全面了解 MySql 中的事务

最近一直在做订单类的项目，使用了事务。我们的数据库选用的是 MySql，存储引擎选用 innoDB，innoDB 对事务有着良好的支持。这篇文章我们一起来扒一扒事务相关的知识。

为什么要有事务？

事务广泛的运用于订单系统、银行系统等多种场景。如果有以下一个场景：A 用户和 B 用户是银行的储户。现在 A 要给 B 转账 500 元。那么需要做以下几件事：

1. 检查 A 的账户余额>500 元；
2. A 账户扣除 500 元；
3. B 账户增加 500 元；

正常的流程走下来，A 账户扣了 500，B 账户加了 500，皆大欢喜。那如果 A 账户扣了钱之后，系统出故障了呢？A 白白损失了 500，而 B 也没有收到本该属于他的 500。以上的案例中，隐藏着一个前提条件：A 扣钱和 B 加钱，要么同时成功，要么同时失败。事务的需求就在于此。

事务是什么？

与其给事务定义，不如说一说事务的特性。众所周知，事务需要满足 ACID 四个特性。

1. A(atomicity) 原子性。一个事务的执行被视为一个不可分割的最小单元。事务里面的操作，要么全部成功执行，要么全部失败回滚，不可以只执行其中的一部分。
2. C(consistency) 一致性。一个事务的执行不应该破坏数据库的完整性约束。如果上述例子中第 2 个操作执行后系统崩溃，保证 A 和 B 的金钱总计是不会变的。
3. I(isolation) 隔离性。通常来说，事务之间的行为不应该互相影响。然而实际情况中，事务相互影响的程度受到隔离级别的影响。文章后面会详述。
4. D(durability) 持久性。事务提交之后，需要将提交的事务持久化到磁盘。即使系统崩溃，提交的数据也不应该丢失。

事务的四种隔离级别

前文中提到，事务的隔离性受到隔离级别的影响。那么事务的隔离级别是什么呢？事务的隔离级别可以认为是事务的"自私"程度，它定义了事务之间的可见性。隔离级别分为以下几种：

1.READ UNCOMMITTED(未提交读)。在 RU 的隔离级别下，事务 A 对数据做的修改，即使没有提交，对于事务 B 来说也是可见的，这种问题叫脏读。这是隔离程度较低的一种隔离级别，在实际运用中会引起很多问题，因此一般不常用。

2.READ COMMITTED(提交读)。在 RC 的隔离级别下，不会出现脏读的问题。事务 A 对数据做的修改，提交之后会对事务 B 可见，举例，事务 B 开启时读到数据 1，接下来事务 A 开启，把这个数据改成 2，提交，B 再次读取这个数据，会读到最新的数据 2。在 RC 的隔离级别下，会出现不可重复读的问题。这个隔离级别是许多数据库的默认隔离级别。

3.REPEATABLE READ(可重复读)。在 RR 的隔离级别下，不会出现不可重复读的问题。事务 A 对数据做的修改，提交之后，对于先于事务 A 开启的事务是不可见的。举例，事务 B 开启时读到数据 1，接下来事务 A 开启，把这个数据改成 2，提交，B 再次读取这个数据，仍然只能读到 1。在 RR 的隔离级别下，会出现幻读的问题。幻读的意思是，当某个事务在读取

某个范围内的值的时候，另外一个事务在这个范围内插入了新记录，那么之前的事务再次读取这个范围的值，会读取到新插入的数据。Mysql 默认的隔离级别是 RR，然而 mysql 的 innoDB 引擎间隙锁成功解决了幻读的问题。

4.SERIALIZABLE(可串行化)。可串行化是最高的隔离级别。这种隔离级别强制要求所有事物串行执行，在这种隔离级别下，读取的每行数据都加锁，会导致大量的锁征用问题，性能最差。

为了帮助理解四种隔离级别，这里举个例子。如图 1，事务 A 和事务 B 先后开启，并对数据 1 进行多次更新。四个小人在不同的时刻开启事务，可能看到数据 1 的哪些值呢？

