大数据平台技术课程实验报告

**实验六：Spark Streaming的操作实践**

姓名：\_\_杨佳森\_\_\_\_\_\_

学号：\_\_\_\_2112080106\_

专业：\_\_\_ 数据科学与大数据技术

年级：\_ 2021

主讲教师：\_ 刘春 \_

实验时间： 2023年5月26日14：00 至 6月2日

实验内容与要求：

[1](#_Toc43287683) Spark Streaming基本知识总结

2 Spark Streaming的操作实践：按照PPT内容进行实践

2.1 SparkStreaming的workCount案例

2.2 Spark Streaming与Flume和Kafka 的整合实践

**特别提醒：**

**（1）基本知识点的总结请使用自己的语言，根据自己的理解去总结，就像写课程笔记一样，不要去网上抄写**

**（2）实践需要有截图以及相应的文字介绍，欢迎大家把自己在实践过程中碰到的问题以及解决方法也记录下来**

# Spark Streaming基本知识的总结

## 1.常见的流计算框架。

### Storm：

最早是由Nathan Marz和他的团队于2010年在数据分析公司BackType开发。

2011年BackType公司被Twitter收购，接着Twitter开源Storm

2014年成为Apache顶级项目。

Storm被业界称为实时版的Hadoop，它与Hadoop、Spark并称为Apache基金会三大顶级的开源项目，是当前流计算技术中的佼佼者和主流。

它将数据流中的数据以元组的形式不断的发送给集群中的不同节点进行分布式处理，能够实现真正的实时处理。

### Spark streaming:

Spark Streaming是Spark软件栈中的一个用于流计算的组件。

在2014年发布的Spark1.0版本中，Spark Streaming已经包含在Spark软件栈中。

它基于Spark的核心批处理计算框架，通过将数据流沿时间轴分成不同的片段，然后交由Spark对不同片段的数据进行批处理来实现流式计算。所以，从严格意义上来说，Spark Streaming实现的并不是流式计算，具有一定的时间延迟，无法做到毫秒级的响应。但是由于Spark处理速度快，Spark Streaming也能够胜任和满足许多场景下的流计算需求。

## 2.Dstream和Dstream graph

Dstream： Discretized Stream，离散数据流。

在编写Spark Streaming应用程序时无需关心具体的RDD，只需要在数据流抽象层次去编写应用逻辑即可。此时，我们定义的Dstream操作所形成的Dstream对象之间的依赖关系被称为Dstream graph。

## 3.创建Dstream对象的数据源。

基础数据源：

这类数据源如文件系统和socket， StreamingContext中有具体的API来支持从这些数据源创建数据流对象。

高级数据源：

这类数据源如Kafka、Flume、HDFS等，我们需要配置额外的依赖来使用它们。

自定义数据源：

Spark Streaming也支持以自定义的方式来创建输入DStream。在这种情况下，我们需要自定义receiver，以便从自定义的数据源接收数据。

## 4.Dstream支持的操作

转换操作、窗口操作、输出操作

## 5.Spark streaming的结构与执行流程



# Spark Streaming的操作实践

## 2.1 SparkStreaming的workCount案例

### 以Socket为数据源

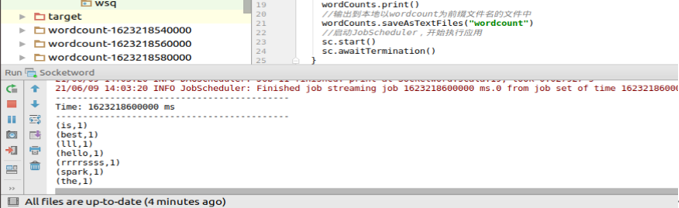
#### Pom.xml文件



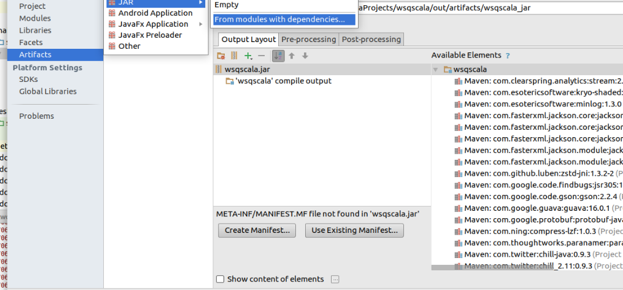
#### Scala Class文件



#### 在idea上运行上述程序



#### 打包集群运行上述程序





### 以文件目录为数据源

Pom.xml文件与前面一致

Scala class文件



先在idea端运行文件，然后再进入/home/yycw/hadoop/data目录下面，vim stm.txt在里面输入数据流，再保存，就可以在idea端查看输出的结果



