大数据实践 4：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 1 台服务器，然后使用 docker 的方式搭建 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 3.3.6
4. Spark 版本：Apache Spark 3.5.1
5. JDK 版本：1.8.0_292-b10
6. Scala 版本：scala2.13.8
7. IDEA 版本：IntelliJ IDEA Community Edition 2023.1
8. MySQL 版本：8.0

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，Hadoop 版本为 Apache Hadoop 3.3.6。集群利用 docker 的方式在一台华为云服务器上进行构建。整个集群的构建需要四个容器，主节点为 master，从节点分别为 slave1、slave2、slave3。

步骤一：在构建好 Hadoop 平台的主节点开启集群：start-all.sh，并进入主节点和从节点的 docker 容器中执行 jps 命令，确保 Hadoop 集群已经正确启动。

```
[root@zj-2023140696 ~]# docker exec -it master bash
root@master:~# jps
4083 Jps
1571 DataNode
1191 SecondaryNameNode
440 ResourceManager
952 NameNode
```

```
root@master:~# ssh slave1
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 4.18.0-80.7.2.el7.aarch64 aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Fri May 10 07:20:30 2024 from 172.17.0.2
root@slave1:~# jps
231 NodeManager
107 DataNode
1596 Jps
```

步骤二：在主节点的 docker 中测试 Hadoop 集群的可用性。

测试的方法有多种：

（1） 试着在 HDFS 上执行一些基本命令来检查文件系统是否响应。例如，列出根目录下的文件：

```
hadoop fs -ls
```

```
root@master:~# hadoop fs -ls /
2024-05-10 07:27:33,849 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
drwxr-xr-x   - root supergroup          0 2024-05-09 23:38 /spark-test
drwxr-xr-x   - root supergroup          0 2024-05-09 22:47 /user
```

（2） 使用如下命令检查集群的节点状态：

```
yarn node -list
```

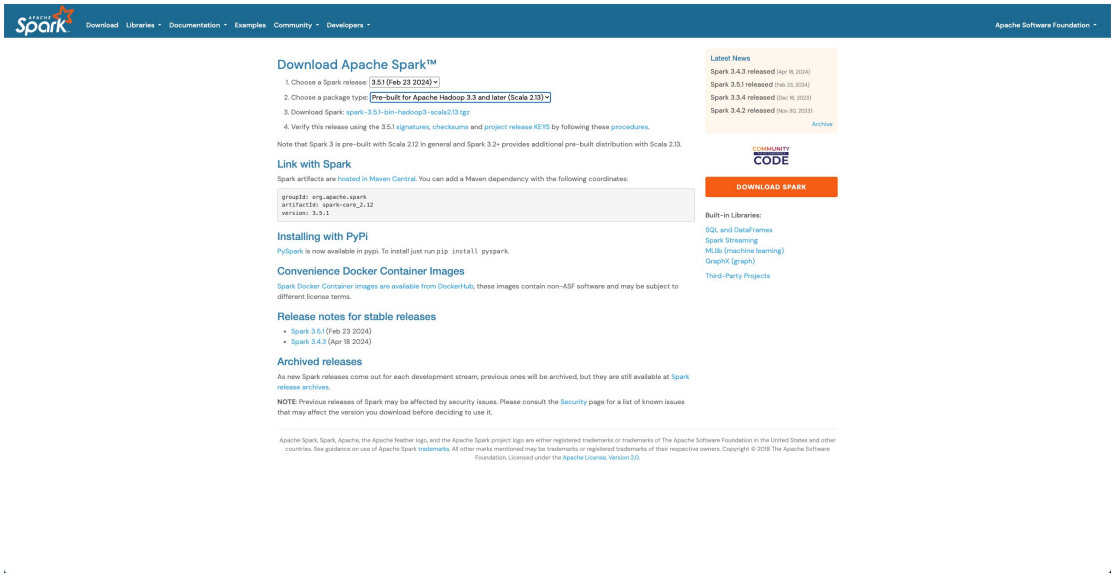
```
root@master:~# yarn node -list
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
2024-05-10 07:28:01,483 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2024-05-10 07:28:01,542 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at master/172.17.0.2:8032
Total Nodes:3
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
    slave2:45337                RUNNING       slave2:8042                              0
    slave3:42177                RUNNING       slave3:8042                              0
    slave1:35533                RUNNING       slave1:8042                              0
```

确保 Hadoop 集群可用。

2、 Spark 集群搭建（docker 的方式）

步骤一：在 master 上解压 spark 压缩包

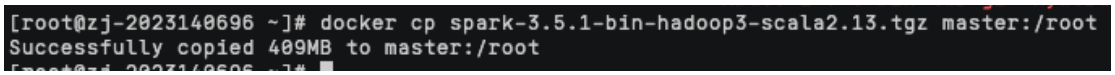
（1） 在官网下载 spark，版本的选择如下图，注意选择 scala 版本为 2.13 的 spark，spark 版本为 3.5.1。



（2） 使用 sftp 上传 spark 压缩包到服务器中，在服务器中能够找到该文件。

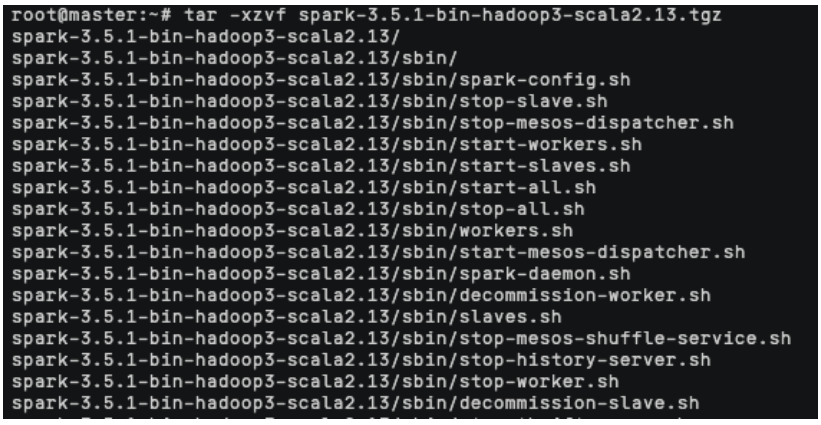
（3） 将服务器中的文件复制到主节点的 docker 容器中

```
docker cp spark-3.5.1-bin-hadoop3-scala2.13.tgz master:/root
```



（4） 进入 docker 中解压 spark 压缩包

```
tar -xzvf spark-3.5.1-bin-hadoop3-scala2.13.tgz
```



（5） 将解压后的文件改名为 spark

```
mv spark-3.5.1-bin-hadoop3-scala2.13 spark
```

步骤二：配置环境变量

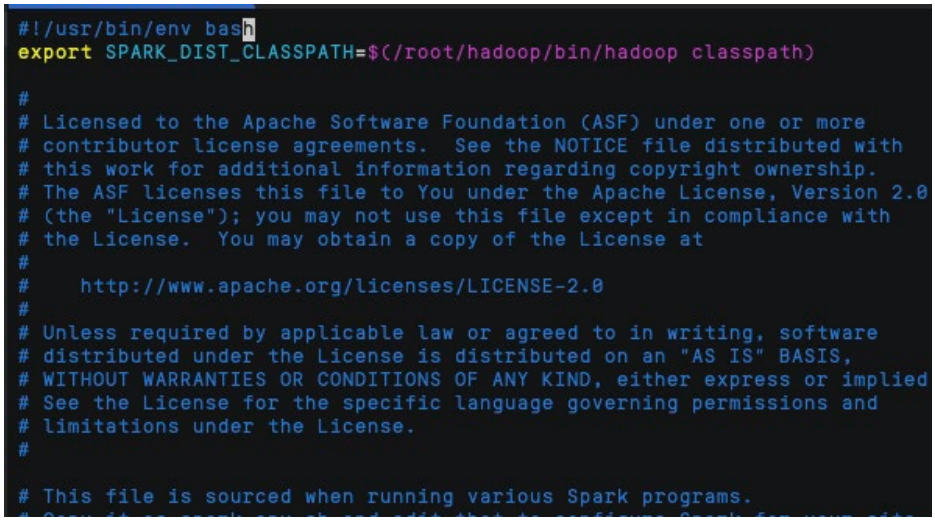
（1） 进入 spark/conf

```
cd spark/conf
mv spark-env.sh.template spark-env.sh
vim spark-env.sh
```

（2） 添加以下内容

```
export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

这里的$(/usr/local/hadoop/bin/hadoop classpath)是指 hadoop 配置文件路径，用于将 Hadoop 的类路径添加到 Spark 中，确保 Spark 能够访问 Hadoop 的库和配置。



（3） 在 shell 配置文件（如 .bashrc 或 .profile）中配置 spark 位置、HADOOP_CONF_DIR 以及 YARN_CONF_DIR。

进入到配置文件中

```
vim .bashrc
```

添加如下配置

```
export SPARK_HOME=/root/spark
export PATH=$SPARK_HOME/bin:$PATH
export HADOOP_CONF_DIR=/root/hadoop/etc/hadoop
export YARN_CONF_DIR=/root/hadoop/etc/hadoop
```

应用新的配置文件

```
source ~/.bashrc
```

步骤三：配置 workers

进入 spark/conf，发现存在一个 workers.template 文件，将其改名为 workers.

进入 workers

```
vim workers
```

添加

```
master
slave1
slave2
slave3
```

步骤四：将配置好的 spark 文件复制到从节点中

由于容器与容器之间无法直接进行文件传输，所以将配置好的文件复制到从节点中需要进行一下步骤：

（1） 将 master 容器中的 spark 文件复制到宿主机

```
docker cp <container_name or container_id>:/path/to/file /path/on/host
```

（2） 将宿主机中的 spark 文件复制到指定从节点中

```
docker cp /path/on/host <container_name or container_id>:/path/to/file
```

步骤五：重启 Hadoop 集群

（1） stop-all.sh (如果你已经启动了 hadoop 需要停止)

（2） start-all.sh

（3） 使用 jps 查看集群是否启动成功；

若发现重启后没有 spark 相关信息（jps 中没有 master 和 worker），则需要进入 spark 配置文件中手动启动 master 以及 worker。

进入 spark 的配置文件

```
cd spark/sbin
bash start-master.sh
bash start-workers.sh
```

```
root@master:~/spark# cd sbin/
root@master:~/spark/sbin# bash start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /root/spark/logs/spark--org.apache.spark.deploy.master.Master-1-master.out
```

