

合肥工业大学

专业课程

(计算机与信息学院)

大数据处理技术实验报告

专 业 班 级 计算机科学与技术 18-2 班

学生姓名及学号 孙淼 2018211958

课 程 教 学 班 号 0521630X--001

任 课 教 师 吴共庆

实 验 指 导 教 师 吴共庆

实 验 地 点 翠六教 负 09

2020~2021 学年第 二 学期

说 明

实验报告是关于实验教学内容、过程及效果的记录和总结，因此，应注意以下事项和要求：

1. 每个实验单元在 50 页的篇幅内完成一份报告。“实验单元”指按照实验指导书规定的实验内容。若篇幅不够，可另附纸。

2、各实验的预习部分的内容是进入实验室做实验的必要条件，请按要求做好预习。

3. 实验报告要求：书写工整规范，语言表达清楚，数据和程序真实。理论联系实际，认真分析实验中出现问题与现象，总结经验。

4. 参加实验的每位同学应独立完成实验报告的撰写，其中程序或相关的设计图纸也可以采用打印等方式粘贴到报告中。严禁抄袭或拷贝，否则，一经查实，按作弊论取，并取消理论课考试资格。

5. 实验报告作为评定实验成绩的依据。

实验序号及名称:实验 2 在 Hadoop 平台上部署 WordCount 程序

实验时间： 2021 年 4 月 20 日

预习内容

一、实验目的和要求：

- 1) 了解 Linux 系统和虚拟机相关知识
- 2) 熟悉 Linux 系统下的 JAVA JDK 安装
- 3) 掌握多台主机虚拟化以及 Hadoop 的联合部署
- 4) 掌握基本的云计算雏形应用

二、实验任务：

通过多台主机的虚拟化以及 Hadoop 的联合部署，在 Hadoop 平台上部署 WordCount 程序，以此来体验基本的云计算雏形应用，并加深对云计算相关知识的了解。

三、实验准备方案，包括以下内容：

系统：虚拟机下的 CentOS 6.5-bare0.1（是 Linux 发行版之一，它是来自于 Red Hat Enterprise Linux 依照开放源代码规定释出的源代码所编译而成）；

Windows 10 系统。

组件：其中一台 namenode，安装系统时命名 master；另外两台为 datanode，安装系统时分别命名为 slave、slave2。

工具：Virtual-Machine-ware WorkStation 15 Player；SSH Secure File Transfer Client（实现 win10 本机和虚拟机安装的 CentOS6.5 的文件互传）；JavaTM1.5.x，必须安装，建议选择 Sun 公司发行的 Java 版本；

ssh 必须安装并且保证 sshd 一直运行，以使用 Hadoop 脚本管理远端 Hadoop 守护进程。

框架:

hadoop-2.5.2

Eclipse

源代码:

```
import java.io.IOException;
import java.util.Iterator;
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 WordCountTest {
    public WordCountTest() {
    }

    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 test");
        job.setJarByClass(WordCountTest.class);
        job.setMapperClass(WordCountTest.TokenizerMapper.class);
        job.setCombinerClass(WordCountTest.IntSumReducer.class);
        job.setReducerClass(WordCountTest.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);
    }

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

        public IntSumReducer() {
        }

        public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable,
Text, IntWritable>.Context context) throws IOException, InterruptedException {
            int sum = 0;

            IntWritable val;
            for(Iterator itr = values.iterator(); itr.hasNext(); sum += val.get()) {
                val = (IntWritable)itr.next();
            }

            this.result.set(sum);
            context.write(key, this.result);
        }
    }

    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
        private static final IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public TokenizerMapper() {
        }

        public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context
context) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());

            while(itr.hasMoreTokens()) {
                this.word.set(itr.nextToken());
                context.write(this.word, one);
            }
        }
    }

