華中科技大學

课 程 实 验 报 告

课程名称：Java 语言程序设计

实验名称：基于内存的搜索引擎设计和实现

院　　系 ：计算机科学与技术

专业班级 ：　ACM1801

学　　号 ：　U201814757

姓　　名 ：　夏媛

指导教师 ：　纪俊文

2021 年 **5** 月 **10** 日

目 录

一、 需求分析 1
1. 题目要求 1
2. 需求分析 2
二、 系统设计 4
1. 概要设计 4
1.1 项目架构 4
1.2 程序模块设计 5
1.3 设计思路及原理 7
1.4 人机界面设计 8
2. 详细设计 8
2.1 模块 hust.cs.javacourse.search.index 8
2.2 模块 hust.cs.javacourse.search.parse 15
2.3 模块 hust.javacourse.search.query 18
三、 软件开发 21
四、 软件测试 21
4.1 自动测试 21
4.2 功能 1 测试 22
4.3 功能 2 测试 23
4.4 功能 3 测试 24
4.5 功能 4 测试 27
五、 特点与不足 28
六、 过程和体会 28
1. 遇到的主要问题和解决方法 28
2. 课程设计的体会 28
七、 源码和说明 29
1. 文件清单及其功能说明 29
2. 用户使用说明书 30

一、 需求分析

1. 题目要求

实现一个基于内存的英文全文检索搜索引擎，需要完成以下功能：

功能 1：将指定目录下的一批.txt 格式的文本文件扫描并在内存里建立倒排索引，这里面包含必须的子功能包括：

（1）读取文本文件的内容；

（2）将内容切分成一个个的单词；

（3）过滤掉其中一些不需要的单词,例如数字、停用词（the, is and 这样的单词）、过短或过长的单词（例如长度小于 3 或长度大于 20 的单词）；

（4）利用 Java 的集合类在内存里建立过滤后剩下单词的倒排索引；

（5）内存里建立好的索引对象可以序列化到文件，同时可以从文件里反序列化成内存里的索引对象；

（6）可以在控制台输出索引的内容。

功能 2：基于构建好的索引，实现单个搜索关键词的全文检索，包含的子功能包括：

（1）根据搜索关键词得到命中的结果集合；

（2）可以计算每个命中的文档的得分，并根据文档得分对结果集排序；

（3）在控制台显示命中的文档的详细信息，如文档的路径、文档内容、命中的关键词信息（如在文档里出现次数）、文档得分；

功能 3：基于构建好的索引，实现二个搜索关键词的全文检索。包含的子功能包括：

（1）支持这二个关键词的与或查询。与关系必须返回同时包含这二个单词的文档集合，或关系返回包含这二个单词中的任何一个的文档集合；

（2）可以计算每个命中的文档的得分，并根据文档得分对结果集排序；

（3）在控制台显示命中的文档的详细信息，如文档的路径、文档内容、命中的关键词信息（如在文档里出现次数）、文档得分；

功能 4：基于构建好的索引，实现包含二个单词的短语检索，即这二个单词必须在作为短语文档里出现，它们的位置必须是相邻的。**这个功能为进阶功能**。

除了以上功能上的要求外，其他要求包括：

（1）针对搜索引擎的倒排索引结构，已经定义好了创建索引和全文检索所需要的抽象类和接口。**学生必须继承这些预定义的抽象类和和实现预定义接口来完成实验的功能，不能修改抽象类和接口里规定好的数据成员、抽象方法；也不能在预定义抽象类和接口里添加自己新的数据成员和方法**。但是实现自己的子类和接口实现类则不作任何限定。

（2）自己实现的抽象类子类和接口实现类里的关键代码必须加上注释，其中每个类、每个类里的公有方法要加上 Javadoc 注释，并自动生成 Java API 文档作为实验报告附件提交。

（3）使用统一的测试文档集合、统一的搜索测试案例对代码进行功能测试，构建好的索

引和基于统一的搜索测试案例的检索结果最后输出到文本文件里作为实验报告附件提交。

（4）本实验只需要基于控制台实现，实验报告里需要提供运行时控制台输出截屏。

关于搜索引擎的倒排索引结构、相关的抽象类、接口定义、还有相关已经实现好的工具类会在单独的 PPT 文档里详细说明。同时也为学生提供了预定义抽象类和接口的 Java API 文档和 UML 模型图。

2. 需求分析

需求 1：读取指定目录下的一批.txt 格式的文本文件，建立倒排索引，并序列化到文件。

实现需求 1 主要基于 IndexBuilder 类中的 buildIndex 方法，该方法的效果为依次遍历和解析指定目录下的文本文件，得到对应的 Document 对象，再加入到索引。该方法的接收参数：一个类型为 String 的参数表示文本文件的根路径。返回类型为 AbstractIndex。

表 1-1 需求 1 的子需求表

子需求	对应方法	接受参数	返回类型	异常捕获
遍历和解析目录下的文件，依次将得到的 document 对象加入索引	IndexBuilder 类的 buildIndex 方法	指定目录： String rootDirectory	AbstractIndex	IOException
解析文本文档，过滤，构造 Document 对象	DocumentBuilder 类的 build 方法	文档 Id:docId; 文档路径：docPath; 文档对应的输入流： termTupleStream	AbstractDocument	
添加文档到索引	Index 类的 addDocument 方法	文档对象： AbstractDocument document	无	无
实现流读取文本文档的三元组	TermTupleScanner 类的的构造函数和 next()方法	构造函数的参数： 缓冲输入流 BufferedReader input	next()方法返回: AbstractTermTuple	
内存里建立好的索引对象序列化到文件	Index 类的 save 方法	写入的目标文件： File file	无	IOException
从文件里反序列化成内存里的索引对象	Index 类的 load 方法	索引文件： File file	无	IOException

需求 2：基于构建好的索引，实现单个搜索关键词的全文检索。

实现需求 2 主要基于 IndexSearcher 中的 search 方法，该方法的效果为基于现有的 Index 根据单个检索词进行搜索。该方法接收参数：第一个参数类型为 AbstractTerm 表示检索词，第二个参数类型为 Sort，表示排序器，返回命中信息列表，类型为 AbstractHit[]。

表 1-2 需求 2 的子需求表

子需求	对应方法	接收参数	返回类型	异常捕获
根据搜索关键词得到命中的结果集合	IndexSearcher 中的 search 方法	AbstractTerm queryTerm; Sort sorter	AbstractHit[]	无
计算每个命中的文档的得分，并根据得分对结果集排序	SimpleSorter 类中的 sort 方法	List<AbstractHit> hits	void	无
在控制台显示命中的详细信息	Hit 类中的 toString 方法	无	String	无

需求 3：基于构建好的索引，实现二个搜索关键词的与查询和或查询。

实现需求 3 主要基于 IndexSearcher 中的 search 方法，该方法的效果为基于现有的 Index 根据两个检索词进行搜索。该方法接收参数：第一个参数类型为 AbstractTerm 表示第一个检索词，第二个参数类型为 AbstractTerm 表示第一个检索词，第三个参数类型为 Sort，表示排序器，第四个参数类型为 LogicalCombination，取值为{AND,OR}，分别表与查询和或查询。返回类型为 AbstractHit[]。

需求 3：基于构建好的索引，实现二个搜索关键词的的短语检索。

实现需求 4 主要基于 IndexSearcher 中的 search 方法，该方法的效果为基于现有的 Index 根据两个检索词进行搜索。该方法接收参数：第一个参数类型为 AbstractTerm 表示第一个检索词，第二个参数类型为 AbstractTerm 表示第一个检索词，第三个参数类型为 Sort，表示排序器。返回类型为 AbstractHit[]。

二、 系统设计

1. 概要设计

1.1 项目架构

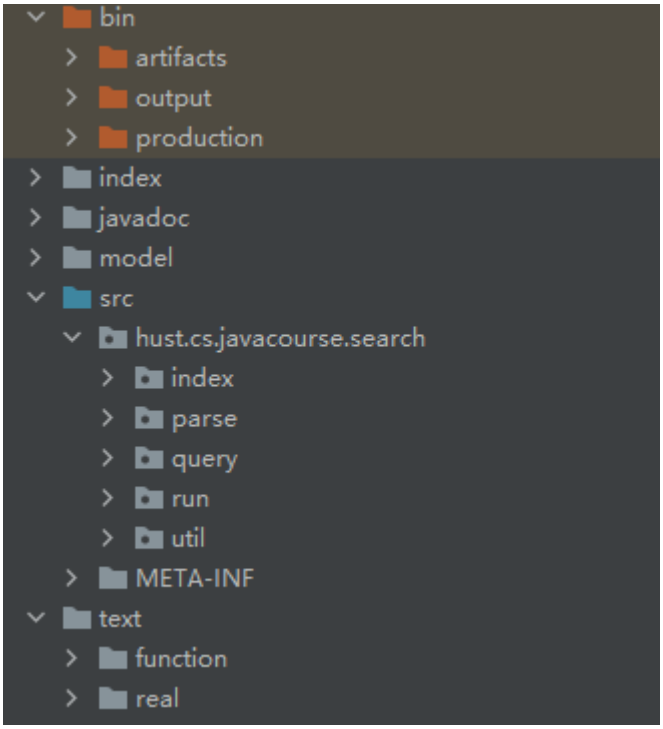

图 2.1 项目结构图

项目结构如图 2.1 所示，各个文件夹表示：

1. index: 存储索引对象的序列化文件；
2. javadoc: javadoc 说明；
3. model: 存储 UML 和 UML 图；
4. text: 保存功能测试数据集和真实测试数据集；
5. src: 程序包的顶级目录，包括：
   a) Index: 包含了倒排索引结构相关的抽象类，impl 文件夹中是抽象类的具体实现；
   b) Parse: 包含了三元组相关的抽象类， impl 文件夹中是抽象类的具体实现；
   c) Query: 包含了搜索相关的抽象类，impl 文件夹中是抽象类的具体实现；
   d) Run: 自定义的测试程序；
   e) Util: 各种工具类。
6. bin: 程序的输出目录，包括：
   a) output: 保存索引对象的文本文件表示和指定搜索词的搜索结果；
   b) production：存有 java 源程序编译后的.class 文件；

c） artifacts：程序的 jar 包。

1.2 程序模块设计

1） **package hust.cs.javacourse.search.index**

包中定义的抽象类和接口如图 2.2 所示：

接口概要

接口	说明
FileSerializable	定义文件序列化接口

类概要

类	说明
AbstractDocument	AbstractDocument是文档对象的抽象父类.
AbstractDocumentBuilder	AbstractDocumentBuilder是Document构造器的抽象父类.
AbstractIndex	AbstractIndex是内存中的倒排索引对象的抽象父类.
AbstractIndexBuilder	AbstractIndexBuilder是索引构造器的抽象父类 需要实例化一个具体子类对象完成索引构造的工作
AbstractPosting	AbstractPosting是Posting对象的抽象父类.
AbstractPostingList	AbstractPostingList是所有PostingList对象的抽象父类.
AbstractTerm	AbstractTerm是Term对象的抽象父类.
AbstractTermTuple	AbstractTermTuple是所有TermTuple对象的抽象父类.

