========================

MySql 复制技术架构讲义：

========================

一、mysql AB 复制技术(主从)（异步复制）

MySQL支持单向、异步(async)复制，复制过程中一个服务器充当主服务器，而一个或多个其它服务器充当从服务器。

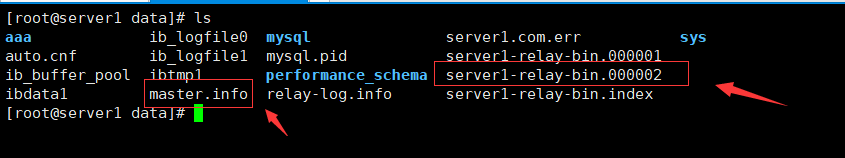
mysql主 - - > mysql从

master slave

把主上的二进制日志(bin-log)的内容传到从上的一个新的日志叫relay-bin-log

从上的 IO 线程 负责传输

从上的 SQL 线程 负责从服务器解析日志



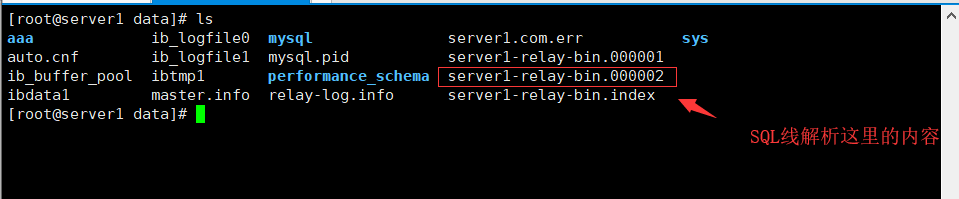
复制的过程：

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 bin 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 bin 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；也就是说salve执行和master一样的操作而达到数据同步的目的



架构图：

客户

｜

｜

web程序，游戏程序（c,php,java.......)

|

|

代理层 (mysqlproxy,amoeba)

|

|

mysql主 ----> mysql从

思考回答题：

1，主打开二进制日志，从要不要开二进制日志?

2，主和从的读写情况怎么分开？

=================================================================

二、一主多从复制构架

－－> salve

master －－> salve

－－> salve

master ——负责写

salve ——负责读

--适合于以读为主的业务，使用多个salve分担读的压力，但要注意的是这种架构，salve越多，那么master复制的压力就越大

=================================================================

三、双主复制架构

master A <－－> master B

discuz

双主架构最大的优点是，两台服务器实现数据互通，两台即能写又读，在任何一台服务器上写数据，另一台都会同步

思考题：

1、双主mysql复制构架，能同时写相同的数据吗?

2，两边可以写不同的数据吗？

可以通过业务程序层，指定一些表的写操作全在一端，另一些表的写操作全在另一端，也就是说两边不会同时写相同的表（当然这是理想状态，因为业务复杂的话会有表的联结等情况）

然后通过指定mysql复制的参数，一部分表或库会从A复制到B，另外一部分表或库从B复制到A就可以避免上面的问题了

或者两个库，一个库master A来写，另一个库master B来写

==========================================================================

前面三个问题也说明了双主架构的第一种应用（就是两边写不关联的数据，互相复制)，双主架构的第二种应用就是解决一主一从架构里主挂掉了的问题。

看下面的分析

mysql主 mysql从

思考：

问题1:如果一主一从，主挂了，slave能不能写，如果能写，主修好启起来后，salve写的数据如何传回给主?

答案:主挂了，slave应该要写，否则论坛不能发贴，只能看贴；

主修好后（假设一小时)，slave写了这一小时的数据应该要想办法传回主才行。

方法一：DBA人为的把salve这一小时写的数据找出来再导入到主(如果很多表，一小时的操作也非常多，可以说几乎不可能找出这小时改变了啥）

方法二：使用类似rsync的同步方法,这是通过存储层来实现数据同步回去。备：是把整个数据目录复制过去。

方法三：DBA人为的把从这一小时改变的二进制日志（要确认slave打开二进制日志)给找出来然后应用到主上去

方法四：直接把一主一从架构改为双主，或者把一主一从反过改成一从一主

问题2:上面的方法无论哪一种都会有一个问题，就是把从写的一小时数据传回给主需要一定的时间

就是假设这一小时slave创建了一个aaa表，主mysql修好启动后，

如果主马上接管从的写功能，这时前端程序马上给了一个insert into aaa的插入操作，这样肯定会出问题，因为创建aaa表的操作还没有从slave那传回来,如何解决?

解决方法一:人为控制主现在不能写，等slave的数据传回来后再写；但这样的话，等待时间太长了

解决方法二:就是把一主一从的架构改成双主的架构（假设master A和master B)

master A ---> master B (平时主B只读，不写，只有到主A挂了，才会写)

挂一个小时

读写一个小时

修复后启动了

把读给A 写留给B

问题3:改成双主后，master B在master A挂掉的一小时内帮他写了数据，A修好启起来后，B的数据可以自动传回A；但是如果数据量大的话，这个传回也可能需要一定时间，如果正好在这个传回时间内也出现了问题2的情况，又如何解决?

解决方法一:人为控制主不能写，这样的话等待时间很短（因为是自动传回来，跟手动导回来速度快很多)，影响不大

解决方法二:就直接把master B 做为新的写服务器，master A启动后，也只是做为读服务器；除非等下一次B挂了，A会重新成为写服务器（当然这是在两台服务器性能相等的情况下)

总结:上面说明双主架构的两种应用

-------------------------------------------------－－－－－

－－ salve －－ salve

四、级联架构 master －－ salve －－ salve

－－ salve －－ salve

bin-log bin-log ?

为了解决一主多从的master的复制压力

可以把slave做为下一级从机的master

增加复制的级联层次，造成的延迟更长

所以这种架构并不是很推荐，一般会进行分拆集群

--------------------------------------------------

五、双主和级联合起来

master ----salve --slave

|

| －－ slave

master －－ salve －－ slave

－－ slave

------------------------------------------------------

优化原则:分

大型结构的数据量太大，要用到数据切分(水平，垂直)

mysqlproxy

amoeba

===========================================================================

mysql 主从复制技术

开始实验

1、关闭iptables,selinux,NetworkManager

2、将主机名和IP写进hosts

3、使用yum rpm包或tar安装数据库，主从两台机安装一致

4、主：开启二进制日志，用于记录对表的增删改操作，并流传给从

5、主从设置server-id值，主的值比从要小

6、主：建立复制的帐号并授权

master A 主服务器ip：192.168.224.10

slave B 从服务器ip： 192.168.224.11

第一步；安装mysql数据库并配置

yum源安装：yum -y install mariadb\*

启动数据库：systemctl start mariadb

修改root初始化密码：mysqladmin -uroot password 123;

