# Lucene全文检索

# 搜索技术理论基础

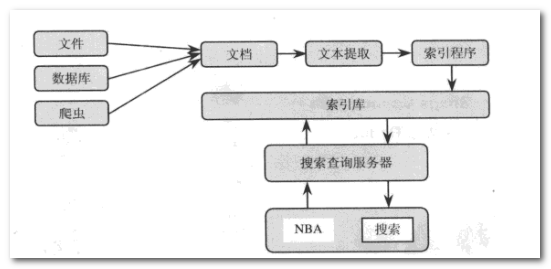
## 搜索引擎的发展历史

* 萌芽：Archie、Gopher
* 起步：Robot（网络机器人）的出现与spider（网络爬虫）
* 发展：excite、galaxy、yahoo
* 繁荣：infoseek，altaVista，Google和baidu

Robot 网络机器人： 自动在网络中运行，完成特定任务的程序 （刷票器、 抢票软件）

spider 网络爬虫 ： 是一种特殊网络机器人， 抓取（下载）并分析网络资源

## 搜索引擎的原理分析



## 搜索技术应用场景和搜索过程

应用场景 ：

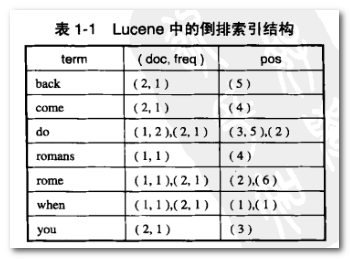
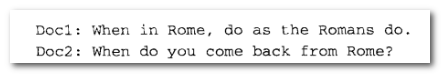
1、 单机软件的搜索（word中的搜索）

2、 站内搜索 （baidu贴吧、论坛、 京东、 taobao）

3、 垂直领域的搜索 （818工作网）

4、 专业搜索引擎公司 （google、baidu）

## 倒排索引



1. 提取资源中关键信息， 建立索引 （目录）
2. 搜索时，根据关键字（目录），找到资源的位置

# 什么是Lucene

**Lucene 是一个基于 Java 的全文信息检索工具包**，它不是一个完整的搜索应用程序，而是为你的应用程序提供索引和搜索功能。Lucene 目前是 Apache Jakarta 家族中的一个开源项目。也是目前最为流行的基于 Java 开源全文检索工具包。

目前已经有很多应用程序的搜索功能是基于 Lucene 的，比如 **Eclipse 的帮助系统的搜索功能**。Lucene 能够为**文本类型**的数据建立索引，所以你只要能把你要索引的数据格式转化的文本的，Lucene 就能对你的文档进行索引和搜索。比如你要对一些 HTML 文档，PDF 文档进行索引的话你就首先需要把 HTML 文档和 PDF 文档转化成文本格式的，然后将转化后的内容交给 Lucene 进行索引，然后把创建好的索引文件保存到磁盘或者内存中，最后根据用户输入的查询条件在索引文件上进行查询。不指定要索引的文档的格式也使 Lucene 能够几乎适用于所有的搜索应用程序。

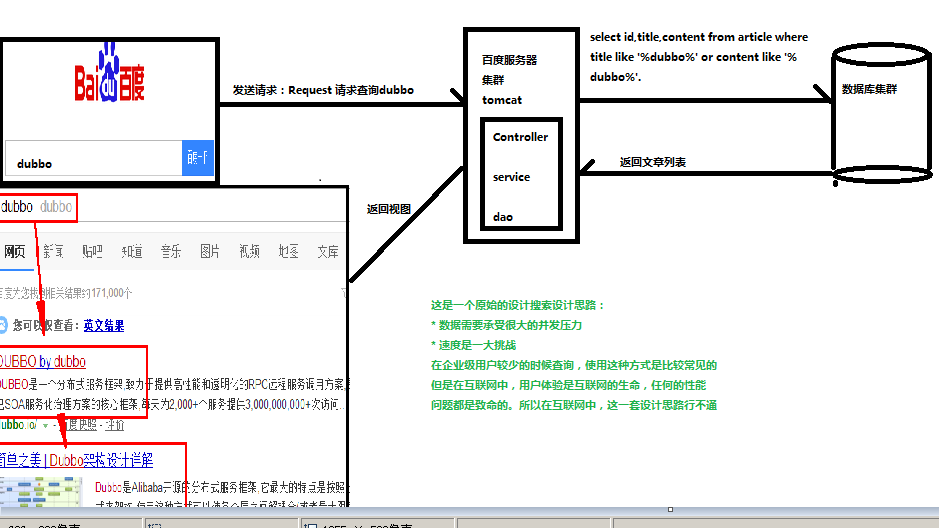
## lucene

* 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式接口，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放源代码工具
* Lucene并不是现成的搜索引擎产品，但可以用来制作搜索引擎产品
* 官网：http://lucene.apache.org/

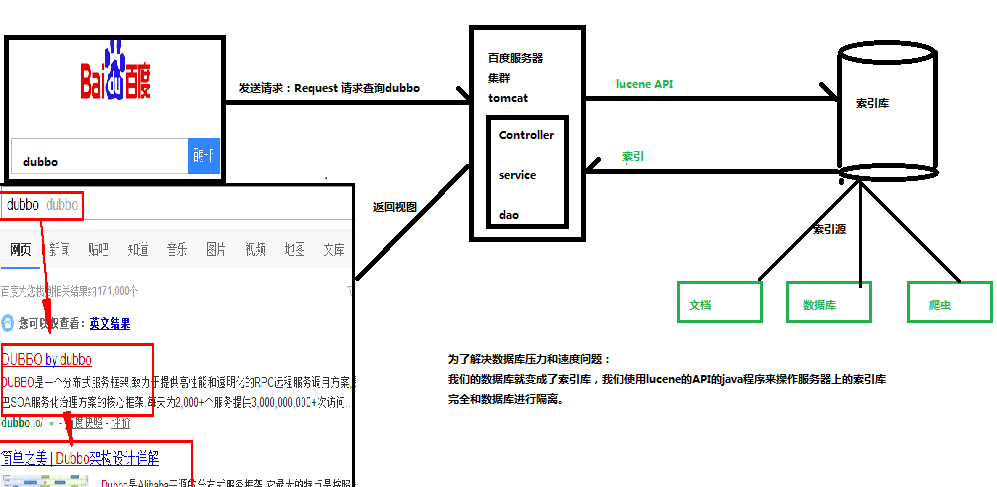
## 什么是全文索引

计算机索引程序通过扫描文章中的每一个词，对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式

## Lucene引入



那么上图就是原始搜索引擎技术，但是数据库压力太大，我们需要使用更好的解决方案来分担数据库的压力。



## 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系统

## 下载

官网： http://lucene.apache.org/



## Lucene主要包结构

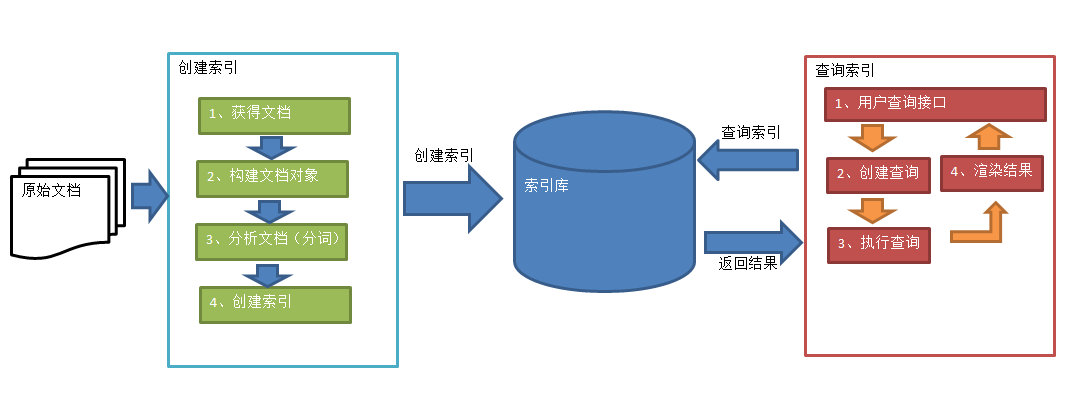
|  |  |
| --- | --- |
| 包名 | 功能 |
| org.apache.lucene.analysis | 语言分析器，主要用于的切词  Lucene提供的分析器实现类在：  lucene-analyzers-common-4.10.3.jar |
| org.apache.lucene.document | 索引存储时的文档结构管理，类似于关系型数据库的表结构 |
| org.apache.lucene.index | 索引管理，包括索引建立、删除等 |
| org.apache.lucene.queryParser | 查询分析器，实现查询关键词间的运算，如与、或、非等, 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实现全文检索的流程

