02 | 日志系统：一条SQL更新语句是如何执行的？

2018-11-16 林晓斌

前面我们系统了解了一个查询语句的执行流程，并介绍了执行过程中涉及的处理模块。相信你还记得，一条查询语句的执行过程一般是经过连接器、分析器、优化器、执行器等功能模块，最后到达存储引擎。

那么，一条更新语句的执行流程又是怎样的呢？

之前你可能经常听DBA同事说，MySQL可以恢复到半个月内任意一秒的状态，惊叹的同时，你是不是心中也会不免会好奇，这是怎样做到的呢？

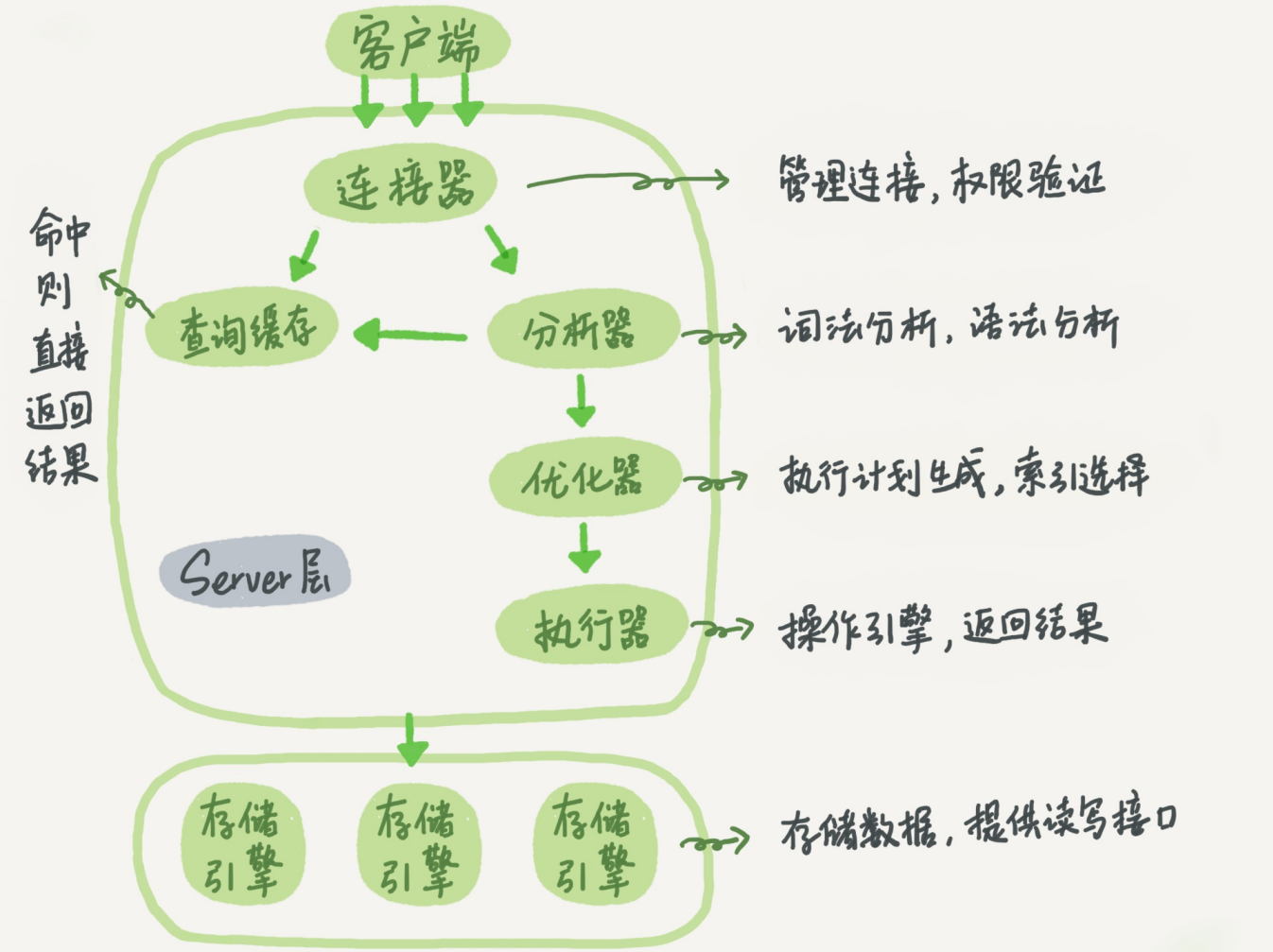
我们还是从一个表的一条更新语句说起，下面是这个表的创建语句，这个表有一个主键ID和一个整型字段c：

```
mysql> create table T(ID int primary key, c int);
```

如果要将ID=2这一行的值加1，SQL语句就会这么写：

```
mysql> update T set c=c+1 where ID=2;
```

前面我有跟你介绍过SQL语句基本的执行链路，这里我再把那张图拿过来，你也可以先简单看看这个图回顾下。首先，可以确定的说，查询语句的那一套流程，更新语句也是同样会走一遍。



