1. 测试数据集

1.1 功能测试数据集

功能测试数据集里只包括 3 个文档，这 3 个文档里包含：单词、数字、单词和数字组合，非常短的单词（长度小于 3），非常长的单词（长度大于 20）。文档中的单词不多，主要用来对程序的索引构建基本功能进行测试，包括：

1：能否正确地解析文档，构建出倒排索引并打印出来；

2：检查倒排索引的内容是否正确，由于文档数量和单词数量很少，所以同学们基本上可以目测测试结果是否正确（即判断每个单词在每个文档里是否出现、出现的次数、出现的位置）；

3：测试当使用过滤器后，是否可以将停用词、过短的单词（如长度小于 3）、过长的单词（如长度大于 20）、包含数字的单词（可以通过正则表达式过滤）过滤掉，从而在倒排索引里不包含这些单词和对应的 PostingList；

请大家一定不要修改这三个测试文档，文档里有些空行是特意留出来，测试同学们的程序是否可以正确处理这些情况。

1.2 真实测试数据集

真实测试数据集目前包含了 15 个文档，是从新闻里摘出来的真实的英文短文。大家在使用 1.1 节的功能测试数据集测试自己代码的索引构建功能基本完成后，就可以用这 15 个文档来构建一个更大的倒排索引，同时测试检索功能。**也请大家不要修改这些文档**。后面会提供更大点的数据集进行测试。

1.3 测试用检索词

真实测试数据集还包括一个文档“用于检索的测试词.txt”，里面包含了可以**对基于 1.2 节的真实测试数据集构建的倒排索引进行检索**的检索词，同学们可以利用里面的检索词对检索功能进行测试。

2. 测试数据集和构建好的索引文件的位置

建议大家将测试数据集放在工程目录的 text 子目录下，在程序里指定测试集的位置时可以用相对路径。同时构建好的索引文件也放置在工程目录的 index 子目录下，在程序里指定索引文件的保存位置时也用相对路径。这样无论工程目录移动到哪里，不影响程序的正常执行。**特别是将工程打包提交后，可以大大方便老师和助教对同学们的代码进行测试**。我的工程目录结构如图 1 所示：

