

**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 ： ACM1901**

**学 号 ： U201915035**

**姓 名 ： 邹雅**

**指导教师 ： 辜希武**

**2021 年 5 月 10 日**

1. **需求分析**
2. **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。**

1. **需求分析**

本实验要求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，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。其中为各个数据结构类编写toString方法时，注意设计输出的格式以及复用情况。功能完成后在程序中编写测试类初步测试程序功能的正确性，对不同的功能、测试数据进行测试。最后使用测试脚本进行更加完备的测试。

注意程序中会发生异常的情况，特别是空指针异常，并通过思考完备性来进行避免。

1. **系统设计**
2. **概要设计**

根据实验文档以及所要实现的具体类，将整个设计分为三个模块：倒排索引构建模块、文档输出流及过滤模块、搜索及命中构建模块，每个模块中类如下图所示：

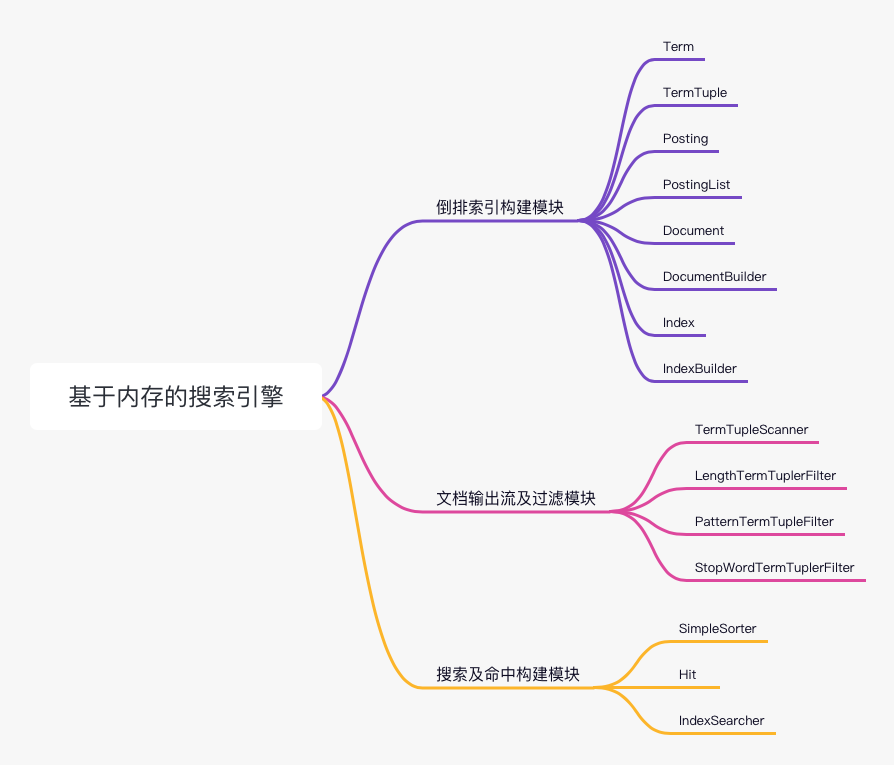


图2.1.1 模块构建导图

倒排索引构建模块负责从文档输出流中构建倒排索引。Term是表示词汇的类；TermTuple表示词汇三元组；Posting表示一个词汇在某一个文档中的信息；PostingList是由Posting组成的list，表示某一个词汇在所有文档中存在的信息；Document是对文件的抽象，保存着该文件所有信息；DocumentBuilder用于从文档的词汇流中构建Document对象；Index是核心，建立并保存倒排索引结构，便于后续搜索查询；IndexBuilder用于从某个目录下的所有文件中构建出Index对象。

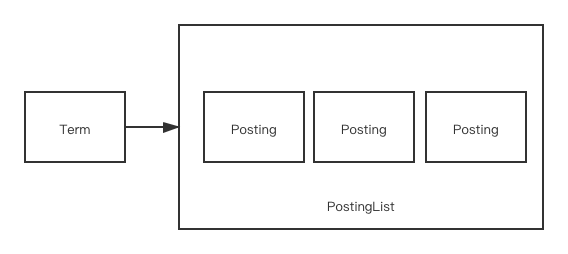


图2.1.2 核心倒排索引结构

文档输出流及过滤模块负责从文本文件中分出词汇并构建词汇三元组输出流，构建过滤器对长度过长或过短的词汇、停用词、不符合字母模式的单词进行过滤。TermTupleScanner负责从文本文件中分出词汇并构建词汇三元组输出流，LengthTermTupleFilter负责过滤长度过长或过短的词汇，PatternTermTupleFilter负责过滤不符合字母模式的单词，StopWordsTermTupleFilter负责过滤停用词。

