实验目的：

（1）了解MapReduce特点,理解其工作机制； （2）能够编写简单人MapReduce程序.

实验内容：

MapReduce是谷歌公司的核心计算模型，Hadoop开源实现了MapReduce。MapReduce将复杂的、运行于大规模集群上的并行计算过程高度抽象到了两个函数：Map和Reduce，并极大地方便了分布式编程工作，编程人员在不会分布式并行编程的情况下，也可以很容易将自己的程序运行在分布式系统上，完成海量数据的计算。  
本教程以一个词频统计任务为主线，详细介绍MapReduce基础编程方法。环境是Ubuntu18.04（或Ubuntu16.04或Ubuntu14.04）、Hadoop3.1.3，开发工具是Eclipse。

1. 词频统计任务要求

首先，在Linux系统本地创建两个文件，即文件wordfile1.txt和wordfile2.txt。在实际应用中，这两个文件可能会非常大，会被分布存储到多个节点上。但是，为了简化任务，这里的两个文件只包含几行简单的内容。需要说明的是，针对这两个小数据集样本编写的MapReduce词频统计程序，不作任何修改，就可以用来处理大规模数据集的词频统计。  
文件wordfile1.txt的内容如下：

I love Spark

I love Hadoop

文件wordfile2.txt的内容如下：

Hadoop is good

Spark is fast

假设HDFS中有一个/user/hadoop/input文件夹，并且文件夹为空，请把文件wordfile1.txt和wordfile2.txt上传到HDFS中的input文件夹下。现在需要设计一个词频统计程序，统计input文件夹下所有文件中每个单词的出现次数，也就是说，程序应该输出如下形式的结果：