案例描述

我们以一个案例来研究全文检索系统架构：实现一个文件的搜索功能，通过关键字搜索文件，凡是文件名或文件内容包括关键字的文件都需要找出来。

## 索引和搜索流程图



1、绿色表示索引过程，对要搜索的原始内容进行索引构建一个索引库，索引过程包括：

确定原始内容即要搜索的内容🡪采集文档🡪创建文档🡪分析文档🡪索引文档

2、红色表示搜索过程，从索引库中搜索内容，搜索过程包括：

用户通过搜索界面🡪创建查询🡪执行搜索，从索引库搜索🡪渲染搜索结果

## 索引流程

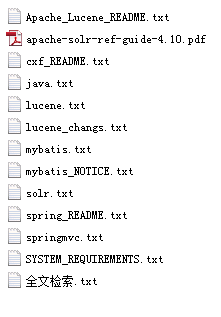
对文档索引的过程，将用户要搜索的文档内容进行索引，索引存储在索引库（index）中。

这里我们要搜索的文档是磁盘上的文本文件，根据案例描述：凡是文件名或文件内容包括关键字的文件都要找出来，这里要对文件名和文件内容创建索引。

原始内容

原始内容是指要索引和搜索的内容。原始内容包括互联网上的网页、数据库中的数据、磁盘上的文件等。

本案例中的原始内容就是磁盘上的文件，如下图：



获取原始内容（信息采集）

从互联网上、数据库、文件系统中等获取需要搜索的原始信息，这个过程就是信息采集，信息采集的目的是为了对原始内容进行索引。

在Internet上采集信息的软件通常称为爬虫或蜘蛛，也称为网络机器人，爬虫访问互联网上的每一个网页，将获取到的网页内容存储起来。

Lucene不提供信息采集的类库，需要自己编写一个爬虫程序实现信息采集，也可以通过一些开源软件实现信息采集，如下：

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。

本案例我们要获取磁盘上文件的内容，可以通过文件流来读取文本文件的内容，对于pdf、doc、xls等文件可通过第三方提供的解析工具读取文件内容，比如Apache POI读取doc和xls的文件内容。

创建文档

获取原始内容的目的是为了索引，在索引前需要将原始内容创建成文档（Document），文档中包括一个一个的域（Field），域中存储内容。

这里我们可以将磁盘上的一个文件当成一个document，Document中包括一些Field（file\_name文件名称、file\_path文件路径、file\_size文件大小、file\_content文件内容），如下图：

Document（文档）

Field（域）

Name：file\_name（文件名称）

Value：springmvc.txt

Field（域）

Name：file\_path（文件路径）

Value：e:/…/XXXXXX.txt

Field（域）

Name：file\_content（文件内容）

Value：XXXXXXXXXXXXXXXX

Field（域）

Name：file\_size（文件大小）

Value：346543

其它Field.。。。。。。。。。。。。

注意：每个Document可以有多个Field，不同的Document可以有不同的Field，同一个Document可以有相同的Field（域名和域值都相同）

分析文档

将原始内容创建为包含域（Field）的文档（document），需要再对域中的内容进行分析，分析的过程是经过对原始文档提取单词、将字母转为小写、去除标点符号、去除停用词等过程生成最终的语汇单元，可以将语汇单元理解为一个一个的单词。

比如下边的文档经过分析如下：

原文档内容：

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.

分析后得到的语汇单元：

lucene、java、full、search、engine。。。。

索引文档

对所有文档分析得出的语汇单元进行索引，索引的目的是为了搜索，最终要实现只搜索被索引的语汇单元从而找到Document（文档）。

注意：创建索引是对语汇单元索引，通过词语找文档，这种索引的结构叫**倒排索引结构**。

传统方法是根据文件找到该文件的内容，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、搜索慢。

**倒排索引结构**是根据内容（词语）找文档，如下图：



**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，它的规模较小，而文档集合较大。**

## 搜索流程

搜索就是用户输入关键字，从索引（index）中进行搜索的过程。根据关键字搜索索引，根据索引找到对应的文档，从而找到要搜索的内容（这里指磁盘上的文件）。

用户

用户可以是自然人，也可以是远程调用的程序。

用户搜索界面

全文检索系统提供用户搜索的界面供用户提交搜索的关键字，搜索完成展示搜索结果。

比如：



Lucene不提供制作用户搜索界面的功能，需要根据自己的需求开发搜索界面。

创建查询

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，比如：

语法 “fileName:spring.txt”表示要搜索Field域的内容为“spring.txt”的文档

语法 “lucene AND java” 表示要搜索即包括关键字“lucene”也包括“java”的文档。

执行搜索

搜索索引过程：

1.根据查询语法在倒排索引词典表中分别找出对应搜索词的索引，从而找到索引所链接的文档链表。

比如搜索语法为“lucene AND java”表示搜索出的文档中即要包括lucene也要包括java。

词lucene

文档lucene.txt

文档lucene\_changes.txt

.....

词java

文档solr.txt

文档lucene.txt

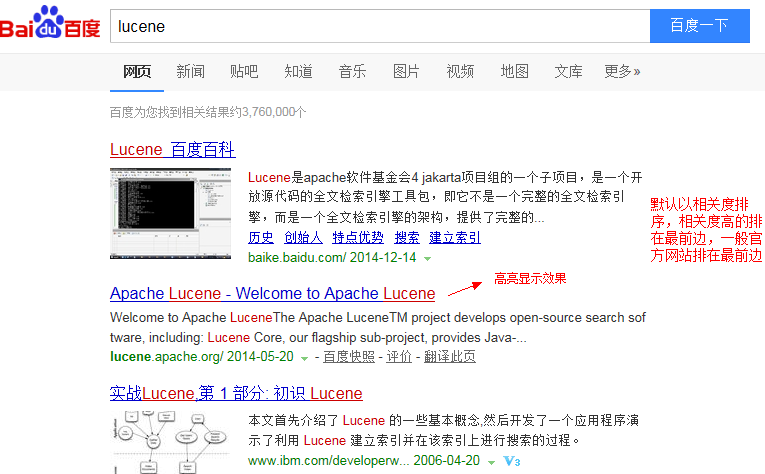
.....

2、由于是AND，所以要对包含lucene或java词语的链表进行交集，得到文档链表应该包括每一个搜索词语

3、获取文档中的Field域数据。

展示结果

以一个友好的界面将查询结果展示给用户，用户根据搜索结果找自己想要的信息，为了帮助用户很快找到自己的结果，提供了很多展示的效果，比如搜索结果中将关键字高亮显示，百度提供的快照等。



# Lucene入门

## 手动创建索引源

### 需求

使用lucene实现简单的搜索功能

开发环境

Jdk：1.7.0\_72

开发工具：eclipse4.3 kepler

Lucene包：

lucene-core-4.10.3.jar

lucene-analyzers-common-4.10.3.jar

lucene-queryparser-4.10.3.jar

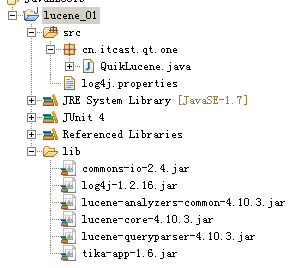
其它：

commons-io-2.4.jar

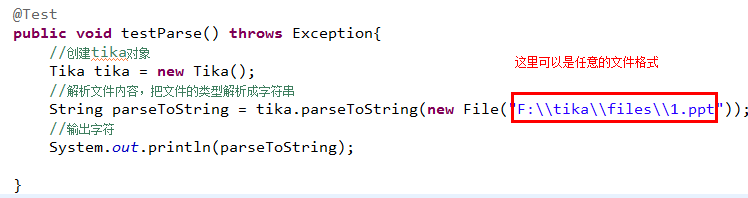
还有TiKa包文件：



### 创建Java工程



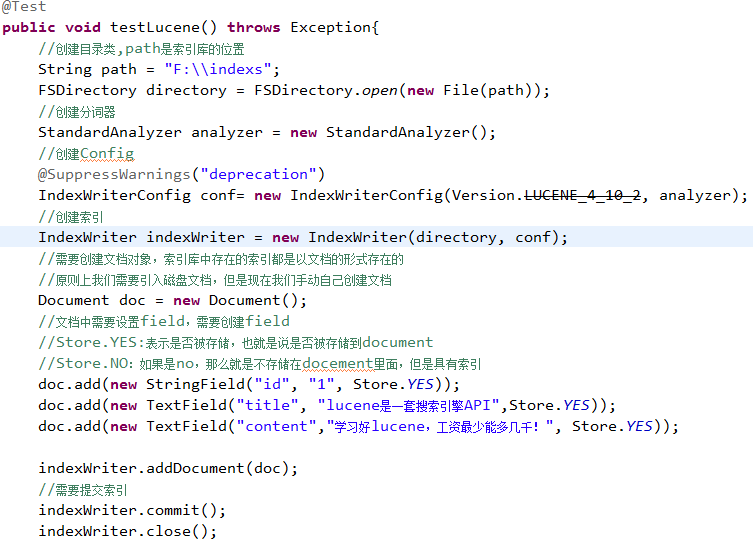
### 编写TiKa测试类



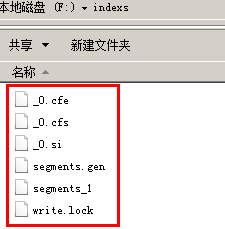
以上就是说明我们索引库中需要的任何文件的词语都可以被检索到。我们使用tika进行对任意文件的文档内容检索。然后lucene就可以利用检索的这些词语构成索引。

