lucene&solr

**第一天**

# 学习目标

* 能够领会Lucene基本原理
* 能够归纳索引库实现原理
* 能够运用实现索引流程的代码
* 能够运用实现检索流程的代码
* 能够描述Lucene分词器
* 能够应用第三方IK分词器

# 搜索介绍

## 搜索实现方案

### 传统实现方案

说明：根据用户输入的搜索关键词（java），应用服务器通过sql语句查询数据库，数据库服务器把查询结果响应给应用服务器，应用服务器把查询结果响应给用户。

特点：用户量大，数据量很大，数据库服务器压力非常大，查询速度慢！！！

1.请求搜索java

应用服务器

（tomcat）

controller

serivce

dao

数据库：mysql

2.select \* from table

where name like ‘%java%’

3.数据库返回查询结果给应用服务器

4.应用服务器把查询结果响应给用户

### lucene实现方案

说明：根据用户输入的搜索关键词（java），应用服务器通过lucene的API搜索索引库，索引库把搜索结果响应应用服务器，应用服务器再把搜索结果响应给用户。

特点：解决用户量大，数据量很大，系统对搜索速度要求高的业务需求（实时查询）。

1.请求搜索java

应用服务器：tomcat

controller

service

dao

索引库

2.通过lucene提供的API搜索索引库

3.索引库返回搜索结果给应用服务器

4.应用服务器再把搜索结果响应给用户

## 数据查询方法

### 顺序扫描法

举个例子：比如我们有大量的文件，文件编号从A，B，C。。。。。。

需求：要找出文件内容中包含有java的所有文件

需求实现：从A文件开始查找，再找B文件，然后再找C文件，以此类推。。。。。

特点：如果文件数量很多，查找速度慢！！！

### 倒排索引法【重点】

举个例子：使用新华字典查找汉字，先找到汉字的偏旁部首，再根据偏旁部首对应的目录（索引）找到目标汉字。

以lucene为例建立倒排索引：

文件一(编号0)：we like java java java

文件二（编号1）：we like lucene lucene lucene

|  |  |  |
| --- | --- | --- |
| Term | (Doc，Freq) | (Pos) |
| we | (0，1) (1，1) | 1. (0) |
| like | (0，1) (1，1) | 1. (1) |
| java | (0，3) | (2，3，4) |
| lucene | (1，3) | (2，3，4) |

说明：

1. 建立倒排索引，就是建立词语与文件的对应关系（词语在什么文件出现，出现了几次，在什么位置出现）
2. 搜索的时候，直接根据搜索关键词（java），在倒排索引中找到目标内容。

## 搜索技术应用场景

1. 单机软件搜索（idea）
2. 站内搜索（京东，淘宝）
3. 垂直搜索（限定行业，比如教育，医疗搜索）
4. 平台搜索（百度，360，搜狗）

# lucene介绍

## lucene是什么

lucene是apache软件基金会下的一个子项目。是一个成熟、免费、开放源代码的全文检索引擎工具包。提供了一套简单易用的API，方便在目标系统中实现全文检索功能。目前已经有很多应用系统的搜索功能是基于lucene来实现。比如eclipse帮助系统的搜索功能。

lucene能够为文本类型的数据建立索引，只需要把数据转换成文本格式，lucene就可以对文档进行索引和搜索。比如常见的word文档、html文档、pdf文档。首先将文档内容转换成文本格式，交给lucene进行索引，把建立好的索引保存在硬盘或者内存中。然后根据用户输入的查询条件，在索引文件中查找，返回查询结果给用户。

## 全文检索是什么

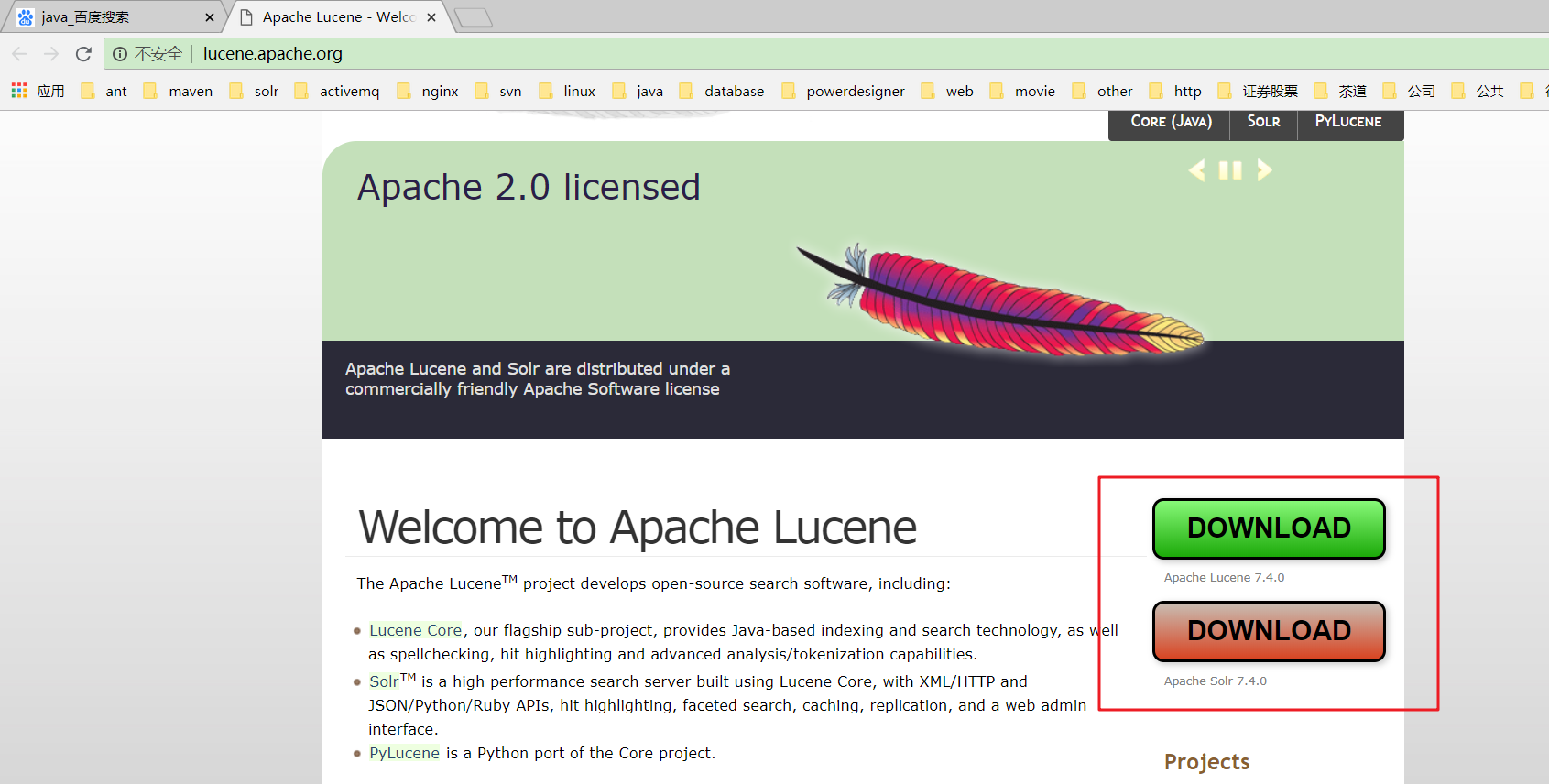
1. 索引流程：计算机通过索引程序扫描文件中的每一个词语，建立词语与文件的对应关系
2. 检索流程：计算机通过检索程序，根据搜索关键词，在索引库查找目标内容