fast 1

good 1

Hadoop 2

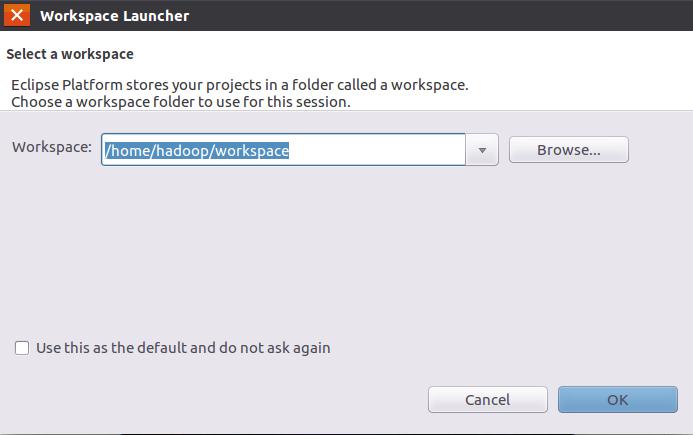
I 2

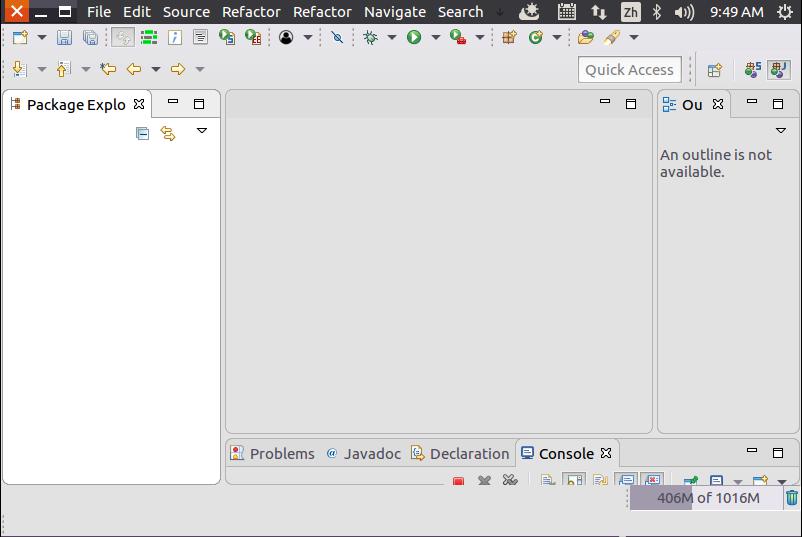
is 2

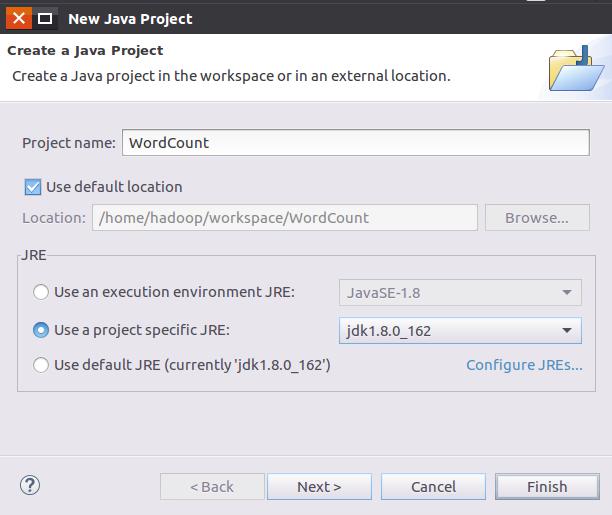
love 2

Spark 2

2. 在Eclipse中创建项目

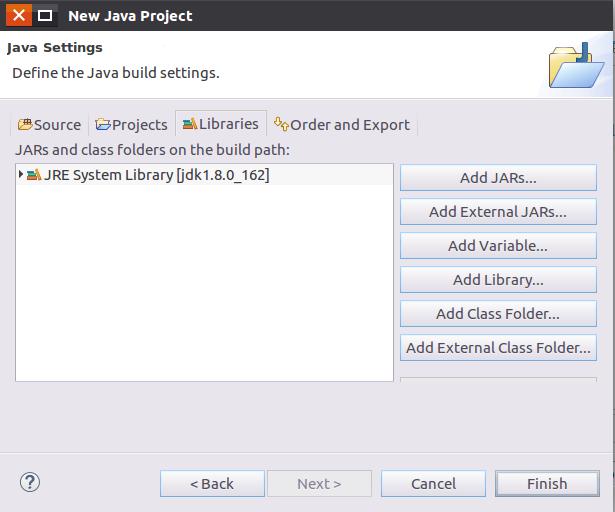
首先，启动Eclipse，启动以后会弹出如下图所示界面，提示设置工作空间（workspace）。  


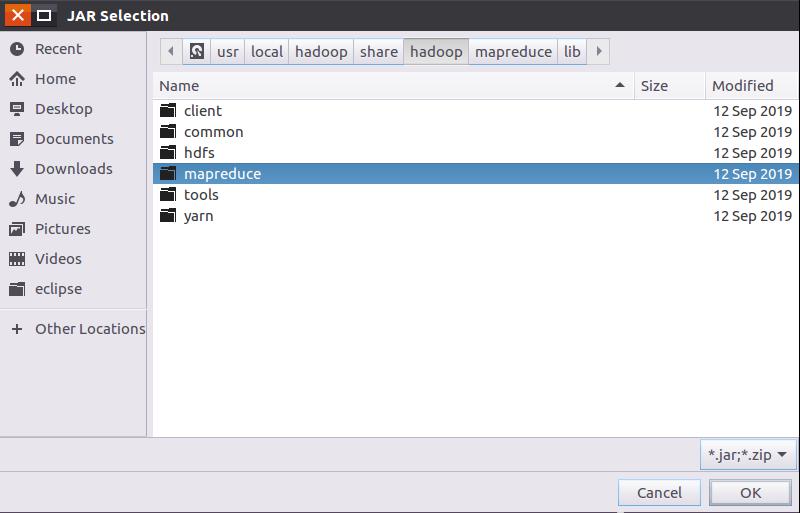
可以直接采用默认的设置“/home/hadoop/workspace”，点击“OK”按钮。可以看出，由于当前是采用hadoop用户登录了Linux系统，因此，默认的工作空间目录位于hadoop用户目录“/home/hadoop”下。  
Eclipse启动以后，呈现的界面如下图所示。  


