# 武汉大学国家网络安全学院

# 实验报告

课程名称 信 息 检 索

专业年级 信息安全 2017 级

姓　　名 高智

学　　号 2016301500067

协 作 者 无

实验学期 2020-2021 学年 第二 学期

课堂时数 32 课外时数

填写时间 2021 年 6 月 20 日

## 实验介绍

**【实验名称】：** 基于 Lucene 的文件信息检索的实现

**【实验目的】：**

Lucene 的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。Lucene 是一套用于全文检索和搜寻的开源程式库，由 Apache 软件基金会支持和提供。Lucene 提供了一个简单却强大的应用程式接口，能够做全文索引和搜寻。在 Java 开发环境里 Lucene 是一个成熟的免费开源工具。就其本身而言，Lucene 是当前以及最近几年最受欢迎的免费 Java 信息检索程序库。人们经常提到信息检索程序库，虽然与搜索引擎有关，但不应该将信息检索程序库与搜索引擎相混淆。

普通的数据库搜索的缺陷:

1、因为没有通过高效的索引方式，所以查询的速度在大量数据的情况下是很慢。

2、搜索效果比较差，只能对用户输入的完整关键字首尾位进行模糊匹配。用户搜索的结果误多输入一个字符，可能就导致查询出的结果远离用户的预期。

所以我们在有特定需求的情况下，使用数据库进行搜索已经不够方便了，所以我们要扩展这些功能。

倒排索引又叫反向索引以字或词为文档中出现的位置情况。


在实际的运用中，我们可以对数据库中原始的数据结构（左图），在业务空闲时事先根据左图内容，创建新的倒排索引结构的数据区域（右图）。用户有查询需求时，先访问倒排索引数据区域（右图），得出文档 id 后，通过文档 id 即可快速，准确的通过左图找到具体的文档内容。本次实验通过使用 lucene 实现一个简单的文件信息检索。

**【实验环境】：**

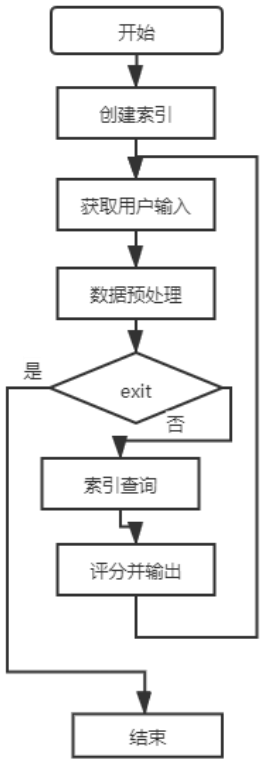
处理器　Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz　1.80 GHz
机带 RAM　8.00 GB (7.89 GB 可用)
系统类型　64 位操作系统，基于 x64 的处理器
Java version 1.8.0_272
Lunece version 4.10.2

Junit 4.12

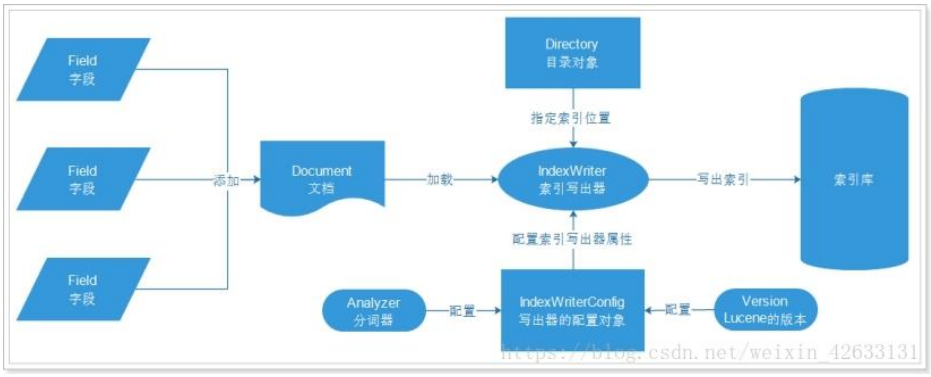
**实验内容**

**【实验方案设计】：**

Lucene 全文检索就是对文档中全部内容进行分词，然后对所有单词建立倒排索引的过程。实验总体流程如下图所示：




（1）创建索引




添加相关依赖：

```xml
<!-- lucene核心库 -->
<dependency>
    <groupId>org.apache.lucene</groupId>
    <artifactId>lucene-core</artifactId>
    <version>${lunece.version}</version>
</dependency>
<!-- Lucene的查询解析器 -->
<dependency>
    <groupId>org.apache.lucene</groupId>
    <artifactId>lucene-queryparser</artifactId>
    <version>${lunece.version}</version>
</dependency>
<!-- lucene的默认分词器库 -->
<dependency>
    <groupId>org.apache.lucene</groupId>
    <artifactId>lucene-analyzers-common</artifactId>
    <version>${lunece.version}</version>
</dependency>
```