```
root@master:~/spark/sbin# bash start-workers.sh
slave1: starting org.apache.spark.deploy.worker.Worker, logging to /root/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-slave1.out
master: starting org.apache.spark.deploy.worker.Worker, logging to /root/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-master.out
slave3: starting org.apache.spark.deploy.worker.Worker, logging to /root/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-slave3.out
slave2: starting org.apache.spark.deploy.worker.Worker, logging to /root/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-slave2.out
```

然后 jps 查看集群信息

```
root@master:~# jps
1571 DataNode
5398 Jps
5318 Worker
1191 SecondaryNameNode
440 ResourceManager
952 NameNode
5084 Master
```

步骤六：运行 spark-shell 命令查看 spark 版本信息

```
root@master:~# spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Scala version 2.13.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_402)
Type in expressions to have them evaluated.
Type :help for more information.
24/05/10 08:33:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://master:4040
Spark context available as 'sc' (master = local[*], app id = local-1715330024327).
Spark session available as 'spark'.

scala>
```

可以看到 spark 版本为 3.5.1，scala 版本为 2.13.8。

步骤七：为了确保 spark 可以正常工作，在 spark shell 中执行一个简单的数据处理任务，例如计算一个数列的总和。

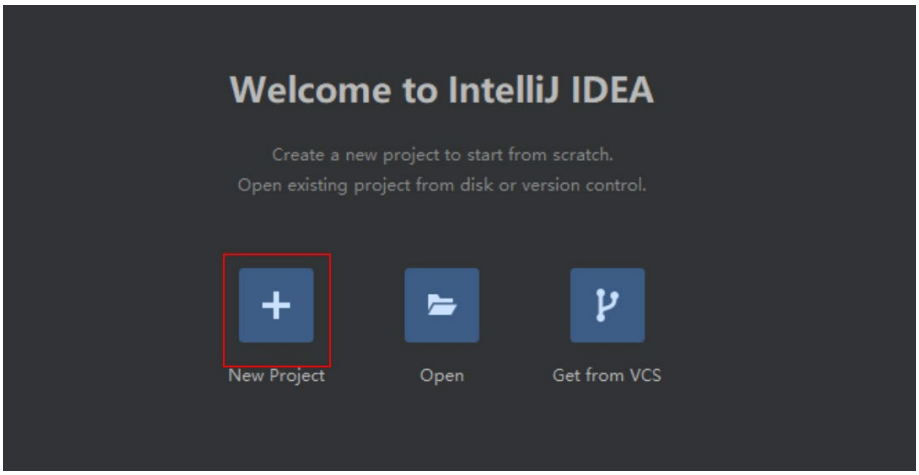
```
val nums = sc.parallelize(Seq(1, 2, 3, 4))
val sum = nums.reduce(_ + _)
println(sum)
```

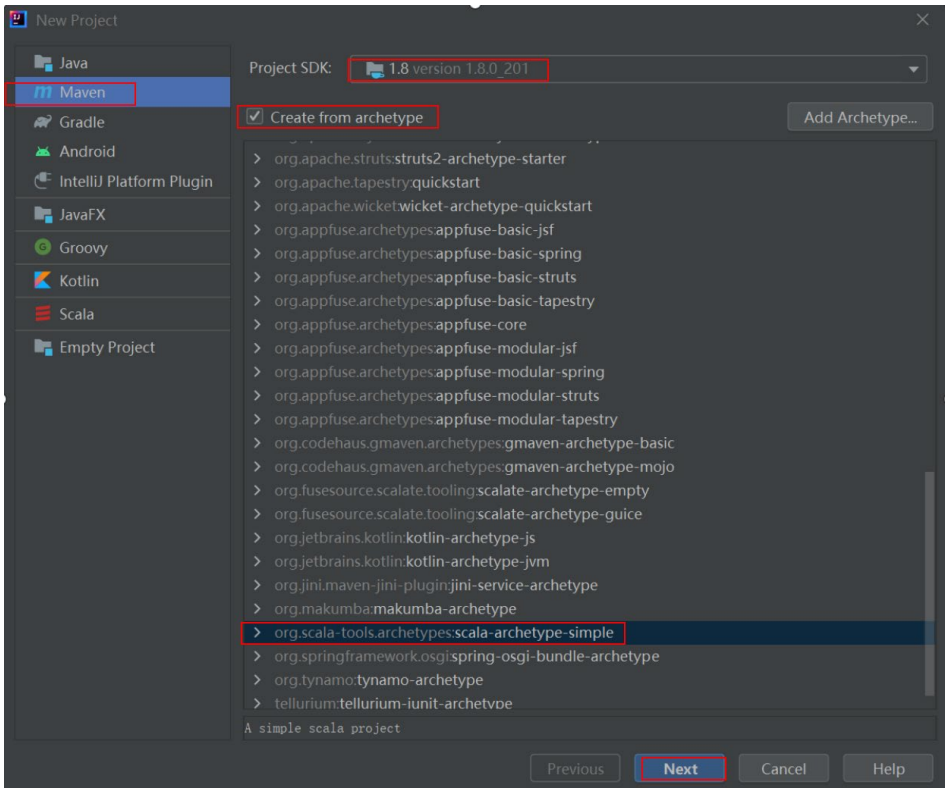
如果这段代码能够无错误执行并输出结果 10，说明 Spark 环境已经正确配置并启动成功。

```
scala> val nums = sc.parallelize(Seq(1, 2, 3, 4))
     | val sum = nums.reduce(_ + _)
     | println(sum)
     |
10
val nums: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:1
val sum: Int = 10

scala>
```

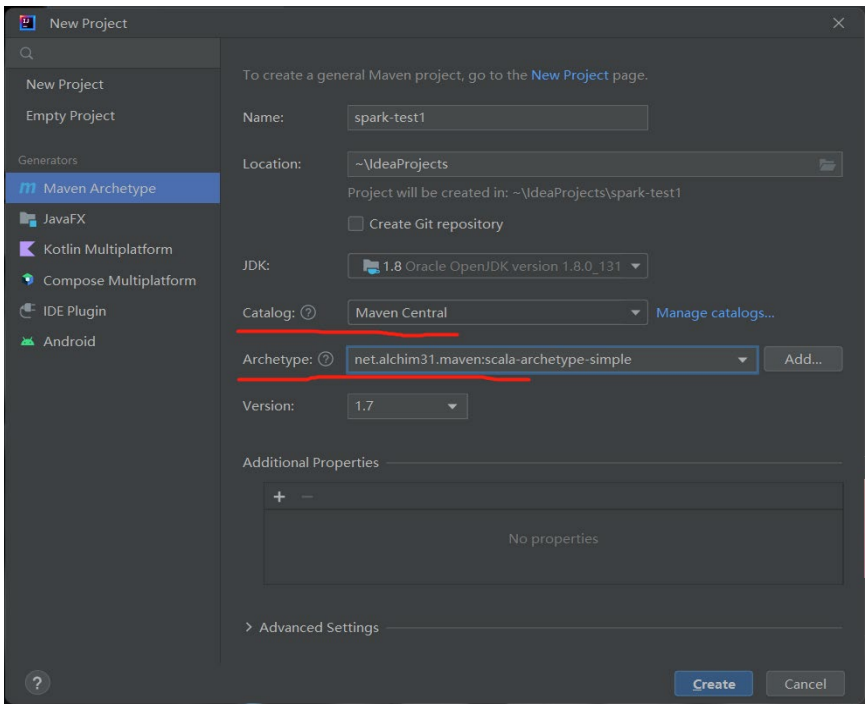
3、 程序编写

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：

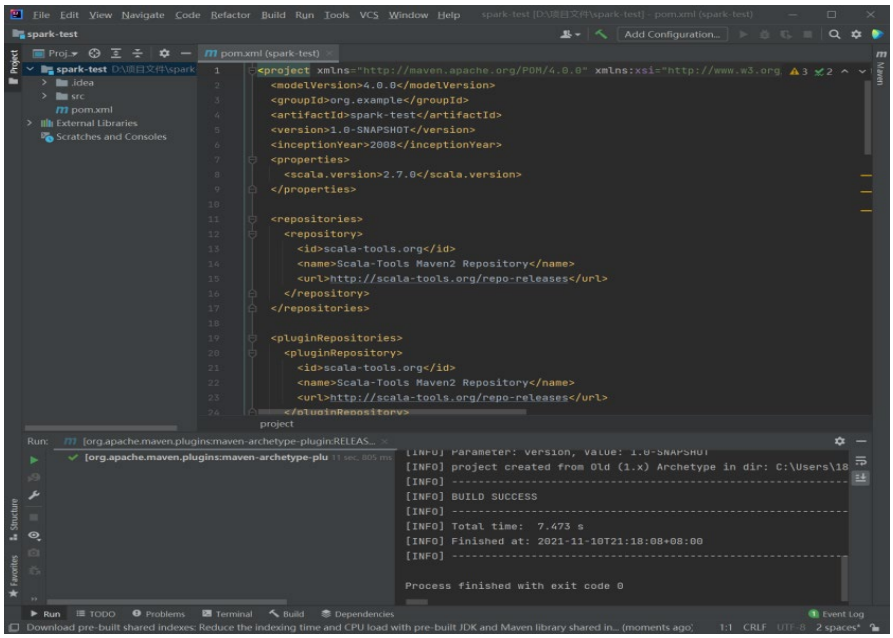




若是使用的 idea 版本较新，如 2022 版本的，则需在创建项目时 catalog 项选择 Maven Central 才能找到 scala-archetype-simple。



进入如下界面表示工程创建成功：



步骤 2：依赖设置：

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）；

```
<properties>
  <maven.compiler.source>1.5</maven.compiler.source>
  <maven.compiler.target>1.5</maven.compiler.target>
  <encoding>UTF-8</encoding>
  <scala.version>2.13.8</scala.version>
  <spark.version>3.5.1</spark.version>
</properties>
```

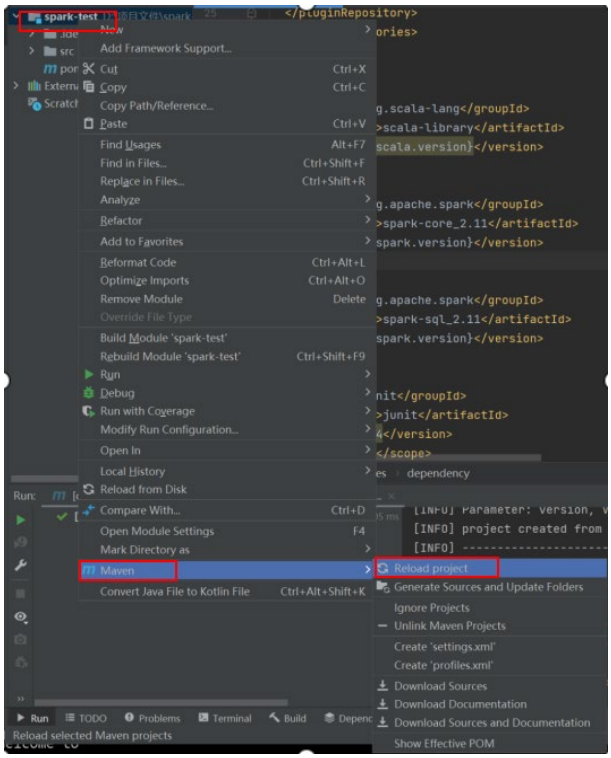
找到 dependency 配置项，添加如下图标红部分的配置，分别是 scala 依赖和 spark 依赖，\${scala.version}表示上述配置的 scala.version 变量；

