

# MYSQL 事务的底层原理 | 京东物流技术团队

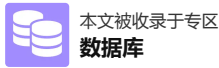
原创

京东云开发者

数据库

2023/11/14 10:05

阅读数 5.2K



进入专区参与更多专题讨论 >

## 事务的底层原理

在事务的实现机制上，MySQL 采用的是 WAL：Write-ahead logging，预写式日志，机制来实现的。

在使用 WAL 的系统中，所有的修改都先被写入到日志中，然后再被应用到系统中。通常包含 redo 和 undo 两部分信息。

为什么需要使用 WAL，然后包含 redo 和 undo 信息呢？举个例子，如果一个系统直接将变更应用到系统状态中，那么在机器掉电重启之后系统需要知道操作是成功了，还是只有部分成功或者是失败了。如果使用了 WAL，那么在重启之后系统可以通过比较日志和系统状态来决定是继续完成操作还是撤销操作。

redo log 称为重做日志，每当有操作时，在数据变更之前将操作写入 redo log，这样当发生掉电之类的情況时系统可以在重启后继续操作。

undo log 称为撤销日志，当一些变更执行到一半无法完成时，可以根据撤销日志恢复到变更之间的状态。

MySQL 中用 redo log 来在系统 Crash 重启之类的情況时修复数据，而 undo log 来保证事务的原子性。

## 事务 id

一个事务可以是一个只读事务，或者是一个读写事务：可以通过 START TRANSACTION READ ONLY 语句开启一个只读事务。

在只读事务中不可以对普通的表进行增、删、改操作，但可以对用户临时表做增、删、改操作。

可以通过 START TRANSACTION READ WRITE 语句开启一个读写事务，或者使用 BEGIN、START TRANSACTION 语句开启的事务默认也算是读写事务。

在读写事务中可以对表执行增删改查操作。

**如果某个事务执行过程中对某个表执行了增、删、改操作，那么 InnoDB 存储引擎就会给它分配一个独一无二的 事务 id，针对 MySQL 5.7 分配方式如下：**

- 对于只读事务来说，只有在它第一次对某个用户创建的临时表执行增、删、改操作时才会为这个事务分配一个事务 id，否则的话是不分配事务 id 的。
- 对于读写事务来说，只有在它第一次对某个表执行增、删、改操作时才会为这个事务分配一个事务 id，否则的话也是不分配事务 id 的。
- 有的时候虽然开启了一个读写事务，但是在这个事务中全是查询语句，并没有执行增、删、改的语句，那也就意味着这个事务并不会被分配一个事务 id。

**这个事务 id 本质上就是一个数字，它的分配策略和隐藏列 row\_id 的分配策略大抵相同，具体策略如下：**

- 服务器会在内存中维护一个全局变量，每当需要为某个事务分配一个事务 id 时，就会把该变量的值当作事务 id 分配给该事务，并且把该变量自增 1。

关于作者



京东云开  
发者  
JDCloud  
关注

文章

1.8K

经验值

5.8W

作者的专辑

大模型时代 (14)

性能优化 (30)

案例分享 (76)

开源 (3)

作者的其它热门文章

事务，不只ACID | 京东物流技术团队

理解Mysql索引原理 | 京东物流技术团队

分布式事务的华丽进阶 | 京东物流技术团队

UDData+StarRocks在京东物流技术团队

热门资讯

1 国人独立开发的开源 Redis 替代品 ioredis 被 Redis 社区封杀

2 GPL 抗辩成功——红帽与甲骨文纠纷迎来重大转折

3 替代 Nginx，CloudNative 框架 Rust 框架

4 Visual Studio Code 官方宣布支持 Windows ARM 架构

5 黄仁勋：别让你的 AI 应用成为“僵尸”

6 马斯克抱怨微软 Windows 11 加入 Linux！

7 开源中国 APP 全新升级，集成大模型对话功能

8 禾赛科技激光雷达故障导致自动驾驶故障

9 马斯克起诉 OpenAI，要求公司恢复开源模型

10 开源日报 | MariaDB 有三不沾；V神建议

- 当系统下一次重新启动时，会将上边提到的 Max Trx ID 属性加载到内存中，将该值加上 256 之后赋值给全局变量，因为在上次关机时该全局变量的值可能大于 Max Trx ID 属性值。
- 这样就可以保证整个系统中分配的事务 id 值是一个递增的数字。先被分配 id 的事务得到的是较小的事务 id，后被分配 id 的事务得到的是较大的事务 id。

mvcc

全称 Multi-Version Concurrency Control，即多版本并发控制，主要是为了提高数据库的并发性能。

同一行数据平时发生读写请求时，会上锁阻塞住。但 MVCC 用更好的方式去处理读写请求，做到在发生读写请求冲突时不用加锁。

这个读是指的快照读，而不是当前读，当前读是一种加锁操作，是悲观锁。

MVCC 原理

在事务并发执行遇到的问题如下：

- 脏读：如果一个事务读到了另一个未提交事务修改过的数据，那就意味着发生了脏读；
- 不可重复读：如果一个事务只能读到另一个已经提交的事务修改过的数据，并且其他事务每对该数据进行一次修改并提交后，该事务都能查询得到最新值，那就意味着发生了不可重复读；
- 幻读：如果一个事务先根据某些条件查询出一些记录，之后另一个事务又向表中插入了符合这些条件的记录，原先的事务再次按照该条件查询时，能把另一个事务插入的记录也读出来，那就意味着发生了幻读，幻读强调的是在一个事务按照某个相同条件多次读取记录时，后读取时读到了之前没有读到的记录，幻读只是重点强调了读取到了之前读取没有获取到的记录。

MySQL 在 REPEATABLE READ 隔离级别下，是可以很大程度避免幻读问题的发生的。

版本链

对于使用 InnoDB 存储引擎的表来说，它的聚簇索引记录中都包含两个必要的隐藏列：

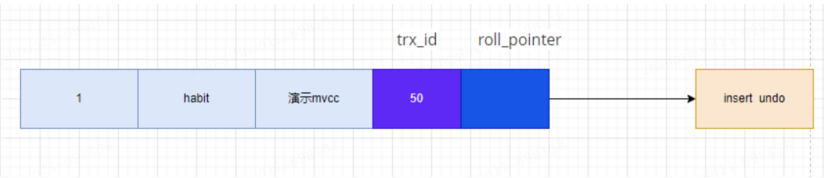
- trx\_id：每次一个事务对某条聚簇索引记录进行改动时，都会把该事务的事务 id 赋值给 trx\_id 隐藏列；
- roll\_pointer：每次对某条聚簇索引记录进行改动时，都会把旧的版本写入到 undo 日志中，然后这个隐藏列就相当于一个指针，可以通过它来找到该记录修改前的信息；

演示

```
-- 创建表
CREATE TABLE mvcc_test (
  id INT,
  name VARCHAR(100),
  domain varchar(100),
  PRIMARY KEY (id)
) Engine=InnoDB CHARSET=utf8;

-- 添加数据
INSERT INTO mvcc_test VALUES(1, 'habit', '演示mvcc');
```

