


尚硅谷大数据之电商实时数仓  
之  
ODS 层实现


(作者：尚硅谷研究院)

版本：V3.0

# 第 1 章 概述

采集到 Kafka 的 topic\_log 和 topic\_db 主题的数据即为实时数仓的 ODS 层，这一层的作用是对数据做原样展示和备份。具体详细流程参照如下文档

  
用户行为采集平台  
V3.0.docx

  
业务数据采集平台  
V3.0.docx

## 1.1 数据有序

本项目要求 Flink 单个并行度的数据严格有序(主要用于数据去重),为了保证这一点,做了如下配置。

### (1) 修改 Kafka 配置文件 server.properties , 将 topic 分区数设置为 4

```
[atguigu@hadoop102 ~]$ cd /opt/module/kafka/  
[atguigu@hadoop102 kafka]$ vim config/server.properties  
# 在配置文件中修改 Kafka 主题的默认分区数  
num.partitions=4
```

三台服务器的配置文件都要修改,修改完毕重启 Kafka,不可直接分发配置文件,因为每个节点的 broker.id 必须唯一。分发后必须修改 broker.id 才能重启 Kafka,否则重启失败。

### (2) 在 Flink 程序中设置并行度为 4

Flink 程序从 Kafka 消费数据时会启动同属于一个消费者组的四个消费者,Kafka 消费者的默认分区分配策略是 Range + CooperativeSticky,消费者数和分区数相同时,每个消费者消费一个分区的数据。只要单分区数据有序,即可保证 Flink 单个并行度数据有序。

本项目 Kafka 版本为 3.0.0,在 Kafka 1.x 及之后的版本中,保证数据单分区有序,条件如下:

- 未开启幂等性

`max.in.flight.requests.per.connection` 需要设置为 1。

该参数指定了生产者在收到服务器响应之前可以发送多少条消息

➤ 开启幂等性

`max.in.flight.requests.per.connection` 需要设置小于等于 5。

原因说明：因为在 kafka1.x 以后，启用幂等后，kafka 服务端会缓存 producer 发来的最近 5 个 request 的元数据，故无论如何，都可以保证最近 5 个 request 的数据都是有序的。

默认情况下幂等性是开启的，`max.in.flight.requests.per.connection` 默认值为 5，所以单分区数据默认是有序的，不需要做任何配置。

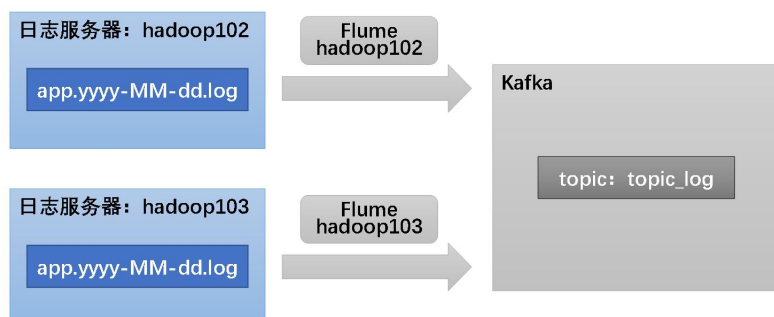
综上，我们可以保证 Flink 程序单个并行度的数据有序。

## 第 2 章 日志数据采集

### 2.1 数据通道



用户行为日志数据通道



## 2.2 上传模拟生成日志数据的 jar 包

将\2.资料\mock\日志\下文件上传到 hadoop202 的/opt/module/applog 目录下

## 2.3 根据需要修改配置文件

(1) 可在 application.yml 文件根据需求生成对应日期的用户行为日志。

```
[atguigu@hadoop202 applog]$ vim application.yml
```

(2) 可在 logback.xml 文件中配置日志生成路径

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <property name="LOG_HOME" value="/opt/module/applog/log" />
    <appender name="console"
class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%msg%n</pattern>
        </encoder>
    </appender>

