

**Lucene**

**全文检索技术**

# 课程计划

1. 什么是全文检索

结构化数据和非结构化数据

对结构化数据的搜索

对非结构化数据的搜索

什么是全文检索

全文检索应用领域

1. 什么是lucene

什么是lucene

Lucene与搜索引擎区别

安装lucene

Lucene主要包结构

1. Lucene入门程序： (重点)
2. 需求：使用Lucene完成图书信息的全文检索功能开发。
3. 互联网搜索引擎架构
4. 索引程序

为什么要采集数据

如何采集数据

创建文档Document

如何创建索引

分词

创建索引

1. 使用Luke查看索引
2. 搜索程序

输入查询语句

搜索语法分析

分词

搜索索引

1. 索引：

Field域属性学习(重点)

索引维护（添加、修改、删除）(重点)

1. 搜索：

使用Query子类对象搜索

使用QueryParser进行搜索(重点)

1. 搜索结果TopDocs(重点)
2. 中文分词器(重点)

# 什么是全文检索

## 结构化数据和非结构化数据

我们生活中的数据总体分为两种：**结构化数据**和**非结构化数据**。

* **结构化数据：**指具有固定格式或有限长度的数据，如数据库中的数据，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。

## 对结构化数据的搜索

对数据库的搜索，用SQL语句。再如对元数据的搜索，如利用windows搜索对文件名，类型，修改时间进行搜索等。

## 对非结构化数据的搜索

1. **顺序扫描法(Serial Scanning)**

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

1. **全文检索(Full-text Search)**

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

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

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

## 什么是全文检索

**全文检索是一种将文件中所有文本与检索项匹配的文字资料检索方法。**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）。**

全文检索就是把文本中的内容拆分成若干个关键词，然后根据关键词创建索引。查询时，根据关键词查询索引，最终找到包含关键词的文章。整个过程类似于查字典的过程。

如下图是对文件搜索的索引：

索引

lucene

java

apache

…

…

目标文档

dataset

XXXXXX.doc、XXXXX.xls、XXXXXX.pdf

上图中，索引栏是从目标数据源中提取出来的词，在进行全文检索时是通过搜索索引（搜索索引中的词）从而找到索引对应的文件即目标数据源。

经过几年的发展，全文检索从最初的字符串匹配程序已经演进到能对超大文本、语音、图像、活动影像等非结构化数据进行综合管理的大型软件。本教程只讨论文本检索。

## 全文检索的应用领域

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

使用全文检索技术可以实现如下功能：

搜索引擎：



站内搜索：



# 什么是Lucene

## 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。提供了完整的查询引擎和索引引擎，部分文本分析引擎。Lucene的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能。

## Lucene与搜索引擎的区别

全文检索系统是按照全文检索理论建立起来的用于提供全文检索服务的软件系统。全文检索系统是一个可以运行的系统，包括建立索引、处理查询返回结果集、增加索引、优化索引结构等功能。例如：百度搜索、eclipse帮助搜索、淘宝网商品搜索。

搜索引擎是全文检索技术最主要的一个应用，例如百度。搜索引擎起源于传统的信息全文检索理论，即计算机程序通过扫描每一篇文章中的每一个词，建立以词为单位的倒排文件，检索程序根据检索词在每一篇文章中出现的频率和每一个检索词在一篇文章中出现的概率，对包含这些检索词的文章进行排序，最后输出排序的结果。全文检索技术是搜索引擎的核心支撑技术。

Lucene和搜索引擎不同，Lucene是一套用java或其它语言写的全文检索的工具包，为应用程序提供了很多个api接口去调用，可以简单理解为是一套实现全文检索的类库，搜索引擎是一个全文检索系统，它是一个单独运行的软件。

