

# 谷粒商城

本地事务与分布式事务



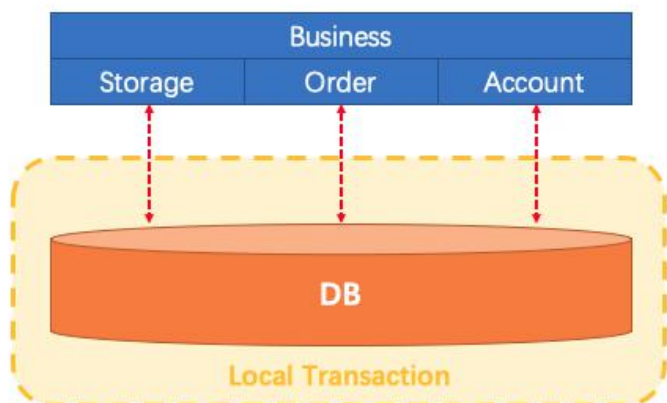
# 一、本地事务

## 1、事务的基本性质

数据库事务的几个特性：原子性(Atomicity)、一致性(Consistency)、隔离性或独立性(Isolation)和持久性(Durability)，简称就是 ACID；

- 原子性：一系列的操作整体不可拆分，要么同时成功，要么同时失败
- 一致性：数据在事务的前后，业务整体一致。
  - 转账。A:1000；B:1000； 转 200 事务成功； A: 800 B: 1200
- 隔离性：事务之间互相隔离。
- 持久性：一旦事务成功，数据一定会落盘在数据库。

在以往的单体应用中，我们多个业务操作使用同一条连接操作不同的数据表，一旦有异常，我们可以很容易的整体回滚；



Business：我们具体的业务代码

Storage：库存业务代码；扣库存

Order：订单业务代码；保存订单

Account：账号业务代码；减账户余额

比如买东西业务，扣库存，下订单，账户扣款，是一个整体；必须同时成功或者失败

一个事务开始，代表以下的所有操作都在同一个连接里面；

## 2、事务的隔离级别

- READ UNCOMMITTED（读未提交）

该隔离级别的事务会读到其它未提交事务的数据，此现象也称之为脏读。

- **READ COMMITTED**（读提交）

一个事务可以读取另一个已提交的事务，多次读取会造成不一样的结果，此现象称为不可重复读问题，Oracle 和 SQL Server 的默认隔离级别。

- **REPEATABLE READ**（可重复读）

该隔离级别是 MySQL 默认的隔离级别，在同一个事务里，select 的结果是事务开始时时间点的状态，因此，同样的 select 操作读到的结果会是一致的，但是，会有幻读现象。MySQL 的 InnoDB 引擎可以通过 next-key locks 机制（参考下文“行锁的算法”一节）来避免幻读。

