主讲老师：Fox

> 【有道云笔记】https://note.youdao.com/s/5CZH2GrM

课程目标：

MySQL 复制（Replication） 是官方提供的主从复制（源到副本的复制）方案，用于将一个 MySQL 的实例同步到另一个实例中。 这是使用最广泛的容灾方案**（重点掌握）**。

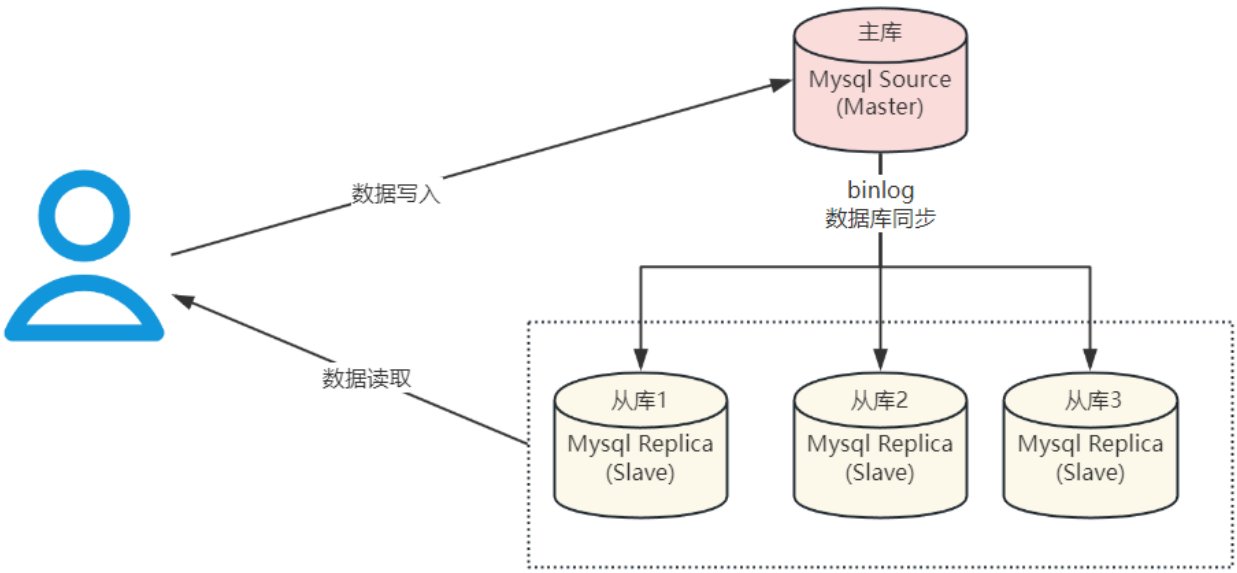
复制（Replication）

什么是复制

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

MySQL Replication是官方提供的主从同步方案，也是用的最广的同步方案。Replication(复制)使来自一个MySQL数据库服务器（称为源（Source））的数据能够复制到一个或多个 MySQL 服务器（称为副本（Replica））。默认情况下，复制是异步的；副本不需要永久连接即可从源接收更新。根据配置，您可以复制所有数据库、指定数据库，甚至某个数据库中的指定表。

> 说明： 旧版本的 MySQL 复制将源（Source）称为主（Master），将副本（Replica）称为从（Slave）



复制的优势：

- 高可用：通过配置一定的复制机制，MySQL 实现了跨主机的数据复制，从而获得一定的高可用能力，如果需要获得更高的可用性，只需要配置多个副本，或者进行级联复制就可以达到目的。
- 性能扩展：由于复制机制提供了多个数据备份，可以通过配置一个或多个副本，将读请求分发至副本节点，从而获得整体上读写性能的提升。
- 异地灾备：只需要将副本节点部署到异地机房，就可以轻松获得一定的异地灾备能力。实际当中，需要考虑网络延迟等可能影响整体表现的因素。

- 交易分离：通过配置复制机制，并将低频、大运算量的交易发送至副本节点执行，就可以避免这些交易与高频交易竞争运算资源，从而避免整体的性能问题。

缺点：

- 没有故障自动转移，容易造成单点故障
- 主库从库之间有主从复制延迟问题，容易造成最终数据的不一致
- 从库过多对主库的负载以及网络带宽都会带来很大的负担

应用场景

- 电子商务平台： 在电商平台中，主从复制可以用于实现读写分离，提高并发处理能力，同时确保数据的一致性。
- 社交网络： 在社交网络应用中，可以利用主从复制来提供快速的读取服务，同时将数据变更复制到从数据库以备份数据。
- 实时监控和报警系统： 在监控系统中，主从复制可以用于实现数据的分布式存储和快速数据查询。
- 新闻和媒体网站： 在高访问量的新闻网站中，可以使用主从复制来提供高可用性和快速的内容访问。
- 金融服务： 在金融行业，数据的安全性和可用性至关重要，主从复制可以用于数据备份和高可用性的实现。

复制的方式

MySQL 8.0支持多种复制方式：

- 传统的方法是**基于从源的二进制日志（binlog）复制事件**，并要求日志文件及其中的位置在源和副本之间进行同步。作为源(数据库更改发生的地方)的 MySQL 实例将更新和更改作为“事件”写入二进制日志。根据所记录的数据库更改，二进制日志中的信息以不同的日志格式存储。副本配置为从源中读取二进制日志，并在副本的本地数据库上执行二进制日志中的事件。

```
#获取binlog文件列表
mysql> show binary logs;
#查看指定binlog文件的内容
mysql> show binlog events in 'binlog.000003';
```

> 官网：https://dev.mysql.com/doc/refman/8.0/en/binlog-replication-configuration-overview.html

- **基于全局事务标识符(GTID)的方式。**基于 GTID 的复制是完全基于事务的，所以很容易确定源和副本是否一致; 只要在源上提交的所有事务也在副本上提交，就可以保证两者之间的一致性。

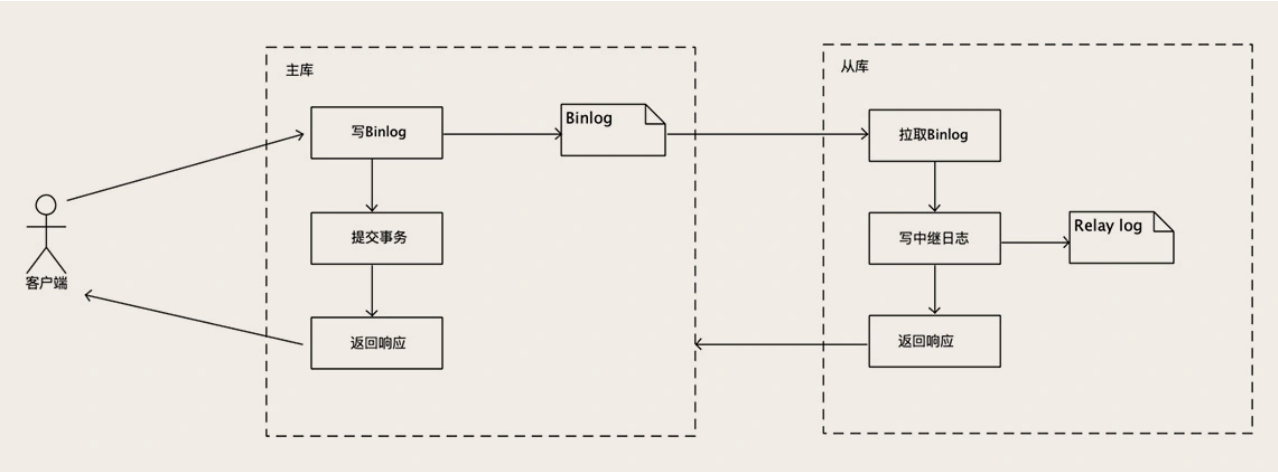
> 官网：https://dev.mysql.com/doc/refman/8.0/en/replication-gtids.html

复制的数据同步类型

MySQL 中的复制支持不同类型的同步。同步的原始类型是单向异步复制，其中一个服务器充当源，而一个或多个其他服务器充当副本。**在 MySQL 8.0中，除了内置的异步复制之外，还支持半同步复制。**使用半同步复制，在返回执行事务的会话之前，对源执行提交，直到至少有一个副本确认它已经接收并记录了事务的事件。**MySQL 8.0还支持延迟复制**，以使副本故意落后于源至少指定的时间。

异步复制

默认情况下，MySQL 采用异步复制的方式，执行事务操作的线程不会等复制 Binlog 的线程。具体的时序你可以看下面这个图：



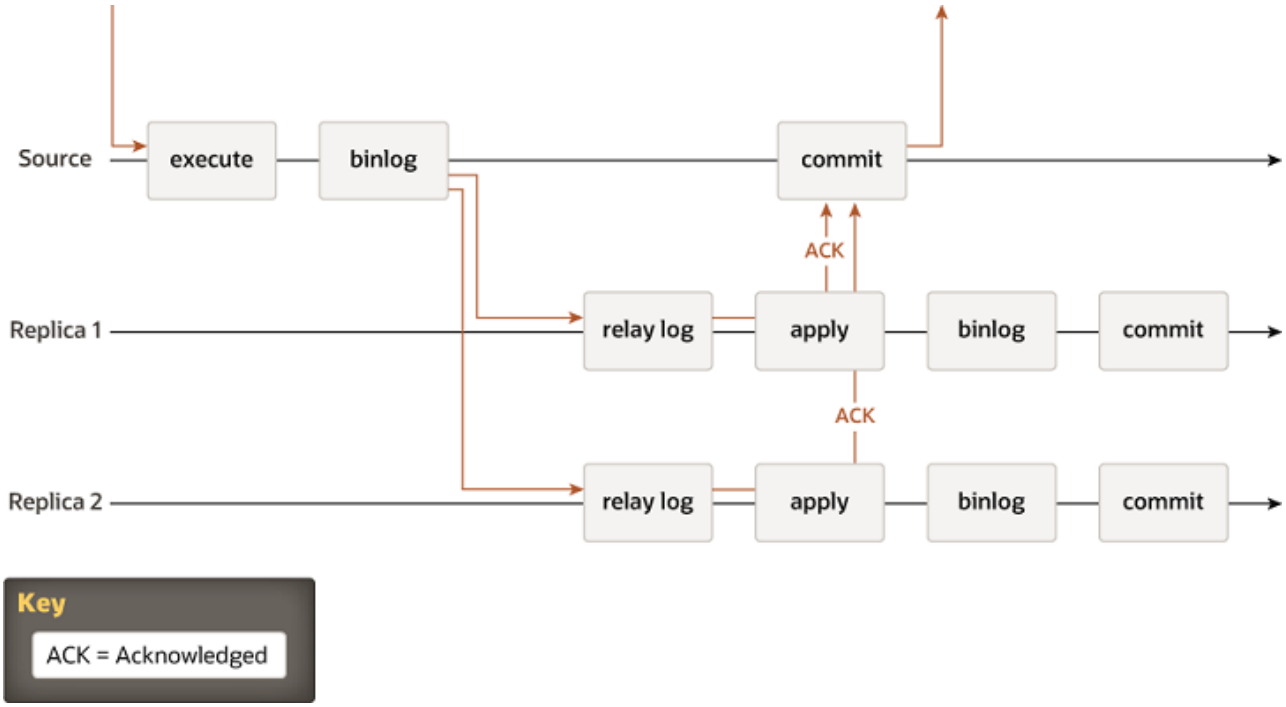
MySQL 主库在收到客户端提交事务的请求之后，会先写入 Binlog，然后再提交事务，更新存储引擎中的数据，事务提交完成后，给客户端返回操作成功的响应。同时，从库会有一个专门的复制线程，从主库接收 Binlog，然后把 Binlog 写到一个中继日志里面，再给主库返回复制成功的响应。从库还有另外一个回放 Binlog 的线程，去读中继日志，然后回放 Binlog 更新存储引擎中的数据。

