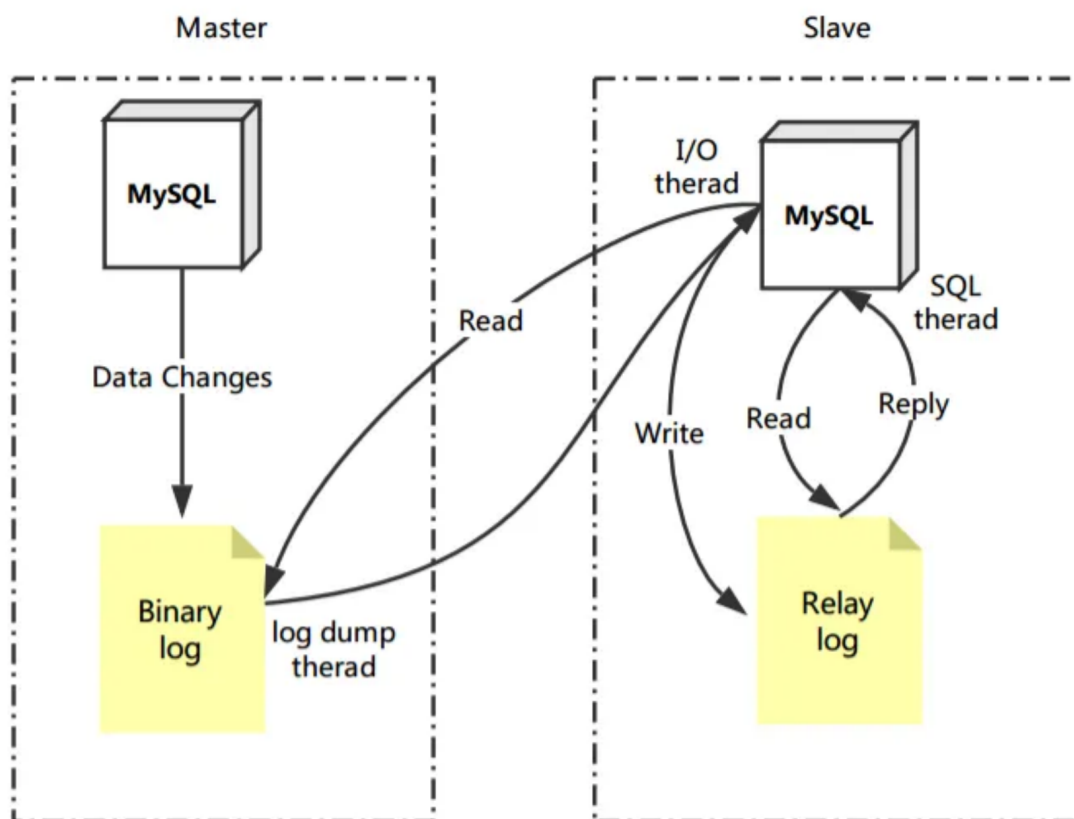


# MySQL集群篇

## 一、集群搭建之主从复制

### 1.1 主从复制原理



- (1) Master 将数据改变记录到二进制日志(binary log)中, 也就是配置文件 log-bin 指定的文件, 这些记录叫做二进制日志事件(binary log events);
- (2) Slave 通过 I/O 线程读取 Master 中的 binary log events 并写入到它的中继日志(relay log);
- (3) Slave 重做中继日志中的事件, 把中继日志中的事件信息一条一条的在本地执行一次, 完成数据在本地的存储, 从而实现将改变反映到它自己的数据(数据重放)。

#### 注意事项:

- 主从服务器操作系统版本和位数一致;
- Master 和 Slave 数据库的版本要一致;
- Master 和 Slave 数据库中的数据要一致;
- Master 开启二进制日志, Master 和 Slave 的 server\_id 在局域网内必须唯一;

### 1.2 binlog和relay日志

- **bin log:** 将数据改变记录到二进制日志(binary log)中, 可用于本机数据恢复和主从同步。
- **relay (中继) log:** 重做中继日志。Slave节点会把中继日志中的事件信息一条一条的在本地执行一次, 实现主从同步, 这个过程也叫数据重放。

## 1.2.1 binlog的三种模式

### 1. ROW模式

日志中会记录成每一行数据被修改的形式，然后在slave端再对相同的数据进行修改

- **优点：**bin-log中可以**不记录执行的sql语句的上下文相关的信息**，仅仅只需要记录那一条记录被修改了，修改成什么样了。所以ROW模式的日志的内容会非常清楚的记录下每一行数据修改的细节。而且不会出现某些特定情况下的**存储过程或函数**，以及**触发器**的调用和触发无法被正确复制的问题。
- **缺点：**ROW模式下，所有的执行的语句当记录到日志中的时候，都将以每行记录的修改记录，这样会产生大量的日志内容。
  - 比如：有这样一条update语句：update product set owner\_member\_id='d' where owner\_member\_id='a'，执行之后，日志中记录的**不是**这条update语句所对应的事件(mysql是以事件的形式来记录bin-log日志)，而是这条语句所更新的每一条记录的变化情况，这样就记录成很多条记录被更新的很多事件，自然bin-log日志的量会很大。

### 2. Statement模式

**Statement模式：**每一条会修改数据的sql都会记录到master的bin-log中。slave在复制的时候sql进程会解析成和原来master端执行过的相同的sql来再次执行。

- **优点：**Statement模式下的优点，首先就是**解决了ROW模式下的缺点**，不需要记录每一行数据的变化，减少bin-log日志量，节约io，提高性能。因为他**只需要记录在master上所执行的语句的细节，以及执行语句时候的上下文的信息**。
- **缺点：**由于它是记录的执行语句，所以为了让这些语句在slave端也能正确执行，那么他还必须记录每条语句在执行的时候的一些相关信息，也就是上下文信息，以保证所有语句在slave端被执行的时候能够得到和在master端执行时候相同的结果。
  - 另外，由于mysql现在发展比较快，很多新功能加入，使mysql的复制遇到了不小的挑战，自然复制的时候涉及到越复杂的内容，bug也就越容易出现。在Statement模式下，目前已经发现的就有不少情况会造成mysql的复制问题，主要是修改数据的时候使用了某些特定的函数或者功能的时候会出现。比如：sleep()在有些版本就不能正确复制。

### 3. Mixed模式

- 实际上就是**前两种模式的结合**，在Mixed模式下，MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志形式，也就是在**Statement和Row**之间选一种。
- 新版本中的Statement模式还是和以前一样，仅仅记录执行的语句。而新版本的MySQL中对ROW模式被做了优化，并不是所有的修改都会以ROW模式来记录，像遇到表结构变更的时候就会以Statement模式来记录，如果sql语句确实就是Update或者Delete等修改数据的语句，那么还是会记录所有行的变更。

## 1.2.2 开启binlog

修改my.cnf文件

在[mysqld]段下添加：

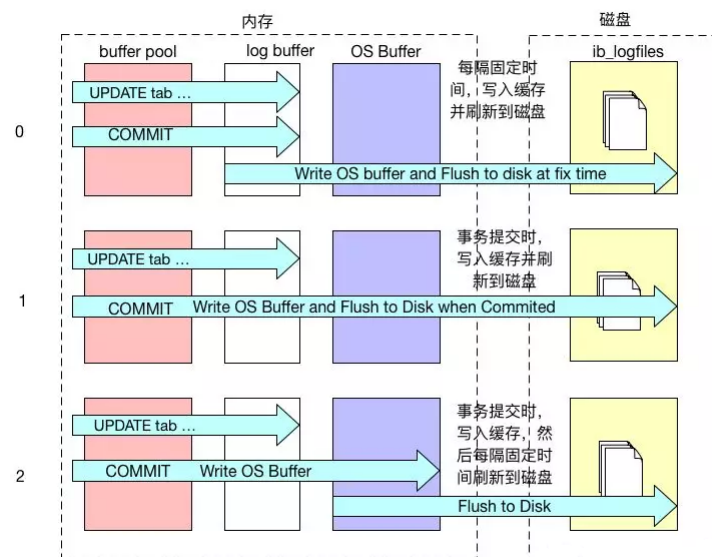
```
1 #binlog刷盘策略
2 sync_binlog=1
3 #需要备份的数据库
4 binlog-do-db=hello
5 #不需要备份的数据库
6 binlog-ignore-db=mysql
7 #启动二进制文件
8 log-bin=mysql-bin
9 #服务器ID
10 server-id=132
```

**sync\_binlog参数：**

- **0**：存储引擎不进行binlog的刷新到磁盘，而由操作系统的文件系统控制缓存刷新。
- **1**：每提交一次事务，存储引擎调用文件系统的sync操作进行一次缓存的刷新，这种方式最安全，但性能较低。
- **n**：当提交的日志组=n时，存储引擎调用文件系统的sync操作进行一次缓存的刷新。

sync\_binlog=0或sync\_binlog大于1，事务被提交，而尚未同步到磁盘。因此，在电源故障或操作系统崩溃时有可能服务器已承诺尚未同步一些事务到二进制日志。因此它是不可能执行例行程序恢复这些事务，他们将会丢失二进制日志。

