**全文检索课程**

**Lucene&Solr（1）**

# 课程计划

1. lucene的介绍
2. 什么是lucene
3. 全文检索的应用场景
4. 全文检索的定义
5. lucene实现全文检索的流程(重点)
6. 入门程序
7. 环境准备，工程搭建
8. 索引流程
9. 搜索流程
10. Field域（重点）
11. 索引的维护
12. 中文分词器

# Lucene介绍

## 什么是Lucene

Lucene是apache下的一个开源的全文检索引擎工具包(类库)。它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。

## 全文检索的应用场景

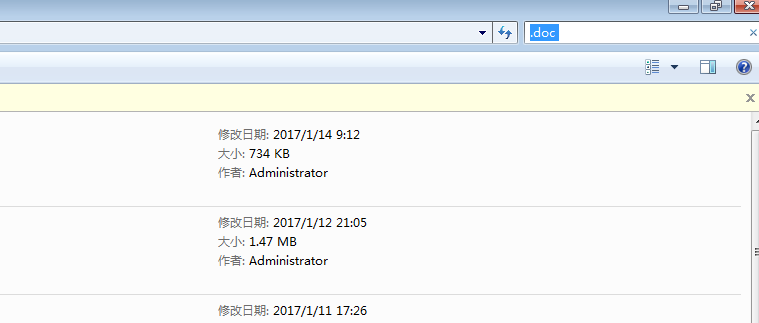
### 搜索引擎



### 站内搜索(重点关注)



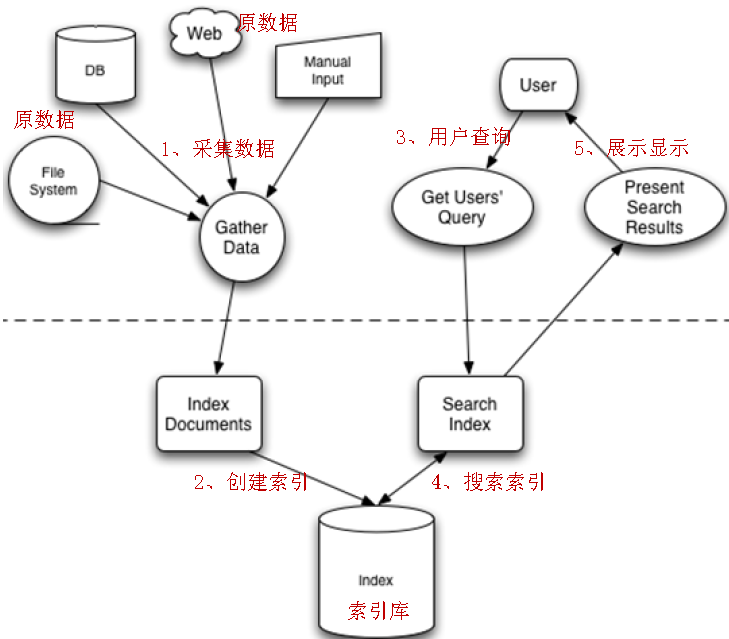
### 文件系统的搜索



## 全文检索定义

全文检索首先将要查询的目标文档中的词提取出来，组成索引，通过查询索引达到搜索目标文档的目的。**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）。**

# Lucene实现全文检索的流程



全文检索的流程分为两大部分：索引流程、搜索流程。

* **索引流程:**即采集数据🡪构建文档对象🡪分析文档（分词）🡪创建索引。
* **搜索流程**:即用户通过搜索界面输入🡪创建查询🡪执行搜索，搜索器从索引库搜索🡪渲染搜索结果。

## 采集数据

### 为什么要采集数据

全文检索要搜索的数据信息格式是多种多样的，比如：搜索引擎（百度， google），通过搜索引擎网站能搜索互联网上的网页(html)、互联网上的音乐(mp3..)、视频(avi..)、pdf电子书等。

全文检索搜索的这些数据称为非结构化数据。

什么是非结构化数据？

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。

如何对结构化数据搜索？

由于结构化数据是固定格式，所以就可以针对固定格式的数据设计算法来搜索，比如数据库like查询，like查询采用顺序扫描法，使用关键字匹配内容，对于内容量大的like查询速度慢。

