32 一起探究下事务的实现原理吧

更新时间：2020-05-19 18:02:56

> 天才就是长期劳动的结果。——牛顿

事务是 MySQL 这样的关系型数据库区别于 NoSQL 的重要方面，是保证数据一致性的重要手段。在之前讲解“事务隔离级别”时曾经说过，事务具有 ACID（原子性、一致性、隔离性和持久性）四大特性。那么，想要搞清楚事务的实现原理，其实就是对应到这四大特性的实现原理。这一节里，我们就一起探索下 MySQL 事务是怎么实现的吧。

1. 原子性概念及其实现

原子本身是化学中的一个名词，是指构成化学元素的最小粒子。对于现代计算机而言，人们对它进行了概念上的扩展，使得原子性不仅仅与 MySQL 事务有关系，它同样会出现在很多场景或技术中。

1.1 什么是原子性

原子性是指多个操作是一个不可分割的工作单位，其中的每一个操作，要么全都做，要么全都不做。原子性是事务最基本的特性，可以想象：如果不具备原子性，当其中的部分操作出现错误或者抛出异常时，就会导致数据丢失或错误，且无法恢复到原始状态。

如果事务中的某一条 SQL 语句执行失败，则已经执行的语句也必须回滚，即退回到事务执行之前的状态，且终止事务的执行。由于原子性用于保证“状态的正确性”，所以，它是数据库最基本的特性。

1.2 原子性是怎么实现的

实现原子性的关键，是当事务出现错误时能够及时回滚（撤销当前事务中已经执行成功的 SQL 语句），对于 InnoDB 而言，它依赖于 undo log（回滚日志）。undo log 的思想是非常简单的：当事务对数据库进行修改时，InnoDB 会生成对应的 undo log；如果事务执行失败或主动调用了 Rollback，则可以利用 undo log 中的信息将数据恢复到事务发生之前的状态。

关于 undo log，我们需要重点关注的是：它是逻辑日志。可以简单的认为：undo log 中存储的是 SQL，事务中执行的每一条 INSERT 都对应了一条 DELETE；同样，对于每一条 UPDATE，也会对应着一条“相反的”UPDATE 语句。

2. 持久性概念及其实现

持久性也是一个比较好理解的特性，它表达的含义是：事务一旦被提交，就不能再执行回滚操作了。如果想要 “恢复” 到事务提交之前的状态，就必须重新提交一个 “相反” 的事务，实现抵消。

2.1 什么是持久性

数据库最基本的功能就是实现对数据的存储，而这种存储一定是 “持久性” 的。持久性的定义如下：

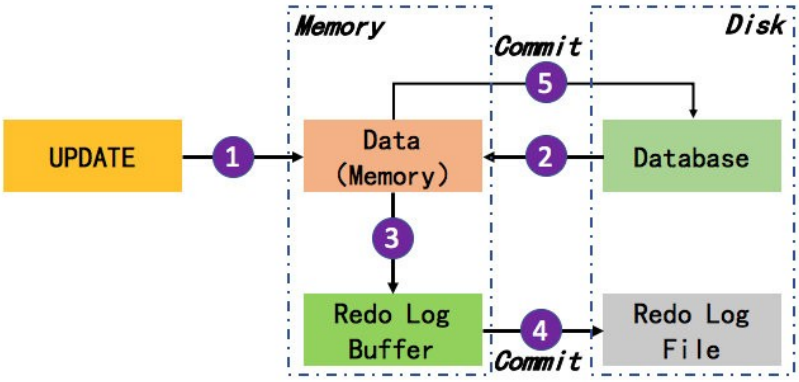
> 事务一旦被提交，它对数据库的改变就是永久性的，之后的其他操作或故障不应该对它有任何影响。

2.2 持久性是怎么实现的

与原子性类似，事务的持久性也是依赖 InnoDB 日志实现的，它叫做 Redo Log（重做日志）。Redo Log 由两部分组成：

- 内存中的重做日志缓冲区，是易失的
- 磁盘上的重做日志文件，是持久的

当我们在事务中对数据进行修改时，Redo Log 的工作流程（以 UPDATE 操作为例）如下图所示：



可以看到，Redo Log 的整个流程包含了5个步骤：

- 执行 UPDATE 操作
- 将原始数据从磁盘读取到内存中，并修改（UPDATE）内存中的数据
- 生成一条重做日志写入到 “重做日志缓冲区” 中，记录数据被修改后的值
- 事务提交时，将 “重做日志缓冲区” 中的内容刷写到 “重做日志文件” 中
- 事务提交之后，将内存中修改的数据写入到磁盘中

通过对原子性和持久性的学习，我们了解了 MySQL 中的另外两种日志（其实是 InnoDB 的）：回滚日志和重做日志。在数据库系统中，回滚日志用于对事务的影响进行撤销，而重做日志则是对已经提交的事务进行重做。

3. 隔离性概念及其实现

简单来说，隔离性是用来解决事务并发的相关问题的。它与原子性、持久性对应于事务本身不同，隔离性所涵盖的是不同事务之间的相互影响。即它所追求的是并发场景下事务之间的相互不干扰。

3.1 什么是隔离性

隔离性指的是：每一个事务内部的操作都与其他事务是隔离的，并发执行的各个事务之间不能相互干扰。数据库所定义的最高级别的隔离性是 “可串行化（Serializable）”，但在实际应用中，由于需要平衡性能，几乎不会选择严格的 “可串行化”。

不论是多么复杂的事务操作，经过拆解，都可以化简为读操作和写操作的组合。那么，隔离性实际上也就是说：

- 一个事务写操作对另一个事务写操作的影响
- 一个事务写操作对另一个事务读操作的影响

3.2 隔离性是怎么实现的

虽然 “可串行化” 可以解决隔离性的问题，但是，它会严重降低数据库的运行性能。所以，数据库定义了不同的隔离级别（在之前已经讲解过，这里不再赘述），交由用户自行去权衡隔离性与数据一致性。

