

(七)MySQL事务篇：ACID原则、事务隔离级别及事务机制原理剖析

竹子爱熊猫 

2022年10月10日 14:24 · 阅读 2487

已关注

引言

“

本文为掘金社区首发签约文章，14天内禁止转载，14天后未获授权禁止转载，侵权必究！

”

众所周知，MySQL 数据库的核心功能就是存储数据，通常是整个业务系统中最重要的一层，可谓是整个系统的“大本营”，因此只要 MySQL 存在些许隐患问题，对于整个系统而言都是致命的。那此刻不妨思考一个问题：

“

MySQL 在接受外部数据写入时，有没有可能会发生问题呢？

”

有人也许会笑着回答：“那怎么可能啊，MySQL 在写入数据时怎么会存在问题呢”。

的确，MySQL 本身在写入数据时并不会有问题，就算部署 MySQL 的机器断电/宕机，其内部也有一套健全的机制确保数据不丢失。但往往风险并不来自于表象，虽然 MySQL 写入数据没问题，但结合业务来看就会有一个很大的隐患，此话怎讲呐？先看案例：



sql 复制代码

 50 35 收藏

```
INSERT INTO `zz_order` VALUES (...);  
INSERT INTO `zz_order_info` VALUES (...);
```

-- 向物流表中插入相应的物流信息

```
INSERT INTO `zz_logistics` VALUES (...);
```

上述的伪 SQL 中，描述的是一个经典下单业务，先扣库存数量、再增加订单记录、再插入物流信息，按照正常的逻辑来看，上面的 SQL 也没有问题。但是请仔细想想！实际的项目中，这三组 SQL 是会由客户端（Java 线程）一条条发过来的，假设执行到「增加订单记录」时，Java 程序那边抛出了异常，会出现什么问题呢？

“

乍一想似乎没问题，但仔细一想：Java 线程执行时出现异常会导致线程执行中断。

”

因为 Java 线程中断了，所以线程不会再向数据库发送「增加订单详情记录、插入物流信息」的 SQL，此刻再来想想这个场景，由于增加订单详情和物流信息的 SQL 都未发送过来，因此必然也不会执行，但此时库存已经扣了，用户钱也付了，但却没有订单和物流信息，这引发的后果估计老板都能杀个程序员祭天了……

“

其实上面列举的这个案例，在数据库中被称之为事务问题，接下来一起聊一聊。

”



一、事务的ACID原则

什么是事务呢？事务通常是由一个或一组 SQL 组成的，组成一个事务的 SQL 一般都是一个业务操作，例如前面聊到的下单：「扣库存数量、增加订单详情记录、插入物流信息」，这一组 SQL 就可以组成一个事务。

“

而数据库的事务一般也要求满足 ACID 原则，ACID 是关系型数据库实现事务机制时



ACID 主要涵盖四条原则，即：

- A/Atomicity：原子性
- C/Consistency：一致性
- I/Isolation：独立性/隔离性
- D/Durability：持久性

那这四条原则分别是什么意思呢？接下来一起聊一聊。

1.1、Atomicity原子性

原子性这个概念，在之前 [《并发编程系列-JMM内存模型》](#) 时曾初次提到过，而在 MySQL 中原子性的含义也大致相同，指组成一个事务的一组 SQL 要么全部执行成功，要么全部执行失败，事务中的一组 SQL 会被看成一个不可分割的整体，当成一个操作看待。

“

好比事务 A 由 ①、②、③ 条 SQL 组成，那这一个事务中的三条 SQL 必须全部执行成功，只要其中任意一条执行失败，例如 ② 执行时出现异常了，此时就会导致事务 A 中的所有操作全部失败。

”

1.2、Consistency一致性

一致性也比较好理解，也就是不管事务发生的前后，MySQL 中原本的数据变化都是一致的，也就是 DB 中的数据只允许从一个一致性状态变化为另一个一致性状态。这句话似乎听起来有些绕，不太好理解对嘛？简单解释一下就是：一个事务中的所有操作，要么一起改变数据库中的数据，要么都不改变，对于其他事务而言，数据的变化是一致的，上栗子：

“

假设此时有一个事务 A，这个事务隶属于一个下单操作，由「①扣库存数量、②增加订单详情记录、③插入物流信息」这三条 SQL 操作组成。

”



是等于最初的库存总数的，比如原本的总库存是 10000 个，此时库存剩余 8888 个，那也就代表着必须要有 1112 条订单数据才行。

“

这也就是前面说的：“事务发生的前后，MySQL 中原本的数据变化都是一致的”，这句话的含义，不可能库存减了，但订单没有增加，这样就会导致数据库整体数据出现不一致。

”

如果出现库存减了，但订单没有增加的情况，就代表着事务执行过程中出现了异常，此时 MySQL 就会利用事务回滚机制，将之前减的库存再加回去，确保数据的一致性。

“

但来思考一个问题，如果事务执行过程中，刚减完库存后，MySQL 所在的服务器断电了咋整？似乎无法利用事务回滚机制去确保数据一致性了撒？对于这点大可不必担心，因为 MySQL 宕机重启后，会通过分析日志的方式恢复数据，确保一致性（对于这点稍后再细聊）。

”



1.3、Isolation独立性/隔离性



简单理解原子性和一致性后，再来看看 ACID 中的隔离性，在有些地方也称之为独立性，意思就是指多个事务之间都是独立的，相当于每个事务都被装在一个箱子中，每个箱子之间都是隔开的，相互之间并不影响，同样上个栗子：

“

假设数据库的库存表中，库存数量剩余 8888 个，此时有 A、B 两个并发事务，这两个事务都是相同的下单操作，由「①扣库存数量、增②加订单详情记录、③插入物流信息」这三条 SQL 操作组成。

