lucene&solr

**第二天**

# 学习目标

* 能够实现索引库操作
* 能够描述Lucene的Field域
* 能够应用LuceneAPI查询索引库
* 能够领会solr基本理论知识
* 能够安装部署solr
* 能够使用solr的java客户端solrJ对索引库操作

# lucene的Field

## Field的特性

Document（文档）是Field（域）的承载体，一个Document是由多个Field组成。Field由名称和值两部分组成，Field的值是要索引的内容，也是要搜索的内容。

* 是否分词（tokenized）

是：将Field的值进行分词处理，分词的目的是为了索引。

比如：商品名称，商品描述。这些内容用户需要输入关键词进行查询，由于内容格式大，内容多，需要进行分词处理建立索引。

否：不做分词处理

比如：订单编号，身份证号。是一个整体，分词以后没有意义，不需要分词。

* 是否索引（indexed）

是：将Field内容进行分词处理后得到的词或者整体Field内容建立索引，存储到索引域。索引的目的是为了搜索。

比如：商品名称，商品描述需要分词建立索引。订单编号，身份证号作为整体建立索引。只要将来要作为用户查询条件的词，都需要索引。

否：不索引。

比如：商品图片路径，不作为查询条件，不需要建立索引。

* 是否存储（stored）

是：将Field值保存到Document中。

比如：商品名称，商品价格。凡是将来在搜索结果页面展现给用户的内容，都需要存储。

否：不存储。

比如：商品描述。内容多格式大，不需要直接在搜索结果页面展现，不做存储。需要的时候可以从关系数据库取。

## 常用的Field种类

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field种类 | 数据  类型 | 是否  分词 | 是否  索引 | 是否  存储 | 说明 |
| StringField(FieldName,  FieldValue,Store.YES) | 字符串 | N | Y | Y或N | 字符串类型Field，不分词，作为一个整体进行索引（比如：身份证号，订单编号），是否需要存储根据Store.YES或Store.NO决定 |
| LongField(FieldName,  FieldValue,Store.YES) | 数值型代表 | Y | Y | Y或N | Long数值型Field代表，分词并且索引（比如：价格），是否需要存储根据Store.YES或Store.NO决定 |
| StoredField(FieldName,  FieldValue) | 重载方法，支持多种类型 | N | N | Y | 构建不同类型的Field，不分词，不索引，存储。  （比如：商品图片路径） |
| TextField(FieldName,  FieldValue,Store.NO) | 文本类型 | Y | Y | Y或N | 文本类型Field，分词并且索引，是否需要存储根据Store.YES或Store.NO决定 |

## 常用Field种类使用

### 需求分析

* 图书Id

是否分词：不需要分词

是否索引：需要索引

是否存储：需要存储

--StringField

* 图书名称

是否分词：需要分词

是否索引：需要索引

是否存储：需要存储

--TextField

* 图书价格

是否分词：（数值型的Field lucene使用内部的分词）

是否索引：需要索引

是否存储：需要存储

--DoubleField

* 图书图片

是否分词：不需要分词

是否索引：不需要索引

是否存储：需要存储

--StoredField

* 图书描述

是否分词：需要分词

是否索引：需要索引

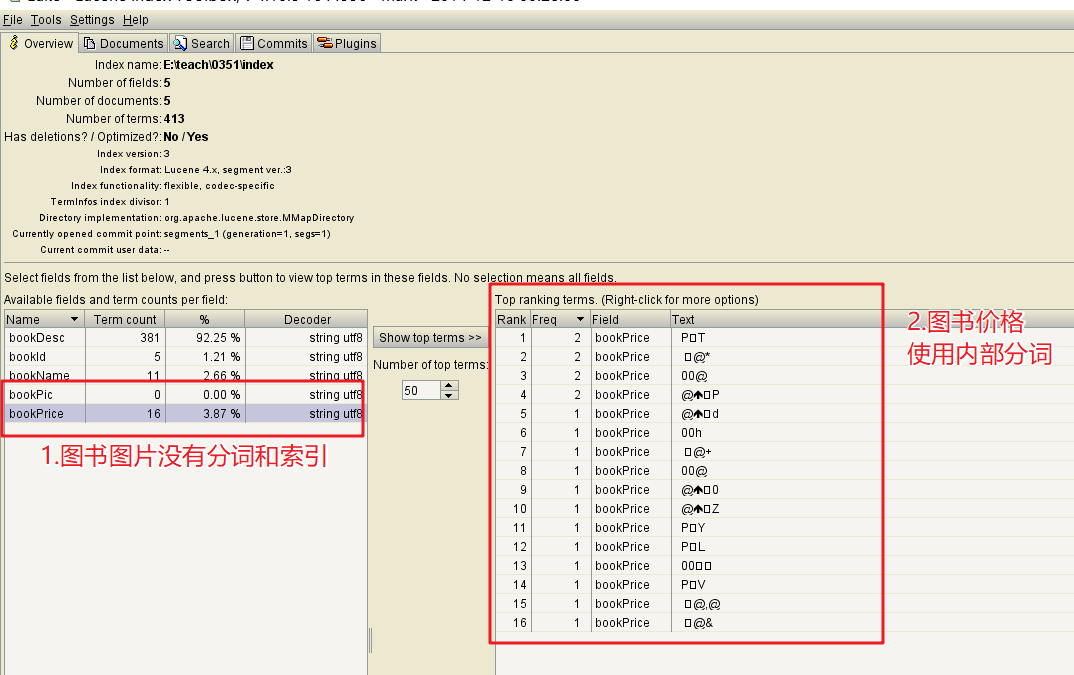
是否存储：不需要存储

--TextField

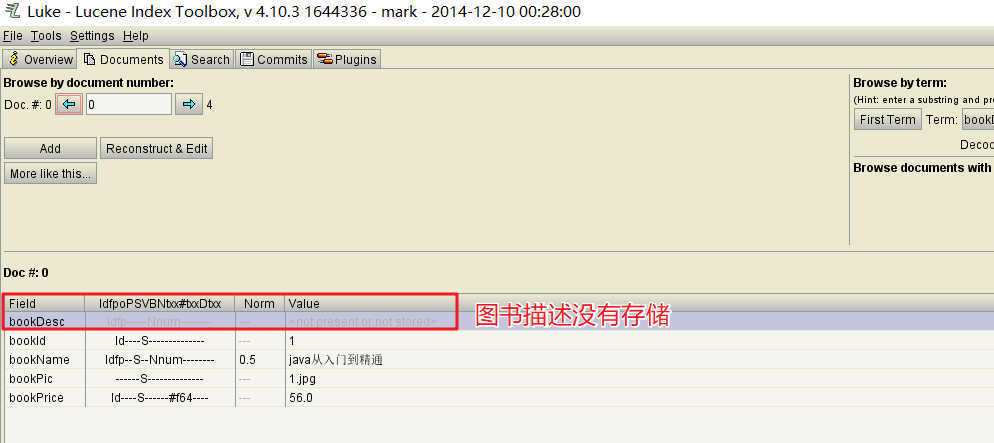
### 修改代码

|  |
| --- |
| */\*\*  \* 索引流程实现  \*/* @Test  **public void** createIndex() **throws** Exception{ *// 1.采集数据* BookDao bookDao = **new** BookDaoImpl();  List<Book> bookList = bookDao.findAllBooks();  *// 2.建立文档对象（Document）* List<Document> documentList = **new** ArrayList<Document>();  **for**(Book book:bookList){  *//创建文档对象* Document doc = **new** Document();   */\*\*  \* 给文档对象添加域  \* 方法：add（）  \* 参数：TextField  \* TextField参数：  \* 参数一：域的名称  \* 参数二：域值  \* 参数三：指定是否把域值保存到文档对象中  \*/  // 图书Id  /\*\*  \* 图书Id  是否分词：不需要分词  是否索引：需要索引  是否存储：需要存储   --StringField  \*/* doc.add(**new** StringField(**"bookId"**,book.getId()+**""**, Field.Store.***YES***));   *// 图书名称  /\*\*  \* 图书名称  是否分词：需要分词  是否索引：需要索引  是否存储：需要存储   --TextField  \*/* doc.add(**new** TextField(**"bookName"**,book.getBookName(), Field.Store.***YES***));  *// 图书价格  /\*\*  \* 图书价格  是否分词：（数值型的Field lucene使用内部的分词）  是否索引：需要索引  是否存储：需要存储   --DoubleField  \*/* doc.add(**new** DoubleField(**"bookPrice"**,book.getPrice(), Field.Store.***YES***));  *// 图书图片  /\*\*  \* 图书图片  是否分词：不需要分词  是否索引：不需要索引  是否存储：需要存储   --StoredField  \*/* doc.add(**new** StoredField(**"bookPic"**,book.getPic()));  *// 图书描述  /\*\*  \* 图书描述  是否分词：需要分词  是否索引：需要索引  是否存储：不需要存储   --TextField  \*/* doc.add(**new** TextField(**"bookDesc"**,book.getBookDesc(), Field.Store.***NO***));   documentList.add(doc);  }  *// 3.建立分析器（Analyzer），用于分词  //Analyzer analyzer = new StandardAnalyzer();  // 使用ik分词器* Analyzer analyzer = **new** IKAnalyzer();  *// 4.建立索引库配置对象（IndexWriterConfig），配置索引库  /\*\*  \* IndexWriterConfig参数：  \* 参数一：指定当前使用的lucene版本信息  \* 参数二：分析器对象  \*/* IndexWriterConfig iwc = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***,analyzer);  *// 5.建立索引库目录对象（Directory），指定索引库的位置* File file = **new** File(**"E:\\teach\\0351\\index\\"**);  Directory directory = FSDirectory.*open*(file);  *// 6.建立索引库操作对象（IndexWriter），把文档对象写入索引库* IndexWriter writer = **new** IndexWriter(directory,iwc);   **for**(Document doc:documentList){  */\*\*  \* addDocument方法：把文档对象写入索引库  \*/* writer.addDocument(doc);  }  *// 7.释放资源* writer.close();  } |

图一：



图二：



# lucene索引库维护

说明：数据保存在关系数据库，需要实现增删改查操作。索引数据保存在索引库，也需要实现增删改查操作。

## 添加索引

参考入门程序内容：



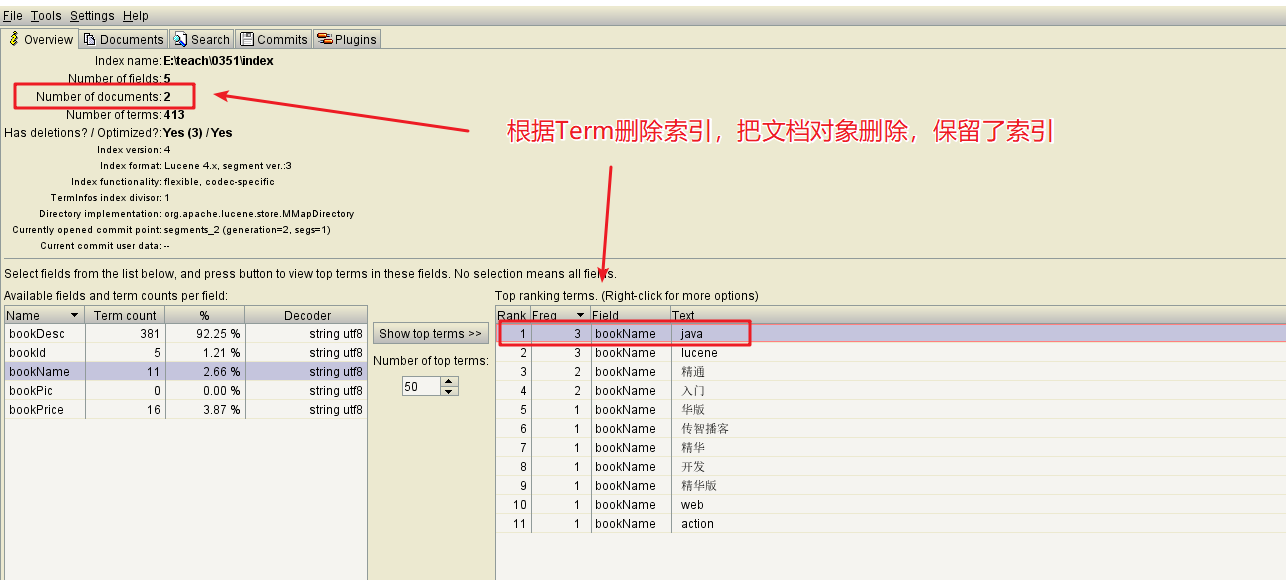
## 删除索引

### 根据Term删除索引

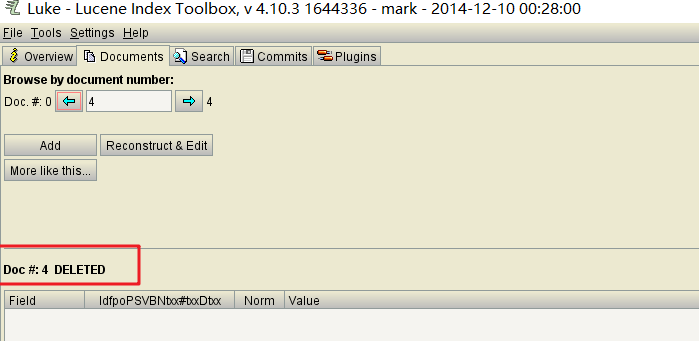
1. 建立分析器对象（Analyzer），用于分词
2. 建立索引库配置对象（IndexWriterConfig），配置索引库
3. 建立索引库目录对象（Directory），指定索引库的位置
4. 建立索引库操作对象（IndexWriter），操作索引库
5. 建立条件对象（Term）
6. 使用IndexWriter对象，执行删除
7. 释放资源

|  |
| --- |
| */\*\*  \* 根据Term删除索引  \*/* @Test  **public void** deleteIndexByTerm() **throws** Exception{ *// 1.建立分析器对象（Analyzer），用于分词* Analyzer analyzer = **new** IKAnalyzer(); *// 2.建立索引库配置对象（IndexWriterConfig），配置索引库* IndexWriterConfig iwc = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***,analyzer);  *// 3.建立索引库目录对象（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***));  *// 4.建立索引库操作对象（IndexWriter），操作索引库* IndexWriter writer = **new** IndexWriter(directory,iwc);  *// 5.建立条件对象（Term）  /\*\*  \* 关系数据库删除数据：delete from table where id=1  \*  \* 需求：删除图书名称域中，包含有java的图书  \*/* Term term = **new** Term(**"bookName"**,**"java"**);  *// 6.使用IndexWriter对象，执行删除* writer.deleteDocuments(term); *// 7.释放资源* writer.close();  } |