MySQL的逻辑架构图

你执行语句前要先连接数据库，这是连接器的工作。

前面我们说过，在一个表上有更新的时候，跟这个表有关的查询缓存会失效，所以这条语句就会把表T上所有缓存结果都清空。这也就是我们一般不建议使用查询缓存的原因。

接下来，分析器会通过词法和语法解析知道这是一条更新语句。优化器决定要使用ID这个索引。然后，执行器负责具体执行，找到这一行，然后更新。

与查询流程不一样的是，更新流程还涉及两个重要的日志模块，它们正是我们今天要讨论的主角：redo log（重做日志）和 binlog（归档日志）。如果接触MySQL，那这两个词肯定是绕不过的，我后面的内容里也会不断地和你强调。不过话说回来，redo log和binlog在设计上有很多有意思的地方，这些设计思路也可以用到你自己的程序里。

重要的日志模块：redo log

不知道你还记不记得《孔乙己》这篇文章，酒店掌柜有一个粉板，专门用来记录客人的赊账记录。如果赊账的人不多，那么他可以把顾客名和账目写在板上。但如果赊账的人多了，粉板总会有记不下的时候，这个时候掌柜一定还有一个专门记录赊账的账本。

如果有人要赊账或者还账的话，掌柜一般有两种做法：

- 一种做法是直接把账本翻出来，把这次赊的账加上去或者扣除掉；
- 另一种做法是先在粉板上记下这次的账，等打烊以后再把账本翻出来核算。

在生意红火柜台很忙时，掌柜一定会选择后者，因为前者操作实在是太麻烦了。首先，你得找到这个人的赊账总额那条记录。你想想，密密麻麻几十页，掌柜要找到那个名字，可能还得带上老花镜慢慢找，找到之后再拿出算盘计算，最后再将结果写回到账本上。

这整个过程想想都麻烦。相比之下，还是先在粉板上记一下方便。你想想，如果掌柜没有粉板的帮助，每次记账都得翻账本，效率是不是低得让人难以忍受？

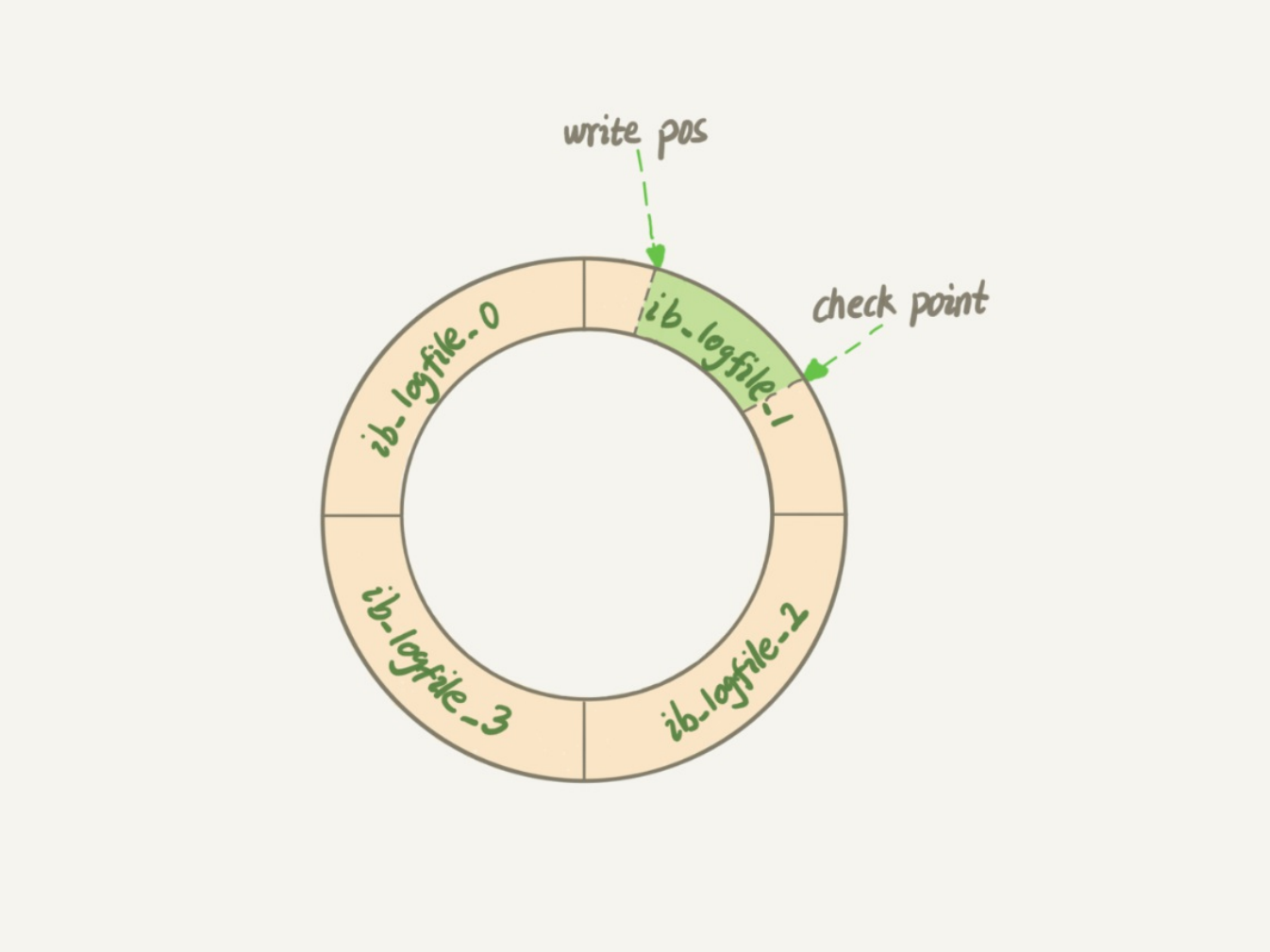
同样，在MySQL里也有这个问题，如果每一次的更新操作都需要写进磁盘，然后磁盘也要找到对应的那条记录，然后再更新，整个过程IO成本、查找成本都很高。为了解决这个问题，MySQL的设计者就用了类似酒店掌柜粉板的思路来提升更新效率。