那么上面既然知道了获取任意文件的索引内容后，那么我就是可以创建索引了。

### 创建索引



**索引库生成了索引：**

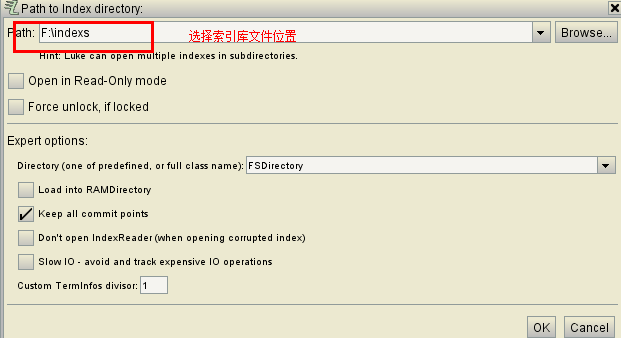


### 使用工具lukeall查看索引

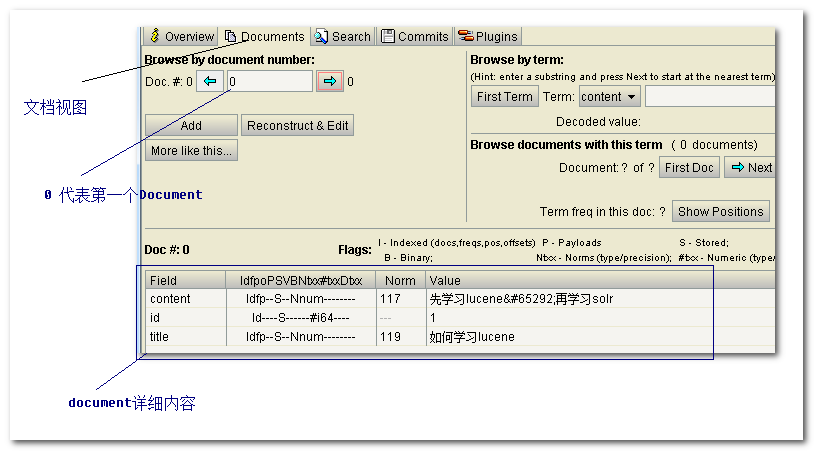
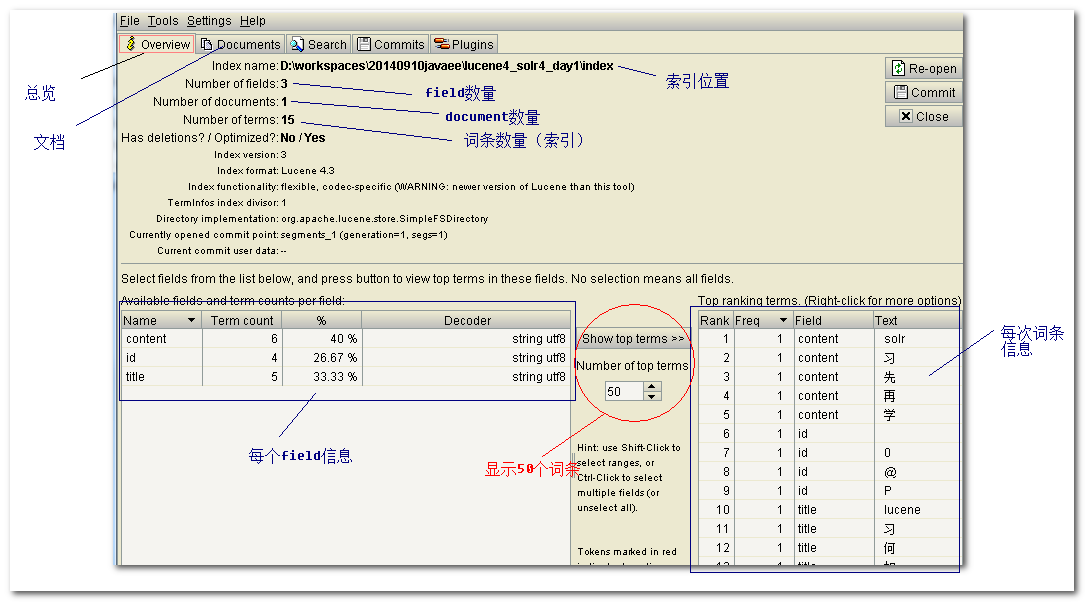


运行bat文件：也可以直接在cmd里面输入命令进行运行：Java –jar jar的路径

运行命令产生对话框，我们需要引入索引库位置：



工具使用示意图：



索引的查找过程： 根据Term词条，对Document信息进行检索

### 查询索引

@Test

**public** **void** seacherIndex() **throws** Exception{

//读取索引

String path="F:\\indexs";

DirectoryReader reader = DirectoryReader.*open*(FSDirectory.*open*(**new** File(path)));

//创建查询索引类IndexSearcher

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

//创建查询关键字

String searchName="一套";

//创建查询解析器

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

//使用查询解析器解析关键字

Query query = queryParser.parse(searchName);

//查询并且返回前10个文档的概要信息，还没有返回文档的内容

TopDocs topDocs = indexSearcher.search(query, 10);

//文档命中的个数，也就是返回文档的个数

**int** totalHits = topDocs.totalHits;

//目前返回的只有一个文档，我们只手动设置了一个文档

System.*out*.println("返回文档个数："+totalHits);

ScoreDoc[] scoreDocs = topDocs.scoreDocs;

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

//文档的ID

System.*out*.println(s.doc);

//文档的打分

System.*out*.println(s.score);

//查询文档内容,根据文档ID进行查询

Document doc = indexSearcher.doc(s.doc);

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

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

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

System.*out*.println(id+"||"+title+"||"+content);

}

}

}

分析：

查询过程：

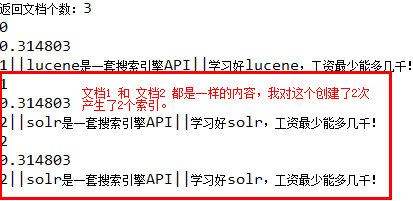
* 读取索引文件
* 准备用户查询关键字(真实项目从页面输入)
* 创建查询解析器解析查询字段QueryParser
* 创建索引解析类解析索引IndexSearcher
* 使用索引查询类查询返回文档概要信息
* 使用索引查询文档IndexSearcher.doc(docID)

那么以上就是我们查询文档的简单过程，整个查询都是查询简单的文档，这个文档是我们自己设置的自动信息。原则上我们互联网查询都是查询的网络爬虫爬下来的信息。

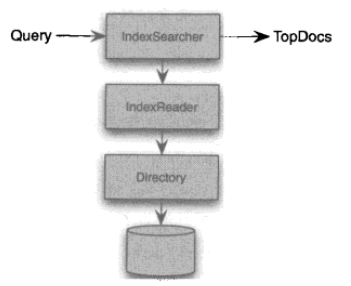
那么以上查询需要注意以下几点：

* 文档的索引ID和我们自己设置的ID没有关系，索引ID是从0开始。
* 查询是根据索引ID进行查询，不是根据我们自己设置ID查询的
* 可以对相同的内容创建多个索引，也就是创建多个文档(document)

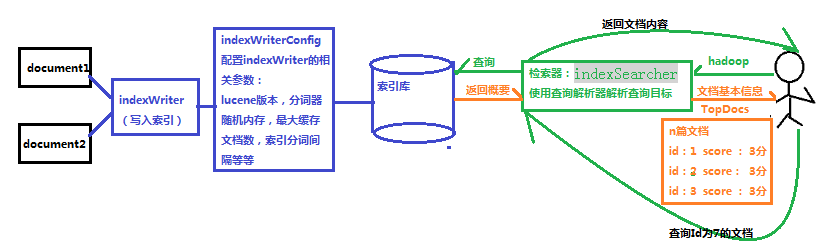
### 查询结果



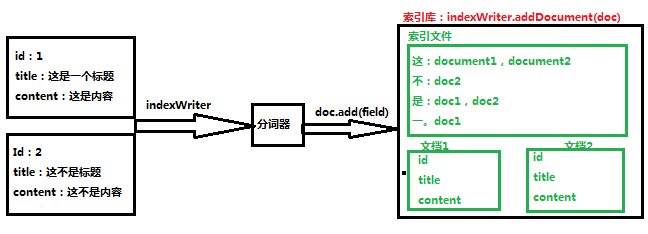
### 查询索引解析图



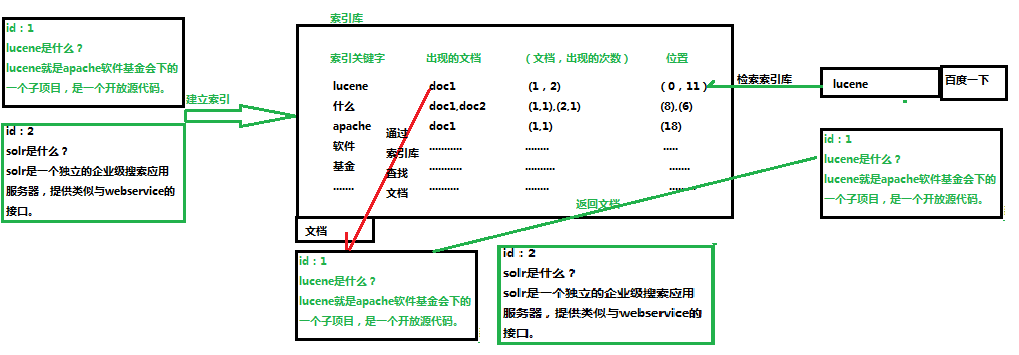
### 简单的程序执行流程



### 索引建立



### 详细索引参考



### 总结

需要注意：

Indexreader打开需要耗费很大的系统资源，建议使用一个DirectoryReader，如果索引进行添加、修改或删除需要打开新的Reader才可以搜索到。

#### Field

Document中装有很多Field。

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  是否分析 | 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. |

