三阶段提交协议很好地解决了二阶段提交协议带来的同步阻塞问题，是一个非常有参考意义的解决方案。但是，极小概率的场景下可能会出现数据的不一致性。因为三阶段提交协议引入了超时机制，如果出现资源管理者（参与者）超时场景就会默认提交成功，如果其没有成功执行，或者其他资源管理者（参与者）出现回滚，那么就会出现数据的不一致性。

6.3 最终一致性解决方案

6.3.1 TCC 模式

二阶段提交协议和三阶段提交协议解决了分布式事务的问题，但在极端情况下仍然可能存在数据的不一致性，此外它对系统的开销比较大，引入事务管理者（协调者）后，比较容易出现单点瓶颈，以及在业务规模不断变大的情况下，系统可伸缩性也会存在问题。需要注意的是，它是同步操作，因此引入事务后，直到全局事务结束才能释放资源，性能可能是一个很大的问题。因此，在高并发场景下很少使用。阿里的技术人员提出了另外一种解决方案：TCC 模式。需要注意的是，很多读者把二阶段提交等同于二阶段提交协议，这是一个误区，事实上，TCC 模式也是一种二阶段提交。

TCC 模式将一个任务拆分三个操作：Try、Confirm 和 Cancel。假如，我们有一个 Func()方法，那么在 TCC 模式中，它就变成了 tryFunc()、confirmFunc()和 cancelFunc()三个方法。

```
tryFunc();
confirmFunc();
cancelFunc();
```

在 TCC 模式中，主业务服务负责发起流程，而从业务服务提供 TCC 模式的 Try、Confirm 和 Cancel 三个操作，还有一个事务管理器的角色负责控制事务的一致性。例如，我们现在有三个业务服务：交易服务、库存服务和支付服务。用户选商品，下订单，紧接着选择支付方式进行付款，针对这笔请求，交易服务会先调用库存服务扣库存，然后交易服务再调用支付服务进行相关的支付操作，最后支付服务会请求第三方支付平台创建交易并扣款。交易服务就是主业务服务，而库存服务和支付服务是从业务服务，如图 6-7 所示。

我们再来梳理 TCC 模式的流程。第一阶段主业务服务调用全部的从业务服务的 Try 操作，并且事务管理器记录操作日志。第二阶段，当全部从业务服务都成功时，再执行 Confirm 操作，否则会执行 Cancel 逆操作进行回滚。流程如图 6-8 所示。