# 参考

正本清源-分布式事务之Seata

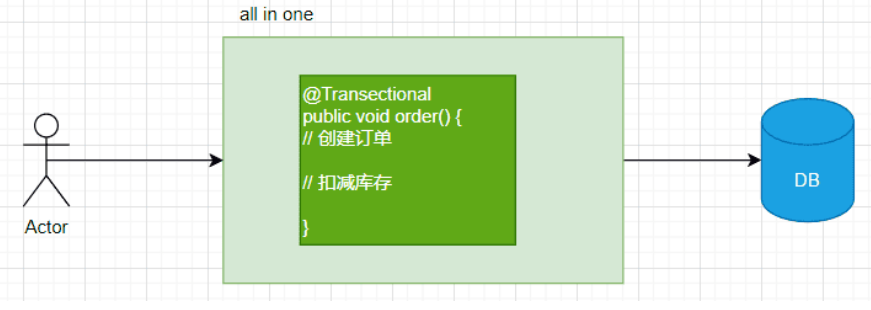
https://pdai.tech/md/arch/arch-z-transection.html

# 理论

**场景：**下单减库存，涉及到订单创建、库存扣减

**单体场景：**

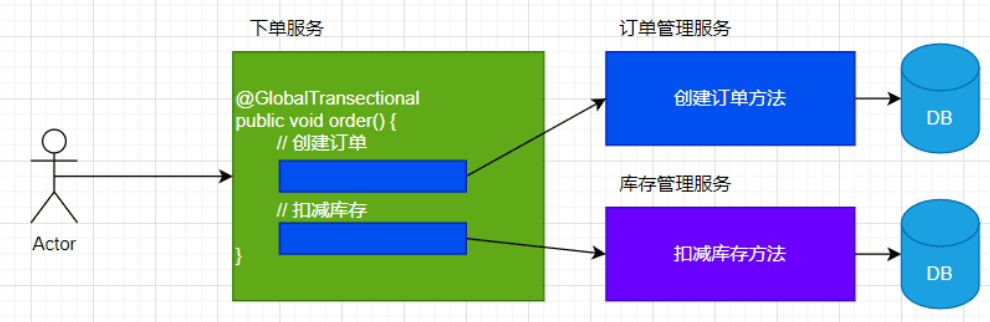
使用同一个数据库，借助本地事务保证，如spring的@Transactional注解



**分布式场景**：

独立的订单中心，库存中心，各自单独的数据库。创建订单和扣减库存，需要同时对订单DB和库存DB进行操作。两步操作必须同时成功，否则就会造成业务混乱，可此时我们只能保证自己服务的数据一致性，无法保证调用其他服务的操作是否成功，所以为了保证整个下单流程的数据一致性，就需要分布式事务介入。

