**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：赵则橦**

**学 号： 2017221305013**

**指导教师：黄俊**

**实验地点：信软楼**

**实验时间：2019/11**

**一、 实验名称：**反向索引

**二、** **实验学时：**4学时

**三、 实验目的：**

1. 熟悉反向索引；

2. 加强 MapReduce 编程能力

**四、 实验原理**

反向索引（Inverted Index），也常被称为倒排索引，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。它是文档检索系统中最常用的数据结构。

文本：

T0：it is what it is

T1：what is it

T2：it is a banana

得到下面的反向文件索引：

a:{2}

banana:{2}

is:{0, 1, 2}

it:{0, 1, 2}

what:{0, 1}

**五、 实验内容**

1） 打开 Eclipse

2） 编写好 txt 文件，把他复制到 hdfs 上作为输入

3） 在 Eclipse 中创建 Inverted Index 项目

4） 在 Eclipse 中编写 Inverted Index 项目

5） 编写 TokenInputFormat.java

6） 编写 ValuePair.java

7） 编写 InvertedIndex.java

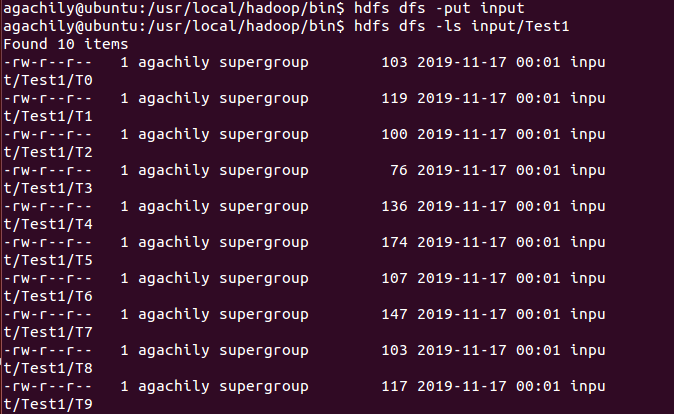
8） 运行程序。

**六、 实验设备及环境**

Ubuntu jdk 1.8 hadoop 2.7.X

**七、 实验步骤**

1. 打开 Eclipse
2. 复制 txt 文件
3. 在 Eclipse 中创建 Inverted Index 项目
4. 在 Eclipse 中编写 Inverted Index 项目
5. 运行程序。
6. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**
7. 在本地编写好 input 文件，里面是实验指导书中 T1—T9 文件，然后复制到 hdfs 上



1. 编写 TokenInputFormat.java

package org.apache.hadoop.examples;

import java.io.IOException;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

import org.apache.hadoop.mapreduce.InputSplit;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.mapreduce.JobContext;

import org.apache.hadoop.mapreduce.RecordReader;

import org.apache.hadoop.mapreduce.TaskAttemptContext;

import org.apache.hadoop.mapreduce.TaskAttemptID;

import org.apache.hadoop.util.LineReader;

import java.util.StringTokenizer;

public class TokenInputFormat extends FileInputFormat<Text, ValuePair> {

    /\*\*

     \* Don't allow the files to be split!

     \*/

    @Override

    protected boolean isSplitable(JobContext ctx, Path filename) {

    // ensure the input files are not splittable!

    return false;

    }

    /\*\*

     \* Just return the record reader

     \* key is the docno

     \*/

    public RecordReader<Text, ValuePair> createRecordReader(InputSplit split,

        TaskAttemptContext ctx)

        throws IOException, InterruptedException {

    return new TokenRecordReader();

    }

    public static class TokenRecordReader extends RecordReader<Text, ValuePair> {

    private long start;

    private long pos;

    private long end;

    private LineReader in;

    private int maxLineLength;

    private Text line;

    private Text key = null;

    private ValuePair value = null;

    private StringTokenizer tokens = null;

    private int tokenPos = 0;

    private String fileID = "0";    // input file id that appears in inverted index

    public void initialize(InputSplit genericSplit,

        TaskAttemptContext context) throws IOException {

        FileSplit split = (FileSplit) genericSplit;

        Configuration job = context.getConfiguration();

        this.maxLineLength = job.getInt("mapred.linerecordreader.maxlength",

            Integer.MAX\_VALUE);

        start = split.getStart();

        end = start + split.getLength();

        final Path file = split.getPath();

        // Assume file name is an integer of file ID

        fileID = file.getName();

        FileSystem fs = file.getFileSystem(job);

        FSDataInputStream fileIn = fs.open(split.getPath());

        in = new LineReader(fileIn, job);

        this.pos = start;

        line = new Text();

        key = new Text();

        value = new ValuePair();

