

第6章Spark Streaming实训

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

需求说明

实现步骤

作业要求

实现参考

测试数据准备

HDFS上测试路径准备

模拟程序**SimulatorMusic**

分析程序MusicCountFilter

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

CH06实训2：使用使用Spark Streaming实现课程实时查找

训练要点

需求说明

实现思路及步骤

作业要求

实现参考

在master主机启动nc程序

分析程序OnlineBlackListFilter

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

需求说明

实现思路及步骤

作业要求

实现参考

测试数据准备

测试文件上传

HDFS上测试路径准备

模拟程序SimulatorAds

分析程序AdTipCount

运行结果

模拟程序运行结果

分析程序运行结果

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

```
1 | hdfs dfs -chmod -R 777
2 | hdfs dfs -mkdir -p /user/myname/tmp/
3 | cd /root/spark/
4 | wget http://biglab.site//b59510spark/file/user_artist_data.tar.gz
5 | tar -xzvf ./user_artist_data.tar.gz
6 | hdfs dfs -put ./user_artist_data.txt /user/myname/
7 |
8 | hdfs dfs -rm -r /user/myname/tmp/streaming
9 | hdfs dfs -rm -r /user/myname/output_music
10 | hdfs dfs -rm -r /user/myname/streaming_music
```

模拟程序 SimulatorMusic

```
1 package chap06t
2
3 import org.apache.hadoop.conf.Configuration
4 import org.apache.hadoop.fs.{FileSystem, FileUtil, Path}
5
6 import org.apache.spark.{SparkConf, SparkContext}
7 import org.apache.spark.rdd.RDD
8
9 import java.text.SimpleDateFormat
10 import java.util.Date
11 import scala.collection.mutable.ListBuffer
12 import scala.io.Source
13 object SimulatorMusic {
14   def main(args: Array[String]) {
15     var i=0
16     var iFileMaxLine=1000
17     val hdfsconf: Configuration = new Configuration();
18     hdfsconf.set("fs.defaultFS", "hdfs://master:9864");
19     System.setProperty("HADOOP_USER_NAME", "root");
20     val hdfsfs: FileSystem = FileSystem.get(hdfsconf);
21     val sparkconf = new
22     SparkConf().setAppName("simulator").setMaster("local")
23     val sc = new SparkContext(sparkconf)
24     // val filename="D:\\tmp\\user_artist_data.txt"
25     // val lines = Source.fromFile(filename).getLines.toList
26     val filename="hdfs://master:9864/user/myname/user_artist_data.txt"
27     val lines = sc.textFile(filename).collect().toList
28     val filerow = lines.length
29     val tmprows:ListBuffer[String]= ListBuffer()
30     //请将下方链接中myname替换为本人姓名全拼
31     val outputtargetpath="hdfs://master:9864/user/myname/streaming_music/"
32     while (true)
33     {
34       val streamingpath=outputtargetpath+"/streamingdata_" +i
35       hdfsfs.delete(new Path(streamingpath),true)
36       i=i+1
37       var j=0
38       while(j<iFileMaxLine)
39       {
40         val linstr=lines(index(filerow)).toString
41         tmprows.+=(linstr)
42         println(linstr)
43         j=j+1
44       }
45       val lineLog: RDD[String] = sc.parallelize(tmprows)
46       lineLog.repartition(1).saveAsTextFile(streamingpath)
47       tmprows.clear()
48       Thread.sleep(1000)
49       log(getNowTime(),streamingpath+" generated")
50     }
51   }
52   def log(date: String, message: String) = {
53     println(date + "----" + message)
54   }
55 }
```

```

53     }
54     def index(length: Int) = {
55         import java.util.Random
56         val rdm = new Random
57         rdm.nextInt(length)
58     }
59     def getNowTime():String={
60         val now:Date = new Date()
61         val dateTimeFormat:SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss")
62         val ntime = dateTimeFormat.format( now )
63         ntime
64     }
65 }
66

