mysql中级教程

我的昵称末尾有空格

目　录

第一章：mysql触发器

第一章：mysql触发器

1.所谓触发器，就是指设置好某个表的某个操作[insert ,update ,delete]时候，同时触发的一个操作[insert,update,delete]

使用场景：比如商城订单操作，当用户完成下单操作后，需要对商品表进行库存减一操作，这时候我们可以采用PHP来做相应的逻辑处理，但如果是使用mysql触发器来操作，就可以节省我们写PHP逻辑代码的时间

我们如果要深入理解mysql触发器，必须掌握触发器的四要素：

1.监控的表[数据库中的某张表]

2.监视的事件[某张表表的insert,update,delete操作]

3.触发时间在 insert,update,delete 等操作前还是操作后

4.需要触发的事件[insert,update,delete]

基本格式：

```
delimiter $$                    --delemiter:声明定界符为 $$
create trigger orderMinusOne --创建一个触发器:orderMinusOne触发器名称
after|before                    --触发时间
insert|delete|update            --监听的事件
on tabName                      --监听的表
for each row                    --行级触发器。mysql不支持语句触发器，所以必须写for each row,每行受影响，触发器都执行
begin                           --开始 xxx
    #SQL语句
end $$                          --结束
delimiter;
```

一个栗子：模拟商品下单场景,用户下单成功之后，根据购买数量相应的商品表商品数量减去购买数。

商品表：

```
SET FOREIGN_KEY_CHECKS=0;
-- ----------------------------
-- Table structure for tb_goods
-- ----------------------------
DROP TABLE IF EXISTS `tb_goods`;
CREATE TABLE `tb_goods` (
  `goods_id` int(11) NOT NULL AUTO_INCREMENT COMMENT '商品id',
  `goods_num` int(11) unsigned NOT NULL COMMENT '商品数量',
  `goods_price` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '商品价格',
  `goods_name` varchar(255) NOT NULL DEFAULT '' COMMENT '商品名',
  `is_rease` tinyint(1) NOT NULL DEFAULT '0' COMMENT '是否发布，0发布，1不发布',
  PRIMARY KEY (`goods_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;
```

```
订单表：
SET FOREIGN_KEY_CHECKS=0;
-- ----------------------------
-- Table structure for tb_order
-- ----------------------------
DROP TABLE IF EXISTS `tb_order`;
CREATE TABLE `tb_order` (
  `order_id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT '订单id',
  `goods_id` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '订单id',
  `buy_num` int(11) NOT NULL DEFAULT '0' COMMENT '购买个数',
  `order_number` varchar(19) NOT NULL DEFAULT '' COMMENT '订单编号',
  `is_valid` tinyint(1) NOT NULL DEFAULT '0' COMMENT '是否有效，0有效，1无效',
  PRIMARY KEY (`order_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;
```

因为触发器会监视mysql的insert,delete,update等操作,所以如果表中新增一列或者减少一列都会被记录,如果需要找到新增列[insert]的字段字,可以使用 new.字段 表示，相应的减少列[delete]操作可以用 old.字段表示。

创建触发器：

```
delimiter $$                    --delemiter $$ 声明定界符为 delimiter
create trigger orderMinusOne    --创建一个触发器:orderMinusOne触发器名称
after                           --触发时间
insert                          --监听的事件
on tb_order                     --监听的表
for each row
begin                           --开始
    update tb_goods set goods_num = goods_num - new.buy_num where goods_id = new
.goods_id
                                -- 商品表商品数量减去购买数量的值
end $$                          --以 $$ 分隔符结束
delimiter;
```

注：对于insert操作,只能使用new.字段,对于delete操作,只能使用old.字段,而update操作,old.字段,new.字段都可以使用

```
查看我们创建的触发器:
show triggers;
```

```
删除触发器命令：
drop trigger triggerName;
```

当然，这只是一个简单的栗子，实际应用场景要比这复杂的多，比如，如果库存字段我们没有设置为非负数极可

能出现超卖的情况。除了逻辑代码与限制字段非负，用触发器我们同样也可以实现。

一个符合生产情况的栗子：

```
delimiter $$                    --delemiter $$ 声明定界符为 delimiter
create trigger orderMinusOne    --创建一个触发器:orderMinusOne触发器名称
before                          --触发时间
insert                          --监听的事件
on tb_order                     --监听的表
declare
goodsnum int                    --declare 声明一个int类型的goodsnum的变量, 用于后续存储查询出来的变量
for each row
begin                           --开始
    select goods_num into goodsnum from tb_goods where goods_id = new.goods_id --将商品表的库存查询出来赋值给goods_num
    if new.buy_num > goodsnum then      --判断,如果购买数量大于库存量的话
        set new.buy_num = goodsnum      --让购买数量等于库存量
    end if
    update tb_goods set goods_num = goods_num - new.buy_num where goods_id = new.goods_id     --执行商品表库存减操作
end $$
delimiter;
```

第二章：mysql存储过程

1.什么是mysql存储过程 存储过程[Stored Procedure]是一组[或者是若干条SQL]为了完成特定功能的SQL语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给定参数[如果该存储过程带有参数]来调用执行它。

2.存储过程的创建

--创建语法：

```
create procedure procedureName() --同一个数据库中，存储过程名具有唯一性
begin
    #SQL语句集
end
```

举个栗子：创建一个查询当前时间的存储过程

```
delimiter $$
create procedure procedureName()
begin
    select now() as nowtime;
end $$
delimiter;
```

这样我们就完成了第一个存储过程的创建，创建成功之后我们如何调用呢，在你的SQL语句中，只需要

```
call procedureName() --select 存储过程名 即可调用
```

```
 当然，你还可以查看所有的存储过程:
show procedure status;
```

3.存储过程中变量的引用 存储过程中可以通过引用变量来完成复杂的编程，在存储过程中使用 declare 来声明变量，

格式如下：

```
declare 变量名[一般不以@开头且具有一定意义[@是mysql的关键字]] 变量类型[变量类型包括：数值类型[tinyint,int/integer,bigint,float,double,decimal],时间日期类型，字符串类型，][default 默认值[可选]]
```

