第6章Spark Streaming

(源自:https://biglab.site)

(版本:Ver1.5-20230828)

学习目标

- 了解Spark Streaming的基本概念及运行原理
- 了解DStream编程模型的概念
- 了解DStream的转换操作
- 了解DStream的窗口操作
- 了解DStream的输出操作

任务背景

书籍是人类进步的阶梯，数字时代的来临，也催生出“书”的新形式，即电子书。

同时，众多售书的电商平台也应运而生。电商平台想要在激烈的竞争中脱颖而出，需要更着重于改善用户体验，并增加用户的黏性，把更多更好的书推荐给读者，扩大他们的知识视野。

用户无法找到适宜的书籍时往往会相信大众的选择，选择购买热度较高的书籍。

基于这种情况，电商平台可以根据现有书籍的评分、销量、用户的评分次数等信息构建书籍热度，将一些热度较高的书推荐给用户，进而改善用户体验，增加用户黏性，激发用户的购买欲。

书籍热度的计算可以根据下式进行，其中，u表示用户的平均评分，x表示用户的评分次数，y表示书籍的平均评分，z表示书籍被评分的次数

$$f(u,x,y,z)=0.3ux+yz$$

目前已采集了某电商网站上用户对书籍的评分数据文件BookRating.txt，数据字段说明如下表。其中Rating字段中评分范围为1～5分。

字段名称	说明
UserID	用户ID
BookID	书籍ID
Rating	用户对书籍的评分

实时计算书籍热度后，可以将热度最高的10本图书的评分数据保存在Hive数据库中，因此需要在Hive数据库中设计一个表，用于保存热度最高的10本图书的评分数据。

Spark会将DataFrame写入Hive并根据DataFrame自动创建表。

为模拟实时数据的流式计算，本章将使用Spark Streaming框架实现书籍评分实时计算分析。

本章任务如下。

1. 首先介绍Spark Streaming基本概念及运行原理；
2. 再详细介绍Spark Streaming框架的DStream编程模型及其基础操作；

3. 最后结合书籍评分数据实例，使用Spark Streaming框架实现书籍热度的实时计算。

学习重点

理论学习

- （1）初探 Spark Streaming。
- （2）掌握 DStream 编程模型。

实验学习

- （1）Spark Streaming 实时更新热门博文。
- （2）Spark Streaming实时更新热门博文。
- （3）过滤打印包含单词error的记录。

学习视频

理论视频学习

实训视频学习

测试数据准备

下载测试数据文件spark_data.tar.gz（若前面已下载过，可跳过）

```
mkdir /root/spark/
cd /root/spark/
wget http://biglab.site/b5951Ospark/file/spark_data.tar.gz
tar -xvzf ./spark_data.tar.gz
hdfs dfs -mkdir /user/myname
```

上传测试数据到hdfs

```
cd /root/spark
hdfs dfs -mkdir -p /user/myname/sparkStreaming/
hdfs dfs -put ./spark_data/BookRating.txt /user/myname/sparkStreaming/
hdfs dfs -put ./spark_data/a.txt /user/myname/sparkStreaming/
hdfs dfs -put ./spark_data/b.txt /user/myname/sparkStreaming/

```

任务6.1初探Spark Streaming

Spark Streaming任务描述

使用Spark Streaming实现书籍热度实时计算，首先需要对Spark Streaming基本概念及运行原理有大致的了解。本节的任务如下，

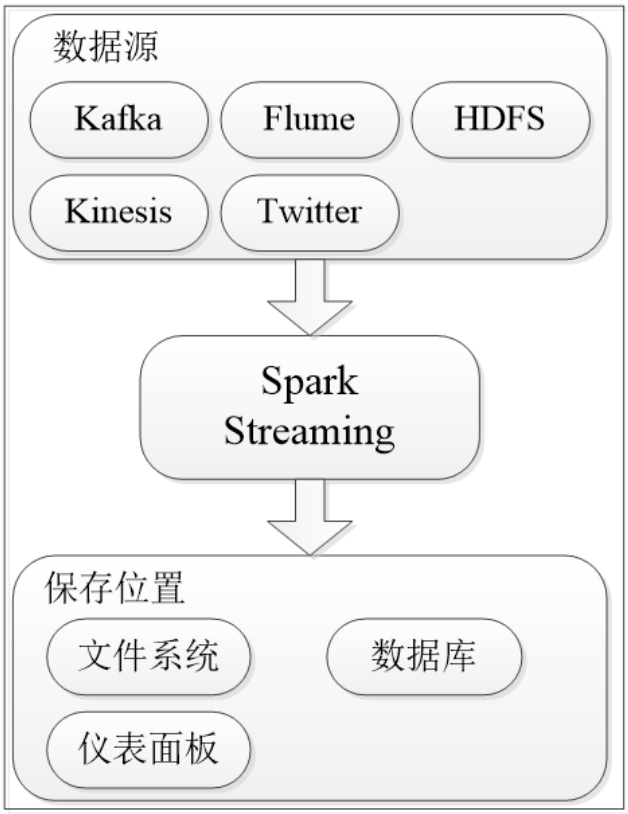
1. 了解Spark Streaming基本概念及运行原理；
2. 学习Spark Streaming程序的简单编写及运行。

Spark Streaming基本概念

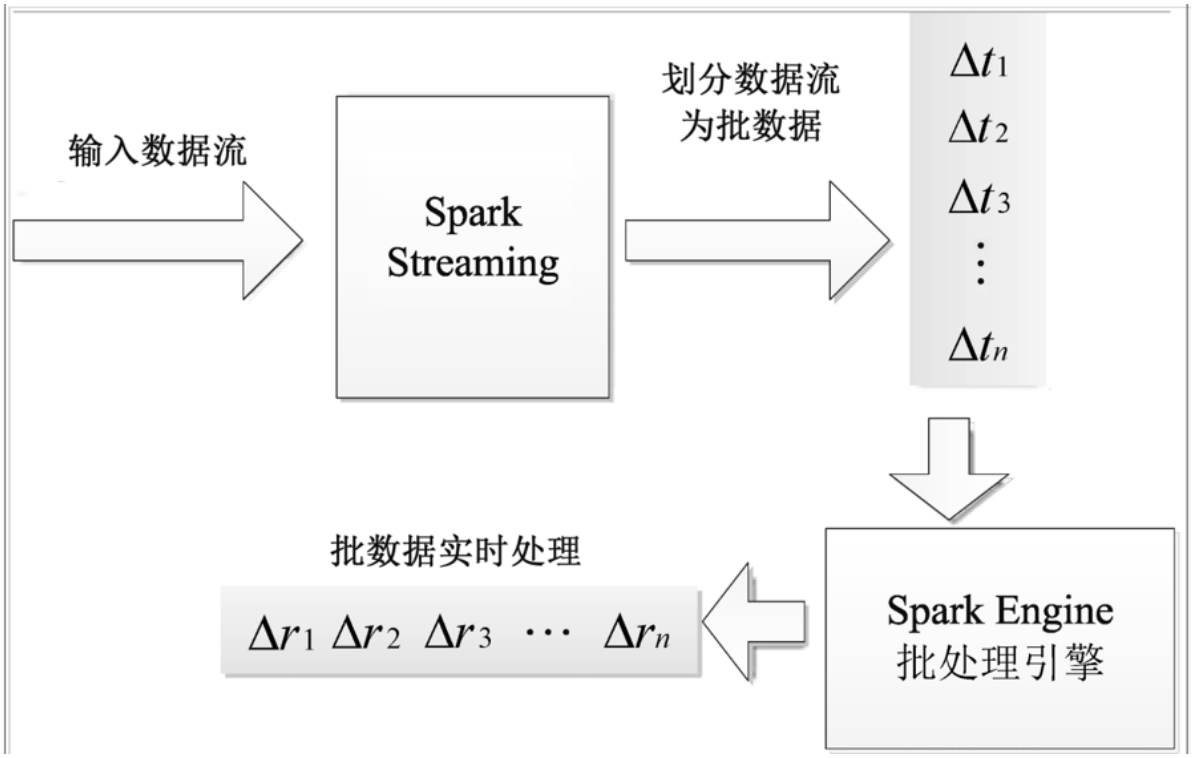
Spark Streaming是Spark的子框架，是Spark生态圈中用于处理流式数据的分布式流式处理框架，具有可伸缩、高吞吐量、容错能力强等特点。

