

# 天津大学

## JAVA 进阶编程第二次实验报告



学 院 智能与计算学部

专 业 软件工程

年 级 2017

姓 名 王新阳

2019 年 3 月 21 日

# JAVA 进阶编程第二次实验报告

## 一、需求分析（描述具体需求）

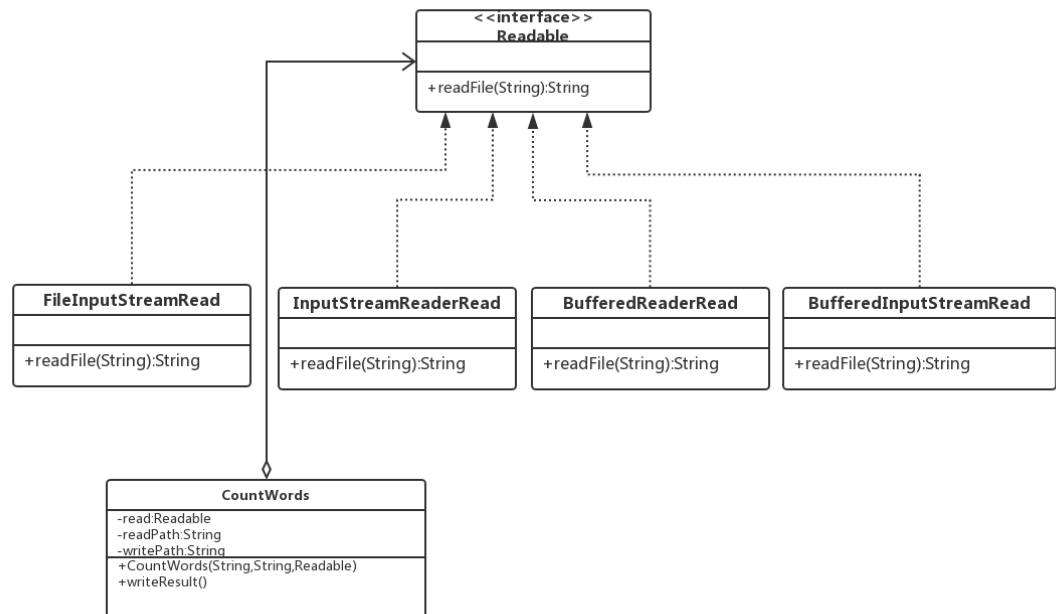
1. 编写程序，统计了不起的盖茨比中各单词出现的频次
2. 尝试使用不同的 stream 进行读文件操作
3. 异常处理

## 二、概要设计（简单描述设计思路，配合 UML 图）

采取策略模式实现使用不同的方式读取文件。

设计一个 Readable 接口，里面有 readFile 方法，FileInputStreamRead, InputStreamReaderRead, BufferedReaderRead, BufferedInputStreamRead 这四个 Java 文件分别实现了 Readable 接口。而这四个文件分别代表了四种读取文件的策略，分别是 FileInputStream、InputStreamReader、BufferedReader、BufferedInputStream。

CountWords 文件中的 writeResult 方法实现了将读取的字符串进行去除标点，统计单词个数并排序以及将结果写入文件。



## 三、详细设计（详细描述具体如何实现，附代码及说明）

**BufferedInputStreamRead** 类使用 **BufferedInputStream** 来读取文件。**BufferedInputStream** 对 **FileInputStream** 进行装饰，使普通的文件输入流具备了内存缓存的功能，通过内存缓冲减少磁盘 io 次数，所以 **BufferedInputStream** 的实例化需要 **FileInputStream**。

由于所给文件编码为 Unicode，故将读取的 byte 数组以 Unicode 编码转换成 String 类型。

```
try {
    File file = new File(path);
    bin = new BufferedInputStream(new FileInputStream(file));
    Long len = file.length();
    byte[] data = new byte[len.intValue()];
    int tmp = 0;
    int end = 0;
    while ((tmp = bin.read()) != -1) {
        data[end++] = (byte) tmp;
    }
    src = new String(data, 0, end, "Unicode");
}
```

