图文 03 用一次数据更新流程，初步了解InnoDB存储引擎的架构设计

用一次数据更新流程，初步了解InnoDB存储引擎的架构设计

如何提问：每篇文章都有评论区，大家可以尽情留言提问，我会逐一答疑

如何加群：购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群，一个非常纯粹的技术交流的地方

具体加群方式，请参见目录菜单下的文档：**《MySQL专栏付费用户如何加群》**（购买后可见）

1、更新语句在MySQL中是如何执行的?

之前我们已经分析了MySQL架构上的整体设计原理，现在对一条SQL语句从我们的系统层面发送到MySQL中，然后一步一步执行这条SQL的流程，都有了一个整体的了解。

我们已经知道了，MySQL最常用的就是InnoDB存储引擎，那么我们今天借助一条更新语句的执行，来初步的了解一下InnoDB存储引擎的架构设计。

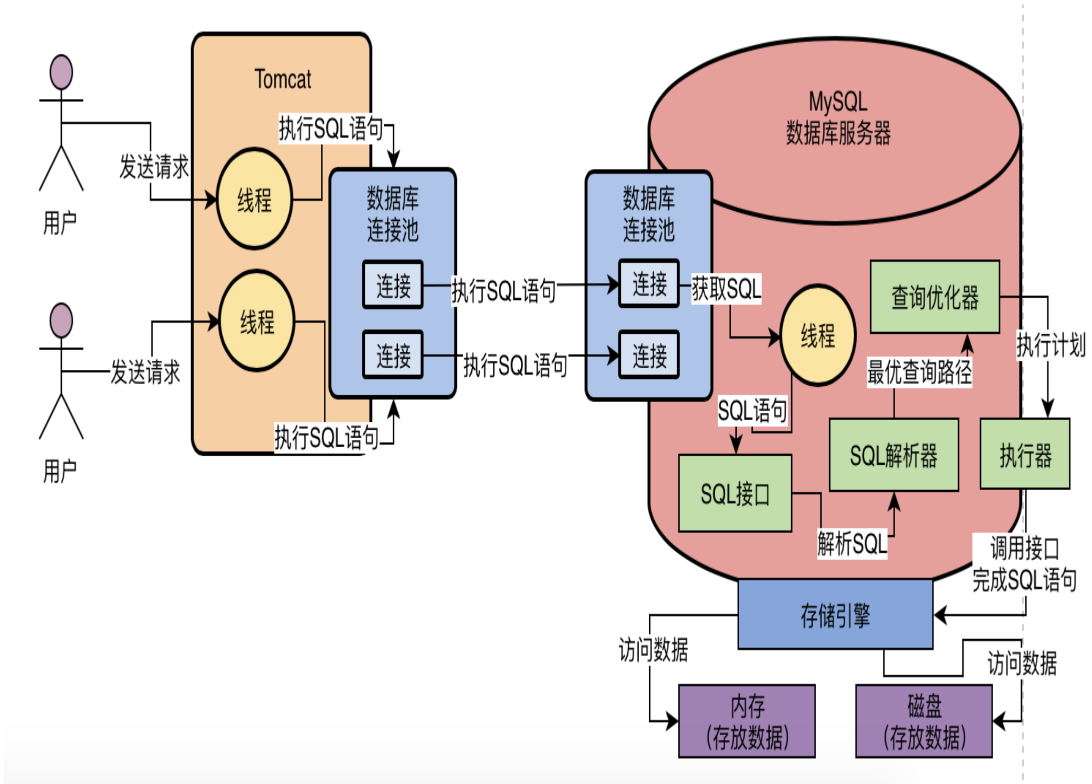
首先假设我们有一条SQL语句是这样的：

```
update users set name='xxx' where id=10
```

那么我们先想一下这条SQL语句是如何执行的?

