第6章Spark Streaming实训

CH06实训1：实时过滤歌曲播放次数超过100次的记录

训练要点

1. Spark Streaming从文件系统读取数据
2. DStream的转换操作
3. DStream的输出操作

需求说明

参考代码6-14编写一个模拟器，根据数据准备环节提供的音乐数据，数据包含3 个字段，分别是userid(用户ID)，artistid(艺术家ID)，playcount(播放次数)，然后编程实现随机生成新的日志文件，并存放在HDFS上的/user/myname/streaming_music/目录下。再通过编程实现由Spark-Streaming监控该目录，筛选出新文件中播放次数大于100的记录，并存放在 HDFS的/user/myname/output_music/目录下

实现步骤

1. 下载测试数据，编写模拟器代码
2. 在 IDEA中编写代码，过滤符合条件的数据
3. 运行程序

作业要求

1. 以截图提供模拟器的代码和过滤符合条件的数据的代码截图
2. 截图在IDEA上运行模拟器的截图，和过滤符合条件的数据程序截图
3. 截图在 HDFS上生成文件/user/myname/output_music/下的内容

实现参考

测试数据准备

根据笔记本内存大小下载文件:

内存小于或等8G的下载： http://bigdata.hddly.cn/b46488/file/chap6/user_artist_data_small.rar

内存大于8G的笔记本下载：http://bigdata.hddly.cn/b46488/file/chap6/user_artist_data.rar

将下载下来的user_artist_data.rar或者user_artist_data_small.rar解压，将解压出来的user_artist_data.txt文件复制到D:\tmp目录下

HDFS上测试路径准备

在master或slave上执行

```
hdfs dfs -chmod -R 777
hdfs dfs -mkdir -p /user/myname/tmp/
cd /root/spark/
wget http://biglab.site//b59510spark/file/user_artist_data.tar.gz
tar -xzvf ./user_artist_data.tar.gz
hdfs dfs -put ./user_artist_data.txt /user/myname/

hdfs dfs -rm -r /user/myname/tmp/streaming
hdfs dfs -rm -r /user/myname/output_music
hdfs dfs -rm -r /user/myname/streaming_music
```

模拟程序SimulatorMusic

```
package chap06t

import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.fs.{FileSystem, FileUtil, Path}

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.rdd.RDD

import java.text.SimpleDateFormat
import java.util.Date
import scala.collection.mutable.ListBuffer
import scala.io.Source
object SimulatorMusic {
  def main(args: Array[String]) {
    var i=0
    var iFileMaxLine=1000
    val hdfsconf: Configuration = new Configuration();
    hdfsconf.set("fs.defaultFS", "hdfs://master:9864");
    System.setProperty("HADOOP_USER_NAME", "root");
    val hdfsfs: FileSystem = FileSystem.get(hdfsconf);
    val sparkconf = new
SparkConf().setAppName("simulator").setMaster("local")
    val sc = new SparkContext(sparkconf)
//    val filename="D:\\tmp\\user_artist_data.txt"
//    val lines = Source.fromFile(filename).getLines.toList
    val filename="hdfs://master:9864/user/myname/user_artist_data.txt"
    val lines = sc.textFile(filename).collect().toList
    val filerow = lines.length
    val tmprows:ListBuffer[String]=  ListBuffer()
    //请将下方链接中myname替换为本人姓名全拼
    val outputtargetpath="hdfs://master:9864/user/myname/streaming_music/"
    while (true)
    {
      val streamingpath=outputtargetpath+"/streamingdata_" +i
      hdfsfs.delete(new Path(streamingpath),true)
      i=i+1
      var j=0
      while(j<iFileMaxLine)
      {
        val linstr=lines(index(filerow)).toString
        tmprows.+=(linstr)
        println(linstr)
        j=j+1
      }
      val linelog: RDD[String] = sc.parallelize(tmprows)
      linelog.repartition(1).saveAsTextFile(streamingpath)
      tmprows.clear()
      Thread sleep 1000
      log(getNowTime(),streamingpath+" generated")
    }
  }
  def log(date: String, message: String)  = {
    println(date + "----" + message)
```

```
  }
  def index(length: Int) = {
    import java.util.Random
    val rdm = new Random
    rdm.nextInt(length)
  }
  def getNowTime():String={
    val now:Date = new Date()
    val datetimeFormat:SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss")
    val ntime = datetimeFormat.format( now )
    ntime
  }
}