类似redo log刷盘机制，如下图：



## 1.2.3 调整binlog日志模式

查看binlog的日志模式：

```

1 mysql> show variables like 'binlog_format';
2 +-----+-----+
3 | variable_name | value |
4 +-----+-----+
5 | binlog_format | ROW   |
6 +-----+-----+
7 1 row in set (0.00 sec)

```

调整binlog的日志模式：binlog的三种格式：STATEMENT、ROW、MIXED。

```

1 mysql> set binlog_format=STATEMENT;
2 Query OK, 0 rows affected (0.00 sec)
3
4 mysql> show variables like 'binlog_format';
5 +-----+-----+
6 | variable_name | value |
7 +-----+-----+
8 | binlog_format | STATEMENT |
9 +-----+-----+
10 1 row in set (0.00 sec)

```

### 1.2.4 查看bin log和relay log日志

因为binlog日志文件：mysql-bin.000005是二进制文件，没法用vi等打开，这时就需要mysql的自带的mysqlbinlog工具进行解码，执行：mysqlbinlog mysql-bin.000005 可以将二进制文件转为可读的sql语句。

```

1 mysqlbinlog --base64-output=decode-rows -v -v mysql-bin.000058 > binlog

```

### 1.2.5 使用命令查看binlog

show master logs，查看所有二进制日志列表，和 show binary logs 同义。

```

1 mysql> show master logs;
2 +-----+-----+
3 | Log_name      | File_size |
4 +-----+-----+
5 | mysql-bin.000001 | 385 |
6 +-----+-----+
7 1 row in set (0.00 sec)

```

使用 show binlog events 命令可以以列表的形式显示日志中的事件信息。

show binlog events命令的格式：

```

1 show binlog events [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count];

```

说明：

- (1) IN 'log\_name': 指定要查询的binlog文件名 (如果省略此参数, 则默认指定第一个binlog文件) ;
- (2) FROM pos: 指定从哪个pos起始点开始查起 (如果省略此参数, 则从整个文件的第一个pos点开始算) ;
- (3) LIMIT 【offset】: 偏移量 (默认为0) ;
- (4) row\_count: 查询总条数 (如果省略, 则显示所有行) 。

```
1 mysql> show binlog events in 'mysql-bin.000001';
2 +-----+-----+-----+-----+-----+-----+
3 | Log_name      | Pos | Event_type      | Server_id | End_log_pos | Info
4 +-----+-----+-----+-----+-----+-----+
5 | mybin.000001 | 4   | Format_desc     | 132       | 123         | Server ver:
6 | mybin.000001 | 123 | Previous_gtid   | 132       | 154         |
7 +-----+-----+-----+-----+-----+-----+
8 2 rows in set (0.00 sec)
```

切换binlog文件:

```
1 mysql> flush logs;
2 Query OK, 0 rows affected (0.00 sec)
```

注意: 刷新日志会生成一个新的日志文件

## 1.3 基于binlog主从复制

### 1.3.1 关闭主从机器的防火墙

安全起见, 云服务器的防火墙不应关闭, 需要将3306端口放行。

```
1 systemctl stop iptables (需要安装iptables服务)
2 systemctl stop firewalld (默认)
3 systemctl disable firewalld.service (设置开启不启动)
```

### 1.3.2 主服务器配置

查看binlog是否开启可以使用命令:

```

1 | mysql> show variables like 'log_bin%';
2 | +-----+-----+
3 | | variable_name | value |
4 | +-----+-----+
5 | | log_bin | OFF |
6 | | log_bin_basename | |
7 | | log_bin_index | |
8 | | log_bin_trust_function_creators | OFF |
9 | | log_bin_use_v1_row_events | OFF |
10 | +-----+-----+
11 | 5 rows in set (0.12 sec)

```

log\_bin如果是 OFF 代表是未开启状态。

```

mysql> show variables like 'log_bin%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| log_bin | ON |
| log_bin_basename | /var/lib/mysql/mysql-bin |
| log_bin_index | /var/lib/mysql/mysql-bin.index |
| log_bin_trust_function_creators | OFF |
| log_bin_use_v1_row_events | OFF |
+-----+-----+
5 rows in set (0.00 sec)

```

### 第一步：修改my.cnf文件

在[mysqld]段下添加：

```

1 | #binlog刷盘策略
2 | sync_binlog=1
3 | #需要备份的数据库
4 | binlog-do-db=hello
5 | #不需要备份的数据库
6 | binlog-ignore-db=mysql
7 | #启动二进制文件
8 | log-bin=mysql-bin
9 | #服务器ID
10 | server-id=132

```

### 第二步：重启mysql服务

```
1 | systemctl restart mysqld
```

。

### 第三步：主机给从机授备份权限

注意：先要登录到MySQL命令客户端

```

1 | mysql> GRANT REPLICATION SLAVE ON *.* TO '从机MySQL用户名'@'从机IP' identified
  | by '从机MySQL密码';

```

示例：

```
1 | GRANT REPLICATION SLAVE ON *.* TO 'root'@'%' identified by 'root';
```

#### 注意事项:

- 1 | 一般不用root帐号，“%”表示所有客户端都可能连，只要帐号，密码正确，此处可用具体客户端IP代替，如39.99.131.178，加强安全。

mysql5.7对密码的强度是有要求的，必须是字母+数字+符号组成的，可以使用如下方法调整密码强度

设置密码长度最低位数

```
mysql> set global validate_password_length=4;
```

设置密码强度级别

```
mysql> set global validate_password_policy=0;
```

validate\_password\_policy有以下取值:

Policy	Tests Performe
0 or LOW	Length
1 or MEDIUM	numeric, lowercase/uppercase, and special characters
2 or STRONG	Length; numeric, lowercase/uppercase, and special characters

默认是1，即MEDIUM，所以刚开始设置的密码必须符合长度，且必须含有数字，小写或大写字母，特殊字符。

#### 第四步：刷新权限

```
1 | mysql> FLUSH PRIVILEGES;
```

#### 第五步：查询master的状态

```
1 | mysql> show master status;
2 |
3 | +-----+-----+-----+-----+-----+
4 | | File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
5 | +-----+-----+-----+-----+-----+
6 | | mysql-bin.000002 | 593      | hello        | mysql              |                    |
7 | +-----+-----+-----+-----+-----+
8 | 1 row in set
```

### 1.3.3 从服务器配置

#### 第一步：修改my.conf文件

```
1 [mysqld]
2 server-id=133
```

#### 第二步：重启mysql服务

```
1 systemctl restart mysqld
```

如果出现此错误：

```
1 Fatal error: The slave I/O thread stops because master and slave have equal
MySQL server UIDs; these UUIDs must be different for replication to work.
```

原因：

```
1 因为mysql是克隆的系统所以mysql的uuid是一样的，所以需要修改。
```

解决方法：

```
1 删除/var/lib/mysql/auto.cnf文件，重新启动MySQL服务。
```

#### 第三步：重启并登录到MySQL进行配置从服务器

```
1 mysql>change master to
2 master_host='39.103.222.177',
3 master_port=3306,
4 master_user='root',
5 master_password='root',
6 master_log_file='mysql-bin.000002',
7 master_log_pos=593,
8 MASTER_AUTO_POSITION=0;
```

注意：

语句中间不要断开，`master_port` 为mysql服务器端口号(无引号)，`master_user` 为执行同步操作的数据库账户，“593”无单引号(此处的593就是 `show master status` 中看到的 `position` 的值，这里的 `mysql-bin.000002` 就是 `file` 对应的值)。

#### 第四步：启动从服务器复制功能

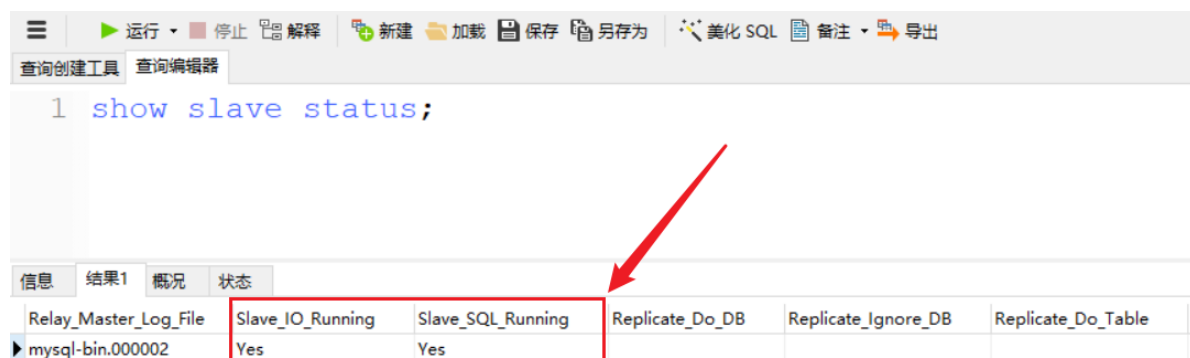
```
1 mysql>start slave;
```



## 第五步：检查从服务器复制功能状态

```
1 mysql> show slave status;
2 .....(省略部分)
3 Slave_IO_Running: Yes //此状态必须YES
4 Slave_SQL_Running: Yes //此状态必须YES
5 .....(省略部分)
```

注：Slave\_IO及Slave\_SQL进程必须正常运行，即YES状态，否则都是错误的状态(如：其中一个NO均属错误)。



The screenshot shows a MySQL management interface with a toolbar at the top containing icons for running, stopping, explaining, creating, loading, saving, and exporting. Below the toolbar, the command 'show slave status;' is entered in the query editor. The results are displayed in a table with columns: Relay\_Master\_Log\_File, Slave\_IO\_Running, Slave\_SQL\_Running, Replicate\_Do\_DB, Replicate\_Ignore\_DB, and Replicate\_Do\_Table. The first row of data shows 'mysql-bin.000002' for the log file, and 'Yes' for both 'Slave\_IO\_Running' and 'Slave\_SQL\_Running'. These two 'Yes' values are enclosed in a red rectangular box, and a red arrow points from the top right towards this box.

