

# Apache Lucene

## #一、lucene的基本概念和使用

## #二、lucene的其他操作和高级使用

## #三、lucene的环境配置

## #四、lucene6.0测试结果

## #五、lucene8.0测试结果

## #六、lucene6.0与lucene8.0对比

## #七、出现的异常错误及解决方法

### 一、lucene的基本概念和使用

#### #1 什么是lucene

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。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。

来源：<https://baike.baidu.com/item/Lucene/6753302?fr=aladdin>

lucene官网：<http://lucene.apache.org/>

lucene是一个开源的全文检索工具包，不是完整的一个检索引擎，而是一个检索引擎的架

构，提供了完整的查询引擎、索引引擎和部分文本分析引擎；lucene提供了应用程序接口，能够做全文检索和搜寻，是目前最流行的基于java开源检索工具包。

## #2 什么是全文检索

在理解全文检索之前需要了解的是数据基本有两种类型：

- **结构化数据**：具有**固定格式和有限长度**；例如：数据库、元数据。
- **非结构化数据**：**不定长或者无固定格式**的数据；例如：邮件、word文档等磁盘文件。

对于结构化的数据，搜索起来很简单，例如在数据库中直接使用SQL语句就可以进行搜索。而对于非结构化数据，基本有以下两种方法：

- **顺序扫描法** (Serial Scanning)
- **全文检索** (Full-text Search)

**顺序扫描法**简单来讲就是对于每个文档，**从头到尾的扫描内容**，若此文档中包含此字符串，则此文档为我们要找的文件，接着找下一个文件，直到扫描完所有的文件，这种方式效率比较低，速度很慢。

全文检索则是把非结构化数据中的一部分**信息提取出来，将它们重新组织成具有一定结构的数据**，这样的方式可以帮助我们进行很快的搜索，这部分从非结构化数据中提取然后重新组织的信息，称之为**索引**。

**先建立索引，再进行搜索**。这样的方式就是**全文检索** (Full-text Search) 。

## #3 lucene的索引

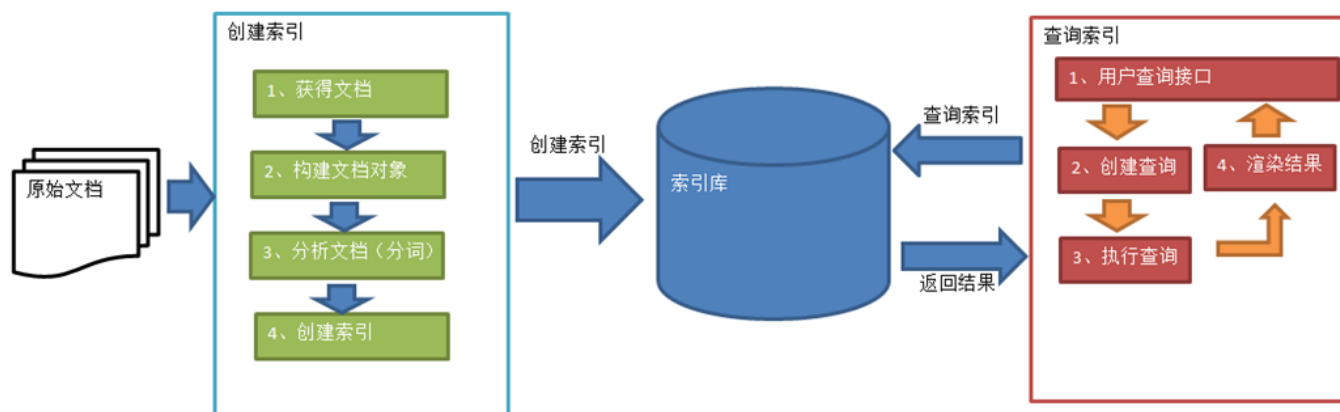
全文检索中建立索引的方式在lucene中就是所谓的倒排索引，也就是建立了一个关键字与文件的**映射关系**。

## Lucene倒排索引原理

- ◆ 假设有两篇文章1和2  
文章1的内容为: Tom lives in Guangzhou,I live in Guangzhou too.  
文章2的内容为: He once lived in Shanghai.
- ◆ 经过分词处理后  
文章1的所有关键词为: [tom] [live] [guangzhou] [i] [live] [guangzhou]  
文章2的所有关键词为: [he] [live] [shanghai]
- ◆ 加上“出现频率”和“出现位置”信息后, 我们的索引结构为:

关键词	文章号[出现频率]	出现位置
guangzhou	1[2]	3, 6
he	2[1]	1
i	1[1]	4
live	1[2],2[1]	2, 5, 2
shanghai	2[1]	3
tom	1[1]	1

## #4 lucene的原理

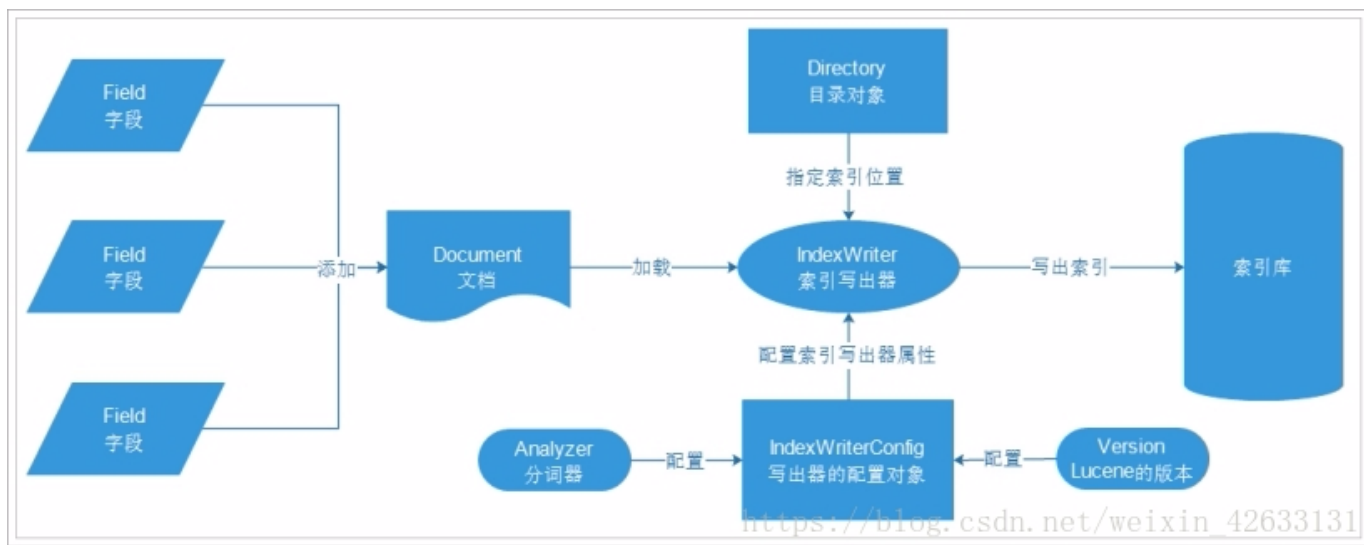


## #5 lucene之索引的创建

1. 获得原始文档（原始内容即要搜索的内容）
2. 采集文档
3. 创建文档
4. 分析文档

## 5. 索引文档

### 👉 创建索引的基本流程



文档Document：数据库中一条具体的记录

字段Field：数据库中的每个字段

目录对象Directory：物理存储位置

写出器的配置对象：需要分词器和lucene的版本

**获取原始文档：**原始文档就是需要进行搜索和建立索引的内容，包括互联网上的网页，磁盘中的文件，数据库中的数据等。不同的数据具有不同的获取方式，如网页可以采用爬虫，磁盘可以通过io流进行获取。

**创建文档：**这里的文档就是lucene中的Document对象，文档中包含各种Field对象，即索引的字段信息，如下图假设一个文件是一个document对象，document对象是一个单条数据，我们对这条数据进行不同字段的拆分，那么其中的字段就可以是文件名、文件大小等内容。我们将会对这样的文档进行索引的建立。

## Document (文件test.txt)

Field1:

Name: 文件名

Value: test.txt

Field2:

Name: 文件大小

Value: 100KB

Field3:

Name: 最后修改时间

Value: 2018-03-04

Field4:

Name: ...

Value: ...

...

**分析文档：**获取了包含field对象的document对象之后，我们需要对document对象进行分析（分词），即对field的内容进行分析，将field的内容分为词汇单元，这其中的操作包括：提取单词，大小写转换，去除符号，去除停用等操作。比如将第一张图中的文章分为了许多关键词。

**创建索引：**对词汇单元进行索引，实现只要通过索引就可以找到文章。索引会被存到内存或者本地磁盘中。

👉 代码实例

```
1 package com.lcl.create;
2 import java.io.File;
3 import org.apache.lucene.analysis.Analyzer;
4 import org.apache.lucene.analysis.standard.StandardAnalyzer;
5 import org.apache.lucene.document.*;
6 import org.apache.lucene.index.IndexWriter;
7 import org.apache.lucene.index.IndexWriterConfig;
8 import org.apache.lucene.store.Directory;
9 import org.apache.lucene.store.FSDirectory;
10 import org.apache.lucene.util.Version;
11 import org.junit.Test;
12 public class CreatIndex {
13     // 创建索引
14     @Test
```

```

15     public void testCreate() throws Exception{
16         //1 创建文档对象
17         Document document = new Document();
18         //创建并添加字段信息。参数：字段的名称、字段的值、是否存储，这里选Store.YES
        代表存储到文档列表Store.NO代表不存储
19         document.add(new StringField("id", "1", Field.Store.YES));
20         // 这里我们title字段需要用TextField，即创建索引又会被分词。
        StringField会创建索引，但是不会被分词
21         document.add(new TextField("title", "谷歌地图之父跳槽facebook",
        Field.Store.YES));
22         //2 索引目录类,指定索引在硬盘中的位置
23         Directory directory = FSDirectory.open(new
        File("C:\\indexDir"));
24         //3 创建分词器对象
25         Analyzer analyzer = new StandardAnalyzer();
26         //4 索引写出工具的配置对象
27         IndexWriterConfig conf = new IndexWriterConfig(Version.LATEST,
        analyzer);
28         //5 创建索引的写出工具类。参数：索引的目录和配置信息
29         IndexWriter indexWriter = new IndexWriter(directory, conf);
30         //6 把文档交给IndexWriter
31         indexWriter.addDocument(document);
32         //7 提交
33         indexWriter.commit();
34         //8 关闭
35         indexWriter.close();
36     }
37     // 批量创建索引
38     @Test
39     public void testCreate2() throws Exception{
40         // 创建文档的集合
41         Collection<Document> docs = new ArrayList<>();
42         // 创建文档对象
43         Document document1 = new Document();
44         document1.add(new StringField("id", "1", Field.Store.YES));
45         document1.add(new TextField("title", "谷歌地图之父跳槽facebook",
        Field.Store.YES));
46         docs.add(document1);
47         // 创建文档对象
48         Document document2 = new Document();
49         document2.add(new StringField("id", "2", Field.Store.YES));
50         document2.add(new TextField("title", "谷歌地图之父加盟FaceBook",

