什么是TCC

大道至简 悟在天成

Andy | 悟纤

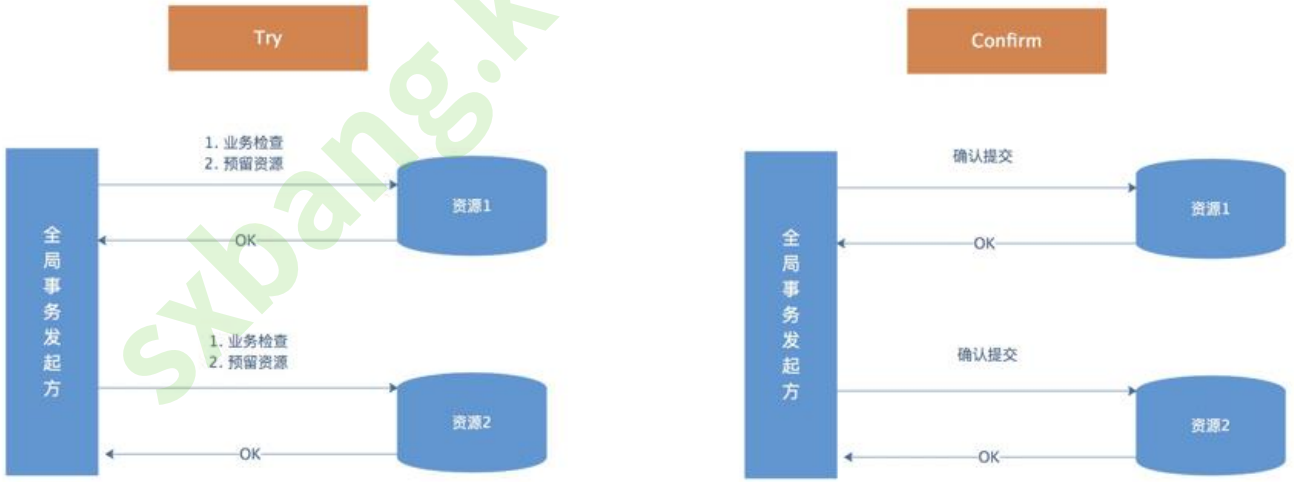
一、什么是TCC

TCC是Try,Confirm,Cancel三个词语的缩写，TCC要求每个分支事务实现三个操作：预处理Try,确认阶段Confirm,撤销Cancel。

Try操作做业务检查和资源预留，confirm做业务确认操作，Cancel实现一个与Try相反的操作即回滚操作。

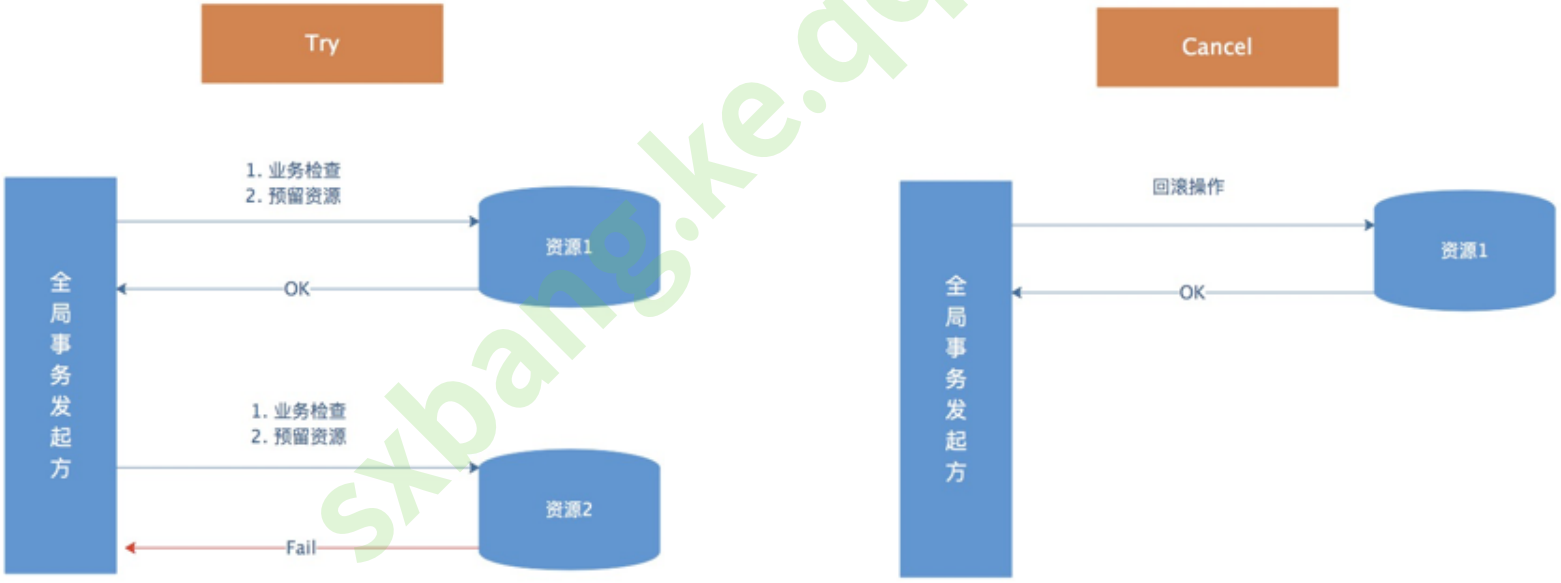
一、什么是TCC

TM首先发起所有的分支事务的try操作，任何一个分支事务的try操作执行失败，TM将会发起所有分支事务的cancel操作；若try成功，TM将会发起所有分支事务的confirm操作，其中confirm/cancel操作若执行失败，TM会进行重试。



一、什么是TCC

分支事务失败的情况：



一、什么是TCC

TCC分三个阶段：

（1）Try业务是做业务检查（一致性）及资源预留（隔离），此阶段是一个初步操作，它和后续的Confirm一起才能真正构成一个完整的业务操作。

一、什么是TCC

TCC分三个阶段：

（2）Confirm阶段是做确认提交，Try阶段所有分支事务执行成功后开始执行Confirm。通常情况下，采用TCC则认为Confirm阶段是不会出错的。即：只要Try成功，Confirm一定会成功。若Confirm阶段真的出错了，需要引入重试机制或人工处理。

