Spark Streaming 实战

目 录

Spark Streaming 实战

1 实例演示

1.1 流数据模拟器

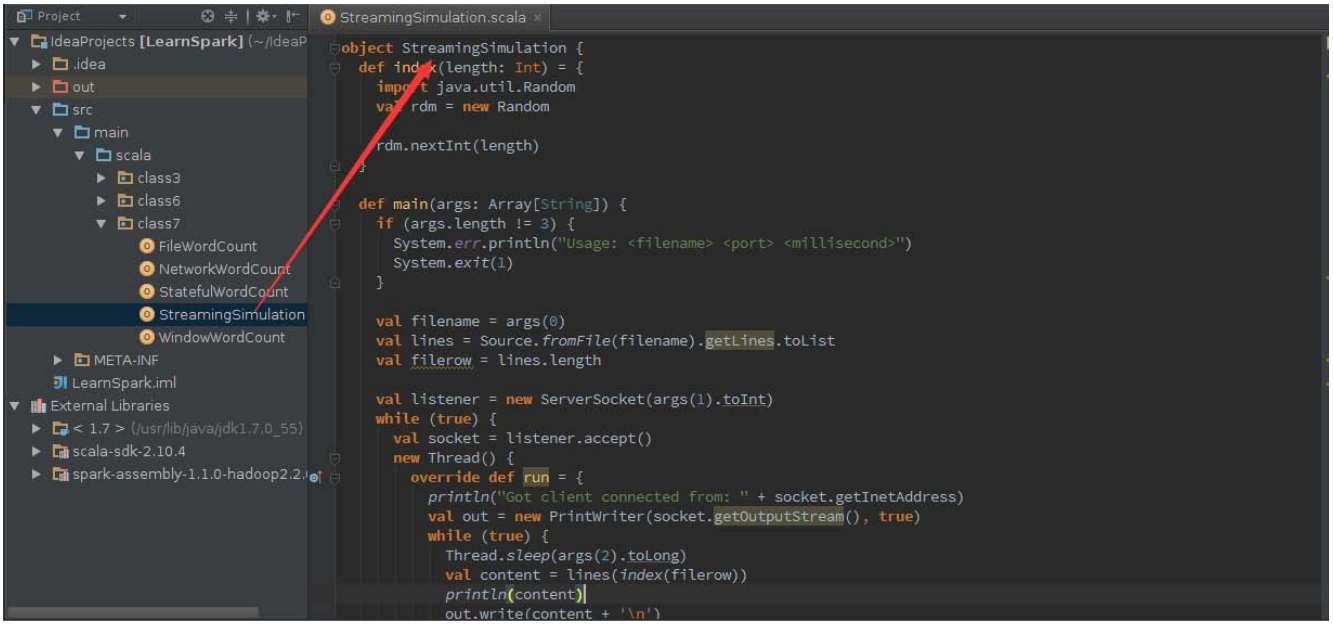
1.1.1 流数据说明

在实例演示中模拟实际情况，需要源源不断地接入流数据，为了在演示过程中更接近真实环境将定义流数据模拟器。该模拟器主要功能：通过 Socket 方式监听指定的端口号，当外部程序通过该端口连接并请求数据时，模拟器将定时将指定的文件数据随机获取发送给外部程序。

1.1.2 模拟器代码

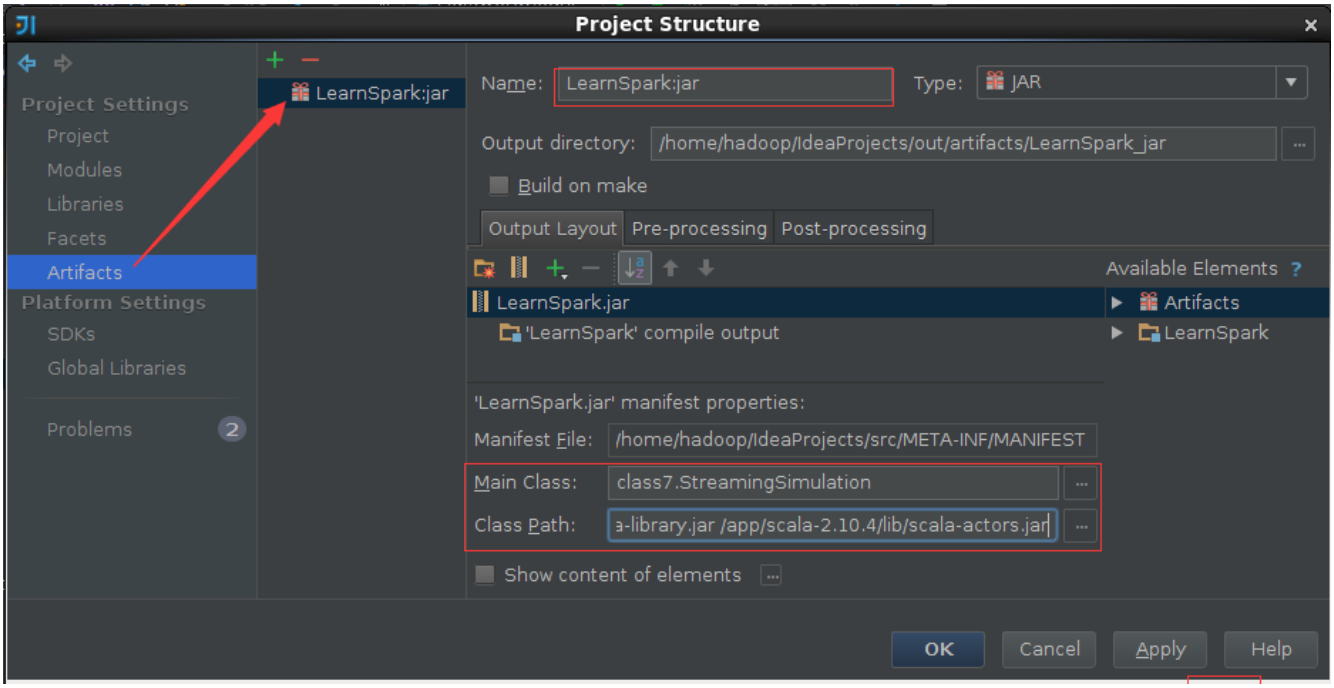
```
import java.io.{PrintWriter}
import java.net.ServerSocket
import scala.io.Source

object StreamingSimulation {
  // 定义随机获取整数的方法
  def index(length: Int) = {
    import java.util.Random
    val rdm = new Random
    rdm.nextInt(length)
  }

  def main(args: Array[String]) {
    // 调用该模拟器需要三个参数，分为为文件路径、端口号和间隔时间（单位：毫秒）
    if (args.length != 3) {
      System.err.println("Usage: <filename> <port> <millisecond>")
      System.exit(1)
    }

    // 获取指定文件总的行数
    val filename = args(0)
```

```
    val lines = Source.fromFile(filename).getLines.toList
    val filerow = lines.length

    // 指定监听某端口，当外部程序请求时建立连接
    val listener = new ServerSocket(args(1).toInt)
    while (true) {
      val socket = listener.accept()
      new Thread() {
        override def run = {
          println("Got client connected from: " + socket.getInetAddress)
          val out = new PrintWriter(socket.getOutputStream(), true)
          while (true) {
            Thread.sleep(args(2).toLong)
            // 当该端口接受请求时，随机获取某行数据发送给对方
            val content = lines(index(filerow))
            println(content)
            out.write(content + '\n')
            out.flush()
          }
          socket.close()
        }
      }.start()
    }
  }
}
```



1.1.3 生成打包文件

【注】可以参见第 3 课《Spark 编程模型（下）--IDEA 搭建及实战》进行打包

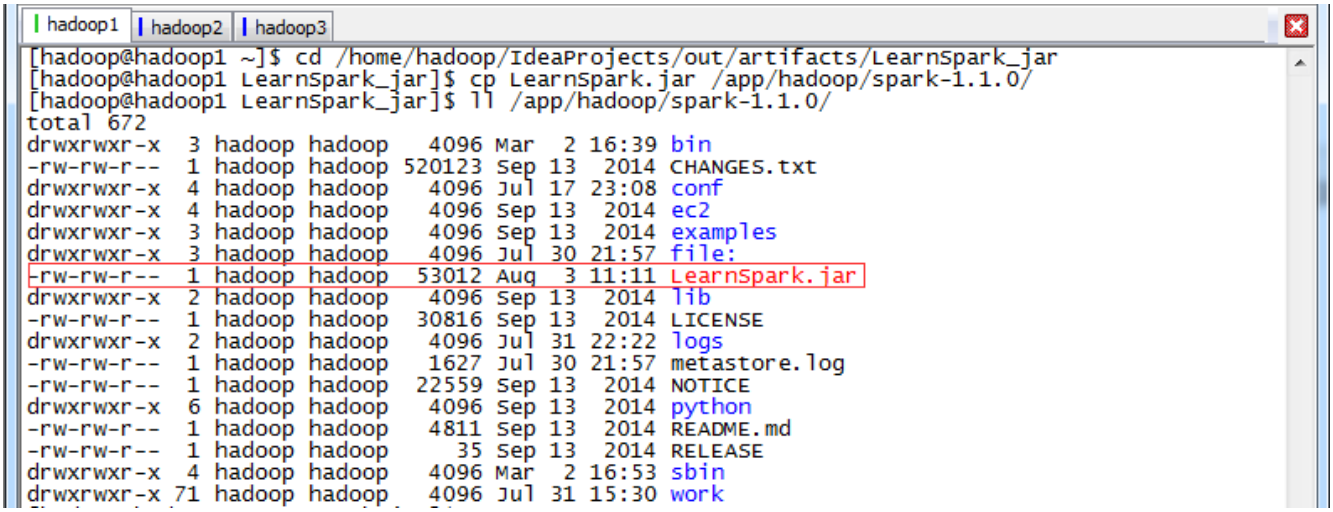


在打包配置界面中，需要在 Class Path 加入：/app/scala-2.10.4/lib/scala-swing.jar /app/scala-2.10.4/lib/scala-library.jar /app/scala-2.10.4/lib/scala-actors.jar ，各个 jar 包之间用空格分开，

