实验六

Hadoop 词频统计实验

一、实验准备：环境与文件创建

1. 环境确认

2. 创建词频文件

二、核心命令操作（对应 Q1-Q3）

1. Q1：启动 Hadoop

2. Q2：上传文件到 HDFS 的 input 文件夹

3. Q3：检查并删除 HDFS 中的 output 目录

三、编写 MapReduce 程序（对应 Q4-Q8，基于 Eclipse）

1. 环境配置（无插件方案，通用）

2. 编写核心代码（WordCount. java）

3. 代码逻辑解释（对应 Q4-Q8）

四、编译打包与运行程序

1. 打包 JAR 文件

2. 运行 MapReduce 程序

3. 查看运行结果

五、实验收尾

一、Hadoop 词频统计实验

1.1 一、实验准备：环境与文件创建

1.1.1 1. 环境确认

确保已在 **Ubuntu 系统** 中安装 Hadoop，并配置好环境变量（可通过在终端输入 `hadoop version` 验证，能显示版本则配置正常）；Windows 系统主要用于 Eclipse 开发（若 Ubuntu 装了 Eclipse 也可直接用）。

1.1.2 2. 创建词频文件

在 Ubuntu 终端中，通过以下命令创建 `wordfile1.txt` 和 `wordfile2.txt`（文件内容需固定，用于后续词频统计）：

```
#创建并编辑wordfile1.txt，输入指定内容
vim wordfile1.txt
#按i进入编辑模式，输入：
Hello Hadoop
Hello Hdfs
Hello MapReduce
#按Esc，输入:wq保存退出
#同理创建wordfile2.txt
vim wordfile2.txt
#输入内容：
Hadoop is good
Hdfs is distributed
MapReduce is efficient
```

1.2 二、核心命令操作（对应 Q1-Q3）

所有命令需在 Ubuntu 终端中执行，且确保当前用户有 Hadoop 操作权限（建议切换到 Hadoop 安装用户，如 `su hadoop`）。

1.2.1 1. Q1：启动 Hadoop

Hadoop 启动需先初始化（首次启动时），再启动集群，命令如下：

```
#启动Hadoop集群（包含NameNode、DataNode、ResourceManager等）
start-dfs.sh  # 启动分布式文件系统（HDFS）
start-yarn.sh # 启动资源调度系统（YARN）
#验证是否启动成功：输入jps，能看到NameNode、DataNode、ResourceManager、NodeManager则正常
jps
```

1.2.2 2. Q2：上传文件到 HDFS 的 input 文件夹

HDFS 中默认没有 `input` 文件夹，需先创建，再上传本地文件：

```
#1. 在HDFS根目录创建input文件夹
hdfs dfs -mkdir /input
#2. 上传本地的wordfile1.txt和wordfile2.txt到HDFS的/input目录
（注意：需替换命令中的"本地文件路径"为实际路径，如\~/wordfile1.txt，\~代表当前用户家目录）
hdfs dfs -put \~/wordfile1.txt /input
hdfs dfs -put \~/wordfile2.txt /input
#验证上传：查看HDFS input文件夹下的文件
hdfs dfs -ls /input
```

1.2.3 3. Q3：检查并删除 HDFS 中的 output 目录

MapReduce 运行时要求输出目录（如 output）**不存在**，否则会报错，需先检查并删除：

```
#1. 检查HDFS根目录是否存在output目录
hdfs dfs -test -e /output && echo "output存在" || echo "output不存在"
#2. 若存在，删除output目录（-r表示递归删除，包含子目录和文件）
hdfs dfs -rm -r /output
#验证删除：再次查看是否存在
hdfs dfs -ls /
```

1.3 三、编写 MapReduce 程序（对应 Q4-Q8，基于 Eclipse）

1.3.1 1. 环境配置（无插件方案，通用）

① 在 Eclipse 中创建**Java Project**（如命名为 WordCountProject）。

② 添加 Hadoop 依赖 JAR 包：

- 找到 Ubuntu 中 Hadoop 的 `share/hadoop` 目录（如 `/usr/local/hadoop/share/hadoop`），将以下目录下的所有 JAR 包导入项目：
  - `common`、`common/lib`、`hdfs`、`hdfs/lib`、`mapreduce`、`mapreduce/lib`、`yarn`、`yarn/lib`。
- 导入方式：右键项目→Build Path→Add External Archives→选中上述 JAR 包。

