大数据平台技术课程实验报告

**实验五：Spark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4月28日14：00 至 28日18：00

实验内容与要求：

[1](#_Toc43287683) Spark基本知识总结

2 Spark的操作实践：按照PPT内容进行实践

2.1 Spark的安装部署

2.2 Scala语言编程实践

2.3 基于Spark shell的WordCount实践

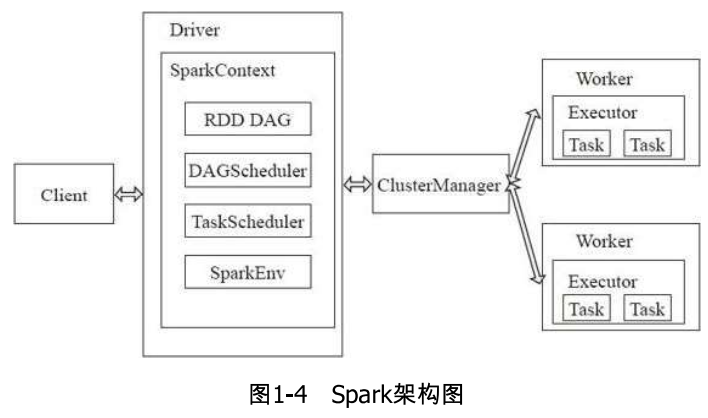
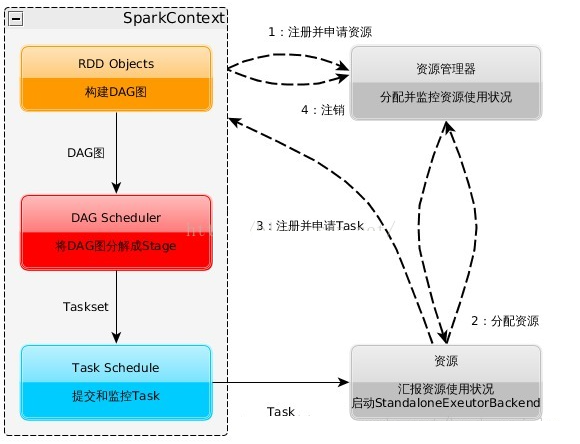
2.4 基于IDEA+Maven的Spark编程实践

2.5 pySpark实践

**特别提醒：**

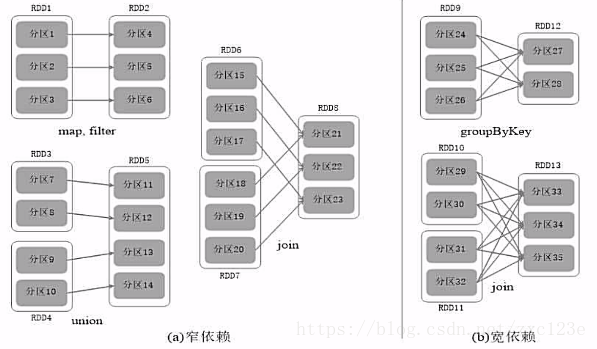
**（1）基本知识点的总结请使用自己的语言，根据自己的理解去总结，就像写课程笔记一样，不要去网上抄写**

**（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**

1. **Spark基本知识的总结**
2. 1、Hadoop、Spark、Storm三大框架比较
3. Hadoop:离线海量数据批处理,基于磁盘的
4. Spark：基于内存。
5. Spark特点：运行速度快，使用DAG执行引擎以支持循环数据流与内存计算
7. 2、容易使用：多种语言编程，通过spark shell进行交互式编程
8. 3、通用性：提供了完整而强大的技术栈，包括sQL查询、流式计算、机器学习和图算法组件
9. 4、运行模式多样：可运行在独立集群模式中，可以运行与hadoop中，也可以运行在AmazonEC2等云环境中，并可以访问HDFS、HBase、Hive等多种数据源
11. Scala：多范式编程语言
12. 函数式编程（lisp语言，Haskell语言）
13. 运行于java平台（jvm，虚拟机），兼容java程序
15. scala特性：具备强大的并发性，支持函数式编程，支持分布式系统，
16. 语法简洁，能提供优雅的API
17. scala兼容java，运行速度快，能融合到hadoop生态圈中。
18. scala是spark的主要编程语言，提供REPL（交互式解释器），提高程序开发效率
19. Spark与Hadoop的对比
21. hadoop的缺点： 1、表达能力有限，只能用map和reduce来表示
22. 2、磁盘开销大
23. 3、延迟高，由于要写磁盘，因此延迟高
24. 4、任务之间的衔接涉及IO开销
26. Spark相对于hadoop MapReduce的优点：
28. 1、不局限于MapReduce，提供多种数据集操作类型，编程模型比Hadoop MapReduce更灵活
29. 2、spark提供内存计算，可将中间结果放到内存中，对于迭代运算效率更高
30. 3、基于DAG的任务调度机制，效率更高
31. 4、已经发展成一个生态系统：遵循“一个软件栈满足不同应用场景”的设计理念
32. Spark与Hadoop的关系
33. 虽然Spark弥补了MapReduce的不足，并且已经发展成能够支持多种计算任务的一栈式解决方案，但是Spark无法完全取代Hadoop。
34. 反而，在使用Hadoop的YARN、HDFS和Hbase等组件来实现资源调度管理、海量数据存储、数据随机访问的同时，Spark已经很好的融入了Hadoop之中。
35. Spark运行架构及流程
36. 
37. 基本概念：
39. Application：用户编写的Spark应用程序。
40. Driver：Spark中的Driver即运行上述Application的main函数并创建SparkContext，创建SparkContext的目的是为了准备Spark应用程序的运行环境，在Spark中有SparkContext负责与ClusterManager通信，进行资源申请、任务的分配和监控等，当Executor部分运行完毕后，Driver同时负责将SparkContext关闭。
41. Executor：是运行在工作节点（WorkerNode）的一个进程，负责运行Task。
42. RDD：弹性分布式数据集，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型。
43. DAG：有向无环图，反映RDD之间的依赖关系。
44. Task：运行在Executor上的工作单元。
45. Job：一个Job包含多个RDD及作用于相应RDD上的各种操作。
46. Stage：是Job的基本调度单位，一个Job会分为多组Task，每组Task被称为Stage，或者也被称为TaskSet，代表一组关联的，相互之间没有Shuffle依赖关系的任务组成的任务集。
47. Cluter Manager：指的是在集群上获取资源的外部服务。目前有三种类型
48. 1) Standalon : spark原生的资源管理，由Master负责资源的分配
49. 2) Apache Mesos:与hadoop MR兼容性良好的一种资源调度框架
50. 3) Hadoop Yarn: 主要是指Yarn中的ResourceManager
51. 一个Application由一个Driver和若干个Job构成，一个Job由多个Stage构成，一个Stage由多个没有Shuffle关系的Task组成。
53. 当执行一个Application时，Driver会向集群管理器申请资源，启动Executor，并向Executor发送应用程序代码和文件，然后在Executor上执行Task，运行结束后，执行结果会返回给Driver，或者写到HDFS或者其它数据库中。
55. 与Hadoop MapReduce计算框架相比，Spark所采用的Executor有两个优点：
57. 利用多线程来执行具体的任务减少任务的启动开销；
58. Executor中有一个BlockManager存储模块，会将内存和磁盘共同作为存储设备，有效减少IO开销；
59. Spark运行基本流程：
60. 
61. 为应用构建起基本的运行环境，即由Driver创建一个SparkContext进行资源的申请、任务的分配和监控
62. 资源管理器为Executor分配资源，并启动Executor进程
63. SparkContext根据RDD的依赖关系构建DAG图，DAG图提交给DAGScheduler解析成Stage，然后把一个个TaskSet提交给底层调度器TaskScheduler处理。
64. Executor向SparkContext申请Task，TaskScheduler将Task发放给Executor运行并提供应用程序代码。
65. Task在Executor上运行把执行结果反馈给TaskScheduler，然后反馈给DAGScheduler，运行完毕后写入数据并释放所有资源。
66. Spark运行架构特点：
68. 每个Application都有自己专属的Executor进程，并且该进程在Application运行期间一直驻留。Executor进程以多线程的方式运行Task。
69. Spark运行过程与资源管理器无关，只要能够获取Executor进程并保存通信即可。
70. Task采用数据本地性和推测执行等优化机制。  
    RDD
71. 一个RDD就是一个分布式对象集合，本质上是一个只读的分区记录集合，每个RDD可分成多个分区，每个分区就是一个数据集片段，并且一个RDD的不同分区可以被保存到集群中不同的节点上，从而可以在集群的不同节点上进行并行计算。
73. RDD提供了一种高端受限的共享内存模型，即RDD是只读的记录分区的集合，不能直接修改，只能基于稳定的物理存储中的数据集创建RDD