倒排索引构建流程如下所示：

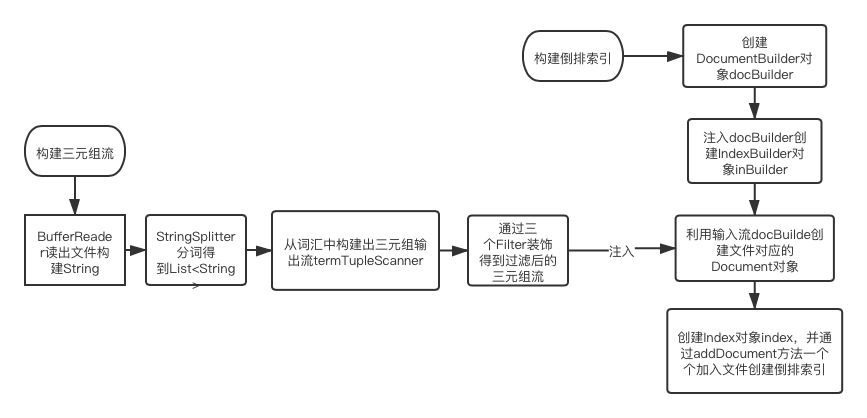


图2.1.3 构建倒排索引流程

搜索及命中构建模块负责从构建好的倒排索引中找出所查找的词并构建命中结果的类对象。SimpleSorter负责对计算命中结果的得分以及根据得分对命中结果排序。Hit是命中结果的数据结构类，保存某个文件的命中结果相关信息。IndexSearcher负责从制定索引文件中加载倒排索引，并对单个或两个检索词的检索。

