链滴

mysql binlog、redolog 和 undolog

作者：AshShawn

原文链接：https://ld246.com/article/1647828159844

来源网站：链滴

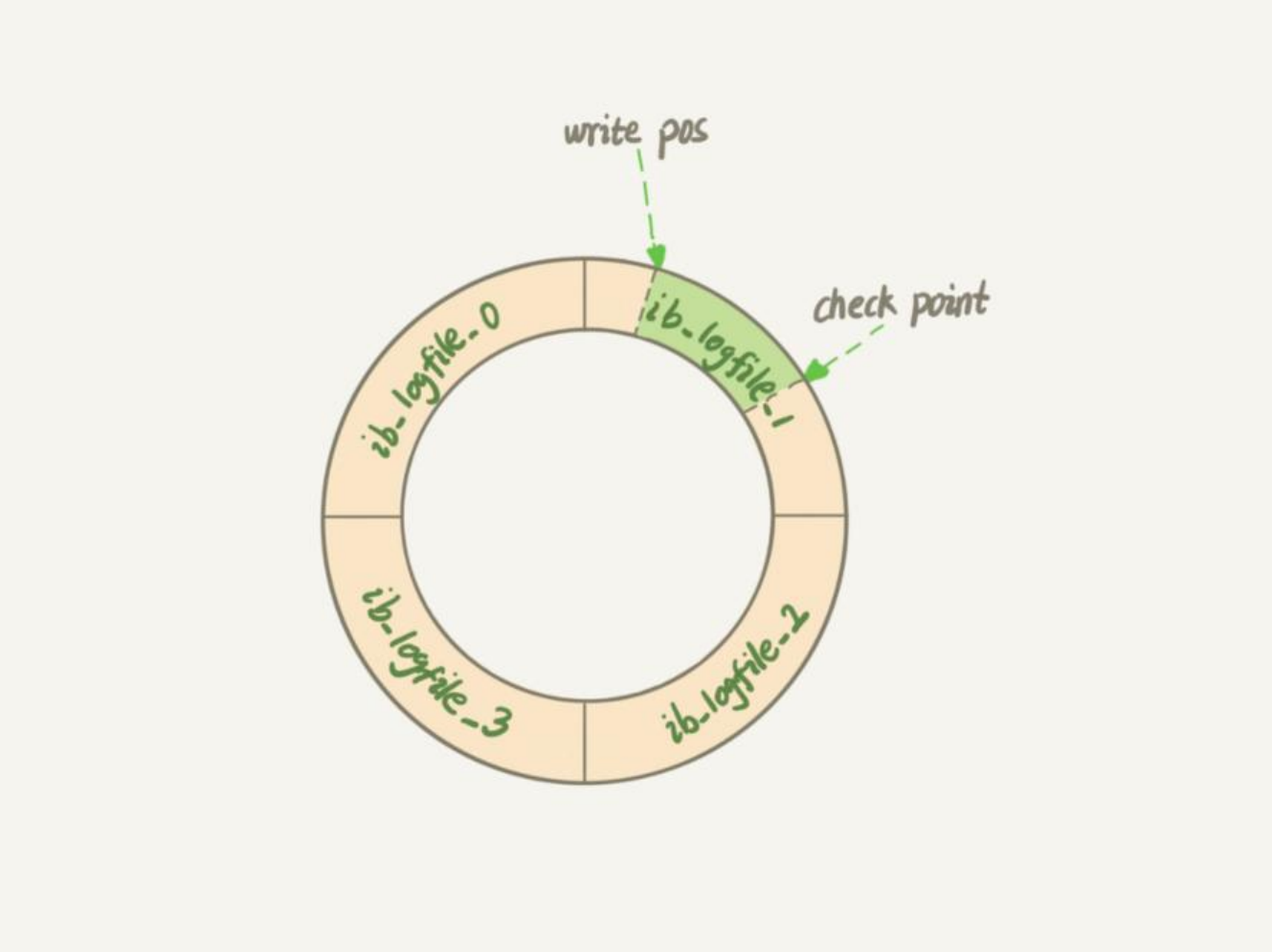
1. redo log

简述: 引擎层的redo log 是为了避免每一次更新操作都去查找记录再修改记录(随机读写),采用WAL技术Write-Ahead Logging写前日志)来转换成顺序读写;

WAL机制有两个好处:1将磁盘随机读写转为顺序写 2.组提交,可大幅提高磁盘的IOPS能力

更新数据时,innoDB先把记录写到redo log中,并更新内存,这时便可以返回结果.同时在InnoDB空闲的候将操作记录更新到磁盘里面.

