23 | MySQL是怎么保证数据不丢的

WAL机制（回顾，2,9，12,15篇文中的相关内容），得到的结论是：只要redo log和binlog保证持久化磁盘，就能保证MySQL异常重启后，数据可以恢复

**binlog的写入机制**

事务的执行过程中，先把日志写到binlog cache,事务提交的时候，在把binlog cache写入到binlog文件中。

一个事务的binlog是不能被拆开的，因此不论这个事务多大，也要确保一次性写入，这涉及到了binlog cache的保存问题。

系统给binlog cache分配了一块内存，每个线程一个，参数binlog\_cache\_size用于控制单个线程内binlog cache所占内存的大小，如果超过了这个是参数规定的大小，就要暂存到磁盘。

事务提交的时候，执行器把binlog cache里的完整事务写入到binlog中，并清空binlog

Cache，状态如图1所示

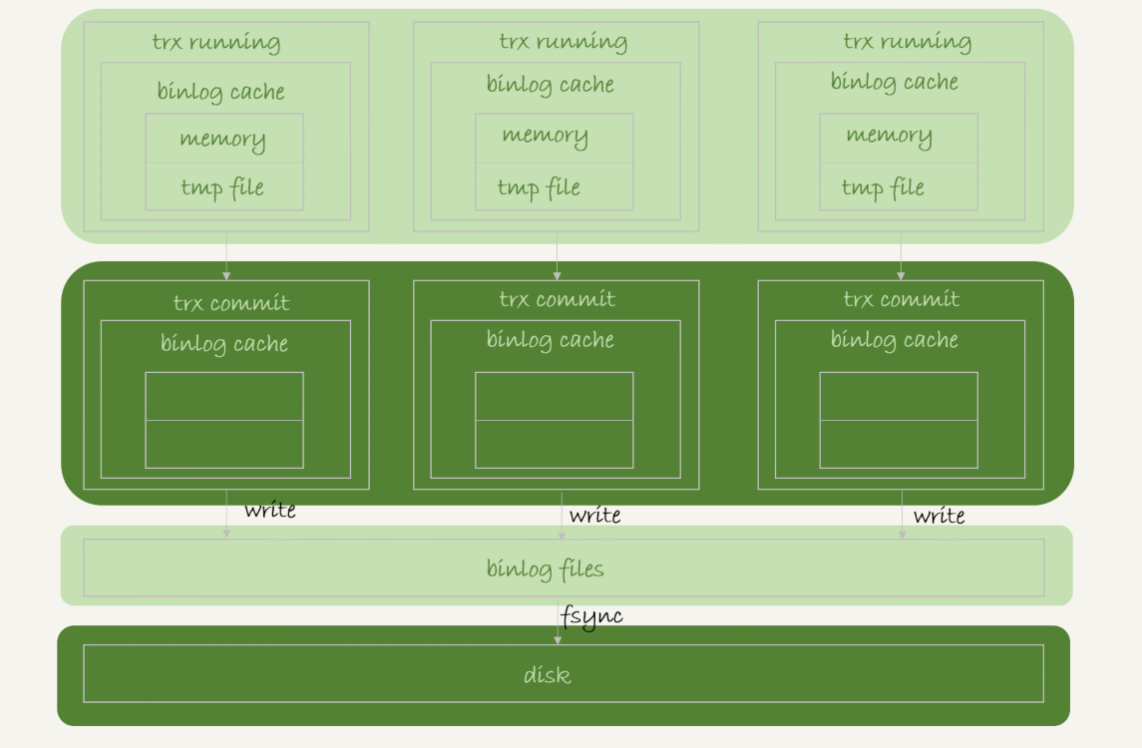


图 1 binlog 写盘状态

每个线程有自己的binlog cache ,但是共用同一个binlog文件

·图中的write，指的是把日志写入到文件系统的page cache,并没有把数据持久化到磁盘，所以速度更快

·图中fsync,才是将数据持久化到磁盘的操作，一般情况下，我们认为fsync才占磁盘的IOPS

Write和fsync的时机，有参数sync\_binlog控制的：

1. sync\_binlog=0的时候，表示每次提交事务都只write，不fsync,
2. Sync-binlog=1的时候，表示每次提交事务都会执行fsync,
3. Sync\_binlog=N(N>1)的时候，表示每次提交事务都write，但积累N个事务后才fsync

