实验二：实践 MapReduce 分布式数据处理

一、 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 Wordcount 程序和工程的编写，将程序打包，最后在先前实验构建的集群上运行程序。

二、 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。

三、 实验环境

1. 系统版本：Centos 7.6；Hadoop 版本： 2.10.2；
   或 docker 集群 hadoop 3.3.6
2. JDK 版本：1.8.0*；
3. IDEA 版本：IDEA2023.3.4

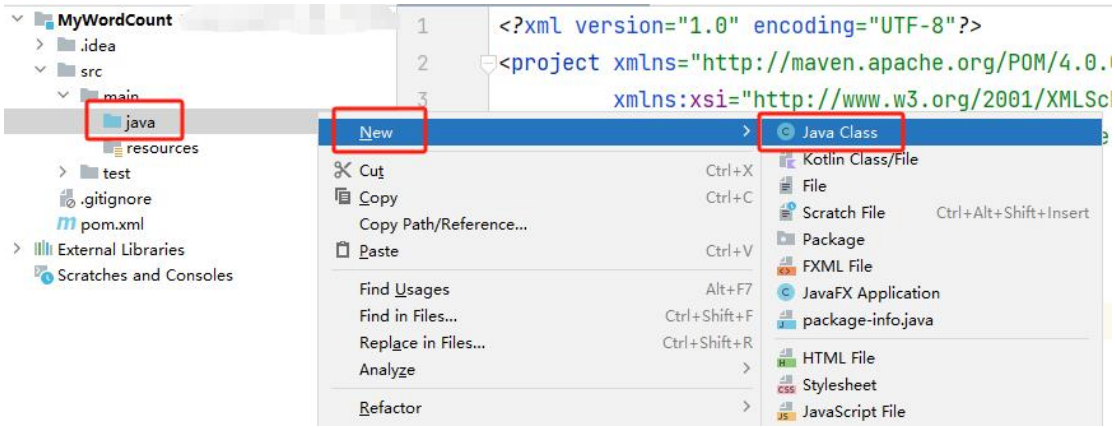
四、 实验步骤

4.1 IDEA 新建工程

创建 Maven 工程，同实验一，新建 MyWordCount 工程，pom.xml 依赖，log4jproperties 等内容和 Maven 设置和实验一保持一致。

4.2 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，回车。

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 {

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

        1 usage
        private final static IntWritable one = new IntWritable( value: 1);
        2 usages
        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);
            }
        }
    }
```

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

private final static IntWritable one = new IntWritable(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> {
    2 usages
    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);
    }
}
```

```
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 {

}
```

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();
```

限定输出参数必须为 2 个，If 的语句好理解，就是运行 WordCount 程序的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);
}
```

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);
```

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);
```

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);
```

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);
}
```


```
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(2);
    }

    Job job = new Job(conf, "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(true) ? 0 :1);
    }
```