如何对非结构化数据搜索？

需要将所有要搜索的非结构化数据通过技术手段采集到一个固定的地方，将这些非结构化的数据想办法组成结构化的数据，再以一定的算法去搜索。

由于数据源头是多种多样的，所以：需要将各种各样的数据按照一定的规则统一封装到lucene的文档对象中。有统一的规则，才能进行数据的存储，搜索。

采集数据的过程就是将数据按照规则封装到lucene文档对象中的过程。

### 如何采集数据

#### 网页采集（了解）

因为目前**搜索引擎**主要搜索数据的来源是互联网，搜索引擎使用一种爬虫程序抓取网页（通过http抓取html网页信息），以下是一些爬虫项目：

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。

heritrix（http://sourceforge.net/projects/archive-crawler/files/），Heritrix 是一个由 java 开发的、开源的网络爬虫，用户可以使用它来从网上抓取想要的资源。其最出色之处在于它良好的可扩展性，方便用户实现自己的抓取逻辑。

lucene:是工具包（API 类库）,它不能独立提供搜索服务，必须嵌入系统中提供搜索服务。

搜索引擎：是一个完整的系统（软件），独立的部署，独立的运行，提供独立的搜索的服务。

#### 数据库采集（掌握）

针对电商站内搜索功能，全文检索的数据源在数据库中，例如：可以通过jdbc访问数据库中图书表的内容。

//1.注册驱动

//2.获取连接

//3.sql语句

//4.创建statement

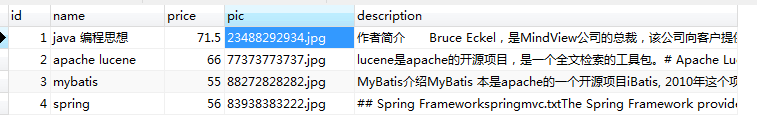
//5.设置参数

//6.执行

//7.获取结果集

//8.关闭资源。

如图：



##### Po

|  |
| --- |
| **public class** Book {  // 图书ID  **private** Integer id;  // 图书名称  **private** String name;  // 图书价格  **private** Float price;  // 图书图片  **private** String pic;  // 图书描述  **private** String description;  } |

##### Dao

|  |
| --- |
| **public interface** BookDao {  // 图书查询  **public** List<Book> queryBookList();  } |

|  |
| --- |
| **public** **class** BookDaoImpl **implements** BookDao {  @Override  **public** List<Book> queryBookList() {  // 数据库链接  Connection connection = **null**;  // 预编译statement  PreparedStatement preparedStatement = **null**;  // 结果集  ResultSet resultSet = **null**;  // 图书列表  List<Book> list = **new** ArrayList<Book>();  **try** {  // 加载数据库驱动  Class.*forName*("com.mysql.jdbc.Driver");  // 连接数据库  connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/book", "root", "itcast");  // SQL语句  String sql = "SELECT \* FROM book";  // 创建preparedStatement  preparedStatement = connection.prepareStatement(sql);  // 获取结果集  resultSet = preparedStatement.executeQuery();  // 结果集解析  **while** (resultSet.next()) {  Book book = **new** Book();  book.setId(resultSet.getInt("id"));  book.setName(resultSet.getString("name"));  book.setPrice(resultSet.getFloat("price"));  book.setPic(resultSet.getString("pic"));  book.setDescription(resultSet.getString("description"));  list.add(book);  }  } **catch** (Exception e) {  e.printStackTrace();  }**finally** {  **if** (resultSet!=**null**) {  **try** {  resultSet.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  **if** (preparedStatement!=**null**) {  **try** {  preparedStatement.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  **if**(connection!=**null**){  **try** {  connection.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  **return** list;  }  } |