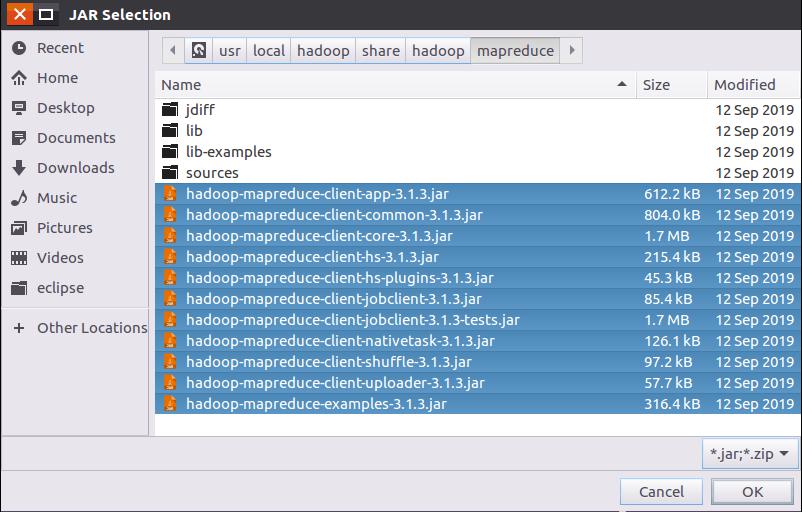
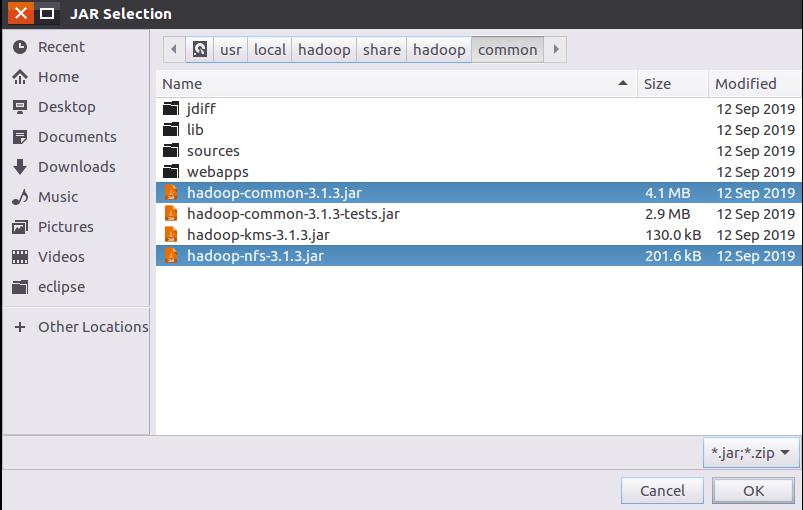
选择“File–>New–>Java Project”菜单，开始创建一个Java工程，弹出如下图所示界面。  


