目录

[原理 1](#_Toc442082900)

[主从同步配置 2](#_Toc442082901)

[主服务器同步用户授权 2](#_Toc442082902)

[配置MySQL主服务器的my.cnf文件 3](#_Toc442082903)

[备机配置： 4](#_Toc442082904)

[常用命令： 5](#_Toc442082905)

[双主配置my.cnf 6](#_Toc442082906)

[binlog\_ignore\_db引起的同步复制故障 7](#_Toc442082907)

[常见错误 11](#_Toc442082908)

[Mysql Binlog三种格式介绍及分析 11](#_Toc442082909)

## 原理

**经过抓包分析，tcpdump -n -i eth0 -A -s0 -v host 192.168.0.253。当从与主处于正常连接状态时（而不是slave第一次启动时），主发生sql操作时，是将binlog主动推送给从服务器。**

**当正常同步之后，如果Slave mysql 停止，如服务停止了，或者设备故障了。那么在slave重新正常后，在这期间的主的变化都会正常同步到slave。**

一 MySQL 复制的基本过程如下：（各部分学习自Google，谢谢）  
1. Slave 上面的IO线程连接上 Master，并请求从指定日志文件的指定位置(或者从最开始的日志)之后的日志内容;

2. Master 接收到来自 Slave 的 IO 线程的请求后，通过负责复制的 IO线程根据请求信息读取指定日志指定位置之后的日志信息，返回给 Slave 端的 IO线程。返回信息中除了日志所包含的信息之外，还包括本次返回的信息在 Master 端的 Binary Log 文件的名称以及在 BinaryLog 中的位置;  
3. Slave 的 IO 线程接收到信息后，将接收到的日志内容依次写入到 Slave 端的RelayLog文件(mysql-relay-lin.xxxxxx)的最末端，并将读取到的Master端的bin-log的文件名和位置记录到 master-info文件中，以便在下一次读取的时候能够清楚的高速Master“我需要从某个bin-log的哪个位置开始往后的日志内容，请发给 我”

4. Slave 的 SQL 线程检测到 Relay Log 中新增加了内容后，会马上解析该 Log 文件中的内容成为在 Master  
端真实执行时候的那些可执行的 Query 语句，并在自身执行这些 Query。这样，实际上就是在 Master 端和 Slave端执行了同样的 Query，所以两端的数据是完全一样的。

## 主从同步配置

安装：

yum install mysql mysql-server #安装

cp /usr/share/mysql/my-medium.cnf /etc/my.cnf #复制配置文件

service mysql start #启动

chkconfig mysql on #设置开机自动启动

mysql\_secure\_installatio n #初始化数据库，删除test库；禁止root远程登录；

mysql root密码：81233744!@#

192.168.0.80；192.168.0.253 的mysql密码修改为 root/A81233744

同步用的账号和密码：HELLO\_backup/HELLO@135!#%

修改mysql的服务端口

vim /etc/my.cnf

## 主服务器同步用户授权

CREATE DATABASE `backup` DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci; 建库

CREATE USER 'HELLO\_backup'@'192.168.0.80' IDENTIFIED BY 'HELLO@135!#%'; 建备份用的用户和密码（建用户时填写允许用户备份操作的IP；在给此用户赋权时的IP必须与此相同，否则赋不上权限）

GRANT FILE,REPLICATION SLAVE ON \*.\* TO 'HELLO\_backup'@'192.168.0.80' IDENTIFIED BY 'HELLO@135!#%'; #给备份用户相应的权限，

**REVOKE FILE,REPLICATION SLAVE ON \*.\* FROM 'HELLO\_backup'@'192.168.0.232'; #收回授权**

如果中间还通过防火墙做的静态地址映射还需要增加防火墙外网口的地址和映射用的地址，否则连不上

GRANT FILE,REPLICATION SLAVE ON \*.\* TO 'HELLO\_backup'@'192.168.0.151' IDENTIFIED BY 'HELLO@135!#%';

GRANT FILE,REPLICATION SLAVE ON \*.\* TO 'HELLO\_backup'@'192.168.0.155' IDENTIFIED BY 'HELLO@135!#%';

