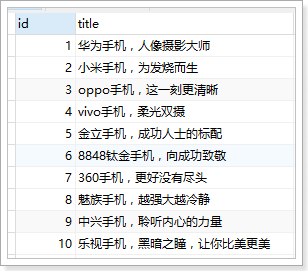
**Lucene**

# 教学笔记

# 了解搜索技术

## 什么是搜索

简单的说，搜索就是搜寻、查找，在IT行业中就是指用户输入关键字，通过相应的算法，查询并返回用户所需要的信息。



线性匹配：

Select \* from 表名 where id=

select \* from item where title like ’%小米%’

结果:

结果中包含： 所有title字段中有 小米 这个词的结果集

是否准确？ 是否高效？

## 新的业务需求

比如，用户在百度文本框中输入，“吃饭睡觉写程序”，会出现的以下结果：



从结果可以看出，百度搜索具备以下明显特点：

1、即使在相关结果数量接近500万时，也能快速得出结果。

2、搜索的结果不仅仅局限于完整的“吃饭睡觉写程序”这一短语，而是将此短语拆分成，“写程序”，“吃饭”，“睡觉”，“程序”等关键字。

3、对拆分后的搜索关键字进行标红显示。

4、......

问题：上述功能，使用大家以前学过的数据库搜索能够方便实现吗？

## 普通的数据库搜索的缺陷

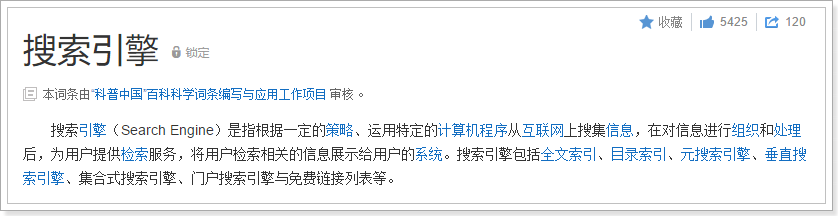
1、因为没有通过高效的索引方式，所以查询的速度在大量数据的情况下是很慢。

2、搜索效果比较差，只能对用户输入的完整关键字首尾位进行模糊匹配。

3、用户搜索时如果多输入一个字符，可能就导致查询出的结果远离用户的预期。

问题： 1.效率问题 不够高效 2.准确度的问题 不够精准

## 搜索引擎



## 搜索引擎的种类

搜索引擎按照功能通常分为垂直搜索和综合搜索。

1、垂直搜索是指专门针对某一类信息进行搜索。例如：会搜网 主要做商务搜索的，并且提供商务信息。除此之外还有爱看图标网、职友集等。

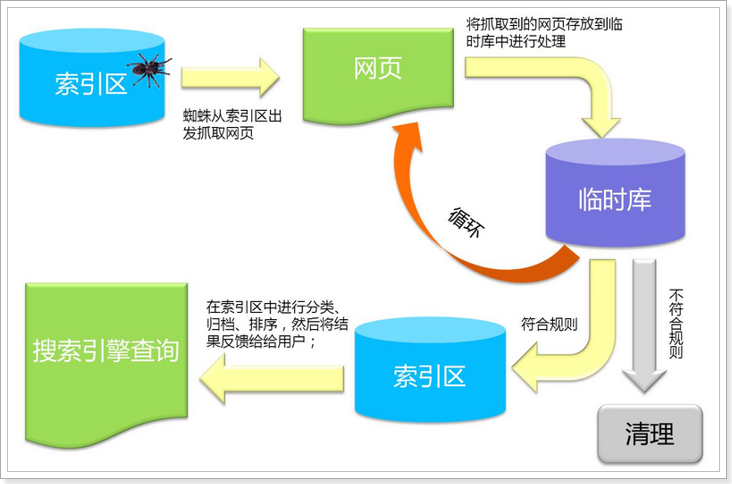
2、综合搜索是指对众多信息进行综合性的搜索。例如：百度、谷歌、搜狗、360搜索等。

3、站内搜索是指对网站内的信息进行的搜索。例如：京东、招聘网站等

4、软件内部搜索，例如word、eclipse等

51job 58同城 拉勾 boss直聘

## 搜索引擎的原理



爬虫—》抓取网页—》临时库—》处理放到索引区—》提供搜索服务

商品表—》处理放到索引区—》提供搜索服务

## 倒排索引技术

倒排索引又叫反向索引（右下图）以字或词为关键字进行索引，表中关键字所对应的记录表项，记录了出现这个字或词的所有文档，每一个表项记录该文档的ID和关键字在该文档中出现的位置情况。



在实际的运用中，我们可以对数据库中原始的数据结构（临时表或者商品表），在业务空闲时事先根据左图内容，创建新的文档列表（左图）及倒排索引区域（右图）。

用户有查询需求时，先访问倒排索引数据区域（右图），得出文档编号后，通过文档文档编号即可快速，准确的通过左图找到具体的文档内容。

这一过程，可以通过我们自己写程序来实现，也可以借用已经抽象出来的通用开源技术来实现。

# Lucene概述

## 什么是Lucene

LOGO：

Lucene Logo

* Lucene是一套用于全文检索和搜寻的开源程序库，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序接口（API），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/>

## 什么是全文检索？

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

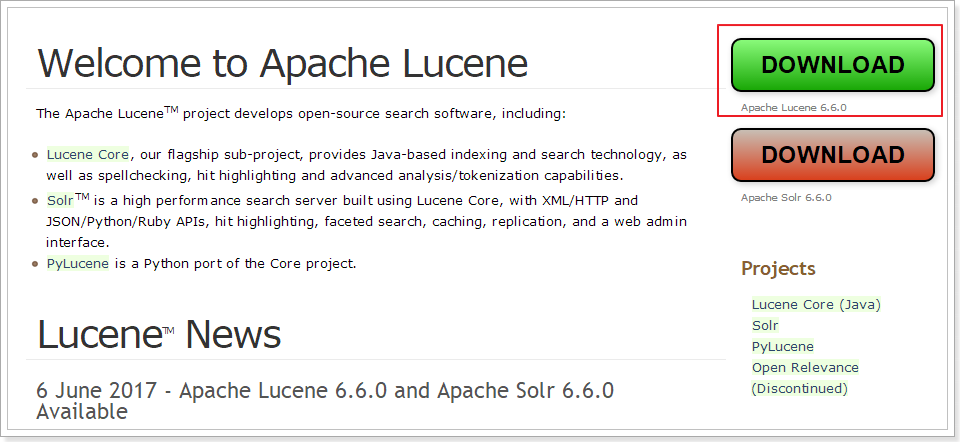
什么是全文检索？ ==> 全部都搜索 ==> 如何实现全部都搜索？ ==> 分词 单词永不重复 单词就是索引 单词是最小的搜索单位