```

```
}  
}  
}
```

实验内容

一、实验用仪器、设备：

实验用设备：

MECHREVO 微机 DESKTOP-6A05AKM

处理器 Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.20 GHz

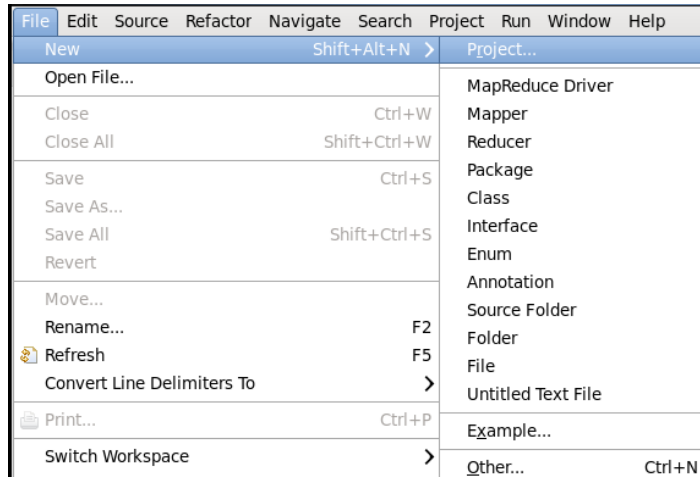
机带 8.00 GB (7.86 GB 可用)

系统类型 64 位操作系统，基于 x64 的处理器

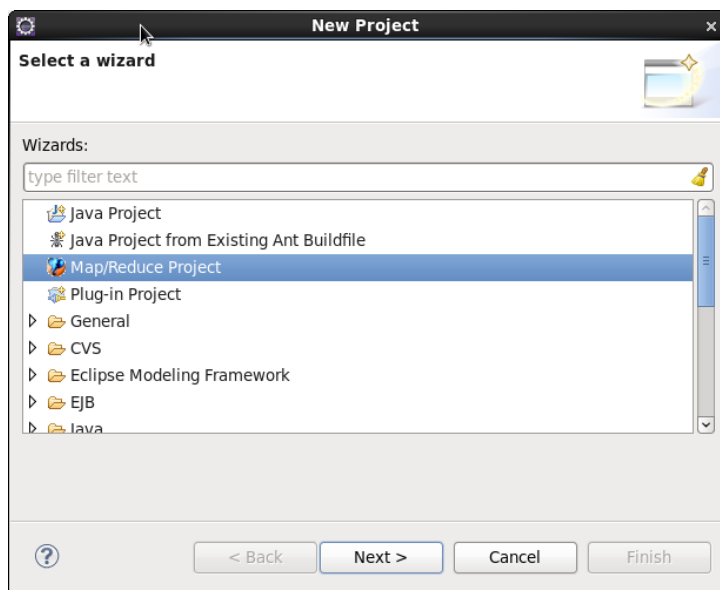
二、实验内容与步骤（过程及数据记录）:

(1) 在 Eclipse 中创建 “WordCount” MapReduce 项目

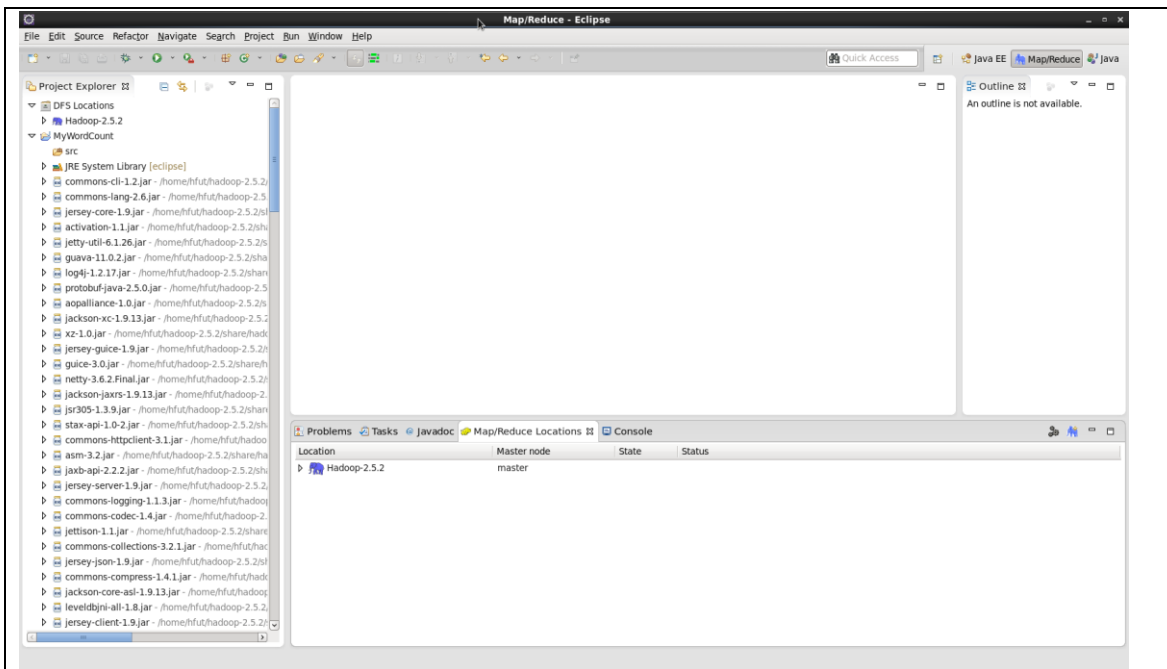
点击 File 菜单，选择 New -> Project…:



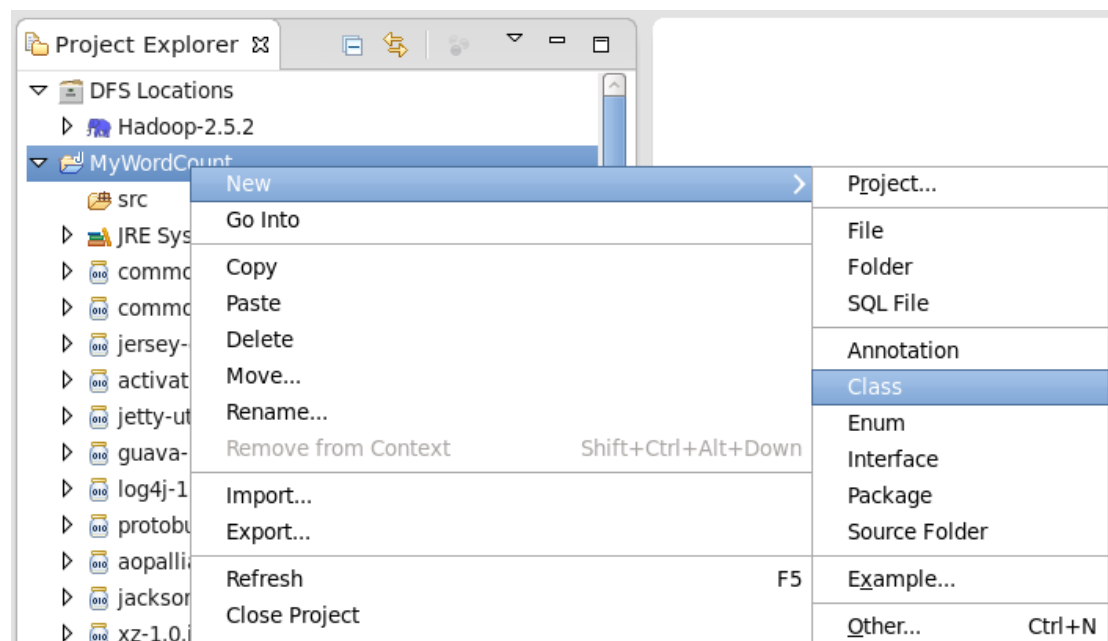
选择 Map/Reduce Project，点击 Next:



填写 Project name 为 MyWordCount，点击 Finish 创建项目。



右键点击 MyWordCount 项目，选择 New -> Class:



在 Name 处填写 WordCountTest。

New Java Class

Java Class

⚠ The use of the default package is discouraged.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

将如下 WordCountTest 的代码复制到该 WordCountTest.java 中。

```
import java.io.IOException;
import java.util.Iterator;
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 WordCountTest {
    public WordCountTest() {
    }

    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 test");
        job.setJarByClass(WordCountTest.class);
        job.setMapperClass(WordCountTest.TokenizerMapper.class);
        job.setCombinerClass(WordCountTest.IntSumReducer.class);
        job.setReducerClass(WordCountTest.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);
    }

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

        public IntSumReducer() {
        }

        public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable,
Text, IntWritable>.Context context) throws IOException, InterruptedException {
            int sum = 0;

            IntWritable val;
            for(Iterator itr = values.iterator(); itr.hasNext(); sum += val.get()) {
                val = (IntWritable)itr.next();
            }
        }
    }
}

```

```

        }

        this.result.set(sum);
        context.write(key, this.result);
    }
}

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
    private static final IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public TokenizerMapper() {
    }

    public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context
context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());

        while(itr.hasMoreTokens()) {
            this.word.set(itr.nextToken());
            context.write(this.word, one);
        }
    }
}
}

```

（2）将 Hadoop 配置文件添加到 “WordCount” MapReduce 项目

将 log4j.properties 复制到 WordCount 项目下的 src 文件夹（~/workspace/WordCount/src）中：

```
[hfut@master ~]$ cp ~/hadoop-2.5.2/etc/hadoop/log4j.properties ~/workspace/MyWordCount/src
```

log4j 用于记录程序的输出日记，需要 log4j.properties 这个配置文件，如果没有复制该文件到项目中，运行程序后在 Console 面板中会出现警告提示：

