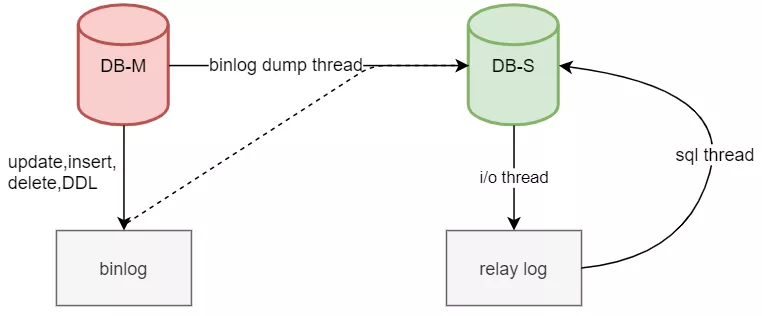
# 背景

MySQL复制不管用那个方式，都离不开binlog方式进行的。GTID作为position方式的延伸，在如今使用环境中带来了很多方便。

1、MySQL复制方式

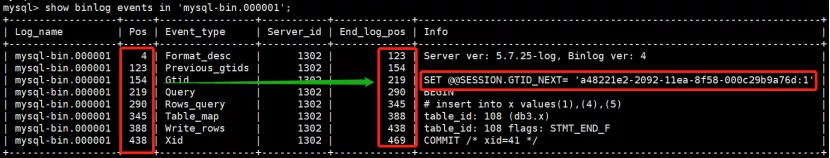
master用户写入数据，生成event记到binary log中， slave接收master上传来的binlog，然后按顺序应用，重现master上的操作。



传统的复制基于(file，pos)，当主从发生宕机，切换的时候有问题。**slave保存的是原master上的(file，pos)，无法直接指向新master上的(file，pos)**。

2、日志记录上position方式和GTID方式区别

直观图对比：



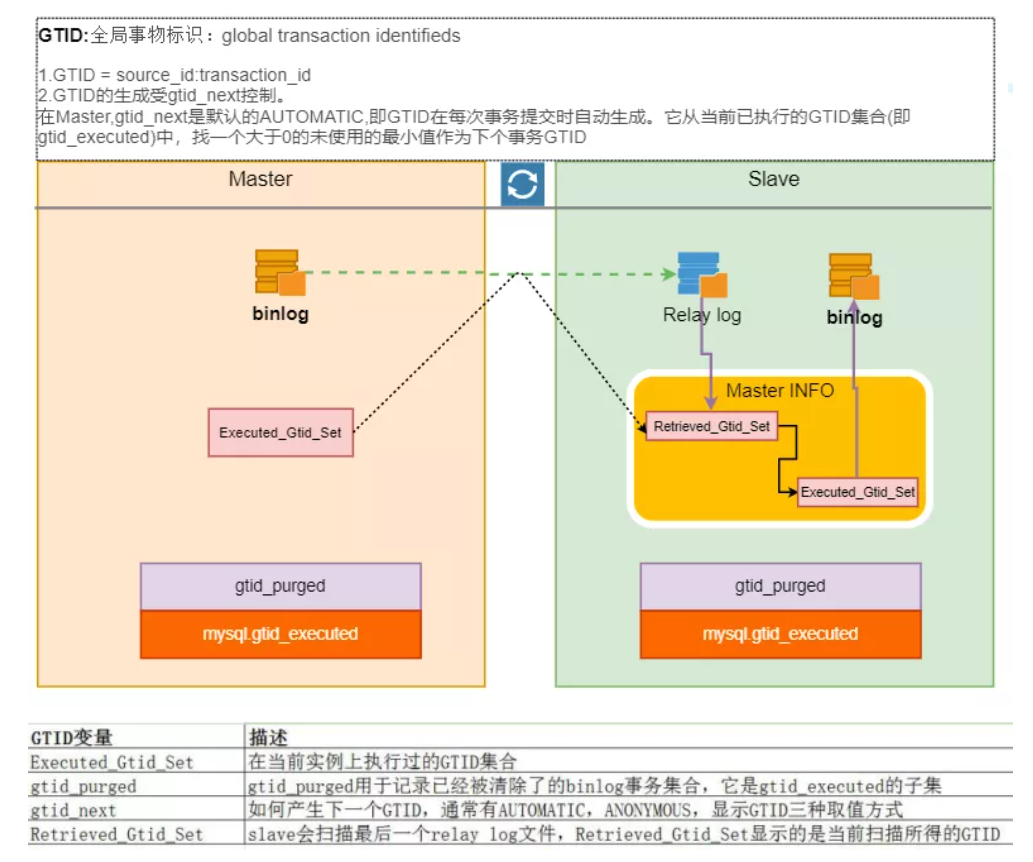
1）主从复制，默认是通过pos复制(postion)方式，将用户进行的每一项操作都进行编号(pos)，每一个event都有一个起始编号，一个终止编号。GTID就是类似于pos的一个作用，全局通用并且日志文件里事件的GTID值是一致的。  
 pos与GTID在日志里是一个标识符，在slave里以不同的方式展现。