```

```

Field.Store.YES));
51     docs.add(document2);
52     // 创建文档对象
53     Document document3 = new Document();
54     document3.add(new StringField("id", "3", Field.Store.YES));
55     document3.add(new TextField("title", "谷歌地图创始人拉斯离开谷歌
加盟Facebook", Field.Store.YES));
56     docs.add(document3);
57     // 创建文档对象
58     Document document4 = new Document();
59     document4.add(new StringField("id", "4", Field.Store.YES));
60     document4.add(new TextField("title", "谷歌地图之父跳槽Facebook与
Wave项目取消有关",
61                               Field.Store.YES));
62     docs.add(document4);
63     // 创建文档对象
64     Document document5 = new Document();
65     document5.add(new StringField("id", "5", Field.Store.YES));
66     document5.add(new TextField("title", "谷歌地图之父拉斯加盟社交网
站Facebook", Field.Store.YES));
67     docs.add(document5);
68     // 索引目录类,指定索引在硬盘中的位置
69     Directory directory = FSDirectory.open(new
File("C:\\indexDir"));
70     // 引入IK分词器
71     Analyzer analyzer = new IKAnalyzer();
72     // 索引写出工具的配置对象
73     IndexWriterConfig conf = new IndexWriterConfig(Version.LATEST,
analyzer);
74     // 设置打开方式: OpenMode.APPEND 会在索引库的基础上追加新索引。
OpenMode.CREATE会先清空原来数据,再提交新的索引
75     conf.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
76     // 创建索引的写出工具类。参数: 索引的目录和配置信息
77     IndexWriter indexWriter = new IndexWriter(directory, conf);
78     // 把文档集合交给IndexWriter
79     indexWriter.addDocuments(docs);
80     // 提交
81     indexWriter.commit();
82     // 关闭
83     indexWriter.close();
84 }
85 }

```

## #6 lucene索引api详解

### ①Document类（文档类）

Document对象是一条原始的数据

例如：图中就有五个Document对象

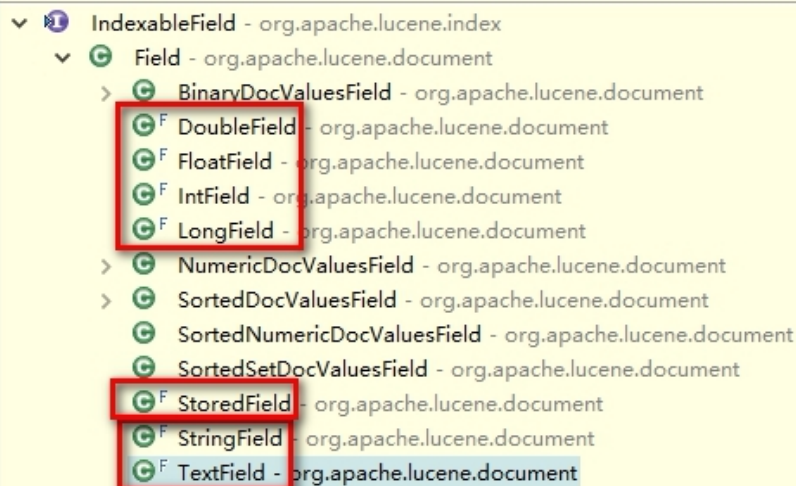
文档编号	文档数据	
	id	title
0	1	谷歌地图之父跳槽Facebook
1	2	谷歌地图之父加盟Facebook
2	3	谷歌地图创始人拉斯离开谷歌加盟Facebook
3	4	谷歌地图之父跳槽Facebook 与wave项目取消有关
4	5	谷歌地图之父拉斯加盟社交网站Facebook

Document中的每一个字段都是一个Field类对象

### ②Field类（字段类）

一个Document中有多个字段，每一个都是一个Field对象，如上图中“id”、“title”都是字段对象。但是字段类下仍有不同的子类，对应了不同需求的字段。





- 1 根据字段Field的类型，选择具体的类
  - 2 DoubleField、FloatField、IntField、LongField、StringField  
一定会被索引，但是一定不分词，不分词查找的时候，就一定要匹配所有的内容，否则搜索不到  
例如：主键id，年龄，生日，日期，价格....
  - 3 TextField一定会被索引，一定会分词
  - 4 StoredField：一定会存储，一定不会被索引
  - 5 Store：是否被存储
- 1 什么样的内容需要被存储？答：查询时，需要显示的内容都应该被存储
  - 2 什么样的内容需要被分词？答：要被搜索的列，就要分词
  - 3 什么样的内容需要被索引？答：要被搜索的列

[https://blog.csdn.net/weixin\\_42633131](https://blog.csdn.net/weixin_42633131)

这些子类如上图中的解释，有不同的特性，基本上是说是否进行分词、建立索引、存储。那么面对不同的字段，我们如何选择不同的字段子类进行建立对象。

#### 1. 如何确定一个字段是否需要被存储？

如果一个字段要显示到最终的结果中，那么一定要存储，否则就不存储

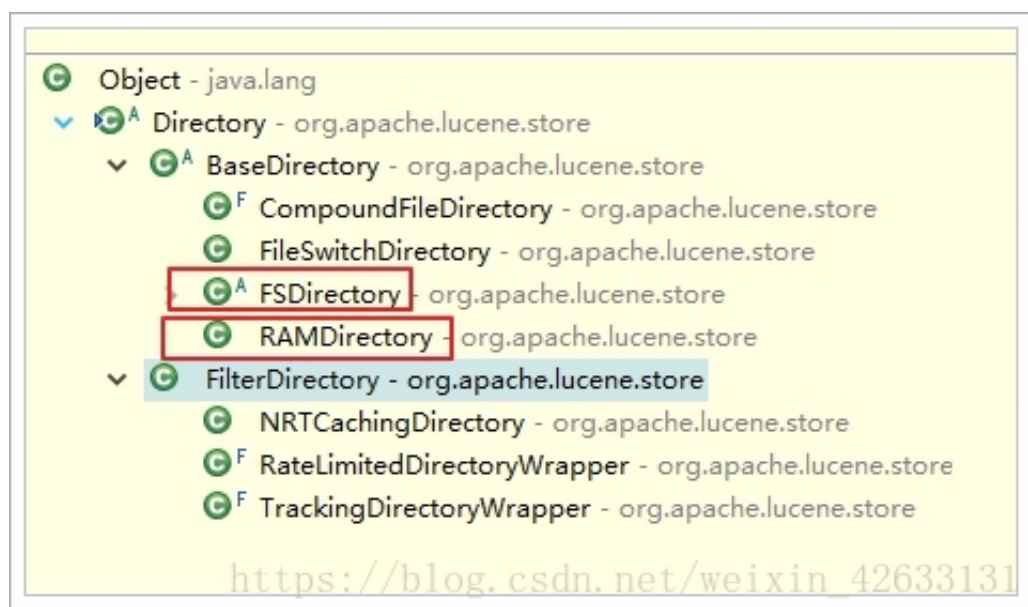
#### 2. 如何确定一个字段是否需要被建立索引？

如果要根据这个字段进行搜索，那么这个字段就必须创建索引。

#### 3. 如何确定一个字段是否需要被分词

前提是这个字段首先要创建索引。然后如果这个字段的值是不可分割的，那么就不需要分词。例如：ID

### ③Directory (目录类)

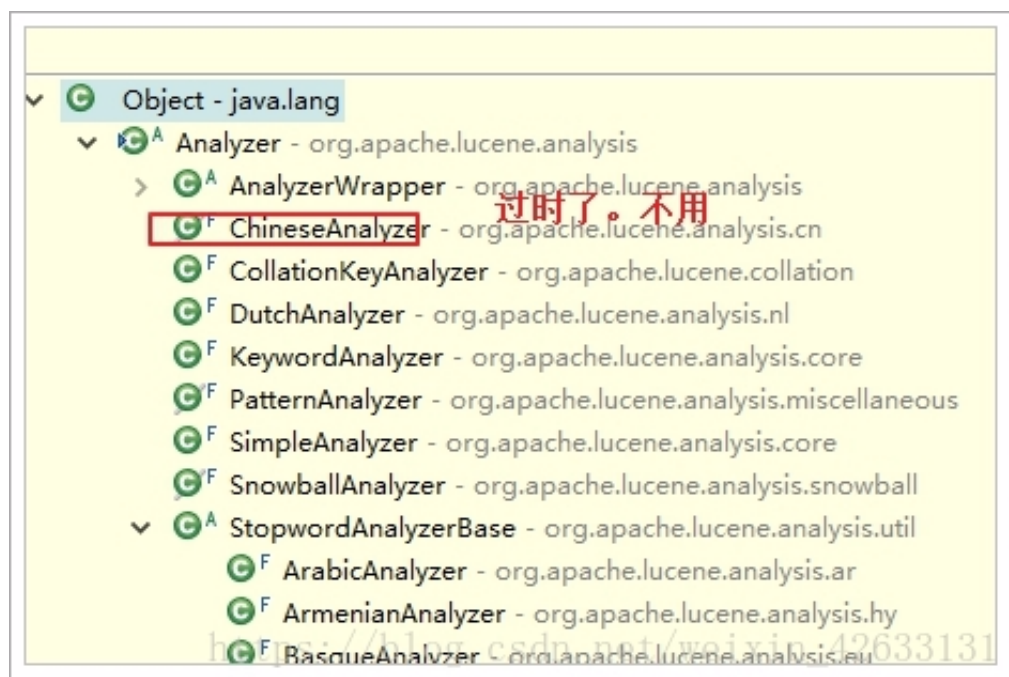


Directory主要有两个子类常用分别是FSDirectory和RAMDirectory，顾名思义他们的区别主要如下

- FSDirectory：文件系统目录，会把索引库指向本地磁盘。  
特点：速度慢但是相对比较安全
- RAMDirectory：内存目录，会把索引存到内存中  
特点：速度快但是不安全