图 2.2 index 包中定义的抽象类和接口图

接口、类之间的关系如下图 2.3 所示：

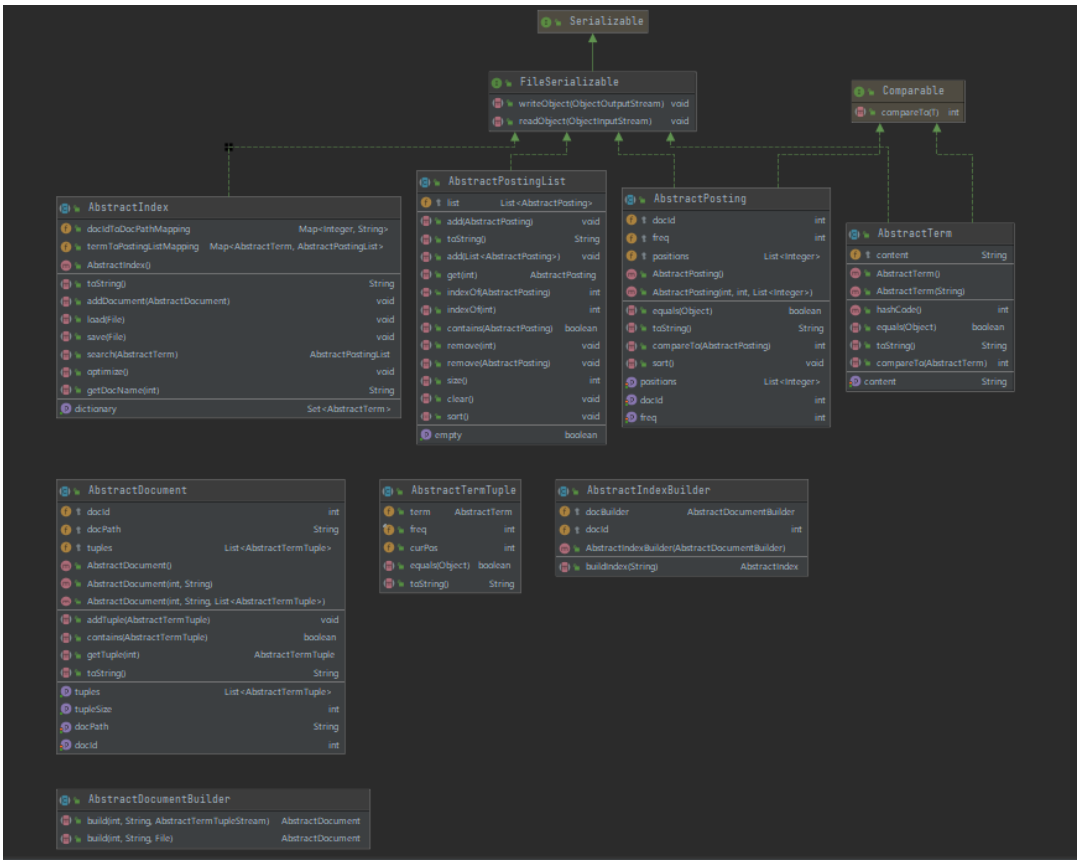

图 2.3 index 包中接口和类的 UML 关系图

2） **package hust.cs.javacourse.search.parser**

包中定义的抽象类如下图 2.4 所示：

类概要

类	说明
AbstractTermTupleFilter	抽象类AbstractTermTupleFilter类型是AbstractTermTupleStream的子类,里面包含另一个AbstractTermTupleStream对象作为输入，并对输入的AbstractTermTupleStream进行过滤, 例如过滤掉所有停用词（the，is are...)对应的三元组 其具体子类需要重新实现next方法以过滤掉不需要的单词对应的三元组.同时可以实现多个不同的过滤器 完成不同的过滤功能，多个过滤器可以形成过滤管道.
AbstractTermTupleScanner	AbstractTermTupleScanner是AbstractTermTupleStream的抽象子类，即一个具体的TermTupleScanner对象就是 一个AbstractTermTupleStream流对象，它利用java.io.BufferedReader去读取文本文件得到一个个三元组TermTuple.
AbstractTermTupleStream	AbstractTermTupleStream是各种TermFreqPosTupleStream对象的抽象父类 TermFreqPosTupleStream是三元组TermTuple流对象，包含了解析文本文件得到的三元组序列

图 2.4 parser 包中定义的抽象类

类之间的关系如下图 2.5 所示：

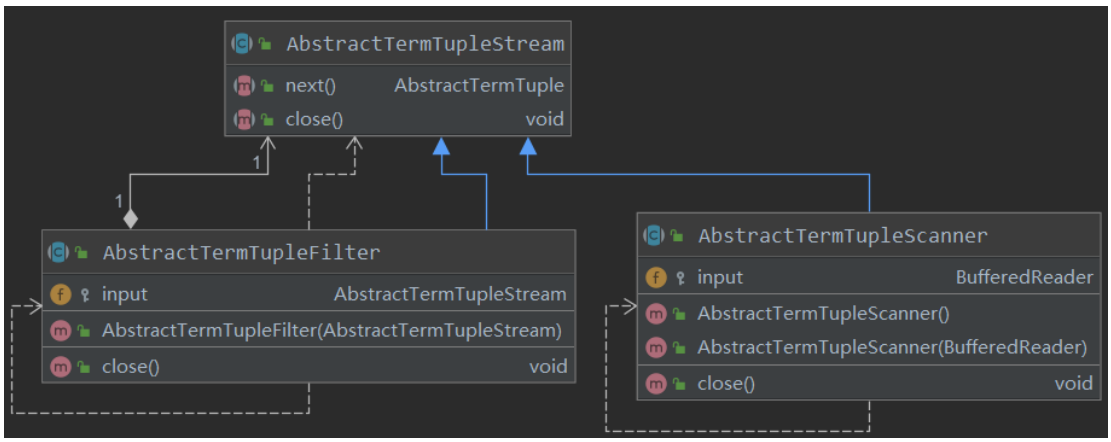

图 2.5 parser 包中类接口和类的 UML 图

3) package hust.cs.javacourse.search.query

包中定义的抽象类和接口如下图 2.6 所示：

接口概要

接口	说明
Sort	Sort定义了对搜索结果排序的接口

类概要

类	说明
AbstractHit	AbstractHit是一个搜索命中结果的抽象类.
AbstractIndexSearcher	AbstractIndexSearcher是检索具体实现的抽象类