点击菜单 Build->Build Artifacts，弹出选择动作，选择 Build 或者 Rebuild 动作，使用如下命令复制打包文件到 Spark 根目录下

cd /home/hadoop/IdeaProjects/out/artifacts/LearnSpark_jar
cp LearnSpark.jar /app/hadoop/spark-1.1.0/

ll /app/hadoop/spark-1.1.0/



1.2 实例 1：读取文件演示

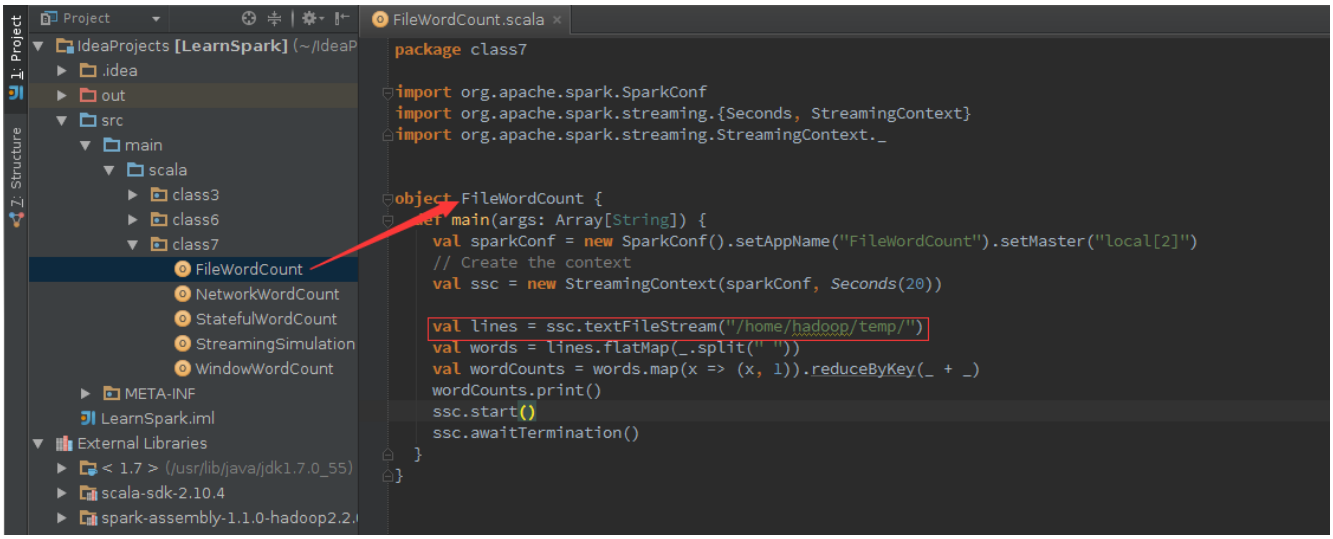
1.2.1 演示说明

在该实例中 Spark Streaming 将监控某目录中的文件，获取在间隔时间段内变化的数据，然后通过 Spark Streaming 计算出改时间段内单词统计数。

1.2.2 演示代码

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

object FileWordCount {
  def main(args: Array[String]) {
    val sparkConf = new
    SparkConf().setAppName("FileWordCount").setMaster("local[2]")

    // 创建 Streaming 的上下文，包括 Spark 的配置和时间间隔，这里时间为间隔 20 秒
    val ssc = new StreamingContext(sparkConf, Seconds(20))

    // 指定监控的目录，在这里为/home/hadoop/temp/
    val lines = ssc.textFileStream("/home/hadoop/temp/")

    // 对指定文件夹变化的数据进行单词统计并且打印
    val words = lines.flatMap(_.split(" "))
```

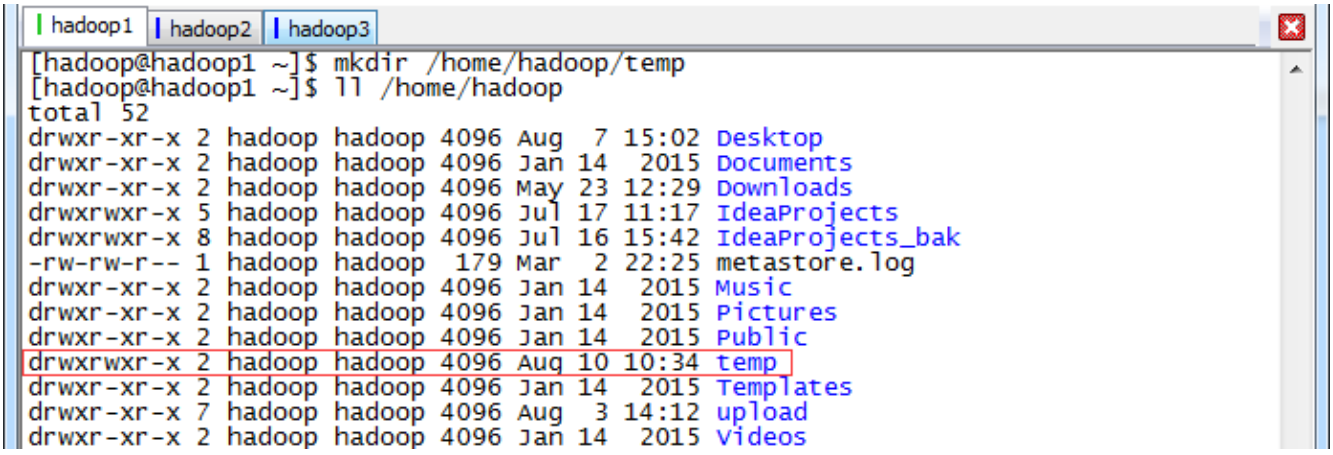
```
    val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
    wordCounts.print()