#### ④Analyzer（分词器类）

分词器类主要负责了创建索引流程中的分析文档操作，提供了分词算法，将文档中的内容进行分词



这些lucene中自带的分词器并不适用于中文的分词，因此我们在这里应当要引入第三方中

文分词器。

```
paoding : Lucene中文分词“庖丁解牛” Paoding Analysis
          在PIII 1G内存个人机器上, 1秒 可准确分词 100万 汉字
          http://code.google.com/p/paoding/
imdict : imdict智能词典所采用的智能中文分词程序
          483.64 (字节/秒), 259517(汉字/秒)
          http://code.google.com/p/imdict-chinese-analyzer/
mmseg4j : 用 Chih-Hao Tsai 的 MMSeg 算法 实现的中文分词器
          complex 1200kb/s左右, simple 1900kb/s左右
          http://code.google.com/p/mmseg4j/
ik : 采用了特有的“正向迭代最细粒度切分算法”, 多子处理器分析模式
          具有50万字/秒的高速处理能力
          http://code.google.com/p/ik-analyzer/
https://blog.csdn.net/weixin_42633131
```

常用的是ik分词器

```
1 <!--ik分词器maven依赖-->
2 <dependency>
3   <groupId>com.janeluo</groupId>
4   <artifactId>ikanalyzer</artifactId>
5   <version>2012_u6</version>
6 </dependency>
```

```
1 //3 创建分词器对象
2 // Analyzer analyzer = new StandardAnalyzer();
3 Analyzer analyzer = new IKAnalyzer();
```

⑤IndexWriterConfig (索引写出配置类)

```
1 //4 索引写出工具的配置对象
2 IndexWriterConfig conf = new IndexWriterConfig(Version.LATEST,
  analyzer);
```

配置类中设置lucene的版本信息和分词器。

⑥IndexWriter (索引写出类)

```
1 //5 创建索引的写出工具类。参数：索引的目录和配置信息
2 IndexWriter indexWriter = new IndexWriter(directory, conf);
3 //6 把文档交给IndexWriter
4 indexWriter.addDocument(document);
5 //7 提交
6 indexWriter.commit();
7 //8 关闭
8 indexWriter.close();
```

## #7 lucene之查询操作

### ①查询基本流程

//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

### ②代码实现

```
1 package com.lcl.Query;
2
3 import org.apache.lucene.document.Document;
4 import org.apache.lucene.index.DirectoryReader;
5 import org.apache.lucene.index.IndexReader;
6 import org.apache.lucene.queryparser.classic.ParseException;
7 import org.apache.lucene.queryparser.classic.QueryParser;
8 import org.apache.lucene.search.IndexSearcher;
9 import org.apache.lucene.search.Query;
10 import org.apache.lucene.search.ScoreDoc;
11 import org.apache.lucene.search.TopDocs;
```

```

12 import org.apache.lucene.store.FSDirectory;
13 import org.junit.Test;
14 import org.wltea.analyzer.lucene.IKAnalyzer;
15
16 import java.io.File;
17 import java.io.IOException;
18
19 public class QueryTest {
20     @Test
21     public void QueryTest1() throws IOException, ParseException {
22         //1 创建读取目录对象
23         FSDirectory fsDirectory = FSDirectory.open(new
File("C:\\indexDir"));
24         //2 创建索引读取工具
25         IndexReader indexReader = DirectoryReader.open(fsDirectory);
26         //3 创建索引搜索工具
27         IndexSearcher indexSearcher = new IndexSearcher(indexReader);
28         //4 创建查询解析器
29         // 创建查询解析器,两个参数: 默认要查询的字段名称, 分词器
30         QueryParser queryParser = new QueryParser("title", new
IKAnalyzer());
31         //5 创建查询对象
32         Query query = queryParser.parse("谷歌");
33         //6 搜索数据
34         // 搜索数据,两个参数: 查询条件对象要查询的最大结果条数
35         // 返回的结果是 按照匹配度排名得分前N名的文档信息 (包含查询到的总条
数信息、所有符合条件的文档的编号信息)。
36         TopDocs topDocs = indexSearcher.search(query, 10);
37         //7 各种操作
38         //7.1 获取返回的条数
39         System.out.println("本次搜索找到"+topDocs.totalHits+"条数据");
40         // 7.2获取得分文档对象 (ScoreDoc) 数组.ScoreDoc中包含: 文档的编
号、文档的得分
41         ScoreDoc[] scoreDocs = topDocs.scoreDocs;
42         // 7.3获取内容
43         for (ScoreDoc scoreDoc : scoreDocs) {
44             //获取ID
45             int docID = scoreDoc.doc;
46             //根据ID去找文档
47             Document doc = indexReader.document(docID);
48             System.out.print("文档id: "+doc.get("id"));
49             System.out.print("文档内容: "+doc.get("title"));

```

```

50         //获取得分
51         System.out.println("文档得分: "+scoreDoc.score);
52     }
53     //7.4关闭查询器
54     indexReader.close();
55     fsDirectory.close();
56 }
57 }

```

### ③具体流程

参考代码

### ④核心API

#### 1) 查询解析器QueryParser

QueryParser(单一字段的查询解析器)

```

1         // 创建查询解析器,两个参数: 默认要查询的字段名称, 分词器
2         QueryParser queryParser = new QueryParser("title", new
IKAnalyzer());
3         // 创建查询对象
4         Query query = queryParser.parse("谷歌");

```

MultiFieldQueryParser(多个字段的查询解析器)

```

1         //查询解析器 两个参数 要查询的字段和解析器
2         MultiFieldQueryParser multiFieldQueryParser = new
MultiFieldQueryParser(new String[]{"id","title"},
3         new IKAnalyzer());
4         //创建查询对象
5         Query query1 = multiFieldQueryParser.parse("1");

```

#### 2) 查询对象Query

通过查询解析器解析关键字, 然后获得查询对象

```

1         // 创建查询解析器,两个参数: 默认要查询的字段名称, 分词器

```

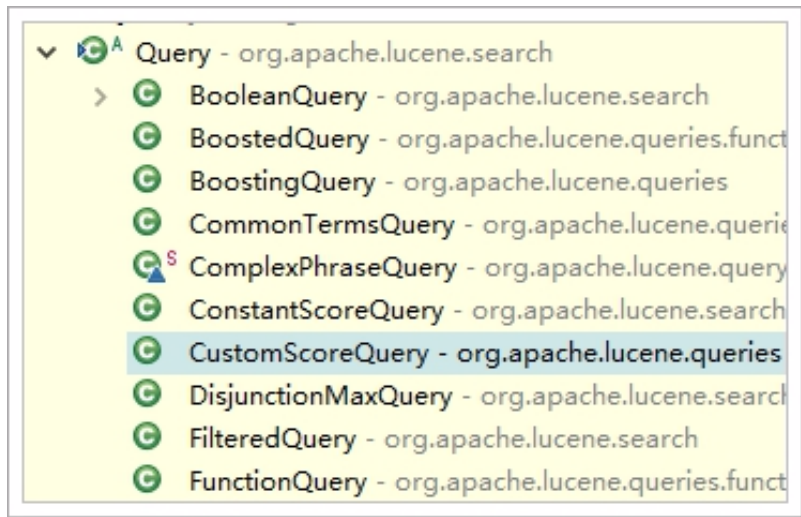


```

2      QueryParser queryParser = new QueryParser("title", new
IKAnalyzer());
3      // 创建查询对象
4      Query query = queryParser.parse("谷歌");

```

## 自定义查询对象(高级查询)



### 3) 索引搜索对象, 执行搜索功能(IndexSearcher)

快速搜索、排序、打分功能均可以由IndexSearch实现

IndexSearcher对象依赖于IndexReader对象来实现

```

1      //1 创建读取目录对象
2      FSDirectory fsDirectory = FSDirectory.open(new
File("C:\\indexDir"));
3      //2 创建索引读取工具
4      IndexReader indexReader = DirectoryReader.open(fsDirectory);
5      //3 创建索引搜索工具
6      IndexSearcher indexSearcher = new IndexSearcher(indexReader);

```

查询后的结果是打分排序后的前几名, 由第二个参数来决定

```

1      // 搜索数据,两个参数: 查询条件对象要查询的最大结果条数
2      // 返回的结果是 按照匹配度排名得分前N名的文档信息 (包含查询到的总条
数信息、所有符合条件的文档的编号信息)。
3      TopDocs topDocs = indexSearcher.search(query, 10);

```

#### 4) 查询结果对象(TopDocs)

TopDocs对象里有下面两个属性

```
1 int totalHits;//查询到的总条数
2 ScoreDoc[] scoreDocs; //得分文档对象的数组
```

```
1 TopDocs topDocs = indexSearcher.search(query, 10);
2     //7 各种操作
3     //7.1 获取返回的条数
4     System.out.println("本次搜索找到"+topDocs.totalHits+"条数据");
5     // 获取得分文档对象（ScoreDoc）数组.SocreDoc中包含：文档的编号、文
    档的得分
6     ScoreDoc[] scoreDocs = topDocs.scoreDocs;
```

#### 5) 得分文档对象(SocerDocs)

SocerDocs对象里有下面两个属性

```
1 int doc;//文档的编号----lucene给文档的一个唯一编号 拿到编号后，我们还需要根
    据编号来获取真正的文档信息
2 float score;//文档的得分信息
```