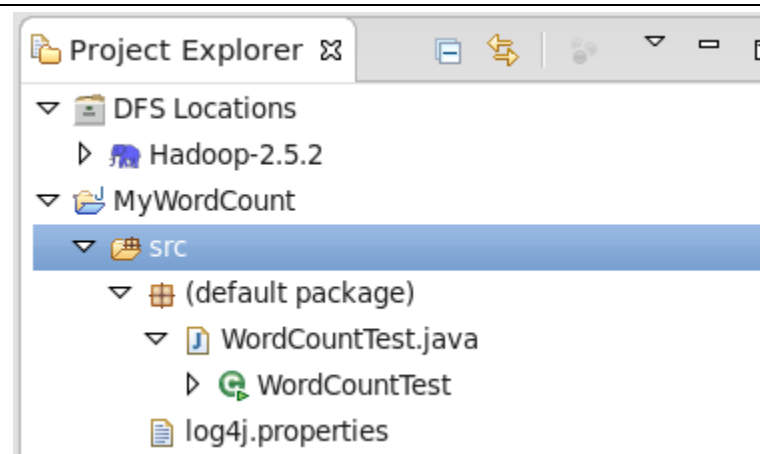
```
log4j:WARN No appenders could be found for logger
(org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
```

```
log4j:WARN Please initialize the log4j system properly.
```

```
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

虽然不影响程序的正确运行的，但程序运行时无法看到任何提示消息（只能看到出错信息）。

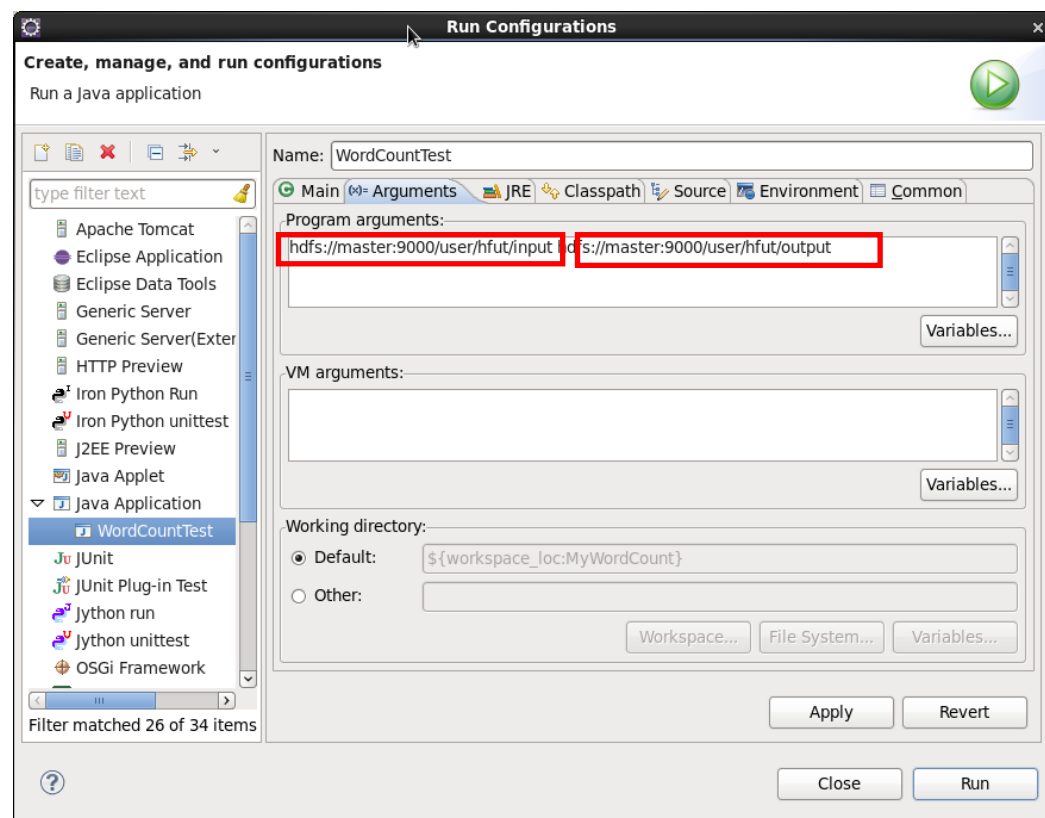
复制完成后，务必右键点击 WordCount 选择 refresh 进行刷新（不会自动刷新，需要手动刷新），可以看到文件结构如下所示：



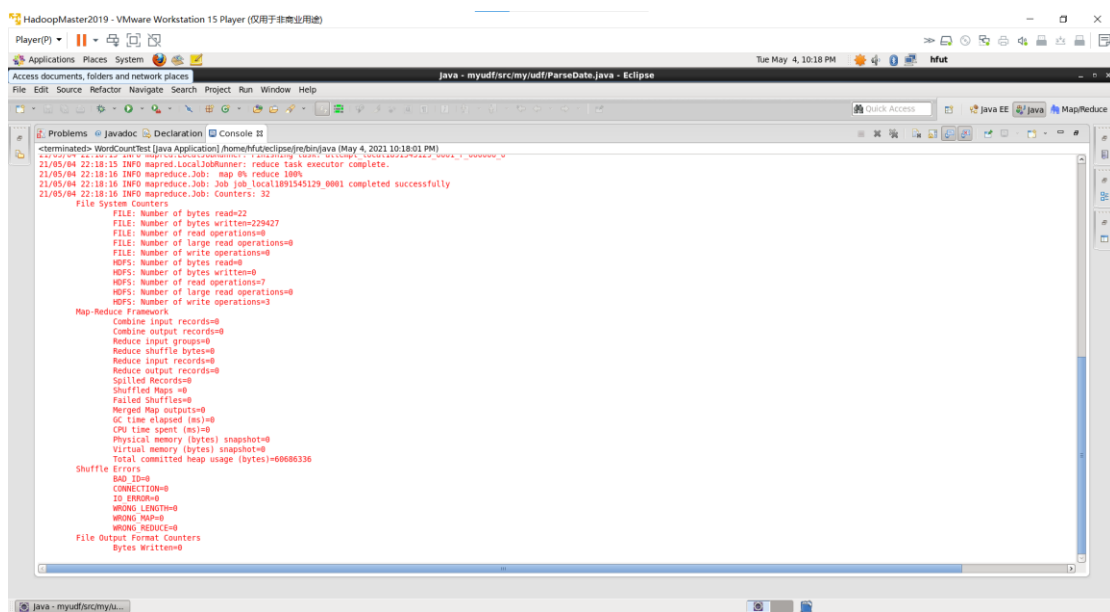
(3) 通过 Eclipse 运行 “MyWordCount” MapReduce 项目

