、消息队列以及文件系统等。

（5）XA是作为资源管理器与[事务](https://baike.baidu.com/item/%E4%BA%8B%E5%8A%A1)管理器的接口标准，XA协议采用两阶段提交方式来管理[分布式事务](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E4%BA%8B%E5%8A%A1)。

3、Atomikos进行事务控制的流程，通过日志来记录操作。从上到下，哪一部发生错误都会回滚。



4、XA连接是一个JTA事务中的参与者。XA连接不支持JDBC的自动提交特性。也就是说应用程序不必在XA连接上调用Java.sql.Connection.commit()或java.sql.Connection.rollback()；

而应用程序应该使用UserTransaction.begin(),UserTransaction.commit()和UserTransaction.rollback()。

5、Atomikos优缺点：

优点：

（1）能够保证数据的强一致性

（2）XA协议简单，使用分布式事务成本低。

缺点：

（1）与本地事务相比性能不理想，不能满足高并发的场景，许多nosql数据库不支持XA协议。

（2）同步阻塞问题。执行过程中，所有参与节点都是事务阻塞型的。当参与者占有公共资源时，其它第三方节点访问公共资源不得不处于阻塞状态。