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

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 之间是可记录空闲空间,当write pos 追上 checkpoint时,需要暂停更新并将据写入磁盘,然后推进下checkpoint

何时把脏页数据刷入磁盘?

概念: buffer pool 中会存在内存数据页,内存数据页与磁盘数据不一致时,该内存页成为脏页,否则称为净页.

1. redo log写满了,系统会停止所有更新操作,推进checkpoint,此时会刷新推进长度范围内的脏页

2. 内存不足,内存页在bufferpool中占用比超出限额(一般为75%),此时需要淘汰一些数据页, 如果淘汰是脏页,则需要刷盘

3. 空闲时期,会刷脏页

4. mysql正常关闭的时候会刷全部脏页

综上所述,刷脏页时在以下两种情况会严重影响性能:

1. 脏页淘汰数目过多,产生性能抖动,影响查询

2. redolog写满,阻塞所有写操作

刷脏页控制策略

1. 磁盘能力

```
show global variables like 'innodb_io_capacity'
--这个值建议设置成磁盘的IOPS
--iops通过fio工具可以测试
 fio -filename=$filename -direct=1 -iodepth 1 -thread -rw=randrw -ioengine=psync -bs=16k
size=500M -numjobs=10 -runtime=10 -group_reporting -name=mytest
 默认值是200,如果是SSD建议设置成20000
```

2. 脏页比例

```
show global variables like 'innodb_max_dirty_pages_pct' --默认75%

select VARIABLE_VALUE into @a from global_status where VARIABLE_NAME = 'Innodb_buffer
pool_pages_dirty';
select VARIABLE_VALUE into @b from global_status where VARIABLE_NAME = 'Innodb_buffer
pool_pages_total';
select @a/@b;
--通过监控脏页比例,不要让其经常接近75%
```

3. 连坐刷脏页

刷脏页时会检查相邻的数据页是否为脏页,如果是,则一起刷盘

```
innodb_flush_neighbors
--为0时表示只刷自身,如果是机械硬盘建议设置为1,如果是SSD,建议设置为0,响应延时会更低
```