首先肯定是我们的系统通过一个数据库连接发送到了MySQL上，然后肯定会经过SQL接口、解析器、优化器、执行器几个环节，解析SQL语句，生成执行计划，接着去由执行器负责这个计划的执行，调用InnoDB存储引擎的接口去执行。

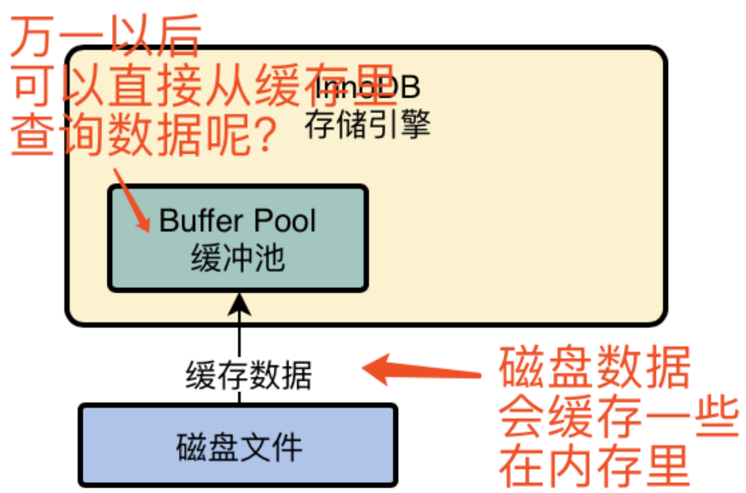
所以先看下图，大致还是会走下图的这个流程



今天我们就来探索一下这个存储引擎里的架构设计，以及如何基于存储引擎完成一条更新语句的执行

2、InnoDB的重要内存结构：缓冲池

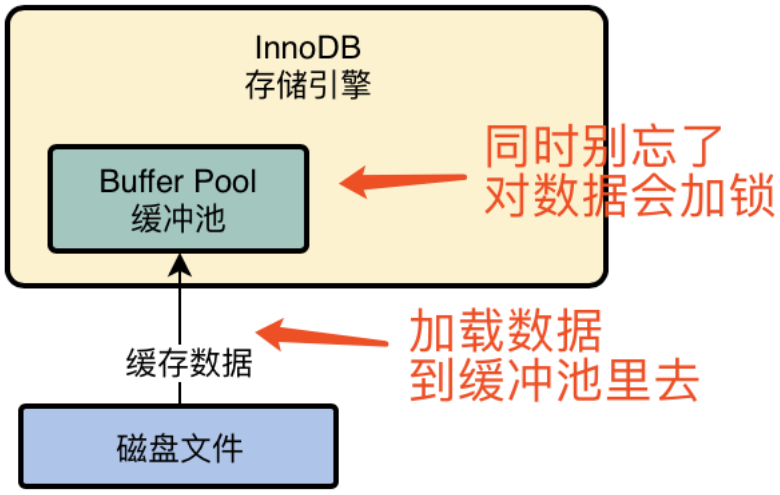
InnoDB存储引擎中有一个非常重要的放在内存里的组件，就是缓冲池（Buffer Pool），这里面会缓存很多的数据，以便于以后在查询的时候，万一你要是内存缓冲池里有数据，就可以不用去查磁盘了，我们看下图。



所以当我们的InnoDB存储引擎要执行更新语句的时候，比如对“id=10”这一行数据，他其实会先将“id=10”这一行数据看看是否在缓冲池里，如果不在的话，那么会直接从磁盘里加载到缓冲池里来，而且接着还会对这行记录加独占锁。