    // 启动 Streaming
    ssc.start()
    ssc.awaitTermination()
  }
}
```



1.2.3 运行代码

第一步 创建 Streaming 监控目录

创建/home/hadoop/temp 为 Spark Streaming 监控的目录 ,通过在该目录中定时添加文件内容，然后由 Spark Streaming 统计出单词个数



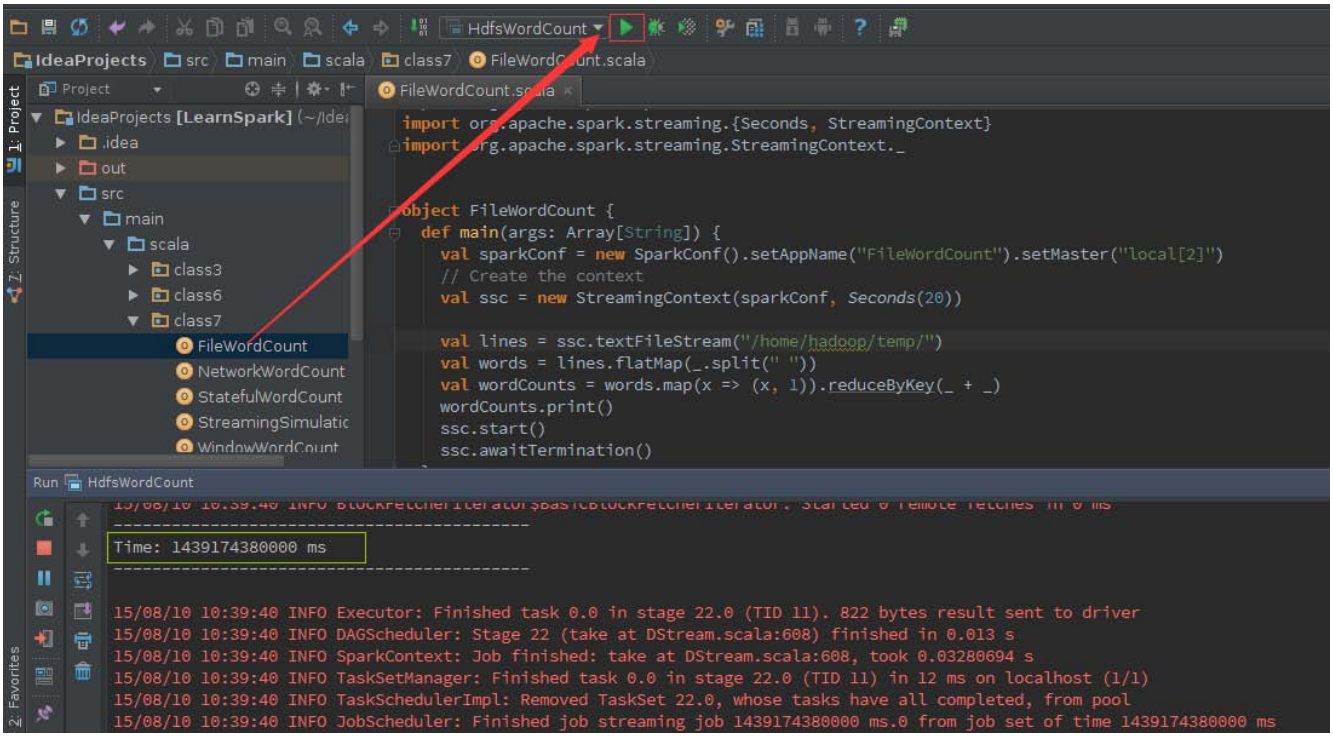
第二步 使用如下命令启动 Spark 集群

$cd /app/hadoop/spark-1.1.0

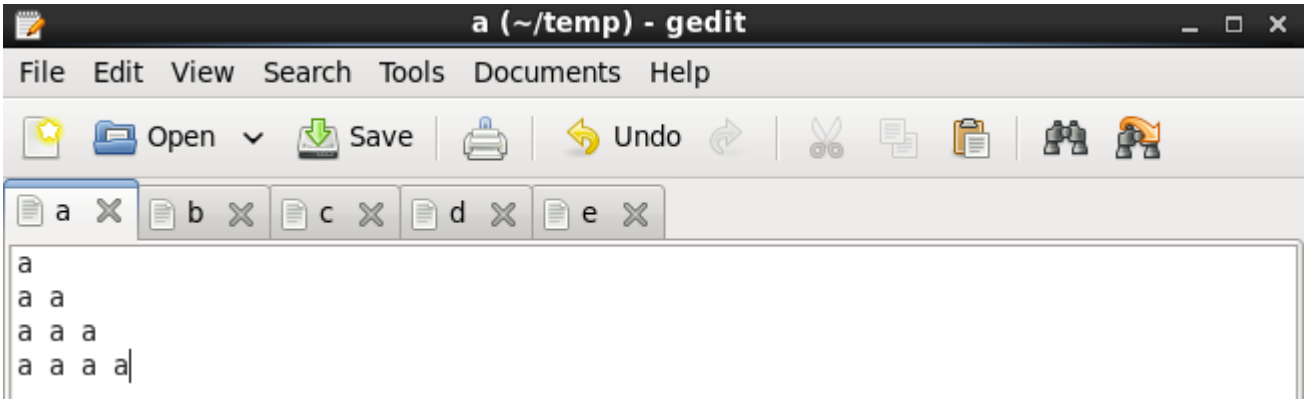
$sbin/start-all.sh

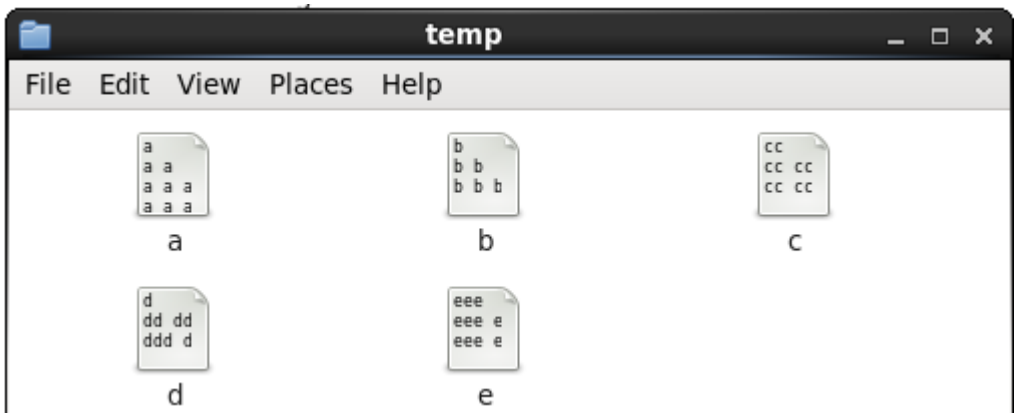
第三步 在 IDEA 中运行 Streaming 程序

在 IDEA 中运行该实例，由于该实例没有输入参数故不需要配置参数，在运行日志中将定时打印时间戳。如果在监控目录中加入文件内容，将输出时间戳的同时将输出单词统计个数。



1.2.4 添加文本及内容

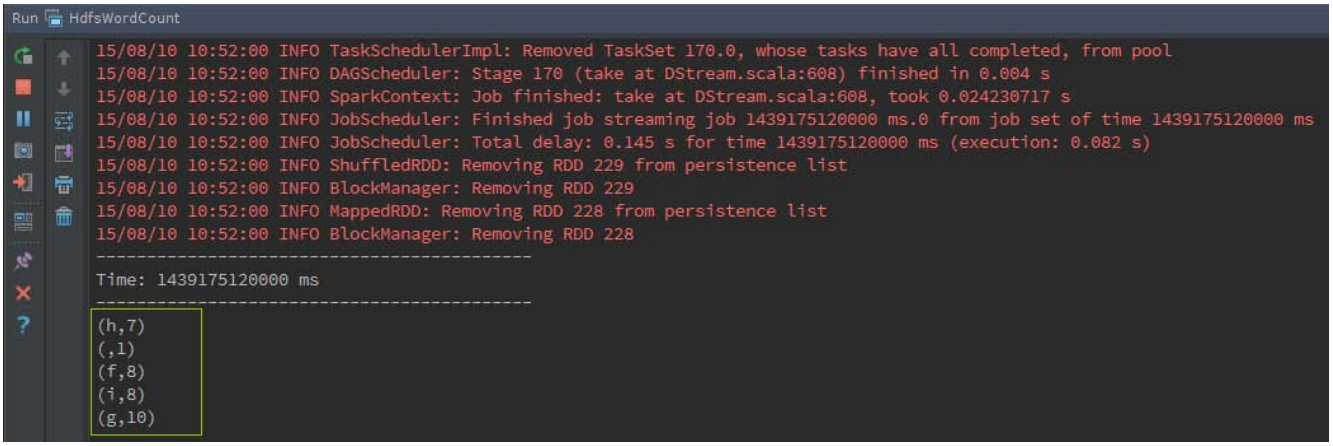




1.2.5 查看结果

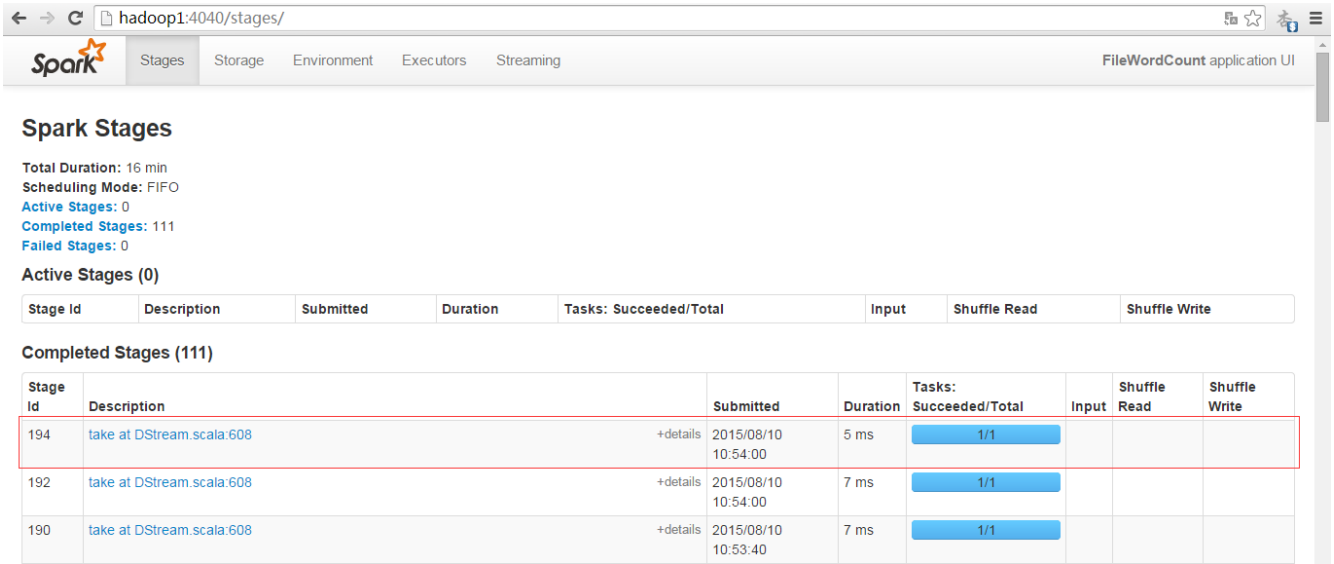
第一步 查看 IDEA 中运行情况

在 IDEA 的运行日志窗口中，可以观察到输出时间戳的同时将输出单词统计个数

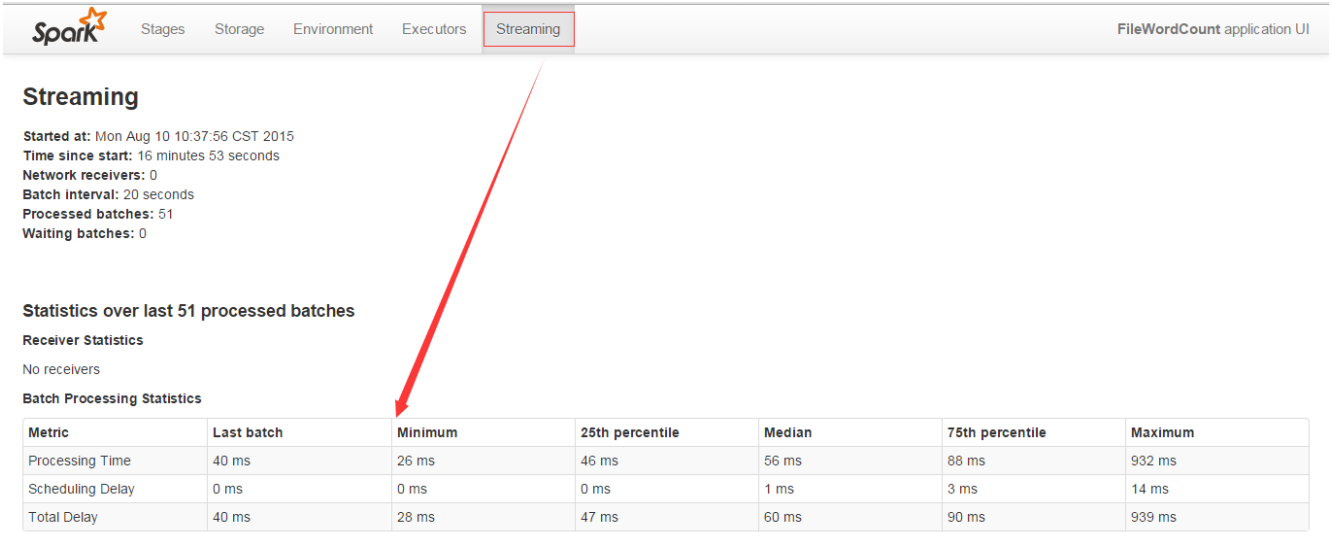


第二步 通过 webUI 监控运行情况

在 http://hadoop1:4040 监控 Spark Streaming 运行情况，可以观察到每 20 秒运行一次作业



并且与其他运行作业相比在监控菜单增加了"Streaming"项目，点击可以看到监控内容：



1.3 实例 2：网络数据演示

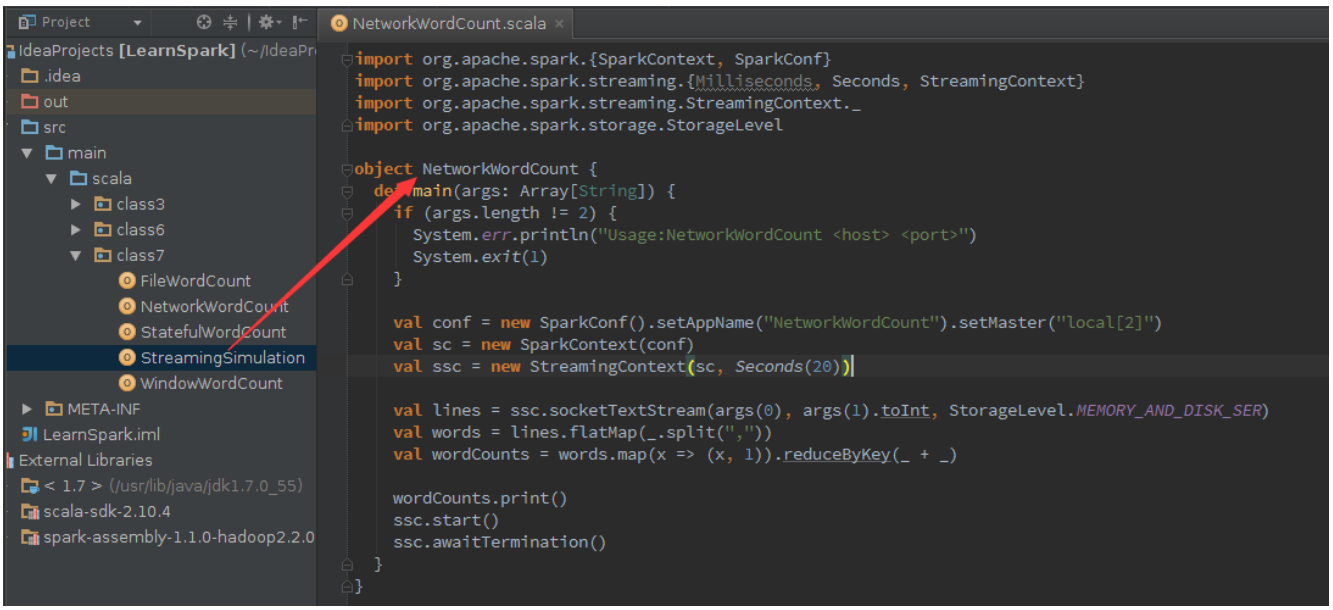
1.3.1 演示说明

在该实例中将由 4.1 流数据模拟以 1 秒的频度发送模拟数据，Spark Streaming 通过 Socket 接收流数据并每 20 秒运行一次用来处理接收到数据，处理完毕后打印该时间段内数据出现的频度，即在各处理段时间之间状态并无关系。

1.3.2 演示代码