如下GlobalTransactional注解采用阿里Seata的AT模式下的注解



## CAP/BASE

分布式的理论基础是CAP，由于P(分区容错）是必选项，所以只能在AP或者CP中选择。

* **分布式理论的CP -> 刚性事务**

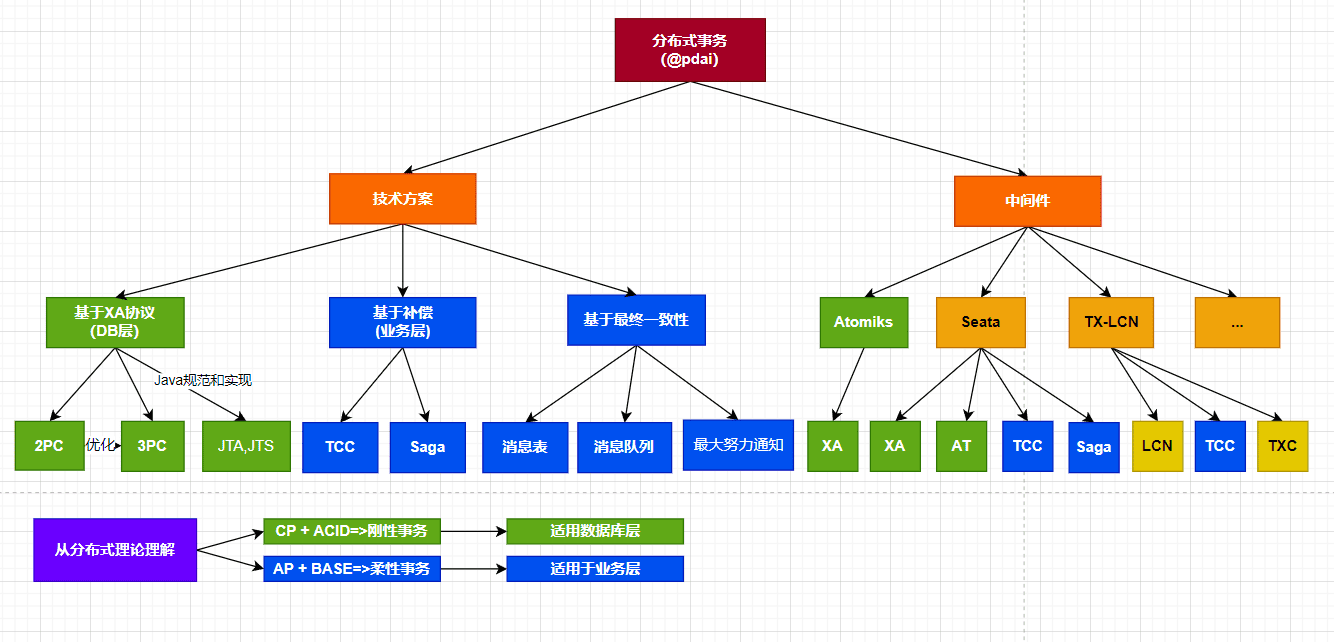
遵循ACID，对数据要求强一致性

* **分布式理论的AP+BASE -> 柔性事务**

遵循BASE，允许一定时间内不同节点的数据不一致，但要求最终一致；

BASE 理论是对 CAP 中的一致性和可用性进行一个权衡的结果，理论的核心思想就是：我们无法做到强一致，但每个应用都可以根据自身的业务特点，采用适当的方式来使系统达到最终一致性

## 分布式事务知识体系



· **刚性事务**：分布式理论的CP，遵循ACID，对数据要求强一致性。

· **XA协议** 是一个基于数据库层面的分布式事务协议，其分为两部分：**事务管理器（Transaction Manager）和本地资源管理器（Resource Manager）**。事务管理器作为一个全局的调度者，负责对各个本地资源管理器统一号令提交或者回滚。主流的诸如Oracle、MySQL等数据库均已实现了XA接口。

* + **二阶提交协议（2PC）**: 根据XA协议衍生出来而来; 引入一个作为协调者的组件来统一掌控所有参与者的操作结果并最终指示这些节点是否要把操作结果进行真正的提交; 参与者将操作成败通知协调者，再由协调者根据所有参与者的反馈情报决定各参与者是否要提交操作还是中止操作。所谓的两个阶段是指：第一阶段：准备阶段 (投票阶段) 和第二阶段：提交阶段（执行阶段）
  + **三阶提交协议（3PC）**: 是对两段提交（2PC）的一种升级优化，**3PC在2PC的第一阶段和第二阶段中插入一个准备阶段**。保证了在最后提交阶段之前，各参与者节点的状态都一致。同时在协调者和参与者中都引入超时机制，当参与者各种原因未收到协调者的commit请求后，会对本地事务进行commit，不会一直阻塞等待，解决了2PC的单点故障问题，但3PC还是没能从根本上解决数据一致性的问题。
* **Java事务规范**
  + **JTA**：Java事务API（Java Transaction API）是一个Java企业版的应用程序接口，在Java环境中，允许完成跨越多个XA资源的分布式事务。
  + **JTS**：Java事务服务（Java Transaction Service）是J2EE平台提供了分布式事务服务的具体实现规范，j2ee服务器提供商根据JTS规范实现事务并提供JTA接口。

