

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3 统计某电商平台网站买家收藏商品数量

实验时间： 2021 年 4 月 25 日

预习内容

一、实验目的和要求：

- 1) 了解多台主机虚拟化以及 Hadoop 的联合部署；
- 2) 掌握基本的云计算雏形应用；
- 3) 熟悉 MapReduce 的执行过程；

二、实验任务：

现有某电商网站用户对商品的收藏数据，记录了用户收藏的商品 id 以及收藏日期，名为 buyer_favorite1。buyer_favorite1 包含：买家 id，商品 id，收藏日期这三个字段，数据以“\t”分割，样本数据及格式如下：

1.	买家 id	商品 id	收藏日期
2.	10181	1000481	2010-04-04 16:54:31
3.	20001	1001597	2010-04-07 15:07:52
4.	20001	1001560	2010-04-07 15:08:27
5.	20042	1001368	2010-04-08 08:20:30
6.	20067	1002061	2010-04-08 16:45:33
7.	20056	1003289	2010-04-12 10:50:55
8.	20056	1003290	2010-04-12 11:57:35
9.	20056	1003292	2010-04-12 12:05:29
10.	20054	1002420	2010-04-14 15:24:12
11.	20055	1001679	2010-04-14 19:46:04
12.	20054	1010675	2010-04-14 15:23:53
13.	20054	1002429	2010-04-14 17:52:45
14.	20076	1002427	2010-04-14 19:35:39
15.	20054	1003326	2010-04-20 12:54:44
16.	20056	1002420	2010-04-15 11:24:49
17.	20064	1002422	2010-04-15 11:35:54
18.	20056	1003066	2010-04-15 11:43:01
19.	20056	1003055	2010-04-15 11:43:06

20.	20056	1010183	2010-04-15 11:45:24
21.	20056	1002422	2010-04-15 11:45:49
22.	20056	1003100	2010-04-15 11:45:54
23.	20056	1003094	2010-04-15 11:45:57
24.	20056	1003064	2010-04-15 11:46:04
25.	20056	1010178	2010-04-15 16:15:20
26.	20076	1003101	2010-04-15 16:37:27
27.	20076	1003103	2010-04-15 16:37:05
28.	20076	1003100	2010-04-15 16:37:18
29.	20076	1003066	2010-04-15 16:37:31
30.	20054	1003103	2010-04-15 16:40:14
31.	20054	1003100	2010-04-15 16:40:16

要求编写 MapReduce 程序，统计每个买家收藏商品数量，并撰写实验报告。

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，以便使用 Hadoop 脚本管理远端 Hadoop 守护进程。

