第4章Spark编程进阶

(源自:https://biglab.site)

(版本:Ver1.8-20230922)

学习目标

- 学习配置Spark开发环境
- 掌握如何新建工程和运行工程
- 掌握运行 Spark程序的方法
- 了解数据分区与持久化的方法

任务背景

通过数据竞赛网站，学生可以浏览和选择合适的竞赛。数据竞赛网站的出现也意味着有相应的用户访问记录；网站运营商通过对用户访问记录的分析，可以对当前网站或竞赛项目进行合理、有效的改良，吸引更多的人参与到竞赛项目中；

某竞赛网站经过几年的运营，保存了众多用户对该网站的访问日志数据。为了依据用户的历史浏览记录研究用户的兴趣爱好，改善用户体验，需要对网站用户的访问日志数据进行分析。现有一份该竞赛网站2020年5月至2021年2月的用户访问日志数据文件raceData.csv；

网站的访问次数分布情况对网站运营商来说是非常重要的指标之一，因此需要针对网站的用户访问日志数据，统计竞赛网站每月的访问量。考虑到网站的用户访问日志数据的数据规模非常庞大，使用一般的数据分析工具处理这些数据的效率很低，因此将使用Spark框架对网站的用户访问日志数据进行统计分析，并且为了模拟真实的生产环境，将使用IntelliJ IDEA工具进行Spark编程。首先介绍IntelliJ IDEA工具及其Scala插件的安装过程，并介绍在IntelliJ IDEA中配置Spark运行环境的过程。再详细介绍如何在开发环境中和集群环境中提交并运行Spark程序，结合竞赛网站用户访问日志数据实例，在IntelliJ IDEA中进行Spark编程、实现网站的访问量统计

学习重点

理论学习

- （1）搭建IDEA开发环境。
- （2）配置Spark运行环境。
- （3）运行Spark程序的方式。
- （4）持久化（缓存）。
- （5）数据分区。

实验学习

- （1）搭建开发环境。
- （2）自定义分区。
- （3）计算价格波动幅度和使用移动平均预测股票涨跌任务。
- （4）竞赛网站访问日志分析。

准备测试环境

链接：https://pan.baidu.com/s/1FF7HHQi9y2kVBXom_Omd_g
提取码：8ft4

软件列表：

软件	版本	安装包称
Spark	3.2.1	spark-3.2.1-bin-hadoop2.7.tgz
JDK(WIN)	1.8	jdk-8u333-windows-x64.exe
IDEA	2018.3.6	ideaIC-2018.3.6.exe 或 https://www.jetbrains.com
Scala插件	2018	scala-intellij-bin-2018.3.6.zip
Scala	2.12.15	scala-2.12.15.tgz
Hadoop	3.1.4	hadoop-3.1.4.tar.gz
IntelliJ IDEA	2018.3.6	ideaIC-2018.3.6.exe
Git	2.39.2	Git-2.39.2-64-bit.exe
TortoiseGit	2.14	TortoiseGit-2.14.0.0-64bit.msi
spark_data	3.2.1	spark_data.tar.gz
winutils	3.1.4	winutils.rar

下载说明：

1. 将列表中的软件和文件下载到D:/spark目录中；
2. 将D:/spark下的spark_data.tar.gz，使用winrar解压到当前路径下
3. 将D:/spark下的spark-3.2.1-bin-hadoop2.7.tgz，使用winrar解压到当前路径下
4. 将D:/spark下的hadoop-3.1.4.tar.gz，使用winrar解压到当前路径下
5. 将D:/spark下的winutils.rar, 使用winrar解压到当前路径下,**然后将winutils\bin目录下的所有文件，包括winutils.exe等复制到hadoop-3.1.4\bin目录下**
6. 运行并安装：D:/spark下jdk-8u333-windows-x64.exe
7. 运行并安装：D:/spark下Git-2.39.2-64-bit.exe
8. 运行并安装：D:/spark下TortoiseGit-2.14.0.0-64bit.msi
9. 运行并安装：D:/spark下ideaIC-2018.3.6.exe 或从 https://www.jetbrains.com下载更新的idea安装

准备测试源码

准备git客户端安装

1. 安装Git-2.39.2-64-bit.exe
2. 安装TortoiseGit-2.14.0.0-64bit.msi

运行git clone下载源码

```
cd existing_repo
git clone https://jihulab.com/biglab-share/spark.git
```

准备测试数据

```
将网盘上下载的spark_data.tar.gz,解压到当前中径，比如：D:\spark\spark_data
```

任务4.1搭建Spark开发环境

任务描述

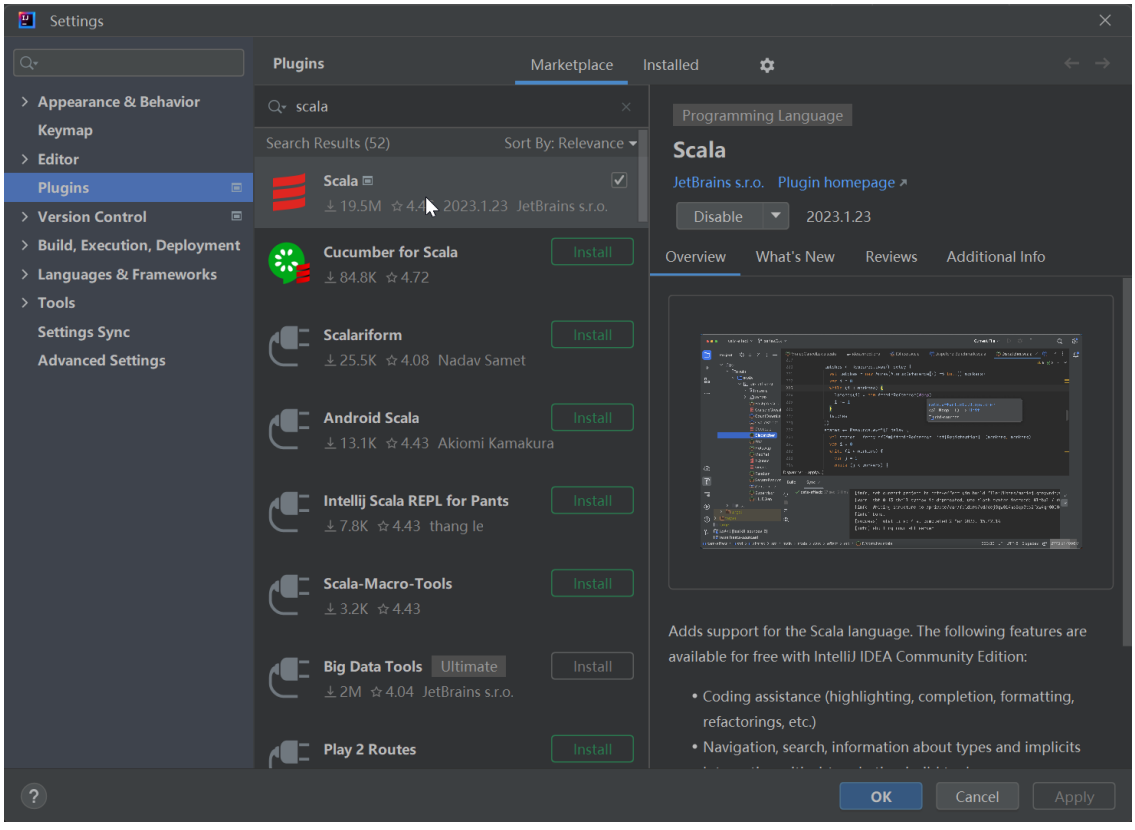
使用Spark框架进行数据分析需要有合适的编程环境。第3章介绍了Spark编程的基础操作，操作的过程是在spark-shell的交互式环境中进行的，这样的交互式环境会对每个指令做出反馈，适合初学者学习或调试代码时使用。

而在真实的生产环境中，完成一个任务通常需要很多行代码，并且需要多个类协作才能实现，因此需要更加适合的开发环境，如IntelliJ IDEA。

本节的任务如下。下载并安装IntelliJ IDEA。在IntelliJ IDEA中安装Scala插件，添加Spark开发依赖包，配置Spark运行环境，实现Spark工程的创建。Spark程序的编写与运行