## lucene与搜索引擎的区别

lucene是一个全文检索引擎工具包，相当于汽车发动机；搜索引擎基于全文检索实现，是一个可以独立运行的软件产品，相当于汽车。

## lucene官方网站

官方网站：http://lucene.apache.org/



# 全文检索流程介绍

## 索引和检索流程图

原始数据

索引流程：

检索流程：

1.采集数据

2.建立文档对象

3.分析文档

4.建立索引

索引库

1.用户查询

2.建立查询对象

3.执行查询

4.返回查询结果

## 索引流程详细介绍

### 原始数据

1. 保存在关系数据库中的业务数据
2. 保存在文件中的数据
3. 网络上的网页文件数据

### 采集数据（获取文档）

1. 通过JDBC操作获取到关系数据库中的业务数据
2. 通过IO流获取文件上的数据
3. 通过爬虫（蜘蛛）程序获取网络上的网页数据

信息采集的开源软件：

Solr（<http://lucene.apache.org/solr>） ，solr是apache的一个子项目，支持从关系数据库、xml文档中提取原始数据。

Nutch（<http://lucene.apache.org/nutch>）, Nutch是apache的一个子项目，包括大规模爬虫工具，能够抓取和分辨web网站数据。

jsoup（http://jsoup.org/ ），jsoup 是一款Java 的HTML解析器，可直接解析某个URL地址、HTML文本内容。它提供了一套非常省力的API，可通过DOM，CSS以及类似于jQuery的操作方法来取出和操作数据。

### 建立文档对象【重点】

说明：文档对象（Document），一个文档对象包含有多个域（Field）。一个文档对象就相当于关系数据库表中的一条记录，一个域就相当于一个字段。

文档对象（Document）

域（Field）

name：bookId

value：1

域（Field）

name：bookName

value：java 编程思想

域（Field）

name：bookPrice

value：80f

域（Field）

..........

结构化数据：长度固定，格式固定

非结构化数据：长度不固定，格式不固定（文档对象是非结构化数据）

### 分析文档对象

把原始数据，转换成文档对象后，使用分析器（分词器）把文档域中的数据切分成一格一格词语。为后续建立索引做准备。

### 建立索引

建立词语和文档的对应关系，词语在什么文档出现，出现了几次，在什么位置出现（倒排索引）。并且保存到索引库。

## 检索流程详细介绍

### 用户

用户可以是自然人，也可以是程序。

### 用户查询

用户在搜索入口界面，输入搜索关键词，执行搜索。



### 建立查询对象【重点】

说明：根据用户输入的搜索关键词，使用分析器分词以后，建立查询对象（Query），Query对象会生成具体的查询语法。bookName:java，表示搜索图书名称域中包含有java的图书。

### 执行搜索

根据查询对象（Query），和Query生成的语法，在索引库中查询目标内容。

### 返回查询结果

提供一个搜索结果页面，把搜索结果友好的展示给用户（搜索关键词是高亮显示，搜索结果有排序）



# lucene入门程序

## 准备环境

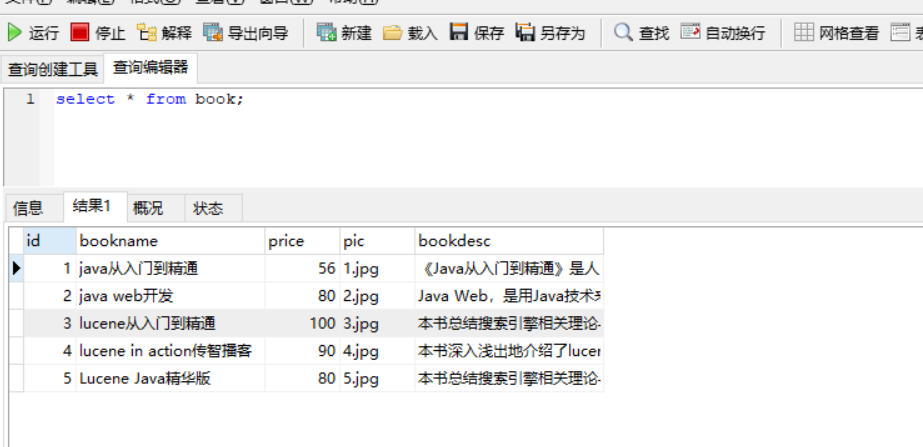
jdk：1.8

ide：idea

数据库：mysql

lucene：4.10.3

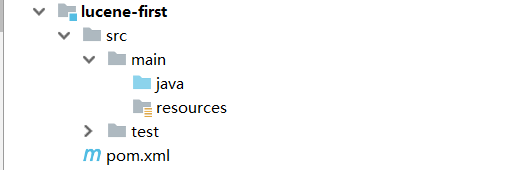
## 准备数据



## 准备lucene



## 创建项目



## 配置pom.xml，导入依赖包

* 导入lucene依赖包
* 数据库驱动包

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>cn.itheima</**groupId**>  <**artifactId**>lucene-first</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>   <**packaging**>jar</**packaging**>   <**properties**>  <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  *<!-- mysql版本 -->* <**mysql.version**>5.1.30</**mysql.version**>  *<!-- lucene版本 -->* <**lucene.version**>4.10.3</**lucene.version**>  </**properties**>   <**dependencies**>  *<!-- mysql数据库依赖 -->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>${mysql.version}</**version**>  </**dependency**>  *<!--lucene依赖包 -->* <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-core</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-analyzers-common</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-queryparser</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  <**scope**>test</**scope**>  </**dependency**>  </**dependencies**>   </**project**> |

## 原始数据采集准备

### 编写图书实体类

|  |
| --- |
| **package** cn.itheima.po;  */\*\*  \* 图书实体类  \*/* **public class** Book {   **private** Integer **id**;  **private** String **bookName**;  **private** Float **price**;  **private** String **pic**;  **private** String **bookDesc**;   **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   **public** String getBookName() {  **return bookName**;  }   **public void** setBookName(String bookName) {  **this**.**bookName** = bookName;  }   **public** Float getPrice() {  **return price**;  }   **public void** setPrice(Float price) {  **this**.**price** = price;  }   **public** String getPic() {  **return pic**;  }   **public void** setPic(String pic) {  **this**.**pic** = pic;  }   **public** String getBookDesc() {  **return bookDesc**;  }   **public void** setBookDesc(String bookDesc) {  **this**.**bookDesc** = bookDesc;  }   **public** String toString() {  **return "Book{"** +  **"id="** + **id** +  **", bookName='"** + **bookName** + **'\''** +  **", price="** + **price** +  **", pic='"** + **pic** + **'\''** +  **", bookDesc='"** + **bookDesc** + **'\''** +  **'}'**;  } } |

### 编写图书dao接口

|  |
| --- |
| **package** cn.itheima.dao;  **import** cn.itheima.po.Book;  **import** java.util.List;  */\*\*  \* 图书dao接口  \*/* **public interface** BookDao {   */\*\*  \* 查询全部图书列表  \*/* List<Book> findAllBooks(); } |

