Lucene

# Lucene入门

## Lucene介绍

什么是Lucene？

Lucene是apache下的一个开放源代码的全文检索引擎工具包。Lucene可以实现全文检索功能，Lucene就是一个工具包（提供了Jar包，实现全文检索的类库）。Lucene只是一个引擎，只是一个工具包，如果使用Lucene开发全文检索功能，Lucene是不能单独运行的。

程序员或开发工程师使用lucene提供的类库可以开发全文检索功能。

## 全文检索技术的应用场景

使用全文检索技术可以实现搜索引擎（百度、google。。），搜索引擎可以搜索互联网上所有的内容（网页、pdf电子书、视频、音乐）。

Lucene和搜索引擎的区别：搜索引擎是对外提供全文检索服务，是可以单独运行。Lucene只是一个工具包不能单独运行的，需要在project中加入lucene的jar包，最终project在JVM运行。

使用全文检索技术可以实现站内搜索，站内搜索只能搜索本网站的信息（网页、pdf电子书、视频、音乐、关系数据库中的信息等等），比如：电商网站搜索商品信息，论坛网站搜索网内帖子。。。

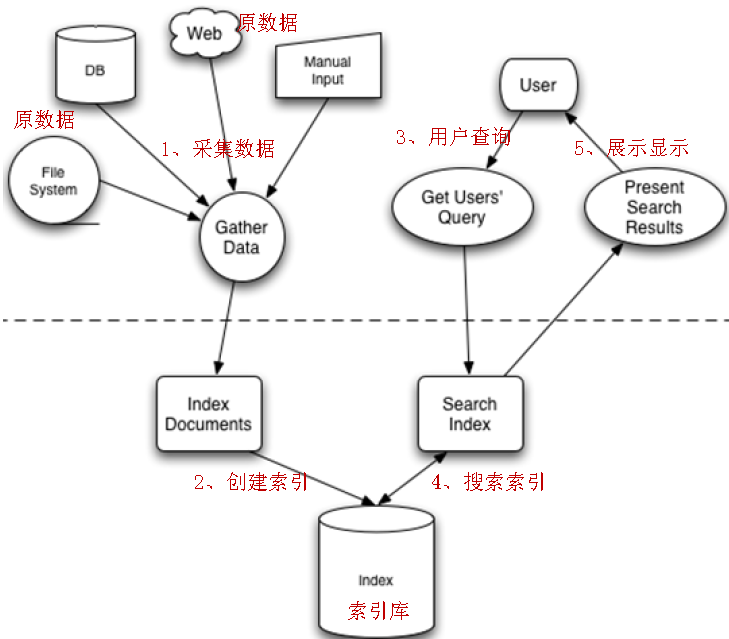
## Lucene实现全文检索过程--概括

全文检索的定义：

全文检索首先将要搜索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）**

如下图Lucene实现全文检索过程（全文检索实现过程）：



全文检索过程分为 索引、搜索两个过程。

索引：

1、从关系数据库中、互联网上、文件系统采集源数据(要搜索的目标信息)

