**Lucene搜索入门**

**前言**

搜索即用给定的关键字（key）去内容库（contents）里找到key。Lucene的搜索大概原理是从contents里提取预定的关键字（preKey）做成一个索引库（indexTable）。

例：

内容库（contents）

数据库表product

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Title | price |
| 1 | 草鱼 | 广西来宾清水草鱼 | 12 |
| 2 | 大米 | 广西来宾小平阳东南小油粘米 | 3 |
| 3 | 苹果 | 陕西凤阳小妈咪苹果 | 9 |
| 4 | 花生 | 广西南宁十万大山野花生 | 99 |

索引库（IndexTable）[多对多规则，此表只是举例，真实生成规则复杂，也不是一张表]

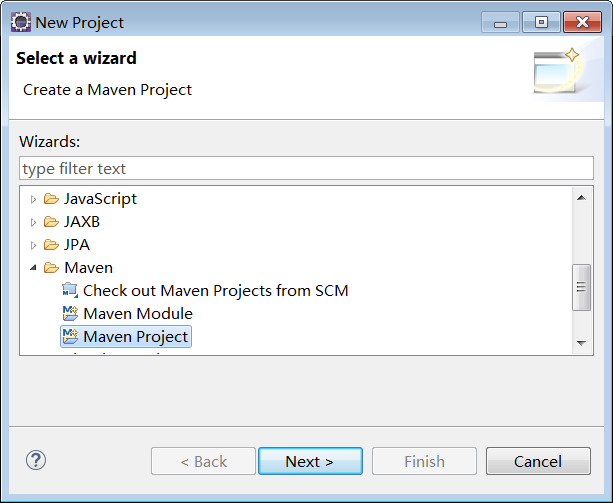
|  |  |
| --- | --- |
| PreKey | ids |
| 鱼，广西，来宾，清水，草鱼 | 1,2,4 |
| 米，大米，小平阳，东南，油粘米 | 2 |
| 果，苹果，陕西，凤阳，妈咪 | 3 |
| 花生，十万大山，野花生 | 4 |

