# MySQL数据库中的日志文件

前言：MySQL日志文件

MySQL生存或相关联的日志文件种类繁多，下面是是MySQL数据库服务相关的几类日志文件：

1、错误日志（error log）：记录MySQL服务进程mysqld在启动/关闭或运行过程中遇到的错误信息；

1. 查询日志（query log）：又可分成两类：

①普通查询日志（general query log）：记录客户端连接信息和执行的SQL语句信息；

②慢查询日志（slow query log）；记录执行时间超出指定值（long\_query\_time）的SQL语句；

3、二进制日志（binary log）：记录数据被修改的相关信息；

1. 错误日志（Erroe Log）

数据库配置文件my.cnf

vim /data/3306/my.cnf

[mysqld\_safe]

log-error=/data/3306/mysql\_oldboy3306.err

pid-file=/data/3306/mysqld.pid

1. 查询日志

## 2.1 普通查询日志（The General Query Log）

**— —工作环境中不开启普通查询日志**

2．1．1 登录到数据库查看是否开启

mysql> show variables like "%\_log%";

+-----------------------------------------+-------------------------------+

| Variable\_name | Value |

| general\_log  **| OFF**  |

| general\_log\_file **| /data/3306/data/db02.log**  |

2．1．2 开启普通查询日志

普通查询日志参数：

mysql> **set global general\_log = ON;**

Query OK, 0 rows affected (0.01 sec)

2．1．3 再查看结果

mysql> show variables like "%\_log%";

+-----------------------------------------+-------------------------------+

| Variable\_name | Value |

| general\_log  **| ON**  |

| general\_log\_file **| /data/3306/data/db02.log**  |

2．1．4 退出数据库查看是否生成日志文件

ll /data/3306/data/ |grep "db02.log"

-rw-rw---- 1 mysql mysql 275 Dec 21 10:26 db02.log

2．1．5 查看日志内容

cat /data/3306/data/db02.log

/application/mysql-5.5.32/bin/mysqld, Version: 5.5.32-log (Source distribution). started with:

Tcp port: 3306 Unix socket: /data/3306/mysql.sock

Time Id Command Argument

151221 10:22:38 5 Query  **show variables like "%\_log%" ##<===这里之前操作命令**

151221 10:26:22 5 Quit

**###普通查询日志生成！**

## 2.2 慢查询日志（The Slow Query Log）

2．2．1 修改配置文件，添加参数

1、记录执行时间超出指定值，没有走索引的

long\_query\_time = 2  ***##多长时间记录***

log-slow-queries = /data/3306/slow.log

log\_queries\_not\_using\_indexes ***##没有使用索引的语句***

**方便缩小目标，把有问题的找出来**

**##切割日志，创建定时任务，mv每天创建一个慢查询日志**



2．2．2 登录到数据库

mysql> show variables like 'log\_slow%';

+------------------+-------+

| Variable\_name | Value |

+------------------+-------+

| log\_slow\_queries | OFF |

+------------------+-------+

1 row in set (0.00 sec)

2．2．3 开启慢查询日志文件参数：

mysql> set global log\_slow\_queries = ON;

Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> show variables like 'log\_slow%';

+------------------+-------+

| Variable\_name | Value |

+------------------+-------+

| log\_slow\_queries | ON |

+------------------+-------+

1 row in set (0.00 sec)

## 2.3 配置查询日志输出路径

log-slow-queries = /data/3306/slow.log

## 2.4 系统日志表的特点：

简单方便

三、二进制日志（Binary Log）

## 3.1 认识二进制日志格式

## 3.2 设置二进制日志格式

数据库配置文件my.cnf

log-bin = /data/3306/mysql-bin

登录到数据库查询

mysql> show variables like "log\_bin%";

+---------------------------------+-------+

| Variable\_name | Value |

+---------------------------------+-------+

| log\_bin | ON | **##记录binlog**

| log\_bin\_trust\_function\_creators | OFF | **##临时不记录binlog**

+---------------------------------+-------+

2 rows in set (0.00 sec)

## 3.3 混合二进制日志记录

## mysql数据中对象修改的二进制日志记录

四、Binlog日志的三种模式

## 4．1 binlog三种日志模式介绍

4．1．1 Statement Level模式

每一条会修改数据的sql都会记录到master的bin-log中。Slave在复制的时候sql进程会解析成和原理master段执行过 的相同的sql来在执行。

优点：statement level下的优点首先就是解决了row level下的全额点，不需要记录每一行数据的变化，减少bin-log日志量，节约IO，提高性能。因为它只需要记录在Master上所执行的语句的细节，以及执行语句时候的上下文的信息。