源数据的来源是很广泛的。

2、将源数据采集到一个统一的地方，要创建索引，将索引创建到一个索引库（文件系统）中

从源数据库中提取关键信息，从关键信息中抽取一个一个词，词和源数据是有关联。

索引库中就是一个一个的库。

搜索：

3、用户执行搜索（全文检索）编写查询关键（全文检索查询语法，类似关系数据库中的sql）

4、从索引库中搜索索引，根据查询关键字搜索索引库中的一个一个词

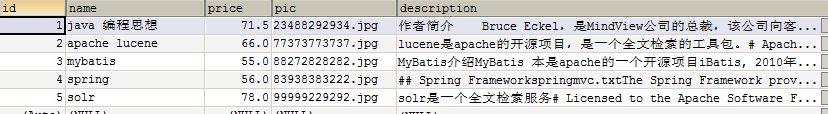
5、展示搜索的结果。

创建索引时词和源数据有关联，索引库中记录了这个关联，如果找到了词就说明找到了源数据（http的网页、pdf电子书）。

## 需求

要用Lucene实现站内搜索，要搜索图书信息。

图书信息在关系数据库中：



## 环境准备

mysql：5.1

java：1.7

Lucene：

Lucene是开发全文检索功能的工具包，从官方网站下载Lucene4.10.3，并解压，lucene的jar加入工程中使用即可。

官方网站：<http://lucene.apache.org/>

版本：lucene4.10.3

使用的jar：

mysql5.1驱动包：mysql-connector-java-5.1.7-bin.jar

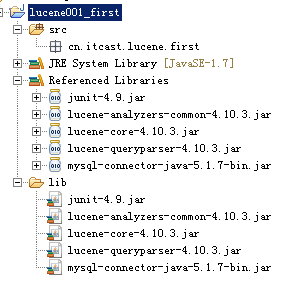
lucene核心包：lucene-core-4.10.3.jar

lucene分析器通用包：lucene-analyzers-common-4.10.3.jar

lucene查询解析器包：lucene-queryparser-4.10.3.jar

junit包：junit-4.9.jar

开发工具：eclipse indigo



## 索引部分

### 为什么要采集数据？

全文检索要搜索的源数据的格式是多种多样的，比如：互联网站上的网页(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

如何搜索结构的数据？

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是根据关键字去搜索的内容中顺序扫描关键字。

select \* from book where description like '%java%'

description 的内容很多，如果要顺序扫描的算法，速度很慢的。

如何搜索非结构的数据？

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。

### 如何采集数据

针对不同的源数据，使用不同的技术进行采集：

1、针对互联网上的数据，使用http协议抓取html网页到本地，生成一个html文件。

2、针对关系数据库中的数据，连接数据库读取表中的数据。

3、针对文件系统中的数据，通过流读取文件系统的文件。

以上技术中使用第一种较多，因为目前全文检索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（了解）：

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。

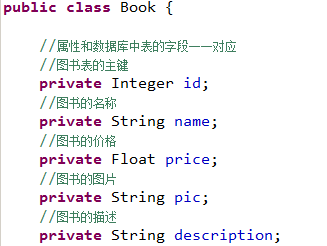
heritrix（http://sourceforge.net/projects/archive-crawler/files/），Heritrix 是一个由 java 开发的、开源的网络爬虫，用户可以使用它来从网上抓取想要的资源。其最出色之处在于它良好的可扩展性，方便用户实现自己的抓取逻辑。

### 采集数据Dao

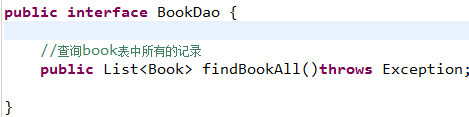
针对本例子的需求，从关系数据库中采集 book表的数据。

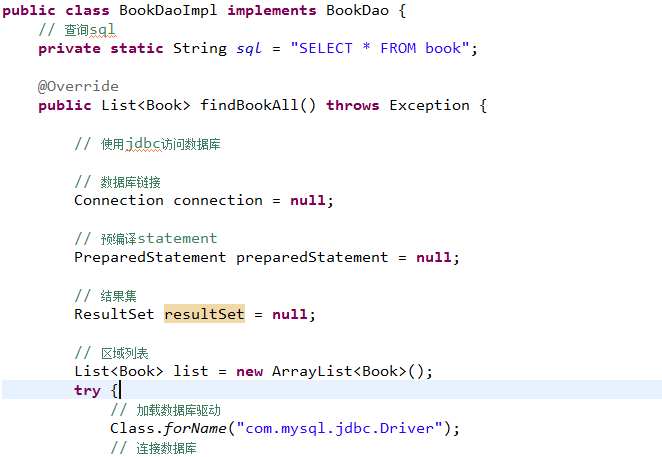
创建dao接口读取book表中的数据。

#### po实体类



#### Dao接口





### 索引文件逻辑结构

索引文件： 将源数据创建索引，最终以索引文件方式存储到文件系统中。

索引文件逻辑结构图：



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：

索引域的用于搜索的，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。

之所以根据词可以找到文档，是因为词是从document中的Field内容抽取出来的。

将Document中的Field的内容进行分词，将分好的词创建索引，索引=Field域名:词（表示从document中的哪个Field抽取的词）

### 创建Docuemnt

使用lucnene的api创建Document。

一个document对应一条book表的记录。

代码如下：

//一个document对应一条book表的记录

//定义List存储document

List<Document> docs = **new** ArrayList<Document>();

//调用dao获取book数据

List<Book> bookList = **new** BookDaoImpl().findBookAll();

**for**(Book book:bookList){

//创建Document

Document doc = **new** Document();

