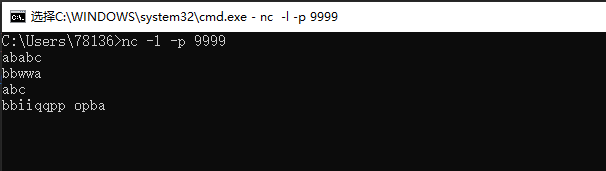
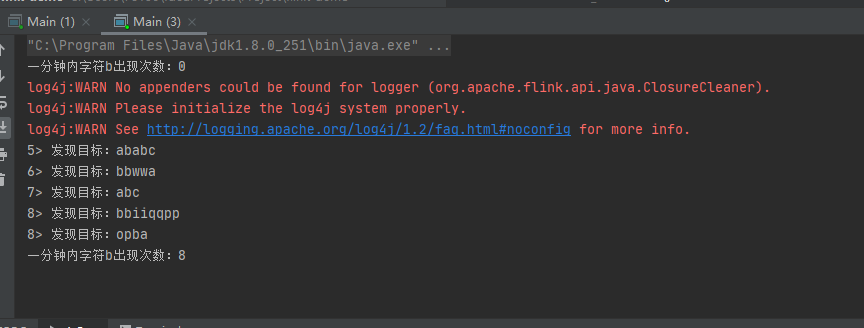


import java.util.{Timer, TimerTask}

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment  
import org.apache.flink.streaming.api.scala.\_  
  
object Main {  
 val *target*="b"  
 var *count* = 0  
  
 class time\_task(timer: Timer) extends TimerTask{  
 override def run(): Unit = {  
 System.*out*.println("一分钟内字符b出现次数："+*count*.toString)  
 *count* = 0;  
 }  
 }  
  
 def main(args: Array[String]) {  
 val s = new Timer()  
 s.schedule(new time\_task(s),0, 60000)  
 val env = StreamExecutionEnvironment.*getExecutionEnvironment* //Linux or Mac:nc -l 9999  
 //Windows:nc -l -p 9999  
 val text = env.socketTextStream("localhost", 9999)  
 val stream = text.flatMap {  
 \_.toLowerCase.split("\\W+") filter {  
 \_.contains(*target*)  
 }  
 }.map {x=>  
 var last = 0  
 while(x.indexOf("b", last) != -1){  
 last = x.indexOf("b", last)+1;  
 *count*+=1  
 }  
 ("发现目标："+x)  
 }  
  
