15 | 答疑文章（一）：日志和索引相关问题

**日志相关问题**

两阶段提交，方便后面的学习

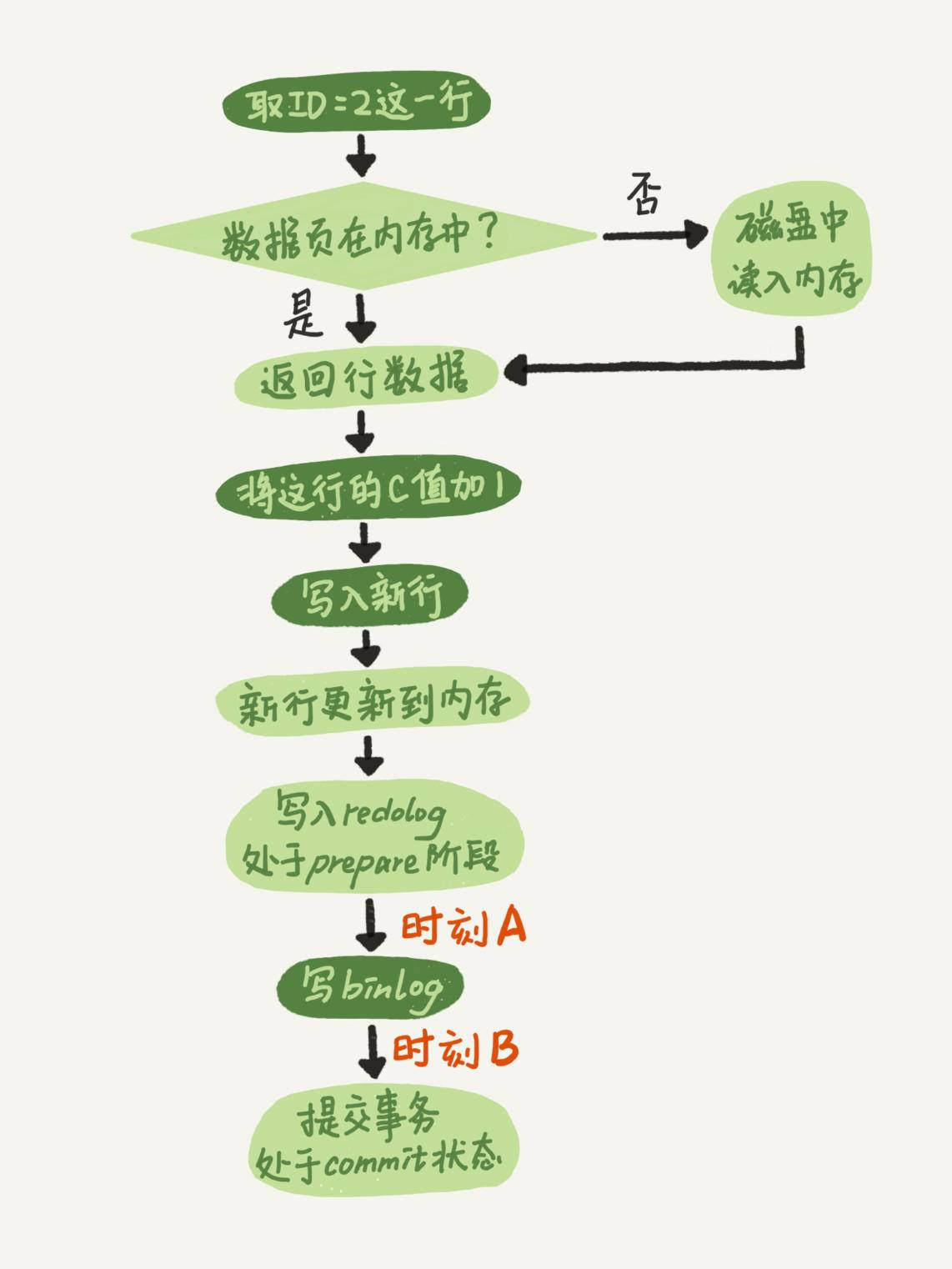


图 1 两阶段提交示意图

这个图不是一个update语句的执行流程吗？怎么还会调用commit语句呢？

两个**commit概念**混淆

1.“commit语句”，是指MySQL语法中，用于提交一个事务的命令，一般跟begin/start transaction配对使用

2.而图中的用到这个“commit步骤”，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，这个事务提交完成了

3.“commit语句”执行的时候，会包含“commit步骤”。

这个例子里面，没有显示的开启事务，因此这个update语句自己就是一个事务，在执行完成后提交事务时，就会用到这个“commit步骤”。

而例子里，没有显式的开启事务，因此这个update语句自己就是一个事务，在执行完后提交事务，就会用到这个“commit步骤”。

分析一下**在两阶段提交的不同时刻，MySQL异常重启会出现什么现象**

如果在图时刻A的地方，也就是写入redo log处于prepare阶段之后，写binlog之前，发生了崩溃（crash）,由于此时binlog还没写，redo log 也还没有提交，所以崩溃恢复时，这个事务会回滚，此时，binlog还没有写，所以也不会传到备库。

