Mysql的各种log（binlog、redo log、undo log）

原创 CodeMan22 于 2021-01-07 16:17:53 发布 2836 收藏 12 版权
分类专栏： mysql 文章标签： mysql

MySQL常见的三种log：

- `binlog`
- `redolog`
- `undolog`

binlog

binlog用于记录数据库执行的写入性操作(不包括查询)信息，以二进制的形式保存在磁盘中。binlog是mysql的逻辑日志，并且由Server层进行记录，使用任何存储引擎的mysql数据库都会记录binlog日志。

可以简单理解为：存储着每条变更的SQL语句（可能不止SQL，还有XID「事务Id」等等）

binlog是通过追加的方式进行写入的，可以通过max_binlog_size参数设置每个binlog文件的大小，当文件大小达到给定值之后，会生成新的文件来保存日志。

> 逻辑日志：可以简单理解为记录的就是sql语句。
> 物理日志：因为mysql数据最终是保存在数据页中的，物理日志记录的就是数据页变更。

```
[root@vm-002 mysql]# mysqlbinlog mysql-bin.000002
.............
# at 624
#160925 21:29:53 server id 1 end_log_pos 796 Query thread_id=3 exec_time=0 error_code=0
SET TIMESTAMP=1474810193/*!*/;
insert into member(`name`,`sex`,`age`,`classid`) values('wangshibo','m',27,'cls1'),
('guohuihui','w',27,'cls2')       #执行的sql语句
/*!*/;
# at 796
#160925 21:29:53 server id 1 end_log_pos 823 Xid = 17              #执行的时间
.............
```


用途

主要有两个作用： 复制 和 恢复数据

- MySQL使用一主多从结构时，从服务器需要与主服务器的数据保持一致，通过binlog来实现的

- 可以通过binlog对数据进行恢复

binlog刷盘时机

对于InnoDB存储引擎而言，只有在事务提交时才会记录biglog，此时记录还在内存中，那么biglog是什么时候刷到磁盘中的呢？mysql通过sync_binlog参数控制biglog的刷盘时机，取值范围是0-N：

- 0：不去强制要求，由系统自行判断何时写入磁盘；
- 1：每次commit的时候都要将binlog写入磁盘；
- N：每N个事务，才会将binlog写入磁盘。

从上面可以看出，sync_binlog最安全的是设置是1，这也是MySQL 5.7.7之后版本的默认值。但是设置一个大一些的值可以提升数据库性能，因此实际情况下也可以将值适当调大，牺牲一定的一致性来获取更好的性能。

binlog日志格式

binlog日志有三种格式，分别为STATMENT、ROW和MIXED。

> 在 MySQL 5.7.7之前，默认的格式是STATEMENT，MySQL 5.7.7之后，默认值是ROW。日志格式通过binlog-format指定。

STATEMENT

基于SQL语句的复制(statement-based replication, SBR)，每一条会修改数据的sql语句会记录到binlog中。

- 优点：不需要记录每一行的变化，减少了binlog日志量，节约了IO, 从而提高了性能；
- 缺点：在某些特定情况下会导致主从数据不一致，比如执行sysdate()、slepp()等。

ROW

基于行的复制(row-based replication, RBR)，不记录每条sql语句的上下文信息，仅需记录哪条数据被修改了。

- 优点：不会出现某些特定情况下的存储过程、或function、或trigger的调用和触发无法被正确复制的问题；
- 缺点：会产生大量的日志，尤其是alter table的时候会让日志暴涨

MIXED

基于STATMENT和ROW两种模式的混合复制(mixed-based replication, MBR)，一般的复制使用STATEMENT模式保存binlog，对于STATEMENT模式无法复制的操作使用ROW模式保存binlog

redo log

为什么需要redo log

事务的四大特性里面有一个是持久性，具体来说就是只要事务提交成功，那么对数据库做的修改就被永久保存下来了，不可能因为任何原因再回到原来的状态。那么mysql是如何保证持久性的呢？最简单的做法是在每次事务提交的时候，将该事务涉及修改的数据页全部刷新到磁盘中。但是这么做会有严重的性能问题，主要体现在两个方面：

- 因为Innodb是以页为单位进行磁盘交互的，而一个事务很可能只修改一个数据页里面的几个字节，这个时候将完整的数据页刷到磁盘的话，太浪费资源了！
- 一个事务可能涉及修改多个数据页，并且这些数据页在物理上并不连续，使用**随机IO写入性能太差**！