## 二、lucene的其他操作和高级使用

### #1 lucene的特殊查询操作

抽取查询方法

```
1     //[1]创建基本查询方法
2     public void Search(Query query) throws IOException {
3         FSDirectory fsDirectory = FSDirectory.open(new
    File("C:\\indexDir"));
4         IndexReader indexReader = DirectoryReader.open(fsDirectory);
```



```

5      IndexSearcher indexSearcher = new IndexSearcher(indexReader);
6      TopDocs topDocs = indexSearcher.search(query, 10);
7      System.out.println("本次搜索找到"+topDocs.totalHits+"条数据");
8      ScoreDoc[] scoreDocs = topDocs.scoreDocs;
9      for (ScoreDoc scoreDoc : scoreDocs) {
10         int docID = scoreDoc.doc;
11         Document doc = indexReader.document(docID);
12         System.out.print("文档id: " + doc.get("id"));
13         System.out.print("文档内容: " + doc.get("title"));
14         System.out.println("文档得分: " + scoreDoc.score);
15     }
16     indexReader.close();
17     fsDirectory.close();
18 }

```

## 1) TermQuery (词条查询)

```

1      /*
2      * 测试普通词条查询
3      * 注意: Term(词条)是搜索的最小单位, 不可再分词。值必须是字符串!
4      */
5      @Test
6      public void testTermQuery() throws Exception {
7          // 创建词条查询对象
8          Query query = new TermQuery(new Term("title", "谷歌地图")); // 此时查不到 谷歌地图可以分为谷歌 和 地图
9          Search(query);
10     }

```

适用场景: 在进行对不可再分的字段进行查询时, 如对id进行查询

## 2) WildcardQuery(通配符查询)

```

1      /*
2      * 测试通配符查询
3      * ? 可以代表任意一个字符
4      * * 可以任意多个任意字符
5      */
6      @Test

```

```

7      public void testWildcardQuery() throws Exception {
8          // 创建查询对象
9          Query query = new WildcardQuery(new Term("title", "*歌*"));
10         Search(query);
11     }

```

通配符查询可以只根据单个字符查询而不是通过词条 (term)

### 3) FuzzyQuery(模糊查询)

```

1      /*
2      * 测试模糊查询
3      */
4      @Test
5      public void testFuzzyQuery() throws Exception {
6          // 创建模糊查询对象:允许用户输错。但是要求错误的最大编辑距离不能超过
7          2
8          // 编辑距离: 一个单词到另一个单词最少要修改的次数 facebook -->
9          facebook 需要编辑1次, 编辑距离就是1
10         Query query = new FuzzyQuery(new Term("title", "fscevoool"));
11         // 可以手动指定编辑距离, 但是参数必须在0~2之间
12         Query query = new FuzzyQuery(new Term("title", "facevoool"), 1);
13         search(query);
14     }

```

模糊查询的“模糊”程度是有限的, 一旦过于“模糊”那么模糊查询无效。可以手动设定模糊距离, 如上述代码中模糊距离为1, 所以此时的字符串参数“facevoool” 的模糊距离为2 所以此时查询不到任何document

### 4) NumericRangeQuery(数值范围查询)

```

1      /*
2      * 测试: 数值范围查询
3      * 注意: 数值范围查询, 可以用来对非String类型的ID进行精确的查找
4      */
5      @Test
6      public void testNumericRangeQuery() throws Exception{
7          // 数值范围查询对象, 参数: 字段名称, 最小值、最大值、是否包含最小
8          值、是否包含最大值

```

```
8         Query query = NumericRangeQuery.newLongRange("id", 2L, 2L,
9             true, true);
10         Search(query);
    }
```

数值范围查询只可以对非String类型的字段进行查询

5) BooleanQuery(组合查询/布尔查询)

逻辑关系有如下几种

名称	含义
BooleanClause.Occur MUST	对于必须出现在匹配文档中的子句，使用此运算符。
BooleanClause.Occur MUST_NOT	对于不得出现在匹配文档中的子句使用此运算符。
BooleanClause.Occur SHOULD	对于应出现在匹配文档中的子句，使用此运算符。
BooleanClause.Occur FILTER	Like MUST except that these clauses do not participate in scoring.

逻辑关系组合效果如下

名称	含义
MUST和MUST	表示"与"的关系，结果为检索子句的交集
MUST和MUST_NOT	查询结果中不能包含MUST_NOT所对应的查询子句的检索结果
MUST_NOT和MUST_NOT	无意义，检索无结果
SHOULD和MUST	无意义，结果为MUST子句的检索结果
SHOULD和MUST_NOT	SHOULD功能同MUST，相当于MUST和MUST_NOT的检索结果
SHOULD和SHOULD	表示"或"的关系，结果为检索子句的并集

```
1      /*
2      * 布尔查询：
3      * 布尔查询本身没有查询条件，可以把其它查询通过逻辑运算进行组合
4      * 交集： Occur.MUST + Occur.MUST
5      * 并集： Occur.SHOULD + Occur.SHOULD
6      * 非： Occur.MUST_NOT
7      */
8      @Test
9      public void testBooleanQuery() throws Exception{
10
11          Query query1 = NumericRangeQuery.newLongRange("id", 1L, 3L,
```

```

    true, true);
12     Query query2 = NumericRangeQuery.newLongRange("id", 2L, 4L,
    true, true);
13     // 创建布尔查询的对象
14     BooleanQuery query = new BooleanQuery();
15     // 组合其它查询
16     query.add(query1, BooleanClause.Occur.MUST_NOT);
17     query.add(query2, BooleanClause.Occur.SHOULD);
18
19     Search(query);
20 }

```

布尔查询同时也可以进行嵌套，即一个布尔查询可以成为另一个布尔查询的条件语句。

## 6) PrefixQuery(前缀查询)

```

1  Term term = new Term("content", "This");
2  /**
3   * 查询content以This开头
4   */
5  PrefixQuery query=new PrefixQuery(term);

```

## 7) PhraseQuery(短语查询)

```

1  /**
2   * 查询标题为《Lucene...框架》，中间不超过5个字符
3   */
4  PhraseQuery phraseQuery=new PhraseQuery();
5  phraseQuery.add(new Term("title", "Lucene"));
6  phraseQuery.add(new Term("title", "框架"));
7  //之间的间隔最大不能超过5个
8  phraseQuery.setSlop(5);

```

setSlop 设置坡度表示两个字符串之间可以插入无关单词的个数

## 8) MatchAllDocsQuery(查询全部)

```
1 Query query=new MatchAllDocsQuery();
```

## #2 对索引的删改操作

### ①修改索引

基本步骤：

```
1 //1 创建文档存储目录
2 //2 创建索引写入器配置对象
3 //3 创建索引写入器
4 //4 创建文档数据
5 //5 修改
6 //6 提交
7 //7 关闭
```

```
1 package com.lcl.Update;
2
3 import org.apache.lucene.document.Document;
4 import org.apache.lucene.document.Field;
5 import org.apache.lucene.document.StringField;
6 import org.apache.lucene.document.TextField;
7 import org.apache.lucene.index.IndexWriter;
8 import org.apache.lucene.index.IndexWriterConfig;
9 import org.apache.lucene.index.Term;
10 import org.apache.lucene.store.FSDirectory;
11 import org.apache.lucene.util.Version;
12 import org.junit.Test;
13 import org.wltea.analyzer.lucene.IKAnalyzer;
14
15 import java.io.File;
16 import java.io.IOException;
17
18 public class IndexUpdateTest {
19     @Test
20     public void TestUpdate() throws IOException {
21         /* 测试：修改索引
```

```

22      * 注意:
23      *   A: Lucene修改功能底层会先删除, 再把新的文档添加。
24      *   B: 修改功能会根据Term进行匹配, 所有匹配到的都会被删除。这样不好
25      *   C: 因此, 一般我们修改时, 都会根据一个唯一不重复字段进行匹配修
    改。例如ID
26      *   D: 但是词条搜索, 要求ID必须是字符串。如果不是, 这个方法就不能
    用。
27      * 如果ID是数值类型, 我们不能直接去修改。可以先手动删除
    deleteDocuments(数值范围查询锁定ID), 再添加。
28      */
29      //1 创建文档存储目录
30      FSDirectory fsDirectory = FSDirectory.open(new
File("C:\\indexDir"));
31      //2 创建索引写入器配置对象
32      IndexWriterConfig indexWriterConfig = new
IndexWriterConfig(Version.LATEST, new IKAnalyzer());
33      //3 创建索引写入器
34      IndexWriter indexWriter = new
IndexWriter(fsDirectory, indexWriterConfig);
35      //4 创建文档数据
36      //[1] 创建新的文档数据
37      Document doc = new Document();
38      doc.add(new StringField("id", "1", Field.Store.YES));
39      doc.add(new TextField("title", "谷歌地图之父跳槽Alibaba ",
Field.Store.YES));
40      //5 修改
41      /* 修改索引。参数:
42      *   词条: 根据这个词条匹配到的所有文档都会被修改
43      *   文档信息: 要修改的新的文档数据
44      */
45      indexWriter.updateDocument(new Term("id", "1"), doc);
46      //6 提交
47      indexWriter.commit();
48      //7 关闭
49      indexWriter.close();
50  }
51 }

```

lucene的修改索引功能是基于某一字段进行查找之后修改, 一般会选择了一个不重复的字段进行匹配, 但是搜索词条时, 这个字段必须要是字符串, 如果不是字符串, 我们需要手动删除该document对象, 在创建一个新的, 用这样的方式进行修改。