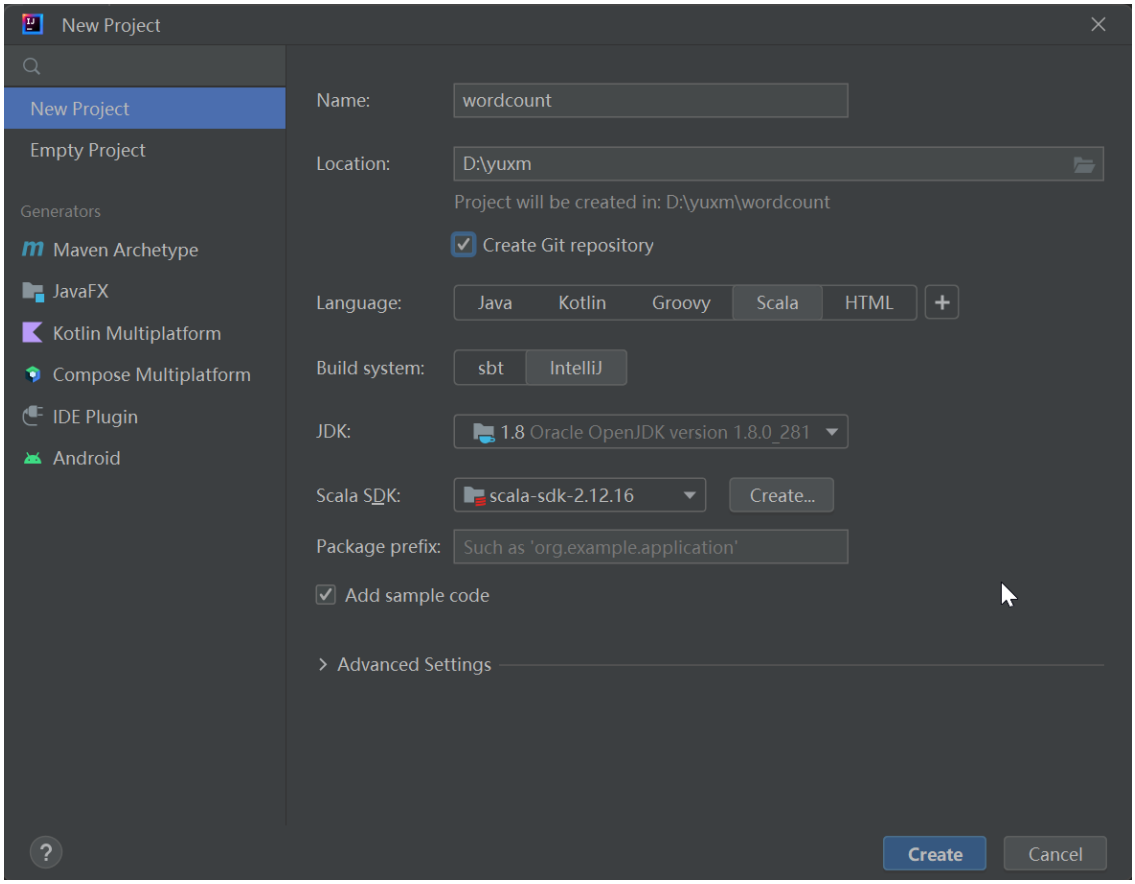
Scala插件安装与使用

- 进入IDEA后，选择菜单上File->Settings->Plugins,搜Scala,->安装



- 创建scala项目

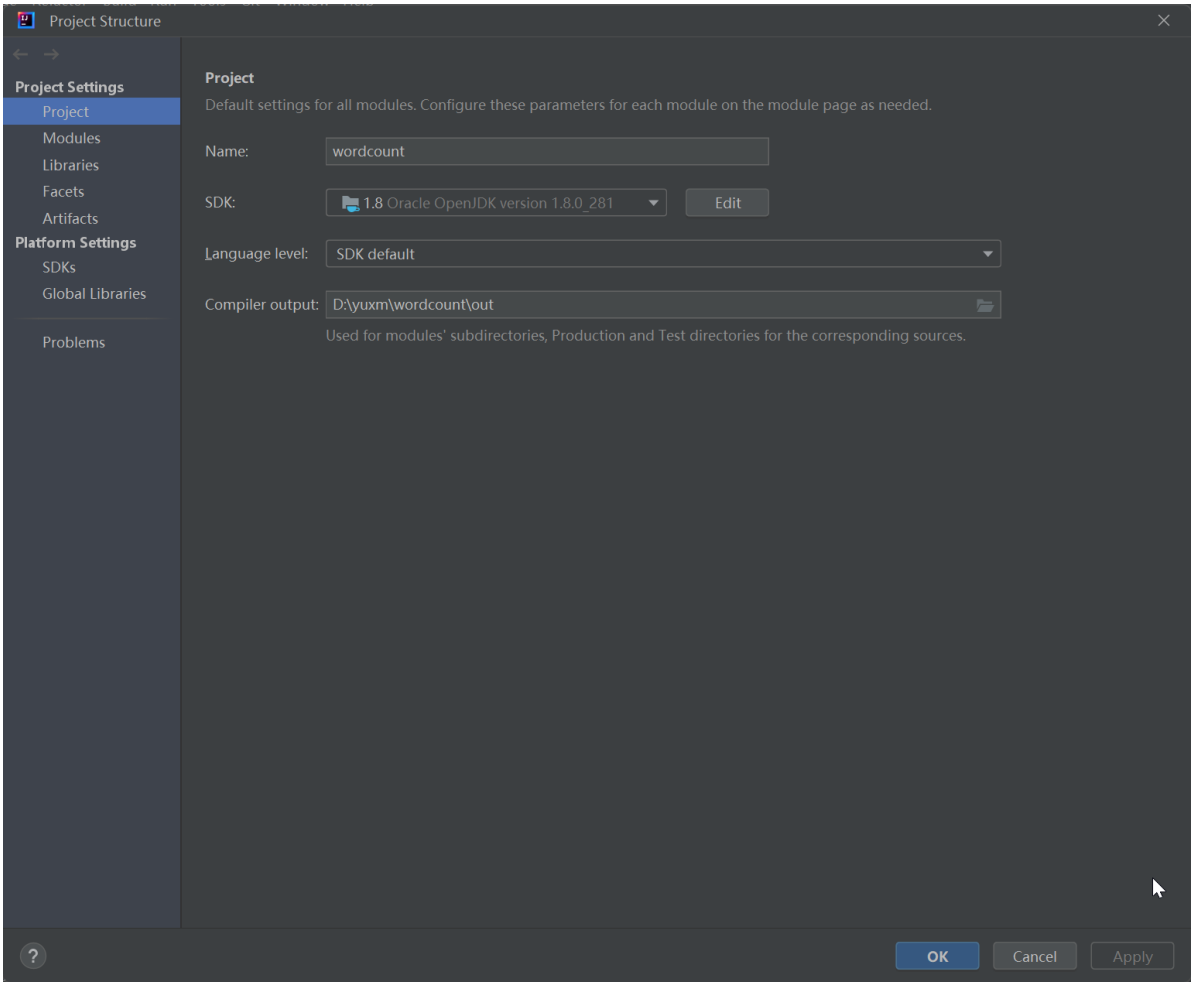
  New Project->
  1，项目名为:wordcount
  2，存放目录建议为d:\myname\wordcount
  3，语言选择：scala
  4，build system: Intellij
  5，Jdk:1.8
  6，scala SDK: scala-sdk-2.12.16(若没有，点击create进入下载)



配置Spark运行环境

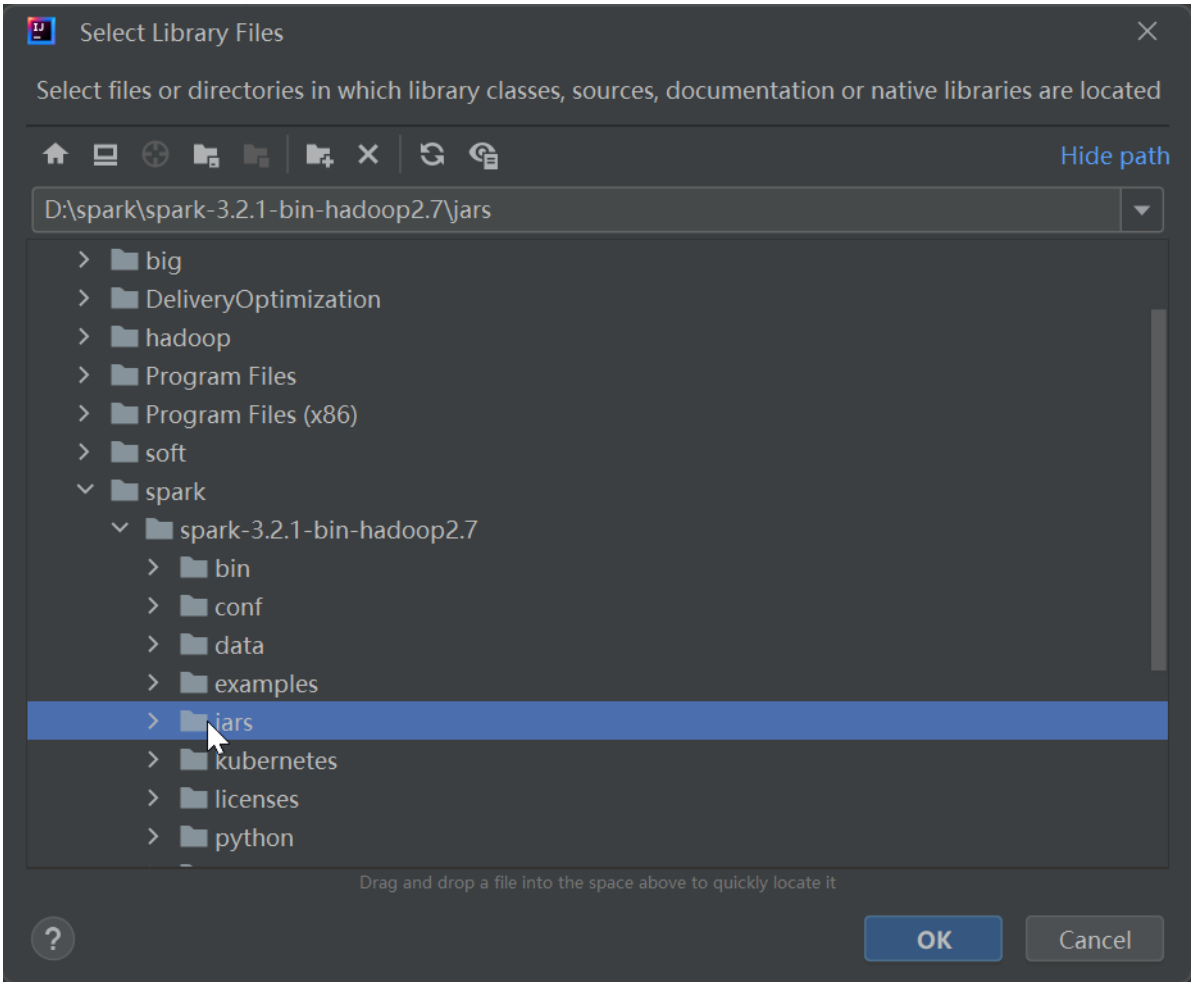
配置Spark开发依赖包

点击菜单栏中的“File”->“Project Structure”，打开所示的界面：

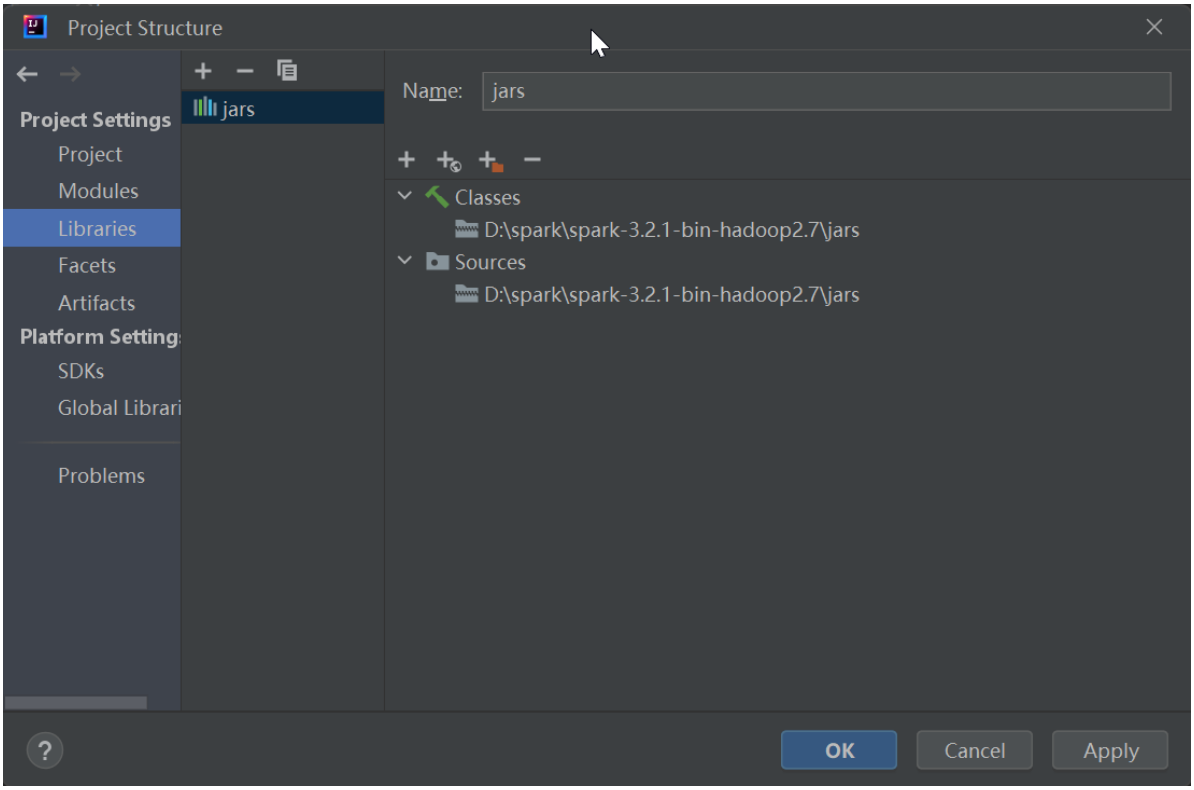


将百度网盘下载下来的：spark-3.2.1-bin-hadoop2.7.tgz 解压到如：D:\spark\spark-3.2.1-bin-hadoop2.7 目录下。

