



北京邮电大学

Beijing University of Posts and Telecommunications

# 华为云鲲鹏大数据基础实验体系 5

## Spark 安装及应用实践

编写负责人：

鄂海红、张忠宝

参编人员：

朱一凡、宋美娜、程祥、

汤子辰、罗子霄、王浩田、魏文定、刘钟允 等

# 华为云鲲鹏大数据基础实验体系 5: Spark 安装及应用实践

## 1.1.1. 实验描述

本实验使用 Scala 语言编写 Spark 程序，完成单词计数任务、独立应用程序实现数据去重任务，并使用 Spark SQL 完成数据库读写任务。实验分为三个部分，首先，在华为云购买 4 台服务器，然后搭建 Hadoop 集群和 Spark 集群(YARN 模式)，接着使用 Scala 语言利用 Spark Core 编写程序，最后将程序打包在集群上运行；其次，使用 Scala 语言编写独立应用程序实现数据去重，并将程序打包在集群上运行；最后，使用 Spark SQL 读写数据库，包括在服务器上安装 MySQL 和通过 JDBC 连接数据库。

实验环境版本要求：

1. 服务器节点数量：4
2. 系统版本：Centos 7.6
3. Hadoop 版本：Apache Hadoop 2.7.7
4. Spark 版本：Apache Spark 2.1.1
5. JDK 版本：1.8.0\_292-b10
6. Scala 版本：scala2.11.8
7. IDEA 版本：IntelliJ IDEA Community Edition 2021.2.3
8. MySQL 版本：8.0

实验需要注意的问题：

序号	问题	解决方案
1	关于环境	在客户端打包程序的时候，使用实验提供的环境，要求 jdk 版本为 1.8，否则会出错；
2	新建类，无 Scala.class	<a href="https://blog.csdn.net/qq_16410733/article/details/85038832">https://blog.csdn.net/qq_16410733/article/details/85038832</a>
3	关于防火墙	一定要检查三个节点是否关闭了防火墙；
4	文件拷贝	注意主节点配置的文件是否全部拷贝给了数据节点，主要包括 (Hadoop 文件夹、hosts、.bash_profile)
5	Hadoop需要进一步验证是否能够正常使用	运行：hadoop jar ./hadoop-2.7.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar pi 10 10 输出结果：3.2000
6	关于 web 页面	web 页面默认在本地端是无法打开的，需要配置华为云的安全组，开放指定的端口，如：50070
7	关于Mysql命令	忘写分号：用“->”补上即可

## 1.1.2. 实验目的

1. 了解服务器配置的过程;
2. 熟悉使用 Scala 编写 Spark 程序;
3. 了解 Spark RDD 的工作原理;
4. 掌握在 Spark 集群上运行程序的方法;
5. 掌握使用Spark SQL读写数据库的方法。

## 1.1.3. 实验步骤

### 1.1.3.1. Spark core Scala 单词计数

#### 1. Hadoop 集群环境测试

在搭建 spark 环境之前,我们要保证我们的 Hadoop 集群的正常工作,这很关键。在第一章和第二章的实验基础上,开展本章实验。

本实验 JDK 版本为 1.8.0\_191-b12, Hadoop 版本为 Apache Hadoop 2.7.7。集群构建在华为云4台服务器上,主节点名称为 yty-2022140804-0001,从节点名称为 yty-2022140804-0002、yty-2022140804-0003、yty-2022140804-0004,请在做本实验时,将命令中的服务器名称改为自己的名称。

**步骤1:**在构建好 Hadoop 平台的主节点开启集群: start-all.sh,并在四个节点的终端执行 jps 命令: 并执行 ifconfig。

```
[root@yty-2022140804-0001 ~]# jps
2177 ResourceManager
1784 NameNode
1997 SecondaryNameNode
2474 Jps
[root@yty-2022140804-0001 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.25 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::f816:3eff:fef2:4a63 prefixlen 64 scopeid 0x20<link>
    ether fa:16:3e:f2:4a:63 txqueuelen 1000 (Ethernet)
    RX packets 707 bytes 107231 (104.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 666 bytes 178489 (174.3 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

图 4.8-1 实验截图

```
[root@yty-2022140804-0002 ~]# jps
1682 DataNode
1789 NodeManager
1932 Jps
[root@yty-2022140804-0002 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.184 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::f816:3eff:fef2:4a03 prefixlen 64 scopeid 0x20<link>
    ether fa:16:3e:f2:4a:03 txqueuelen 1000 (Ethernet)
    RX packets 527 bytes 79806 (77.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 487 bytes 156614 (152.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

图 4.8-2 实验截图

```
[root@yty-2022140804-0003 ~]# jps
1792 NodeManager
1934 Jps
1685 DataNode
[root@yty-2022140804-0003 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.130 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::f816:3eff:fef2:4acc prefixlen 64 scopeid 0x20<link>
    ether fa:16:3e:f2:4a:cc txqueuelen 1000 (Ethernet)
    RX packets 569 bytes 83127 (81.1 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 540 bytes 164542 (160.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

图 4.8-3 实验截图

```
[root@yty-2022140804-0004 ~]# jps
1791 NodeManager
1681 DataNode
1932 Jps
[root@yty-2022140804-0004 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.164 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::f816:3eff:fef2:4aee prefixlen 64 scopeid 0x20<link>
    ether fa:16:3e:f2:4a:ee txqueuelen 1000 (Ethernet)
    RX packets 572 bytes 69021 (67.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 577 bytes 90182 (88.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

图 4.8-4 实验截图

**步骤2:** 在主节点的 root 测试 Hadoop 的集群可用性，并执行 ifconfig。

```
hadoop jar ./hadoop-2.7.7/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.7.jar pi 10
10
```





```
cd /home/modules/hadoop-2.7.7/etc/hadoop
```

```
<property>
  <name>yarn.nodemanager.pmem-check-enabled</name>
  <value>>false</value>
</property>
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>>false</value>
</property>
```

```
scp yarn-site.xml yty-2022140804-0002:/home/modules/hadoop-2.7.7/etc/hadoop/yarn-site.xml
scp yarn-site.xml yty-2022140804-0003:/home/modules/hadoop-2.7.7/etc/hadoop/yarn-site.xml
scp yarn-site.xml yty-2022140804-0004:/home/modules/hadoop-2.7.7/etc/hadoop/yarn-site.xml
```

3) 使用 `ipsc` 查看集群是否启动成功;

```
spark-submit --class org.apache.spark.examples.SparkPi --master yarn --num-executors 4 --driver
-memory 1g --executor-memory 1g --executor-cores 1 spark-2.1.1-bin-hadoop2.7/examples/jars/s
park-examples_2.11-2.1.1.jar 10
```

```
22/10/31 11:04:31 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose  
leted, from pool  
22/10/31 11:04:31 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at  
nished in 1.209 s  
22/10/31 11:04:31 INFO scheduler.DAGScheduler: Job 0 finished: reduce at  
ook 1.473942 s  
Pi is roughly 3.141703141703142  
22/10/31 11:04:31 INFO server.ServerConnector: Stopped Spark@e194ea3{f  
}  
22/10/31 11:04:31 INFO handler.ContextHandler: Stopped o.s.j.s.ServletC  
7/stages/stage/kill,null,UNAVAILABLE,@Spark}
```

**步骤 6:** 运行 spark-shell 命令，查看 spark 和 scala 版本信息

```
22/10/31 11:05:46 WARN util.NativeCodeLoader: Unable to load native-hadoop lib
platform... using builtin-java classes where applicable
22/10/31 11:05:54 WARN metastore.ObjectStore: Version information not found in
ve.metastore.schema.verification is not enabled so recording the schema versi
22/10/31 11:05:54 WARN metastore.ObjectStore: Failed to get database default,
chObjectException
22/10/31 11:05:56 WARN metastore.ObjectStore: Failed to get database global_t
NoSuchObjectException
Spark context Web UI available at http://192.168.0.25:4040
Spark context available as 'sc' (master = local[*, app id = local-1667185548)
Spark session available as 'spark'.
Welcome to

  ____  __
 / ___/  / /
/ /   /  / /
/ /___/  / /
/_____/  / /
       /_/

version 2.1.1

Using Scala version 2.11.8 (Eclipse OpenJ9 VM, Java 1.8.0_292-ea)
Type in expressions to have them evaluated.
Type :help for more information.
```