倒排索引

总结：对文档（数据）中每一个词都做索引。

## Lucene下载及版本问题

官网：



* 目前最新的版本是6.x系列，但是大多数企业中依旧使用4.x版本，比较稳定。本次课程我们使用4.10.2版本

## Lucene与Solr的关系

Lucene：底层的API，工具包

Solr：基于Lucene开发的企业级的搜索引擎产品

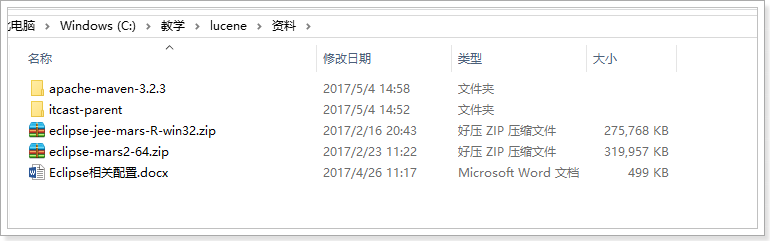
# Lucene的基本使用

使用Lucene的API来实现对索引的增（创建索引）、删（删除索引）、改（修改索引）、查（搜索数据）。

## 准备工作

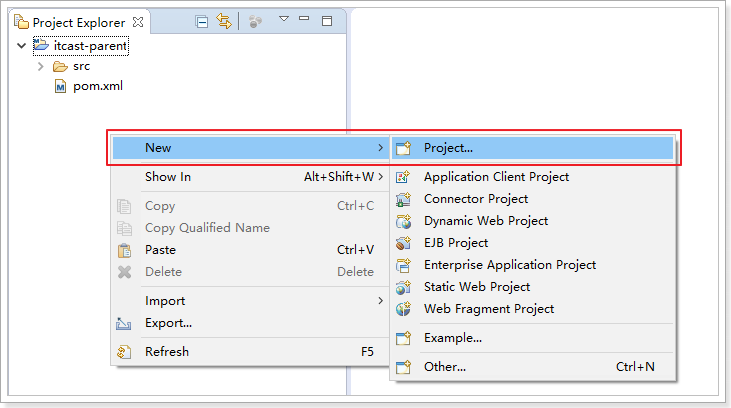
### 工具和环境（截图）

Maven以及eclipse的配置参考《Eclipse相关配置.docx》

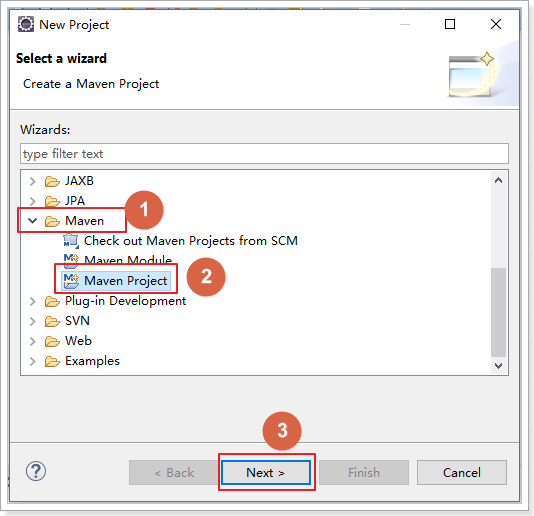


### 创建lucene工程

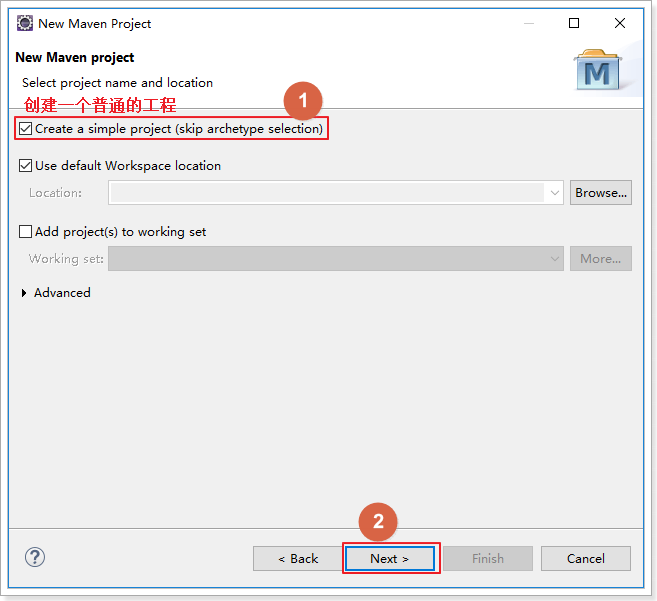
1. 右键🡪New🡪Project…



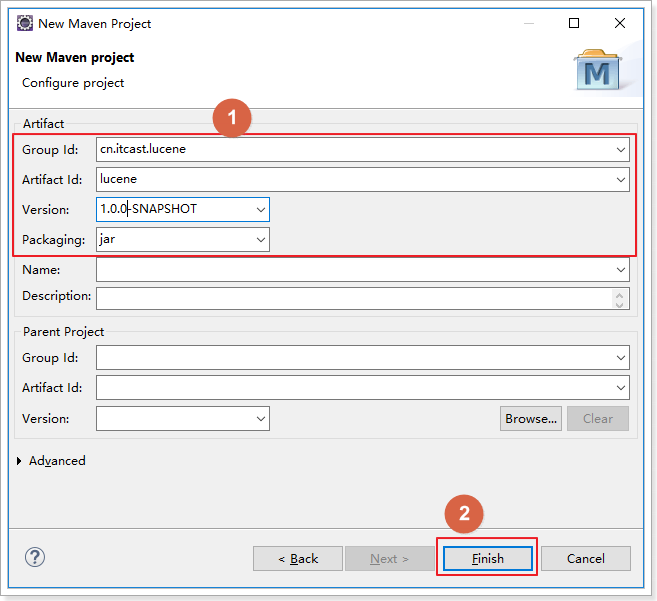
1. 打开New Project窗口，点击maven🡪Maven Project🡪Next>