又一个栗子：

```
delimiter $$                        --delimiter 定义结束符
create procedure procedureName()
begin                                --开始存储过程
     declare age int default 18;   --声明一个int类型的age变量，并且默认值为18
     declare myname varchar(255) default 'xiaoxiao'; --声明一个myname的varchar类
型的变量，默认值为'xiaoxiao'
     select CONCAT(myname, age) as descr;            --查询刚刚定义的变量并取别名为d
escr
end $$                                               --结束存储过程
delimiter;                                           --将mysql的存储过程设置为 ;
```

既然我们可以为存储过程赋值，那么我们可以想到是否可以在存储过程中进行变量的计算，答案也是也可以的，再来一个栗子：

```
delimiter $$
create procedure procedureName()
begin
 declare age int default 18;
 declare height int default 171;
 set age := age + 18;      -- 变量的加法运算，变量的运算格式为，set 变量名 :=变量名 运
算符[+, -, * , /]值
 select age;
end $$
delimiter;
--调用
call procedureName ()      --结果为 age 36
```

4.mysql中的流程控制

举个栗子说明一下：

```
delimiter $$
create procedure procedureName()
begin
 declare age int default 18;
 declare height int default 171;
 set age := age + 18; -- 变量的加法运算，变量的运算格式为，set 变量名 :=变量名 运算符（
+, -, * , /）值
 if age > 30 then
    select '30岁，中年啦';
 else
    select '小年轻';
 end if;              -- end if结束if语句块
end $$
delimiter;
```

调用的结果为：'30岁，中年啦';

类似的语句还有 case...when...

5.存储过程参数的传递

存储过程主要分为三种参数类型,in[输入参数],out[输出参数],INOUT[输入输出参数] IN[输入]参数类型:该参数的值必须在调用存储过程时指定，如果在调用存储过程中修改该参数的值则该值不能被返回

IN参数类型:

基本格式:

```
delimiter $$
create procedure InType(in type char) -- 参数类型[IN/OUT/INOUT,参数名,参数类型[int,char,varchar等...]]
begin
IF type='A' THEN
    SELECT '我是IN参数A';
ELSEIF  type='B' THEN
    SELECT '我是IN参数B';
ELSE
    SELECT '其他';
END IF;
end $$
delimiter;
```

调用:

```
call InType('A'); -- 输出'我是IN参数A'
```

OUT[输出]参数类型:

```
delimiter $$
create procedure outType(OUT type int)
begin
  secelt type = 1+2;
end $$
delimiter;
```

调用：

```
call outType(@type); --输出 3
```

INOUT[输入输出]参数类型:

```
delimiter $$
create procedure inoutType(INOUT ntime INT)
begin
  declare nowtime INT(11);                  --定义变量
  IF ntime > (select unix_timestamp) THEN   --判断
     SET nowtime = (select unix_timestamp);
  ELSE
     SET nowtime = ntime;
  END IF;
  SET ntime = nowtime;                      --返回赋值
end $$
delimiter;
-- 传递一个时间戳,mysql对当前时间戳进行判断,如果传递进来的时间戳大于当前时间戳,设置nowtime变量为当前时间戳,否则则为传递进来的时间戳,并输出结果
```

一个开发过程中使用存储过程快速填充表的栗子:

栗子中user表有四个字段,分别为,name,age,sex,is_delete,快速为表填充6百万的测试数据,如下:

```
delimiter $$
create procedure fillData()
begin
set @i =1;            --set 定义变量
while @i<=6000000 do --开始循环,循环变量小于6000000
insert into user (name,age,sex,is_delete)values(CONCAT("我是第",@i,"个name"),(SELECT RAND() * 20),(SELECT RAND() * 2),(SELECT RAND() * 1)); --批量向表里面插入6百万条数据, CONCAT为拼接字符串
set @i = @i+1;        --变量累加
end while;            --结束循环
end $$
delimiter;
```

第三章：mysql游标

游标 对应SQL语句执行取得的N条结果，而对应N条结果集组成的资源，取出资源的接口/句柄，就是游标[从结果集中依次一条一条的取]

游标的5个基本概念:[类似于PHP的文件处理,先打开一个文件,读取文件内容,关闭文件...]

1.定义游标

```
declare cursorName[游标名] cursor for select sql语句;
```

2.打开游标

```
open cursorName
```

3.游标的使用,fetch

```
declare 变量1 数据类型[与列值的数据类型相同]
fetch [NEXT | PRIOR | FIRST | LAST] from cursorName [ INTO 变量1[变量N…] ]
```

4.关闭游标

```
close cursorName
```

5.释放游标

```
deallocate cursorName;
```

还是用一个栗子来说明游标的使用,在游标中我们可以灵活的添加条件,以达到获取到我们期望的数据集

```
delimiter $$
create procedure getGoods()
begin
    declare rowgoodsid int;     --声明一个存储goods_id的int类型的变量 rowgoodsid
    declare rowgoodsnum int;
    declare rowgoodsname varchar(255);

    declare rests int default 1;
    declare getGoodsAll cursor for select goods_id,goods_num,goods_name from tb
_goods;   --声明游标
    declare continue handlers for NOT FOUND set rests :=0; - - 如果fetch不到值,就
将rests设置为0
```

```
    select count(goods_id) into totalNum from tb_goods;

    open getGoodsAll;           --打开游标
    fetch getGoodsAll into rowgoodsid,rowgoodsnum,rowgoodsname; --事先fetch,防止
后续取到空集

    repeat                      -- 开始循环,取值
    select rowgoodsid,rowgoodsnum,rowgoodsnum; --取值
    fetch getGoodsAll into rowgoodsid,rowgoodsnum,rowgoodsname; --fetch取值
    until rests = 0 end repeat;                -- 如果fetch不到值,结束循环
    -- 这里同样可以使用while循环
    close getGoodsAll;                         --关闭游标
end $$
delimiter;
```

第四章：mysql权限控制

权限控制可以做什么?mysql主从控制,防止mysql非法链接,杜绝delete,drop等危险操作,增强安全性等等

mysql的连接方式为:

```
mysql -h 主机地址 -u 用户名 -p 用户密码
```

那么在我们本机为什么可以直接使用 mysql -u root -p来直接登录我们的mysql呢,是因为在mysql的mysql库的user表中有我们的相关账号,显然,这在测试或者开发环境中并没有什么问题,如果是生产环境,显然是隐藏巨大的安全隐患的,此时,权限控制就显得尤为重要

