一、MySQL主从复制及读写分离

1.MySQL Replication 概述

MySQL Replication优点:

2.MySQL复制类型

1.异步复制(Asynchronous replication)

2.全同步复制(Fully synchronous replication)

3.半同步复制(Semisynchronous replication)

4.无数据丢失的半同步复制

3.MySQL支持的复制方式

基于SQL语句的复制(statement-based replication, SBR):

基于行的复制(row-based replication, RBR):

混合模式复制(mixed-based replication, MBR):

1.RBR的优点

2.RBR的缺点

3.SBR的优点

4.SBR 的缺点

4.MySQL主从复制的工作过程

5.复制过程的限制

实验

部署MySQL主从异步复制

1.所有机器上的操作:

2.在MySQL Master.上 的配置NTP时间同步服务器

3.安装mariadb

4.配置mysql master服务器

二、主从数据库相关命令

三、部署MySQL主从半同步复制

半同步复制支持多种插件:

半同步复制插件:

查看插件是否安装:

在主节点安装并启用插件:

查看插件允许状态

启动插件:

从节点启动IO线程:

在主节点创建数据库mydb:

在主节点创建数据表:

四、MySQL主主高可用方案

实验:

1.双主-互为主从

2.MySQL高可用方案

总结:

五、基于Amoeba读写分离

1.读写分离

2.MySQL读写分离方案

1.基于程序代码内部实现

2.基于中间代理层实现

实验:

环境建立在一主两从的环境上继续的

1、在主机Amoeba上安装java环境

2、安装并配置Amoeba

3、配置Amoeba读写分离，两个Slave读负载均衡

4.客户端测试 192.168.200.105

一、MySQL主从复制及读写分离

1.MySQL Replication 概述

MySQL Replication 俗称MySQL AB复制或主从复制（至少两台机器，一主一从）

是MySQL官方推荐的数据同步技术

数据同步基本过程为从数据库会实时去读取主数据库的二进制日志文件

按照日志中记录对从库进行同样的操作，以达到数据同步效果

MySQL Replication优点:

- 通过增加从服务器来提高数据库平台的可靠性能。
  在主服务器上执行写入和更新
  在从服务器上向外提供读功能
  可以动态地调整从服务器的数量，从而调整数据库平台的高性能

- 提高数据安全性，
  因为数据已复制到从服务器，（复制是实时的，主要干啥。从就要干啥 delete drop）
  主数据库数据异常时，
  可以将从服务器复制进程终止来达到保护数据完整性的特点

- 在主服务器上生成实时数据，
  而在从服务器上分析这些数据，
  从而缓解主服务器的性能

2.MySQL复制类型

1.异步复制(Asynchronous replication)

MySQL默认的复制即是异步的，

主库在执行完客户端提交的事务后会立即将结果返给给客户端，

并不关心从库是查已经接收并处理了事务，

这样就会有一个问题，

主库如果down掉了，

此时主上已经提交的事务可能并没有传到从库服务器上，

如果此时，强行将从提升为主，可能会导致新主上的数据不完整。

默认情况下MySQL5.5/5.6/5.7和mariaDB10.0/10.1的复制功能是异步的

优点：

给客户端的反馈特别快

缺点：

数据可靠性不高

2.全同步复制(Fully synchronous replication)

指当主库执行完一个事务

所有的从库都执行了该事务才返回给客户端

因为需要等待所有从库执行完该事务才能返回

所以全同步复制的性能必然会收到严重的影响

返回客户端的响应速度也会被拖慢

优点：

数据可靠性特别高

缺点：

反馈给客户的时间特别长

3.半同步复制(Semisynchronous replication)

MySQL5.5由Google贡献的补丁才开始支持半同步复制(semi Replication)模式，

半同步复制介于异步复制和全同步复制之间，

主库在执行完客户端提交的事务后不是立刻返回给客户端，

而是等待至少一个从库接收到并写到relay log中才返回给客户端。

相对于异步复制，半同步复制提高了数据的安全性，

同时它也造成了一定程度的延迟，这个延迟最少是一个TCP/IP往返的时间（建立连接的时间）

所以，半同步复制最好在低延时的网络中使用。

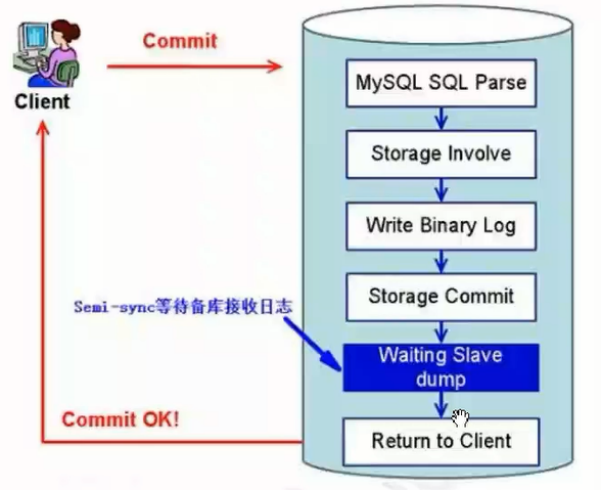
当出现超时情况时，源主服务器会暂时切换到异步复制模式，

直到至少有一台设置为半同步复制模式的从服务器及时收到信息为止

注意：

半同步复制模式在主服务器和从服务器同时启用，否则主服务器默认使用异步复制模式

半同步复制的原理图：



半同步复制的潜在问题

客户端事务在存储引擎层提交后，在得到从库同步的过程中，主库宕机了

此时可能的情况有两种：

➢事务还没发送到从库上

此时，客户端会收到事务提交失败的信息（从还没拿到数据）

客户端会重新提交该事务到新的主上

当宕机的主库重新启动后

以从库的身份重新加入到该主从结构中

发现，该事务在从库中被提交了两次，一次是之前作为主的时候，一次是被新主同步过来的

➢事务已经发送到从库上

此时，从库已经收到并应用了该事务

但是客户端仍然会收到事务提交失败的信息

重新提交该事务到新的主上

4.无数据丢失的半同步复制

针对上述潜在问题，MySQL5.7 引入了一种新的半同步方案：

Loss-Less 半同步复制

针对上面这个图，"Waiting Slave dump”被调整到“Storage Commit"之前 （位置调换）

当然，之前的半同步方案同样支持

MySQL5.7.2引入了一个新的参数进行控制 rpl_semi_sync_master_wait_point

rpl_semi_sync_master_wait_point 有两种取值

➢AFTER_ SYNC：这个即新的半同步方案，Waiting Slave dump在Storage Commit之前

➢AFTER_ COMMIT：老的半同步方案，如图所示

3.MySQL支持的复制方式

基于SQL语句的复制(statement-based replication, SBR):

在主服务器上执行的SQL语句

在从服务器上执行同样的SQL语句

效率比较高

基于行的复制(row-based replication, RBR):

主服务器把表的行变化作为事件写入到二进制日志中，

(只执行对数据有变化的，增删改 insert update delete)

主服务器把代表了行变化的事件复制到从服务中

数据精准度比较高

混合模式复制(mixed-based replication, MBR):

先采用基于语句的复制，一旦发现基于语句无法精确复制时，再采用行

在MySQL5.1.4之前的版本都只能使用基于SQL语句的复制方式

MySQL5.6.5 和往后的版本是基于global transaction identifiers(GTIDs)来进行事务复制

当使用GTIDs时可以大大简化复制过程，

因为GTIDs完全基于事务

只要在主服务器上提交了事务，那么从服务器就一定会执行该事务

通过设置服务器的系统变量binlog.format来指定要使用的格式

1.RBR的优点

➢ 任何情况都可以被复制，

这对复制数据来说是最安全可靠的

更少的行级锁表

➢ 和其他大多数数据库系统的复制技术一样

➢ 多数情况下，从服务器上的表如果有主键的话，复制就会快很多

2.RBR的缺点

➢ Binlog 文件较大

➢ 复杂的回滚时binlog,中会包含大量的数据

主服务器.上执行多个UPDATE语句时

所有发生变化的记录都会写到binlog 中

而且只写为一个操作事物

这会导致频繁发生binlog的并发写问题

➢ 不能通过查看日志来审计执行过的sql语句

不过可以通过使用mysglbinlog-base64-output=decode-rows -verbose来查看数据的变动

3.SBR的优点

➢ 历史悠久，技术成熟binlog文件较小

➢ binlog中包含了所有数据库更改信息，可以据此来审核数据库的安全等情况

➢ binlog可以用于实时的还原，而不仅仅用于复制

➢ 主从版本可以不一 样，从服务器版本可以比主服务器版本高

4.SBR 的缺点

SBR的缺点

➢不是所有的UPDATE,语句都能被复制，尤其是包含不确定操作的时候

➢复制需要进行全表扫描(WHERE语句中没有使用到索引)的UPDATE时，

需要比RBR请求更多的行级锁

➢对于一些复杂的语句，在从服务器上的耗资源情况会更严重

➢如下函数的语句不能被正确地复制: load file(); uuid(), uuid shor(); user); found_rows();
sysdate(); get lock(); is free. lock(); is used lock(); master pos wait(); rand(); release lock();
sleep(); version();