选择“Libraries”选项，单击“+”按钮，选择“Java”选项在弹出的界面中找到Spark安装目录下的jars文件夹（如：D:\spark\spark-3.2.1-bin-hadoop2.7\jars），将整个文件夹导入，如图所示点击“OK”：



OK确认后,如图：



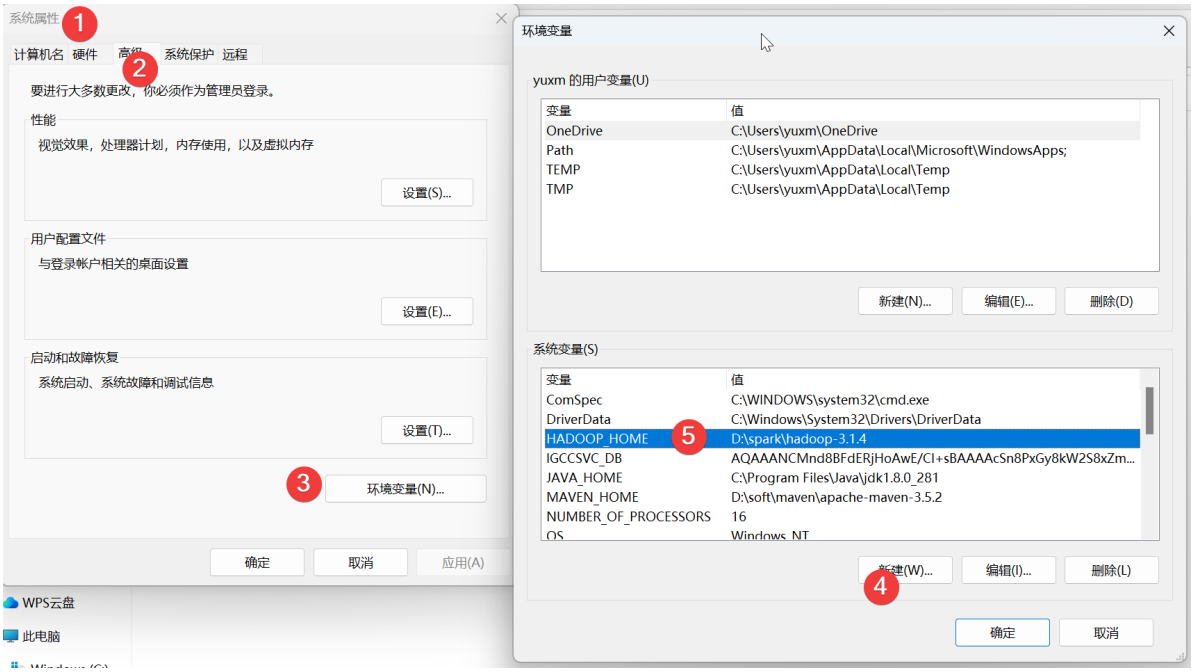
配置系统环境变量

添加HADOOP_HOME环境变量

右击我的电脑-》属性-》高级-》环境变量-》系统变量-》新建

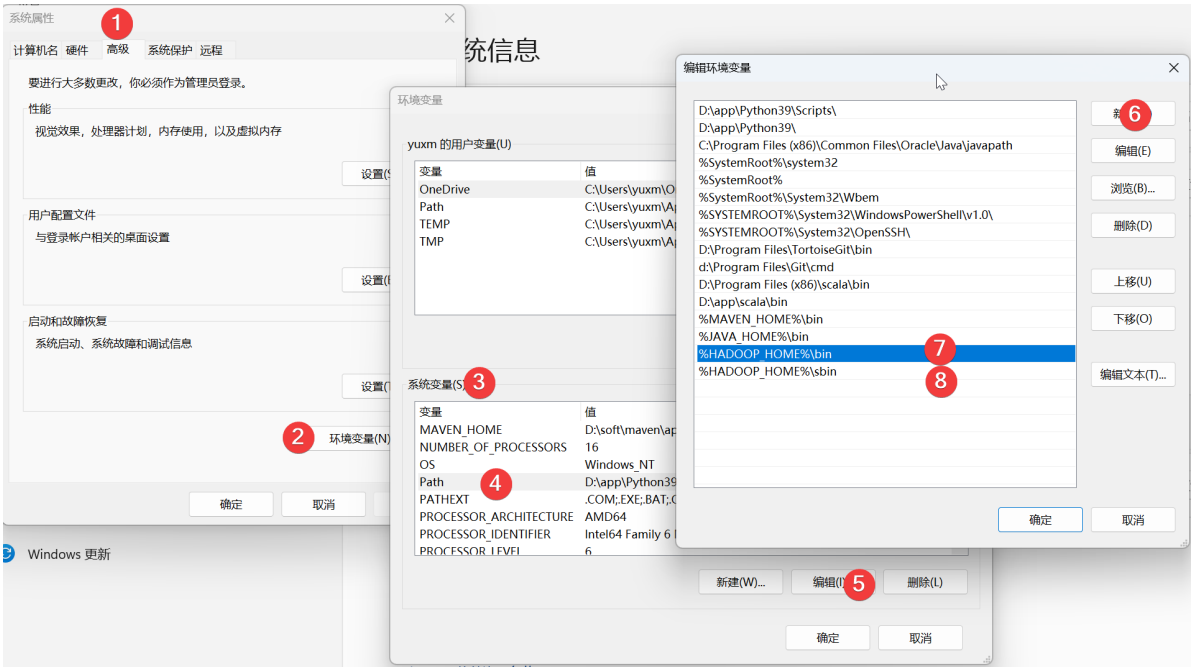
HADOOP_HOME=D:\spark\hadoop-3.1.4

(具体路径根据hadoop3.3.1的路径修改)



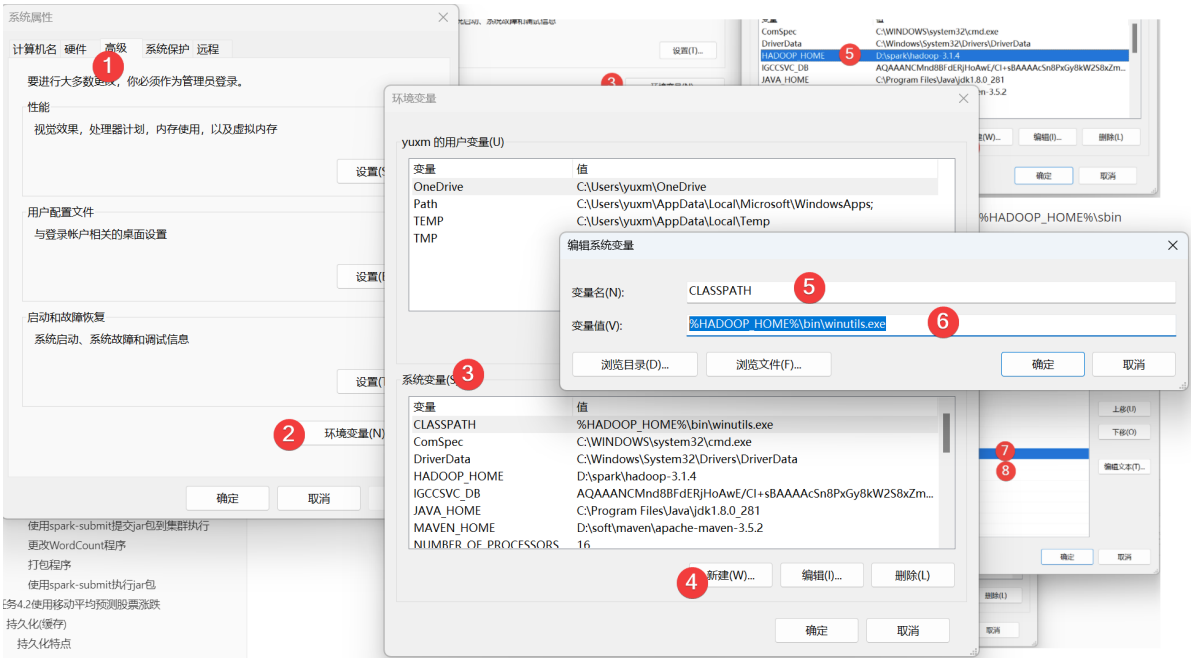
修改Path环境变量

右击我的电脑-》属性-》高级-》环境变量-》系统变量-》修改Path,并添加: %HADOOP_HOME%\bin 和 %HADOOP_HOME%\sbin



添加CLASSPATH环境变量

右击我的电脑-》属性-》高级-》环境变量-》系统变量-》添加CLASSPATH = %HADOOP_HOME%\bin\winutils.exe



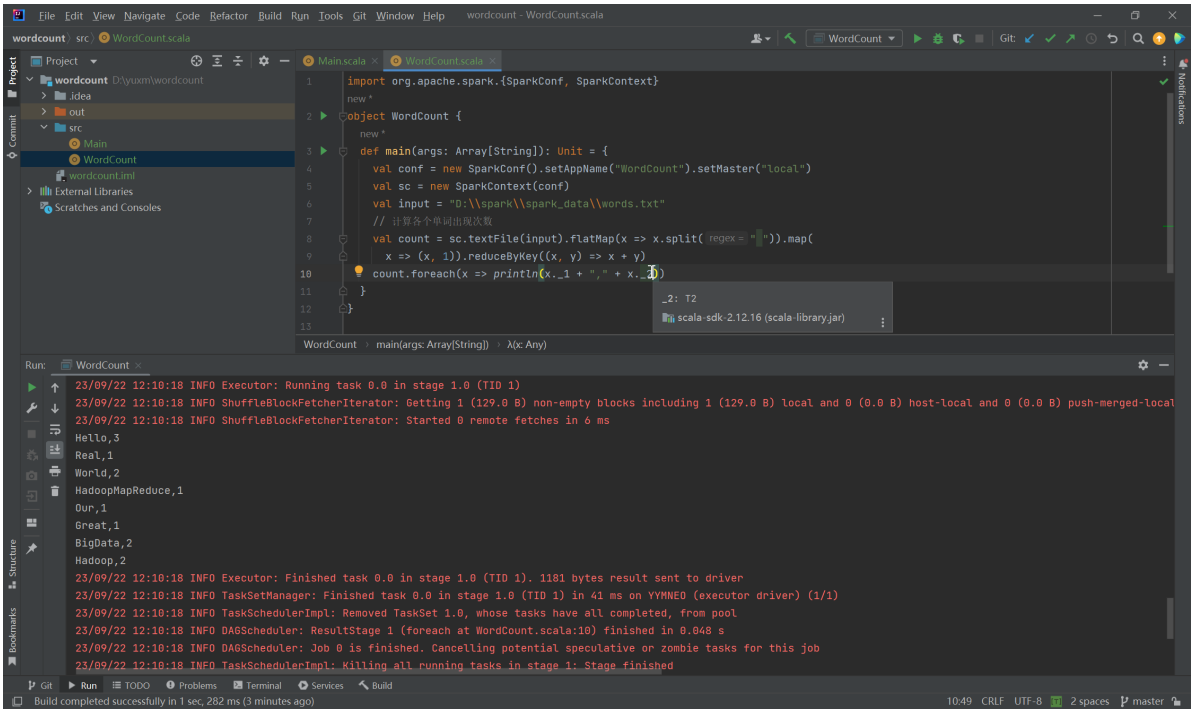
配置完成后，需要重启电脑才能生效

重启电脑。

编写Spark程序WordCount

```scala
import org.apache.spark.{SparkConf, SparkContext}
object WordCount {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("WordCount").setMaster("local")
    val sc = new SparkContext(conf)
    val input = "D:\\spark\\spark_data\\words.txt"
    // 计算各个单词出现次数
    val count = sc.textFile(input).flatMap(x => x.split(" ")).map(
      x => (x, 1)).reduceByKey((x, y) => x + y)
    count.foreach(x => println(x._1 + "," + x._2))
  }
}
```