 stream.print()  
 env.execute("Window Stream WordCount")  
 }  
}

设置一个Timer计时器，每过一分钟输出统计结果





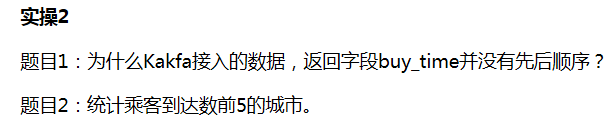
ababc:2次

bbwwa:2次

abc:1次

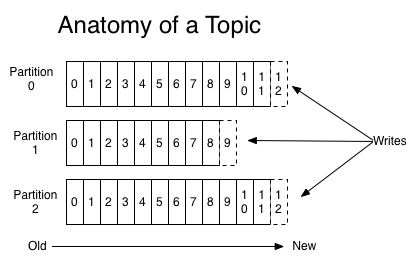
bbiiqqpp:2次

opba:1次 共八次



题目1

因为Kafka的topic里面的消息顺序是顺序写入的。在存储的过程中，每条消息都是被顺序写到磁盘上的。同样消费者消费的时候也是按着顺序来消费的。

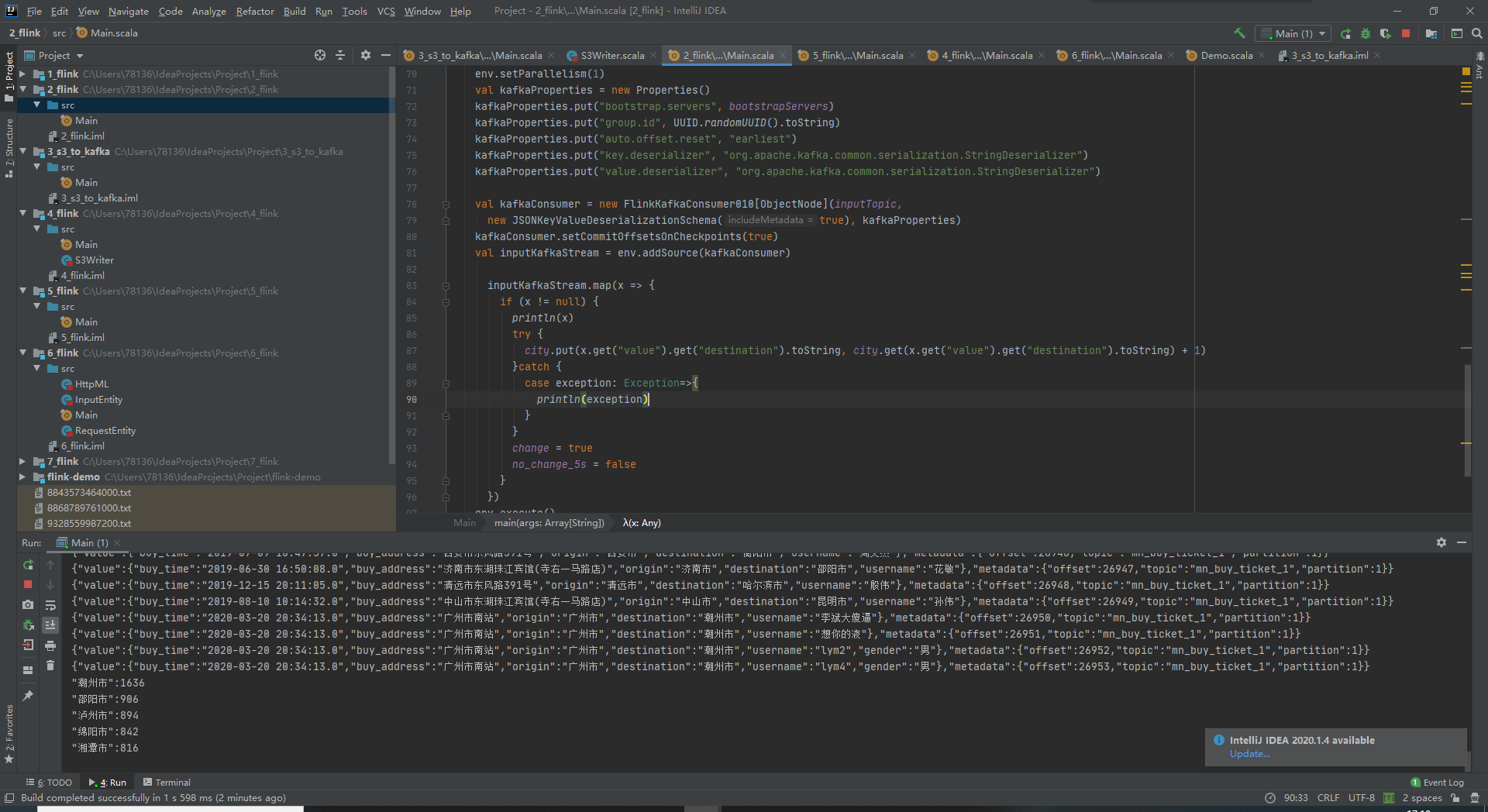


题目2

import java.util

import java.util.{ArrayList, Collections, Comparator, Map, Properties, Set, Timer, TimerTask, UUID}  
import scala.collection.JavaConversions.\_  
  
import org.apache.flink.shaded.jackson2.com.fasterxml.jackson.databind.node.ObjectNode  
import org.apache.flink.streaming.api.scala.\_  
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010  
import org.apache.flink.streaming.util.serialization.JSONKeyValueDeserializationSchema  
  
  
  
object Main {  
 */\*\*  
 \* 输入的主题名称  
 \*/* val *inputTopic* = "mn\_buy\_ticket\_1"  
 */\*\*  
 \* 文件是否修改  
 \*/* var *change* = false  
 */\*\*  
 \* 是否5秒未修改文件  
 \*/* var *no\_change\_5s* = true  
 */\*\*  
 \* kafka地址  
 \*/* val *bootstrapServers* = "bigdata35.depts.bingosoft.net:29035,bigdata36.depts.bingosoft.net:29036,bigdata37.depts.bingosoft.net:29037"  
 */\*\*  
 \* 所有城市作为目的地的数量  
 \*/* var *city* = new util.HashMap[String,Int]()  
 class time\_task(timer: Timer) extends TimerTask{  
 override def run(): Unit = {  
 //若没有数据或数据未提取完毕则不执行  
 if(!*change*) return  
 if (!*no\_change\_5s*) {  
 *no\_change\_5s* = true  
 return  
 }  
  
 *change* = false  
 //将Map装入List中用于城市到达数的排序  
 val list = new ArrayList[Map.Entry[String, Int]](*city*.entrySet)  
 //然后通过比较器来实现排序  
 Collections.*sort*(list, new Comparator[Map.Entry[String, Int]] { //升序排序  
 def compare(o1:Map.Entry[String, Int], o2 :Map.Entry[String, Int]): Int={  
 if (o1.getValue < o2.getValue) return 1  
 if (o1.getValue == o2.getValue)return 0  
 return -1  
 }  
 })  
 //输出到达数前五的城市  
 var count = 0  
 for (mapping <- list) {  
 System.*out*.println(mapping.getKey + ":" + mapping.getValue)  
 count += 1  
 if (count == 5) return  
 }  
 }  
 }  
  
 def main(args: Array[String]): Unit = {  
 var s = new Timer()  
 s.schedule(new time\_task(s),1000, 5000)  
  
 val env = StreamExecutionEnvironment.*getExecutionEnvironment* env.setParallelism(1)  
 val kafkaProperties = new Properties()  
 kafkaProperties.put("bootstrap.servers", *bootstrapServers*)  
 kafkaProperties.put("group.id", UUID.*randomUUID*().toString)  
 kafkaProperties.put("auto.offset.reset", "earliest")  
 kafkaProperties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")  
 kafkaProperties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")  
  
 val kafkaConsumer = new FlinkKafkaConsumer010[ObjectNode](*inputTopic*,  
 new JSONKeyValueDeserializationSchema(true), kafkaProperties)  
 kafkaConsumer.setCommitOffsetsOnCheckpoints(true)  
 val inputKafkaStream = env.addSource(kafkaConsumer)  
  
 inputKafkaStream.map(x => {  
 if (x != null) {  
 *println*(x)  
 try {//提取数据中达到的城市，并在Map中进行统计  
 *city*.put(x.get("value").get("destination").toString, *city*.get(x.get("value").get("destination").toString) + 1)  
 }catch {  
 case exception: Exception=>{  
 *println*(exception)  
 }  
 }  
 //设置修改标记，表示有新数据写入  
 *change* = true  
 *no\_change\_5s* = false  
 }  
 })  
 env.execute()  
 }  
}

使用map进行目的地个数的存储，数据提取完毕后建立List<Map>数组。之后借助于Collections的sort(List<T> list, Comparator<? super T> c)方法对map的value进行排序，该方法根据指定比较器产生的顺序对指定列表进行排序。

