分布式事务

1、什么是分布式系统?

部署在不同结点上的系统通过网络交互来完成协同工作的系统。

比如：充值加积分的业务，用户在充值系统向自己的账户充钱，在积分系统中自己积分相应的增加。充值系统和积

分系统是两个不同的系统，一次充值加积分的业务就需要这两个系统协同工作来完成。

2、什么是事务?

事务是指由一组操作组成的一个工作单元，这个工作单元具有原子性（atomicity）、一致性（consistency）、隔 离性（isolation）和持久性（durability）。

原子性：执行单元中的操作要么全部执行成功，要么全部失败。如果有一部分成功一部分失败那么成功的操作要全 部回滚到执行前的状态。

一致性：执行一次事务会使用数据从一个正确的状态转换到另一个正确的状态，执行前后 数据都是完整的。

隔离性：在该事务执行的过程中，任何数据的改变只存在于该事务之中，对外界没有影响，事务 与事务之间是完全的隔离的。只有事务提交后其它事务才可以查询到最新的数据。

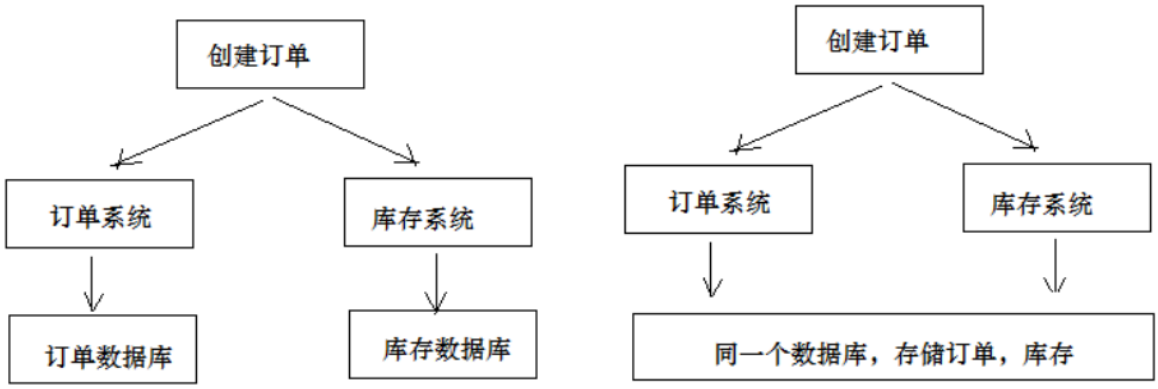
持久性：事务完成后对数据的改 变会永久性的存储起来，即使发生断电宕机数据依然在。

3、什么是本地事务?

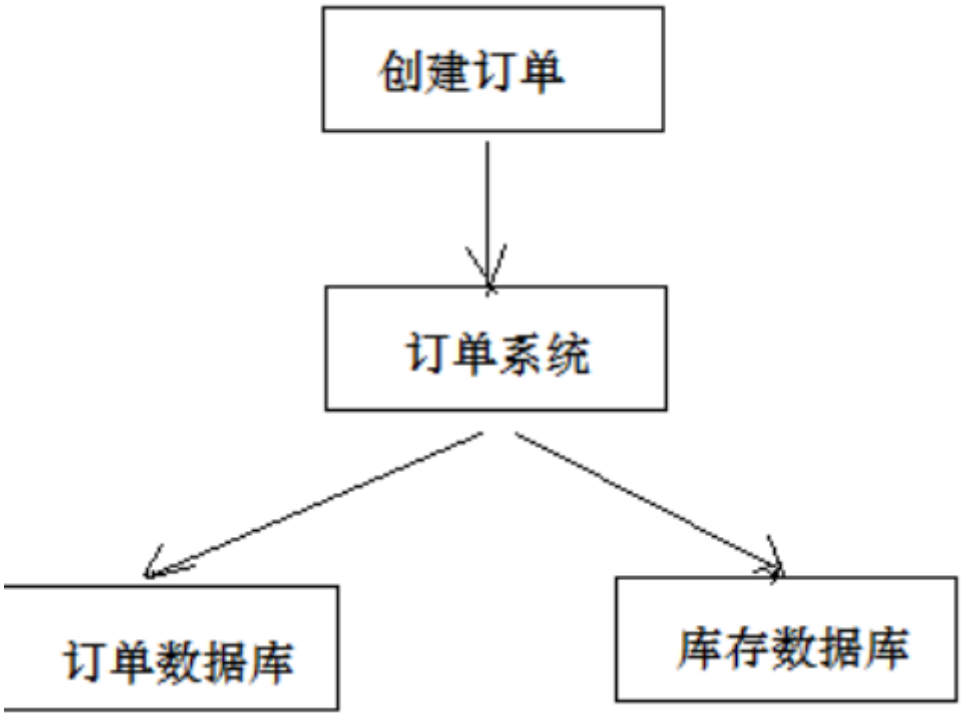
本地事务就是用关系数据库来控制事务，关系数据库通常都具有ACID特性，传统的单体应用通常会将数据全部存储在一个数据库中，会借助关系数据库来完成事务控制。

4、什么是分布式事务?

