29.MySQL主从架构之上的演变-GTID,(增强)半同步,过滤复制,延时从库√

一.主从复制演变-GTID（mysql5.7版本及之后推荐）

1.GTID简介

```
GTID (Global Transaction ID)是全局事务ID,当在主库上提交事务或者被从库应用时，可以定位和追踪每一个事务。
保证了每个在主库上提交的事务在集群中有一个唯一的ID。
这种方式强化了数据库的主备一致性，故障恢复以及容错能力。
核心特性：全局唯一，具备幂等性。重复性的操作不会再运行
```

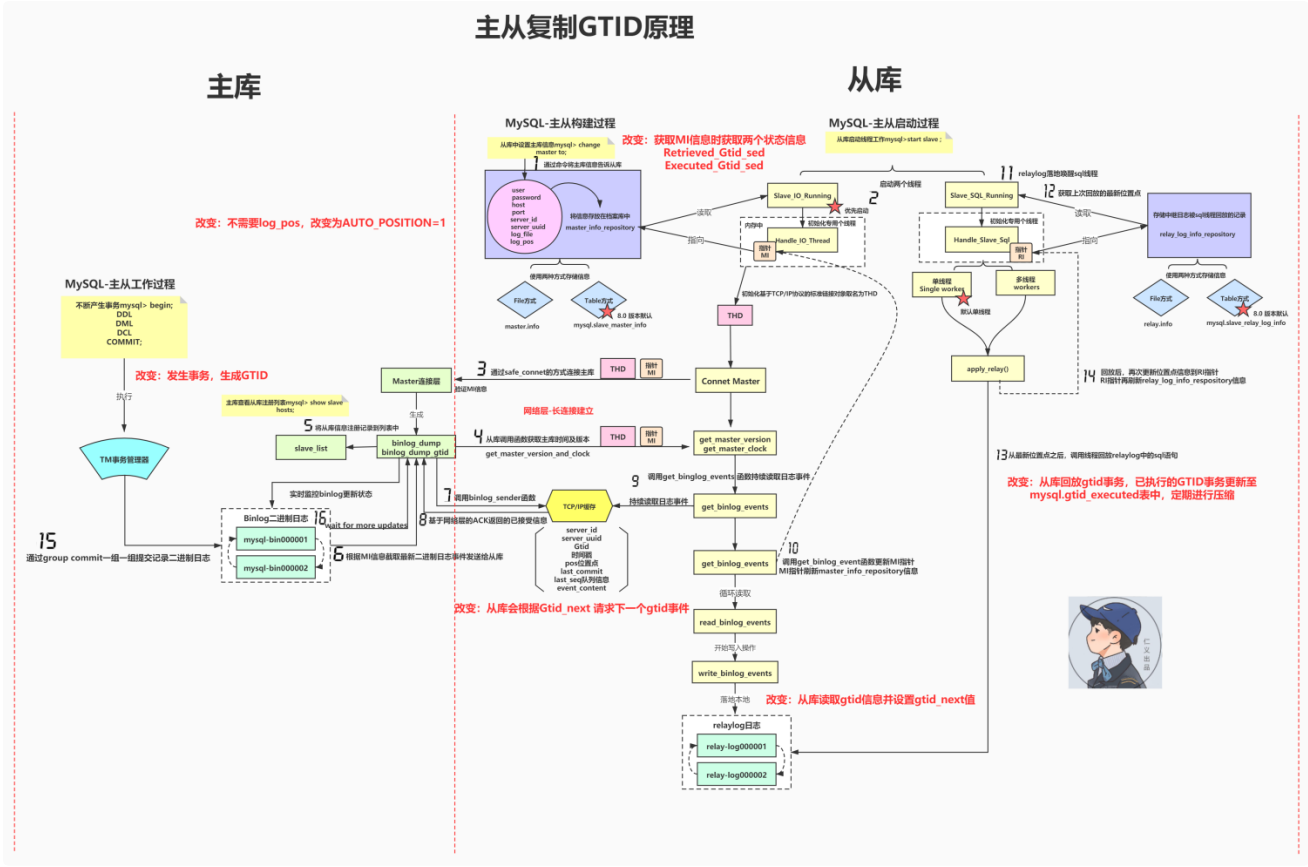
2.GTID核心参数

```
gtid-mode=on                        启用GTID模式，不启动就是普通的复制结构
enforce-gtid-consistency=true       强制GTID的一致性
log-slave-updates=1                 强行从库记录binlog日志
主从架构环境下备份mdp 不要加--set-gtid-purged=OFF
因为这个参数会在备份文件中显示主库备份过来的gtid号，从库会根据备份文件中的global.gtid_purged来决定从库gtid从哪里开始。
```

