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 ：　CS2008

学　　号 ：　U202015533

姓　　名 ：　徐瑞达

指导教师 ：　辜希武

2022 年 6 月 1 日

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:扫描指定目录下扩展名为.txt 的文本文件并在内存里建立倒排索引

其中需要实现的功能如下：

（1） 在指定目录下，逐个读取文本文件，建立存储每个文本文件信息的数据结构(Document)，用于存储每个文本文件的以下信息：文档编号、文档路径、文档包含的三元组；

（2） 在存储文本文件包含的三元组时，需要将文档内容切分成一个个的单词，然后构建每个单词的三元组(TermTuple)：单词内容、出现的次数（始终为 1）、出现的位置；

（3） 在向 Document 对象中存储三元组时，需要使用过滤器过滤掉不满足停用词规则、单词长度限制、正则表达式限制（这里过滤掉数字、汉字等内容）的单词；

（4） 根据由文本文件构建的 Documents 建立索引(Index)：索引包含两个哈希表——文档编号和文档路径的映射，用于查询某编号对应的文档；单词和对应 PostingList 的映射，用于查询某个单词出现的信息；

（5） 构建单词和对应 PostingList 的映射时，需要遍历 Document 的三元组以统计单词出现的信息，并要对 PostingList 进行排序操作以便于查询；

（6） 索引对象的序列化与反序列化；

（7） 通过复用 toString()方法实现将索引内容转化为格式化良好的字符串，并能够输出到文本文件中。

需求 2:基于构建的索引，实现单个搜索关键词的全文检索

（1）根据搜索关键词查询索引并将命中结果存储至特定的数据结构(Hit 数组)中;

（2）计算每个命中的文档的得分时，根据关键词在文档中的出现次数计分，分数相同时文档编号较小的命中结果在前；

（3）对查询结果进行格式良好的输出展示，未查询到单词时应输出提示。

需求 3:基于构建的索引，实现两个搜索关键词的全文检索

（1）应根据用户的 AND 或 OR 查询需求，进行查询结果的取舍；

（2）对查询结果进行格式良好的输出展示，未查询到单词时应输出提示。

需求 4：基于构建的索引，实现二个相邻关键词的全文检索

（1）支持二个相邻关键词的查询。先对两个关键词进行查询，对于查询结果，当命中结果在同一文档中时，计算两个关键词的位置差，若相差为 1，则说明在该文档中存在这两个相邻关键词；

（2）对查询结果进行格式良好的输出展示，未查询到单词时应输出提示。

其他的需求：

（1）记录查询历史，便于用户查询后查看结果；

（2）输出索引至文本文件中，便于用户查看索引结果；

二、系统设计

1. 概要设计

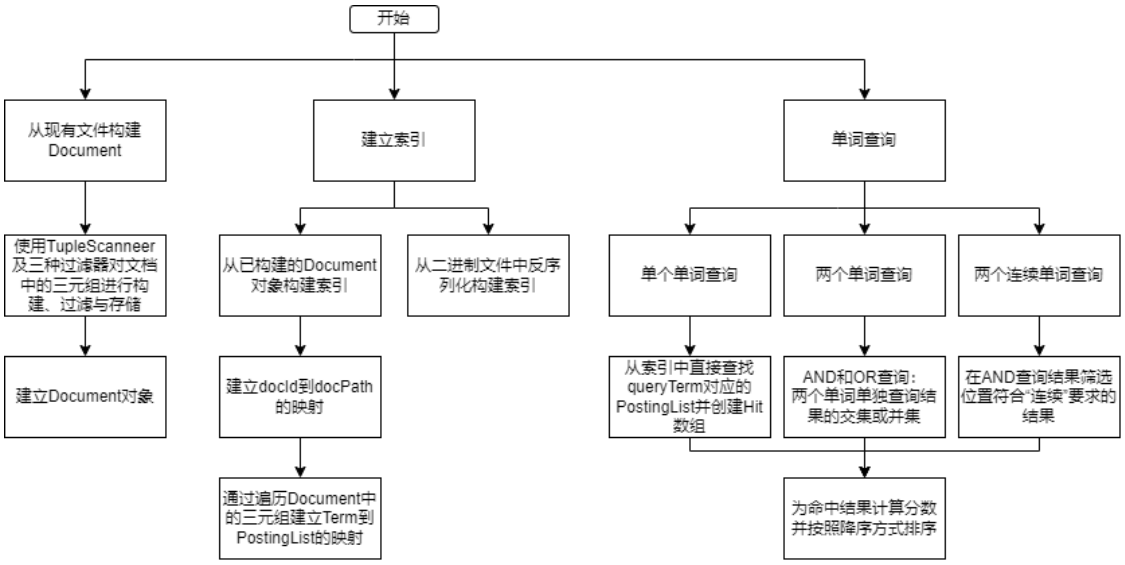

图 1 总体结构流程图

项目总体结构分为三部分：构建 Document，建立索引 Index，单词查询。

在构建 Document 部分，需要使用 parse 包中 AbstractTermTupleStream 的具体子类，其中，TermTupleScanner 用于解析文档中的单词并构建对应的三元组，LengthTermTupleFilter、PatternTermTupleFilter、StopWordTermTupleFilter 分别用于单词长度过滤、单词模式过滤、停用词过滤；在 DocumentBuilder 中使用 AbstractTermTupleStream 的具体子类获取符合要求的三元组并存储在 Document 对象中；