## ②删除索引

### 基本步骤：

//1 创建文档对象目录

//2 创建索引写入器配置对象

//3 创建索引写入器

//4 删除

//5 提交

//6 关闭

👉代码

```
1 package com.lcl.Delete;
2
3 import org.apache.lucene.index.IndexWriter;
4 import org.apache.lucene.index.IndexWriterConfig;
5 import org.apache.lucene.store.Directory;
6 import org.apache.lucene.store.FSDirectory;
7 import org.apache.lucene.util.Version;
8 import org.junit.Test;
9 import org.wltea.analyzer.lucene.IKAnalyzer;
10
11 import java.io.File;
12
13 public class IndexDeleteTest {
14     /*
15      * 演示：删除索引
16      * 注意：
17      * 一般，为了进行精确删除，我们会根据唯一字段来删除。比如ID
18      * 如果是用Term删除，要求ID也必须是字符串类型！
19      */
20     @Test
21     public void testDelete() throws Exception {
22         // 创建目录对象
23         Directory directory = FSDirectory.open(new File("indexDir"));
24         // 创建配置对象
25         IndexWriterConfig conf = new IndexWriterConfig(Version.LATEST,
```

```

new IKAnalyzer());
26      // 创建索引写出工具
27      IndexWriter writer = new IndexWriter(directory, conf);
28      // 根据词条进行删除
29      //      writer.deleteDocuments(new Term("id", "1"));
30      // 根据query对象删除,如果ID是数值类型,那么我们可以用数值范围查询锁定一个具体的ID
31      //      Query query = NumericRangeQuery.newLongRange("id", 2L,
2L, true, true);
32      //      writer.deleteDocuments(query);
33      // 删除所有
34      writer.deleteAll();
35      // 提交
36      writer.commit();
37      // 关闭
38      writer.close();
39  }
40 }

```

## #3 lucene的高级使用

### ①高亮显示

```

1  package com.lcl;
2
3  import org.apache.lucene.document.Document;
4  import org.apache.lucene.index.DirectoryReader;
5  import org.apache.lucene.index.IndexReader;
6  import org.apache.lucene.queryparser.classic.QueryParser;
7  import org.apache.lucene.search.IndexSearcher;
8  import org.apache.lucene.search.Query;
9  import org.apache.lucene.search.ScoreDoc;
10 import org.apache.lucene.search.TopDocs;
11 import org.apache.lucene.search.highlight.Formatter;
12 import org.apache.lucene.search.highlight.Highlighter;
13 import org.apache.lucene.search.highlight.QueryScorer;
14 import org.apache.lucene.search.highlight.SimpleHTMLFormatter;
15 import org.apache.lucene.store.Directory;

```



```
16 import org.apache.lucene.store.FSDirectory;
17 import org.junit.Test;
18 import org.wltea.analyzer.lucene.IKAnalyzer;
19
20 import java.io.File;
21
22 public class HighLightTest {
23     // 高亮显示
24     @Test
25     public void testHighlighter() throws Exception {
26         // 目录对象
27         Directory directory = FSDirectory.open(new
File("C:\\indexDir"));
28         // 创建读取工具
29         IndexReader reader = DirectoryReader.open(directory);
30         // 创建搜索工具
31         IndexSearcher searcher = new IndexSearcher(reader);
32
33         QueryParser parser = new QueryParser("title", new
IKAnalyzer());
34         Query query = parser.parse("谷歌地图");
35         // 格式化器
36         Formatter formatter = new SimpleHTMLFormatter("<em>", "
</em>");
37         QueryScorer scorer = new QueryScorer(query);
38         // 准备高亮工具
39         Highlighter highlighter = new Highlighter(formatter, scorer);
40         // 搜索
41         TopDocs topDocs = searcher.search(query, 10);
42         System.out.println("本次搜索共" + topDocs.totalHits + "条数据");
43         ScoreDoc[] scoreDocs = topDocs.scoreDocs;
44         for (ScoreDoc scoreDoc : scoreDocs) {
45             // 获取文档编号
46             int docID = scoreDoc.doc;
47             Document doc = reader.document(docID);
48             System.out.println("id: " + doc.get("id"));
49             String title = doc.get("title");
50             // 用高亮工具处理普通的查询结果,参数: 分词器,要高亮的字段的名
称,高亮字段的原始值
51             String hTitle = highlighter.getBestFragment(new
IKAnalyzer(), "title", title);
52             System.out.println("title: " + hTitle);
```

```

53         // 获取文档的得分
54         System.out.println("得分: " + scoreDoc.score);
55     }
56 }
57 }

```

高亮显示的原理就是在查询字段中的命中部分前后添加一个<em></em>标签

## ②排序

```

1  // 排序
2  @Test
3  public void testSortQuery() throws Exception {
4      // 目录对象
5      Directory directory = FSDirectory.open(new File("indexDir"));
6      // 创建读取工具
7      IndexReader reader = DirectoryReader.open(directory);
8      // 创建搜索工具
9      IndexSearcher searcher = new IndexSearcher(reader);
10
11      QueryParser parser = new QueryParser("title", new
12      IKAnalyzer());
13
14      Query query = parser.parse("谷歌地图");
15
16      // 创建排序对象,需要排序字段SortField, 参数: 字段的名称、字段的类
17      // 型、是否反转如果是false, 升序。true降序
18      Sort sort = new Sort(new SortField("id", SortField.Type.LONG,
19      true));
20
21      // 搜索
22      TopDocs topDocs = searcher.search(query, 10, sort);
23      System.out.println("本次搜索共" + topDocs.totalHits + "条数据");
24
25      ScoreDoc[] scoreDocs = topDocs.scoreDocs;
26      for (ScoreDoc scoreDoc : scoreDocs) {
27          // 获取文档编号
28          int docID = scoreDoc.doc;
29          Document doc = reader.document(docID);
30          System.out.println("id: " + doc.get("id"));
31          System.out.println("title: " + doc.get("title"));
32      }
33  }

```

对document对象检索后进行排序后存在ScoreDoc数组中

### ③分页

```
1 // 分页
2 @Test
3 public void testPageQuery() throws Exception {
4     // 实际上Lucene本身不支持分页。因此我们需要自己进行逻辑分页。我们要
    准备分页参数:
5     int pageSize = 2; // 每页条数
6     int pageNum = 3; // 当前页码
7     int start = (pageNum - 1) * pageSize; // 当前页的起始条数
8     int end = start + pageSize; // 当前页的结束条数（不能包含）
9
10    // 目录对象
11    Directory directory = FSDirectory.open(new File("indexDir"));
12    // 创建读取工具
13    IndexReader reader = DirectoryReader.open(directory);
14    // 创建搜索工具
15    IndexSearcher searcher = new IndexSearcher(reader);
16
17    QueryParser parser = new QueryParser("title", new
    IKAnalyzer());
18    Query query = parser.parse("谷歌地图");
19
20    // 创建排序对象,需要排序字段SortField, 参数: 字段的名称、字段的类
    型、是否反转如果是false, 升序。true降序
21    Sort sort = new Sort(new SortField("id", Type.LONG, false));
22    // 搜索数据, 查询0~end条
23    TopDocs topDocs = searcher.search(query, end, sort);
24    System.out.println("本次搜索共" + topDocs.totalHits + "条数据");
25
26    ScoreDoc[] scoreDocs = topDocs.scoreDocs;
27    for (int i = start; i < end; i++) {
28        ScoreDoc scoreDoc = scoreDocs[i];
29        // 获取文档编号
30        int docID = scoreDoc.doc;
31        Document doc = reader.document(docID);
32        System.out.println("id: " + doc.get("id"));
33        System.out.println("title: " + doc.get("title"));
```

```
34     }
35 }
36
```

## 三、lucene的环境配置

### #1 lucene的依赖

#### Lucene快速入门—添加依赖

- 单元测试: junit
- 核心库: lucene-core
- 分词器: lucene-analyzers-common
- 查询解析器: lucene-queryparser
- 高亮显示: lucene-highlighter

[https://blog.csdn.net/weixin\\_42633131](https://blog.csdn.net/weixin_42633131)

#### 👉 maven依赖

```
1      <!-- lucene 4.10.2完整依赖示例 -->
2  <properties>
3      <lunece.version>4.10.2</lunece.version>
4  </properties>
5  <dependencies>
6      <dependency>
7          <groupId>junit</groupId>
8          <artifactId>junit</artifactId>
9          <version>4.12</version>
10     </dependency>
11     <!-- lucene核心库 -->
12     <dependency>
13         <groupId>org.apache.lucene</groupId>
14         <artifactId>lucene-core</artifactId>
15         <version>${lunece.version}</version>
```

```

16     </dependency>
17     <!-- Lucene的查询解析器 -->
18     <dependency>
19         <groupId>org.apache.lucene</groupId>
20         <artifactId>lucene-queryparser</artifactId>
21         <version>${lunece.version}</version>
22     </dependency>
23     <!-- lucene的默认分词器库 -->
24     <dependency>
25         <groupId>org.apache.lucene</groupId>
26         <artifactId>lucene-analyzers-common</artifactId>
27         <version>${lunece.version}</version>
28     </dependency>
29     <!-- lucene的高亮显示 -->
30     <dependency>
31         <groupId>org.apache.lucene</groupId>
32         <artifactId>lucene-highlighter</artifactId>
33         <version>${lunece.version}</version>
34     </dependency>
35 </dependencies>

```

## #2 IKAnalyzers分词器

```

1     <!-- ik分词器maven依赖-->
2     <dependency>
3         <groupId>com.janeluo</groupId>
4         <artifactId>ikanalyzer</artifactId>
5         <version>2012_u6</version>
6     </dependency>

```

## 四、lucene6.0测试结果

搜索关键字：手机

```
"C:\Program Files\Java\jdk1.8.0_05\bin\java.exe" ...
```

搜索用时：78ms

本次搜索找到8568条数据

文档id: 38086文档内容: 包邮正版 日光绘画 我是这样学会手机摄影的 手机摄影 书 一学就会的手机拍摄技巧 手机摄影  
文档id: 45568文档内容: 【送手机膜】Xiaomi/小米红手机米3S标准版全网通4G智能大屏手机文档得分: 18.935032  
文档id: 67976文档内容: 米熙运动手臂包手机包 跑步手机臂袋健身手腕包 户外手机臂套男女文档得分: 18.547817  
文档id: 52815文档内容: 法国PELLIOT跑步手机臂包女手腕包跑步装备男运动手机臂套手腕包文档得分: 18.063663  
文档id: 67362文档内容: 迷你运动健身跑步手机臂包女手腕包苹果6plus华为多功能手机包文档得分: 17.664705  
文档id: 10046文档内容: 赛鲸 懒人手机支架 床头支架多功能通用创意手机夹子桌面懒人神器文档得分: 17.044529  
文档id: 10168文档内容: Sony/索尼 F8332 XZ 手机 双卡双待 64G内存 智能手机文档得分: 17.044529  
文档id: 11417文档内容: 全新正品 Xiaomi/小米 小米手机5s Plus 双摄像头拍照智能手机文档得分: 17.044529  
文档id: 11418文档内容: Xiaomi/小米 小米手机5S 4g大屏智能超声波指纹解锁金属拍照手机文档得分: 17.044529  
文档id: 11427文档内容: Xiaomi/小米 小米手机5 全网通高配版 4g大屏智能指纹识别手机文档得分: 17.044529

