二

33 MySQL调优之事务：高并发场景下的数据库事务调优

你好，我是刘超。

数据库事务是数据库系统执行过程中的一个逻辑处理单元，保证一个数据库操作要么成功，要么失败。谈到他，就不得不提 ACID 属性了。数据库事务具有以下四个基本属性：原子性（Atomicity）、一致性（Consistent）、隔离性（Isolation）以及持久性（Durable）。正是这些特性，才保证了数据库事务的安全性。而在 MySQL 中，鉴于 MyISAM 存储引擎不支持事务，所以接下来的内容都是在 InnoDB 存储引擎的基础上进行讲解的。

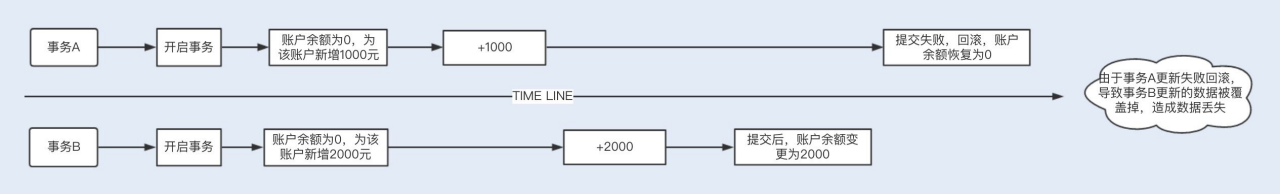
我们知道，在 Java 并发编程中，可以多线程并发执行程序，然而并发虽然提高了程序的执行效率，却给程序带来了线程安全问题。事务跟多线程一样，为了提高数据库处理事务的吞吐量，数据库同样支持并发事务，而在并发运行中，同样也存在着安全性问题，例如，修改数据丢失，读取数据不一致等。

在数据库事务中，事务的隔离是解决并发事务问题的关键， 今天我们就重点了解下事务隔离的实现原理，以及如何优化事务隔离带来的性能问题。

并发事务带来的问题

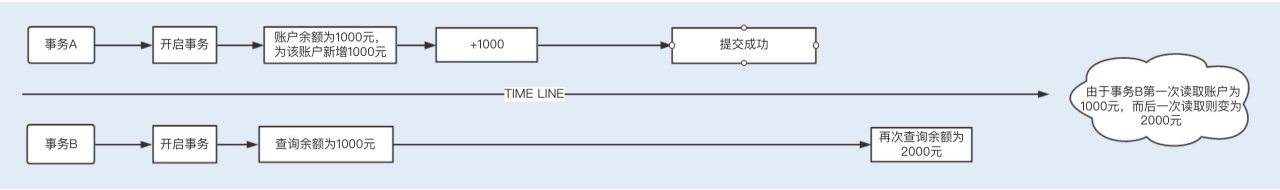
我们可以通过以下几个例子来了解下并发事务带来的几个问题：

\1. 数据丢失

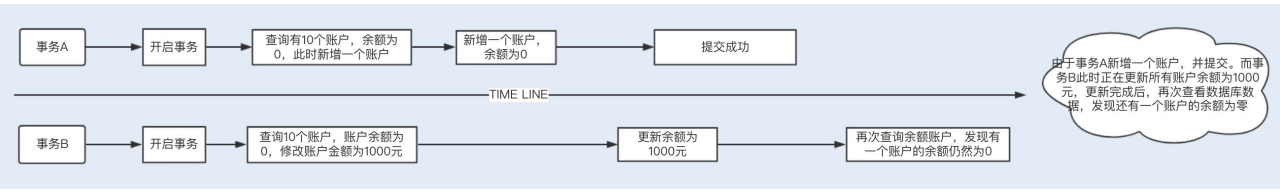


\2. 脏读

\3. 不可重复读



\4. 幻读



事务隔离解决并发问题

以上 4 个并发事务带来的问题，其中，数据丢失可以基于数据库中的悲观锁来避免发生，即在查询时通过在事务中使用 select xx for update 语句来实现一个排他锁，保证在该事务结束之前其他事务无法更新该数据。