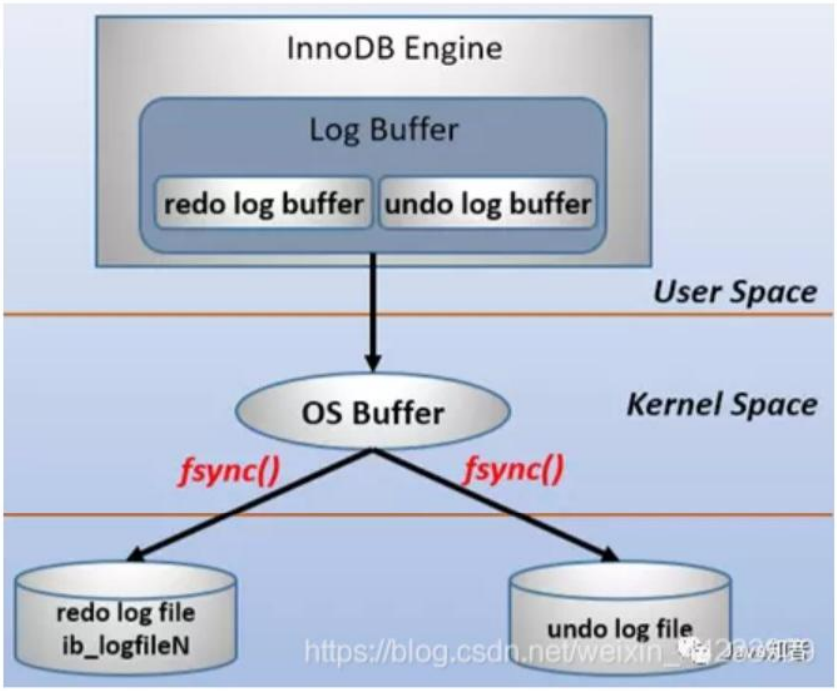
因此mysql设计了redo log，具体来说就是只记录事务对数据页做了哪些修改，这样就能完美地解决性能问题了(相对而言**文件更小并且是顺序IO**)。

基本概念

redo log是在innodb即`存储引擎层`产生的，物理日志。

redo log包括两部分：一个是内存中的日志缓冲(redo log buffer)，另一个是磁盘上的日志文件(redo log file)。mysql每执行一条DML语句，先将记录写入redo log buffer，后续某个时间点再一次性将多个操作记录写到redo log file。这种先写日志，再写磁盘的技术就是MySQL里经常说到的WAL(`Write-Ahead Logging`) 技术。

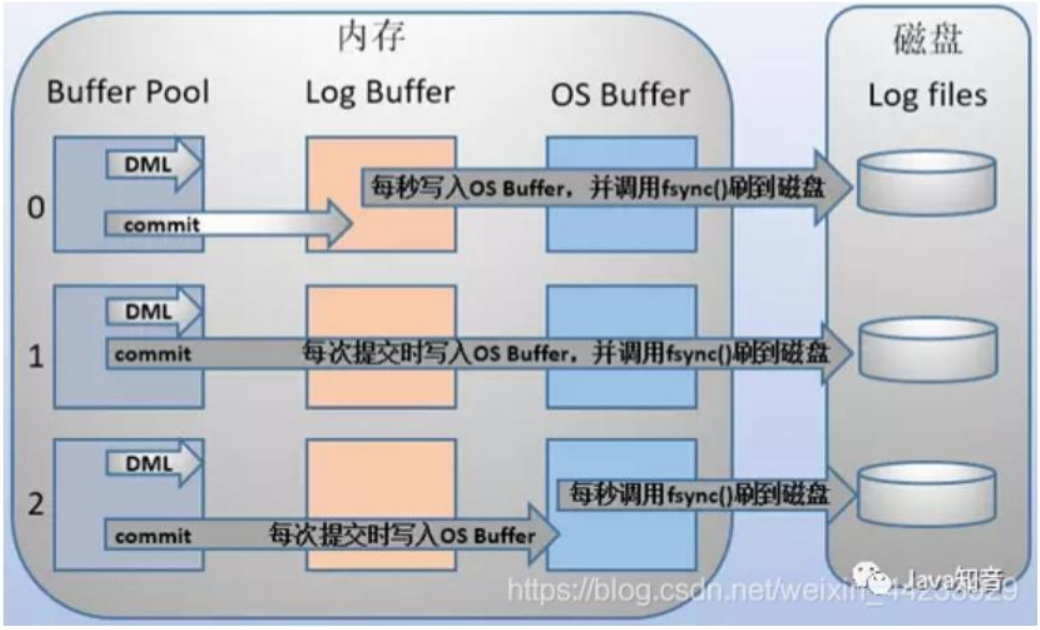
在计算机操作系统中，用户空间(user space)下的缓冲区数据一般情况下是无法直接写入磁盘的，中间必须经过操作系统内核空间(kernel space)缓冲区(OS Buffer)。因此，redo log buffer写入redo log file实际上是先写入OS Buffer，然后再通过系统调用fsync()将其刷到redo log file中，过程如下：