```
import org.apache.spark.{SparkContext, SparkConf}
import org.apache.spark.streaming.{Milliseconds, Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel

object NetworkWordCount {
  def main(args: Array[String]) {
    val conf = new
    SparkConf().setAppName("NetworkWordCount").setMaster("local[2]")
    val sc = new SparkContext(conf)
    val ssc = new StreamingContext(sc, Seconds(20))

    // 通过 Socket 获取数据，该处需要提供 Socket 的主机名和端口号，数据保存在内存和硬盘中
    val lines = ssc.socketTextStream(args(0), args(1).toInt,
    StorageLevel.MEMORY_AND_DISK_SER)
```

```
    // 对读入的数据进行分割、计数
    val words = lines.flatMap(_.split(","))
    val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)

    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```



1.3.3 运行代码

第一步 启动流数据模拟器

启动 4.1 打包好的流数据模拟器，在该实例中将定时发送/home/hadoop/upload/class7 目录下的 people.txt 数据文件（该文件可以在本系列配套资源目录/data/class7 中找到），其中 people.txt 数据内容如下：

```
Michael
Andy
Justin

```

模拟器 Socket 端口号为 9999，频度为 1 秒，

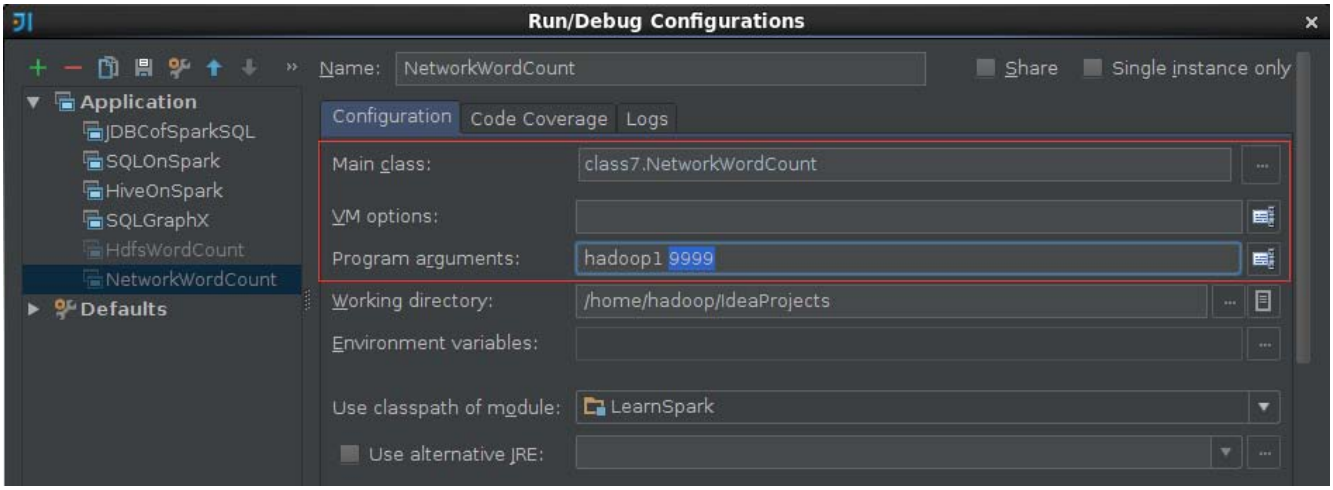
```
$cd /app/hadoop/spark-1.1.0
$java -cp LearnSpark.jar class7.StreamingSimulation /home/hadoop/upload/class7/people.txt 9999 1000
```

```
hadoop1 | hadoop2 | hadoop3
[hadoop@hadoop1 ~]$ cd /app/hadoop/spark-1.1.0
[hadoop@hadoop1 spark-1.1.0]$ ls
bin          conf  examples          lib      logs            NOTICE   README.md  sbin
CHANGES.txt  ec2   LearnSpark.jar    LICENSE  metastore.log   python   RELEASE    work
[hadoop@hadoop1 spark-1.1.0]$
[hadoop@hadoop1 spark-1.1.0]$ java -cp LearnSpark.jar class7.StreamingSimulation /home/hadoop
load/class7/people.txt 9999 1000
```

在没有程序连接时，该程序处于阻塞状态

第二步 在 IDEA 中运行 Streaming 程序

在 IDEA 中运行该实例，该实例需要配置连接 Socket 主机名和端口号，在这里配置参数机器名为 hadoop1 和端口号为 9999



1.3.4 查看结果

第一步 观察模拟器发送情况

IDEA 中的 Spark Streaming 程序运行与模拟器建立连接，当模拟器检测到外部连接时开始发送测试数据，数据是随机的在指定的文件中获取一行数据并发送，时间间隔为 1 秒

```
hadoop1 | hadoop2 | hadoop3
[hadoop@hadoop1 spark-1.1.0]$ java -cp LearnSpark.jar class7.StreamingSimulation /home/hadoop
load/class7/people.txt 9999 1000
Got client connected from: /192.168.10.111
Andy
Michael
Justin
Andy
Andy
Justin
Andy
Michael
Justin
Michael
Michael
Justin
Justin
Justin
```

第二步 在监控页面观察执行情况

在 webUI 上监控作业运行情况，可以观察到每 20 秒运行一次作业

Active Stages (1)

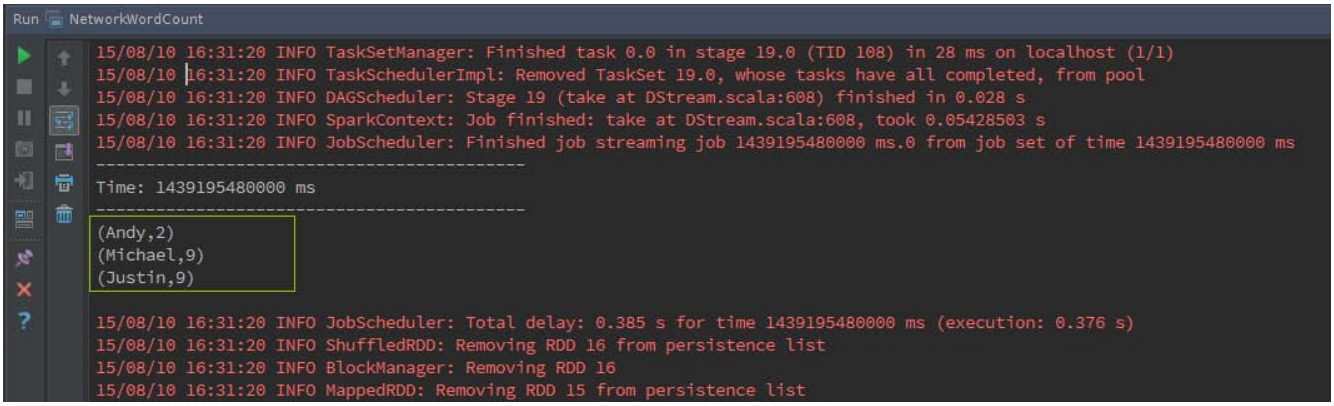
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
0	runJob at ReceiverTracker.scala:275 +details (kill)	2015/08/10 16:29:40	1.7 min	0/1			

Completed Stages (15)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
19	take at DStream.scala:608 +details	2015/08/10 16:31:20	28 ms	1/1			
17	take at DStream.scala:608 +details	2015/08/10 16:31:20	30 ms	1/1			
18	map at MappedDStream.scala:35 +details	2015/08/10 16:31:20	0.2 s	20/20			3.3 KB
15	take at DStream.scala:608 +details	2015/08/10 16:31:00	8 ms	1/1			
13	take at DStream.scala:608 +details	2015/08/10 16:31:00	32 ms	1/1			
14	map at MappedDStream.scala:35 +details	2015/08/10 16:31:00	0.2 s	20/20			3.3 KB
11	take at DStream.scala:608 +details	2015/08/10 16:30:40	19 ms	1/1			

