



第一章 lucene 技术（一）

课程目标

- 1、 Lucene 概述
- 2、 Lucene 开发的一般过程
- 3、 创建索引和查询索引

课程内容

1、 Lucene 概述

1.1) 介绍：

Lucene 是 apache 软件基金会 4 jakarta 项目组的一个子项目，是一个[开放源代码](#)的全文检索引擎工具包，但它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分[文本分析](#)引擎（英文与德文两种西方语言）。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序接口，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。

1.2) 数据分类：

我们生活中的数据总体分为两种：结构化数据和非结构化数据。

结构化数据：指具有固定格式或有限长度的数据，如数据库，元数据等。

非结构化数据：指不定长或无固定格式的数据，如邮件，word 文档等磁盘上的文件

1.3) 非结构化查询数据的方式：

（1）顺序扫描法(Serial Scanning)

所谓顺序扫描，比如要找内容包含某一个字符串的文件，就是一个文档一个文档的看，对于每一个文档，从头看到尾，如果此文档包含此字符串，则此文档为我们要找的文件，接着看下一个文件，直到扫描完所有的文件。如利用 windows 的搜索也可以搜索文件内容，只是相当的

慢。

(2) 全文检索(Full-text Search)

将非结构化数据中的一部分信息提取出来，重新组织，使其变得有一定结构，然后对此有一定结构的数据进行搜索，从而达到搜索相对较快的目的。这部分从非结构化数据中提取出的然后重新组织的信息，我们称之为索引。

例如：字典。字典的拼音表和部首检字表就相当于字典的索引，对每一个字的解释是非结构化的，如果字典没有音节表和部首检字表，在茫茫辞海中找一个字只能顺序扫描。然而字的某些信息可以提取出来进行结构化处理，比如读音，就比较结构化，分声母和韵母，分别只有几种可以一一列举，于是将读音拿出来按一定的顺序排列，每一项读音都指向此字的详细解释的页数。我们搜索时按结构化的拼音搜到读音，然后按其指向的页数，便可找到我们的非结构化数据——也即对字的解释。

这种先建立索引，再对索引进行搜索的过程就叫全文检索(Full-text Search)。

虽然创建索引的过程也是非常耗时的，但是索引一旦创建就可以多次使用，全文检索主要处理的是查询，所以耗时间创建索引是值得的。

(3) 如何实现全文检索：

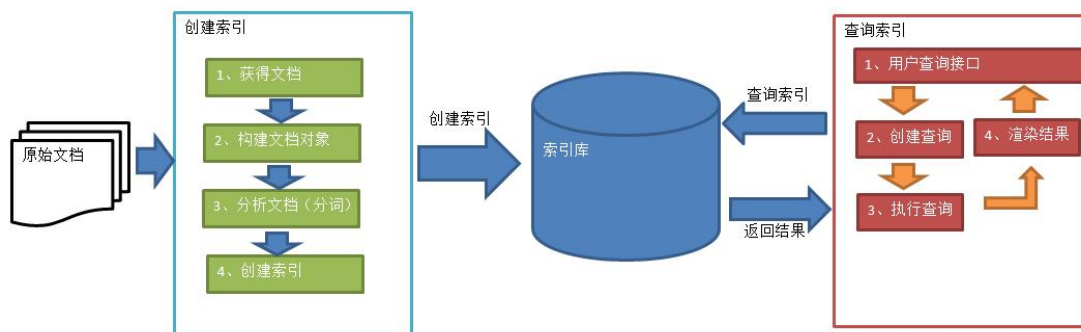
可以使用 Lucene 实现全文检索。Lucene 是 apache 下的一个开放源代码的全文检索引擎工具包。提供了完整的查询引擎和索引引擎，部分文本分析引擎。Lucene 的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能。

(4) 全文检索应用场景：

对于数据量大、数据结构不固定的数据可采用全文检索方式搜索，比如百度、Google 等搜索引擎、论坛站内搜索、电商网站站内搜索等。

2、 Lucene 实现全文检索的流程：

2.1) Lucene 实现全文检索的原理图：



1、绿色表示索引过程，对要搜索的原始内容进行索引构建一个索引库，索引过程包括：确定原始内容即要搜索的内容→采集文档→创建文档→分析文档→索引文档

2、红色表示搜索过程，从索引库中搜索内容，搜索过程包括：

用户通过搜索界面→创建查询→执行搜索，从索引库搜索→渲染搜索结果

2.2) 创建索引：

对文档索引的过程，将用户要搜索的文档内容进行索引，索引存储在索引库（index）中。

这里我们要搜索的文档是磁盘上的文本文件，根据案例描述：**凡是文件名或文件内容包括关键字的文件都要找出来，这里要对文件名和文件内容创建索引。**

2.2.1) 获得原始文档：

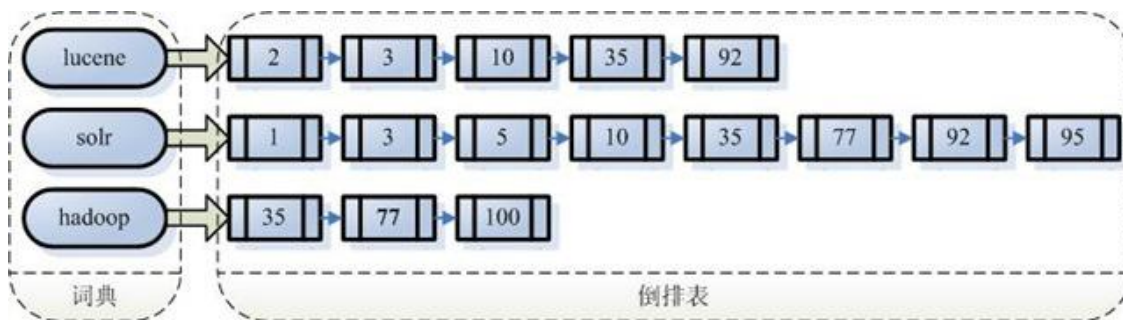
| | | |
|---|---------|---|
| 1. create web page. txt | 47 | 1 |
| 2. Serving Web Content. txt | 35 | |
| apache lucene. txt | 724 | |
| Apache_Lucene_README. txt | 724 | |
| cx4_README. txt | 3,770 | |
| lucene_changs. txt | 501,873 | |
| mybatis. txt | 699 | |
| mybatis_NOTICE. txt | 3,194 | |
| spring. txt | 83 | |
| spring_README. txt | 3,255 | |
| springmvc. txt | 2,126 | |
| SYSTEM_REQUIREMENTS. txt | 957 | |
| Welcome to the Apache Solr project. txt | 5,464 | |
| 全文检索. txt | 1,687 | |
| 什么是全文检索 java. txt | 1,687 | |

2.2.2) 创建文档对象：

获取原始内容的目的是为了索引，在索引前需要将原始内容创建成文档（Document），文档中包括一个一个的域（Field），域中存储内容。

这里我们可以将磁盘上的一个文件当成一个 document，Document 中包括一些 Field（file_name 文件名称、file_path 文件路径、file_size 文件大小、file_content 文件内容），如下图：

倒排索引结构是根据内容（词语）找文档，如下图：



倒排索引结构也叫反向索引结构，包括索引和文档两部分，索引即词汇表，它的规模较小，而文档集合较大。

2.3) 查询索引：

查询索引也是搜索的过程。搜索就是用户输入关键字，从索引（index）中进行搜索的过程。根据关键字搜索索引，根据索引找到对应的文档，从而找到要搜索的内容（这里指磁盘上的文件）。