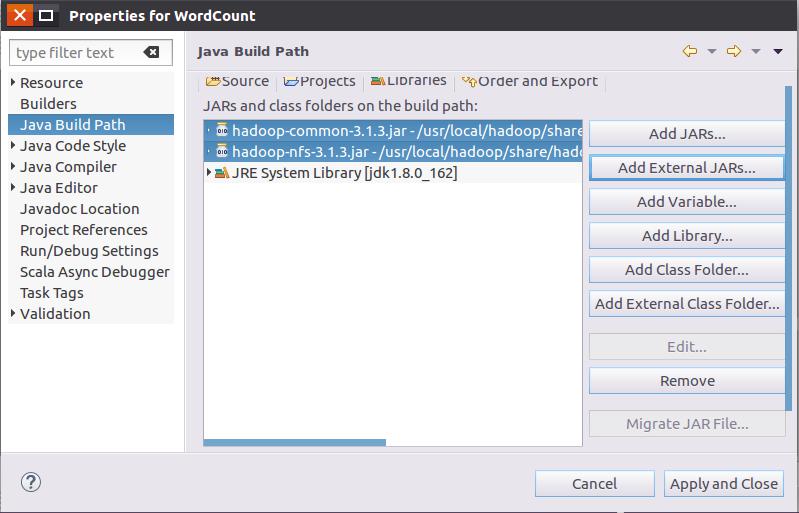
在“Project name”后面输入工程名称“WordCount”，选中“Use default location”，让这个Java工程的所有文件都保存到“/home/hadoop/workspace/WordCount”目录下。在“JRE”这个选项卡中，可以选择当前的Linux系统中已经安装好的JDK，比如jdk1.8.0\_162。然后，点击界面底部的“Next>”按钮，进入下一步的设置。

3. 为项目添加需要用到的JAR包

进入下一步的设置以后，会弹出如下图所示界面。  


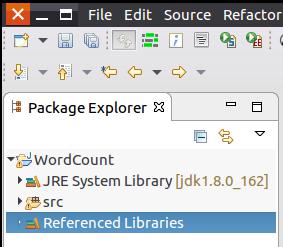
需要在这个界面中加载该Java工程所需要用到的JAR包，这些JAR包中包含了与Hadoop相关的Java API。这些JAR包都位于Linux系统的Hadoop安装目录下，对于本教程而言，就是在“/usr/local/hadoop/share/hadoop”目录下。点击界面中的“Libraries”选项卡，然后，点击界面右侧的“Add External JARs…”按钮，弹出如下图所示界面。  


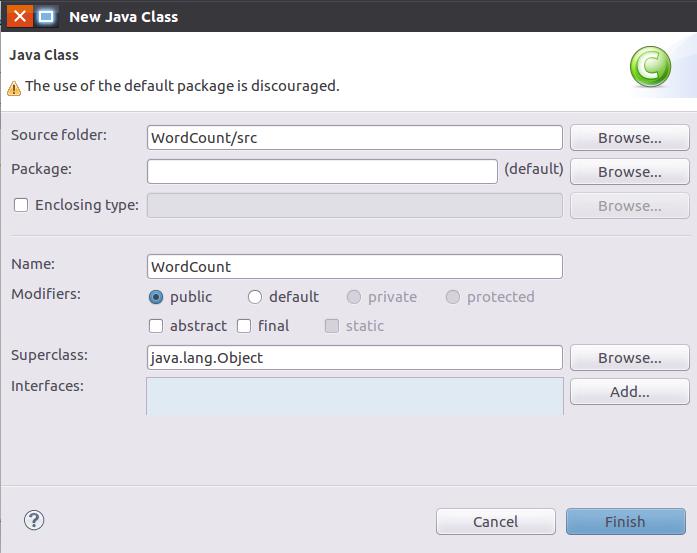
在该界面中，上面有一排目录按钮（即“usr”、“local”、“hadoop”、“share”、“hadoop”、“mapreduce”和“lib”），当点击某个目录按钮时，就会在下面列出该目录的内容。  
为了编写一个MapReduce程序，一般需要向Java工程中添加以下JAR包：  
（1）“/usr/local/hadoop/share/hadoop/common”目录下的hadoop-common-3.1.3.jar和haoop-nfs-3.1.3.jar；  
（2）“/usr/local/hadoop/share/hadoop/common/lib”目录下的所有JAR包；  
（3）“/usr/local/hadoop/share/hadoop/mapreduce”目录下的所有JAR包，但是，不包括jdiff、lib、lib-examples和sources目录，具体如下图所示。  
  
（4）“/usr/local/hadoop/share/hadoop/mapreduce/lib”目录下的所有JAR包。  
比如，如果要把“/usr/local/hadoop/share/hadoop/common”目录下的hadoop-common-3.1.3.jar和haoop-nfs-3.1.3.jar添加到当前的Java工程中，可以在界面中点击相应的目录按钮，进入到common目录，然后，界面会显示出common目录下的所有内容（如下图所示）。  


