<https://mp.weixin.qq.com/s/JrApetd6xTb-69oWjn75fA>

MySQL里边还有几种常见的log，分别为：

* undo log
* binlog
* redo log

## 一、binlog

binlog其实在日常的开发中是听得很多的，因为很多时候数据的更新就依赖着binlog。

举个很简单的例子：我们的数据是保存在数据库里边的嘛，现在我们对某个商品的某个字段的内容改了（数据库变更），而**用户检索的出来数据是走搜索引擎的**。为了让用户能搜到最新的数据，我们需要把引擎的数据也改掉。

一句话：**数据库的变更，搜索引擎的数据也需要变更**。

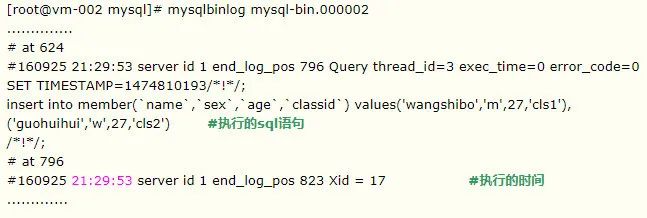
于是，我们就会监听binlog的变更，如果binlog有变更了，那我们就需要将变更写到对应的数据源。

### 什么是binlog？

binlog记录了数据库表结构和表数据变更，比如update/delete/insert/truncate/create。它不会记录select（因为这没有对表没有进行变更）

### binlog长什么样？

binlog我们可以简单理解为：存储着每条变更的SQL语句（当然从下面的图看来看，不止SQL，还有XID「事务Id」等等）



### binlog一般用来做什么

主要有两个作用：**复制和恢复数据**

MySQL在公司使用的时候往往都是**一主多从**结构的，从服务器需要与主服务器的数据保持一致，这就是通过binlog来实现的

数据库的数据被干掉了，我们可以通过binlog来对数据进行恢复。

因为binlog记录了数据库表的变更，所以我们可以用binlog进行复制（主从复制)和恢复数据。

## 二、redo log

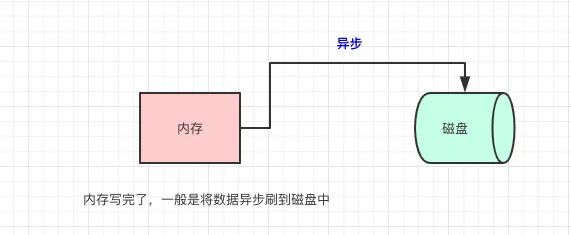
假设我们有一条sql语句：

update user\_table set name='java3y' where id = '3'

MySQL执行这条SQL语句，肯定是先把id=3的这条记录查出来，然后将name字段给改掉。这没问题吧？

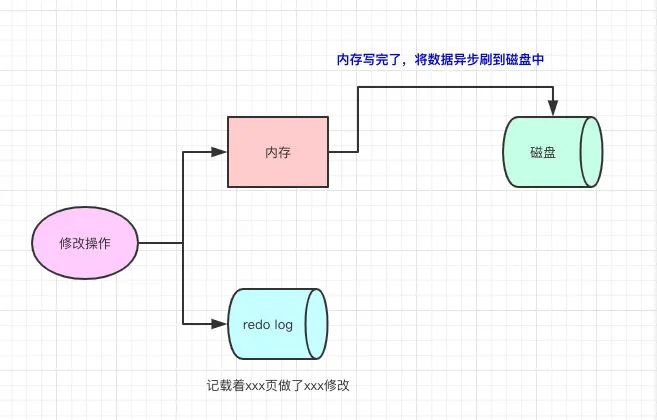
实际上Mysql的基本存储结构是**页**(记录都存在页里边)，所以MySQL是先把这条记录所在的**页**找到，然后把该页加载到内存中，将对应记录进行修改。

