用户画像课堂笔记

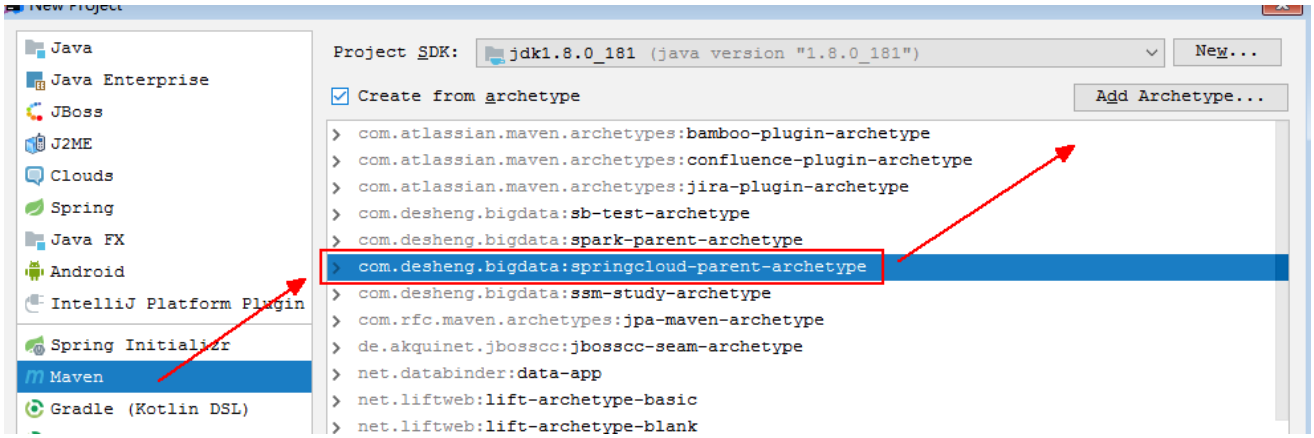
前后端分离

前端程序参见personas-web-readme.pdf笔记

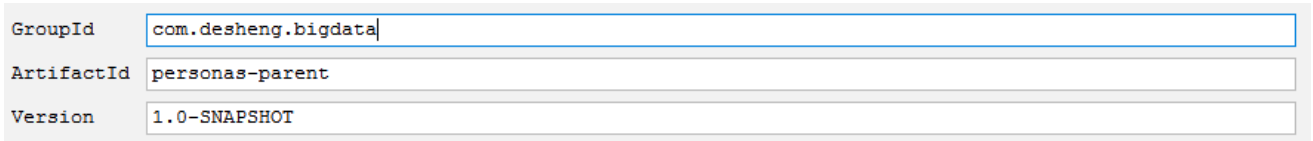
1. 后端项目

1.1. 基础项目构建

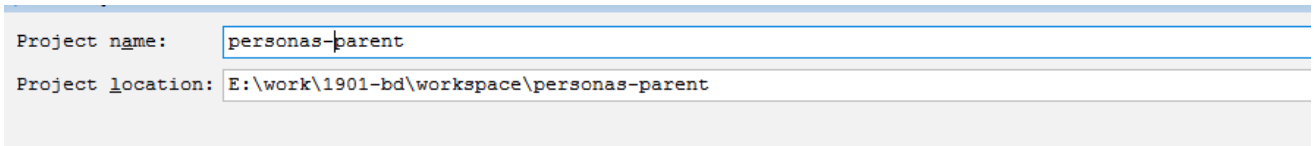
基于之前构建的maven的archetype原型进行项目构建。



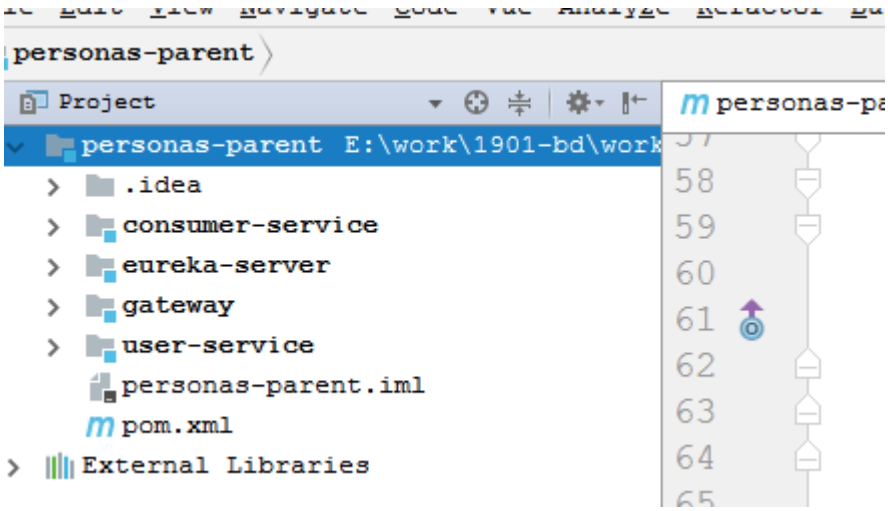
指定maven坐标



存储位置



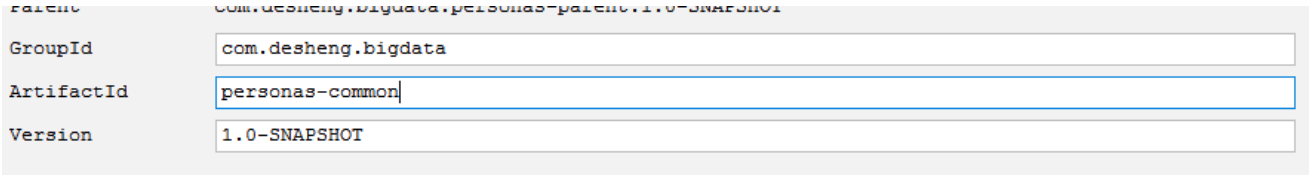
创建好的项目原型



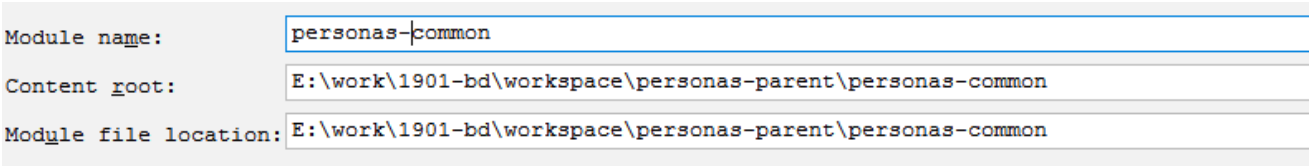
1.2. 新增基础模块

1.2.1. 通用的common模块

坐标

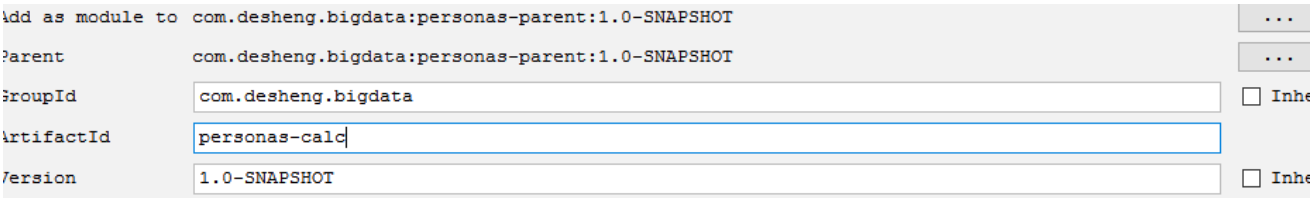


存储位置

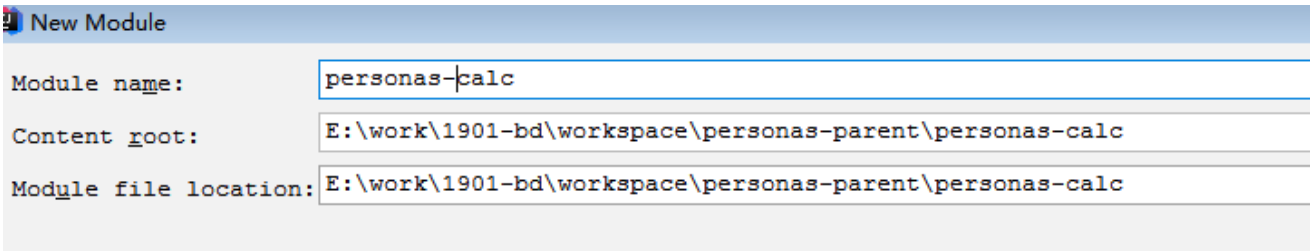


1.2.2. 大数据计算模块

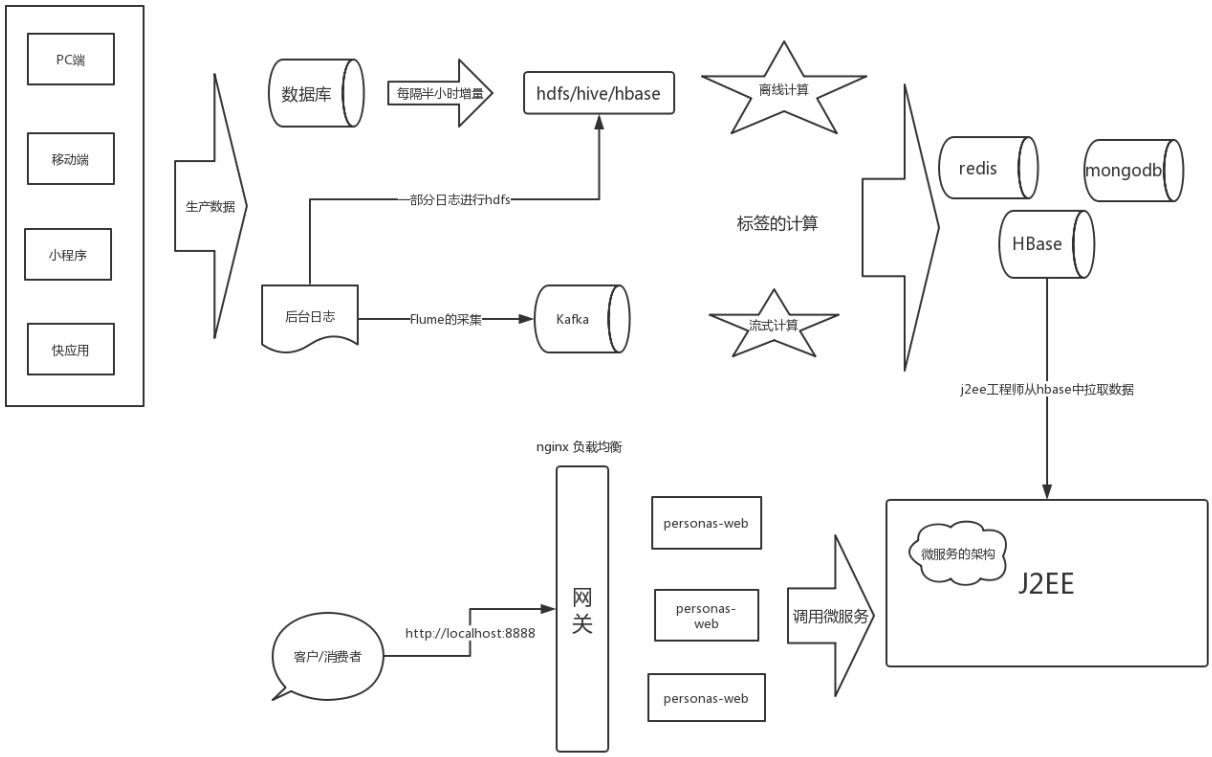
坐标



存储位置



2. 数据流程架构



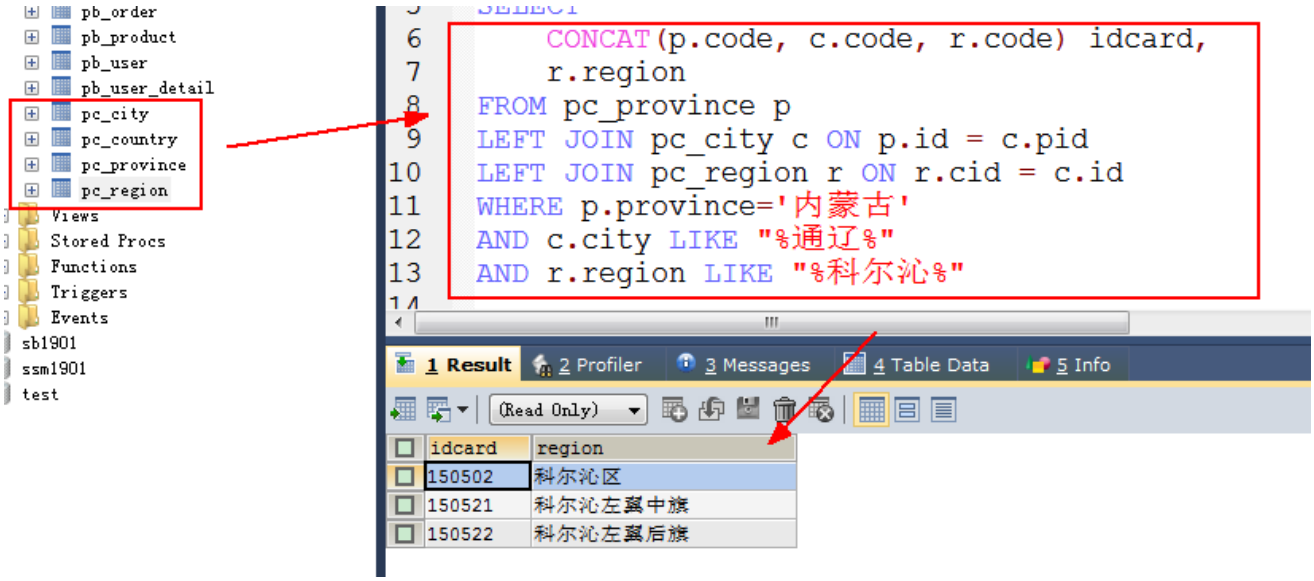
3. 基础数据处理

3.1. 用户交易及基础信息

在数据库中personas.sql即可，通过在root用户下给bigdata用户开发访问权限。

GRANT ALL PRIVILEGES ON `personas`.* TO 'bigdata'@'%' IDENTIFIED BY 'sorry'。

```
### 3.1.1. 关于地理信息表的说明:
```



3.1.2. 用户表信息

用户基础信息表

```
CREATE TABLE `pb_user` (
  `userid` int(11) NOT NULL AUTO_INCREMENT COMMENT '用户id',
  `username` varchar(50) DEFAULT NULL COMMENT '用户名',
  `password` varchar(50) DEFAULT NULL COMMENT '密码',
  `gender` tinyint(1) DEFAULT NULL COMMENT '性别',
  `telephone` varchar(11) DEFAULT NULL COMMENT '手机号',