文件系统的采集数据：IO （了解）

## 图书的索引文件的逻辑结构说明



* 文档域：

对非结构化的数据统一格式为document文档格式，一个文档有多个field域，不同的文档其field的个数可以不同，建议相同类型的文档包括相同的field。

本例子一个document对应一条 book表的记录。

* 索引域：

用于搜索，搜索程序将从索引域中搜索一个一个词，根据词找到对应的文档。

将Document中的Field的内容进行分词，将分好的词创建索引。

索引域 包括两个部分：

一部分是：域的名称

一部分是：要搜索的内容

比如：bookname:java 要从域的名称为bookname上搜索内容为java的文档对象。

# 入门程序

## 需求

使用Lucene实现**图书类商品的索引和搜索功能**。

## 环境准备

* Jdk环境：1.7
* 开发工具：eclipse mars2
* 数据库环境：mysql 5
* Lucene：4.10.3

### 数据库准备



### Lucene下载安装

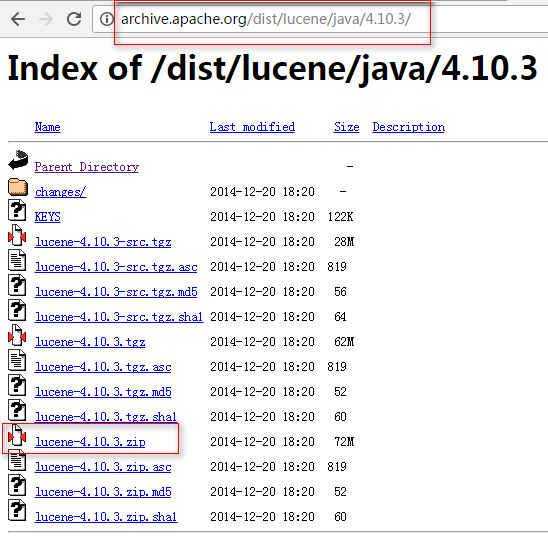
Lucene是开发全文检索功能的工具包，使用时从官方网站下载，并解压。

官方网站：<http://lucene.apache.org/>

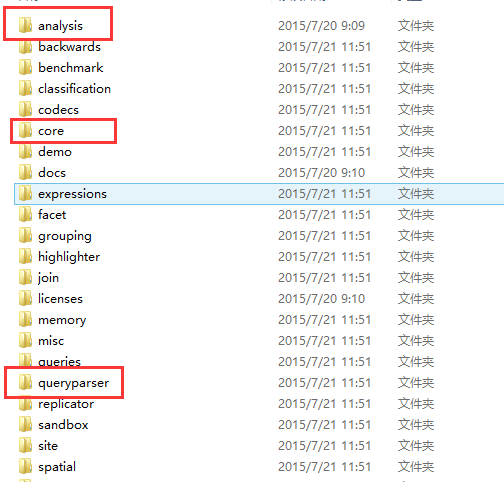
下载地址：<http://archive.apache.org/dist/lucene/java/>

下载版本：4.10.3

JDK要求：1.7以上（从版本4.8开始，不支持1.7以下）

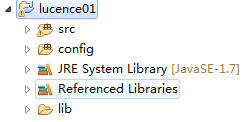


需要关注的目录:



## 工程搭建

### 第一步：创建java工程



### 第二步：添加jar包

入门程序只需要添加以下jar包：

* mysql5.1驱动包：mysql-connector-java-5.1.7-bin.jar
* 核心包：lucene-core-4.10.3.jar
* 分词器通用包：lucene-analyzers-common-4.10.3.jar
* junit包：junit-4.9.jar

## 索引流程

对文档索引的过程，就是将用户要**搜索的文档内容**进行**索引**，然后把**索引**存储在**索引库（index）**中。索引的流程：采集数据--》构建文档--》分析--》创建索引

采集数据：参考3.1的代码，创建PO和DAO

### 创建索引

创建索引流程：

索引目录流对象

Directory

索引写对象

IndexWriter

索引库

文档

Document

分词器

Analyzer

**IndexWriter**是索引过程的核心组件，通过IndexWriter可以**创建新索引、更新索引、删除索引**操作。IndexWriter需要通过**Directory**对索引进行存储操作。

**Directory**描述了索引的存储位置，底层封装了I/O操作，负责对索引进行存储。它是一个抽象类，它的子类常用的包括**FSDirectory**（在文件系统存储索引）、**RAMDirectory**（在内存存储索引）。

#### 创建Document

采集数据的目的是为了索引，在索引前需要将原始内容创建成**文档（Document），文档（Document）中包括一个一个的域（Field）。**

##### 代码实现

|  |
| --- |
| // 第一步采集数据：(jdbc采集数据)  BookDao dao = **new** BookDaoImpl();  List<Book> queryBookList = dao.queryBookList();  // 将数据采集放到docment对象中  Document doc = **null**;  List<Document> docList = **new** ArrayList<>();  **for** (Book book : queryBookList) {  // 根据需求创建不同的field  doc = **new** Document();  Field id = **new** TextField("id", book.getId().toString(), Store.***YES***);  Field name = **new** TextField("name", book.getName(), Store.***YES***);  Field price = **new** TextField("price", book.getPrice().toString(), Store.***YES***);  Field pic = **new** TextField("pic", book.getPic(), Store.***YES***);  Field desc = **new** TextField("description", book.getDescription(), Store.***YES***);  // 将field域，添加到文档对象中  doc.add(id);  doc.add(name);  doc.add(price);  doc.add(pic);  doc.add(desc);  docList.add(doc);  } |

#### 分词

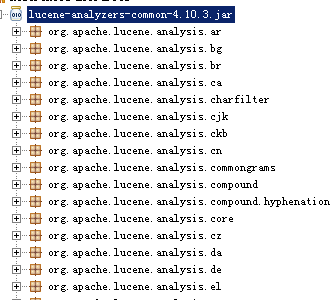
##### 分词过程

在对Document中的内容索引之前需要使用分词器（analyzer）进行分词，主要过程就是分词、过虑两步。

* **分词**就是将采集到的文档内容切分成一个一个的词，具体应该说是将Document中Field的value值切分成一个一个的词。
* **过虑**包括去除标点符号、去除停用词（的、是、a、an、the等）、大写转小写、词的形还原（复数形式转成单数形式、过去式转成现在式。。。）等。

什么是停用词？停用词是为节省存储空间和提高搜索效率，搜索引擎在索引页面或处理搜索请求时会自动忽略某些字或词，这些字或词即被称为Stop Words(停用词)。比如语气助词、副词、介词、连接词等，通常自身并无明确的意义，只有将其放入一个完整的句子中才有一定作用，如常见的“的”、“在”、“是”、“啊”等。

Lucene作为了一个工具包提供不同国家的分词器，如下图：



注意由于语言不同分词器的切分规则也不同，本例子使用StandardAnalyzer，它可以对用英文进行分词。

例如源文档内容为：Lucene is a Java full-text search engine.

