**Lucene**

**全文检索技术**

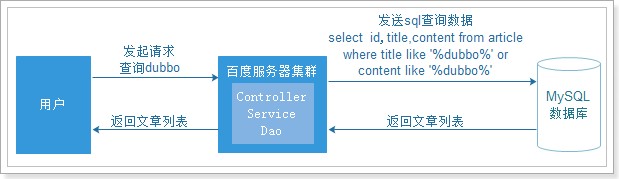
# 课程计划

1. Lucene介绍
2. 全文检索流程介绍
   1. 索引流程
   2. 搜索流程
3. Lucene入门程序
   1. 索引实现
   2. 搜索实现
4. Field域类型
5. 索引库维护
6. 分词器
   1. 分词介绍
   2. IK分词器

# 搜索技术理论基础

## 为什么要学习Lucene

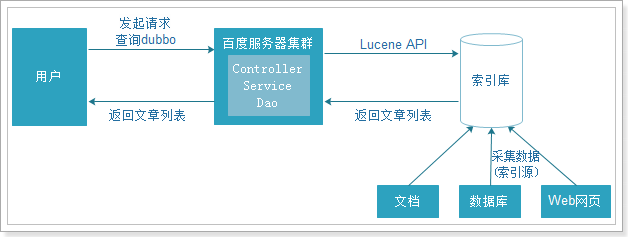
原来的方式实现搜索功能，我们的搜索流程如下图：



上图就是原始搜索引擎技术，如果用户比较少而且数据库的数据量比较小，那么这种方式实现搜索功能在企业中是比较常见的。

但是数据量过多时，数据库的压力就会变得很大，查询速度会变得非常慢。我们需要使用更好的解决方案来分担数据库的压力。

现在的方案（使用Lucene），如下图



为了解决数据库压力和速度的问题，我们的数据库就变成了索引库，我们使用Lucene的API的来操作服务器上的索引库。这样完全和数据库进行了隔离。

## 数据查询方法

### 顺序扫描法

所谓顺序扫描，例如要找内容包含一个字符串的文件，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。

这种方法是顺序扫描方法，数据量大就搜索慢。

### 倒排索引

先举一个栗子：

例如我们使用新华字典查询汉字，新华字典有偏旁部首的目录（索引），我们查字首先查这个目录，找到这个目录中对应的偏旁部首，就可以通过这个目录中的偏旁部首找到这个字所在的位置（文档）。

Lucene会对文档建立倒排索引

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

## 搜索技术应用场景

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）

# Lucene介绍

## 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

## 什么是Lucene

Lucene是apache软件基金会4 jakarta项目组的一个子项目，是一个开放源代码的全文检索引擎工具包，但它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎（英文与德文两种西方语言）。

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，或者是以此为基础建立起完整的全文检索引擎。

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

## Lucene与搜索引擎的区别

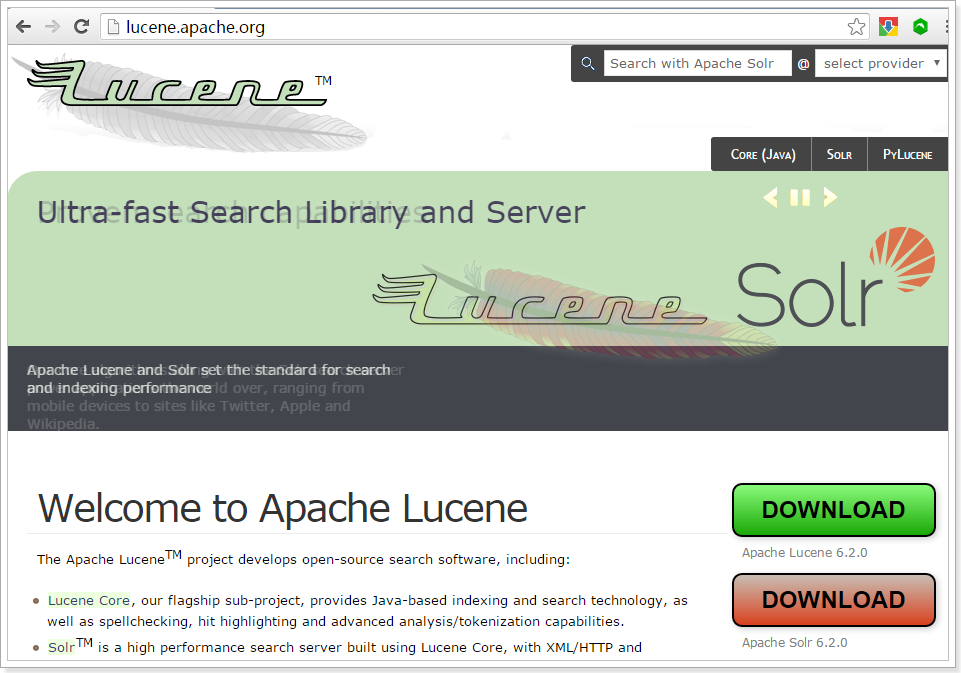
全文检索系统是按照全文检索理论建立起来的用于提供全文检索服务的软件系统，包括建立索引、处理查询返回结果集、增加索引、优化索引结构等功能。例如：百度搜索、eclipse帮助搜索、淘宝网商品搜索等。