因此，出现IO瓶颈的场景里，将sync\_binlog设置成一个比较大的值，可以提升性能。在实际的业务场景中，考虑到丢失日志量的可控性，一般不建议将这参数设成0，比较常见的是将是设置为100-1000中的某个值

但是将sync设置为N，对一个的风险是：如果主机发生异常重启，会丢失最近N个事务的binlog日志。

**Redo log的写入机制**

专栏 15篇，介绍了redo\_log\_buffer,事务在执行过程中，生成的redo log是要先写到redo log buffer的。

Redo log buffer的内容**不需要每次生成后都直接持久化到磁盘**

如果事务执行期间MySQL发生异常重启，这部分日志就丢了，由于事务并没有提交，所以这是日志丢了也不会有损。

事务没有提交的时候，redo log buffer中的部分日志有没有可能被持久化到磁盘呢？

三个状态对应图2的三个颜色块



图 2 MySQL redo log 存储状态

三种状态分别是：

1. 存在redo log buffer中，物理上是MySQL进程内存中，就是图中红色部分
2. 写到磁盘（write）,但是没有持久化（fsync）,物理上是在文件系统的page cache里面，也就是图中黄色的部分
3. 持久化到磁盘，对应的是hard\_disk,也就是图中绿色的部分

日志写到redo log buffer是很快的，write到page cache也差不多，但是持久化到磁盘的速度就慢了。

为了控制redo log 的写入策略，InnoDB提供了innodb\_flush\_log\_at\_trx\_commit参数，3中可能取值

1. 设置为0，表示每次事务提交时都只把redo log留在redo log buffer中
2. 设置为1，表示每次事务提交时都将redo log直接持久化到磁盘
3. 设置为2，表示每次事务提交时都只是把redo log写到page cache

InnoDB有一个后台线程，每隔1秒，会把redo log buffer中的日志，调用write写到文件系统page cache,然后调用fsync持久化到磁盘。

**注意，事务执行中间过程的redo log也是直接写在redo log buffer中，这些redo log也会被后台线程一起持久化大磁盘，**也就是说，一个没有提交事务的redo log,也可能已经持久化到磁盘的。

实际上，除了后台线程每秒一次的轮询操作外，还有两种场景会让一个没有提交的事务的redo log写入到磁盘

1. **一种是,redo log buffer占用的空间即将达到innodb\_log\_buffer\_size一般的时候，后台线程会主动刷盘，**注意，由于这个事务比没有提交，所以这个写盘动作只是write,而没有调用fsync,也就是只留在了文件系统的page cache.
2. **另一种是，并行的事务提交的时候，顺带将这个事务的redo log buffer持久化到磁盘。**假设一个事务A执行到一半，已经写了一些redo log到buffer中，这个时候有另外一个线程的事务B提交，如果Innodb\_flush\_log\_at\_trx\_commit设置是1，那么按这参数的逻辑，事务B要把redo log buffer里的日志全部持久化到磁盘，此时，就会带上事务A在redo log buffer里的日志一起持久化到磁盘。

注意，介绍过两阶段提交，时序上redo log先prepare，再binlog,最后再把redo log commit.

如果把innodb\_flush\_log\_at\_trx\_commit设置为1，那么redo log在prepare阶段就要持久化一次，因为有一个崩溃恢复逻辑是要依赖于prepare的redo log，再加上binlog来恢复的。

（恢复15篇）

每秒一次后台轮询刷盘，再加上崩溃恢复这个逻辑，InnoDB就认为redo log在commit的时候就不需要fsync了，只会write到文件系统的page cache中就够了。

通常数MySQL的双“1”配置，指的就是sync\_binlog和innodb\_flush\_log\_at\_trx\_commit都设置为1，就是一个事务完整提交前，需要等待两次刷盘，一次是redo log(prepare阶段)，

一次是binlog.