图一：



图二：

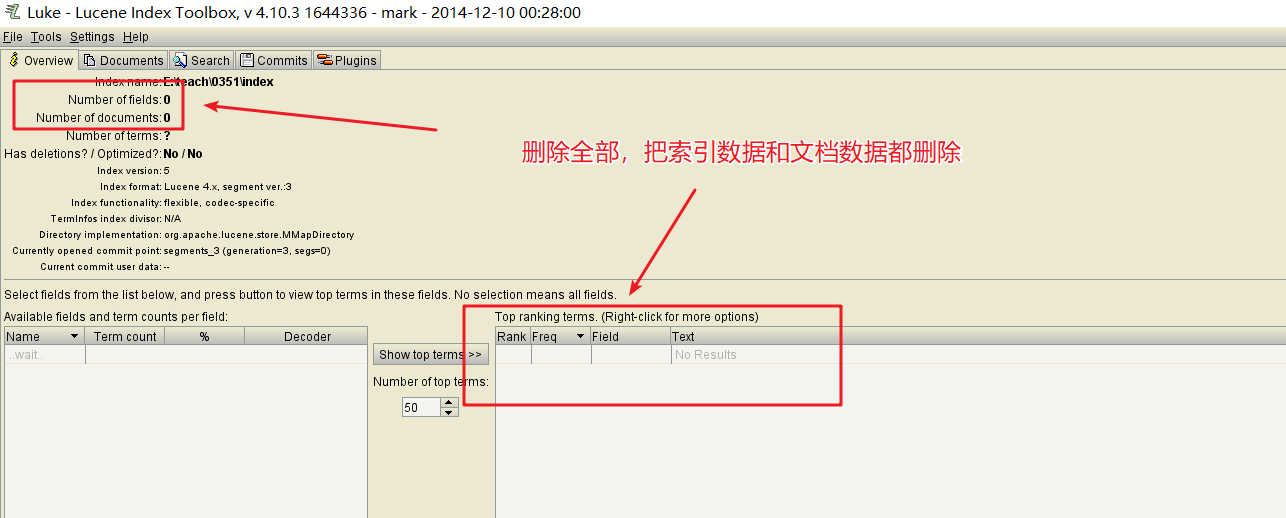


### 删除全部索引【慎用】

1. 建立分析器对象（Analyzer），用于分词
2. 建立索引库配置对象（IndexWriterConfig），配置索引库
3. 建立索引库目录对象（Directory），指定索引库的位置
4. 建立索引库操作对象（IndexWriter），操作索引库
5. 使用IndexWriter，执行删除
6. 释放资源

|  |
| --- |
| */\*\*  \* 删除全部索引  \*/* @Test  **public void** deleteAllIndex() **throws** Exception{ *// 1.建立分析器对象（Analyzer），用于分词* Analyzer analyzer = **new** IKAnalyzer(); *// 2.建立索引库配置对象（IndexWriterConfig），配置索引库* IndexWriterConfig iwc = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***,analyzer);  *// 3.建立索引库目录对象（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***));  *// 4.建立索引库操作对象（IndexWriter），操作索引库* IndexWriter writer = **new** IndexWriter(directory,iwc);  *// 5.使用IndexWriter，执行删除* writer.deleteAll(); *// 6.释放资源* writer.close();  } |

图一：

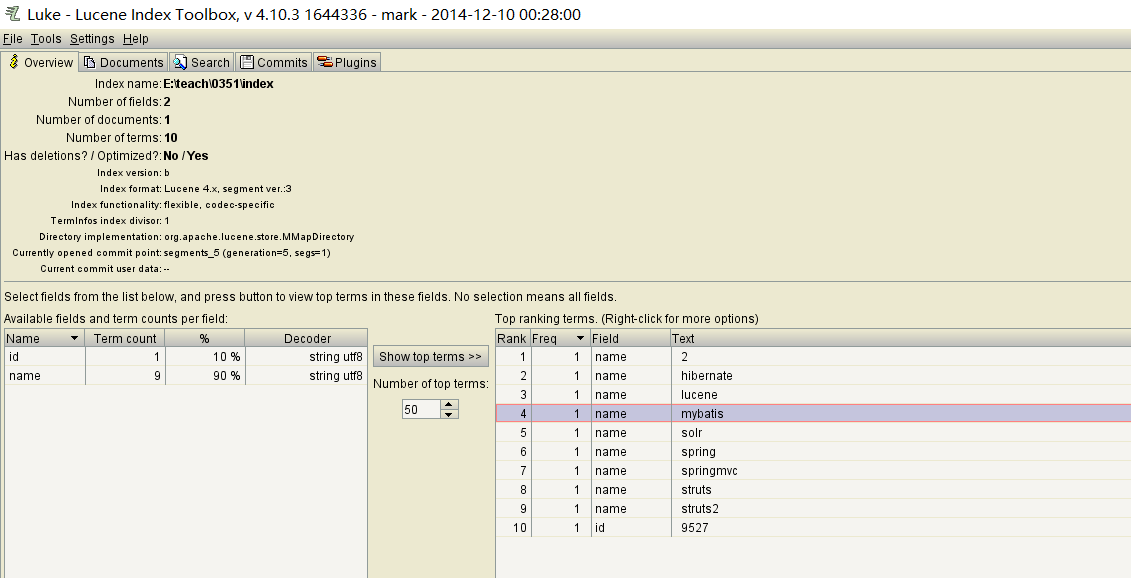


## 更新索引

说明：lucene是根据Term执行索引的更新，先根据Term执行搜索，搜索到执行更新；搜索不到执行添加。

1. 建立分析器对象（Analyzer），用于分词
2. 建立索引库配置对象（IndexWriterConfig），配置索引库
3. 建立索引库目录对象（Directory），指定索引库的位置
4. 建立索引库操作对象（IndexWriter），操作索引库
5. 建立文档对象（Document）
6. 建立条件对象（Term）
7. 使用IndexWriter，执行更新
8. 释放资源

|  |
| --- |
| */\*\*  \* 更新索引  \*/* @Test  **public void** updateIndex() **throws** Exception{ *// 1.建立分析器对象（Analyzer），用于分词* Analyzer analyzer = **new** IKAnalyzer(); *// 2.建立索引库配置对象（IndexWriterConfig），配置索引库* IndexWriterConfig iwc = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***,analyzer); *// 3.建立索引库目录对象（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***)); *// 4.建立索引库操作对象（IndexWriter），操作索引库* IndexWriter writer = **new** IndexWriter(directory,iwc);  *// 5.建立文档对象（Document）* Document doc = **new** Document();  doc.add(**new** TextField(**"id"**,**"9527"**, Field.Store.***YES***));  *//doc.add(new TextField("name","mybatis and springmvc and lucene and solr", Field.Store.YES));   // 测试更新* doc.add(**new** TextField(**"name"**,**"struts2 and hibernate and spring and mybatis and springmvc and lucene and solr"**, Field.Store.***YES***));  *// 6.建立条件对象（Term）* Term term = **new** Term(**"name"**,**"mybatis"**);  *// 7.使用IndexWriter，执行更新* writer.updateDocument(term,doc); *// 8.释放资源* writer.close();  } |



# lucene搜索

说明：在搜索中关心Query对象的创建。

## Query对象的两种创建方式

### 使用Query之类对象

#### 常用的Query子类对象

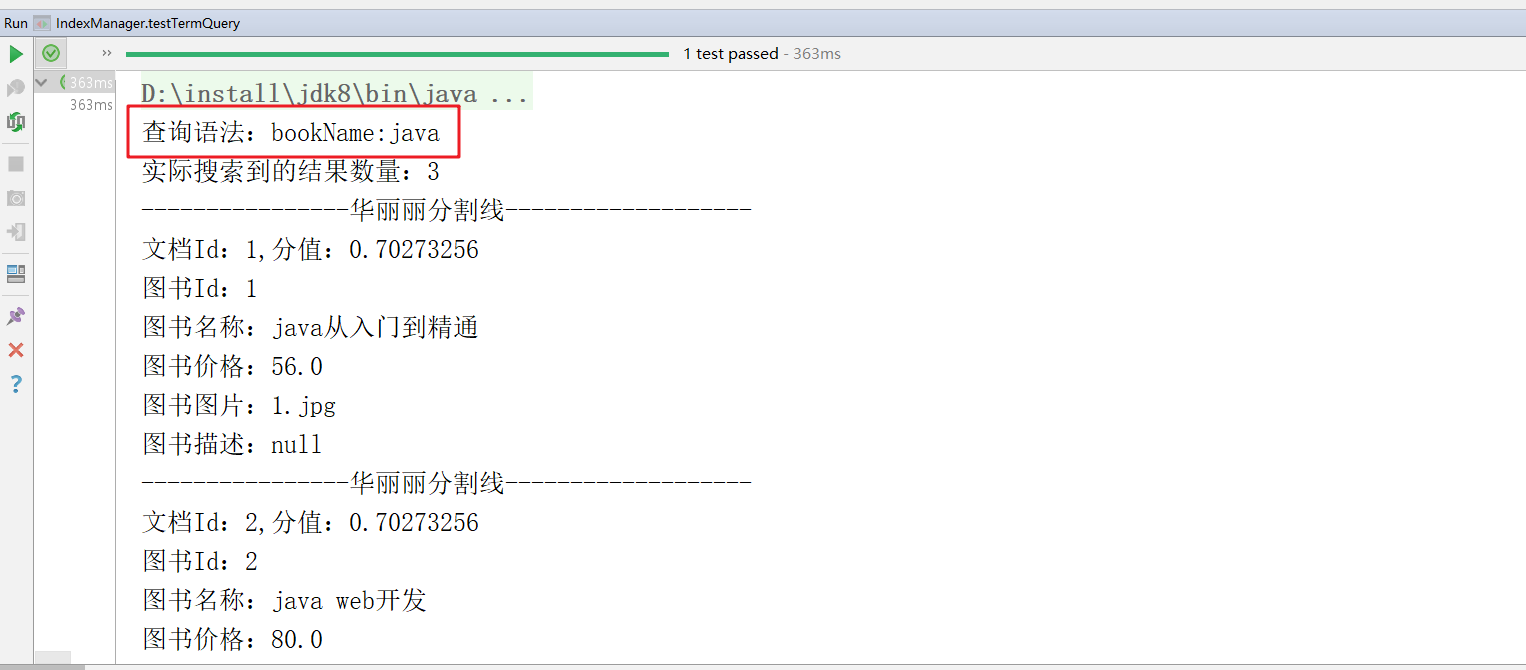
|  |  |
| --- | --- |
| 子类对象 | 说明 |
| TermQuery | 不使用分析器，对关键词做精确匹配搜索。比如：订单编号、身份证号 |
| NumericRangeQuery | 数字范围查询，比如：图书价格大于80，小于100 |
| BooleanQuery | 布尔查询，实现组合条件查询。  组合关系有：   1. MUST与MUST表示“与”，即“交集” 2. MUST与MUST NOT，包含前者，排除后者 3. MUST NOT与MUST NOT没有意义 4. SHOULD与MUST表示MUST，SHOULD失去意义 5. SHOULD与MUST NOT 等于MUST与MUST NOT 6. SHOULD与SHOULD表示“或”，即“并集” |

#### 常用Query子类对象使用

##### 使用TermQuery

需求：查询图书名称域中包含有java的图书。

|  |
| --- |
| */\*\*  \* 封装搜索方法  \*/* **private void** searcher(Query query) **throws** Exception{   *// 查询语法* System.***out***.println(**"查询语法："**+query);  *// 1.建立索引库的目录（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***));  *// 2.建立索引读取对象（IndexReader），把索引数据读取到内存中* IndexReader reader = DirectoryReader.*open*(directory);  *// 3.建立索引搜索对象（IndexSearcher），执行搜索* IndexSearcher searcher = **new** IndexSearcher(reader);  *// 4.使用IndexSearcher执行搜索，返回搜索结果集（TopDocs）  /\*\*  \* search方法：执行搜索  \* 参数：  \* 参数一：查询对象  \* 参数二：指定搜索结果排序后的前n个（前10个）  \*  \* topN分析  \*/* TopDocs topDoc = searcher.search(query, 10);  *// 5.处理结果集  // 5.1.实际搜索到的结果数量* System.***out***.println(**"实际搜索到的结果数量："**+topDoc.**totalHits**);   *// 5.2.打印实际的数据  /\*\*  \* ScoreDoc中：只有文档id和分值信息  \*/* ScoreDoc[] scoreDocs = topDoc.**scoreDocs**;  **for**(ScoreDoc sd:scoreDocs){  System.***out***.println(**"----------------华丽丽分割线-------------------"**);  *// 取出文档Id和分值* **float** score = sd.**score**;  **int** docId = sd.**doc**;  System.***out***.println(**"文档Id："**+docId+**",分值："**+score);   *// 根据文档Id从文档域中取数据* Document doc = searcher.doc(docId);   System.***out***.println(**"图书Id："**+doc.get(**"bookId"**));  System.***out***.println(**"图书名称："**+doc.get(**"bookName"**));  System.***out***.println(**"图书价格："**+doc.get(**"bookPrice"**));  System.***out***.println(**"图书图片："**+doc.get(**"bookPic"**));  System.***out***.println(**"图书描述："**+doc.get(**"bookDesc"**));  } *// 6.释放资源* reader.close();  } |
| */\*\*  \* 使用TermQuery  \* 需求：查询图书名称域中包含有java的图书。  \*/* @Test **public void** testTermQuery() **throws** Exception{  *// 1.创建查询对象* TermQuery tq = **new** TermQuery(**new** Term(**"bookName"**,**"java"**));   *// 2.执行搜索* **this**.searcher(tq); } |