同时，Spark Streaming能够和Spark SQL、Spark MLlib、Spark GraphX进行无缝集成，可以从Kafka、Flume、HDFS、Kinesis等数据源中获取数据，而且不仅可以通过调用map()、reduce()、join()等方法处理数据，也可以使用机器学习算法、图算法处理数据。

如图，经Spark Streaming处理后的最终结果可以保存在文件系统（如HDFS）、数据库（如MySQL）中或使用仪表面板进行实时展示。



Spark Streaming运行原理



Spark Streaming初步使用

使用Spark Streaming一般需要进行如下的操作。

1. 创建StreamingContext对象。
2. 创建InputDStream。
3. 操作Dstream。
4. 启动Spark Streaming。

实验一：监听Socket消息

在slave1上安装和使用nc软件

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

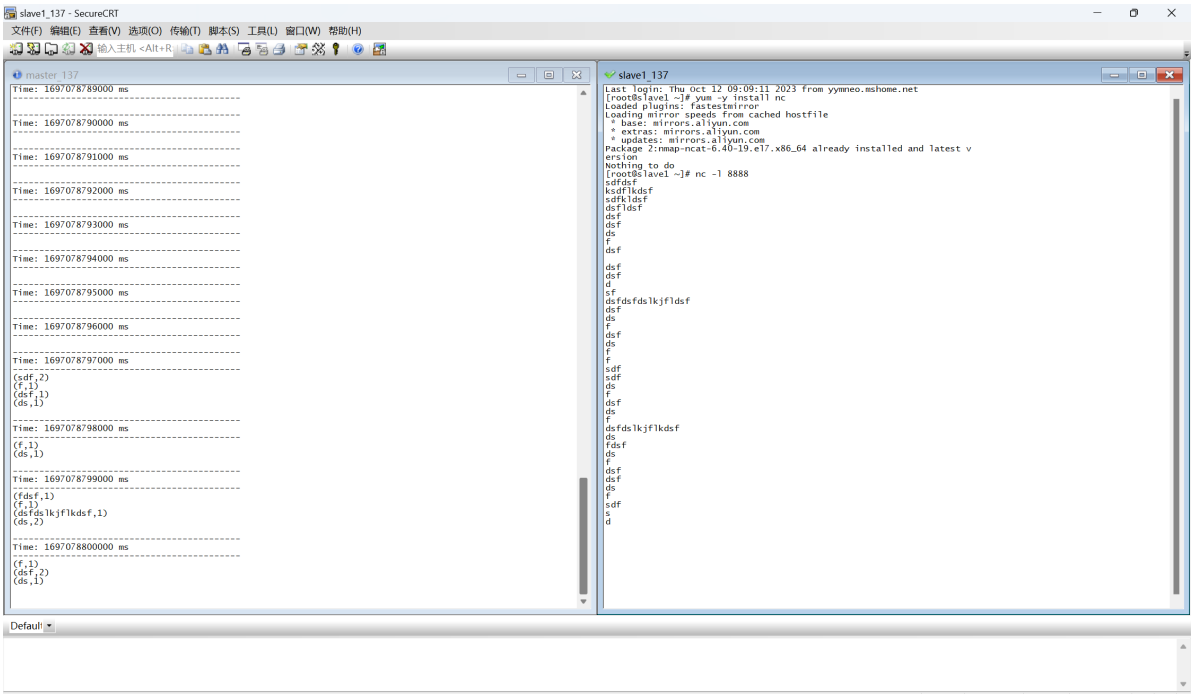
在master主机的spark-shell执行

```
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.dstream.DStream
import org.apache.spark.streaming.Duration
import org.apache.spark.streaming.Seconds
// 设置日志级别
sc.setLogLevel("WARN")
// 从SparkConf创建StreamingContext并指定1s的批处理大小
val ssc = new StreamingContext(sc, Seconds(1))
// 启动连接到slave1 8888端口上，使用收到的数据创建DStream
val lines = ssc.socketTextStream("slave1", 8888)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()
```

```
// 启动流计算环境StreamingContext
ssc.start()
```

监听Socket运行结果

如图，左侧是master上运行结果，右侧是slave1 的输入结果，随着右侧内容的输入，左侧会不断地输出计算结果



nc是netcat的简写，是一个功能强大的网络工具，有着网络界的瑞士军刀美誉。nc命令在linux系统中实际命令是ncat，nc是软连接到ncat。nc命令的主要作用如下：

- 实现任意TCP/UDP端口的侦听，nc可以作为server以TCP或UDP方式侦听指定端口
- 端口的扫描，nc可以作为client发起TCP或UDP连接
- 机器之间传输文件
- 机器之间网络测速

```
val wordCounts2=words.countByWindow(Seconds(3),Seconds(1))
 wordCounts2.print()

import org.apache.spark.streaming.StreamingContext
 import org.apache.spark.streaming.StreamingContext._
 import org.apache.spark.streaming.dstream.DStream
 import org.apache.spark.streaming.Duration
 import org.apache.spark.streaming.Seconds
 //设置日志级别
 sc.setLogLevel("WARN")
 //从SparkConf创建StreamingContext并指定1s的批处理大小
 val ssc=new StreamingContext(sc,Seconds(1))
 ssc.checkpoint("hdfs://master:9864/spark/checkpoing")
 //启动连接到slave1 8888端口上，使用收到的数据创建DStream
 val lines=ssc.socketTextStream("slave1",8888)
 val words=lines.flatMap(_.split(" "))
 val wordCounts2=lines.countByWindow(Seconds(3),Seconds(1))
 wordCounts2.print()
 //启动流计算环境StreamingContext
```

```
ssc.start()
```

实验二：监听HDFS目录

监听hdfs目录:/user/root/sparkStreaming/temp，分别上传a.txt,b.txt到该目录，就会在 10s内对文件中的单词进行计数并打印出来

