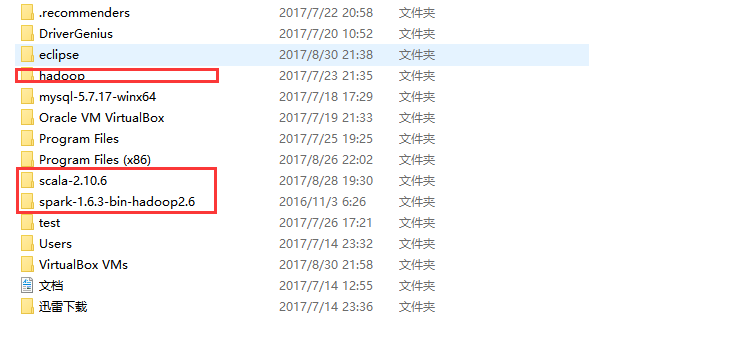
# 大数据平台部分实践作业9#

# 环境配置

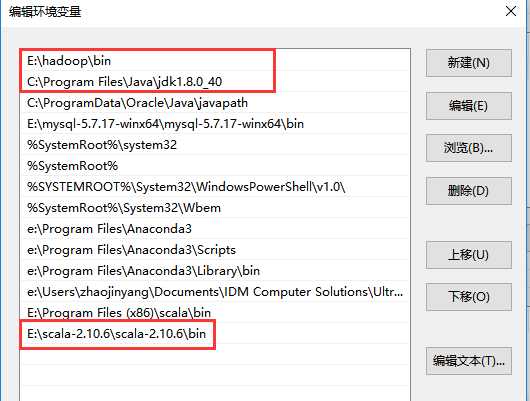
在IntelliJ下运行scala程序需要配置hadoop、spark、scala、JDK等环境变量。