”



障了并发事务的顺序执行，一个未完成事务不会影响另外一个未完成事务。

“

隔离性在底层是如何实现的呢？基于 `MySQL` 的锁机制和 `MVCC` 机制做到的（后续《MySQL事务与锁原理篇》再详细去讲）。

”

1.4、Durability持久性

相较于之前的原子性、一致性、隔离性来说，持久性是 `ACID` 原则中最容易理解的一条，持久性是指一个事务一旦被提交，它会保持永久性，所更改的数据都会被写入到磁盘做持久化处理，就算 `MySQL` 宕机也不会影响数据改变，因为宕机后也可以通过日志恢复数据。

“

也就相当于你许下一个诺言之后，那你无论遇到什么情况都会保证做到，就算遇到山水洪灾、地球毁灭、宇宙爆炸.....任何情况也好，你都会保证完成你的诺言为止。

”

二、MySQL的事务机制综述

刚刚说到的 `ACID` 原则是数据库事务的四个特性，也可以理解为实现事务的基础理论，那接下来一起看看 `MySQL` 所提供的事务机制。在 `MySQL` 默认情况下，一条 `SQL` 会被视为一个单独的事务，同时也无需咱们手动提交，因为默认是开启事务自动提交机制的，如若你想要将多条 `SQL` 组成一个事务执行，那需要显式的通过一些事务指令来实现。

2.1、手动管理事务

在 `MySQL` 中，提供了一系列事务相关的命令，如下：

- `start transaction` | `begin` | `begin work`：开启一个事务

当需要使用事务时，可以先通过 `start transaction` 命令开启一个事务，如下：

```
-- 开启一个事务
start transaction;

-- 第一条SQL语句
-- 第二条SQL语句
-- 第三条SQL语句

-- 提交或回滚事务
commit || rollback;
```

sql 复制代码

对于上述 `MySQL` 手动开启事务的方式，相信大家都不陌生，但大家有一点应该会存在些许疑惑：事务是基于当前数据库连接而言的，而不是基于表，一个事务可以由操作不同表的多条 `SQL` 组成，这句话什么意思呢？看下图：

上面画出了两个数据库连接，假设连接 `A` 中开启了一个事务，那后续过来的所有 `SQL` 都会被加入到一个事务中，也就是图中连接 `A`，后面的 `SQL②`、`SQL③`、`SQL④`、`SQL⑤` 这四条都会被加入到一个事务中，只要在未曾收到 `commit/rollback` 命令之前，这个连接来的所有 `SQL` 都会加入到同一个事务中，因此对于这点要牢记，开启事务后一定要做提交或回滚处理。

“

不过在连接 `A` 中开启事务，是不会影响连接 `B` 的，这也是我说的：事务是基于当前数据库连接的，每个连接之间的事务是具备隔离性的，比如上个真实栗子~

”

此时先打开两个 `cmd` 命令行，然后用命令连接 `MySQL`，或者也可以用 `Navicat`、`SQLyog` 等数据库可视化工具，新建两个查询，如下：

这里插个小偏门知识：当你在 `Navicat`、`SQLyog` 这类可视化工具中，新建一个查询时，本质上它就是给你建立了一个数据库连接，每一个新查询都是一个新的连接。

然后开始在每个查询中编写对应的 `SQL` 命令，并在查询窗口 ① 中开启一个事务。





首页 ▾

探

Q



```
SELECT * FROM zz_users ;
```

user_id	user_name	user_sex	password	register_time
1	熊猫	女	6666	2022-08-14 15:22:01
2	竹子	男	1234	2022-09-14 16:17:44
3	子竹	男	4321	2022-09-16 07:42:21
4	1111	男	8888	2022-09-17 23:48:29

-- 开启事务

```
start transaction;
```

-- 修改 ID=4 的姓名为：黑熊

```
update `zz_users` set `user_name` = "黑熊" where `user_id` = 4;
```

-- 删除 ID=1 的行数据

```
delete from `zz_users` where `user_id` = 1;
```

-- 再次查询一次数据

```
SELECT * FROM `zz_users`;
```

user_id	user_name	user_sex	password	register_time
2	竹子	男	1234	2022-09-14 16:17:44
3	子竹	男	4321	2022-09-16 07:42:21
4	黑熊	男	8888	2022-09-17 23:48:29

观察上面的结果，对比开启事务前后的的表数据查询，在事务中分别修改、删除一条数据后，再次查询表数据时会观察到表数据已经变化，此时再去查询窗口 ② 中查询表数据：



sql 复制代码

```
SELECT * FROM `zz_users`;
```

user_id	user_name	user_sex	password	register_time
1	熊猫	女	6666	2022-08-14 15:22:01
2	竹子	男	1234	2022-09-14 16:17:44
3	子竹	男	4321	2022-09-16 07:42:21
4	1111	男	8888	2022-09-17 23:48:29



50



35



收藏

接改变的数据不会影响第二个连接。

“

其实具体的原因是由于 `MySQL` 事务的隔离机制造成的，但对于这点后续再去分析。

”

此时在查询窗口 ① 中，输入 `rollback` 命令，让当前事务回滚：



sql 复制代码

```
-- 回滚当前连接中的事务
```

```
rollback;
```

```
-- 再次查询表数据
```

```
SELECT * FROM `zz_users`;
```

user_id	user_name	user_sex	password	register_time
1	熊猫	女	6666	2022-08-14 15:22:01
2	竹子	男	1234	2022-09-14 16:17:44
3	子竹	男	4321	2022-09-16 07:42:21
4	1111	男	8888	2022-09-17 23:48:29