2.GTID复制工作原理（改变）

```
gtid格式 ：
server_uuid:seq
a. MASTER 发生事务,生成GTID,记录binlog。
b. slave  在MI中包含Retrieved_Gtid_sed , Executed_Gtid_sed 这两个参数信息
          根据gtid_next请求下一个GTID的事件。
c. Master 发送 binlog到SLAVE.
d. slave 接收二进制日志，存储至relaylog，读取gtid信息并设置gtid_next值。
e. slave SQL 回放GTID事务。
  1. 检测本地binlog是否有该GTID
  2. 应用事务，并更新本地binlog(log_slave_updates)
注：
已执行的gtid信息更新至mysql.gtid_executed表中，并定期进行压缩。（reset master 会清空此表）
```

3.原理图



4.传统主从模式演变“GTID”（5.7版本及之后建议）

4.1 重新推倒搭建传统主从架构

清空原有环境

```
各个节点上做如下操作
pkill mysqld
rm -rf /data/3306/*
mkdir -p /data/3306/data /data/3306/binlog
chown -R mysql.mysql /data/*
```

准备配置文件

Bash | Copy

```
配置文件先注释掉配置gtid的参数
主库db01:
cat > /etc/my.cnf <<EOF
[mysqld]
basedir=/usr/local/mysql
datadir=/data/3306/data
socket=/tmp/mysql.sock
server_id=51
port=3306
secure-file-priv=/tmp
log_bin=/data/3306/binlog/mysql-bin
binlog_format=row
#gtid-mode=on
#enforce-gtid-consistency=true
#log-slave-updates=1
[mysql]
prompt=db01 [\\d]>
EOF

slave1(db02):
cat > /etc/my.cnf <<EOF
[mysqld]
basedir=/usr/local/mysql
datadir=/data/3306/data
socket=/tmp/mysql.sock
server_id=52
port=3306
secure-file-priv=/tmp
log_bin=/data/3306/binlog/mysql-bin
binlog_format=row
#gtid-mode=on
#enforce-gtid-consistency=true
#log-slave-updates=1
[mysql]
prompt=db02 [\\d]>
EOF

slave2(db03):
cat > /etc/my.cnf <<EOF
[mysqld]
basedir=/usr/local/mysql
datadir=/data/3306/data
socket=/tmp/mysql.sock
server_id=53
port=3306
secure-file-priv=/tmp
log_bin=/data/3306/binlog/mysql-bin
binlog_format=row
#gtid-mode=on
#enforce-gtid-consistency=true
```

```
#log-slave-updates=1
[mysql]
prompt=db03 [\\d]>
EOF
```

初始化数据

Bash | Copy

```
各个节点上做如下操作
mysqld --initialize-insecure --user=mysql --basedir=/usr/local/mysql --datadir=/data/3306/data
```

重启数据库

Bash | Copy

```
各个节点上做如下操作
/etc/init.d/mysqld start
```

主库创建用户

Bash | Copy

```
mysql -e "create user repl@'10.0.0.%' identified with mysql_native_password by '123';grant replication slave or
mysql -e "create user root@'10.0.0.%' identified with mysql_native_password by '123';grant all on *.* to root@
```