运行结果如



运行Spark程序

在开发环境下运行 Spark

通过SparkConf对象的SetMaster方法连接到集群环境，直接运行 Spark程序

通过setMaster(URL),URL运行模式有：

运行模式	含义
local	使用一个工作线程在本地模式下运行Spark（即非并行）
local[K]	使用*K*个工作线程在本地模式下运行Spark（理想情况下，将*K*设置为机器上的内核数）
local[*]	在本地模式下运行Spark，工作线程与机器上的逻辑内核一样多
spark://HOST:PORT	连接到指定端口的Spark独立集群上，默认为7077端口
mesos://HOST:PORT	连接到指定端口的Mesos集群
yarn	根据配置的值连接到YARN集群，使用yarn-client或yarn-cluster模式
--deploy-mode client	客户端运行模式
--deploy-mode cluster	集群运行模式

在集群环境中运行Spark

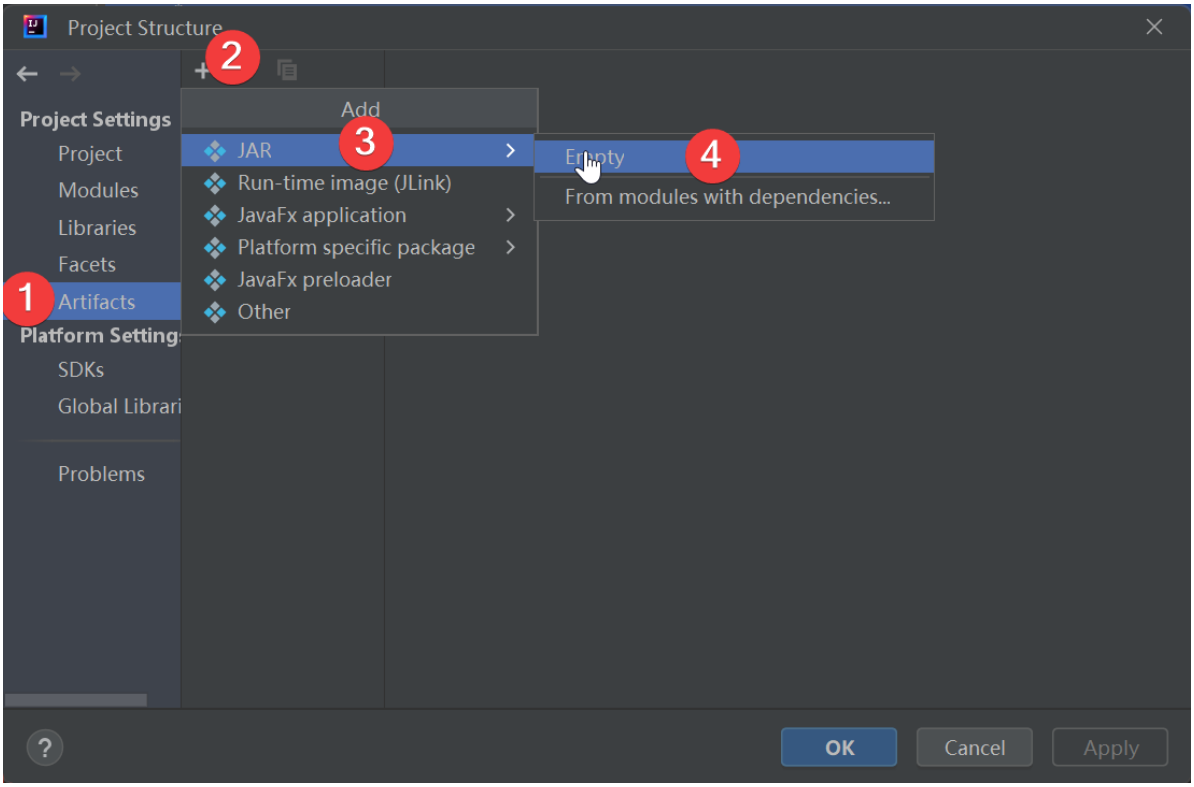
使用spark-submit提交jar包到集群执行

添加WordCount1程序

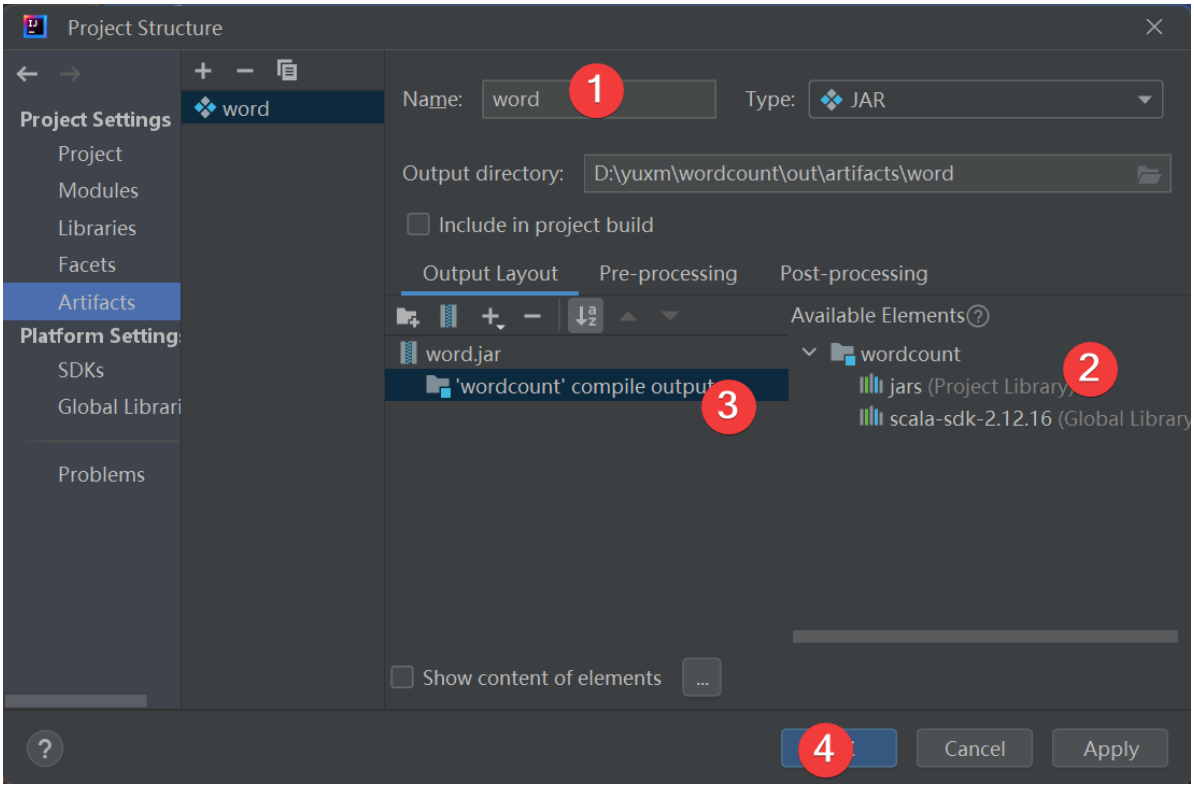
```
import org.apache.spark.{SparkConf, SparkContext}
case object WordCount1
{
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("WordCount").setMaster("local")
    val sc = new SparkContext(conf)

    val input = args(0)
    val output = args(1)
    //计算各个单词出现次数
    val count = sc.textFile(input).flatMap(x => x.split(" ")).map(x =>
      (x, 1)).reduceByKey((x, y) => x + y)
    count.foreach(x => println(x._1 + "," + x._2))
    count.repartition(1).saveAsTextFile(output)
  }
}

```

打包程序

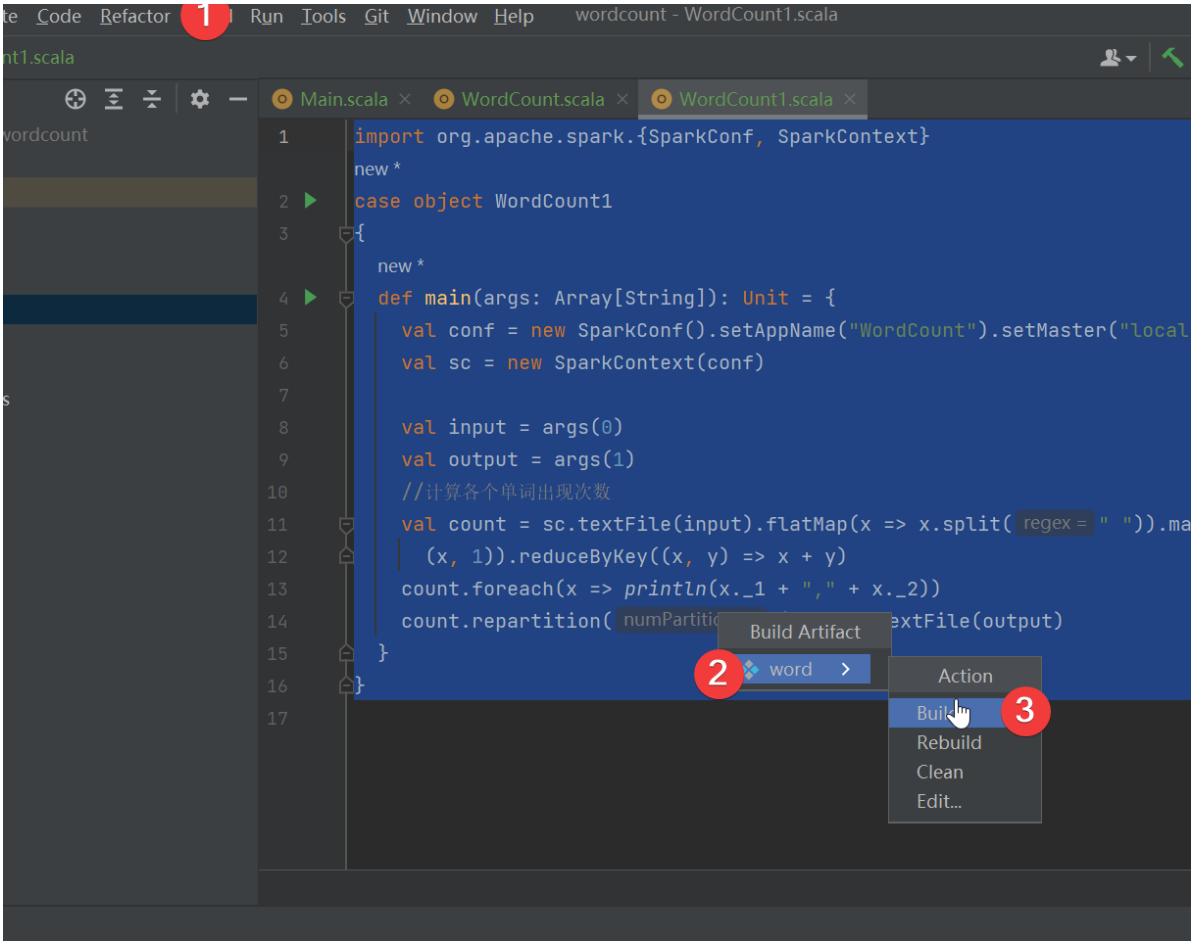
- 选择菜单File->Project Structure->Artifacts->"+"->JAR->Empty



