北京郵電大學

Beijing University of Posts and Telecommunications

华为云鲲鹏大数据基础实验体系 4
实践 MapReduce 分布式数据处理

编写负责人：鄂海红、程祥

参编人员：张忠宝、朱一凡、宋美娜、

刘钟允、汤子辰、罗子霄、王浩田、魏文定 等

华为云鲲鹏大数据实验体系 4：实践 MapReduce 分布式数据处理

1.1.1. 实践 MapReduce 分布式数据处理

1.1.1.1 实验描述

本实验使用 IDEA 构建大数据工程，通过 Java 语言编写 WordCount 程序并通过集群运行，完成单词计数任务。首先，在本地进行 IDEA 的安装，接着使用 IDEA 构建大数据工程并编写 Wordcount 程序，最后将程序打包在先前实验构建的集群上运行程序。

使用的软件版本：

1. 系统版本：Centos7.5；
2. Hadoop 版本：Apache Hadoop 2.7.7；
3. JDK 版本：1.8.*；
4. IDEA 版本：IDEA2021.2。

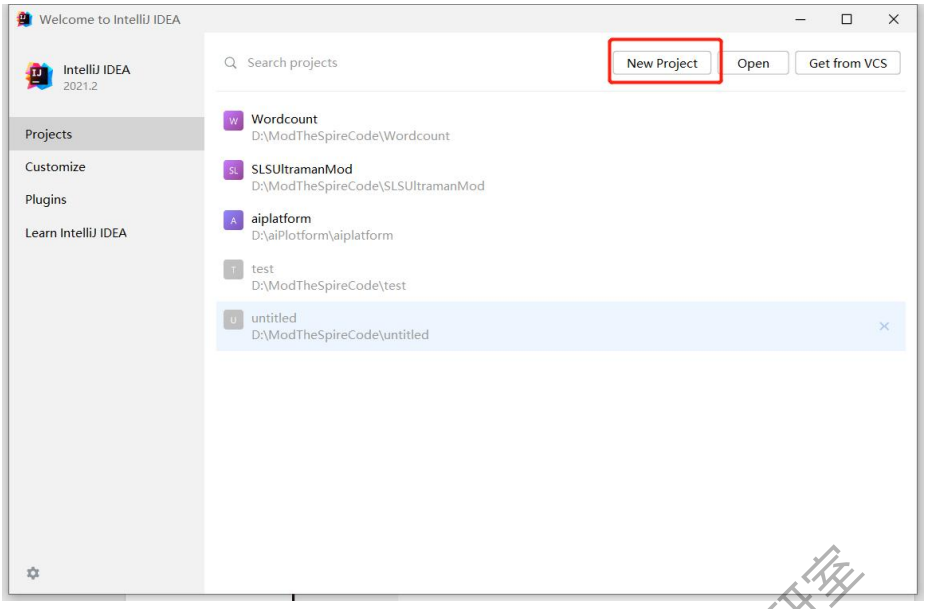
1.1.1.2 实验目的

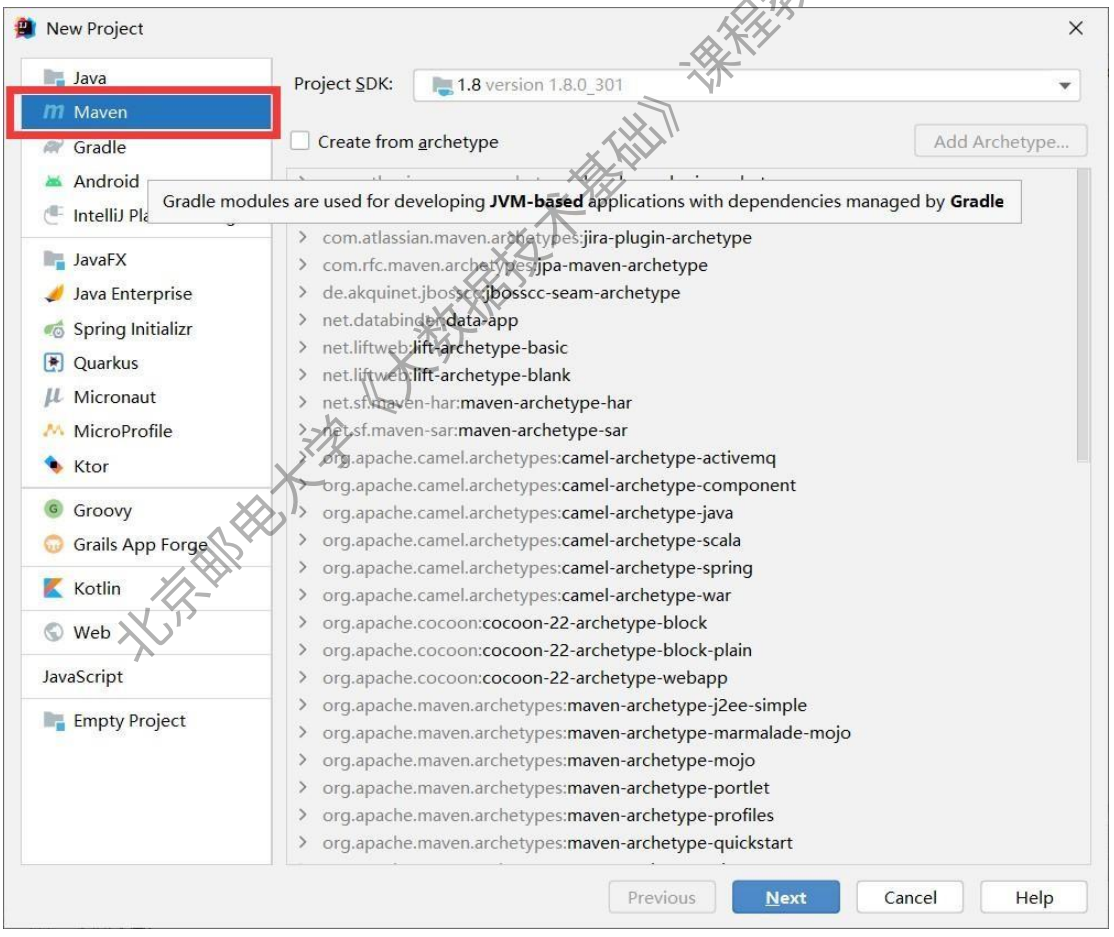
1. 了解 IDEA 构建大数据工程的过程；
2. 熟悉使用 Java 语言编写大数据程序；
3. 了解 MapReduce 的工作原理；
4. 掌握在集群上运行程序的方法。