从库用mdp全备主库，并恢复

Bash | Copy

```
从库远程连接备份主库
mysqldump -uroot -p123 -h 10.0.0.51 -P 3306 -A --master-data=2 --single-transaction -R -E --triggers >/tmp/ful
从库恢复主库数据
mysql -e "source /tmp/full.sql"
```

启动主从复制

```
0.获取起点
[root@db02 ~]# grep  "\--\ CHANGE MASTER" /tmp/full.sql
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=1187;
1.从库设置起点
[root@db02 ~]#mysql -e \
"CHANGE MASTER TO \
  MASTER_HOST='10.0.0.51',\
  MASTER_USER='repl', \
  MASTER_PASSWORD='123', \
  MASTER_PORT=3306, \
  MASTER_LOG_FILE='mysql-bin.000001', \
  MASTER_LOG_POS=1187, \
  MASTER_CONNECT_RETRY=10;"
GTID模式:
从库设置复制起点
[root@db03 data]# mysql -e \
"CHANGE MASTER TO \
  MASTER_HOST='10.0.0.51',\
  MASTER_USER='repl', \
  MASTER_PASSWORD='123', \
  MASTER_PORT=3306, \
  MASTER_AUTO_POSITION=1;"

 2.启动专用线程
 [root@db02 ~]# mysql -e "start slave;"
 3.查看线程状态
 [root@db02 ~]#  mysql -e "show slave status \G"|grep "Running:"
              Slave_IO_Running: Yes
              Slave_SQL_Running: Yes
```

4.2.开始传统主从结构到GTID模式的转变

检查各个节点的gtid是否开启

```
[root@db01 ~]# mysql -e "select @@enforce_gtid_consistency;"
+--------------------------+
| @@enforce_gtid_consistency |
+--------------------------+
| OFF                      |
+--------------------------+
[root@db01 ~]# mysql -e "select @@gtid_mode;"
+-------------+
| @@gtid_mode |
+-------------+
| OFF         |
+-------------+
```

各个节点修改enforce_gtid_consistency

```bash
mysql -e "set global enforce_gtid_consistency =warn;"
注意:
 所有节点均先将其修改为 WARN,同时注意查看日志是否出现警告信息，生产环境想调整为GTID模式时，需提前一段时间调整此参数，观察一段时间，
 开启后观察数据库日志，只有在无警告的情况下才可以进行下一步的操作。
```

各个节点修改enforce_gtid_consistency = on

强制一致性

```bash
mysql -e "set global enforce_gtid_consistency = on;"
mysql -e "select @@enforce_gtid_consistency;"
+--------------------------------+
| @@enforce_gtid_consistency |
+--------------------------------+
| ON                             |
+--------------------------------+
```

各个节点修改gtid_mode = off_permissive

没有开启前的旧事务默认是匿名的，开启之后可以支持匿名事务的复制。可以同时兼容新旧事务。

```bash
mysql -e "set global gtid_mode = off_permissive;"
```

各个节点修改gtid_mode=on_permissive

开启这个参数后，新事物开始基于GTID。同时允许复制的事务为匿名和GTID的。

```bash
mysql -e "set global gtid_mode=on_permissive;"
```

各从节点检查剩余事务数

当匿名旧事物全部复制结束后，所有新事务基于GTID。保证GTID的兼容性

```
mysql -e  "show status like 'ongoing_anonymous_transaction_count';"
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Ongoing_anonymous_transaction_count | 0     |
+---------------------------------------+-------+
当查看剩余值为0时，进行下一步
手动刷新日志，开始新的日志记录
mysql -e  "flush logs;"
```

各个节点启用gtid_mode（先主后从）

```
mysql -e  "set global gtid_mode=on;"
```

从库在线切换复制模式

传统主从基于postion号切换到基于GTID号

```
mysql -e "stop slave; change master to master_auto_position=1;start slave;"
```