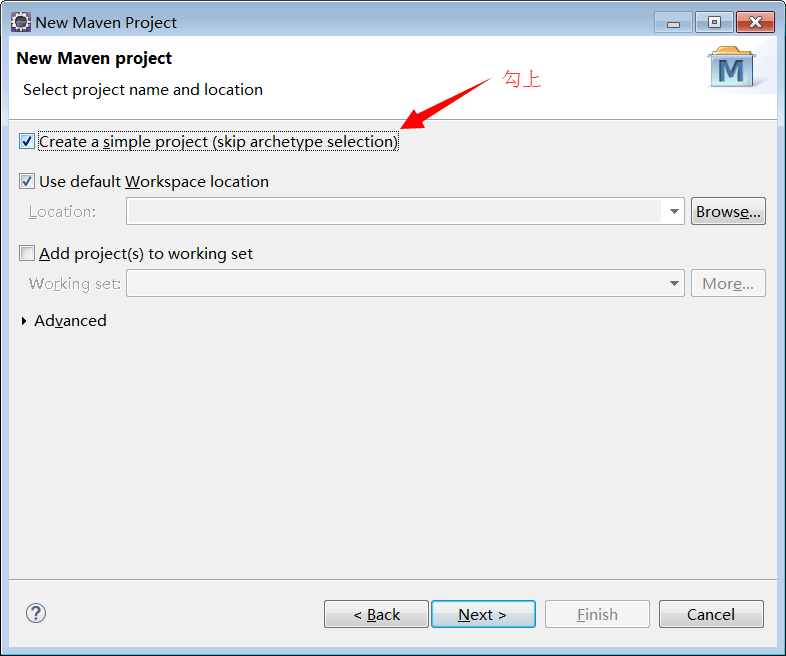
给定key=“广西”，则搜索出id为1,2,4的记录。

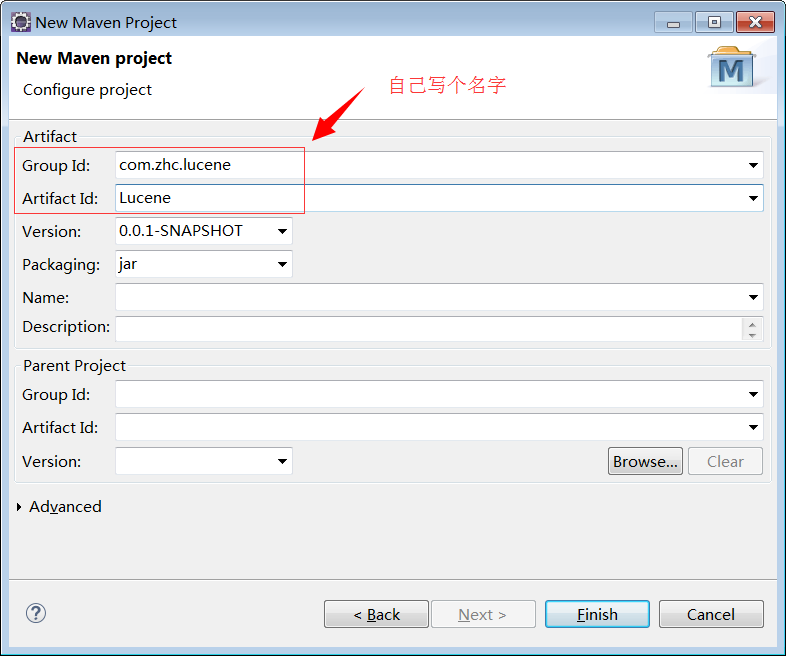
下面具体讲Lucen的操作。

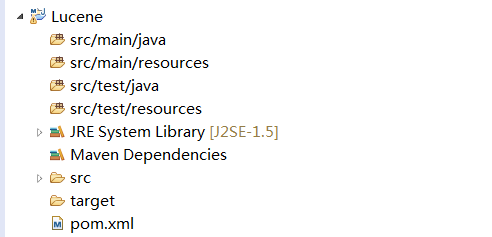
内容步骤：

1. 在eclipse中创建一个Maven工程

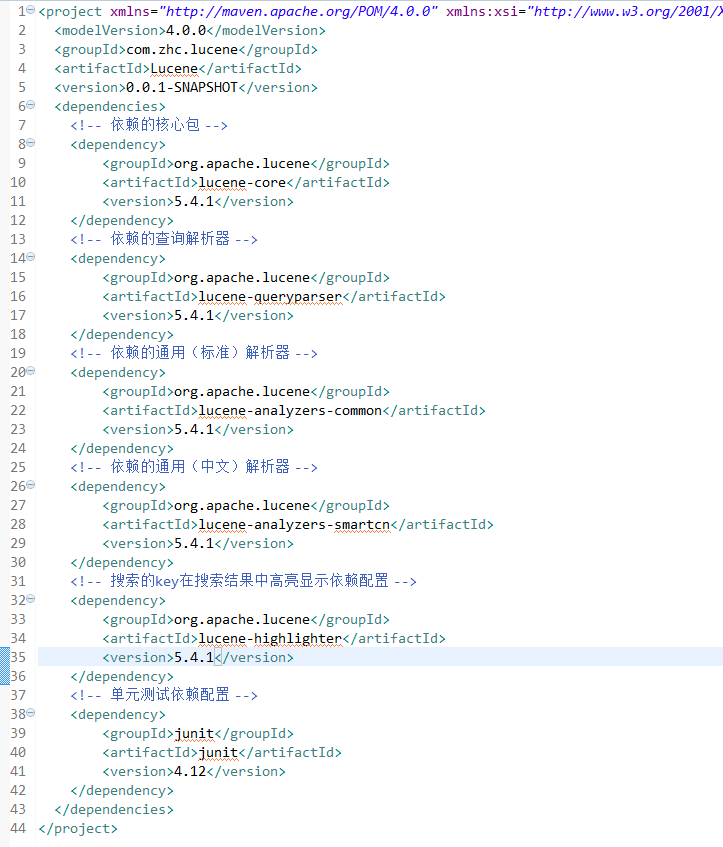






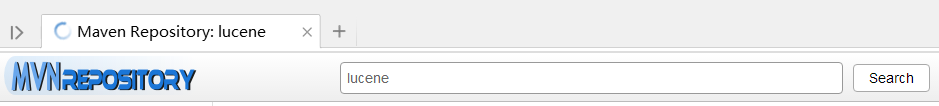
