day04-MySQL主从架构设计（重点）

一、主从架构概述

1. 场景说明

某同学刚入职公司，在熟悉公司业务环境的时候，发现他们的数据库架构是一主两从，但是两台从数据库和主库不同步。询问得知，已经好几个月不同步了，但是每天会全库备份主服务器上的数据到从服务器上，由于数据量不是很大，所以一直没有人处理主从不同步的问题。这次正好问到了，于是乎就安排该同学处理一下这个主从不同步的问题。

> 主服务器对外提供业务数据，负责业务数据的增删改查操作。从服务器默认不对外提供服务，和主服务器一样，都处于长时间运行状态，在运行过程中，从服务器会自动从主服务器拉取并同步数据，提供了一个在线热备解决方案。

2. 主从架构学习目标

① 熟悉MySQL数据库常见的主从架构

② 理解MySQL主从架构的实现原理（背诵、记忆）

③ 掌握MySQL主从架构的搭建（重点掌握）

3. 什么是主从复制?

主从复制可以实现将数据从一台数据库服务器（master）复制到一台到多台数据库服务器(slave)

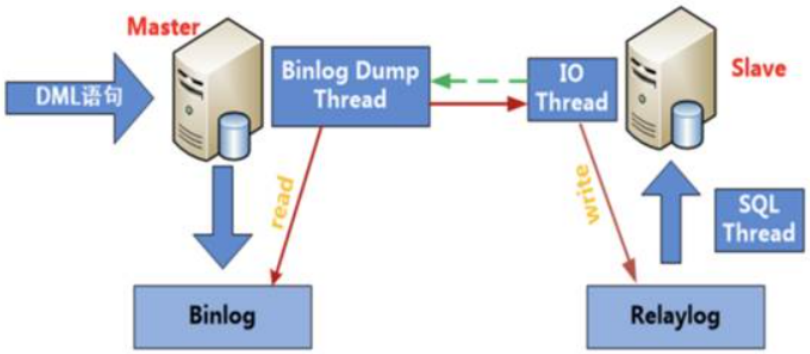
默认情况下，属于异步复制，所以无需维持长连接

解决问题：① 数据实时备份 ② 缓解服务器压力（读操作可以分散到slave服务器）=> MyCAT（读写分离软件）

简单来说，master将数据库的改变写入二进制日志，slave同步这些二进制日志，并根据这些二进制日志进行数据重演操作，实现数据异步同步。

> 同步复制：从服务器拉取主服务器的数据时，主服务器增删改数据时，从服务器必须马上同步，等待从服务器同步完成后，主服务器才能继续新的事务操作。优点：两端数据高度一致；缺点：阻塞主服务器的事务操作
>
> 异步复制：从服务器拉取主服务器的数据时，主服务器增删改数据时，从服务器可以异步复制，等待空闲时间在进行拉取，在这个过程中，不会阻塞主服务器业务。优点：不会阻塞主服务器的事务操作；缺点：可能会出现主从同步延迟的情况。

4. MySQL复制原理（背诵）



> binlog二进制日志，relaylog重写日志（负责把主服务器的DML在slave服务器重写执行一遍）
>
> binlog保存了用户对数据库的增删改事务操作（SQL语句）、relaylog重写日志（中继日志），当从服务器从主服务器拉取到二进制日志数据时，会首先写入到relaylog重写日志中。

mysqldump --single-transaction --master-data

详细描述：

前提：主服务器开启binlog二进制日志，从服务器开启relaylog中继日志。

① slave端的IO线程发送请求给master端的binlog dump线程

② master端binlog dump线程获取二进制日志信息(文件名和位置信息)发送给slave端的IO线程

③ salve端IO线程获取到的内容依次写到slave端relay log里，并把master端的bin-log文件名和位置记录到master.info里

④ salve端的SQL线程，检测到relay log中内容更新，就会解析relay log里更新的内容，并执行这些操作，从而达到和master数据一致

> master：主服务器；slave：从服务器。

> 注：主从复制也是备份的一种，属于在线热备。到这里就学过3种备份了：逻辑备份、物理备份、在线热备。

二、传统主从复制（AB复制）设计

1. MySQL主从复制环境准备

传统AB复制架构(M-S)，说明：mysql数据库，版本为8.0.40

环境说明：

IP	主机名	角色
192.168.88.101	node1.itcast.cn	master(主)
192.168.88.102	node2.itcast.cn	slave(从)

安装前准备：① 配置IP、主机名 ② 配置IP与主机映射 => /etc/hosts ③ 关闭防火墙与SELinux ④ 时间同步 ⑤ 安装必备软件，如vim、wget、rsync

设置主机名称

```
代码块
1   hostnamectl set-hostname node1.itcast.cn
2   hostnamectl set-hostname node2.itcast.cn
3
4   su或者bash指令
```

配置IP与主机映射

```
1   vim /etc/hosts
2   尾部追加如下内容
3   192.168.88.101 node1 node1.itcast.cn
4   192.168.88.102 node2 node2.itcast.cn
```

关闭防火墙与SELinux

```
代码块
1   systemctl stop firewalld
2   systemctl disable firewalld
3
4   setenforce 0
5   vim /etc/selinux/config
```

安装一些依赖软件（系统必备软件）

```
1   dnf install vim wget rsync -y
```

2. 搭建主从复制思路

1. master、slave必须安装相同版本的mysql数据库软件
2. master端必须开启二进制日志；slave端必须开启relay log中继日志
3. master端和slave端的server-id号不能一致 => my.cnf => server-id = 10
4. 同步master端数据之前，要删除data数据目录下的auto.cnf文件 => uuid编号（每个mysql实例都是唯一的）

master => mysql(uuid编号 => 数据库初始化自动生成) => /export/server/mysql/data/auto.cnf

slave数据目录是把master中的data同步过来，导致两个MySQL公用同一个uuid编号

5. slave端配置向master来同步数据
    - master端必须创建一个复制用户
    - 保证master和slave端初始数据一致
    - 配置主从复制（slave端）