1. 解压hadoop、spark、scala

将hadoop、spark、scala解压，并将hadoop、spark、scala中的bin目录分别配置在环境变量中。



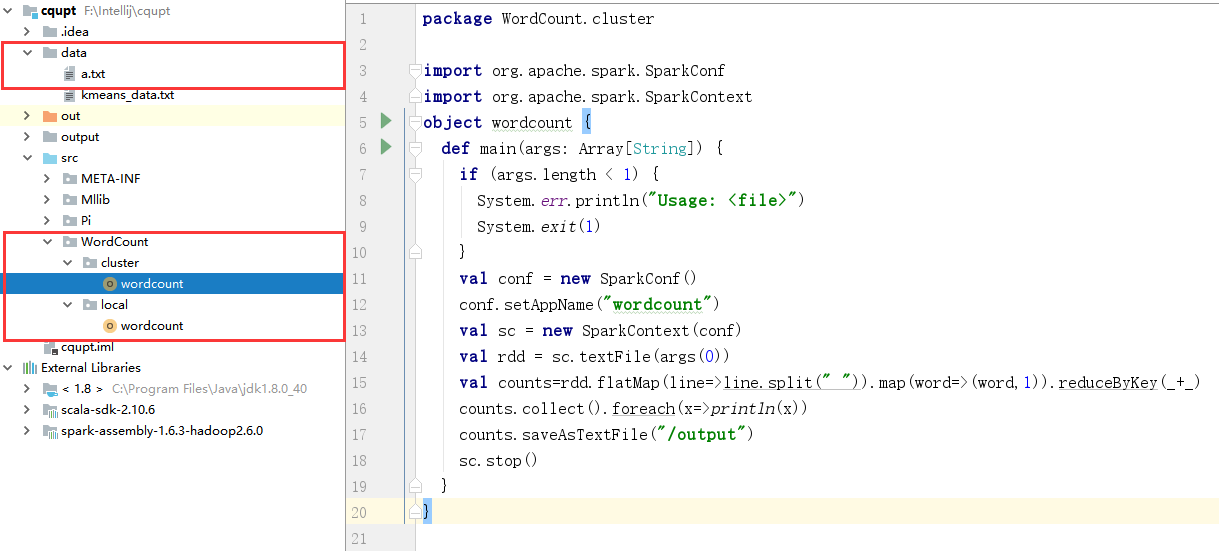
（2）、配置hadoop、spark、scala环境变量



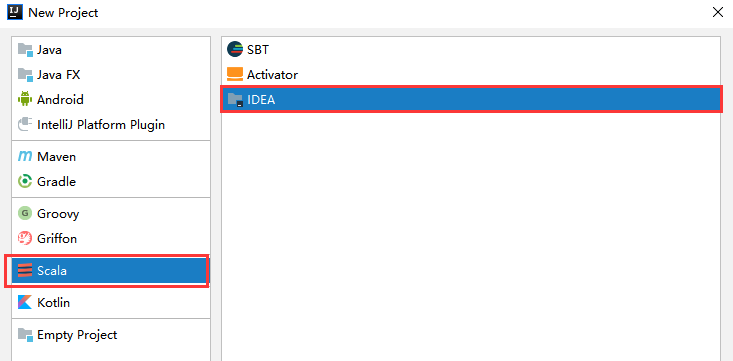
# Spark应用编程---wordcount

## 2.1、wordcound工程

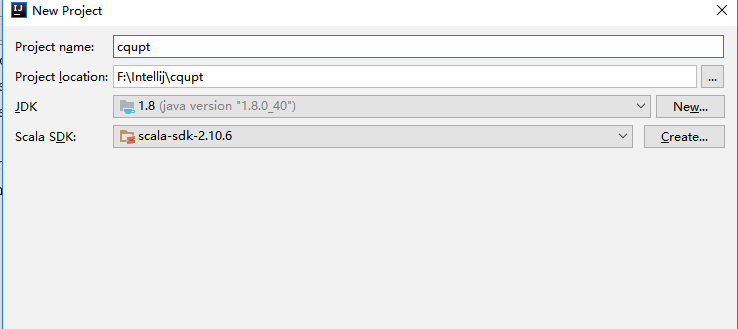
（1）、工程结构图如下所示：



（2）、新建工程/file/new/project

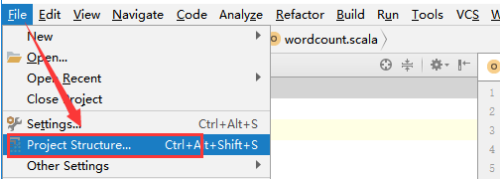


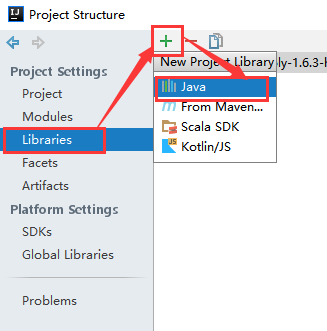
（3）、填写工程相应的名字、位置、导入相应JDK和Scala环境变量位置

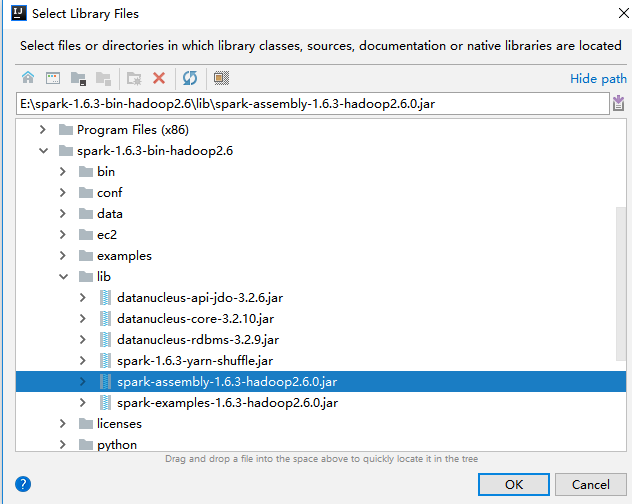


（4）、导入spark的jar包

将spark解压包下lib/spark-assembly-1.6.3-hadoop2.6.0.jar导入中

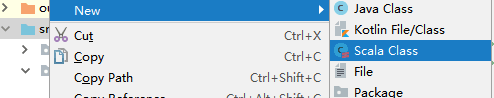




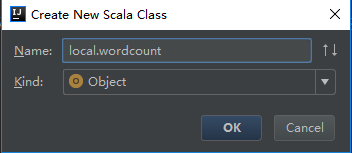


## 2.2、local模式

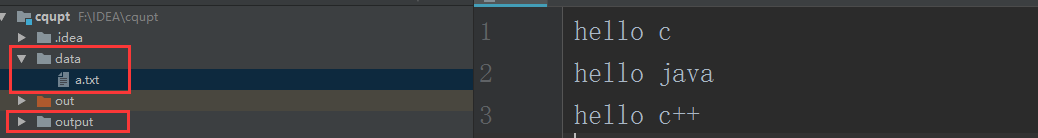
（1）、创建scala程序cqupt/src/new/Scala Class



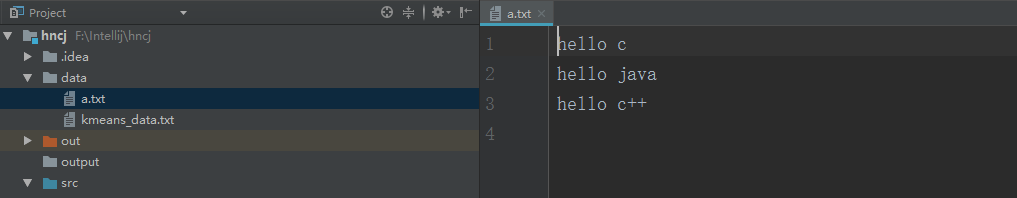
！！！注意命名格式



1. 、创建输入/输出文件

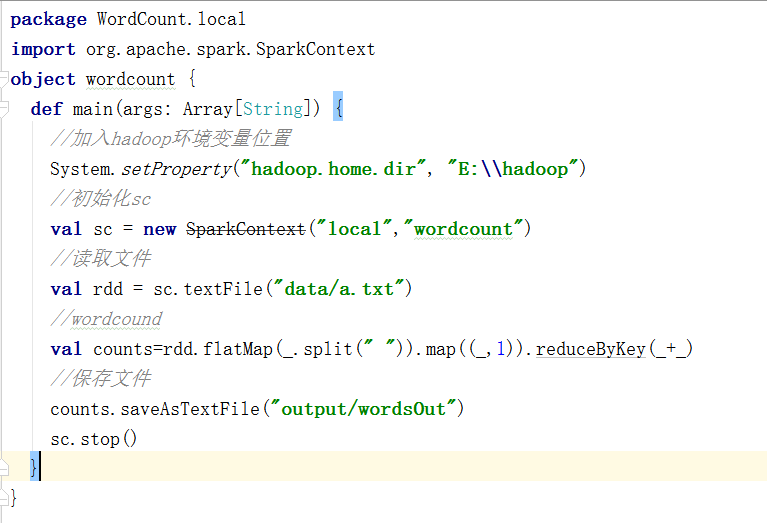


（3）、在cqupt下建立data文件夹放入数据



（4）、在本地编写程序

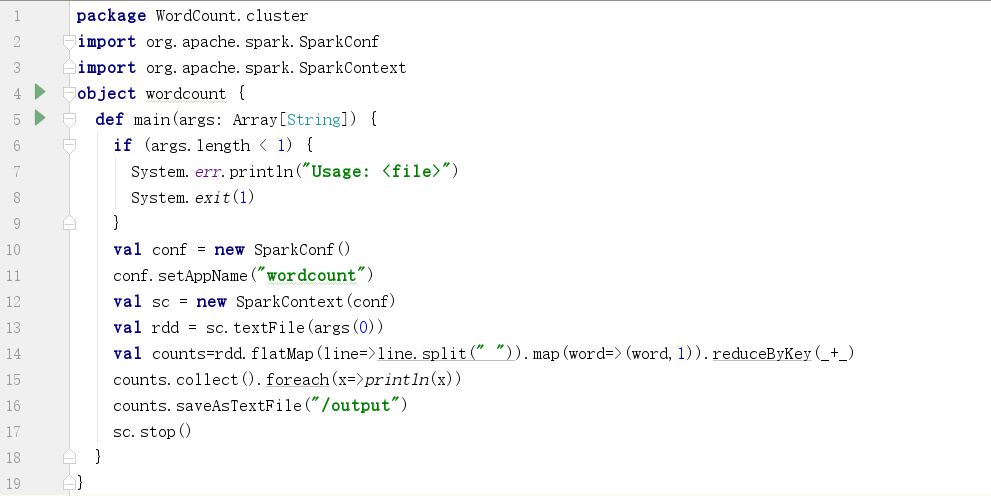
**package** WordCount.local  
**import** org.apache.spark.SparkContext  
**object** wordcount {  
 **def** main(args: Array[String]) {  
 *//加入hadoop环境变量位置* System.*setProperty*(**"hadoop.home.dir"**, **"E:\\hadoop"**)  