### 源代码

**pom.xml文件：**

**<project xmlns="http://maven.apache.org/POM/4.0.0"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">**

**<modelVersion>4.0.0</modelVersion>**

**<groupId>com.liu</groupId>**

**<artifactId>socketSparkStreaming</artifactId>**

**<version>1.0-SNAPSHOT</version>**

**<dependencies>**

**<!-- 对spark core的依赖 -->**

**<dependency>**

**<groupId>org.apache.spark</groupId>**

**<artifactId>spark-core\_2.11</artifactId>**

**<version>2.4.5</version>**

**</dependency>**

**<!-- 对spark streaming的依赖 -->**

**<dependency>**

**<groupId>org.apache.spark</groupId>**

**<artifactId>spark-streaming\_2.11</artifactId>**

**<version>2.4.5</version>**

**</dependency>**

**</dependencies>**

**</project>**

**Scala class文件：**

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

**object SocketWordCount {**

**def main(args :Array[String]) = {**

**//创建一个streamingContext对象，在本地运行，两个线程**

**//设置划分数据流为片段的时间间隔为20秒**

**val sc = new StreamingContext("local[2]", "socketWordCount", Seconds(20) )**

**// 创建一个数据流对象，连接到serverIP:serverPort, 比如 localhost:9999**

**val lines = sc.socketTextStream("localhost", 9999)**

**//将输入数据流中的每一行以空格为分隔符分割为单词**

**val words = lines.flatMap(line=>line.split(" "))**

**//统计一个时间片内的单词个数**

**ValwordCounts=words.map(word=>(word,1)).reduceByKey((a,b)=>a+b)**

**//将每个时间片中的前10个单词打印到控制台**

**wordCounts.print()**

**//输出到本地以wordcount为前缀文件名的文件中**

**wordCounts.saveAsTextFiles("wordcount")**

**//启动JobScheduler，开始执行应用**

**sc.start()**

**sc.awaitTermination()**

**}**

**}**

1. **以文件目录为数据源**

**Pom.xml与以socket为数据源的相同**

**Scala class文件：**

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

**object SocketWordCount {**

**def main(args :Array[String]) = {**

**//创建一个streamingContext对象，在本地运行，两个线程**

**//设置划分数据流为片段的时间间隔为20秒**

**val sc = new StreamingContext("local[2]", "fileWordCount", Seconds(20) )**

**// 创建一个数据流对象，连接到一个文件目录**

**val lines = sc.textFileStream(file:///home/hadoop/data)**

**//将输入数据流中的每一行以空格为分隔符分割为单词**

**val words = lines.flatMap(line=>line.split(" "))**

**//统计一个时间片内的单词个数**

**valwordCounts=words.map(word=>(word,1)).reduceByKey((a,b)=>a+b)**

**//将每个时间片中的前10个单词打印到控制台**

**wordCounts.print()**

**//输出到本地以wordcount为前缀文件名的文件中**

**wordCounts.saveAsTextFiles("wordcount")**

**//启动JobScheduler，开始执行应用**

**sc.start()**

**sc.awaitTermination()**

**}**

**}**

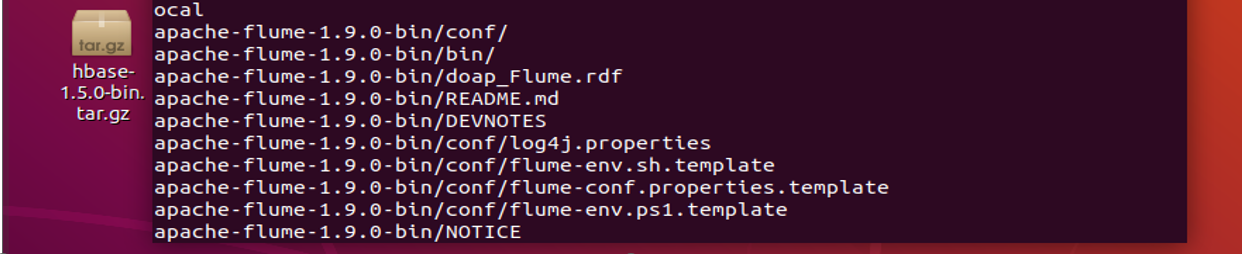
## 2.2 Spark Streaming与Flume和Kafka 的整合实践