## 安装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。

官方网站：<http://lucene.apache.org/>

版本：lucene4.10.3

Jdk要求：1.7以上

## Lucene主要包结构

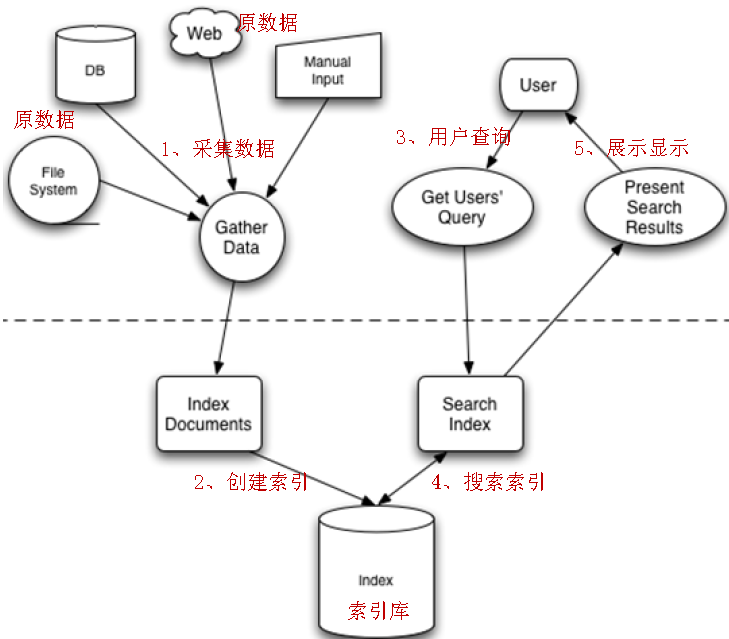
|  |  |
| --- | --- |
| 包名 | 功能 |
| org.apache.lucene.analysis | 语言分析器，主要用于的切词  Lucene提供的分析器实现类在：  lucene-analyzers-common-4.10.3.jar |
| org.apache.lucene.document | 索引存储时的文档结构管理，类似于关系型数据库的表结构 |
| org.apache.lucene.index | 索引管理，包括索引建立、删除等 |
| org.apache.lucene.queryParser | 查询分析器，实现查询关键词间的运算，如与、或、非等, 生成查询表达式， |
| org.apache.lucene.search | 检索管理，根据查询条件，检索得到结果 |
| org.apache.lucene.store | 数据存储管理，包括一些I/O操作 |
| org.apache.lucene.util | 公用类 |

# Lucene全文检索过程入门

## 需求

我们对全文检索的分析以一个电商网站的站内搜索为例进行分析，本例子以搜索图书为例进行讲解。

## 互联网搜索引擎系统架构



整个全文检索过程分为两个过程，索引和搜索。

**1、采集数据**

将关系数据库中的数据、互联网的数据、文件系统中文件等原始数据采集到某一个地方。

**2、创建索引**

对第一步采集到的数据创建索引，将索引文件存储到文件系统（可以将索引文件存储到内存中）中。

创建索引之前需要将原始数据中内容抽取一个一个词，对词创建索引。

**3、定义查询语句**

要想根据自己的需求去搜索索引库，定义好查询语句（类似sql语言）。

**4、搜索索引**

拿着上一步定义查询语句去搜索索引。

**5、解析搜索的结果**

解析出搜索结果按照用户的需求进行展示，将结果进行排序（搜索结果已经按照相关度进行排序），对结果中关键字进行高亮显示。

电商系统中商品搜索的技术架构如下：

Java Application

Lucene

索引库（存储索引）

Mysql数据库（存储商品信息）

JDBC

Mysql驱动

api调用

api调用

本地IO流水

TCP

## 准备环境

jdk1.7

lucene-4.10.3

mysql 5.1

eclipse indigo

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

创建java工程加入以上jar 包。

## 【索引过程】

### 为什么要采集数据？

全文检索要搜索的数据信息格式多种多样，拿搜索引擎（百度， google）来说，通过搜索引擎网站能搜索互联网站上的网页(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。**