检索流程如下所示：

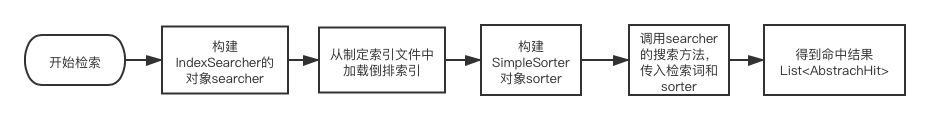


图2.1.4 检索流程图

1. **详细设计**

1. 倒排索引构建模块中的类的具体实现。其实现的UML图如下所示：

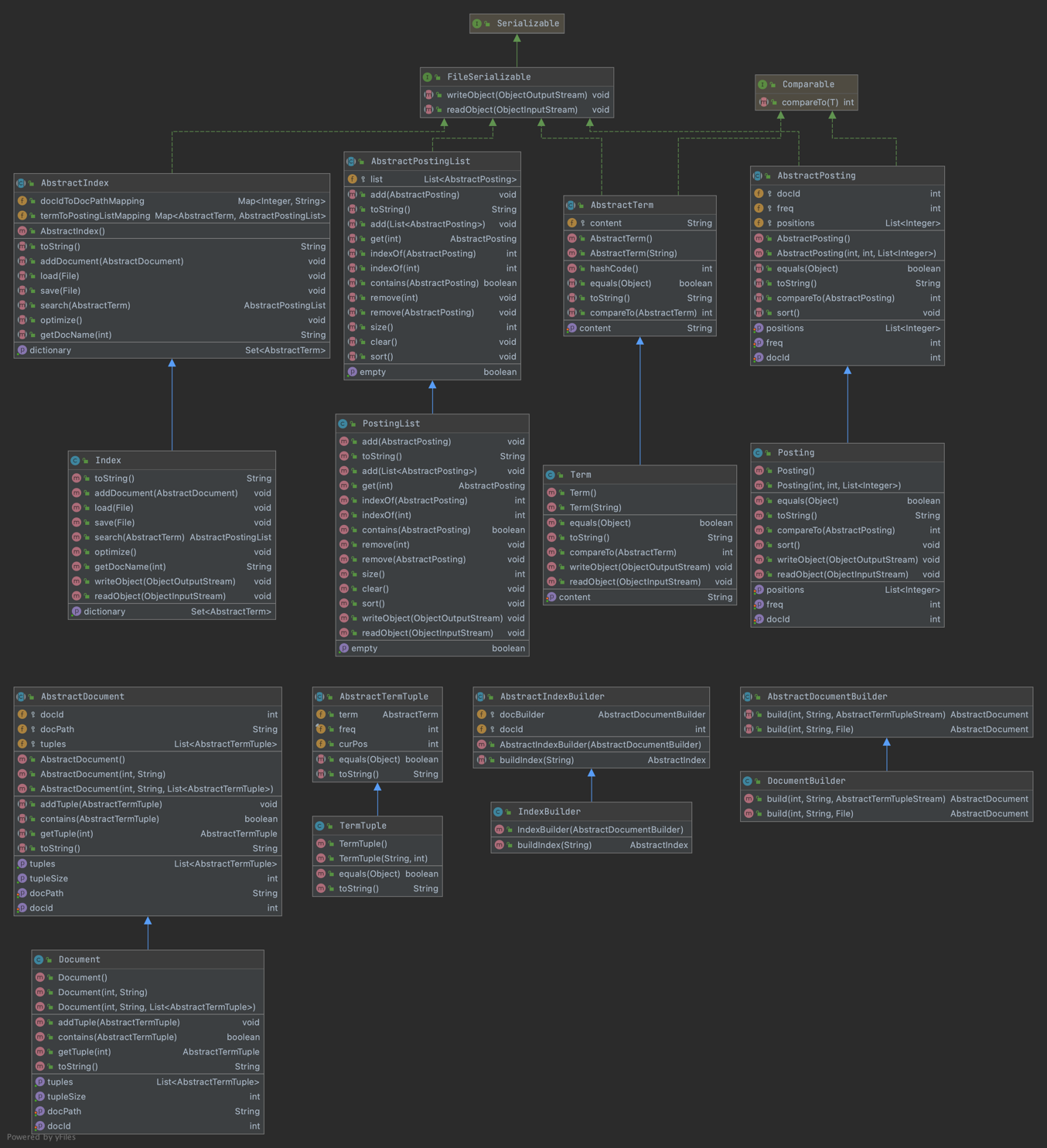


图4.2.1 倒排索引构建模块中的类的UML图

（1）**Term**：对单词的封装。有一个String类型的数据成员content，保存单词的内容。

**（2）TermTuple**：对单词在文档中存在情况的封装。有三个数据成员，term代表单词，freq代表单词出现的频率，定义成常量1，curPos代表单词出现的当前位置。

**（3）Posting**：用于存储某一个单词在某一个文档中的信息。

有三个数据成员，docId表示单词存在的当前文档id，freq表示单词在该文档中出现的次数，positions表示单词每次出现的位置。类中实现了sort方法，用于将positions进行排序。

**（4）PostingList**：由Posting组成的list，表示某一个词汇在所有文档中存在的信息。

在这个类里面实现了add方法，可以一次加入一个posting也可以一次加入多个posting； get方法，通过下标获取对应的posting；indexOf方法，可以由传入的Posting对象或由docId检索获取；remove方法，可以由传入的Posting对象或由docId检索删除；还有size、contains、isEmpty、clear和sort方法，其中sort是根据docId大小对Posting对象进行排序;

**（5）Document**：对文本文件的抽象，保存着该文件所有信息。

有三个数据成员，docId表示文档id，docPath表示文档的绝对路径，tuples保存着文档对应的单词三元组对象，使用ArrayList进行保存。类中实现的方法除去getter和setter，有addTuple方法，通过传入的AbstractTermTuple类型的对象加入文档，且排除重复的情况；contains方法，判断是否包含制定的三元组对象；getTupleSize方法，返回文档中三元组对象的个数；getTuple方法，通过传入的index下标获取对应的三元组。

**（6）DocumentBuilder**：用于从文档的词汇流中构建Document对象。

没有数据成员，只有两个构造**Document**对象的重载方法build。一个是由解析文本文件得到的三元组流TermTupleStream对象来构建**Document**，实现很简单，一个个读取三元组调用**Document**对象的addTuple方法加入其中即可。另一个build方法是由给定的File对象来构建，实现也很简单，利用File对象封装得到一个BufferedReader对象并传入构建的TermTupleScanner对象，即可构建该File对象对应的文本文件的三元组流；但构建好之后还要利用三个filter类对该三元组层层过滤，得到我们真正想要的词汇三元组流，再传入第一个build方法进行构建Document对象。

对三元组流过滤采用装饰者模式的设计思想，所有的三元组流的声明类型都是AbstractTermTupleStream，过滤类构造传入的对象类型也是AbstractTermTupleStream，即装饰者和被装饰者有相同的超类型。既然装饰者和被装饰者有相同的超类型，在任何需要被装饰者（原对象）的场合，可以用装饰过的对象来代替它。所以在经过过滤器后的三元组流在某种程度上与原三元组是等价的，我们只不过是改变了它的next方法所得到的对象。我们用LengthTermTupleFilter、PatternTermTupleFilter、StopWordTermTupleFilter三个类来装饰原TermTupleScanner类型的对象，得到长度满足要求、符合字母模式且不是停用词的单词三元组流。