点击“finish”，工程创建完成如下：

1. 打开配置文件 pom.xml编辑里面的内容如下：

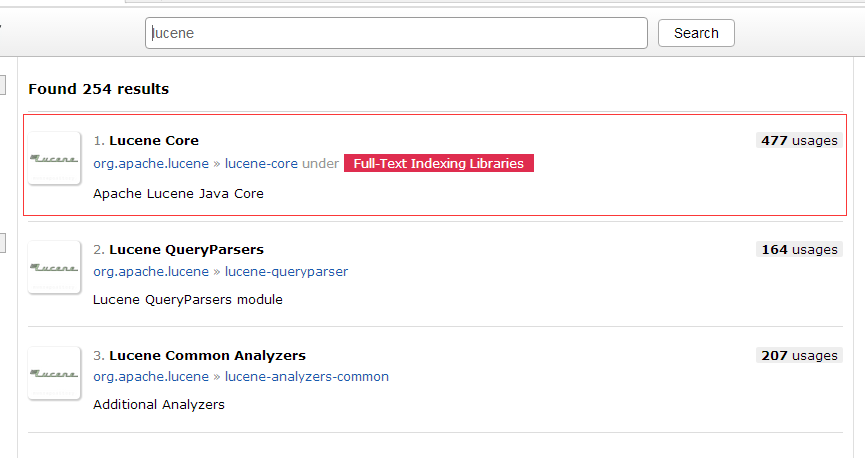


说明：

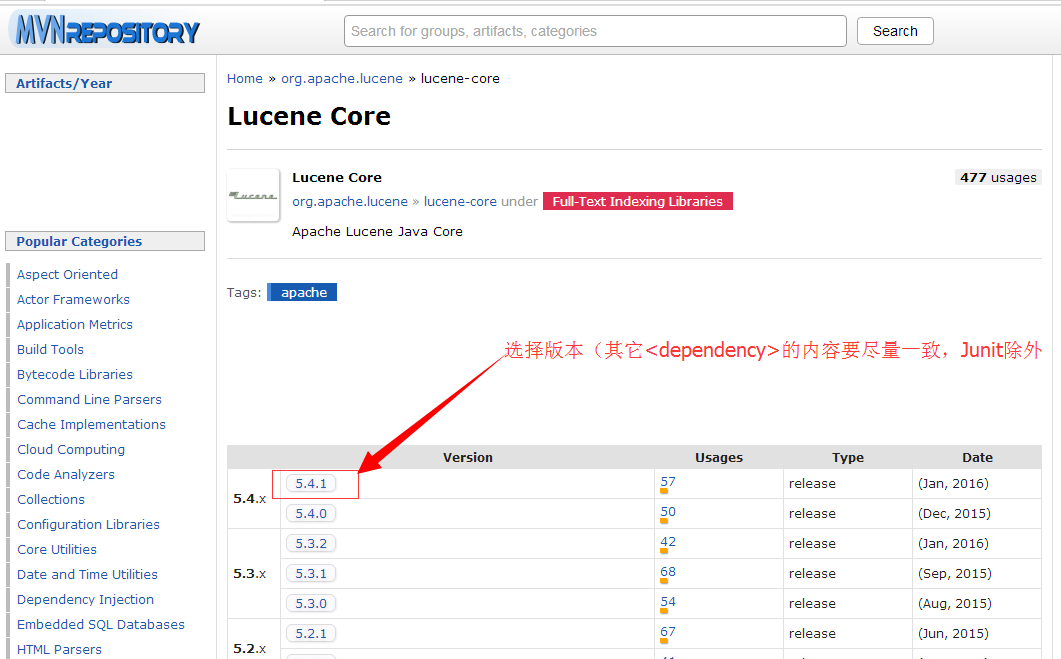
1. 增加了<dependencies>标签及其内部内容。
2. 每个<dependency>内容寻找方法,以第一个<dependency>内容为例，如下：
3. 打开<http://mvnrepository.com>
4. 在头条搜索框输入lucene搜索