修改主、从配置文件 /etc/my.conf

主：master:在[mysqld]标签下加入下面二句

［mysqld]

log-bin=mysql-bin

server-id=1

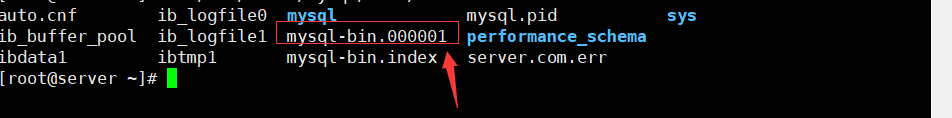
备：slave：在[mysqld]标签下加入下面一句

[mysqld]

server-id=2

完成配置后重新启动mariadb服务 systemctl restart mariadb

主服务器上ll /var/lib/mysql/ 看下是否已经有二进制日志,mysql-bin.000001



第二步：建立复制的帐号并授权

主:master操作：建立aa帐号，密码111

mysql> grant super,replication slave on \*.\* to 'aa'@'%' identified by '111';

mysql> flush privileges;

mysql8.0 后版本,先创建用户密码为Mysql138#$

mysql> create user 'aa'@'%' identified with mysql\_native\_password by 'Mysql138#$';

然后授权

mysql> grant super,replication slave on \*.\* to 'aa'@'%';

Query OK, 0 rows affected, 1 warning (0.03 sec)

授权完成后，在从服务器上登陆测试 登陆成功后就退出

mysql -uaa -h192.168.224.10 -p111

备：-h后面接IP为登陆服务器的IP

第三步：给master加上锁，防止有新的数据写入

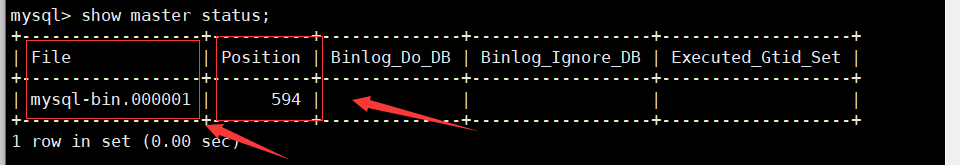
mysql> flush tables with read lock;

查看二进制日志记录的位置，配置从服务器要用,只要服务器开启二进制日志后才允许当主

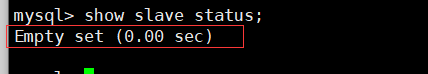
show master status; --查看主服务器配置

show slave status; --查看从服务器配置

mysql> show master status;



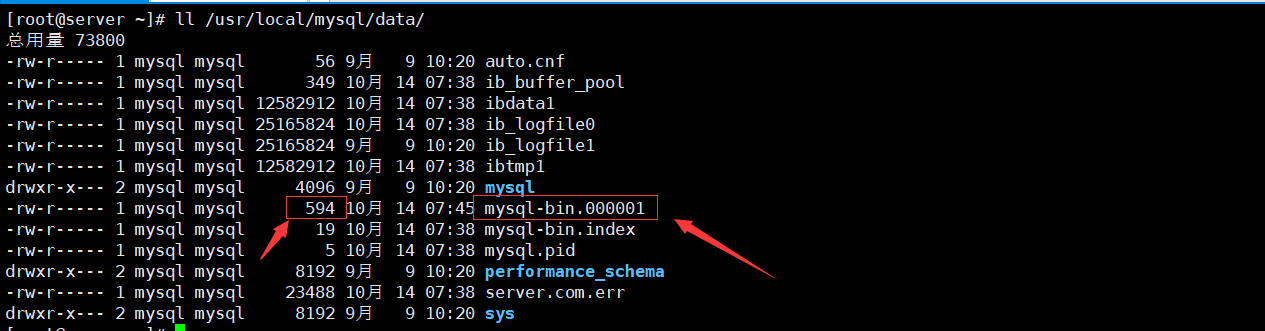
备：尝试在主服务器上使用命令show slave status;查看是否有数据，因这台服务器是主服务器，所以条命令是看不到任何东西，是空的。



mysql> exit

查看主服务器的日志文件名是否和上面查出来的一致，配置从服务器要用

[root@master mysql]# ll /var/lib/mysql



# 对照文件名是否一致

第四步：在从服务器slave端，配置mysql主从关系

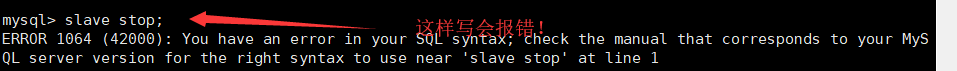
切到从服务器192.168.224.11上面

先登陆数据库

# mysql -p

停止从服务器复制线程：备：默认该线程就是停止状态，执行该命令是告诉数据库该线程明确的要停止。

mysql> stop slave;



配置主从关系：备：该台服务器为从时，需要配置连接主服务器的信息，从服务器能不能在主服务器上面获取二进制日志，全靠下面的配置信息。

mysql> change master to --配置从服务器连接主服务器

-> master\_user='aa', --主服务器建立复制的帐号

-> master\_password='111', --密码

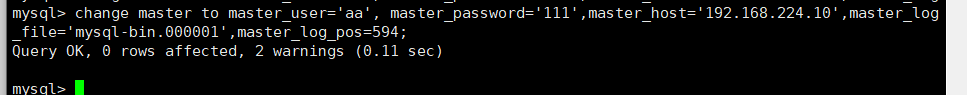
-> master\_host='192.168.224.10', --主服务器IP

-> master\_port=3306, --端口，注：不能加引号

-> master\_log\_file='mysql-bin.000001', --主上面查到的文件名

-> master\_log\_pos=594; --主上面查到的位置号

change master to master\_user='aa',master\_password='111',master\_host='server.com',master\_log\_file='mysql-bin.000001',master\_log\_pos=594;