1.1.1.3 实验步骤

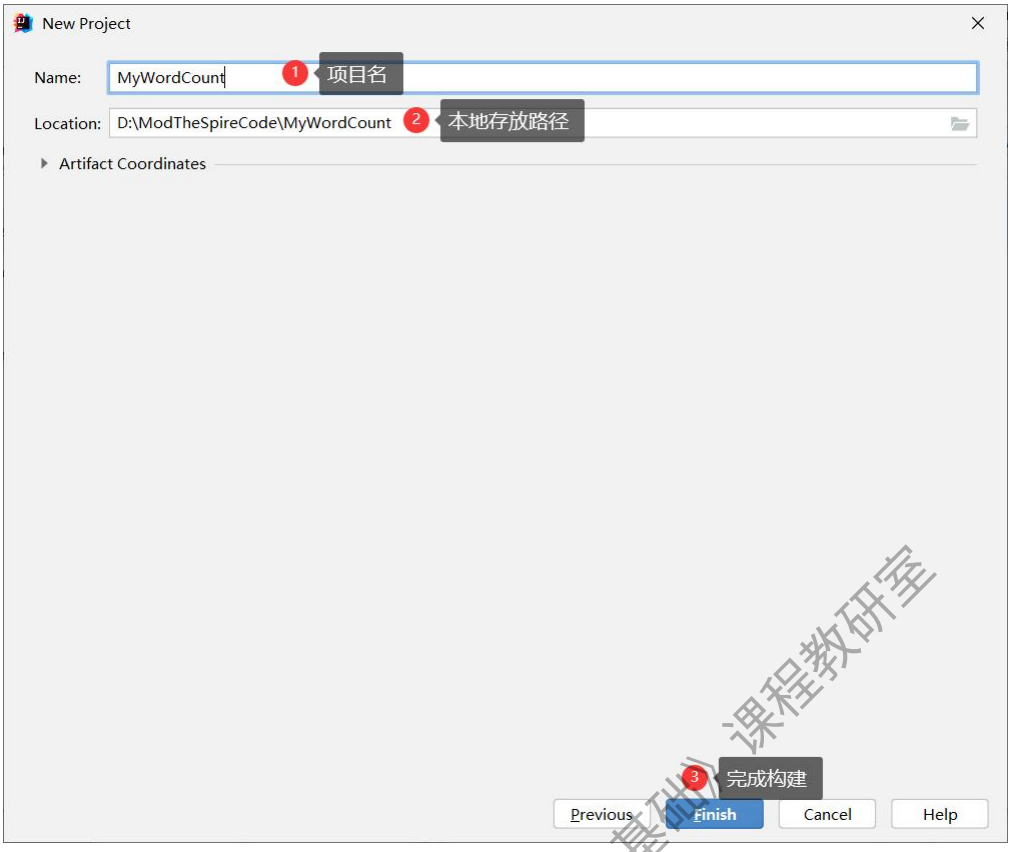
1. IDEA 构建大数据工程

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：

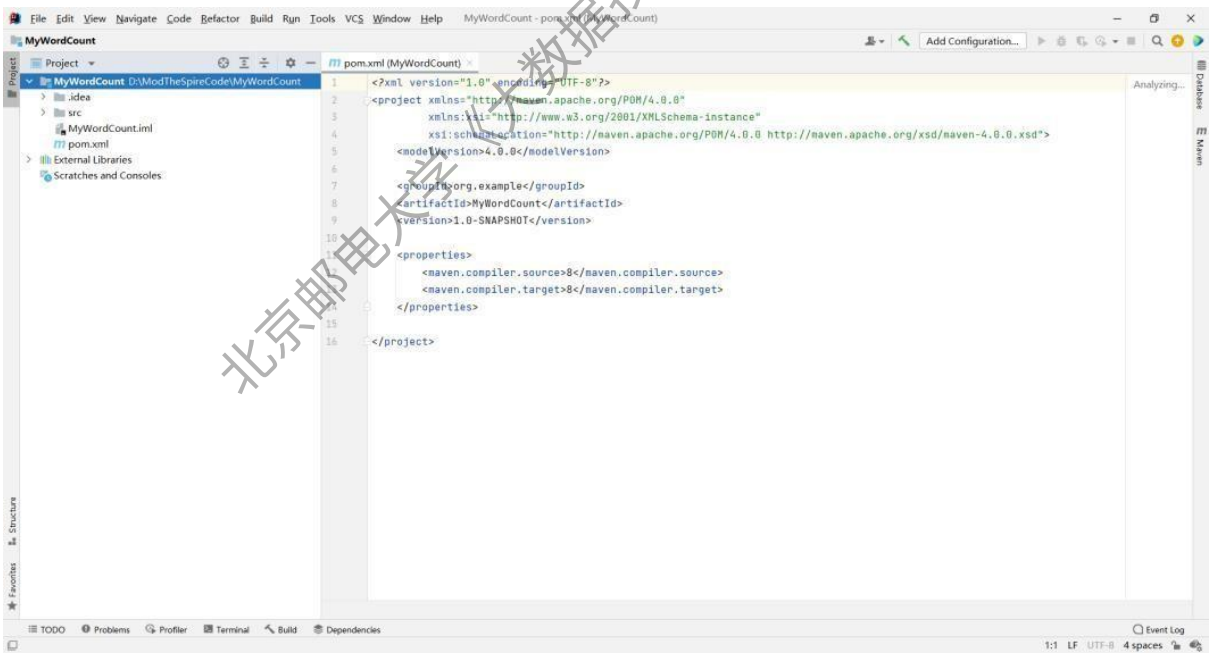




点击New Project，然后点击选择 Maven 项目，点击Next，之后输入 Name，选择Location，并点击Finish。



进入如下界面表示工程创建成功：



步骤 2：依赖设置：

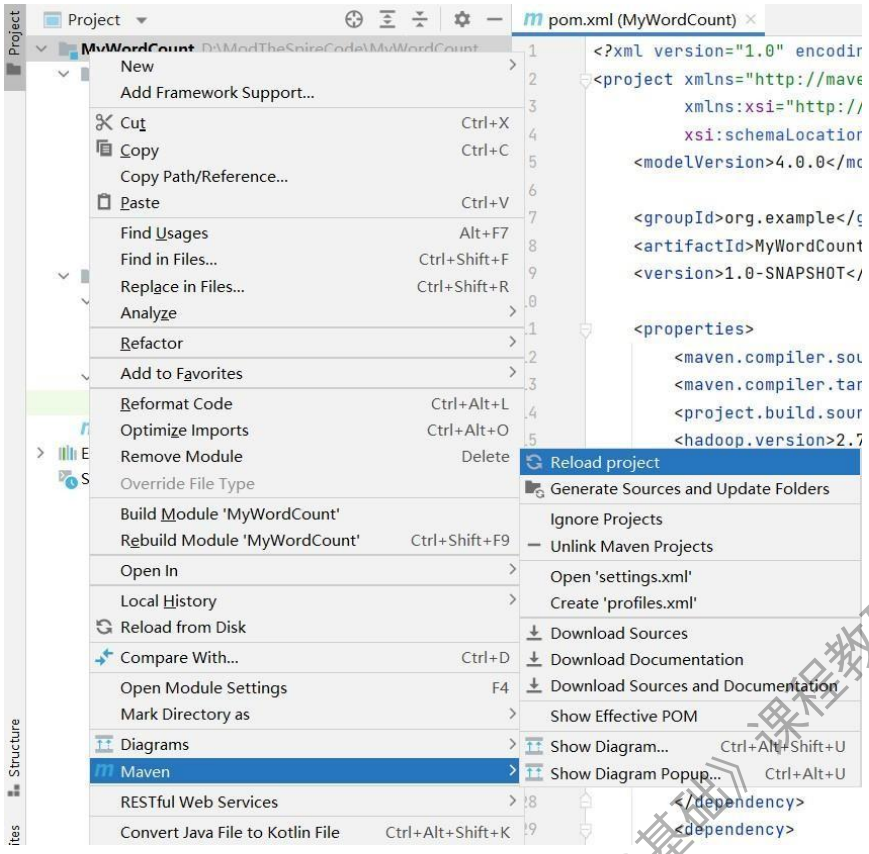
(1) 在 pom.xml 文件中找到 properties 配置项，新增 hadoop 版本号（此处对应 hadoop 安装版本）；

```
<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <hadoop.version>2.7.7</hadoop.version>
</properties>
```

(2) 找到 dependency 配置项（若无则手动添加），添加如下图标红部分的配置，这部分是 hadoop 的依赖，${hadoop.version}表示上述配置的 hadoop.version 变量；