假设插入该记录的事务 id=50，那么该条记录的展示如图：



文章 7 访问：



**大目**  
文章 1.2K 访问



**tobe的呓语**  
文章 16 访问



**charmingv**  
文章 164 访问



**numax**  
文章 2 访问

热门软件

Rivet AI Agent - 开源

OneOS - 轻量级物联网

llhttp - 高性能、高可

KeyDB - Redis 的高性

remote-sh - Web 界

GrimoireLab - 用于软

SwingAnimation - iO

OneFuzz - 模糊测试平

Drawio - 在线图表网

xrepo - 跨平台 C/C++

Fedora CoreOS - 针对

Linux 发行版

TipDM - 数据库建模平

x-patrol - Github 泄露

DoctorGPT - 融合了 C

Excalidraw - 虚拟白板

GoCity - 3D 可视化展

BabyOS - 为 MCU 项

架

OSSChat - 开源协作信

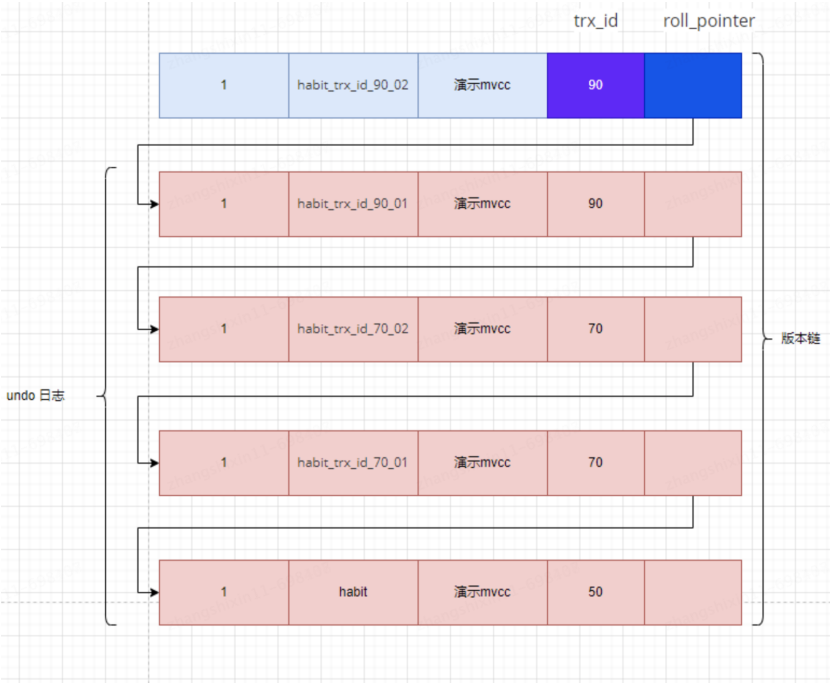
Taichi - 特效编程语言

mimalloc - 通用内存分



begin	
	begin
update mvcc_test set name='habit_trx_id_70_01' where id=1	
update mvcc_test set name='habit_trx_id_70_02' where id=1	
commit	
	update mvcc_test set name='habit_trx_id_90_01' where id=1
	update mvcc_test set name='habit_trx_id_90_02' where id=1
	commit

每次对记录进行改动，都会记录一条 undo 日志，每条 undo 日志也都有一个 roll\_pointer 属性，可以将这些 undo 日志都连起来，串成一个链表。



对该记录每次更新后，都会将旧值放到一条 undo 日志中，就算是该记录的一个旧版本，随着更新次数的增多，所有的版本都会被 roll\_pointer 属性连接成一个链表，把这个链表称之为版本链，版本链的头节点就是当前记录最新的值。另外，每个版本中还包含生成该版本时对应的事务 id。于是可以利用这个记录的版本链来控制并发事务访问相同记录的行为，那么这种机制就被称之为：**多版本并发控制，即 MVCC。**

ReadView

对于使用 READ UNCOMMITTED 隔离级别的事务来说，由于可以读到未提交事务修改过的记录，所以直接读取记录的最新版本就好了。

对于使用 SERIALIZABLE 隔离级别的事务来说，InnoDB 使用加锁的方式来访问记录。

对于使用 READ COMMITTED 和 REPEATABLE READ 隔离级别的事务来说，都必须保证读到已经提交了的事务修改过的记录，也就是说假如另一个事务已经修改了记录但是尚未提交，是不能直接读取最新版本的记录的，核心问题就是：READ COMMITTED 和 REPEATABLE READ 隔离级别在不可重复读和幻读上的区别是从哪里来的，其实结合前面的知识，这两种隔离级别关键是需要判断一下版本链中的哪个版本是当前事务可见的。

为此，InnoDB 提出了一个 ReadView 的概念，这个 ReadView 中主要包含 4 个比较重要的内容：

小值；

- max\_trx\_id：表示在生成 ReadView 时系统中应该分配给下一个事务的 id 值，注：max\_trx\_id 并不是 m\_ids 中的最大值，事务 id 是递增分配的。比方说现在有 id 为 1，2，3 这三个事务，之后 id 为 3 的事务提交了。那么一个新的读事务在生成 ReadView 时，m\_ids 就包括 1 和 2，min\_trx\_id 的值就是 1，max\_trx\_id 的值就是 4；
- creator\_trx\_id：表示生成该 ReadView 的事务的事务 id；

有了这个 ReadView，这样在访问某条记录时，只需要按照下边的步骤判断记录的某个版本是否可见：

1. 如果被访问版本的 trx\_id 属性值与 ReadView 中的 creator\_trx\_id 值相同，意味着当前事务在访问它自己修改过的记录，所以该版本可以被当前事务访问；
2. 如果被访问版本的 trx\_id 属性值小于 ReadView 中的 min\_trx\_id 值，表明生成该版本的事务在当前事务生成 ReadView 前已经提交，所以该版本可以被当前事务访问；
3. 如果被访问版本的 trx\_id 属性值大于或等于 ReadView 中的 max\_trx\_id 值，表明生成该版本的事务在当前事务生成 ReadView 后才开启，所以该版本不可以被当前事务访问；
4. 如果被访问版本的 trx\_id 属性值在 ReadView 的 min\_trx\_id 和 max\_trx\_id 之间  $min\_trx\_id < trx\_id < max\_trx\_id$ ，那就需要判断一下 trx\_id 属性值是不是在 m\_ids 列表中，如果在，说明创建 ReadView 时生成该版本的事务还是活跃的，该版本不可以被访问；如果不在，说明创建 ReadView 时生成该版本的事务已经被提交，该版本可以被访问；
5. 如果某个版本的数据对当前事务不可见的话，那就顺着版本链找到下一个版本的数据，继续按照上边的步骤判断可见性，依此类推，直到版本链中的最后一个版本。如果最后一个版本也不可见的话，那么就意味着该条记录对该事务完全不可见，查询结果就不包含该记录；