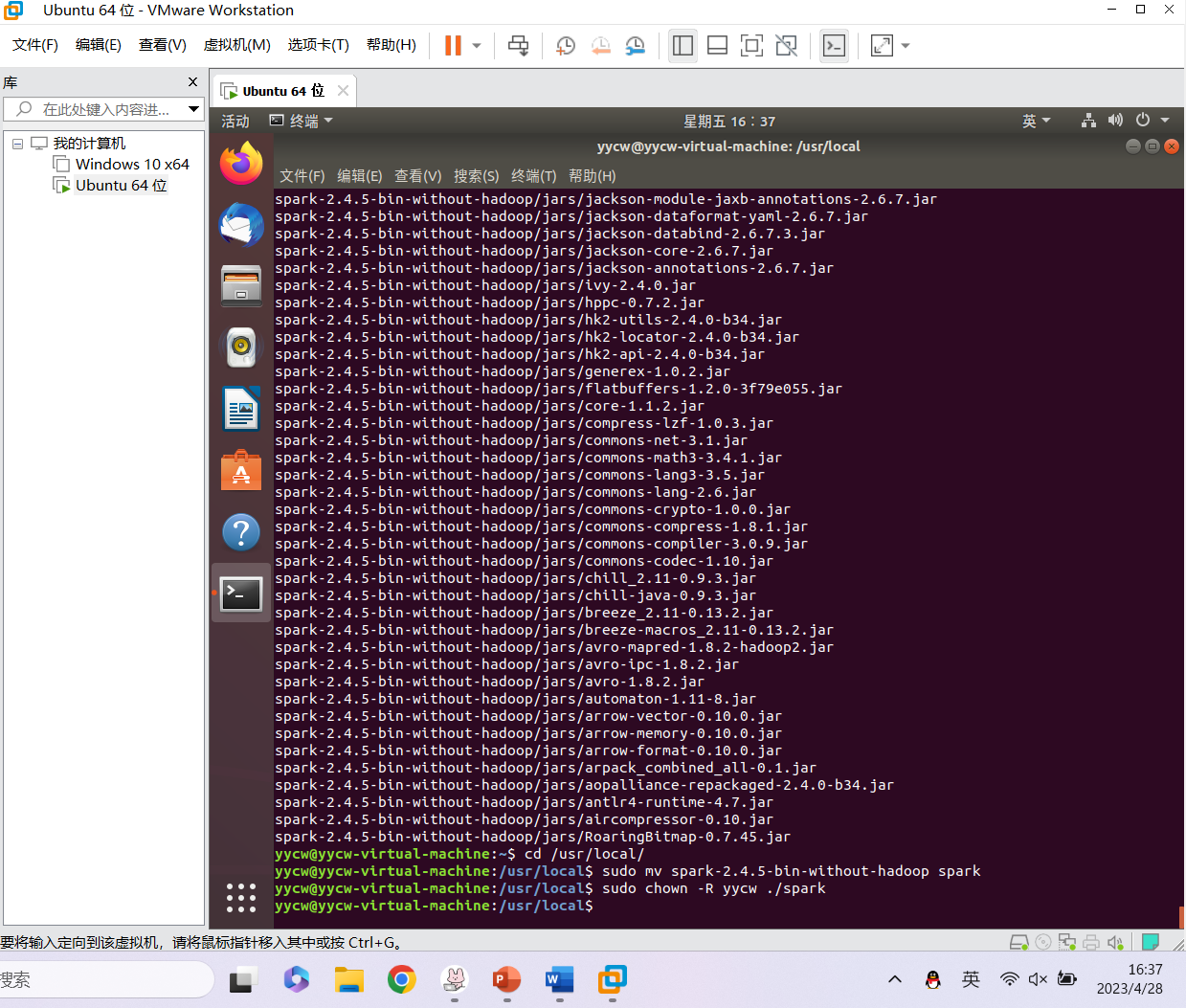


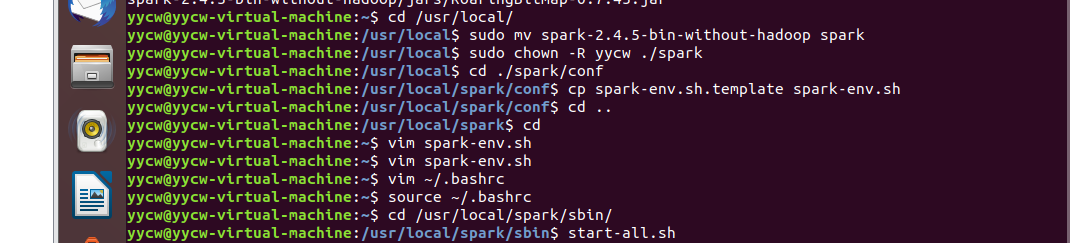
1. RDD读入外部数据源进行创建
2. RDD经过一系列的转换（Transformation）操作，没一次都会产生不同的RDD供下一个转换操作使用
3. 最后一个RDD经过“动作”操作进行转换并输出到外部数据源
4. 优点：惰性调用、管道化、避免同步等待，不需要保存中间结果。这和Java8中Stream的概念极其类似。
6. 窄依赖和宽依赖
7. 窄依赖：表现为一个父RDD的分区对应于一个子RDD的分区或者多个父RDD的分区对应于一个子RDD的分区。
8. 宽依赖：表现为存在一个父RDD的一个分区对应一个子RDD的多个分区。  
   