### 编写图书dao实现类

|  |
| --- |
| **package** cn.itheima.dao.impl;  **import** cn.itheima.dao.BookDao; **import** cn.itheima.po.Book;  **import** java.sql.Connection; **import** java.sql.DriverManager; **import** java.sql.PreparedStatement; **import** java.sql.ResultSet; **import** java.util.ArrayList; **import** java.util.List;  */\*\*  \* 图书dao实现类  \*/* **public class** BookDaoImpl **implements** BookDao {   */\*\*  \* 查询全部图书列表  \*/* **public** List<Book> findAllBooks() {  *// 定义结果集集合list* List<Book> bookList = **new** ArrayList<Book>();   Connection con = **null**;  PreparedStatement psmt =**null**;  ResultSet rs = **null**;  **try**{  *// 1.加载驱动* Class.*forName*(**"com.mysql.jdbc.Driver"**);  *// 2.创建数据库连接对象* con = DriverManager  .*getConnection*(**"jdbc:mysql://127.0.0.1:3306/82\_lucene"**, **"root"**, **"admin"**);  *// 3.编写sql* String sql = **"select** *\** **from book"**;  *// 4.创建statement语句对象* psmt = con.prepareStatement(sql);  *// 5.设置参数   // 6.执行* rs = psmt.executeQuery();  *// 7.处理结果集* **while**(rs.next()){  *// 创建图书对象* Book book = **new** Book();   *// 图书Id* book.setId(rs.getInt(**"id"**));  *// 图书名称* book.setBookName(rs.getString(**"bookname"**));  *// 图书价格* book.setPrice(rs.getFloat(**"price"**));  *// 图书图片* book.setPic(rs.getString(**"pic"**));  *// 图书描述* book.setBookDesc(rs.getString(**"bookdesc"**));   bookList.add(book);  }    }**catch**(Exception e){  e.printStackTrace();  }**finally** {  *// 8.释放资源* **try**{  **if**(rs != **null**) rs.close();  **if**(psmt != **null**) psmt.close();  **if**(con != **null**) con.close();  }**catch**(Exception e){  e.printStackTrace();  }   }   **return** bookList;  }   } |

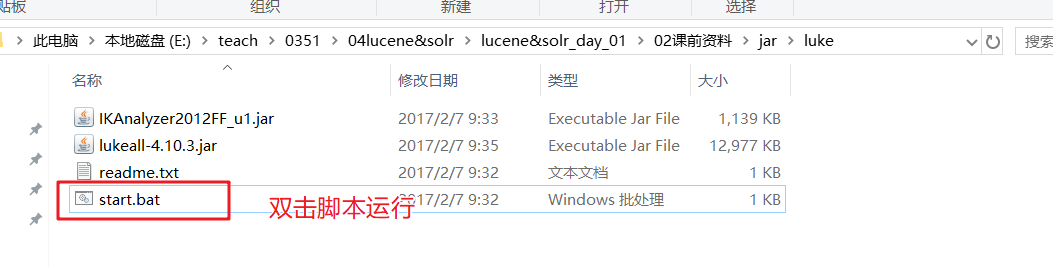
## 索引流程实现

1. 采集数据
2. 建立文档对象（Document）
3. 建立分析器（Analyzer），用于分词
4. 建立索引库配置对象（IndexWriterConfig），配置索引库
5. 建立索引库目录对象（Directory），指定索引库的位置
6. 建立索引库操作对象（IndexWriter），把文档对象写入索引库
7. 释放资源

|  |
| --- |
| **package** cn.itheima.index;   **import** cn.itheima.dao.BookDao; **import** cn.itheima.dao.impl.BookDaoImpl; **import** cn.itheima.po.Book; **import** org.apache.lucene.analysis.Analyzer; **import** org.apache.lucene.analysis.standard.StandardAnalyzer; **import** org.apache.lucene.document.Document; **import** org.apache.lucene.document.Field; **import** org.apache.lucene.document.TextField; **import** org.apache.lucene.index.IndexWriter; **import** org.apache.lucene.index.IndexWriterConfig; **import** org.apache.lucene.store.Directory; **import** org.apache.lucene.store.FSDirectory; **import** org.apache.lucene.util.Version; **import** org.junit.Test;  **import** java.io.File; **import** java.util.ArrayList; **import** java.util.List;  */\*\*  \* 索引管理类  \*/* **public class** IndexManager {   */\*\*  \* 索引流程实现  \*/* @Test  **public void** createIndex() **throws** Exception{ *// 1.采集数据* BookDao bookDao = **new** BookDaoImpl();  List<Book> bookList = bookDao.findAllBooks();  *// 2.建立文档对象（Document）* List<Document> documentList = **new** ArrayList<Document>();  **for**(Book book:bookList){  *//创建文档对象* Document doc = **new** Document();   */\*\*  \* 给文档对象添加域  \* 方法：add（）  \* 参数：TextField  \* TextField参数：  \* 参数一：域的名称  \* 参数二：域值  \* 参数三：指定是否把域值保存到文档对象中  \*/  // 图书Id* doc.add(**new** TextField(**"bookId"**,book.getId()+**""**, Field.Store.***YES***));   *// 图书名称* doc.add(**new** TextField(**"bookName"**,book.getBookName(), Field.Store.***YES***));  *// 图书价格* doc.add(**new** TextField(**"bookPrice"**,book.getPrice()+**""**, Field.Store.***YES***));  *// 图书图片* doc.add(**new** TextField(**"bookPic"**,book.getPic(), Field.Store.***YES***));  *// 图书描述* doc.add(**new** TextField(**"bookDesc"**,book.getBookDesc(), Field.Store.***YES***));   documentList.add(doc);  }  *// 3.建立分析器（Analyzer），用于分词* Analyzer analyzer = **new** StandardAnalyzer();  *// 4.建立索引库配置对象（IndexWriterConfig），配置索引库  /\*\*  \* IndexWriterConfig参数：  \* 参数一：指定当前使用的lucene版本信息  \* 参数二：分析器对象  \*/* IndexWriterConfig iwc = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***,analyzer);  *// 5.建立索引库目录对象（Directory），指定索引库的位置* File file = **new** File(**"E:\\teach\\0351\\index\\"**);  Directory directory = FSDirectory.*open*(file);  *// 6.建立索引库操作对象（IndexWriter），把文档对象写入索引库* IndexWriter writer = **new** IndexWriter(directory,iwc);   **for**(Document doc:documentList){  */\*\*  \* addDocument方法：把文档对象写入索引库  \*/* writer.addDocument(doc);  }  *// 7.释放资源* writer.close();  } } |