**（7）Index**：核心类，建立并保存倒排索引结构，便于后续搜索查询。

有两个数据成员，都是用TreeMap类型保存的，可见两个数据成员都是mapping映射。docIdToDocPathMapping是文档id到文档绝对路径的映射，数据类型为Map<Interger,String>。termToPostingListMapping是单词到单词在文档中的信息PostingList的映射，数据类型为Map<AbstractTerm,AbstractPostingList>。

Index类中核心方法为addDocument方法，其余不在此赘述。addDocument传入一个AbstractDocument类型的对象，并读取该文档对应的信息来构建更新倒排索引。首先把该文档的id信息及path绝对路径加入docIdToDocPathMapping中；对于三元组list，其处理方式需要分情况来讨论。依次读取文档中的三元组，如果该词汇不在termToPostingListMapping中，则构建一个新的映射加入；如果词汇在该数据成员中，则搜寻在该词汇对应的PostingList对象中是否存在当前文件的Posting对象，如果不存在则构建Posting对象加入PostingList对象中，若存在只需将该三元组中的位置信息加入该Posting对象并更新频率信息即可。

**（8）Indexbuilder**：用于从某个目录下的所有文件中构建出Index对象。

有两个数据成员。docBuilder是初始化时必须传入的对象（该类没有缺省构造函数），用于构建索引时建立文档Document对象，docId用于对加入的文档计数。只有一个方法buildIndex，该方法传入一个String类型的根目录，方法负责把该目录下的文本文件全部解析构建成倒排索引，即构建Index对象。实现首先用FileUtil中的静态方法list将该根目录下所有文件名返回，构建一个Index对象index作为最终所需要的倒排索引结构对象，在遍历文件名利用docBuilder创建文本文件对应的Document对象，然后把该对象传入index的addDocument方法，即把文本文件加入倒排索引index中，最后返回这个index即可。

2. 文档输出流及过滤模块中的类的具体实现。其实现的UML图如下所示：

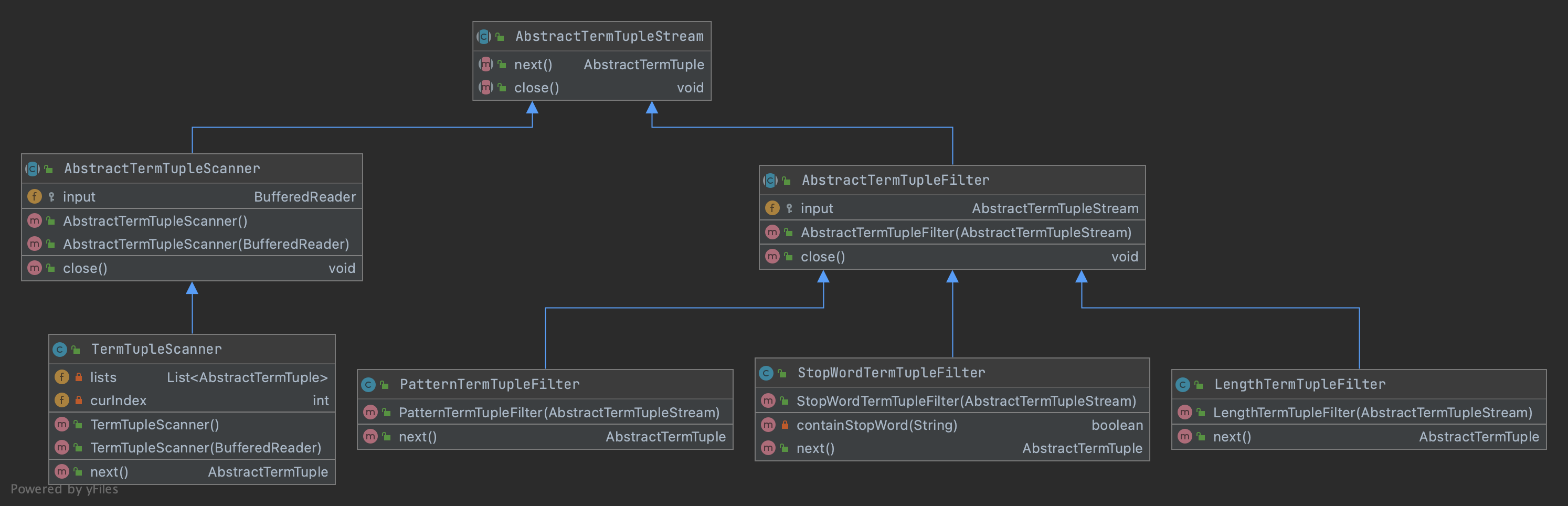


图4.2.2 文档输出流及过滤模块中的类的UML图

(1)**TermTupleScanner**：负责从文本文件中分出词汇并构建词汇三元组输出流.