```
<dependencies>
  <dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.12</artifactId>
    <version>${spark.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.12</artifactId>
    <version>${spark.version}</version>
  </dependency>
```

完整的 dependencies 如下：注意核对依赖的版本信息！以防出现 maven 中依赖版本与部署的版本不一致的情况！

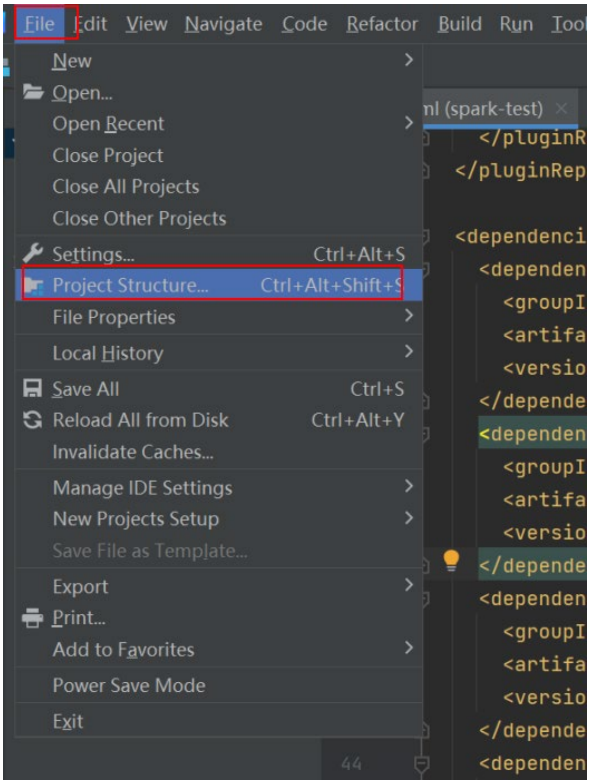
```
<dependencies>
  <dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.12</artifactId>
    <version>${spark.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.12</artifactId>
    <version>${spark.version}</version>
  </dependency>

  <!-- Test -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.specs</groupId>
    <artifactId>specs</artifactId>
    <version>1.2.5</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.scalatest</groupId>
    <artifactId>scalatest</artifactId>
    <version>1.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

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；

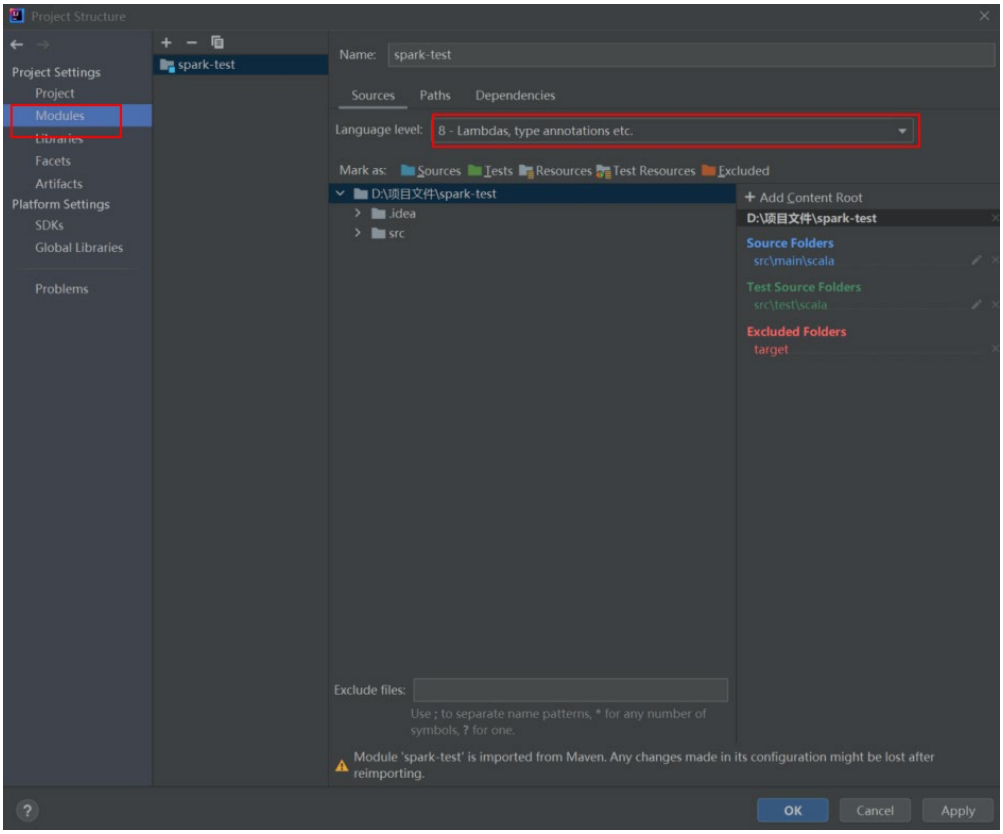


步骤 3：设置语言环境：

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

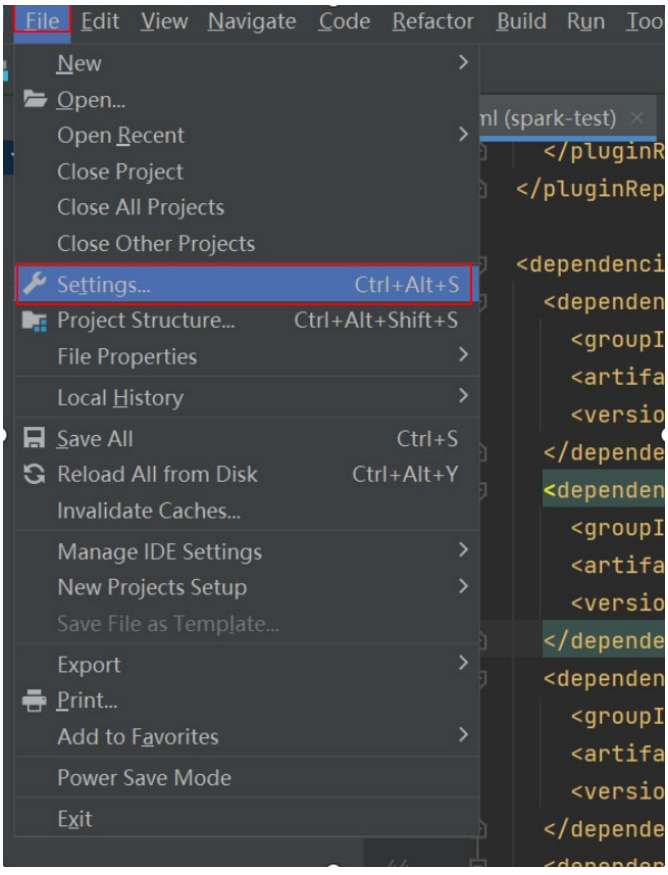


弹出如下对话框，选择 Modules，选择 Language level 为 8，然后点击 Apply，点击 OK；

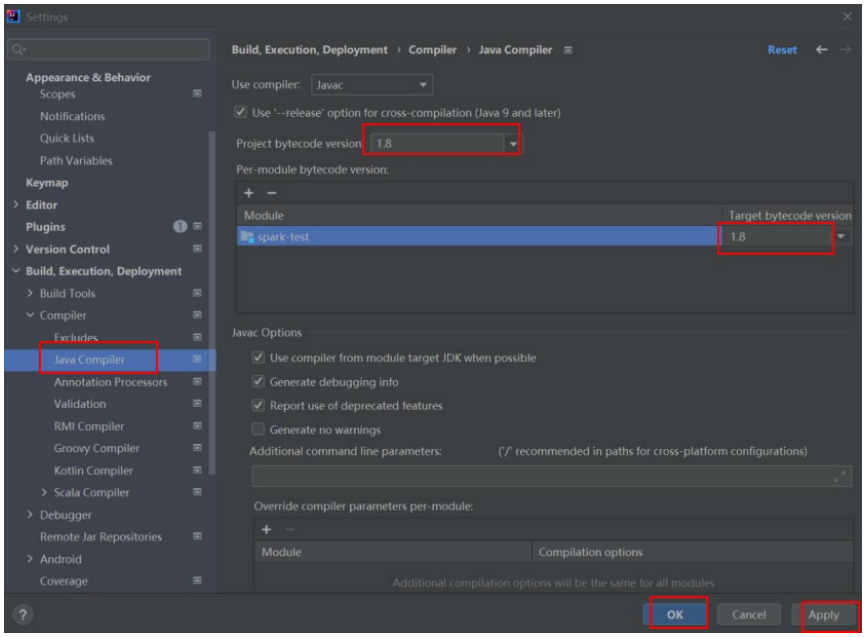


步骤 4：设置 java Compiler 环境：

点击菜单栏中的 file，选择 Setting；

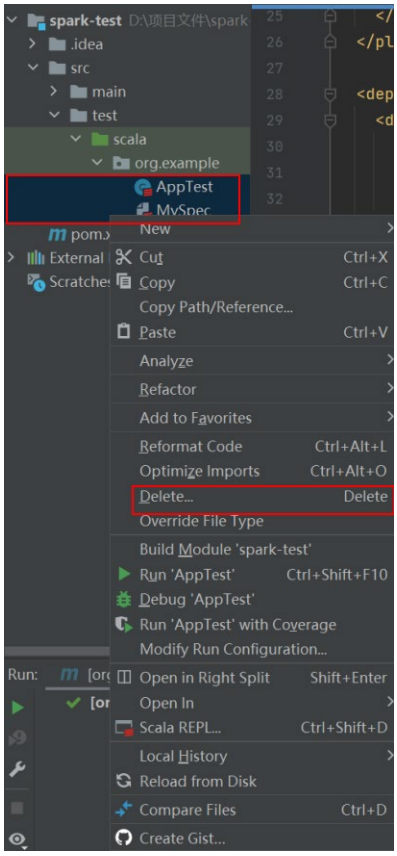


弹出如下对话框，依次选择 Build，Execution—->Compiler—->Java Compiler，设置图中的 Project bytecode version 为 1.8，设置图中的 Target bytecode version 为 1.8，然后依次点击 Apply 和 OK；

