图文 04 借着更新语句在InnoDB存储引擎中的执行流程，聊聊binlog是什么?

借着更新语句在InnoDB存储引擎中的执行流程，聊聊binlog是什么?

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、上一讲思考题解答：redo日志刷盘策略的选择建议

先给大家解释一下上一讲的思考题，我给大家的一个建议，其实对于redo日志的三种刷盘策略，我们通常建议是设置为1

也就是说，提交事务的时候，redo日志必须是刷入磁盘文件里的。

这样可以严格的保证提交事务之后，数据是绝对不会丢失的，因为有redo日志在磁盘文件里可以恢复你做的所有修改。

如果要是选择0的话，可能你提交事务之后，mysql宕机，那么此时redo日志没有刷盘，导致内存里的redo日志丢失，你提交的事务更新的数据就丢失了；

如果要是选择2的话，如果机器宕机，虽然之前提交事务的时候，redo日志进入os cache了，但是还没进入磁盘文件，此时机器宕机还是会导致os cache里的redo日志丢失。

所以对于数据库这样严格的系统而言，一般建议redo日志刷盘策略设置为1，保证事务提交之后，数据绝对不能丢失。

2、MySQL binlog到底是什么东西?

接着我们来看看MySQL binlog到底是个什么东西?

实际上我们之前说的redo log，他是一种偏向物理性质的重做日志，因为他里面记录的是类似这样的东西，“对哪个数据页中的什么记录，做了个什么修改”。

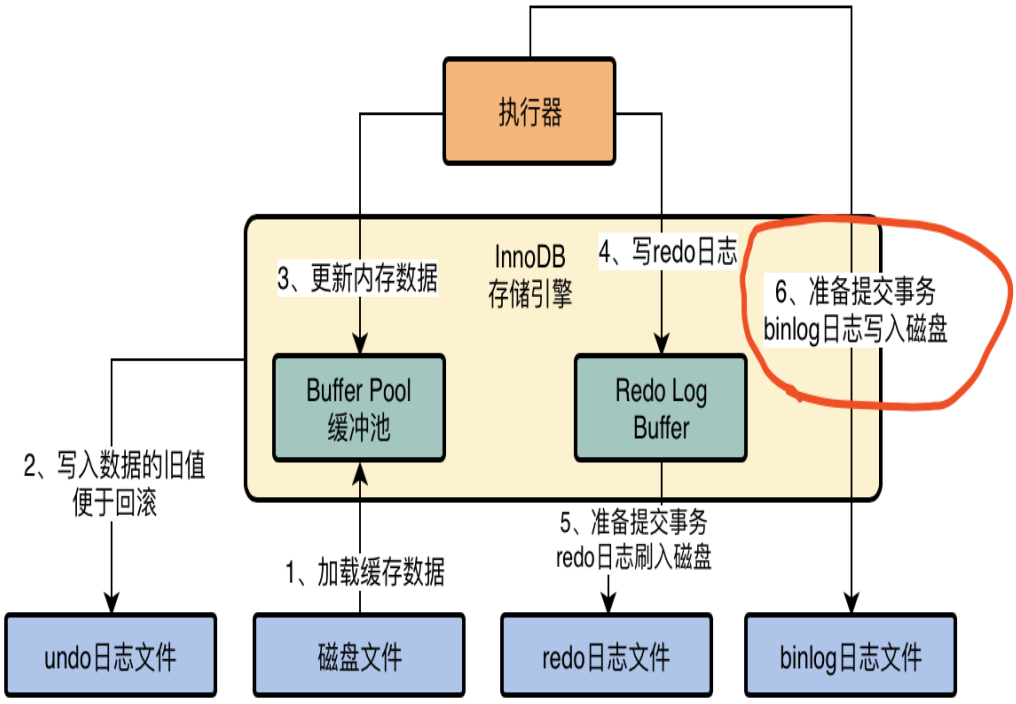
而且redo log本身是属于InnoDB存储引擎特有的一个东西。

而binlog叫做归档日志，他里面记录的是偏向于逻辑性的日志，类似于“对users表中的id=10的一行数据做了更新操作，更新以后的值是什么”

binlog不是InnoDB存储引擎特有的日志文件，是属于mysql server自己的日志文件。

3、提交事务的时候，同时会写入binlog

所以其实我们上一讲讲到，在我们提交事务的时候，会把redo log日志写入磁盘文件中去。然后其实在提交事务的时候，我们同时还会把这次更新对应的binlog日志写入到磁盘文件中去，如下图所示。