结果很明显，当事务回滚后，之前所做的数据更改操作全部都会撤销，恢复到事务开启前的表数据。当然，如果不手动开启事务，执行下述这条 `SQL` 会发生什么情况呢？



sql 复制代码

```
update `zz_users` set `user_name` = "黑熊" where `user_id` = 4;
```

会直接修改表数据，并且其他连接可见，因为 `MySQL` 默认将一条 `SQL` 视为单个事务，同时默认开启自动提交事务，也就是上面这条 `SQL` 执行完了之后就会自动提交。



sql 复制代码

```
-- 查看 自动提交事务 是否开启
```

```
SHOW VARIABLES LIKE 'autocommit';
```

```
-- 关闭或开启自动提交
```

```
SET autocommit = 0|1|ON|OFF;
```



2.2、事务回滚点

在上面简单阐述了事务的基本使用，但假设目前有一个事务，由很多条 SQL 组成，但是我想让其中一部分执行成功后，就算后续 SQL 执行失败也照样提交，这样可以做到吗？从前面的理论上来看，一个事务要么全部执行成功，要么全部执行失败，似乎做不到啊，但实际上是可以做到的，这里需要利用事务的回滚点机制。

“

在某些 SQL 执行成功后，但后续的操作有可能成功也有可能失败，但不管成功亦或失败，你都想让前面已经成功的操作生效时，此时就可在当前成功的位置设置一个回滚点。当后续操作执行失败时，就会回滚到该位置，而不是回滚整个事务中的所有操作，这个机制则称之为事务回滚点。

”

在 MySQL 中提供了两个关于事务回滚点的命令：

- `savepoint point_name`：添加一个事务回滚点
- `rollback to point_name`：回滚到指定的事务回滚点

以前面的案例来演示效果，如下：

```
sql 复制代码

-- 先查询一次用户表
SELECT * FROM `zz_users`;

-- 开启事务
start transaction;

-- 修改 ID=4 的姓名为：黑熊
update `zz_users` set `user_name` = "黑熊" where `user_id` = 4;

-- 添加一个事务回滚点：update_name
savepoint update_name;

-- 删除 ID=1 的行数据
delete from `zz_users` where `user_id` = 1;

-- 回滚到 update_name 这个事务点
rollback to update_name;

-- 再次查询一次数据
SELECT * FROM `zz_users`;

-- 提交事务
```

“

但要注意：回滚到事务点后不代表着事务结束了，只是事务内发生了一次回滚，如果要结束当前这个事务，还依旧需要通过 `commit|rollback;` 命令处理。

”

其实借助事务回滚点，可以很好的实现失败重试，比如对事务中的每个 SQL 添加一个回滚点，当执行一条 SQL 时失败了，就回滚到上一条 SQL 的事务点，接着再次执行失败的 SQL，反复执行到所有 SQL 成功为止，最后再提交整个事务。

“

当然，这个只是理论上的假设，实际业务中不要这么干~

”



2.3、MySQL事务的隔离机制



OK~，在前面做的小测试中，咱们会发现不同的数据库连接中，一个连接的事务并不会影响其他连接，当时也稍微的提过一嘴：这是基于事务隔离机制实现的，那接下来重点聊一聊 MySQL 的事务隔离机制。其实在 MySQL 中，事务隔离机制分为了四个级别：

- ① `Read uncommitted/RU`：读未提交
- ② `Read committed/RC`：读已提交
- ③ `Repeatable read/RR`：可重复读
- ④ `Serializable`：序列化/串行化

上述四个级别，越靠后并发控制度越高，也就是在多线程并发操作的情况下，出现问题的几率越小，但对应的也性能越差，MySQL 的事务隔离级别，默认为第三级别：`Repeatable read` 可重复读，但如若想要真正理解这几个隔离级别，得先明白几个因为并发操作造成的问题。

2.3.1、脏读、幻读、不可重复读问题

首先来看看脏读，脏读的意思是指一个事务读到了其他事务还未提交的数据，也就是当前事务读到的数据，由于还未提交，因此有可能会回滚，如下：

比如上图中，DB 连接①/事务 A 正在执行下单业务，目前扣减库存、增加订单两条 SQL 已经完成了，恰巧此时 DB 连接②/事务 B 跑过来读取了一下库存剩余数量，就将事务 A 已经扣减之后的库存数量读回去了。但好巧不巧，事务 A 在添加物流信息时，执行异常导致事务 A 全部回滚，也就是原本扣的库存又会增加回去。

“

在个案例中，事务 A 先扣减了库存，然后事务回滚时又加了回去，但连接②已经将扣减后的库存数量读回去操作了，这个过程就被称为数据库脏读问题。这个问题很严重，会导致整个业务系统出现问题，数据最终错乱。

”

数据库的不可重复读问题

再来看看不可重复读问题，不可重复读问题是指在一个事务中，多次读取同一数据，先后读取到的数据不一致，如下：

你没看错，就是对前面那张图稍微做了一点改造，事务 A 执行下单业务时，因为添加物流信息的时候出错了，导致整个事务回滚，事务回滚完成后，事务 A 就结束了。但事务 B 却并未结束，在事务 B 中，在事务 A 执行时读取了一次剩余库存，然后在事务回滚后又读取了一次剩余库存，仔细想想：B 事务第一次读到的剩余库存是扣减之后的，第二次读到的剩余库存则是扣减之前的（因为 A 事务回滚又加回去了）。

“

在上述这个案例中，同一个事务中读取同一数据，结果却并不一致，也就说明了该数

结合上述可重复读的定义，再去理解不可重复读问题会容易很多，重点是理解可重复、不可重复这个词义，为了更形象化一点，举个生活中的案例：

“

一张卫生纸，我先拿去擦了一下桌子上的污水渍，然后又放回了原位，当我想上厕所再次拿起时，它已经无法使用了，这就代表着一张卫生纸是不可重复使用的。

”