枚举概要

枚举	说明
AbstractIndexSearcher.LogicalCombination	多个检索词的逻辑组合

图 2.6 query 包中定义的抽象类和接口

接口、类之间的关系如下图 2.7 所示：

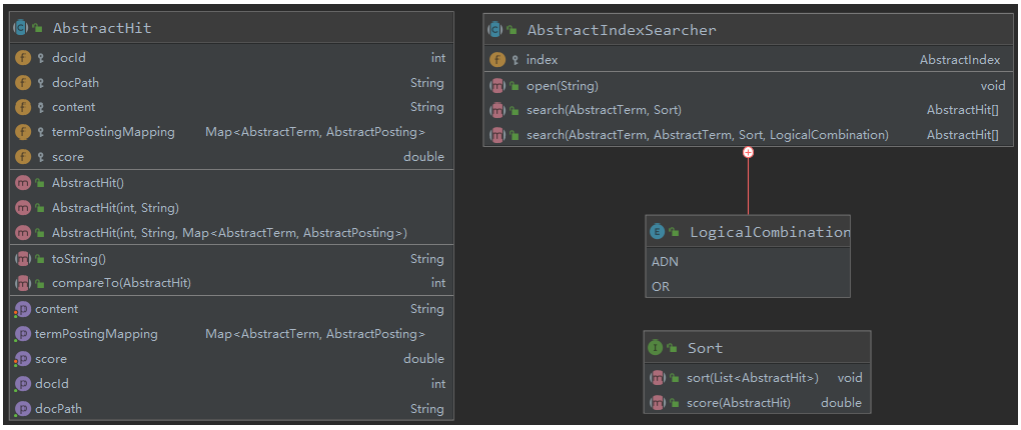

图 2.7 query 包中类接口和类的 UML 图

1.3 设计思路及原理

要建立索引对象，首先针对每个文档建立 Document 对象，Document 对象即文档中每个单词的三元组列表。如下图 2.8 所示：

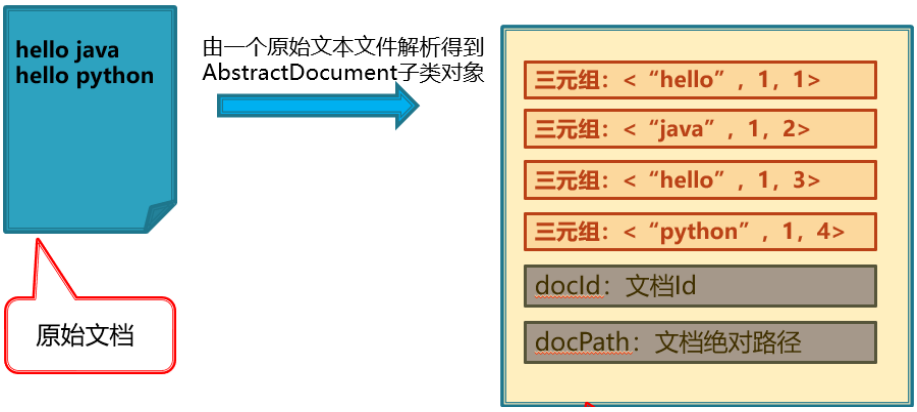

图 2.8 Document 对象的建立图

再使用 Document 对象建立索引，倒排索引结构如图 2.9 所示：

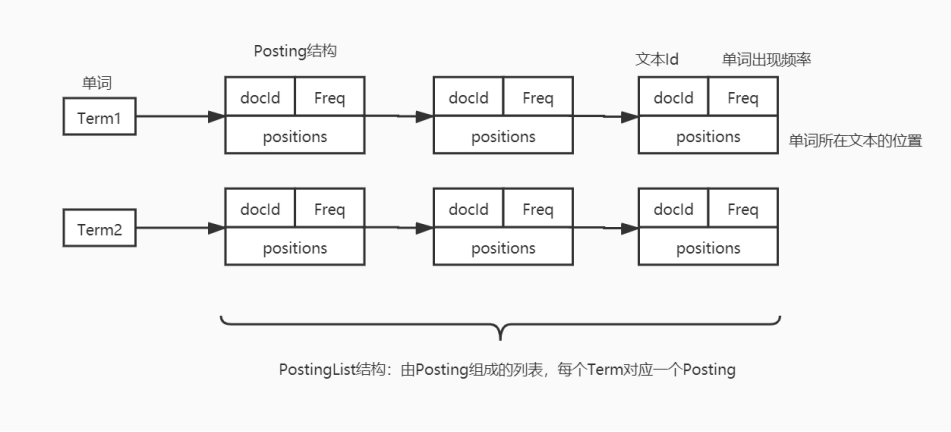

图 2.9 倒排索引结构图

在进行搜索时，如果搜索单个关键词，直接使用该关键词在 Index 中查找即可；如果要查

询两个单词的与查询（或查询），则分别使用两个关键词在 Index 中查找得到两个 PostingList，再使用一个二重循环比较文档 Id；如果要实现两个单词组成的短语查询，则不仅要比较文档 Id，还要比较单词所在位置 positions。

1.4 人机界面设计

1) TestIndexBuilder.java

该程序用于测试序列化和反序列化功能，给出 3 个选项：

1. 创建功能测试数据集的索引
2. 创建真实测试数据集的索引
3. 从已有的序列化索引文件反序列化

当用户选择 1 或 2 选项时，程序分别从\text\function 和\text\read 目录读取文本文件创建索引，将索引序列化写入\index\index.dat 文件中，并在终端打印索引；当用户选择 3 选项时，程序从\index\index.dat 文件中反序列化得到索引，并输出到终端。

2) TestIndexSearcher

用于测试搜索功能，首先由用户选择测试数据集，而后用户输入关键词查询，有 5 种选项：

1. 单个搜索关键词
2. 两个关键词的或查询（格式：Word1 | Word2）
3. 两个关键词的与查询（格式：Word1 & Word2）
4. 两个关键词的短语检索
5. 输入 exit 退出

重复上述选项选择直至用户输入 exit 退出。

2. 详细设计

2.1 模块 hust.cs.javacourse.search.index

1. Document：AbstractDocument 的实现类

Document 对象是解析一个文本文件得到结果，文档对象里面包含：文档 id、文档的绝对路径、文档包含的三元组对象列表，一个三元组对象是抽象类 AbstractTermTuple 的子类实例。

Document 中的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public getDocId	无	int	获取文档 id
public setDocId	int	void	设置文档 id
public getDocPath	无	String	获取文档绝对路径

public setDocPath	String	void	设置文档绝对路径
public getTuples	无	List<AbstractTermTuple>	获取文档包含的三元组列表
public addTuple	AbstractTermTuple	void	向文档对象里添加三元组，要求不能有内容重复的三元组
public contains	AbstractTermTuple	Boolean	判断是否包含指定的三元组
public getTuple	int	AbstractTermTuple	获取指定下标的三元组
public getTupleSize	无	int	获取文档对象包含的三元组个数
public toString	无	String	获得 Document 的字符串表示

2. DocumentBuilder：AbstractDocumentBuilder 的实现类

DocumentBuiler 类是 Document 对象的构造器，实现功能：解析文本文档得到的三元组流 TermTupleStream，产生 Document 对象。

它的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public build	int，文档 id String，文档绝对路径 File，文档对应 File 对象	AbstractDocument	由给定的 File，构造 Document 对象
public build	int，文档 id String，文档绝对路径 AbstractTermTupleStream，文档对应的 TupleStream	AbstractDocument	由解析文本得到的 TermTupleStream 构造 Document 对象

方法具体实现：

第一个 build 方法：由给定的 File 对象，通过 Java 中的 BufferedReader 对象来读取文本文件的内容，然后由这个 reader 以及过滤器类来构建一个三元组流 AbstractTermTupleStream，然后通过调用第二个 build 方法，最终实现文档对象的构造。

第二个 build 方法：先由给定的文档 id 和文档绝对路径来构建一个新的空的文档对象，然后通过三元组文档流中的 next 方法来遍历文档的三元组，并通过文档对象中的 addTuple 方法实现文档三元组的更新，当流遍历完，即 next 出来的对象是 null 时，证明文档流遍历完毕，关闭文档流并返回文档对象。

3. Index：AbstractIndex 的实现类

Index 对象是包含了文档集合的倒排索引。内存中的倒排索引结构为 HashMap， 其中 key

为 Term 对象，value 为对应的 PostingList 对象。另外在 AbstractIndex 里还定义了从 docId 和 docPath 之间的映射关系。

Index 类包含的字段有：

限定符和类型	字段和说明
protected java.util.Map<java.lang.Integer,java.lang.String>	docIdToDocPathMapping 内存中的 docId 和 docPath 的映射关系，key 为 docId，value 为对应的 docPath.
protected java.util.Map<AbstractTerm,AbstractPostingList>	termToPostingListMapping 内存中的倒排索引结构为 HashMap，key 为 Term 对象，value 为对应的 PostingList 对象.

Index 类中的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public toString	无	String	获得索引的字符串表示
public addDocument	AbstractDocument	void	添加文档到索引
public load	File	void	从索引文件里反序列化出索引
public save	File	void	将内存中索引序列化储存到文件
pubilc search	AbstractTerm	AbstractPostingList	获取指定单词的 PostingList
public getDictionary	无	Set<AbstractTerm>	获取索引中的所有单词
public optimize	无	void	对索引进行优化： PostingList 按照文档 id 排序 Posting 中的 positions 排序
public getdocName	int	String	根据文档 id 获取文档绝对路径
public writeObject	ObjectOutputStream	void	将内容写入二进制文件
public readObject	ObjectInputStream	void	从二进制文件读取内容

方法具体实现：

addDocument：

首先通过 HashMap 的 put 方法的 API 来实现 docIdToDocPathMapping 的更新；对于更新 termToPostingListMapping 分为以下几步：首先构造一个空的 Map，key 类型为 AbstractTerm，value 类型为 List<Integer>；然后遍历文档的三元组 tuples，如果 map 的 key 中已经含有某个 Term 了，那么更新相应的 value 即可，如果不包含某个 Term，那么通过 put 更新 map；然后遍历这个 map，如果 termToPostingListMapping 中已经含有某个 Term，用 add 的 API 更新 value，

如果是新的 Term，用 put 的 API 更新 termToPostingListMapping。addDocument 方法的程序流程图如图 2.10 所示：