  `email` varchar(50) DEFAULT NULL COMMENT 'email地址',
  `birthday` date DEFAULT NULL COMMENT '生日',
  `idcard` varchar(50) DEFAULT NULL COMMENT '身份证号',
  `register_time` datetime DEFAULT NULL COMMENT '注册时间',
  `province` varchar(20) DEFAULT NULL COMMENT '收货地址之省份',
  `city` varchar(20) DEFAULT NULL COMMENT '说活地址之城市',
  `client_type` int(1) DEFAULT NULL COMMENT '终端类型：0，pc端；1，移动端；2，小程序；3，快应
用',
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB AUTO_INCREMENT=1010002 DEFAULT CHARSET=utf8
```

用户详细信息表

```
CREATE TABLE `pb_user_detail` (
  `detail_id` int(11) NOT NULL AUTO_INCREMENT COMMENT '详情补充表的id',
  `userid` int(11) DEFAULT NULL COMMENT '用户id',
  `education` int(1) DEFAULT NULL COMMENT '学历：0未知，1初中及其以下，2高中，3中专，4大专，5本科，6研究生',
  `profession` varchar(20) DEFAULT NULL COMMENT '职业',
  `marriage` int(1) DEFAULT NULL COMMENT '婚姻状况：0未婚，1已婚，2离异',
  `has_children` int(11) DEFAULT NULL COMMENT '是否有小孩，0无小孩，1有小孩',
  `has_car` int(11) DEFAULT NULL COMMENT '是否有车：0无车，1有车',
  `has_house` int(11) DEFAULT NULL COMMENT '是否有房：0无房，1有房',
  `phone_brand` varchar(20) NOT NULL COMMENT '手机品牌',
  PRIMARY KEY (`detail_id`)
) ENGINE=InnoDB AUTO_INCREMENT=11010002 DEFAULT CHARSET=utf8