3. MySQL8主从复制实践

3.1 配置Master主服务器

既要安装MySQL软件，也要初始化MySQL数据库

```
#!/bin/bash
#1.安装依赖软件
echo "正在安装依赖软件..."
yum -y install libaio &> /dev/null
if [ $? -ne 0 ];then
    echo "libaio安装失败"
    exit 1
fi
#2.判断是否有压缩包，如果有，则执行解压缩操作
echo "正在判断是否有压缩包，如果有进行解压缩操作..."
if [ -f mysql-8.0.40-linux-glibc2.17-x86_64.tar.xz ]; then
    tar -xf mysql-8.0.40-linux-glibc2.17-x86_64.tar.xz
    ls -l mysql-8.0.40-linux-glibc2.17-x86_64
fi
#3.判断系统中是否安装过mariadb软件，如果有对其进行卸载操作
echo "正在判断系统中是否安装过mariadb软件，如果有对其进行卸载操作..."
rpm -qa | grep mariadb | xargs -r dnf remove -y
[ -f /etc/my.cnf ] && rm -rf /etc/my.cnf
#4.创建mysql系统账号
id mysql &> /dev/null
[ $? -ne 0 ] && useradd -r -s /sbin/nologin mysql
#5.创建/export/server目录，然后移动mysql压缩包解压后的文件到/export/server目录下
rm -rf /export/server
mkdir -p /export/server
mv mysql-8.0.40-linux-glibc2.17-x86_64 /export/server/mysql
#6.进入mysql目录，对其进行初始化操作
echo "正在进入mysql目录，对其进行初始化操作..."
cd /export/server/mysql
bin/mysqld --initialize --user=mysql --basedir=/export/server/mysql --
datadir=/export/server/mysql/data 2>&1 | tee /tmp/mysqld.log | grep password
```

```
    | awk '{print $NF}' > /tmp/mysql_temp_password.txt
#7.设置ssl加密传输连接
bin/mysql_ssl_rsa_setup --datadir=/export/server/mysql/data &> /dev/null
#8.设置my.cnf与mysqld.service文件
echo "正在设置my.cnf与mysqld.service文件..."
cat > /etc/my.cnf <<EOF
[mysqld]
port=3306
basedir=/export/server/mysql
datadir=/export/server/mysql/data
socket=/tmp/mysql.sock
character_set_server=utf8
collation-server=utf8_unicode_ci
EOF

cat > /etc/systemd/system/mysqld.service <<EOF
[Unit]
Description=MySQL Server
After=network.target

[Service]
User=mysql
Group=mysql
Type=forking

# MySQL 执行命令及路径
ExecStart=/export/server/mysql/bin/mysqld --daemonize --pid-
file=/export/server/mysql/data/mysqld.pid
ExecStop=/export/server/mysql/bin/mysqladmin --defaults-
file=/export/server/mysql/my.cnf shutdown

# Ensure MySQL has sufficient time to start up
TimeoutSec=600

# PID 文件路径
PIDFile=/export/server/mysql/data/mysqld.pid

# Enable these options to auto-restart the service if it crashes
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
#9.刷新后台服务，然后启动mysqld
echo "正在刷新后台服务，然后启动mysqld..."
systemctl daemon-reload
```

```
systemctl start mysqld
systemctl enable mysqld
#10.重置mysql管理员密码为123456
echo "正在重置mysql管理员密码..."
cd /export/server/mysql
temp_password=`cat /tmp/mysql_temp_password.txt`
bin/mysqladmin -uroot password '123456' -p$temp_password
#11.把mysql的bin目录添加到环境变量中
echo 'export PATH=$PATH:/export/server/mysql/bin' >> /etc/profile
source /etc/profile
#12.解决mysql客户端首次无法登录问题
[ ! -f /lib64/libncurses.so.5 ] && ln -s /lib64/libncurses.so.6
/lib64/libncurses.so.5
[ ! -f /lib64/libtinfo.so.5 ] && ln -s /lib64/libtinfo.so.6
/lib64/libtinfo.so.5
#13.弹出提示，MySQL安装成功
echo "MySQL安装成功，软件安装路径：/export/server/mysql，数据库初始密码：123456！"
```

3.2 配置Slave从服务器

只需要安装MySQL软件，不需要初始化也不需要启动MySQL软件。

```
#!/bin/bash
#1.安装依赖软件
echo "正在安装依赖软件..."
yum -y install libaio &> /dev/null
if [ $? -ne 0 ];then
    echo "libaio安装失败"
    exit 1
fi
#2.判断是否有压缩包，如果有，则执行解压缩操作
echo "正在判断是否有压缩包，如果有进行解压缩操作..."
if [ -f mysql-8.0.40-linux-glibc2.17-x86_64.tar.xz ]; then
    tar -xf mysql-8.0.40-linux-glibc2.17-x86_64.tar.xz
    ls -l mysql-8.0.40-linux-glibc2.17-x86_64
fi
#3.判断系统中是否安装过mariadb软件，如果有对其进行卸载操作
echo "正在判断系统中是否安装过mariadb软件，如果有对其进行卸载操作..."
rpm -qa | grep mariadb | xargs -r dnf remove -y
[ -f /etc/my.cnf ] && rm -rf /etc/my.cnf
#4.创建mysql系统账号
id mysql &> /dev/null
[ $? -ne 0 ] && useradd -r -s /sbin/nologin mysql
#5.创建/export/server目录，然后移动mysql压缩包解压后的文件到/export/server目录下
rm -rf /export/server
```

```
mkdir -p /export/server
mv mysql-8.0.40-linux-glibc2.17-x86_64 /export/server/mysql
#6.进入mysql目录，对其进行初始化操作
echo "正在进入mysql目录，从服务器无需进行初始化操作..."
cd /export/server/mysql
#7.设置ssl加密传输连接
bin/mysql_ssl_rsa_setup --datadir=/export/server/mysql/data &> /dev/null
#8.设置my.cnf与mysqld.service文件
echo "正在设置my.cnf与mysqld.service文件..."
cat > /etc/my.cnf <<EOF
[mysqld]
port=3306
basedir=/export/server/mysql
datadir=/export/server/mysql/data
socket=/tmp/mysql.sock
character_set_server=utf8
collation-server=utf8_unicode_ci
EOF

cat > /etc/systemd/system/mysqld.service <<EOF
[Unit]
Description=MySQL Server
After=network.target

[Service]
User=mysql
Group=mysql
Type=forking

# MySQL 执行命令及路径
ExecStart=/export/server/mysql/bin/mysqld --daemonize --pid-
file=/export/server/mysql/data/mysqld.pid
ExecStop=/export/server/mysql/bin/mysqladmin --defaults-
file=/export/server/mysql/my.cnf shutdown

# Ensure MySQL has sufficient time to start up
TimeoutSec=600

# PID 文件路径
PIDFile=/export/server/mysql/data/mysqld.pid

# Enable these options to auto-restart the service if it crashes
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