信息	结果1	概况	状态			
Relay_Master_Log_File	Slave_IO_Running	Slave_SQL_Running	Replicate_Do_DB	Replicate_Ignore_DB	Replicate_Do_Table	F
mysql-bin.000002	Yes	Yes				

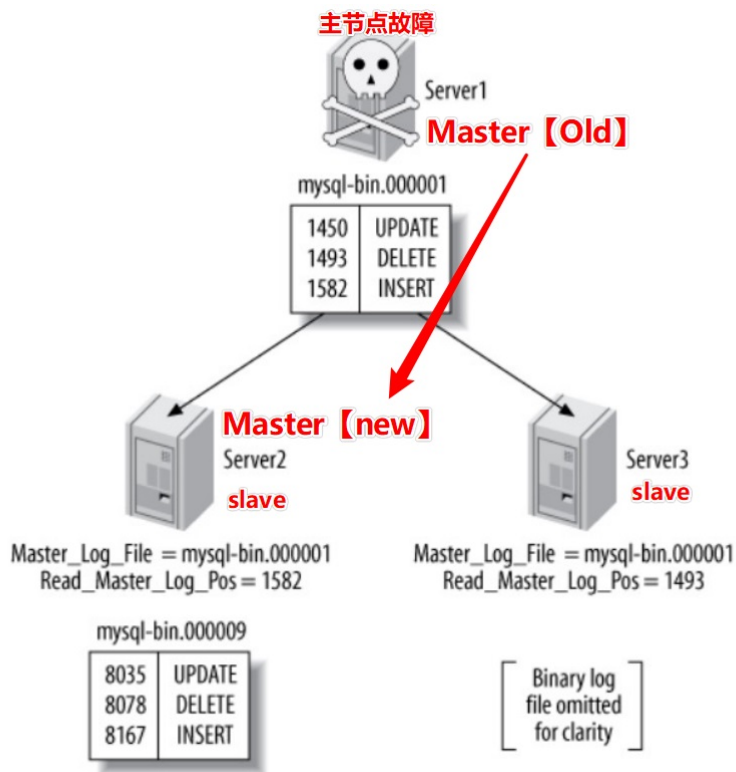
以上操作过程，从服务器配置完成。

### 1.3.4 测试

搭建成功之后，往主机中插入数据，看看从机中是否有数据

#### 1.3.4 主从同步模式的潜在问题：如何确定master\_log\_pos?

```
1 mysql> change master to
2 master_host='39.103.222.177',
3 master_port=3306,
4 master_user='root',
5 master_password='root',
6 master_log_file='mysql-bin.000002',
7 master_log_pos=593,
8 MASTER_AUTO_POSITION=0;
```



## 1.4 基于GTID的主从复制

### 1.4.1 什么是GTID?

GTID即全局事务ID (global transaction identifier), 其保证为每一个在主上提交的事务在复制集群中可以生成一个唯一的ID。GTID最初由google实现, 官方MySQL在5.6才加入该功能。**MySQL主从结构在一主一从情况下对于GTID来说就没有优势了, 而对于2台主以上的结构优势异常明显, 可以在数据不丢失的情况下切换新主。**

GTID实际上是由UUID+TID (即transactionId)组成的。其中UUID(即server\_uuid)产生于auto.cnf文件, 是一个MySQL实例的唯一标识。TID代表了该实例上已经提交的事务数量, 并且随着事务提交单调递增, 所以GTID能够保证每个MySQL实例事务的执行。**GTID在一组复制中, 全局唯一。**通过GTID的UUID可以知道这个事务在哪个实例上提交的。

**GTID可以保证不会重复执行同一个事务, 并且会补全没有执行的事务;**

```
1 | cat /var/lib/mysql/auto.cnf
```

```
[root@k8s-master01 ~]# cat /var/lib/mysql/auto.cnf
[auto]
server-uuid=79633990-a991-11ec-a07c-00163e0b0f0a
```

下面是一个GTID的具体形式:

```
1 | mysql> show master status;
2 | +-----+-----+-----+-----+-----+
3 | File      | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set
4 | +-----+-----+-----+-----+-----+
   |
```

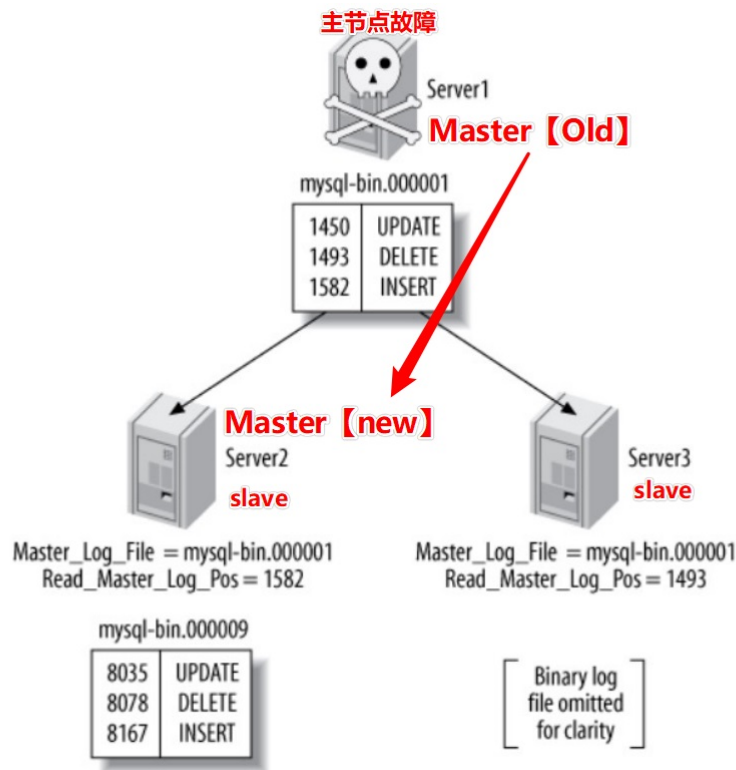
```

5 | bin.000004 | 1147 | hello | mysql | 79633990-a991-11ec-a07c-00163e0b0f0a:1-57 |
6 +-----+-----+-----+-----+-----+
7 1 row in set (0.00 sec)
8
9 GTID:79633990-a991-11ec-a07c-00163e0b0f0a:1-57
10 UUID:79633990-a991-11ec-a07c-00163e0b0f0a
11 transactionId:1-57
12
13 在整个复制架构中GTID 是不变化的,即使在多个连环主从也不会变。
14
15 例如: ServerA ----> ServerB ----> ServerC
16 GTID从在ServerA、ServerB、ServerC 中都是一样的。

```

### 1.4.2 GTID解决了什么问题?

通过GTID可以很方便的进行复制结构上的**故障转移**, 新主设置, 这就很好地解决了下面这个图所展现出来的问题。



#### 如果没有GTID:

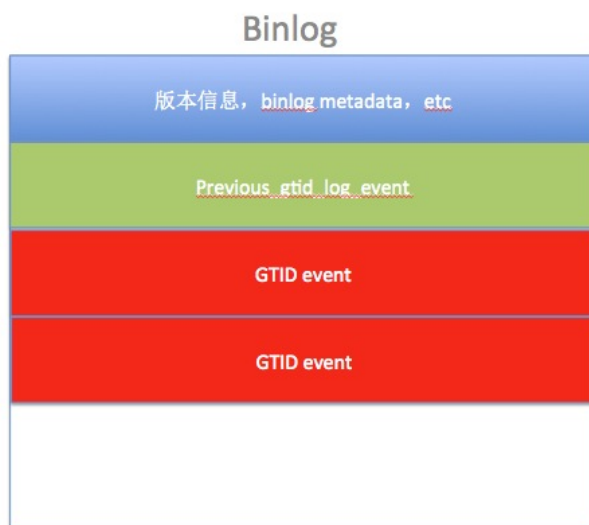
- Server1(Master)崩溃: 根据从上show slave status获得Master\_log\_File/Read\_Master\_Log\_Pos的值来看
  - Server1(master)最新pos是1582
  - Server2(Slave)是1582, 已经跟上了主
  - Server3(Slave)是1493, 没有跟上主
- 这时要是把Server2提升为主, Server3变成Server2的从。
- 如果没有全局事务ID, 在Server3上执行change的时候就需要做一些计算, 保证之前的事务都执行完毕了! 进行主节点故障转移的时候就比较繁琐