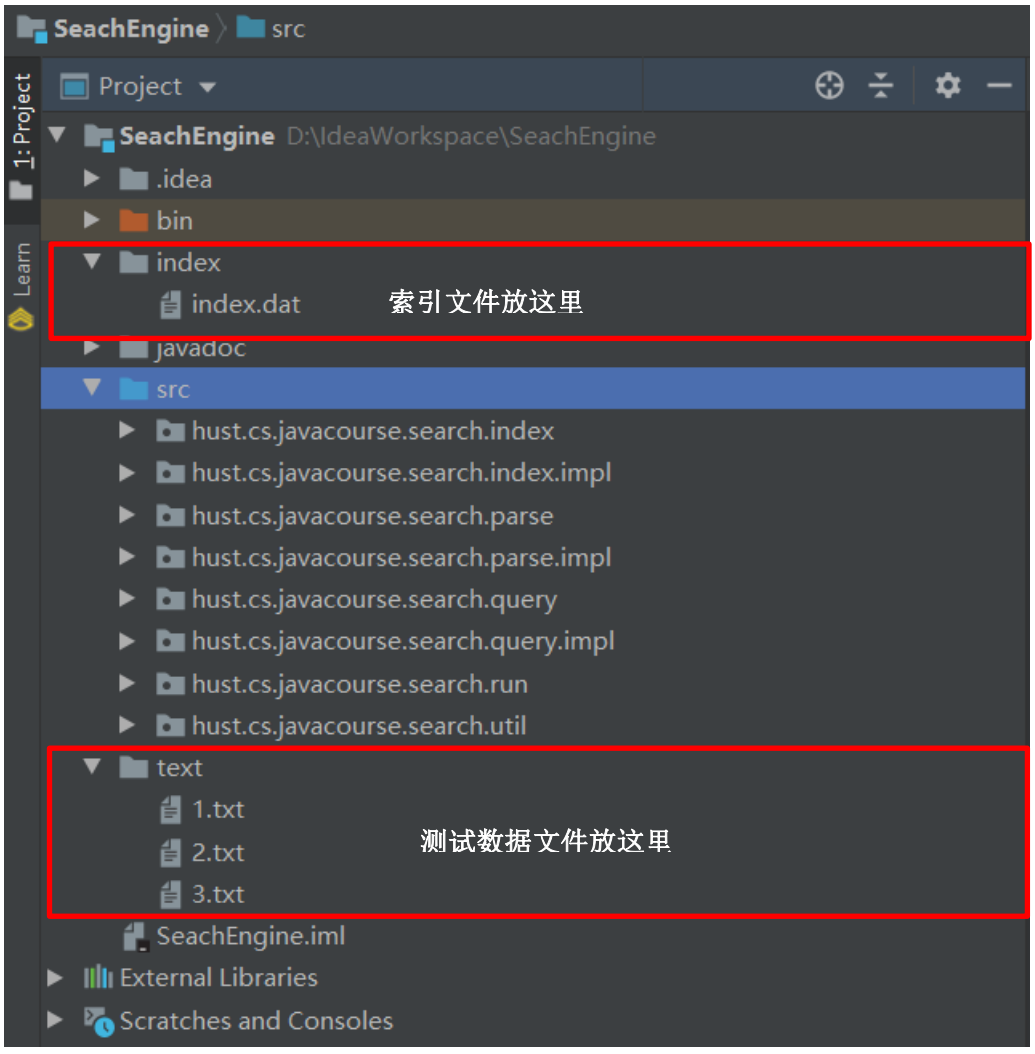

图 1 包含测试数据文件和索引文件的工程结构示意图

采用这样的目录结构还有个好处可以利用 Config 文件里定义好的静态变量直接得到这些目录的绝对路径，见下面第 3 节介绍。

3. Config 文件说明

为了方便同学们完成实验，在 hust.cs.javacourse.search.util 包里的 Config 类里，定义了下面的静态变量自动获取同学们的工程根目录，同时在工程根目录定义了另外二个静态变量获取工程根目录下的索引文件目录和测试数据文件目录：

```
/**
 * Java 工程HOME 目录, System.getProperty("user.dir")返回当前JAVA 工程目录
 */
public static String PROJECT_HOME_DIR= System.getProperty("user.dir");
/**
 * <pre>
 * 索引文件的目录,以相对路径指定索引文件目录,将索引文件保存在当前工程目录下的 index 子目录中.
 * 这样做的好处：索引文件目录是相对路径,无论你把整个工程放在什么位置，程序都可以正常运行.
 * </pre>
 */
public static String INDEX_DIR = PROJECT_HOME_DIR + "/index/";
/**
 * <pre>
 *  文本文件的目录,以相对路径指定文本文件目录 ,将文本文件保存在当前工程目录下的text 子目录中
 *  这样做的好处：文本文件目录是相对路径,无论你把整个工程放在什么位置，程序都可以正常运行.
 * </pre>
 *
 */
public static String DOC_DIR  = PROJECT_HOME_DIR + "/text/";
```

因此，同学们在代码任何位置，通过 Config. PROJECT_HOME_DIR、Config. INDEX_DIR、Config. DOC_DIR 这三个静态变量可以得到工程根目录、索引文件目录、测试文件目录。

另外在 Config 文件里面还定义了如下的静态常量方便大家实现基于正则表达式的过滤器和基于单词长度的过滤器：

```
/**
 * <pre>
 * 单词过滤的正则表达式.
 * 例如正则表达式指定只保留由字母组成的term,其他的term 全部过滤掉,不写入倒排索引
 * </pre>
 */
public static String TERM_FILTER_PATTERN = "[a-zA-z]+";

/**
 * <pre>
 * 基于单词的最小长度过滤单词.
 * 例如指定最短单词长度为3，长度小于3 的单词过滤掉，不写入倒排索引
 * </pre>
```

```
 */
public static int TERM_FILTER_MINLENGTH = 3;

/**
 * <pre>
 * 基于单词的最小长度过滤单词.
 * 例如指定最长单词长度为20，长度大于20的单词过滤掉，不写入倒排索引
 * </pre>
 */
public static int TERM_FILTER_MAXLENGTH = 20;
```