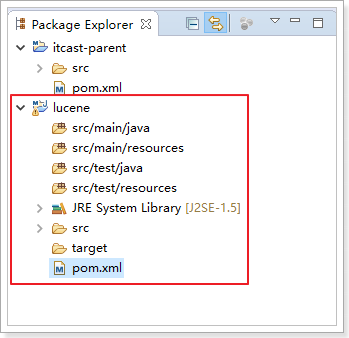
1. 打开New Maven Poject窗口，选择“Create a simple project（skip archetype selection）”🡪Next>



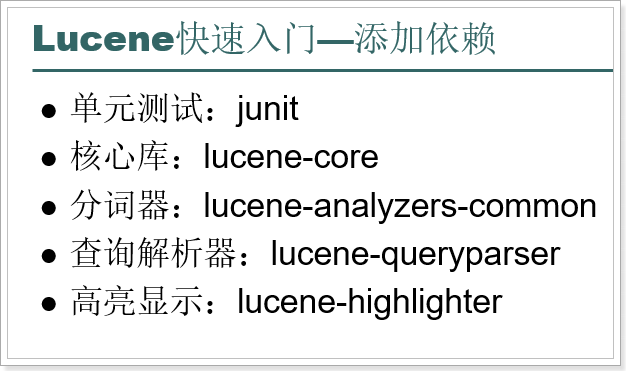
1. 打开New Maven Project窗口，填写相应的坐标信息如下：



1. 创建完成后



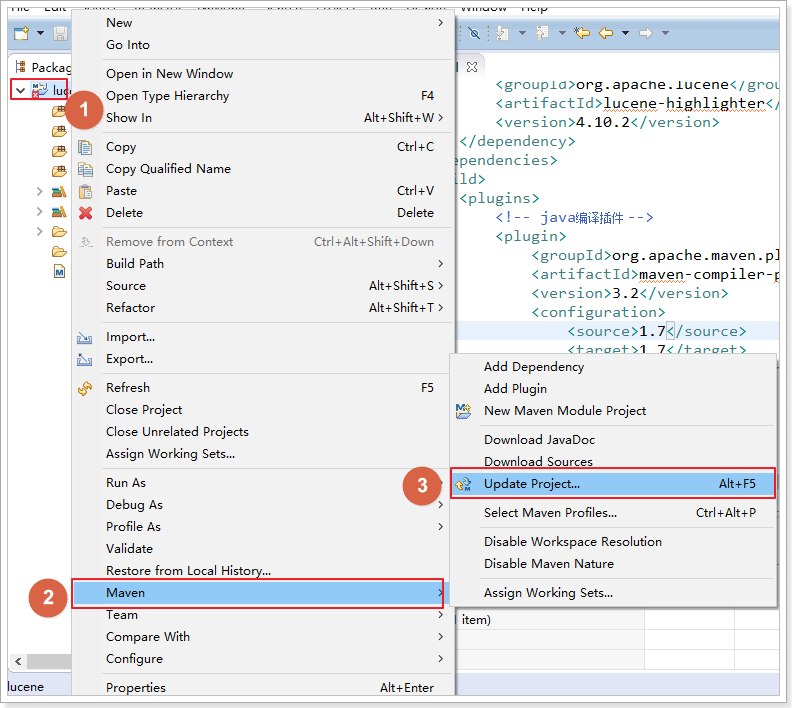
### 引入依赖



|  |
| --- |
| <dependencies>  <!-- Junit单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>  <!-- lucene核心库 -->  <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-core</artifactId>  <version>4.10.2</version>  </dependency>  <!-- Lucene的查询解析器 -->  <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-queryparser</artifactId>  <version>4.10.2</version>  </dependency>  <!-- lucene的默认分词器库 -->  <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-analyzers-common</artifactId>  <version>4.10.2</version>  </dependency>  <!-- lucene的高亮显示 -->  <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-highlighter</artifactId>  <version>4.10.2</version>  </dependency>  </dependencies>  <build>  <plugins>  <!-- java编译插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.2</version>  <configuration>  <source>1.7</source>  <target>1.7</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build> |