在建立索引部分，可以从构建的 Document 对象构建索引，此时需要遍历 Document 中的三元组建立 term 对应 PostingList 的映射表；也可以从二进制文件中反序列化构建索引，使用 Index 类的方法 load()即可；

在单词查询部分，根据三种不同的单词查询模式分别查询，并对结果进行打分排序以输出命中结果。

2. 详细设计

(1) 存储结构及其构建（index 包中的类）

① Term

Term 类是对单词内容（String）的简单包装，该类继承自 AbstractTerm 抽象类，具体子类实例为一个单词。

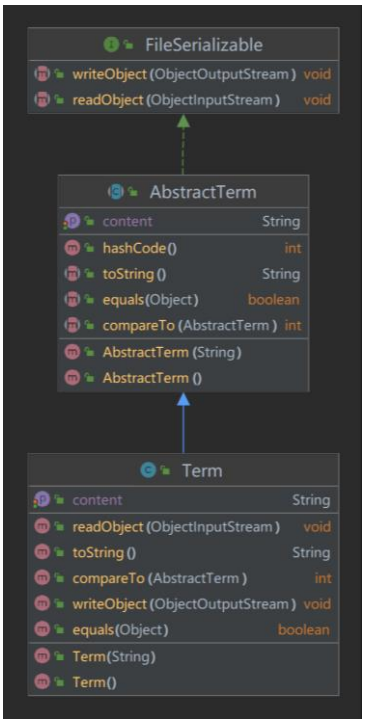

图 2　Term 类 UML 图

② TermTuple

TermTuple 类包含三个数据成员——单词内容 term、出现频率 freq（始终为 1）、在文档中出现的位置 curPos。该类继承自 AbstractTermTuple 抽象类，具体子类实例为某个单词某次出现信息的三元组。

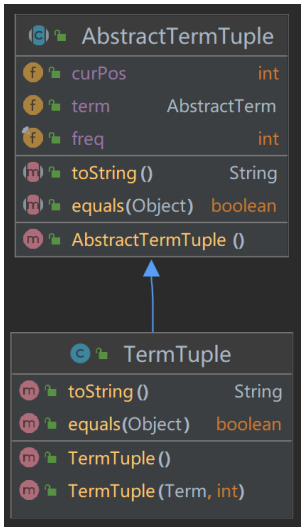

图 3　TermTuple 类 UML 图

③ Posting

Posting 类包含三个数据成员——文档编号 docId，单词在该文档中出现的次数 freq，单词出现的位置数组 positions。该类继承自 AbstractPosting 抽象类，具体子类实例表示存储某个单词在某个文档中的位置信息。

主要方法如下：

sort 方法：对 positions 进行升序排序。

图 4　Posting 类 UML 图

④ PostingList

Posting 类包含一个数据成员——Posting 数组。该类继承自 AbstractPostingList 抽象类，具体子类实例表示存储某个单词在指定目录下所有文档中的位置信息。

主要方法如下：

add 方法：可以向 PostingList 中添加单个 Posting 或者加入 Posting 数组，并且仅当待加入的 Posting 不在当前 PostingList 时加入。

indexOf 方法：通过具体 Posting 或者 docId 获取其下标。

remove 方法：删除某 docId 对应的 Posting 或者直接根据 Posting 删除。

sort 方法：根据 docId 对 Posting 数组进行升序排序。

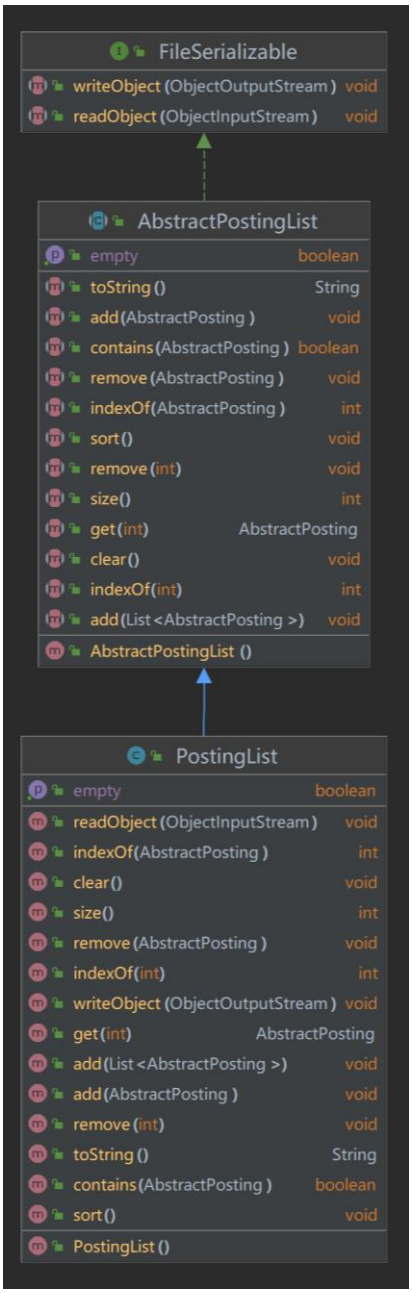

图 5　PostingList 类 UML 图

⑤ Document

Document 类包含三个数据成员——文档编号 docId、文档路径 docPath、三元组数组 tuples。该类继承自 AbstractDocument 抽象类，具体子类实例存储某个文档的信息及其中的三元组列表。