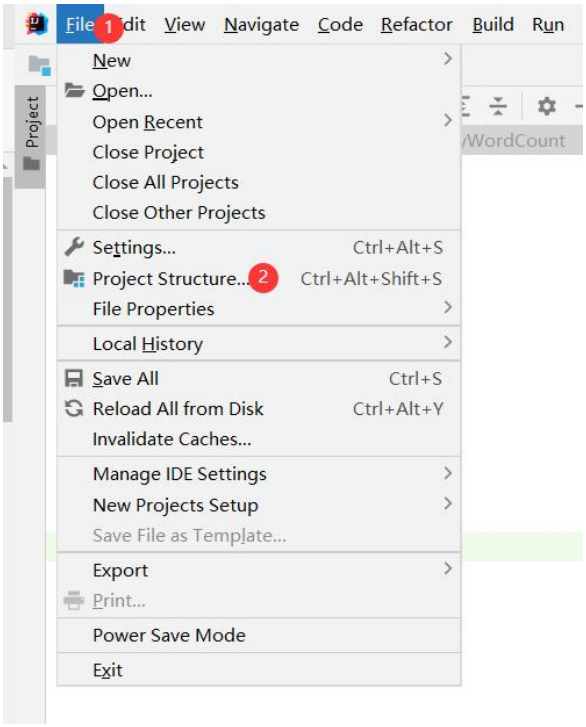
```
<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <hadoop.version>2.7.7</hadoop.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-hdfs</artifactId>
        <version>${hadoop.version}</version>
    </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，则可以右键点击工程名，依次选择 Maven—->Reload project，即可根据 pom.xml 文件导入依赖包；

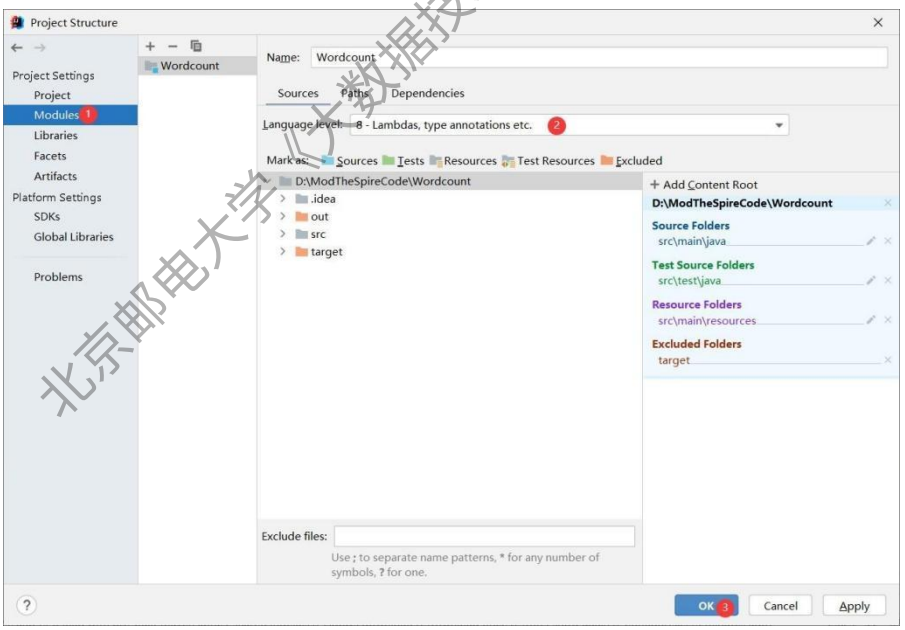


步骤 3：设置语言环境：

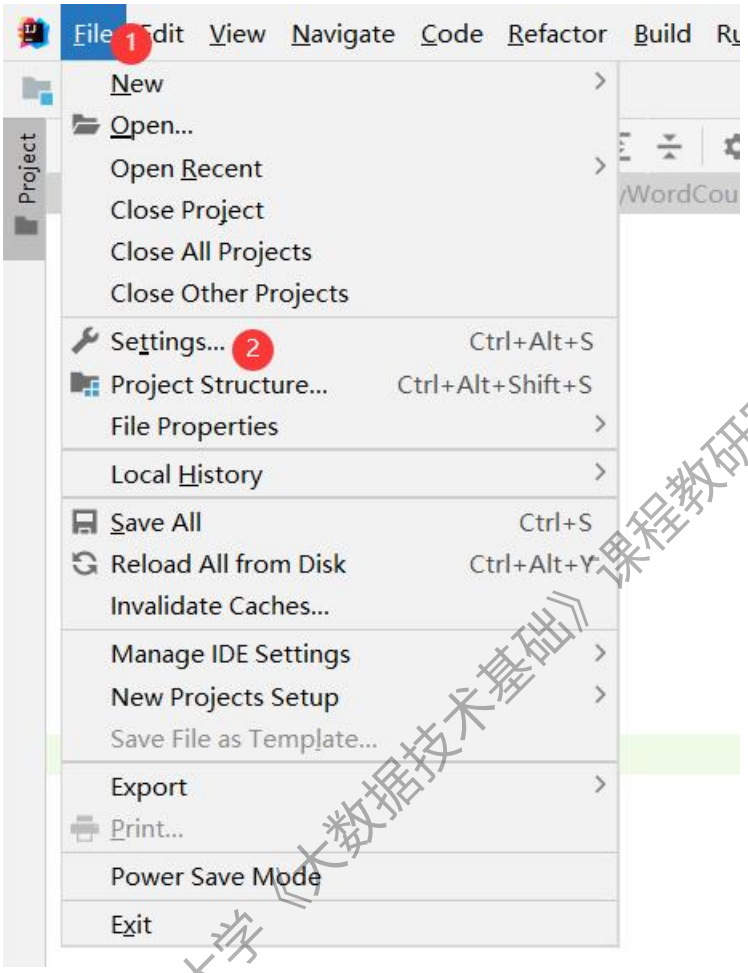
1） 设置语言环境 language level，点击菜单栏中的 file，选择 Project Structure；



