# 背景

二进制日志包含数据库的所有更改记录，包括数据和结构两方面。二进制日志不记录SELECT 或SHOW 等不修改数据的操作。运行带有二进制日志的服务器会带来轻微的性能影响。二进制日志能保证数据库出故障时数据是安全的。只有完整的事件或事务会被记录或回读。

为什么应该使用二进制日志？

● 复制：使用二进制日志，可以把对服务器所做的更改以流式方式传输到另一台服务器上。从（slave）服务器充当镜像副本，也可用于分配负载。接受写入的服务器称为主（master）服务器。

● 时间点恢复：假设你在星期日的00：00进行了备份，而数据库在星期日的08：00出现故障。使用备份可以恢复到周日00：00的状态；而使用二进制日志可以恢复到周日08：00的状态。

# 概述

Binlog它记录了所有的**DDL和DML**(除了数据查询语句)语句，以**事件（EVENT）**形式记录，还包含语句所执行的消耗的时间，MySQL的二进制日志是**事务安全型**的。

一般来说开启二进制日志大概会有1%的性能损耗（MySQL官方测试数据），虽然会降低性能，但是binlog可以用于主从复制（replication）和point-in-time的恢复，所以还是很有必要开启的。

参考：

<https://database.51cto.com/art/202009/626540.htm>

**查看：**

SHOW BINLOG EVENT IN ‘mysqld.00001’\G;

**配置：**

通过参数log-bin[=name]可以开启二进制日志，如果不指定name，默认二进制日志文件名为主机名，后缀名为二进制日志的序列号，所在路径为数据库所在目录（datadir）。

# 分类

## 二进制日志索引文件

## 二进制日志文件

二进制日志包括两类文件：二进制日志索引文件（文件名后缀为.index）用于记录所有的二进制文件，二进制日志文件（文件名后缀为.00000\*）记录数据库所有的DDL和DML(除了数据查询语句)语句事件。

# 日志格式

binlog有三种格式：Statement, Row和Mixed.

基于SQL语句的复制（statement-based replication, SBR）

基于行的复制（row-based replication, RBR）

混合模式复制（mixed-based replication, MBR）

## Statement

记录的是逻辑SQL，每一条修改操作的sql都会记录在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, 提高了性能。

**缺点：**无法完全保证slave节点与master节点数据完全一致。

由于记录的只是执行语句，为了这些语句能在slave上正确运行，因此还必须记录每条语句在执行的时候的一些相关信息，以保证所有语句能在slave得到和在master端执行的时候相同的结果。另外mysql的复制，像一些特定函数的功能，slave可与master上要保持一致会有很多相关问题。

相比row能节约多少性能与日志量，这个取决于应用的SQL情况，正常同一条记录修改或者插入row格式所产生的日志量还小于statement产生的日志量，但是考虑到如果带条件的update操作，以及整表删除，alter表等操作，row格式会产生大量日志，因此在考虑是否使用row格式日志时应该根据应用的实际情况，其所产生的日志量会增加多少，以及带来的IO性能问题。

## Row

5.1.5版本的MySQL才开始支持row level的复制，它不记录sql语句上下文相关信息，仅保存哪条记录被修改。

**优点：**binlog中可以不记录执行的sql语句的上下文相关的信息，仅需要记录那一条记录被修改成什么了。所以row的日志内容会非常清楚的记录下每一行数据修改的细节。而且不会出现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：**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。

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，像遇到表结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。

## Mixed

从5.1.8版本开始，MySQL提供了Mixed格式，实际上就是Statement与Row的结合。

在Mixed模式下，（默认情况下）一般的语句修改使用statment格式保存binlog，如一些函数，statement无法完成主从复制的操作，则采用row格式保存binlog，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。

会采用row格式的情况包括：

1、表的存储引擎为NDB，这时对表的DML操作都会以ROW格式记录；

2、使用了UUID()、USER()、CURRENT\_USER()、FOUND\_ROWS()、ROW\_COUNT()等不确定函数；

3、使用了INSERT DELAY语句；

4、使用了用户定义函数（UDF）；

5、使用了临时表（temporary table）。

注：不需要死记硬背这些情况会使用row，之所以不使用statement模式，就是因为简单的逻辑SQL无法实现数据的回放，比如使用UUID()，这个是随机的，无法保证每次结果都一样，为了保证主从复制数据一致性，则必须是记录数据的变化信息。

此外，binlog\_format参数还有对于存储引擎的限制：



在通常情况下，我们将参数binlog\_format设置为ROW，这可以为数据库的恢复和复制带来更好的可靠性。但是不能忽略的一点是，这会带来二进制文件大小的增加，这些语句的ROW格式可能需要更大的容量。而由于复制是采用传输二进制日志方式实现的，因此复制的网络开销也会增加。

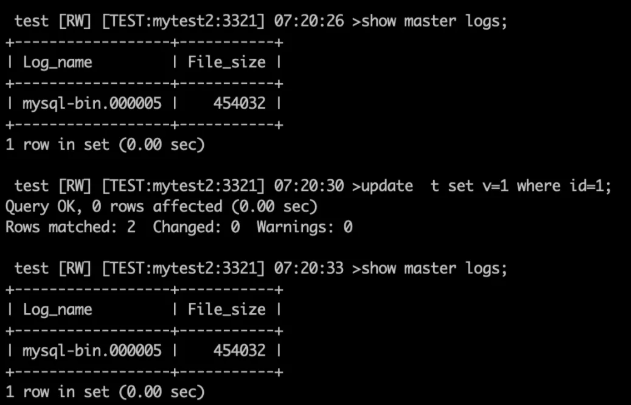
要查看二进制日志文件内容，必须通过MySQL提供的工具mysqlbinlog。对于STATEMENT格式的二进制日志文件，在使用mysqlbinlog后，看到的就是执行的逻辑SQL语句。但是，如果使用ROW格式记录，会发现mysqlbinlog的结果变得“不可读”，其实只要加上参数-v或-vv就能清楚地看到执行的具体信息了（-vv会比-v显示更新的类型）。