```

分析程序MusicCountFilter

```

1  package chap06t
2
3  import org.apache.hadoop.fs.Path
4  import org.apache.spark.{SparkConf, SparkContext}
5  import org.apache.spark.streaming.{Seconds, StreamingContext}
6  import org.apache.hadoop.conf.Configuration
7  import org.apache.hadoop.fs.{FileSystem, FileUtil, Path}
8  import org.apache.spark.repl.Main.sparkContext
9
10 import java.sql.{Connection, DriverManager}
11
12 object MusicCountFilter {
13
14     def main(args: Array[String]) = {
15         //请将下方链接中myname替换为本人姓名全拼
16         // hdfs://master:9864/user/myname/streaming_music/
17         val streamingpath = "hdfs://master:9864/user/myname/streaming_music"
18         val targetpath: String = "hdfs://master:9864/user/myname/output_music/"
19         val hdfsconf: Configuration = new Configuration();
20         hdfsconf.set("fs.defaultFS", "hdfs://master:9864");
21         System.setProperty("HADOOP_USER_NAME", "root");
22         val hdfsfs: FileSystem = FileSystem.get(hdfsconf);
23         var i = 0;
24         val ifiltermax = 100;
25         var strtar: String = targetpath.+("result").+(i.toString)
26
27         val sparkconf = new
SparkConf().setMaster("local[2]").setAppName("musiccount")
28         val ssc = new StreamingContext(sparkconf, Seconds(10))
29         ssc.sparkContext.setLogLevel("ERROR") //设置日志等级, 减小日志输出
30
31         println("streamingpath:" + streamingpath)
32         val lines = ssc.textFileStream(streamingpath)
33         // 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
34         // lines.foreachRDD { rdd =>
35         //     rdd.foreach(println)
36         // }

```

```

37     println("lines:")
38     lines.print(5)
39     val ItemPairs = lines.map(line => (line, line.split(" ")(2).toInt))
40     println("ItemPairs:")
41     ItemPairs.print(5)
42     val filters = ItemPairs.filter(x => x._2 > ifiltermax)
43     println("filters:")
44     filters.print(5)
45     filters.foreachRDD(rdd => {
46         if (rdd.count() > 0) {
47             hdfsfs.delete(new Path(strtar), true)
48             rdd.map(_._1).repartition(1).saveAsTextFile(strtar)
49             i += 1
50             strtar = targetpath.+("result").+(i.toString)
51             println("strtar: " + strtar)
52         }
53     }
54 )
55
56 ssc.start()
57 ssc.awaitTermination()
58 ssc.stop()
59 }
60
61 }
62

```

运行结果

模拟程序运行结果

```

Run: SimulatorMusic > MusicCountFilter >
1000002 1330 2
1000019 1045783 1
1000022 1001365 9
1000019 1000427 16
1000022 1091409 1
1000022 2132348 1
1000022 1001297 1
1000019 1000422 4
1000028 1000323 4
1000002 1262670 2
1000019 1004402 1
1000022 1137805 1
1000028 1001779 2
1000025 599 9
1000029 1001055 2
1000022 352 19
1000028 793 143