此时，有个疑问，从MySQL看到TPS是每秒2万的话，每秒就会写4万次磁盘。但是用工具测试出来，磁盘能力也就两万左右，怎么是实现2万的TPS?

**组提交机制（group commit）**

日志逻辑序列号（log sequence number,LSN）的概念，LSN是单词递增的，用来对应redo log的一个个写入点，每次写入长度为length的redo log,LSN的值就会加上length.

LSN也会写到InnoDB的数据页中，来确保数据页不会被多次执行重复的redo log.关于LSN和redo log 、checkpoint的关系

如图3 ，是三个并发事务（trx1,trx2,trx3 ）在prepare阶段，都写完redo log buffer，持久化到磁盘的过程，对应的LSN分别是50，120和160

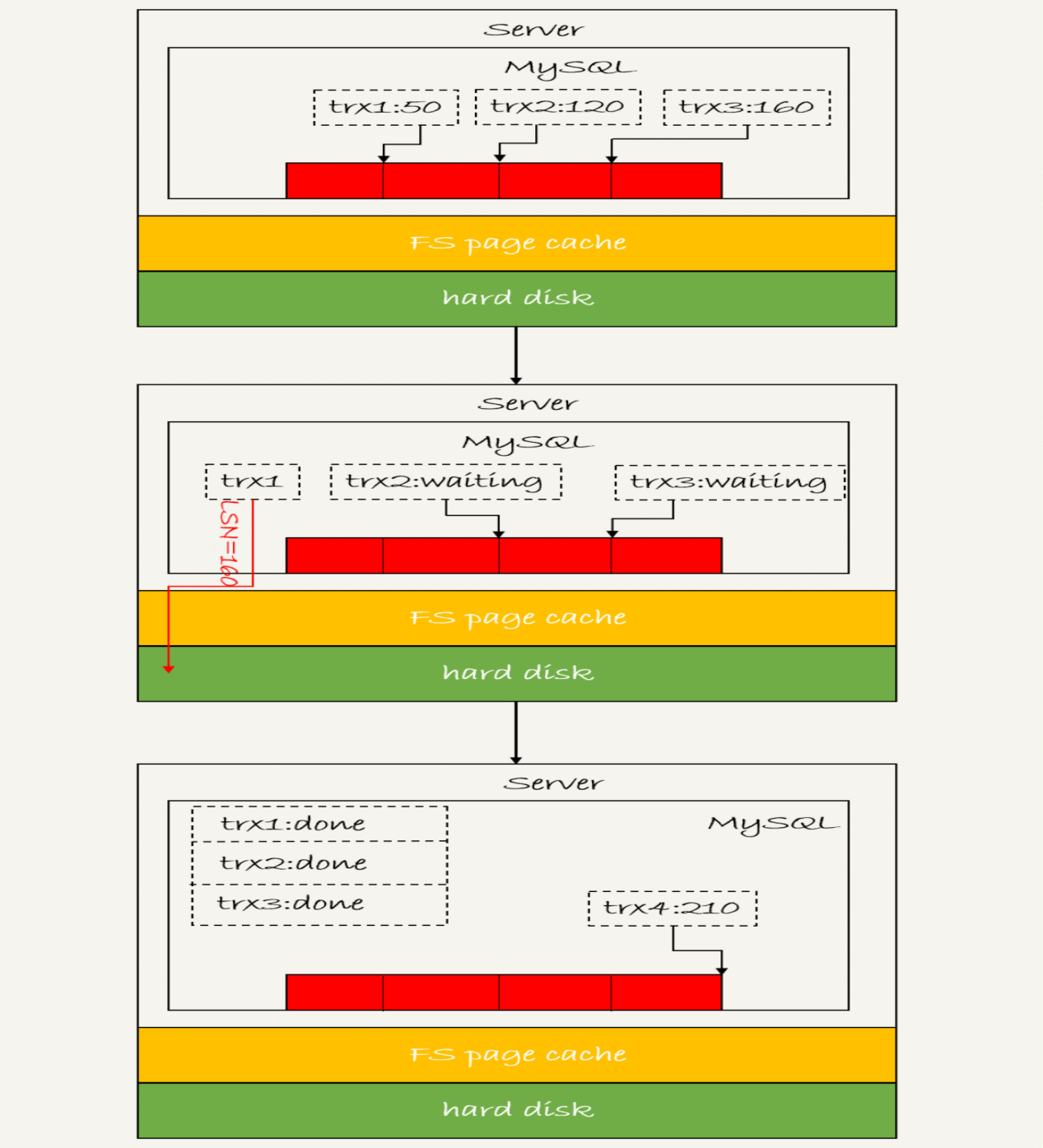


图3 redo log组提交

1. trx1是第一个到达的，会被选为这组的leader;
2. 等trx1要开始写盘的时候，这个组里面已经有三个事务了，这时候LSN也变成了160
3. trx1去写盘的时候，带的就是LSN=160,因此等trx1返回时，所有LSN小于等于160的redo log ,都已经被持久化到磁盘
4. 这时候trx2和trx3就可以直接返回了。