主要方法如下：

addTuple 方法：向 tuples 中添加一个三元组。

getTuple 方法：根据下标获取某个三元组。

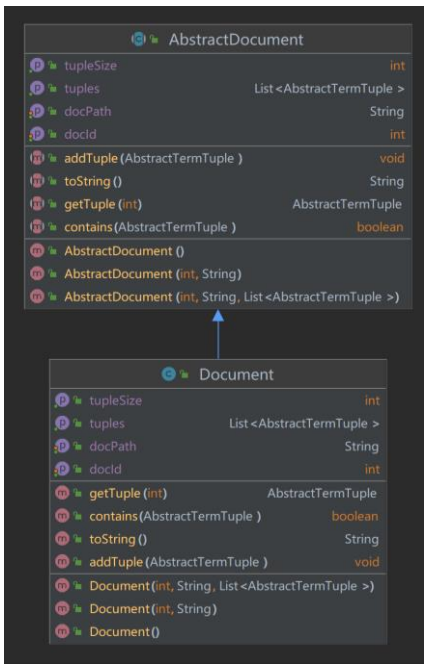

图 6　Document 类 UML 图

⑥ DocumentBuilder

DocumentBuilder 类继承自 AbstractDocumentBuilder 抽象类，具体子类实例用于根据文件或者三元组流构建 Document 对象。

主要方法如下：

build (int docId, String docPath, AbstractTermTupleStream termTupleStream)方法：

参数：文档的编号 docId，文档的路径 docPath，三元组流 termTupleStream

返回值：构建好的 Document 对象

说明：根据 TermTupleStream 逐个读取三元组并存储至 Document 对象中并返回 Document 对象。

build(int docId, String docPath, File file)方法：

参数：文档的编号 docId，文档的路径 docPath，指向某文件的 File 对象 file

返回值：构建好的 Document 对象

说明：根据 TermTupleStream 逐个读取三元组并存储至 Document 对象中并返回 Document 对象，其中根据 File 参数、TermTupleScanner、三个过滤器使用装饰者模式包装一个经过过滤的三元组流，并调用 build (int docId, String docPath, AbstractTermTupleStream termTupleStream)方法构建 Document 对象。

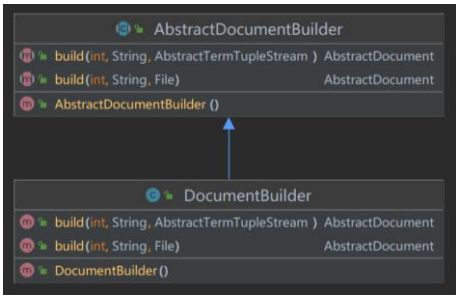

图 7　DocumentBuilder 类 UML 图

⑦ Index

Index 类包含两个数据成员——文档编号 docId 到文档路径 docPath 的映射、单词 Term 到位置信息列表 PostingList 的映射。该类继承自 AbstractIndex 抽象类，具体子类实例存储指定目录下所有文件的索引。

主要方法如下：

addDocument 方法：向索引中添加 Document 中的信息并更新两个映射表，其流程图如图 8 所示：

参数：待添加的 AbstractDocument 对象 document

返回值：无返回值

说明：该方法先根据 document 的 docId 和 docPath 更新 docIdToDocPathMapping 哈希表；然后根据 document 中的三元组列表构建一个单词 term 到该单词在文档出现信息 Posting 的映射表；最后对于该 termPostingMap 的每一对键值，向索引中更新相关信息。

