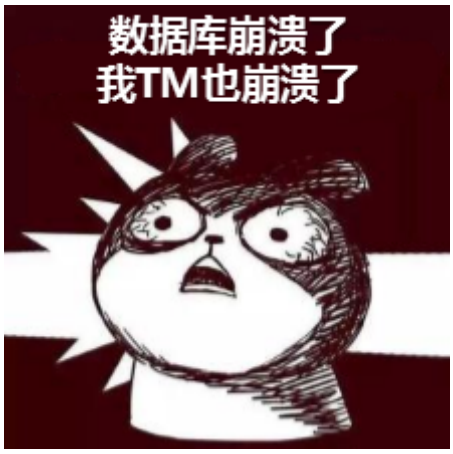
二进制日志相关概念

我们先聊聊binlog是怎样用于时间点恢复的。

假如我们每天晚上12点进行一次数据库备份，此处不考虑数据量，备份时间等其他因素，那么在本次备份完成后到下次备份开始前的这段时间段中，如果数据库服务崩溃了，我们应该怎样恢复呢?



如果我们只依靠上一次的数据库备份进行恢复，那么我们最多只能恢复到上一个12点时的数据，但是12点以后的数据则会丢失，所以，我们还需要依靠二进制日志（binlog），我们先用上一次的备份将数据库恢复至最近一次12点时的样子，再利用binlog，将12点之后的所有操作"重放"一遍，由于上次备份之时到数据库崩溃之时之间的所有操作完完全全的重新执行了一遍，所以我们可以将数据库中的数据恢复至崩溃之时的样子，而不至于丢失数据，这就是binlog用于恢复时的作用，如果你使用过oracle，那么你肯定会认为，mysql中的binlog与oracle中的归档日志特别像，其实它们存在的目的都是差不多的。

我们对binlog已经有了初步的概念，我们已经知道，binlog记录了所有的修改了数据库的语句，那么，我们来想一个问题，假设，我们在执行某条修改数据库的语句时，用到了user()函数，那么当我们执行这条语句时，语句根据当前用户的信息修改了数据库，然后这条语句将被记录到binlog中，但是，语句被记录在binlog中时，并没有记录当前用户的信息，而是记录了"user()"这个函数本身的几个字符，当我们根据binlog再次重放这条语句时，如果重放这条语句的用户与这条语句被记录时所使用的用户不同，那么语句执行后的结果就不同，这样就导致根据binlog重放操作时，并不能得到与我们预想的完全一致的数据，这种情况在数据恢复时是不允许的，在主从复制时也不是我们期望看到的，所以，为了能够完全的还原日志被记录时的操作，我们应该记录对应语句到底修改了哪些行，并且记录对应语句对这些行进行了哪些修改，只有这样，才能保证我们在重放binlog时，执行的操作与记录日志时的操作是完全一致的，这样是安全了，但是，我们设想一下，如果我们执行了一条update语句，这条update语句涉及到10000行数据的修改，那么，我们就需要将这条sql涉及到的10000行数据修改都记录到binlog中，以便重放二进制日志时，能够还原当时的操作，这样与只记录一条update语句来说，记录的信息量就大的多了，这样想想，这两种记录方式还是各有优势的。当然，到底是让binlog以记录语句的模式进行记录，还是以记录数据修改的模式进行记录，这些都是数据库管理员可以决定的，我们可以设置mysql通过哪种模式记录binlog，mysql中，有三种模式，statement、row、mixed，我们来总结一下binlog的这三种记录模式。

二进制日志有3种记录方式，三种方式如下：

statement模式：记录对数据库做出修改的语句，比如，update A set test='test',如果使用statement模式，那么这条update语句将会被记录到二进制日志中，使用statement模式时，优点是binlog日志量少，IO压力小，性能较高，缺点是为了能够尽量的完全一致的还原操作，除了记录语句本身以外，可能还需要记录一些相关的信息，而且，在使用一些特定的函数时，并不能保证恢复操作与记录时完全一