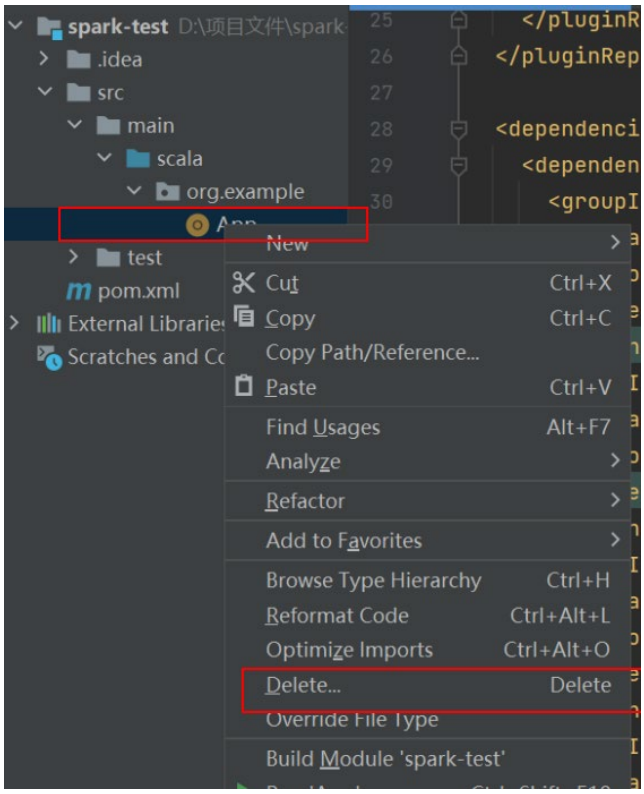


步骤 5：文件配置：

如下图删除测试环境 test 中的测试类；

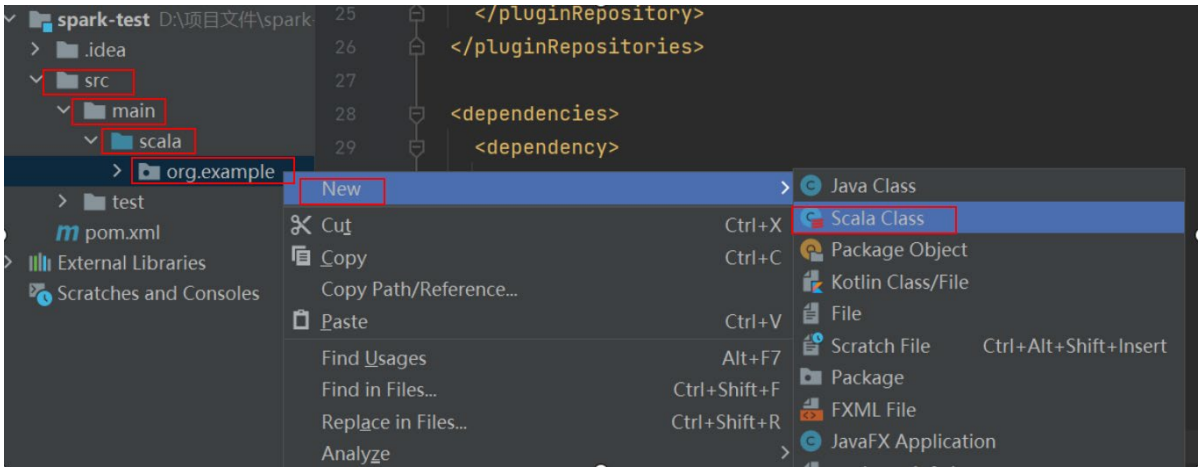


如下图删除，main 文件夹中，包名下的 App 文件；

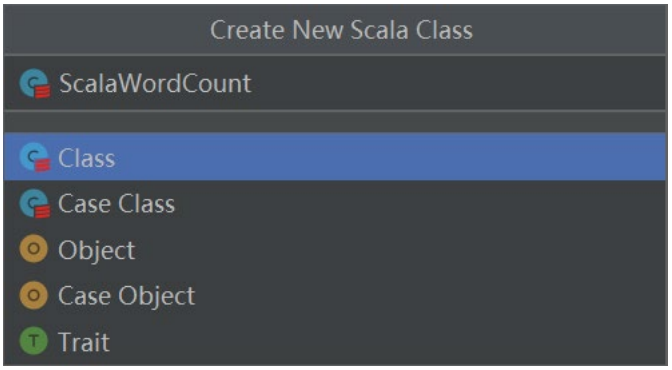


步骤 6： 程序编写：

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



弹出如下对话框，输入类名 ScalaWordCount，回车



在类 ScalaWordCount 中新建伴生对象 object ScalaWordCount；

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

在伴生对象 object ScalaWordCount 中创建 main 方法;

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

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 sparkkgraphx")
```

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；

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)
```

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)
```

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 = " ")})
```

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)})
```

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)
```

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)
```

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(","))
```

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；

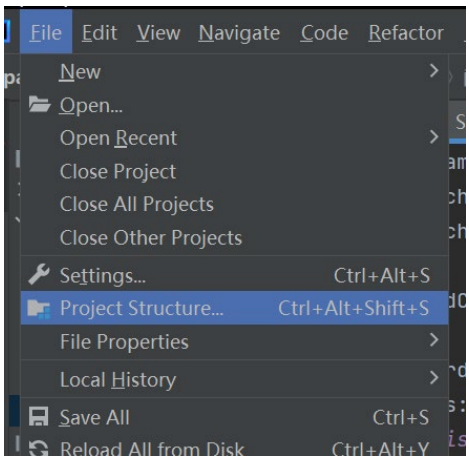
```
ret.saveAsTextFile( path = "hdfs://yty-2022140804-0001:9000/spark_test")
sc.stop
```

完整程序如下：

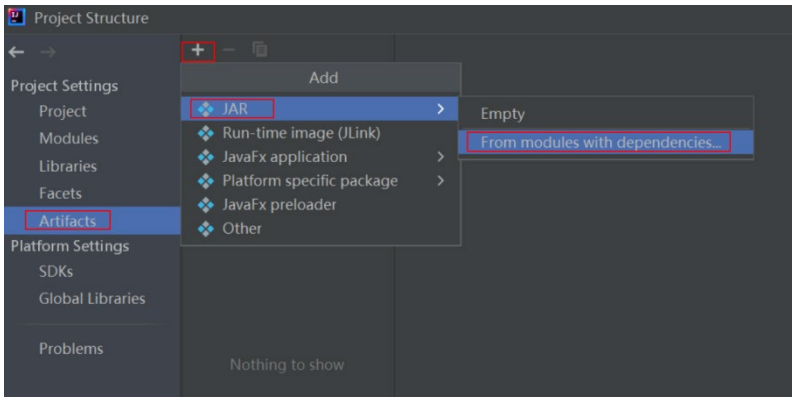
```
package org.example
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}

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( regex = " ")})
    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
  }
}
```

4、 程序打包及运行

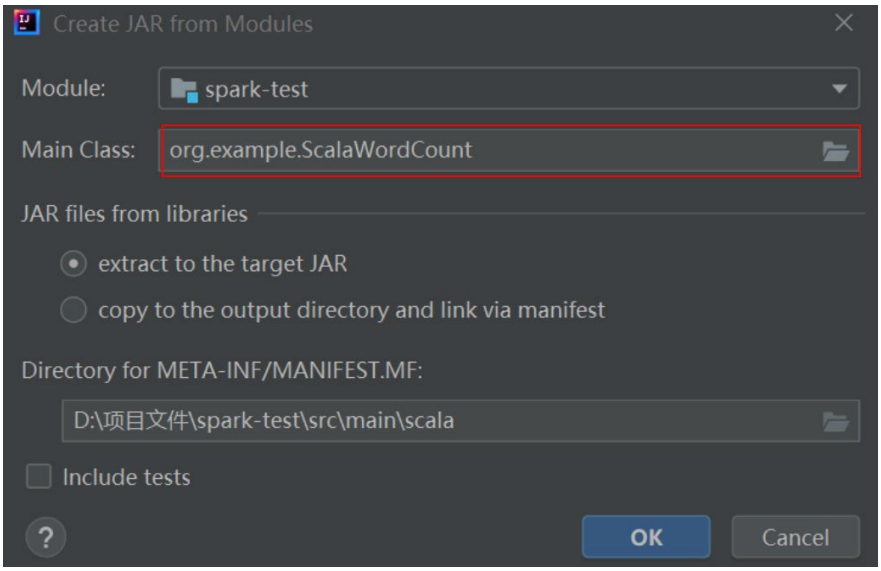
步骤 1：打开 File->Project Structure：



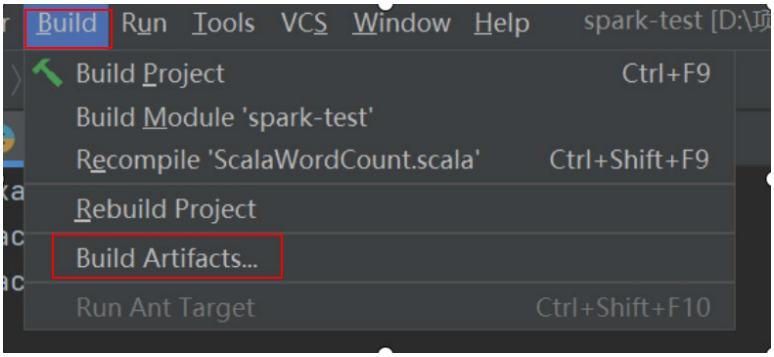
步骤 2：Project Settings 栏下的 Artifacts，点击“+”,选择 JAR->From modules with dependencies…:



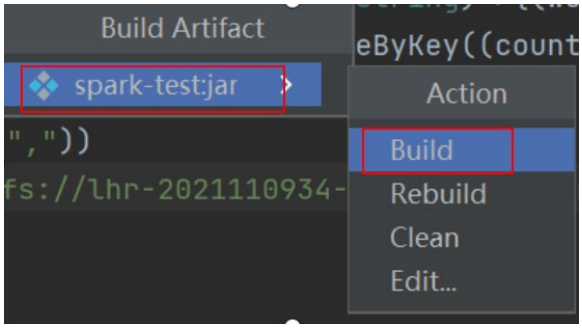
步骤 3：填选主类名称：



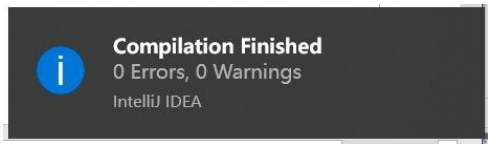
步骤 4：选择 Build->Artifacts:



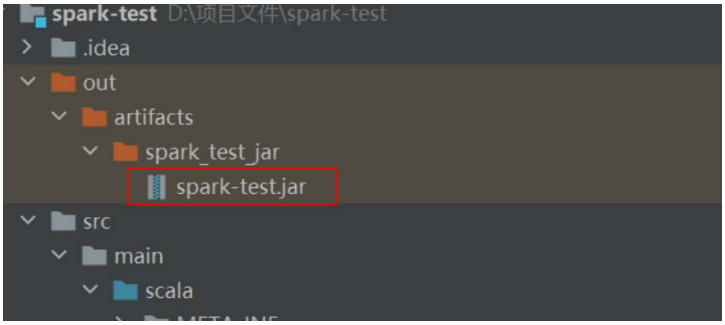
步骤 5：选择 Build：



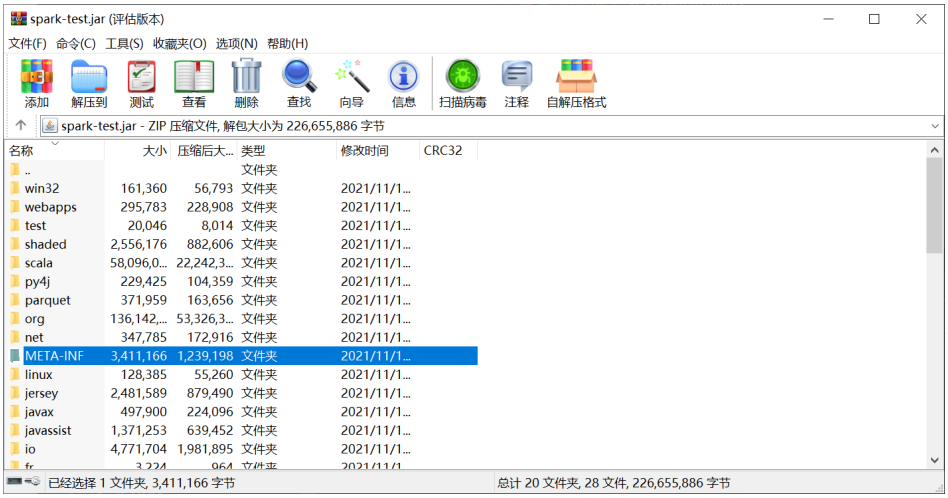
建立完成：



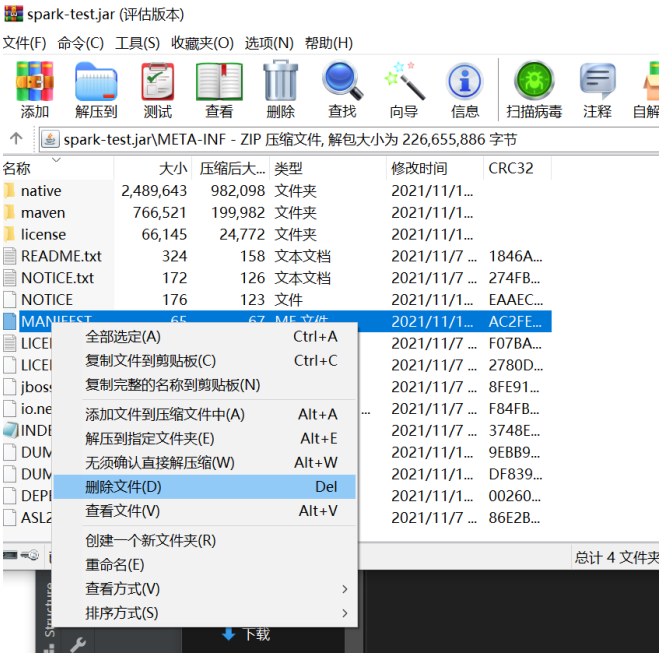
步骤 6：使用压缩软件打开生成的 jar 包：



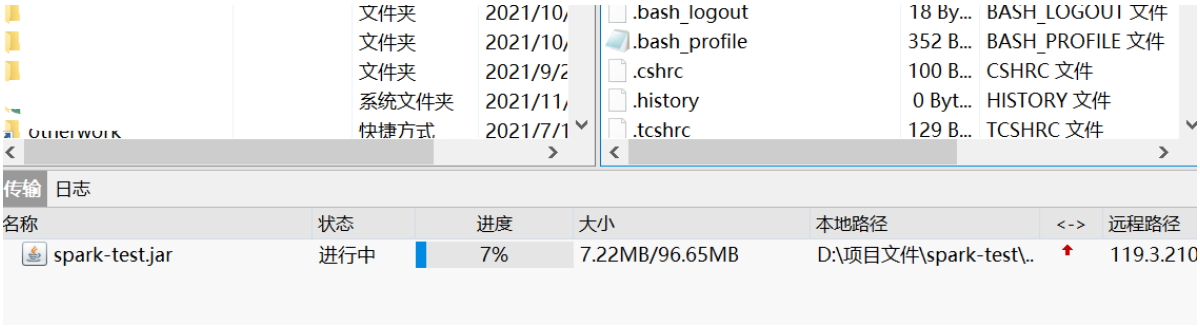
步骤 7：找到 META-INF 目录



步骤 8：删除 MANIFEST.MF 文件



步骤 9：使用 Xftp 上传处理后的 jar 包到服务器：



步骤 10：将宿主机中的文件复制到 master 的 docker 容器中

具体文件复制方法见 Spark 集群搭建部分步骤四.

步骤 11: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：

```
24/05/10 08:54:16 INFO TaskSetManager: Starting task 1.0 in stage 4.0 (TID 10) (slave3, executor 1, partition 1, NODE_LOCAL, 7692 bytes)
24/05/10 08:54:16 INFO TaskSetManager: Finished task 0.0 in stage 4.0 (TID 9) in 37 ms on slave3 (executor 1) (1/3)
24/05/10 08:54:16 INFO TaskSetManager: Starting task 2.0 in stage 4.0 (TID 11) (slave3, executor 1, partition 2, NODE_LOCAL, 7692 bytes)
24/05/10 08:54:16 INFO TaskSetManager: Finished task 1.0 in stage 4.0 (TID 10) in 12 ms on slave3 (executor 1) (2/3)
24/05/10 08:54:16 INFO TaskSetManager: Finished task 2.0 in stage 4.0 (TID 11) in 13 ms on slave3 (executor 1) (3/3)
24/05/10 08:54:16 INFO YarnScheduler: Removed TaskSet 4.0, whose tasks have all completed, from pool
24/05/10 08:54:16 INFO DAGScheduler: ResultStage 4 (collect at ScalaWordCount.scala:27) finished in 0.067 s
24/05/10 08:54:16 INFO DAGScheduler: Job 1 is finished. Cancelling potential speculative or zombie tasks for this job
24/05/10 08:54:16 INFO YarnScheduler: Killing all running tasks in stage 4: Stage finished
24/05/10 08:54:16 INFO DAGScheduler: Job 1 finished: collect at ScalaWordCount.scala:27, took 0.178357 s
```

步骤 11：在 hdfs 上查看程序的输出：

```
root@master:~# hadoop fs -ls /
2024-05-10 08:57:32,925 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
drwxr-xr-x   - root supergroup          0 2024-05-09 23:38 /spark-test
drwxr-xr-x   - root supergroup          0 2024-05-09 22:47 /user
```

```
[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)
```

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.txt 的样例如下：

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

输入文件 B.txt 的样例如下：

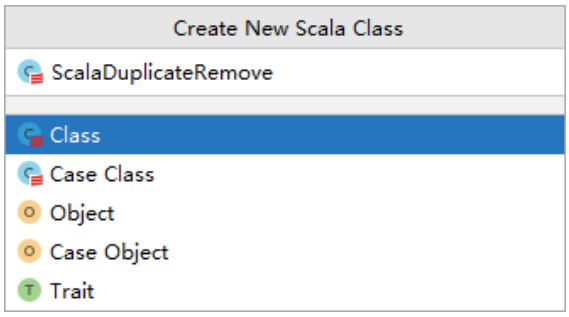
20170101 y
20170102 y
20170103 x
20170104 z
20170105 y

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

在 1.1.3.1 的项目中，如下图依次打开 src—>main—>scala，在 org.example 上点击右键，创建 Scala Class，命名为：ScalaDuplicateRemove；



在类 ScalaDuplicateRemove 中新建伴生对象 object ScalaDuplicateRemove；

```
object ScalaDuplicateRemove {

}
```

在伴生对象 object ScalaDuplicateRemove 中创建 main 方法；

```
object ScalaDuplicateRemove {
  new *
  def main(args: Array[String]): Unit = {
```

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“Scala Duplicate Remover”，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 local 模式，参数为“local”，属性设置完成后赋值给常量 sparkConf；

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

```
val conf = new SparkConf().setAppName("Scala Duplicate Remover").setMaster("local")
val sc = new SparkContext(conf)
```

定义 basePath 为 A.txt 和 B.txt 的存放目录”/user/root”， 从指定路径读取两个输入文本文件；

```
val basePath = "/user/root/"
val linesA = sc.textFile(basePath + "A.txt")
val linesB = sc.textFile(basePath + "B.txt")
```

使用 union 方法合并两个 RDD（弹性分布式数据集），调用 distinct 方法去除重复的行，使用 sortBy(identity) 对结果进行排序。这里 identity 函数简单地返回其输入值，意味着按自然顺序排序。

```
val lines = linesA.union(linesB).distinct().sortBy(identity)
```

将处理后的数据保存到 /user/root/C 目录。这会在该目录下创建多个结果文件，每个文件是一个 RDD 分区的输出。

```
lines.saveAsTextFile(basePath + "C")
```