提交事务和复制这两个流程在不同的线程中执行，互相不会等待，这是异步复制。异步复制的劣势是，可能存在主从延迟，如果主节点宕机，可能会丢数据。

半同步复制

MySQL 从 5.7 版本开始，增加一种半同步复制（Semisynchronous Replication）的方式。这种机制与异步复制相比主要有如下区别：

- 主节点在收到客户端的请求后，必须在完成本节点日志写入的同时，还需要等待至少一个从节点完成数据同步的响应之后（或超时），才会响应请求。
- 从节点只有在写入 relay-log 并完成刷盘之后，才会向主节点响应。
- 当从节点响应超时时，主节点会将同步机制退化为异步复制。在至少一个从节点恢复，并完成数据追赶后，主节点会将同步机制恢复为半同步复制。



可以看出，相比于异步复制，半同步复制在一定程度上提高了数据的可用性，在未退化至异步复制时，如果主节点宕机，此时数据已复制至至少一台从节点。同时，由于向客户端响应时需要从节点完成响应，相比于异步复制，此时多出了主从节点上网络交互的耗时以及从节点写文件并刷盘的耗时，因此整体上集群对于客户端的响应性能表现必然有所降低。

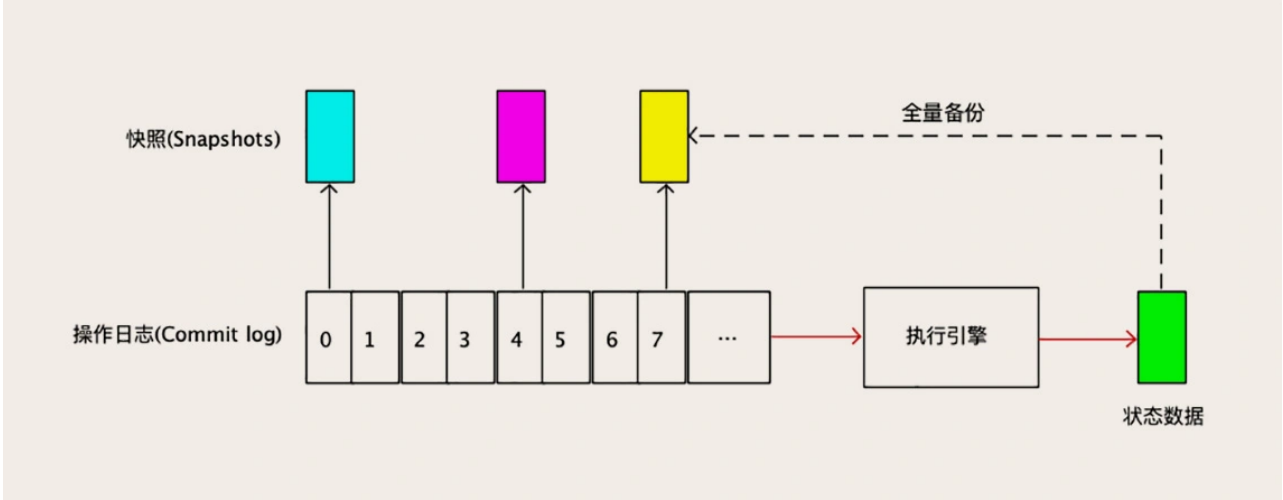
半同步复制有两个重要的参数：

- **rpl_semi_sync_master_wait_slave_count**（8.0.26之后改为rpl_semi_sync_source_wait_for_replica_count）：至少等待数据复制到几个从节点再返回。这个数量配置的越大，丢数据的风险越小，但是集群的性能和可用性就越差。
- **rpl_semi_sync_master_wait_point**（8.0.26之后改为rpl_semi_sync_source_wait_point）：这个参数控制主库执行事务的线程，是在提交事务之前（AFTER_SYNC）等待复制，还是在提交事务之后（AFTER_COMMIT）等待复制。默认是 AFTER_SYNC，也就是先等待复制，再提交事务，这样就不会丢数据。

设计理念：复制状态机——几乎所有的分布式存储都是这么复制数据的

在 MySQL 中，无论是复制还是备份恢复，依赖的都是全量备份和 Binlog，全量备份相当于备份那一时刻的一个数据快照，Binlog 则记录了每次数据更新的变化，也就是操作日志。这种基于“快照 + 操作日志”的方法，不是 MySQL 特有的。比如说，Redis Cluster 中，它的全量备份称为 Snapshot，操作日志叫 backlog，它的主从复制方式几乎和 MySQL 是一模一样的。Elasticsearch用的是 translog，它备份和恢复数据的原理和实现方式也是完全一样的。

任何一个存储系统，无论它存储的是什么数据，用什么样的数据结构，都可以抽象成一个状态机。存储系统中的数据称为状态（也就是 MySQL 中的数据），状态的全量备份称为快照（Snapshot），就像给数据拍个照片一样。我们按照顺序记录更新存储系统的每条操作命令，就是操作日志（Commit Log，也就是 MySQL 中的 Binlog）。

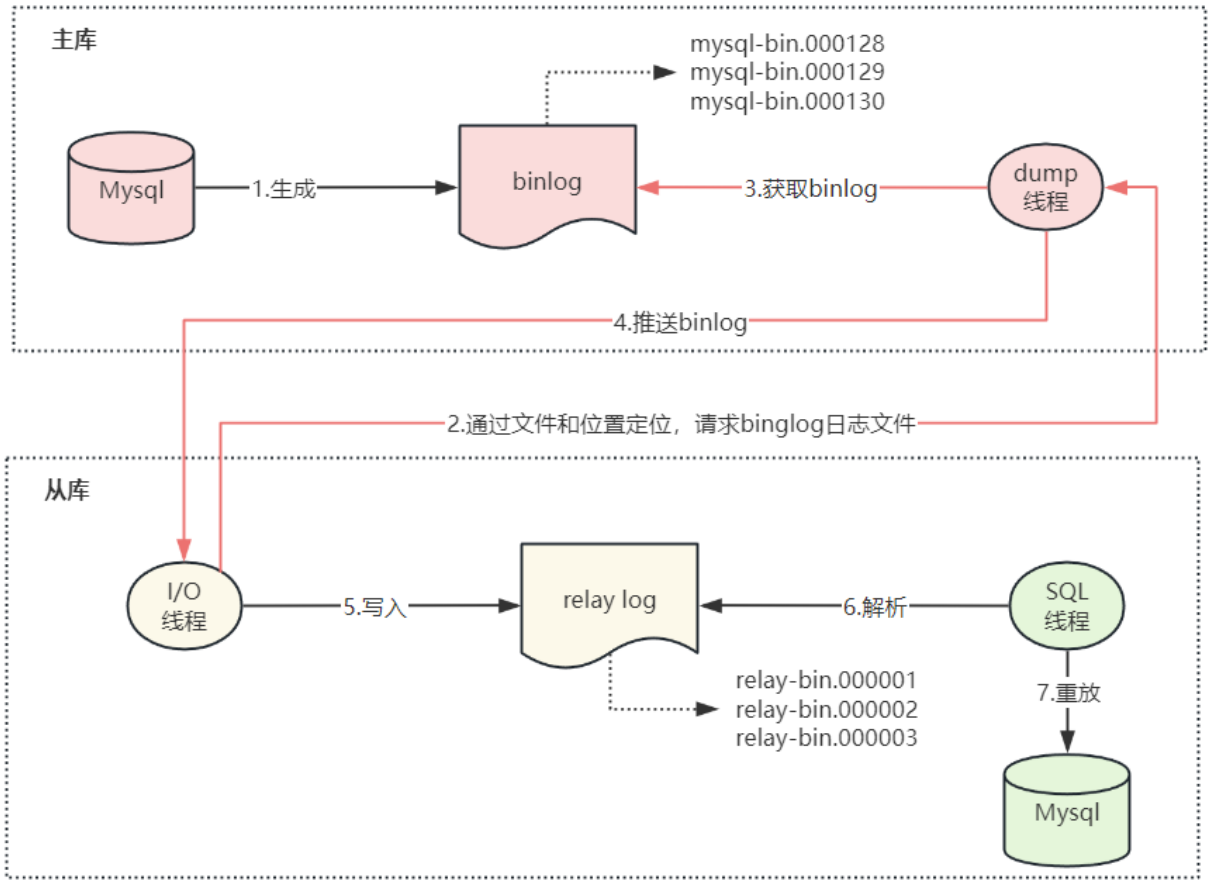


复制数据的时候，只要基于一个快照，按照顺序执行快照之后的所有操作日志，就可以得到一个完全一样的状态。在从节点持续地从主节点上复制操作日志并执行，就可以让从节点上的状态数据和主节点保持同步。

主从同步做数据复制时，一般可以采用几种复制策略。

- 性能最好的方法是异步复制，主节点上先记录操作日志，再更新状态数据，然后异步把操作日志复制到所有从节点上，并在从节点执行操作日志，得到和主节点相同的状态数据。异步复制的劣势是，可能存在主从延迟，如果主节点宕机，可能会丢数据。
- 另外一种常用的策略是半同步复制，主节点等待操作日志最少成功复制到 N 个从节点上之后，再更新状态，这种方式在性能、高可用和数据可靠性几个方面都比较平衡，很多分布式存储系统默认采用的都是这种方式。

基于binlog位点同步的主从复制原理



1、主库会生成多个 binlog 日志文件。

2、从库的 I/O 线程请求指定文件和指定位置的 binlog 日志文件（位点）。

3、主库 dump 线程获取指定位点的 binlog 日志。

4、主库按照从库发送给来的位点信息读取 binlog，然后推送 binlog 给从库。

5、从库将得到的 binlog 写到本地的 relay log (中继日志) 文件中。

6、从库的 SQL 线程读取和解析 relay log 文件。

7、从库的 SQL 线程重放 relay log 中的命令。

异步复制示例

1）快速创建mysql实例

> 参考：Docker 安装 MySQL8.0

利用Docker快速搭建Mysql8一主两从复制架构

角色	server_id	ip:port
mysql-source (主)	10	192.168.65.185:3307
mysql-replica1(从1)	11	192.168.65.185:3308
mysql-replica2(从2)	12	192.168.65.185:3309