“

一个大铁锤，我先拿去敲一下松掉的桌腿，然后放回了原位，当我又想敲一下墙上的钉子再次拿起时，这个大铁锤是没有发生任何变化的，可以再次用来敲钉子，这就代表大铁锤是可以重复使用的。

”

相信结合这两个栗子，更能让你明白可重复与不可重复的概念定义。

数据库的幻读问题

对于幻读的解释在网上也有很多资料，但大部分资料是这样描述幻读问题的：

“

幻读：指同一个事务内多次查询返回的结果集不一样。比如同一个事务 A，在第一次查询表的数据行数时，发现表中有 n 条行记录，但是第二次以同等条件查询时，却发现 n+1 条记录，这就好像产生了幻觉。

”



做过电商业务的小伙伴都清楚，一般用户购买商品后付的钱会先冻结在平台上，然后由平台在固定的时间内结算用户款，例如七天一结算、半月一结算等方式，在结算业务中通常都会涉及到核销处理，也就是将所有为「已签收状态」的订单改为「已核销状态」。

“

此时假设连接①/事务 A 正在执行「半月结算」这个工作，那首先会读取订单表中所有状态为「已签收」的订单，并将其更改为「已核销」状态，然后将用户款打给商家。

”

但此时恰巧，某个用户的订单正好到了自动确认收货的时间，因此在事务 A 刚刚改完表中订单的状态时，事务 B 又向表中插入了一条「已签收状态」的订单并提交了，当事务 A 完成打款后，再次查询订单表，结果会发现表中还有一条「已签收状态」的订单数据未结算，这就好像产生了幻觉一样，这才是真正的幻读问题。

“

当然，这样讲似乎还不是很令人理解，再举个更通俗易懂的栗子，假设此时平台要升级，用户表中的性别字段，原本是以「男、女」的形式保存数据，现在平台升级后要求改为「0、1」代替。

因此事务 A 开始更改表中所有数据的性别字段，当负责执行事务 A 的线程正在更改最后一条表数据时，此时事务 B 来了，正好向用户表中插入了一条「性别=男」的数据并提交了，然后事务 A 改完原本的最后一条数据后，当再次去查询用户表时，结果会发现表中依旧还存在一条「性别=男」的数据，似乎又跟产生了幻觉一样。

”

经过上述这两个案例，大家应该能够理解真正的幻读问题，发生幻读问题的原因是在于：另外一个事务在第一个事务要处理的目标数据范围之内新增了数据，然后先于第一个事务提交造成的问题。

数据库脏写问题



的问题，这个问题也被称之为更新丢失问题。

2.3.2、事务的四大隔离级别

在上面连续讲了脏读、不可重复读以及幻读三个问题，那这些问题该怎么解决呢？其实四个事务隔离级别，解决的实际问题就是这三个，因此一起来看看各级别分别解决了什么问题：

- ①读未提交：处于该隔离级别的数据库，脏读、不可重复读、幻读问题都有可能发生。
- ②读已提交：处于该隔离级别的数据库，解决了脏读问题，不可重复读、幻读问题依旧存在。
- ③可重复读：处于该隔离级别的数据库，解决了脏读、不可重复读问题，幻读问题依旧存在。
- ④序列化/串行化：处于该隔离级别的数据库，解决了脏读、不可重复读、幻读问题都不存在。

前面提到过，MySQL 默认是处于第三级别的，可以通过如下命令查看目前数据库的隔离级别：

sql 复制代码

```
-- 查询方式①
SELECT @@tx_isolation;

-- 查询方式②
show variables like '%tx_isolation%';

+-----+-----+
| Variable_name | Value                |
+-----+-----+
| tx_isolation  | REPEATABLE-READ     |
+-----+-----+
```

其实数据库不同的事务隔离级别，是基于不同类型、不同粒度的锁实现的，因此想要真正搞懂隔离机制，还需要弄明白MySQL的锁机制，事务与锁机制二者之间本身就是相辅相成的关系，锁是为了解决并发事务的一些问题而存在的，但对于锁的内容在后续的《MySQL锁篇》再细聊，这里就简单概述一下。

这里先说明一点，事务是基于数据库连接的，数据库连接在《MySQL架构篇》中曾说过：数据库连接本身会有一条工作线程来维护，也就是说事务的执行本质上就是工作线程在执行，因此所谓的并发事务也就是指多条线程并发执行。

多线程其实是咱们的老朋友了，在之前的 [《并发编程系列》](#) 中，几乎将多线程的底裤都翻出来了，因此结合多线程角度来看，脏读、不可重复读、幻读这一系列问题，本质上就是一些线程安全问题，因此需要通过锁来解决，而根据锁的粒度、类型，又分出了不同的事务隔离级别。



读未提交级别

这种隔离级别是基于「写互斥锁」实现的，当一个事务开始写某一个数据时，另外一个事务也来操作同一个数据，此时为了防止出现问题则需要先获取锁资源，只有获取到锁的事务，才允许对数据进行写操作，同时获取到锁的事务具备排他性/互斥性，也就是其他线程无法再操作这个数据。



但虽然这个级别中，写同一数据时会互斥，但读操作却并不是互斥的，也就是当一个事务在写某个数据时，就算没有提交事务，其他事务来读取该数据时，也可以读到未提交的数据，因此就会导致脏读、不可重复读、幻读一系列问题出现。