所以，一次组提交里面，组成员越多，节约磁盘IOPS的效果越好，但如果只有单线程压测，就是一个事务对应一次持久化操作了

在并发更新场景下，第一个事务写完redo log buffer以后，接下来这个fsync越晚调用，组员可能越多，节约IOPS的效果就越好

为了让一次fsync带的组员越多，MySQL有个有趣的优化：拖时间，在介绍两阶段提交的时候

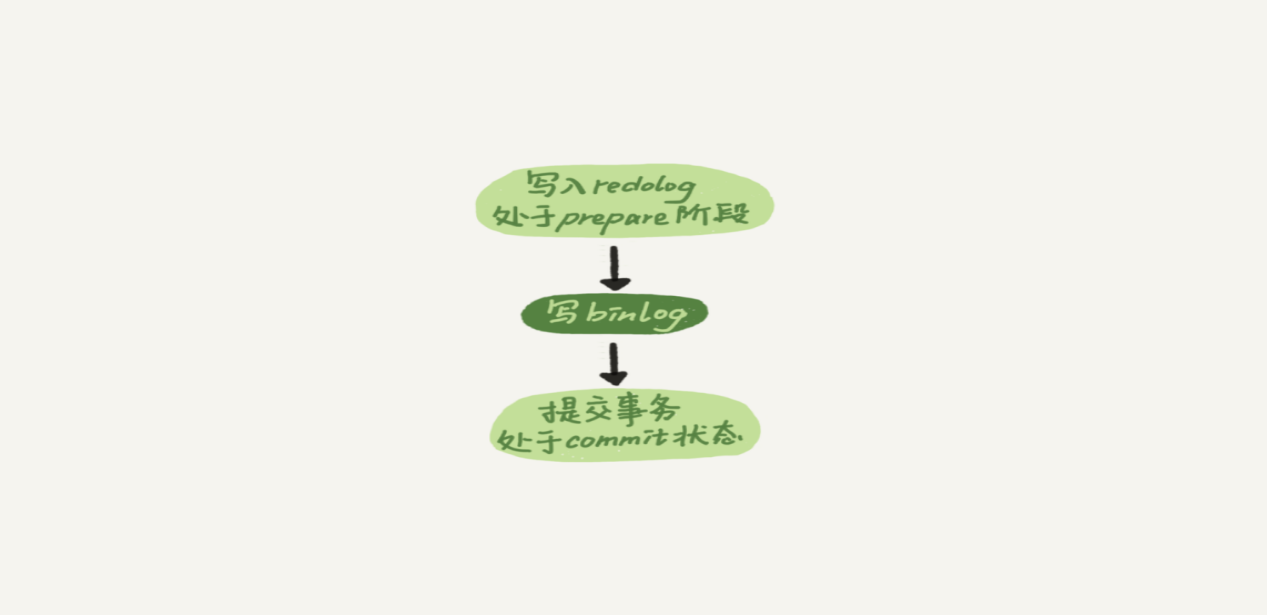


图4 两阶段提交