2.3.1) 用户查询接口：



Lucene 不提供制作用户搜索界面的功能，需要根据自己的需求开发搜索界面。

2.3.2) 创建查询：

用户输入查询关键字执行搜索之前需要先构建一个查询对象，查询对象中可以指定查询要搜索的 Field 文档域、查询关键字等，查询对象会生成具体的查询语法，

例如：

语法 “fileName:lucene” 表示要搜索 fileName 域的内容为 “lucene” 的文档

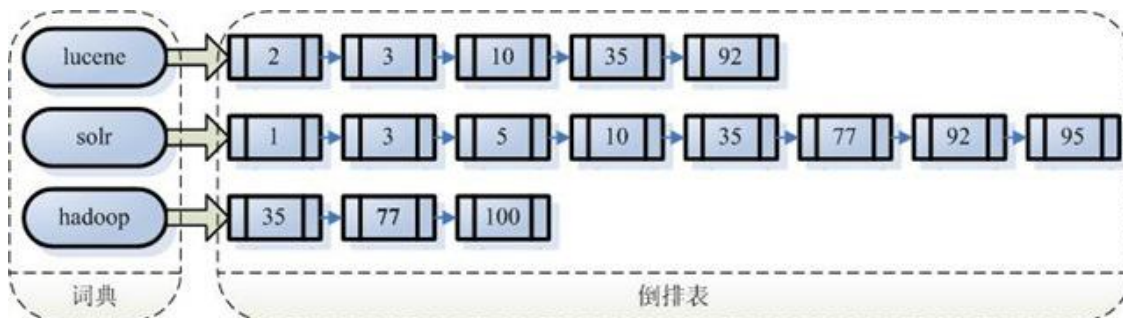
2.3.3) 执行查询：

搜索索引过程：

根据查询语法在倒排索引词典表中分别找出对应搜索词的索引，从而找到索引所链接的文档链表。

比如搜索语法为“fileNme:lucene”表示搜索出 fileNme 域中包含 Lucene 的文档。

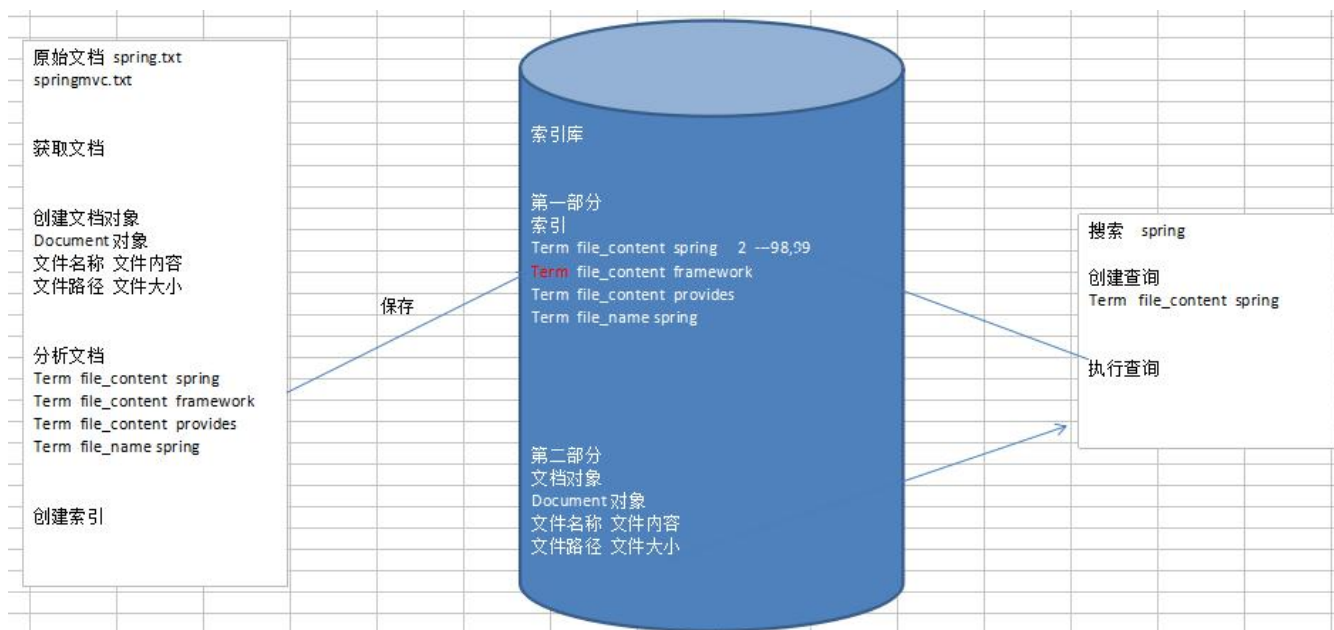
搜索过程就是在索引上查找域为 fileNme，并且关键字为 Lucene 的 term，并根据 term 找到文档 id 列表。



2.3.4) 渲染结果:

以一个友好的界面将查询结果展示给用户，用户根据搜索结果找自己想要的信息，为了帮助用户很快找到自己的结果，提供了很多展示的效果，比如搜索结果中将关键字高亮显示，百度提供的快照等。

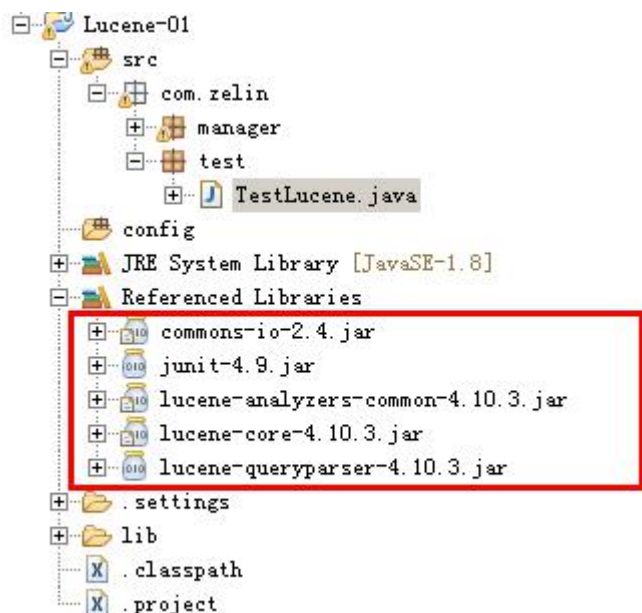
2.4) Lucene 实现创建索引、查询索引原理图:



3、 Lucene 索引库操作：

3.1) 添加索引库：

第一步：添加 jar 包：



第二步：编写 **LuceneManager** 类，定义创建索引库方法：

```
/**
 * 作者：王峰
 * 文件：LuceneManager.java
 * 类名：LuceneManager
 * 时间：2018 年 6 月 29 日 上午 9:15:46
 * 功能：
 */
public class LuceneManager {
    //创建索引
    public void createIndex() throws IOException{
        //1.定义索引库的存放目录
        Directory directory = FSDirectory.open(new File("e:/myindex"));
        //2.构建分词器(分析器) ,StandardAnalyzer 是标准分词器，官方建议使用的
        Analyzer analyzer = new StandardAnalyzer();
        //3.创建索引输出流的配置
        IndexWriterConfig config = new IndexWriterConfig(Version.LATEST,analyzer);
        //4.构造 IndexWriter 输出流对象
        IndexWriter writer = new IndexWriter(directory, config);
        //5.读取 e:/searchdocument 下的文件并遍历成 File 对象(原始文档)
        File searchDocs = new File("e:/searchdocument");
        //6.遍历得到原始文档中所有的文件对象
        File[] files = searchDocs.listFiles();
        for (File file : files) {
            //7.将文档对象存放到索引库中
            //7.1)得到文件相关的属性
            String fileName = file.getName(); //文件名
            String filePath = file.getAbsolutePath(); //得到文件路径
            String fileContent = FileUtils.readFileToString(file); //读取文件内容
            long fileSize = FileUtils.sizeOf(file); //得到文件大小
            //7.2)根据文件相关的属性定义域对象
            //定义文件名域(分析、索引、存储)
            Field fileNameField = new TextField("fileNameField", fileName, Store.YES);
            //定义文件路径域(不分析，不索引，必须存储)
            Field filePathField = new StoredField("filePathField", filePath);
            //定义文件内容域(分析、索引、存储)
            Field fileContextField = new
            TextField("fileContextField",fileContent,Store.YES);
            //定义文件大小域(分析、索引、存储)
            Field fileSizeField = new LongField("fileSizeField", fileSize, Store.YES);
            //7.3)将上面定义的各种域添加到文档中
            Document document = new Document();
            document.add(fileNameField);
            document.add(filePathField);
            document.add(fileContextField);
        }
    }
}
```



```
document.add(fileSizeField);
//8.利用 IndexWriter 对象将文档对象输出到索引库中
writer.addDocument(document);
}
//9.关闭输出流
writer.close();
}
}
```

第三步: 测试创建索引库

```
public class TestLucene {
    private LuceneManager manager;
    @Before
    public void init(){
        manager = new LuceneManager();
    }
    //测试创建索引库
    @Test
    public void testCreateIndex() throws IOException{
        manager.createIndex();
        System.out.println("创建索引库成功!");
    }
}
```

| Field 类 | 数据类型 | Analyzed 是否分析 | Indexed 是否索引 | Stored 是否存储 | 说明 |
|--|---------------------|------------------|-----------------|----------------|--|
| StringField(FieldName, FieldValue,Store.YES)) | 字符串 | N | Y | Y 或 N | 这个 Field 用来构建一个字符串 Field, 但是不会进行分析, 会将整个串存储在索引中, 比如(订单号,姓名等) 是否存储在文档中用 Store.YES 或 Store.NO 决定 |
| LongField(FieldName, FieldValue,Store.YES) | Long 型 | Y | Y | Y 或 N | 这个 Field 用来构建一个 Long 数字型 Field, 进行分析和索引, 比如(价格) 是否存储在文档中用 Store.YES 或 Store.NO 决定 |
| StoredField(FieldName, FieldValue) | 重载方法, 支持多种类 型 | N | N | Y | 这个 Field 用来构建不同类型 Field 不分析, 不索引, 但要 Field 存储在文档中 |
| TextField(FieldName, FieldValue, Store.NO) 或 TextField(FieldName, reader) | 字符串 或 流 | Y | Y | Y 或 N | 如果是一个 Reader, lucene 猜测内容 比较多,会采用 Unstored 的策略. |

3.2) 查询索引库：

第一步： 在 **LuceneManager** 中定义如下的查询索引库的方法：

//查询索引

```
public void searchIndex() throws Exception{
    //1.获取索引库所在的目录对象
    Directory directory = FSDirectory.open(new File("e:/myindex"));
    //2.根据目录对象得到相应的输入流
    IndexReader reader = DirectoryReader.open(directory );
    //3.根据输入流对象构建索引搜索器对象
    IndexSearcher searcher = new IndexSearcher(reader );
    //4.构建一个查询对象
    Query query = new TermQuery(new Term("fileContextField", "java"));
    //5.根据搜索器对象及查询对象得到前 10 条记录(其中 TopDocs 中存放的是文档的 id),这里的
    //10 代表只显示的记录条数
    TopDocs docs = searcher.search(query , 10);
    //测试查询的条数（总共有 n 条）
    int totalHits = docs.totalHits;
    System.out.println("满足条件的有: " + totalHits + "条!");
    //6.遍历前 10 条记录
    for(ScoreDoc sd : docs.scoreDocs){
        //得到文档的 id
        int docID = sd.doc;
        //利用搜索器，传递文档的 id，查询出对应的文档对象
        Document doc = searcher.doc(docID);
        //文件名域
        IndexableField fileNameField = doc.getField("fileNameField");
        //文件内容域
        IndexableField fileContentField = doc.getField("fileContextField");
        //文件路径域
        IndexableField filePathField = doc.getField("filePathField");
        //文件大小域
        IndexableField fileSizeField = doc.getField("fileSizeField");
        //输出内容
        System.out.println("文件名: " + fileNameField.stringValue());
        //System.out.println("文件内容: " + fileContentField.stringValue());
        System.out.println("文件路径: " + filePathField.stringValue());
        System.out.println("文件大小: " + fileSizeField.stringValue());
        System.out.println("-----");
    }
    //关闭流
}
```

```
reader.close();
}
```

第二步：测试查询索引：

//测试查询索引库

```
@Test
public void testSearchIndex() throws Exception{
    manager.searchIndex();
}
```

第三步：查询索引结果如下：

满足条件的有：10条！

文件名：SYSTEM_REQUIREMENTS.txt

文件路径：e:\searchdocument\SYSTEM_REQUIREMENTS.txt

文件大小：957

文件名：apache lucene.txt

文件路径：e:\searchdocument\apache lucene.txt

文件大小：724

文件名：Apache_Lucene_README.txt

文件路径：e:\searchdocument\Apache_Lucene_README.txt

文件大小：724

文件名：Welcome to the Apache Solr project.txt

文件路径：e:\searchdocument\Welcome to the Apache Solr project.txt

文件大小：5464

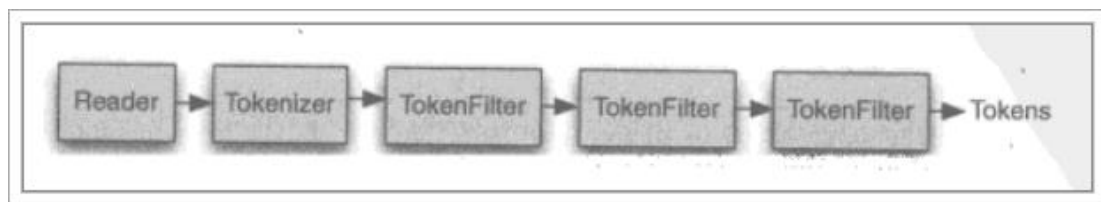
文件名：什么是全文检索 java.txt

文件路径：e:\searchdocument\什么是全文检索 java.txt

文件大小：1687

补充知识点：

1) 分词器执行原理分析：



从一个 Reader 字符流开始，创建一个基于 Reader 的 Tokenizer 分词器，经过三个 TokenFilter 生成语汇单元 Tokens。

要看分析器的分析效果，只需要看 TokenStream 中的内容就可以了。每个分析器都有一个方法 tokenStream，返回一个 tokenStream 对象。

2) 分词器的分词效果原码:

//查看标准分析器的分词效果

```
public void testTokenStream() throws Exception {
    //创建一个标准分析器对象
    Analyzer analyzer = new StandardAnalyzer();
    //获得 tokenStream 对象
    //第一个参数: 域名, 可以随便给一个
    //第二个参数: 要分析的文本内容
    TokenStream tokenStream = analyzer.tokenStream("test",
"The Spring Framework provides a comprehensive programming and configuration model.");
    //添加一个引用, 可以获得每个关键词
    CharTermAttribute charTermAttribute = tokenStream.addAttribute(CharTermAttribute.class);
    //添加一个偏移量的引用, 记录了关键词的开始位置以及结束位置
    OffsetAttribute offsetAttribute = tokenStream.addAttribute(OffsetAttribute.class);
    //将指针调整到列表的头部
    tokenStream.reset();
    //遍历关键词列表, 通过 incrementToken 方法判断列表是否结束
    while(tokenStream.incrementToken()) {
        //关键词的起始位置
        System.out.println("start->" + offsetAttribute.startOffset());
        //取关键词
        System.out.println(charTermAttribute);
        //结束位置
        System.out.println("end->" + offsetAttribute.endOffset());
    }
    tokenStream.close();
}
```

3) Lucene 自带的中文分词器:

- StandardAnalyzer:

单字分词: 就是按照中文一个字一个字地进行分词。如: “我爱中国”, 效果: “我”、“爱”、“中”、“国”。

- CJKAnalyzer

二分法分词: 按两个字进行切分。如: “我是中国人”, 效果: “我是”、“是中”、“中国” “国人”。

上边两个分词器无法满足需求。

- SmartChineseAnalyzer

对中文支持较好, 但扩展性差, 扩展词库, 禁用词库和同义词库等不好处理

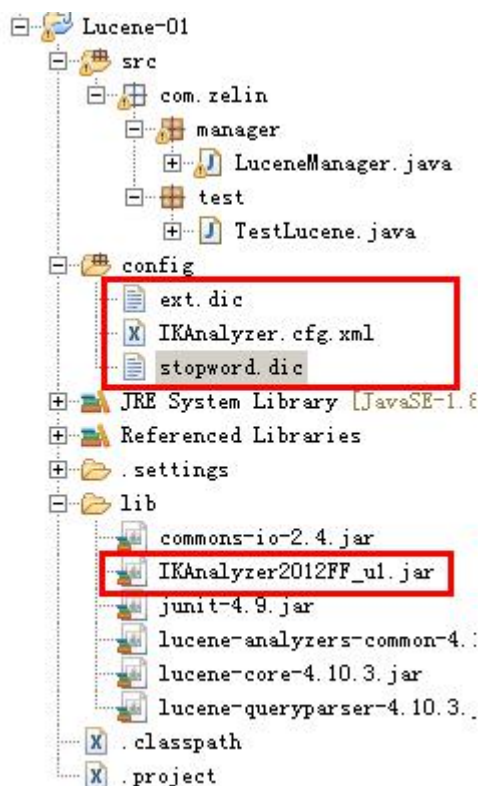
4) 第三方中文分词器:

- paoding: 庖丁解牛最新版在 <https://code.google.com/p/paoding/> 中最多支持 Lucene 3.0, 且最新提交的代码在 2008-06-03, 在 svn 中最新也是 2010 年提交, 已经过时, 不予考虑。

- mmseg4j: 最新版已从 <https://code.google.com/p/mmseg4j/> 移至 <https://github.com/chenlb/mmseg4j-solr>, 支持 Lucene 4.10, 且在 github 中最新提交代码是 2014 年 6 月, 从 09 年~14 年一共有: 18 个版本, 也就是一年几乎有 3 个大小版本, 有较大的活跃度, 用了 mmseg 算法。
- IK-analyzer: 最新版在 <https://code.google.com/p/ik-analyzer/> 上, 支持 Lucene 4.10 从 2006 年 12 月推出 1.0 版开始, IKAnalyzer 已经推出了 4 个大版本。最初, 它是以开源项目 Luence 为应用主体的, 结合词典分词和文法分析算法的中文分词组件。从 3.0 版本开始, IK 发展为面向 Java 的公用分词组件, 独立于 Lucene 项目, 同时提供了对 Lucene 的默认优化实现。在 2012 版本中, IK 实现了简单的分词歧义排除算法, 标志着 IK 分词器从单纯的词典分词向模拟语义分词衍化。但是也就是 2012 年 12 月后没有在更新。**[重点掌握, 开发中常用]**
- ansj_seg: 最新版本在 https://github.com/NLPchina/ansj_seg tags 仅有 1.1 版本, 从 2012 年到 2014 年更新了大小 6 次, 但是作者本人在 2014 年 10 月 10 日说明: “可能我以后没有精力来维护 ansj_seg 了”, 现在由“nlp_china”管理。2014 年 11 月有更新。并未说明是否支持 Lucene, 是一个由 CRF (条件随机场) 算法所做的分词算法。
- imdict-chinese-analyzer: 最新版在 <https://code.google.com/p/imdict-chinese-analyzer/>, 最新更新也在 2009 年 5 月, 下载源码, 不支持 Lucene 4.10。是利用 HMM (隐马尔科夫链) 算法。
- Jcseg: 最新版本在 git.oschina.net/lionsoul/jcseg, 支持 Lucene 4.10, 作者有较高的活跃度。利用 mmseg 算法。

5) 中文分词器 IK-analyzer 的使用:

第一步: 添加 jar 包及配置文件:



说明:

1) IK 分词器中 IKAnalyzer.cfg.xml 文件是全局配置文件, 其内容如下:

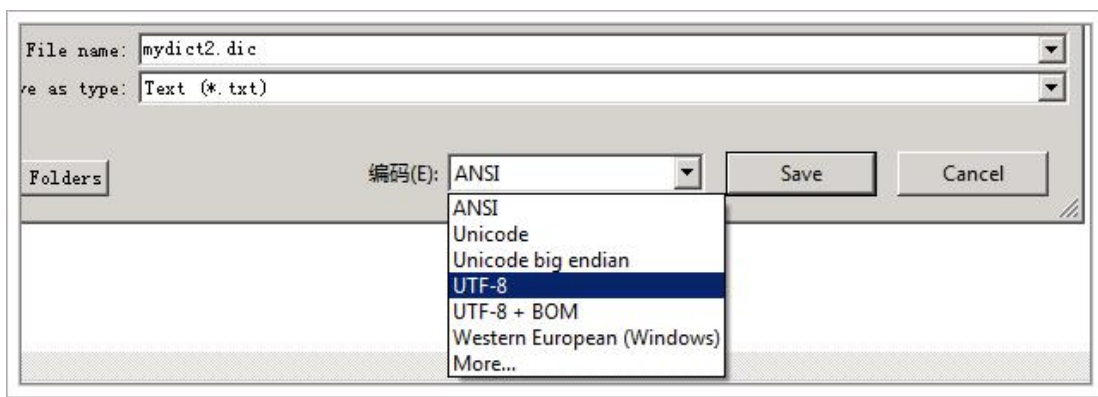
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>IK Analyzer 扩展配置</comment>
    <!--用户可以在这里配置自己的扩展字典 -->
    <entry key="ext_dict">ext.dic;</entry>
    <!--用户可以在这里配置自己的扩展停止词字典 -->
    <entry key="ext_stopwords">stopword.dic;</entry>
</properties>
```

定义扩展词库

定义停用词库

注意: mydict.dic 和 ext_stopword.dic 文件的格式为 UTF-8, 注意是无 BOM 的 UTF-8 编码。

使用 EditPlus.exe 保存为无 BOM 的 UTF-8 编码格式, 如下图:



2) ext.dic 文件中定义了扩展的词库，例如：

高富帅
二维表

3) Stopword.dic 文件中定义了停用词库，如：

去
中
的
用
二
维
表
来

第二步：在 LuceneManager 中使用 IK 分词器：

//Lucene 中的标准分词器代码

