一主一从环境搭建

1. 环境准备

```
机器A（主机）：
系统：Ubuntu20.04
IP地址：172.16.190.111
mysql版本：mysql Ver 8.0.33

机器B（从机）：
系统：Ubuntu20.04
IP地址：172.16.190.31
mysql版本：mysql Ver 8.0.33
```

2. 搭建过程

2. 1 配置主机（机器A） /etc/my.cnf文件

```
添加：
[mysql]
server-id=1
read-only=0
log-bin=mysql-bin

重启mysql
systemctl restart mysql
```

2.2 配置从机（机器B） /etc/my.cnf文件

```
添加：
[mysql]
server-id=2
read-only=1
relay-log=mysql-relay

重启mysql
systemctl restart mysql
```

2.3 建立账户并授权（机器A）

```
CREATE USER 'zfx'@'%' IDENTIFIED WITH mysql_native_password BY 'Zfx@password123456';

GRANT REPLICATION SLAVE ON *.* TO 'zfx'@'%';
```

2.4 查看master状态（机器A）

```
show master status;
```

```
mysql> show master status
    -> ;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000002 |      658 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

2.5 从机配置 （机器B）

```
1. 设置关联主机的配置

CHANGE MASTER TO
MASTER_HOST='172.16.190.111',
MASTER_USER='zfx',
MASTER_PASSWORD='Zfx@password123456',
MASTER_LOG_FILE='mysql-bin.000002',
MASTER_LOG_POS=658;

2. 开启同步

START SLAVE;

3. 查看从机状态

show slave status\G;
```

```
mysql> show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Connecting to source
                  Master_Host: 172.16.190.111
                  Master_User: zfx
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql_bin.000002
          Read_Master_Log_Pos: 658
               Relay_Log_File: mysql-relay.000001
                Relay_Log_Pos: 4
        Relay_Master_Log_File: mysql_bin.000002
             Slave_IO_Running: Connecting
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 658
              Relay_Log_Space: 157
              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: 2003
                Last_IO_Error: Error connecting to source 'zfx@172.16.190.111:3306'. This was attempt 1/86400, with a delay of 60 seconds between attempts. Message: Can't connect to MySQL server on '172.16.190.111:3306' (111)
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 0
                  Master_UUID:
             Master_Info_File: mysql.slave_master_info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Replica has read all relay log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp: 230619 20:03:42
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
         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)
```

失败了。。。

---

3. 问题与解决

毫无疑问，第一次搭建出现了失败的情况，即，Slave_IO_Running = Connecting。说明连接不上机器A的mysql服务器。

以下是问题解决过程：

3. 1 防火墙

首先尝试了下打开防火墙3306端口的访问

```
firewall-cmd --zone=public --add-port=3306/tcp --permanent
firewall-cmd --reload
```

3. 2 账户权限

观察上述创建的名为zfx的账户权限是否存在问题?

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| cloud              |
| db_test            |
| filestore          |
| information_schema |
| mycloud            |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
8 rows in set (0.00 sec)

mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select user, host from user
    -> ;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| zfx              | %         |
| debian-sys-maint | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
6 rows in set (0.00 sec)

mysql> show grants for zfx
    -> ;
+-------------------------------------------+
| Grants for zfx@%                          |
+-------------------------------------------+
| GRANT REPLICATION SLAVE ON *.* TO `zfx`@`%` |
+-------------------------------------------+
1 row in set (0.00 sec)
```

账户权限没有问题。

3. 3 端口检查

查看一下3306端口的情况

```
netstat -apn | grep 3306
```

```
sophia@sophia:/$ netstat -apn | grep 3306
（并非所有进程都能被检测到，所有非本用户的进程信息将不会显示，如果想看到所有信息，则必须切换到 root 用户）
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:33060         0.0.0.0:*               LISTEN      -
```

到这里发现了问题所在，**监听的端口是正常的，但是却绑定了回旋地址，只能本地连接。**

然后找到 `/etc/mysql/mysql.conf.d/mysqld.cnf` 文件中的配置信息，发现 `bind-address` 和 `mysqlx-bind-address` 的设置果然是127.0.0.1.