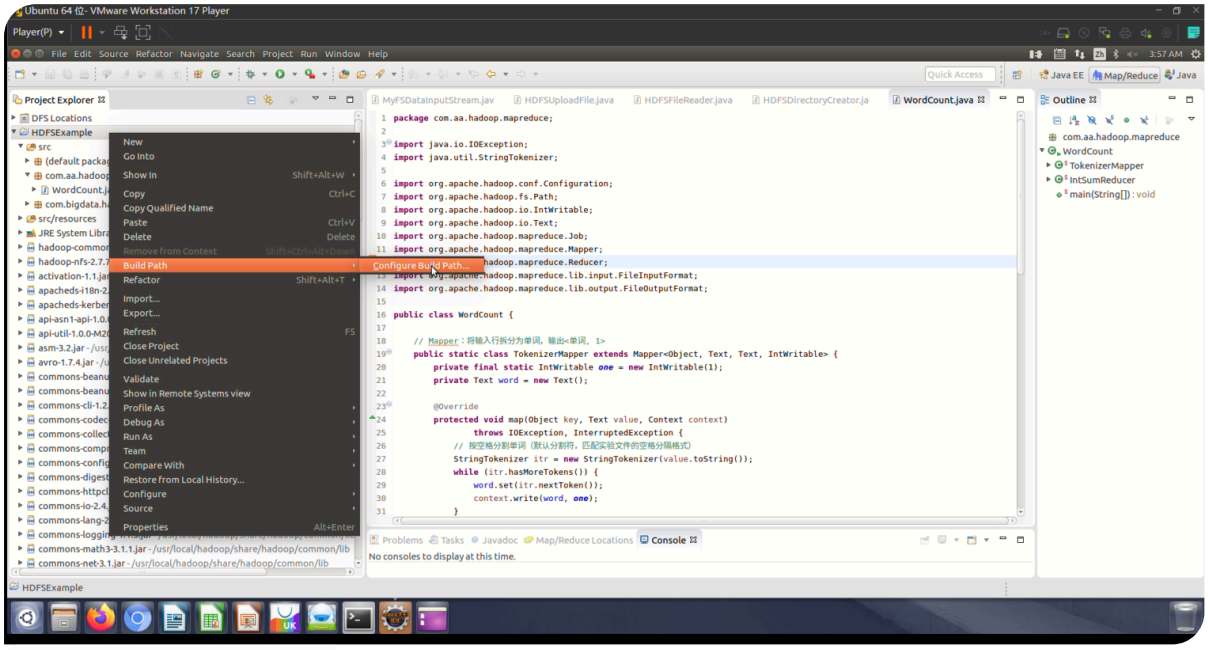

图 1.3.1-1 导入方式-1

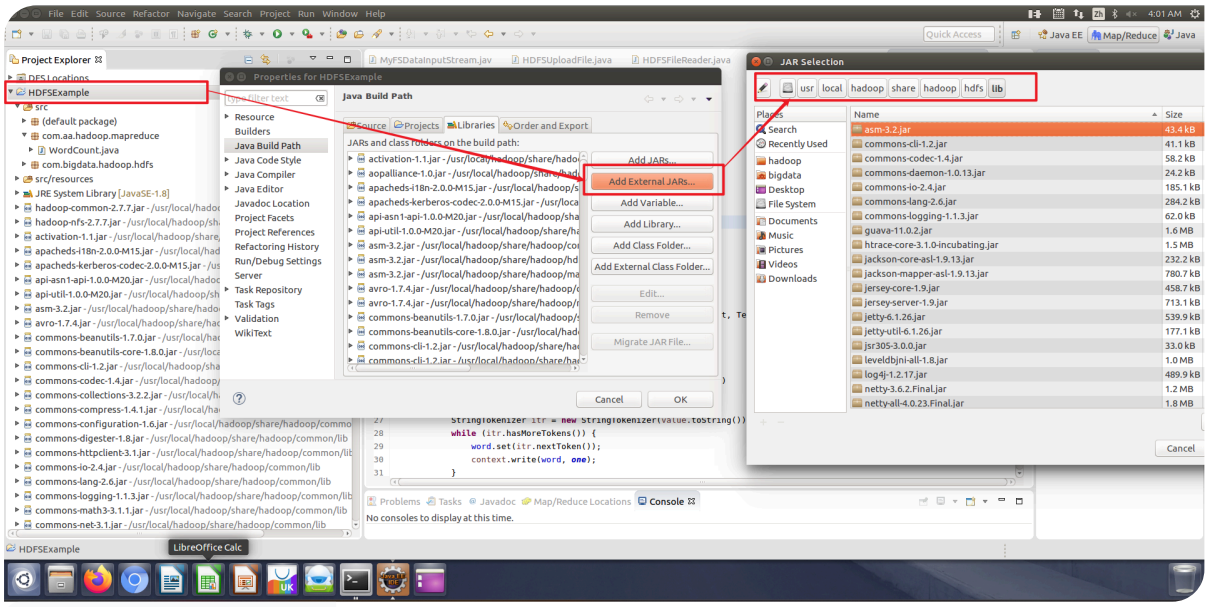

图 1.3.1-2 导入方式-2

1.3.2 2. 编写核心代码（WordCount. java）

在项目的 `src` 目录下创建 `com.aa.mapreduce.wordcount` 包，新建 `WordCount.java` 文件，代码如下（已包含 Map、Reduce、Main 逻辑）：

```java
package com.aa.hadoop.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;

public class WordCount {

    // Mapper: 将输入行拆分为单词，输出<单词, 1>
    public static class TokenizerMapper extends Mapper<Object, Text,
Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        protected void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {
            // 按空格分割单词（默认分割符，匹配实验文件的空格分隔格式）
            StringTokenizer itr = new
StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    // Reducer: 汇总同一单词的计数，输出<单词, 总次数>
    public static class IntSumReducer extends Reducer<Text,
IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        @Override
        protected void reduce(Text key, Iterable<IntWritable> values,
Context context)
                throws IOException, InterruptedException {
            int sum = 0;
            // 累加同一单词的所有计数