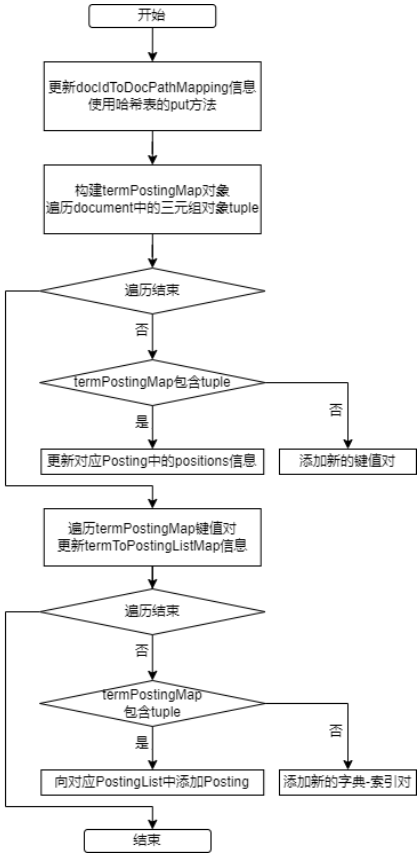

图 8　addDocument 流程图

readObject 方法：向 index.dat 文件中序列化对象。

writeObject 方法：从 index.dat 文件中反序列化更新 index 信息。

load 方法：从文件 file 中反序列化更新 index 对象，其中直接调用 readObject 实现。

save 方法：向文件 file 中序列化 index 对象，其中直接调用 writeObject 实现。

search 方法：根据参数 term 返回其对应的 PostingList 表。

getDictionary 方法：返回存储的字典，直接调用 termToPostingListMapping 的 keySet()。

getDocName 方法：根据 docId 返回文档的路径。

optimize 方法：排序每个 term 的 PostingList 并排序 PostingList 中的每个 Posting，以优化查询效率。

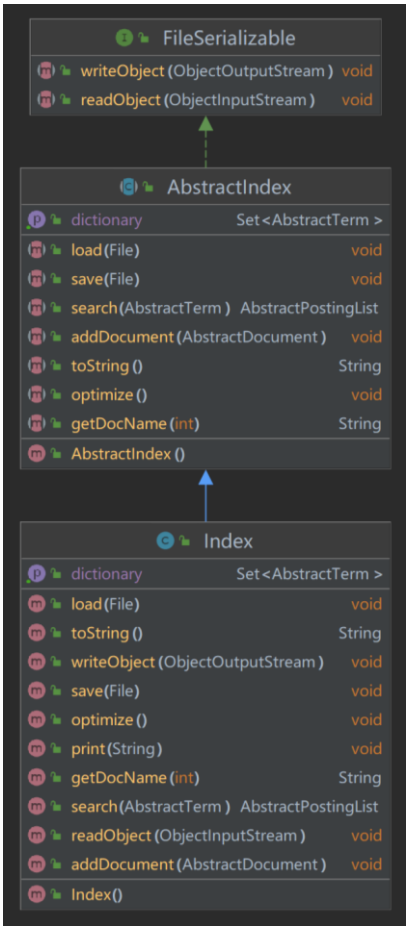

图 9　Index 类 UML 图

⑧ IndexBuilder

IndexBuilde 包含两个数据成员——用于构建 index 的 AbstractDocumentBuilder 对象 docBuilder，docId 计数器。该类继承自 AbstractIndexBuilder 抽象类，具体子类实例用于构建 Index 对象。

主要方法如下：

buildIndex 方法：遍历指定目录下的文件构建索引：

参数：指定目录 rootDirectory

返回值：构建好的 index 对象

说明：该方法遍历指定目录下的文件构建 document 对象并向索引中添加 document，添加完成后序列化 index 对象至指定路径中。

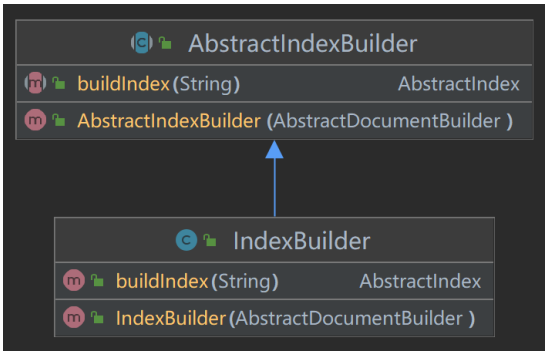

图 10　IndexBuilder 类 UML 图

(2) 文件解析（parse 包中的类）

① TermTupleScanner

TermTupleScanner 类包含两个数据成员——作为输入流对象的 BufferedReader 类的实例 input，存储从文件中读取的三元组的列表 tuples。该类继承自 AbstractTermTupleScanner 抽象类（又继承至 AbstractTermTupleStram 抽象类），具体子类实例用于获取文档中的三元组流。

主要方法如下：

构造方法 TermTupleScanner(BufferedReader input)：

参数：输入流对象的 BufferedReader 类的实例 input

说明：该构造方法使用输入流对象 input，从文本中读取一行数据，使用 StringSplitter 划分单词并为每个单词构建 tuple 对象存入 tuples 数组中（其中要将字母转换为小写）。

next 方法：

说明：当 tuples 不为空时返回 tuples 的第一个三元组并删除之，当 tuples 为空时返回 null。

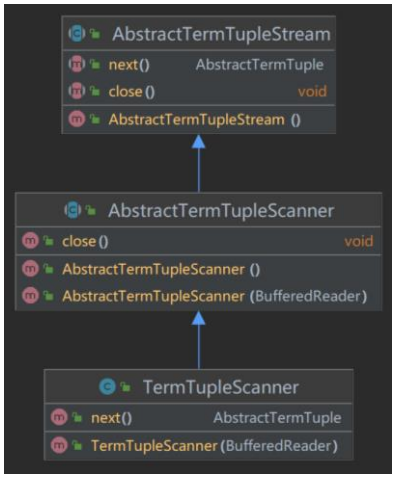

图 11　TermTupleScanner 类 UML 图

② LengthTermTupleFilter

LengthTermTupleFilter 类包含两个数据成员——作为输入流对象的 AbstractTermTupleStream 类的实例 input，存储从文件中读取的三元组的列表 tuples。该类继承自 AbstractTermTupleFilter 抽象类（又继承至 AbstractTermTupleStram 抽象类），具体子类实例用于过滤掉三元组流 input 中不满足长度要求的三元组。

主要方法如下：

next 方法：

说明：当三元组流 input 的 next 方法的返回值不为空时比较三元组中的单词内容的长度是否符合要求，若不满足要求则继续调用 next 方法，否则返回当前 termTuple，当 next 返回值为空时则返回 null。