> 这里也可以只添加要操作的机器，或者修改为0.0.0.0监听所有网卡，允许所有ip访问。
>
> `mysqlx-bind-address`是监听基于X protocol的连接的地址，X protocol是MySQL的一种新协议，可以通过MySQL Shell或其它的客户端软件使用。
>
> MySQL在发布时自带了MySQL X插件，该插件通过X protocol提供一个类似于MongoDB的服务，实现与客户端的交互。这个协议使用了protobuf进行通信，从可扩展性、性能和安全性上做了提升。

为了后续实验方便，在此处直接注释掉这两个参数。

问题解决。

3.4 重新配置从机

首先查看一下master的状态

```
show master status;
```

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000004 |      157 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

然后配置从机的SLAVE信息

```
1. 设置关联主机的配置

CHANGE MASTER TO
MASTER_HOST='172.16.190.111',
MASTER_USER='zfx',
MASTER_PASSWORD='Zfx@password123456',
MASTER_LOG_FILE='mysql-bin.000004',
MASTER_LOG_POS=157;

2. 开启同步

START SLAVE;

3. 查看从机状态

show slave status\G;
```

```
mysql> show slave status\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for source to send event
                  Master_Host: 172.16.190.111
                  Master_User: zfx
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000004
          Read_Master_Log_Pos: 157
               Relay_Log_File: mysql-relay.000002
                Relay_Log_Pos: 326
        Relay_Master_Log_File: mysql-bin.000004
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

搭建成功。

4. 测试

在机器A中创建一个表，并插入数据，然后观察机器B数据库的变化。

```
create database db_test;
use db_test;
create table tb_user (
  id int(11) primary key not null auto_increment,
  name varchar(50) not null,
  sex varchar(1)
)engine=innodb default charset=utf8mb4;
insert into tb_user(id, name, sex)
values(204670, 'tom', '1'),
(204671, 'jerry', '0'),
(204672, 'alex', '1'),
(204673, 'janica', '0'),
(204674, 'barry', '1');
```

机器B观察数据库变化：

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| db_test            |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> use db_test
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from tb_user;
+--------+--------+------+
| id     | name   | sex  |
+--------+--------+------+
| 204670 | tom    | 1    |
| 204671 | jerry  | 0    |
| 204672 | alex   | 1    |
| 204673 | janica | 0    |
| 204674 | barry  | 1    |
+--------+--------+------+
5 rows in set (0.00 sec)
```

成功!

5. 问题与总结

Q1：在失败和成功的过程中发现：虽然没有对数据库进行增删改查等一系列操作，但是重新配置master状态下的binlog日志会有不同程度的变化?

在查询和相关资料和阅读binlog日志的过程中发现，binlog日志的改变并非只有在对数据库操作时（如事务的提交、语句的执行、数据的更改）发生改变，一些系统变量的更改也会影响binlog的改变，在重新配置config文件时，参数的修改触发了binlog的记录。

Q2：在第一次失败之后，第二次的尝试中重新配置了MASTER的相关参数，即`CHANGE MASTER TO`语句，每次出错是否都需要重新配置参数?

经后面调查发现，这个参数是不需要重新执行的，因为 `CHANGE MASTER TO` 只是为IO线程提供参数。如果slave从来没有连接过master，那么必须使用`CHANGE MASTER TO` 语句来生成IO线程所需要的信息，这些信息记录在master.info中。这个文件是`CHANGE MASTER TO`成功之后立即生成的，以后每次启动IO线程，都会自动读取这个文件来连接master，不需要先执行`CHANGE MASTER TO`语句。

例如，可以随时stop slave来停止复制线程，然后再随时start slave，只要master.info存在，且没有人为修改过它，IO线程就一定不会出错。这是因为master.info会随着IO线程的执行而更新，无论读取到master binlog的哪个位置，都会记录下这个位置，如此一来，IO线程下次启动的时候就知道从哪里开始监控master binlog。