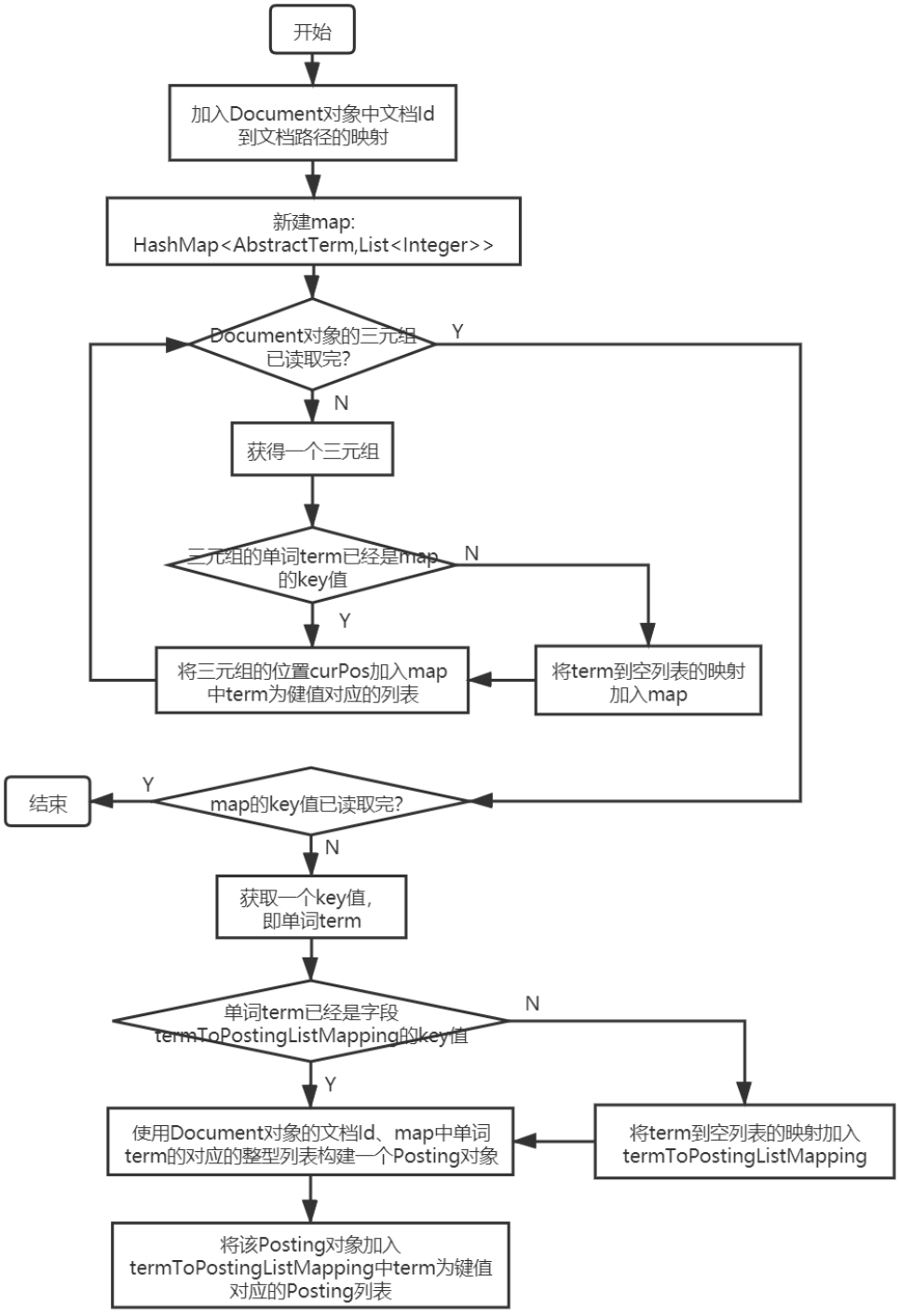

图 2.10 addDocument 方法的程序流程图

load：

通过调用 readObject 方法即可实现，这个方法会抛出 IOException，需要捕获异常。

save：

通过调用 saveObject 方法即可实现，这个方法会抛出 IOException，需要捕获异常。

search：

通过调用 Map 的 get 方法的 API 即可实现返回相应 key 的 value。

getDictionary：

通过调用 Map 的 keySet 方法的 API 即可实现返回 key 的集合。

optimize：

遍历 termToPostingListMapping，调用 PostingList 中的 sort 方法以及 Posting 中的 sort 方法。

writeObject：

通过调用 ObjectOutputStream 对象的 writeObject 方法的 API 即可实现序列化。

readObject：

通过调用 ObjectInputStream 对象的 readObject 方法的 API 即可实现反序列化。

4. IndexBuilder：AbstractIndexBuilder 的实现类

实现功能：构造索引对象。

IndexBuilder 类的字段有：

限定符和类型	字段和说明
protected AbstractDocumentBuilder	docBuilder 构建索引必须解析文档构建 Document 对象，因此包含 AbstractDocumentBuilder 的子类对象
protected int	docId docId 计数器，每当解析一个文档并写入索引，计数器应该+1

构造器：

构造器和说明
IndexBuilder(AbstractDocumentBuilder docBuilder)

IndexBuilder 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public buildIndex	String	AbstractIndex	构建指定目录下的所有文本文件的倒排索引

方法具体实现：

buildIndex：

首先构建一个 Index 对象，接着通过 FileUtil 类中的 list 方法，构建出一个类型为 String 的 List，List 中的每一项都是一个目录下文件对应的绝对路径，然后遍历这个 List，通过调用 docBuilder 中的 build 方法构建出一个文档对象，再通过 index 的 addDocument 方法来更新 index，最后通过 index 中的 save 方法将内存中的倒排索引保存至二进制文件。

5. Posting：AbstractPosting 的实现类

Posting 对象代表倒排索引里每一项，一个 Posting 对象包括:包含单词的文档 id，单词在文档里出现的次数，单词在文档里出现的位置列表（位置下标不是以字符为编号，而是以单词为单位进行编号）。实现下面二个接口:

Comparable：可比较大小（按照 docId 大小排序）

FileSerializable：可序列化到文件或从文件反序列化

Posting 类的构造器有：

构造器和说明
Posting() 缺省构造函数
Posting(int docId, int freq, java.util.List<java.lang.Integer> positions) 构造函数

Posting 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public equals	Object	boolean	判断两个 Posting 是否相同
public getDocId	无	int	获取文档 id
public setDocId	int	void	设置文档 id
public getFreq	无	int	获取单词的出现次数
pubilc setFreq	int	void	设置单词的出现次数
public getPositions	无	List<Integer>	获取单词的出现位置列表
public setPositions	List<Integer>	void	设置单词的出现位置列表
public compareTo	AbstractPosting	int	根据文档 id 比较两个 Posting
public sort	无	void	对内部 positions 进行排序
public readObject	ObjectInputStream	void	从二进制文件读取内容
public writeObject	ObjectOutputStream	void	写入二进制文件

6. PostingList：AbstractPostingList 的实现类

PostingList 对象包含了一个单词的 Posting 列表，必须实现下面接口:

FileSerializable：可序列化到文件或从文件反序列化。

PostingList 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public toString	无	String	返回 PostingList 的字符串表示

public add	AbstractPosting	void	添加 Posting，不能重复
public add	List< AbstractPosting >	void	添加 Posting 列表，不能重复
public get	int	AbstractPosting	获取指定下标的 Posting
pubilc indexOf	AbstractPosting	int	返回某个 Posting 的下标
public indexOf	int	int	获取指定文档 id 的 Posting 下标
public contains	AbstractPosting	boolean	判断是否包含指定 Posting 对象
public remove	AbstractPosting	void	移除指定 Posting
public remove	int	void	移除指定下标的 Posting
public size	void	int	获取 PostingList 的大小
public clear	void	void	清空 PostingList
public isEmpty	void	boolean	判断 PostingList 是否为空
public sort	void	void	根据文档 id 对 PostingList 排序
public writeObject	ObjectOutputStream	void	将内容序列化写入二进制文件
public readObject	ObjectInputStream	void	将二进制文件反序列化写入内存

7. Term：AbstractTerm 的实现类

Term 对象表示文本文档里的一个单词，必须实现下面二个接口:
Comparable：可比较大小（字典序）,为了加速检索过程，字典需要将单词进行排序。
FileSerializable：可序列化到文件或从文件反序列化。

Term 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public toString	无	String	返回 Term 的字符串表示
public equals	Object	boolean	判断两个 Term 是否相同
public getContent	void	String	获取 content 内容
public setContent	String	void	设置 content 内容
pubilc compareTo	AbstractTerm	int	比较两个 Term 大小（按字典序）
public writeObject	ObjectOutputStream	void	将内容序列化写入二进制文件
public readObject	ObjectInputStream	void	将二进制文件反序列化写入内存

8. TermTuple：AbstractTermTuple 的实现类

一个 TermTuple 对象为三元组(单词，出现频率，出现的当前位置)，当解析一个文档时，每解析到一个单词，应该产生一个三元组，其中 freq 始终为 1(因为单词出现了一次).

TermTuple 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public toString	无	String	返回 TermTuple 的字符串表示
public equals	Object	boolean	判断两个 TermTuple 是否相同

方法具体实现：

public boolean equals(Object obj)：

首先通过 instanceof 判断 obj 是否为 TermTuple 类型，不是返回 false，是的话继续比较各个属性是否相同，值属性用==判断即可，引用类型用 Object.equals 来判断。

2.2 模块 hust.cs.javacourse.search.parse

1. TermTupleScanner：AbstractTermTupleScanner 的实现类

AbstractTermTupleScanner 是 AbstractTermTupleStream 的抽象子类，即一个具体的 TermTupleScanner 对象就是一个 AbstractTermTupleStream 流对象，它利用 java.io.BufferedReader 去读取文本文件得到一个个三元组 TermTuple。

TermTupleScanner 类的字段有：

限定符和类型	字段和说明
protected java.io.BufferedReader	input input 作为输入流对象，读取文本文件得到一个个三元组 TermTuple
int	curPos 单词出现位置，初始为 0
private List<TermTuple>	tuples 解析文本文件获得的三元组

TermTupleScanner 类的构造器：

构造器和说明
TermTupleScanner(BufferedReader input) 根据 input 构造出 tuples

TermTupleScanner 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public next	void	AbstractTermTuple	获取下一个三元组
public close	void	void	关闭流

方法具体实现：

public TermTupleScanner(BufferedReader input)：

首先用 super 执行父类 AbstractTermTupleScanner 的构造函数，然后通过 BufferedReader 的 readline 方法的 API 对 input 进行逐行读取，读取一行之后，用 StringSplitter 工具类对一行的内容进行分割，形成一个 List，List 中的内容是一个个分割出来的单词，随后对 List 进行遍历，对 tuples 进行更新，循环直到 readline 返回值为 null，还要注意捕获 IOException 的异常。该方法的程序流程图如图 2.11 所示：

