eclipse开发环境搭建

一、实验目的

安装eclipse和hadoop相关插件后成功运行wordcount。

二、实验内容（实验过程、步骤及实验结果）

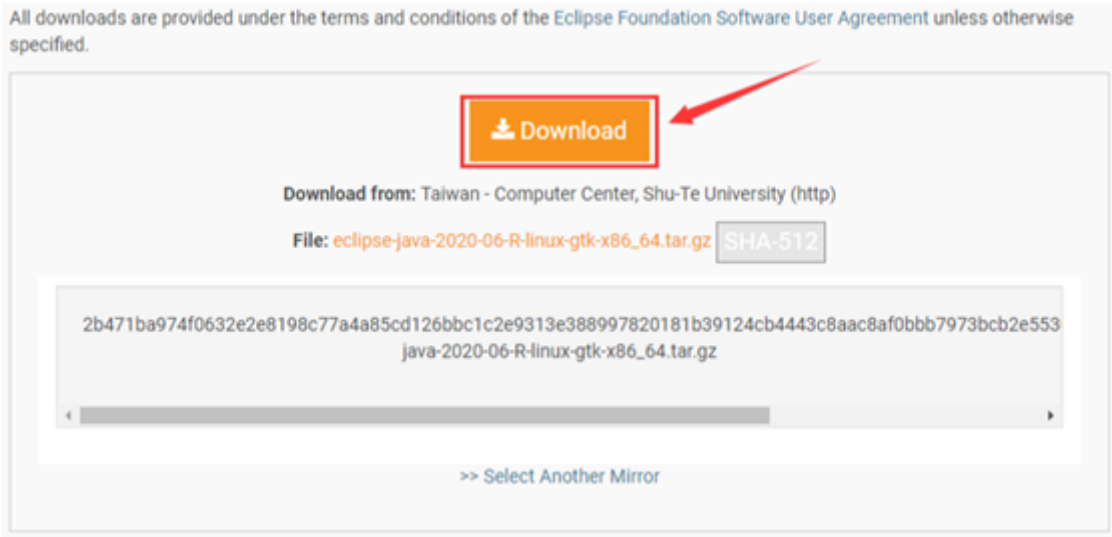
1.安装eclipse

1.1下载安装包

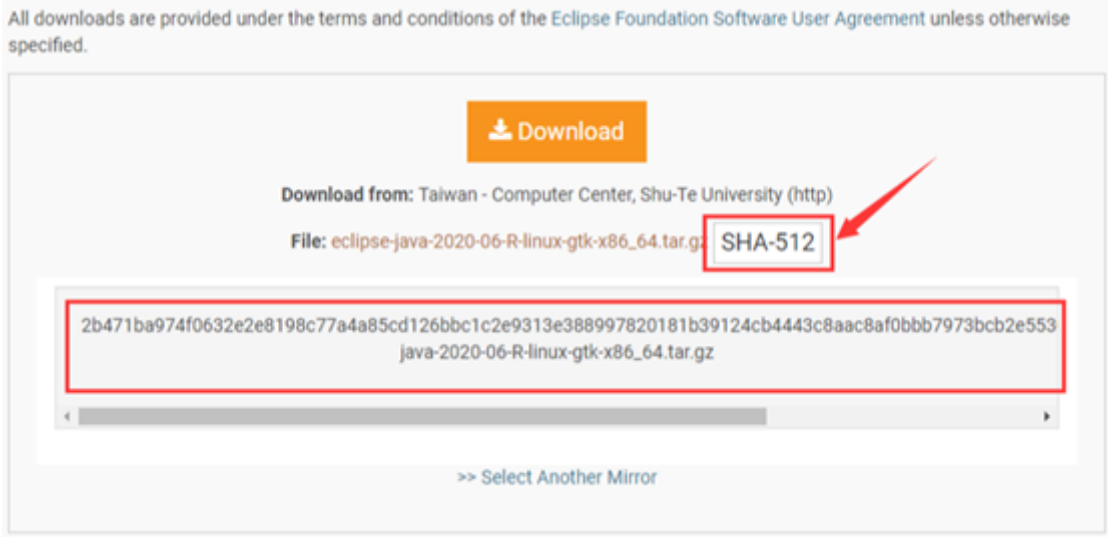
官网地址：https://www.Eclipse.org/downloads/packages/release/2020-06/r