缺点：由于它是记录的执行语句，所以为了然这些语句在slave段也能正确执行，那么它还必须记录每条语句在执行的时候的一些相关信息，也就是上下文信息，以保证所有语句在slave端被执行的时候能够得到和在master端执行时候相同的结果。另外就是，由于MySQL现在发展比较快，很多的新功能不断的加入，使MySQL的复制遇到了不小的挑战，自然复制的还好涉及到越复杂的你，bug也就越容易出现。在statement level下目前也就发现的就有不少情况会造成MySQL的复制出现问题，主要是修改数据的时候使用了某些特定的函数或者功能的时候会出现，比如：sleep()函数在有些版本中就不能正确复制，子存储过程中使用了last\_insert\_id()函数，可能会使slave和master上得到不一致的id等等。由于row level是基于每一行来记录的变化，所以不会出现类似的问题。

4．1．2 Row Level模式

日志中会记录成每一行数据被修改的形式，然后在slave端在对相同的数据进行修改。

优点：在row level模式下，bin-log中科院不记录执行的sql语句的上下文相关的信息，仅仅只需要记录那一跳记录被修改了，修改成什么样了。所以row level的日志内容会非常清楚的**记录下每一行数据修改的细节**，非常容易理解。而且不会某些特定情况下的存储过程，或function，以及trigger的调用和触发无法被正确复制的问题。

缺点：row level下，所有的执行的语句当记录到日志中的时候，都将以每行记录的修改来记录，这样可能会产生大量的日志内容，比如有这样一条update语句：update product setowner\_member\_id=‘b’where owner\_member\_id=’a’，执行之后，日志中记录的不是这条update语句所对应的事件（MySQL以事件的形式来记录bin-log日志），而死这条语句所更新的每一条记录的变化情况，这样就记录成很多条记录被更新的很多个事件。自然，bin-log日志的量就会很大。尤其是当执行alter table之类的语句的时候，产生的日志量是惊人的。因为MySQL对于alter table之类的表结构变更语句的处理方式是整个表的每一条记录都需要变动，实际上技术重建了整个表。那么该表的每一条记录都会被记录到日志中。

4．1．3 Mixed模式

实际上技术前两种模式的结合。在Mixed模式下，mysql会根据执行的每一条具体的sql语句来区分对待记录的日志形式，也就是在Statement和Row之间选择一种。新版本中的Statement level还是和以前一样，仅仅记录执行的语句。而新版本的MySQL中对row level模式也被做了优化，并不是所有的修改都会以row level来记录，像遇到表架构变更的时候就会以Statement模式来记录，如果sql语句确实就是update或者delete等修改数据的语句，那么还是会记录所有行的变更。

## 4．2 调整binlog日志模式方法

4．2．1在配置文件中修改方法

默认：

mysql> show variables like '%binlog\_format%';

+---------------+-----------+

| Variable\_name | Value |

+---------------+-----------+

| binlog\_format | STATEMENT |

+---------------+-----------+

1 row in set (0.00 sec)

4．2．2在配置文件中参数如下：

log-bin=mysql-bin

#binlog\_format=”STATEMENT”

#binlog\_format=”ROW”

binlog\_format=”MIXED”

4．2．3在线修改立即生效方法

**运行时在线修；**

mysql> set session binlog\_format = 'STATEMENT'; ###session当前生效

mysql> set session binlog\_format = 'ROW';

mysql> set session binlog\_format = 'MIXED';

**全局生效**

mysql> set global binlog\_format = 'STATEMENT';

mysql> set global binlog\_format = 'ROW';

mysql> set global binlog\_format = 'MIXED';

mysql> show variables like '%binlog\_format%';

+---------------+-------+

| Variable\_name | Value |

+---------------+-------+

| binlog\_format | ROW |

+---------------+-------+

1 row in set (0.00 sec)

**不重启永久修改立即生效方法**

配置文件my.cnf默认就是混合默认mixed

cat /application/mysql/support-files/my-small.cnf |grep "binlog\_format"

#binlog\_format=mixed

登录mysql命令操作使其永久生效

mysql> set global binlog\_format = 'STATEMENT';

mysql> quit;

Bye

mysql -uroot -poldboy123 -S /data/3306/mysql.sock

mysql> show variables like '%binlog\_format%';

+---------------+-------+

| Variable\_name | Value |

+---------------+-------+

| binlog\_format | STATEMENT |

+---------------+-------+

1 row in set (0.00 sec)

mysql> set global binlog\_format = 'MIXED';

mysql> quit;

Bye

mysql -uroot -poldboy123 -S /data/3306/mysql.sock

mysql> show variables like '%binlog\_format%';

+---------------+-------+

| Variable\_name | Value |

+---------------+-------+

| binlog\_format | MIXED |

+---------------+-------+

1 row in set (0.00 sec)