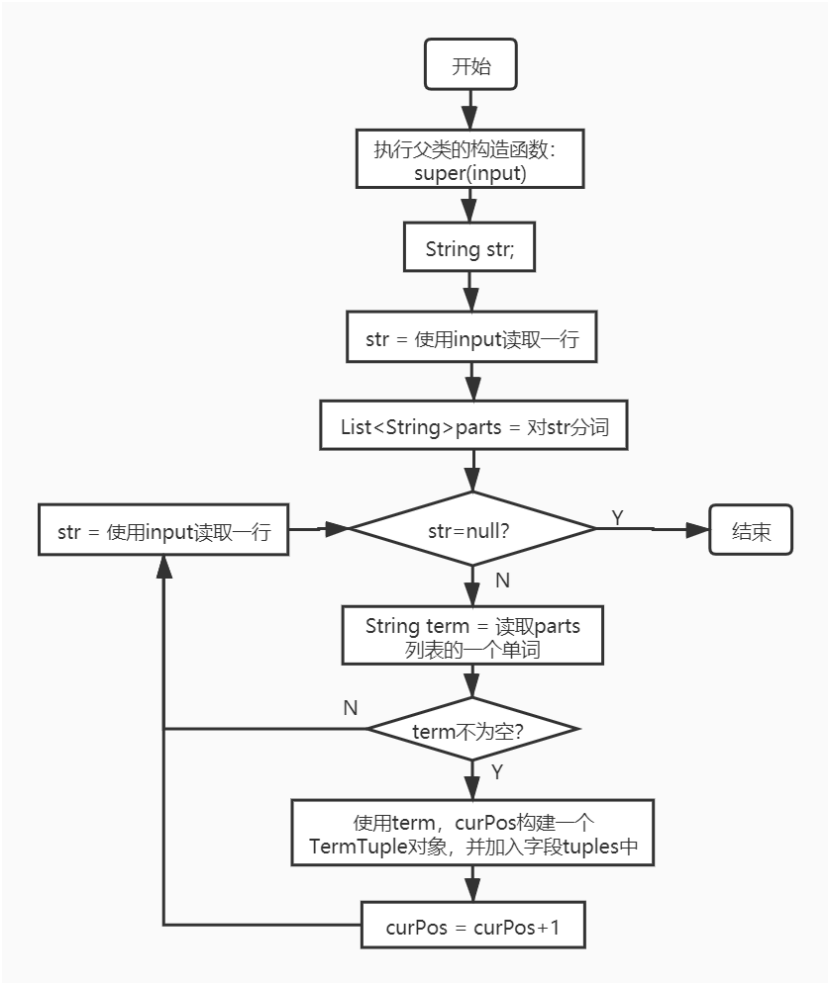

图 2.11 TermTupleScanner 构造方法程序流程图

public AbstractTermTuple next()：

如 tuples 的列表大小不为 0，返回 tuples 中的第一个元素并将其从 tuples 中移除，如果 size 为 0，则返回 null。

2. StopWordTermTupleFilter：AbstractTermTupleFilter 的实现类

用于过滤包含停顿词的三元组。

StopWordTermTupleFilter 类包含的字段：

限定符和类型	字段和说明
private List<String>	stopWord 所有停顿词的列表
protected AbstractTermTupleStream	input Filter 的输入，类型为 AbstractTermTupleStream

StopWordTermTupleFilter 类的构造器：

构造器和说明
StopWordTermTupleFilter(AbstractTermTupleStream input) 构造函数，将预置的停顿词列表更新至 stopWord

StopWordTermTupleFilter 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public next	void	AbstractTermTuple	获取下一个三元组，过滤掉停顿词

方法具体实现：

public AbstractTermTuple next()：

根据 input 的 next 获取下一个三元组，如果是 null，则返回 null；如果三元组的 Term 是一个停顿词，则用 while 循环获取下一个，直至不是停顿词，返回该三元组。

3. PatternTermTupleFilter：AbstractTermTupleFilter 的实现类

用于过滤 Term 内容包含除字母之外的三元组。

PatternTermTupleFilter 包含的字段有：

限定符和类型	字段和说明
protected AbstractTermTupleStream	input Filter 的输入，类型为 AbstractTermTupleStream

PatternTermTupleFilter 类的构造器：

构造器和说明
PatternTermTupleFilter(AbstractTermTupleStream input) 构造函数

PatternTermTupleFilter 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public next	void	AbstractTermTuple	获取下一个三元组，过滤掉 Term 内容包含除字母之外的三元组

方法具体实现：

public AbstractTermTuple next()：

根据 input 的 next 获取下一个三元组，如果是 null，则返回 null；如果三元组的 Term 是一个包含除字母之外的单词，则用 while 循环获取下一个，直至 Term 是只包含字母的单词，返回该三元组。

4. LengthTermTupleFilter：AbstractTermTupleFilter 的实现类

用于过滤 Term 内容过长（大于 20）或过短（小于 3）的三元组。

LengthTermTupleFilter 类包含的字段有：

限定符和类型	字段和说明
protected AbstractTermTupleStream	input Filter 的输入，类型为 AbstractTermTupleStream

LengthTermTupleFilter 的构造器：

构造器和说明
LengthTermTupleFilter(AbstractTermTupleStream input) 构造函数

LengthTermTupleFilter 的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public next	void	AbstractTermTuple	获取下一个三元组，过滤掉 Term 内容过长或过短的三元组

方法具体实现：

public AbstractTermTuple next()：

根据 input 的 next 获取下一个三元组，如果是 null，则返回 null；如果三元组的 Term 过长或过短，则用 while 循环获取下一个，直至 Term 符合条件，返回该三元组。

2.3 模块 hust.javacourse.search.query

1. Hit：AbstractHit 的实现类

AbstractHit 是一个搜索命中结果的抽象类。该类子类要实现 Comparable 接口。实现该接口是因为需要必须比较大小，用于命中结果的排序。

Hit 类包含的字段有：

限定符和类型	字段和说明
protected java.lang.String	content 文档原文内容，显示搜索结果时有用
protected int	docId 文档 id
protected java.lang.String	docPath 文档绝对路径
protected double	score 该命中文档的得分，文档的得分通过 Sort 接口计算.每个文档得分默认值为 1.0
protected java.util.Map<AbstractTerm,AbstractPosting>	termPostingMapping 命中的单词和对应的 Posting 键值对，对计算文档得分有用，对于一个查询命中结果，一个 term 对应的是 Posting 而不是 PostingList

Hit 类构造器：

构造器和说明
Hit() 默认构造函数
Hit(int docId, java.lang.String docPath) 构造函数
Hit(int docId, java.lang.String docPath, java.util.Map<AbstractTerm,AbstractPosting> termPostingMapping) 构造函数

Hit 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public getDocId	无	int	获取文档 id
public getContent	无	String	获取文章内容
public getDocPath	无	String	获取文档绝对路径
public setContent	String	void	设置文档内容
public getScore	无	double	获取文章得分
public setScore	double	void	设置文章得分
public getTermPositingMapping	void	Map<AbstractTerm, AbstractPosting>	获取命中的单词对应的 Posting 键值对
public compareTo	AbstractHit	int	根据得分比较两个 Hit，得分相同根据文章 id 比较
public toString	无	String	获得 Hit 的字符串表示

方法具体实现：

String toString()：

为了实现 Hit 对象的 toString 方法，可以调用已经实现的其他对象的 toString 方法。这里重点讲下输出文档内容中搜索词的高亮显示功能。由于搜索引擎默认大小写不敏感，所以构造的索引对象的单词 term 和用于搜索的关键词都是小写的，但是需要输出的源文档仍要分大小写。为了解决这个问题，生成了两个原始文档内容的副本：

String content_o = this.content;　//复制文档原文内容，用于替换，最后输出

String tmp = this.content.toLowerCase();　//文档原文内容全部变为小写形式

这样，对于 this.Map 中存储的每个命中的单词 regex（必定是小写形式），使用 tmp.indexOf() 方法找到 regex 在 content 中的 beginIndex 和 endIndex，再使用 content.substring(beginIndex,endIndex)方法得到 regex 在原始文档中的表示 regex1，最后使用 content_o.repalceAll 方法将 regex1 替换为高亮显示。

最后返回 content_o，在 IDEA 打印时有高亮功能。能顺利通过自动测试，但是打包为 jar 文件后，控制台不支持高亮，反而会乱码，所以在打包生成 jar 文件时取消了高亮显示功能。

2. IndexSearcher：AbstractIndexSearcher 的实现类

用于对内存中的倒排索引进行搜索。该类有一个嵌套类：

AbstractIndexSearcher.LogicalCombination：用于表示对两个搜索词的与或关系。

IndexSearcher 类包含的字段有：

限定符和类型	字段和说明
protected AbstractIndex	index 内存中的索引，子类对象被初始化时为空

IndexSearcher 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public open	String	void	从指定索引文件打开索引并加载到 index 对象中
public search	AbstractTerm, Sort	AbstractHit[]	根据单个检索词进行搜索
public search	AbstractTerm, AbstractTerm, Sort, LogicalCombination	AbstractHit[]	根据两个检索词以及两个检索词的逻辑关系进行搜索

3. SimpleSort：Sorter 接口的实现类

根据得分对命中列表进行排序。

SimpleSort 类的主要方法有：

限定符和方法名	接收参数	返回类型	功能说明
public sort	List<AbstractHit>	void	对命中的集合根据分数进行排序
public score	AbstractHit	double	给命中的文章根据检索词出现频率打分

三、 软件开发

开发环境介绍：

1. 操作系统：Windows10(64 位)
2. JDK 版本：jdk13.01

```
C:\Users\13780>java -version
java version "13.0.1" 2019-10-15
Java(TM) SE Runtime Environment (build 13.0.1+9)
Java HotSpot(TM) 64-Bit Server VM (build 13.0.1+9, mixed mode, sharing)
```

图 3.1 JDK 版本图

3. IntelliJ IDEA Ultimate 2020.3
4. 使用 IDEA 的功能生成 jar 包，选择 TestInexSearcher 作为主类，在 jar 包目录下，运行命令：java -jar SearchEngineForStudent.jar