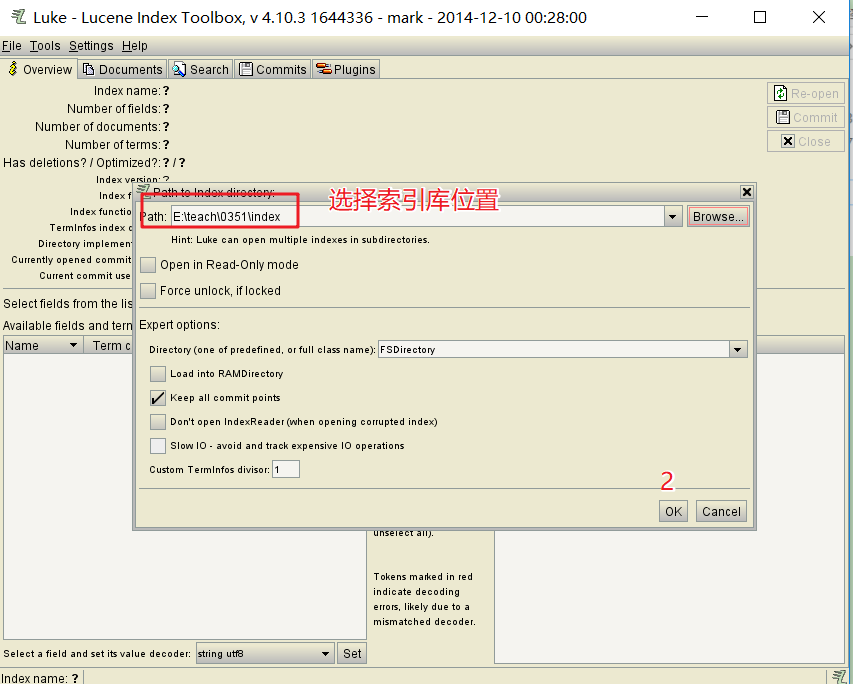


## 使用luke工具查看索引库

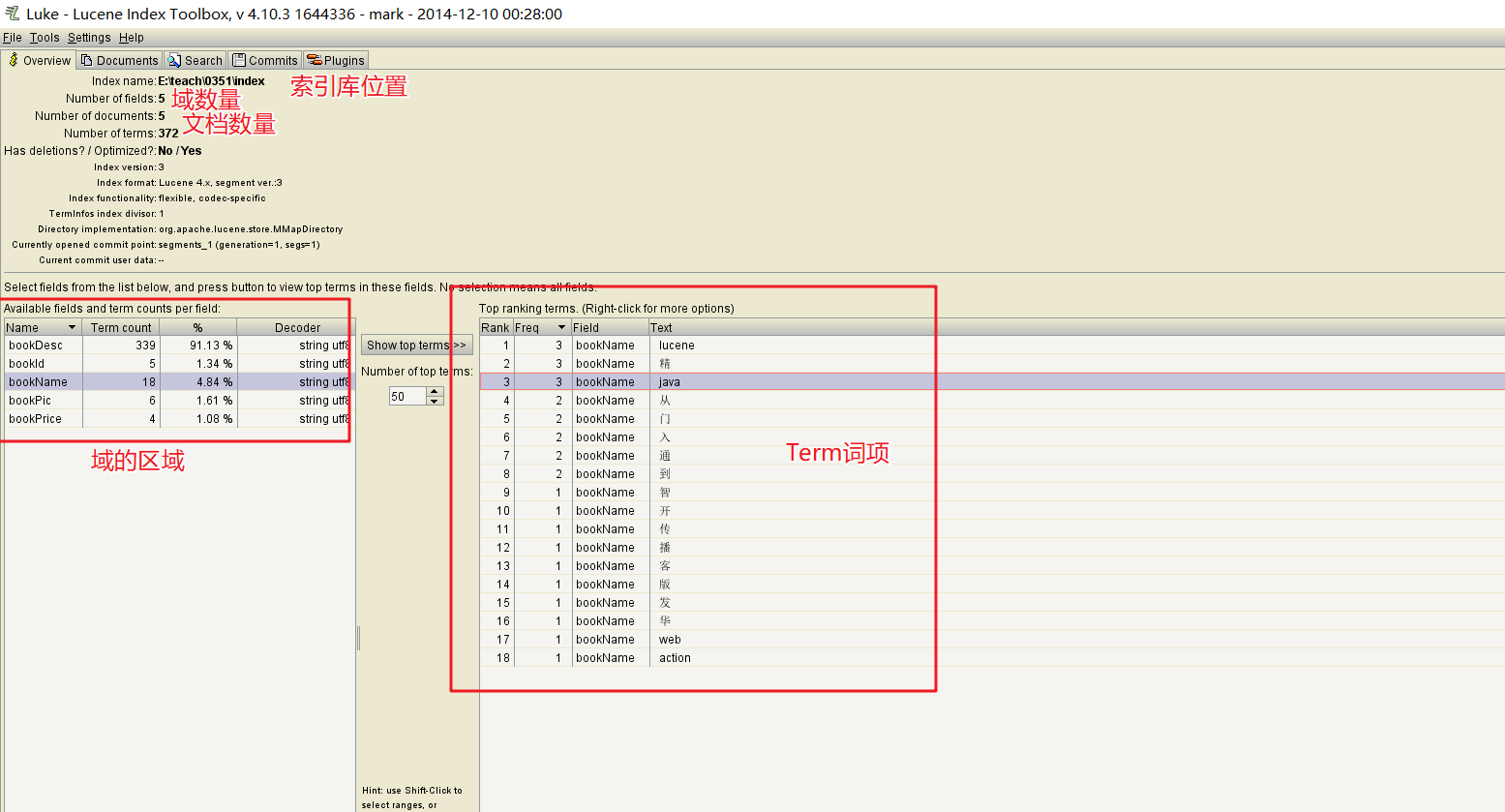


说明：前提是配置好jdk的环境变量。

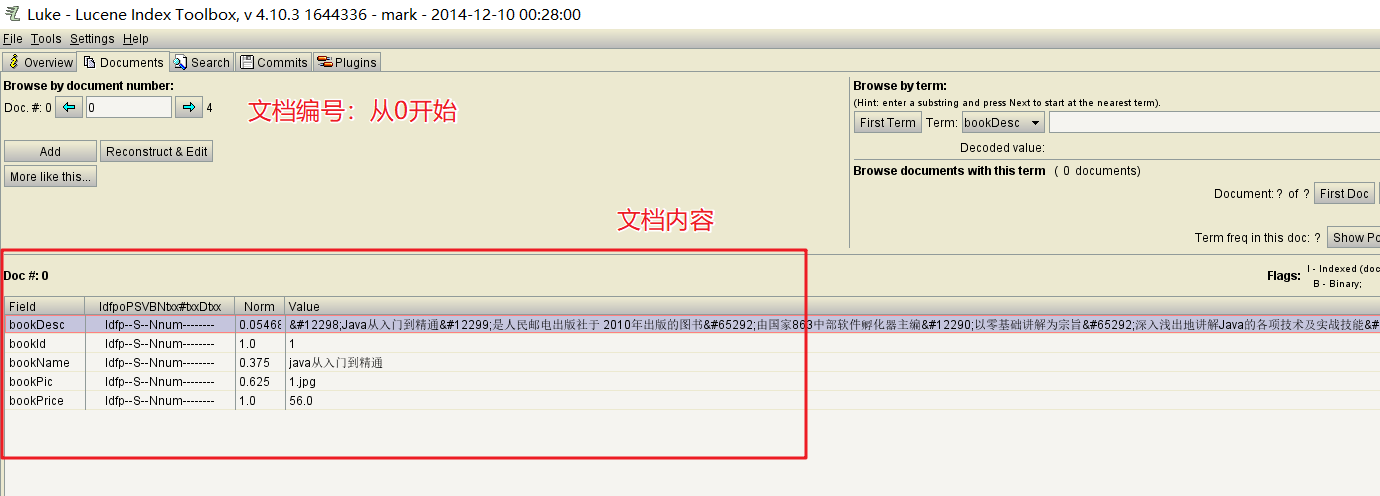
运行界面一：



运行界面二：

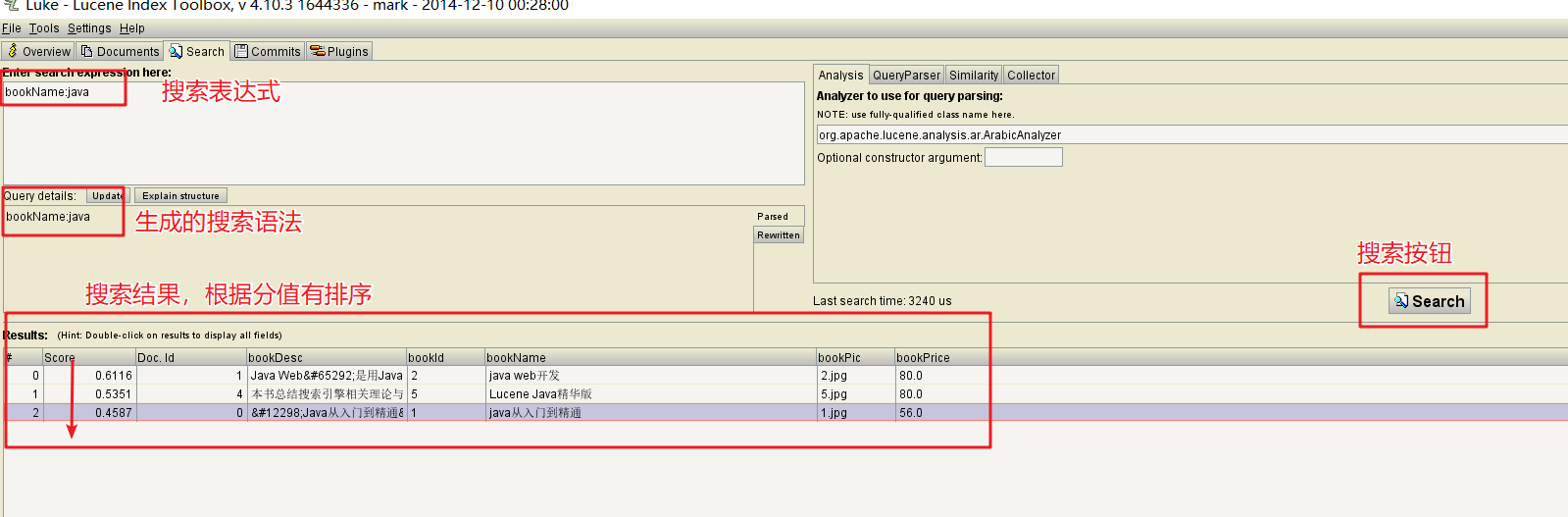


运行界面三：



## 检索流程实现

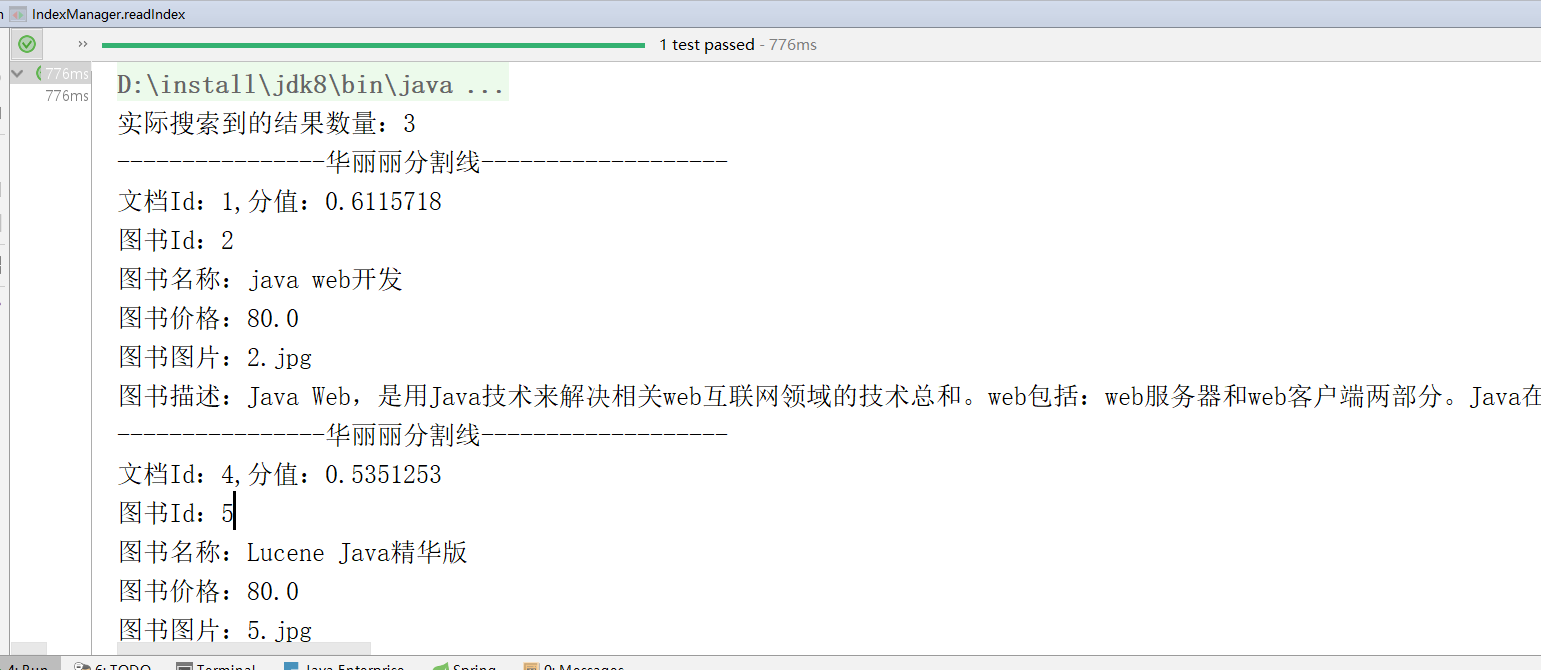
### 使用luke工具执行搜索



### 代码实现

1. 建立分析器对象（Analyzer），用于分词
2. 建立查询对象（Query）
3. 建立索引库的目录（Directory），指定索引库的位置
4. 建立索引读取对象（IndexReader），把索引数据读取到内存中
5. 建立索引搜索对象（IndexSearcher），执行搜索
6. 使用IndexSearcher执行搜索，返回搜索结果集（TopDocs）
7. 处理结果集
8. 释放资源

|  |
| --- |
| */\*\*  \* 检索流程实现  \*/* @Test  **public void** readIndex() **throws** Exception{ *// 1.建立分析器对象（Analyzer），用于分词* Analyzer analyzer = **new** StandardAnalyzer();  *// 2.建立查询对象（Query）  // 2.1.建立查询解析器对象  // 参数一：默认搜索域；参数二：分析器对象* QueryParser parser = **new** QueryParser(**"bookName"**,analyzer);   *// 2.2.使用查询解析器对象，解析表达式，实例化Query对象* Query query = parser.parse(**"bookName:java"**);  *// 3.建立索引库的目录（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***));  *// 4.建立索引读取对象（IndexReader），把索引数据读取到内存中* IndexReader reader = DirectoryReader.*open*(directory);  *// 5.