FileInputStreamRead 类使用 FileInputStream 读取文件，其具体代码实现类似于 BufferedInputStreamRead 类故不作详细说明。

```
try {
    File file = new File(path);
    in = new FileInputStream(file);
    Long len = file.length();
    byte[] data = new byte[len.intValue()];
    int tmp = 0;
    int end = 0;
    while ((tmp = in.read()) != -1) {
        data[end++] = (byte) tmp;
    }
    src = new String(data, 0, end, "Unicode");
}
```

InputStreamReaderRead 类使用 InputStreamReader 读取文件，由于所给文件编码格式 Unicode 与系统默认编码方式不同，所以需要传入编码参数 Unicode，而显然 InputStreamReader 又是对 InputStream 的装饰。

```
File file = new File(path);
String src = "";
InputStreamReader ir = null;
try {
    ir = new InputStreamReader(new FileInputStream(file), "Unicode");
    Long len = file.length();
    byte[] data = new byte[len.intValue()];
    int tmp = 0;
    int end = 0;
    while ((tmp = ir.read()) != -1) {
        //对读取数据进行初步过滤
        if (Character.isLetter((char) tmp) || (char) tmp == ' ' || (char) tmp == '\n'
            || Character.isDigit((char) tmp)) {
            data[end++] = (byte) tmp;
        }
    }
    src = new String(data, 0, end);
}
```

我在读取字符时做了简单的过滤，过滤掉了一些不可见字符以及标点符号，而

这也导致了输出结果与其他三种方式有些许的不同。

BufferedReaderRead 类使用 BufferedReader 读取文件，BufferedReader 提供更高效的 readLine 方法所以代码实现与上述三种类更为简单。未避免不同行之间单词间隔消失，我在不同行拼接时加入了空格。

```
File file = new File(path);
String src = "";
BufferedReader br = null;
try {
    br = new BufferedReader(new InputStreamReader(new FileInputStream(file), "Unicode"));
    String line;
    while ((line = br.readLine()) != null) {
        src += line + " ";
    }
}
```

CountWords 内实现了数据的过滤、统计单词出现次数排序以及写入文件。

```
String src = this.read.readFile(readPath);
//过滤数据
src = src.trim();
src = src.replaceAll("\\p{P}", "");
String pattern = "\\s";
```

使用正则表达式 `p{P}` 匹配所有标点符号并将其替换成空，在根据空白符将文章分成单词数组。

```
String[] split = src.split(pattern);
```

使用 map 存储数据，以便利用其 key 的不可重复性统计数据。其中在统计时会出现""的情况，故将其过滤。

```
Map<String, Integer> map = new TreeMap<>();
for (int i = 0; i < split.length; i++) {
    String key = split[i];
    key = key.toLowerCase();
    if (key.equals("")) { // 去除诡异的""的情况
        continue;
    }
    int count = map.containsKey(key) ? map.get(key) + 1 : 1;
    map.put(key, count);
}
```

排序，重写 Comparator 的 compare 方法，利用 Collections 的 sort 函数进行数据的排序，由于 Collections 的 sort 函数只能排序 list 所以将 map 拆分成元素为 map.entry 的 list。利用 Sort 函数对元素为 map.entry 的 list 进行排序。

```

List<Map.Entry<String, Integer>> list = new ArrayList<Map.Entry<String, Integer>>();
list.addAll(map.entrySet());
Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
    @Override
    public int compare(Entry<String, Integer> o1, Entry<String, Integer> o2) {
        // TODO Auto-generated method stub
        return o2.getValue() - o1.getValue();
    }
});

OutputStream out = null;
try {
    File output = new File(writePath);
    out = new FileOutputStream(output);
    // 创建文件
    if (!output.exists()) {
        output.createNewFile();
    }

    // 写入文件
    Iterator<Entry<String, Integer>> it = list.iterator();
    while (it.hasNext()) {
        Entry<String, Integer> temp = it.next();
        String str = temp.getKey() + " " + temp.getValue() + "\r\n";
        byte outData[] = str.getBytes();

        for (int x = 0; x < outData.length; x++) {
            out.write(outData[x]);
        }
    }
}

