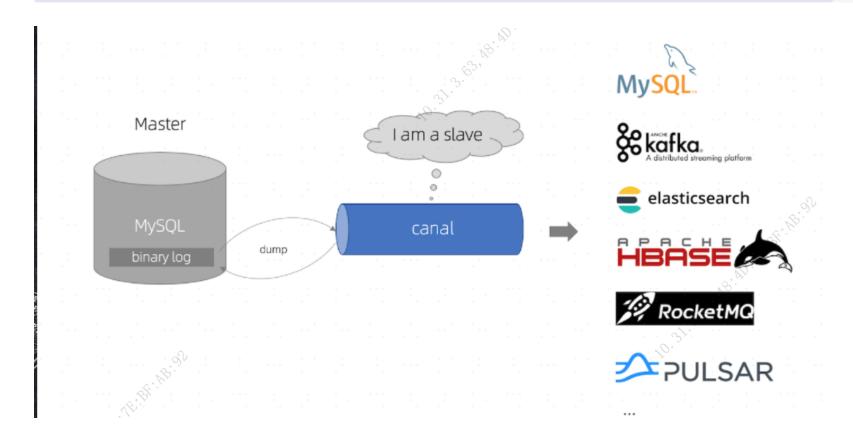
## 一、简介

- 1 1.canal,译意为水道/管道/沟渠,阿里开源的框架,主要用途是基于 MySQL 数据库增量日志解析,提供增量数据订阅和消费。
- 2 2.canal的工作原理就是把自己伪装成MySQL slave,模拟MySQL slave的交互协议向MySQL Mater发送 dump协议,
- MySQL mater收到canal发送过来的dump请求,开始推送binary log给canal,然后canal解析binary log,再发送到存储目的地,比如MySQL,等等。



# 二、准备MySQL

```
1 //支持版本 5.1.x , 5.5.x , 5.6.x , 5.7.x , 8.0.x
2 1.创建挂载目录
3 mkdir -vp /home/docker/mysql/{conf,data,logs}
5 2.挂载并启动容器
  docker run --name mysql \
   -e TZ=Asia/Shanghai \
8 -e MYSQL_ROOT_PASSWORD=root \
9 -v /home/docker/mysql/data:/var/lib/mysql \
   -v /home/docker/mysql/conf:/etc/mysql/mysql.conf.d \
  -v /home/docker/mysql/logs:/logs \
   -p 3306:3306 \
13 -p 33060:33060 \
14 --privileged=true \
15 --restart=always \
16 -d mysql:5.7.21 \
17 --character-set-server=utf8mb4 \
   --collation-server=utf8mb4_general_ci
20 3.配置mysqld.cnf
21 touch /home/docker/mysql/conf/mysqld.cnf
22 //写入配置
   cat > /home/docker/mysql/conf/mysqld.cnf << EOF</pre>
24 [mysqld]
25 log-bin=mysql-bin
   binlog-format=row
27 server_id=3306
28 log_timestamps=SYSTEM
29 default-time-zone='+8:00'
30 EOF
```

```
31 //重启容器生效
  docker restart mysql;
  4. 创建一个用户并授权
35 docker exec -it mysql bash
36 > mysql -uroot -proot;
   //如果执行报已经存在该用户 drop user canal@'%'; flush privileges;
38 > create user canal identified by 'canal';
39 > grant SELECT, REPLICATION SLAVE, REPLICATION CLIENT on *.* to 'canal'@'%';
40 > flush privileges;
41
42 5. 检查状态
43 //是否为ON,为OFF重启容器
44 > show variables like 'log_bin';
45 //查看binlog
46 //记录这两个参数 mysql-bin.000001 | 154
47 > show master status;
48
```