GRANT ALL PRIVILEGES ON \*.\* TO 'HELLO\_backup'@'192.168.0.155' IDENTIFIED BY 'HELLO@135!#%';

flush privileges; 每次赋权后必须刷新

**三、把MySQL主服务器中的数据库osyunweidb导入到MySQL从服务器中**1、导出数据库osyunweidb

备注：在导出之前可以先进入MySQL控制台执行下面命令

flush tables with read lock;    #生产环境必须先锁定。数据库只读锁定命令，防止导出数据库的时候有数据写入。这个命令是全局读锁定，执行了命令之后所有库所有表都被锁定只读。一般都是用在数据库联机备份，这个时候数据库的写操作将被阻塞，读操作顺利进行。解锁的语句也是unlock tables。

mysqldump -u root -p osyunweidb > /home/osyunweidbbak.sql    #在MySQL主服务器进行操作，导出数据库osyunweidb到/home/osyunweidbbak.sql

unlock tables;   #解除锁定

2、导入数据库到MySQL从服务器

mysql  -u root -p  #进入从服务器MySQL控制台

create database osyunweidb;   #创建数据库

use osyunweidb    #进入数据库

source  /home/osyunweidbbak.sql  #导入备份文件到数据库

mysql -u osyunweidbbak -h 192.168.21.169 -p  #测试在从服务器上登录到主服务器

## ****MySQL主服务器配置****

use backup; 将backup库设置为当前库

create table mytest (username varchar(20),password varchar(20)); 创建一个测试用表

vi /etc/my.cnf #编辑配置文件，在[mysqld]部分添加下面内容

server-id=1 #设置服务器id，为1表示主服务器，注意：如果原来的配置文件中已经有这一行，就不用再添加了。

log-bin=mysql-bin #启动MySQL二进制日志系统，注意：如果原来的配置文件中已经有这一行，就不用再添加了。

binlog\_format           = MIXED #建议使用MIXED格式。使用show variables like 'binlog\_format'查看；

expire\_logs\_days    = 10 #binlog过期清理时间，根据每日生成的日志量，磁盘空间等设置过期时间。

max\_binlog\_size         = 100M #每个日志文件大小的最大值

binlog-do-db=backup #需要同步的数据库名，如果有多个数据库，可重复此参数，每个数据库一行

binlog-ignore-db=mysql #不同步mysql数据库

binlog-ignore-db=information\_schema

binlog-ignore-db=performance\_schema

service mysql restart #重启MySQL

mysql -u root -p #进入mysql控制台

 查看主服务器，出现以下类似信息

mysql> show master status;

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

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |

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

| mysql-bin.000001 | 106 | backup | |

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

1 row in set (0.00 sec)

查看server-id：

mysql> show variables like 'server\_id';

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

| Variable\_name | Value |

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

| server\_id | 1 |

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

1 row in set (0.00 sec)

## Mysql从服务器配置

vi /etc/my.cnf    #编辑配置文件，在[mysqld]部分添加下面内容  
server-id=2    #配置文件中已经有一行server-id=1，修改其值为2，主从不能相同  
log-bin=mysql-bin   #启动MySQL二进制日志系统，如果只做从数据库则不起也可以，建议开启。

binlog\_format  = MIXED #建议使用MIXED格式。使用show variables like 'binlog\_format'查看；

expire\_logs\_days    = 10 #binlog过期清理时间，根据每日生成的日志量，磁盘空间等设置过期时间。

max\_binlog\_size  = 100M #每个日志文件大小的最大值

read-only = 1 #设置为只读，以免被误写入而导致主从不同步  
replicate-do-db=backup    #需要同步的数据库名，如果有多个数据库，可重复此参数，每个数据库一行  
replicate-ignore-db=mysql    #不同步mysql系统数据库

:wq!    #保存退出  
service mysql restart   #重启MySQL

**注意：MySQL 5.1.7版本之后，已经不支持把master配置属性写入my.cnf配置文件中了，配置文件中只需要把同步的数据库和要忽略的数据库写入即可。**  
mysql  -u root -p  #进入MySQL控制台  
slave stop;   #停止slave同步进程

进入主库，锁定主库表：

flush tables with read lock; #生产环境必须先锁定。  
change master to master\_host='192.168.0.155', MASTER\_PORT=3306,master\_user='HELLO\_backup',master\_password='HELLO@135!#%',master\_log\_file='mysql-bin.000001' ,master\_log\_pos=106;   #执行同步语句（执行同步必须在mysql> stop slave; 的状态下进行）  
start slave;    #开启slave同步进程

