什么？事务提交后，数据丢了？

原创 尼恩架构团队 技术自由圈 2025年05月24日 11:46 湖北

尼恩说在前面：

在40岁老架构师 尼恩的**读者交流群**(50+)中，最近有小伙伴拿到了一线互联网企业如得物、阿里、滴滴、极兔、有赞、shein 希音、shopee、百度、网易的面试资格，遇到很多很重要的面试题：

> - MySQL崩溃，重启后发现有些已经提交的事务对数据的修改丢失了？ 你分析一下 是什么原因。
> - 什么情况导致了“事务已经提交，数据却丢失”呢？

前几天 小伙伴面试 美团，遇到了这个问题。但是由于 没有回答好，导致面试挂了。

小伙伴面试完了之后，来求助尼恩。那么，遇到 这个问题，该如何才能回答得很漂亮，才能 让面试官刮目相看、口水直流。

所以，尼恩给大家做一下系统化、体系化的梳理，使得大家内力猛增，可以充分展示一下大家雄厚的 “技术肌肉”，**让面试官爱到 “不能自已、口水直流”**，然后实现”offer直提”。

当然，这道面试题，以及参考答案，也会收入咱们的 《尼恩Java面试宝典》V145版本PDF集群，供后面的小伙伴参考，提升大家的 3高 架构、设计、开发水平。

> 最新《尼恩 架构笔记》《尼恩高并发三部曲》《尼恩Java面试宝典》的PDF，请关注本公众号【技术自由圈】获取，后台回复：领电子书

原始的 面试问题

下面是小伙伴面试遇到的问题：

MySQL崩溃，重启后发现有些已经提交的事务对数据的修改丢失了，不是说事务能保证ACID一致性么，什么情况导致了”事务已经提交，数据却丢失”呢？

事务提交，数据为什么会丢失？

大家知道， 在 mysql 三大日志中，通过 redo log 保证ACID了 持久性。

MySQL三大日志 详细内容，参考 尼恩团队的文章：美团面试：binlog、redolog、undo log底层原理是啥？分别实现ACID哪个特性？（尼恩图解，史上最全）

mysql崩溃恢复时，使用的是redo log 恢复 。

但是redo log日志有一个内存优化机制，根据参数，可能每隔1S从 log buffer 写入 os cache，然后刷盘，

这时如果崩溃，redo log日志在缓存中，有可能丢失1S的数据。

为什么要有redo log ？

事务提交后，必须将事务对数据页的修改刷(fsync)到磁盘上，才能保证事务的ACID特性。

这个刷盘，是一个随机写，随机写性能较低，如果每次事务提交都刷盘，会极大影响数据库的性能。

随机写性能差，有什么优化方法呢？

架构设计中有两个常见的优化方法：

- 先写日志(Write-Ahead Logging 预写日志)，将随机写优化为顺序写；
- 将每次写优化为批量写；

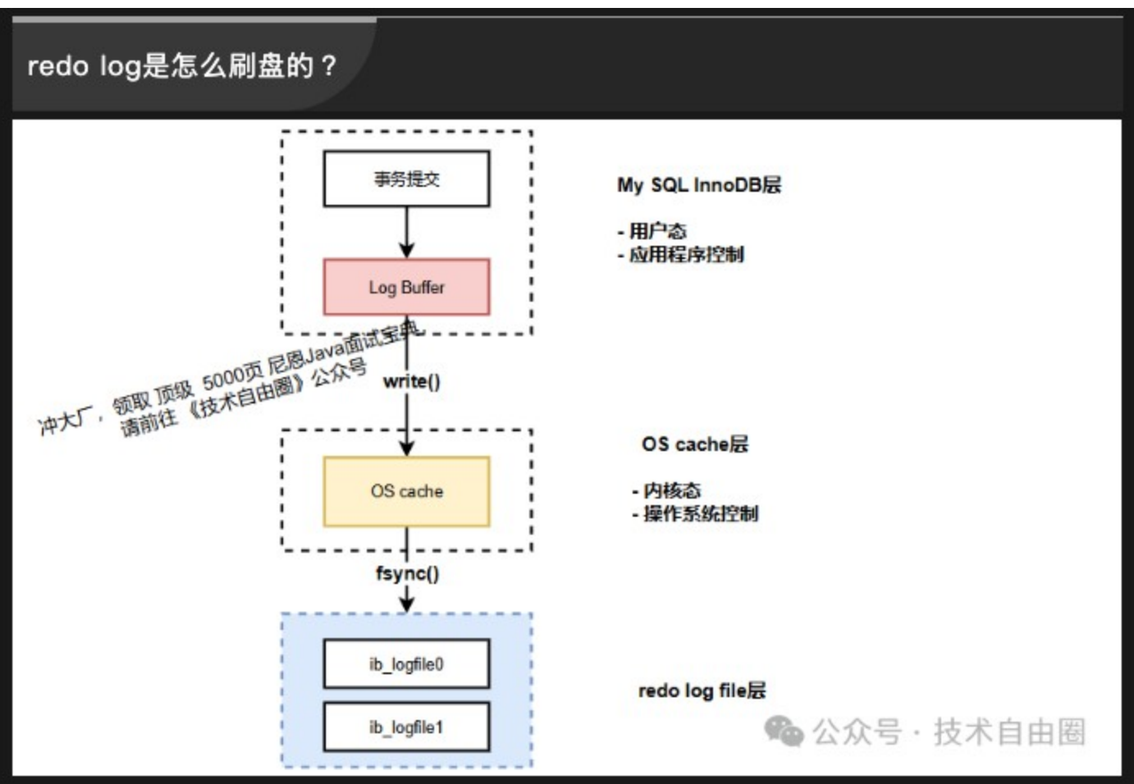
这两个优化，mysql都用上了。

先说第一个优化，将对数据的修改先顺序写到日志里，这个日志就是redo log。

假如某一时刻，数据库崩溃，还没来得及将数据页刷盘，数据库重启时，会重做redo log里的内容，以保证已提交事务对数据的影响被刷到磁盘上。

一句话，redo log是为了保证已提交事务的ACID的一致性，同时能够提高数据库性能的技术。

既然redo log能保证事务的ACID的一致性，那为什么还会出现，“事务提交了，数据库崩溃，丢数据”的问题呢？继续看redo log刷盘实现细节。



通过上面的示意图，简单说明下redo log的三层架构：

(1) 粉色，是InnoDB的一项很重要的内存结构(In-Memory Structure)，日志缓冲区(Log Buffer)，这一层，是MySQL应用程序用户态；

(2) 黄色，是操作系统的缓冲区(OS cache)，这一层，是OS内核态；

(3) 蓝色，是落盘的日志文件；

redo log最终落盘的步骤如何？

第一步：事务提交的时候，会写入Log Buffer，这里调用的是MySQL自己的函数WriteRedoLog；

第二步：只有当MySQL发起系统调用写文件write时，Log Buffer里的数据，才会写到OS cache。

> 注意，MySQL系统调用完write之后，就认为文件已经写完，如果不flush，什么时候落盘，是操作系统决定的；
>
> 比如：有时候打日志，明明printf了，tail -f却看不到，就是这个原因，操作系统还没有刷盘。

第三步：由操作系统（当然，MySQL也可以主动flush）将OS cache里的数据，最终fsync到磁盘上；

思考下面问题？

（1）操作系统为什么要缓冲数据到OS cache里，而不直接刷盘呢？

这里就是将"每次写"优化为"批量写"，以提高操作系统性能。

（2）数据库为什么要缓冲数据到Log Buffer里，而不是直接write呢？

这也是"每次写"优化为"批量写"思路的体现，以提高数据库性能。

> 这个优化思路，非常常见，高并发的MQ落盘，高并发的业务数据落盘，都可以使用。

（3）redo log的三层架构，MySQL做了一次批量写优化，OS做了一次批量写优化，确实能极大提升性能，但有什么副作用吗？

这个副作用，就是可能丢失数据：

- 事务提交时，将redo log写入Log Buffer，就会认为事务提交成功；
- 如果写入Log Buffer的数据，write入OS cache之前，数据库崩溃，就会出现数据丢失；
- 如果写入OS cache的数据，fsync入磁盘之前，操作系统崩溃，也可能出现数据丢失；