    }

    public boolean nextKeyValue() throws IOException {

        boolean splitEnds = false;

        while (tokens == null || !tokens.hasMoreTokens()) {

        int lineSize = in.readLine(line, maxLineLength,

            Math.max((int) Math.min(Integer.MAX\_VALUE, end - pos),

            maxLineLength));

        if (lineSize == 0) {

            splitEnds = true;

            break;

        }

        pos += lineSize;

        tokens = new StringTokenizer(line.toString(), " ,");

        }

        if (splitEnds) {

        key = null;

        value = null;

        line = null;

        tokens = null;

        return false;

        } else

        return true;

    }

    @Override

    public Text getCurrentKey() {

        key.set(tokens.nextToken());

        tokenPos ++;

        return key;

    }

    @Override

    public ValuePair getCurrentValue() {

        value.set(fileID, tokenPos);

        return value;

    }

    /\*\*

     \* Get the progress within the split

     \*/

    public float getProgress() {

        if (start == end) {

        return 0.0f;

        } else {

        return Math.min(1.0f, (pos - start) / (float) (end - start));

        }

    }

    public synchronized void close() throws IOException {

        if (in != null) {

        in.close();

        }

    }

    }

    public static void main(String[] args)

        throws IOException {

    String fn = args[0];

    Configuration conf = new Configuration();

    FileSplit split = new FileSplit(new Path(fn), 0, 10000000, null);

    TokenRecordReader irr = new TokenRecordReader();

    TaskAttemptContext ctx = new TaskAttemptContext(conf,

        new TaskAttemptID("hello", 12, true, 12, 12));

    irr.initialize(split, ctx);

    while (irr.nextKeyValue()) {

        System.out.println(irr.getCurrentKey() + ": " + irr.getCurrentValue());

    }

    }

}

1. 编写ValuePair.java

**package** org.apache.hadoop.examples;

**import** java.io.\*;

**import** org.apache.hadoop.io.\*;

**public** **class** ValuePair **implements** WritableComparable<ValuePair> {

**private** Text one;

**private** IntWritable two;

**public** **void** set(Text first, IntWritable second) {

    one = first;

    two = second;

    }

**public** **void** set(String first, **int** second) {

    one.set(first);

    two.set(second);

    }

**public** ValuePair() {

    set(**new** Text(), **new** IntWritable());

    }

**public** ValuePair(Text first, IntWritable second) {

    set(first, second);

    }

**public** ValuePair(String first, **int** second) {

    set(first, second);

    }

**public** Text getFirst() {

**return** one;

    }

**public** IntWritable getSecond() {

**return** two;

    }

    @Override

**public** **void** write(DataOutput out) **throws** IOException {

    one.write(out);

    two.write(out);

    }

    @Override

**public** **void** readFields(DataInput in) **throws** IOException {

    one.readFields(in);

    two.readFields(in);

    }

    @Override

**public** **int** hashCode() {

**return** one.hashCode();

    }

    @Override

**public** **boolean** equals(Object o) {

**if** (o **instanceof** ValuePair) {

        ValuePair tp = (ValuePair)o;

**return** one.equals(tp.one);

    }

**return** **false**;

    }

    @Override

**public** String toString() {

**return** "(" + one + ", " + two + ")";

    }

    @Override

**public** **int** compareTo(ValuePair tp) {

**int** cmp = one.compareTo(tp.one);

**if** (cmp != 0) {

**return** cmp;

    }

**return** two.compareTo(tp.two);

    }

**public** **static** **class** Comparator **extends** WritableComparator {

**private** **static** **final** Text.Comparator TEXT\_COMPARATOR = **new** Text.Comparator();

**private** **static** **final** IntWritable.Comparator INT\_COMPARATOR = **new** IntWritable.Comparator();

**public** Comparator() {

**super**(ValuePair.**class**);

    }

    @Override

**public** **int** compare(**byte**[] b1, **int** s1, **int** l1,

**byte**[] b2, **int** s2, **int** l2) {

**try** {

**int** oneL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1);

**int** oneL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);

**int** cmp = TEXT\_COMPARATOR.compare(b1, s1, oneL1, b2, s2, oneL2);

**if** (cmp != 0) {

**return** cmp;

        }

**return** INT\_COMPARATOR.compare(b1, s1+oneL1, l1-oneL1,

                        b2, s2+oneL2, l2-oneL2);

        } **catch** (IOException e) {

**throw** **new** IllegalArgumentException(e);

        }

    }

    @Override

**public** **int** compare(WritableComparable a, WritableComparable b) {

**if** (a **instanceof** ValuePair && b **instanceof** ValuePair) {

**return** ((ValuePair) a).compareTo((ValuePair) b);

        }

**return** **super**.compare(a, b);

    }

    }

**static** {

    WritableComparator.define(ValuePair.**class**, **new** Comparator());