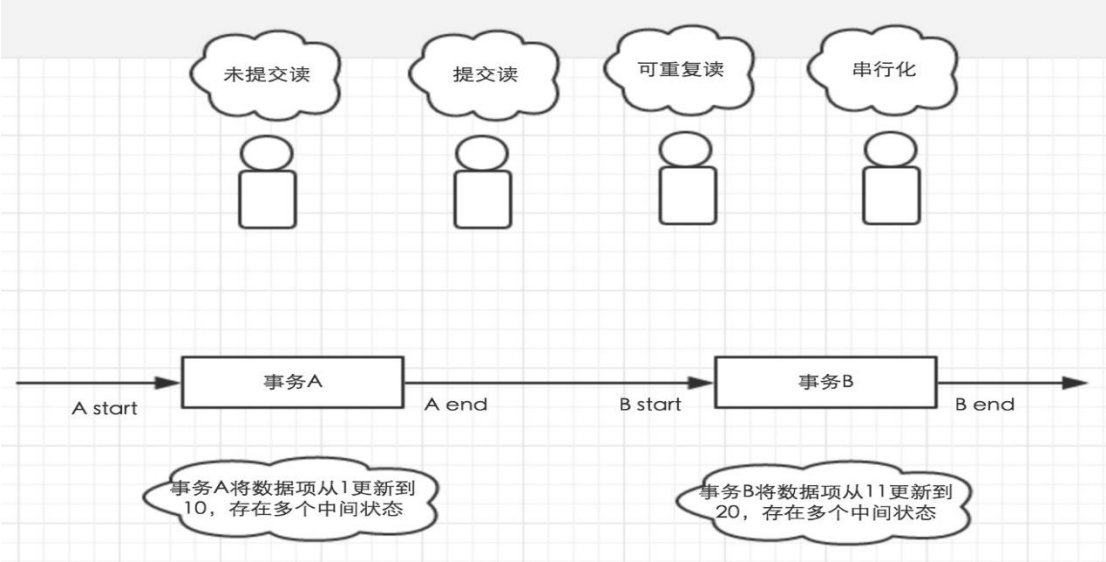

图 1

第一个小人，可能读到 1-20 之间的任何一个。因为未提交读的隔离级别下，其他事务对数据的修改也是对当前事务可见的。第二个小人可能读到 1，10 和 20，他只能读到其他事务已经提交了的数据。第三个小人读到的数据去决于自身事务开启的时间点。在事务开启时，读到的是多少，那么在事务提交之前读到的值就是多少。第四个小人，只有在 A end 到 B start 之间开启，才有可能读到数据，而在事务 A 和事务 B 执行的期间是读不到数据的。因为第四小人读数据是需要加锁的，事务 A 和 B 执行期间，会占用数据的写锁，导致第四个小人等待锁。