主库模拟事务操作，查看从库状态是否有GTID记录

```
主库:
create database test;
从库查看:
mysql -e "show slave status \G"
Retrieved_Gtid_Set: 389ea959-b194-11eb-a24b-000c29edc386:1
Executed_Gtid_Set: 389ea959-b194-11eb-a24b-000c29edc386:1

```

各个节点修改配置文件永久生效

```
gtid-mode=on
enforce-gtid-consistency=true
log-slave-updates=1
```

二.主从复制演变–(增强)半同步（半同步mysql5.5版本出现，5.6+gc技术开始使用 5.7增强半同步）

主从数据最终一致性的保证

1.预备知识

1.1 预备知识–2pc机制

Bash | Copy

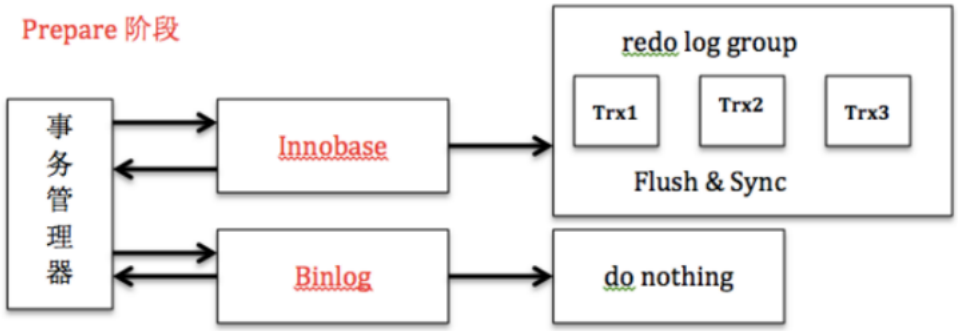
```
2PC机制
  客户端下发commit命令，此时进入真正的两阶段提交，两阶段提交分为prepare和commit两个阶段
## prepare阶段:
  prepare分为binlog prepare和innobase redo prepare,其中binlog prepare几乎不做操作，innobase prepare会更新事务状态。

# commit阶段:
  commit阶段被分为了三个阶段，分别是flush，sync，commit。其中 flush操作会进行线程binlog cache的文件写入，再将用户binlog cache刷新到文件；sync操作负责binlog的落盘；commit操作负责更新server层的最大事务提交数量(和并行复制相关),然后innobase 再次更新事务状态。提交结束。
```

