第二贴redoLog 和binLog

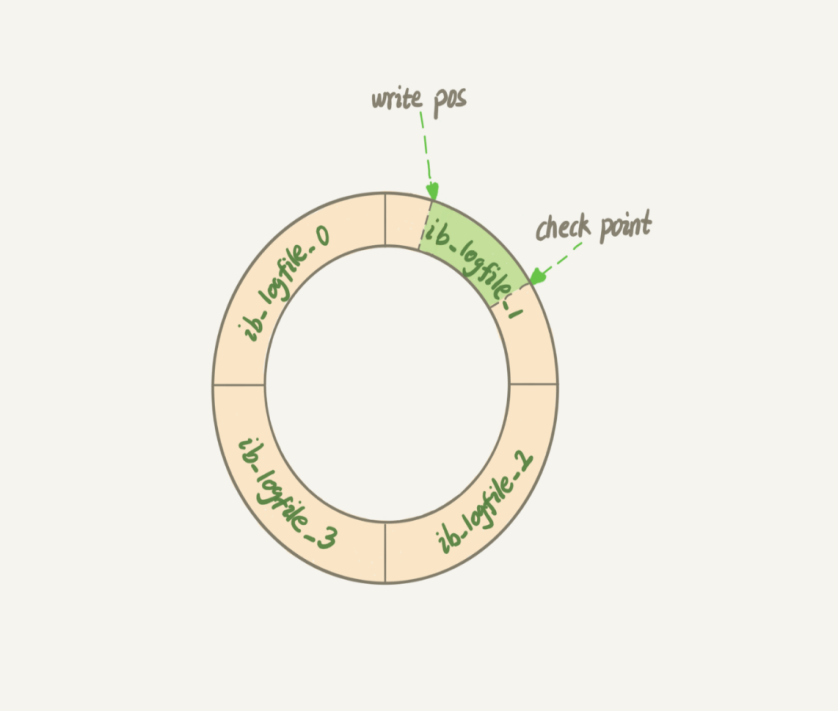
RedoLog是InnoDB特有的日志，

Binlog 是server层的

Redolog 先将修改写入到内存，系统空闲，然后将更新的操作写入到磁盘，WAL全称是

Write-Ahead Loggin。先写日志后，再写磁盘，

InnoDB的redo log是固定大小的，比如可以配置为一组4个文件，每个文件的大小是1GB,



Write-pos是当前位置，一边写一边后移，写写到第 3 号文件末尾后就回到 0 号文件开头。checkpoint 是当前要擦除的位置，也是往后推移并且循环的，擦除记录前要把记录更新到数据文件。

write pos 和 checkpoint 之间的是空余的空间可以写新的操作，如果write-pos追上checkpoint表示没有空间更新操作，此时不能再执行新的更新，得停下来先擦掉一些记录，把checkpoint推进一下

有了 redo log，InnoDB 就可以保证即使数据库发生异常重启，之前提交的记录都不会丢失，这个能力称为crash-safe.

**Binlog**

有两种模式

statement格式记录的sql,

row格式会记录行的内容，记两条，更新前和后都有

还有一种mixed 格式，文中未提到

**我想你肯定会问，为什么会有两份日志呢？**

因为最开始是MySQL并没有InnoDB引擎，自带MyISAM引擎，但是引擎没有crash-safe能力，binlog日志只能用于归档，而InnoDB是以插件的形式引入到MySQL,只依靠binlog是没有crash-safe能力的，所以InnoDB使用另一套日志系统--也就是redolog实现crash-safe

Update T set c = c+1 where ID= 2;