### 解决工程小红叉

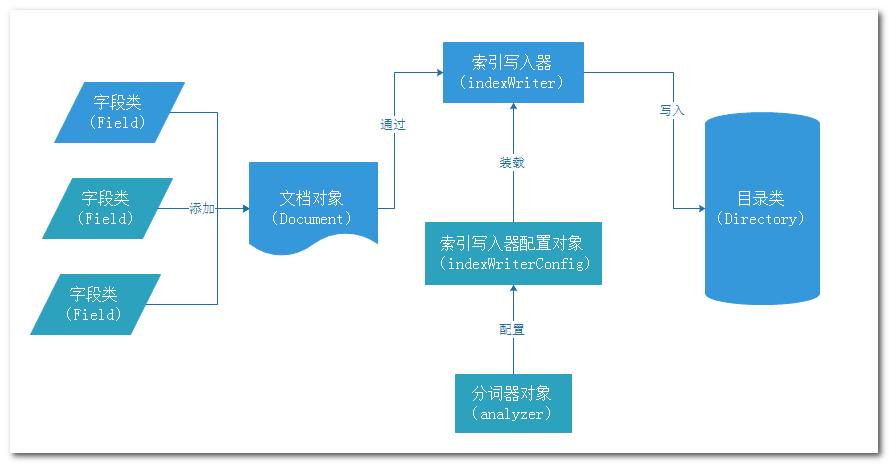
右键工程🡪Maven🡪Update Project…，即可解决。



## 创建索引

### 创建索引的流程：

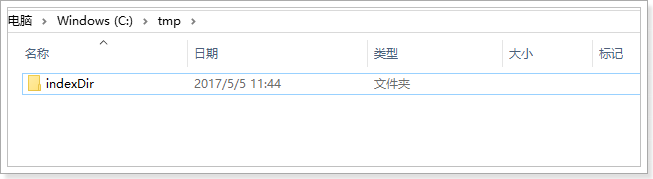




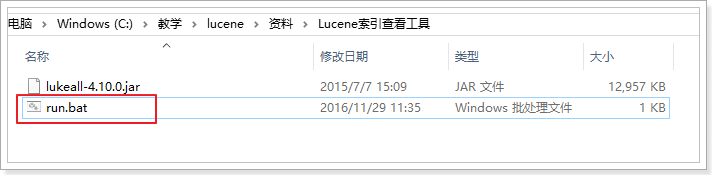
### 代码实现

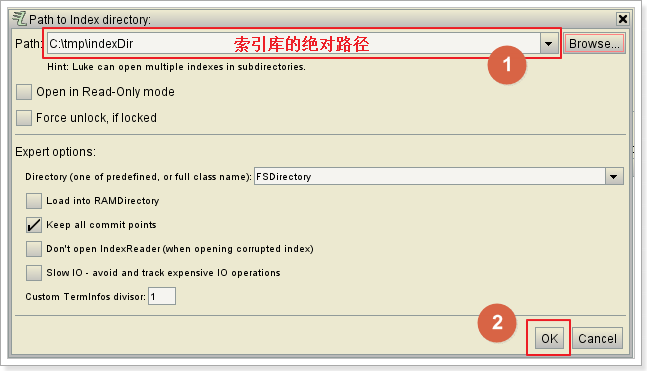
|  |
| --- |
| @Test  **public** **void** indexCreate() **throws** IOException {  // 创建文档对象  Document document = **new** Document();  // 添加字段，参数Field是一个接口，要new实现类的对象(StringField, TextField)  // StringField的实例化需要3个参数：1-字段名，2-字段值，3-是否保存到文档，Store.YES存储，NO不存储  document.add(**new** StringField("id", "1", Store.***YES***));  // TextField：创建索引并提供分词，StringField创建索引但不分词  document.add(**new** TextField("title", "谷歌地图之父跳槽FaceBook", Store.***YES***));    // 创建目录对象，指定索引库的存放位置；FSDirectory文件系统；RAMDirectory内存  Directory directory = FSDirectory.*open*(**new** File("C:\\tmp\\indexDir"));  // 创建分词器对象  Analyzer analyzer = **new** StandardAnalyzer();  // 创建索引写入器配置对象，第一个参数版本VerSion.LATEST,第一个参数分词器  IndexWriterConfig conf = **new** IndexWriterConfig(Version.***LATEST***, analyzer);  // 创建索引写入器  IndexWriter indexWriter = **new** IndexWriter(directory , conf);  // 向索引库写入文档对象  indexWriter.addDocument(document);  // 提交  indexWriter.commit();  // 关闭  indexWriter.close();  } |

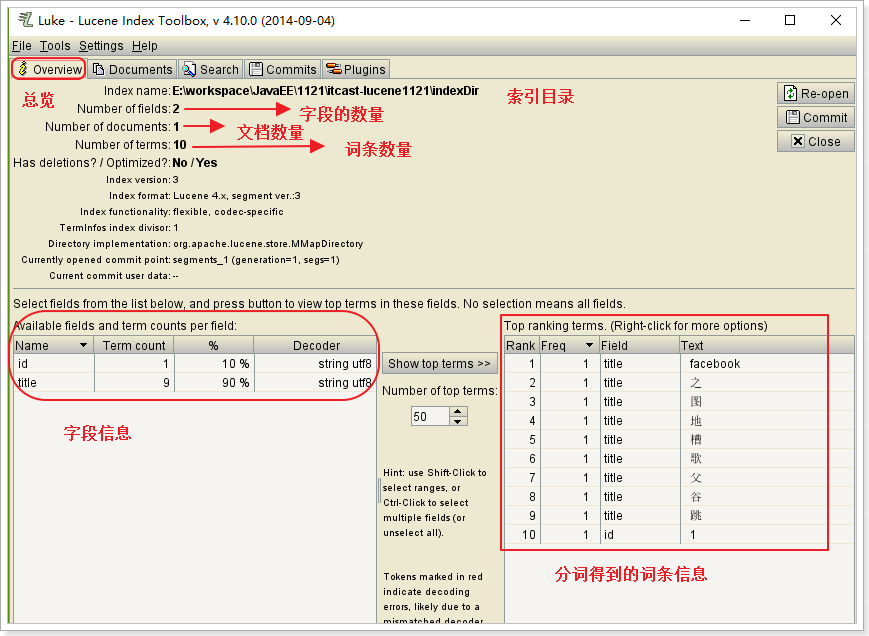
### 运行测试

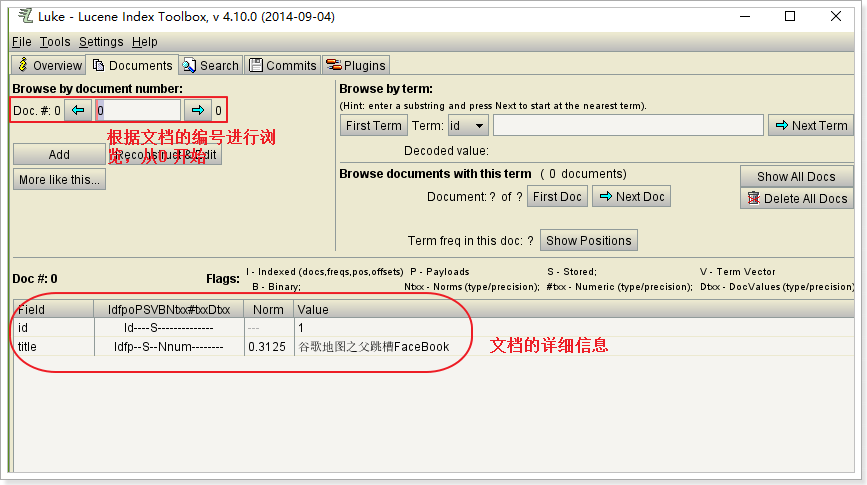


### 索引查看工具









### API详解

#### Document（文档类）

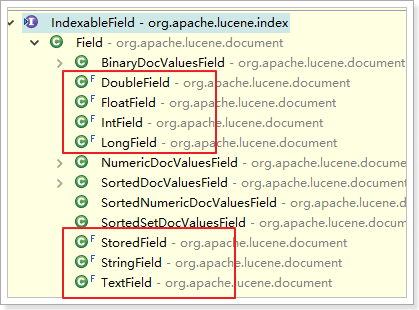
Document：文档对象，是一条原始的数据



#### Field（字段类）

一个Document中可以有很多个不同的字段，每一个字段都是一个Field类的对象。

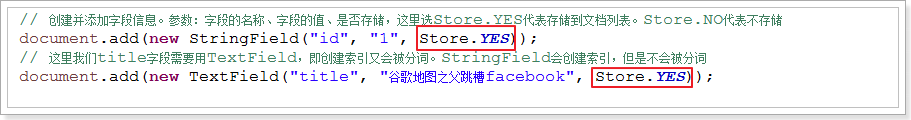
一个Document中的字段其类型是不确定的，因此Field类就提供了各种不同的子类，来对应这些不同类型的字段。