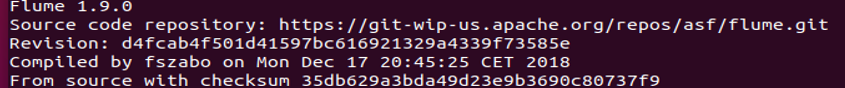
### Spark Streaming与Flume的结合

#### Flume的数据传输方式的简单了解

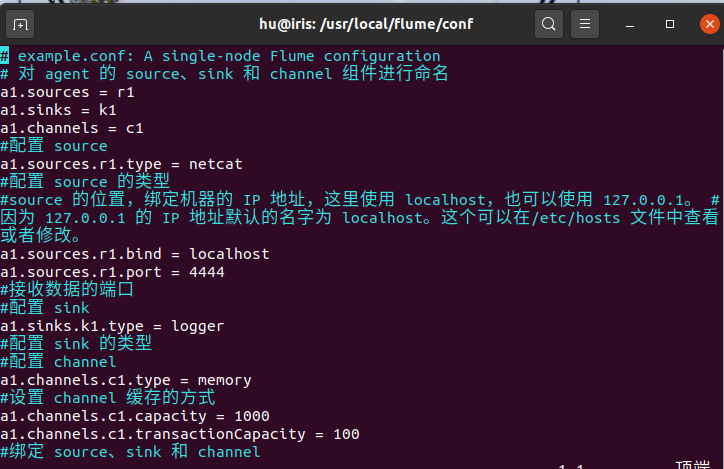
以event进行传输，Flume将传输的数据封装为event，event是Flume内部数据传输的最基本单元。一个event包括了event headers、event body，其中event body就包含了Flume收集和传输的日志信息，保证可靠传输的缓存机制。在event从source流向channel再到sink的过程中，为了保证数据传输的可靠性，event在送达sink之前，会在channel中进行缓存，直到event可靠到达sink之后，channel才会删除缓存的数据。