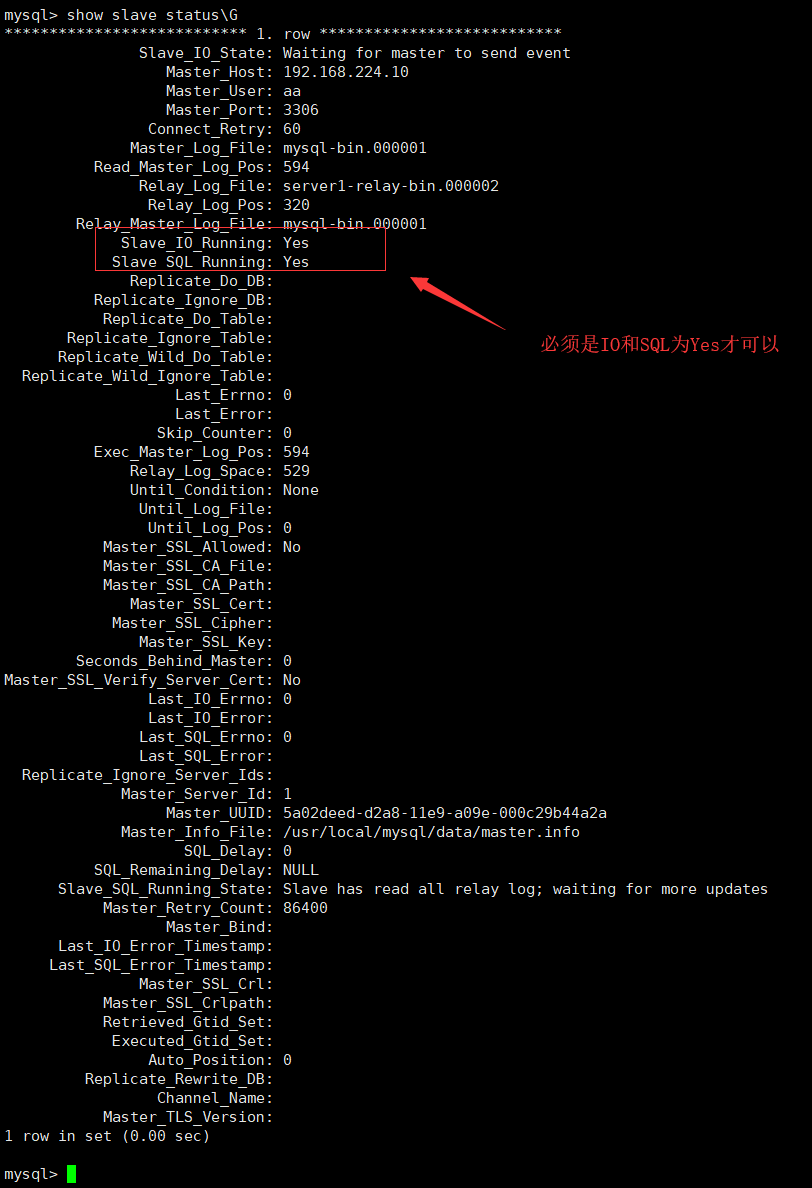


配置连接主服务器信息后，在从服务器上启动slave线程。

mysql> start slave; --启动命令

启动完成后，再查看slave线程连接主服务器的状态

mysql> show slave status\G



第五步：在主服务器master端解锁mysql：

mysql> unlock tables;

第六步：测试

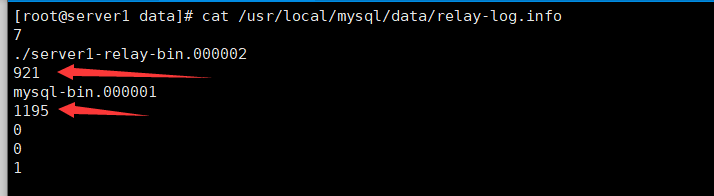
在主服务器上创建库、表和插入表数据后，在从服务器查看是否有相同数据

第七步：查看日志文件：在从服务器上查看(192.168.224.11)

# cat /var/lib/mysql/master.info



# cat /var/lib/mysql/relay-log.info



# mysqlbinlog /var/lib/mysql/mysqld-relay-bin.000002

题外：

尝试登陆从服务器:192.168.224.11

# mysql -uroot -p123456

在从服务器上面登陆库、表并使用insert插入数据。

这时再登陆主服务器:192.168.224.10

# mysql -uroot -p123456

在主服务器上查看是否有从服务器创建和插入的数据。

备：只有master写，slave可以看到，slave写，master看不到

如果复制出现问题

（要模拟问题的话，在从上创建一个库，然后在主上也创建这个库，就会冲突，造成复制出现问题），

重做复制集群只需要重新在从上执行stop slave; change master to ...... ; start slave;

测试：把从重启后，再上去查看状态，还是连接的，没什么影响

把主重启后，再去slave上去查看状态，发现重试时间为60秒，等60秒后又自动连接OK了

--如果想要改这个60秒的话，可以把从服务器上的master.info文件里的60直接改成30(rpm版测试有点问题)

========================================================================

1主多从的做法

->slave

master

->slave

就是上面的步骤有几个从就做几次

======================================================================

双主架构

master - master

把上面的1主1从，反过来做一遍，在上面的slave机也打开二进制日志，反着做一遍就行，注意做的过程中，保证数据一致

做好的架构，只要两边不同时操作相同的数据，就不会出现复制不一致的问题；

或者是在业务程序层，将一部分的写操作定义到master A,另一部分的写操作定义到master B

================================================================================

master A 主服务器ip：192.168.224.10

slave B 从服务器ip：192.168.224.11

slave C 从服务器IP：192.168.224.12

级联架构

master A -slave B -slave C

把中间的从也要打开二进制日志。但是它默认不把应用master的操作记录到自己的二进制日志。所以需要打开一个参数让它记录，才可以传给第三级的从

本文需要验证的疑问：

从库做为其他从库的主库时 log-slave-updates参数是必须要添加的，因为从库要作为其他从库的主库，必须添加该参数。该参数就是为了让从库从主库复制数据时可以写入到binlog日志，为什么要用这个参数写binlog日志呢，不是在配置文件中开启log-bin =mysql-bin选项就可以吗？

答：从库开启log-bin参数，如果直接往从库写数据，是可以记录log-bin日志的，但是从库通过I0线程读取主库二进制日志文件，然后通过SQL线程写入的数据，是不会记录binlog日志的。也就是说从库从主库上复制的数据，是不写入从库的binlog日志的。所以从库做为其他从库的主库时需要在配置文件中添加log-slave-updates参数。

在/etc/my.cnf文件里加上以下二句：

max\_allowed\_packet=20M --代表应用的日志大小

log-slave-updates=1 --代表可以读取上一级从服务器应用主服务器的日志

在服务器里查看参数

show variables like '%max\_allowed%';

show variables like '%log\_slave%';

架构图：

master A slave B slave C

192.168.224.10 192.168.224.11 192.168.224.12

sever-id=1 server-id=2 server-id=3

log-bin=mysql-bin log-bin=mysql-bin

log-slave-updates=1 log-slave-updates=1 这台不需要加这个参数

日志应用流程：

bin-log relay-bin-log relay-bin-log

->master A 创建一个库时日志会写进bin-log(二进制日志)

->这时slave B去master A读取bin-log再将应用日志放到本地的relay-bin-log日志里

->slave C这时去读取slave B的bin-log和relay-bin-log日志(因为slave B添加了允许其它服务器读取我的应用日志log-slave-updates=1)