另外还有个静态变量是用来将字符串分成一个个的 Term，如下所示：

```
/**
 * <pre>
 * 将字符串切分成单词时所需的正则表达式.
 * 例如根据中英文的逗号,分号,句号，问号，冒号,感叹号，中文顿号，空白分割符进行切分
 * </pre>
 *
 */

public static String STRING_SPLITTER_REGEX = "[,|，|;|；|.|。|?|？|:|：|!|！|、|\\s|\"]+";
```

同时 hust.cs.javacourse.search.util.StringSplitter 已经基于 Config 文件里定义的这个静态变量 Config. STRING_SPLITTER_REGEX 实现了将字符串拆分成一个个 Term 的方法。在 StringSplitter 类里的 main 方法里给出了 Demo 代码。大家可以直接使用这个类。如果觉得对 StringSplitter 的字符串分词效果不满意，修改这个正则表达式即可。

4. 用测试数据集进行测试

4.1 功能性测试数据集的测试效果示意

如果不采用任何过滤器，则用功能性测试数据集构建索引的结果示意如图 2 所示：

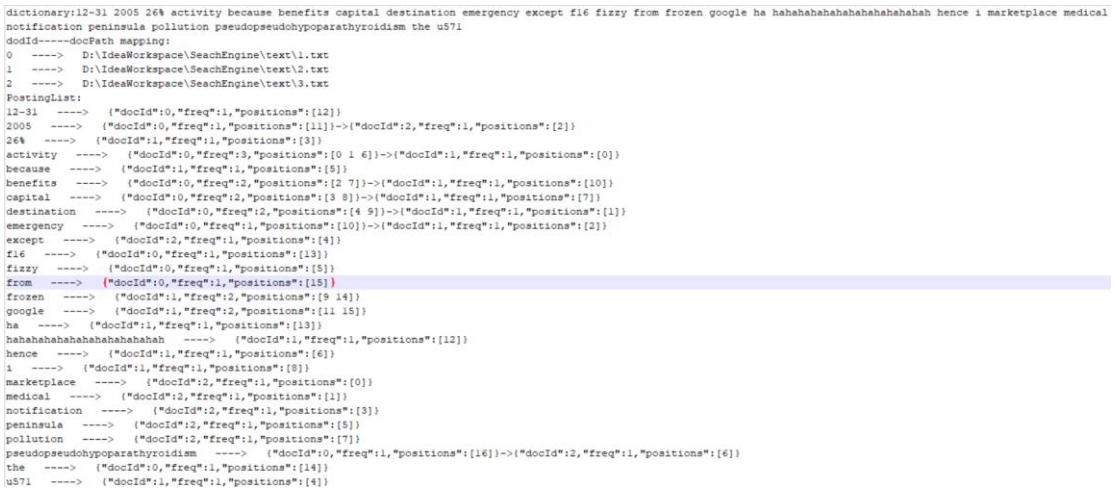

图 2 用功能性测试数据集构建索引结果(不带任何过滤器)

如果加上停用词过滤器、正则表达式过滤器、单词长度过滤器，则将停用词、包含非字母字符的单词、长度小于 3 或大于 20 的单词全部过滤掉后，构建索引的结果示意如图 3 所示：