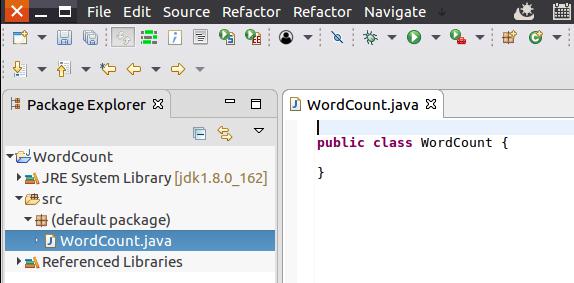
请在界面中用鼠标点击选中hadoop-common-3.1.3.jar和haoop-nfs-3.1.3.jar，然后点击界面右下角的“确定”按钮，就可以把这两个JAR包增加到当前Java工程中，出现的界面如下图所示。  


从这个界面中可以看出，hadoop-common-3.1.3.jar和haoop-nfs-3.1.3.jar已经被添加到当前Java工程中。然后，按照类似的操作方法，可以再次点击“Add External JARs…”按钮，把剩余的其他JAR包都添加进来。需要注意的是，当需要选中某个目录下的所有JAR包时，可以使用“Ctrl+A”组合键进行全选操作。全部添加完毕以后，就可以点击界面右下角的“Finish”按钮，完成Java工程WordCount的创建。

4. 编写Java应用程序

下面编写一个Java应用程序，即WordCount.java。请在Eclipse工作界面左侧的“Package Explorer”面板中（如下图所示），找到刚才创建好的工程名称“WordCount”，然后在该工程名称上点击鼠标右键，在弹出的菜单中选择“New–>Class”菜单。  


选择“New–>Class”菜单以后会出现如下图所示界面。  


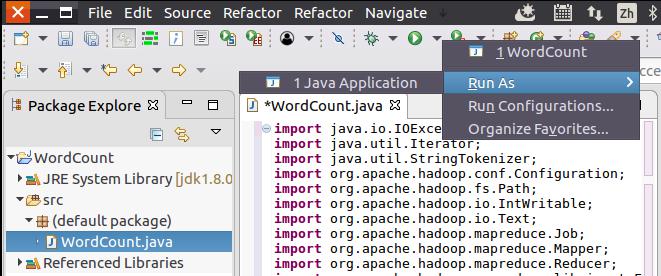
在该界面中，只需要在“Name”后面输入新建的Java类文件的名称，这里采用名称“WordCount”，其他都可以采用默认设置，然后，点击界面右下角“Finish”按钮，出现如下图所示界面。  


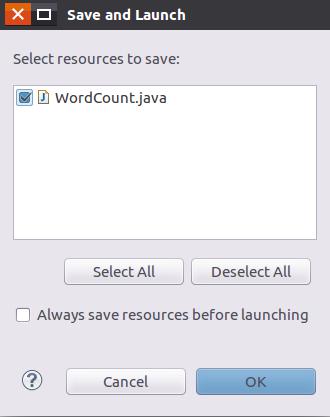
可以看出，Eclipse自动创建了一个名为“WordCount.java”的源代码文件，并且包含了代码“public class WordCount{}”，请清空该文件里面的代码，然后在该文件中输入完整的词频统计程序代码，具体如下：