```

分析程序运行结果

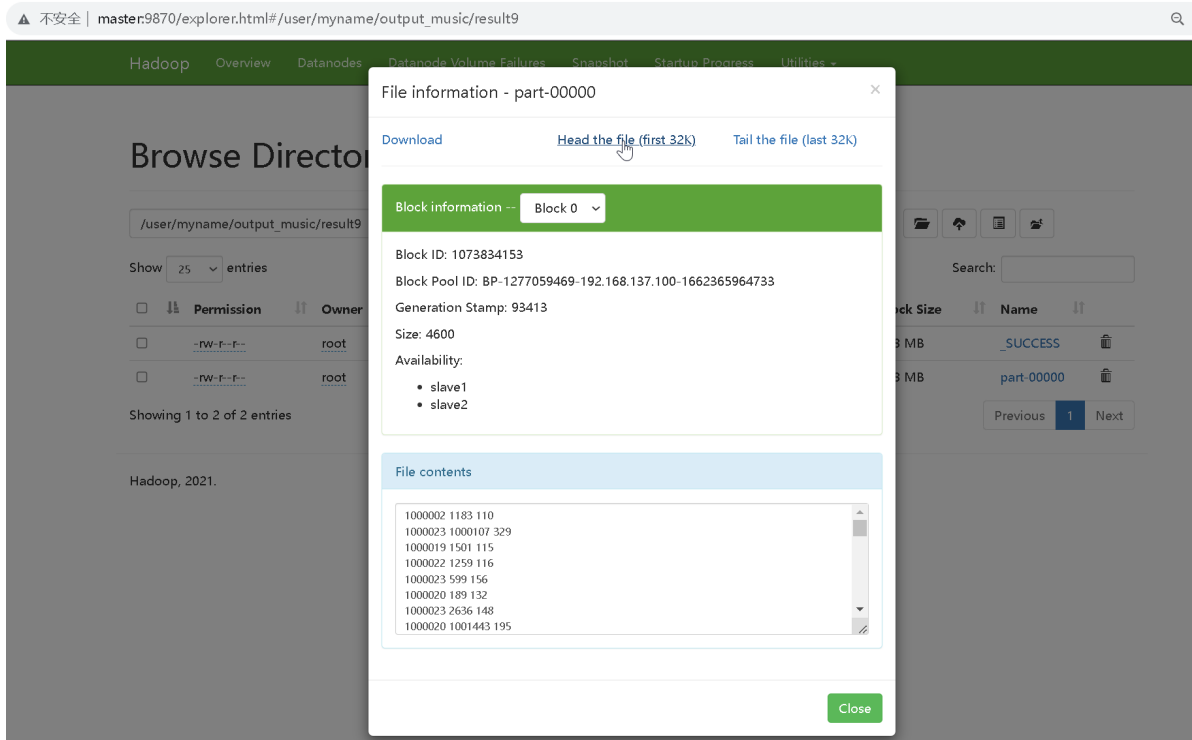
```

b59510 - MusicCountFilter.scala [b59510]
MusicCountFilter
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) {
    // ...
  }
})
}
main(args: Array[String]) { lambda(rdd: Any)
}
Run: SimulatorMusic x MusicCountFilter x
1000023 1013818 1
...
-----
Time: 1698764940000 ms
-----
(1000019 1040419 1,1)
(1000023 1238056 6,6)
(1000022 1000846 4,4)
(1000002 1015982 13,13)
(1000023 1013818 1,1)
...
-----
Time: 1698764940000 ms
-----
(1000022 1001115 165,165)

```

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端口输入，输入的课程数据如表6—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leftOuterJoin()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

```
1 | yum -y install nc
2 | nc -l 8888
```

在master主机的nc中输入消息

```
1 121 Hadoop
2 123 Java
3 069 HBase
4 223 Spark
5 078 Hive
```

分析程序OnlineBlackListFilter

```
1 package chap06t
2
3 import org.apache.spark.SparkConf
4 import org.apache.spark.streaming.{Seconds, StreamingContext}
5
6 object OnlineBlackListFilter {
7     def main(args: Array[String]): Unit = {
8         val conf = new
9 SparkConf().setMaster("local[*]").setAppName("OnlineBlackListFilter")
10        // 设置batch interval为10秒
11        val ssc = new StreamingContext(conf, Seconds(10))
12
13        // 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)
14        val blacklist = Array(("Hadoop", true), ("Spark", true))
15        //把Array变成RDD
16        val blacklistRDD = ssc.sparkContext.parallelize(blacklist)
17
18        val adsclickstream = ssc.socketTextStream("master", 8888)
19
20        val formattedadsclickstream = adsclickstream.map(item =>{val data =
21 item.split(" "); (data(1),data(0))})
22
23        val validateads = formattedadsclickstream.transform(userclickrdd => {
24            // 左连接
25            val joinedblacklistrdd = userclickrdd.leftOuterJoin(blacklistRDD)
26
27            //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
28            //选出来。
29            joinedblacklistrdd.filter(joineditem =>
30                if (joineditem._2._2.getOrElse(false)) {
31                    true
32                } else {
33                    false
34                }
35            )
36        })
37
38        validateads.print()
39
40        ssc.start()
41        ssc.awaitTermination()
42        // ssc.stop()
43    }
44 }
```