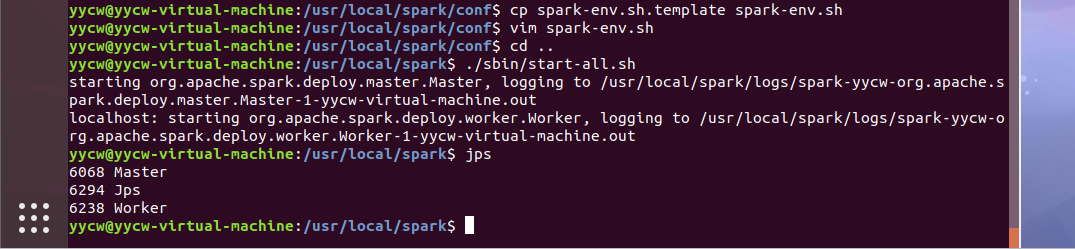


1. **Spark的操作实践**

**2.1 Spark的安装部署实践**

****

****

****

**2.2 Scala语言编程**

**Scala语言的变量和类型**

****

**数据的基本操作**

****

**定义一个类**

****

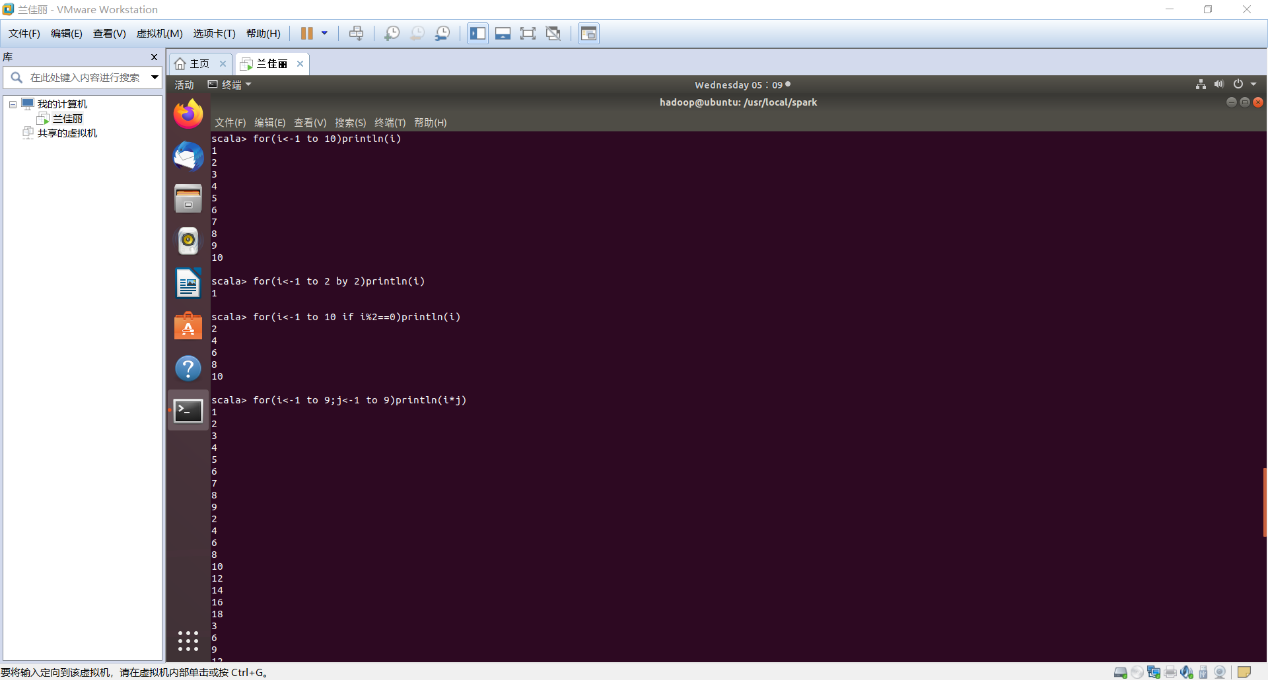
**Lambda表达式**

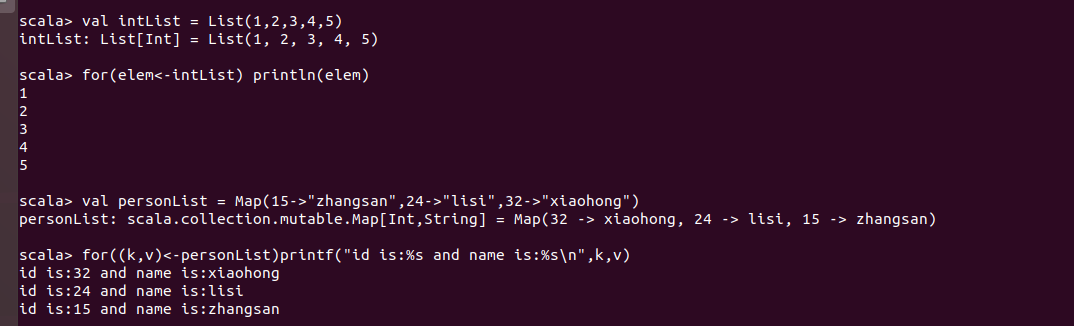
****

**Scala的控制结构**

****

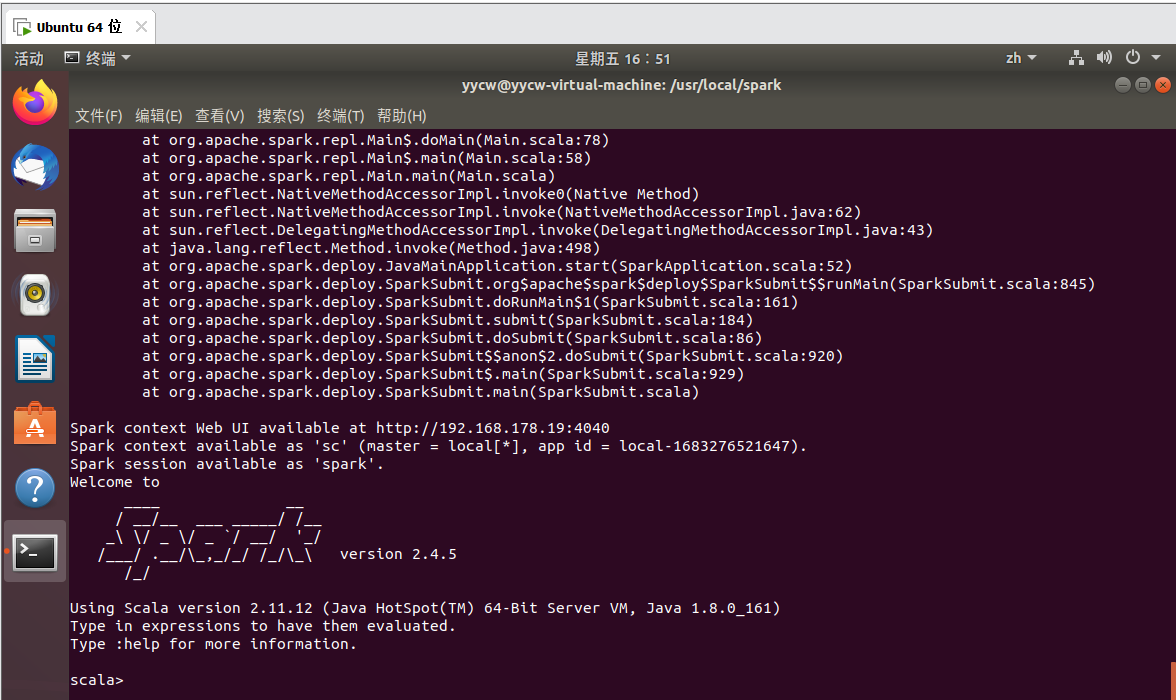
****

****

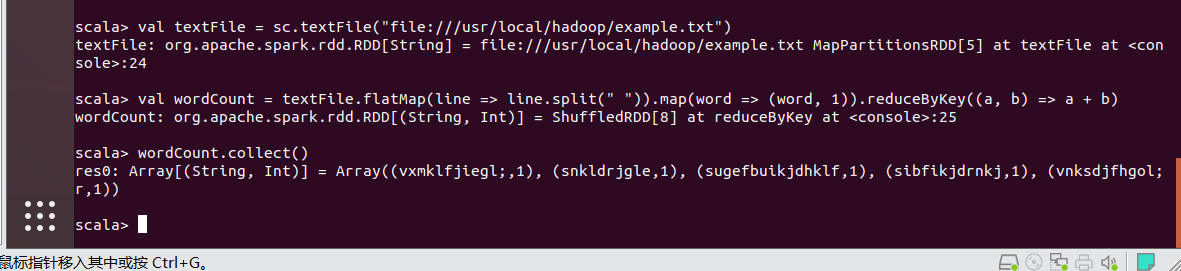
****

**2.3 基于Spark shell的WordCount实践**

**启动Spark shell**

****

**从本地文件读取数据进行wordCount**

****

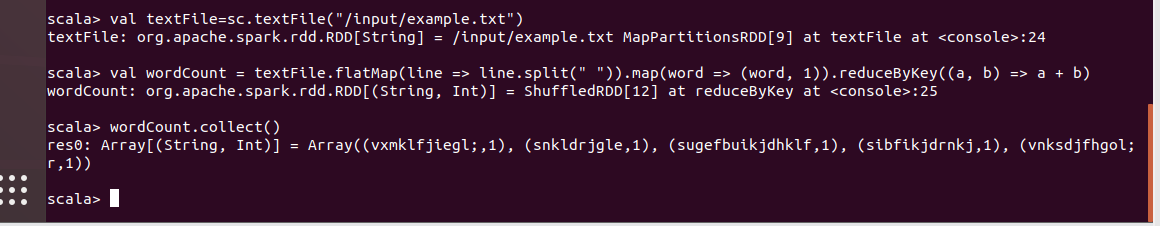
