1. 基础概念

1.1 什么是事务

事务可以看做是一次大的活动，它由不同的小活动组成，这些活动要么全部成功，要么全部失败。

1.2 本地事务

在计算机系统中，更多的是通过关系型数据库来控制事务，这是利用数据库本身的事务特性来实现的，因此叫数据库事务，由于应用主要靠关系数据库来控制事务，而数据库通常和应用在同一个服务器，所以基于关系型数据库的事务又被称为本地事务。

数据库事务的四大特性：ACID

A（Atomic）：原子性，构成事务的所有操作，要么都执行完成，要么全部不执行，不可能出现部分成功部分失败的情况。

C（Consistency）：一致性，在事务执行前后，数据库的一致性约束没有被破坏。比如：张三向李四转 100 元，转账前和转账后的数据是正确状态这叫一致性，如果出现张三转出 100 元，李四账户没有增加 100 元这就出现了数 据错误，就没有达到一致性。

I（Isolation）：隔离性，数据库中的事务一般都是并发的，隔离性是指并发的两个事务的执行互不干扰，一个事务不能看到其他事务的运行过程的中间状态。通过配置事务隔离级别可以比避免脏读、重复读问题。

D（Durability）：持久性，事务完成之后，该事务对数据的更改会持久到数据库，且不会被回滚。

数据库事务在实现时会将一次事务的所有操作全部纳入到一个不可分割的执行单元，该执行单元的所有操作要么都成功，要么都失败，只要其中任一操作执行失败，都将导致整个事务的回滚。

2. 分布式事务基础理论

2.1 CAP理论

CAP 是 Consistency、Availability、Partition tolerance 三个单词的缩写，分别表示一致性、可用性、分区容忍性。

2.2 BASE 理论

1. Base 理论介绍

   BASE 是 Basically Available（基本可用）、Soft state（软状态）和 Eventually consistent （最终一致性）三个短语的缩写。BASE 理论是对 CAP 中 AP 的一个扩展，通过牺牲强一致性来获得可用性，当出现故障允许部分不可用但要保证核心功能可用，允许数据在一段时间内是不一致的，但最终达到一致状态。满足BASE理论的事务，我们称之为“**柔性事务**”。

- **基本可用**：分布式系统在出现故障时，允许损失部分可用功能，保证核心功能可用。如电商网站交易付款出现问题了，商品依然可以正常浏览。
- **软状态**：由于不要求强一致性，所以BASE允许系统中存在中间状态（也叫**软状态**），这个状态不影响系统可用性，如订单的"支付中"、“数据同步中”等状态，待数据最终一致后状态改为“成功”状态。
- **最终一致**：最终一致是指经过一段时间后，所有节点数据都将会达到一致。如订单的"支付中"状态，最终会变 为“支付成功”或者"支付失败"，使订单状态与实际交易结果达成一致，但需要一定时

间的延迟、等待。

3. 分布式事务解决方案之 2PC

前面学习了分布式事务的基础理论，以理论为基础，针对不同的分布式场景业界常见的解决方案有2PC、3PC、TCC、可靠消息最终一致性、最大努力通知这几种。

3.1 什么是 2PC

2PC 即两阶段提交协议，是将整个事务流程分为两个阶段，准备阶段（Prepare phase）、提交阶段（commit phase），2 是指两个阶段，P 是指准备阶段，C 是指提交阶段。

举例：张三和李四好久不见，老友约起聚餐，饭店老板要求先买单，才能出票。这时张三和李四分别抱怨近况不如意，囊中羞涩，都不愿意请客，这时只能AA。只有张三和李四都付款，老板才能出票安排就餐。但由于张三和李四都是铁公鸡，形成了尴尬的一幕：

准备阶段：老板要求张三付款，张三付款。老板要求李四付款，李四付款。

提交阶段：老板出票，两人拿票纷纷落座就餐。

例子中形成了一个事务，若张三或李四其中一人拒绝付款，或钱不够，店老板都不会给出票，并且会把已收款退回。

整个事务过程由事务管理器和参与者组成，店老板就是事务管理器，张三、李四就是事务参与者，事务管理器负责决策整个分布式事务的提交和回滚，事务参与者负责自己本地事务的提交和回滚。