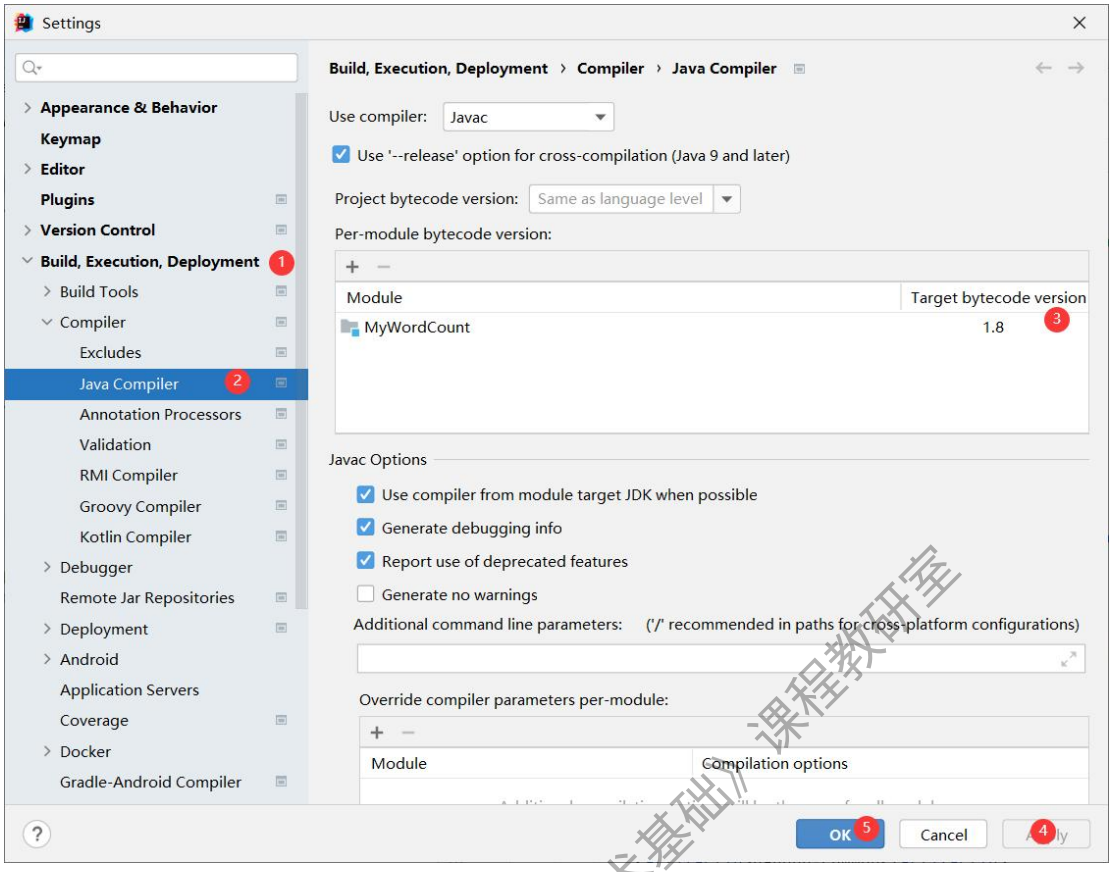
弹出如下对话框，选择 Modules，选择 Language level 为 8，然后点击 Apply，点击 OK；



步骤4： 设置java Compiler 环境：1)点击菜单栏中的file，选择Setting；

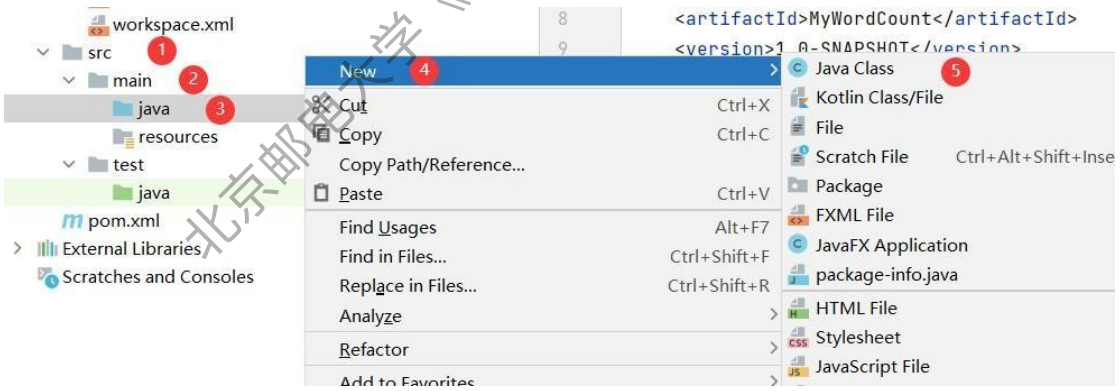


2）弹出如下对话框，依次选择 Build，Execution—->Compiler—->Java Compiler，设置图中的 Project bytecode version 为1.8，设置图中的 Target bytecode version 为1.8，然后依次点击 Apply 和OK；



2. WordCount 程序编写

(1) 如下图依次打开 src—>main—>java，在 java 上点击右键，创建 Java Class；



(2) 弹出如下对话框，输入类名 WordCount，点击ok

(3) 在类WordCount 中添加TokenizerMapper 类，并在该类中实现map 函数；map 函数负责统计输入文件中单词的数量；