运行结果

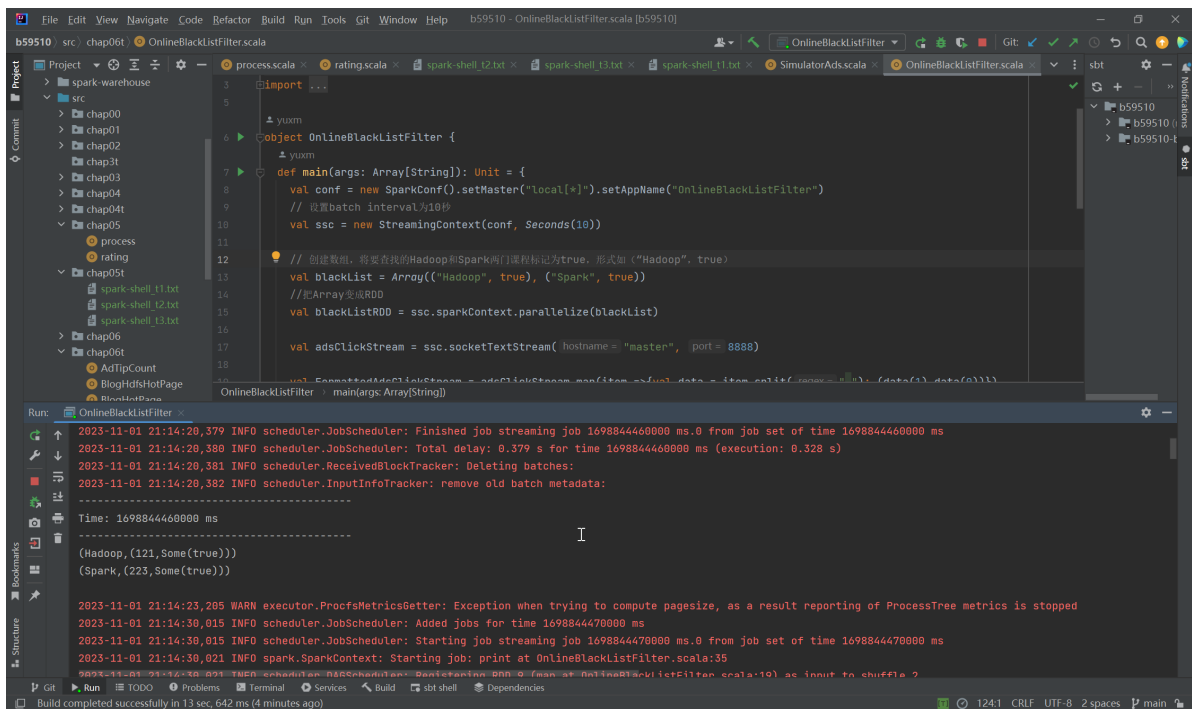
master主机上nc运行结果

```
master_137
Installed:
  nmap-ncat.x86_64 2:6.40-19.e17

Dependency Installed:
  libpcap.x86_64 14:1.5.3-13.e17_9

Complete!
[root@master ~]# nc -l 8888
121 Hadoop
123 Java
069 HBase
223 Spark
078 Hive
```

Idea上OnlineBlackListFilter运行结果



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help b59510 - OnlineBlackListFilter.scala [b59510]
Project: spark-warehouse
src
  chap00
  chap01
  chap02
  chap03
  chap04
  chap04t
  chap05
    process
    rating
    spark-shell_11.txt
    spark-shell_12.txt
    spark-shell_13.txt
  chap06
  chap06t
    AdTipCount
    BlogHitsHotPage
    BlogHitsPage
    BlogHitsPage
  OnlineBlackListFilter
    OnlineBlackListFilter.scala
Run: OnlineBlackListFilter
2023-11-01 21:14:20,379 INFO scheduler.JobScheduler: Finished job streaming job 1698844460000 ms.0 from job set of time 1698844460000 ms
2023-11-01 21:14:20,380 INFO scheduler.JobScheduler: Total delay: 0.379 s for time 1698844460000 ms (execution: 0.328 s)
2023-11-01 21:14:20,381 INFO scheduler.ReceivedBlockTracker: Deleting batches:
2023-11-01 21:14:20,382 INFO scheduler.InputInfoTracker: remove old batch metadata:
-----
Time: 1698844460000 ms
-----
(Hadoop, (121, Some(true)))
(Spark, (223, Some(true)))