**从HDFS读取数据进行wordCount**

在输入val wordCount = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b)时报错

org.apache.hadoop.mapred.InvalidInputException: Input path does not exist: hdfs://localhost:9000/home/hadoop/example.txt

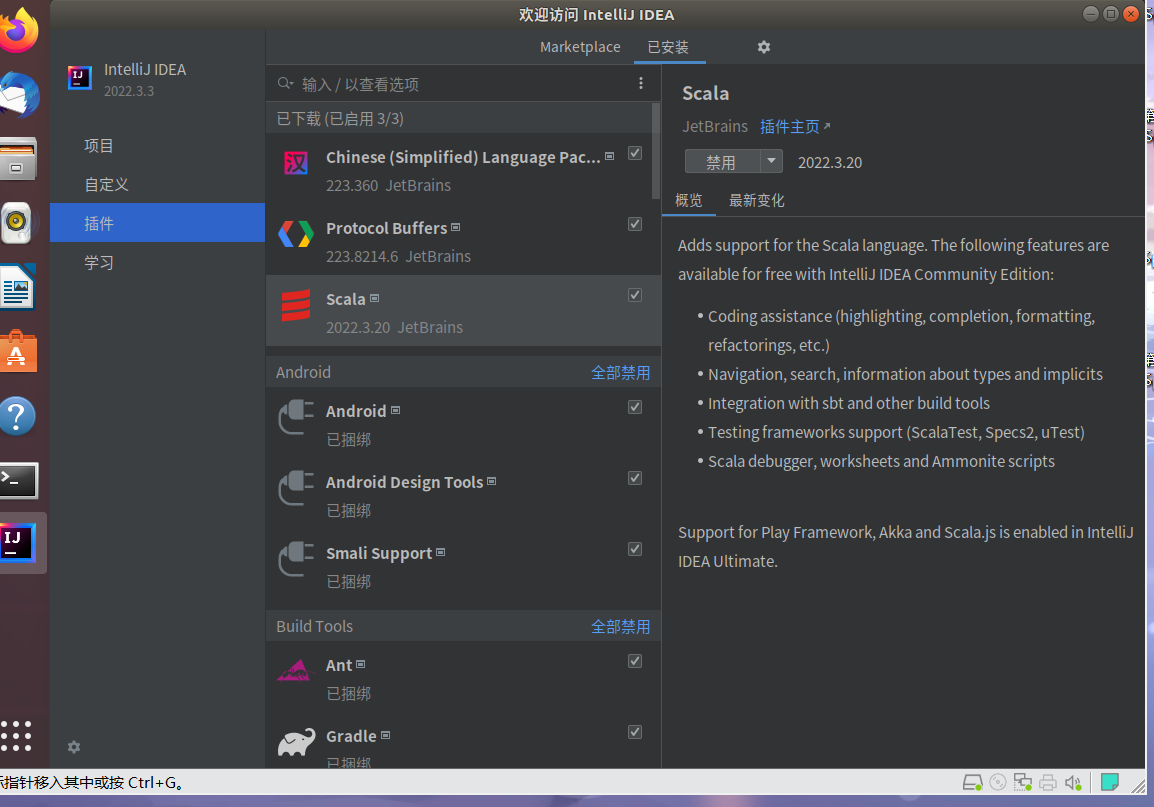
当时的example.txt并没有上传到HDFS中，导致访问失败

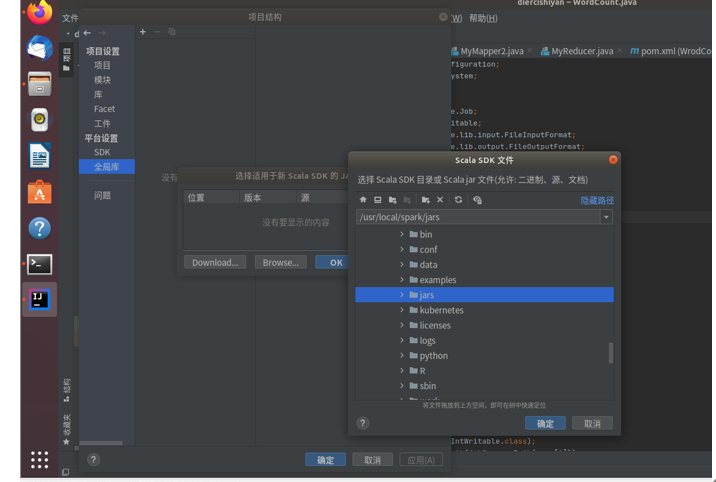
只需要将其上传即可hdfs dfs -put /usr/local/hadoop/example.txt /input