在计算机中部分关系数据库如 Oracle、MySQL 支持两阶段提交协议，如下图：

1. 准备阶段（Prepare phase）：事务管理器给每个参与者发送 Prepare 消息，每个数据库参与者在本地执行事务，并写本地的 Undo/Redo 日志，此时事务没有提交。（**Undo 日志是记录修改前的数据，用于数据库回滚，Redo 日志是记录修改后的数据，用于提交事务后写入数据文件**）
2. 提交阶段（commit phase）：如果事务管理器收到了参与者的执行失败或者超时消息时，直接给每个参与者发送回滚（Rollback）消息；否则，发送提交（Commit）消息；参与者根据事务管理器的指令执行提交或者回滚操作，并释放事务处理过程中使用的锁资源。注意：**必须在最后阶段释放锁资源**。

3.2 解决方案

3.2.2 Seata 方案

Seata 是由阿里中间件团队发起的开源项目 Fescar，后更名为 Seata，它是一个是开源的分布式事务框架。

4. 分布式事务解决方案之TCC

4.1 什么是TCC事务

TCC 分为三个阶段：

1. **Try** 阶段是做完业务检查（一致性）及资源预留（隔离），此阶段仅是一个初步操作，它和后续的 Confirm 一起才能真正构成一个完整的业务逻辑。
2. **Confirm** 阶段是做确认提交，Try 阶段所有分支事务执行成功后开始执行 Confirm。通常情况下，采用 TCC 则认为 Confirm 阶段是不会出错的。即：只要 Try 成功，Confirm 一定成功。若 Confirm 阶段真的出错了，需引入重试机制或人工处理。
3. **Cancel** 阶段是在业务执行错误需要回滚的状态下执行分支事务的业务取消，预留资源释放。通常情况下，采用 TCC 则认为 Cancel 阶段也是一定成功的。若 Cancel 阶段真的出错了，需引入重试机制或人工处理。

5. 分布式事务解决方案之可靠消息最终一致性

5.1 什么是可靠消息最终一致性事务

可靠消息最终一致性方案是指当事务发起方执行完成本地事务后并发出一条消息，事务参与方（消息消费者）一定能够接收消息并处理事务成功，此方案强调的是只要消息发给事务参与方最终事务要达到一致。

此方案是利用消息中间件完成，如下图：

事务发起方（消息生产方）将消息发给消息中间件，事务参与方从消息中间件接收消息，事务发起方和消息中间件之间，事务参与方（消息消费方）和消息中间件之间都是通过网络通信，由于网络通信的不确定性会导致分布式事务问题。

因此可靠消息最终一致性方案要解决以下几个问题：

1. **本地事务与消息发送的原子性问题**

   本地事务与消息发送的原子性问题即：事务发起方在本地事务执行成功后消息必须发出去，否则就丢弃消息。即实现本地事务和消息发送的原子性，要么都成功，要么都失败。本地事务与消息发送的原子性问题是实现可靠消息最终一致性方案的关键问题。

   下面这种操作，先发送消息，在操作数据库：

   ```
   begin transaction;
       //1.发送MQ
       //2.数据库操作
   commit transation;
   ```

   这种情况下无法保证数据库操作与发送消息的一致性，因为可能发送消息成功，数据库操作失败。那么第二种方案，先进行数据库操作，再发送消息：

   ```
   begin transaction;
       //1.数据库操作
       //2.发送MQ
   commit transation;
   ```

   这种情况下貌似没有问题，如果发送 MQ 消息失败，就会抛出异常，导致数据库事务回滚。但如果是超时异常，数据库回滚，但 MQ 其实已经正常发送了，同样会导致不一致。