➢在日志中出现如下警告信息的不能正确地复制:

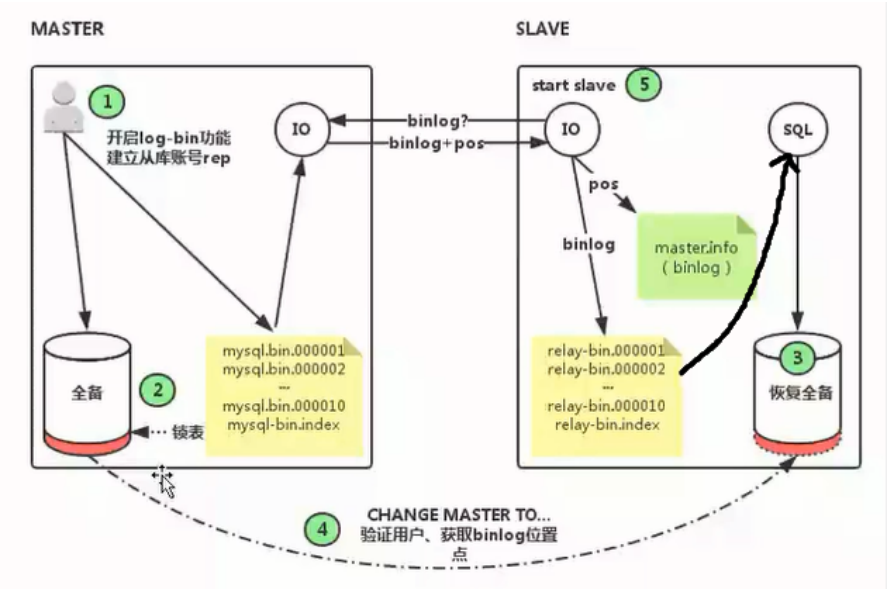
[Warning]Statementisnotsafetologinstatement format.

或者在客户端中出现show warnings

➢Insert...select语句会执行大量的行级锁表

无论选择哪种方式复制，都会影响到复制的效率以及服务器的损耗，以及数据一致性问题
目前其实没有很好的客观手段去评估一个系统更适合哪种方式的复制

4.MySQL主从复制的工作过程



！！！！！

master开启binlog二进制日志功能开启，创建主从同步的账号，当有数据进行写入的操作的时候，它不仅要写入它的磁盘上，还会将数据库变更的语句记录到binary log里面（即bin log里面）

对于从来说，它首先会开启一个IO线程，这个IO线程会和主的IO线程建立连接，建立完连接后，来读取它的日志，看一下主现在用的几号日志的什么位置（第几册笔记的第多少页），从会从这里拿走binlog的号码及binlog的位置，拿完之后，从会将binlog同步到自己的relay log里面，会把pos放在一个masterinfo文件（master详细信息）里；然后从会开启一个SQL线程，SQL线程会从relay log里面读取（relay log里面记录的都是SQL语句），读取完之后，会将SQL语句执行，来达到数据重放的效果，从的数据产生变更

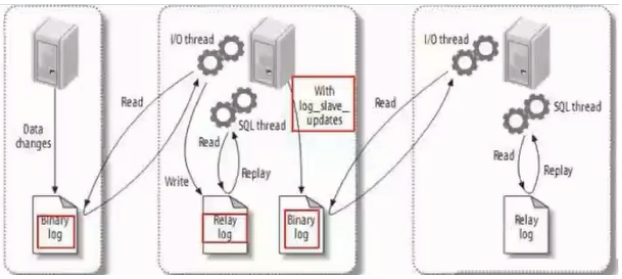
整个配置过程，需要用CHANGE MASTER TO 来指向新主

该过程的第一部分就是master记录二进制日志

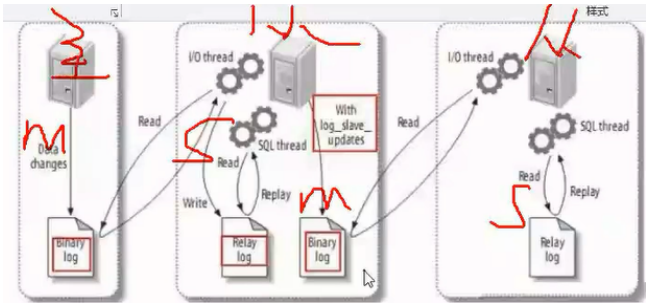
在每个事务更新数据完成之前，master在二日志记录这些改变

即使事务中的语句都是交叉执行的

在事件写入二进制日志完成后，master 通知存储引擎提交事务



主　从　从



1. 在每个事务更新完成数据之前

   Master会在二进制日志（Binary log，binlog）中记录这些改变数据的语句

   当MySQL将事务串行的写入二进制日志完成后

   Master通知存储引擎提交事务并将数据写入磁盘

2. Slave开启一个I/O工作线程

   在Master之间建立一个连接

   然后开始Binlog dump process

   Binlog dump process会从Master的二进制日志中读取操作事件

   如果已经跟Master达到一致状态，它会催眠并等待Master产生新的操作事件

   I/O线程将这些改变数据的事件写入自己的中继日志（relay log）

3. SQL Slave Thread（SQL工作线程）是处理MySQL Replication过程的最后一步

   SQL线程从中继日志读取操作事件

   重放其中的事件从而更新Slave服务器的数据，使slave与Master中的数据保持一致

   只要该线程与I/O线程保持一致

   中继日志通常会在OS的缓存中，所以中继日志的开销很小

MySQL5.6以前的版本复制过程有一个很重要的限制——复制在slave.上是串行化的

也就是说master上的并行更新操作不能在slave上并行操作

MySQL5.6 版本参数slave-parallel-workers=1表示启用多线程功能

会产生主从复制的延时的问题

MySQL5.6开始，增加了一一个新特性，是加入了全局事务ID (GTID)来强化数据库的主

备一致性，故障恢复，以及容错能力。

官方文档：http://dev.mysql.com/doc/refman/5.6/en/replication-gtids.html

5.复制过程的限制

➢MySQL5.6之前的版本复制操作在Slave上执行是串行化的，Master 上的并行更新会导致数据复制延迟

面试回答：我们用的5.6版本的，在从上面开启，做线程复制了

➢所有MYSQL服务器的版本都要高于3.2,还有一个基本的原则就是从服务器的数据库版本可以高于主服务器数据库的版本，但是不可以低于主服务器的数据库版本

关于主从同步的监控问题，Mysql有主从同步的状态信息，

可以通过命令show slave status获取，

除了获知当前是否主从同步正常工作，

另外一个重要指标就是Seconds_Behind Master,

从字面理解，它表示当前MySQL主从数据的同步延迟，单位是秒

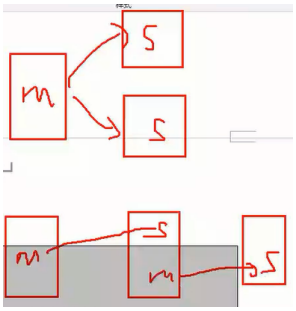
课外知识点：

但这个指标从DBA的角度并不能简单的理解为延迟多少秒，感兴趣的同学可以自己去研究，

但对于应用来说，简单的认为是主从同步的时间差就可以了，另外，当主从同步停止以后，重新启动同步，这个数值可能会是几万秒，取决于主从同步停止的时间长短，我们可以认为数据此时有很多天没有同步了，而这个数值越接近零，则说明主从同步延迟最小，我们可以采集这个指标并汇聚曲线图，来分析我们的数据库的同步延迟曲线，然后根据此曲线，给出一个合理的阀值，主从同步的时延小于阀值时，我们认为从库是同步的，此时可以安全的从从库读取数据。