```

3.1.3 商品订单表

- 商品类别表

```
CREATE TABLE `pb_category` (
  `id` int(20) NOT NULL AUTO_INCREMENT COMMENT '商品类别id',
  `name` varchar(50) DEFAULT NULL COMMENT '类别名称',
  `description` varchar(1000) DEFAULT NULL COMMENT '描述',
  `level` int(11) DEFAULT NULL COMMENT '类别级别',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1183 DEFAULT CHARSET=utf8
```

- 商品表

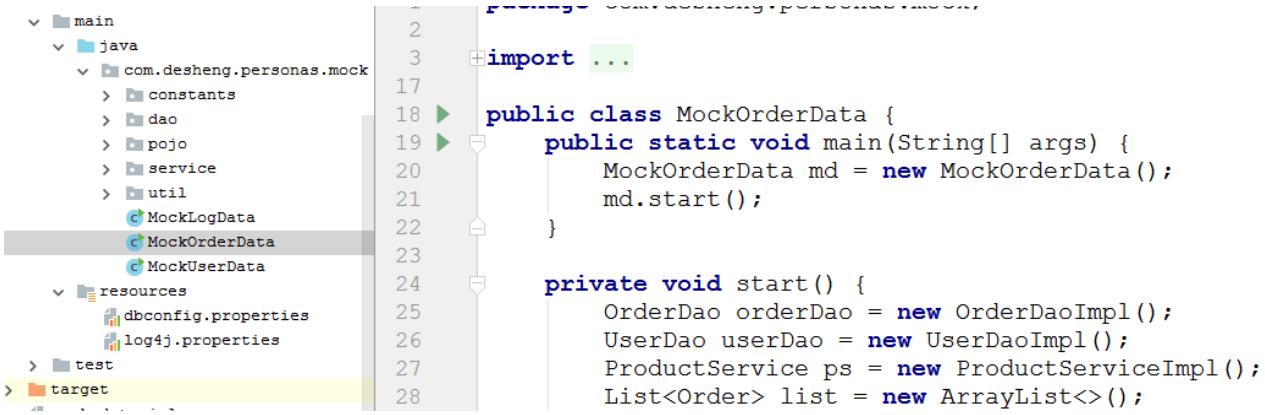
```
CREATE TABLE `pb_product` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '商品id',
  `cid` int(20) DEFAULT NULL COMMENT '商品类别id',
  `mechart_id` int(20) DEFAULT NULL COMMENT '商家id',
  `name` varchar(200) DEFAULT NULL COMMENT '商品名称',
  `description` varchar(1000) DEFAULT NULL COMMENT '商品描述',
  `price` float DEFAULT NULL COMMENT '商品价格',
  `num` int(20) DEFAULT NULL COMMENT '库存',
  `pic_url` varchar(500) DEFAULT NULL COMMENT '图片地址',
  `brand` varchar(50) DEFAULT NULL COMMENT '品牌',
  `create_time` timestamp NULL DEFAULT NULL COMMENT '创建时间',
  `update_time` timestamp NULL DEFAULT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`),
  KEY `category_id_fk` (`cid`)
) ENGINE=InnoDB AUTO_INCREMENT=143771131488370 DEFAULT CHARSET=utf8
```