一、什么是TCC

TCC分三个阶段：

（3） Cancel阶段是在业务执行错误需要回滚的状态下执行分支事务的取消，预留资源释放。通常情况下，采用TCC则认为Cancel阶段也是一定能成功的。若Cancel阶段真的错误了，需要引入重试机制或人工处理。

一、什么是TCC

TM事务管理器说明：

TM事务管理器可以实现为独立的服务，也可以让全局事务发起方充当TM的角色，TM独立出来是为了成为公用组件，是为了考虑系统结构和软件复用。

TCC异常处理

大道至简 悟在天成

Andy | 悟纤

前言

TCC在开发中处理起来还是比较麻烦的，在实际开发中，我们会碰到这样的一些问题：

(1) 没有执行try方法，却执行了cancel方法 —— 空回滚

(2) 重复执行，导致数据不一致 —— 幂等性。

(3) cancel比try先执行 —— 悬挂。

一、TCC异常处理

1.1 空回滚

描述：

在没有调用TCC资源Try方法的情况下，调用了二阶段的Cancel方法，Cancel方法需要识别出这是一个空回滚，然后直接返回成功。

一、TCC异常处理

1.1 空回滚

出现原因

当一个分支事务所在服务器宕机或者网络异常，分支事务调用为失败，这个时候是没有执行try阶段的，当故障恢复后，分布式事务进行回滚则会调用二阶段的Cancel方法，从而形成空回滚。