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

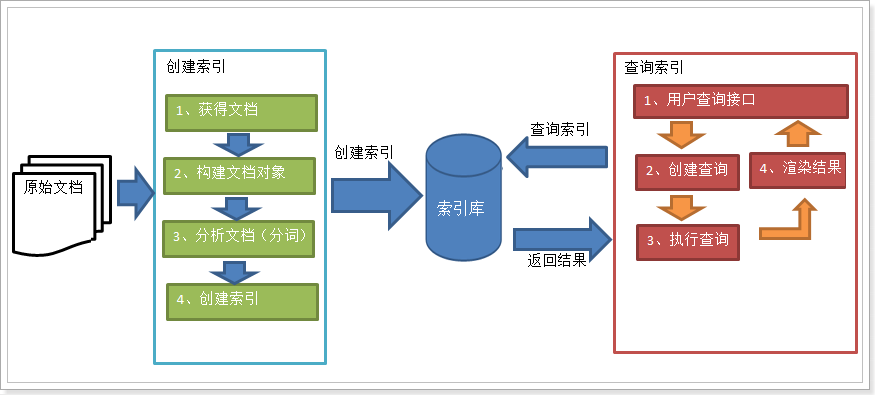
## Lucene官网

官网： http://lucene.apache.org/



# Lucene全文检索的流程

## 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🡪获得文档🡪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。

### 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。

### 获得文档（采集数据）

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，采集数据的目的是为了对原始内容进行索引。

采集数据分类：

1、对于互联网上网页，可以使用工具将网页抓取到本地生成html文件。

2、数据库中的数据，可以直接连接数据库读取表中的数据。

3、文件系统中的某个文件，可以通过I/O操作读取文件的内容。

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。

### 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，如下图：

Document（文档）

Field（域）

Name：id（图书id）

Value：1

Field（域）

Name：name（图书名称）

Value：lucene

Field（域）

Name：price（图书价格）

Value：66

Field（域）

Name：desc（图书描述）

Value：lucene是apache的开源项目，是java开发的检索的工具

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，分析成为一个一个的单词。

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.

分析后得到的词：

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）。

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。

倒排索引结构是根据内容（词汇）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。

### 用户

就是使用搜索的角色，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。如下图：



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查询关键字、要搜索的Field文档域等，查询对象会生成具体的查询语法，比如：

name:lucene表示要搜索name这个Field域中，内容为“lucene”的文档。

desc:lucene AND desc: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。

例如搜索语法为“desc:lucene AND desc:java”表示搜索出的文档中既要包括lucene也要包括java。

词lucene

文档lucene

文档solr

.....

词java

文档lucene

文档java编程思想

.....

1. 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。

### 渲染结果

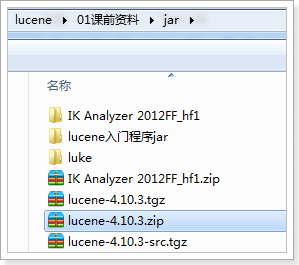
以一个友好的界面将查询结果展示给用户，用户根据搜索结果找自己想要的信息，为了帮助用户很快找到自己的结果，提供了很多展示的效果，比如搜索结果中将关键字高亮显示，百度提供的快照等。



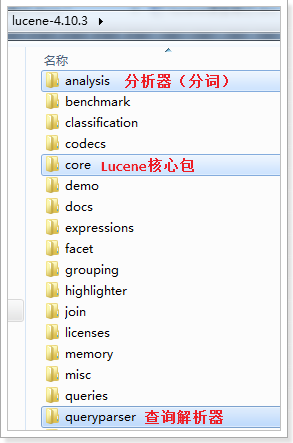
# Lucene入门

## Lucene准备

Lucene可以在官网上下载。课程已经准备好了Lucene的文件，我们使用的是4.10.3版本，文件位置如下图：



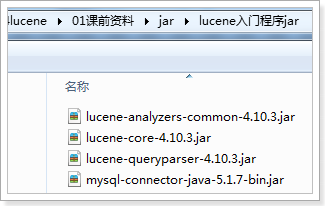
解压后的效果：



使用这三个文件的jar包，就可以实现lucene功能

本教程使用的数据是MySQL数据库的数据，所以还需要MySQL的连接包

学员编写的时候，也可以直接复制准备好的jar包，位置如下图：



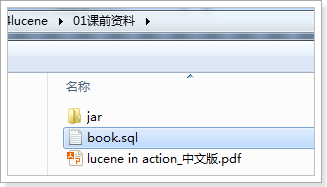
## 开发环境

JDK： 1.7 （Lucene4.8以上，必须使用JDK1.7及以上版本）

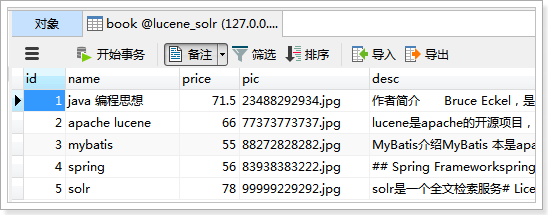
IDE： eclipse Mars2

数据库： MySQL

数据库脚本位置如下图：

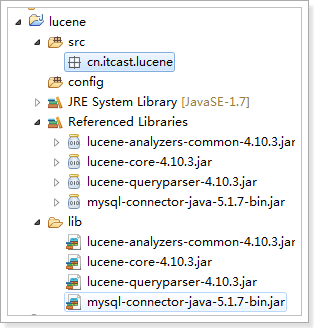


导入到MySQL效果如下图：



## 创建Java工程

创建java工程测试即可，效果如下：



## 索引流程

### 数据采集

在电商网站中，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。

#### 创建pojo

**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 desc;

get/set。。。

}

#### 创建DAO接口

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

/\*\*

\* 查询所有的book数据

\*

\* **@return**

\*/

List<Book> queryBookList();

