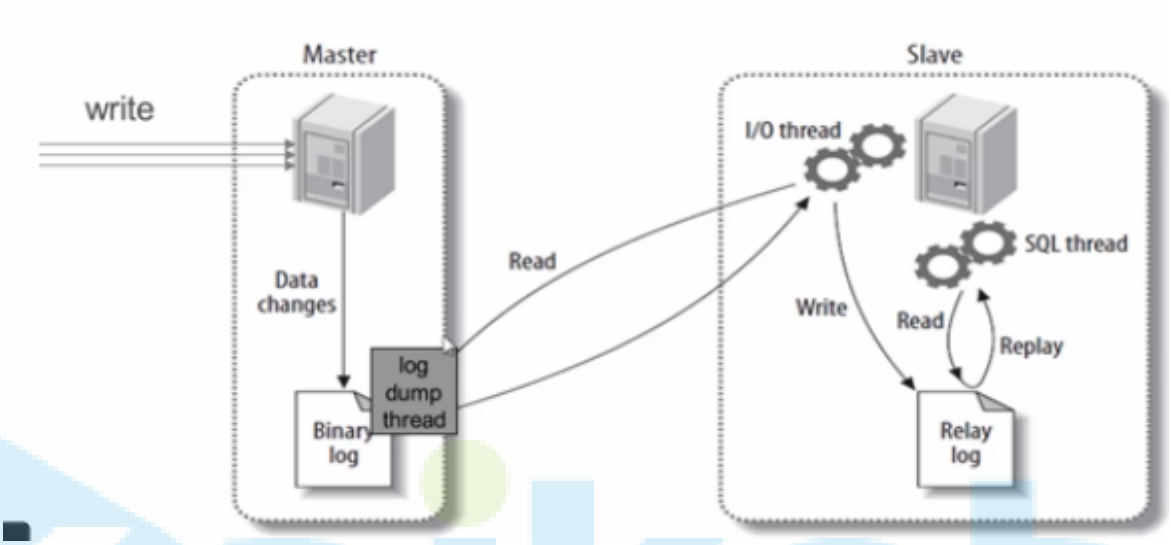
MySQL集群篇

一、集群搭建之主从复制

1、主从复制原理



2、binlog和relay日志

bin log:

bin log 记录所有数据的更改，可用于本机数据恢复和主从同步。

relay（中继） log:

- Mysql 主节点将binlog写入本地，从节点定时请求增量binlog，主节点将binlog同步到从节点。
- 从节点单独进程会将binlog 拷贝至本地 relaylog中。
- 从节点定时重放relay log。

1）binlog的三种模式

1.statement level模式

每一条会修改数据的sql都会记录到master的bin-log中。slave在复制的时候sql进程会解析成和原来master端执行过的相同的sql来再次执行。
优点：statement level下的优点，首先就是解决了row level下的缺点，不需要记录每一行数据的变化，减少bin-log日志量，节约io，提高性能。因为他只需要记录在master上所执行的语句的细节，以及执行语句时候的上下文的信息。
缺点：由于它是记录的执行语句，所以为了让这些语句在slave端也能正确执行，那么他还必须记录每条语句在执行的时候的一些相关信息，也就是上下文信息，以保证所有语句在slave端被执行的时候能够得到和在master端执行时候相同的结果。另外就是,由于mysql现在发展比较快，很多的新功能加入，使mysql的复制遇到了不小的挑战,自然复制的时候涉及到越复杂的内容，bug也就越容易出现。在statement level下，目前已经发现的就有不少情况会造成mysql的复制问题，主要是修改数据的时候使用了某些特定的函数或者功能的时候会出现，比如sleep()在有些版本就不能正确复制。