- **SERIALIZABLE**（序列化）

在该隔离级别下事务都是串行顺序执行的，MySQL 数据库的 InnoDB 引擎会给读操作隐式加一把读共享锁，从而避免了脏读、不可重复读和幻读问题。

### 3、事务的传播行为

**1、PROPAGATION\_REQUIRED:** 如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。

**2、PROPAGATION\_SUPPORTS:** 支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。

**3、PROPAGATION\_MANDATORY:** 支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。

**4、PROPAGATION\_REQUIRES\_NEW:** 创建新事务，无论当前存不存在事务，都创建新事务。

**5、PROPAGATION\_NOT\_SUPPORTED:** 以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

**6、PROPAGATION\_NEVER:** 以非事务方式执行，如果当前存在事务，则抛出异常。

**7、PROPAGATION\_NESTED:** 如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与 PROPAGATION\_REQUIRED 类似的操作。

### 4、SpringBoot 事务关键点

#### 1、事务的自动配置

TransactionAutoConfiguration

#### 2、事务的坑

在同一个类里面，编写两个方法，内部调用的时候，会导致事务设置失效。原因是没有用到

代理对象的缘故。

解决：

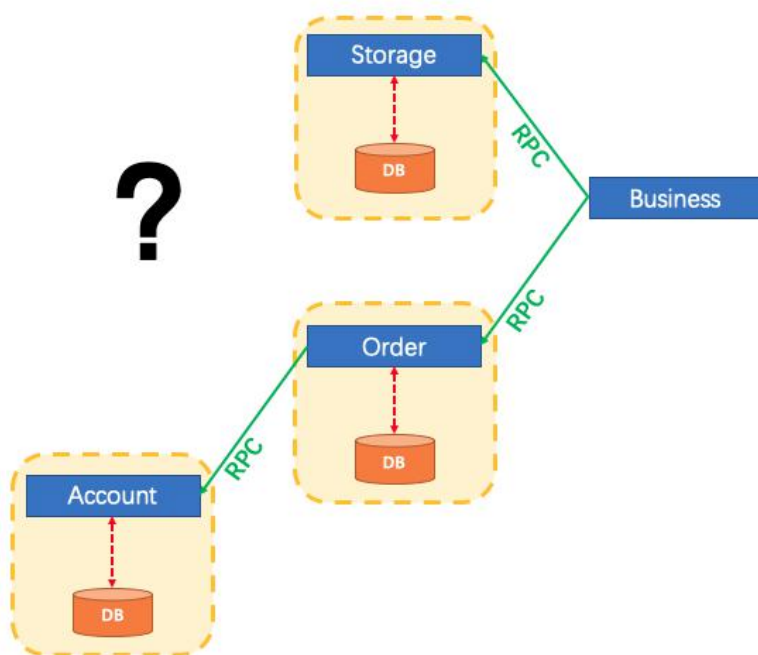
- 0)、导入 spring-boot-starter-aop
- 1)、`@EnableTransactionManagement(proxyTargetClass = true)`
- 2)、`@EnableAspectJAutoProxy(exposeProxy=true)`
- 3)、`AopContext.currentProxy()` 调用方法

## 二、分布式事务

### 1、为什么有分布式事务

分布式系统经常出现的异常

机器宕机、网络异常、消息丢失、消息乱序、数据错误、不可靠的 TCP、存储数据丢失...



分布式事务是企业集成中的一个技术难点，也是每一个分布式系统架构中都会涉及到的一个东西，特别是在微服务架构中，几乎可以说是无法避免。

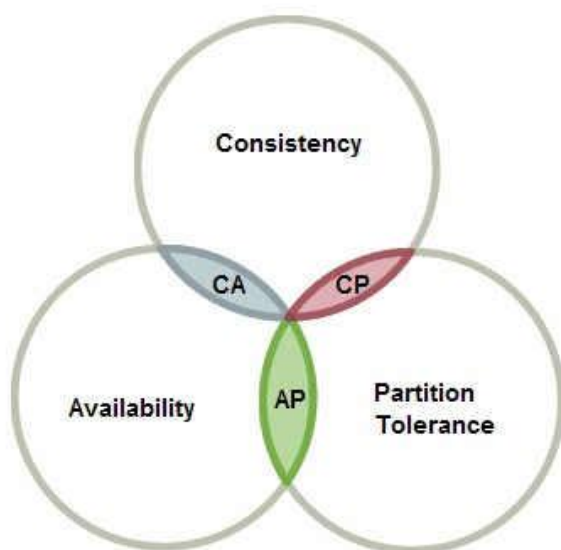
## 2、CAP 定理与 BASE 理论

### 1、CAP 定理

CAP 原则又称 CAP 定理，指的是在一个分布式系统中

- 一致性（Consistency）：
  - 在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）
- 可用性（Availability）
  - 在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）
- 分区容错性（Partition tolerance）
  - 大多数分布式系统都分布在多个子网络。每个子网络就叫做一个区（partition）。分区容错的意思是，区间通信可能失败。比如，一台服务器放在中国，另一台服务器放在美国，这就是两个区，它们之间可能无法通信。

CAP 原则指的是，这三个要素最多只能同时实现两点，**不可能三者兼顾**。



一般来说，分区容错无法避免，因此可以认为 CAP 的 P 总是成立。CAP 定理告诉我们，剩下的 C 和 A 无法同时做到。

分布式系统中实现一致性的 raft 算法、paxos

<http://thesecretlivesofdata.com/raft/>

## 2、面临的问题

对于多数大型互联网应用的场景，主机众多、部署分散，而且现在的集群规模越来越大，所以节点故障、网络故障是常态，而且要保证服务可用性达到 99.99999%（N 个 9），即保证 P 和 A，舍弃 C。