# 三、搭建 Canal 服务端

37 canal.admin.user = admin

```
1 1.创建挂载目录
2 mkdir -p /home/docker/canal/{conf,logs}
  touch /home/docker/canal/conf/{instance.properties, canal.properties}
5 2. 写入配置
   [instance.properties]
7 cat > /home/docker/canal/conf/instance.properties << EOF</pre>
8 # 数据库地址
9 canal.instance.master.address=10.207.0.169:3306
10 # binlog日志名称
   canal.instance.master.journal.name=mysql-bin.000001
12 # mysql主库链接时起始的binlog偏移量
  canal.instance.master.position=154
  # mysql主库链接时起始的binlog的时间戳
15 canal.instance.master.timestamp=
16 canal.instance.master.gtid=
17 # 在MySQL服务器授权的账号密码
18 canal.instance.dbUsername=canal
19 canal.instance.dbPassword=canal
20 # 字符集
21 canal.instance.connectionCharset = UTF-8
22 # enable druid Decrypt database password
23 canal.instance.enableDruid=false
24 # 监听所有表,也可以写具体的表名用逗号隔开
25 canal.instance.filter.regex=.*\\..*
26 # 数据解析表的黑名单,多个表用逗号隔开
  canal.instance.filter.black.regex=
28
29
   【canal.properties】
30
31 //select password('123456') 生成密码密文 *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9
32 cat > /home/docker/canal/conf/canal.properties << EOF</pre>
  # canal admin config
  canal.register.ip = 10.207.0.169
  canal.admin.manager = 10.207.0.169:8090
  canal.admin.port = 11110
```

```
38 canal.admin.passwd = 6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9
39 canal.admin.register.auto = true
40 canal.admin.register.cluster =
EOF
42
43 3.挂载并启动容器
44 // docker stop canal;docker rm canal
45 docker run --name canal -p 10010:11111 -d \
-v /home/docker/canal/conf/instance.properties:/home/admin/canal-server/conf/example/instance.properties \
-v /home/docker/canal/logs/:/home/admin/canal-server/logs/ \
--privileged=true \
49 canal/canal-server:v1.1.5
50
51 //没生效,暂时注释掉
52 //-v /home/docker/canal/conf/canal.properties:/home/admin/canal-server/conf/canal.properties \
53
```

```
[root@node2 canal]# docker logs canal
DOCKER_DEPLOY_TYPE=VM
==> INIT /alidata/init/02init-sshd.sh
==> EXIT CODE: 0
==> INIT /alidata/init/fix-hosts.py
==> EXIT CODE: 0
==> INIT DEFAULT
Generating SSH1 RSA host key: [ OK ]
Starting sshd: [ OK ]
Starting crond: [ OK ]
==> INIT DONE
==> RUN /home/admin/app.sh
==> START ...
start canal ...
start canal successful
==> START SUCCESSFUL ...
[root@node2 canal]#
```