2023-11-01 21:14:23,205 WARN executor.ProcfsMetricsGetter: Exception when trying to compute pagesize, as a result reporting of ProcessTree metrics is stopped
2023-11-01 21:14:30,015 INFO scheduler.JobScheduler: Added jobs for time 1698844470000 ms
2023-11-01 21:14:30,015 INFO scheduler.JobScheduler: Starting job streaming job 1698844470000 ms.0 from job set of time 1698844470000 ms
2023-11-01 21:14:30,021 INFO spark.SparkContext: Starting job: print at OnlineBlackListFilter.scala:35
2023-11-01 21:14:30,021 INFO spark.SparkContext: Starting job: print at OnlineBlackListFilter.scala:35
Build completed successfully in 13 sec, 642 ms (4 minutes ago)
```

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

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

测试文件上传

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

HDFS上测试路径准备

在master或slave上执行

```
1 | hdfs dfs -chmod -R 777
2 | hdfs dfs -rm -r /user/myname/streaming_ad
3 | hdfs dfs -mkdir -p /user/myname/streaming_ad/
4 |
```

模拟程序SimulatorAds

```
1 | package chap06t
2 |
3 | import org.apache.hadoop.fs.FileSystem
4 | import org.apache.hadoop.conf.Configuration
5 | import org.apache.hadoop.fs.{FileSystem, FileUtil, Path}
6 |
7 | import org.apache.spark.{SparkConf, SparkContext}
8 | import org.apache.spark.rdd.RDD
9 |
10 | import java.text.SimpleDateFormat
11 | import java.util.Date
12 | import scala.collection.mutable.ListBuffer
13 | object SimulatorAds {
14 |   def main(args: Array[String]) {
15 |     var i = 0
16 |     var iFileMaxLine = 200
17 |     val hdfsconf: Configuration = new Configuration();
18 |     hdfsconf.set("fs.defaultFS", "hdfs://master:9864");
19 |     System.setProperty("HADOOP_USER_NAME", "root");
20 |     val hdfsfs: FileSystem = FileSystem.get(hdfsconf);
21 |     val sparkconf = new
SparkConf().setAppName("simulator").setMaster("local")
```

```

22     val sc = new SparkContext(sparkconf)
23     sc.setLogLevel("ERROR")
24     val filename = "hdfs://master:9864/user/myname/agent.log"
25     val lines = sc.textFile(filename).collect().toList
26     val filerow = lines.length
27     val tmprows: ListBuffer[String] = ListBuffer()
28     //请将下方链接中myname替换为本人姓名全拼
29     val outputtargetpath = "hdfs://master:9864/user/myname/streaming_ad/"
30     while (true) {
31         val streamingpath = outputtargetpath + "/streamingdata_" + i
32         hdfsfs.delete(new Path(streamingpath), true)
33         i = i + 1
34         var j = 0
35         while (j < iFileMaxLine) {
36             val linstr = lines(index(filerow)).toString
37             tmprows.+=(linstr)
38             println(linstr)
39             j = j + 1
40         }
41         val lineLog: RDD[String] = sc.parallelize(tmprows)
42         lineLog.repartition(1).saveAsTextFile(streamingpath)
43         tmprows.clear()
44         Thread.sleep(1000)
45         log(getNowTime(), streamingpath + " generated")
46     }
47 }
48
49 def log(date: String, message: String) = {
50     println(date + "----" + message)
51 }
52
53 def index(length: Int) = {
54     import java.util.Random
55     val rdm = new Random
56     rdm.nextInt(length)
57 }
58
59 def getNowTime(): String = {
60     val now: Date = new Date()
61     val dateTimeFormat: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss")
62     val ntime = dateTimeFormat.format(now)
63     ntime
64 }
65 }
66

```

分析程序AdTipCount