 *//初始化sc* **val** sc = **new** SparkContext(**"local"**,**"wordcount"**)  
 *//读取文件* **val** rdd = sc.textFile(**"data/a.txt"**)  
 *//wordcound* **val** counts=rdd.flatMap(\_.split(**" "**)).map((\_,1)).reduceByKey(\_+\_)  
 *//保存文件* counts.saveAsTextFile(**"output/wordsOut"**)  
 sc.stop()  
 }  
}



## 2.3、clustet/client模式

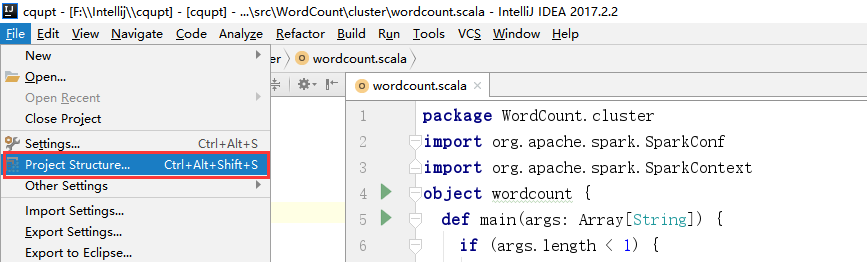
（1）、编写Scala程序

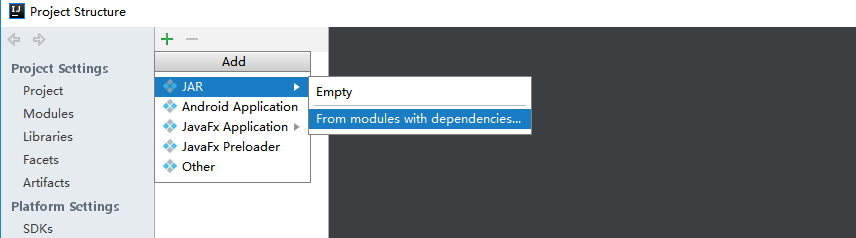
**package** WordCount.cluster  
**import** org.apache.spark.SparkConf  
**import** org.apache.spark.SparkContext  
**object** wordcount {  
 **def** main(args: Array[String]) {  
 *//文件不存在* **if** (args.length < 1) {  
 System.*err*.println(**"Usage: <file>"**)  
 System.*exit*(1)  
 }  
 **val** conf = **new** SparkConf()  
 conf.setAppName(**"wordcount"**)  
 *//初始化sc* **val** sc = **new** SparkContext(conf)  
 *//读取文件* **val** rdd = sc.textFile(args(0))  
 *//wordcound* **val** counts=rdd.flatMap(line=>line.split(**" "**)).map(word=>(word,1)).reduceByKey(\_+\_)  
 counts.collect().foreach(x=>*println*(x))  
 *//保存文件* counts.saveAsTextFile(**"/output"**)  
 sc.stop()  
 }  
}