```

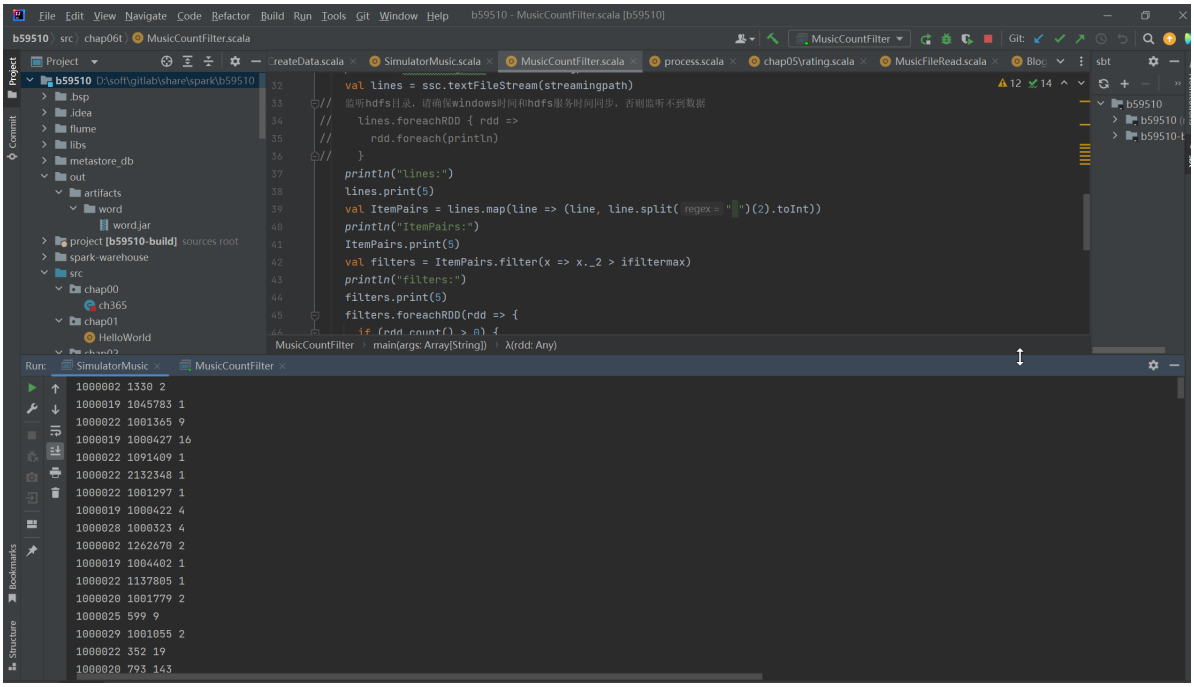
分析程序MusicCountFilter

```
package chap06t

import org.apache.hadoop.fs.Path
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.fs.{FileSystem, FileUtil, Path}
import org.apache.spark.repl.Main.sparkContext

import java.sql.{Connection, DriverManager}

object MusicCountFilter {

  def main(args: Array[String]) = {
    //请将下方链接中myname替换为本人姓名全拼
//    hdfs://master:9864/user/myname/streaming_music/
    val streamingpath = "hdfs://master:9864/user/myname/streaming_music"
    val targetpath: String = "hdfs://master:9864/user/myname/output_music/"
    val hdfsconf: Configuration = new Configuration();
    hdfsconf.set("fs.defaultFS", "hdfs://master:9864");
    System.setProperty("HADOOP_USER_NAME", "root");
    val hdfsfs: FileSystem = FileSystem.get(hdfsconf);
    var i = 0;
    val ifiltermax = 100;
    var strtar: String = targetpath.+("result").+(i.toString)

    val sparkconf = new SparkConf().setMaster("local[2]").setAppName("musiccount")
    val ssc = new StreamingContext(sparkconf, Seconds(10))
    ssc.sparkContext.setLogLevel("ERROR") //设置日志等级，减小日志输出

    println("streamingpath:" + streamingpath)
    val lines = ssc.textFileStream(streamingpath)
//  监听hdfs目录，请确保windows时间和hdfs服务时间同步，否则监听不到数据
//    lines.foreachRDD { rdd =>
//      rdd.foreach(println)
//    }
```