```
EOF
#9.刷新后台服务，然后启动mysqld
echo "正在刷新后台服务，暂不启动mysqld..."
systemctl daemon-reload
systemctl enable mysqld
#10.把mysql的bin目录添加到环境变量中
echo 'export PATH=$PATH:/export/server/mysql/bin' >> /etc/profile
source /etc/profile
#11.解决mysql客户端首次无法登录问题
[ ! -f /lib64/libncurses.so.5 ] && ln -s /lib64/libncurses.so.6 /lib64/libncurses.so.5
[ ! -f /lib64/libtinfo.so.5 ] && ln -s /lib64/libtinfo.so.6 /lib64/libtinfo.so.5
#13.弹出提示，MySQL安装成功
echo "MySQL安装成功，软件安装路径：/export/server/mysql，数据库暂未初始化，数据库暂未启动！"
```

注意：暂时不需要初始化数据库文件，只是安装好了和master相同版本的mysql数据库软件；后面向master来同步所有数据。

master：从0-1安装MySQL，需要初始化，需要start启动服务！

slave：从0-1安装，但是又和master有所不同，不需要初始化，因为数据来源于master，不需要启动mysqld，没有数据目录本身也无法启动！

注意：以上两个脚本执行都必须通过source mysql8.sh

问题：./mysql8.sh、sh/bash mysql8.sh 与 source mysql8.sh执行有何不同?

./mysql8.sh、sh/bash mysql8.sh都代表直接运行脚本，这两种方式运行脚本都会产生一个子进程，程序所在终端（主进程），两者之间相互独立，子进程环境变量无法影响父进程中的环境变量。

source执行脚本时，虽然也会产生一个子进程，但是不仅会影响子进程中的环境变量，也会影响父进程中的环境变量。

3.3 修改MySQL主从配置（核心）

master服务器 => my.cnf

```
cat > /etc/my.cnf <<EOF
[mysqld]
basedir=/export/server/mysql
datadir=/export/server/mysql/data
socket=/tmp/mysql.sock
port=3306
log-error=/export/server/mysql/master.err
```

```
log-bin=/export/server/mysql/data/binlog
server-id=10
character_set_server=utf8mb4
collation-server=utf8mb4_unicode_ci
EOF
```

master主服务器配置完成后，先在主服务器端为其创建一个master.err的日志文件并授权

```
touch /export/server/mysql/master.err
chown mysql.mysql /export/server/mysql/master.err
systemctl restart mysqld
```

slave服务器 => my.cnf => 开启了relaylog（重写日志 => 把主服务器binlog重写）

```
cat > /etc/my.cnf <<EOF
[mysqld]
basedir=/export/server/mysql
datadir=/export/server/mysql/data
socket=/tmp/mysql.sock
port=3306
log-error=/export/server/mysql/slave.err
log-bin=/export/server/mysql/data/binlog
relay-log=/export/server/mysql/data/relaylog
server-id=20
character_set_server=utf8mb4
collation-server=utf8mb4_unicode_ci
EOF
```

从服务器配置完成后，也需要提前创建slave.err错误日志文件

代码块

```
touch /export/server/mysql/slave.err
chown mysql.mysql /export/server/mysql/slave.err
```

3.4 启动Master主服务器并创建同步账号

在master主数据库中，创建同步账号

代码块

```
CREATE USER 'slave'@'%' IDENTIFIED WITH mysql_native_password BY '123';
```

```
2
3   with mysql_native_password：为了兼容早期MySQL版本以前方便一些第三方软件，如DataGrip、
    Navicat等等进行远程连接
```

授予用户slave REPLICATION SLAVE权限和REPLICATION CLIENT权限，用于在主从库之间同步数据。

```
1   GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'slave'@'%';
2
3   权限的目的是为了读取二进制日志以及拉取二进制中的数据信息
4   REPLICATION SLAVE, REPLICATION CLIENT
```

查看主SQL状态

```
1   show master status;
```

记录下File和Position的值，并且不进行其他操作以免引起Position的变化。

运行效果：

```
1   mysql> show master status;
2   +---------------+----------+--------------+------------------+----------------
    ---+
3   | File     | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
4   +---------------+----------+--------------+------------------+----------------
    ---+
5   | binlog.000003 |   682 |       |         |          |
6   +---------------+----------+--------------+------------------+----------------
    ---+
7   1 row in set (0.00 sec)
```

3.5 同步master数据目录到slave

停止master服务器上的mysql

```
1   systemctl stop mysqld
```

切换到mysql目录

```
cd /export/server/mysql
```

删除auto.cnf文件

```
rm -f data/auto.cnf
```

说明：auto.cnf文件里保存的是每个数据库实例的UUID信息，代表数据库的唯一标识

同步master数据到slave

```
rsync -av /export/server/mysql/data node2:/export/server/mysql/
```

新知识点：早期数据同步都是通过scp实现上传下载，scp底层是SSH协议。scp上传下载效率较低

新命令：rsync同步指令，实现Linux与Linux之间的数据同步，rsync -av

启动master和slave数据库

master:

```
systemctl start mysqld
```

slave:

```
chown -R mysql.mysql /export/server/mysql
systemctl start mysqld
```

注意：我们启动master和slave机器上的mysqld时，都容易出现启动不了的情况，遇到这种情况不要急，一定先要.err错误日志 => 只要环境问题，通过日志几乎100%可以解决。

3.6 在slave端同步master数据

master加琐

先加锁，防止两边数据不一致

```
mysql> flush tables with read lock;
```

查看当前数据库的二进制日志写到什么位置（只有打开二进制日志，这句命令才有结果）

```
mysql> show master status;
+---------------+----------+--------------+------------------+-------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+---------------+----------+--------------+------------------+-------------------+
| binlog.000004 |      1207|              |                  |                   |
+---------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

slave服务器执行以下操作：配置主从信息，但还未开始真正同步。

slave实现同步到master => change replication source

```
mysql> CHANGE REPLICATION SOURCE TO
  SOURCE_HOST='192.168.88.101',
  SOURCE_USER='slave',
  SOURCE_PASSWORD='123',
  SOURCE_LOG_FILE='binlog.000004',
  SOURCE_LOG_POS=1207;
```

在slave服务器上启动同步

```
mysql> start slave;
mysql> show slave status\G
         ......
         Slave_IO_Running: Yes       代表成功连接到master并且下载日志
         Slave_SQL_Running: Yes      代表成功执行日志中的SQL语句
         Seconds_Behind_Master: 0    代表主从延迟的秒数，如果为0，代表状态最佳，主从没有延迟

;与\G都代表SQL的结尾，有所不同在于分号是横向展示，而\G把每一列纵向显示，适合大数据展示!
```

回到master主服务器，在mysql里面，进行解锁操作

```
代码块
```

```
mysql> unlock tables;
```

3.7 测试主从复制结果

master中创建数据库、数据表并插入数据

```
create database db_itheima;
use db_itheima;

create table students(
        id int primary key,
        name varchar(20)
) default charset=utf8;

insert into students values (1, 'Tom');
insert into students values (2, 'Rose');
```

在slave端验证：

```
mysql> show databases;
mysql> use db_itheima;
mysql> show tables;
mysql> select * from students;
```

特别注意：一旦两者配置为主从以后，主master节点既可以读数据也可以写数据，但是一定一定要注意slave服务器只能读数据，不能写数据，一旦写入，集群主从马上报错！！！

常见问题

如果启动slave报如下错误：

代码块

```
mysql> start slave;
ERROR 1872 (HY000): Slave failed to initialize relay log info structure from
the repository
```

> 解决方案：在node2服务器上，删除relay-log.info，重启mysqld服务然后重新配置CHANGE REPLICATION SOURCE TO重新同步，重新启动slave。

① 两个线程都是No，这不是错误，一般是因为没有启动主从 => `start slave;`

② 某一个错误，可能是由于之前配置有问题 => `stop slave; reset slave;` 重新change replication

注意：mysql8.0.40版本，停止操作官方添加一个新的指令 `stop replica; start replica;`

③ uuid相同，auto.cnf之前忘记删除了 => IO错误 => uuid => 更改完成后，重启mysqld

④ server-id也要不同，server-id => IO错误 => server-id => 更改完成后，重启mysqld

⑤ 两边数据不一致导致同步一直失败 => SQL错误 => 小数据量，只有1-2条不同步，可以考虑跳过这个操作；如果不同步内容较多，就只能把data目录下文件删除，重新同步，重新配置

⑥ 防火墙没关 => IO错误（Connecting）

4、预留作业

☆ xtrabackup增量备份，操作明白

☆ 把node1、node2恢复快照，重新搭建传统主从复制（AB复制）

☆ 主从同步中，Seconds_Behind_Master代表主从延迟，查询一下，如果从库主从延迟特别高，可能原因以及解决方案有哪些（面试题）

三、基于全局事务标识符GTID主从复制（重点）

作用：主从复制在工作中可能会有两种形式（传统AB复制，基于binlog日志 + pos点位实现复制）+（mysql5.7以后版本新增的基于GTID的主从复制），相对于传统AB复制，基于GTID的主从复制在两方面比较灵活：

配置灵活，不需要关心binlog及点位，直接配置，自动追踪

跳过异常，也比较灵活，简单操作就可以解决主从复制中的异常信息（SQL异常）

1、基于binlog点位主从复制痛点分析

痛点 1：首次开启主从复制的步骤复杂

- 第一次开启主从同步时，要求从库和主库是一致的。
- 找到主库的 binlog 位点。
- 设置从库的 binlog 位点。
- 开启从库的复制线程。

痛点 2：恢复主从复制的步骤复杂

- 找到从库复制线程停止时的位点。

- 解决复制异常的事务。无法解决时就需要手动跳过指定类型的错误，比如通过设置 slave_skip_errors=1032,1062。当然这个前提条件是跳过这类错误是无损的。（1062 错误是插入数据时唯一键冲突；1032 错误是删除数据时找不到行）

不论是首次开启同步时需要找位点和设置位点，还是恢复主从复制时，设置位点和忽略错误，**这些步骤都显得过于复杂，而且容易出错**。所以 MySQL 5.6 版本引入了 GTID，彻底解决了这个困难。

2、基于全局事务标识符（GTID）复制

官网：https://dev.mysql.com/doc/refman/8.0/en/replication-gtids.html

事务：增、删、改操作

事务标识符：每执行一次事务操作（增、删、改），系统都会给其定义一个唯一编号，往往是一个很长的字符串。

GTID是一个基于原始mysql服务器生成的一个已经被成功执行的全局事务ID，它由服务器ID以及事务ID组合而成。这个全局事务ID不仅仅在原始服务器上唯一，在所有存在主从关系的mysql服务器上也是唯一的。正是因为这样一个特性使得mysql的主从复制变得更加简单，以及数据库一致性更可靠。