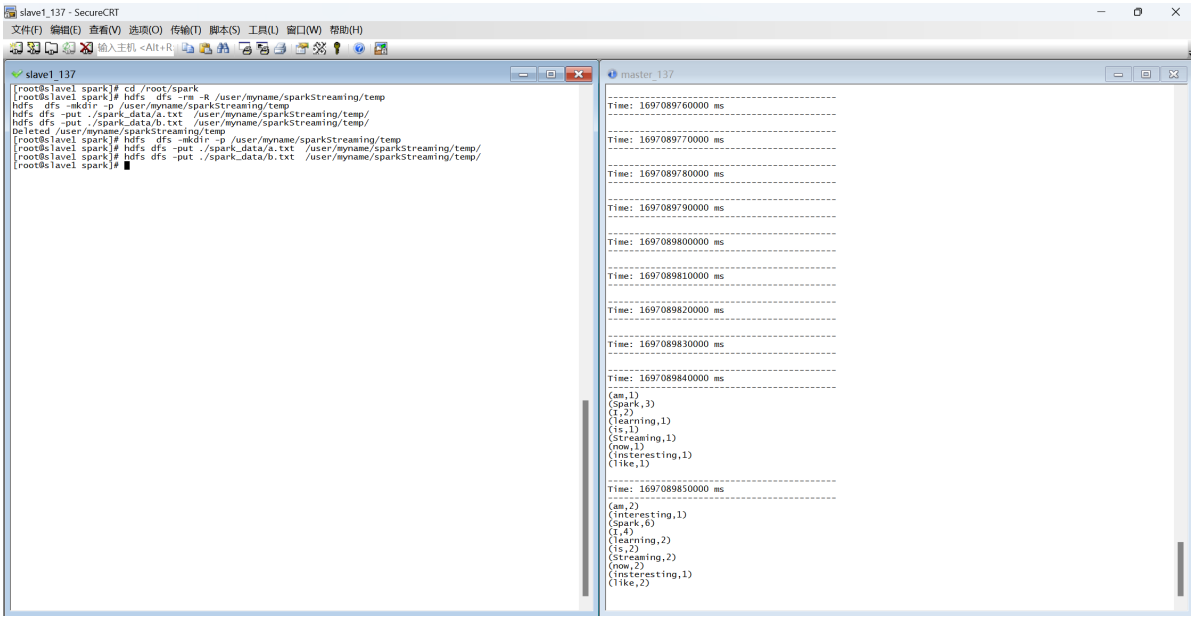
在master主机spark-shell执行

```
import org.apache.spark.streaming.{Seconds,StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.dstream.DStream
//设置日志级别
sc.setLogLevel("WARN")
//从SparkConf创建StreamingContext并指定1s的批处理大小
val ssc=new StreamingContext(sc,Seconds(10))
//启动连接到slave1 8888端口上，使用收到的数据创建DStream
val lines=ssc.textFileStream("/user/myname/sparkStreaming/temp")
val words=lines.flatMap(_.split(" "))
val wordCounts=words.map(x=>(x,1)).reduceByKey(_+_)
wordCounts.print()
//启动流计算环境StreamingContext
ssc.start()
ssc.awaitTermination()
```

在slave1上创建hdfs并上传测试文件

```
cd /root/spark
hdfs  dfs -rm -R /user/myname/sparkStreaming/temp
hdfs  dfs -mkdir -p /user/myname/sparkStreaming/temp
hdfs dfs -put ./spark_data/a.txt  /user/myname/sparkStreaming/temp/
hdfs dfs -put ./spark_data/b.txt  /user/myname/sparkStreaming/temp/
```

监听HDFS目录运行结果



任务6.2掌握DStream编程模型

任务描述

DStream是Spark Streaming中一个非常重要的概念。

Spark Streaming读取数据时会得到DStream编程模型，且DStream提供了一系列操作方法。

本节的任务如下。

1. 了解DStream的基本概念。
2. 学习DStream的转换操作、窗口操作以及输出操作。

了解DStream编程模型

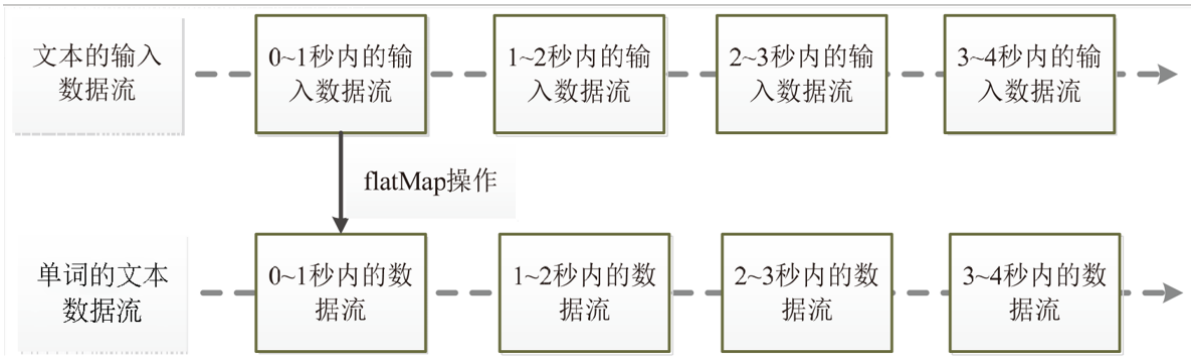
DStream是Spark Streaming对内部实时数据流的抽象描述，可将DStream理解为持续性的数据流。

1. 可以通过外部数据源获取DStream；
2. 也可以通过DStream现有的高级操作（如转换操作）获得DStream。

DStream代表着一系列的持续的RDD，DStream中的每个RDD都是按一小段时间分割开的RDD，如下图

对DStream的任何操作都会转化成对底层RDDs的操作。

以单词计数为例，获取文本数据形成文本的输入数据流lines DStream，使用flatMap()方法进行扁平化操作并进行分割，得到每一个单词，形成单词的文本数据流words DStream。



对DStream进行操作的方法根据操作的类型可以分成3类，即转换操作、窗口操作和输出操作

使用DStream转换操作

DStream转换操作常用的方法及说明

方法	描述
map(func)	对源DStream的每个元素应用func函数并返回一个新的DStream
flatMap(func)	类似map操作，不同的是每个元素可以被映射成0个或者多个输出元素
filter(func)	对源DStream中的每一个元素应用func函数进行计算，如果func函数返回结果为true，则保留该元素，否则丢弃该元素，返回一个新的DStream
union(otherStream)	合并两个DStream，生成一个包含两个DStream中所有元素的新的DStream
count()	统计DStream中每个RDD包含的元素的个数，得到一个只有一个元素的RDD构成的DStream
reduce(func)	对源DStream中的每个元素应用func函数进行聚合操作，返回一个内部所包含的RDD只有一个元素的新DStream
countByKey()	计算DStream中每个RDD内的元素出现的频次，并返回新的DStream[(K,Long)]，其中K是RDD中元素的类型，Long是元素出现的频次
reduceByKey(func, [numTasks])	以一个键值RDD为目标，K为键，V为值。当一个(K，V)键值对的DStream被调用时，返回(K,V)键值对的新DStream，其中每个键的值都使用聚合函数func汇总。配置numTasks可以设置不同的并行任务数
join(otherStream, [numTasks])	当调用的是(K,V1)和(K,V2)键值对的两个DStream时，返回元素为(K,(V1,V2))键值对的一个新DStream
cogroup(otherStream, [numTasks])	当被调用的两个DStream分别含有(K,V1)和(K,V2)键值对时，返回一个元素为(K, Seq[V1], Seq[V2])的新的DStream
transform(func)	通过对源DStream的每个RDD应用func函数返回一个新的DStream，用于在DStream上进行RDD的任意操作