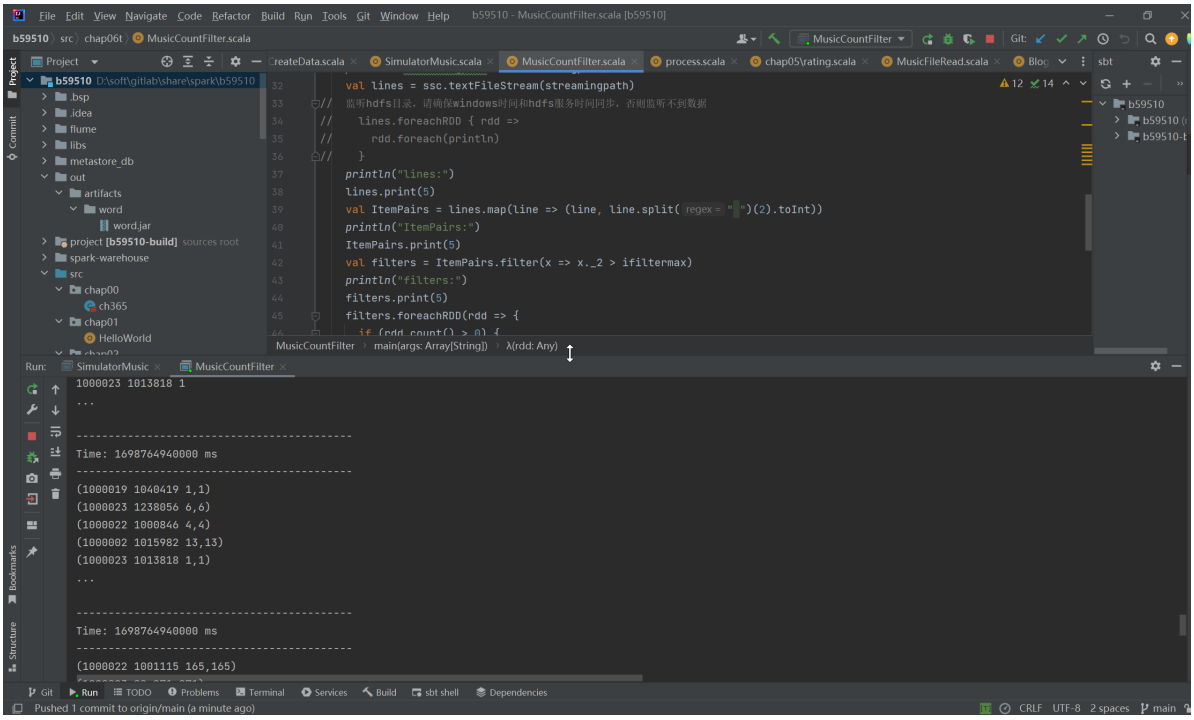
```
    println("lines:")
    lines.print(5)
    val ItemPairs = lines.map(line => (line, line.split(" ")(2).toInt))
    println("ItemPairs:")
    ItemPairs.print(5)
    val filters = ItemPairs.filter(x => x._2 > ifiltermax)
    println("filters:")
    filters.print(5)
    filters.foreachRDD(rdd => {
      if (rdd.count() > 0) {
        hdfsfs.delete(new Path(strtar), true)
        rdd.map(_._1).repartition(1).saveAsTextFile(strtar)
        i += 1
        strtar = targetpath.+("result").+(i.toString)
        println("strtar: " + strtar)
      }
    }
    )

    ssc.start()
    ssc.awaitTermination()
    ssc.stop()
  }

}