1.修改连接host主机

```
update user set host = 'host' where user = 'root'; --host主机名,root用户名
flush privileges; -- 冲刷权限
```

2.修改用户密码

```
update user set password = password('你的密码') where user = '用户名';
flush privileges;
```

3.查看当前mysql有哪些用户

切换到mysql库下:

select Host,User,Password from user;

select * from user where user = 'user'; -- 查看某一个用户具有哪些全局权限

3.全局授权,针对库

```
--新增用户
grent [权限[all,create,drop,insert,selete,update,select]] on *.* to user@'host'
 identified by 'password';
--创建 user用户在所有库[第一个*],所有表[第二个*]具有什么权限
-user表示用户名,host表示主机,password表示密码
--如果给all 权限,创建的用户可以在库中进行所有操作,相应的,具体授某一个权限,那创建的用户就只拥有一个权限
flush privileges;
revoke [权限[all,create,drop,insert,selete,update,select]] on *.* from user@'ho
st';--回收权限
flush privileges;
```

4.查看当前用户Db级权限

```
select * from db;
grent [权限[all,create,drop,insert,selete,update,select]] on 库名.* to user@'hos
t';
flush privileges;
--回收同上
```

5.针对表的权限

```
grent [权限[all,create,drop,insert,selete,update,select]] on 库名.表名 to user@'h
ost';
flush privileges;
select * from tables_priv; --查看表级权限
--回收同上
```

第五章：mysql主从复制

如下文图片未正常显示,请移步链接:https://blog.csdn.net/AnPHPer/article/details/80255177

一、为什么要搭建主从服务器和实现读写分离

1.总结起来就一点,实现并发吞吐和负载能力。通过搭建主从服务器实现读写分离,提高MySQL的负载能力

2.主从的基本实现原理 (本例 主服务器ip: 192.168.2.187,后面简称 master;从服务器ip: 192.168.2.199后面简称 slave)

(1) 主(master)服务器配置 bin-log

(2) 从(slave)服务器配置 relay-log

(3) 主(master)服务器为从服务器授权读取bin-log的账号

(4) 从(slave)服务器使用账号连接主服务器

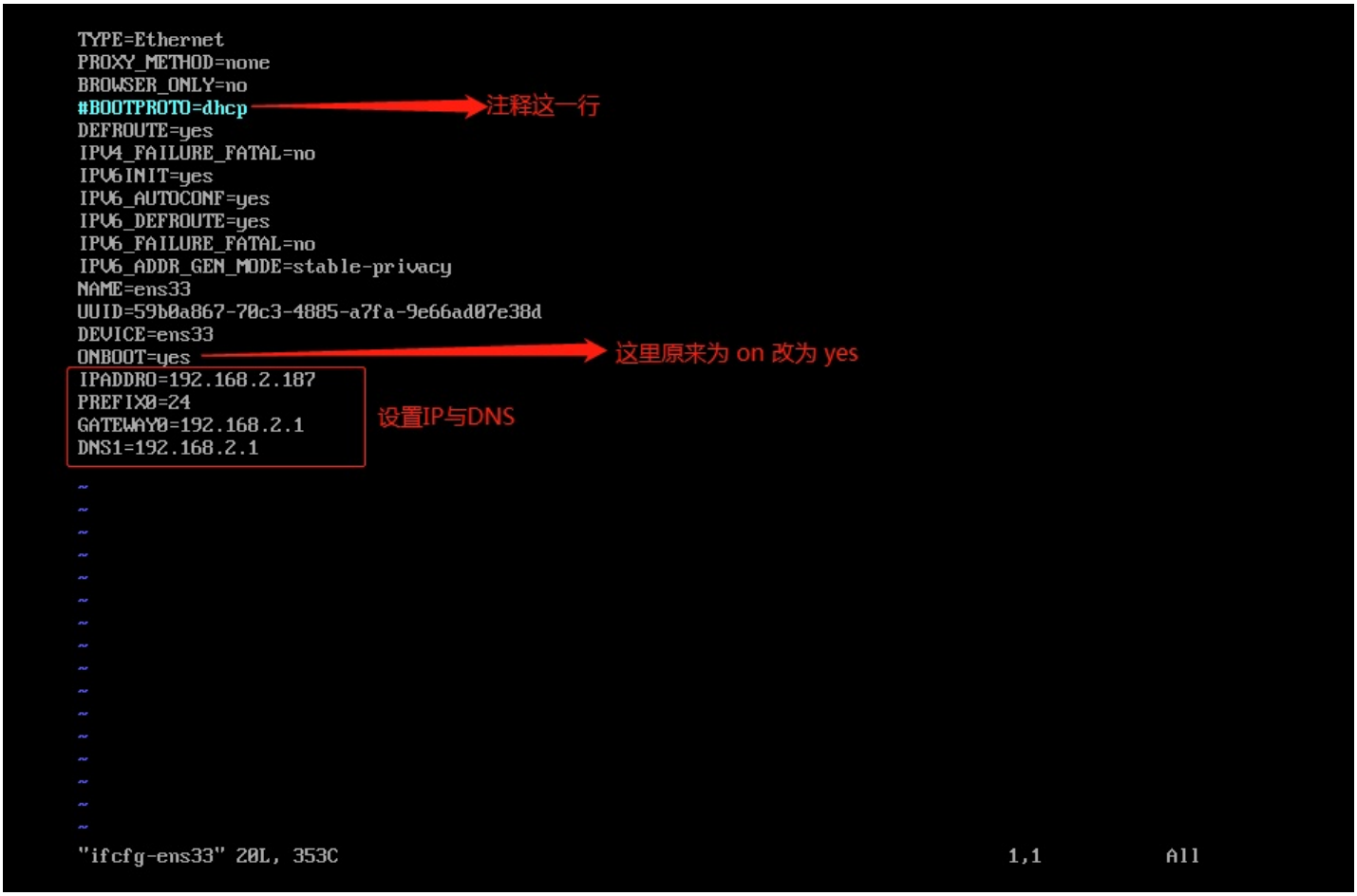
(5) 从(slave)服务器读取主服务器的 bin-log

二、准备工作->网络设置

① 分别设置两台虚拟机的网络,使其能够相互 ping 通

centOS7 的网络配置文件为 /etc/sysconfig/network-scripts/ifcfg-ens33(某些为: /etc/sysconfig/network-scripts/ifcfg-eno16777736)