### 如何采集数据？

采集数据技术有哪些？

1、对于互联网上网页采用http将网页抓取到本地生成html文件。

2、如果数据在数据库中就连接数据库读取表中的数据。

3、如果数据是文件系统中的某个文件，就通过文件系统读取文件的内容。

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和一些爬虫所用到的技术框架（了解）：

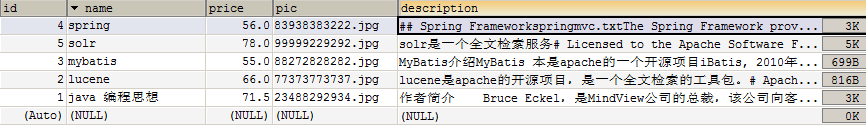
Solr（<http://lucene.apache.org/solr>） ，solr是apache的一个子项目，支持从关系数据库、xml文档中提取原始数据。

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。

针对电商站内搜索功能，全文检索的数据源在数据库中，需要通过jdbc访问数据库中book表的内容。



Book.java

|  |
| --- |
| **public** **class** Book {    **private** Integer id;  **private** String name;  **private** Float price;  **private** String pic;  **private** String description; |

创建Dao负责采集数据库中的数据：

**public** **interface** BookDao {

//图书查询

**public** List<Book> findBookList()**throws** Exception;

}

**public** **class** BookDaoImpl **implements** BookDao {

// 查询sql

**private** **static** String *sql* = "SELECT \* FROM book";

@Override

**public** List<Book> findBookList() **throws** Exception {

// 数据库链接

Connection connection = **null**;

// 预编译statement

PreparedStatement preparedStatement = **null**;

// 结果集

ResultSet resultSet = **null**;

// 区域列表

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/lucene", "root", "mysql");

// 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**){

resultSet.close();

}

**if**(preparedStatement!=**null**){

preparedStatement.close();

}

**if**(connection!=**null**){

connection.close();

}

}

**return** list;

}

}

### 创建文档Document

Document是Lucene对索引对外表示的逻辑结构，Document采用NOSql的存储理念，每个Document中有很多Field组成，Field中存储key/value对即name:XXX，value：XXX

创建索引前要将要索引的内容用Document表示，这一步就相当于提取非结构数据的信息然后结构化为Document形式 。

创建索引就是针对Document中的内容进行索引，确切的说是针对Document中Field域中的内容进行索引。

BookDao bookDao = **new** BookDaoImpl();

// 从数据库中查询图书信息

List<Book> bookList = bookDao.findBookList();

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

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

//document文档

Document document = **new** Document();

//商品id

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

Store.*YES*);

//商品名称

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

//商品价格

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

Store.*YES*);

//商品图片

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

//商品描述

TextField description = **new** TextField("description",

book.getDescription(), Store.*YES*);

//将Field添加到文档中

document.add(id);

document.add(name);

document.add(price);

document.add(pic);

document.add(description);

docs.add(document);

}

### 如何创建索引？(图)

1. **索引文件逻辑结构：**



* 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。

文档域作用：将非结构化数据进行统一结构化，将document中field中的value值进行分词。

* 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=Field域名:词

* 倒排索引表

传统方法是已知文件在文件中找内容，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**是根据关键字（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。**

1. **创建索引流程：**

通过索引流对象执行I/O

Directory

通过索引操作对象创建索引

IndexWriter

索引库

创建文档

Document

对文档内容进行分词

Analyzer

**1、对原始内容信息抽取，并创建Document文档。**

**2、使用Analyzer分词器对Document文档中的内容进行分词。**

**3、通过IndexWriter创建索引**

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。

**4、通过Directory索引流对象将索引文件写入文件系统**

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）。

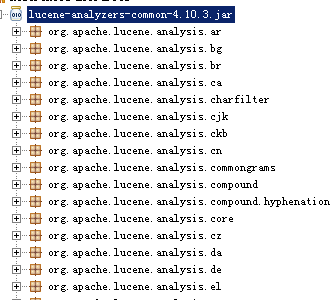
### 分词

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、词的形态还原（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：