    }

}

1. 编写InvertedIndex.java

**package** org.apache.hadoop.examples;

**import** java.io.IOException;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

**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;

/\*\*

 \*

 \* @author Ming

 \*/

**public** **class** InvertedIndex {

**public** **static** **class** TokenizerMapper

**extends** Mapper<Text, ValuePair, Text, ValuePair> {

    @Override

**public** **void** map(Text key, ValuePair value, Context context) **throws** IOException, InterruptedException {

    // TokenInputFormat has generate (word, (fileID, wordPosition))

    // so mapper just spill it to reducer

        key.set(key.toString().toLowerCase());

        context.write(key, value);

    }

    }

**public** **static** **class** IndexReducer

**extends** Reducer<Text, ValuePair, Text, Text> {

**private** Text postings = **new** Text();

    @Override

**public** **void** reduce(Text key, Iterable<ValuePair> values,

        Context context) **throws** IOException, InterruptedException {

        String list = "";

**for** (ValuePair val : values) {

        list += " " + val.toString();

        }

        postings.set(list);

        context.write(key, postings);

    }

    }

**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: InvertedIndex <in-dir> <out-dir>");

        System.exit(2);

    }

    // remove the old output dir

    FileSystem.get(conf).delete(**new** Path(otherArgs[1]), **true**);

    Job job = **new** Job(conf, "Inverted Indexer");

    job.setJarByClass(InvertedIndex.**class**);

    job.setInputFormatClass(TokenInputFormat.**class**);

    job.setMapperClass(InvertedIndex.TokenizerMapper.**class**);

    //job.setCombinerClass(InvertedIndex.IndexReducer.class);

    job.setReducerClass(InvertedIndex.IndexReducer.**class**);

    job.setMapOutputKeyClass(Text.**class**);

    job.setMapOutputValueClass(ValuePair.**class**);

    job.setOutputKeyClass(Text.**class**);

    job.setOutputValueClass(Text.**class**);

//    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));

//    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

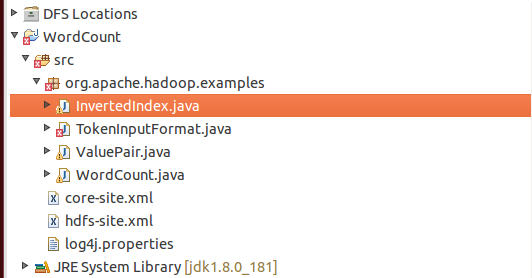
    FileInputFormat.addInputPath(job, **new** Path(args[0]));

    FileOutputFormat.setOutputPath(job, **new** Path(args[1]));

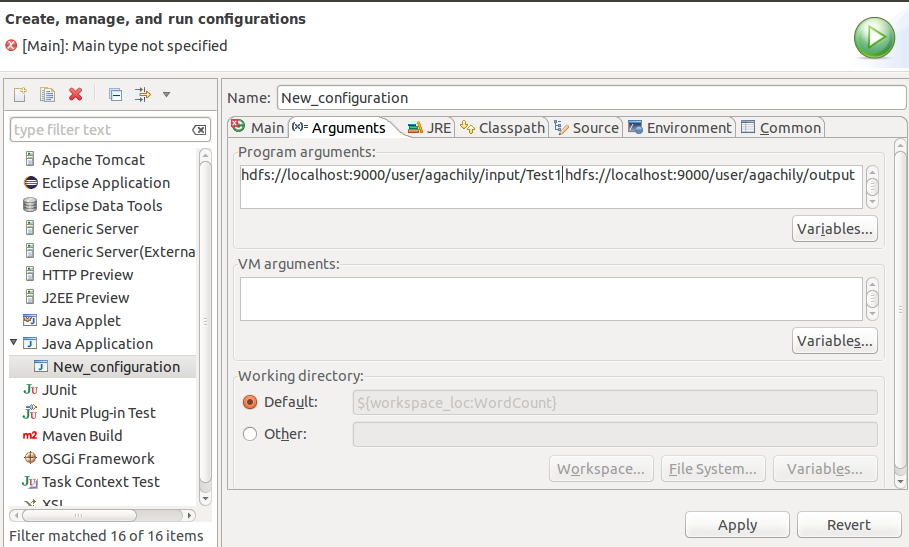
    System.exit(job.waitForCompletion(**true**) ? 0 : 1);