有两个数据成员：curIndex用于统计当前单词位置；lists用于存放从文本中解析出来的词汇三元组列表，数据类型是List<AbstractTermTuple>。在含参构造函数中，传入一个BufferedReader类型的对象input，是对文本文件的一个抽象，实现得到lists需要不断调用input.readLine()，将读取到的所有字符连接在一起，然后创建一个StringSplitter对象调用splitByRegex方法来将这个字符串分割成一个个词汇保存，再遍历将词汇list转化为三元组list即可。有一个实例方法next()，就是通过curIndex的值得到当前的三元组返回。

三个具体的Filer都是通过对AbstractTermTupleStream不断调用next方法来获取单词，进而将符合要求的返回，不符合要求的则直接跳过。

**(2)StopWordTermTupleFilter**：通过StopWords类中的STOP\_WORDS数组来判断是否是停用词，如果是则递归调用寻找下一个词汇是不是停用词，不是则保留。

**(3)PatternTermTupleFilter**：通过Config中的TERM\_FILTER\_PATTERN进行正则匹配，符合则保留，不符合则递归调用寻找下一个词汇是否满足。

**(4)LengthTermTupleFilter**：通过Config中规定的TERM\_FILTER\_MAXLENGTH以及TERM\_FILTER\_MINLENGTH来判断单词是否在规定的长度范围之内，跳过不满足要求的单词，递归查找。