注：要区分行的格式和binlog的格式。

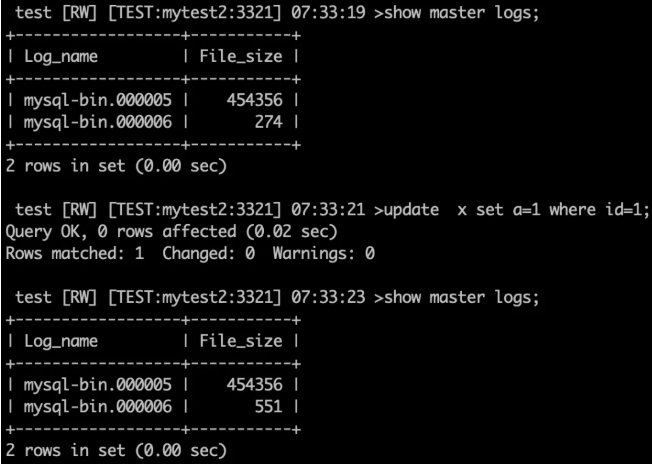
## 区别

**update字段为相同的值是否会记录binlog？**

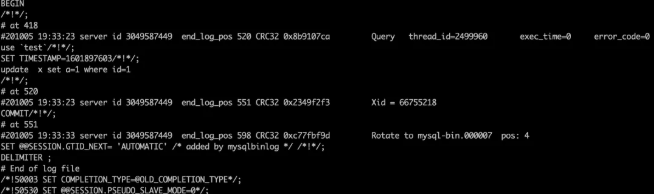
binlog\_format为ROW模式



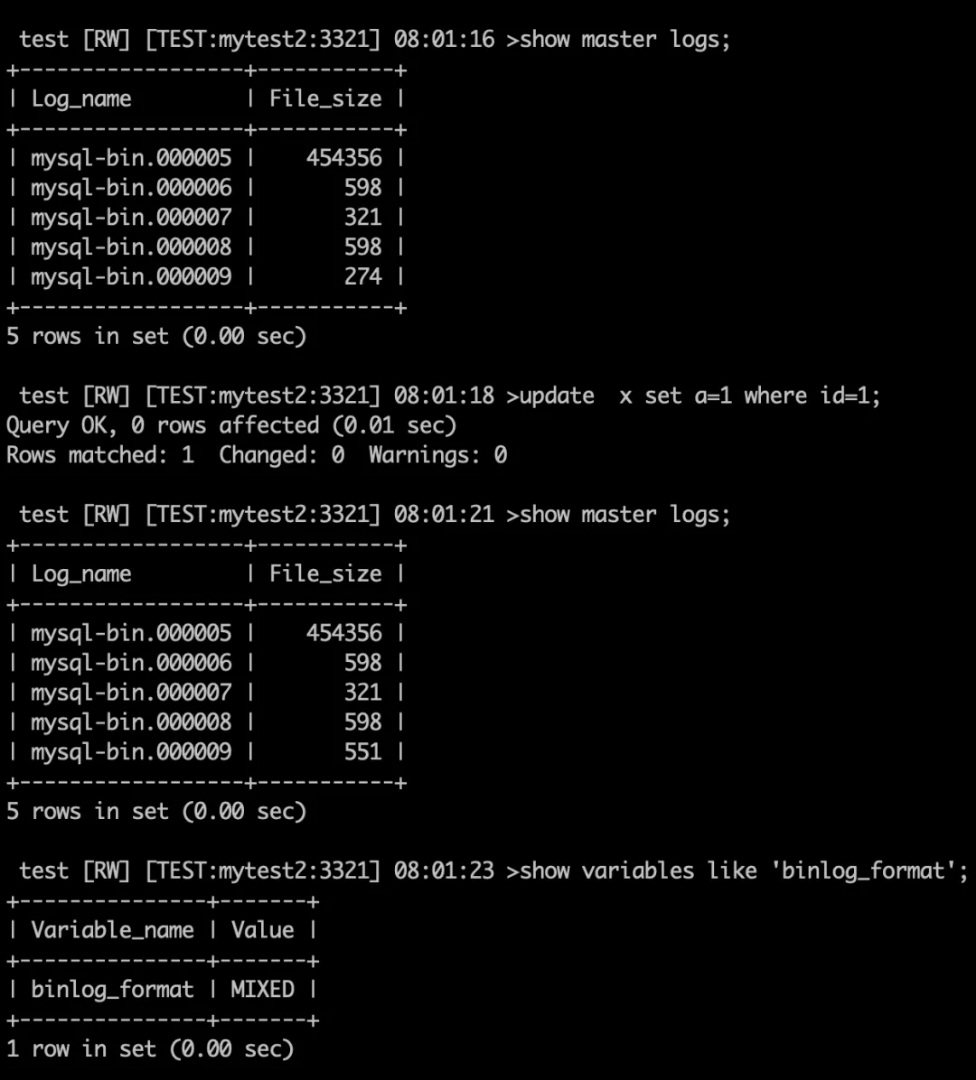
binlog\_format为STATEMENT模式



解析binlog内容，完整的记录了update语句。



binlog\_format为MIXED模式



当 row\_format 为mixed或者statement格式是，binlog 的大小发生改变，不管是否真的更新数据，MySQL都记录执行的sql 到binlog。