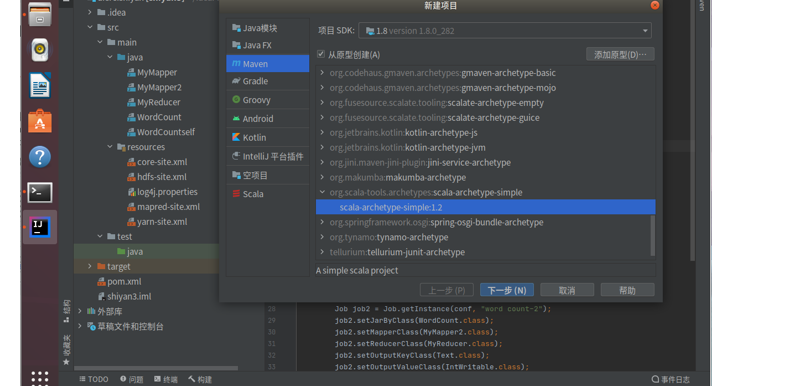
**2.4基于IDEA+Maven的Spark编程实践**

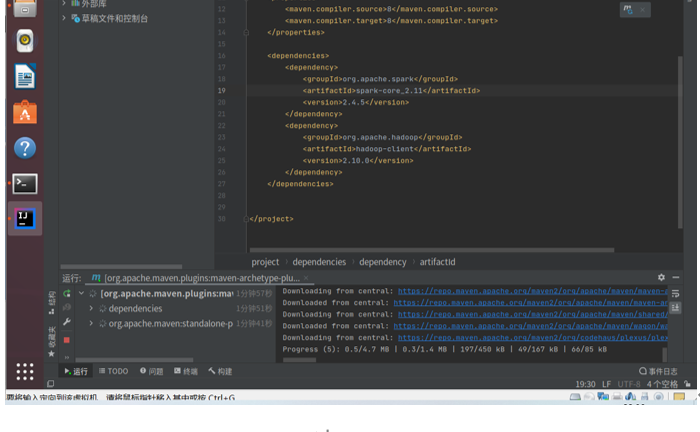
**在IDEA中安装Scala插件与SDK**

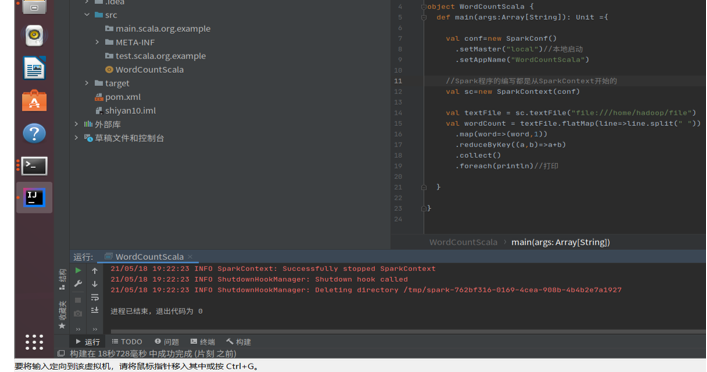
****

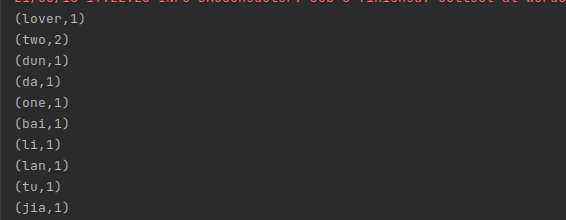
****

**基于Scala的WordCount Spark程序**

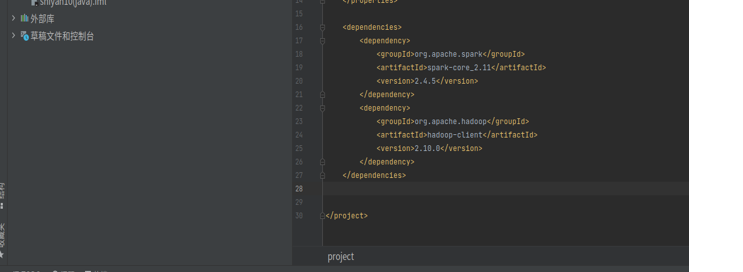
****

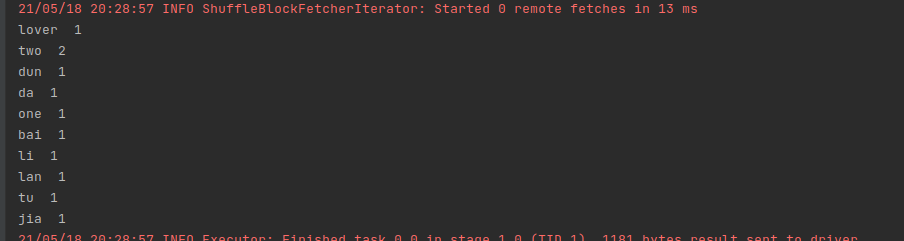
****

****

****

**基于java的WordCount Spark程序**

****

**Spark与Hbase的整合**

（1）、Hbase:一个高可靠、高性能、面向列、可伸缩的分布式数据库，主要用来存储非结构化和半结构化的松散数据

（2）、基于Scala语言新建一个maven项目来测试如何读写Hbase中的数据

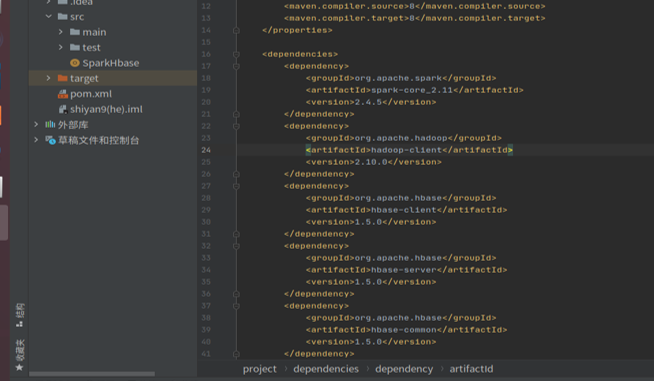