分词之后：

lucene

java

full

text

search

engine

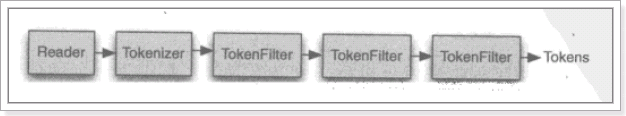
如下是org.apache.lucene.analysis.standard.standardAnalyzer的部分源码：

|  |
| --- |
| @Override  **protected** TokenStreamComponents createComponents(**final** String fieldName, **final**Reader reader) {  **final**StandardTokenizer src = **new**~~StandardTokenizer~~(getVersion(), reader);  src.setMaxTokenLength(maxTokenLength);  TokenStream tok = **new**~~StandardFilter~~(getVersion(), src);  tok = **new**~~LowerCaseFilter~~(getVersion(), tok);  tok = **new**~~StopFilter~~(getVersion(), tok, stopwords);  **returnnew** TokenStreamComponents(src, tok) {  @Override  **protectedvoid** setReader(**final** Reader reader) **throws** IOException {  src.setMaxTokenLength(StandardAnalyzer.**this**.maxTokenLength);  **super**.setReader(reader);  }  };  } |

Tokenizer是分词器，负责将reader转换为语汇单元即进行分词，Lucene提供了很多的分词器，也可以使用第三方的分词，比如IKAnalyzer一个中文分词器。

tokenFilter是分词过滤器，负责对语汇单元进行过滤，tokenFilter可以是一个过滤器链，Lucene提供了很多的分词器过滤器，比如大小写转换、去除停用词等。

如下图是语汇单元的生成过程：（）



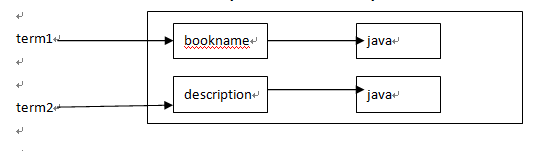
**从一个Reader字符流开始，创建一个基于Reader的Tokenizer分词器，经过三个TokenFilter生成语汇单元Token。**

Token就是分词过程中产生的对象（**包含分词的词语内容，该词在文本中的开始和结束位置**）。

term 是搜索时的最小单元。

为了便于理解分词后的一个单词可以叫做一个term，不同的域中拆分出来的相同的单词是不同的term。

例如：图书信息里面，图书名称中的java和图书描述中的java具有不同的含义，也就对应不同的term。



（Term包含两个部分：域的名称以及关键字（分词后的单词））；

##### 实现分词的代码实现

|  |
| --- |
| // 分析文档，对文档中的field域进行分词  Analyzer analyzer = **new** StandardAnalyzer(); |