    <appender name="rollingFile"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>${LOG_HOME}/app.%d{yyyy-MM-dd}.log</fileNamePattern>

        </rollingPolicy>
        <encoder>
            <pattern>%msg%n</pattern>
        </encoder>
    </appender>

    <!-- 将某一个包下日志单独打印日志 -->
    <logger name="com.atgugu.gmall2020.mock.log.util.LogUtil"
        level="INFO" additivity="false">
        <appender-ref ref="rollingFile" />
        <appender-ref ref="console" />
    </logger>

    <root level="error" >
        <appender-ref ref="console" />
    </root>
</configuration>
```

## 2.4 在 hadoop202 上生成日志并落盘

(1) 进入到/opt/module/applog 路径，执行以下命令

```
[atguigu@hadoop202 applog]$ java -jar  
gmall2020-mock-log-2021-10-10.jar
```

(2) 在/opt/module/applog/log 目录下查看生成日志

```
[atguigu@hadoop202 log]$ ll
```

## 2.5 集群日志生成脚本，落盘到 hadoop202 和 203 上

(1) 在/home/atguigu/bin 目录下创建脚本 lg.sh

```
[atguigu@hadoop202 bin]$ vim lg.sh
```

(2) 在脚本中编写如下内容

```
#!/bin/bash  
  
for i in hadoop202 hadoop203; do  
    echo "===== $i ====="  
    ssh $i "cd /opt/module/applog/; java -jar  
gmall2020-mock-log-2021-10-10.jar >/dev/null 2>&1 &"  
done
```

(3) 修改脚本执行权限

```
[atguigu@hadoop202 bin]$ chmod u+x lg.sh
```

(4) 将 202 上/opt/module/applog 内容拷贝到至 hadoop203 的/opt/module/applog/

```
rsync -av /opt/module/applog/ hadoop203:/opt/module/applog
```

(5) 启动脚本

```
[atguigu@hadoop202 module]$ lg.sh
```

(6) 分别在 hadoop202、hadoop203 的/opt/module/applog/log 目录查看生成的数据

```
[atguigu@hadoop202 logs]$ ls  
app.2020-06-14.log  
[atguigu@hadoop203 logs]$ ls  
app.2020-06-14.log
```

## 2.6 通过 flume 将落盘的日志数据采集到 kafka 主题

用户行为日志，一般是没有历史数据的，故日志只需要准备某一天的数据

(1) 准备 flume 配置文件

```
[atguigu@hadoop202 flume]$ vim job/file_to_kafka.conf  
  
#为各组件命名  
a1.sources = r1  
a1.channels = c1  
  
#描述 source
```

```

a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/module/applog/log/app.*
a1.sources.r1.positionFile = /opt/module/flume/tailedir_position.json
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = com.atguigu.flume.interceptor.ETLInterceptor$Builder

#描述 channel
a1.channels.c1.type = org.apache.flume.channel.kafka.KafkaChannel
a1.channels.c1.kafka.bootstrap.servers = hadoop102:9092,hadoop103:9092
a1.channels.c1.kafka.topic = topic_log
a1.channels.c1.parseAsFlumeEvent = false

#绑定 source 和 channel 以及 sink 和 channel 的关系
a1.sources.r1.channels = c1

```

## (2) 编写 flume 启停脚本

```

#!/bin/bash

case $1 in
"start"){
    for i in hadoop202 hadoop203
    do
        echo " -----启动 $i 采集 flume-----"
        ssh $i "nohup /opt/module/flume/bin/flume-ng agent -n a1 -c
/opt/module/flume/conf/ -f /opt/module/flume/job/file_to_kafka.conf >/dev/null
2>&1 &"
    done
};;
"stop"){
    for i in hadoop202 hadoop203
    do
        echo " -----停止 $i 采集 flume-----"
        ssh $i "ps -ef | grep file_to_kafka | grep -v grep |awk '{print
\$2}' | xargs -n1 kill -9 "
    done
};;
esac

```

## (3) 启动 Zookeeper

## (4) 启动 Kafka

**注意：为了后续 Flink 读取和 kafka 并行度一直，我们统一设置并行度为 4**

## (5) 启动一个命令行 Kafka 消费者，消费 topic\_log 主题的数据