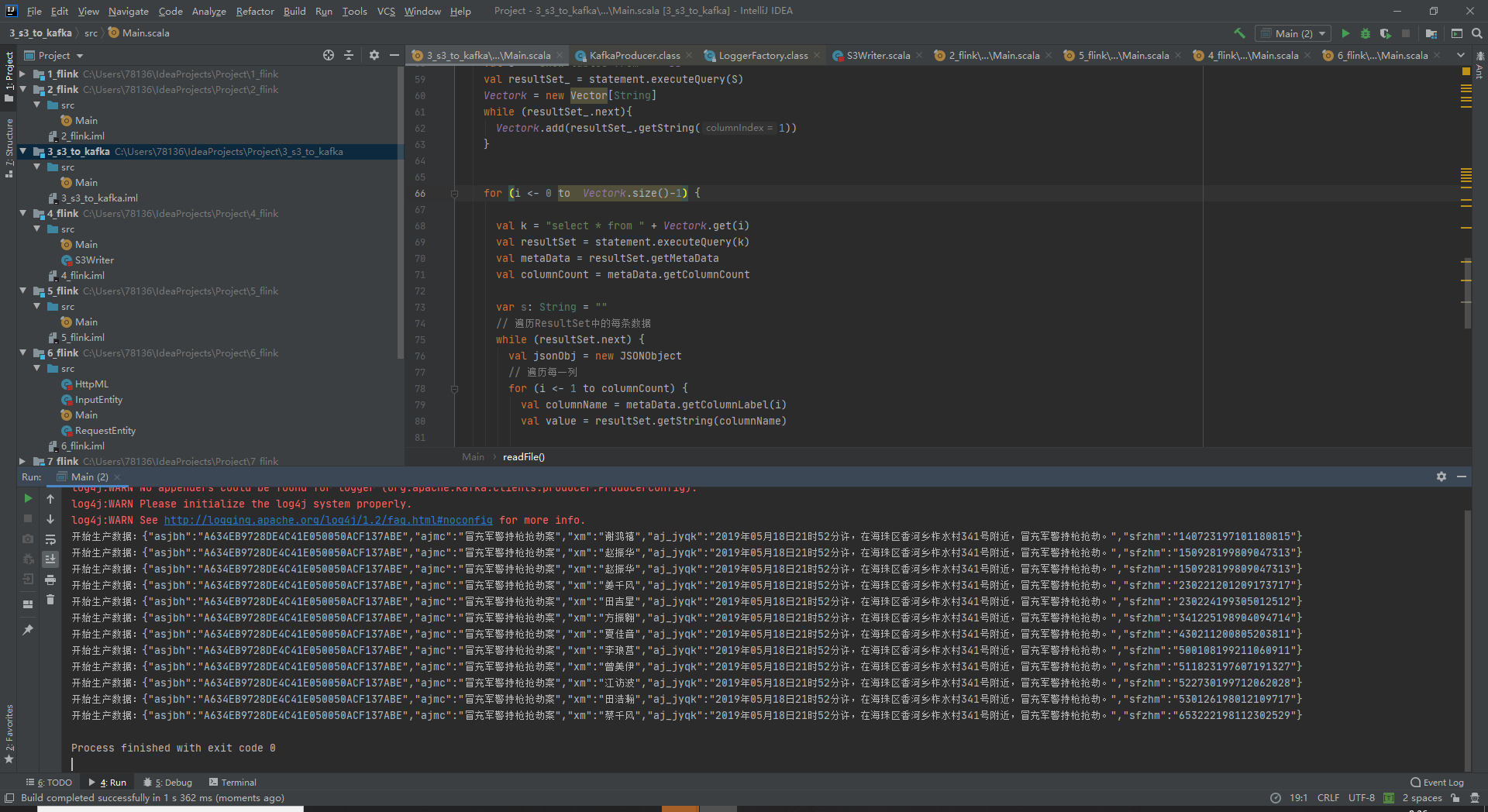




导入[mysql-connector-java-8.0.25.jar](https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.0.25/mysql-connector-java-8.0.25.jar)

import java.util.Properties

import com.bingocloud.{ClientConfiguration, Protocol}  
import com.bingocloud.auth.BasicAWSCredentials  
import com.bingocloud.services.s3.AmazonS3Client  
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerRecord}  
import org.nlpcn.commons.lang.util.IOUtil  
import java.util.List  
import java.sql.DriverManager  
import java.sql.ResultSetMetaData  
import java.util.Vector  
  
import com.bingocloud.util.json.JSONObject  
  
  
object Main {  
  
 //kafka参数  
 val *topic* = "libin9"  
 val *bootstrapServers* = "bigdata35.depts.bingosoft.net:29035,bigdata36.depts.bingosoft.net:29036,bigdata37.depts.bingosoft.net:29037"  
  
 //mysql参数  
 var *IP* = "bigdata130.depts.bingosoft.net";  
 var *PORT* = "24306";  
 var *DB* ="user22\_db";  
 var *UESER* = "user22";  
 var *PASSWORD* = "pass@bingo22";  
 val *properties* = new Properties()  
 val *url* = "jdbc:mysql://" + *IP* + ":" + *PORT* + "/" + *DB*;  
 //设置设备属性  
 *properties*.setProperty("driverClassName", "org.apache.hive.jdbc.HiveDriver")  
 *properties*.setProperty("user", *UESER*)  
 *properties*.setProperty("password", *PASSWORD*);  
 //建立连接  
 val *connection* = DriverManager.*getConnection*(*url*, *properties*)  
  
 def main(args: Array[String]): Unit = {  
 *readFile*()  
 }  
  
 var *Vectork* = new Vector[String]  
  
 */\*\*  
 \* 从sql中读取数据  
 \*/* def readFile(): Unit = {  
 //找到数据库中所有的表  
 val statement = *connection*.createStatement  
 val S = "show tables from " + *DB* val resultSet\_ = statement.executeQuery(S)  
 *Vectork* = new Vector[String]  
 //把所有的表名保存到Vector容器中  
 while (resultSet\_.next){  
 *Vectork*.add(resultSet\_.getString(1))  
 }  
  
 //对所有的表进行遍历提取数据  
 for (i <- 0 to *Vectork*.size()-1) {  
  
 //从mysql读取数据  
 val k = "select \* from " + *Vectork*.get(i)  
 val resultSet = statement.executeQuery(k)  
 val metaData = resultSet.getMetaData  
 val columnCount = metaData.getColumnCount  
  
 var s: String = ""  
 // 遍历ResultSet中的每条数据  
 while (resultSet.next) {  
 val jsonObj = new JSONObject  
 // 遍历每一列  
 for (i <- 1 to columnCount) {  
 val columnName = metaData.getColumnLabel(i)  
 val value = resultSet.getString(columnName)  
 //转化成json格式  
 jsonObj.put(columnName, value)  
 }  
 //将转化后的json转成String类发送给kafka  
 *produceToKafka*(jsonObj.toString)  
 }  
 }  
  