第三步 IDEA 运行情况

在 IDEA 的运行窗口中，可以观测到的统计结果，通过分析在 Spark Streaming 每段时间内单词数为 20，正好是 20 秒内每秒发送总数。



1.4 实例 3：销售数据统计演示

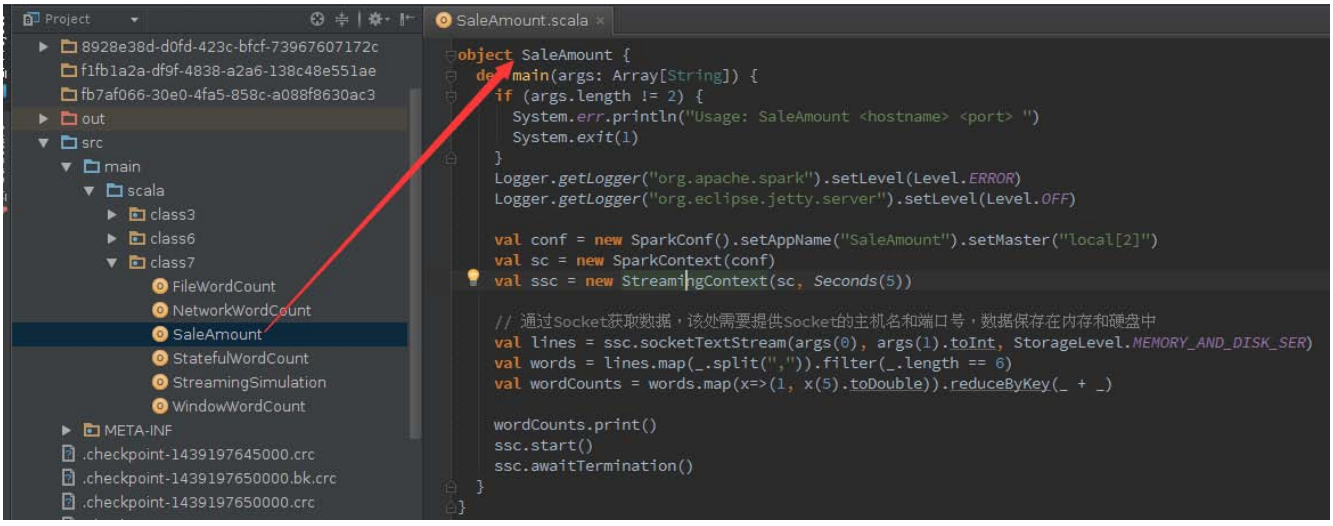
1.4.1 演示说明

在该实例中将由 4.1 流数据模拟器以 1 秒的频度发送模拟数据（销售数据），Spark Streaming 通过 Socket 接收流数据并每 5 秒运行一次用来处理接收到数据，处理完毕后打印该时间段内销售数据总和，需要注意的是各处理段时间之间状态并无关系。

1.4.2 演示代码

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.{SparkContext, SparkConf}
import org.apache.spark.streaming.{Milliseconds, Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel
```

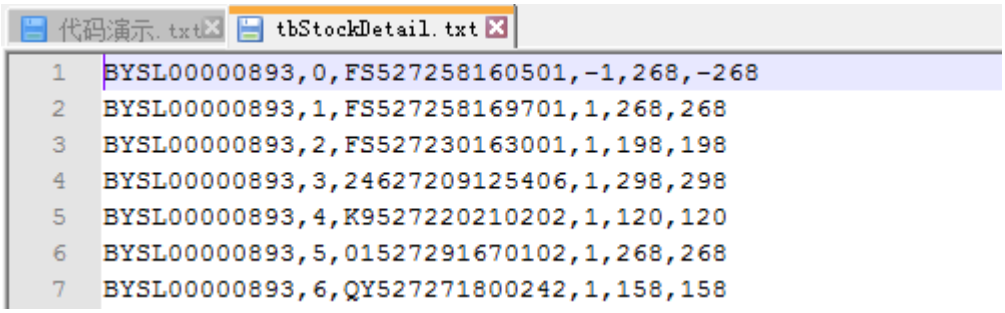
```
object SaleAmount {
  def main(args: Array[String]) {
    if (args.length != 2) {
      System.err.println("Usage: SaleAmount <hostname> <port> ")
      System.exit(1)
    }
    Logger.getLogger("org.apache.spark").setLevel(Level.ERROR)
    Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

    val conf = new SparkConf().setAppName("SaleAmount").setMaster("local[2]")
    val sc = new SparkContext(conf)
    val ssc = new StreamingContext(sc, Seconds(5))

   // 通过 Socket 获取数据，该处需要提供 Socket 的主机名和端口号，数据保存在内存和硬盘中
    val lines = ssc.socketTextStream(args(0), args(1).toInt,
    StorageLevel.MEMORY_AND_DISK_SER)
    val words = lines.map(_.split(",")).filter(_.length == 6)
    val wordCounts = words.map(x=>(1, x(5).toDouble)).reduceByKey(_ + _)

    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```



1.4.3 运行代码

第一步 启动流数据模拟器

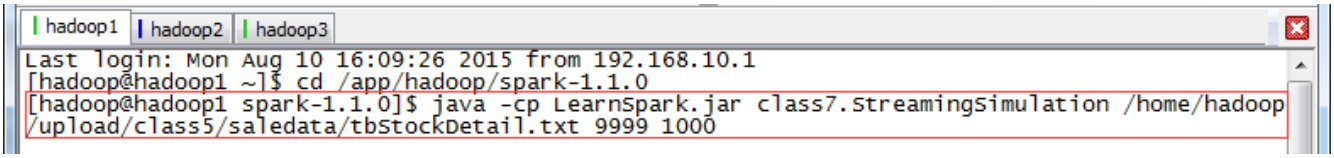
启动 4.1 打包好的流数据模拟器，在该实例中将定时发送第五课 /home/hadoop/upload/class5/saledata 目录下的 tbStockDetail.txt 数据文件（参见第五课《5.Hive（下）--Hive 实战》中 2.1.2 数据描述，该文件可以在本系列配套资源目录 /data/class5/saledata 中找到），其中表 tbStockDetail 字段分别为订单号、行号、货品、数量、金额，数据内容如下：



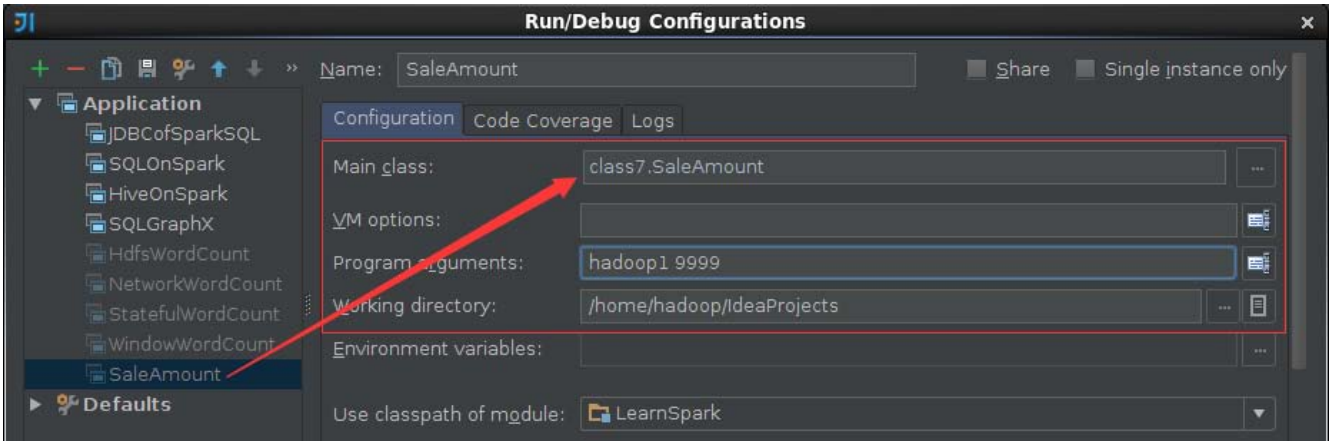
模拟器 Socket 端口号为 9999，频度为 1 秒

$cd /app/hadoop/spark-1.1.0

$java -cp LearnSpark.jar class7.StreamingSimulation /home/hadoop/upload/class5/saledata/tbStockDetail.txt 9999 1000



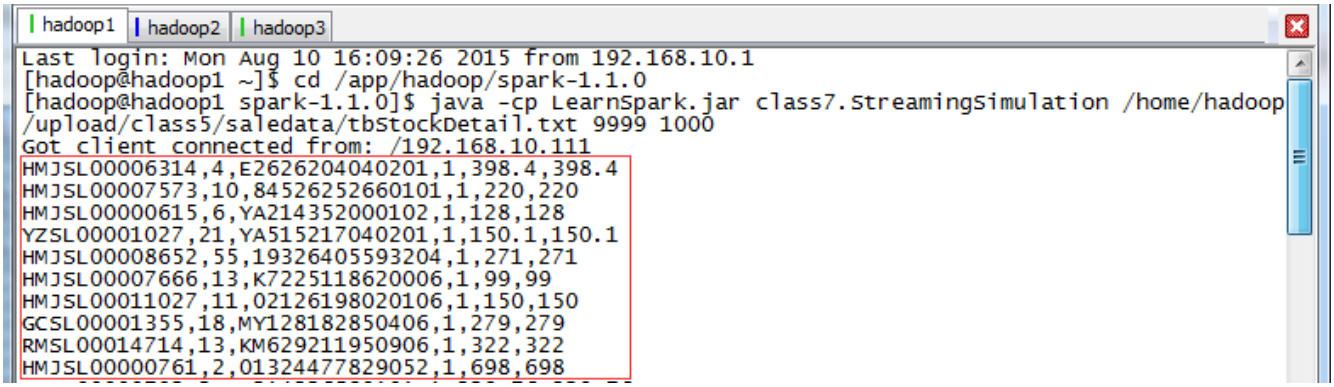
在 IDEA 中运行该实例，该实例需要配置连接 Socket 主机名和端口号，在这里配置参数机器名为 hadoop1 和端口号为 9999