而粉板和账本配合的整个过程，其实就是MySQL里经常说到的WAL技术，WAL的全称是Write-Ahead Logging，它的关键点就是先写日志，再写磁盘，也就是先写粉板，等不忙的时候再写账本。

具体来说，当有一条记录需要更新的时候，InnoDB引擎就会先把记录写到redo log（粉板）里面，并更新内存，这个时候更新就算完成了。同时，InnoDB引擎会在适当的时候，将这个操作记录更新到磁盘里面，而这个更新往往是在系统比较空闲的时候做，这就像打烊以后掌柜做的事。

如果今天赊账的不多，掌柜可以等打烊后再整理。但如果某天赊账的特别多，粉板写满了，又怎么办呢？这个时候掌柜只好放下手中的活儿，把粉板中的一部分赊账记录更新到账本中，然后把这些记录从粉板上擦掉，为记新账腾出空间。

与此类似，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，那么这块“粉板”总共就可以记录4GB的操作。从头开始写，写到末尾就又回到开头循环写，如下面这个图所示。



write pos是当前记录的位置，一边写一边后移，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“粉板”上还空着的部分，可以用来记录新的操作。如果write pos追上checkpoint，表示“粉板”满了，这时候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**。

要理解crash-safe这个概念，可以想想我们前面赊账记录的例子。只要赊账记录记在了粉板上或写在了账本上，之后即使掌柜忘记了，比如突然停业几天，恢复生意后依然可以通过账本和粉板上的数据明确赊账账目。

重要的日志模块：binlog

前面我们讲过，MySQL整体来看，其实就有两块：一块是Server层，它主要做的是MySQL功能层面的事情；还有一块是引擎层，负责存储相关的具体事宜。上面我们聊到的粉板redo log是InnoDB引擎特有的日志，而Server层也有自己的日志，称为binlog（归档日志）。

我想你肯定会问，为什么会有两份日志呢？

因为最开始MySQL里并没有InnoDB引擎。MySQL自带的引擎是MyISAM，但是MyISAM没有crash-safe的能力，binlog日志只能用于归档。而InnoDB是另一个公司以插件形式引入MySQL的，既然只依靠binlog是没有crash-safe能力的，所以InnoDB使用另外一套日志系统——也就是redo log来实现crash-safe能力。

