# 11.Lucene

## 11.1.全文检索概述

### 11.1.1.背景

2014年数据

YouTube 1分钟上传视频400小时

亚马逊1分钟交易60万美金

google1分钟翻译7000万单词

siri1分钟回答很多问题

优点：信息丰富，可以获取各种资源

缺点：信息量太大，搜索、获取变得复杂，搜的慢，准确性低

### 11.1.2.全文检索

**介绍**

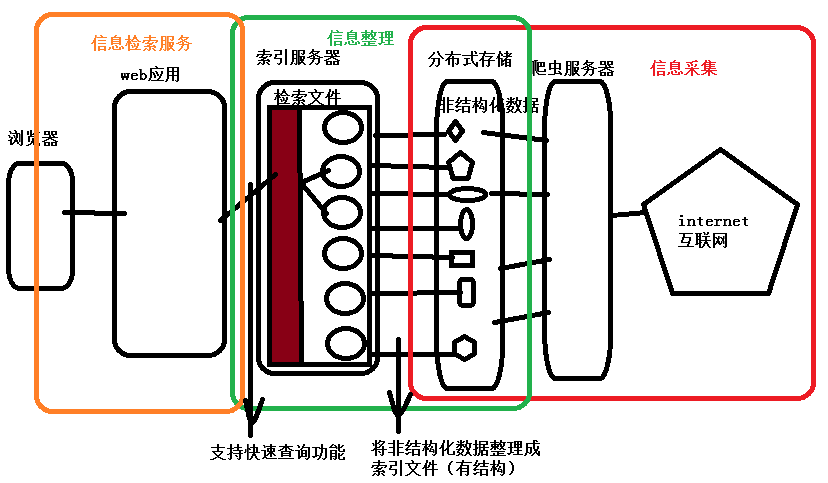
百度百科：全文数据库是全文检索系统的主要构成部分。所谓全文数据库是将一个完整的信息源的全部内容转化为计算机可以识别、处理的信息单元而形成的数据集合。全文数据库不仅存储了信息，而且还有对全文数据进行词、字、段落等更深层次的编辑、加工的功能，而且所有全文数据库无一不是海量信息数据库。

**搜索引擎结构**

信息采集：将数据收集过来（大数据存储）

信息整理：对数据进行计算整理（全文检索结构）

信息检索服务：创建web应用，供客户端搜索



## 11.2.lucene概述

### 11.2.1.lucene介绍

由美国搜索的领军人物Doug Cutting（狗哥），2000年创建的一个搜索引擎工具包

特点：

1.基于java开发

2.稳定，索引性能高（速度快，创建索引、搜索索引）

3.现代磁盘单机索引每小时150G数据

4.对内存要求低-1mb栈内存

5.索引数据结构只占整体索引文件的20%

6.支持多种搜索功能：短语搜索、词项搜索、关键字搜索、布尔搜索、范围搜索

### 11.2.2.需要了解的概念

**查询（query）**：

全文检索技术查询，实际上是一个复杂的内存对象，对不同的查询功能封装的实现类。

**文档（document）：**

搜索的索引文件中保存的有结构的数据对象，表示一个领域不可拆分的数据单位（一个网页、一个商品）

**域属性（field）：**

一个文档作为数据整体不可拆分，但是一个文档对象的所有数据都是组合成的，通过field域属性组合（相当于数据库rows是由多个column组成）

**词项（term）：**

具备最小意义的一个词语，查询关键字或者数据本身的字符串，可以经过词条化处理，处理结果就是一个一个词项

例如：腾讯首席执行官马华腾的身价已经超过了范冰冰

词条化处理：

腾讯、首席、执行、执行官、马华、马华腾、的、身价、已经、超过了、范冰冰

### 11.2.3.lucene的索引文件-倒排索引算法

正向索引：wordcount.txt hello(5) word(3)...