图 2 罗列了不同隔离级别所面对的问题。

隔离级别	发生脏读	不可重复读	发生幻读	加锁读
未提交读	√	√	√	×
提交读	×	√	√	×
可重复读	×	×	√	×
可串行化	×	×	×	√

图 2

很显然，隔离级别越高，它所带来的资源消耗也就越大(锁)，因此它的并发性能越低。准确的说，在可串行化的隔离级别下，是没有并发的。

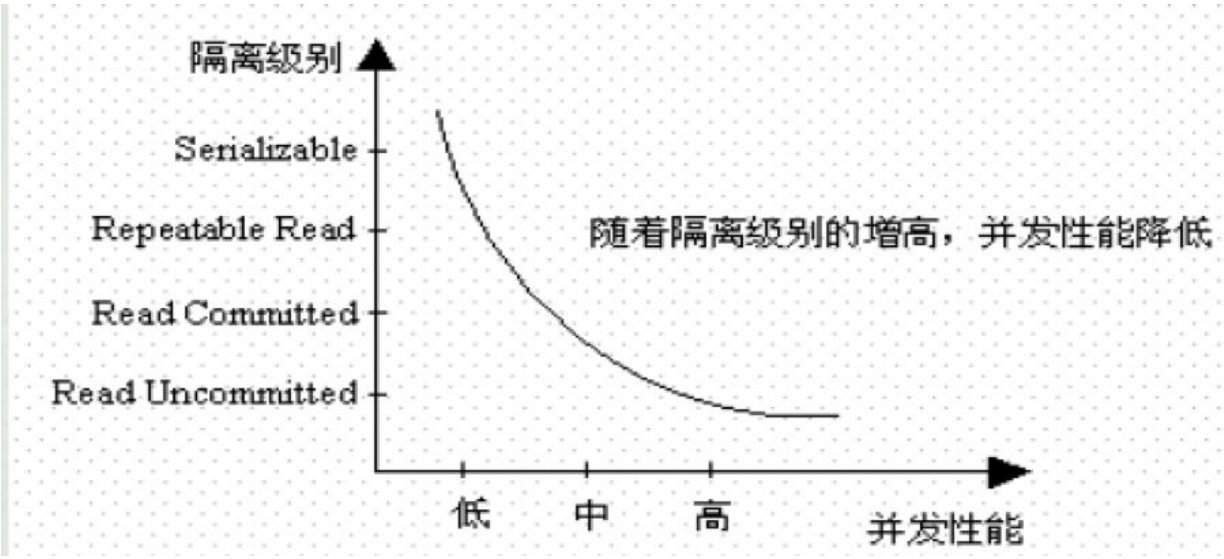

图 3

MySql 中的事务

事务的实现是基于数据库的存储引擎。不同的存储引擎对事务的支持程度不一样。mysql 中支持事务的存储引擎有 innoDB 和 NDB。innoDB 是 mysql 默认的存储引擎，默认的隔离级别是 RR，并且在 RR 的隔离级别下更进一步，通过多版本**并发控制**（MVCC，Multiversion Concurrency Control ）解决不可重复读问题，加上间隙锁（也就是并发控制）解决幻读问题。因此 innoDB 的 RR 隔离级别其实实现了串行化级别的效果，而且保留了比较好的并发性能。

事务的隔离性是通过锁实现，而事务的原子性、一致性和持久性则是通过事务日志实现。说到事务日志，不得不说的就是 redo 和 undo。

1.redo log

在 innoDB 的存储引擎中，事务日志通过重做(redo)日志和 innoDB 存储引擎的日志缓冲(InnoDB Log Buffer)实现。事务开启时，事务中的操作，都会先写入存储引擎的日志缓冲中，在事务提交之前，这些缓冲的日志都需要提前刷新到磁盘上持久化，这就是 DBA 们口中常说的"日志先行"(Write-Ahead Logging)。当事务提交之后，在 Buffer Pool 中映射的数据文件才会慢慢刷新到磁盘。此时如果数据库崩溃或者宕机，那么当系统重启进行恢复时，就可以根据 redo log 中记录的日志，把数据库恢复到崩溃前的一个状态。未完成的事务，可以继续提交，也可以选择回滚，这基于恢复的策略而定。

在系统启动的时候，就已经为 redo log 分配了一块连续的存储空间，以顺序追加的方式记录 Redo Log，通过顺序 IO 来改善性能。所有的事务共享 redo log 的存储空间，它们的 Redo Log 按语句的执行顺序，依次交替的记录在一起。如下一个简单示例：

记录 1：<trx1, insert...>

记录 2：<trx2, delete...>

记录 3：<trx3, update...>

记录 4：<trx1, update...>

记录 5：<trx3, insert...>

2.undo log

undo log 主要为事务的回滚服务。在事务执行的过程中，除了记录 redo log，还会记录一定量的 undo log。undo log 记录了数据在每个操作前的状态，如果事务执行过程中需要回滚，就可以根据 undo log 进行回滚操作。单个事务的回滚，只会回滚当前事务做的操作，并不会影响到其他的事务做的操作。

以下是 undo+redo 事务的简化过程

假设有 2 个数值，分别为 A 和 B,值为 1，2

1. start transaction;
2. 记录 A=1 到 undo log;
3. update A = 3;
4. 记录 A=3 到 redo log;
5. 记录 B=2 到 undo log;
6. update B = 4;
7. 记录 B = 4 到 redo log;
8. 将 redo log 刷新到磁盘
9. commit