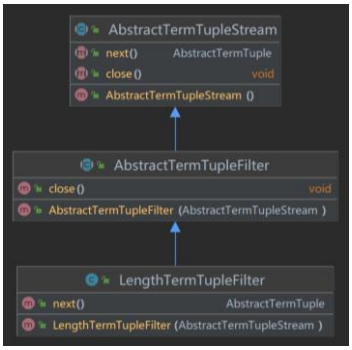

图 12　LengthTermTupleFilter 类 UML 图

③ PatternTermTupleFilter

PatternTermTupleFilter 类包含两个数据成员——作为输入流对象的 AbstractTermTupleStream 类的实例 input，存储从文件中读取的三元组的列表 tuples。该类继承自 AbstractTermTupleFilter 抽象类（又继承至 AbstractTermTupleStram 抽象类），具体子类实例用于过滤掉三元组流 input 中不满足正则表达式要求（只有英文单词满足要求）的三元组。

主要方法如下：

next 方法：

说明：当三元组流 input 的 next 方法的返回值不为空时比较三元组中的单词内容是否满足正则表达式，若不满足则继续调用 next 方法，否则返回当前 termTuple，当 next 返回值为空时则返回 null。

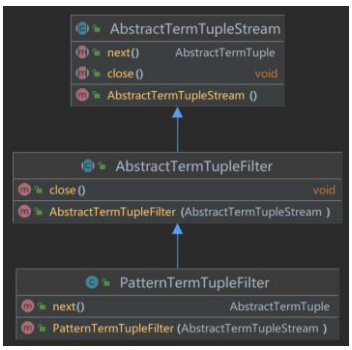

图 13　PatternTermTupleFilter 类 UML 图

④ StopWordTermTupleFilter

StopWordTermTupleFilter 类包含两个数据成员——作为输入流对象的 AbstractTermTupleStream 类的实例 input，存储从文件中读取的三元组的列表 tuples。该类继承自 AbstractTermTupleFilter 抽象类（又继承至 AbstractTermTupleStram 抽象类），具体子类实例用于过滤掉三元组流 input 中在停用词表中的三元组。

主要方法如下：

next 方法：

说明：当三元组流 input 的 next 方法的返回值不为空时比较三元组中的单词内容是否在停用词表，若不满足则继续调用 next 方法，否则返回当前 termTuple，当 next 返回值为空时则返回 null。

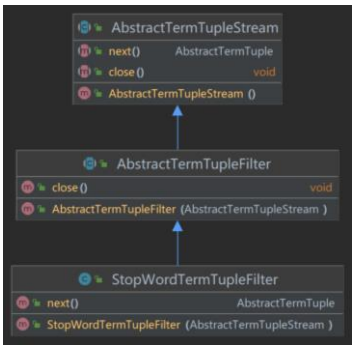

图 14　StopWordTermTupleFilter 类 UML 图

(3) 单词查询（query 包中的类）

① Hit

Hit 类包含 5 个数据成员——文档编号 docId、文档路径 docPath、文档内容 content、命中单词和对应的 Posting 键值对 termPostingMapping、命中文档的得分 score。该类继承自 AbstractHit 抽象类，具体子类实例用于存储命中结果的相关信息。

主要方法如下：

compareTo 方法：

说明：根据 score 来比较两个命中结果，score 越大、文档得分越高、Hit 越大。

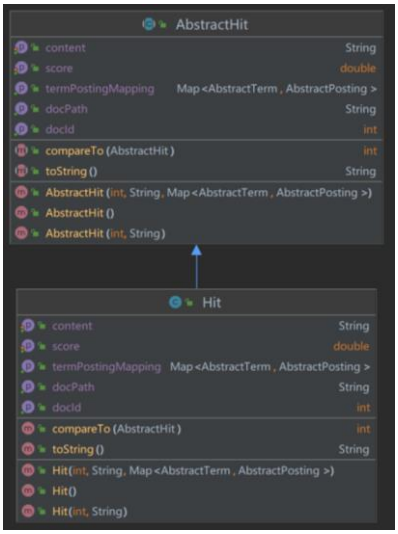

图 15　Hit 类 UML 图

② IndexSearcher

IndexSearcher 类包含两个数据成员——内存中的索引 index，多个检索词的逻辑组合枚举类型 LogicalCombination。该类继承自 AbstractIndexSearcher 抽象类，具体子类实例用于查询关键词并返回查询结果。

主要方法如下：

open 方法：

说明：从 indexFile 对应的文件中反序列化加载 index 对象。

search(AbstractTerm queryTerm, Sort sorter)方法：

参数：检索词 queryTerm、排序对象 sorter

返回值：命中结果数组

说明：该方法中，先根据 queryTerm 得到其在索引 index 中的 postingList，如果不为 null 则将根据其大小构建 Hit 数组，并为每个 posting 构建 Hit 对象，最后使用 sorter 对 Hit 列表进行排序，并返回 Hit 列表的数组形式拷贝，否则返回 null。

search(AbstractTerm queryTerm1, AbstractTerm queryTerm2, Sort sorter, LogicalCombination logicalCombination)方法：

参数：检索词一 queryTerm1、检索词二 queryTerm2、排序对象 sorter、检索词逻辑组合 logicalCombination

返回值：命中结果数组

说明：该方法中，先根据 queryTerm1 和 queryTerm2 分别得到其在索引 index 中的 postingList1 和 postingList2；然后根据 logicalCombination 的内容分别处理。如果逻辑组合词为 null，则为查询两个连续关键词，应取两个关键词查询结果的交集，并比较两个单词在同一个文档中的位置是否相邻；如果逻辑组合词为 OR，则取两个关键词查询结果的并集；如果逻辑组合词为 AND，则取两个关键词查询结果的交集。最后使用 sorter 对 Hit 列表进行排序，并返回 Hit 列表的数组形式拷贝。

