# 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**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(参见MySQL官方中文手册5.1.24版)。

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。

# 分类

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

每一条会修改数据的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：**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

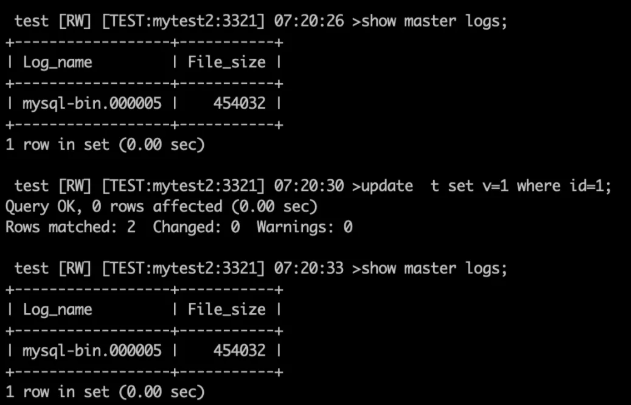
从5.1.8版本开始，MySQL提供了Mixed格式，实际上就是Statement与Row的结合。

在Mixed模式下，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。

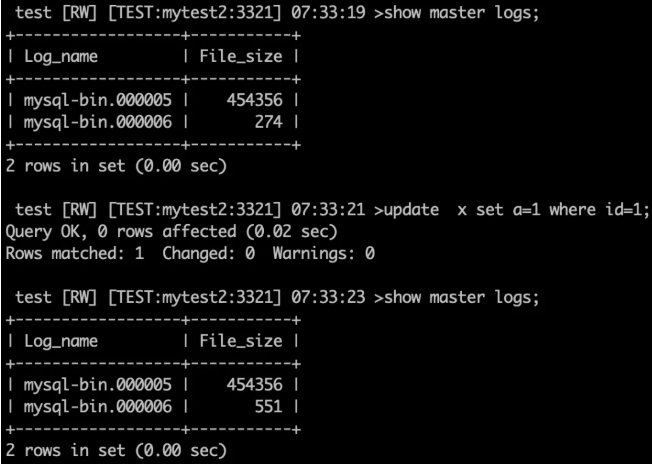
## 区别

**update字段为相同的值是否会记录binlog？**

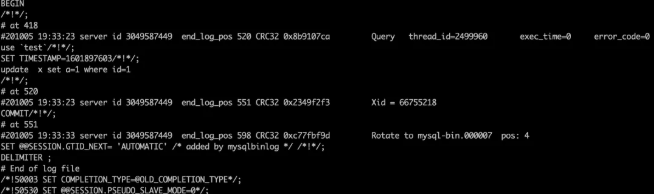
binlog\_format为ROW模式



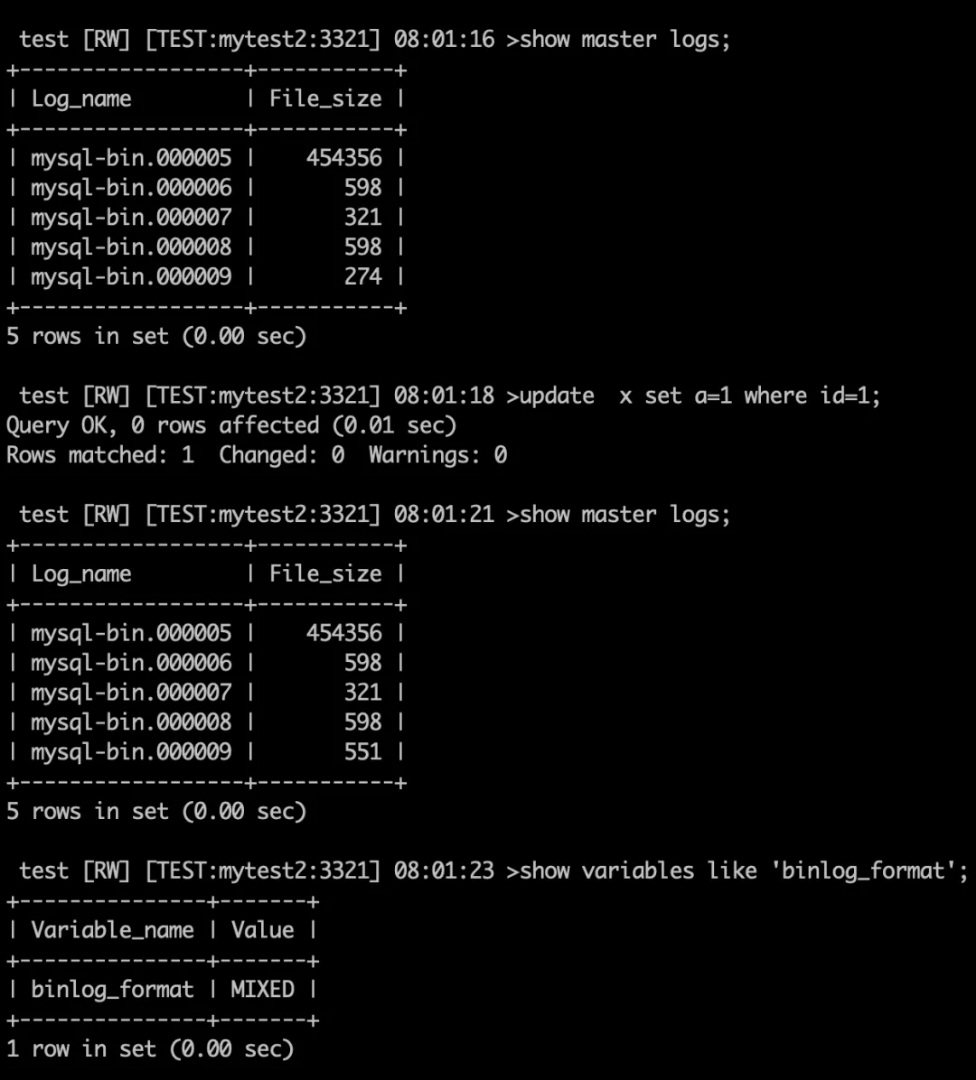
binlog\_format为STATEMENT模式



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。

# 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，这些情况包括：
4. 表的存储引擎为NDB，这时对表的DML操作都会以ROW格式记录；
5. 使用了UUID()、USER()、CURRENT\_USER()、FOUND\_ROWS()、ROW\_COUNT()等不确定函数；
6. 使用了INSERT DELAY语句；
7. 使用了用户定义函数（UDF）；
8. 使用了临时表（temporary table）。

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）。

# 操作命令

## 开启

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\_%';

## 查看所有binlog日志列表

mysql> show master logs;

## 查看master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

mysql> show master 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;

# 解析工具

mysqlbinlog