下载Eclipse IDE for Java Develpers Linux版本，点击【Linux 64-bit】

在弹出的页面，点击Download进行下载，保存到自己想保存的目录里



然后，返回上面的下载页面，点击SHA-512，生成 sha512散列值。



在下载目录中创建文件Eclipse.sha512并将红框中的内容复制进去。保存退出。 等安装包下载完成以后，在目录下执行如下命令进行完整性校验，sha512sum -c Eclipse.sha512

校验时，生成下载文件的sha512值，并和网站提供的值进行对比，如果一致，则返回OK，否则返回错误信息。如果验证不一致，需要重新下载。

1.2解压eclipse

```
## 解压到/apps/
sudo tar -xzvf eclipse-java-2020-06-R-linux-gtk-x86_64.tar.gz -C /apps/
## 从/下解压的文件夹属于root用户，修改所属用户
cd /apps/
sudo chown chen:chen eclipse
```

1.3配置eclipse

```
## 建立jre软链接
mkdir /apps/eclipse/jre
ln -s /apps/java/bin /apps/eclipse/jre
```

如果不做这一步,eclipse无法启动

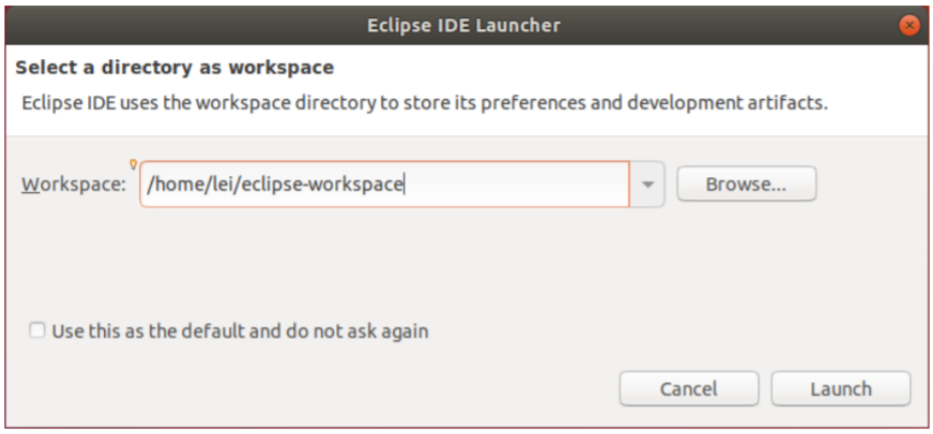

```
## 创建图标
sudo vim /usr/share/applications/eclipse.desktop
## 添加如下内容，注意空格，直接粘贴复制可能会存在问题
################################################################################
[Desktop Entry]
Encoding=UTF-8
Name=eclipse
Comment=eclipse
Exec=/apps/eclipse/eclipse
Icon=/apps/eclipse/icon.xpm
Terminal=false
StartupNotify=true
Type=Application
Categories=Application;Development;
################################################################################
## 赋予图标可执行权限
sudo chmod U+x /usr/share/applications/eclipse.desktop
```