因为我们想一下，在我们更新“id=10”这一行数据的时候，肯定是不允许别人同时更新的，所以必须要对这行记录加独占锁

至于锁的详细分析，我们后续也会有，大家不用着急，在这里先初步了解即可，我们看下面的图

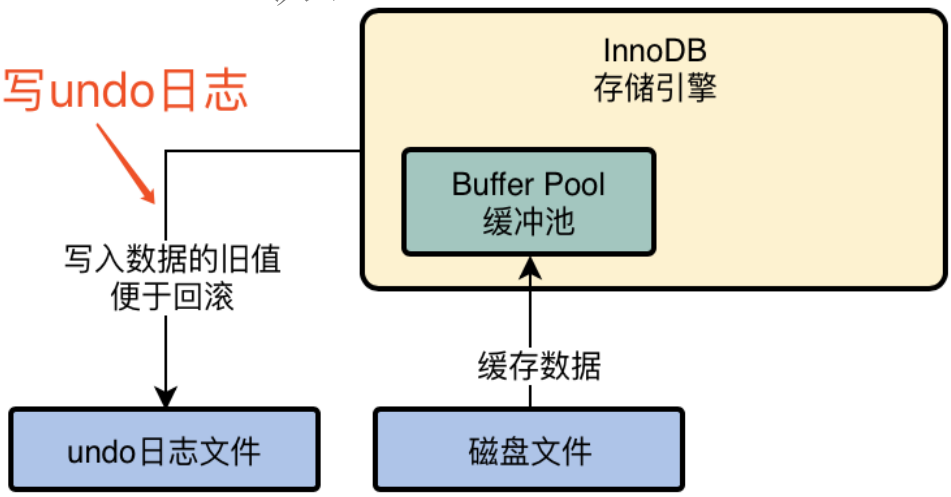


3、undo日志文件：如何让你更新的数据可以回滚?

接着下一步，假设“id=10”这行数据的name原来是“zhangsan”，现在我们要更新为“xxx”，那么此时我们得先把要更新的原来的值“zhangsan”和“id=10”这些信息，写入到undo日志文件中去。

其实稍微对数据库 有一点了解的同学都应该知道，如果我们执行一个更新语句，要是他是在一个事务里的话，那么事务提交之前我们都是可以对数据进行回滚的，也就是把你更新为“xxx”的值回滚到之前的“zhangsan”去。

所以为了考虑到未来可能要回滚数据的需要，这里会把你更新前的值写入undo日志文件，我们看下图。



4、更新buffer pool中的缓存数据

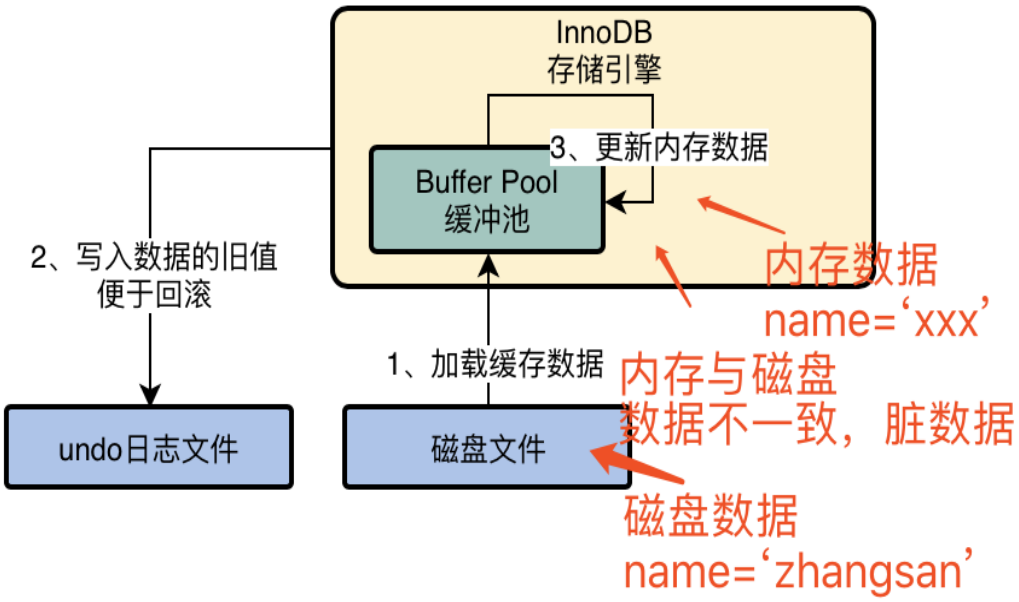
当我们把要更新的那行记录从磁盘文件加载到缓冲池，同时对他加锁之后，而且还把更新前的旧值写入undo日志文件之后，我们就可以正式开始更新这行记录了，更新的时候，先是会更新缓冲池中的记录，此时这个数据就是脏数据了。

这里所谓的更新内存缓冲池里的数据，意思就是把内存里的“id=10”这行数据的name字段修改为“xxx”

那么为什么说此时这行数据就是脏数据了呢?