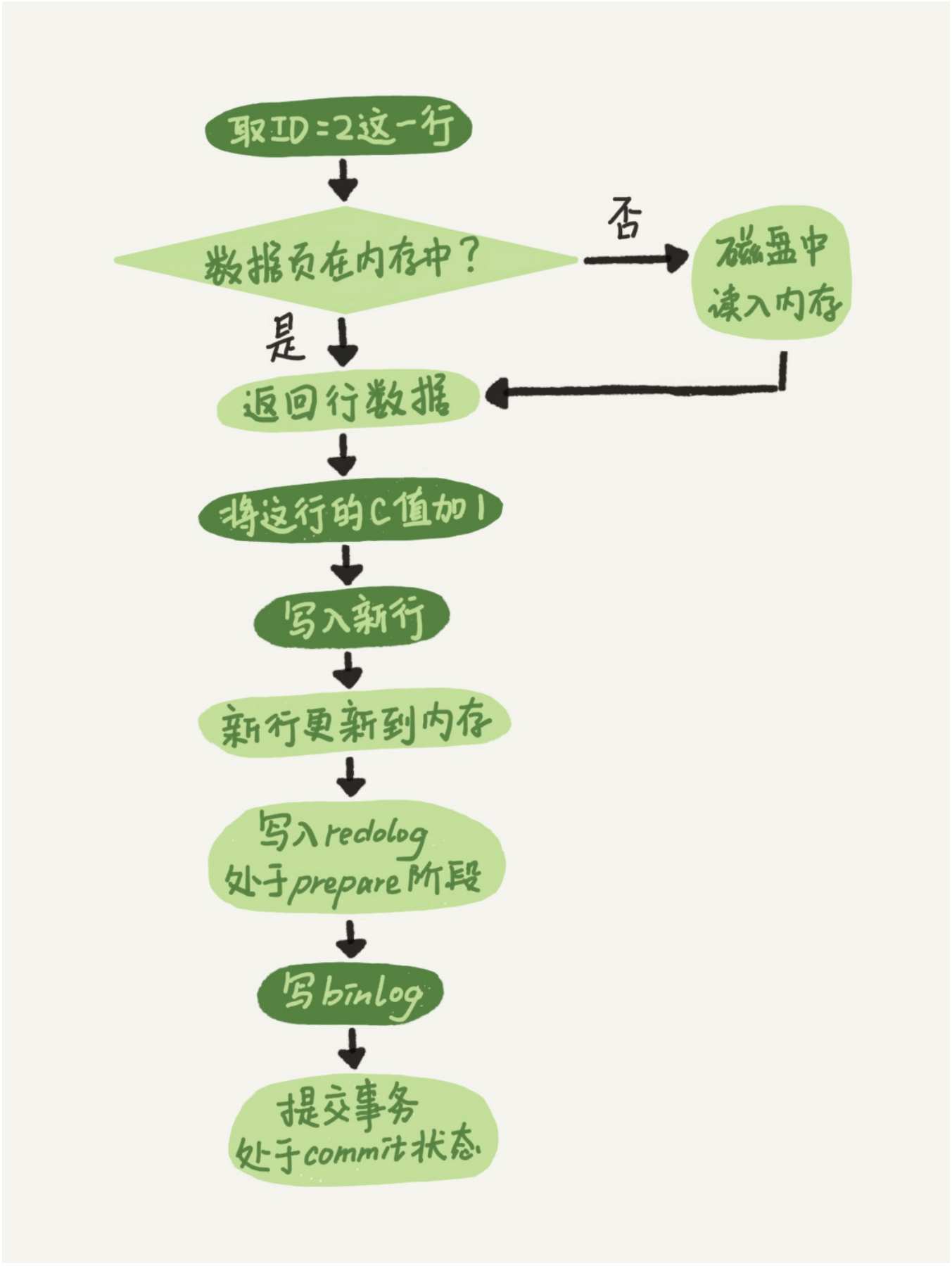
这两种日志有以下三点不同。

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，记录的是“在某个数据页上做了什么修改”；binlog是逻辑日志，记录的是这个语句的原始逻辑，比如“给ID=2这一行的c字段加1 ”。
3. redo log是循环写的，空间固定会用完；binlog是可以追加写入的。“追加写”是指binlog文件写到一定大小后会切换到下一个，并不会覆盖以前的日志。

有了对这两个日志的概念性理解，我们再来看执行器和InnoDB引擎在执行这个简单的update语句时的内部流程。

1. 执行器先找引擎取ID=2这一行。ID是主键，引擎直接用树搜索找到这一行。如果ID=2这一行所在的数据页本来就在内存中，就直接返回给执行器；否则，需要先从磁盘读入内存，然后再返回。
2. 执行器拿到引擎给的行数据，把这个值加上1，比如原来是N，现在就是N+1，得到新的一行数据，再调用引擎接口写入这行新数据。
3. 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，此时redo log处于prepare状态。然后告知执行器执行完成了，随时可以提交事务。
4. 执行器生成这个操作的binlog，并把binlog写入磁盘。
5. 执行器调用引擎的提交事务接口，引擎把刚刚写入的redo log改成提交（commit）状态，更新完成。

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。



update语句执行流程

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

为什么必须有“两阶段提交”呢？这是为了让两份日志之间的逻辑一致。要说明这个问题，我们得从文章开头的那个问题说起：**怎样让数据库恢复到半个月内任意一秒的状态？**

前面我们说过了，binlog会记录所有的逻辑操作，并且是采用“追加写”的形式。如果你的DBA承诺说半个月内可以恢复，那么备份系统中一定会保存最近半个月的所有binlog，同时系统会定期做整库备份。这里的“定期”取决于系统的重要性，可以是一天一备，也可以是一周一备。

当需要恢复到指定的某一秒时，比如某天下午两点发现中午十二点有一次误删表，需要找回数据，那你可以这么做：

- 首先，找到最近的一次全量备份，如果你运气好，可能就是昨天晚上的一个备份，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，重放到中午误删表之前的那个时刻。