大家可以在这个图里看到一些变动，就是我把跟InnoDB存储引擎进行交互的组件加入了之前提过的执行器这个组件，他会负责跟InnoDB进行交互，包括从磁盘里加载数据到Buffer Pool中进行缓存，包括写入undo日志，包括更新Buffer Pool里的数据，以及写入redo log buffer，redo log刷入磁盘，写binlog，等等。

实际上，执行器是非常核心的一个组件，负责跟存储引擎配合完成一个SQL语句在磁盘与内存层面的全部数据更新操作。

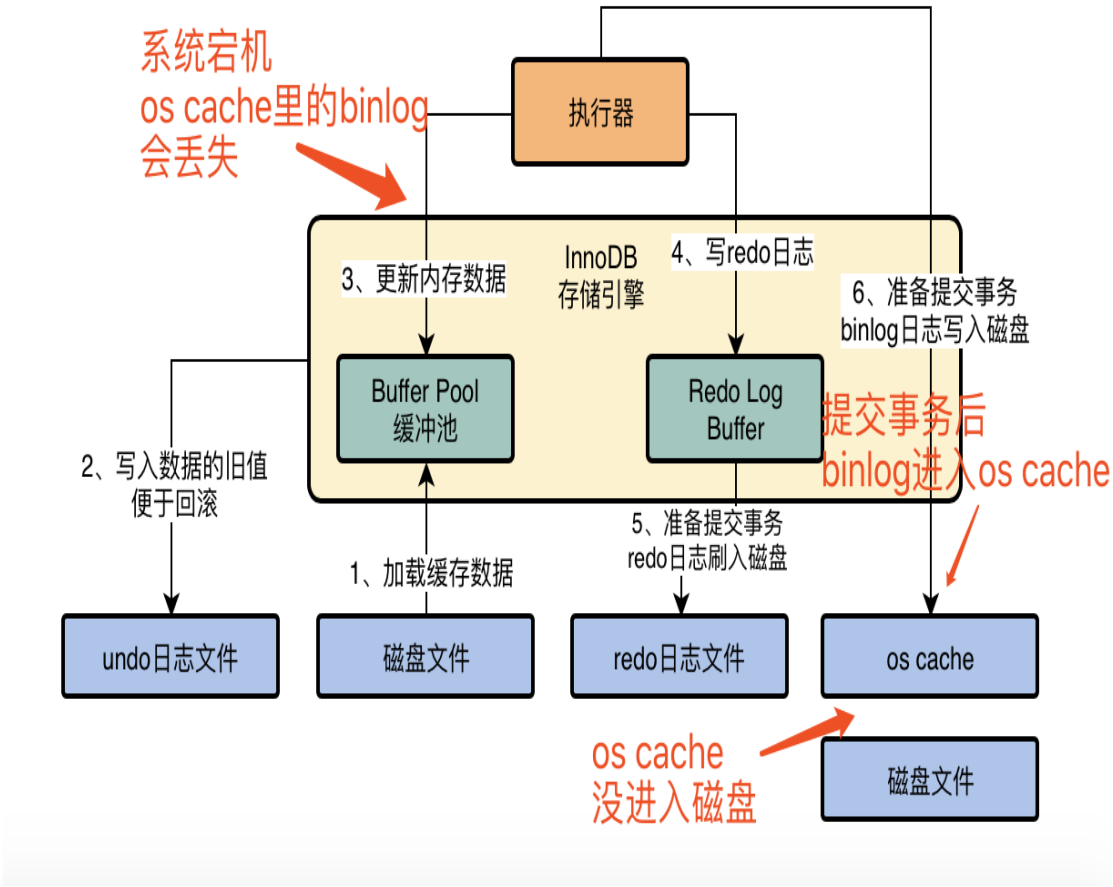
而且我们在上图可以看到，我把一次更新语句的执行，拆分为了两个阶段，上图中的1、2、3、4几个步骤，其实本质是你执行这个更新语句的时候干的事。