2.rowlevel模式

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

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

2）开启binlog

修改my.cnf文件

在[mysqld]段下添加：

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

sync_binlog参数：

0：存储引擎不进行binlog的刷新到磁盘，而由操作系统的文件系统控制缓存刷新。

1：每提交一次事务，存储引擎调用文件系统的sync操作进行一次缓存的刷新，这种方式最安全，但性能较低。

n：当提交的日志组=n时，存储引擎调用文件系统的sync操作进行一次缓存的刷新。

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

3）调整binlog日志模式

查看binlog的日志模式：

```
mysql> show variables like 'binlog_format';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.00 sec)
```

调整binlog的日志模式：

binlog的三种格式：`STATEMENT`、`ROW`、`MIXED`。

```
mysql> set binlog_format=STATEMENT;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'binlog_format';
+---------------+-----------+
| Variable_name | Value     |
+---------------+-----------+
| binlog_format | STATEMENT |
+---------------+-----------+
1 row in set (0.00 sec)
```

4）查看bin log和relay log日志

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

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

3、基于binlog主从复制

1）关闭主从机器的防火墙

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

2）主服务器配置

查看binlog是否开启可以使用命令：

```
mysql> show variables like 'log_bin%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| log_bin                         | OFF   |
| log_bin_basename                |       |
| log_bin_index                   |       |
| log_bin_trust_function_creators | OFF   |
| log_bin_use_v1_row_events       | OFF   |
+---------------------------------+-------+
5 rows in set (0.12 sec)
```

log_bin如果是`OFF`代表是未开启状态。

第一步：修改my.cnf文件

在[mysqld]段下添加：

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

第二步：重启mysql服务

```
systemctl restart mysqld
```

第三步：主机给从机授备份权限

注意：先要登录到MySQL命令客户端

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

示例：

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

注意事项：

```
一般不用root帐号，“%”表示所有客户端都可能连，只要帐号，密码正确，此处可用具体客户端IP代替，如192.168.145.226，加强安全。
```

> mysql5.7对密码的强度是有要求的，必须是字母+数字+符号组成的，可以使用如下方法调整密码强度
>
> 设置密码长度最低位数
> `mysql> set global validate_password_length=4;`
> 设置密码强度级别
> `mysql> set global  validate_password_policy=0;`
>
> validate_password_policy有以下取值：

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

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

第四步：刷新权限

```
mysql> FLUSH PRIVILEGES;
```

第五步：查询master的状态

```
mysql> show master status;

+------------------+-- -------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |      410 | hello        | mysql            |
+------------------+----------+--------------+------------------+

1 row in set
```

3）从服务器配置

第一步：修改my.conf文件

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

第二步：删除UUID文件

如果出现此错误：

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

原因：

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

解决方法：

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

第三步：重启并登录到MySQL进行配置从服务器

```
mysql>change master to
master_host='192.168.68.132',
master_port=3306,
master_user='root',
master_password='root',
master_log_file='mysql-bin.000002',
master_log_pos=1190,
MASTER_AUTO_POSITION=0;
```

注意：

语句中间不要断开，`master_port`为mysql服务器端口号(**无引号**)，`master_user`为执行同步操作的数据库账户，“410”无单引号(此处的`410`就是`show master status`中看到的`position`的值，这里的`mysql-bin.000001`就是`file`对应的值)。

第四步：启动从服务器复制功能

```
mysql>start slave;
```

第五步：检查从服务器复制功能状态

```
mysql> show slave status \G;

.....(省略部分)
Slave_IO_Running: Yes //此状态必须YES
Slave_SQL_Running: Yes //此状态必须YES
.....(省略部分)
```

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

查询创建工具 查询编辑器

```
1  show slave status
```

信息 结果1 概况 状态

Relay_Master_Log_File	Slave_IO_Running	Slave_SQL_Running	Replicate_Do_DB
mysql-bin.000001	Yes	Yes	