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，标准分词器可以对英文内容进行分词，它先按照空格将内容切分成一个一个词，再去除a/an/the等停用词、大写转小写、词语形态变换（比如将名称复数形式转成单数形式等）：

如下：

原内容：

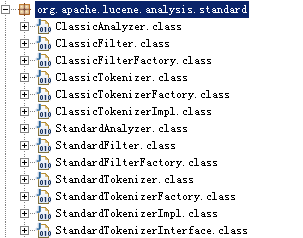
Lucene is a Java full-text search engine. Lucene is not a complete

application, but rather a code library and API that can easily be used

to add search capabilities to applications.

分词、过虑后得到的词：

lucene、java、full、search、engine。。。。。



代码如下：

// 创建标准分析器

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

### 创建索引

// 创建标准分词器

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

// 索引操作配置信息

IndexWriterConfig indexWriterConfig = **new** IndexWriterConfig(

Version.*LUCENE\_4\_10\_3*, standardAnalyzer);

// 创建索引目录流对象

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

// 定义索引操作对象

IndexWriter indexWriter = **new** IndexWriter(directory, indexWriterConfig);

// 遍历目录 下的文件生成的文档，调用indexWriter方法创建索引

**for** (Document document : docs) {

indexWriter.addDocument(document);

}

//提交

indexWriter.commit();

// 索引操作流关闭

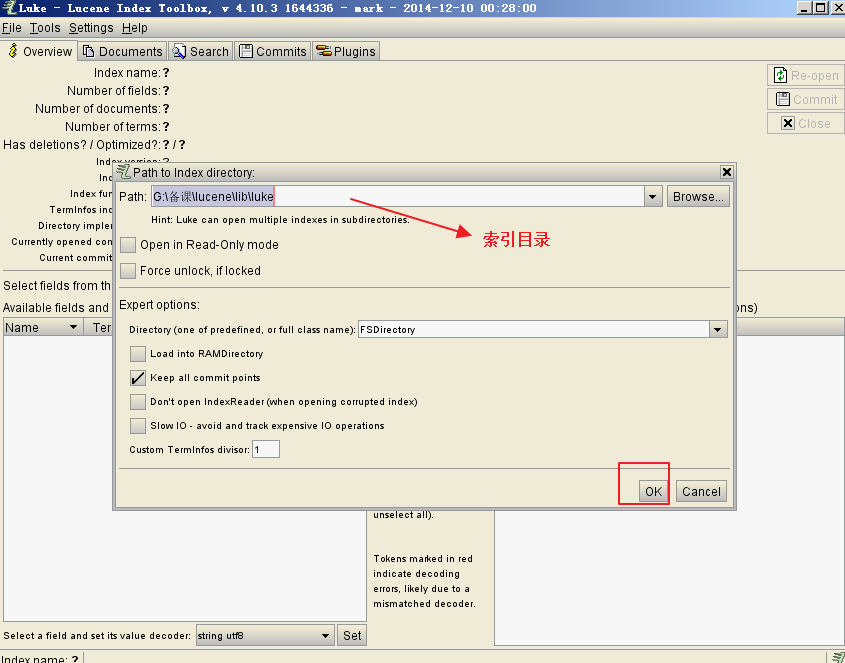
indexWriter.close();