## 搜索关键字：电脑

搜索用时：21ms

本次搜索找到572条数据

文档id: 24599文档内容: 公牛开关插座网线插座电脑网络插座面板电脑插座网络开关面板G07文档得分: 7.496553  
文档id: 33060文档内容: 阿雨生活 欧式实木电脑椅家用职员办公椅会议椅升降小电脑椅子文档得分: 7.496553  
文档id: 54176文档内容: Asus/华硕 小K -固态学生电脑商务办公游戏轻薄15.6寸笔记本电脑文档得分: 7.496553  
文档id: 64799文档内容: Targus/泰格斯绒面双肩包男女电脑休闲包欧美时尚电脑包TSB828文档得分: 7.496553  
文档id: 140844文档内容: 新品 费雪智玩学习小电脑（双语）CDG86 宝宝音乐小电脑玩具文档得分: 7.496553  
文档id: 10130文档内容: 守望先锋4G独显游戏电脑主机组装电脑LOL组装机DIY台式机非I5办公文档得分: 6.775917  
文档id: 11543文档内容: 蔓斯菲尔电脑桌 台式家用简约现代笔记本电脑桌简易书架办公桌文档得分: 6.775917  
文档id: 11611文档内容: 亿家达 简易电脑桌台式家用办公桌写字桌书桌 简约现代台式电脑桌文档得分: 6.775917  
文档id: 18480文档内容: 电脑桌台式桌家用简约现代办公桌简易小书桌笔记本电脑桌子写字台文档得分: 6.775917  
文档id: 28500文档内容: 瑞戈瑞士军刀男电脑双肩包16寸商务电脑包学生运动女背包文档得分: 6.775917

## 搜索关键字：零食

```
"C:\Program Files\Java\jdk1.8.0_05\bin\java.exe" ...
```

搜索用时：71ms

本次搜索找到2190条数据

文档id: 20141文档内容: 波奇网 宠物零食 柏可心狗零食100g 狗狗磨牙肉干零食训狗零食文档得分: 20.507929  
文档id: 47198文档内容: 怡亲 狗狗零食鸡翅洁齿棒宠物零食泰迪零食金毛零食狗狗磨牙棒文档得分: 20.507929  
文档id: 20022文档内容: 泰迪狗零食 训练零食 三明治800g 狗零食 宠物零食 狗狗零食除臭文档得分: 20.013046  
文档id: 35611文档内容: 怡亲 狗狗零食冻干鳕鱼肉30G宠物零食金毛零食泰迪零食成幼犬零食文档得分: 20.013046  
文档id: 35623文档内容: 怡亲狗狗零食鸡肉条切片6包宠物零食金毛零食泰迪零食成幼犬零食文档得分: 20.013046  
文档id: 11814文档内容: 猫零食路斯猫薄荷猫饼干鸡肉猫磨牙小鱼饼干幼猫零食宠物猫咪零食文档得分: 18.719627  
文档id: 20031文档内容: 泰迪狗零食 磨牙棒洁齿骨 奶骨800g 训练除臭狗零食 磨牙棒狗零食文档得分: 18.719627  
文档id: 21696文档内容: e-WEITA 猫零食 猫咪清洁牙齿零食 薄荷味颗粒零食 除臭 磨牙文档得分: 18.719627  
文档id: 30161文档内容: 良品铺子笋干零食泡椒竹笋干零食即食香辣脆笋片笋尖零食泡椒笋文档得分: 18.719627  
文档id: 35598文档内容: 猫零食包邮 怡亲幼猫零食切片小鱼干肉条猫磨牙肉干猫咪零食文档得分: 18.719627

# 五、lucene8.0测试结果

## 搜索关键字：手机



```
搜索用时: 125ms
本次搜索找到1284+ hits条数据
文档id: 38086文档内容: 包邮正版 日光绘画 我是这样学会手机摄影的 手机摄影 书 一学就会的手机拍摄技巧 手机摄影技巧
文档id: 67976文档内容: 米熙运动手臂包手机包 跑步手机臂袋健身手腕包 户外手机臂套男女文档得分: 9.106666
文档id: 45568文档内容: 【送手机膜】Xiaomi/小米红手机米3S标准版全网通4G智能大屏手机文档得分: 8.648654
文档id: 10046文档内容: 赛鲸 懒人手机支架 床头支架多功能通用创意手机夹子桌面懒人神器文档得分: 8.559173
文档id: 30547文档内容: 小米旗舰店正品手机自拍神器小米手机伸缩线控自拍杆即插即用神器文档得分: 8.440393
文档id: 138197文档内容: HTC M10H 联通公开版 智能手机 htc 10 骁龙820 手机文档得分: 8.440393
文档id: 52815文档内容: 法国PELLIOT跑步手机臂包女手臂包跑步装备男运动手机臂套手腕包文档得分: 8.350592
文档id: 10168文档内容: Sony/索尼 F8332 XZ 手机 双卡双待 64G内存 智能手机文档得分: 8.324865
文档id: 11417文档内容: 全新正品 Xiaomi/小米 小米手机5s Plus 双摄像头拍照智能手机文档得分: 8.212458
文档id: 30541文档内容: Xiaomi/小米 红米手机3S 大屏指纹解锁智能手机 金属机身超长待机文档得分: 8.212458
```

搜索关键字：电脑

```
搜索用时: 47ms
本次搜索找到572 hits条数据
文档id: 140844文档内容: 新品 费雪智玩学习小电脑（双语）CDG86 宝宝音乐小电脑玩具文档得分: 3.6055934
文档id: 33060文档内容: 阿雨生活 欧式实木电脑椅家用职员办公椅会议椅升降小电脑椅子文档得分: 3.5526161
文档id: 54176文档内容: Asus/华硕 小K -固态学生电脑商务办公游戏轻薄15.6寸笔记本电脑文档得分: 3.5011733
文档id: 24599文档内容: 公牛开关插座网线插座电脑网络插座面板电脑插座网络开关面板G07文档得分: 3.4511988
文档id: 64799文档内容: Targus/泰格斯绒面双肩包男女电脑休闲包欧美时尚电脑包TSB828文档得分: 3.4511988
文档id: 11611文档内容: 亿家达 简易电脑桌台式家用办公桌写字桌书桌 简约现代台式电脑桌文档得分: 3.402631
文档id: 28500文档内容: 瑞戈瑞士军刀男电脑双肩包16寸商务电脑包学生运动女背包文档得分: 3.402631
文档id: 37330文档内容: 亿家达电脑桌台式家用 电脑桌简约现代桌子简约书桌办公桌写字台文档得分: 3.355411
文档id: 149419文档内容: 电脑护目镜抗疲劳防辐射眼镜电脑镜男女款潮平光镜防蓝光平镜眼睛文档得分: 3.355411
文档id: 153099文档内容: 2016新款电脑双肩包男初中高中学生书包涂鸦印花防水潮电脑背包女文档得分: 3.355411

Process finished with exit code 0
```

搜索关键字：零食

```
搜索用时: 79ms
本次搜索找到1650+ hits条数据
文档id: 20022文档内容: 泰迪狗零食 训练零食 三明治800g 狗零食 宠物零食 狗狗零食除臭文档得分: 9.782759
文档id: 35611文档内容: 怡亲 狗狗零食冻干鳕鱼肉30G宠物零食金毛零食泰迪零食成幼犬零食文档得分: 9.64998
文档id: 35623文档内容: 怡亲狗狗零食鸡肉条切片6包宠物零食金毛零食泰迪零食成幼犬零食文档得分: 9.64998
文档id: 20141文档内容: 波奇网 宠物零食 柏可心狗零食100g 狗狗磨牙肉干零食训狗零食文档得分: 9.433445
文档id: 47198文档内容: 怡亲 狗狗零食鸡翅洁齿棒宠物零食泰迪零食金毛零食狗狗磨牙棒文档得分: 9.433445
文档id: 35631文档内容: 怡亲 狗狗零食高汤鸡心片40g*6支装宠物零食金毛零食泰迪幼犬零食文档得分: 9.204457
文档id: 21696文档内容: e-WEITA 猫零食 猫咪清洁牙齿零食 薄荷味颗粒零食 除臭 磨牙文档得分: 9.093368
文档id: 47539文档内容: 海狸先生扇贝即食零食香辣大连特产虾夷贝海鲜零食扇贝肉真空零食文档得分: 8.903575
文档id: 109913文档内容: 【零食礼包】喵在岛 7味零食 年货礼盒装海产零食大礼包 山东特产文档得分: 8.81162
文档id: 11814文档内容: 猫零食路斯猫薄荷猫饼干鸡肉猫磨牙小鱼饼干幼猫零食宠物猫咪零食文档得分: 8.721543
```

## 六、lucene6.0与lucene8.0对比

### #1测试环境与代码

Intelij IDEA、Maven、JDK 1.8、14w条天猫产品数据

代码如下：

POJO：Product类

```
1 package com.lcl.pojo;
2
3 public class Product {
4     //10001,房屋卫士自流平美缝剂瓷砖地砖专用双组份真瓷胶防水填缝剂镏金色,品质建
    材,398.00,上海,540785126782
5     int id;
6     String name;
7     String category;
8     float price;
9     String place;
10    String code;
11    public int getId() {
12        return id;
13    }
14    public void setId(int id) {
15        this.id = id;
16    }
17    public String getName() {
18        return name;
19    }
20    public void setName(String name) {
21        this.name = name;
22    }
23    public String getCategory() {
24        return category;
25    }
26    public void setCategory(String category) {
27        this.category = category;
28    }
29    public float getPrice() {
30        return price;
31    }
32    public void setPrice(float price) {
33        this.price = price;
34    }
35    public String getPlace() {
36        return place;
37    }
38    public void setPlace(String place) {
39        this.place = place;
40    }
41}
```



```

42     public String getCode() {
43         return code;
44     }
45     public void setCode(String code) {
46         this.code = code;
47     }
48     @Override
49     public String toString() {
50         return "Product [id=" + id + ", name=" + name + ", category="
+ category + ", price=" + price + ", place="+ place + ", code=" + code
+ "]\n";
51     }
52 }