}

#### 创建DAO接口实现类

使用jdbc实现

**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/solr", "root", "root");

// 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.setDesc(resultSet.getString("desc"));

list.add(book);

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** list;

}

}

### 实现索引流程

1. 采集数据
2. 创建Document文档对象
3. 创建分析器（分词器）
4. 创建IndexWriterConfig配置信息类
5. 创建Directory对象，声明索引库存储位置
6. 创建IndexWriter写入对象
7. 把Document写入到索引库中
8. 释放资源

**public** **class** CreateIndexTest {

@Test

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

// 1. 采集数据

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

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

// 2. 创建Document文档对象

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

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

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

// Document文档中添加Field域

// 图书Id

// Store.YES:表示存储到文档域中

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

// 图书名称

document.add(**new** TextField("name", book.getName().toString(), Store.***YES***));

// 图书价格

document.add(**new** TextField("price", book.getPrice().toString(), Store.***YES***));

// 图书图片地址

document.add(**new** TextField("pic", book.getPic().toString(), Store.***YES***));

// 图书描述

document.add(**new** TextField("desc", book.getDesc().toString(), Store.***YES***));

// 把Document放到list中

documents.add(document);

}

// 3. 创建Analyzer分词器,分析文档，对文档进行分词

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

// 4. 创建Directory对象,声明索引库的位置

Directory directory = FSDirectory.*open*(**new** File("C:/itcast/lucene/index"));

// 5. 创建IndexWriteConfig对象，写入索引需要的配置

IndexWriterConfig config = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***, analyzer);

// 6.创建IndexWriter写入对象

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

// 7.写入到索引库，通过IndexWriter添加文档对象document

**for** (Document doc : documents) {

indexWriter.addDocument(doc);

}

// 8.释放资源

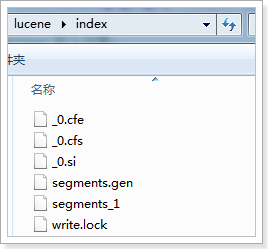
indexWriter.close();

}

}

执行效果：

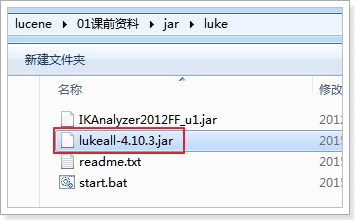
在文件夹中出现了以下文件，表示创建索引成功



### 使用Luke查看索引

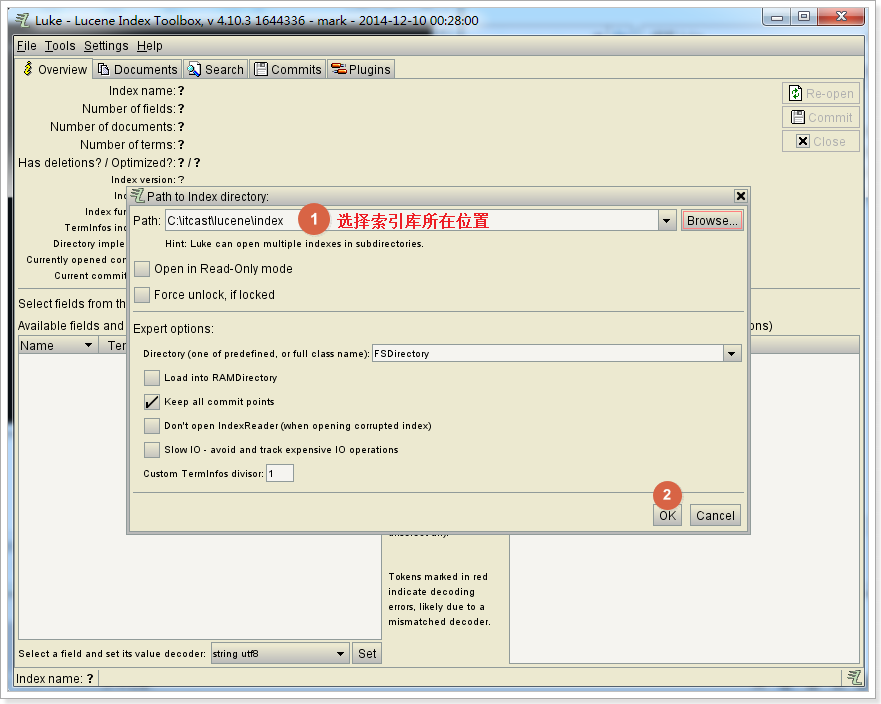
Luke作为Lucene工具包中的一个工具（http://www.getopt.org/luke/），可以通过界面来进行索引文件的查询、修改

