**第7章  MapReduce应用程序编写说明**

**1.实验步骤**

1.1  **启动Hadoop集群**

**在模板中，我们已经配置好了Hadoop伪分布式环境，同学们不需要再次配置，可以直接启动使用。**

**步骤1.**  **启动Hadoop**

打开一个终端模拟器，通过下列命令启动Hadoop：

start-all.sh

**步骤2.**  **验证Hadoop是否启动成功**

通过下列命令，查看相应的JVM进程确定Hadoop是否启动成功：

jps

当有以下5个进程启动时，则说明Hadoop已经成功启动。

1.2  **项目工程创建**

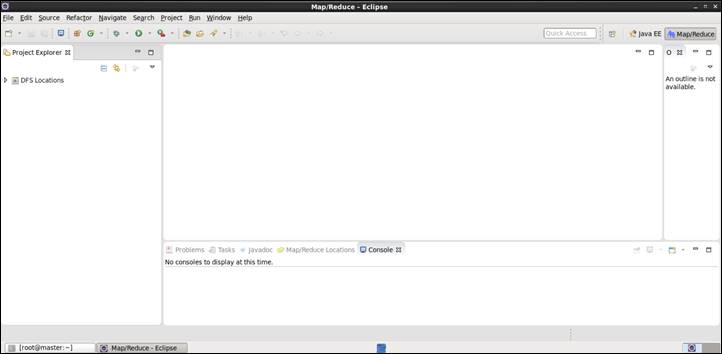
**步骤1.**  **创建工程**

1.   双击桌面上的Eclipse软件图标，进入开发界面，当提示选择工作目录时，选择使用默认的工作目录/root/workspace即可：



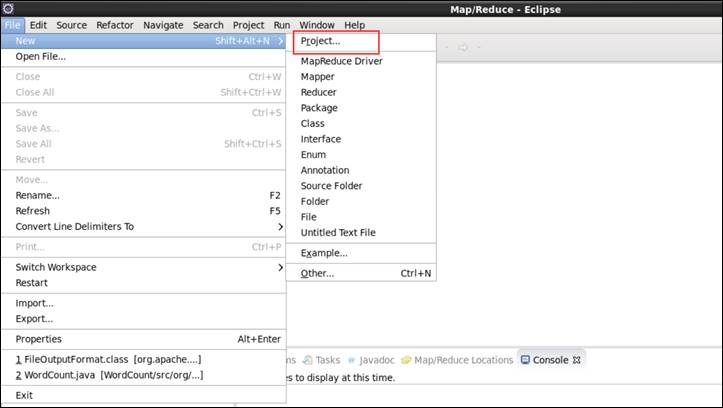
**图 1.4 工作目录选择**

2.   Eclipse主界面如下所示：



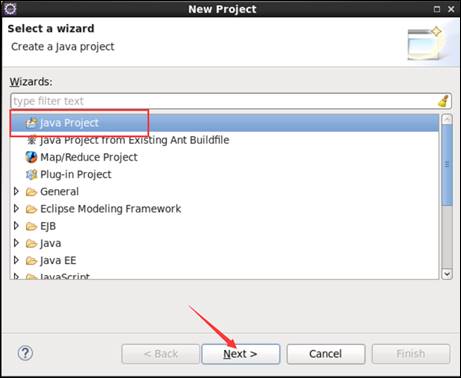
**图 1.5 Eclipse主界面**

3.   创建一个“Project”，操作步骤如下所示：



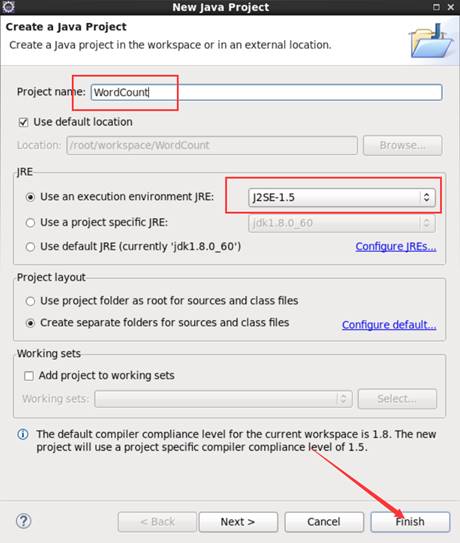
**图 1.6 创建项目**

4.   当选择项目类型时，我们选择“Java Project”：



