分布式事务了解吗？你们是如何解决分布式事务问题的？

题目难度：★★★★

知识点标签：分布式事务

学习时长：30分钟

题目描述

分布式事务了解吗？你们是如何解决分布式事务问题的？

解题思路

> 面试官问题可以从几个方面来回答：理解分布式事务，描述清楚具体项目中解决方案

分布式事务

事务的四大特性（ACID）：

原子性（Atomicity）：事务作为一个整体被执行，包含在其中的对数据库的操作要么全部被执行，要么都不执行。
一致性（Consistency）：事务应确保数据库的状态从一个一致状态转变为另一个一致状态。一致状态是指数据库中的数据应满足完整性约束。除此之外，一致性还有另外一层语义，就是事务的中间状态不能被观察到（这层语义也有说应该属于原子性）。
隔离性（Isolation）：多个事务并发执行时，一个事务的执行不应影响其他事务的执行，如同只有这一个操作在被数据库所执行一样。
持久性（Durability）：已被提交的事务对数据库的修改应该永久保存在数据库中。在事务结束时，此操作将不可逆转。

单机事务是通过将操作限制在一个会话内通过数据库本身的锁以及日志来实现ACID，那么分布式环境下该如何保证ACID特性呢？

当我们的单个数据库的性能产生瓶颈的时候，我们可能会对数据库进行分区，这里所说的分区指的是物理分区，分区之后可能不同的库就处于不同的服务器上了，这个时候单个数据库的ACID已经不能适应这种情况了，而在这种ACID的集群环境下，再想保证集群的ACID几乎是很难达到，或者即使能达到那么效率和性能会大幅下降，最为关键的是再很难扩展新的分区了，这个时候如果再追求集群的ACID会导致我们的系统变得很差，这时我们就需要引入一个新的理论原则来适应这种集群的情况，就是 CAP 原则或者叫CAP定理。

分布式事务解决方案

现在的分布式事务实现方案有多种，有些已经被淘汰，如基于XA的两段式提交、TCC解决方案，还有本地消息表、MQ事务消息，还有一些开源的事务中间件，如LCN、GTS。

1、基于XA的两阶段提交方案

XA 它包含两个部分：事务管理器和本地资源管理器。其中本地资源管理器往往由数据库实现，比如 Oracle、DB2 这些商业数据库都实现了 XA 接口，而事务管理器作为全局的协调者，负责各个本地资源的提交和回滚。

两阶段提交方案应用非常广泛，几乎所有商业OLTP (On-Line Transaction Processing)数据库都支持XA协议。但是两阶段提交方案开发复杂、锁定资源时间长，对性能影响很大，基本不适合解决微服务事务问题。

2、TCC解决方案

TCC方案在电商、金融领域落地较多。TCC方案其实是两阶段提交的一种改进。其将整个业务逻辑的每个分支显式的分成了Try、Confirm、Cancel三个操作。

- Try 阶段主要是对业务系统做检测及资源预留，完成业务的准备工作。
- Confirm 阶段主要是对业务系统做确认提交，Try阶段执行成功并开始执行 Confirm阶段时，默认Confirm阶段是不会出错的。即：只要Try成功，Confirm一定成功。
- Cancel 阶段主要是在业务执行错误，需要回滚的状态下执行的业务取消，预留资源释放。

事务开始时，业务应用会向事务协调器注册启动事务。之后业务应用会调用所有服务的try接口，完成一阶段准备。之后事务协调器会根据try接口返回情况，决定调用confirm接口或者cancel接口。如果接口调用失败，会进行重试。

微服务倡导服务的轻量化、易部署，而TCC方案中很多事务的处理逻辑需要应用自己编码实现，复杂且开发量大。

3、本地消息表 (异步确保)

本地消息表其实是国外的 ebay 搞出来的这么一套思想。

这个大概意思是这样的：

1. A 系统在自己本地一个事务里操作同时，插入一条数据到消息表；
2. 接着 A 系统将这个消息发送到 MQ 中去；
3. B 系统接收到消息之后，在一个事务里，往自己本地消息表里插入一条数据，同时执行其他的业务操作，如果这个消息已经被处理过了，那么此时这个事务会回滚，这样**保证不会重复处理消息**；
4. B 系统执行成功之后，就会更新自己本地消息表的状态以及 A 系统消息表的状态；
5. 如果 B 系统处理失败了，那么就不会更新消息表状态，那么此时 A 系统会定时扫描自己的消息表，如果有未处理的消息，会再次发送到 MQ 中去，让 B 再次处理；
6. 这个方案保证了最终一致性，哪怕 B 事务失败了，但是 A 会不断重发消息，直到 B 那边成功为止。