3. 搜索及命中构建模块中的类的具体实现。其实现的UML图如下所示：

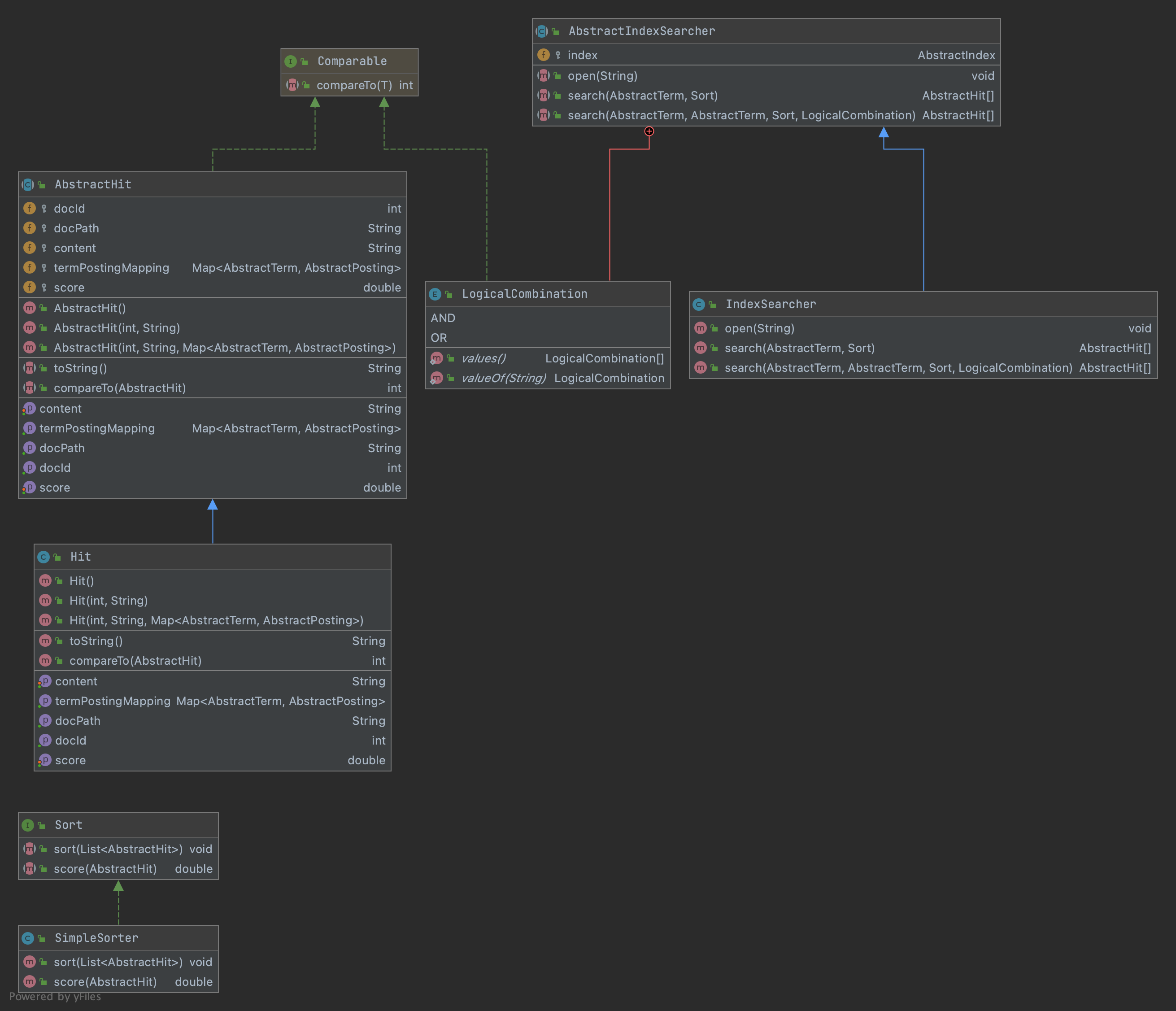


图4.2.3 搜索及命中构建模块中的类的UML图

1. **Hit**：保存查询的一个结果，数据成员有docId、docPath、content、termPostingMapping、score，其中的content要保存对应的文档中的全部内容，termPostingMapping中保存查询到的命中的单词和对应的Posting键值对，score保存当前查询结果的分数。
2. **IndexSearcher**：根据构建好的索引搜索词条，数据成员index用于保存要用的索引。有三个实例方法：open方法将索引文件加载到index中；search方法有两个，分别实现对一个词和对两个词的检索。对于一个词的检索，遍历index，找到词汇对应对PostingList对象，针对每一个Posting建立Hit对象并保存，利用传入SimpleSorter对象对每一个检索建立的Hit对象求出它相应的分数，再对Hit数组排序后返回最终结果；对于两个词的搜索，对两个词分别进行调用一个词的检索方法后并保存相应的结果，针对结合规则，对搜索结果进行合并，利用传入SimpleSorter对象排序后返回Hit数组。
3. **SimpleSorter**：score方法计算命中文档的得分，根据词条出现的频率进行赋值，返回得分的负值；sort方法对传入的命中结果集合根据文档得分排序。

注：所有必要的基本方法都进行了实现，如toString，writeObject, readObject, equals，compareTo方法以及必要的构造函数。

1. **软件开发**

* 操作系统:macOS BigSur 11.3.1
* IDE及配置:Intellij IDEA JDK13
* 测试平台:Intellij IDEA / iTerm