图 4.8-8 实验截图

可以看到spark版本为2.1.1， scala版本为2.11.8。

### 3. Scala 程序编写

**步骤 1:** 创建项目，打开 IDEA(IDEA 需要在自己电脑上安装)，创建工程：

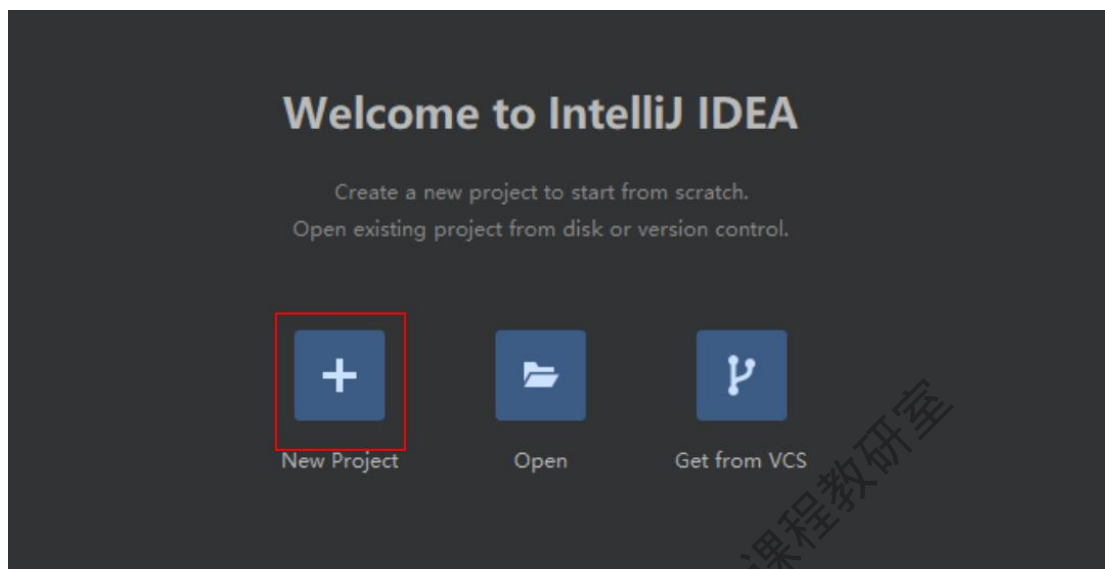


图 4.8-9 实验截图

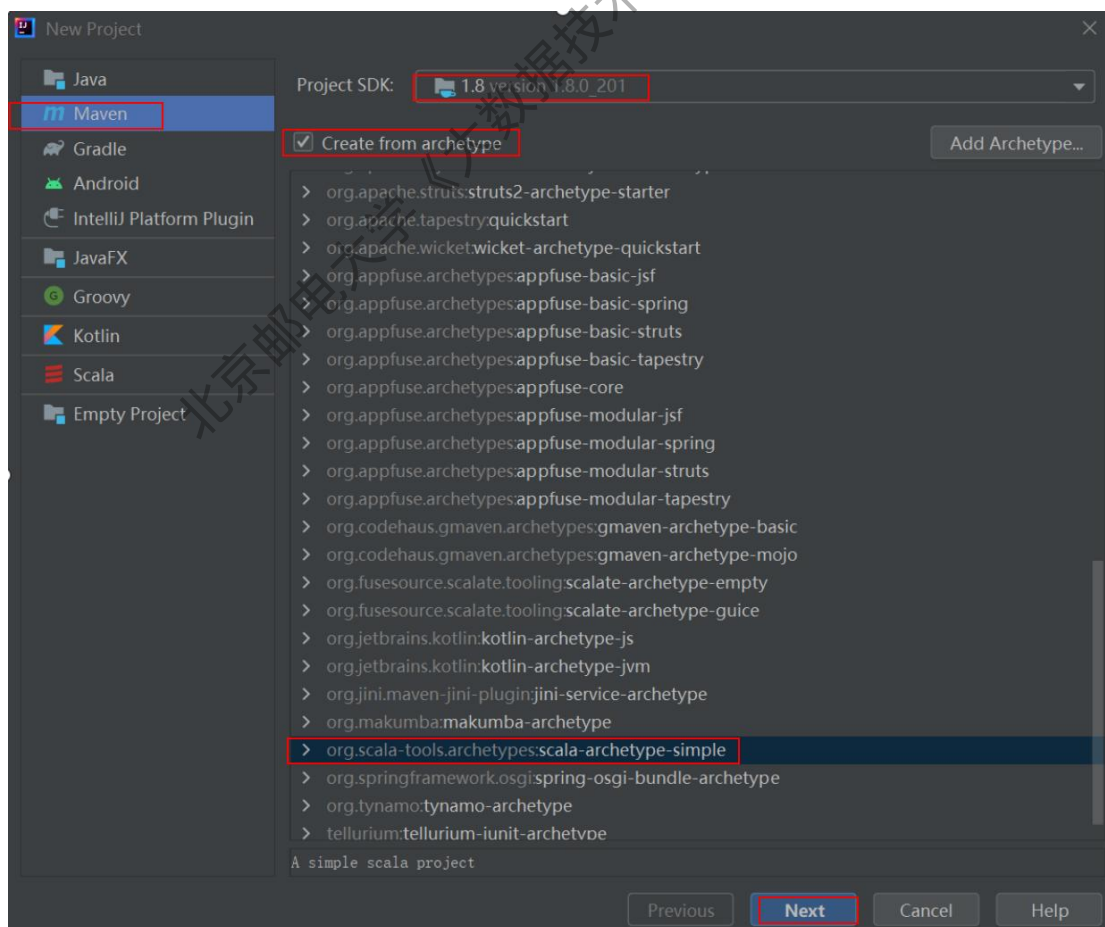


图 4.8-10 实验截图

若是使用的idea版本较新，如2022版本的，则需在创建项目时catalog项选择Maven Central才能找到scala-archetype-simple。