既然如此，那么在我第一次配置没有注释掉bind-address导致连接失败后，我在主服务器上修改了配置信息，并创建了新数据库D和表T，并插入了信息，导致了binlog的改变。而在此前的binlog信息早已被我通过CHANGE MASTER TO语句记录在机器B中。 当机器A产生了新的binlog记录之后，机器B有两种可以选择的操作：①stop slave、reset slave、重新配置slave，那么机器A数据库D和表T就不在旧的binlog中，无法同步到slave。②使用之前配置过的binlog信息，那么机器A所记录的信息也会同步到slave中。

5.3 binlog何时会产生一个新的文件

三种情况：

- MySQL服务器重启
- 使用flush logs命令
- binlog文件超过`max_binlog_size`的设置大小值（默认为1G）时，会生成一个新的binlog文件

5.4 数据备份

在上面的实验中，机器A里面在此之前还有其他数据库（之前自己搭建使用的云盘系统的数据库，存放了自己的一些文件），这些数据库并没有复制到机器B中，所以严格意义上来讲，我的两个机器的数据库的数据并不是严格一致的。

假设在真实业务场景中，在配置从库之前，主库已经“服务”了很长时间，存储了很多数据。此时，我需要将主库原来的数据全部拷贝到从库，以使得两个数据库的数据能够高度一致，应该怎么做?

经过调研，一共有三种数据备份方式：

首先，为了安全，必须先锁表：

```
flush tables with read lock;
```

锁表会导致写阻塞以及innodb的commit操作。然后查看master的binlog坐标并做记录。

接下来进行数据库备份：

- 冷备份，直接进行 copy。**这种情况只适用于没有新写入操作。严谨一点，只适合拷贝完成前master不能有写入操作。**

  > 假设要复制所有的数据库（包含innodb表），先关闭mysql，然后将整个datadir拷贝到slave上
  >
  > ```
  > shell> systemctl stop mysql
  > shelk> rsync -avz /data 172.16.190.111:/ # 从库执行
  > mysql> unlock tables;
  > ```

  注意，在冷备份的时候，需要将备份到目标主机上的DATADIR/auto.conf删除，这个文件中记录的是mysql server的UUID，而master和slave的UUID必须不能一致。

- 使用mysqldump进行备份恢复

  > 待学习研究

- 使用xtrabackup进行备份恢复

> 待学习研究

从调研来看，**基于xtrabackup的备份方法使用频率最高**，因为其安全性高、速度快。后续着重学习该技术（**mark**）。

5.5 主从复制的原理

主从复制的核心是数据库中的重要日志文件---binlog二进制文件。它记录了对数据库更新的事件，记录了所有的DDL和DML语句（不包含查询语句select、show等）。

主从复制的过程基于三个线程来操作，分别是`binlog dump`线程、`I/O`线程和`SQL`线程。其中，`binlog dump`线程位于master节点上，其它两个线程位于slave节点。

主从复制的核心流程：

- master接收到一个写请求，会将写请求的更新操作记录到binlog日志中
- `binlog dump`线程会读取master节点上的binlog日志，然后将binlog日志发送给slave节点上的`I/O`线程。在主库读取事件时，会给binlog加锁，读取完成后，再释放掉锁，防止数据不一致的情况。
- slave节点的`I/O`线程接收到binlog日志后，将binlog日志先写入到本地的中继日志（relaylog）中。
- slave节点的`SQL`线程读取relaylog，解析为增删改操作，重做一遍在master中的操作，还原数据。

5.6 主从复制的数据一致性问题

由于主库和从库之间的复制时间受数据量的大小、网络性能等影响，所以进行读写分离的时候，也常伴随着数据不一致的问题，比如从节点里查询数据，但从节点还没有从relay log中恢复数据，从而导致数据不一致等等。

那么按照数据一致性的强弱划分，有以下三种复制方式：

- 全同步复制

  > 当主库执行完一个事务之后，要求所有的从库也都必须执行完该事务，才可以返回处理结果给客户端；因此，虽然全同步复制数据一致性得到保证了，但是主库完成一个事物需要等待所有从库也完成，性能就比较低了。

- 异步复制

  > 当主库提交事物后，会通知binlog dump线程发送binlog日志给从库，一旦binlog dump线程将binlog日志发送给从库之后，不需要等到从库也同步完成事务，主库就会将处理结果返回给客户端。