在分布式系统中一次操作由多个系统协同完成，这种一次事务操作涉及多个系统通过网络协同完成的过程称为分布式事务。这里强调的是多个系统通过网络协同完成一个事务的过程，并不强调多个系统访问了不同的数据库，即使 多个系统访问的是同一个数据库也是分布式事务，



另外一种分布式事务的表现是，一个应用程序使用了多个数据源连接了不同的数据库，当一次事务需要操作多个数 据源，此时也属于分布式事务，当系统作了数据库拆分后会出现此种情况。



5、分布式事务有哪些场景?

1) 电商系统中的下单扣库存

电商系统中，订单系统和库存系统是两个系统，一次下单的操作由两个系统协同完成

2）金融系统中的银行卡充值

在金融系统中通过银行卡向平台充值需要通过银行系统和金融系统协同完成。

3）教育系统中下单选课业务

在线教育系统中，用户购买课程，下单支付成功后学生选课成功，此事务由订单系统和选课系统协同完成。

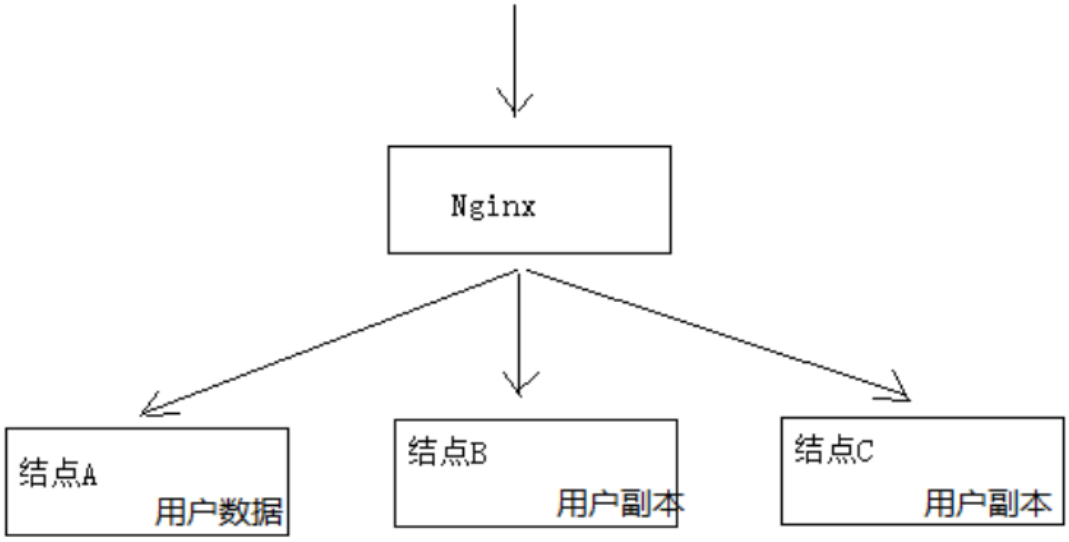
4） SNS系统的消息发送

在社交系统中发送站内消息同时发送手机短信，一次消息发送由站内消息系统和手机通信系统协同完成。

CAP理论

如何进行分布式事务控制？CAP理论是分布式事务处理的理论基础，了解了CAP理论有助于我们研究分布式事务的 处理方案。

CAP理论是：分布式系统在设计时只能在一致性(Consistency)、可用性(Availability)、分区容忍性(Partition Tolerance)中满足两种，无法兼顾三种



一致性(Consistency)：服务A、B、C三个结点都存储了用户数据， 三个结点的数据需要保持同一时刻数据一致 性。

可用性(Availability)：服务A、B、C三个结点，其中一个结点宕机不影响整个集群对外提供服务，如果只有服务A结 点，当服务A宕机整个系统将无法提供服务，增加服务B、C是为了保证系统的可用性。

分区容忍性(Partition Tolerance)：分区容忍性就是允许系统通过网络协同工作，分区容忍性要解决由于网络分区 导致数据的不完整及无法访问等问题。

分布式系统不可避免的出现了多个系统通过网络协同工作的场景，结点之间难免会出现网络中断、网延延迟等现 象，这种现象一旦出现就导致数据被分散在不同的结点上，这就是网络分区。

分布式系统能否兼顾C、A、P?