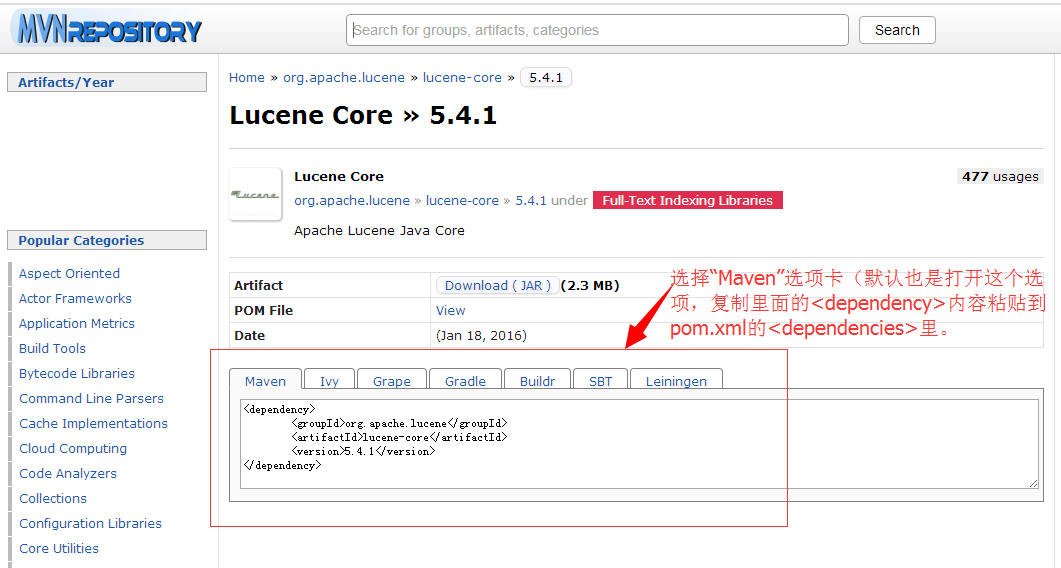
1. 搜索结果如下，选择含有lucene-core的第一个结果



点击进入



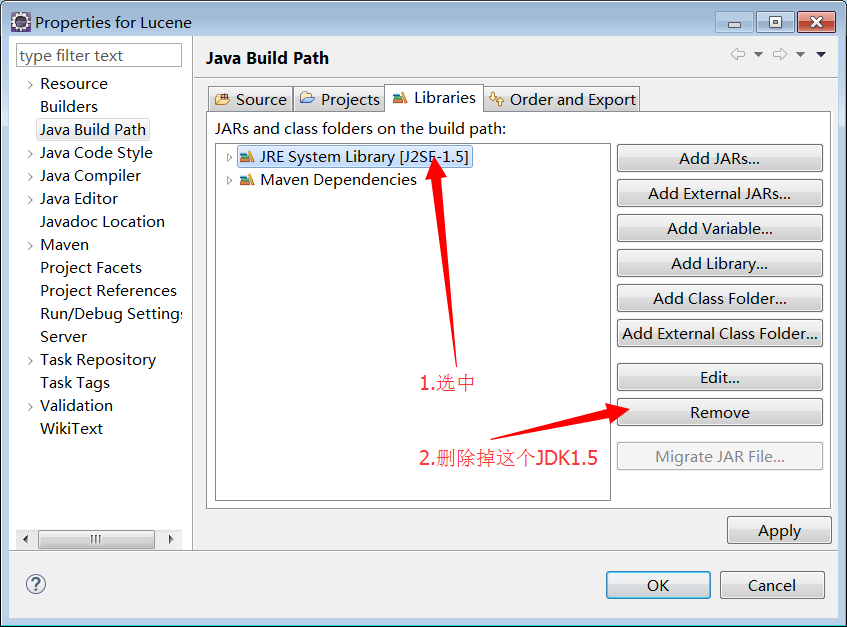
点击版本号进入：

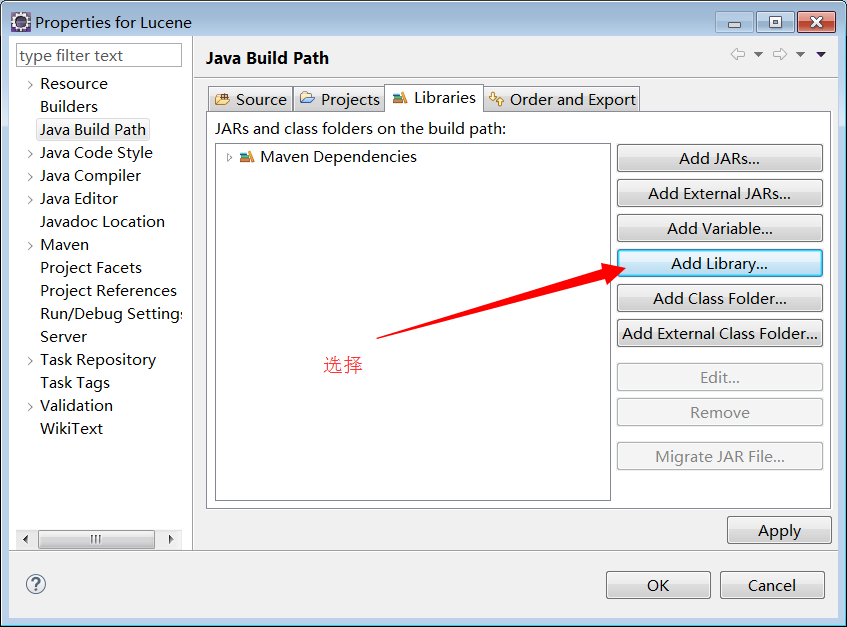


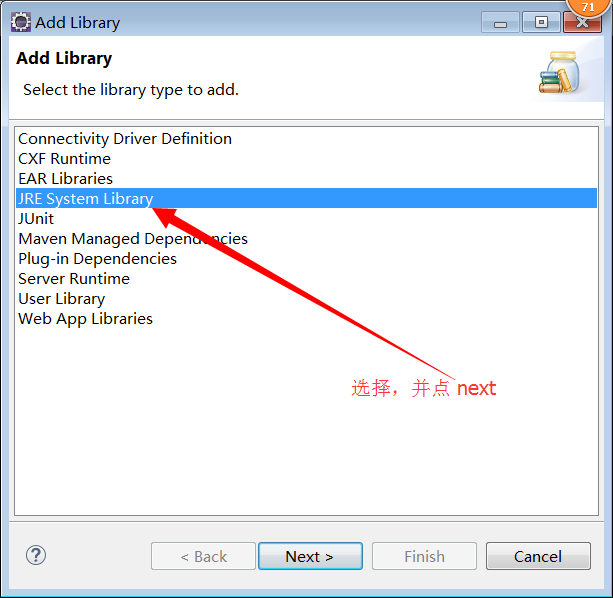
其它几个<dependency>的内容一样的寻找方法。注意版本（当前我用lucene 5.4.1）