```
public class WordCount {

    public static class TokenizerMapper
            extends Mapper<Object, Text, Text, IntWritable>{

        private final static IntWritable one = new IntWritable( value: 1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
```

(4) 在类WordCount 中添加IntSumReducer 类，并在该类中实现reduce 函数；reduce 函数合并之前map 函数统计的结果，并输出最终结果；

```
    public static class IntSumReducer
            extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context
        ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
```

(5) 在类WordCount 中添加main 方法；

```
public static void main(String[] args) throws Exception {

}
```

(6) 创建Configuration 对象，运行MapReduce 程序前都要初始化 Configuration，该类主要是读取 MapReduce 系统配置信息；

```
Configuration conf = new Configuration();
```

(7) 限定输出参数必须为 2 个，If 的语句好理解，就是运行 WordCount 程序时候一定是两个参数，如果不是就会输出错误提示并退出；

```
String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
if (otherArgs.length != 2) {
    System.err.println("Usage: wordcount <in> <out>");
    System.exit( status: 2);
}
```

(8) 创建Job 对象，第一行构建一个job，构建时候有两个参数，一个是 conf，一个是这个 job 的名称。

第二行就是装载程序员编写好的计算程序，例如程序类名就是 WordCount 了。虽然编写 MapReduce 程序只需要实现 Map 函数和 Reduce 函数，但是实际开发要实现三个类，第三个类是为了配置 MapReduce 如何运行 Map 和 Reduce 函数，准确的说就是构建一个 MapReduce 能执行的 job，例如 WordCount 类。

第三行和第五行就是装载 Map 函数和 Reduce 函数实现类了，这里多了个第四行，这个是装载 Combiner 类。