```

运行结果

模拟程序运行结果

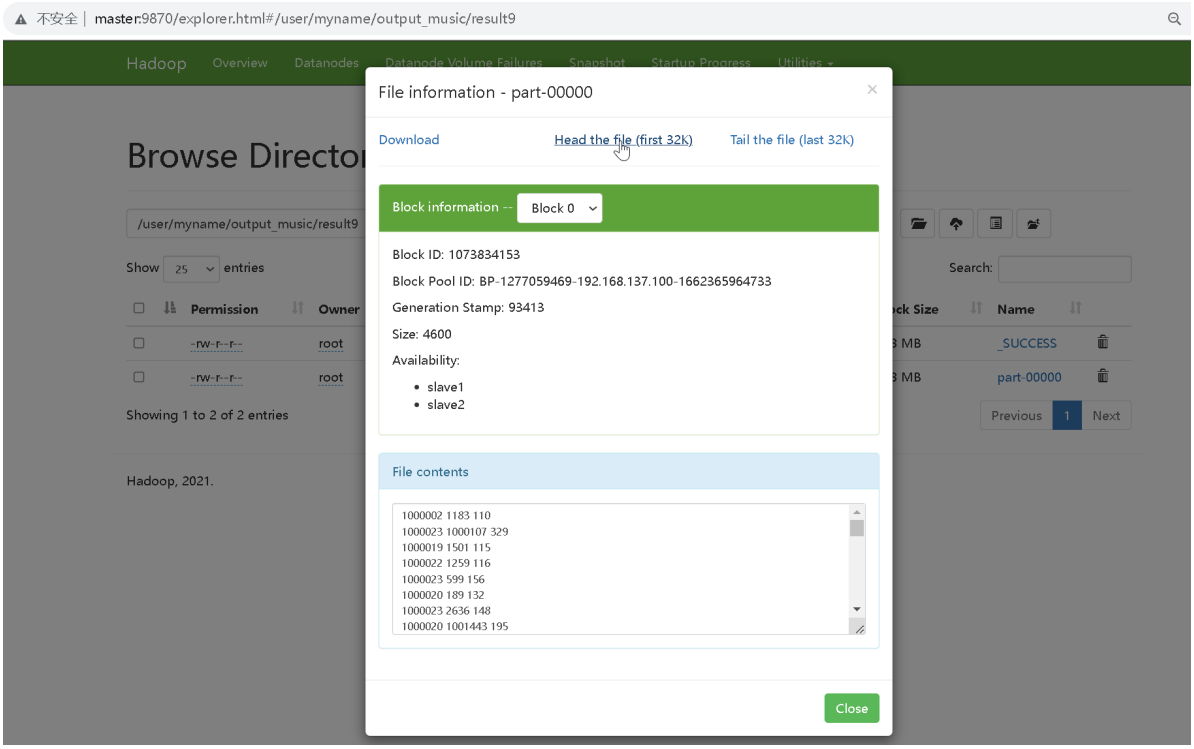


分析程序运行结果



HDFS结果文件

目录：hdfs://master:9864/user/myname/output_music/



CH06实训2：使用使用Spark Streaming实现课程实时查找

训练要点

1. 掌握使用socketTextStream连接端口获取数据源的方法。
2. 掌握DStream的转换操作

需求说明

某高校为大数据相关专业的学生开设了多门课程，为了能够实时地查找出目标课程，
需要在IntelliJ IDEA中使用spark Streaming编程实现从一台服务器的8888端口上接收课程数据，课程数据需手动在服务器的8888端口输人，输人的课程数据如表6一7所示，每一条数据有2个字段，分别表示课程编号和课程名称，以空格分隔。现目标课程是"Hadoop" 和"Spark"，需要查询两门课程及对应课程编号。

输入的课程数据：

---

121 Hadoop
123 Java
069 HBase
223 Spark
078 Hive

实现思路及步骤

1. 在IntelliJ IDEA中配置好Spark Streaming开发环境.
2. 启动IntelliJ IDEA，并进行Spark Streaming编程.
3. 在一台服务器（master节点）中查看是否安装了nc软件，若没有安装nc软件，则先安装nc软件.
4. 在master节点上用nc启动8888端口.
5. 在IntelliJ IDEA中使用socketTextStream监听8888端口，获取数据.
6. 使用map()方法将每一条数据以空格分割，并转化成“（课程名称，课程编号）"的形式.
7. 创建数组，将要查找的"Hadoop"和"Spark"两门课程标记为true,形式如:“ ("Hadoop",true)"，并使用parallelize把数组转化成RDD.
8. 使用left0uterJoin()方法对步骤(6)得到的RDD数据与步骤 (7）得到的RDD数据进行左外连接，最终形成“（课程名称，（课程编号，true））"形式的数据.
9. 使用getOrElse函数判断数据是否含有"true"字段，并使用filter把含有"true" 的数据筛选出来