2) 配置mysql主从复制

- **主节点**

2.1）创建挂载目录

```
mkdir -p /mysql/replication/source/data /mysql/replication/source/conf
/mysql/replication/source/log
```

2.2) 准备配置文件

```
vim /mysql/replication/source/conf/custom.cnf
[mysql]
# 设置mysql客户端默认编码
default-character-set=utf8

[mysqld]
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
datadir         = /var/lib/mysql

secure-file-priv= NULL

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

# 服务器唯一ID，默认是1
server-id=10

# 启用二进制日志
log-bin=mysql-bin


# 最大连接数
max_connections=1000

# 设置默认时区
default-time_zone='+8:00'

# 0:区分大小写
# 1:不区分大小写
```

```
lower_case_table_names=1

!includedir /etc/mysql/conf.d/

```

> pid-file: 这是MySQL服务器的进程ID文件的位置。通过这个文件，您可以在系统上找到正在运行的MySQL服务器的进程。
> socket: 这是MySQL服务器用于本地通信的Unix套接字文件的位置。
> datadir: 这是MySQL服务器存储其数据文件的位置。
> secure-file-priv: 这是一个用于限制LOAD_FILE()和SELECT ... INTO OUTFILE命令的文件路径。如果此选项被设置，那么这两个命令只能用于读取在这个路径下的文件。设置为NULL表示禁用这个功能。
> symbolic-links: 如果设置为0，MySQL服务器将不允许在数据目录中使用符号链接。这有助于防止安全风险。
> server-id: 每个MySQL服务器实例在复制时需要有一个唯一的ID。这有助于区分不同的服务器，特别是在复制环境中。
> log-bin: 启用二进制日志记录所有对数据库的更改，这对于复制和恢复操作是必要的。
> max_connections: 这是MySQL服务器可以接受的最大并发连接数。
> default-time_zone: 这设置了MySQL服务器的默认时区。
> lower_case_table_names: 这决定了MySQL如何存储和比较表名。设置为1意味着表名不区分大小写（但在文件系统中它们仍然会区分大小写）。
> !includedir /etc/mysql/conf.d/: 这告诉MySQL服务器从/etc/mysql/conf.d/目录中包含其他配置文件。这意味着该目录下的任何.cnf或.ini文件都会被合并到这个主配置文件中。
> replicate_do_db： 待同步的数据库日志
> replicate_ignore_db：不同步的数据库日志

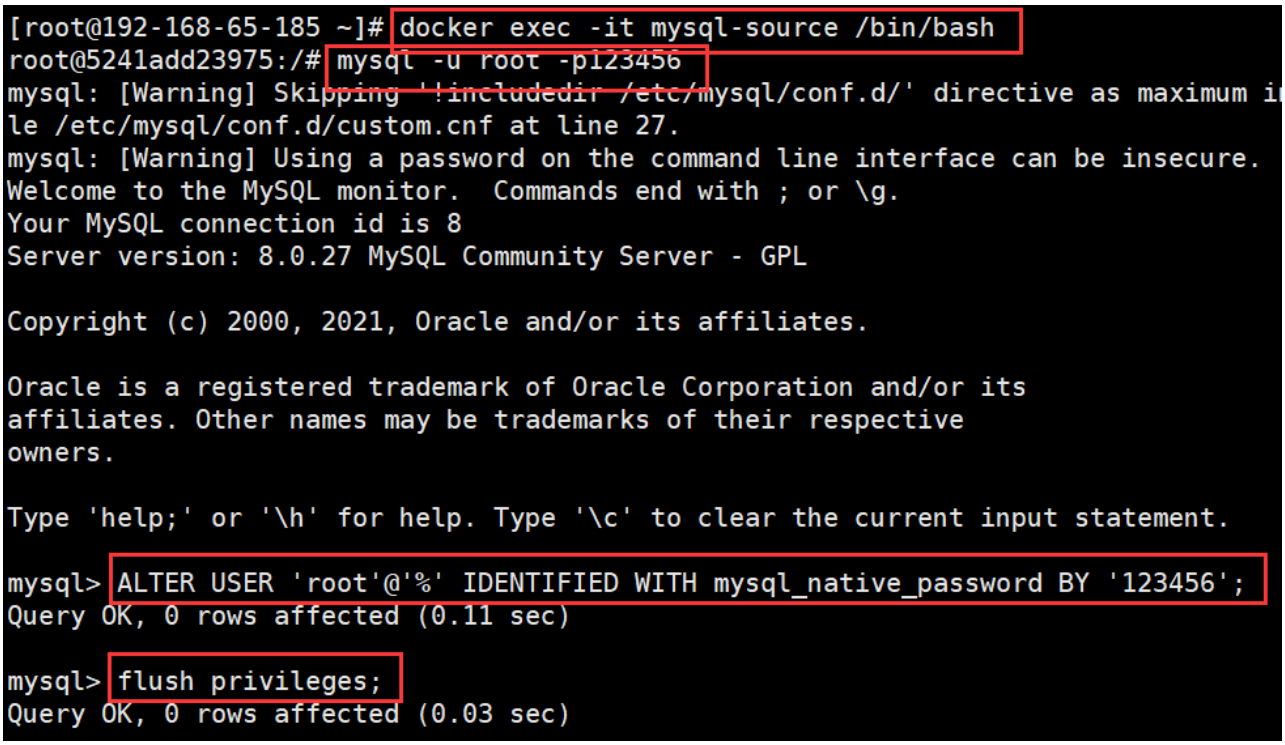
2.3）运行mysql容器

```
# 创建主从复制的网络
docker network create --driver bridge mysql-source-replica
#运行mysql容器
docker run  -d  \
--name mysql-source \
--privileged=true \
--restart=always \
--network  mysql-source-replica \
-p 3307:3306 \
```

```
-v /mysql/replication/source/data:/var/lib/mysql \
-v /mysql/replication/source/conf:/etc/mysql/conf.d  \
-v /mysql/replication/source/log:/logs \
-e MYSQL_ROOT_PASSWORD=123456 \
-e TZ=Asia/Shanghai mysql:8.0.27 \
--lower_case_table_names=1
```

2.4）配置远程访问

```
docker exec -it mysql-source /bin/bash
mysql -u root -p

ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
flush privileges;
```



- **从节点1**

```
# 创建挂载目录
mkdir -p /mysql/replication/replica1/data /mysql/replication/replica1/conf
/mysql/replication/replica1/log

#准备配置文件
vim /mysql/replication/replica1/conf/custom.cnf
```

```
[mysql]
# 设置mysql客户端默认编码
default-character-set=utf8

[mysqld]
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
datadir         = /var/lib/mysql

secure-file-priv= NULL

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

# 服务器唯一ID，默认是1
server-id=11

# 启用二进制日志
log-bin=mysql-bin

# 最大连接数
max_connections=1000

# 设置默认时区
default-time_zone='+8:00'

# 0:区分大小写
# 1:不区分大小写
lower_case_table_names=1

!includedir /etc/mysql/conf.d/


#运行mysql容器
docker run  -d  \
--name mysql-replica1 \
--privileged=true \
--restart=always \
--network  mysql-source-replica \
-p 3308:3306 \
```

```
-v /mysql/replication/replica1/data:/var/lib/mysql \
-v /mysql/replication/replica1/conf:/etc/mysql/conf.d  \
-v /mysql/replication/replica1/log:/logs \
-e MYSQL_ROOT_PASSWORD=123456 \
-e TZ=Asia/Shanghai mysql:8.0.27 \
--lower_case_table_names=1

#配置远程访问
docker exec -it mysql-replica1 /bin/bash
mysql -u root -p

ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
flush privileges;
```

- **从节点2**

```
mkdir -p /mysql/replication/replica2/data /mysql/replication/replica2/conf
/mysql/replication/replica2/log

#准备配置文件
vim /mysql/replication/replica2/conf/custom.cnf
[mysql]
# 设置mysql客户端默认编码
default-character-set=utf8

[mysqld]
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
datadir         = /var/lib/mysql

secure-file-priv= NULL

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

# 服务器唯一ID，默认是1
server-id=12

# 启用二进制日志
```

```
log-bin=mysql-bin

# 最大连接数
max_connections=1000

# 设置默认时区
default-time_zone='+8:00'

# 0:区分大小写
# 1:不区分大小写
lower_case_table_names=1

!includedir /etc/mysql/conf.d/

#运行mysql容器
docker run  -d  \
--name mysql-replica2 \
--privileged=true \
--restart=always \
--network  mysql-source-replica \
-p 3309:3306 \
-v /mysql/replication/replica2/data:/var/lib/mysql \
-v /mysql/replication/replica2/conf:/etc/mysql/conf.d  \
-v /mysql/replication/replica2/log:/logs \
-e MYSQL_ROOT_PASSWORD=123456 \
-e TZ=Asia/Shanghai mysql:8.0.27 \
--lower_case_table_names=1

#配置远程访问
docker exec -it mysql-replica2 /bin/bash
mysql -u root -p

ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
flush privileges;
```

```
[root@192-168-65-185 ~]# docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED       STATUS       PORTS
                    NAMES
e820c2b00698   mysql:8.0.27         "docker-entrypoint.s…"   4 hours ago   Up 4 hours   33060/tcp, 0.0.0.0:3309->3306/tc
p, :::3309->3306/tcp   mysql-replica2
32375ec71856   mysql:8.0.27         "docker-entrypoint.s…"   4 hours ago   Up 4 hours   33060/tcp, 0.0.0.0:3308->3306/tc
p, :::3308->3306/tcp   mysql-replica1
5241add23975   mysql:8.0.27         "docker-entrypoint.s…"   4 hours ago   Up 4 hours   33060/tcp, 0.0.0.0:3307->3306/tc
p, :::3307->3306/tcp   mysql-source
```

3）主库配置复制用户

每个副本使用一个 MySQL 用户名和密码连接到源，因此在源上必须有一个用户帐户，副本可以使用该帐户进行连接。

```
# 连接主库mysql-source
CREATE USER 'fox'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
GRANT REPLICATION SLAVE ON *.* TO 'fox'@'%';
flush privileges;
```