实验一：监听Socket消息并转换操作

在slave1上安装和使用nc软件

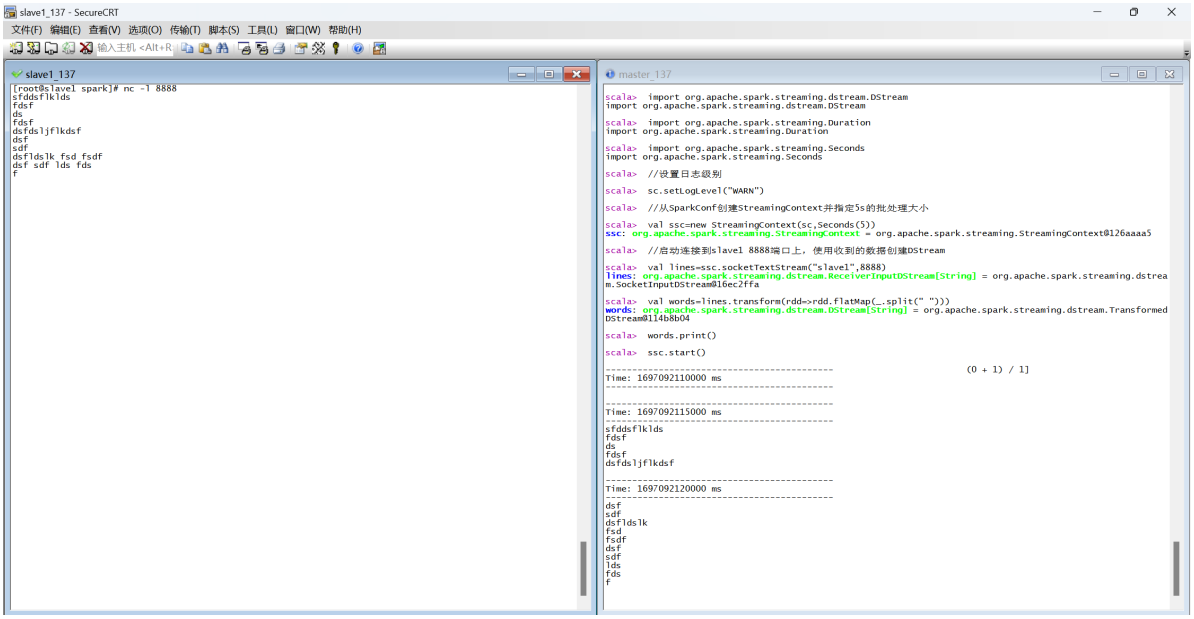
```
yum -y install nc
nc -l 8888
```

在master主机的spark-shell执行

```
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.dstream.DStream
import org.apache.spark.streaming.Duration
import org.apache.spark.streaming.Seconds
//设置日志级别
sc.setLogLevel("WARN")
//从SparkConf创建StreamingContext并指定5s的批处理大小
```

```
9  val ssc=new StreamingContext(sc,Seconds(5))
10 //启动连接到slave1 8888端口上，使用收到的数据创建DStream
11 val lines=ssc.socketTextStream("slave1",8888)
12 val words=lines.transform(rdd=>rdd.flatMap(_.split(" ")))
13 words.print()
14 ssc.start()
```

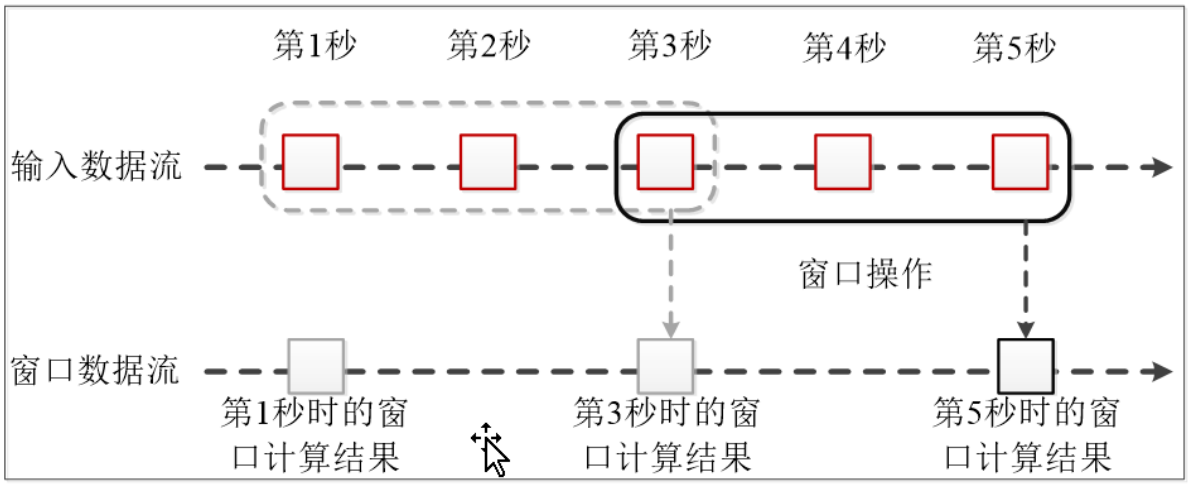
监听Socket消息并转换操作运行结果：



使用DStream窗口操作

窗口函数，就是在DStream流上，以一个可配置的长度为窗口，以一个可配置的速率向前移动窗口，根据窗口函数的具体内容，对窗口内的数据执行计算操作，每次掉落在窗口内的RDD的数据会被聚合起来执行计算操作，然后生成的RDD会作为Window DStream的一个RDD。

下图表述的是滑动窗口长度为3秒，这三秒内的3个RDD会被聚合起来进行处理，然后过了两秒钟，又会对最近三秒内的数据执行滑动窗口计算。所以每个滑动窗口操作，都必须指定两个参数，窗口长度以及滑动间隔，而且这两个参数值都必须是batch（批处理时间）间隔的整数倍。



常用的窗口转换操作方法如下表。这些操作都需要两个参数，windowLength（窗口长度）和slideInterval（时间间隔）

方法	描述
window(windowLength, slideInterval)	返回一个基于源DStream的窗口批次计算后得到的新DStream
countByWindow(windowLength, slideInterval)	返回基于滑动窗口的DStream中的元素的数量
reduceByWindow(func,windowLength, slideInterval)	基于滑动窗口对源DStream中的元素进行聚合操作，得到一个新的DStream
reduceByKeyAndWindow(func,windowLength, slideInterval, [numTasks])	基于滑动窗口对元素为(K,V)键值对的DStream中的值，按K使用func函数进行聚合操作，得到一个新的DStream
reduceByKeyAndWindow(func,invFunc, windowLength, slideInterval, [numTasks])	一个更高效的reduceByKeyAndWindow()的实现版本，其中每个窗口的统计量是使用前一个窗口的新数据和“反向减少”离开窗口的旧数据来实现的。例如，计算t+4秒这个时刻过去5秒窗口的WordCount，可以将t+3秒时刻过去5秒的统计量加上[t+3秒,t+4秒]的统计量，再减去[t−2秒,t−1秒]的统计量，这种方法可以复用中间3秒的统计量，提高统计的效率
countByValueAndWindow(windowLength, slideInterval, [numTasks])	基于滑动窗口计算源DStream中每个RDD内每个元素出现的频次并返回DStream[(K,Long)]，其中K是RDD中元素的类型，Long是元素频次。与countByValue一样，reduce任务的数量可以通过一个可选参数进行配置