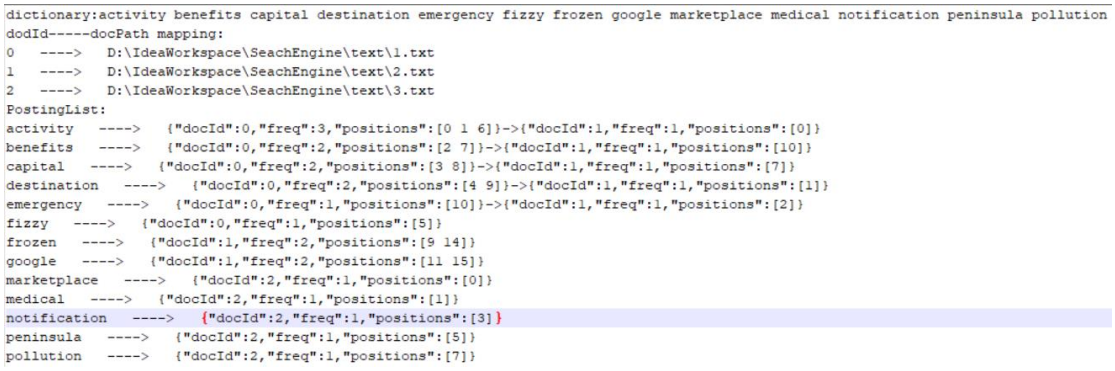

图 3 用功能性测试数据集构建索引结果(带三种过滤器)

同学们可以放大看到这二个图的内容。同学们的代码如果不改变 Config 里的配置，用功能性测试数据集构建好的索引应该和我是一样的（输出的格式可能不同）。

4.2 用真实测试数据集的测试效果示意

采用真实测试数据集，加上三个过滤器，构建索引的结果示意如图 4 所示（单词太多，部分结果）：

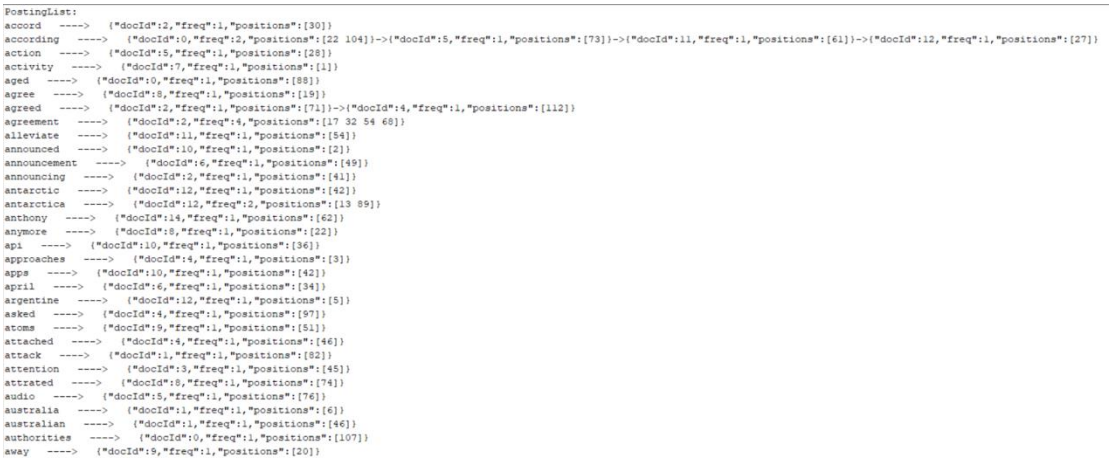

图 4 用真实测试数据集构建索引结果(带三种过滤器)