· **柔性事务**：分布式理论的AP，遵循BASE，允许一定时间内不同节点的数据不一致，但要求最终一致。

· **基于业务层**

* + **TCC**: TCC（Try-Confirm-Cancel）又被称补偿事务，TCC与2PC的思想很相似，事务处理流程也很相似，但2PC是应用于在DB层面，TCC则可以理解为在应用层面的2PC，是需要我们编写业务逻辑来实现。
  + **SAGA**：Saga是由一系列的本地事务构成。每一个本地事务在更新完数据库之后，会发布一条消息或者一个事件来触发Saga中的下一个本地事务的执行。如果一个本地事务因为某些业务规则无法满足而失败，Saga会执行在这个失败的事务之前成功提交的所有事务的补偿操作。Saga的实现有很多种方式，其中最流行的两种方式是：基于事件的方式和基于命令的方式。
* **最终一致性**
  + **消息表**：本地消息表的方案最初是由 eBay 提出，核心思路是将分布式事务拆分成本地事务进行处理。
  + **消息队列**：基于 MQ 的分布式事务方案其实是对本地消息表的封装，将本地消息表基于 MQ 内部，其他方面的协议基本与本地消息表一致。
  + **最大努力通知**：最大努力通知也称为定期校对，是对MQ事务方案的进一步优化。它在事务主动方增加了消息校对的接口，如果事务被动方没有接收到消息，此时可以调用事务主动方提供的消息校对的接口主动获取。

## 刚性事务

XA接口是双向的系统接口，在事务管理器（Transaction Manager）以及一个或多个资源管理器（Resource Manager）之间形成通信桥梁。也就是说，在基于XA的一个事务中，我们可以针对多个资源进行事务管理，例如一个系统访问多个数据库，或即访问数据库、又访问像消息中间件这样的资源。这样我们就能够实现在多个数据库和消息中间件直接实现全部提交、或全部取消的事务。XA规范不是java的规范，而是一种通用的规范; Java 中的规范是JTA和JTS：Java事务API（Java Transaction API）是一个Java企业版的应用程序接口，在Java环境中，允许完成跨越多个XA资源的分布式事务；Java事务服务（Java Transaction Service）是J2EE平台提供了分布式事务服务的具体实现规范，j2ee服务器提供商根据JTS规范实现事务并提供JTA接口。

### 两阶段提交2PC

根据XA协议衍生出来而来; 简单而言：参与者（participant）用来管理资源，协调者（coordinator）用来协调事务状态

两段提交（2PC - Prepare & Commit）是指两个阶段的提交：

* **第一阶段: 准备阶段；** 
  + 协调者向所有参与者发送 REQUEST-TO-PREPARE
  + 当参与者收到REQUEST-TO-PREPARE 消息后, 它向协调者发送消息PREPARED或者NO，表示事务是否准备好；如果发送的是NO，那么事务要回滚；
* **第二阶段: 提交阶段。** 
  + 协调者收集所有参与者的返回消息, 如果所有的参与者都回复的是PREPARED， 那么协调者向所有参与者发送COMMIT 消息；否则，协调者向所有回复PREPARED的参与者发送ABORT消息；
  + 参与者如果回复了PREPARED消息并且收到协调者发来的COMMIT消息，或者它收到ABORT消息，它将执行提交或回滚，并向协调者发送DONE消息以确认。

#### 两段提交（2PC）的缺点

二阶段提交看似能够提供原子性的操作，但它存在着严重的缺陷：

* 网络抖动导致的数据不一致：第二阶段中协调者向参与者发送commit命令之后，一旦此时发生网络抖动，导致一部分参与者接收到了commit请求并执行，可其他未接到commit请求的参与者无法执行事务提交。进而导致整个分布式系统出现了数据不一致。
* 超时导致的同步阻塞问题：2PC中的所有的参与者节点都为事务阻塞型，当某一个参与者节点出现通信超时，其余参与者都会被动阻塞占用资源不能释放。
* 单点故障的风险：由于严重的依赖协调者，一旦协调者发生故障，而此时参与者还都处于锁定资源的状态，无法完成事务commit操作。虽然协调者出现故障后，会重新选举一个协调者，可无法解决因前一个协调者宕机导致的参与者处于阻塞状态的问题。

2PC只适用两个数据库（数据库实现了XA协议）之间；2PC有诸多问题和不便，在实践中一般很少使用

### 三阶段提交3PC

## 柔性事务

### TCC

### 本地消息表

### MQ事务方案（可靠消息事务）

### 最大努力通知

# 中间件Seata

### AT模式

### XA模式

### TCC模式

### Saga模式