这样你的临时库就跟误删之前的线上库一样了，然后你可以把表数据从临时库取出来，按需要恢复到线上库去。

好了，说完了数据恢复过程，我们回来说说，为什么日志需要“两阶段提交”。这里不妨用反证法来进行解释。

由于redo log和binlog是两个独立的逻辑，如果不用两阶段提交，要么就是先写完redo log再写binlog，或者采用反过来的顺序。我们看看这两种方式会有什么问题。

仍然用前面的update语句来做例子。假设当前ID=2的行，字段c的值是0，再假设执行update语句过程中在写完第一个日志后，第二个日志还没有写完期间发生了crash，会出现什么情况呢？

1. **先写redo log后写binlog**。假设在redo log写完，binlog还没有写完的时候，MySQL进程异常重启。由于我们前面说过的，redo log写完之后，系统即使崩溃，仍然能够把数据恢复回来，所以恢复后这一行c的值是1。
   但是由于binlog没写完就crash了，这时候binlog里面就没有记录这个语句。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。
   然后你会发现，如果需要用这个binlog来恢复临时库的话，由于这个语句的binlog丢失，这个临时库就会少了这一次更新，恢复出来的这一行c的值就是0，与原库的值不同。

2. **先写binlog后写redo log**。如果在binlog写完之后crash，由于redo log还没写，崩溃恢复以后这个事务无效，所以这一行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。

你可能会说，这个概率是不是很低，平时也没有什么动不动就需要恢复临时库的场景呀？

其实不是的，不只是误操作后需要用这个过程来恢复数据。当你需要扩容的时候，也就是需要再多搭建一些备库来增加系统的读能力的时候，现在常见的做法也是用全量备份加上应用binlog来实现的，这个“不一致”就会导致你的线上出现主从数据库不一致的情况。

简单说，redo log和binlog都可以用于表示事务的提交状态，而两阶段提交就是让这两个状态保持逻辑上的一致。

小结

今天，我介绍了MySQL里面最重要的两个日志，即物理日志redo log和逻辑日志binlog。

redo log用于保证crash-safe能力。innodb_flush_log_at_trx_commit这个参数设置成1的时候，表示每次事务的redo log都直接持久化到磁盘。这个参数我建议你设置成1，这样可以保证MySQL异常重启之后数据不丢失。

sync_binlog这个参数设置成1的时候，表示每次事务的binlog都持久化到磁盘。这个参数我也建议你设置成1，这样可以保证MySQL异常重启之后binlog不丢失。

我还跟你介绍了与MySQL日志系统密切相关的“两阶段提交”。两阶段提交是跨系统维持数据逻辑一致性时常用的一个方案，即使你不做数据库内核开发，日常开发中也有可能会用到。

文章的最后，我给你留一个思考题吧。前面我说到定期全量备份的周期“取决于系统重要性，有的是一天一备，有的是一周一备”。那么在什么场景下，一天一备会比一周一备更有优势呢？或者说，它影响了这个数据库系统的哪个指标？

你可以把你的思考和观点写在留言区里，我会在下一篇文章的末尾给出我的答案。

感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。

精选留言

super blue cat

288

我可以认为redo log 记录的是这个行在这个页更新之后的状态，binlog 记录的是sql吗？

2018-11-16

作者回复

Redo log不是记录数据页“更新之后的状态”，而是记录这个页 “做了什么改动”。

Binlog有两种模式，statement 格式的话是记sql语句， row格式会记录行的内容，记两条，更新前和更新后都有。

（谢谢你提这个问题，为了不打断文章思路，这个点没在正文写，但是又是很重要的点）

2018-11-16

Jason

239

首先谈一下，学习后的收获

redo是物理的，binlog是逻辑的；现在由于redo是属于InnoDB引擎，所以必须要有binlog，因为你可以使用别的引擎

保证数据库的一致性，必须要保证2份日志一致，使用的2阶段式提交；其实感觉像事务，不是成功就是失败，不能让中间环节出现，也就是一个成功，一个失败

如果有一天mysql只有InnoDB引擎了，有redo来实现复制，那么感觉oracle的DG就诞生了，物理的速度也将远超逻辑的，毕竟只记录了改动向量

binlog几大模式，一般采用row，因为遇到时间，从库可能会出现不一致的情况，但是row更新前后都有，会导致日志变大

最后2个参数，保证事务成功，日志必须落盘，这样，数据库crash后，就不会丢失某个事务的数据了

其次说一下，对问题的理解
备份时间周期的长短，感觉有2个方便
首先，是恢复数据丢失的时间，既然需要恢复，肯定是数据丢失了。如果一天一备份的话，只要找到这天的全备，加入这天某段时间的binlog来恢复，如果一周一备份，假设是周一，而你要恢复的数据是周日某个时间点，那就，需要全备+周一到周日某个时间点的全部binlog用来恢复，时间相比前者需要增加很多；看业务能忍受的程度
其次，是数据库丢失，如果一周一备份的话，需要确保整个一周的binlog都完好无损，否则将无法恢复；而一天一备，只要保证这天的binlog都完好无损；当然这个可以通过校验，或者冗余等技术来实现，相比之下，上面那点更重要

不对的地方，望大神指点

2018-11-16

作者回复

置顶了

你说的物理复制业界有团队在做了

而且官方新版本也把MySQL的表结构都收归InnoDB管理了

2018-11-16

lfn

71

老师,今天MYSQL第二讲中提到binlog和redo log, 我感觉binlog很多余，按理是不是只要redo log就够了?[费解]
您讲的时候说redo log是InnoDB的要求，因为以plugin的形式加入到MySQL中，此时binlog作为Server层的日志已然存在，所以便有了两者共存的现状。但我觉得这并不能解释我们在只用InnoDB引擎的时候还保留Binlog这种设计的原因.