  MySQL主从复制的默认复制策略就是异步复制，但是有如下两个问题：

  - 主库只管自己执行完事务，就可以将处理结果返回给客户端，而不用关心从库是否执行完事务，这就可能导致短暂的主从数据不一致的问题了，比如刚在主库插入的新数据，如果马上在从库查询，就可能查询不到。
  - 当主库提交事物后，如果宕机挂掉了，此时可能binlog还没来得及同步给从库，这时候如果为了恢复故障切换主从节点的话，就会出现数据丢失的问题，所以异步复制虽然性能高，但数据一致性上是最弱的。

- 半同步复制

> 半同步复制原理是在客户端提交COMMIT之后不直接将结果返回给客户端，而是等待至少有一个从库收到了Binlog，并且写入到中继日志中，再返回给客户端。这样做的好处就是提高了数据的一致性，当然相比于异步复制来说，至少多增加了一个网络连接的延迟，降低了主库写的效率。

当然，随着数据一致性变强一点，性能自然会下降一点，因为至少需要等待至少一个从库确认接收到binlog日志的响应时间t，如果超过时间t，则半同步复制就会退化为异步复制。

此外，MySQL5.7.2后的版本对半同步复制做的一个改进，解决之前存在的幻读问题：

> 为什么半同步复制存在幻读问题?
>
> 当主库成功提交事物并处于等待从库确认的过程中，这个时候，从库都还没来得及返回处理结果给客户端，但因为主库存储引擎内部已经提交事务了，所以，其他客户端是可以到从主库中读到数据的。但是，如果下一秒主库突然挂了，此时正好下一次请求过来，因为主库挂了，就只能把请求切换到从库中，因为从库还没从主库同步完数据，所以，从库中当然就读不到这条数据了，和上一秒读取数据的结果对比，就造成了幻读的现象了。

在之后，提出了增强半同步复制，通过一个配置参数`rpl_semi_sync_master_wait_point = AFTER_SYNC`以解决幻读问题，核心思想是主库在存储引擎提交事务前，必须先收到从库数据同步完成的确认信息后，才能提交事务。（**为事务提交增加一个条件**）

一主多从环境搭建

1.环境准备

```
机器A（主机）：
系统：Ubuntu20.04
IP地址：172.16.190.111
mysql版本：mysql Ver 8.0.33

机器B（从机）：
系统：Ubuntu20.04
IP地址：172.16.190.31
mysql版本：mysql Ver 8.0.33

机器C（从机）：
系统：Ubuntu20.04
IP地址：172.16.190.35
mysql版本：mysql Ver 8.0.33
```

2. 搭建过程

在原有的基础上，将机器C设置为机器A的slave。同时，为了验证上面Q1的想法，机器C的MASTER信息采用和配置机器B时相同的参数，这样的话可以保证机器B和机器C的同步一致性。

2.1 配置从机（机器C） /etc/my.cnf文件

```
添加:
[mysql]
server-id=3
read-only=1
relay-log=mysql-relay

重启mysql
systemctl restart mysql
```

2.2 配置从机（机器C）

```
1. 设置关联主机的配置

CHANGE MASTER TO
MASTER_HOST='172.16.190.111',
MASTER_USER='zfx',
MASTER_PASSWORD='Zfx@password123456',
MASTER_LOG_FILE='mysql-bin.000004',
MASTER_LOG_POS=157;

2. 开启同步

START SLAVE;

3. 查看从机状态

show slave status\G;
```

```
mysql> start slave;
Query OK, 0 rows affected, 1 warning (0.06 sec)

mysql> show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for source to send event
                  Master_Host: 172.16.190.111
                  Master_User: zfx
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000008
          Read_Master_Log_Pos: 157
               Relay_Log_File: mysql-relay.000010
                Relay_Log_Pos: 373
        Relay_Master_Log_File: mysql-bin.000008
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
```