1. **软件测试**

运行自动测试脚本，得到结果All Test Suite，如下图所示：

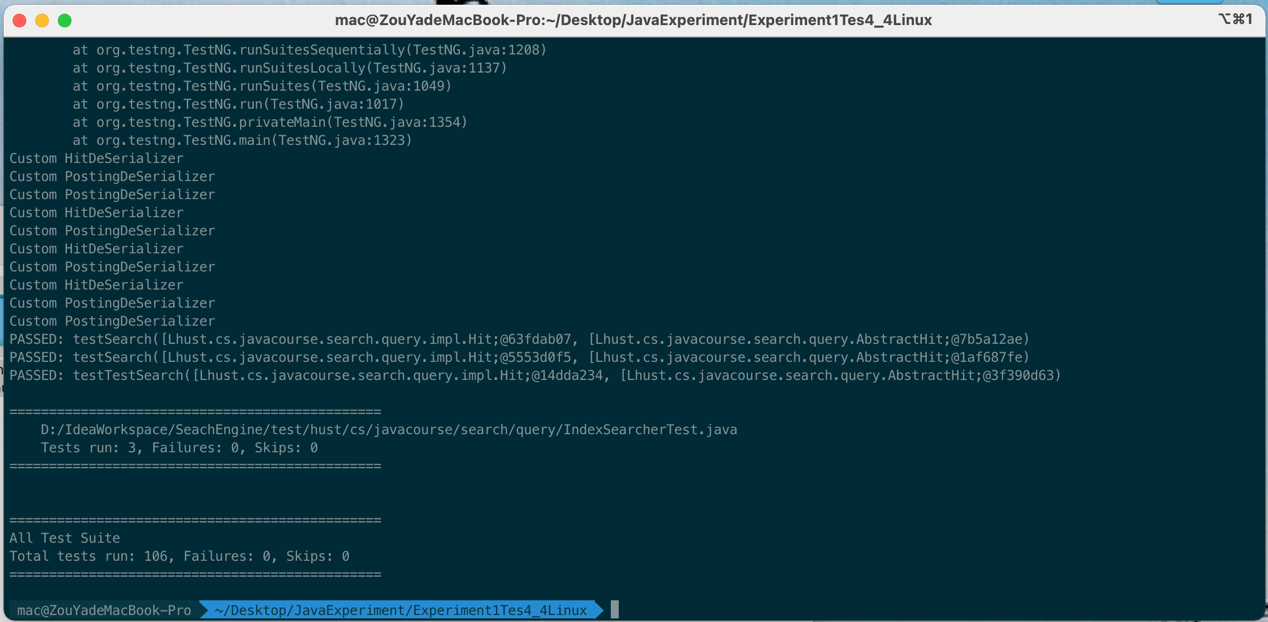


图4.1 自动测试脚本运行结果图

构建好的索引和基于统一的搜索测试案例的检索结果文本文件输出结果如下两张图所示：

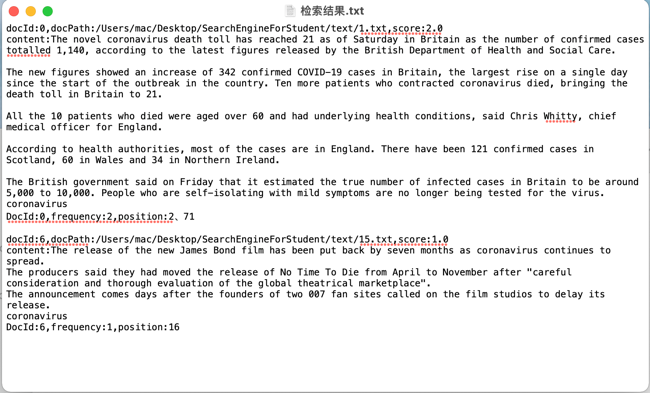
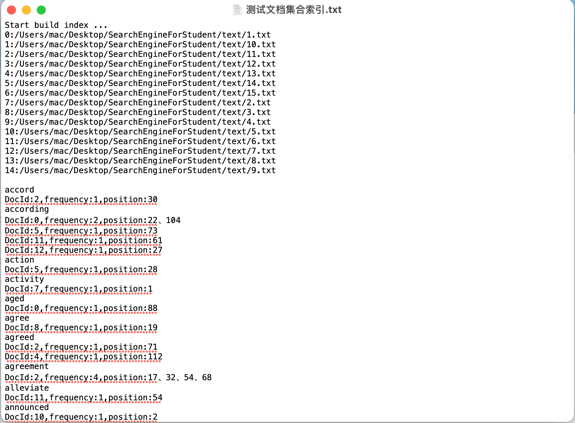


图4.2、图4.3文本输出结果截图

另：Java API文档、构建好的索引和基于统一的搜索测试案例的检索结果文本文件见实验附件。

1. **特点与不足**
2. **技术特点**

这次实验是在给定的框架之下实现指定的功能，比较有特色的主要有以下几点:

1.数据类的序列化以及反序列化功能。序列化机制允许将实现序列化的Java对象转换位字节序列，这些字节序列可以保存在磁盘上，或通过网络传输，以达到目的主机的目标进程，以后通过反序列化恢复成原来的对象。序列化机制使得对象可以脱离程序的运行而独立存在。序列化和反序列化不需要我们去关心文件的存储格式，特别是反序列化时不需要管对象的每个属性内容保存的格式。在序列化过程中充分利用了成员实现的序列化反序列化方法，与在toString方法中调用成员的toString方法有异曲同工之妙。

2.设计模式中的装饰者模式。如前所述，对三元组流过滤采用装饰者模式的设计思想，我们用LengthTermTupleFilter、PatternTermTupleFilter、StopWordTermTupleFilter三个类来装饰原TermTupleScanner类型的对象，得到长度满足要求、符合字母模式且不是停用词的单词三元组流。装饰者模式的好处是可以用装饰后的对象来直接替代原来的对象，因为他们都有相同的超类型。

3.对Java的API进行充分利用。如在实现排序的方法中，只要调用Collection.sort(List<T>)方法，传入被排序的list即可。比如在类Index中的getDictionary方法中，需要返回所有保存的Term，可以直接调用map结构的keySet方法返回一个Set对象，包含了所有的Term。可见熟悉Java的API对于学好并熟练使用Java非常重要。