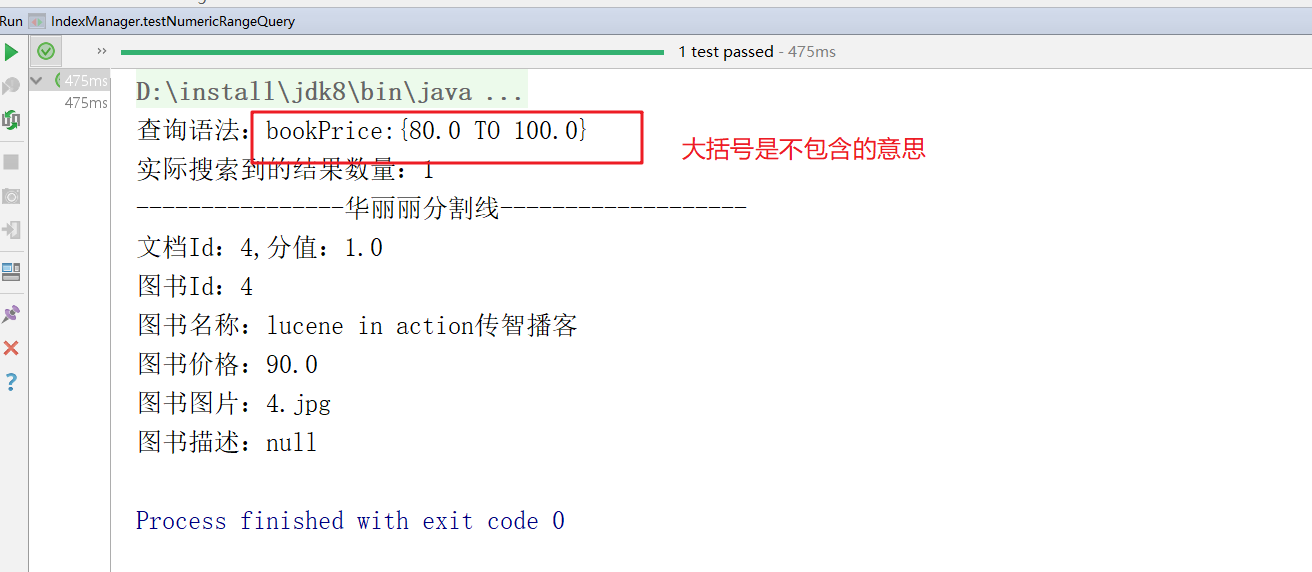


##### 使用NumericRangeQuery

需求：查询图书价格在80到100之间的图书。

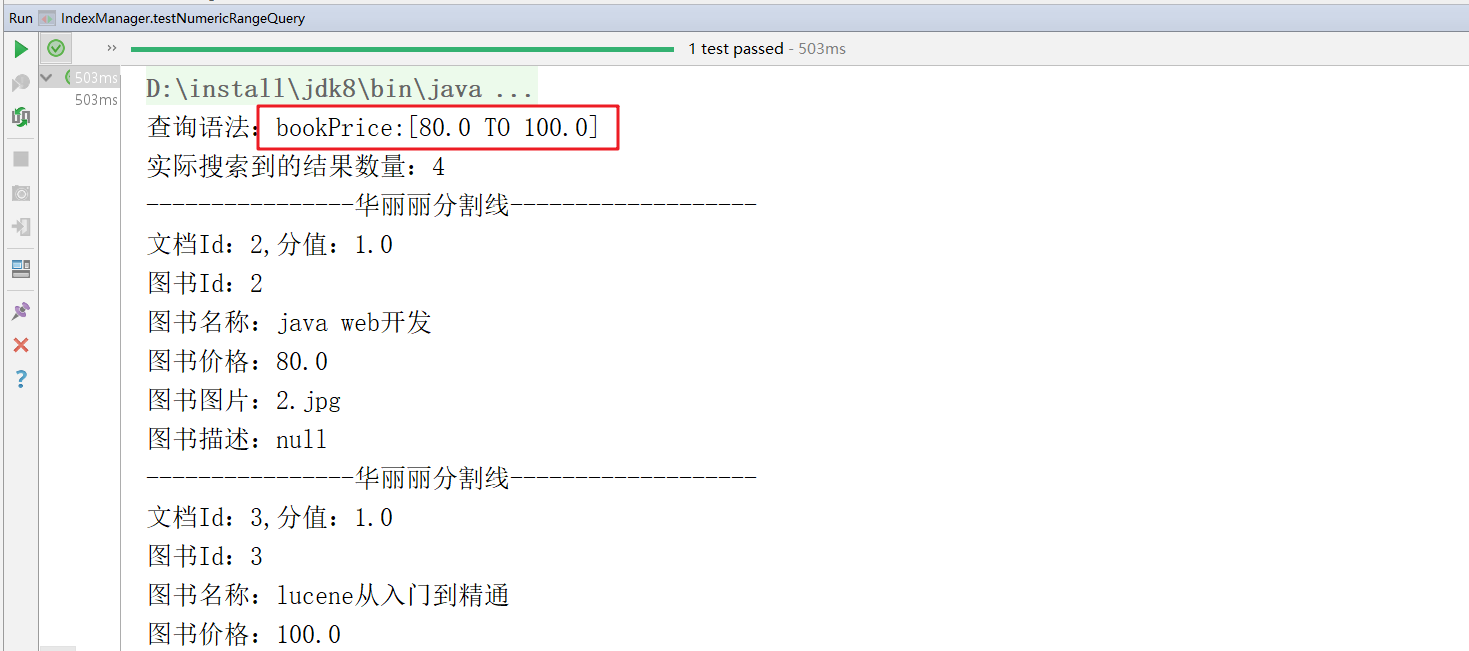
不包含 边界值：

|  |
| --- |
| */\*\*  \* 使用NumericRangeQuery  \* 需求：查询图书价格在80到100之间的图书。  \*/* @Test **public void** testNumericRangeQuery() **throws** Exception{  *// 1.创建查询对象  /\*\*  \* 参数：  \* field：域的名称  \* min：最小范围边界值  \* max：最大范围边界值  \* minInclusive:是否包含最小编界值  \* maxInclusive:是否包含最大边界值  \*/  // 不包含边界值* NumericRangeQuery nrq = NumericRangeQuery.*newDoubleRange*(**"bookPrice"**,80d,100d,**false**,**false**);   *// 2.执行搜索* **this**.searcher(nrq); } |



包含边界值：

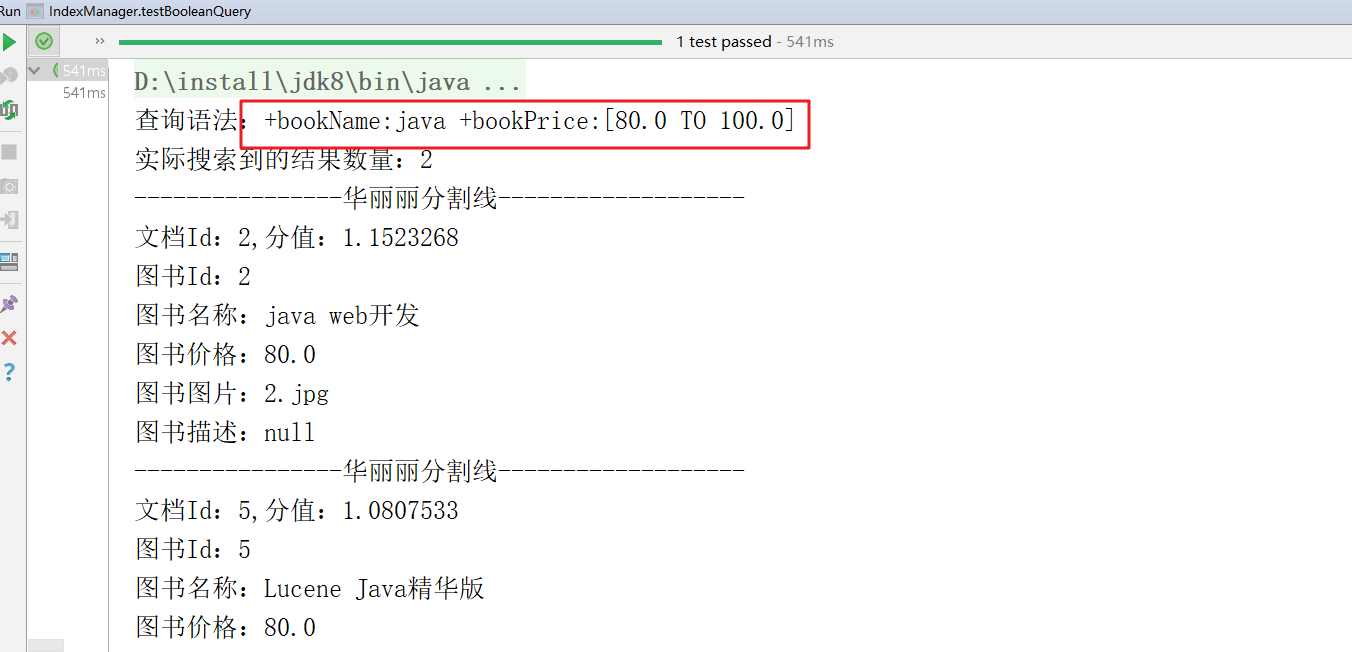
|  |
| --- |
| */\*\*  \* 使用NumericRangeQuery  \* 需求：查询图书价格在80到100之间的图书。  \*/* @Test **public void** testNumericRangeQuery() **throws** Exception{  *// 1.创建查询对象  /\*\*  \* 参数：  \* field：域的名称  \* min：最小范围边界值  \* max：最大范围边界值  \* minInclusive:是否包含最小编界值  \* maxInclusive:是否包含最大边界值  \*/  // 不包含边界值  // NumericRangeQuery nrq = NumericRangeQuery.newDoubleRange("bookPrice",80d,100d,false,false);   // 包含边界值* NumericRangeQuery nrq = NumericRangeQuery.*newDoubleRange*(**"bookPrice"**,80d,100d,**true**,**true**);   *// 2.执行搜索* **this**.searcher(nrq); } |



##### 使用BooleanQuery

需求：查询图书名称域中包含有java的图书，并且价格在80到100之间（包含边界值）。

|  |
| --- |
| */\*\*  \* 使用BooleanQuery  \*需求：查询图书名称域中包含有java的图书，并且价格在80到100之间（包含边界值）。  \*/* @Test **public void** testBooleanQuery() **throws** Exception{  *// 1.创建查询对象一* TermQuery query1 = **new** TermQuery(**new** Term(**"bookName"**,**"java"**));   *// 2.创建查询对象二* NumericRangeQuery query2 = NumericRangeQuery.*newDoubleRange*(**"bookPrice"**,80d,100d,**true**,**true**);   *// 3.创建组合查询条件* BooleanQuery bq = **new** BooleanQuery();  bq.add(query1, BooleanClause.Occur.***MUST***);  bq.add(query2, BooleanClause.Occur.***MUST***);   *// 4.执行搜索'* **this**.searcher(bq); } |



### 使用QueryParser解析表达式，实例化Query对象

#### 表达式语法

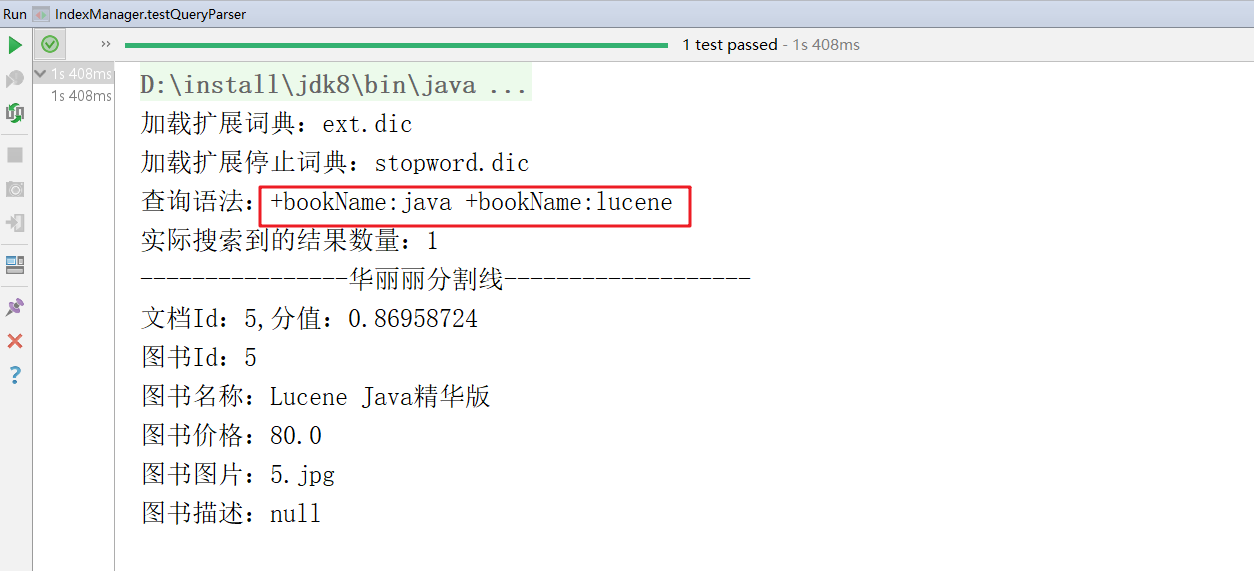
* 关键词基本查询：域名+“:”+关键词，比如：bookname:lucene
* 范围查询：域名+“:”+[最小值 TO 最大值]，比如：price:[80 TO 100]。需要注意QueryParser不支持数字范围查询，仅适用于字符串范围查询。如果有数字范围查询需求，请使用NumericRangeQuery。
* 组合查询：

|  |  |
| --- | --- |
| Occur.MUST搜索条件必须满足，相当于AND | + |
| Occur.SHOULD搜索条件可选，相当于OR | 空 |
| Occur.MUST\_NOT搜索条件不能满足，相当于NOT非 | - |