4）查看 master 机器的状态

使用 SHOW MASTER STATUS 语句确定当前二进制日志文件的名称和位置

```
# 主库上执行
SHOW MASTER STATUS;
```

```
mysql>  SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000003 |     1273 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

“文件”列显示日志文件的名称，“位置”列显示文件内的位置。

5) 从节点设置主库信息

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

在从库上执行 `CHANGE REPLICATION SOURCE TO` 语句(来自 MySQL 8.0.23)或 `CHANGE MASTER TO`语句(在 MySQL 8.0.23之前)

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='source_host_name',
    ->     MASTER_USER='replication_user_name',
    ->     MASTER_PASSWORD='replication_password',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;

Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO
```

```
    ->     SOURCE_HOST='source_host_name',
    ->     SOURCE_USER='replication_user_name',
    ->     SOURCE_PASSWORD='replication_password',
    ->     SOURCE_LOG_FILE='recorded_log_file_name',
    ->     SOURCE_LOG_POS=recorded_log_position;
```

从库1和从库2上执行

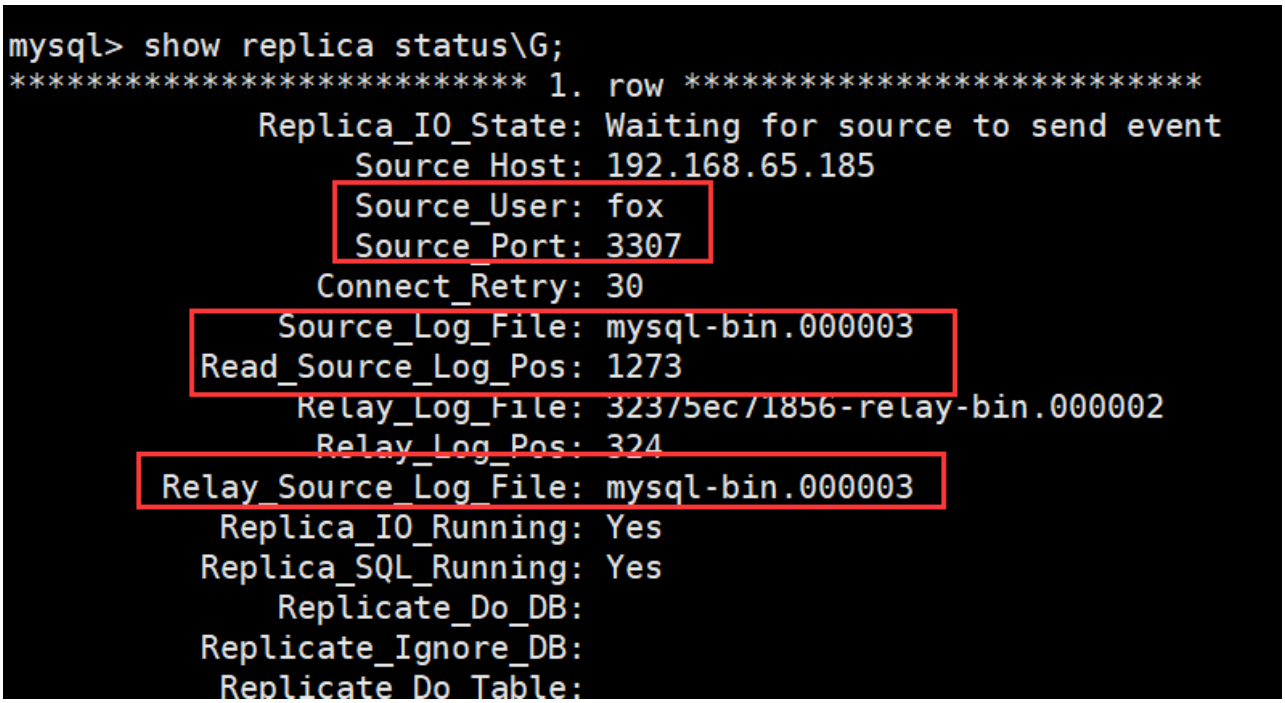
```
# from MySQL 8.0.23 执行下面的命令。
change replication source to source_host='192.168.65.185', source_user='fox',
source_password='123456', source_port=3307, source_log_file='mysql-bin.000003',
source_log_pos=1273, source_connect_retry=30;
```

> source_host：主数据库的IP地址；
> source_port：主数据库的运行端口；
> source_user：在主数据库创建的用于同步数据的用户账号；
> source_password：在主数据库创建的用于同步数据的用户密码；
> source_log_file：指定从数据库要复制数据的日志文件，通过查看主数据的状态，获取File参数；
> source_log_pos：指定从数据库从哪个位置开始复制数据，通过查看主数据的状态，获取Position参数；
> source_connect_retry：连接失败重试的时间间隔，单位为秒。

```
ype 'help;' or '\h' for help. Type '\c' to clear the current input statement.

ysql> change replication source to source_host='192.168.65.185', source_user='fox', source_password='123456', source_por
=3307, source_log_file='mysql-bin.000003', source_log_pos=1273, source_connect_retry=30;
uery OK, 0 rows affected, 2 warnings (1.37 sec)
```

6）开启从库

```
#开启从库
start slave;  或者  start replica;
#查看从库状态
show slave status \G;  或者 show replica status \G;
```



看到Replica_SQL_Running_State: Replica has read all relay log; waiting for more updates基本说明配置成功了，已经开始了主从复制。

7）测试主从复制功能

测试脚本

```
SET NAMES utf8mb4;
SET FOREIGN_KEY_CHECKS = 0;

CREATE DATABASE IF NOT EXISTS test;
USE test;
-- ----------------------------
-- Table structure for user
-- ----------------------------
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
  `name` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `address` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `last_updated` bigint(0) NULL DEFAULT NULL,
  `is_deleted` int(0) NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 2 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci ROW_FORMAT = Dynamic;

```

```
-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES (2, '张三', '广州白云山', 1691563465, 0);

SET FOREIGN_KEY_CHECKS = 1;
```

半同步复制示例

> 文档：https://dev.mysql.com/doc/refman/8.0/en/replication-semisync-installation.html

1）安装半同步插件

- 主节点

```
#from MySQL 8.0.26:
INSTALL PLUGIN rpl_semi_sync_source SONAME 'semisync_source.so';

#验证是否成功安装
SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%semi%';
```

```
mysql> INSTALL PLUGIN rpl_semi_sync_source SONAME 'semisync_source.so';
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
    ->        FROM INFORMATION_SCHEMA.PLUGINS
    ->        WHERE PLUGIN_NAME LIKE '%semi%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| rpl_semi_sync_source | ACTIVE        |
+----------------------+---------------+
1 row in set (0.00 sec)
```

- 从节点

```
#from MySQL 8.0.26:
INSTALL PLUGIN rpl_semi_sync_replica SONAME 'semisync_replica.so';
```

```
3
4 #验证是否成功安装
5 SELECT PLUGIN_NAME, PLUGIN_STATUS
6        FROM INFORMATION_SCHEMA.PLUGINS
7        WHERE PLUGIN_NAME LIKE '%semi%';
```

```
mysql> INSTALL PLUGIN rpl_semi_sync_replica SONAME 'semisync_replica.so';
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
    ->        FROM INFORMATION_SCHEMA.PLUGINS
    ->        WHERE PLUGIN_NAME LIKE '%semi%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| rpl_semi_sync_replica | ACTIVE        |
+----------------------+---------------+
1 row in set (0.00 sec)
```

2）开启半同步功能

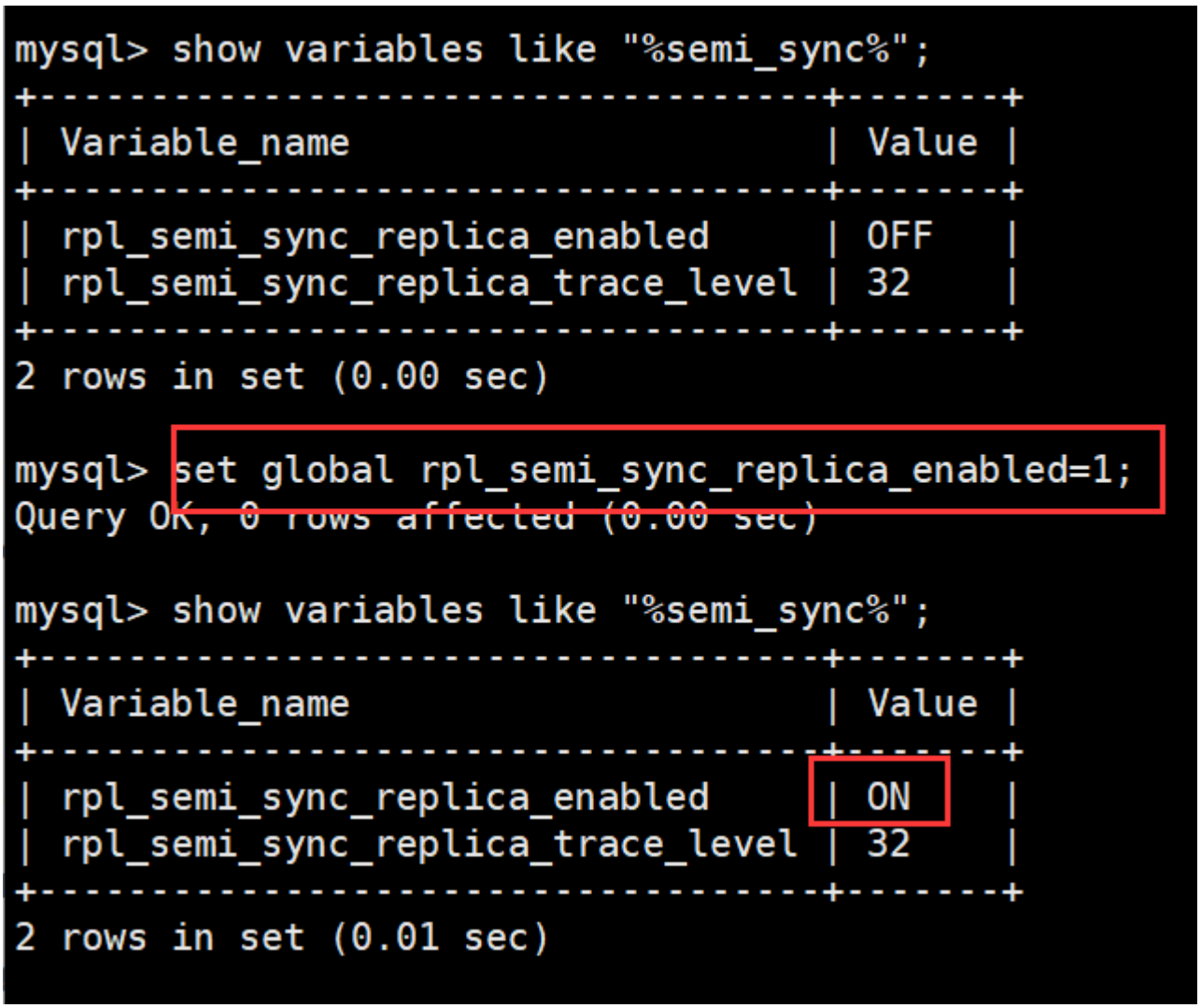
- 主节点

```
1 SET GLOBAL rpl_semi_sync_source_enabled=1;
2 #查看是否开启
3 show variables like "%semi_sync%";
```

```
mysql> SET GLOBAL rpl_semi_sync_source_enabled=1;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like "%semi_sync%";
+-----------------------------------------------+------------+
| Variable_name                                 | Value      |
+-----------------------------------------------+------------+
| rpl_semi_sync_source_enabled                  | ON         |
| rpl_semi_sync_source_timeout                  | 10000      |
| rpl_semi_sync_source_trace_level              | 32         |
| rpl_semi_sync_source_wait_for_replica_count   | 1          |
| rpl_semi_sync_source_wait_no_replica          | ON         |
| rpl_semi_sync_source_wait_point               | AFTER_SYNC |
+-----------------------------------------------+------------+
6 rows in set (0.00 sec)
```