- 商品订单表

```
CREATE TABLE `pb_order` (
  `id` int(20) NOT NULL AUTO_INCREMENT COMMENT '订单id',
  `uid` int(20) DEFAULT NULL COMMENT '用户id',
  `pid` int(20) DEFAULT NULL COMMENT '商品id',
  `cid` int(20) DEFAULT NULL COMMENT '类别id',
```

```
  `create_time` timestamp NULL DEFAULT NULL COMMENT '创建时间',
  `pay_time` timestamp NULL DEFAULT NULL COMMENT '支付时间',
  `pay_type` int(2) DEFAULT NULL COMMENT '支付类型 0网银，1支付宝，2微信，3信用卡',
  `pay_status` int(2) DEFAULT NULL COMMENT '支付状态 0 未支付，1 已支付，2 支付成功，3 支付失
败，4 退款中，5 退款成功',
  `amount` double DEFAULT NULL COMMENT '支付金额',
  `coupon_amount` double DEFAULT NULL COMMENT '优惠券金额',
  `total_amount` double DEFAULT NULL COMMENT '总金额',
  `refund_amount` double DEFAULT NULL COMMENT '退款金额',
  `num` int(11) DEFAULT NULL COMMENT '商品数量',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=300001 DEFAULT CHARSET=utf8
```

3.1.4. 模拟数据生成程序说明



3.2. 用户基础数据和交易数据提取

3.2.1. 说明

用户的基础数据和交易数据分别存储在pb_user/pb_user_detail表中，交易数据在pb_order中，为了方面后期的统计，需要将这些数据拉取到hdfs中进行存储。同时这些数据也在不间断的产生，所以我们的拉取思路的操作每隔半小时做一次增量拉取(更新)。后期基于这个思路来执行的离线的统计。

拉取数据，因为是从rdbms拉取到hdfs的操作，可以使用sqoop操作来完成，还有人选择使用sql脚本来拉取数据。

3.2.2. 使用sqoop拉取数据

- 设计存储数据的目录

  hdfs://ns1/orginal/personas/user/yyyy/MM/dd/hhmm

  hdfs://ns1/orginal/personas/user-detail/yyyy/MM/dd/hhmm

  hdfs://ns1/orginal/personas/order/yyyy/MM/dd/hhmm

  创建通用父目录

  ```
  hdfs dfs -mkdir -p hdfs://ns1/orginal/personas/user/
  hdfs dfs -mkdir -p hdfs://ns1/orginal/personas/user-detail/
  hdfs dfs -mkdir -p hdfs://ns1/orginal/personas/order/
  ```

- 拉取数据的编程
  - 使用sqoop拉取pb-user数据

1. 基于shell获取格式化日期

```
[bigdata@bigdata01 sqoop]$ date -d"yesterday" +%Y/%m/%d/%H%M
2019/07/11/0132
[bigdata@bigdata01 sqoop]$ date -d"tomrrow" +%Y/%m/%d/%H%M
date: invalid date `tomrrow'
You have new mail in /var/spool/mail/bigdata
[bigdata@bigdata01 sqoop]$ date -d"tomorrow" +%Y/%m/%d/%H%M
2019/07/13/0133
[bigdata@bigdata01 sqoop]$ date -d"the day after tomorrow" +%Y/%m/%d/%H%M
date: invalid date `the day after tomorrow'
[bigdata@bigdata01 sqoop]$ date -d"day after tomorrow" +%Y/%m/%d/%H%M
date: invalid date `day after tomorrow'
[bigdata@bigdata01 sqoop]$ date -d"1 day ago" +%Y/%m/%d/%H%M
2019/07/11/0134
You have new mail in /var/spool/mail/bigdata
[bigdata@bigdata01 sqoop]$ date -d"2 day ago" +%Y/%m/%d/%H%M
2019/07/10/0134
[bigdata@bigdata01 sqoop]$ date -d"100 day ago" +%Y/%m/%d/%H%M
2019/04/03/0134
[bigdata@bigdata01 sqoop]$ date -d"-100 day ago" +%Y/%m/%d/%H%M
2019/10/20/0134
[bigdata@bigdata01 sqoop]$ date -d"-100 day ago" +%Y/%m/%d/%H%M
```

基于linux脚本来获取格式化指定的日期

2. 编写脚本正常执行问题

编写的脚本是在windows中执行的，通过set ff命令查询，发现文件格式为dos，这个文件格式在linux中无法正常执行

```
fileformat=dos
```

正常的在linux的文件格式为unix

```
fileformat=unix
```

修改操作，在sh文件的底行编辑模式下面，操作 set ff=unix

```
:set ff=unix
```

3.脚本内容

```
#!/bin/sh

#################################################################
##
## push user info 2 hdfs
## created by old li
## 2019/07/12 16:20:50
## 佛祖保佑，永无bug~
```

```
#################################################################

## hdfs://ns1/orginal/personas/user/yyyy/MM/dd/hhmm

SQOOP_BIN=/home/bigdata/app/sqoop/bin

USER_HOME_PREFIX=hdfs://ns1/orginal/personas/user/

INPUT_DATE_DIR=`date -d"0 day ago" +%Y/%m/%d/%H%M`

## 半小时前的时间
START_TIME=`date -d"30 minute ago" +"%Y-%m-%d %H:%M:%S"`
## 当前时间
END_TIME=`date -d"0 day ago" +"%Y-%m-%d %H:%M:%S"`


## import
QUERY_SQL="
     SELECT
       u.*
     FROM pb_user u
     WHERE u.register_time >= '2019-05-29 11:38:32'
     AND u.register_time < '${END_TIME}'
     AND \$CONDITIONS
    "

echo "QUERY_SQL":$QUERY_SQL

${SQOOP_BIN}/sqoop import \
--connect "jdbc:mysql://192.168.43.1:3306/personas?
useUnicode=true&characterEncoding=UTF-8" \
--username bigdata \
--password sorry \
--delete-target-dir \
--fields-terminated-by '\001' \
--target-dir ${USER_HOME_PREFIX}/${INPUT_DATE_DIR} \
--query "${QUERY_SQL}" \
--split-by "userid"
```