#### 使用QueryParser

需求：查询图书名称域中包含有java，并且图书名称域中包含有lucene的图书。

|  |
| --- |
| */\*\*  \* 使用QueryParser  \* 需求：查询图书名称域中包含有java，并且图书名称域中包含有lucene的图书。  \*/* @Test **public void** testQueryParser() **throws** Exception{  *// 1.建立分析器对象* Analyzer analyzer = **new** IKAnalyzer();   *// 2.建立查询解析器对象* QueryParser parser = **new** QueryParser(**"bookName"**,analyzer);   *// 解析表达式，实例化Query对象  /\*\*  \* bookName:java AND bookName:lucene  \*/* Query query = parser.parse(**"bookName:java AND bookName:lucene"**);   *// 3.执行搜索* **this**.searcher(query); } |



注意事项：表达式中的关键词AND/OR/NOT，必须要大写。

# lucene相关度排序

## 相关度介绍

什么是相关度？相关度指两个事物之间的关联关系（相关性）。lucene中指的是搜索关键词（Term）与搜索结果之间的相关性。比如：搜索bookname域中包含java的图书，则根据java在bookname中出现的次数和位置来判断相关性。

## 相关度打分

lucene对查询关键字和索引文档的相关度进行打分，得分高的就排在前边。如何打分呢？lucene是在用户进行检索时实时根据搜索的关键字计算出来的，分两步：

1）计算出词（Term）的权重

2）根据词的权重值，计算文档相关度得分。

什么是词的权重？

通过索引部分的学习，明确索引的最小单位是一个Term(索引词典中的一个词)。搜索也是从索引域中查询Term，再根据Term找到文档。Term对文档的重要性称为权重，影响Term权重有两个因素：

* Term Frequency (tf)：

指此Term在此文档中出现了多少次。tf 越大说明越重要。

词(Term)在文档中出现的次数越多，说明此词(Term)对该文档越重要，如“Lucene”这个词，在文档中出现的次数很多，说明该文档主要就是讲Lucene技术的。

* Document Frequency (df)：

指有多少文档包含此Term。df 越大说明越不重要。

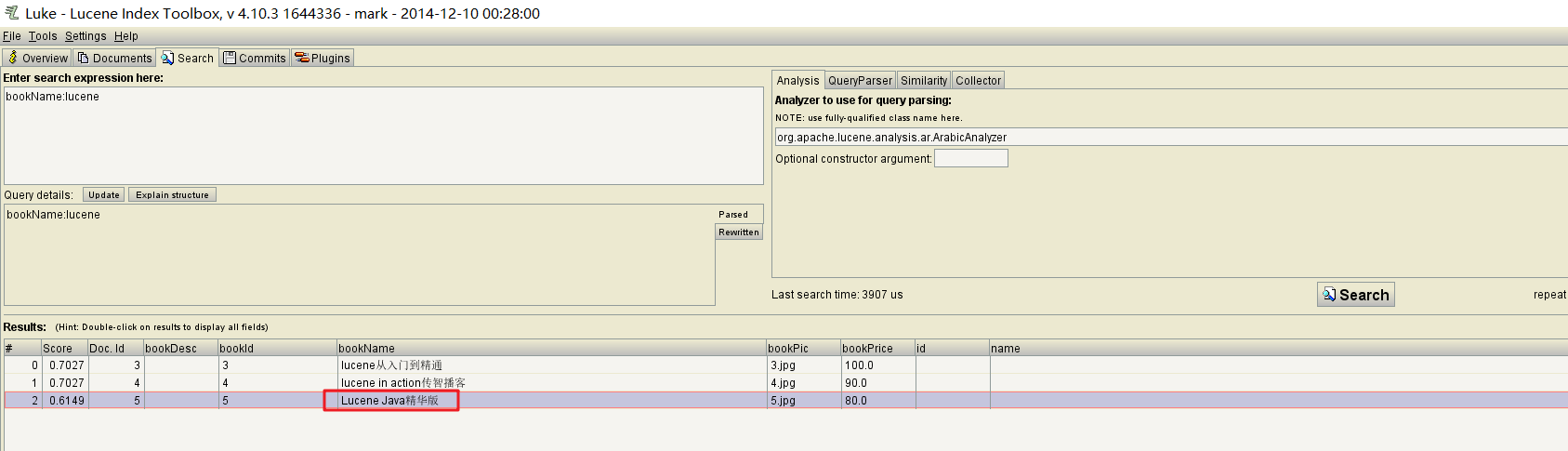
比如，在一篇英语文档中，this出现的次数更多，就说明越重要吗？不是的，有越多的文档包含此词(Term), 说明此词(Term)太普通，不足以区分这些文档，因而重要性越低。

## 相关度设置

说明：lucene是设置关键词Term的加权值（boost），改变相关度评分，从而影响搜索结果集排序。默认是有加权值的，默认的加权值都是1。

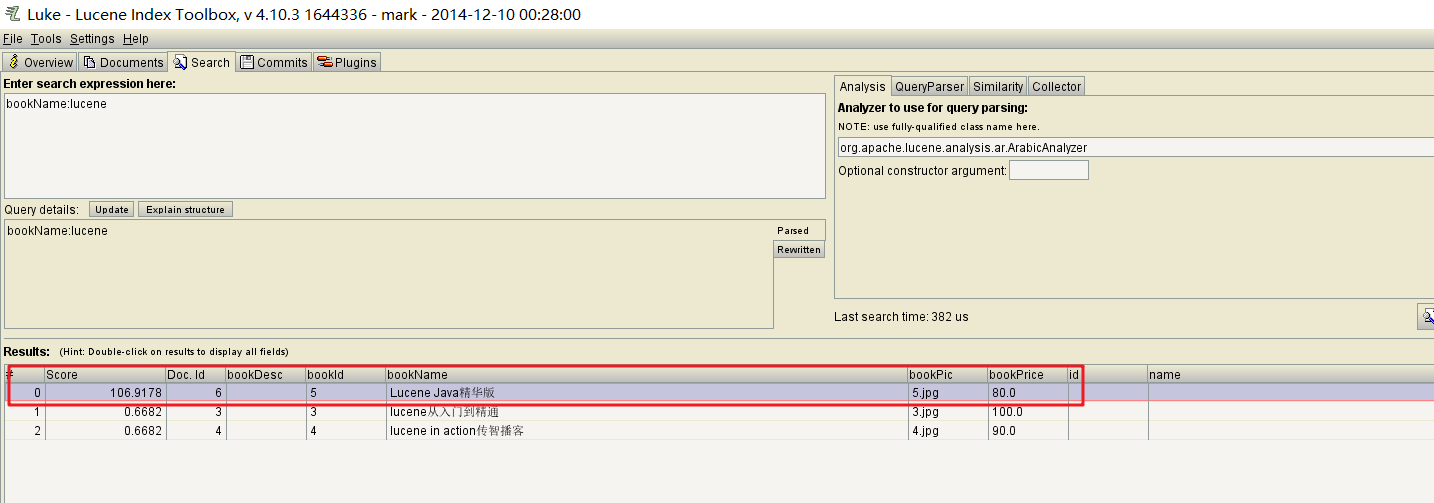
### 需求

说明：Lucene java精华版的作者，找到我们说给200块，期望在我们系统的搜索结果集中把这本书拍到第一的位置去，放置一个月的时间。



### 广告设置中

|  |
| --- |
| */\*\*  \* 更新索引(设置加权值，相关度排序)  \*/* @Test  **public void** updateIndexBoost() **throws** Exception{ *// 1.建立分析器对象（Analyzer），用于分词* Analyzer analyzer = **new** IKAnalyzer(); *// 2.建立索引库配置对象（IndexWriterConfig），配置索引库* IndexWriterConfig iwc = **new** IndexWriterConfig(Version.***LUCENE\_4\_10\_3***,analyzer); *// 3.建立索引库目录对象（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***)); *// 4.建立索引库操作对象（IndexWriter），操作索引库* IndexWriter writer = **new** IndexWriter(directory,iwc);  *// 5.建立文档对象（Document）* Document doc = **new** Document();  */\*\*  \* 5 Lucene Java精华版 80 5.jpg 本书总结搜索引擎相关理论与实际  \*/* doc.add(**new** StringField(**"bookId"**,**"5"**, Field.Store.***YES***));   TextField nameField = **new** TextField(**"bookName"**,**"Lucene Java精华版"**, Field.Store.***YES***);  *// 设置加权值* nameField.setBoost(200);   doc.add(nameField);   doc.add(**new** DoubleField(**"bookPrice"**,80d, Field.Store.***YES***));  doc.add(**new** StoredField(**"bookPic"**,**"5.jpg"**));  doc.add(**new** TextField(**"bookDesc"**,**"本书总结搜索引擎相关理论与实际"**, Field.Store.***NO***));  *// 6.建立条件对象（Term）* Term term = **new** Term(**"bookId"**,**"5"**);  *// 7.使用IndexWriter，执行更新* writer.updateDocument(term,doc); *// 8.释放资源* writer.close();  } |



# lucene高亮显示

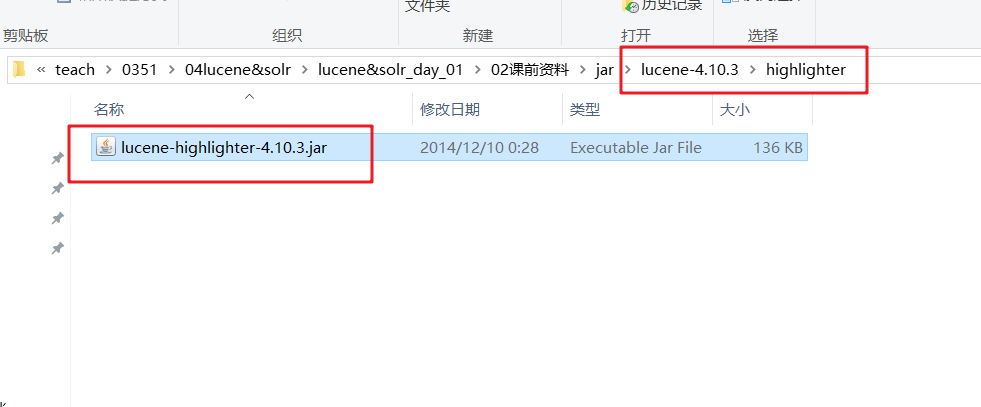
## 高亮显示介绍

说明：高亮显示指的是在搜索结果中，对搜索关键词进行突出显示（增加颜色）。



## 高亮实现

说明：lucene提供了高亮显示的组件。



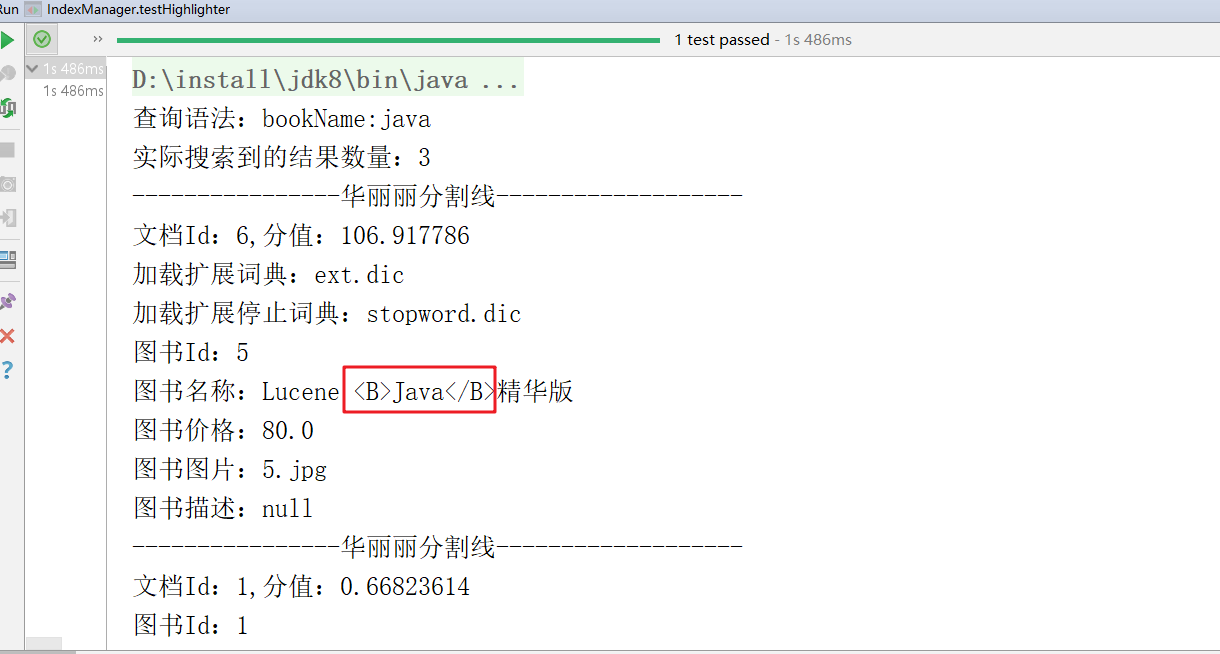
### 配置pom.xml加入依赖

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>cn.itheima</**groupId**>  <**artifactId**>lucene-second</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>   <**packaging**>jar</**packaging**>   <**properties**>  <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  *<!-- mysql版本 -->* <**mysql.version**>5.1.30</**mysql.version**>  *<!-- lucene版本 -->* <**lucene.version**>4.10.3</**lucene.version**>  *<!-- ik分词器版本 -->* <**ik.version**>2012\_u6</**ik.version**>  </**properties**>   <**dependencies**>  *<!-- mysql数据库依赖 -->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>${mysql.version}</**version**>  </**dependency**>  *<!--lucene依赖包 -->* <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-core</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-analyzers-common</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-queryparser</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  *<!--lucene高亮显示 -->* <**dependency**>  <**groupId**>org.apache.lucene</**groupId**>  <**artifactId**>lucene-highlighter</**artifactId**>  <**version**>${lucene.version}</**version**>  </**dependency**>  *<!-- ik分词器 -->* <**dependency**>  <**groupId**>com.janeluo</**groupId**>  <**artifactId**>ikanalyzer</**artifactId**>  <**version**>${ik.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  </**dependency**>  </**dependencies**>   </**project**> |