```
public void tokenStream() throws IOException{
    //创建一个标准分析器对象
    //Analyzer analyzer = new StandardAnalyzer();
    //使用 IK 中文分词器
    Analyzer analyzer = new IKAnalyzer();
    //获得 tokenStream 对象
    //第一个参数：域名，可以随便给一个
    //第二个参数：要分析的文本内容
    TokenStream tokenStream = analyzer.tokenStream("test",
    "高富帅可以用二维表结构来逻辑表达实现的数据");
    //添加一个引用，可以获得每个关键词
    CharTermAttribute charTermAttribute =
tokenStream.addAttribute(CharTermAttribute.class);
    //添加一个偏移量的引用，记录了关键词的开始位置以及结束位置
    OffsetAttribute offsetAttribute =
tokenStream.addAttribute(OffsetAttribute.class);
    //将指针调整到列表的头部
```

```
tokenStream.reset();  
//遍历关键词列表，通过 incrementToken 方法判断列表是否结束  
while(tokenStream.incrementToken()) {  
    //关键词的起始位置  
    System.out.println("start->" + offsetAttribute.startOffset());  
    //取关键词  
    System.out.println(charTermAttribute);  
    //结束位置  
    System.out.println("end->" + offsetAttribute.endOffset());  
}  
tokenStream.close();  
}
```

第三步：测试运行效果：

加载扩展词典：ext.dic

加载扩展停止词典：stopword.dic

start->0

高富帅

end->3

start->3

可以用

end->6

start->3

可以

end->5

start->6

二维表

end->9

start->6

二维

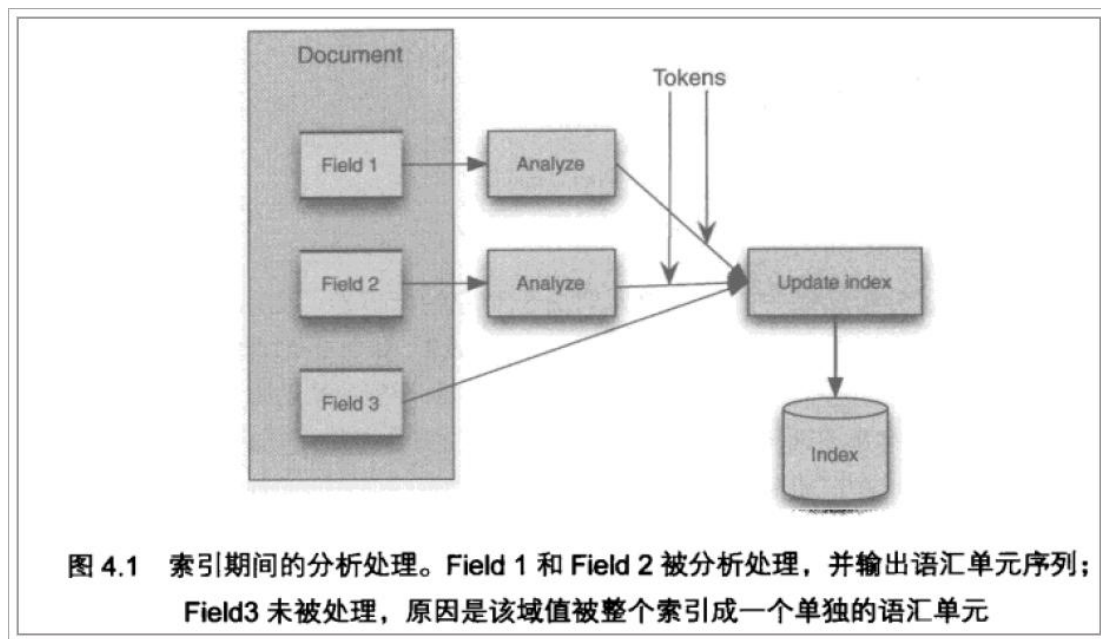
end->8

start->9

结构

了解：在创建索引时使用分析器：

输入关键字进行搜索，当需要让该关键字与文档域内容所包含的词进行匹配时需要对文档域内容进行分析，需要经过 Analyzer 分析器处理生成语汇单元（Token）。分析器分析的对象是文档中的 Field 域。当 Field 的属性 tokenized（是否分词）为 true 时会对 Field 值进行分析，如下图：



对于一些 Field 可以不用分析：

- 1、不作为查询条件的内容，比如文件路径
- 2、不是匹配内容中的词而匹配 Field 的整体内容，比如订单号、身份证号等。

4、 Lucene 索引库维护：（掌握）

4.1) 索引库的添加

第一步：在 LuceneManager 中定义方法：

//添加索引库

```
public void addIndexLib() throws IOException{
    Directory d = FSDirectory.open(new File("E://myindex"));
    Analyzer analyzer = new IKAnalyzer();
    IndexWriterConfig conf = new IndexWriterConfig(Version.LATEST,
analyzer );
    IndexWriter writer = new IndexWriter(d , conf);
    Document doc = new Document();
    //向文档中添加域
    doc.add(new TextField("fnField","新添加的文件名域",Store.YES));
    doc.add(new TextField("fcField","新添加的文件内容域",Store.YES));
    //将文档存放到索引库中
    writer.addDocument(doc );
    //关闭流
    writer.close();
}
```

第二步: 测试方法:

//添加新文档到索引库

@Test

```
public void testAddIndexLib() throws IOException{
    manager.addIndexLib();
    System.out.println("添加到索引库成功!");
}
```

第三步: 测试效果:

| Field | IdfpSVBNbxtbxDtx | Norm | Value |
|----------------|--------------------|--------|-----------------------------|
| fcField | ldfp--S--Nnum----- | 0.4375 | 新添加的文件内容域 |
| fileContextFie | ldfp----Nnum----- | --- | <not present or not stored> |
| fileNameFie | ldfp----Nnum----- | --- | <not present or not stored> |
| filePathField | ----- | --- | <not present or not stored> |
| fileSizeField | ld----- | --- | <not present or not stored> |
| fnField | ldfp--S--Nnum----- | 0.375 | 新添加的文件名域 |

4.2) 索引库的修改:(先删除再添加)

第一步: 在 LucenuManager 中添加修改索引库的方法:

//修改索引库 (先删除再添加,在删除库中索引的序号仍保留)

```
public void updateIndexLib() throws IOException{
    //1、获取索引输出流对象
    IndexWriter writer = this.getIndexWriter();
    //2、构造一个需要删除的 term
    Term term = new Term("fileNameField", "apache");
    //3、构造一个需要添加的文档对象
    Document doc = new Document();
    doc.add(new TextField("AA_Field", "修改文档时的 AA 域的内容",Store.YES));
    doc.add(new TextField("BB_Field", "修改文档时的 BB 域的内容",Store.YES));
    //4、对索引库中的文档进行修改操作(参数 1: 要删除的 term, 参数 2: 要添加的 document)
    writer.updateDocument(term, doc);
    //5、关闭流
    writer.close();
}
```

第二步: 测试修改索引库的方法:

//修改索引库的文档

@Test