```

使用 FileOutputStream 写入文件。

#### 四、调试分析（在实验过程中遇到的问题以及如何解决）

由于所给数据编码格式与系统默认的不同，在本次实验中就遇到了大量的编码问题。其中 FileInputStreamRead 与 InputStreamReaderRead 类通过在 byte 数组生成 String 时设定其编码格式解决。而 InputStreamReader 本身就可以通过传编码参数解决编码问题。而在 BufferedReader 转换编码格式中遇到了困难。

```

/**
 * Creates a buffering character-input stream that uses a default-sized
 * input buffer.
 *
 * @param in A Reader
 */
public BufferedReader(Reader in) {
    this(in, defaultCharBufferSize);
}

```

阅读源码可知，BufferedReader 的实例化需要一个 Reader 的子类，所以简单

的想到了使用 `FileReader`,但是 `FileReader` 读取文件是使用的系统默认的编码方式,而阅读源码可知其并不能改变编码格式。

后上网查阅资料得知, `InputStreamReader` 是字节转换为字符的桥梁。它可以在构造器重新指定编码的方式,如果不指定的话将采用底层操作系统的默认编码方式。

```
public InputStreamReader(InputStream in, String charsetName)
    throws UnsupportedOperationException
{
    super(in);
    if (charsetName == null)
        throw new NullPointerException("charsetName");
    sd = StreamDecoder.forInputStreamReader(in, this, charsetName);
}
```

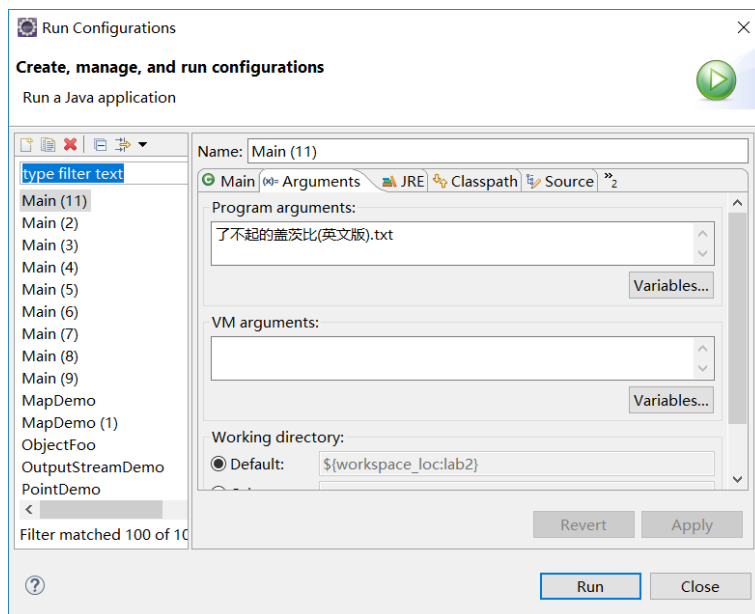
```
br = new BufferedReader(new InputStreamReader(new FileInputStream(file),"Unicode"));
```