倒排索引：hello(5 wordcount.txt\hello.txt)

**（1）获取源数据（网页）**

新浪网页1：

title：美国又一次发动了贸易战

content：特朗普发表声明，在消灭伊朗之前，先撮一撮中国经济

publish：新浪，新华网

网易网页2：

title：美国贸易战不怕死

content：中国外交官痛批特朗普的声明

publish：网易，新华网

author：莫言

**（2）封装源数据为document**

document1：

-fileds：

url：[www.sina.com/a/b/cc](http://www.sina.com/a/b/cc)

title：美国又一次发动了贸易战

content：特朗普发表声明，在消灭伊朗之前，先贸易战撮一撮中国经济

publish：新浪，新华网

document2:

-fileds：

url：[www.163.com/5/6/88](http://www.163.com/5/6/88)

title：美国贸易战不怕死

content：中国外交官痛批特朗普的声明

publish：网易，新华网

author：莫言

**（3）计算分词并合并结果**

分词：词项、documentId、filed、频率、偏移量、位移等参数

document1（id=1）

美国（1,title），贸易战（1,[title,content]），特朗普（1,content），中国（1,content）

新华网（1,publish）

document2（id=2）

美国（2,title），贸易战（2,[title]），特朗普（2,content），中国（2,content）

新华网（2,publish）

合并分词：

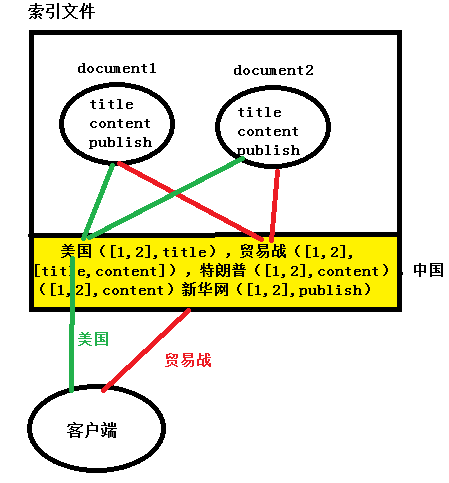
美国（[1,2],title），贸易战（[1,2],[title,content]），特朗普（[1,2],content），中国（[1,2],content）

新华网（[1,2],publish）

最终，这些分词需要放入索引文件的所有结构数据中

**（4）将document输出到索引文件保存**

将document和词项经过整理存入索引文件



## 11.3.lucene测试代码

### 11.3.1.lucene分词实现

lucene为了支持丰富的语言环境分词计算，lucene只实现了部分分词（主要是英文），对于没有实现的语言环境，提供了实现的接口

### 11.3.2.lucene自带分词器

1.创建maven工程（quickstart）

2.pom引入相关依赖（课前资料\01-课前资料\09-Lucene\lucene测试依赖.txt）

|  |
| --- |
| <dependency> <!-- 查询相关jar包 -->  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-queryparser</artifactId>  <version>6.0.0</version>  </dependency>  <dependency> <!-- lucene自带只能中文分词器jar包 -->  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-analyzers-smartcn</artifactId>  <version>6.0.0</version>  </dependency>  <dependency> <!-- 测试用到的lucene工具包 -->  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-analyzers-common</artifactId>  <version>6.0.0</version>  </dependency>  <dependency> <!-- 测试用到的lucene核心包 -->  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-core</artifactId>  <version>6.0.0</version>  </dependency> |