- 在弹出窗中输入： jar包名: word; 双击右下侧栏的wordcount compile output,则自动转移到左侧； 点击OK

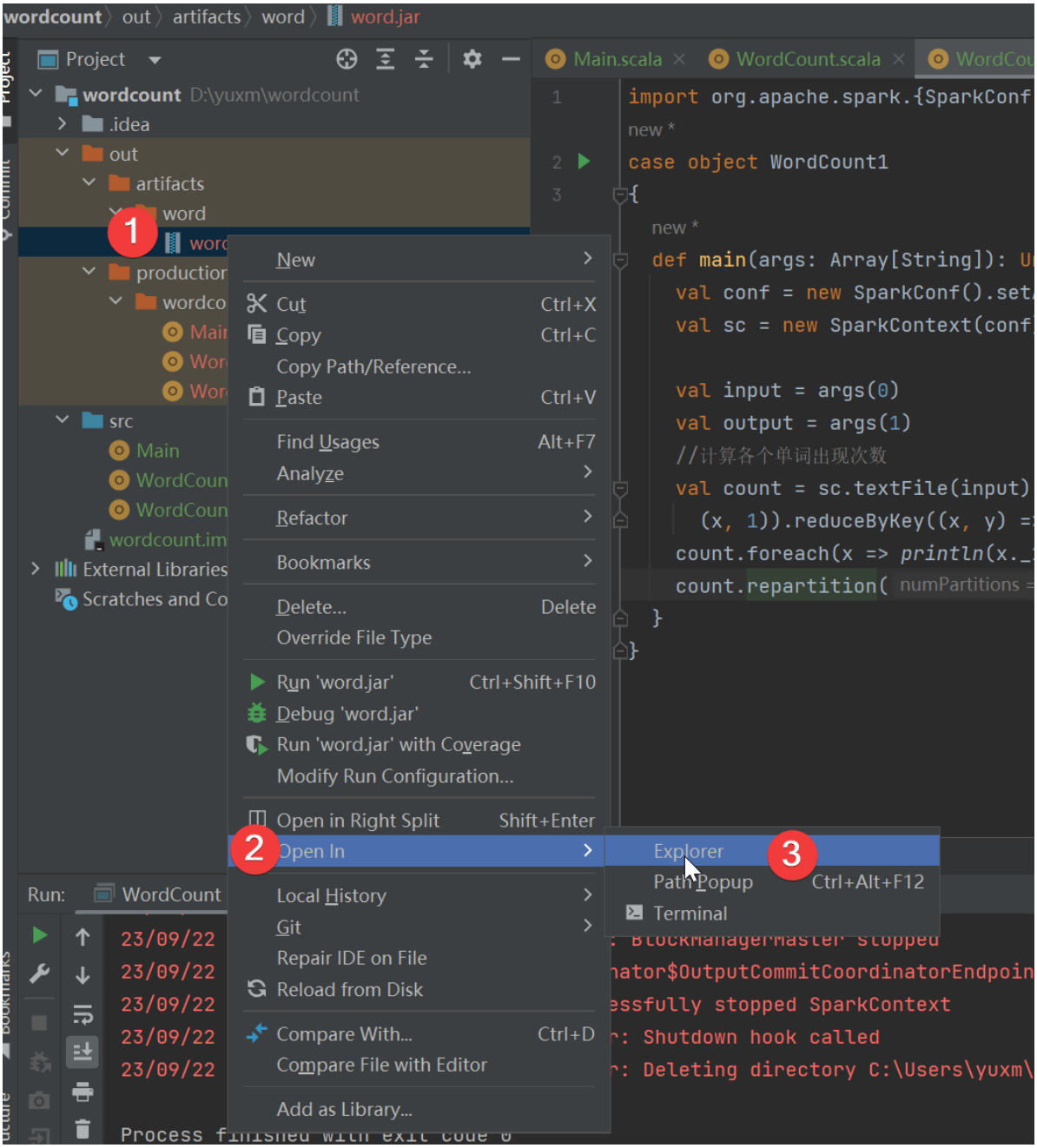


- 编译生成artifact: 选择菜单栏中的“Build->Build Artifacts->word->Build,命令

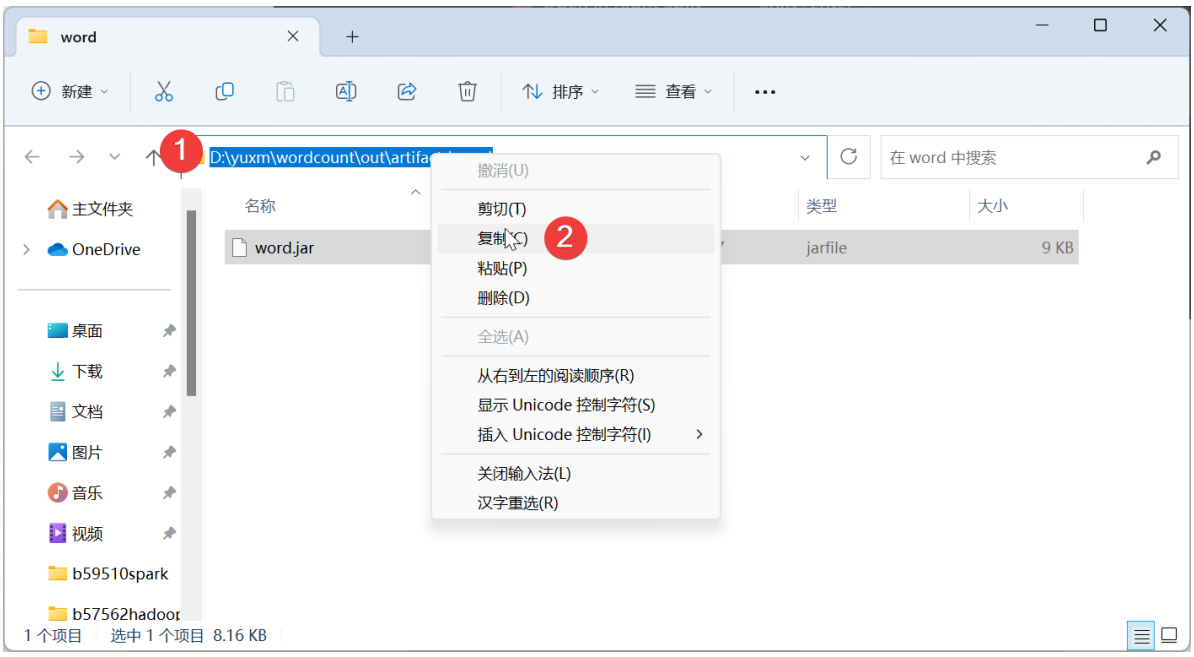


复制输出路径

- 打开jar所在目录: 右击out目录下的word.jar，Open In Explorer:

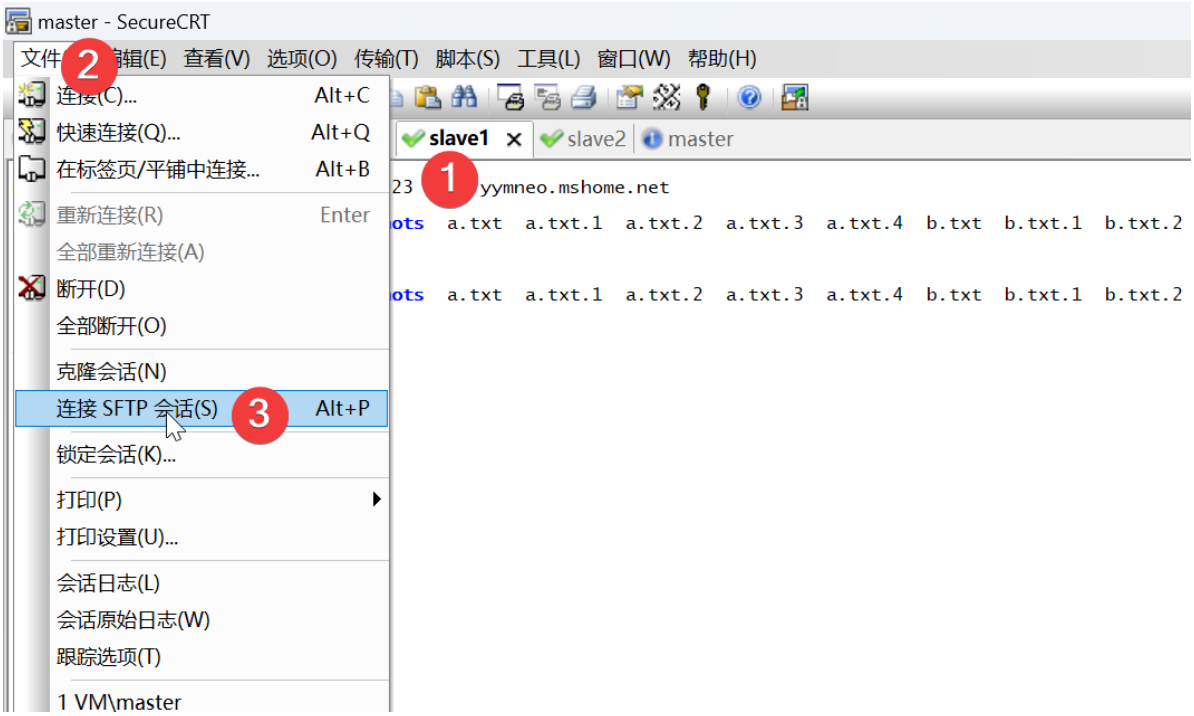


- 复制jar所在的目录 Path，例如： D:\yuxm\wordcount\out\artifacts\word



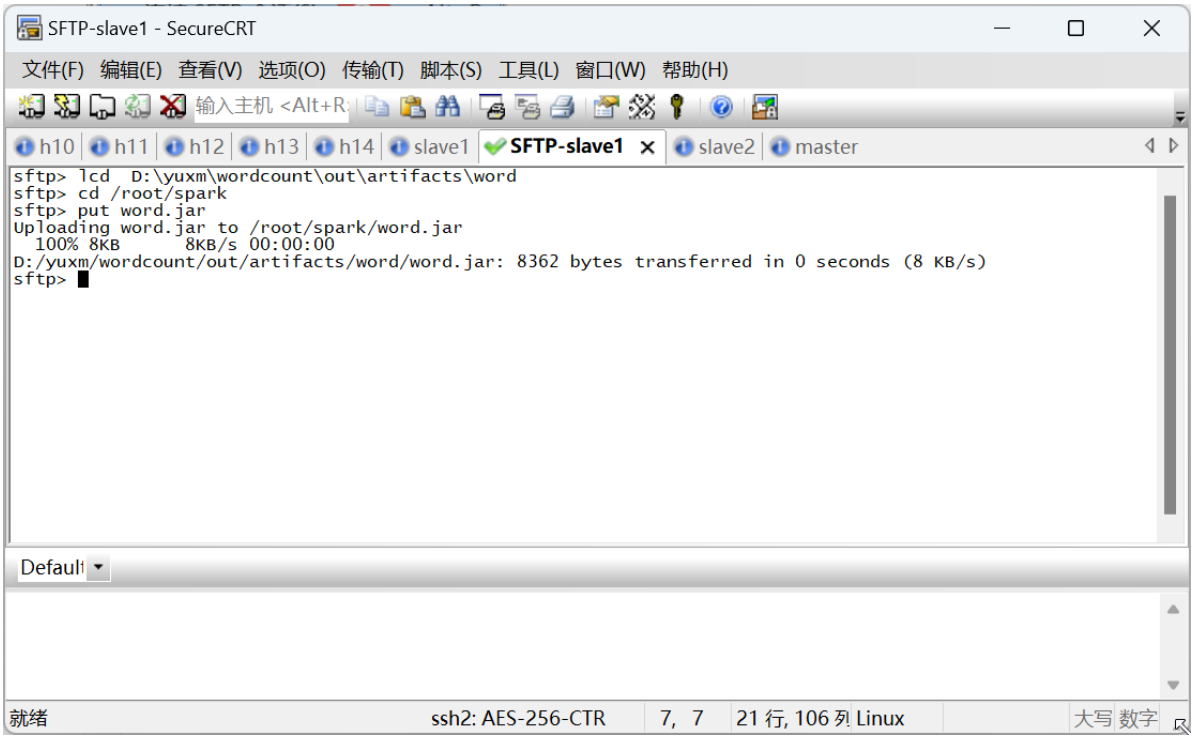
上传程序到集群目录下

使用secureCrt连接集群的从机slave1，该机曾在上一章中下载测试数据过。打开文件-》连接SFTP会话



lcd jar所有的目录Path，如：lcd D:\yuxm\wordcount\out\artifacts\word

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



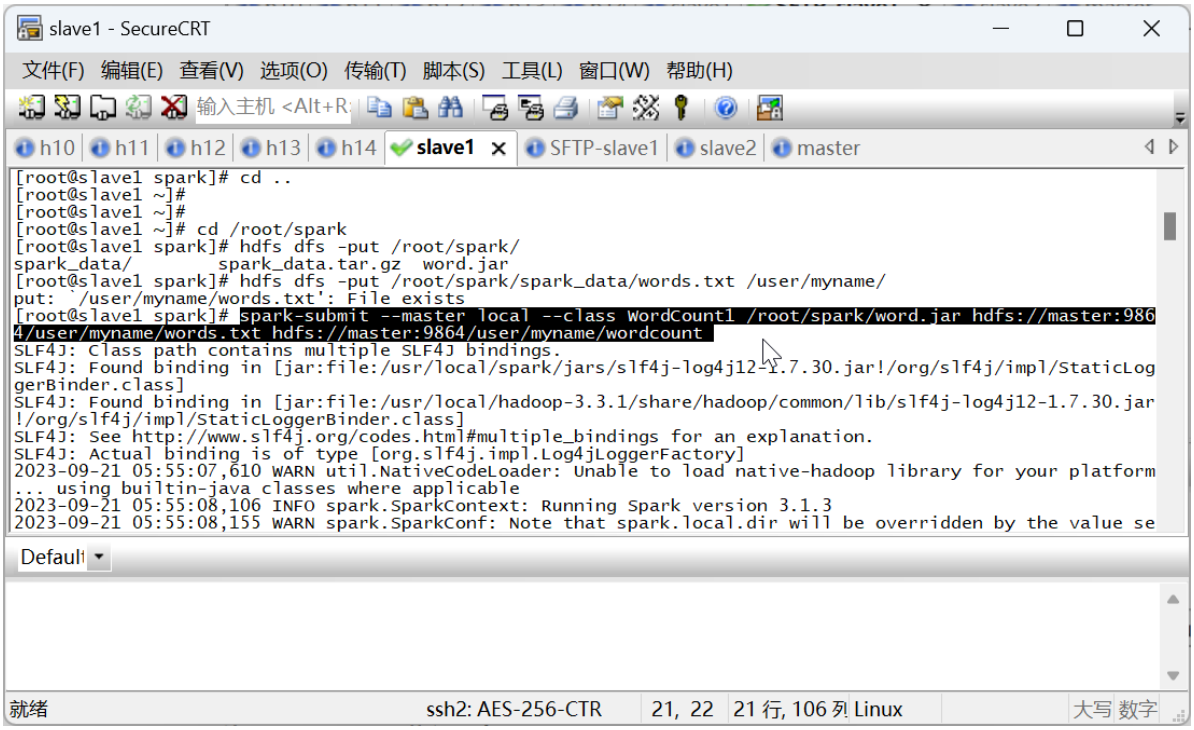
使用spark-submit执行jar包

切换到slave1标签，执行如下命令：

```
hdfs dfs -put /root/spark/spark_data/words.txt /user/myname/
spark-submit --master local --class WordCount1 /root/spark/word.jar
hdfs://master:9864/user/myname/words.txt
hdfs://master:9864/user/myname/wordcount
```

注意：第2行是由两行组成，两个hdfs文件分别表示待分析文件，和分析结果文件，要与上一行一起执行！

运行结果如：



spark-submit语法参考：

spark-submit提交JAR包到集群有一定的格式要求，需要设置一些参数，语法如下。

spark-submit --class --master --deploy-mode --conf <"key=value"> ... # other options [application-arguments]

spark-submit参数解释

--class：应用程序的入口点，指主程序。

--master：指定要连接的集群URL。

--deploy-mode：是否将驱动程序部署在工作节点（cluster）或本地作为外部客户端（client）。

--conf：设置任意Spark配置属性，即允许使用key=value格式设置任意的SparkConf配置选项。

application-jar：包含应用程序和所有依赖关系的捆绑JAR的路径。

application-arguments：传递给主类的main方法的参数。

任务4.2统计分析竞赛网站用户访问日志数据

任务描述

网站的访问量是衡量网站吸引力的重要指标，raceData.csv文件中有2020年和2021年共两个年份的数据，按年份统计竞赛网站每月的访问量，展示的结果会更加清晰，方便网站运营人员后续的分析。

本节的任务如下：

1. 学习Spark的持久化（缓存）和数据分区。
2. 在IntelliJ IDEA中使用Spark编程统计竞赛网站每月的访问量。
3. 按年份分区，将存储结果保存至HDFS中

设置RDD持久化

由于Spark RDD是惰性求值的，因此如果需要多次使用同一个RDD，那么调用行动操作时每次都需要计算该RDD并执行它依赖的其他转换操作。

在需要进行多次迭代的计算中，由于常常需要多次使用同一组数据，因此计算资源消耗会非常大，为了避免多次计算同一个RDD，可以在Spark中设置数据持久化

持久化特点

- 由参与计算该RDD的节点各自保存自己所求出的分区数据
- 在再一次需要计算该RDD时将不需要重新计算，直接取各分区保存好的数据即可
- 如某节点故障，Spark需要用到缓存时会重算丢失的分区
- 在内存充足的情况下，可以使用双副本保存的高可靠机制

持久化操作

RDD的持久化操作有cache()和persist()两种方法。cache()方法默认使用StorageLevel的MEMORY_ONLY。

可通过org.apache.spark.storage.StorageLevel

常用的StorageLevel存储级别有：

级别	所需内存空间程度	所需CPU计算时间程度	是否在内存中	是否在磁盘上	备注
MEMORY_ONLY	高	低	是	否	数据仅保留在内存中
MEMOTY_ONLY_SER	低	高	是	否	数据序列化后保存在内存中
MEMORY_AND_DISK	高	中等	部分	部分	数据先写到内存中，内存放不下则溢写到磁盘上
MEMORY_AND_DISK_SER	低	高	部分	部分	序列化的数据先写到内存中，内存不足则溢写到磁盘上
DISK_ONLY	低	高	否	是	数据仅存在磁盘上

存储级别设置

- 第一次计算后就会根据存储级别保存数据
- 只有未曾设置存储级别的RDD才能设置存储级别，已设置的不能修改
- 针对内存中的存储策略，如果内存不足的话，会使用LRU缓存策略清除最老的分区
- 缓存在磁盘上的数据不会清除
- 内存不够存放所有的数据，则数据不会持久化，下次操作时没有持久化的数据需要从源头处重新计算
- 人为清除不需要的缓存，使用unpersist()

实例源码

```
val data=sc.parallelize(List(1,2,3,4,5,6)).map(x=>x*x)
println(data.sum)
println(data.mean)
```

```
import org.apache.spark.storage.StorageLevel
val data=sc.parallelize(List(1,2,3,4,5,6)).map(x=>x*x)
data.persist(StorageLevel.MEMORY_ONLY)
println(data.sum)
println(data.mean)
```

运行结果:

```
scala> import org.apache.spark.storage.StorageLevel
import org.apache.spark.storage.StorageLevel

scala> val data=sc.parallelize(List(1,2,3,4,5,6)).map(x=>x*x)
data: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[16] at map at
<console>:29

scala> data.persist(StorageLevel.MEMORY_ONLY)
res8: data.type = MapPartitionsRDD[16] at map at <console>:29

scala> println(data.sum)
91.0

scala> println(data.mean)
15.166666666666666

scala>
```

可见在计算平均值时明显比计算求和要快很多,分别是0.27s和5.24s。 持久化后，求和计算时会对data重新进行计算，求平均直接从内存中读取持久化数据，所以更快。