- 从节点

```
set global rpl_semi_sync_replica_enabled=1;
```



3) 重启从节点上的I/O线程

如果在运行时启用副本上的半同步复制，您还必须启动复制I/O(接收端)线程(如果已经运行，则先停止它)，以使副本连接到源端并注册为半同步副本。

```
STOP REPLICA IO_THREAD;
START REPLICA IO_THREAD;
```

4)半同步复制测试

测试：当从节点响应超时时，主节点会将同步机制退化为异步复制。 从节点恢复后，同步机制是否会恢复为半同步复制

```
# 修改主节点半同步属性
```

```
set global rpl_semi_sync_source_wait_for_replica_count=2;
set global rpl_semi_sync_source_timeout=100000;

#停掉从节点2
docker stop mysql-replica2

#恢复从节点2
docker start mysql-replica2
#从节点2中执行
set global rpl_semi_sync_replica_enabled=1;
STOP REPLICA IO_THREAD;
START REPLICA IO_THREAD;
```

基于binlog位点主从复制痛点分析

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。

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

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）。

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。

GTID结构

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:

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

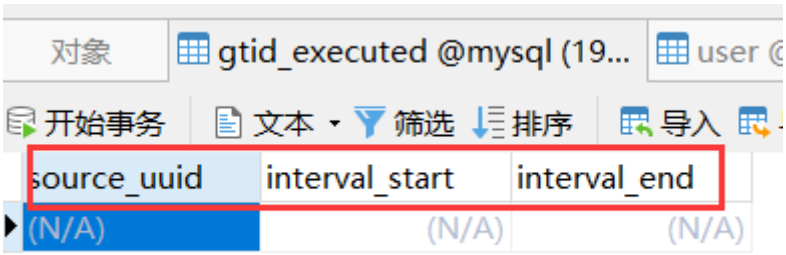
源自同一服务器的多个单一gtid或gtid范围也可以包含在单个表达式中，gtid或范围以冒号分隔，如下例所示:

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

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
```

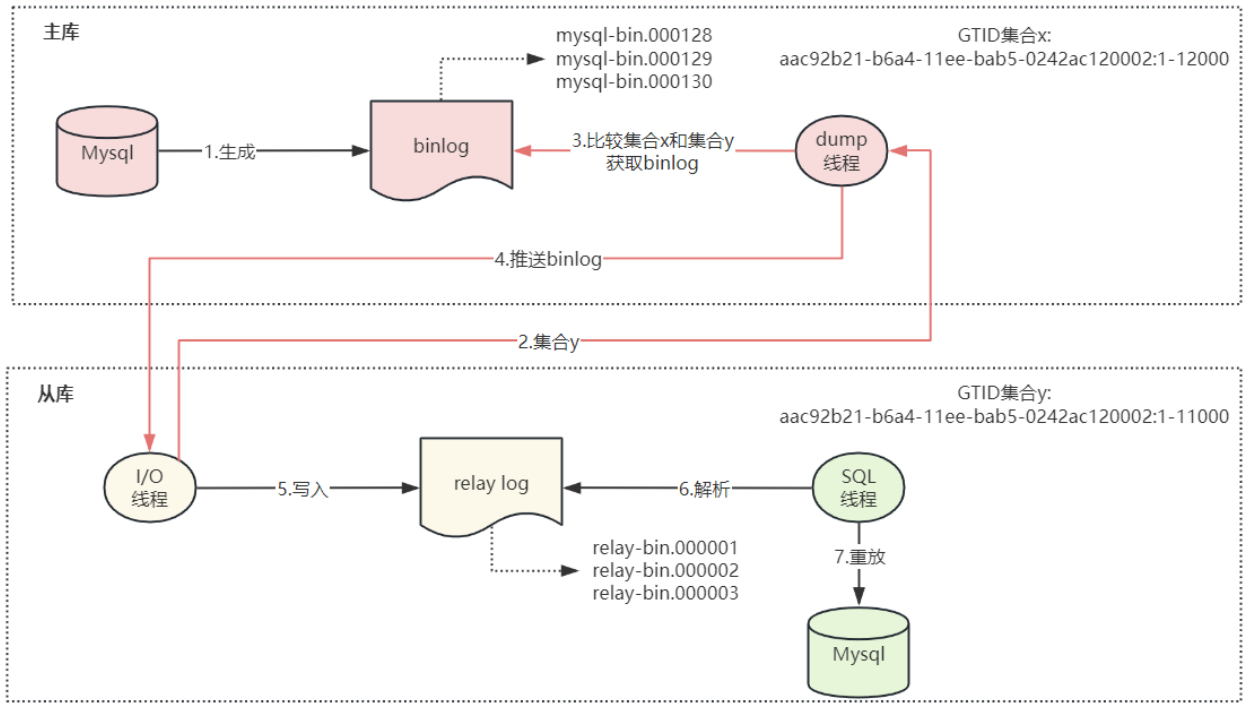
GTID存储在mysql数据库中名为gtid_executed的表中。该表中的一行包含它所代表的每个GTID或GTID集合的起始服务器的UUID，以及该集合的开始和结束事务id。



GTID工作原理

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。
- 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。

GTID的配置

1） 修改主库配置

修改主库的配置文件

```
#GTID:
#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
```

2）修改从库配置

修改从库配置文件

```
#GTID:
gtid_mode=on
enforce_gtid_consistency=on
```

从节点设置主库信息

```
# 从库配置同步参数
mysql> CHANGE MASTER TO
      >     MASTER_HOST = host,
      >     MASTER_PORT = port,
      >     MASTER_USER = user,
      >     MASTER_PASSWORD = password,
      >     MASTER_AUTO_POSITION = 1;

Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO
      >     SOURCE_HOST = host,
      >     SOURCE_PORT = port,
```

```
13      >     SOURCE_USER = user,
14      >     SOURCE_PASSWORD = password,
15      >     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。

基于GTID主从复制示例

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

在前面基于binlog日志主从复制的mysql服务上配置GTID复制。

1) 同步所有的mysql服务器

在主从服务器上都执行下面的命令：

```
# 设置MySQL服务器的全局只读模式
mysql> SET @@GLOBAL.read_only = ON;
```

> 注意：只有在使用已经在进行复制而不使用gtid的服务器时才需要此步骤。对于新服务器，请继续执行步骤3。

```
[root@192-168-65-185 ~]# docker exec -it mysql-source /bin/bash
root@5241add23975:/# mysql -uroot -p123456
mysql: [Warning] Skipping '!includedir /etc/mysql/conf.d/' directive as maximum include recursion level was reached in f
le /etc/mysql/conf.d/custom.cnf at line 31.
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 30
Server version: 8.0.27 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SET @@GLOBAL.read_only = ON;
Query OK, 0 rows affected (0.00 sec)
```

2) 停止所有的服务器

```
docker stop mysql-source mysql-replica1 mysql-replica2
```

3) 主从节点都启用GTID

修改custom.cnf

```
# 启用GTID
```

```
gtid_mode=ON
enforce-gtid-consistency=ON
```

启动主从节点

```
docker start mysql-source mysql-replica1 mysql-replica2
```

4）从节点配置基于GTID的自动定位

进入从节点，执行下面命令：

```
mysql> stop replica;
mysql> change replication source to source_host='192.168.65.185', source_user='fox', source_password='123456', source_port=3307,source_auto_position=1;
```

```
mysql> stop replica;
Query OK, 0 rows affected (0.39 sec)

mysql> change replication source to source_host='192.168.65.185', source_user='fox', source_password='123456', source_port=3307,source_auto_position=1;
Query OK, 0 rows affected, 2 warnings (1.12 sec)

mysql> start replica;
Query OK, 0 rows affected (0.38 sec)
```

5) 开启从库复制，并禁用只读模式

```
# 开启从库复制
mysql> start replica;
# 只有在步骤1中将服务器配置为只读时，才需要执行以下步骤。要允许服务器再次开始接受更新
mysql> SET @@GLOBAL.read_only = OFF;
```

查看从库状态是否正常

```
mysql> show replica status \G
```

```
                  Master_User: fox
                  Master_Port: 3307
                Connect_Retry: 30
              Master_Log_File: mysql-bin.000005
          Read_Master_Log_Pos: 156
               Relay_Log_File: 32375ec71856-relay-bin.000002
                Relay_Log_Pos: 371
        Relay_Master_Log_File: mysql-bin.000005
             Slave_IO_Running: Yes
            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: 156
              Relay_Log_Space: 587
              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: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 10
                  Master_UUID: aac92b21-b6a4-11ee-bab5-0242ac120002
             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:
```

主从切换演练

场景1： 模拟主库down机、从库1数据同步完成、从库2数据未同步完成

1）从库2停止复制

```
mysql> stop replica;
```

2）主库创建测试数据

```
INSERT INTO `test`.`user` VALUES (12, 'fox', NULL, NULL, NULL);
```

3) 查询数据

从库1

```
mysql> select * from test.user;
+----+--------+-----------------+--------------+------------+
| id | name   | address         | last_updated | is_deleted |
+----+--------+-----------------+--------------+------------+
|  2 | 张三   | 广州白云山      |   1691563465 |          0 |
|  9 | mark   | NULL            |         NULL |       NULL |
| 10 | fork   | NULL            |         NULL |       NULL |
| 11 | 楼兰   | NULL            |         NULL |       NULL |
| 12 | fox    | NULL            |         NULL |       NULL |
+----+--------+-----------------+--------------+------------+
```

从库2

```
mysql> select * from test.user;
+----+--------+-----------------+--------------+------------+
| id | name   | address         | last_updated | is_deleted |
+----+--------+-----------------+--------------+------------+
|  2 | 张三   | 广州白云山      |   1691563465 |          0 |
|  9 | mark   | NULL            |         NULL |       NULL |
| 10 | fork   | NULL            |         NULL |       NULL |
| 11 | 楼兰   | NULL            |         NULL |       NULL |
+----+--------+-----------------+--------------+------------+
```