**总结：**

基于row模式时，server层匹配到要更新的记录，发现新值和旧值一致，不做更新，就直接返回，也不记录binlog。

基于 statement 或者 mixed格式，MySQL执行 update 语句，并把更新语句记录到binlog。

# 8.0新特性

## binlog压缩

### 概述

二进制日志（binlog）是MySQL日志结构中重要的部分；记录了数据的更改操作，用于数据恢复、数据复制以及审计。然而在众多实际场景中经常发生高并发引起binlog暴涨的问题将挂载点空间占满以及主从网络带宽成为瓶颈时主从延时过大。8.0.20版本推出binlog压缩功能，有效缓解甚至解决此类问题。

MySQL从8.0.20开始集成ZSTD算法，开启压缩功能后；以事务为单位进行压缩写入二进制日志文件，降低原文件占用的磁盘空间。压缩后的事务以压缩状态有效负载在复制流中发送到从库（MGR架构中为组member）或客户端（例如 mysqlbinlog）。

官网连接：<https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-20.html>

### 原理

1、开启压缩功能后，通过ZSTD算法对每个事务进行压缩，写入二进制日志。

2、新版本更改了libbinlogevents，新增Transaction\_payload\_event作为压缩后的事务表示形式。

class Transaction\_payload\_event : public Binary\_log\_event {

protected:

const char \*m\_payload;

uint64\_t m\_payload\_size;

transaction::compression::type m\_compression\_type;

uint64\_t m\_uncompressed\_size;

3、新增Transaction\_payload\_event编码器/解码器，用于实现对压缩事务的编码和解码。

namespace binary\_log {

namespace transaction {

namespace compression {

enum type {

/\* No compression. \*/

NONE = 0,

/\* ZSTD compression. \*/

ZSTD = 1,

};

4、在mysqlbinlog中设计和实现每个事务的解压缩和解码，读取出来的日志与未经压缩的原日志相同，并打印输出所用的压缩算法，事务形式，压缩大小和未压缩大小，作为注释。

#200505 16:24:24 server id 1166555110 end\_log\_pos 2123 CRC32 0x6add0216 Transaction\_Payload payload\_size=863 compression\_type=ZSTD uncompressed\_size=2184

# Start of compressed events!

5、从库（或MGR-member）在接收已压缩的binlog时识别Transaction\_payload\_event，不进行二次压缩或解码。以原本的压缩状态写入中继日志；保持压缩状态。回放日志的解码和解压缩过程由SQL线程负责。

总结日志压缩过程为：

1）单位事务需要提交并记录 binlog。

2）压缩编码器在缓存中通过ZSTD算法压缩以及编码该事务。

3）将缓存中压缩好的事务写入日志中，落盘。

日志读取过程为：

客户端工具（mysqlbinlog、sql 线程）对压缩日志进行解压缩、解码。解压出原本未压缩的日志进行读取或回放。

### 注意事项

1、压缩功能以事务为单位进行压缩，不支持非事务引擎。

2、仅支持对ROW 模式的binlog进行压缩。

3、目前仅支持ZSTD压缩算法，但是，底层设计是开放式的，因此后续官方可能会根据需要添加其他压缩算法（例如zlib或lz4）。

4、压缩动作是并行进行的，并且发生在binlog落盘之前的缓存步骤中。

5、压缩过程占用本机CPU及内存资源。在主从延迟的场景中，如果性能瓶颈时，网络带宽、压缩功能可以有效缓解主从延迟；但是如果性能瓶颈是本机自身处理能力，那么压缩功能反而可能加大主从延迟。

### 特性测试

MySQL版本：8.0.20

架构：一主一从半同步

测试方案：

1、搭建好 MySQL 8.0.20 的主从架构

2、主从上开启压缩功能、并设置压缩等级，默认为 3，随着压缩级别的增加，数据压缩率也会增加，但同时 CPU 及内存的资源消耗也将增加。

mysql> set binlog\_transaction\_compression=on;

mysql> set binlog\_transaction\_compression\_level\_zstd=10;

3、查看压缩前后相同SQL产生的binlog大小。

压缩前binlog大小约为300M

-rw-r----- 1 mysql mysql 251M May 6 09:31 mysql-bin.000001

-rw-r----- 1 mysql mysql 50M May 6 09:31 mysql-bin.000002

压缩后binlog大小约150M

-rw-r----- 1 mysql mysql 148M May 6 09:32 mysql-bin.000004

4、查看压缩前后相同SQL在低主从带宽的网络环境中tps的比较。

限制网络速率

tc qdisc add dev eth0 root handle 1:0 netem delay 100ms

tc qdisc add dev eth0 parent 1:1 handle 10: tbf rate 256kbit buffer 1600 limit 3000

压缩前压测结果：

SQL statistics:

queries performed:

read: 3976

write: 1136

other: 568

total: 5680

transactions: 284 (9.17 per sec.)

queries: 5680 (183.32 per sec.)

压缩后压测结果：

SQL statistics:

queries performed:

read: 4746

write: 1356

other: 678

total: 6780

transactions: 339 (10.15 per sec.)

queries: 6780 (202.92 per sec.)