- 拉取pb_user_detail数据

```
#!/bin/sh

#################################################################
##
## push user detail info 2 hdfs
## created by old li
## 2019/07/12 16:20:50
## 佛祖保佑，永无bug~
#################################################################

## hdfs://ns1/orginal/personas/user/yyyy/MM/dd/hhmm
```

```
SQOOP_BIN=/home/bigdata/app/sqoop/bin

USER_HOME_PREFIX=hdfs://ns1/orginal/personas/user-detail

INPUT_DATE_DIR=`date -d"0 day ago" +%Y/%m/%d/%H%M`

## 半小时前的时间
START_TIME=`date -d"30 minute ago" +"%Y-%m-%d %H:%M:%S"`
## 当前时间
END_TIME=`date -d"0 day ago" +"%Y-%m-%d %H:%M:%S"`


## import
QUERY_SQL="
     SELECT
       d.*
     FROM pb_user_detail d
     left join pb_user u on d.userid = u.userid
     WHERE u.register_time >= '2019-05-29 11:38:32'
     AND u.register_time < '${END_TIME}'
     AND \$CONDITIONS
    "

echo "QUERY_SQL":$QUERY_SQL

${SQOOP_BIN}/sqoop import \
--connect "jdbc:mysql://192.168.43.1:3306/personas?
useUnicode=true&characterEncoding=UTF-8" \
--username bigdata \
--password sorry \
--delete-target-dir \
--fields-terminated-by '\001' \
--target-dir ${USER_HOME_PREFIX}/${INPUT_DATE_DIR} \
--query "${QUERY_SQL}" \
--split-by "detail_id"
```

- 拉取pb_order数据

```
#!/bin/sh

#################################################################
##
## push order info 2 hdfs
## created by old li
## 2019/07/12 16:20:50
## 佛祖保佑，永无bug~
#################################################################

## hdfs://ns1/orginal/personas/user/yyyy/MM/dd/hhmm

SQOOP_BIN=/home/bigdata/app/sqoop/bin

ORDER_HOME_PREFIX=hdfs://ns1/orginal/personas/order/
```

```
INPUT_DATE_DIR=`date -d"0 day ago" +%Y/%m/%d/%H%M`
## 半小时前的时间
START_TIME=`date -d"30 minute ago" +"%Y-%m-%d %H:%M:%S"`
## 当前时间
END_TIME=`date -d"0 day ago" +"%Y-%m-%d %H:%M:%S"`


## import
QUERY_SQL="
     SELECT
       o.*
     FROM pb_order o
     WHERE o.create_time >= '2019-05-29 11:38:32'
     AND o.create_time < '${END_TIME}'
     AND \$CONDITIONS
    "

echo "QUERY_SQL":$QUERY_SQL

${SQOOP_BIN}/sqoop import \
--connect "jdbc:mysql://192.168.43.1:3306/personas?
useUnicode=true&characterEncoding=UTF-8" \
--username bigdata \
--password sorry \
--fields-terminated-by '\001' \
--target-dir ${ORDER_HOME_PREFIX}/${INPUT_DATE_DIR} \
--query "${QUERY_SQL}" \
--split-by "id" \
-m 1 \
--append
```

3.3. 开始计算基础标签

3.3.0 分析和模块构建

3.3.0.1. 分析

年代标签，也就是50后，60后，70后，80后，90后，00后，10后。这些相对动态的标签是需要用户的birthday计算出来的，birthday在pb_user表中。最简单的做法就是获取birthday中的年代并提取对应的十年就行了。

手机运营商标签和邮件运营商也都需要基于pb_user表中的数据。根据手机号计算出是移动、联通还是电信；根据邮箱计算出是网易、还是新浪等等主流邮箱即可。

所以，统计是分为两方面的，一方面是单一个体的标签，另一方面是汇总的群体数据。

剁手指数，需要从oder中获取。

品牌偏好从收藏数据中获取。

3.3.0.2. 模块构建

personas-calc的maven依赖

```
<properties>
    <scala.version>2.11</scala.version>
    <spark.version>2.2.2</spark.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_${scala.version}</artifactId>
        <version>${spark.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming_${scala.version}</artifactId>
        <version>${spark.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming-kafka-0-10_${scala.version}</artifactId>
        <version>${spark.version}</version>
    </dependency>
```