**图 1.7 选择项目类型**

5.   然后将项目命名为“WordCount”，同时配置运行环境选项，在此我们选择“J2SE-1.5”，然后点击“Finish”完成创建：



**图 1.8 项目命名**

6.   当出现提示框时，点击“Yes”即可：



**图 1.9 提示框选择**

**步骤2.**  **依赖包导入**

在创建的工程中导入项目所需要的hadoop jar包，需要导入的hadoop安装目录下的jar文件包括:

/usr/cx/hadoop-2.7.1/share/hadoop/common/\*.jar

/usr/cx/hadoop-2.7.1/share/hadoop/common/lib/\*.jar

/usr/cx/hadoop-2.7.1/share/hadoop/hdfs/\*.jar

/usr/cx/hadoop-2.7.1/share/hadoop/hdfs/lib/\*.jar

/usr/cx/hadoop-2.7.1/share/hadoop/mapreduce/\*.jar

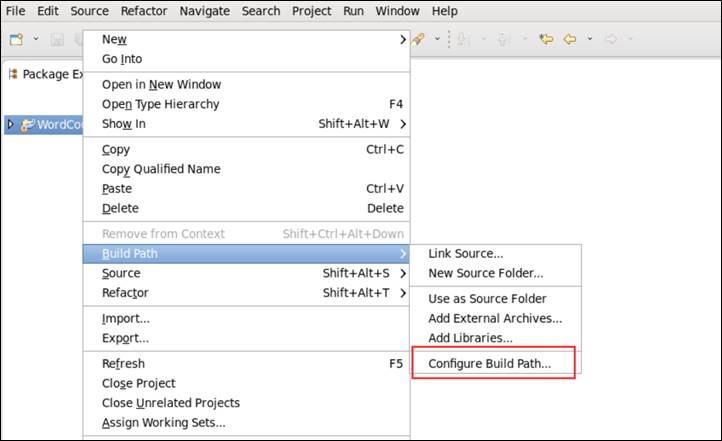
/usr/cx/hadoop-2.7.1/share/hadoop/mapreduce/lib/\*.jar

/usr/cx/hadoop-2.7.1/share/hadoop/tools/\*.jar

/usr/cx/hadoop-2.7.1/share/hadoop/yarn/\*.jar

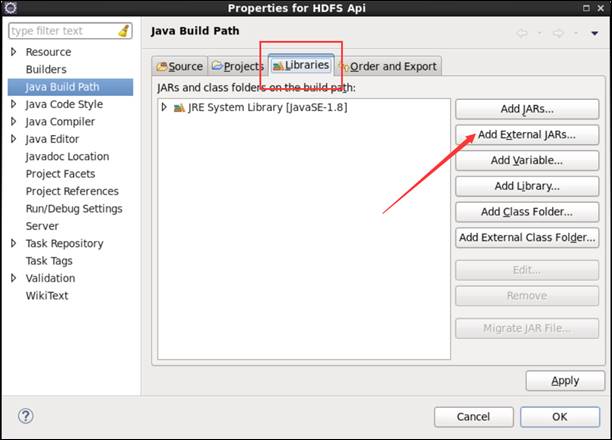
/usr/cx/hadoop-2.7.1/share/hadoop/yarn/lib/\*.jar