两种日志的不同点

**1.**执行器先找引擎取Id=2这一行，ID是主键，引擎直接用树搜索引找到这一行，如果ID=2,所在数据也在内存中，直接返回给执行器，否则需要从磁盘读入到内存，然后在返回。

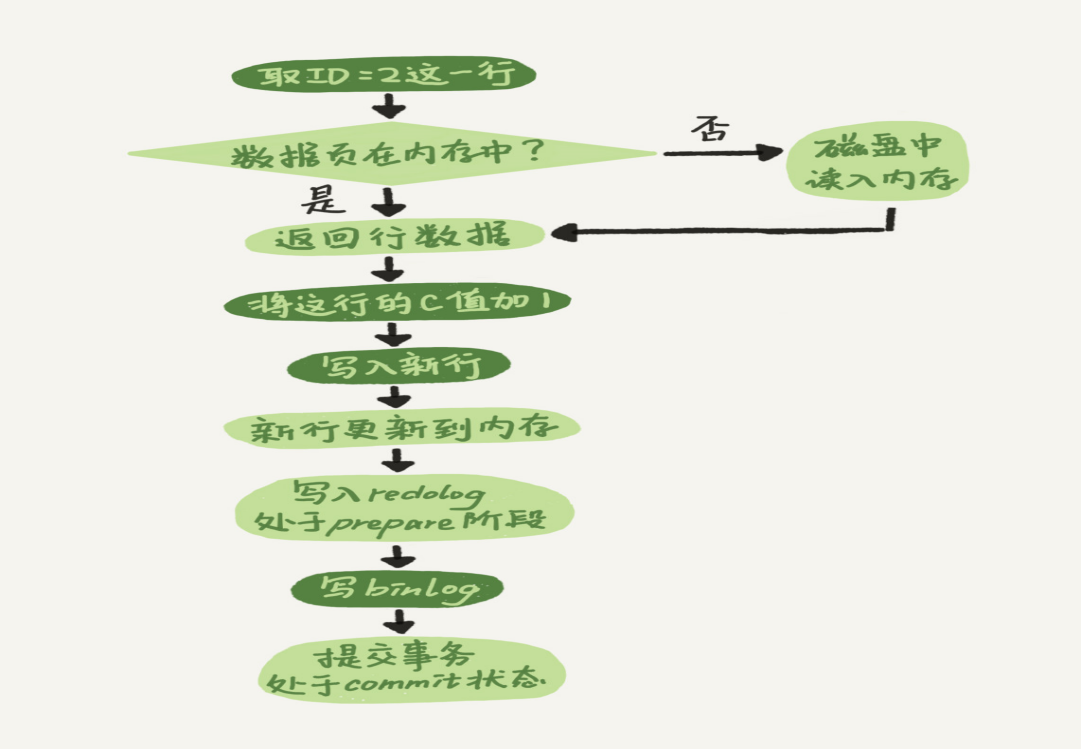
**2**.执行器拿到引擎给的行数据，把这个值加1,得到新的一行数据，在调用引擎接口写入这行新数据

**3.**引擎将这行数据更新到内存中，同时将这个更新操作记录的redolog中，此时redolog出入prepare状态，然后告诉执行器执行完了，随时可以提交事务

**4**.执行器生成这个操作的binlog,并把binlong写入磁盘

**5.**执行器调用引擎接口提交事务，引擎把刚刚写入的redolog改成commit状态，更细完成

**这里我给出这个 update 语句的执行流程图，图中浅色框表示是在 InnoDB 内部执行的，深色框表示是在执行器中执行的。**



你可能注意到了，最后三步看上去有点“绕”，将 redo log 的写入拆成了两个步骤：prepare和 commit，这就是**"两阶段提交"**。

**两阶段提交**-- 目的是保持两个热值逻辑上一致

binlog会记录所有的逻辑操作，并采用“追加写”的形式

两种日志有以下三点不同

1. redo log 是InnoDB引擎特有的；binlog是MySQL的Server层实现的，所有引擎都可以使用
2. Redo log 是物理日志，记录在“某个数据页上做了什么修改”，binlog是逻辑日志，记录是这个语句原始逻辑，比如：“给ID=2这一行c字段加1”

3.redo log是循环写，空间固定会用完；binlog是可以追加写入的，“追加写”是指binlog文件写到一个大小会切换到下一个，并且不覆盖以前的日志。

**数据恢复**

首先，找到最近的一次全量备份，从这个备份恢复到临时库

然后，从备份的时间点开始，将备份的binlog依次取出来，重新放到中午误删表之前的那个时刻

**为什么用两阶段提交，用反证法说明？**

仍然使用前边的sql,假设当前的ID=2的行，字段c的值是0，在假定执行update语句过程中在写完第一个日志，第二个日志还没有写完期间发生creash,会出现什么情况？

1先写redolog后写binlog

假定在写完redolog,binlog还没有写完的时候，MySQL进程异常重启，由于我们前面说过redo log写完，系统即是崩溃，仍然能够把数据恢复回来，所以回复后这一行c的值是1.