**结论**

1、MySQL新推出的binlog压缩功能，当压缩级别设置为10时，压缩率约为50%左右，能够较大程度减少binlog所占用的空间。

2、压缩功能能够一定程度提升因网络带宽所带来的主从延迟，集群tps不降低，略微提升。

# 参数

max\_binlog\_size：指定了单个二进制日志文件最大值，如果超过该值，则产生新的二进制日志文件后缀名+1，并记录到.index文件。

binlog\_cache\_size：控制缓冲大小，默认大小32K，基于会话的，因此每开启一个事务就分配一个binlog\_cache\_size大小的缓存，所以不能设置过大。当一个事务的记录大于binlog\_cache\_size时，MySQL会把缓冲中的日志写入一个临时文件中，因此该值又不能设置太小。

通过SHOW GLOBAL STATUS命令查看binlog\_cache\_use、binlog\_cache\_disk\_use的状态，可以判断当前binlog\_cache\_size的设置是否合理。binlog\_cache\_use记录了使用缓冲写二进制日志的次数，binlog\_cache\_disk\_use记录了使用临时文件写二进制日志的次数。

sync\_binlog：表示每写缓冲多少次就要同步到磁盘。如果设置为1，表示采用同步写磁盘的方式来写二进制日志，这时候写操作不使用操作系统的缓冲来写二进制日志。sync\_binlog的默认值为0，如果使用InnoDB存储引擎进行复制，并且想得到最大的可用性，建议将该值设置为ON（对数据库IO系统带来一定影响）。

但是，即使将sync\_binlog设置为1，还是会有一种情况导致问题发生。当使用InnoDB存储引擎时，在一个事务发出COMMIT动作之前，由于sync\_binlog为1，因此会将二进制日志立即写入磁盘。如果这时写入了二进制日志，但是提交还没有发生，并且此时发生了宕机，那么在MySQL数据库下次启动时，由于COMMIT操作并没有发生，这个事务被回滚。但是二进制日志已经记录了该事物信息，不能被回滚。这个问题可以通过参数innodb\_support\_xa设置为1来解决，虽然innodb\_support\_xa与XA事务有关，但是它同时也确保了二进制日志和InnoDB存储引擎文件的同步。

binlog-do-db：表示需要写入哪些库的日志，默认为空，表示需要同步所有库的日志到二进制日志。

binlog-ignore-db：表示需要忽略写入哪些库的日志，默认为空，表示需要同步所有库的日志到二进制日志。

log-slave-update：如果当前数据库是复制中的slave节点，则它不会将从master取得并执行的二进制日志写入自己的二进制文件中。如果需要写入，要设置log-slave-update。如果需要搭建master->slave->slave这种架构的复制，则必须设置该参数。

binlog\_format：记录二进制日志的格式。在MySQL5.1之前，没有这个参数，所有二进制文件的格式都是基于SQL语句（statement）级别的，因此基于这个格式的二进制日志文件的复制（Replication）和Oracle的逻辑Standby有点类似。

该值可以设置为STATEMENT、ROW和MIXED。

1、STATEMENT格式下，记录的是逻辑SQL语句。

2、ROW格式下，记录表的行更改情况。

3、MIXED格式下，MySQL默认采用STATEMENT格式进行二进制文件记录，但是在一些情况下会使用ROW格式。

# 操作命令

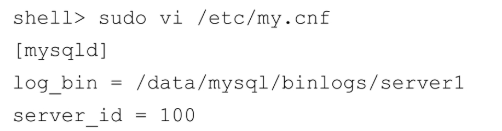
## 开启日志

要启用二进制日志，必须设置 log\_bin 和 server\_id 并重新启动服务器。可以在log\_bin 内提及 path和 base 名称。例如，log\_bin 设置为/data/mysql/binlogs/server1，二进制日志存储在/data/mysql/binlogs文件夹中名为server1.000001、server1.000002等的日志文件中。每当服务器启动或刷新日志时，或者当前日志的大小达到max\_binlog\_size时，服务器都会在系列中创建一个新文件。每个二进制日志的位置都在server1.index文件中被维护。

### 启用二进制日志

1. 启用二进制日志并设置server\_id。在自己常用的编辑器中打开MySQL配置文件并添加以下代码。选择server\_id，使其在基础架构中对每个MySQL服务器都是唯一的。

也可以简单地把log\_bin变量放在my.cnf中，不赋予任何值。在这种情况下，二进制日志是在数据目录中创建的。可以使用主机名作为目录名称。



**扩展：**

开启binlog日志：

1）vi编辑打开mysql配置文件

# vi /usr/local/mysql/etc/my.cnf

在[mysqld] 区块

设置/添加 log-bin=mysql-bin 确认是打开状态(值 mysql-bin 是日志的基本名或前缀名)；

重启mysqld服务使配置生效

# pkill mysqld

# /usr/local/mysql/bin/mysqld\_safe --user=mysql &

2）也可登录mysql服务器，通过mysql的变量配置表，查看二进制日志是否已开启 单词：variable[ˈvɛriəbəl] 变量

登录服务器

# /usr/local/mysql/bin/mysql -uroot -p123456

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

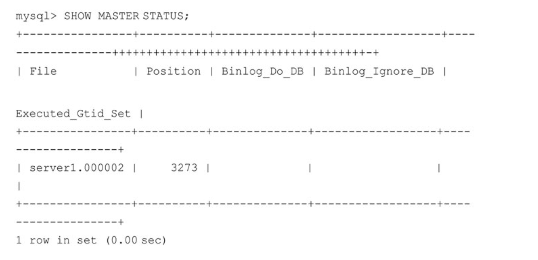
1. 重新启动MySQL服务器：



1. 验证是否创建了二进制日志：



1. 执行SHOW BINARY LOGS；或SHOW MASTER LOGS；，以显示服务器的所有二进制日志。
2. 执行命令SHOW MASTER STATUS；以获取当前的二进制日志位置：



一旦server1.000001达到max\_binlog\_size（默认为1 GB），一个新文件server1.000002就会被创建，并被添加到server1.index中。可以使用 SET@@global.max\_binlog\_size=536870912动态设置max\_binlog\_size。