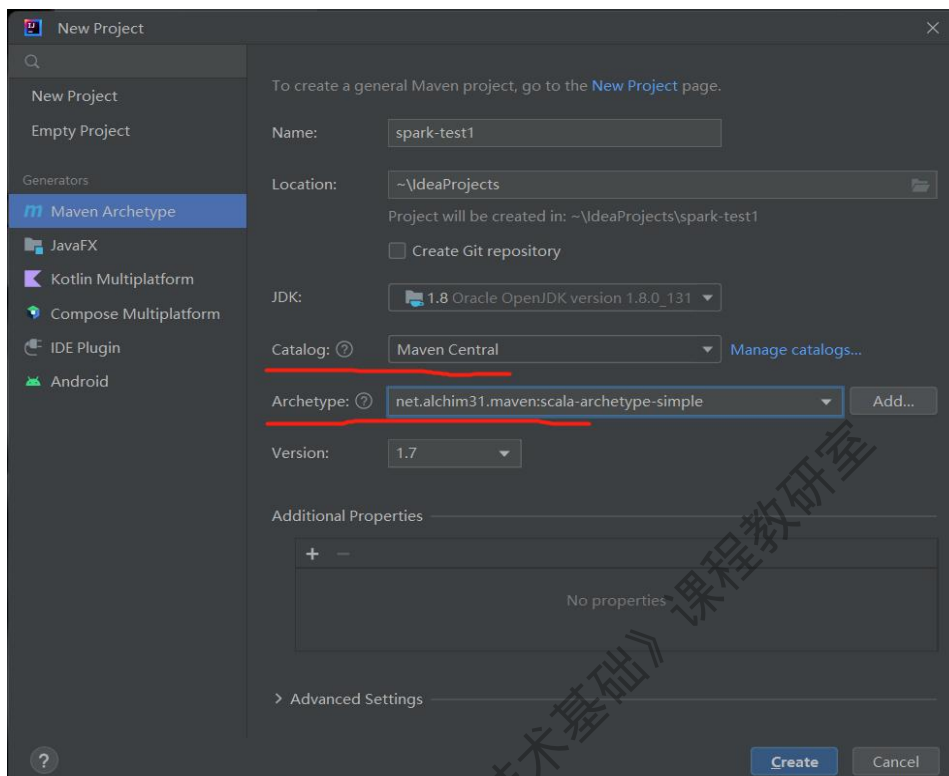


图 4.8-11 实验截图

进入如下界面表示工程创建成功：

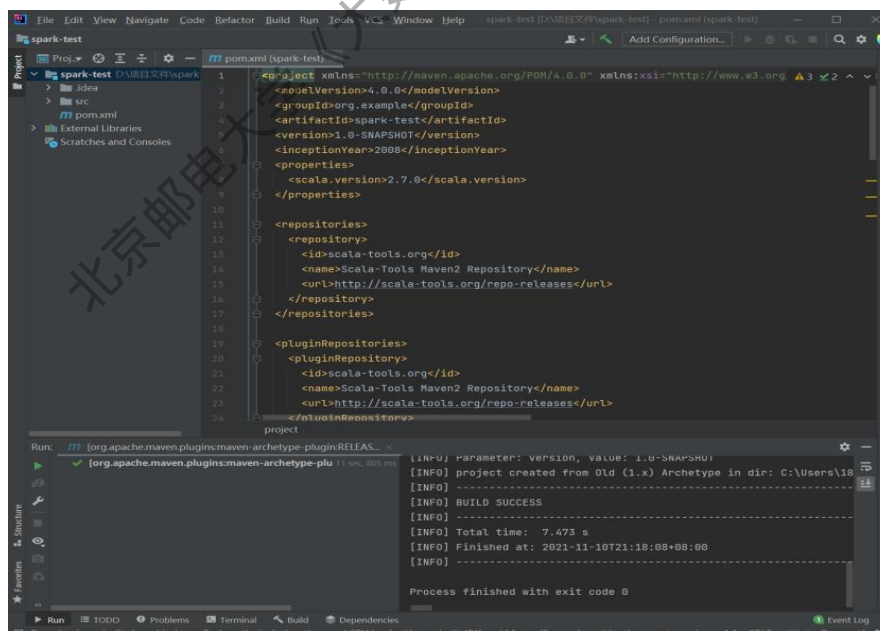
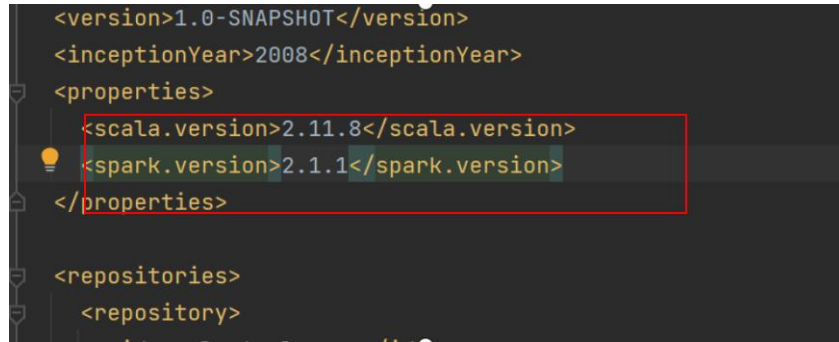


图 4.8-12 实验截图

## 步骤 2：依赖设置：

1) 在 pom.xml 文件中找到 properties 配置项，修改 scala 版本号（此处对应 scala 安装版本），并添加 spark 版本号（此处对应 spark 安装版本）；





```

<version>1.0-SNAPSHOT</version>
<inceptionYear>2008</inceptionYear>
<properties>
  <scala.version>2.11.8</scala.version>
  <spark.version>2.1.1</spark.version>
</properties>

<repositories>
  <repository>

```

图 4.8-13 实验截图

2) 找到 dependency 配置项，添加如下图标红部分的配置，分别是 scala 依赖和 spark 依赖，\\${scala.version}表示上述配置的 scala.version 变量；



```

<dependencies>
  <dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.11</artifactId>
    <version>${spark.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>${spark.version}</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>

```

图 4.8-14 实验截图

3) 一般修改 pom.xml 文件后，会提示 enable auto-import，点击即可，如果没有提示，则可以点击工程名，依次选择 Maven—>Reimport，即可根据 pom.xml 文件导入依赖包；