```

/opt/module/kafka/bin/kafka-console-consumer.sh --bootstrap-server
hadoop202:9092 --topic topic_log

```

## (6) 修改两个日志服务器（hadoop202、hadoop203）中的

/opt/module/applog/application.yml 配置文件，将 mock.date 参数改为某天的日

期

(7) 执行日志生成脚本 lg.sh

(8) 执行 flume 采集脚本

```
fl.sh start
```

(9) 观察命令行 Kafka 消费者是否消费到数据。

```
e : query , item : 0 , item_type : sku_id , order : 1 , pos_id : 4 , { "display_type" : promotion , item : 0 , item_type : sku_id , order : 2 , pos_id : 3 } , { "display_type" : promotion , item : 32 , item_type : sku_id , order : 3 , pos_id : 4 } , { "display_type" : query , item : 6 , item_type : sku_id , order : 4 , pos_id : 1 } , { "display_type" : recommend , item : 27 , item_type : sku_id , order : 5 , pos_id : 4 } , { "display_type" : query , item : 33 , item_type : sku_id , order : 6 , pos_id : 5 } , { "display_type" : query , item : 32 , item_type : sku_id , order : 7 , pos_id : 5 } , { "display_type" : promotion , item : 7 , item_type : sku_id , order : 8 , pos_id : 4 } , { "display_type" : query , item : 30 , item_type : sku_id , order : 9 , pos_id : 4 } , { "page" : { "during_time" : 4357 , item : 1 , item_type : sku_id , last_page_id : home , page_id : goods_detail , source_type : query } , ts : 1645424156000 }