=========================================================================

2主多从+级联

2主就是把前面的双主架构做一遍，多从就是在新服务器上把从服务做一遍，如果要加级联就是再把级联架构做一遍。

=========================================================================

=======================

mysql多源复制

=======================

警告：mysql多源复制技术仅支持5.7以上版本mysql \*\*\*\*\*\*

1、多源复制的特性

1.1 和主从复制技术一样，都是通过mysql的二进制日志（binlog）来实现数据的同步

2、什么地方需要用到多源复制

2.1 在此之前，我们讲到了mysql主从复制技术，但考虑一下，如果我有两台不同数据的mysql服务器，该如何才能实现同时使用一个slave进行数据同步备份呢？

提示：在做多源复制之前先把之前搭建好的双主关掉

参数讲解：

master端

gtid-mode=on # 是否开启多源同步技术

enforce-gtid-consistency=1

skip\_slave\_start=1

log-bin=mysql-bin

server-id=1 # 指定mysql主从ID（该ID在主从集群中是唯一的）

slave端

server-id=3 # 指定mysql主从ID（该ID在主从集群中是唯一的）

gtid-mode = on # 是否开启多源同步技术

binlog\_gtid\_simple\_recovery=1

enforce\_gtid\_consistency=1

master\_info\_repository=TABLE

relay\_log\_info\_repository=TABLE

replicate\_ignore\_db=mysql

skip\_slave\_start = 1

一、配置文件：

Master1和Master2:

#GTID

gtid-mode = on

binlog\_gtid\_simple\_recovery=1

enforce\_gtid\_consistency=1

binlog\_format = row

skip\_slave\_start = 1

log-bin = mysql-bin

server-id=1 # master1和master2的id不能相同

Slave:

#binlog

binlog\_format=row

server-id=3

log-bin=mysql-bin

#GTID

gtid-mode=on

binlog\_gtid\_simple\_recovery=1

enforce\_gtid\_consistency=1

#修改MySQL存储master-info和relay-info的方式，即从文件存储改为表存储

master\_info\_repository=TABLE

relay\_log\_info\_repository=TABLE

replicate\_ignore\_db=mysql　　#忽略mysql库的同步

skip\_slave\_start=1

二、主库创建复制账号：

Master1：grant replication slave,replication client on \*.\* to aa@'%'identified by 'aaa';

Master2：grant replication slave,replication client on \*.\* to bb@'%'identified by 'bbb';

8.0版本，先创建用户

master1:

mysql> create user 'aa1'@'%' identified with mysql\_native\_password by 'Mysql138#$';

再授权

mysql> grant replication slave,replication client on \*.\* to 'aa1'@'%';

master2:

mysql> create user 'bb1'@'%' identified with mysql\_native\_password by 'Mysql138#$';

mysql> grant replication slave,replication client on \*.\* to 'bb1'@'%';

三：从库启动复制

mysql> change master to master\_user='aa1',master\_password='Mysql138#$',master\_host='lzx4.com',master\_port=3306,master\_auto\_position=1 for channel'Master\_1';

mysql> change master to master\_user='bb1',master\_password='Mysql138#$',master\_host='lzx1.com',master\_port=3306,emaster\_auto\_position=1 for channel'Master\_2';

5.7

mysql> change master to master\_host='server.com',master\_port=3306,master\_user='aa',master\_password='aaa',master\_auto\_position=1 for channel 'Master\_1';

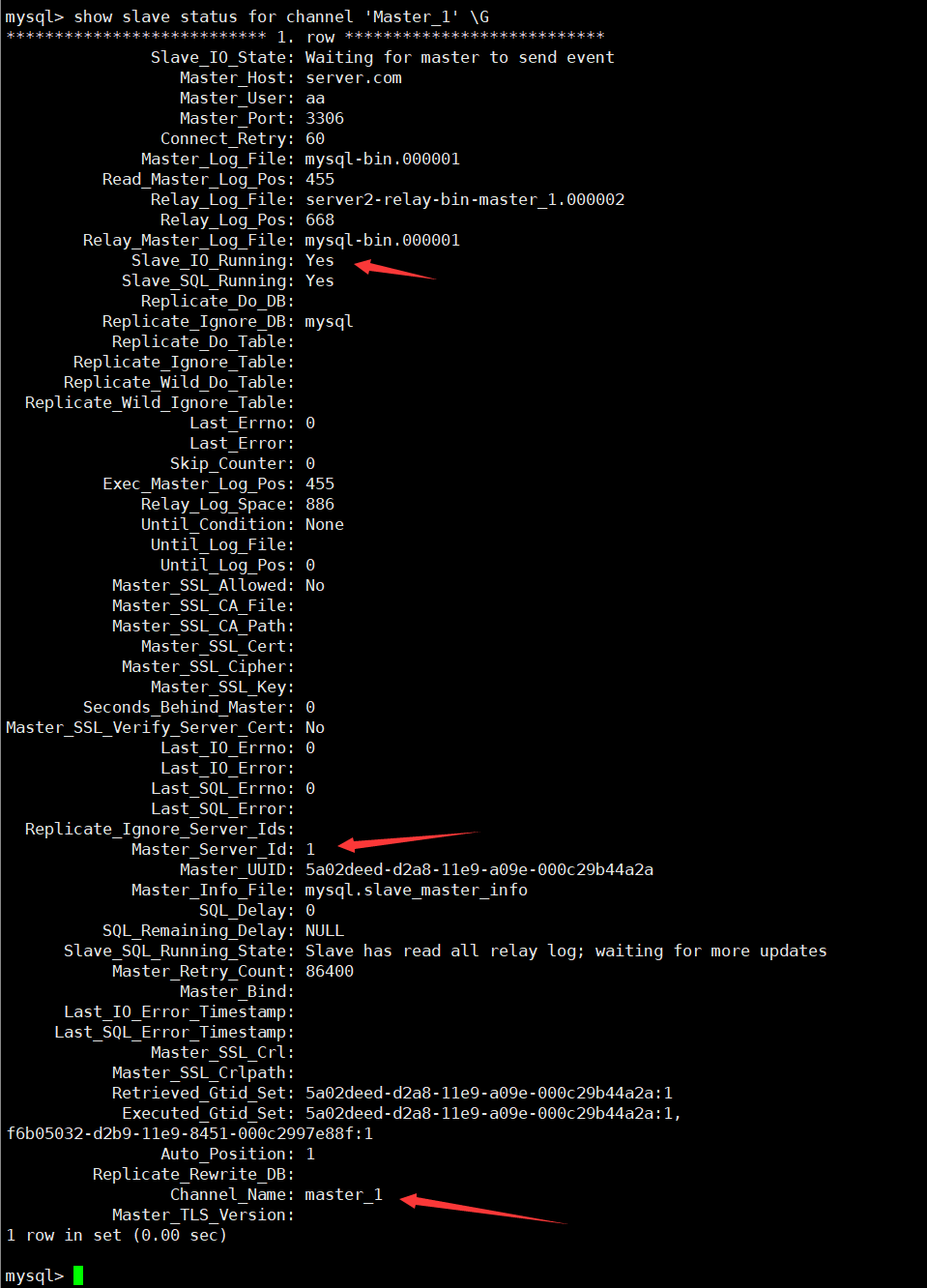
mysql> change master to master\_host='server1.com',master\_port=3306,master\_user='bb',master\_password='bbb',master\_auto\_position=1 for channel 'Master\_2';

mysql> start slave;