该方法的流程图如图 16 所示：

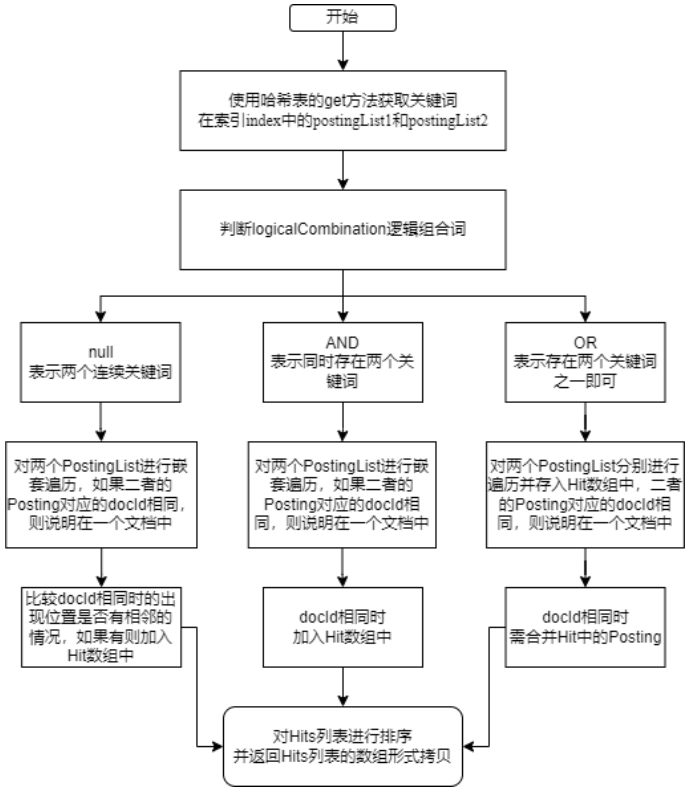

图 16　两个检索词的 search 方法流程图

printQuery 方法（添加的自定义方法）：将用户的查询记录与结果存入到日志文件 query.txt 中便于用户查询。

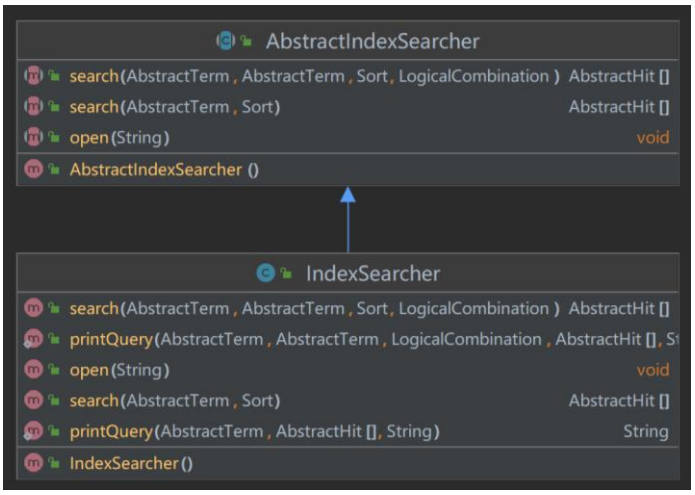

图 17　IndexSearcher 类 UML 图

③ SimpleSorter

SimpleSorter 类实现了 Sort 接口，具体子类实例用于对 Hit 列表进行排序或者得到某个命中结果 Hit 的得分。

主要方法如下：

sort 方法：

参数：待排序的 Hit 列表 hits

说明：使用 List 类的排序方法依据 score 对 hits 进行降序排序。

score 方法：

参数：待计算分数的命中结果 hit

说明：该方法根据 hit 中的 termPostingMap，通过简单相加每个 posting 的 freq 计算得分。

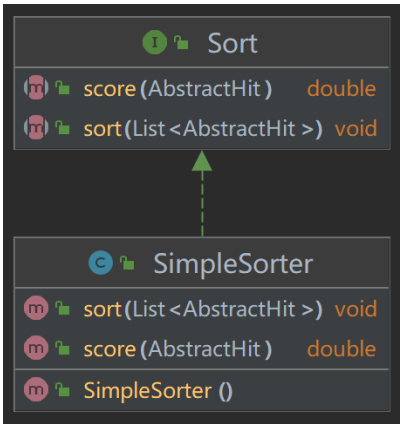

图 18　Sort 类 UML 图

三、软件开发

本项目在 **Windows11** 下的 IntelliJ **IDEA** Ultimate Edition 2022.1 中进行项目构建、运行和

调试，使用的 jdk 版本为 openjdk version **13.0.11**。

四、软件测试

1. 自动测试脚本运行结果

```
================================================
All Test Suite
Total tests run: 106, Failures: 0, Skips: 0
================================================
```