实验一：监听Socket消息并使用window

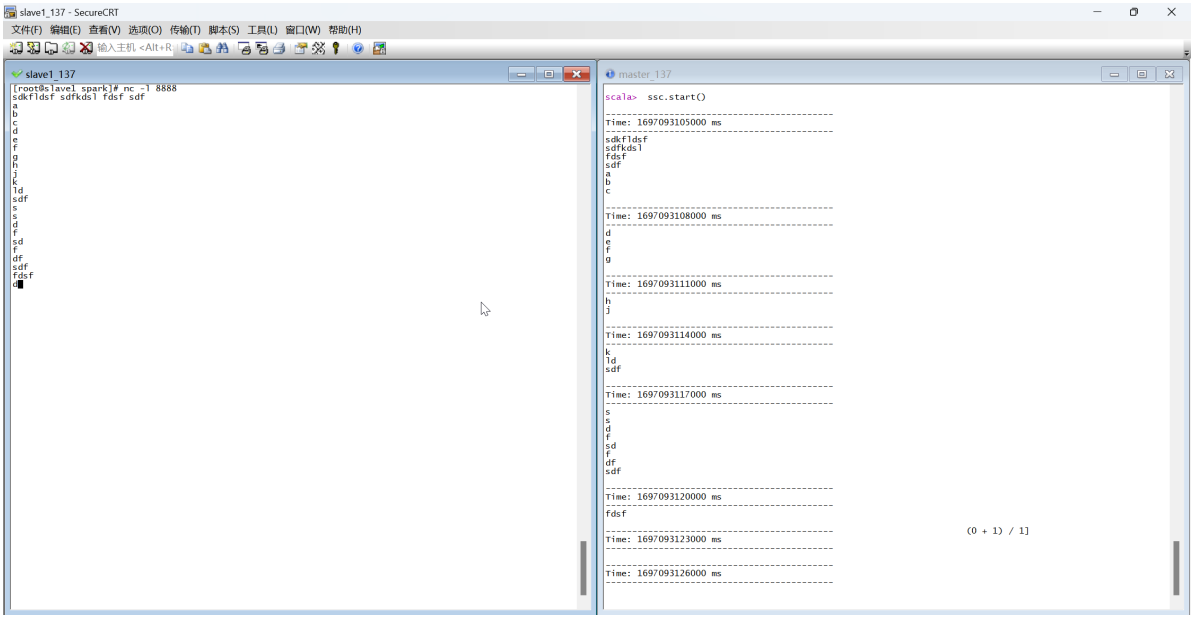
在slave1上安装和使用nc软件

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

在master主机的spark-shell执行

```
import org.apache.spark.streaming.{Seconds,StreamingContext}
import org.apache.spark.streaming.StreamingContext._
//设置日志级别
sc.setLogLevel("WARN")
//从SparkConf创建StreamingContext并指定1s的批处理大小
val ssc=new StreamingContext(sc,Seconds(1))
//启动连接到slave1 8888端口上，使用收到的数据创建DStream
val lines=ssc.socketTextStream("slave1",8888)
val words=lines.flatMap(_.split(" "))
val windowWords=words.window(Seconds(3),Seconds(3))
windowWords.print()
ssc.start()
```

监听Socket消息并使用window结果:



实验二：监听Socket消息并使用reduceByKeyAndWindow

在slave1上安装和使用nc软件

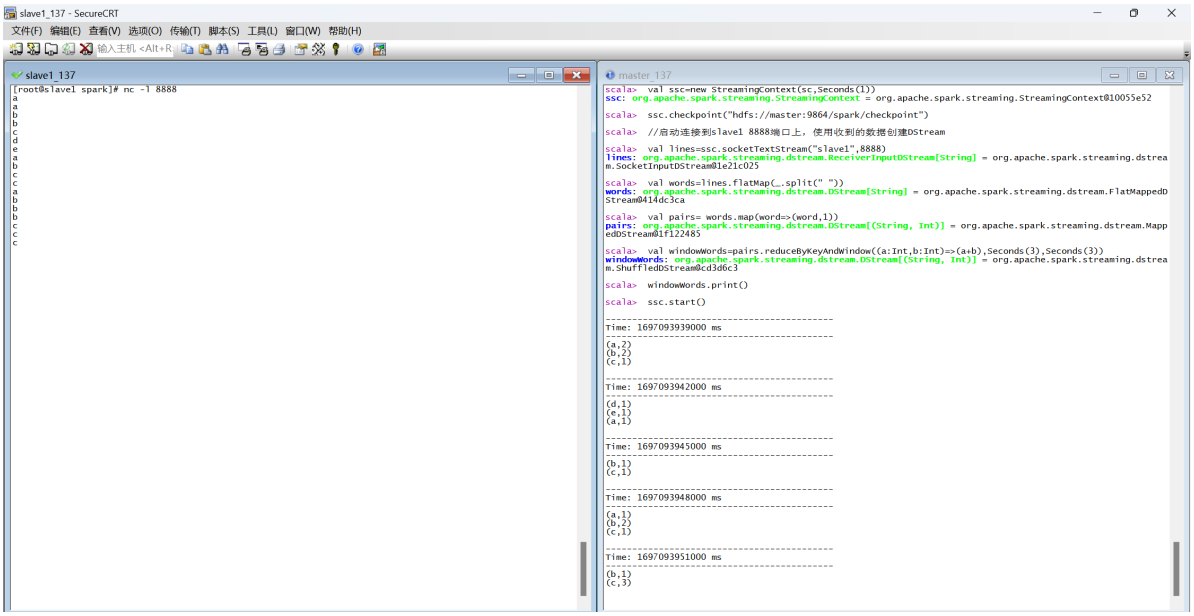
```
yum -y install nc
nc -l 8888
```

在master主机的spark-shell执行

```
import org.apache.spark.streaming.{Seconds,StreamingContext}
import org.apache.spark.streaming.StreamingContext._
//设置日志级别
sc.setLogLevel("WARN")
//从SparkConf创建StreamingContext并指定5s的批处理大小
val ssc=new StreamingContext(sc,Seconds(1))
ssc.checkpoint("hdfs://master:9864/spark/checkpoint")
//启动连接到slave1 8888端口上，使用收到的数据创建DStream
val lines=ssc.socketTextStream("slave1",8888)
val words=lines.flatMap(_.split(" "))
val pairs= words.map(word=>(word,1))
val windowWords=pairs.reduceByKeyAndWindow((a:Int,b:Int)=>
(a+b),Seconds(3),Seconds(3))
windowWords.print()
```

```
14  ssc.start()
```

监听Socket消息并使用reduceByKeyAndWindow结果



使用DStream输出操作

特点

是真正的触发操作，类似action操作

常用输出操作

方法	描述
print()	在Driver中输出DStream中数据的前10个元素
saveAsTextFiles(prefix, [suffix])	将DStream中的内容以文本的形式保存为文本文件，其中每次批处理间隔内产生的文件再单独保存为文件夹，文件夹以prefix_TIME_IN_MS[.suffix]的方式命名
saveAsObjectFiles(prefix, [suffix])	将DStream中的内容按对象序列化，并且以SequenceFile的格式保存。其中每次批处理间隔内产生的文件以prefix_TIME_IN_MS[.suffix]的方式命名
saveAsHadoopFiles(prefix, [suffix])	将DStream中的内容以文本的形式保存为Hadoop文件，其中每次批处理间隔内产生的文件以prefix_TIME_IN_MS[.suffix]的方式命名
foreachRDD(func)	基本的输出操作，将func函数应用于DStream中的RDD上，输出数据至外部系统，如保存RDD到文件或网络数据库等