```

1 package chap06t
2
3 import org.apache.spark.SparkConf
4 import org.apache.spark.streaming.dstream.DStream
5 import org.apache.spark.streaming.{Seconds, StreamingContext}
6 object AdTipCount {

```

```

7   def main(args: Array[String]): Unit = {
8       val streamingpath = "hdfs://master:9864/user/myname/streaming_ad/"
9       val sc = new SparkConf().setMaster("local[2]").setAppName("AdTipCount")
10      // 第二个参数表示批量处理的周期（采集数据的周期）
11      val ssc = new StreamingContext(sc, Seconds(10))
12      // 设置日志级别
13      ssc.sparkContext.setLogLevel("ERROR") //ERROR,WARN
14      // 1.获取原始数据： 时间戳， 省份， 城市， 用户， 广告
15      val lines = ssc.textFileStream(streamingpath)
16      // 2.将原始数据进行结构转换。方便统计
17      // 时间戳， 省份， 城市， 用户， 广告 => ((省份， 广告)， 1)
18      val map = lines.map(
19          line => {
20              val datas = line.split(" ")
21              ((datas(1), datas(4)), 1)
22          }
23      )
24
25      // 3.将转换结构后的数据，进行分组聚合
26      // ((省份， 广告)， 1) => ((省份， 广告)， sum)
27      val reduce: DStream[((String, String), Int)] = map.reduceByKey(_ + _)
28
29      // 4.将聚合的结果进行结构的转换
30      // ((省份， 广告)， sum) => (省份， (广告， sum))
31      val newMap: DStream[(String, (String, Int))] = reduce.map {
32          case ((prv, ad), sum) => {
33              (prv, (ad, sum))
34          }
35      }
36
37      // 5.将转换结构后的数据根据省份进行分组
38      // (省份， [(广告A, sumA), (广告B, sumB)])
39      val group = newMap.groupByKey()
40
41      // 6.将分组后的数据组内排序（降序），取前3名
42      val result: DStream[(String, List[(String, Int)])] = group.mapValues(
43          iter => {
44              iter.toList.sortBy(_._2)(Ordering.Int.reverse).take(3)
45          }
46      )
47      result.print()
48      ssc.start()
49      ssc.awaitTermination()
50      ssc.stop()
51  }
52 }
53

```

运行结果

模拟程序运行结果

```
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  
}
```

Run: SimulatorAds - AdTipCount

```
2023-11-01 12:20:02---nosr://master:9864/user/myname/streaming_ad/streamingdata_ov generated  
1516609145936 8 2 76 18  
1516609212518 1 8 75 16  
1516609159110 0 3 22 0  
1516609174236 9 4 29 6  
1516609186337 3 9 42 21  
1516609188353 9 8 88 28  
1516609169193 6 4 39 24  
1516609167176 7 6 5 15  
1516609213526 3 3 77 22  
1516609240717 6 1 94 14  
1516609151027 3 9 42 21  
1516609153050 2 8 84 26  
1516609191372 0 8 88 28  
1516609211511 3 7 19 4  
1516609230647 0 4 36 21  
1516609144913 3 1 31 10  
1516609164148 7 2 98 28  
1516609236685 3 2 61 8
```

分析程序运行结果

```
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  
}
```

Run: SimulatorAds - AdTipCount

```
-----  
(4,List((21,10), (2,9), (4,9)))  
(8,List((12,16), (20,16), (11,13)))  
(6,List((9,11), (8,11), (14,10)))  
(0,List((4,12), (9,11), (28,11)))  
(2,List((28,12), (6,10), (15,9)))  
(7,List((16,17), (4,12), (22,11)))  
(5,List((18,12), (25,9), (16,8)))  
(9,List((21,14), (28,11), (0,9)))  
(3,List((25,11), (14,11), (6,11)))  
(1,List((28,11), (6,9), (23,9)))  
-----  
Time: 1698812650000 ms  
-----  
(4,List((1,14), (12,10), (17,10)))  
(8,List((13,11), (19,10), (11,10)))  
(6,List((7,8), (21,8), (10,8)))  
(0,List((2,14), (24,11), (13,9)))
```

Build completed successfully in 1 sec, 218 ms (2 minutes ago)