进入主库，解锁主库标：

unlock tables;#解锁。  
  
SHOW SLAVE STATUS\G   #查看slave同步信息，出现以下内容  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
               Slave\_IO\_State: Waiting for master to send event  
                  Master\_Host: 192.168.21.169  
                  Master\_User: osyunweidbbak  
                  Master\_Port: 3306  
                Connect\_Retry: 60  
              Master\_Log\_File: mysql-bin.000019  
          Read\_Master\_Log\_Pos: 7131  
               Relay\_Log\_File: MySQLSlave-relay-bin.000002  
                Relay\_Log\_Pos: 253  
        Relay\_Master\_Log\_File: mysql-bin.000019  
             **Slave\_IO\_Running: Yes**  
**Slave\_SQL\_Running: Yes**  
              Replicate\_Do\_DB: osyunweidb  
          Replicate\_Ignore\_DB: mysql  
           Replicate\_Do\_Table:  
       Replicate\_Ignore\_Table:  
1 row in set (0.00 sec)

测试：

在主备数据库服务器上：

CREATE DATABASE `backup` DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci; 建库

create table cc(id int auto\_increment,name varchar(20),primary key(id)); 建表

insert into cc (name) values('Mr.chai'); 插数据

在备数据库服务器上查看是否有新建的表和数据。

## 常用命令：

1、用户创建完，赋完权限之后在备服务器上用此命令进行测试。如果能进行登录则表明权限和连通性没问题。

mysql -u osyunweidbbak -h 192.168.21.169 -p  #测试在从服务器上登录到主服务器

1、查看用户及权限

SELECT DISTINCT CONCAT('User: ''',user,'''@''',host,''';') AS query FROM mysql.user;

Show grants for 'HELLO\_backup'@'10.34.34.232';

允许root用户从192.168.0.253访问

mysql>GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'192.168.0.253' IDENTIFIED BY 'A81233744';

mysql> flush privileges; 修改权限之后需要执行此句以使之生效。

1、手动同步主数据库；必须在slave stop的状态下；

mysql>change master to master\_host='192.168.0.155',master\_user='HELLO\_backup',master\_password='HELLO@135!#%',master\_log\_file='mysql-bin.000002' ,master\_log\_pos=2394;

2、查看slave状态

mysql> show slave status\G;

3、手动从主数据库下载

mysql> load data from master;

4、在主备上可以查看各自的进程状态，主上一个BinlogDump，从上两个一个是I/O一个是SQL进程，I/O进程负责接收更新，SQL负责写入本地库。

show processlist;

5、查看参数配置

show variables like 'slave%';

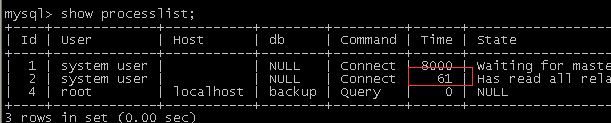
6、查看某用户权限

Show grants for 'HELLO\_backup'@'192.168.0.232';

7、修改配置文件my.cnf，在[mysqld]下添加slave\_net\_timeout = 600；此参数的默认值是3600（秒，1小时），是指 slave 端（备数据库）的 I/O 线程处于 “waiting for master to send event”状态，如果这个等待状态超过 slave\_net\_timeout 时间，就会触发重连 master 的动作。

slave\_net\_timeout = 600 此参数的合理值需要在实际环境中进行一下测试，时间太长，容易导致同步不及时，时间太短，则备机会频繁连接主机。

在一个已经建立主从复制关系的系统里面,正常情况下,由从库向主库发送一个 COM\_BINLOG\_DUMP 命令后,主库有新的binlog event，会向备库发送binlog。但是由于网络故障或者其他原因（如主从数据库之间跨防火墙）导致主库与从库的连接断开或者主库长时间没有向从库发送binlog。例如该例子中数据库集群 10s 左右还没有写入的情况，超过slave\_net\_timeout设置的值，从库会向主库发起重连请求。5.6 版本slave 发起重连请求时，MySQL都会判断有没有用明文的用户名密码,如果有则发出上述信息到error.log。