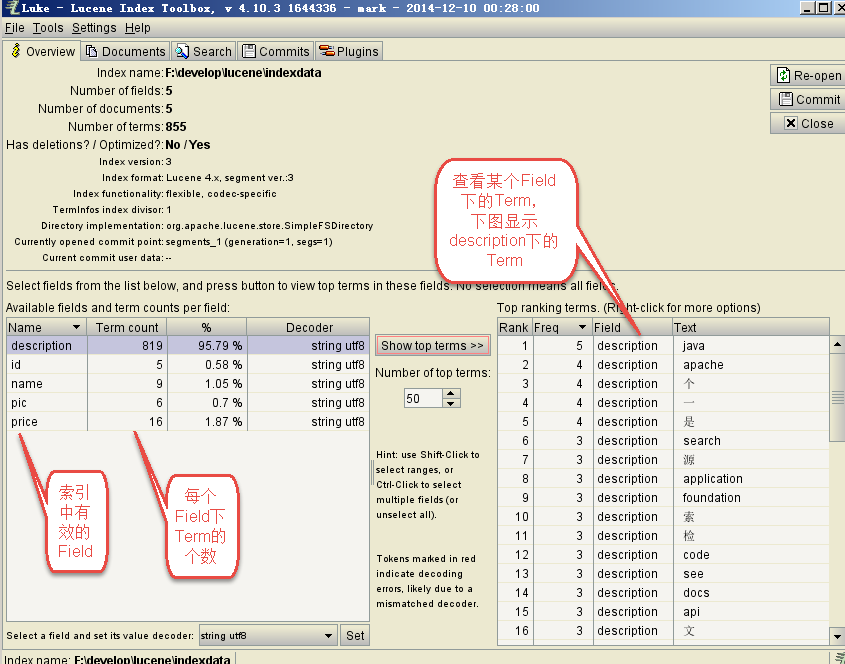
## 使用Luke查看索引

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

打开Luke方法：

cmd运行：java -jar lukeall-4.10.3.jar





## 【搜索过程】

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

Directory

索引库

创建查询对象

Query

搜索结果

TopDocs

1、创建查询对象query，这里就是指定搜索语句，就相当于查询关系数据要输入sql语句。

2、通过索引搜索对象IndexSearcher搜索索引

3、IndexSearcher通过IndexReader读取索引文件中Document。

4、IndexReader最终通过Directory索引流对象执行I/O操作从索引文件中读取内容。

5、IndexSearcher搜索完成返回搜索结果TopDocs。

6、解析TopDocs的内容展示给用户。

### 输入查询语句

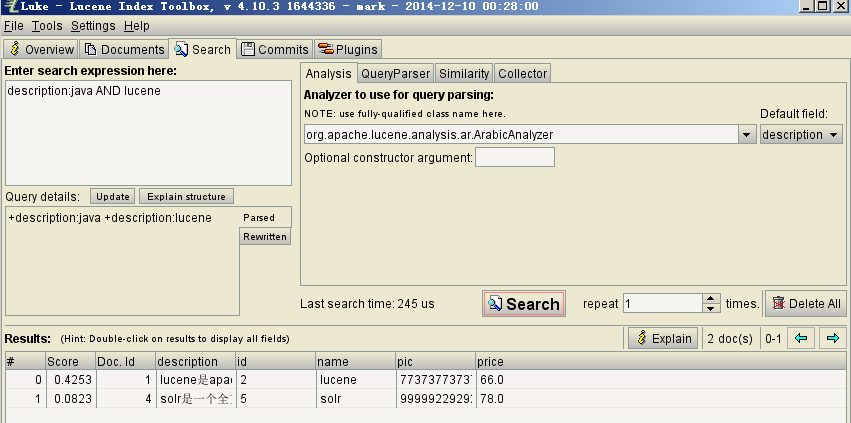
同数据库的sql一样，lucene全文检索也有固定的语法：

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

说明用户想找一个description中包括java关键字和lucene关键字的文档。

如下是使用luke搜索的例子：



### 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：



### 搜索索引

根据关键字从索引中找到索引信息，索引即词term，term与document相关联，找到了term就找到了关联的document，从document取出Field中的信息即是要搜索的信息。

代码如下：

//搜索索引

@Test

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

//分析器

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

//查询解析器

//第一个构造参数是默认搜索的域

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

//搜索语法，由于queryParser中指定的默认搜索的域这里的filed名称可以省略

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

//索引目录流对象

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

//索引读取对象

IndexReader indexReader = IndexReader.*open*(directory);