```
public void testUpdateIndexLib() throws IOException{
    manager.updateIndexLib();
    System.out.println("修改索引库文档成功!");
}
```

第三步：运行效果显示：

Browse by document number:

Doc. #: 0 16

Doc #: 12 DELETED

| Field | IdfpoPSVBNbox#boxDbox | Norm | Value |
|-------|-----------------------|------|-------|
| | | | |

Doc. #: 0

Doc #: 16

| Field | IdfpoSvNBx#bxDx | Norm | Value |
|------------------|--------------------|-------|-----------------------------|
| AA_Field | ldfp--S--Nnum----- | 0.375 | 修改文档时的AA域的内容 |
| BB_Field | ldfp--S--Nnum----- | 0.375 | 修改文档时的BB域的内容 |
| fcField | ldfp-----Nnum----- | --- | <not present or not stored> |
| fileContextField | ldfp-----Nnum----- | --- | <not present or not stored> |
| fileNameField | ldfp-----Nnum----- | --- | <not present or not stored> |
| filePathField | ----- | --- | <not present or not stored> |
| fileSizeField | ldp----- | --- | <not present or not stored> |
| fnField | ldfp-----Nnum----- | --- | <not present or not stored> |

4.3) 删除满足条件的索引库文档:

第一步：在 LuceneManager 中定义删除方法：

//删除满足条件的索引库文档

```
public void deleteIndexLib() throws IOException{
    //1、获取索引输出流对象
    IndexWriter writer = this.getIndexWriter();
    //2.指定要删除的 term
    Term term = new Term("fileContextField","mybatis");
    //3.删除指定的文档
    writer.deleteDocuments(term);
}
```

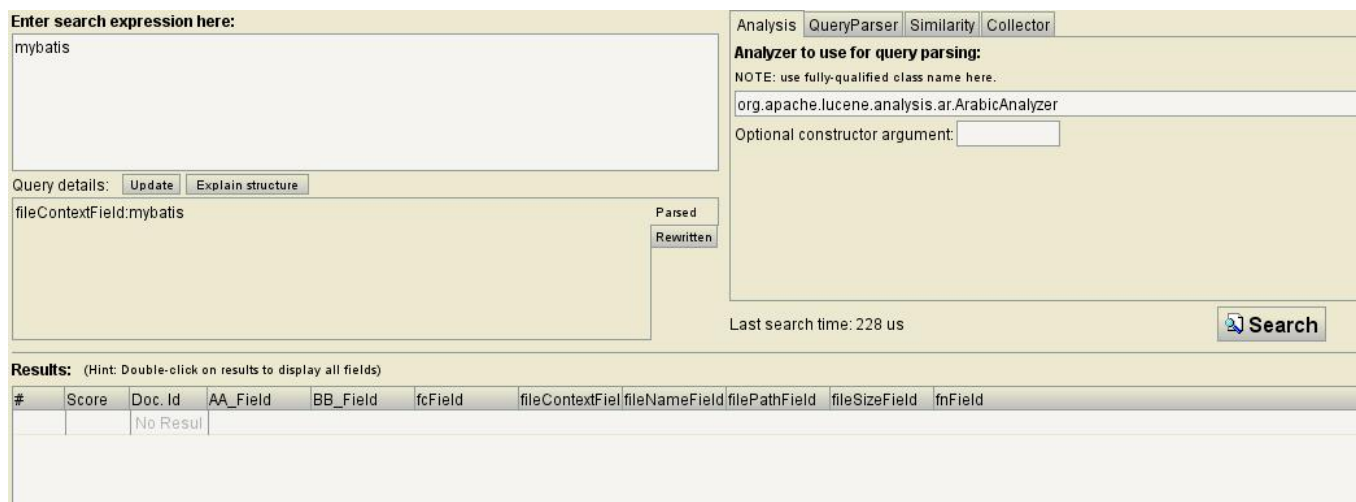
```
//4.关闭流
writer.close();
}
```

第二步：测试删除方法：

//删除索引库中指定条件的文档

```
@Test
public void testdeleteIndexLib() throws IOException{
    manager.deleteIndexLib();
    System.out.println("删除索引库文档成功！");
}
```

第三步：测试删除结果：



Enter search expression here:

mybatis

Query details:

fileContextField:mybatis

Parsed

Analysis

Analyzer to use for query parsing:
NOTE: use fully-qualified class name here.
org.apache.lucene.analysis.ar.ArabicAnalyzer
Optional constructor argument:

Last search time: 228 us

Results: (Hint: Double-click on results to display all fields)

| # | Score | Doc.Id | AA_Field | BB_Field | fcField | fileContextFiel | fileNameField | filePathField | fileSizeField | fnField |
|---|-------|-----------|----------|----------|---------|-----------------|---------------|---------------|---------------|---------|
| | | No Result | | | | | | | | |

4.4) 删除索引库所有文档：

第一步：在 LuceneManger 中定义删除所有索引库的方法：

//删除索引库中的所有的文档

```
public void deleteAllIndexLib() throws IOException{
    // 1、获取索引输出流对象
    IndexWriter writer = this.getIndexWriter();
    // 2、一次性删除所有索引库（慎用）
    writer.deleteAll();
    // 3、关闭流
    writer.close();
}
```

第二步：定义测试方法：

//测试删除所有

```
@Test
public void testDeleteAllIndexLib() throws IOException{
```



```

manager.deleteAllIndexLib();
System.out.println("删除索引库所有文档成功!");
}

```

补充：将如下三个常用功能抽取出来，形成三个方法：

```

/**
 * 根据索引查询器及 query 对象打印查询到的内容
 * @param searcher
 * @param query
 * @throws IOException
 */
private void printRS(IndexSearcher searcher, Query query) throws IOException {
    //5.根据搜索器对象及查询对象得到前 10 条记录(其中 TopDocs 中存放的是文档的 id),这里的
    //10 代表只显示的记录条数
    TopDocs docs = searcher.search(query, 10);
    //测试查询的条数(总共有 n 条)
    int totalHits = docs.totalHits;
    System.out.println("满足条件的有: " + totalHits + "条!");
    //6.遍历前 10 条记录
    for(ScoreDoc sd : docs.scoreDocs){
        //得到文档的 id
        int docID = sd.doc;
        //利用搜索器,传递文档的 id,查询出对应的文档对象
        Document doc = searcher.doc(docID);
        //文件名域
        IndexableField fileNameField = doc.getField("fileNameField");
        //文件内容域
        IndexableField fileContentField = doc.getField("fileContextField");
        //文件路径域
        IndexableField filePathField = doc.getField("filePathField");
        //文件大小域
        IndexableField fileSizeField = doc.getField("fileSizeField");
        //输出内容
        System.out.println("文件名: " + fileNameField.stringValue());
        //System.out.println("文件内容: " + fileContentField.stringValue());
        System.out.println("文件路径: " + filePathField.stringValue());
        System.out.println("文件大小: " + doc.get("fileSizeField")); //可以传递
        key(field 的名称)得到值