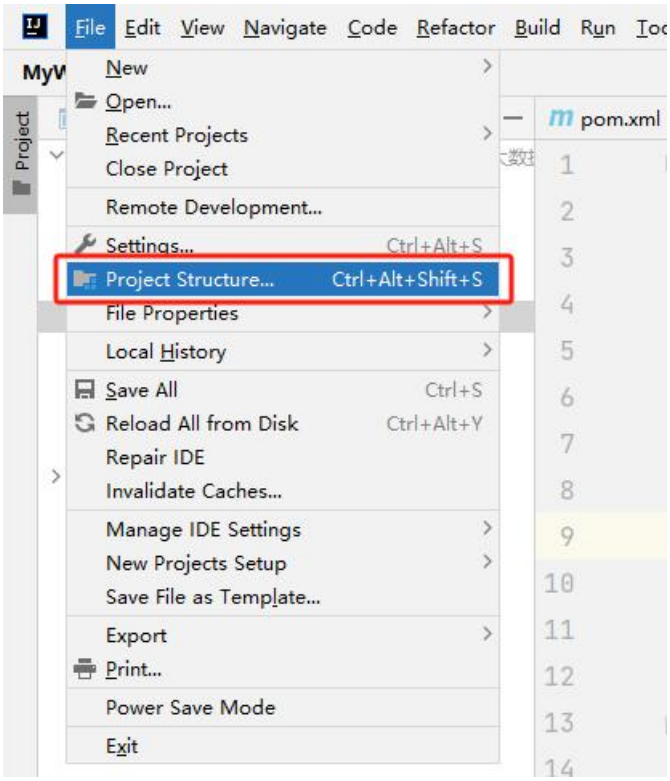
6. 本实验需要导入的包总结如下：

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;
import java.util.StringTokenizer;
```

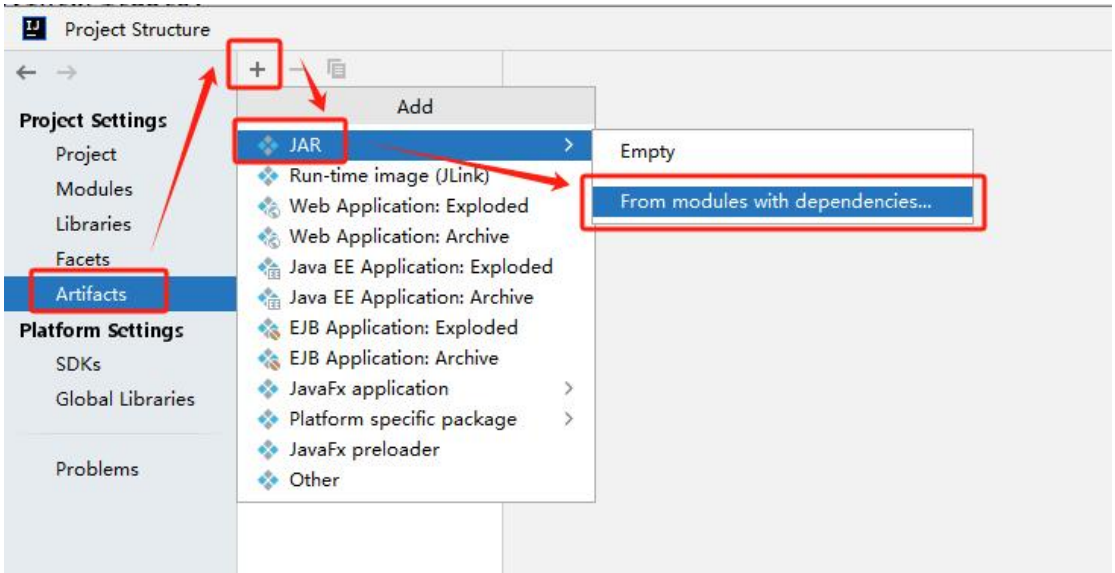

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;
import java.util.StringTokenizer;
```