当然，我们也可以基于乐观锁来避免，即将某一字段作为版本号，如果更新时的版本号跟之前的版本一致，则更新，否则更新失败。剩下 3 个问题，其实是数据库读一致性造成的，需要数据库提供一定的事务隔离机制来解决。

我们通过加锁的方式，可以实现不同的事务隔离机制。在了解事务隔离机制之前，我们不妨先来了解下 MySQL 都有哪些锁机制。

InnoDB 实现了两种类型的锁机制：共享锁（S）和排他锁（X）。共享锁允许一个事务读数据，不允许修改数据，如果其他事务要再对该行加锁，只能加共享锁；排他锁是修改数据时加的锁，可以读取和修改数据，一旦一个事务对该行数据加锁，其他事务将不能再对该数据加任务锁。

熟悉了以上 InnoDB 行锁的实现原理，我们就可以更清楚地理解下面的内容。

在操作数据的事务中，不同的锁机制会产生以下几种不同的事务隔离级别，不同的隔离级别分别可以解决并发事务产生的几个问题，对应如下：

未提交读（Read Uncommitted）：在事务 A 读取数据时，事务 B 读取和修改数据加了

共享锁。这种隔离级别，会导致脏读、不可重复读以及幻读。

已提交读（Read Committed）：在事务 A 读取数据时增加了共享锁，一旦读取，立即释放锁，事务 B 读取修改数据时增加了行级排他锁，直到事务结束才释放锁。也就是说，事务 A 在读取数据时，事务 B 只能读取数据，不能修改。当事务 A 读取到数据后，事务 B 才能修改。这种隔离级别，可以避免脏读，但依然存在不可重复读以及幻读的问题。

可重复读（Repeatable Read）：在事务 A 读取数据时增加了共享锁，事务结束，才释放锁，事务 B 读取修改数据时增加了行级排他锁，直到事务结束才释放锁。也就是说，事务 A 在没有结束事务时，事务 B 只能读取数据，不能修改。当事务 A 结束事务，事务 B 才能修改。这种隔离级别，可以避免脏读、不可重复读，但依然存在幻读的问题。

可序列化（Serializable）：在事务 A 读取数据时增加了共享锁，事务结束，才释放锁，事务 B 读取修改数据时增加了表级排他锁，直到事务结束才释放锁。可序列化解决了脏读、不可重复读、幻读等问题，但隔离级别越来越高的同时，并发性会越来越低。

InnoDB 中的 RC 和 RR 隔离事务是基于多版本并发控制（MVVC）实现高性能事务。一旦数据被加上排他锁，其他事务将无法加入共享锁，且处于阻塞等待状态，如果一张表有大量的请求，这样的性能将是无法支持的。

MVVC 对普通的 Select 不加锁，如果读取的数据正在执行 Delete 或 Update 操作，这时读取操作不会等待排它锁的释放，而是直接利用 MVVC 读取该行的数据快照（数据快照是指在该行的之前版本的数据，而数据快照的版本是基于 undo 实现的，undo 是用来做事务回滚的，记录了回滚的不同版本的行记录）。MVVC 避免了对数据重复加锁的过程，大大提高了读操作的性能。

锁具体实现算法

我们知道，InnoDB 既实现了行锁，也实现了表锁。行锁是通过索引实现的，如果不通过索引条件检索数据，那么 InnoDB 将对表中所有的记录进行加锁，其实就是升级为表锁了。

行锁的具体实现算法有三种：record lock、gap lock 以及 next-key lock。record lock 是专门对索引项加锁；gap lock 是对索引项之间的间隙加锁；next-key lock 则是前面两种的组合，对索引项以其之间的间隙加锁。

只在可重复读或以上隔离级别下的特定操作才会取得 gap lock 或 next-key lock，在 Select、Update 和 Delete 时，除了基于唯一索引的查询之外，其他索引查询时都会获取 gap lock 或 next-key lock，即锁住其扫描的范围。