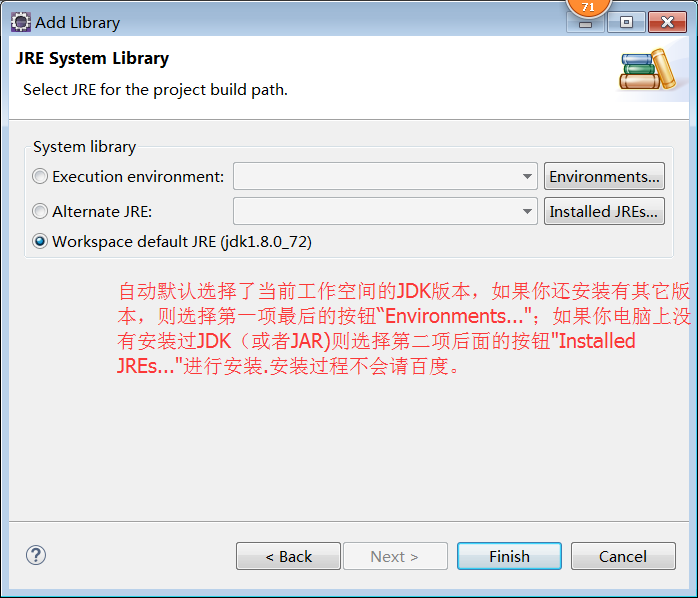
1. 修改工程默认的JDK版本，默认的是JDK1.5，修改成JDK1.7以上，我电脑安装JDK1.8所以修改成JDK1.8。（JDK1.8就是JDK8,同理JDK1.5是JDK5,叫法不同而已）

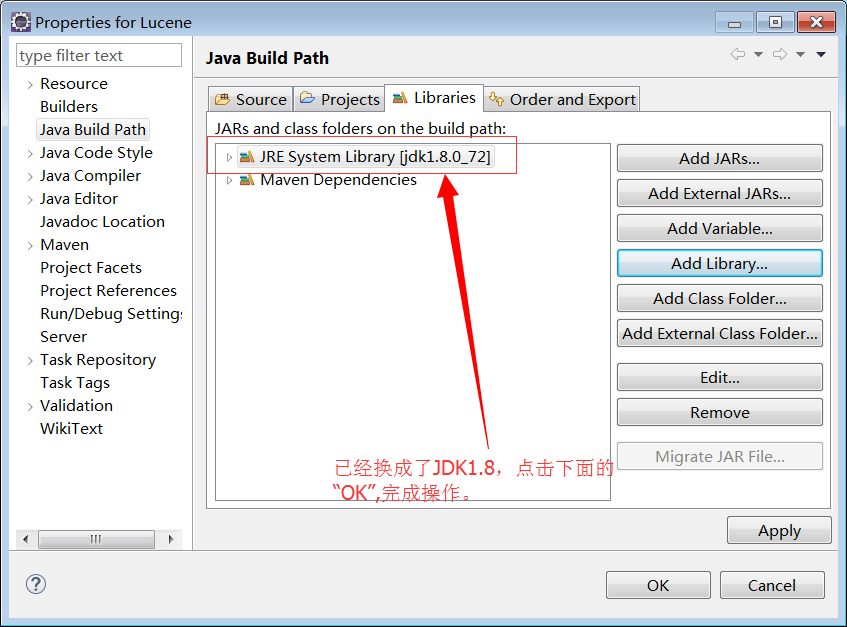
操作：对工程名“Lucene”右键—>Build Path—>configure Build Path…