8、--logs-slave-updates  参数  
这个是在my.cnf文件配置的   
通常情况，从服务器从主服务器接收到的更新不记入它的二进制日志。该选项告诉从服务器将其SQL线程执行的更新记入到从服务器自己的二进制日志。为了使该 选项生效，还必须用--logs-bin选项启动从服务器以启用二进制日志。如果想要应用链式复制服务器，应使用--logs-slave- updates。例如，可能你想要这样设置：   
A -> B -> C   
也就是说，A为从服务器B的主服务器，B为从服务器C的主服务器。为了能工作，B必须既为主服务器又为从服务器。你必须用--logs-bin启动A和B以启用二进制日志，并且用--logs-slave-updates选项启动B。

在my.cnf文件设置此选项

log\_slave\_updates=1

当然在这种机制下可能有的同学会存在这么个问题：  
如果a->b b->a 这样的双master架构下，a,b都打开log\_slave\_updates选项会不会出现无限循环的状态。  
mysql已经考滤到了这个问题，每条bin-log都会记录执行语句的源server\_id.当slave读到语句的server\_id等于本身的ID的时候，不会执行，所以我们不用担心a,b会不会无限循环下去。

基于以上这种情况，mysql的replication集群将更加灵活，你如果需要可以做成各种各样的链式复制。比如 a->b b->a b中设置log\_slave\_updates后还可以b->c. 这样a,c中的数据也是一致的。

## 双主配置my.cnf

1. log\_slave\_updates = 1    #添加(将复制事件写入binlog,一台服务器既做主库又做从库此选项必须要开启)
2. replicate-same-server-id=0 #添加(防止MySQL循环更新)
3. relay\_log\_recovery = 1   #添加(MySQLrelay\_log的自动修复功能)

\*从MySQL5.5.X版本才开始支持，增加了relay\_log\_recovery参数，这个参数的作用是：当slave从库宕机后，假如relay-log损坏了，导致一部分中继日志没有处理，则自动放弃所有未执行的relay-log，并且重新从master上获取日志，这样就保证了relay-log的完整性。默认情况下该功能是关闭的，将relay\_log\_recovery的值设置为 1时，可在slave从库上开启该功能，建议开启。

1. server-id：数据库标识，每个数据库标识必须唯一；
2. auto\_increment\_increment=2 ：这是循环镜像里最重要的参数之一，表示自动增量为2，就是指字段一次递增多少，这将允许最多2台数据库加入这个循环镜像的阵列，而自动递增字段不会重复。当然如果有10台主可以将这个值设成10。
3. auto\_increment\_offset=1 ：这是循环镜像里最重要的参数之一，表示自增字段的起始值，每个主数据库的偏移值必须唯一，且在1和auto\_increment\_increment之间（也就是说两个主数据库的这个设置分别是1和2）。

如果按以下设置，肯定不会出现这个问题，但如果业务有要求，ID必须连续，那就不能设置这两个参数了：

主1： 递增初始值是1，一次递增2。也就是值将是1、3、5.。。。  
auto-increment-increment=2   
auto-increment-offset=1   
主2： 递增初始值是2，一次递增2。也就是值将是2、4、6.。。。  
auto-increment-increment=2   
auto-increment-offset=2

这样配置之后，两台设备在生成auto\_increment的列时就是主1是1、3、5、7奇数；主2是2、4、6、8偶数。注意，生成的值不一定连续，最后这个字段有可能是，1、2、3、4、5、7、8、10；逻辑是这样，假如主1 执行insert 3条记录，则这个字段是1、3、5，主2再执行insert的时候，是从6开始，6、8、10。然后主1再11、13、15。

目录

[架构图及原理 1](#_Toc441915182)

[主机1的配置 1](#_Toc441915183)

[主机2的配置 2](#_Toc441915184)

## 架构图及原理

首先，配置mysql双主，见相关文档。

使用keepalived做高可用，当主设备发生主机故障时切换到备机。

然而keepalived只能做到对网络故障和keepalived本身的监控，即当出现网络故障或者keepalived本身出现问题时，进行切换。但是这些还不够，我们还需要监控keepalived所在服务器上的mysql进程，根据mysql进程的运行状态决定是否需要进行主备切换。这个时候，我们可以通过编写脚本对业务进程进行检测监控。当然我们可以通过添加更多脚本监控所有需要监控的点，触发主备切换。

\*注意\*这里两个主机都没有配置nopreempt，如果配置了nopreempt就必须在监控脚本中kill掉keepalived 否则不会发生切换。抢占模式有时候由于交换机的arp保持时间，发生切换的时候，可能会有几秒钟的IP不通的情况发生（一般2秒左右），对于数据库的操作可能会有3秒左右，具体需要实际测试。