建立索引搜索对象（IndexSearcher），执行搜索* IndexSearcher searcher = **new** IndexSearcher(reader);  *// 6.使用IndexSearcher执行搜索，返回搜索结果集（TopDocs）  /\*\*  \* search方法：执行搜索  \* 参数：  \* 参数一：查询对象  \* 参数二：指定搜索结果排序后的前n个（前10个）  \*  \* topN分析  \*/* TopDocs topDoc = searcher.search(query, 10);  *// 7.处理结果集  // 7.1.实际搜索到的结果数量* System.***out***.println(**"实际搜索到的结果数量："**+topDoc.**totalHits**);   *// 7.2.打印实际的数据  /\*\*  \* ScoreDoc中：只有文档id和分值信息  \*/* ScoreDoc[] scoreDocs = topDoc.**scoreDocs**;  **for**(ScoreDoc sd:scoreDocs){  System.***out***.println(**"----------------华丽丽分割线-------------------"**);  *// 取出文档Id和分值* **float** score = sd.**score**;  **int** docId = sd.**doc**;  System.***out***.println(**"文档Id："**+docId+**",分值："**+score);   *// 根据文档Id从文档域中取数据* Document doc = searcher.doc(docId);   System.***out***.println(**"图书Id："**+doc.get(**"bookId"**));  System.***out***.println(**"图书名称："**+doc.get(**"bookName"**));  System.***out***.println(**"图书价格："**+doc.get(**"bookPrice"**));  System.***out***.println(**"图书图片："**+doc.get(**"bookPic"**));  System.***out***.println(**"图书描述："**+doc.get(**"bookDesc"**));  } *// 8.释放资源* reader.close();  } |



