主从报错GTID_MODE = ON cannot be set to ANONYMOUS

高鹏（八怪） 老叶茶馆 2025年05月28日 07:00 北京

一、相关报错

最近遇到这个错误，版本是MySQL 8.0.26，使用模拟的版本是MySQL 8.0.23，错误如下，

```
                   Last_Error: Error executing row event: '@@SESSION.GTID_NEXT cannot be
...
               Last_SQL_Error: Error executing row event: '@@SESSION.GTID_NEXT cannot be
```

出现这个错误后主从异常中断，查看日志数据库不久前crash过一次(OOM)，但是系统未重启。

二、相关概念

2.1 匿名事务

这个错误中有一个关键的提示词ANONYMOUS，也就是我们常说的匿名事务，而匿名事务实际上就是无GTID的事务，可能有2种情况，

- 5.7 后每个事务都有GTID event，如果GTID没有开启，则使用匿名GTID event。
- 5.6 中如果不开启GTID，是不包含GTID event的。

这两者都叫匿名事务，也就是在没有开启GTID功能的数据库上事务生成的binlog都叫匿名GTID事务。

如果数据库在GTID_MODE = ON 的情况下，是不允许执行匿名GTID 事务的。

如果要在从库执行匿名的GTID 事务，通常要修改GTID_MODE，在线打开GTID的时候通常会使用到这个参数，参数的含义如下：

- **OFF(0)**: Both new and replicated transactions must be anonymous.（生成的是匿名事务，从库也只能应用匿名事务）
- **OFF_PERMISSIVE(1)**: New transactions are anonymous. Replicated transactions can be either anonymous or GTID transactions.（生成的是匿名事务，从库可以应用匿名和GTID事务）
- **ON_PERMISSIVE(2)**: New transactions are GTID transactions. Replicated transactions can be either anonymous or GTID transactions.（生成的是GTID事务，从库可以应用匿名和

GTID事务）

- **ON(3)**: Both new and replicated transactions must be GTID transactions（生成的是GTID事务，从库也只能应用GTID事务）

在较新(MySQL 8.0.23和之后)的版本中，从库即便在 `GTID_MODE = ON` 的情况下也可以执行匿名的GTID事务，但是这个特性需要额外的配置，也就是给匿名的事务分配GTID的功能，可以在通道上指定 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 来完成，但是这个功能似乎很少使用。

2.2 事务的event

当然这部分经常提及的在MySQL 5.7之后每个常规的事务(非XA事务)都包含了 `GTID EVENT/QUERY EVENT/MAP EVENT/DML EVENT/XID EVENT` ，我们通常把 `GTID EVENT` 当做事务的第一个event，而 `XID EVENT` 当做事务提交的event。

2.3 主从的配置方式

主从的配置常见的有3种方式，

- 方式1：GTID 开启，且使用**GTID AUTO_POSTITION**
- 方式2：GTID 开启，但是使用**传统POSTITION**
- 方式3：GTID 关闭，只能使用**传统POSTITION**

而使用**GTID AUTO_POSTITION**和**传统POSTITION**有着非常大的区别，这点区别主要启动I/O线程的时候，初始化从哪个位点开始拉取主库的binlog。

其次这三种配置方式都很常见，第二种比较特殊，看起来GTID 都开启了，是不是可以直接切换为**GTID AUTO_POSTITION**方式，这通常是不行的，这点需要注意，因为这个时候从库的GTID 可能和主库是有出入的，这可能来自于搭建的方式或者过滤等设置，如果贸然改为**GTID AUTO_POSTITION**可能导致报错。

2.4 传统的POSTITION 下I/O 线程位点的存储

在**传统POSTITION**模式下，I/O线程拉取主库binlog的位点，存储在 `mysql.slave_master_info` 中(5.7也可能在文件中，依赖参数 `master_info_repository` 的配置，8.0默认为**TABLE**)，但是这个表默认的参数下并不是实时更新的，其由参数 `sync_master_info` 进行控制，为多少一个event写入relay log后进行一次同步，也就是这种情况下 `mysql.slave_master_info` 表中的信息和 `show slave status` 中的信息是不一致的 `Master_Log_File\Read_Master_Log_Pos` ，因为后者来自内存。

三、错误分析

我们的环境实际上是MySQL 8.0.26下的GTID 开启，但是使用**传统POSTITION**的主从，参数全部默认，也就是说 `master_info_repository` 为**TABLE**， `sync_master_info` 默认为10000，代表多少个event进行一次将内存postition信息写入到表中。