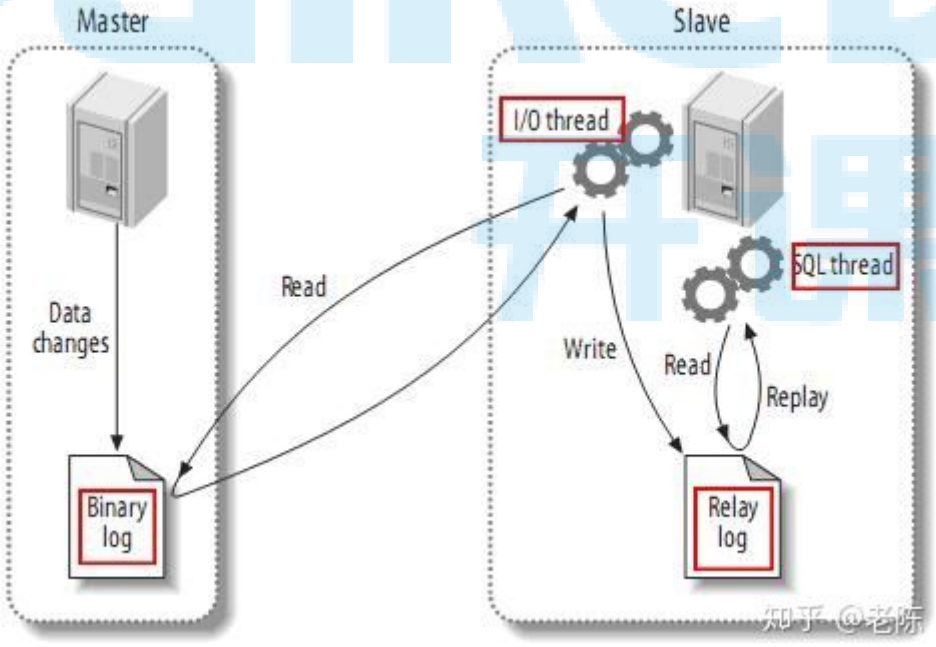
以上操作过程，从服务器配置完成。

4）测试

搭建成功之后，往主机中插入数据，看看从机中是否有数据

4、主从同步延迟的原因及解决办法

mysql 用主从同步的方法进行读写分离，减轻主服务器的压力的做法现在在业内做的非常普遍。 主从同步基本上能做到实时同步。我从别的网站借用了主从同步的原理图。



在配置好了， 主从同步以后， 主服务器会把更新语句写入binlog, 从服务器的IO 线程(这里要注意，5.6.3 之前的IO线程仅有一个，5.6.3之后的有多线程去读了，速度自然也就加快了)回去读取主服务器的binlog 并且写到从服务器的Relay log 里面，然后从服务器的 的SQL thread 会一个一个执行 relay log 里面的sql， 进行数据恢复。

relay 就是 传递, relay race 就是接力赛的意思

1、主从同步的延迟的原因

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

2、主从同步延迟的解决办法

实际上主从同步延迟根本没有什么一招制敌的办法， 因为所有的SQL必须都要在从服务器里面执行一遍，但是主服务器如果不断的有更新操作源源不断的写入， 那么一旦有延迟产生， 那么延迟加重的可能性就会原来越大。 当然我们可以做一些缓解的措施。

a. 我们知道因为主服务器要负责更新操作， 他对安全性的要求比从服务器高， 所有有些设置可以修改，比如sync_binlog=1，innodb_flush_log_at_trx_commit = 1 之类的设置，而slave则不需要这么高的数据安全，完全可以讲sync_binlog设置为0或者关闭binlog，innodb_flushlog，innodb_flush_log_at_trx_commit 也 可以设置为0来提高sql的执行效率 这个能很大程度上提高效率。另外就是使用比主库更好的硬件设备作为slave。