一主两从 与 主从从 的区别：

主从从 ---连级复制



实验

5台机器，

先用三台

MySQL Master IP: 192.168.200.107

MySQL Slave1 IP: 192.168.200.108

MySQL Slave2 IP: 192.168.200.109

部署MySQL主从异步复制

1.所有机器上的操作:

```
[root@master ~]# systemctl stop firewalld
[root@master ~]# setenforce 0
[root@master ~]# iptables -F
```

2.在MySQL Master.上 的配置NTP时间同步服务器

如果上不了网的时候：

把自己做一台时间同步服务器

[root@master ~]# vim /etc/ntp.conf

末尾加入

server 127.127.1.0

fudge 127.127.1.0 stratum 8

[root@master ~]# systemctl enable ntpd

[root@master ~]# systemctl start ntpd

从服务器，与主服务器进行时间同步

[root@slave1 ~]# ntpdate 192.168.200.107

[root@slave2 ~]# ntpdate 192.168.200.107

3.安装mariadb

所有机器上的操作

[root@localhost ~]# yum -y install mariadb mariadb-devel mariadb-server

4.配置mysql master服务器

1、在/etc/my.cnf 中修改或者增加如下内容:

[root@master ~]# vim /etc/my.cnf

```
server-id=1
# 服务器的标识符，三台机器的server-id一定不能相同，来标识主从或集群中机器的角色的
log-bin=mysql-binlog
# 开启二进制日志功能，mysql-binlog 二进制日志的前缀名，一个文件分为前缀和后缀
log-slave-updates=true
# 可以让从做更新类的操作
```

重启mysal服务器

[root@master ~]# systemctl start mariadb.service

[root@master ~]# mysql

2、创建Replication用户

既然是主从同步，就需要其他两个从拥有同步的权限

MariaDB [(none)]> grant replication slave on *.* to 'myslave'@'192.168.200.%' identified by '123456';

MariaDB [(none)]> flush privileges;

3、获得Master DB的相关信息

MariaDB [(none)]> show master status;

```
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| mysql-binlog.000003 |      477 |              |                  |
+--------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

4、备份Master原有数据0

建议不繁忙的时候做，

备份时会锁表，就是没法往里面写新的东西，但是可以看 （备份还原）

如果在生产环境中Master服务器已经运行一段时间，

或者Master服务器上已经存在数据

为了保证所有数据的一致性，需要先将Master目前已有的数据全部导给Slave服务器

备份的方法有很多，可以直接备份数据文件，也可以使用mysqldump.工具

全新搭建的环境不存在数据备份的问题

[root@mysql-master ~]# mysqldump -u root --all-databases > /root/alldbbackup.sql

[root@mysql-master ~]# scp /root/alldbbackup.sql root@192.168.200.108:/root/

[root@mysql-master ~]# scp /root/alldbbackup.sql root@192.168.200.109:/root/

5、在MySQL Slave上的配置

导入Master的备份脚本

[root@localhost ~]# mysql -u root -p < /root/alldbbackup.sql

从库连接主库进行测试，如果连接成功说明主库配置成功，证明授权没有问题

[root@slave1 ~]# mysql -umyslave -p123456 -h 192.168.200.107

修改MySQL配置文件

[root@slave1 ~]# vim /etc/my.cnf

[mysqld]

server-id=2　　机器标识符

relay-log=relay-log-bin　　启用中继日志的功能

relay-log-index=slave-relay-bin.index　　定义中继日志的索引，帮助查找数据

[root@slave2 ~]# vim /etc/my.cnf

[mysqld]

server-id=3

relay-log=relay-log-bin

relay-log-index=slave-relay-bin.index

修改完后重启数据库

[root@slave1 ~]# systemctl start mariadb.service

[root@slave2 ~]# systemctl start mariadb.service

192.168.200.108 & 192.168.200.109

MariaDB [(none)]> stop slave;　　先将从的角色关掉

192.168.200.108 & 192.168.200.109

MariaDB [(none)]> change master to

-> master_host='192.168.200.107',　　//主服务器的IP地址

-> master_user='myslave',　　//主服务器授权的用户

-> master_password='123456',　　//主服务器授权的密码

-> master_log_file='mysql-binlog.000003',　　//主服务器二进制日志的文件名

这里在主服务器上查找文件，MariaDB [(none)]> show master status;

-> master_log_pos=477;　　//日志文件的开始位置

MariaDB [(none)]> start slave;　　角色开启

MariaDB [(none)]> show slave status\G　　检测是否配置成功

到这实验就成功了~

注意：

关注这几行：

Last_IO_Errno: 0

Last_IO_Error:

Last_SQL_Errno: 0

Last_SQL_Error:

因为有时候，下面这两个可能为no ,具体错误会跟在上面对应的位置

Slave_IO_Running: Yes

Slave_SQL_Running: Yes

如果两个都是no，就是主从同步没有开启，或者没成功

6、测试复制是否成功

在Master服务器上创建一个数据库或者表，

到Slave服务器上查看，如果配置成功就，可以成功同步

所有机器：

MariaDB [(none)]> show databases;

192.168.200.107：

MariaDB [(none)]> create database sofia;

Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> use sofia;

Database changed

MariaDB [sofia]> create table new(name char(20),phone char(20));

在从的上面验证：

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sofia              |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

192.168.200.108

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sofia              |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

192.168.200.109

主从复制成功~

报以下错误的解决方法：

```
ERROR 1201(HY000):Could now initialize master info structure; more error messages can be
found in the MySQL error log

stop slave;
reset slave;
CHANGE MASTER TO
MASTER_HOST='192.168.200.111',
MASTER_USER='myslave',
MASTER_PASSWORD='123456',
MASTER_LOG_FILE='mysql-binlog.000003',
MASTER_LOG_POS=477;
start slave;
```

数据不同步解决办法：

```
MySQL [(none)]> stop slave;
Query OK, 0 rows affected (0.00 sec)

MySQL [(none)]> set global sql_slave_skip_counter=1;
Query OK, 0 rows affected (0.00 sec)

MySQL [(none)]> start slave;
Query OK, 0 rows affected (0.00 sec)
```

二、主从数据库相关命令

- change master; //从服务器上修改参数使用
- slave stop; slave start; //开始停 止从数据库
- show slave status\G //显示从库状态信息
- show master status\G //显示主库状态信息
- purge master logs to 'binlog.00004';

//此命令非常小心，删除主数据库没用的二进制日志文件。

如果误删除，那么从库就没有办法自动更新了

如果当前操作的丛库以前曾经与其他MySQL服务器建立过主从关系，

会发现即使在my.cnf文件中更改了主服务器的位置，

但是MSQL仍然在试图连接就旧的主服务器的现象。

发生这种问题的时候，

可以通过清除masterinfo这个缓存文件或者在mysgl中通过命令来进行设置。

方式如下:

1.删除 master.info方法

这个文件通常位于数据文件存放目录里,可以直接将其删除，然后重新启动服务器。

[root@master ~]# ls /var/lib/mysql/

这个目录下面有master.info

2.mysql命令方法

如果不方便重新服务器时，就只能使用mysqL命令来操作

MariaDB [(none)]> reset slave; 重置一下就行

三、部署MySQL主从半同步复制

半同步复制支持多种插件:

插件位置： /us/ib64/mysl/plugin/*

```
[root@master ~]# rpm -ql mariadb-server | grep semisync
/usr/lib64/mysql/plugin/semisync_master.so
/usr/lib64/mysql/plugin/semisync_slave.so
```

半同步复制插件:

```
semisync_master.so
semisync_slave.so
```

查看插件是否安装:

```
MariaDB [(none)]> show plugins;
```

在主节点安装并启用插件:

```
MariaDB [(none)]> install plugin rpl_semi_sync_master soname 'semisync_master.so';
MariaDB [(none)]> show plugins;
| rpl_semi_sync_master           | ACTIVE   | REPLICATION        | semisync_master.so | GPL  |
```

查看插件允许状态

```
MariaDB [(none)]> show global variables like '%semi%';
| rpl_semi_sync_master_enabled       | OFF   |
```

启动插件:

```
MariaDB [(none)]> set @@global.rpl_semi_sync_master_enabled=on;
MariaDB [(none)]> show global variables like '%semi%';
| rpl_semi_sync_master_enabled       | ON    |
```

配置从192.168.200.108

