分布式事务—2PC、3PC、TCC

数据库事务的4个特性ACID，原子性、一致性（事务前后的状态一致性）、隔离性（脏读/不可重复读/幻读）、持久性。

分布式事务是为了解决微服务架构（分布式系统）中不同节点之间的数据一致性问题。一个请求在多个微服务调用链中，所有服务的数据处理要么全部成功，要么全部回滚。

XA分布式事务协议包括两阶段提交（2PC）和三阶段提交（3PC）两种实现。

·**2PC（two-phase commit protocol）**是一个非常经典的强一致、中心化的原子提交协议。这里所说的中心化是指协议中有两类节点：一个是中心化协调者节点（coordinator）和N个参与者节点（partcipant）。

**第一阶段：请求/表决阶段**

分布式事务发起者发起请求调用协调者，协调者分别向多个参与者发起“事务预处理（prepare/vote\_request）”请求，参与者开启本地事务操作数据库，但在执行完成后并不会立马提交数据库本地事务，而是先向协调者报告，vote\_commit/ready。

**第二阶段：提交/执行阶段**

等所有参与者都向协调者报告了vote\_commit/ready，协调者会向所有参与者发送“全局提交确认通知（global\_commit）”，此时参与者节点就会完成自身本地数据库事务的提交，并最终将提交结果回复“ack”消息给协调者，然后协调者就会向调用方返回分布式事务处理完成的结果。

**第二阶段：提交/执行阶段（回滚）**

只要有参与者向协调者反馈“Vote\_Abort”的消息，此时分布式事务协调者就会向所有的参与者发起事务回滚的消息（“global\_rollback”），此时各个参与者节点就会回滚本地事务，释放资源，并且向协调者节点发送“ack”确认消息，协调者节点就会向调用方返回分布式事务处理失败的结果。

2PC存在的问题：

1. **性能问题。**执行过程中间，节点都处于阻塞状态。各个操作数据库的节点此时都占用着数据库资源，只有当所有节点准备完毕，事务协调者才会通知进行全局提交，参与者进行本地事务提交后才会释放资源。
2. **协调者单点故障问题。**一旦事务协调者节点挂掉，会导致参与者收不到提交或回滚的通知，从而导致参与者节点始终处于事务无法完成的中间状态。
3. **丢失消息导致的数据不一致问题。**在第二个阶段，如果发生局部网络问题，一部分事务参与者收到了提交消息，另一部分事务参与者没收到提交消息，那么就会导致节点间数据的不一致问题

**·三阶段提交（3PC）**，其在两阶段提交的基础上增加了CanCommit阶段，并引入了超时机制。一旦事务参与者迟迟没有收到协调者的Commit请求，就会自动进行本地commit，这样相对有效地解决了协调者单点故障的问题。

**第一阶段：CanCommit**

事务询问阶段，是否能完成本次事务，尝试获取数据库锁。所有参与者都返回yes后进入到阶段二。只要有一个参与者返回No或者超时，整个分布式事务就会中断，协调者就会向所有的参与者发送“abort”请求。

**第二阶段：PreCommit**

事务预提交，参与者接收到该请求后执行事务操作，并将Undo和Redo信息记录到事务日志中。参与者执行完事务操作后（此时属于未提交事务的状态），就会向协调者反馈“Ack”。所有参与者都返回ack才进入阶段三。如果有一个参与者未完成PreCommit的反馈或者反馈超时，那么协调者都会向所有的参与者节点发送abort请求，从而中断事务