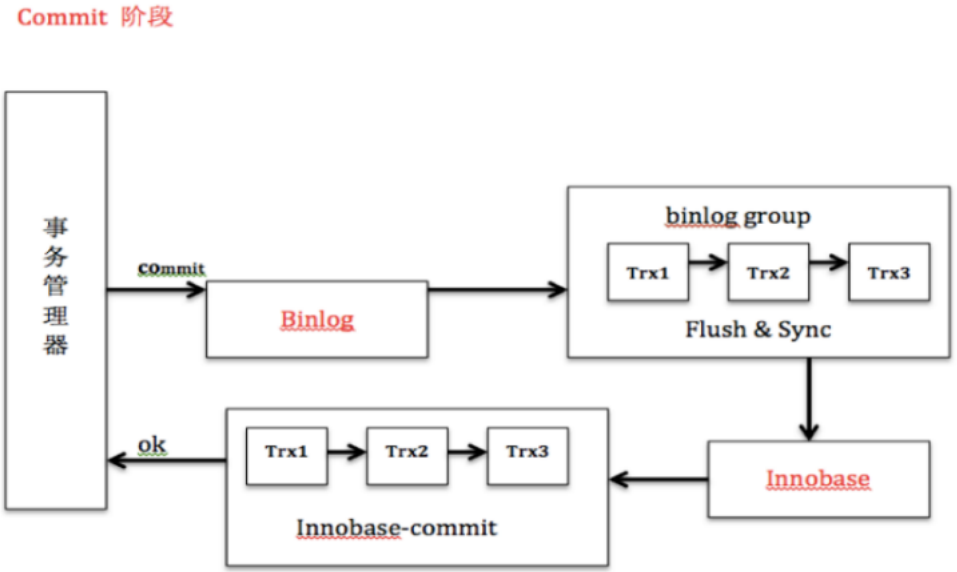
1.2 预备知识–group commit

Bash | Copy

```
group commit流程:
## FLUSH 阶段:
1)      持有Lock_log mutex [leader持有，follower等待]
2)      获取队列中的一组binlog(队列中的所有事务)
3)      将binlog buffer到OS cache
4)      通知dump线程dump binlog

## SYNC阶段
1)      释放Lock_log mutex，持有Lock_sync mutex[leader持有，follower等待]
2)      将一组binlog 落盘(sync动作，最耗时，假设sync_binlog为1)

## COMMIT阶段
1)      释放Lock_sync mutex，持有Lock_commit mutex[leader持有，follower等待]
2)      遍历队列中的事务，逐一进行innodb commit
3)      释放Lock_commit mutex
4)      唤醒队列中等待的线程

配套参数:
binlog_group_commit_sync_no_delay_count=N
binlog_group_commit_sync_delay=M
```

1.3 2pc机制和group commit原理图


Prepare 阶段
事务管理器
Innobase
Binlog
redo log group
Trx1
Trx2
Trx3
Flush & Sync
do nothing


Commit 阶段
事务管理器
commit
Binlog
binlog group
Trx1
Trx2
Trx3
Flush & Sync
ok
Trx1
Trx2
Trx3
Innobase-commit
Innobase

2.半同步和增强半同步-工作原理

2.1 半同步(after_commit)

Bash | Copy

```
提交后：
主服务器将每个事务写入其二进制日志和从服务器，同步二进制日志，并将事务提交到存储引擎。
主服务器在提交后等待从服务器对事务接收的确认。
在接收到确认后，主机将结果返回给客户机，然后客户机可以继续。
```

2.2 增强半同步(after_sync)

Bash | Copy

```
主进程将每个事务写入其二进制日志和从属服务器，并将二进制日志同步到磁盘。
主服务器等待同步后从服务器确认事务接收。
在收到确认后，主服务器将事务提交给存储引擎，并将结果返回给客户端，然后客户端可以继续。
```

2.3 半同步与增强半同步-原理图

2.4 半同步与增强半同步-版本使用情况

Bash | Copy

```
5.5开始支持半同步复制,但是没有GC（group commit）机制,性能极差,几乎没人使用。
5.6 版本时 ,加入了GC机制,半同步复制开始被接受.使用的是半同步（after_commit）机制,但是是在redo commit之后进行等待ACK确认.
这里会有一个痛点,如果主库redo commit阶段宕机宕机了,从库又获取到了binlog,会出现从库比主库数据"多"的问题.导致数据不一致.
5.7版本+以后,加入了增强半同步（after_sync）机制,在binlog commit(binlog sync disk)阶段,等待从库ACK,不管谁宕机,都能保证最终一

另外:
不管哪种方式,还会出现,如果ACK超时,会被切换为异步复制的模式.还是有数据不一致的风险.
如果公司能容忍,可以使用这种架构,建议使用增强半同步+GTID模式.
如果不能容忍,可以使用MGR  PXC .
```

2.5 半同步与增强半同步-配置

2.5.1 加载插件

```
生产中的高可用环境中建议主从插件都加载
主
db01 [(none)]>INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
从
db02 [(none)]>INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

2.5.2 查看是否加载成功

```
show plugins;
 | rpl_semi_sync_master            | ACTIVE   | REPLICATION        | semisync_master.so | GPL
 | rpl_semi_sync_slave             | ACTIVE   | REPLICATION        | semisync_slave.so | GPL
```

2.5.3 开启加载插件

```
主
db01 [(none)]> SET GLOBAL rpl_semi_sync_master_enabled = 1;
从
db02 [(none)]>SET GLOBAL rpl_semi_sync_slave_enabled = 1;
```

2.5.4 重启从库上的IO线程