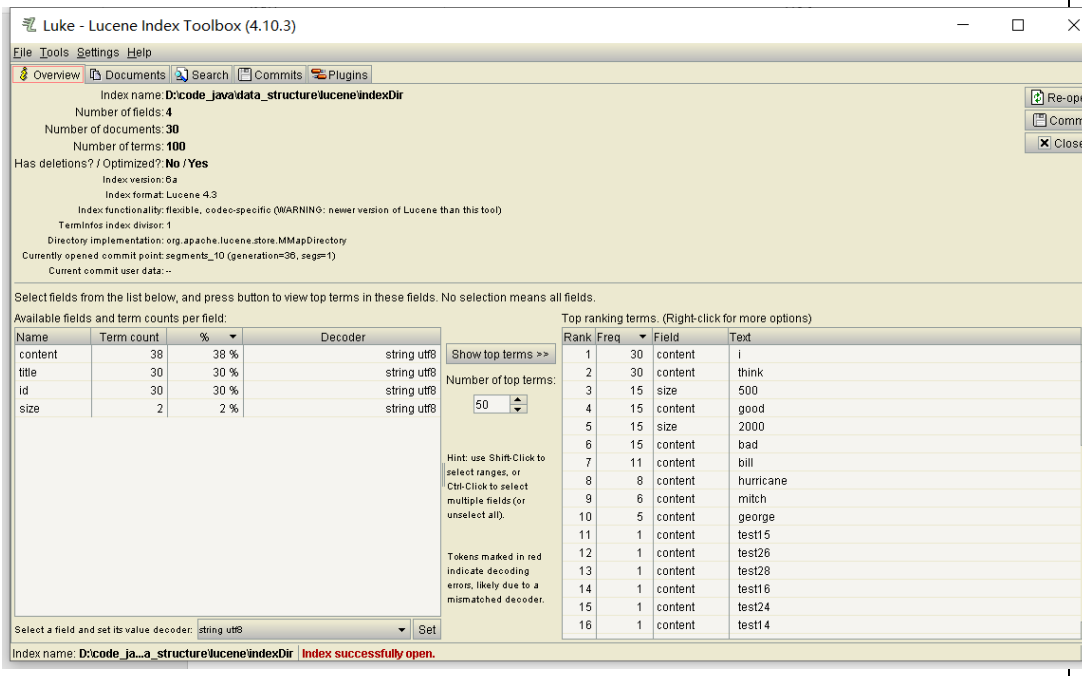
1 创建文档对象
2 创建存储目录
3 创建分词器
4 创建索引写入器的配置对象
5 创建索引写入器对象
6 将文档交给索引写入器
7 提交
8 关闭

```java
public static void testCreate() throws IOException {
    Directory directory = FSDirectory.open(new File( pathname: "indexDir"));
    Analyzer analyzer = new StandardAnalyzer();
    IndexWriterConfig conf = new IndexWriterConfig(Version.LATEST, analyzer);
    //conf.setOpenMode(IndexWriterConfig.OpenMode.APPEND);
    conf.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
    IndexWriter indexWriter = new IndexWriter(directory, conf);
    Random rand = new Random(System.currentTimeMillis());
    for (int i = 1; i < 16; i++) {
        Document document = new Document();
        String temp = "This is the test" + i + ", and I think it is good.";
        int j = rand.nextInt( bound: 4);
        temp += random[j];
        document.add(new StringField( name: "id", String.valueOf(i), Field.Store.YES));
        document.add(new TextField( name: "title",  value: "test" + i, Field.Store.YES));
        document.add(new TextField( name: "content", temp, Field.Store.YES));
        document.add(new TextField( name: "size",  value: "2000", Field.Store.YES));
        indexWriter.addDocument(document);
    }
    for (int i = 16; i < 31; i++) {
        Document document1 = new Document();
        String temp = "This is the test" + i + ", and I think it is bad.";
        int j = rand.nextInt( bound: 4);
        temp += random[j];
        document1.add(new StringField( name: "id", String.valueOf(i), Field.Store.YES));
        document1.add(new TextField( name: "title",  value: "test" + i, Field.Store.YES));
        document1.add(new TextField( name: "content", temp, Field.Store.YES));
        document1.add(new TextField( name: "size",  value: "500", Field.Store.YES));
        indexWriter.addDocument(document1);
    }
    indexWriter.commit();
    indexWriter.close();
```