##### 分词器使用的时机

索引时使用：

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分词器处理生成语汇单元（Token）

搜索时使用：

对搜索关键字进行分析和索引分析一样，使用Analyzer对搜索关键字进行分析、分词处理，使用分析后每个词语进行搜索，匹配。

**注意：搜索使用的分词器要和索引使用的分词器一致。**

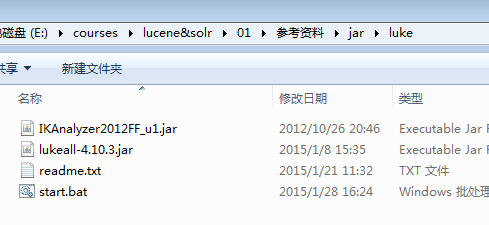
#### 创建索引的完整代码如下：

|  |
| --- |
| **public** **void** createIndex() **throws** IOException {  // 第一步采集数据：(jdbc采集数据)  BookDao dao = **new** BookDaoImpl();  List<Book> queryBookList = dao.queryBookList();  // 将数据采集放到docment对象中  Document doc = **null**;  List<Document> docList = **new** ArrayList<>();  **for** (Book book : queryBookList) {  // 根据需求创建不同的field  doc = **new** Document();  Field id = **new** TextField("id", book.getId().toString(), Store.***YES***);  Field name = **new** TextField("name", book.getName(), Store.***YES***);  Field price = **new** TextField("price", book.getPrice().toString(), Store.***YES***);  Field pic = **new** TextField("pic", book.getPic(), Store.***YES***);  Field desc = **new** TextField("description", book.getDescription(), Store.***YES***);  // 将field域，添加到文档对象中  doc.add(id);  doc.add(name);  doc.add(price);  doc.add(pic);  doc.add(desc);  docList.add(doc);  }  //创建一个indexWriter对象(通过它反向推理出需要的条件)  //先构造一个directory 指定索引库的位置，一般使用文件系统的目录。  FSDirectory directory = FSDirectory.*open*(**new** File("G:\\index01"));  //创建一个分词器 对文档中的Field域进行分词  Analyzer analyzer = **new** StandardAnalyzer();//标准分词器  IndexWriterConfig config = **new** IndexWriterConfig(Version.***LATEST***, analyzer);  IndexWriter writer = **new** IndexWriter(directory,config);  //创建索引  **for** (Document document : docList) {  writer.addDocument(document);  }  //关闭资源  writer.close();  } |

#### 使用Luke查看索引

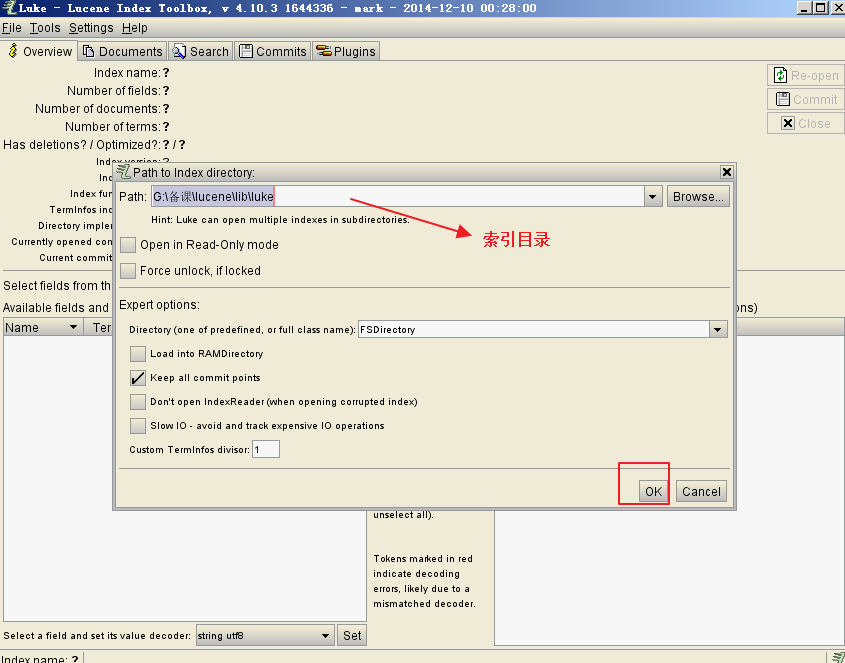
Luke作为Lucene工具包中的一个工具（http://www.getopt.org/luke/），可以通过界面来进行索引文件的查询、修改。

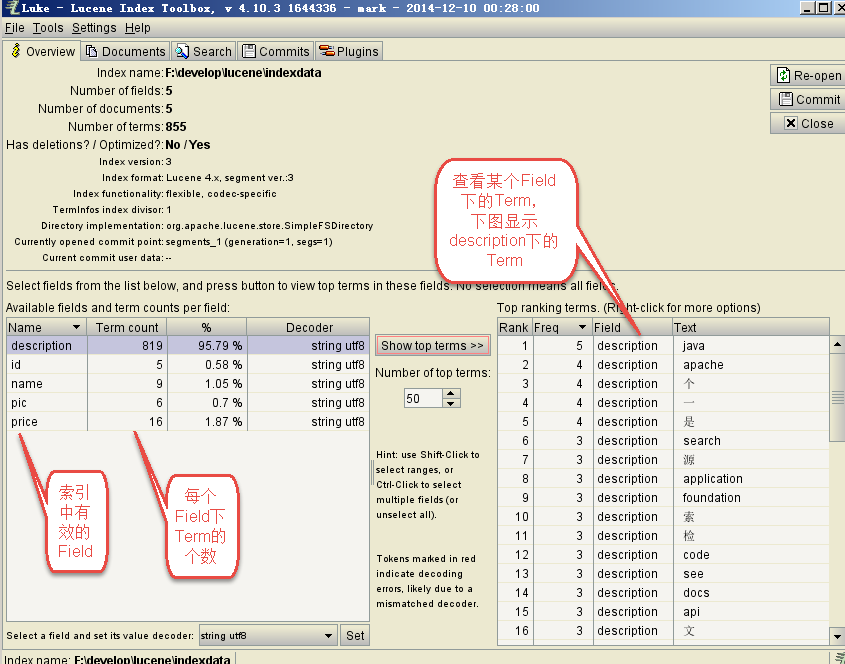
luke 所在的目录：



打开Luke方法：

* 命令运行：cd 到luke 所在的目录,cmd运行：java -jar lukeall-4.10.3.jar
* 手动执行：双击lukeall-4.10.3.jar
* 使用脚本start.bat