1.   将鼠标移动到项目名称上，然后点击鼠标右键，在弹出的菜单中选择“Configure Build Path”：



**图 1.10 配置项目**

2.   在弹出的界面中选择“Libraries”菜单，然后点击“Add External JARs”按钮，进行相关jar文件增加：



**图 1.11 添加依赖包**

3.   增加/usr/cx/hadoop-2.7.1/share/hadoop/common/目录下所有以.jar结尾的文件：



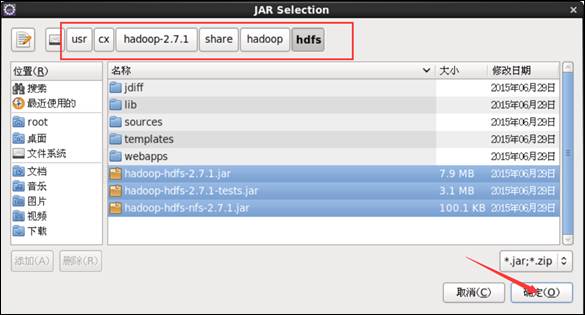
**图 1.12 common依赖包添加**

4.   按照上述的方式，增加/usr/cx/hadoop-2.7.1/share/hadoop/common/lib目录下所有以.jar结尾的文件：



**图 1.13 common/lib依赖包添加**

5.   按照上述的方式，增加/usr/cx/hadoop-2.7.1/share/hadoop/hdfs目录下所有以.jar结尾的文件：



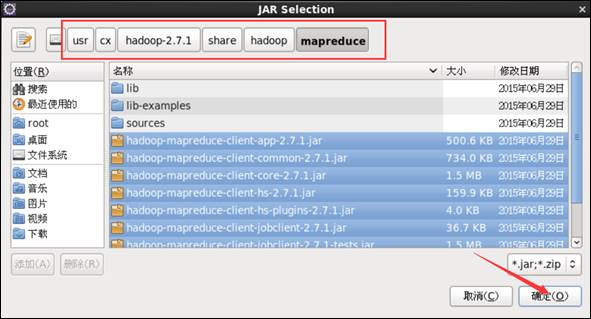
**图 1.14 hdfs依赖包添加**

6.   按照上述的方式，增加/usr/cx/hadoop-2.7.1/share/hadoop/hdfs/lib目录下所有以.jar结尾的文件：



**图 1.15 hdfs/lib依赖包添加**

7.   按照上述的方式，增加/usr/cx/hadoop-2.7.1/share/hadoop/mapreduce目录下所有以.jar结尾的文件：