3.3.1. 用户年代标、 邮件运营商、手机运营商标签

- 主要的计算job

```
package com.desheng.bigdata.personas.job

import com.desheng.bigdata.personas.common.Constants
import com.desheng.bigdata.personas.common.db.{HBaseUtils, JedisUtils}
import com.desheng.bigdata.personas.common.device.{DeviceUtils, EmailUtils}
import com.desheng.bigdata.personas.entity.User
import org.apache.hadoop.hbase.TableName
import org.apache.hadoop.hbase.client.Put
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}

import scala.collection.mutable.ArrayBuffer

/**
  * 用户的基础标签统计作业
  *     用户年代
  *     手机运营商
  *     邮箱运营商
  *
  *  spark-core的操作
  */
object UserBaseTagJob {
    def main(args: Array[String]): Unit = {
        if(args == null || args.length < 1) {
            System.err.println(
```

```
                """
                  |Parameter Errors ! Usage: <inputpath>
                """.stripMargin)
            System.exit(-1)
        }

        val Array(inputpath) = args

        val conf = new SparkConf()
                .setMaster("local[*]")
                .setAppName("UserBaseTagJob")
        val sc = new SparkContext(conf)

        //加载外部数据
        val lines = sc.textFile(inputpath)

        val users:RDD[User] = lines.map(line => User.line2User(line)).filter(user =>
user.userid != -1L)

        //基于转换之后的user数据进行统计
        /*val userTagPairs = users.flatMap(user => {

            val ab = ArrayBuffer[(String, Int)]()
            //处理用户年代
            val yearAge = calcYearAge(user.birthday)
            //处理手机号 -->运营商
            val telOperator = getTelOperator(user.telephone)
            //处理邮箱
            val emailOperator = getEmailOperator(user.email)


            ab.append((yearAge, 1))
            ab.append((telOperator, 1))
            ab.append((emailOperator, 1))

            ab
        }).reduceByKey(_+_)*/

        val userTagPairs = users.mapPartitions(partition => {
            val ab = ArrayBuffer[(String, Int)]()
            val connection = HBaseUtils.getHBaseConnection
            val table =
connection.getTable(TableName.valueOf(Constants.TABLE_USER_BASE_TAG))

            partition.foreach(user => {
                //处理用户年代
                val yearAge = calcYearAge(user.birthday)
                //处理手机号 -->运营商
                val telOperator = getTelOperator(user.telephone)
                //处理邮箱
                val emailOperator = getEmailOperator(user.email)
                val rowkey = user.userid.toString.reverse.getBytes()
                val put = new Put(rowkey)//使用useride作为行键
```

```
                put.addColumn(Constants.CF_BASE_USER, Constants.COL_YEAR_AGE,
yearAge.getBytes())
                put.addColumn(Constants.CF_BASE_USER, Constants.COL_TEL_OP,
telOperator.getBytes())
                put.addColumn(Constants.CF_BASE_USER, Constants.COL_EMAIL_OP,
emailOperator.getBytes())

                table.put(put)
                ab.append((yearAge, 1))
                ab.append((telOperator, 1))
                ab.append((emailOperator, 1))
            })
            table.close()
            HBaseUtils.release(connection)

            ab.iterator
        }).reduceByKey(_+_)

        /**
          *  用户个体的标签数据
          *     写入hbase
          *  汇总的数据
          *     增量的更新---使用redis/mongodb一次搞定 incrBy
          */
        userTagPairs.foreachPartition(partition => {
            val jedis = JedisUtils.getJedis
            partition.foreach{case (key, count) => {
                val prefix = key.substring(0, key.indexOf("_"))
                val tag = key.substring(key.indexOf("_") + 1)
                prefix match {
                    case Constants.PREFIX_YEAR_AGE => {
                        jedis.hincrBy(Constants.TABLE_USER_SUMMARY_YEAR, tag, count)
                    }
                    case Constants.PREFIX_TEL => {
                        jedis.hincrBy(Constants.TABLE_USER_SUMMARY_TEL, tag, count)
                    }
                    case Constants.PREFIX_EMAIL => {
                        jedis.hincrBy(Constants.TABLE_USER_SUMMARY_EMAIL, tag, count)
                    }
                }
            }}
            JedisUtils.release(jedis)
        })
        sc.stop()
    }

    //yyyy-MM-dd
    def calcYearAge(birthday: String): String = {
        Constants.PREFIX_YEAR_AGE + "_" + birthday.charAt(2) + "0s"
    }

    def getTelOperator(tel: String): String = {
```

```
        s"${Constants.PREFIX_TEL}_${DeviceUtils.getOperatorByTel(tel)}"
    }
    def getEmailOperator(email: String): String = {
        s"${Constants.PREFIX_EMAIL}_${EmailUtils.getOperatorByEmail(email)}"
    }
}
```

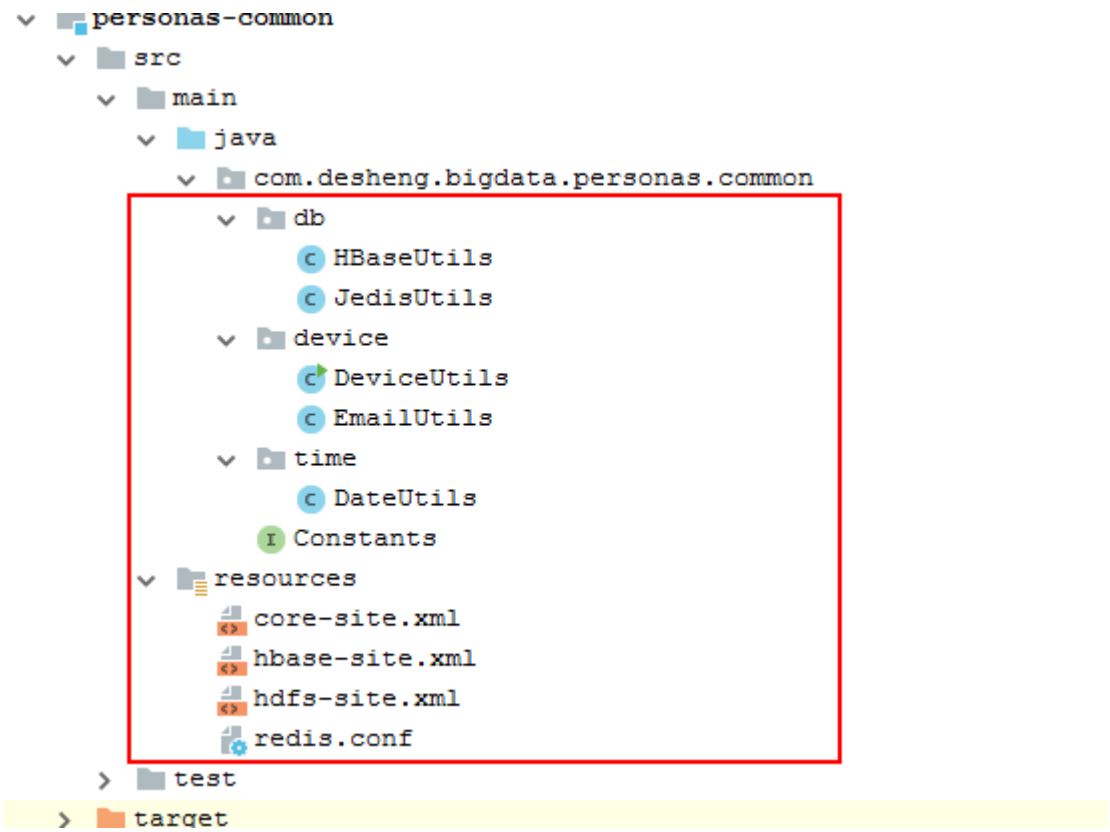
- case class User

```
case class User(
   userid:Long,
   username:String,
   password:String,
   gender:Short,
   telephone:String,
   email:String,
   birthday:String,
   idcard:String,
   registerTime: Date,
   province:String,
   city:String,
   clientType:Int) extends Serializable

object User {

    def line2User(line:String):User = {
        val fields = line.split(Constants.FILE_FIELD_SEPARATOR)
        if(fields == null || fields.length != 12) {
            User(-1, null, null, 0.toShort, null, null, null, null, null, null, null,
0)
        } else {
            val userid = fields(Constants.INDEX_USER_ID).toLong
            val username = fields(Constants.INDEX_USER_NAME)
            val password = fields(Constants.INDEX_USER_PASSWORD)
            val gender = if(fields(Constants.INDEX_USER_GENDER).toBoolean) 1.toShort
else 0.toShort
            val telephone = fields(Constants.INDEX_USER_TELEPHONE)
            val email = fields(Constants.INDEX_USER_EMAIL)
            val birthday = fields(Constants.INDEX_USER_BIRTHDAY)
            val idcard = fields(Constants.INDEX_USER_ID_CARD)
            val registerTime =
DateUtils.parseTime(fields(Constants.INDEX_USER_REGISTER_TIME))
            val province = fields(Constants.INDEX_USER_PROVINCE)
            val city = fields(Constants.INDEX_USER_CITY)
            val clientType = fields(Constants.INDEX_USER_CLIENT_TYPE).toInt
            User(userid, username, password, gender, telephone, email, birthday,
idcard, registerTime, province, city, clientType)
        }
    }
}
```