## 主机1的配置

! Configuration File for keepalived

global\_defs {

notification\_email {

18609836313@wo.cn

cycwll@163.com

}

notification\_email\_from localhost@localhost.com

smtp\_server 127.0.0.1

smtp\_connect\_timeout 30

router\_id LVS\_DEVEL\_232

}

vrrp\_script chk\_mysql {

script "killall -0 mysqld"

interval 1

weight -15

}

vrrp\_instance VI\_1 {

state BACKUP

interface ens33

virtual\_router\_id 81

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 123qwe

}

virtual\_ipaddress {

10.34.34.239/24

}

}

## 主机2的配置

! Configuration File for keepalived

global\_defs {

notification\_email {

18609836313@wo.cn

cycwll@163.com

}

notification\_email\_from localhost@localhost.com

smtp\_server 127.0.0.1

smtp\_connect\_timeout 30

router\_id LVS\_DEVEL\_233

}

vrrp\_script chk\_mysql {

script "killall -0 mysqld"

interval 1

weight -15

}

vrrp\_instance VI\_1 {

state BACKUP

interface ens33

virtual\_router\_id 81

priority 90

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 123qwe

}

virtual\_ipaddress {

10.34.34.239/24

}

}

## binlog\_ignore\_db引起的同步复制故障

\*在binlog\_format 设置为ROW格式时，并且主服务器my.cnf配置文件设置了参数：binlog\_ignore\_db=xxx时会出现此问题。

将mysql的binlog\_fromat 设置为MIXED格式可以解决此问题。 slave服务器是可以同步执行-e 的操作的。

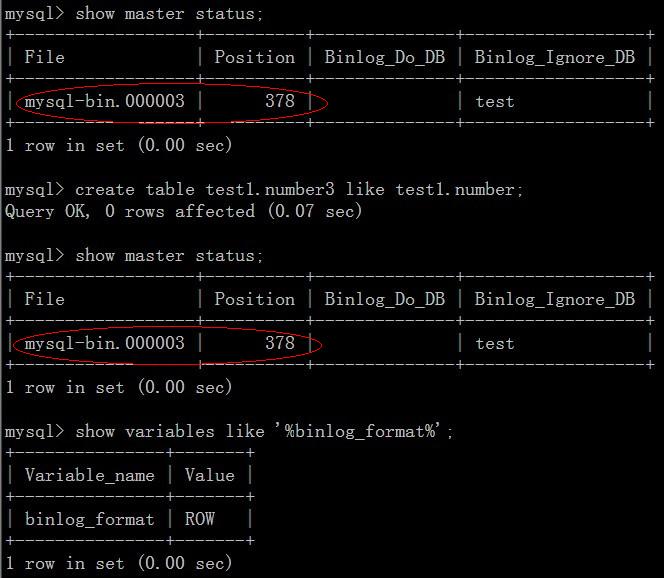
mysql -u root -p -e "insert into backup.test values(11)"

关于[MySQL的 binlog\_format 的几中类型详见本文最后的介绍(Mixed,Statement,Row)](http://blog.chinaunix.net/uid-167175-id-3516604.html)

今天一个同事跟我说了一个问题，"mysql master使用了binlog\_ignore\_db一个库以后，使用mysql -e 执行的所有语句就不写binlog了？"

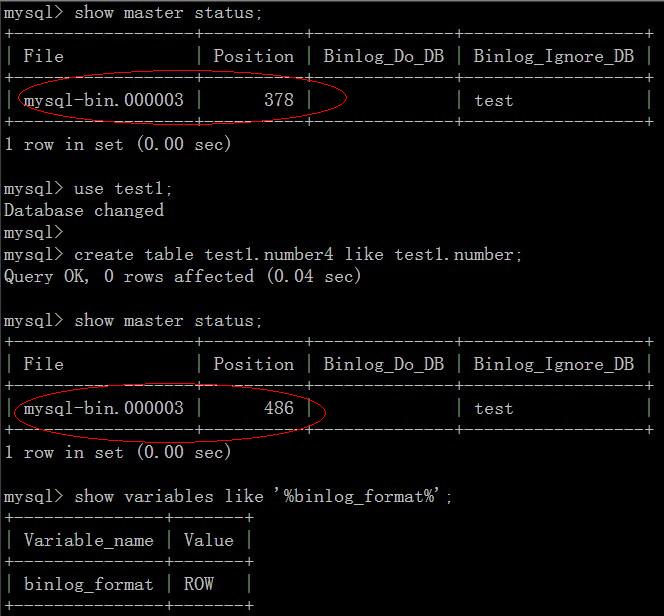
询问了他的情况，他是想在主从复制时，有一个库不复制，查了他的my.cnf配置，binlog格式化为row，跟他要了当时的语句，如下：