这些子类有一些不同的特性：

1. 创建索引

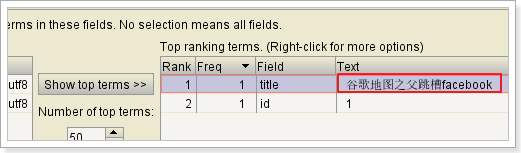
DoubleField、FloatField、IntField、LongField、StringField、TextField这些子类一定会被创建索引。但是不一定会被存储到文档列表。要通过构造函数中的参数Store来指定：如果Store.YES代表存储，Store.NO代表不存储



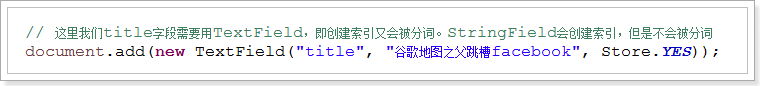
2）创建索引，又会分词

TextField即创建索引，又会被分词。StringField等会创建索引，但是不会被分词。

如果不分词，会造成整个字段作为一个词条，除非用户完全匹配，否则搜索不到：



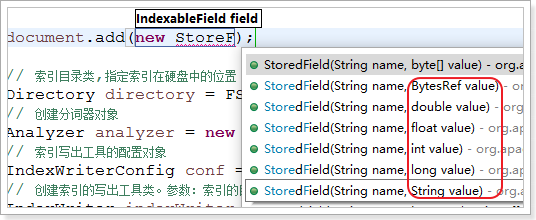
我们一般，需要搜索的字段，都会做分词：

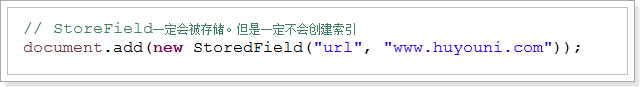


3）是否存储

StoreField一定会被存储，但是一定不创建索引

StoredField可以创建各种数据类型的字段：





问题1：如何确定一个字段是否需要存储？

如果一个字段要显示到最终的结果中，那么一定要存储，否则就不存储

问题2：如何确定一个字段是否需要创建索引？

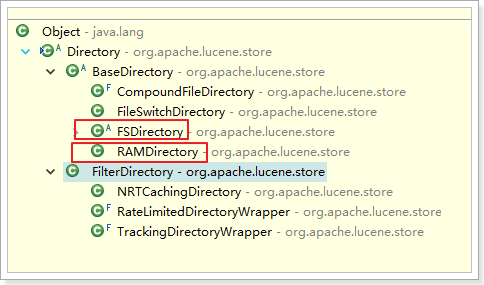
如果要根据这个字段进行搜索，那么这个字段就必须创建索引。

问题3：如何确定一个字段是否需要分词？

前提是这个字段首先要创建索引。然后如果这个字段的值是不可分割的，那么就不需要分词。例如：ID

#### Directory（目录类）

指定索引要存储的位置



FSDirectory：文件系统目录，会把索引库保存到本地磁盘。

特点：速度略慢，但是比较安全

RAMDirectory：内存目录，会把索引库保存在内存。

特点：速度快，但是不安全

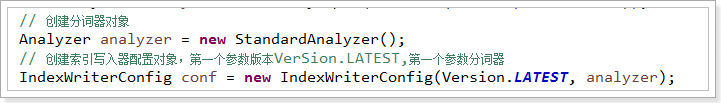
noSQL：not only sql

Mongodb

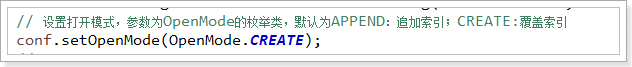
Redis

#### IndexWriterConfig（索引写入器配置类）

1. 设置配置信息：Lucene的版本和分词器类型

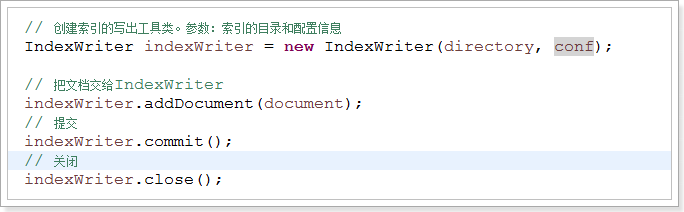


2）设置OpenMode参数



#### IndexWriter（索引写入器类）

* 索引写出工具，作用就是 实现对索引的增（创建索引）、删（删除索引）、改（修改索引）

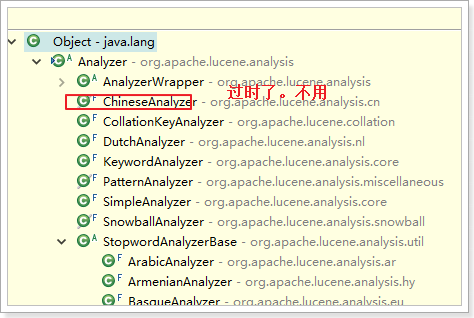


* 可以一次创建一个，也可以批量创建索引

|  |
| --- |
| @Test  **public** **void** testCreater2() **throws** IOException{    // 创建文档对象集合  List<Document> docs = **new** ArrayList<>();    // 创建文档对象  Document document1 = **new** Document();  document1.add(**new** StringField("id", "1", Store.***YES***));  document1.add(**new** TextField("title", "谷歌地图之父跳槽FaceBook", Store.***YES***));  docs.add(document1);  // 创建文档对象  Document document2 = **new** Document();  document2.add(**new** StringField("id", "2", Store.***YES***));  document2.add(**new** TextField("title", "谷歌地图之父加盟FaceBook", Store.***YES***));  docs.add(document2);  // 创建文档对象  Document document3 = **new** Document();  document3.add(**new** StringField("id", "3", Store.***YES***));  document3.add(**new** TextField("title", "谷歌地图创始人拉斯离开谷歌加盟Facebook", Store.***YES***));  docs.add(document3);  // 创建文档对象  Document document4 = **new** Document();  document4.add(**new** StringField("id", "4", Store.***YES***));  document4.add(**new** TextField("title", "谷歌地图之父跳槽Facebook与Wave项目取消有关", Store.***YES***));  docs.add(document4);  // 创建文档对象  Document document5 = **new** Document();  document5.add(**new** StringField("id", "5", Store.***YES***));  document5.add(**new** TextField("title", "谷歌地图之父拉斯加盟社交网站Facebook", Store.***YES***));  docs.add(document5);    // 索引库对象  Directory directory = FSDirectory.