数据库对于隔离级别的实现使用的是 “并发控制机制” 对在同一时间执行的事务进行控制，而 InnoDB 则是通过锁来实现 “并发控制机制”。InnoDB 同时支持表锁和行锁，且出于对性能的考虑，绝大多数情况下使用的都是行锁。

在 MySQL 中，有两种方式可以查看 InnoDB 当前持有的锁。首先，我们需要在两个会话中做一些 “操作”。如下所示：

```
-- 在 “会话 A” 中开启事务
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

-- “会话 A” 更新 worker 表的记录
mysql> UPDATE worker SET name = 'qinyi' WHERE id = 2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

-- 在 “会话 B” 中开启事务
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

-- “会话 B” 更新 worker 表的记录，此时操作会被 Hang 住
mysql> UPDATE worker SET name = 'qinyi' WHERE id = 2;
```

由于“会话 B”获取不到锁，最终会报“获取锁超时”错误。在“会话 B”等待锁的时间里，我们可以去查看锁的信息（第一种方式）：

```
-- 通过查看 information_schema 系统库中的 innodb_locks 表可以看到事务占用锁的信息
-- lock_trx_id 标识的是事务 id
-- lock_type 是 RECORD，代表行锁（行记录锁）
-- lock_mode 是 X，代表写锁
mysql> SELECT * FROM information_schema.innodb_locks;
+---------------+-------------+-----------+-----------+------------------------+------------+------------+-----------+----------+-----------+
| lock_id       | lock_trx_id | lock_mode | lock_type | lock_table             | lock_index | lock_space | lock_page | lock_rec | lock_data |
+---------------+-------------+-----------+-----------+------------------------+------------+------------+-----------+----------+-----------+
| 38934:147:3:3 | 38934       | X         | RECORD    | `imooc_mysql`.`worker` | PRIMARY    |        147 |         3 |        3 | 2         |
| 38933:147:3:3 | 38933       | X         | RECORD    | `imooc_mysql`.`worker` | PRIMARY    |        147 |         3 |        3 | 2         |
+---------------+-------------+-----------+-----------+------------------------+------------+------------+-----------+----------+-----------+
2 rows in set, 1 warning (0.01 sec)
```

我们还可以通过 `SHOW ENGINE INNODB STATUS` 命令查看当前 InnoDB 存储引擎的详细信息，这其中当然也包含了锁相关的信息（第二种方式）。如下所示（由于打印的内容太多，省略了无关的信息）：

```
mysql> SHOW ENGINE INNODB STATUS;
---TRANSACTION 38934, ACTIVE 29 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 5, OS thread handle 123145537167360, query id 169 localhost root updating

---TRANSACTION 38933, ACTIVE 88 sec
2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 2, OS thread handle 123145536331776, query id 173 localhost root starting
SHOW ENGINE INNODB STATUS
```

4. 一致性概念及其实现

一致性是事务四大特性中最难理解的一个，它关注的是数据的可见性，中间状态（事务在执行过程中，既不是开始，也没有结束）的数据对外是不可见的，只有最初状态和最终状态的数据对外可见。

4.1 什么是一致性

一致性是指事务将数据库从一种一致性状态变为下一种一致性状态，在事务开始之前和之后，数据库的完整性约束（实体完整性、列完整性、外键约束、用户自定义的完整性）没有被破坏。

关于一致性，最经典的例子就是“转账”，即 A 与 B 之间进行转账操作（这是一次事务），无论这中间发生了什么（成功或失败的转账动作），它们的总和必须是不变的。不过，这里还有一层含义：转账操作要么全部成功，要么全部失败，这不是事务的原子性吗？但是，仔细分析，你会发现，原子性和一致性的侧重点是不同的。

4.2 一致性是怎么实现的

之前所介绍的原子性、持久性和隔离性都是为了保证数据库状态的一致性，所以，一致性是数据库事务追求的终极目标。但同时，我们不能仅仅依赖于数据库层面保证一致性，应用层面同样需要做出努力。

实现一致性的方法包括：

- 保证原子性、持久性和隔离性，它们是一致性保证的基础
- 数据库层面的限制，例如插入的字符串长度不能超过列限制（约束）
- 应用代码的判断，校验出错及时抛出异常

5. 总结

支持事务是 InnoDB 最核心的特性之一，而事务的 ACID 四大特性是保障应用程序正确运行的基石。原子性和持久性是最容易理解的两大特性；严格的隔离性会对性能有较大影响，在实际的应用中会相对有所权衡；一致性不仅是数据库自身的完整性要求，同样也对应用程序的开发者带来了挑战。

6. 问题

原子性和一致性的侧重点分别是什么？

你在应用程序中是怎样考虑一致性的？出现问题时，你怎样解决的呢？

你听过分布式事务吗？它是怎么实现的呢？

7. 参考资料

《高性能 MySQL（第三版）》

《MySQL技术内幕：InnoDB存储引擎（第2版）》

MySQL 官方文档：Using InnoDB Transaction and Locking Information

MySQL 官方文档：SHOW ENGINE Statement

MySQL 官方文档：The INFORMATION_SCHEMA INNODB_LOCKS Table

MySQL 官方文档：InnoDB Locking and Transaction Model

MySQL 官方文档：InnoDB In-Memory Structures

MySQL 官方文档：InnoDB On-Disk Structures

}

31 存储是本质，理解 InnoDB 存储引擎也就理所当然

33 为大型电商平台设计高可用数据库系统