```
MariaDB [(none)]> install plugin rpl_semi_sync_slave soname 'semisync_slave.so';
MariaDB [(none)]> show plugins;
MariaDB [(none)]> show global variables like '%semi%';
MariaDB [(none)]> set @@global.rpl_semi_sync_slave_enabled=on;
MariaDB [(none)]> show global variables like '%semi%';
```

从节点启动IO线程:

```
MariaDB [(none)]> stop slave io_thread;
Query OK, 0 rows affected (0.00 sec)
MariaDB [(none)]> start slave io_thread;
Query OK, 0 rows affected (0.00 sec)
```

注意：

半同步是在异步的基础上做的，得先把异步做好；

在主的上面查看：

```
MariaDB [(none)]> show global status like '%semi%';
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |
```

在主节点创建数据库mydb:

192.168.200.107：

```
MariaDB [(none)]> create database mydb character set 'utf8';
```

从：

192.168.200.108：

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb               |
| mysql              |
| performance_schema |
| sofia              |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

在主节点创建数据表:

```
MariaDB [(none)]> use mydb;
MariaDB [mydb]> create table tbl1 (id int,name varchar(100));
MariaDB [mydb]> show global status like '%semi%';
```

数据表产生变化：

```
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |
| Rpl_semi_sync_master_net_avg_wait_time     | 958   |
| Rpl_semi_sync_master_net_wait_time         | 1917  |
| Rpl_semi_sync_master_net_waits             | 2     |
| Rpl_semi_sync_master_no_times              | 0     |
| Rpl_semi_sync_master_no_tx                 | 0     |
| Rpl_semi_sync_master_status                | ON    |
| Rpl_semi_sync_master_timefunc_failures     | 0     |
| Rpl_semi_sync_master_tx_avg_wait_time      | 1252  |
| Rpl_semi_sync_master_tx_wait_time          | 2505  |
| Rpl_semi_sync_master_tx_waits              | 2     |
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
| Rpl_semi_sync_master_wait_sessions         | 0     |
| Rpl_semi_sync_master_yes_tx                | 2     |
+--------------------------------------------+-------+
MariaDB [mydb]> insert into tbl1 values (1,'tom');
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |
| Rpl_semi_sync_master_net_avg_wait_time     | 783   |
| Rpl_semi_sync_master_net_wait_time         | 2351  |
| Rpl_semi_sync_master_net_waits             | 3     |
| Rpl_semi_sync_master_no_times              | 0     |
| Rpl_semi_sync_master_no_tx                 | 0     |
| Rpl_semi_sync_master_status                | ON    |
| Rpl_semi_sync_master_timefunc_failures     | 0     |
| Rpl_semi_sync_master_tx_avg_wait_time      | 955   |
| Rpl_semi_sync_master_tx_wait_time          | 2867  |
```

```
| Rpl_semi_sync_master_tx_waits              | 3     |
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
| Rpl_semi_sync_master_wait_sessions         | 0     |
| Rpl_semi_sync_master_yes_tx                | 3     |
+--------------------------------------------+-------+
```

思考问题:

1、一主多从和主从从的区别?

一主多从：多个从从一个主上面来同步数据；

主从从：连级

2、如果部署了主从还需要做数据备份吗?

需要；主 delete --> 从 delete

3、备份可以在从服务器上完成

缓解主的压力

4、公司数据库架构可以是主从同步吗?

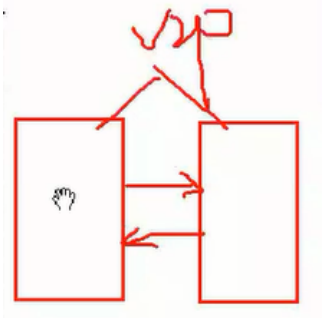
主从同步只是一个数据同步方案，并不是一个特别完美的架构

故障的自动切换；

主从复制只是一个单向的复制；只是从主同步过去；之后的数据会写到从的上面；

以后master上面是没有新的数据的，所以需要反过来做一次同步；

所以就变成了主从双向复制



5、主从复制有延迟的缓解方法?优缺点

用MySQL5.6，开启多线程，

缺点：数据的一致性差

优点：当主的上面执行误操作的时候，不会立马同步到从的上面，不会那么及时；有时间去中断它们的主从复制

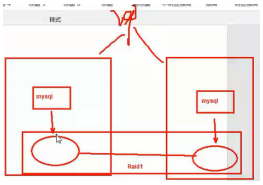
四、MySQL主主高可用方案

- **MySQL 主主+Keepalived （官方的方案）**

  两台机器互为主从

- **MySQL+DRBD+Heartbeat （各种社区的）**

  数据块级的同步，底层磁盘做的同步



在企业中，数据库高可用一直是企业的重中之重，

中小企业很多都是使用mysql主主方案，一主多从，读写分离等，

但是单主存在单点故障，从库切换成主库需要作改动。

因此，如果是双主或者多主，就会增加mysqL入口，增加高可用。

不过多主需要考虑自增长ID问题，这个需要特别设置配置文件，

比如双主，可以使用奇偶，

总之，主之间设置自增长ID相互不冲突就能完美解决自增长ID冲突问题。

主主方案实现思路

1、两台mysqL 都可读写，互为主备。默认只使用一台masterA 负责数据的写入，

另一台masterB备用处于备用状态;

2、masterA 是masterB的主库，masterB 又是masterA的主库，它们互为主从;

3、两台主库之间做高可用,可以采用keepalived等方案，使用VIP对外提供服务;

4、所有提供服务的从服务器与masterB进行主从同步(双主多从) --根据企业的实际情况考虑;

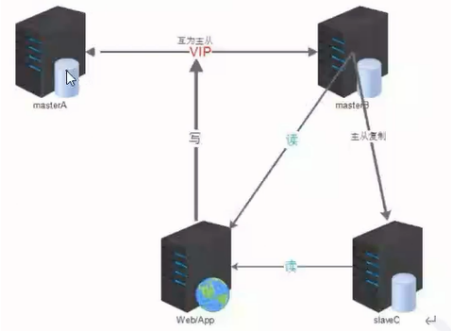
5、建议采用高可用策略的时候，

masterA 或masterB均不因宕机恢复后而抢占VIP (非抢占模式);

这样做可以在一定程度上保证主库的高可用,在一台主库down掉之后,

可以在极短的时间内切换到另一台主库上，尽可能减少主库宣机对业务造成的影响，

减少了主从同步给生产主库带来的压力



但是也有几个不足的地方:

➢ masterB 可能会一直处于空闲状态(可以用它当从库，负责部分查询);

➢ 主库后面提供服务的从库要等masterB先同步完了数据后才能去masterB上去同步数据，这样可能会造成一一定程度的同步延时;

实验:

1.双主-互为主从

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.00 sec)
```

修改配置文件 192.168.200.107：

```
[root@master ~]# vim /etc/my.cnf
max_binlog_size=1024M                # binlog单文件最大值
auto_increment_offset=1               #自增长ID，起始值为1
auto_increment_increment=2            #奇数ID
replicate-ignore-db=information_schema          # 忽略不同步主从的数据库
replicate-ignore-db=performance_schema
replicate-ignore-db=test
replicate-ignore-db=mysql

max_connections=3000         #最大连接数
max_connect_errors=30          # 连接错误超时
```

```
skip-character-set-client-handshake      # 跳过客户机的字符集的检查，防止乱码
init-connect='set names utf8'            #设置连接执行SQL语句的字符集是utf8
character-set-server=utf8                # 设置服务端的字符集是utf8
wait_timeout=1800                  # 请求的最大连接时间
interactive_timeout=1800
sql_mode=no_engine_substitution,strict_trans_tables           #使用SQL的模式来进行复制的

relay-log=relay-log-bin      #它也要变成第二台机器的从，开启relay-log的功能
relay-log-index=slave-relay-bin.index
```

重启：

```
[root@master ~]# systemctl restart mariadb.service
```

192.168.200.107重新授权给机器：192.138.200.108

为了保证用户名 密码一致

```
MariaDB [(none)]> grant replication slave on *.* to 'repl'@'192.168.200.108' identified by '123456';
MariaDB [(none)]> flush privileges;
MariaDB [(none)]> show master status;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| mysql-binlog.000004       |      486 |              |                  |
+--------------------+----------+--------------+------------------+
```