调用 SparkContext 对象 sc 的 stop 方法，释放 spark 程序所占用的资源：

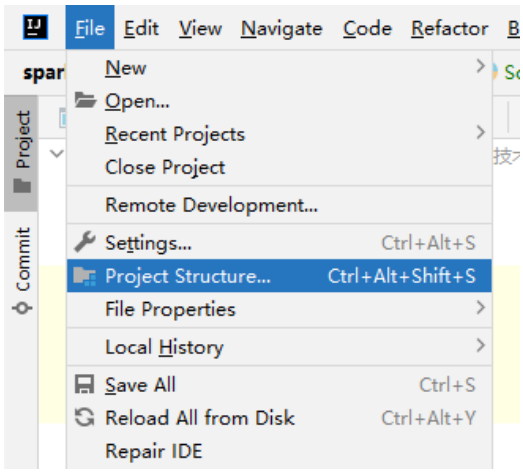
```
sc.stop()
```

完整程序如下：

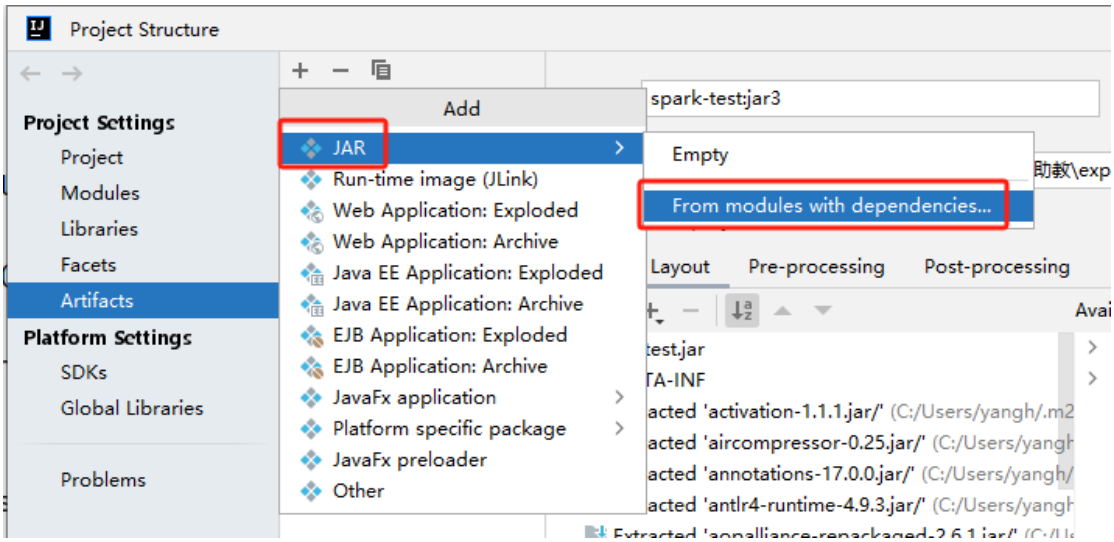
```
package org.example

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

new *
class ScalaDuplicateRemove {

}

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

    val conf = new SparkConf().setAppName("Scala Duplicate Remover").setMaster("local")
    val sc = new SparkContext(conf)

    val basePath = "/user/root/"
    val linesA = sc.textFile(basePath + "A.txt")
    val linesB = sc.textFile(basePath + "B.txt")

    val lines = linesA.union(linesB).distinct().sortBy(identity)

    lines.saveAsTextFile(basePath + "C")

    sc.stop()
  }
}
```

程序打包及运行

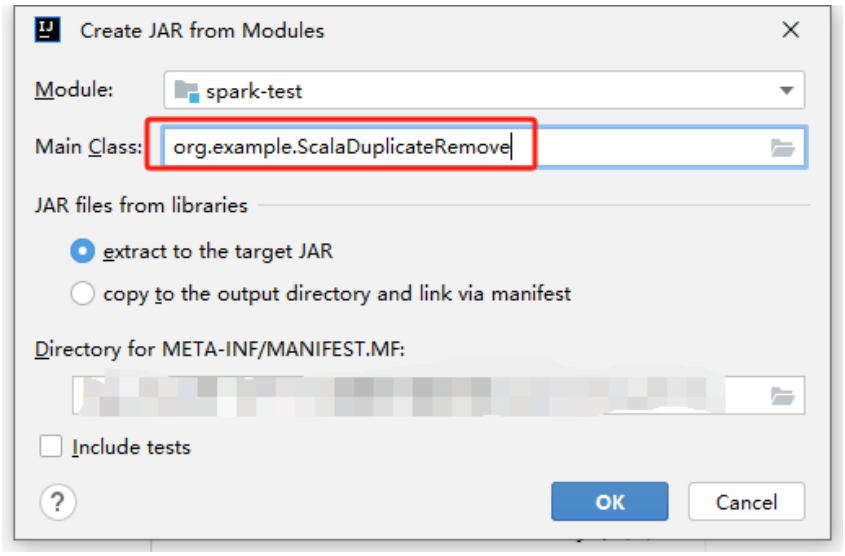
步骤 1：打开 File->Project Structure：

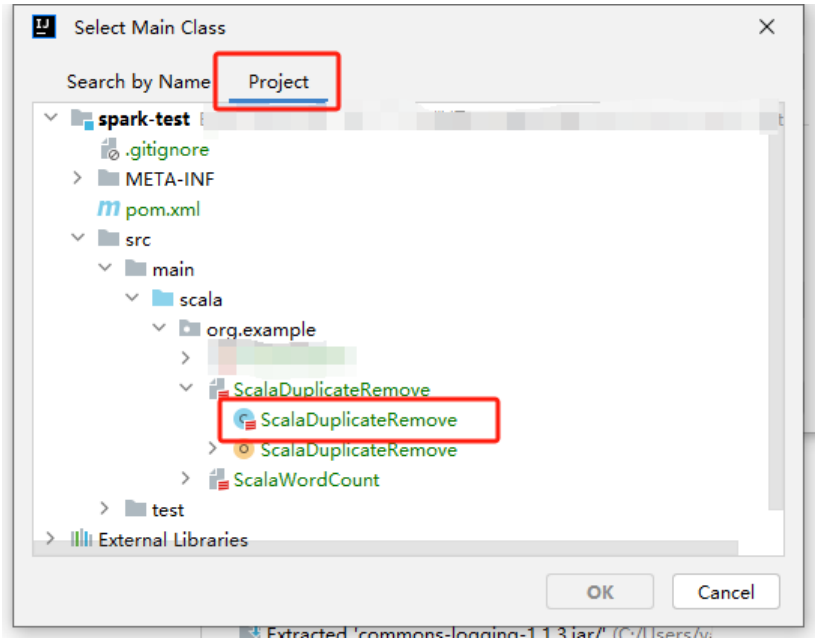


步骤 2：Project Settings 栏下的 Artifacts，点击“+”,选择 JAR->From modules with dependencies…:

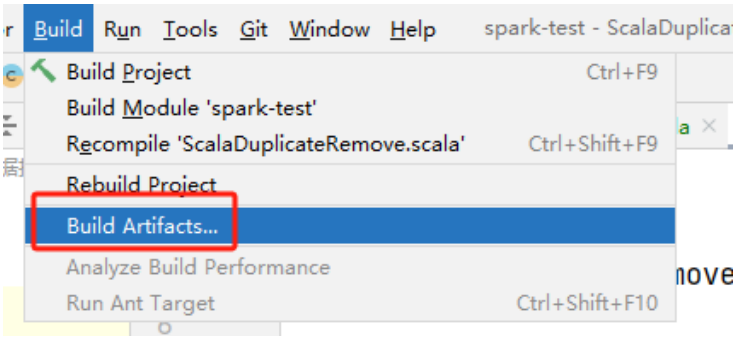


步骤 3：填选主类名称，点击文件夹图标，点击 Project，选择 ScalaDuplicateRemove：

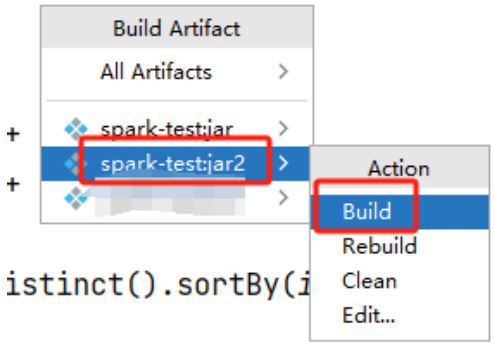




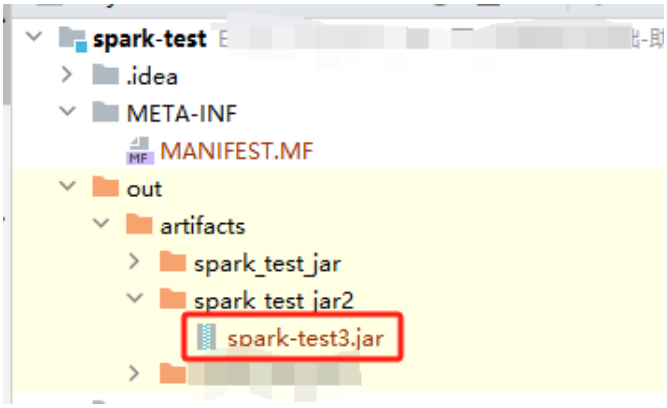
步骤 4：选择 Build->Artifacts:



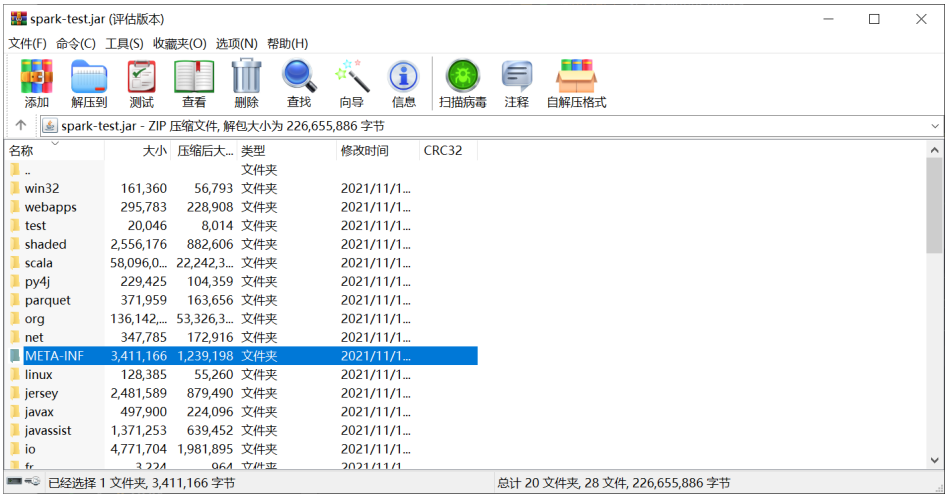
步骤 5：选择 spark-test:jar2 进行 Build：



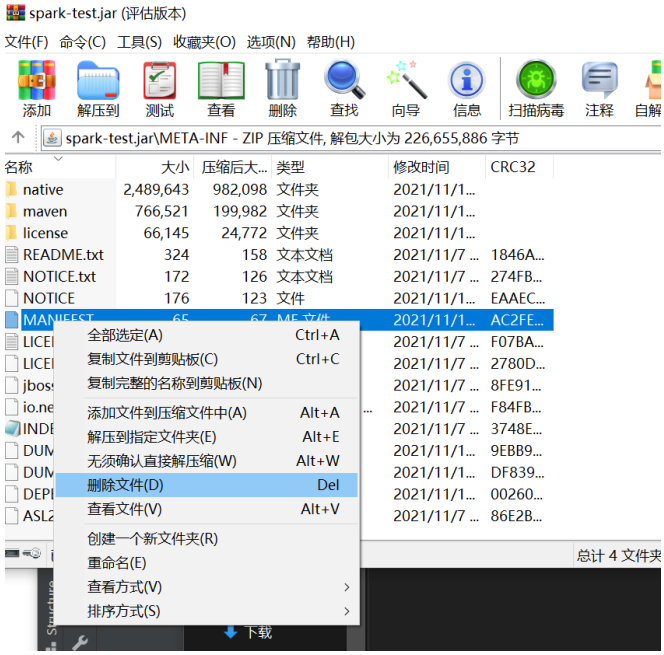
步骤 6：使用压缩软件打开生成的 jar 包：



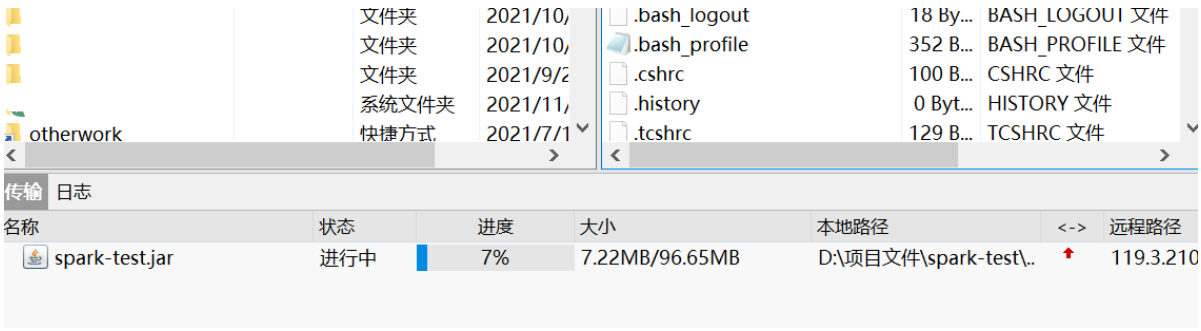
步骤 7：找到 META-INF 目录



步骤 8：删除 MANIFEST.MF 文件



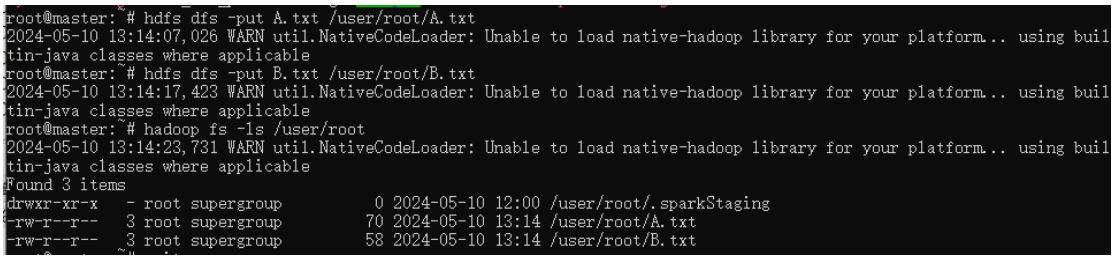
步骤 9：使用 Xftp 上传处理后的 jar 包（**为避免与 1.1.3.1 实验生成的 jar 包重名，可以在上传之前将本实验生成的 jar 包重命名，比如 spark-test2.jar**）、A.txt、B.txt 到服务器：



步骤 10：将宿主机中的文件（上传的 jar 包、A.txt、B.txt）复制到 master 的 docker 容器中

```
docker cp A.txt master:/root/
docker cp A.txt master:/root/
docker cp spark-test2.jar master:/root/
```

步骤 11：将上传至 master 节点的 A.txt、B.txt 复制到 hdfs 中，在 master 节点中运行 hdfs dfs -put A.txt /user/root/A.txt （B.txt 同理），使用 hadoop fs -ls /user/root 查看 hdfs 中是否已经成功复制 A.txt、B.txt。



步骤 11:使用 spark-submit 命令，在 master 节点上运行程序：（**注意 jar 包名与上传名称保持一致**）

```
spark-submit --class org.example. ScalaDuplicateRemove --master yarn --num-executors 3
--driver-memory 1g --executor-memory 1g --executor-cores 1 spark-test2.jar
```

得到如下结果：

```
[root@zj-2023140696 ~]# docker exec -it master bash
root@master:~# spark-submit --class org.example.ScalaDuplicateRemove --master yarn --num-executors 3 --driver-memory 1g
--executor-memory 1g --executor-cores 1 spark-test3.jar
24/05/10 13:15:26 INFO SparkContext: Running Spark version 3.5.1
24/05/10 13:15:26 INFO SparkContext: OS info Linux, 4.18.0-80.7.2.el7.aarch64, aarch64
24/05/10 13:15:26 INFO SparkContext: Java version 1.8.0_402
24/05/10 13:15:27 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
24/05/10 13:15:27 INFO ResourceUtils: ==============================================================
24/05/10 13:15:27 INFO ResourceUtils: No custom resources configured for spark.driver.
24/05/10 13:15:27 INFO ResourceUtils: ==============================================================
24/05/10 13:15:27 INFO SparkContext: Submitted application: Scala Duplicate Remover
24/05/10 13:15:27 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, a
mount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount
: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
24/05/10 13:15:27 INFO ResourceProfile: Limiting resource is cpus at 1 tasks per executor
24/05/10 13:15:27 INFO ResourceProfileManager: Added ResourceProfile id: 0
24/05/10 13:15:27 INFO SecurityManager: Changing view acls to: root
24/05/10 13:15:27 INFO SecurityManager: Changing modify acls to: root
24/05/10 13:15:27 INFO SecurityManager: Changing view acls groups to:
24/05/10 13:15:27 INFO SecurityManager: Changing modify acls groups to:
24/05/10 13:15:27 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view perm
issions: root; groups with view permissions: EMPTY; users with modify permissions: root; groups with modify permissions:
 EMPTY
24/05/10 13:15:27 INFO Utils: Successfully started service 'sparkDriver' on port 37873.
24/05/10 13:15:27 INFO SparkEnv: Registering MapOutputTracker
24/05/10 13:15:27 INFO SparkEnv: Registering BlockManagerMaster
24/05/10 13:15:27 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topo
logy information
24/05/10 13:15:27 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
24/05/10 13:15:27 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
24/05/10 13:15:27 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-17b426f9-b8b4-44de-8d12-9209562d35dd
24/05/10 13:15:27 INFO MemoryStore: MemoryStore started with capacity 366.3 MiB
24/05/10 13:15:27 INFO SparkEnv: Registering OutputCommitCoordinator
24/05/10 13:15:27 INFO JettyUtils: Start Jetty 0.0.0.0:4040 for SparkUI
24/05/10 13:15:27 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
24/05/10 13:15:27 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
24/05/10 13:15:27 INFO Utils: Successfully started service 'SparkUI' on port 4042.
```

步骤 12：在 hdfs 上查看程序的输出：

```
root@master:~# hadoop fs -cat /user/root/C/part-00000
2024-05-10 13:15:59,463 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
tin-java classes where applicable
20170101        x
20170101        y
20170102        y
20170103        x
20170104        y
root@master:~# hadoop fs -cat /user/root/C/part-00001
2024-05-10 13:17:01,309 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
tin-java classes where applicable
20170104        z
20170105        y
20170105        z
20170106        z
```

1.1.3.3. 使用 Spark SQL 读写数据库

步骤 1：在 master 节点中下载并安装 MySQL

apt-get update

```
root@master:~# apt-get update
Hit:1 http://ports.ubuntu.com/ubuntu-ports jammy InRelease
Get:2 http://ports.ubuntu.com/ubuntu-ports jammy-updates InRelease [119 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports jammy-backports InRelease [109 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports jammy-security InRelease [110 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports jammy-updates/universe arm64 Packages [1301 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 Packages [1778 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports jammy-updates/restricted arm64 Packages [1751 kB]
Get:8 http://ports.ubuntu.com/ubuntu-ports jammy-backports/universe arm64 Packages [29.9 kB]
Get:9 http://ports.ubuntu.com/ubuntu-ports jammy-security/main arm64 Packages [1516 kB]
Get:10 http://ports.ubuntu.com/ubuntu-ports jammy-security/universe arm64 Packages [1015 kB]
Get:11 http://ports.ubuntu.com/ubuntu-ports jammy-security/restricted arm64 Packages [1688 kB]
Fetched 9418 kB in 8s (1115 kB/s)
Reading package lists... Done
```

