任务背景

为了进一步保证数据库的安全和可靠，在保证数据安全和服务正常的情况下，将原有的单机MySQL做成基于GTIDs主从模式，让数据更加安全可靠。

任务要求

1. 备份数据库
2. 在不影响服务正常使用的情况下，搭建基于GITDs的主从模式。

课程目标

- 了解什么是MySQL的Replication
- **理解MySQL的Replication的架构原理**
- **掌握MySQL的基本复制架构的搭建（M-S重点）**
- 了解和掌握基于GTIDs的复制特点及搭建

理论储备

一、MySQL集群概述

1. 集群的主要类型

- 高可用集群（High Available Cluster，HA）
- 高可用集群是指通过**特殊的软件**把独立的服务器连接起来，组成一个能够**提供故障切换**（Fail Over）功能的集群

2. 如何衡量高可用

可用性级别(指标)	年度宕机时间	描述	叫法
99%	3.65天/年	基本可用系统	2个9
99.9%	8.76小时/年	可用系统	3个9
99.99%	52.6分钟/年	高可用系统	4个9
99.999%	5.3分钟/年	抗故障系统	5个9
99.9999%	32秒/年	容错系统	6个9

计算方法：

```
1年 = 365天 = 8760小时
99% = 8760 * 1% = 8760 * 0.01 = 87.6小时=3.65天
99.9 = 8760 * 0.1% = 8760 * 0.001 = 8.76小时
99.99 = 8760 * 0.0001 = 0.876小时 = 0.876 * 60 = 52.6分钟
99.999 = 8760 * 0.00001 = 0.0876小时 = 0.0876 * 60 = 5.26分钟
```

3. 常用的集群架构

- **MySQL Replication**
- MySQL Cluster
- MySQL Group Replication （MGR） 5.7.17
- MariaDB Galera Cluster
- MHA|Keepalived|HeartBeat|Lvs，Haproxy等技术构建高可用集群

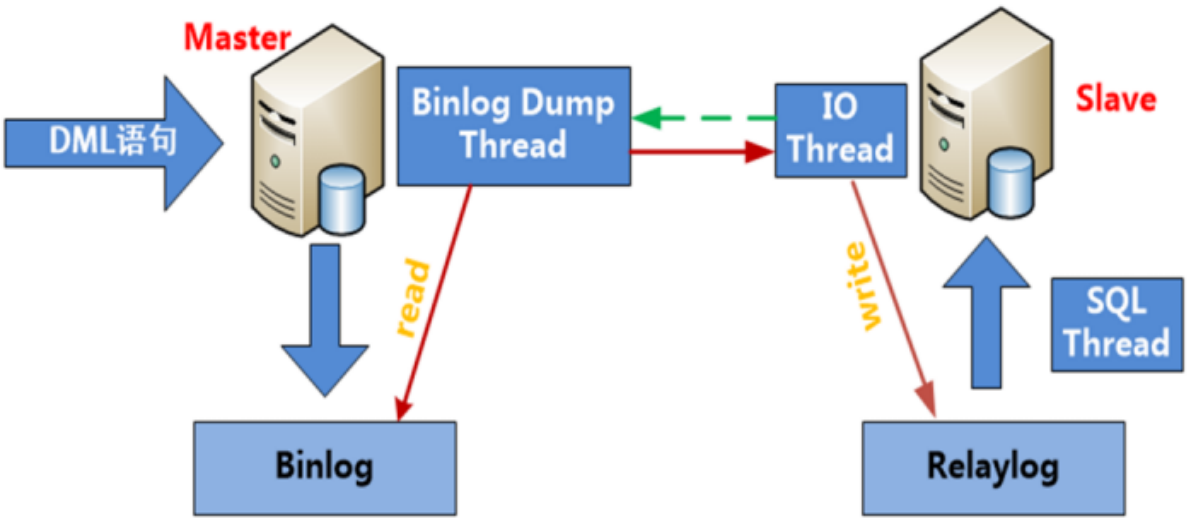
二、MySQL复制简介

1. 什么是MySQL复制

- Replication可以实现将数据从一台数据库服务器（master）复制到一台到多台数据库服务器(slave)
- 默认情况下，属于**异步**复制，所以无需维持长连接

2. MySQL复制原理

简单来说，master将数据库的改变写入**二进制日志**，slave同步这些二进制日志，并根据这些二进制日志进行**数据重演操作**，实现数据异步同步。

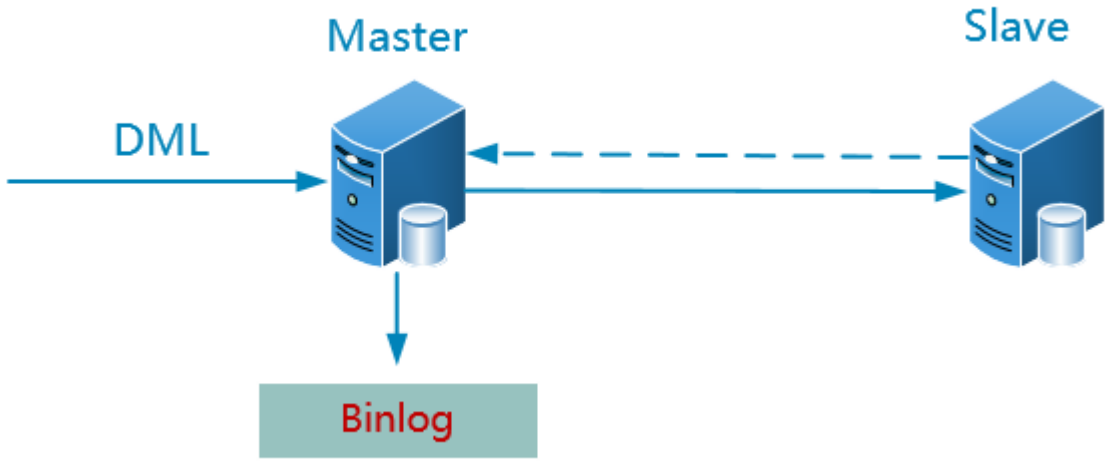


详细描述：

1. slave端的IO线程连上master端，请求
2. master端返回给slave端，bin log文件名和位置信息
3. IO线程把master端的bin log内容依次写到slave端relay log里，并把master端的bin-log文件名和位置记录到master.info里。
4. salve端的sql线程，检测到relay log中内容更新，就会解析relay log里更新的内容，并执行这些操作