按win键，搜索eclipse,可见图标，点击，打开。

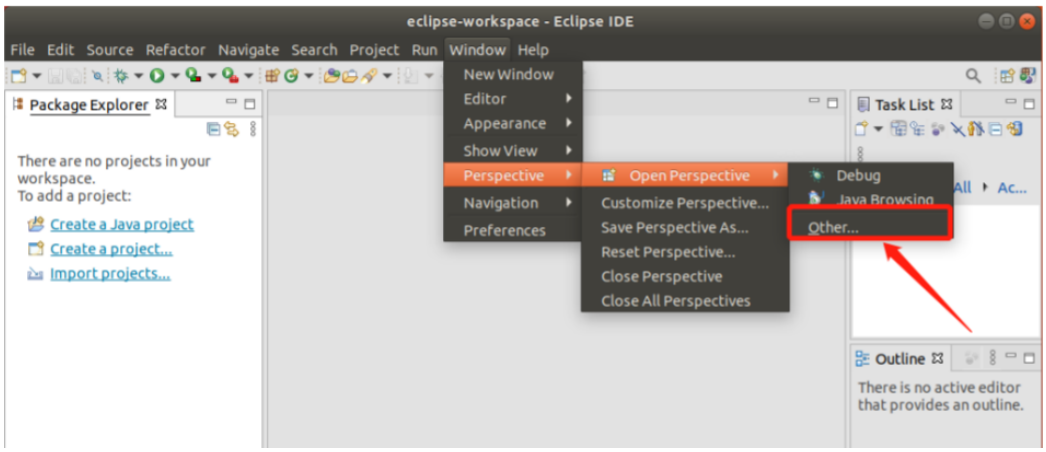
2.安装hadoop插件

```
## 将hadoop-eclipse-plugin-2.6.0.jar拷贝到/apps/eclipse/dropins插件目录
cp hadoop-eclipse-plugin-2.6.0.jar /apps/eclipse/dropins/
```

打开eclipse，设置workspace路径，点击launch。



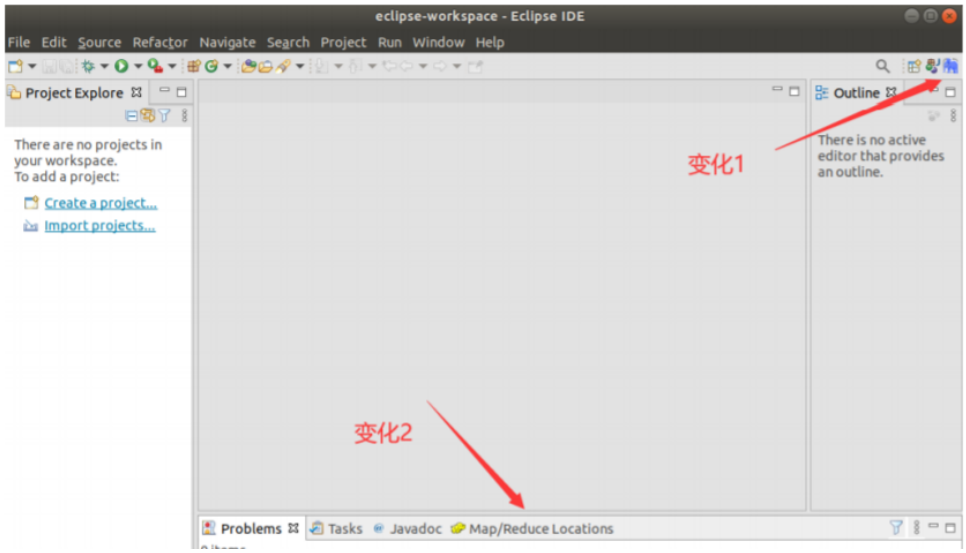
关闭welcome界面，然后如下图操作。（perspective是一个包含一系列视图和内容的可视容器）