在 1-8 的任意一步系统宕机，事务未提交，该事务就不会对磁盘上的数据做任何影响。如果在 8-9 之间宕机，恢复之后可以选择回滚，也可以选择继续完成事务提交，因为此时 redo log 已经持久化。若在 9 之后系统宕机，内存映射中变更的数据还来不及刷回磁盘，那么系统恢复之后，可以根据 redo log 把数据刷回磁盘。

所以，redo log 其实保障的是事务的持久性和一致性，而 undo log 则保障了事务的原子性。

分布式事务

分布式事务的实现方式有很多，既可以采用 innoDB 提供的原生的事务支持，也可以采用消息队列来实现分布式事务的最终一致性。这里我们主要聊一下 innoDB 对分布式事务的支持。

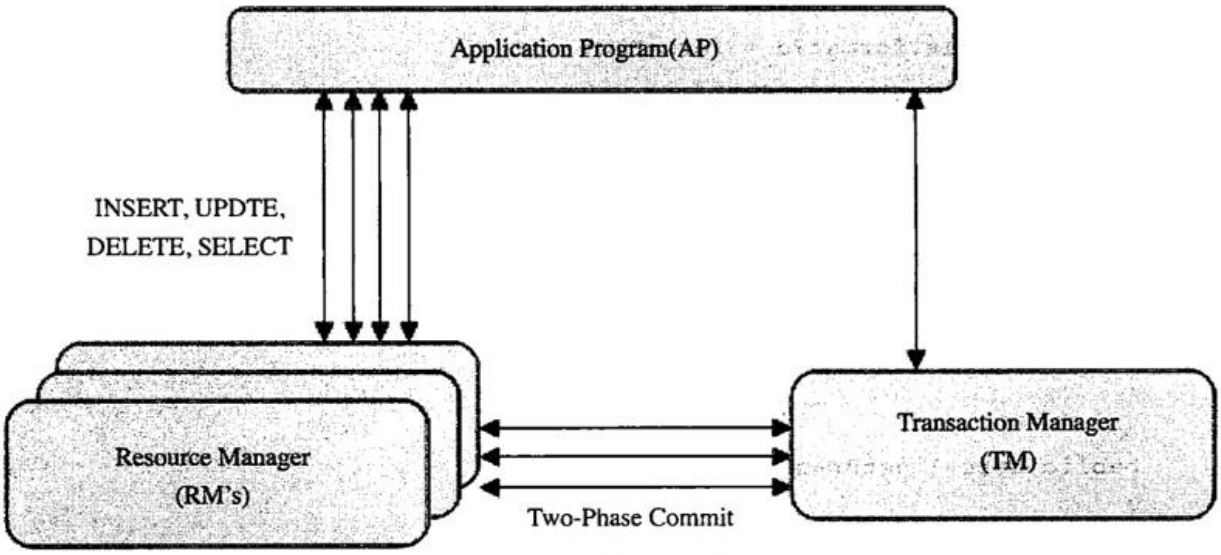

图7-1 分布式事务模型

如图，mysql 的分布式事务模型。模型中分三块：应用程序（AP）、资源管理器（RM）、事务管理器（TM）。

应用程序定义了事务的边界，指定需要做哪些事务；

资源管理器提供了访问事务的方法，通常一个数据库就是一个资源管理器；

事务管理器协调参与了全局事务中的各个事务。

分布式事务采用两段式提交（two-phase commit）的方式。第一阶段所有的事务节点开始准备，告诉事务管理器 ready。第二阶段事务管理器告诉每个节点是 commit 还是 rollback。如果有一个节点失败，就需要全局的节点全部 rollback，以此保障事务的原子性。

总结

什么时候需要使用事务呢？我想，只要业务中需要满足 ACID 的场景，都需要事务的支持。尤其在订单系统、银行系统中，事务是不可或缺的。这篇文章主要介绍了事务的特性，以及 mysql innoDB 对事务的支持。事务相关的知识远不止文中所说，本文仅作抛砖引玉，不足之处还望读者多多见谅。