 }  
  
 */\*\*  
 \* 把数据写入到kafka中  
 \*  
 \** ***@param*** *s3Content 要写入的内容  
 \*/* def produceToKafka(s3Content: String): Unit = {  
 val props = new Properties  
 props.put("bootstrap.servers", *bootstrapServers*)  
 props.put("acks", "all")  
 props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")  
 props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")  
 val producer = new KafkaProducer[String, String](props)  
 val dataArr = s3Content.split("\n")  
 for (s <- dataArr) {  
 if (!s.trim.isEmpty) {  
 val record = new ProducerRecord[String, String](*topic*, null, s)  
 *println*("开始生产数据：" + s)  
 producer.send(record)  
 }  
 }  
 producer.flush()  
 producer.close()  
 }  
}





程序

导入[mysql-connector-java-8.0.25.jar](https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.0.25/mysql-connector-java-8.0.25.jar)

Main.scala：

import java.sql.DriverManager

import java.util.{Properties, UUID}  
  
import org.apache.flink.api.common.serialization.SimpleStringSchema  
import org.apache.flink.streaming.api.scala.\_  
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010  
  
object Main {  
 //输入的kafka主题名称  
 val *inputTopic* = "libin9"  
 //kafka地址  
  
 val *bootstrapServers* = "bigdata35.depts.bingosoft.net:29035,bigdata36.depts.bingosoft.net:29036,bigdata37.depts.bingosoft.net:29037"  
   
 def main(args: Array[String]): Unit = {  
  
 val env = StreamExecutionEnvironment.*getExecutionEnvironment* env.setParallelism(1)  
 val kafkaProperties = new Properties()  
 kafkaProperties.put("bootstrap.servers", *bootstrapServers*)  
 kafkaProperties.put("group.id", UUID.*randomUUID*().toString)  
 kafkaProperties.put("auto.offset.reset", "earliest")  
 kafkaProperties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")  
 kafkaProperties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")  
  
 val kafkaConsumer = new FlinkKafkaConsumer010[String](*inputTopic*, new SimpleStringSchema, kafkaProperties)  
 kafkaConsumer.setCommitOffsetsOnCheckpoints(true)  
  
 val inputKafkaStream = env.addSource(kafkaConsumer)  
 inputKafkaStream.writeUsingOutputFormat(new S3Writer())  
 env.execute()  
 }  
}