### 代码实现

1. 建立查询分值对象（QueryScorer），计算分值
2. 建立输出片段对象（Fragmenter），用于把内容切片
3. 建立高亮组件对象（Highlighter）
4. 建立分析器对象（Analyzer）
5. 使用TokenSoures类，获取文档对象的流对象（TokenStream）
6. 使用Highlighter，完成高亮显示

|  |
| --- |
| */\*\*  \* 封装搜索方法(实现高亮显示)  \*/* **private void** searcherHighlighter(Query query) **throws** Exception{   *// 查询语法* System.***out***.println(**"查询语法："**+query);  *// 1.建立索引库的目录（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***));  *// 2.建立索引读取对象（IndexReader），把索引数据读取到内存中* IndexReader reader = DirectoryReader.*open*(directory);  *// 3.建立索引搜索对象（IndexSearcher），执行搜索* IndexSearcher searcher = **new** IndexSearcher(reader);  *// 4.使用IndexSearcher执行搜索，返回搜索结果集（TopDocs）  /\*\*  \* search方法：执行搜索  \* 参数：  \* 参数一：查询对象  \* 参数二：指定搜索结果排序后的前n个（前10个）  \*  \* topN分析  \*/* TopDocs topDoc = searcher.search(query, 10);   *// 增加高亮实现=====================================start  // 1.建立查询分值对象（QueryScorer），计算分值* QueryScorer qs = **new** QueryScorer(query);  *// 2.建立输出片段对象（Fragmenter），用于把内容切片* Fragmenter fragmenter = **new** SimpleSpanFragmenter(qs);  *// 3.建立高亮组件对象（Highlighter）* Highlighter highlighter = **new** Highlighter(qs);  *// 设置输出片段对象* highlighter.setTextFragmenter(fragmenter);  *// 4.建立分析器对象（Analyzer）* Analyzer analyzer = **new** IKAnalyzer();   *// 增加高亮实现=====================================end  // 5.处理结果集  // 5.1.实际搜索到的结果数量* System.***out***.println(**"实际搜索到的结果数量："**+topDoc.**totalHits**);   *// 5.2.打印实际的数据  /\*\*  \* ScoreDoc中：只有文档id和分值信息  \*/* ScoreDoc[] scoreDocs = topDoc.**scoreDocs**;  **for**(ScoreDoc sd:scoreDocs){  System.***out***.println(**"----------------华丽丽分割线-------------------"**);  *// 取出文档Id和分值* **float** score = sd.**score**;  **int** docId = sd.**doc**;  System.***out***.println(**"文档Id："**+docId+**",分值："**+score);   *// 根据文档Id从文档域中取数据* Document doc = searcher.doc(docId);   *// 高亮显示图书名称* String bookName = doc.get(**"bookName"**);  **if**(bookName !=**null**){  *// 5.使用TokenSoures类，获取文档对象的流对象（TokenStream）  /\*\*  \* getTokenStream：获取文档对象的流对象  \* 参数：  \* 参数一：文档对象  \* 参数二：高亮显示的域的名称  \* 参数：分析器  \*/* TokenStream tokenStream = TokenSources.*getTokenStream*(doc, **"bookName"**, analyzer);   *// 6.使用Highlighter，完成高亮显示  /\*\*  \* getBestFragment:获取高亮显示内容  \* 参数：  \* 参数一：文档对象的流对象  \* 参数二：高亮显示的目标内容  \*/* bookName = highlighter.getBestFragment(tokenStream,bookName);  }    System.***out***.println(**"图书Id："**+doc.get(**"bookId"**));   System.***out***.println(**"图书名称："**+bookName);   System.***out***.println(**"图书价格："**+doc.get(**"bookPrice"**));  System.***out***.println(**"图书图片："**+doc.get(**"bookPic"**));  System.***out***.println(**"图书描述："**+doc.get(**"bookDesc"**));  } *// 6.释放资源* reader.close();  } |
| */\*\*  \* 测试高亮显示  \* 需求：查询图书名称域中包含有java的图书  \*/* @Test **public void** testHighlighter() **throws** Exception{  *// 1.创建查询对象* TermQuery tq = **new** TermQuery(**new** Term(**"bookName"**,**"java"**));   *// 2.执行搜索* **this**.searcherHighlighter(tq); } |



### 实现自定义html标签高亮显示

1. 建立html格式化标签对象SimpleHTMLFormatter
2. 建立Highligter对象，指定使用SimpleHTMLFormatter

|  |
| --- |
| */\*\*  \* 封装搜索方法(实现高亮显示)  \*/* **private void** searcherHighlighter(Query query) **throws** Exception{   *// 查询语法* System.***out***.println(**"查询语法："**+query);  *// 1.建立索引库的目录（Directory），指定索引库的位置* Directory directory = FSDirectory.*open*(**new** File(***INDEX\_PATH***));  *// 2.建立索引读取对象（IndexReader），把索引数据读取到内存中* IndexReader reader = DirectoryReader.*open*(directory);  *// 3.建立索引搜索对象（IndexSearcher），执行搜索* IndexSearcher searcher = **new** IndexSearcher(reader);  *// 4.使用IndexSearcher执行搜索，返回搜索结果集（TopDocs）  /\*\*  \* search方法：执行搜索  \* 参数：  \* 参数一：查询对象  \* 参数二：指定搜索结果排序后的前n个（前10个）  \*  \* topN分析  \*/* TopDocs topDoc = searcher.search(query, 10);   *// 增加高亮实现=====================================start  // 1.建立查询分值对象（QueryScorer），计算分值* QueryScorer qs = **new** QueryScorer(query);  *// 2.建立输出片段对象（Fragmenter），用于把内容切片* Fragmenter fragmenter = **new** SimpleSpanFragmenter(qs);  *// 3.建立高亮组件对象（Highlighter）  //Highlighter highlighter = new Highlighter(qs);   // 增加自定义html高亮显示标签处理 // 1.建立html格式化标签对象SimpleHTMLFormatter  /\*\*  \* 参数：  \* preTag：html标签的开始部分  \* postTag：html标签的结束部分  \*/* SimpleHTMLFormatter formatter = **new** SimpleHTMLFormatter(**"<font color='red'>"**,**"</font>"**); *// 2.建立Highligter对象，指定使用SimpleHTMLFormatter* Highlighter highlighter = **new** Highlighter(formatter,qs);    *// 设置输出片段对象* highlighter.setTextFragmenter(fragmenter);  *// 4.建立分析器对象（Analyzer）* Analyzer analyzer = **new** IKAnalyzer();   *// 增加高亮实现=====================================end  // 5.处理结果集  // 5.1.实际搜索到的结果数量* System.***out***.println(**"实际搜索到的结果数量："**+topDoc.**totalHits**);   *// 5.2.打印实际的数据  /\*\*  \* ScoreDoc中：只有文档id和分值信息  \*/* ScoreDoc[] scoreDocs = topDoc.**scoreDocs**;  **for**(ScoreDoc sd:scoreDocs){  System.***out***.println(**"----------------华丽丽分割线-------------------"**);  *// 取出文档Id和分值* **float** score = sd.**score**;  **int** docId = sd.**doc**;  System.***out***.println(**"文档Id："**+docId+**",分值："**+score);   *// 根据文档Id从文档域中取数据* Document doc = searcher.doc(docId);   *// 高亮显示图书名称* String bookName = doc.get(**"bookName"**);  **if**(bookName !=**null**){  *// 5.使用TokenSoures类，获取文档对象的流对象（TokenStream）  /\*\*  \* getTokenStream：获取文档对象的流对象  \* 参数：  \* 参数一：文档对象  \* 参数二：高亮显示的域的名称  \* 参数：分析器  \*/* TokenStream tokenStream = TokenSources.*getTokenStream*(doc, **"bookName"**, analyzer);   *// 6.使用Highlighter，完成高亮显示  /\*\*  \* getBestFragment:获取高亮显示内容  \* 参数：  \* 参数一：文档对象的流对象  \* 参数二：高亮显示的目标内容  \*/* bookName = highlighter.getBestFragment(tokenStream,bookName);  }    System.***out***.println(**"图书Id："**+doc.get(**"bookId"**));   System.***out***.println(**"图书名称："**+bookName);   System.***out***.println(**"图书价格："**+doc.get(**"bookPrice"**));  System.***out***.println(**"图书图片："**+doc.get(**"bookPic"**));  System.***out***.println(**"图书描述："**+doc.get(**"bookDesc"**));  } *// 6.释放资源* reader.close();  } |



# solr介绍

## solr是什么

Solr是Apache下的一个顶级开源项目，基于Lucene的全文搜索服务，可以独立运行于Jetty，Tomcat等web容器中。

使用Solr可以很方便的实现全文索引和搜索功能，如下：

* 创建索引

客户端可以通过http的post方法向solr服务器发送一个描述Field及其内容的xml文档，solr服务器根据xml文档添加、更新、删除索引

* 搜索索引

客户端可以通过http的get方法向solr服务器发送请求，对solr服务器返回的xml、json格式结果进行解析。solr不提供构建页面UI的功能。solr提供了一个管理界面，通过管理界面可以查询solr的配置和运行情况。

## solr与lucene的区别

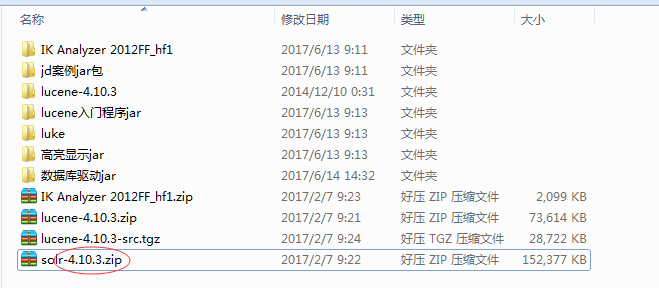
Lucene是一个全文检索工具包，提供了索引引擎和查询引擎功能，不是一个完整的服务，不能独立运行。Solr是基于Lucene构建的全文检索服务，是一个完整的服务，可以独立运行于web容器中。基于Solr可以快速构建企业级的搜索引擎。

# solr安装部署

## 下载solr

下载地址：<http://archive.apache.org/dist/lucene/solr/>

说明：solr与lucene是同步更新的，课程中使用4.10.3版本。



## 解压solr



目录说明：

bin：solr运行脚本

contrib：solr的一些扩展jar包，用于增强solr功能

dist：build过程中生成的war和jar文件，以及一些依赖文件

docs：solr的API文档

example：solr工程的例子目录

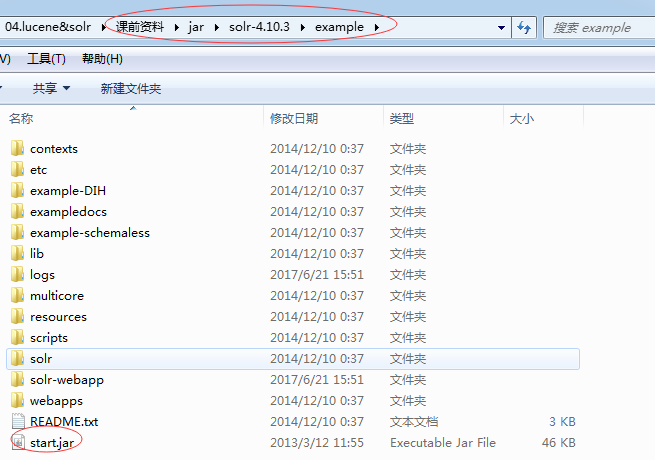
* example/solr：标准的SolrHome，包含一个默认的SolrCore
* example/multicore：包含了在Solr的multicore中设置的多个Core目录
* example/webapps：包含了一个solr.war，该war可作为solr的运行示例工程，我们部署的solr服务就是用的它