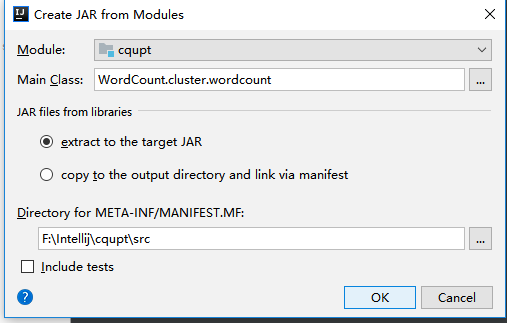


## 2.4、导出wordcound包

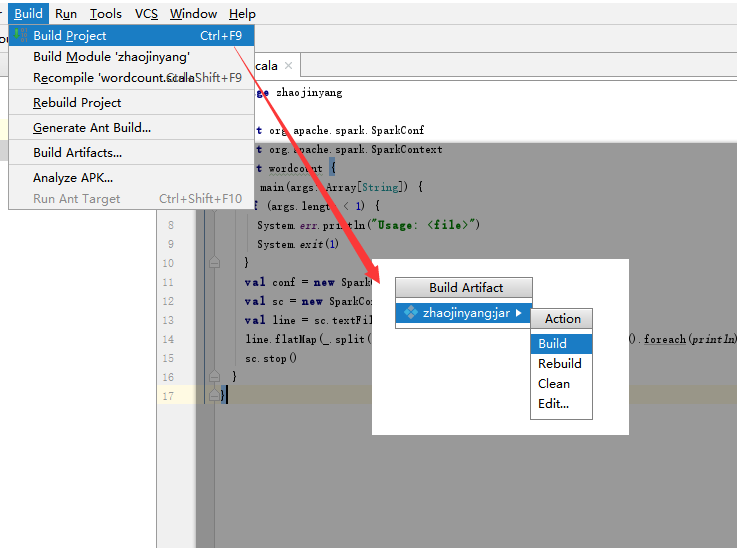
（1）、选择File -> Project Structure -> Artifact, 选择‘+’----->Jar---->From Modeles with dependencies ,选择main函数，之后要指定下输出的位置。







（2）选择Build -> Build Artifacts -> jar包名 -> Build，直到编译器左下角出现completed successfully。

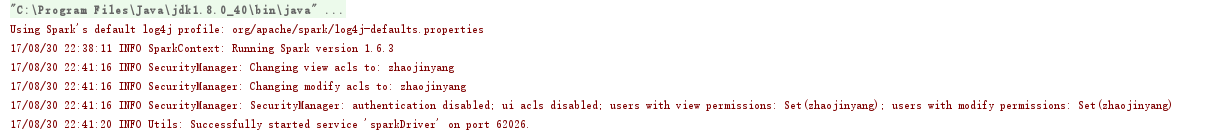


（3）将导入的cqupt.jar上传到hadoop的/usr/local目录下。

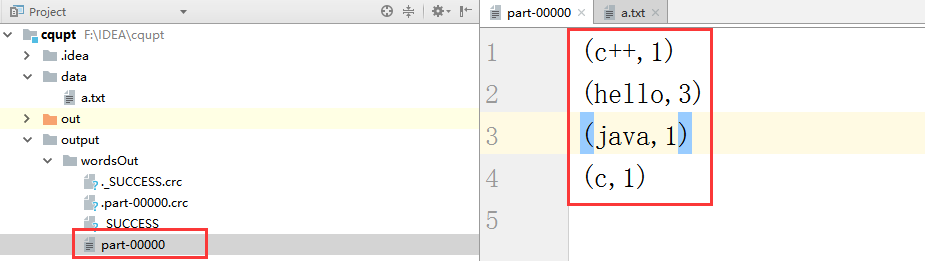
## 1.5、运行测试

### **1.5.1、本地测试**

1. 、在intelliJ中运行local.wordcound，我们再控制台观察结果

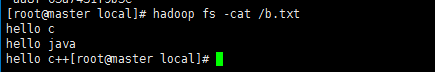


（2）、我们在intelliJ左侧看到生成的output文件中wordcound，打开我们发现结果如下所示



### **1.5.1、集群测试**

1. 、将b.txt文件上传到hdfs中根目录下。



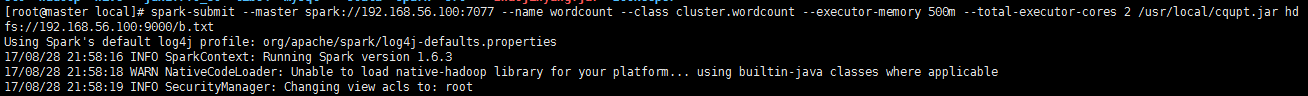
（2）、从master节点进入到/usr/local/spark/sbin启动Spark

cd /usr/local/spark/sbin

开启：

start-all.sh

（3）、执行程序

spark-submit --master spark://192.168.56.100:7077 --name wordcount --class WordCount.cluster.wordcount --executor-memory 500m --total-executor-cores 2 /usr/local/cqupt.jar hdfs://192.168.56.100:9000/b.txt

参数解释：

--name wordcount 表示提交任务名。

--class WordCount.cluster.wordcount表示执行的方法，带包名。

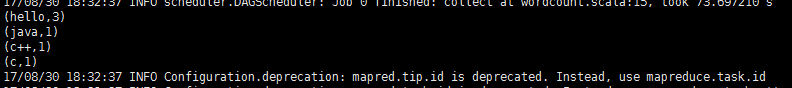
--executor-memory 500m表示给每个executor指定使用内存。