2. **事务参与方接收消息的可靠性**

   事务参与方必须能够从消息队列接收到消息，如果接收消息失败可以重复接收消息。

3. **消息重复消费的问题**

   由于网络2的存在，若某一个消费节点超时但是消费成功，此时消息中间件会重复投递此消息，就导致了消息的重复消费。

   要解决消息重复消费的问题就要实现事务参与方的方法幂等性。

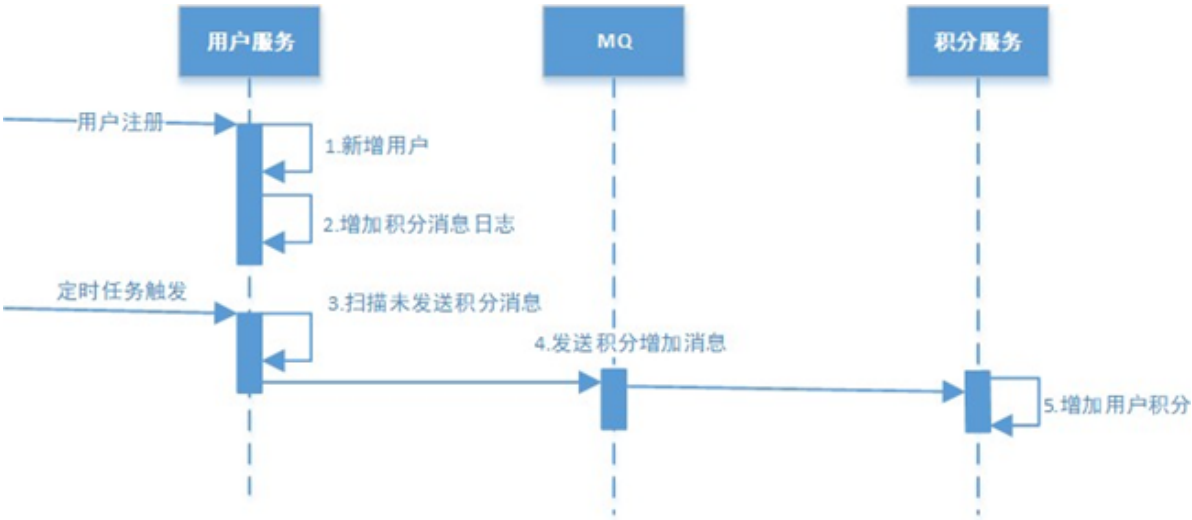
5.2 解决方案

上节讨论了可靠消息最终一致性事务方案需要解决的问题，本节讨论具体的解决方案。

5.2.1 本地消息表方案

本地消息表这个方案最初是 eBay 提出的，此方案的核心是通过本地事务保证数据业务操作和消息的一致性，然后通过定时任务将消息发送至消息中间件，待确认消息发送给消费方成功再将消息删除。

下面以注册送积分为例来说明：下例共有两个微服务交互，用户服务和积分服务，用户服务负责添加用户，积分服务负责增加积分。



交互流程如下：

1. **用户注册**

   用户服务在本地事务新增用户和增加 "积分消息日志"。（用户表和消息表通过本地事务保证一致）

   ```
   begin transaction
       //1.新增用户
       //2.存储积分消息日志
   commit transation
   ```

   这种情况下，本地数据库操作与存储积分消息日志处于同一个事务中，本地数据库操作与记录消息日志操作具备原子性。

2. **定时任务扫描日志**

   如何保证将消息发送给消息队列呢?

   经过第一步消息已经写到消息日志表中，可以启动独立的线程，定时对消息日志表中的消息进行扫描并发送至消息中间件，在消息中间件反馈发送成功后删除该消息日志，否则等待定时任务下一周期重试。

3. **消费消息**

   如何保证消费者一定能消费到消息呢?