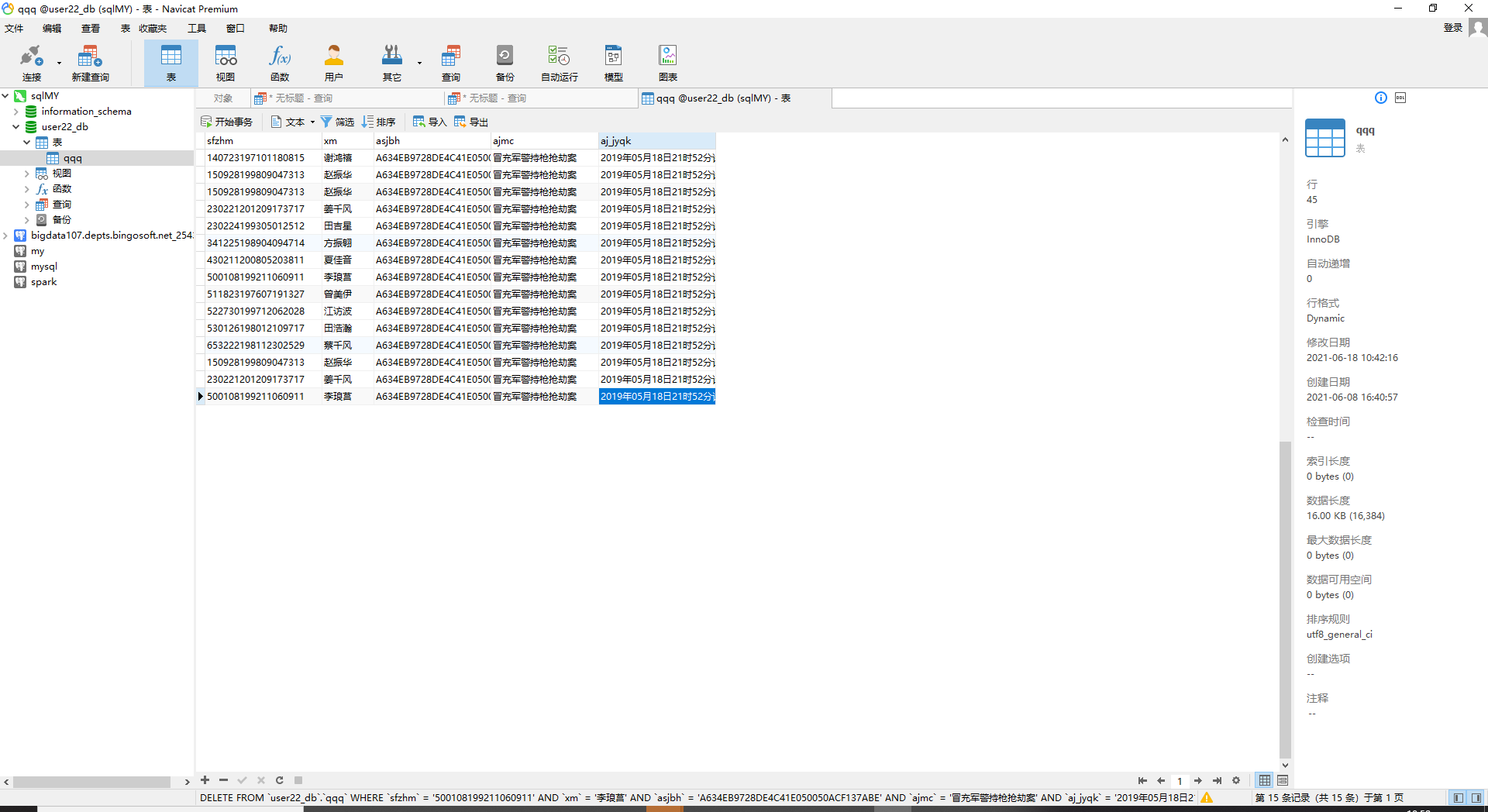
S3Writer.scala：

import java.sql.DriverManager

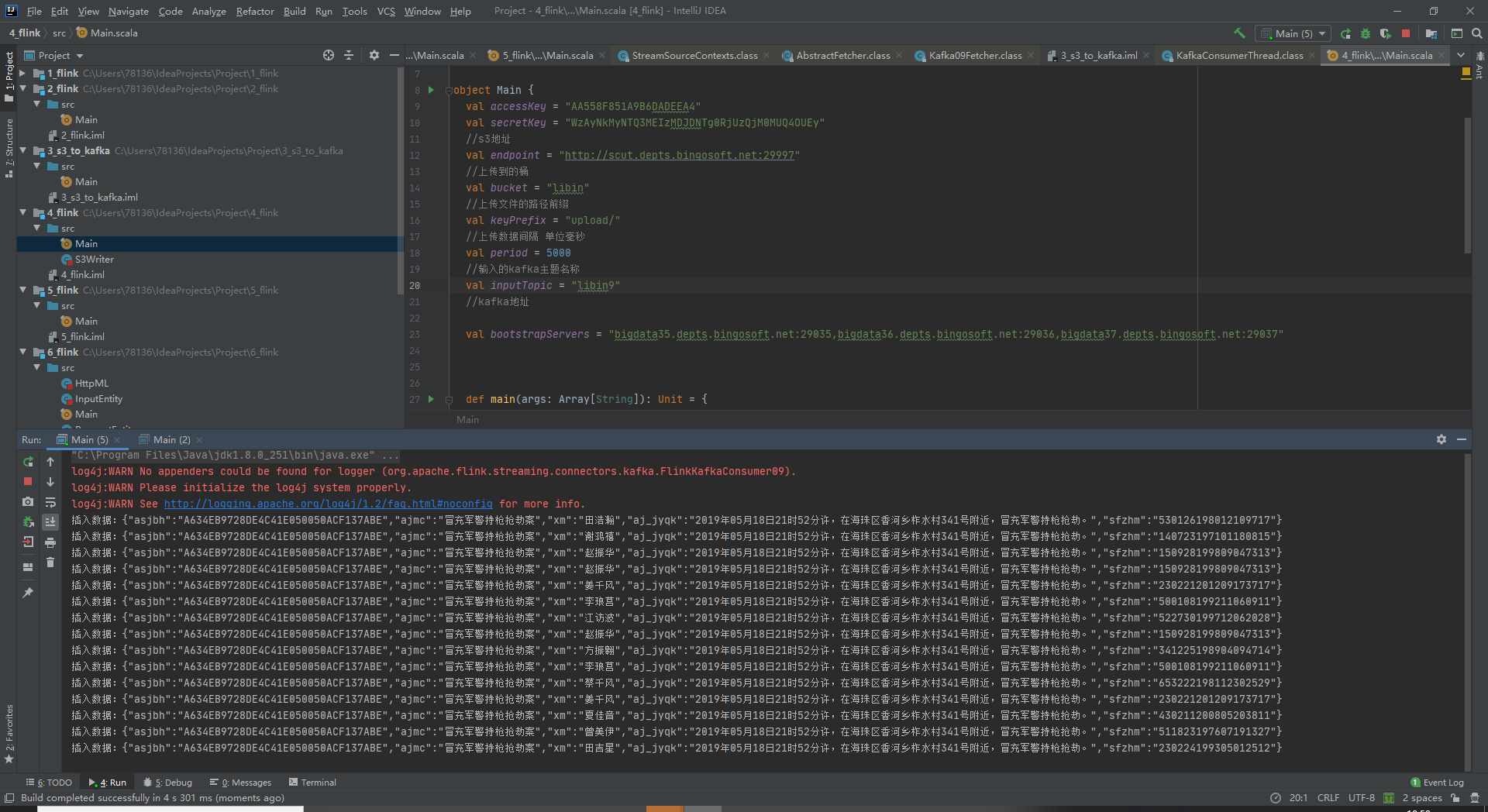
import java.sql.Connection  
import java.util  
import java.util.{Properties, Timer, TimerTask}  
  
import com.bingocloud.util.json.JSONObject  
import org.apache.commons.lang3.StringUtils  
import org.apache.flink.api.common.io.OutputFormat  
import org.apache.flink.configuration.Configuration  
  
//写作S3Writer，读作mysqlWriter  
class S3Writer() extends OutputFormat[String] {  
  
 val *bootstrapServers* = "bigdata35.depts.bingosoft.net:29035,bigdata36.depts.bingosoft.net:29036,bigdata37.depts.bingosoft.net:29037"  
 var *IP* = "bigdata130.depts.bingosoft.net";  
 var *PORT* = "24306";  
 var *DB* ="user22\_db";  
 var *UESER* = "user22";  
 var *PASSWORD* = "pass@bingo22";  
 val *properties* = new Properties()  
 val *url* = "jdbc:mysql://" + *IP* + ":" + *PORT* + "/" + *DB*;  
  
