# 使用MySQL组复制的限制和局限性

本节列出和解释了组复制相关的要求和限制。

## 1.组复制的要求

要使用组复制，每个MySQL节点必须满足以下条件：

### 1.1 基本要求

1. InnoDB存储引擎：数据必须存储在事务型的InnoDB存储引擎中。事务以乐观形式执行，然后在提交前会检测冲突问题。如果有冲突，为了维护组中一致性，有些事务必须回滚。这意味着需要事务型的存储引擎。此外，InnoDB 存储引擎提供了一些额外的功能，它们结合组复制时能更好地管理和处理冲突。
2. Primary Keys：每张需要被组复制的表都必须显式定义一个主键。主键在判断事务是否冲突扮演极其重要的角色：通过主键来准确识别每个事务中修改了表中的哪些行。(实际上是将主机hash成写集，然后由certifier来并发事务之间的检测冲突性)
3. 使用IPv4 地址：MySQL组复制使用的组通信引擎组件只支持 IPv4。因此，必须使用IPv4的网络。
4. 良好的网络性能：组复制设计的初衷是部署在集群环境中，集群中的节点相互之间都非常近，因此除了网络延迟，网络带宽也会影响组复制。

### 1.2 配置上的要求

组中的每个成员都必须配置以下选项：

1. 必须开启二进制日志：设置`--log-bin[=log_file_name]`。MySQL组复制会复制二进制日志的内容，因此必须开启二进制日志。
2. Slave Updates Logged：设置`--log-slave-updates`。节点需要记录applier已应用的日志。组中的每个节点都需要记录它们所接收到并应用的所有事务，这是必须的，因为恢复过程是依赖于组中参与者的二进制日志来进行的。因此，组中每个成员都必须保留每个事务的副本，即使某事务不是在该节点上开始的。
3. Row Format的二进制日志：设置`--binlog-format=row`。组复制依赖于基于行格式的二进制日志，以便在组中传播所发生的更改能保持一致性。而且，在探测组中不同节点间发生的并发事务是否冲突时，需要从行格式的日志中提取一些内容来做比较。
4. 开启GTID复制：设置`--gtid-mode=ON`。组复制使用GTID(全局事务ID)来精确跟踪每个节点上已经提交了哪些事务。也因此可以推断出某节点上要执行的事务是否和已执行的事务(每个节点上都有副本)冲突。换句话说，GTID是整个组复制判断事务是否冲突的基础。
5. Replication Information Repositories：设置`--master-info-repository=TABLE`和`--relay-log-info-repository=TABLE`。applier需要将 master 和 relay log 的元数据信息写入到系统表 mysql.slave_master_info 和 mysql.slave_relay_log_info 中。这保证了组复制插件具有一致性恢复的能力和复制的元数据事务管理能力。
6. Transaction Write Set Extraction：设置`--transaction-write-setextraction=XXHASH64`，以便将行写入到二进制日志中时，节点也收集写集。写集基于每行的主键，是唯一标识被更改行的标签的简化形式，该标签后续会用于探测事务冲突性。
7. Multithreaded Appliers：(某些旧版本没有该要求)可以将组复制成员配置为多线程appliers，使得可以并行应用事务。需要设置`--slave-parallel-workers=N`(N是applier线程数量)、`--slavepreserve-commit-order=1`以及`--slave-parallel-type=LOGICAL_CLOCK`。`--slaveparallel-workers=N`表示启用多applier线程，组复制依赖于建立在所有参与节点都以相同顺序接收和应用、提交事务的一致性机制，因此还必须设置`--slave-preserve-commit-order=1`以保证并行事务的最终提交是和原事务所在顺序位置一致的。最后，为了决定哪些事务可以并行执行，relay log 必须包含由`--slave-parallel-ype=LOGICAL_CLOCK`生成的事务父信息(transaction parent information)。当尝试加入一个只设置了`--slave-parallel-workers`大于0，却没有设置其他两项的新成员，将会报错并阻止它的加入。

## 2.组复制的限制(局限性)

下面是使用组复制已知的限制：

资料整理
火龙果软件
荐

1. Replication Event Checksums：由于对复制事件校验的设计缺陷，目前组复制不能使用它们。因此，需要设置`--binlog-checksum=NONE`。
2. Gap Locks：在验证阶段中(certification process)，不会考虑 Gap Locks，因此在 InnoDB 的外部无法获取任何关于Gap 锁的信息。

> 注意:
>
> 除非你的应用程序或业务需求依赖于REPEATABLE READ(MySQL默认该隔离级别)，否则建议在组复制中使用READ COMMITTED隔离级别。在READ COMMITTED隔离级别中，InnoDB基本上不会使用Gap Locks，这将使得InnoDB自带的冲突探测能和组复制的冲突探测相互对齐从而保持一致。

1. Table Locks and Named Locks：验证阶段(certification process)中不考虑表锁和命名锁(见get_lock())。
2. 不支持 SERIALIZABLE 隔离级别：在多主模型下，默认不支持该隔离级别。如果在多主模型下设置了该隔离级别，将拒绝提交事务。
3. 不支持并发的 DDL 和 DML 操作：不支持在多主模型下不同节点上同时执行DDL和DML修改同一对象。在某节点上对某对象执行DDL语句时，如果在其他节点同时执行DML修改该对象，将有一定风险探测到冲突。(译注：是 DDL+DML 的并发，DDL+DDL 的并发也不允许。这是因为MySQL中没有DDL事务，不能保证DDL的原子性，当DDL和DML同时操作某一个对象，可能DDL修改后，DML将因为对象结构的改变而无法执行，继而回滚)
4. 不支持级联的外键约束：多主模型的组(所有节点都配置了`group_replication_single_primary_mode=OFF`)不支持多级外键依赖，特别是表上定义了级联的外键约束(CASCADING foreign key constraints)。这是因为多主模型下执行外键约束的级联操作可能会出现未检测到的冲突，从而导致组内成员间数据不一致。因此，我们推荐在使用多主模型时，在每个节点上都设置`group_replication_enforce_update_everywhere_checks=ON`以避免出现未检测到的冲突。在单主模型下没有这种问题，因为没有并发写操作，从而不可能会出现未被探测到的冲突。
5. 大事务可能会错误：如果一个事务非常大，导致GTID的内容非常多，以至于无法在 5 秒内通过网络传输完成，这时组成员间的通信将失败。要避免该问题，可以尽可能地限制事务的大小。例如，将`LOAD DATA INFILE`的文件切割为多个小块。
6. 多主模型可能出现死锁：在多主模型下，`SELECT ... FOR UPDATE`语句可能会导致死锁。这是因为组内成员之间不会共享锁资源(译注：share nothing)，因此这样的语句可能达不到预期的结果。