#### IndexSearcher

IndexSearcher搜索方法如下：

|  |  |
| --- | --- |
| 方法 | 说明 |
| indexSearcher.search(query, n) | 根据Query搜索，返回评分最高的n条记录 |
| indexSearcher.search(query, filter, n) | 根据Query搜索，添加过滤策略，返回评分最高的n条记录 |
| indexSearcher.search(query, n, sort) | 根据Query搜索，添加排序策略，返回评分最高的n条记录 |
| indexSearcher.search(booleanQuery, filter, n, sort) | 根据Query搜索，添加过滤策略，添加排序策略，返回评分最高的n条记录 |

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 |

注意：

Search方法需要指定匹配记录数量n：indexSearcher.search(query, n)

TopDocs.totalHits：是匹配索引库中所有记录的数量

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

根据以上简单的列子，相信大家会对lucene有了一个初步的认识。

经过上面我们所说的lucene的简单的引入原理：我们初步明白了几个问题？

* 必须要有索引源：磁盘文件，网页等等，最主要是网页
* 必须要对索引源建立索引
* 那么建立索引必须先经过分词器

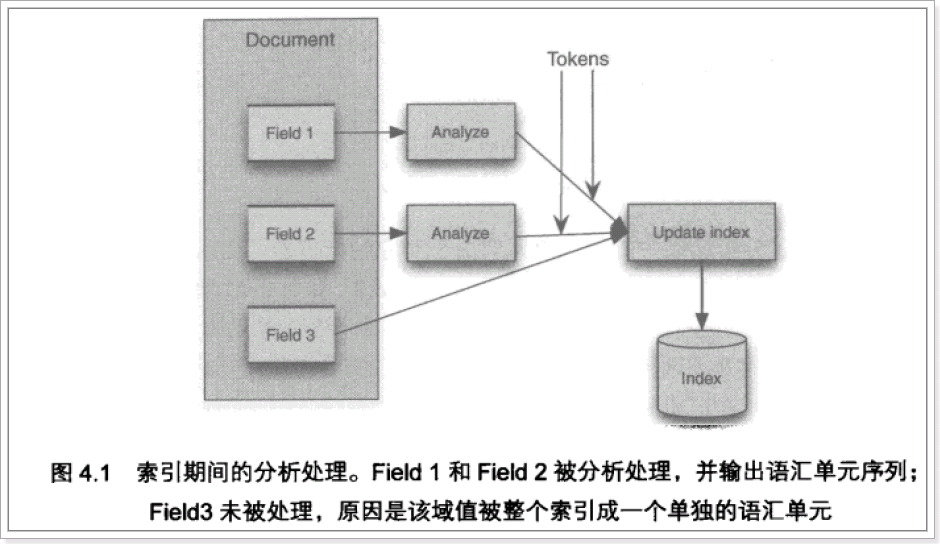
有了以上的概念后，那么下面我们来详细的研究一下lucene的索引和分词器。

# 分词器

## Analyzer使用时机

索引时使用Analyzer

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分析器处理生成语汇单元（Token）。分析器分析的对象是文档中的Field域。当Field的属性tokenized（是否分词）为true时会对Field值进行分析，如下图：



对于一些Field可以不用分析：

1、不作为查询条件的内容，比如文件路径

2、不是匹配内容中的词而匹配Field的整体内容，比如订单号、身份证号等。

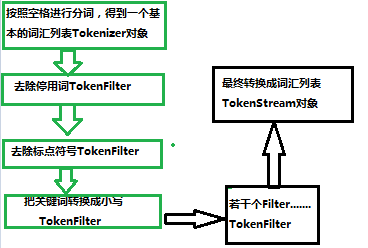
搜索时使用Analyzer

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。比如：搜索关键字：spring web ，经过分析器进行分词，得出：spring web拿词去索引词典表查找 ，找到索引链接到Document，解析Document内容。

对于匹配整体Field域的查询可以在搜索时不分析，比如根据订单号、身份证号查询等。

**注意：搜索使用的分析器要和索引使用的分析器一致。**

### 分词过程



### 分词器测试

根据以上流程图：我们发现最终我们生成了词汇单元，那么现在我们利用词汇单元来测试我们分词器

#### StandAnalyzer



#### 测试结果

start>>:0

我

end>>:1

start>>:1

是

end>>:2

start>>:2

中

end>>:3

start>>:3

国

end>>:4

start>>:4

人

end>>:5

start>>:6

我

end>>:7

start>>:7

爱

end>>:8

start>>:8

传

end>>:9

start>>:9

智

end>>:10

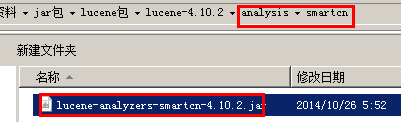
根据测试结果：我们发现标准分词器是把每一个汉字全部拆分开来，这样是不是很不优化，浪费存储空间。

#### CJKAnalyzer

这个分词器不管是不是词语，他都给拆分成2个字一组的词语。

#### SmartChineseAnalyzer

这个分词器需要导入jar文件：



中国国内提供的一款分词器

这一个对中文支持稍好，但是这样的分词器还是不能满足我们的要求，对英文处理时常丢失英文字母。

## 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。

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。

* SmartChineseAnalyzer

对中文支持较好，但扩展性差，扩展词库，禁用词库和同义词库等不好处理

第三方产品介绍

 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。

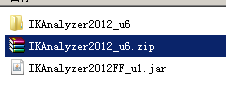
 ansj\_seg：最新版本在 <https://github.com/NLPchina/ansj_seg> tags仅有1.1版本，从2012年到2014年更新了大小6次，但是作者本人在2014年10月10日说明：“可能我以后没有精力来维护ansj\_seg了”，现在由”nlp\_china”管理。2014年11月有更新。并未说明是否支持Lucene，是一个由CRF（条件随机场）算法所做的分词算法。

 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

导入jar文件



#### 测试IK分词器

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

测试代码如下：

//创建分析器

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

//得到TokenStream

TokenStream tokenStream = analyzer.tokenStream("content", **new** StringReader("我是中国人，我爱传智播客！"));

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 -->

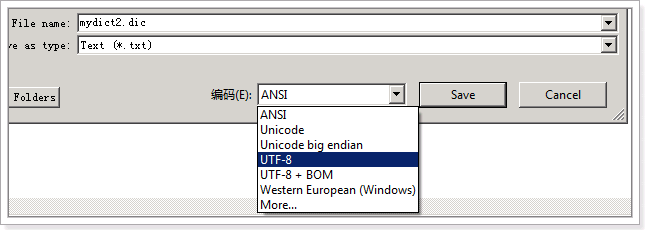
<entry key=*"ext\_stopwords"*>dicdata/ext\_stopword.dic</entry>

</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：



### 测试代码



### 查询结果

加载扩展词典：ext.dic

加载扩展停止词典：stopword.dic

start>>:0

我

end>>:1

start>>:2

中国人

end>>:5

start>>:2

中国

end>>:4

start>>:3

国人

end>>:5

start>>:6

我

end>>:7

start>>:8

爱

end>>:9

start>>:9

传

end>>:10

start>>:10

智

end>>:11

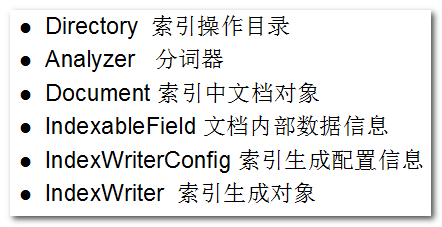
可以看见，也这个字没有出现在我们的词语分配中，我们在停词器中把也停词了，也不参与词语的分配。

注意：dic文件必须使用utf8的编码方式，并且是无bom的utf8，只能使用editPlus编辑。

不然分词器不起作用。

# LuceneAPI详解

## 创建索引API



### Directory

Directory 索引路径，常用的两种：文件和内存

FSDirectory 文件磁盘路径

FSDirectory directory = FSDirectory.open(new File("D:/indexDir"));

RAMDirectory是是基于内存的索引库：

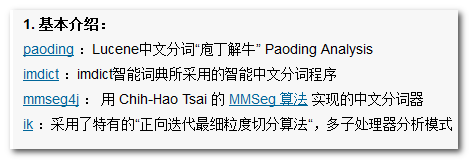
RAMDirectory 内存路径，依赖文件磁盘路径FSDirectory

RAMDirectory ramDirectory = new RAMDirectory(directory, new IOContext());