时刻B,也就是binlog写完了，redo log还没有commit前发生crash,那崩溃恢复的时候MySQL会怎么处理。

崩溃恢复时的判断规则

1. 如果redo log里面的事务是完整的，也就是已经有了commit标识，则直接跳
2. 如果redo log里面的事务只有完整的prepare,则判断对应的事务binlog是否存在并完整
3. 如果是，则提交事务
4. 否则，回滚事务

时刻B发生crash对应的是2a的情况，崩溃恢复过程中事务会被提交

**追问：MySQL怎么知道binlog是完整的？**

回答：一个事务的binlog是有完整格式的；

·statement 格式的binlog,最后会有Commit

·row格式的binlog，最后会有一个XID event.

另外，MySQL 5.6.2版本以后，还引入了binlog-checksum参数，用来验证binlog内容的正确性，对于binlog日志由于磁盘原因，可能会在日志中间出错的情况，MySQL可以通过校验checksum的结果来发现，所以，MySQL还是有办法验证事务binlog的完整性的。

**追问2：redo log和binlog是怎么关联起来的？**

**回答：**它们有一个共同的数据字段，叫XID,崩溃恢复时，会按顺序扫描redo log

·如果碰到既有prepare，又有commit的redo log，就直接提交

·如果碰到只有prepare,而没有commit的redo log,就拿XID去binlog找对应的事务

**追问3：处于prepare阶段的redo log加上完整的binlog，重启就能恢复，MySQL为什么要**这么设计？

回答：其实，这个问题还是在反证法说到的数据与备份的一致性有关，此时B,也就是binlog写完整以后MySQL发生崩溃，此时binlog已经写入了，之后就会从库（或者用这个binlog恢复出来的库）使用，所以，主库上也要提交这个事务。

**追问4：如果这样的话，为什么还有两阶段提交呢？干脆先redo log写完，再写binlog。崩溃恢复时，必须得两个日志都完整才可以，是不是一样的逻辑？**

回答：其实，两阶段提交时经典的分布式系统问题，并不是MySQL独有的。

如果必须要举一个场景，来说明这么做的必要性的话，那就是事务的持久性问题

对于InnoDB引擎来说，如果redo log提交完成了，事务就不能回滚了（如果这还允许回滚，就可能覆盖掉别的事务的更新）。而如果redo log直接提交，然后binlog写入的时候失败，InnoDB又回滚不了，数据和redo log日志又不一致了。

两阶段提交就是为了给所有人一个机会，当每个人都说“ok”时，在一起提交

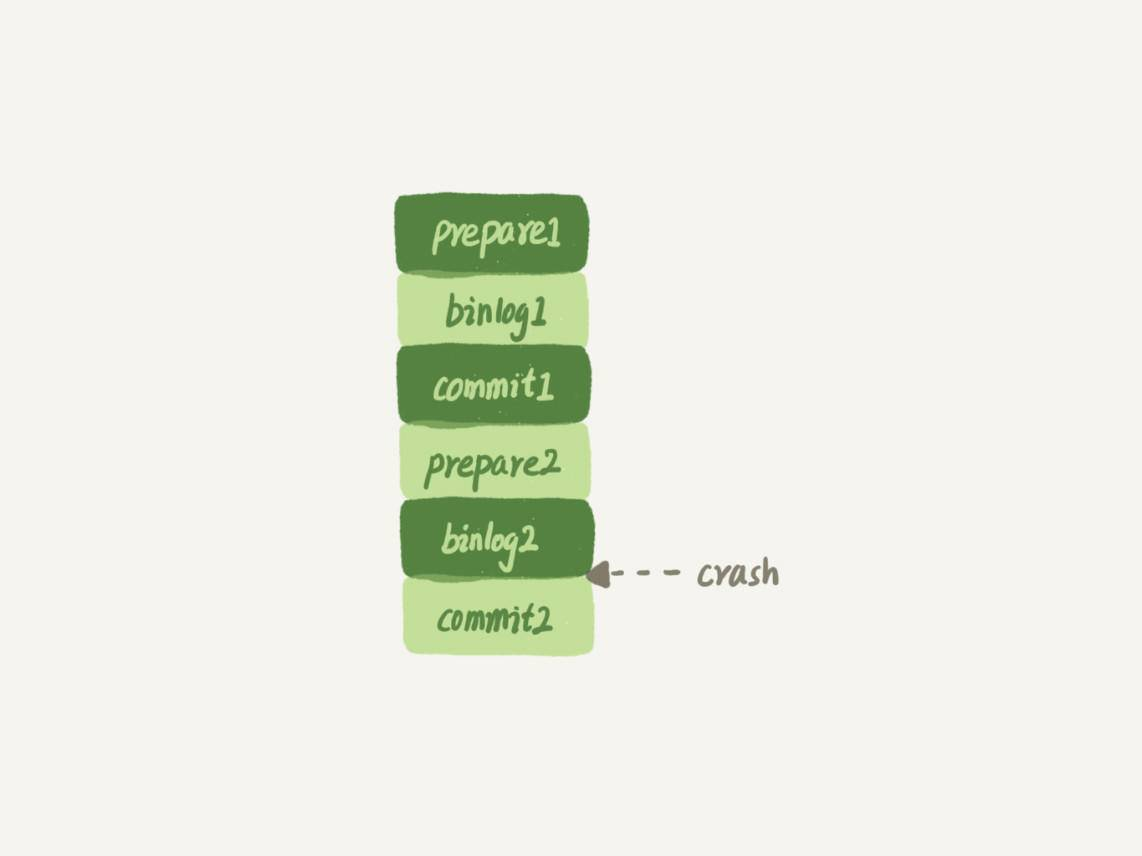
**追问5：不引入两个日志，也就没有两阶段提交的必要了，只用binlog来支持崩溃恢复，又能支持归档，不就可以了吗？**