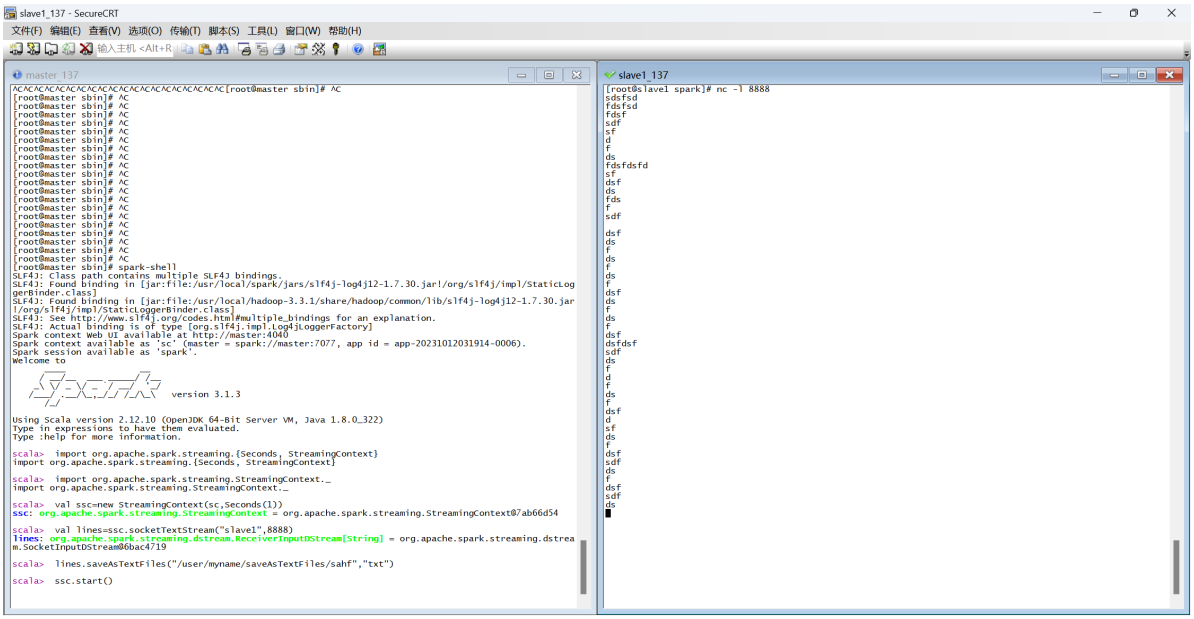
实验一：使用saveAsTextFiles输出到文件

def saveAsTextFiles(prefix: String, suffix: String = ""): Unit

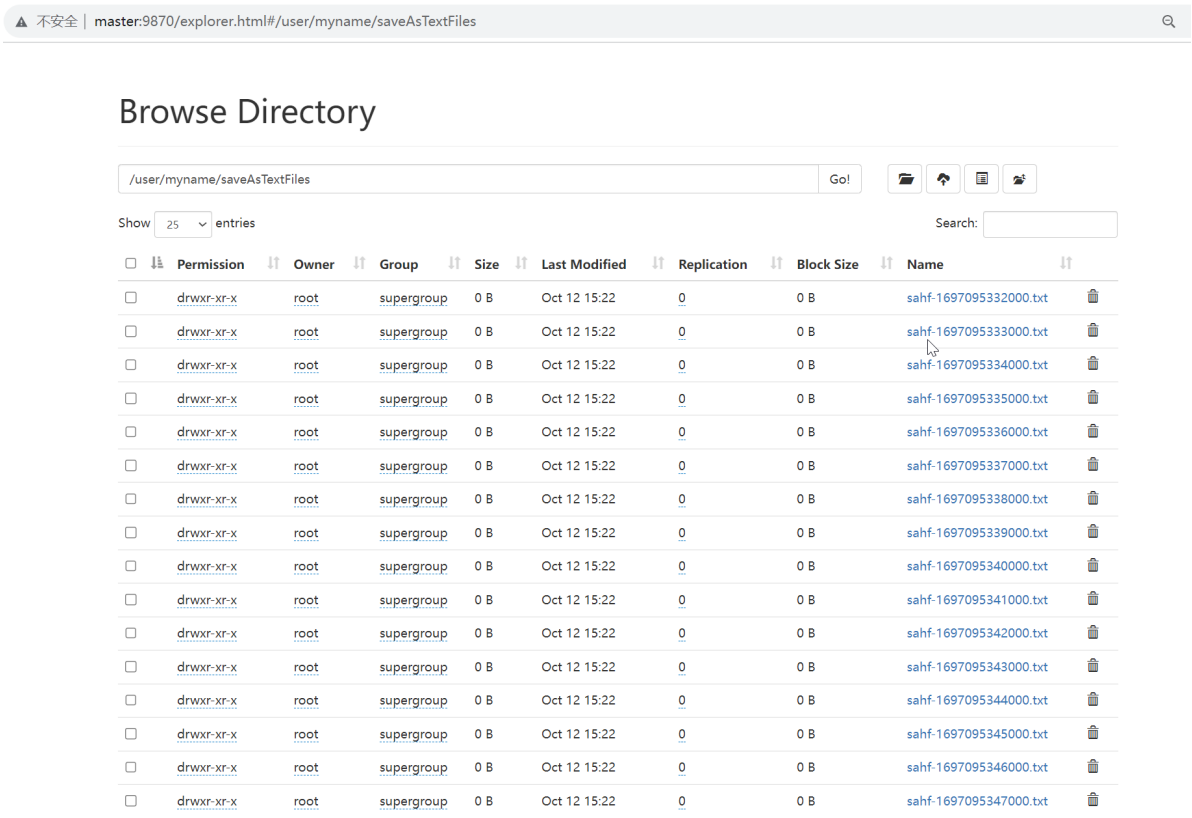
在master主机的spark-shell执行

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
val ssc=new StreamingContext(sc,Seconds(1))
val lines=ssc.socketTextStream("slave1",8888)
lines.saveAsTextFiles("/user/myname/saveAsTextFiles/sahf","txt")
ssc.start()
```

在master上运行结果



在master:9870web界面上结果：



实验二：使用foreachRDD输出mysql

在slave1上安装和使用nc软件

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

在IDEA上完成WriteDataToMysql源码

```
package main.java

import java.sql.{Connection, DriverManager, PreparedStatement}
 import org.apache.spark.SparkConf
 import org.apache.spark.streaming._
 import org.apache.spark.streaming.Seconds
 object WriteDataToMysql {
  def main(args: Array[String]): Unit = {
   val conf: SparkConf = new
SparkConf().setAppName("WriteDataToMySQL").setMaster("local[*]")
   val ssc = new StreamingContext(conf, Seconds(5))
   val ItemsStream = ssc.socketTextStream("slave1", 8888)
   val ItemPairs = ItemsStream.map(line => (line.split(",")(0), 1))
   val ItemCount = ItemPairs.reduceByKeyAndWindow((v1: Int, v2: Int) => v1 +
v2, Seconds(60), Seconds(10))
   val hottestWord = ItemCount.transform(itemRDD => {
    val top3 = itemRDD.map(pair => (pair._2,
pair._1)).sortByKey(false).map(pair => (pair._2, pair._1)).take(3)
    ssc.sparkContext.makeRDD(top3)
   })
   hottestWord.foreachRDD(rdd => {
    rdd.foreachPartition(partitionOfRecords => {
     val url = "jdbc:mysql://home.hddly.cn:53306/test?useSSL=false"
     val user = "test"
     val password = "test"
     Class.forName("com.mysql.jdbc.Driver")
     val conn = DriverManager.getConnection(url, user, password)
 //     conn.prepareStatement("delete from searchKeyWord where
1=1").executeUpdate()
     conn.setAutoCommit(false)
     val stmt = conn.createStatement()
     partitionOfRecords.foreach(record => {
      println("WriteDataToMysql:record:" + record._1 + "," + record._2 )
      stmt.addBatch("insert into searchKeyWord
(insert_time,keyword,search_count) values (now(),'" + record._1 + "','" +
record._2 + "')")
     })

     stmt.executeBatch()
     conn.commit()
    })
   })
   ssc.start()
   ssc.awaitTermination()
   ssc.stop()
  }
```