2）GTID的生成受gtid\_next控制。  
 在Master上，gtid\_next是默认的AUTOMATIC，即GTID在每次事务提交时自动生成。它从当前已执行的GTID集合(即gtid\_executed)中，找一个大于0的未使用的最小值作为下个事务GTID。同时将GTID写入到binlog(set gtid\_next记录)，在实际的更新事务记录之前。  
 在Slave上，从binlog先读取到主库的GTID(即set gtid\_next记录)，而后执行的事务采用该GTID（因为GTID已经做到了全局唯一，所以我们它指向的不再是传统(file，pos)方式那样指向原来的主无法指向切换后的新主，它指向的GTID就是最新的结果了）。

# 概述

## 简介





参考：

<https://www.cnblogs.com/gomysql/p/7417995.html>

## 特点

### 优点

1. slave在做同步复制时，无须找到binlog日志和POS点，直接change master to master\_auto\_position=1即可，**自动找点同步**；
2. 搭建主从复制简单
3. 更简单的实现failover，不像传统方式那样在需要找log\_file和log\_Pos；
4. 复制集群有一个统一的方式识别复制位置，给集群管理带来了便利；
5. 正常情况下，GTID是连续没有空洞的，因此主从库出现数据冲突时，可以用添加空事物的方式进行跳过；
6. MySQL5.7.6版本开始可以在线升级gtid模式。

### 缺点

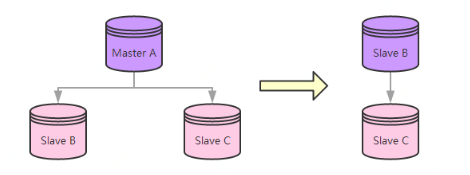
1. GTID同步复制是基于事务。所以Myisam表不支持，这可能导致多个GTID分配给同一个事务；
2. CREATE TABLE ...SELECT语句不支持。因为该语句会被拆分成createtable 和insert两个事务，并且这个两个事务被分配了同一个GTID，这会导致insert被备库忽略掉；
3. 不支持CREATE TEMPORARY TABLE、DROP TEMPORARYTABLE 临时表操作；
4. Errant transaction问题：即从库不能进行任何事物型操作，会引入新的GTID，当binlog被清除后，再进行主从切换，会导致其他从库找不到此GTID，从而挂载不上。

# 原理

参考：

基于GTID复制实现的工作原理：<http://mysql.taobao.org/monthly/2020/05/09/>

GTID的最大特性就是它的Failover能力，如下架构，当主库A crash时，需要进行主从切换，将B或C其中一台提升为主，传统模式我们无法确认哪台数据较新，由于同一个事务在每台机器上所在的binlog名字和位置都不一样，那么怎么找到C当前同步停止点，对应B的master\_log\_file和master\_log\_pos，需要通过程序对比或者借助MHA等工具。