- 一个GTID在一个服务器上只执行一次，避免重复执行导致数据混乱或者主从不一致。
- GTID用来代替传统AB复制方法，不再使用MASTER_LOG_FILE+MASTER_LOG_POS开启复制。而是使用MASTER_AUTO_POSTION=1的方式开始复制。
- 在传统的replica（从服务器）端，binlog是不用开启的，但是在GTID中replica端的binlog是必须开启的，目的是记录执行过的GTID（强制）。

master主服务器：开启binlog二进制日志

slave从服务器：既要开启relaylog中继日志，也需要开启binlog二进制日志（获取GTID编号）

3、GTID的优势

- 更简单的实现 failover，不用以前那样在需要找位点（log_file 和 log_pos）。
- 更简单的搭建主从复制。
- 比传统的AB复制更加安全。
- GTID 是连续的没有空洞的，保证数据的一致性，零丢失。

4、GTID结构（学会阅读GTID）

GTID表示为一对坐标，由冒号(:)分隔，如下所示:

```
GTID = source_id:transaction_id
```

- source_id标识source服务器，即源服务器唯一的server_uuid，由于GTID会传递到replica，所以也可以理解为源ID。
- transaction_id是一个序列号，由事务在源上提交的顺序决定。序列号的上限是有符号64位整数（2^63-1）

例如，最初要在UUID为 `3E11FA47-71CA-11E1-9E33-C80AA9429562` 的服务器上提交的第23个事务具有此GTID

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:23
```

GTID集合是由一个或多个GTID或GTID范围组成的集合。来自同一服务器的一系列gtid可以折叠成单个表达式，如下所示:

代码块

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
```

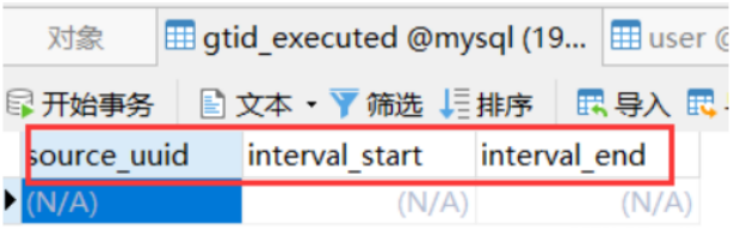
源自同一服务器的多个单一gtid或gtid范围也可以包含在单个表达式中，gtid范围以冒号分隔，如下例所示:

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:1-3:11:47-49

1-3：事务1-3
11：第11个事务
47-49：事务47-49
```

GTID集合可以包括单个GTID和GTID范围的任意组合，也可以包括来自不同服务器的GTID。