弹出如下图的窗口。

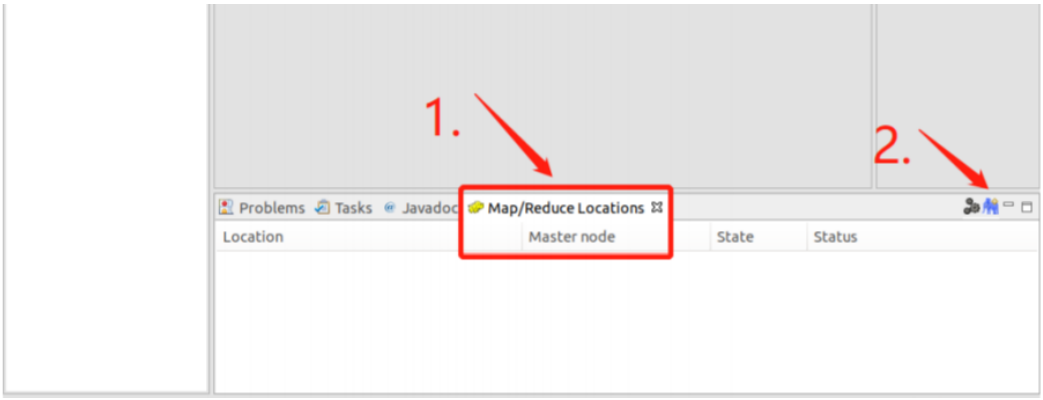
因为我们将 hadoop-Eclipse-plugin-2.6.0.jar 拷贝到了/apps/Eclipse/dropins/下，所以这里多出一下 Map/Reduce 选项。如果是在 Eclipse 打开的状态下拷贝的，需要重启一下才能显出 Map/Reduce 这一项。

选择【Map/Reduce】，并点击【Open】，可以看到窗口中，有两个变化。（右上角操 作布局切换、面板窗口）



3.添加 Hadoop 配置，连接 Hadoop 集群

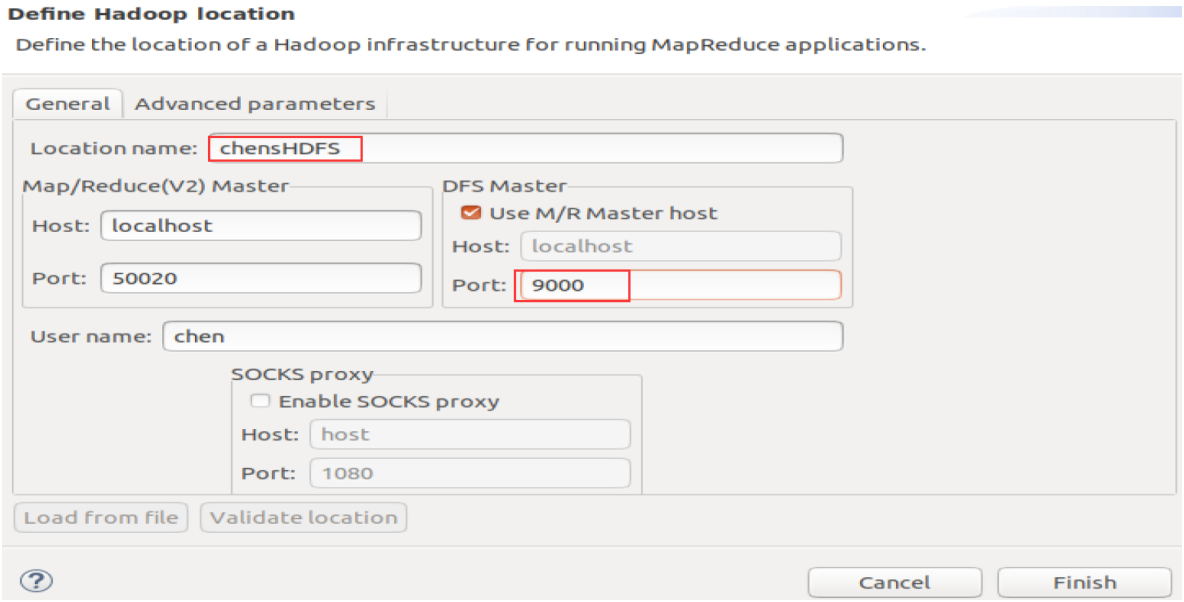
接下来，添加 Hadoop 配置，连接 Hadoop 集群。在下图中先选中 1，再点击 2 。



