本文是分布式事务系列文章的第二篇，我们将讨论TCC机制

基于消息的分布式事务处理机制

基于消息的分布式事务处理机制可以总结为如下两个阶段：

- 第一阶段：在事务发起方的本地事务中产生消息。在第一篇文章的例子中，是在transaction写入操作的同一个数据库事务中记录消息（消息入队），消息中包含了transaction的相关信息。
- 第二阶段：消息的消费方提供幂等操作来处理这些消息。这里也分为两种情况。一种是业务自身提供幂等操作，业务在处理时通过一些业务标识字段来判断某一条消息是否已经被处理过。另外一种情况，业务自身不支持幂等操作，则可以引入一个去重表来实现幂等性。在第一篇文章的例子中，就引入了updates_applied表来判断消息是否被处理过。

该机制具有如下特征：

- 消息的处理是异步的。意味着消息的生产方没有办法同步获取到消息的处理结果。这个特性限制了该机制只能适用于那些不要求各个业务方同步得到结果的情况。比如对于下单操作来说，用户积分的改动可能属于这类场景。
- 整个流程无回滚或者补偿机制。消息消费方的处理失败并不会导致消息生产方已提交操作的回滚。对于消费方来说，在碰到提交失败的情况，只能反复重试直到提交成功。

基于消息的机制总体来说比较简单。消息队列还能对消息起到削峰填谷的作用，所以不用担心分布式事务各个参与系统的性能是否匹配的问题。同时，该机制也不涉及回滚和补偿操作，总体来说实现成本较低。

基于补偿的分布式事务处理机制

有一些业务场景，要求各参与方能同步获取到执行结果。这类业务场景就不能用基于消息的机制了。

比如，用户下单时，我们需要对用户钱包余额和库存做扣减。扣减成功与否，将直接影响当前下单结果。对于这类业务场景，整个分布式事务成功与否，取决于所有参与方的执行结果。

对于这类业务场景，我们需要引入补偿机制。基于补偿机制的下单操作步骤如下：

1. 创建订单，订单状态为处理中。
2. 调用用户钱包服务，扣减用户钱包余额。
3. 调用库存服务，扣减库存。
4. 如果步骤2和步骤3都成功，则将订单状态改为成功。

如果第3步执行失败，则需要调用第2步用户钱包服务的补偿操作，将余额的扣减补偿回去。

相较于基于消息的机制来说，补偿机制需要给每个操作增加相应的补偿操作，从实现上来说更加复杂。

TCC模式

基于补偿的机制能很好地处理分布式事务各业务参与方同步获取执行结果的问题。但这个机制有一个问题，那就是会让用户看到即将被补偿的数据中间状态。

比如上面提到的下单流程，如果第2步用户钱包扣减成功了，但第3步库存扣减失败了，整个事务本来会被回滚。但用户却在用户钱包服务的补偿操作被成功调用之前，看到了被错误扣减后的余额。

这里我们引入TCC模式来解决这个问题：

- Try阶段：完成业务检查，预留业务资源。
- Confirm阶段：直接使用Try阶段预留的业务资源。
- Cancel阶段：取消Try阶段预留的业务资源。

我们继续用上面的下单操作来进一步解释TCC。

用户钱包表wallet字段如下：

```
wallet_id，钱包id
balance，余额
```

为了达到业务资源预留的目的，我们为用户钱包服务引入一张余额冻结表balance_frozen_record，字段如下：

```
record_id，冻结记录id
wallet_id，对应wallet.wallet_id
frozen_amount，冻结金额
state，有效状态， true为有效，false为无效
```

用户钱包服务提供的几个关键API接口如下：

- 余额查询接口（参数：wallet_id）。直接返回wallet表中的balance字段。
- 余额冻结接口（参数：amount和wallet_id）。对应于Try阶段。如果amount小于该钱包的可用余额，则冻结成功，往balance_frozen_record插入一条state=true的记录；如果amount大于可用余额，则冻结失败。可用余额等于wallet表中的balance字段减去 select sum（frozen_amount）from balance_frozen_record where wallet_*id = $wallet*_id and state = true。
- 取消冻结接口（参数：record_id）。对应于Cancel阶段。等幂操作。只需要将balance_frozen_record中对应的记录状态设置为false即可。

```
update balance_frozen_record set state = false where record_id = $record_id and state = true;
```

- 确认提交接口（参数：record_id）。对应于Confirm阶段。等幂操作。将record_id对应金额变动反映到wallet表balance中，同时将该记录状态设置为false。亦即：

```
begin transaction
 select frozen_amount, wallet_id from balance_frozen_record where record_id = $record_id and status = true;
 //$frozen_amount和$wallet_id是上面的select语句中查询出来的结果
```

```
4   if frozen_amount 和 wallet_id 有值（第一次调用）
5   update wallet set amount = amount - $frozen_amount where wallet_id = $wa
llet_id;
6   update balance_frozen_record set status = false where record_id = $recor
d_id;
7   end if
8  end transaction
```

TCC模式具有下面的特点：

- 未提交的中间状态将不再对用户可见。用户看到的余额不会反映这些中间状态。
- 需要所有业务实现预留接口。大部分业务都能被改造成预留的方式，但某些业务不能预留，比如是第三方公司提供的业务，那就没办法使用这种模式。

上面使用冻结表的方式，还带来了另外一个好处。也就是将对wallet表中balance的并发修改操作，变成了数据库插入操作。在某种程度上提高了系统的吞吐量。

参考：https://zhuanlan.zhihu.com/p/95799141