作业要求

1. 以截图提供nc -l 8888运行的截图
2. 截图在IDEA上运行分析程序OnlineBlackListFilter的截图，包括源码和控制台运行结果截图

实现参考

在master主机启动nc程序

```
yum -y install nc
nc -l 8888
```

在master主机的nc中输入消息

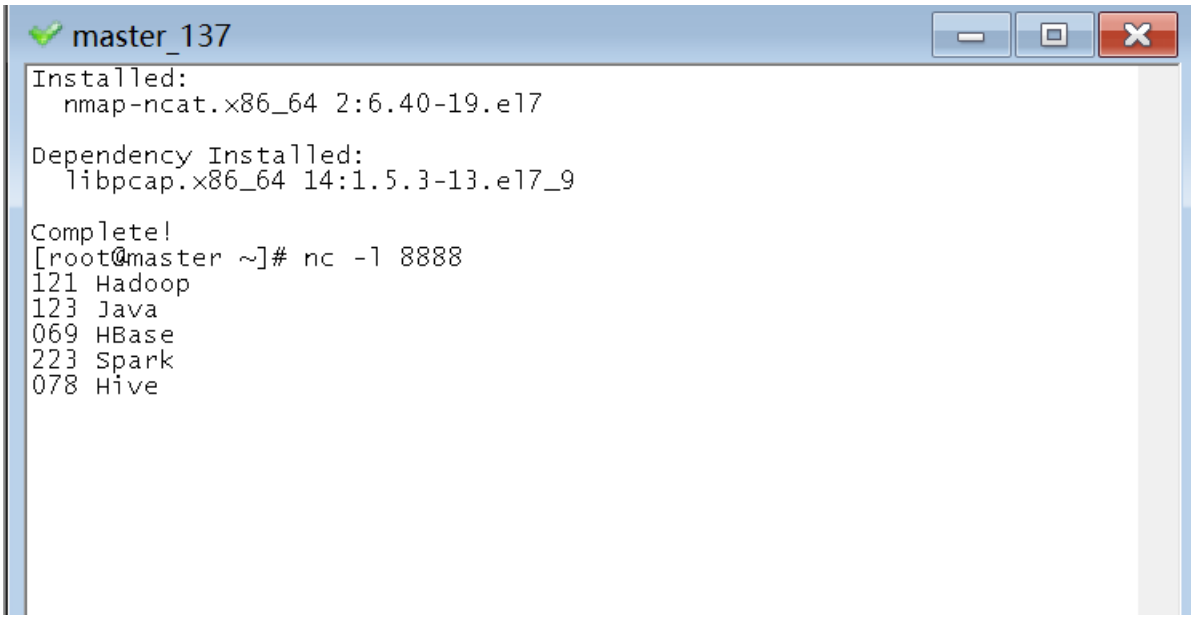
```
121 Hadoop
123 Java
069 HBase
223 Spark
078 Hive
```