### 禁用会话的二进制日志

有些情况下我们不希望将执行语句复制到其他服务器上。为此，可以使用以下命令来禁用该会话的二进制日志：



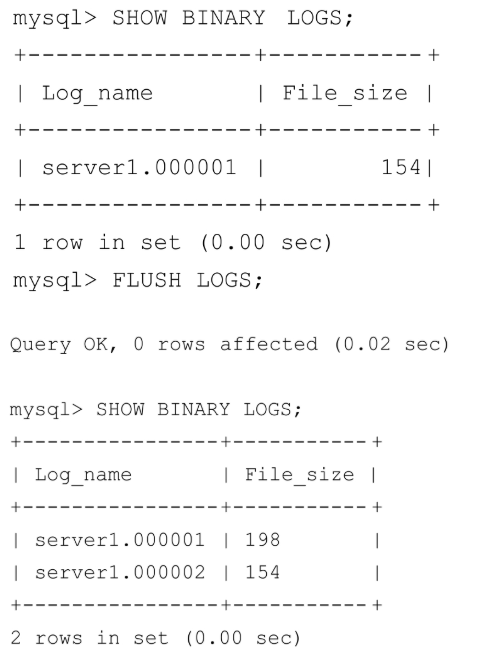
在这条语句后的所有SQL语句都不会被记录到二进制日志中，不过这仅仅是针对该会话的。

要重新启用二进制日志，可以执行以下操作：



### 移至下一个日志

可以使用FLUSH LOGS命令关闭当前的二进制日志并打开一个新的二进制日志：



### 清理二进制日志

随着写入次数的增多，二进制日志会消耗大量空间。如果放任不管，这些写入操作将很快占满磁盘空间，因此清理它们至关重要。

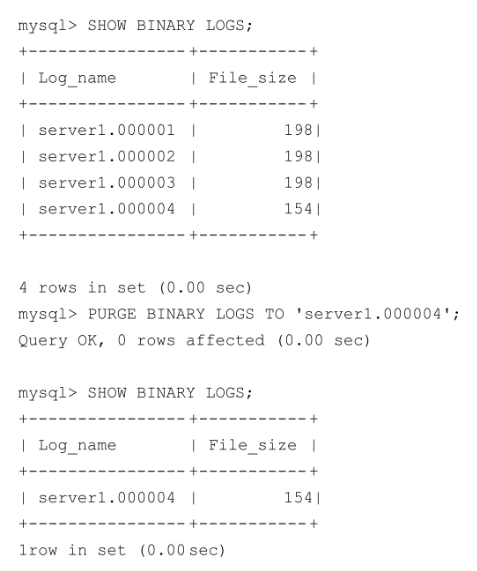
1. 使用binlog\_expire\_logs\_seconds 和expire\_logs\_days 设置日志的到期时间。

如果想以天为单位设置到期时间，请设置 expire\_logs\_days。例如，如果要删除两天之前的所有二进制日志，请SET@@global.expire\_logs\_days=2。如果将该值设置为0，则禁用设置会自动到期。

如果想以更细的粒度来设置到期时间，可以使用 binlog\_expire\_logs\_seconds变量，它能够以秒为单位来设置二进制日志过期时间。

这个变量的效果和 expire\_logs\_days 的效果是叠加的。例如，如果expire\_logs\_days是1并且binlog\_expire\_logs\_seconds是43200，那么二进制日志就会每 1.5 天清除一次。这与将binlog\_expire\_logs\_seconds设置为129600、将expire\_logs\_days设置为0的效果是相同的。在MySQL 8.0 中，binlog\_expire\_logs\_seconds 和expire\_logs\_days必须设置为0，以禁止自动清除二进制日志。

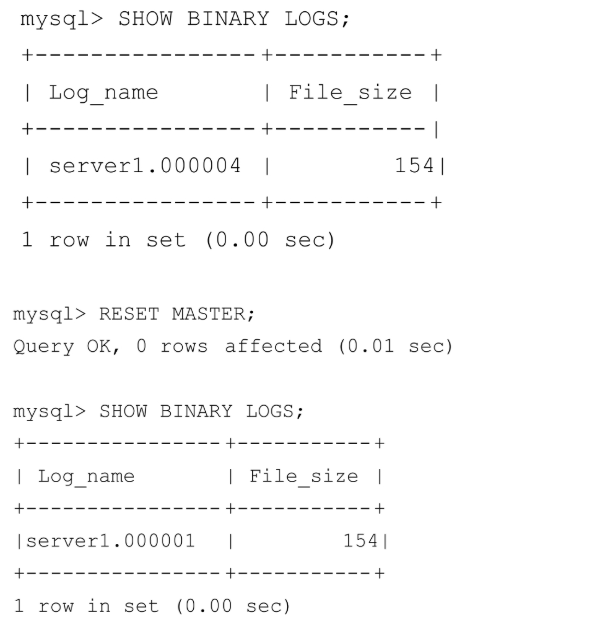
1. 要手动清除日志，请执行PURGE BINARY LOGS TO＇＜file\_name＞＇。例如，有server1.000001、server1.000002、server1.000003和server1.000004文件，如果执行 PURGEBINARY LOGS TO＇server1.000004＇，则从server1.000001 到 server1.000003 的所有文件都会被删除，但文件server1.000004不会被删除：



除了指定某个日志文件，还可以执行命令PURGE BINARY LOGS BEFORE＇2017-08-03 15：45：00＇。除了使用BINARY，还可以使用MASTER。

mysql＞ PURGE MASTER LOGS TO ＇server1.000004＇可以实现和之前语句一样的效果。

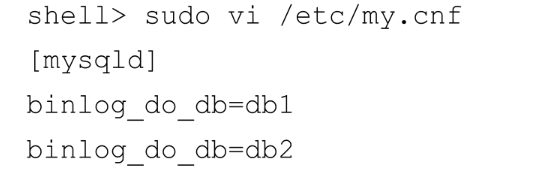
1. 要删除所有二进制日志并再次从头开始，请执行RESET MASTER：



## 忽略要写入二进制日志的数据库

可以通过在my.cnf中指定--binlog-do-db=db\_name选项，来选择将哪些数据库写入二进制日志。要指定多个数据库，就必须使用此选项的多个实例。由于数据库的名字可以包含逗号，因此如果提供逗号分隔列表，则该列表将被视为单个数据库的名字。需要重新启动MySQL服务器才能使更改生效。