框架：

hadoop-2.5.2

Eclipse

实验内容

一、实验用仪器、设备：

实验用设备：

MECHREVO 微机 DESKTOP-6A05AKM

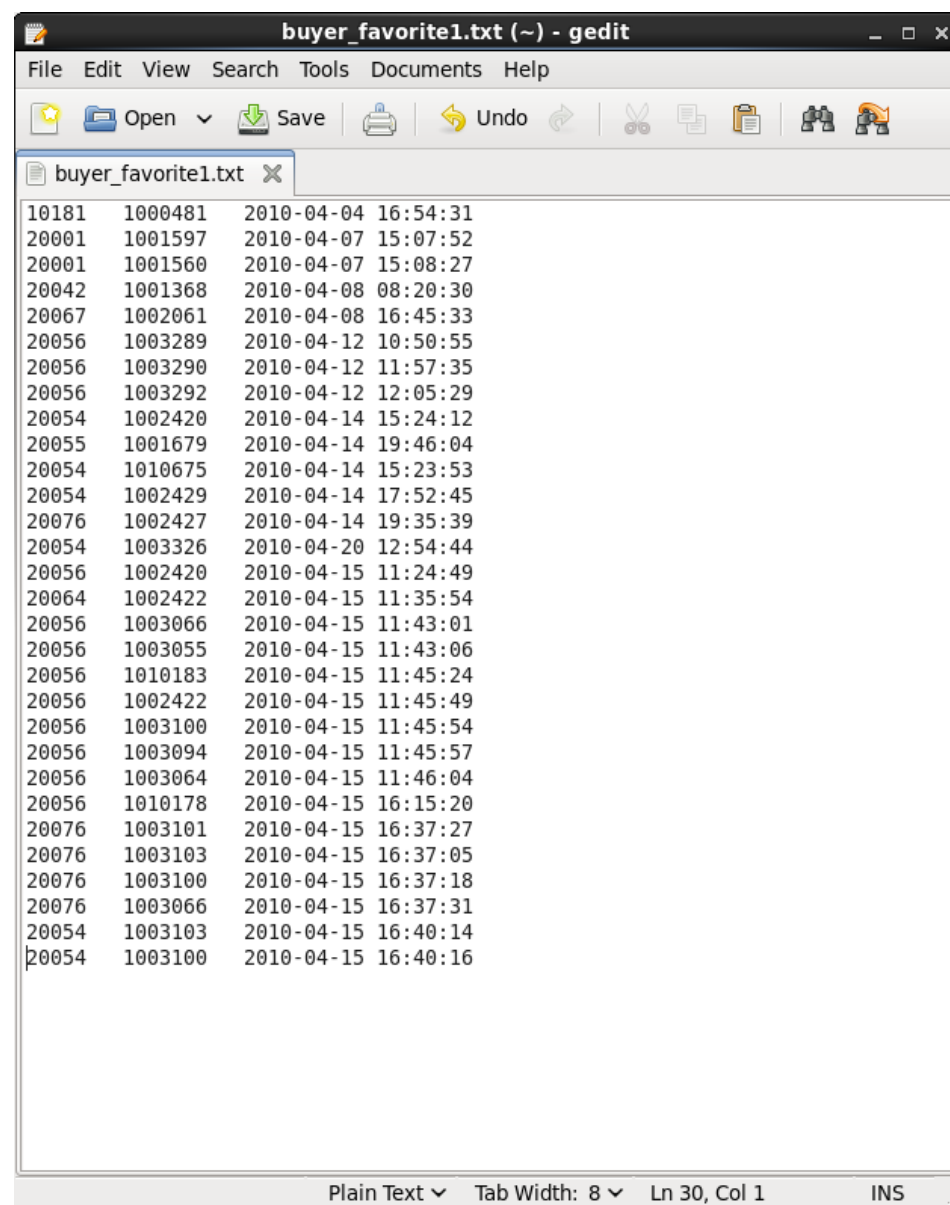
处理器 Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.20 GHz

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）：

首先我们在虚拟机上的/home/hfut/test3 位置创建新文本 buyer_favorite1.txt，
buyer_favorite1.txt 的内容如下：



The screenshot shows a gedit text editor window titled "buyer_favorite1.txt (~) - gedit". The window contains a list of data entries, each consisting of three columns: a number, a code, and a date-time stamp. The entries are as follows:

10181	1000481	2010-04-04 16:54:31
20001	1001597	2010-04-07 15:07:52
20001	1001560	2010-04-07 15:08:27
20042	1001368	2010-04-08 08:20:30
20067	1002061	2010-04-08 16:45:33
20056	1003289	2010-04-12 10:50:55
20056	1003290	2010-04-12 11:57:35
20056	1003292	2010-04-12 12:05:29
20054	1002420	2010-04-14 15:24:12
20055	1001679	2010-04-14 19:46:04
20054	1010675	2010-04-14 15:23:53
20054	1002429	2010-04-14 17:52:45
20076	1002427	2010-04-14 19:35:39
20054	1003326	2010-04-20 12:54:44
20056	1002420	2010-04-15 11:24:49
20064	1002422	2010-04-15 11:35:54
20056	1003066	2010-04-15 11:43:01
20056	1003055	2010-04-15 11:43:06
20056	1010183	2010-04-15 11:45:24
20056	1002422	2010-04-15 11:45:49
20056	1003100	2010-04-15 11:45:54
20056	1003094	2010-04-15 11:45:57
20056	1003064	2010-04-15 11:46:04
20056	1010178	2010-04-15 16:15:20
20076	1003101	2010-04-15 16:37:27
20076	1003103	2010-04-15 16:37:05
20076	1003100	2010-04-15 16:37:18
20076	1003066	2010-04-15 16:37:31
20054	1003103	2010-04-15 16:40:14
20054	1003100	2010-04-15 16:40:16