修改192.168.200.108的配置文件：

```
[root@slave1 ~]# vim /etc/my.cnf
```

与主的不同的主要有两处：

```
server-id=2
auto_increment_offset=2           # 起始值为2.偶数ID
```

重启服务：

```
[root@slave1 ~]# systemctl restart mariadb.service
[root@slave1 ~]# mysql
```

授权：

```
MariaDB [(none)]> grant replication slave on *.* to 'repl'@'192.168.200.107'identified by '123456';
MariaDB [(none)]> flush privileges;
```

查看：

```
MariaDB [(none)]> show master status;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| mysql-binlog.000001 |      485 |              |                  |
+--------------------+----------+--------------+------------------+
1 row in set (0.09 sec)
```

设置192.168.200.107：

```
MariaDB [(none)]> stop slave;
MariaDB [(none)]> change master to
    -> master_host='192.168.200.108',
    -> master_user='repl',
    -> master_password='123456',
```

```
    -> master_log_file='mysql-binlog.000001',
    -> master_log_pos=485;
MariaDB [(none)]> start slave;
MariaDB [(none)]> show slave status\G
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Ignore_DB: information_schema,performance_schema,test,mysql
```

设置192.168.200.108：

```
MariaDB [(none)]> stop slave;
MariaDB [(none)]> change master to
master_host='192.168.200.107',master_port=3306,master_user='repl',master_password='123456',master_log_file='mysql-binlog.000004',master_log_pos=486;
MariaDB [(none)]> start slave;
MariaDB [(none)]> show slave status\G
```

到这里的时候，双主已经做完啦~　互为主从已经做完，但还未做高可用嘞。。。

测试：

192.168.200.107

```
MariaDB [(none)]> create database test01;
```

192.168.200.108

```
MariaDB [(none)]> show databases;

MariaDB [(none)]> create database test02;
```

192.168.200.107

```
MariaDB [(none)]> show databases;

[root@master ~]# ls /var/lib/mysql/
```

查看日志：

```
[root@master ~]# mysqlbinlog --no-defaults -v /var/lib/mysql/mysql-binlog.000004
```

2.MySQL高可用方案

配置：192.168.200.107

```
[root@master ~]# yum -y install keepalived
[root@master ~]# vim /etc/keepalived/keepalived.conf
! Configuration File for keepalived
global_defs {
   router_id LVS_MASTER-A
}
vrrp_script mysql {
    script "/opt/mysql.sh"
    interval 2
    weight -5
    fall 2
    rise 1
}
vrrp_instance VI_1 {
    state BACKUP
    interface ens32
    virtual_router_id 53
```

```
    priority 100
    nopreempt
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.200.254
    }
}
[root@master ~]# vim /opt/mysql.sh
#!/bin/bash
counter=$(netstat -na|grep "LISTEN"|grep "3306"|wc -l)
if [ "${counter}" -eq 0 ]; then
    systemctl stop keepalived
fi
[root@master ~]# chmod +x /opt/mysql.sh
[root@master ~]# systemctl start keepalived.service
[root@master ~]# ip a
inet 192.168.200.254/32 scope global ens32
```

配置192.168.200.108：

```
[root@slave1 ~]# vim /etc/keepalived/keepalived.conf
   router_id LVS_MASTER-B
    priority 99
[root@slave1 ~]# vim /opt/mysql.sh
[root@slave1 ~]# systemctl restart keepalived.service
[root@slave1 ~]# tail -30 /var/log/messages
```

在重启服务时遇到的问题

https://www.cnblogs.com/taosim/articles/2639520.html

测试VIP转移

192.168.200.107

模拟故障

```
[root@master ~]# systemctl stop mariadb.service
[root@master ~]# systemctl status keepalived.service
● keepalived.service - LVS and VRRP High Availability Monitor
   Loaded: loaded (/usr/lib/systemd/system/keepalived.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
```

再重新开启服务：

```
[root@master ~]# systemctl start mariadb.service
[root@master ~]# systemctl start keepalived.service
```

查看192.168.200.108：

```
[root@slave1 ~]# ip a
```

因为是非抢占模式，所以VIP依然在

在远程客户端测试：

两台机都授权

[root@master ~]# mysql

MariaDB [(none)]> grant all on *.* to 'root'@'192.168.200.%' identified by '123456';

MariaDB [(none)]> flush privileges;

在这里。出了小问题。。。

当拿192.168.200.107，登录数据库时，登不上

[root@master ~]# mysql -uroot -p123456 -h192.168.200.254

ERROR 1045 (28000): Access denied for user 'root'@'master' (using password: YES)

重增加授权

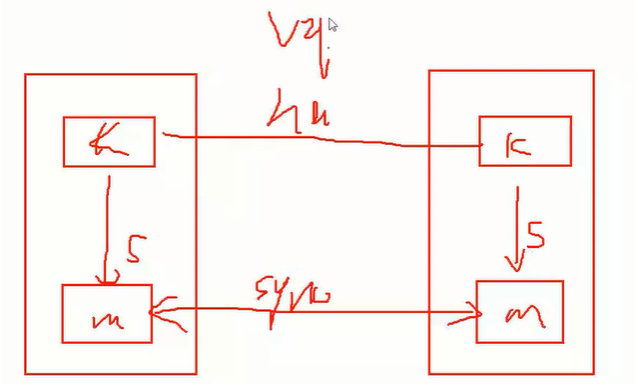
MariaDB [(none)]> grant all on *.* to 'root'@'master' identified by '123456';

再次登录：[root@master ~]# mysql -uroot -p123456 -h192.168.200.254

成功登录上了。。。

参考：https://blog.csdn.net/jiaping0424/article/details/51253202

--->这里长时间登不上也可以考虑网络环境中是不是有VIP冲突的原因



环境准备：

[root@master ~]# systemctl stop keepalived.service

[root@master ~]# vim /etc/my.cnf　　# 还原内容

来一波误操作~

[root@master ~]# rm -rf /var/lib/

```
[root@master ~]# ls /var/lib/
nfs
[root@master ~]#
```

```
[root@slave1 ~]# systemctl stop keepalived.service
[root@slave1 ~]# ls /var/lib/
AccountsService  color     flatpak   hyperv      machines        ntp          rpcbind         sss         upower
alsa             colord    fprint    initramfs   misc            os-prober    rpm             stateless   vmware
alternatives     cs        games     ipa-client  mlocate         PackageKit   rpm-state       systemd     xkb
authconfig       dav       gdm       iscsi       mysql           plymouth     rsyslog         tpm         yum
bluetooth        dbus      geoclue   libvirt     net-snmp        polkit-1     samba           tuned
certmonger       dhclient  gssproxy  lldpad      NetworkManager  postfix      selinux         udisks2
chrony           dnsmasq   haproxy   logrotate   nfs             pulse        setroubleshoot  unbound
[root@slave1 ~]#
```

在192.168.200.108：

[root@slave1 ~]# rm -rf /var/lib/mysql/*

删除磁盘上已经存储的数据

[root@slave1 ~]# vim /etc/my.cnf　　# 还原修改前的

这时跟全新安装的一样的

重启服务：

[root@slave1 ~]# systemctl restart mariadb.service

192.168.200.109：

[root@slave2 ~]# vim /etc/my.cnf

[root@slave2 ~]# rm -rf /var/lib/mysql/*

到192.168.200.107：

[root@master ~]# ls /var/lib/mysql/

aria_log.00000001 aria_log_control ibdata1 ib_logfile0 ib_logfile1 mysql performance_schema test

[root@master ~]# rm -rf /var/lib/mysql/*

[root@master ~]# systemctl restart mariadb

Job for mariadb.service failed because the control process exited with error code. See "systemctl status mariadb.service" and "journalctl -xe" for details.