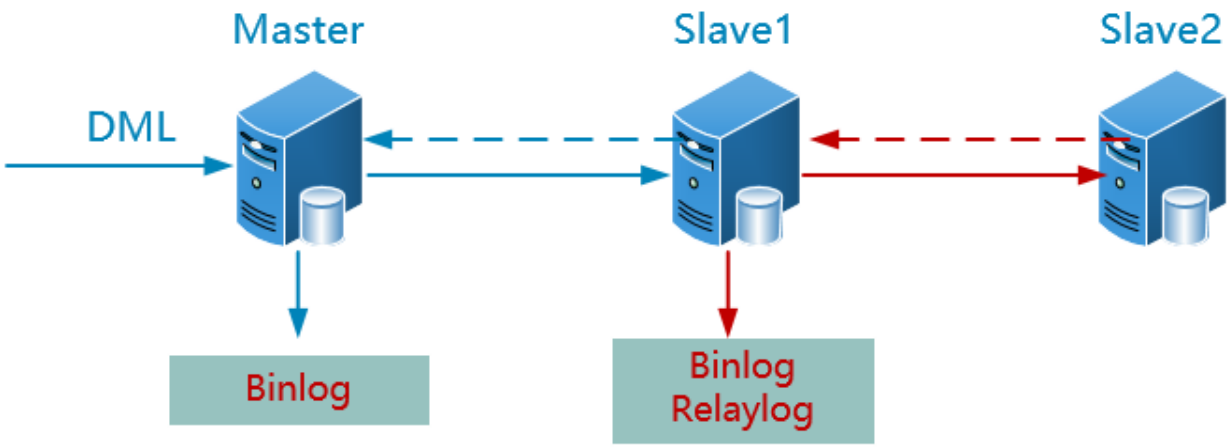
3. MySQL复制架构

3.1 双机热备（AB复制）



默认情况下，master接受读写请求，slave只接受读请求以减轻master的压力。

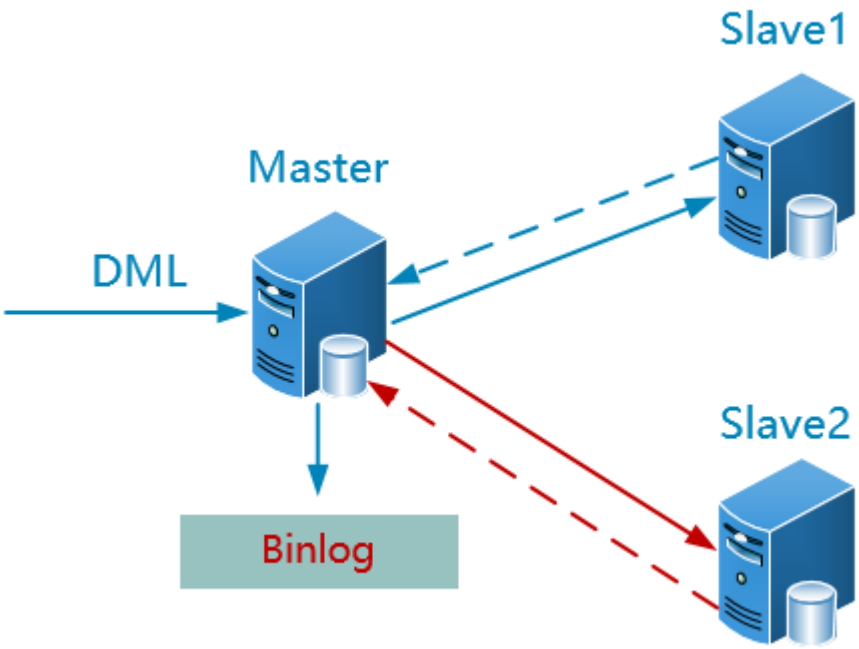
3.2 级联复制



优点：进一步分担读压力

缺点：slave1 出现故障，后面的所有级联slave服务器都会同步失败

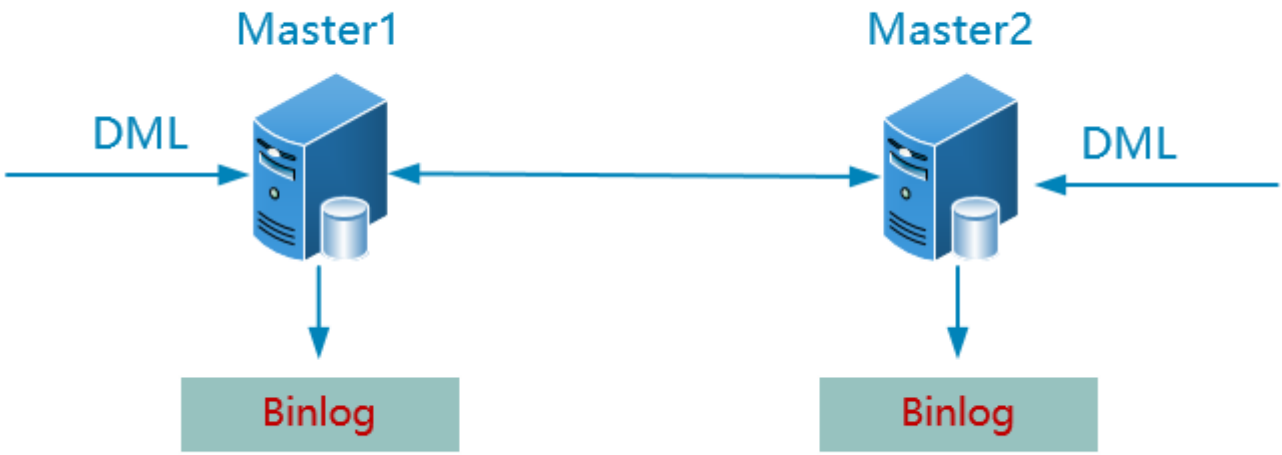
3.3 并联复制



优点：解决上面的slave1的单点故障，同时也分担读压力

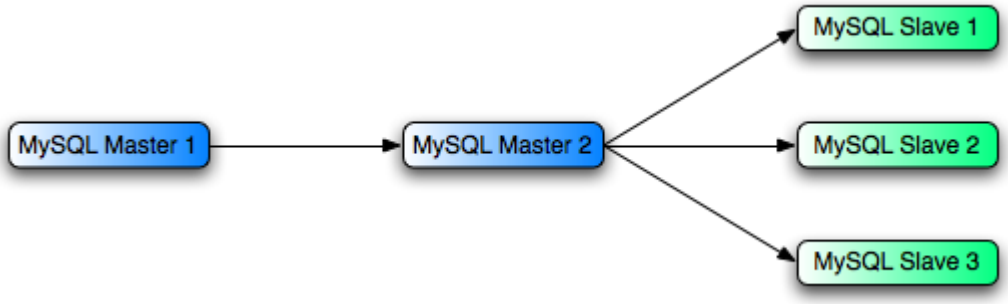
缺点：间接增加master的压力（传输二进制日志压力）

3.4 双主复制



特点：

从命名来看，两台master好像都能接受读、写请求，但实际上，往往运作的过程中，同一时刻只有其中一台master会接受写请求，另外一台接受读请求。



三、MySQL复制搭建

1. M—>S架构

```
环境准备及要求:
Master:10.1.1.10  master
Slave:10.1.1.20   slave

1. 关闭防火墙和selinux
# systemctl stop firewalld
# systemctl disable firewalld       永久关闭防火墙，开机不要自动启动
# systemctl list-unit-files|grep firewalld      查看是否开机自动关闭
firewalld.service                             disabled

2. 更改主机名
# hostnamectl set-hostname master.heima.cc      master上面
# hostnamectl set-hostname slave.heima.cc       slave上面

3. hosts文件中两台服务器主机名和ip地址一一对应起来
# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.1.1.10   master.heima.cc master
10.1.1.20   slave.heima.cc  slave

4. 系统时间需要同步
1）搭建时间同步服务器
IP:10.1.1.10  时间同步服务器
2）master和slave写计划任务每隔5分钟同步时间
*/5 * * * * rdate -s 10.1.1.10

5. master和slave的数据库版本保持一致



思路:
0. slave必须安装相同版本的mysql数据库软件
1. master端必须开启二进制日志；slave端必须开启relaylog日志
2. master端和slave端的server_id号必须不能一致
3. 告诉slave需要同步的master主机的IP、user、password等..
```

具体步骤：