mysql -e "create table db.tb like db.tb1" 演示：



结果创建的表，Slave上一个都没有，导致杯具发生。

到底是什么原因引起的呢？那就是没有使用use 库名导致的，如果使用了，就可以记录binlog，如图：



所以，如果想在Slave上忽略一个库的复制，最好不要用binlog\_ignore\_db这个参数，使用replicate-ignore-db = yourdb，取代之。

[MySQL binlog\_format (Mixed,Statement,Row)](http://blog.chinaunix.net/uid-167175-id-3516604.html)

MySQL 5.5 中对于二进制日志 (binlog) 有 3 种不同的格式可选：Mixed,Statement,Row，默认格式是 Statement。总结一下这三种格式日志的优缺点。  
  
MySQL Replication 复制可以是基于一条语句 (Statement Level) ，也可以是基于一条记录 (Row Level)，可以在 MySQL 的配置参数中设定这个复制级别，不同复制级别的设置会影响到 Master 端的 bin-log 日志格式。

**1. Row**  
日志中会记录成每一行数据被修改的形式，然后在 slave 端再对相同的数据进行修改。

**优点**：在 row 模式下，bin-log 中可以不记录执行的 SQL 语句的上下文相关的信息，仅仅只需要记录那一条记录被修改了，修改成什么样了。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，非常容易理解。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**：在 row 模式下，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：

|  |  |
| --- | --- |
| 1 | **UPDATE** product **SET** owner\_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) ，而是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，实际上就是重建了整个表。那么该表的每一条记录都会被记录到日志中。

**2. Statement**  
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 进程会解析成和原来 master 端执行过的相同的 SQL 再次执行。

**优点**：在 statement 模式下，首先就是解决了 row 模式的缺点，不需要记录每一行数据的变化，减少了 bin-log 日志量，节省 I/O 以及存储资源，提高性能。因为他只需要记录在 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**：在 statement 模式下，由于他是记录的执行语句，所以，为了让这些语句在 slave 端也能正确执行，那么他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 端杯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，自然复制的时候涉及到越复杂的内容，bug 也就越容易出现。在 statement 中，目前已经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被正确复制，在存储过程中使用了 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 是基于每一行来记录的变化，所以不会出现类似的问题。

**3. Mixed**  
从官方文档中看到，之前的 MySQL 一直都只有基于 statement 的复制模式，直到 5.1.5 版本的 MySQL 才开始支持 row 复制。从 5.0 开始，MySQL 的复制已经解决了大量老版本中出现的无法正确复制的问题。但是由于存储过程的出现，给 MySQL Replication 又带来了更大的新挑战。另外，看到官方文档说，从 5.1.8 版本开始，MySQL 提供了除 Statement 和 Row 之外的第三种复制模式：Mixed，实际上就是前两种模式的结合。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 statment 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 模式也被做了优化，并不是所有的修改都会以 row 模式来记录，比如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。

**其他参考信息**

**除以下几种情况外，在运行时可以动态改变 binlog 的格式**：  
. 存储流程或者触发器中间；  
. 启用了 NDB；  
. 当前会话使用 row 模式，并且已打开了临时表；

**如果 binlog 采用了 Mixed 模式，那么在以下几种情况下会自动将 binlog 的模式由 statement 模式变为 row 模式**：  
. 当 DML 语句更新一个 NDB 表时；  
. 当函数中包含 UUID() 时；  
. 2 个及以上包含 AUTO\_INCREMENT 字段的表被更新时；  
. 执行 INSERT DELAYED 语句时；  
. 用 UDF 时；  
. 视图中必须要求运用 row 时，例如建立视图时使用了 UUID() 函数；

**设定主从复制模式**：

|  |  |
| --- | --- |
| 1  2  3  4 | log-bin=mysql-bin  #binlog\_format="STATEMENT"  #binlog\_format="ROW"  binlog\_format="MIXED" |