{ "common" : { "ar" : 370000 , ba : Xiaomi , ch : xiaomi , is_new : 0 , md : Xiaomi Mix2 , mid : mid_122084 , os : Android 11.0 , uid : 137 , vc : v2.1.132 } , start : { "entry" : icon , loading_time : 15482 , open_ad_id : 11 , open_ad_ms : 5147 , open_ad_skip_ms : 3427 } , ts : 1645424155000 }
{ "common" : { "ar" : 370000 , ba : Xiaomi , ch : xiaomi , is_new : 0 , md : Xiaomi Mix2 , mid : mid_122084 , os : Android 11.0 , uid : 137 , vc : v2.1.132 } , displays : [ { "display_type" : activity , item : 2 , item_type : activity_id , order : 1 , pos_id : 4 } , { "display_type" : activity , item : 1 , item_type : activity_id , order : 2 , pos_id : 4 } , { "display_type" : promotion , item : 25 , item_type : sku_id , order : 3 , pos_id : 5 } , { "display_type" : query , item : 15 , item_type : sku_id , order : 4 , pos_id : 3 } , { "display_type" : query , item : 24 , item_type : sku_id , order : 5 , pos_id : 5 } , { "display_type" : query , item : 4 , item_type : sku_id , order : 6 , pos_id : 2 } , { "display_type" : query , item : 28 , item_type : sku_id , order : 7 , pos_id : 4 } , { "display_type" : promotion , item : 35 , item_type : sku_id , order : 8 , pos_id : 3 } ] , page : { "during_time" : 14829 , page_id : home } , ts : 1645424155000 }
```

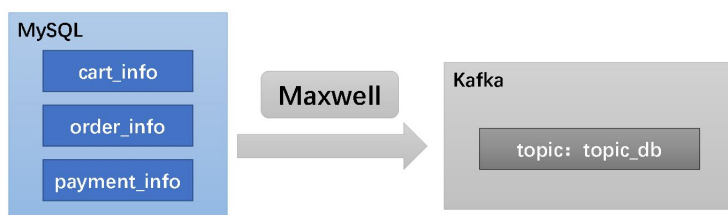
## 第 3 章 业务数据采集

### 3.1 数据通道



尚硅谷

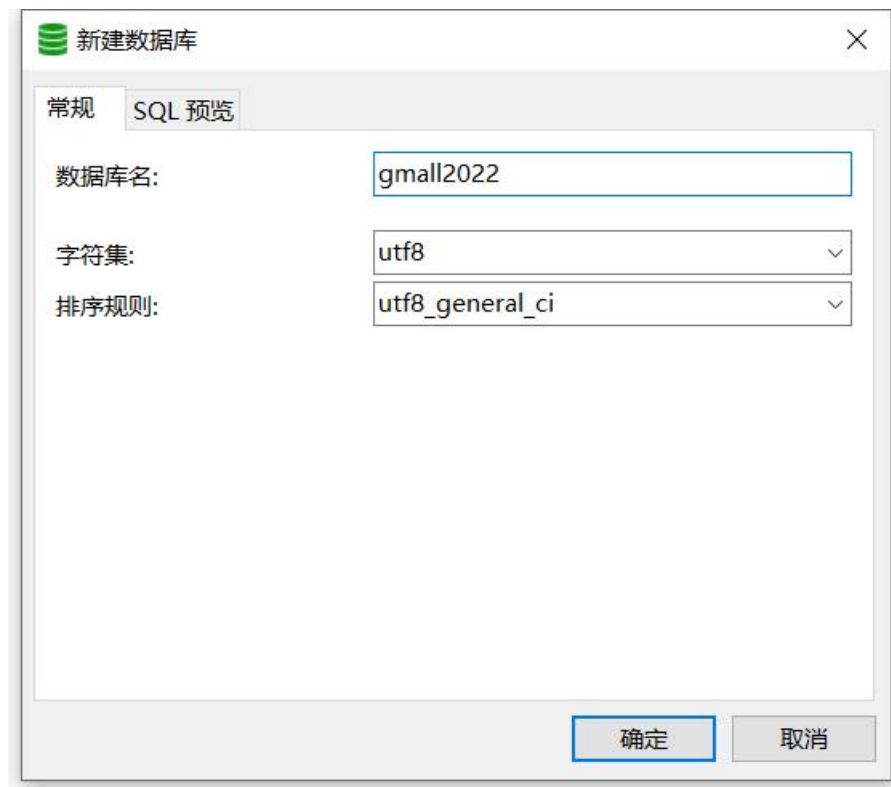
#### 增量表数据通道



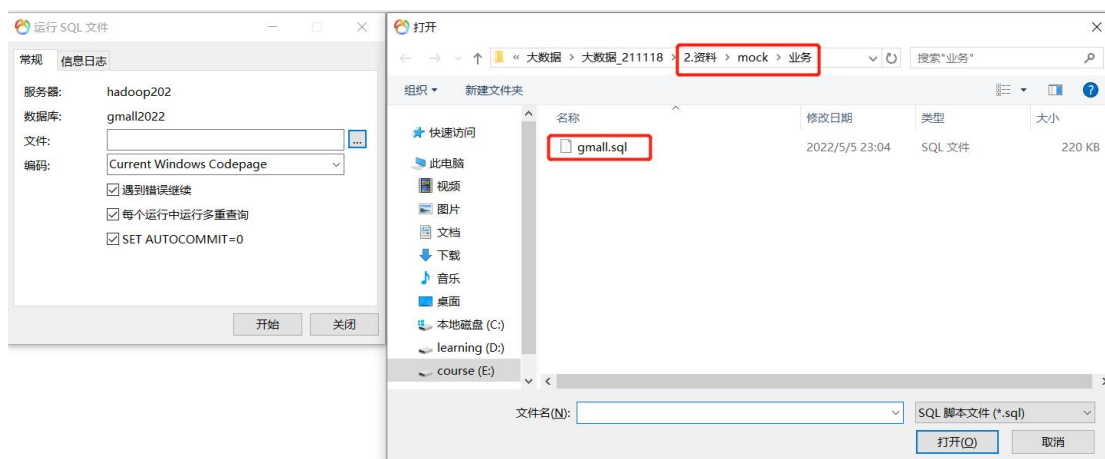
让天下没有难学的技术

## 3.2 MySQL 的准备

### 3.2.1 创建实时业务数据库



### 3.2.2 导入建表数据



### 3.2.3 修改/etc/my.cnf 文件