四、 软件测试

4.1 自动测试

自动测试结果如图 4.1 所示：

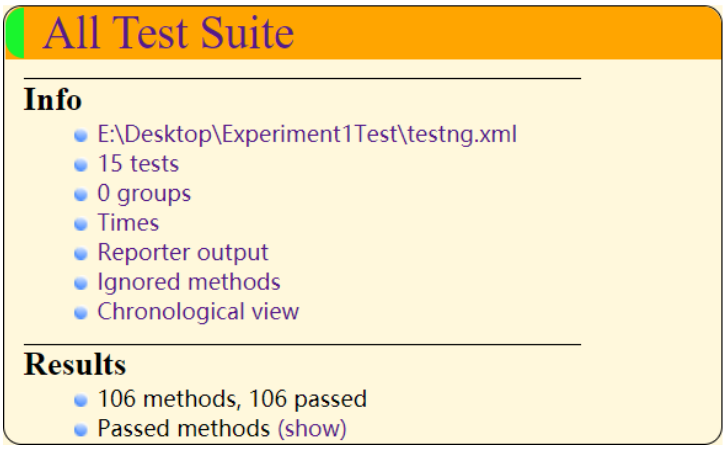

图 4.1 自动测试结果图

4.2 功能 1 测试

功能 1：将指定目录下的一批.txt 格式的文本文件扫描并在内存里建立倒排索引，内存里建立好的索引对象可以序列化到文件。

1) 测试 1：

测试用例：功能测试数据集。

测试输出：未经过滤的索引如下图 4.2 所示：

```
创建倒排索引，选择模式：
1. 从功能测试数据集读取文档内容创建
2. 从真实测试数据集读取文档内容创建
3. 从已有的序列化索引文件反序列化进行创建
请输入数字：1
文档目录：
E:\Desktop\Java新实验一\SearchEngineForStudent/text/
倒排索引内容：
docIdToDocPath:
docId: 0, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\1.txt
docId: 1, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\2.txt
docId: 2, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\3.txt
termToPosting:
12-31= [(docId=0,freq=1,positions=[12])],
2005= [(docId=0,freq=1,positions=[11]), (docId=2,freq=1,positions=[2])],
26%= [(docId=1,freq=1,positions=[3])],
activity= [(docId=0,freq=3,positions=[0, 1, 6]), (docId=1,freq=1,positions=[0])],
because= [(docId=1,freq=1,positions=[5])],
benefits= [(docId=0,freq=2,positions=[2, 7]), (docId=1,freq=1,positions=[10])],
capital= [(docId=0,freq=2,positions=[3, 8]), (docId=1,freq=1,positions=[7])],
destination= [(docId=0,freq=2,positions=[4, 9]), (docId=1,freq=1,positions=[1])],
emergency= [(docId=0,freq=1,positions=[10]), (docId=1,freq=1,positions=[2])],
except= [(docId=2,freq=1,positions=[4])],
```

图 4.2 功能 1 测试 1 结果图

2) 测试 2：

测试用例：功能测试数据集。

测试输出：经过过滤的索引如下图 4.3 所示：

```
倒排索引内容：
docIdToDocPath:
docId: 0, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\1.txt
docId: 1, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\2.txt
docId: 2, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\3.txt
termToPosting:
activity= [(docId=0,freq=3,positions=[0, 1, 6]), (docId=1,freq=1,positions=[0])],
benefits= [(docId=0,freq=2,positions=[2, 7]), (docId=1,freq=1,positions=[10])],
capital= [(docId=0,freq=2,positions=[3, 8]), (docId=1,freq=1,positions=[7])],
destination= [(docId=0,freq=2,positions=[4, 9]), (docId=1,freq=1,positions=[1])],
emergency= [(docId=0,freq=1,positions=[10]), (docId=1,freq=1,positions=[2])],
fizzy= [(docId=0,freq=1,positions=[5])],
frozen= [(docId=1,freq=2,positions=[9, 14])],
google= [(docId=1,freq=2,positions=[11, 15])],
marketplace= [(docId=2,freq=1,positions=[0])],
medical= [(docId=2,freq=1,positions=[1])],
notification= [(docId=2,freq=1,positions=[3])],
peninsula= [(docId=2,freq=1,positions=[5])],
pollution= [(docId=2,freq=1,positions=[7])],
```

图 4.3 功能 1 测试 2 结果图

3) 测试 3：从文件里反序列化成内存里的索引对象

测试用例：功能测试数据集 1.txt 2.txt 3.txt 经过滤构建出来的索引文件 index.dat。

测试输出：反序列化出的索引如下图 4.4 所示：

```
创建倒排索引，选择模式：
1. 从功能测试数据集读取文档内容创建
2. 从真实测试数据集读取文档内容创建
3. 从已有的序列化索引文件反序列化进行创建
请输入数字：3
倒排索引内容：
docIdToDocPath:
docId: 0, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\1.txt
docId: 1, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\2.txt
docId: 2, path = E:\Desktop\Java新实验一\SearchEngineForStudent\text\function\3.txt
termToPosting:
activity= [(docId=0,freq=3,positions=[0, 1, 6]), (docId=1,freq=1,positions=[0])],
benefits= [(docId=0,freq=2,positions=[2, 7]), (docId=1,freq=1,positions=[10])],
capital= [(docId=0,freq=2,positions=[3, 8]), (docId=1,freq=1,positions=[7])],
destination= [(docId=0,freq=2,positions=[4, 9]), (docId=1,freq=1,positions=[1])],
emergency= [(docId=0,freq=1,positions=[10]), (docId=1,freq=1,positions=[2])],
fizzy= [(docId=0,freq=1,positions=[5])],
frozen= [(docId=1,freq=2,positions=[9, 14])],
google= [(docId=1,freq=2,positions=[11, 15])],
marketplace= [(docId=2,freq=1,positions=[0])],
medical= [(docId=2,freq=1,positions=[1])],
notification= [(docId=2,freq=1,positions=[3])],
peninsula= [(docId=2,freq=1,positions=[5])],
pollution= [(docId=2,freq=1,positions=[7])],
```

图 4.4 功能 1 测试 3 结果图

4.3 功能 2 测试

功能 2：基于构建好的索引，实现单个搜索关键词的全文检索，包含的子功能包括：

（1）根据搜索关键词得到命中的结果集合；

（2）可以计算每个命中的文档的得分，并根据文档得分对结果集排序；

（3）在控制台显示命中的文档的详细信息，如文档的路径、文档内容、命中的关键词信息（如在文档里出现次数）、文档得分；

测试用例均为真实测试数据集下的 1-15 文本文件。

1) 测试 1：选择测试数据集。

选择真实测试数据集，构建索引对象写入序列化文件，从序列化文件中读取数据建立 InexSearcher 对象，如图 4.5 所示：

```
选择数据集，将索引对象序列化到文件：
1. 功能测试数据集
2. 真实测试数据集
3. 输入exit退出
请输入你的选择: 2
------------------索引对象已写入序列化文件--------------------
从序列化文件中读取倒排索引，进行查询，输入格式：
1. 单个搜索关键词
2. 两个关键词的与查询 ( 格式：Word1 & Word2 )
3. 两个关键词的或查询 ( 格式：Word1 | Word2 )
4. 两个单词的短语检索 ( 格式：Word1 Word2 )
5. 输入exit退出查询
请输入需要查询的单词:
```

图 4.5 功能 2 测试 1 结果图

2) 测试 2：对单个检索词进行搜索

输入检索词合法，如：coronavirus，检索结果如图 4.6 所示，检索词在文档中高亮显示。

```
请输入需要查询的单词: coronavirus
------------------------------------------------------------
Hit{
docId=0
docPath='E:\Desktop\Java新实验一\SearchEngineForStudent\text\real\1.txt
content='The novel coronavirus death toll has reached 21 as of Saturday in Britain as the number of co

The new figures showed an increase of 342 confirmed COVID-19 cases in Britain, the largest rise on a s

All the 10 patients who died were aged over 60 and had underlying health conditions, said Chris Whitty

According to health authorities, most of the cases are in England. There have been 121 confirmed cases

The British government said on Friday that it estimated the true number of infected cases in Britain t
score=2.0
termPostingMapping:
{coronavirus=(docId=0,freq=2,positions=[2, 71])}
}
------------------------------------------------------------
Hit{
docId=6
docPath='E:\Desktop\Java新实验一\SearchEngineForStudent\text\real\15.txt
content='The release of the new James Bond film has been put back by seven months as coronavirus conti
The producers said they had moved the release of No Time To Die from April to November after "careful
```

图 4.6 功能 2 测试 2 结果图

3) 测试 3：对单个检索词进行搜索，检索词为停顿词，如：and。

```
请输入需要查询的单词: and
Warning: 停用词: and
未搜索到任何结果
```