luke所在位置如下图：

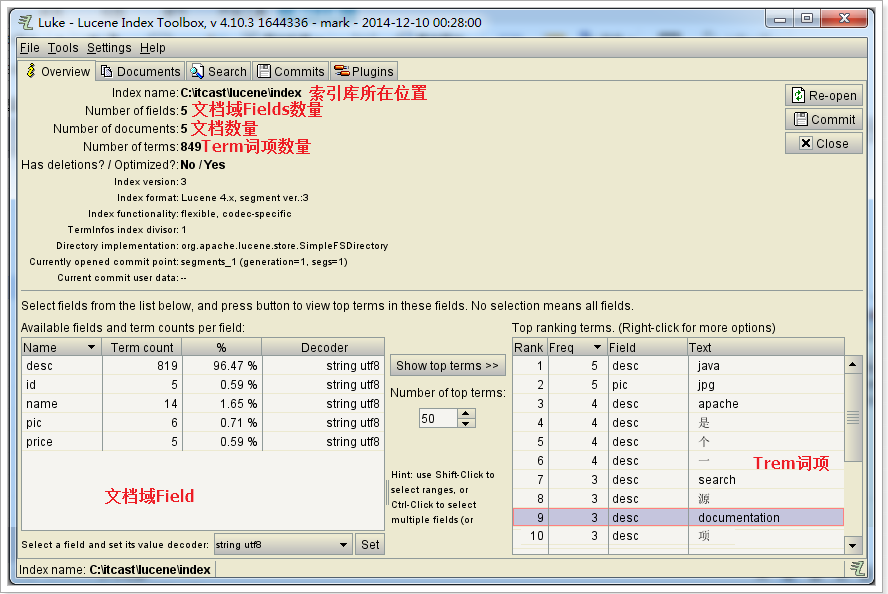


打开Luke方法：打开cmd命令行运行命令：java -jar lukeall-4.10.3.jar

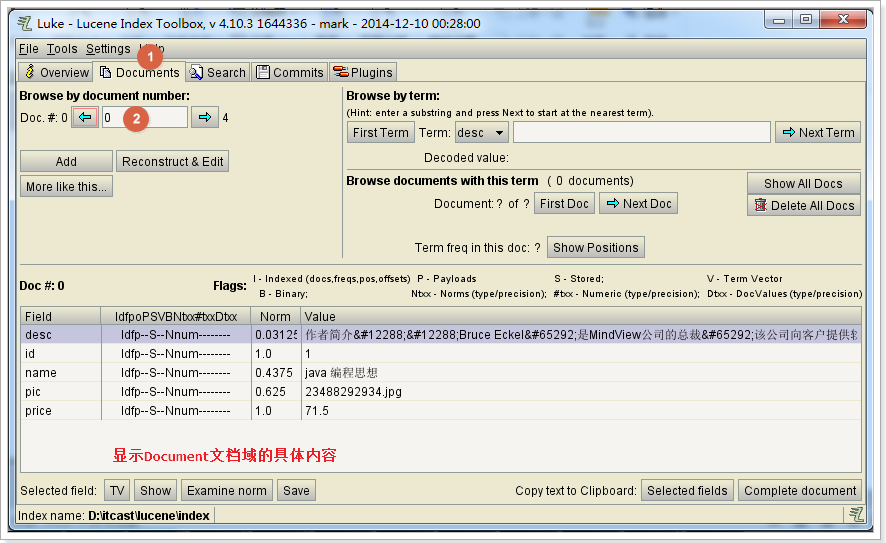
打开后，使用如下图：



下图是索引域的展示效果：



下图是文档域展示效果



## 搜索流程

### 输入查询语句

Lucene可以通过query对象输入查询语句。同数据库的sql一样，lucene也有固定的查询语法：

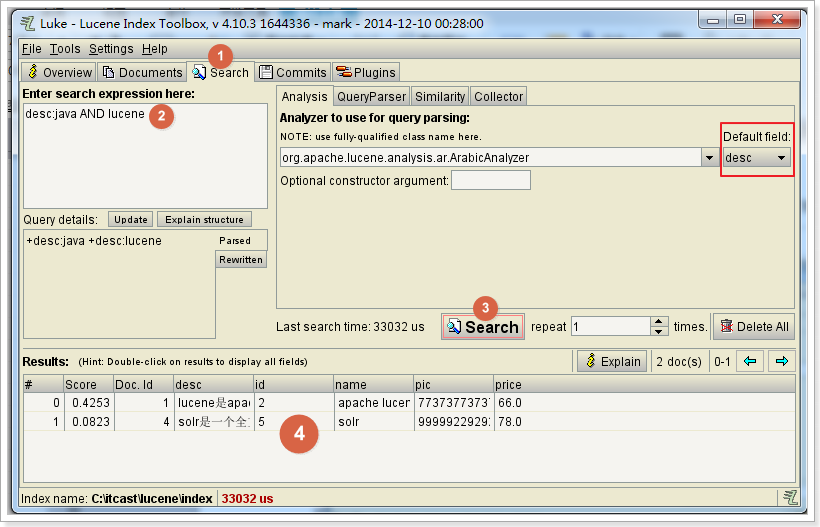
最基本的有比如：AND, OR, NOT 等（必须大写）

举个栗子:

用户想找一个desc中包括java关键字和lucene关键字的文档。

它对应的查询语句：desc:java AND desc:lucene

如下图是使用luke搜索的例子：



#### 搜索分词

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学习”，分词后为java和学习两个词，与java和学习有关的内容都搜索出来了，如下：