### Analyzer

分词器：最简单的分词器是StandardAnalyzer，将文字一个一个的分开，在英文的分词中比较适合，但是在中文的分词，由于都是词，这个分词器不太适合。

实际开发会使用企业级分词器 ，如下：



学习IK分词器的使用

Google上提供IK分词器，只能支持Lucene3.x ， 下载 IKAnalyzer2012FF\_u1.jar （被改动过 支持lucene4.x ）

1） 导入jar包 IKAnalyzer2012FF\_u1.jar

2） 复制 IKAnalyzer.cfg.xml、stopword.dic 到src

扩展词典： 有些词IK分词器不识别 例如“传智播客”

停用词典： 有些词不需要建立索引

3）使用：Analyzer ikAnalyzer = new IKAnalyzer();

### Document和IndexableField

Document文档 和 IndexableField 属性

IndexableField 三个实现类 ： LongField、StringField（不分词）、TextField (分词)

StringField 不分词，建立索引， 内容建立一个Term

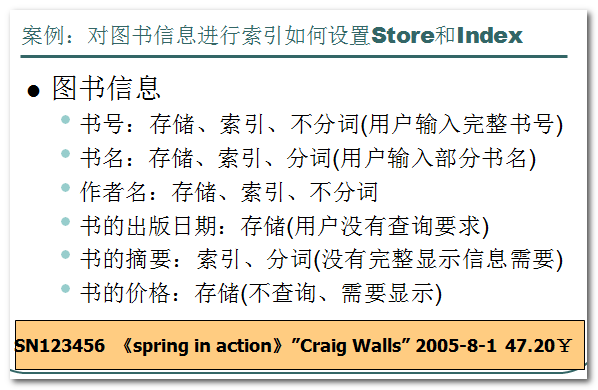
TextField 分词，建立索引， 内容分为多个Term

Store.YES 会在document存储信息，搜索结果会有

Store.NO 不会保存document，搜索结果没有

如何决定字段是否该被索引，是否该被存储呢？

把握一个原则：如果想被查，就索引。想显示，就存储。



如果某个字段需要存储，不需要索引

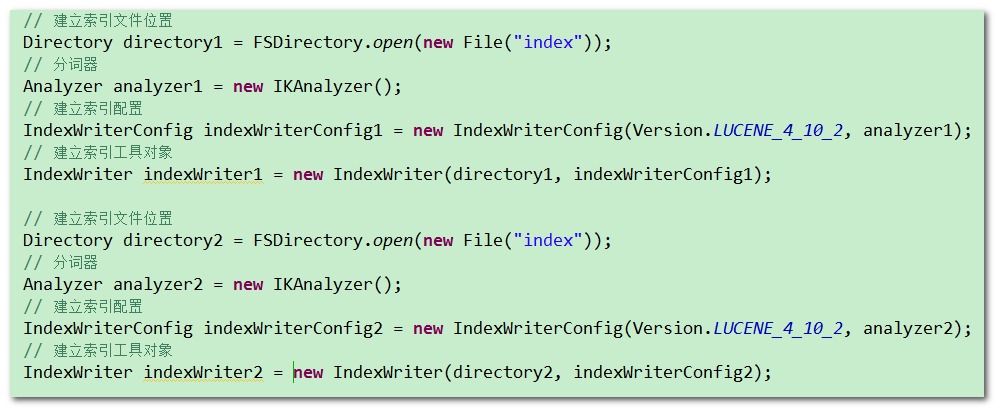
|  |
| --- |
| FieldType testType = new FieldType();  testType.setTokenized(false);//不分词  testType.setIndexed(true);//索引  testType.setStored(true);//存储  Field test = new Field("title", "如何学习solr", testType);  document.add(test); |

### IndexWriter

IndexWriter 创建索引

1、IndexWriter操作索引目录时，对目录进行加锁 write.lock

2、当两个writer同时操作索引目录的时候，会报错

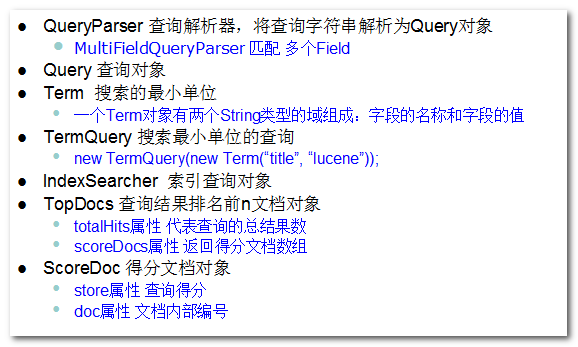


锁获取失败异常：org.apache.lucene.store.LockObtainFailedException: Lock obtain timed out: NativeFSLock@D:\workspaces\20140910javaee\lucene4\_solr4\_day1\index\write.lock

3、解决方案： 提供工具类，static提供唯一IndexWriter

|  |
| --- |
| public class LuceneUtils {  private static Directory directory;  private static Analyzer analyzer;  private static IndexWriter indexWriter;  static {  try {  directory = FSDirectory.open(new File("index"));  analyzer = new IKAnalyzer();  IndexWriterConfig indexWriterConfig = new IndexWriterConfig(Version.LUCENE\_4\_10\_2, analyzer);  indexWriter = new IndexWriter(directory, indexWriterConfig);  // 虚拟机退出时关闭  Runtime.getRuntime().addShutdownHook(new Thread() {  @Override  public void run() {  System.out.println("关闭IndexWriter...");  try {  indexWriter.close();  } catch (IOException e) {  e.printStackTrace();  }  }  });  } catch (Exception e) {  e.printStackTrace();  }  }  // 获取IndexWriter  public static IndexWriter getIndexWriter() {  return indexWriter;  }  } |

## 查询索引API



搜索编程步骤；

1) 查询内容转换Query对象 ---- 通过QueryParser

2）通过IndexSearcher 进行搜索 search(query, n)

n代表返回排名靠前多少条数据

3）搜索索引库后，获取TopDocs

提供 totalHits 查询结果总数， 提供scoreDocs 返回得分文档对象集合

4）遍历每个ScoreDoc

提供score属性，代表当前搜索文档得分

indexSearcher.doc(scoreDoc.doc) 获取Document对象

### 两种QueryParser

#### QueryParser

针对单一字段，解析分词搜索 QueryParser

注意：这种解析方式是需要Analyzer解析器的，是针对于在索引算法中被分词的项

**查询索引解析过程分词算法必须和添加索引分词算法一致**。不然可能就会查询不到。

查询解析使用分词算法会对查询字段进行分词，这个分词算法分到的词我我们添加索引时分到的词是一样的，所以他就能匹配到。

|  |
| --- |
| //你想找什么？  String searchContent = "学校";  //创建查询转换器，设置搜索的filed  QueryParser parser = new QueryParser("title", new IKAnalyzer());  //转换  Query parse = parser.parse(searchContent);    //创建查询器  IndexSearcher indexSearcher = new IndexSearcher(DirectoryReader.open(FSDirectory.open(new File("D:\\indexDir"))));  //搜索前n名结果：按照得分排序的结果  TopDocs topDocs = indexSearcher.search(parse, Integer.MAX\_VALUE);  //打印结果集  System.out.println("搜索结果"+topDocs.totalHits);  for (ScoreDoc scoreDoc : topDocs.scoreDocs) {  System.out.println("得分："+scoreDoc.score);  Document doc = indexSearcher.doc(scoreDoc.doc);  System.out.println(doc.get("id"));  System.out.println(doc.get("title"));  System.out.println(doc.get("content"));  System.out.println();  } |

#### MultiFieldQueryParser

针对多字段，进行分词搜索 MultiFieldQueryParser

|  |
| --- |
| //你想找什么？  String searchContent = "学校";  //创建查询转换器，设置搜索的filed  MultiFieldQueryParser parser = new MultiFieldQueryParser(new String[]{"title", "content"},new IKAnalyzer());  Query parse = parser.parse(searchContent); |

### 6种query

#### 词条搜索， TermQuery使用

Term代表词条，被分词后词语（使用TermQuery 不会分词）

|  |
| --- |
| String searchContent = "分";  Query parse = new TermQuery(new Term("content", searchContent)); |

词条就是索引库最小搜索单位，不可再分，如果词条 没有“分”，搜索无结果

#### 模糊搜索， WildcardQuery使用

? 代表一个字符

\* 代表任意多个字符

|  |
| --- |
| String searchContent = "基";  Query parse = new WildcardQuery(new Term("content", "\*"+searchContent+"\*")); |

#### 相似度搜索 FuzzyQuery 使用

允许用户输入有错误，显示相似单词搜索结果

例如：你想搜索lucene，但是写成了lvcnee，那么仍能搜出结果

|  |
| --- |
| String searchContent = "lvsene";  Query parse = new FuzzyQuery(new Term("content", searchContent)); |

为什么 lvcnee 可以匹配lucene词条？？----levenshtein（[编辑距离](http://baike.baidu.com/view/2020247.htm" \t "_blank)）算法