图 4.7 功能 2 测试 3 结果图

4.4 功能 3 测试

功能 3：基于构建好的索引，实现二个搜索关键词的全文检索。包含的子功能包括：

（1）支持这二个关键词的与或查询。与关系必须返回同时包含这二个单词的文档集合，或关系返回包含这二个单词中的任何一个的文档集合；

（2）可以计算每个命中的文档的得分，并根据文档得分对结果集排序；

（3）在控制台显示命中的文档的详细信息，如文档的路径、文档内容、命中的关键词信息（如在文档里出现次数）、文档得分；

同样，测试用例均为真实测试数据集下的 1~15 文本文件。

1) 测试 1：对两个检索词进行逻辑与搜索

输入检索词合法，如：coronavirus & government，测试结果如下图 4.8 所示：

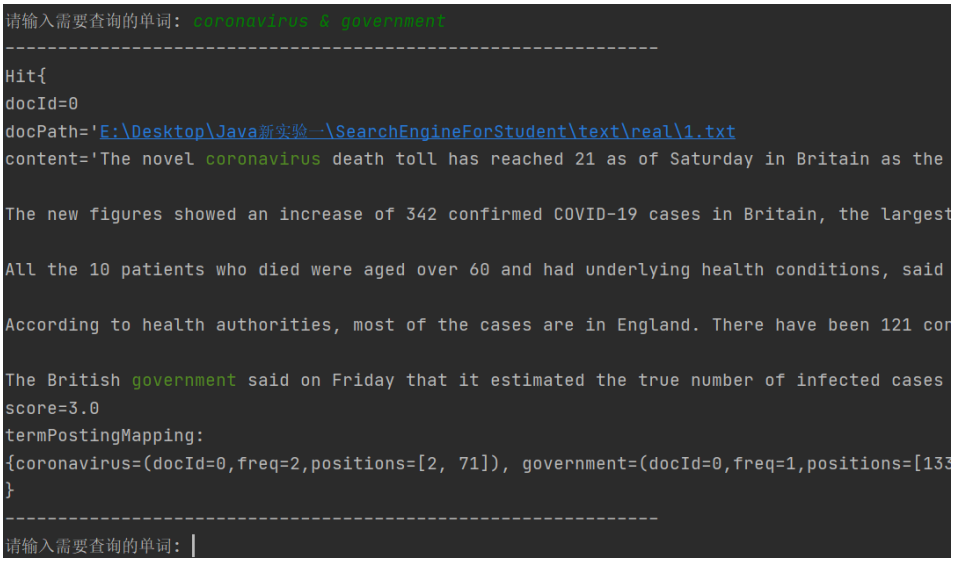

图 4.8 功能 3 测试 1 结果图

2) 测试 2：输入的两个检索词中有停等词：

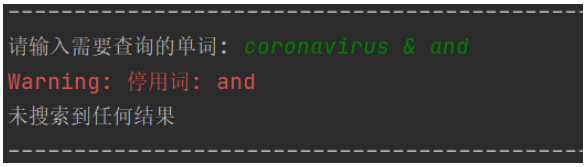

图 4.9 功能 3 测试 2 结果图

3）测试 3：对两个检索词进行逻辑与搜索

输入检索词合法，如：date & government，测试结果如图 4.10 所示：

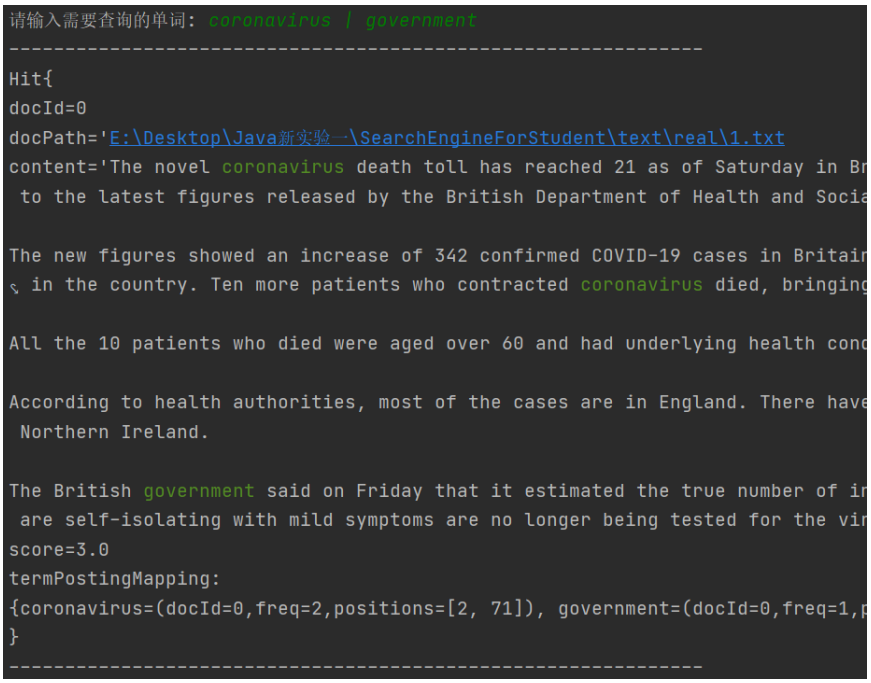

图 4.10 功能 3 测试 3 结果图

4）测试 4：对两个检索词进行逻辑或搜索

输入检索词合法，如：coronavirus & government，测试结果如图 4.11 所示：

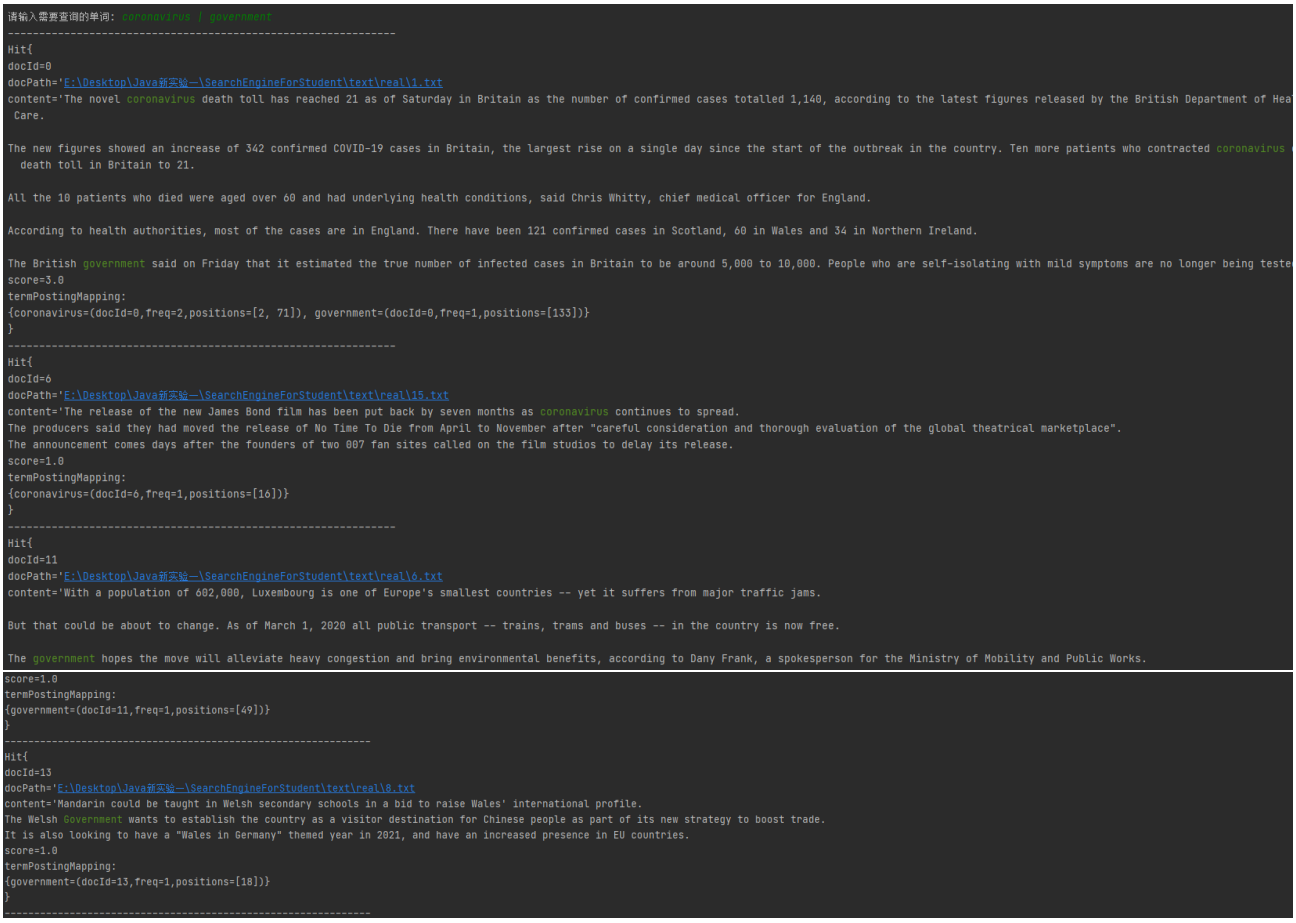

图 4.11 功能 3 测试 4 结果图

5) 测试 5：输入模式不合法：

```
请输入需要查询的单词: coronavirus + government
逻辑关系解析失败
```

图 4.12 功能 3 测试 5 结果图

4.5 功能 4 测试

功能 4：基于构建好的索引，实现二个单词组成的短语的全文检索。包含的子功能包括：

（1）支持这二个关键词的短语检索。返回包含这二个单词组成的短语的文档集合；

（2）可以计算每个命中的文档的得分，并根据文档得分对结果集排序；

（3）在控制台显示命中的文档的详细信息，如文档的路径、文档内容、命中的关键词信息（如在文档里出现次数）、文档得分；

测试数据集为真实测试数据集。

1) 测试 1：输入合法

输入 james bond，测试结果如图 4.13 所示：

```
请输入需要查询的单词: james bond
------------------------------------------------------------
Hit{
docId=6
docPath='E:\Desktop\Java新实验一\SearchEngineForStudent\text\real\15.txt
content='The release of the new James Bond film has been put back by seven month
The producers said they had moved the release of No Time To Die from April to No
 the global theatrical marketplace".
The announcement comes days after the founders of two 007 fan sites called on th
score=1.0
termPostingMapping:
{james bond=(docId=6,freq=1,positions=[5])}
}
```

图 4.13 功能 4 测试 1 结果图

2) 测试 2：输入不合法

比如输入两个以上的单词：james bond film，测试结果如图 4.14 所示：

```
----------------------------------------
请输入需要查询的单词: james bond film
逻辑关系解析失败
```

图 4.14 功能 4 测试 2 结果图

3) 测试 3：输入的两个单词中含有停等词，如：the new

```
请输入需要查询的单词: the new
Warning: 停用词: the
未搜索到任何结果
```