```
41  }
```

打包上传到master

由于slave1上运nc,所以，使用sftp上传word.jar到master上

```
lcd D:\yuxm\wordcount\out\artifacts\word
cd /root/spark/
put word.jar
```

结果如：

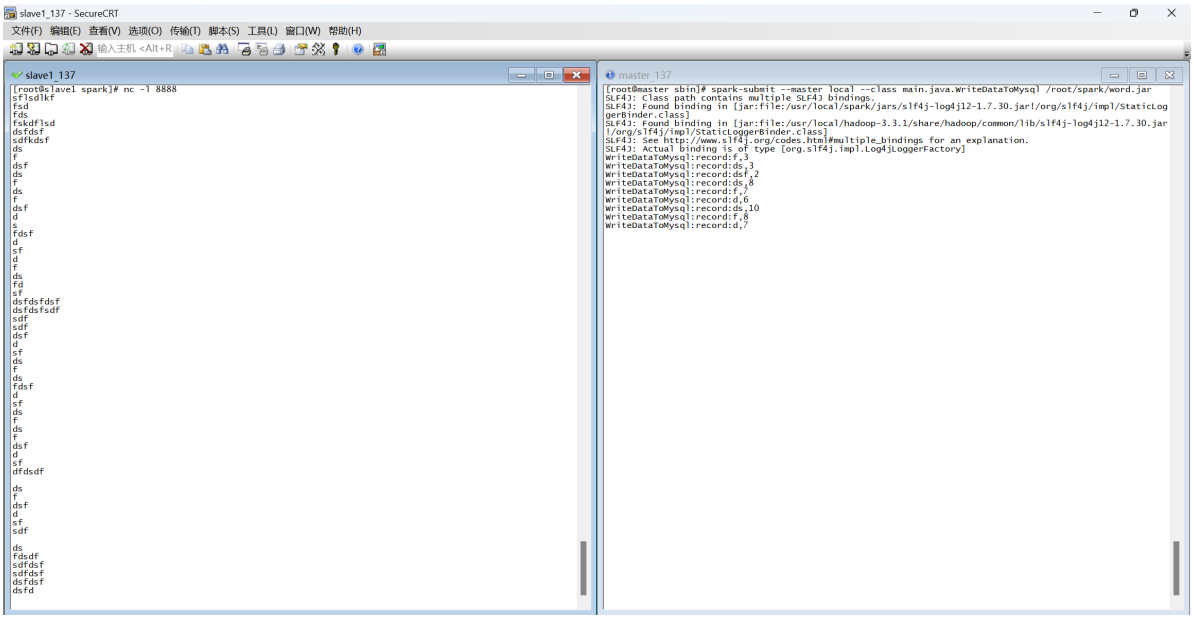
```
sftp> lcd D:\yuxm\wordcount\out\artifacts\word
sftp> cd /root/spark/
sftp> put word.jar
Uploading word.jar to /root/spark/word.jar
  100% 29KB     29KB/s 00:00:00
D:/yuxm/wordcount/out/artifacts/word/word.jar: 30327 bytes transferred in 0
seconds (29 KB/s)
sftp>

```

任务执行：

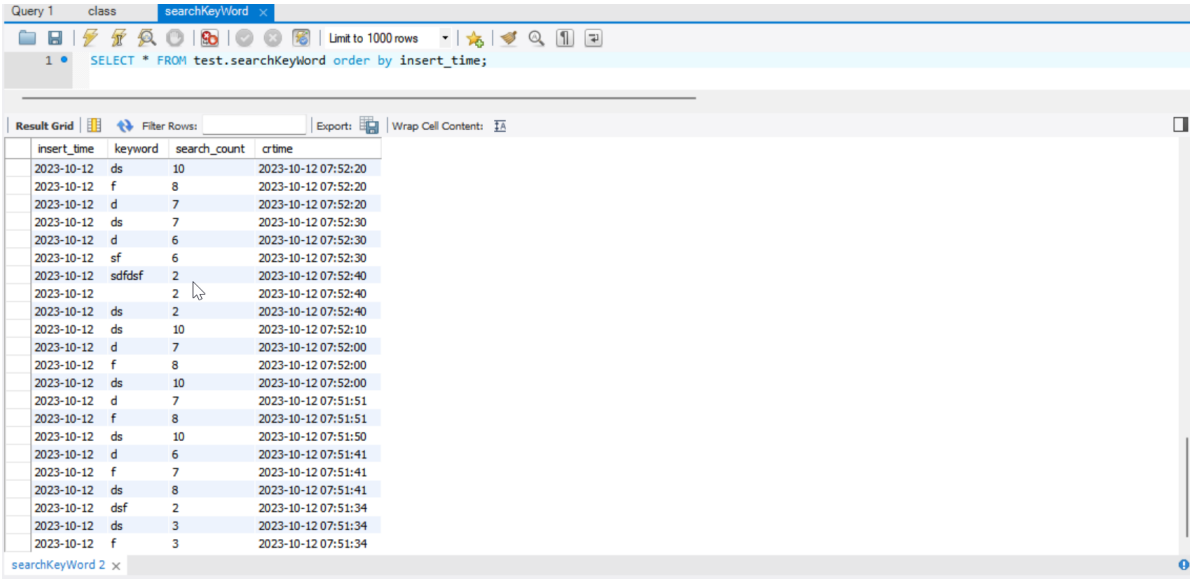
切换到shell命令行执行脚本

```
spark-submit --master local --class main.java.WriteDataToMysql
/root/spark/word.jar
```

运行结果：



Mysql表查询结果：



任务6.3实现书籍热度实时计算

任务描述

掌握了Spark Streaming的DStream编程模型的基础操作后，即可使用Spark Streaming框架解决实际的实时数据流处理问题。本节的任务如下。使用Spark Streaming实时计算书籍热度；根据书籍热度进行降序排序，获取热度最高的10本图书；将最后的结果写入Hive中。

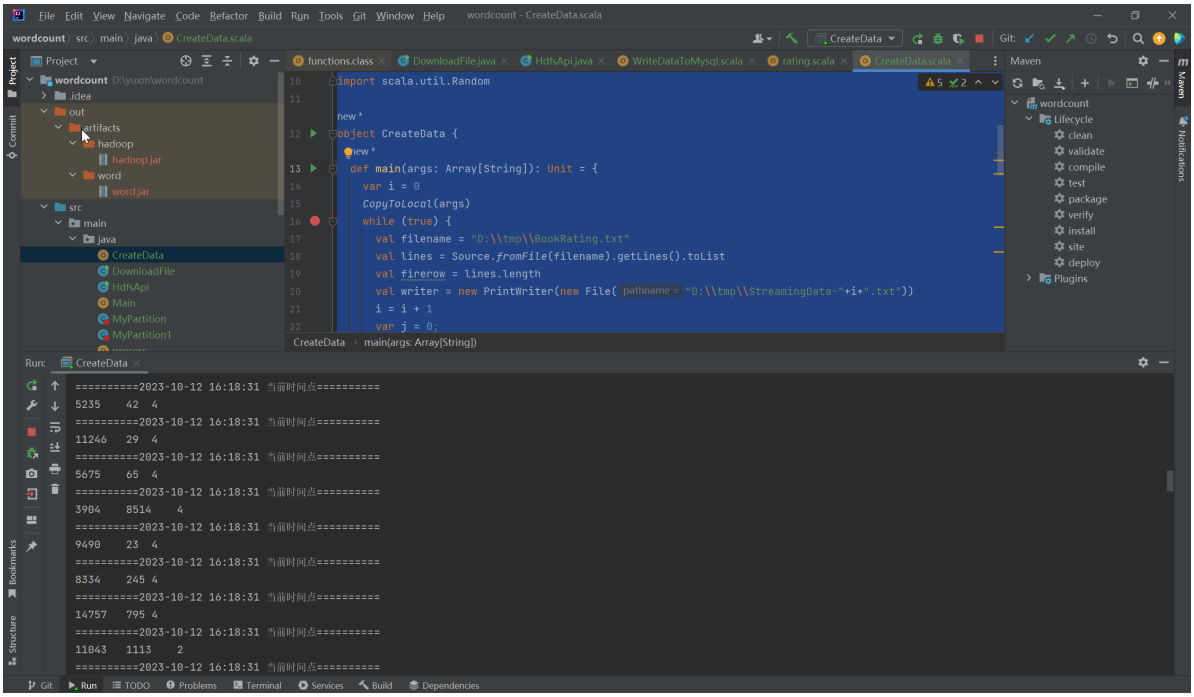
程序一：模拟程序生成日志

模拟程序源码：

```
package chap06

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

import java.io.{File, IOException, PrintWriter}
import java.text.SimpleDateFormat
import java.util.Date
import scala.io.Source
import scala.util.Random

object CreateData {
  def main(args: Array[String]): Unit = {
    var i = 0
    CopyToLocal(args)
    while (true) {
      val filename = "D:\\tmp\\BookRating.txt"
      val lines = Source.fromFile(filename).getLines().toList
      val firerow = lines.length
      val writer = new PrintWriter(new File("D:\\tmp\\StreamingData-
"+i+".txt"))
      i = i + 1
      var j = 0;
      while (j < 100) {
        writer.write(lines(index(firerow)) + "\n")
```