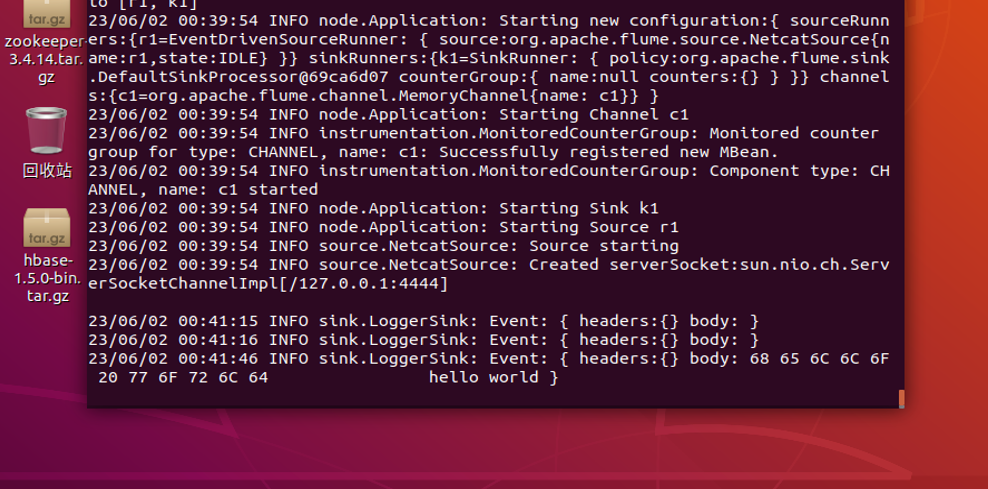
#### Flume的安装部署





#### Flume的使用示例

****

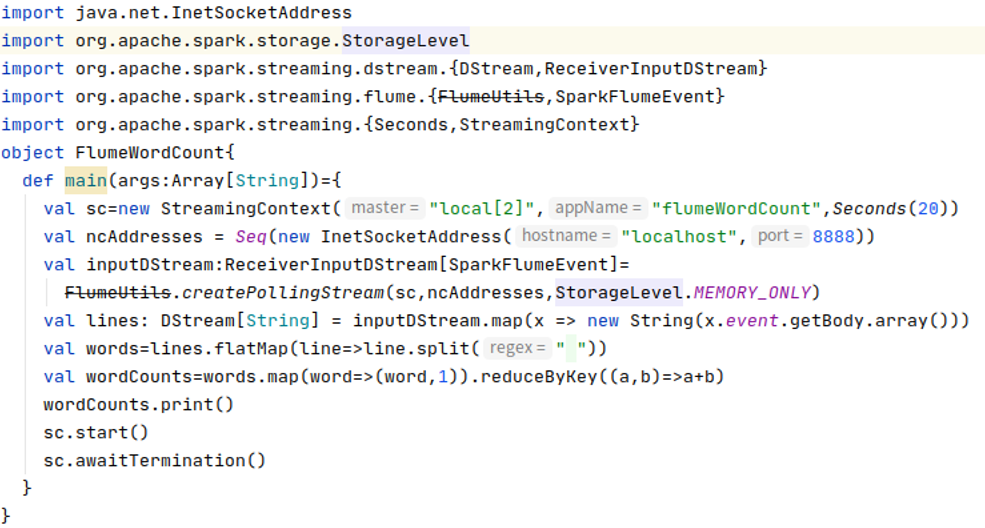


#### Flume与Spark Streaming整合

##### Pom文件



##### Scala Class文件



##### 运行结果



### Spark Streaming与FKafka 的整合

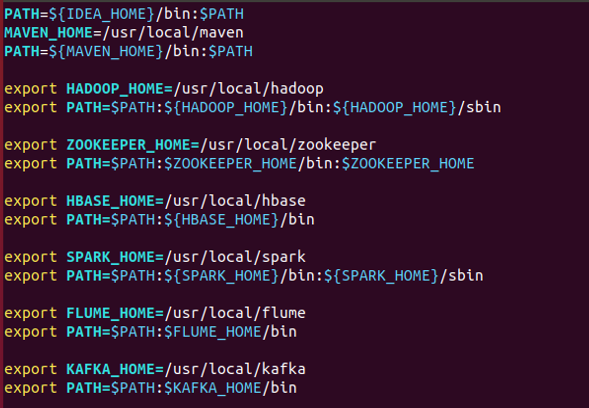
#### Kafka的介绍

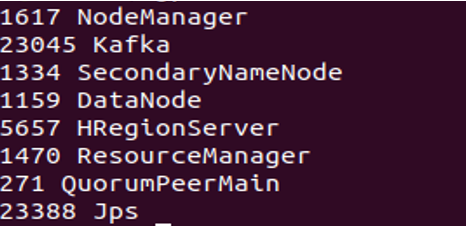
Kafka按照消息的主题topic将消息分类存储。消息的生产者在向Kafka集群写入消息的时候，必须指定消息的主题。Kafka又将一个主题下的消息分为多个分区partition。

当消息的生产者在写入一条指定主题的消息时，会根据一定的分配策略比如基于key值的hash策略将该条消息发送到该主题下的某个分区。

该条消息会以日志的形式追加到该分区当前记录的后面，并按照写入的时间顺序分配一个单调递增的顺序编号。这个编号也称为offset，是该条消息在分区内的ID，所以，每个主题下的分区是一个有序的消息序列。