apt-get install mysql-server

```
root@master:~# apt-get install mysql-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libaio1 libcgi-fast-perl libcgi-pm-perl libclone-perl libencode-locale-perl libevent-core-2.1-7
  libevent-pthreads-2.1-7 libfcgi-bin libfcgi-perl libfcgi0ldbl libgdbm-compat4 libgdbm6 libhtml-parser-perl
  libhtml-tagset-perl libhtml-template-perl libhttp-date-perl libhttp-message-perl libio-html-perl
  liblwp-mediatypes-perl libmecab2 libnuma1 libperl5.34 libprotobuf-lite23 libtimedate-perl liburi-perl mecab-ipadic
  mecab-ipadic-utf8 mecab-utils mysql-client-8.0 mysql-client-core-8.0 mysql-common mysql-server-8.0
  mysql-server-core-8.0 netbase perl perl-modules-5.34 psmisc
Suggested packages:
  gdbm-l10n libdata-dump-perl libipc-sharedcache-perl libbusiness-isbn-perl libwww-perl mailx tinyca perl-doc
  libterm-readline-gnu-perl | libterm-readline-perl-perl make libtap-harness-archive-perl
The following NEW packages will be installed:
  libaio1 libcgi-fast-perl libcgi-pm-perl libclone-perl libencode-locale-perl libevent-core-2.1-7
  libevent-pthreads-2.1-7 libfcgi-bin libfcgi-perl libfcgi0ldbl libgdbm-compat4 libgdbm6 libhtml-parser-perl
  libhtml-tagset-perl libhtml-template-perl libhttp-date-perl libhttp-message-perl libio-html-perl
  liblwp-mediatypes-perl libmecab2 libnuma1 libperl5.34 libprotobuf-lite23 libtimedate-perl liburi-perl mecab-ipadic
  mecab-ipadic-utf8 mecab-utils mysql-client-8.0 mysql-client-core-8.0 mysql-common mysql-server mysql-server-8.0
  mysql-server-core-8.0 netbase perl perl-modules-5.34 psmisc
0 upgraded, 38 newly installed, 0 to remove and 6 not upgraded.
Need to get 37.3 MB of archives.
After this operation, 286 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 mysql-common all 5.8+1.0.8 [7212 B]
Get:2 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 mysql-client-core-8.0 arm64 8.0.36-0ubuntu0.22.04.1
[2950 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 mysql-client-8.0 arm64 8.0.36-0ubuntu0.22.04.1 [22.7
 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 libaio1 arm64 0.3.112-13build1 [7018 B]
Get:5 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 libevent-core-2.1-7 arm64 2.1.12-stable-1build3 [91.1 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 libevent-pthreads-2.1-7 arm64 2.1.12-stable-1build3 [7588 B]
Get:7 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 libmecab2 arm64 0.996-14build9 [188 kB]
Get:8 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 libnuma1 arm64 2.0.14-3ubuntu2 [22.4 kB]
Get:9 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 libprotobuf-lite23 arm64 3.12.4-1ubuntu7.22.04.1 [19
2 kB]
Get:10 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 mysql-server-core-8.0 arm64 8.0.36-0ubuntu0.22.04.1
 [17.0 MB]
```

步骤 2：启动 MySQL

```
service mysql start
```

```
root@master:~# service mysql start
 * Starting MySQL database server mysqld                                    [ OK ]
```

```
mysql -uroot -p （默认密码为root）
```

```
root@master:~# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.36-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

MySQL 8 默认使用 caching_sha2_password 作为其认证插件，这可能与一些客户端库不兼容。更改用户的认证插件回到 mysql_native_password：

```
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'root'; （最后一个root为用户的MySQL密码）
```

更新权限：

```
FLUSH PRIVILEGES;
```

```
mysql> ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'root';
Query OK, 0 rows affected (0.01 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

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：

```
create database spark;
use spark;
create table student(id int(4), name char(20), gender char(4), age int(4));
insert into student values(1, 'Li', 'F', 23);
insert into student values(2, 'Wang', 'M', 24);
select * from student;
```

```
mysql> create database spark;
Query OK, 1 row affected (0.01 sec)

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 (0.03 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)

mysql>
```

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
```

```
root@master:~# wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.24.tar.gz
--2024-05-10 13:52:21--  https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.24.tar.gz
Resolving dev.mysql.com (dev.mysql.com)... 69.192.12.46, 2600:1417:76:589::2e31, 2600:1417:76:58b::2e31
Connecting to dev.mysql.com (dev.mysql.com)|69.192.12.46|:443... connected.
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: https://cdn.mysql.com//archives/mysql-connector-java-8.0/mysql-connector-java-8.0.24.tar.gz [following]
--2024-05-10 13:52:22--  https://cdn.mysql.com//archives/mysql-connector-java-8.0/mysql-connector-java-8.0.24.tar.gz
Resolving cdn.mysql.com (cdn.mysql.com)... 23.77.214.217
Connecting to cdn.mysql.com (cdn.mysql.com)|23.77.214.217|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4079141 (3.9M) [application/x-tar-gz]
Saving to: 'mysql-connector-java-8.0.24.tar.gz'

mysql-connector-java-8.0.24.t 100%[=============================================>]   3.89M  5.65MB/s    in 0.7s

2024-05-10 13:52:26 (5.65 MB/s) - 'mysql-connector-java-8.0.24.tar.gz' saved [4079141/4079141]
```

```
tar -xzvf mysql-connector-java-8.0.24.tar.gz
```

```
root@master:~# tar -xzvf mysql-connector-java-8.0.24.tar.gz
mysql-connector-java-8.0.24/
mysql-connector-java-8.0.24/src/
mysql-connector-java-8.0.24/src/build/
mysql-connector-java-8.0.24/src/build/java/
mysql-connector-java-8.0.24/src/build/java/documentation/
mysql-connector-java-8.0.24/src/build/java/instrumentation/
mysql-connector-java-8.0.24/src/build/misc/
mysql-connector-java-8.0.24/src/build/misc/debian.in/
```

```
mv mysql-connector-java-8.0.24/mysql-connector-java-8.0.24.jar /root/spark/jars/（注意用户自己的spark安装路径）
```

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/jars/mysql-connector-java-8.0.24.jar --driver-class-path /root/spark/jars/mysql-connector-java-8.0.24.jar （注意用户自己的spark安装路径）
```

```
root@master:~# spark-shell --jars /root/spark/jars/mysql-connector-java-8.0.24.jar --driver-class-path /root/spark/jars/mysql-connector-java-8.0.24.jar
24/05/10 13:59:33 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Scala version 2.13.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_402)
Type in expressions to have them evaluated.
Type :help for more information.
24/05/10 13:59:41 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
24/05/10 13:59:41 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
Spark context Web UI available at http://master:4042
Spark context available as 'sc' (master = local[*], app id = local-1715349581364).
Spark session available as 'spark'.

scala>
```

spark.read.format(“jdbc”)操作可以实现对 MySQL 数据库的读取。执行以下命令（**从第二行开始的竖线为换行自动输入**）连接数据库，读取数据并显示：（**要求贴出命令和结果的截图**）

```
val jdbcDF = spark.read.format("jdbc").
       | option("url", "jdbc:mysql://localhost:3306/spark").
       | option("driver", "com.mysql.cj.jdbc.Driver").
       | option("dbtable", "student").
       | option("user", "root").
       | option("password", "root").
       | load()
jdbcDF.show()
```

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

scala> jdbcDF.show()
+---+--------------------+------+---+
| id|                name|gender|age|
+---+--------------------+------+---+
|  1|Li                  |  F   | 23|
|  2|Wang                |  M   | 24|
+---+--------------------+------+---+


scala>
```

（运行这一步如果遇见下面的问题，请把“localhost”改为“**127.0.0.1**”,应该就可以解决这个问题，完成后续的实验。该问题一般只出现在使用 docker 进行实验时。）

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：

同 1.1.3.2 实验，在 1.1.3.1 的项目中，依次打开 src—>main—>scala，在 org.example 上点击右键，创建 Scala Class，命名为：InsertStudent；（若遇见上述问题，需要把下述程序中的“localhost”也改为“**127.0.0.1**”。）

```scala
package org.example

import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}

new *
class InsertStudent {

}
new *
object InsertStudent {
  new *
  def main(args: Array[String]): Unit = {
    // 初始化 SparkSession
    val spark = SparkSession.builder()
      .appName( name = "Insert Student")
      .master( master = "local")
      .getOrCreate()

    val sc = spark.sparkContext

    // 学生信息RDD
    val studentData = Array("3 Zhang M 26", "4 liu M 27")
    val studentRDD = sc.parallelize(studentData).map(_.split( regex = "\\s+"))

    // 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(attrs => Row(attrs(0).toInt, attrs(1), attrs(2), attrs(3).toInt))

    // 建立DataFrame
    val studentDF = spark.createDataFrame(rowRDD, schema)

    // JDBC连接参数
    val jdbcUrl = "jdbc:mysql://localhost:3306/spark"
    val connectionProperties = new java.util.Properties()
    connectionProperties.put("user", "root")
    connectionProperties.put("password", "root")
    connectionProperties.put("driver", "com.mysql.cj.jdbc.Driver")

    // 将数据写入数据库
    studentDF.write
      .mode( saveMode = "append")
      .jdbc(jdbcUrl, table = "spark.student", connectionProperties)

    // 停止SparkSession
    spark.stop()
  }
}
```

同 1.1.3.2 实验的打包方式，注意修改 jar 包名称并删除压缩包中的 MANIFEST.MF 文件，将 jar 包上传到服务器，并上传到 master 容器中。在 master 容器中，使用 spark-submit 命令运行程序。

```
spark-submit --class org.example.InsertStudent --master yarn --num-executors 3 --driver-memory 1g --executor-memory 1g --executor-cores 1 spark-test3.jar （注意使用自己的jar包名称）
```

```
root@master:~# spark-submit --class org.example.InsertStudent --master yarn --num-executors 3 --driver-memory 1g --execu
tor-memory 1g --executor-cores 1 spark-test4.jar
24/05/10 15:38:57 INFO SparkContext: Running Spark version 3.5.1
24/05/10 15:38:57 INFO SparkContext: OS info Linux, 4.18.0-80.7.2.e17.aarch64, aarch64
24/05/10 15:38:57 INFO SparkContext: Java version 1.8.0_402
24/05/10 15:38:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
24/05/10 15:38:57 INFO ResourceUtils: ==============================================================
24/05/10 15:38:57 INFO ResourceUtils: No custom resources configured for spark.driver.
24/05/10 15:38:57 INFO ResourceUtils: ==============================================================
24/05/10 15:38:57 INFO SparkContext: Submitted application: Insert Student
24/05/10 15:38:57 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, a
mount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount
: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
24/05/10 15:38:57 INFO ResourceProfile: Limiting resource is cpus at 1 tasks per executor
24/05/10 15:38:57 INFO ResourceProfileManager: Added ResourceProfile id: 0
24/05/10 15:38:57 INFO SecurityManager: Changing view acls to: root
24/05/10 15:38:57 INFO SecurityManager: Changing modify acls to: root
24/05/10 15:38:57 INFO SecurityManager: Changing view acls groups to:
24/05/10 15:38:57 INFO SecurityManager: Changing modify acls groups to:
24/05/10 15:38:57 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view perm
issions: root; groups with view permissions: EMPTY; users with modify permissions: root; groups with modify permissions:
 EMPTY
```

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**）

```
root@master:~# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 32
Server version: 8.0.36-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select * from student;
ERROR 1046 (3D000): No database selected
mysql> use spark;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
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)
```

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， 3 个 Spark 程序 jar 包。
完成 Spark RDD 和 Spark SQL 数据处理实践。

实验评分标准，提交的实验报告中应包含：

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。