            for (IntWritable val : values) {
```

```
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    // 主方法：配置Job并提交任务
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        // 显式指定HDFS地址（与你的Hadoop配置一致，避免路径歧义）
        conf.set("fs.defaultFS", "hdfs://localhost:9000");

        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        // 配置Mapper、Reducer类
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class); // Combiner优化：本地先汇总，减少网络传输
        job.setReducerClass(IntSumReducer.class);
        // 配置输出键值对类型（与Mapper输出、Reducer输出一致）
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // 关键修改：仅添加两个目标文件作为输入（排除其他无关文件）
        FileInputFormat.addInputPath(job, new Path("/input/wordfile1.txt"));
        FileInputFormat.addInputPath(job, new Path("/input/wordfile2.txt"));
        // 输出路径：必须是HDFS中不存在的目录（每次运行前需删除旧目录）
        FileOutputFormat.setOutputPath(job, new Path("/output_wordcount_final"));

        // 提交Job并等待完成（0表示成功，1表示失败）
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

1.3.3 3. 代码逻辑解释（对应 Q4-Q8）

问题	答案（基于上述代码）
Q4	**key**：输入文本的“行偏移量”（即当前行在整个文件中的起始字节位置，如第 0 行偏移量为 0，第 1 行偏移量为 15 等，类型为 Object）；**value**：输入文本的“一行内容”（如 `Hello Hadoop`，类型为 Text）。

问题	答案（基于上述代码）
Q5	**itr**：`StringTokenizer` 对象，存储对“当前行文本（value）按空白字符拆分后的所有单词”（如对 `Hello Hadoop` 拆分后，itr 包含 `Hello` 和 `Hadoop` 两个元素）。
Q6	**word**：存储“当前遍历到的单个单词”（如循环中每次从 itr 取出的 `Hello`、`Hadoop` 等，类型为 Text）；**one**：固定值 `1`（类型为 IntWritable），代表当前单词在该行出现 1 次。
Q7	**key**：“去重后的单个单词”（Map 输出后会按 key 分组，如所有 `Hello` 会被分到同一组，key 为 `Hello`，类型为 Text）；**values**：“同一单词对应的所有 value 集合”（即多个 `1`，类型为 Iterable，如 `Hello` 出现 3 次，values 包含 3 个 `1`）。
Q8	**result**：“同一单词的总出现次数”（将 values 中的所有 `1` 累加后的结果，如 `Hello` 累加后为 3，类型为 IntWritable）。

1.4 四、编译打包与运行程序

1.4.1 1. 打包 JAR 文件

① 在 Eclipse 中右键项目→Export→Java→JAR file→Next。

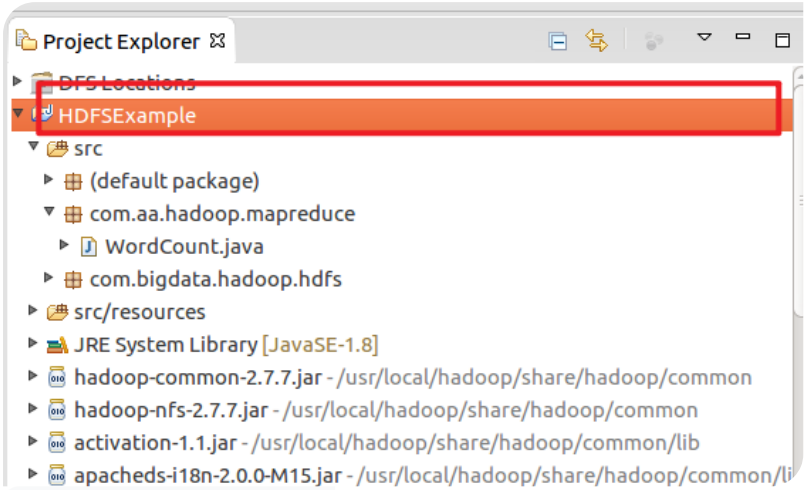

图 1.4.1-1 编译打包-1

② 选择要导出的 `WordCount.java`（确保勾选 `src` 目录下的包和类），指定输出路径（如 Ubuntu 的 `/usr/local/hadoop/myapp/WordCount.jar`），点击 Finish。

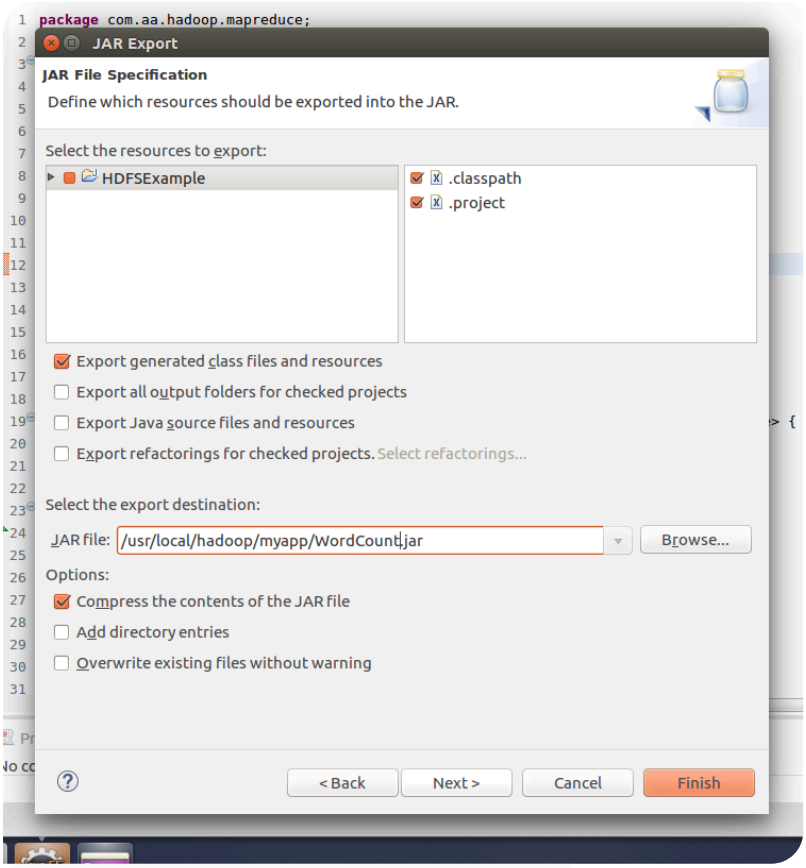

图 1.4.1-2 image-20251030190716028

③ 将打包好的 JAR 文件复制到 Ubuntu 的 `/usr/local/hadoop/myapp/` 目录（若 `myapp` 不存在，先通过 `mkdir /usr/local/hadoop/myapp` 创建）。

1.4.2 2. 运行 MapReduce 程序

在 Ubuntu 终端中执行以下命令，提交 Job 到 Hadoop 集群：