--total-executor-cores 2 表示所有的executor使用的总CPU核数。

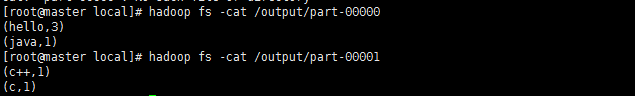
/cqupt.jar 表示jar文件所在的路径。

hdfs://192.168.56.100:9000/b.txt 表示要进行操作的txt文件在HDFS上的路径。

1. 、运行结果



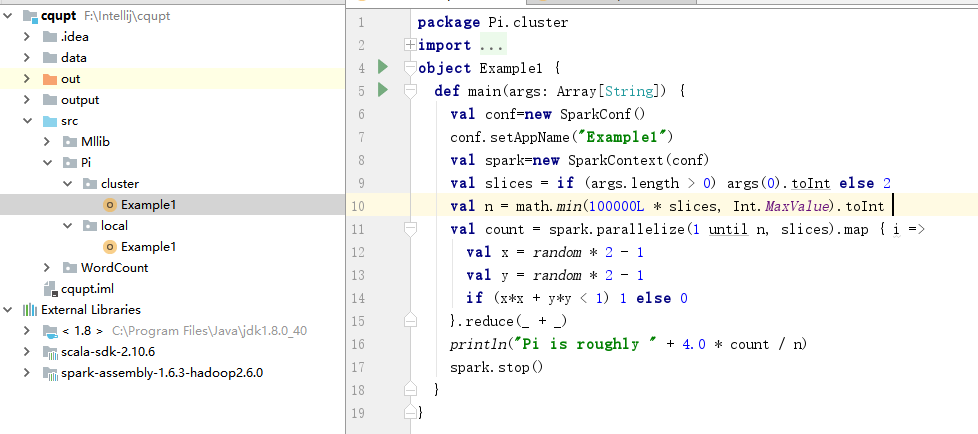
1. 、在hdfs上保存的结果



# Spark应用编程---Pi

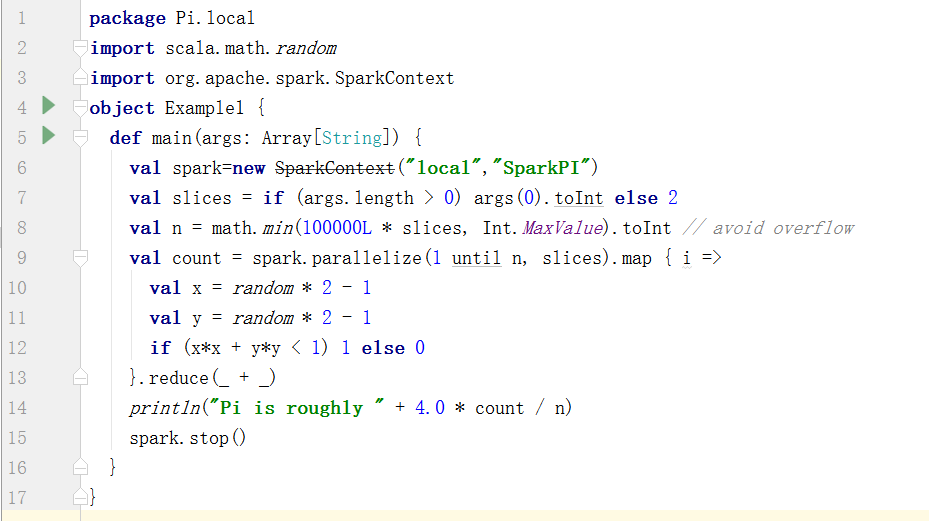
## 2.1、Pi工程

1. 、工程结构图如下所示：



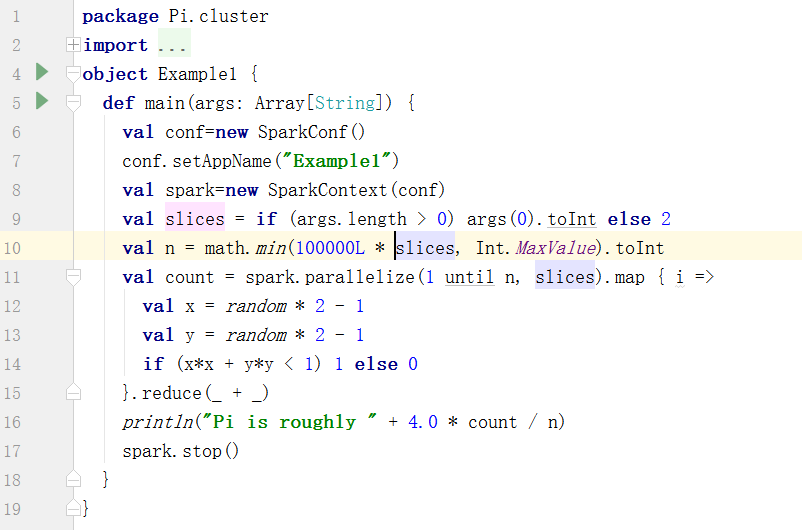
## 2.2、local模式

**package** Pi.local  
**import** scala.math.*random***import** org.apache.spark.SparkContext  
**object** Example1 {  
 **def** main(args: Array[String]) {  
 **val** spark=**new** SparkContext(**"local"**,**"SparkPI"**)  
 **val** slices = **if** (args.length > 0) args(0).toInt **else** 2  
 **val** n = math.*min*(100000L \* slices, Int.*MaxValue*).toInt *// avoid overflow* **val** count = spark.parallelize(1 until n, slices).map { i =>  
 **val** x = *random* \* 2 - 1  
 **val** y = *random* \* 2 - 1  
 **if** (x\*x + y\*y < 1) 1 **else** 0  
 }.reduce(\_ + \_)  
 *println*(**"Pi is roughly "** + 4.0 \* count / n)  
 spark.stop()  
 }  
}



## 2.3、clustet/client模式

**package** Pi.cluster  
**import** scala.math.*random***import** org.apache.spark.{SparkConf, SparkContext}  
**object** Example1 {  
 **def** main(args: Array[String]) {  
 **val** conf=**new** SparkConf()  
 conf.setAppName(**"Example1"**)  
 **val** spark=**new** SparkContext(conf)  
 **val** slices = **if** (args.length > 0) args(0).toInt **else** 2  
 **val** n = math.*min*(100000L \* slices, Int.*MaxValue*).toInt  
 **val** count = spark.parallelize(1 until n, slices).map { i =>  
 **val** x = *random* \* 2 - 1  
 **val** y = *random* \* 2 - 1  
 **if** (x\*x + y\*y < 1) 1 **else** 0  
 }.reduce(\_ + \_)  
 *println*(**"Pi is roughly "** + 4.0 \* count / n)  
 spark.stop()  
 }  
}