很显然，从库1同步了最新数据，比从库2数据新

场景2：将主库宕机，从库1升级为主库、从库2切换主库为从库1（新的主库），观察从库2是否同步未完成的事务

1）停止主库

```
docker stop mysql-source
```

2）设置新主库

设置replica1 为replica2的主库，因为replica1的数据是完整的。

按照普通复制方法，需要计算主库的log_pos和从库设置成主库的log_pos，可能出现错误

因为同一事务的GTID在所有节点上的值一致，那么根据replica2当前停止点的GTID就能定位到要主库的GTID，所以直接在replica2上执行change即可

```
# replica1上创建复制用户
CREATE USER 'fox'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
GRANT REPLICATION SLAVE ON *.* TO 'fox'@'%';
flush privileges;
```

```
# replica2上执行 从replica1查询
mysql> stop replica;
mysql> change replication source to source_host='192.168.65.185',source_port=3308,source_user='fox',source_password='123456',source_auto_position=1;
 mysql> start replica;
```

```
mysql> stop replica;
Query OK, 0 rows affected (0.10 sec)

mysql> change replication source to
    ->   source_host='192.168.65.185',
    ->   source_port=3308,
    ->   source_user='fox',
    ->   source_password='123456',
    ->   source_auto_position=1;
Query OK, 0 rows affected, 2 warnings (2.10 sec)

mysql> start replica;
Query OK, 0 rows affected (0.65 sec)
```

查询同步结果

```
mysql> select * from test.user;
+----+--------+-----------------+--------------+------------+
| id | name   | address         | last_updated | is_deleted |
+----+--------+-----------------+--------------+------------+
|  2 | 张三   | 广州白云山      |   1691563465 |          0 |
|  9 | mark   | NULL            |         NULL |       NULL |
| 10 | fork   | NULL            |         NULL |       NULL |
| 11 | 楼兰   | NULL            |         NULL |       NULL |
| 12 | fox    | NULL            |         NULL |       NULL |
+----+--------+-----------------+--------------+------------+
```

场景3：模拟从库删除测试表，主库对表进行插入操作。观察从库复制是否报错

1）replica1删除test.user表，主库插入新记录

```
# 从库1 删除user表
drop table test.user;
# 主库插入新记录
INSERT INTO `test`.`user` VALUES (14, 'AAA', NULL, NULL, NULL);
```

2) 查看从库同步情况

```
# 从库1执行
mysql> show replica status\G
```

```
          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 'aac92b21-b6a4-11ee-bab5-0242ac120002:6' at master log mysql-bin.000008, end
log_pos 1335. 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_Source_Log_Pos: 1078
```

报错信息：事务aac92b21-b6a4-11ee-bab5-0242ac120002:6执行失败

Coordinator stopped because there were error(s) in the worker(s). The most recent failure being: Worker 1 failed executing transaction 'aac92b21-b6a4-11ee-bab5-0242ac120002:6' at master log mysql-bin.000008, end_log_pos 1335. See error log and/or performance_schema.replication_applier_status_by_worker table for more details about this failure or others, if any.

3)在主库继续进行其他事务，观察gitd是否复制成功

```
# 主库插入新记录
INSERT INTO `test`.`user` VALUES (15, 'BBB', NULL, NULL, NULL);
```

从库状态

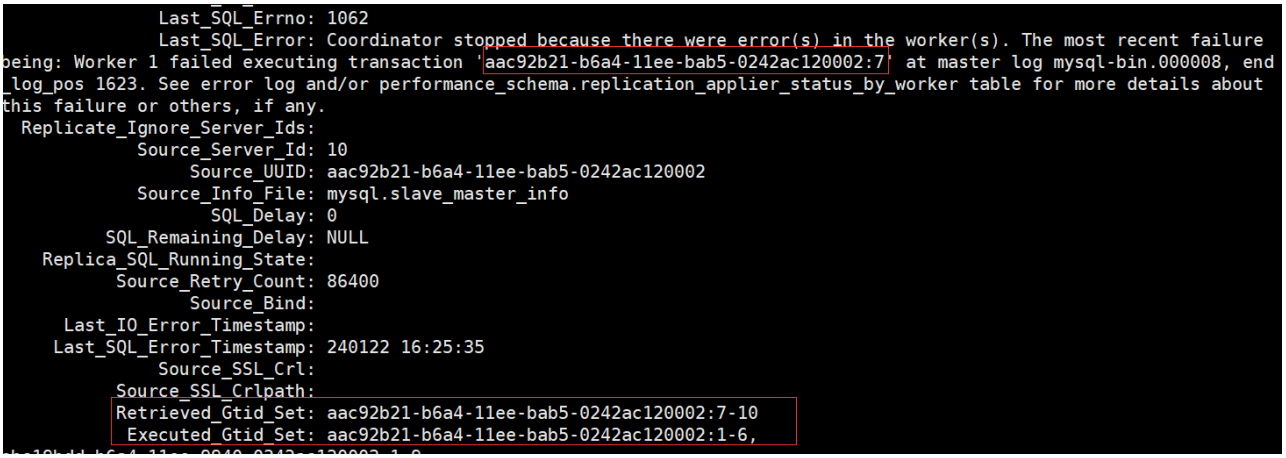
```
             Last_SQL_Errno: 1146
             Last_SQL_Error: Coordinator stopped because there were error(s) in the worker(s). The most recent failure
being: Worker 1 failed executing transaction 'aac92b21-b6a4-11ee-bab5-0242ac120002:6' at master log mysql-bin.000008, end
log_pos 1335. See error 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:
             Source_Server_Id: 10
                  Source_UUID: aac92b21-b6a4-11ee-bab5-0242ac120002
             Source_Info_File: mysql.slave_master_info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
    Replica_SQL_Running_State:
           Source_Retry_Count: 86400
                  Source_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp: 240122 15:40:53
               Source_SSL_Crl:
           Source_SSL_Crlpath:
           Retrieved_Gtid_Set: aac92b21-b6a4-11ee-bab5-0242ac120002:4-7
            Executed_Gtid_Set: aac92b21-b6a4-11ee-bab5-0242ac120002:1-5,
cbc19bdd-b6a4-11ee-9940-0242ac120003:1-6
```

可以看出从库复制中断 （注意：删除了表，无法插入）

4）复制中断修复：采用从库跳过错误事务修复

因为从库user表已经删了（user表中部分数据不是利用gtid复制过去的），先从主库将表数据拷贝到从库

获取从库最新状态



从库执行

```
# 1.停止从库1复制进程
mysql> stop replica;
# 2.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= 'aac92b21-b6a4-11ee-bab5-0242ac120002:7';
# 3.设置空事物
mysql> BEGIN; COMMIT;
# 4.恢复自增事物号
mysql> SET SESSION GTID_NEXT = AUTOMATIC;
# 5.启动从库1复制进程
mysql> start replica;
# 再次查询会发现主库数据已经同步过来了
mysql> select * from test.user;
```

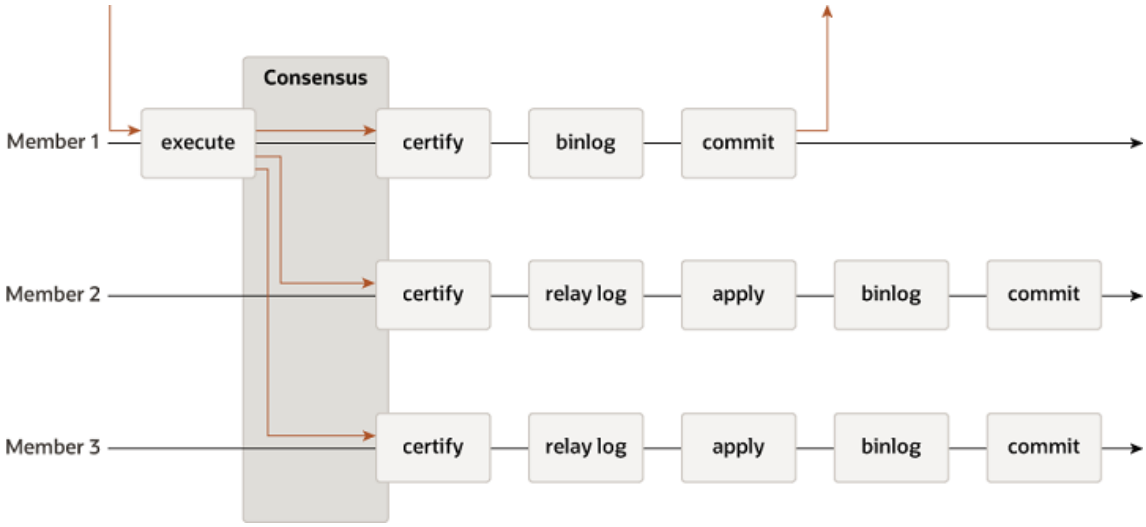
```
nysql> SET @@SESSION.GTID_NEXT= 'aac92b21-b6a4-11ee-bab5-0242ac120002:7';
Query OK, 0 rows affected (0.00 sec)

nysql> BEGIN; COMMIT;
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.24 sec)

nysql> SET SESSION GTID_NEXT = AUTOMATIC;
Query OK, 0 rows affected (0.01 sec)

nysql> start replica;
Query OK, 0 rows affected (0.32 sec)

nysql> select * from test.user;
+----+--------+-----------------+--------------+------------+
| id | name   | address         | last_updated | is_deleted |
+----+--------+-----------------+--------------+------------+
|  2 | 张三   | 广州白云山      |   1691563465 |          0 |
|  9 | mark   | NULL            |         NULL |       NULL |
| 10 | fork   | NULL            |         NULL |       NULL |
| 11 | 楼兰   | NULL            |         NULL |       NULL |
| 14 | AAA    | NULL            |         NULL |       NULL |
| 15 | BBB    | NULL            |         NULL |       NULL |
| 16 | CCC    | NULL            |         NULL |       NULL |
+----+--------+-----------------+--------------+------------+
7 rows in set (0.00 sec)
```

组复制（Group Replication）

> 文档：https://dev.mysql.com/doc/refman/8.0/en/group-replication.html

什么是组复制

由于传统异步复制的缺陷，可能会导致主从数据不一致的问题，在主节点异常宕机时从节点可能造成数据丢失。基于这个缺陷，Mysql5.7.17推出了一个高可用与高扩展的解决方案Mysql Group Replication(简称MGR)，将原有的gtid复制功能进行了增强，支持单主模式和多主模式。组复制在数据库层面上做到了只要集群中大多数主机可用，则服务可用，也就是说3台服务器的集群，允许其中1台宕机。



Group Replication提供了分布式状态机复制，服务器之间具有很强的协调性。当服务器属于同一组时，它们会自动进行协调。该组可以在具有自动选主的单主模式下运行，在这种模式下，一次只有一台服务器接受更新。或者，对于更高级的用户，可以在多主模式下部署该组，其中所有服务器都可以接受更新，即使它们是并发执行的。