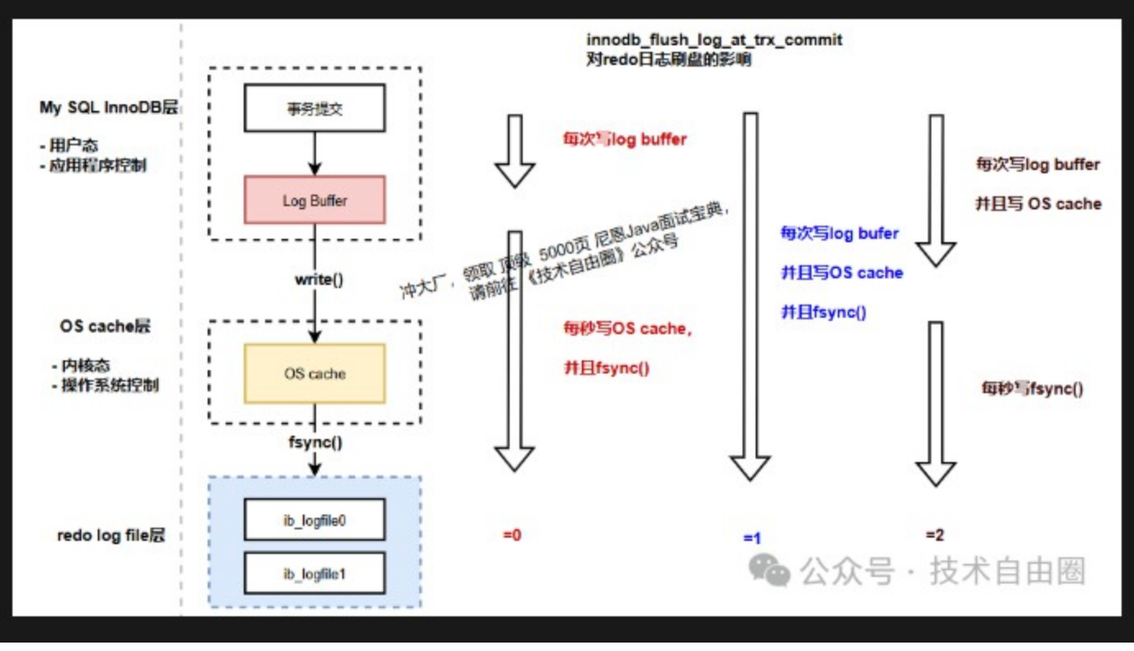
> 如上文所说，应用程序系统调用完write之后（不可能每次write后都立刻flush，这样写日志很蠢），就认为写成功了，操作系统何时fsync，应用程序并不知道，如果操作系统崩溃，数据可能丢失。

所以任何脱离业务的技术方案都是耍流氓：

（1）有些业务允许低效，就可以实现数据一点不丢失；

（2）有些业务必须高性能高吞吐，就需要容忍少量数据丢失；

MySQL有一个参数：**innodb_flush_log_at_trx_commit** 能够控制事务提交时，刷redo log的策略。目前有三种策略：



策略一：最佳性能(innodb_flush_log_at_trx_commit=0)

- 每隔一秒，才将Log Buffer中的数据批量write入OS cache，同时MySQL主动fsync。
- 这种策略，如果数据库崩溃，有一秒的数据丢失。

策略二：强一致(innodb_flush_log_at_trx_commit=1)

- 每次事务提交，都将Log Buffer中的数据write入OS cache，同时MySQL主动fsync。
- 这种策略，是InnoDB的默认配置，为的是保证事务ACID特性。

策略三：折衷(innodb_flush_log_at_trx_commit=2)

- 每次事务提交，都将Log Buffer中的数据write入OS cache；操作系统决定刷盘时机（默认每秒一次 fsync）
- MySQL 后台线程每秒也会主动触发一次 fsync，确保日志落盘
- 这种策略，如果操作系统崩溃，可能有一秒的数据丢失

策略三，如果操作系统崩溃，最多有一秒的数据丢失。因为OS也会fsync，MySQL主动fsync的周期是一秒，所以最多丢一秒数据。

策略三，磁盘IO次数不确定，因为操作系统的fsync频率并不是MySQL能控制的。

面试题回答

回答面试题：**"事务提交了，数据库崩溃，重启后丢失了数据"**，

有很大的可能，是将 innodb_flush_log_at_trx_commit 参数设置为 0 或者 2 了

- 在值=0，数据库崩溃 时， 有可能丢失1S数据
- 在值=2，操作系统崩溃 时，有可能丢失1S数据

高并发的业务，InnoDB运用哪种刷盘策略最合适？

高并发业务，行业最佳实践，是使用第三种折衷配置（=2），这是因为：

- 配置为2和配置为0，性能差异并不大，因为将数据从Log Buffer拷贝到OS cache，虽然跨越用户态与内核态，但毕竟只是内存的数据拷贝，速度很快；
- 配置为2和配置为0，安全性差异巨大，操作系统崩溃的概率相比MySQL应用程序崩溃的概率，小很多，设置为2，只要操作系统不崩溃，也绝对不会丢数据。

此问题涉及到的底层原理

这个问题涉及到的底层原理， 非常复杂，跟redo log 的刷盘机制有关，搞清楚这道面试题，得先理解MySQL的几个基础原理

- 修改语句的执行流程
- 事务的两阶段提交
- 三大日志机制，undo log，redo log和bin log

下面先看一下MySQL这三个基础原理

SQL语句的执行流程

分为两个 维度介绍：

- SQL 查询语句执行流程
- SQL 更新 语句的执行流程

SQL 查询语句执行流程

查询语句执行流程，跟这道面试题无关，可以通过下面2篇文章了解下

查询语句执行流程参考下面两篇文章：

- 网易面试：说说MySQL一条SQL语句的执行过程？
- 京东面试：一条sql 执行过程是什么？分析 SQL的解析和优化的原理？

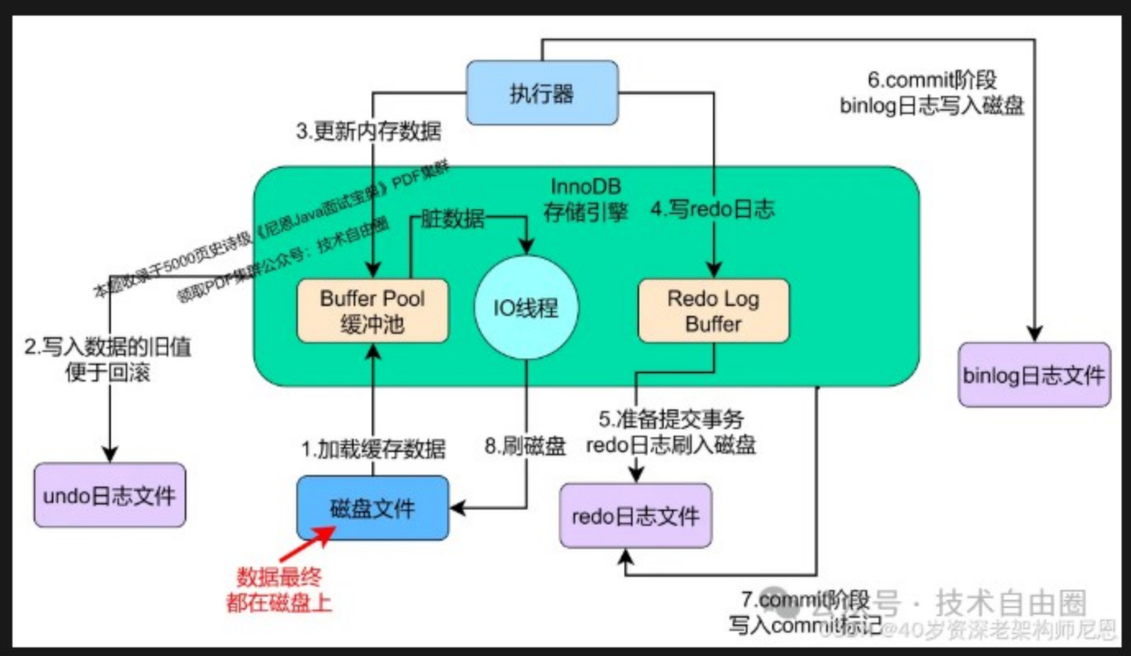
SQL 更新 语句的执行流程

事务提交，数据确丢失问题，关键在于修改语句执行过程中的日志落盘。

所以， 需要重点分析下修改语句的执行流程， 修改语句的执行流程详细分析，参考

- 美团面试：binlog、redolog、undo log底层原理是啥？分别实现ACID哪个特性？（尼恩图解，史上最全）
- 京东面试：一条sql 执行过程是什么？分析 SQL的解析和优化的原理？