致。

row模式：记录对数据库做出修改的语句所影响到的数据行以及这些行的修改，比如，update A set test='test'，如果使用row模式，那么这条update语句所影响到的行所对应的修改，将会记录到binlog中，比如，A表中有1000条数据，那么当执行这条update语句以后，由于1000条数据都会被修改，所以会有1000行数据被记录到二进制日志中，以及它们是怎样被修改的，使用row模式时，优点是能够完全的还原或者复制日志被记录时的操作，缺点是记录日志量较大，IO压力大，性能消耗较大。

mixed模式：混合使用上述两种模式，一般的语句使用statment方式进行保存，如果遇到一些特殊的函数，则使用row模式进行记录，这种记录方式被称之为mixed，看上去这种方式似乎比较美好，但是在生产环境中，为了保险起见，一般会使用row模式。

管理员可以使用binlog_format变量设置二进制日志的记录方式，为了使配置永久生效，我们可以在my.cnf配置文件中加入如下配置。

binlog_format=row

上述配置表示使用row模式记录binlog

看完了上述概念以后，是不是已经对二进制日志文件有了一定的了解呢？趁热打铁，我们继续聊二进制日志，先把理论了解完了，再动手实践。

二进制日志文件，顾名思义，它是二进制的，所以我们不能直接使用cat命令进行查看，而是需要通过一些别的命令查看其内容，而且，二进制日志文件，有"事件"和"位置"的概念，什么是事件呢？通俗的讲，我们可以把binlog中的每一条记录当做一个"事件"，因为binlog记录了所有对数据库进行的修改，所以，我们可以认为，数据库的修改被记录到二进制日志中，这些记录每一条都可以理解为一个"事件"，由于二进制日志文件是二进制的，所以，我们可以把整个二进制文件想象成一个字节序列，假设，二进制日志文件刚开始是空的，从第1个字节开始记录，假设记录第一个"事件"（第一条记录），需要15个字节，那么第一个事件的开始"位置"就是1，结束"位置"就是15，由于前15个字节已经被第一个事件占用，那么当我们想要通过二进制日志记录第二个事件时，则需要从第15个字节向后开始记录，假设记录第二个"事件"需要20个字节，那么第二个事件在binlog中的起始"位置"就是15，结束"位置"就是35，以此类推，这就是二进制日志中，"事件"与"位置"的概念，"事件"被称为events，"位置"被称为position，如果你现在还不能很清晰的理解他们，没有关系，当我们动手实践的时候，你自然会明白。

二进制日志相关参数

好了，理论理解的差不多了，我们来看看与二进制日志有关的一些参数，有了之前的理论基础，再来了解他们，应该很容易了。

log_bin : 此变量用于控制是否开启二进制日志，而且这是一个只读变量，什么意思呢？咱们慢慢聊，默认情况下，当我们启动数据库以后，在当前数据库连接中查看此变量的值，此变量值可能为OFF，表示不记录二进制日志，如果想要记录二进制日志，只需将此值设置为二进制日志的文件名即可，但是需要注意的是，我们无法在当前会话中使用set命令设置log_bin的值，因为它是一个只读变量，我们只能通过修改my.cnf的方式，设置log_bin的值，假设，我们编辑my.cnf文件，设置log_bin的值为mybinlog，那么，在mysql的数据目录中，将会自动生成一个以mybinlog为文件名前缀的二进制日志文件，如果想要再次禁用binlog，只需要将log_bin这一行配置从my.cnf文件中注释即可，或者将其删除，重启mysql服务后，再次查看log_bin的值，其值为OFF，注意，不要直接在my.cnf文件中将log_bin的值设置为ON或者OFF，如果这样做，你将会看到以ON或者OFF为文件名前缀的二进制日志文件。换句话说，如果my.cnf配置文件中没有log_bin的配置，则表示未开启二进制日志，如果my.cnf中存在log_bin的配置，那么则表示开启了二进制日志，同时二进制日志文件的名称将会以log_bin对应的值为文件名前缀，同时，二进制日志文件的后缀名会进行自动编号，每次日志滚动后，后缀名编号自动加1。

sql_log_bin：此变量用于标识当前会话中的操作是否会被记录于二进制日志，此变量值设置为ON，则表示在当前数据库连接中，对数据库进行修改的语句将会被记录到binlog中，此变量值设置为OFF，则表示在当前数据库连接中，对数据库进行的修改的语句将不会被记录到binlog中，在主从复制结构中，这些语句由于没有被binlog记录，所以也不会同步到从节点中。换句话说，即使在my.cnf配置文件中设置了log_bin的值，当前会话中，如果sql_log_bin的值设置为OFF，当前会话的操作也不会记录在二进制日志中。而且需要注意的是，sql_log_bin是一个会话界别的变量，只能在当前会话中使用set sql_log_bin命令进行设置，不能使用set global sql_log_bin命令进行设置，因为它是会话级别的变量，而且，sql_log_bin也不能配置在my.cnf文件中，否则可能会无法启动mysql。

