|  |
| --- |
| **object** StreamWordCount {   **def** main(args: Array[String]): Unit = {   *//离线任务是创建SparkContext，现在要实现实时计算，用StreamingContext* **val** conf = **new** SparkConf().setAppName(**"StreamWordCount"**).setMaster(**"local[2]"**)  **val** sc = **new** SparkContext(conf)  *//StreamingContext是对SparkContext的包装，包了一层就增加了实时的功能  //第二个参数是小批次产生的时间间隔* **val** ssc = **new** StreamingContext(sc, *Milliseconds*(5000))   *//有了StreamingContext，就可以创建SparkStreaming的抽象了DSteam  //从一个socket端口中读取数据  //在Linux上用yum安装nc  //yum install -y nc* **val** lines: ReceiverInputDStream[String] = ssc.socketTextStream(**"192.168.137.46"**, 8888)  *//对DSteam进行操作，你操作这个抽象（代理，描述），就像操作一个本地的集合一样  //切分压平* **val** words: DStream[String] = lines.flatMap(\_.split(**" "**))  *//单词和一组合在一起* **val** wordAndOne: DStream[(String, Int)] = words.map((\_, 1))  *//聚合* **val** reduced: DStream[(String, Int)] = wordAndOne.reduceByKey(\_+\_)  *//打印结果(Action)* reduced.print()   *//启动sparksteaming程序* ssc.start()  *//等待优雅的退出* ssc.awaitTermination()    }  } |

|  |
| --- |
| **object** KafkaWordCount {   **def** main(args: Array[String]): Unit = {  **val** conf = **new** SparkConf().setAppName(**"KafkaWordCount"**).setMaster(**"local[\*]"**)  **val** ssc = **new** StreamingContext(conf, *Seconds*(5))   **val** zkQuorum = **"master:2181,server01:2181,server02:2181"  val** groupId = **"g1"  val** topic = *Map*[String, Int](**"bkd"** -> 1)   *//创建DStream，需要KafkaDStream* **val** data: ReceiverInputDStream[(String, String)] = KafkaUtils.*createStream*(ssc, zkQuorum, groupId, topic)  *//对数据进行处理  //Kafak的ReceiverInputDStream[(String, String)]里面装的是一个元组（key是写入的key，value是实际写入的内容）* **val** lines: DStream[String] = data.map(\_.\_2)  *//对DSteam进行操作，你操作这个抽象（代理，描述），就像操作一个本地的集合一样  //切分压平* **val** words: DStream[String] = lines.flatMap(\_.split(**" "**))  *//单词和一组合在一起* **val** wordAndOne: DStream[(String, Int)] = words.map((\_, 1))  *//聚合* **val** reduced: DStream[(String, Int)] = wordAndOne.reduceByKey(\_+\_)  *//打印结果(Action)* reduced.print()  *//启动sparksteaming程序* ssc.start()  *//等待优雅的退出* ssc.awaitTermination()    }  } |

|  |
| --- |
| **package** com.bkd.spark  **import** org.apache.spark.{HashPartitioner, SparkConf} **import** org.apache.spark.streaming.dstream.{DStream, ReceiverInputDStream} **import** org.apache.spark.streaming.kafka.KafkaUtils **import** org.apache.spark.streaming.{Seconds, StreamingContext}  **object** StatefulKafkaWordCount {   */\*\*  \* 第一个参数：聚合的key，就是单词  \* 第二个参数：当前批次产生批次该单词在每一个分区出现的次数  \* 第三个参数：初始值或累加的中间结果  \*/* **val** *updateFunc* = (iter: Iterator[(String, Seq[Int], Option[Int])]) => {  *//iter.map(t => (t.\_1, t.\_2.sum + t.\_3.getOrElse(0)))* iter.map{ **case**(x, y, z) => (x, y.sum + z.getOrElse(0))}  }   **def** main(args: Array[String]): Unit = {   **val** conf = **new** SparkConf().setAppName(**"StatefulKafkaWordCount"**).setMaster(**"local[\*]"**)   **val** ssc = **new** StreamingContext(conf, *Seconds*(5))   *//如果要使用课更新历史数据（累加），那么就要把终结结果保存起来* ssc.checkpoint(**"./ck"**)   **val** zkQuorum = **"master:2181,server01:2181,server02:2181"  val** groupId = **"g100"  val** topic = *Map*[String, Int](**"bkd"** -> 1)   *//创建DStream，需要KafkaDStream* **val** data: ReceiverInputDStream[(String, String)] = KafkaUtils.*createStream*(ssc, zkQuorum, groupId, topic)  *//对数据进行处理  //Kafak的ReceiverInputDStream[(String, String)]里面装的是一个元组（key是写入的key，value是实际写入的内容）* **val** lines: DStream[String] = data.map(\_.\_2)  *//对DSteam进行操作，你操作这个抽象（代理，描述），就像操作一个本地的集合一样  //切分压平* **val** words: DStream[String] = lines.flatMap(\_.split(**" "**))  *//单词和一组合在一起* **val** wordAndOne: DStream[(String, Int)] = words.map((\_, 1))  *//聚合* **val** reduced: DStream[(String, Int)] = wordAndOne.updateStateByKey(*updateFunc*, **new** HashPartitioner(ssc.sparkContext.defaultParallelism), **true**)  *//打印结果(Action)* reduced.print()  *//启动sparksteaming程序* ssc.start()  *//等待优雅的退出* ssc.awaitTermination()   }  } |