//创建Field域

//图书id的field

//参数：域名、field中存储的value值，是否存储

TextField id = **new** TextField("id", book.getId().toString(), Store.*YES*);

//图书名称的field

TextField name = **new** TextField("name", book.getName(), Store.*YES*);

//图书价格

FloatField price = **new** FloatField("price", book.getPrice(), Store.*YES*);

//图书图片

TextField pic = **new** TextField("pic", book.getPic(), Store.*YES*);

//图书的描述

TextField description = **new** TextField("description", book.getDescription(), Store.*YES*);

//将field加入document中

doc.add(id);

doc.add(name);

doc.add(price);

doc.add(pic);

doc.add(description);

//将一个doc加入List<document>中

docs.add(doc);

}

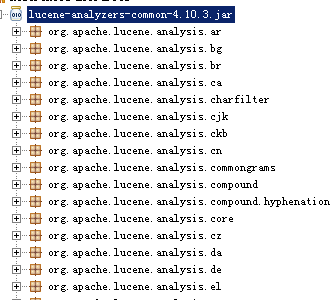
### 分词

在对Docuemnt中的内容索引之前需要经过分词、过虑两步。

分词就是将原始文档内容切分成一个一个的词也就是将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参、过去式转成现在式。。。）等。

Lucene作为了一个工具包提供不同国家的分词器，如下图：



本例子使用StandardAnaylzer标准分词器，支持英语内容分词。

### 创建索引

索引过程：

索引目录流对象

Directory

索引操作对象

IndexWriter

索引库

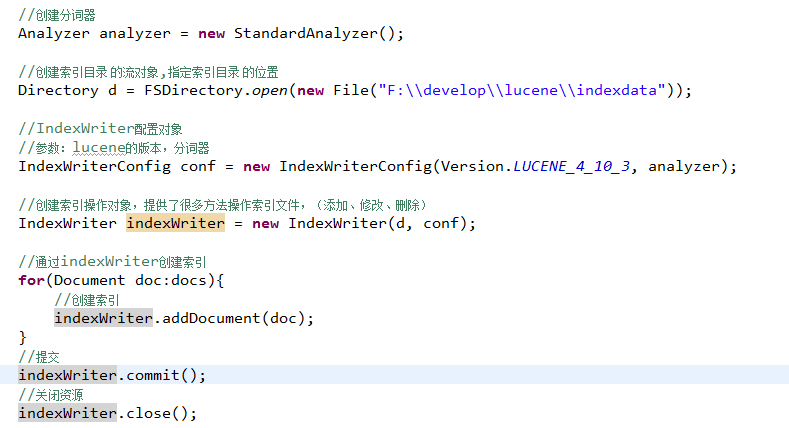
文档

Document

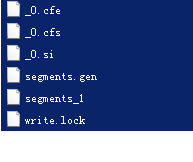
分词器

Analyzer

代码：



创建索引成功：