然后上图中的5和6两个步骤，是从你提交事务开始的，属于提交事务的阶段了。

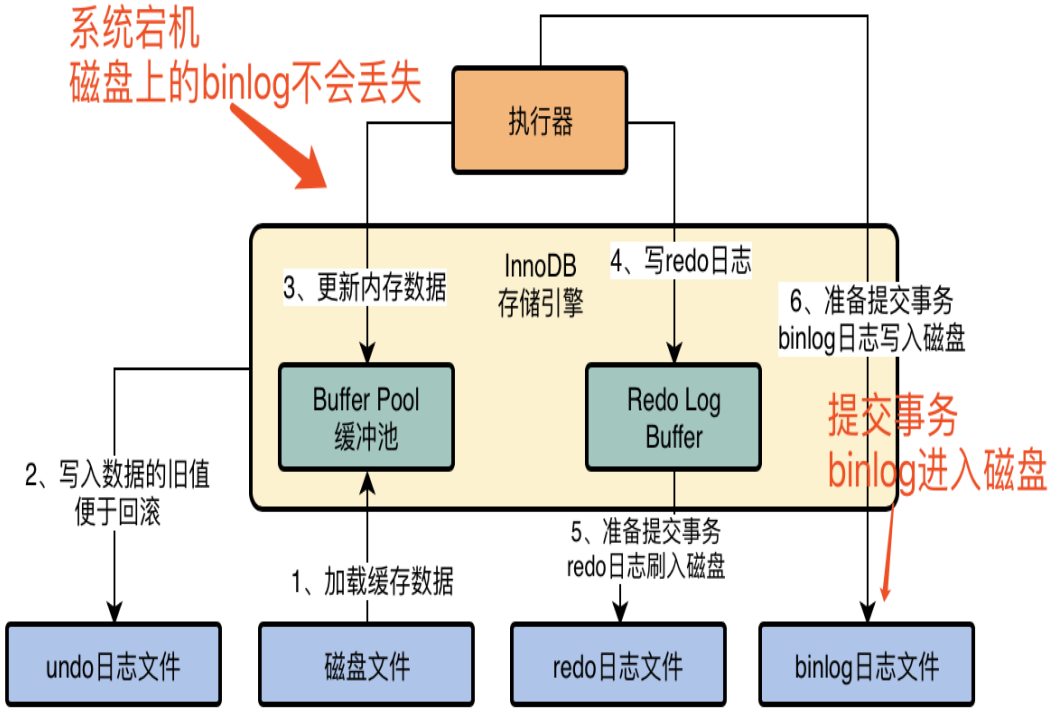
4、binlog日志的刷盘策略分析

对于binlog日志，其实也有不同的刷盘策略，有一个**sync_binlog**参数可以控制binlog的刷盘策略，他的默认值是0，此时你把binlog写入磁盘的时候，其实不是直接进入磁盘文件，而是进入os cache内存缓存。

所以跟之前分析的一样，如果此时机器宕机，那么你在os cache里的binlog日志是会丢失的，我们看下图的示意



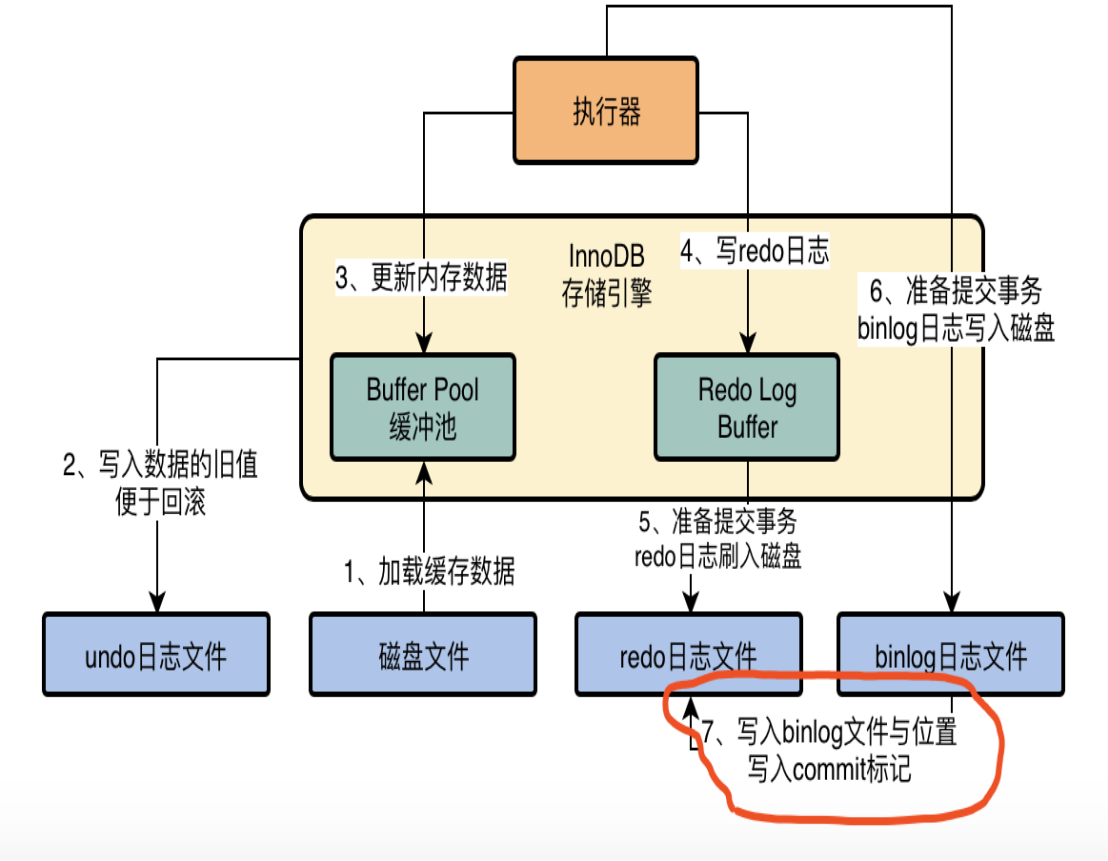
如果要是把**sync_binlog**参数设置为1的话，那么此时会强制在提交事务的时候，把binlog直接写入到磁盘文件里去，那么这样提交事务之后，哪怕机器宕机，磁盘上的binlog是不会丢失的，如下图所示