**图 1.16 mapreduce依赖包添加**

8.   按照上述的方式，增加/usr/cx/hadoop-2.7.1/share/hadoop/mapreduce/lib目录下所有以.jar结尾的文件：



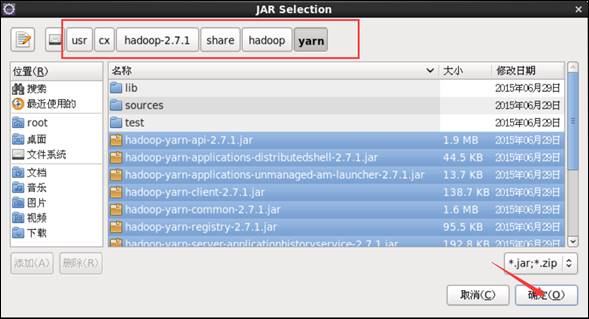
**图 1.17 mapreduce/lib依赖包添加**

9.   按照上述的方式，增加/usr/cx/hadoop-2.7.1/share/hadoop/tools/lib目录下所有以.jar结尾的文件：



**图 1.18 tools/lib依赖包添加**

10.  按照上述的方式，增加/usr/cx/hadoop-2.7.1/share/hadoop/yarn目录下所有以.jar结尾的文件：



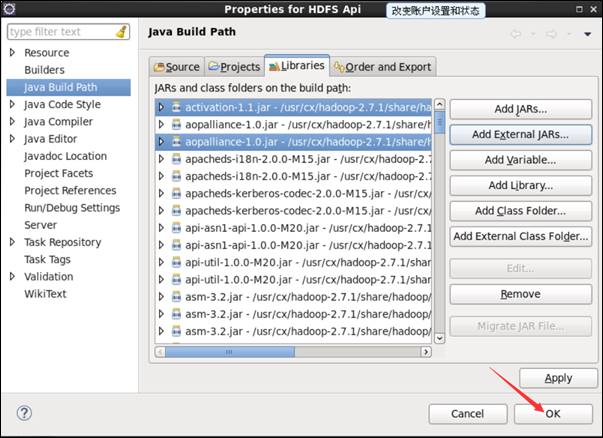
**图 1.19 yarn依赖包添加**

11.  按照上述的方式，增加/usr/cx/hadoop-2.7.1/share/hadoop/yarn/lib目录下所有以.jar结尾的文件



**图 1.20 yarn/lib依赖包添加**

12.  上述文件添加完毕后，点击“OK”按钮，完成添加操作：



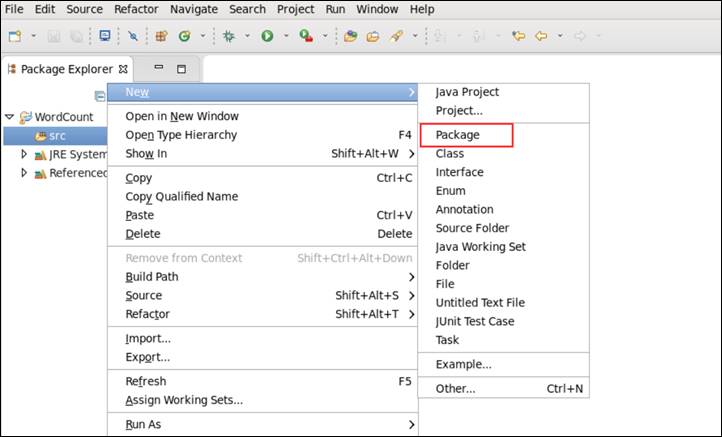
**图 1.21 完成依赖包添加**

以上路径下的jar包必须导入，如果没有正确导入，程序将无法正常运行。

**步骤3.**  **Package创建**

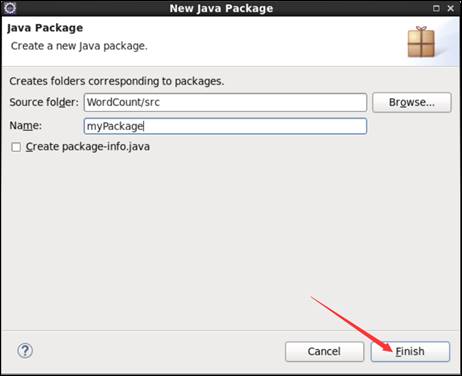
依赖包导入完毕后，需要在项目中创建一个Package，后续只需要针对于对应的实验，在此Package下增加类文件实现HDFS的增删改查操作即可。