分析程序OnlineBlackListFilter

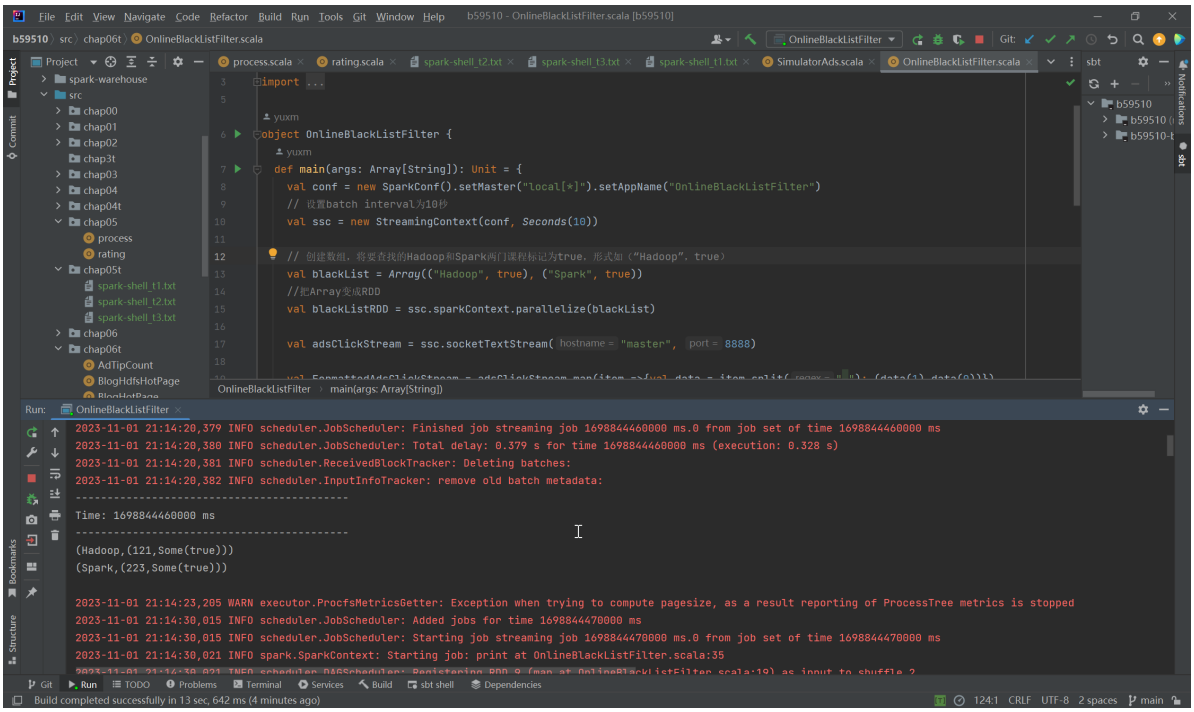
```
package chap06t

import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}

object OnlineBlackListFilter {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setMaster("local[*]").setAppName("OnlineBlackListFilter")
    // 设置batch interval为10秒
    val ssc = new StreamingContext(conf, Seconds(10))

    // 创建数组，将要查找的Hadoop和Spark两门课程标记为true，形式如（“Hadoop”，true）
    val blackList = Array(("Hadoop", true), ("Spark", true))
    //把Array变成RDD
    val blackListRDD = ssc.sparkContext.parallelize(blackList)

    val adsClickStream = ssc.socketTextStream("master", 8888)

    val FormattedAdsClickStream = adsClickStream.map(item =>{val data = item.split(" "); (data(1),data(0))})

    val validateAds = FormattedAdsClickStream.transform(userClickRDD => {
        // 左连接
      val joinedBlackListRDD = userClickRDD.leftOuterJoin(blackListRDD)

       //使用getOrElse判断数据是否含有“true”字段，并使用filter把含有“true”的数据筛选出来。
      joinedBlackListRDD.filter(joinedItem =>
        if (joinedItem._2._2.getOrElse(false)) {
          true
        } else {
          false
        }
      )
    })

    validateAds.print()

    ssc.start()
    ssc.awaitTermination()
//    ssc.stop()
  }

}

```

运行结果

master主机上nc运行结果



Idea上OnlineBlackListFilter运行结果



CH06实训3：使用Spark Streaming实时统计广告点击量前3名

训练要点

1. 掌握使用spark Streaming从文件系统中读取数据
2. 掌握DStream的转换操作
3. 掌握DStream的输出操作

需求说明

在现代商业社会中，依靠广告，利用先进的媒介与传播技术，加强产、供、销之间的信息联系，传播商情，对实现国民经济高质量发展越来越重要。现有一份2020年9月某 10个省份的广告数据文件agent.log，记录了用户对某些广告的访问信息，部分数据如表6．8 所示，文件共有5列数据，分别代表时间戳、省份ID、城市ID、用户ID和广告ID

用户对广告的访问信息部分数据

16609143867 6 7 64 16

1660914386994 75 18

16609143869 1 7 87 12

166091438692 8 929

16609143869 6 7 84 24

16609143869 1 8 95 5

16609143869 8 1 90 29

16609143869 3 3 36 16

16609143869 3 3 54 22

166091438697 6 33 5

为了研究不同省份在2020年9月的广告点击量情况，现要求使用spark Streaming实时统计每个省份的广告点击量前3名，输出结果为“（（省份ID,((广告A,sum),(广告B,sum),(广告C,sum)))"的形式。广告A、广告B、广告c分别代表该省份的前3名广告对应的广告ID, sum表示该省份的前3名广告对应的点击量