3.测试分词器

中文为例

standarAnalyzer：标准分词器：每个字分开

smartChineseAnalyzer：智能中文分词器：按照词语分隔

simpleAnalyzer：简单分词器：按照符号分隔

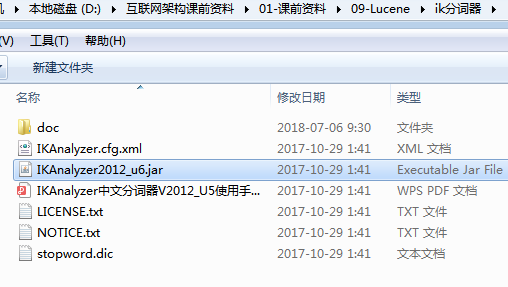
whitspaceAnalyzer：空格分词器：按照空格分隔

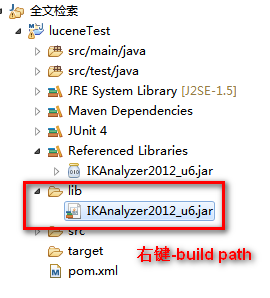
|  |
| --- |
| **package** cn.tedu.lucene.test;  **import** java.io.StringReader;  **import** org.apache.lucene.analysis.Analyzer;  **import** org.apache.lucene.analysis.TokenStream;  **import** org.apache.lucene.analysis.cn.smart.SmartChineseAnalyzer;  **import** org.apache.lucene.analysis.core.SimpleAnalyzer;  **import** org.apache.lucene.analysis.core.WhitespaceAnalyzer;  **import** org.apache.lucene.analysis.standard.StandardAnalyzer;  **import** org.apache.lucene.analysis.tokenattributes.CharTermAttribute;  **import** org.junit.Test;  **public** **class** AnalyzerTest {  //定义一个方法，传入两个参数：分词器和字符串，根据传入的分词器和字符串进行分词  **public** **void** printAnalyzer(Analyzer analyzer,String msg) **throws** Exception{  //获取流读取字符串  StringReader reader = **new** StringReader(msg);  //将字符串流和分词器流对接  TokenStream token = analyzer.tokenStream("test", reader);  //从token中获取词项  //将token的指针重置  token.reset();  //定义一个attibute，用来将分词封装到attibude中  CharTermAttribute attribute = token.getAttribute(CharTermAttribute.**class**);  //获取分词计算结果属性  **while**(token.incrementToken()){  System.***out***.println(attribute.toString());  }  }  @Test  **public** **void** run() **throws** Exception{  Analyzer a1 = **new** StandardAnalyzer();  Analyzer a2 = **new** SimpleAnalyzer();  Analyzer a3 = **new** WhitespaceAnalyzer();  Analyzer a4 = **new** SmartChineseAnalyzer();  String msg = "我喜欢玩,王者荣耀和英雄联盟";  System.***out***.println("\*\*\*\*\*\*\*\*\*标准分词器\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  printAnalyzer(a1, msg);  System.***out***.println("\*\*\*\*\*\*\*\*\*简单分词器\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  printAnalyzer(a2, msg);  System.***out***.println("\*\*\*\*\*\*\*\*\*空格分词器\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  printAnalyzer(a3, msg);  System.***out***.println("\*\*\*\*\*\*\*\*\*智能中文分词器\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  printAnalyzer(a4, msg);  }  } |

### 11.3.3.IK分词器

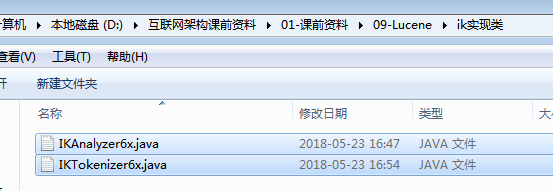
在中文分词器中，IK分词器比较常用，可以支持es的插件配置；ik分词器除了可以计算内嵌的分词逻辑，还可以做停用和扩展

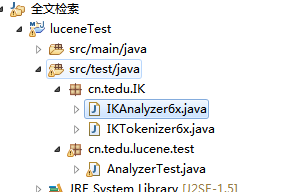
导包（课前资料\01-课前资料\09-Lucene\ik分词器\IKAnalyzer2012\_u6.jar）





将IK分词器实现类拷贝到项目中



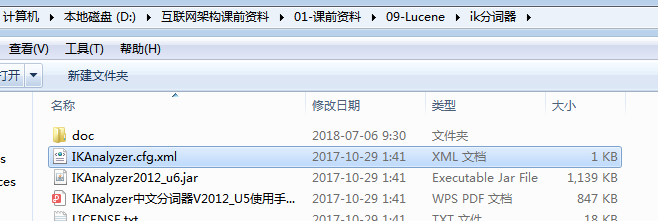


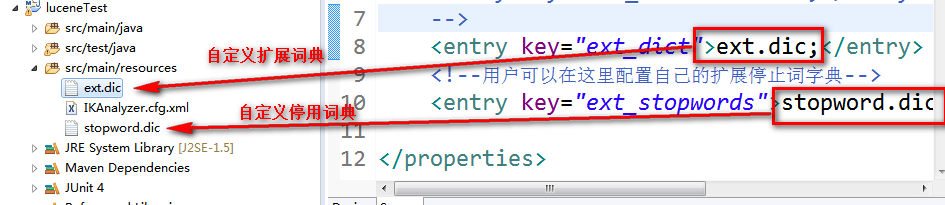
在测试代码中添加IK分词器测试：

|  |
| --- |
| Analyzer a5 = **new** IKAnalyzer6x();  System.***out***.println("\*\*\*\*\*\*\*\*\*智能中文分词器\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  printAnalyzer(a5, msg); |