最终通过上述方法实例化了使用 `Unicode` 编码读取文件的 `BufferedReader` 类。

将文件放到 C 盘会遇到文件读写权限限制的问题,放到其他盘即可解决该问题。

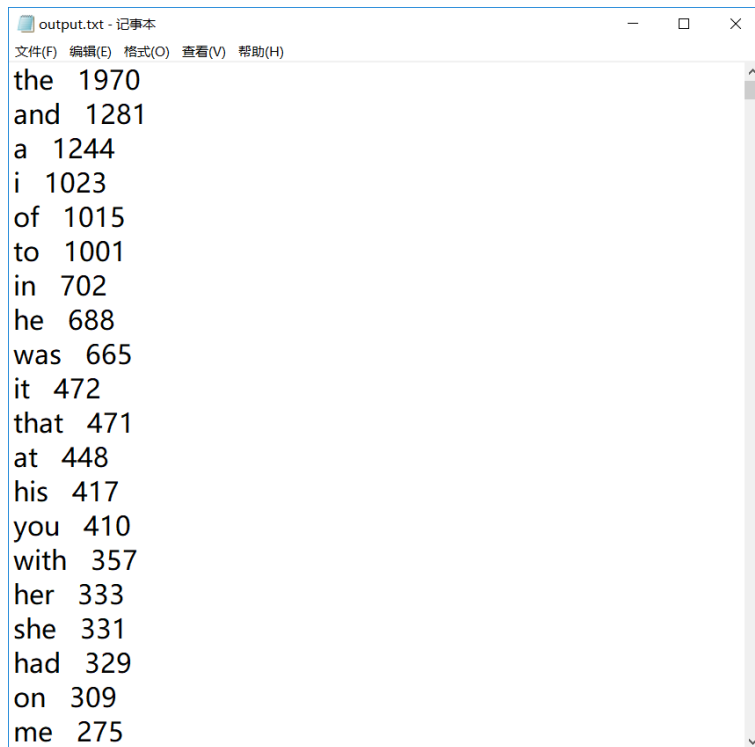
## 五、测试结果（描述输入和输出）

输入：



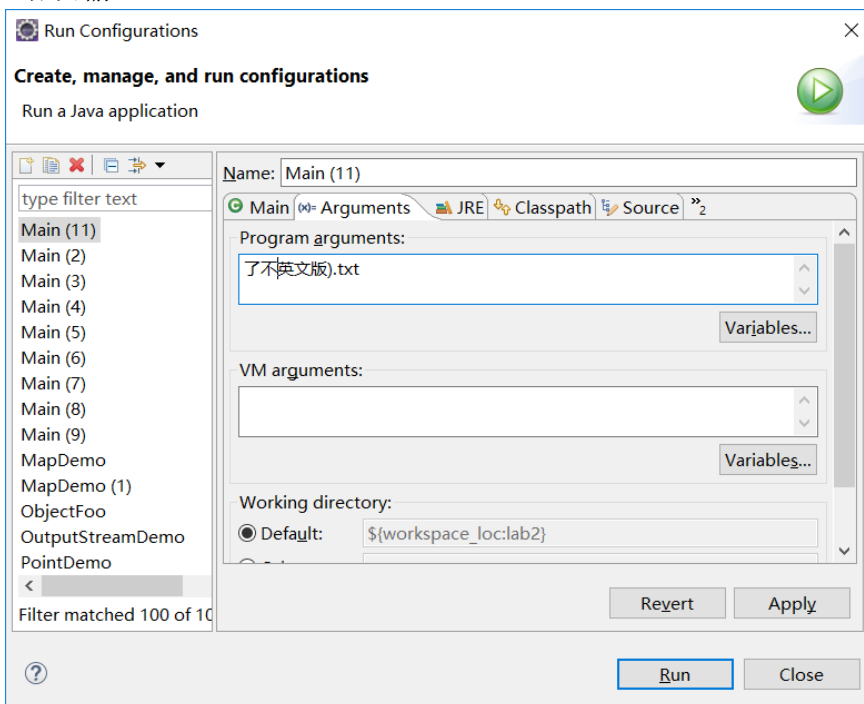
输出：

在项目目录的 `output` 文件夹中生成 `output.txt` 里面存放有运行结果。



the 1970  
and 1281  
a 1244  
i 1023  
of 1015  
to 1001  
in 702  
he 688  
was 665  
it 472  
that 471  
at 448  
his 417  
you 410  
with 357  
her 333  
she 331  
had 329  
on 309  
me 275

错误输入:



输出:

\\lab2\data\了不英文版).txt (系统找不到指定的文件。)

## 六、总结

通过本次实验，我对于 Java 的 IO 有了进一步的认识，对于计算机的编码有了更深一步的了解，了解到了 Java 在设计 IO 的一些类时所采用的装饰模式等设计模式，让我对于 Java 设计模式有了更深刻的理解。通过阅读 Java 源代码来理解 Java 一些类的使用以及修改 Bug，提高了我对程序的理解。