这个方案说实话最大的问题就在于**严重依赖于数据库的消息表来管理事务**啥的，会导致如果是高并发场景咋办呢？咋扩展呢？所以一般确实很少用。

4、MQ事务消息

直接基于 MQ 来实现事务，不再用本地的消息表。有一些第三方的MQ是支持事务消息的，比如RocketMQ，他们支持事务消息的方式也是类似于采用的二阶段提交，但是市面上一些主流的MQ都是不支持事务消息的，比如 RabbitMQ 和 Kafka 都不支持。

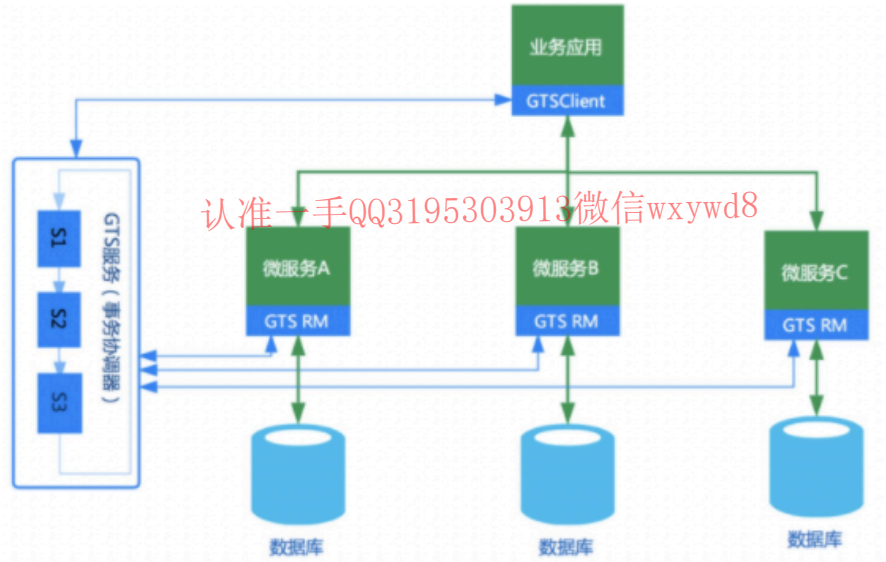
实现思想为：

- A 系统先发送一个 prepared 消息到 mq，如果这个 prepared 消息发送失败那么就直接取消操作别执行了；
- 如果这个消息发送成功过了，那么接着执行本地事务，如果成功就告诉 mq 发送确认消息，如果失败就告诉 mq 回滚消息；
- 如果发送了确认消息，那么此时 B 系统会接收到确认消息，然后执行本地的事务；
- mq 会自动**定时轮询**所有 prepared 消息回调你的接口，问你，这个消息是不是本地事务处理失败了，所有没发送确认的消息，是继续重试还是回滚？一般来说这里你就可以查下数据库看之前本地事务是否执行，如果回滚了，那么这里也回滚吧。这个就是避免可能本地事务执行成功了，而确认消息却发送失败了。
- 这个方案里，要是系统 B 的事务失败了咋办？重试咯，自动不断重试直到成功，如果实在是不行，要么就是针对重要的资金类业务进行回滚，比如 B 系统本地回滚后，想办法通知系统 A 也回滚；或者是发送报警由人工来手工回滚和补偿。

这种方案缺点就是实现难度大，而且主流MQ不支持。

5、分布式事务中间件解决方案

分布式事务中间件其本身并不创建事务，而是基于对本地事务的协调从而达到事务一致性的效果。典型代表有：阿里的GTS（https://www.aliyun.com/aliware/txc）、开源应用LCN。



总结

分布式事务本身是一个技术难题，是没有一种完美的方案应对所有场景的，具体还是要根据业务场景团队讨论选择。