也可以start slave for channel 'Master\_1 '启动单个channel的复制。

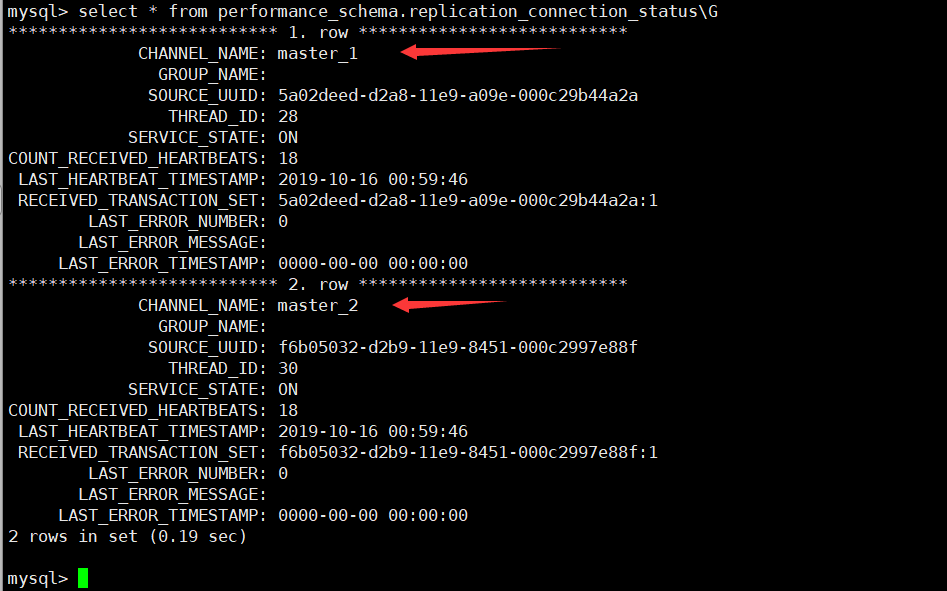
四：查看slave状态

【也可以直接show slave status for channel 'Master\_1' \G查看各个channel的复制状态】



通过查看performance\_schema相关的表查看同步状态：

mysql> select \* from performance\_schema.replication\_connection\_status\G



五、数据同步检查：

Master1建库建表并插入数据：

mysql> create database master1;

mysql> use master1;

mysql> create table test1(id int);

mysql> insert into test1 values(1);

Master2建库建表并插入数据：

mysql> create database master2;

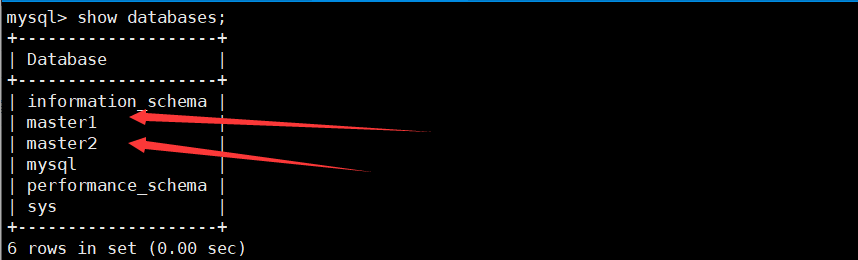
mysql> use master2;

mysql> create table test1(id int);

mysql> insert into test1 values(1);

Slave查看数据是否同步：

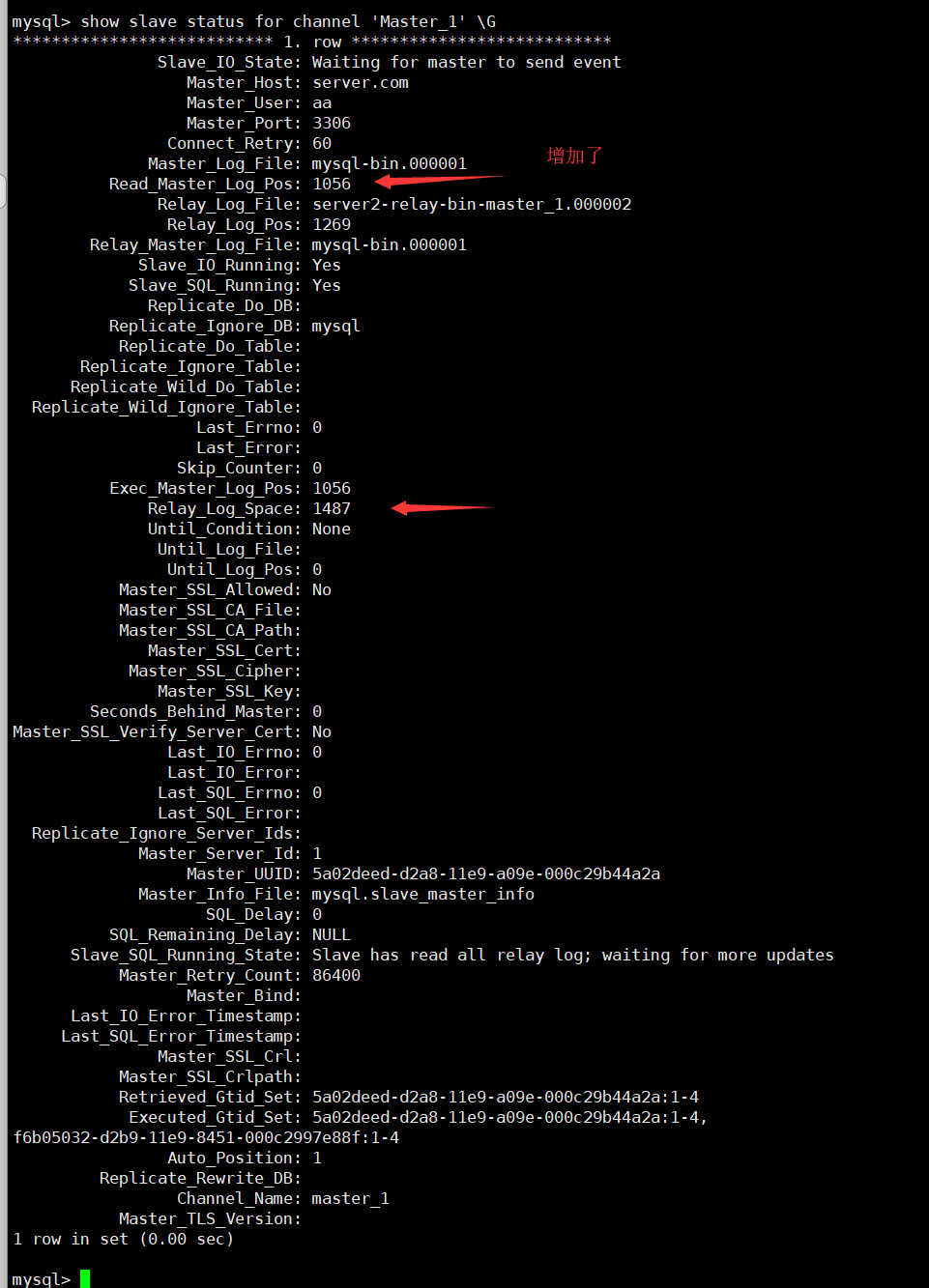
mysql> show databases;