打开my.cnf并添加以下行



binlog-do-db 上的行为从基于语句的日志记录更改为基于行的日志记录，就像mysqlbinlog实用程序中的--database选项一样。在基于语句的日志记录中，只有那些默认数据库（即用USE选择的）的语句才会被写入二进制日志。使用 binlog-do-db 选项时应该非常小心，因为它的工作方式与你在使用基于语句的日志记录时的方式不同。

## 查看日志列表

mysql> show master logs;

## 查看节点状态

即最后(最新)一个binlog日志的编号名称，及其最后一个操作事件pos结束点(Position)值

查看主节点状态信息：

mysql> show master status;

查看从节点状态信息：

mysql> show slave status;

## 刷新log日志

自此刻开始产生一个新编号的binlog日志文件

mysql> flush logs;

注：每当mysqld服务重启时，会自动执行此命令，刷新binlog日志；在mysqldump备份数据时加 -F 选项也会刷新binlog日志；

## 重置(清空)所有binlog日志

mysql> reset master;

## 启动复制

Start slave可以指定线程类型：IO\_THREAD，SQL\_THREAD，如果不指定，两个都启动。

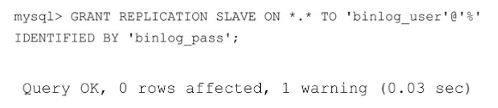
注：具体工作流程参考《MySQL主从复制》。

# 二进制日志备份

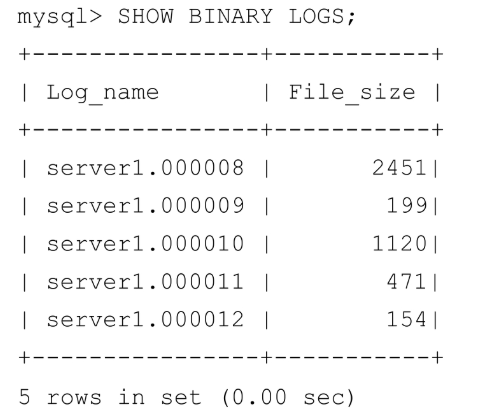
备份二进制日志：该进程将二进制日志从数据库服务器流式传输到远程备份服务器。既可以从从服务器也可以从主服务器进行二进制日志备份。如果你正在从主服务器进行二进制日志备份，并在从服务器进行实际备份，则应使用--dump-slave获取相应的主日志位置。如果你使用的是mydumper或XtraBackup，则主和从二进制日志位置会被同时提供。

**操作步骤：**

1. 在服务器上创建一个复制用户，并设置一个强密码：

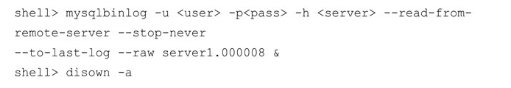


1. 检查服务器上的二进制日志：

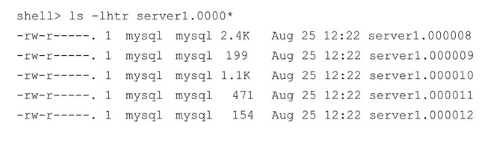


你可以在服务器上找到第一个可用的二进制日志，可以从这里开始备份。在这个例子中，它是server1.000008。

1. 登录到备份服务器并执行以下命令，会将二进制日志从MySQL服务器复制到备份服务器。你可以使用nohup或disown：



1. 验证是否正在备份二进制日志：



# 解析工具

## mysqlbinlog

可以使用mysqlbinlog实用程序（MySQL已包含）从二进制日志中提取内容，并将其应用到其他服务器。

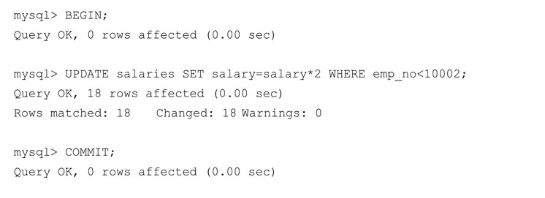
1、准备工作

使用各种二进制格式执行几条语句。如果把binlog\_format设置为GLOBAL级别（全局范围），必须断开并重新连接，以使更改生效。如果想保持连接，请把binlog\_format设置为SESSION级别（会话范围）。