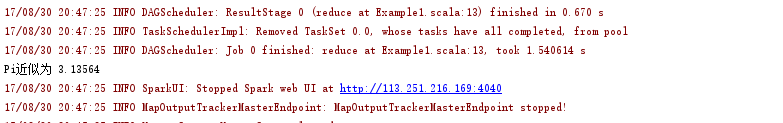


将导出的cqupt.jar上传到hadoop的/usr/local目录下。

## 2.4、运行测试

### **2.4.1、本地测试**

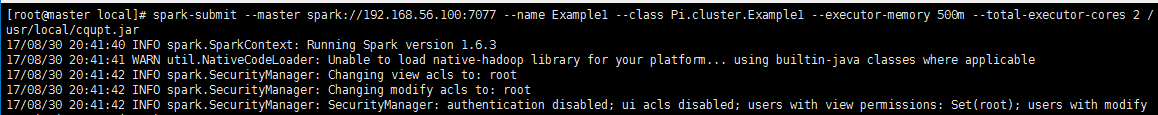
在intelliJ上运行结果为：



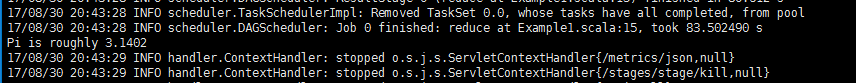
### **2.4.1、集群测试**

（1）、执行程序

spark-submit --master spark://192.168.56.100:7077 --name Example1 --class Pi.cluster.Example1 --executor-memory 500m --total-executor-cores 2 /usr/local/cqupt.jar



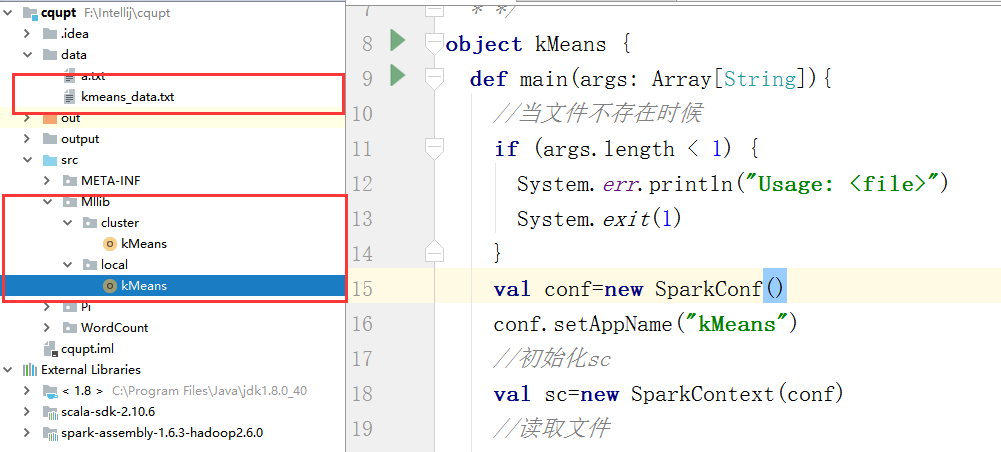
1. 、运行结果



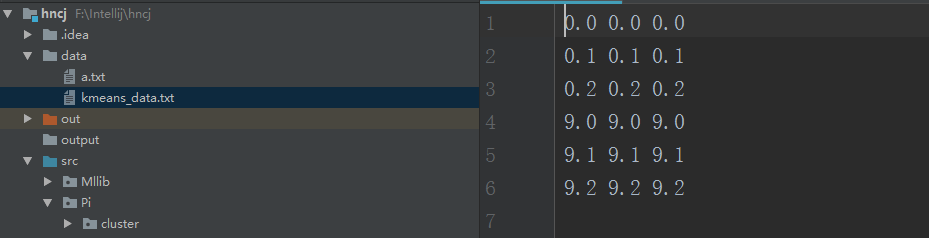
# Spark应用编程---kMeans

## 3.1、kMeans工程

(1)、工程结构图如下所示：



1. 、在cqupt下建立data文件夹放入数据



## 3.2、local模式

**package** Mllib.local  
**import** org.apache.spark.SparkContext  
**import** org.apache.spark.mllib.clustering.KMeans  
**import** org.apache.spark.mllib.linalg.Vectors  
*/\*\*  
 \* 掌握调用Mllib库函数编写机器学习算法应用程序kMeans  
 \* \*/***object** kMeans {  
 **def** main(args: Array[String]){  
 *//加入hadoop环境变量位置* System.*setProperty*(**"hadoop.home.dir"**,**"e:\\hadoop"**)  
 *//初始化sc* **val** sc=**new** SparkContext(**"local"**,**"kMeans"**)  
 *//读取文件* **var** data=sc.textFile(**"data/kmeans\_data.txt"**)  
 *//格式化数据集：按照空格切分数据并转为double类型* **var** dataset=data.map(x=>Vectors.*dense*(x.split(**" "**).map(\_.toDouble)))  
 *//最大迭代次数* **var** maxIteration=20  
 *//划分最大簇数* **var** maxClass=5  
 *//构建训练模型* **val** model=KMeans.*train*(dataset,maxClass,maxIteration)  
 *//输出簇中心  
 println*(**"Output cluster center："**)  
 **for**(c<-model.clusterCenters){  
 *println*(**" "**+c.toString)  
 }  
 *//使用误差平方之和估计数据模型* **val** cost=model.computeCost(dataset)  
 *println*(**"Sum of squares of errors="**+cost)  
 *//使用模型测试单点数据  
 println*(**"data 0.2 0.2 0.2 is belongs to cluster:"**+model.predict(Vectors.*dense*(**"0.2 0.2 0.2"**.split(**" "**).map(\_.toDouble))))  
 *println*(**"data 9 9 9 is belongs to cluster:"**+model.predict(Vectors.*dense*(**"9 9 9"**.split(**" "**).map(\_.toDouble))))  
 *//将结果打印output/kMeans1* **val** result1=model.predict(dataset)  
 result1.saveAsTextFile(**"output/kMeans1"**)  
 *//将数据集和结果 打印output/kMeans2* **val** result2=data.map{  
 line=>  
 **val** dataset1=Vectors.*dense*(line.split(**" "**).map(\_.toDouble))  
 **val** predictresult=model.predict(dataset1)  
 line+**" "**+predictresult  
 }.saveAsTextFile(**"output/kMeans2"**)  
 *//关闭sc* sc.stop()  
  