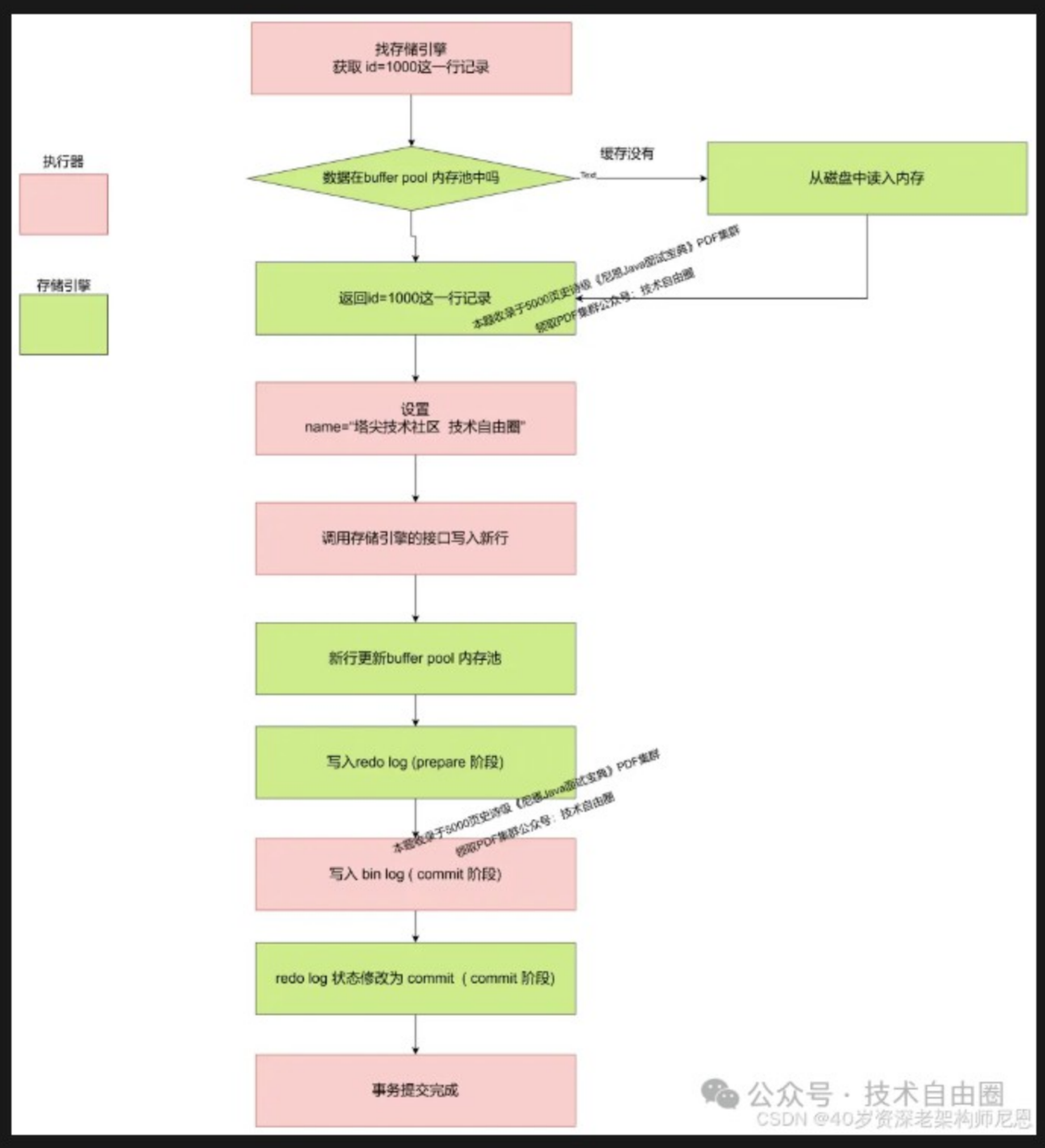
通过一张图简单说下修改语句的执行流程



以下面一条简单的 SQL 语句为例，我们来解释下执行器和 InnoDB 存储引擎在更新时做了哪些事情：

```
update table set name="塔尖技术社区  技术自由圈"  where id = 1000;
```

如下图所示：



(1) 执行器：找存储引擎取到 id = 1000 这一行记录

(2) 存储引擎：根据主键索引树找到这一行，如果 id = 1000 这一行所在的数据页本来就在内存池（Buffer Pool）中，就直接返回给执行器；否则，需要先从磁盘读入内存池，然后再返回

(3) 执行器：拿到存储引擎返回的行记录，把 name 字段设置为 "塔尖技术社区 技术自由圈"，得到一行新的记录，然后再调用存储引擎的接口写入这行新记录

(4) 存储引擎：将这行新数据更新到内存中，同时将这个更新操作记录到 redo log 里面，为 redo log 中的事务打上 prepare 标识。然后告知执行器执行完成了，随时可以提交事务

(5) 执行器：生成这个操作的 bin log，并把 bin log 写入磁盘

(6) 执行器：调用存储引擎的提交事务接口(7) 存储引擎：把刚刚写入的 redo log 状态改成提交（commit）状态，更新完成

> 注意不要把这里的提交事务和我们 sql 语句中的提交事务 commit 命令搞混了哈，我们这里说的提交事务，指的是事务提交过程中的一个小步骤，也是最后一步。当这个步骤执行完成后，commit 命令就执行成功了。

实际的场景中，上图片中的写入 redo log 和写入 bin log，并不等同于写入磁盘文件，可能仅仅写入内存（Buffer Pool）了，这也是这个面试题的关键，如果没有内存中的日志数据，没有有效落盘，数据库崩溃，就可能丢失数据。