```
Job job = new Job(conf, jobName: "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
```

(9) 定义输出的key/value 的类型，也就是最终存储在HDFS 上结果文件的key/value 的类型。

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

(10) 第一行就是构建输入的数据文件，第二行是构建输出的数据文件，两者均从参数读入。

最后一行如果 job 运行成功了，程序就会正常退出。

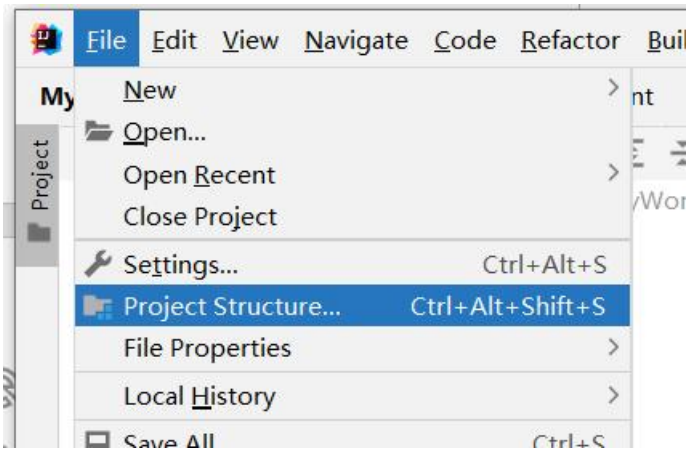
```
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
```

1)完整主类如下：

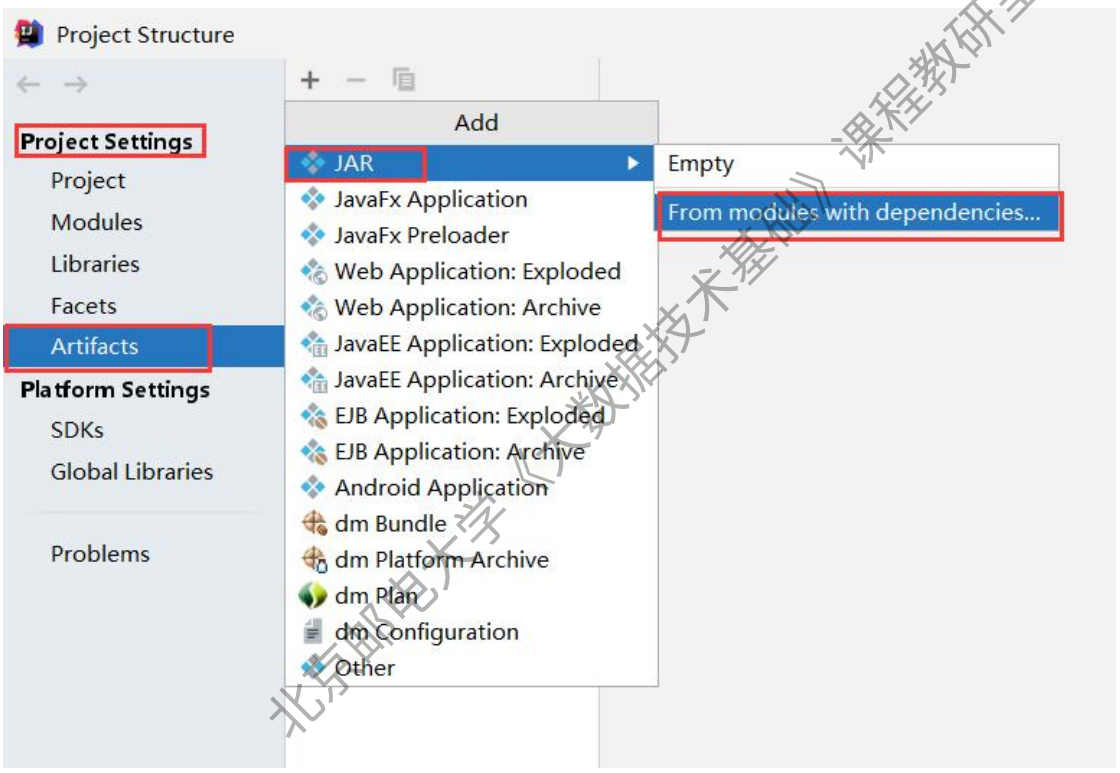
```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: wordcount <in> <out>");
        System.exit( status: 2);
    }
    Job job = new Job(conf,  jobName: "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
}
```


3. 程序打包与运行