如图设置网络



保存退出并重启网络即可

```
service network restart
```

三、准备工作->安装MySQL

1.两台Mysql服务器

2.安装

① 博主使用的是centOS7 64位,(PS:如果不会安装centOS7,请参照https://blog.csdn.net/anphper/article/details/80251223)

② 安装Mysql前需要先安装几个基本命令

(1) wget

```
 yum -y install wget
```

(2) telnet

```
yum -y install telnet
```

(3) ifconfig

```
yum provides ifconfigyum whatprovides ifconfigyum install net-tools
```

(4) rz sz

```
yum -y install lrzsz
```

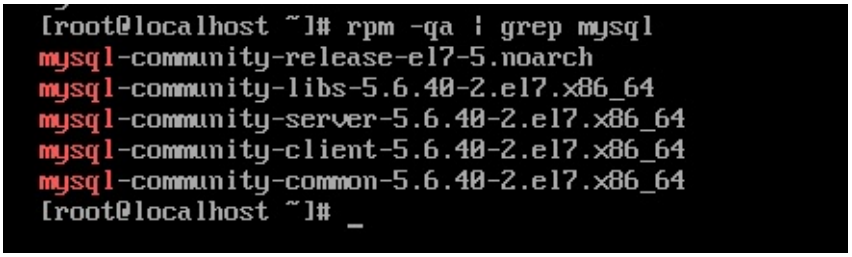
(5) vim

```
yum -y install vim*
```

③ 安装MySQL

(1) 检查系统中是否有MySQL 若没有返回则系统没有安装MySQL,若如下图,则已经安装,不需要再次安装

```
rpm -qa | grep mysql
```



(2) 下载mysql源

```
wget http://repo.mysql.com/mysql-community-release-el7-5.noarch.rpm
```

(3) 安装 mysql-community-release-el7-5.noarch.rpm 包(mysql的依赖包)

```
rpm mysql-community-release-el7-5.noarch.rpm
```

(4) 安装MySQL

```
 yum install mysql-server
```

(5) 重启服务

```
service mysqld restart
```

(6) 安装完成后还需要对MySQL设置密码,因为初次安装是没有密码的

登录mysql

```
mysql -u root -pmysql > use mysql;mysql > update user set password=password('12
3456') where user='root';mysql > exit;
```

重启MySQL

四、 关闭防火墙(关闭防火墙的原因是使得从服务器可以连接主服务器)

centOS7 关闭防火墙的命令不再是 service iptables stop

① 查看防火墙状态

```
 firewall-cmd --state
```

② 关闭防火墙

```
systemctl stop firewalld.service
```

```
[root@localhost ~]# firewall-cmd --state
not running
[root@localhost ~]# systemctl stop firewalld.service
Warning: firewalld.service changed on disk. Run 'systemctl daemon-reload' to reload units.
[root@localhost ~]#
```

③ 确认防火墙状态

```
firewall-cmd --state
```

```
[root@localhost ~]# firewall-cmd --state
not running
[root@localhost ~]#
```

如图则已关闭

五、主从搭建

① 分别 修改两台服务器的 mysql 配置文件 /etc/my.cnf（master 的 server-id 设置为1)

```
server-id = 1  --主服务器 server-id 一般设置为1,从服务器设置为IP端的后几位log_bin = mysql-bin
```

```
[client]
port = 3306
socket = /tmp/mysql.sock
default-character-set = utf8mb4

[mysql]
prompt="MySQL [\d]> "
no-auto-rehash

[mysqld]
port = 3306
socket = /tmp/mysql.sock

basedir = /usr/local/mysql
datadir = /data/mysql
pid-file = /data/mysql/mysql.pid
user = mysql
bind-address = 0.0.0.0
server-id = 1

init-connect = 'SET NAMES utf8mb4'
character-set-server = utf8mb4

skip-name-resolve
#skip-networking
back_log = 300

max_connections = 1258
max_connect_errors = 6000
open_files_limit = 65535
table_open_cache = 1024
max_allowed_packet = 500M
binlog_cache_size = 1M
max_heap_table_size = 8M
tmp_table_size = 128M
-                                                    36,0-1        Top
```

② 确保主从服务器上面数据库/表一致,便于同步

③ 主服务器配置

创建一个专门用于同步的账号

```
grant replication slave on *.* to 'lili'@'%' identified by '123456';--  *.* 意思为允许连接任何库任何表,lili 为连接的账号名,%为任何的连接源,123456为连接的密码
```

```
Thanks to the contributor - Magnus udd
mysql root@localhost:(none)> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| theCentOS7         |
+--------------------+
4 rows in set
Time: 0.011s
mysql root@localhost:(none)> create database zhucong;
Query OK, 1 row affected
Time: 0.002s
mysql root@localhost:(none)> use zhucong;
You are now connected to database "zhucong" as user "root"
Time: 0.002s
mysql root@localhost:zhucong> create table users(
(1064, u"You have an error in your SQL syntax; check the manual that corresponds to your MySQL serve
r version for the right syntax to use near '' at line 1")
mysql root@localhost:zhucong> create table users(`id` int unsigned auto_increment,`name` varchar(255
                           -> ) not null,primary key(`id`))engine = InnoDB default charset=utf8;
Query OK, 0 rows affected
Time: 0.033s
mysql root@localhost:zhucong> grant replication slave on *.* to 'lisi'@'%' identified by '123456';
Query OK, 0 rows affected
Time: 0.002s
mysql root@localhost:zhucong> _


 [F3] Multiline: OFF
```

要将输入定向到该虚拟机，请在虚拟机内部单击或按 Ctrl+G。

查看主服务器状态:

```
show master status;
```

```
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000006 | 908      |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set
Time: 0.012s
mysql root@localhost:(none)>
```