修改语句的执行流程，设计到事务的2阶段提交，那为什么事务要2阶段提交，2阶段提交怎么完成的？

事务的 2阶段提交

两阶段提交原理很简单，将redo log的写入拆成了两个步骤prepare和commit。

这，就是大名鼎鼎的 两阶段提交。

为什么要2阶段提交？

两阶段提交，是为了保证redo log 和bin log数据的一致性

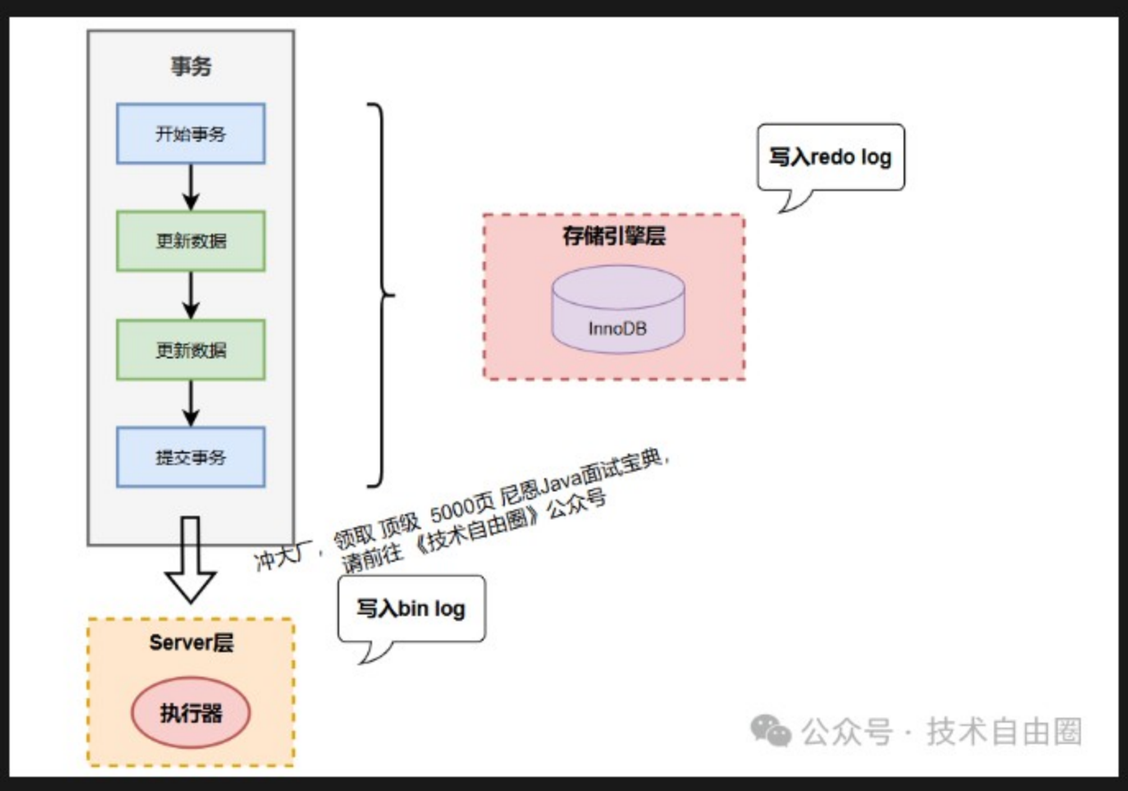
redo log和bin log的基本功能：

- redo log（重做日志）让 InnoDB存储引擎 拥有了崩溃恢复能力。
- bin log（归档日志）保证了MySQL集群架构（主主，主从复制）的数据一致性。

虽然它们都属于持久化的保证，但是则重点不同。

在执行更新语句过程，会记录redo log 与 binlog两块日志，以基本的事务为单位

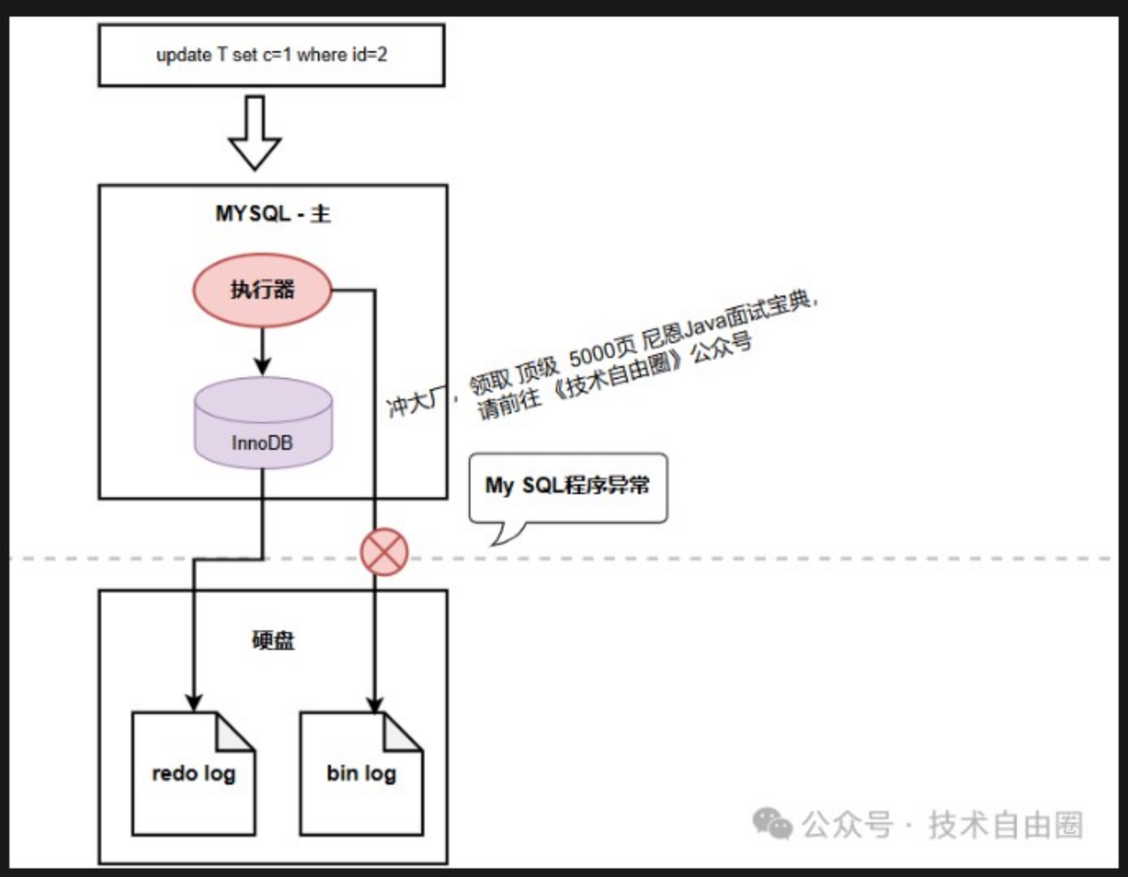
redo log 在事务执行过程中可以不断写入，而binlog只有在提交事务时才写入，所以redo log与binlog的写入时机不一样。



回到正题，redo log与binlog两份日志之间的逻辑不一致，会出现什么问题？

我们以update语句为例，SQL语句为update T set c=1 where id=2。

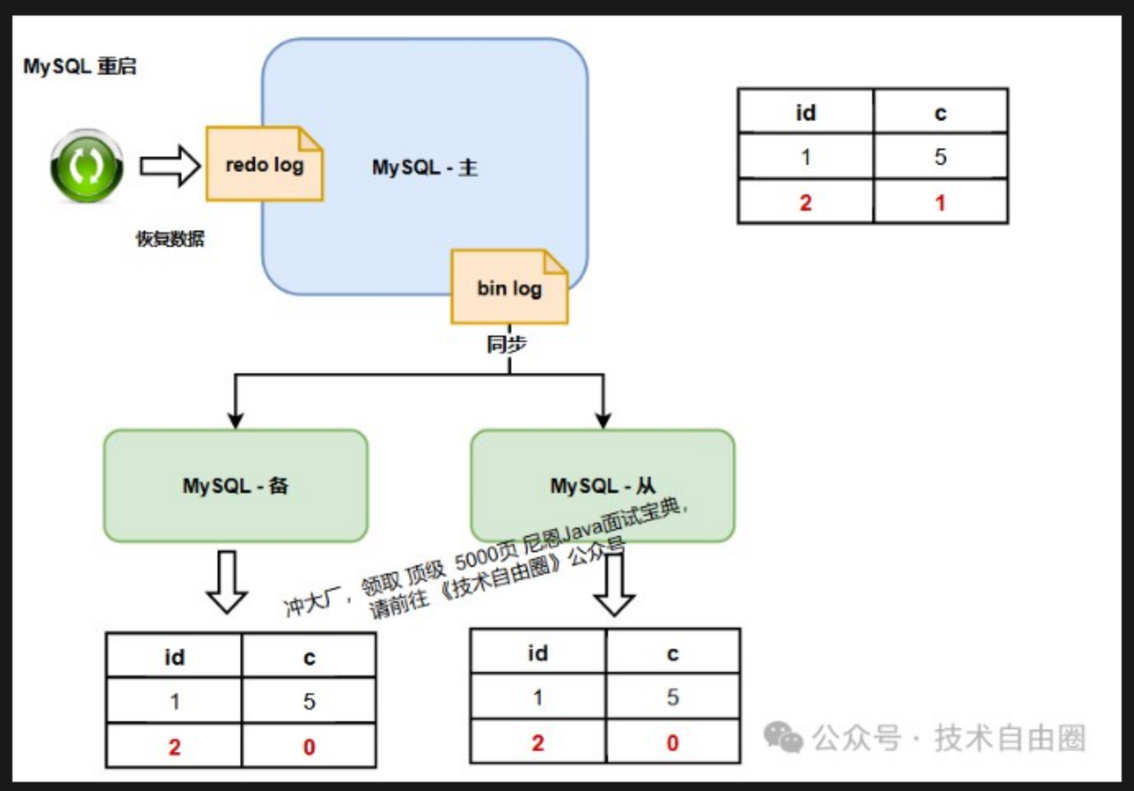
假设执行过程中写完redo log日志后，binlog日志写期间发生了异常，会出现什么情况呢？