```

Util工具类:

```

1  package com.lcl.Util;
2
3  import com.lcl.pojo.Product;
4  import org.apache.commons.io.FileUtils;
5
6  import java.io.File;
7  import java.io.IOException;
8  import java.util.ArrayList;
9  import java.util.List;
10
11 public class ProductUtil {
12
13     private static List<Product> file2list(String filename) throws
IOException {
14         File file = new File(filename);
15         List<String> lines = FileUtils.readLines(file, "UTF-8");
16         List<Product> products = new ArrayList<Product>();
17         for (String line : lines) {
18             Product product = line2product(line);
19             products.add(product);
20         }
21         return products;
22     }
23 //10001,房屋卫士自流平美缝剂瓷砖地砖专用双组份真瓷胶防水填缝剂镏金色,品质建

```

材,398.00,上海,540785126782

```
24     private static Product line2product(String line) {
25         String[] fields = line.split(",");
26         Product product = new Product();
27         product.setId(Integer.parseInt(fields[0]));
28         product.setName(fields[1]);
29         product.setCategory(fields[2]);
30         product.setPrice(Float.parseFloat(fields[3]));
31         product.setPlace(fields[4]);
32         product.setCode(fields[5]);
33         return product;
34     }
35     public static List<Product> listCreat() throws IOException {
36         String filename = "C:\\\\java\\data\\140k_products.txt";
37         List<Product> products = file2list(filename);
38         System.out.println(products.size());
39         return products;
40     }
41 }
```

索引创建类: IndexCreateStart类

```
1  package com.lcl.IndexCreat;
2
3  import com.lcl.Util.ProductUtil;
4  import com.lcl.ikAnalyzer.MyIkAnalyzer;
5  import com.lcl.pojo.Product;
6  import org.apache.lucene.analysis.Analyzer;
7  import org.apache.lucene.document.Document;
8  import org.apache.lucene.document.Field;
9  import org.apache.lucene.document.TextField;
10 import org.apache.lucene.index.IndexWriter;
11 import org.apache.lucene.index.IndexWriterConfig;
12 import org.apache.lucene.store.FSDirectory;
13 import org.junit.jupiter.api.Test;
14 import org.wltea.analyzer.lucene.IKAnalyzer;
15
16 import java.io.IOException;
17 import java.nio.file.Paths;
18 import java.util.ArrayList;
```

```
19 import java.util.Collection;
20 import java.util.List;
21
22
23
24 public class IndexCreateStart {
25
26     /**
27      * 创建Document对象
28      * @param p
29      * @return
30      * @throws IOException
31      */
32     private Document docCreat(Product p) throws IOException {
33         Document doc = new Document();
34         doc.add(new TextField("id", String.valueOf(p.getId()),
Field.Store.YES));
35         doc.add(new TextField("name", p.getName(), Field.Store.YES));
36         doc.add(new TextField("category", p.getCategory(),
Field.Store.YES));
37         doc.add(new TextField("price", String.valueOf(p.getPrice()),
Field.Store.YES));
38         doc.add(new TextField("place", p.getPlace(),
Field.Store.YES));
39         doc.add(new TextField("code", p.getCode(), Field.Store.YES));
40         return doc;
41     }
42
43     /**
44      * 批量添加集合创建
45      * @param document
46      */
47     private void docsCreat(List<Document> documents, Document
document){
48         documents.add(document);
49
50     }
51     @Test
52     public void indexCreat() throws IOException {
53         FSDirectory fsDirectory =
FSDirectory.open(Paths.get("C:\\java\\indexDir"));
54         // Analyzer myIkAnalyzer = new IKAnalyzer();
```

```

55     Analyzer myIkAnalyzer=new MyIkAnalyzer();
56     IndexWriterConfig conf = new IndexWriterConfig(myIkAnalyzer);
57     //TODO 下一步配置IndexWriter
58     List<Document> documents = new ArrayList<>();
59     IndexWriter writer = new IndexWriter(fsDirectory,
60     conf);
61     List<Product> products = ProductUtil.listCreat();
62     for (Product product : products) {
63         docsCreat(documents,docCreat(product));
64     }
65     System.out.println("documents长度为: "+documents.size());
66     writer.addDocuments(documents);
67     writer.commit();
68     writer.close();
69 }
70 }

```

## 查询测试类：QueryTest类

```

1  package com.lcl.QueryTest;
2
3  import com.lcl.ikAnalyzer.MyIkAnalyzer;
4  import org.apache.lucene.document.Document;
5  import org.apache.lucene.index.DirectoryReader;
6  import org.apache.lucene.index.IndexReader;
7  import org.apache.lucene.queryparser.classic.ParseException;
8  import org.apache.lucene.queryparser.classic.QueryParser;
9  import org.apache.lucene.search.IndexSearcher;
10 import org.apache.lucene.search.Query;
11 import org.apache.lucene.search.ScoreDoc;
12 import org.apache.lucene.search.TopDocs;
13 import org.apache.lucene.store.FSDirectory;
14 import org.junit.jupiter.api.Test;
15
16 import java.io.IOException;
17 import java.nio.file.Paths;
18
19 public class QueryTest01 {
20     @Test

```

```
21     public void QueryTest1() throws IOException, ParseException {
22         //1 创建读取目录对象
23         FSDirectory fsDirectory =
24         FSDirectory.open(Paths.get("C:\\java\\indexDir"));
25         //2 创建索引读取工具
26         IndexReader indexReader = DirectoryReader.open(fsDirectory);
27         //3 创建索引搜索工具
28         IndexSearcher indexSearcher = new IndexSearcher(indexReader);
29         //4 创建查询解析器
30         // 创建查询解析器,两个参数: 默认要查询的字段名称, 分词器
31         QueryParser queryParser = new QueryParser("name", new
32         MyIkAnalyzer());
33         //5 创建查询对象
34         Query query = queryParser.parse("中国");
35         //6 搜索数据
36         // 搜索数据,两个参数: 查询条件对象要查询的最大结果条数
37         // 返回的结果是 按照匹配度排名得分前N名的文档信息 (包含查询到的总条
38         数信息、所有符合条件的文档的编号信息)。
39         long t1 = System.currentTimeMillis();
40         TopDocs topDocs = indexSearcher.search(query, 10);
41         long t2 = System.currentTimeMillis();
42         System.out.println("搜索用时: " + (t2 - t1) + "ms");
43         //7 各种操作
44         //7.1 获取返回的条数
45         System.out.println("本次搜索找到" + topDocs.totalHits + "条数据");
46         // 7.2 获取得分文档对象 (ScoreDoc) 数组。ScoreDoc中包含: 文档的编
47         号、文档的得分
48         ScoreDoc[] scoreDocs = topDocs.scoreDocs;
49         // 7.3 获取内容
50         for (ScoreDoc scoreDoc : scoreDocs) {
51             //获取ID
52             int docID = scoreDoc.doc;
53             //根据ID去找文档
54             Document doc = indexReader.document(docID);
55             System.out.print("文档id: " + doc.get("id"));
56             System.out.print("文档内容: " + doc.get("name"));
57             //获取得分
58             System.out.println("文档得分: " + scoreDoc.score);
59         }
60         //7.4 关闭查询器
61         indexReader.close();
62         fsDirectory.close();
63     }
```

```
59     }  
60 }
```

## #2 索引建立效率对比

lucene 6.0






✓ Tests passed: 1 of 1 test – 28 s 496 ms

147939

documents长度为: 147939

lucene6.0 索引经过21669ms建立成功

文件目录下的索引文件如下

 _0.cfe	2019/4/30 13:43	CFE 文件	1 KB
 _0.cfs	2019/4/30 13:43	CFS 文件	22,601 KB
 _0.si	2019/4/30 13:43	SI 文件	1 KB
 segments_1	2019/4/30 13:43	文件	1 KB
 write.lock	2019/4/30 13:42	LOCK 文件	0 KB

lucene 8.0

✓ Tests passed: 1 of 1 test – 23 s 910 ms






"C:\Program Files\Java\jdk1.8.0\_05"

147939

documents长度为: 147939

lucene6.0 索引经过20806ms建立成功

文件目录如下

 _0.cfe	2019/4/30 13:47	CFE 文件	1 KB
 _0.cfs	2019/4/30 13:47	CFS 文件	22,705 KB
 _0.si	2019/4/30 13:47	SI 文件	1 KB
 segments_1	2019/4/30 13:47	文件	1 KB
 write.lock	2019/4/30 13:46	LOCK 文件	0 KB

多次运行之后所需时间保持在一定范围内，基本与上图结果相近。

### #3 查询效率对比

使用了普通的查询方式，效率对比如[lucene#4 lucene6.0实测](#)与[lucene#5 lucene8.0实测](#)中所示，在当前电脑配置和相同14w条数据的环境下，lucene6.0的查询效率相对快一些。

经过查阅资料 lucene 8.0

#### 查询执行

新版本对术语查询、短语查询和布尔查询进行了优化，在不需要总点击数的时候，可以有效地跳过不具竞争性的文档。根据实际查询和数据分布的不同，查询的速度会在慢几个百分点和快几倍之间变化，对术语查询和纯粹析取（pure disjunctions）来说更是如此。

- `TopDocs.totalHits` 现在是一个能给出实际点击数下限的对象。
- `IndexSearcher` 的 `search` 和 `searchAfter` 方法可以精确地计算总点击数，现在最大值只能可达 1,000，以便查询优化的默认启用。
- 现在需要查询才能产生非负分数。

查询速度在lucene8已经有提升，但是可能在当前数据分布的情况下不能体现。

### #4 其他8.0新特性

#### 编解码器

- 发布当下的索引得分，会影响到附带的跳过数据（skip data）。这也是在点击计数不需要的情况下，术语查询对热门点击集合进行优化的方式。
- Doc 值引入了跳转表，以便可以不中断地运行，这对稀疏字段来说有很大作用。
- 现在术语索引 `FST` 会在堆外加载使用 `MMapDirectory` 的非主键字段，从而减少这些字段的堆使用。