在保证分区容忍性的前提下一致性和可用性无法兼顾，如果要提高系统的可用性就要增加多个结点，如果要保证数 据的一致性就要实现每个结点的数据一致，结点越多可用性越好，但是数据一致性越差。

所以，在进行分布式系统设计时，同时满足“一致性”、“可用性”和“分区容忍性”三者是几乎不可能的。

CAP有哪些组合方式?

1、CA：放弃分区容忍性，加强一致性和可用性，关系数据库按照CA进行设计。

2、AP：放弃一致性，加强可用性和分区容忍性，追求最终一致性，很多NoSQL数据库按照AP进行设计。

说明：这里放弃一致性是指放弃强一致性，强一致性就是写入成功立刻要查询出最新数据。追求最终一致性是指允

许暂时的数据不一致，只要最终在用户接受的时间内数据 一致即可。

3、CP：放弃可用性，加强一致性和分区容忍性，一些强一致性要求的系统按CP进行设计，比如跨行转账，一次转 账请求要等待双方银行系统都完成整个事务才算完成。

说明：由于网络问题的存在CP系统可能会出现待等待超时，如果没有处理超时问题则整理系统会出现阻塞。

总结:

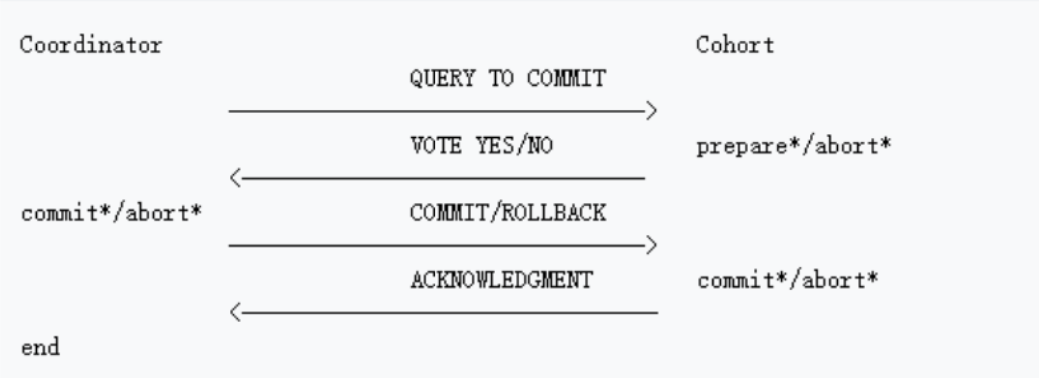
在分布式系统设计中AP的应用较多，即保证分区容忍性和可用性，牺牲数据的强一致性（写操作后立刻读取到最 新数据），保证数据最终一致性。比如：订单退款，今日退款成功，明日账户到账，只要在预定的用户可以接受的 时间内退款事务走完即可。

解决方案

两阶段提交协议(2PC)

为解决分布式系统的数据一致性问题出现了两阶段提交协议（2 Phase Commitment Protocol），两阶段提交由 协调者和参与者组成，共经过两个阶段和三个操作，部分关系数据库如Oracle、MySQL支持两阶段提交协议，本节 讲解关系数据库两阶段提交协议。