1.4.4 查看结果

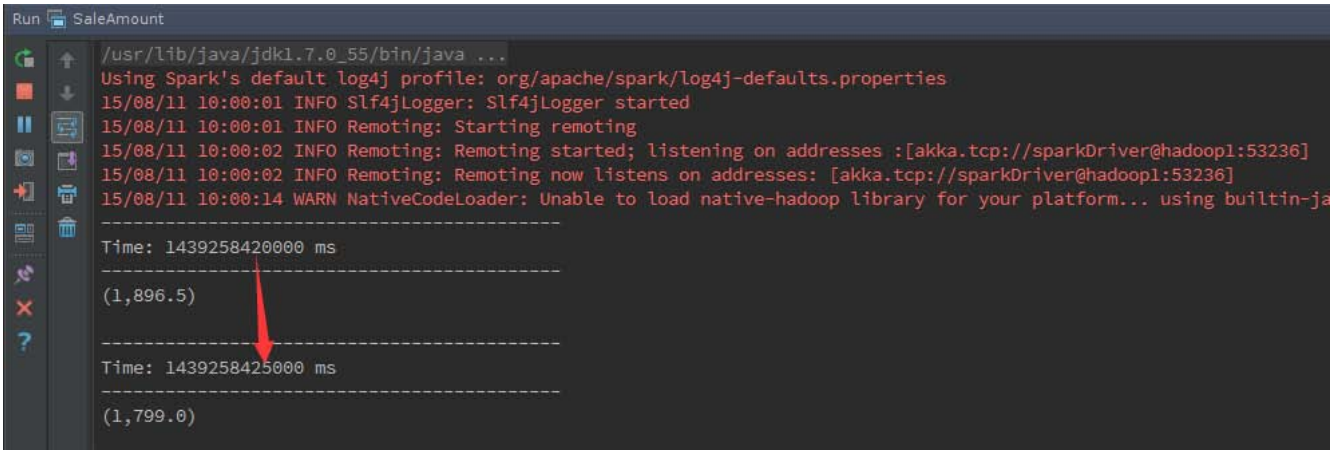
第一步 观察模拟器发送情况

IDEA 中的 Spark Streaming 程序运行与模拟器建立连接，当模拟器检测到外部连接时开始发送销售数据，时间间隔为 1 秒



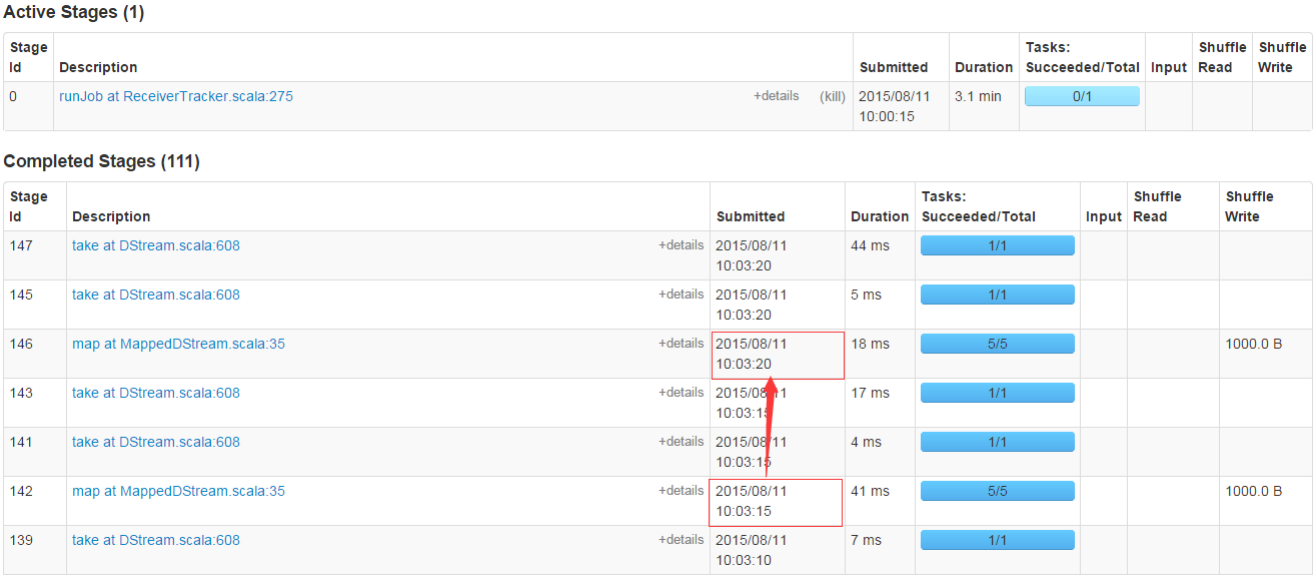
第二步 IDEA 运行情况

在 IDEA 的运行窗口中，可以观察到每 5 秒运行一次作业（两次运行间隔为 5000 毫秒），运行完毕后打印该时间段内销售数据总和。



第三步 在监控页面观察执行情况

在 webUI 上监控作业运行情况，可以观察到每 5 秒运行一次作业



1.5 实例 4：Stateful 演示

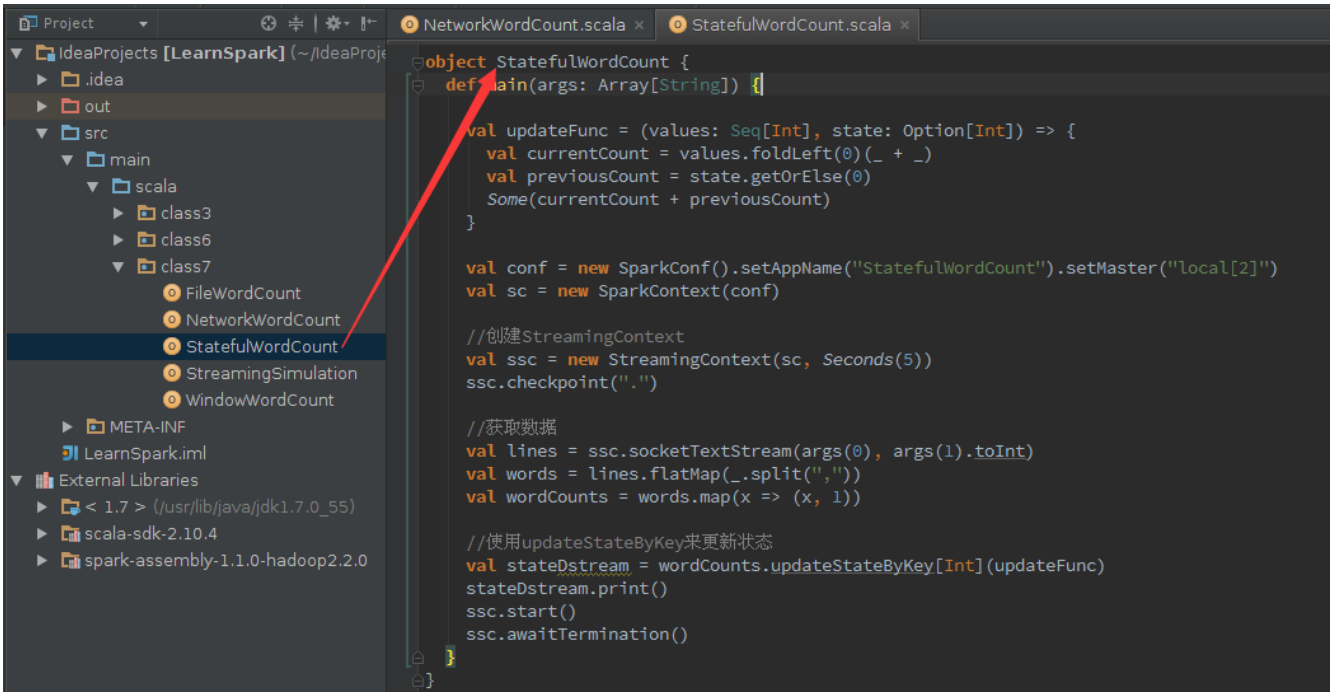
1.5.1 演示说明

该实例为 Spark Streaming 状态操作 ,模拟数据由 4.1 流数据模拟以 1 秒的频度发送 ,Spark Streaming 通过 Socket 接收流数据并每 5 秒运行一次用来处理接收到数据 ,处理完毕后打印程序启动后单词出现的频度，相比较前面 4.3 实例在该实例中各时间段之间状态是相关的。

1.5.2 演示代码

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.{SparkContext, SparkConf}
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._

object StatefulWordCount {
  def main(args: Array[String]) {
    if (args.length != 2) {
      System.err.println("Usage: StatefulWordCount <filename> <port> ")
      System.exit(1)
    }
    Logger.getLogger("org.apache.spark").setLevel(Level.ERROR)
    Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

    // 定义更新状态方法，参数 values 为当前批次单词频度，state 为以往批次单词频度
    val updateFunc = (values: Seq[Int], state: Option[Int]) => {
      val currentCount = values.foldLeft(0)(_ + _)
      val previousCount = state.getOrElse(0)
      Some(currentCount + previousCount)
    }

    val conf = new
   SparkConf().setAppName("StatefulWordCount").setMaster("local[2]")
    val sc = new SparkContext(conf)

    // 创建 StreamingContext，Spark Steaming 运行时间间隔为 5 秒
    val ssc = new StreamingContext(sc, Seconds(5))
```

```
    // 定义 checkpoint 目录为当前目录
    ssc.checkpoint(".")

    // 获取从 Socket 发送过来数据
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val words = lines.flatMap(_.split(","))
    val wordCounts = words.map(x => (x, 1))

    // 使用 updateStateByKey 来更新状态，统计从运行开始以来单词总的次数
    val stateDstream = wordCounts.updateStateByKey[Int](updateFunc)
    stateDstream.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```



1.5.3 运行代码

第一步 启动流数据模拟器

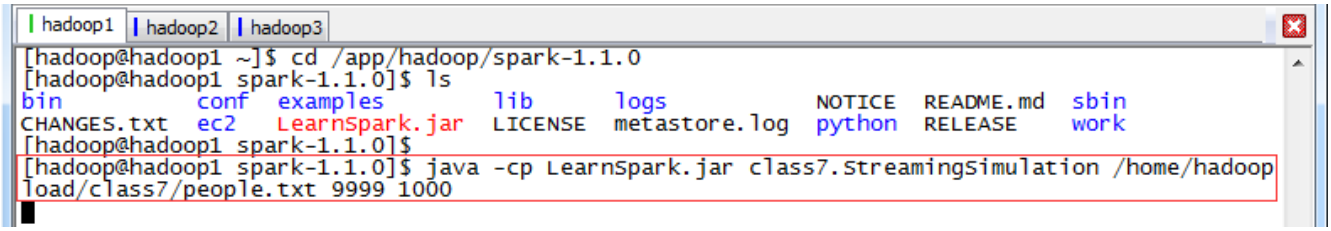
启动 4.1 打包好的流数据模拟器，在该实例中将定时发送/home/hadoop/upload/class7 目录下的 people.txt 数据文件（该文件可以在本系列配套资源目录/data/class7 中找到），其中 people.txt 数据内容如下：

```
1  Michael
2  Andy
3  Justin
4
```