- 工具类



3.3.2. 剁手指数标签

3.3.2.0. 分析

剁手指数，在给定的标准之下用户购买金额、购买频率、购买客单价的综合汇总的一个结果。

剁手指数计算公式=支付金额平均值 * 0.3 + 支付金额最大值 * 0.3 + 下单频率 * 0.4

这里要进行统计的是基于用户过往1年的数据。

显然是一个离线计算。pb_order表。

3.3.2.1. 编码

```
package com.desheng.bigdata.personas.job

import com.desheng.bigdata.personas.common.Constants
import com.desheng.bigdata.personas.common.db.HBaseUtils
import com.desheng.bigdata.personas.common.number.NumberUtils
import com.desheng.bigdata.personas.entity.Order
import org.apache.hadoop.hbase.TableName
import org.apache.hadoop.hbase.client.Put
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{DataFrame, Dataset, SparkSession}
import org.apache.spark.{SparkConf, SparkContext}
/**
  * 剁手指数统计
  * 需要加载的pb_order表中的数据
  * 将结果录入到hbase中的user表
  */
```

```scala
object ChopperIndexJob {
    def main(args: Array[String]): Unit = {
        if(args == null || args.length < 1) {
            System.err.println(
                """
                  |Parameter Errors ! Usage: <inputpath>
                """.stripMargin)
            System.exit(-1)
        }

        val Array(inputpath) = args

        val conf = new SparkConf()
            .setMaster("local[*]")
            .setAppName("ChopperIndexJob")
        val spark = SparkSession.builder()
                    .config(conf)
                    .getOrCreate()

        //加载外部数据
        import spark.implicits._
        val lines = spark.sparkContext.textFile(inputpath)
        val orderRDD:RDD[Order] = lines.map(line => Order.makeLine2Order(line))

        val orderDS:Dataset[Order] = orderRDD.toDS()

        orderDS.createOrReplaceTempView("pb_order")

        val sql =
            """
              |SELECT
              |   uid,
              |   ROUND(SUM(amount) / COUNT(1) / 10000, 2) kdj,
              |   ROUND(MAX(amount) / 10000, 2) zdje,
              |   ROUND(COUNT(1) / 52, 2) frequency
              |FROM pb_order
              |GROUP BY uid
            """.stripMargin
        val uid2Info:DataFrame = spark.sql(sql)
        //加权求取的剁手指数 剁手指数计算公式=支付金额平均值 * 0.3 + 支付金额最大值 * 0.3 + 下单频
率 * 0.4

        //将结果录入到hbase表中
        uid2Info.foreachPartition(rows => {
            val connection = HBaseUtils.getHBaseConnection
            val table =
connection.getTable(TableName.valueOf(Constants.TABLE_USER_BASE_TAG))
            for(row <- rows) {
                val rowkey = row.getAs[Long]("uid").toString.reverse.getBytes()

                val kdj = row.getAs[Double]("kdj")//客单价
                val zdje = row.getAs[Double]("zdje")//最大金额
                val frequency = row.getAs[Double]("frequency") // 剁手频率
```

```
                var chopperIndex = (0.3 * kdj + 0.3 * zdje + 0.4 * frequency) % 100
                chopperIndex = NumberUtils.formatDouble(chopperIndex, 2)
                val put = new Put(rowkey)

                put.addColumn(Constants.CF_BASE_USER, Constants.COL_CHOPPER_INDEX,
chopperIndex.toString.getBytes())
                table.put(put)
            }
            table.close()
            HBaseUtils.release(connection)
        })
        spark.stop()
    }
}
```

3.3.3. 品牌偏好

3.3.3.1. 品牌偏好分析

所谓品牌偏好就是说，用户更加喜欢，或者倾向于某些品牌。每一个用户都有喜欢、收藏、或者购买的品牌若干，所以就没有必要把每一个品牌都列举出来吧。这里只需要列举出其中前三个品牌即可。

来源数据，基于用户实时的行为数据，反映用户的品牌变迁，这里还可以通过订单，最后通过加权分配来计算出用户对每个品牌的偏好程度。这里为了计算，只使用前者数据。

浏览商品行为：商品id 商品类别id 浏览时间、停留时间、用户id 终端类别,用户ip 收藏商品行为：商品id 商品类别id 操作时间、操作类型（收藏，取消）、用户id、终端类别、用户ip 购物车行为：商品id 商品类别id 操作时间、操作类型（加入，删除）、用户id、终端类别、用户ip 关注商品:商品id 商品类别id 操作时间、操作类型（关注，取消）、用户id、终端类别、用户ip

3.3.3.2. 编码

处理Mock-Data模块。为了能够进行正常的打包，把程序中的personas-parent的信息去除，否则打包失败。

修改之后的pom文件为

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <!--<parent>
        <artifactId>personas-parent</artifactId>
        <groupId>com.desheng.bigdata</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>-->
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.desheng.bigdata</groupId>
    <artifactId>mock-data</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
```

```
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.17</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.32</version>
        </dependency>
        <dependency>
            <groupId>commons-dbutils</groupId>
            <artifactId>commons-dbutils</artifactId>
            <version>1.6</version>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.8</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <!-- 编译有外部依赖的插件-->
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                    <archive>
                        <!--<manifest>
                          <mainClass></mainClass>
                        </manifest>-->
                    </archive>
```

```
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

3.3.3.2.1. 数据的生成

使用MockLogData完成行为数据的生成。

```
#!/bin/sh

JAR_PATH=/home/bigdata/shells/personas/streaming/mock-data-1.0-SNAPSHOT-jar-with-
dependencies.jar
java -cp \
${JAR_PATH} \
com.desheng.personas.mock.MockLogData
```

```
drwxrwxr-x 2 bigdata bigdata    4096 Jul 15 01:19 logs
drwxrwxr-x 3 bigdata bigdata    4096 Jul 15 01:26 mock-data
-rw-rw-r-- 1 bigdata bigdata 6091635 Jul 15 01:25 mock-data-1.0-SNAPSHOT-jar-with-dependencies.jar
-rwxrw-r-- 1 bigdata bigdata     172 Jul 15 01:19 mock-streaming-data.sh
You have new mail in /var/spool/mail/bigdata
```