*open*(**new** File("C:\\tmp\\indexDir"));  // 创建索引写入器配置对象，1-版本，2-分词器：标准分词器  IndexWriterConfig conf = **new** IndexWriterConfig(Version.***LATEST***, **new** StandardAnalyzer());  // OpenMode打开模式，枚举类，CREATE：覆盖，APPEND：追加，  conf.setOpenMode(OpenMode.***CREATE***);  // 创建索引写入器对象  IndexWriter indexWriter = **new** IndexWriter(directory, conf);    // 执行写入操作  indexWriter.addDocuments(docs);  // 提交  indexWriter.commit();  // 关闭  indexWriter.close();  } |

### Analyzer分词器

提供分词算法，可以把文档中的数据按照算法分词



这些分词器，并没有合适的中文分词器，因此一般我们会用第三方提供的分词器：



一般我们用IK分词器。

#### IK分词器

官网：<https://code.google.com/p/ik-analyzer/>

* 概述



林良益

IK分词器官方版本是不支持Lucene4.X的，有人基于IK的源码做了改造，支持了Lucene4.X：

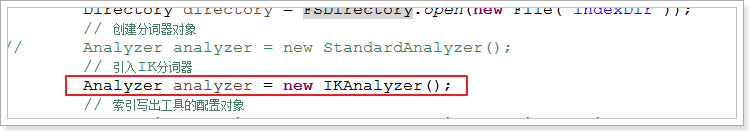


#### 基本使用

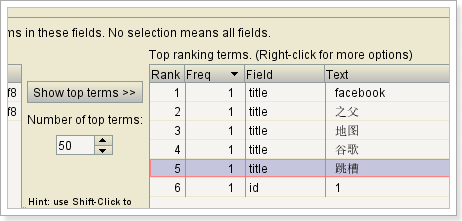
引入IK分词器的依赖：

|  |
| --- |
| <!-- 引入IK分词器 -->  <dependency>  <groupId>com.janeluo</groupId>  <artifactId>ikanalyzer</artifactId>  <version>2012\_u6</version>  </dependency> |

使用IK分词器，改造程序：



中文分词更专业：

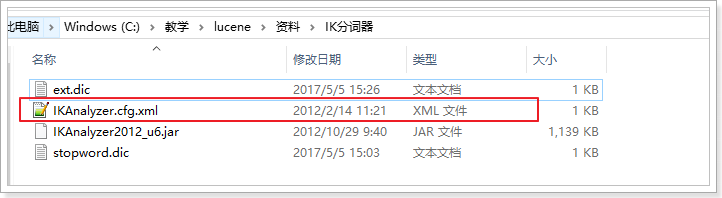


#### 自定义词库

扩展词典（新创建词功能）： 有些词IK分词器不识别 例如：“传智播客”，“碉堡了”

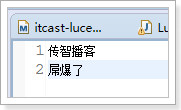
停用词典（停用某些词功能）： 有些词不需要建立索引 例如：“哦”，“啊”，“的”

IK分词器的词库有限，新增加的词条可以通过配置文件添加到IK的词库中，也可以把一些不用的词条去除：

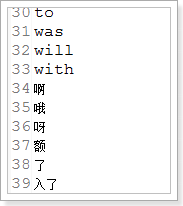




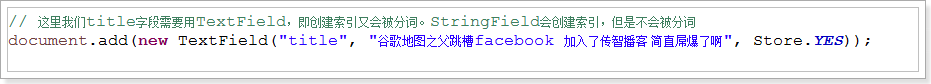
扩展词典：用来引入一些自定义的新词



停止词典：用来停用一些不必要的词条



结果：分词中，加入了我们新的词，被停用的词语没有被分词：





## 查询索引数据

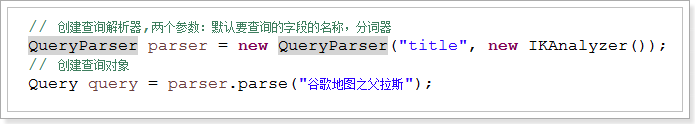
### 代码实现

|  |
| --- |
| @Test  **public** **void** testSearcher() **throws** IOException, ParseException{    // 初始化索引库对象  Directory directory = FSDirectory.*open*(**new** File("C:\\tmp\\index"));    // 索引读取工具  IndexReader indexReader = DirectoryReader.*open*(directory);  // 索引搜索对象  IndexSearcher indexSearcher = **new** IndexSearcher(indexReader);    // 创建查询解析器对象  QueryParser parser = **new** QueryParser("title", **new** IKAnalyzer());  // 创建查询对象  Query query = parser.parse("谷歌");  // 执行搜索操作，返回值topDocs包含命中数，得分文档  TopDocs topDocs = indexSearcher.search(query, Integer.***MAX\_VALUE***);  // 打印命中数  System.***out***.println("一共命中："+topDocs.totalHits+"条数据");  // 获得得分文档数组对象，得分文档对象包含得分和文档编号  ScoreDoc[] scoreDocs = topDocs.scoreDocs;  **for** (ScoreDoc scoreDoc : scoreDocs) {  System.***out***.println("得分："+scoreDoc.score);  // 文档的编号  **int** doc = scoreDoc.doc;  System.***out***.println("编号："+doc);  // 获取文档对象，通过索引读取工具  Document document = indexReader.document(doc);  System.***out***.println("id:"+document.get("id"));  System.***out***.println("title:"+document.get("title"));  }  } |