**第三阶段：DoCommit**

参与者节点在收到提交请求后就会各自执行事务提交操作，并向协调者节点反馈“Ack”消息，协调者收到所有参与者的Ack消息后完成事务。

**·补偿事务TCC（Try-Confirm-Cancel）**

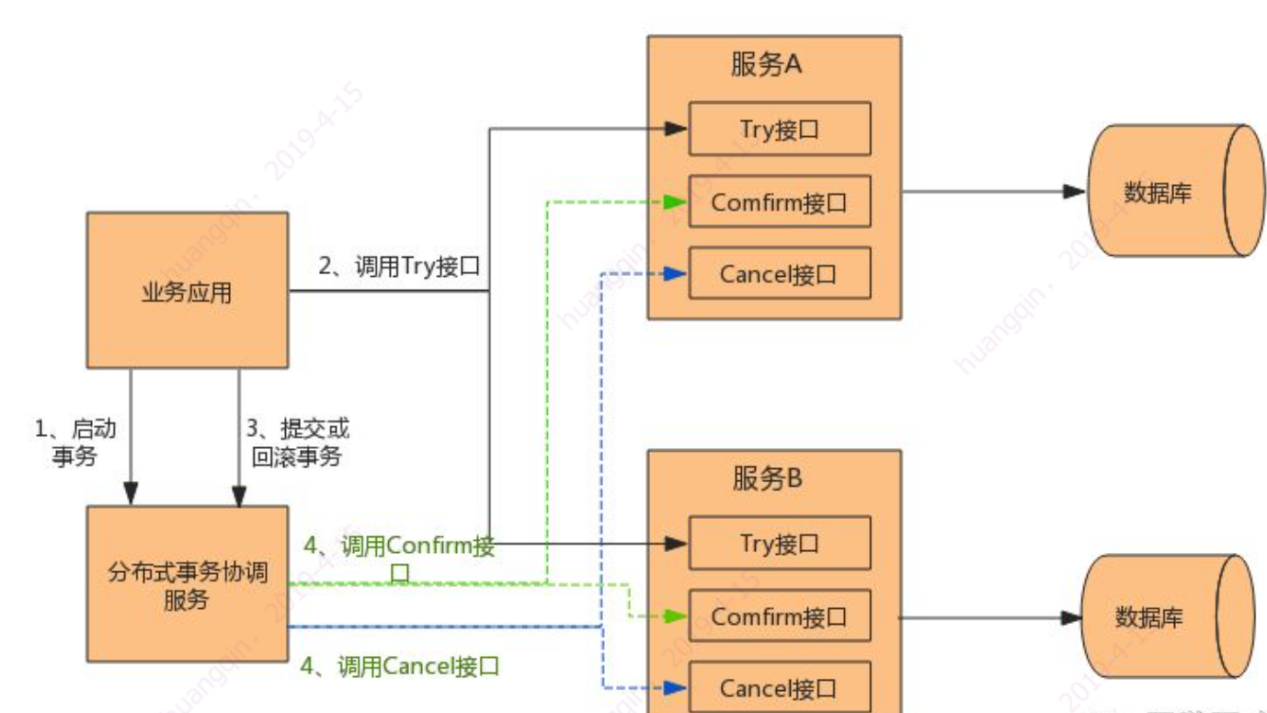
Try阶段：主要是对业务系统做检测及资源预留。

Confirm阶段：确认执行业务操作。

Cancel阶段：取消执行业务操作。

TCC事务的处理流程与2PC两阶段提交类似，不过2PC通常都是在跨库的DB层面，而TCC本质上就是一个应用层面的2PC，需要通过业务逻辑来实现。这种分布式事务的实现方式的优势在于，可以让应用**自己定义数据库操作的粒度，使降低锁冲突、提高吞吐量成为可能**。

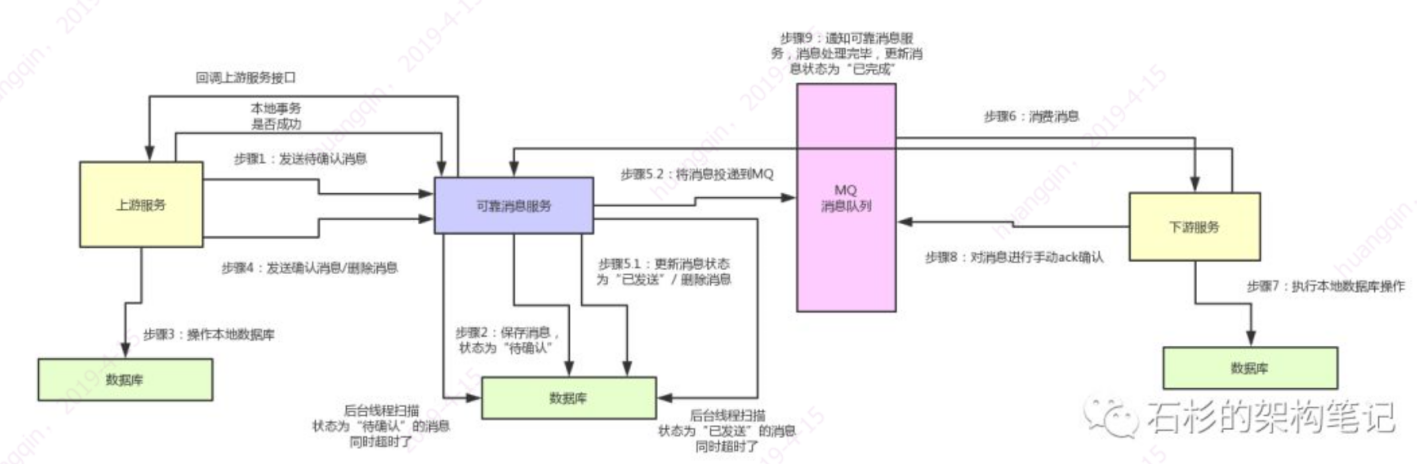
而不足之处则在于**对应用的侵入性非常强，业务逻辑的每个分支都需要实现try、confirm、cancel三个操作。此外，其实现难度也比较大，需要按照网络状态、系统故障等不同的失败原因实现不同的回滚策略。为了满足一致性的要求，confirm和cancel接口还必须实现幂等。**



**TCC适用于同步调用的业务。**

**·使用MQ实现异步调用服务的分布式事务。**

**可靠消息最终一致性。**



1. 上游服务发送一条消息到可靠消息服务，可靠消息服务将消息存储到数据库，状态为“待确认”。
2. 上游服务执行本地数据库操作，操作成功后向可靠消息服务发送确认消息，状态更新为“已发送”，同时将消息发送给MQ。【该两步操作是一个事务，如果投递MQ失败，本地消息状态回滚为“待确认”，等定时扫描程序发现该消息状态长时间为“待确认”，再回调上游服务，确认该消息已确认，继续改状态投递MQ】
3. 上游服务执行本地数据库操作失败后向可靠消息服务发送删除消息，则删除原消息。
4. 下游服务从MQ消费消息，如果操作本地数据库成功，则通知可靠消息服务，此时该消息状态会改为“已完成”。
5. 如果下游服务没有成功消费消息，等定时扫描程序发现该消息状态长时间为“已发送”，再次尝试重新投递到MQ，下游服务要保证幂等。