在 MySQL 中，READ COMMITTED 和 REPEATABLE READ 隔离级别的一个非常大的区别就是它们生成 ReadView 的时机不同。

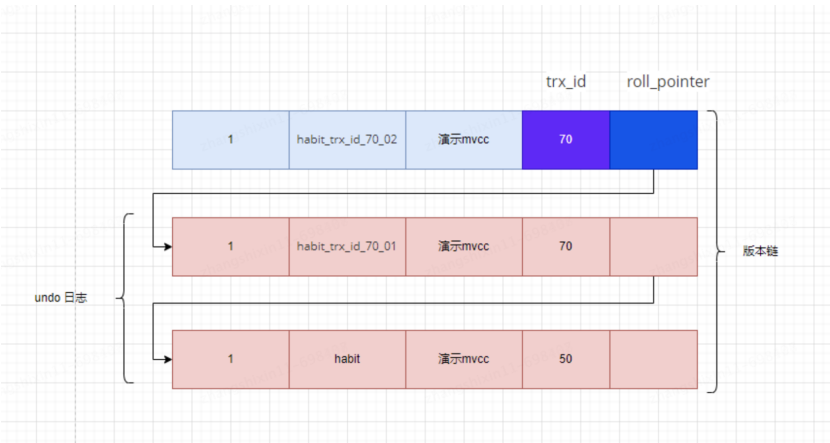
还是以表 mvcc\_test 为例，假设现在表 mvcc\_test 中只有一条由事务 id 为 50 的事务插入的一条记录，接下来看一下 READ COMMITTED 和 REPEATABLE READ 所谓的生成 ReadView 的时机不同到底不同在哪里。

**READ COMMITTED**：每次读取数据前都生成一个 ReadView；

比方说现在系统里有两个事务 id 分别为 70、90 的事务在执行：

```
-- T 70
UPDATE mvcc_test SET name = 'habit_trx_id_70_01' WHERE id = 1;
UPDATE mvcc_test SET name = 'habit_trx_id_70_02' WHERE id = 1;
```

此时表 mvcc\_test 中 id 为 1 的记录得到的版本链表如下所示：



假设现在有一个使用 READ COMMITTED 隔离级别的事务开始执行：

```
-- 得到的列 name 的值为'habit'
```

这个 SELECE1 的执行过程如下：

在执行 SELECT 语句时会先生成一个 ReadView，ReadView 的 m\_ids 列表的内容就是 [70, 90]，min\_trx\_id 为 70，max\_trx\_id 为 91，creator\_trx\_id 为 0。

然后从版本链中挑选可见的记录，从图中可以看出，最新版本的列 name 的内容是 habit\_trx\_id\_70\_02，该版本的 trx\_id 值为 70，在 m\_ids 列表内，所以不符合可见性要求第 4 条：如果被访问版本的 `trx_id` 属性值在 ReadView 的 `min_trx_id` 和 `max_trx_id` 之间  $min\_trx\_id < trx\_id < max\_trx\_id$ ，那就需要判断一下 `trx_id` 属性值是不是在 `m_ids` 列表中，如果在，说明创建 ReadView 时生成该版本的事务还是活跃的，该版本不可以被访问；如果不在，说明创建 ReadView 时生成该版本的事务已经被提交，该版本可以被访问。根据 `roll_pointer` 跳到下一个版本。

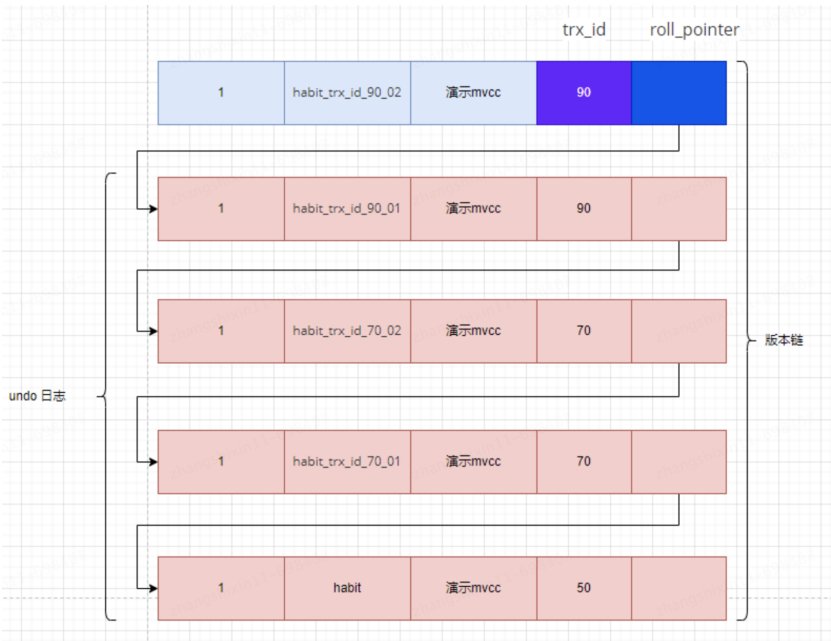
下一个版本的列 name 的内容是 habit\_trx\_id\_70\_01，该版本的 trx\_id 值也为 70，也在 m\_ids 列表内，所以也不符合要求，继续跳到下一个版本。

下一个版本的列 name 的内容是 habit，该版本的 trx\_id 值为 50，小于 ReadView 中的 min\_trx\_id 值，所以这个版本是符合要求的第 2 条：**如果被访问版本的 `trx_id` 属性值小于 ReadView 中的 `min_trx_id` 值，表明生成该版本的事务在当前事务生成 ReadView 前已经提交，所以该版本可以被当前事务访问。**最后返回的版本就是这条列 name 为 habit 的记录。

之后，把事务 id 为 70 的事务提交一下，然后再到事务 id 为 90 的事务中更新一下表 mvcc\_test 中 id 为 1 的记录：

```
-- T 90
UPDATE mvcc_test SET name = 'habit_trx_id_90_01' WHERE id = 1;
UPDATE mvcc_test SET name = 'habit_trx_id_90_02' WHERE id = 1;
```

此时表 mvcc 中 id 为 1 的记录的版本链就长这样：



然后再到刚才使用 READ COMMITTED 隔离级别的事务中继续查找这个 id 为 1 的记录，如下：

```
-- 使用 READ COMMITTED 隔离级别的事务
BEGIN;
-- SELECE1: Transaction 70、90 均未提交
SELECT * FROM mvcc_test WHERE id = 1; -- 得到的列 name 的值为'habit'
```

这个 SELECE2 的执行过程如下：

在执行 SELECT 语句时又会单独生成一个 ReadView，该 ReadView 的 m\_ids 列表的内容就是 [90]，min\_trx\_id 为 90，max\_trx\_id 为 91，creator\_trx\_id 为 0。

然后从版本链中挑选可见的记录，从图中可以看出，最新版本的列 name 的内容是 habit\_trx\_id\_90\_02，该版本的 trx\_id 值为 90，在 m\_ids 列表内，所以不符合可见性要求，根据 roll\_pointer 跳到下一个版本。

下一个版本的列 name 的内容是 habit\_trx\_id\_90\_01，该版本的 trx\_id 值为 90，也在 m\_ids 列表内，所以也不符合要求，继续跳到下一个版本。

下一个版本的列 name 的内容是 habit\_trx\_id\_70\_02，该版本的 trx\_id 值为 70，小于 ReadView 中的 min\_trx\_id 值 90，所以这个版本是符合要求的，最后返回这个版本中列 name 为 habit\_trx\_id\_70\_02 的记录。

以此类推，如果之后事务 id 为 90 的记录也提交了，再次在使用 READ COMMITTED 隔离级别的事务中查询表 mvcc\_test 中 id 值为 1 的记录时，得到的结果就是 habit\_trx\_id\_90\_02 了。