mysql> select \* from master1.test1;

mysql> select \* from master2.test2;

再次查看从库状态：



mysql> show slave status\G

=======================

Mysql 基于表复制技术

=======================

复制库、表和表内容参数：可使用show slave status\G 命令看到以下参数

Replicate\_Do\_DB: --接受哪个库的复制：必须在指定的库下操作才能复制。包括表结构和表数据。

Replicate\_Ignore\_DB: --忽略哪个库的复制

Replicate\_Do\_Table: --接受哪个表的复制：注：只能指定某个库下的某个表的内容更新与修改，例：aaa.a3

Replicate\_Ignore\_Table: --忽略哪个表的复制

Replicate\_Wild\_Do\_Table: --通配符表示复制哪些表：注：接收某个库下的所有表内容的修改，例:aaa.%

Replicate\_Wild\_Ignore\_Table: --通配符表示忽略哪些表



实例说明：

master A slave B

1、先把两台做成ab复制的架构

此时在master上做任何创建库、表或插入数据等动作，从服务器都会复制过来。

2、现在的业务需求是：

要求：把master的aaa库下的表复制到slave上，其它的库都不要

3、为了满足需求我们在slave b的my.cnf配置文件的[mysqld]参数组里加上

replicate\_do\_db=aaa --只接受aaa库的复制

4、重启slave b，然后测试

测试结果为

->在master上创建aaa库:create database aaa;

->在master创建aaa.a1表:create table aaa.a1(id int(1));

->并插入数据insert into aaa.a1 values(1);

->回到slave上查看有aaa库，也有a1表，但没有数据

如果要同步aaa.a1表下的数据，master要进到aaa库下操作才可以

->在master上操作，use aaa;然后再插入数据：insert into aaa.a1 values(2);

->回到slave上查看aaa库的a1表数据，这时就有了

如果想要复制该库下所有的表数据，这时就要加入另外一个参数

5、解决方法：

在slave b上的my.cnf文件的[mysqld]标签下加上以下参数，就可以复制aaa库下的所有表和表内容了

replicate\_wild\_do\_table=aaa.% --%是通配符

6、重启slave b 然后测试

测试步骤：

->在master创建aaa.a2表:create table aaa.a2(id int(1));

->并插入数据insert into aaa.a2 values(1);

->并给aaa.a1表也插入数据：insert into aaa.a1 values(2);

->回到slave上查看aaa库下是否有a2表，a2和a1表是否有上面插入的数据。

表复制技术完成

=================================================================================

=============================

Mysql复制技术 之 半同步技术

=============================

这个技术是Mysql5.5版本之后的新功能。

前面我们学到的MySQL复制技术，是叫异步技术：

异步技术的概述：

--------------

在主服务器上每执行完一条事务命令后都会写进二进制的日志，从服务器通过IO线程监控主服务器的二进制日志，每当主服务器的二进制日志发生改变，从服务器就会启动复制

异步技术的特点：

--------------

是允许主从之间的数据存在一定的延迟，对网络要求不高，这样设计的目的是基于数据库的高可用性，为了保证master不受slave的影响，并且异步复制使得master处于一种性能最优的状态

缺点：

在主从架构中使用异步技术，如果master停机，会发生刚刚已提交的事务，slave未能及时复制过来的可能。

半同步技术特点：

--------------

master每操作一个事务,要等待slave应用这个事物后给master确认信号。这样master才能把操作成功执行。这样保证了主从数据的绝对一致，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

下面开始配置半同步复制技术：

==========================

环境要求:

搭建好 mysql主从异步复制

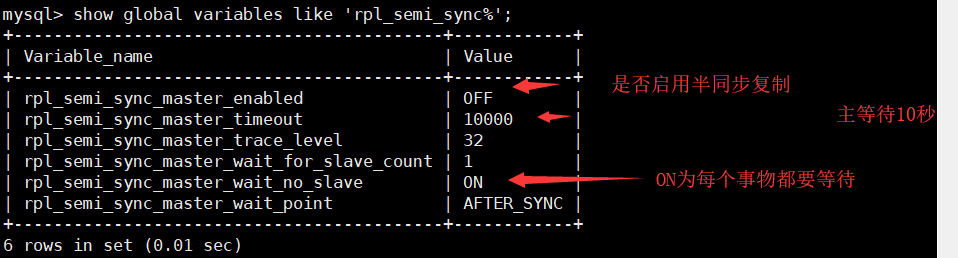
1、在master上安装半同步插件

---------------------------

mysql > install plugin rpl\_semi\_sync\_master soname 'semisync\_master.so';

备注：删除主服务器插件的方法 mysql > uninstall plugin rpl\_semi\_sync\_master;

mysql > show global variables like 'rpl\_semi\_sync%'; --安装成功后，会查询出以下4项参数



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

| 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。

| rpl\_semi\_sync\_master\_trace\_level | 32 | --监控等级，32代表的是等待

| 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

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

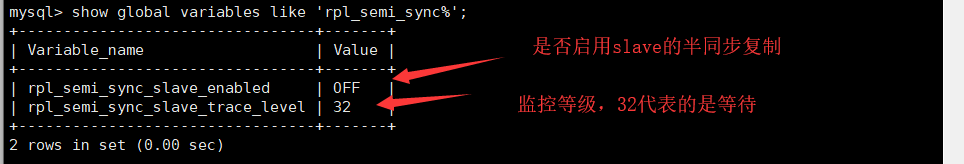
2、在slave上安装插件

--------------------

mysql > install plugin rpl\_semi\_sync\_slave soname 'semisync\_slave.so';