现在就可能存在一个问题：**如果在内存中把数据改了，还没来得及落磁盘，而此时的数据库挂了怎么办**？显然这次更改就丢了。

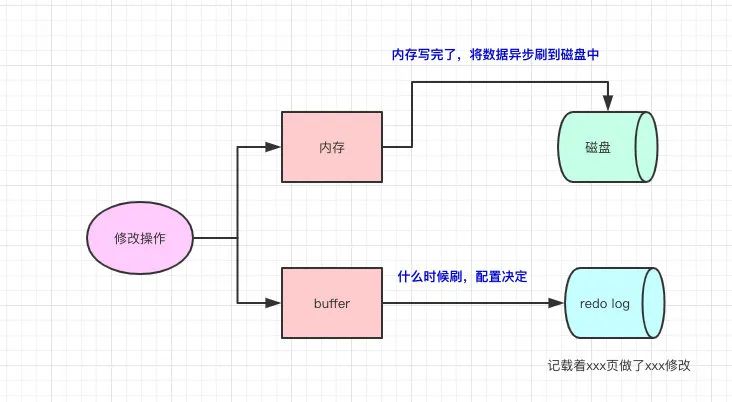


如果每个请求都需要将数据**立马**落磁盘之后，那速度会很慢，MySQL可能也顶不住。所以MySQL是怎么做的呢？

MySQL引入了redo log，内存写完了，然后会写一份redo log，这份redo log记载着这次**在某个页上做了什么修改**。



其实写redo log的时候，也会有buffer，是先写buffer，再真正落到磁盘中的。至于从buffer什么时候落磁盘，会有配置供我们配置。



写redo log也是需要写磁盘的，但它的好处就是顺序IO（我们都知道顺序IO比随机IO快非常多）。

所以，redo log的存在为了：当我们修改的时候，写完内存了，但数据还没真正写到磁盘的时候。此时我们的数据库挂了，我们可以根据redo log来对数据进行恢复。因为redo log是顺序IO，所以**写入的速度很快**，并且redo log记载的是物理变化（xxxx页做了xxx修改），文件的体积很小，**恢复速度很快**。

## 三、binlog和redo log的区别

看到这里，你可能会想：binlog和redo log 这俩也太像了吧，都是用作”恢复“的。

其实他俩除了"恢复"这块是相似的，很多都不一样，下面看我列一下。

### 存储的内容

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）。

所以在搜索资料的时候会有这样的说法：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的作用是为**持久化**而生的。写完内存，如果数据库挂了，那我们可以通过redo log来恢复内存还没来得及刷到磁盘的数据，将redo log加载到内存里边，那内存就能恢复到挂掉之前的数据了。

binlog的作用是复制和恢复而生的。

主从服务器需要保持数据的一致性，通过binlog来同步数据。

如果整个数据库的数据都被删除了，binlog存储着所有的数据变更情况，那么可以通过binlog来对数据进行恢复。

又看到这里，你会想：”如果整个数据库的数据都被删除了，那我可以用redo log的记录来恢复吗？“**不能**

因为功能的不同，redo log 存储的是物理数据的变更，如果我们内存的数据已经刷到了磁盘了，那redo log的数据就无效了。所以redo log不会存储着**历史**所有数据的变更，**文件的内容会被覆盖的**。

### binlog和redo log 写入的细节

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是有事务的，事务的四大特性之一：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）。

上面也提到，在修改的数据的时候，binlog会记载着变更的类容，redo log也会记载着变更的内容。（只不过一个存储的是物理变化，一个存储的是逻辑变化）。那他们的写入顺序是什么样的呢？

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。

于是新有的问题又出现了：我写其中的某一个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，不再写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）