4.3 程序打包与运行

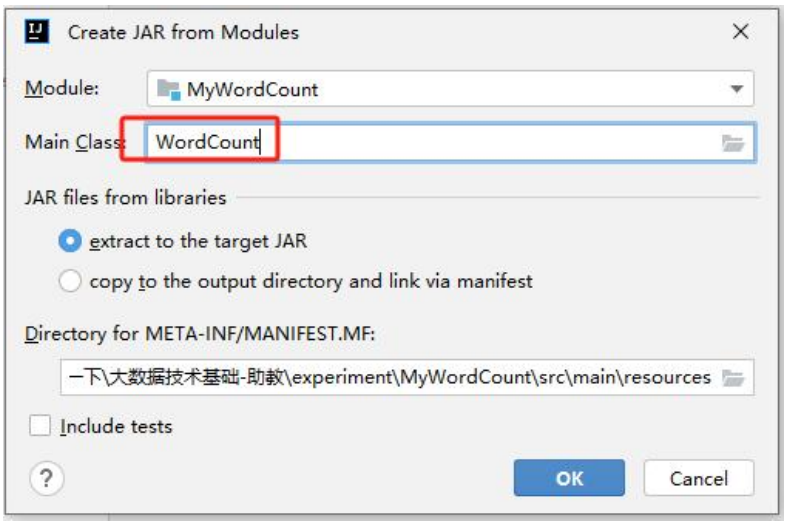
1. 打开 File->Project Structure…

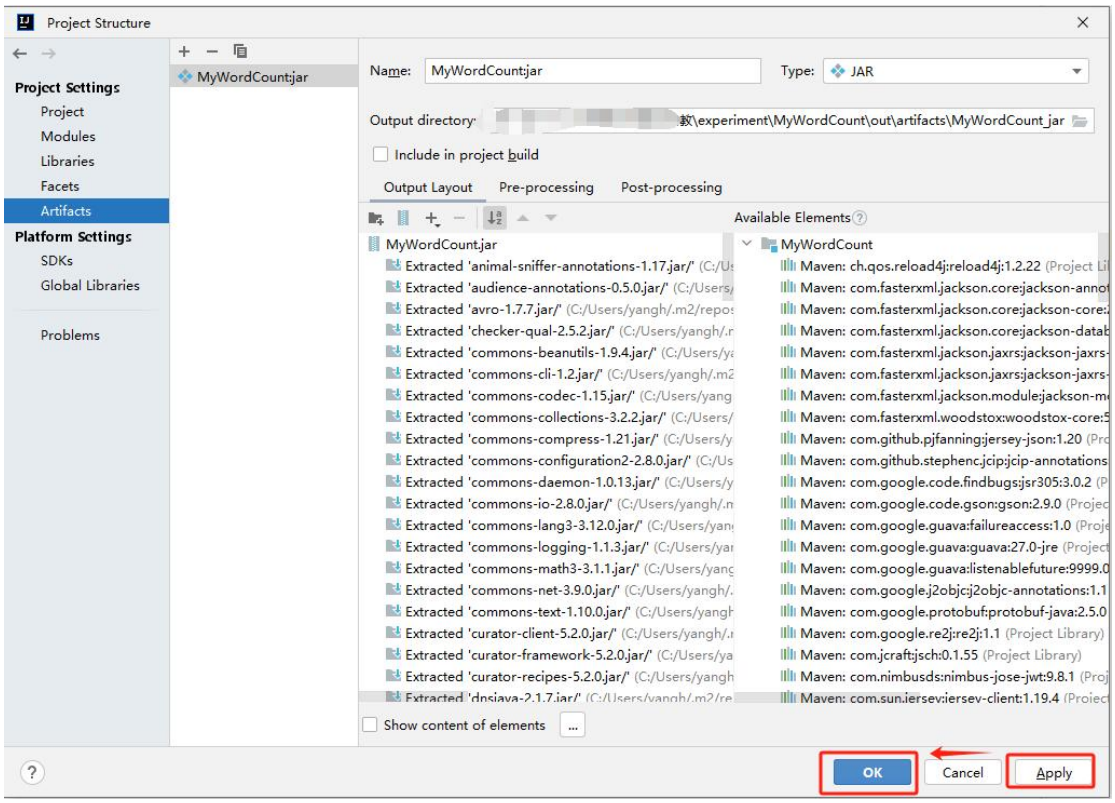


2. 依次点击 Artifacts，加号”+”，JAR，”From modules with dependencies…”