**也可以在运行时动态修改 binlog 的格式。例如**：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | mysql> **SET** **SESSION** binlog\_format = 'STATEMENT';  mysql> **SET** **SESSION** binlog\_format = 'ROW';  mysql> **SET** **SESSION** binlog\_format = 'MIXED';  mysql> **SET** GLOBAL binlog\_format = 'STATEMENT';  mysql> **SET** GLOBAL binlog\_format = 'ROW';  mysql> **SET** GLOBAL binlog\_format = 'MIXED'; |

**两种模式的对比**：  
**Statement 优点**  
历史悠久，技术成熟；  
产生的 binlog 文件较小；  
binlog 中包含了所有数据库修改信息，可以据此来审核数据库的安全等情况；  
binlog 可以用于实时的还原，而不仅仅用于复制；  
主从版本可以不一样，从服务器版本可以比主服务器版本高；

**Statement 缺点**：  
不是所有的 UPDATE 语句都能被复制，尤其是包含不确定操作的时候；  
调用具有不确定因素的 UDF 时复制也可能出现问题；  
运用以下函数的语句也不能被复制：  
\* LOAD\_FILE()  
\* UUID()  
\* USER()  
\* FOUND\_ROWS()  
\* SYSDATE() (除非启动时启用了 –sysdate-is-now 选项)  
INSERT … SELECT 会产生比 RBR 更多的行级锁；  
复制须要执行全表扫描 (WHERE 语句中没有运用到索引) 的 UPDATE 时，须要比 row 请求更多的行级锁；  
对于有 AUTO\_INCREMENT 字段的 InnoDB 表而言，INSERT 语句会阻塞其他 INSERT 语句；  
对于一些复杂的语句，在从服务器上的耗资源情况会更严重，而 row 模式下，只会对那个发生变化的记录产生影响；  
存储函数(不是存储流程 )在被调用的同时也会执行一次 NOW() 函数，这个可以说是坏事也可能是好事；  
确定了的 UDF 也须要在从服务器上执行；  
数据表必须几乎和主服务器保持一致才行，否则可能会导致复制出错；  
执行复杂语句如果出错的话，会消耗更多资源；

**Row 优点**  
任何情况都可以被复制，这对复制来说是最安全可靠的；  
和其他大多数数据库系统的复制技能一样；  
多数情况下，从服务器上的表如果有主键的话，复制就会快了很多；  
复制以下几种语句时的行锁更少：  
\* INSERT … SELECT  
\* 包含 AUTO\_INCREMENT 字段的 INSERT  
\* 没有附带条件或者并没有修改很多记录的 UPDATE 或 DELETE 语句  
执行 INSERT，UPDATE，DELETE 语句时锁更少；  
从服务器上采用多线程来执行复制成为可能；

**Row 缺点**  
生成的 binlog 日志体积大了很多；  
复杂的回滚时 binlog 中会包含大量的数据；  
主服务器上执行 UPDATE 语句时，所有发生变化的记录都会写到 binlog 中，而 statement 只会写一次，这会导致频繁发生 binlog 的写并发请求；  
UDF 产生的大 BLOB 值会导致复制变慢；  
不能从 binlog 中看到都复制了写什么语句(加密过的)；  
当在非事务表上执行一段堆积的 SQL 语句时，最好采用 statement 模式，否则很容易导致主从服务器的数据不一致情况发生；  
另外，针对系统库 MySQL 里面的表发生变化时的处理准则如下：  
如果是采用 INSERT，UPDATE，DELETE 直接操作表的情况，则日志格式根据 binlog\_format 的设定而记录；  
如果是采用 GRANT，REVOKE，SET PASSWORD 等管理语句来做的话，那么无论如何都要使用 statement 模式记录；  
使用 statement 模式后，能处理很多原先出现的主键重复问题；

## 常见错误

mysql主从复制，经常会遇到错误而导致slave端复制中断，这个时候一般就需要人工干预，跳过错误才能继续  
跳过错误有两种方式：  
1.跳过指定数量的事务：  
mysql>slave stop;  
mysql>SET GLOBAL SQL\_SLAVE\_SKIP\_COUNTER = 1        #跳过一个事务  
mysql>slave start  
  
**2.修改mysql的配置文件，通过slave\_skip\_errors参数来跳所有错误或指定类型的错误**  
vi /etc/my.cnf  
[mysqld]  
#slave-skip-errors=1062,1053,1146 #跳过指定error no类型的错误  
#slave-skip-errors=all #跳过所有错误