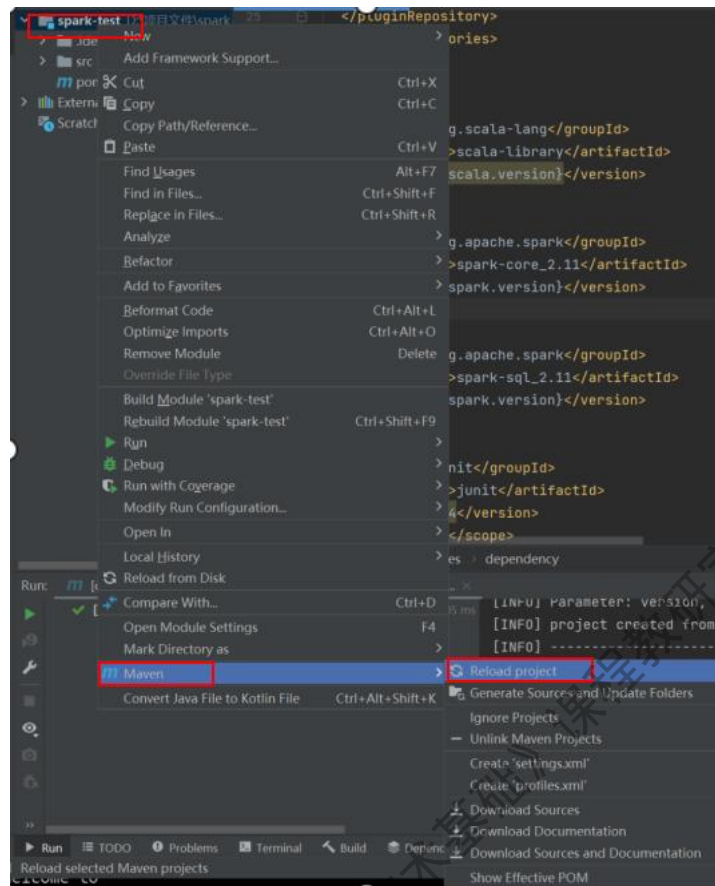


图 4.8-15 实验截图

### 步骤 3：设置语言环境：

- 1) 设置语言环境 language level，点击菜单栏中的 file，选择 Project Structure；

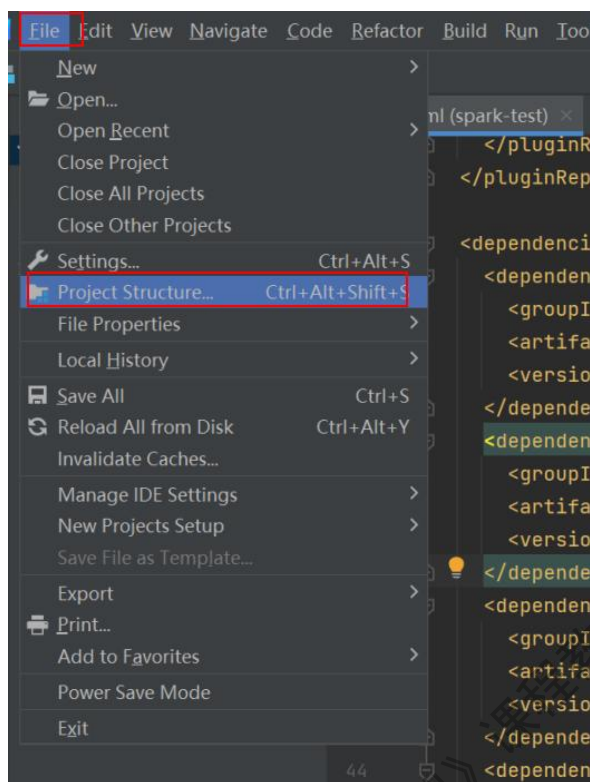


图 4.8-16 实验截图

弹出如下对话框，选择 Modules，选择 Language level 为 8，然后点击 Apply，点击 OK；

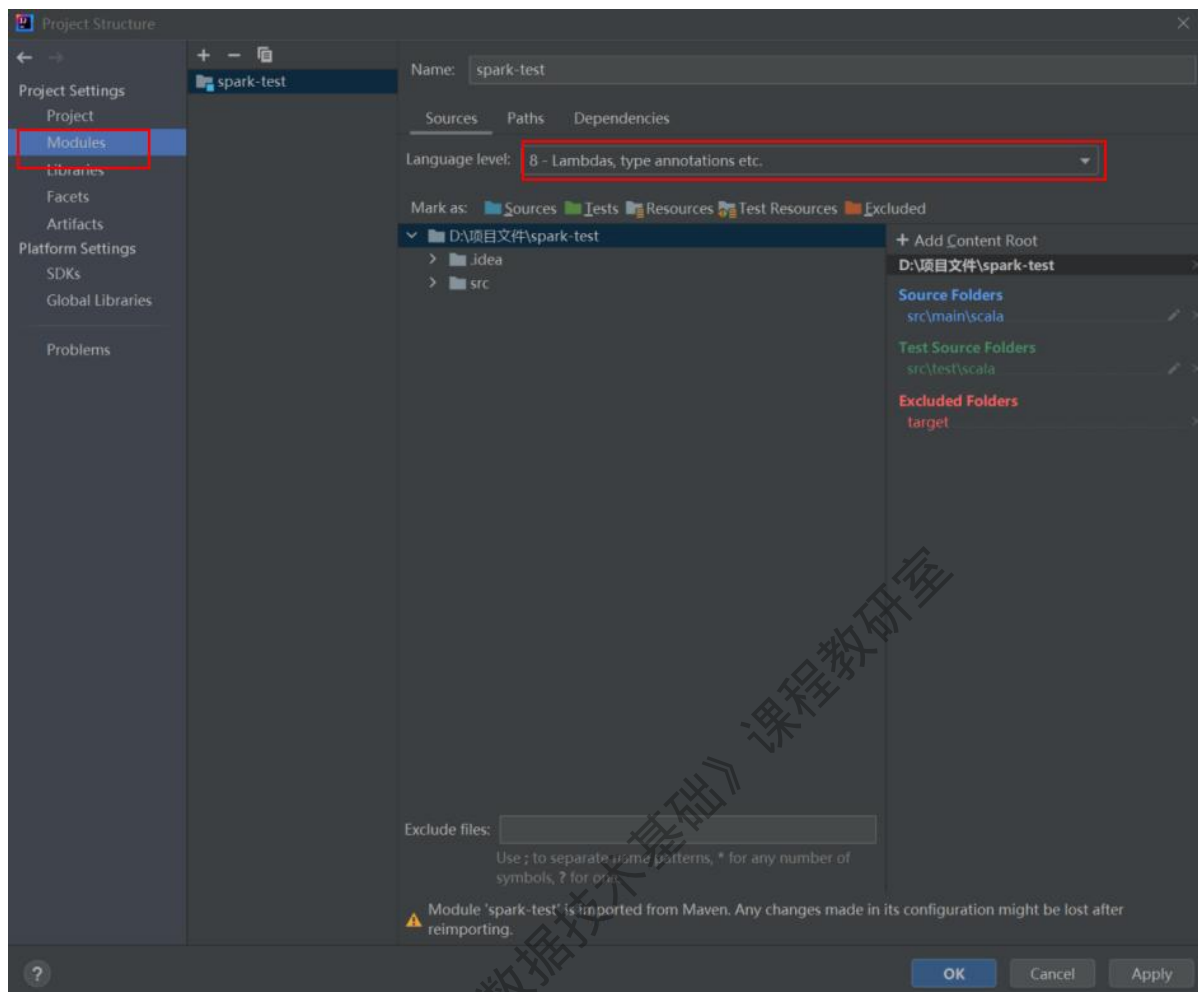


图 4.8-17 实验截图

**步骤 4：**设置 java Compiler 环境：

1) 点击菜单栏中的 file，选择 Setting；

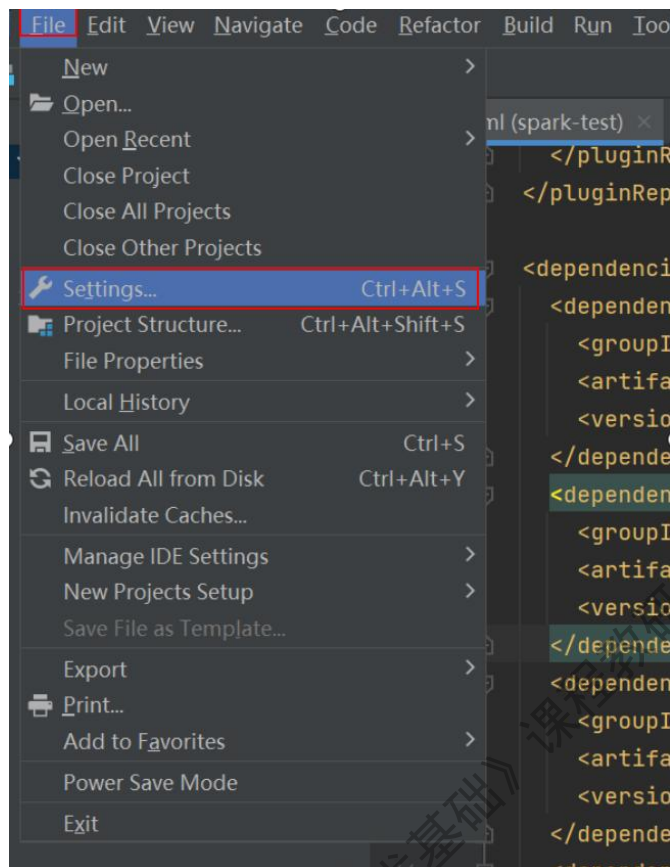


图 4.8-18 实验截图

2) 弹出如下对话框，依次选择 Build，Execution—>Compiler—>Java Compiler，设置图中的 Project bytecode version 为 1.8，设置图中的 Target bytecode version 为 1.8，然后依次点击 Apply 和 OK；

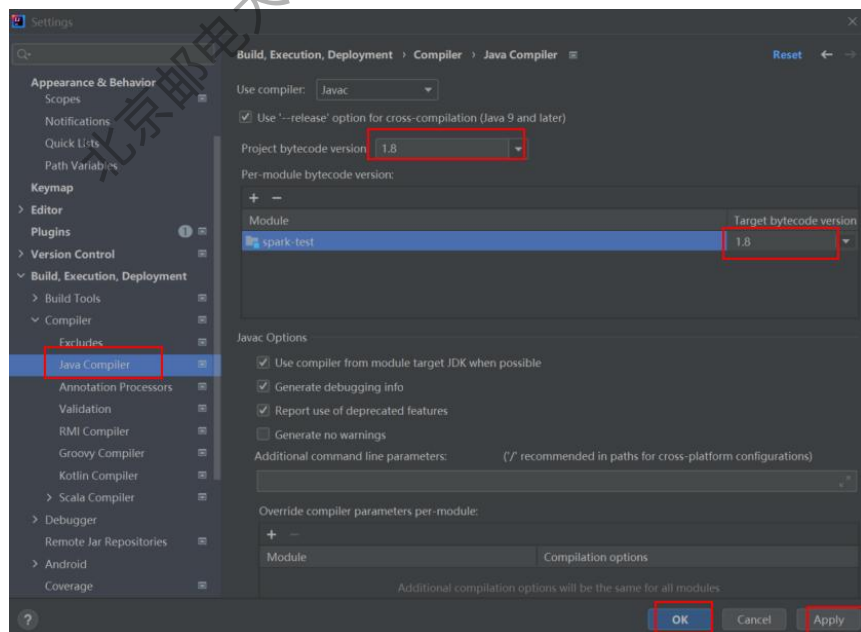


图 4.8-19 实验截图



**步骤 5：文件配置：**

1) 如下图删除测试环境 test 中的测试类，即AppTest和MySpec两个文件；

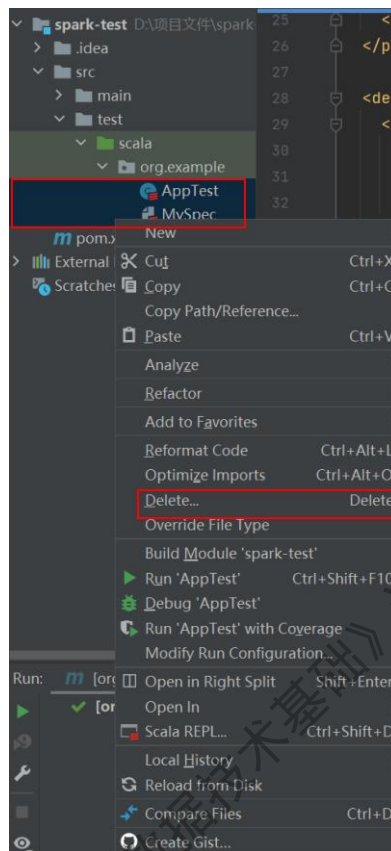


图 4.8-20 实验截图

2) 如下图删除，main 文件夹中，包名下的 App 文件；

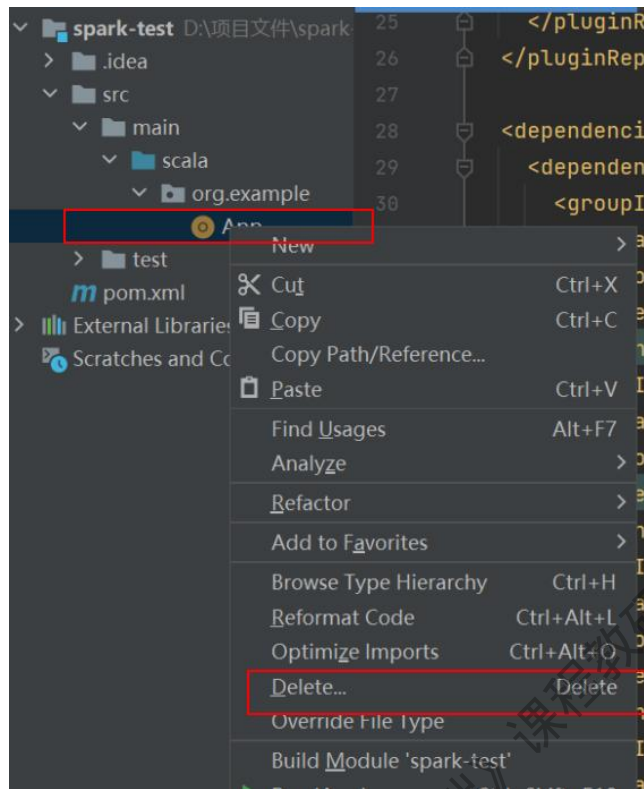


图 4.8-21 实验截图

**步骤 6： 程序编写：**

1) 如下图依次打开 src—>main—>scala, 在 org.example 上点击右键, 创建 Scala Class;