优化高并发事务

通过以上讲解，相信你对事务、锁以及隔离级别已经有了一个透彻的了解了。清楚了问题，我们就可以聊聊高并发场景下的事务到底该如何调优了。

1. 结合业务场景，使用低级别事务隔离

在高并发业务中，为了保证业务数据的一致性，操作数据库时往往会使用到不同级别的事务隔离。隔离级别越高，并发性能就越低。

那换到业务场景中，我们如何判断用哪种隔离级别更合适呢？我们可以通过两个简单的业务来说下其中的选择方法。

我们在修改用户最后登录时间的业务场景中，这里对查询用户的登录时间没有特别严格的准确性要求，而修改用户登录信息只有用户自己登录时才会修改，不存在一个事务提交的信息被覆盖的可能。所以我们允许该业务使用最低隔离级别。

而如果是账户中的余额或积分的消费，就存在多个客户端同时消费一个账户的情况，此时我们应该选择 RR 级别来保证一旦有一个客户端在对账户进行消费，其他客户端就不可能对该账户同时进行消费了。

2. 避免行锁升级表锁

前面讲了，在 InnoDB 中，行锁是通过索引实现的，如果不通过索引条件检索数据，行锁将会升级到表锁。我们知道，表锁是会严重影响到整张表的操作性能的，所以我们应该避免他。

3. 控制事务的大小，减少锁定的资源量和锁定时间长度

你是否遇到过以下 SQL 异常呢？在抢购系统的日志中，在活动区间，我们经常可以看到这种异常日志：

```
MySQLQueryInterruptedException: Query execution was interrupted
```

由于在抢购提交订单中开启了事务，在高并发时对一条记录进行更新的情况下，由于更新记录所在的事务还可能存在其他操作，导致一个事务比较长，当有大量请求进入时，就可能导致一些请求同时进入到事务中。

又因为锁的竞争是不公平的，当多个事务同时对一条记录进行更新时，极端情况下，一个更新操作进去排队系统后，可能会一直拿不到锁，最后因超时被系统打断踢出。

在用户购买商品时，首先我们需要查询库存余额，再新建一个订单，并扣除相应的库存。这一系列操作是处于同一个事务的。

以上业务若是在两种不同的执行顺序下，其结果都是一样的，但在事务性能方面却不一样：

执行顺序1	执行顺序2
1. 开启事务 2. 查询库存，判断库存是否满足 3. 新建订单 4. 扣除库存 5. 提交或回滚	1. 开启事务 2. 查询库存，判断库存是否满足 3. 扣除库存 4. 新建订单 5.提交或回滚

这是因为，虽然这些操作在同一个事务，但锁的申请在不同时间，只有当其他操作都执行完，才会释放所有锁。因为扣除库存是更新操作，属于行锁，这将会影响到其他操作该数据的事务，所以我们应该尽量避免长时间地持有该锁，尽快释放该锁。

又因为先新建订单和先扣除库存都不会影响业务，所以我们可以将扣除库存操作放到最后，也就是使用执行顺序 1，以此尽量减小锁的持有时间。

总结

其实 MySQL 的并发事务调优和 Java 的多线程编程调优非常类似，都是可以通过减小锁粒度和减少锁的持有时间进行调优。在 MySQL 的并发事务调优中，我们尽量在可以使用低事务隔离级别的业务场景中，避免使用高事务隔离级别。

在功能业务开发时，开发人员往往会为了追求开发速度，习惯使用默认的参数设置来实现业务功能。例如，在 service 方法中，你可能习惯默认使用 transaction，很少再手动变更事务隔离级别。但要知道，transaction 默认是 RR 事务隔离级别，在某些业务场景下，可能并不合适。因此，我们还是要结合具体的业务场景，进行考虑。

思考题

以上我们主要了解了锁实现事务的隔离性，你知道 InnoDB 是如何实现原子性、一致性和持久性的吗？

上一页 下一页