b. 就是把，一台从服务器当度作为备份使用， 而不提供查询， 那边他的负载下来了， 执行relay log 里面的SQL效率自然就高了。

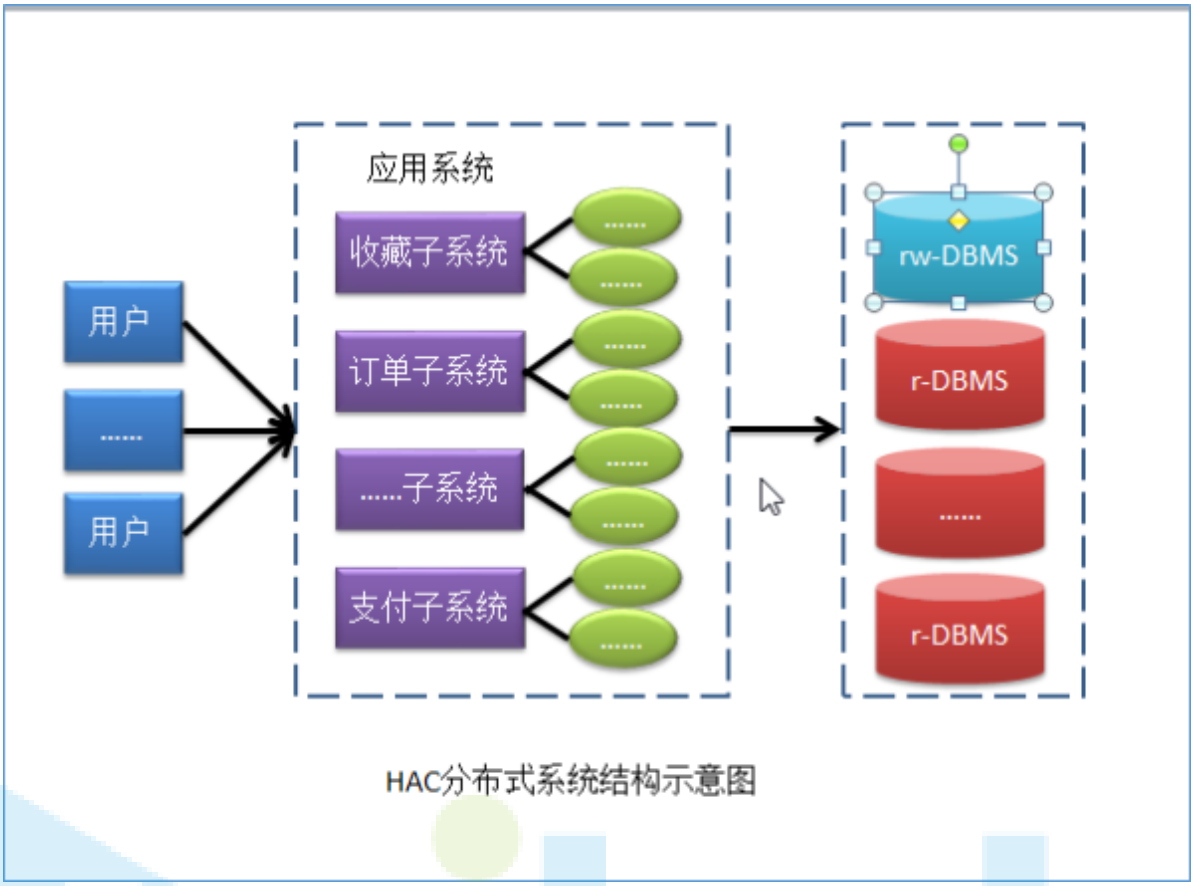
c. 增加从服务器喽，这个目的还是分散读的压力， 从而降低服务器负载。

3、判断主从延迟的方法

MySQL提供了从服务器状态命令，可以通过 `show slave status` 进行查看， 比如可以看看Seconds_Behind_Master参数的值来判断，是否有发生主从延时。
其值有这么几种：
NULL： 表示io_thread或是sql_thread有任何一个发生故障，也就是该线程的Running状态是No,而非Yes.
0： 该值为零，是我们极为渴望看到的情况，表示主从复制状态正常