#### Kafka的安装部署

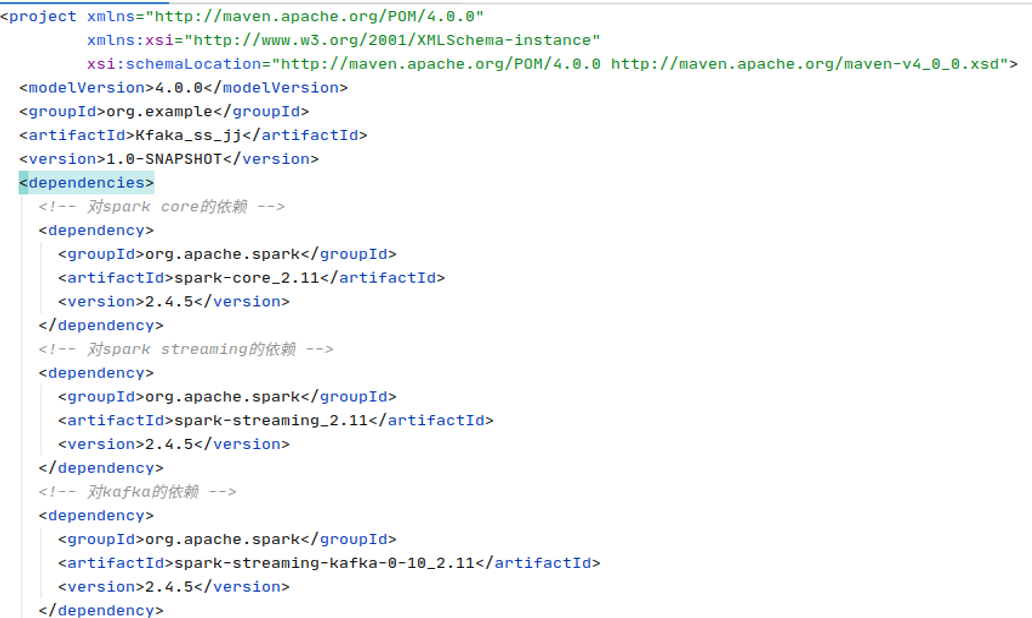




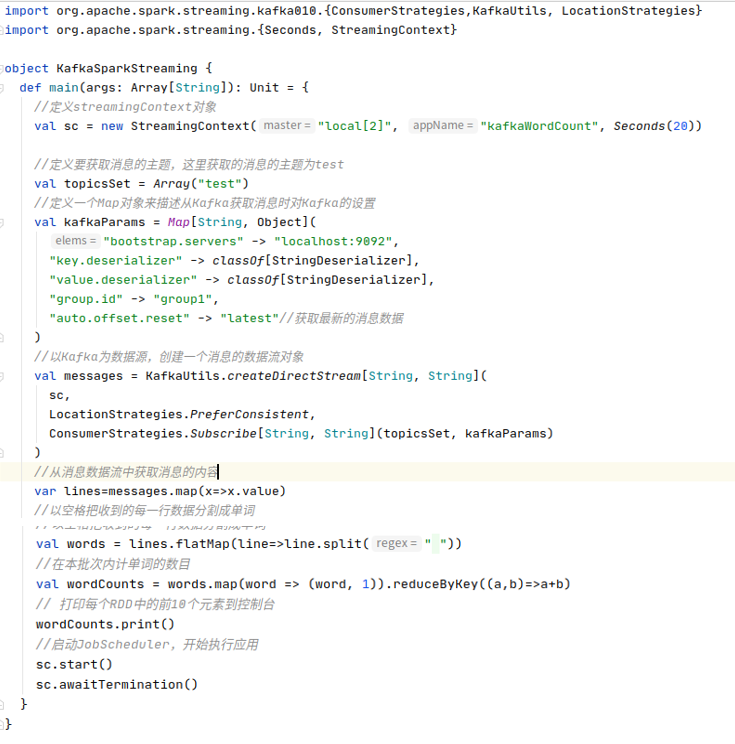
#### Kafka与Spark Streaming整合

##### 在IDEA中新建一个基于Maven的word count项目

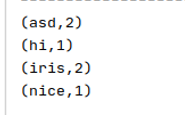
###### Pom.xml文件



###### Scala Class文件



###### 运行结果



### 源代码

**1.flume与spark streaming的整合**

配置和启动flume：

**# 对agent的source、sink和channel组件进行命名**

**a1.sources = r1**

**a1.sinks = k1**

**a1.channels = c1**

**#配置source**

**#配置source的类型**

**a1.sources.r1.type = netcat**

**#source的位置，绑定机器的IP地址，这里使用localhost，也可以使用127.0.0.1。**

**#因为127.0.0.1的IP地址默认的名字为localhost。这个可以在/etc/hosts文件中查看或者修改。**

**a1.sources.r1.bind = localhost**

**#接收数据的端口**

**a1.sources.r1.port = 4444**

**#配置sink，设置sink的类型为spark streaming自定义的类型**

**a1.sinks.k1.type = org.apache.spark.streaming.flume.sink.SparkSink**

**a1.sinks.k1.hostname=localhost**

**a1.sinks.k1.port = 8888**

**#配置channel**

**#设置channel缓存的方式**

**a1.channels.c1.type = memory**

**a1.channels.c1.capacity = 1000**

**a1.channels.c1.transactionCapacity = 100**

**#绑定source、sink和channel**

**a1.sources.r1.channels = c1**

**a1.sinks.k1.channel = c1**

创建word count项目，pom.xml文件：

**<project xmlns="http://maven.apache.org/POM/4.0.0"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">**

**<modelVersion>4.0.0</modelVersion>**

**<groupId>com.liu</groupId>**

**<artifactId>socketSparkStreaming</artifactId>**

**<version>1.0-SNAPSHOT</version>**

**<dependencies>**

**<!-- 对spark core的依赖 -->**

**<dependency>**

**<groupId>org.apache.spark</groupId>**

**<artifactId>spark-core\_2.11</artifactId>**

**<version>2.4.5</version>**

**</dependency>**

**<!-- 对spark streaming的依赖 -->**

**<dependency>**

**<groupId>org.apache.spark</groupId>**

**<artifactId>spark-streaming\_2.11</artifactId>**

**<version>2.4.5</version>**

**</dependency>**

**<dependency>**

**<groupId>org.apache.spark</groupId>**

**<artifactId>spark-streaming-flume\_2.11</artifactId>**

**<version>2.4.2</version>**

**</dependency>**

**</dependencies>**

**</project>**

Scala class文件：

**import java.net.InetSocketAddress**

**import org.apache.spark.storage.StorageLevel**

**import org.apache.spark.streaming.dstream.{DStream, ReceiverInputDStream}**

**import org.apache.spark.streaming.flume.{FlumeUtils, SparkFlumeEvent}**

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