GTID出现后，这个问题就显得非常简单。由于同一事务的GTID在所有节点上的值一致，那么根据C当前停止点的GTID就能唯一定位到B上的GTID。甚至由于MASTER\_AUTO\_POSITION功能的出现，我们都不需要知道GTID的具体值，直接使用CHANGE MASTER TO MASTER\_HOST='xxx', MASTER\_AUTO\_POSITION=1命令就可以直接完成failover的工作。

# 源码

参考：

BUG：GTID 和 DELAYED

<http://mysql.taobao.org/monthly/2014/09/01/>

<http://mysql.taobao.org/monthly/2014/09/02/>

BUG：GTID 和 binlog\_checksum

<http://mysql.taobao.org/monthly/2014/09/03/>

# 操作

## 开启



## 参数

show global variables like '%gtid%';



### gtid\_next

### gtid\_executed

当前执行gtid信息

mysql> SELECT @@GLOBAL.GTID\_EXECUTED;

+----------------------------------------------+

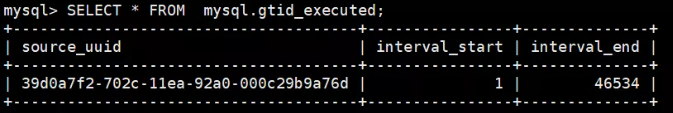
| @@GLOBAL.GTID\_EXECUTED |

+----------------------------------------------+

| 39d0a7f2-702c-11ea-92a0-000c29b9a76d:1-46534 |

+----------------------------------------------+

mysql> SELECT \* FROM mysql.gtid\_executed;



mysql.gtid\_executed表是由MySQL服务器提供给内部使用的。它允许副本在副本上禁用二进制日志记录时使用GTID，并允许在二进制日志丢失时保留GTID状态。RESET MASTER命令，gtid\_executed表将被清除。

**服务意外停止的情况下，当前二进制日志文件中的gtid集不会保存在gtid\_executed表。在恢复期间，这些gtid将从二进制日志文件添加到表中，以便可以继续复制**。

若MySQL服务器启用了二进制日志，则表mysql.gtid\_executed的更新仅在二进制rotation时发生，因为发生重启等情况依旧可以通过扫描二进制日志判断得知当前运行的GTID位置。

简单来说，该表会记录当前执行的GTID：

在MySQL 5.6中必须配置参数log\_slave\_updates的最重要原因在于当slave重启后，无法得知当前slave已经运行到的GTID位置，因为变量gtid\_executed是一个内存值：

MySQL 5.7将gtid\_executed这个值给持久化。采用的技巧与MySQL 5.6处理SQL thread保存位置的方式一样，即将GTID值持久化保存在一张InnoDB表中，并与用户事务一起进行提交，从而实现数据的一致性。

**触发条件：**

在binlog发生rotate(flush binary logs/达到max\_binlog\_size)或者关闭服务时，会把所有写入到binlog中的Gtid信息写入到mysql.gtid\_executed表。

从库：如果没有开启log\_bin或者没有开启log\_slave\_updates，从库在应用relay-log中的每个事务会执行一个insert mysql.gtid\_executed操作。

### gtid\_owned

### gtid\_purged

### Retrieved\_Gtid\_Set

Retrieved\_Gtid\_Set：从库已经接收到主库的事务编号

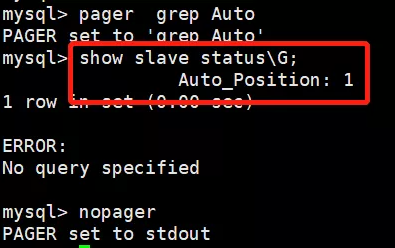
### Executed\_Gtid\_Set

Executed\_Gtid\_Set：已经执行的事务编号

## 判断复制方式

**如何判断复制方式GTID还是pos**

Show slave status查看Auto\_Position字段。0是pos方式，1是gtid方式。



gtid变更为pos方式：

change master to master\_auto\_position=0;