备：删除从服务器插件的方法 mysql > uninstall plugin rpl\_semi\_sync\_slave;

mysql > show global variables like 'rpl\_semi\_sync%'; --从服务器安装成功后，会查询出以下2项参数



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

| Variable\_name | Value |

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

| rpl\_semi\_sync\_slave\_enabled | OFF | ----是否启用slave的半同步复制

| rpl\_semi\_sync\_slave\_trace\_level | 32 | ----监控等级，32代表的是等待

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

3、在主服务器master开启半同步复制

---------------------------------

mysql > set global rpl\_semi\_sync\_master\_enabled=on; --开启命令

在master查看状态

master > show global status like 'rpl\_semi\_sync%';

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

| Variable\_name | Value |

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

| Rpl\_semi\_sync\_master\_clients | 0 | --启用半同步复制的slave数量，现在是0，

| Rpl\_semi\_sync\_master\_net\_avg\_wait\_time | 0 | --master等待slave回复确认的平均等待时间。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 | --master 等待超时的次数

| 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在处理每个事务的平均等待时间

| Rpl\_semi\_sync\_master\_tx\_wait\_time | 0 | --master在处理事物等待的总时间

| 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 | --表示这次事物成功从slave返回一次确认信号

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

4、在从服务器slave开启半同步复制

--------------------------------

mysql > set global rpl\_semi\_sync\_slave\_enabled=on; --开启命令

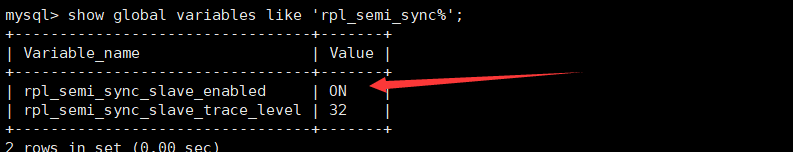
mysql > stop slave IO\_THREAD;

mysql > start slave IO\_THREAD; --重启IO

直接stop slave; start slave; 也可以

在slave上查看状态

slave > show global status like 'rpl\_semi\_sync%';



5、测试半同步：

-------------

在主服务器上往aaa.a1表里插入数据：

master > insert into aaa.a1 values (3);

mysql> show global status like 'rpl\_semi\_sync%'; --再查看半同步状态，和前面进行对比

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

| 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 | 814 | --master等待slave回复确认的平均等待时间。单位毫秒

| Rpl\_semi\_sync\_master\_net\_wait\_time | 814 | --master总的等待时间。单位毫秒

| Rpl\_semi\_sync\_master\_net\_waits | 1 | --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 | --master 等待超时的次数

| 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在处理每个事务的平均等待时间

| Rpl\_semi\_sync\_master\_tx\_wait\_time | 0 | --master在处理事物等待的总时间

| 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 | 1 | --表示这次事物成功从slave返回一次确认信号

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

14 rows in set (0.00 sec)

6、模拟错误，把slave上的mysql停掉 /etc/init.d/mysql56 stop

----------------------------------------------------------

--再回到master上往aaa.a1表里插入数据。

mysql > insert into aaa.a1 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)

mysql > insert into aaa.a1 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)

mysql> show global status like 'rpl\_semi\_sync%'; --再查看半同步状态

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

| Variable\_name | Value |

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

| Rpl\_semi\_sync\_master\_clients | 0 |

| Rpl\_semi\_sync\_master\_net\_avg\_wait\_time | 884 |

| Rpl\_semi\_sync\_master\_net\_wait\_time | 8845 |

| Rpl\_semi\_sync\_master\_net\_waits | 10 |

| Rpl\_semi\_sync\_master\_no\_times | 1 | --表示同步已经停止了一次

| Rpl\_semi\_sync\_master\_no\_tx | 3 | --有3个同步超时的事务

| Rpl\_semi\_sync\_master\_status | OFF | --表示同步状态为关闭状态

| Rpl\_semi\_sync\_master\_timefunc\_failures | 0 |

| Rpl\_semi\_sync\_master\_tx\_avg\_wait\_time | 1093 |

| Rpl\_semi\_sync\_master\_tx\_wait\_time | 8748 |

| Rpl\_semi\_sync\_master\_tx\_waits | 8 |

| Rpl\_semi\_sync\_master\_wait\_pos\_backtraverse | 0 |

| Rpl\_semi\_sync\_master\_wait\_sessions | 0 |

| Rpl\_semi\_sync\_master\_yes\_tx | 9 |

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

14 rows in set (0.00 sec)

再次把slave启动。

mysql> show global variables like 'rpl\_semi\_sync%';

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

| Variable\_name | Value |

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

| rpl\_semi\_sync\_slave\_enabled | OFF | --看到半同步复制已经关闭了，切换成了异步模式

| rpl\_semi\_sync\_slave\_trace\_level | 32 |

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

再执行下开启动作，才可以重新打开半同步复制模式。

mysql > set global rpl\_semi\_sync\_slave\_enabled=on;

mysql > stop slave IO\_THREAD;

mysql > start slave IO\_THREAD;

slave启起来后，查看表，发现刚才slave关闭期间的那几条数据还是会自动复制过来，数据又回到一致

=======

思考题：

=======

半同步技术模式：

在主服务器上只要执行：

set global rpl\_semi\_sync\_master\_enabled=on; 这条语句就能开启

在从服务器上只要执行以下3条命令能就开启

set global rpl\_semi\_sync\_slave\_enabled=on;

stop slave IO\_THREAD;

start slave IO\_THREAD;

一但主服务器或从服务器mysql服务停止或重启，半同步就会失效，有什么办法可以让他永久生效。

======================================================================

================

mysql的延时复制

================

延迟的复制特点：

可自行设置复制间隔时间，可以防止主节点数据误删，查看数据库历史状态等

在MySQL 的主从复制基础上操作：

命令语法：CHANGE MASTER TO MASTER\_DELAY = 30; #备：设置备节点延迟的时间，单位秒。

change master to master\_user='aa',master\_password='111',master\_host='server.com',master\_log\_file='mysql-bin.000001',master\_log\_pos=1663,master\_delay=30;

在slave从服务器上操作：

1、在slave从服务器上将slave服务停止

语法：mysql> stop slave;