[root@master ~]# systemctl status mariadb -l

```
[root@master ~]# ls -ld /var/lib/mysql/
drwxr-xr-x 2 root root 6 4月  24 08:48 /var/lib/mysql/
[root@slave1 ~]# ls -ld /var/lib/mysql/
drwxr-xr-x 5 mysql mysql 177 4月  24 08:31 /var/lib/mysql/
```

拿root传过去的，权限不同

修改权限

[root@master ~]# chown mysql.mysql /var/lib/mysql/

重启服务

[root@master ~]# systemctl restart mariadb

[root@master ~]# ls /var/lib/mysql/

[root@master ~]# mysql

总结:

数据目录/var/lib/mysql/

源码或二进制安装数据目录=/usr/local/mysql/data/

先删掉原来的/var/lib/mysql/

重启服务之后，会生成新的需要的文件

之前的授权配置就清空了，相当于全新安装的，省去了重新安装

使用scp传数据，存在权限的问题，在后续可能还有问题

```
[root@master ~]# ls -l /var/lib/
总用量 4
drwxr-xr-x   4 root  root    32 4月  24 08:35 AccountsService
drwxr-xr-x   2 root  root     6 4月  24 08:35 alsa
drwxr-xr-x   2 root  root  4096 4月  24 08:35 alternatives
drwx------   3 root  root    18 4月  24 08:35 authconfig
drwxr-xr-x   2 root  root     6 4月  24 08:35 bluetooth
drwxr-xr-x   5 root  root    46 4月  24 08:35 certmonger
drwxr-xr-x   2 root  root     6 4月  24 08:35 chrony
drwxr-xr-x   3 root  root    17 4月  24 08:35 color
drwxr-xr-x   4 root  root    67 4月  24 08:35 colord
drwxr-xr-x   2 root  root     6 4月  24 08:35 cs
drwx------   2 root  root     6 4月  24 08:35 dav
drwxr-xr-x   2 root  root    24 4月  24 08:35 dbus
drwxr-xr-x   2 root  root     6 4月  24 08:35 dhclient
drwxr-xr-x   2 root  root     6 4月  24 08:35 dnsmasq
drwxr-xr-x   3 root  root    34 4月  24 08:35 flatpak
drwxr-xr-x   2 root  root     6 4月  24 08:35 fprint
drwxr-xr-x   2 root  root     6 4月  24 08:35 games
drwxrwx---   7 root  root    97 4月  24 08:35 gdm
drwxr-xr-x   2 root  root     6 4月  24 08:35 geoclue
drwxr-xr-x   4 root  root    35 4月  24 08:35 gssproxy
drwxr-xr-x   2 root  root     6 4月  24 08:35 haproxy
drwxr-xr-x   2 root  root     6 4月  24 08:35 hyperv
drwxr-xr-x   2 root  root     6 4月  24 08:35 initramfs
drwxr-xr-x   4 root  root    35 4月  24 08:35 ipa-client
drwxr-xr-x   8 root  root    90 4月  24 08:35 iscsi
drwxr-xr-x   8 root  root    93 4月  24 08:35 libvirt
drwxr-xr-x   2 root  root     6 4月  24 08:35 lldpad
drwxr-xr-x   2 root  root    30 4月  24 08:35 logrotate
drwx------   2 root  root     6 4月  24 08:35 machines
drwxr-xr-x   2 root  root    37 4月  24 08:35 misc
drwxr-x---   2 root  root    24 4月  24 08:35 mlocate
drwxr-xr-x   5 mysql mysql  284 4月  24 08:58 mysql
drwxr-xr-x   4 root  root    45 4月  24 08:35 net-snmp
drwx------   2 root  root   169 4月  24 08:35 NetworkManager
drwxr-xr-x.  5 root  root   105 4月  24 08:35 nfs
drwxr-xr-x   2 root  root     6 4月  24 08:35 ntp
drwxr-xr-x   2 root  root     6 4月  24 08:35 os-prober
drwxr-xr-x   2 root  root    29 4月  24 08:35 PackageKit
drwxr-xr-x   2 root  root    27 4月  24 08:35 plymouth
```

```
[root@slave1 ~]# ls -l /var/lib/
总用量 8
drwxr-xr-x.  4 root          root            32 1月   1 01:50 AccountsService
drwxr-xr-x.  2 root          root             6 8月   3 2017 alsa
drwxr-xr-x.  2 root          root          4096 4月  23 00:16 alternatives
drwx------.  3 root          root            18 1月   1 02:00 authconfig
drwxr-xr-x.  2 root          root             6 9月  12 2017 bluetooth
drwxr-xr-x.  5 root          root            46 1月   1 01:51 certmonger
drwxr-xr-x.  2 chrony        chrony           6 4月  13 2018 chrony
drwxr-xr-x.  3 root          root            17 1月   1 01:48 color
drwxr-xr-x.  4 colord        colord          67 1月   1 02:21 colord
drwxr-xr-x.  2 root          root             6 8月   4 2017 cs
drwx------.  2 apache        apache           6 4月  21 2018 dav
drwxr-xr-x.  2 root          root            24 1月   1 02:19 dbus
drwxr-xr-x.  2 root          root             6 4月  13 2018 dhclient
drwxr-xr-x.  2 root          root             6 4月  11 2018 dnsmasq
drwxr-xr-x.  3 root          root            34 1月   1 01:51 flatpak
drwxr-xr-x.  2 root          root             6 6月  24 2014 fprint
drwxr-xr-x.  2 root          root             6 4月  11 2018 games
drwxrwx--T.  7 gdm           gdm             97 4月  15 23:55 gdm
drwxr-xr-x.  2 geoclue       geoclue          6 8月   2 2017 geoclue
drwxr-xr-x.  4 root          root            55 4月  22 18:09 gssproxy
drwxr-xr-x.  2 haproxy       haproxy          6 4月  11 2018 haproxy
drwxr-xr-x.  2 root          root             6 4月  11 2018 hyperv
drwxr-xr-x.  2 root          root             6 4月  11 2018 initramfs
drwxr-xr-x.  4 root          root            35 1月   1 01:46 ipa-client
drwxr-xr-x.  8 root          root            90 1月   1 01:50 iscsi
drwxr-xr-x.  8 root          root            93 1月   1 01:51 libvirt
drwxr-xr-x.  2 root          root             6 11月  6 2016 lldpad
drwxr-xr-x.  2 root          root            30 4月  24 03:20 logrotate
drwx------.  2 root          root             6 1月   1 01:49 machines
drwxr-xr-x.  2 root          root            37 1月   1 02:19 misc
drwxr-x---.  2 root          slocate         24 4月  24 03:20 mlocate
drwxr-xr-x   5 mysql         mysql          177 4月  24 08:31 mysql
drwxr-xr-x.  4 root          root            45 1月   1 01:48 net-snmp
drwx------.  2 root          root           169 4月  23 08:13 NetworkManager
drwxr-xr-x.  5 root          root           105 1月   1 01:51 nfs
drwxr-xr-x.  2 ntp           ntp              6 4月  13 2018 ntp
drwxr-xr-x.  2 root          root             6 11月  6 2016 os-prober
drwxr-xr-x.  2 root          root            29 1月   1 02:21 PackageKit
```

后期需要根据需要对其他文件或目录的权限进行修改

调用不同的东西的时候需要用到不同的用户身份

比如：上面重启服务失败报错

Cannot change ownership of the database directories to the 'mysql'

权限没给到

小知识点：

[root@slave1 ~]# scp --help

[root@slave1 ~]# man scp

-p   Preserves modification times, access times, and modes from the original file.

把权限复制过去就可以避免这个问题了

在工作中，只有主从，主库MySQL的数据被误删除了 ，只要在从库上做一个完整备份，然后将备份扔给主库，主一还原就可以啦

192.168.200.107

[root@master ~]# mysql

MariaDB [(none)]> grant replication slave on *.* to 'repl'@'192.168.200.%' identified by '123456';

```
MariaDB [(none)]> flush privileges;
MariaDB [(none)]> show master status;
| mysql-binlog.000003 |      474 |              |                  |
```

192.168.200.108

```
[root@slave1 ~]# mysql
MariaDB [(none)]> change master to
master_host='192.168.200.107',master_port=3306,master_user='repl',master_password='123456',master_log_file='mysql-binlog.000003',master_log_pos=474;
MariaDB [(none)]> start slave;
MariaDB [(none)]> show slave status\G
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

TIPS:
有时候MySQL登不上，可以重启一下，试试，就可以登上了

192.168.200.109

```
[root@slave2 ~]# vim /etc/my.cnf
[root@slave2 ~]# rm -rf /var/lib/mysql/*
[root@slave2 ~]# mysql
MariaDB [(none)]> change master to
master_host='192.168.200.107',master_port=3306,master_user='repl',master_password='123456',ma
ster_log_file='mysql-binlog.000003',master_log_pos=474;
MariaDB [(none)]> start slave;
MariaDB [(none)]> show salve status;
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