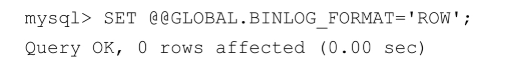
更改为基于语句的复制（SBR）：



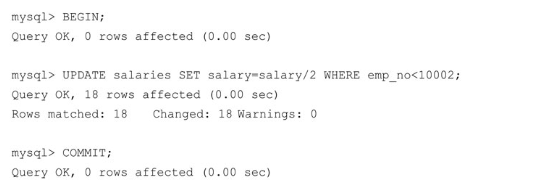
更新几行：



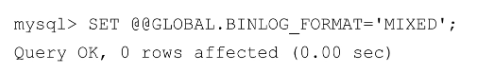
更改为基于行的复制（RBR）：



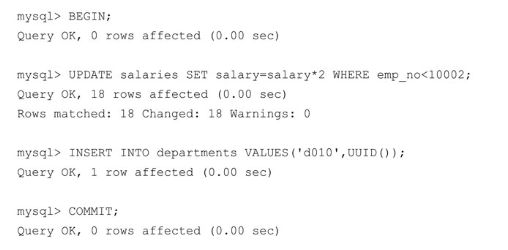
更新几行：



改为MIXED格式：



更新几行：

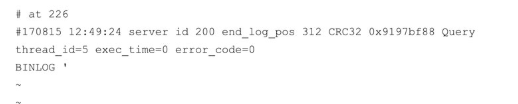


1. 操作步骤

要显示日志server1.000001的内容，请执行以下操作：



你会得到类似下面这样的输出结果：



在第一行中，＃at后面的数字表示二进制日志文件中事件的起始位置（文件偏移量）。第二行包含了语句在服务器上被启用的时间戳。时间戳后面跟着 server ID、end\_log\_pos、thread\_id、exec\_time和error\_code。

● server id：产生该事件的服务器的server\_id值（在这个例子中为200）。

● end\_log\_pos：下一个事件的开始位置。

● thread\_id：指示哪个线程执行了该事件。

● exec\_time：在主服务器上，它代表执行事件的时间；在从服务器上，它代表从服务器的最终执行时间与主服务器的开始执行时间之间的差值，这个差值可以作为备份相对于主服务器滞后多少的指标。

● error\_code：代表执行事件的结果。零意味着没有错误发生。

**回顾：**

1. 我们在基于语句的复制中执行了UPDATE语句，而且在二进制日志中记录了相同的语句。除了保存在服务器上，会话变量也被保存在二进制日志中，以在从库上复制相同的行为：



1. 当使用基于行的复制（RBR）时，会以二进制格式对整行（而不是语句）进行保存，而且二进制格式不能读取。此外，你可以观察长度，单个更新语句会生成很多数据。



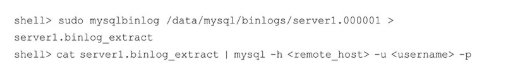
1. 当使用MIXED格式时，UPDATE语句被记录为SQL语句，而INSERT语句以基于行的格式被记录，因为INSERT有非确定性的UUID()函数：



提取的日志可以被传送给MySQL以回放事件。在重放二进制日志时最好使用force选项，这样即使force选项卡在某个点上，执行也不会停止。稍后，你可以查找错误并手动修复数据。



或者也可以先保存到文件中，稍后执行：



### 根据时间和位置进行抽取

我们可以通过指定位置从二进制日志中提取部分数据。假设你想做时间点恢复。假如在 2017-08-1912：18：00 执行了 DROP DATABASE 命令，并且最新的可用备份是在2017-08-19 12：00：00 做的，该备份已经恢复。现在，需要恢复从 12：00：01 至2017-08-19 12：17：00 的数据。请记住，如果提取完整的日志，它还将包含 DROP DATABASE命令，该命令将再次擦除数据。

可以通过--start-datetime和--stop-datatime选项来指定提取数据的时间窗口。



使用时间窗口的缺点是，你会失去灾难发生那一刻的事务。为避免这种情况，必须在二进制日志中使用事件的文件偏移量。

一个连续的备份会保存它已完成备份的所有binlog文件的偏移量。备份恢复后，必须从备份的偏移量中提取binlog。

假设备份的偏移量为471，执行DROP DATABASE命令的偏移量为1793。可以使用--start-position和--stop-position选项来提取偏移量之间的日志：



请确保DROP DATABASE命令在提取的binlog中不再出现。

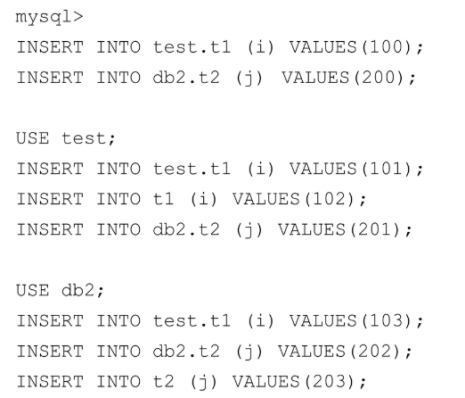
### 基于数据库进行提取

使用--database选项可以过滤特定数据库的事件。如果多次提交，则只有最后一个选项会被考虑。这对于基于行的复制非常有效。但对于基于语句的复制和MIXED，只有当选择默认数据库时才会提供输出。

以下命令从employees 数据库中提取事件：



正如MySQL 8文档中所解释的，假设二进制日志是通过使用基于语句的日志记录执行这些语句而创建的：



mysqlbinlog--database=test 不输出前两个 INSERT 语句，因为没有默认数据库。

mysqlbinlog--database=test 输出USE test后面的三条INSERT语句，但不是USE db2后面的三条INSERT语句。

因为没有默认数据库，mysqlbinlog--database=db2 不输出前两条INSERT语句。

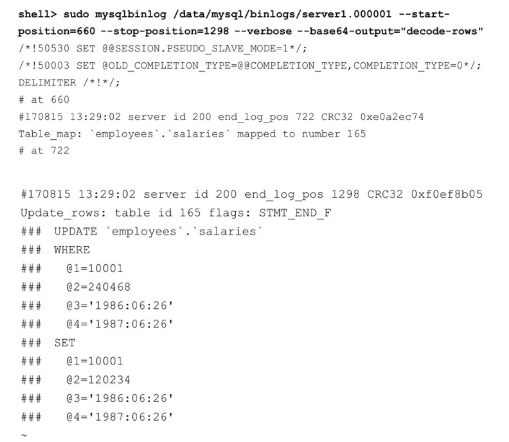
mysqlbinlog--database=db2不会输出USE test后的三条INSERT语句，但会输出在USE db2之后的三条INSERT语句。