2018-11-16

作者回复

binlog还不能去掉。

一个原因是，redolog只有InnoDB有，别的引擎没有。

另一个原因是，redolog是循环写的，不持久保存，binlog的“归档”这个功能，redolog是不具备的。

2018-11-16

Godruoyi

47

Bin log 用于记录了完整的逻辑记录，所有的逻辑记录在 bin log 里都能找到，所以在备份恢复时，是以 bin log 为基础，通过其记录的完整逻辑操作，备份出一个和原库完整的数据。

在两阶段提交时，若 redo log 写入成功，bin log 写入失败，则后续通过 bin log 恢复时，恢复的数据将会缺失一部分。(如 redo log 执行了 update t set status = 1，此时原库的数据 status

已更新为 1，而 bin log 写入失败，没有记录这一操作，后续备份恢复时，其 status = 0，导致数据不一致）。

若先写入 bin log，当 bin log 写入成功，而 redo log 写入失败时，原库中的 status 仍然是 0，但是当通过 bin log 恢复时，其记录的操作是 set status = 1，也会导致数据不一致。

其核心就是， redo log 记录的，即使异常重启，都会刷新到磁盘，而 bin log 记录的， 则主要用于备份。

我可以这样理解吗？还有就是如何保证 redo log 和 bin log 操作的一致性啊？

2018-11-16

作者回复

几乎全对，除了这个“两阶段提交时，若redo log写入成功，但binlog写入失败...”这句话。

实际上，因为是两阶段提交，这时候redolog只是完成了prepare, 而binlog又失败，那么事务本身会回滚，所以这个库里面status的值是0。

如果通过binlog 恢复出一个库，status值也是0。

这样不算丢失，这样是合理的结果。

两阶段就是保证一致性用的。
你不用担心日志写错，那样就是bug了...

2018-11-16

DanielAnton 24

有个问题请教老师，既然write pos和checkout都是往后推移并循环的，而且当write pos赶上checkout的时候要停下来，将checkout往后推进，那么是不是意味着write pos的位置始终在checkout后面，最多在一起，而这和老师画的图有些出入，不知道我的理解是不是有些错误，请老师指教。

2018-11-16

作者回复

因为是“循环”的，图中这个状态下，write_pos 往前写，写到3号文件末尾，就回到0号继续写，这样你再理解看看“追”的状态。

刚好借你这个问题，说明一下，文中“write pos和checkpoint之间的是'粉板'上还空着的部分，可以用来记录新的操作。”
这句话，说的“空着的部分”，就是write pos 到3号文件末尾，再加上0号文件开头到checkpoint的部分。

2018-11-16

小张 23

老师您好，有一个问题，如果在非常极端的情况下，redo log被写满，而redo log涉及的事务均未提交，此时又有新的事务进来时，就要擦除redo log，这就意味着被修改的的脏页此时要被迫被flush到磁盘了，因为用来保证事务持久性的redo log就要消失了。但如若真的执行了这样的操作，数据就在被commit之前被持久化到磁盘中了。当真的遇到这样的恶劣情况时，mysql会如何处理呢，会直接报错吗？还是有什么应对的方法和策略呢？

2018-11-27

作者回复

，会想到这么细致的场景

这些数据在内存中是无效其他事务读不到的（读到了也放弃），同样的，即使写进磁盘，也没关系，再次读到内存以后，还是原来的逻辑

2018-11-27

哇！怎么这么大个

15

老师您好，我之前是做运维的，通过binlog恢复误操作的数据，但是实际上，我们会后知后觉，误删除一段时间了，才发现误删除，此时，我把之前误删除的binlog导入，再把误删除之后binlog导入，会出现问题，比如主键冲突，而且binlog导数据，不同模式下时间也有不同，但是一般都是row模式，时间还是很久，有没什么方式，时间短且数据一致性强的方式

2018-12-20

作者回复

其实恢复数据只能恢复到误删之前到一刻，

误删之后的，不能只靠binlog来做，因为业务逻辑可能因为误删操作的行为，插入了逻辑错误的语句，

所以之后的，跟业务一起，从业务快速补数据的。只靠binlog补出来的往往不完整

2018-12-20

ricktian

14

redo log的机制看起来和ring buffer一样的；

另外有个和高枕、思雨一样的疑问，如果在重启后，需要通过检查binlog来确认redo log中处于prepare的事务是否需要commit，那是否不需要二阶段提交，直接以binlog的为准，如果binlog中不存在的，就认为是需要回滚的。这个地方，是不是我漏了什么，拉不通。。。 麻烦老师解下疑，多谢～

