https://mp.weixin.qq.com/s/LZTCxYBiC0O-GhUGXg7GAg

**binlog/redo log/undo log？再也不会傻傻分不清楚了**

mysql日志主要包括错误日志、查询日志、慢查询日志、事务日志、二进制日志几大类。

## ****# 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。

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。

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。

### ****binlog使用场景****

在实际应用中，binlog的主要使用场景有两个，分别是主从复制和数据恢复。

1. 主从复制：在Master端开启binlog，然后将binlog发送到各个Slave端，Slave端重放binlog从而达到主从数据一致。
2. 数据恢复：通过使用mysqlbinlog工具来恢复数据。

### ****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。

* STATMENT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****

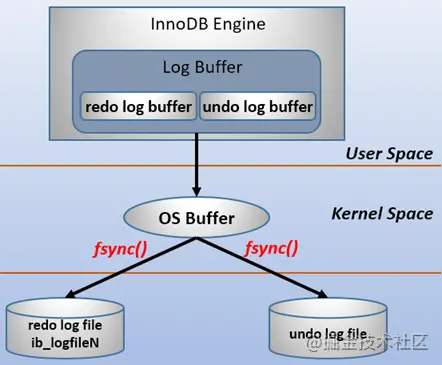
我们都知道，事务的四大特性里面有一个是持久性，具体来说就是只要事务提交成功，那么对数据库做的修改就被永久保存下来了，不可能因为任何原因再回到原来的状态。那么mysql是如何保证持久性的呢？最简单的做法是在每次事务提交的时候，将该事务涉及修改的数据页全部刷新到磁盘中。但是这么做会有严重的性能问题，主要体现在两个方面：

1. 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！
2. 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)。

### ****redo log基本概念****

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：

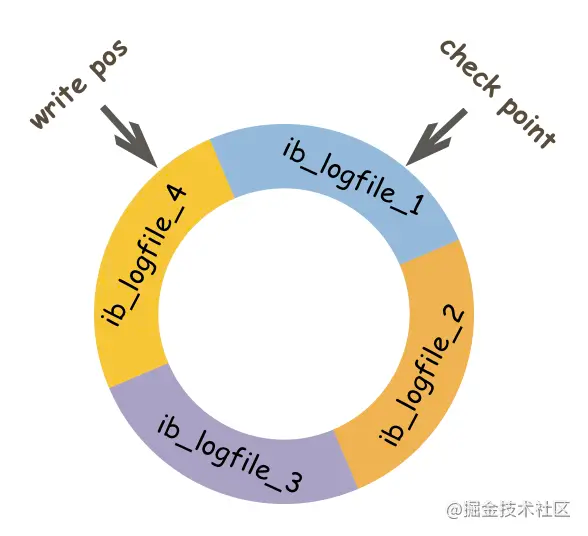


mysql支持三种将redo log buffer写入redo log file的时机，可以通过innodb\_flush\_log\_at\_trx\_commit参数配置，各参数值含义如下：



### ****redo log记录形式****

前面说过，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：



同时我们很容易得知，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与binlog区别****



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。

## ****# undo log****

数据库事务四大特性中有一个是原子性，具体来说就是 原子性是指对数据库的一系列操作，要么全部成功，要么全部失败，不可能出现部分成功的情况。实际上，原子性底层就是通过undo log实现的。

undo log主要记录了数据的逻辑变化，比如一条INSERT语句，对应一条DELETE的undo log，对于每个UPDATE语句，对应一条相反的UPDATE的undo log，这样在发生错误时，就能回滚到事务之前的数据状态。