但是由于在这个隔离级别中加了「写互斥锁」，因此不会存在多个事务同时操作同一数据的情况，因此这个级别中解决了前面说到的脏写问题。

读已提交级别

在这个隔离级别中，对于写操作同样会使用「写互斥锁」，也就是两个事务操作同一事务时，会出现排他性，而对于读操作则使用了一种名为 **MVCC** 多版本并发控制的技术处理，也就是有事务中的 **SQL** 需要读取当前事务正在操作的数据时，**MVCC** 机制不会让另一个事务读取正在修改的数据，而是读取上一次提交的数据（也就是读原本的老数据）。

解还是简单的说一下其过程，同样有两个事务 A、B。



事务 A 的主要工作是负责更新 ID=1 的这条数据，事务 B 中则是读取 ID=1 的这条数据。此时当 A 正在更新数据但还未提交时，事务 B 开始读取数据，此时 MVCC 机制则会基于表数据的快照创建一个 ReadView，然后读取原本表中上一次提交的老数据。然后等事务 A 提交之后，事务 B 再次读取数据，此时 MVCC 机制又会创建一个新的 ReadView，然后读取到最新的已提交的数据，此时事务 B 中两次读到的数据并不一致，因此出现了不可重复读问题。

当然，对于 MVCC 机制以及锁机制这里暂时先不展开叙述，后续会开单章讲解。

可重复读级别

在这个隔离级别中，主要就是解决上一个级别中遗留的不可重复读问题，但 MySQL 依旧是利用 MVCC 机制来解决这个问题的，只不过在这个级别的 MVCC 机制会稍微有些不同。在读已提交级别中，一个事务中每次查询数据时，都会创建一个新的 ReadView，然后读取最近已提交的事务数据，因此就会造成不可重复读的问题。

“

而在可重复读级别中，则不会每次查询时都创建新的 ReadView，而是在一个事务中，只有第一次执行查询会创建一个 ReadView，在这个事务的生命周期内，所有的查询都会从这一个 ReadView 中读取数据，从而确保了一个事务中多次读取相同数据是一致的，也就是解决了不可重复读问题。

”

虽然在这个隔离级别中，解决了不可重复读问题，但依旧存在幻读问题，也就是事务 A 在对表中多行数据进行修改，比如前面的举例，将性别「男、女」改为「0、1」，此时事务 B 又插入了一条性别为男的数据，当事务 A 提交后，再次查询表时，会发现表中依旧存在一条性别为男的数据。



上就可以得知：序列化意思是将所有的事务按序排队后串行化处理，也就是操作同一张表的事务只能一个一个执行，事务在执行前需要先获取表级别的锁资源，拿到锁资源的事务才能执行，其余事务则陷入阻塞，等待当前事务释放锁。

“

但这种隔离级别会导致数据库的性能直线下降，毕竟相当于一张表上只能允许单条线程执行了，虽然安全等级最高，可以解决脏写、脏读、不可重复读、幻读等一系列问题，但也是代价最高的，一般线上很少使用。

”

这种隔离级别解决问题的思想很简单，之前我们分析过，产生一系列问题的根本原因在于：多事务/多线程并发执行导致的，那在这个隔离级别中，直接将多线程化为了单线程，自然也就从根源上避免了问题产生。

“

是不是非常“银杏花”，虽然我解决不了问题，但我可以直接解决制造问题的人。

”

略微提一嘴：其实在 `RR` 级别中也可以解决幻读问题，就是使用临键锁（间隙锁+行锁）这种方式来加锁，但具体的还是放在《MySQL锁篇》详细阐述。

2.3.3、事务隔离机制的命令

简单认识 MySQL 事务隔离机制后，接着来看看一些关于事务隔离机制的命令：



sql 复制代码

```
-- 方式①：查询当前数据库的隔离级别
SELECT @@tx_isolation;

-- 方式②：查询当前数据库的隔离级别
show variables like '%tx_isolation%';
```



50



35



收藏



-- 这里和上述的那条命令作用相同，是第二种设置的方式

```
set tx_isolation = 'repeatable-read';
```

-- 设置隔离级别为最高的serializable级别（全局生效）

```
set global.tx_isolation = 'serializable';
```

上述实际上一眼就能看懂，唯一要注意的在于：如果想要让设置的隔离级别在全局生效，一定要记得加上 `global` 关键字，否则生效范围是当前会话，也就是针对于当前数据库连接有效，在其他连接中依旧是原本的隔离级别。

三、MySQL的事务实现原理

到这里为止，一些 MySQL 事务相关的概念和基础就已经讲明白了，现在重点来聊一聊 MySQL 事务究竟是怎么实现的呢？先把结论抛出来：「MySQL 的事务机制是基于日志实现的」。为什么是基于日志实现的呢？一起来展开聊一聊。

3.1、正常SQL的事务机制

在前面聊到过的一点：「MySQL 默认开启事务的自动提交，并且将一条 SQL 视为一个事务」。那 MySQL 在何种情况下会将事务自动提交呢？什么情况下又会自动回滚呢？想要弄明白这个问题，首先得回顾一下之前讲过的《SQL执行篇-写入SQL的执行流程》，在讲写入类型 SQL 的执行流程时，曾讲过一点：任意一条写 SQL 的执行都会记录三个日志：`undo-log`、`redo-log`、`bin-log`。

- `undo-log`：主要记录 SQL 的撤销日志，比如目前是 `insert` 语句，就记录一条 `delete` 日志。
- `redo-log`：记录当前 SQL 归属事务的状态，以及记录修改内容和修改页的位置。
- `bin-log`：记录每条 SQL 操作日志，只要是用于数据的主从复制与数据恢复/备份。

在写 SQL 执行记录的三个日志中，`bin-log` 暂且不需要关心，这个跟事务机制没关系，重点是 `undo-log`、`redo-log` 这两个日志，其中最重要的是 `redo-log` 这个日志。

