MySQL中有以下日志文件，分别是：

　　1：**重做日志（redo log）**

　　2：**回滚日志（undo log）**

　　3：**二进制日志（binlog）**

　　4：**错误日志（errorlog）**

　　5：**慢查询日志（slow query log）**

　　6：**一般查询日志（general log）**

　　7：**中继日志（relay log）。**

其中重做日志和回滚日志与事务操作息息相关，二进制日志也与事务操作有一定的关系，这三种日志，对理解MySQL中的事务操作有着重要的意义。

**一、重做日志（redo log）**

**作用：**

　　确保事务的持久性。redo日志记录事务执行后的状态，用来恢复未写入data file的已成功事务更新的数据。防止在发生故障的时间点，尚有脏页未写入磁盘，在重启[mysql](https://www.2cto.com/database/MySQL/)服务的时候，根据redo log进行重做，从而达到事务的持久性这一特性。

**内容：**

　　物理格式的日志，记录的是物理数据页面的修改的信息，其redo log是顺序写入redo log file的物理文件中去的。

**什么时候产生：**

　　事务开始之后就产生redo log，redo log的落盘并不是随着事务的提交才写入的，而是在事务的执行过程中，便开始写入redo log文件中。

**什么时候释放：**

　　当对应事务的脏页写入到磁盘之后，redo log的使命也就完成了，重做日志占用的空间就可以重用（被覆盖）。

**对应的物理文件：**

　　默认情况下，对应的物理文件位于[数据库](https://www.2cto.com/database/" \t "_blank)的data目录下的ib\_logfile1&ib\_logfile2

　　innodb\_log\_group\_home\_dir 指定日志文件组所在的路径，默认./ ，表示在数据库的数据目录下。

　　innodb\_log\_files\_in\_group 指定重做日志文件组中文件的数量，默认2

**关于文件的大小和数量，由以下两个参数配置：**

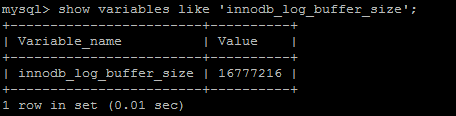
　　innodb\_log\_file\_size 重做日志文件的大小。

　　innodb\_mirrored\_log\_groups 指定了日志镜像文件组的数量，默认1

**其他：**

　　很重要一点，redo log是什么时候写盘的？前面说了是在事物开始之后逐步写盘的。

　　之所以说重做日志是在事务开始之后逐步写入重做日志文件，而不一定是事务提交才写入重做日志缓存，原因就是，重做日志有一个缓存区Innodb\_log\_buffer，Innodb\_log\_buffer的默认大小为8M(这里设置的16M),Innodb存储引擎先将重做日志写入innodb\_log\_buffer中。



　　然后会通过以下三种方式将innodb日志缓冲区的日志刷新到磁盘

　　Master Thread 每秒一次执行刷新Innodb\_log\_buffer到重做日志文件。

　　每个事务提交时会将重做日志刷新到重做日志文件。

　　当重做日志缓存可用空间 少于一半时，重做日志缓存被刷新到重做日志文件

　　由此可以看出，重做日志通过不止一种方式写入到磁盘，尤其是对于第一种方式，Innodb\_log\_buffer到重做日志文件是Master Thread线程的定时任务。

　　因此重做日志的写盘，并不一定是随着事务的提交才写入重做日志文件的，而是随着事务的开始，逐步开始的。

另外引用《MySQL技术内幕 Innodb 存储引擎》（page37）上的原话：

　　即使某个事务还没有提交，Innodb存储引擎仍然每秒会将重做日志缓存刷新到重做日志文件。

　　这一点是必须要知道的，因为这可以很好地解释再大的事务的提交（commit）的时间也是很短暂的。

**二、回滚日志（undo log）**

**作用：**

　　保证数据的原子性，保存了事务发生之前的数据的一个版本，可以用于回滚，同时可以提供多版本并发控制下的读（MVCC），也即非锁定读

**内容：**

　　逻辑格式的日志，在执行undo的时候，仅仅是将数据从逻辑上恢复至事务之前的状态，而不是从物理页面上操作实现的，这一点是不同于redo log的。

**什么时候产生：**

　　事务开始之前，将当前是的版本生成undo log，undo 也会产生 redo 来保证undo log的可靠性

**什么时候释放：**

　　当事务提交之后，undo log并不能立马被删除，而是放入待清理的链表，由purge线程判断是否由其他事务在使用undo段中表的上一个事务之前的版本信息，决定是否可以清理undo log的日志空间。