测试：
192.168.200.107

```
MariaDB [(none)]> create database sofia;
MariaDB [(none)]> show databases;
```

192.168.200.108

```
MariaDB [(none)]> show databases;
```

192.168.200.109

```
MariaDB [(none)]> show databases;
```

到这就做好了，主从复制（一主两从）

五、基于Amoeba读写分离

中文名---变形虫

1.读写分离

在实际的生成环境中，如果对数据库的读和写都在同一个数据库服务器中操作
无论是在安全性，高可用还是高并发等各个方面都不能完全满足实际需求的
因此一般来说都是通过主从复制(Master-Slave)的方式来同步数据
再通过读写分离来提升数据的高并发负载能力这样的方案来进行部署

简单来说，读写分离就是只在主服务器上写，只在从服务器上读，

基本的原理是让主数据库处理事务性查询

而从数据库处理select查询，

数据库复制被用来把事务性查询导致的改变更新同步到集群中的从数据库

2.MySQL读写分离方案

1.基于程序代码内部实现

在代码中根据selectinsert 进行路由分类，这类方法也是目前大型生产环境应用最广泛的

优点：性能好，因为在程序代码中实现，不需要曾加额外的设备作为硬件开支

缺点：需要开发人员来实现，运维人员无从下手

当开发实现不了的时候，运维来做：

找一台机器或服务来进行分离从而实现

2.基于中间代理层实现

代理一般位于客户端和服务器之间，

代理服务器接到客户端请求后通过判断后转发到后端数据库，代表性程序:

(1) mysql-proxy

为mysql开发早期开源项目

通过其自带的lua脚本进行SQL判断

虽然是mysql的官方产品，但是mysql官方不建议将其应用到生产环境

(2) Amoeba (变形虫)

由陈思儒开发，曾就职与阿里巴巴，该程序由java语言进行开发

阿里巴巴将其应用于生成环境，它不支持事物和存储过程

通过程序代码实现mysqL读写分离自然是一个不错的选择

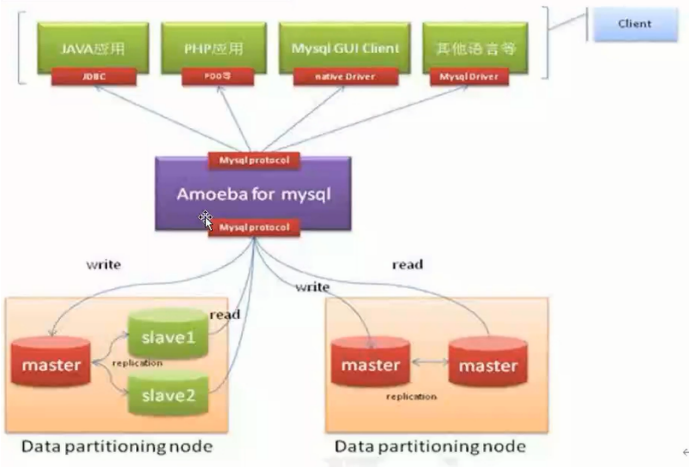
但是并不是所有的应用都适合在程序代码中实现读写分离

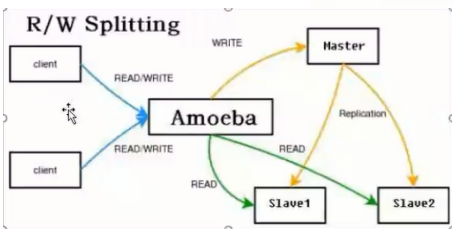
像一些大型复杂的jaya 应用

如果在程序代码中实现读写分离对代码改动就较大

像这种应用一般考虑使用代理层来实现

实验:





MySQL Master IP: 192.168.200.107

MySQL Slave1 IP: 192.168.200.108

MySQL Slave2 IP: 192.168.200.109

MySQL Amoeba IP: 192.168.200.110

MySQL Client IP: 192.168.200.105

Amoeba（变形虫）项目开源框架于2008年发布一款Amoeba for mysql软件

这个软件致力于mysql的分布式数据库前端代理层

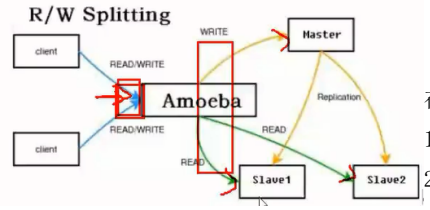
主要为应用层访问mysql的时候充当SQL路由功能

并具有**负载均衡**，高可用性，SQL过滤，**读写分离**

可路由到相关的目标数据库，可并发请求多台数据库

通过Amoeba能够完成多数据源的高可用，负载均衡，数据切片的功能

目前Amoeba已经在很多企业的生产线上使用



在实验中注意，涉及两套用户名和密码

1.用户连接Amoeba

2.Amoeba连接数据库

环境建立在一主两从的环境上继续的

1、在主机Amoeba上安装java环境

因为Amoeba是基于jdk1.5版本开发的，

所以官方推荐使用1.5或者1.6版本，高版本不建议使用

```
[root@Amoeba ~]# java -version
openjdk version "1.8.0_161"
OpenJDK Runtime Environment (build 1.8.0_161-b14)
OpenJDK 64-Bit Server VM (build 25.161-b14, mixed mode)
[root@Amoeba ~]# which java
/usr/bin/java
[root@Amoeba ~]# rm -rf /usr/bin/java
[root@Amoeba ~]# chmod +x jdk-6u31-linux-x64.bin
[root@Amoeba ~]# ./jdk-6u31-linux-x64.bin
Press Enter to continue.....          # 敲回车确认
Done.
[root@Amoeba ~]# mv jdk1.6.0_31/ /usr/local/java            # Java是随便起的名
[root@Amoeba ~]# vim /etc/profile
末尾插入
export JAVA_HOME=/usr/local/java
export CLASSPATH=$CLASSPATH:$JAVA_HOME/bin:$JAVA_HOME/jre/lib
export PATH=$PATH:$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$HOME/bin
export AMOEBA_HOME=/usr/local/amoeba
export PATH=$PATH:$AMOEBA_HOME/bin
[root@Amoeba ~]# source /etc/profile
[root@Amoeba ~]# java -version
java version "1.6.0_31"
Java(TM) SE Runtime Environment (build 1.6.0_31-b04)
Java HotSpot(TM) 64-Bit Server VM (build 20.6-b01, mixed mode)
```

2、安装并配置Amoeba

```
[root@Amoeba ~]# mkdir /usr/local/amoeba
[root@Amoeba ~]# tar xf amoeba-mysql-binary-2.2.0.tar.gz -C /usr/local/amoeba/
[root@Amoeba ~]# chmod -R 755 /usr/local/amoeba
```

3、配置Amoeba读写分离，两个Slave读负载均衡

在Master、Slave1、Slave2服务器中配置Amoeba的访问授权

```
MariaDB [(none)]> grant all on *.* to 'test'@'192.168.200.%' identified by '123.com';
MariaDB [(none)]> flush privileges;
```

修改amoeba的主配置文件amoeba.xml

```
[root@Amoeba ~]#  cd /usr/local/amoeba/conf/
[root@Amoeba conf]# cp amoeba.xml amoeba.xml-$(date +%F)        # 备份加时间戳
[root@Amoeba conf]# ls
amoeba.xml        amoeba.xml-2020-04-24
30                                      <property name="user">amoeba</property>
        # 给前端客户用的名字
32                                      <property name="password">123456</property>
        # 客户端连接使用
115                 <property name="defaultPool">master</property>
        # 默认的池，默认找master
```

下面这两行去注释

```
118                 <property name="writePool">master</property>       #写入的池为master
119                 <property name="readPool">slaves</property>        #读的池为slaves
```

slaves是个组，组内可以有很多个成员

修改dbServers.xml文件

```
[root@Amoeba conf]# cp dbServers.xml dbServers.xml-$(date +%F)            #备份
[root@Amoeba conf]# vim dbServers.xml
 26                         <property name="user">test</property>        # 用户名
```

注意对密码这两行的处理

```
 28                         <!--  mysql password-->
 29                         <property name="password">123.com</property>
```

声明服务器：

```
 45         <dbServer name="master"  parent="abstractServer">
 47                         <property name="ipAddress">192.168.200.107</property>
 51         <dbServer name="slave1"  parent="abstractServer">
 53                         <property name="ipAddress">192.168.200.108</property>
 57         <dbServer name="slave2"  parent="abstractServer">
 58                 <factoryConfig>
 59                         <property name="ipAddress">192.168.200.109</property>
 60                 </factoryConfig>
```