“

`redo-log` 是一种 WAL(Write-ahead logging) 预写式日志，在数据发生更改之前会先记录日志，也就是在 SQL 执行前会先记录一条 `prepare` 状态的日志，然后再执行数据的写操作。





首页 ▾

探

Q



InnoDB 引擎中不会直接将数据写入到磁盘文件中，而是会先写到 BufferPool 缓冲区中，当 SQL 被成功写入到缓冲区后，紧接着会将 redo-log 日志中相应的记录改为 commit 状态，然后再由 MySQL 刷盘机制去做具体的落盘操作。

“

因为默认情况下，一条 SQL 会被当成一个事务，数据写入到缓冲区后，就代表执行成功，因此会自动修改日志记录为 commit 状态，后续则会由 MySQL 的后台线程执行刷盘动作。

”

举个伪逻辑的例子，例如下述这条插入 SQL 的执行过程大致如下：



sql 复制代码

```
-- 先记录一条状态为 prepare 的日志
-- 然后执行SQL，在缓冲区中更改对应的数据
INSERT INTO `zz_users` VALUES (5,"黑竹","男","9999","2022-09-24 23:48:29");
-- 写入缓冲区成功后，将日志记录改为 commit状态
-- 返回 [Affected rows: 1]，MySQL后台线程执行刷盘动作
```

一条 SQL 语句组成的事务，其执行过程是不是很容易理解~，接着来看看手动开启事务的实现。



3.2、多条SQL的事务机制



先把前面的案例搬下来，如下：



sql 复制代码

```
-- 开启事务
start transaction;
-- 修改 ID=4 的姓名为：黑熊（原本user_name = 1111）
update `zz_users` set `user_name` = "黑熊" where `user_id` = 4;
-- 删除 ID=1 的行数据
delete from `zz_users` where `user_id` = 1;
-- 提交事务
COMMIT;
```



50



35



收藏



①当 MySQL 执行时，碰到 `start transaction;` 的命令时，会将后续所有写操作全部先关闭自动提交机制，也就是后续的所有写操作，不管有没有成功都不会将日志记录修改为 `commit` 状态。

②先在 `redo-log` 中为第一条 SQL 语句，记录一条 `prepare` 状态的日志，然后再生成对应的撤销日志并记录到 `undo-log` 中，然后执行 SQL，将要写入的数据先更新到缓冲区。

③再对第二条 SQL 语句做相同处理，如果有更多条 SQL 则逐条依次做相同处理.....，这里简单的说一下撤销日志长啥样，大致如下：



sql 复制代码

```
-- 第一条修改SQL的撤销日志（将修改的姓名字段从 黑熊 改回 1111）
update `zz_users` set `user_name` = "1111" where `user_id` = 4;
-- 第二条删除SQL的撤销日志（将删除的行数据再次插入）
INSERT INTO `zz_users` VALUES (1,"熊猫","女","6666","2022-08-14 15:22:01");
```

④直到碰到了 `rollback`、`commit` 命令时，再对前面的所有写 SQL 做相应处理：



如果是 `commit` 提交事务的命令，则先将当前事务中，所有的 SQL 的 `redo-log` 日志改为 `commit` 状态，然后由 MySQL 后台线程做刷盘，将缓冲区中的数据落入磁盘存储。



如果是 `rollback` 回滚事务的命令，则在 `undo-log` 日志中找到对应的撤销 SQL 执行，将缓冲区内更新过的数据全部还原，由于缓冲区的数据被还原了，因此后台线程在刷盘时，依旧不会改变磁盘文件中存储的数据。



OK~，其实事务机制的底层实现也并不麻烦，稍微一推导、一思考就能想明白的道理。



当然，大家有兴趣的再去推导一下：事务撤销点是怎么实现的呢？其实也并不难的，略加思考即可以得到答案。



3.3、事务的恢复机制



现在再来思考一个问题，有没有这么一种可能呢？也就是当 SQL 执行时，数据还没被刷写到磁盘中，结果数据库宕机了，那数据是不是就丢了啊？毕竟本地磁盘中的数据，在 MySQL 重启后依旧存在，但缓冲区中还未被刷到磁盘的数据呢？因为缓冲区位于内存中，所以里面的数据重启是不会存在的撒？

“

对于这个问题呢实际上并不需要担心，因为前面聊到过 redo-log 是一种预写式日志，会先记录日志再去更新缓冲区中的数据，所以就算缓冲区的数据未被刷写到磁盘，在 MySQL 重启时，依旧可以通过 redo-log 日志重新恢复未落盘的数据，从而确保数据的持久化特性。

”

当然，有人或许又会问：那如果在记录 redo-log 日志时，MySQL 芭比Q了咋整？如果遇到了这个问题呢，首先得恭喜你，你的运气属于很棒，能碰到这个问题的几率足够你买彩票中五百万了~

“

玩笑归玩笑，现在回归话题本身，这个问题总不能让它存在是不？毕竟有这个问题对于系统而言也是个隐患啊，但仔细一思考，其实这个问题不必多虑，为啥？推导一下。

”

首先看看前面的那种情况：「数据被更新到缓冲区但没刷盘，然后 MySQL 宕机了，MySQL 会通过日志恢复数据」。这里要注意的是：数据被更新到缓冲区代表着 SQL 执行成功了，此时客户端会收到 MySQL 返回的写入成功提示，只是没有落盘而言，所以 MySQL 重启后只需要再次



但如果在记录日志的时候 MySQL 宕机了，这代表着 SQL 都没执行成功，SQL 没执行成功的话，MySQL 也不会向客户端返回任何信息，因为 MySQL 一直没返回执行结果，因此会导致客户端连接超时，而一般客户端都会有超时补偿机制的，比如会超时后重试，如果 MySQL 做了热备/灾备，这个重试的时间足够 MySQL 重启完成了，因此用户的操作依旧不会丢失（对于超时补偿机制，在各大数据库连接池中是有实现的）。