mysql支持三种将redo log buffer写入redo log file的时机，可以通过innodb_flush_log_at_trx_commit

参数配置，各参数值含义如下：

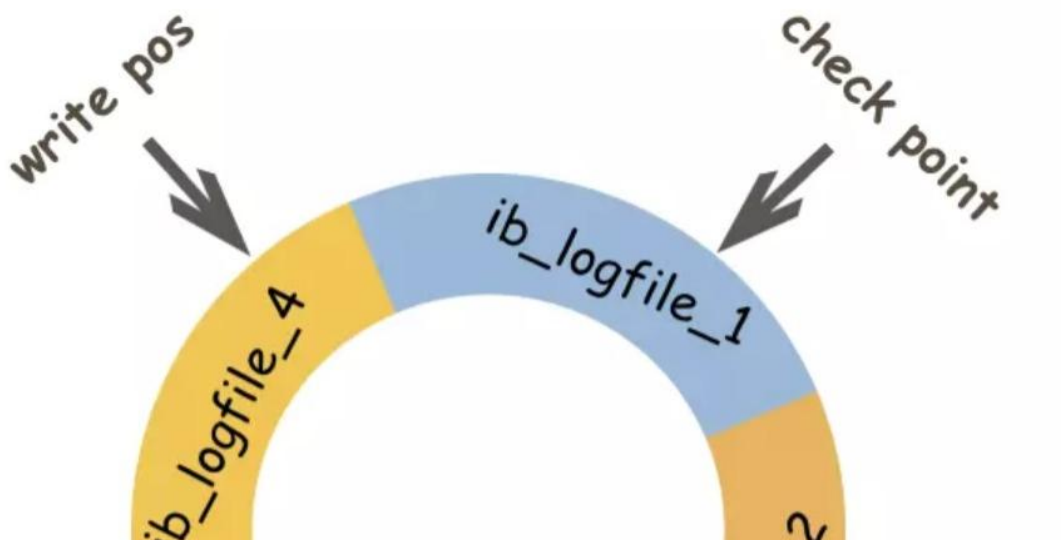
参数值	含义
0（延迟写）	事务提交时不会将redo log buffer中日志写入到os buffer，而是每秒写入os buffer并调用fsync()写入到redo log file中。也就是说设置为0时是(大约)每秒刷新写入到磁盘中的，当系统崩溃，会丢失1秒钟的数据。
1（实时写，实时刷）	事务每次提交都会将redo log buffer中的日志写入os buffer并调用fsync()刷到redo log file中。这种方式即使系统崩溃也不会丢失任何数据，但是因为每次提交都写入磁盘，IO的性能较差。
2（实时写，延迟刷）	每次提交都仅写入到os buffer，然后是每秒调用fsync()将os buffer中的日志写入到redo log file。

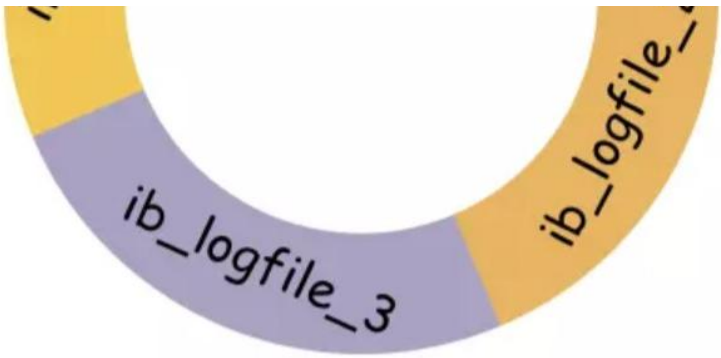


Mysql的 基本存储结构是页 (记录都存在页里边)，所以MySQL是先把这条记录所在的页找到，然后把该页加载到内存中，将对应记录进行修改

redo log记录形式

redo log实际上记录数据页的变更，而这种变更记录是没必要全部保存，因此redo log实现上采用了大小固定，循环写入的方式，当写到结尾时，会回到开头循环写日志。如下图：





在innodb中，既有redo log需要刷盘，还有数据页也需要刷盘，redo log存在的意义主要就是降低对数据页刷盘的要求。在上图中，write pos表示redo log当前记录的LSN(逻辑序列号)位置，check point表示数据页更改记录刷盘后对应redo log所处的LSN(逻辑序列号)位置。

write pos到check point之间的部分是redo log空着的部分，用于记录新的记录；check point到write pos之间是redo log待落盘的数据页更改记录。当write pos追上check point时，会先推动check point向前移动，空出位置再记录新的日志。