生成的bin_log文件　　二进制文件写入的位置

④ 从服务器配置

⑴ 从服务器配置

```
--读取master的bin_logchange master to master_host = '192.168.2.187',master_user=
'lili',master_password='123456',master_log_file='mysql-bin.000006',master_log_p
os = 643; --master_host 主服务器ip--master_user 主服务器上分配的账号--master_passwo
rd 主服务器为账号分配的密码--master_log_file bin-log日志--master_log_pos 从日志的多少
二进制位置开始读
```

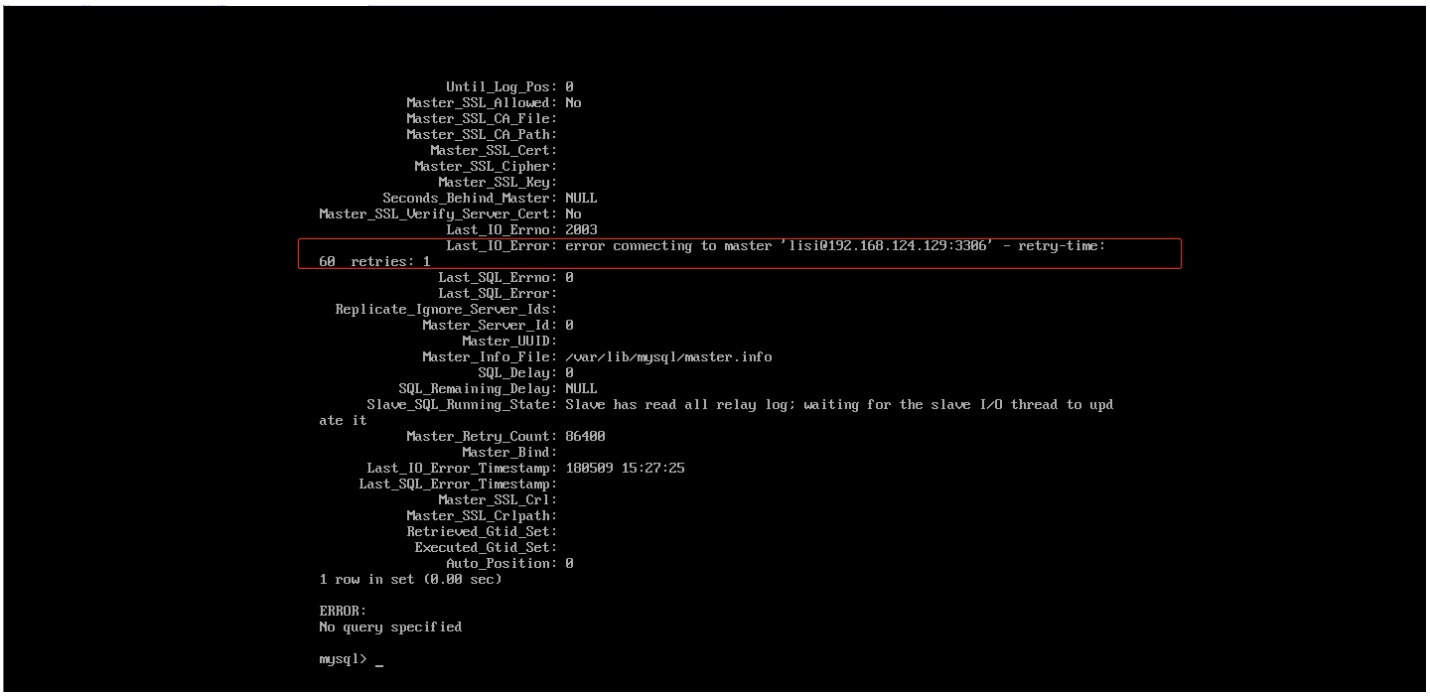
```
flush privileges --冲刷权限
```

(2) 开启从服务器

```
start slave;
```

(3) 查看从服务器状态

```
show slave status\G;
```



发现一条error(设置主从之后从服务器的探针会一直去尝试连接主服务器,每60S探测一次),说明主从失败

分析原因,防火墙之前我们已经禁用了,考虑两台服务器是否能 ping 通,3306端口是否被禁用

相互ping (主ping从 : ping 192.168.2.199)(从 ping 主 : ping 192.168.2.187) 发现可以 ping 通

然后主从分别查看端口 3306 占用情况

发现端口并没有占用,再次关闭防火墙,重启服务,重启Mysql 再次查看状态 show slave status\G; 主从OK

```
                  Until_Log_File:
                   Until_Log_Pos: 0
              Master_SSL_Allowed: No
              Master_SSL_CA_File:
              Master_SSL_CA_Path:
                 Master_SSL_Cert:
               Master_SSL_Cipher:
                  Master_SSL_Key:
           Seconds_Behind_Master: 0
   Master_SSL_Verify_Server_Cert: No
                   Last_IO_Errno: 0
                   Last_IO_Error:
                  Last_SQL_Errno: 0
                  Last_SQL_Error:
    Replicate_Ignore_Server_Ids:
               Master_Server_Id: 1
                    Master_UUID: d3b0e276-4ac2-11e8-8507-000c292da829
               Master_Info_File: /var/lib/mysql/master.info
                      SQL_Delay: 0
            SQL_Remaining_Delay: NULL
        Slave_SQL_Running_State: Slave has read all relay log; waiting for the slave I/O thread to upd
   ate it
              Master_Retry_Count: 86400
                     Master_Bind:
         Last_IO_Error_Timestamp:
        Last_SQL_Error_Timestamp:
                  Master_SSL_Crl:
              Master_SSL_Crlpath:
              Retrieved_Gtid_Set:
               Executed_Gtid_Set:
                   Auto_Position: 0
   1 row in set (0.00 sec)

   ERROR:
   No query specified

   mysql>
```