```
2174B383-5441-11E8-B90A-C80AA9429562:1-3, 24DA167-0C0C-11E8-8442-
00059A3C7B00:1-19

2174B383-5441-11E8-B90A-C80AA9429562:1-3代表第1台服务器事务1-3
24DA167-0C0C-11E8-8442-00059A3C7B00：1-19代表第2台服务器事务1-19
```

GTID存储在mysql数据库中名为gtid_executed的表中。该表中的一行包含它所代表的每个GTID或GTID集合的起始服务器的UUID，以及该集合的开始和结束事务id。



5、GTID工作原理（理解）

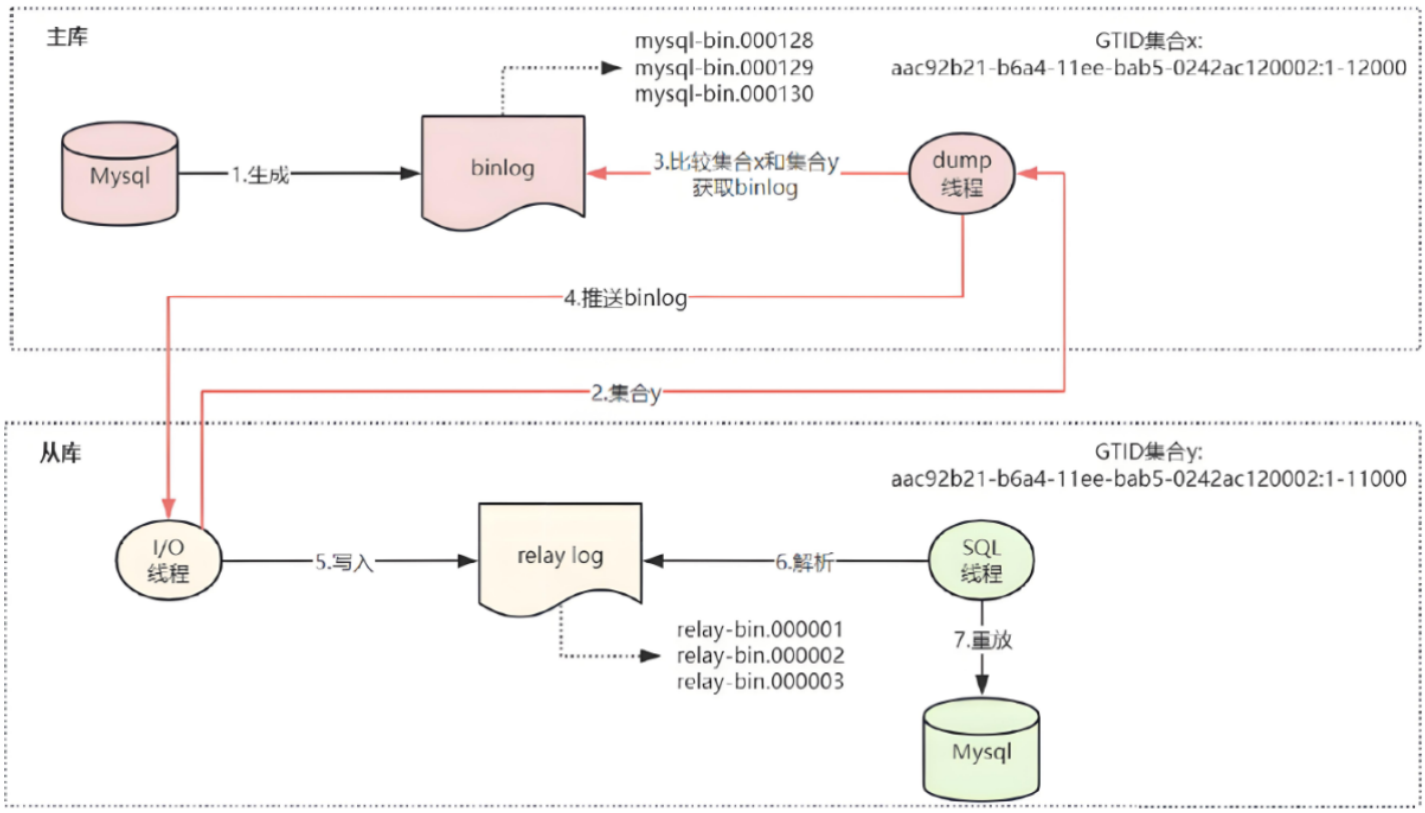
GTID：全局事务ID编号，server_uuid + 事务序号，单独MySQL服务器还是主从集群环境中，都是唯一的。

作用：帮助各位小伙伴更好理解GTID相对于传统AB复制的不同！！！

问题：基于GTID的主从复制，既不需要指定二进制文件名称，也不需要指定二进制文件位置？那GTID的主从复制是如何捕获差异内容，实现主从同步呢？

主库计算主库 GTID 集合和从库 GTID 的集合的差集，主库推送差集 binlog 给从库。

当从库设置完同步参数后，主库 A 的 GTID 集合记为集合 x，从库 B 的 GTID 集合记为 y。从库同步的逻辑如下：



- 从库 B 指定主库 A，基于主备协议建立连接（AB服务器首先建立主从复制）。
- 从库 B 把集合 y 发给主库 A。

- 主库 A 计算出集合 x 和集合 y 的差集，也就是集合 x 中存在，集合 y 中不存在的 GTID 集合。比如集合 x 是 1~100，集合 y 是 1~90，那么这个差集就是 91~100。这里会判断集合 x 是不是包含有集合 y 的所有 GTID，如果不是则说明主库 A 删除了从库 B 需要的 binlog，主库 A 直接返回错误。
- 主库 A 从自己的 binlog 文件里面，找到第一个不在集合 y 中的事务 GTID，也就是找到了 91。
- 主库 A 从 GTID = 91 的事务开始，往后读 binlog 文件，按顺序取 binlog，然后发给 B。
- 从库 B 的 I/O 线程读取 binlog 文件生成 relay log，SQL 线程解析 relay log，然后执行 SQL 语句。

GTID 同步方案和位点同步的方案区别是：

- 位点同步方案是通过人工在从库上指定哪个位点，主库就发哪个位点，不做日志的完整性判断。
- 而 GTID 方案是通过主库来自动计算位点的，不需要人工去设置位点，对运维人员友好。

6、GTID的配置与实现

作用：基于GTID实现主从复制（重点）

文档：https://dev.mysql.com/doc/refman/8.0/en/replication-gtids-howto.html

环境说明：

关闭防火墙+SELINUX、配置IP与主机映射、时间同步、安装依赖包

IP	主机名	角色
192.168.88.101	node1.itcast.cn	master(主)
192.168.88.102	node2.itcast.cn	slave(从)

安装前准备：① 配置IP、主机名 ② 配置IP与主机映射 => /etc/hosts ③ 关闭防火墙与SELinux ④ 时间同步 ⑤ 安装必备软件，如vim、wget、rsync

设置主机名称

代码块

```
hostnamectl set-hostname node1.itcast.cn
hostnamectl set-hostname node2.itcast.cn

su或者bash指令
```

配置IP与主机映射

代码块

```
vim /etc/hosts
尾部追加如下内容
192.168.88.101 node1 node1.itcast.cn
192.168.88.102 node2 node2.itcast.cn
```

关闭防火墙与SELinux

代码块

```
systemctl stop firewalld
systemctl disable firewalld

setenforce 0
vim /etc/selinux/config
```

安装一些依赖软件（系统必备软件）

代码块

```
dnf install vim wget rsync -y
```

时间同步参考时间同步文档 => CentOS9时间同步（飞书）

到此环境准备完毕了！

数据库环境准备：请基于master.sh以及slave.sh自行安装mysql8软件，并进行数据同步

master：安装mysql并进行初始化

代码块

```
source master.sh
```

slave：只安装mysql不进行初始化

```
source slave.sh
```

停止mysqld，然后同步数据到slave中

master:

```
systemctl stop mysqld
rm -rf /export/server/mysql/data/auto.cnf
rsync -av /export/server/mysql/data node2:/export/server/mysql/
```

第一步：修改主库配置

修改主库的配置文件

```
cat > /etc/my.cnf <<EOF
[mysqld]
basedir=/export/server/mysql
datadir=/export/server/mysql/data
socket=/tmp/mysql.sock
port=3306
log-error=/export/server/mysql/master.err
log-bin=/export/server/mysql/data/binlog
server-id=10
character_set_server=utf8mb4
collation-server=utf8mb4_unicode_ci
gtid_mode=on
enforce_gtid_consistency=on
log_slave_updates=1
EOF
```

GTID配置说明:

启用全局事务标识符（GTID）模式

gtid_mode=on

强制GTID的一致性。这意味着在执行事务时，MySQL将确保所有涉及的服务器都使用相同的GTID集。

enforce_gtid_consistency=on

log_slave_updates=1

> `log_slave_updates=1` 是 MySQL 主从复制中的一个关键参数，它决定了从库（Slave）是否将**从主库（master）同步过来的数据变更事件**记录到自己的二进制日志（binlog）

第二步：修改从库配置

修改从库配置文件

```
cat > /etc/my.cnf <<EOF
```

```
[mysqld]
basedir=/export/server/mysql
datadir=/export/server/mysql/data
socket=/tmp/mysql.sock
port=3306
log-error=/export/server/mysql/slave.err
log-bin=/export/server/mysql/data/binlog
relay-log=/export/server/mysql/data/relaylog
server-id=20
character_set_server=utf8mb4
collation-server=utf8mb4_unicode_ci
gtid_mode=on
enforce_gtid_consistency=on
log_slave_updates=1
read-only=on
EOF
```

master节点/slave节点，重启MySQL