licenes：solr相关的许可信息

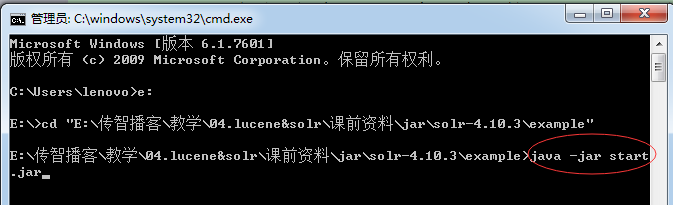
## 准备安装环境

solr需要运行在servlet容器中，并且solr4.10.3要求jdk版本要在1.7以上。solr本身集成了Jetty服务器，可以直接启动运行。

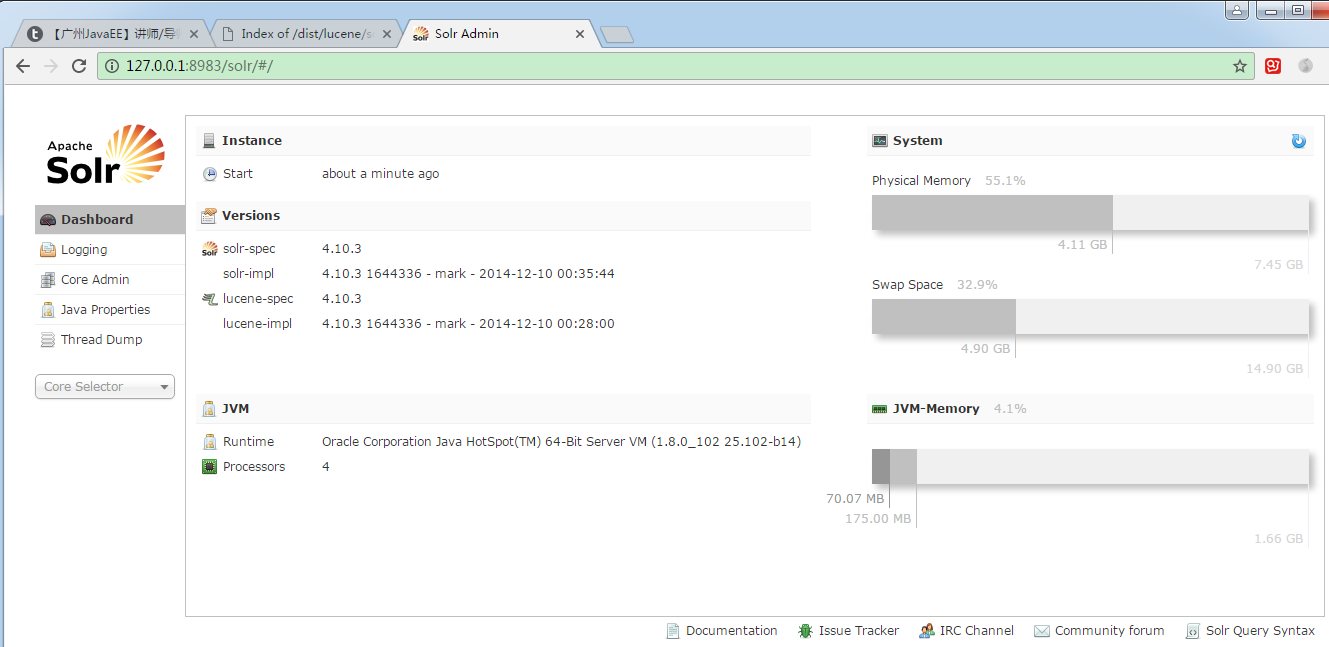
打开cmd命令行窗口，进入solr解压目录中的example目录。



执行：java -jar start.jar



启动成功，访问：<http://127.0.0.1:8983/solr>



说明：在企业项目中，一般都使用tomcat服务器构建solr服务。

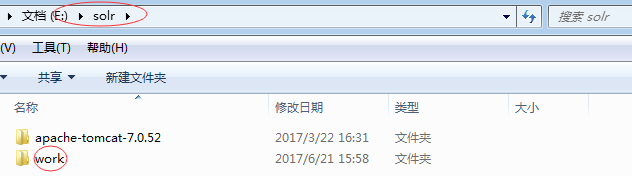
需要环境：

* Solr：4.10.3
* Jdk环境：1.8（solr4.10 要求jdk1.7以上版本）
* 服务器：Tomcat 7

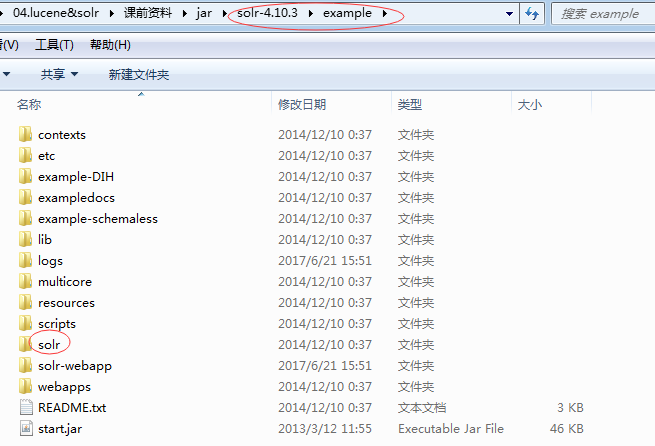
## SolrCore配置

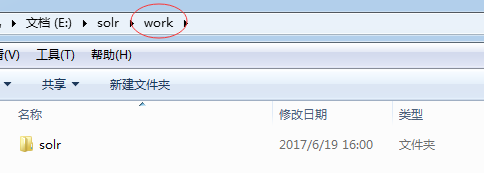
SolrHome目录是solr服务运行的主目录。一个SolrHome目录包含有多个SolrCore。SolrCore目录中包含了运行solr实例的配置文件和数据文件（日志和索引文件）。每一个SolrCore提供独立的索引和搜索服务。

### 建立work目录，作为SolrHome根目录

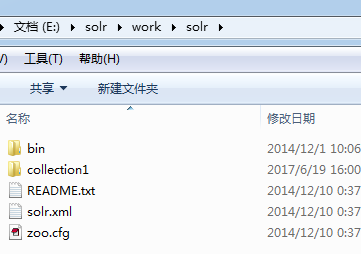


### 将solr解压目录中的solr目录，拷贝到work目录下

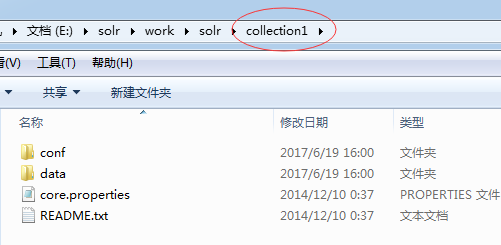




SolrHome目录：



SolrCore目录：



说明：

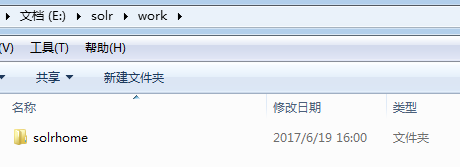
conf：SolrCore运行配置信息

data：SolrCore存放日志和索引文件的目录

core.properties：SolrCore的信息，比如：名称

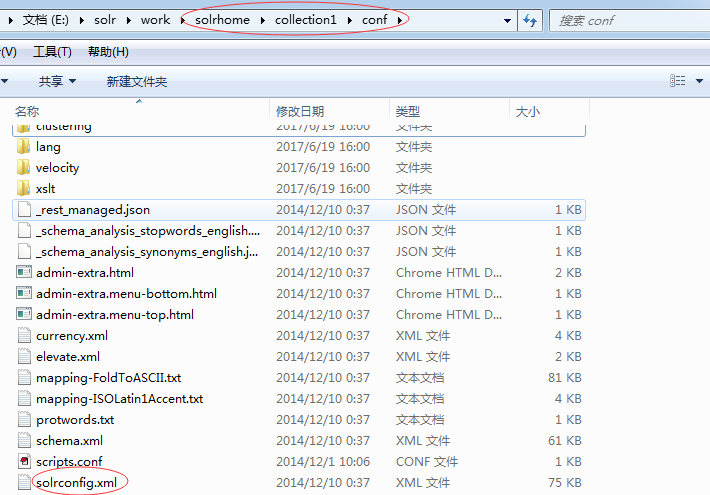
### 创建SolrCore

将work目录下的solr目录，重命名为：solrhome。（修改名称是为了方便管理，可以不修改）



### 配置SolrCore

配置SolrCore目录下的conf/solrconfig.xml文件：

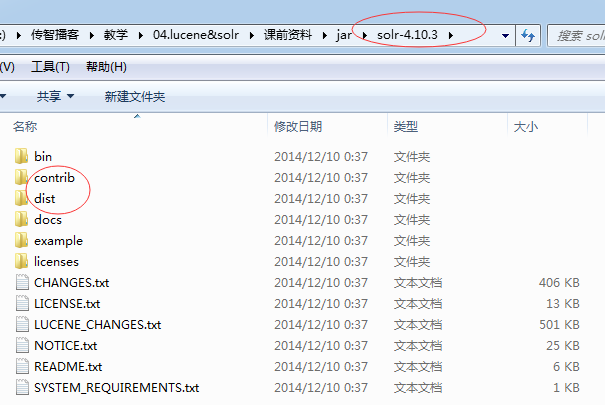


说明：solrconfig.xml文件是配置SolrCore实例的相关信息。默认情况下可以不做修改。在企业项目中会需要修改三个常用的标签：lib标签、datadir标签、requestHandler标签。

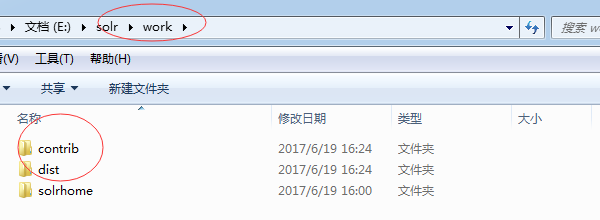
#### lib标签

在solrconfig.xml中可以加扩展载一些的jar，如果需要使用，则首先要把这些jar复制到指定的目录，我们复制到SolrHome同级目录。

复制solr解压文件目录中的contrib和dist文件目录。



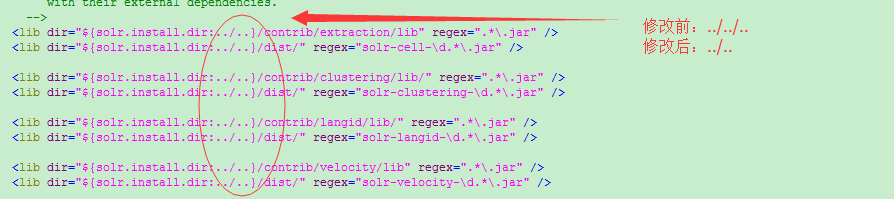
复制到work目录下：



修改solrconfig.xml文件，加载扩展jar包：

solr.install.dir表示${SolrCore}的目录位置，需要如下修改：

./ 表示当前目录 ../表示上一级目录

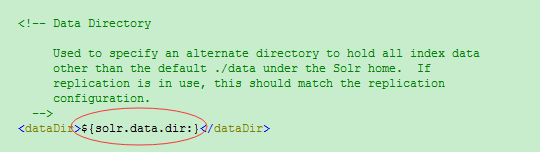


#### datadir标签

配置SolrCore的data目录。

data目录用来存放SolrCore的索引文件和tlog日志文件。

solr.data.dir表示${SolrCore}/data的目录位置

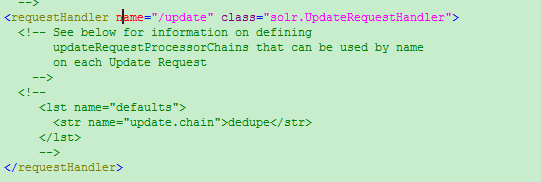


说明：一般不需要修改。

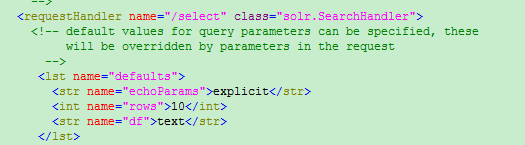
#### requestHandler标签

requestHandler请求处理器，定义了索引和搜索的访问方式。

通过/update维护索引，可以完成索引的添加、修改、删除操作。



通过/select搜索索引



设置搜索参数完成搜索，搜索参数也可以设置一些默认值，如下：

<requestHandler name="/select" class="solr.SearchHandler">

<!-- 设置默认的参数值，可以在请求地址中修改这些参数-->

<lst name="defaults">

<str name="echoParams">explicit</str>

<int name="rows">10</int><!--显示数量-->

<str name="wt">json</str><!--显示格式-->

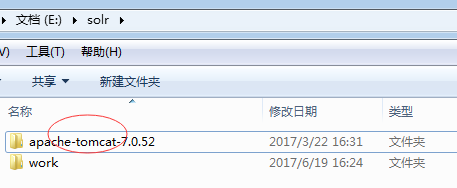
<str name="df">text</str><!--默认搜索字段-->

</lst>

</requestHandler>

## Solr工程部署

### 安装tomcat

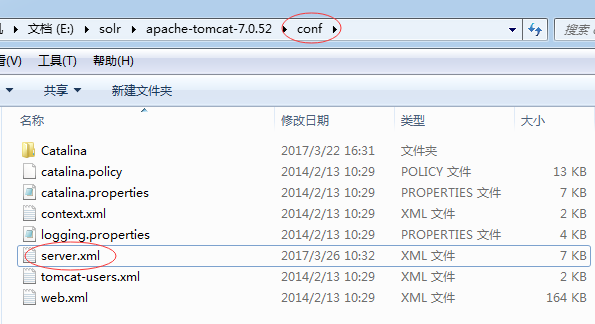


#### 删除不需要的应用

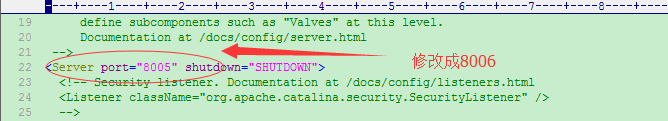


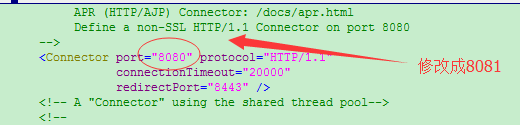
#### 修改服务端口

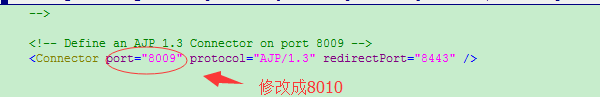
修改conf/server.xml文件。（修改的目的是避免端口冲突）。



修改以下三个端口：

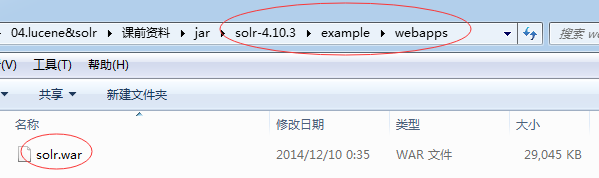




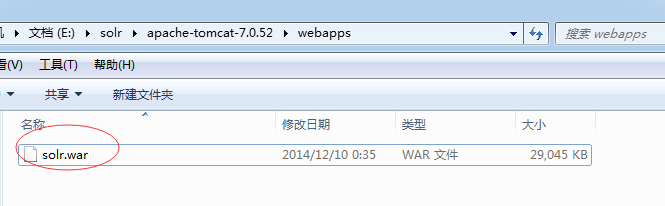


### 部署solr.war

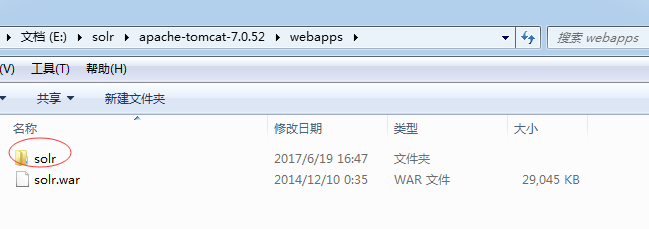
#### solr解压包下的solr-4.10.3\example\webapps目录中拷贝solr.war，到tomcat的webapps目录