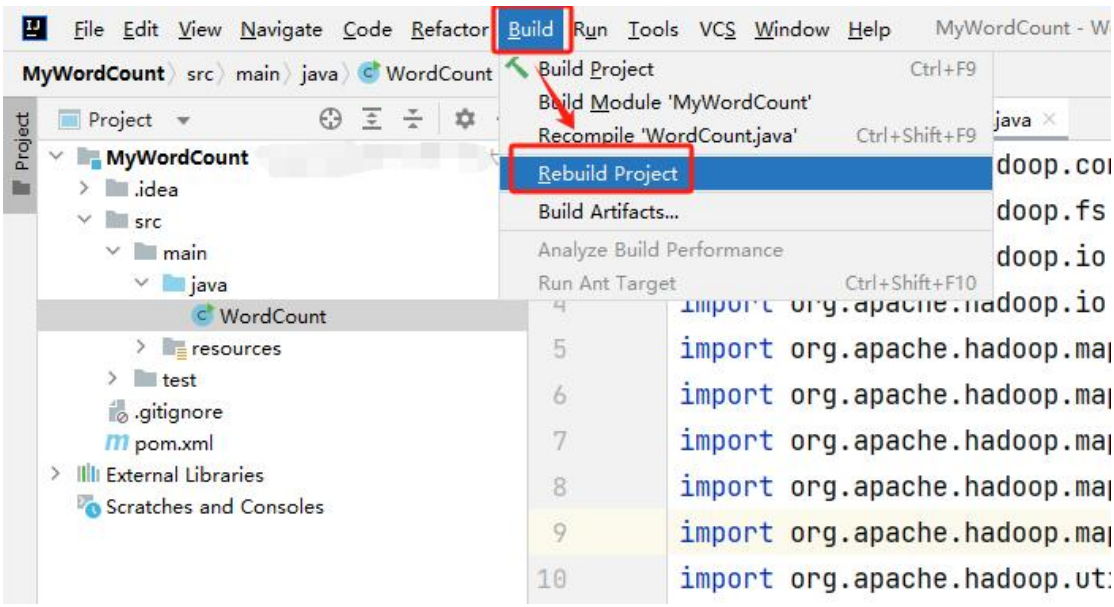


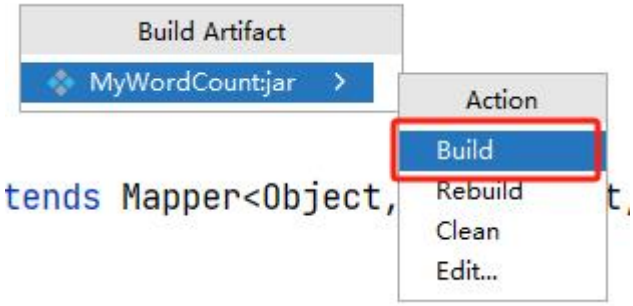
3. 填写主类名称 WordCount，确认后点击”Applay”和”OK”



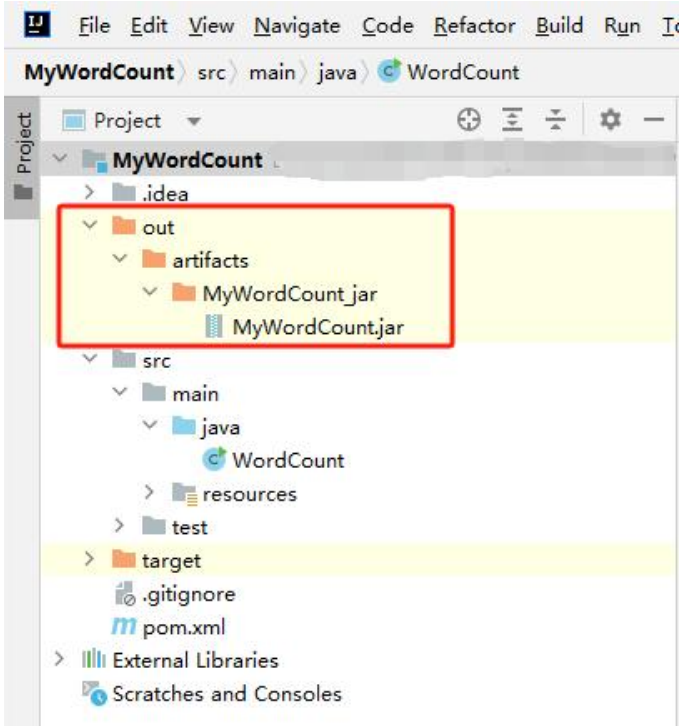


4. 选择 Build->Build Artifacts…->Build





5. 之后会在 out 文件夹下生成 jar 包



6. 使用 SCP 命令上传至服务器或容器（以容器为例）

scp ./MyWordCount.jar root@123.249.66.19:~/

docker cp MyWordCount.jar 10f69568df93:/root/ (容器编号记得修改）

```
[root@zj-2023140696 ~]# docker ps
CONTAINER ID   IMAGE          COMMAND   CREATED       STATUS         PORTS                                          NAMES
10f69568df93   hello-hadoop   "bash"    2 weeks ago   Up 4 minutes   0.0.0.0:8020->9870/tcp, :::8020->9870/tcp      master
cf41d6fc2212   hello-hadoop   "bash"    2 weeks ago   Up 4 minutes                                                  slave2
d7c33340a6d4   hello-hadoop   "bash"    2 weeks ago   Up 4 minutes                                                  slave1
41b9ad02fa46   hello-hadoop   "bash"    2 weeks ago   Up 4 minutes                                                  slave3
[root@zj-2023140696 ~]# docker cp MyWordCount.jar 10f69568df93:/root/
Successfully copied 69.3MB to 10f69568df93:/root/
[root@zj-2023140696 ~]#
```