### 四、客户端连接

```
1 1.依赖
2 <!-- canal -->
3 <dependency>
      <groupId>com.alibaba.otter
4
      <artifactId>canal.client</artifactId>
      <version>1.1.4
  </dependency
8
9 2.工具类
import com.alibaba.otter.canal.client.CanalConnector;
import com.alibaba.otter.canal.client.CanalConnectors;
import com.alibaba.otter.canal.protocol.CanalEntry;
import com.alibaba.otter.canal.protocol.Message;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.stereotype.Component;
16
  import java.net.InetSocketAddress;
  import java.util.List;
  import static com.alibaba.otter.canal.protocol.CanalEntry.*;
21
   /**
22
   * canal 拦截 mysql binlog
24
25 @Component
26 public class CannalClient implements InitializingBean {
```

```
28
      private final static int BATCH_SIZE = 1000;
29
      @Override
30
      public void afterPropertiesSet() throws Exception {
          // 创建链接
          CanalConnector connector = CanalConnectors.newSingleConnector(new InetSocketAddress("10.207.0.169", 10010), "e
   "canal");
          try {
34
              //打开连接
              connector.connect();
              //订阅数据库表,全部表
              connector.subscribe(".*\\..*");
38
              //回滚到未进行ack的地方,下次fetch的时候,可以从最后一个没有ack的地方开始拿
              connector.rollback();
40
              while (true) {
41
42
                  // 获取指定数量的数据
                  Message message = connector.getWithoutAck(BATCH_SIZE);
43
44
                  //获取批量ID
45
                  long batchId = message.getId();
46
                  //获取批量的数量
47
                  int size = message.getEntries().size();
48
                  //如果没有数据
49
                  if (batchId == -1 || size == 0) {
50
                     try {
51
                          //线程休眠2秒
52
                         Thread.sleep(2000);
                     } catch (InterruptedException e) {
54
                         e.printStackTrace();
                      }
56
                  } else {
57
                     //如果有数据,处理数据
                      printEntry(message.getEntries());
59
60
                  //进行 batch id 的确认。确认之后,小于等于此 batchId 的 Message 都会被确认。
61
                  connector.ack(batchId);
62
63
          } catch (Exception e) {
64
              e.printStackTrace();
65
          } finally {
66
              connector.disconnect();
67
68
69
70
71
       * 打印canal server解析binlog获得的实体类信息
72
       */
73
      private static void printEntry(List<Entry> entrys) {
74
75
          for (Entry entry : entrys) {
              if (entry.getEntryType() == EntryType.TRANSACTIONBEGIN || entry.getEntryType() == EntryType.TRANSACTIONEND
76
                  //开启/关闭事务的实体类型, 跳过
77
                  continue;
78
79
              //RowChange对象,包含了一行数据变化的所有特征
80
              //比如isDdl 是否是ddl变更操作 sql 具体的ddl sql beforeColumns afterColumns 变更前后的数据字段等等
81
              RowChange rowChage;
82
              try {
83
                  rowChage = RowChange.parseFrom(entry.getStoreValue());
84
              } catch (Exception e) {
85
                  throw new RuntimeException("ERROR ## parser of eromanga-event has an error , data:" + entry.toString()
86
```

```
//获取操作类型: insert/update/delete类型
88
               EventType eventType = rowChage.getEventType();
89
               //打印Header信息
90
               System.out.println();
91
               System.out.println(String.format("> binlog及偏移量[%s:%s] , 库/表[%s,%s] , 操作类型[%s]",
92
                       entry.getHeader().getLogfileName(), entry.getHeader().getLogfileOffset(),
93
                       entry.getHeader().getSchemaName(), entry.getHeader().getTableName(),
94
                       eventType));
95
               //判断是否是DDL语句
96
               if (rowChage.getIsDdl()) {
97
                   System.out.println("> DDL sql: " + rowChage.getSql());