第一次编辑 v --- u

第二次编辑 n --- e 交换位置



maxEdis 取值 0 、1、2 三个整数 ，最多可以编辑两次

#### 数字范围搜索 NumericRangeQuery 使用

问题：查询 5-15 id的Document对象

批量创建30个Document的索引

|  |
| --- |
| for (int i = 1; i < 30; i++) {  Document document = new Document();  // 为document添加字段  document.add(new LongField("id", (long)i, Store.YES));  FieldType testType = new FieldType();  testType.setIndexed(true);  testType.setStored(false);  Field test = new Field("title", "如何学习solr", testType);  document.add(test);  document.add(new TextField("content", "solr是基于lucene的技术", Store.YES));  writer.addDocument(document);  } |

使用 NumericRangeQuery 进行范围搜索

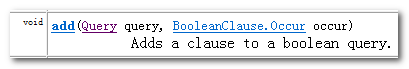
|  |
| --- |
| //查询id在1-10之间的，包含1和10  Query parse = NumericRangeQuery.newLongRange("id", 1L, 10L, true, true); |

#### 查询索引库所有内容 MatchAllDocsQuery

|  |
| --- |
| Query parse = new MatchAllDocsQuery(); |

#### 组合查询 BooleanQuery的使用

提供 add，用于添加其它Query



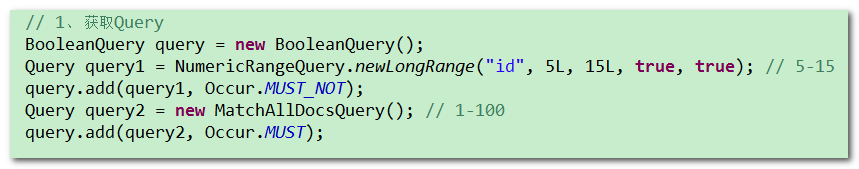
occur可以取值 MUST 必须满足 、 MUST\_NOT 必须不满足， SHOULD 可以满足

常见组合：

1） MUST + MUST 交集

2） SHOULD + SHOULD 并集

3） MUST + MUST\_NOT 补集



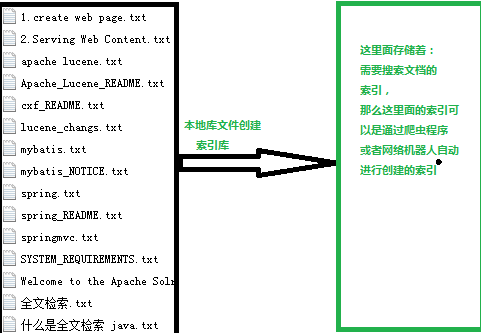
# 读取磁盘文档创建索引

## 需求

读取磁盘txt文件，对磁盘文件创建索引。

实现一个资源管理器的搜索功能，通过关键字搜索，凡是文件名或文件内容包括关键字的文件都要找出来。

注意：该入门程序只对文本文件(.txt)搜索



注意：这个实例我们需要对本地磁盘文件创建索引，然后对本地文件进行搜索。

思考：我们这个文档里面有很多内容，这些内容都必须被索引库扫描到，装载到索引库中

那么我们想想，也就是我们需要读取文档中的内容，把文档中的内容扫描到索引库中以便于我们进行搜索。

那么我们现在需要做的事就是把各种格式文件内容进行读取

现在思考：我们怎么才能读取各种格式的文件内容呢？(ppt,pdf,txt……..)

我们唯一能想到的是流，以流的方式进行读取文件，那么这种方式行不行呢？当然可以，但是比较麻烦，那么现在我们使用lucene里面提供的一种方式：使用TIka进行读取。非常简单。

## 创建索引源文件

创建数据源目录F:\develop\lucene\searchsource ，该目录存放要搜索的原始文件

## 创建索引目录

创建索引目录：F:\indexs

## 创建文档集合工具类

思考：由于我们现在文件比较多，那么我们能不能使用一种方法把文件封装成document集合呢？

当然可以，我们使用如下方式进行封装：

**public** **class** LuceneUtils {

**public** **static** List<Document> file2Document(String searchPath) **throws** Exception{

//需求，我们需要读取磁盘文件，对磁盘文件进行搜索

//创建document的集合

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

//获取文件

File folder = **new** File(searchPath);

**if** (!folder.isDirectory()) {

**return** **null**;

}

// 获取目录 中的所有文件

File[] files = folder.listFiles();

**for** (File file : files) {

//文件名称

String fileName = file.getName();

System.*out*.println(fileName);

//使用Tika来读取任意文件

Tika tika = **new** Tika();

String fileContent = tika.parseToString(file);

//创建文档

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

//创建各各Field域

//文件名

Field field\_fileName = **new** StringField("fileName", fileName, Store.*YES*);

//文件内容，由于读取文件内容很大，所以我们不保存

Field field\_fileContent = **new** TextField("fileContent", fileContent, Store.*NO*);

//将解析文件放入document

doc.add(field\_fileName);

doc.add(field\_fileContent);

list.add(doc);

}

**return** list;

}

}

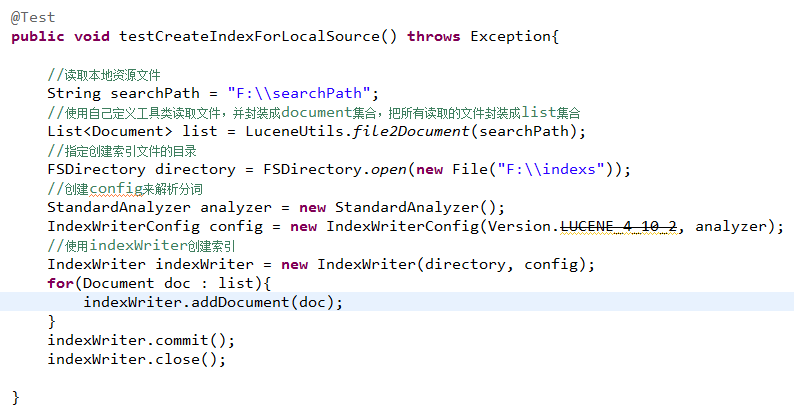
分析：以上代码就是读取目录下所有文件，并且把每一个文件封装成file对象以便于我们操作。针对与file对象，我们使用tika类读取文件内容，我们需要对文件内容建立索引。

然后在把内容放入我们lucene能解析的Field域中，添加到我们的document中。这样我们的document集合就搞定了。

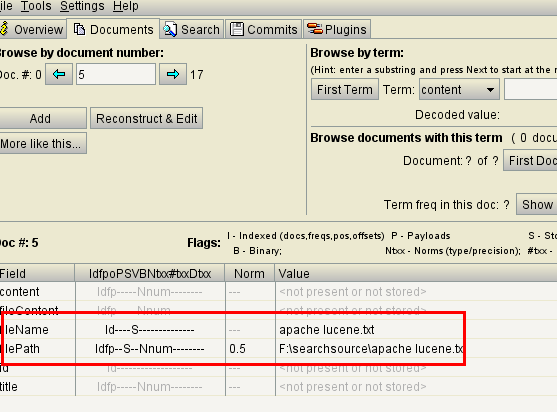
## 创建索引

上面已经读取源文件，并且把源文件放入了document，然而document是我们解析文件的基本单位，所以现在我们就方便多了。

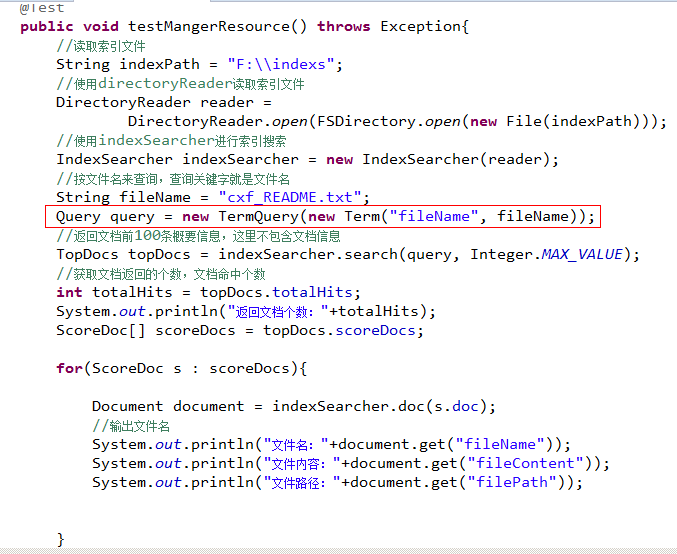
创建索引代码如下：



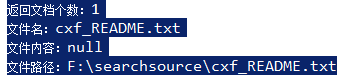
## luke查看索引情况



## 查询索引



## 查询结果



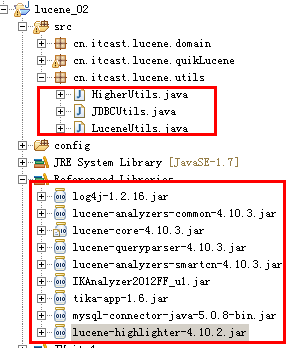
# 索引库CRUD结合jdbc练习

## 为什么结合jdbc

1. 我们在插入索引数据的同时，将数据备份在jdbc中一份，可以用于恢复索引库
2. 有些数据需要索引，放到lucene中，有些数据不需要索引，放到jdbc中

## 准备工作

### 创建Java工程



1. 将IndexSearcher对象获取，加入LuceneUtils 工具类



根据以上工具类，直接提供一下方法：

|  |
| --- |
| // 获取IndexWriter  **public** **static** IndexWriter getIndexWriter() {  **return** *indexWriter*;  }  //获取indexsearcher  **public** **static** IndexSearcher getIndexSearcher() **throws** Exception {  **return** **new** IndexSearcher(DirectoryReader.*open*(*directory*));  } |

