34 | MySQL调优之事务：高并发场景下的数据库事务调优

2019-08-08 刘超

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。

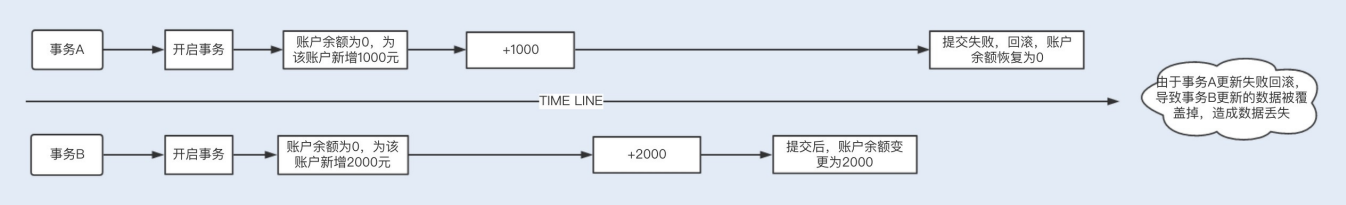
我们知道，在Java并发编程中，可以多线程并发执行程序，然而并发虽然提高了程序的执行效率，却给程序带来了线程安全问题。事务跟多线程一样，为了提高数据库处理事务的吞吐量，数据库同样支持并发事务，而在并发运行中，同样也存在着安全性问题，例如，修改数据丢失，读取数据不一致等。

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。

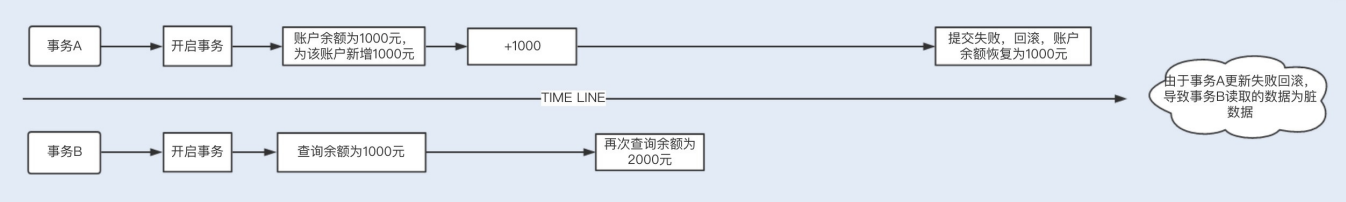
并发事务带来的问题

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：

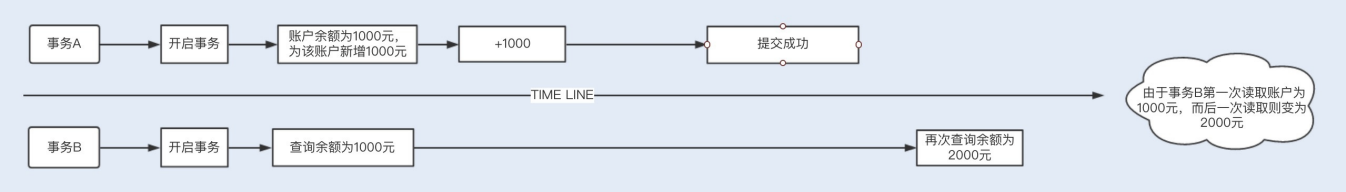
1.数据丢失



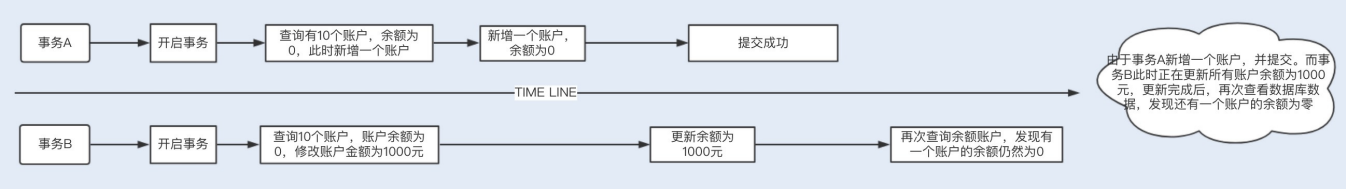
2.脏读



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读取数据加了共享锁，修改数据时加了排它锁。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（MVCC）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。