98
99
               //获取RowChange对象里的每一行数据,打印出来
100
               for (CanalEntry.RowData rowData : rowChage.getRowDatasList()) {
101
                   //删除语句
                   if (eventType == EventType.DELETE) {
103
                       printColumn(rowData.getBeforeColumnsList());
104
                   } else if (eventType == EventType.INSERT) {
105
                       printColumn(rowData.getAfterColumnsList());
106
                   } else {
107
                       //变更前的数据
108
                       System.out.println(">> 变更前数据");
109
                       printColumn(rowData.getBeforeColumnsList());
110
                       //变更后的数据
111
                       System.out.println(">> 变更后数据");
112
                       printColumn(rowData.getAfterColumnsList());
113
114
115
116
117
118
       private static void printColumn(List<Column> columns) {
119
           for (Column column : columns) {
               System.out.println(column.getName() + " : " + column.getValue() + " update=" + column.getUpdated());
121
122
124 }
125
```

## 五、测试

17 show master status;

```
1 use sf_mall;
2
3 -- mysql-bin.000001,12232,"","",""
4 show master status;
5
6 -- 1.insert
7 insert into t_order(order_no, amount, count, addr) values
8 ('202205270006', 120, 1, '这是test canal的一条信息');
9
10 -- mysql-bin.000001,12578,"","",""
11 show master status;
12
13 -- 2.update
14 update t_order set count = 3 where order_no = '202205270006';
15
16 -- mysql-bin.000001,12981,"","",""
```

```
Debug:
      🗐 App 🗴
             [] TS:II: בייסון [Lestantedmatu] אורט באווי באווי באוויביים באוויביים באוויביים (הוויביים אוויביים להייסול מהיי
        [] 12:11:31.811 [restartedMain] DEBUG com.sf.App - [logStarting,56] - Running with Spring Boot v2.3.0.RELEASE, Spring v5.2.6.RELEASE
        [] 12:11:31.811 [restartedMain] INFO com.sf.App - [logStartupProfileInfo,651] - No active profile set, falling back to default profiles
1
        [] 12:11:36.672 [restartedMain] INFO o.a.c.h.Http11NioProtocol - [log,173] - Initializing ProtocolHandler ["http-nio-8088"]
   ₽
        [] 12:11:36.672 [restartedMain] INFO o.a.c.c.StandardService - [log,173] - Starting service [Tomcat]
III:
        [] 12:11:36.672 [restartedMain] INFO o.a.c.c.StandardEngine - [log,173] - Starting Servlet engine: [Apache Tomcat/9.0.35]
        [] 12:11:37.142 [restartedMain] INFO o.a.c.c.C.[.[.[/] - [log,173] - Initializing Spring embedded WebApplicationContext
•
       > binlog及偏移量[mysql-bin.000001:12448] ,库[sf_mall,t_order] ,操作类型[INSERT]
1/2
        id : 5 update=true
       order_no : 202205270006
                               update=true
Ô
        amount : 120 update=true
        count : 1 update=true
苡
        addr : 这是test canal的一条信息
                                    update=true
        phone : update=true
        create_date : 2022-05-31 12:14:40
                                        _update=true
   > binlog及偏移量[mysql-bin.000001:12786] ,库[sf_mall,t_order] ,操作类型[UPDATE]
   >> 变更前数据
```

```
id : 5 " update=false
order_no : 202205270006
                          update=false
amount: 120
               update=false
            update=false
count: 1
addr : 这是test canal的一条信息
                                update=false
phone :
           update=false
create_date : 2022-05-31 12:14:40
                                    update=false
>> 变更后数据
id : 5
         update=false
order_no : 202205270006
                          update=false
               update=false
amount: 120
            update=true
count : 3
addr : 这是test canal的一条信息
                                update=false
phone :
           update=false
create_date : 2022-05-31 12:14:40
                                    update=false
```