**总结：**使用 READ COMMITTED 隔离级别的事务在每次查询开始时都会生成一个独立的 ReadView。

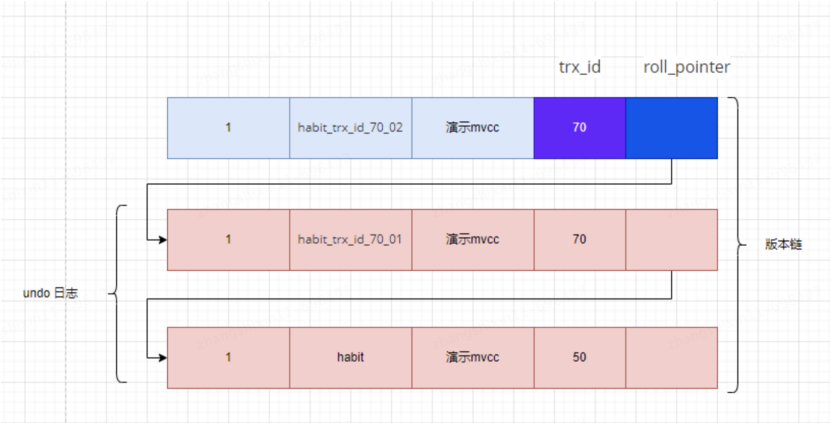
**REPEATABLE READ：**在第一次读取数据时生成一个 ReadView；

对于使用 REPEATABLE READ 隔离级别的事务来说，只会在第一次执行查询语句时生成一个 ReadView，之后的查询就不会重复生成了。

比方说现在系统里有两个事务 id 分别为 70、90 的事务在执行：

```
-- T 70
UPDATE mvcc_test SET name = 'habit_trx_id_70_01' WHERE id = 1;
UPDATE mvcc_test SET name = 'habit_trx_id_70_02' WHERE id = 1;
```

此时表 mvcc\_test 中 id 为 1 的记录得到的版本链表如下所示：



假设现在有一个使用 REPEATABLE READ 隔离级别的事务开始执行：

```
-- 使用 REPEATABLE READ 隔离级别的事务
BEGIN;
-- SELECE1: Transaction 70、90 未提交
SELECT * FROM mvcc_test WHERE id = 1; -- 得到的列name 的值为'habit'
```

这个 SELECE1 的执行过程如下：

在执行 SELECT 语句时会先生成一个 ReadView，ReadView 的 m\_ids 列表的内容就是 [70, 90]，min\_trx\_id 为 70，max\_trx\_id 为 91，creator\_trx\_id 为 0。



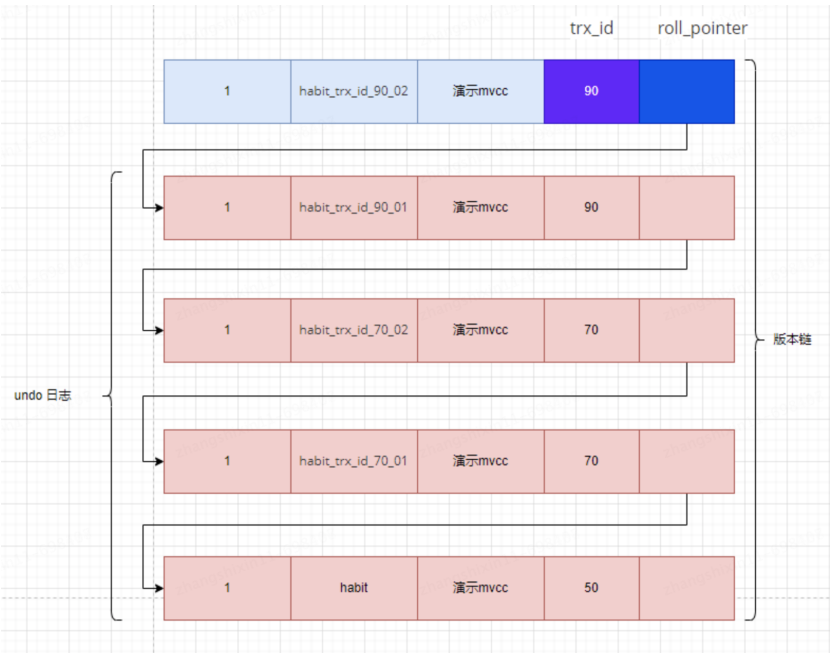
下一个版本的列 name 的内容是 habit\_trx\_id\_70\_01，该版本的 trx\_id 值也为 70，也在 m\_ids 列表内，所以也不符合要求，继续跳到下一个版本。

下一个版本的列 name 的内容是 habit，该版本的 trx\_id 值为 50，小于 ReadView 中的 min\_trx\_id 值，所以这个版本是符合要求的，最后返回的就是这条列 name 为 habit 的记录。

之后，把事务 id 为 70 的事务提交一下，然后再到事务 id 为 90 的事务中更新一下表 mvcc\_test 中 id 为 1 的记录：

```
-- 使用 REPEATABLE READ 隔离级别的事务
BEGIN;
UPDATE mvcc_test SET name = 'habit_trx_id_90_01' WHERE id = 1;
UPDATE mvcc_test SET name = 'habit_trx_id_90_02' WHERE id = 1;
```

此刻，表 mvcc\_test 中 id 为 1 的记录的版本链就长这样：



然后再到刚才使用 REPEATABLE READ 隔离级别的事务中继续查找这个 id 为 1 的记录，如下：

```
-- 使用 REPEATABLE READ 隔离级别的事务
BEGIN;
-- SELECE1: Transaction 70、90 均未提交
SELECT * FROM mvcc_test WHERE id = 1; -- 得到的列 name 的值为'habit'
-- SELECE2: Transaction 70 提交，Transaction 90 未提交
SELECT * FROM mvcc_test WHERE id = 1; -- 得到的列 name 的值为'habit'
```

这个 SELECE2 的执行过程如下：

因为当前事务的隔离级别为 REPEATABLE READ，而之前在执行 SELECE1 时已经生成过 ReadView 了，所以此时直接复用之前的 ReadView，之前的 ReadView 的 m\_ids 列表的内容就是 [70, 90]，min\_trx\_id 为 70，max\_trx\_id 为 91，creator\_trx\_id 为 0。

然后从版本链中挑选可见的记录，从图中可以看出，最新版本的列 name 的内容是 habit\_trx\_id\_90\_02，该版本的 trx\_id 值为 90，在 m\_ids 列表内，所以不符合可见性要求，根据 roll\_pointer 跳到下一个版本。

下一个版本的列 name 的内容是 habit\_trx\_id\_90\_01，该版本的 trx\_id 值为 90，也在 m\_ids 列表内，所以也不符合要求，继续跳到下一个版本。

示。继续跳到下一行版本。

下一个版本的列 name 的内容是 habit，该版本的 trx\_id 值为 50，小于 ReadView 中的 min\_trx\_id 值 70，所以这个版本是符合要求的，最后返回给用户的版本就是这条列 name 为 habit 的记录。

也就是说两次 SELECT 查询得到的结果是重复的，记录的列 name 值都是 habit，这就是可重复读的含义。如果之后再吧事务 id 为 90 的记录提交了，然后再到刚才使用 REPEATABLE READ 隔离级别的事务中继续查找这个 id 为 1 的记录，得到的结果还是 habit。