二、集群搭建之读写分离

1、读写分离的理解



HAC分布式系统结构示意图

名词解释：

```
HAC：High Availability Cluster，高可用集群
```

注意事项：

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

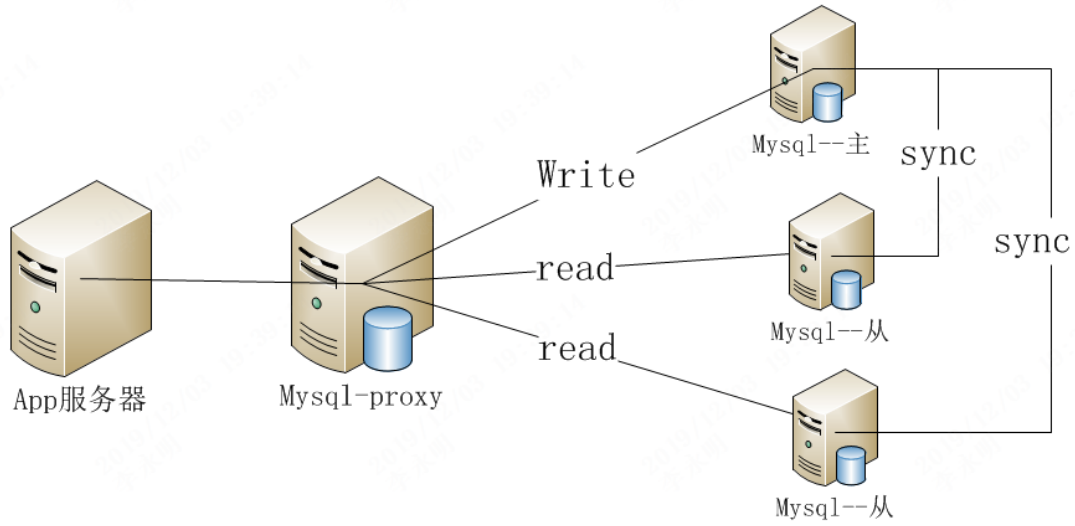
2、读写分离演示需求

```
MySQL master：132

MySQL slave ：133
```

3、MySQL-Proxy

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



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

4、Atlas

1）简介

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

官方文档：

https://github.com/Qihoo360/Atlas/blob/master/README_ZH.md

2）下载安装

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

下载好了之后，进行安装

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

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

```
[root@mysql132 Downloads]# cd /usr/local/mysql-proxy/
[root@mysql132 mysql-proxy]# ll
total 4
drwxr-xr-x 2 root root   71 Sep 28 17:02 bin
drwxr-xr-x 2 root root   21 Sep 28 17:11 conf
drwxr-xr-x 3 root root 4096 Sep 28 17:02 lib
drwxr-xr-x 2 root root   36 Sep 28 17:12 log
[root@mysql132 mysql-proxy]#
```

bin目录下放的都是可执行文件

1. “encrypt”是用来生成MySQL密码加密的，在配置的时候会用到
2. “mysql-proxy”是MySQL自己的读写分离代理
3. “mysql-proxyd”是360弄出来的，后面有个“d”，服务的启动、重启、停止。都是用他来执行的

conf目录下放的是配置文件

1. “test.cnf”只有一个文件，用来配置代理的，可以使用vim来编辑

lib目录下放的是一些包，以及Atlas的依赖

log目录下放的是日志，如报错等错误信息的记录

3）配置

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

```
[root@localhost bin]# ./encrypt root
DAJnl8cVzy8=
```