3.3.3.2.2. 数据的采集

使用flume+kafka的采集过程

flume和kafka的配置文件: 注意：多个source或者多个sink之间使用空格隔开，不是逗号

```
a1.sources = scan follow collect cart
a1.sinks = k1-scan k2-follow k3-collect k4-cart
a1.channels = c1

# 监听的是某个文件中的新增数据
a1.sources.scan.type = exec
a1.sources.scan.command = tail -F
/home/bigdata/shells/personas/streaming/logs/scan.http.log

a1.sources.follow.type = exec
a1.sources.follow.command = tail -F
/home/bigdata/shells/personas/streaming/logs/follow.http.log

a1.sources.collect.type = exec
a1.sources.collect.command = tail -F
/home/bigdata/shells/personas/streaming/logs/collect.http.log
```

```
a1.sources.cart.type = exec
a1.sources.cart.command = tail -F
/home/bigdata/shells/personas/streaming/logs/cart.http.log

# Describe the sink ---kafka
a1.sinks.k1-scan.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k1-scan.kafka.bootstrap.servers = bigdata01:9092,bigdata02:9092,bigdata03:9092
a1.sinks.k1-scan.kafka.topic = personas-scan-topic
a1.sinks.k1-scan.kafka.producer.acks = 1
a1.sinks.k1-scan.kafka.producer.linger.ms = 1

a1.sinks.k2-follow.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k2-follow.kafka.bootstrap.servers =
bigdata01:9092,bigdata02:9092,bigdata03:9092
a1.sinks.k2-follow.kafka.topic = personas-follow-topic
a1.sinks.k2-follow.kafka.producer.acks = 1
a1.sinks.k2-follow.kafka.producer.linger.ms = 1

a1.sinks.k3-collect.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k3-collect.kafka.bootstrap.servers =
bigdata01:9092,bigdata02:9092,bigdata03:9092
a1.sinks.k3-collect.kafka.topic = personas-collect-topic
a1.sinks.k3-collect.kafka.producer.acks = 1
a1.sinks.k3-collect.kafka.producer.linger.ms = 1

a1.sinks.k4-cart.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k4-cart.kafka.bootstrap.servers = bigdata01:9092,bigdata02:9092,bigdata03:9092
a1.sinks.k4-cart.kafka.topic = personas-cart-topic
a1.sinks.k4-cart.kafka.producer.acks = 1
a1.sinks.k4-cart.kafka.producer.linger.ms = 1


# 不建议使用memory，使用file
a1.channels.c1.type = memory
a1.channels.c1.capacity = 100000
a1.channels.c1.transactionCapacity = 100000

# Bind the source and sink to the channel
a1.sources.scan.channels = c1
a1.sources.cart.channels = c1
a1.sources.collect.channels = c1
a1.sources.follow.channels = c1

a1.sinks.k1-scan.channel = c1
a1.sinks.k2-follow.channel = c1
a1.sinks.k3-collect.channel = c1
a1.sinks.k4-cart.channel = c1
```

启动程序：

> nohup bin/flume-ng agent -n a1 -c conf -f conf/personas-1901.conf >/dev/null 2>&1 &

3.3.3.2.3. 数据的整合

使用SparkStreaming+kafka整合，这里使用新版本的方式进行整合。

```
val kafkaParams = Map[String, Object](
  "bootstrap.servers" -> "localhost:9092,anotherhost:9092",
  "key.deserializer" -> classOf[StringDeserializer],
  "value.deserializer" -> classOf[StringDeserializer],
  "group.id" -> "use_a_separate_group_id_for_each_stream",
  "auto.offset.reset" -> "latest",
  "enable.auto.commit" -> (false: java.lang.Boolean)
)

val topics = Array("topicA", "topicB")
val stream = KafkaUtils.createDirectStream[String, String](
  streamingContext,
  PreferConsistent,
  Subscribe[String, String](topics, kafkaParams)
)

stream.map(record => (record.key, record.value))
```

参看KafkaManager.scala

```
package com.desheng.bigdata.personas.util

import com.desheng.bigdata.personas.common.db.JedisUtils
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.TopicPartition
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.dstream.InputDStream
import org.apache.spark.streaming.kafka010.{ConsumerStrategies, KafkaUtils,
LocationStrategies, OffsetRange}

import scala.collection.mutable

object KafkaManager {

    /**
      */
    def storeOffsets(offsetRanges: Array[OffsetRange], group:String) = {
        val jedis = JedisUtils.getJedis
        for (offsetRange <- offsetRanges) {
            val topic = offsetRange.topic
            val partition = offsetRange.partition
            val offset = offsetRange.untilOffset
            val field = s"${group}|${partition}"
            jedis.hset(topic, field, offset.toString)
        }
        JedisUtils.release(jedis)
    }
```

```
    def createMessage(ssc: StreamingContext, kafkaParams: Map[String, Object],
                      topics:Set[String]): InputDStream[ConsumerRecord[String, String]]
= {
        //step 1 读取偏移量
        val fromOffsets:Map[TopicPartition, Long] = getFromOffsets(topics,
kafkaParams("group.id").toString)
        var messages:InputDStream[ConsumerRecord[String, String]] = null
        if(!fromOffsets.isEmpty) {
            messages = KafkaUtils.
                createDirectStream[String, String](ssc,
                LocationStrategies.PreferConsistent,
                ConsumerStrategies.Subscribe[String, String](topics, kafkaParams,
fromOffsets))

        } else {
            messages = KafkaUtils.
                createDirectStream[String, String](ssc,
                LocationStrategies.PreferConsistent,
                ConsumerStrategies.Subscribe[String, String](topics, kafkaParams))
        }
        messages
    }
    /*

     */
    def getFromOffsets(topics:Set[String], group:String):Map[TopicPartition, Long] = {
        val fromOffset = mutable.Map[TopicPartition, Long]()
        import scala.collection.JavaConversions._
        val jedis = JedisUtils.getJedis
        for(topic <- topics) {
            val map = jedis.hgetAll(topic)
            for((field, value) <- map) {//field=group|partition
                val partition = field.substring(field.indexOf("|") + 1).toInt
                val offset = value.toLong
                fromOffset.put(new TopicPartition(topic, partition), offset)
            }
        }
        JedisUtils.release(jedis)
        fromOffset.toMap
    }
}
```

注意: idea编码的问题--编码GBK的不可映射字符

解决方法，在对应的pom中添加如下配置

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```