**尚硅谷大数据项目**

**电商分析系统**

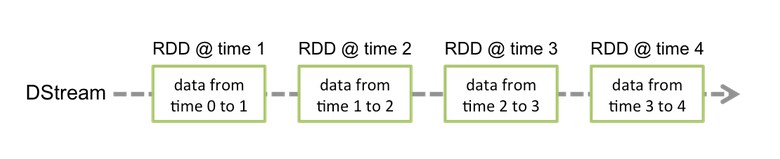
**V2.0**

**实时统计：**

# 1 预备知识

什么是DStream

    DSream 代表了一系列连续的RDD，DStream中每个RDD包含特定时间间隔的数据



常用算子

转换操作：

|  |  |
| --- | --- |
| 算子名称 | 说明 |
| map | 对每个传入的元素，返回一个新的元素 |
| flatMap | 对每个传入的元素，返回一个或多个元素 |
| fliter | 对传入的元素返回 true 或 false，返回 false 的元素将会被过滤掉 |
| union | 将连个 Dsrteam 进行合并 |
| count | 返回元素的个数 |
| reduce | 对所有 value 进行聚合 |
| countByKey | 对元素按照值进行分组，对每个组进行计数，最后返回<K, V>的格式 |
| reduceByKey | 对 Key 对应的 values 进行聚合 |
| join | 对两个 Dstream 进行 join 操作，每个链接起来的 pair，作为新 Dstream 的 RDD 的一个元素 |
| transform {rdd=>rdd} | //driver 整个任务只执行一次  xxxxDstream.transform{  //driver 周期性执行  xxxrdd=>rdd.map{//executor 中执行}  }  xxxxDstream.map{  //executor  } |
| updateStateByKey | 为每一个rdd中的key ，保留状态，持续更新  ps: 在大规模数据场景下，可以用redis替代 |
| window | 滑动窗口，滑动窗口和滑动步长必须是取数周期的整数倍 |

输出操作

|  |  |
| --- | --- |
| 算子名称 | 说明 |
| saveAsTextFiles | 保存到文件系统 |
| saveAsHadoopFiles | 保存到Hdfs中 |
| foreachRDD | 针对每个时间点的RDD做相应操作 |