配置Atlas，使用vim进行编辑

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

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

> 这是用来登录到Atlas的管理员的账号与密码，与之对应的是“#Atlas监听的管理接口IP和端口”，也就是说需要设置管理员登录的端口，才能进入管理员界面，默认端口是2345，也可以指定IP登录，指定IP后，其他的IP无法访问管理员的命令界面。方便测试，我这里没有指定IP和端口登录。
>
> ```
> #管理接口的用户名
> admin-username = admin
> #管理接口的密码
> admin-password = admin
> ```
>
> 这是用来配置主数据的地址与从数据库的地址，这里配置的主数据库是132，从数据库是133
>
> ```
> #Atlas后端连接的MySQL主库的IP和端口，可设置多项，用逗号分隔
> proxy-backend-addresses = 192.168.246.132:3306
>
> #Atlas后端连接的MySQL从库的IP和端口，@后面的数字代表权重，用来作负载均衡，若省略则默认为1，可设置多项，用逗号分隔
> proxy-read-only-backend-addresses =
> 192.168.246.132:3306@1,192.168.246.133:3306@2
> ```
>
> 这个是用来配置MySQL的账户与密码的，我的MySQL的用户是buck,密码是hello,刚刚使用Atlas提供的工具生成了对应的加密密码

```
#用户名与其对应的加密过的MySQL密码，密码使用PREFIX/bin目录下的加密程序encrypt加密，下行的user1和user2为示例，将其替换为你的MySQL的用户名和加密密码！
pwds = root:DAJnl8cVzy8=
```

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

```
#Atlas监听的工作接口IP和端口
proxy-address = 0.0.0.0:1234

#Atlas监听的管理接口IP和端口
admin-address = 0.0.0.0:2345
```

启动Atlas

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

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

```
[root@localhost bin]# mysql -h127.0.0.1 -P2345 -uuser -ppwd
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.0.99-agent-admin

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

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

```
mysql> select * from help;
+---------------------------+-------------------------------------------
-----------+
| command                   | description
          |
+---------------------------+-------------------------------------------
-----------+
| SELECT * FROM help        | shows this help
          |
```

```
| SELECT * FROM backends      | lists the backends and their state
          |
| SET OFFLINE $backend_id     | offline backend server, $backend_id is
backend_ndx's id |
| SET ONLINE $backend_id      | online backend server, ...
          |
| ADD MASTER $backend         | example: "add master 127.0.0.1:3306", ...
          |
| ADD SLAVE $backend          | example: "add slave 127.0.0.1:3306", ...
          |
| REMOVE BACKEND $backend_id | example: "remove backend 1", ...
          |
| SELECT * FROM clients       | lists the clients
          |
| ADD CLIENT $client          | example: "add client 192.168.1.2", ...
          |
| REMOVE CLIENT $client       | example: "remove client 192.168.1.2", ...
          |
| SELECT * FROM pwds          | lists the pwds
          |
| ADD PWD $pwd                | example: "add pwd user:raw_password", ...
          |
| ADD ENPWD $pwd              | example: "add enpwd user:encrypted_password",
...       |
| REMOVE PWD $pwd             | example: "remove pwd user", ...
          |
| SAVE CONFIG                 | save the backends to config file
          |
| SELECT VERSION              | display the version of Atlas
          |
+-----------------------------+-----------------------------------------------
-----------+
16 rows in set (0.00 sec)

mysql>
```

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

```
[root@localhost bin]# mysql -h127.0.0.1 -P1234 -uroot -proot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.0.81-log

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql>
```

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

4）读写分离测试

使用语句`mysql -uroot -proot -P1234 --protocol=tcp -e"select @@hostname"`来测试读写分离。

```
[root@mysql132 bin]# mysql -uroot -proot -P1234 --protocol=tcp -e"select @@hostname"
mysql: [Warning] Using a password on the command line interface can be insecure.
+------------+
| @@hostname |
+------------+
| mysql132   |
+------------+
[root@mysql132 bin]# mysql -uroot -proot -P1234 --protocol=tcp -e"select @@hostname"
mysql: [Warning] Using a password on the command line interface can be insecure.
+------------+
```