编辑好后，保存即可。

源代码如下所示：

```
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();

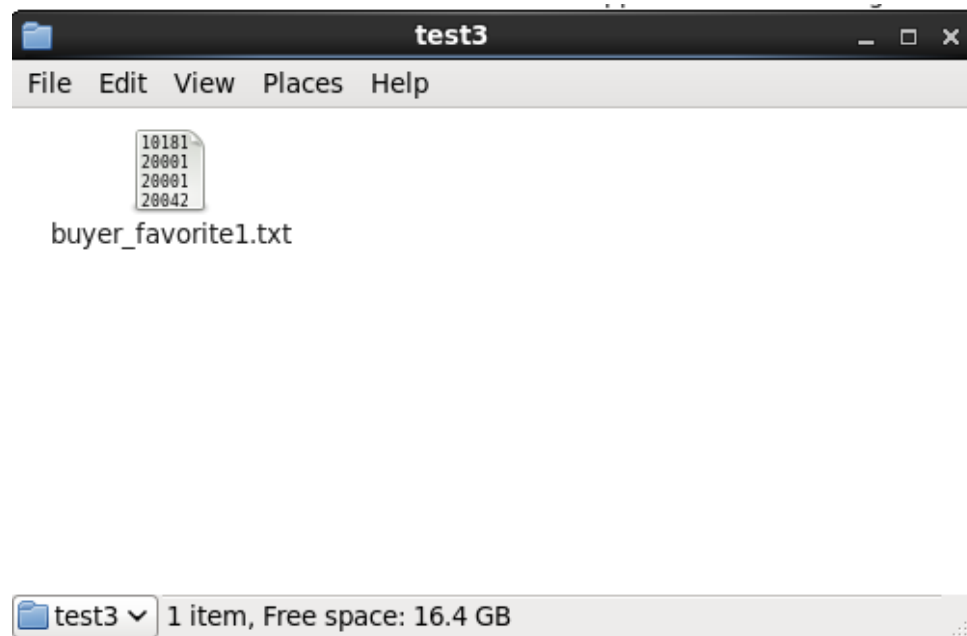
        public TokenizerMapper() {
        }

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

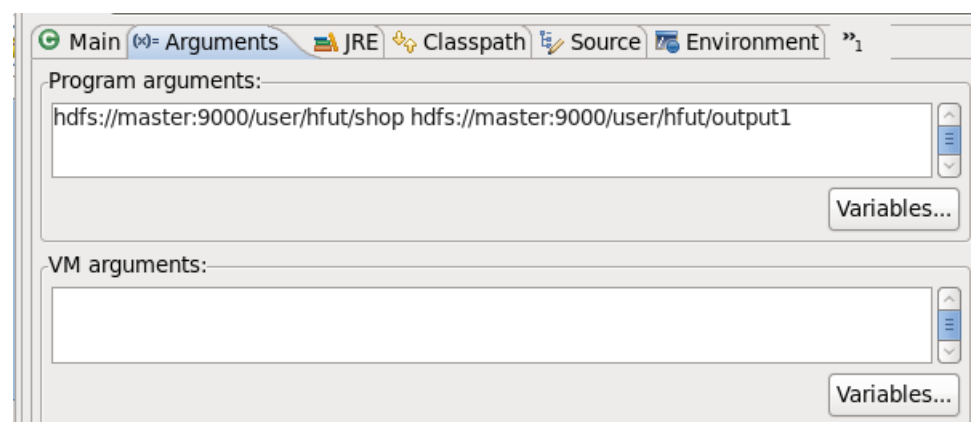
            while(itr.hasMoreTokens()) {
                this.word.set(itr.nextToken());
                context.write(this.word, one);
            }
        }
    }
}

```

右键点击 WordCountTest.java，选择 Run As -> Run Configurations，修改运行时的相关参数（同实验八原理，根据我们之前的设置的路径如下可知，我们的 input 和 output 应该如下所示。）



hdfs://master:9000/user/hfut/shop hdfs://master:9000/user/hfut/output1



刷新 DFS Location 后能看到输出的 output1 文件夹。双击 part-r-00000 文件，预期是可以看到程序运行的结果：

```
10181 1
20001 2
20042 1
20054 6
20055 1
20056 12
20064 1
20067 1
20076 5
```