```
# 主服务器
touch /export/server/mysql/master.err
chown -R mysql.mysql /export/server/mysql
systemctl start mysqld

# 从服务器
touch /export/server/mysql/slave.err
chown -R mysql.mysql /export/server/mysql
systemctl start mysqld
```

从节点设置主库信息

以下是官网提供的设置参考（仅参考，需要调整为自己master服务器信息）

https://dev.mysql.com/doc/refman/8.0/en/replication-gtids-howto.html

```
mysql> CHANGE REPLICATION SOURCE TO
    >     SOURCE_HOST = host,
    >     SOURCE_PORT = port,
    >     SOURCE_USER = user,
    >     SOURCE_PASSWORD = password,
    >     SOURCE_AUTO_POSITION = 1;
```

> SOURCE_AUTO_POSITION = 1： 这告诉从服务器使用自动位置跟踪功能，以便它可以自动从主服务器获取最新的二进制日志事件，而无需手动指定位置。

具体配置代码：

master主服务器创建同步账号：

```
mysql> CREATE USER 'slave'@'%' IDENTIFIED WITH mysql_native_password BY '123';
mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'slave'@'%';
```

slave从服务器基于change replication source进行同步操作

```
mysql> change replication source to
  source_host='192.168.88.101',
  source_port=3306,
  source_user='slave',
  source_password='123',
  source_auto_position=1;
```

> 和AB复制最大的不同就是不需要寻找binlog以及对应的pos点位，只需要source_auto_position=1就可以自动同步！

第三步：开启从库复制

```
mysql> start replica;
```

查看复制状态

代码块

```
mysql> show slave status\G
...
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
...
```

7、主从复制报错修复

模拟从库删除测试表、主库对表进行插入操作

master数据库中：

```
create database db_itheima;
use db_itheima;

create table students(
        id int primary key,
        name varchar(20)
) default charset=utf8;

insert into students values (1, 'Tom');
insert into students values (2, 'Rose');
```

slave数据库中，错误的插入一条记录

编辑slave服务器中的/etc/my.cnf文件

代码块

```
删除最后一行 => read-only=on
然后重启mysqld
systemctl restart mysqld
```

```
insert into students values (3, 'Jack');
```

回到master数据库，也重新插入一条记录

```
insert into students values (3, 'Jennifer');
```

slave从服务器查看同步状态：

```
show slave status\G
```

观察从库复制是否报错

查看从库同步状态：

```
mysql> show slave status \G
...
Slave_IO_Running: Yes
Slave_SQL_Running: No
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 1146
Last_Error: Coordinator stopped because there were error(s) in the worker(s). The most recent failure being: Worker 1 failed executing transaction 'f1b88047-a5ea-11ed-8ee1-246e9657f7a0:7' at master log mysql-bin.000011, end_log_pos 868. See error log and/or performance_schema.replication_applier_status_by_worker table for more details about this failure or others, if any.
Skip_Counter: 0
Exec_Master_Log_Pos: 550
Relay_Log_Space: 1285
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: NULL
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 1146
Last_SQL_Error: Coordinator stopped because there were error(s) in the worker(s). The most recent failure being: Worker 1 failed executing transaction 'f1b88047-a5ea-11ed-8ee1-246e9657f7a0:7' at master log mysql-bin.000011, end_log_pos 868. See error log and/or performance_schema.replication_applier_status_by_worker table for more details about this failure or others, if any.
Replicate_Ignore_Server_Ids:
Master_Server_Id: 1
Master_UUID: f1b88047-a5ea-11ed-8ee1-246e9657f7a0
Master_Info_File: mysql.slave_master_info
SQL_Delay: 0
SQL_Remaining_Delay: NULL
Slave_SQL_Running_State:
```

```
Master_Retry_Count: 86400
Master_Bind:
Last_IO_Error_Timestamp:
Last_SQL_Error_Timestamp: 230227 14:53:19
Master_SSL_Crl:
Master_SSL_Crlpath:
Retrieved_Gtid_Set: f1b88047-a5ea-11ed-8ee1-246e9657f7a0:1-7
Executed_Gtid_Set: f1b88047-a5ea-11ed-8ee1-246e9657f7a0:1-6
Auto_Position: 1
Replicate_Rewrite_DB:
Channel_Name:
Master_TLS_Version:
Master_public_key_path:
Get_master_public_key: 0
Network_Namespace:
1 row in set, 1 warning (0.01 sec)
```

复制报错信息

```
Retrieved_Gtid_Set: f1b88047-a5ea-11ed-8ee1-246e9657f7a0:1-7
Executed_Gtid_Set: f1b88047-a5ea-11ed-8ee1-246e9657f7a0:1-6
事务接收了1-7，但7没有执行成功。f1b88047-a5ea-11ed-8ee1-246e9657f7a0:7
```

在主库继续进行其他事务，观察gitd是否复制成功

```
mysql> create table test02_01(id int ,name varchar(10));
Query OK, 0 rows affected (0.08 sec)
mysql> insert into test02_01 values(1,'jkl');
Query OK, 1 row affected (0.00 sec)
```

从库状态

```
mysql> show replica status\G
...
Slave_IO_Running: Yes
Slave_SQL_Running: No
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
```