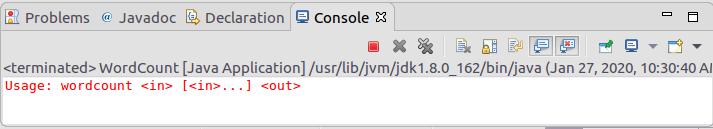
1. import java.io.IOException;
2. import java.util.Iterator;
3. import java.util.StringTokenizer;
4. import org.apache.hadoop.conf.Configuration;
5. import org.apache.hadoop.fs.Path;
6. import org.apache.hadoop.io.IntWritable;
7. import org.apache.hadoop.io.Text;
8. import org.apache.hadoop.mapreduce.Job;
9. import org.apache.hadoop.mapreduce.Mapper;
10. import org.apache.hadoop.mapreduce.Reducer;
11. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13. import org.apache.hadoop.util.GenericOptionsParser;
14. public class WordCount {
15. public WordCount() {
16. }
17. public static void main(String[] args) throws Exception {
18. Configuration conf = new Configuration();
19. String[] otherArgs = (new GenericOptionsParser(conf, args)).getRemainingArgs();
20. if(otherArgs.length < 2) {
21. System.err.println("Usage: wordcount <in> [<in>...] <out>");
22. System.exit(2);
23. }
24. Job job = Job.getInstance(conf, "word count");
25. job.setJarByClass(WordCount.class);
26. job.setMapperClass(WordCount.TokenizerMapper.class);
27. job.setCombinerClass(WordCount.IntSumReducer.class);
28. job.setReducerClass(WordCount.IntSumReducer.class);
29. job.setOutputKeyClass(Text.class);
30. job.setOutputValueClass(IntWritable.class);
31. for(int i = 0; i < otherArgs.length - 1; ++i) {
32. FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
33. }
34. FileOutputFormat.setOutputPath(job, new Path(otherArgs[otherArgs.length - 1]));
35. System.exit(job.waitForCompletion(true)?0:1);
36. }
37. public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
38. private static final IntWritable one = new IntWritable(1);
39. private Text word = new Text();
40. public TokenizerMapper() {
41. }
42. public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {
43. StringTokenizer itr = new StringTokenizer(value.toString());
44. while(itr.hasMoreTokens()) {
45. this.word.set(itr.nextToken());
46. context.write(this.word, one);
47. }
48. }
49. }
50. public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
51. private IntWritable result = new IntWritable();
52. public IntSumReducer() {
53. }
54. public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {
55. int sum = 0;
56. IntWritable val;
57. for(Iterator i$ = values.iterator(); i$.hasNext(); sum += val.get()) {
58. val = (IntWritable)i$.next();
59. }
60. this.result.set(sum);
61. context.write(key, this.result);
62. }
63. }
64. }

Java

5. 编译打包程序

现在就可以编译上面编写的代码。可以直接点击Eclipse工作界面上部的运行程序的快捷按钮，当把鼠标移动到该按钮上时，在弹出的菜单中选择“Run as”，继续在弹出来的菜单中选择“Java Application”，如下图所示。  