因为这个时候磁盘上“id=10”这行数据的name字段还是“zhangsan”，但是内存里这行数据已经被修改了，所以就会叫他是脏数据。

我们看下图，我同时把几个步骤的序号标记出来了。



5、Redo Log Buffer：万一系统宕机，如何避免数据丢失?

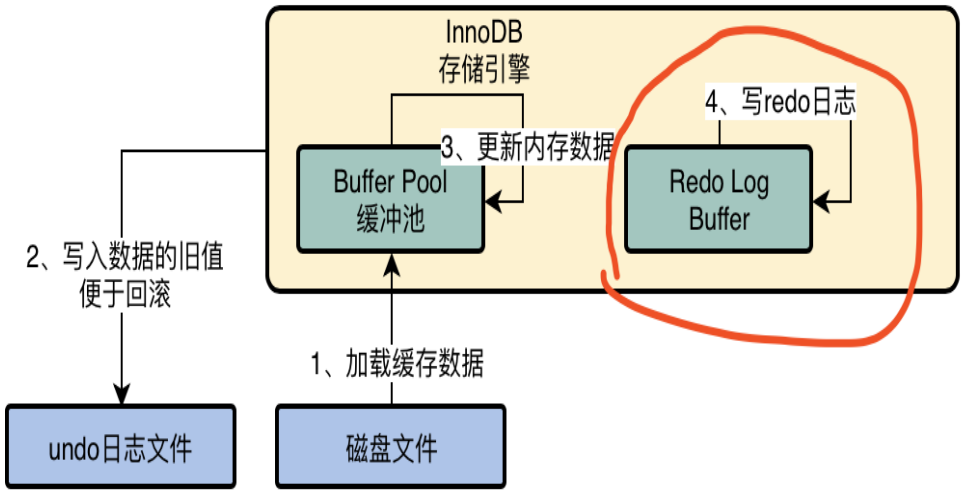
接着我们来思考一个问题，按照上图的说明，现在已经把内存里的数据进行了修改，但是磁盘上的数据还没修改

那么此时万一MySQL所在的机器宕机了，必然会导致内存里修改过的数据丢失，这可怎么办呢?

这个时候，就必须要把对内存所做的修改写入到一个**Redo Log Buffer**里去，这也是内存里的一个缓冲区，是用来存放redo日志的

所谓的redo日志，就是记录下来你对数据做了什么修改，比如对“id=10这行记录修改了name字段的值为xxx”，这就是一个日志。

我们先看下图的示意



这个redo日志其实是用来在MySQL突然宕机的时候，用来恢复你更新过的数据的，但是我们现在还没法直接讲解redo是如何使用的，毕竟现在redo日志还仅仅停留在内存缓冲里