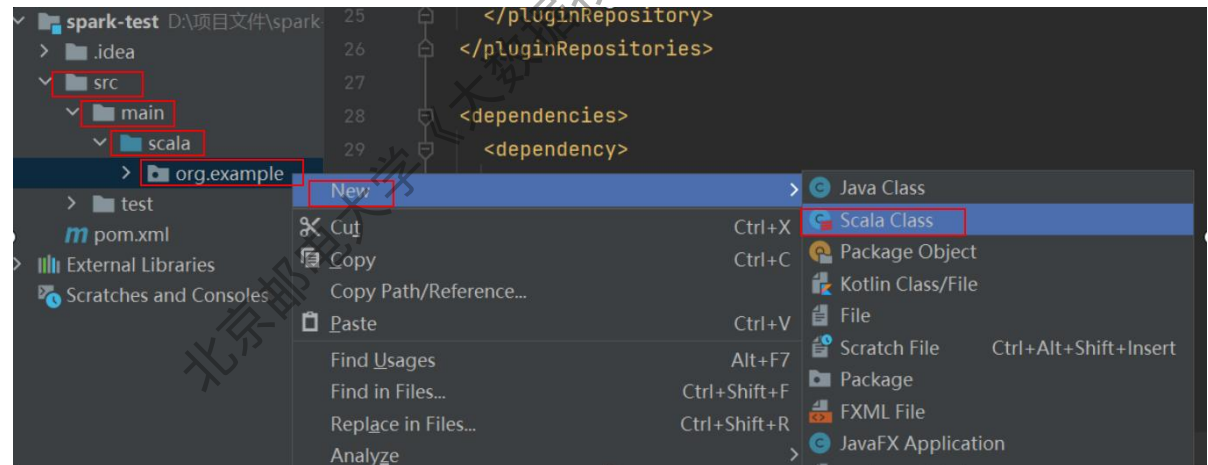


图 4.8-22 实验截图

2) 弹出如下对话框, 输入类名 ScalaWordCount, 回车

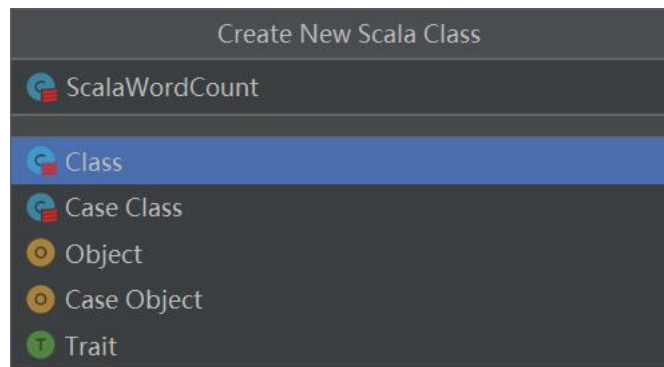


图 4.8-23 实验截图

- 3) 在类 ScalaWordCount 中新建伴生对象 object ScalaWordCount;

```
class ScalaWordCount {  
}  
object ScalaWordCount{
```

图 4.8-24 实验截图

- 4) 在伴生对象 object ScalaWordCount 中创建 main 方法;

```
object ScalaWordCount{  
  def main(args:Array[String]):Unit={
```

图 4.8-25 实验截图

- 5) 在 main 方法中创建列表 List 对象并赋值给常量 list, 列表中包含 4 个元素, 分别是: “hello hi hi spark”, “hello spark hello hi sparksql”, “hello hi hi sparkstreaming”, “hello hi sparkgraphx”;

```
val list=List("hello hi hi spark",  
  "hello spark hello hi sparksql",  
  "hello hi hi sparkstreaming",  
  "hello hi sparkgraphx")
```

图 4.8-26 实验截图

- 6) 创建 SparkConf 对象, 对 Spark 运行属性进行配置, 调用该对象的 setAppName 方法设置 Spark 程序的名称为“word-count”, 调用 setMaster 方法设置 Spark 程序运行模式, 一般分为两种: 本地模式和 yarn 模式, 这里我们采用 yarn 模式, 参数为“yarn”, 属性设置完成后赋值给常量 sparkConf;

- 7) 创建 SparkContext, 参数为 sparkconf, 赋值给常量 sc, 该对象是 Spark 程序的入口;

```
val sparkConf=new SparkConf().setAppName("word-count").setMaster("yarn")  
val sc=new SparkContext(sparkConf)
```

图 4.8-27 实验截图

- 8) 调用 SparkContext 对象 sc 的方法 parallelize, 参数为列表对象 list, 该方法使用现成的 scala 集合生成 RDD lines, 类型为 String(RDD 为 Spark 计算中的最小数据单元), 该 RDD

存储元素是字符串语句;

```
val lines:RDD[String]=sc.parallelize(list)
```

图 4.8-28 实验截图

9) 调用 RDD 对象 lines 的 flatMap 方法按照空格切分 RDD 中的字符串元素, 并存入新的 RDD 对象 words 中, 参数类型为 String, 该 RDD 存储的元素是每一个单词;

```
val lines:RDD[String]=sc.parallelize(list)
val words:RDD[String]=lines.flatMap((line:String)=>{line.split( regex = " ")})
```

图 4.8-29 实验截图

10) 调用 RDD 对象 words 的 map 方法, 将 RDD 中的每一个单词转换为 kv对, key 是 String 类型的单词, value 是 Int 类型的 1, 并赋值给新的 RDD 对象 wordAndOne, 参数为 (String, Int) 类型键值对;

```
val wordAndOne:RDD[(String,Int)]=words.map((word:String)=>{(word,1)})
```

图 4.8-30 实验截图

11) 调用 RDD 对象 wordAndOne 的 reduceByKey 方法, 传入的参数为两个Int 类型变量, 该方法将 RDD 中的元素按照 Key 进行分组, 将同一组中的 value 值进行聚合操作, 得到 valueRet, 最终返回 (key, valueRet) 键值对, 并赋值给新的 RDD 对象 wordAndNum, 参数为 (String, Int) 类型键值对;

```
val ret=wordAndNum.sortBy(kv=>kv._2, ascending = false)
```

图 4.8-31 实验截图

12) 调用 RDD 对象 wordAndNum 的 sortBy 方法, 第一个参数为 kv 对中的value, 即单词出现次数, 第二个参数为 boolean 类型, true 表示升序, false 表示降序;

```
val ret=wordAndNum.sortBy(kv=>kv._2, ascending = false)
```

图 4.8-32 实验截图

13) 调用 ret 对象的 collect 方法, 获取集合中的元素, 再调用 mkString 方法, 参数为“,”, 将集合中的元素用逗号连接成字符串, 调用 println 方法打印输出在控制台;

```
print(ret.collect().mkString(","))
```

图 4.8-33 实验截图

14) 调用 ret 对象的 saveAsTextFile, 该方法的参数为运行时指定的参数, 此方法的用处是将 Spark 程序运行的结果保存到指定路径, 一般是把结果保存到 HDFS 中, 所以这里的参数定义为: hdfs://yty-2022140804-0001:8020/spark\_test, HDFS 根目录中不存在 spark\_test 此目录, spark 程序会自动创建该目录; 调用SparkContext 对象 sc 的 stop 方法, 释放 spark 程序所占用的资源;

```
ret.saveAsTextFile( path = "hdfs://yty-2022140804-0001:9000/spark_test")
sc.stop
```

图 4.8-34 实验截图

15) 完整程序如下:

```

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

class ScalaWordCount{
|
}

object ScalaWordCount{
  def main(args:Array[String]):Unit={
    val list=List("hello hi hi spark",
      "hello spark hello hi sparksql",
      "hello hi hi sparkstreaming",
      "hello hi sparkgraphx")

    val sparkConf = new SparkConf().setAppName("word-count").setMaster("yarn")
    val sc=new SparkContext(sparkConf)
    val lines:RDD[String]=sc.parallelize(list)
    val words:RDD[String]=lines.flatMap((line:String)=>{line.split(" ")})
    val wordAndOne:RDD[(String,Int)]=words.map((word:String)=>{(word,1)})
    val wordAndNum:RDD[(String,Int)]=wordAndOne.reduceByKey((count1:Int,count2:Int)=>{count1+count2})
    val ret=wordAndNum.sortBy(kv=>kv._2, ascending = false)
    print(ret.collect().mkString(", "))
    ret.saveAsTextFile( path = "hdfs://yty-2022140804-0001:8020/spark-test")
    sc.stop
  }
}

```

图 4.8-35 实验截图

#### 4. 程序打包及运行

步骤 1: 打开 File->Project Structure:

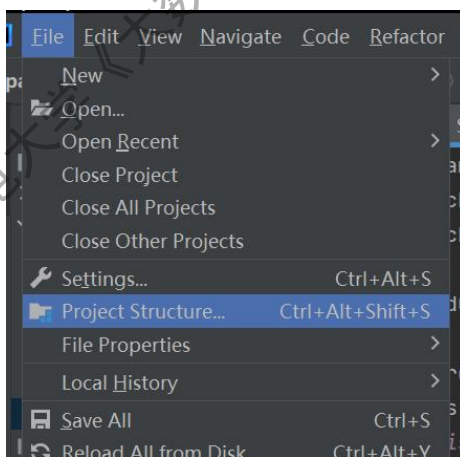


图 4.8-36 实验截图

步骤 2: Project Settings 栏下的 Artifacts, 点击“+”,选择 JAR->From modules with dependencies....:



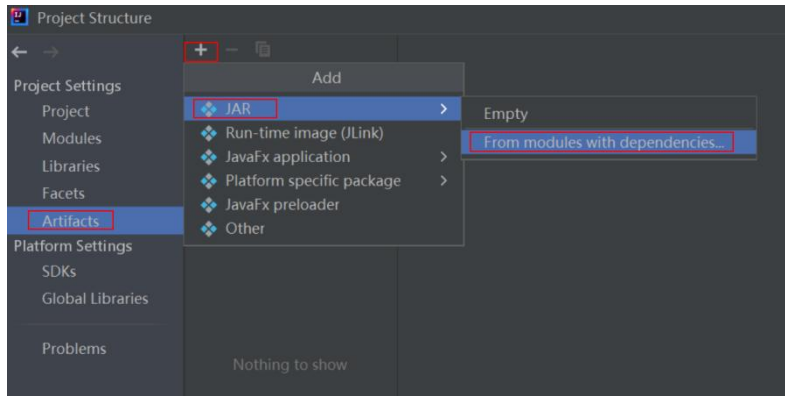


图 4.8-37 实验截图

步骤 3：填选主类名称：

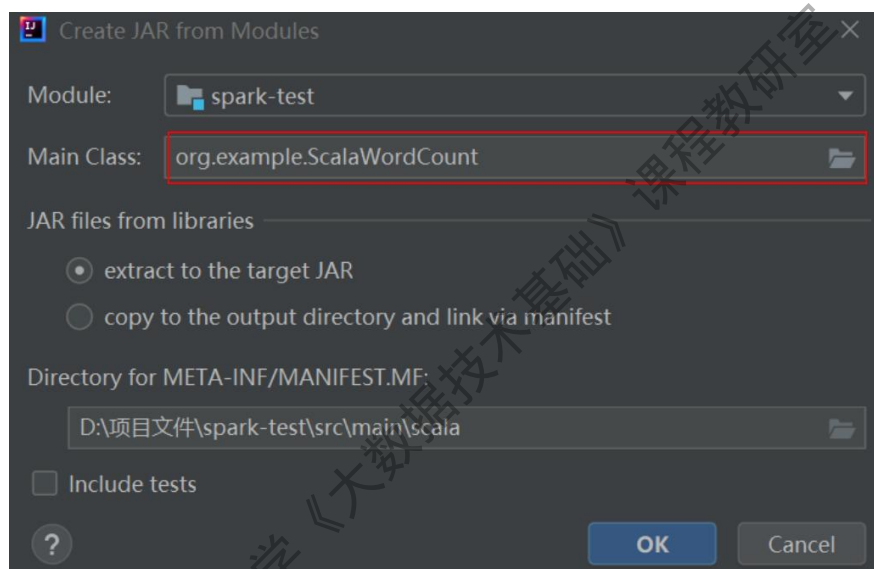


图 4.8-38 实验截图

步骤 4：选择 Build->Artifacts:

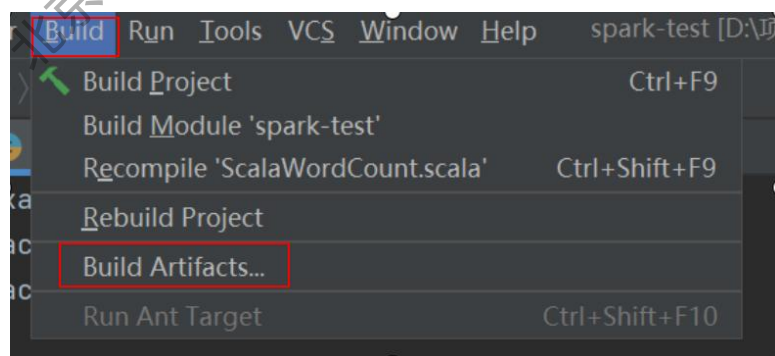


图 4.8-39 实验截图

步骤 5：选择 Build:

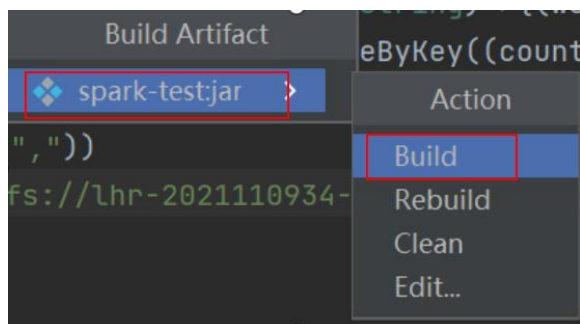


图 4.8-40

建立完成：

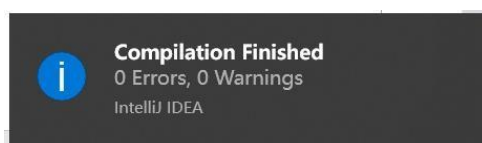


图 4.8-41

步骤 6：使用压缩软件打开生成的 jar 包：



图 4.8-42

步骤 7：找到 META-INF 目录

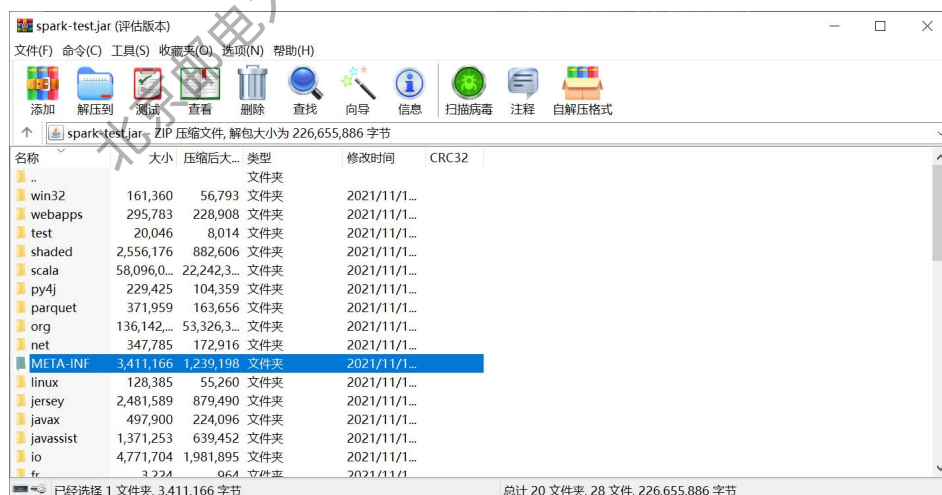


图 4.8-43

步骤 8：删除 MANIFEST.MF 文件

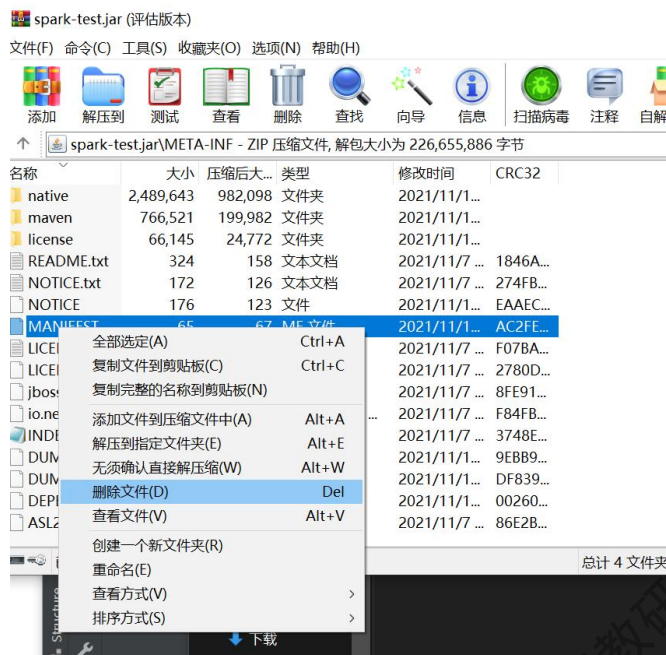


图 4.8-44

步骤 9：使用 Xftp 上传处理后的 jar 包到服务器：

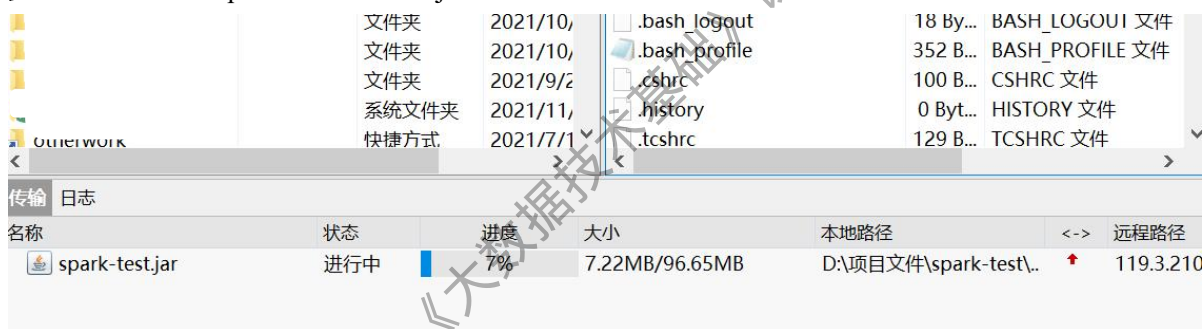


图 4.8-45

步骤 10：使用 spark-submit 命令，在 Hadoop 上运行程序：

```
spark-submit --class org.example.ScalaWordCount --master yarn -- num-executors 3 --driver-memory 1g --executor-memory 1g --executor-cores 1 spark-test.jar
```

得到如下结果：

```
22/10/31 12:55:54 INFO scheduler.TaskSetManager: Starting task 2.0 in stage 4.0 (TID 11, yty-2022140804-0002, executor 2, partition 2, NODE_LOCAL, 5818 bytes)
22/10/31 12:55:54 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0 (TID 9) in 55 ms on yty-2022140804-0002 (executor 2) (1/3)
22/10/31 12:55:54 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 4.0 (TID 10) in 62 ms on yty-2022140804-0003 (executor 1) (2/3)
22/10/31 12:55:54 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 4.0 (TID 11) in 34 ms on yty-2022140804-0002 (executor 2) (3/3)
22/10/31 12:55:54 INFO cluster.YarnScheduler: Removed TaskSet 4.0, whose tasks have all completed, from pool
22/10/31 12:55:54 INFO scheduler.DAGScheduler: ResultStage 4 (collect at ScalaWordCount.scala:22) finished in 0.090 s
22/10/31 12:55:54 INFO scheduler.DAGScheduler: Job 1 finished: collect at ScalaWordCount.scala:22, took 0.232221 s
(hi,6), (hello,5), (spark,2), (sparksql,1), (sparkstreaming,1), (sparkgraphx,1)22/10/31 12:55:55 INFO storage.BlockManagerInfo: Removed broadcast_3_piece0 on 192.168.0.25:41257 in memory (size: 2025.0 B, free: 434.4 MB)
```

图 4.8-46 实验截图

步骤 11：在 hdfs 上查看程序的输出：

```
[root@yty-2022140804-0001 ~]# hadoop fs -ls /
22/10/31 13:56:09 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
Found 3 items
drwxr-xr-x - root supergroup          0 2022-10-31 13:55 /spark-test
drwx----- - root supergroup          0 2022-10-31 11:05 /tmp
drwxr-xr-x - root supergroup          0 2022-10-31 10:43 /user
```

图 4.8-47 实验截图

```
[root@yty-2022140804-0001 ~]# hadoop fs -cat /spark-test/part-00000
22/10/31 13:57:13 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
(hi,6)
(hello,5)
[root@yty-2022140804-0001 ~]# hadoop fs -cat /spark-test/part-00001
22/10/31 13:57:26 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
(spark,2)
[root@yty-2022140804-0001 ~]# hadoop fs -cat /spark-test/part-00003
22/10/31 13:57:32 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
cat: '/spark-test/part-00003': No such file or directory
[root@yty-2022140804-0001 ~]# hadoop fs -cat /spark-test/part-00002
22/10/31 13:57:49 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
(sparksql,1)
(sparkstreaming,1)
(sparkgraphx,1)
```

图 4.8-48 实验截图

### 1.1.3.2. 使用 RDD 编写独立应用程序实现数据去重

对于两个输入文件A和B，编写Spark独立应用程序，对两个文件进行合并，并剔除其中重复的内容，得到一个新文件C。下面是输入文件和输出文件的一个样例，供参考。（要求运行截图及代码）

输入文件A的样例如下：

```
20170101    x
20170102    y
20170103    x
20170104    y
20170105    z
20170106    z
```

输入文件B的样例如下：

```
20170101    y
20170102    y
20170103    x
20170104    z
20170105    y
```

根据输入的文件A和B合并得到的输出文件C的样例如下：

```
20170101    x
20170101    y
```



```
20170102 y
20170103 x
20170104 y
20170104 z
20170105 y
20170105 z
20170106 z
```

可参考方法:

- 从文件中读取数据创建 RDD

```
val text = sc.textFile("A.txt")
```

```
val A = sc.textFile( path = "A.txt")
```

图 4.8-49

可以把txt文件放到与jar包同一路径下,避免出现找不到文件的问题。首先将txt文件通过put命令上传到hdfs中。上传完成后可以用ls命令查看到文件已经上传到hdfs。

```
[root@yty-2022140804-0001 ~]# hadoop fs -put A.txt
22/11/01 10:49:06 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
[root@yty-2022140804-0001 ~]# hadoop fs -put B.txt
22/11/01 10:49:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
[root@yty-2022140804-0001 ~]# hadoop fs -ls
22/11/01 10:49:23 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
Found 3 items
drwxr-xr-x - root supergroup          0 2022-10-31 13:55 .sparkStaging
-rw-r--r--  3 root supergroup        70 2022-11-01 10:49 A.txt
-rw-r--r--  3 root supergroup        58 2022-11-01 10:49 B.txt
[root@yty-2022140804-0001 ~]#
```

图 4.8-50

- 把 RDD 写入到文本文件中

```
text.saveAsTextFile("writeback")
```

此处的writeback应该为hdfs中的一个空文件夹,运行后会在文件夹中生成输出文件,用cat显示其中内容即可。

```
C.saveAsTextFile( path = "hdfs://yty-2022140804-0001:8020/C.txt")
```

图 4.8-51

- join()对于给定的两个输入数据集(K,V1)和(K,V2),只有在两个数据集中都存在的 key 才会被输出,最终得到一个(K,(V1,V2))类型的数据集。

```
RDD1.join(RDD2)
```

- union()合并变换将两个 RDD 合并为一个新的 RDD,重复的记录不会被剔除。

```
RDD1.union(RDD2)
```

- distinct()可去除RDD中重复的数据。

```
RDD1.distinct()
```

本次实验可以在上一次实验构建的IDEA项目中进行代码的编写。构建方法参考4.1.3和



4.1.4节中的内容。构建完成后，将打好的jar包通过工具上传到服务器，用spark-submit命令运行（将jar包名称替换一下即可）。

运行完成后，用hadoop fs -ls /输出文件夹命令查看hdfs中的输出结果。

```
[root@yty-2022140804-0001 ~]# hadoop fs -ls /
22/11/01 11:02:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 4 items
drwxr-xr-x - root supergroup          0 2022-11-01 11:01 /C.txt
drwxr-xr-x - root supergroup          0 2022-10-31 13:55 /spark-test
drwx----- - root supergroup          0 2022-10-31 11:05 /tmp
drwxr-xr-x - root supergroup          0 2022-10-31 10:43 /user
```

图 4.8-52

用cat命令分别查看输出文件中的内容。

```
[root@yty-2022140804-0001 ~]# hadoop fs -cat /C.txt/part-00000
22/11/01 11:03:50 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
20170101      x
20170101      y
20170102      y
[root@yty-2022140804-0001 ~]# hadoop fs -cat ^C
[root@yty-2022140804-0001 ~]# hadoop fs -cat /C.txt/part-00001
22/11/01 11:04:12 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
20170103      x
20170104      y
[root@yty-2022140804-0001 ~]# hadoop fs -cat /C.txt/part-00002
22/11/01 11:04:18 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
20170104      z
20170105      y
[root@yty-2022140804-0001 ~]# hadoop fs -cat /C.txt/part-00003
22/11/01 11:04:27 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
20170105      z
20170106      z
[root@yty-2022140804-0001 ~]#
```

图 4.8-53 实验截图

可以看到两个文件被成功合并。

### 1.1.3.3. 使用 Spark SQL 读写数据库

#### 步骤1：下载并安装MySQL

下载：wget <https://dev.mysql.com/get/mysql80-community-release-el7-3.noarch.rpm>

安装源：yum localinstall mysql80-community-release-el7-3.noarch.rpm

安装依赖：rpm --import <https://repo.mysql.com/RPM-GPG-KEY-mysql-2022>

安装：yum -y install mysql-community-server

#### 步骤2：MySQL数据库设置

首先启动MySQL：systemctl start mysqld.service

查看MySQL运行状态：systemctl status mysqld.service

```

[root@yty-2022140804-0001 ~]# systemctl start mysqld.service
[root@yty-2022140804-0001 ~]# systemctl status mysqld.service
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2022-11-01 11:14:12 CST; 11s ago
     Docs: man:mysqld(8)
           http://dev.mysql.com/doc/refman/en/using-systemd.html
   Process: 4111 ExecStartPre=/usr/bin/mysqld_pre_systemd (code=exited, status=0/SUCCESS)
   Main PID: 4187 (mysqld)
    Status: "Server is operational"
   CGroup: /system.slice/mysqld.service
           └─4187 /usr/sbin/mysqld

Nov 01 11:14:05 yty-2022140804-0001 systemd[1]: Starting MySQL Server...
Nov 01 11:14:12 yty-2022140804-0001 systemd[1]: Started MySQL Server.