## 3、BASE 理论

是对 CAP 理论的延伸，思想是即使无法做到强一致性（CAP 的一致性就是强一致性），但可以采用适当的采取弱一致性，即**最终一致性**。

BASE 是指

- 基本可用（Basically Available）
  - 基本可用是指分布式系统在出现故障的时候，允许损失部分可用性（例如响应时间、功能上的可用性），允许损失部分可用性。需要注意的是，基本可用绝不等价于系统不可用。
    - ◆ 响应时间上的损失：正常情况下搜索引擎需要在 0.5 秒之内返回给用户相应的查询结果，但由于出现故障（比如系统部分机房发生断电或断网故障），查询结果的响应时间增加到了 1~2 秒。
    - ◆ 功能上的损失：购物网站在购物高峰（如双十一）时，为了保护系统的稳定性，部分消费者可能会被引导到一个降级页面。
- 软状态（Soft State）
  - 软状态是指允许系统存在中间状态，而该中间状态不会影响系统整体可用性。分布式存储中一般一份数据会有多个副本，允许不同副本同步的延时就是软状态的体现。mysql replication 的异步复制也是一种体现。
- 最终一致性（Eventual Consistency）
  - 最终一致性是指系统中的所有数据副本经过一定时间后，最终能够达到一致的状态。弱一致性和强一致性相反，最终一致性是弱一致性的一种特殊情况。

## 4、强一致性、弱一致性、最终一致性

从客户端角度，多进程并发访问时，更新过的数据在不同进程如何获取的不同策略，决定了不同的一致性。对于关系型数据库，要求更新过的数据能被后续的访问都能看到，这是**强一致性**。如果能容忍后续的部分或者全部访问不到，则是**弱一致性**。如果经过一段时间后要求能访问到更新后的数据，则是**最终一致性**。

### 3、分布式事务几种方案

#### 1)、2PC 模式

数据库支持的 2PC【2 phase commit 二阶提交】，又叫做 XA Transactions。

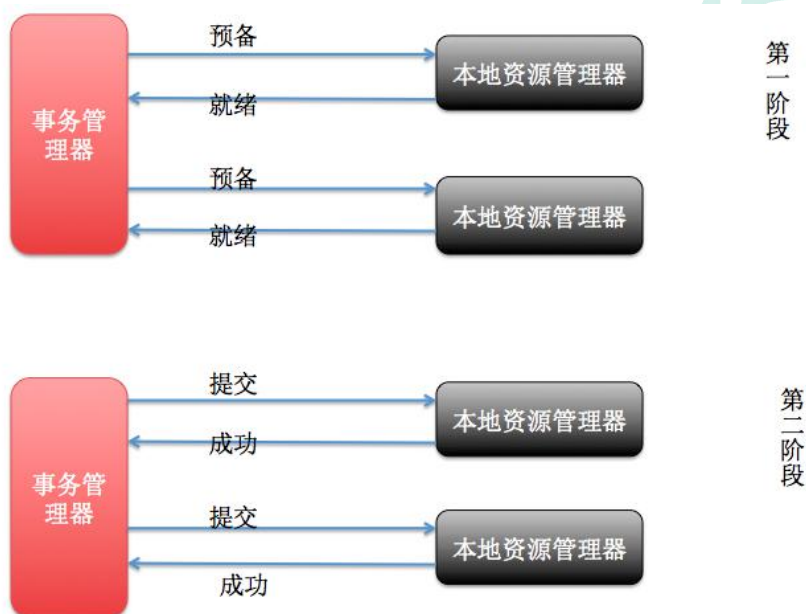
MySQL 从 5.5 版本开始支持，SQL Server 2005 开始支持，Oracle 7 开始支持。