一、TCC异常处理

1.1 空回滚

解决思路

解决思路就是需要知道第一阶段是否执行了，如果执行了，那么就是正常回滚；如果没有执行就是空回滚。

全局事务ID会贯穿整个分布式调用链，在额外增加一张分支事务记录表，其中有全局事务ID和分支事务ID。

> (1) 第一阶段Try方法里插入一条记录，表示第一阶段执行了。
>
> (2) Cancel接口里读取记录，如果该记录存在，则正常回滚；如果该记录不存在，则是空回滚。

1.2 幂等性

描述

幂等性：重复调用多次产生的业务结果与调用一次产生的业务结果相同。

1.2 幂等性

出现原因

为了确保网络异常或者超时的情况下，TCC还能正常工作，TCC会通过重试机制确保方法的执行，那么try/confirm/cancel就会出现可能被多次调用的情况，这样就会引发数据不一致，为了保证在重试机制的情况下，数据的一致性，那么就要确保方法执行的幂等性。

1.2 幂等性

解决思路

解决思路就是在上述分支记录中增加执行状态，每次执行都查询该状态。

1.3 悬挂

描述

悬挂就是对于一个分支事务，其二阶段Cancel接口比Try接口先执行。

1.3 悬挂

出现原因

出现原因是在RPC调用分支事务try时，先注册分支事务，在执行RPC调用，如果此时RPC调用的网络发生拥堵，通常RPC调用是有超时时间的，RPC超时之后，TM就会通知RM回滚该分布式事务，可能回滚完成后，RPC请求才到达参与者真正执行，而try方法预留的资源只有该分支事务才能使用，该分支事务第一阶段预留的资源就再也没有人能够处理了，对于这种情况称为悬挂，即预留资源后没法继续处理。

1.3 悬挂

解决思路

解决思路如果二阶段执行完成，那么第一阶段就不能再执行。在执行第一阶段的时候判断该全局事务下“分支事务记录表”中是否已经存在二阶段事务记录，如果有则不执行try。

一、TCC异常处理

小结

（1）幂等性TCC：重试机制引起；涉及到try/comfirm/cancel方法；确保幂等性就是要保证方法多次调用的结果是一样的。

（2）空回滚C：网络异常引起，未执行try，执行了cancel；涉及到cancel方法。

（3）悬挂T：网络异常引起，cancel比try先执行，导致资源悬挂无法被处理；涉及到try方法；

一、TCC异常处理

小结

(1) Try (2) ： try要关注幂等性和悬挂。

(2) Confirm (1) ： confirm只需要关注幂等性。

(3) Cancel (2) ： cancel需要关注幂等性和空回滚。

TCC案例分析

大道至简 悟在天成

Andy | 悟纤

TCC案例分析

场景：A转账30元给B，AB账户在不同的服务。

TCC案例分析

方案1：

账户A：

```
try:
        //检查余额是否够30元
        //扣减30元

  //远程调用服务，进行转账。
confirm:
        //空
cancel:
        //增加30元。
```