#### 如果使用GTID:

- 这个问题在5.6的GTID出现后，就显得非常的简单。
- 由于同一事务的GTID在所有节点上的值一致，那么根据Server3当前停止点的GTID就能定位到Server2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命令就可以直接完成failover的工作。

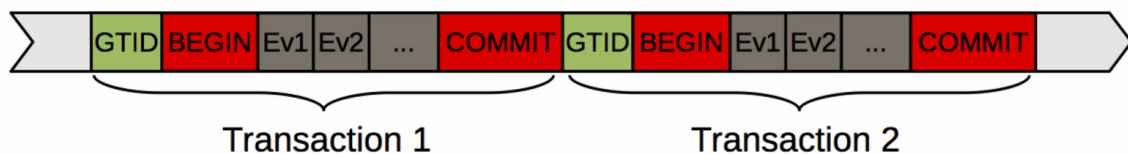
**使用GTID需要注意:** 在构建主从复制之前，在一台将成为主的实例上进行一些操作（如数据清理等），通过GTID复制，这些在主从成立之前的操作也会被复制到从服务器上，引起复制失败。也就是说通过GTID复制都是从最先开始的事务日志开始，即使这些操作在复制之前执行。比如在server1上执行一些drop、delete的清理操作，接着在server2上执行change的操作，会使得server2也进行server1的清理操作。

### 1.4.3 GTID在binlog的结构:



**Previous\_gtid\_log\_event:** Previous\_gtid\_log\_event 在每个binlog 头部都会有。

**GTID event 结构:**



### 1.4.4 GTID和Binlog之间的关系是?

```

1  * 假设有4个binlog: bin.001,bin.002,bin.003,bin.004
2  * bin.001 : Previous-GTIDS=empty; binlog_event有: 1-40
3  * bin.002 : Previous-GTIDS=1-40; binlog_event有: 41-80
4  * bin.003 : Previous-GTIDS=1-80; binlog_event有: 81-120
5  * bin.004 : Previous-GTIDS=1-120; binlog_event有: 121-160
6
7  1. 假设现在我们要找GTID=$A, 那么MySQL的扫描顺序为: 从最后一个binlog开始扫描(即:
   bin.004)
8  2. bin.004的Previous-GTIDS=1-120, 如果$A=140 > Previous-GTIDS,那么肯定在bin.004
   中
9  3. bin.004的Previous-GTIDS=1-120, 如果$A=88 包含在Previous-GTIDS中,那么继续对比上一
   个binlog文件 bin.003,然后再循环前面2个步骤,直到找到为止

```

## 1.4.5 配置GTID主从复制

### 1. 修改master、slave服务器的my.cnf文件

```

1  #开启GTID模式(必选)
2  gtid_mode=ON
3  #强制gtid一致性(必选)
4  enforce-gtid-consistency=true

```

### 2. 重启mysql

```

1  [root@mysql132 ~]# systemctl restart mysqld

```

### 3. 从服务器中执行change master

```

1  mysql> change master to
2  master_host='39.103.222.177',
3  master_port=3306,
4  master_user='root',
5  master_password='root',
6  master_auto_position = 1;

```

### 4. 开启同步

```

1  START SLAVE;

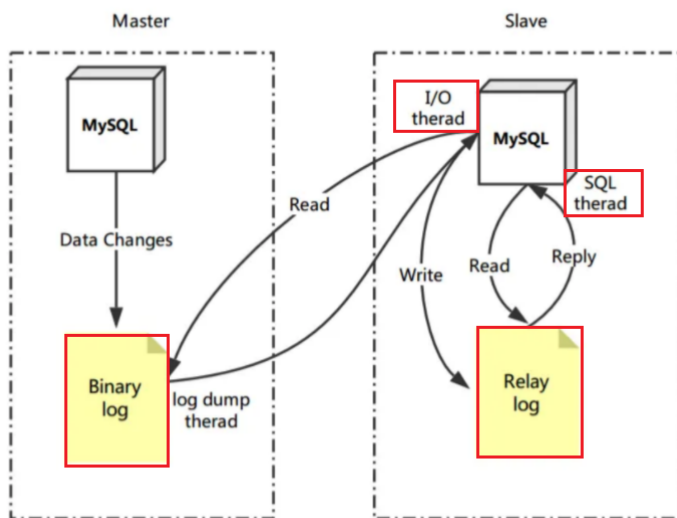
```

## 1.5 主从同步延迟的原因及解决办法

mysql 用主从同步的方法进行读写分离减轻主服务器的压力, 在业内这种做法非常普遍。主从同步基本上能做到实时, 但是偶尔也会出现主从同步延迟, 主要原因如下:

## 1.5.1 原因

我们知道，一个服务器开放N个链接给客户端来连接的，这样会有大并发的更新操作，但是从服务器的里面读取binlog的线程仅有一个，当某个SQL在从服务器上执行的时间稍长或者由于某个SQL要进行锁表就会导致，主服务器的SQL大量积压，未被同步到从服务器里。这就导致了主从不一致，也就是主从延迟。



在配置好了，主从同步以后，主服务器会把更新语句写入binlog，从服务器的IO线程回去读取主服务器的binlog，并且写到从服务器的Relay log里面，然后从服务器的SQL thread会一个一个执行relay log里面的sql，进行数据恢复。

注意：5.6.3之前的IO线程仅有一个。5.6.3之后有多线程去读，速度会大幅提升，所以主从延迟也会相对少一些

## 1.5.2 解决办法

实际上主从同步延迟根本没有什么一招制敌的办法，因为所有的SQL必须都要在从服务器里面执行一遍，但是主服务器如果不断的有更新操作源源不断的写入，那么一旦有延迟产生，那么延迟加重的可能性就会原来越大。

当然我们可以做一些缓解的措施。

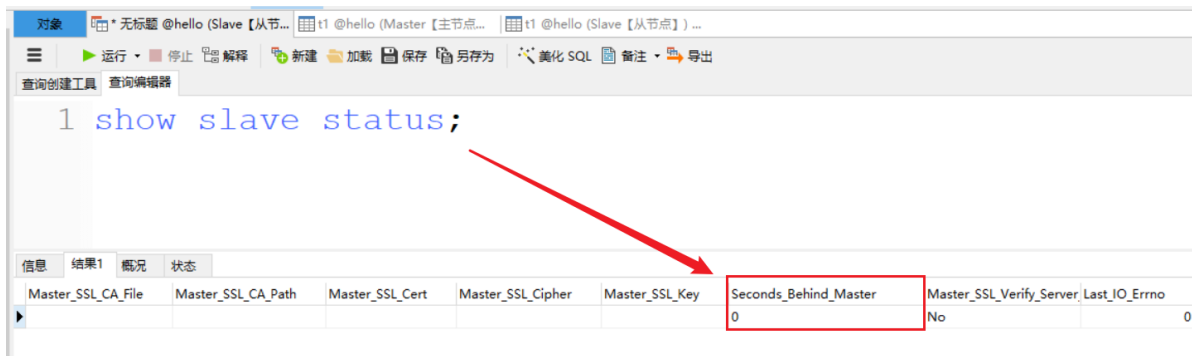
1. 分库，将主库拆分成几个表，那么主库的写并发就会降低很多，主从延迟可以忽略不计
2. 打开mysql的并发复制，如果单个库的写并发很高
3. 重写代码，插入一条数据以后，尽量不要马上去查数据，插入数据直接去更新，不要查询
4. 如果确实存在这样的业务，必须插入数据后马上查询到，对这个查询设置直连主库