4. 合理的redolog大小

redolog过小会导致刷盘几率过高,写能力会归零

redolog写满涉及到哪些操作

1. 把相对应的数据页中的脏页持久化到磁盘,checkpoint往前推

2. 由于redo log还记录了undo的变化,undo log buffer也要持久化进undo log

3. 当innodb_flush_log_at_trx_commit设置为非1,还要把内存里的redo log持久化到磁盘上

4. redo log还记录了change buffer的改变,那么还要把change buffer purge到idb

以及merge change buffer.merge生成的数据页也是脏页,也要持久化到磁盘

2. binlog

简述: Server层的binlog记录的是原始逻辑的语句,属于归档日志

binlog有两种模式,记录statement即sql语句,记录row即记录更改前后的两条row记录

binlog写入机制

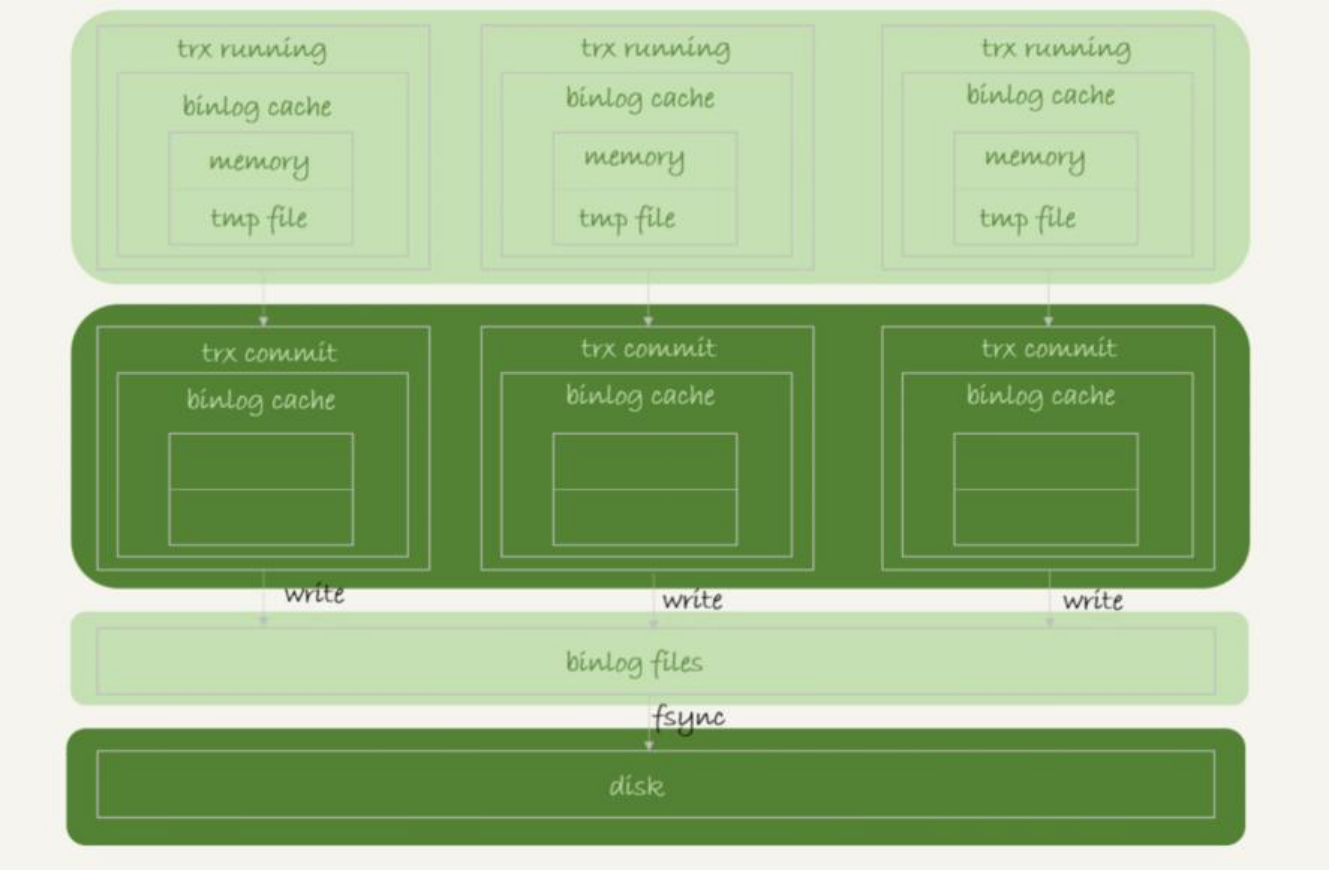
1. 事务执行过程中先把日志写到binlog cache ， 事务提交的时候再把binlog cache 写入到binlog文中

2. binlog cache ， 系统为每个线程分配了一片binlog cache内存。参数 binlog_cache_size控制单线程内binlog cache大小。如果超过这个大小就要暂存到磁盘

3. 事务提交的时候，执行器把binlog cache 里完整的事务写入binlog中。并清空binlog cache。

4. 每个线程都有自己的binlog cache，共用一份binlog文件

5. 下图write，是把日志写入到文件系统的page cache，内存中，没有持久化到磁盘，所以速度比较，图中的fsync是将数据持久化到磁盘，占用磁盘的IOPS



write 和 fsync 的时机，是由参数 sync_binlog 控制的：

- sync_binlog=0 的时候，表示每次提交事务都只 write，不 fsync；
- sync_binlog=1 的时候，表示每次提交事务都会执行 fsync；
- sync_binlog=N(N>1) 的时候，表示每次提交事务都 write，但累积 N 个事务后才 fsync。一般建设置成100~1000中的某个数值