1. 先把binlog从binlog cache中写到磁盘上的binlog文件
2. 调用fsync持久化

MySQL为了让组提交的效果更好，把redo log做fsync的时间拖到了步骤1之后，也就是上面的图变成了这样

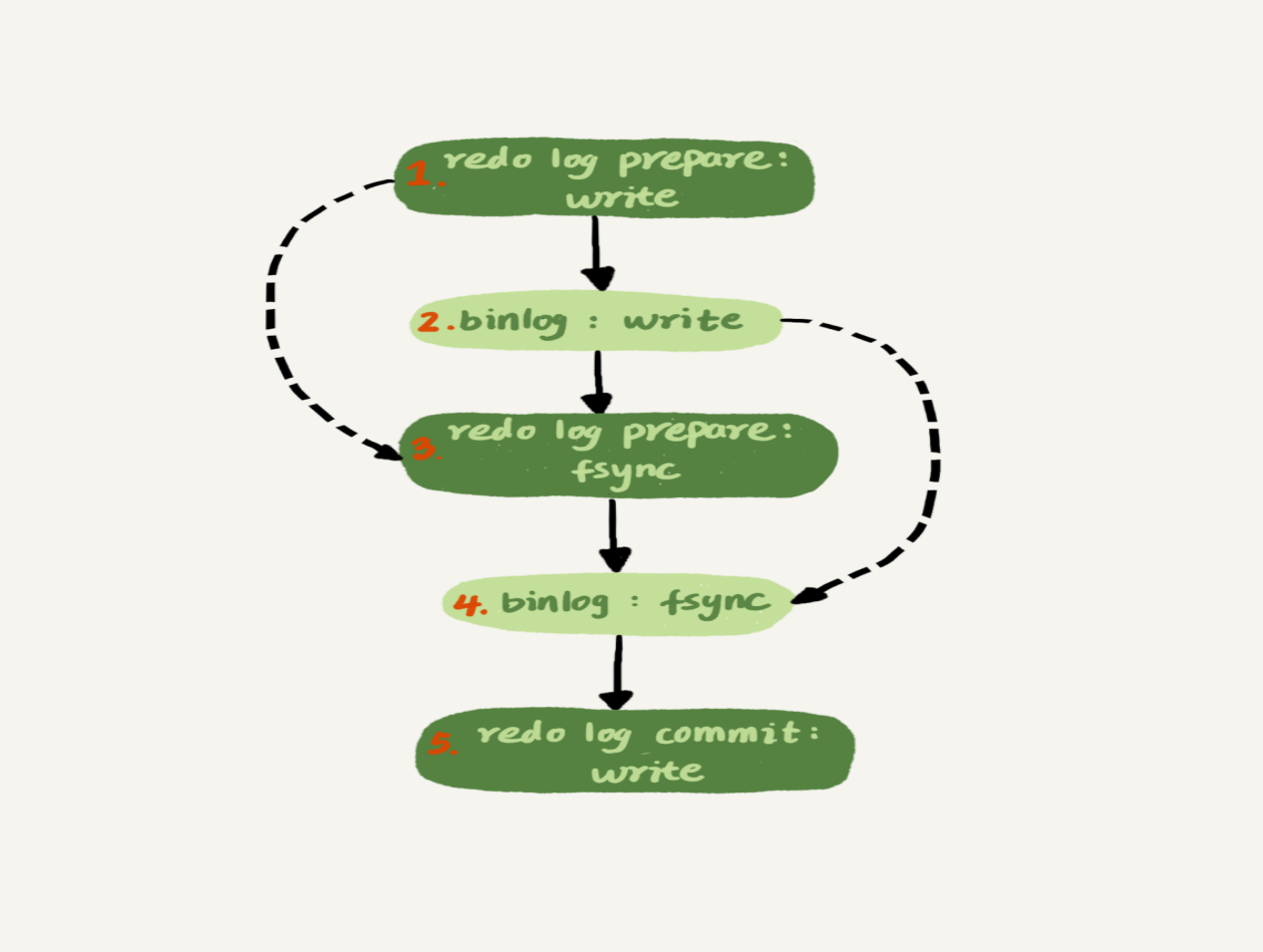


图 5 两阶段提交细化

这么一来，binlog可以组提交了，在执行图5中第4步把binlog fsync到磁盘时，如果有多个事务的binlog已经写完，也是一起持久化的，这样可以减少IOPS的消耗