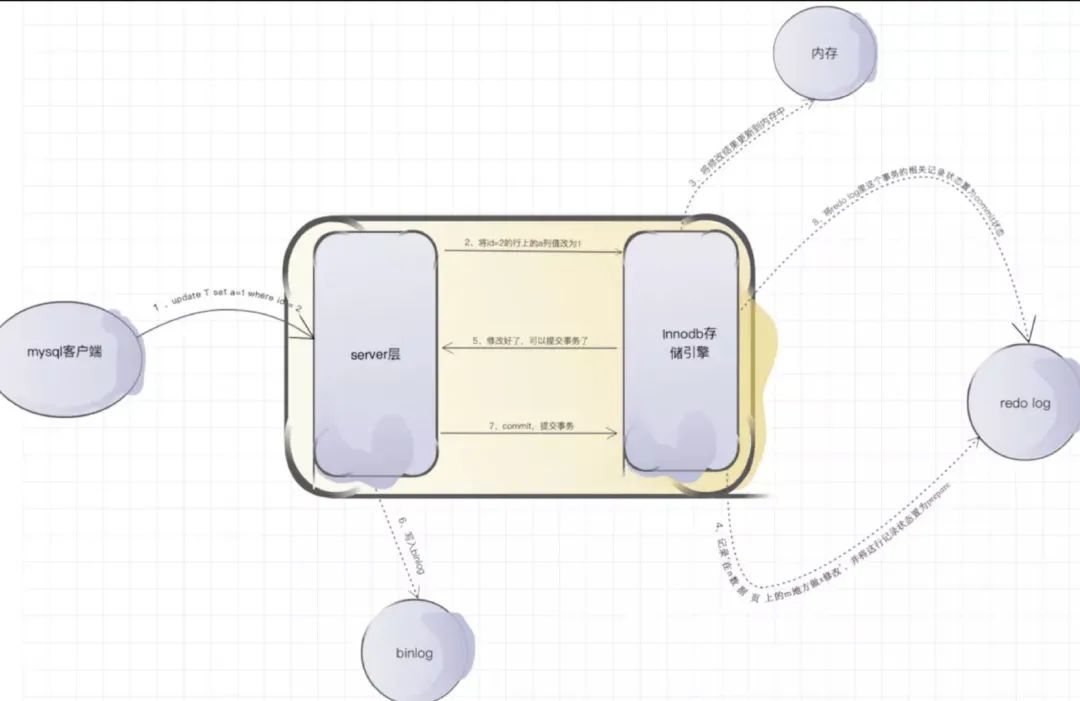
如果写redo log和binlog都成功了，那这次算是事务才会真正成功。

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）

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。

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：InnoDB redo log 写盘，InnoDB 事务进入 prepare 状态

阶段2：binlog 写盘，InooDB 事务进入 commit 状态

每个事务binlog的末尾，会记录一个 XID event，标志着事务是否提交成功，也就是说，恢复过程中，binlog 最后一个 XID event 之后的内容都应该被 purge。



## 四、什么是undo log

undo log有什么用？

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

undo log主要存储的也是逻辑日志，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。

这也应该容易理解，毕竟回滚嘛，跟需要修改的操作相反就好，这样就能达到回滚的目的。因为支持回滚操作，所以我们就能保证：“**一个事务包含多个操作，这些操作要么全部执行，要么全都不执行**”。【原子性】

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。

五、redo-log的刷盘策略

* 策略1, set global innodb\_flush\_log\_at\_trx\_commit = 0

该方式可以获得最佳性能。

每隔1秒种，才将Log Buffer中的数据批量写入OS Cache，同时MySQL主动fsync。

这种策略，如果 [数据库](http://www.codercto.com/category/database.html) 崩溃，会有1秒的数据丢失。

* 策略2, set global innodb\_flush\_log\_at\_trx\_commit = 1

该方式可以获得强一致性。

每次事务提交，都将Log Buffer中数据写入到OS Cache，同时MySQL主动fsync。

这种策略是InnoDB默认配置，是为了保证事务ACID特性。

* 策略3, set global innodb\_flush\_log\_at\_trx\_commit = 2

该方式则是一种trade-off的结果。

每次事务提交，都将Log Buffer中的数据写入OS Cache；

每隔1秒，MySQL主动将OS Cache中数据批量fsync。

这种策略，如果操作系统崩溃，最多会有1秒的数据丢失。（磁盘IO次数是不确定的，因为OS的fsync的频率并不是MySQL能控制的；OS也会进行fsync，而MySQL主动fsync的周期是1秒，所以最多丢1秒数据）

那么，透漏个小秘密，高并发业务场景下，最佳实践会如何配置呢？

答案是 set global innodb\_flush\_log\_at\_trx\_commit = 2 。因为不仅可以保证性能，也相对可以保障安全性（只要操作系统不出问题，数据就不会丢。而比起MySQL应用出问题，操作系统出问题概率小很多）