图 4.15 功能 4 测试 3 结果图

五、 特点与不足

1. 技术特点

了解了实际项目编写中一些常见的设计模式的思想，并将其运用到本实验当中，如装饰者模式等。通过比较单词的位置列表，完成了进阶功能。此外，还实现了打印命中文档时高亮显示搜索单词，因为搜索引擎存储索引对象和进行搜索时大小写不敏感，但原始文档的大小写还是要正常输出，为了解决这个问题我想了很久。另外，在 run 包里，有多个用于测试的 java 源程序，实现多种功能。

2. 不足和改进的建议

在实验过程中有些算法的代码不够精简，没有考虑到算法的时间空间复杂度。而且在实现具体类的过程中，只是一味地跟着老师预先设置好的抽象类进行实现，自己没有什么独立思考。个人的建议是希望可以删除一部分设计思路，留给学生们自己去发挥，但是这样就不能实现自动测试的功能了。另外，希望自动测试程序报错时能给出更多错误提示，方便代码修改。

六、 过程和体会

1. 遇到的主要问题和解决方法

在是实现 termTupleScanner 以及 termTupleFilter 的时候遇到了不小的困难，先前没有了解过对数据流的处理，对流没有一种具体的概念，也没有了解过装饰者模式的设计模式，看到 ppt 一长串的 new 的时候简直一头雾水，经过仔细研读 ppt 以及到网上查阅相关资料，终于理清了装饰者模式的设计思想，最终完成了相关模块的编写；还有一个问题在于序列化以及反序列化操作，一开始是对这些完全没有概念，通过老师的帮助文档，让我对序列化反序列化操作有了初步的理解并完成了相关模块的实现。

2. 课程设计的体会

此次实验让我深刻体验到了什么是所谓的面向对象编程，其主要思想就是把一个庞大的系统抽离成一个个具体的模块，每个模块中有自己的属性以及方法，最后再将这些模块整合到这个系统中去。这次的实验是对学生代码能力的一种考验，不同于之前的 C 语言实验或者是数据结构实验，此次实验培养了学生们写真实项目代码的能力，虽然离代码真正落地上线还有很长一段距离，但是通过这次试验让学生们了解了许多知识如常见的设计模式，序列化以及反序列化操作。学生通过逐步完善每一个类最终完成一个项目，是挺有成就感的一件事。

七、 源码和说明

1. 文件清单及其功能说明

提交的文件清单如图 7.1 所示，Experiment1Test 是自动测试文件，源程序编译后的.class 文件已放入\Experiment1Test\betest 目录下；SearchEngineForStudent 是工程目录。

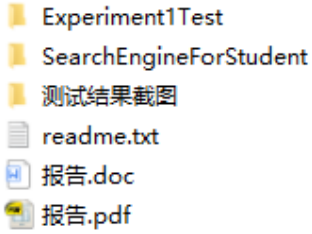

图 7.1 提交的文件清单图

打开工程目录 SearchEngineForStudent，如下图 7.2 所示：

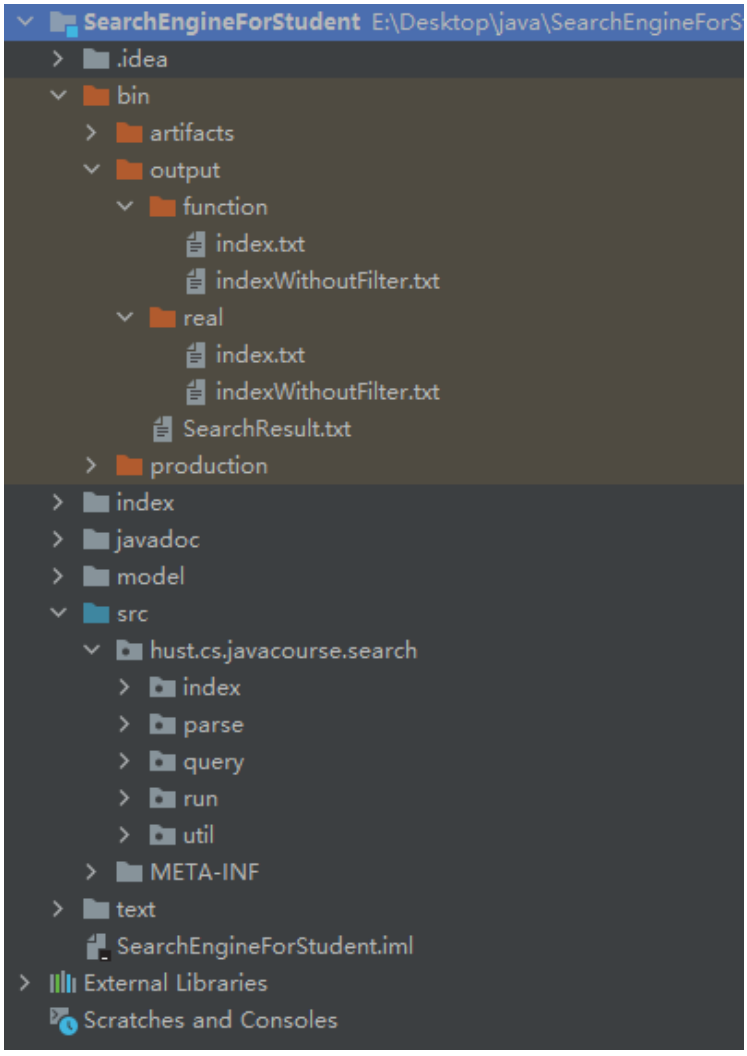

图 7.2 工程目录 SearchEngineForStudent 图

其中各个文件夹表示：

1. bin：程序的输出目录，包括：
   a) output：保存两个数据集构建的索引对象的文本文件和指定搜索词的搜索结果；
   b) production：存有 java 源程序编译后的.class 文件；
   c) artifacts：程序打包后的 jar 包，支持控制台运行；
2. index：存储索引对象的序列化文件；
3. javadoc：javadoc 说明；
4. model：存储 UML 和 UML 图；
5. text：保存功能测试数据集和真实测试数据集；
6. bin：程序的输出目录，包括：
   a) output：保存索引对象的文本文件表示和指定搜索词的搜索结果；
   b) production：存有 java 源程序编译后的.class 文件；
   c) artifacts：程序的 jar 包。
7. src：程序包的顶级目录；

2. 用户使用说明书

1） 在 IDEA 上运行

使用 IntelliJ IDEA Ultimate 2020.3 软件，打开 SearchEngineForStudent 工程目录，可以运行 hust.cs.javacourse.search.run 包下的源程序，如图 7.3 所示。

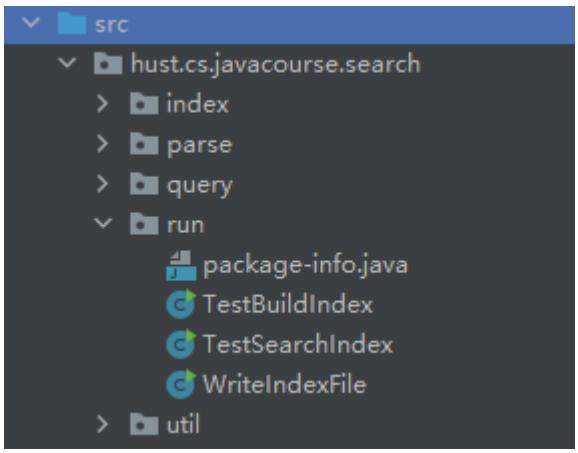

图 7.3 hust.cs.javacourse.search.run 包下的源程序图

其中 WriteIndexFile 用于实现将一些结果输出到.\bin\output 目录下，用户不需要运行 WriteIndexFile 源程序；TestBuildIndex 可以运行，用于测试构建索引对象以及序列化和反序列化功能；TestSearchIndex 可以运行，用于在选择的测试数据集上检索关键词。

2) 命令行运行

程序已打包成 jar 包，并设置了主类为 TestSearchIndex。在工程目录下的\bin\artifacts\SearchEngineForStudent_jar 目录下，打开终端，输入命令为：

java -jar SearchEngineForStudent.jar

命令行运行如图 7.4 所示：

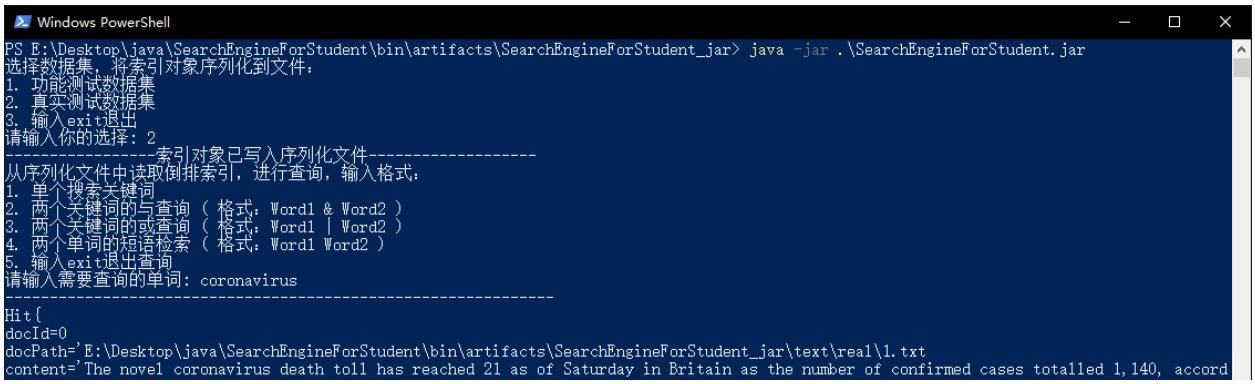

图 7.4 命令行运行程序图