点击工具栏中的 Run 图标，或者右键点击 Project Explorer 中的 WordCountTest.java，选择 Run As -> Run on Hadoop，就可以运行 MapReduce 程序了。不过由于没有指定参数，运行时提示 “Usage: wordcount”，需要通过 Eclipse 设定一下运行参数。

右键点击刚创建的 WordCount.java，选择 Run As -> Run Configurations，在此处可以设置运行时的相关参数（如果 Java Application 下面没有 WordCount，那么需要先双击 Java Application）。切换到 “Arguments” 栏，在 Program arguments 处填写 “hdfs://master:9000/user/hfut/input hdfs://master:9000/user/hfut/output” 就可以了。

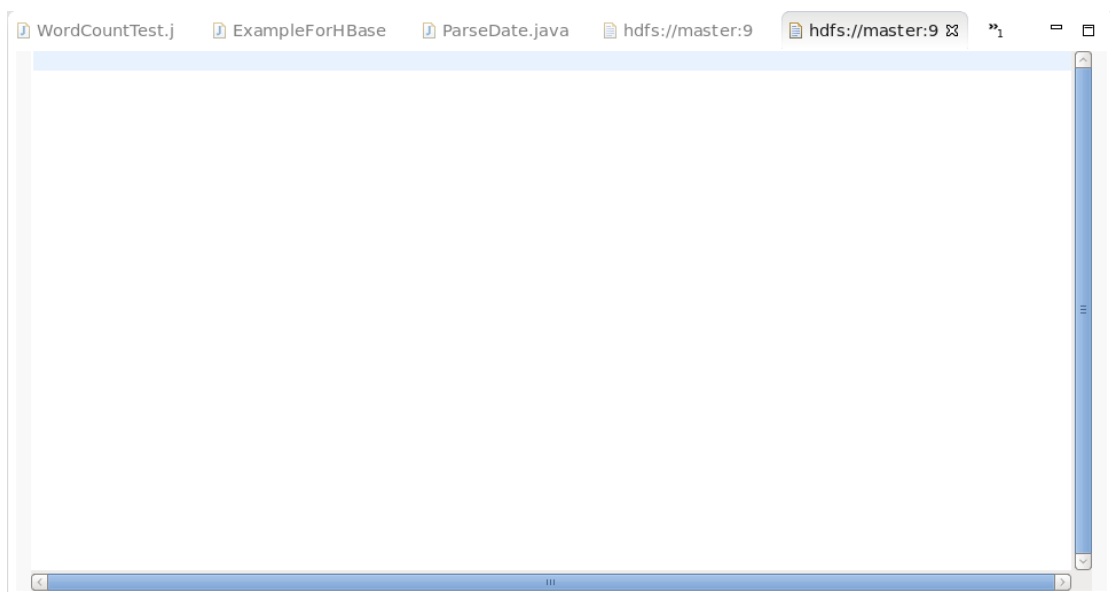


点击 Run 运行程序，可以看到运行成功的提示。