7. 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，内容如下（不一定一模一样）：

```
root@master: ~

hello world
Hello world
Hello World
hadoop spark
hadoop Spark
hadoop spark
dog fish
dog Fish
Dog fish
~
~
~
~
~
```

8. 运行 jar 包

通过以下命令执行程序，格式为”hadoop jar <jar 包> <input 文件或路径> <output 路径>”

这里可能需要在 HDFS 上创建文件夹：

hdfs dfs -mkdir <文件夹>

需要截图一下过程：

```
                Total committed heap usage (bytes)=286785536
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=40
2024-04-19 07:44:01,457 INFO mapred.LocalJobRunner: Finishing task: attempt_local801457894_0001_r_000000_0
2024-04-19 07:44:01,457 INFO mapred.LocalJobRunner: reduce task executor complete.
2024-04-19 07:44:01,943 INFO mapreduce.Job:  map 100% reduce 100%
2024-04-19 07:44:01,943 INFO mapreduce.Job: Job job_local801457894_0001 completed successfully
2024-04-19 07:44:01,952 INFO mapreduce.Job: Counters: 36
        File System Counters
                FILE: Number of bytes read=138519862
                FILE: Number of bytes written=140875408
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=74
                HDFS: Number of bytes written=40
                HDFS: Number of read operations=15
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=4
                Map output records=6
                Map output bytes=60
                Map output materialized bytes=66
                Input split bytes=115
                Combine input records=6
                Combine output records=5
                Reduce input groups=5
                Reduce shuffle bytes=66
                Reduce input records=5
                Reduce output records=5
                Spilled Records=10
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=6
                Total committed heap usage (bytes)=573046784
        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=37
        File Output Format Counters
                Bytes Written=40
```

若出现以下问题，

```
2024-04-19 07:31:02,430 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-04-19 07:31:02,498 INFO impl.YarnClientImpl: Submitted application application_1713509381271_0002
2024-04-19 07:31:02,537 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1713509381271_0002/
2024-04-19 07:31:02,537 INFO mapreduce.Job: Running job: job_1713509381271_0002
2024-04-19 07:37:03,597 INFO mapreduce.Job: Job job_1713509381271_0002 running in uber mode : false
2024-04-19 07:37:03,600 INFO mapreduce.Job:  map 0% reduce 0%
```

参考：https://blog.csdn.net/u014646662/article/details/82890443

9. 运行结果（使用命令查看）

```
root@master:~# hadoop fs -cat /test/output/part-r-00000
2024-04-19 07:45:54,103 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
HELLO   1
WORLD   1
World   1
hello   2
world   1
root@master:~#
```

五、实验结果与分析

1. 粘贴 4.3 jar 包生成图（1 分）
2. 粘贴 4.3 执行命令截图（1 分，要求包含学号、姓名简写信息）
3. 粘贴 4.3 步骤 8 执行命令结果图（2 分）
4. 粘贴 4.3 步骤 9 执行命令结果图（2 分）
5. 提交 jar 包（jar 包无法直接提交可以打包到压缩包提交）（3 分）
6. 解释 wordcount 程序代码（3 分），3 个类的作用（1 分），类之间的关系（1 分)