说明：

1. 索引库中保存了索引域和文档域
2. 索引域保存倒排索引数据（用于执行搜索）
3. 文档域保存实际的业务数据（用于搜索结果取数据）

# 分词器

## 分词器介绍

在对文档（Document）中的内容进行索引前，需要对域（Field）中的内容使用分析对象（分词器）进行分词。分词的目的是为了索引，索引的目的是为了搜索。过程是先分词，再过滤。

* 分词：将Document中Field域的值切分成一个一个的单词。具体的切分方法（算法）根据使用的分词器而不同。
* 过滤：去除标点符号，去除停用词（的、啊、是、is、the、a等），词的大写转换小写。

停用词说明：停用词是指为了节省存储空间和提高搜索效率，搜索引擎在索引内容或处理搜索请求时会自动忽略的字词，这些字或词被称为“stop words”。比如语气助词、副词、介词、连接词等等，通常自身没有明确的含义，只有放在一个上下文语句中才有意义，比如常见的有：的、在、是、啊等。

分词流程图：

reader

分词器：

Tokenizer

过滤器：

TokenFilter

过滤器：

TokenFilter

......

term

term

.......

举例子：

原始文档：Lucene is a Java full-text search engine

分析以后的词：lucene java full text search engine

## 分词器使用

### 索引流程使用

把原始数据转换成文档对象后，使用分词器把文档域中的内容切分成一个一个的词语，目的是方便后续建立索引。

### 检索流程使用

根据用户输入的搜索关键词，使用分词器对象分析以后，建立成查询对象（Query），在索引库中查找目标内容。

注意事项：索引流程和检索流程使用分词器要一致。

## 中文分词器

### 中文分词器介绍

我们知道英文本身是以单词为单位，单词与单词之间，句子之间通常是空格、逗号、句号分隔。因此对于英文，可以简单的以空格来判断某个字符串是否是一个词，比如：I love China，love和China很容易被程序处理。

但是中文是以字为单位的，字与字再组成词，词再组成句子。中文：我爱中国，电脑不知道“爱中”是一个词，还是“中国”是一个词？所以我们需要一定的规则来告诉电脑应该怎么切分，这就是中文分词器所要解决的问题。常见的有一元切分法“我爱中国”：我、爱、中、国。二元切分法“我爱中国”：我爱，爱中、中国。

### lucene提供的中文分词器

* StandardAnalyzer分词器

单字分词器：一个字切分成一个词，一元切分法。

* CJKAnalyzer分词器

二元切分法：把相邻的两个字，作为一个词

* SmartChineseAnalyzer分词器

通常一元切分法，二元切分法都不能满足我们的业务需求。SmartChineseAnalyzer对中文支持较好，但是扩展性差，针对扩展词库、停用词均不好处理。

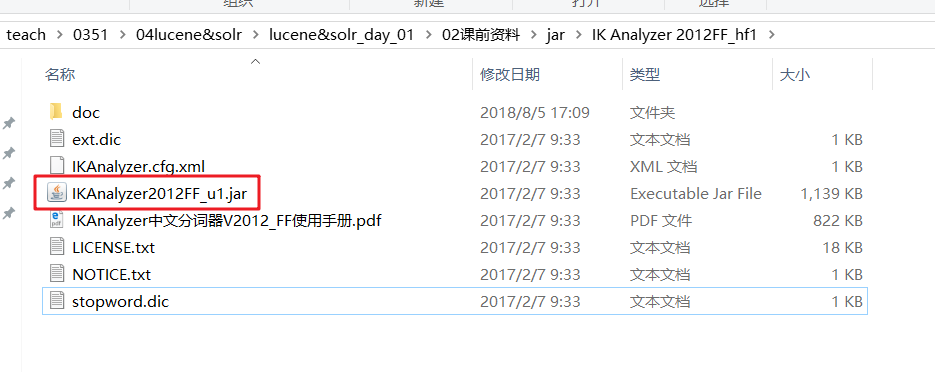
说明：lucene提供的中文分词器，只做了解，企业项目中不推荐使用。

### 第三方的中文分词器

* paoding：庖丁解牛分词器，最新版本在<https://code.google.com/p/paoding/>可以下载。由于没有持续更新，只支持到lucene3.0，项目中不予以考虑使用。
* mmseg4j：最新版已从<https://code.google.com/p/mmseg4j/>移至<https://github.com/chenlb/mmseg4j-solr>。支持Lucene4.10，且在github中有持续更新，使用的是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。适合在项目中应用。

## 使用ik分词器

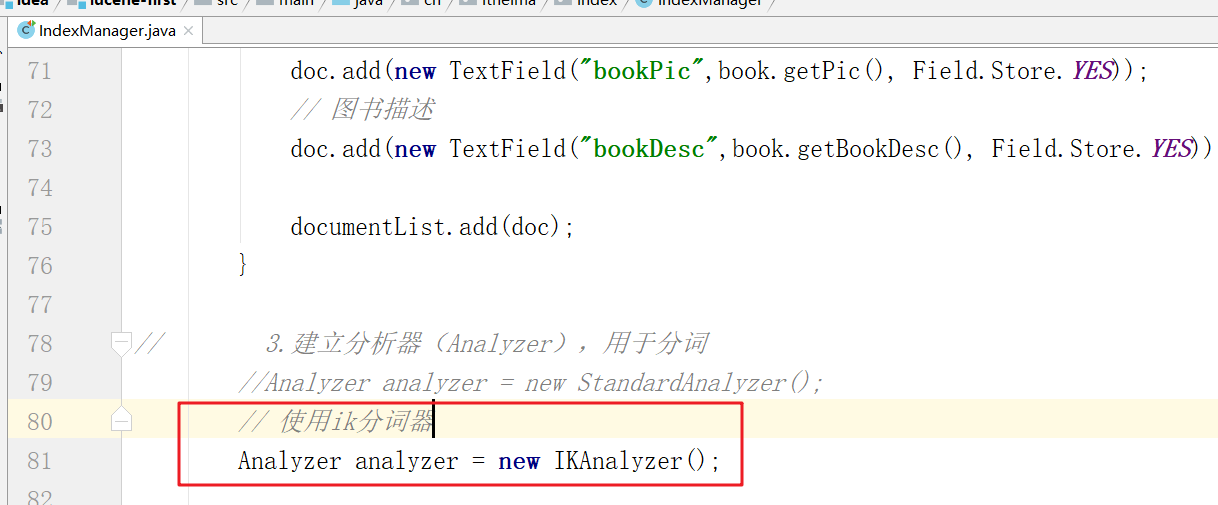
说明：ik分词器本身就是对lucene提供的分词器Analyzer扩展实现，使用方式与lucene的分词器一致。