binlog_format： 此变量值得含义上文已经解释过，此变量的值决定了二进制日志的记录方式，此变量的值可以设置为statement、row、mixed，分别表示以语句的形式记录二进制日志，以数据修改的形式记录二进制日志，以混合的方式记录二进制日志，安全保险起见，推荐使用row的方式记录。

max_binlog_size：设置单个二进制日志文件的最大大小，以字节为单位，超过此值大小，则二进制日志文件会自动滚动，比如设置为500M为524288000。

sync_binlog：还记的我们总结的事务日志的相关文章吗？当我们把innodb_flush_log_at_trx_commit设置为1的时候，表示事务提交时，事务日志立马从内存刷写到磁盘中的事务日志文件中，而sync_binlog对于二进制日志的作用，就像innodb_flush_log_at_trx_commit对于事务日志的作用，由于二进制日志一开始存在于内存（binlog_cache）中，如果将sync_binlog设置为1，则表示每1次事务提交之后，都会立即将内存中的二进制日志立即同步到磁盘中的二进制日志文件中，如果将sync_binlog设置为0，则表示当事务提交之后，mysql不会立即将内存中的binlog刷写到磁盘中的binlog日志文件中，而是由文件系统决定什么时候刷写，这可能取决于文件系统的缓存机制，当此值设置为0时，一旦操作系统宕机，那么将丢失未从内存中同步到磁盘中的binlog，所以，当此值设置为0时，安全性最差，但是性能最高，当此值设置为1时，安全性最高，性能最差，除了将此值设置为0或1，还能设置为N，假设将此值设置为3，则表示每3次事务提交后，将binlog从内存刷写到磁盘一次，值设置的越大，有可能丢失的日志数据将会越多，当然，性能会越好，在追求安全的情况下，推荐设置为1，但是听说，此值设置为0和设置为1时在性能上的差距还是比较明显的，如果设置为0或N，最好为操作系统准备带有备用电源的缓存。

实践

说了这么多理论，我们来动手实践一下，在实践中，在回过头来看刚才的理论，会更加明了。

默认情况下，二进制日志可能未开启，查询如下

```
www.zsythink.net> show variables like "%log_bin";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | OFF   |
| sql_log_bin   | ON    |
+---------------+-------+
2 rows in set (0.00 sec)
```


我们可以手动修改my.cnf文件，在[mysqld]配置段中加入如下配置，表示开启binlog，并且设置二进制日志文件名前缀为mybinlog。

```
log_bin=mybinlog
```

重启mysql服务后，再次查看log_bin的值，可以看出，二进制日志功能已经开启。

```
www.zsythink.net> show variables like "%log_bin";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
| sql_log_bin   | ON    |
+---------------+-------+
2 rows in set (0.00 sec)
```


查看对应的数据文件目录，发现其中已经存在了以mybinlog开头的二进制日志文件，默认编号从000001开始，而且能看到一个名为mybinlog.index的文件，这个文件的作用我们一会再说。

```
[www.zsythink.net]# pwd
/var/lib/mysql
[www.zsythink.net]# ll mybin*
-rw-rw----. 1 mysql mysql 245 Mar  3 16:49 mybinlog.000001
-rw-rw----. 1 mysql mysql  18 Mar  3 16:49 mybinlog.index
[www.zsythink.net]#
```


回到mysql的命令行下，使用如下两条命令中的任何一条，均可在mysql中查看二进制日志的文件列表

show binary logs;

show master logs;

```
www.zsythink.net> show binary logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mybinlog.000001 |       245 |
+------------------+-----------+
1 row in set (0.00 sec)

www.zsythink.net> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mybinlog.000001 |       245 |
+------------------+-----------+
1 row in set (0.00 sec)

www.zsythink.n
```