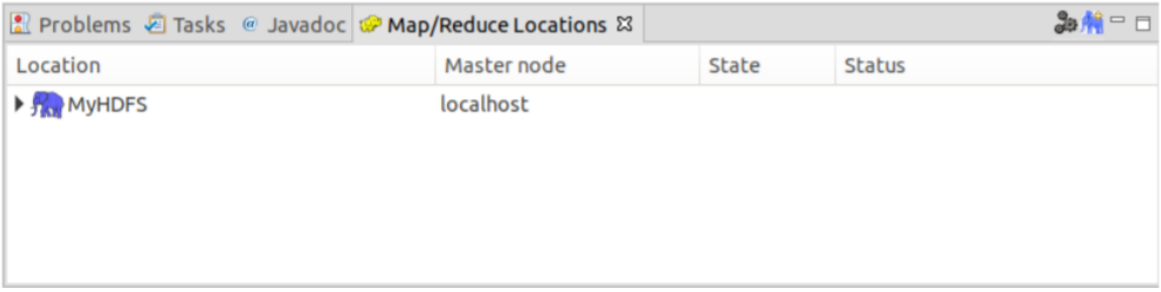
在弹出的窗口中，添加 Hadoop 相关配置。 Location name，是为此配置起的一个名字。

DFS Master，是连接 HDFS 的主机名和端口号。我们在安装 Hadoop 时，在 coresite.xml 文件中进行了设置。

```
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
```



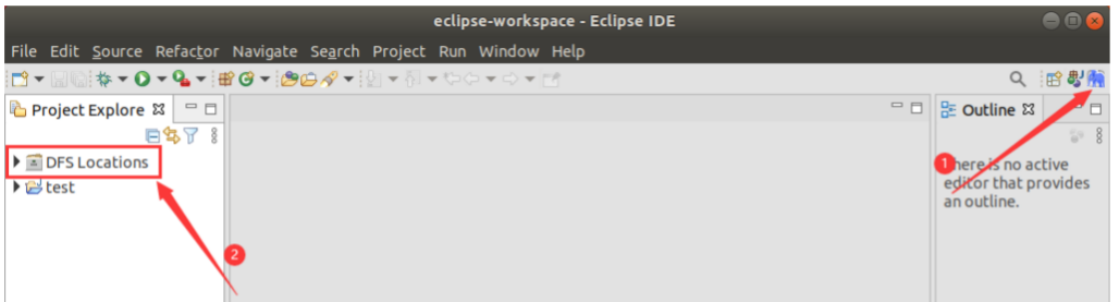
点击 Finish 保存配置。配置好的 HDFS Location 就显示在下方的窗口中。



4.hadoop插件的使用

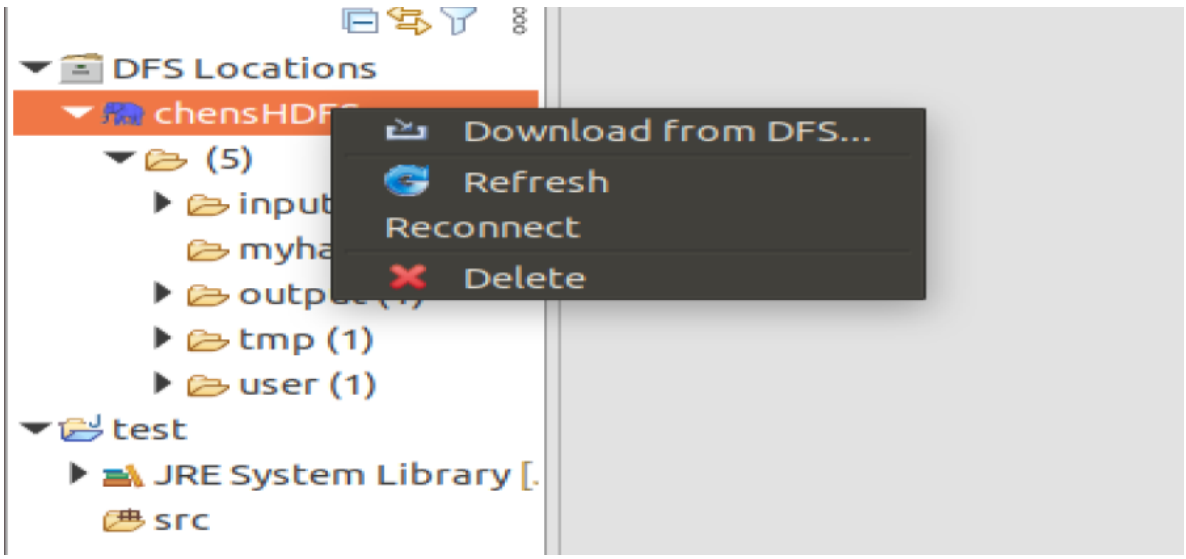
```
## 启动hadoop
/apps/hadoop/sbin/start-all.sh
```