数据分区

Spark RDD是多个分区组成的数据集合，在分布式程序中，通信的代价是很大的，因此控制数据分区、减少网络传输是提高整体性能的一个重要的方面。

分区特点

- 数据分区是为了减少网络传输提高整体性能
- 只有键值对类型的RDD才能设置分区方式；非键值对类型的RDD分区值是None
- 系统是根据一个针键的函数对元素进行分区的，虽然不能控制每个键划分到哪个节点，但可以控制相同的键落在同一分区
- 每个RDD的分区 ID范围是0~numParitions-1

设置分区方式

RDD.partitionBy(分区方式)

分区方式

- 哈希分区，HashPartitioner
- 范围分区，RangePartitioner
- 自定义分区

自定义分区

需要继承org.apache.spark.Partitioner类

需要实现三个方法

- def numPartitions:Int
  - 返回想要创建的分区个数
- def getParatition(key:Any)
  - 需要对输入的key做处理，
    然后返回该key的分区ID,
    范围一定是0~numPartitions-1
- equals(other:Any)
  - 判断相等

自定义分区实例源码

MyPartition

```
import org.apache.spark.Partitioner
class MyPartition(numParts:Int) extends Partitioner{
  override def numPartitions: Int = numParts
  override def getPartition(key: Any): Int = {
   if (key.toString().toInt%2==0) {
    0
   } else {
    1
   }
  }
  override def equals(other: Any): Boolean = other match {
   case mypartition: MyPartition =>
    mypartition.numPartitions == numPartitions
   case _ =>
    false
  }
 }
```

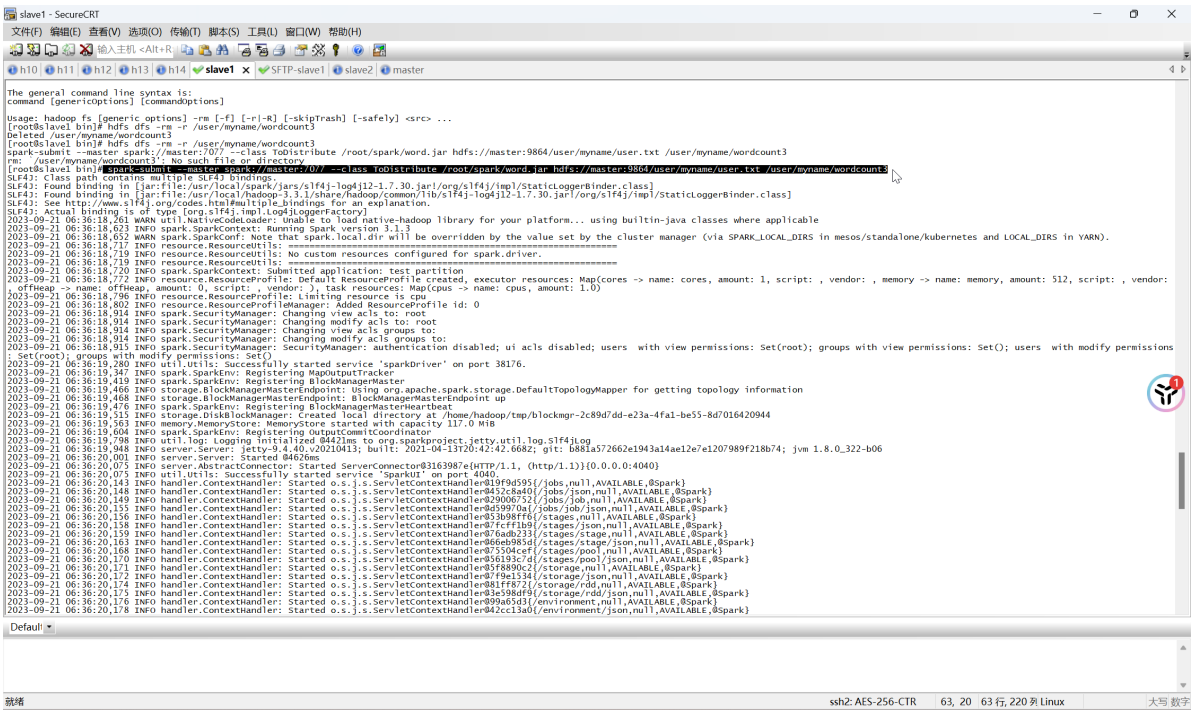
ToDistribute

```scala
import org.apache.spark.{SparkConf,SparkContext}
object ToDistribute {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("test partition")
    val sc = new SparkContext(conf)
    val input = args(0)
    val output = args(1)
    val data = sc.textFile(input).map{x=>val y = x.split(",");(y(0),y(1))}
    val data2 = data.partitionBy(new MyPartition(2))
    data2.saveAsTextFile(output)
   }
 }
```

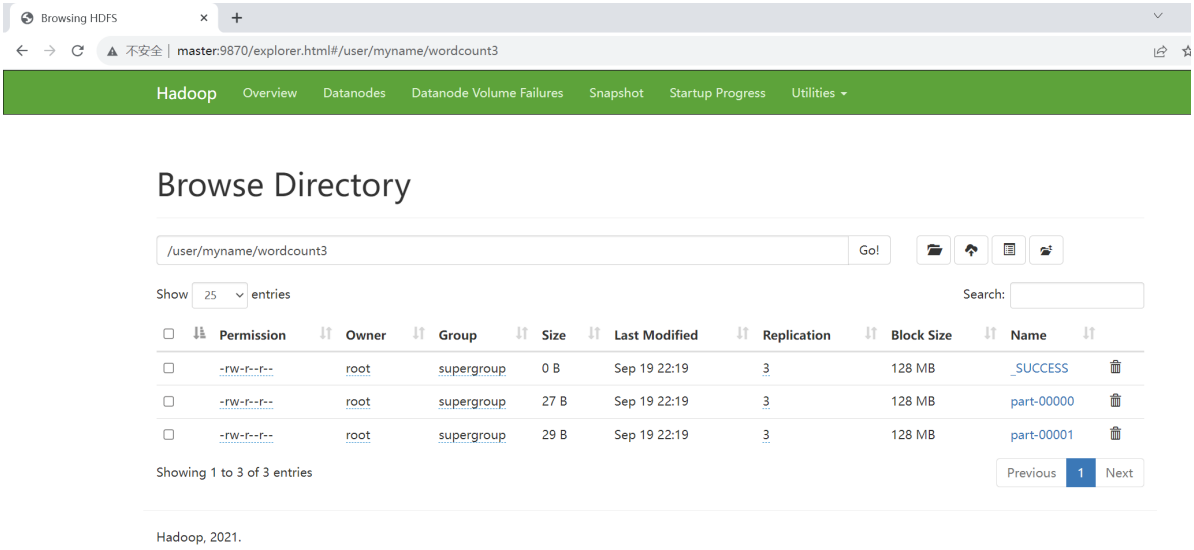
测试运行

- 编译，build Artifact
- 上传build后的word.jar
- yarn方式运行word.jar

```
hdfs dfs -rm -r /user/myname/wordcount3
spark-submit --master spark://master:7077 --class ToDistribute
/root/spark/word.jar hdfs://master:9864/user/myname/user.txt
/user/myname/wordcount3
```



- hdfs上运行结果



重新分区

两种方法重新分区:coalesce和repartition

coalesce

函数定义:

coalesce(numPartitions :Int,shuffle:Boolean=false

- numPartitions:重新分区数
- shuffle:是否shuffle,默认false,
  分区数只能比原分区小;
  true时,分区大小都可以

repartition

函数定义:
repartition(numPartitions:Int)

测试源码

```
val rdd=sc.parallelize(List(1,2,3,4,5,6,7))
rdd.partitions.size
val rdd1=rdd.coalesce(5)
rdd1.partitions.size
val rdd2=rdd.coalesce(5,true)
rdd2.partitions.size
rdd.repartition(10).partitions.size
```

运行结果:

```
scala> val rdd=sc.parallelize(List(1,2,3,4,5,6,7))
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize
at <console>:24

scala> rdd.partitions.size
res0: Int = 4

scala> val rdd1=rdd.coalesce(5)
```

```
rdd1: org.apache.spark.rdd.RDD[Int] = CoalescedRDD[1] at coalesce at
<console>:25

scala> rdd1.partitions.size
res1: Int = 4

scala> val rdd2=rdd.coalesce(5,true)
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[5] at coalesce at
<console>:25

scala> rdd2.partitions.size
res2: Int = 5

scala> rdd.repartition(10).partitions.size
res3: Int = 10
```

获取分区方式

RDD.partitioner.isDefined是否有值
RDD.partitioner.get,获取spark.Partitioner对象

计算竞赛网站每月的访问量

分析:

对于任何一个网站而言，访问量都是很重要的，访问量可以直观地反映出该网站当前的受欢迎程度。而对于竞赛网站而言，访问量同时还可以反映出当前时间段的竞赛项目热度。对网站的访问量进行统计，可以为网站后台运营、维护提供合理建议，更为重要的是可以帮助网站运营人员对竞赛的举办时间、周期等事宜进行更合理的规划。在原始数据中存在一个时间字段，表示用户访问网站的时间，格式为“yyyy/MM/dd HH:mm:ss”

现需要对网站每月的访问量进行统计，因此需要对该字段进行多次分割。通过textFile()方法读取原始数据。使用map()方法对原始数据进行映射，获取“yyyy-MM”格式的时间，将时间映射为键值对，键为年和月，值为1，形如“(2020-5,1)”。通过reduceByKey()方法对相同键的值进行累加。

源码

```
import org.apache.spark.{Partitioner, SparkConf, SparkContext}
import org.apache.spark.rdd.RDD

object raceRdd {
  def main(args: Array[String]): Unit = {
    val sparkConf = new
SparkConf().setMaster("local").setAppName("WordCount")
    val sc = new SparkContext(sparkConf)
    val input = args(0)
    sc.setLogLevel("WARN")
    val data = sc.textFile(input)
    val split: RDD[(String, Int)] = data.map(x => x.split(",")(4))
      .map(s => s.split(" "))
      .map(y => y(0).split("/"))
      .map(z => (z(0) + "-" + z(1), 1))
      .reduceByKey(_ + _)
```