### 11.3.4.对IK分词器扩展与停用

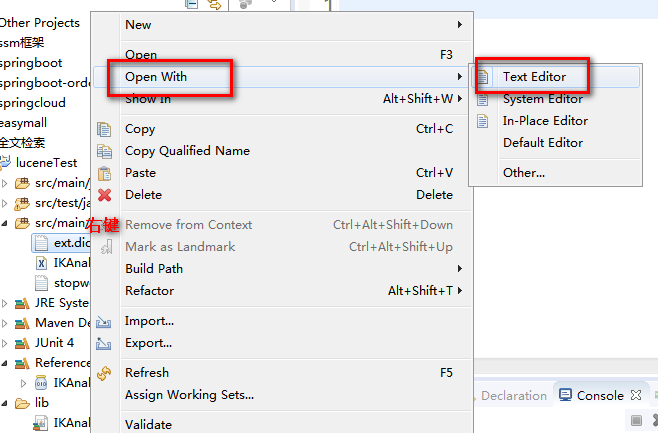
引入IKAnalyzer.cfg.xml配置文件（课前资料\01-课前资料\09-Lucene\ik分词器\IKAnalyzer.cfg.xml）拷贝到当前项目的src/main/resources



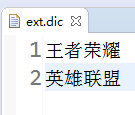


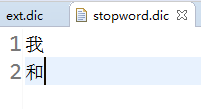
注意：新建的ext.dic和stopword.dic的编码需要与项目编码保持一致（utf-8）