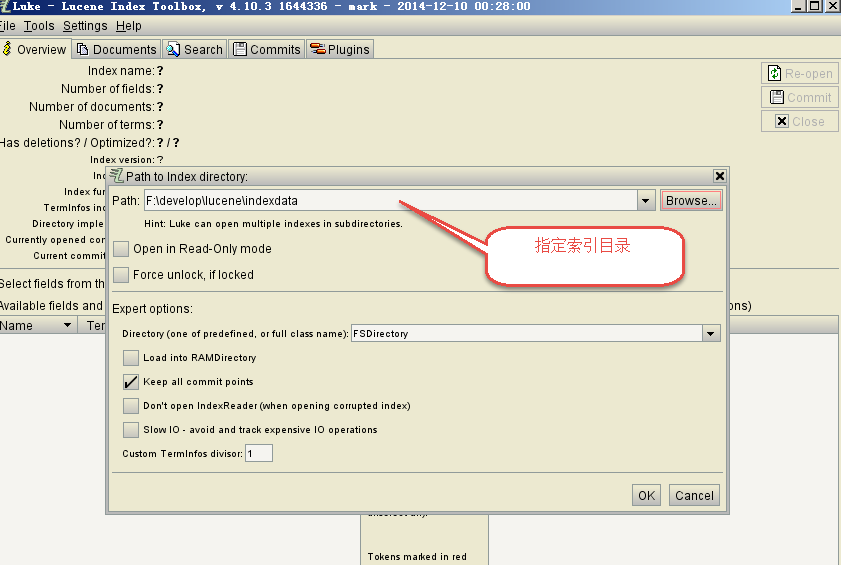
### 使用Luke查询索引文件内容

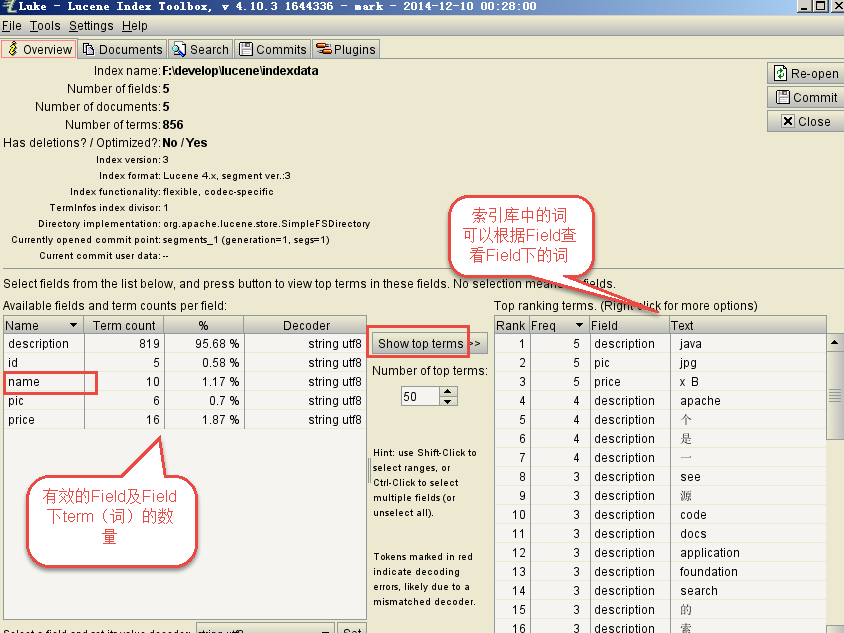
Luke作为Lucene工具包中的一个工具（http://www.getopt.org/luke/），用于查询、修改lucene的索引文件。

打开Luke方法：

cd luke所在目录

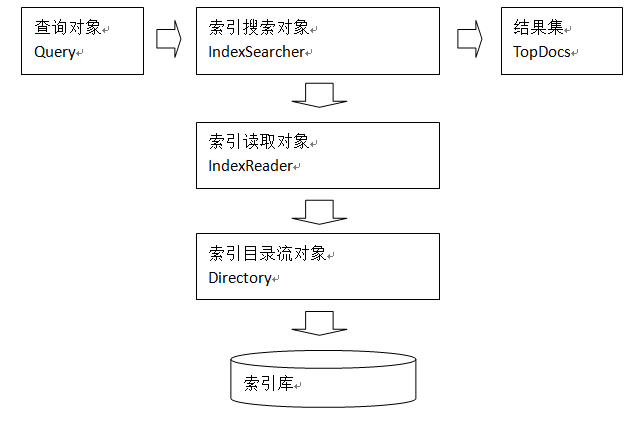






## 搜索部分

### 搜索过程



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）

指定查询语法，相当于sql语句。

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、通过IndexReader索引读取对象，对应的索引维护对象IndexWriter

IndexReader读取索引目录 的索引文件

4、IndexReader需要索引流对象读取索引目录内容，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

通过Query查询对象输入查询语句。

同数据库的sql一样，lucene全文检索也有固定的语法：

最基本的有比如：AND, OR, NOT 等

举个例子，用户输入语句：description:java AND lucene

意义：查询description域内容有java并且有lucene的文档。

### 搜索语法分析

由于查询语句为特殊的语法所以lucene会首先对语法进行分析，判断语法关键字的正确性，比如NOT、AND、OR，分别抽取出语法关键字和搜索关键字。

### 分词

和索引过程的分词一样，这里要对用户输入的关键字进行分词，一般情况索引和搜索使用的分词器一致。

比如：输入搜索关键字“java培训”，分词后为java和培训两个词，与java和培训有关的内容都搜索出来了，如下：



搜索过程使用的分词器要和索引时使用的分词器一致。

### 搜索代码

@Test

**public** **void** searchIndex() **throws** ParseException, IOException{

//分词,搜索过程使用的分词器要和索引时使用的分词器一致

Analyzer analyzer = **new** StandardAnalyzer();