        System.out.println("-----");
    }
}

```



```
/**
 * 返回 IndexSearcher 对象
 * @return
 * @throws IOException
 */
public IndexSearcher getIndexSearcher() throws IOException{
    //1.获取索引库所在的目录对象
    Directory directory = FSDirectory.open(new File("e:/myindex"));
    //2.根据目录对象得到相应的输入流
    IndexReader reader = DirectoryReader.open(directory );
    //3.根据输入流对象构建索引搜索器对象
    return new IndexSearcher(reader );
}

/**
 * 获取 IndexWriter
 * @return
 * @throws IOException
 */
private IndexWriter getIndexWriter() throws IOException {
    Directory d = FSDirectory.open(new File("E://myindex"));
    Analyzer analyzer = new IKAnalyzer();
    IndexWriterConfig conf = new IndexWriterConfig(Version.LATEST, analyzer );
    IndexWriter writer = new IndexWriter(d , conf);
    return writer;
}
```

4.5) 查询索引库: (重点掌握)

对要搜索的信息创建 Query 查询对象，Lucene 会根据 Query 查询对象生成最终的查询语法，类似关系数据库 Sql 语法一样 Lucene 也有自己的查询语法，比如：“name:lucene”表示查询 Field 的 name 为“lucene”的文档信息。

可通过两种方法创建查询对象：

1) 使用 Lucene 提供 Query 子类

Query 是一个抽象类，lucene 提供了很多查询对象，比如 TermQuery 项精确查询，NumericRangeQuery 数字范围查询等。

如下代码：

```
Query query = new TermQuery(new Term("name", "lucene"));
```

2) 使用 QueryParser 解析查询表达式

QueryParser 会将用户输入的查询表达式解析成 Query 对象实例。

如下代码：

```
QueryParser queryParser = new QueryParser("name", new IKAnalyzer());
Query query = queryParser.parse("name:lucene");
```

第一部分：利用 Lucene 提供的 Query 子类实现：

4.5.1) 查询所有：MatchAllDocsQuery 对象的使用

第一步：在 **LuceneManager** 中定义方法：

//查询所有文档

```
public void findAllDocs() throws IOException{
    //1.得到 IndexSearcher 对象
    IndexSearcher searcher = this.getIndexSearcher();
    //2.获取查询的 query 对象，这里的 MatchAllDocsQuery，返回的是查询所有文档
    Query query = new MatchAllDocsQuery();
    //3.调用打印输出方法输出查询到的结果
    this.printRS(searcher, query);
    //4.关闭流
    searcher.getIndexReader().close();
}
```

第二步：测试方法：

//测试查询所有文档

```
@Test
public void testFindAllDocs() throws IOException{
    manager.findAllDocs();
}
```

第三步：运行结果如下：

满足条件的有：15条！

文件名：1.create web page.txt

文件路径：e:\searchdocument\1.create web page.txt

文件大小：47

文件名：2.Serving Web Content.txt

文件路径：e:\searchdocument\2.Serving Web Content.txt

文件大小：35

文件名：apache lucene.txt

文件路径：e:\searchdocument\apache lucene.txt

文件大小：724

文件名：Apache_Lucene_README.txt

文件路径：e:\searchdocument\Apache_Lucene_README.txt

文件大小：724

文件名：cxf_README.txt

文件路径：e:\searchdocument\cxf_README.txt

文件大小：3770

4.5.2) TermQuery 对象的使用

一般查询的是不经过分析器的数据。

第一步：在 **LuceneManager** 中定义查询方法：

```
/**
 * 查询指定文档 TermQuery
 */
public void findDocsByTermQuery() throws IOException{
    //1.得到 IndexSearcher 对象
    IndexSearcher searcher = this.getIndexSearcher();
    //2.构造 TermQuery 对象
    TermQuery termQuery = new TermQuery(new
Term("fileNameField", "apache"));
    //3.调用打印输出方法输出查询到的结果
    this.printRS(searcher, termQuery);
    //4.关闭流
    searcher.getIndexReader().close();
}
```

第二步：测试查询方法：

```
//测试通过 TermQuery 查询
@Test
public void testFindDocsByTermQuery() throws IOException{
    manager.findDocsByTermQuery();
}
```

第三步：测试查询结果：

满足条件的有：3条！

文件名：apache lucene.txt

文件路径：e:\searchdocument\apache lucene.txt

文件大小：724

文件名：Apache_Lucene_README.txt

文件路径：e:\searchdocument\Apache_Lucene_README.txt

文件大小：724

文件名：Welcome to the Apache Solr project.txt

文件路径：e:\searchdocument\Welcome to the Apache Solr project.txt

文件大小：5464

4.5.3) NumericRangeQuery 对象的使用

第一步：在 **LuceneManager** 类中定义方法：

```
/**
 * 根据范围进行查询,NumericRangeQuery 查询对象的使用
 */
public void findDocsByNumericRangeQuery() throws IOException{
    //1.得到 IndexSearcher 对象
    IndexSearcher searcher = this.getIndexSearcher();
    //2.定义 NumericRangeQuery 查询对象
    //参数 1:代表查询的域名 参数 2:代表查询的最小值 参数 3:代表查询的最大值 参
    数 4: 代表有无包含最小值 参数 5: 有无包含最大值
    NumericRangeQuery<Long> query =
        NumericRangeQuery.newLongRange("fileSizeField", 0L, 100L,
true, false);
    //3.调用打印输出方法输出查询到的结果
    this.printRS(searcher, query);
    //4.关闭流
    searcher.getIndexReader().close();
}
```

第二步：定义测试方法：

```
//测试通过 NumericRangeQuery 进行查询
@Test
public void testfindDocsByNumericRangeQuery() throws IOException{
    manager.findDocsByNumericRangeQuery();
}
```

第三步：测试结果如下：

满足条件的有：3条！

文件名：1.create web page.txt

文件路径：e:\searchdocument\1.create web page.txt

文件大小：47

文件名：2.Serving Web Content.txt

文件路径：e:\searchdocument\2.Serving Web Content.txt

文件大小：35

文件名：spring.txt

文件路径：e:\searchdocument\spring.txt

文件大小：83



4.5.4) BooleanQuery 对象的用法:

第一步：在 **LuceneManager** 中定义查询方法：

```
/**
 * BooleanQuery: 用于将多个查询条件组织起来
 * Occur.MUST: 必须满足此条件，相当于 and
 * Occur.SHOULD: 应该满足，但是不满足也可以，相当于 or
 * Occur.MUST_NOT: 必须不满足。相当于 not
 * @throws IOException
 */
public void findDocsByBooleanQuery() throws IOException{
    //1.得到 IndexSearcher 对象
    IndexSearcher searcher = this.getIndexSearcher();
    //2.构造用于 BooleanQuery 的基本查询
    Query query1 = new TermQuery(new Term("fileNameField","lucene"));
    Query query2 = new TermQuery(new Term("fileContextField","lucene"));
    //3.构造 BooleanQuery 对象
    BooleanQuery query = new BooleanQuery();
    query.add(query1, Occur.SHOULD);//Occur.SHOULD: 应该满足，但是不满足也可以，相当于 or
    query.add(query2, Occur.MUST);//Occur.MUST: 必须满足此条件，相当于 and
    //3.调用打印输出方法输出查询到的结果
    this.printRS(searcher, query);
    //4.关闭流
    searcher.getIndexReader().close();
}
```

第二步：测试查询方法：