0. slave安装mysql数据库软件

```
[root@slave ~]# id mysql
id: mysql: no such user
[root@slave ~]# useradd -r -s /sbin/nologin mysql
[root@slave ~]# mkdir /usr/local/mysql
[root@slave ~]# cd /usr/src/
[root@slave src]# ls
debug  kernels  mysql-5.6.35-linux-glibc2.5-x86_64.tar.gz
[root@slave src]# tar xf mysql-5.6.35-linux-glibc2.5-x86_64.tar.gz
[root@slave src]# ls
debug  kernels  mysql-5.6.35-linux-glibc2.5-x86_64  mysql-5.6.35-linux-glibc2.5-x86_64.tar.gz
[root@slave src]# cp -a mysql-5.6.35-linux-glibc2.5-x86_64/* /usr/local/mysql/
[root@slave src]# chown -R mysql. /usr/local/mysql/
```

注意：

暂时不需要初始化数据库文件，只是安装好了和master相同版本的mysql数据库软件；后面向master来同步所有数据。

1. 修改配置文件（master和slave）

master：

```
vim /etc/my.cnf
[root@master ~]# cat /etc/my.cnf
[mysql]
socket = /tmp/mysql.sock
user = root
password = 123
[mysqld]
basedir = /data/mysql35
datadir = /data/mysql35/data
port = 3306
socket = /tmp/mysql.sock
log_bin = /data/mysql35/data/mybinlog
server_id = 10    必须不一致，用来标识数据库实例
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
```

slave：

```
[root@slave ~]# cat /etc/my.cnf
[mysqld]
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data
port = 3306
socket = /tmp/mysql.sock
#log_bin = /usr/local/mysql/data/mybinlog
server_id = 20
relay_log = /usr/local/mysql/data/relay.log      需要开启开启中继日志
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
```