**MySQL组复制系列文章：**

MySQL的组复制可以配置为**单主模型**和**多主模型**两种工作模式，它们都能保证MySQL的高可用。以下是两种工作模式的特性简介：

- 单主模型：从复制组中众多个MySQL节点中**自动选举**一个master节点，只有master节点可以写，其他节点自动设置为read only。当master节点故障时，会自动选举一个新的master节点，选举成功后，它将设置为可写，其他slave将指向这个新的master。
- 多主模型：复制组中的任何一个节点都可以写，因此没有master和slave的概念，只要突然故障的节点数量不太多，这个多主模型就能继续可用。

资料整理
火龙果软件
荐

虽然多主模型的特性很诱人，但缺点是要配置和维护这种模式，必须要深入理解组复制的理论，更重要的是，多主模型限制较多，其一致性、安全性还需要多做测试。

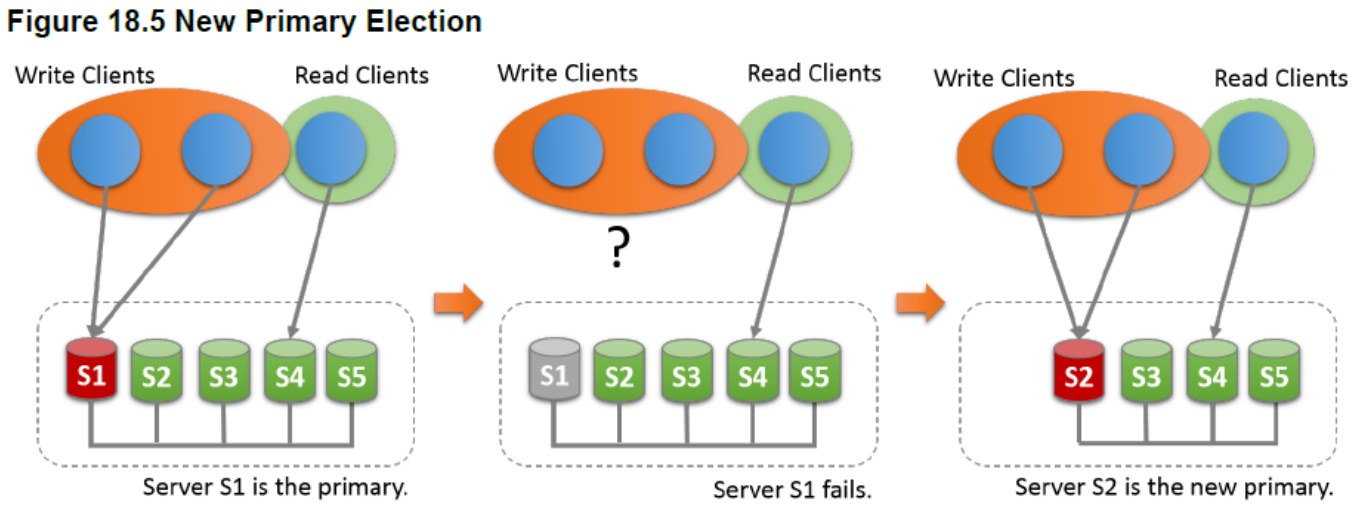
而使用单主模型的组复制就简单的太多了，唯一需要知道的就是它会自动选举master节点这个特性，因为它的维护一切都是自动进行的，甚至对于管理人员来说，完全可以不用去了解组复制的理论。

虽然单主模型比多主模型的性能要差，但它没有数据不一致的危险，加上限制少，配置简单，基本上没有额外的学习成本，所以多数情况下都是配置单主模型的组复制，即使是PXC和MariaDB也如此。

## 1.单主模型组复制的理论基础

虽说组复制的单主模型很简单，但有必要了解一点和单主模型有关的理论，尽管不了解也没什么问题，毕竟一切都是自动的。

如下图，master节点为s1，其余为slave节点。




组复制一切正常时，所有的写操作都路由到s1节点上，所有的读操作都路由到s2、s3、s4或s5上。当s1节点故障后，组复制自动选举新的master节点。假如选举s2为新master成功后，s3、s4和s5将指向s2，写操作将路由到s2节点上。

至于如何改变客户端的路由目标，这不是组复制应该考虑的事情，而是客户端应用程序应该考虑的事情。实际上，更好的方式是使用中间件来做数据库的路由，比如MySQL Router、ProxySQL、amoeba、cobar、mycat。

## 1.1 如何加入新节点

上面一直说，单主模型是自动选举主节点的，那么如何选举？

首先，在第一个MySQL节点s1启动时，一般会将其设置为组的引导节点，所谓引导就是在启动组复制功能时去创建一个复制组。当然，这并非强制要求，也可以设置第二个启动节点作为组的引导节点。因为组内没有其他节点，所以这第一个节点会直接选为master节点。

然后，如果有第二个节点要加入组时，新节点需要征得组的同意，因为目前只有一个节点，所以只需s1节点同意即可。新节点在加入组时，首先会联系s1，与s1建立**异步复制**的通道，并从s1节点处获取s2上目前缺失的数据，等到s1和s2节点上的数据同步后，s2节点就会真正成为组中的新成员。当然，实际过程要比这里复杂一些，本文不会过多讨论。

如果还有新节点(比如s3节点)继续加入组，s3将从s1或s2中选一个，并与之建立异步复制的通道，然后获取缺失的数据，同步结束后，如果s1和s2都同意s3加入，那么s3将会组中的新成员。其余节点加入组也依次类推。

有两点需要注意：

1. **新节点加入组时，如何选择联系对象？**

   上面说加入第二个节点s2时会联系s1，加入s3时会联系s1、s2中的任意一个。实际上，新节点加入组时联系的对象，称为donor，意为数据供应者。新节点会和选中的donor建立异步复制通道，并从donor处获取缺失的数据。

资料整理 火龙果软件 荐

在配置组复制时，需要指定种子节点列表。当新节点加入组时，只会联系种子节点，也即是说，只有种子节点列表中的节点才有机会成为donor，没有在种子节点列表中的节点不会被新节点选中。但**建议，将组中所有节点都加入到种子列表中**。

当联系第一个donor失败后，会向后联系第二个donor，再失败将联系第三个donor，如果所有种子节点都联系失败，在等待一段时间后再次从头开始联系第一个donor。依此类推，直到加组失败报错。

2. **新节点加入组时，需要征得哪些节点的同意？**

实际上，新节点加组涉及到组的决策：是否允许它加组。在组复制中，所有的决策都需要组中大多数节点达成一致，也即是达到法定票数。所谓大多数节点，指的是`N/2+1`(N是组中目前节点总数)，例如目前组中有5个节点，则需要3个节点才能达到大多数的要求。

## 1.2 如何选举新的master

当主节点s1故障后，组复制的故障探测机制就能发现这个问题并报告给组中其他成员，组中各成员根据收集到的其他成员信息，会比较各成员的权重值(由变量`group_replication_member_weigth`控制)，权重值最高的优先成为新的Master。如果有多个节点具有相同的最高权重值，会按字典顺序比较它们的server_uuid值，最小的(升序排序，最小值在最前面)优先成为新的master。