1. **不足和改进的建议**

实验过程中，由于老师设计好的倒排索引结构、相关的抽象类、接口定义还有相关已经实现好的工具类都非常完整且有详细的说明，这让我在实验过程中只是简单地跟着这些思路去实现具体的方法。个人思考的空间比较小，当把在使用的流程中各个类的方法串联起来想清楚，然后就是去学习具体实现的技术点（比如查API、了解序列化和反序列化等等）。实践还是让我学到很多东西，但在思考设计方面可能应该让我们有更多的空间。

1. **过程和体会**
2. **遇到的主要问题和解决方法**

1.对于具有含参数构造函数的类，继承抽象类后忘记了加入相应的含参数构造函数，导致运行自动脚本测试时出现很多空指针异常。构造函数没办法通过继承得来，没有规定构造函数的类只有一个默认的缺省构造函数。

2.序列化和反序列化。在看完老师给的文档后，对于基本类型的序列化我比较清楚。但遇到如List<T>、Map<T1,T2>这样的数据类型，我就有点犹豫是否要通过遍历来序列化。后来查阅了资料才知道可以直接序列化。

3.忘记了要将输入的文本转化为小写。在Config类中的IGNORE\_CASE为true，应该要忽略大小写，但我在第一遍编程时没有注意到这一点，直到运行自动测试脚本的时候发现不能匹配，仔细看了报错信息才想起来有提到过这一点并在读入文档的时候根据该boolean值来进行大小写转换。

4.对于String类型的数据，下意识就会用==来判断是否相同，但这是错误的，String类型为引用类型，应该调用equals方法进行比较判断。这在我的StopWordTermTupleFilter类的编程中体现出来了，不能成功地过滤停用词。

1. **课程设计的体会**

这是学习Java以来接触到第一个大型设计的实践项目，同时也是我认为在所有课程做过的实验中比较有意义的一个，这和简单的写算法题实验不同。在完成这次实验的过程中，我体会到了从一个较大型项目的功能规划划分，到一个个模块，再到一个个抽象类的设计，最终把项目简化成具体方法实现。这样的设计过程带着我去思考大型项目的搭建和设计。此外，我也对设计模式有了初步的了解，让我体会到设计的精妙对编程的简化和提升。在这个实验中，用到了装饰者模式、策略模式还有迭代器的思想。这引发了我对设计模式的学习兴趣，我想通过对设计模式的学习，对我日后编程应该有很大的帮助。

同时这个实验让我对面向对象编程有了更深刻的体会（之前接触过别的OOP语言，但在Java中才系统地学了面向对象思想），体会到了面向对象的封装、继承、多态性特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护。

在具体编程过程中，我认识到熟悉一门语言的API的重要性，使用好API在我们的编程中能起到举重若轻的作用。比如我如果不知道Collections.sort(List<T>)方法，那么我可能就要多次编写排序的代码，即使不谈性能问题，代码也会因此变得复杂冗长。

总体来说，我对这次Java实验的体验还是比较满意的，这让我从中学到很多。

1. **源码和说明**
2. **文件清单及其功能说明**

提交文件夹内文件如下所示：

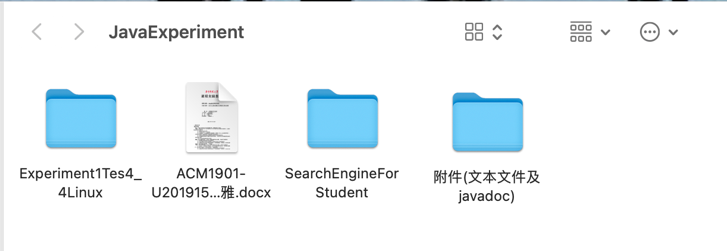


图7.1.1 提交文件夹

其中Experiment1Tes4\_4Linux文件夹为自动测试包，SearchEngineForStudent为工程文件夹，附件文件夹为所要提交的测试文本文件以及文档API的javadoc输出文件。

1. **用户使用说明书**
2. 由于实验以及测试是在macOS平台进行的，工程所生成的.class文件已经放入子目录betest中，所以测试的时候需要运行test.sh，观察自动测试的结果即可。（另说明，macOS运行脚本文件，需要通过vim将test.sh的格式设置成unix，这与Linux平台下运行不同）。
3. 将要检索的文档加到项目工程的text目录下，通过自行编写的测试类进行测试。注意在运行检索测试类之前需要先运行倒排索引构建测试类，因为检索依赖index.bat文件，所以需要先构建倒排索引再能进行检索。
4. **源代码**

见附件中的SearchEngineForStudent文件夹中src子目录。