大家稍安勿躁，继续往下看

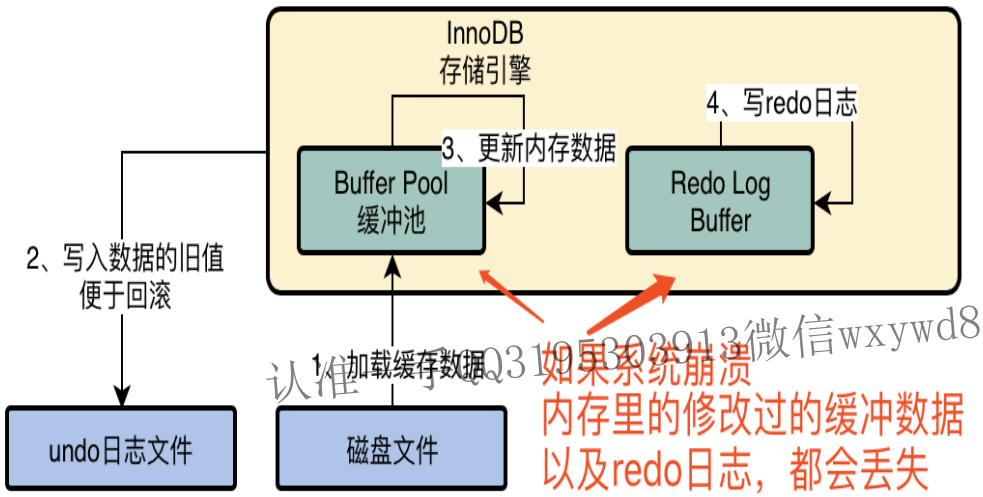
6、如果还没提交事务，MySQL宕机了怎么办?

这里我们假设每个人看专栏的人，都对MySQL的基本SQL语法、事务的基本概念以及索引的基本概念有一个基础的了解，因为但凡一个后端工程师，要跟数据库打交道，必然会跟这些概念有一定的了解。

所以我们都知道，其实在数据库中，哪怕执行一条SQL语句，其实也可以是一个独立的事务，只有当你提交事务之后，SQL语句才算执行结束。

所以这里我们都知道，到目前为止，其实还没有提交事务，那么此时如果MySQL崩溃，必然导致内存里Buffer Pool中的修改过的数据都丢失，同时你写入Redo Log Buffer中的redo日志也会丢失

我们看下图



那么此时数据丢失要紧吗?

其实是不要紧的，因为你一条更新语句，没提交事务，就代表他没执行成功，此时MySQL宕机虽然导致内存里的数据都丢失了，但是你会发现，磁盘上的数据依然还停留在原样子。

也就是说，“id=1”的那行数据的name字段的值还是老的值，“zhangsan”，所以此时你的这个事务就是执行失败了，没能成功完成更新，你会收到一个数据库的异常。然后当mysql重启之后，你会发现你的数据并没有任何变化。