其中，XA 是一个两阶段提交协议，该协议分为以下两个阶段：

第一阶段：事务协调器要求每个涉及到事务的数据库预提交(precommit)此操作，并反映是否可以提交。

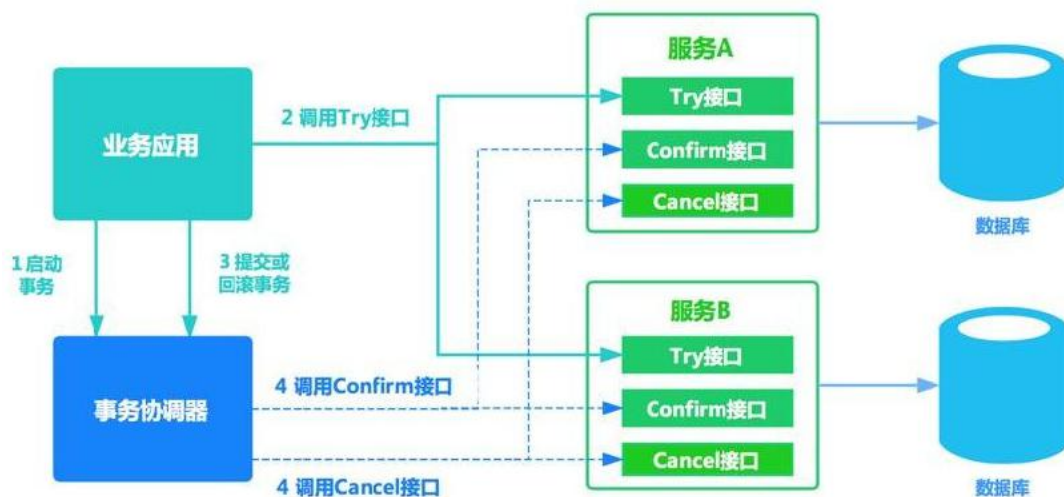
第二阶段：事务协调器要求每个数据库提交数据。

其中，如果有任何一个数据库否决此次提交，那么所有数据库都会被要求回滚它们在此事务中的那部分信息。



- XA 协议比较简单，而且一旦商业数据库实现了 XA 协议，使用分布式事务的成本也比较低。
- **XA 性能不理想**，特别是在交易下单链路，往往并发量很高，XA 无法满足高并发场景
- XA 目前在商业数据库支持的比较理想，在 **mysql 数据库**中支持的不太理想，mysql 的 XA 实现，没有记录 prepare 阶段日志，主备切换回导致主库与备库数据不一致。
- 许多 nosql 也没有支持 XA，这让 XA 的应用场景变得非常狭隘。
- 也有 3PC，引入了超时机制（无论协调者还是参与者，在向对方发送请求后，若长时间未收到回应则做出相应处理）







### 3)、柔性事务-最大努力通知型方案

按规律进行通知，不保证数据一定能通知成功，但会提供可查询操作接口进行核对。这种方案主要用在与第三方系统通讯时，比如：调用微信或支付宝支付后的支付结果通知。这种方案也是结合 MQ 进行实现，例如：通过 MQ 发送 http 请求，设置最大通知次数。达到通知次数后即不再通知。

案例：银行通知、商户通知等（各大交易业务平台间的商户通知：多次通知、查询校对、对账文件），支付宝的支付成功异步回调



#### 4)、柔性事务-可靠消息+最终一致性方案（异步确保型）

实现：业务处理服务在业务事务提交之前，向实时消息服务请求发送消息，实时消息服务只记录消息数据，而不是真正的发送。业务处理服务在业务事务提交之后，向实时消息服务确认发送。只有在得到确认发送指令后，实时消息服务才会真正发送。

防止消息丢失：

/\*\*

\* 1、做好消息确认机制（publisher, consumer【手动ack】）

\* 2、每一个发送的消息都在数据库做好记录。定期将失败的消息再次发送一遍

\*/

```
CREATE TABLE `mq_message` (  
  `message_id` char(32) NOT NULL,  
  `content` text,  
  `to_exchane` varchar(255) DEFAULT NULL,  
  `routing_key` varchar(255) DEFAULT NULL,  
  `class_type` varchar(255) DEFAULT NULL,  
  `message_status` int(1) DEFAULT '0' COMMENT '0-新建 1-已发送 2-错误抵达 3-已抵达',  
  `create_time` datetime DEFAULT NULL,  
  `update_time` datetime DEFAULT NULL,  
  PRIMARY KEY (`message_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```