六、 测试主从

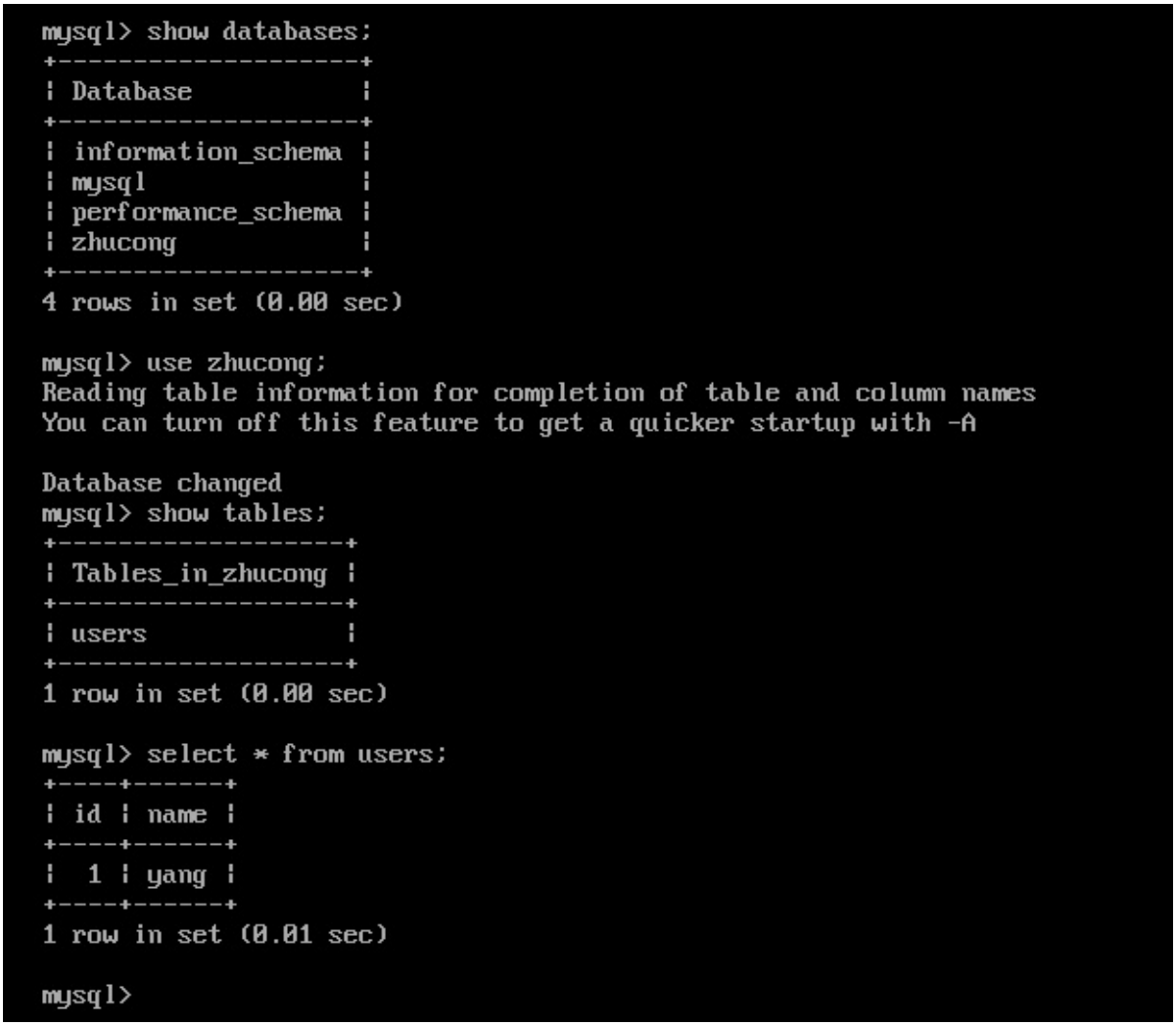
① 主服务器创建一张表

```
mysql root@localhost:(none)> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| theCentOS7         |
+--------------------+
4 rows in set
Time: 0.011s
mysql root@localhost:(none)> create database zhucong;
Query OK, 1 row affected
Time: 0.002s
mysql root@localhost:(none)> use zhucong;
You are now connected to database "zhucong" as user "root"
Time: 0.002s
mysql root@localhost:zhucong> create table users(
(1064, u"You have an error in your SQL syntax; check the manual that corresponds to your MySQL serve
r version for the right syntax to use near '' at line 1")
mysql root@localhost:zhucong> create table users(`id` int unsigned auto_increment,`name` varchar(255
                           -> ) not null,primary key(`id`))engine = InnoDB default charset=utf8;
Query OK, 0 rows affected
Time: 0.033s
mysql root@localhost:zhucong> _




 [F3] Multiline: OFF
```

② 插入一条数据

```
mysql root@localhost:(none)> use zhucong;
You are now connected to database "zhucong" as user "root"
Time: 0.002s
mysql root@localhost:zhucong> show tables;
+--------------------+
| Tables_in_zhucong  |
+--------------------+
| users              |
+--------------------+
1 row in set
Time: 0.010s
mysql root@localhost:zhucong> select * from users;
+----+------+
| id | name |
+----+------+
0 rows in set
Time: 0.011s
mysql root@localhost:zhucong> insert into users(name)values('yang');
Query OK, 1 row affected
Time: 0.002s
mysql root@localhost:zhucong> select * from users;
+----+------+
| id | name |
+----+------+
| 1  | yang |
+----+------+
1 row in set
Time: 0.010s
mysql root@localhost:zhucong> _
```

③ 查看从服务器



④ 测试OK,其他如 增删改查 测试也是没问题的,主从搭建完毕

当然你还可以为从服务器分配具体的权限,比如只能允许读,写,改

eg. 授权代码如下,此处就不一一介绍了:

```
grant all/create/drop/insert/delete/update/select on *.* to lili@'%' identified
 by password--为lili用户授予在任何库任何表的 所有/创建/删除/插入/删除/更新/查询 权限
```

第六章：mysql优化

如下文图片未正常显示,请移步链接:https://blog.csdn.net/AnPHPer/article/details/81160630

1.常见的一些优化场景

①.合理的表字段设计 比如身份证,手机号码字段,字段长度是已知的,那么我们使用char就要比varchar好很多 在如时间字段,很多人喜欢用datetime来存储时间字段,然后这样却很浪费表空间,换成int会更好.