2. 初始化数据，使两边数据一致。（以master为主）

master数据库已经有业务数据，需要停服务拷贝到slave服务器。

1)停止master上的mysql服务

```
[root@master ~]# mysqladmin shutdown -p123
2)删除auto.cnf文件，里面保存的是当前数据库的UUID
[root@master ~]# cd /data/mysql35/data/
[root@master data]# cat auto.cnf
[auto]
server-uuid=64f8758d-fd63-11e8-91d3-000c2990b8ff
[root@master data]# rm -f auto.cnf
3)将master上面的所有数据文件同步到slave上
[root@master data]# rsync -av /data/mysql35/data/ 10.1.1.20:/usr/local/mysql/data
root@10.1.1.20's password:


3. 启动master和slave上的数据库
[root@master mysql]# service mysql35 start
Starting MySQL.. SUCCESS!

[root@slave mysql]# mysqld_safe --user=mysql &


4. master端创建授权用户
mysql> grant replication slave on *.* to 'slave'@'10.1.1.%' identified by '123';
mysql> flush privileges;

5. 查看master的正在写的二进制文件名和位置
mysql> flush tables with read lock; 先加锁，防止两边数据不一致;如果业务还未上线，这个就没有必要了
Query OK, 0 rows affected (0.00 sec)
mysql> show master status; 只有打开二进制日志，这句命令才有结果，表示当前数据库的二进制日志写到什么位置
mysql> show master status;
+-----------------+----------+--------------+------------------+-------------------+
| File            | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----------------+----------+--------------+------------------+-------------------+
| mybinlog.000002 |      405 |              |                  |                   |
+-----------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)


二进制文件名        正在写入的位置

6. slave端设定复制信息
mysql> change master to
master_host='10.1.1.10',master_user='slave',master_password='123',master_port=3306,master_log_fi
le='mybinlog.000002',master_log_pos=405;

说明:
master_host   master的IP
master_user   复制的用户
master_password  复制用户密码
master_port       master的端口号
master_log_file   master正在写的二进制文件名，锁表后查看的
master_log_pos    master正在写的二进制位置

7. 启动复制线程，开始同步
mysql> start slave;
```

```
mysql> show slave status \G;

            Slave_IO_Running: Yes 代表成功连接到master并且下载日志
            Slave_SQL_Running: Yes 代表成功执行日志中的SQL语句


mysql> show slave status\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.1.1.10
                  Master_User: slave
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mybinlog.000002
          Read_Master_Log_Pos: 405
               Relay_Log_File: relay.000002
                Relay_Log_Pos: 282
        Relay_Master_Log_File: mybinlog.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes

回到master端解锁:
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)

8. 测试验证
master写—>slave可以看到
slave写—>master看不到



总结说明:
1. Master-Slave搭建必须保证初始数据一致
2. 如果master和slave初始数据不一致导致主从同步失效可以使用sql_slave_skip_counter临时跳过该事务
SET GLOBAL sql_slave_skip_counter = N  N代表跳过N个事务
3. 传统的AB复制方式可以使用变量:sql_slave_skip_counter,基于GTIDs的方式不支持

注意:跳过事务应该在slave上进行
mysql> SET GLOBAL sql_slave_skip_counter =1;
```

情况:
　假设master故障,数据已经损害了、丢失了,那么最简单的方法就是重装master数据库,把master作为slave的从,原来的slave就变成新架构的主

注意:
　确保新的架构复制成功之后,回到slave服务器,把数据目录下的master.info文件删除,否则下次如果slave重启数据库服务,会自动连接master;slave IO线程把master端的bin log内容依次写到slave端relay bin log里,并把master端的bin-log文件名和位置记录到master.info里

2. M1—>M2架构

步骤：
1. 保证2台msyql数据库的初始数据一致
略
2. 修改配置文件
master:
```
[mysqld]
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data
port = 3307
socket = /usr/local/mysql/mysql.sock
log-error = /usr/local/mysql/data/mysql3307.err
log-bin = /usr/local/mysql/data/mybinlog
server_id = 10
relay_log = /usr/local/mysql/data/relay
[client]
socket=/usr/local/mysql/mysql.sock
```

2) slave1上
```
[root@slave1 ~]# cat /etc/my.cnf
[mysqld]
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data
port = 3307
socket = /usr/local/mysql/mysql.sock
log-error = /usr/local/mysql/data/mysql3307.err
log-bin = /usr/local/mysql/data/mybinlog
server_id = 20
relay_log = /usr/local/mysql/data/relay
[client]
socket=/usr/local/mysql/mysql.sock
```

3. 启动数据库

4. 两边创建授权用户并查看master状态

5. 两边配置主从

6. 测试验证

总结：
在架构工作正常的情况下，master1没有出现故障的情况下，尽可能不要给master1写请求的同时也给master2写请求。

故障情况：
状况1：
如果master2出现故障，而且仅仅是简单的故障，没有出现数据丢失，那么只需要重新启动master2，master2会把落后的数据自动同步。

状况2:
如果master2出现严重故障，数据已经丢失了，建议重新使用master1过去某个备份去搭建master2

修复步骤:
1) 停止master1的复制线程: stop slave
2) 重新搭建master2
3) 重新设定master1向新的master2复制的设定，然后start slave

3. M-S1-S2架构

打开log-slave-updates=1，让第一台传过来relay日志记录到自己的二进制日志
思路:
先搭建好主从—>然后在加入slave2

1. 环境准备
master:10.1.1.10 master上mysql数据库已搭建成功，并且有相应数据
slave1:10.1.1.20 已经和master互为主从

slave2:10.1.1.30 新环境

2. 在slave2上创建用户
useradd -r -s /sbin/nologin mysql

3. slave2上同步slave1上的数据
1) 停止slave1上的mysql服务
[root@slave1 ~]# mysqladmin shutdown -p123

2) 物理拷贝所有的数据文件/usr/local/mysql(安装目录和数据目录一起)
[root@slave1 ~]# rsync -av /usr/local/mysql 10.1.1.30:/usr/local/

4. 修改slave1和slave2的配置文件
slave1上:
vim /etc/my.cnf
[mysqld]
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data
port = 3306
socket = /tmp/mysql.sock
log_bin = /usr/local/mysql/data/mybinlog
server_id = 20
relay_log = /usr/local/mysql/data/relay.log
log-slave-updates=1

slave2上:

[mysqld]
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data

```
port = 3306
socket = /tmp/mysql.sock
server_id = 30
relay_log = /usr/local/mysql/data/relay.log
```