**对应的物理文件：**

　　MySQL5.6之前，undo表空间位于共享表空间的回滚段中，共享表空间的默认的名称是ibdata，位于数据文件目录中。

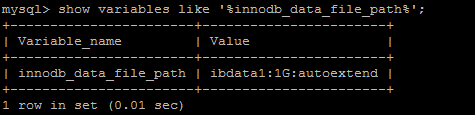
　　MySQL5.6之后，undo表空间可以配置成独立的文件，但是提前需要在配置文件中配置，完成数据库初始化后生效且不可改变undo log文件的个数

　　如果初始化数据库之前没有进行相关配置，那么就无法配置成独立的表空间了。

**关于MySQL5.7之后的独立undo 表空间配置参数如下：**

　　innodb\_undo\_directory = /data/un[dos](https://www.2cto.com/os/dos/)pace/ –undo独立表空间的存放目录 innodb\_undo\_logs = 128 –回滚段为128KB innodb\_undo\_tablespaces = 4 –指定有4个undo log文件

　　如果undo使用的共享表空间，这个共享表空间中又不仅仅是存储了undo的信息，共享表空间的默认为与MySQL的数据目录下面，其属性由参数innodb\_data\_file\_path配置。



**其他：**

　　undo是在事务开始之前保存的被修改数据的一个版本，产生undo日志的时候，同样会伴随类似于保护事务持久化机制的redolog的产生。

　　默认情况下undo文件是保持在共享表空间的，也即ibdatafile文件中，当数据库中发生一些大的事务性操作的时候，要生成大量的undo信息，全部保存在共享表空间中的。

　　因此共享表空间可能会变的很大，默认情况下，也就是undo 日志使用共享表空间的时候，被“撑大”的共享表空间是不会也不能自动收缩的。

　　因此，mysql5.7之后的“独立undo 表空间”的配置就显得很有必要了。

**三、二进制日志（binlog）：**

**作用：**

　　用于复制，在主从复制中，从库利用主库上的binlog进行重播，实现主从同步。

　　用于数据库的基于时间点的还原。

**内容：**

　　逻辑格式的日志，可以简单认为就是执行过的事务中的sql语句。

　　但又不完全是sql语句这么简单，而是包括了执行的sql语句（增删改）反向的信息，也就意味着delete对应着delete本身和其反向的insert；update对应着update执行前后的版本的信息；insert对应着delete和insert本身的信息。

　　在使用mysqlbinlog解析binlog之后一些都会真相大白。

　　因此可以基于binlog做到类似于oracle的闪回功能，其实都是依赖于binlog中的日志记录。

**什么时候产生：**

　　事务提交的时候，一次性将事务中的sql语句（一个事物可能对应多个sql语句）按照一定的格式记录到binlog中。

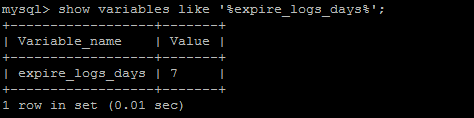
　　这里与redo log很明显的差异就是redo log并不一定是在事务提交的时候刷新到磁盘，redo log是在事务开始之后就开始逐步写入磁盘。

　　因此对于事务的提交，即便是较大的事务，提交（commit）都是很快的，但是在开启了bin\_log的情况下，对于较大事务的提交，可能会变得比较慢一些。

　　这是因为binlog是在事务提交的时候一次性写入的造成的，这些可以通过测试验证。

**什么时候释放：**

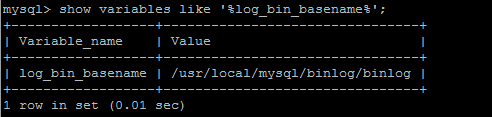
　　binlog的默认是保持时间由参数expire\_logs\_days配置，也就是说对于非活动的日志文件，在生成时间超过expire\_logs\_days配置的天数之后，会被自动删除。