1.   将鼠标移动到项目对应的src目录上，然后点击鼠标右键，在弹出的菜单中点击“Package”，如下图所示：



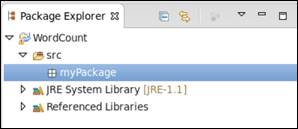
**图 1.22 添加Package**

2.   将创建的Package命名为“myPackage”，然后点击“Finish”：



**图 1.23 Package信息填写**

3.   创建完成后的项目结构如下图所示：

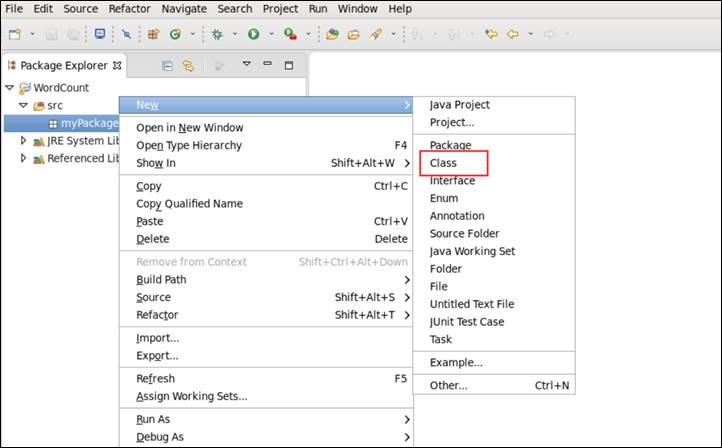


**图 1.24 项目整体结构**

1.3  **工程类文件编写**

**步骤1.**  **添加类文件**

1.   将鼠标移动到上文创建的“myPackage”上，点击鼠标右键增加一个类（Class）文件，如下图所示：



**图 1.25 添加类文件**

2.   将文件命名为“WordCountMain”，并选择“主函数类型”，然后点击“Finish”完成创建，如下图所示：



**图 1.26 详细信息填写**

**步骤2.**  **程序编写**

在创建的文件中写入下列代码：

package myPackage;

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;

public class WordCountMain {

    //继承Mapper接口，设置map的输入类型为<Object, Text>;输出类型为<Text, IntWritable>

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

         //one 表示单词出现一次

         private final static IntWritable one = new IntWritable(1);

         //word用于存储切下的单词

         private Text word = new Text();

         public void map(Object key, Text value, Context context) throws IOException, InterruptedException{

             StringTokenizer itr = new StringTokenizer(value.toString());//对输入的行切词

             while(itr.hasMoreTokens()){

                  //切下的单词存入word

                  word.set(itr.nextToken());

                  context.write(word, one);

             }

         }

    }

    //继承Reducer接口，设置Reduce的输入类型为<Text, IntWritable>,输出类型为<Text, IntWritable>

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

         //result记录单词的频数

         private IntWritable result = new IntWritable();

         public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException{

             int sum = 0;

             //对获取的<key, value-list>计算value的和

             for(IntWritable val : values){

                  sum += val.get();

             }

             //将频数设置到result中

             result.set(sum);

             //收集结果

             context.write(key, result);

         }

    }

    public static void main(String[] args) throws IllegalArgumentException, IOException, ClassNotFoundException, InterruptedException {

         // TODO Auto-generated method stub

         Configuration conf = new Configuration();

         conf.set("fs.defaultFS","hdfs://master:9000");

         //配置作业名

         Job job = null;

         try {

             job = new Job(conf, "word count");

         } catch (IOException e) {

             // TODO Auto-generated catch block

             e.printStackTrace();

         }

         //配置作业的各个类

         job.setJarByClass(WordCountMain.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(args[0]));

         FileOutputFormat.setOutputPath(job, new Path(args[1]));

         if (job.waitForCompletion(true)) {

             System.out.println("mapreduce success!");

         }

    }

}

WordCount程序在Map阶段接收输入的<key, value>(key是当前输入的行号，value是对应行的内容)，然后对此行内容进行切词，每切下一个词就将其组织成<word, 1>的形式输出，表示word出现了一次。

在Reduce阶段，会接收到<word,{1,1,1,1…}>形式的数据，也就是特定单词及其出现次数的情况，其中“1”表示word的频数。所以Reduce每接受一个<word,{1,1,1,1…}>,就会在word的频数上加l，最后组织成<word,sum>的形式直接输出。

1.4  **程序测试**

**步骤1.**  **输入数据源上传**

1.   打开终端模拟器，通过下面的命令在HDFS中创建工作目录/WordCount：

hadoop fs -mkdir /WordCount

2.   通过下面的命令可以查看工作目录是否创建成功：

hadoop fs -ls /

3.   通过下面的命令查看输入数据源文件的前5行内容（我们以Hadoop的NameNode日志文件作为输入源来进行测试）:

head -n 5 /usr/cx/hadoop-2.7.1/logs/hadoop-root-namenode-master.out

4.   通过下列命令将输入源文件上传到HDFS中：

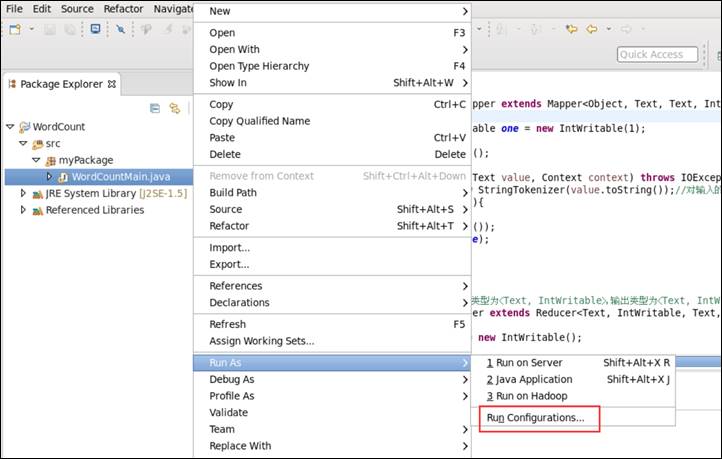
hadoop fs -put /usr/cx/hadoop-2.7.1/logs/hadoop-root-namenode-master.out /WordCount/

5.   通过下列命令可以查看文件是否上传成功：

hadoop fs -ls /WordCount

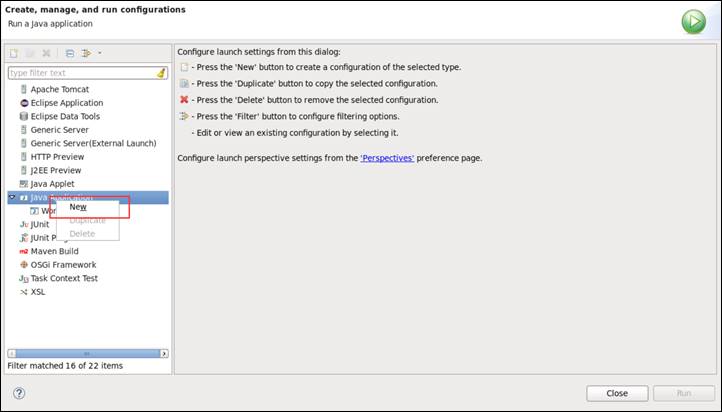
**步骤2.**  **程序测试运行**

1.   将鼠标移动到类文件WordCountMain上，点击鼠标右键，进行运行环境配置，如下图所示：



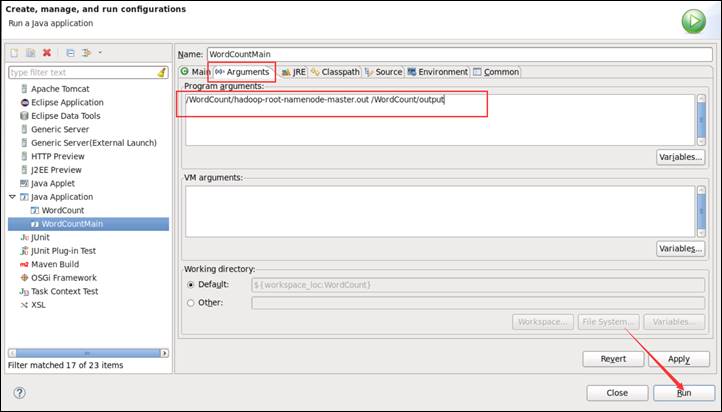
**图 1.27 环境配置**

2.   在“Java Application”选项上点击鼠标右键，选择“New”，此时会自动显示出我们所创建的WordCountMain，如下图所示：



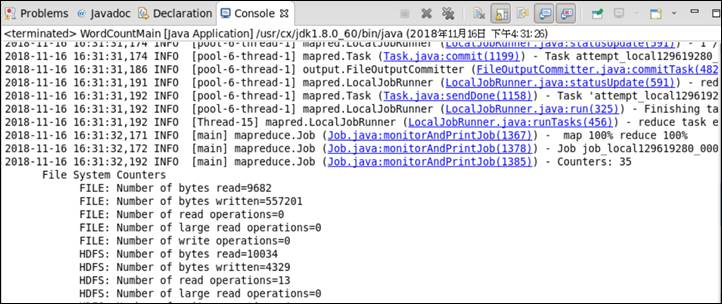
**图 1.28 程序识别**

3.   在“Arguments”中添加输入文件的绝对路径以及结果输出目录（中间用空格分隔），其中输入文件的绝对路径为/WordCount/hadoop-root-namenode-master.out，结果输出目录为/WordCount/output，然后点击“Run”按钮：