但需要注意，变量`group_replication_member_weigth`是从MySQL 5.7.20开始提供的，在MySQL 5.7.17到5.7.19之间没有该变量。此时将根据它们的server_uuid值进行排序选举。具体的规则可自行测试。

## 1.3 最多允许多少个节点故障

MySQL组复制使用Paxos分布式算法来提供节点间的分布式协调。正因如此，它要求组中大多数节点在线才能达到法定票数，从而对一个决策做出一致的决定。

大多数指的是`N/2+1`(N是组中目前节点总数)，例如目前组中有5个节点，则需要3个节点才能达到大多数的要求。所以，允许出现故障的节点数量如下图：

| 组的大小 | 大多数数量 | 故障容忍数量 |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 2 | 0 |
| 3 | 2 | 1 |
| 4 | 3 | 1 |
| 5 | 3 | 2 |
| 6 | 4 | 2 |
| 7 | 4 | 3 |

## 1.5 更多组复制的理论

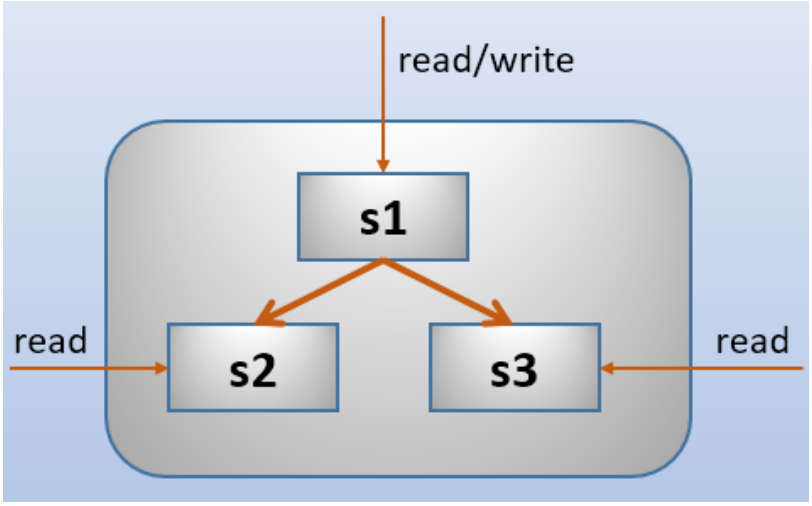
若想了解更多组复制的理论以及组复制中每一个过程的细节，请参考我另一篇文章MySQL组复制理论透彻分析，或者阅读我对MySQL官方手册关于组复制的翻译。

## 2.配置单主模型

本文配置3个节点的单主模型组复制。配置很简单，基本上就是在常规复制选项的基础上多了几个选项、多了几步操作。

拓扑图如下：

资料整理
火龙果软件
荐




具体环境细节如下：

| 节点名称 | MySQL版本 | 客户端接口(eth0) | 组内通信接口(eth0) | 数据状态 |
|---|---|---|---|---|
| s1 | MySQL 5.7.22 | 192.168.100.21 | 192.168.100.21 | 全新实例 |
| s2 | MySQL 5.7.22 | 192.168.100.22 | 192.168.100.22 | 全新实例 |
| s3 | MySQL 5.7.22 | 192.168.100.23 | 192.168.100.23 | 全新实例 |

发现了每个节点都给了两个接口吗？我这里配置它们都使用同一个接口eth0。其中：

1. 客户端接口是mysqld向外提供数据库服务的，对应端口是3306，例如php程序连接MySQL执行一个查询语句时就使用该地址。
2. 组内节点通信接口用于组内各节点消息传递，组内两两节点建立一条消息传递的TCP连接。所以，3个节点需要建立的组内通信连接为：`s1<-->s2、s1<-->s3、s2<-->s3`。

**请确保这3个节点的主机名不同，且能正确解析为客户端接口的地址**(别搞错地址了)，因为在连接donor进行数据恢复的时候，是通过主机名进行解析的。所以，所有节点都要先配置好不同的主机名，并修改/etc/hosts文件。对于克隆出来的实验主机，这一步骤很关键。以centos 7为例：

```
# s1上:
hostnamectl set-hostname --static s1.longshuai.com
hostnamectl -H root@192.168.100.22 set-hostname s2.longshuai.com
hostnamectl -H root@192.168.100.23 set-hostname s3.longshuai.com

# 写/etc/hosts
# s1上:
cat >>/etc/hosts<<eof
    192.168.100.21 s1.longshuai.com
```

资料整理 火龙果软件 荐

```
    192.168.100.22 s2.longshuai.com
    192.168.100.23 s3.longshuai.com
eof
scp /etc/hosts 192.168.100.22:/etc
scp /etc/hosts 192.168.100.23:/etc
```

## 2.1 配置组内第一个节点s1

**1.先提供配置文件。**

```
[mysqld]
datadir=/data
socket=/data/mysql.sock

server-id=100                       # 必须
gtid_mode=on                        # 必须
enforce_gtid_consistency=on         # 必须
log-bin=/data/master-bin            # 必须
binlog_format=row                   # 必须
binlog_checksum=none                # 必须
master_info_repository=TABLE        # 必须
relay_log_info_repository=TABLE     # 必须
relay_log=/data/relay-log           # 必须，如果不给，将采用默认值
log_slave_updates=ON                # 必须
sync-binlog=1                       # 建议
log-error=/data/error.log
pid-file=/data/mysqld.pid

transaction_write_set_extraction=XXHASH64         # 必须
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"  # 必须
loose-group_replication_start_on_boot=off        # 建议设置为OFF
```

资料整理 火龙果软件 荐

```
loose-group_replication_member_weigth = 40   # 非必需，mysql 5.7.20才开始支持该选项
loose-group_replication_local_address="192.168.100.21:20001"   # 必须，下一行也必须
loose-group_replication_group_seeds="192.168.100.21:20001,192.168.100.22:20002"
```

想要使用组复制，要求还是挺多的。分析一下上面的配置选项：

- (1).因为组复制基于GTID，所以必须开启gtid_mode和enforce_gtid_consistency。
- (2).组复制必须开启二进制日志，且必须设置为行格式的二进制日志，这样才能从日志记录中收集信息且保证数据一致性。所以设置log_bin和binlog_format。
- (3).由于MySQL对复制事件校验的设计缺陷，组复制不能对他们校验，所以设置binlog_checksum=none。
- (4).组复制要将master和relay log的元数据写入到mysql.slave_master_info和mysql.slave_relay_log_info中。
- (5).组中的每个节点都保留了完整的数据副本，它是share-nothing的模式。所以所有节点上都必须开启log_slave_updates，这样新节点随便选哪个作为donor都可以进行异步复制。
- (6).sync_binlog是为了保证每次事务提交都立刻将binlog刷盘，保证出现故障也不丢失日志。
- (7).最后的6行是组复制插件的配置。以`loose_`开头表示即使启动组复制插件，MySQL也继续正常允许下去。这个前缀是可选的。
- (8).倒数第6行表示写集合以`XXHASH64`的算法进行hash。**所谓写集，是对事务中所修改的行进行的唯一标识，在后续检测并发事务之间是否修改同一行冲突时使用**。它基于主键生成，所以使用组复制，表中必须要有主键。
- (9).倒数第5行表示这个复制组的名称。它必须是一个有效的UUID值。嫌可以直接和上面一样全写字母a。在Linux下，可以使用`uuidgen`工具来生成UUID值。

```
1
2
[root@xuexi ~]# uuidgen
09c38ef2-7d81-463e-bdb4-9459b2c0e49b
```