7. 实验中应对各个截图进行简单解释，证明理解截图含义（1 分）

附录：可能出现的 BUG

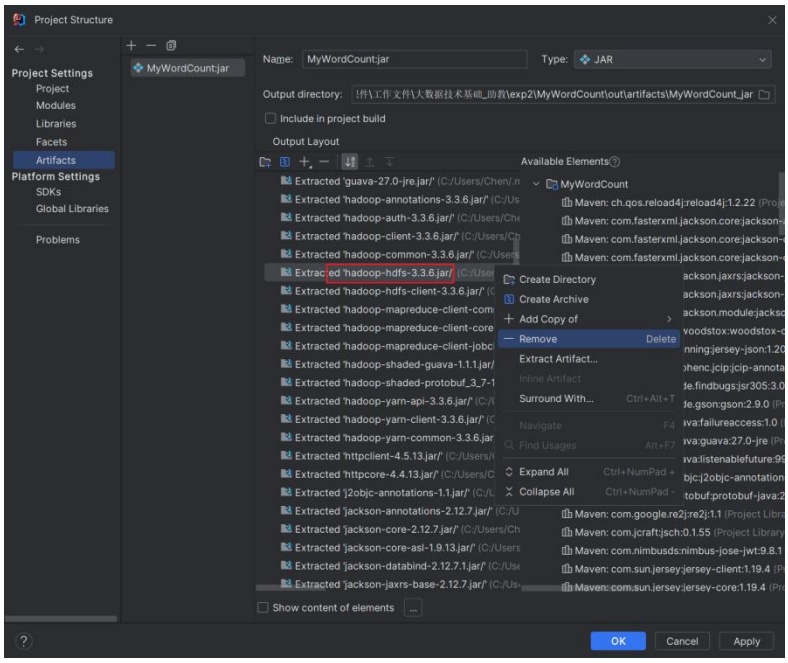
若执行出错，可修改 java 代码，打印程序收到的命令行参数，查看 input 和 output 设置的参数是否正确，例如

```java
        job.setOutputValueClass(IntWritable.class);
        if (otherArgs.length == 2) {
            System.out.println(otherArgs[0]);
            System.out.println(otherArgs[1]);
            FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
            FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
            System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
        } else if (otherArgs.length == 3 ) {
            System.out.println(otherArgs[0]);
            System.out.println(otherArgs[1]);
            System.out.println(otherArgs[2]);

            FileInputFormat.addInputPath(job, new Path(otherArgs[1]));
            FileOutputFormat.setOutputPath(job, new Path(otherArgs[2]));
            System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
        } else {
```

当使用 docker 时若出现如下错误：

```
root@master: ~   root@slave1: ~   root@slave2: ~   root@slave3: ~   Windows PowerSh

        File Input Format Counters
                Bytes Read=36
        File Output Format Counters
                Bytes Written=16
root@master:~# hadoop jar MyWordCount.jar WordCount /input/file /output/wordcount2
Num of other Args: 3
WordCount
/input/file
/output/wordcount2
Exception in thread "main" java.lang.IllegalAccessError: class org.apache.hadoop.hdfs.web.HftpFileSystem cannot access i
ts superinterface org.apache.hadoop.hdfs.web.TokenAspect$TokenManagementDelegator
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:756)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:473)
        at java.net.URLClassLoader.access$100(URLClassLoader.java:74)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:369)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:363)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:362)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:348)
        at java.util.ServiceLoader$LazyIterator.nextService(ServiceLoader.java:370)
        at java.util.ServiceLoader$LazyIterator.next(ServiceLoader.java:404)
        at java.util.ServiceLoader$1.next(ServiceLoader.java:480)
        at org.apache.hadoop.fs.FileSystem.loadFileSystems(FileSystem.java:3521)
        at org.apache.hadoop.fs.FileSystem.getFileSystemClass(FileSystem.java:3566)
        at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:3608)
```

请移除 hadoop-hdfs-3.3.6.jar 包后重新构建


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
MyWordCount:jar
Name: MyWordCount:jar
Type: JAR
Output directory: !件\工作文件\大数据技术基础_助教\exp2\MyWordCount\out\artifacts\MyWordCount_jar
Include in project build
Output Layout
Available Elements
MyWordCount
Maven: ch.qos.reload4j:reload4j:1.2.22
Maven: com.jcraft:jsch:0.1.55
Maven: com.nimbusds:nimbus-jose-jwt:9.8.1
Maven: com.sun.jersey:jersey-client:1.19.4
Create Directory
Create Archive
Add Copy of
Remove Delete
Extract Artifact...
Inline Artifact
Surround With... Ctrl+Alt+T
Navigate F4
Find Usages Alt+F7
Expand All Ctrl+NumPad +
Collapse All Ctrl+NumPad -
Show content of elements
OK
Cancel
Apply