新建一个 Java 工程 test。点击【File】=>【New】=>【Project】; 在对话框中选择【Java Project】，点击【Next】。 在对话框中输入【Project name】，点击【Finish】。 在弹出的窗口中点击【Open Perspective】。 新建的 Project 会显示在左侧的 Package Exploer 中。 点击右上角的 mapreduce 图标，左侧的 Package Exploer 会切换到 Project Exploer。 DFS Location 也显示在这里。




```
## 使用命令查看hdfs目录
hadoop fs -ls /
```

右键 chensHDFS 下的文件夹，在弹出的菜单中，可以对 HDFS 进行操作，比如上传下载文 件或目录，在 HDFS 上创建文件夹，删除文件或文件夹。 操作以后，右键，选择 Refresh，可用刷新 HDFS 目录。（如果刷新不出来，重启 Eclipse 即可）。



5.运行wordcount

5.1准备数据文件

```
## 在hdfs上创建文件夹/input/wordcount
hadoop fs -mkdir /input/wordcount
## 在/data目录下创建文件testfile
vim /data/testfile
## 并写入如下内容
################################################################################
##
hello big data
hello hadoop
hello hdfs
################################################################################
##

## 将/data/testfile上传HDFS上/input/wordcount目录
hadoop fs -put /data/testfile /input/wordcount/
## 我们统计文件/input/wordcount/testfile 出现的词的词频。
```

5.2准备jar包

```
## 创建big_data_tools/hadoop3lib文件夹，用于存放为外部的 jar 文件
## 将/apps/hadoop/share/hadoop目录下的common,hdfs,mapreduce,yarn 4个子目录中的jar文件## 以及这4个子目录下lib文件夹中的所有jar文件都复制到hadoop3lib里
```

5.3完成代码

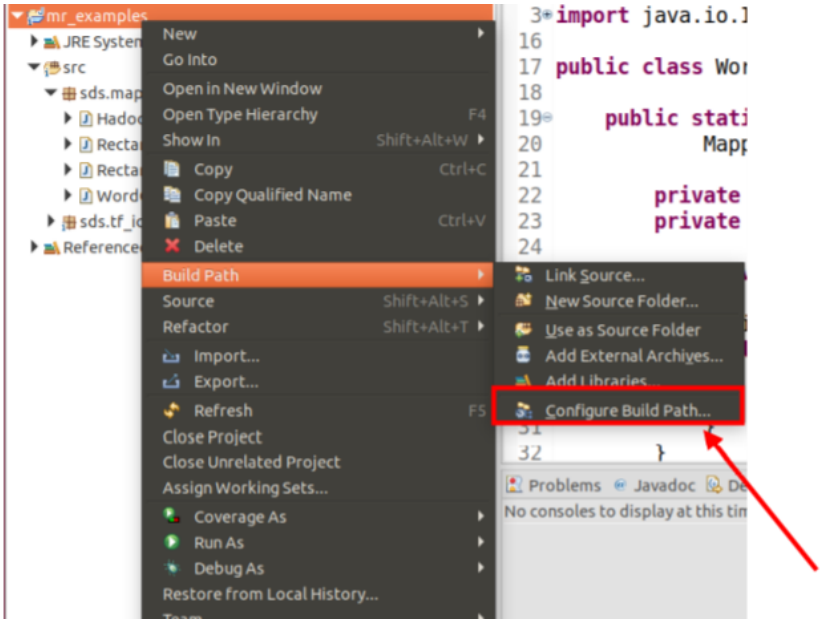
创建 Project【mr_example】，在项目中创建 Package【sds.mapreduce】，然后创建 Class【WordCount】，将在中间窗口中自动打开 WordCount.java 文件，删除其中的代码，将下面的代码复制其中并保存 .

```
package sds.mapreduce;
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;
public class WordCount {
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());
```