- (10).倒数第4行表示组复制功能不随MySQL实例启动而启动。虽然，可以将组复制插件和启动组复制功能的选项写在配置文件里，但强烈建议不要如此，而是每次手动去配置。
- (11).倒数第3行表示该节点在组中的权重为40。权重越高，自动选举为primary节点的优先级就越高。
- (12).倒数第2行表示本机上用于**组内各节点之间通信的地址和端口**。
- (13).最后一行，设置本组的种子节点。种子节点的意义在前文已经解释过了。

现在配置文件已经提供。可以启动mysql实例了。

```
[root@xuexi ~]# systemctl start mysqld
```

**2.创建复制用户。**

连上s1节点。创建用于复制的用户。我这里创建的用户为`repl`，密码为`P@ssword1!`。

```
1
2
mysql> create user repl@'192.168.100.%' identified by 'P@ssword1!';
mysql> grant replication slave on *.* to repl@'192.168.100.%';
```

**3.配置节点加组时的通道。这是组复制的一个关键。**

在新节点加入组时，首先要选择donor。新节点和donor之间的异步复制就是通过一个名为`group_replication_recovery`的通道(通道名固定，不可使用自定义通道)进行数据恢复的，经过数据恢复后，新节点填充了它缺失的那部分数据，这样就和组内其他节点的数据保持了同步。

这个恢复过程比较复杂，它是一种分布式恢复。本文不介绍这个，在MySQL组复制理论透彻分析中，我对此做了详细的说明。

执行`change master to`语句设置恢复通道。

资料整理 火龙果软件 荐

```
mysql> change master to
            master_user='repl',
            master_password='P@ssword1!'
            for channel 'group_replication_recovery';
```

**这里的用户名、密码和通道在组复制中有一个专门的术语：通道凭据(channel credentials)。**通道凭据是连接donor的关键。

当执行完上面的语句后，就生成了一个**该通道的relay log文件**(注意称呼：该通道的relay log，后面还有另一个通道的relay log)。如下，其中前缀"relay-log"是配置文件中"relay_log"选项配置的值。

```
[root@xuexi ~]# ls -l /data/*group*
/data/relay-log-group_replication_recovery.000001
/data/relay-log-group_replication_recovery.index
```

`group_replication_recovery`通道的relay log用于新节点加入组时，**当新节点联系上donor后，会从donor处以异步复制的方式将其binlog复制到这个通道的relay log中，新节点将从这个recovery通道的relay log中恢复数据。**

前面配置文件中已经指定了master和relay log的元数据信息要记录到表中，所以这里可以先查看下关于relay log的元数据。

```
mysql> select * from mysql.slave_relay_log_info\G
*************************** 1. row ***************************
  Number_of_lines: 7
   Relay_log_name: /data/relay-log-group_replication_recovery.000001
    Relay_log_pos: 4
  Master_log_name:
   Master_log_pos: 0
        Sql_delay: 0
Number_of_workers: 0
               Id: 1
     Channel_name: group_replication_recovery
```

如果要查看连接master的元数据信息，则查询`mysql.slave_master_info`表。不过现在没必要查，因为啥都还没做呢。

**4.安装组复制插件，并启动组复制功能。**

一切就绪后，可以开启mysql实例的组复制功能了。

```
mysql> install plugin group_replication soname 'group_replication.so';
```

资料整理
火龙果软件
荐

然后开启组复制功能。

```
mysql> set @@global.group_replication_bootstrap_group=on;
mysql> start group_replication;
mysql> set @@global.group_replication_bootstrap_group=off;
```

这里的过程很重要，需要引起注意。在开启组复制之前，设置全局变量`group_replication_bootstrap_group`为on，这表示稍后启动的组复制功能将引导组，也就是创建组并配置组，这些都是自动的。配置引导变量为ON后，再开启组复制插件功能，也就是启动组复制。最后将引导变量设回OFF，之所以要设置回OFF，是为了避免下次重启组复制插件功能时再次引导创建一个组，这样会存在两个名称相同实际却不相同的组。

**这几个过程不适合放进配置文件中，强烈建议手动执行它们的。否则下次重启mysql实例时，会自动重新引导创建一个组。同理，除了第一个节点，其他节点启动组复制功能时，不应该引导组，所以只需执行其中的start语句，千万不能开启`group_replication_bootstrap_group`变量引导组。**

这里的几个过程，应该形成一个习惯，在启动第一个节点时，这3条语句同时执行，在启动其他节点时，只执行start语句。

当启动组复制功能后，将生成另一个通道`group_replication_applier`的相关文件。

```
[root@xuexi ~]# ls -l /data/*group*
/data/relay-log-group_replication_applier.000001
/data/relay-log-group_replication_applier.000002
/data/relay-log-group_replication_applier.index
/data/relay-log-group_replication_recovery.000001
/data/relay-log-group_replication_recovery.index
```

是否还记得刚才用于恢复的通道`group_replication_recovery`？这个applier通道是干什么的？**常规复制有两个复制线程：io线程和sql线程，在组复制中，不再称之为io_thread和sql_thread，取而代之的是receiver、certifier和applier。**这里简单介绍一下它们的作用：

- receiver的作用类似于io线程，用于接收组内个节点之间传播的消息和事务。也用于接收外界新发起的事务。
- applier的作用类似于sql线程，用于应用relay log中的记录。不过，组复制的relay log不再是relay log，而是这里的组复制relay log：`relay-log-group_replication_applier.00000N`。
- certifier的作用在receiver接收到消息后，验证是否有并发事务存在冲突问题。冲突检测通过后，这条消息就会写入到组复制的relay log中，等待applier去应用。

并不是说组复制中没有io线程和sql线程，而是称呼改变了，receiver和applier实际上就是io_therad和sql_thread。

**5.验证组中节点并测试插入不满足组复制要求的数据。**

至此，这个节点的组复制已经配置完成了。现在需要查看这个节点是否成功加入到组中，**成功加入组的标志是被设置为"ONLINE"。**只要没有设置为ONLINE，就表示组中的这个节点是故障的。

查看的方式是通过查询`performance_schema`架构下的`replication_group_members`表。在这个架构下，有几张对于维护组复制来说非常重要的表，这里的`replication_group_members`是其中一张。关于其他的表，我会在有需要的地方或者其他文章中解释。