MVCC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MVCC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）。MVCC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？

期待在留言区看到你的见解。也欢迎你点击“请朋友读”，把今天的内容分享给身边的朋友，邀请他一起讨论。

精选留言

许童童

14

binlog + redo log 两阶段提交保证持久性

事务的回滚机制 保证原子性 要么全部提交成功 要么回滚

undo log + MVCC 保证一致性 事务开始和结束的过程不会其它事务看到 为了并发可以适当破坏一致性

2019-08-08

作者回复

数据库基础非常扎实，赞

2019-08-09

QQ怪

5

默认transaction用的是数据库默认的隔离级别不是一定是RR，只是用MySQL默认是RR

2019-08-08

作者回复

对的

2019-08-09

胡峣

5

老师能否重点讲一下record lock、gap lock 以及 next-key lock？

2019-08-08

作者回复

好的，后面安排

2019-08-09

黎波拉小建 1

一个问题困扰我挺久希望解答，spring的事务隔离和db的事务隔离机制有什么关系，这么问吧比如db里默认是RR的隔离级别（默认也肯定会有一个隔离级别），我spring的事务里就不用配置隔离级别了？如果spring的事务里的代码并没有db操作我也能设置spring的事务隔离级别？其次就是设置db级别的事务隔离的话，那如果业务不一样的话是不是理论上可以把库拆开来？因为肯定有一些库用到了不需要的更高隔离级别影响性能。

2019-10-21

作者回复

文中解释了怎么实现了数据库的事务隔离级别，那就是通过锁来实现的，Spring的封装只是使用了 MySQL提供的标准接口，只要在设置transaction的时候设置事务隔离级别。

如果不想影响不同业务相互之间的性能，可以通过拆库实现。

2019-10-26

咬尖月牙儿 1

老师，您这些数据库的相关知识是从官网还是某些书籍中获取的？我的数据库进阶的知识比较浅，看官网觉得有点费劲，不知道从哪入手去看去学

2019-08-28

作者回复

可以阅读一些基础入门的书籍，也是作者已经消化梳理过了的，会比较通俗易懂。

2019-09-07

黎波拉小建 0

有个问题请教下，我看到说 redolog 为了提升持久化的IO效率而产生，把每次事务写数据文件改成写redoLog然后再批量写redoLog到数据文件。我就有个问题了，redoLog和数据文件不都是磁盘读么？并且写redolog也需要在事务中同步写，因为异步写不能保证redolog必定写成功。。。求教

2019-10-11

作者回复

在40讲中，讲到了redo log的工作原理，redo log 又包括了内存中的日志缓冲（redo log buffer）以及保存在磁盘的重做日志文件（redo log file），前者存储在内存中，容易丢失，后者持久化在磁盘中，不会丢失。

MySQL支持用户自定义在commit时如何将log buffer中的日志刷log file中。这种控制通过变量 innodb_flush_log_at_trx_commit 的值来决定。该变量有3种值：0、1、2，默认为1，即每提交一次事务都写到磁盘中，而设置为0时，为每秒将事务写到磁盘中。

2019-10-13

ASCE1885 0

Serializable 翻译为 可串行化 会更合理一些。

2019-10-10

作者回复

收到建议

2019-10-13

张三丰

0

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。

这句话怎么理解呢？事务B读取和修改数据加了共享锁？修改数据不是只能加排它锁么？

2019-10-04

作者回复

事务B读取数据加了共享锁，修改数据时加了排它锁

2019-10-06

风轻扬

0

老师，对数据库锁不是很熟悉。以下结构是正确的吗？
数据库锁=乐观锁 + 悲观锁[共享锁、排它锁[行锁、表锁]]
这些是数据库所有的锁吗?

