



大数据技术基础课程实验报告

实验二：实践 MapReduce 分布式数据处理

付容天

学号 2020211616

班级 2020211310

计算机学院（国家示范性软件学院）

2023 年 3 月 16 日

实验过程与分析

在这部分的实验中，我按照实验指导书，先后完成了：

- (1) 打开 IDEA 创建项目，并创建 Maven 工程；
- (2) 依赖设置，比如 pom.xml 文件中的 properties 配置项等；
- (3) 设置语言环境，保证 Language Level 为 8；
- (4) 设置 Java Compiler 环境，使 Project Bytecode Version 为 1.8；
- (5) WordCount 程序编写，并将程序打包运行，生成 jar 包。

得到结果如下：

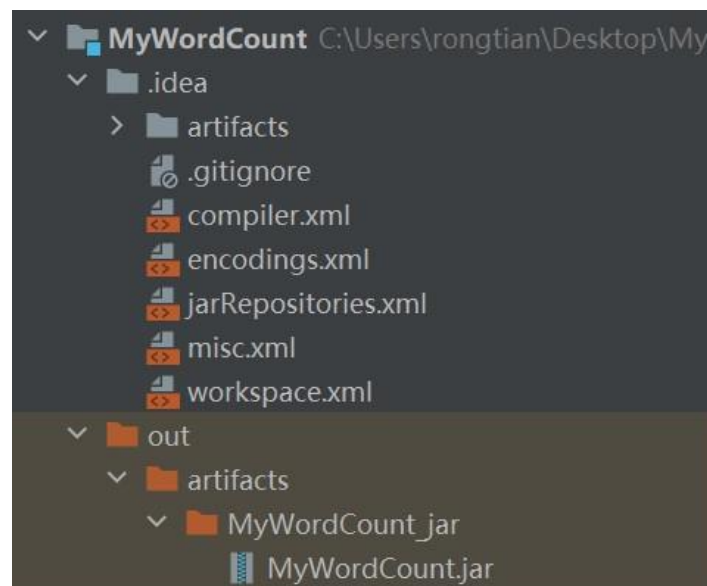


图 1：工程目录（对应指导书图 28）

上图 1 中显示了项目的结构，.idea 文件夹中是 IDEA 配置文件，是在创建 maven 项目时自动创建的。out 文件夹则存放了项目的运行结果，在本项目中就是存放了 MyWordCount 相关的 jar 包。

然后，再接着进行下面的操作：

- (6) 在压缩软件中打开 jar 包，删除其中的/META-INF/MANIFEST.MF 文件；
- (7) 使用 WinSCP 将 jar 包上传到服务器；
- (8) 在本地新建 2020211616-frt-input.txt 文件，也上传到服务器；
- (9) 使用 put 命令将 input 文件传到 hdfs 中；
- (10) 执行如下图 2 的指令，得到相应的输出如下图 3 所示。

```
root@frt-2020211616-0001:~# hadoop jar MyWordCount.jar WordCount /testmr/2020211616-frt-input.txt /testmr/2020211616-frt-output
23/03/15 21:58:24 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
23/03/15 21:58:25 INFO client.RMProxy: Connecting to ResourceManager at node1/192.168.0.34:8032
23/03/15 21:58:26 INFO input.FileInputFormat: Total input paths to process : 1
23/03/15 21:58:26 INFO mapreduce.JobSubmitter: number of splits:1
23/03/15 21:58:26 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1678887305213_0002
23/03/15 21:58:26 INFO impl.YarnClientImpl: Submitted application application_1678887305213_0002
```

图 2：执行 jar 包（对应指导书图 33）

```

root@frt-2020211616-0001:~
GC time elapsed (ms)=123
CPU time spent (ms)=1040
Physical memory (bytes) snapshot=355336192
Virtual memory (bytes) snapshot=2581463040
Total committed heap usage (bytes)=173080576
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=157
File Output Format Counters
Bytes Written=65

```

图 3: 执行结果 (对应指导书图 34)

上图 2 中划红线处是在 hadoop 上执行 jar 包的命令, 其含义是运行 jar 包 MyWordCount.jar, 对应的输入文件为 2020211616-frt-input.txt, 输出则为 2020211616-frt-out (文件夹)。输入和输出均在文件夹/testmr 下。通过查看 output 文件夹, 可以发现结果存放在 part-r-00000 中。

上图 3 为执行结果, 可以看到 CPU 时间为 1040 毫秒、各项错误均为 0。读入的字节数为 157 (来自 2020211616-frt-input.txt 文件), 输出的字节数为 65 (写入到了 part-r-00000 文件中)。

最后进行:

(11) 在 hdfs 上查看结果 (注意 output 是一个文件夹), 结果如下图 4 所示。

```

root@frt-2020211616-0001:~
Error: No command named '-cat' was found. Perhaps you meant 'hadoop cat'
[root@frt-2020211616-0001 ~]# hadoop fs -cat /testmr/2020211616-frt-output/part-r-00000
23/03/15 22:00:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for y
2020211616      3
dog      3
fish     3
frt      3
hadoop   3
hello    3
spark    3
world    3
[root@frt-2020211616-0001 ~]# _

```

图 4: 运行结果 (对应指导书图 35)

WordCount 代码解释

现在来对 Java 源代码进行简要解释和分析, 主要关注三个类的作用以及类之间的关系。WordCount 源代码如下图所示:

```

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;
import java.io.IOException;
import java.util.StringTokenizer;

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> <out>");
            System.exit(2);
        }

        Job = new Job(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);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

在上面的代码中，主要部分的分析与用处简析如下：

- (1) WordCount 类：单词计数类，这个例子是初学 hadoop 的常见例子，可以帮助我们更好地了解和学习 hadoop。在本次实验中，WordCount 类中有 TokenizerMapper 类、IntSumReducer 类和 main 方法三个部分；
- (2) TokenizerMapper 类：该类的基础为 Mapper 类（hadoop 框架提供的基本类，在头文件 org.apache.hadoop.mapreduce 中），该类中的 map 方法负责实现 hadoop 单词计数功能的 map 阶段：生成“(word, 1)”结构；
- (3) IntSumReducer 类：该类的基础为 Reducer 类（hadoop 框架提供的基本类，在头文件 org.apache.hadoop.mapreduce 中），该类中的 reduce 方法（传入的参数为 map 阶段生成的 context）负责将 map 阶段生成的“(word, 1)”结构合并为计数结果；
- (4) Main 方法中首先进行了 conf 定义，Configuration 是 hadoop 中五大组件的公用类（在头文件 org.apache.hadoop.conf 中），该类是作业配置信息类，任何作用配置必须经过 Configuration 来传递（Configuration 可以实现多个 mapper 和 reducer 任务之间的信息共享）。然后进行作业 job 的定义，并将 job 中 mapper 和 reducer 等内容设置为我们自己编写的 TokenizerMapper 类和 IntSumReducer 类，最后设置好输入输出格式即可。