TCC案例分析

方案1：

账户B：

```
try:
        //增加30元
confirm:
        //空
cancel:
        //减少30元。
```

TCC案例分析

方案1的问题分析：

(1) 如果账户A的try没有执行，在cancel则多加了30元。

(2) 由于try,confirm,cancel都是由单独的线程去调用，且会出现重复调用，所以都要实现幂等。

(3) 账户B在try中增加30元，当try执行完成后可能会有其他线程消费了。

(4) 如果账户B的try没有执行，执行了cancel，账户B就少了30元。

TCC案例分析

方案1的问题解决：

(1) 账户A的cancel方法需要判断try是否执行，正常执行try后方可执行cancel。

(2) try/comfirm/cancel实现幂等。

(3) 账户B在try方法中不允许执行更新账户金额，在confirm中更新金额。

(4) 账户B的cancel方法需要判断try方法是否执行，正常执行try之后方可执行cancel。

TCC案例分析

优化方案：

账户A：

```
try:
                //try幂等校验：isExistTry(xid);
                //try悬挂处理：isExistCancel(xid);
                //检查余额是否够30元： money > =30
                //扣减30元 : -30
                //插入try执行记录，用户幂等判断。（有些框架底层会实现幂等操作，不用业务层实现）

                //远程调用服务，进行转账。
confirm:
                //空
cancel:
                //cancel幂等校验：isExistCancel(xid);
                //cancel空回滚处理：!isExistTry(xid); # try没有执行，cancel不允许执行。
                //增加30元。

                //插入cancel执行记录，用户幂等判断。
```

TCC案例分析

优化方案：

账户B：

```
try:
        //空
confirm:
        //confirm幂等校验
        //正式增加30元
        //插入confirm执行记录，用于幂等判断。
cancel:

        //空
```

TCC常用框架

大道至简 悟在天成

Andy | 悟纤

TCC常用框架

以下数据采集于2019年11月20日

框架名称	Github地址	star数量
tcc-transaction	https://github.com/changmingxie/tcc-transaction	4.2k
Hmily	https://github.com/yu199195/hmily	2.6k
ByteTCC	https://github.com/liuyangming/ByteTCC	2.1k
EasyTransaction	https://github.com/QNJR-GROUP/EasyTransaction	1.9k

TCC常用框架

分布式事务研究以及功能测试相关结果（RPC、事务日志没有全部验证）：

框架名称	幂等性	嵌套调用	RPC框架支持	默认支持事务日志存储方式	可靠性验证
tcc-transaction	框架不支持	嵌套调用尝试失败	不耦合RPC框架，提供：dubbo、http的相关demo	DB、redis、zk、file	通过
Hmily	框架不支持	支持嵌套调用	dubbo、motan、springcloud，提供：dubbo相关demo	redis、mongodb、zk，file，DB	通过
ByteTCC	框架支持	嵌套调用尝试失败	dubbo、springcloud	file	通过
EasyTransaction	框架支持，业务可选择启用（框架支持情况下影响性能）	支持嵌套调用	Dubbo、SpringCloud、Ribbon/Eureka	DB、Redis	通过

我们在2PC讲到了Seata、tx-lcn对于TCC也是支持的，使用哪个框架就优先考虑使用哪个框架的TCC。

使用TX-LCN实战TCC分布式事务

大道至简 悟在天成

Andy | 悟纤

前言

TX-LCN对于TCC天然的支持，我们要学习TCC，有LCN的基础，学习TCC就很快，另外TX-LCN支持LCN+TCC并存。

TCC实现

改造lcn-tx-transfer-user-service1

只需要修改increse方法，编写try/confirm/cancel方法即可：

TCC实现

Try：

```
@Override
@Transactional
@TccTransaction(confirmMethod="confirmIncrese",cancelMethod="cancelIncrese")
public void increse(Long uid, BigDecimal money) {
    Account account = accountRepository.findByUid(uid);
    if(account == null) {
       throw new RuntimeException("Account not exist");
    }
    if (ERROR_USER_ID.equals(uid)) {
         throw new RuntimeException("account branch exception");
    }

}
```

TCC实现

Confirm：

```
public void confirmIncrese(Long uid, BigDecimal money) {
    Account account = accountRepository.findByUid(uid);
    account.setMoney(account.getMoney().add(money));
    accountRepository.save(account);

}
```

TCC实现

Cancel：

```
public void cancelIncrese(Long uid, BigDecimal money) {
        System.out.println("AccountServiceImpl.cancelIncrese()");
                Account account = accountRepository.findByUid(uid);
       //减少钱
       account.setMoney(account.getMoney().subtract(money));
       accountRepository.save(account);

}
```

运行测试：http://127.0.0.1:2222/transfer?money=10