```
db02 [(none)]>STOP SLAVE IO_THREAD;
db02 [(none)]>START SLAVE IO_THREAD;
```

2.5.4 查看是否在运行

```
主
db01 [(none)]>show status like 'Rpl_semi_sync_master_status';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Rpl_semi_sync_master_status | ON    |
+------------------------------+-------+
从
db02 [(none)]>show status like 'Rpl_semi_sync_slave_status';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Rpl_semi_sync_slave_status | ON    |
+-----------------------------+-------+
```

2.5.5 其他的优化参数 （主从节点都设置）

```
db01 [(none)]>show variables like '%semi%';
+-------------------------------------------+------------+
| Variable_name                             | Value      |
+-------------------------------------------+------------+
| rpl_semi_sync_master_enabled              | ON         |  是否开启
| rpl_semi_sync_master_timeout              | 10000      |  ack等待时间 默认10000ms(10s)
| rpl_semi_sync_master_trace_level          | 32         |
| rpl_semi_sync_master_wait_for_slave_count | 1          |  多少台返回ACK
| rpl_semi_sync_master_wait_no_slave        | ON         |
| rpl_semi_sync_master_wait_point           | AFTER_SYNC |  等待点
+-------------------------------------------+------------+
```

2.5.6 加载group commit (组提交)参数（主从节点都设置）

```
db01 [(none)]>set global binlog_group_commit_sync_delay=1;
db01 [(none)]>set global binlog_group_commit_sync_no_delay_count=1000;
```

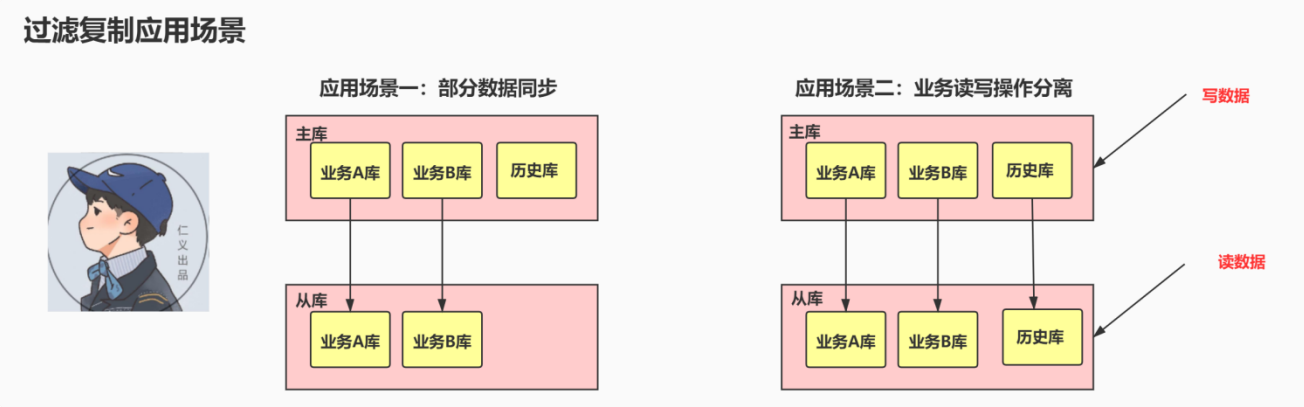
三.主从复制演变–过滤复制

1.简介

```
选择 需求的库或者表进行选择性复制
```

2.应用场景

```
1.业务的分离.
2.部分数据同步.
```



3.过滤复制实现配置操作

3.1 主库方面

```
是否记录binlog来控制
db01 [(none)]>show master status;
binlog_do_db     :  白名单
binlog_ignore_db :  黑名单
白名单和黑名单配置一个名单控制即可
```

3.2 从库方面