## 搜索流程

搜索过程：

索引搜索对象

IndexSearcher

索引读取对象

IndexReader

索引目录流对象

DirectoryReader

索引库

查询对象

Query

结果集

TopDocs

1、用户定义查询语句，用户确定查询什么内容（输入什么关键字）

2、IndexSearcher索引搜索对象，定义了很多搜索方法，程序员调用此方法搜索。

3、IndexReader索引读取对象，它对应的索引维护对象IndexWriter，IndexSearcher通过IndexReader读取索引目录中的索引文件

4、Directory索引流对象，IndexReader需要Directory读取索引库，使用FSDirectory文件系统流对象

5、IndexSearcher搜索完成，返回一个TopDocs（匹配度高的前边的一些记录）

### 输入查询语句

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 “name:lucene”表示要搜索Field域为name内容为“lucene”的文档

### 执行查询

通过倒排索引的方式进行查询文档。(了解)

* 倒排索引表

传统方法是先找到文件，再找内容，如何在文件中找内容，在文件内容中匹配搜索关键字，这种方法是**顺序扫描方法**，数据量大就搜索慢。

**倒排索引结构**是根据内容（词语）找文档，**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，它是在索引中匹配搜索关键字，由于索引内容量有限并且采用固定优化算法搜索速度很快，找到了索引中的词汇，词汇与文档关联，从而最终找到了文档。**

比如：name:lucene 表示要从name域中查询lucene的文档，如下图查询出文档的ID列表。



### 获取结果，并渲染结果

(jsp)

将查询的结果集进行处理，并渲染到页面，提供给用户一个友好的界面。

例如：



### 开发步骤

第一步：创建一个Directory对象，也就是索引库存放的位置。

第二步：创建一个indexReader对象，需要指定Directory对象。

第三步：创建一个indexsearcher对象，需要指定IndexReader对象

第四步：创建一个TermQuery对象，指定查询的域和查询的关键词。

第五步：执行查询。

第六步：返回查询结果。遍历查询结果并输出。

第七步：关闭IndexReader对象

代码实现如下：

|  |
| --- |
| @Test  **public** **void** doIndexSearch() **throws** IOException{  //第一步骤：指定一个索引库的位置  FSDirectory directory = FSDirectory.*open*(**new** File("G:\\index01"));  //第二步：创建一个indexReader对象  IndexReader reader = DirectoryReader.*open*(directory);  //第三步：创建一个indexSearcher对象  IndexSearcher searcher = **new** IndexSearcher(reader);  //第四步：创建一个termquery对象，指定查询的域和关键字  TermQuery query = **new** TermQuery(**new** Term("name", "lucene"));  //第五步：执行查询  TopDocs topDocs = searcher.search(query, 10);  //第六步：执行查询结果，遍历查询结果并输出  System.***out***.println(">>>总记录数"+topDocs.totalHits);    ScoreDoc[] docs = topDocs.scoreDocs;    **for** (ScoreDoc scoreDoc : docs) {  **int** id = scoreDoc.doc;//文档的id  Document doc = searcher.doc(id);//获取doc对象    System.***out***.println(doc.get("id"));  System.***out***.println(doc.get("name"));  System.***out***.println(doc.get("price"));  System.***out***.println(doc.get("pic"));  //System.out.println(doc.get("description"));  }  //第七步：关闭资源  reader.close();  } |

# Field域

## Field属性

Field是文档中的域，包括Field名和Field值两部分，一个文档可以包括多个Field，Document只是Field的一个承载体，Field值即为要索引的内容，也是要搜索的内容。