由于binlog没写完就异常，这时候binlog里面没有对应的修改记录。

因此，之后用binlog日志恢复数据时，就会少这一次更新，恢复出来的这一行 c 值是 0。

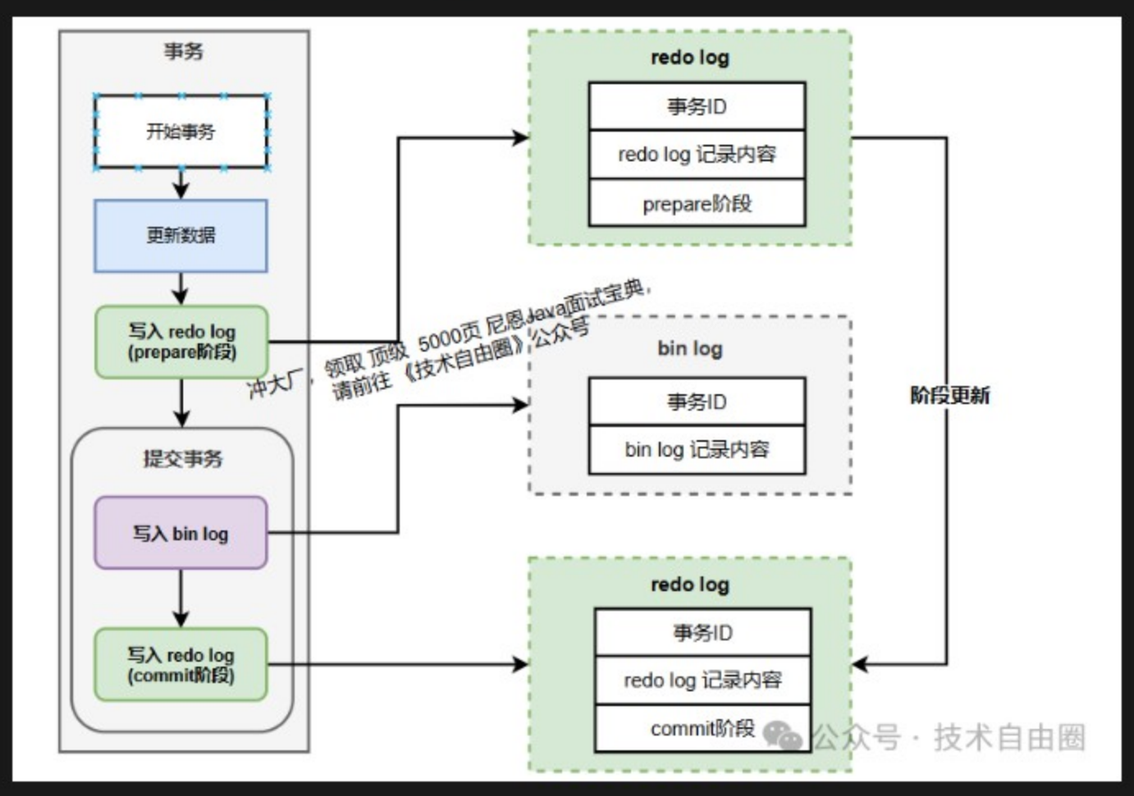
而原库因为redo log日志恢复，这一行 c 值是1，最终数据不一致。如下图



什么是两阶段提交？

为了解决两份日志之间的一致性问题，InnoDB存储引擎使用两阶段提交方案。

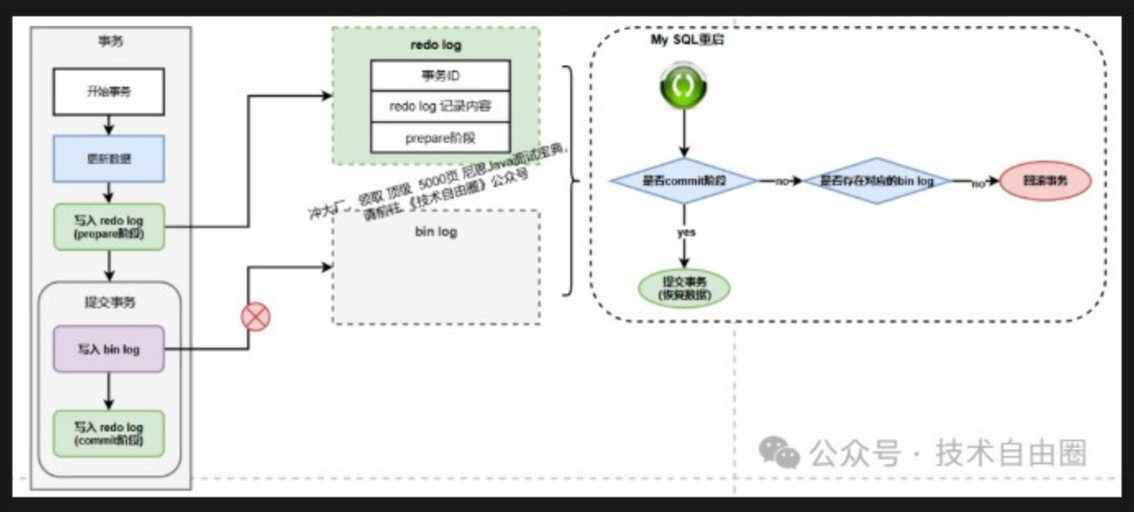
原理很简单，将redo log的写入拆成了两个步骤prepare和commit，这就是两阶段提交。



如下图，使用两阶段提交后，在写入binlog时发生异常，

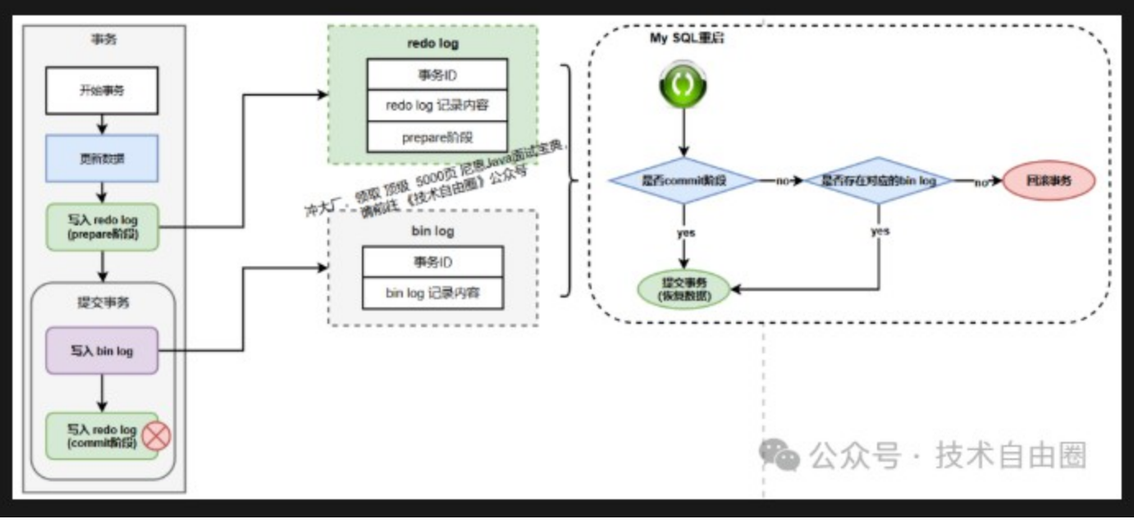
崩溃恢复时，因为MySQL根据redo log日志恢复数据，

发现redo log还处于prepare阶段，并且没有对应binlog日志，就会回滚该事务。



再看一个场景，下图，bin log已经写入，在redo log 设置 commit 阶段发生异常，那会不会回滚事务呢？

这时并不会回滚事务，它会执行上图框住的逻辑，虽然redo log是处于prepare阶段，但是能通过事务id找到对应的binlog日志，所以MySQL认为是完整的，就会提交事务恢复数据。