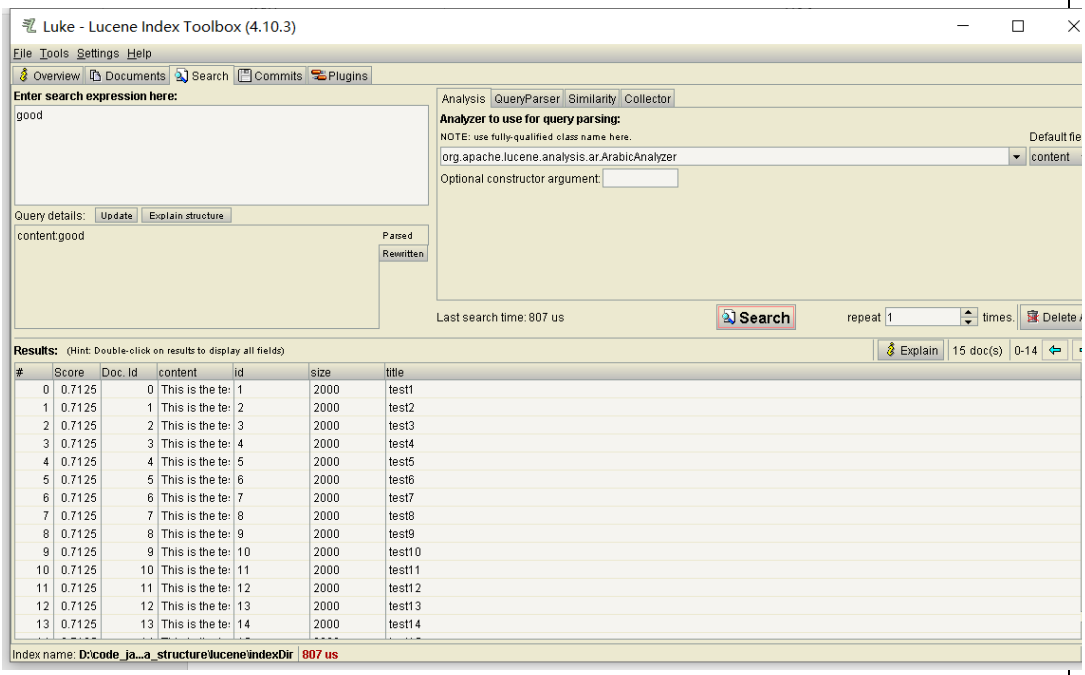
由于没有测试文件，所以在创建索引的过程中，我们直接自己生成一些 document，然后再创建索引。

## (2) 使用工具查看索引

在 GitHub - DmitryKey/luke: This is mavenised Luke: Lucene Toolbox Project 下载对应的 luke，需要和 Lucene 版本对应。




即可查看索引文件。




在此处，我们也可以进行简单的 search 操作。

(3)获取用户输入

```
public static void main(String[] args) throws IOException, ParseException {
    testCreate();
    while (true) {
        //search --hits=123 qwe qw eqw
        Scanner s = new Scanner(System.in);
        String command = s.nextLine();
        if (command.equals("exit")) {
            break;
        }
        String[] commands = command.split( regex: " ", limit: 2);
        if (!commands[0].equals("search")) {
            System.out.println("FORMAT ERROR!");
            continue;
        }
        String[] commands1 = commands[1].split( regex: " ", limit: 2);
        if (commands1[0].contains("--hits=")) {
            K = Integer.parseInt(commands1[0].split( regex: "--hits=")[1]);
            System.out.println(K);
            commands[1] = commands1[1];
        }
        command = commands[1];
        command = command.replaceAll( regex: "[^a-zA-Z ]", replacement: "");
        command = command.toLowerCase();
        if (command.equals("")) {
            System.out.println("FORMAT ERROR!");
            continue;
        }
        testSearch(command);
    }
}
```

(4)查询索引数据

1 创建读取目录对象
2 创建索引读取工具
3 创建索引搜索工具
4 创建查询解析器
5 创建查询对象
6 搜索数据
7 各种操作

```
String[] commands = command.split( regex: " ");
for (int i = 0; i < commands.length; i++) {
    if (commands[i].equals("and")) {
        commands[i] = "AND";
    } else if (commands[i].equals("not")) {
        commands[i] = "NOT";
    } else if (commands[i].equals("or")) {
        commands[i] = "OR";
    }
}
command = "";
for (int i = 0; i < commands.length - 1; i++) {
    command += commands[i] + " ";
}
command += commands[commands.length - 1];
```