```
//测试通过 BooleanQuery 进行查询
@Test
public void testFindDocsByBooleanQuery() throws IOException{
    manager.findDocsByBooleanQuery();
}
```

第三步：运行结果如下：

满足条件的有：5条！

文件名：apache lucene.txt

文件路径：e:\searchdocument\apache lucene.txt

文件大小：724

文件名：lucene_changs.txt

文件路径：e:\searchdocument\lucene_changs.txt

文件大小：501873

文件名：Apache_Lucene_README.txt

文件路径：e:\searchdocument\Apache_Lucene_README.txt

文件大小：724

文件名：SYSTEM_REQUIREMENTS.txt

文件路径：e:\searchdocument\SYSTEM_REQUIREMENTS.txt

文件大小：957

文件名：Welcome to the Apache Solr project.txt

文件路径：e:\searchdocument\Welcome to the Apache Solr project.txt

文件大小：5464

第二部分：利用 QueryParser 实现：

4.5.5) 通过 QueryParser 进行分析查询：

第一步：在 LuceneManager 类中定义查询方法：

```
/**
 * 通过 QueryParser 查询分析器进行查询
 * @throws Exception
 */
public void findDocsByQueryParser() throws Exception{
    //1.得到 IndexSearcher 对象
    IndexSearcher searcher = this.getIndexSearcher();
    //2.定义 QueryParser 对象
    QueryParser parser = new QueryParser("fileContextField",new IKAnalyzer());
    //通过 parser 对象得到 query 对象(此例会对下面这句话进行分词解析，其语汇单元为：
    lucene java development)
    Query query = parser.parse("lucene is java development.");
    //3.调用打印输出方法输出查询到的结果
    this.printRS(searcher, query);
    //4.关闭流
    searcher.getIndexReader().close();
}
```

第二步：测试查询方法：

//测试通过 QueryParser 进行查询

```
@Test
public void testFindDocsByQueryParser() throws Exception{
    manager.findDocsByQueryParser();
}
```

第三步：查询结果如下：

加载扩展词典：ext.dic

加载扩展停止词典：stopword.dic

满足条件的有：11条！

文件名：apache lucene.txt

文件路径：e:\searchdocument\apache lucene.txt

文件大小：724

文件名：Apache_Lucene_README.txt

文件路径：e:\searchdocument\Apache_Lucene_README.txt

文件大小：724

文件名：lucene_changs.txt

文件路径：e:\searchdocument\lucene_changs.txt

文件大小：501873

文件名：SYSTEM_REQUIREMENTS.txt

文件路径：e:\searchdocument\SYSTEM_REQUIREMENTS.txt

文件大小：957

文件名：Welcome to the Apache Solr project.txt

文件路径：e:\searchdocument\Welcome to the Apache Solr project.txt

文件大小：5464

4.5.5) 通过 QueryParser 结合简易语法进行分析查询：

简易查询语法：

1、基础的查询语法，关键词查询：

域名+ “:” +搜索的关键词

例如：content:java

2、范围查询

域名+ “:” +[最小值 TO 最大值]

例如：size:[1 TO 1000]

范围查询在 lucene 中支持数值类型，不支持字符串类型。在 solr 中支持字符串类型。

3、组合条件查询

1) +条件 1+条件 2：两个条件之间是并且的关系 and

例如：+filename:apache +content:apache

2) +条件 1 条件 2：必须满足第一个条件，应该满足第二个条件



例如：+filename:apache content:apache

3) 条件 1 条件 2：两个条件满足其一即可。

例如：filename:apache content:apache

4) -条件 1 条件 2：必须不满足条件 1，要满足条件 2

例如：-filename:apache content:apache

| | |
|-----------------------------------|---------|
| Occur.MUST 查询条件必须满足，相当于 and | +(加号) |
| Occur.SHOULD 查询条件可选，相当于 or | 空(不用符号) |
| Occur.MUST_NOT 查询条件不能满足，相当于 not 非 | -(减号) |

第二种写法：

条件 1 AND 条件 2

条件 1 OR 条件 2

条件 1 NOT 条件 2

第一步：在 LuceneManager 类中定义查询方法：

```
/**
 * 通过 QueryParser 查询分析器进行查询
 * @throws Exception
 */
public void findDocsByQueryParser() throws Exception{
    //1.得到 IndexSearcher 对象
    IndexSearcher searcher = this.getIndexSearcher();
    //2.定义 QueryParser 对象
    QueryParser parser = new QueryParser("fileContextField",new
    IKAnalyzer());

    //通过 parser 对象得到 query 对象(此例会对下面这句话进行分词解析，其语汇单元
    为: lucene java development)
    //Query query = parser.parse("lucene is java development.");
    //Query query = parser.parse("fileContextField:apache");
    //++号相当于: and 空格: 相当于: or -号相当于: not
    Query query = parser.parse("+fileNameField:lucene
    fileContextField:lucene");
    //3.调用打印输出方法输出查询到的结果
    this.printRS(searcher, query);
    //4.关闭流
    searcher.getIndexReader().close();
}
```

第二步：测试查询方法：

//测试通过 QueryParser 进行查询

@Test

```
public void testFindDocsByQueryParser() throws Exception{
```

```
manager.findDocsByQueryParser();  
}
```

第三步：测试查询结果：

加载扩展词典：ext.dic

加载扩展停止词典：stopword.dic

满足条件的有：3条！

文件名：apache lucene.txt

文件路径：e:\searchdocument\apache lucene.txt

文件大小：724

文件名：lucene_changs.txt

文件路径：e:\searchdocument\lucene_changs.txt

文件大小：501873

文件名：Apache_Lucene_README.txt

文件路径：e:\searchdocument\Apache_Lucene_README.txt

文件大小：724

4.5.6) 通过 MultiFieldQueryParser 进行多域分析查询：

第一步：在 LuceneManager 类中定义查询方法：

```
/**  
 * 通过多个域进行查询  
 * @throws IOException  
 * @throws ParseException  
 */  
public void findDocsByMultiFieldQueryParser() throws Exception{  
    //1.得到 IndexSearcher 对象  
    IndexSearcher searcher = this.getIndexSearcher();  
    //2.定义要查询的多个域  
    String[] fields = {"fileNameField","fileContextField"};  
    //3.构建查询分析器  
    QueryParser parser = new MultiFieldQueryParser(fields ,new IKAnalyzer());  
    //4.通过查询分析器得到一个查询对象  
    Query query = parser.parse("lucene is java");  
    //5.调用打印输出方法输出查询到的结果  
    this.printRS(searcher, query);  
    //6.关闭流  
    searcher.getIndexReader().close();  
}
```

第二步：测试查询方法：

```
//测试通过 MultiFieldQueryParser 进行查询
```

@Test

```
public void testFindDocsByMultiFieldQueryParser() throws Exception{  
    manager.findDocsByMultiFieldQueryParser();  
}
```

第三步：运行结果如下：

满足条件的有：11条！

文件名：apache lucene.txt

文件路径：e:\searchdocument\apache lucene.txt

文件大小：724

文件名：Apache_Lucene_README.txt

文件路径：e:\searchdocument\Apache_Lucene_README.txt

文件大小：724

文件名：lucene_changs.txt

文件路径：e:\searchdocument\lucene_changs.txt

文件大小：501873

文件名：什么是全文检索 java.txt

文件路径：e:\searchdocument\什么是全文检索 java.txt

文件大小：1687

文件名：SYSTEM_REQUIREMENTS.txt

文件路径：e:\searchdocument\SYSTEM_REQUIREMENTS.txt

文件大小：957