那么**在relay_log_recovery为OFF(默认值)**的情况下，从库异常重启的情况下就可能出现 `Master_Log_File\Read_Master_Log_Pos` 丢失的情况，这种几乎是必然的。

比如当前 `show slave status` 的 `Read_Master_Log_Pos` 是 100000 ，那么重启后这个值可能回退为90001(**在relay_log_recovery为OFF(默认值)**)，而I/O线程从90001这个点重新拉取binlog，那么可能出现2种情况，

- 如果90001 刚好是一个事务的开始，也就是GTID event，那么方式2会正常运行，因为GTID会过滤掉所有重复执行的GTID，而方式3会报错比如主键冲突等，因为没有GTID就没有过滤的功能，只能重新做一次。
- 如果90001 是事务的中间，这种情况方式2会爆出匿名事务的错误，也就是开头主从的报错，同时会检测到这种情况，可能出现如下日志，

```
An unexpected event sequence was detected by the IO thread while queuing the event receiv
```

而这个日志正是I/O线程检测到，relay log 新的拉取点并不是GTID EVENT而爆出来的，源码就在I/O线程的代码里面，下面我们看一下这部分的注释，截取部分如下，

```
Transaction boundary errors might happen mostly because of bad master
positioning in 'CHANGE MASTER TO' (or bad manipulation of master.info)
when GTID auto positioning is off.
The IO thread will keep working and queuing events regardless of the
transaction parser error, but we will throw another warning message to
log the relay log file and position of the parser error to help
forensics.
```

源码注释也提示了，在非**GTID AUTO_POSTITION**的主从中可能出现这种情况，但是没有直接停止拉取，而是选择了报警。

接着当SQL线程执行relay log的event的时候，发现并没有找到有效的GTID EVENT，因此报错 `Error executing row event: '@@SESSION.GTID_NEXT cannot be set to ANONYMOUS when @@GLOBAL.GTID_MODE = ON.'` 。

而实际上 `sync_relay_log` 也是默认值10000，代表relay log多少个event后做一次fsync，这个案例中因为系统没有重启，因此OS cache种的缓存没有丢失，因此relay log的完整行问题不大，也就是没有因为OS cache丢失导致，event的损坏。

四、相关测试

测试就简单了，我们确认一下从库的参数设置，如下，

- relay_log_recovery:OFF，默认值
- GTID_MODE:ON
- sync_relay_log/sync_relay_log_info:均为10000，默认值
- sync_master_info:10
- slave_parallel_workers:0
- skip-slave-start

这里设置 `sync_master_info` 为10是为了更好的测试故障，因为10000个event同步表一次需要较大的事务，改为10后多做几个小事务后就可以重现， `slave_parallel_workers` 更改为0是为了排除多线程的影响， `skip-slave-start` 是为了避免自动启动主从，更好的观察重启后的现象。

搭建主从使用传统POSITION方式如下，

```
reset  slave all;
change master to
master_host='192.168.1.64',
master_user='repl',
master_password='******',
master_port=3329,
MASTER_LOG_FILE = 'mysql-bin.000001',
MASTER_LOG_POS = 7422 ;
```

主库执行

```
mysql> show master status \G
*************************** 1. row ***************************
             File: mysql-bin.000001
         Position: 7422
     Binlog_Do_DB:
 Binlog_Ignore_DB:
Executed_Gtid_Set: 0da7e8d8-b6a3-11ef-a048-000c29d3a738:1-21
1 row inset (0.00 sec)

mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.01 sec)

mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.01 sec)
```

```
mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.00 sec)

mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.01 sec)

mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.02 sec)

mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.00 sec)

mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.01 sec)

mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.01 sec)

mysql> insert into test values(1,1,1,1);
Query OK, 1 row affected (0.01 sec)
```

从库查看

```
mysql> show slave status \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting forsource to send event
                  Master_Host: 192.168.1.64
                  Master_User: repl
                  Master_Port: 3329
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 10536
               Relay_Log_File: relay-bin.000002
                Relay_Log_Pos: 3439
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
...
          Exec_Master_Log_Pos: 10536
...
           Retrieved_Gtid_Set: 0da7e8d8-b6a3-11ef-a048-000c29d3a738:22-30
            Executed_Gtid_Set: 0da7e8d8-b6a3-11ef-a048-000c29d3a738:1-30
                Auto_Position: 0
```