回答：只保留binlog，然后可以把提交流程改成这样：->数据更新大内存->写binlog-提交事务,是不是也可以体统崩溃恢复的能力？

答案是不可以

如果是历史原因的话，那就是InnoDB并不是MySql的原生引擎，MySQL的原生引擎是MyISAM，设计之初就没有支持崩溃恢复

而如果说实现上的原因，就有很多了，就按照问题中的说：只用binlog来实现崩溃恢复的流程，如图没有redo log了。



**图 2** 只用binlog支持崩溃恢复

这样的流程下，binlog还是不支持崩溃恢复的：binlog没有能力恢复“数据页”

如果在图中标的位置，也就是binlog2写完了，但是整个事务还没有commit时，MySQL发生了crash。

重启后，引擎内部事务2会回滚，然后应用binlog2可以补回来；但是对于事务1来说，系统已经提交完成了，不会再应用一次binlog1.

但是，InnoDB引擎使用的WAL技术，执行事务时，写完内存和日志，事务就算完成后。如果之后崩溃了，要依赖于日志来恢复数据页。

也就是说在图中这个位置发生崩溃的话，事务1也是可能丢失了的，而是数据页级的丢失，此时，binlog里面并没有记录数据页的更新细节，是补不回来的。

如果说，优化一下binlog的内容，让它来记录数据页的更新可以吗？但这其实就是又做了一个redo log出来。

所以，至少现在的binlog能力，还不能支持崩溃恢复。

**追问6：那能不能反过来，只用redo log,不要binlog？**

回答：如果只能从崩溃恢复的角度来讲是可以的，可以把binlog关掉，这样就没有两阶段提交了，但系统依然是crash-safe的。

一个是归档，redo log是循环写的，写到末尾是要回到开头继续写的，这样历史日志没法保留，redo log也就起不到归档的作用了。

一个是MySQL系统依赖于binlog,binlog作为MySQL一开始就有的功能，被用在了很多地方，其中，MySQL系统高可用的基础，就是binlog复制。

**追问7：redo log一般设置多大？**

回答：redo log太小的话，会导致很快就被写满，然后不得不强行刷redo log,这样WAL机制的能力就发挥不出来。

所以，如果是现在常见的几个TB的磁盘的话，直接redo log设置为4个，每隔文件1GB

**追问8：正常运行中的实例，数据写入后的最终落盘 是从redo log更新过来的还是从buffer pool更新过过来的？**

回答：这个涉及到了redo log里面到底是什么的问题

实际上，redo log并没有记录数据页的完整数据，所以它并没有能力自己去更新磁盘数据页，也就不存在“最终落盘，是由redo log更新过去”的情况。

1. 如果是正常运行的实例的话，数据页被修改后，跟磁盘的数据不一致，称为“脏页”，最终落盘，就是把内存中的数据页写盘，这个过程，甚至与redo log毫无关系
2. 在崩溃恢复场景中，InnoDB如果判断到一个数据页可能在崩溃恢复的时候丢失了更新，就会将它读到内存，然后让redo log更新内存内容，更新完成后，内存页变成脏页，就回到了第一种情况的状态。

追问9：redo log buffer是什么？是先修改内存，还是先写redo log文件？

回答：这两个问题可以一起回答

在一个事务更新过程中，日志要写多次的，比如下面的事务

begin;

insert into t1 ...

insert into t2 ...

commit;

这个事务要往两个表插入记录，插入数据的过程中，生成的日志得先保存起来，但又不能在还没commit时就直接写到redo log文件里。

所以，redo log buffer就是一块内存，用来先存redo log日志的，也即是说，在执行第一个insert时，数据的内存被修改了，redo log buffer也写入了日志

但是，真正把日志写到redo log文件（文件名是ib\_logfiel+数字）,是在执行commit语句的时候做的。