你们项目中有遇到分布式事务问题，你是如何解决的呢？

回答：
1.单体项目架构中在多数据源的情况下 会发生 分布式事务问题
解决办法：jta+ Atomikos；
2.分布式在 RPC 远程调用过程中 分布式事务问题
在 RPC 接口调用的过程中 A 调用 B 服务接口之后，当 A 接口报错 无法回滚 B 接口的事务 导致最终 A 接口事务回滚了，B 接口事务没有回滚，需要解决分布式事务问题。
如果 A 调用 B 服务接口，如果 B 服务接口直接报错了，响应错误的状态码给 A 接口，
A 接口在根据该错误的状态码判断 如果等于 错误状态码 则直接手动回滚 A 接口中的事务。

分布式事务有哪些解决方案呢？

回答：
1.单体项目多数据源 可以 jta+ Atomikos；
2.基于 MQ 的形式解决 最终一致性的思想；
3.基于 RocketMQ 解决分布式事务 采用事务消息；
4.LCN 采用 LCN 模式 假关闭连接 （目前已经被淘汰） 官网已经无法访问； (基于 2PC)
5.Alibaba 的 Seata 背景非常强大，已经成为了主流 但是性能一般；(基于 2PC)
以上适合于在微服务架构中，不适合于和外部接口保证分布式事务问题。
6. 跨语言的方式实现解决分布式事务问题 类似于支付宝回调方式
如果项目是追求快速响应 建议采用 MQ 最终一致性方案 实现解决分布式事务问题。

2PC、3PC 应用场景？

学习到主流分布式事务解决框架
LCN、Seata

如何基于 MQ 解决分布式事务问题

核心思想就是最终一致性 短暂延迟这是允许。

1.生产者：必须确保消息投递到 MQ 成功；
Ack 消息确认机制
同步或者异步的形式
方式 1：Confirms
方式 2：事务消息
如果生产者投递消息失败的情况下，则通过日志记录下来 后期通过
定时任务自动补偿 投递 msg。

2.MQ 服务器端：需要将消息持久化，避免 MQ 宕机之后消息丢失; Mq 服务器端 在默认的情况下 都会对队列中的消息实现持久化
持久化硬盘。
刷盘同步（严格意义上保证消息不丢失）或者是异步 有可能会丢失。
3.消费者：必须确保消息消费成功（同时需要注意幂等性问题）;
3.1 在 rabbitmq 情况下：
必须要将消息消费成功之后，才会将该消息从 mq 服务器端中移除。
3.2 在 kafka 中的情况下：
不管是消费成功还是消费失败，该消息都不会立即从 mq 服务器端移除。手动提交 offset
如果消费者消费失败的情况下则 MQ 会采用间隔的形式不断重试重试 。
重试过程中需要解决幂等性问题
如何解决幂等问题：根据 msgid 作为全局 id 根据该全局 id 提前查询下该 数据是否已经插入了如果插入了不能够继续插入，db 层面根据该 msgid 创建一个唯一约束，防止 db 层面重复插入。
4.延迟问题：提高消费的速率
1.MQ 消费者批量消费
2.MQ 消费者集群消费
该流程环境 是不需要解决消息顺序一致性。

LCN 模式解决分布式事务原理

1. 发起方与参与方与我们的 LCN 管理器全局事务协调者一直保持长连接；
2. 发起方在调用接口之前会使用 Aop 生成一个全局的事务分组 id；
3. 发起方在调用之后的时候会在请求头中传递该全局事务分组 id；
4. 参与方从请求头中获取该事务分组 id，当前业务执行完毕之后不会提交该事务，则会使用假关闭。
5. 发起方调用接口完之后，如果出现异常的情况下，会通知给协调者回滚该事务，协调者在通知给参与方实现回滚事务

该模式存在的缺陷：
LCN 基于数据源假关闭 代理数据源 事务不会提交--

事务如果不提交的话 有可能会导致 行锁-

如果事务协调者宕机呢？如何解决-----事务协调者集群的
站在架构角度思考
支付状态----改成已经支付、 但是积分没有增加 允许 补偿的形式
支付状态----已经回滚了 未支付 但是积分增加---不允许的

如果所有协调者都宕机了，参与方会根据----设定超时机制
如果协调者没有及时给参与方发送提交还是回滚通知 则直接
认为超时 回滚 后期 就可以通过发起方结果 采用定时任务的形式
来进行 补偿。

Seata 模式解决分布式事务原理

seata 是一款开源的分布式事务解决方案，致力于提供高性能和简单易用的分布式事务服务，Seata 将为用户提供了 AT、TCC、SAGA 和 XA 事务模式，为用户打造一站式的分布式解决方案。
有阿里巴巴背书，该框架的活跃度最近几年活跃度非常高。
https://github.com/seata/seata seata 官网
https://yq.aliyun.com/zt/593075 阿里云 GTS

1.TM（发起方）连接到我们的 TC 事务协调者，创建一个全局的事务的 xid，保存到 ThreadLocal 中；
2.TM（发起方）和 RM(参与方)都被 Seata 的数据数据源实现代理，在原生的 sql 之前和之后保存原来和修改后日志到 undo_log 中，方便后期实现回滚。
3.TM（发起方）使 feign 客户端调用接口时候，在 ThreadLocal 中获取 xid，设置到请求头中；
4.RM(参与方)从请求中获取到该 xid，设置到 ThreadLoacl 中，同时也会向 seataserver 注册该分支事务。
5.TM(发起方)将当前本地事务的结果，告诉给协调者 TC，协调者 TC 在通知所有的分支是否回滚。
6. TM(发起方)如果调用接口成功之后抛出异常的情况下，告诉给协调者 TC，协调者 TC 在通知所有的分支根据根据全局的 xid 和分支事务的 id 查询分支数据源的 undo_log 日志表逆向生成 sql 语句实现回滚，同时删除对应的 undo_log 日志
7. TM(发起方)如果调用接口成功之后没有抛出任何的异常，告诉给协调者 TC，协调者 TC 在通知所有的分支根据根据全局的 xid 和分支事务的 id 查询分支数据源的 删除对应的 undo_log 日志表

At 模式存在缺陷：有可能存在 短暂脏读问题。

哪些场景写不建议使用 seata 解决分布

式事务

1. 如果我们的接口需要快速响应 避免超时问题 建议使用 mq 的模式
解决分布式事务问题，使用 seata 2pc 模式容易导致我们的接口超时。
2.不同语言之间需要保证数据的一致性问题，可以采用类似于支付宝异步回调的形式。