但如若又有小伙伴纠结：我 MySQL 也没做热备/灾备这类的方案呐，此时咋整呢？



如果是这样的情况，那就只能自认倒霉了，毕竟 MySQL 挂了一直不重启，不仅仅当前的 SQL 会丢失，后续平台上所有的用户操作都会无响应，这属于系统崩溃级别的灾难了，因此只能靠完善系统架构来解决。



四、MySQL事务篇总结

一点点看到这里，《MySQL事务篇》也就接近了尾声，在本篇中对事务机制一点点去引出，慢慢的到事务机制的概述、并发事务的问题、事务的隔离级别、事务的实现原理等诸多方面进行了全面剖析，但大家应该也略微有些不尽兴，毕竟对于隔离级别的具体实现并未讲到，这是由于 MySQL 事务与锁机制之间有着千丝万缕的关系，所以在《MySQL锁篇》中会再次详细讲到事务隔离机制的。



当然，由于目前是分布式/微服务架构横行的时代，所以也引出了新的问题，即「**分布式事务问题**」，这个问题又需要通过全新的事务机制去处理了，对于这点再讲完《MySQL分库分表》后，会再单开一章《分布式事务篇》去详细阐述，这里头的学问很大~





- 原子性要求事务中所有操作要么全部成功，要么全部失败，这点是基于 `undo-log` 来实现的，因为在该日志中会生成相应的反 `SQL`，执行失败时会利用该日志来回滚所有写入操作。
- 持久性要求的是所有 `SQL` 写入的数据都必须能落入磁盘存储，确保数据不会丢失，这点则是基于 `redo-log` 实现的，具体的实现过程在前面事务恢复机制讲过。
- 隔离性的要求是一个事务不会受到另一个事务的影响，对于这点则是通过锁机制和 `MVCC` 机制实现的，只不过 `MySQL` 屏蔽了加锁和 `MVCC` 的细节，具体的会在后续章节中细聊。
- 一致性要求数据库的整体数据变化，只能从一个一致性状态变为另一个一致性状态，其实前面的原子性、持久性、隔离性都是为了确保这点而存在的。

分类：

后端

标签：

数据库

MySQL

Java

文章被收录于专栏：

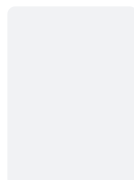


全解MySQL数据库

从MySQL整体架构出发，到SQL优化、MySQL索引、慢查询优化...

关注专栏

相关课程



VIP SpringBoot 源码解读与原理分析

LinkedBear LV.3

3613购买

¥39.9



VIP Hello, 分布式锁

编程界的小... LV.2

1507购买

¥29.9

评论

输入评论 (Enter换行, Ctrl + Enter发送)





codcod Java后端开发工程师

1天前

不可重复读这里的例子是不是应该改成提交事务成功呢，一般将不可重复读的情况下都是在解决脏读的问题，理解为事务隔离级别未提交读。那这时候事务A如果时回滚的话，事务B第一次读取库存和第二次应该是一样的。

点赞 1



竹子爱熊猫 (作者)

1小时前

不可重复读的含义是：在一次事务中多次读取同一数据，每次读出来的事务都不一致。

但我这个确实是基于RU级别画的图，RC级别下就是一条数据被多个事务反复修改，并各自提交了事务，因此导致一个事务在读的时候，多次读取的数据不一致，你理解意思就好啦😂，图我就不画了，之前的草图删掉了，重新画有点费劲。

点赞 回复



辰风

4天前

2.1 这里是不是应该是：后面的SQL②、SQL③、SQL④、SQL⑤这四条都会被加入到一个事务中呀

点赞 1



竹子爱熊猫 (作者)

4天前

对的，手误打错了，我这边修正一下。

点赞 回复

iBUYPOWER

4天前

卧槽 写的真好

点赞 1



竹子爱熊猫 (作者)

4天前

三克油👍👍

点赞 回复

BinarnZx 后端开发 @ 自己跳动

7天前

50

35

收藏



竹子爱熊猫 (作者)

7天前

哈哈，提前谢过啦，但掘金没有收费专栏的概念，也包括我自己的技术分享文章不会搞成收费形式的，能够帮到你就好啦~

但后续可能出一本关于面试的小册，有兴趣的话你可以持续关注 🤔

👍 点赞 💬 回复

BinaryZx 回复 竹子爱熊猫

6天前

必须的 🤔

“哈哈，提前谢过啦，但掘金没有收费专栏的概念，也包括我自己的技术分享文...”

👍 点赞 💬 回复

BinaryZx   后端开发 @ 自己跳动

7天前

太清晰了，帮我理清了好多东西，感谢大佬

👍 点赞 💬 1



竹子爱熊猫 (作者)

7天前

三克油~，感谢认可 🤔

👍 点赞 💬 回复

JAVA旭阳   JAVA开发工程师

10天前

写的条理逻辑都很清晰，讲的很明白了

👍 点赞 💬 1



竹子爱熊猫 (作者)

10天前

三克油~，感谢认可

👍 点赞 💬 回复

青衣白马 

10天前

大佬，对于那些写入缓冲区但是没有刷盘的sql，redo-log中是有其他的标签来标记吗

👍 50

💬 35

☆ 收藏



首页 ▾

探



这里呢，内容就牵扯有些多了，写入了缓冲区但没落盘的数据，在内存中会被标记为脏页或者叫做变更页，然后会由专门的后台线程来刷盘，当一个事务的写操作全部完成并提交后，redo-log、bin-log中会写入一条事务已提交的数据日志，主要是用来做灾难恢复和数据同步的。

如果你对于脏页、变更页，还有对于日志这些不太熟悉的话，你可以跟着专栏慢慢往后...
[展开](#)

👍 1 💬 回复



青衣白马 回复 竹子爱熊猫

5天前

好的，谢谢大佬

“这里呢，内容就牵扯有些多了，写入了缓冲区但没落盘的数据，在内存中会被标...”