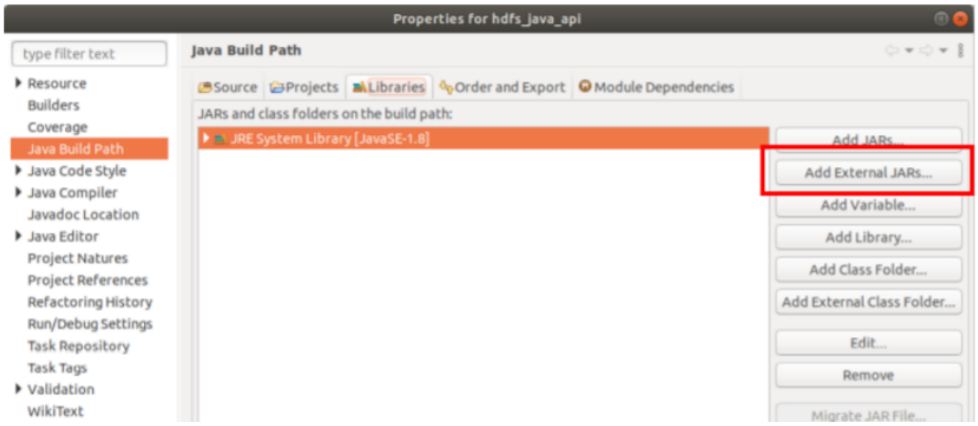
```
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
context.write(word, one);
}
}
}
public static class IntSumReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
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> [<in>...]
<out>");
System.exit(2);
}
Job job = Job.getInstance(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);
for (int i = 0; i < otherArgs.length - 1; ++i) {
FileInputFormat.addInputPath(job, new
Path(otherArgs[i]));
}
FileOutputFormat.setOutputPath(job, new
Path(otherArgs[otherArgs.length - 1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

5.4添加外部jar文件

在 Package Explorer 工程名 mr_example 上点击右键，选择【Build Path】=> 【Configure Build Path...】。



在【Java Build Path】对话框中，选择【Libraries】标签，点击右侧的【Add External JARs...】按钮。



在弹出的jar文件选择窗口,切换到目录hadoop3libs，全选目录中的jar文件，点击【open】.

回到【Java Build Path】对话框，点击【Apply and Close】，完成添加 jar 包。

5.4运行代码

因为代码中设置了从命令行获取参数，所以运行时，需要提供参数的值。在 Eclipse 中点 击右键，选择【Run As】=>【Run Configurations...】，

在弹出的【Run configurations】对话框中，从左边的列表中选择我们要运行的 Java Application【WordCount】。然后在右边选择【Arguments】标签，在【Program arguments】中提供需要的参数值，也就是我们要统计词频的输入文件和存放结果的输出 目录，各个参数用**空格**隔开。注意如果输出目录在 HDFS 上已存在，需要先删除，否则会 报错。

```
hdfs://localhost:9000/input/wordcount/testfile
hdfs://localhost:9000/output/wordcount
```

这里 HDFS 上的地址需要写全，前面需要加上 hdfs://localhost:9000/。设置好以后点击 【Run】。

运行结束以后，我们可以在【Project Explorer】中的【DFS Locations】中看到，在

/output 目录下生成了子目录/wordcount，双击打开其中的文件 part-r-00000，可以看到词频统计结果与我们预期的结果是一致的。

也可以在命令行查看

```
hadoop fs -cat /output/wordcount/part-r-00000
############################
big     1
data    1
hadoop  1
hdfs    1
hello   3
############################
```