```
[atguigu@hadoop202 module]$ sudo vim /etc/my.cnf
```



```
server-id= 1
log-bin=mysql-bin
binlog_format=row
binlog-do-db=gmall2022
```

注意：binlog-do-db 根据自己的情况进行修改，指定具体要同步的数据库

## 3.2.4 重启 MySQL 使配置生效

```
sudo systemctl restart mysqld
```

到/var/lib/mysql 目录下查看初始文件大小 154

```
[atguigu@hadoop202 ~]$ sudo ls -l mysql
总用量 188516
-rw-r-----. 1 mysql mysql      56 6月 18 12:45 auto.cnf
-rw-r-----. 1 mysql mysql    1675 6月 18 12:45 ca-key.pem
-rw-r--r--. 1 mysql mysql    1074 6月 18 12:45 ca.pem
-rw-r--r--. 1 mysql mysql    1078 6月 18 12:45 client-cert.pem
-rw-r-----. 1 mysql mysql    1679 6月 18 12:45 client-key.pem
drwxr-x---. 2 mysql mysql    4096 6月 18 14:14 gmall
drwxr-x---. 2 mysql mysql    4096 8月 19 13:03 gmall2020
-rw-r-----. 1 mysql mysql     1033 8月 19 13:34 ib_buffer_pool
-rw-r-----. 1 mysql mysql 79691776 8月 19 13:34 ibdata1
-rw-r-----. 1 mysql mysql 50331648 8月 19 13:34 ib_logfile0
-rw-r-----. 1 mysql mysql 50331648 6月 18 12:45 ib_logfile1
-rw-r-----. 1 mysql mysql 12582912 8月 19 13:34 ibtmp1
drwxr-x---. 2 mysql mysql     8192 6月 18 23:49 metastore
drwxr-x---. 2 mysql mysql     4096 6月 18 12:45 mysql
-rw-r-----. 1 mysql mysql      154 8月 19 13:34 mysql-bin.000001
-rw-r-----. 1 mysql mysql       19 8月 19 13:34 mysql-bin.index
srwxrwxrwx. 1 mysql mysql        0 8月 19 13:34 mysql.sock
-rw-r-----. 1 mysql mysql        6 8月 19 13:34 mysql.sock.lock
drwxr-x---. 2 mysql mysql     8192 6月 18 12:45 performance_schema
-rw-r-----. 1 mysql mysql    1679 6月 18 12:45 private_key.pem
-rw-r--r--. 1 mysql mysql     451 6月 18 12:45 public_key.pem
-rw-r--r--. 1 mysql mysql    1078 6月 18 12:45 server-cert.pem
-rw-r-----. 1 mysql mysql    1679 6月 18 12:45 server-key.pem
drwxr-x---. 2 mysql mysql     8192 6月 18 12:45 sys
drwxr-x---. 2 mysql mysql      52 7月 2 17:52 test
```

## 3.2.5 模拟生成数据

(1) 把将\2.资料\mock\业务\下的 jar 和 properties 文件上传到 hadoop202 的

/opt/module/dblog 目录下

(2) 修改 application.properties 中数据库连接信息

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://hadoop202:3306/gmall2022?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT%2B8
spring.datasource.username=root
spring.datasource.password=123456

logging.pattern.console=%m%n
```