2、配置延迟复制时间，不用重新配置，直接添加下面这个语法就可以了

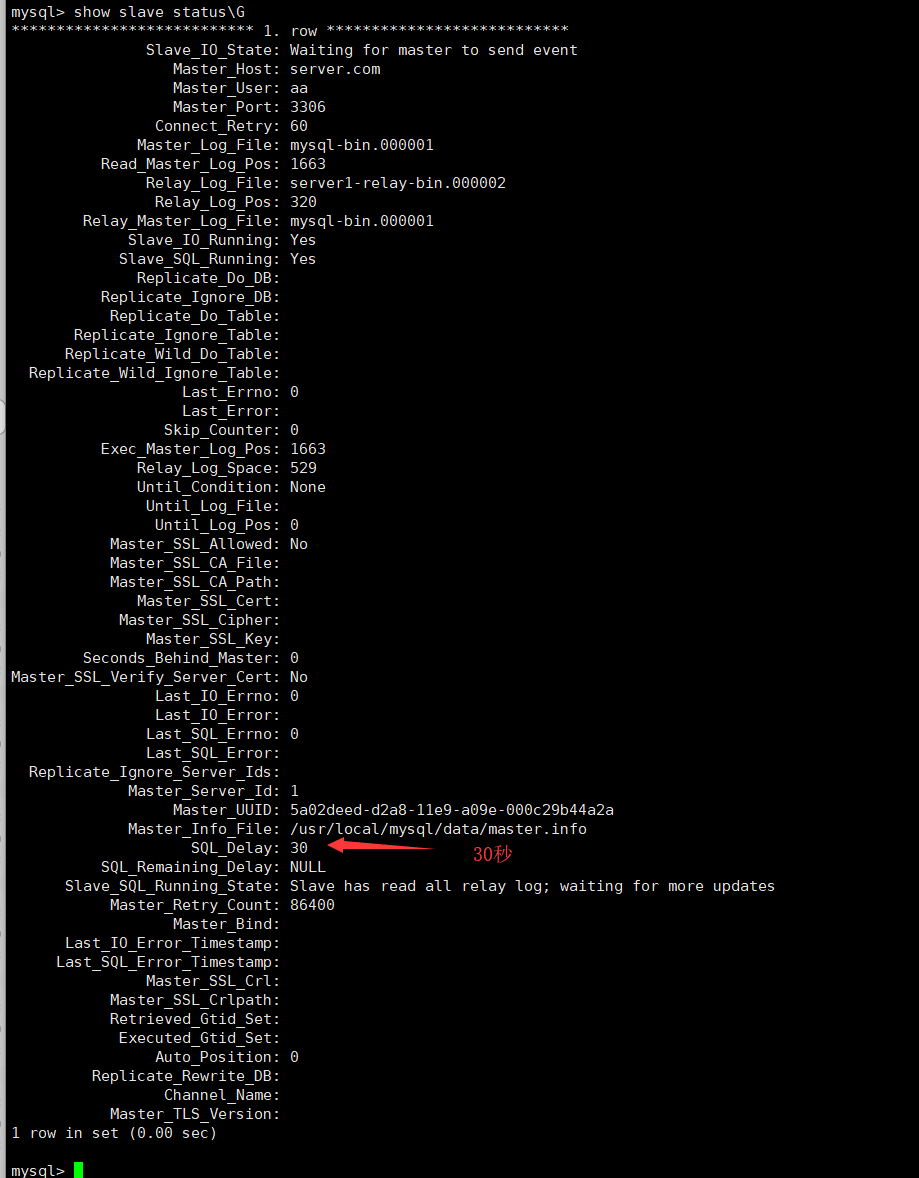
语法：change master to master\_delay=30;

3、启动slave线程

语法：mysql> start slave;

4、查看状态

语法：mysql> show slave status\G



已经将延时0秒变成了延时30秒。

4、测试，在master主服务器上插入1条数据

语法：mysql> insert into ccc.aa values(6);

->回到slave从服务器上查询主刚插入的数据：select \* from ccc.aa;

->主服务器插入完成后，马上回到从服务器查询是查询不到的，要等待30秒过后才可以

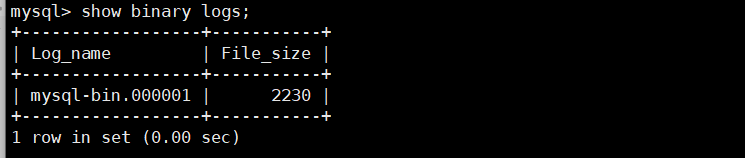
============================================

补充：mysql二进制日志管理

查看所有二进制日志：

1、直接查看数据目录下的文件

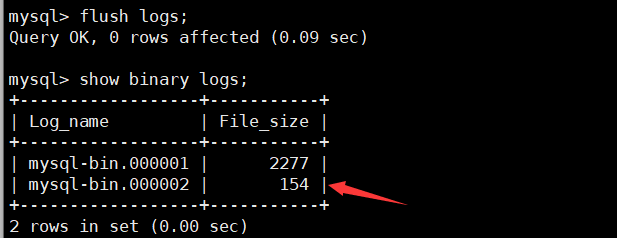
2、使用SQL语句：show binary logs;



二进制日志滚动：

1、每次重启MySQL都会促发一次二进制日志滚动，文件后缀名+1

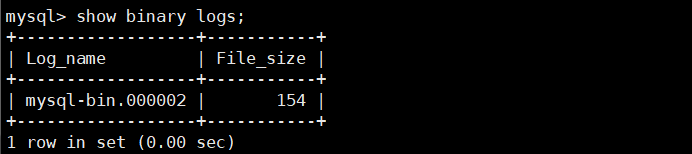
2、手动滚动；可使用 flush logs语句，执行后，二进制日志将会发生一次滚动，后缀名+1



二进制日志清除：

1、按文件名删除

mysql > purge binary logs to 'mysql-bin.000002'; --删除000002之前的二进制日志，不包含000002



2、按时间段删除

mysql > purge binary logs before '2017-03-10 10:10:00'; --删除指定时间之前的二进制日志

================================================================

作业题：

思考题：

假设在A上误删除一条数据，用二进制日志来恢复不太方便，因为日志里记录的是删除的操作，并没有记录这条数据具体是什么，所以你要在所有的日志里找到当初插入这条数据时的记录，这是很麻烦的。

原理理解题：

题1:mysql架构中,一主多从、双主、级联这三种的区别和优缺点

题2:某公司mysql数据库运行了一年，现在要求搭建mysqlAB复制，检查主库后，发现它这一年没有使用二进制日志，请问如何做复制？

题3:在实际大数据量的数据库中，因为负载太高或者网络问题，造成主从复制时有延时，此时你是DBA，请问如何处理？

题4: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迁移到惠州的机房

1，在惠州机房做一个lamp，mysql双主架构，web做rsync远程实时同步

2，把网站域名对应的IP地址A记录改成惠州的IP

3，当深圳机房服务器无连接时，就可以关闭深圳机房的服务器了

===============================结束线==================================