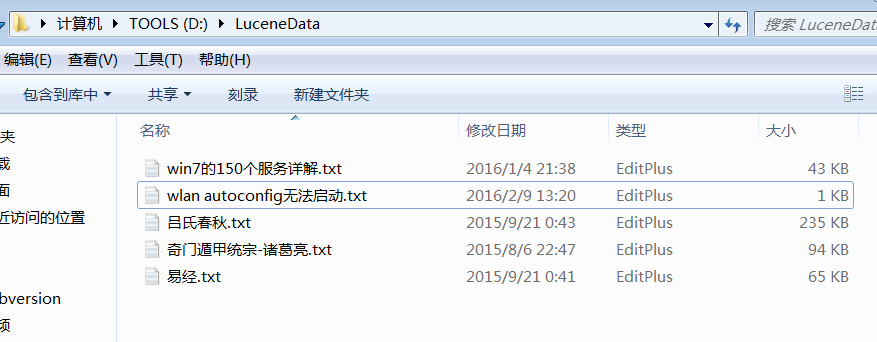






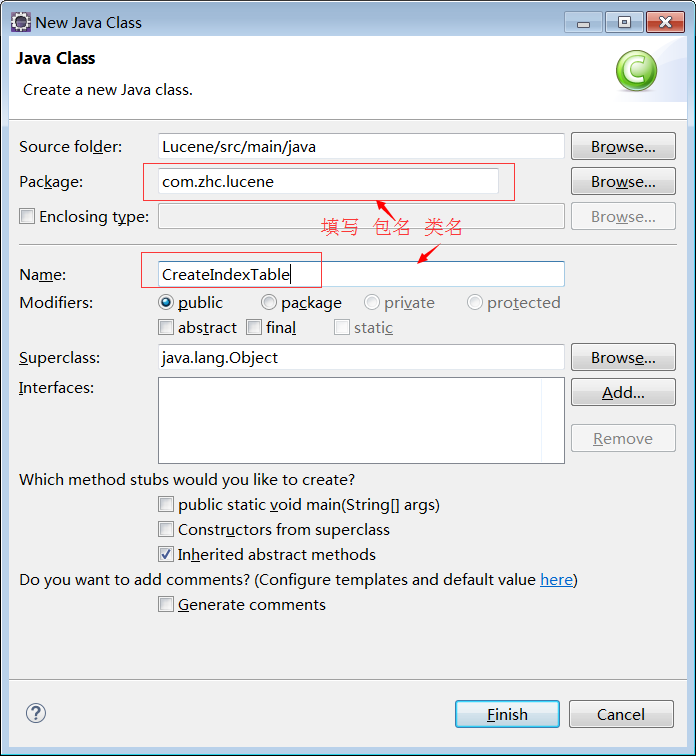
1. 准备数据（前言中的contents）

在D盘下的LuceneData文件夹里准备了5个文件。



1. 写java代码生成索引表（前方中的indexTable）

在工程的src/main文件夹下创建生成索引表的工具类



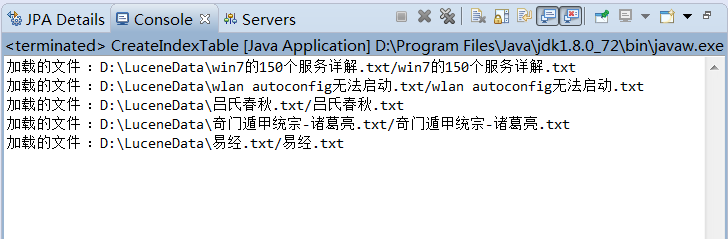
实现CreateIndexTable.java类

|  |
| --- |
| **package** com.zhc.lucene;  **import** java.io.File;  **import** java.io.FileReader;  **import** java.io.IOException;  **import** java.nio.file.Paths;  **import** org.apache.lucene.analysis.cn.smart.SmartChineseAnalyzer;  **import** org.apache.lucene.document.Document;  **import** org.apache.lucene.document.Field;  **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.store.FSDirectory;  **public** **class** CreateIndexTable {  **private** IndexWriter writer;//索引表生成对象，可以看成是一个有规则的输出流对象  /\*\*  \* 构造方法  \* **@param** contentDir 等待创建索引的内容路径  \* **@param** indexTableDir 索引表保存的路径  \* **@throws** IOException  \*/  **public** CreateIndexTable(String contentDir,String indexTableDir) **throws** IOException{  //读取内容  FSDirectory dir = FSDirectory.*open*(Paths.*get*(indexTableDir));  //创建一个标准分词器，所谓分词器，就是在全内容中选择出关键字（不适用于中文，所以我们选择建议一个中文的分词器）  //StandardAnalyzer analyzer = new StandardAnalyzer();  SmartChineseAnalyzer analyzer = **new** SmartChineseAnalyzer();  //创建索引分析器配置实例  IndexWriterConfig config = **new** IndexWriterConfig(analyzer);  writer = **new** IndexWriter(dir,config);  //读入等等创建索引的内容(因为是多个文件所以用listFiles,如果是只有一个文件 也不会错)  File []files = **new** File(contentDir).listFiles();  Document document = **null**;  **for** (File file : files) {  document = **new** Document();//创建一个等待分词的文档  //可以理解为给索引表添加一个content字段，读内容为从file文件内容里提取的关键字，不保存文档内容进表  document.add(**new** TextField("content", **new** FileReader(file)));  //为索引表增加fileName字段，内容为文件名，保存文件名进索引表  document.add(**new** StringField("fileName", file.getName(),Field.Store.***YES***));  //为索引表增加fullPath字段，内容为文件全路径，保存文件路径到索引表  document.add(**new** StringField("fullPath", file.getCanonicalPath(),Field.Store.***YES***));  //写索引表对象把定义好规则的文档写入索引表  writer.addDocument(document);  }  //写完索引表，关闭写对象实例。  writer.close();  }  } |