   这里可以使用 MQ 的 ack（即消息确认）机制，消费者监听 MQ，如果消费者接收到消息并且业务处理完成后向 MQ 发送 ack（即消息确认），此时说明消费者正常消费消息完成，MQ 将不再向消费者推送消息，否则消费者会不断重试向消费者来发送消息。

   积分服务接收到"增加积分"消息，开始增加积分，积分增加成功后向消息中间件回应 ack，否则消息中间件将重复投递此消息。

   由于消息会重复投递，积分服务的"增加积分"功能需要实现幂等性。

5.3 小结

可靠消息最终一致性就是保证消息从生产方经过消息中间件传递到消费方的一致性：

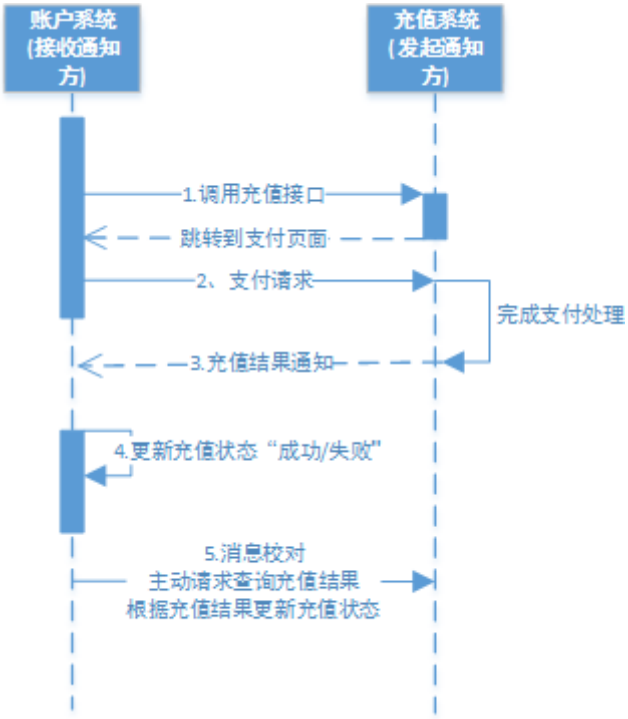
1. 本地事务与消息发送的原子性问题。
2. 事务参与方接收消息的可靠性。

可靠消息最终一致性事务适合执行周期长且实时性要求不高的场景。引入消息机制后，同步的事务操作变为基于消息执行的异步操作，避免了分布式事务中的同步阻塞操作的影响，并实现了两个服务的解耦。

6 分布式事务解决方案之最大努力通知

6.1 什么是最大努力通知

最大努力通知也是一种解决分布式事务的方案，下边是一个是充值的例子：



交互流程：

1. 账户系统调用充值系统接口
2. 充值系统完成支付处理向账户发起充值结果通知，若通知失败，则充值系统按策略进行重复通知
3. 账户系统接收到充值结果通知修改充值状态
4. 账户系统未接收到通知会主动调用充值系统的接口查询充值结果

通过上边的例子我们总结最大努力通知方案的目标：**发起通知方通过一定的机制最大努力将业务处理结果通知到接收方。**

具体包括：

1. 有一定的消息重复通知机制。因为接收通知方可能没有接收到通知，此时要有一定的机制对消息重复通知
2. 消息校对机制。如果尽最大努力也没有通知到接收方，或者接收方消费消息后要再次消费，此时可由接收方主动向通知方查询消息信息来满足需求。

最大努力通知与可靠消息一致性有什么不同？

1. **解决方案思想不同**

可靠消息一致性，发起通知方需要保证将消息发出去，并且将消息发到接收通知方，消息的可靠性关键由发起通知方来保证。最大努力通知，发起通知方尽最大的努力将业务处理结果通知为接收通知方，但是可能消息接收不到，此时需要接 收通知方主动调用发起通知方的接口查询业务处理结果，通知的可靠性关键在接收通知方。