👍 点赞 💬 回复

[查看更多回复 ▾](#)



用户474437536...

1月前

清晰，全面，受教了。👍👍👍

👍 点赞 💬 1



竹子爱熊猫 (作者)

1月前

三克油~，感谢认可。

👍 点赞 💬 回复



那个学java的上...

1月前

我才几天没看，就更新这么多，肝！

👍 点赞 💬 1



竹子爱熊猫 (作者)

1月前

哈哈，有十天左右没来了~🤔


👍 点赞 💬 回复

👍 50



💬 35


☆ 收藏

 点赞  1

 竹子爱熊猫 （作者）1月前


三克油~，感谢认可。

 点赞  回复



觅忘勿念 1月前

打卡，

 点赞  1

 竹子爱熊猫 （作者）1月前


欢迎~

 点赞  回复



 借故 1月前

打卡

 点赞  1

 竹子爱熊猫 （作者）1月前


欢迎~


 点赞  回复



 狗霸天  后端开发1月前

先赞后看~

 点赞  1

 竹子爱熊猫 （作者）1月前

三克油~，哈哈 

 点赞  回复



点赞 1



竹子爱熊猫 (作者)

1月前

😂事务这块还是比较简单的，主要弄明白undo-log日志，就能理解事务的实现原理，就是后续的事务隔离机制，实现的过程稍微复杂一点点，用到了锁、mvcc机制，但总体来说也不难的，加油

1 回复

StoneDB 数据库架构师 @ 石原...

1月前

大佬，你太能肝了吧

2 1



竹子爱熊猫 (作者)

1月前

这是签约的原因，所以更新频率会在三天一篇，后续其他专栏的速度估计没这么快😂。

2 回复



CC_LKL

1月前

事务点是不是基于undo log的版本链机制实现的呀？

1 1



竹子爱熊猫 (作者)

1月前

对的👍

点赞 回复

相关推荐

Java中文社群 | 1月前 | 后端 · 掘金·日新计划 · Java

面试突击89：事务隔离级别和传播机制有什么区别？

👁 3843 18 4

50

35

☆ 收藏



首页 ▾

探



1403 40 24

竹子爱熊猫 | 1月前 | 数据库 · MySQL · Java

(五)MySQL索引应用篇：建立索引的正确姿势与使用索引的最佳指南！

5388 74 58

竹子爱熊猫 | 25天前 | 数据库 · MySQL · Java

(十二)MySQL之内存篇：深入探寻数据库内存与Buffer Pool的奥妙！

1404 25 29

竹子爱熊猫 | 2月前 | Java · 数据库 · MySQL

(二)全解MySQL：一条SQL语句从诞生至结束的多姿多彩历程！

3897 81 36

Java中文社群 | 2月前 | 后端 · 面试 · Java

面试突击84：Spring 有几种事务隔离级别？

5934 16 3

Melo_ | 3月前 | Java · 后端 · MySQL

「MySQL高级篇」MySQL之MVCC实现原理&&事务隔离级别的实现

3281 46 3

~藕爸~ | 4年前 | 数据库 · 后端 · 架构

10亿级订单系统分库分表设计思路！

2.4w 428 22

Java中文社群 | 4月前 | 后端 · 面试 · Java

面试突击61：说一下MySQL事务隔离级别？

5694 28 3

Java3y | 4年前 | 数据库 · MySQL · HTTPS

数据库两大神器【索引和锁】

9.8w 1179 69

50

35

收藏



首页 ▾

探



5.9w 1179 41

咖啡拿铁 | 4年前 | 后端 · 数据库 · 微服务

再有人问你分布式事务，把这篇扔给他

👁 12.7w 🍎 1161 💬 61

竹子爱熊猫 | 1月前 | MySQL · 数据库 · Java

(十)全解MySQL之死锁问题分析、事务隔离与锁机制的底层原理剖析

👁 2931 🍎 55 💬 26

民工哥技术之路 | 4年前 | 数据库 · 服务器 · MySQL

MySQL 分库分表方案，总结的非常好！

👁 4.8w 🍎 166 💬 9

芋道源码_芳芳 | 5年前 | MySQL · Java · 架构

JDBC PreparedStatement 实现原理【推荐阅读】

👁 4338 🍎 43 💬 评论

竹子爱熊猫 | 1月前 | Java · 数据库 · MySQL

(三)MySQL之库表设计篇：一、二、三、四、五范式、BC范式与反范式详解！

👁 4573 🍎 55 💬 29

bojiangzhou | 1年前 | MySQL · 后端

MySQL系列 (9) — 事务隔离性之MVCC

👁 2111 🍎 12 💬 2

美得让人心动 | 3年前 | Java · 架构 · MySQL

BAT等一线互联网公司中，Java开发的招聘标准

👁 629 🍎 点赞 💬 评论

大闲人柴毛毛 | 4年前 | 架构 · 数据库 · 微服务

常用的分布式事务解决方案

👁 6.3w 🍎 329 💬 26

🍎 50

💬 35

★ 收藏



2.1W 360 36

友情链接：

Vivaro新能源 Mahindra Pik Up LEVC LCV ALFA G1 说说你对闭包的理解？ 闭包 最简单的服务响应时长优化
OpenKruise v SSH框架“Hibern 以OneFlow为例探索 从小白开始的编程体验 (2