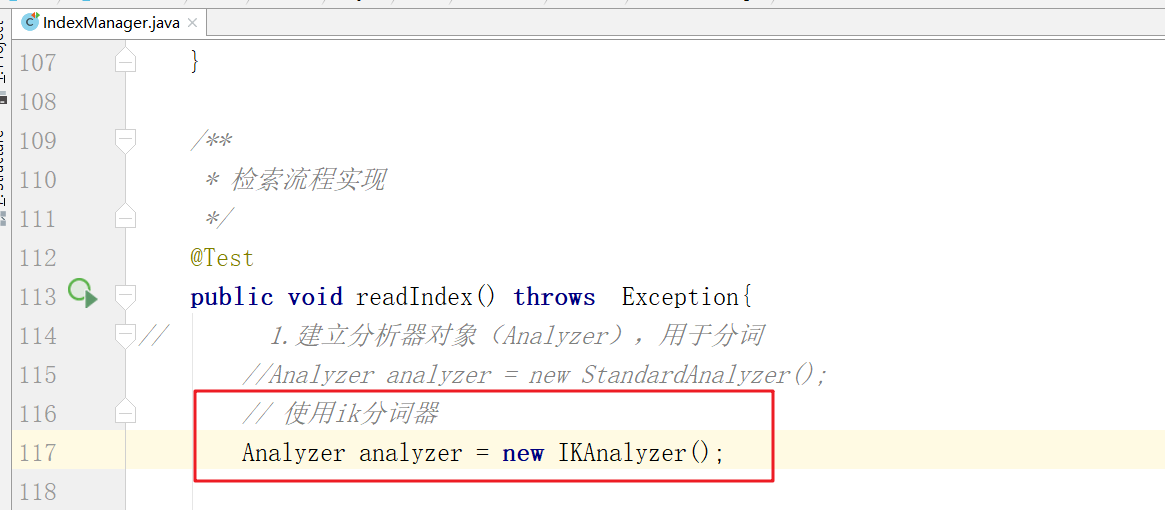
### 配置pom.xml，导入ik依赖

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>cn.itheima</**groupId**>  <**artifactId**>lucene-first</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>   <**packaging**>jar</**packaging**>   <**properties**>  <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  *<!-- mysql版本 -->* <**mysql.version**>5.1.30</**mysql.version**>  *<!-- lucene版本 -->* <**lucene.version**>4.10.3</**lucene.version**>  *<!-- ik分词器版本 -->* <**ik.version**>2012\_u6</**ik.version**>  </**properties**>   <**dependencies**>  *<!-- mysql数据库依赖 -->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>${mysql.version}</**version**>  </**dependency**>  *<!--lucene依赖包 -->* <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-core</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-analyzers-common</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-queryparser</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  *<!-- ik分词器 -->* <**dependency**>  <**groupId**>com.janeluo</**groupId**>  <**artifactId**>ikanalyzer</**artifactId**>  <**version**>${ik.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  </**dependency**>  </**dependencies**>   </**project**> |