```
        val time = new Date().getTime
        val format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
        println("=" * 10 + format.format(time) + " 当前时间点" + "=" * 10)
        println(lines(index(firerow)))
        j = j + 1
      }
      writer.close()
      Thread sleep 60000
    }
  }

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

  @throws[IOException]
  def CopyToLocal(args: Array[String]): Unit = {
    //获取配置
    val conf = new Configuration
    conf.set("fs.defaultFS", "hdfs://master:9864/")
    //获取文件系统
    val fs = FileSystem.get(conf)
    //声明源文件路径和目标路径
    val fromPath = new Path("/user/myname/sparkStreaming/BookRating.txt")
    val toPath = new Path("D:/tmp")
    //调用copyToLocalFile方法下载文件到本地
    fs.copyToLocalFile(false, fromPath, toPath, true)
    //关闭文件系统
    fs.close()
  }

}


```

在IDEA上运行模拟程序



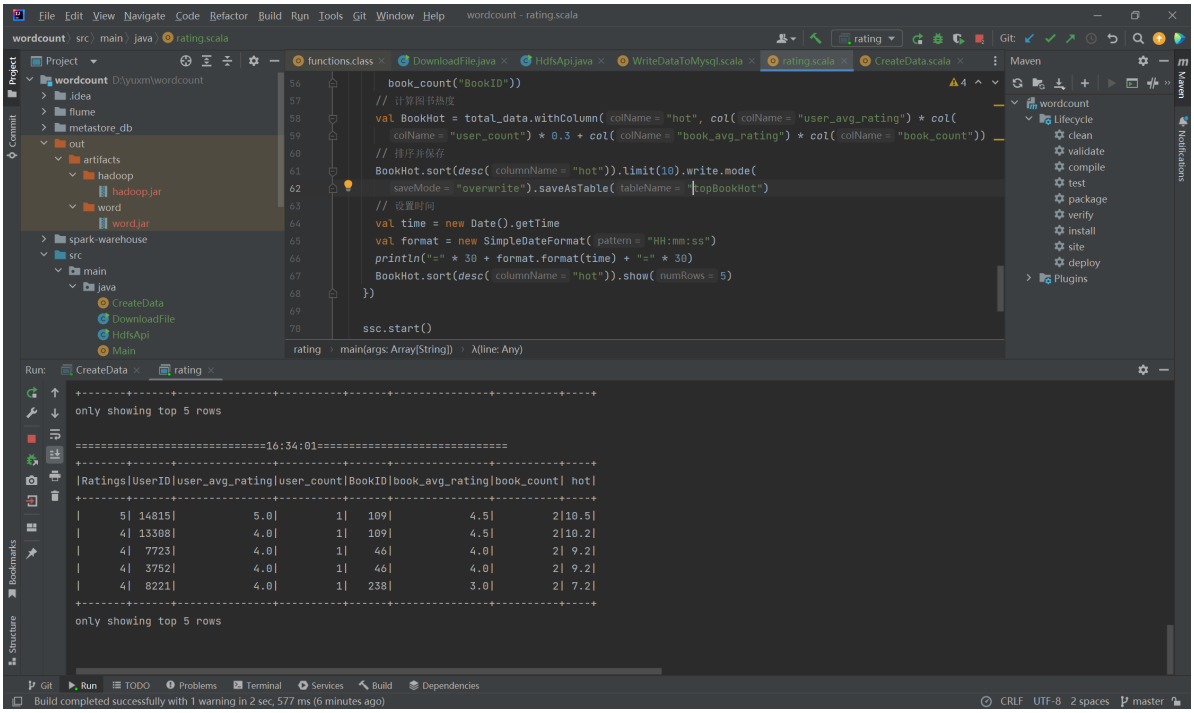
程序二：实时统计书籍热度

统计热度源码

```scala
package chap06

import java.text.SimpleDateFormat
import java.util.Date
import org.apache.spark.sql.hive.HiveContext
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.{SparkConf, SparkContext}

object rating {
  def main(args: Array[String]): Unit = {
    // 实例化SparkContext，设置日志输出等级为ERROR
    val conf = new SparkConf().setAppName("book").setMaster("local[*]")
    val sc = new SparkContext(conf)
    val hiveContext = new HiveContext(sc)
    sc.setLogLevel("ERROR")
    // 设置批次窗口时间间隔和日志文件存放点
    val ssc = new StreamingContext(sc, Seconds(60))
    ssc.checkpoint("./flume")
    // 获取数据流
    val stream = ssc.textFileStream("D:\\tmp\\")
    // 以转译字符分割数据
    // stream.print()
    val splitData = stream.map {
      x => val y = x.split("\t"); (y(0), y(1), y(2).toInt)
    }
    // splitData.print()
    // 使用foreachRDD将DStream转换为RDD
    splitData.foreachRDD(line => {
      import hiveContext.implicits._
      import org.apache.spark.sql.functions._
```

```
31        // 将RDD数据转换为DataFrame处理
32        val dataFrame = line.toDF("UserID", "BookID", "Ratings")
33        // 根据需求计算用户平均分，书本平均分，用户的评分次数，书本的被评分次数
34        val user_rating = dataFrame.groupBy(
35          "UserID").avg("Ratings").withColumnRenamed(
36          "avg(Ratings)", "user_avg_rating")
37        val book_rating = dataFrame.groupBy(
38          "BookID").avg("Ratings").withColumnRenamed(
39          "avg(Ratings)", "book_avg_rating")
40        val user_count = dataFrame.groupBy(
41          "UserID").count().withColumnRenamed("count", "user_count")
42        val book_count = dataFrame.groupBy(
43          "BookID").count().withColumnRenamed("count", "book_count")
44        // 合并4份dataFrame
45        val data_user_rating = dataFrame.join(
46          user_rating, user_rating("UserID") === dataFrame("UserID")).drop(
47          user_rating("UserID"))
48        val data_user = data_user_rating.join(
49          user_count, user_count("UserID") ===
data_user_rating("UserID")).drop(
50          user_count("UserID"))
51        val data_user_book_rating = data_user.join(
52          book_rating, book_rating("BookID") === data_user("BookID")).drop(
53          book_rating("BookID"))
54        val total_data = data_user_book_rating.join(book_count,
55          book_count("BookID") === data_user_book_rating("BookID")).drop(
56          book_count("BookID"))
57        // 计算图书热度
58        val BookHot = total_data.withColumn("hot", col("user_avg_rating") *
col(
59          "user_count") * 0.3 + col("book_avg_rating") * col("book_count"))
60        // 排序并保存
61        BookHot.sort(desc("hot")).limit(10).write.mode(
62          "overwrite").saveAsTable("topBookHot")
63        // 设置时间
64        val time = new Date().getTime
65        val format = new SimpleDateFormat("HH:mm:ss")
66        println("=" * 30 + format.format(time) + "=" * 30)
67        BookHot.sort(desc("hot")).show(5)
68      })
69
70      ssc.start()
71      ssc.awaitTermination()
72    }
73  }
74
```

在IDEA上运行统计热度



版本历史

- Ver1.1-20221009
  - 增加随堂练习
  - 增加实训作业
- Ver1.2-20221020
  - 修改实训数据准备内容
  - 大段脚本采集gitee的链接
    - 使用gitee.com地址,需要先注册并登陆
- Ver1.3-20221025
  - 更新异常情况：运行没有数据
- Ver1.4-20221030
  - 更新理论课的视频
- Ver1.5-20230828
  - 增加markdown版本

上一章节 下一章节