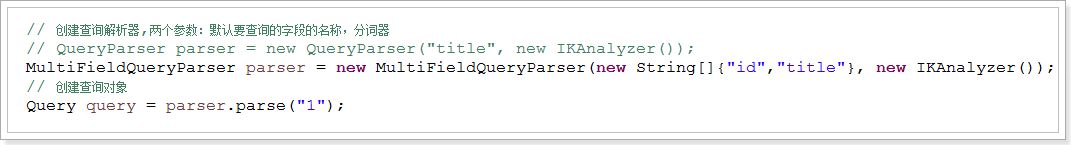
### 核心API

#### QueryParser（查询解析器）

1）QueryParser（单一字段的查询解析器）

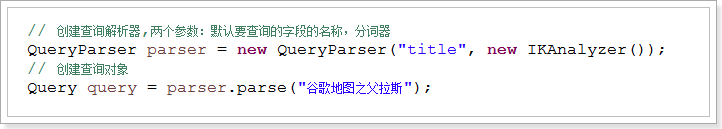


2）MultiFieldQueryParser（多字段的查询解析器）



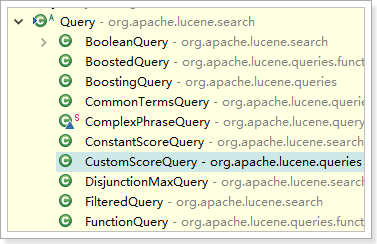
#### Query（查询对象，包含要查询的关键词信息）

* 1）通过QueryParser解析关键字，得到查询对象



* 2）自定义查询对象（特殊查询）

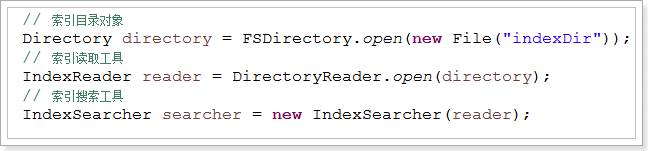
我们可以通过Query的子类，直接创建查询对象，实现高级查询（后面详细讲）



#### IndexSearcher（索引搜索对象，执行搜索功能）

IndexSearcher可以帮助我们实现：快速搜索、排序等功能。

IndexSearcher需要依赖IndexReader类



查询后得到的结果，就是打分排序后的前N名结果。N可以通过第2个参数来指定：



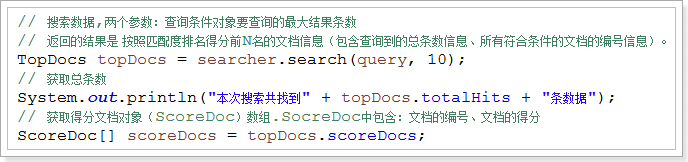
#### TopDocs（查询结果对象）

通过IndexSearcher对象，我们可以搜索，获取结果：TopDocs对象

在TopDocs中，包含两部分信息：

int totalHits ：查询到的总条数

ScoreDoc[] scoreDocs ： 得分文档对象的数组



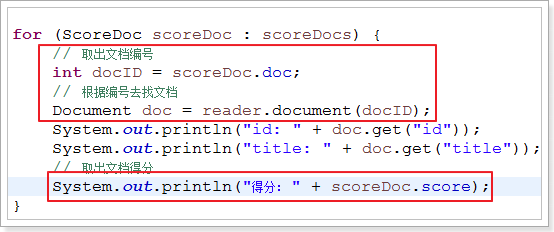
#### ScoreDoc(得分文档对象)

ScoreDoc是得分文档对象，包含两部分数据：

int doc ：文档的编号

float score ：文档的得分信息

拿到编号后，我们还需要根据编号来获取真正的文档信息



### 特殊查询

抽取公用的搜索方法：

|  |
| --- |
| **public** **void** search(Query query) **throws** Exception {  // 创建目录对象  Directory directory = FSDirectory.*open*(**new** File("C:\\tmp\\indexDir"));  // 索引的读取对象  IndexReader indexReader = DirectoryReader.*open*(directory);  // 索引的搜索工具  IndexSearcher searcher = **new** IndexSearcher(indexReader);  // 尝试查询，1-查询对象，2-查询的条数  // 返回的是前n条文档的对象，topDocs：包含文档的总条数，文档的得分数组  TopDocs topDocs = searcher.search(query, 10);  System.***out***.println("搜索的命中总条数：" + topDocs.totalHits);  // 获取得分文档的数组，得分文档包含文档编号以及得分  ScoreDoc[] scoreDocs = topDocs.scoreDocs;  **for** (ScoreDoc scoreDoc : scoreDocs) {  System.***out***.println("文档编号" + scoreDoc.doc);  System.***out***.println("文档得分" + scoreDoc.score);  // 根据编号查询文档  Document document = indexReader.document(scoreDoc.doc);  System.***out***.println(document.get("id"));  System.***out***.println(document.get("title"));  }  } |

#### TermQuery（词条查询）

|  |
| --- |
| /\*\*  \* 词条查询  \* 查询条件必须是最小粒度不可再分割的内容  \* 场景：不可分割的字段可以采用，比如id  \* 缺点：只能查询一个词，例如可以查询"谷歌"，不能查询"谷歌地图"  \* **@throws** IOException  \* **@throws** ParseException  \*/  @Test  **public** **void** testTermSearcher() **throws** IOException, ParseException{    // 创建查询对象  Query query = **new** TermQuery(**new** Term("title", "谷歌"));  // 执行搜索操作  searcher(query);  } |

#### WildcardQuery（通配符查询）

|  |
| --- |
| @Test  **public** **void** testWildCardQuery() **throws** Exception {  // 查询条件对象（通配符  // ?：通配一个字符  // \*：通配多个字符  Query query = **new** WildcardQuery(**new** Term("title", "\*歌\*"));    search(query);  } |

#### FuzzyQuery（模糊查询）

|  |
| --- |
| @Test  **public** **void** testFuzzyQuery() **throws** Exception {  // 查询条件对象（模糊查询  // 参数：1-词条，查询字段及关键词，关键词允许写错；2-允许写错的最大编辑距离，并且不能大于2（0~2）  // 最大编辑距离：facebool-->facebook需要编辑的次数，包括大小写  Query query = **new** FuzzyQuery(**new** Term("title", "facebook"), 1);    search(query);  } |

#### NumericRangeQuery（数值范围查询）

|  |
| --- |
| @Test  **public** **void** testNumericRangeQuery() **throws** Exception {  // 查询条件对象（数值范围查询  // 查询非String类型的数据或者说是一些继承Numeric类的对象的查询  // 1-字段；2-最小值；3-最大值；4-是否包含最小值；5-是否包含最大值  Query query = NumericRangeQuery.*newLongRange*("id", 2l, 4l, **true**, **true**);    search(query);  } |