原文链接：mysql binlog、redolog 和 undolog

redo log写入机制

为了控制 redo log 的写入策略，InnoDB 提供了 innodb_flush_log_at_trx_commit 参数。

- 设置为 0 的时候，表示每次事务提交时都只是把 redo log 留在 redo log buffer 中；
- 设置为 1 的时候，表示每次事务提交时都将 redo log 直接持久化到磁盘；这意味着redolog在prepare阶段就要持久化一次.
- 设置为 2 的时候，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，就会把 redo log buffer 中的日志，调用 write 写到文件系统的 page cache，然后调用 fsync 持久化到磁盘。（没有被提交的redo log也会被持久化到磁盘）

双1配置:

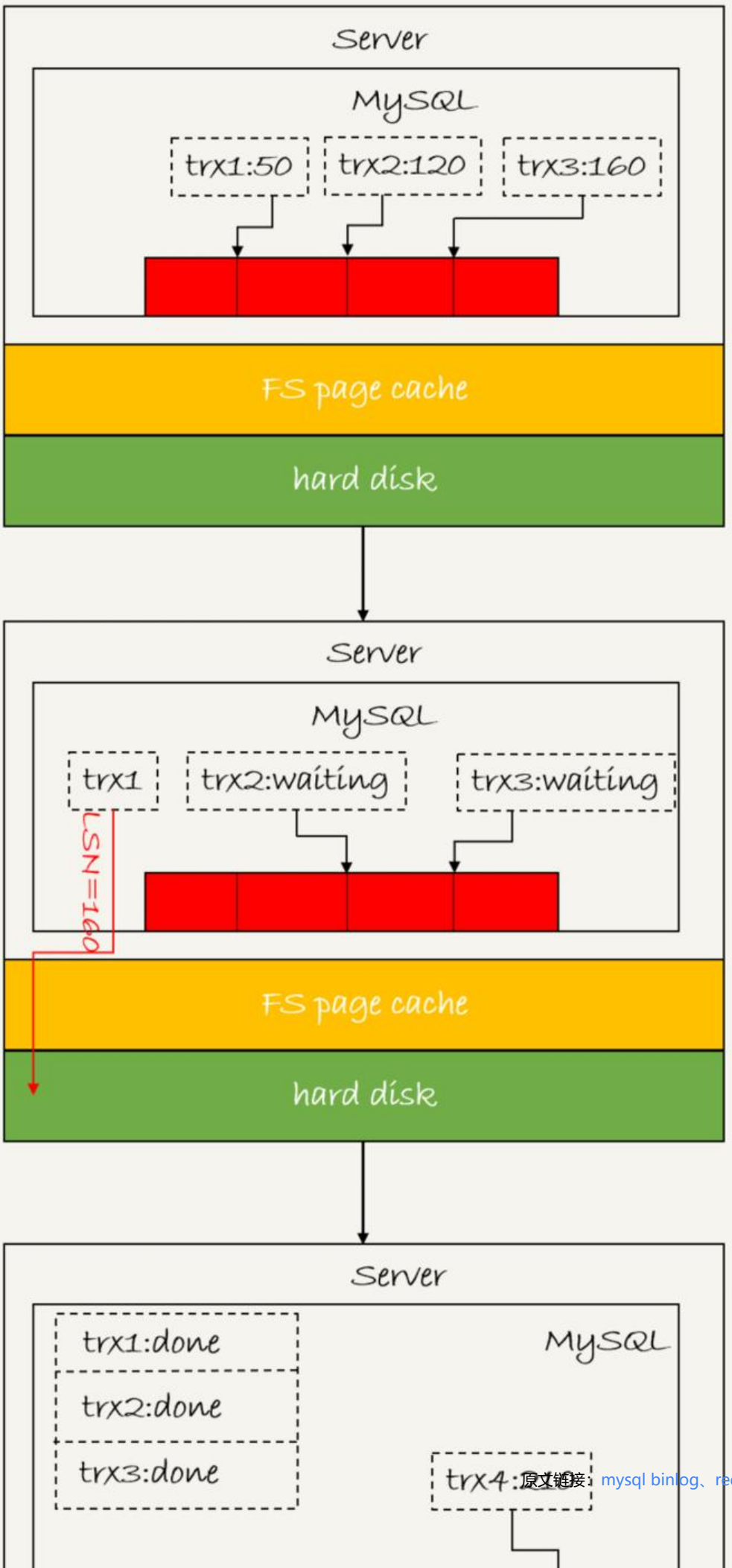
sync_binlog 和 innodb_flush_log_at_trx_commit 都设置成 1,即一个事务在完整提交前,需要等待两刷盘,一次是redo log (prepare阶段),一次是binlog