```
| @@hostname |
+------------+
| mysql133   |
+------------+
[root@mysql132 bin]# mysql -uroot -proot -P1234 --protocol=tcp -e"select @@hostname"
mysql: [Warning] Using a password on the command line interface can be insecure.
+------------+
| @@hostname |
+------------+
| mysql133   |
+------------+
[root@mysql132 bin]# mysql -uroot -proot -P1234 --protocol=tcp -e"select @@hostname"
mysql: [Warning] Using a password on the command line interface can be insecure.
+------------+
| @@hostname |
+------------+
| mysql132   |
+------------+
```

5、MySQL Router

1）简介

MySQL Router最早是作为MySQL-Proxy的替代方案出现的。作为一个轻量级中间件，MySQL Router可在应用程序和后端MySQL服务器之间提供透明路由和负载均衡，从而有效提高MySQL数据库服务的高可用性与可伸缩行。

MySQL Router 2.0是其初始版本，适用于MySQL Fabric用户，但已被弃用，不再支持。MySQL Router 2.1为支持MySQL InnoDB Cluster而引入，MySQL Router 8.0则是MySQL Router 2.1上的扩展，版本号与MySQL服务器版本号保持一致。即Router 2.1.5作为Router 8.0.3（以及MySQL Server 8.0.3）发布，2.1.x分支被8.0.x取代。这两个分支完全兼容。当前最新版本为8.0.17，MySQL强烈建议使用Router 8与MySQL Server 8和5.7一起使用。

2）下载安装

```
> wget https://cdn.mysql.com//Downloads/MySQL-Router/mysql-router-8.0.20-el7-x86_64.tar.gz
```

MySQL Router的安装过程依赖于所使用的操作系统和安装介质，二进制包的安装通常非常简单，而源码包则需要先编译再安装。例如在Linux上的安装最新的MySQL Router二进制包，只需要用mysql用户执行一条解压命令就完成了：

```
> tar zxf mysql-router-8.0.20-el7-x86_64.tar.gz
```

3）配置

创建mysqlrouter.conf并写入如下内容：

```
[logger]
level = INFO

```

```
[routing:secondary]
bind_address = localhost
bind_port = 7001
destinations = 192.168.68.132:3306,192.168.68.133:3306
routing_strategy = round-robin

[routing:primary]
bind_address = localhost
bind_port = 7002
destinations = 192.168.68.132:3306,192.168.68.133:3306
routing_strategy = first-available
```

1. 这里设置了两个路由策略：通过本地7001端口，循环连接到192.168.68.132:3306、192.168.68.133:3306两个MySQL实例，由round-robin路由策略所定义；
2. 通过本地7002端口配置了MySQL写入实例，并设置首个可用策略。首个可用策略使用目标列表中的第一个可用服务器，即当192.168.68.132:3306可用时，所有7002端口的连接都转发到它，否则转发到后面的服务器，以此类推。Router不会检查数据包，也不会根据分配的策略或模式限制连接，因此应用程序可以据此确定将读写请求发送到不同的服务器。
3. 本例中可将读请求发送到本地7001端口，将读负载均衡到三台服务器。同时将写请求发送到7002，这样只写一个服务器，从而实现的读写分离。

4）启动并测试

进入mysql-router/bin目录下，启动mysqlrouter：

```
./mysqlrouter -c mysqlrouter.conf &
```

测试：

```
[root@mysql132 bin]# mysql -uroot -proot -P7001 --protocol=tcp -e"select @@hostname"
```

```
[root@mysql132 bin]# mysql -uroot -proot -P7001 --protocol=tcp -e"select @@hostname"
mysql: [Warning] Using a password on the command line interface can be insecure.
+-------------+
| @@hostname |
+-------------+
| mysql132   |
+-------------+
[root@mysql132 bin]# mysql -uroot -proot -P7001 --protocol=tcp -e"select @@hostname"
mysql: [Warning] Using a password on the command line interface can be insecure.
+-------------+
| @@hostname |
+-------------+
| mysql133   |
+-------------+
[root@mysql132 bin]#
```