//搜索对象

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

//执行搜索，取出符合记录的前100条

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

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

//文档id

**int** docID = scoreDoc.doc;

//根据id从IndexReader中获取document

Document doc = indexReader.document(docID);

//从document中获取Field的信息

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

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

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

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

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

System.*out*.println("文档id："+docID);

System.*out*.println("图书id："+id);

System.*out*.println("图书名称："+name);

System.*out*.println("图书价格："+price);

// System.out.println("图片描述："+description);

}

//关闭流

indexReader.close();

}

# 索引

## Field域的属性

### 索引问题

**是否需要分词**

分词就是将一句话拆分成一个一个的词，分词的目的是索引

**是否需要索引**

索引的目的是为了搜索，所以要看是否需要搜索

**是否需要存储**

存储的目的是为了显示，为了提取要显示的数据。

### 需求

根据数据库中的图书信息表结构，改造document中filed类型

### 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。

1. **是否分词(tokenized)**

是：将Field值分词其目的是将词进行索引

比如：图书名称、图书简介等，这些内容用户要输入关键字搜索，由于搜索的内容格式大、内容多需要分析后将语汇单元索引。

否：不作分词处理

比如：图书id、订单号、身份证号等

1. **是否索引(indexed)**

是：索引的目的是为了搜索，可以将Field分词后进行索引也可以对整个Field进行索引。

比如：图书名称、图书简介分词后进行索引，订单号、身份证号不用分词但也要索引，这些将来都要作为查询条件，如果将来查询要根据图书表的主键id查询，那图书id也要索引。

否：不索引无法搜索到

比如：图片路径等，不用作为查询条件的不用索引。

1. **是否存储(stored)**

是：将Field值存储在文档中，存储在文档中的Field才可以从Document中获取

比如：图书名称、订单号，凡是将来要从Document中获取的Field都要存储。

否：不存储Field值，不存储的Field无法通过Document获取

比如：图书简介，内容较大不用存储。如果要向用户展示商品简介可以从系统的关系数据库中获取商品简介，思路是从lucene索引获取图书id，根据图书id查询关系数据库。

### Field常用类型

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 | 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. |

### 创建Field代码如下

修改原来的创建Document代码：

//商品id，不分词，要索引，要存储

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

//商品名称，要分词，要索引，要存储

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

//商品价格，要分词，要索引，要存储

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

Store.*YES*);

//商品图片，不分词，不索引，要存储

StoredField pic = **new** StoredField("pic", book.getPic());

//商品描述，要分词，要索引，不存储

TextField description = **new** TextField("description",

book.getDescription(), Store.*NO*);

## 【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

同入门程序。

### 删除索引

需要指定条件删除。

最佳实践：删除文档时根据主键唯一标识去匹配删除。

1）删除符合条件的索引

//删除索引

@Test

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

// 创建标准分析器

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

// 索引操作配置信息

IndexWriterConfig indexWriterConfig = **new** IndexWriterConfig(

Version.*LUCENE\_4\_10\_3*, standardAnalyzer);

// 创建索引目录流对象

Directory directory = FSDirectory.*open*(**new** File(*indexFolder*));

// 定义索引操作对象

//第一个构造参数是默认搜索的域

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

//搜索语法，由于queryParser中指定的默认搜索的域这里的filed名称可以省略

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

//索引操作对象

IndexWriter indexWriter = **new** IndexWriter(directory, indexWriterConfig);

//删除符合搜索语法的索引记录

indexWriter.deleteDocuments(query);

//索引删除后无法恢复

indexWriter.commit();

// 关闭索引操作流

indexWriter.close();

}

说明：根据query删除索引，满足条件的将全部删除，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比如图书id，它对应数据库表中的主键，删除时根据此主键Field删除。

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。

2）删除全部索引

//删除全部索引

indexWriter.deleteAll();