LSN:

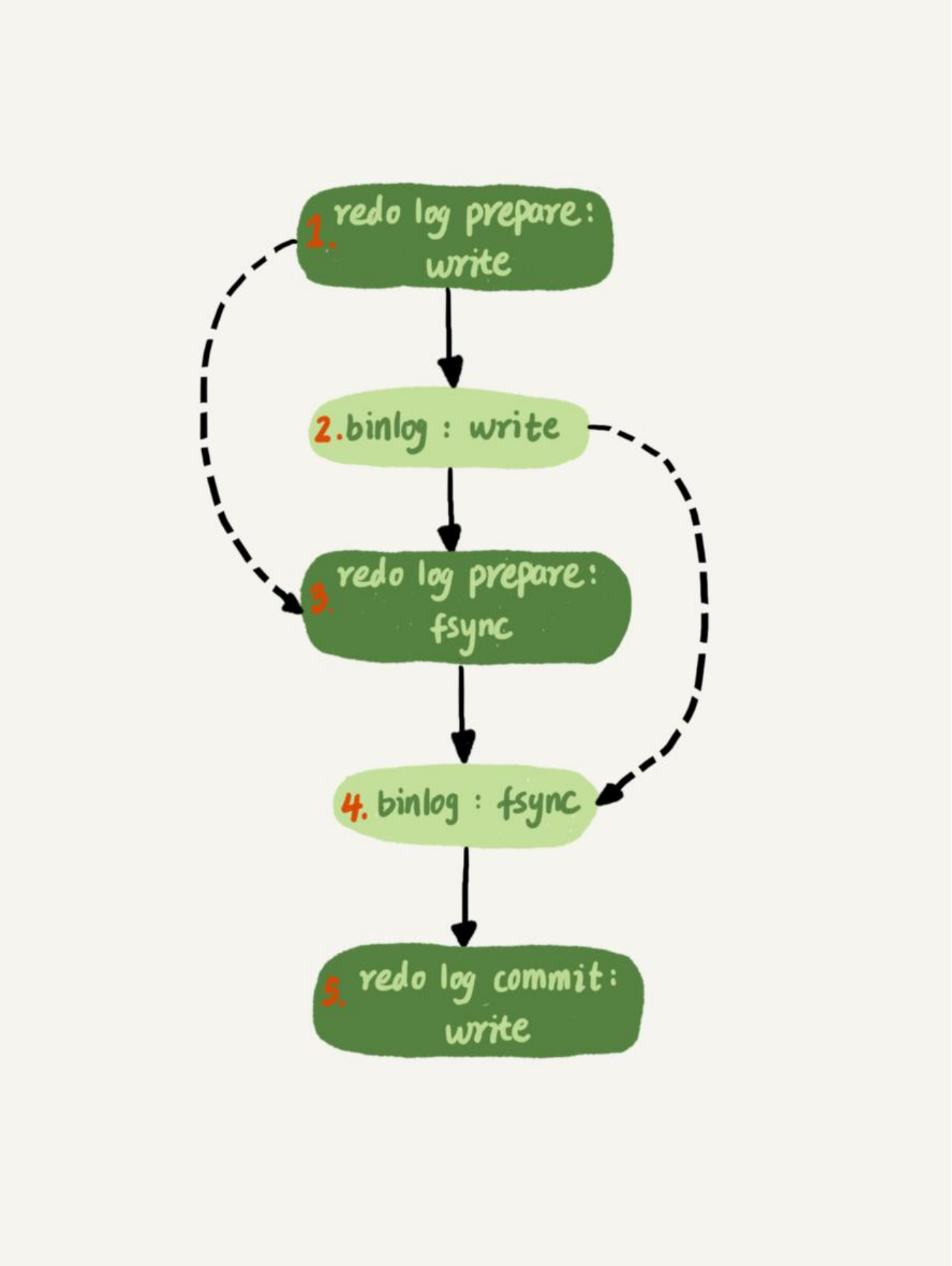
日志逻辑序列号,单调递增,用来对应redolog的一个个写入点,每次写入长度为 length 的 redo log， LN 的值就会加上 length。