由于该MASTER的参数配置和3.4节中相同，所以机器C和机器B的数据库状态是相同的。

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| db_test            |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use db_test;
Database changed
mysql> select * from tb_user;
+--------+--------+------+
| id     | name   | sex  |
+--------+--------+------+
| 204670 | tom    | 1    |
| 204671 | jerry  | 0    |
| 204672 | alex   | 1    |
| 204673 | janica | 0    |
| 204674 | barry  | 1    |
+--------+--------+------+
5 rows in set (0.00 sec)

mysql>
```

成功

2.3 测试

机器A在表`tb_user`中插入一条记录：

```
user db_test;
insert into tb_user values(204675, 'frost', 0);
```

查看机器B和机器C的同步状态：

```
mysql> use db_test
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from tb_user;
+--------+--------+------+
| id     | name   | sex  |
+--------+--------+------+
| 204670 | tom    | 1    |
| 204671 | jerry  | 0    |
| 204672 | alex   | 1    |
| 204673 | janica | 0    |
| 204674 | barry  | 1    |
| 204675 | frost  | 0    |
+--------+--------+------+
6 rows in set (0.00 sec)
```

成功！

3. 从节点升为主节点

以上已经搭建了一主多从的环境。此时，假设主机A宕机，需要将从机B升级为主机，然后机器C作为机器B的从机。

3.1 模拟主机宕机

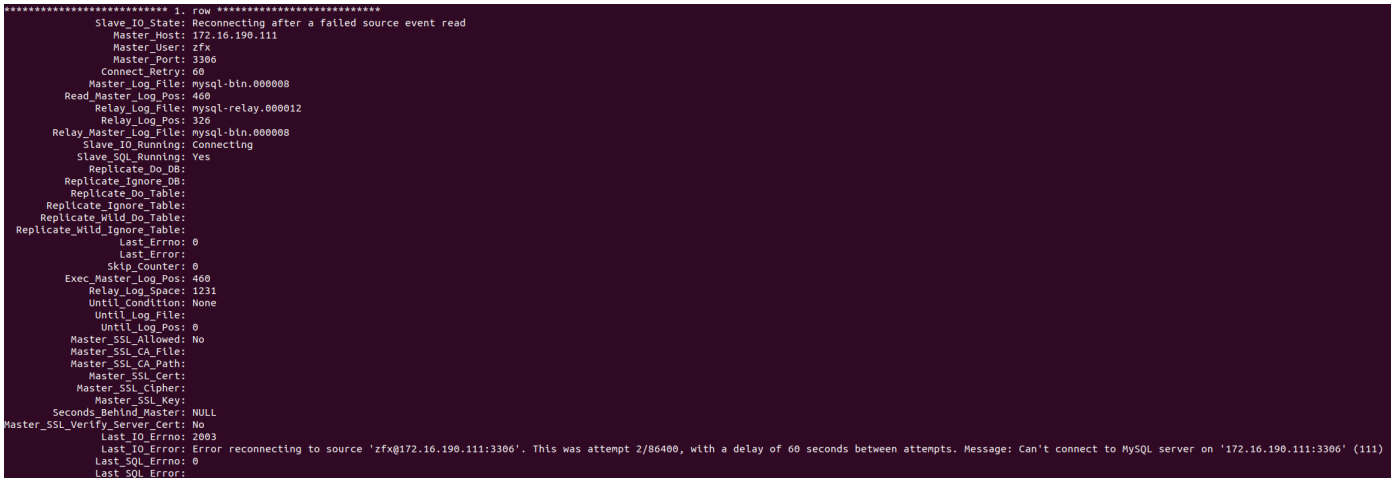
停止机器A中的mysql服务的运行：

```
systemctl stop mysql
```

3.2 配置机器B

首先，需要保证所有从机的中继日志relay-log全部读取完毕（由于本实验环境只有很小的数据，所以基本不存在relay-log没有读取完毕的情况。但是当操作数据很多时，relay-log可能会存在还未读取完毕。）

```
stop slave io_thread;
show slave status\G;
```



然后，重新配置`my.cnf`文件信息，将`read-only`设为0，注释掉`relay-log`，添加`bin-log`相关信息，这里仍将日志命名为mysql-bin。

同时注释掉`/etc/mysql/mysql.conf.d/mysqld.cnf`文件中的`bind-address`和`mysqlx-bind-address`参数。