5、基于binlog和redo log完成事务的提交

当我们把binlog写入磁盘文件之后，接着就会完成最终的事务提交，此时会把本次更新对应的binlog文件名称和这次更新的binlog日志在文件里的位置，都写入到redo log日志文件里去，同时在redo log日志文件里写入一个commit标记。

在完成这个事情之后，才算最终完成了事务的提交，我们看下图的示意。



6、最后一步在redo日志中写入commit标记的意义是什么?

这时候肯定有同学会问了，最后在redo日志中写入commit标记有什么意义呢?
说白了，他其实是用来保持redo log日志与binlog日志一致的。

我们来举个例子，假设我们在提交事务的时候，一共有上图中的5、6、7三个步骤，必须是三个步骤都执行完毕，才算是提交了事务。那么在我们刚完成步骤5的时候，也就是redo log刚刷入磁盘文件的时候，mysql宕机了，此时怎么办?

这个时候因为没有最终的事务commit标记在redo日志里，所以此次事务可以判定为不成功。不会说redo日志文件里有这次更新的日志，但是binlog日志文件里没有这次更新的日志，不会出现数据不一致的问题。

如果要是完成步骤6的时候，也就是binlog写入磁盘了，此时mysql宕机了，怎么办?

同理，因为没有redo log中的最终commit标记，因此此时事务提交也是失败的。

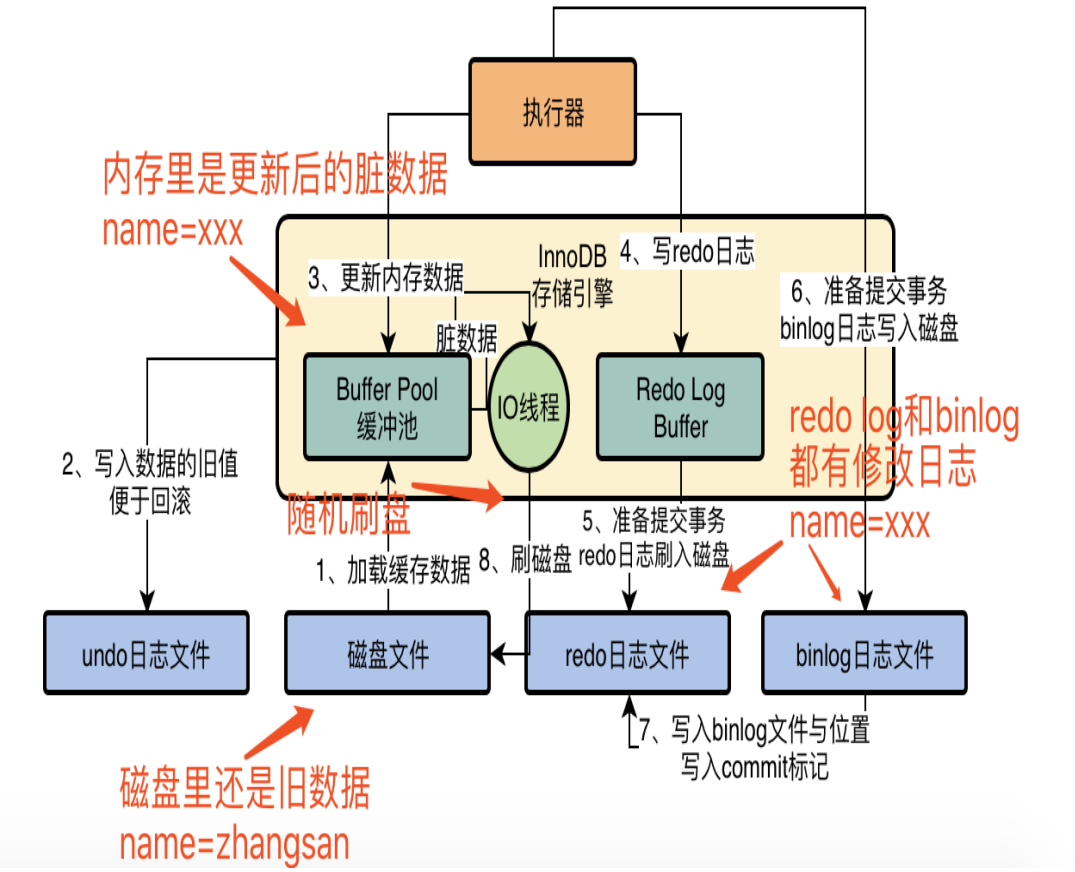
必须是在redo log中写入最终的事务commit标记了，然后此时事务提交成功，而且redo log里有本次更新对应的日志，binlog里也有本次更新对应的日志 ，redo log和binlog完全是一致的。