主从延迟确实比较高，建议拆库拆表！

## 1.5.3 判断延迟的方法

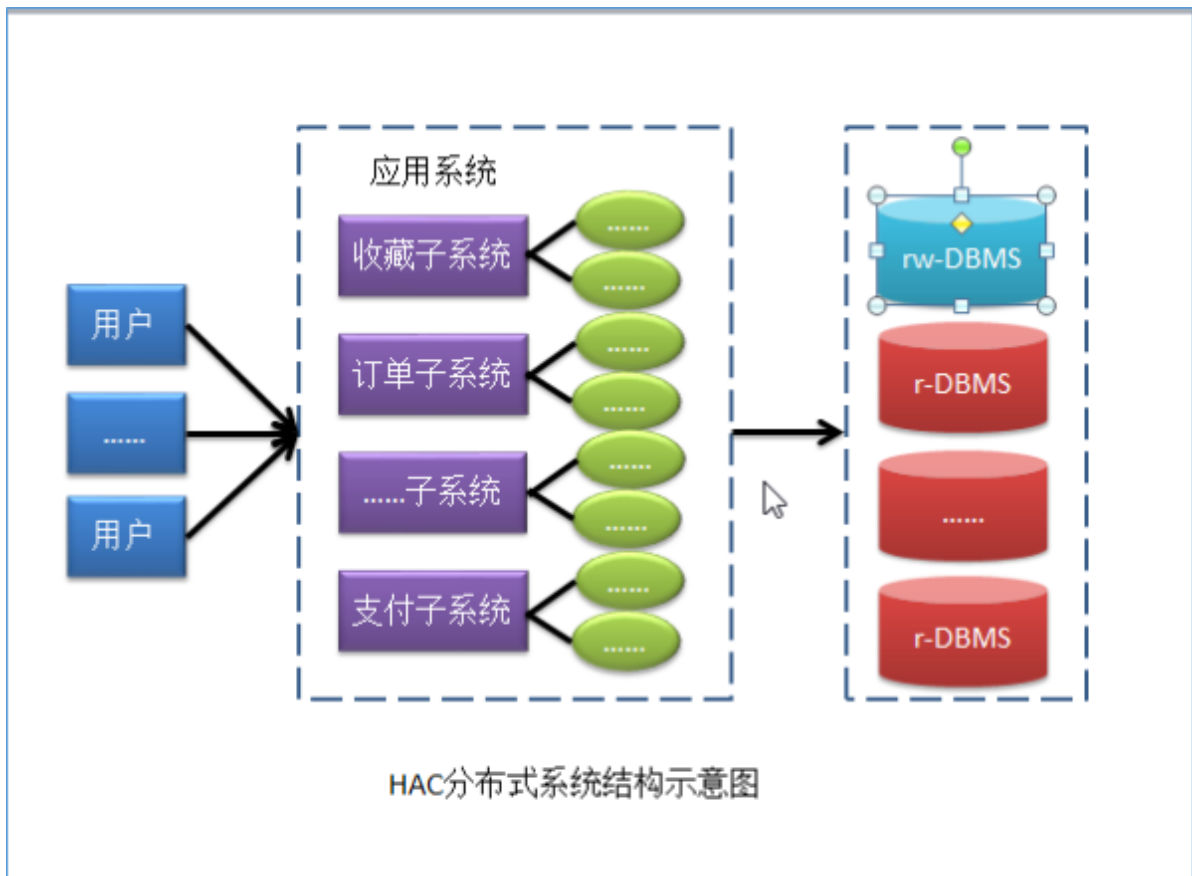
MySQL提供了从服务器状态命令，可以通过 `show slave status` 进行查看，比如可以看看 [Seconds Behind Master](#) 参数的值来判断，是否有发生主从延时。其值有这么几种：

- NULL：表示io\_thread或是sql\_thread有任何一个发生故障，也就是该线程的Running状态是No，而非Yes。
- 0：该值为零，表示主从复制状态正常



## 二、集群搭建之读写分离

### 2.1 读写分离的理解



名词解释:

- HAC: High Availability Cluster, 高可用集群

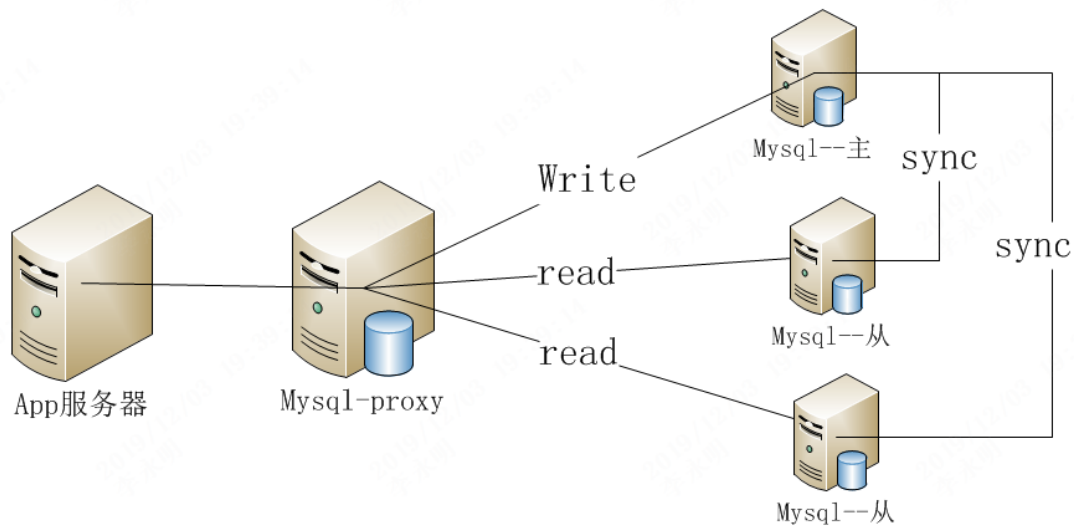
**注意事项:** MySQL的主从复制, 只会保证主机对外提供服务, 而从机是不对外提供服务的, 只是在后台为主机进行备份。

### 2.2 读写分离演示需求

- MySQL master: server-id=132
- MySQL slave: server-id=133

## 2.3 MySQL-Proxy

mysql-proxy是mysql官方提供的mysql中间件服务，提供读写分离的功能。



MySQLProxy虽然可以实现读写分离的操作，但是MySQLProxy官方并没有推出稳定版，其中的坑还是挺多的，并不推荐在生产环境使用。官方推荐使用MySQLRouter，所以关于MySQLProxy的使用大家做了解内容即可。

## 2.4 Atlas

### 2.4.1 简介

Atlas是由 Qihoo 360公司Web平台部基础架构团队开发维护的一个基于MySQL协议的数据中间层项目。它在MySQL官方推出的MySQL-Proxy 0.8.2版本的基础上，修改了大量bug，添加了很多功能特性。目前该项目在360公司内部得到了广泛应用，很多MySQL业务已经接入了Atlas平台，每天承载的读写请求数达几十亿条。同时，有超过50家公司在生产环境中部署了Atlas。



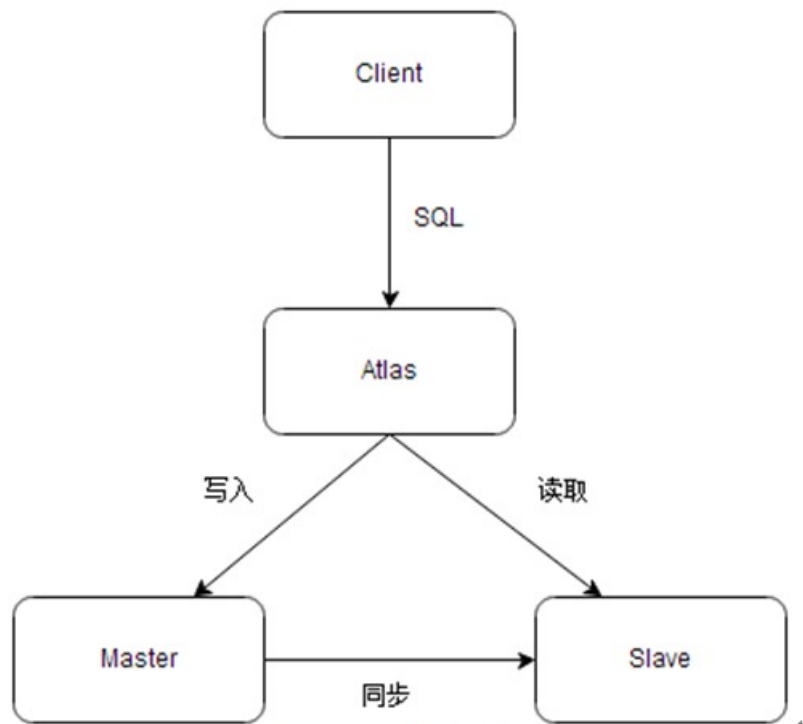


图 2 Atlas 总体架构

官方文档:

[https://github.com/Qihoo360/Atlas/blob/master/README\\_ZH.md](https://github.com/Qihoo360/Atlas/blob/master/README_ZH.md)

## 2.4.2 下载安装

```
1 | > wget https://github.com/Qihoo360/Atlas/releases/download/2.2.1/Atlas-2.2.1.el6.x86_64.rpm
```

下载好了之后, 进行安装

```
1 | > rpm -ivh Atlas-2.2.1.el6.x86_64.rpm
```

安装好了, 它会默认在"/usr/local/mysql-proxy"下给你生成4个文件夹, 以及需要配置的文件, 如下:

```
[root@k8s-master01 mysql-proxy]# pwd
/usr/local/mysql-proxy
[root@k8s-master01 mysql-proxy]# ll
总用量 16
drwxr-xr-x 2 root root 4096 3月 24 14:51 bin
drwxr-xr-x 2 root root 4096 3月 24 14:51 conf
drwxr-xr-x 3 root root 4096 3月 24 14:51 lib
drwxr-xr-x 2 root root 4096 12月 17 2014 log
```