资料整理
火龙果软件
荐

```
mysql> select * from performance_schema.replication_group_members;
+---------------------------+--------------------------------------+---------------------+-------------+-----------
----+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST         | MEMBER_PORT |
MEMBER_STATE |
+---------------------------+--------------------------------------+---------------------+-------------+-----------
----+
| group_replication_applier | a659234f-6aea-11e8-a361-000c29ed4cf4 | xuexi.longshuai.com |        3306 | ONLINE
|
+---------------------------+--------------------------------------+---------------------+-------------+-----------
----+
```

如果不方便观看，换一种显示方式：

```
mysql> select * from performance_schema.replication_group_members\G
*************************** 1. row ***************************
CHANNEL_NAME: group_replication_applier
   MEMBER_ID: a659234f-6aea-11e8-a361-000c29ed4cf4
 MEMBER_HOST: xuexi.longshuai.com
 MEMBER_PORT: 3306
MEMBER_STATE: ONLINE
```

请注意这里的每一行，包括member_host，它是对外连接的地址，所以应该设置它的DNS解析为提供MySQL数据库服务的接口地址。这很重要，如果你不想去修改DNS解析，可以在启动组复制之前，设置report_host变量为对外的IP地址，或者将其写入到配置文件中。

现在，组中的这个节点已经是ONLINE了，表示可以对外提供组复制服务了。

稍后，将向组中加入第二个节点s2和第三个节点s3，但在加入新节点之前，先向s1节点写入一些数据，顺便测试一下开启组复制后，必须使用InnoDB、表中必须有主键的限制。

下面创建4个表：t1和t4是InnoDB表，t3和t4具有主键。

```
create table t1(id int);
create table t2(id int)engine=myisam;
create table t3(id int primary key)engine=myisam;
create table t4(id int primary key);
```

资料整理
火龙果软件
荐

虽说组复制对这些有限制，但是创建时是不会报错的。

向这4张表中插入数据：

```
1
2
3
4
insert into t1 values(1);
insert into t2 values(1);
insert into t3 values(1);
insert into t4 values(1);
```

会发现只有t4能插入成功，t1、t2、t3都插入失败，报错信息如下：

```
ERROR 3098 (HY000): The table does not comply with the requirements by an external plugin.
```

意思是该表不遵从外部插件(即组复制插件)的要求。

最后，查看下二进制日志中的事件。为了排版，我将显示结果中的日志名称列去掉了。

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
mysql> SHOW BINLOG EVENTS in 'master-bin.000004';
+------+----------------+-----------+-------------+-------------------------------------------------------------------
----+
| Pos  | Event_type     | Server_id | End_log_pos | Info
|
```

资料整理
火龙果软件
荐

```
+------+----------------+-----------+-------------+-------------------------------------------------------------------
----+
|    4 | Format_desc    |       100 |         123 | Server ver: 5.7.22-log, Binlog ver: 4
|
|  123 | Previous_gtids |       100 |         150 |
|
|  150 | Gtid           |       100 |         211 | SET @@SESSION.GTID_NEXT= 'a659234f-6aea-11e8-a361-
000c29ed4cf4:1' |
|  211 | Query          |       100 |         399 | CREATE USER 'repl'@'192.168.100.%' IDENTIFIED WITH 'password'
|
|  399 | Gtid           |       100 |         460 | SET @@SESSION.GTID_NEXT= 'a659234f-6aea-11e8-a361-
000c29ed4cf4:2' |
|  460 | Query          |       100 |         599 | GRANT REPLICATION SLAVE ON *.* TO 'repl'@'192.168.100.%'
|
|  599 | Gtid           |       100 |         660 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:1' |
|  660 | Query          |       100 |         719 | BEGIN
|
|  719 | View_change    |       100 |         858 | view_id=15294216022242634:1
|
|  858 | Query          |       100 |         923 | COMMIT
|
|  923 | Gtid           |       100 |         984 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:2' |
|  984 | Query          |       100 |        1083 | create database gr_test
|
| 1083 | Gtid           |       100 |        1144 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:3' |
| 1144 | Query          |       100 |        1243 | use `gr_test`; create table t1(id int)
|
| 1243 | Gtid           |       100 |        1304 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:4' |
| 1304 | Query          |       100 |        1416 | use `gr_test`; create table t2(id int)engine=myisam
|
| 1416 | Gtid           |       100 |        1477 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:5' |
| 1477 | Query          |       100 |        1601 | use `gr_test`; create table t3(id int primary
key)engine=myisam   |
| 1601 | Gtid           |       100 |        1662 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:6' |
| 1662 | Query          |       100 |        1773 | use `gr_test`; create table t4(id int primary key)
|
| 1773 | Gtid           |       100 |        1834 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:7' |
| 1834 | Query          |       100 |        1905 | BEGIN
|
| 1905 | Table_map      |       100 |        1949 | table_id: 117 (gr_test.t4)
|
| 1949 | Write_rows     |       100 |        1985 | table_id: 117 flags: STMT_END_F
|
| 1985 | Xid            |       100 |        2012 | COMMIT /* xid=63 */
|
+------+----------------+-----------+-------------+-------------------------------------------------------------------
----+
```

除了正常的事务对应的事件，需要关注的三行是：

资料整理
火龙果软件
荐

```
BEGIN
View_change <----> view_id=15294216022242634:1
COMMIT
```

组复制中，每个决策都需要组中大多数节点达成一致，包括新节点加组、离组的决定。实际上，组复制中内置了一个组成员服务，这个服务负责组成员的配置以及动态捕获组中成员列表，这个成员列表成为**成员视图**。每个视图都有一个view id，view id的第一部分是创建组时随机生成的，只要组不停止，这部分就不改变，第二部分是从1开始的单调递增的数值。每当有成员加组、离组时，都会触发这个服务对组成员进行重新配置，每次组成员的重新配置，view id的第二部分都会单调递增地加1，表示这是新的成员视图，新的组成员视图需要得到组中大多数节点的同意，所以这个消息要在组中进行传播。如何传播？就是通过将视图更改的事件作为一个事务写进binlog中，然后在组中到处复制，这样每个节点都可收到视图变化的消息，并对此做出回应，同意之后再commit这个事务。如果足够细心，会发现这个事务的提交和下面插入数据的提交(`COMMIT /* xid=63 */`)方式不一样。如果不理解也没关系，这个理论并不影响组复制的使用。

再仔细一看，还可以发现MySQL中的DDL语句是没有事务的。所以，绝不允许不同节点上对同一个对象并发执行"DDL+DML"和"DDL+DDL"，冲突检测机制会探测到这样的冲突。

## 2.2 向组中添加新节点

当组中已有第一个节点后，需要做的是向组中添加新的节点。这里以添加s2和s3为例。

### 2.2.1 添加新节点前要做什么

前面多次提到，新节点在加入组的时候，会先选择一个donor，并通过异步复制的方式从这个donor处获取缺失的数据，以便在成功加入组的时候它的数据和组中已有的节点是完全同步的，这样才能向外界客户端提供查询。

这里的重点在于异步复制，既然是复制，它就需要复制binlog，并通过应用binlog中的记录来写数据。如果在加入组之前，组中的数据量已经非常大，那么这个异步复制的过程会很慢，而且还会影响donor的性能，毕竟它要传输大量数据出去。

本来加入新节点的目的就是对组复制进行扩展，提高它的均衡能力，现在因为异步复制慢，反而导致性能稍有下降，新节点短期内还无法上线向外提供服务。这有点背离原本的目标。

再者，如果组中的节点purge过日志，那么新节点将无法从donor上获取完整的数据。这时新节点上的恢复过程会让它重新选择下一个donor。但很可能还是会失败，因为实际环境中，既然purge了某节点上的一段日志，很可能同时会去所有节点上也Purge。(注意，purge不是事件，不会写入到binlog中，所以不会复制到其它节点上，换句话说，某节点Purge后，那么它的binlog和其它节点的binlog是不一致的)。

所以，**在新节点加入组之前，应该先通过备份恢复的方式，从组中某节点上备份目前的数据到新节点上，然后再让新节点去加组**，这样加组的过程将非常快，且能保证不会因为purge的原因而加组失败。至于如何备份恢复，参见我的另一篇文章：[将slave恢复到master指定的坐标]。

我这里做实验的环境，所有节点都是刚安装好的全新实例，数据量小，也没purge过日志，所以直接加入到组中就可以。

### 2.2.2 添加第二个节点

仍然先是提供配置文件。配置文件和第一个节点基本相同，除了几个需要保持唯一性的选项。

配置文件内容如下：

资料整理
火龙果软件
荐