```

图 4.8-54 实验截图

此时MySQL已经开始正常运行,不过要想进入MySQL还得先找出此时root用户的密码,通过如下命令可以在日志文件中找出密码:

grep "password" /var/log/mysqld.log

```

[root@yty-2022140804-0001 ~]# grep "password" /var/log/mysqld.log
2022-11-01T03:14:08.317124Z 6 [Note] [MY-010454] [Server] A temporary password is generated for root@localhost: hZpkUAx.k1p8
[root@yty-2022140804-0001 ~]#

```

图 4.8-55 实验截图

如下命令进入数据库:

mysql -uroot -p

输入初始密码(是上面图片最后面的 hZpkUAx.k1p8),此时不能做任何事情,因为MySQL默认必须修改密码之后才能操作数据库:

ALTER USER 'root'@'localhost' IDENTIFIED BY 'new password';

其中'new password'替换成你要设置的密码,注意:密码设置必须要大小写字母数字和特殊符号(,/,等),不然不能配置成功。

**步骤3:** 通过JDBC连接数据库

在MySQL Shell环境中,先输入create database spark来创建spark数据库,再输入下面SQL语句完成数据库和表的创建:

```

mysql> use spark;
Database changed
mysql> create table student(id int(4), name char(20), gender char(4), age int(4));
Query OK, 0 rows affected, 2 warnings (0.02 sec)

mysql> insert into student values(1, 'Li', 'F', 23);
Query OK, 1 row affected (0.00 sec)

mysql> insert into student values(2, 'Wang', 'M', 24);
Query OK, 1 row affected (0.00 sec)

mysql> select * from student;
+-----+-----+-----+-----+
| id   | name | gender | age |
+-----+-----+-----+-----+
| 1    | Li   | F      | 23  |
| 2    | Wang | M      | 24  |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

图 4.8-56 实验截图

要想顺利连接MySQL数据库，还需要使用MySQL数据库驱动程序。请到MySQL官网下载MySQL的JDBC驱动程序，把该驱动程序解压缩到Spark的安装目录下：

```
wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.24.tar.gz
tar -xvzf mysql-connector-java-8.0.24.tar.gz
mv mysql-connector-java-8.0.24/mysql-connector-java-8.0.24.jar
/root/spark-2.1.1-bin-hadoop2.7/jars/
```

启动一个spark shell。启动spark shell时，必须指定MySQL连接驱动jar包，命令如下：

```
spark-shell --jars /root/spark-2.1.1-bin-hadoop2.7/jars/mysql-connector-java-8.0.24.jar
--driver-class-path /root/spark-2.1.1-bin-hadoop2.7/jars/mysql-connector-java-8.0.24.jar
spark.read.format("jdbc")操作可以实现对MySQL数据库的读取。执行以下命令连接数据库，读取数据并显示：（要求贴出命令和结果的截图）
```

```
scala> val jdbcDF = spark.read.format("jdbc").
| option("url", "jdbc:mysql://localhost:3306/spark").
| option("driver", "com.mysql.jdbc.Driver").
| option("dbtable", "student").
| option("user", "root").
| option("password", "Yao2503611945!").
| load()
jdbcDF: org.apache.spark.sql.DataFrame = [id: int, name: string ... 2 more fields]

scala> jdbcDF.show()
22/11/01 11:53:48 WARN util.SizeEstimator: Failed to check whether UseCompressedOops is set;
assuming yes
+---+-----+-----+---+
| id|name|gender|age|
+---+-----+-----+---+
| 1| Li|      F| 23|
| 2|Wang|      M| 24|
+---+-----+-----+---+
```

图 4.8-57 实验截图

在通过JDBC连接MySQL数据库时，需要通过option()方法设置相关的连接参数，下表给出了各个参数的含义。

参数名称	参数的值	含义
url	jdbc:mysql://localhost:3306/spark	数据库的连接地址
driver	com.mysql.jdbc.Driver	数据库的JDBC驱动
dbtable	student	所要访问的表
user	root	用户名
password		用户密码

#### 步骤4：向MySQL数据库写入数据

在idea中编写向数据库中插入数据的代码：



```

object InsertStudent{
  def main(args: Array[String]): Unit = {
    val sparkConf = new SparkConf().setAppName("insert-student").setMaster("local")
    val sc = new SparkContext(sparkConf)
    //学生信息RDD
    val studentRDD = sc.parallelize(Array("3 Zhang M 26", "4 Liu M 27")).map(_.split(" "))
    //模式信息
    val schema = StructType(List(
      StructField("id", IntegerType, true),
      StructField("name", StringType, true),
      StructField("gender", StringType, true),
      StructField("age", IntegerType, true)))
    // Row对象
    val rowRDD = studentRDD.map(p => Row(p(0).toInt, p(1).trim, p(2).trim,
      p(3).toInt))
    //建立起Row对象和模式之间的关系
    val studentDF = new SQLContext(sc).createDataFrame(rowRDD, schema)
    // JDBC连接参数
    val prop = new Properties()
    prop.put("user", "root")
    prop.put("password", "Yao2503611945!")
    prop.put("driver", "com.mysql.jdbc.Driver")
    //连接数据库,append
    studentDF.write.mode("append")
      .jdbc("jdbc:mysql://localhost:3306/spark", "spark.student", prop)
  }
}

```

图 4.8-58

打包并删除jar包中的MANIFEST.MF文件，将jar包上传到服务器，使用spark-submit命令运行程序。

```

[root@yty-2022140804-0001 ~]# spark-submit --class org.example.InsertStudent --master yarn -
-num-executors 3 --driver-memory 1g --executor-memory 1g --executor-cores 1 spark-test2.jar
22/11/01 12:07:43 INFO spark.SparkContext: Running Spark version 2.1.1
22/11/01 12:07:43 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
22/11/01 12:07:43 INFO spark.SecurityManager: Changing view acls to: root
22/11/01 12:07:43 INFO spark.SecurityManager: Changing modify acls to: root
22/11/01 12:07:43 INFO spark.SecurityManager: Changing view acls groups to:
22/11/01 12:07:43 INFO spark.SecurityManager: Changing modify acls groups to:
22/11/01 12:07:43 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui a
cls disabled; users with view permissions: Set(root); groups with view permissions: Set();
users with modify permissions: Set(root); groups with modify permissions: Set()
22/11/01 12:07:43 INFO util.Utils: Successfully started service 'sparkDriver' on port 36495.
22/11/01 12:07:43 INFO spark.SparkEnv: Registering MapOutputTracker

```

图 4.8-59

运行之后，可以到MySQL shell环境中使用sql语句查询student表，可以发现新增加的两条记录。（可以在命令行里打印一些文字表示运行成功，要求截图中包含代码和运行结果）

```

mysql> select * from student;
+----+-----+-----+-----+
| id  | name  | gender | age  |
+----+-----+-----+-----+
| 1   | Li    | F      | 23   |
| 2   | Wang  | M      | 24   |
| 3   | Zhang | M      | 26   |
| 4   | Liu   | M      | 27   |
+----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>

```

图 4.8-60 实验截图

### 1.1.4. 实验结果与评分标准

实验结束后应得到：1 个安装好 Hadoop 和 Spark 集群，1 个 Spar 程序 jar 包。  
完成 Spark RDD 和 Spark SQL 数据处理实践。

实验中应该注意的重点步骤包含：

1. Hadoop 集群测试结果（截图）；
2. Spark 集群搭建完成的测试结果（截图）；
3. Scala 单词计数实验结果（截图）；
4. RDD 编程结果（截图）；
5. Spark sql 读写数据库结果（截图）；
6. 整体实验报告撰写（截图）。

实验结果截图需要按要求带有ifconfig的ip信息。

北京邮电大学《大数据技术基础》课程教研室