- **bin目录:** 下放的都是可执行文件
  - "encrypt"是用来生成MySQL密码加密的, 在配置的时候会用到
  - "mysql-proxy"是MySQL自己的读写分离代理
  - "mysql-proxyd"是360弄出来的, 后面有个"d", 服务的启动、重启、停止。都是用他来执行的
- **conf目录:** 下放的是配置文件
  - "test.cnf"只有一个文件, 用来配置代理的, 可以使用vim来编辑
- **lib目录:** 下放的是一些包, 以及Atlas的依赖

- **log目录**: 下放的是日志, 如报错等错误信息的记录

### 2.4.3 配置

进入bin目录, 使用encrypt来对数据库的密码进行加密, 我的MySQL数据的用户名是root, 密码是root, 我需要对密码进行加密

```
1 [root@localhost bin]# ./encrypt root
2 DAJn18cvzy8=
```

配置Atlas, 使用vim进行编辑

```
1 [root@localhost conf]# cd /usr/local/mysql-proxy/conf/
2 [root@localhost conf]# vim test.cnf
```

进入后, 可以在Atlas进行配置, 360写的中文注释都很详细, 根据注释来配置信息, 其中比较重要, 需要说明的配置如下:

这是用来登录到Atlas的管理员的账号与密码, 与之对应的是“#Atlas监听的管理接口IP和端口”, 也就是说需要设置管理员登录的端口, 才能进入管理员界面, 默认端口是2345, 也可以指定IP登录, 指定IP后, 其他的IP无法访问管理员的命令界面。方便测试, 我这里没有指定IP和端口登录。

```
1 #管理接口的用户名
2 admin-username = admin
3 #管理接口的密码
4 admin-password = admin
```

这是用来配置主数据的地址与从数据库的地址, 这里配置的主数据库是132, 从数据库是133

```
1 #Atlas后端连接的MySQL主库的IP和端口, 可设置多项, 用逗号分隔
2 proxy-backend-addresses = 39.103.222.177:3306
3
4 #Atlas后端连接的MySQL从库的IP和端口, @后面的数字代表权重, 用来作负载均衡, 若省略则默
  认为1, 可设置多项, 用逗号分隔
5 proxy-read-only-backend-addresses = 39.99.131.178:3306@1
6 # proxy-read-only-backend-addresses =
  39.99.131.178:3306@1,39.103.182.34:3306@2
```

这个是用来配置MySQL的账户与密码的, 我的MySQL的用户是buck,密码是hello,刚刚使用Atlas提供的工具生成了对应的加密密码

```
1 #用户名与其对应的加密过的MySQL密码, 密码使用PREFIX/bin目录下的加密程序encrypt加
  密, 下行的user1和user2为示例, 将其替换为你的MySQL的用户名和加密密码!
2 pwds = root:DAJn18cvzy8=
```

这是设置工作接口与管理接口的, 如果ip设置的“0.0.0.0”就是说任意IP都可以访问这个接口, 当然也可以指定IP和端口, 方便测试我这边没有指定, 工作接口的用户名密码与MySQL的账户对应的, 管理员的用户密码与上面配置的管理员的用户密码对应。

```

1 #Atlas监听的工作接口IP和端口
2 proxy-address = 0.0.0.0:1234
3
4 #Atlas监听的管理接口IP和端口
5 admin-address = 0.0.0.0:2345

```

## 2.4.4 启动Atlas

```

1 [root@localhost bin]# ./mysql-proxyd test start
2 OK: MySQL-Proxy of test is started

```

使用如下命令，进入Atlas的管理模式 `mysql -h127.0.0.1 -P2345 -uuser -ppwd`，能进去说明Atlas正常运行，因为它会把自己当成一个MySQL数据库，所以在不需要数据库环境的情况下，也可以进入到MySQL数据库模式。

```

1 [root@localhost bin]# mysql -h127.0.0.1 -P2345 -uuser -ppwd
2 Welcome to the MySQL monitor.  Commands end with ; or \g.
3 Your MySQL connection id is 1
4 Server version: 5.0.99-agent-admin
5
6 Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
7
8 Oracle is a registered trademark of Oracle Corporation and/or its
9 affiliates. Other names may be trademarks of their respective
10 owners.
11
12 Type 'help;' or '\h' for help. Type '\c' to clear the current input
13 statement.
14 mysql>

```

可以访问“help”表，来看MySQL管理员模式都能做些什么。可以使用SQL语句来访问

```

1 mysql> select * from help;
2 +-----+-----+
3 | command | description |
4 +-----+-----+
5 | SELECT * FROM help | shows this help |
6 | SELECT * FROM backends | lists the backends and their state |
7 | SET OFFLINE $backend_id | offline backend server, $backend_id is |
  | backend_ndx's id |
8 | SET ONLINE $backend_id | online backend server, ... |
9 | ADD MASTER $backend | example: "add master 127.0.0.1:3306", ... |
10 | ADD SLAVE $backend | example: "add slave 127.0.0.1:3306", ... |

```

```

11 | REMOVE BACKEND $backend_id | example: "remove backend 1", ...
    |
12 | SELECT * FROM clients      | lists the clients
    |
13 | ADD CLIENT $client         | example: "add client 192.168.1.2", ...
    |
14 | REMOVE CLIENT $client      | example: "remove client 192.168.1.2", ...
    |
15 | SELECT * FROM pwds         | lists the pwds
    |
16 | ADD PWD $pwd               | example: "add pwd user:raw_password", ...
    |
17 | ADD ENPWD $pwd             | example: "add enpwd user:encrypted_password",
... |
18 | REMOVE PWD $pwd            | example: "remove pwd user", ...
    |
19 | SAVE CONFIG                | save the backends to config file
    |
20 | SELECT VERSION              | display the version of Atlas
    |
21 +-----+-----+
    +-----+
22 16 rows in set (0.00 sec)
23
24 mysql>

```

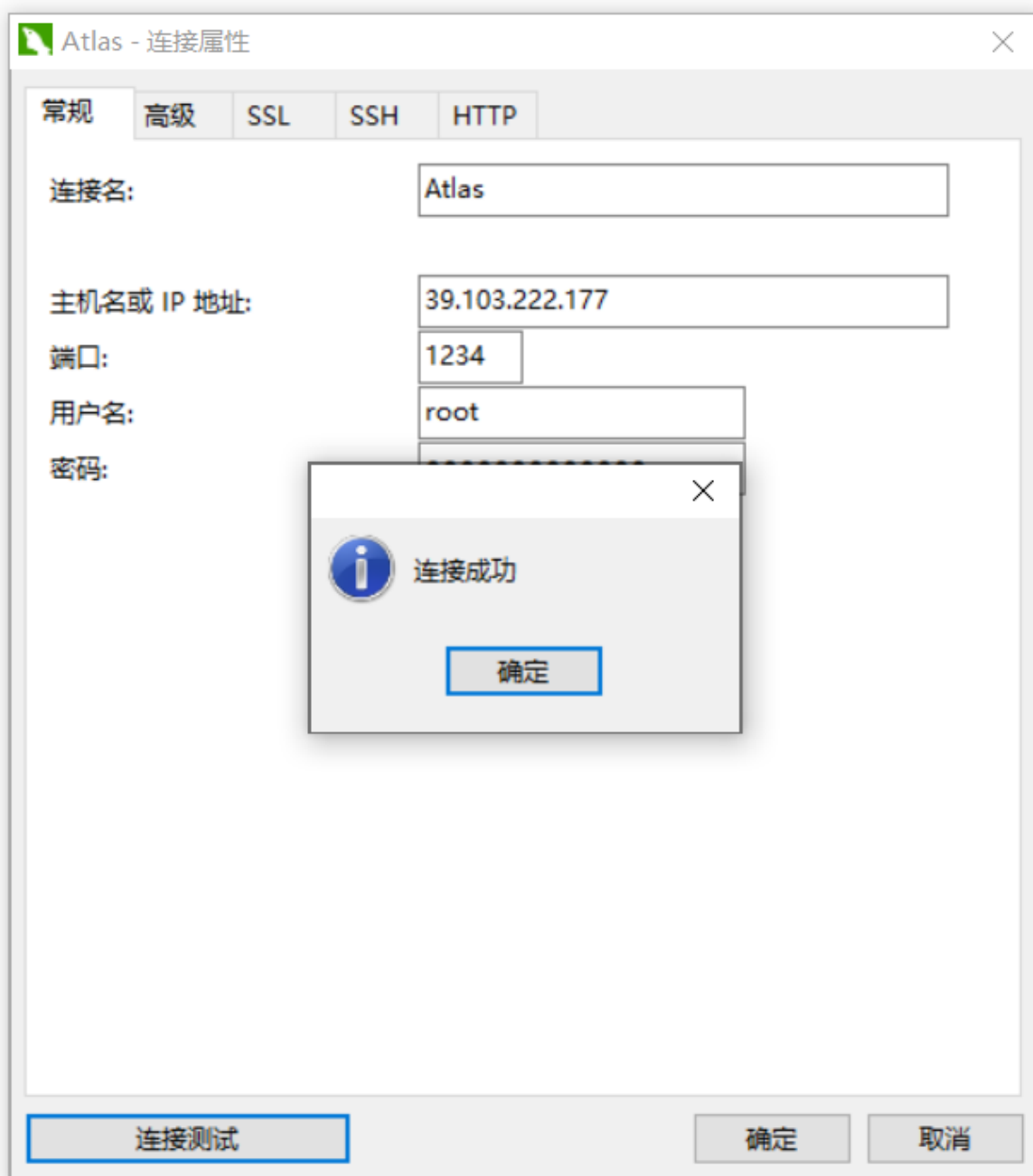
也可以使用工作接口来访问，使用命令 `mysql -h127.0.0.1 -P1234 -uroot -proot`

```

1 [root@localhost bin]# mysql -h127.0.0.1 -P1234 -uroot -proot
2 Welcome to the MySQL monitor.  Commands end with ; or \g.
3 Your MySQL connection id is 1
4 Server version: 5.0.81-log
5
6 Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
7
8 Oracle is a registered trademark of Oracle Corporation and/or its
9 affiliates. Other names may be trademarks of their respective
10 owners.
11
12 Type 'help;' or '\h' for help. Type '\c' to clear the current input
    statement.
13
14 mysql>