### 代码实现

1. 创建Query搜索对象

2. 创建Directory流对象,声明索引库位置

3. 创建索引读取对象IndexReader

4. 创建索引搜索对象IndexSearcher

5. 使用索引搜索对象，执行搜索，返回结果集TopDocs

6. 解析结果集

7. 释放资源

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 |

代码实现

public class SearchIndexTest {

@Test

public void testSearchIndex() throws Exception {

// 1. 创建Query搜索对象

// 创建分词器

Analyzer analyzer = new StandardAnalyzer();

// 创建搜索解析器，第一个参数：默认Field域，第二个参数：分词器

QueryParser queryParser = new QueryParser("desc", analyzer);

// 创建搜索对象

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

// 2. 创建Directory流对象,声明索引库位置

Directory directory = FSDirectory.*open*(new File("C:/itcast/lucene/index"));

// 3. 创建索引读取对象IndexReader

IndexReader reader = DirectoryReader.*open*(directory);

// 4. 创建索引搜索对象

IndexSearcher searcher = new IndexSearcher(reader);

// 5. 使用索引搜索对象，执行搜索，返回结果集TopDocs

// 第一个参数：搜索对象，第二个参数：返回的数据条数，指定查询结果最顶部的n条数据返回

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

System.*out*.println("查询到的数据总条数是：" + topDocs.totalHits);

// 获取查询结果集

ScoreDoc[] docs = topDocs.scoreDocs;

// 6. 解析结果集

for (ScoreDoc scoreDoc : docs) {

// 获取文档

int docID = scoreDoc.doc;

Document doc = searcher.doc(docID);

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

System.*out*.println("docID:" + docID);

System.*out*.println("bookId:" + doc.get("id"));

System.*out*.println("name:" + doc.get("name"));

System.*out*.println("price:" + doc.get("price"));

System.*out*.println("pic:" + doc.get("pic"));

// System.out.println("desc:" + doc.get("desc"));

}

// 7. 释放资源

reader.close();

}

}

# Field域类型

## 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。

* **是否分词(tokenized)**

是：作分词处理，即将Field值进行分词，分词的目的是为了索引。

比如：商品名称、商品描述等，这些内容用户要输入关键字搜索，由于搜索的内容格式大、内容多需要分词后将语汇单元建立索引

否：不作分词处理

比如：商品id、订单号、身份证号等

* **是否索引(indexed)**

是：进行索引。将Field分词后的词或整个Field值进行索引，存储到索引域，索引的目的是为了搜索。

比如：商品名称、商品描述分析后进行索引，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。

否：不索引。

比如：图片路径、文件路径等，不用作为查询条件的不用索引。

* **是否存储(stored)**

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

比如：商品名称、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

比如：商品描述，内容较大不用存储。如果要向用户展示商品描述可以从系统的关系数据库中获取。

## 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 | 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) | Long型 | 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  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修改

### 修改分析

图书id：

是否分词：不用分词，因为不会根据商品id来搜索商品

是否索引：不索引，因为不需要根据图书ID进行搜索

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，因为要根据图书名称的关键词搜索。

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，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。

不存储是不在lucene的索引域中记录，节省lucene的索引文件空间。

如果要在详情页面显示描述，解决方案：

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

### 代码修改

对之前编写的testCreateIndex()方法进行修改。

代码片段

// Document文档中添加域

// 图书Id

// Store.YES:表示存储到文档域中

// 不分词，不索引，储存

document.add(**new** StoredField("id", book.getId().toString()));

// 图书名称

// 分词，索引，储存

document.add(**new** TextField("name", book.getName().toString(), Store.***YES***));

// 图书价格

// 分词，索引，储存

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

// 图书图片地址

// 不分词，不索引，储存

document.add(**new** StoredField("pic", book.getPic().toString()));

// 图书描述

// 分词，索引，不储存

document.add(**new** TextField("desc", book.getDesc().toString(), Store.***NO***));

# 索引维护

## 需求

管理人员通过电商系统更改图书信息，这时更新的是关系数据库，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。

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。

@Test

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

// 创建Directory流对象

Directory directory = FSDirectory.*open*(**new** File("C:/itcast/lucene/index"));

IndexWriterConfig config = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***, **null**);

// 创建写入对象

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

// 根据Term删除索引库，name:java

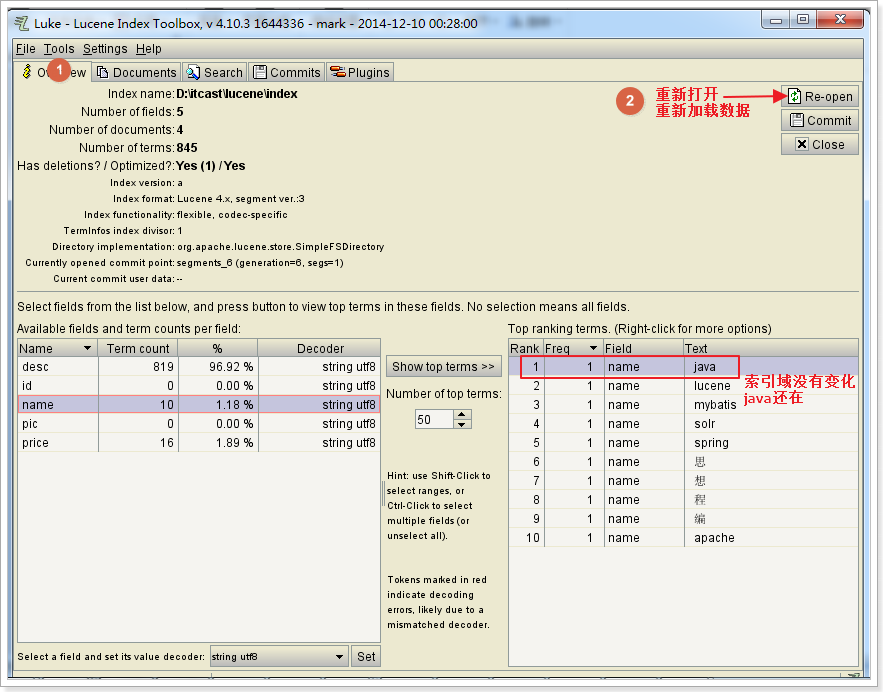
indexWriter.deleteDocuments(**new** Term("name", "java"));