可以看到，目前，mysql中只有一个二进制日志，日志文件名为mybinlog.000001，文件大小为245个字节。

我们知道，二进制日志记录了对数据库所作出的修改，那么，我们执行一些sql语句，修改一下数据，看看这些语句会不会被记录到二进制日志中。

首先，我们查看一下二进制日志的记录模式，从下图可以看出，默认的记录模式为statement，那么二进制日志会记录被执行的sql语句。

```
www.zsythink.net> show variables like "%binlog_format%";
+---------------+-----------+
| Variable_name | Value     |
+---------------+-----------+
| binlog_format | STATEMENT |
+---------------+-----------+
1 row in set (0.00 sec)
```


现在，我们来执行一些sql，测试一下，看看会不会被二进制日志记录，我们知道，二进制日志只会记录对数据库进行修改的语句，select语句时不会被记录的，所以，我们先执行一条select语句，执行完毕后，再次查看二进制日志的大小，看看其大小会不会发生变化。

```
www.zsythink.net> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mybinlog.000001 |       245 |
+------------------+-----------+
1 row in set (0.00 sec)

www.zsythink.net> select * from zsythink.a;
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
4 rows in set (0.04 sec)

www.zsythink.net> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mybinlog.000001 |       245 |
+------------------+-----------+
1 row in set (0.00 sec)

www.zsythink.net>
```


如上图所示，执行select语句后，二进制日志的大小并没有发生变化，那么我们执行一条delete语句试试。

```
www.zsythink.net> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mybinlog.000001 |       245 |
+------------------+-----------+
1 row in set (0.00 sec)

www.zsythink.net> delete from a where id in (3,4);
Query OK, 2 rows affected (0.17 sec)

www.zsythink.net> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mybinlog.000001 |       442 |
+------------------+-----------+
1 row in set (0.00 sec)
```


如上图所示，二进制日志的大小已经改变，当我们执行了delete语句以后，二进制日志文件大小从245个字节变成了442个字节，应该是已经记录了刚才的语句了，为了确定binlog已经记录了刚才的语句，我们来查看一下二进制日志的内容。

我们可以使用如下语句，查看二进制日志文件的内容。

show binlog events in '二进制日志文件名';

示例如下

```
www.zsythink.net> show binlog events in 'mybinlog.000001';
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------------+
| Log_name        | Pos | Event_type  | Server_id | End_log_pos | Info                                          |
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------------+
| mybinlog.000001 |   4 | Format_desc |         1 |         245 | Server ver: 5.5.44-MariaDB-log, Binlog ver: 4 |
| mybinlog.000001 | 245 | Query       |         1 |         317 | BEGIN                                         |
| mybinlog.000001 | 317 | Query       |         1 |         415 | use `zsythink`; delete from a where id in (3,4) |
| mybinlog.000001 | 415 | Xid         |         1 |         442 | COMMIT /* xid=16 */                           |
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------------+
4 rows in set (0.00 sec)
```


从上图可以看出，刚才的删除语句已经被记录到了mybinlog.000001这个二进制日志中，还记得我们执行了delete语句以后，二进制日志文件的大小从245个字节变为了442个字节吗？从上述binlog可以看出，当我们执行了delete语句以后，mysql把这条语句作为了一个单语句事务进行了提交，这是因为我们的autocommit参数设置为了ON，具体原因我们在总结"事务"时已经结束过，此处我们大概描述一下，当我们开启自动提交以后，mysql会把每条sql当做一个单语句事务进行自动提交，所以，在binlog中，我们看到，我们执行了一条delete语句，但是binlog中却自动把这条语句拆分成了3个"事件",事务开始"事件"，delete语句执行的"事件"，事务提交"事件"，而第一个事件的开始"位置"就是245，第三个事件的结束"位置"就是442，这两个数字是不是很眼熟，没错，就是我们执行delete语句前后，二进制日志文件的大小，我们可以把这三个事件看做一个整体，它们之所以被记录到二进制日志中，就是因为我们刚才执行了那条delete语句，而且，仔细看delete语句对应的事件，可以发现，我们只是执行了delete语句，但是事件中自动添加了"use zsythink",这是为了保险起见，确定重放这些语句时不会出现偏差而产生的。上图中，每一条记录都是一个"事件"（event），每一个事件都有自己的"开始位置"（start position）和"结束位置"（end position），这就是我们在理论部分总结的"事件"与"位置"。