2PC协议流程图：



1）第一阶段：准备阶段（prepare）

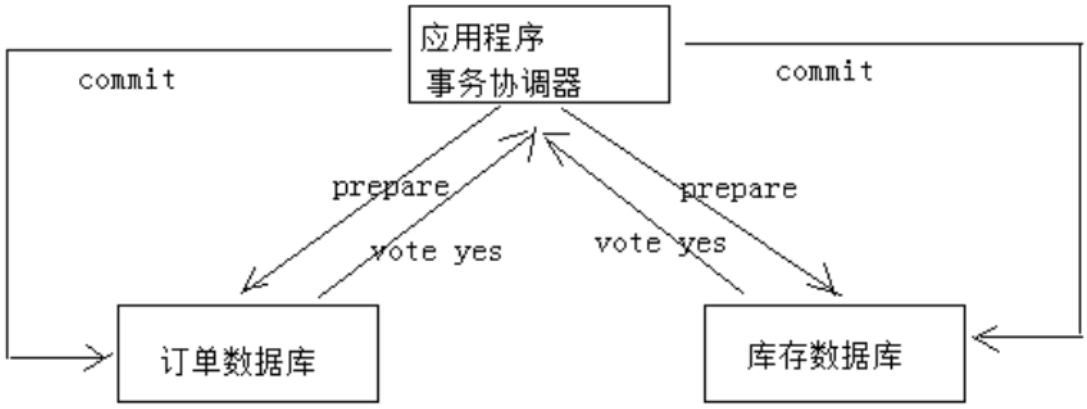
协调者通知参与者准备提交订单，参与者开始投票。

协调者完成准备工作向协调者回应Yes。

2）第二阶段：提交(commit)/回滚(rollback)阶段

协调者根据参与者的投票结果发起最终的提交指令。

如果有参与者没有准备好则发起回滚指令。



1、应用程序连接两个数据源。

2、应用程序通过事务协调器向两个库发起prepare，两个数据库收到消息分别执行本地事务（记录日志），但不提 交，如果执行成功则回复yes，否则回复no。

3、事务协调器收到回复，只要有一方回复no则分别向参与者发起回滚事务，参与者开始回滚事务。
4、事务协调器收到回复，全部回复yes，此时向参与者发起提交事务。如果参与者有一方提交事务失败则由事务协 调器发起回滚事务。

2PC的优点：实现强一致性，部分关系数据库支持（Oracle、MySQL等）。

缺点：整个事务的执行需要由协调者在多个节点之间去协调，增加了事务的执行时间，性能低下。

解决方案有：springboot+Atomikos or Bitronix

事务补偿（TCC）

TCC事务补偿是基于2PC实现的业务层事务控制方案，它是Try、Confifirm和Cancel三个单词的首字母，含义如下：

1、Try 检查及预留业务资源

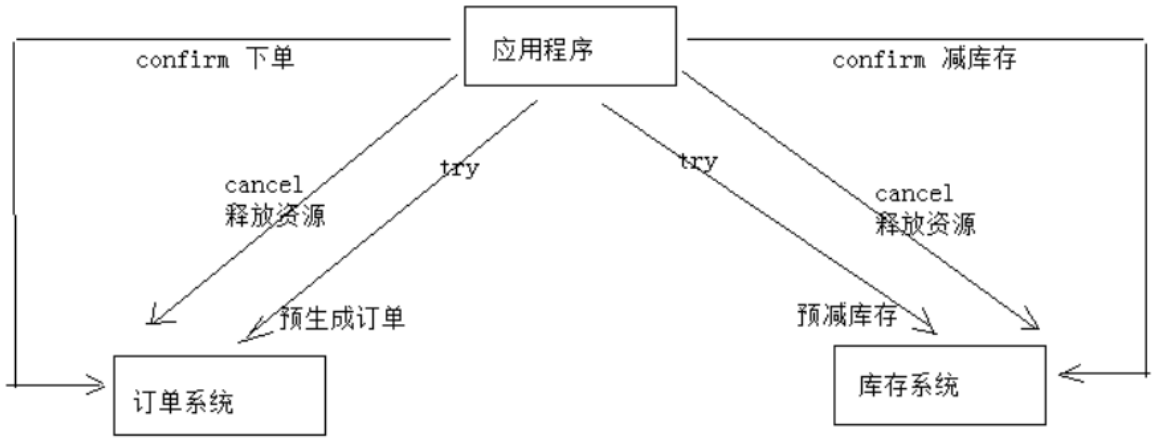
完成提交事务前的检查，并预留好资源。

2、Confifirm 确定执行业务操作

对try阶段预留的资源正式执行。

3、Cancel 取消执行业务操作

对try阶段预留的资源释放。



1、Try

下单业务由订单服务和库存服务协同完成，在try阶段订单服务和库存服务完成检查和预留资源。

订单服务检查当前是否满足提交订单的条件（比如：当前存在未完成订单的不允许提交新订单）。

库存服务检查当前是否有充足的库存，并锁定资源。

2、Confifirm

订单服务和库存服务成功完成Try后开始正式执行资源操作。

订单服务向订单写一条订单信息。

库存服务减去库存。

3、Cancel

如果订单服务和库存服务有一方出现失败则全部取消操作。

订单服务需要删除新增的订单信息。

库存服务将减去的库存再还原。

优点：最终保证数据的一致性，在业务层实现事务控制，灵活性好。

缺点：开发成本高，每个事务操作每个参与者都需要实现try/confifirm/cancel三个接口。

注意：TCC的try/confifirm/cancel接口都要实现幂等性，在为在try、confifirm、cancel失败后要不断重试。

什么是幂等性?