**图 1.29 参数配置**

4.   在主界面的“Console”子菜单中，可以实时查看到程序的运行结果：



**图 1.30 程序运行结果**

5.   打开终端模拟器，通过下面的命令在HDFS中查看结果输出目录是否生成：

hadoop fs -ls /WordCount

6.   MapReduce默认会将结果存储到生成目录下的part-r-00000文件中，可以通过下面的命令进一步查看所生成的文件的内容：

hadoop fs -cat /WordCount/output/part-r-00000

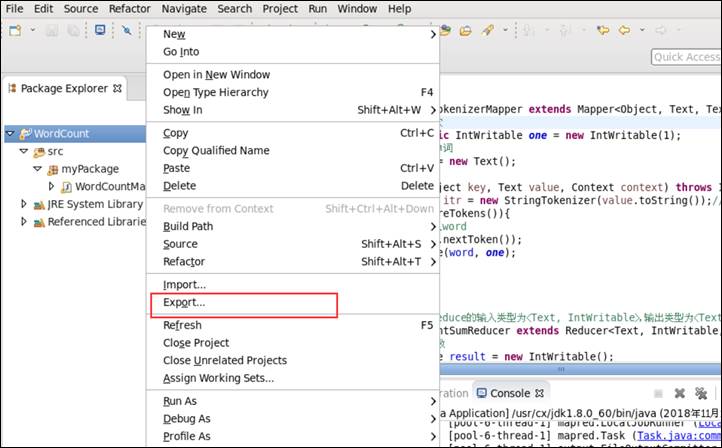
1.5  **在集群中提交任务**

如果希望在集群中进行任务的提交及运行，我们需要将工程打包，然后以命令行的形式将任务提交到集群中。

**步骤1.**  **程序打包**

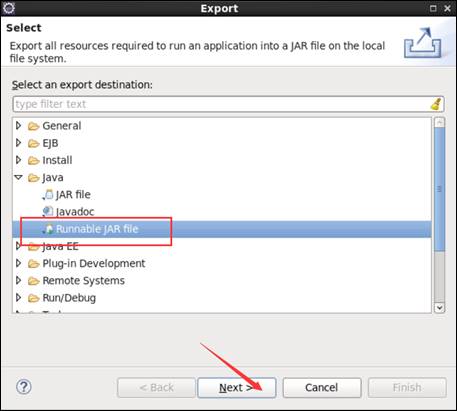
程序测试成功后，接下来便可以将程序打包，然后在集群中提交运行。

1.   将鼠标移动到项目名称上，点击鼠标右键，选择“Export”，如下图所示：



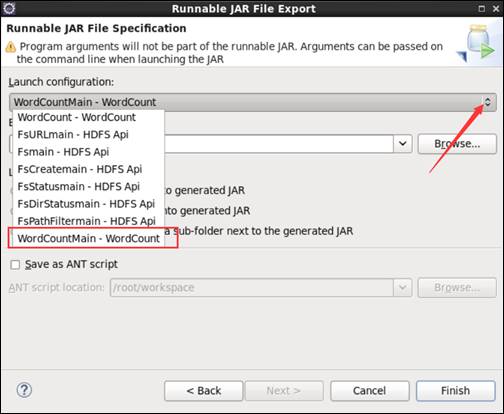
**图 1.31 导出项目**

2.   在弹出的选择菜单中，选择“Runnable JAR file”：



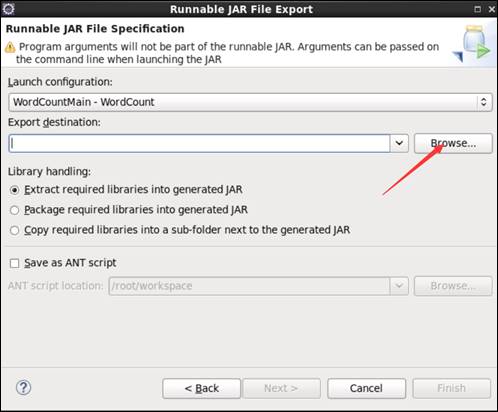
**图 1.32 项目类型选择**

3.   在“Launch configuration”的下拉菜单中选择上文配置的“WordCountMain”：



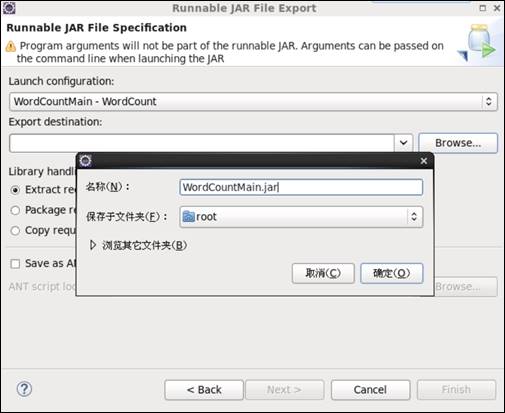