组提交机制:

原文链接：mysql binlog、redolog 和 undolog


Server
MySQL
trx1:50
trx2:120
trx3:160
FS page cache
hard disk
Server
MySQL
trx1
trx2:waiting
trx3:waiting
LSN=160
FS page cache
hard disk
Server
trx1:done
trx2:done
trx3:done
MySQL
trx4:
原文链接：mysql binlog、redolog 和 undolog

trx1第一个到达,被选为leader;trx1去写盘的时候带的是该组最大的LSN,即160,当trx1返回时,改组所小于等于160的redo log 均被写入磁盘,trx2和trx3可以直接返回



如上图,redo log 尽量将1写内存与3 fsync尽量分隔开,为的是更好的拖延时间,让组提交的效率更高

原文链接：mysql binlog、redolog 和 undolog

同样,binlog也进行了组提交,但由于binlog 的 write 和 fsync 间的间隔时间短,效率并不是特别高

binlog_group_commit_sync_delay --参数，表示延迟多少微秒后才调用 fsync;
binlog_group_commit_sync_no_delay_count --参数，表示累积多少次以后才调用 fsync

3. binlog和redolog的区别

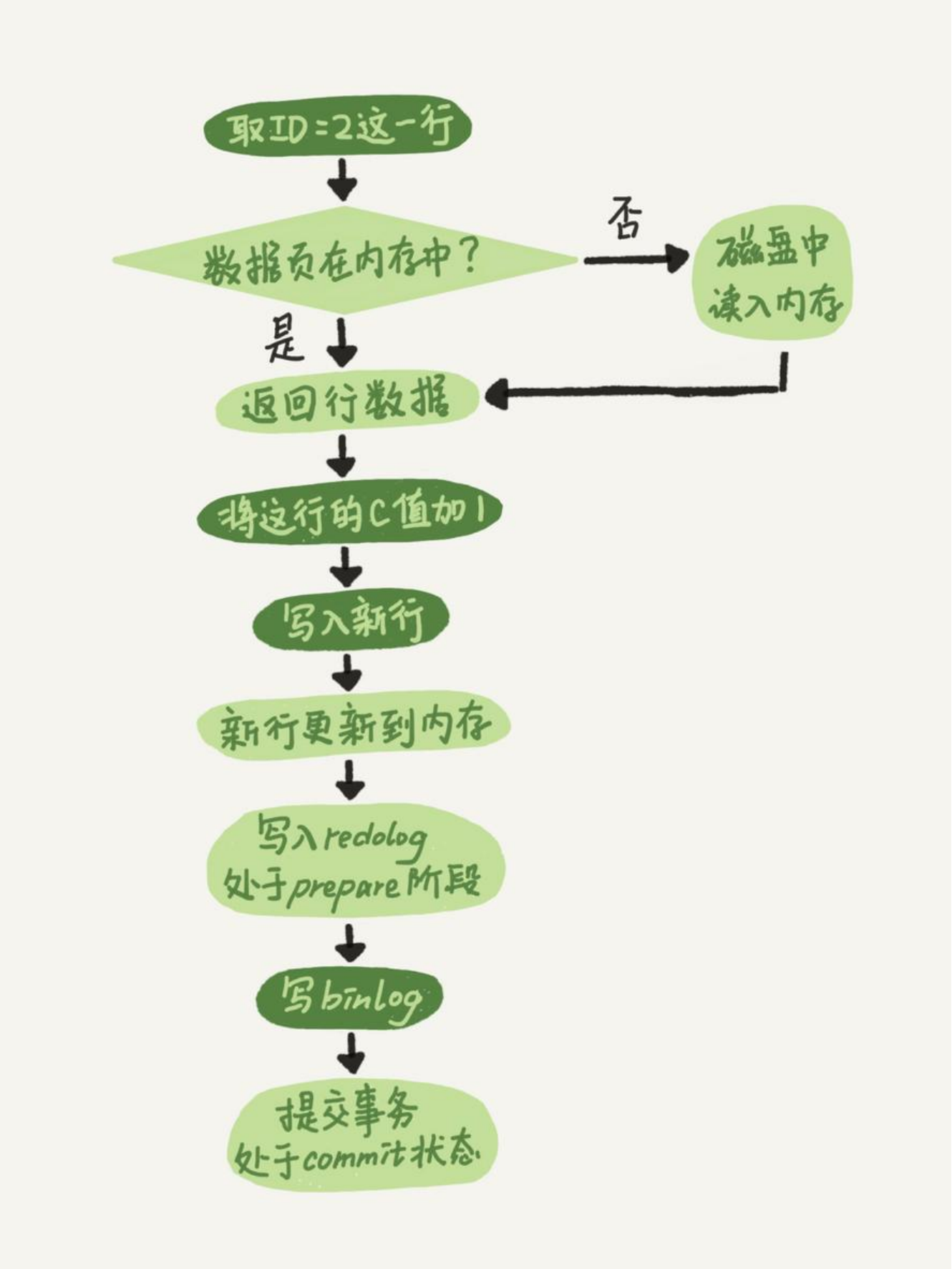
● redo log 是 InnoDB 引擎特有的；binlog 是 MySQL 的 Server 层实现的，所有引擎都可以使用。