但是实际上没有，原因很简单，就是没有把我们的 txt 文件成功导入到 DFS 的 input 中去，而是在/test3 中，所以我们在 root 下运行指令

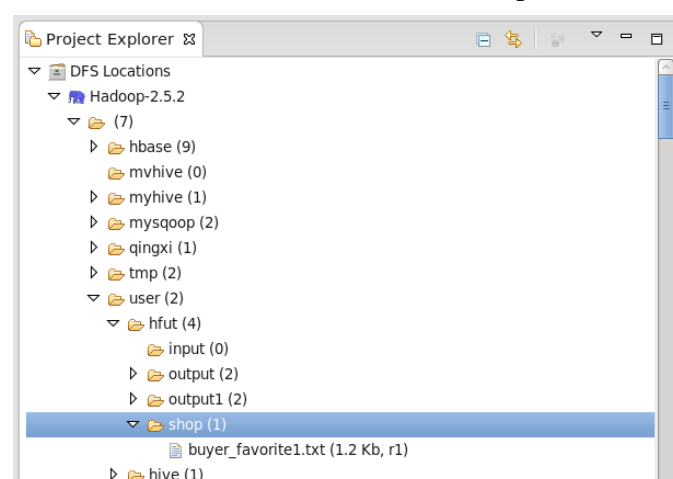
```
[root@master hfut]# hadoop fs -put buyer_favorite1.txt /user/hfut/shop
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hfut/hadoop-2.5.2/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hfut/hbase-1.1.2/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
21/05/04 23:41:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
put: Permission denied: user=root, access=WRITE, inode="/user/hfut/shop":hfut:supergroup:drwxr-xr-x
[root@master hfut]#
```

但是上传文件失败了，检索了资料，原因是/user 这是文件的所有者是 HDFS 权限，也就是只有 HDFS 才能对这个文件进行 sudo 的操作，那么接下来我们便可以使用其他办法：

那么我们就使用最简单的指令完成：

```
[hfut@master ~]$ hadoop fs -moveFromLocal test3/buyer_favorite1.txt /user/hfut/shop
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hfut/hadoop-2.5.2/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hfut/hbase-1.1.2/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
21/05/04 23:58:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[hfut@master ~]$
```

然后我们检查 hdfs 中的/user/hfut/shop



成功！

PS：这里我们应该用 `hdfs dfs -copyFromLocal` 的拷贝指令的，这样的话本地文件中还能保留一份副本。

实验进行到这里就变得明朗了，我们先删除之前的空文件夹 `output1`，若不删除该目录，程序会在下一次执行 `WordCount` 程序时报错

然后直接运行 `java` 程序，控制台情况如下：

```
<terminated> WordCountTest [Java Application] /home/hfut/eclipse/jre/bin/java (May 5, 2021 12:03:37 AM)
FILE: Number of bytes written=433701
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=2402
HDFS: Number of bytes written=735
HDFS: Number of read operations=15
HDFS: Number of large read operations=0
HDFS: Number of write operations=4
Map-Reduce Framework
  Map input records=31
  Map output records=120
  Map output bytes=1500
  Map output materialized bytes=1019
  Input split bytes=118
  Combine input records=120
  Combine output records=70
  Reduce input groups=70
  Reduce shuffle bytes=1019
  Reduce input records=70
  Reduce output records=70
  Spilled Records=140
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=1769
  CPU time spent (ms)=0
  Physical memory (bytes) snapshot=0
  Virtual memory (bytes) snapshot=0
  Total committed heap usage (bytes)=331096064
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=1201
File Output Format Counters
  Bytes Written=735
```