//查询分析器

//第一个参数：指定默认搜索的域,第二个：分词器

QueryParser queryParser = **new** QueryParser("description", analyzer);

//创建查询对象

Query query = queryParser.parse("description:java");

//创建索引目录 的流对象,指定索引目录 的位置

Directory d = FSDirectory.*open*(**new** File("F:\\develop\\lucene\\indexdata"));

//索引读取对象

//指定读取索引的目录

IndexReader indexReader =IndexReader.~~open~~(d);

//索引搜索对象

IndexSearcher indexSearcher =**new** IndexSearcher(indexReader);

//执行搜索

//第一个参数：query查询对象，第二个：取出匹配度高的前100条

TopDocs topDocs = indexSearcher.search(query, 100);

//取出匹配上的文档

ScoreDoc[] scoreDocs = topDocs.scoreDocs;

**for**(ScoreDoc scoreDoc:scoreDocs){

//document的id

**int** docID = scoreDoc.doc;

//从indexReader根据docID获取document

Document doc = indexReader.document(docID);

//取出doc中的field域的内容

//参数指定field域名

String id = doc.get("id");

String name = doc.get("name");

Float price = Float.*parseFloat*(doc.get("price"));

String pic = doc.get("pic");

System.*out*.println("=====================");

System.*out*.println("图书的id："+id);

System.*out*.println("图书的name："+name);

System.*out*.println("图书的价格："+price);

System.*out*.println("图书的图片："+pic);

}

//关闭资源

indexReader.close();

}

# Lucene深入

## Field属性

### 需求

由于图书的描述信息内容量大，实际开发中不会在搜索结果页面显示图书描述，而会在图书的详情页面显示所有描述，所以图书描述不需要存储在document，这样可以节省索引空间；

由于用户输入的关键字会根据图书描述信息搜索所以需要对图书描述信息进行分词并且索引，因为分词的目的是为了索引，而索引的目的是为了搜索。

用户输入关键字不会匹配图片，所以不用对图书图片地址进行分词也不用进行索引，由于图书图片字段存储了图片的地址其目的是为了在搜索结果页面显示图片，所以需要在document中存储图片地址，这样才能从搜索的document中取出图片从而在页面显示。

### Field属性

* 是否分词(Tokenized)

是：分词的目的是为了索引

否：当搜索关键要匹配整个field域的值时，就不分词，例如：图书id主键，订单号，身份证号

* 是否索引(Indexed)

是：索引的目的是为了搜索

否：图片地址，因为不会根据图片地址来搜索

* 是否存储(Stored)

是：将field的内容存储到索引文件中，将来搜索出来document，从索引文件（lucene）中取出来，存储的目的是为了将来搜索到document，从lucene取出值。

否：不在查询结果页面显示的就不用存储，比如：图书描述，商品简介。

图书id：

是否分词：不用分词，因为主键是一个整体，搜索关键字只会匹配主键这个整体来查询

是否索引：因为程序中可能会根据图书的id主键搜索document所以要索引

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

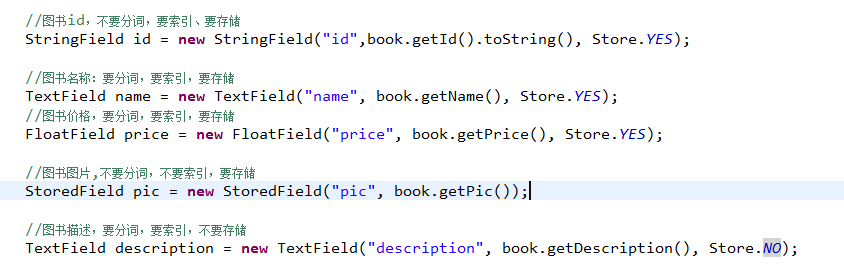
是否存储：因为图书描述内容量大，不在查询结果页面直接显示，不存储。

不存储是来不在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。

### 代码实现

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



## 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。

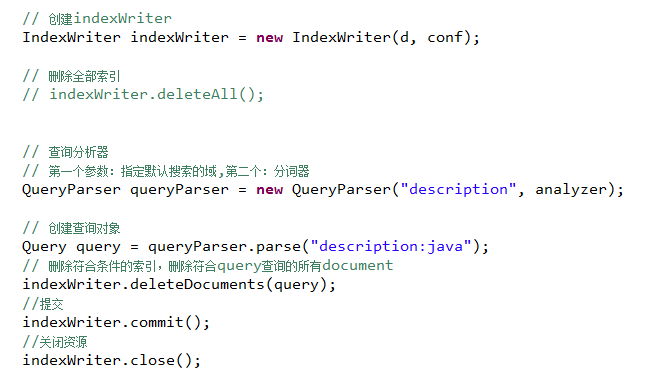
### 添加索引

调用 indexWriter.addDocument（doc）添加索引。

详细代码略

### 删除索引

1、删除符合条件的索引



2、删除全部记录

indexWriter.deleteAll()