 *properties*.setProperty("driverClassName", "org.apache.hive.jdbc.HiveDriver")  
 *properties*.setProperty("user", *UESER*)  
 *properties*.setProperty("password", *PASSWORD*);  
 var *connection*:Connection= null  
  
  
 override def configure(configuration: Configuration): Unit = {  
 *connection* = DriverManager.*getConnection*(*url*, *properties*)  
 }  
  
 override def open(taskNumber: Int, numTasks: Int): Unit = {  
  
 }  
 //获取到数据it后直接写入mysql  
 override def writeRecord(it: String): Unit = {  
 this.synchronized {  
 if (StringUtils.*isNoneBlank*(it)) {  
 //将数据转化成json格式  
 var cnt\_it = new JSONObject(it)  
 var keyV = ""  
 var op = ""  
  
 //通过Iterator对json进行遍历，访问key  
 var it\_ = cnt\_it.keys()  
 while(it\_.hasNext()){  
 //访问json数据中所有的key  
 val key = it\_.next.toString  
 keyV = keyV+key+','  
 op = op+"?,"  
 }  
  
 //生成sql命令，准备插入数据  
 keyV = keyV.substring(0,keyV.length-1)  
 op = op.substring(0,op.length-1)  
 val sql = "INSERT INTO qqq(" + keyV+") VALUES ("+op+")"  
  
 var pst = *connection*.prepareStatement(sql)  
 //梅开二度，再次通过Iterator对json进行遍历，访问value  
 it\_ = cnt\_it.keys()  
 var k = 1  
 while(it\_.hasNext()){  
 val key = it\_.next.toString  
 //访问json数据中所有的value，并插入mysql  
 pst.setString(k, cnt\_it.getString(key))  
 k += 1  
 }  
 //更新操作，插入完毕  
 pst.executeUpdate()  
 *println*("插入数据："+it)  
  
 }  
 }  
 }  
  
  
 override def close(): Unit = {  
 }  
}

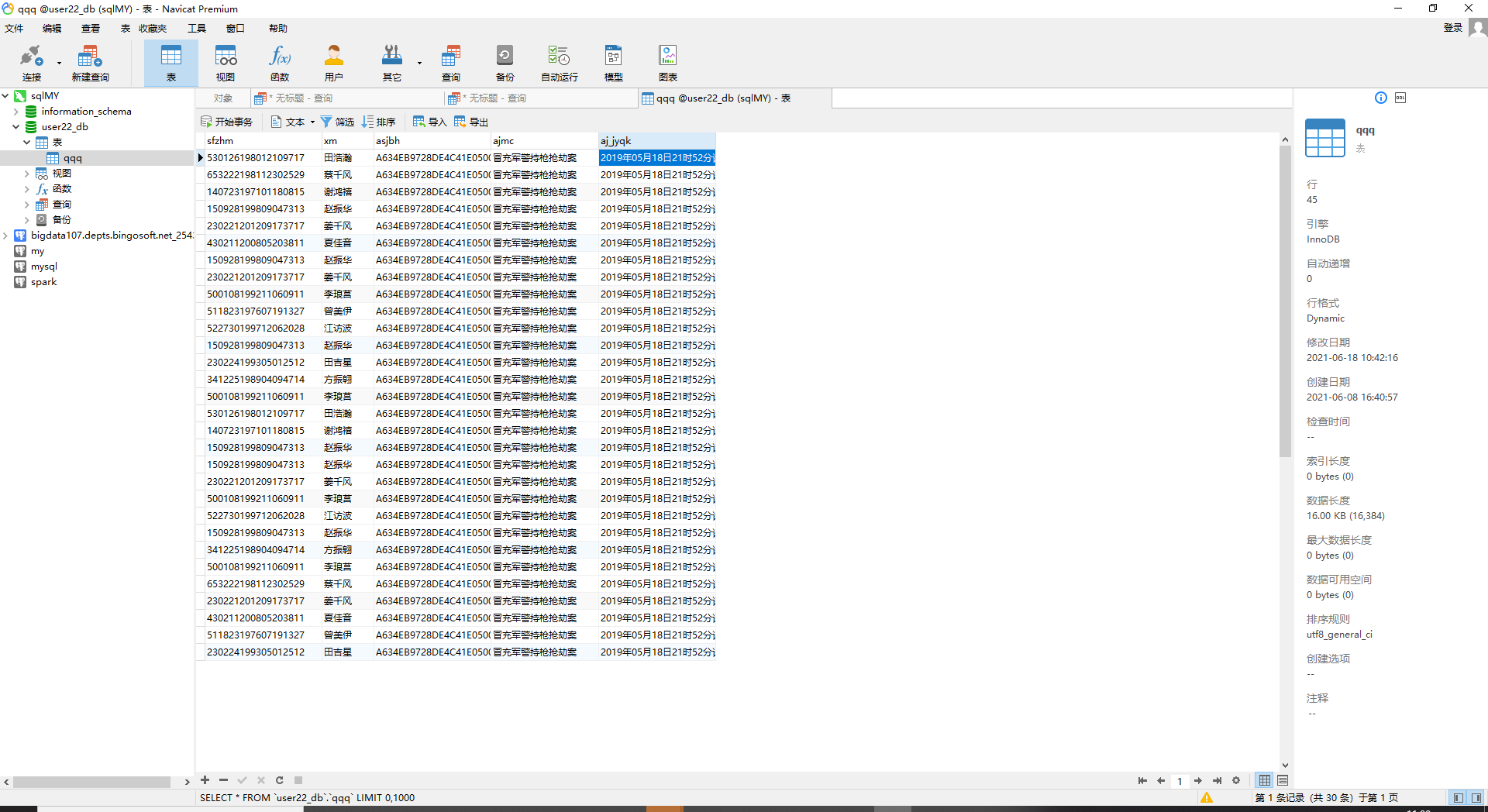
利用writeRecord提取出来的每一个it转换成JSONObject格式。在对JSONObject进行分析，提取出每一个key和value