其实，我们还能从指定的位置查看二进制日志，示例如下

```
www.zsythink.net> show binlog events in 'mybinlog.000001' from 317;
+-----------------+-----+------------+-----------+-------------+-----------------------------------------------+
| Log_name        | Pos | Event_type | Server_id | End_log_pos | Info                                          |
+-----------------+-----+------------+-----------+-------------+-----------------------------------------------+
| mybinlog.000001 | 317 | Query      |         1 |         415 | use `zsythink`; delete from a where id in (3,4) |
| mybinlog.000001 | 415 | Xid        |         1 |         442 | COMMIT /* xid=16 */                           |
```


也能限制显示出的记录数量，如只显示二进制日志中的前3行，示例如下，

```
www.zsythink.net> show binlog events in 'mybinlog.000001' limit 3;
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------------+
| Log_name        | Pos | Event_type  | Server_id | End_log_pos | Info                                          |
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------------+
| mybinlog.000001 |   4 | Format_desc |         1 |         245 | Server ver: 5.5.44-MariaDB-log, Binlog ver: 4 |
| mybinlog.000001 | 245 | Query       |         1 |         317 | BEGIN                                         |
| mybinlog.000001 | 317 | Query       |         1 |         415 | use `zsythink`; delete from a where id in (3,4) |
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------------+
3 rows in set (0.00 sec)
```


还能够指定从第几行之后开始，显示几行，例如，从第1行之后开始显示，显示之后的2行

```
www.zsythink.net> show binlog events in 'mybinlog.000001' limit 1,2;
+-----------------+-----+------------+-----------+-------------+-----------------------------------------------+
| Log_name        | Pos | Event_type | Server_id | End_log_pos | Info                                          |
+-----------------+-----+------------+-----------+-------------+-----------------------------------------------+
| mybinlog.000001 | 245 | Query      |         1 |         317 | BEGIN                                         |
| mybinlog.000001 | 317 | Query      |         1 |         415 | use `zsythink`; delete from a where id in (3,4) |
+-----------------+-----+------------+-----------+-------------+-----------------------------------------------+
2 rows in set (0.01 sec)
```


或者将上述示例结合在一起使用，比如从245位置开始，从245位置之后的第1行开始显示，显示2行。

```
www.zsythink.net> show binlog events in 'mybinlog.000001' from 245 limit 1,2;
+-----------------+-----+------------+-----------+-------------+-----------------------------------------------+
| Log_name        | Pos | Event_type | Server_id | End_log_pos | Info                                          |
+-----------------+-----+------------+-----------+-------------+-----------------------------------------------+
| mybinlog.000001 | 317 | Query      |         1 |         415 | use `zsythink`; delete from a where id in (3,4) |
| mybinlog.000001 | 415 | Xid        |         1 |         442 | COMMIT /* xid=16 */                           |
+-----------------+-----+------------+-----------+-------------+-----------------------------------------------+
2 rows in set (0.00 sec)
```


我们能够使用如下命令，查看当前在使用哪个二进制日志，以及二进制日志记录到哪一个位置了

show master status;

```
www.zsythink.net> show master status;
+-----------------+----------+--------------+------------------+
| File            | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----------------+----------+--------------+------------------+
| mybinlog.000001 |      442 |              |                  |
+-----------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```


如上图所示，目前使用的二进制日志是mybinlog.000001,对应记录的位置为442，我们之前说过，二进制日志是会滚动的，当单个二进制日志达到指定的大小，则会滚动，当然，我们也可以手动的进行日志滚动，使用flush logs命令即可滚动二进制日志，重启mysql服务时二进制日志也会自动滚动。那么，我们现在来手动滚动一下二进制日志。

如下图所示，使用flush logs命令手动滚动二进制日志以后，再次查看二进制日志的状态，已经自动切换使用最新的binlog了，二进制日志文件后缀名自动加一，变为了000002，再次查看二进制日志列表，已经存在两个二进制日志文件了。

```
www.zsythink.net> flush logs;
Query OK, 0 rows affected (0.16 sec)

www.zsythink.net> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mybinlog.000002 |      245 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)

www.zsythink.net> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mybinlog.000001 |       484 |
| mybinlog.000002 |       245 |
+------------------+-----------+
2 rows in set (0.00 sec)
```


那么我们到文件系统的数据文件路径中查看一下，如下图，已经存在两个二进制日志文件了，这时，我们查看一下mybinlog.index文件，这个文件不是二进制的，可以使用cat命令直接查看。