**对应的物理文件：**

　　配置文件的路径为log\_bin\_basename，binlog日志文件按照指定大小，当日志文件达到指定的最大的大小之后，进行滚动更新，生成新的日志文件。

　　对于每个binlog日志文件，通过一个统一的index文件来组织。



**其他：**

　　二进制日志的作用之一是还原数据库的，这与redo log很类似，很多人混淆过，但是两者有本质的不同

**作用不同**：redo log是保证事务的持久性的，是事务层面的，binlog作为还原的功能，是数据库层面的（当然也可以精确到事务层面的），虽然都有还原的意思，但是其保护数据的层次是不一样的。

**内容不同**：redo log是物理日志，是数据页面的修改之后的物理记录，binlog是逻辑日志，可以简单认为记录的就是sql语句

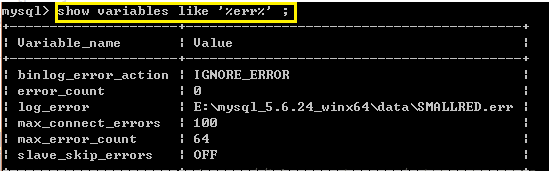
　　另外，两者日志产生的时间，可以释放的时间，在可释放的情况下清理机制，都是完全不同的。

　　恢复数据时候的效率，基于物理日志的redo log恢复数据的效率要高于语句逻辑日志的binlog

　　关于事务提交时，redo log和binlog的写入顺序，为了保证主从复制时候的主从一致（当然也包括使用binlog进行基于时间点还原的情况），是要严格一致的，MySQL通过两阶段提交过程来完成事务的一致性的，也即redo log和binlog的一致性的，理论上是先写redo log，再写binlog，两个日志都提交成功（刷入磁盘），事务才算真正的完成。

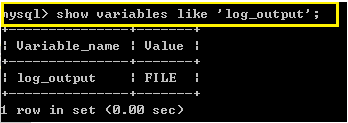
**四、错误日志**  
　　错误日志记录着mysqld启动和停止,以及服务器在运行过程中发生的错误的相关信息。在默认情况下，系统记录错误日志的功能是关闭的，错误信息被输出到标准错误输出。  
　　指定日志路径两种方法:  
　　　　编辑my.cnf 写入 log-error=[path]  
　　　　通过命令参数错误日志 mysqld\_safe –user=mysql –log-error=[path] &

显示错误日志的命令（如下图所示）



**五、普通查询日志 general query log**

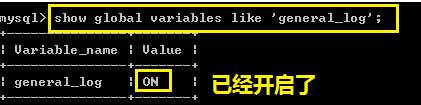
　　记录了服务器接收到的每一个查询或是命令，无论这些查询或是命令是否正确甚至是否包含语法错误，general log 都会将其记录下来 ，记录的格式为 {Time ，Id ，Command，Argument }。也正因为mysql服务器需要不断地记录日志，开启General log会产生不小的系统开销。 因此，Mysql默认是把General log关闭的。

查看日志的存放方式：show variables like ‘log\_output’;  


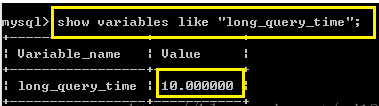
　　如果设置mysql> set global log\_output=’table’ 的话，则日志结果会记录到名为gengera\_log的表中，这表的默认引擎都是CSV  
　　如果设置表数据到文件set global log\_output=file;  
　　设置general log的日志文件路径：  
　　　　set global general\_log\_file=’/tmp/general.log’;  
　　　　开启general log： set global general\_log=on;  
　　　　关闭general log： set global general\_log=off;



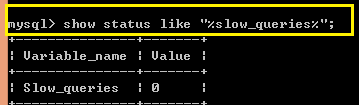
然后在用：show global variables like ‘general\_log’



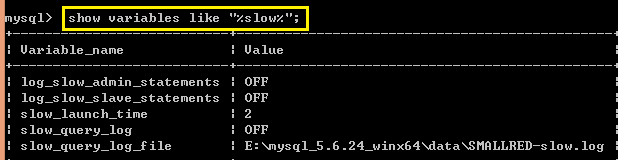
**六、慢查询日志**  
　　慢日志记录执行时间过长和没有使用索引的查询语句，报错select、update、delete以及insert语句，慢日志只会记录执行成功的语句。  
　　1. 查看慢查询时间：   
　　show variables like “long\_query\_time”;默认10s



　　2. 查看慢查询配置情况：   
　　show status like “%slow\_queries%”;



　　3. 查看慢查询日志路径：   
　　show variables like “%slow%”;



4. 开启慢日志



　　查看已经开启：