复制到：



#### 解压solr.war包

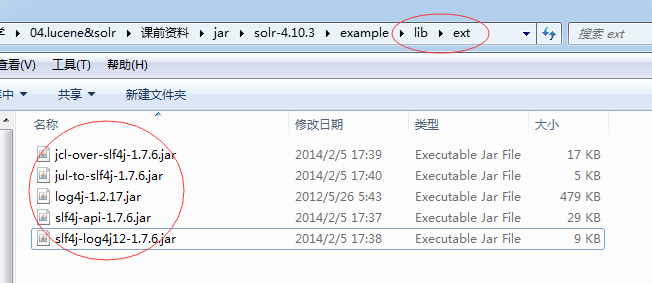


#### 删除solr.war包

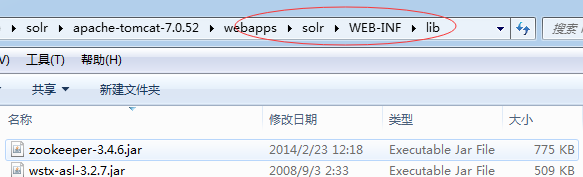
说明：删除为了防止tomcat启动的时候，再进行解压，覆盖已经解压配置好的solr。

#### 加入solr服务扩展jar包

把solr解压包下solr-4.10.3\example\lib\ext目录下的所有jar包拷贝到Tomcat部署的solr的WEB-INF/lib文件夹

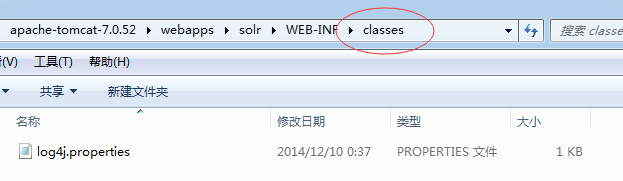


复制到：



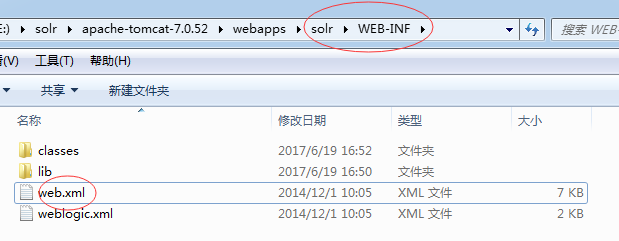
#### 准备log4j.properties 日志文件

把solr解压包下solr-4.10.3\example\resources\log4j.properties文件，复制到Tomcat的webapps下，solr中的WEB-INF\classes目录下（如果没有classes目录，创建一个）：



#### 配置web.xml

修改web.xml，让Tomcat使用JNDI的方式告诉solr服务器SolrHome位置。

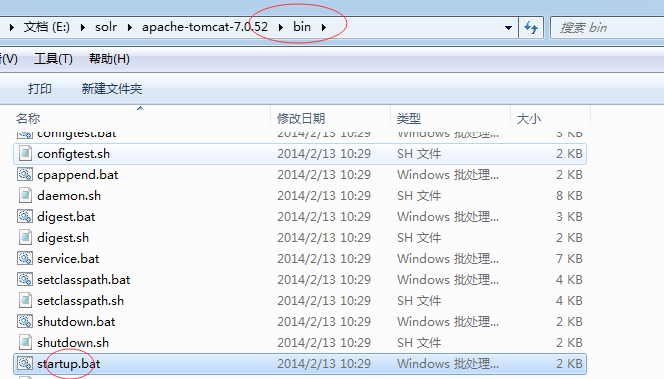


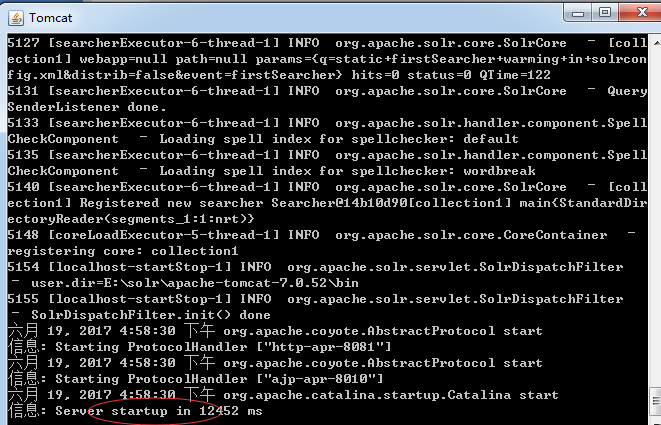
打开env-entry注释，修改env-entry-value为solrhome目录位置。



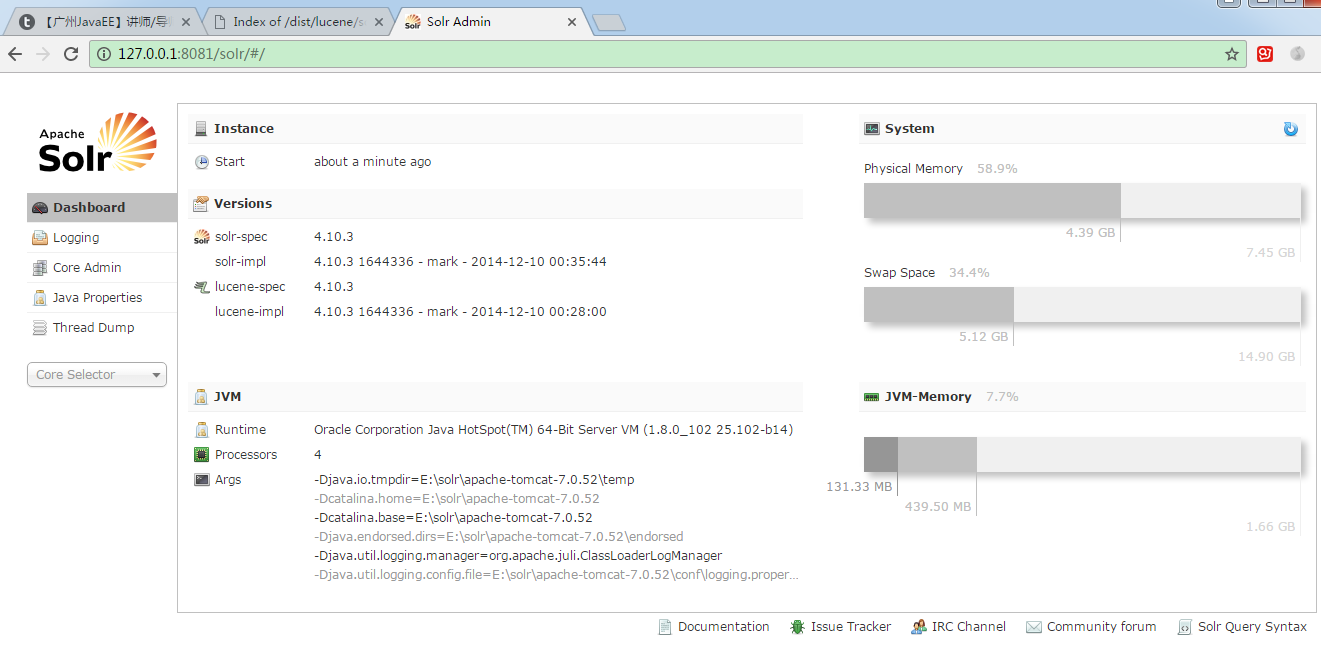
#### 启动tomcat服务

运行tomcat安装bin目录中的startup.bat文件：





访问：<http://127.0.0.1:8081/solr>



### solr管理界面介绍

#### Dashboard

仪表盘，显示Solr实例运行的时间、版本、系统资源、JVM等信息。

#### Logging

日志

#### CoreAdmin

Solr Core的管理界面。在这里可以添加SolrCore实例。

#### Java Properties

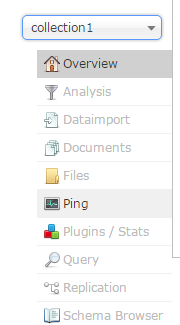
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