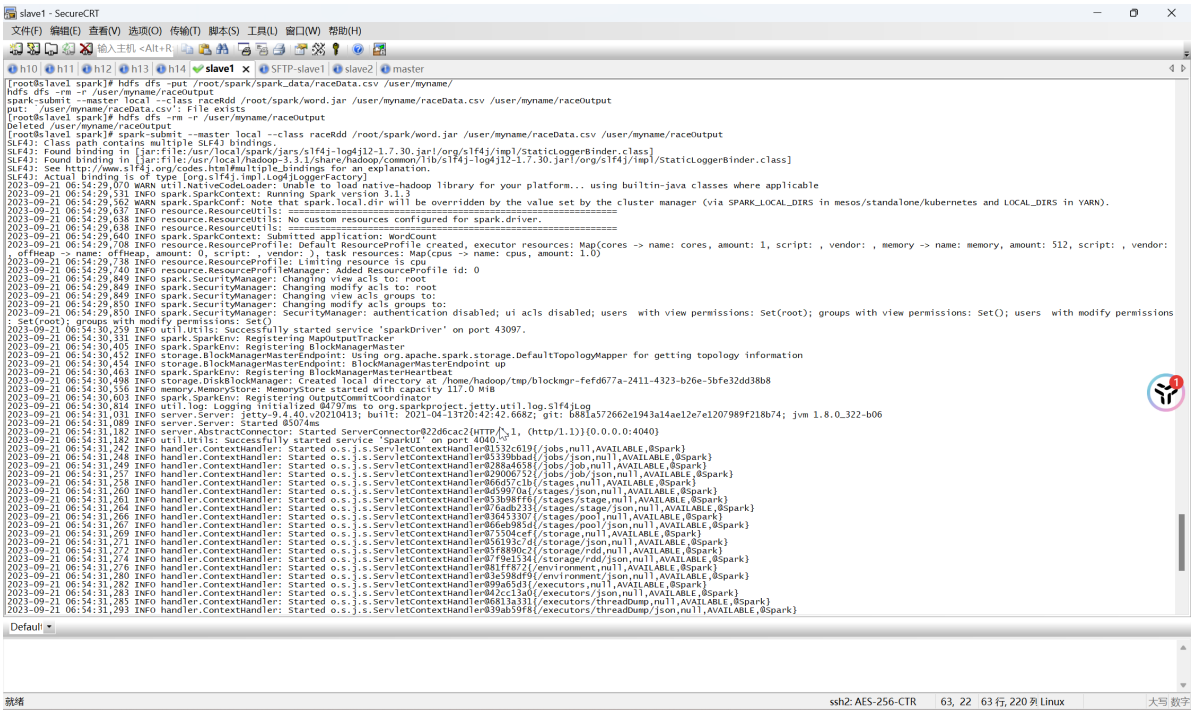
```
    split.collect().foreach(println)
    val output = args(1)
    val parition: RDD[(String, Int)] = split.partitionBy(new MyPartitioner)
    parition.saveAsTextFile(output)
    sc.stop()
  }
  class MyPartitioner extends Partitioner {
    //分区数量
    override def numPartitions: Int = 2
    ///  根据数据的key值返回数据所在的分区索引（从0开始）
    override def getPartition(key: Any): Int = {
      key match {
        case "2021-1" => 1
        case "2021-2" => 1
        case _ => 0
      }
    }
  }
}
```

编译并上传jar包

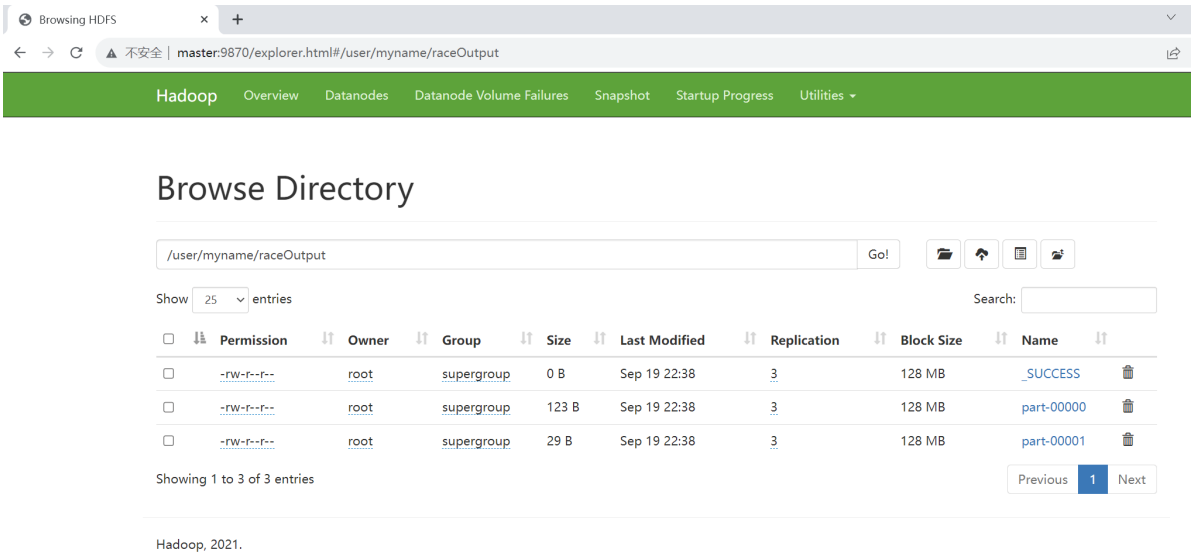
在集群上运行

```
hdfs dfs -put /root/spark/spark_data/raceData.csv /user/myname/
hdfs dfs -rm -r /user/myname/raceOutput
spark-submit --master local --class raceRdd /root/spark/word.jar
/user/myname/raceData.csv /user/myname/raceOutput
```

运行结果：



HDFS上运行结果：



实训作业

竞赛网站访问日志分析

训练要点

- (1)搭建Spurk工程环境。
- (2) Spark编程。
- (3)通过spark-submit提交应用。

需求说明

某竞赛网站每年都会开展数据挖据的竞赛，在竞赛期间网站会有大量人群访问，生成了大量的用户访向记录。现在提供2016年10月到2017年6月的部分脱敏访问日志数据。日志数据的基本内容如图所示，仅提供以下6个字段：
属性名称,属性解析
Id,序号
Content_id,网页ID
Page_path,网址
Userid,用户ID
Sessionid,缓存生成ID
Date_time,访问时间
要求根据提供的用户访问日志数据，利用Spark技术统计访向的用户数、被访问的不同网页个数以及每月的访问量，并将结果保存到HDFS上。

实现思路及步骤

- (1)配置好Spark的IntelliJ IDEA开发环境。
- (2)启动IntelliJ IDEA，并进行Spark编程。
- (3)对访向记录中的网页去重，统计本周期内被访问网页的个数。
- (4) userid为用户注册登录的标识，对userid去重，统计登录用户的数量。
- (5)按月统计访问记录数。
- (6)将结果保存到不同文件中。
- (7)打包Spark工程，在集群提交应用程序

要求

- 1，截图IDEA上的源码
- 2，截图crt上运行的结果
- 3，截图在hdfs上的运行结果

数据准备

```
mkdir -p /root/spark
 cd /root/spark
 wget http://bigdata.hddly.cn//b46488/file/chap4/viewlog.tar.gz
 tar -xzvf ./viewlog.tar.gz
 hdfs dfs -put ./jc_content_viewlog.txt  /user/myname/

```

源码参考

```
package chap4

 import org.apache.spark.rdd.RDD
 import org.apache.spark.{SparkConf, SparkContext}

 //spark-submit :
 //spark-submit --master yarn --deploy-mode client  --class  chap4.LogCount /root/word.jar /user/myname/jc_content_viewlog.txt /user/myname/out_wy_count /user/myname/out_user_count /user/myname/out_ny_count
 object LogCount {
  def main(args: Array[String]): Unit = {

   if (args.length < 2) {
    println("请指定input和output")
    System.exit(1) //非0表示非正常退出程序
   }

   //TODO 1.env/准备sc/SparkContext/Spark上下文执行环境
   val conf: SparkConf = new SparkConf().setAppName("wc").setMaster("local")
   val sc: SparkContext = new SparkContext(conf)
   sc.setLogLevel("WARN")

   //TODO 2.source/读取数据
   //RDD:A Resilient Distributed Dataset (RDD):弹性分布式数据集,简单理解为分布式集合!使用起来和普通集合一样简单!
   //RDD[就是一行行的数据]
   val logs_all: RDD[Array[String]] = sc.textFile(args(0)).map {
    _.split(",")
   }
   //TODO 3.transformation/数据操作/转换
   //对访问记录中的网页去重，统计本周期内被访问网页的个数
  //
478896,1043,/jszx/1043.jhtml,14884,F6D362B9AFAC436D153B7084EF3BA332,2017-03-01 00:23:07
   val wy_log: RDD[String] = logs_all.map(x => (x(1).toString)).distinct()
   val wy_count: RDD[(String, Int)] = wy_log.map(("wy_zs", _)).groupByKey().map(x => (x._1, x._2.size))

```

```
    //userid为用户注册登录的标识，对userid去重，统计登录用户的数量
    val user_log: RDD[String] = logs_all.map(x => (x(3).toString)).distinct()
    val user_count: RDD[(String, Int)] = user_log.map(("user_zs",
_)).groupByKey().map(x => (x._1, x._2.size))
    //按月统计访问记录数
    val logs_all_new = logs_all.map { x => (x(0), x(1), x(2), x(3), x(4),
x(5), date_time(x(5))) }
    val ny_count: RDD[(String, Int)] = logs_all_new.map(x => (x._7,
1)).reduceByKey((a, b) => a + b)

    //TODO 4.sink/输出
    //输出到指定path(可以是文件/夹)
    println("be writing to :" + args(1))
    wy_count.repartition(1).saveAsTextFile(args(1))
    println("be writing to :" + args(2))
    user_count.repartition(1).saveAsTextFile(args(2))
    println("be writing to :" + args(3))
    ny_count.repartition(1).saveAsTextFile(args(3))
    //为了便于查看Web-UI可以让程序睡一会
    Thread.sleep(1000 * 60)

    //TODO 5.关闭资源
    sc.stop()
  }

  //获取年月，时间段作为输入参数
  def date_time(date: String): String = {
    if (date.trim.length == "2017-03-01 00:23:07".length) {
      val nianye = date.trim.substring(0, 7)
      nianye
    }
    else
      "1900-01"
  }


}
```

crt运行参考

```
hdfs dfs -rm -r  hdfs://master:9864/user/myname/out_wy_count
hdfs dfs -rm -r  hdfs://master:9864/user/myname/out_user_count
hdfs dfs -rm -r  hdfs://master:9864/user/myname/out_ny_count

spark-submit --master yarn --deploy-mode client  --class  chap4.LogCount
/root/spark/word.jar hdfs://master:9864/user/myname/jc_content_viewlog.txt
hdfs://master:9864/user/myname/out_wy_count
hdfs://master:9864/user/myname/out_user_count
hdfs://master:9864/user/myname/out_ny_count
```

作业要求

- 1，截图IDEA上的chap4.LogCount 源码
- 2，截图crt上运行chap4.LogCount的命令行，及运行结果
- 3， 截图hdfs上/user/myname/out_wy_count，out_user_count，out_ny_count三个目录下的文件内容

版本历史

- Ver1.2-20220928
- Ver1.3-20221003
    - 更新课堂练习和实训内容
- Ver1.4-20221006
    - 增加实训视频学习内容,修改部分脚本
- Ver1.5-20221009
    - 增加理论知识的视频学习内容
- Ver1.6-20221015
    - 增加实训的视频学习
- Ver1.7-20230828
    - 增加markdown版本
- Ver1.8-20230922
    - 更新案例和文档内容

上一章节 下一章节