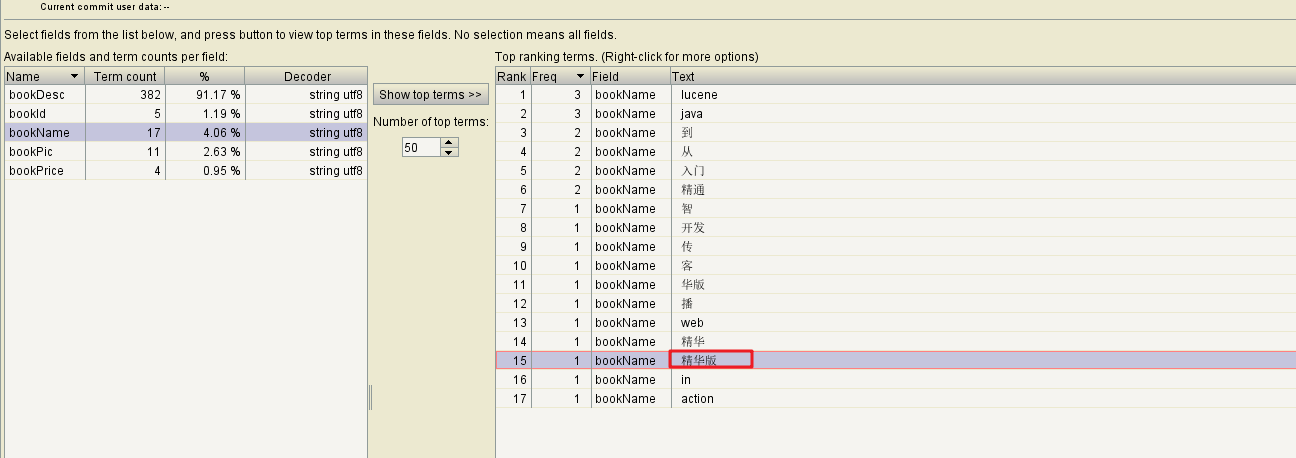
### 修改索引流程分词器



### 修改检索流程分词器

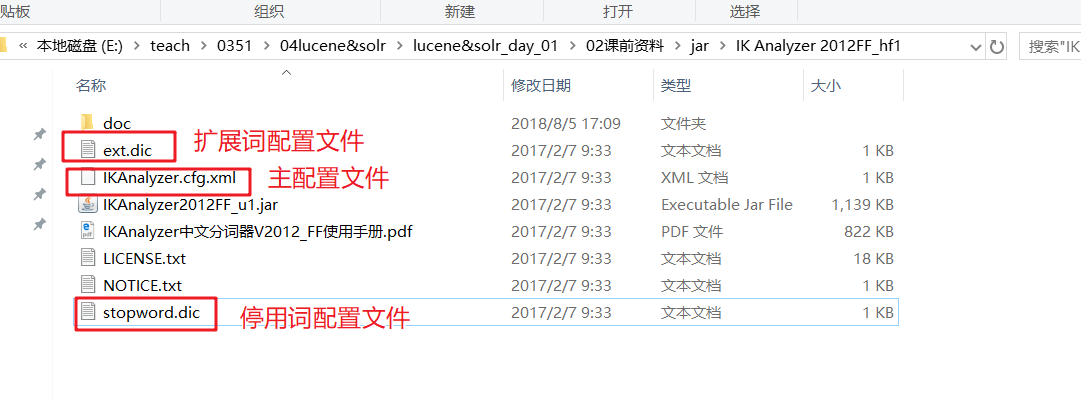


### 重新创建索引



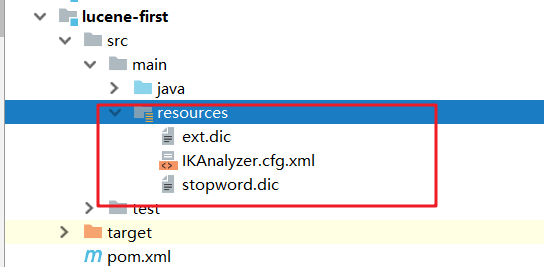
## 扩展中文词库

说明：在企业项目中，有一些词语根据业务需要不需要分词，需要作为一个整体，比如：传智播客。有一些词语会过时，不需要，要作为停用词处理。通过配置文件方式实现。



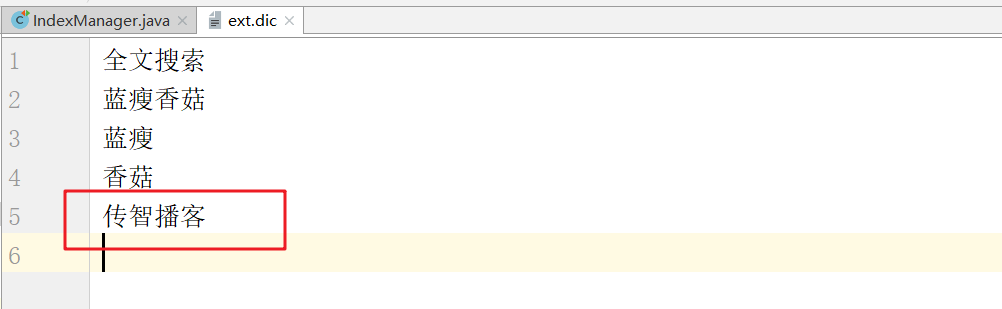
### 导入ik分词器的配置文件

说明：ik分词器的配置文件只能放到类的根路径。



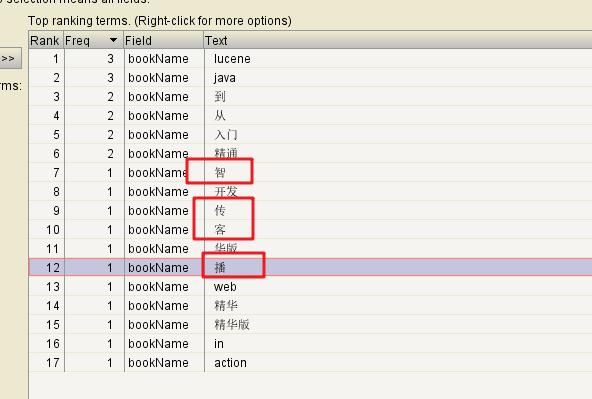
### 扩展词演示

#### 在ext.dic中增加扩展词：传智播客

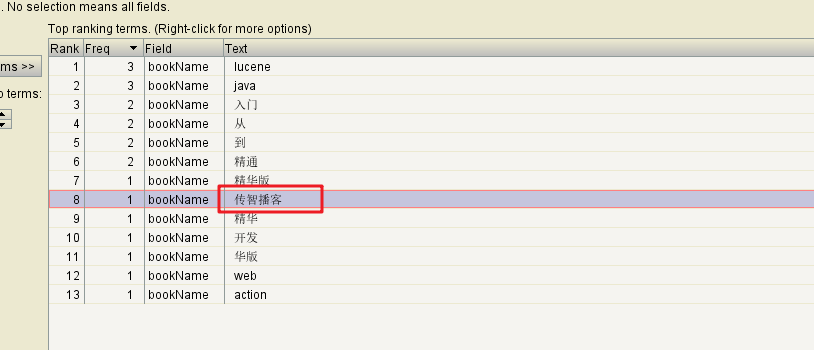


#### 重新创建索引

增加扩展词前：

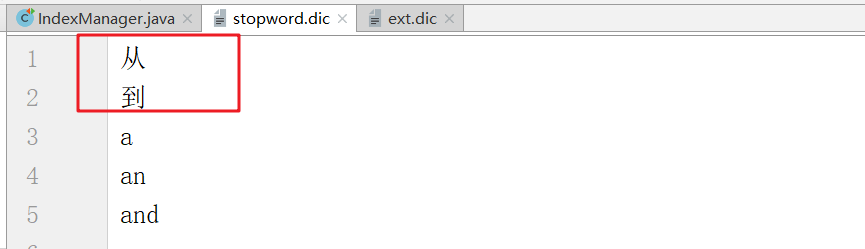


增加扩展词后：



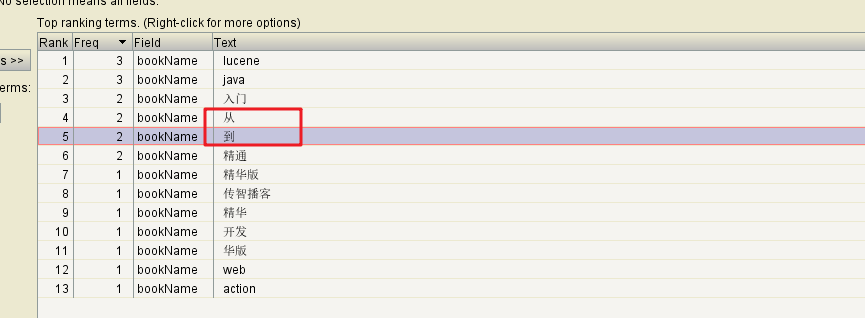
### 停用词演示

#### 在stopword.dic文件中增加停用词：从，到



#### 重新建立索引

增加停用词前：



增加停用词后：



# 扩展

## lucene分页搜索

说明：lucene是在内存中实现的分页。

|  |
| --- |
| */\*\*  \* 检索流程实现（实现分页搜索）  \*/* @Test  **public void** readIndexPage() **throws** Exception{ *// 1.建立分析器对象（Analyzer），用于分词  //Analyzer analyzer = new StandardAnalyzer();  // 使用ik分词器* Analyzer analyzer = **new** IKAnalyzer();  *// 2.建立查询对象（Query）  // 2.1.建立查询解析器对象  // 参数一：默认搜索域；参数二：分析器对象* QueryParser parser = **new** QueryParser(**"bookName"**,analyzer);   *// 2.2.使用查询解析器对象，解析表达式，实例化Query对象* Query query = parser.parse(**"bookName:java"**);  *// 3.建立索引库的目录（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***));  *// 4.建立索引读取对象（IndexReader），把索引数据读取到内存中* IndexReader reader = DirectoryReader.*open*(directory);  *// 5.建立索引搜索对象（IndexSearcher），执行搜索* IndexSearcher searcher = **new** IndexSearcher(reader);  *// 6.使用IndexSearcher执行搜索，返回搜索结果集（TopDocs）  /\*\*  \* search方法：执行搜索  \* 参数：  \* 参数一：查询对象  \* 参数二：指定搜索结果排序后的前n个（前10个）  \*  \* topN分析  \*/* TopDocs topDoc = searcher.search(query, 10);  *// 7.处理结果集  // 7.1.实际搜索到的结果数量* System.***out***.println(**"实际搜索到的结果数量："**+topDoc.**totalHits**);   *// 7.2.打印实际的数据  /\*\*  \* ScoreDoc中：只有文档id和分值信息  \*/* ScoreDoc[] scoreDocs = topDoc.**scoreDocs**;  *// 增加分页的处理=================================start  // 1.当前页* **int** page=2;*//默认搜索第一页   // 2.页面大小* **int** pageSize = 2;*// 默认每一页显示2条   // 3.开始记录索引* **int** start = (page-1)\*pageSize;   *// 4.结束记录索引  /\*\*  \* 普通的情况：start+pageSize  \* 最后一页：scoreDocs.length  \*/* **int** end = Math.*min*(start+pageSize,scoreDocs.**length**);   *// 增加分页的处理=================================end  // for(ScoreDoc sd:scoreDocs){* **for**(**int** i=start;i<end;i++){  System.***out***.println(**"----------------华丽丽分割线-------------------"**);  *// 取出文档Id和分值* **float** score = scoreDocs[i].**score**;  **int** docId = scoreDocs[i].**doc**;  System.***out***.println(**"文档Id："**+docId+**",分值："**+score);   *// 根据文档Id从文档域中取数据* Document doc = searcher.doc(docId);   System.***out***.println(**"图书Id："**+doc.get(**"bookId"**));  System.***out***.println(**"图书名称："**+doc.get(**"bookName"**));  System.***out***.println(**"图书价格："**+doc.get(**"bookPrice"**));  System.***out***.println(**"图书图片："**+doc.get(**"bookPic"**));  System.***out***.println(**"图书描述："**+doc.get(**"bookDesc"**));  } *// 8.释放资源* reader.close();  } |