所以此时如果mysql宕机，不会有任何的问题。

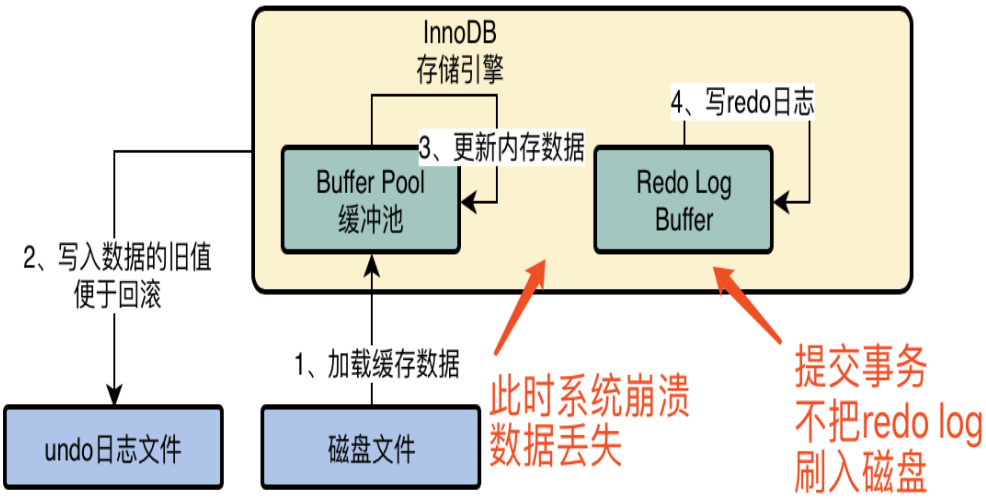
7、提交事务的时候将redo日志写入磁盘中

接着我们想要提交一个事务了，此时就会根据一定的策略把redo日志从redo log buffer里刷入到磁盘文件里去。

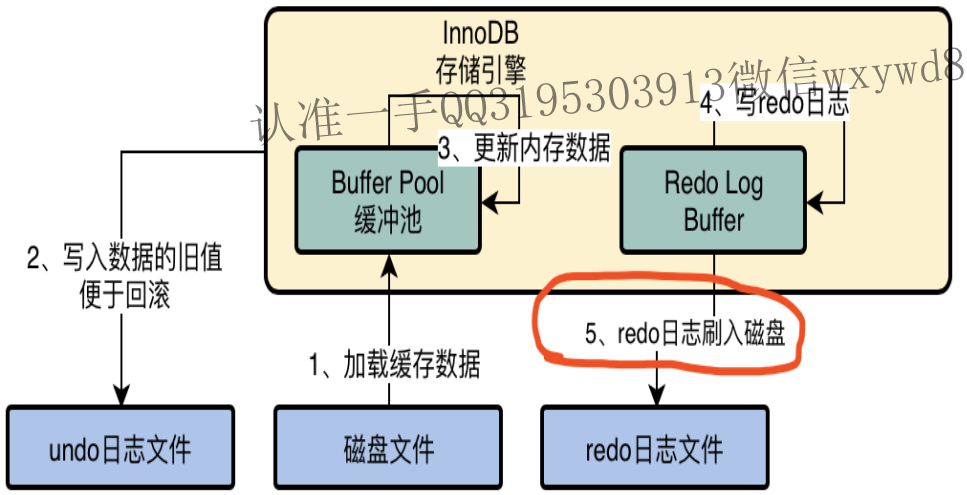
此时这个策略是通过innodb_flush_log_at_trx_commit来配置的，他有几个选项。

当这个参数的值为0的时候，那么你提交事务的时候，不会把redo log buffer里的数据刷入磁盘文件的，此时可能你都提交事务了，结果mysql宕机了，然后此时内存里的数据全部丢失。

相当于你提交事务成功了，但是由于MySQL突然宕机，导致内存中的数据和redo日志都丢失了，我们看下图:



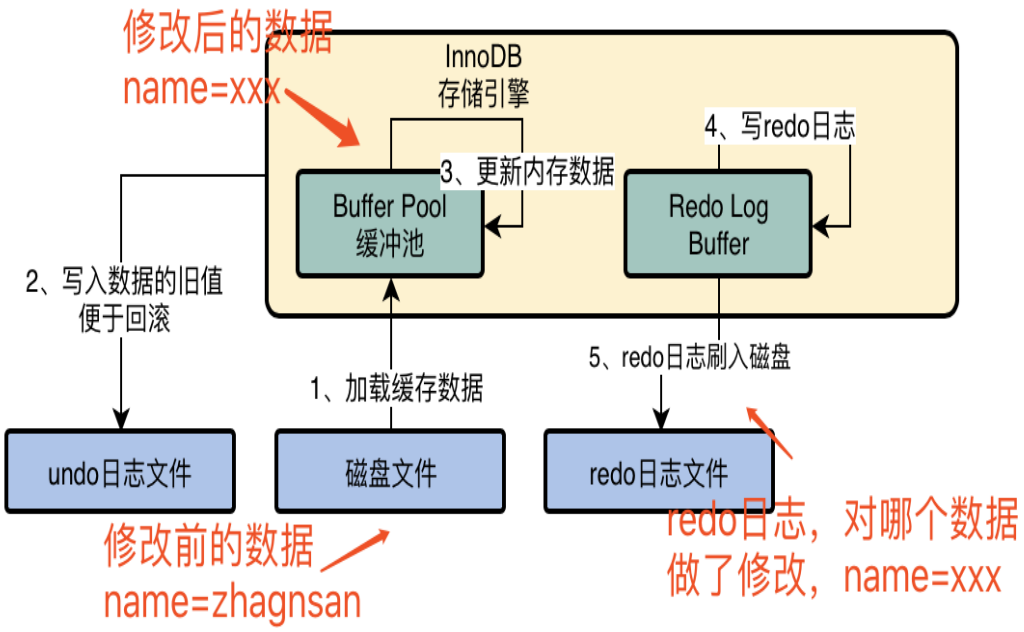
当这个参数的值为1的时候，你提交事务的时候，就必须把redo log从内存刷入到磁盘文件里去，只要事务提交成功，那么redo log就必然在磁盘里了，我们看下图：