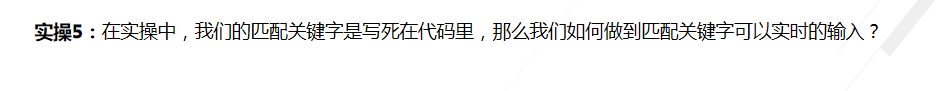
插入前：



插入后：







import scala.collection.JavaConversions.\_

import java.util  
import java.util.{Map, Properties, Scanner, UUID, HashSet,Set}  
  
import com.bingocloud.util.json.JSONObject  
import org.apache.flink.api.common.serialization.SimpleStringSchema  
import org.apache.flink.streaming.api.scala.\_  
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010  
  
object Main {  
 val *inputTopics*: util.ArrayList[String] = new util.ArrayList[String]() {  
 {  
 add("mn\_buy\_ticket\_1") //车票购买记录主题  
 add("mn\_hotel\_stay\_1") //酒店入住信息主题  
 add("mn\_monitoring\_1") //监控系统数据主题  
 }  
 }  
 val *bootstrapServers* = "bigdata35.depts.bingosoft.net:29035,bigdata36.depts.bingosoft.net:29036,bigdata37.depts.bingosoft.net:29037"  
  
 //所有人名的信息保存在Map中，每个人的信息记录下相应Key的Vector中  
 var *Name\_data*:Map[String, Set[String]] = new util.HashMap[String, Set[String]]()  
  
 //新建类继承线程，保证能够实时输入  
 class find\_name extends Thread{  
 override def run(): Unit = {  
 super.run()  
 val input = new Scanner(System.*in*)  
 while(true) {  
 System.*out*.println("请输入查询姓名")  
 //等待搜索名字输入  
 val val1 = input.next();  
 //查找该名字下的所有信息  
 val all = *Name\_data*.get(val1)  
  
 //如果找到此人则输出所有信息  
 if (all == null) *println*("查无此人")  
 else for (i <- all) *println*("查询结果："+i)  
 }  
 }  
 }  
  
 var *s* = new find\_name()  
 def main(args: Array[String]): Unit = {  
 *s*.start()  
 val env = StreamExecutionEnvironment.*getExecutionEnvironment* env.setParallelism(1)  
 val kafkaProperties = new Properties()  
 kafkaProperties.put("bootstrap.servers", *bootstrapServers*)  
 kafkaProperties.put("group.id", UUID.*randomUUID*().toString)  
 kafkaProperties.put("auto.offset.reset", "earliest")  
 kafkaProperties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")  
 kafkaProperties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")  
  
 //由于数据被污染，所以为了防止读入非法json数据报错，先以String类型读入  
 val kafkaConsumer = new FlinkKafkaConsumer010[String](*inputTopics*, new SimpleStringSchema, kafkaProperties)  
 kafkaConsumer.setCommitOffsetsOnCheckpoints(true)  
 val inputKafkaStream = env.addSource(kafkaConsumer)  
  
 inputKafkaStream.map(x => {  
 if (x != null) {  
 //杜绝传入非法数据，从我做起  
 try {  
 //把kafka数据从String转成json方便操作  
 val cnt\_it = new JSONObject(x)  
 //提取数据中的姓名  
 val name = cnt\_it.get("username").toString()  
 if (name != null) {  
 if (*Name\_data*.get(name) == null) {  
 *Name\_data*.put(name, new HashSet[String]())  
 }  
 //以姓名作为Key，信息为Value，将信息存入Map中，  
 *Name\_data*.get(name).add(x)  
 *println*("读取数据：" + x + " " + *Name\_data*.get(name).size())  
 }  
 } catch {  
 case exception: Exception => {  
 *println*(exception)  
 }  
 }  
 }  
 })  
 env.execute()  
 }  
}

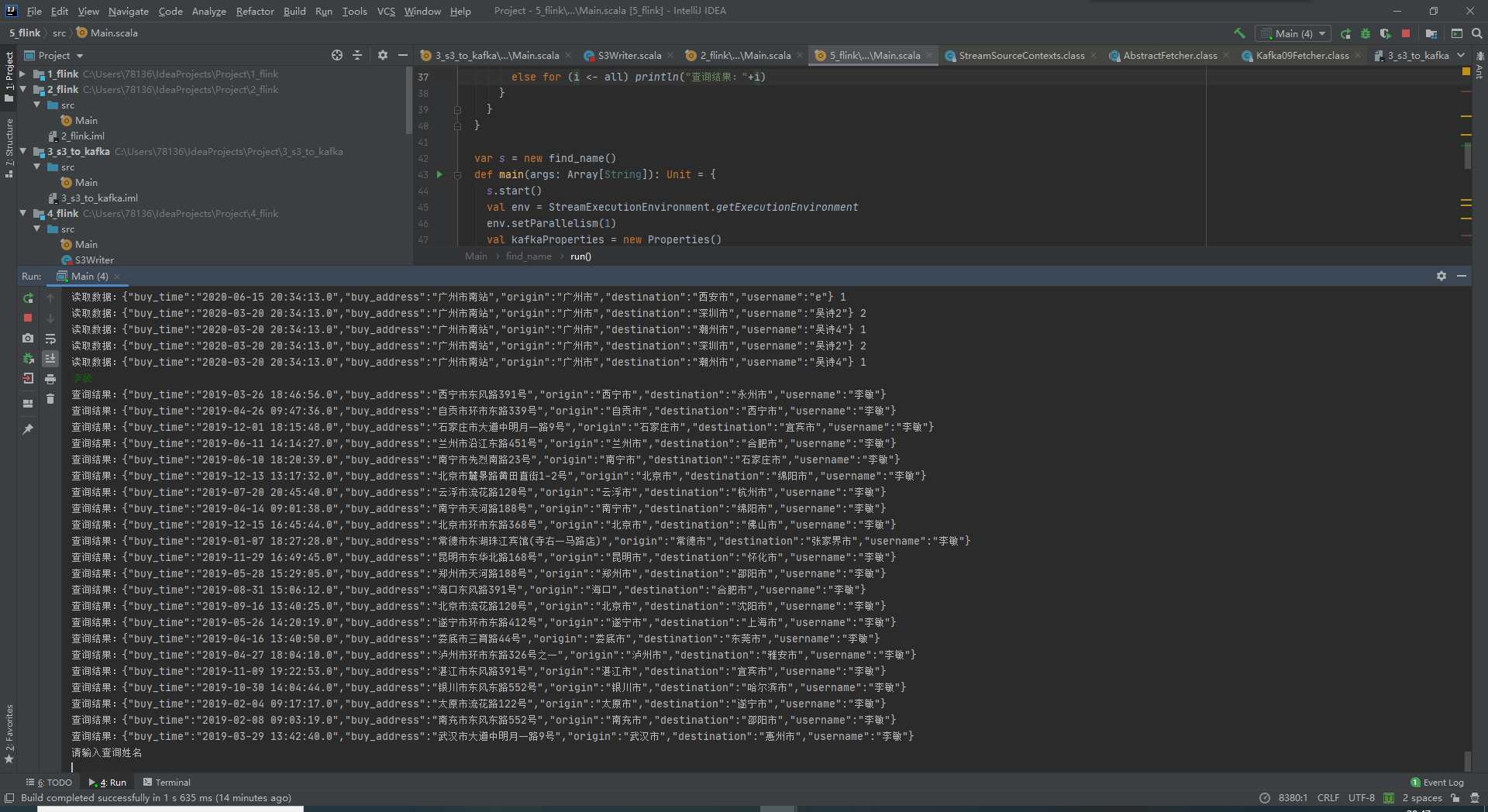
因为topic被传入非法数据，所以我们提取kafka中数据先以String类型进行提取，然后再在try中转化成json类型，防止直接提取json时程序异常终止。