```
#切换到Hadoop安装目录（如/usr/local/hadoop）

cd /usr/local/hadoop

#运行JAR包，格式：hadoop jar JAR路径 主类全限定名 输入路径 输出路径

hadoop jar /usr/local/hadoop/myapp/WordCount.jar
com.aa.hadoop.mapreduce.WordCount /input/wordfile1.txt
/input/wordfile2.txt /output_wordcount_final


```

1.4.3 3. 查看运行结果

程序运行成功后，HDFS 会生成 `output` 目录，包含结果文件 `part-r-00000`，通过以下命令查看：

```
#查看output目录下的文件
hdfs dfs -ls /output

#执行下面的命令查看结果
hdfs dfs -cat /output_wordcount_final/part-r-00000
```


结果：

```
Hadoop  2
Hdfs    2
Hello   3
MapReduce   2
distribute  1
efficient   1
good    1
is  3
```

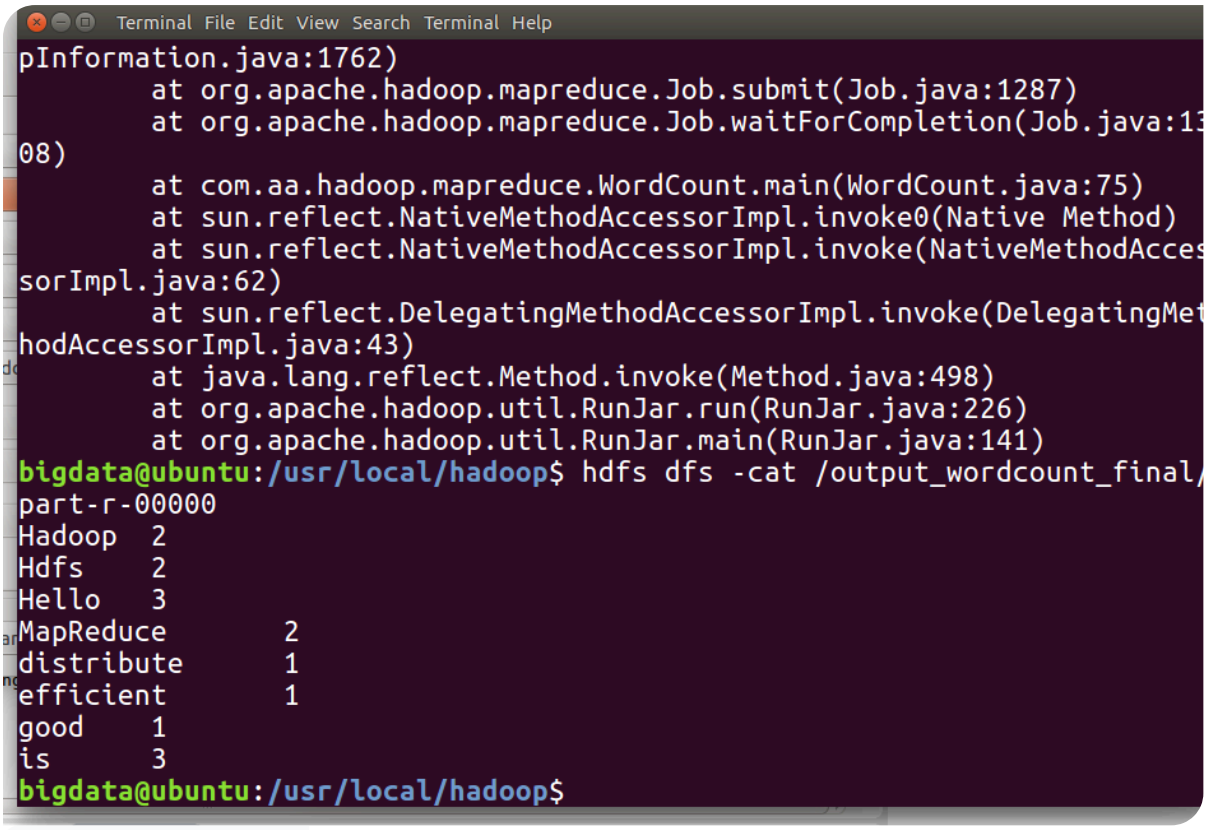

图 1.4.3-1 运行结果

1.5 五、实验收尾

① 运行完成后，可停止 Hadoop 集群：

```
stop-yarn.sh   # 停止YARN
stop-dfs.sh    # 停止HDFS
```