```
<terminated> WordCountTest [Java Application] /home/hfut/eclipse/re/bin/java (May 4, 2021 10:18:01 PM)
21/05/04 22:18:15 INFO mapred.LocalJobRunner: reduce task executor complete.
21/05/04 22:18:16 INFO mapreduce.Job: map 0% reduce 100%
21/05/04 22:18:16 INFO mapreduce.Job: Job (job_local1891545129_0001) completed successfully
21/05/04 22:18:16 INFO mapreduce.Job: Counters: 32
File System Counters
  FILE: Number of bytes read=22
  FILE: Number of bytes written=229427
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=0
  HDFS: Number of bytes written=0
  HDFS: Number of read operations=7
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=3
Map-Reduce Framework
  Combine input records=0
  Combine output records=0
  Reduce input groups=0
  Reduce shuffle bytes=0
  Reduce input records=0
  Reduce output records=0
  Spilled Records=0
  Shuffled Maps =0
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=0
  CPU time spent (ms)=0
  Physical memory (bytes) snapshot=0
  Virtual memory (bytes) snapshot=0
  Total committed heap usage (bytes)=60686336
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=0
```

刷新 DFS Location 后也能看到输出的 output 文件夹。双击 part-r-000000 文件，可以看到程序运行的结果，由于指导书中未指出 input 的文件，所以我们的 input 是空的，output 自然也是空的，具体的、有实际 input 的 WordCount 运行实例将在实验 3 中体现。



三、实验结果分析、思考题解答：

对于实验二，我在中间遇到了不少问题，比如没有输出，或是 Eclipse 导入包的一些问题，但是只要多查，多想，我发现没有什么问

题是前人没有遇到过的，只要你善于去检索，只要不是很前沿的问题都能搜得到，就算你到最后也没配置成功，可你也会学到不少东西，而且实验越往后做，又可能会触类旁通的解决之前你遇到的问题。

四、感想、体会、建议：

在云计算及其相关应用日益火爆的今天，我们通过课堂上老师讲述的云计算相关的基本知识,已经对于利用 Hadoop 的配置实现一个较为简单的云计算环境有了一定的认识，因此,在本课程的实验中，我们采用 Linux 下的 Hadoop 搭建来运行一些简单的程序比如 Wordcount 来达成对云计算的实现的理论知识的学习效果；

也让我对课堂上老师提到的：“MapReduce 将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数：Map 和 Reduce。编程容易，不需要掌握分布式并行编程细节，也可以很容易把自己的程序运行在分布式系统上，完成海量数据的计算。MapReduce 采用“分而治之”策略，一个存储在分布式文件系统的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个 Map 任务并行处理。MapReduce 设计的一个理念就是“计算向数据靠拢”，而不是“数据向计算靠拢”，因为，移动数据需要大量的网络传输开销。MapReduce 框架采用了 Master/Slave 架构，包括一个 Master 和若干个 Slave。Master 上运行 JobTracker，Slave 上运行 TaskTracker。Hadoop 框架是用 Java 实现的,但是,MapReduce 应用程序则不一定要用 Java 来写。”有了更深刻的理解。

但是值得注意的一点是若要使用本实验中的 MapReduce 来处理数

据集(或任务)，其必须具备如下特点:待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。

在后续的实验三中，我们将会进一步进行实操，对 MapReduce 进一步的熟悉。

实验成绩：

指导教师签名：

年 月 日