## 自定义评分

新的 `FeatureField` 允许将诸如 pagerank 的静态特征有效地集成到分数中，并且新的 `LongPoint#newDistanceFeatureQuery` 和 `LatLonPoint#newDistanceFeatureQuery` 方法可以分别从新近度 (recency) 和地理距离来提升分数。这些新帮手针对不需要总点击数的情况进行了优化。举个例子，如果 pagerank 在您的分数中有着高的权重，那么 Lucene 往往能跳过低 pagerank 值的文档。

来源: <http://www.linuxeden.com/a/46025>

## 七、出现的异常错误及解决方法

#1 在 lucene5.0 之后 `FSDirectory.open()` 中 `open` 的参数是 NIO 下的 `Path` 类而不是原来 IO 下的 `File` 类，应采用如下解决方式

```
1 //5.0版本以前
2 String path = " ... ";
3 directory = FSDirectory.open(new File(path));
```

```
1 //5.0版本以后
2 String path = " ... ";
3 directory = FSDirectory.open(Paths.get(path));
```

#2 在使用 lucene6.0 时 配置 `IndexWriterConfig` 时不再如同以前一样需要在构造方法中配置 `Version`

lucene



而 IndexWriterConfig 6.6 中

```
//无参构造方法
public IndexWriterConfig() {
    this(new StandardAnalyzer());
}
//有参构造方法
public IndexWriterConfig(Analyzer analyzer) {
    super(analyzer);
    this.writer = new SetOnce();
}
```

可以看出，在 6.6 版本中 version 不再是必要的，并且，存在无参构造方法，可以直接使用默认的 StandardAnalyzer 分词器。

### #3 关于IKAnalyzer

在lucene6.0中，用老版本的IK分词器会报错

```
java.lang.AbstractMethodError: org.apache.lucene.analysis.Analyzer.createComponents(Ljava/lang/String;)Lorg/a

at org.apache.lucene.analysis.Analyzer.tokenStream(Analyzer.java:176)
at org.apache.lucene.document.Field.tokenStream(Field.java:570)
at org.apache.lucene.index.DefaultIndexingChain$PerField.invert(DefaultIndexingChain.java:708)
at org.apache.lucene.index.DefaultIndexingChain.processField(DefaultIndexingChain.java:417)
at org.apache.lucene.index.DefaultIndexingChain.processDocument(DefaultIndexingChain.java:373)
at org.apache.lucene.index.DocumentsWriterPerThread.updateDocuments(DocumentsWriterPerThread.java:272)
at org.apache.lucene.index.DocumentsWriter.updateDocuments(DocumentsWriter.java:412)
at org.apache.lucene.index.IndexWriter.updateDocuments(IndexWriter.java:1333)
at org.apache.lucene.index.IndexWriter.addDocuments(IndexWriter.java:1312)
at com.lcl.IndexCreat.CreateIndexStart.indexCreat(IndexCreatStart.java:65) <24 internal calls>
at java.util.ArrayList.forEach(ArrayList.java:1234) <13 internal calls>
at java.util.ArrayList.forEach(ArrayList.java:1234) <25 internal calls>
```

原因在于lucene6与IK此时版本并不匹配，而且IK分词器已经太久没更新，所以我们需要自己对IK分词器进行修改

👉代码如下

新建两个类

MyIKTokenizer.java

MyIkAnalyzer.java

```
1 import java.io.IOException;
2 import java.io.Reader;
3
4 import org.apache.lucene.analysis.Tokenizer;
5 import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;
6 import org.apache.lucene.analysis.tokenattributes.OffsetAttribute;
```

```

7 import org.apache.lucene.analysis.tokenattributes.TypeAttribute;
8 import org.wltea.analyzer.core.IKSegmenter;
9 import org.wltea.analyzer.core.Lexeme;
10
11 public class MyIKTokenizer extends Tokenizer {
12     // IK分词器实现
13     private IKSegmenter _IKImplement;
14
15     // 词元文本属性
16     private final CharTermAttribute termAtt;
17     // 词元位移属性
18     private final OffsetAttribute offsetAtt;
19     // 词元分类属性（该属性分类参考org.wltea.analyzer.core.Lexeme中的分类
    常量）
20     private final TypeAttribute typeAtt;
21     // 记录最后一个词元的结束位置
22     private int endPosition;
23
24     public MyIKTokenizer(Reader in) {
25         this(in, false);
26     }
27
28     public MyIKTokenizer(Reader in, boolean useSmart) {
29         offsetAtt = addAttribute(OffsetAttribute.class);
30         termAtt = addAttribute(CharTermAttribute.class);
31         typeAtt = addAttribute(TypeAttribute.class);
32         _IKImplement = new IKSegmenter(input, useSmart);
33     }
34
35     @Override
36     public boolean incrementToken() throws IOException {
37         // 清除所有的词元属性
38         clearAttributes();
39         Lexeme nextLexeme = _IKImplement.next();
40         if (nextLexeme != null) {
41             // 将Lexeme转成Attributes
42             // 设置词元文本
43             termAtt.append(nextLexeme.getLexemeText());
44             // 设置词元长度
45             termAtt.setLength(nextLexeme.getLength());
46             // 设置词元位移
47             offsetAtt.setOffset(nextLexeme.getBeginPosition(),

```

```

48         nextLexeme.getEndPosition());
49         // 记录分词的最后位置
50         endPosition = nextLexeme.getEndPosition();
51         // 记录词元分类
52         typeAtt.setType(nextLexeme.getLexemeTypeString());
53         // 返会true告知还有下个词元
54         return true;
55     }
56     // 返会false告知词元输出完毕
57     return false;
58 }
59
60 public void reset() throws IOException {
61     super.reset();
62     _IKImplement.reset(input);
63 }
64
65 @Override
66 public final void end() {
67     // set final offset
68     int finalOffset = correctOffset(this.endPosition);
69     offsetAtt.setOffset(finalOffset, finalOffset);
70 }
71
72 }

```

```

1  import java.io.Reader;
2  import java.io.StringReader;
3
4  import org.apache.lucene.analysis.Analyzer;
5  import org.apache.lucene.util.IOUtils;
6
7  public class MyIkAnalyzer extends Analyzer {
8
9      @Override
10     protected TokenStreamComponents createComponents(String arg0) {
11         Reader reader=null;
12         try{
13             reader=new StringReader(arg0);
14             MyIKTokenizer it = new MyIKTokenizer(reader);

```

```

15         return new Analyzer.TokenStreamComponents(it);
16     }finally {
17         IOUtils.closeWhileHandlingException(reader);
18     }
19 }
20
21 }

```

此时如果代码报错如下

```

java.lang.AssertionError: TokenStream implementation classes or at least their incrementToken() implementation
    at org.apache.lucene.analysis.TokenStream.assertFinal(TokenStream.java:123)
    at org.apache.lucene.analysis.TokenStream.<init>(TokenStream.java:99)
    at org.apache.lucene.analysis.Tokenizer.<init>(Tokenizer.java:45)
    at com.lcl.ikAnalyzer.MyIKTokenizer.<init>(MyIKTokenizer.java:30)
    at com.lcl.ikAnalyzer.MyIKTokenizer.<init>(MyIKTokenizer.java:27)
    at com.lcl.ikAnalyzer.MyIkAnalyzer.createComponents(MyIkAnalyzer.java:16)
    at org.apache.lucene.analysis.Analyzer.tokenStream(Analyzer.java:176)
    at org.apache.lucene.document.Field.tokenStream(Field.java:570)
    at org.apache.lucene.index.DefaultIndexingChain$PerField.invert(DefaultIndexingChain.java:708)
    at org.apache.lucene.index.DefaultIndexingChain.processField(DefaultIndexingChain.java:417)
    at org.apache.lucene.index.DefaultIndexingChain.processDocument(DefaultIndexingChain.java:373)
    at org.apache.lucene.index.DocumentsWriterPerThread.updateDocuments(DocumentsWriterPerThread.java:272)
    at org.apache.lucene.index.DocumentsWriter.updateDocuments(DocumentsWriter.java:412)
    at org.apache.lucene.index.IndexWriter.updateDocuments(IndexWriter.java:1333)
    at org.apache.lucene.index.IndexWriter.addDocuments(IndexWriter.java:1312)
    at com.lcl.IndexCreat.IndexCreateStart.indexCreat(IndexCreateStart.java:65) <24 internal calls>
    at java.util.ArrayList.forEach(ArrayList.java:1234) <13 internal calls>

```

那么我们需要在MyIKTokenizer类前用final修饰。

#4 在使用lucene6.0所建立索引用8.0进行查询会报错如下

```

C:\Program Files\Java\jdk1.8.0_05\bin\java.exe ...
org.apache.lucene.index.IndexFormatTooOldException: Format version is not supported (resource BufferedChecksum
    at org.apache.lucene.codecs.CodecUtil.checkHeaderNoMagic(CodecUtil.java:213)
    at org.apache.lucene.index.SegmentInfos.readCommit(SegmentInfos.java:305)
    at org.apache.lucene.index.SegmentInfos.readCommit(SegmentInfos.java:289)
    at org.apache.lucene.index.StandardDirectoryReader$1.doBody(StandardDirectoryReader.java:61)
    at org.apache.lucene.index.StandardDirectoryReader$1.doBody(StandardDirectoryReader.java:58)
    at org.apache.lucene.index.SegmentInfos$FindSegmentsFile.run(SegmentInfos.java:680)
    at org.apache.lucene.index.StandardDirectoryReader.open(StandardDirectoryReader.java:81)
    at org.apache.lucene.index.DirectoryReader.open(DirectoryReader.java:63)
    at com.lcl.QueryTest.QueryTest01.QueryTest1(QueryTest01.java:25) <24 internal calls>
    at java.util.ArrayList.forEach(ArrayList.java:1234) <13 internal calls>
    at java.util.ArrayList.forEach(ArrayList.java:1234) <25 internal calls>

```

lucene 8.0不能使用lucene 6.0所创建的索引文件，因不兼容而产生错误