与传统复制相比，Group Replication有以下大幅改进：

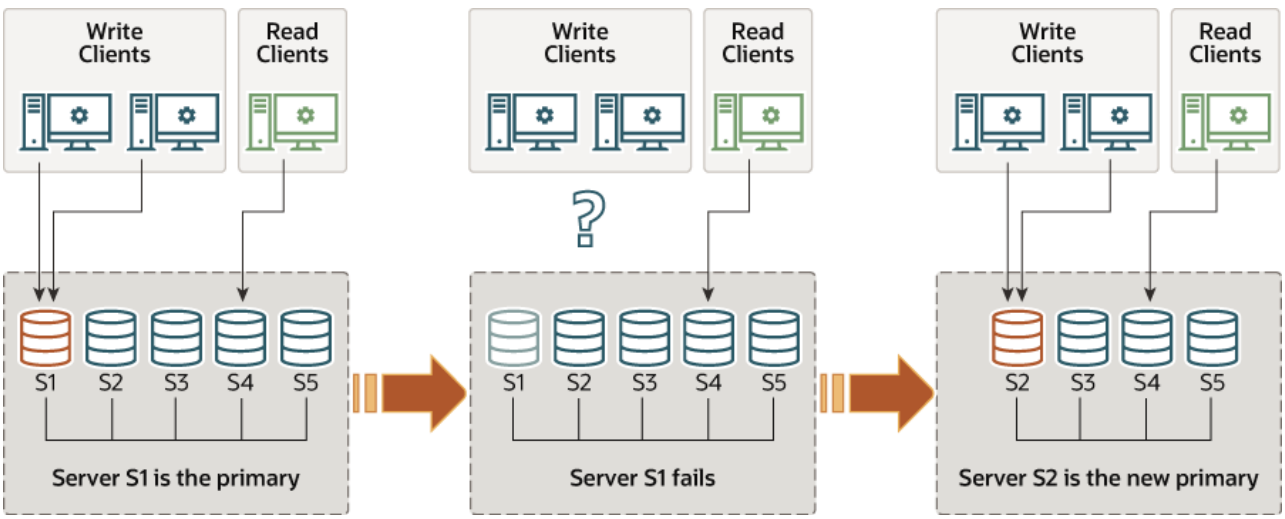
- 传统复制的主从复制方式有一个主和不等数量的从。主节点执行的事务会异步发送给从节点，在从节点重新执行。而Group Replication采用整组写入的方式，避免了单点争用。
- Group Replication在传输数据时使用了Paxos协议。Paxos协议保证了数据传输的一致性和原子性。基于Paxos协议，Group Replication构建了一个分布式的状态复制机制，这是实现多主复制的核心技术。
- Group Replication提供了多写方案，为多活方案带来了实现的可能。

MGR 能保证数据库服务的连续可用，却无法处理如下问题：当一个组成员变为不可用时，连接到它的客户端必须被重定向或故障转移到其他组成员。此时需要使用连接器、负载均衡器、路由器或某种形式的中间件，例如 MySQL Router 8.0 。MGR 本身不提供这些工具。此时便引入了 InnoDB Cluster ，后面详述。

单主模式

组中的每个MySQL服务器实例都可以在独立的物理主机上运行，这是部署组复制的推荐方式。

在单主模式下(group_replication_single_primary_mode=ON)，组中只有一个主服务器，该主服务器被设置为读写模式。组中的所有其他成员都被设置为只读模式(super_read_only=ON)。



查找primary的两种方式

```
# 方式1：查询performance_schema.replication_group_members的MEMBER_ROLE列
mysql> SELECT MEMBER_HOST, MEMBER_ROLE FROM
performance_schema.replication_group_members;
+--------------------------+-------------+
| MEMBER_HOST              | MEMBER_ROLE |
+--------------------------+-------------+
| remote1.example.com      | PRIMARY     |
| remote2.example.com      | SECONDARY   |
| remote3.example.com      | SECONDARY   |
+--------------------------+-------------+

# 方式2：查看group_replication_primary_member变量状态
mysql> SHOW STATUS LIKE 'group_replication_primary_member';
```

单主模式部署示例

> 文档：https://dev.mysql.com/doc/refman/8.0/en/group-replication-configuring-instances.html

1) 部署三个MySQL Server实例

可以利用docker快速部署3个MySQL实例

角色	server_id	DB Port	内部通信
mgr-node1(primary)	1	3321>3306	mgr-node1:33061
mgr-node2(Secondary)	2	3322>3306	mgr-node2:33061
mgr-node3(Secondary)	3	3323>3306	mgr-node3:33061

```
# 创建组复制的网络  保证三个mysql容器之间可以通过容器名访问
docker network create --driver bridge mgr-network
mkdir -p /mysql/mgr/node1/data /mysql/mgr/node1/conf /mysql/mgr/node1/log
mkdir -p /mysql/mgr/node2/data /mysql/mgr/node2/conf /mysql/mgr/node2/log
mkdir -p /mysql/mgr/node3/data /mysql/mgr/node3/conf /mysql/mgr/node3/log

#运行mysql容器
docker run  -d  \
--name mgr-node1 \
```

```
--privileged=true \
--restart=always \
--network  mgr-network \
-p 3321:3306 \
-v /mysql/mgr/node1/data:/var/lib/mysql \
-v /mysql/mgr/node1/conf:/etc/mysql/conf.d  \
-v /mysql/mgr/node1/log:/logs \
-e MYSQL_ROOT_PASSWORD=123456 \
-e TZ=Asia/Shanghai mysql:8.0.27 \
--lower_case_table_names=1

docker run  -d  \
--name mgr-node2 \
--privileged=true \
--restart=always \
--network  mgr-network \
-p 3322:3306 \
-v /mysql/mgr/node2/data:/var/lib/mysql \
-v /mysql/mgr/node2/conf:/etc/mysql/conf.d  \
-v /mysql/mgr/node2/log:/logs \
-e MYSQL_ROOT_PASSWORD=123456 \
-e TZ=Asia/Shanghai mysql:8.0.27 \
--lower_case_table_names=1

docker run  -d  \
--name mgr-node3 \
--privileged=true \
--restart=always \
--network  mgr-network \
-p 3323:3306 \
-v /mysql/mgr/node3/data:/var/lib/mysql \
-v /mysql/mgr/node3/conf:/etc/mysql/conf.d  \
-v /mysql/mgr/node3/log:/logs \
-e MYSQL_ROOT_PASSWORD=123456 \
-e TZ=Asia/Shanghai mysql:8.0.27 \
--lower_case_table_names=1
```

2）配置组复制实例

以mgr-node1配置为例，创建/mysql/mgr/node1/conf/custom.cnf，添加以下配置：

```
1
2  [mysql]
3  # 设置mysql客户端默认编码
4  default-character-set=utf8
5
6  [mysqld]
7  pid-file        = /var/run/mysqld/mysqld.pid
8  socket          = /var/run/mysqld/mysqld.sock
9  datadir         = /var/lib/mysql
10
11 secure-file-priv= NULL

12 # Disabling symbolic-links is recommended to prevent assorted security risks
13 symbolic-links=0
14
15 #对于Group Replication，数据必须存储在InnoDB事务存储引擎中
16 disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"
17
18 #指定sever_id，三个Mysql实例需要分别改为对应的sever_id
19 server_id=1
20 # 必须开启GTID支持
21 gtid_mode=ON
22 enforce_gtid_consistency=ON
23
24 # 启用二进制日志
25 log-bin=mysql-bin
26
27 #组复制设置
28 #实例启动时会将组复制插件加载到插件列表中
29 plugin_load_add='group_replication.so'
30 # 组名三个节点必须保证一致，必须是UUID，可以使用 SELECT UUID()生成一个
31 group_replication_group_name="117dc7ea-b9bd-11ee-9bdb-0242ac120002"
32 # 插件在服务器启动时不自动启动，可以等配置好服务器之后手动启动
33 group_replication_start_on_boot=off
34 # 配置与组内其他成员通信是使用的主机名和端口,内部通讯端口，推荐使用 33061
35 group_replication_local_address= "mgr-node1:33061"
36 # 设置组成员的主机名和端口
37 group_replication_group_seeds= "mgr-node1:33061,mgr-node2:33061,mgr-node3:33061"
38 # 通常会在实例运行时配置group_replication_bootstrap_group，以确保只有一个成员实际引导组
```

```
group_replication_bootstrap_group=off


# 最大连接数
max_connections=10000

# 设置默认时区
default-time_zone='+8:00'

# 0:区分大小写
# 1:不区分大小写
lower_case_table_names=1

!includedir /etc/mysql/conf.d/
```

mgr-node1和mgr-node3同上，注意配置文件路径和修改server_id和group_replication_local_address

```
#对于Group Replication，数据必须存储在InnoDB事务存储引擎中
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"

#指定sever_id，三个Mysql实例需要分别改为对应的sever_id
server_id=2
# 必须开启GTID支持
gtid_mode=ON
enforce_gtid_consistency=ON

# 启用二进制日志
log-bin=mysql-bin

#组复制设置
plugin_load_add='group_replication.so'
# 组名三个节点必须保证一致，必须是UUID，可以使用 SELECT UUID()生成一个
group_replication_group_name="117dc7ea-b9bd-11ee-9bdb-0242ac120002"
# 插件在服务器启动时不自动启动，可以等配置好服务器之后手动启动
group_replication_start_on_boot=off
# 配置与组内其他成员通信是使用的主机名/IP和端口
group_replication_local_address= "mgr-node2:33061"
# 设置组成员的主机名和端口
group_replication_group_seeds= "mgr-node1:33061,mgr-node2:33061,mgr-node3:33061"
# 通常会在实例运行时配置group_replication_bootstrap_group，以确保只有一个成员实际引导组
group_replication_bootstrap_group=off
```

对应mgr-node2的配置

3）配置用于分布式恢复的用户凭证

mgr-node1上配置

3.1）重新启动mysql实例

```
docker restart mgr-node1 mgr-node2 mgr-node3
```

3.2）禁用二进制日志记录，以便在每个实例上分别创建复制用户

```
# mgr-node1为例
[root@192-168-65-185 ~]# docker exec -it mgr-node1 /bin/bash
root@0955d5f390c6:/# mysql -uroot -p123456
mysql> SET SQL_LOG_BIN=0;

```

3.3) 创建MySQL用户,授予所需的权限

```
mysql>  CREATE USER fox@'%' IDENTIFIED  BY '123456';
        GRANT REPLICATION SLAVE ON *.* TO fox@'%';
        GRANT CONNECTION_ADMIN ON *.* TO fox@'%';
        GRANT BACKUP_ADMIN ON *.* TO fox@'%';
        GRANT GROUP_REPLICATION_STREAM ON *.* TO fox@'%';
        FLUSH PRIVILEGES;
```