**object FlumeWordCount {**

**def main(args :Array[String]) = {**

**//创建一个streamingContext对象，在本地运行，两个线程**

**//设置划分数据流为片段的时间间隔为20秒**

**val sc = new StreamingContext("local[2]", "flumeWordCount", Seconds(20) )**

**//定义一个flume的sink的机器和端口**

**val ncAddresses = Seq(new InetSocketAddress("localhost",8888))**

**//获取flume中的数据**

**val inputDstream:ReceiverInputDStream[SparkFlumeEvent]=**

**FlumeUtils.createPollingStream(sc, ncAddresses, StorageLevel.MEMORY\_ONLY)**

**//将flume输出的event中的数据取出，并转成字符串**

**val lines: DStream[String] = inputDstream.map(x => new String(x.event.getBody.array()))**

**//将输入数据流中的每一行以空格为分隔符分割为单词**

**val words = lines.flatMap(line=>line.split(" "))**

**//统计一个时间片内的单词个数**

**val wordCounts = words.map(word => (word, 1)).reduceByKey((a,b)=>a+b)**

**//将每个时间片中的前10个单词打印到控制台**

**wordCounts.print()**

**//启动JobScheduler，开始执行应用**

**sc.start()**

**sc.awaitTermination()**

**}**

**}**

**2.Kafka与spark streaming的整合**

pom.xml文件：

**<project xmlns="http://maven.apache.org/POM/4.0.0"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">**

**<modelVersion>4.0.0</modelVersion>**

**<groupId>com.liu</groupId>**

**<artifactId>socketSparkStreaming</artifactId>**

**<version>1.0-SNAPSHOT</version>**

**<dependencies>**

**<!-- 对spark core的依赖 -->**

**<dependency>**

**<groupId>org.apache.spark</groupId>**

**<artifactId>spark-core\_2.11</artifactId>**

**<version>2.4.5</version>**

**</dependency>**

**<!-- 对spark streaming的依赖 -->**

**<dependency>**

**<groupId>org.apache.spark</groupId>**

**<artifactId>spark-streaming\_2.11</artifactId>**

**<version>2.4.5</version>**

**</dependency>**

**<!-- 对kafka的依赖 -->**

**<dependency>**

**<groupId>org.apache.spark</groupId>**

**<artifactId>spark-streaming-kafka-0-10\_2.11</artifactId>**

**<version>2.4.5</version>**

**</dependency>**

**</dependencies>**

**</project>**

Scala class文件：

**import org.apache.kafka.common.serialization.StringDeserializer**

**import org.apache.spark.streaming.kafka010.{ConsumerStrategies,**

**KafkaUtils, LocationStrategies}**

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

**object KafkaSparkStreaming {**

**def main(args: Array[String]): Unit = {**

**//定义streamingContext对象**

**val sc = new StreamingContext("local[2]", "kafkaWordCount", Seconds(20) )**

**//定义要获取消息的主题，这里获取的消息的主题为test**

**val topicsSet = Array("test")**

**//定义一个Map对象来描述从Kafka获取消息时对Kafka的设置**

**val kafkaParams = Map[String, Object](**

**"bootstrap.servers" -> "localhost:9092",**

**"key.deserializer" -> classOf[StringDeserializer],**

**"value.deserializer" -> classOf[StringDeserializer],**

**"group.id" -> "group1",**

**"auto.offset.reset" -> "latest"//获取最新的消息数据**

**)**