## MVCC 下的幻读解决和幻读现象

REPEATABLE READ 隔离级别下 MVCC 可以解决不可重复读问题，那么幻读呢？MVCC 是怎么解决的？幻读是一个事务按照某个相同条件多次读取记录时，后读取时读到了之前没有读到的记录，而这个记录来自另一个事务添加的新记录。

可以想想，在 REPEATABLE READ 隔离级别下的事务 T1 先根据某个搜索条件读取到多条记录，然后事务 T2 插入一条符合相应搜索条件的记录并提交，然后事务 T1 再根据相同搜索条件执行查询。结果会是什么？按照 ReadView 中的比较规则中的第 3 条和第 4 条不管事务 T2 比事务 T1 是否先开启，事务 T1 都是看不到 T2 的提交的。

但是，在 REPEATABLE READ 隔离级别下 InnoDB 中的 MVCC 可以很大程度地避免幻读现象，而不是完全禁止幻读。怎么回事呢？来看下面的情况：

mvcc_test (5r x 3c)		
id	name	domain
1	habit	演示mvcc
3	mamian	mamian
9	jesen	jesen
15	kanliang	kanliang
21	daming	daming

首先在事务 T1 中执行：select \* from mvcc\_test where id = 30；这个时候是找不到 id = 30 的记录的。

在事务 T2 中，执行插入语句：insert into mvcc\_test values(30,'luxi','luxi');

此时回到事务 T1，执行：

```
update mvcc_test set domain='luxi_t1' where id=30;
select * from mvcc_test where id = 30;
```

事务 T1 很明显出现了幻读现象。

在 REPEATABLE READ 隔离级别下，T1 第一次执行普通的 SELECT 语句时生成了一个 ReadView，之后 T2 向 mvcc\_test 表中新插入一条记录并提交。

ReadView 并不能阻止 T1 执行 UPDATE 或者 DELETE 语句来改动这个新插入的记录，由于 T2 已经提交，因此改动该记录并不会造成阻塞，但是这样一来，这条新记录的 trx\_id 隐藏列的值就变成了 T1 的事务 id。之后 T1



## mvcc 总结

从上边的描述中可以看出，所谓的 MVCC ( Multi-Version Concurrency Control ，多版本并发控制 ) 指的就是在使用 READ COMMITTD、REPEATABLE READ 这两种隔离级别的事务在执行普通的 SELECT 操作时访问记录的版本链的过程，这样子可以使不同事务的读写、读写操作并发执行，从而提升系统性能。

READ COMMITTD、REPEATABLE READ 这两个隔离级别的一个很大不同就是：生成 ReadView 的时机不同，READ COMMITTD 在每一次进行普通 SELECT 操作前都会生成一个 ReadView，而 REPEATABLE READ 只在第一次进行普通 SELECT 操作前生成一个 ReadView，之后的查询操作都重复使用这个 ReadView 就好了，从而基本上可以避免幻读现象。

## InnoDB 的 Buffer Pool

对于使用 InnoDB 作为存储引擎的表来说，不管是用于存储用户数据的索引，包括：聚簇索引和二级索引，还是各种系统数据，都是以页的形式存放在表空间中的，而所谓的表空间只不过是 InnoDB 对文件系统上一个或几个实际文件的抽象，也就是说数据还是存储在磁盘上的。

但是磁盘的速度慢，所以 InnoDB 存储引擎在处理客户端的请求时，当需要访问某个页的数据时，就会把完整的页的数据全部加载到内存中，即使只需要访问一个页的一条记录，那也需要先把整个页的数据加载到内存中。将整个页加载到内存中后就可以进行读写访问了，在进行完读写访问之后并不着急把该页对应的内存空间释放掉，而是将其缓存起来，这样将来有请求再次访问该页面时，就可以省去磁盘 IO 的开销了。

## Buffer Pool

InnoDB 为了缓存磁盘中的页，在 MySQL 服务器启动的时候就向操作系统申请了一片连续的内存，这块连续内存叫做：Buffer Pool，中文名：缓冲池。

默认情况下 Buffer Pool 只有 128M 大小。

查看该值：`show variables like 'innodb_buffer_pool_size';`

可以在启动服务器的时候配置 `innodb_buffer_pool_size` 参数的值，它表示 Buffer Pool 的大小，配置如下：

```
[server]
innodb_buffer_pool_size = 268435456
```

其中，268435456 的单位是字节，也就是指定 Buffer Pool 的大小为 256M，Buffer Pool 也不能太小，最小值为 5M，当小于该值时会自动设置成 5M。

启动 MySQL 服务器的时候，需要完成对 Buffer Pool 的初始化过程，就是先向操作系统申请 Buffer Pool 的内存空间，然后把它划分成若干对控制块和缓存页。但是此时并没有真实的磁盘页被缓存到 Buffer Pool 中，之后随着程序的运行，会不断的有磁盘上的页被缓存到 Buffer Pool 中。

在 Buffer Pool 中会创建多个缓存页，默认的缓存页大小和在磁盘上默认的页大小是一样的，都是 16KB。

### 那么怎么知道该页在不在 Buffer Pool 中呢？

在查找数据的时候，先通过哈希表中查找 key 是否在哈希表中，如果在证明 Buffer Pool 中存在该缓存信息，如果不存在证明不存该缓存信息，则通过读取磁盘加载该页信息放到 Buffer Pool 中，哈希表中的 key 是通过表空间号 + 页号组成的，value 是 Buffer Pool 的缓存页。

## flush 链表的管理

如果修改了 Buffer Pool 中某个缓存页的数据，那它就和磁盘上的页不一致了，这样的缓存页也被称为：脏页。最简单的做法就是每发生一次修改就立即同步到磁盘上对应的页上，但是频繁的往磁盘写数据会严重的影响





这样，整个数据库所有的缓存页都可以刷到磁盘上啦，如果 Buffer Pool 被设置得很小，那么一次访问少云非常慢。

所以，需要再创建一个存储脏页的链表，凡是修改过的缓存页对应的控制块都会作为一个节点加入到一个链表中，因为这个链表节点对应的缓存页都是需要被刷新到磁盘上的，所以也叫 flush 链表。

## 刷新脏页到磁盘

后台有专门的线程每隔一段时间负责把脏页刷新到磁盘，这样可以不影响用户线程处理正常的请求。

从 flush 链表中刷新一部分页面到磁盘，后台线程也会定时从 flush 链表中刷新一部分页面到磁盘，刷新的速率取决于当时系统是不是很忙。这种刷新页面的方式被称之为：BUF\_FLUSH\_LIST。

## redo 日志

### redo 日志的作用

InnoDB 存储引擎是以页为单位来管理存储空间的，增删改查操作其实本质上都是在访问页面，包括：读页面、写页面、创建新页面等操作。在真正访问页面之前，需要把在磁盘上的页缓存到内存中的 Buffer Pool 之后才可以访问。但是在事务的时候又强调过一个称之为持久性的特性，就是说对于一个已经提交的事务，在事务提交后即使系统发生了崩溃，这个事务对数据库中所做的更改也不能丢失。