**图 1.33 信息配置**

4.   点击“Browse”按钮，设置文件导出目录：



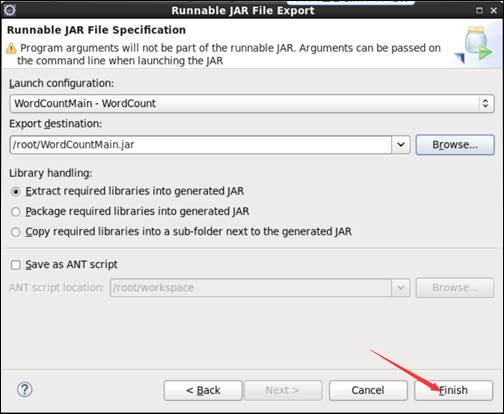
**图 1.34 配置目标路径**

5.   设置将打包程序命名为WordCountMain.jar，并存储到/root目录下：



**图 1.35 名称配置**

6.   配置完成后点击“Finish”：



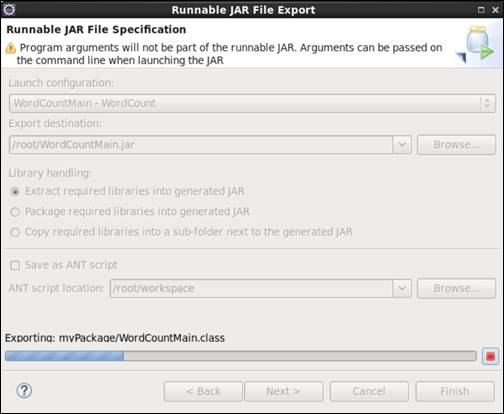
**图 1.36 执行项目导出**

7.   当出现提示信息时，点击“OK”：



**图 1.37 提示框信息**

8.   等待打包完成即可：



**图 1.38 导出运行**



**图 1.39 导出完成**

9.   程序打包完成后，在终端模拟器中通过下面的命令可以查看文件是否成功生成：

ll /root/WordCountMain.jar

**步骤2.**  **任务提交**

1.   通过下面的命令，指定HDFS文件的绝对路径作为第一个参数（注：输出路径在HDFS上必须是不存在的，否则运行会报错），将程序在集群中提交运行：

hadoop jar /root/WordCountMain.jar hdfs://master:9000/WordCount/hadoop-root-namenode-master.out hdfs://master:9000/WordCount/output1

2.   MapReduce默认会将结果存储到生成目录下的part-r-00000文件中，可以通过下面的命令可以查看所生成的文件的内容：

hadoop fs -cat /WordCount/output1/part-r-00000