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

建议：删除索引时根据主键的Field进行删除，以免误删除数据。本例子主键Feild 即id删除。

### 更新索引

更新思路：先查询、再删除、再添加。

建议根据主键field查询document，避免误更新。



## 搜索

### Query查询方法

用户使用lucene搜索，需要定义自己的查询语法。

lucene中提供两大查询：

1、一种方法直接query对象方法，直接使用query对象构造查询语法

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

如下代码：

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

2、另一种方法使用QueryParser查询解析器，构造查询语法

手动编写查询语法：

如下代码：

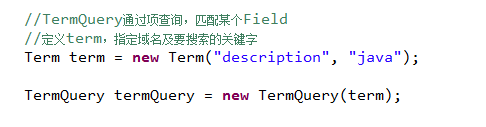
QueryParser queryParser = **new** QueryParser("name", **new** StandardAnalyzer());

Query query = queryParser.parse("name:lucene");

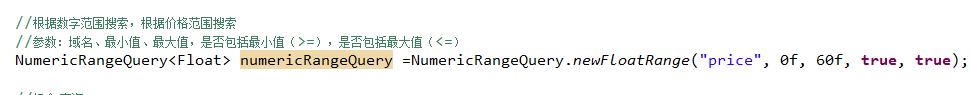
QueryParser方法最直接，开发中常用 的方法。

### 直接使用query对象查询

TermQuery通过项查询，匹配某个Field



NumericRangeQuery 数字范围搜索：



BooleanQuery，布尔查询，实现组合条件查询：



### QueryParser

* 项查询：

FieldName : value

* 范围查询：

FieldName:[min TO max]

注意：QueryParse不支持对数字范围的搜索，它支持字符串范围。数字范围搜索建议使用NumericRangeQuery。

* 组合查询：

上边BooleanQuery例子的查询表达式如下：

+price:[0 TO 200] +name:java

上边的表达式表示price的值在0和200之间且name为“java”，必须满足的条件使用+（加号）表示。

与BooleanQuery中Occur对应的符号如下：

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

* 关键字查询

AND：关键字1 AND 关键字2

两个关键字都匹配上条件满足。

OR：关键字1 OR 关键字2

两个关键字匹配一个条件满足

NOT：关键字1 NOT 关键字2

关键字1满足，关键字2不满足

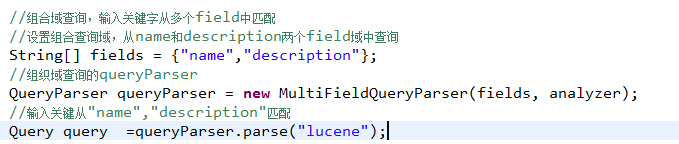
技巧：

通过System.out.println(query对象);查看query内部执行的语法 。

建议在实际开发中使用queryParser查询。

### MultiFieldQueryParser(常用)

通过MuliFieldQueryParse对多个域查询，比如商品信息查询，输入关键字需要从商品名称和商品内容中查询。



### TopDocs

Lucene搜索结果可通过TopDocs遍历，TopDocs类提供了少量的属性，如下：

|  |  |
| --- | --- |
| 方法或属性 | 说明 |
| totalHits | 匹配搜索条件的总记录数 |
|  |  |
|  |  |
|  |  |
| scoreDocs | 顶部匹配记录，匹配度高的记录 |