    }

}

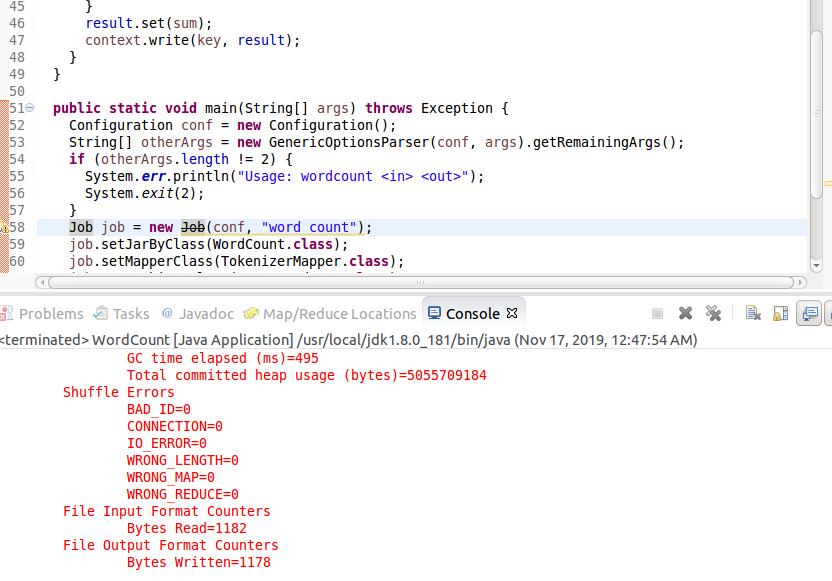


1. 对运行进行配置

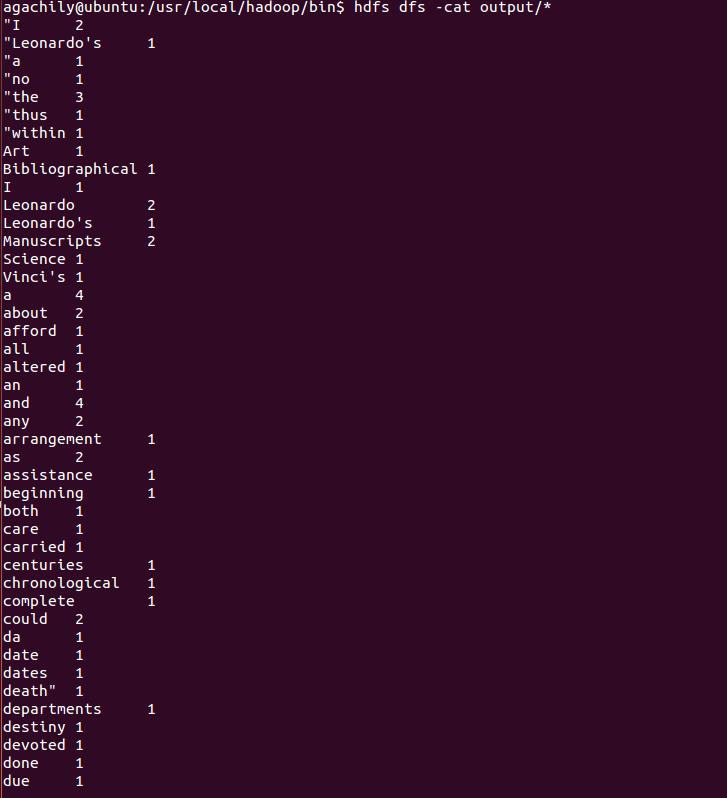


运行结果：

控制台



终端



1. **总结及心得体会：**

通过这次实验，我在eclipse上编写了实现反向索引的java程序。这次实验总的来说十分的简单，但其中有一个问题就是创建到eclipse中的每个程序，都要在运行配置中指明文件输入路径和输出路径。另一个关键的点就是在运行之前要注意把core-site和hadfs-site两个hadoop文件拉到工程目录下。才可以成功运行。

在之前的实验中我对如何运行mapreduce的程序还比较生疏但在这一次的实验中，我对如何进行这次额操作更加的熟练了，包括创建java文件，配置运行环境，向hdfs中上传文件等操作，另一个小的点就是要注意不同输出时要更改路径，不能用一样的。

回顾这四个实验，我是在两天之内做完的，这两天时间虽短，却学到和经历了许多，我了解到了hadoop具体是做什么的和其中的一些组件的作用到最后能够较为熟练地使用mapreduce进行创建，运行，文件的操作。在其中我也有环境配不出来，虚拟机崩溃，命令无法执行等挫折和痛苦，也有实验成功的喜悦，在做实验的过程中，成长的不仅仅是我对云计算的了解和认知，更有我的心性。

**十、对本实验过程及方法、手段的改进建议：**

无。

**报告评分：**

**指导教师签字：**