3.4）如果前面禁用了二进制日志，再次启用二进制日志

```
mysql> SET SQL_LOG_BIN=1;
```

3.5）创建复制用户后，必须向服务器提供用于分布式恢复的用户凭据。

- 使用CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO设置的用户凭据以明文形式存储在服务器上的复制元数据存储库中。当组复制启动时，它们将被应用，包括当系统变量group_replication_start_on_boot设置为ON时自动启动。
- 在START GROUP_REPLICATION上指定的用户凭据仅保存在内存中，并通过STOP GROUP_REPLICATION语句或服务器关闭来删除。必须发出START GROUP_REPLICATION语句来再次提供凭据，因此无法使用这些凭据自动启动Group Replication。这种指定用户凭据的方法有助于保护组复制服务器免受未经授权的访问。此功能从MySQL 8.0.21开始支持。

```
#from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO SOURCE_USER='fox', SOURCE_PASSWORD='123456' FOR
CHANNEL 'group_replication_recovery';
```

注意：如果不想配置ssl，可以配置参数group_replication_recovery_public_key_path=ON来在请求复制用户密钥时给公钥

```
#实例加入集群需要获取公钥
set global group_replication_recovery_get_public_key=on;
```

4) 指定mgr-node1引导组（primary节点），启用组复制

为了安全地引导组，连接到mgr-node1并执行以下语句：

```
#mgr-node1启动组复制，并且作为primary
mysql> SET GLOBAL group_replication_bootstrap_group=ON;
        START GROUP_REPLICATION;
        SET GLOBAL group_replication_bootstrap_group=OFF;
# 查询组成员信息，mgr-node1是否是primary
mysql> SELECT MEMBER_HOST, MEMBER_ROLE FROM
performance_schema.replication_group_members;
```

```
mysql> SET GLOBAL group_replication_bootstrap_group=ON;
Query OK, 0 rows affected (0.00 sec)

mysql>         START GROUP_REPLICATION;
Query OK, 0 rows affected (2.98 sec)

mysql>         SET GLOBAL group_replication_bootstrap_group=OFF;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT MEMBER_HOST, MEMBER_ROLE FROM performance_schema.replication_group_members;
+--------------+-------------+
| MEMBER_HOST  | MEMBER_ROLE |
+--------------+-------------+
| 0955d5f390c6 | PRIMARY     |
+--------------+-------------+
1 row in set (0.01 sec)
```

为了验证后续其他节点入组情况，下面将创建一个表并向其中添加一些数据进行验证。

```
CREATE DATABASE IF NOT EXISTS test;
USE test;
CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL);
INSERT INTO t1 VALUES (1, 'Fox');
```

5）向组中添加实例mgr-node2和mgr-node3

```
#添加复制用户
```

```
SET SQL_LOG_BIN=0;
CREATE USER fox@'%' IDENTIFIED BY '123456';
GRANT REPLICATION SLAVE ON *.* TO fox@'%';
GRANT CONNECTION_ADMIN ON *.* TO fox@'%';
GRANT BACKUP_ADMIN ON *.* TO fox@'%';
GRANT GROUP_REPLICATION_STREAM ON *.* TO fox@'%';
FLUSH PRIVILEGES;
SET SQL_LOG_BIN=1;
CHANGE REPLICATION SOURCE TO SOURCE_USER='fox', SOURCE_PASSWORD='123456' FOR CHANNEL 'group_replication_recovery';
set global group_replication_recovery_get_public_key=on;

# mgr-node2和mgr-node3启用主复制
mysql> START GROUP_REPLICATION;
# 查询组成员信息
mysql> SELECT MEMBER_HOST, MEMBER_ROLE FROM performance_schema.replication_group_members;
```

```
mysql> START GROUP_REPLICATION;
Query OK, 0 rows affected (3.69 sec)

mysql> SELECT MEMBER_HOST, MEMBER_ROLE FROM performance_schema.replication_group_members;
+--------------+-------------+
| MEMBER_HOST  | MEMBER_ROLE |
+--------------+-------------+
| 0955d5f390c6 | PRIMARY     |
| 89898361f7b6 | SECONDARY   |
| 73aacd314597 | SECONDARY   |
+--------------+-------------+
3 rows in set (0.00 sec)
```

确认数据同步情况

```
SELECT * FROM test.user;
```

```
mysql> START GROUP_REPLICATION;
ERROR 3092 (HY000): The server is not configured properly to be an active member of the group. Please see more details on
 error log.
mysql> reset master;
Query OK, 0 rows affected (0.35 sec)

mysql> reset slave;
Query OK, 0 rows affected, 1 warning (2.41 sec)

mysql> START GROUP_REPLICATION;
Query OK, 0 rows affected (11.88 sec)

mysql> SELECT * FROM test.t1;
+----+-----+
| c1 | c2  |
+----+-----+
|  1 | Fox |
+----+-----+
1 row in set (0.00 sec)
```

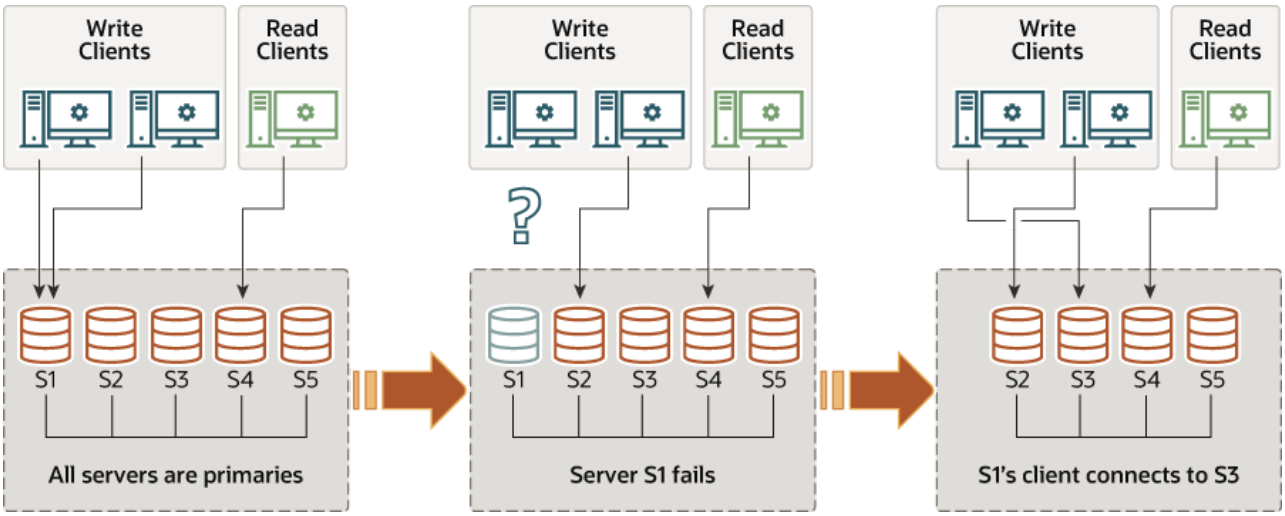
如果数据没有正常同步，可以通过docker log -f mgr-node2排查问题

多主模式

> 文档：https://dev.mysql.com/doc/refman/8.0/en/group-replication-multi-primary-mode.html

在多主模式下(group_replication_single_primary_mode=OFF)，没有成员具有特殊的角色。任何与其他组成员兼容的成员在加入组时都被设置为读写模式，并且可以处理写事务，即使它们是并发发布的。

如果一个成员停止接受写事务，例如，在服务器意外退出的情况下，连接到它的客户端可以被重定向或故障转移到处于读写模式的任何其他成员。Group Replication本身不处理客户端故障转移，因此需要使用中间件框架(如MySQL Router 8.0)、代理、连接器或应用程序本身来安排。



多主模式部署示例

在前面单主模式基础上改为多主模式

1）三个节点都关闭单主模式

三个节点都执行如下操作

```
# 停止组复制
stop GROUP_REPLICATION;

# 关闭单主模式
set global group_replication_single_primary_mode=off;

# 开启多主一致性检查
set global group_replication_enforce_update_everywhere_checks=ON;
```

```
ysql> #
ysql> stop GROUP_REPLICATION;
uery OK, 0 rows affected (4.10 sec)

ysql>
ysql> #
ysql> set global group_replication_single_primary_mode=off;
uery OK, 0 rows affected (0.00 sec)

ysql>
ysql> #
ysql> set global group_replication_enforce_update_everywhere_checks=ON;
uery OK, 0 rows affected (0.00 sec)
```

2）选择mgr-node1引导组复制

```
# 开启组复制引导
set global group_replication_bootstrap_group=on;
# 开启组复制
start group_replication;

# 关闭组复制引导
set global group_replication_bootstrap_group=off;
```

```
mysql> set global group_replication_enforce_update_everywhere_checks=ON;
Query OK, 0 rows affected (0.00 sec)

mysql> #
mysql> set global group_replication_bootstrap_group=on;
Query OK, 0 rows affected (0.00 sec)

mysql> #
mysql> start group_replication;
Query OK, 0 rows affected (2.77 sec)

mysql>
mysql> #
mysql> set global group_replication_bootstrap_group=off;
Query OK, 0 rows affected (0.00 sec)
```

3）mgr-node2和mgr-node3开启组复制

```
# 开启组复制
start group_replication;

# 查看到添加到组复制集群的服务器信息
select * from performance_schema.replication_group_members;
```

```
mysql> start group_replication;
Query OK, 0 rows affected (3.49 sec)

mysql> select * from performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+--------------+-------------
----+---------------+----------------------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE | MEMBER_R
OLE | MEMBER_VERSION | MEMBER_COMMUNICATION_STACK |
+---------------------------+--------------------------------------+-------------+-------------+--------------+-------------
----+---------------+----------------------------+
| group_replication_applier | d9c7cf54-b9bb-11ee-b180-0242ac130002 | c38b1bc9f9c7 |        3306 | ONLINE       | PRIMARY
    | 8.0.27         | XCom                       |
| group_replication_applier | dd7bb0de-b9bb-11ee-883e-0242ac130003 | 5528b6a1a2a4 |        3306 | ONLINE       | PRIMARY
    | 8.0.27         | XCom                       |
| group_replication_applier | edee7e73-b9bb-11ee-8012-0242ac130004 | 68f94a233ffe |        3306 | ONLINE       | PRIMARY
    | 8.0.27         | XCom                       |
+---------------------------+--------------------------------------+-------------+-------------+--------------+-------------
----+---------------+----------------------------+
```

4) 测试

- mgr-node1插入数据，看mgr-node2是否可以查到

```
#mgr-node1
INSERT INTO test.t1 VALUES (2, 'aaa');
#mgr-node2
select * from test.t1;
```