模拟器 Socket 端口号为 9999，频度为 1 秒

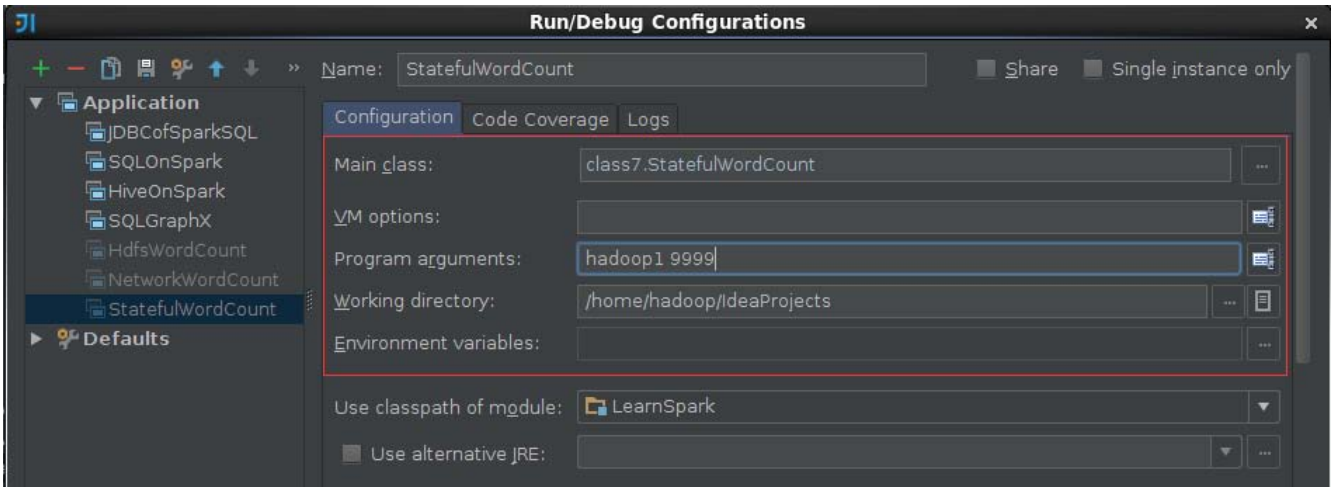
$cd /app/hadoop/spark-1.1.0

$java -cp LearnSpark.jar class7.StreamingSimulation /home/hadoop/upload/class7/people.txt 9999 1000



在没有程序连接时，该程序处于阻塞状态，在 IDEA 中运行 Streaming 程序

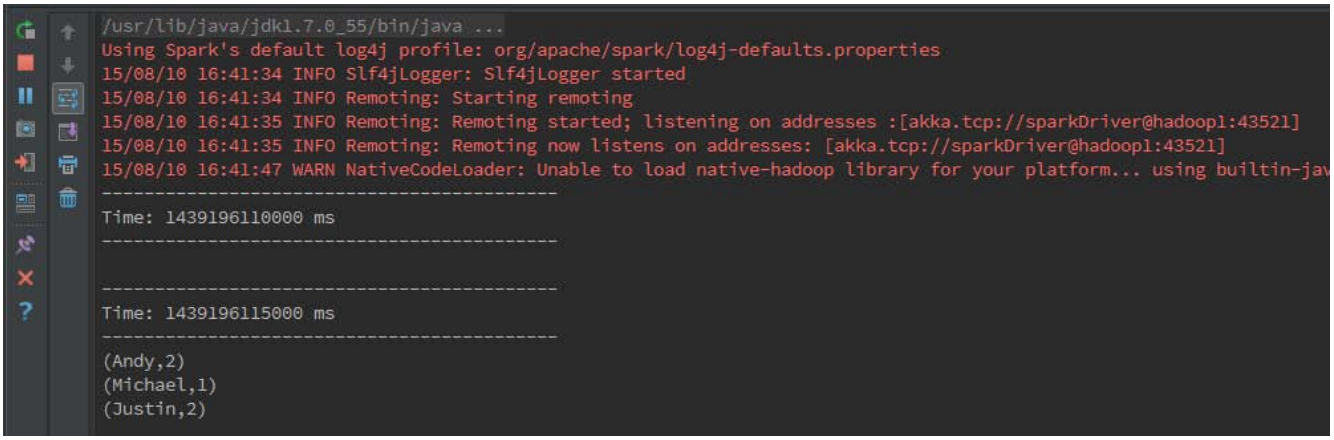
在 IDEA 中运行该实例，该实例需要配置连接 Socket 主机名和端口号，在这里配置参数机器名为 hadoop1 和端口号为 9999



1.5.4 查看结果

第一步 IDEA 运行情况

在 IDEA 的运行窗口中，可以观察到第一次运行统计单词总数为 1，第二次为 6，第 N 次为 5(N-1)+1，即统计单词的总数为程序运行单词数总和。



第二步 在监控页面观察执行情况

在 webUI 上监控作业运行情况，可以观察到每 5 秒运行一次作业

Active Stages (1)

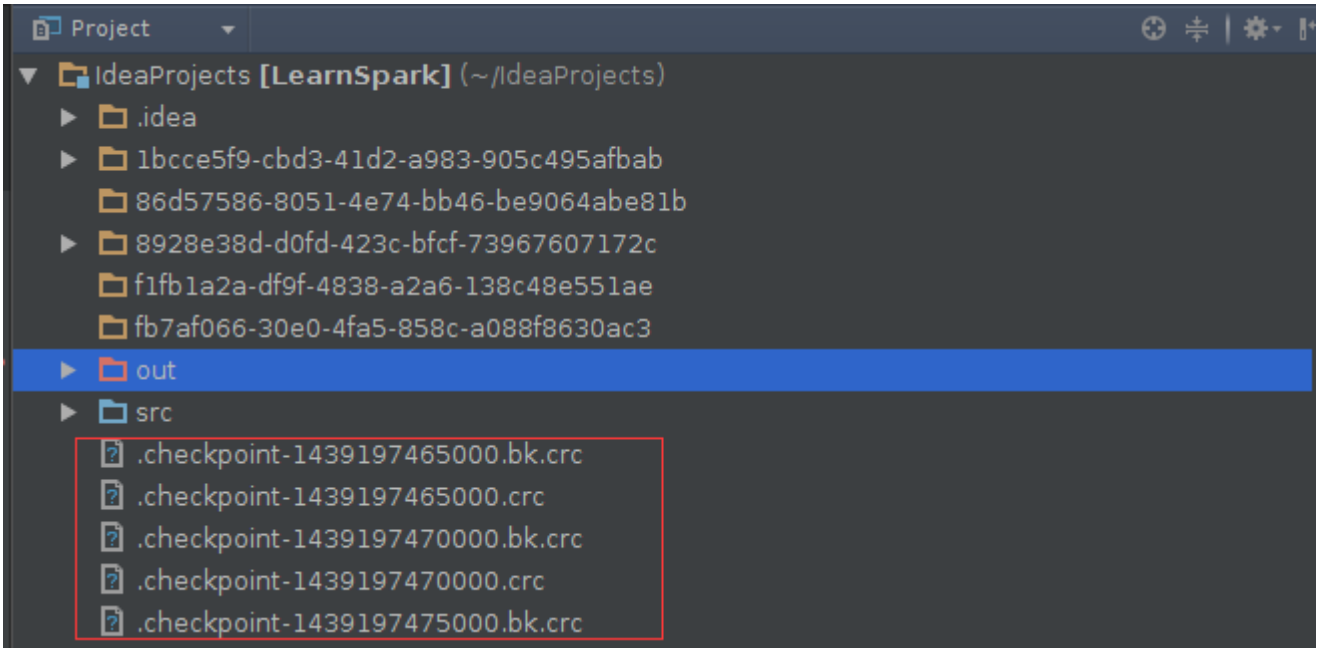
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
0	runJob at ReceiverTracker.scala:275 +details (kill)	2015/08/10 16:41:49	40 s	0/1			

Completed Stages (27)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
44	take at DStream.scala:608 +details	2015/08/10 16:42:25	3 ms	1/1	181.0 B		
41	take at DStream.scala:608 +details	2015/08/10 16:42:25	0.2 s	2/2	334.0 B		
38	take at DStream.scala:608 +details	2015/08/10 16:42:25	21 ms	1/1	153.0 B		
39	map at MappedDStream.scala:35 +details	2015/08/10 16:42:25	60 ms	5/5			853.0 B
36	take at DStream.scala:608 +details	2015/08/10 16:42:20	41 ms	1/1	181.0 B		
34	take at DStream.scala:608 +details	2015/08/10 16:42:20	18 ms	1/1	153.0 B		
35	map at MappedDStream.scala:35 +details	2015/08/10 16:42:20	57 ms	5/5			846.0 B
33	take at DStream.scala:608 +details	2015/08/10 16:42:15	15 ms	1/1	181.0 B		

第三步 查看 CheckPoint 情况

在项目根目录下可以看到 checkpoint 文件



1.6 实例 5：Window 演示

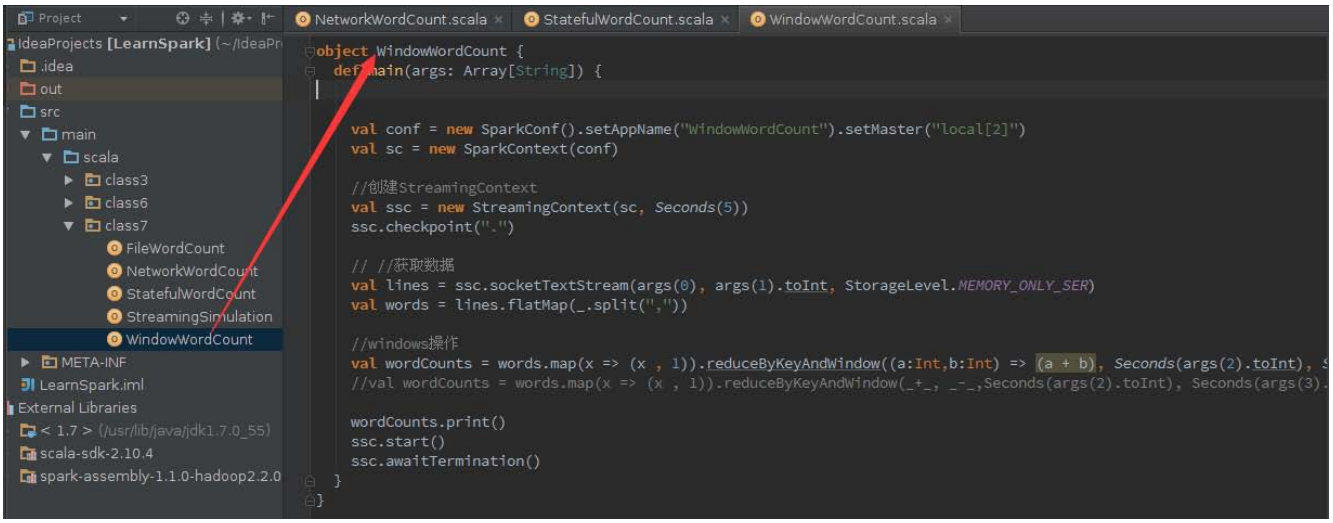
1.6.1 演示说明

该实例为 Spark Streaming 窗口操作 ,模拟数据由 4.1 流数据模拟以 1 秒的频度发送 ,Spark Streaming 通过 Socket 接收流数据并每 10 秒运行一次用来处理接收到数据，处理完毕后打印程序启动后单词出现的频度。相比前面的实例，Spark Streaming 窗口统计是通过 reduceByKeyAndWindow()方法实现的，在该方法中需要指定窗口时间长度和滑动时间间隔。