```
[www.zsythink.net]# pwd
/var/lib/mysql
[www.zsythink.net]# ll mybinlog*
-rw-rw---- 1 mysql mysql 484 Mar  4 10:53 mybinlog.000001
-rw-rw---- 1 mysql mysql 245 Mar  4 10:53 mybinlog.000002
-rw-rw---- 1 mysql mysql  36 Mar  4 10:53 mybinlog.index
[www.zsythink.net]# cat mybinlog.index
./mybinlog.000001
./mybinlog.000002
[www.zsythink.net]#
```


如上图所示，mybinlog.index中其实就是记录了有哪些二进制日志文件而已。

好了，回到mysql中，我们说过，通过sql_log_bin参数，能够控制当前数据库连接会话中的sql语句是否会被记录到binlog中，那么，我们来测试一下。

首先，查看一下当前的二进制日志文件中的内容，除了一条默认的记录以外，当前二进制日志文件中没有其他任何事件记录。

```
www.zsythink.net> show binlog events in 'mybinlog.000002';
+----------------+-----+-------------+-----------+-------------+---------------------------------------+
| Log_name       | Pos | Event_type  | Server_id | End_log_pos | Info                                  |
+----------------+-----+-------------+-----------+-------------+---------------------------------------+
| mybinlog.000002 |   4 | Format_desc |         1 |         245 | Server ver: 5.5.44-MariaDB-log, Binlog ver: 4 |
+----------------+-----+-------------+-----------+-------------+---------------------------------------+
1 row in set (0.00 sec)
```


那么此时，我们将当前会话中的sql_log_bin设置为OFF，然后执行一条插入语句。

```
www.zsythink.net> set sql_log_bin=OFF;
Query OK, 0 rows affected (0.01 sec)

www.zsythink.net> show variables like "%log_bin";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
| sql_log_bin   | OFF   |
+---------------+-------+
2 rows in set (0.00 sec)

www.zsythink.net> insert into zsythink.a values(3);
Query OK, 1 row affected (0.02
```


再次查看当前使用的二进制日志，发现并没有任何新的记录，当我们觉得当前会话中的某些语句没有被记录到二进制日志文件中的必要的时候，则可以将sql_log_bin设置为OFF。

```
www.zsythink.net> show binlog events in 'mybinlog.000002';
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------+
| Log_name        | Pos | Event_type  | Server_id | End_log_pos | Info                                    |
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------+
| mybinlog.000002 |   4 | Format_desc |         1 |         245 | Server ver: 5.5.44-MariaDB-log, Binlog ver: 4 |
+-----------------+-----+-------------+-----------+-------------+-----------------------------------------+
1 row in set (0.00 sec)
```


刚才的操作一直都是在statement模式下进行的实验，现在我们开启sql_log_bin,同时将二进制日志文件的记录模式改为row，再看看二进制日志的记录是什么样的。

修改my.cnf配置文件，将binlog_format设置为row。

```
log_bin=mybinlog
binlog_format=row
```

重启mysql服务后，查看binlog的记录模式，已经改为row

```
www.zsythink.net> show variables like "%binlog_format%";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.00 sec)
```


由于刚才已经重启了mysql服务，所以，binlog自动滚动了，当前使用的二进制日志文件为mybinlog.000003

```
www.zsythink.net> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mybinlog.000001 |       484 |
| mybinlog.000002 |       264 |
| mybinlog.000003 |       245 |
+------------------+-----------+
3 rows in set (0.00 sec)

www.zsythink.net> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mybinlog.000003 |      245 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```


现在，执行一条delete语句

```
www.zsythink.net> delete from a where id=2;
Query OK, 1 row affected (0.01 sec)
```

查看二进制日志，记录如下。

```
www.zsythink.net> show binlog events in 'mybinlog.000003';
+-----------------+-----+-------------+-----------+-------------+-------------------------------------------+
| Log_name        | Pos | Event_type  | Server_id | End_log_pos | Info                                      |
+-----------------+-----+-------------+-----------+-------------+-------------------------------------------+
| mybinlog.000003 |   4 | Format_desc |         1 |         245 | Server ver: 5.5.44-MariaDB-log, Binlog ver: 4 |
| mybinlog.000003 | 245 | Query       |         1 |         317 | BEGIN                                     |
| mybinlog.000003 | 317 | Table_map   |         1 |         361 | table_id: 33 (zsythink.a)                 |
| mybinlog.000003 | 361 | Delete_rows |         1 |         395 | table_id: 33 flags: STMT_END_F            |
| mybinlog.000003 | 395 | Xid         |         1 |         422 | COMMIT /* xid=19 */                       |
+-----------------+-----+-------------+-----------+-------------+-------------------------------------------+
5 rows in set (0.00 sec)
```