如果只在内存的 Buffer Pool 中修改了页面，假设在事务提交后突然发生了某个故障，导致内存中的数据都失效了，那么这个已经提交的事务对数据库中所做的更改也就跟着丢失了，这是所不能忍受的。那么如何保证这个持久性呢？一个很简单的做法就是在事务提交完成之前把该事务所修改的所有页面都刷新到磁盘，但是这个简单粗暴的做法有些问题：

1. 刷新一个完整的数据页太浪费了；有时候仅仅修改了某个页面中的一个字节，但是在 InnoDB 中是以页为单位来进行磁盘 IO 的，也就是说在该事务提交时不得不将一个完整的页面从内存中刷新到磁盘，一个页面默认是 16KB 大小，只修改一个字节就要刷新 16KB 的数据到磁盘上显然是太浪费了。
2. 随机 IO 刷起来比较慢；一个事务可能包含很多语句，即使是一条语句也可能修改许多页面，该事务修改的这些页面可能并不相邻，这就意味着在将某个事务修改的 Buffer Pool 中的页面刷新到磁盘时，需要进行很多的随机 IO，随机 IO 比顺序 IO 要慢，尤其对于传统的机械硬盘来说。

只是想让已经提交的事务对数据库中数据所做的修改永久生效，即使后来系统崩溃，在重启后也能把这种修改恢复出来。其实没有必要在每次事务提交时就把该事务在内存中修改过的全部页面刷新到磁盘，只需要把修改了哪些东西记录一下就好，比方说：某个事务将系统表空间中的第 5 号页面中偏移量为 5000 处的那个字节的值 0 改成 5 只需要记录一下：**将第 5 号表空间的 5 号页面的偏移量为 5000 处的值更新为：5。**

这样在事务提交时，把上述内容刷新到磁盘中，即使之后系统崩溃了，重启之后只要按照上述内容所记录的步骤重新更新一下数据页，那么该事务对数据库中所做的修改又可以被恢复出来，也就意味着满足持久性的要求。因为在系统崩溃重启时需要按照上述内容所记录的步骤重新更新数据页，所以上述内容也被称之为：**重做日志，即：redo log**。与在事务提交时将所有修改过的内存中的页面刷新到磁盘中相比，只将该事务执行过程中产生的 redo log 刷新到磁盘的好处如下：

1. redo log 占用的空间非常小存储表空间 ID、页号、偏移量以及需要更新的值所需的存储空间是很小的；
2. redo log 是顺序写入磁盘的在执行事务的过程中，每执行一条语句，就可能产生若干条 redo log，这些日志是按照产生的顺序写入磁盘的，也就是使用顺序 IO；

### redo log 的写入过程

InnoDB 为了更好的进行系统崩溃恢复，把一次原子操作生成的 redo log 都放在了大小为 512 字节的块（block）中。

为了解决磁盘速度过慢的问题而引入了 Buffer Pool。同理，写入 redo log 时也不能直接写到磁盘上，实际上在服务器启动时就向操作系统申请了一大片称之为 redo log buffer 的连续内存空间，即：redo log 缓冲区，也可





向 log buffer 中写入 redo log 的过程是顺序的，也就是先往前边的 block 中写，当该 block 的空闲空间用完之后再往下一个 block 中写。

## redo log 刷盘时机

log buffer 什么时候会写入到磁盘呢？

- log buffer 空间不足时，如果不停的往这个有限大小的 log buffer 里塞入日志，很快它就会被填满。InnoDB 认为如果当前写入 log buffer 的 redo log 量已经占满了 log buffer 总容量的大约一半左右，就需要把这些日志刷新到磁盘上。
- 事务提交时，必须要把修改这些页面对应的 redo log 刷新到磁盘。
- 后台有一个线程，大约每秒都会刷新一次 log buffer 中的 redo log 到磁盘。
- 正常关闭服务器时等等。

## undo 日志

事务需要保证原子性，也就是事务中的操作要么全部完成，要么什么也不做。但是偏偏有时候事务执行到一半会出现一些情况，比如：

- 情况一：事务执行过程中可能遇到各种错误，比如服务器本身的错误，操作系统错误，甚至是突然断电导致的错误。
- 情况二：程序员可以在事务执行过程中手动输入 ROLLBACK 语句结束当前的事务的执行。

这两种情况都会导致事务执行到一半就结束，但是事务执行过程中可能已经修改了很多东西，为了保证事务的原子性，需要把东西改回原先的样子，这个过程就称之为回滚，即：rollback，这样就可以造成这个事务看起来什么都没做，所以符合原子性要求。

每当要对一条记录做改动时，都需要把回滚时所需的東西都给记录下来。

比方说：

- 插入一条记录时，至少要把这条记录的主键值记录下来，之后回滚的时候只需要把这个主键值对应的记录删掉。
- 删除了一条记录，至少要把这条记录中的内容都记录下来，这样之后回滚时再把由这些内容组成的记录插入到表中。
- 修改了一条记录，至少要把修改这条记录前的旧值都记录下来，这样之后回滚时再把这条记录更新为旧值。

这些为了回滚而记录的这些东西称之为撤销日志，即：undo log。这里需要注意的一点是，由于查询操作并不会修改任何用户记录，所以在查询操作执行时，并不需要记录相应的 undo log。

## undo 日志的格式

为了实现事务的原子性，InnoDB 存储引擎在实际进行增、删、改一条记录时，都需要先把对应的 undo 日志记录下来。一般每对一条记录做一次改动，就对应着一条 undo 日志，但在某些更新记录的操作中，也可能对应着 2 条 undo 日志。

一个事务在执行过程中可能新增、删除、更新若干条记录，也就是说需要记录很多条对应的 undo 日志，这些 undo 日志会被从 0 开始编号，也就是说根据生成的顺序分别被称为第 0 号 undo 日志、第 1 号 undo 日志、...、第 n 号 undo 日志等，这个编号也被称之为 undo no。

这些 undo 日志是被记录到类型为 FIL\_PAGE\_UNDO\_LOG 的页面中。这些页面可以从系统表空间中分配，也可以从一种专门存放 undo 日志的表空间，也就是所谓的 undo tablespace 中分配。





来源：京东云开发者社区 自圆其说 Tech 转载请注明来源

© 著作权归作者所有

举报



点击引领话题

## 热门内容

更多精彩内容

Anthropic 官宣 Claude 3 大模型系列

VersionFox 0.2.4 发布：一个工具管理所有运行时版本！

【开源】:iphone: 首个零代码快准稳 UI 录制回放 :rocket: ...

:tada: DDD 即服务 | Wow 2.16.8 发布

CXYGZL 实现钉钉、飞书和微信全面覆盖！！

功能强大的开源数据中台系统 DataCap 2024.02.1 发布

官宣！知名 IaC 工具 Crossplane 宣布 KCL 登陆其官方函数...

「外部参数」功能已上线，爷们儿速来体验！！

Apache SeaTunnel 2.3.4 版本发布：功能升级，性能提升

织梦 CMS “混沌” 往事二十年

OPPO手机为何可靠？我们携手人民日报为你解读

【反哺开源】我们计划把“这个”商业化功能贡献给Apach...

2023RT-Thread开发者大会议程更新啦！德赛西威、瑞芯微...

想要一个龙年头像，在线等挺急的

RT-Thread v5.0.2 发布

为啥不建议用BeanUtils.copyProperties拷贝数据 | 京东云...