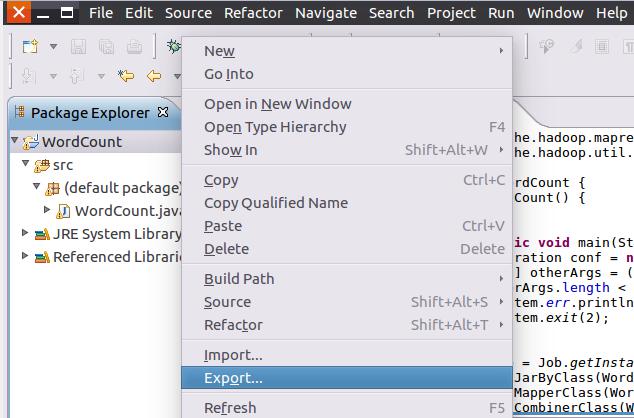
然后，会弹出如下图所示界面。  


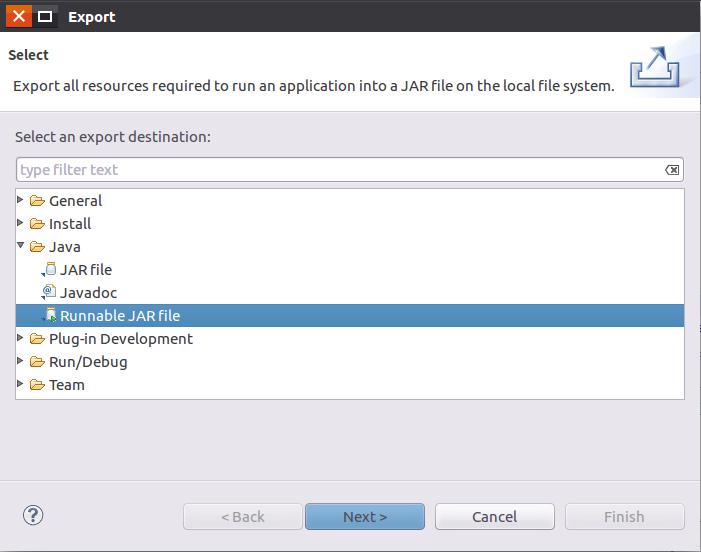
点击界面右下角的“OK”按钮，开始运行程序。程序运行结束后，会在底部的“Console”面板中显示运行结果信息（如下图所示）。  


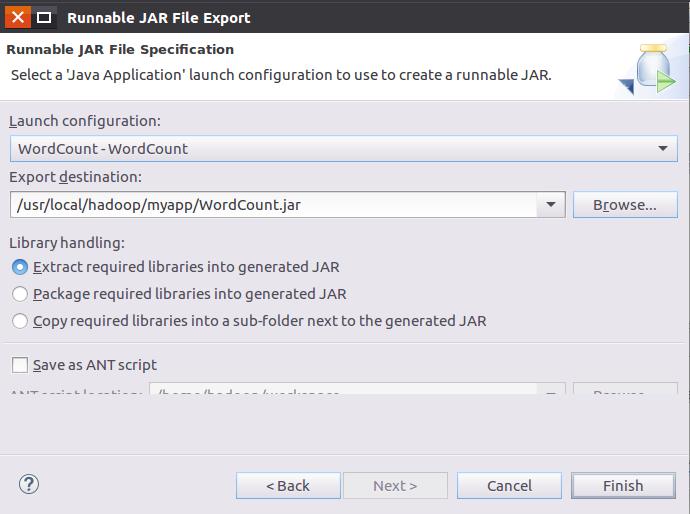
下面就可以把Java应用程序打包生成JAR包，部署到Hadoop平台上运行。现在可以把词频统计程序放在“/usr/local/hadoop/myapp”目录下。如果该目录不存在，可以使用如下命令创建：

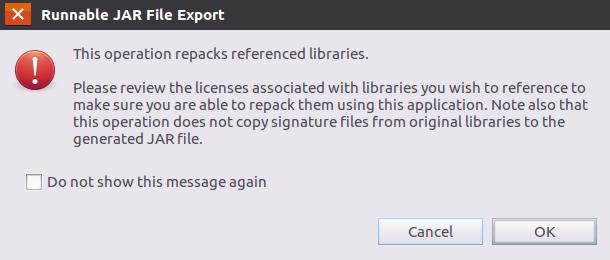
1. cd /usr/local/hadoop
2. mkdir myapp

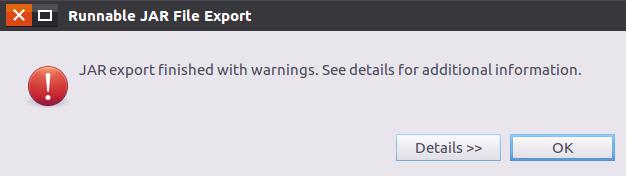
Shell 命令

首先，请在Eclipse工作界面左侧的“Package Explorer”面板中，在工程名称“WordCount”上点击鼠标右键，在弹出的菜单中选择“Export”，如下图所示。  


然后，会弹出如下图所示界面。  


在该界面中，选择“Runnable JAR file”，然后，点击“Next>”按钮，弹出如下图所示界面。  


在该界面中，“Launch configuration”用于设置生成的JAR包被部署启动时运行的主类，需要在下拉列表中选择刚才配置的类“WordCount-WordCount”。在“Export destination”中需要设置JAR包要输出保存到哪个目录，比如，这里设置为“/usr/local/hadoop/myapp/WordCount.jar”。在“Library handling”下面选择“Extract required libraries into generated JAR”。然后，点击“Finish”按钮，会出现如下图所示界面。  