search(query, n)，n表示取出匹配度高的前n条记录。

totalHits：匹配到索引库的document总数，不受search(query, n)中的n值的影响。

topDocs.scoreDocs：匹配度高的记录，scoreDocs的大小是小于等于受search(query, n)中的n值，

scoreDocs的大小是小于等于totalHits

**实际开发中，页面上是不显示totalHits，页面上要显示的是匹配度高的记录。**

## 相关度排序

### 什么是相关度

相关度排序是查询结果按照与查询关键字的相关性进行排序，越相关的越靠前。比如搜索“Lucene”关键字，与该关键字最相关的文档应该排在前边。

在进行关键搜索时Lucene根据关键和文档相关度打一个分，最终的查询结果根据分值降序排序。分值越高相关度越高。

可以人为去干预打分，影响结果排序，由于某些市场营销需求需要去干预打分。

### 相关度打分

就排在前边。如何打分呢？Lucene是在用户进行检索时实时根据搜索的关键字计算出来打分，影响 打分的因素：

计算出词（Term）的权重：

根据Term找到文档，Term对文档的重要性称为权重，影响Term权重有两个因素：

* Term Frequency (tf) ：

指此Term在此文档中出现了多少次。tf 越大说明权重越大。词(Term)在文档中出现的次数越多，说明此词(Term)对该文档越重要，如“Lucene”这个词，在文档中出现的次数很多，说明该文档主要就是讲Lucene技术的。

* Document Frequency (df)

有多少文档包含次Term。df 越大说明权重越小，因为这个term可能是一个停止词。

比如，在一篇英语文档中，this出现的次数更多，就说明越重要吗？不是的，有越多的文档包含此词(Term), 说明此词(Term)太普通，不足以区分这些文档，因而重要性越低。

Lucene使用空间向量模型的算法计算相关度打分，根据搜索关键字和文档的相关度来计算打分，打分前要计算机词的权重(Term对文档的重要性)。

### 设置boost值影响 打分

可以手动设置boost值影响相关度打分。

**总结影响 相关度打分因素：**

**1、 词本身对于文档的权重**

**2、boost值**

**通过从上边两个因素入门最终干预搜索结果排序。**

boost是一个加权值（默认加权值为1.0f），它可以影响权重的计算。

#### 索引时设置boost值

可以将某个document的description加权值设置为100.0f，结果搜索java时如果内容可以匹配到关键字就可以把该Document排在前边。



#### 搜索时设置boost值

在执行搜索时对某个域进行加权，在进行**组合域查询**时，匹配到加权值高的域最后计算的相关度得分就高。通常把标题、书名等域的加权值设置高点。

比如：根据关键字从图书名称、图书描述中搜索，如果名称匹配上表示图书与关键字相关度很高。

在搜索时，设置哪个域的boost 权值高。

通过MultiFieldQueryParser完成组合域搜索。



实际在开发中，一定根据Field的内容意义决定应该对哪个域的boost加权。

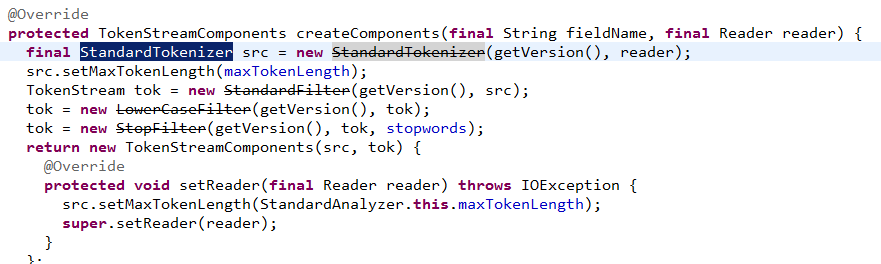
比如：图书搜索，将图书名称加权，网站的搜索，将图站的title加权。

## 中文分词器

### Analyzer执行过程

Analyzer执行过程分为分词、过虑两个过程。

源代码：



TokenStream是语汇单元流，tokenStream是一个抽象类，它是所有分词器的基类，如下图：

tokenStream

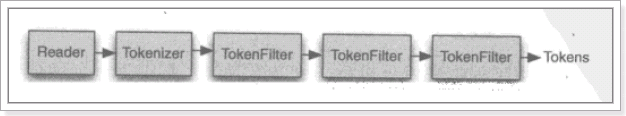
Tokenizer（分词器）

TokenFilter（分词过滤器）

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儿，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。