```
[mysqld]
datadir=/data
socket=/data/mysql.sock

server-id=110                       # 必须，每个节点都不能相同
gtid_mode=on                        # 必须
enforce_gtid_consistency=on         # 必须
log-bin=/data/master-bin            # 必须
binlog_format=row                   # 必须
binlog_checksum=none                # 必须
master_info_repository=TABLE        # 必须
relay_log_info_repository=TABLE     # 必须
relay_log=/data/relay-log           # 必须，如果不给，将采用默认值
log_slave_updates=ON                # 必须
sync-binlog=1                       # 建议
log-error=/data/error.log
pid-file=/data/mysqld.pid

transaction_write_set_extraction=XXHASH64         # 必须
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"  # 必须
loose-group_replication_start_on_boot=off        # 建议设置为OFF
loose-group_replication_member_weigth = 20   # 非必需，mysql 5.7.20才开始支持该选项
loose-group_replication_local_address="192.168.100.22:20002"   # 必须，下一行也必须
loose-group_replication_group_seeds="192.168.100.21:20001,192.168.100.22:20002"
```

这里和s1的配置文件相比，只修改了`server-id`和`group_replication_local_address`以及权重值。

然后执行`change master to`，选择一个donor(此刻只有s1能选)，并和donor建立通道连接。

```
mysql> change master to
            master_user='repl',
            master_password='P@ssword1!'
            for channel 'group_replication_recovery';
```

资料整理 火龙果软件 荐

这时，就已经选择好donor，并和donor建立通道连接了。如果去s1上查看，可以看到这个通道的连接。下面的查询结果中，第二行就是和s2建立的连接，通道为`group_replication_recovery`。

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
```

资料整理
火龙果软件
荐

```
54
mysql> select * from mysql.slave_master_info\G
*************************** 1. row ***************************
       Number_of_lines: 25
       Master_log_name:
        Master_log_pos: 4
                  Host: <NULL>
             User_name:
         User_password:
                  Port: 0
         Connect_retry: 60
           Enabled_ssl: 0
                Ssl_ca:
            Ssl_capath:
              Ssl_cert:
            Ssl_cipher:
               Ssl_key:
Ssl_verify_server_cert: 0
             Heartbeat: 30
                  Bind:
    Ignored_server_ids: 0
                  Uuid:
           Retry_count: 86400
               Ssl_crl:
           Ssl_crlpath:
 Enabled_auto_position: 1
          Channel_name: group_replication_applier
           Tls_version:
*************************** 2. row ***************************
       Number_of_lines: 25
       Master_log_name:
        Master_log_pos: 4
                  Host:
             User_name: repl
         User_password: P@ssword1!
                  Port: 3306
         Connect_retry: 60
           Enabled_ssl: 0
                Ssl_ca:
            Ssl_capath:
              Ssl_cert:
            Ssl_cipher:
               Ssl_key:
Ssl_verify_server_cert: 0
             Heartbeat: 0
                  Bind:
    Ignored_server_ids: 0
                  Uuid:
           Retry_count: 86400
               Ssl_crl:
           Ssl_crlpath:
 Enabled_auto_position: 0
          Channel_name: group_replication_recovery
           Tls_version:
2 rows in set (0.00 sec)
```

资料整理
火龙果软件
荐

然后回到s2节点上，安装组复制插件，并开启组复制功能。

```
mysql> install plugin group_replication soname 'group_replication.so';
mysql> start group_replication;
```

组复制启动成功后，查看是否处于online状态。(请无视我这里的Member_host字段，这是我设置了report_host变量的结果)

```
mysql> select * from performance_schema.replication_group_members\G
*************************** 1. row ***************************
CHANNEL_NAME: group_replication_applier
   MEMBER_ID: a5165443-6aec-11e8-a8f6-000c29827955
 MEMBER_HOST: 192.168.100.22
 MEMBER_PORT: 3306
MEMBER_STATE: ONLINE
*************************** 2. row ***************************
CHANNEL_NAME: group_replication_applier
   MEMBER_ID: a659234f-6aea-11e8-a361-000c29ed4cf4
 MEMBER_HOST: xuexi.longshuai.com
 MEMBER_PORT: 3306
MEMBER_STATE: ONLINE
```

再查看数据是否已经同步到s2节点。其实显示了ONLINE，就一定已经同步。

资料整理
火龙果软件
荐

```
mysql> show tables from gr_test;
+-------------------+
| Tables_in_gr_test |
+-------------------+
| t1                |
| t2                |
| t3                |
| t4                |
+-------------------+
4 rows in set (0.00 sec)

mysql> select * from gr_test.t4;
+----+
| id |
+----+
|  1 |
+----+
```

查看binlog事件。会发现内容已经复制，且view id又发生了一次变化。

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
mysql> show binlog events in 'master-bin.000002';
+------+----------------+-----------+-------------+-------------------------------------------------------------------
```

资料整理
火龙果软件
荐