从上图中的记录可以看出，刚才的delete语句并没有被记录，而是被拆分成了两个事件，记录于了二进制日志文件中，而上图中的table_id，就是造成淘宝物流主从复制结构中主从数据不一致的原因，此处不是我们讨论的重点，有兴趣的同学可以在网上搜索如下标题。

淘宝物流MySQL slave复制数据丢失问题

那么此处，我们把刚才常用的能够查看binlog信息的语句总结一下：

查看二进制日志文件列表

show master logs;

show binary logs;

查看当前正在使用的二进制日志文件

show master status;

查看二进制日志文件中的事件（查看binlog内容），可以使用如下语句，如果不明白如下语句的含义，可以对照上文中的示例查看，此处只用于总结。

show binlog events

show binlog events in ‘mybinlog.000001’

show binlog events in ‘mybinlog.000001’from 245

show binlog events in ‘mybinlog.000001’limit 3

show binlog events in ‘mybinlog.000001’limit 2,5

show binlog events in ‘mybinlog.000001’from 245 limit 10

show binlog events in ‘mybinlog.000001’from 245 limit 4,20

mysqlbinlog命令

其实，除了在mysql提示符中使用show binlog events命令查看日志内容以外，还能在文件系统中使用mysqlbinlog命令查看对应的二进制日志，比如，查看mybinlog.000001，示例如下

```
[www.zsythink.net]# mysqlbinlog mybinlog.000001
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#170304 10:30:06 server id 1  end_log_pos 245   Start: binlog v 4, server v 5.5.44-MariaDB-log created
ROLLBACK/*!*/;
BINLOG '
ria6WA8BAAAA8QAAAPUAAAAAAAQANS41LjQ0LU1hcmlhREItbG9nAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAACuJrpYEzgNAAgAEgAEBAQEEgAA2QAEGggAAAAICAgCAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAALXHRjg==
'/*!*/;
# at 245
#170304 10:30:48 server id 1  end_log_pos 317   Query   thread_id=2     exec_time=0     error_code=0
SET TIMESTAMP=1488594648/*!*/;
SET @@session.pseudo_thread_id=2/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1, @@session
SET @@session.sql_mode=0/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C utf8 *//*!*/;
SET @@session.character_set_client=33,@@session.collation_connection=33,@@session.collation_server=33/
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
BEGIN
/*!*/;
# at 317
#170304 10:30:48 server id 1  end_log_pos 415   Query   thread_id=2     exec_time=0     error_code=0
use `zsythink`/*!*/;
SET TIMESTAMP=1488594648/*!*/;
delete from a where id in (3,4)
/*!*/;
# at 415
#170304 10:30:48 server id 1  end_log_pos 442   Xid = 16
COMMIT/*!*/;
# at 442
#170304 10:53:15 server id 1  end_log_pos 484   Rotate to mybinlog.000002  pos: 4
DELIMITER ;
# End of log file
ROLLBACK /* added by mysqlbinlog */;
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;
[www.zsythink.net]#
```


每个二进制日志文件都会有固定的开头内容，这些内容是一些版本信息和属性信息，如上图中的红色标注内容，我们不用在意它们，我们可能会查看的一般为上图中蓝色标注的内容。

我们也可以使用如下命令从指定位置开始查看二进制日志。

mysqlbinlog –start-position 415 mybinlog.000001

或者指定，从哪个位置开始，到那个位置结束，查看这之间的二进制日志文件。

mysqlbinlog –start-position 317 –stop-position 442 mybinlog.000001

还可以从指定的时间开始查看，比如，查看2017年3月4日10点40以后的日志。

mysqlbinlog –start-datetime “2017-3-4 10:40:00” mybinlog.000001;

当然，也可以指定结束时间，比如

mysqlbinlog –start-datetime “2017-3-4 10:40:00” –stop-datetime “2017-3-4 10:55:00” mybinlog.000001;