// 释放资源

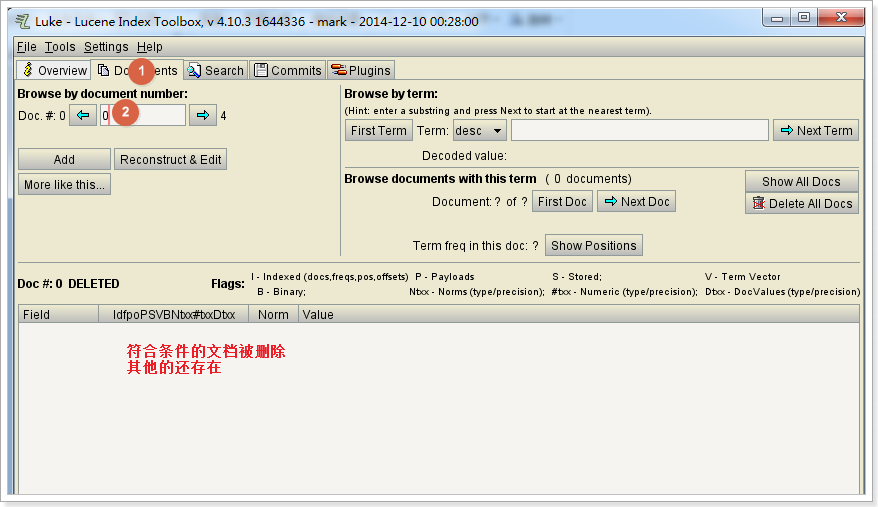
indexWriter.close();

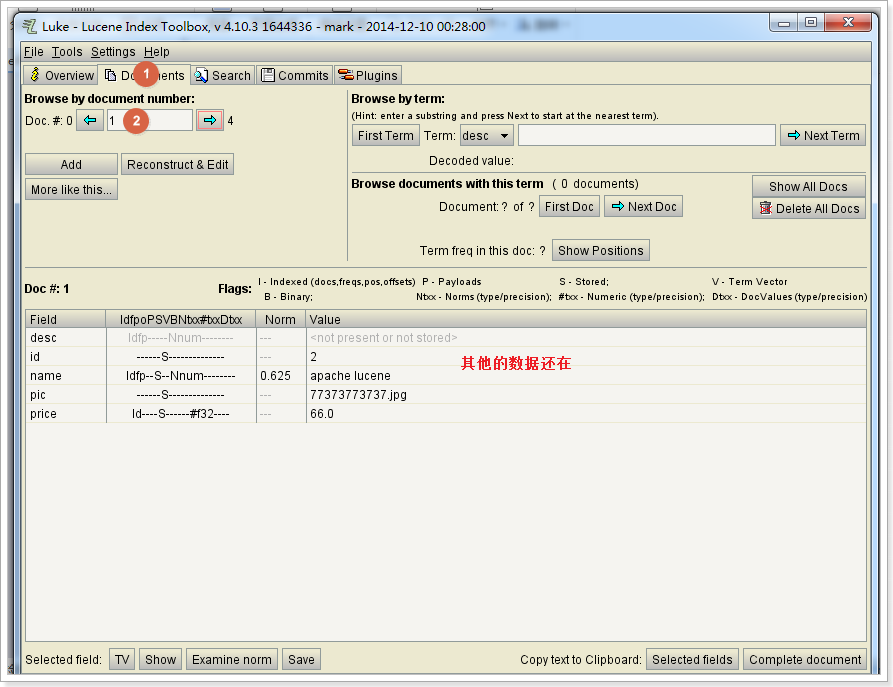
}

效果如下图：索引域没有变化



文档域数据被删除掉





### 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。

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。

代码：

@Test

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

// 创建Directory流对象

Directory directory = FSDirectory.*open*(**new** File("D:/itcast/lucene/index"));

IndexWriterConfig config = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***, **null**);

// 创建写入对象

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

// 根据Term删除索引库，name:java

// indexWriter.deleteDocuments(new Term("name", "java"));