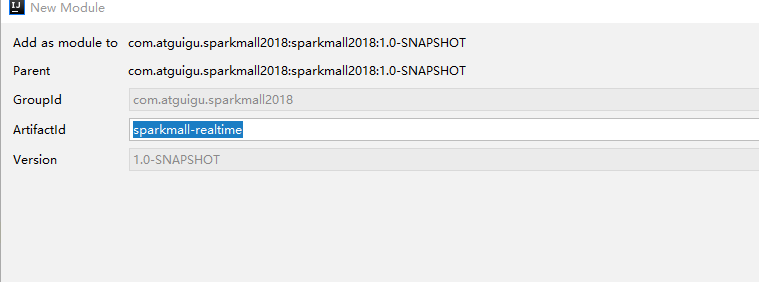
# 2 数据准备

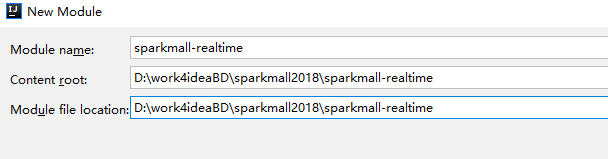
模拟数据生产的类，在mock工程中添加

|  |
| --- |
| **import** java.util.Properties  **import** com.atguigu.sparkmall.common.conf.ConfigurationUtil **import** com.atguigu.sparkmall.common.model.CityInfo **import** com.atguigu.sparkmall.mock.util.{RanOpt, RandomOptions} **import** org.apache.kafka.clients.producer.{KafkaProducer, ProducerConfig, ProducerRecord}  **import** scala.collection.mutable.ArrayBuffer **import** scala.util.Random  **object** MockRealTimeGenerator {   */\*\*  \* 模拟的数据    \* 格式 ：timestamp area city userid adid  \* 某个时间点 某个地区 某个城市 某个用户 某个广告  \*/* **def** generateMockData(): Array[String] = {  **val** array = ArrayBuffer[String]()  **val** CityRandomOpt = *RandomOptions*(*RanOpt*( *CityInfo*(1,**"北京"**,**"华北"**),30),  *RanOpt*(*CityInfo*(1,**"上海"**,**"华东"**),30),  *RanOpt*(*CityInfo*(1,**"广州"**,**"华南"**),10),  *RanOpt*(*CityInfo*(1,**"深圳"**,**"华南"**),20),  *RanOpt*(*CityInfo*(1,**"天津"**,**"华北"**),10))   **val** random = **new** Random()  *// 模拟实时数据：  // timestamp province city userid adid* **for** (i <- 0 to 50) {   **val** timestamp = System.*currentTimeMillis*()  **val** cityInfo = CityRandomOpt.getRandomOpt()  **val** city = cityInfo.city\_name  **val** area = cityInfo.area  **val** adid = 1+random.nextInt(6)  **val** userid = 1+random.nextInt(6)   *// 拼接实时数据* array += timestamp + **" "** + area + **" "** + city + **" "** + userid + **" "** + adid  }  array.toArray  }   **def** createKafkaProducer(broker: String): KafkaProducer[String, String] = {   *// 创建配置对象* **val** prop = **new** Properties()  *// 添加配置* prop.put(ProducerConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, broker)  prop.put(ProducerConfig.*KEY\_SERIALIZER\_CLASS\_CONFIG*, **"org.apache.kafka.common.serialization.StringSerializer"**)  prop.put(ProducerConfig.*VALUE\_SERIALIZER\_CLASS\_CONFIG*, **"org.apache.kafka.common.serialization.StringSerializer"**)   *// 根据配置创建Kafka生产者* **new** KafkaProducer[String, String](prop)  }    **def** main(args: Array[String]): Unit = {   *// 获取配置文件commerce.properties中的Kafka配置参数* **val** config = ConfigurationUtil.*apply*(**"config.properties"**).*config* **val** broker = config.getString(**"kafka.broker.list"**)  **val** topic = **"ads\_log"** *// 创建Kafka消费者* **val** kafkaProducer = *createKafkaProducer*(broker)   **while** (**true**) {  *// 随机产生实时数据并通过Kafka生产者发送到Kafka集群中* **for** (line <- *generateMockData*()) {  kafkaProducer.send(**new** ProducerRecord[String, String](topic, line))  *println*(line)  }  Thread.*sleep*(2000)  }  } } |

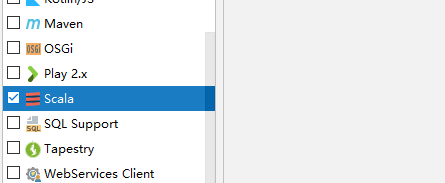
# 3工程搭建

新建模块





添加frameword



**pom.xml**

|  |
| --- |
| *<?***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**>sparkmall2018</**artifactId**>  <**groupId**>com.atguigu.sparkmall2018</**groupId**>  <**version**>1.0-SNAPSHOT</**version**>  </**parent**>  <**modelVersion**>4.0.0</**modelVersion**>   <**artifactId**>sparkmall-realtime</**artifactId**>   <**dependencies**>   <**dependency**>  <**groupId**>com.atguigu.sparkmall2018</**groupId**>  <**artifactId**>sparkmall-common</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-core\_2.11</**artifactId**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-sql\_2.11</**artifactId**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-hive\_2.11</**artifactId**>  </**dependency**>   </**dependencies**> </**project**> |