```
Replicate_Wild_Ignore_Table:
Last_Errno: 1146
Last_Error: Coordinator stopped because there were error(s) in the worker(s). The most recent failure being: Worker 1 failed executing transaction 'f1b88047-a5ea-11ed-8ee1-246e9657f7a0:7' at master log mysql-bin.000011, end_log_pos 868. See error log and/or performance_schema.replication_applier_status_by_worker table for more details about this failure or others, if any.
Skip_Counter: 0
Exec_Master_Log_Pos: 550
Relay_Log_Space: 1851
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: NULL
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 1146
Last_SQL_Error: Coordinator stopped because there were error(s) in the worker(s). The most recent failure being: Worker 1 failed executing transaction 'f1b88047-a5ea-11ed-8ee1-246e9657f7a0:7' at master log mysql-bin.000011, end_log_pos 868. See error log and/or performance_schema.replication_applier_status_by_worker table for more details about this failure or others, if any.
Replicate_Ignore_Server_Ids:
Master_Server_Id: 1
Master_UUID: f1b88047-a5ea-11ed-8ee1-246e9657f7a0
Master_Info_File: mysql.slave_master_info
SQL_Delay: 0
SQL_Remaining_Delay: NULL
Slave_SQL_Running_State:
Master_Retry_Count: 86400
Master_Bind:
Last_IO_Error_Timestamp:
Last_SQL_Error_Timestamp: 230227 14:53:19
Master_SSL_Crl:
Master_SSL_Crlpath:
Retrieved_Gtid_Set: f1b88047-a5ea-11ed-8ee1-246e9657f7a0:1-9
Executed_Gtid_Set: f1b88047-a5ea-11ed-8ee1-246e9657f7a0:1-6
Auto_Position: 1
```

```
Replicate_Rewrite_DB:
Channel_Name:
Master_TLS_Version:
Master_public_key_path:
Get_master_public_key: 0
Network_Namespace:
1 row in set, 1 warning (0.00 sec)
```

事务，7-9未备执行，也就是说后续复制中断

```
Retrieved_Gtid_Set: f1b88047-a5ea-11ed-8ee1-246e9657f7a0:1-9
Executed_Gtid_Set: f1b88047-a5ea-11ed-8ee1-246e9657f7a0:1-6
```

解决方案：

在实际工作中，如果主从配置不同步，出现了异常情况，解决方案有二

情况一：如果错误事务较少，可以尝试跳过错误事务，进行修复。

情况二：如果错误事务较多，必须要重新配置了主从同步，把主服务器数据进行导出，然后在从服务器进行重新导入，然后重新配置主从。

采用从库跳过错误事务修复

停止slave进程

```
mysql> STOP REPLICA;
```

设置事务号，事务号从 Retrieved_Gtid_Set 获取，在session里设置gtid_next，即跳过这个GTID

```
mysql> SET @@SESSION.GTID_NEXT= 'f1b88047-a5ea-11ed-8ee1-246e9657f7a0:7'

案例演示（实际改成你们自己的事务）：
set @@SESSION.GTID_NEXT='这个位置到底如何编写';

Retrieved_Gtid_Set: 0883a39c-eb54-11ef-879d-000c29d9e0c0:1-12
Executed_Gtid_Set: 08817f5d-eb54-11ef-9fcd-000c296d8526:1-2
第一步：找两者差异
接收到1-12，实际执行1-2，从第3个事务开始同步异常，所以要尝试跳过事务编号3的事务
第二步：找主机uuid
主机uuid主要看接收端uuid编号 => Retrieved_Gtid_Set
```

```
12  set @@SESSION.GTID_NEXT='0883a39c-eb54-11ef-879d-000c29d9e0c0:3';
```

设置空事务，填充跳过的事务（让事务编号连续）

```
1  mysql> BEGIN; COMMIT;
```

恢复自增事务号

```
1  mysql> SET SESSION GTID_NEXT = AUTOMATIC;
```

启动slave进程

代码块

```
1  mysql> START REPLICA;
```

事务已经跳过，创建表已经同步

```
1  mysql> show tables;
```

重新同步以后，可以在主节点，删除冲突数据或者异常数据，重新插入，让两端高度一致！

四、面试题：MySQL主从延迟的原因

面试题：MySQL主从延迟比较高通常有哪些原因，如何解决?

可能原因

1. **网络延迟**:
   - 网络不稳定或带宽不足会导致从库接收主库的复制数据缓慢。
2. **磁盘I/O性能**:
   - 从库磁盘I/O性能差，导致SQL线程写入数据耗时较长。
3. **从库负载高**:
   - 从库处理大量查询导致SQL线程的同步速度减缓。
4. **大事务处理**:

- 主库上的大事务会导致生成二进制日志速度过快，从库无法及时应用这些更改。

5. **主库变更频率高**：
   - 主库频繁的写入操作诱发过多的数据需复制。

解决方案

1. **优化网络配置**：
   - 增加网络带宽，减少主从之间的网络延迟。
2. **提高硬件性能**：
   - 升级从库的硬件配置，例如使用SSD提高磁盘I/O。
3. **减少从库负载**：
   - 使用多个从库实现读写分离如mycat读写分离操作，保证复制线程不受查询处理影响。
4. **优化事务处理**：
   - 尽量减少主库大事务频率，将操作分解成多个小事务。
   - 例如：如果需要插入大量记录，可将其拆分为多个较小的批量插入。
5. **调优MySQL配置**：
   - 调整 `innodb_flush_log_at_trx_commit` 等参数以提高写入效率，增加复制线程数量，如调整 `slave_parallel_workers` 。
   - `innodb_flush_log_at_trx_commit` **参数调整**
     - 默认值为 `1` ，表示每次事务提交都会同步日志到磁盘。为了降低磁盘I/O，提高性能，可以设置为 `2` ，表示在事务提交时只写入日志缓存，定期刷入磁盘。
   - **增加复制线程数量 (** `slave_parallel_workers` **)**
     - MySQL 5.7及以上版本支持并行复制。通过增加从库复制线程数可以加速从库应用事务。
     - 在/etc/my.cnf中设置 slave_parallel_workers=4 # 根据服务器性能调整（最小为4，一般可以设置为CPU核心数的1/2或1/4），16核CPU => 4/8，32核CPU => 8/16

今日重点

- [ ] Xtrabackup物理备份 全量备份与恢复 增量备份与恢复
- [ ] 背诵和理解MySQL主从工作原理
- [ ] 从0-1搭建传统主从（AB复制）=> 难点
- [ ] 基于GTID的主从复制以及主从延迟问题解决