```

如果工作接口可以进入了，就可以在Windows平台下，使用Navicat来连接数据库，填写对应的host, Port, 用户名, 密码就可以。



## 2.4.5 测试

```
1 # 进入Atlas的管理模式
2 mysql -h127.0.0.1 -P2345 -uuser -ppwd
3 # 查看所有节点
4 SELECT * FROM backends;
```

```
mysql> SELECT * FROM backends;
+-----+-----+-----+-----+
| backend_ndx | address           | state | type |
+-----+-----+-----+-----+
| 1 | 39.103.222.177:3306 | up    | rw   |
| 2 | 39.99.131.178:3306 | up    | ro   |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

修改Slave的数据，Master不会同步Slave数据。此时读取数据，观察Master和Slave的数据那个被查询到了

### 三、基于主从复制的高可用方案

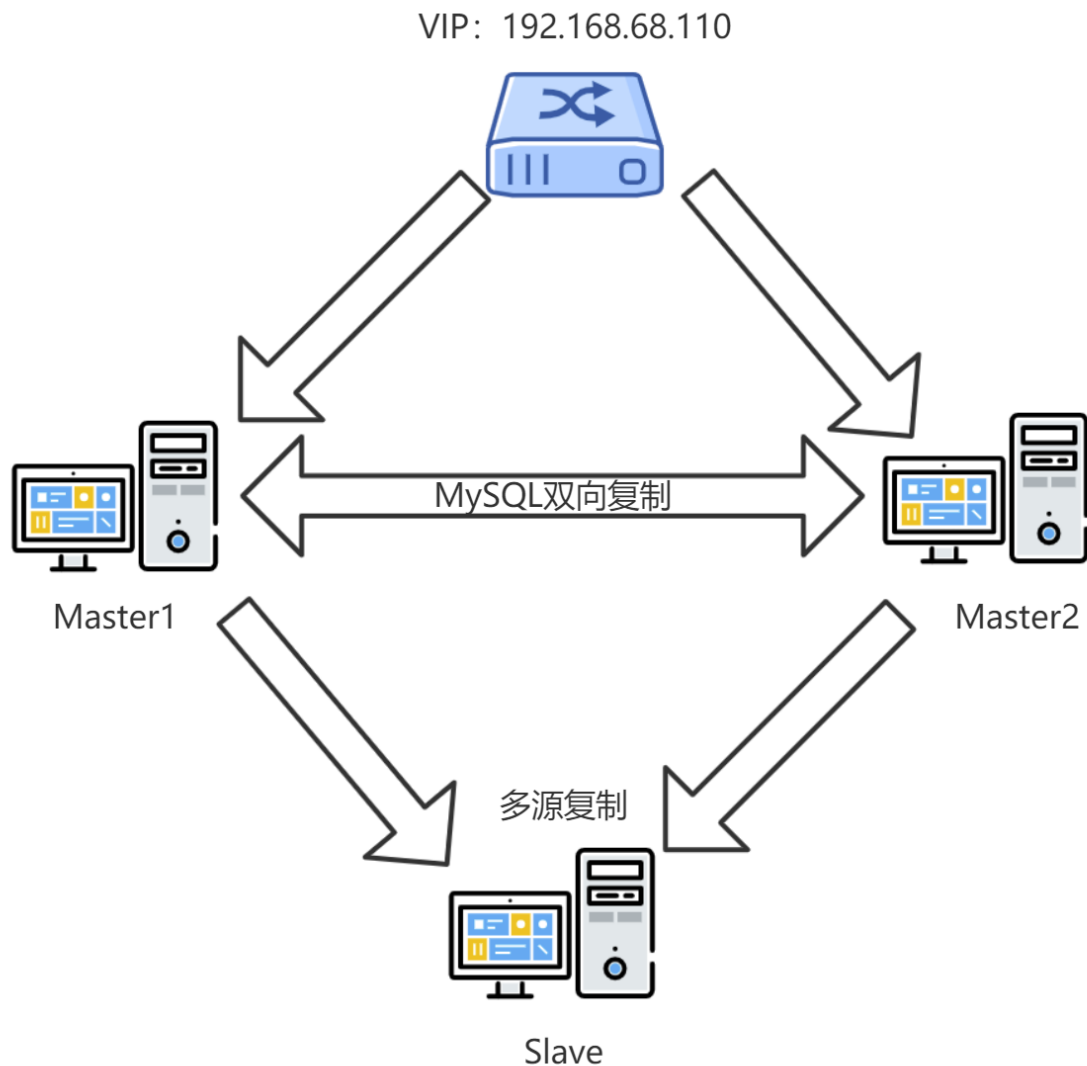
---

**双节点主从 + keepalived/heartbeat方案**，一般来说，中小型规模的时候，采用这种架构是最省事的。两个节点可以采用简单的一主一从模式，或者双主模式，并且放置于**同一个VLAN**中，在master节点发生故障后，利用keepalived/heartbeat的高可用机制实现快速切换到slave节点。

在这个方案里，有几个需要注意的地方：

- 把两个节点的**auto\_increment\_increment**（自增起始值）和**auto\_increment\_offset**（自增步长）设成不同值。其目的是为了避免master节点意外宕机时，可能会有部分binlog未能及时复制到slave上被应用，从而会导致slave新写入数据的自增值和原先master上冲突了，因此一开始就使其错开；当然了，如果有合适的容错机制能解决主从自增ID冲突的话，也可以不这么做；
- slave节点服务器配置不要太差，否则更容易导致复制延迟。作为热备节点的slave服务器，硬件配置不能低于master节点；
- 如果对延迟问题很敏感的话，可考虑使用MariaDB分支版本，或者直接上线MySQL 5.7最新版本，利用多线程复制的方式可以很大程度降低复制延迟；
- keepalived的检测机制需要适当完善，不能仅仅只是检查mysqld进程是否存活，或者MySQL服务端口是否可通，还应该进一步做数据写入或者运算的探测，判断响应时间，如果超过设定的阈值，就可以启动切换机制；
- keepalived最终确定进行切换时，还需要判断slave的延迟程度。需要事先定好规则，以便决定在延迟情况下，采取直接切换或等待何种策略。直接切换可能因为复制延迟有些数据无法查询到而重复写入；
- keepalived或heartbeat自身都无法解决**脑裂**的问题，因此在进行服务异常判断时，可以调整判断脚本，通过对第三方节点补充检测来决定是否进行切换，可降低**脑裂**问题产生的风险。

双节点主从+keepalived/heartbeat方案架构示意图见下：



### 3.1 配置双主集群

MySQL的可以配置两台服务器互为主从。用来搭建mysql的高可用架构，用来保证mysql服务器宕机的时候，能够自动的切换的另一台mysql服务器。主从的配置可以是基于日志点的也可以是基于GTID的，我们上面讲到了GTID的配置，所以我们现在配置一个基于GTID的双主集群。

#### 1) master1配置

修改 `/etc/my.cnf` 文件

```
1  # 服务器id, 一般是ip的最后一段
2  server-id=132
3  # 开启binlog
4  log-bin=mysql-bin
5  # 表示自增长字段每次递增的量, 其默认值是1, 取值范围是1 .. 65535
6  auto_increment_increment=2
7  # 表示自增长字段从那个数开始, 他的取值范围是1 .. 65535, 另外一台服务器的offset为2, 防止
   生成的主键冲突
8  auto_increment_offset=1
9  # 开启基于GTID的复制
10 gtid_mode = on
11 # 只记录对基于gtid的复制安全的语句
12 enforce-gtid-consistency=true
```

## 2) master2配置

```
1 server-id=133
2 log-bin=mysql-bin
3 auto_increment_increment=2
4 # 生成主键从2开始
5 auto_increment_offset=2
6 gtid_mode = on
7 enforce-gtid-consistency=true
```

## 3) 建立主从关系

如果之前已经开启的主从复制，建议使用 `stop slave` 关闭。在每个节点上切换binlog，执行如下语句：