图 19　自动测试脚本运行结果图

2. Windows 批处理命令运行结果

① 主界面运行图

```
************************************************
**           Welcome To Search Engine         **
************************************************
** 0. Exit The Engine                         **
** 1. Build Index From Existing Text Files    **
** 2. Build Index From Existing Index File    **
** 3. Search Certain Term(s)                  **
** 4. Save Index into text file and display   **
************************************************
Please Input Your Choice:
```

图 20　主界面运行图

② 打印构建索引结果图

```
]
docId-----docPath mapping:
0   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\1.txt
1   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\10.txt
2   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\11.txt
3   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\12.txt
4   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\13.txt
5   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\14.txt
6   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\15.txt
7   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\2.txt
8   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\3.txt
9   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\4.txt
10   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\5.txt
11   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\6.txt
12   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\7.txt
13   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\8.txt
14   ---->   D:\HUST-Courses\7_Java_Experiment\SearchEngineForStudent\text\9.txt
term-----postingList mapping:
accord   ---->   {"docId":2,"freq":1,"positions":[30]}
according   ---->   {"docId":0,"freq":2,"positions":[22, 104]}->{"docId":5,"freq":1,"positions":[73]}-
":1,"positions":[61]}->{"docId":12,"freq":1,"positions":[27]}
action   ---->   {"docId":5,"freq":1,"positions":[28]}
activity   ---->   {"docId":7,"freq":1,"positions":[1]}
aged   ---->   {"docId":0,"freq":1,"positions":[88]}
agree   ---->   {"docId":8,"freq":1,"positions":[19]}
agreed   ---->   {"docId":2,"freq":1,"positions":[71]}->{"docId":4,"freq":1,"positions":[112]}
agreement   ---->   {"docId":2,"freq":4,"positions":[17, 32, 54, 68]}
alleviate   ---->   {"docId":11,"freq":1,"positions":[54]}
announced   ---->   {"docId":10,"freq":1,"positions":[2]}
announcement   ---->   {"docId":6,"freq":1,"positions":[49]}
```