// 全部删除

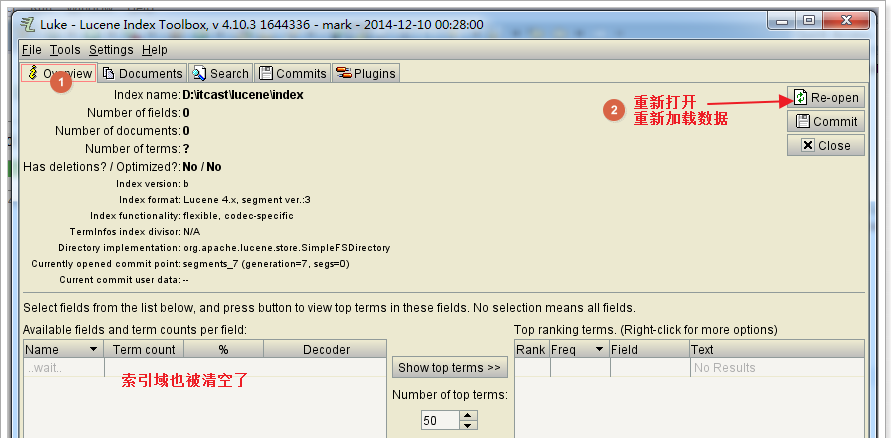
indexWriter.deleteAll();

// 释放资源

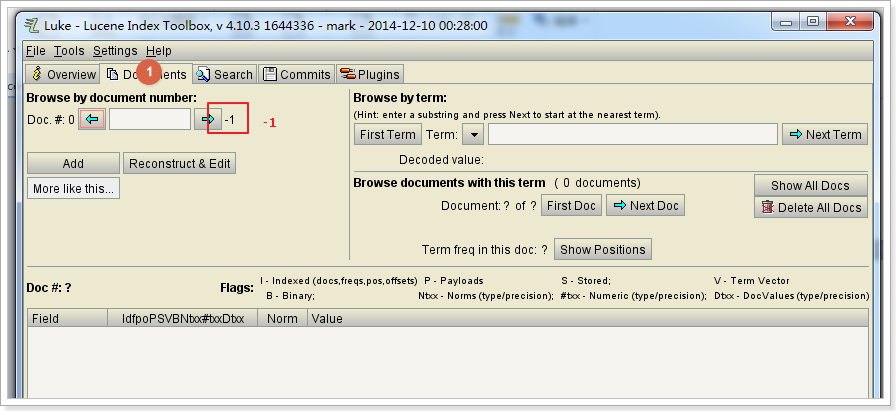
indexWriter.close();

}

索引域数据清空



文档域数据也清空



## 修改索引

更新索引是先删除再添加，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，确定更新记录存在执行更新操作。

如果更新索引的目标文档对象不存在，则执行添加。

代码

@Test

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

// 创建分词器

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

// 创建Directory流对象

Directory directory = FSDirectory.*open*(**new** File("C:/itcast/lucene/index"));

IndexWriterConfig config = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***, analyzer);

// 创建写入对象

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

// 创建Document

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

document.add(**new** TextField("id", "1002", Store.***YES***));

document.add(**new** TextField("name", "lucene测试test 002", Store.***YES***));

// 执行更新，会把所有符合条件的Document删除，再新增。

indexWriter.updateDocument(**new** Term("name", "test"), document);

// 释放资源

indexWriter.close();

}

# 分词器

## 分词理解

在对Docuemnt中的内容进行索引之前，需要使用分词器进行分词 ，分词的目的是为了搜索。分词的主要过程就是先分词后过滤。

* 分词：采集到的数据会存储到document对象的Field域中，分词就是将Document中Field的value值切分成一个一个的词。
* 过滤：包括去除标点符号过滤、去除停用词过滤（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。

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。

对于分词来说，不同的语言，分词规则不同。Lucene作为一个工具包提供不同国家的分词器，本例子使用StandardAnalyzer，它可以对用英文进行分词。

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);

**return** **new** TokenStreamComponents(src, tok) {

@Override

**protected** **void** setReader(**final** Reader reader) **throws** IOException {

src.setMaxTokenLength(StandardAnalyzer.**this**.maxTokenLength);

**super**.setReader(reader);

}

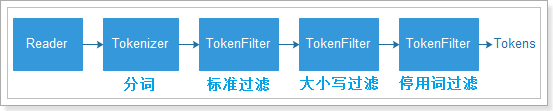
};

}

Tokenizer就是分词器，负责将reader转换为语汇单元即进行分词处理，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。

比如下边的文档经过分析器分析如下：

* 原文档内容：

Lucene is a Java full-text search engine.