## 操作

### gtid设置

**1、gtid设置**

gtid\_mode=ON #必选

enforce-gtid-consistency=true #必选

log-bin=mysql #5.6必选 5.7.5和它之后可选，为了高可用，最好设置

server-id=1 #开启log-bin的必须设置

log-slave-updates=ON # 5.6必选 5.7.5和它之后可选，为了高可用切换，最好设置ON

### gtid跳过gtid\_next

**2、gtid跳过gtid\_next**

stop slave;

set gtid\_next='d74faa2d-5819-11e8-b248-ac853db70398:10603';

begin;commit;

set gtid\_next='automatic';

start slave;

备注：该操作类似于sql\_slave\_skip\_counter，只是跳过错误，不能保证数据一致性，需要人工介入，固强烈建议从机开启read\_only=1

### gtid清除gtid\_pureged

**3、gtid清除gtid\_pureged**

命令的实际意义：因没有binlog信息（expire\_logs\_days），不考虑这些gtid确认和回滚。常用备份恢复，搭建从库的时候使用。

自动触发机制：flush，服务器重新启动

使用场景：

在副本上禁用二进制日志记录提交的复制事务的GTIDs。

写入二进制日志文件的事务的GTIDs，该文件现在已被清除。

通过语句set @@GLOBAL.gtid\_purged显式添加到集合中的gtid。

mysqldump --set-gtid-purged=off/on 参数;

是否将GTID\_PURGED’添加到输出中

### gtid升级

**4、gtid升级**

pos升级gtid方式，条件允许建议重新搭建从库的方式。以下方式存在风险。

gtid\_mode可选值

ON：完全打开GTID，如果打开状态的备库接受到不带GTID的事务，则复制中断

ON\_PERMISSIV：可以认为是打开gtid前的过渡阶段，主库在设置成该值后会产生GTID，同时备库依然容忍带GTID和不带GTID的事务

OFF\_PERMISSIVE：可以认为是关闭GTID前的过渡阶段，主库在设置成该值后不再生成GTID,备库在接受到带GTID和不带GTID事务都可以容忍。主库在关闭GTID时，执行事务会产生一个Anonymous\_Gtid事件，会在备库执行：set @@session.gtid\_next=‘anonymous’

OFF：彻底关闭GTID，如果关闭状态的备库收到带GTID的事务，则复制中断

**从position模式切换到GTID模式：**

1) 在每个sever执行WARN模式:

这一步设置之后，使得所有事物都允许违反GTID的一致性

mysql>SET @@GLOBAL.ENFORCE\_GTID\_CONSISTENCY = WARN;

#这是第一个重要步骤. 您必须确保在进入下一步骤之前不会在错误日志中生成警告.

2) 在每个sever执行ON模式:

以确保所有的事务都不能违反GTID一致性

mysql>SET @@GLOBAL.ENFORCE\_GTID\_CONSISTENCY = ON;

3) 在每个sever执行OFF模式:

这一步表示，新的事务是匿名的，同事允许复制的事务是GTID或是匿名的

mysql>SET @@GLOBAL.GTID\_MODE = OFF\_PERMISSIVE;

#需要确保这一步操作在所有的服务器上执行

4) 在每个sever执行ON模式:

这一步表示，新的事务是GTID的，同事允许复制的事务是GTID或是匿名的

mysql>SET @@GLOBAL.GTID\_MODE = ON\_PERMISSIVE;

#需要确保这一步操作在所有的服务器上执行

5) 在每个服务器上，等待状态变量：ONGOING\_ANONYMOUS\_TRANSACTION\_COUNT为零。可以使用如下方式查询:

mysql>SHOW STATUS LIKE 'ONGOING\_ANONYMOUS\_TRANSACTION\_COUNT';

#在所有从库上查询该状态，必须为0 才能进行下一步。该状态宝石已标示为匿名的正在#进行的事务数量，如果状态值为0表示无事务等待被处理

等待生成到步骤5的所有事务复制到所有服务器. 可以在不停止更新的情况下执行此操作：唯一重要的是所有anonymous transactions都被复制了.