两阶段提交总结

可以看到，所谓两阶段提交，其实就是把 redo log 的写入拆分成了两个步骤：prepare 和 commit。

所以，为什么要这样设计呢？这样设计怎么就能够实现崩溃恢复呢？

根据两阶段提交，崩溃恢复时的判断规则是这样的：

第一情况：如果 redo log 里面的事务是完整的，也就是已经有了 commit 标识，则直接提交

第二情况：如果 redo log 里面的事务处于 prepare 状态，则判断对应的事务 binlog 是否存在并完整- 如果 binlog 存在并完整，则提交事务；- 否则，回滚事务。

结合上面的两阶段提交， 实现了事务的持久性 和一致性。

回到面试题：**事务已提交，崩溃，重启却丢失数据**，关键跟那个日志有关呢

- redo log（重做日志）让InnoDB存储引擎拥有了崩溃恢复能力。
- bin log（归档日志）保证了MySQL集群架构（主主，主从复制）的数据一致性， 也 保障了 redo log 里面的事务处于 prepare 状态 的 持久性。

崩溃恢复，关键是跟redo log有关，MySQL三大日志都非常重要，下面简单总结回顾下

简单回顾：MySQL三大日志

MySQL三大日志 详细内容，参考尼恩团队下面的文章

📄 美团面试：binlog、redolog、undo log底层原理是啥？分别实现ACID哪个特性？（尼恩图解，史上最全）

1、undo log日志

下面对undo log日志，总结和回顾下

undo log 回滚日志，保证ACID的原子性和隔离性，事务回滚rollback功能就是通过 undolog实现的，undolog主要功能如下：

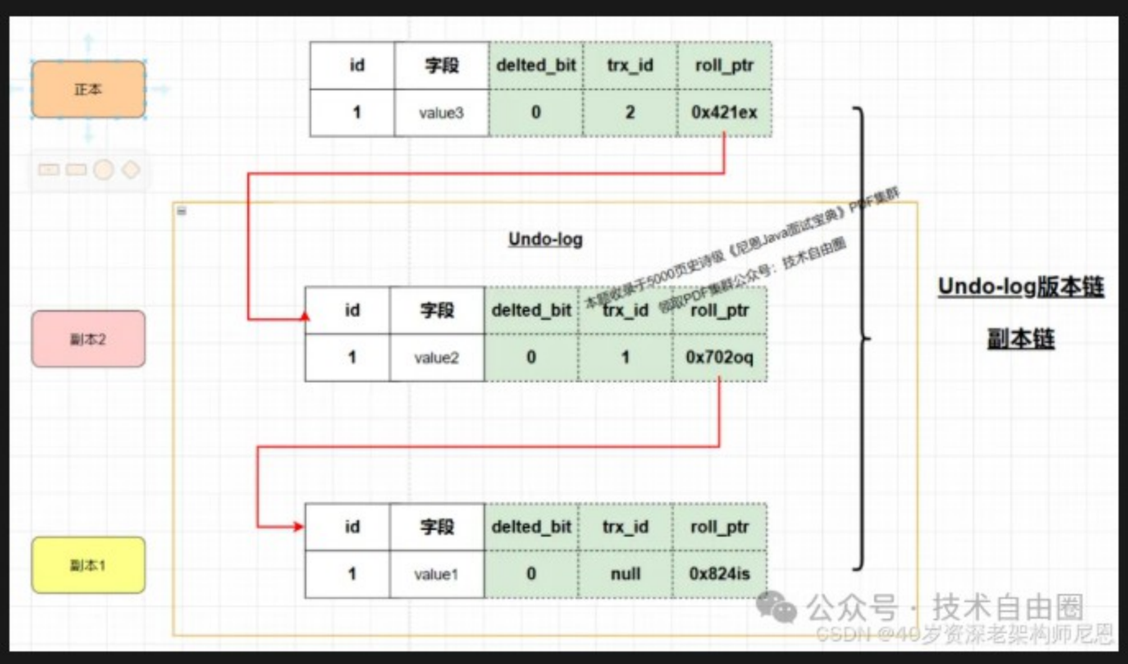
- 事务回滚
- MVCC 支持

事务回滚（原子性）

在undo log日志中记录事务中的反向操作

- 事务进行insert操作，undo log记录delete操作
- 事务进行delete操作，undo log记录insert操作
- 事务进行update操作（value1 改为value2 ），undolog记录update操作（value2 改为value3 ）

开启事务后，对表中某条记录进行修改（将该记录字段值由value1 ——> value2 ——> value3 ），如果从整个修改过程中出现异常，事务就会回滚，字段的值就回到最初的起点（值为value1 ）



- trx_id代表事务id，记录了这一系列事务操作是基于哪个事务；
- roll_pointer代表回滚指针，就是当要发生rollback回滚操作时，就通过roll_pointer进行回滚，这个链表称为版本链。构建多版本链，支持精确回滚到特定版本

undo log MVCC支持（隔离性）

Undo Log 为 MVCC 提供多版本数据快照，实现非阻塞读与隔离性。

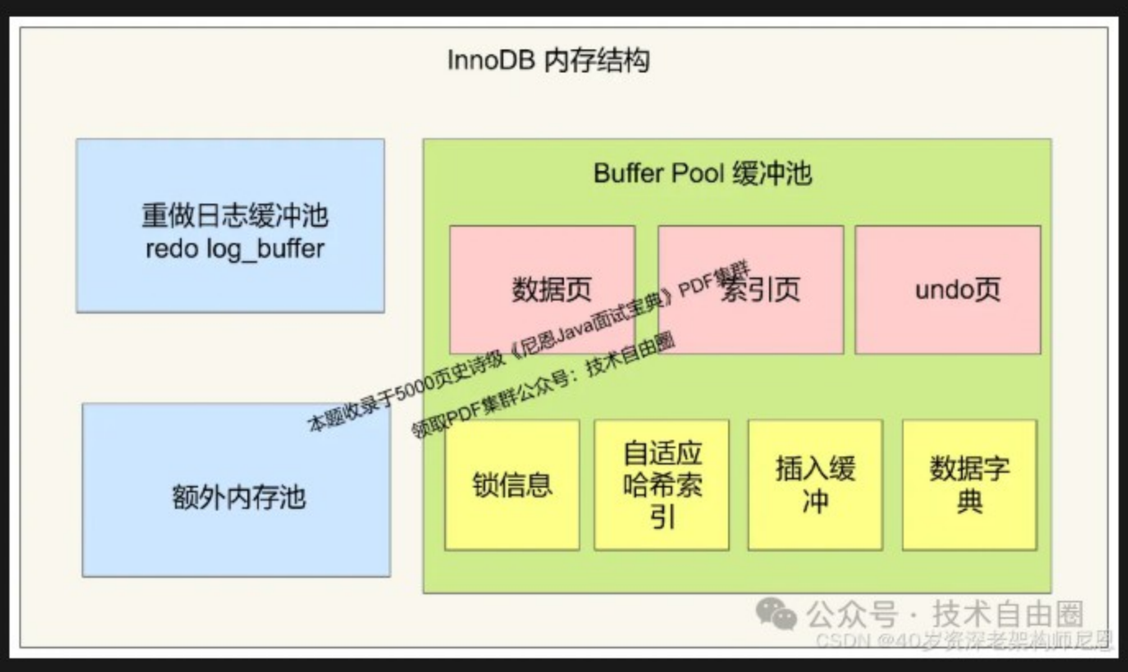
- 版本链复用：每个事务通过 trx_id 和 roll_pointer 访问对应版本数据，避免读写冲突
- ReadView 机制：结合隐藏字段（如 DB_TRX_ID）和 Undo Log 版本链，决定事务可见的数据版本
- 隔离级别适配：支持可重复读（RR）和读已提交（RC）等隔离级别，减少锁竞争

MVCC和事务的隔离性，请参见尼恩团队另外一篇重要文章：

内存优化机制

Undo Log 通过内存缓存和异步清理优化性能

- Undo Pool 缓存：Undo 页缓存在内存中，加速回滚和 MVCC 访问
- Redo Log 保护：Undo 页的修改会记录到 Redo Log，确保崩溃后仍可恢复
- 异步清理：事务提交后，Purge 线程回收不再需要的 Undo 页，减少内存占用



2、redo log 日志

redo log保证了ACID的 持久性，是这道面试题分析的重点？

Redo Log是InnoDB存储引擎特有的物理日志，记录数据页的物理修改（如表空间号、页号、偏移量、修改值），而非逻辑SQL语句（bin log是逻辑日志，存储的是逻辑sql）

