# 分布式事务的总结与思考

随着业务的增长，数据、并发都在增长，一般来说数据库是最先出现的瓶颈。当单台数据库服务器不能支撑业务的时候，读写分离，双主多从、缓存，数据比较大的可以分表、分区，以上可以参看我之前的文章。

如果业务量再大，或者在架构上用到现在最流行的“微服务”，数据库分库就不可避免了。这就衍生出分布式事务这一问题来。

一个身边的小例子，支付宝转账到银行卡。我的支付宝扣了100，银行卡就得加100，两个操作都得成功，不能我支付宝扣了钱，银行卡没加钱，这就需要某种办法来避免这种错误的发生，最简单方便的，我们可以采用支付宝曾用过的解决方案——手工对账。

开玩笑了！

## 二阶段提交（2PC）

举个例子，有两位会计进行记账操作，A会计要在自家的账本上扣100，B会计要在自家账本上加100。A会计在北京，B会计在成都，如何知道对方操作成功了呢（保证一致性）？这时候就要多出来一位协调人C。

我们把“A-100，B+100”这个操作交给C，C开始协调A、B进行如下操作：

**第一阶段：**

C同时打电话给A、B：A/B啊，有个账要记一下，你们现在都有空吧？

A/B回：有空。

C对A：扣100。A这边就执行操作，并做记录“让我扣100”，做记录是为了失败回滚用。

C对B：加100。B这边也执行操作，并做记录“让我加100”。

A/B对C：我记好了。如果执行失败，就回“出事了，没记好”。注意，这时候虽然记录已经改变了，但是事务还未最终提交，资源还上着锁，还剩下签字画押这一步。

**第二阶段：**

C接到A/B都操作成功的回复，告诉A/B：大家都搞定了，这事OK了，签字画押吧。

A/B分别对自己的操作签字画押归档，整个事务完成。

如果有B回复“执行失败”，并没有加100，C就对所有人说：出事了，你们恢复原样吧。

A回滚，原来扣的100再加回去。

## 事务补偿（TCC）

如果我们采用2PC，遇上这种情况，就比较挠头了：A会计记账很快，一声令下马上就记好了。而B会计说我们记账麻烦，得走流程，要各级领导签字，通常情况下需要一天。这样一来A就且等着吧。

为了解决这个问题，就出现了TCC。

Try阶段：

A先检查账户的钱够不够扣100，如果够，冻结100。

B检查账户是不是可用。

Confirm阶段：

A把冻结的100减了，B增加100。如果某一个操作失败了，则重试。一般来说Try阶段保证了操作是可行的（A的钱够，并冻结了100；B的账户没有被冻结。），所以Confirm可以不断重试。

Cancel阶段：

在Try阶段，如果A冻结了100成功，但是B却发现账户以及被冻结了，没法增加，那么A久执行Cancel操作，把原来冻结的100再释放出来。

注意Confirm和Cancel需要实现幂等性，可参考我之前的文章。

一旦Try阶段都很顺利，那么A扣100执行的快，一旦执行成功就提交，接着忙你的去。不用等B。

B执行的慢，那就慢慢执行。

这个好处就是节省了A的资源，但是坏处就是数据的隔离性和一致性是保证不了的。

## 消息队列（MQ）

A写了一封给B的信，交给邮差，信的内容是：你要在自己的账户上加100。A告诉邮差，这封信你先拿着，我什么时候让你发，你再发给B。

A在自己的账户里扣100成功，告诉邮差，发吧。如果失败，告诉邮差，那信扔了吧。

B接到信，在自己账户里加了100。

A和B对账户的操作都是本地事务，所以很好处理，问题就出在邮差那。

1. 如果A扣了钱，但是通知邮差发信的时候出问题了，怎么办？

邮差应该有个定时检查消息的机制，过一段时间看一下手头上要发而未发的信，然后去问发信人，我手头上有你几封信，现在是发还是不发。

1. 邮差给B发信的时候出现了异常，比如B不在家收不到信，怎么办？这就要靠MQ的重试机制了，多送几次。如果送了好多次还是没法送给B，那就需要告诉A，你还是把你的操作回滚一下吧。

## 思考

多说几句个人的感受。一旦牵扯到分布式、微服务这个级别的系统，所谓的事务就是个伪命题。因为基于CAP原则，想实现ACID实在是不可能——你是个胖子，就别强求身轻如燕了（也不是没有这种情况，比如洪金宝，但是这世界有几个洪金宝呢？），应该追求BASE，在一段时间内达到数据的最终一致性就可以了。