6) GTID\_MODE = ON在每所有服务器上执行:

mysql>SET @@GLOBAL.GTID\_MODE = ON;

7) 修改每个my.cnf文件:

gtid-mode=ON

ENFORCE\_GTID\_CONSISTENCY = ON

8) 上面复制虽然配置了GTID模式，但还是基于Binlog方式的。可通过选项MASTER\_AUTO\_POSITION设置为1，把复制调整为基于GTID模式的复制，具体操作如下:

mysql>STOP SLAVE [FOR CHANNEL 'channel'];

mysql>CHANGE MASTER TO MASTER\_AUTO\_POSITION = 1 [FOR CHANNEL 'channel'];

mysql>START SLAVE [FOR CHANNEL 'channel'];

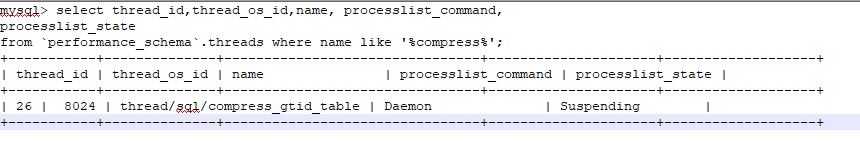
### gtid压缩gtid\_executed\_compression\_period

**5、gtid压缩gtid\_executed\_compression\_period**

启用GTID时，服务器会定期在mysql.gtid\_executed表上执行此类压缩。通过设置gtid\_executed\_compression\_period系统变量，可以控制压缩表之前允许的事务数，从而控制压缩率。该变量的默认值为1000; 这意味着，默认情况下，在每1000次事务之后执行表压缩。

将gtid\_executed\_compression\_period设置为0可以防止执行压缩; 但是，如果执行此操作，应该为gtid\_executed表可能需要的磁盘空间量的大幅增加做好准备。

使用以下语句查询：



备注：如发现processlist\_state值一直是: "Compressing gtid\_executed table"说明进行压缩。记录锁的内存从操作系统申请，所以当表gtid\_executed不断增大时，最终会导致MySQL OOM。

### binlog\_gtid\_simple\_recovery

**6、binlog\_gtid\_simple\_recovery**

MySQL启动或重启时在搜索GTID期间迭代二进制日志文件的方式。就是为了初始化gtid\_executed，gtid\_purged参数，扫描binlog或则event相关信息。

MySQL5.7.7或更老版本的二进制日志，需设置binlog\_gtid\_simple\_recovery=FALSE，如果存在非gtid的binlog比较多的时候，会非常影响性能的。

# 相关函数

## 基本函数

|  |  |  |
| --- | --- | --- |
| 基本函数 | 功能 | 备注 |
| GTID\_SUBSET(set1,set2) | 检查子集：set1<=set2 | YES返回1，NO返回0 |
| GTID\_SUBTRACT(set1,set2) | 计算子集：set1-set2 | 返回set1中存在，不在set2中的GTID |
| WAIT\_FOR\_EXECUTED\_GTID\_SET(set1,[timeout]) | 等待备机回放set1，完成或超时 | 1. 回放完成返回0，超时返回1，错误返回其他值； 2. Timeout为0，表示不超时，一直等待 |