说明：将索引目录的索引信息全部删除，直接彻底删除，无法恢复。

### 更新索引

流程：先删除后添加文档。

最佳实践：删除文档时根据主键唯一标识去匹配删除。

//创建term第一个参数是field，第二个参数是匹配的值

Term term =**new** Term("id", "1");

//更新document

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

//商品id，不分词，要索引，要存储

StringField id = **new** StringField("id","1",Store.*YES*);

//商品名称，要分词，要索引，要存储

TextField name = **new** TextField("name","java 编程思想第三版", Store.*YES*);

doc.add(id);

doc.add(name);

//更新索引

indexWriter.updateDocument(term, doc);

//提交

indexWriter.commit();

//关闭

indexWriter.close();

说明：更新索引是先删除再添加，根据term找到匹配field的document，并删除，再添加新的document到索引库。 建议将term设置为主键Field这样根据主键Field只会删除一个文档。

# 搜索

## Query查询

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

可通过两种方法创建查询对象：

1）使用Lucene提供Query子类

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

如下代码：

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

2）使用QueryParse解析查询表达式

QueryParse会将用户输入的查询表达式解析成Query对象实例。

如下代码：

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

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

## TermQuery

TermQuery通过项查询，匹配某个Field

//创建查询对象

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

//搜索索引 目录

Directory directory = FSDirectory.open(new File(indexFolder));

//定义IndexReader

IndexReader reader = DirectoryReader.open(directory);

//创建indexSearcher

IndexSearcher indexSearcher = new IndexSearcher(reader);

//执行搜索

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

## NumericRangeQuery

NumericRangeQuery，指定数字范围查询，如下：

//文件大小在0到1024的文件

NumericRangeQuery<Long> numericRangeQuery = NumericRangeQuery

.*newLongRange*("price", 0l, 200l, **true**, **true**);

## BooleanQuery

BooleanQuery，布尔查询，**实现组合条件查询**，如下：

// 数字范围查询

NumericRangeQuery<Long> numericRangeQuery = NumericRangeQuery

.newLongRange("price", 0l, 200l, true, true);

//定义Boolean查询

BooleanQuery booleanQuery = new BooleanQuery();

//必须满足price在0到200范围的条件

booleanQuery.add(numericRangeQuery, Occur.MUST);

//根据文件名搜索

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

//查询名称“java”

booleanQuery.add(query, Occur. MUST);

IndexReader reader = DirectoryReader.open(directory);

IndexSearcher searcher = new IndexSearcher(reader);

TopDocs topDocs = searcher.search(booleanQuery, 100);

说明：

Occur.MUST 查询条件必须满足，相当于and

Occur.SHOULD 查询条件可选，相当于or

Occur.MUST\_NOT 查询条件不能满足，相当于not非

Occur.MUST、Occur.MUST\_NOT、Occur.SHOULD通常组合使用，有以下6种组合：

1．MUST和MUST：取得两个查询子句的交集。

2．MUST和MUST\_NOT：表示查询结果中不能包含MUST\_NOT所对应得查询子句的检索结果。

3．SHOULD与MUST\_NOT：连用时，功能同MUST和MUST\_NOT。

4．SHOULD与MUST连用时，结果为MUST子句的检索结果,但是SHOULD可影响排序。

5．SHOULD与SHOULD：表示“或”关系，最终检索结果为所有检索子句的并集。

6． MUST\_NOT和MUST\_NOT：无意义，检索无结果。单独使用Occur.MUST\_NOT无结果。

## QueryParser

QueryParser使用方法：

//f是默认搜索的域

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

// 指定查询语法 ，如果不指定description就搜索默认的域

Query query2 = queryParser.parse("description:java");

或：

Query query2 = queryParser.parse("java");

上边介绍的基于类的查询方法，使用QueryParser可用下边的查询语法 实现，

* 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

## MultiFieldQueryParser

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。

代码：

//设置组合查询域

String[] fields = {"name","description"};