2、创建数据库和表

|  |
| --- |
| create database lucene;  CREATE TABLE article(  id VARCHAR(10),  title VARCHAR(200),  content TEXT  ); |

1. 创建实体article

|  |
| --- |
| **package** cn.itcast.lucene.domain;  **public** **class** Article {  **private** String id;  **private** String title;  **private** String content;  **public** String getId() {  **return** id;  }  **public** **void** setId(String id) {  **this**.id = id;  }  **public** String getTitle() {  **return** title;  }  **public** **void** setTitle(String title) {  **this**.title = title;  }  **public** String getContent() {  **return** content;  }  **public** **void** setContent(String content) {  **this**.content = content;  }  @Override  **public** String toString() {  **return** "Article [id=" + id + ", title=" + title + ", content="  + content + "]";  }      } |

1. 创建jdbc资源文件jdbc.properties

|  |
| --- |
| driverClass= com.mysql.jdbc.Driver  url= jdbc:mysql://localhost:3306/lucene  user= root  password= admin |

1. 创建jdbc工具类

|  |
| --- |
| package com.itcast.utils;  import java.io.IOException;  import java.io.InputStream;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;  import java.util.Properties;  import com.itcast.bean.Article;  /\*\*  \* jdbc工具类 1、获取链接 2、释放资源  \*  \* @author wilson  \*  \*/  public class JDBCUtils {  private static String driverClass;  private static String url;  private static String user;  private static String password;  /\*\*  \* 加载配置文件  \*/  static {  // 注意，jdbc.properties必须在src下  InputStream inputStream = JDBCUtils.class.getClassLoader()  .getResourceAsStream("jdbc.properties");  Properties p = new Properties();  try {  p.load(inputStream);  driverClass = p.getProperty("driverClass");  url = p.getProperty("url");  user = p.getProperty("user");  password = p.getProperty("password");  } catch (IOException e) {  e.printStackTrace();  }  }  /\*\*  \* 获取链接  \*  \* @return  \* @throws ClassNotFoundException  \* @throws SQLException  \*/  public static Connection getSource() throws ClassNotFoundException,  SQLException {  // 打招呼  Class.forName(driverClass);  // 建桥  return DriverManager.getConnection(url, user, password);  }  /\*\*  \* 释放资源  \*/  public static void release(Connection conn, Statement stmt, ResultSet rs) {  if (rs != null) {  try {  rs.close();  } catch (SQLException e) {  e.printStackTrace();  }  rs = null;  }  if (stmt != null) {  try {  stmt.close();  } catch (SQLException e) {  e.printStackTrace();  }  stmt = null;  }  if (conn != null) {  try {  conn.close();  } catch (SQLException e) {  e.printStackTrace();  }  conn = null;  }  }  public static void insertIndex(Article article) throws SQLException {  Connection conn = null;  PreparedStatement stmt = null;  try {  conn = JDBCUtils.getSource();  String sql = "insert into article values(?,?,?)";  stmt = conn.prepareStatement(sql);  stmt.setString(1, article.getId());  stmt.setString(2, article.getTitle());  stmt.setString(3, article.getContent());  stmt.executeUpdate();  } catch (Exception e) {  conn.rollback();  e.printStackTrace();  } finally {  JDBCUtils.release(conn, stmt, null);  }  }  public static void deleteIndex(String id) throws SQLException {  Connection conn = null;  PreparedStatement stmt = null;  try {  conn = JDBCUtils.getSource();  String sql = "delete from article where id = ?";  stmt = conn.prepareStatement(sql);  stmt.setString(1, id);  stmt.executeUpdate();  } catch (Exception e) {  conn.rollback();  e.printStackTrace();  } finally {  JDBCUtils.release(conn, stmt, null);  }  }  public static void updateIndex(Article article,String id) throws SQLException {  Connection conn = null;  PreparedStatement stmt = null;  try {  conn = JDBCUtils.getSource();  String sql = "update article set id =? ,title=?,content=? where id = ?";  stmt = conn.prepareStatement(sql);  stmt.setString(1, article.getId());  stmt.setString(2, article.getTitle());  stmt.setString(3, article.getContent());  stmt.setString(4, id);  stmt.executeUpdate();  } catch (Exception e) {  conn.rollback();  e.printStackTrace();  } finally {  JDBCUtils.release(conn, stmt, null);  }  }  public static Article selectIndex(String id) throws SQLException {  Connection conn = null;  PreparedStatement stmt = null;  ResultSet rs = null;  Article article = null;  try {  conn = JDBCUtils.getSource();  String sql = "select \* from article where id = ?";  stmt = conn.prepareStatement(sql);  stmt.setString(1, id);  rs = stmt.executeQuery();  article = new Article();  while(rs.next()){  article.setId(rs.getString("id"));  article.setTitle(rs.getString("title"));  article.setContent(rs.getString("content"));  }  } catch (Exception e) {  conn.rollback();  e.printStackTrace();  } finally {  JDBCUtils.release(conn, stmt, null);  }  return article;  }    } |

### crud索引库和数据库

|  |
| --- |
| package cn.itcast.lucene.quikLucene;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.List;  import org.apache.lucene.analysis.Analyzer;  import org.apache.lucene.document.Document;  import org.apache.lucene.index.IndexWriter;  import org.apache.lucene.index.Term;  import org.apache.lucene.queryparser.classic.MultiFieldQueryParser;  import org.apache.lucene.search.IndexSearcher;  import org.apache.lucene.search.Query;  import org.apache.lucene.search.ScoreDoc;  import org.apache.lucene.search.TopDocs;  import org.wltea.analyzer.lucene.IKAnalyzer;  import cn.itcast.lucene.domain.Article;  import cn.itcast.lucene.utils.JDBCUtils;  import cn.itcast.lucene.utils.LuceneUtils;  public class LuceneCRUD {    //添加索引的同时添加到数据库  public void insertIndexAndDatabase(Article article) throws Exception{  //添加到数据库当中  JDBCUtils.insertIndex(article);  //添加索引库  Document doc = LuceneUtils.articleToDoc(article);  //获取indexWriter  IndexWriter indexWriter = LuceneUtils.getIndexWriter();  //添加索引  indexWriter.addDocument(doc);  indexWriter.commit();  indexWriter.close();  }  //删除索引  public void deleteIndex(String id) throws Exception{    //先删除数据库索引，根据实际情况决定是否删除  JDBCUtils.deleteIndex(id);  //删除索引库中索引  IndexWriter indexWriter = LuceneUtils.getIndexWriter();  indexWriter.deleteDocuments(new Term("id", id));  indexWriter.commit();  }    //修改  public void updateIndex(Article article,String id) throws Exception{  //修改数据库索引  JDBCUtils.updateIndex(article, id);  //修改索引库  //获取document  Document doc = LuceneUtils.articleToDoc(article);  IndexWriter indexWriter = LuceneUtils.getIndexWriter();  indexWriter.updateDocument(new Term("id", id), doc);    indexWriter.commit();    }    public List<Article> selectIndex(String searchName,int pageNo,int count,String...fields) throws Exception{    List<Article> list = new ArrayList<Article>();  //使用什么查询方法：使用QueryParser查询方法  Analyzer analyzer = new IKAnalyzer();  //由于使用的分词Field，这里必须使用查询解析器进行查询  MultiFieldQueryParser mQueryParser = new MultiFieldQueryParser(fields, analyzer);  Query query = mQueryParser.parse(searchName);  //获取查询索引核心对象  IndexSearcher indexSearcher = LuceneUtils.getIndexSearcher();  TopDocs topDocs = indexSearcher.search(query, pageNo\*count);  //分页查询  for(int i=(pageNo-1)\*count;i<topDocs.totalHits;i++){  //获取打分最高的记录  ScoreDoc scoreDoc = topDocs.scoreDocs[i];  //文档  Document doc = indexSearcher.doc(scoreDoc.doc);  //把文档转换成对象  Article article = LuceneUtils.docToArticle(doc);  list.add(article);  }  return list;  }  } |