  
 }  
}

## 3.3、clustet/client模式

**package** Mllib.cluster  
**import** org.apache.spark.{SparkConf, SparkContext}  
**import** org.apache.spark.mllib.clustering.KMeans  
**import** org.apache.spark.mllib.linalg.Vectors  
*/\*\*  
 \* 掌握调用Mllib库函数编写机器学习算法应用程序kMeans  
 \* \*/***object** kMeans {  
 **def** main(args: Array[String]){  
 *//当文件不存在时候* **if** (args.length < 1) {  
 System.*err*.println(**"Usage: <file>"**)  
 System.*exit*(1)  
 }  
 **val** conf=**new** SparkConf()  
 conf.setAppName(**"kMeans"**)  
 *//初始化sc* **val** sc=**new** SparkContext(conf)  
 *//读取文件* **var** data=sc.textFile(args(0))  
 *//格式化数据集：按照空格切分数据并转为double类型* **var** dataset=data.map(x=>Vectors.*dense*(x.split(**" "**).map(\_.toDouble)))  
 *//最大迭代次数* **var** maxIteration=20  
 *//划分最大簇数* **var** maxClass=5  
 *//构建训练模型* **val** model=KMeans.*train*(dataset,maxClass,maxIteration)  
 *//输出簇中心  
 println*(**"Output cluster center："**)  
 **for**(c<-model.clusterCenters){  
 *println*(**" "**+c.toString)  
 }  
 *//使用误差平方之和估计数据模型* **val** cost=model.computeCost(dataset)  
 *println*(**"Sum of squares of errors="**+cost)  
 *//使用模型测试单点数据  
 println*(**"data 0.2 0.2 0.2 is belongs to cluster:"**+model.predict(Vectors.*dense*(**"0.2 0.2 0.2"**.split(**" "**).map(\_.toDouble))))  
 *println*(**"data 9 9 9 is belongs to cluster:"**+model.predict(Vectors.*dense*(**"9 9 9"**.split(**" "**).map(\_.toDouble))))  
 *//将结果打印output/kMeans1* **val** result1=model.predict(dataset)  
 result1.saveAsTextFile(**"/output/kMeans1"**)  
 *//将数据集和结果 打印output/kMeans2* **val** result2=data.map{  
 line=>  
 **val** dataset1=Vectors.*dense*(line.split(**" "**).map(\_.toDouble))  
 **val** predictresult=model.predict(dataset1)  
 line+**" "**+predictresult  
 }.saveAsTextFile(**"/output/kMeans2"**)  
 *//关闭sc* sc.stop()  
 }  
}

将导出的cqupt.jar上传到hadoop的/usr/local目录下。

## 3.4、运行测试

### **3.4.1、本地测试**

1. 、在intelliJ中运行local.wordcound，我们再控制台观察结果：

Output cluster center：

[9.0,9.0,9.0]

[0.05,0.05,0.05]

[0.2,0.2,0.2]

[9.2,9.2,9.2]

[9.1,9.1,9.1]

Sum of squares of errors=0.01500000000000001

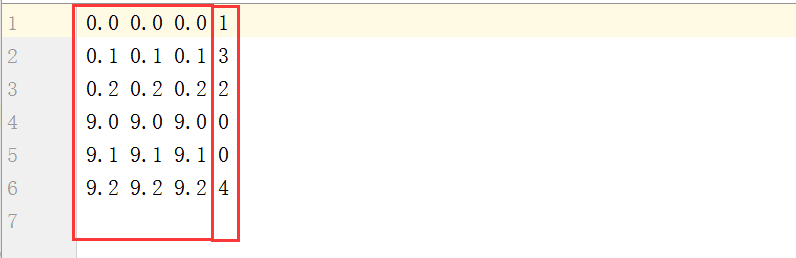
data 0.2 0.2 0.2 is belongs to cluster:2

data 9 9 9 is belongs to cluster:0

（2）、我们在intelliJ左侧看到生成的kMeans文件中，打开我们发现结果如下所示



前三列为原始数据，后一列为分的簇



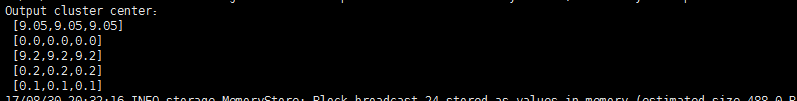
### **3.4.1、集群测试**

（1）、执行程序

spark-submit --master spark://192.168.56.100:7077 --name kMeans --class Mllib.cluster.kMeans --executor-memory 500m --total-executor-cores 2 /usr/local/cqupt.jar hdfs://192.168.56.100:9000/kmeans\_data.txt