//创建查询解析器

QueryParser queryParser = new MultiFieldQueryParser(fields, new StandardAnalyzer());

//查询名称和内容中包括“java”关键字的文档

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

# 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,相关度前边的记录 |

注意：

Search方法需要指定匹配前n条记录：indexSearcher.search(query, n)，n表示指定要获取相关度高的记录数。

TopDocs.totalHits：是匹配索引库中所有记录的数量，不受search中n的影响

TopDocs.scoreDocs：匹配相关度高的前边记录数组，scoreDocs的长度小于等于search方法指定的参数n，scoreDocs的长度小于等于totalHits

# 中文分词器

## Analyzer执行过程

Analyzer是一个抽象类，在Lucene的lucene-analyzers-common包中提供了很多分析器，比如：org.apache.lucene.analysis.standard.standardAnalyzer标准分词器，它是Lucene的核心分词器，它对分析文本进行分词、大写转成小写、去除停用词、去除标点符号等操作过程。

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。

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

final StandardTokenizer src = new StandardTokenizer(getVersion(), reader);//创建分词器

src.setMaxTokenLength(maxTokenLength);

TokenStream tok = new StandardFilter(getVersion(), src);//创建标准分词过滤器

tok = new LowerCaseFilter(getVersion(), tok);//在标准分词过滤器的基础上加大小写转换过滤

tok = new StopFilter(getVersion(), tok, stopwords);//在上边过滤器基础上加停用词过滤

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

tokenStream

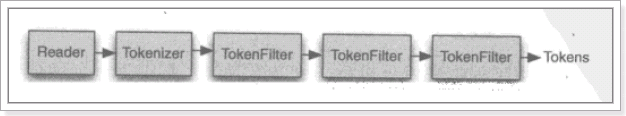
Tokenizer（分词器）

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。

## 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。

## 第三方产品介绍

* 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。

* mmseg4j

用 Chih-Hao Tsai 的 MMSeg 算法(http://technology.chtsai.org/mmseg/)实现的中文分词器，并实现 lucene 的 analyzer 和 solr 的TokenizerFactory 以方便在Lucene和Solr中使用。MMSeg 算法有两种分词方法：Simple和Complex，都是基于正向最大匹配。Complex 加了四个规则过滤。官方说：词语的正确识别率达到了 98.41%。mmseg4j 已经实现了这两种分词算法。

* imdict-chinese-analyzer

imdict-chinese-analyzer 是 imdict智能词典 的智能中文分词模块，算法基于隐马尔科夫模型(Hidden Markov Model, HMM)，是中国科学院计算技术研究所的ictclas中文分词程序的重新实现（基于Java），可以直接为lucene搜索引擎提供简体中文分词支持。

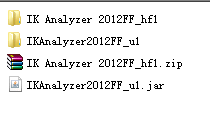
* ictclas4j

ICTCLAS4j中文分词系统是sinboy在中科院张华平和刘群老师的研制的FreeICTCLAS的基础上完成的一个java开源分词项目，简化了原分词程序的复杂度，旨在为广大的中文分词爱好者一个更好的学习机会。关于ICTCLAS分词系统的讨论，请访问google group关于ictclas分词系统的讨论组http://groups.google.com/group/ictclas

ICTCLAS汉语分词系：http://ictclas.org/

* 庖丁解牛分词

PaodingAnalzyer，最后版本是2.0.4，更新时间是2008-06-03，不支持Lucene3.0，目前已不再更新。

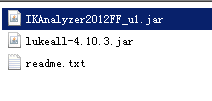


## 中文分词器测试

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





## 使用中文分词器

如果要使用中文分词器则在索引和搜索都要使用中文分词器，并且分词器一致。

//选择中文分词器

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

## 自定义词库

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和ext\_stopword.dic文件的格式为UTF-8，注意是无BOM 的UTF-8 编码。

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