```
[mysqld]
server-id=2
read-only=0
log-bin=mysql-bin
# relay-log=mysql-relay

重启mysql
systemctl restart mysql
```

然后进入mysql，由于在之前的设置中，只同步了数据库，用户信息并没有同步，所以这里需要重新创建和授权用户，为了后续实验方便和同步该用户到机器C中，在这里记录一下master的状态。但在此之前，先停掉slave和删掉slave相关信息：

```
# 停掉slave
stop slave;
# 删除slave信息
reset slave;

# 重新生成二进制日志。
reset master;

# 查看当前binlog
show master status;

# 创建用户并授权（和2.3中相同）

CREATE USER 'zfx'@'%' IDENTIFIED WITH mysql_native_password BY 'Zfx@password123456';

GRANT REPLICATION SLAVE ON *.* TO 'zfx'@'%';
```

```
mysql> show master status;
+------------------+----------+--------------+------------------+--------------
----+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_
Set |
+------------------+----------+--------------+------------------+--------------
----+
| mysql-bin.000001 |     1025 |              |                  |
    |
+------------------+----------+--------------+------------------+--------------
----+
1 row in set (0.00 sec)
```

到此为止，机器B的配置结束，机器B已经变为新的主机。

3.3 配置机器C

首先，将原有的slave信息重置（当然，要先保证relay-log全部读取完毕）：

```
# 停掉slave
stop slave;
# 删除slave信息
reset slave;
```

然后，重新传递MASTER参数

```
CHANGE MASTER TO
MASTER_HOST='172.16.190.31',
MASTER_USER='zfx',
MASTER_PASSWORD='Zfx@password123456',
MASTER_LOG_FILE='mysql-bin.000001',
MASTER_LOG_POS=1025;

2. 开启同步

START SLAVE;

3. 查看从机状态

show slave status\G;
```

成功

这时候，就可以在mysql数据库的user表中，查看到机器C也创建了名为`zfx`的用户。

```
mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select user, host from user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| zfx              | %         |
| debian-sys-maint | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
6 rows in set (0.00 sec)

mysql> use db_test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

3.4 测试

在机器B中插入一条数据，查看同步状态：

```
user db_test;
insert into tb_user values(204676, 'savitar', 1);
```

在机器C中查看：

```
mysql> select * from tb_user;
+--------+---------+------+
| id     | name    | sex  |
+--------+---------+------+
| 204670 | tom     | 1    |
| 204671 | jerry   | 0    |
| 204672 | alex    | 1    |
| 204673 | janica  | 0    |
| 204674 | barry   | 1    |
| 204675 | frost   | 0    |
| 204676 | savitar | 1    |
+--------+---------+------+
7 rows in set (0.00 sec)
```

成功！

4. 机器A重启后变为机器B的从节点

4.1 配置机器A

假设此时机器A宕机修复，重启机器之后，将其设置为机器B的从节点，使服务器继续以一主双从的模式运行。

对机器A进行以下配置：

```
# 1.首先修改my.cnf文件，将read-only设置为1，命名relay-log
[mysql]
server-id=1
read-only=1
log-bin=mysql-bin
relay-log=mysql-relay

# 2. 重置master信息
reset master

# 3. 添加MASTER参数
CHANGE MASTER TO
MASTER_HOST='172.16.190.31',
MASTER_USER='zfx',
MASTER_PASSWORD='Zfx@password123456',
MASTER_LOG_FILE='mysql-bin.000001',
MASTER_LOG_POS=1025;
# 为了保证数据的一致性，这个地方采用的binlog和position为3.2中的记录，即主节点A宕机之后，主节点B的binlog名称和位置，因为此时主库和从库的数据是一致的。当主节点A恢复之后，必须得宕机时刻的机器B的binlog位置重新恢复数据，否则会丢失数据。

2. 开启同步

START SLAVE;

3. 查看从机状态

show slave status\G;
```