#### Thread Dump

显示Solr Server中当前活跃线程信息，同时也可以跟踪线程运行栈信息。

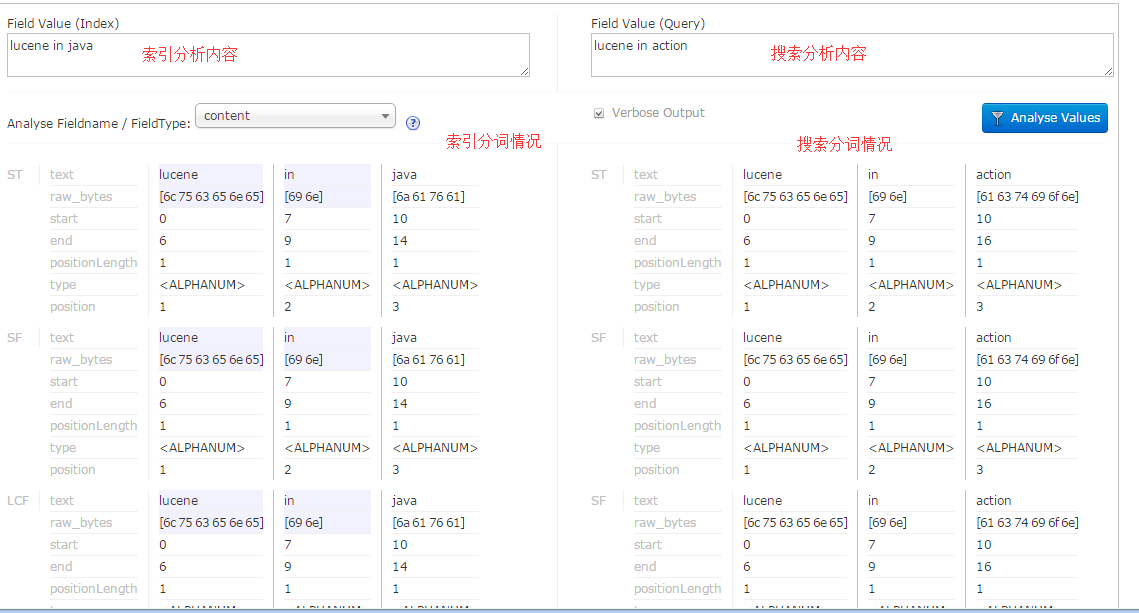
#### Core selector

选择一个SolrCore进行详细操作，如下：



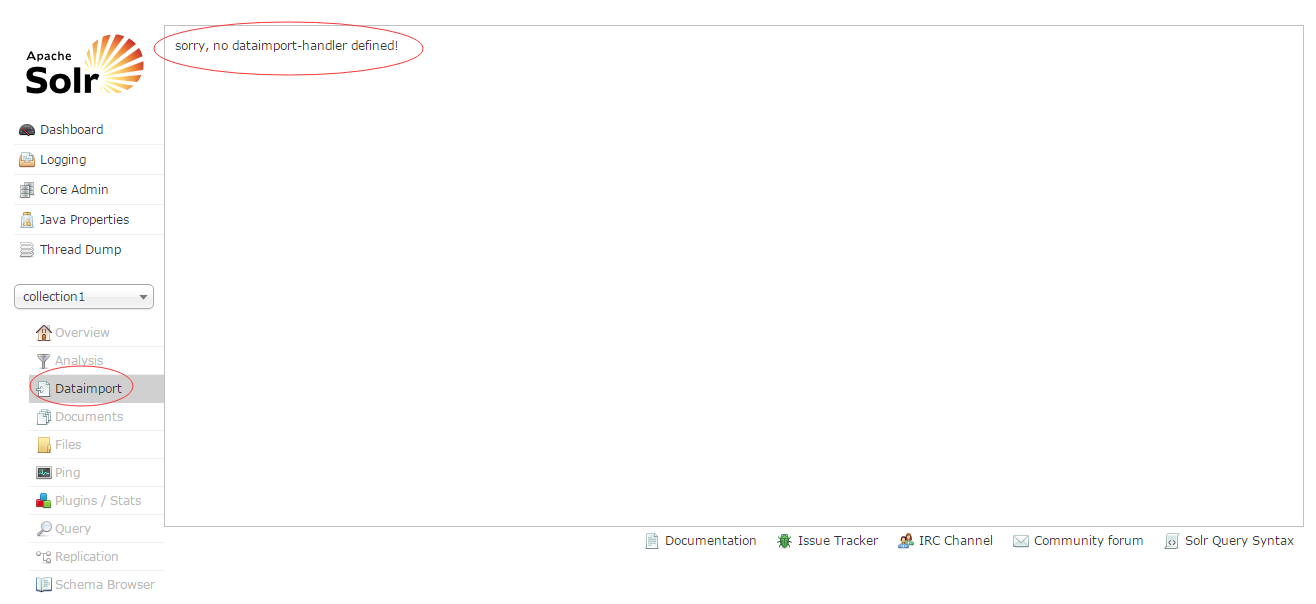
##### Analysis

通过此界面可以测试索引分析器和搜索分析器的执行情况



##### Dataimport

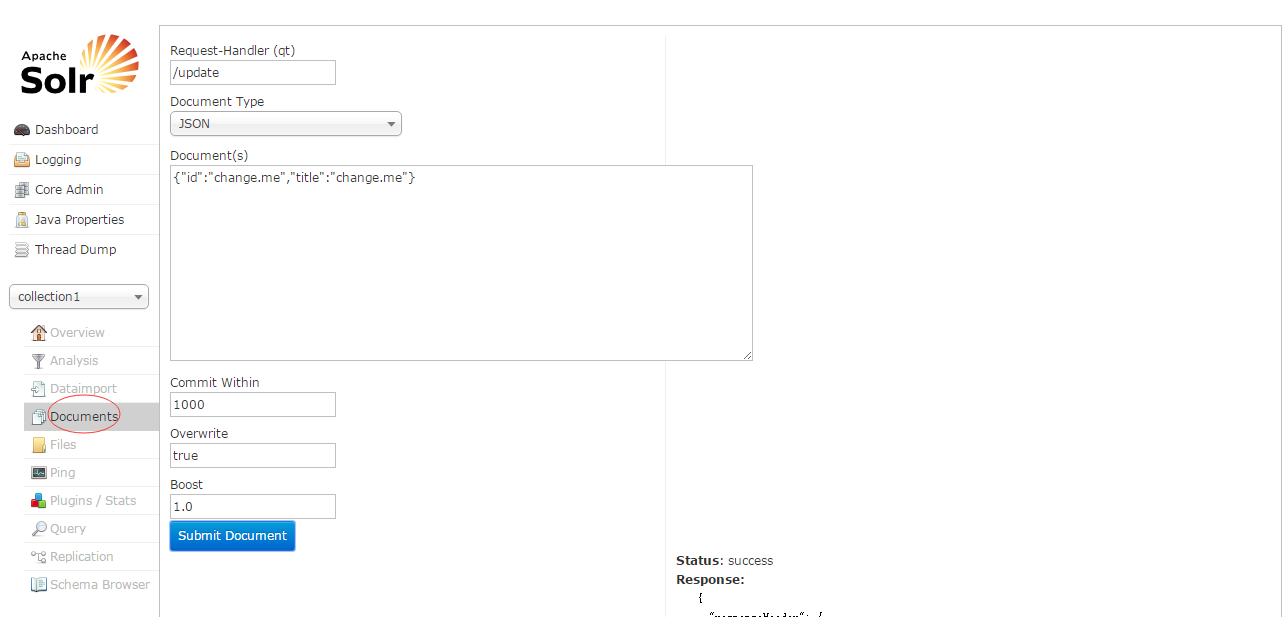
可以定义数据导入处理器，从关系数据库将数据导入到Solr索引库中。默认没有配置，需要手工配置。



##### Documents

通过/update表示更新索引，solr默认根据id（唯一约束）域来更新Document的内容，如果根据id值搜索不到id域则会执行添加操作，如果找到则更新。

通过此菜单可以创建索引、更新索引、删除索引等操作，界面如下：

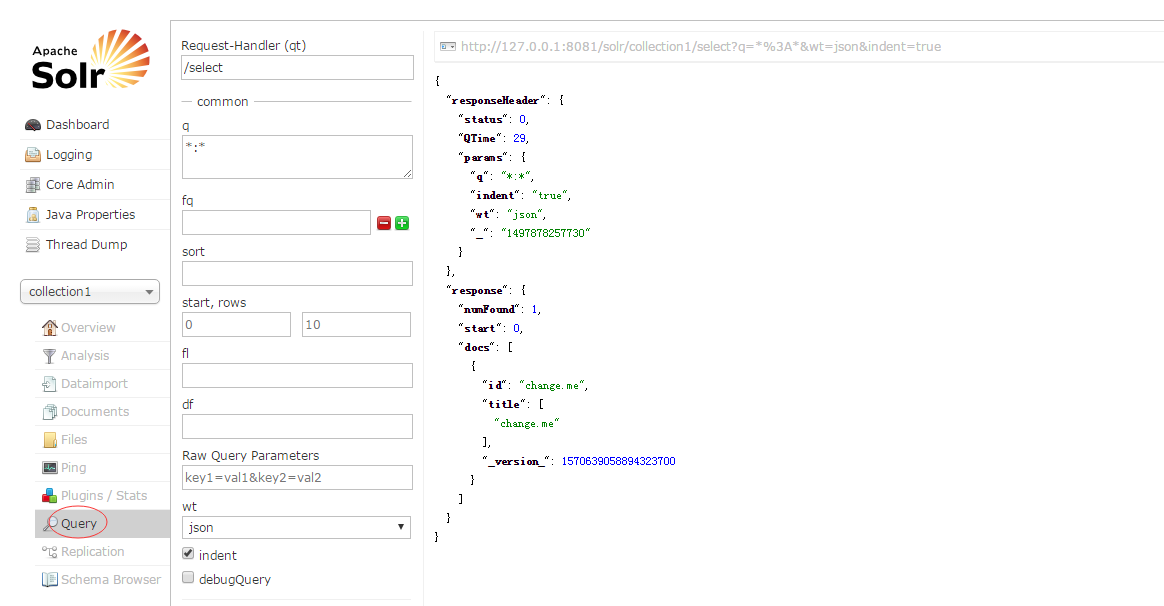


overwrite="true" ： solr在做索引的时候，如果文档已经存在，就用xml中的文档进行替换。

commitWithin="1000" ： solr 在做索引的时候，每隔1000（1秒）毫秒，做一次文档提交。

##### Query

通过/select执行搜索索引，必须指定“q”查询条件方可搜索。



##### 删除索引

方式一：根据id删除

|  |
| --- |
| <delete><id>change.me</id></delete>  <commit/> |

方式二：根据条件删除

|  |
| --- |
| <delete><query>title:change.me</query></delete>  <commit/> |
| 删除全部：  <delete><query>\*:\*</query></delete>  <commit/> |

# solrj使用

## solrj是什么

说明：用于访问solr的java客户端程序，提供了操作solr服务的增删改查方法。如下图：

应用服务器：

J2EE程序

solrj客户端策程序

搜索服务器：

solr服务

索引库

1.通过solrj访问solr服务

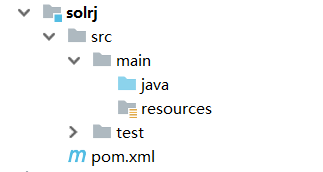
2.最终由solr完成索引库的操作

## 需求

使用solrj访问solr服务，完成索引的增删改查操作。

## 需求实现

### 创建项目



### 配置pom.xml，导入依赖包

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>cn.itheima</**groupId**>  <**artifactId**>solrj</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  <**packaging**>jar</**packaging**>   <**properties**>  <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  *<!-- solrj版本 -->* <**solrj.version**>4.10.3</**solrj.version**>  *<!-- log4j日志版本 -->* <**slf4j.version**>1.7.7</**slf4j.version**>  <**log4j.version**>1.2.17</**log4j.version**>  *<!-- jcl版本 -->* <**jcl.version**>1.7.6</**jcl.version**>  </**properties**>   <**dependencies**>  *<!-- solrj依赖 -->* <**dependency**>  <**groupId**>org.apache.solr</**groupId**>  <**artifactId**>solr-solrj</**artifactId**>  <**version**>${solrj.version}</**version**>  </**dependency**>  *<!-- log4j日志包 -->* <**dependency**>  <**groupId**>log4j</**groupId**>  <**artifactId**>log4j</**artifactId**>  <**version**>${log4j.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-api</**artifactId**>  <**version**>${slf4j.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-log4j12</**artifactId**>  <**version**>${slf4j.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>jcl-over-slf4j</**artifactId**>  <**version**>${jcl.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>jul-to-slf4j</**artifactId**>  <**version**>${jcl.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>    </**dependency**>  </**dependencies**>   </**project**> |

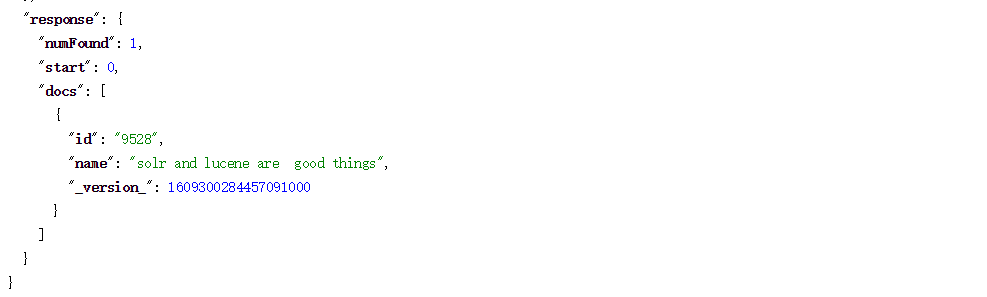
### 代码实现

#### 添加（更新）索引

说明：solr是根据id域执行索引的更新。先根据id域执行执行搜索，搜索到执行更新；搜索不到执行添加。

1. 建立HttpSolrServer对象，连接solr服务
2. 建立文档对象（SolrInputDocument）
3. 使用HttpSolrServer对象，执行添加（更新）
4. 提交

|  |
| --- |
| **package** cn.itheima.solrj;  **import** org.apache.solr.client.solrj.impl.HttpSolrServer; **import** org.apache.solr.common.SolrInputDocument; **import** org.junit.Test;  */\*\*  \* solrj入门案例程序  \*/* **public class** SolrjDemo {   */\*\*  \* 添加（更新）索引  \*/* @Test  **public void** addOrUpdateIndex() **throws** Exception{ *// 1.建立HttpSolrServer对象，连接solr服务  /\*\*  \* baseURL：http://127.0.0.1:8082/solr/  \*/* HttpSolrServer server = **new** HttpSolrServer(**"http://127.0.0.1:8082/solr/"**);  *// 2.建立文档对象（SolrInputDocument）* SolrInputDocument doc = **new** SolrInputDocument();  doc.addField(**"id"**,**"9528"**);  *//doc.addField("name","solr is a good things");   // 测试更新* doc.addField(**"name"**,**"solr and lucene are good things"**);  *// 3.使用HttpSolrServer对象，执行添加（更新）* server.add(doc);  *// 4.提交* server.commit();  } } |



#### 删除索引

##### 根据id删除

1. 建立HttpSolrServer对象，连接solr服务
2. 使用HttpSolrServer对象执行删除
3. 提交

|  |
| --- |
| */\*\*  \* 根据Id删除索引  \*/* @Test  **public void** deleteIndexById() **throws** Exception{ *// 1.建立HttpSolrServer对象，连接solr服务* HttpSolrServer server = **new** HttpSolrServer(**"http://127.0.0.1:8082/solr/"**); *// 2.使用HttpSolrServer对象执行删除* server.deleteById(**"9528"**);  *// 3.提交* server.commit();  } |

##### 根据条件删除

1. 建立HttpSolrServer对象，连接solr服务
2. 使用HttpSolrServer对象，执行删除
3. 提交

|  |
| --- |
| */\*\*  \* 根据条件删除索引  \*/* @Test  **public void** deleteIndexByQuery() **throws** Exception{ *// 1.建立HttpSolrServer对象，连接solr服务* HttpSolrServer server = **new** HttpSolrServer(**"http://127.0.0.1:8082/solr/"**); *// 2.使用HttpSolrServer对象执行删除* server.deleteByQuery(**"name:lucene"**);  *// 3.提交* server.commit();  } |

#### 查询索引

1. 建立HttpSolrServer对象，连接solr服务
2. 建立查询对象（SolrQuery）
3. 使用HttpSolrServer对象执行搜索，返回QueryResposne对象
4. 从QueryResponse对象中获取查询的结果集（SolrDocumentList）
5. 处理结果集

|  |
| --- |
| */\*\*  \* 查询索引  \*/* @Test  **public void** queryIndex() **throws** Exception{ *// 1.建立HttpSolrServer对象，连接solr服务* HttpSolrServer server = **new** HttpSolrServer(**"http://127.0.0.1:8082/solr/"**); *// 2.建立查询对象（SolrQuery）* SolrQuery sq = **new** SolrQuery(**"\*:\*"**);  *// 3.使用HttpSolrServer对象执行搜索，返回QueryResposne对象* QueryResponse queryResponse = server.query(sq);  *// 4.从QueryResponse对象中获取查询的结果集（SolrDocumentList）* SolrDocumentList results = queryResponse.getResults();  *// 5.处理结果集  // 5.1.实际搜索到的结果数量* System.***out***.println(**"实际搜索到的结果数量："**+results.getNumFound());  **for**(SolrDocument doc:results){  System.***out***.println(**"------------------华丽丽分割线--------------------"**);  System.***out***.println(**"id域："**+doc.get(**"id"**));  System.***out***.println(**"name域："**+doc.get(**"name"**));  }  } |