那么只要提交事务成功之后，redo日志一定在磁盘文件里，此时你肯定会有一条redo日志说了，“我此时对哪个数据做了一个什么修改，比如name字段修改为xxx了”。

然后哪怕此时buffer pool中更新过的数据还没刷新到磁盘里去，此时内存里的数据是已经更新过的“name=xxx”，然后磁盘上的数据还是没更新过的“name=zhangsan”。

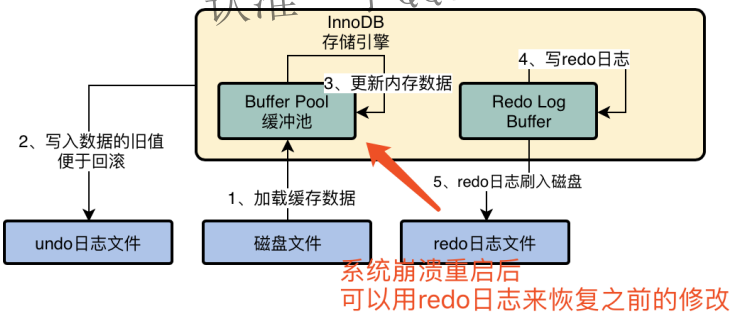
我们看下图，提交事务之后，可能处于的一个状态。



此时如果说提交事务后处于上图的状态，然后mysql系统突然崩溃了，此时会如何？会丢失数据吗？

肯定不会啊，因为虽然内存里的修改成name=xxx的数据会丢失，但是redo日志里已经说了，对某某数据做了修改name=xxx。

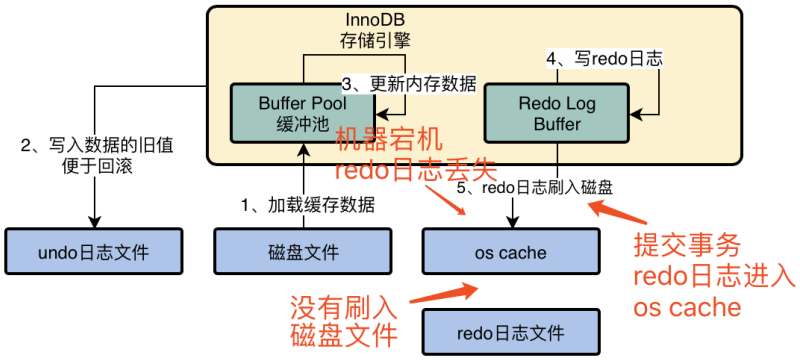
所以此时mysql重启之后，他可以根据redo日志去恢复之前做过的修改，我们看下图。



最后来看看，如果innodb_flush_log_at_trx_commit参数的值是2呢？

他的意思就是，提交事务的时候，把redo日志写入磁盘文件对应的os cache缓存里去，而不是直接进入磁盘文件，可能1秒后才会把os cache里的数据写入到磁盘文件里去。

这种模式下，你提交事务之后，redo log可能仅仅停留在os cache内存缓存里，没实际进入磁盘文件，万一此时你要是机器宕机了，那么os cache里的redo log就会丢失，同样会让你感觉提交事务了，结果数据丢了，看下图。



8、小思考题：三种redo日志刷盘策略到底选择哪一种?

今天给大家留一个小的思考题，大家觉得在提交事务的时候，我们对redo日志的刷盘策略应该选择哪一种？每一种刷盘策略的优缺点分别是什么？为什么?

欢迎大家在评论区留言和我交流

End