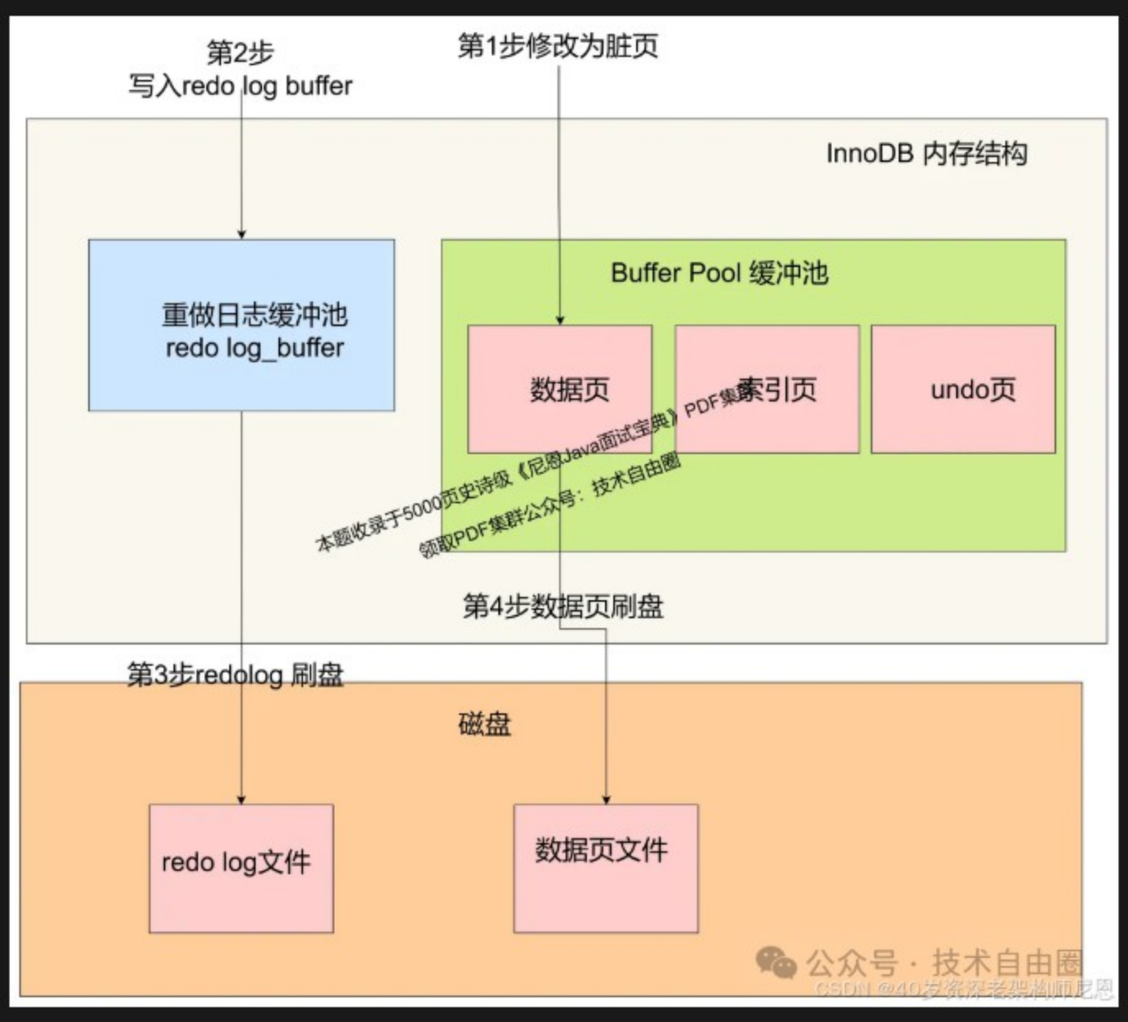
Redo Log是MySQL实现Crash-Safe的核心机制。当数据库宕机重启时，InnoDB通过Redo Log重放未落盘的事务修改，恢复数据一致性。事务提交前崩溃可通过Undo Log回滚，提交后崩溃则通过Redo Log恢复。

下面介绍下redo log的工作原理

WAL机制

Redo Log基于Write-Ahead Logging（WAL）机制，即“先写日志，后写磁盘”。事务提交时，先将修改记录写入Redo Log Buffer并刷盘，再异步将内存中的脏页写入磁盘。这一机制通过顺序写（日志）替代随机写（数据页），显著降低IO开销。

即使事务提交后脏页未落盘，Redo Log的存在仍能保证数据可恢复，从而提升性能并保障持久性。



redo log 和 undo log 配合起来的作用就是：

- 事务提交前崩溃，通过 undo log 回滚事务
- 事务提交后崩溃，通过 redo log 恢复事务

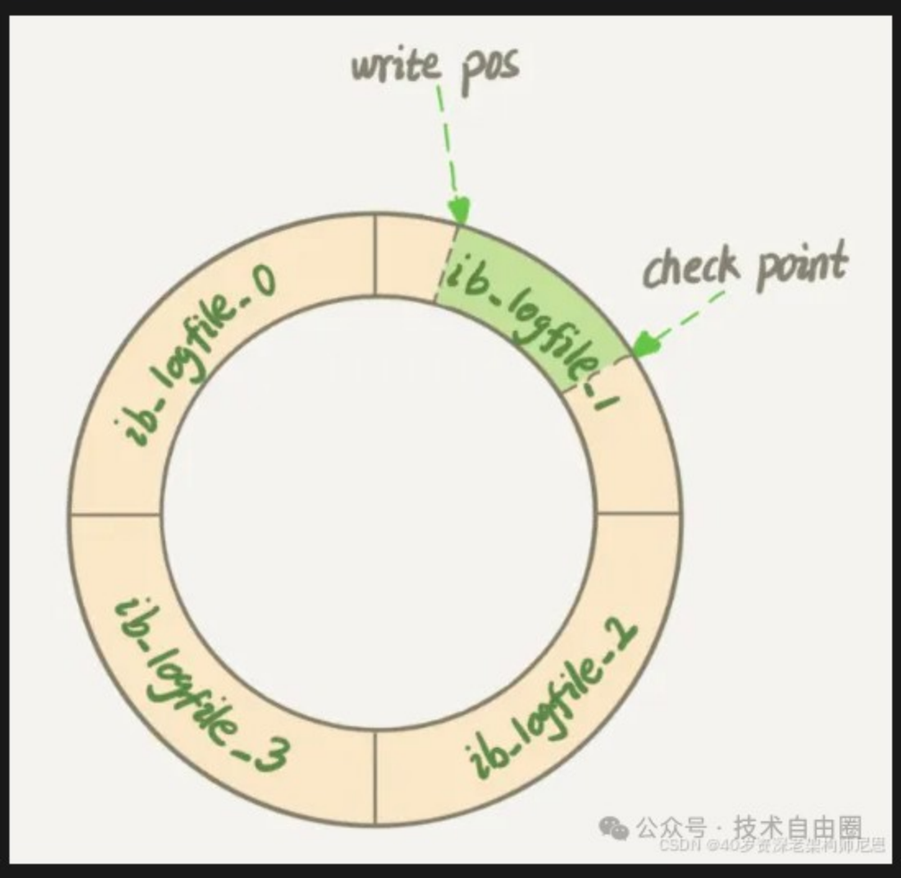
循环写入

Redo Log采用固定大小的循环写入方式，由多个日志文件组成文件组（如4个1GB文件）。

通过`write pos`（当前写入位置）和`checkpoint`（可擦除位置）两个指针管理日志覆盖。

当日志写满时，`write pos`追上`checkpoint`，触发CheckPoint操作，将脏页刷盘并推进`checkpoint`。

这种环形结构避免了日志文件无限增长，同时确保旧日志在数据页落盘后可安全覆盖。



- write pos：当前记录写到的位置，或者说 当前redo log文件写到了哪个位置
- checkpoint：当前要擦除的位置，或者说 目前redo log文件哪些记录可以被覆盖