图 21　打印构建索引结果图

③ 查询一个关键词结果图

图 22　查询一个关键词结果图

④ 查询两个关键词（AND）时输入单词不满足要求的结果图

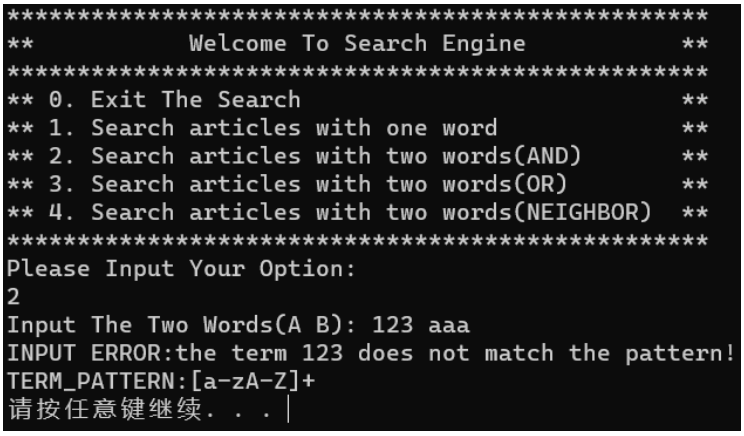

图 23　查询两个关键词（AND）结果图

⑤ 查询两个关键词（OR）结果图

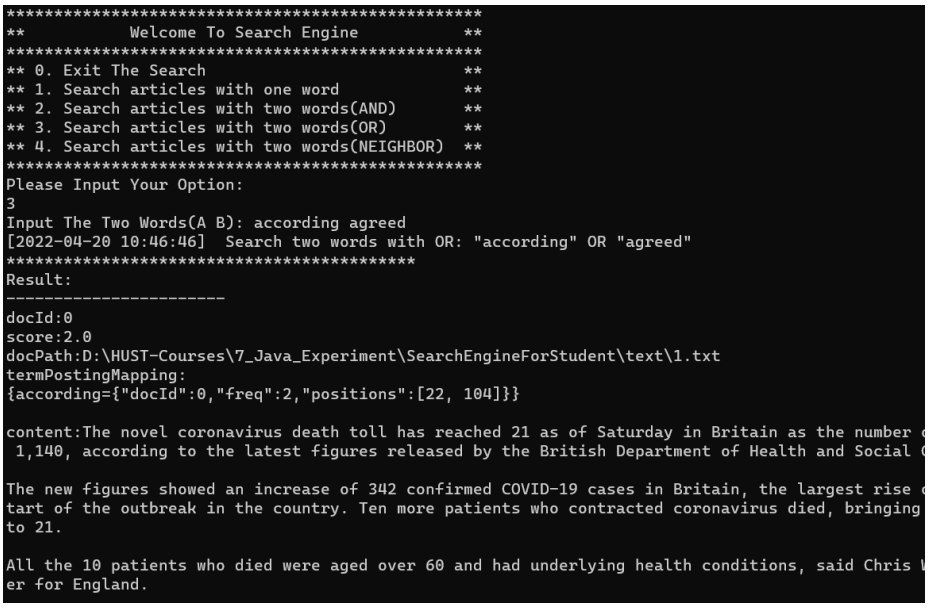

图 24　查询两个关键词（OR）结果图

⑥ 查询两个连续关键词结果图

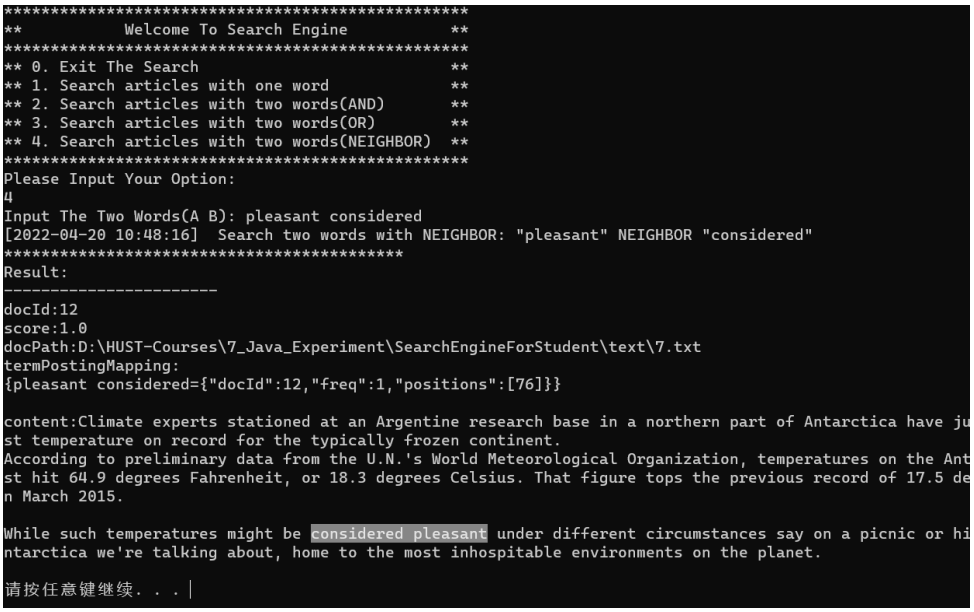

图 25　查询两个连续关键词结果图

⑦ index 索引输出的文本文件图

图 26　index 索引输出的文本文件图

⑧ query 查询输出的文本文件图

图 27　query 查询输出的文本文件图

五、特点与不足

1. 技术特点

本项目较有特色之处主要如下：

（1） 实现查询两个连续单词的进阶功能。

（2） 利用 Java API 实现相关功能。对于 ArrayList，通过使用 sort 方法和 Comparator 实现排序，使用 contains 方法进行去重操作；对于 HashMap，通过使用 keySet 方法返回字典序列，使用 foreach 方法快捷遍历哈希表，使用 get 方法返回某个 key 对应的 value。

（3） 通过复用方法使代码精简易懂。在上层数据结构中通过复用下层数据结构的 toString 方法、反序列化和序列化方法、sort 方法避免了代码的冗余与重复。

（4） 为用户提供了较为良好的控制台操作体验，每次完成用户选择的操作后，控制台将会自动清屏，避免了用户之前操作对控制台页面的占用，提高交互体验。具体界面见测试结果部分。

（5） 用户可以选择将构建的索引输出到文本文件中并打印查看，同时程序会将用户的查询操作和结果自动输出至 query.txt 文件中，便于用户查看。

2. 不足和改进的建议

（1）该项目主要在控制台中运行，未使用图形化界面，查询结果等信息不便于查看，后期可以尝试通过图形化界面高亮显示文件中的命中关键词，提高用户交互体验。

六、过程和体会

1. 遇到的主要问题和解决方法

（1） AbstractTermTupleStream 的子类 TermTupleScanner 的实现

在实现 TermTupleScanner 时，由于对于装饰者模式不理解，误以为该类就要对文档中的三元组进行过滤，在这里浪费了一些时间，在仔细理解装饰者模式后，成功实现了该类的相关方法。

（2） IndexSearcher 的实现

在两个关键词的查询中，对于不同的逻辑组合词，需要完成不同的功能，由于在刚开始写代码时未搞懂 termPostingMap 哈希表，导致在对查询结果的合并或筛选时出现问题，在理清数据结构的含义后，慢慢试错，最终成功实现了该方法。

（3） toString 方法的复用

由于一些类的 toString 方法未复用或者一些数据未包含在其中，导致在测试时总会出现问题，在询问老师和同学后解决了问题。

2. 课程设计的体会

本次实验最大的体会就是对于工程的模块化设计有了一些理解。该工程中主要利用已经定义好的抽象类实现其具体子类来完成相关功能，并分为四个包——index、parse、search、run，代码层次逻辑清晰，让我对模块化设计有了初步了解。同时在完成 parse 包中的相关类时，对

装饰者模式有了更深刻的理解。

七、源码和说明

1. 文件清单及其功能说明

（1） Experiment1Test 为自动测试文件所在文件夹；

（2） SearchEngineForStudent 为项目工程所在文件夹，；

（3） CS2008-U202015533-徐瑞达.docx 为实验报告。

2. 用户使用说明书

（1） 将 Experiment1Test 文件夹中 test.bat 文件内的 JDK 路径替换为测试机器上的路径后，在控制台中运行该脚本，即可查看自动测试结果；

（2） 将项目工程中 run.bat 文件内的 JDK 路径替换为测试机器上的路径后，将需检索的文档移至项目工程的 text 目录下，在控制台中运行该脚本，即可启动搜索引擎以测试。

3. 源代码

具体源码可见项目工程中的 src 目录。