5. 启动slave1和slave2的mysql服务，但是不要启动slave1的IO线程
注意：尽可能在业务维护时间操作（停服务）或者在启动slave1之前将master锁表

1） 删除刚刚同步过来的auto.cnf

```
[root@slave2 data]# export PATH=$PATH:/usr/local/mysql/bin/
[root@slave2 data]# echo "export PATH=$PATH:/usr/local/mysql/bin/" >>/etc/profile
[root@slave2 data]# rm -f auto.cnf
```

2）启动服务

```
[root@slave1 data]# mysqld_safe --user=mysql &
[root@slave2 data]# mysqld_safe --user=mysql &
```

注意：slave1启动玩服务后需要停止IO线程

```
mysql> stop slave;
Query OK, 0 rows affected (0.01 sec)
```

6.slave2配置同步信息向slave1

1）slave1上创建授权用户

```
mysql> grant replication slave on *.* to 'slave'@'10.1.1.30' identified by '123';
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

2）查看slave1上的二进制日志文件信息

```
mysql> show master status\G
*************************** 1. row ***************************
             File: mybinlog.000001
         Position: 406
     Binlog_Do_DB:
 Binlog_Ignore_DB:
Executed_Gtid_Set:
```

3）在slave2上面设置同步信息

```
mysql> change master to
master_host='10.1.1.20',master_user='slave',master_password='123',master_port=3306,master_log_fi
le='mybinlog.000001',master_log_pos=406;
ERROR 1198 (HY000): This operation cannot be performed with a running slave; run STOP SLAVE
first
```

原因：slave在运行
解决：

```
mysql> stop slave
```

再次配置然后启动slave，检查是否ok：

```
mysql> show slave status\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.1.1.20
                  Master_User: slave
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mybinlog.000001
          Read_Master_Log_Pos: 406
               Relay_Log_File: relay.000002
                Relay_Log_Pos: 282
        Relay_Master_Log_File: mybinlog.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

4）打开slave1上的slave

验证IO和SQL线程都是YES表示配置成功：

```
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

7. 测试验证

master上更新数据，slave1和slave2都可以同步；

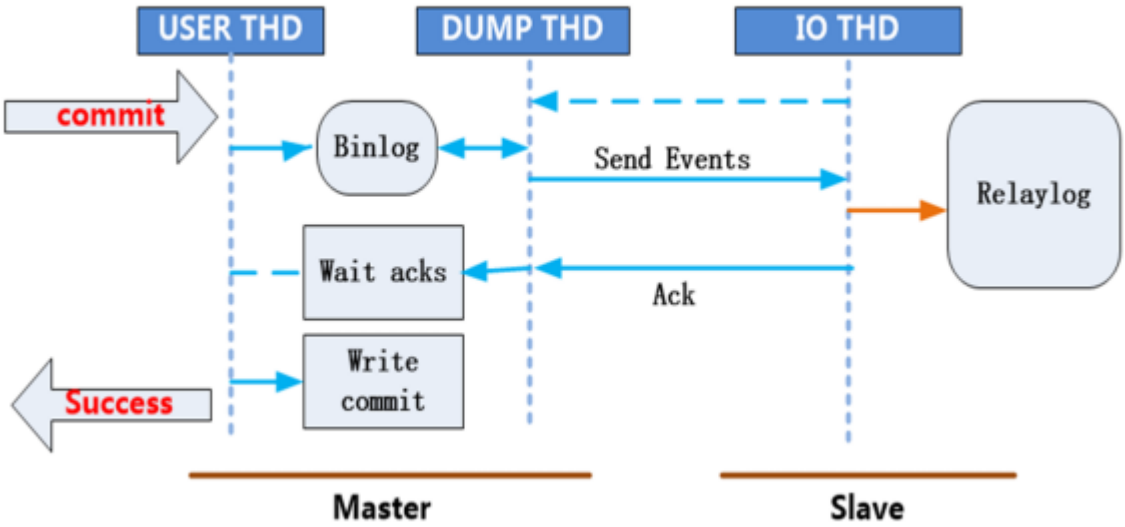
关闭slave1，再次更新master数据，slave2不能同步

启动slave1后，slave1和slave2又再次同步了数据

4. 总结

上面的复制架构默认都是异步的，也就是主库将binlog日志发送给从库，这一动作就结束了，并不会验证从库是否接受完毕。这样可以提供最佳的性能。但是同时也带来了很高的风险，当主服务器或者从服务器发生故障时，极有可能从服务器没有接到主服务器发过来的binglog日志，这样就会导致主从数据不一致，甚至导致数据丢失。为了解决该问题，mysql5.5引入了半同步复制模式。

5. 半同步复制



所谓的半同步复制就是master每commit一个事务(简单来说就是做一个改变数据的操作）,要确保slave接受完主服务器发送的binlog日志文件并写入到自己的中继日志relay log里，然后会给master信号，告诉对方已经接收完毕，这样master才能把事物成功commit。这样就保证了master-slave的数据绝对的一致（但是以牺牲master的性能为代价).但等待时间也是可以调整的。

搭建步骤：

```
mysql半同步复制等待时间超时后(默认时间为10秒)，会自动转换成异步复制
架构:
master—>slave

步骤:
第一大步:
先要搭建好mysqlAB异步复制

第二大步:在异步基础上转成半同步复制
需要安装插件:
# ls /usr/local/mysql/lib/plugin/semisync_*
/usr/local/mysql/lib/plugin/semisync_master.so  master上
/usr/local/mysql/lib/plugin/semisync_slave.so      slave上

1、在master上安装这个插件
mysql> install plugin rpl_semi_sync_master soname 'semisync_master.so';
Query OK, 0 rows affected (0.00 sec)