龙蜥解锁倚天底层算力，加速数据智能化关键技术落地

Spring Boot 信息泄露总结

国内低代码平台有哪些？盘点十大低代码平台排名

log4j2同步日志引发的性能问题 | 京东物流技术团队

细说 MySQL 用户安全加固策略

【就在后天！】10月14日上海线下培训：RT-Thread × 瑞...

Anolis OS 23 基于 AMD 实例的 AI 推理优化实践 | 龙蜥技术

欢迎参与2024年 OpenTiny 开源项目用户调研

开发者力荐龙蜥社区KOS服务器操作系统：迁移平滑 运行稳...

机密计算容器前沿探索与 AI 场景应用

阿里云推出 3.x Java 探针，解锁应用观测与治理的全新姿势

强化学习的一周「GitHub 热点速览」

CVE-2023-01799提权利用

【实践篇】一次Paas化热部署实践分享 | 京东云技术团队

MySQL 8.1 和 8.2 中 EXPLAIN 的新玩法

数据库应急方案

阿里云主导《Serverless 计算安全指南》国际标准正式立项！

Taro：高性能小程序的最佳实践 | 京东云技术团队

把Mybatis Generator生成的代码加上想要的注释

RAC to RAC DG搭建学习

语言大模型的推理技巧

Conti 泄露资料 | 套路不新颖，攻击“狠” 立体！

鲜衣怒马少年时 | GreptimeDB 开源一周年回顾

进击的 Serverless：Java 应用如何从容地面对突增流量

jar包的精细化运营，Java模块化简介 | 京东云技术团队

当平台工程遇上DevEx：打造卓越的开发者体验

火山引擎DataTester升级MAB功能，助力企业营销决策

领跑 AI 时代，龙蜥操作系统大会如约而至

MYSQL学习PPT 2023版

阿里云微服务引擎 MSE 2023 年 8 月产品动态

:fire::fire: Java(solon) -VS- Go(gin) 之内存与并发测试

DBeaver 24.0.0 发布

【店滴云】接入微信第三方服务，打通微信流量，提升商业...

账号管理支持批量测试资产可连接性，JumpServer 堡垒机 ...

RWKV-6-Finch 3B 模型于 2 月 29 日开源

【比较mybatis、lazy、sqltoy、mybatis-flex、easy-quer...

2024游戏行业开发者报告：1/3开发者受裁员影响，大部分...

印度：部署人工智能及生成式 AI 模型须先获得批准

🔥 周热点 | Ubuntu开始“锈化”；GPL抗辩成功；国产ior...

easyAI v1.1.7 新版本发布，Java 原生 AI 算法开发引擎

可观测 AIOps 的智能监控和诊断实践 | QCon 全球软件开...

使用eBPF加速阿里云服务网格ASM

智慧商业：探索分布式云技术为企业创造商业价值，减少成...

12月18日北京见啦！全面进化一云多芯分论坛等你来 | 202...

UBC SDK日志级别重复率优化实践

阿里云 ACK 云上大规模 Kubernetes 集群高可靠性保障实战

掌握高性能SQL的34个秘诀🔗多维度优化与全方位指南

《Hive编程指南》读书笔记

伸向Markdown的黑手，知名博客平台曝出LFI漏洞

对于表SCOTT.DEPT的简单视图SCOTT.DEPT\_VIEW。如下i...

SAE 2.0，让容器化应用开发更简单

MySQL 迁移完不能快速导出数据了？

京东科技设计稿转代码平台——画眉介绍

关注潜在的整数越界问题 | 京东物流技术团队

上海站活动回顾 | 聚焦私募视野，助力量化投融资交易

【腾讯技术答人挑战赛】答题赢iPad、Switch与海量鹅厂公...

我是如何实现Go性能5倍提升的？

【交付高质量，用户高增长】-用户增长质量保证方法论

【专访浪潮信息】构建开放公平的社区生态，中国服务器操...

文末福利免费送 | AI 时代数据库技术专题沙龙，名额仅剩 2...

ASM字节码操作类库(打开java语言世界通往字节码世界的...

老知识复盘-SQL从提交到执行到底经历了什么 | 京东云技术...

MySQL 备份恢复最佳实践：终极指南

LLUG 2023·成都+COSCon'23 年度大戏！龙蜥邀您热辣相...

千人千面的用户体验，从HTTP响应做起

关于「日志采样」的一些思考及实践

EasyCtrl，轻松搞定角色控制器输入！

多数据源管理：掌握@DS注解的威力 | 京东云技术团队

神经网络是如何工作的？ | 京东云技术团队

平安校园智慧安防监控摄像头的布控建议

智能视频监控平台EasyCVR级联后，上级平台如何获取下级...

SpringMvc集成开源流量监控、限流、熔断降级、负载保护...

Oracle19c\_RU数据库补丁实施

专访|OpenTiny 开源社区 常浩：完成比完美更重要

政采云基于 Dubbo 的混合云数据跨网实践

Farewell to Pika, Embracing the Arrival of PikiwiDB in 2...

网络层的攻击方法/入侵检测 第五期：IP 伪装，伪装IP... 你相信 MySQL 数据库与 MySQL 的区别么？

全站热门评论

- D

**dwcz** 2024-02-18 22:12

中国法治的水平：二创可以演绎原创，三创不能演绎二创。
- 

**老盖** 2024-02-01 14:11

windows没希望了，一群阿三，越做越差
- 

**爱吃生梨** 2024-01-31 15:35

百小僧果然是刷新中国开源底线里程碑式的人物，这不追随者来了。而且从.net扩散到java
- 

**osc\_94406955** 2024-03-01 09:29

预计该问题会在 24 小时内彻底解决..... 今天3月1日了,bug神奇的消失
- 

**小xu中年** 2024-03-05 09:31

还有beetlsql
- 

**二小欧巴** 2024-03-05 00:06

最不务实的就是小米
- T

**tedx53** 2024-02-27 09:42

高考状元的试卷给我抄，我也能轻松上清华
- 

**Z生生不息** 2024-01-31 18:33

没有人要求你必须维护必须开源。但是你有权力要求用户“不得以任何形式传播及公开”吗？这是 AGPL 开源协议所允许的吗？如果你连开源协议都搞不清楚，早点把仓库归档吧。整得跟小学生耍流氓一样。😏😏😏
- 

**高排量低炭烧** 2024-02-26 21:29

鸿蒙只是人家现学的，人家本来薪水就这么高，而不是新手培训完就值这个数
- 

**魔力猫** 2024-02-01 09:35

这篇说的好听，但你心里，公有制就是白嫖，你之前公告写着“ioGame21 在线文档依旧采用自愿付费模式策略。简单的说，我们提供了最新在线文档的白嫖方式，如果你打算跟进框架最新版本的，依旧可以选择白嫖在线文档。”，“综上，想继续白嫖的，请跟进最新版本。”不是吗？这是什么理念？我看更像精神分裂的理念，伪君子 and 真小人之间来回切换，嘴上都不一致。
- 

**大风起兮9527** 2024-02-06 08:57

这事没啥好讨论的啊，代码如果开始的时候是开源，那么你有闭源的权利，但是要声明之后的版本开始，不能回溯。文档也是如此，但是有一点，如果文档虽然没有声明开源，但是自己在公开场合发布过，你可以建议请求不要随意传播，但是不能强硬的禁止。关于白嫖，这个说法过分了，对于这些想要商业化的项目，开源伊始的初心，无非是面推广费、免费测试。各有所图，不要互相指责。
- 