但是由于binlog没有写完crash,这时候binlog里面没有记录这个语句，因此，之后备份日志的时候，存起来的binlog里面就没有这条语句。

如果需要要这个binlog来恢复临时库的话，由于这个binlog的丢失，这个临时库就会少一次的更新，恢复出来的c的值是0，与原库的值不同。

2 先写binlog 再写redo log

如果在binlog写完之后crash,由于redo log还没有写，崩溃恢复以后这个事务无效，所以这一行的c的值是0。但是binlog里面已经记录了把“c”从0改成1这个日志。所以，在之后用binlog来恢复的时候就多了一个事务出来，恢复出来的的这一行c的值就是1，与原库的值不同。

可以看到，如果不使用“两阶段提交”，那么数据库的状态就有可能和用它的日志恢复出来的库的状态不一致

**总结两个日志 redo log和bin log**

redo log 用于保证crash-save能力，innodb\_flush\_log\_at\_trx\_commit这个参数设置为1 ,表示每次事务的redo log都直接持久化到磁盘，这个参数建议设置为1，这样可以保证MySQL异常重启之后数据不丢失。

Sync\_binlog 这个参数设置为1的时候，表示每次binlog都持久化到磁盘，这个参数也建议设置为1，这样可以保证MySQL异常重启之后binlog不丢失。

**讨论题**

定期全量备份的周期“取决于系统重要性”，有一天一备，有的一周一备，那么什么场景下一天一备比一周一备更有优势呢？或者说，它影响这个数据库系统的那个指标？

好处是“最常恢复时间”更短。

在一天一备的模式里，更坏的情况下需要应用一天的binlog。比如，每天0点做一次全量备份，而恢复出一个到昨天晚上23点的备份。

一周一备最会的情况就要应用一周的binlog了。

系统对应的指标就是：RTO(恢复目标时间)

跟频繁的备份需要消耗更多的存储空间，所以RTO是成本换回来的。

**别人总结**

先说下阅读的收获：

1、更新的流程先写redo日志，写完后更新内存，到这里操作就直接返回了。后续的流程是生成此操作的binlog，然后写到磁盘

2、redo日志是存储引擎实现的，记录的在某个数据页做了什么修改，固定大小，默认为4GB，可以循环写，解决了每次更新操作写磁盘、查找记录、然后更新整个过程效率低下的问题，redo日志将磁盘的随机写变成了顺序写，这个机制是WAL，先写日志再刷磁盘。一句话，redo日志保证了事务ACID的特性

3、binglog日志Server层实现的，记录的是语句的原始逻辑，比如给ID=1的行的状态设置成1，追加写，上个文件写完回切换成下一个文件，类似滚动日志

4、两阶段提交，是为了解决redo log和binlog不一致问题的，这里的不一致是指redo log写成功binlog写失败了，由于恢复是根据binlog恢复的，这样恢复的时候就会少一条更新操作，导致和线上库的数据不一致。具体而言，两阶段是指prepare阶段和commit阶段，写完redo log进入prepare阶段，写完binlog进入commit阶段。

然后说下由redo log联想到之前遇到的一个问题：一个普通的select查询超过30ms，经过和DBA的联合排查，确认是由于MySQL“刷脏”导致的。

所谓刷脏就是由于内存页和磁盘数据不一致导致了该内存页是“脏页”，将内存页数据刷到磁盘的操作称为“刷脏”。刷脏是为了避免产生“脏页”，主要是因为MySQL更新先写redo log再定期批量刷到磁盘的，这就导致内存页的数据和磁盘数据不一致，为了搞清楚为什么“刷脏”会导致慢查，我们先分析下redo log再哪些场景会刷到磁盘。

场景1：redo log写满了，此时MySQL会停止所有更新操作，把脏页刷到磁盘

场景2：系统内存不足，需要将脏页淘汰，此时会把脏页刷到磁盘

场景3：系统空闲时，MySQL定期将脏页刷到磁盘

可以想到，在场景1和2都会导致慢查的产生，根据文章提到的，redo log是可以循环写的，那么即使写满了应该也不会停止所有更新操作吧，其实是会的，文中有句话“粉板写满了，掌柜只能停下手中的活，把粉板的一部分赊账记录更新到账本中，把这些记录从粉板删除，为粉板腾出新的空间”，这就意味着写满后是会阻塞一段时间的。

那么问题来了，innodb存储引擎的刷脏策略是怎么样的呢？通常而言会有两种策略：全量（sharp checkpoint）和部分（fuzzy checkpoint）。全量刷脏发生在关闭数据库时，部分刷脏发生在运行时。部分刷脏又分为定期刷脏、最近最少使用刷脏、异步/同步刷脏、脏页过多刷脏。暂时先写到这，后面打算写文详细介绍。