## junit测试

|  |
| --- |
| @Test  public void testInsert() throws IOException, SQLException {  LuceneCRUD luceneCRUD = new LuceneCRUD();  Article article = new Article();  article.setId("1");  article.setTitle("这是测试lucene的创建索引");  article.setContent("这是测试的内容");  luceneCRUD.insertIndex(article);  }  @Test  public void testupdate() throws IOException, SQLException {  LuceneCRUD luceneCRUD = new LuceneCRUD();  Article article = new Article();  article.setId("2");  article.setTitle("这是测试lucene的创建索引2");  article.setContent("这是测试的内容2");  luceneCRUD.updateIndex(article, "1");  }  @Test  public void testdelete() throws IOException, SQLException {  LuceneCRUD luceneCRUD = null;  try {  luceneCRUD = new LuceneCRUD();  } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  luceneCRUD.deleteIndex("1");  }  @Test  public void testselect() throws Exception {  LuceneCRUD luceneCRUD = new LuceneCRUD();  List<Article> selectIndex = luceneCRUD.selectIndex("lucene", 1, 10, "title","content");  for (Article article2 : selectIndex) {  System.out.println(article2.getId());  System.out.println(article2.getTitle());  System.out.println(article2.getContent());  System.out.println();  }  } |

## 索引的调优(自学了解)

内存路径 RAMDirectory 与 硬盘路径 FSDirectory 结合完成索引

1、 程序启动时，从FSDirectory 加载所有索引内容到内存RAMDirectory

2、 建立索引，对RAMDirectory

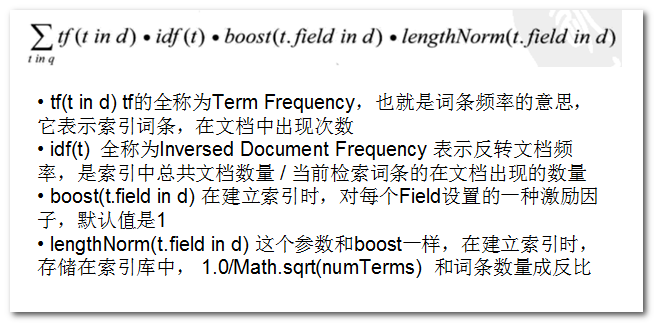
3、 建立索引，对FSDirectory

|  |
| --- |
| Directory directory = FSDirectory.open(new File("D:\\indexDir"));  RAMDirectory ramDirectory = new RAMDirectory(directory, new IOContext());  Document document = new Document();  document.add(new StringField("id", "2", Store.YES));  document.add(new TextField("title", "aaa", Store.YES));  document.add(new TextField("content", "aaa", Store.YES));  IndexWriter writer = new IndexWriter(ramDirectory, new IndexWriterConfig(Version.LUCENE\_4\_10\_2, new IKAnalyzer()));  writer.addDocument(document);  IndexWriter writer2 = new IndexWriter(directory, new IndexWriterConfig(Version.LUCENE\_4\_10\_2, new IKAnalyzer()));  writer2.addDocument(document);  writer.close();  writer2.close(); |

# lucene4.x得分和分页

## 索引库搜索得分排序规则

### 了解公式



* tf词频： 查找的词在某个文档中查询词语出现次数
* idf 反转文档频率：文档总数/词在多少文档出现

例子：

例1.有很多不同的数学公式可以用来计算TF-IDF。这边的例子以上述的数学公式来计算。词频 (TF) 是一词语出现的次数除以该文件的总词语数。假如一篇文件的总词语数是100个，而词语“母牛”出现了3次，那么“母牛”一词在该文件中的词频就是3/100=0.03。一个计算文件频率 (DF) 的方法是测定有多少份文件出现过“母牛”一词，然后除以文件集里包含的文件总数。所以，如果“母牛”一词在1,000份文件出现过，而文件总数是10,000,000份的话，其逆向文件频率就是 lg(10,000,000 / 1,000)=4。最后的TF-IDF的分数为0.03 \* 4=0.12。

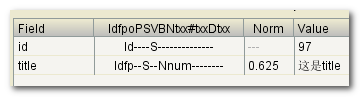
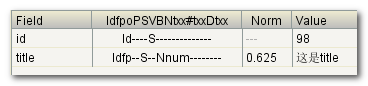
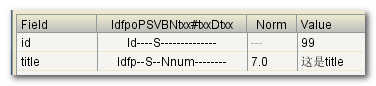
* boost 激励因子 ： 默认是1，可以手动调整
* lengthNorm ： 和Field分词数量（Terms） 成反比 ， 默认没有分词，值是 1

### 验证得分

通过设置Boost参数，改变搜索得分

|  |
| --- |
| @Test  public void createIndex() throws IOException {  for (int i = 1; i <= 100; i++) {  Document document = new Document();  document.add(new StringField("id", i+"", Store.YES));  TextField textField = new TextField("title", "这是title"+"", Store.YES);  if (i==99) {  textField.setBoost(10);  }  document.add(textField);  IndexWriter indexWriter = LuceneUtils.getIndexWriter();  indexWriter.addDocument(document);  indexWriter.commit();  }  } |

当修改Field Boost信息，保存索引库 Norm信息中



测试查询得分

|  |
| --- |
| @Test  public void searchIndex() throws ParseException, IOException {  // 你想找什么？  String searchContent = "这是";  // 创建查询转换器，设置搜索的filed  QueryParser parser = new QueryParser("title", new IKAnalyzer());  // 转换  Query parse = parser.parse(searchContent);  LuceneUtils.writeDocument(parse);  } |

## 索引分页操作

通过IndexSearcher提供 search(query, n) 这里n 代表查询前多少条信息



分页思路： 先查询结果记录位置前多少条，在遍历ScoreDoc时，取出需要的数据

# lucene4.x 查询结果的高亮显示

## 高亮结果显示的原理

1、 对搜索结果中 关键字（搜索词）， 两端添加HTML标签

2、 通过CSS样式，对特有标签进行修饰，显示为高亮

lucene在对目标内容进行高亮显示时，会截取目标内容中一段文本 （摘要），再对摘要中关键字进行高亮（两端添加HTML）

## 通过lucene完成高亮结果处理

第一步： 完成普通搜索代码

|  |
| --- |
| public void searchIndex() throws ParseException, IOException {  // 你想找什么？  String searchContent = "这是";  // 创建查询转换器，设置搜索的filed  QueryParser parser = new QueryParser("title", new IKAnalyzer());  // 转换  Query parse = parser.parse(searchContent);  IndexSearcher indexSearcher = new IndexSearcher(  DirectoryReader.open(FSDirectory.open(new File("D:/indexDir/"))));  // 搜索前n名结果：按照得分排序的结果  TopDocs topDocs = indexSearcher.search(parse, Integer.MAX\_VALUE);  // 打印结果集  System.out.println("搜索结果" + topDocs.totalHits);  for (ScoreDoc scoreDoc : topDocs.scoreDocs) {  System.out.println("得分：" + scoreDoc.score);  Document doc = indexSearcher.doc(scoreDoc.doc);  System.out.println("id：" + doc.get("id"));  System.out.println("标题：" + doc.get("title"));  System.out.println("内容：" + doc.get("content"));  System.out.println();  }  } |

第二步： 引入高亮器的jar包

导入 lucene-highlighter-4.10.2.jar

第三步： 定义和配置高亮器对象

|  |
| --- |
| // 1、 定义高亮器  Formatter formatter = new SimpleHTMLFormatter("<font color='red'>", "</font>");  Scorer fragmentScorer = new QueryScorer(parse);  Highlighter highlighter = new Highlighter(formatter, fragmentScorer);  // 设置摘要长度  highlighter.setTextFragmenter(new SimpleFragmenter(20)); |

第四步： 对查询结果 高亮标记

|  |
| --- |
| //设置高亮  String title = doc.get("title");  String titleBestFragment = highlighter.getBestFragment(new IKAnalyzer(), "title", title);  System.out.println("标题：" + titleBestFragment); |

### 高亮工具类

HigherUtils

|  |
| --- |
| package com.itcast.utils;  import java.io.IOException;  import org.apache.lucene.search.Query;  import org.apache.lucene.search.highlight.Formatter;  import org.apache.lucene.search.highlight.Highlighter;  import org.apache.lucene.search.highlight.InvalidTokenOffsetsException;  import org.apache.lucene.search.highlight.QueryScorer;  import org.apache.lucene.search.highlight.Scorer;  import org.apache.lucene.search.highlight.SimpleFragmenter;  import org.apache.lucene.search.highlight.SimpleHTMLFormatter;  import org.wltea.analyzer.lucene.IKAnalyzer;  public class HigherUtils {  private Highlighter highlighter;  /\*\*  \* 在构造函数中初始化高亮器  \* @param query  \*/  public HigherUtils(Query query) {  Formatter formatter = new SimpleHTMLFormatter("<font color='red'>",  "</font>");  Scorer fragmentScorer = new QueryScorer(query);  highlighter = new Highlighter(formatter, fragmentScorer);  // 设置摘要长度  highlighter.setTextFragmenter(new SimpleFragmenter(20));  }  /\*\*  \* 将对应的field转换为高亮显示  \* @param field  \* @param value  \* @return  \* @throws IOException  \* @throws InvalidTokenOffsetsException  \*/  public String changeHigner(String field, String value)  throws IOException, InvalidTokenOffsetsException {  String titleBestFragment = highlighter.getBestFragment(  new IKAnalyzer(), field, value);  return titleBestFragment;  }  } |