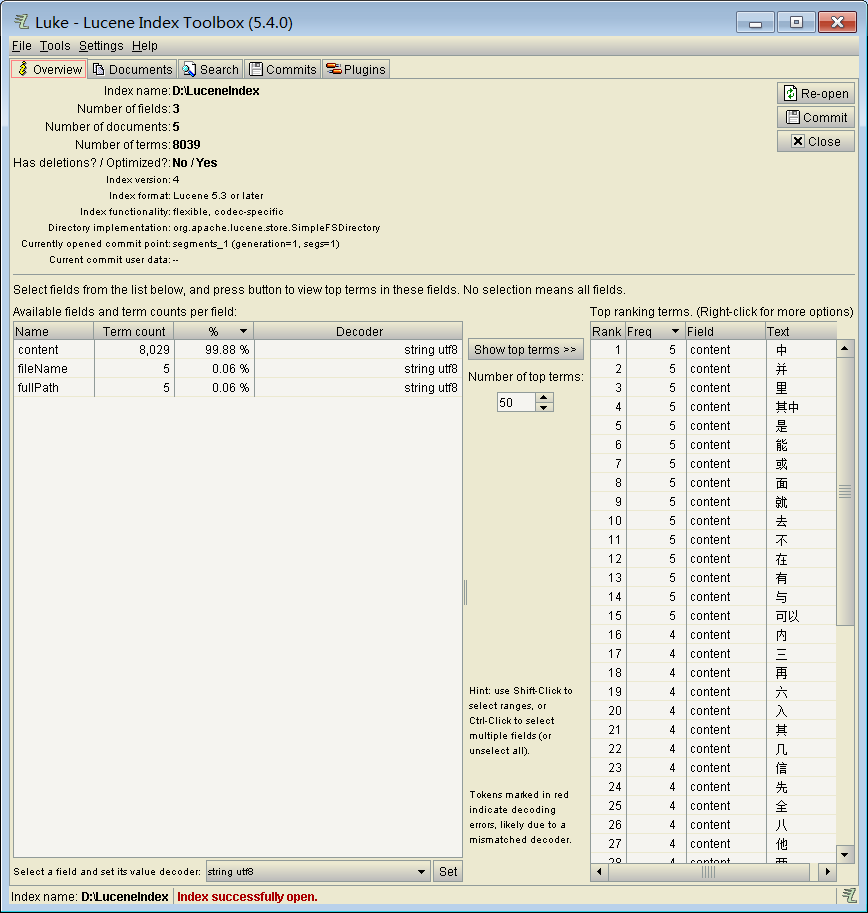
1. 运行生成索引表

为了看到加载过程，在for循环里加了一行代码：

System.***out***.println("加载的文件 ："+file.getCanonicalPath()+"/"+file.getName());