```
----+
| Pos  | Event_type     | Server_id | End_log_pos | Info
|
+------+----------------+-----------+-------------+-------------------------------------------------------------
----+
|    4 | Format_desc    |       110 |         123 | Server ver: 5.7.22-log, Binlog ver: 4
|
|  123 | Previous_gtids |       110 |         150 |
|
|  150 | Gtid           |       100 |         211 | SET @@SESSION.GTID_NEXT= 'a659234f-6aea-11e8-a361-
000c29ed4cf4:1' |
|  211 | Query          |       100 |         399 | CREATE USER 'repl'@'192.168.100.%' IDENTIFIED WITH 'password'
|
|  399 | Gtid           |       100 |         460 | SET @@SESSION.GTID_NEXT= 'a659234f-6aea-11e8-a361-
000c29ed4cf4:2' |
|  460 | Query          |       100 |         599 | GRANT REPLICATION SLAVE ON *.* TO 'repl'@'192.168.100.%'
|
|  599 | Gtid           |       100 |         660 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:1' |
|  660 | Query          |       100 |         719 | BEGIN
|
|  719 | View_change    |       100 |         858 | view_id=15294216022242634:1
|
|  858 | Query          |       100 |         923 | COMMIT
|
|  923 | Gtid           |       100 |         984 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:2' |
|  984 | Query          |       100 |        1083 | create database gr_test
|
| 1083 | Gtid           |       100 |        1144 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:3' |
| 1144 | Query          |       100 |        1243 | use `gr_test`; create table t1(id int)
|
| 1243 | Gtid           |       100 |        1304 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:4' |
| 1304 | Query          |       100 |        1416 | use `gr_test`; create table t2(id int)engine=myisam
|
| 1416 | Gtid           |       100 |        1477 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:5' |
| 1477 | Query          |       100 |        1601 | use `gr_test`; create table t3(id int primary
key)engine=myisam   |
| 1601 | Gtid           |       100 |        1662 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:6' |
| 1662 | Query          |       100 |        1773 | use `gr_test`; create table t4(id int primary key)
|
| 1773 | Gtid           |       100 |        1834 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa:7' |
| 1834 | Query          |       100 |        1893 | BEGIN
|
| 1893 | Table_map      |       100 |        1937 | table_id: 112 (gr_test.t4)
|
| 1937 | Write_rows     |       100 |        1973 | table_id: 112 flags: STMT_END_F
|
| 1973 | Xid            |       100 |        2000 | COMMIT /* xid=31 */
|
| 2000 | Gtid           |       100 |        2061 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-
```

资料整理
火龙果软件
荐

```
aaaaaaaaaaaa:8' |
| 2061 | Query          |       100 |        2120 | BEGIN
|
| 2120 | View_change    |       100 |        2299 | view_id=15294216022242634:2
|
| 2299 | Query          |       100 |        2364 | COMMIT
|
+------+----------------+-----------+-------------+-----------------------------------------------------------------
----+
```

### 2.2.3 添加第三个节点

和加入s2节点几乎一致。所以这里做个步骤的简单总结：

**1.配置主机名和DNS解析。**

**2.提供配置文件，并启动MySQL实例。**

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
datadir=/data
socket=/data/mysql.sock

server-id=120
gtid_mode=on
enforce_gtid_consistency=on
log-bin=/data/master-bin
binlog_format=row
binlog_checksum=none
master_info_repository=TABLE
relay_log_info_repository=TABLE
relay_log=/data/relay-log
log_slave_updates=ON
sync-binlog=1
log-error=/data/error.log
```

资料整理
火龙果软件
荐

```
pid-file=/data/mysqld.pid

transaction_write_set_extraction=XXHASH64
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
loose-group_replication_start_on_boot=off
loose-group_replication_member_weigth = 30
loose-group_replication_local_address="192.168.100.23:20003"
loose-group_replication_group_seeds="192.168.100.21:20001,192.168.100.22:20002"
```

**3.连上新实例，设置恢复通道的凭据。**

```
1
2
3
4
change master to
            master_user='repl',
            master_password='P@ssword1!'
            for channel 'group_replication_recovery';
```

**4.安装组复制插件，并启动组复制功能。**

```
1
2
install plugin group_replication soname 'group_replication.so';
start group_replication;
```

**5.查看新节点是否已经处于ONLINE。**

```
select * from performance_schema.replication_group_members\G
```

# 3.加组失败

## 3.1 新节点一直处于recoveing？

当加入一个新节点时，一切配置都正确，但是新节点死活就是不同步数据，随便执行一个语句都卡半天，查看`performance_schema.replication_group_members`表时，还发现这个新节点一直处于recovering装态。

这时，请查看新节点的错误日志。以下是我截取出来的一行。

```
1
2
[root@xuexi ~]# tail /data/error.log
2018-06-19T17:41:22.314085Z 10 [ERROR] Plugin group_replication reported: 'There was an error when connecting to the donor server. Please check that group_replication_recovery channel credentials and all MEMBER_HOST column values of performance_schema.replication_group_members table are correct and DNS resolvable.'
```

很显然，连接donor的时候出错，让我们检测通道凭据，并且查看member_host字段的主机名是否正确解析。一切正确配置的情况下，通道凭据是没错的，错就错在member_host的主机名。

当和donor建立通道连接时，首先会通过member_host字段的主机名去解析donor的地址。这个主机名默认采取的是操作系统默认的主机名，而非ip地址。所以，必须设置DNS解析，或者/etc/hosts文件，将member_host对应的主机名解析为donor的ip地址。

我这里之所以显示错误，是因为我在测试环境下，所有节点的主机名都相同：xuexi.longshuai.com。所以新节点会将这个主机名解析到本机。

资料整理
火龙果软件
荐

## 3.2 新节点包含了额外的gtid事务？

如果新节点中包含了额外的数据，例如，新节点上多了一个用户，创建这个用户是会产生gtid事务的，当这个节点要加入到组时会报错。以下是error.log中的内容：

```
1
2
3
4
5
2018-06-24T12:56:29.300453Z 0 [ERROR] Plugin group_replication reported: 'This member has more executed
transactions than those present in the gro
up. Local transactions: 48f1d8aa-7798-11e8-bf9a-000c29296408:1-2 > Group transactions: 481024ff-7798-11e8-89da-
000c29ff1054:1-4,
bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb:1-6'
2018-06-24T12:56:29.300536Z 0 [ERROR] Plugin group_replication reported: 'The member contains transactions not
present in the group. The member wi
ll now exit the group.'
```

错误已经指明了是新节点的事务和组中的事务不匹配，有多余的事务，从而导致加组失败。

如果你已经明确这些多余的事务造成的数据不会影响到组中的节点，正如多了一个可能永远也用不上的用户，或者多了几个和组复制完全无关的数据库。

这时，可以将这些无关紧要的gtid给删掉，但是想删除这些gtid还真没那么容易。purge日志不行，停掉MySQL后删日志文件也不行，把binlog关掉再打开也不行。它们都会把以前的事务记录到Previous_gtid中。真正行之有效的方法是将全局变量executed_gtid设置为空。方法为：

```
mysql> reset master;
```

然后，再去加组。

# 4.组复制维护：停止、重启组复制功能

操作组复制的语句只有两个。

```
1
2
start group_replication;
stop group_replication;
```

但是，和组复制相关的变量却有好几个。

当要停止组中的某个成员中的组复制功能时，需要在那个节点上执行`stop group_replication`语句。但一定要注意，在执行这个语句之前，必须要保证这个节点不会向外提供MySQL服务，否则有可能会有新数据写入(例如主节点停止时)，或者读取到过期数据。