```
sql线程是否回放来控制
db02 [(none)]>show slave status\G;
配置文件书写格式
库级别
replicate_do_db=world（参数后只能书写一个，控制多库写多行参数）
repSlicate_ignore_db=test
表级别
replicate_do_table=world.city
replicate_ignore_table:test.t100w
模糊设置
replicate_wild_do_table=oldboy.t*
replicate_wild_ignore_table=oldguo.t*
三组配置配置一组即可控制
```

3.3 从库配置过滤复制实列

3.3.1 关闭从库slave_sql线程

首先保证主从环境是正常可以使用的

```
db02 [(none)]>stop slave sql_thread;
```

3.3.2 设置库级别的白名单

```
针对数据库实列下的cry库设置白名单，也就是从库只针对cry库进行复制
1.在线设置（无需重启，立即生效）
db02 [(none)]>change replication filter replicate_do_db= (cry);
2.永久生效加入配置文件中
vim /etc/my.cnf
[mysqld]
replicate_do_db=cry;
```

3.3.3 查看从库配置结果

```
db02 [(none)]>show slave status\G;
Replicate_Do_DB: cry
```

3.3.4 开启从库slave_sql线程

```
db02 [(none)]>start slave sql_thread;
```

3.3.5 测试（1.主库对cry库操作，从库是否会复制？）

1.主库对cry库操作，从库是否会复制？------结果从库会复制

```
1.主库对cry库进行操作
1.1 cry库下创建表
db01 [cry]>create table d(id int);
1.2 对创建表的插入数据
db01 [cry]>insert into d values(1);
2.从库查看cry库下是否复制数据
db02 [(none)]>select * from cry.d;
+------+
| id   |
+------+
|    1 |
+------+
结果从库会复制
```

2.主库对除cry的其他库操作，从库是否会复制？------结果从库不会复制

```
1.主库对除cry之外的其他库操作
1.1 创建一个新库
db01 [cry]>create database cry1;
2.从库查看是否复制数据
db02 [(none)]>show databases;
+--------------------+
| Database           |
+--------------------+
| cry                |
| error              |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
| xxx                |
+--------------------
结果从库不会复制
```

3.4 取消过滤复制配置操作

```
1.停止线程
db02 [(none)]>stop slave sql_thread;
2.取消配置
db02 [(none)]>CHANGE REPLICATION FILTER REPLICATE_DO_DB = ();
3.开启线程
db02 [(none)]>start slave sql_thread;
```

四.主从复制演变–延时从库

1.介绍

```
是我们人为配置的一种特殊从库，人为配置从库复制操作与主库延时'N'小时
```

2.延时从库作用

```
首先我们数据库损坏分为物理损坏和逻辑损坏。
普通的主从结构和高可用架构可以很好解决物理损坏，可是没办法解决逻辑损坏。
所以设置延时从库，当主库发生逻辑损坏，我们在设置的延时时间里只需要通过延时从库中的日子恢复主库数据，不用全盘恢复主库，节省了时间，提高
```

3.延时从库原理及配置参数

3.1 原理

```
SQL线程延时:数据已经写入relaylog中了,SQL线程"慢点"运行
一般企业建议3-6小时,具体看公司运维人员对于故障的反应时间。
```

3.2 配置(从库操作)

3.2.1 关闭从库线程

```
db02 [(none)]>stop slave;
```

3.2.2 配置延时从库参数

```
延时时间设置为300秒
db02 [(none)]>CHANGE MASTER TO MASTER_DELAY = 300;
```

3.2.3 开启从库线程

```
db02 [(none)]>start  slave;
```

3.2.4 查看从库线程

```
db02 [(none)]>show slave status\G;
SQL_Delay: 300                人为设置的延时时间
SQL_Remaining_Delay: NULL     距离从库复制主库操作还有多长时间
```

4.延时从库的应用---如何恢复?

4.1 故障恢复思路

```
a. 发现问题
b. 停掉从库线程（或者针对性的停止sql线程）
c. 控制SQL回放日志的截止位置点.
d. 找回数据,快速恢复业务
```

4.2 故障模拟