删除插件的方法:
mysql > uninstall plugin rpl_semi_sync_master;
mysql> show global variables like 'rpl_semi_sync%';  --安装OK后，主上会多几个参数
+-------------------------------------+-------+
| Variable_name                       | Value |
+-------------------------------------+-------+
| rpl_semi_sync_master_enabled        | OFF   |  --是否启用master的半同步复制
| rpl_semi_sync_master_timeout        | 10000 |  --默认主等待从返回信息的超时间时间，10秒。动态可调
| rpl_semi_sync_master_trace_level    | 32    |     用于开启半同步复制模式时的调试级别，默认是32
```

```
| rpl_semi_sync_master_wait_no_slave | ON    |  --是否允许每个事物的提交都要等待slave的信号.on为每一个事物都等待，off则表示slave追赶上后，也不会开启半同步模式，需要手动开启
+------------------------------------+-------+
官方手册里都有解释：5.1.1 Server Option and Variable Reference
rpl_semi_sync_master_trace_level
Default  32
The semisynchronous replication debug trace level on the master. Four levels are defined:
1 = general level (for example, time function failures)  一般等级
16 = detail level (more verbose information) 更详细的信息
32 = net wait level (more information about network waits)  网络等待等级
64 = function level (information about function entry and exit)  函数等级  （有关函数输入和退出的信息）

2、在slave上安装插件
slave> install plugin rpl_semi_sync_slave soname 'semisync_slave.so';
Query OK, 0 rows affected (0.03 sec)

slave>  show global variables like 'rpl_semi_sync%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| rpl_semi_sync_slave_enabled     | OFF   |  slave是否启用半同步复制
| rpl_semi_sync_slave_trace_level | 32    |
+---------------------------------+-------+

3, master上激活半同步复制
master> set global rpl_semi_sync_master_enabled =on;
Query OK, 0 rows affected (0.00 sec)

4, slave上激活半同步复制
slave> set global rpl_semi_sync_slave_enabled=on;
slave> stop slave IO_THREAD;
slave> start slave IO_THREAD;

5, 在master查看状态
master > show global status like 'rpl_semi_sync%';
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |有一个从服务器启用半同步复制
| Rpl_semi_sync_master_net_avg_wait_time     | 0     |master等待slave回复的平均等待时间。单位毫秒
| Rpl_semi_sync_master_net_wait_time         | 0     |master总的等待时间。单位毫秒
| Rpl_semi_sync_master_net_waits             | 0     |master等待slave回复的总的等待次数
| Rpl_semi_sync_master_no_times              | 0     |master关闭半同步复制的次数
| Rpl_semi_sync_master_no_tx                 | 0     |表示从服务器确认的不成功提交的数量
| Rpl_semi_sync_master_status                | ON    |标记master现在是否是半同步复制状态
| Rpl_semi_sync_master_timefunc_failures     | 0     |master调用时间（如gettimeofday())失败的次数
| Rpl_semi_sync_master_tx_avg_wait_time      | 0     |master花在每个事务上的平均等待时间
| Rpl_semi_sync_master_tx_wait_time          | 0     |master花在事物上总的等待时间
| Rpl_semi_sync_master_tx_waits              | 0     |master事物等待次数
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |后来的先到了，而先来的还没有到的次数
| Rpl_semi_sync_master_wait_sessions         | 0     |当前有多少个session因为slave回复而造成等待
| Rpl_semi_sync_master_yes_tx                | 0     |表示从服务器确认的成功提交数量
```

```
+-------------------------------------------+-------+

6，在slave上查看状态就只有下面一条信息
slave > show global status like 'rpl_semi_sync%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Rpl_semi_sync_slave_status | ON    |
+----------------------------+-------+


第三大步：测试
工作原理：当slave从库的IO_Thread 线程将binlog日志接受完毕后，要给master一个确认，如果超过10s未收到slave的接收确认信号，那么就会自动转换为传统的异步复制模式。
正常情况下，master插入一条记录，查看slave是否有成功返回

master > insert into a values (3);
Query OK, 1 row affected (0.01 sec)

master > show global status like 'rpl_semi_sync%_yes_tx';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Rpl_semi_sync_master_yes_tx |  1    |   --表示这次事物成功从slave返回一次确认信号
+-----------------------------+-------+

模拟故障：当slave挂掉后，master这边更改操作
service stop mysql
或者直接停止slave的IO_thread线程
stop slave io_thread;
master> insert into a values (4);
Query OK, 1 row affected (10.00 sec)   --这次插入一个值需要等待10秒（默认的等待时间）

master> insert into a values (5);
Query OK, 1 row affected (0.01 sec) --现在自动转成了原来的异步模式（类似oracle DG里的最大性能模式）


再次把slave启动，看到半同步复制没启来，是异步模式
重新按下面的步骤把同步模式再启起来就可以了
slave> set global rpl_semi_sync_slave_enabled=on;
slave> stop slave IO_THREAD;
slave> start slave IO_THREAD;

或者可以将该参数写入到配置文件中：
master：rpl_semi_sync_master_enabled=1
slave：rpl_semi_sync_slave_enabled=1

结果：master需要等到slave确认后才能提交，如果等不到确认消息，master等待10s种后自动变成异步同步;slave启起来后，master上改变的数据还是会自动复制过来，数据又回到一致

等待时间可以动态调整：
mysql> set global rpl_semi_sync_master_timeout=3600000;
mysql> show global variables like 'rpl_semi_sync%';
```

```
+-----------------------------------+---------+
| Variable_name                     | Value   |
+-----------------------------------+---------+
| rpl_semi_sync_master_enabled      | ON      |
| rpl_semi_sync_master_timeout      | 3600000 |
| rpl_semi_sync_master_trace_level  | 32      |
| rpl_semi_sync_master_wait_no_slave | ON      |
+-----------------------------------+---------+
```

mysql开源管理工具 —>maatkit --perl写的

maatkit-7540.tar.gz

--在mysql AB的slave上安装（只需要在slave上安装，包含下面的步骤都是在slave上做的）