2018-11-16

作者回复

文章中有提到“binlog没有被用来做崩溃恢复”，

历史上的原因是，这个是一开始就这么设计的，所以不能只依赖binlog。

操作上的原因是，binlog是可以关的，你如果有权限，可以set sql_log_bin=0关掉本线程的binlog日志。 所以只依赖binlog来恢复就靠不住。

@高枕、@思雨 也看下这个讨论

2018-11-16

高枕

320

我再来说下自己的理解 。

1 prepare阶段 2 写binlog 3 commit

当在2之前崩溃时

重启恢复：后发现没有commit，回滚。备份恢复：没有binlog 。

一致

当在3之前崩溃

重启恢复：虽没有commit，但满足prepare和binlog完整，所以重启后会自动commit。备份：有binlog. 一致

2018-11-16

作者回复

，get 完成

2018-11-16

某、人

194

1.首先客户端通过tcp/ip发送一条sql语句到server层的SQL interface

2.SQL interface接到该请求后，先对该条语句进行解析，验证权限是否匹配

3.验证通过以后，分析器会对该语句分析,是否语法有错误等

4.接下来是优化器器生成相应的执行计划，选择最优的执行计划

5.之后会是执行器根据执行计划执行这条语句。在这一步会去open table,如果该table上有MDL，则等待。

如果没有，则加在该表上加短暂的MDL(S)

(如果opend_table太大,表明open_table_cache太小。需要不停的去打开frm文件)

6.进入到引擎层，首先会去innodb_buffer_pool里的data dictionary(元数据信息)得到表信息

7.通过元数据信息,去lock info里查出是否会有相关的锁信息，并把这条update语句需要的锁信息写入到lock info里(锁这里还有待补充)

8.然后涉及到的老数据通过快照的方式存储到innodb_buffer_pool里的undo page里,并且记录undo log修改的redo

(如果data page里有就直接载入到undo page里，如果没有，则需要去磁盘里取出相应page的数据，载入到undo page里)

9.在innodb_buffer_pool的data page做update操作。并把操作的物理数据页修改记录到redo log buffer里

由于update这个事务会涉及到多个页面的修改，所以redo log buffer里会记录多条页面的修改信息。

因为group commit的原因，这次事务所产生的redo log buffer可能会跟随其它事务一同flush并且sync到磁盘上

10.同时修改的信息，会按照event的格式,记录到binlog_cache中。(这里注意binlog_cache_size是transaction级别的,不是session级别的参数,

一旦commit之后，dump线程会从binlog_cache里把event主动发送给slave的I/O线程)

11.之后把这条sql,需要在二级索引上做的修改，写入到change buffer page，等到下次有其他sql

需要读取该二级索引时，再去与二级索引做merge
(随机I/O变为顺序I/O,但是由于现在的磁盘都是SSD,所以对于寻址来说,随机I/O和顺序I/O差距不大)
12.此时update语句已经完成，需要commit或者rollback。这里讨论commit的情况，并且双1
13.commit操作，由于存储引擎层与server层之间采用的是内部XA(保证两个事务的一致性,这里主要保证redo log和binlog的原子性),
所以提交分为prepare阶段与commit阶段
14.prepare阶段,将事务的xid写入，将binlog_cache里的进行flush以及sync操作(大事务的话这步非常耗时)
15.commit阶段，由于之前该事务产生的redo log已经sync到磁盘了。所以这步只是在redo log里标记commit
16.当binlog和redo log都已经落盘以后，如果触发了刷新脏页的操作，先把该脏页复制到double write buffer里，把doublewrite buffer里的刷新到共享表空间，然后才是通过page cleaner线程把脏页写入到磁盘中
老师，你看我的步骤中有什么问题嘛？我感觉第6步那里有点问题,因为第5步已经去open table了，第6步还有没有必要去buffer里查找元数据呢?这元数据是表示的系统的元数据嘛,还是所有表的？谢谢老师指正

2018-11-16

作者回复

其实在实现上5是调用了6的过程了的，所以是一回事。MySQL server 层和InnoDB层都保存了表结构，所以有书上描述时会拆开说。

这个描述很详细，同时还有点到我们后面要讲的内通（编辑快来，有人来砸场子啦）

2018-11-16

justd

63

一个完整的交易过程：账本记上 卖一瓶可乐（redo log为 prepare状态），然后收钱放入钱箱（bin log记录）然后回过头在账本上打个勾（redo log置为commit）表示一笔交易结束。
如果收钱时交易被打断，回过头来整理此次交易，发现只有记账没有收钱，则交易失败，删掉账本上的记录（回滚）；如果收了钱后被终止，然后回过头发现账本有记录（prepare）而且钱箱有本次收入（bin log），则继续完善账本（commit），本次交易有效。