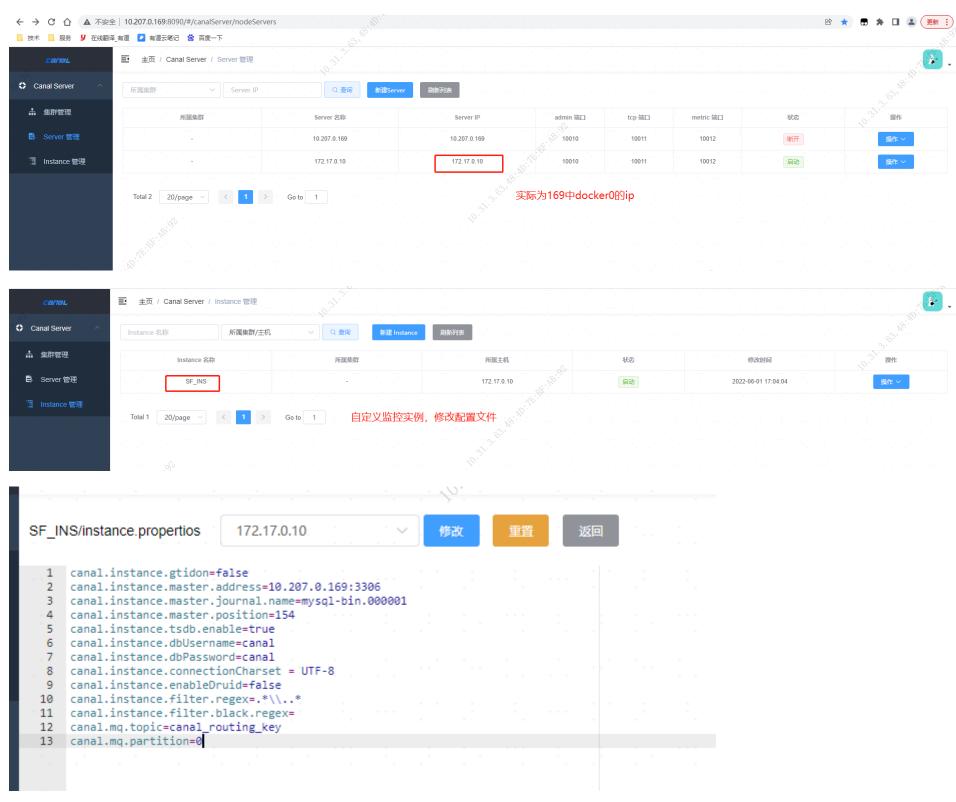
### 六、web界面

// 如需要指定外部mysql,可以使用下面脚本初始化

```
? V1.0.0_2__create_....sql 3.92KB
```

```
1 // docker stop canal-admin;docker rm canal-admin; docker stop canal-server;docker rm canal-server
2 //WEB
3 docker run -d -it \
4 -p 8090:8089 \
5 -e server.port=8089 \
6 -e canal.adminUser=admin \
7 -e canal.adminPasswd=admin \
8 --name=canal-admin \
  -m 1024m canal/canal-admin
11
12 //SERVER 11110-通信端口 11111-服务端口
13 docker run -d -it \
14 -p 10010:11110 \
15 -p 10011:11111 \
16 -p 10012:11112 \
17 -p 9100:9100 \
  -e canal.admin.manager=10.207.0.169:8090 \
19 -e canal.admin.port=10010 \
  -e canal.admin.user=admin \
```

```
22 --name=canal-server \
23 -m 4096m canal/canal-server
24
25 2.访问,发现之前启动的server没有
26 http://10.207.0.169:8090 // admin/123456 登录
27
28
29 推送到mq
http://www.voycn.com/article/shiyong-docker-bushu-canalbingjiangxiaoxituisongdao-rabbitmq
```



#### Java客户端

```
Set() throws Exception {

or = CanalConnectors.newSingleConnector(new InetSocketAddress( hostname: "10.207.0.169", port: 10011), destination "SF_INS", username: "canal", password: "canal");
```

修改一条数据

```
Debugger
     > binlog及偏移量[mysql-bin.000001:5995] ,库/表[sf_mall,t_order] ,操作类型[UPDATE]
     >> 变更前数据
     id : 4 update=false
  ⋽
Ш
      order_no : 202205270004
                        update=false
      amount: 1000 update=false
              update=false
      count : 4
      addr : 广州市番禺区星海学院
                         update=false
             update=false
%
      >> 变更后数据
0
      id : 4
            update=false
     order_no : 202205270004 update=false
文
     amount : 1000
                update=false
     count : 5 update=true
     addr : 广州市番禺区星海学院
                        update=false
      phone :
            update=false
```