这两个文件不能用windows编辑器打开



配置扩展词典和停用词典：





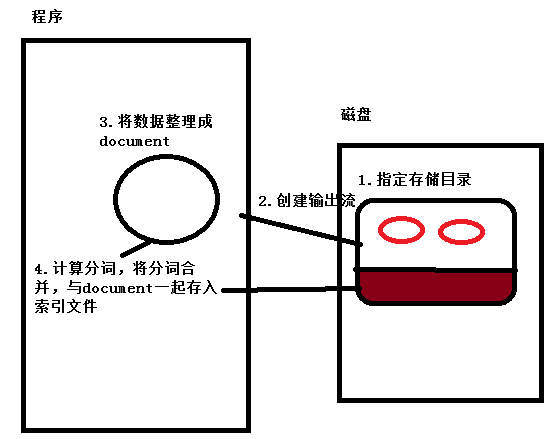
再次执行分词：

## 11.4.对索引的操作

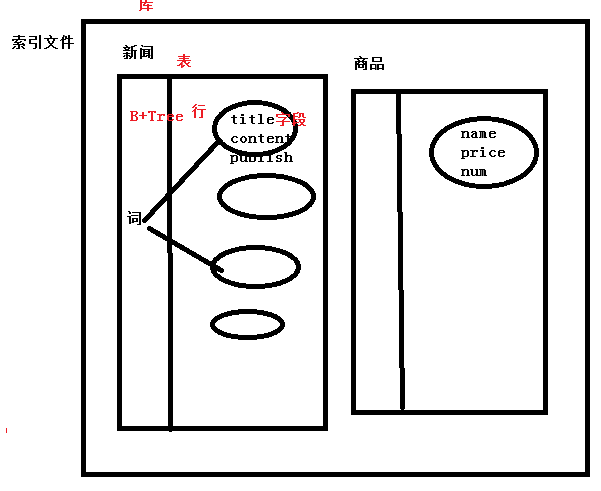
### 11.4.1.创建索引文件

读取源数据（jdbc读取），案例手动填写数据，实现几个document数据存储到索引文件

步骤：



**索引文件类比数据库**



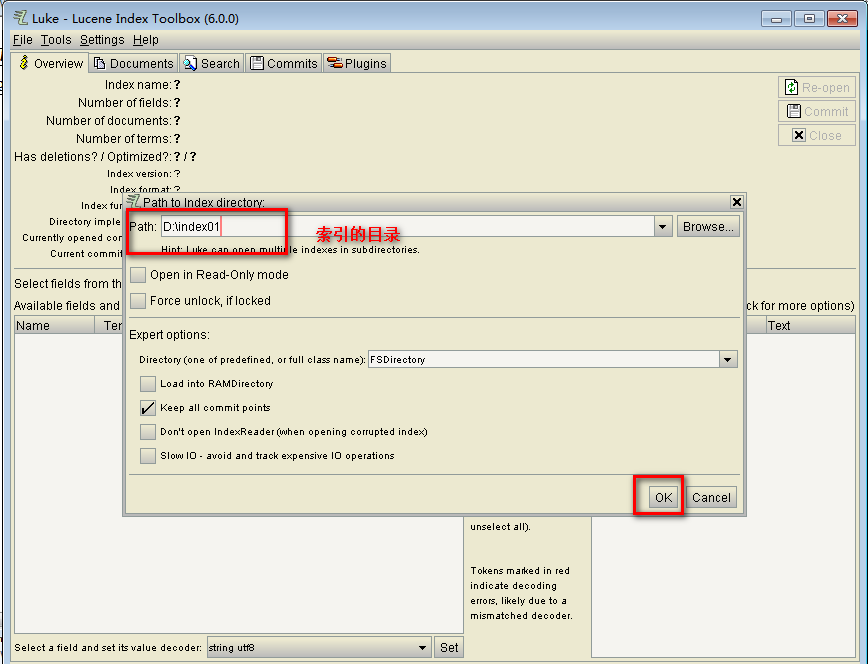
**创建索引代码：**

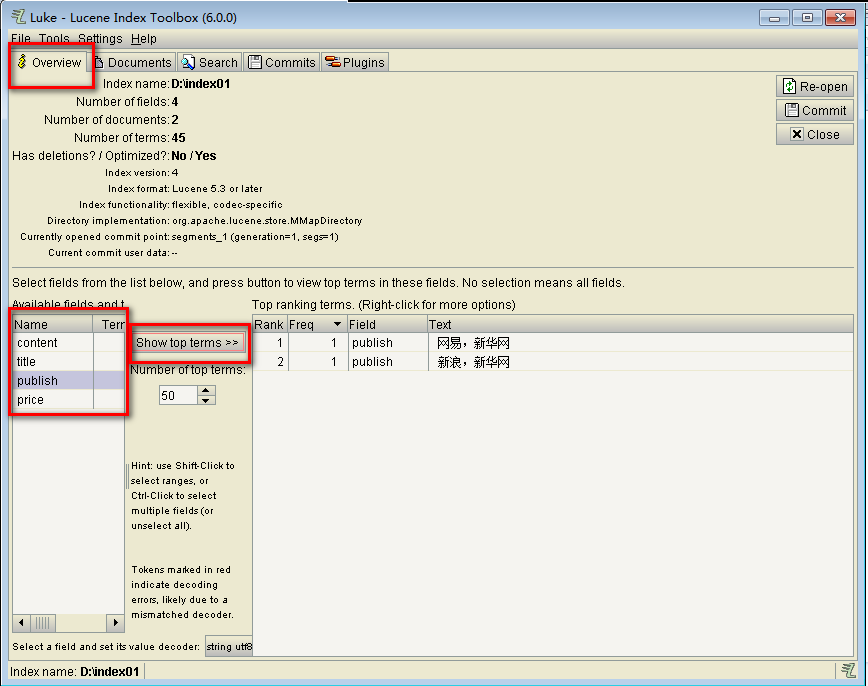
|  |
| --- |
| **package** cn.tedu.lucene.test;  **import** java.nio.file.Path;  **import** java.nio.file.Paths;  **import** org.apache.lucene.analysis.Analyzer;  **import** org.apache.lucene.document.Document;  **import** org.apache.lucene.document.DoublePoint;  **import** org.apache.lucene.document.Field.Store;  **import** org.apache.lucene.document.StringField;  **import** org.apache.lucene.document.TextField;  **import** org.apache.lucene.index.IndexWriter;  **import** org.apache.lucene.index.IndexWriterConfig;  **import** org.apache.lucene.index.IndexWriterConfig.OpenMode;  **import** org.apache.lucene.store.FSDirectory;  **import** org.junit.Test;  **import** cn.tedu.IK.IKAnalyzer6x;  **public** **class** CreateIndex {  @Test  **public** **void** createIndex() **throws** Exception{  //1.指定存储目录 D:/index01  Path path = Paths.*get*("D:/index01");  //2.将路径封装成lucene能识别的dir对象  FSDirectory dir = FSDirectory.*open*(path);  //3.创建IK分词器  Analyzer analyzer = **new** IKAnalyzer6x();  //创建一个输出配置，将分词器配置进去  IndexWriterConfig config = **new** IndexWriterConfig(analyzer);  //设置输出时是覆盖还是追加  config.setOpenMode(OpenMode.***CREATE***);  //4.将配置指定到输出流  IndexWriter writer = **new** IndexWriter(dir,config);  //5.准备2个document  Document doc1 = **new** Document();  Document doc2 = **new** Document();  //6.为document添加数据  doc1.add(**new** TextField("title","美国又一次发动了贸易战",Store.***YES***));  doc1.add(**new** TextField("content",  "特朗普发表声明，在消灭伊朗之前，先贸易战撮一撮中国经济",Store.***NO***));  doc1.add(**new** StringField("publish","新浪，新华网", Store.***YES***));  doc2.add(**new** TextField("title","美国贸易战不怕死",Store.***YES***));  doc2.add(**new** TextField("content",  "中国外交官痛批特朗普的声明",Store.***YES***));  doc2.add(**new** StringField("publish","网易，新华网", Store.***YES***));  doc2.add(**new** DoublePoint("price", 5000));  //将document写到索引文件中  writer.addDocument(doc1);  writer.addDocument(doc2);  writer.commit();  }  } |