* **是否分词(tokenized)**

是，将field的内容分成一个一个单词。分词的**目的：分词目的为了索引**

**商品的名称，商品的介绍。**

否，不分词，将内容作为一个整体存储。

商品ID 身份证号，**图片路径**

* **是否索引(indexed)**

是, 将field的值建立索引**，索引的目的：索引的目的为了搜索。**

**商品的名称，商品的介绍**

**否，不建立索引**

**图片路径**

* **是否存储(stored)**

**是， 存储field的值。 存储的目的：（为了展示在页面）**

**商品名称，图片路径**

**否， 不存储field的值。**

**商品介绍。如果需要展示，根据ID从数据库查询展示在详情页面。**

## Field常用类型（重点）

下边列出了开发中常用的Filed类型，注意Field的属性，根据需求选择：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field类 | 数据类型 | Analyzed  是否分词 | 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，但是不会进行分词，会将整个串存储在索引中，比如(订单号,身份证号等)  是否存储在文档中用Store.YES或Store.NO决定 |
| LongField(FieldName, FieldValue,Store.YES)  FloatField(FieldName, FieldValue,Store.YES) | Long型/Float  /。。。 | Y | Y | Y或N | 这个Field用来构建一个Long数字型Field，进行分词和索引，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（图片路径）  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. |

## Field入门程序代码修改如下

图书id：

是否分词：不用分词，因为不会根据图书id来搜索图书

是否索引：索引，可能需要作为其他索引的查询条件

是否存储：要存储，因为查询结果页面需要使用id这个值。

图书名称：

是否分词：要分词，因为要将图书的名称内容分词索引，根据关键搜索图书名称抽取的词。

是否索引：要索引。

是否存储：要存储。

图书价格：

是否分词：要分词，**lucene对数字型的值只要有搜索需求的都要分词和索引，因为lucene对数字型的内容要特殊分词处理**，本例子可能要根据价格范围搜索，需要分词和索引。

是否索引：要索引

是否存储：要存储

图书图片地址：

是否分词：不分词

是否索引：不索引

是否存储：要存储

图书描述：

是否分词：要分词

是否索引：要索引

是否存储：因为图书描述内容量大，不在查询结果页面直接显示，不存储。

不存储是用做不在lucene的索引文件中记录，节省lucene的索引文件空间，如果要在详情页面显示描述，思路：

从lucene中取出图书的id，根据图书的id查询关系数据库中book表得到描述信息。

部分代码如下：

|  |
| --- |
| document.add(**new** StringField("id",book.getId().toString(),Store.***YES***));//不分词，索引，要存储  document.add(**new** TextField("name",book.getName(),Store.***YES***));//分词，索引，存储  document.add(**new** FloatField("price",book.getPrice().floatValue(),Store.***YES***));//分词，索引，存储  document.add(**new** StoredField("pic",book.getPic()));//不分词，不索引，要存储  document.add(**new** TextField("description", book.getDescription(), Store.***NO***));//分词，索引，不存储 |

注意：先删除原来的index库,如果正在使用luke 还需要先关闭luke 。

# 索引维护

## 需求

管理人员通过电商系统更改图书信息，这时更新的是数据库，如果使用lucene搜索图书信息需要在数据库表book信息变化时及时更新lucene索引库。

涉及到索引的维护。

## 添加索引

调用indexWriter.addDocument（doc）添加索引。

参考入门程序的创建索引。

## 删除索引

### 删除指定索引

根据**Term项**删除索引，满足条件的将全部删除。

|  |
| --- |
| // 删除索引  @Test  **Public void** deleteIndex() **throws** Exception {  // 1、指定索引库目录  Directory directory = FSDirectory.*open*(**new** File("E:\\11-index\\0720"));  // 2、创建IndexWriterConfig  IndexWriterConfig cfg = **new** IndexWriterConfig(Version.*LATEST*,  **new** StandardAnalyzer());  // 3、创建IndexWriter  IndexWriter writer = **new** IndexWriter(directory, cfg);  // 4、通过IndexWriter来删除索引  // b)、删除指定索引  writer.deleteDocuments(**new** Term("filename", "apache"));  // 5、关闭IndexWriter  writer.close();  } |

### 删除全部索引（慎用）

将索引目录的索引信息全部删除，直接彻底删除，无法恢复。慎用！！！

|  |
| --- |
| // 删除索引  @Test  **publicvoid** deleteIndex() **throws** Exception {  // 1、指定索引库目录  Directory directory = FSDirectory.*open*(**new** File("E:\\11-index\\0720"));  // 2、创建IndexWriterConfig  IndexWriterConfig cfg = **new** IndexWriterConfig(Version.*LATEST*,  **new** StandardAnalyzer());  // 3、创建IndexWriter  IndexWriter writer = **new** IndexWriter(directory, cfg);  // 4、通过IndexWriter来删除索引  // a)、删除全部索引  writer.deleteAll();  // 5、关闭IndexWriter  writer.close();  } |