启动innodb的时候，不管上次是正常关闭还是异常关闭，总是会进行恢复操作。因为redo log记录的是数据页的物理变化，因此恢复的时候速度比逻辑日志(如binlog)要快很多。

重启innodb时，首先会检查磁盘中数据页的LSN，如果数据页的LSN小于日志中的LSN，则会从checkpoint开始恢复。

还有一种情况，在宕机前正处于checkpoint的刷盘过程，且数据页的刷盘进度超过了日志页的刷盘进度，此时会出现数据页中记录的LSN大于日志中的LSN，这时超出日志进度的部分将不会重做，因为这本身就表示已经做过的事情，无需再重做。

redo log	binlog	
文件大小	redo log的大小是固定的。	binlog可通过配置参数max_binlog_size设置每个binlog文件的大小。
实现方式	redo log是InnoDB引擎层实现的，并不是所有引擎都有。	binlog是Server层实现的，所有引擎都可以使用binlog日志
记录方式	redo log 采用循环写的方式记录，当写到结尾时，会回到开头循环写日志。	binlog 通过追加的方式记录，当文件大小大于给定值后，后续的日志会记录到新的文件上
适用场景	redo log适用于崩溃恢复(crash-safe)	binlog适用于主从复制和数据恢复

binlog和redo log

由binlog和redo log的区别可知：binlog日志只用于归档，只依靠binlog是没有crash-safe能力的。但只有redo log也不行，因为redo log是InnoDB特有的，且日志上的记录落盘后会被覆盖掉。因此需要binlog和redo log二者同时记录，才能保证当数据库发生宕机重启时，数据不会丢失。

> crash-safe：redo log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

区别：

存储内容

binlog记载的是update/delete/insert这样的SQL语句，而redo log记载的是物理修改的内容（xxxx页修改了xxx）。

`redo log` 记录的是数据的`物理变化`， `binlog` 记录的是数据的`逻辑变化`

功能

redo log是为持久性、恢复数据用的，而binlog不仅可以用于恢复数据，还用于主从复制。

redo log是一个环形结构，是有限的，数据刷到磁盘后redo log就无效，如果redo log满了，mysql还得停一下来刷新数据到磁盘，也就是刷脏页。

写入细节

redo log是MySQL的`InnoDB`引擎所产生的。

binlog无论MySQL用什么引擎，都会有的。

InnoDB事务的持久性就是靠redo log来实现的（如果写入内存成功，但数据还没真正刷到磁盘，如果此时的数据库挂了，我们可以靠redo log来恢复内存的数据，这就实现了持久性）

redo log事务开始的时候，就开始记录每次的变更信息，而binlog是在事务提交的时候才记录。

问题出现了：写其中的某一个log，失败了，那会怎么办？现在我们的前提是先写redo log，再写binlog，我们来看看：

- 如果写redo log失败了，那我们就认为这次事务有问题，回滚，不去写binlog。
- 如果写redo log成功了，写binlog，写binlog写一半了，但失败了怎么办？我们还是会对这次的事务回滚，将无效的binlog给删除（因为binlog会影响从库的数据，所以需要做删除操作）
- 如果写redo log和binlog都成功了，那这次算是事务才会真正成功。

简单来说：MySQL需要保证redo log和binlog的数据是一致的，如果不一致，那就乱套了。

- 如果redo log写失败了，而binlog写成功了。那假设内存的数据还没来得及落磁盘，机器就挂掉了。那主从服务器的数据就不一致了。（从服务器通过binlog得到最新的数据，而主服务器由于redo log没有记载，没法恢复数据）。而且Mysql迅速恢复的时候redo log没有这条信息，而binlog又有，如果用binlog去恢复数据，会让Mysql看起来像是多做了一些事。
- 如果redo log写成功了，而binlog写失败了。那从服务器就拿不到最新的数据了，而且以后做数据备份的时候也会丢失了这条记录。

MySQL通过 `两阶段提交` 来保证redo log和binlog的数据是一致的。

过程：

- 阶段1：redo log 写盘，处于 prepare 状态
- 阶段2：binlog 写盘，redo log 处于 commit 状态

undo log

undo log主要有两个作用： `回滚` 和 `多版本控制(MVCC)`

在数据修改的时候，不仅记录了redo log，还记录undo log，如果因为某些原因导致事务失败或回滚了，可以用undo log进行回滚（保证了原子性）

undo log主要存储的是 `逻辑日志` ，用来回滚的 `相反操作日志` 。比如我们要insert一条数据了，那undo log会记录的一条对应的delete日志。我们要update一条记录时，它会记录一条对应相反的update记录。

因为undo log存储着修改之前的数据，相当于一个前版本，MVCC实现的是读写不阻塞，读的时候只要返回前一个版本的数据就行了。