```
# tar xf maatkit-7540.tar.gz -C /usr/src/
# cd /usr/src/maatkit-7540/
```

安装方法README文件里有写

```
# perl Makefile.PL   --如果不成功，需要安装perl有关的多个包，可以yum install perl* ；如果安装报错，检查是否安装mysql-share包
# make install
```

```
[root@vm2 maatkit-7540]# ls bin/    --这些命令，就是各个管理工具
mk-archiver               mk-purge-logs
mk-checksum-filter        mk-query-advisor
mk-config-diff            mk-query-digest
mk-deadlock-logger        mk-query-profiler
mk-duplicate-key-checker  mk-show-grants
mk-error-log              mk-slave-delay
mk-fifo-split             mk-slave-find
mk-find                   mk-slave-move
mk-heartbeat              mk-slave-prefetch
mk-index-usage            mk-slave-restart
mk-kill                   mk-table-checksum
mk-loadavg                mk-table-sync
mk-log-player             mk-table-usage
mk-merge-mqd-results      mk-tcp-model
mk-parallel-dump          mk-upgrade
mk-parallel-restore       mk-variable-advisor
mk-profile-compact        mk-visual-explain
```

--使用--help查看一个命令的使用方法

```
# mk-slave-delay --help  启动和停止slave服务器使其滞后master
mk-slave-delay starts and stops a slave server as needed to make it lag behind
the master.  The SLAVE-HOST and MASTER-HOST use DSN syntax, and values are
copied from the SLAVE-HOST to the MASTER-HOST if omitted.  For more details,
please use the --help option, or try 'perldoc /usr/bin/mk-slave-delay' for
complete documentation.
```

```
# man mk-slave-delay   man文档
```

```
mk-slave-delay --delay 1m --interval 15s --run-time 10m slavehost
--delay：延迟1m复制
--interval：间隔15s去检查slave状态
--quiet：运行时不显示输出信息
--run-time：运行时间，默认一直运行

mysql AB(无论同步或异步)正在运行OK的情况下，使用下面的命令在slave上运行;做之间建议把时间同步一下

# mk-slave-delay --defaults-file=/usr/local/mysql/etc/my.cnf --delay=1m --interval=15s --
user=root --password=123  --quiet localhost &
表示延时1分钟，才会应用SQL线程；这里是测试所以才使用很小的时间，实际情况可以调成1小时或2小时；间隔15s去检测slave是否需要被关闭或者启动


测试：

在master上随便插入几条数据

然后在slave上发现没有马上同步过来
slave > show slave status\G;   --查看状态会发现SQL线程状态为NO

        Slave_IO_Running: Yes
        Slave_SQL_Running: NO


大概等1分钟，就会自动延时同步过来了；
--注意:日志已经传到slave的relay-bin log里了，但由SQL线程延时去解析


问题1：
如果现在一个小公司mysql数据库已经跑了一年，现在要搭建mysqlAB复制，你检查了主库后，发现它这一年都没有使用二进制日志，请问如何做复制？

1、打开主库二进制日志，然后重启
2、全备主库，然后恢复到备库
3、搭建AB复制，指定从备份完成的那个日志position开始复制

问题2：
如果一个lamp架构在深圳机房在运行，如何尽量无影响的把这个lamp迁移到广州的机房

1、在广州那边先搭建lamp，其中web内容可以和深圳的这边做rsync远程实时同步，mysql做双主
2、改DNS的A记录，把IP由深圳的换成广州的
3、过一段时间（等深圳服务器没有连接了），就可以关闭深圳服务器了。
   stop slave;
```

6. 基于GTIDs的复制

6.1 关于GTIDs概述

- **什么是GTIDs以及有什么特点?**

1. GTIDs（Global transaction identifiers）全局事务标识符，是mysql 5.6新加入的一项技术

2. 当使用GTIDs时，每一个事务都可以被识别并且跟踪
3. 添加新的slave或者当发生故障需要将master身份或者角色迁移到slave上时，都无需考虑是哪一个二进制日志以及哪个position值，极大简化了相关操作
4. GTIDs是完全基于事务的，因此不支持MYISAM存储引擎
5. GTID由source_id和transaction_id组成：
   1）source_id来自于server_uuid,可以在auto.cnf中看到
   2）transation_id是一个序列数字，自动生成.

- **使用GTIDs的限制条件有哪些?**

1. 不支持非事务引擎（MYisam），因为可能会导致多个gtid分配给同一个事务
2. create table ... select 语句不支持（主库语法报错）
3. create/drop **temporary table** 语句不支持
4. 必须使用enforce-gtid-consistency参数
5. sql-slave-skip-counter不支持(传统的跳过错误方式)
6. GTID复制环境中必须要求统一开启和GTID或者关闭GTID
7. 在mysql 5.6.7之前，使用mysql_upgrade命令会出现问题

- **GTID的生命周期包含以下部分：**

GTID的生成和生命周期包括以下步骤：

1. 事务在主服务器上执行并提交。
使用主服务器的UUID和此服务器上尚未使用的最小非零事务序列号为此事务分配GTID; GTID被写入主服务器的二进制日志（紧接在日志中的事务本身之前）。

2. 在将二进制日志数据传输到slave并存储在slave的中继日志中之后，slave读取GTID并设置其gtid_next系统的值变量作为这个GTID。这告诉从服务器必须使用此GTID记录下一个事务。

请务必注意，slave设置gtid_next在会话上下文中。

3. slave验证此GTID尚未用于在其自己的二进制日志中记录事务。如果未使用此GTID，则slave设备将写入GTID，应用事务，并将事务写入其二进制日志。通过首先读取和检查事务的GTID，在处理事务本身之前，从设备不仅保证在从设备上没有应用具有此GTID的先前事务，而且还保证没有其他会话已经读取此GTID但尚未提交相关交易。换句话说，不允许多个客户端同时应用同一事务。

4. 因为gtid_next非空，所以slave不会尝试为此事务生成GTID，而是写入存储在此变量中的GTID，即从master服务器获取的GTID，紧接在其二进制日志中的事务之前。

```
1. A transaction is executed and committed on the master.
This transaction is assigned a GTID using the master's UUID and the smallest nonzero transaction
sequence number not yet used on this server; the GTID is written to the master's binary log
(immediately preceding the transaction itself in the log).