#### BooleanQuery（组合查询）

|  |
| --- |
| @Test  **public** **void** testBooleanQuery() **throws** Exception {    Query query1 = NumericRangeQuery.*newLongRange*("id", 2l, 4l, **true**, **true**);  Query query2 = NumericRangeQuery.*newLongRange*("id", 0l, 3l, **true**, **true**);    // boolean查询本身没有查询条件，它可以组合其他查询  BooleanQuery query = **new** BooleanQuery();  // 交集： Occur.MUST + Occur.MUST  // 并集：Occur.SHOULD + Occur.SHOULD  // 非：Occur.MUST\_NOT  query.add(query1, Occur.***SHOULD***);  query.add(query2, Occur.***SHOULD***);    search(query);  } |

## 修改索引

|  |
| --- |
| /\*\*  \* 更新索引  \* 本质先删除再添加  \* 先删除所有满足条件的文档，再创建文档  \* 因此，更新索引通常要根据唯一字段  \* **@throws** IOException  \*/  @Test  **public** **void** testUpdate() **throws** IOException{    // 创建文档对象  Document document = **new** Document();  document.add(**new** StringField("id", "9", Store.***YES***));  document.add(**new** TextField("title", "谷歌地图之父跳槽FaceBook", Store.***YES***));    // 索引库对象  Directory directory = FSDirectory.*open*(**new** File("C:\\tmp\\index"));  // 索引写入器配置对象  IndexWriterConfig conf = **new** IndexWriterConfig(Version.***LATEST***, **new** IKAnalyzer());  // 索引写入器对象  IndexWriter indexWriter = **new** IndexWriter(directory, conf);    // 执行更新操作  indexWriter.updateDocument(**new** Term("id", "1"), document);  // 提交  indexWriter.commit();  // 关闭  indexWriter.close();    } |

## 删除索引

|  |
| --- |
| @Test  **public** **void** testDelete() **throws** IOException {  // 创建目录对象  Directory directory = FSDirectory.*open*(**new** File("C:\\tmp\\indexDir"));  // 创建索引写入器配置对象  IndexWriterConfig conf = **new** IndexWriterConfig(Version.***LATEST***, **new** IKAnalyzer());  // 创建索引写入器对象  IndexWriter indexWriter = **new** IndexWriter(directory, conf);  // 执行删除操作(根据词条)，要求id字段必须是字符串类型  // indexWriter.deleteDocuments(new Term("id", "5"));  // 根据查询条件删除  // indexWriter.deleteDocuments(NumericRangeQuery.newLongRange("id", 2l, 4l, true, false));  // 删除所有  indexWriter.deleteAll();  // 提交  indexWriter.commit();  // 关闭  indexWriter.close();  } |

# Lucene的高级使用

## 高亮显示

原理：

1）给所有关键字加上一个HTML标签



2）给这个特殊的标签设置CSS样式



代码实现：

// 高亮显示

@Test

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

// 目录对象

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

// 创建读取工具

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

// 创建搜索工具

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

QueryParser parser = **new** QueryParser("title", **new** IKAnalyzer());

Query query = parser.parse("谷歌地图");

// 格式化器

Formatter formatter = **new** SimpleHTMLFormatter("<em>", "</em>");

Scorer scorer = **new** QueryScorer(query);

// 准备高亮工具

Highlighter highlighter = **new** Highlighter(formatter, scorer);

// 搜索

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

System.***out***.println("本次搜索共" + topDocs.totalHits + "条数据");

ScoreDoc[] scoreDocs = topDocs.scoreDocs;

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

// 获取文档编号

**int** docID = scoreDoc.doc;

Document doc = reader.document(docID);

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

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

// 用高亮工具处理普通的查询结果,参数：分词器，要高亮的字段的名称，高亮字段的原始值

String hTitle = highlighter.getBestFragment(**new** IKAnalyzer(), "title", title);

System.***out***.println("title: " + hTitle);

// 获取文档的得分

System.***out***.println("得分：" + scoreDoc.score);

}

}

## 排序

// 排序

@Test

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

// 目录对象

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

// 创建读取工具

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

// 创建搜索工具

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

QueryParser parser = **new** QueryParser("title", **new** IKAnalyzer());

Query query = parser.parse("谷歌地图");

// 创建排序对象,需要排序字段SortField，参数：字段的名称、字段的类型、是否反转如果是false，升序。true降序

Sort sort = **new** Sort(**new** SortField("id", Type.***LONG***, **true**));

// 搜索

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

System.***out***.println("本次搜索共" + topDocs.totalHits + "条数据");

ScoreDoc[] scoreDocs = topDocs.scoreDocs;

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

// 获取文档编号

**int** docID = scoreDoc.doc;

Document doc = reader.document(docID);

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

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

}

}

## 分页

物理分页：limit

逻辑分页：代码

// 分页

@Test

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

// 实际上Lucene本身不支持分页。因此我们需要自己进行逻辑分页。我们要准备分页参数：

**int** pageSize = 2;// 每页条数

**int** pageNum = 3;// 当前页码

**int** start = (pageNum - 1) \* pageSize;// 当前页的起始条数

**int** end = start + pageSize;// 当前页的结束条数（不能包含）

// 目录对象

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

// 创建读取工具

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

// 创建搜索工具

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

QueryParser parser = **new** QueryParser("title", **new** IKAnalyzer());

Query query = parser.parse("谷歌地图");

// 创建排序对象,需要排序字段SortField，参数：字段的名称、字段的类型、是否反转如果是false，升序。true降序

Sort sort = **new** Sort(**new** SortField("id", Type.***LONG***, **false**));

// 搜索数据，查询0~end条

TopDocs topDocs = searcher.search(query, end,sort);

System.***out***.println("本次搜索共" + topDocs.totalHits + "条数据");

ScoreDoc[] scoreDocs = topDocs.scoreDocs;

**for** (**int** i = start; i < end; i++) {

ScoreDoc scoreDoc = scoreDocs[i];

// 获取文档编号

**int** docID = scoreDoc.doc;

Document doc = reader.document(docID);

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

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

}

}

## 得分算法

* Lucene会对搜索结果打分，用来表示文档数据与词条关联性的强弱，得分越高，表示查询的匹配度就越高，排名就越靠前！其算法公式是：