幂等性是指同一个操作无论请求多少次，其结果都相同。

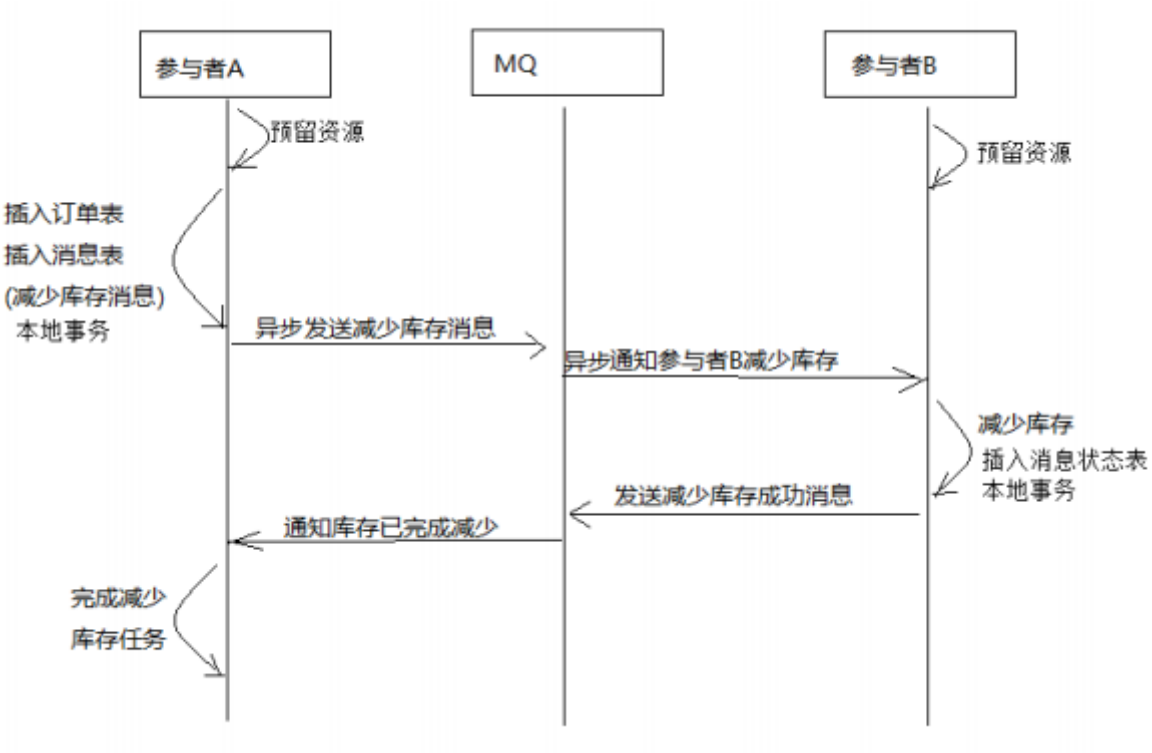
幂等操作实现方式有：

1、操作之前在业务方法进行判断如果执行过了就不再执行。

2、缓存所有请求和处理的结果，已经处理的请求则直接返回结果。北京市昌平区建材城西路金燕龙办公楼一层 电话：400-618-9090

3、在数据库表中加一个状态字段（未处理，已处理），数据操作时判断未处理时再处理。

消息队列实现最终一致



1、订单服务和库存服务完成检查和预留资源。

2、订单服务在本地事务中完成添加订单表记录和添加“减少库存任务消息”。

3、由定时任务根据消息表的记录发送给MQ通知库存服务执行减库存操作。

4、库存服务执行减少库存，并且记录执行消息状态（为避免重复执行消息，在执行减库存之前查询是否执行过此

消息）。

5、库存服务向MQ发送完成减少库存的消息。

6、订单服务接收到完成库存减少的消息后删除原来添加的“减少库存任务消息”。

实现最终事务一致要求：预留资源成功理论上要求正式执行成功，如果执行失败会进行重试，要求业务执行方法实

现幂等。

优点：

由MQ按异步的方式协调完成事务，性能较高。

不用实现try/confifirm/cancel接口，开发成本比TCC低。

缺点：

此方式基于关系数据库本地事务来实现，会出现频繁读写数据库记录，浪费数据库资源，另外对于高并发操作不是

最佳方案。