2. After the binary log data is transmitted to the slave and stored in the slave's relay log,
the slave reads the GTID and sets the value of its gtid_next system variable as this GTID. This
tells the slave that the next transaction must be logged using this GTID.It is important to note
that the slave sets gtid_next in a session context.
```

3. The slave verifies that this GTID has not already been used to log a transaction in its own binary log. If this GTID has not been used, the slave then writes the GTID, applies the transaction, and writes the transaction to its binary log. By reading and checking the transaction's GTID first, before processing the transaction itself, the slave guarantees not only that no previous transaction having this GTID has been applied on the slave, but also that no other session has already read this GTID but has not yet committed the associated transaction. In other words, multiple clients are not permitted to apply the same transaction concurrently.

4. Because gtid_next is not empty, the slave does not attempt to generate a GTID for this transaction but instead writes the GTID stored in this variable—that is, the GTID obtained from the master—immediately preceding the transaction in its binary log.

总结：有了GTID大大的简化了复制的过程，降低了维护的难度

6.2 基于GTIDs的配置

在生产环境中，大多数情况下使用的MySQL5.6基本上都是从5.5或者更低的版本升级而来，这就意味着之前的mysql replication方案是基于传统的方式部署，并且已经在运行，因此，接下来我们就利用已有的环境升级至基于GITDs的Replication

步骤：
1、将master和slave服务器都设置为read-only
```
mysql>set @@global.read_only=ON;
```
2、停止两台服务器的mysql服务
3、开启GTIDs
注意：
1、开启GITDs需要在master和slave上都配置gtid-mode, log-bin, log-slave-updates, enforce-gtid-consistency（该参数在5.6.9之前是--disable-gtid-unsafe-statement）
2、其次，slave还需要增加skip-slave-start参数,目的是启动的时候，先不要把slave起来，需要做一些配置
master:
```
[mysqld]
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data
port = 3307
socket = /usr/local/mysql/mysql.sock
log-error = /usr/local/mysql/data/mysql3307.err
log-bin = /usr/local/mysql/data/mybinlog
server_id = 10
gtid-mode=on
log-slave-updates
enforce-gtid-consistency
[client]
socket=/usr/local/mysql/mysql.sock
```

slave:
```
[mysqld]
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data
port = 3307
socket = /usr/local/mysql/mysql.sock
```

```
log-error = /usr/local/mysql/data/mysql3307.err
log-bin = /usr/local/mysql/data/mybinlog
server_id = 20
relay_log = /usr/local/mysql/data/relay
log-slave-updates=1
gtid-mode=on
enforce-gtid-consistency
skip-slave-start

[client]
socket=/usr/local/mysql/mysql.sock
```

4、启动master和slave的数据库

```
[root@master ~]# service mysql start
Starting MySQL.. SUCCESS!
[root@slave1 ~]# service mysql start
Starting MySQL.. SUCCESS!
```

5. 重新配置slave

1）确保master上有授权用户

2）在slave上配置同步

```
mysql> change master to
master_host='10.1.1.5',master_user='slave',master_password='123',master_port=3307,master_auto_po
sition=1;

master_auto_position=1 //关键点和传统复制模式不一样

mysql> start slave;
```

5、关闭read-only模式

```
mysql> set @@global.read_only=OFF;
```

补充：基于GTIDs复制手动跳过事务

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';      指定需要跳过的GTIDs编号
BEGIN;      开始一个空事务
COMMIT;
SET GTID_NEXT='AUTOMATIC';     使用下一个自动生成的全局事务ID。


mysql> stop slave;
Query OK, 0 rows affected (0.00 sec)

mysql> SET @@SESSION.GTID_NEXT= '044e6392-cf9b-11e8-a748-000c294ca304:3';
Query OK, 0 rows affected (0.00 sec)
```

说明：需要跳过哪个事务，需要手动查看relaylog文件得到

```
[root@slave1 data]# ../bin/mysqlbinlog relay.000003|less
。。。。
# at 756
#181015 12:04:45 server id 10  end_log_pos 817 CRC32 0x5374f49e         GTID [commit=yes]
SET @@SESSION.GTID_NEXT= '044e6392-cf9b-11e8-a748-000c294ca304:3'/*!*/;
```

```
mysql> BEGIN;
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> SET @@SESSION.GTID_NEXT= 'AUTOMATIC';
Query OK, 0 rows affected (0.00 sec)

mysql> start slave;
Query OK, 0 rows affected (0.36 sec)

mysql> show slave status\G
```

7. 其他

- 跟复制相关的文件

```
master.info：用于保存slave连接至master的相关信息，包括服务器地址，用户名，密码等
relay-log.info：保存在当前的slave节点上已经复制的当前二进制日志和本地relay log日志的对应关系
```

- 复制的监控和维护

```
purge 命令
    mysql> purge binary logs to 'mysql-bin.000008';
    指明这个二进制文件之前的所有的文件都会被清理

    mysql> purge binary logs before '2017-04-23 20:46:26';
    指明二进制日志文件中这个时间点之前的所有的事件都会被清理

监控操作
    mysql> show master status; 显示主节点正在写哪个二进制日志文件
    mysql> show binlog events; 显示在二进制文件中记录了哪些信息
    mysql> show binary logs ; 显示记录了哪些二进制日志文件
    mysql> show slave status; 显示从服务器的状态
    mysql> show processlist; 显示当前启用的线程
```

课后实战