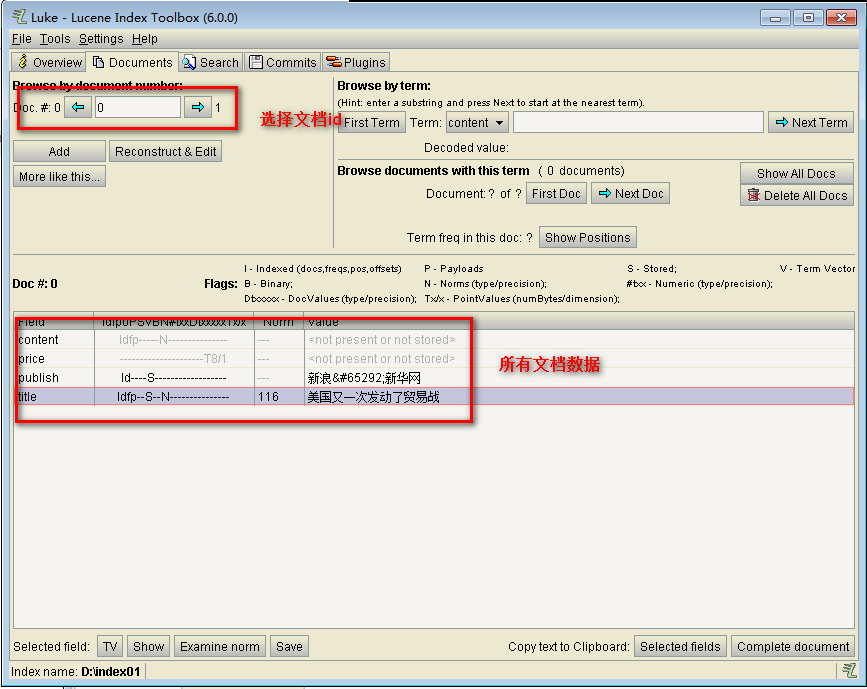
在D:/index01下生成一些二进制文件

利用工具luke查看（lucene写的小工具），需要与lucene的版本保持一致

课前资料\01-课前资料\09-Lucene\luke-6.0.0-luke-release\luke.bat







字符串相关

TextField:正常分词

StringField：完整保存，不分词（url、imgurl）

Store.NO\YES

如果是YES，正常存储，也正常分词（将分词放入索引结构）

如果是NO，不存储，但是正常分词，分词也会放入索引结构，可以通过这个词搜到这个文档，但是看到的文档中是看不到这个内容的

数字相关：

DoublePoint，IntPoint、LongPoint

不进行索引（不分词，也不存储数据），只保留域的数字特点

一般配置StringField一起使用，数字特点用来进行范围搜索，字符串用来显示

doc2.add(**new** DoublePoint("price", 5000));