声明组：

```
 63         <dbServer name="slaves" virtual="true">
 69                         <property name="poolNames">slave1,slave2</property>
```

注意：

```
<property name="loadbalance">1</property>
```

这里面的“1”指的是1=ROUNDROBIN，轮询

```
[root@Amoeba conf]# nohup /usr/local/amoeba/bin/amoeba start &
# nohup不依赖任何终端       & 放到后台启动
[root@Amoeba conf]# netstat -lnpt
tcp6       0      0 :::8066                 :::*                    LISTEN      32307/java
[root@Amoeba conf]# netstat -anpt | grep :3306            #查看是否与MySQL服务器连接
```

```
tcp6       0      0 192.168.200.110:47542   192.168.200.108:3306    ESTABLISHED 32307/java
tcp6       0      0 192.168.200.110:47966   192.168.200.109:3306    ESTABLISHED 32307/java
tcp6       0      0 192.168.200.110:42948   192.168.200.107:3306    ESTABLISHED 32307/java
```

到这读写分离的实验就查不到啦~

4.客户端测试 192.168.200.105

```
[root@localhost ~]# cd /etc/yum.repos.d/
[root@localhost yum.repos.d]# ls
```

只剩本地源

```
[root@localhost ~]# yum -y install mariadb mariadb-devel
[root@localhost ~]# mysql -uamoeba -p123456 -h 192.168.200.110 -P8066
```

测试

在Mysql主服务器上创建一个表， 会自动同步到各个从服务器上，

然后关掉各个从服务器上的Slave功能，在分别插入语句测试

192.168.200.107

```
MariaDB [(none)]> create database db_test;
MariaDB [(none)]> use db_test;
MariaDB [db_test]> create table student (id int(10),name varchar(10),address varchar(20));
```

192.168.200.108 & 192.168.200.109

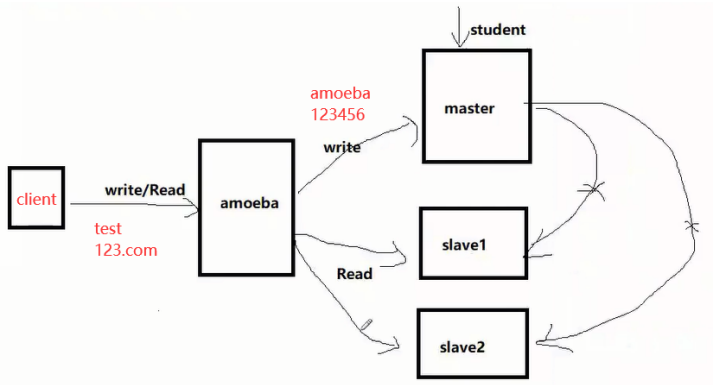
```
MariaDB [(none)]> show databases;
```

192.168.200.109

```
MariaDB [(none)]> use db_test;
MariaDB [db_test]> show tables;
```

停止slave功能

```
MariaDB [db_test]> stop slave;
```



192.168.200.107--master

```
MariaDB [db_test]> insert into student values('1','sofia','this_is_master');
```

192.168.200.108--slave1

```
MariaDB [db_test]> insert into student values('2','sofia','this_is_slave1');
```

192.168.200.109--slave2

```
MariaDB [db_test]> insert into student values('3','sofia','this_is_slave2');
```

192.168.200.105--client上查询

```
MySQL [(none)]> use db_test;
MySQL [db_test]> show tables;
MySQL [db_test]> select * from student;
```

```
MySQL [db_test]> select * from student;
+------+-------+---------------+
| id   | name  | address       |
+------+-------+---------------+
|    2 | sofia | this_is_slave1 |
+------+-------+---------------+
1 row in set (0.07 sec)

MySQL [db_test]> select * from student;
+------+-------+---------------+
| id   | name  | address       |
+------+-------+---------------+
|    3 | sofia | this_is_slave2 |
+------+-------+---------------+
1 row in set (0.01 sec)

MySQL [db_test]> select * from student;
+------+-------+---------------+
| id   | name  | address       |
+------+-------+---------------+
|    2 | sofia | this_is_slave1 |
+------+-------+---------------+
1 row in set (0.05 sec)

MySQL [db_test]> select * from student;
+------+-------+---------------+
| id   | name  | address       |
+------+-------+---------------+
|    3 | sofia | this_is_slave2 |
+------+-------+---------------+
1 row in set (0.05 sec)
```

轮询着出现

在客户端写入语句：

MySQL [db_test]> insert into student values('4','sofia','this_is_write_test');

当两台slave都修复时：

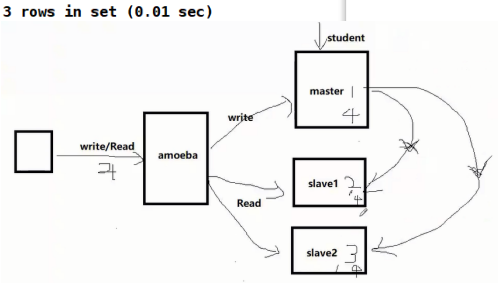
MariaDB [db_test]> start slave;

192.168.200.105--client上查询

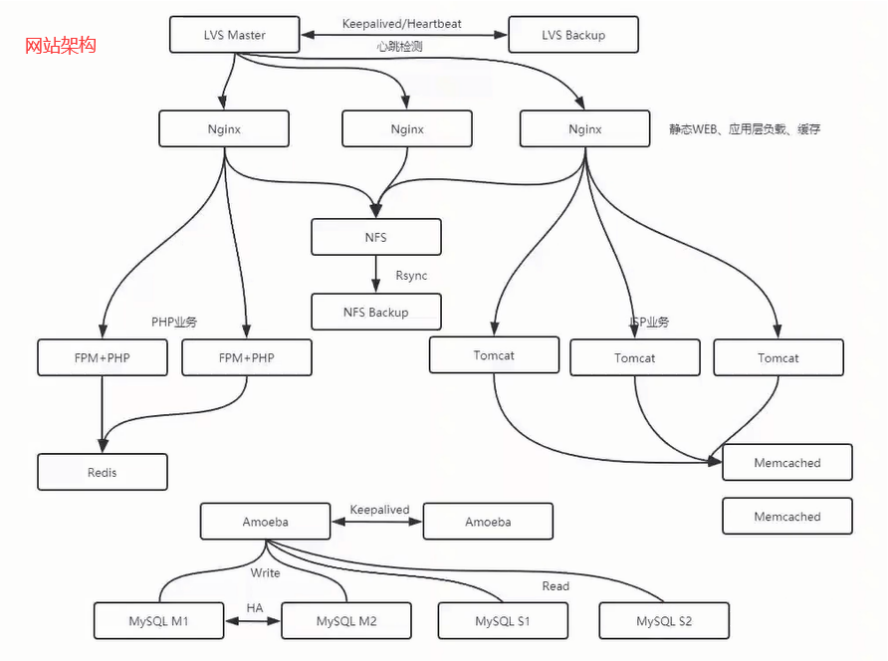
MySQL [db_test]> select * from student;

```
MySQL [db_test]> select * from student;
+------+-------+-----------------+
| id   | name  | address         |
+------+-------+-----------------+
|    2 | sofia | this_is_slave1  |
|    1 | sofia | this_is_master  |
|    4 | sofia | this_is_wri_test |
+------+-------+-----------------+
3 rows in set (0.01 sec)

MySQL [db_test]> select * from student;
+------+-------+-----------------+
| id   | name  | address         |
+------+-------+-----------------+
|    3 | sofia | this_is_slave2  |
|    1 | sofia | this_is_master  |
|    4 | sofia | this_is_wri_test |
+------+-------+-----------------+
3 rows in set (0.01 sec)
```



网站架构


LVS Master
Keepalived/Heartbeat
心跳检测
LVS Backup
Nginx
Nginx
Nginx
静态WEB、应用层负载、缓存
NFS
Rsync
NFS Backup
PHP业务
JSP业务
FPM+PHP
FPM+PHP
Tomcat
Tomcat
Tomcat
Redis
Memcached
Memcached
Amoeba
Keepalived
Amoeba
Write
Read
HA
MySQL M1
MySQL M2
MySQL S1
MySQL S2