```
[root@mysql132 bin]# mysql -uroot -proot -P7002 --protocol=tcp -e"select @@hostname"
```

```
mysql: [Warning] Using a password on the command line interface can be insecure.
+------------+
| @@hostname |
+------------+
| mysql132   |
+------------+
[root@mysql132 bin]# mysql -uroot -proot -P7002 --protocol=tcp -e"select @@hostname"
mysql: [Warning] Using a password on the command line interface can be insecure.
+------------+
| @@hostname |
+------------+
| mysql132   |
+------------+
[root@mysql132 bin]#
```

由上可见，发送到本地7001端口的请求，被循环转发到三个服务器，而发送到本地7002端口的请求，全部被转发到192.168.68.132:3306。

routing_strategy是MySQL Router的核心选项，从8.0.4版本开始引入，当前有效值为first-available、next-available、round-robin、round-robin-with-fallback。顾名思义，该选项实际控制路由策略，即客户端请求最终连接到哪个MySQL服务器实例。相对于以前版本mode的选项，routing_strategy选项更为灵活，并且不能同时设置routing_strategy和mode，静态路由的设置只能选择其中之一。对于InnoDB Cluster而言，该设置时可选的，缺省使用round-robin策略。

- **round-robin**：每个新连接都以循环方式连接到下一个可用的服务器，以实现负载平衡。
- **round-robin-with-fallback**：用于InnoDB Cluster。每个新的连接都以循环方式连接到下一个可用的SECONDARY服务器。如果SECONDARY服务器不可用，则以循环方式使用PRIMARY服务器。
- **first-available**：新连接从目标列表路由到第一个可用服务器。如果失败，则使用下一个可用的服务器，如此循环，直到所有服务器都不可用为止。
- **next-available**：与first-available类似，新连接从目标列表路由到第一个可用服务器。与first-available不同的是，如果一个服务器被标记为不可访问，那么它将被丢弃，并且永远不会再次用作目标。重启Router后，所有被丢弃服务器将再次可选。此策略向后兼容MySQL Router 2.x中mode为read-write的行为。

三、基于主从复制的高可用方案（了解）

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

在这个方案里，有几个需要注意的地方：

- 把两个节点的**auto_increment_increment**（自增起始值）和**auto_increment_offset**（自增步长）设成不同值。其目的是为了避免master节点意外宕机时，可能会有部分binlog未能及时复制到slave上被应用，从而会导致slave新写入数据的自增值和原先master上冲突了，因此一开始就使其错开；当然了，如果有合适的容错机制能解决主从自增ID冲突的话，也可以不这么做；
- slave节点服务器配置不要太差，否则更容易导致复制延迟。作为热备节点的slave服务器，硬件配置不能低于master节点；
- 如果对延迟问题很敏感的话，可考虑使用MariaDB分支版本，或者直接上线MySQL 5.7最新版本，利用多线程复制的方式可以很大程度降低复制延迟；
- keepalived的检测机制需要适当完善，不能仅仅只是检查mysqld进程是否存活，或者MySQL服务端口是否可通，还应该进一步做数据写入或者运算的探测，判断响应时间，如果超过设定的阈值，就可以启动切换机制；

- keepalived最终确定进行切换时，还需要判断slave的延迟程度。需要事先定好规则，以便决定在延迟情况下，采取直接切换或等待何种策略。直接切换可能因为复制延迟有些数据无法查询到而重复写入；
- keepalived或heartbeat自身都无法解决**脑裂**的问题，因此在进行服务异常判断时，可以调整判断脚本，通过对第三方节点补充检测来决定是否进行切换，可降低**脑裂**问题产生的风险。

双节点主从+keepalived/heartbeat方案架构示意图见下：