### 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。

中文分词器是按照词库进行分词，我爱中国分词后是：我、爱、中国，如果将“我爱中国”加入词库，分词后的结果是：我、爱、中国、我爱中国 四个词。

### Lucene自带中文分词器

* 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。

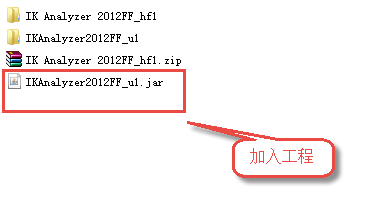
### 第三方中文分词器

使用ik-analyzer：

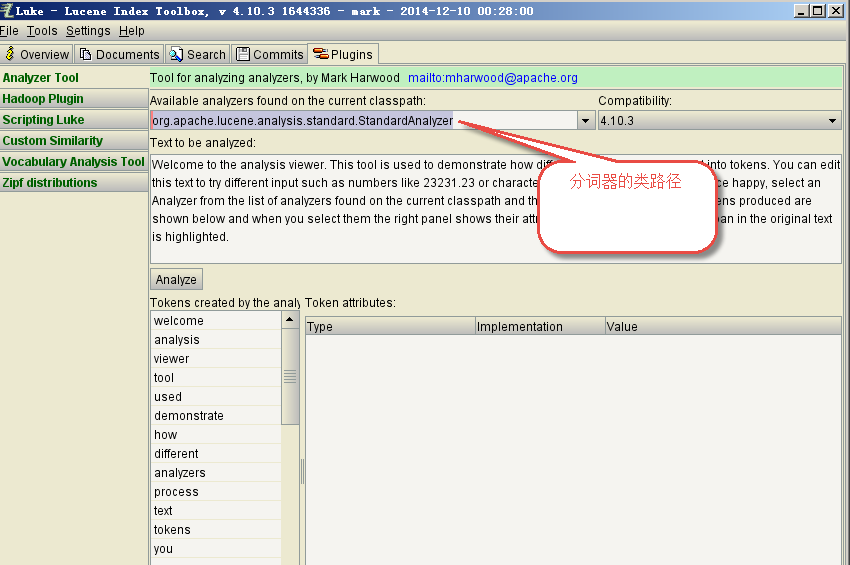
* ik-analyzer：

IK Analyzer是一个开源的，基于java语言开发的轻量级的中文分词工具包。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。

下载：ik-analyzer



### 使用luke测试中文分词

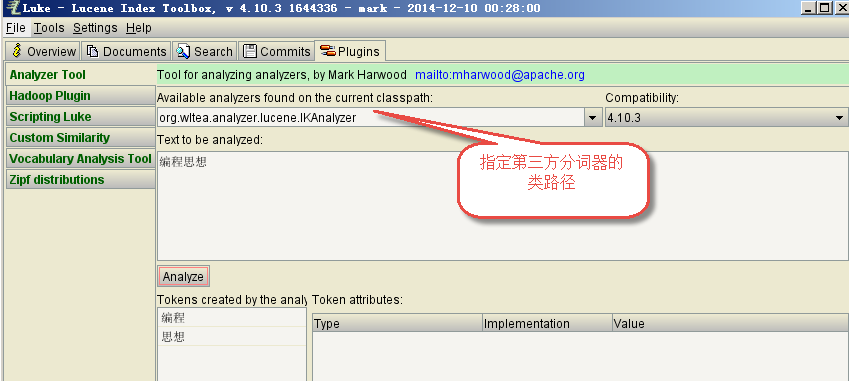


使用Luke测试第三方分词器分词效果，需通过java.ext.dirs加载jar包:

可简单的将第三方分词器和lukeall放在一块儿，cmd下运行：



java -Djava.ext.dirs=. -jar lukeall-4.10.3.jar



### 如何使用中文分词器

如果使用中文分词器ik-analyzer，就在索引和搜索程序中使用一致的分词器ik-analyzer。

索引使用中文器：



搜索使用中文分词器：



### 扩展中文分词器词库

在classpath下定义IKAnalyzer.cfg.xml文件，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 -->

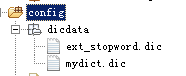
<entry key=*"ext\_dict"*>dicdata/mydict.dic</entry>

<!-- 用户可以在这里配置自己的扩展停用词字典 -->

<entry key=*"ext\_stopwords"*>dicdata/ext\_stopword.dic</entry>

</properties>

在classpath下的编辑dicdata/mydict.dic文件，此文件中存储扩展词库，在dicdata/ext\_stopword.dic文件中存放停用词。



注意：mydict.dic文件的编码格式 为utf-8 而不是 utf-8 + bom格式！！！！