②尽量避免like模糊查询 like模糊查询基本为新手常用查询手段,然后在一张上百万数据量的表中,like会导致查询时间变得非常缓慢,如果非左匹配还会导致索引失效,推荐如下替代方式 使用系统函数

```
LOCATE(substr,str)[第一个语法返回字符串 str中子字符串substr的第一个出现位置。第二个语法返回字符串 str中子字符串substr的第一个出现位置, 起始位置在pos。如若substr 不在str中, 则返回值为0。]
```

```
INSTR(str,substr)[返回字符串 str 中子字符串的第一个出现位置。这和LOCATE()的双参数形式相同, 除非参数的顺序被颠倒。]
```

替代like

③随机取出某张表的几条数据,一般开发人员的写法为:

```
SELECT * FROM tabName WHERE  ORDER BY RAND() LIMIT N; --数据量上百万,这样的一条SQL语句将会在扫描表和获得随机数上浪费大量时间
 优化思路为可以使用子查询,嵌套查询
```

```
SELECT * FROM `tableName` WHERE id >= (SELECT floor(RAND() * ((SELECT MAX(id) FROM`tableName`) - (SELECT MIN(id) FROM `tableName`)) + (SELECT MIN(id) FROM `tableName`))) ORDER BY id LIMIT N
```

这样即使上千万数据的表也能实现秒级查询

2.浅谈一些优化技巧,优化分别从以下几个步骤入手:

- 获取有性能问题的SQL
- 通过慢查询日志获取有性能问题的SQL
- 慢查询日志内容
- 实时获取有性能问题的SQL
- SQL预处理解析
- 如何确定查询消耗时间
- 优化特定的SQL

一、获取有性能问题的SQL

1. 通过用户反馈获取存在问题的SQL，此用户一般为测试人员，例执行某一个查询非常慢（不推荐）
2. 通过慢查询日志获取存在性能问题的SQL
3. 实时获取存在性能问题的SQL

以下着重介绍上述2,3点

二、通过慢查询日志获取有性能问题的SQL

相关参数：

① 启动/停止记录慢查询日志

```
set global slow_query_log = on/off # 还可以通过其他方式定时开关闭
```

② 指定慢查询的存储路径及文件名

```
slow_query_log_file #默认保存在mysql的数据目录中
```

③ 指定记录慢查询日志SQL执行时间的阈值，以秒为单位

```
long_query_time
```

其中MySQL慢查询日志记录的包括：查询语句，数据修改语句，事务回滚的SQL。慢查询默认时间为10s，通常修改为1毫秒

④ 是否记录未使用索引的SQL

```
log_queries_not_using_innndexes
```

三、两种方式分析慢查询日志内容

```
#user@Host:sbtest[sbtest]@localhost[] Id:7#Query_time:0.000233#lock_time:0.0001
20#Rows_sent:1              #扫描的数据行数#Rows_examined:1          #返回的数据行数S
ET timestamp = 1458612917 #SQL执行时间select id from sbtest where id = 1; #执行的
SQL
```