首先，处理用户的输入，将 and、or、not 三个逻辑运算转为大写字母，用于进行基本的布尔搜索操作。

```
MySelfScore mySelfScore = new MySelfScore();
mySelfScore.searchBySelfScore(command, K);
System.out.println("--------" + command + "----------");
//System.out.println(command);
Directory directory = FSDirectory.open(new File( pathname: "indexDir"));
IndexReader reader = DirectoryReader.open(directory);
IndexSearcher searcher = new IndexSearcher(reader);
Analyzer analyzer = new StandardAnalyzer();
QueryParser parser = new QueryParser( f: "content", analyzer);
Query query = parser.parse(command);
TopDocs topDocs = searcher.search(query, K);
if (topDocs.totalHits == 0) {
    System.out.println("None");
}
ScoreDoc[] scoreDocs = topDocs.scoreDocs;
int i = 1;
for (ScoreDoc scoreDoc : scoreDocs) {
    int docID = scoreDoc.doc;
    Document doc = reader.document(docID);
    System.out.print(i++);
    System.out.print(" [" + scoreDoc.score + "]");
    System.out.println(" " + doc.get("title"));
    System.out.println(doc.get("content"));
}
System.out.println();
reader.close();
```

在此处，我们使用两种评分机制，一种是系统自带的，另一种是我们自己定义的 `MySelfScore`

```java
@Override
public float customScore(int doc, float subQueryScore, float valSrcScore)
        throws IOException {
    FieldCache.Longs longs = FieldCache.DEFAULT.getLongs(context.reader(), s: "size", b: false);
    long size = longs.get(doc);
    if (size > 1000) {
        return subQueryScore * valSrcScore * 100;
    }
    return subQueryScore * valSrcScore;
}
```

**【实验结果分析】：**

用户输入测试：我们在此处获取用户输入，并进行预处理，如果用户没有输入 search 则返回 FORMAT ERROR，随后获取—hits=后边的整数，即排名前 K 个文档中的 K 的值，随后使用正则表达式，去除所有非字母和空格的字符，随后将所有字母转为小写字母，在进行相关 query 的查询操作。

```
aasd
FORMAT ERROR!
search
FORMAT ERROR!
search good
1 [71.251625] test1
This is the test1, and I think it is good. mitch.
2 [71.251625] test2
This is the test2, and I think it is good. bill.
3 [71.251625] test3
This is the test3, and I think it is good. mitch.
4 [71.251625] test4
This is the test4, and I think it is good. hurricane.
5 [71.251625] test5
This is the test5, and I think it is good. mitch.
--------good----------
1 [0.71251625] test1
This is the test1, and I think it is good. mitch.
2 [0.71251625] test2
This is the test2, and I think it is good. bill.
3 [0.71251625] test3
This is the test3, and I think it is good. mitch.
4 [0.71251625] test4
This is the test4, and I think it is good. hurricane.
5 [0.71251625] test5
This is the test5, and I think it is good. mitch.
```

```
search --hits=2 good
2
1 [71.251625] test1
This is the test1, and I think it is good. mitch.
2 [71.251625] test2
This is the test2, and I think it is good. bill.
--------good----------
1 [0.71251625] test1
This is the test1, and I think it is good. mitch.
2 [0.71251625] test2
This is the test2, and I think it is good. bill.
```

自定义评分标准：获取文档的 size 域，并判断该 size 是否大于 1000，如果大于 1000，则将其评分扩大为原来的 100 倍。

```
search good and bad
--------good AND bad----------
None
```

```
search --hits=2 good and hurricane
2
1 [124.07515] test1
This is the test1, and I think it is good. hurricane.
2 [124.07515] test2
This is the test2, and I think it is good. hurricane.
--------good AND hurricane----------
1 [1.2407515] test1
This is the test1, and I think it is good. hurricane.
2 [1.2407515] test2
This is the test2, and I think it is good. hurricane.
```

**【实验总结】：**

本次实验通过使用 lucene 实现了一个简单的文件信息检索，学习了其与传统的数据库查询以及顺序扫描法的区别，学习了全文检索（倒排索引）的优点，了解了 lucene 的评分机制，并自定义了评分机制。

遇到的难点：

1、用户输入的数据不总是合法的，所以我们在获取用户数据的时候需要考虑到大量的情况，来保证程序的健壮性。

2、在自定义评分机制的时候，查阅了大量的资料，通过自定义类来实现。

3、通过文档，学习了 lucene 的倒排索引的使用，索引的创建以及查询。

| 评语及评分（指导教师） |
| --- |
| **【评语】：**<br>该实验报告设计并实现了基于 Lucene 的文件信息检索，距达到设计目标有一段距离。该实验报告篇幅较短且内容不够充实，缺乏数据集说明、定量评价，未对不同算法进行横向比较。<br><br>**评分：75**<br><br>**日期：2021-06-25** |