### 提取行事件显示

默认情况下，基于行的复制日志显示为二进制格式。要查看行信息，必须将--verbose或-v选项传递给mysqlbinlog。行事件的二进制格式以注释的伪SQL语句的形式显示，其中的行以＃＃＃开始。可以看到，单个更新语句被改写为了每行的UPDATE语句。



如果你只想查看没有二进制行信息的伪 SQL 语句，请指定--base64-output=〝decode-rows〝以及--verbose：



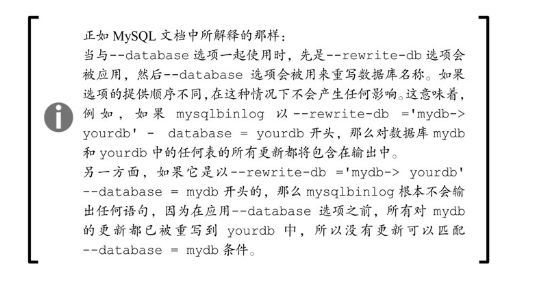
### 重写数据库名称

假设你想将生产服务器上的employees 数据库的二进制日志恢复为开发服务器上的employees\_dev，可以使用--rewrite-db=＇from\_name-＞ to\_name＇选项。这会将所有from\_name重写为to\_name。

要转换多个数据库，请多次指定该选项：



可以看到上面使用了 use`employees\_dev`/\*！\*/；语句。因此，在恢复时，所有更改将应用于employees\_dev数据库。



### 在恢复时禁用二进制日志

在恢复二进制日志的过程中，如果你不希望mysqlbinlog进程创建二进制日志，则可以使用--disable-log-bin选项：

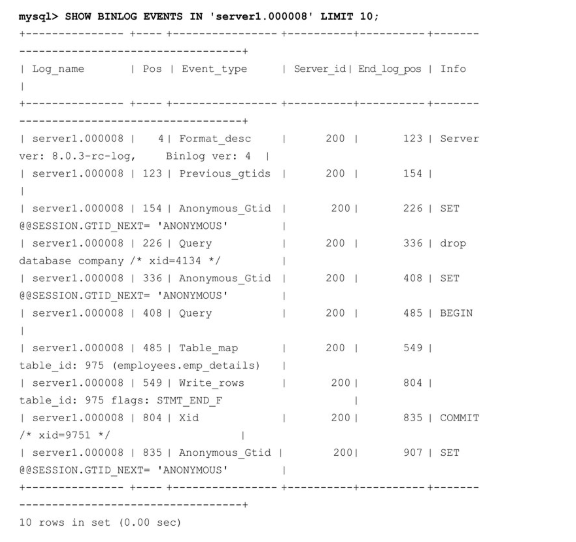


可以看到SQL\_LOG\_BIN=0被写入binlog恢复文件中，这将防止创建binlog。

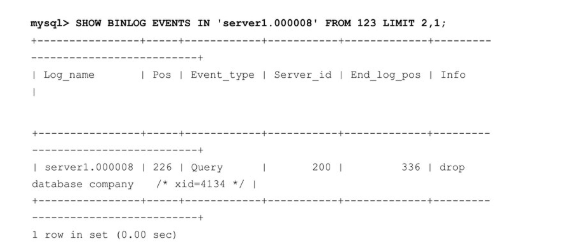


### 显示二进制日志文件中的事件

除了使用mysqlbinlog，还可以使用SHOW BINLOG EVENTS命令来显示事件。以下命令将显示server1.000008二进制日志中的事件。如果未指定LIMIT，则显示所有事件：



也可以指定位置和偏移量：



# 迁移工具

## Mysqlbinlogmove

由于二进制日志占用越来越多的空间，有时你可能希望更改二进制日志的位置，可以按照以下步骤操作。单独更改 log\_bin 是不够的，必须迁移所有二进制日志并在索引文件中更新位置。mysqlbinlogmove工具可以自动执行这些任务，简化你的工作。

操作步骤：

你需要先安装MySQL工具集，以使用mysqlbinlogmove脚本。

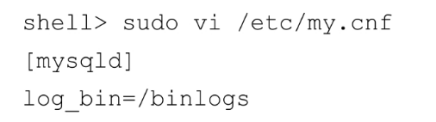
1. 停止MySQL服务器的运行：



1. 启动mysqlbinlogmove工具。如果要将二进制日志从/data/mysql/binlogs更改为/binlogs，则应使用以下命令。如果base name不是默认名称，则必须通过--bin-log-basename 选项设定basename：



1. 编辑my.cnf文件并更新log\_bin的新位置：



1. 启动MySQL服务器：



如果有大量的二进制日志，服务器的停机时间会很长。为了避免这种情况，可以使用--server选项来重新定位所有二进制日志——但是当前正在使用的二进制日志（具有较高序列号）除外。然后停止服务器的运行，使用上述方法，并重新定位最后一个二进制日志，这会快很多，因为只有一个文件存在。然后你可以更改my.cnf并启动服务器。例如：



# 应用

二进制有两个最重要的使用场景:

1. MySQL Replication在Master端开启binlog，Mster把它的二进制日志传递给slaves来达到master-slave数据一致的目的。

通过复制和执行二进制日志使得远程的MySQL数据库（一般称为slave或standby）与一台MySQL数据库（一般称为master或primary）进行实时同步。

1. 数据恢复，某些数据的恢复需要二进制日志，例如，如果需要恢复数据库全量备份的文件，可以通过二进制日志进行point-in-time恢复。

通过使用mysqlbinlog工具来使恢复数据。

3、审计（audit）：用户通过二进制日志中的信息来进行审计，判断是否有对数据库进行注入的攻击。