## 扩展功能

|  |  |  |
| --- | --- | --- |
| 复合查询操作 | 拓展功能 | 备注 |
| GTID\_SUBTRACT(set1,'') | 将字符串按照GTID正规化 | 利用set1与空集相减，达到将set1字段正规化的目的，比如相同的uuid段合并 |
| GTID\_SUBSET(set1,set2) AND GTID\_SUBSET(set2,set1) | 判断集合相等 | 判断set1是否与set2相等，相等返回1，否则返回0 |
| 1. CONCT(set1,’,’,set2) 2. GTID\_SUBTRACT(CONCAT(set1,’,’,set2)) | 计算并集 | 1. 直接将set1和set2两个字符串合并，分隔符为逗号 2. 将set1,set2合并后，再进行正则处理，可输出标准GTID字符串 |
| GTID\_SUBTRACT(set1,GTID\_SUBTRACT(set1,set2)) | 计算交集 | set1-(set1-set2)，返回set1与set2的交集 |
| GTID\_SUBSET(set1,GTID\_SUBTRACT(set1,set2)) | 判断集合不相交 | set1与set2不相交，则返回1，其他返回0 |
| GTID\_SUBTRACT(set1,GTID\_SUBTRACT(set1,CONCAT(uuid,’:1-’,(1<<63)-2))) | 指定uuid获取GTID | 提取set1中指定uuid的部分GTID |
| GTID\_SUBTRACT(set1,CONCAT(uuid,’:1-’,(1<<63)-2)) | 去除指定uuid部分 | 删除set1中指定uuid |

# 限制

到目前为止已经发展完善，但存在一些场景是受限的。

1、create table xxx as select：

拆分成两部分：

create table xxxx like table;

insert into xxxx select \*from table;

2、临时表的限制

使用GTID复制模式：

1)、不支持create temporary table 和 drop temporary table。

2)、在autocommit=1的情况下可以创建临时表，

3)、Master端创建临时表不产生GTID信息，所以不会同步到slave，但是在删除临时表的时候会产生GTID会导致，主从中断.

3、事务操作

涉及非事务性存储引擎的更新，非事务性存储引擎事务性存储引擎更新表则不能在同一条语句或同一事务中执行。

4、mysql\_upgrade

GTID模式和mysql\_upgrade。在启用全局事务标识符(GTIDs)的情况下运行时，不要通过mysql\_upgrade(——write binlog选项)启用二进制日志记录。

5、sql\_slave\_skip\_counter

传统模式的跳过postion方式gtid模式下不支持。

# 复制错误

## 手动跳过错误事物（在从库上）

STOP SLAVE;

RESET MASTER;

SET @@GLOBAL.GTID\_PURGED

='f2b6c829-9c87-11e4-84e8-deadeb54b599:1-32';

START SLAVE;

上面这些命令的用意是，忽略f2b6c829-9c87-11e4-84e8-deadeb54b599:32这个GTID事务，下一次事务接着从33这个GTID开始，即可跳过上述错误。

**注：**无论是否开启了GTID，都可以使用percona的pt-slave-restart工具去跳过错误。

## Errant transaction问题修复

此问题主要是采用GTID复制的情况下，在slave上进行了事物操作，此时这台slave就多出来一个或多个其他slave节点和master节点没有的事务。我们知道Binlog默认保留7天，7天后这些事物产生的binlog会被删除，当发生failover的时，这个slave被提升为主，由于其他从库已经找不到新主库事物所产生的binlog，此时其他从库会挂载不上，造成数据库单点，十分危险。

1. 传统解决方案：通过在主库手动设置下一次事物GTID，执行一条空事物，实现跟从库一致



1. 通过类似一种“欺骗”方（优先选择此方式，无需操作主库）

我们可能知道从库是不允许有事物直接写入的，可能为了测试某条事物是否正确，这条事物生成的GTID完全可以不要，但是gtid\_executed是不允许删除的。我们是否有其他方式呢？

通过上面的解释可以知道：

gtid\_purged：已清除的binlog中包含的GTID集合，purged掉的GTID会包含到gtid\_executed中；

gtid\_executed：用来保存已经执行过的GTID，当reset master时，此值会被清空；

知道了这两点，我们是否可以通过reset master来清空gtid\_executed，再手动指定gtid\_purged，来同步到gtid\_executed中来实现跟主库一致。



1. mysqlslavetrx优雅处理方式

此方法使用较少，不做详细介绍，可以查看官方文档

https://dev.mysql.com/doc/mysql-utilities/1.6/en/mysqlslavetrx.html

https://www.percona.com/blog/2015/12/02/gtid-failover-with-mysqlslavetrx-fix-errant-transactions/