这样就成功将原主机A设置为从机，变身为B的slave~

```
mysql> use db_test;
Database changed
mysql> select * from tb_user;
+--------+---------+------+
| id     | name    | sex  |
+--------+---------+------+
| 204670 | tom     | 1    |
| 204671 | jerry   | 0    |
| 204672 | alex    | 1    |
| 204673 | janica  | 0    |
| 204674 | barry   | 1    |
| 204675 | frost   | 0    |
| 204676 | savitar | 1    |
+--------+---------+------+
7 rows in set (0.00 sec)
```

4.2 问题排查

然而，在4.1成功之前，遇到过一个错误：

错误描述：由于在4.1中配置`MASTER`信息时传的是宕机之后机器B的`binlog filename`和`binlog position`，这是为了在新的主节点机器B上执行的新的语句能够在机器A恢复之后进行同步，从而保证数据的一致性。但是在启动机器A之后，报了一个错误（原图记录忘记截图），主要意思是在机器A中执行语句`SET @@SESSION.GTID_NEXT='ANONYMOUS'`报出了坐标错误。

原因分析：排查发现，该错误的原因是在MySQL 5.6版本新加了一个全局事务编号（GTID），并在MySQL 5.7版本之后默认开启，会自动生成`SET @@SESSION.GTID_NEXT='ANONYMOUS'`语句，为提交的事务自动生成一个编号。

该错误的出现可能是因为我在记录该位置之前对数据库进行了删除操作，然后在从库里执行恢复语句的时候定位不到行数据，所以产生了报错。因为在将机器B升为主机的时候，自己做了几个数据测试，导致binlog变化有点大（彼时还没意识到问题的严重性），并且两个实验并不是同一天做的，前面写的几个测试语句产生了后面的千奇百怪的bug。不过这个bug也是让自己深刻意识到了，在保证数据一致性的时候需要注意到的各种情况。

问题解决：既然问题出现在恢复删除语句无法定位行数据，那么就选择跳过此`position`。首先查看`mysql.bin.000001`的信息：

```
mysql> show binlog events in 'mysql-bin.000001'
    -> ;
+------------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------------------------------------------------------+
| Log_name         | Pos  | Event_type     | Server_id | End_log_pos | Info                                                                                                                    |
+------------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------------------------------------------------------+
| mysql-bin.000001 |    4 | Format_desc    |         2 |         126 | Server ver: 8.0.33-0ubuntu0.20.04.2, Binlog ver: 4                                                                      |
| mysql-bin.000001 |  126 | Previous_gtids |         2 |         157 |                                                                                                                         |
| mysql-bin.000001 |  157 | Anonymous_Gtid |         2 |         236 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'                                                                                    |
| mysql-bin.000001 |  236 | Query          |         2 |         314 | BEGIN                                                                                                                   |
| mysql-bin.000001 |  314 | Table_map      |         2 |         381 | table_id: 89 (db_test.tb_user)                                                                                          |
| mysql-bin.000001 |  381 | Delete_rows    |         2 |         431 | table_id: 89 flags: STMT_END_F                                                                                          |
| mysql-bin.000001 |  431 | Xid            |         2 |         462 | COMMIT /* xid=82 */                                                                                                     |
| mysql-bin.000001 |  462 | Anonymous_Gtid |         2 |         541 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'                                                                                    |
| mysql-bin.000001 |  541 | Query          |         2 |         617 | BEGIN                                                                                                                   |
| mysql-bin.000001 |  617 | Table_map      |         2 |         815 | table_id: 62 (mysql.user)                                                                                               |
| mysql-bin.000001 |  815 | Delete_rows    |         2 |         994 | table_id: 62 flags: STMT_END_F                                                                                          |
| mysql-bin.000001 |  994 | Xid            |         2 |        1025 | COMMIT /* xid=125 */                                                                                                    |
| mysql-bin.000001 | 1025 | Anonymous_Gtid |         2 |        1104 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'                                                                                    |
| mysql-bin.000001 | 1104 | Query          |         2 |        1307 | CREATE USER 'zfx'@'%' IDENTIFIED WITH 'mysql_native_password' AS '*83EAD7EA7DBC6AACCB7C3514A4FCB8D46ED9C798' /* xid=130 */ |
| mysql-bin.000001 | 1307 | Anonymous_Gtid |         2 |        1384 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'                                                                                    |
| mysql-bin.000001 | 1384 | Query          |         2 |        1526 | GRANT REPLICATION SLAVE ON *.* TO 'zfx'@'%' /* xid=131 */                                                               |
| mysql-bin.000001 | 1526 | Anonymous_Gtid |         2 |        1605 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'                                                                                    |
| mysql-bin.000001 | 1605 | Query          |         2 |        1683 | BEGIN                                                                                                                   |
| mysql-bin.000001 | 1683 | Table_map      |         2 |        1750 | table_id: 89 (db_test.tb_user)                                                                                          |
| mysql-bin.000001 | 1750 | Write_rows     |         2 |        1800 | table_id: 89 flags: STMT_END_F                                                                                          |
| mysql-bin.000001 | 1800 | Xid            |         2 |        1831 | COMMIT /* xid=147 */                                                                                                    |
+------------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------------------------------------------------------+
21 rows in set (0.00 sec)
```