可以忽略该界面的信息，直接点击界面右下角的“OK”按钮，启动打包过程。打包过程结束后，会出现一个警告信息界面，如下图所示。  


可以忽略该界面的信息，直接点击界面右下角的“OK”按钮。至此，已经顺利把WordCount工程打包生成了WordCount.jar。可以到Linux系统中查看一下生成的WordCount.jar文件，可以在Linux的终端中执行如下命令：

1. cd /usr/local/hadoop/myapp
2. ls

Shell 命令

可以看到，“/usr/local/hadoop/myapp”目录下已经存在一个WordCount.jar文件。

5. 运行程序

在运行程序之前，需要启动Hadoop，命令如下：

1. cd /usr/local/hadoop
2. ./sbin/start-dfs.sh

Shell 命令

在启动Hadoop之后，需要首先删除HDFS中与当前Linux用户hadoop对应的input和output目录（即HDFS中的“/user/hadoop/input”和“/user/hadoop/output”目录），这样确保后面程序运行不会出现问题，具体命令如下：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -rm -r input
3. ./bin/hdfs dfs -rm -r output

Shell 命令

然后，再在HDFS中新建与当前Linux用户hadoop对应的input目录，即“/user/hadoop/input”目录，具体命令如下：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -mkdir input

Shell 命令

然后，把之前在第7.1节中在Linux本地文件系统中新建的两个文件wordfile1.txt和wordfile2.txt（假设这两个文件位于“/usr/local/hadoop”目录下，并且里面包含了一些英文语句），上传到HDFS中的“/user/hadoop/input”目录下，命令如下：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -put ./wordfile1.txt input
3. ./bin/hdfs dfs -put ./wordfile2.txt input

Shell 命令

如果HDFS中已经存在目录“/user/hadoop/output”，则使用如下命令删除该目录：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -rm -r /user/hadoop/output

Shell 命令

现在，就可以在Linux系统中，使用hadoop jar命令运行程序，命令如下：

1. cd /usr/local/hadoop
2. ./bin/hadoop jar ./myapp/WordCount.jar input output

Shell 命令

上面命令执行以后，当运行顺利结束时，屏幕上会显示类似如下的信息：

……//这里省略若干屏幕信息

2020-01-27 10:10:55,157 INFO mapreduce.Job: map 100% reduce 100%

2020-01-27 10:10:55,159 INFO mapreduce.Job: Job job\_local457272252\_0001 completed successfully

2020-01-27 10:10:55,174 INFO mapreduce.Job: Counters: 35

File System Counters

FILE: Number of bytes read=115463648

FILE: Number of bytes written=117867638

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=283

HDFS: Number of bytes written=40

HDFS: Number of read operations=24

HDFS: Number of large read operations=0

HDFS: Number of write operations=5

Map-Reduce Framework

Map input records=9

Map output records=24

Map output bytes=208

Map output materialized bytes=140

Input split bytes=236

Combine input records=24

Combine output records=12

Reduce input groups=6

Reduce shuffle bytes=140

Reduce input records=12

Reduce output records=6

Spilled Records=24

Shuffled Maps =2

Failed Shuffles=0

Merged Map outputs=2

GC time elapsed (ms)=0

Total committed heap usage (bytes)=1291321344

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=113

File Output Format Counters

Bytes Written=40

词频统计结果已经被写入了HDFS的“/user/hadoop/output”目录中，可以执行如下命令查看词频统计结果：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -cat output/\*

Shell 命令

上面命令执行后，会在屏幕上显示如下词频统计结果：

Hadoop 2

I 2

Spark 2

fast 1

good 1

is 2

love 2

至此，词频统计程序顺利运行结束。需要注意的是，如果要再次运行WordCount.jar，需要首先删除HDFS中的output目录，否则会报错。