## [Mysql Binlog三种格式介绍及分析](http://blog.csdn.net/vhomes/article/details/8082734)

**一．Mysql Binlog格式介绍**

Mysql binlog日志有三种格式，分别为Statement,MiXED,以及ROW！

**1.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，提高性能。(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格式日志时应该跟据应用的实际情况，其所产生的日志量会增加多少，以及带来的IO性能问题。)

**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端执行时候相同 的结果。另外mysql 的复制,像一些特定函数功能，slave可与master上要保持一致会有很多相关问题(如sleep()函数， last\_insert\_id()，以及user-defined functions(udf)会出现问题).

使用以下函数的语句也无法被复制：

\* LOAD\_FILE()

\* UUID()

\* USER()

\* FOUND\_ROWS()

\* SYSDATE() (除非启动时启用了 --sysdate-is-now 选项)

同时在INSERT ...SELECT 会产生比 RBR 更多的行级锁

**2.Row:**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rowlevel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,比如一条update语句，修改多条记录，则binlog中每一条修改都会有记录，这样造成binlog日志量会很大，特别是当执行alter table之类的语句的时候，由于表结构修改，每条记录都发生改变，那么该表每一条记录都会记录到日志中。

**3.Mixedlevel:** 是以上两种level的混合使用，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.新版本的MySQL中队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。至于update或者delete等修改数据的语句，还是会记录所有行的变更。

**二.Binlog基本配制与格式设定**

**1.基本配制**

Mysql BInlog日志格式可以通过mysql的my.cnf文件的属性binlog\_format指定。如以下：

binlog\_format           = MIXED                 //binlog日志格式

log\_bin                  =目录/mysql-bin.log    //binlog日志名

expire\_logs\_days    = 7                //binlog过期清理时间

max\_binlog\_size    100m                    //binlog每个日志文件大小

**2.Binlog日志格式选择**

Mysql默认是使用Statement日志格式，推荐使用MIXED.

由于一些特殊使用，可以考虑使用ROWED，如自己通过binlog日志来同步数据的修改，这样会节省很多相关操作。对于binlog数据处理会变得非常轻松,相对mixed，解析也会很轻松(当然前提是增加的日志量所带来的IO开销在容忍的范围内即可)。

**3.mysqlbinlog格式选择**

mysql对于日志格式的选定原则:如果是采用 INSERT，UPDATE，DELETE 等直接操作表的情况，则日志格式根据 binlog\_format 的设定而记录,如果是采用 GRANT，REVOKE，SET PASSWORD 等管理语句来做的话，那么无论如何 都采用 SBR 模式记录

**三．Mysql Binlog日志分析**

通过MysqlBinlog指令查看具体的mysql日志，如下:

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

SET TIMESTAMP=1350355892/\*!\*/;

BEGIN

/\*!\*/;

# at 1643330

#121016 10:51:32 server id 1  end\_log\_pos 1643885        Query     thread\_id=272571   exec\_time=0   error\_code=0

SET TIMESTAMP=1350355892/\*!\*/;

Insert into T\_test….)

/\*!\*/;

# at 1643885

#121016 10:51:32 server id 1  end\_log\_pos 1643912        Xid = 0

COMMIT/\*!\*/;

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**1.开始事物的时间:**

SET TIMESTAMP=1350355892/\*!\*/;

BEGIN

**2.sqlevent起点**

#at 1643330 :为事件的起点，是以1643330字节开始。

**3.sqlevent 发生的时间点**

#121016 10:51:32:是事件发生的时间，

**4.serverId**

server id 1 :为master 的serverId

**5.sqlevent终点及花费时间，错误码**

end\_log\_pos 1643885:为事件的终点，是以1643885 字节结束。

execTime 0: 花费的时间

error\_code=0:错误码

Xid:事件指示提交的XA事务

**Mixed日志说明：**

在slave日志同步过程中，对于使用now这样的时间函数，MIXED日志格式，会在日志中产生对应的unix\_timestamp()\*1000的时间字符串，slave在完成同步时，取用的是sqlEvent发生的时间来保证数据的准确性。另外对于一些功能性函数slave能完成相应的数据同步，而对于上面指定的一些类似于UDF函数，导致Slave无法知晓的情况，则会采用ROW格式存储这些Binlog，以保证产生的Binlog可以供Slave完成数据同步。