doc2.add(**new** StringField("price","￥5000元",Store.***YES***));

lucene支持多种搜索功能

词项查询：TermQuery

多域查询：MultiFieldQuery

布尔查询：BooleanQuery

范围查询：Int\Double\Long rangQuery

模糊查询（用的少）

通配查询（用的少）

### 11.4.2.词项查询-TermQuery

通过指定一个域，指定一个词语，在索引搜索这个词项，将搜到的document返回，可以指定搜的条数

步骤：

1.指定索引文件位置

2.创建一个搜索对象searcher

3.构造一个具体的查询条件

4.遍历打印结果

|  |
| --- |
| **package** cn.tedu.lucene.test;  **import** java.io.IOException;  **import** java.nio.file.Path;  **import** java.nio.file.Paths;  **import** org.apache.lucene.document.Document;  **import** org.apache.lucene.index.DirectoryReader;  **import** org.apache.lucene.index.IndexReader;  **import** org.apache.lucene.index.Term;  **import** org.apache.lucene.search.IndexSearcher;  **import** org.apache.lucene.search.Query;  **import** org.apache.lucene.search.ScoreDoc;  **import** org.apache.lucene.search.TermQuery;  **import** org.apache.lucene.search.TopDocs;  **import** org.apache.lucene.store.FSDirectory;  **import** org.junit.Test;  **public** **class** IndexSearch {  //词项查询  @Test  **public** **void** termQuery() **throws** IOException{  //1.指定存储目录 D:/index01，并封装为dir  Path path = Paths.*get*("D:/index01");  FSDirectory dir = FSDirectory.*open*(path);  //2.构造搜索对象，基于输入流  IndexReader reader = DirectoryReader.*open*(dir);  IndexSearcher searcher = **new** IndexSearcher(reader);  //3.构造搜索实现类  Query query = **new** TermQuery(**new** Term("content","中国"));  //4.所搜并返回结果topDocs,topDocs封装了数据的评分、命中条数、所有数据  TopDocs topDocs = searcher.search(query,100);  //打印最大评分与命中条数  System.***out***.println("最大评分："+topDocs.getMaxScore());  System.***out***.println("命中条数："+topDocs.totalHits);  //获取所有的document  ScoreDoc[] scoreDocs = topDocs.scoreDocs;  //遍历scoreDocs  **for** (ScoreDoc scoreDoc : scoreDocs) {  //scoreDoc封装了document以及它的评分、Id等  System.***out***.println("当前document的评分为："+scoreDoc.score);  System.***out***.println("当前document的Id为："+scoreDoc.doc);  //获取document  Document document = searcher.doc(scoreDoc.doc);  //打印document信息  System.***out***.println("title:"+document.get("title"));  System.***out***.println("content:"+document.get("content"));  System.***out***.println("publish:"+document.get("publish"));  System.***out***.println("price:"+document.get("price"));  }  }  } |

### 11.4.3.多域查询-MultiFieldQuery

通过指定多个域，指定一个查询语句，可以对查询语句进行分词解析，然后对分词结果查询并评分，将搜到的document返回，可以指定搜的条数

|  |
| --- |
| @Test  **public** **void** termQuery() **throws** Exception{  //1.指定存储目录 D:/index01，并封装为dir  Path path = Paths.*get*("D:/index01");  FSDirectory dir = FSDirectory.*open*(path);  //2.构造搜索对象，基于输入流  IndexReader reader = DirectoryReader.*open*(dir);  IndexSearcher searcher = **new** IndexSearcher(reader);  //3.构造词项搜索实现类  //Query query = new TermQuery(new Term("content","中国"));    //3.2.构造多域查询的实现类  //准备查询域的数组  String[] fields = {"title","content"};  //创建解析器  //可以对搜索的词进行分词，然后进行搜索并对文档的匹配度评分  MultiFieldQueryParser parser =  **new** MultiFieldQueryParser(fields,**new** IKAnalyzer6x());  Query query = parser.parse("特朗普有没有打伊朗和中国");  //4.所搜并返回结果topDocs,topDocs封装了数据的评分、命中条数、所有数据  TopDocs topDocs = searcher.search(query,100);  //打印最大评分与命中条数  System.***out***.println("最大评分："+topDocs.getMaxScore());  System.***out***.println("命中条数："+topDocs.totalHits);  //获取所有的document  ScoreDoc[] scoreDocs = topDocs.scoreDocs;  //遍历scoreDocs  **for** (ScoreDoc scoreDoc : scoreDocs) {  //scoreDoc封装了document以及它的评分、Id等  System.***out***.println("当前document的评分为："+scoreDoc.score);  System.***out***.println("当前document的Id为："+scoreDoc.doc);  //获取document  Document document = searcher.doc(scoreDoc.doc);  //打印document信息  System.***out***.println("title:"+document.get("title"));  System.***out***.println("content:"+document.get("content"));  System.***out***.println("publish:"+document.get("publish"));  System.***out***.println("price:"+document.get("price"));  }  } |