建议参照关系数据库基于主键删除方式，所以在创建索引时需要创建一个主键Field，删除时根据此主键Field删除。

索引删除后将放在Lucene的回收站中，Lucene3.X版本可以恢复删除的文档，3.X之后无法恢复。

## 更新索引

更新索引是先删除再添加，如果没有查到文档，直接添加。建议对更新需求采用此方法并且要保证对已存在的索引执行更新，可以先查询出来，确定更新记录存在，再执行更新操作。

|  |
| --- |
| /\*\*  \* 更新 是先删除再添加。如果没有文档，就直接添加。  \* **@throws** Exception  \*/  @Test  **public** **void** update() **throws** Exception {  // 获取indexWriter对象  IndexWriter indexwriter = getIndexwriter();  //先构建更新后的文档  Document document = **new** Document();  document.add(**new** TextField("name", "lucene", Store.***YES***));  //查询出name为lucne的文档，再将该文档更新成新的。如果不存在直接添加文档 已经验证过了。  indexwriter.updateDocument(**new** Term("name","java"), document);  //关闭流  indexwriter.close();  } |

# 中文分词器

## 什么是中文分词器

学过英文的都知道，英文是以单词为单位的，**单词与单词之间以空格或者逗号句号隔开**。而中文则以字为单位，字又组成词，字和词再组成句子。**所以对于英文，我们可以简单以空格判断某个字符串是否为一个单词**，比如I love China，love 和 China很容易被程序区分开来；但中文“我爱中国”就不一样了，电脑不知道“中国”是一个词语还是“爱中”是一个词语。**把中文的句子切分成有意义的词，就是中文分词，也称切词**。我爱中国，分词的结果是：我爱中国。

目的：以现有的方式分词中文，不能满足我们的需求，需要使用第三方的分词器，中文分词器。

## 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。

## 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**。词的扩展 以及停用词的扩展。
* 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。

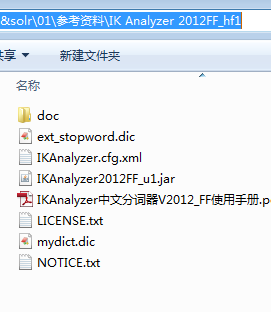
## 使用中文分词器IKAnalyzer

IKAnalyzer继承Lucene的Analyzer抽象类，使用IKAnalyzer和Lucene自带的分词器方法一样，将Analyzer测试代码改为IKAnalyzer测试中文分词效果。

可以配置扩展词典和停用词词典。

如果使用中文分词器ik-analyzer，就需要在索引和搜索流程程序中使用一致的分词器ik-analyzer。

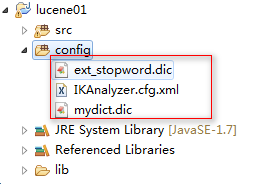
### 添加jar包



第一步：将jar包添加到lib 并且添加到buildpath中。

第二步：将3个文件（mydict.dic,IKAnalyzer.cfg.xml,ext\_stopword.dic） 拷贝到classpath

先新建source folder 名字叫config 如图：



IKAnalyzer.cfg.xml: 配置如下

|  |
| --- |
| <?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"*>mydict.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"*>ext\_stopword.dic</entry>  </properties> |

### 修改分词器代码

|  |
| --- |
| // 创建中文分词器  Analyzer analyzer = **new** IKAnalyzer(); |

## 扩展中文词库的内容。

如上图：

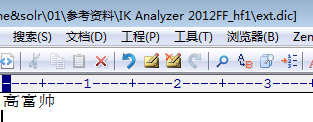
扩展词典文件为：mydict.dic

停用词词典文件为：ext\_stopword.dic

如果想配置扩展词和停用词，就创建扩展词的文件和停用词的文件，文件的编码要是utf-8。

注意：**不要用记事本保存扩展词文件和停用词文件，那样的话，格式中是含有bom的**。

添加扩展词文件：ext.dic，内容如下：



保存时，请保存为UTF-8

## 测试分词效果（了解）

|  |
| --- |
| **public** **void** testAnalyzer() **throws** IOException{  //1.创建analyzer (ik)  Analyzer analyzer = **new** IKAnalyzer();  //2.获取tokenstream (分的词都在此对象中)  //第一个参数：就是域的名称，可以不写或者""  //第二个参数：分析的词内容  TokenStream tokenStream = analyzer.tokenStream("", "高富帅this is a test");//高富帅是假的  //3.指定一个引用 （指定 词的引用 或者 偏移量）  CharTermAttribute addAttribute = tokenStream.addAttribute(CharTermAttribute.**class**);  //4.设置一个偏移量的引用  OffsetAttribute offsetAttribute = tokenStream.addAttribute(OffsetAttribute.**class**);  //5.调用tokenstream的rest方法 重置  tokenStream.reset();  //6.通过wihle 循环 遍历单词列表  **while**(tokenStream.incrementToken()){  ///打印  System.***out***.println("start>>"+offsetAttribute.startOffset());  System.***out***.println(addAttribute.toString());//打印单词  System.***out***.println("end>>"+offsetAttribute.endOffset());  }  tokenStream.close();  //关闭流  } |