注意：如果生成较慢，可根据配置情况适当调整配置项

➤ 运行 jar 包

```
[atguigu@hadoop202 dblog]$ java -jar gmall2020-mock-db-2021-11-14.jar
```

- 再次到到/var/lib/mysql 目录下，查看 index 文件的大小

```
-rw-r----- 1 mysql mysql      177 4月  11 2021 mysql-bin.000001
-rw-r----- 1 mysql mysql      642 4月  11 2021 mysql-bin.000002
-rw-r----- 1 mysql mysql      154 5月   7 23:59 mysql-bin.000003
-rw-r----- 1 mysql mysql      154 5月   8 00:07 mysql-bin.000004
-rw-r----- 1 mysql mysql      177 5月   8 00:07 mysql-bin.000005
-rw-r----- 1 mysql mysql      154 5月   8 09:45 mysql-bin.000006
-rw-r----- 1 mysql mysql      154 5月   8 19:12 mysql-bin.000007
-rw-r----- 1 mysql mysql      177 5月   9 00:28 mysql-bin.000008
-rw-r----- 1 mysql mysql    595679 5月   9 00:36 mysql-bin.000009
```

### 3.3 安装 Maxwell

- 将/2.资料/工具下的 maxwell-1.25.0.tar.gz 上传到/opt/software 目录下
- 解压 maxwell-1.25.0.tar.gz 到/opt/module 目录

```
[atguigu@hadoop202 module]$ tar -zxvf /opt/software/maxwell-1.25.0.tar.gz -C /opt/module/
```

### 3.4 初始化 Maxwell 元数据库

- 在 MySQL 中建立一个 maxwell 库用于存储 Maxwell 的元数据

```
[atguigu@hadoop202 module]$ mysql -uroot -p123456
mysql> CREATE DATABASE maxwell ;
```

- 设置安全级别

```
mysql> set global validate_password_length=4;
mysql> set global validate_password_policy=0;
```

- 分配一个账号可以操作该数据库

```
mysql> GRANT ALL ON maxwell.* TO 'maxwell'@'%' IDENTIFIED BY '123456';
```

- 分配这个账号可以监控其他数据库的权限

```
mysql> GRANT SELECT ,REPLICATION SLAVE , REPLICATION CLIENT ON *.* TO
maxwell@'%';
```

### 3.5 使用 Maxwell 监控抓取 MySQL 数据

- 拷贝配置文件

```
[atguigu@hadoop202 maxwell-1.25.0]$ cp config.properties.example
config.properties
```

- 修改配置文件

```

producer=kafka
kafka.bootstrap.servers=hadoop202:9092,hadoop203:9092,hadoop204:9092
#需要添加
kafka_topic=topic_db
# mysql login info
host=hadoop202
user=maxwell
password=123456
#需要添加 后续初始化会用
client_id=maxwell_1

```

注意：默认还是输出到指定 Kafka 主题的一个 kafka 分区，因为多个分区并行可能会打乱

binlog 的顺序

如果要提高并行度，首先设置 kafka 的分区数>1,然后设置 producer\_partition\_by 属性

可选值 producer\_partition\_by=database|table|primary\_key|random| column

➤ 在/home/atguigu/bin 目录下编写 maxwell.sh 启动脚本

```

[atguigu@hadoop202 maxwell-1.25.0]$ vim /home/atguigu/bin/maxwell.sh
/opt/module/maxwell-1.25.0/bin/maxwell --config /opt/module/maxwell-1.25.0/config.properties >/dev/null
2>&1 &

```

➤ 授予执行权限

```

[atguigu@hadoop202 maxwell-1.25.0]$ sudo chmod +x /home/atguigu/bin/maxwell.sh

```

➤ 运行启动程序

```

[atguigu@hadoop202 maxwell-1.25.0]$ maxwell.sh
[atguigu@hadoop202 maxwell-1.25.0]$ xcall.sh jps
----- hadoop202 -----
6320 Application
5252 Kafka
7896 Jps
4811 QuorumPeerMain
7835 Maxwell