将`MASTER_LOG_POS`设置为1307，跳过1025的语句，然后启动SLAVE，最终成功。

5. 总结

5.1 主从复制方式

从上面4.1的问题中，经过深度调研，了解到了主从复制的方式其实有两种，一种就是基于`binlog`文件名字和POS做定位，传递给MASTER相关信息，然后进行数据的复制；另一种是在5.6版本引入的GTID（全局事务ID）机制，在5.7版本以后默认开启，即基于GTID的复制方式，由UUID和事务ID两个部分组成，具有如下特点：

- GTID事务是全局唯一性的，并且一个事务对应一个GTID值。
- 一个GTID值在同一个MySQL实例上只会执行一次。

> GTID工作流程：
>
> - master节点在更新数据的时候，会在事务前产生GTID信息，一同记录到binlog日志中。
> - slave节点的io线程将binlog写入到本地relay log中。
> - 然后SQL线程从relay log中读取GTID，设置gtid_next的值为该gtid，然后对比slave端的binlog是否有记录。
> - 如果有记录的话，说明该GTID的事务已经运行，slave会忽略。
> - 如果没有记录的话，slave就会执行该GTID对应的事务，并记录到binlog中。

在上面的实验过程中，其实是默认开启了GTID复制方式的，但是在配置过程中，为了详细地熟悉机制，我还是先使用了传统复制方式赋予MASTER信息。

相对于传统复制方式的执行语句，GTID复制的语句更加便捷，不需要指定文件名称和位置:

```
CHANGE MASTER TO
MASTER_HOST="0.0.0.0",
MASTER_PORT=3306,
MASTER_USER='USER',
MASTER_PASSWORD='PASSWORD',
MASTER_AUTO_POSITION=1
```

同时，除了开启binlog相关参数外，主从节点还需要额外添加如下参数配置：

```
gtid_mode=on    # 开启GTID
enforce-gtid-consistency=on  # 需要同步设置该参数
log-slave-updates=1      # 5.6 版本需要开启该参数
```

5.2 宕机主节点自动转移方法

在做主节点宕机，从节点升级为主节点实验时，就在想有没有什么方法可以在集群环境中，当主节点宕机时，随机选择从节点自动升级为主节点呢？答案肯定是有的，比如：

MHA（Master High Availability）是MySQL高可用的一个相对成熟的解决方案。其本质原理可以总结为：

> - 从宕机的master中保存binlog events
> - 在所有从节点中找到具有最新更新的slave
> - 将上面具有最新更新的slave的relay log到其它的slave
> - 应用从master保存的binlog events
> - 提升slave为master

> - 其它的slave连接新的master

可以看到该步骤和上面手动设置的步骤几乎相同，只不过在实际网络环境中，需要考虑到更多的可能性，比如slave节点的选择须是所有的slave中具有最新更新的从节点，这样能够最大程度的保证数据一致性。

> 思考：如果主机宕机，如硬件故障，MHA并没有办法连接到主机并保存其二进制的日志，从而导致丢失部分新的数据。怎么办?
>
> 似乎可以配合半同步复制，来确保至少有一个从节点已经接收到主节点的事务，从而减少丢失数据的可能性。

其它方式还有**Keepalived**，**Heartbeat**等实现MySQL的高可用。