实现思路及步骤

1. 编写模拟器代码。在一台服务器中编写脚本，实现在一个文件夹中批量生成文件，并从agent.log文件中随机抽取200条数据送至每一个生成的文件中
2. 在脚本中实现将生成的文件上传到HDFS上。
3. 启动spark-shell
4. 在spark-shell中创建spark Streaming连接对象，并设置窗口长度为10秒。
5. 使用textFileStream()方法监听HDFS上的文件目录，获取数据
6. 使用map()方法将每一个文件的每一条数据以空格分割，并转化成“（（省份ID,广告(D),I)"的形式。
7. 使用reduceByKey()方法对转换后的数据进行分组聚合，转化成“（（省份ID,广告 (D),sum)"的形式。
8. 使用map()方法对聚合的结果进行结构的转换，转化成“（省份ID,(广告ID,sum))" 的形式。
9. 使用groupByKey()方法将转换结构后的数据根据省份进行分组，转化成“（省份 ID,((广告A,sum),(广告B,sum),(广告C,sum)))"的形式。
10. 使用mapvalues()方法和sortBy()方法对分组后的数据进行组内的降序排序，取前3名

作业要求

1. 以截图提供hdfs上/user/myname/目录列表，包含有文件agent.log的截图
2. 截图在IDEA上运行分析程序AdTipCount的截图，包括源码和控制台运行结果截图

实现参考

测试数据准备

```
cd /root/spark
rm -f spark_t_data.tar.gz*
wget https://biglab.site//b59510spark/file/spark_t_data.tar.gz
tar -xzvf ./spark_t_data.tar.gz
```

测试文件上传

```
cd /root/spark
hdfs dfs -put ./spark_t_data/agent.log /user/myname/
hdfs dfs -ls /user/myname/agent.log
```

HDFS上测试路径准备

在master或slave上执行

```
hdfs dfs -chmod -R 777
hdfs dfs -rm -r /user/myname/streaming_ad
hdfs dfs -mkdir -p /user/myname/streaming_ad/

```

模拟程序SimulatorAds

```
package chap06t

import org.apache.hadoop.fs.FileSystem
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.fs.{FileSystem, FileUtil, Path}

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.rdd.RDD

import java.text.SimpleDateFormat
import java.util.Date
import scala.collection.mutable.ListBuffer
object SimulatorAds {
  def main(args: Array[String]) {
    var i = 0
    var iFileMaxLine = 200
    val hdfsconf: Configuration = new Configuration();
    hdfsconf.set("fs.defaultFS", "hdfs://master:9864");
    System.setProperty("HADOOP_USER_NAME", "root");
    val hdfsfs: FileSystem = FileSystem.get(hdfsconf);
    val sparkconf = new
SparkConf().setAppName("simulator").setMaster("local")
```

```
    val sc = new SparkContext(sparkconf)
    sc.setLogLevel("ERROR")
    val filename = "hdfs://master:9864/user/myname/agent.log"
    val lines = sc.textFile(filename).collect().toList
    val filerow = lines.length
    val tmprows: ListBuffer[String] = ListBuffer()
    //请将下方链接中myname替换为本人姓名全拼
    val outputtargetpath = "hdfs://master:9864/user/myname/streaming_ad/"
    while (true) {
      val streamingpath = outputtargetpath + "/streamingdata_" + i
      hdfsfs.delete(new Path(streamingpath), true)
      i = i + 1
      var j = 0
      while (j < iFileMaxLine) {
        val linstr = lines(index(filerow)).toString
        tmprows.+=(linstr)
        println(linstr)
        j = j + 1
      }
      val linelog: RDD[String] = sc.parallelize(tmprows)
      linelog.repartition(1).saveAsTextFile(streamingpath)
      tmprows.clear()
      Thread sleep 1000
      log(getNowTime(), streamingpath + " generated")
    }
  }

  def log(date: String, message: String) = {
    println(date + "----" + message)
  }

  def index(length: Int) = {
    import java.util.Random
    val rdm = new Random
    rdm.nextInt(length)
  }

  def getNowTime(): String = {
    val now: Date = new Date()
    val datetimeFormat: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss")
    val ntime = datetimeFormat.format(now)
    ntime
  }
}