● 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，记录的这个语句的原始逻辑，比如“给 ID=2 这一行的 c 字段加 1 ”。

● redo log 是循环写的，空间固定会用完；binlog 是可以追加写入的。 binlog 文件写到一定大小后切换到下一个。

4. 两阶段提交

update语句的执行流程图,深色表示server层,浅色表示innoDB层:



时刻A发生crash:

binlog未写,redolog未提交,应用恢复后会回滚事务,

原文链接：mysql binlog、redolog 和 undolog

时刻B发生crash:

redolog完整但出于prepare阶段,若binlog记录完整,则会自动提交事务,否则,回滚事务

问题一: 既然redolog和binlog记录完整就可以确认事务完整性了,那还有必要执行提交事务这个操作吗直接先写redolog再写binlog,不就行了?

binlog只是记录了做了什么操作,没有记录状态,而redolog是记录状态的,当发生崩溃恢复时,我们通过验redolog中的状态来判断是否需要去检查binlog,当为prepare时,校验binlog是否完整,当为commit时则不必校验binlog了. 如果每次都校验redolog和binlog来确认事务完整性则对性能来说有些浪费

问题二: 上图中的commit和事务中的commit是一回事吗?

不是, 上图中所有过程发生在事务commit的过程中

5. 参数

innodb_flush_log_at_trx_commit //建议设置成1,每次事务的redolog都会直接持久化到磁盘

sync_binlog //建议设置成1,每次事务的binlog都会持久化到磁盘

6. 常见疑问

1.不能只有redolog或者binlog吗,不是依旧可以还原数据吗?

答:只有redolog情况,redolog是环形的,且容量有限,且redolog是innodb引擎级别的日志

只有binlog, binlog没有被用来做崩溃恢复,一开始就是这么设计的,而且binlog可以被关闭,依靠binlo靠不住

2.IO瓶颈如何提升性能

● 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时间，但没有丢失数据的风险。

● 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，主机掉电时会丢 binlog 日志。

● 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。

3. 为什么 binlog cache 是每个线程自己维护的，而 redo log buffer 是全局共用的?

binlog是逻辑日志,需要用于主从同步,必须保证不能被打断,所以一个事务的binlog必须连着写

redo log是物理日志,生成的日志可以写到redolog buffer中,其他事务提交时可以一起被写入磁盘中,page页天生就是共有数据

4. 数据库crash-safe保证的是什么:

如果客户端收到事务成功的消息，事务就一定持久化了；

如果客户端收到事务失败（比如主键冲突、回滚等）的消息，事务就一定失败了；

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。