提取出数据后，先分离出姓名，然后以姓名作为Key，信息为Value，将信息存入Map中。由于一个人有多个信息，所以在Map以Set作为Value类型保存该姓名下所有信息。Set继承于Collection接口，是一个不允许出现重复元素，并且无序的集合。所以当我们在记录信息时，能自动过滤掉重复的信息。

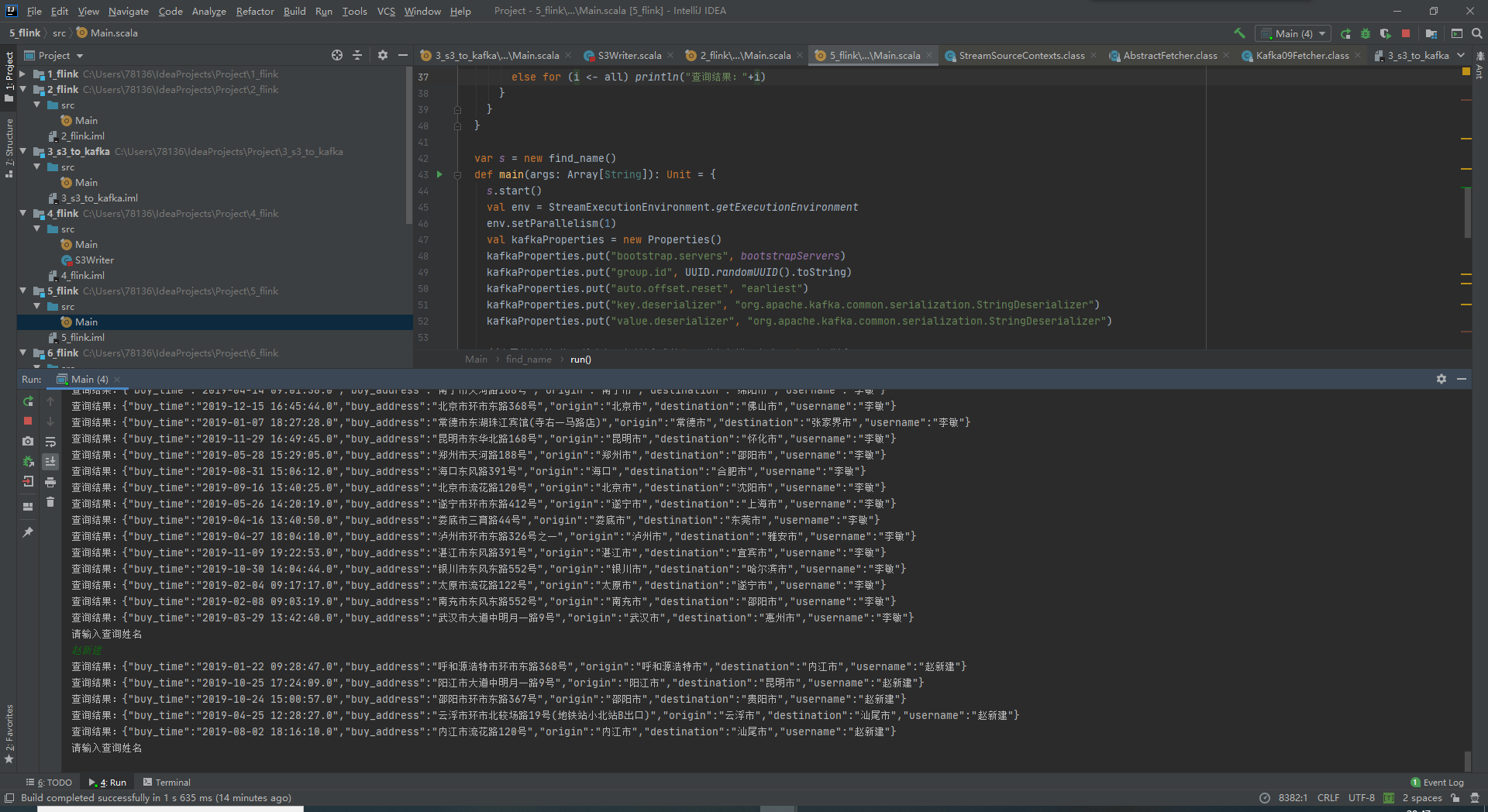
新建线程持续监听控制台输入，等待输入姓名后访问以输入为Key访问保存在Map中的所有信息。

输入结果：

查找“*李敏*”的行踪



查找“*赵新建*”的行踪



查找“*李雨泽*”的行踪