SF\_INS.log 刷新 2022-06-01 17:04:07.544 [canal-instance-scan-0] INFO c.a.o.c.i.spring.support.PropertyPlaceholderConfigurer - Properties resource not found: class path resource [SF\_INS/instance.properties] cannot be opened because it does not exist 2022-06-01 17:04:07.799 [canal-instance-scan-0] INFO c.a.o.c.i.spring.support.PropertyPlaceholderConfigurer - Properties resource not found; class path resource [SF\_INS/instance.properties] cannot be opened because it does not exist 2022-06-01 17:04:08.478 [canal-instance-scan-0] INFO c.a.otter.canal.instance.spring.CanalInstanceWithSpring - start CanalInstance for 1-SF\_INS 2022-06-01 17:04:08.490 [canal-instance-scan-0] WARN c.a.o.canal.parse.inbound.mysql.dbsync.LogEventConvert - --> init table filter: 2022-06-01 17:04:08:490 [canal-instance-scan-0] WARN c.a.o.canal.parse.inbound.mysql.dbsync.LogEventConvert - --> init table black filter:  $2022-06-01\ 17:04:08.494\ [canal-instance-scan-0]\ INFO\ c.a. otter. canal. instance. core. Abstract Canal Instance - start successful....$ 2022-06-01 17:04:08.612 [destination = SF\_INS , address = /10.207.0.169:3306 , EventParser] WARN c.a.o.c.p.inbound.mysql.rds.RdsBinlogEventParserProxy - ---> begin to find start position, it will be long time for reset or first position 2022-06-01 17:04:08.637 [destination = SF\_INS], address = /10.207.0.169:3306, EventParser] WARN c.a.o.c.p.inbound.mysql.rds.RdsBinlogEventParserProxy - prepare to find start position mysql-bin.000001:4:1654072697000 2022-06-01 17:04:09.568 [destination = SF\_INS], address = /10.207.0.169:3306, EventParser] WARN c.a.o.c.p.inbound.mysql.rds.RdsBinlogEventParserProxy - ---> find start position successfully, EntryPosition[included=false.journalName=mysql-2022-06-01 17:11:18.205 [New I/O server worker #1-1] INFO c.a.otter.canal.instance.core.AbstractCanalInstance - subscribe filter change to .\*...\* 2022-06-01 17;11:18.205 [New I/O server worker #1-1] WARN c.a.o.canal.parse.inbound.mysql.dbsync.LogEventConvert - --> init table filter: ^.\*\..\*\$ 2022-06-0147:17:49.776 [Thread-3] INFO c.a.otter.canal.instance.core.AbstractCanalInstance - stop CanalInstance for null-SF\_INS 2022-06-01 17:17:50.643 [Thread-3] INFO c.a.otter.canal.instance.core.AbstractCanalInstance - stop successful.... 2022-06-01 17:17:53.458 [canal-instance-scan-0] INFO c.a.o.c.i.spring.support.PropertyPlaceholderConfigurer - Properties resource not found: class path resource [SF\_INS/instance.properties] cannot be opened because it does not exist 2022-06-01 17:17:53.679 [canal-instance-scan-0] INFO c.a.o.c.i.spring.support.PropertyPlaceholderConfigurer - Properties resource not found: class path resource [SF\_INS/instance.properties] cannot be opened because it does not exist 2022-06-01 17:17:54.479 [canal-instance-scan-0] INFO c.a.otter.canal.instance.spring.CanalInstanceWithSpring - start CannalInstance for 1-SF\_INS 2022-06-01 17:17:54.492 [canal-instance-scan-0] WARN c.a.o.canal.parse.inbound.mysql.dbsync.LogEventConvert - --> init table filter : ^.\*\..\*\$  $2022-06-01\ 17:17:54.492\ [canal-instance-scan-0]\ WARN\ c.a.o. canal. parse. inbound. mysql. dbsync. Log Event Convert---> init table black filter:$ 2022-06-01 17:17:54.529 [canal-instance-scan-0] INFO c.a.otter.canal.instance.core.AbstractCanalInstance - subscribe filter change to .\*\... 2022-06-01 17:17:54.529 [canal-instance-scan-0] WARN c.a.o.canal.parse.inbound.mysql.dbsync.LogEventConvert - --> init table filter: ^.\*\..\*\$  $2022-06-01\ 17:17:54.530\ [canal-instance-scan-0]\ INFO\ c.a. otter. canal. instance. core. Abstract Canal Instance-start successful ...$ 2022-06-01 17:17:54.603 [destination = SF\_INS , address = /10.207.0.169:3306 , EventParser] WARN c.a.o.c.p.inbound.mysql.rds.RdsBinlogEventParserProxy - ---> begin to find start position, it will be long time for reset or first position 2022-06-01 17:17:54.644 [destination = SF\_INS], address = /10.207.0.169:3306, EventParser] WARN c.a.o.c.p.inbound.mysql.rds.RdsBinlogEventParserProxy - prepare to find start position just last position ("identity":("slaveld":-1, "sourceAddress":("addres 2022-06-01 17:17:55.102 [destination = SF\_INS], address = /10.207.0.169:3306, EventParser] WARN c.a.o.c.p.inbound.mysql.rds.RdsBinlogEventParserProxy - ---> find start position successfully, EntryPosition[included=false.journalName=mysql.rds.RdsBinlogEventParserProxy - ---> f bin.000001,position=5355,serverId=3306,gtid=,timestamp=1654073746000] cost: 488ms , the next step is binlog dump 2022-06-01 17:24:04.316 [New I/O server worker #1-1] INFQ ca.otter.canal.instance.core.AbstractCanalInstance - subscribe filter change to .\*\..\* 2022-06-01 17:24:04.317 [New I/O server worker #1-1] WARN c.a.o.canal.parse.inbound.mysql.dbsync.LogEventConvert - --> init table filter : ^.\*\..\*\$ 没日志

#### 【异常】:

问题: canal容器启动失败,配置文件属于挂载外部文件,容器内没有权限读取

原因:centos7 安全子系统 Selinux 禁止了一些安全权限,导致进行挂载目录时出现这个错误

解决:可以在 docker run 命令中加入 --privileged=true 设置,给容器加上特定权限

#### 问题: 启动canal, 报canal-admin密码错误

2022-05-31 15:19:06.842 [main] INFO com.alibaba.otter.canal.deployer.CanalLauncher - ## load canal configurations
2022-05-31 15:19:07.323 [main] ERROR com.alibaba.otter.canal.deployer.CanalLauncher - ## Something goes wrong when starting up the canal Server:
com.alibaba.otter.canal.common.CanalException: load manager config failed.
Caused by: com.alibaba.otter.canal.common.CanalException: requestGet for canal config error: auth :admin is failed

原因:配置需要用 select password('xx') 生成的密文,123456 -> 6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9