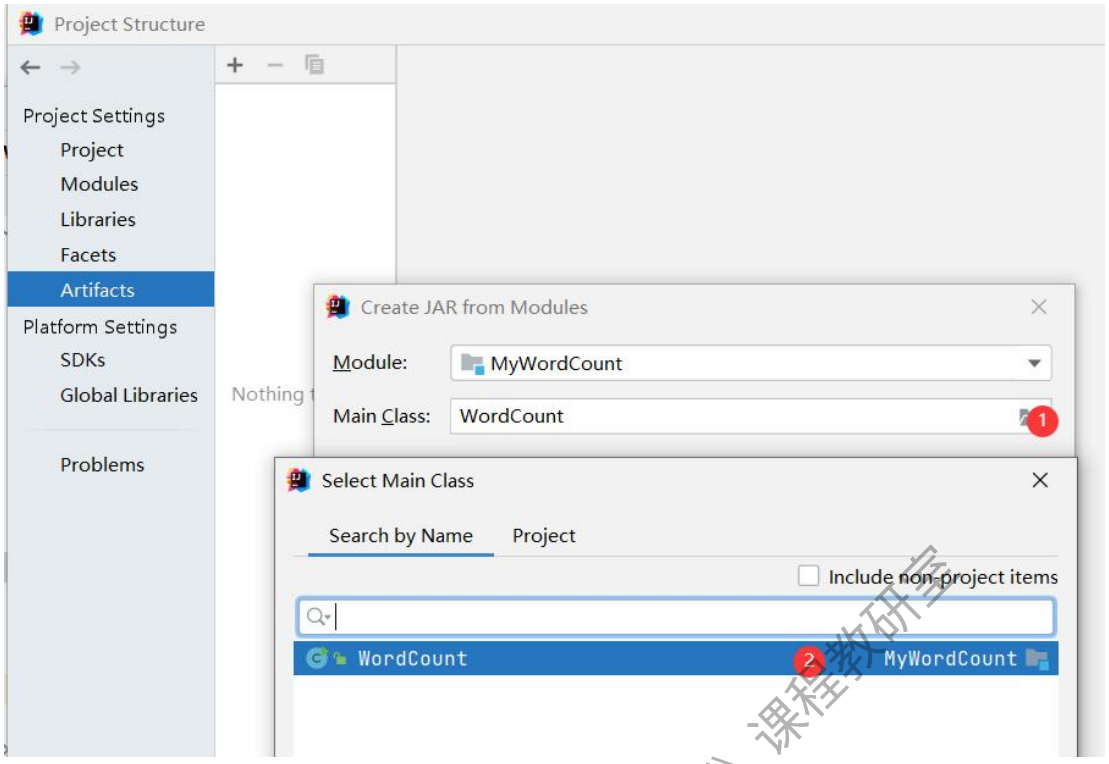
步骤 1：打开 File->ProjectStructure：



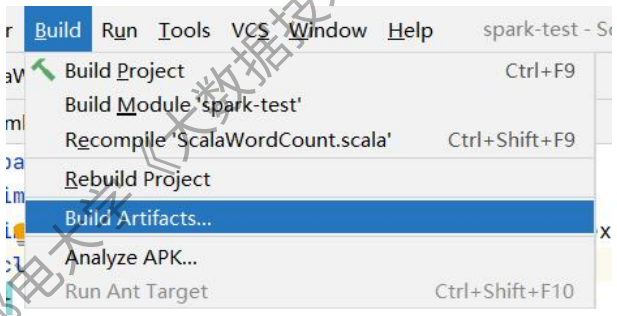
步骤 2：Project Settings 栏下的 Artifacts，点击“+”，选择 JAR->From modules with dependencies…：



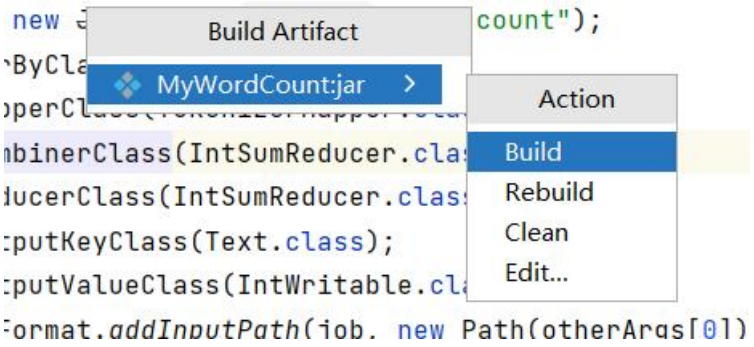
步骤 3：填写主类名称：



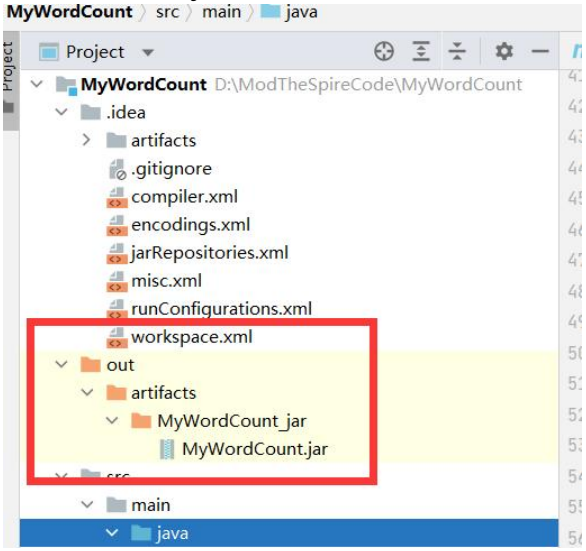
步骤 4：选择 Build->Artifacts：



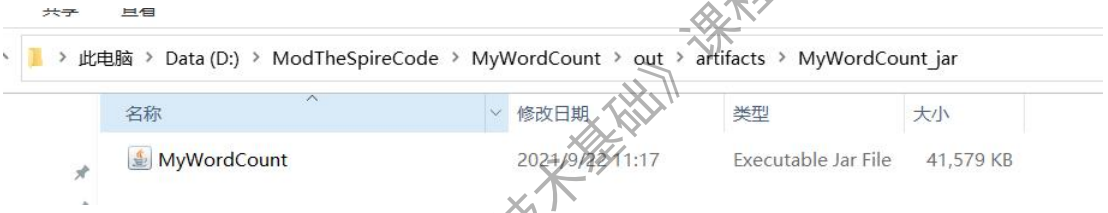
步骤 5：选择 Build：



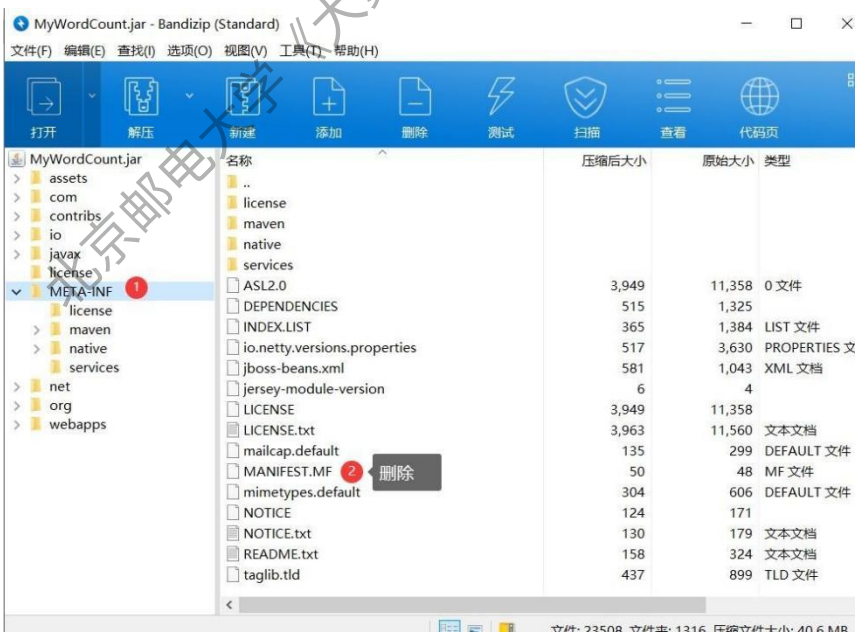
建立完成后会在 out 文件夹下生成 jar 包：



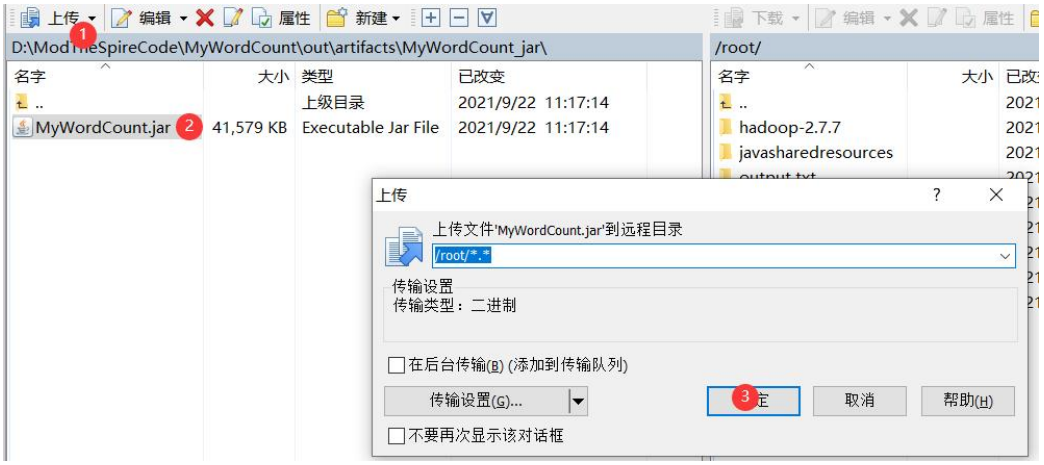
步骤 6：使用压缩软件打开生成的 jar 包（可以去本地电脑里找）：



步骤 7：找到 META-INF 目录，并删除 MANIFEST.MF 文件



步骤 8：使用 WinSCP 上传处理后的 jar 包到服务器：



步骤9：构建输入文件，可在自己电脑构建然后上传到服务器中。格式为学号-姓名简写-input.txt，可通过cat 命令打印检查。