这个时候我们发现执行的点和拉取的点都是10536，但是这个时候 `sync_master_info` 中的信息是10107。

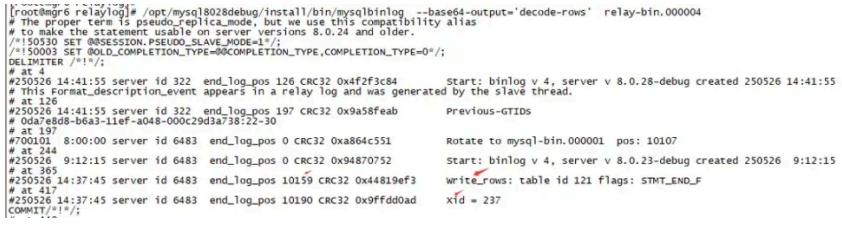
从库执行kill -9 后，重启从库

这个时候我们查看一下 `show slave status` 的信息如下，这个时候并没有启动I/O和SQL线程

```
mysql> show slave status \G
*************************** 1. row ***************************
               Slave_IO_State:
                  Master_Host: 192.168.1.64
                  Master_User: repl
                  Master_Port: 3329
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 10107    ### 这里回退了
       Relay_Master_Log_File: mysql-bin.000001
            Slave_IO_Running: No
           Slave_SQL_Running: No
...
          Exec_Master_Log_Pos: 10536
...
           Retrieved_Gtid_Set: 0da7e8d8-b6a3-11ef-a048-000c29d3a738:22-30
            Executed_Gtid_Set: 0da7e8d8-b6a3-11ef-a048-000c29d3a738:1-30
                Auto_Position: 0
```

启动I/O线程

这个时候我们启动I/O线程，I/O线程会从10107的postition点进行拉取，拉取的relay log我们就发现只有事务的后半部分内容，不包含GTID EVENT了。



如果SQL线程执行这个事务，必然报错。

启动SQL线程

启动SQL线程后就报错了如下，

五、如何规避这个问题

实际上这个问题很容易规避，我们可以开启slave carsh safe 就可以了。

这个实际上就是开启参数 `relay_log_recovery=ON`，这种情况下，启动的时候就会使用SQL线程的执行位点来覆盖I/O线程的拉取位点，这样不会在使用错误的I/O线程位点来拉取relay log了，SQL线程也自然不会报错了。

当然 `relay_log_recovery=ON` 的情况下老的relay log全部会被清理，这是使用这个参数的弊端，但是重新拉取一般没有问题，除非想保留relay log的情况下。

其次就这个问题而言如果是使用的**GTID AUTO_POSTITION**方式，也不会出现这种问题，因为**GTID AUTO_POSTITION**的方式使用的是GTID进行定位，不会用到I/O线程错误的位点进行拉取。

一般来讲正常的主从配置一定要打开 `relay_log_recovery=ON`，不仅可以避免这种情况，而且可以避免relay log损坏的问题。

同时设置 `relay_log_recovery=ON` 对MGR也有效，也是可以使用的，搭建的时候也是建议使用**GTID AUTO_POSTITION**方式。

最后修复也比较简单，因为这个库有GTID，因此可以简单的根据GTID来获取正确的position，重新指定一下正确的位点就可以了。

全文完。

- 借助AI将PostgreSQL迁移到GreatSQL（文末可免费领红包封面）
- MySQL/GreatSQL 应对安全扫描的绝招
- MySQL 8.0/8.4执行DDL会丢数据？是，但影响有限
- 运行 GreatSQL 时为什么要求关闭透明大页
- GreatSQL 运行时内存太高，超过90%怎么办（重发，附解决办法）
- 该开始关注 MySQL 8.4 了
- 慎重升级到MySQL 8.0.38及更高版本（文末附规避风险办法）
- Slave SQL线程与PXB FTWRL死锁问题分析
- GreatSQL统计信息维护管理
- GreatSQL死锁案例分析及扩展解读
- MySQL复制从库延迟优化思路
- MySQL复制从库延迟原因深入分析
- 源码解析丨一次慢SQL排查之旅
- 面试题：INSERT...t...SELECT s会对s表加锁吗
- MTS性能监控你知道多少
- MySQL对derived table的优化处理与使用限制
- MySQL一次大量内存消耗的跟踪

想看更多技术好文，点个“**在看**”吧

八怪 · 目录

上一篇 · 无损半同步复制下，主从切换后数据一致吗？