7、后台IO线程随机将内存更新后的脏数据刷回磁盘

现在我们假设已经提交事务了，此时一次更新“update users set name='xxx' where id=10”，他已经把内存里的buffer pool中的缓存数据更新了，同时磁盘里有redo日志和binlog日志，都记录了把我们指定的“id=10”这行数据修改了“name='xxx'”。

此时我们会思考一个问题了，但是这个时候磁盘上的数据文件里的“id=10”这行数据的name字段还是等于zhangsan这个旧的值啊!

所以MySQL有一个后台的IO线程，会在之后某个时间里，随机的把内存buffer pool中的修改后的脏数据给刷回到磁盘上的数据文件里去，我们看下图：



当上图中的IO线程把buffer pool里的修改后的脏数据刷回磁盘的之后，磁盘上的数据才会跟内存里一样，都是name=xxx这个修改以后的值了！

在你IO线程把脏数据刷回磁盘之前，哪怕mysql宕机崩溃也没关系，因为重启之后，会根据redo日志恢复之前提交事务做过的修改到内存里去，就是id=10的数据的name修改为了xxx，然后等适当时机，IO线程自然还是会把这个修改后的数据刷到磁盘上的数据文件里去的

8、基于更新数据的流程，总结一下InnoDB存储引擎的架构原理

大家通过一次更新数据的流程，就可以清晰地看到，InnoDB存储引擎主要就是包含了一些buffer pool、redo log buffer等内存里的缓存数据，同时还包含了一些undo日志文件，redo日志文件等东西，同时mysql server自己还有binlog日志文件。

在你执行更新的时候，每条SQL语句，都会对应修改buffer pool里的缓存数据、写undo日志、写redo log buffer几个步骤；

但是当你提交事务的时候，一定会把redo log刷入磁盘，binlog刷入磁盘，完成redo log中的事务commit标记；最后后台的IO线程会随机的把buffer pool里的脏数据刷入磁盘里去。

9、思考题：执行更新操作的时候，为什么不能执行修改磁盘上的数据？

好了，今天的文章接近尾声，咱们再来思考一个问题：

> 为什么MySQL在更新数据的时候，要大费周章的搞这么多事情，包括buffer pool、redo log、undo log、binlog、事务提交、脏数据。引入了一大堆的概念，有复杂的流程和步骤。
>
> 为什么他反而最关键的修改磁盘里的数据，要通过IO线程不定时的去执行？
>
> 为什么他不干脆直接就每次执行SQL语句，直接就更新磁盘里的数据得了？

希望大家踊跃思考，在评论区给出自己的答案。

End

狸猫技术窝精品专栏及课程推荐：

- 《从零开始带你成为消息中间件实战高手》
- 《21天互联网Java进阶面试训练营》（分布式篇）
- 《互联网Java工程师面试突击》（第1季）
- 《互联网Java工程师面试突击》（第3季）
- 《从零开始带你成为JVM实战高手》