```

➤ 启动 Kafka 消费客户端，观察结果

```

[atguigu@hadoop202 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server
hadoop202:9092 --topic topic_db

```

➤ 执行/opt/module/dblog 下的 jar 生成模拟数据

```

[atguigu@hadoop202 dblog]$ java -jar gmall2020-mock-db-2021-11-14.jar
{"database": "gmall2021", "table": "user_info", "type": "update", "ts": "1610504502", "xid": "118213", "commit": true, "data": {"id": 1, "login_name": "66axzspu60",
"nick_name": "茗茗1", "passwd": null, "name": "茗茗", "phone_num": "13717127677", "email": "66axzspu60@355.net", "head_img": null, "user_level": "2", "birthda
y": "1974-01-13", "gender": "F", "create_time": "2021-01-13 09:52:56", "operate_time": null, "status": null}, "old": {"nick_name": "茗茗"}}

```

注意：如果需要监控 DDL 变化，在启动 Maxwell 的时候添加参数-output\_ddl

## 3.6 同步维度历史数据

实时计算不考虑历史的事实数据，但要考虑历史维度数据。因此要对维度相关的业务表

做一次全量同步

(1) 与维度相关的业务表如下

```
activity_info
activity_rule
activity_sku
base_category1
base_category2
base_category3
base_province
base_region
base_trademark
coupon_info
coupon_range
financial_sku_cost
sku_info
spu_info
user_info
```

(2) 切换到 /home/atguigu/bin 目录，创建 mysql\_to\_kafka\_init.sh 文件

```
[atguigu@hadoop202 maxwell]$ cd ~/bin
[atguigu@hadoop202 bin]$ vim mysql_to_kafka_init.sh
```

(3) 在脚本中添加如下内容

```
#!/bin/bash

# 该脚本的作用是初始化所有的业务数据，只需执行一次

MAXWELL_HOME=/opt/module/maxwell-1.25.0

import_data() {
    $MAXWELL_HOME/bin/maxwell-bootstrap --database gmall2022 --table $1 --config $MAXWELL_HOME/config.properties
}

case $1 in
    "activity_info")
        import_data activity_info
        ;;
    "activity_rule")
        import_data activity_rule
        ;;
    "activity_sku")
        import_data activity_sku
        ;;
    "base_category1")
        import_data base_category1
        ;;
endcase
```

```

"base_category2")
    import_data base_category2
    ;;
"base_category3")
    import_data base_category3
    ;;
"base_province")
    import_data base_province
    ;;
"base_region")
    import_data base_region
    ;;
"base_trademark")
    import_data base_trademark
    ;;
"coupon_info")
    import_data coupon_info
    ;;
"coupon_range")
    import_data coupon_range
    ;;
"financial_sku_cost")
    import_data financial_sku_cost
    ;;
"sku_info")
    import_data sku_info
    ;;
"spu_info")
    import_data spu_info
    ;;
"user_info")
    import_data user_info
    ;;
"all")
    import_data activity_info
    import_data activity_rule
    import_data activity_sku
    import_data base_category1
    import_data base_category2
    import_data base_category3
    import_data base_province
    import_data base_region
    import_data base_trademark
    import_data coupon_info
    import_data coupon_range
    import_data financial_sku_cost
    import_data sku_info
    import_data spu_info
    import_data user_info
    ;;
esac

```

#### (4) 增加执行权限

```
[atguigu@hadoop202 bin]$ chmod +x mysql_to_kafka_init.sh
```

#### (5) 执行脚本(注意：需要启动 maxwell 本身进程)

```
[atguigu@hadoop202 bin]$ mysql_to_kafka_init.sh all
```

(6) 启动 Kafka 消费者，观察数据是否写入 Kafka

```
[atguigu@hadoop202 bin]$ kafka-console-consumer.sh  
--bootstrap-server hadoop202:9092 --topic topic_db
```