2018-11-17

文@雨路

55

老师，我想问下如果提交事务的时候正好重启那么redo log和binlog会怎么处理？此时redo log处于prepare阶段，如果不接受这条log，但是binlog已经接受，还是说binlog会去检查redo log的状态，状态为prepare的不会恢复？

2018-11-16

作者回复

好问题表示中间那段你都听明白了

Binlog如果已经接受，那么redolog是prepare, binlog已经完整了对吧，这时候崩溃恢复过程会认可这个事务，提交掉。（你可以分析下这种情况下，是否符合我们要达到的“用binlog恢复的库跟原库逻辑相同”这个要求）

2018-11-16

Mao

48

老师，您好。您说MySQL 使用WAL，先写日志再写磁盘。请教一个问题，

执行一条Update 语句后，马上又执行一条 select * from table limit 10。

如果刚刚update的记录，还没持久化到磁盘中，而偏偏这个时候的查询条件，又包含了刚刚update的记录。

那么这个时候，是从日志中获取刚刚update的最新结果，还是说，先把日志中的记录先写磁盘，再返回最新结果？

2018-11-19

cyberbit

40

我理解备份就是救命药加后悔药，灾难发生的时候备份能救命，出现错误的时候备份能后悔。事情都有两面性，没有谁比谁好，只有谁比谁合适，完全看业务情况和需求而定。一天一备恢复时间更短，binlog更少，救命时候更快，但是后悔时间更短，而一周一备正好相反。我自己的备份策略是设置一个16小时延迟复制的从库，充当后悔药，恢复时间也较快。再两天一个全备库和binlog，作为救命药,最后时刻用。这样就比较兼顾了。

2018-11-16

作者回复

你是有故事的人

2018-11-16

WL

37

为该讲的内容总结了几个问题, 大家复习的时候可以先尝试回答这些问题检查自己的掌握程度:

1. redo log的概念是什么? 为什么会存在.
2. 什么是WAL(write-ahead log)机制, 好处是什么.
3. redo log 为什么可以保证crash safe机制.
4. binlog的概念是什么, 起到什么作用, 可以做crash safe吗?
5. binlog和redolog的不同点有哪些?
6. 物理一致性和逻辑一直性各应该怎么理解?
7. 执行器和innoDB在执行update语句时候的流程是什么样的?
8. 如果数据库误操作, 如何执行数据恢复?
9. 什么是两阶段提交, 为什么需要两阶段提交, 两阶段提交怎么保证数据库中两份日志间的逻辑一致性(什么叫逻辑一致性)?
10. 如果不是两阶段提交, 先写redo log和先写bin log两种情况各会遇到什么问题?

2018-11-24

文@雨路

30

老师，也就是说状态恢复的过程会去同时检查redo log和binlog？不然怎么能确定一个prepare的redo log已经写好了binlog，因为不检查的话不能确定到底是写好了binlog之后奔溃的还是写之前奔溃的，也就不能确定这个prepare log的合法性。如果检查的话那么不用两阶段提交好像也没什么问题，无论先写哪个日志都可以。

2018-11-16

lionetes

27

昨天上午 恢复别人误操作配置表数据，幸好有xtarbackup凌晨的全量备份，只提取了改表的ibd文件，然后在本地 做了 一个一样的空表，释放该表空间，加载 提取后的ibd文件，提取昨天零晨到九点的binlog文件 筛选改表这个时段的操作记录 增量更新到本地导出csv 导入线上 。binlog太tm重要了

2018-11-17

作者回复

你选择了最优路径

2018-11-18

小美

23

老师，我这想请教两个问题：
1.写redo日志也是写io（我理解也是外部存储）。同样耗费性能。怎么能做到优化呢
2.数据库只有redo commit 之后才会真正提交到数据库吗

2018-11-16

作者回复

1. Redolog是顺序写，并且可以组提交，还有别的一些优化，收益最大是是这两个因素；

2.是这样，正常执行是要commit 才算完，但是崩溃恢复过程的话，可以接受“redolog prepare并且binlog完整” 的情况

2018-11-16

Lugyedo

17

redo log和bin log怎么对应

2018-11-16

作者回复

事务ID （比较细节，就没在正文里写了

2018-11-16

黄金的太阳

17

请教老师，redo log是为了快速响应SQL充当了粉板，这里有两个疑问
1. redo log本身也是文件，记录文件的过程其实也是写磁盘，那和文中提到的离线写磁盘操作有何区别？
2.响应一次SQL我理解是要同时操作两个日志文件？也就是写磁盘两次？

2018-11-16

作者回复

你的理解是对的。

1. 写redo log是顺序写，不用去“找位置”，而更新数据需要找位置

2. 其实是3次（redolog两次 binlog 1次）。不过在并发更新的时候会合并写

2018-11-16