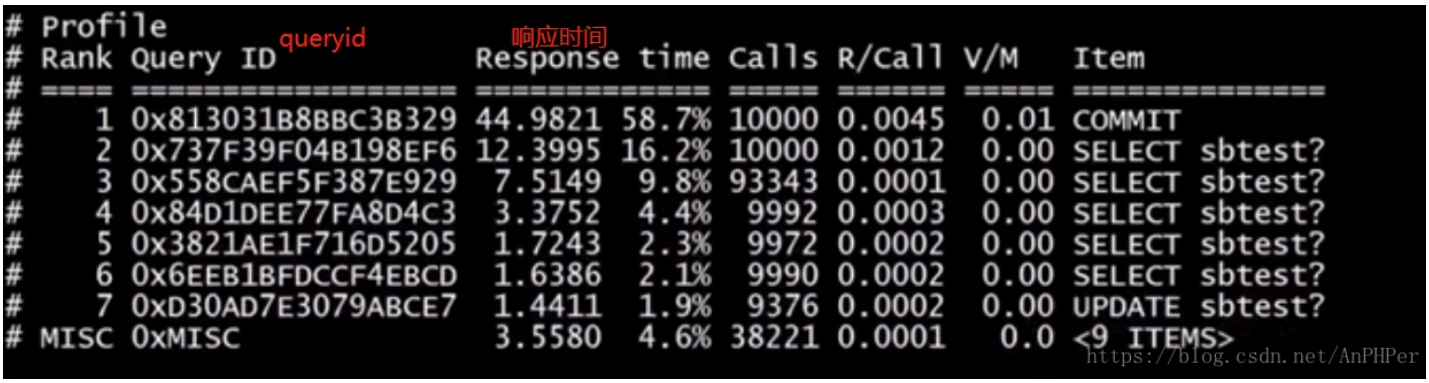
(1) 常用的慢查询分析工具1

mysqldumpslow：汇总除查询条件外其他完全相同的SQL，并将分析结果按照参数中所指定的顺序输出

命令：

```
mysqldumpslow -s r -t 10 slow -mysql.log #-t 表示 按总时间。还有以下几个参数，c:总次
```

数。t:总时间。l:锁的时间。r:总数据行。at,al,ar:表示t,l,r的平均数



(2) 常用的慢查询分析工具2

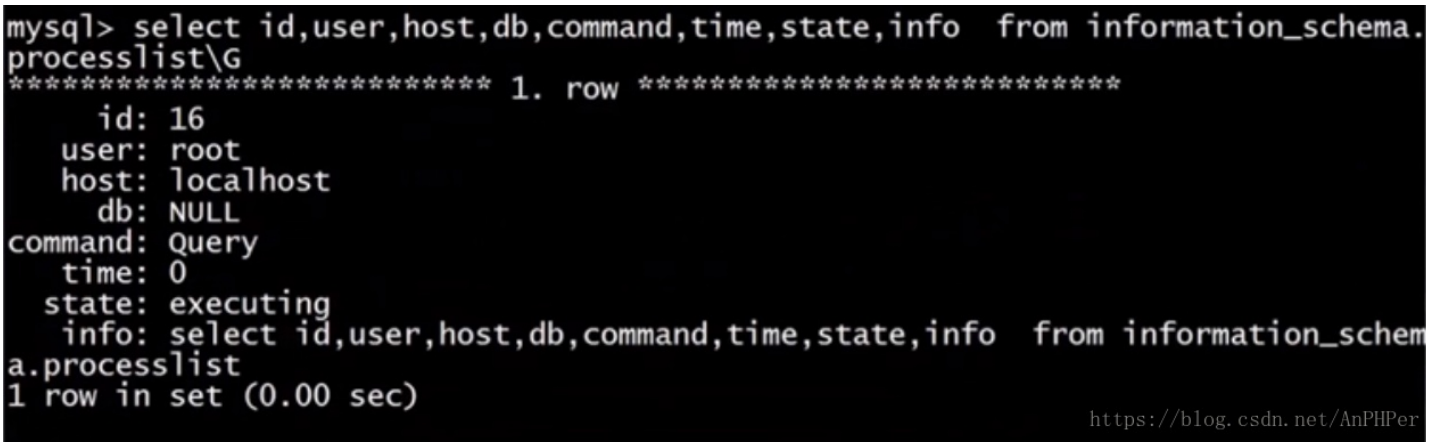
pt-query-digest : 使用慢查询日志获取有性能问题的SQL并生成查询报告

命令：

```
pt-query-digest \--explain h=127.0.0.1,u=username,p=password \slow -mysql.log
```

四、实时获取有性能问题的SQL

利用MySQL下 information_schema 库下的 processlist 表，实时获取有问题的SQL



五、SQL解析预处理

MySQL执行查询语句的流程为：客服端发送SQL请求，SQL服务器判断是否命中缓存，MySQL进行SQL解析预处理，优化器生成对应的执行计划，根据执行计划调用API查询数据，最后将结果返回给客户端。

(1) 打开查询缓存（对于一个读写频繁的系统使用查询缓存很可能会降低查询处理效率）

```
query_cache_type      #设置缓存是否开启或关闭query_cache_size      #设置查询缓存的内存大小query_cache_limit     #设置查询缓存可用的存储最大值query_cache_wlock_invalidate #设置数据表被锁后是否返回缓存中的数据query_cache_min_res_unit #设置查询缓存分配的内存最小单位
```

(2) SQL预处理阶段

检查SQL语法是否正确并生成查询计划

MySQL生成错误的查询计划可能的原因：

① 统计信息不准确

② 查询计划中的成本估算不等同于实际计划成本

③ MySQL优化器认为的最后与实际存在一定偏差

造成以上的原因可能为：

① MySQL不考虑其他并发查询

② MySQL的既定规则

③ 调用了MySQL存储过程及用户自定的函数

比如执行了一条错误的SQL，select id from a where id = -1;

(3) 查询优化器支持的SQL

① 重新定义表的关联顺序

② 将外连接转化成内连接

③ 使用等价变化规则 如 where id.>5 and id =5 转换为 where id>=5

③ 优化 count(id) min(id) max(id) 统计行的最大最小值等

④ 将一个表达式转换为常数，比如在求某一个时间字段时，不用系统函数而采用给定的固定值

⑤ 子查询优化，可以将子查询转化为关联查询

⑥ 对 not in ， <>等的优化

六、两种方式确定查询消耗时间

(1) profile (mysql5.5以前版本)

```
set profile = 1;         #启动profile###执行查询语句show profiles;          #查看每一个查询语句消耗的总的时间信息show profile for query query_id #查询某个query_id在每个阶段消耗的时间show profile cpu query query_id #查询某个query_id在每个阶段消耗的CPU信息
```

(2) performance_schema(mysql5.5以后版本)

① 启动proformance_schema

```
UPDATE `setup_instruments`
SET enabled='YES',TIMED='YES'  WHERE NAME LIKE 'stage%';

UPDATE setup_consumers
SET enabled='YES'  WHERE NAME LIKE 'events%';
```


② 查看其它线程所消耗的时间

```
SELECT a.THREAD_ID, SQL_TEXT,c.EVENT_NAME,(c.TIMER_END -
        c.TIMER_START)/1000000000 AS'DURATION (ms)'
FROM events_statements_history_long a
JOIN threads b ON a.`THREAD_ID`=b.`THREAD_ID`
JOIN events_stages_history_long c ON c.`THREAD_ID`=b.`THREAD_ID`
AND c.`EVENT_ID` BETWEEN a.EVENT_ID AND a.END_EVENT_ID
WHERE b.`PROCESSLIST_ID`=CONNECTION_ID()
AND a.EVENT_NAME= 'statement/sql/select'
ORDER BY a.THREAD_ID,c.EVENT_ID
```


七、举例优化特定的SQL

(1) 在千万级的表中删除或修改百万行数据（分批次修改，并单次修改完间隔几秒再执行操作）

大表的更新和删除

```
DELIMITER $$
USE `imooc`$$
DROP PROCEDURE IF EXISTS `p_delete_rows`$$
CREATE DEFINER=`root`@`127.0.0.1` PROCEDURE `p_delete_rows`()
BEGIN
        DECLARE v_rows INT;
        SET v_rows = 1;
        WHILE v_rows >0
        DO
                DELETE FROM sbtest1 WHERE id >= 90000 AND id <= 190000 LIMIT 5000;
                SELECT ROW_COUNT() INTO v_rows;
                SELECT SLEEP(5);
        END WHILE;
END$$
DELIMITER ;
```


(2) 对大表的结构进行修改

```
pt-online-schema-change  \
--alter="MODIFY c VARCHAR(150) NOT NULL DEFAULT ""  \
--user=root --password=PassWord  D=imooc,t=sbtest4  \
--charset=utf8 --execute
```

pt-online工具修改大表结构

(3) 优化 not in，<>查询 ，例：

原SQL：

```
select aid from a where aid not in (select aid from b)
```

优化后SQL:

```
select aid from a aa left join b bb on aa.aid = bb.bid where bb.bid IS NULL
```

(4) 优化主键最左匹配

例：

```
原：select aid from a where addtime > '2018-07-22' and aid >10;优：select aid fr
om a where aid > 10 and addtime > '2018-07-22';
```