```
1.创建库
create database relaydb;
2.创建表
use relaydb
create table t1(id int);
3.表中插入数据
insert into t1 values(1),(2),(3);
commit;
insert into t1 values(11),(12),(13);
commit;
insert into t1 values(111),(112),(113);
commit;
4.模拟误操作
mysql> drop database relaydb;
```

4.3 情景恢复

Bash | Copy

```
1.停止从库sql线程
db02 [(none)]>stop slave sql_thread;
2.将从库的延时时间设置为0，方便我们的恢复操作不需要等待
db02 [(none)]>change master to master_delay=0;
3.查看此时的从库线程
db02 [(none)]>show slave status\G;
Slave_IO_Running: Yes
Slave_SQL_Running: No
SQL_Delay: 0
SQL_Remaining_Delay: NULL
4.控制SQL回放日志的截止位置点.
4.1 分析relaylog日志
db02 [(none)]>db02 [(none)]>show relaylog events in 'db02-relay-bin.000002';
+----------------------+------+----------------+-----------+-------------+-------------------------------
| Log_name             | Pos  | Event_type     | Server_id | End_log_pos | Info
+----------------------+------+----------------+-----------+-------------+-------------------------------
| db02-relay-bin.000002 |    4 | Format_desc    |        52 |         125 | Server ver: 8.0.20, Binlog ver
| db02-relay-bin.000002 |  125 | Previous_gtids |        52 |         156 |
| db02-relay-bin.000002 |  156 | Rotate         |        51 |           0 | mysql-bin.000006;pos=1383
| db02-relay-bin.000002 |  203 | Format_desc    |        51 |           0 | Server ver: 8.0.20, Binlog ver
| db02-relay-bin.000002 |  324 | Gtid           |        51 |        1460 | SET @@SESSION.GTID_NEXT= 'b02c
| db02-relay-bin.000002 |  401 | Query          |        51 |        1577 | create database relaydb /* xid
| db02-relay-bin.000002 |  518 | Gtid           |        51 |        1654 | SET @@SESSION.GTID_NEXT= 'b02c
| db02-relay-bin.000002 |  595 | Query          |        51 |        1771 | use `relaydb`; create table t1
| db02-relay-bin.000002 |  712 | Gtid           |        51 |        1850 | SET @@SESSION.GTID_NEXT= 'b02c
| db02-relay-bin.000002 |  791 | Query          |        51 |        1928 | BEGIN
| db02-relay-bin.000002 |  869 | Table_map      |        51 |        1979 | table_id: 164 (relaydb.t1)
| db02-relay-bin.000002 |  920 | Write_rows     |        51 |        2029 | table_id: 164 flags: STMT_END_
| db02-relay-bin.000002 |  970 | Xid            |        51 |        2060 | COMMIT /* xid=837 */
| db02-relay-bin.000002 | 1001 | Gtid           |        51 |        2139 | SET @@SESSION.GTID_NEXT= 'b02c
| db02-relay-bin.000002 | 1080 | Query          |        51 |        2217 | BEGIN
| db02-relay-bin.000002 | 1158 | Table_map      |        51 |        2268 | table_id: 164 (relaydb.t1)
| db02-relay-bin.000002 | 1209 | Write_rows     |        51 |        2318 | table_id: 164 flags: STMT_END_
| db02-relay-bin.000002 | 1259 | Xid            |        51 |        2349 | COMMIT /* xid=839 */
| db02-relay-bin.000002 | 1290 | Gtid           |        51 |        2428 | SET @@SESSION.GTID_NEXT= 'b02c
| db02-relay-bin.000002 | 1369 | Query          |        51 |        2506 | BEGIN
```

GTID%EF%BC%88mysql5.7%E7%89%88%E6%9C%AC%E5%8F%8A%E4%B9%8B%E5%90%8E%E6%8E%A8%E8%8D%90%EF%BC%891.GTID%E7%AE%80%E4%BB%8BGTID%20(Global%20Transaction%20Id