通常情况下第3步执行的很快，索引binlog的write和fsync间的间隔时间短，导致集合到一起持久化的binlog比较少，因此binlog的组提交的效果通常不如redo log的效果那么好

如果想提升binlog组提交的效果，可以通过设置binlog\_gourp\_commit\_sync\_delay和binlog\_group\_commit\_sync\_no\_delay\_count来实现

1. binlog\_group\_commit\_sync\_delay参数，表示延迟多少微妙后才调用fsync
2. Binlog\_group\_commit\_sync\_no\_delay\_count参数，表示积累多少次以后调用fsync

这两个条件是或的关系，也就是只要有一个满足条件就会调用fsync

所以，当binlog\_group\_commit\_sync\_delay设置为0的时候

Binlog\_group\_commit\_sync\_no\_delay\_count也无效了。

WAL机制主要得益于两个方面

1. redo log和binlog都是顺序写，磁盘的顺序写比随机写速度快
2. 组提交机制，可以大幅度降低磁盘的IOPS消耗

**如果MySQL现在出现性能瓶颈，而且在瓶颈IO上，可以考虑那些方法来提升呢？**

可以考虑三种方法

1. 设置binlog\_group\_commit\_sync\_delay和binlog\_group\_commit\_sync\_no\_delay\_count参数，减少binlog的写盘次数，这个方法是基于“额外的故意等待”来实现的，因此可能会增加语句的响应时间，但没有丢失数据的风险
2. 将sync\_binlog设置为大于1的值（比较常见是100-1000），这样做的风险是，主机掉电是会丢binlog日志
3. 将innodb\_flush\_log\_at\_trx\_commit设置为2，这样做的风险是，主机掉电的时候会丢数据

不建议把innodb\_flush\_log\_at\_trx\_commit设置为0，因为把这个参数设置为成0，表示redo log只存在内存中，这样的话MySQL本身异常重启也会丢数据，风险太大，而redo log写到文件系统的page cache的数据页很快，所以将这个参数设置成2跟设置为0其实性能差不多，但这样做MySQL异常重启时就不会丢数据了，相比之下风险会更小。

**问题1** 执行一个update语句后，再去执行hexdump命令直接查看ibd文件内容，为什么没有看到数据的改变呢？

**回答**：这可能是因为WAL机制的原因，update语句执行完后，InnoDB只保证了写完了redo log,内存、可能还没来得急将数据写到磁盘

**问题2** 为什么binlog cache是每个线程自由维护的，而redo log buffer是全局公用的？

**回答**：MySQL这么设计的主要原因是：binlog是不能被“打断的”，一个事务的binlog必须连续写，因此要整个事务完成后，在一起写到文件里。

而redo log 并没有这个要求，中间有生成的日志可以写到redo log buffer中，redo log buffer

中的内容还能“搭便车”，其他事务提交的时候可以被一起写到磁盘中。

**问题4：**如果binlog写完盘后发生crash,这时候还没给客户端答复就重启了，等客户端再重连进来，发现事物已经提交成功了，这个是不是bug？

回答：不是

整个事务都已经提交成功了，redo log commit完成，备库也收到binlog并执行了，但是主库和客户端网络断开，导致事务成功的包返回不回去，这时候客户端也会收到“网络断开”的异常，这种只能算是“事务成功的，不能认为是bug”.

实际上数据库的crash-safe保证的是：

1.如果客户端收到事务成功的消息，事务就一定持久化了

2.如果客户端收到事务失败（不如主键冲突，回滚等）的消息，事务就一定失败了

3.如果客户端收到“执行异常”的消息，应用需要重连后通过查询当前状态来继续后续的逻辑。此时数据库只需要保证内部（数据和日志之间，主库和备库之间）一致就可以了。