（3）、该项目将读取我们在介绍Hbase时创建的usr\_beha表中的数据

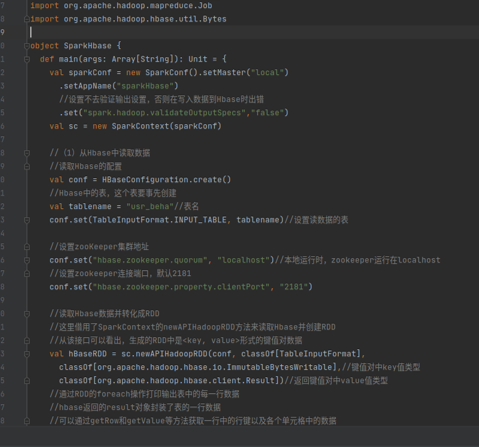
（4）、该表包含了两个列族：attr和beha。

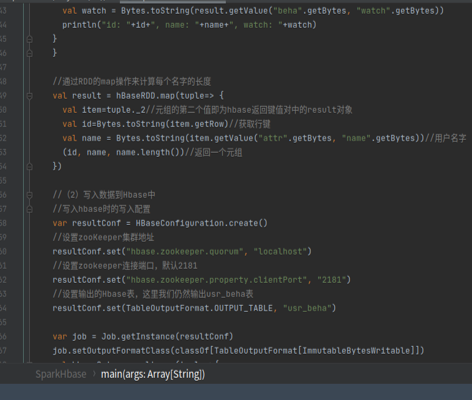
attr: attr列族主要存储用户属性数据，目前只包含了一个名为name的列

beha: beha列族主要存储用户的行为数据，目前只包含了一个名为watch的列

（5）、项目的任务：首先从该表中读取数据，然后计算每个名字的长度，并将长度作为一个新的列写入到attr列族下。



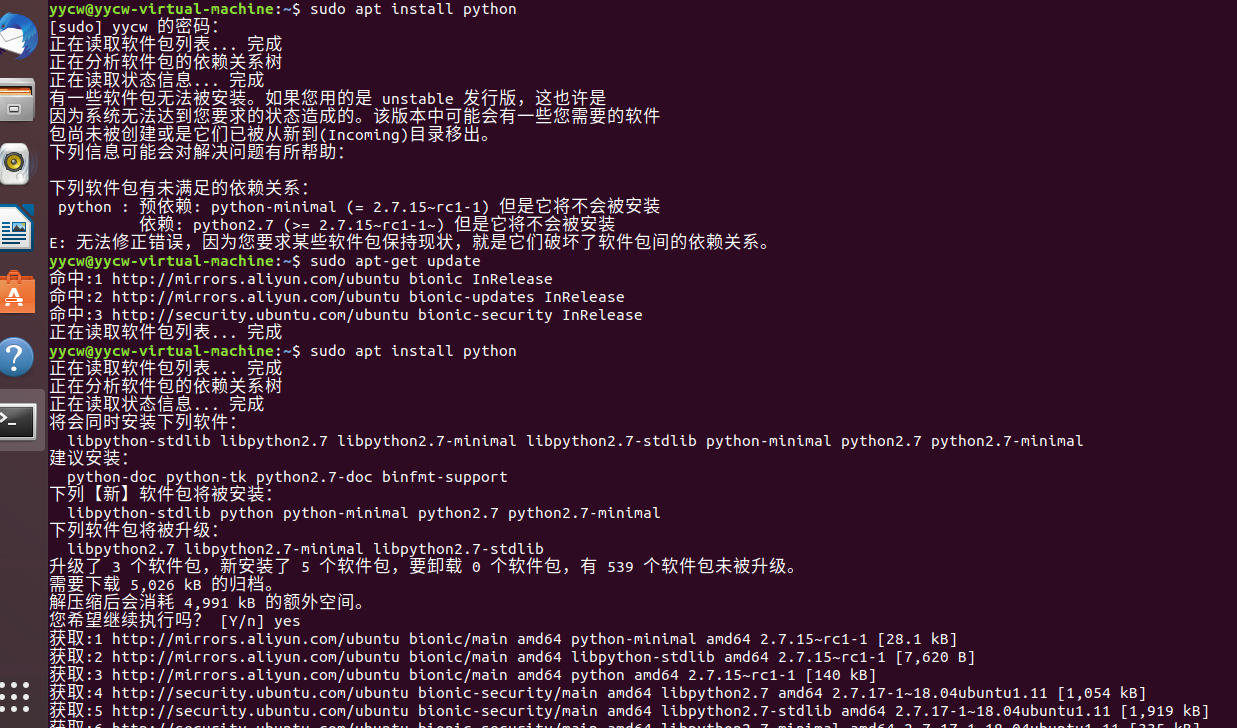




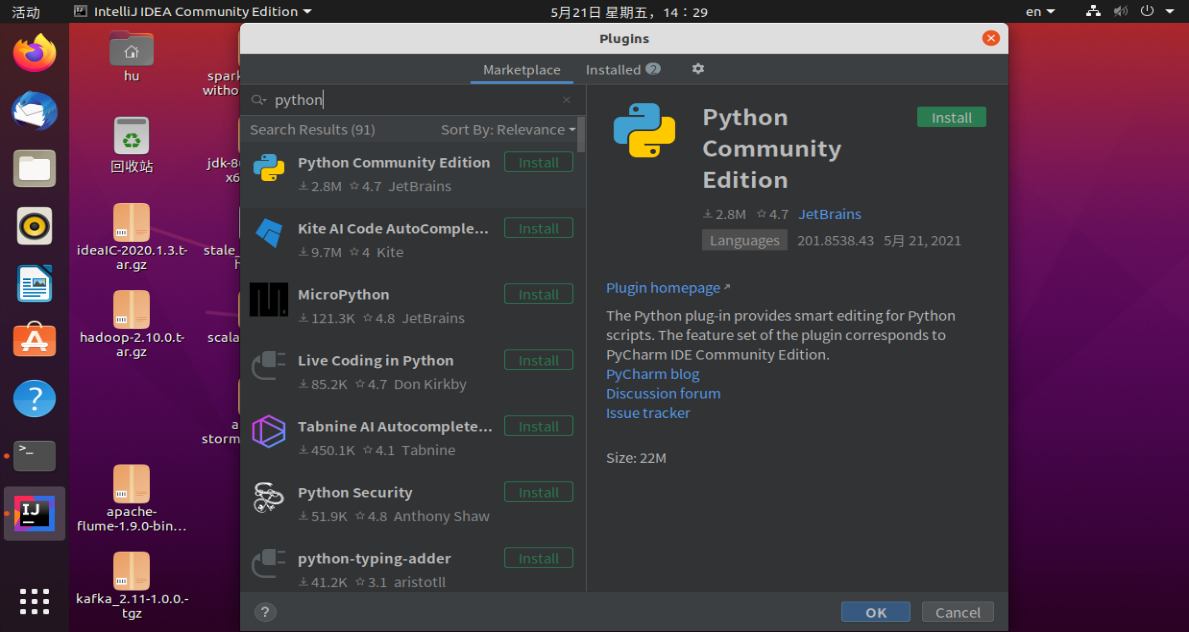
**2.5 pySpark实践**

**Python的安装**

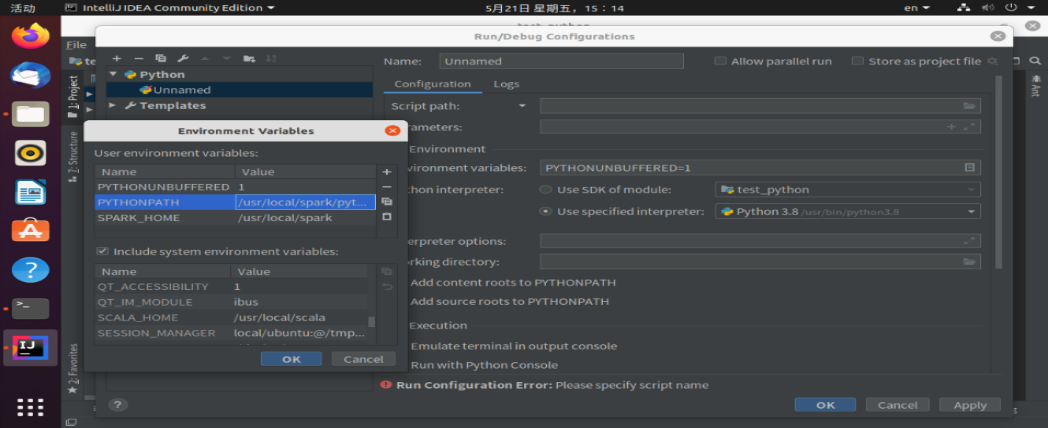
在安装python时遇到错误，找到之前更新的源，删除有问题的源，然后更新源 sudo apt-get update，最后安装ptyhon即可。