所以，要安全地**重启整个组**，最佳方法是先停止所有非主节点的MySQL实例(不仅是停止组复制功能)，然后停止主节点的MySQL实例，再先重启主节点，在这个节点上引导组，并启动它的组复制功能。最后再将各slave节点加入组。

如果只是想**停止某单个节点**，如果这个节点是主节点，那么停止整个MySQL实例，如果是slave节点，那么只需停止它的组复制功能即可。当它们需要再次加组时，只需执行`start group_replication`语句。

那么，如何知道哪个节点是主节点？

# 5.查找复制组中的主节点

资料整理
火龙果软件
荐

只有单主模型的组复制才需要查找主节点，多主模型没有master/slave的概念，所以无需查找。

```
1
2
3
4
5
6
7
8
mysql> SELECT VARIABLE_VALUE FROM performance_schema.global_status
       WHERE VARIABLE_NAME='group_replication_primary_member';
+--------------------------------------+
| VARIABLE_VALUE                       |
+--------------------------------------+
| a659234f-6aea-11e8-a361-000c29ed4cf4 |
+--------------------------------------+
1 row in set (0,00 sec)
```

或者：

```
mysql> SHOW STATUS LIKE 'group_replication_primary_member';
```

这样查找只是获取了主节点的uuid，可以表连接的方式获取主节点主机名。

```
1
2
3
4
5
6
7
8
9
10
select b.member_host the_master,a.variable_value master_uuid
    from performance_schema.global_status a
    join performance_schema.replication_group_members b
    on a.variable_value = b.member_id
    where variable_name='group_replication_primary_member';
+------------------+--------------------------------------+
| the_master       | master_uuid                          |
+------------------+--------------------------------------+
| s1.longshuai.com | a659234f-6aea-11e8-a361-000c29ed4cf4 |
+------------------+--------------------------------------+
```

## 6.测试：组复制的自动选举和容错

组复制中，有两种节点离组的情况：自愿离组、非自愿离组。

1. 自愿离组：执行`stop group_replication;`语句。
   - (1).执行该语句表示该节点自愿离组，它会触发视图自动配置，并将该视图更改操作复制到组内所有节点，直到大多数节点都同意新的视图配置，该节点才会离组。
   - (2).节点自愿离组时，不会丢失法定票数。所以无论多少个节点自愿离组，都不会出现"达不到大多数"的要求而阻塞组。
   - (3).举个例子，5个节点的组，自愿退出一个节点A后，这个组的大小为4。这个组认为节点A从来都没有出现过。

资料整理
火龙果软件
荐

2. 非自愿离组：除了上面自愿离组的情况，所有离组的情况都是非自愿离组。比如节点宕机，断网等等。
   - (1).节点非自愿离组时，故障探测机制会检测到这个问题，于是向组中报告这个问题。然后会触发组视图成员自动配置，需要大多数节点同意新视图。
   - (2).非自愿离组时，组的大小不会改变，无论多少个节点的组，节点非自愿退出后，组大小还是5，只不过这些离组的节点被标记为非ONLINE。但注意，组的视图配置会改变，因为离组的节点状态需要标记为非ONLINE。
   - (3).非自愿离组时，会丢失法定票数。所以，当非自愿离组节点数量过多时，导致组中剩余节点数量达不到大多数的要求，组就会被阻塞。
   - (4).举个例子，5节点的组，非自愿退出1个节点A后，这个组的大小还是5，但是节点A在新的视图中被标记为unreachable或其他状态。当继续非自愿退出2个节点后，组中只剩下2个ONLINE节点，这时达不到大多数的要求，组就会被阻塞。

目前，组中有3个节点：s1、s2和s3，其中s1是主节点。

现在将主节点直接关机或者断掉网卡，模拟非自愿离组。

```
# s1上:
shell> ifconfig eth0 down
```

然后查看s2上的错误日志。可以看到选举新主节点的过程。

```
[Warning] group_replication reported: 'Member with address s1.longshuai.com:3306 has become unreachable.'
[Note] group_replication reported: '[GCS] Removing members that have failed while processing new view.'
[Warning] group_replication reported: 'Members removed from the group: s1.longshuai.com:3306'
[Note] group_replication reported: 'Primary server with address s1.longshuai.com:3306 left the group. Electing new Primary.'
[Note] group_replication reported: 'A new primary with address s2.longshuai.com:3306 was elected, enabling conflict detection until the new primary applies all relay logs.'
[Note] group_replication reported: 'This server is working as primary member.'
[Note] group_replication reported: 'Group membership changed to s2.longshuai.com:3306, s3.longshuai.com:3306 on view 15294358712349771:4.'
```

这里将s2选为新的主节点，且告知成员视图中目前组中成员变为s2和s3。

可以测试下，是否能向新的主节点s2中插入数据。

```
# s2上:
mysql> insert into gr_test.t4 values(333);
```

如果再将s3停掉呢？还能继续写入数据吗？

```
# 在s3上:
shell> ifconfig eth0 down
```

资料整理 火龙果软件 荐

回到s2，插入数据看看：

```
# s2上:
mysql> insert into gr_test.t4 values(3333);
```

发现无法插入，一直阻塞。

查看下s2的错误日志：

```
[Warning] group_replication reported: 'Member with address s3.longshuai.com:3306 has become unreachable.'
[ERROR] group_replication reported: 'This server is not able to reach a majority of members in the group. This server will now block all updates. The server will remain blocked until contact with the majority is restored. It is possible to use group_replication_force_members to force a new group membership.'
```

已经说明了，s3移除后，组中的成员无法达到大多数的要求，所以将复制组给阻塞了。如果想要修复组，可以强制生成一个新的组成员视图。

如果这时候，将s1和s3的网卡启动，s1和s3还会加入到组中吗？以下是s2上的错误日志：

```
[Warning] group_replication reported: 'Member with address s3.longshuai.com:3306 is reachable again.'
[Warning] group_replication reported: 'The member has resumed contact with a majority of the members in the group. Regular operation is restored and transactions are unblocked.'
```

发现s3加入了，但s1未加入。为什么？因为s1节点上只是停掉了网卡，mysql实例以及组复制功能还在运行，而且它的角色还保持为主节点。这时候，s1和s2、s3已经出现了所谓的"网络分裂"，对于s2和s3来说，s1被隔离，对于s1来说，s2和s3被隔离。当s1的网卡恢复后，它仍然保留着自己的主节点运行，但因为它达不到大多数的要求，所以s1是被阻塞的，如果网卡长时间没有恢复，则s1会被标记为ERROR。

这种情况下的s1，要让它重新加入到组中，应该重启组复制，更安全的方法是重启mysql实例，因为组可能还没有标记为ERROR，这个组暂时还存在，它与s2、s3所属的组同名，可能会导致脑裂问题。

如果是自愿离组呢？可以测试下，无论自愿退出多少个节点，只要组中还有节点，组都不会被阻塞。

资料整理
火龙果软件
荐