消费kafka的工具类 ， 在common中增加KafkaUtil

|  |
| --- |
| **import** org.apache.kafka.clients.consumer.ConsumerRecord **import** org.apache.kafka.common.serialization.StringDeserializer **import** org.apache.spark.streaming.StreamingContext **import** org.apache.spark.streaming.dstream.InputDStream **import** org.apache.spark.streaming.kafka010.{ConsumerStrategies, KafkaUtils, LocationStrategies}     **object My**KafkaUtil {   **val** *config* = *ConfigurationUtil*(**"config.properties"**).*config* **val** *broker\_list* = *config*.getString(**"kafka.broker.list"**)   *// kafka消费者配置* **val** *kafkaParam* = *Map*(  **"bootstrap.servers"** -> *broker\_list*,*//用于初始化链接到集群的地址* **"key.deserializer"** -> *classOf*[StringDeserializer],  **"value.deserializer"** -> *classOf*[StringDeserializer],  *//用于标识这个消费者属于哪个消费团体* **"group.id"** -> **"commerce-consumer-group"**,  *//如果没有初始化偏移量或者当前的偏移量不存在任何服务器上，可以使用这个配置属性  //可以使用这个配置，latest自动重置偏移量为最新的偏移量* **"auto.offset.reset"** -> **"latest"**,  *//如果是true，则这个消费者的偏移量会在后台自动提交,但是kafka宕机容易丢失数据  //如果是false，会需要手动维护kafka偏移量* **"enable.auto.commit"** -> (**true**: java.lang.Boolean)  )   *// 创建DStream，返回接收到的输入数据  // LocationStrategies：根据给定的主题和集群地址创建consumer  // LocationStrategies.PreferConsistent：持续的在所有Executor之间分配分区  // ConsumerStrategies：选择如何在Driver和Executor上创建和配置Kafka Consumer  // ConsumerStrategies.Subscribe：订阅一系列主题* **def** getKafkaStream(topic: String,ssc:StreamingContext): InputDStream[ConsumerRecord[String,String]]={  **val** dStream = KafkaUtils.*createDirectStream*[String,String](ssc, LocationStrategies.*PreferConsistent*,ConsumerStrategies.*Subscribe*[String,String](*Array*(topic),*kafkaParam*))  dStream  } } |

|  |
| --- |
| **object** RedisUtil {   **var** *jedisPool*:JedisPool=**null   def** getJedisClient: Jedis = {  **if**(*jedisPool*==**null**){  *println*(**"开辟一个连接池"**)  **val** config = *ConfigurationUtil*(**"config.properties"**).*config* **val** host = config.getString(**"redis.host"**)  **val** port = config.getInt(**"redis.port"**)   **val** jedisPoolConfig = **new** JedisPoolConfig()  jedisPoolConfig.setMaxTotal(100) *//最大连接数* jedisPoolConfig.setMaxIdle(20) *//最大空闲* jedisPoolConfig.setMinIdle(20) *//最小空闲* jedisPoolConfig.setBlockWhenExhausted(**true**) *//忙碌时是否等待* jedisPoolConfig.setMaxWaitMillis(500)*//忙碌时等待时长 毫秒* jedisPoolConfig.setTestOnBorrow(**true**) *//每次获得连接的进行测试   jedisPool*=**new** JedisPool(jedisPoolConfig,host,port)  }  *println*(**s"jedisPool.getNumActive = $**{*jedisPool*.getNumActive}**"**)  *println*(**"获得一个连接"**)  *jedisPool*.getResource  } } |

# 需求四：

实时数据分析： 广告黑名单实时统计

# 1 需求简介

实现实时的动态黑名单机制：将**每天**对某个广告**点击**超过 **100** 次的用户拉黑。

注：黑名单保存到redis中。

已加入黑名单的用户不再进行检查。

# 2 数据源字段：

*timestamp province city userid adid*

|  |  |
| --- | --- |
| 字段名 |  |
| timestamp | **当前时间毫秒** |
| area | **地区：华北、华南** |
| city | **城市名： 北京、上海、广州、深圳、天津** |
| userid | **用户id 1-6** |
| adid | **广告id 1-6** |

topic: ads\_log

# 3 目标数据字段

redis type: set

|  |  |
| --- | --- |
| key | value |
| blacklist | 存储黑名单用户userId |

# 4 思路：

附注：

Kafka测试

生产端

|  |
| --- |
| /bigdata/kafka\_2.11-0.11.0.2/bin/kafka-console-producer.sh --broker-list hadoop1:9092,hadoop2:9092,hadoop3:9092 --topic ads\_log |

消费端

|  |
| --- |
| /bigdata/kafka\_2.11-0.11.0.2/bin/kafka-console-consumer.sh --bootstrap-server hadoop1:9092,hadoop2:9092:9092,hadoop3:9092 --topic ads\_log --from-beginning |