```

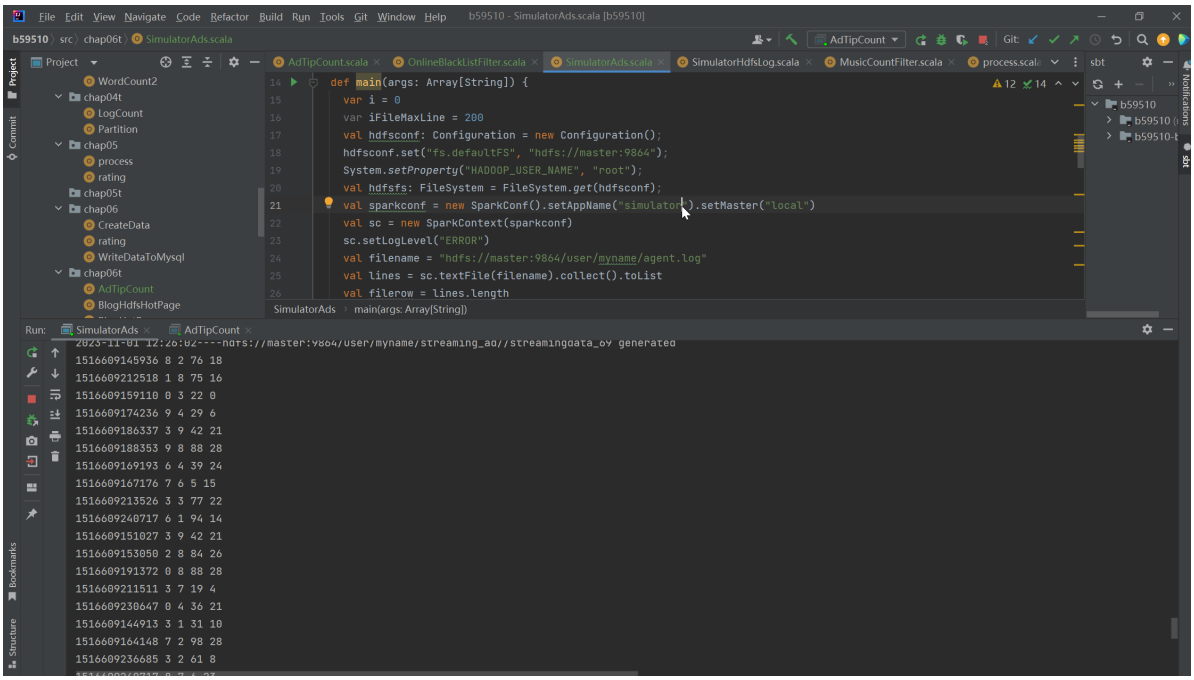
分析程序AdTipCount

```
package chap06t

import org.apache.spark.SparkConf
import org.apache.spark.streaming.dstream.DStream
import org.apache.spark.streaming.{Seconds, StreamingContext}
object AdTipCount {
```

```scala
  def main(args: Array[String]): Unit = {
    val streamingpath = "hdfs://master:9864/user/myname/streaming_ad/"
    val sc = new SparkConf().setMaster("local[2]").setAppName("AdTipCount")
    // 第二个参数表示批量处理的周期（采集数据的周期）
    val ssc = new StreamingContext(sc, Seconds(10))
    // 设置日志级别
    ssc.sparkContext.setLogLevel("ERROR") //ERROR,WARN
    //    1.获取原始数据： 时间戳， 省份， 城市， 用户， 广告
    val lines = ssc.textFileStream(streamingpath)
    //   2.将原始数据进行结构转换。方便统计
    //   时间戳， 省份， 城市， 用户， 广告  =》 （（省份， 广告）， 1）
    val map = lines.map(
      line => {
        val datas = line.split(" ")
        ((datas(1), datas(4)), 1)
      }
    )

    //    3.将转换结构后的数据，进行分组聚合
    //      （（省份， 广告）， 1） =》 （（省份， 广告）， sum）
    val reduce: DStream[((String, String), Int)] = map.reduceByKey(_ + _)

    //    4.将聚合的结果进行结构的转换
    //     （（省份， 广告）， sum） =》 （省份， （广告， sum））
    val newMap: DStream[(String, (String, Int))] = reduce.map {
      case ((prv, ad), sum) => {
        (prv, (ad, sum))
      }
    }

    //    5.将转换结构后的数据根据省份进行分组
    //     （省份， 【（广告A， sumA），（广告B， sumB）】）
    val group = newMap.groupByKey()

    //    6.将分组后的数据组内排序（降序），取前3名
    val result: DStream[(String, List[(String, Int)])] = group.mapValues(
      iter => {
        iter.toList.sortBy(_._2)(Ordering.Int.reverse).take(3)
      }
    )
    result.print()
    ssc.start()
    ssc.awaitTermination()
    ssc.stop()
  }
}

```

运行结果

模拟程序运行结果



分析程序运行结果