2019-09-22

作者回复

是的

2019-10-02

godtrue

0

请问老师数据库的事务本质是不是都是利用单库的事务机制实现的?分布式数据库事务，只是将控制事务提交或回滚的动作往上层提升啦?

2019-09-14

作者回复

是的

2019-09-15

小学生

0

老师update语句是行锁，那insert语句是什么锁呢

2019-09-10

作者回复

也是行锁

2019-09-11

加载中.....

0

老师好，有个问题请教下。文章中说：“可重复读（Repeatable Read）可以避免脏读、不可重复读，但依然存在幻读的问题。”
我之前了解的是，SQL标准不要求RR解决幻读，但InnoDB的 RR下 是不会产生幻读的。
而且自己实验也是InnoDB在RR下没有幻读的现象。操作如下：

sessionA: begin;
sessionA: select * from t where id>1;(2条记录)
sessionB: insert into t(id,a) values(5,5);
sessionA: select * from t where id>1;(还是2条看不到5,5。没有产生幻读)

2019-08-29

作者回复

这是RR下使用的间隙锁，所以不会出现幻读

2019-09-08

小布丁 0

老师数据库的事务和Spring的事务是一回事吗？写业务代码的时候，两种不同的事务分别起什么作用呢？或者说控制范围有何不同呢？

2019-08-23

作者回复

两者实际上是一回事，Spring的事务也是基于数据库事务封装的方法。

2019-08-24

L. 0

老师，对于可重复读(Repeatable Read)的事务级别可以避免不可重复读的现象有个疑问：
对于事务A来说，它在获得共享锁期间修改了数据，比如把A改为B，修改完成后释放共享锁。在A获得共享锁期间，事务B看到的数据是A，释放共享锁后，事务B才获得排他锁，然后看到的数据是B。两次的数据不一样啊，还是没有避免不可重复读。。。。不知道我理解的哪里不对，望老师指点。。。

2019-08-14

作者回复

粗略看好像没什么问题，但仔细看就能看出问题了。

事务A存在读和改两个操作，事务B也同样存在读和改两个操作。事务A在读时，由于是共享锁，事务B也能读到该数据，当事务A进行修改就需要上排他锁了，此时事务B由于已经对该数据加了共享锁，事务A需要等待事务B释放共享锁才能获取排他锁来修改数据。同样事务B也在等待事务A释放共享锁。这种操作会导致死锁的出现。

我们一般用一个读的事务操作和一个读写事务操作来理解RR事务隔离级别。

2019-08-15

Liam 0

老师好，查询未加索引时行锁升级为表锁这里有个疑问，mvvc机制下select不是不加锁吗？除非是in share mode或for update

2019-08-12

作者回复

对的

2019-08-12

星星滴蓝天

0

老师能否多讲点innodb锁。最近我们老是出现锁等待的情况，老师可否给一些优化的思路

2019-08-09

作者回复

嗯嗯，后面会讲到死锁和锁等待的问题

2019-08-12

苏志辉

0

RR是基于MVVC的，而后者对于select不加锁，那么如果事务a有两次查询，事务b在a的两次查询之间做了修改，要保证可重复读，a两次读取的都是b改之前的快照吗？

2019-08-09

作者回复

对的，快照版本不大于事务版本号

2019-09-08

LW

0

思考题：通过redo log和undo log实现

2019-08-08

作者回复

对的，redo log保证事务的原子性以及持久性，undo log保证事务的一致性。

2019-08-09

张学磊

0

MySQL通过事务实战原子性，一个事务内的DML语句要么全部成功要么全部失败。通过redo log和undo log实现持久性和一致性，当执行DML语句时会将操作记录到redo log中并记录与之相反的操作到undo log中，事务一旦提交，就将该redolog中的操作，持久化到磁盘上，事务回滚，则执行undo log中记录的操作，恢复到执行前的状态。

2019-08-08

作者回复

对的

2019-08-09

撒旦的堕落

0

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。 老师 这个虽然降低了更新库存表那行锁持有时间 但是不是增加了订单表锁定的时间了么 还是说一个事务数据插入操作 并不会受到另一个事务数据插入操作的影响

2019-08-08