用luke查看生成的索引表如下：



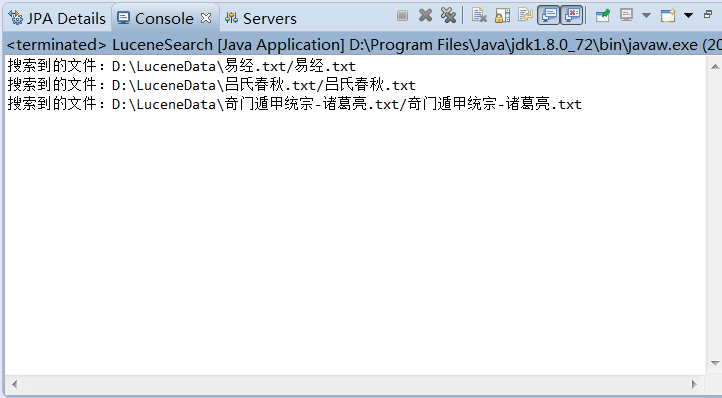
1. 写查询类LuceneSearch.java

|  |
| --- |
| **package** com.zhc.lucene;  **import** java.nio.file.Paths;  **import** org.apache.lucene.analysis.cn.smart.SmartChineseAnalyzer;  **import** org.apache.lucene.document.Document;  **import** org.apache.lucene.index.DirectoryReader;  **import** org.apache.lucene.queryparser.classic.QueryParser;  **import** org.apache.lucene.search.IndexSearcher;  **import** org.apache.lucene.search.Query;  **import** org.apache.lucene.search.ScoreDoc;  **import** org.apache.lucene.search.TopDocs;  **import** org.apache.lucene.store.FSDirectory;  **public** **class** LuceneSearch {  /\*\*  \* 构造器  \* **@param** indexDir 索引表的位置  \* **@param** key 查询关键字  \* **@throws** Exception  \*/  **public** LuceneSearch(String indexDir,String key)**throws** Exception{  //建立索引表文件路径  FSDirectory directory = FSDirectory.*open*(Paths.*get*(indexDir));  //建立读入索引文件对象  DirectoryReader reader = DirectoryReader.*open*(directory);  //生成搜索器,将用它来操作搜索  IndexSearcher searcher = **new** IndexSearcher(reader);      //生成分词搜索规则（用什么规则生成的索引表，就用什么规则去搜索索引表）  //StandardAnalyzer anaylzer = new StandardAnalyzer();//通用标准分词规则  SmartChineseAnalyzer analyzer = **new** SmartChineseAnalyzer();//中文分词规则 （建立索引表时就用的这个）  //根据对索引表搜索的字段和分词器创建搜索转换器  QueryParser parser = **new** QueryParser("content", analyzer);  //根据搜索关键字，获取搜索规则（底层实现算法，到此处还只是一些设置，还不是真正耗时的搜索工作）  Query query = parser.parse(key);    //开始搜索工作，根据以上设置的搜索规则，搜索出前30条数据。  TopDocs results = searcher.search(query, 30);  //显示搜索结果  Document doc = **null**;  **for** (ScoreDoc score : results.scoreDocs) {  //根据文件标识从搜索器里取得搜索到的文件  doc = searcher.doc(score.doc);  //显示搜索到的文件的路径和名字，通过索引表的字段获取。  System.***out***.println("搜索到的文件："+doc.get("fullPath")+"/"+doc.get("fileName"));    }  //关闭读入文件流  reader.close();  }  } |

在工程下的src/test/java文件夹里创建一个测试类LuceneSearchTest.java

|  |
| --- |
| **package** com.zhc.lucene;  **import** org.junit.Test;  **public** **class** LuceneSearchTest {    @Test  **public** **void** search(String[] args) {  String indexDir = "D:\\LuceneIndex";  String key = "正者";  **try** {  **new** LuceneSearch(indexDir, key);  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

搜索结果如下：



到此一个Lucene搜索就完全走通了，但要真实的工作，还要多学习扩展，下面再增加一些常用的Lucene应用知识。

1. 对索引表的搜索是前面所讲重点，但对索引表的增删除也很重要。多了解IndexWriter类的API,如下方法及其重载。

|  |
| --- |
| writer.addDocument(doc);  writer.deleteDocuments(arg0);  writer.updateDocument(arg0, arg1); |

1. 可以给索引表某个字段的特定值加减权值（被搜索到的机率）

实现：如更新fileName字段的权限代码修改如下：

|  |
| --- |
| //为索引表增加fileName字段，内容为文件名，保存文件名进索引表  StringField fileNameField =  **new** StringField("fileName", file.getName(),Field.Store.***YES***);  **if**(file.getName().contains("吕氏")){  fileNameField.setBoost(1.2f);//默认值为1，大于1则更优先被搜索到，小于1于反之。  }  document.add(fileNameField); |

1. 对索引表指定字体搜索

实现：创建query时则有所改变

Term t=new Term(col,key);//col为索引表字段名，key为搜索关键字

Query query=new TermQuery(t);//创建query

4)key可以是表达式（查看文档），

如: String key = “中国~”，和逻辑 ：与为AND,或为空格，…

1. 指定荐范围(对fileName搜索从a字符开始b结束)

TermRangeQuery query=new TermRangeQuery("fileName", new BytesRef("b".getBytes()), new BytesRef("c".getBytes()), true, true);

1. 指定数字范围（对字段为数字的才可以用数字范围，如id，本次数据没数字类型）

NumericRangeQuery<Integer> query=NumericRangeQuery.newIntRange("id", 1, 2, true, true);

1. 指定前缀（对fileName是a开头的）

PrefixQuery query=new PrefixQuery(new Term("fileName","a"));

1. 多条件(query的构造有所不同，多注意)

|  |
| --- |
| NumericRangeQuery<Integer> query1=NumericRangeQuery.newIntRange("id", 1, 2, true, true);  PrefixQuery  query2=new PrefixQuery(new Term("city","a"));  BooleanQuery.Builder  query=new BooleanQuery.Builder();  query.add(query1,BooleanClause.Occur.MUST);//MUST为和其它条件是或逻辑  query.add(query2,BooleanClause.Occur.MUST); |

1. 分页

searcher. search(query, 10);//10就是10条数据

1. 高亮

要两处改变

1. 生成添加样式对象（设置key中内容中为红色）

|  |
| --- |
| QueryScorer scorer=new QueryScorer(query);  Fragmenter fragmenter=new SimpleSpanFragmenter(scorer);  SimpleHTMLFormatter simpleHTMLFormatter=  new SimpleHTMLFormatter("<b><font color='red'>","</font></b>");  Highlighter highlighter=new Highlighter(simpleHTMLFormatter, scorer);  highlighter.setTextFragmenter(fragmenter); |

1. 将样式添加到结果

|  |
| --- |
| Document doc=is.doc(scoreDoc.doc);  String desc=doc.get("content");  if(content!=null){  TokenStream tokenStream=analyzer.tokenStream("content ", new StringReader(content));  System.out.println(highlighter.getBestFragment(tokenStream, content));  } |