基于图 4 所示的索引构建结果，采用 coronavirus 作为检索词的检索结果如图 5 所示：

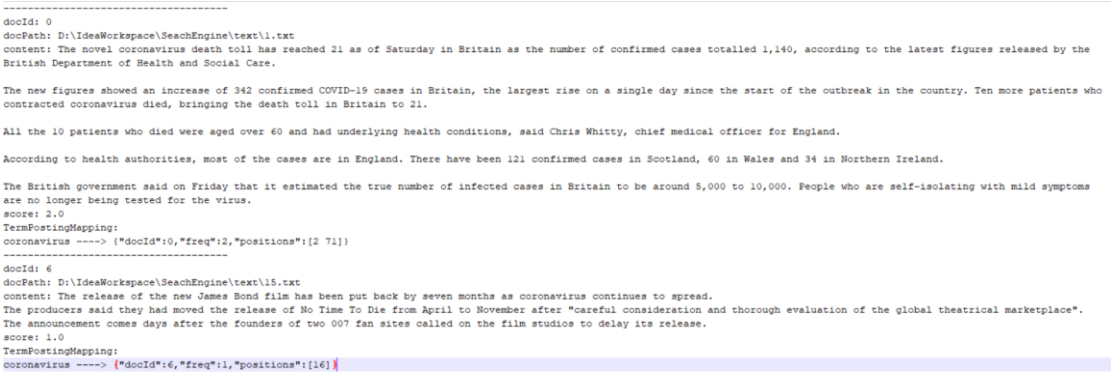

图 5 采用 coronavirus 作为检索词的检索结果

同学们放大图片结果可以看到，采用 coronavirus 作为检索词命中了二个结果：第一个结果的得分为 2，因此排在前面；第二个结果的得分为 1，因此排在第二位。同时检索结果也显示了命中的文档的原文内容，由于不是图形化界面，否则可以根据命中结果中单词的位置信息，将原文里的 coronavirus 单词进行 Highlight 显示。

4.3 测试代码示意

如果大家完成了所有抽象类和接口的具体实现后，构建索引的测试代码示例应该是这样的：

```
public class TestBuildIndex {
    public static void main(String[] args){
        AbstractDocumentBuilder documentBuilder = new DocumentBuilder();
        AbstractIndexBuilder indexBuilder = new IndexBuilder(documentBuilder);
        String rootDir = Config.DOC_DIR;
        System.out.println("Start build index ...");
        AbstractIndex index = indexBuilder.buildIndex(rootDir);
        index.optimize();
        System.out.println(index);  //控制台打印 index 的内容


        //测试保存到文件
        String indexFile = Config.INDEX_DIR + "index.dat";
        index.save(new File(indexFile));    //索引保存到文件


        //测试从文件读取
        AbstractIndex index2 = new Index();  //创建一个空的 index
        index2.load(new File(indexFile));       //从文件加载对象的内容
        System.out.println("\n-------------------\n");
        System.out.println(index2);  //控制台打印 index2 的内容
    }
}
```

在构建好索引后，检索功能的测试代码示例应该是这样的：

```
public class TestSearchIndex {
    public static void main(String[] args){
        Sort simpleSorter = new SimpleSorter();
        String indexFile = Config.INDEX_DIR + "index.dat";
        AbstractIndexSearcher searcher = new IndexSearcher();
        searcher.open(indexFile);

        AbstractHit[] hits = searcher.search(new Term("coronavirus"), simpleSorter);
        for(AbstractHit hit : hits){
            System.out.println(hit);
        }

    }
}
```

在索引构建过程中加上过滤器的示例代码可以是在抽象类 AbstractDocument 的子类对抽象方法 public AbstractDocument build(int docId, String docPath, File file)给出的具体实现中添加：

```java
@Override
public AbstractDocument build(int docId, String docPath, File file) {
    AbstractDocument document = null;
    AbstractTermTupleStream ts = null;
    try {
        ts= new TermTupleScanner(new BufferedReader(new InputStreamReader(new
FileInputStream(file))));
        ts = new StopWordTermTupleFilter(ts);   //再加上停用词过滤器
        ts = new PatternTermTupleFilter(ts);    //再加上正则表达式过滤器
        ts = new LengthTermTupleFilter(ts);     //再加上单词长度过滤器
        document = build(docId,docPath,ts);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    finally {
        ts.close();
    }
    return document;
}
```

这里是在程序里添加不同功能的过滤器。但实际应用中不应该在程序里将代码写死（如果还需要新的过滤器，就得修改源代码还得再重新编译），最好的方式是应该写一个配置文件，在配置文件里配置索引构建所需的过滤器，这里关键是在配置文件里要指定所需过滤器的具体实现类的完全限定类名，然后 Java 程序启动后读取这个配置文件，利用反射机制构造好过滤器对象，注入到 AbstractDocument 的子类对象里。Spring 框架中的一个重要功能：对象注入（Object Injection）就是做的这类事情。**但是本实验不做这类要求。**