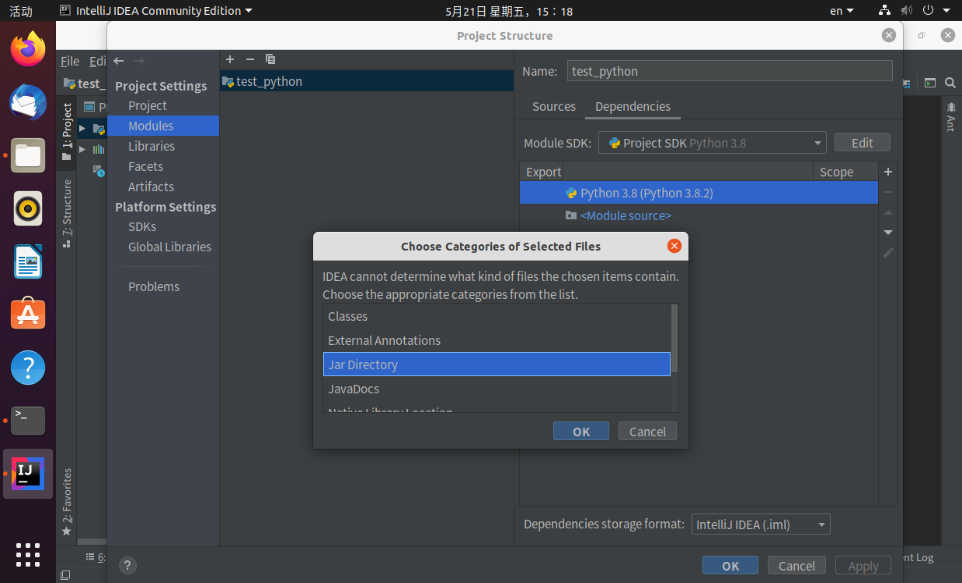
****

**Python编辑插件的安装**

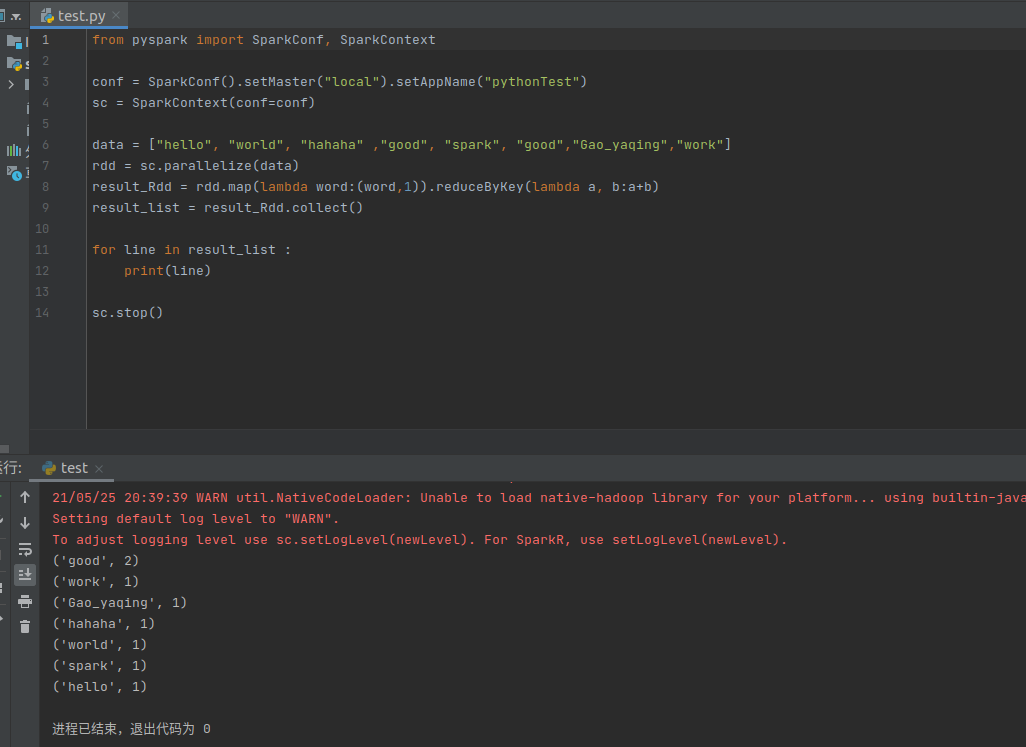
****

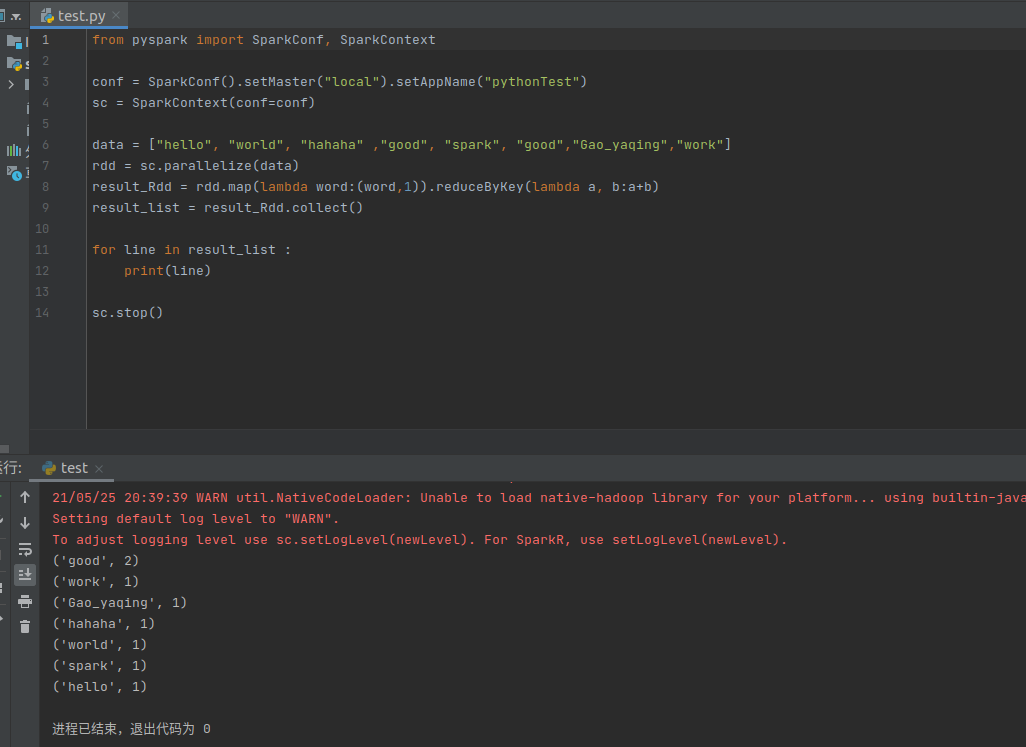
**配置python项目环境**

****

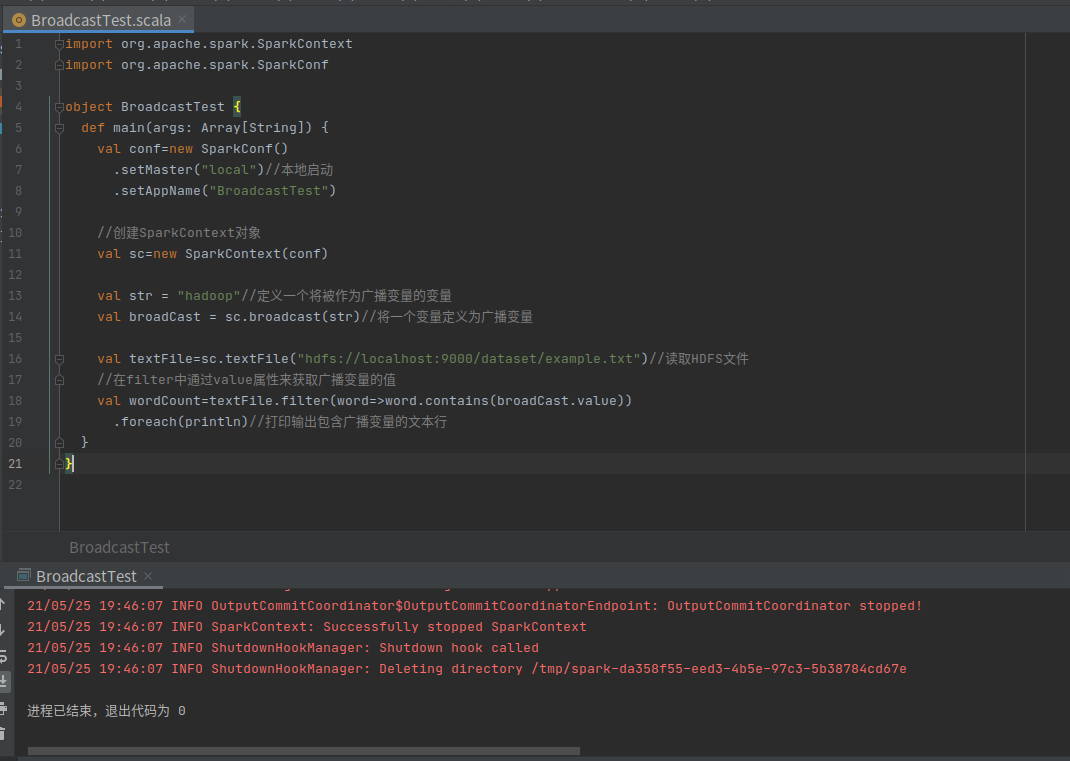


**测试**

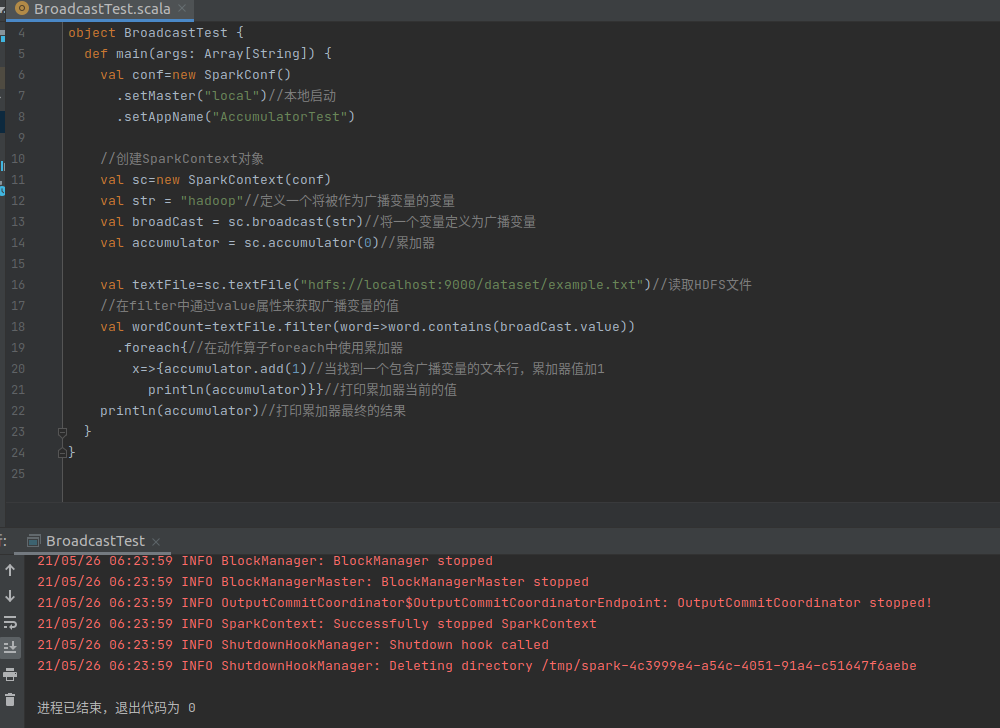




**Spark的共享变量**



**累加器**



**实践基于pySpark的Kmeans实现**

1. import math
2. from pyspark import SparkConf, SparkContext
4. #计算一个点到各个聚类中心的距离，返回距离最近的聚类的序号或者ID
5. def closestCluster(p, centers):
6. closestclusterid = 0#距离最近的聚类的序号
7. closestdist = float("+inf")
8. for i in range(len(centers)):
9. tempdist=math.sqrt(math.pow((p[0]-centers[i][0]),2)+
10. math.pow((p[1]-centers[i][1]),2))
11. if tempdist < closestdist:
12. closestdist = tempdist
13. closestclusterid = i
14. return closestclusterid
16. #主程序
17. if name == "main":
18. conf = SparkConf().setAppName("kmeans")
19. sc = SparkContext(conf=conf)
20. #测试数据，共5个点
21. data=[(1, 2), (3, 4), (2, 6), (7, 2), (4,9)]
22. #利用parallelize函数创建rdd，并进行缓存，因为该RDD在后面的迭代计算中会反复用到
23. datardd = sc.parallelize(data).cache()
25. k=2#设置聚类个数为2
26. convergedist = float(1)#设置停止迭代时，新旧聚类中心间的距离之和的上限
27. #随机选择2个聚类中心，参数False设定为取出样本后不放回
28. centerpointslist = datardd.takeSample(False, k)
29. #每次迭代之和，新旧聚类中心间距离的和，初始设置为大于convergedist
30. tempdist=convergedist+1
32. #开始迭代，当tempdist小于等于convergedist时，停止迭代
33. while tempdist > convergedist:
34. #通过map操作计算每个点距离最近的聚类中心点，返回的是一个rdd，rdd中的元素为
35. #(closestclusterid, (p,1))形式的元组。(p,1)中1的作用是便于后面统计一个聚类中点的个数
36. closestrdd = datardd.map( lambda p:
37. (closestCluster(p, centerpointslist), (p, 1)))
38. #利用reduceByKey操作将每个聚类中的点的x和y分别相加。其中p1c1[1]和p2c2[1]的值#分别是1，将它们相加的作用是统计一个聚类中点或者成员的个数
39. pointstatsrdd = closestrdd.reduceByKey( lambda p1c1, p2c2:
40. ((p1c1[0][0] + p2c2[0][0],p1c1[0][1] + p2c2[0][1]),
41. p1c1[1] + p2c2[1]))
42. #计算各个聚类新的中心，也就是将每个聚类前面相加的x和y的值，除以每个聚类点的个数
43. #其中st[0]是聚类的id，st[1][0][0]是汇总的x，st[1][0][1]是汇总的y，st[1][1]是聚类点的个数
44. newcenterpointslist = pointstatsrdd.map( lambda st:
45. (st[0], (st[1][0][0]/ st[1][1],st[1][0][1]/ st[1][1]))).collect()
47. #计算新的各个聚类中心与老的聚类中心间欧式距离的和
48. tempdist=0
49. for (clusterkey, newcenterpoint) in newcenterpointslist:
50. x2=math.pow((newcenterpoint[0]-centerpointslist[clusterkey][0]),2)
51. y2=math.pow((newcenterpoint[1]-centerpointslist[clusterkey][1]),2)
52. dist=math.sqrt(x2+y2)
53. tempdist+=dist
54. #利用新得到的聚类中心的坐标更新各个聚类的中心
55. for (clusterkey, newcenterpoint) in newcenterpointslist:
56. centerpointslist[clusterkey] = newcenterpoint
58. #打印输出得到的聚类的中心点坐标
59. print("Final centers: " + str(centerpointslist))
60. clusterresultlist=closestrdd.collect()
61. for item in clusterresultlist:
62. print item[1][0], item[0]#输出各个点以及他们所属的聚类
64. sc.stop()