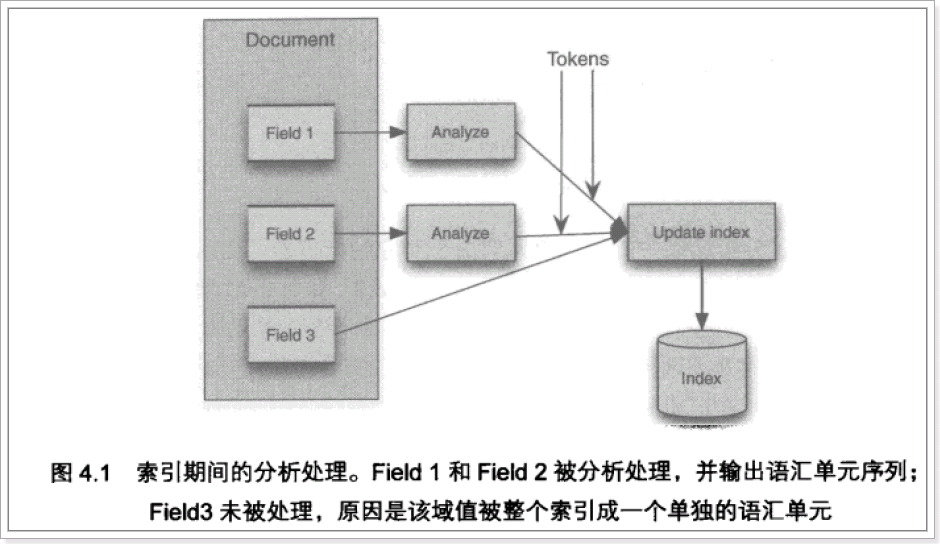
* 分析后得到的多个语汇单元：

lucene 、java、full、text、search、engine

## 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。**

## 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。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。

而中文则以字为单位，字又组成词，字和词再组成句子。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

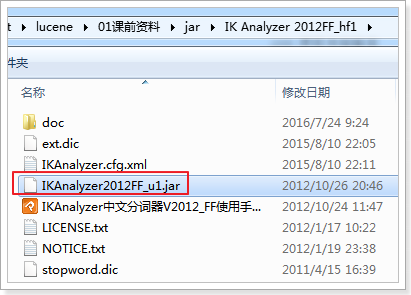
* paoding： 庖丁解牛最新版在 <https://code.google.com/p/paoding/> 中最多支持Lucene 3.0，且最新提交的代码在 2008-06-03，在svn中最新也是2010年提交，已经过时，不予考虑。
* mmseg4j：最新版已从 <https://code.google.com/p/mmseg4j/> 移至 <https://github.com/chenlb/mmseg4j-solr>，支持Lucene 4.10，且在github中最新提交代码是2014年6月，从09年～14年一共有：18个版本，也就是一年几乎有3个大小版本，有较大的活跃度，用了mmseg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。
* 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自带的分析器方法一样，将Analyzer测试代码改为IKAnalyzer测试中文分词效果。

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

### 添加jar包



### 修改分词器代码

@Test

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

// 1. 采集数据

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

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

// 2. 创建Document文档对象

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

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

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

// Document文档中添加Field域

// 图书Id

// Store.YES:表示存储到文档域中

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

// 图书名称

document.add(**new** TextField("name", book.getName().toString(), Store.***YES***));

// 图书价格

document.add(**new** TextField("price", book.getPrice().toString(), Store.***YES***));

// 图书图片地址

document.add(**new** TextField("pic", book.getPic().toString(), Store.***YES***));

// 图书描述

document.add(**new** TextField("desc", book.getDesc().toString(), Store.***YES***));

// 把Document放到list中

documents.add(document);

}

// 3. 创建Analyzer分词器,分析文档，对文档进行分词

// Analyzer analyzer = new StandardAnalyzer();

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

// 4. 创建Directory对象,声明索引库的位置

Directory directory = FSDirectory.*open*(**new** File("C:/itcast/lucene/index"));

// 5. 创建IndexWriteConfig对象，写入索引需要的配置

IndexWriterConfig config = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***, analyzer);

// 6.创建IndexWriter写入对象

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

// 7.写入到索引库，通过IndexWriter添加文档对象document

**for** (Document doc : documents) {

indexWriter.addDocument(doc);

}

// 8.释放资源

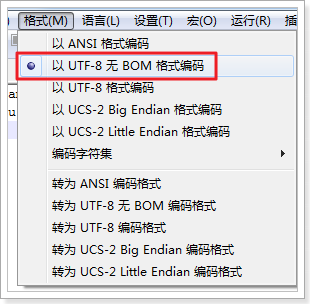
indexWriter.close();

}

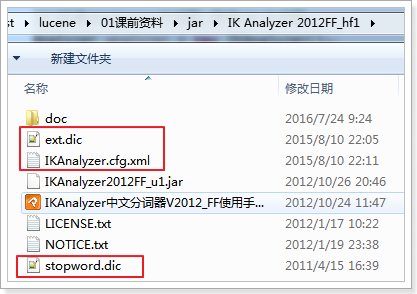
## 扩展中文词库

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。

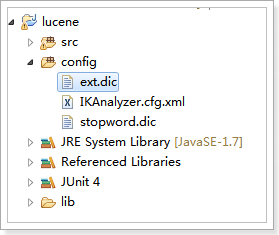
注意：不要用window自带的记事本保存扩展词文件和停用词文件，那样的话，格式中是含有bom的。



从ikanalyzer包中拷贝配置文件



拷贝到资源文件夹中



IKAnalyzer.cfg.xml配置文件

<?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 -->

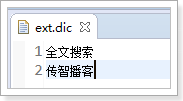
<entry key=*"ext\_dict"*>ext.dic;</entry>

<!--用户可以在这里配置自己的扩展停止词字典-->

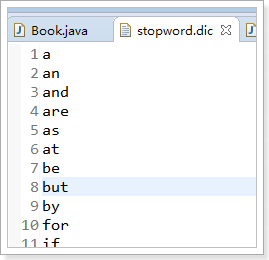
<entry key=*"ext\_stopwords"*>stopword.dic;</entry>

</properties>

中文词库，添加新词的地方



stopword.dic是存放停用词的地方



最终分词效果