1. 、运行结果

输入的簇中心为：



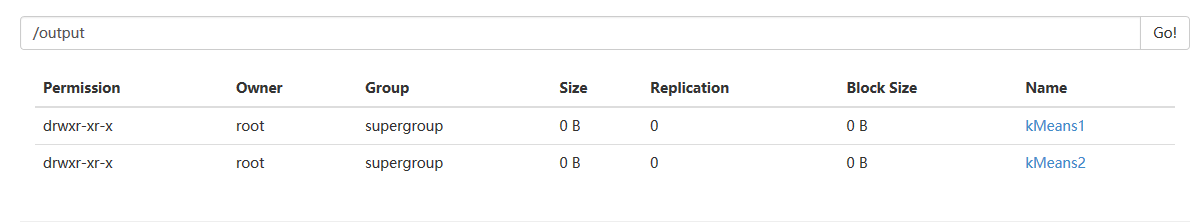
使用误差平方之和估计数据模型：

IMG_256

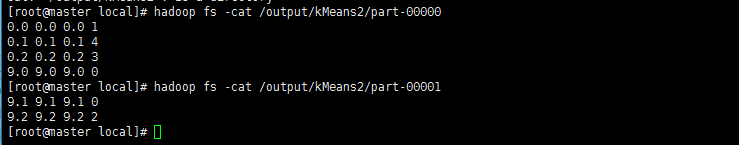
使用模型测试单点数据：



将结果保存在hdfs上中：



在kMeans2中观察原数据集通过kMeans聚类后的结果为：



# 错误总结

（1）、抛出异常：

17/08/28 12:14:18 ERROR Shell: Failed to locate the winutils binary in the hadoop binary path  
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

问题分析：

可能没有配置hadoop环境变量，使得运行程序时找不到winutils.exe，解决方法配置hadoop环境变量引入winutils.exe

解决办法一：

System.setProperty("hadoop.home.dir", "E:\\hadoop");

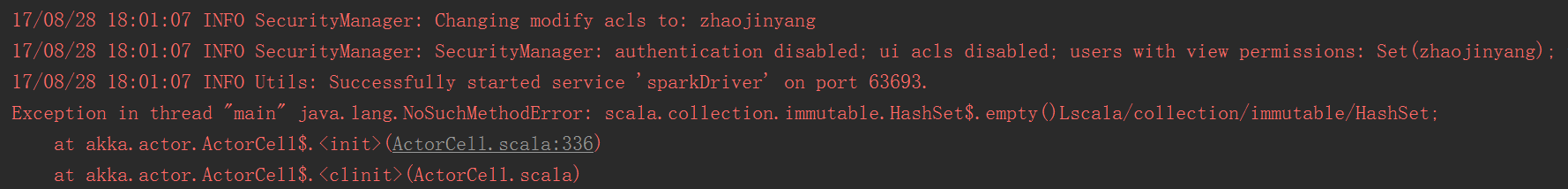
解决方法二：

下载winutils.exe，下载地址是

https://github.com/srccodes/hadoop-common-2.2.0-bin，下载后是文件名是hadoop-common-2.2.0-bin-master.zip,随便解压到一个目录，然后在系统变量path里增加$HADOOP\_HOME\bin 即可。再次运行程序，正常执行。

（2）、抛出异常

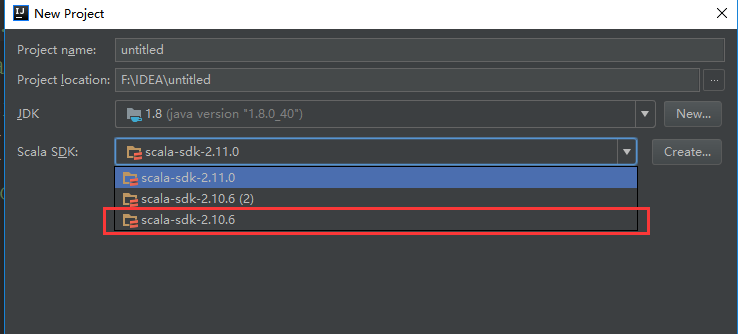
Exception in thread "main" java.lang.NoSuchMethodError: scala.collection.immutable.HashSet$.empty()Lscala/collection/immutable/HashSet;



原因分析：

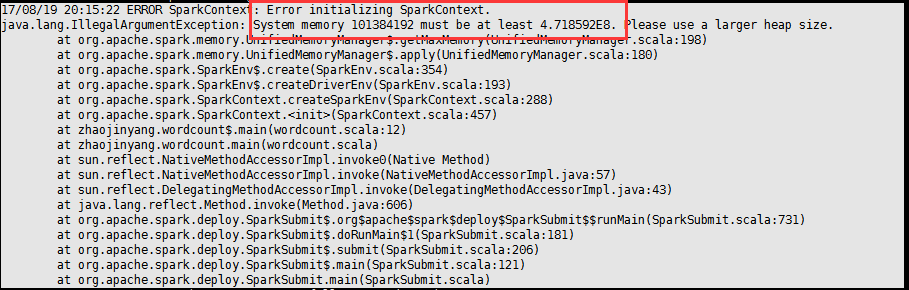
scala-sdk和spark版本不比配问题

解决办法：scala-sdk和spark版本不比配问题，在网上下载新的scala，更改scala版本，可以解决问题。



（3）、抛出异常：

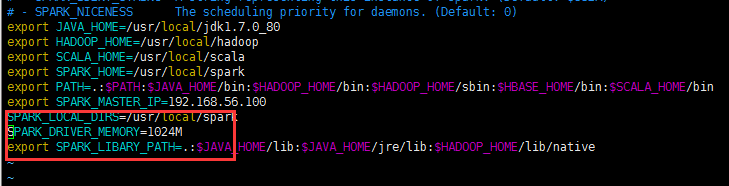
如果出现内存不足的问题，可能使在spark-env.sh配置文档问题。



解决办法：

将SPARK\_DRIVER\_MEMORY修改为1024M

SPARK\_DRIVER\_MEMORY=1024M



（4）、关于IntelliJ运行速度问题

原来未优化IntelliJ之前，运行一个程序，到达下图位置会停止1-2min,运行速度偏慢。

IMG_256

优化IntelliJ方法：

如图用记事本打开

idea.exe.vmoptions

idea64.exe.vmoptions



如下图修改

idea.exe.vmoptions

idea64.exe.vmoptions