我们去检查一下 `output1`

结果如下所示，原来是我忘记了修改 java 代码。

hdfs://master:9000/user/hfut/output1/part-r-00000

08:20:30 1

1000481 1

1001368 1

1001560 1

1001597 1

1001679 1

1002061 1

1002420 2

1002422 2

1002427 1

1002429 1

1003055 1

1003064 1

1003066 2

1003094 1

1003100 3

1003101 1

1003103 2

1003289 1

1003290 1

1003292 1

1003326 1

1010178 1

1010183 1

1010675 1

10181 1

10:50:55 1

11:24:49 1

11:35:54 1

11:43:01 1

11:43:06 1

11:45:24 1

11:45:49 1

11:45:54 1

11:45:57 1

11:46:04 1

11:57:35 1

12:05:29 1

12:54:44 1

15:07:52 1

15:08:27 1

15:23:53 1

15:24:12 1

所以我们接下来根据 buyer_favorite1.txt 的格式可以看出,我们需要的数据只在第一列,其他列的数据是不需要记录的,那么遍历每行,只取第一列即可,其他就 break 循环,进入下一行,所以正确的源代码应该是:

```

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());
        int i = 0;

        while(itr.hasMoreTokens()) {
            i++;
            if(i==1){
                this.word.set(itr.nextToken());
                context.write(this.word, one);
                break;
            }
            itr.nextToken();
        }
    }
}

```

执行结果如下：

```
Problems @ Javadoc Declaration Console
<terminated> WordCountTest [Java Application] /home/hfut/eclipse/jre/bin/java (May 5, 2021 1:03:56 AM)
FILE: Number of bytes written=45510
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=2402
HDFS: Number of bytes written=73
HDFS: Number of read operations=15
HDFS: Number of large read operations=0
HDFS: Number of write operations=4
Map-Reduce Framework
Map input records=31
Map output records=30
Map output bytes=300
Map output materialized bytes=114
Input split bytes=118
Combine input records=30
Combine output records=9
Reduce input groups=9
Reduce shuffle bytes=114
Reduce input records=9
Reduce output records=9
Spilled Records=18
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=47
CPU time spent (ms)=0
Physical memory (bytes) snapshot=0
Virtual memory (bytes) snapshot=0
Total committed heap usage (bytes)=331096064
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=1201
File Output Format Counters
Bytes Written=73
```

运行结束后，我们打开 output1 查看结果，正确！

hdfs://master:9	*WordCountTest.	hdfs://master:9	hdfs://master:9	hdfs://master:9
10181	1			
20001	2			
20042	1			
20054	6			
20055	1			
20056	12			
20064	1			
20067	1			
20076	5			

三、实验结果分析、思考题解答：

实验主要的问题就是运行程序后结果一直没有变化，花费了一点时间将其解决，具体解决方案都很详尽的写在了上面的实验报告内。

本实验使我进一步熟悉了如何将 Java 文件部署到 HDFS 上，如何在 Hadoop 上运行

MapReduce，更让我们自主的去写一个大数据架构内的 Java 代码，这还是第一次，我想，这也为我未来的大数据学习打下了坚实的基础。


四、感想、体会、建议：

实验三的代码与实验二唯一不同之处在于下面的代码，也就是只取第一列数据累加即可。

```
public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    int i = 0;

    while(itr.hasMoreTokens()) {
        i++;
        if(i==1){
            this.word.set(itr.nextToken());
            context.write(this.word, one);
            break;
        }
        itr.nextToken();
    }
}
```

现在是凌晨 1:11，我终于把三个实验都完成了，因为自己有点执着，总要求着自己完完整整，一步不漏的把 19 个综合实验写完，进而对整个实验有一个俯视的理解，再去写三个小实验，所以课时内一直在完成综合实验，直到 5 月 2 日才把 19 个综合实验给完成了（如下），一共写了快 300 页。

	综合设计报告-孙淼.docx 作者: 孙 淼	修改日期: 2021/5/4 23:29 大小: 31.1 MB
---	----------------------------------	-------------------------------------

所以我 5 月 3 日，5 月 4 日才开始做三个主实验，不过有综合实验的铺垫，做这三个小实验也就不算什么难事了，一天 10 个小时的工作量，两天就完成了，也就相当于每个实验连做带写报告也就 7 个小时左右。并不是很困难，但是实验过程很有趣，解决问题的刹那更是心情舒畅，总体的实验安排也很合理，在有限的课时内层次分明地尽可能的包揽了主流的大数据相关的框架、工具和技术，让我们算是进入了大数据技术的大门了，也让我度过了一个比以往要充实的多的劳动节假期。

实验成绩：

指导教师签名：

年 月 日