### 11.4.4.布尔查询-BooleanQuery

在多种查询条件同时存在时，各自对应一批查询结果，可以利用布尔查询封装多个查询结果的逻辑关系（MUST\MUST\_NOT\SHOULD\FILTER）

FILTER与MUST意义相同，但结果无评分

|  |
| --- |
| @Test  **public** **void** termQuery() **throws** Exception{  //1.指定存储目录 D:/index01，并封装为dir  Path path = Paths.*get*("D:/index01");  FSDirectory dir = FSDirectory.*open*(path);  //2.构造搜索对象，基于输入流  IndexReader reader = DirectoryReader.*open*(dir);  IndexSearcher searcher = **new** IndexSearcher(reader);  //3.构造词项搜索实现类  //Query query = new TermQuery(new Term("content","中国"));    //3.2构造多域查询的实现类  //准备查询域的数组  /\*String[] fields = {"title","content"};  //创建解析器  //可以对搜索的词进行分词，然后进行搜索并对文档的匹配度评分  MultiFieldQueryParser parser =  new MultiFieldQueryParser(fields,new IKAnalyzer6x());  Query query = parser.parse("特朗普有没有打伊朗和中国");\*/    //3.3.布尔查询  //构造多个查询条件(哪种查询都可以，以词项为例)  Query query1 = **new** TermQuery(**new** Term("content","普"));  Query query2 = **new** TermQuery(**new** Term("title","发动"));  //封装布尔条件  BooleanClause bc1 = **new** BooleanClause(query1,Occur.***FILTER***);  BooleanClause bc2 = **new** BooleanClause(query2,Occur.***MUST\_NOT***);  //创建一个布尔查询器  Query query = **new** BooleanQuery.Builder()  .add(bc1).add(bc2).build();    //4.所搜并返回结果topDocs,topDocs封装了数据的评分、命中条数、所有数据  TopDocs topDocs = searcher.search(query,100);  //打印最大评分与命中条数  System.***out***.println("最大评分："+topDocs.getMaxScore());  System.***out***.println("命中条数："+topDocs.totalHits);  //获取所有的document  ScoreDoc[] scoreDocs = topDocs.scoreDocs;  //遍历scoreDocs  **for** (ScoreDoc scoreDoc : scoreDocs) {  //scoreDoc封装了document以及它的评分、Id等  System.***out***.println("当前document的评分为："+scoreDoc.score);  System.***out***.println("当前document的Id为："+scoreDoc.doc);  //获取document  Document document = searcher.doc(scoreDoc.doc);  //打印document信息  System.***out***.println("title:"+document.get("title"));  System.***out***.println("content:"+document.get("content"));  System.***out***.println("publish:"+document.get("publish"));  System.***out***.println("price:"+document.get("price"));  }  } |

### 11.4.5.范围查询

针对数字类型进行范围查询，可以查询指定范围的document

注意：document必须保持时指定了数字属性：DoublePoint

Query query = DoublePoint.*newRangeQuery*("price",1000, 6000);

### 11.4.6.模糊查询

对某个词的近似值进行计算，然后查询近似值

例如：查statement，写错为：statament

FuzzyQuery query = new FuzzyQuery (new Term("content","普"));

### 11.4.7.通配查询

即通配符“?”

例如查：“特朗?”，会匹配特朗普、特朗通、特朗小、特朗羊

WildcardQuery query = **new** WildcardQuery(**new** Term("content","特朗?"))