2. **两者的业务应用场景不同**

   可靠消息一致性关注的是交易过程的事务一致，以异步的方式完成交易。最大努力通知关注的是交易后的通知事务，即将交易结果可靠的通知出去。

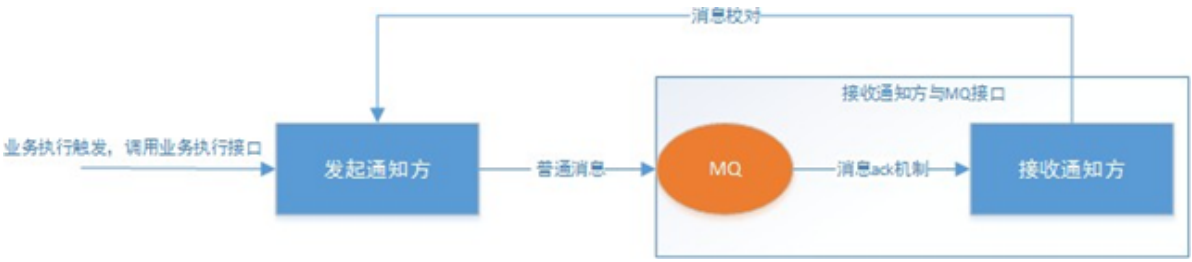
3. **技术解决方向不同**

   可靠消息一致性要解决消息从发出到接收的一致性，即消息发出并且被接收到。最大努力通知无法保证消息从发出到接收的一致性，只提供消息接收的可靠性机制。可靠机制是，最大努力的将消息通知给接收方，当消息无法被接收方接收时，由接收方主动查询消息（业务处理结果）

6.2 解决方案

通过对最大努力通知的理解，采用 MQ 的 ack 机制就可以实现最大努力通知。

方案1：



本方案是利用 MQ 的 ack 机制由 MQ 向接收通知方发送通知，流程如下：

1. 发起通知方将通知发给 MQ。使用普通消息机制将通知发给MQ。

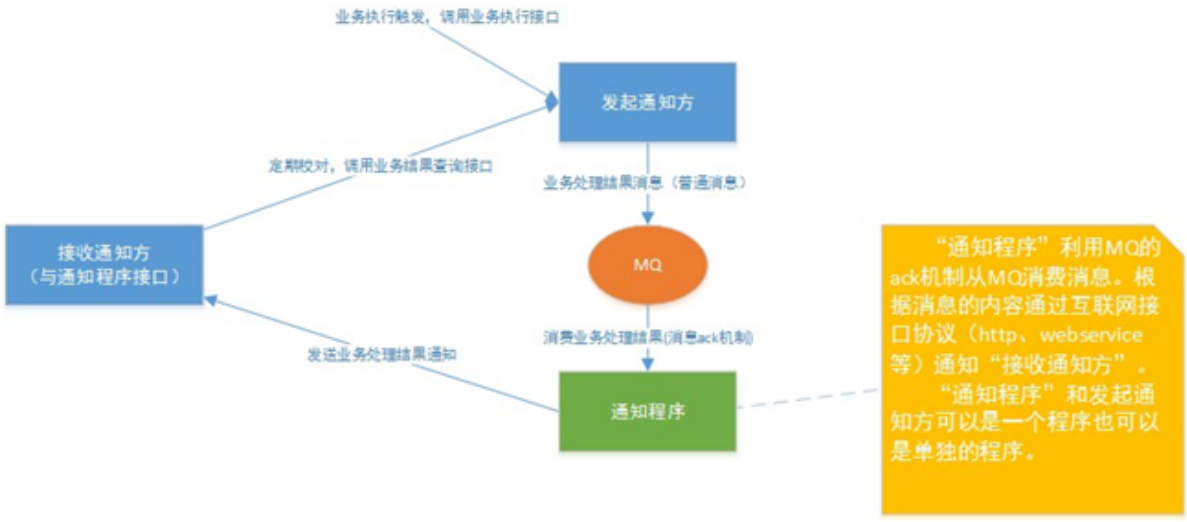
   注意：**如果消息没有发出去可由接收通知方主动请求发起通知方查询业务执行结果。**（后边会讲）