**xiaozhi2015** 2024-03-05 09:22

放开行政手段，社会自己会变革，有些人总是高高在上，妄想掌控一切
- 

**战场原礼亚** 2024-03-04 21:06

什么样的土壤长什么样的庄稼
- 

**yl-yue** 2024-02-18 13:49

操作也是相当炸裂，原来还可以这样玩，字太多了没读透，我的理解，就算是鸡肋专利也够他们吹嘘了，要是限制性专利，那是相当牛逼，开源行业要炸。
- 

**zhangjinsongok** 2024-02-01 16:29

开源世界的孤胆英雄。
- 

**sprouting** 2024-03-05 09:55

除了模型问题，显卡应该也是一个问题。哎，没想到，计算机发展成为了很多普通程序员看不懂的地步了
- dantezhu** 2024-02-28 11:20



OSCHINA

Gitee

DevOps 资讯 专区 问答 活动 软件库 Tool 博客 培训 众包

大家都在搜...

记得小猴初见123 2024-02-29 16:55

百小僧，出列

K

kylexy

2024-02-26 10:38

大实话。。。

monkey\_cici 2024-03-05 08:54

先免费抢占用户，再要钱....

小xu中年

2024-03-05 09:30


咖啡

梅子酒好吃 2024-03-04 19:07

恶意诽谤别人不好的：)

小肥侠 2024-02-19 17:51

所以将开源软件打包到应用商店，是真的有市场。

魔力猫 2024-02-01 09:48

开源不是不可以收费，基本上也没人认为开源项目里有收费项目是什么十恶不赦的罪过。问题在于，你要合理合法，要符合开源的道德。最近的事件，要么是某和尚绑票，不合理不合法更没道德，绑票拿赎金。要么就是这位，明明合法的事情，偏偏发个歧视公告，张嘴白嫖闭嘴白嫖，过嘴瘾有意思吗？开源生意，哪怕你心理一万个不愿意，但是既然你做了这个生意，伪君子人设好歹你不干的时候再扔呀，一会儿伪君子一会儿真小人，觉得不找骂才怪。

无法选中 2024-03-05 10:44

这没用native image吧

不断的重复 2024-03-04 22:43

继续支持

C

cassan

2024-03-01 22:19

开源了，我们国内的公司又可以申请知识产权了

雪

雪舞庐州

2024-03-05 10:13

界面太粗糙了，整体像个Demo软件

monkey\_cici 2024-02-26 11:39

开源系统还是要看民企的深度统信和华为欧拉...

gmg 2024-02-26 22:23

有点好奇为什么发布这种表面看起来吸引眼球的标题。

J

Johnnyxc

2024-03-05 09:42

我朝预备DY牛批了。

加百列Gabriel 2024-03-04 21:53

原来说的是印度啊，我还以为说的是印度呢

ClouGence

2024-03-05 10:03

是 Win 7 吗？Win7 暂时没有适配，如果是Win 7 以上的系统出现这个问题可以加下我们的微信群，里面有技术人员跟您沟通解决的

泰利明 2024-03-05 09:50

柜帐

2

2cong

2024-02-26 11:21

如果让我抄，我就会！

T

transtone

2024-03-04 21:03

国内参与开源的开发者那么多，为什么一直是国外公司项目摘桃子呢，这帮闲人的媚外心态功不可没。

复二呆 2024-03-05 08:52

图色彩饱和度再低点更好





- **买房也用券** 2024-03-05 08:33  
不能乱叫"军儿",会被起诉
- **msscn** 2024-03-04 20:52  
所以。spring yyds
- **开源减流** 2024-03-05 09:08  
有具体资金的支持力度么？
- o

**osc\_91229770** 2024-02-18 12:03  
这也申请专利，这个不是正常crud，常规操作吗
- **西迷岛主** 2024-01-31 13:33  
技术员的耻辱
- **Yoona520** 2024-02-24 17:44  
国外那个P站的技术水准可不低，毕竟服务全世界除CN之外的人
- **ClouGence**  2024-03-05 10:10  
感谢建议，目前这个阶段还是主要在丰富产品功能上，界面这一块会在下一个阶段进行改善，相信不会让你失望。
- **lanceadd** 2024-03-04 18:15  
可以用gitee直接同步这个github仓库的
- Q

**roomsss** 2024-03-05 09:48  
小心我们僧哥自创个redis 中国. 然后自封为redis mvp. mpp,
- **天朝八阿哥** 2024-02-29 10:32  
虽然不懂，但表示很赞，比随便就冠以“国产”“自主研发”之类的让人舒心太多了
- **闲大赋** 2024-02-18 20:49  
我对此专利的解读『 <https://my.oschina.net/xiandafu/blog/11043929> 』
- **simba\_sailor** 2024-03-04 23:07  
open ai 不open了，确实有违成立初衷。
- **桃源人** 2024-03-05 08:31  
这图黄的有点晃眼
- **买房也用券** 2024-03-05 08:35  
版本帝又开始更新了.哈哈
- **梅子酒好吃** 2024-03-05 10:45  
下次我用native image再测测看
- U

**unameuname** 2024-02-19 14:02  
第一次钓鱼，别TM给我鱼竿，我只要鱼。
- **买房也用券** 2024-01-31 14:25  
这几天OSC还真热闹,看了半天,对于个人和小厂开源总结出一个规律: 用的人少了就用"开源,永久免费"诸如此类的卖吆喝(当然,真正能做到的人也是有的,这个不能一杆子都打死); 用的人多了就开始各种套路割韭菜,割不到了就说作者要生存,你们都是白嫖党,不尊重作者的劳动成果; 反观大厂的开源作品倒是活的好好的:Druid,fastjson,dubbo,Doris 总之是国内个人和小厂开源的作品少用,慎用或不用,真心没必要给自己找麻烦;
- **魔力猫** 2024-02-01 14:19  
最近这些事件的主角，初心绝对不是什么为了创造啥，为的只是赚钱。开源只是赚钱手段，而且是觉得自己为了钱放弃很多的那种心态，心理先把自己当成了牺牲者，然后认为所有人都应该补偿自己。不拿钱补偿就是忘恩负义的白眼狼！
- **哈库纳** 2024-03-05 09:47  
还有 dbVisitor 功能齐全。
- **8ug\_icu** 2024-03-05 09:08





OSCHINA

Gitee

DevOps

资讯

专区

问答

活动

软件库

Tool

博客

培训

众包

大家都在搜...

关键是什么业务？10个人，2023年，一年，赚2000多万，泼天的富贵啊



ViperWhip

2024-03-05 08:59

有点乱，没太看懂，照理说一旦软件开源，很多人都贡献了代码之后，还存在所谓的著作权和所有权一说吗？

OSCHINA 社区

- 关于我们
- 联系我们
- 加入我们
- 合作伙伴
- Open API

攻略

- 项目运营
- Awesome 软件（持续更新中）

在线工具

- Gitee.com
- 企业研发管理
- CopyCat-代码克隆检测
- 实用在线工具
- 国家反诈中心APP下载

QQ群



229767317

公众号



视频号