```
[root@2022110892-lny-0001 ~]# cat 2022110892-lny.txt
hello world
hello world
hello world
hadoop spark
hadoop spark
hadoop spark
dog fish
dog fish
dog fish[root@2022110892-lny-0001 ~]#
```

步骤 10：使用“hadoop jar <jar 包名> <主函数> <其余参数>”命令，在 hadoop 运行程序：

注意：输入文件应该放入 hdfs 中，下图中参数路径均为 hdfs 内的路径，文件放入 hdfs 可以参考：https://blog.csdn.net/u014419014/article/details/78056143

首先创建输入文件目录：

```
[root@2022110892-lny-0001 ~]# hadoop fs -mkdir -p /data/wordcount
```

将输入文件放入创建的目录当中：

```
[root@2022110892-lny-0001 ~]# hadoop fs -put 2022110892-lny.txt /data/wordcount
```

```
[root@2022110892-lny-0001 ~]# hadoop fs -ls /data/wordcount
22/09/28 21:19:18 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   3 root supergroup        109 2022-09-23 20:37 /data/wordcount/2022110892-lny.txt
```

创建输出目录：

```
[root@2022110892-lny-0001 ~]# hadoop fs -mkdir -p /output/
```

运行：

```
[root@2022110892-lny-0001 ~]# hadoop jar WordCount.jar WordCount /data/wordcount/2022110892-lny.txt /output/wordcountresult2
```

得到如下图运行结果：

```
        Total megabyte-milliseconds taken by all reduce
    Map-Reduce Framework
            Map input records=9
            Map output records=18
            Map output bytes=174
            Map output materialized bytes=76
            Input split bytes=130
            Combine input records=18
            Combine output records=6
            Reduce input groups=6
            Reduce shuffle bytes=76
            Reduce input records=6
            Reduce output records=6
            Spilled Records=12
            Shuffled Maps =1
            Failed Shuffles=0
            Merged Map outputs=1
            GC time elapsed (ms)=121
            CPU time spent (ms)=850
            Physical memory (bytes) snapshot=355074048
            Virtual memory (bytes) snapshot=2585657344
            Total committed heap usage (bytes)=173670400
    Shuffle Errors
            BAD_ID=0
            CONNECTION=0
            IO_ERROR=0
            WRONG_LENGTH=0
            WRONG_MAP=0
            WRONG_REDUCE=0
    File Input Format Counters
            Bytes Read=109
    File Output Format Counters
            Bytes Written=46
```

步骤 11：在 hdfs 上查看程序的输出：

```
[root@2022110892-lny-0001 ~]#  hadoop fs -text /output/wordcountresult2/part-r-00000
22/09/29 10:33:53 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
dog     3
fish    3
hadoop  3
hello   3
spark   3
world   3
```

1.1.1.4 实验结果与评分标准

实验结束后应得到：wordcount 程序代码打包后的 jar 包，对实验结果及程序代码的解释和分析报告。

完成 MapReduce 分布式数据处理实践。

实验评分标准，提交的实验报告中应包含：

1) wordcount 程序代码打包后的 jar 包及实验分析报告；
2) 完整代码截图；
3) 代码 Maven 构建成功的截图；
4) 在 Hadoop 中提及 jar 的操作截图（需要截图包括服务器名称）；
5) 完成程序运行的结果截图（要求包含程序运行的系统时间）；
6) 通过在 hdfs 上查看程序的输出的截图。

实验中应对各个截图进行简单解释，证明理解截图含义。