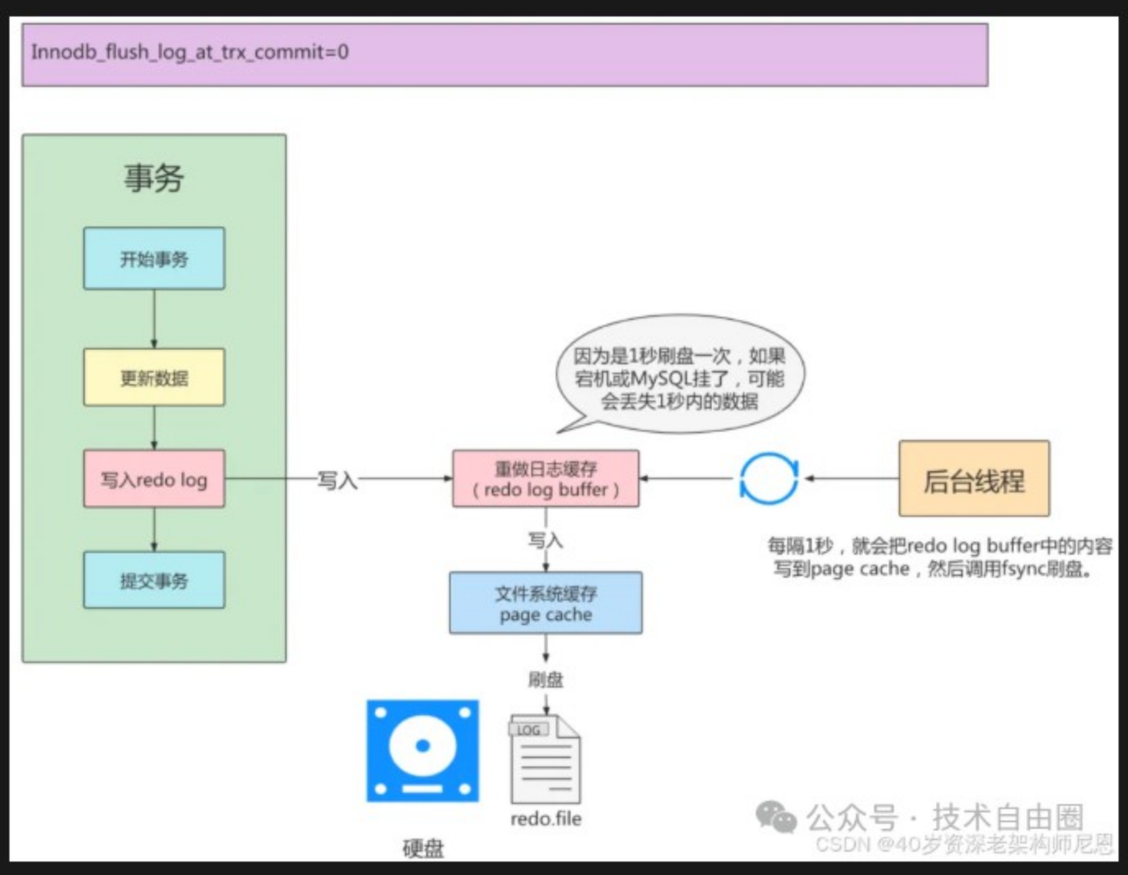
这两个指针把整个环形划成了几部分

- write pos - checkpoint：待写入的部分
- checkpoint - write pos：还未刷入磁盘的记录

刷盘策略

Redo Log刷盘由`innodb_flush_log_at_trx_commit`参数控制：

- 1（默认）：每次事务提交时强制刷盘（fsync），确保数据不丢失，但性能最低
- 2：事务提交时写入OS缓存（Page Cache），由操作系统异步刷盘，性能较高但存在5秒内数据丢失风险
- 0：每秒后台刷盘一次，性能最优但可能丢失最近1秒数据
- 不同策略平衡了可靠性与性能，适用于不同业务场景（如金融系统选1，高并发场景选2或0）



3、bin log日志

bin log – 一致性 + 持久性

什么是bin log日志

Binlog（Binary Log） 是 MySQL 的核心日志机制，记录所有对数据库的 DDL 和 DML 变更操作（如增删改表结构、数据），但不包括查询语句（如 SELECT）。

bin log 核心作用是实现 数据恢复 和 主从复制一致性。

redo log 和bin log的区别：

- redo log（重做日志）让InnoDB存储引擎拥有了崩溃恢复能力。
- binlog（归档日志）保证了MySQL集群架构的数据一致性。

通过一张图比较下二者区别

特性	Bin Log	Redo Log
归属	MySQL Server 层	InnoDB 存储引擎
日志类型	逻辑日志（SQL 或行变更）	物理日志（数据页修改）
写入方式	追加写入（文件无限增长）	循环写入（固定大小）
持久化时机	事务提交时	事务提交时（强制刷盘）
主要用途	主从复制、时间点恢复	崩溃恢复、事务持久性
存储引擎依赖	与存储引擎无关	仅 InnoDB

bin log日志格式

- STATEMENT：记录 SQL 语句，日志量小但存在主从不一致风险（如使用 NOW()）。
- ROW：记录行级变更（旧值/新值），数据一致性高但日志量大。
- MIXED：默认模式，自动切换 STATEMENT 和 ROW，平衡性能与一致性。

bin log日志刷盘参数

刷盘时机：事务提交时，日志先写入内存缓存（binlog_cache），再根据 sync_binlog 参数决定是否持久化到磁盘。

参数配置：

- sync_binlog=0：依赖系统刷盘，性能高但数据易丢失。
- sync_binlog=1：每次提交立即刷盘，最安全但性能损耗大。
- sync_binlog=N：累积 N 个事务后刷盘，折中方案。

遇到问题，找老架构师取经

借助此文的问题 套路 ，大家可以 放手一试，**保证 offer直接到手，还有可能会 涨薪 100%-200%。**

后面，尼恩java面试宝典回录成视频， 给大家打造一套进大厂的塔尖视频。

在面试之前，建议大家系统化的刷一波 5000页《尼恩Java面试宝典PDF》，里边有大量的大厂真题、面试难题、架构难题。

很多小伙伴刷完后， 吊打面试官， 大厂横着走。

在刷题过程中，如果有啥问题，大家可以来 找 40岁老架构师尼恩交流。

另外，如果没有面试机会，**可以找尼恩来改简历、做帮扶。**

遇到职业难题，找老架构取经， 可以省去太多的折腾，省去太多的弯路。

尼恩指导了大量的小伙伴上岸，前段时间，**刚指导 32岁 高中生，冲大厂成功。特批 成为 架构师，年薪 50W，逆天改命 ！！！。**

狠狠卷，实现 “offer自由” 很容易的， 前段时间一个武汉的跟着尼恩卷了2年的小伙伴， 在极度严寒/痛苦被裁的环境下， offer拿到手软， 实现真正的 “offer自由” 。

冲大厂 案例： 全网顶尖、高薪案例， 进大厂拿高薪， 实现薪酬腾飞、人生逆袭

涨一倍：从30万 涨 60万，3年经验小伙 冲大厂成功，逆天了 ！！！

阿里+美团offer：25岁 屡战屡败 绝望至极。找尼恩转架构升级，1个月拿到阿里+美团offer，逆天改命年薪 50W

阿里offer：6年一本 不想 混小厂了。狠卷1年 拿到 得物 + 阿里 offer ， 彻底上岸 ，逆天改命

字节 offer：3年经验 ， 足足折腾1年后绝望了，找尼恩陪跑， 2个月 逆天改命 ，拿到 字节&携程 offer

小米offer：7年 普通小二本，冲 小米 成功，年薪60W， 吊打一大票 985/211，狠卷1年，足足涨了50%，人生逆袭

拼多多offer：2年经验60W破全网记录，顶尖案例， 2年经验年薪60W ，3大厂offer， 双非一本 秒杀985、秒杀211，逆天改命

美团offer：被裁后涨80%，大赚了。从小厂被裁，拿4大厂offer（申通、顺丰、携程、美团），大涨80%，26岁小伙被裁赚翻了

逆天大涨：暴涨200%，29岁/7年/双非一本 ， 从13K涨到 37K ，如何做到的？

逆天改命：27岁被裁2月，转P6降维攻击，2个月提 JD/PDD 两大offer，时来运转，人生翻盘!! 大逆袭!!