```
1 # 使用新的binlog
2 mysql> flush logs;
3 # 清空binlog
4 mysql> reset master;
```

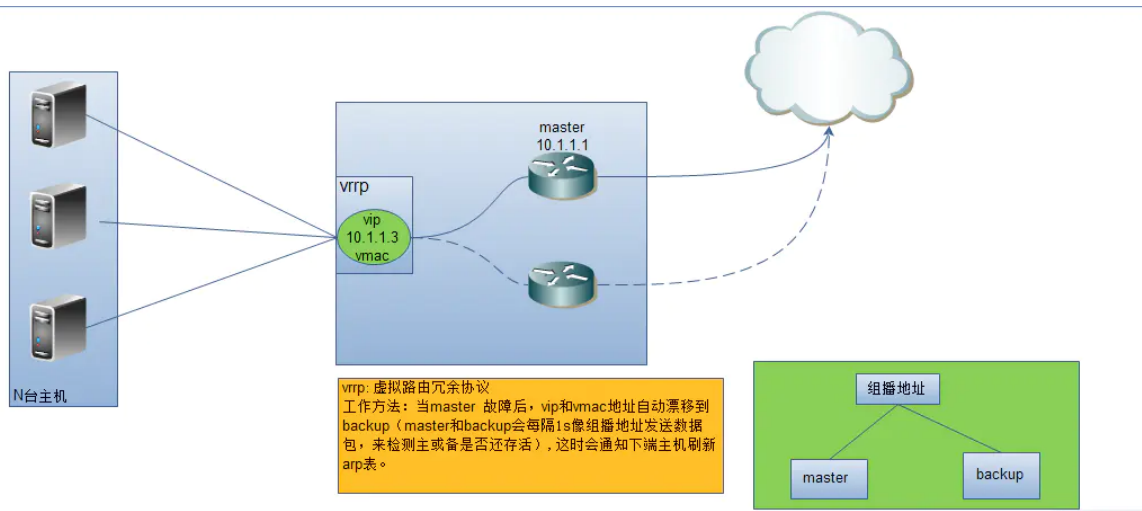
然后使用 `change master` 语句建立主从关系：

```
1 mysql> change master to
2 master_host='192.168.68.132',
3 master_port=3306,
4 master_user='root',
5 master_password='root',
6 master_auto_position = 1;
```

## 3.2 安装keepalived

Keepalived高可用服务对之间的故障切换转移，是通过 VRRP (Virtual Router Redundancy Protocol ,虚拟路由器冗余协议) 来实现的。在 Keepalived服务正常工作时，主 Master节点会不断地向备节点发送（多播的方式）心跳消息，用以告诉备Backup节点自己还活着，当主 Master节点发生故障时，就无法发送心跳消息，备节点也就因此无法继续检测到来自主 Master节点的心跳了，于是调用自身的接管程序，接管主Master节点的 IP资源及服务。而当主 Master节点恢复时，备Backup节点又会释放Master节点故障时自身接管的IP资源及服务，恢复到原来的备用角色。





## 什么是VRRP呢?

VRRP, 全称 Virtual Router Redundancy Protocol, 中文名为虚拟路由冗余协议, VRRP的出现就是为了解决静态路由的单点故障问题, VRRP是通过一种竞选机制来将路由的任务交给某台VRRP路由器的。

### 1) 安装keepalived

安装keepalived非常简单可以直接使用yum方式在线安装:

```
1 | yum install keepalived -y
```

获取配置文件路径

```
1 | rpm -qc keepalived
2 | /etc/keepalived/keepalived.conf
3 | /etc/sysconfig/keepalived
```

### 2) 配置keepalived

```
1 | # 配置通知的email
2 | global_defs {
3 |     notification_email {
4 |         acassen@firewall.loc
5 |         failover@firewall.loc
6 |         sysadmin@firewall.loc
7 |     }
8 |     notification_email_from Alexandre.Cassen@firewall.loc
9 |     smtp_server 192.168.200.1
10 |     smtp_connect_timeout 30
11 |     router_id LVS_DEVEL
12 |     vrrp_skip_check_adv_addr
13 |     vrrp_strict
14 |     vrrp_garp_interval 0
15 |     vrrp_gna_interval 0
16 | }
17 | # 检查mysql脚本, 定时执行
18 | vrrp_script check_run {
```

```

19 script "/usr/local/check_run.sh"
20 interval 3
21 }
22 # 设置虚拟ip
23 vrrp_instance VI_1 {
24     # 当前节点的状态MASTER、BACKUP
25     state MASTER
26     # 当前服务器使用的网卡名称，使用ifconfig查看
27     interface eth0
28     #VRRP组名，两个节点的设置必须一样
29     virtual_router_id 51
30     #Master节点的优先级（1-254之间）
31     priority 100
32     #组播信息发送间隔，两个节点设置必须一样
33     advert_int 1
34     #设置验证信息，两个节点必须一致
35     authentication {
36         auth_type PASS
37         auth_pass 1111
38     }
39     #虚拟IP,对外提供MySQL服务的IP地址
40     virtual_ipaddress {
41         172.26.233.206
42     }
43 }
44

```

### 3) 检查脚本check\_run.sh

```

1  #!/bin/bash
2  . /root/.bashrc
3  count=1
4
5  while true
6  do
7
8  mysql -uroot -pitxiongge@1 -s /var/lib/mysql/mysql.sock -e "select now();" >
/dev/null 2>&1
9  i=$?
10 ps aux | grep mysqld | grep -v grep > /dev/null 2>&1
11 j=$?
12 if [ $i = 0 ] && [ $j = 0 ]
13 then
14     exit 0
15 else
16     if [ $i = 1 ] && [ $j = 0 ]
17     then
18         exit 0
19     else
20         if [ $count -gt 5 ]
21         then
22             break
23         fi
24         let count++
25         continue
26     fi
27 fi

```

```
28
29 done
30
31 systemctl stop keepalived.service
```

#### 4) 启动keepalived

```
1 systemctl start keepalived
```

#### 5) 查看vip

```
1 [root@k8s-master01 ~]# ip addr
2 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
   default qlen 1000
3     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4     inet 127.0.0.1/8 scope host lo
5         valid_lft forever preferred_lft forever
6 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group
   default qlen 1000
7     link/ether 00:16:3e:0b:0f:0a brd ff:ff:ff:ff:ff:ff
8     inet 172.26.233.197/20 brd 172.26.239.255 scope global dynamic eth0
9         valid_lft 301467836sec preferred_lft 301467836sec
10    inet 172.26.233.110/32 scope global eth0
11        valid_lft forever preferred_lft forever
```

### 3.3 配置多源复制Slave节点

MySQL 5.7已经开始支持了多源复制，MySQL 5.7之前只能实现一主一从、一主多从或者多主多从的复制，如果想实现多主一从的复制，只好使用MariaDB，但是MariaDB又与官方的MySQL版本不兼容的，在MySQL 5.7版本已经可以实现多主一从的复制了。MySQL 5.7版本相比之前的版本，无论在功能还是性能、安全等方面都已经提升了不少，值得大家去研究和使用。

#### 多主一从架构带来的好处：

- 一、在从服务器进行数据汇总，如果我们的主服务器进行了分库分表的操作，为了实现后期的一些数据统计功能，往往需要把数据汇总在一起再统计。
- 二、如果我们想在从服务器时时对主服务器的数据进行备份，在MySQL 5.7之前每一个主服务器都需要一个从服务器，这样很容易造成资源浪费，同时也加大了DBA的维护成本，但MySQL 5.7引入多源复制，可以把多个主服务器的数据同步到一个从服务器进行备份。

#### 1) 将Master节点的数据同步到Slave节点

略

#### 2) 配置my.cnf

```
1 server-id=134
2 gtid_mode=ON
3 enforce-gtid-consistency=ON
4 master_info_repository=table
5 relay_log_info_repository=table
```

### 3) 配置多源复制

```
1 mysql> change master to
2 master_host='192.168.68.133',
3 master_port=3306,
4 master_user='root',
5 master_password='root',
6 master_auto_position = 1
7 FOR CHANNEL 'm-133';
```

和普通复制不同的是需要增加 `FOR CHANNEL 'xxx'` 语句指定不同的频道复制。由于是多源复制必须指定参数 `master_info_repository=table`

### 4) 配置跳过的GTID集合

```
1 #master节点 :
2 mysql> flush logs;
3 mysql> show global variables like 'gtid_executed' \G
4
5 #slave节点:
6 mysql> reset master;
7 Query OK, 0 rows affected (0.00 sec)
8
9 mysql> set global gtid_purged='79633990-a991-11ec-a07c-00163e0b0f0a:1-
10 3,cb1ee3b3-a996-11ec-a2ca-00163e146e7d:1';
11 Query OK, 0 rows affected (0.00 sec)
12
13 #启动节点
14 mysql> start slave for channel 'm-132';
15
16 #查看某一频道的复制状态
17 mysql> show slave status for channel 'm-132' \G
```