1.6.2 演示代码

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.{SparkContext, SparkConf}
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

object WindowWordCount {
  def main(args: Array[String]) {
    if (args.length != 4) {
      System.err.println("Usage: WindowWorldCount <filename> <port>
      <windowDuration> <slideDuration>")
      System.exit(1)
    }
    Logger.getLogger("org.apache.spark").setLevel(Level.ERROR)
    Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

    val conf = new
    SparkConf().setAppName("WindowWordCount").setMaster("local[2]")
    val sc = new SparkContext(conf)

     // 创建 StreamingContext
    val ssc = new StreamingContext(sc, Seconds(5))
     // 定义 checkpoint 目录为当前目录
    ssc.checkpoint(".")

    // 通过 Socket 获取数据，该处需要提供 Socket 的主机名和端口号，数据保存在内存和硬盘中
    val lines = ssc.socketTextStream(args(0), args(1).toInt, StorageLevel.MEMORY_ONLY_SER)
    val words = lines.flatMap(_.split(","))

    // windows 操作，第一种方式为叠加处理，第二种方式为增量处理
    val wordCounts = words.map(x => (x , 1)).reduceByKeyAndWindow((a:Int,b:Int) =>
    (a + b), Seconds(args(2).toInt), Seconds(args(3).toInt))
    //val wordCounts = words.map(x => (x , 1)).reduceByKeyAndWindow(_ +_,
    _ -_,Seconds(args(2).toInt), Seconds(args(3).toInt))
```

```
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```



1.6.3 运行代码

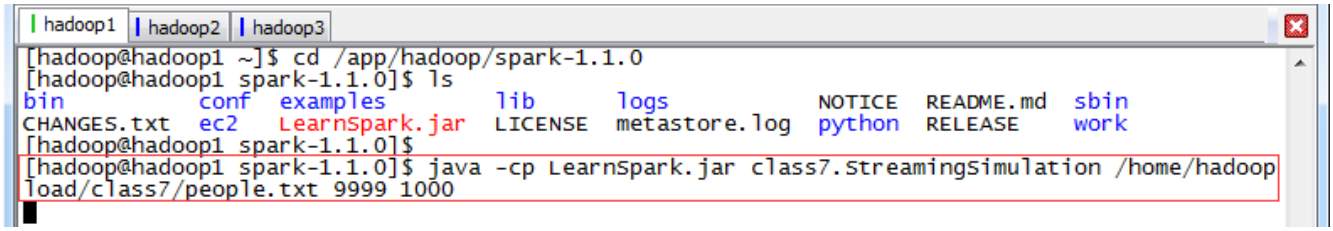
第一步 启动流数据模拟器

启动 4.1 打包好的流数据模拟器，在该实例中将定时发送/home/hadoop/upload/class7 目录下的 people.txt 数据文件（该文件可以在本系列配套资源目录/data/class7 中找到），其中 people.txt 数据内容如下：

```
Michael
Andy
Justin

```

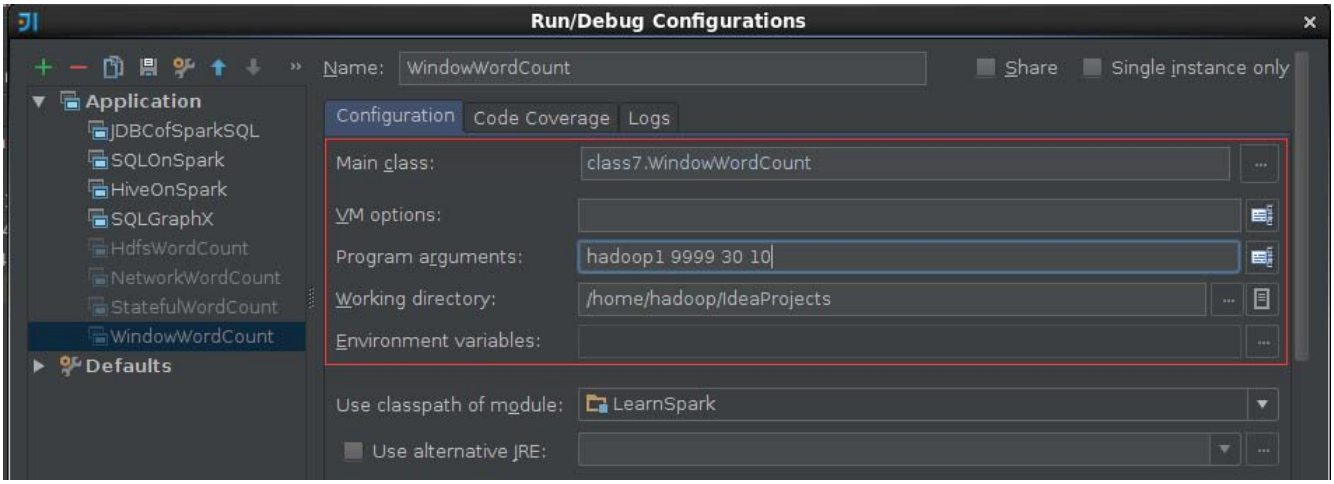
模拟器 Socket 端口号为 9999，频度为 1 秒

```
$cd /app/hadoop/spark-1.1.0
$java -cp LearnSpark.jar class7.StreamingSimulation /home/hadoop/upload/class7/people.txt 9999 1000
```



在没有程序连接时，该程序处于阻塞状态，在 IDEA 中运行 Streaming 程序

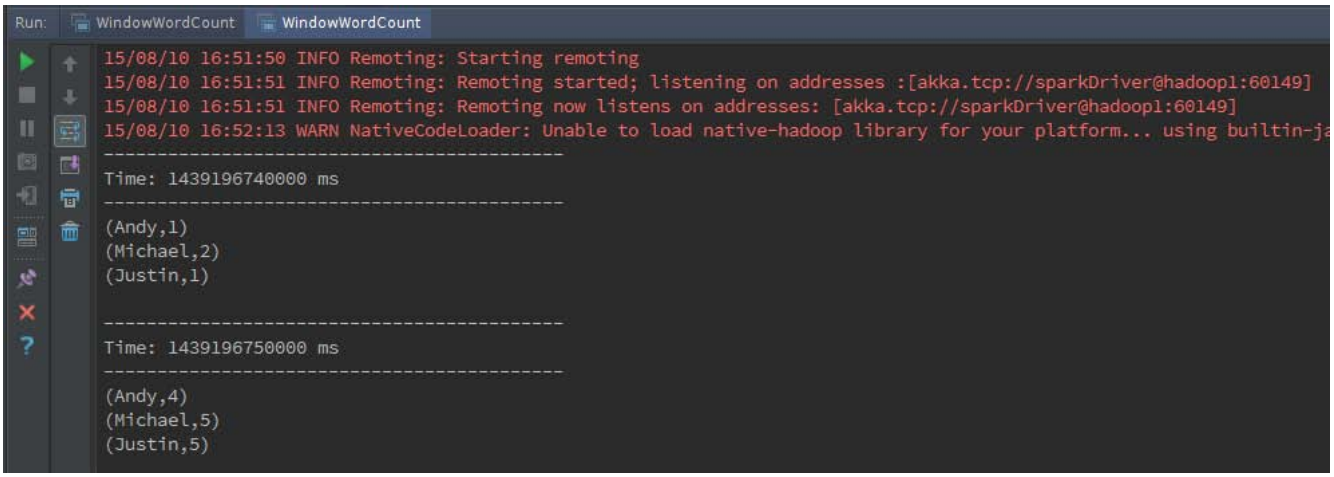
在 IDEA 中运行该实例，该实例需要配置连接 Socket 主机名和端口号，在这里配置参数机器名为 hadoop1、端口号为 9999、时间窗口为 30 秒和滑动时间间隔 10 秒



1.6.4 查看结果

第一步 IDEA 运行情况

在 IDEA 的运行窗口中，可以观察到第一次运行统计单词总数为 4，第二次为 14，第 N 次为 10(N-1)+4，即统计单词的总数为程序运行单词数总和。



第二步 在监控页面观察执行情况

在 webUI 上监控作业运行情况，可以观察到每 10 秒运行一次作业

Active Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
0	runJob at ReceiverTracker.scala:275 +details (kill)	2015/08/10 16:46:03	18 min	0/1			

Completed Stages (338)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
1494	take at DStream.scala:608 +details	2015/08/10 17:04:00	6 ms	1/1	4.0 B		
1487	take at DStream.scala:608 +details	2015/08/10 17:04:00	11 ms	1/1	4.0 B		
1489	map at MappedDStream.scala:35 +details	2015/08/10 17:04:00	53 ms	10/10			1692.0 B
1480	take at DStream.scala:608 +details	2015/08/10 17:03:50	5 ms	1/1	4.0 B		
1473	take at DStream.scala:608 +details	2015/08/10 17:03:50	10 ms	1/1	4.0 B		
1475	map at MappedDStream.scala:35 +details	2015/08/10 17:03:50	52 ms	10/10			1697.0 B

参考资料：

（1）《Spark Streaming》 http://blog.debugo.com/spark-streaming/