2. 接收通知方监听 MQ。
3. 接收通知方接收消息，业务处理完成回应 ack。
4. 接收通知方若没有回应 ack 则 MQ 会重复通知。

   MQ会按照间隔 1min、5min、10min、30min、1h、2h、5h、10h的方式，逐步拉大通知间隔，直到达到通知要求的时间窗口上限。

5. 接收通知方可通过消息校对接口来校对消息的一致性。

方案 2：



交互流程如下：

1. 发起通知方将消息发给 MQ。

   使用可靠消息一致方案中的事务消息保证本地事务和消息的原子性，最终将通知先发给 MQ。

2. 通知程序监听 MQ，接收 MQ 的消息。

   方案 1 中接收通知方直接监听 MQ，方案 2 中由通知程序监听 MQ。

   通知程序若没有回应 ack 则 MQ 会重复通知。

3. 通知程序通过互联网接口协议（如 http、webservice）调用接收通知方案接口，完成通知。

   通知程序调用接收通知方案接口成功就表示通知成功，即消费 MQ 消息成功，MQ 将不再向通知程序投递通知消息。

4. 接收通知方可通过消息校对接口来校对消息的一致性。

方案1和方案2的不同点：

1. 方案 1 中接收通知方与 MQ 接口，即接收通知方案监听 MQ，此方案主要应用与内部应用之间的通知。
2. 方案 2 中由通知程序与 MQ 接口，通知程序监听 MQ，收到 MQ 的消息后由通知程序通过互联网接口协议调用接收通知方。此方案主要应用于外部应用之间的通知，例如支付宝、微信的支付结果通知。

6.3 小结

最大努力通知方案是分布式事务中对一致性要求最低的一种，适用于一些最终一致性时间敏感度低的业务；最大努力通知方案需要实现如下功能：

1. 消息重复通知机制
2. 消息校对机制

7 总结

分布式事务对比分析

2PC 最大的诟病是一个阻塞协议。RM 在执行分支事务后需要等待 TM 的决定，此时服务会阻塞并锁定资源。由于其阻塞机制和最差时间复杂度高，因此，这种设计不能适应随着事务涉及的服务数量增加而扩展的需要，很难用于并发较高以及子事务生命周期较长（long-running transactions） 的分布式服务中。

如果拿**TCC**事务的处理流程与2PC两阶段提交做比较，2PC 通常都是在跨库的 DB 层面，而 TCC 则在应用层面的处理，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**应用自己定义数据操作的粒度，使得降低锁冲突、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。典型的使用场景：满减，登录送优惠券等。

可靠消息最终一致性事务适合执行周期长且实时性要求不高的场景。引入消息机制后，同步的事务操作变为基于消息执行的异步操作, 避免了分布式事务中的同步阻塞操作的影响，并实现了两个服务的解耦。典型的使用场景：注册送积分，登录送优惠券等。

最大努力通知是分布式事务中要求最低的一种,适用于一些最终一致性时间敏感度低的业务；允许发起通知方处理业务失败，在接收通知方收到通知后积极进行失败处理，无论发起通知方如何处理结果都会不影响到接收通知方的后续处理；发起通知方需提供查询执行情况接口，用于接收通知方校对结果。典型的使用场景：银行通知、支付结果通知等。

	2PC	TCC	可靠消息	最大努力通知
一致性	强一致性	最终一致性	最终一致性	最终一致性
吞吐量	低	中	高	高
实现复杂度	易	难	中	易

总结

在条件允许的情况下，我们尽可能选择本地事务单数据源，因为它减少了网络交互带来的性能损耗，且避免了数据弱一致性带来的种种问题。若某系统频繁且不合理的使用分布式事务，应首先从整体设计角度观察服务的拆分是否 合理，是否高内聚低耦合？是否粒度太小？分布式事务一直是业界难题，因为网络的不确定性，而且我们习惯于拿分布式事务与单机事务 ACID 做对比。

无论是数据库层的 XA、还是应用层 TCC、可靠消息、最大努力通知等方案，都没有完美解决分布式事务问题，它们不过是各自在性能、一致性、可用性等方面做取舍，寻求某些场景偏好下的权衡。