## 文本预处理

为了更方便地提取XML里面的内容，采用Jsoup库对xml进行解析，可以直接提取出具有特定属性的标签及其内容。

我把提取之后的文档信息保存到PageInfo类的实例之中，然后再将其交给下一步处理。代码如下：

|  |
| --- |
| /\*  \* This function us to retrieve those contents from a <document> tag.  \* As some document node do not have specific childnodes, I use try-catch to avoid crash down  \*/  **public** PageInfo ReadDocNode(**int** index)  {  Element currentDocNode = allDocs.get(index);  PageInfo pageInfo = **new** PageInfo();  org.jsoup.select.Elements childElements = currentDocNode.children();  String keywords = " ";  **try** {  keywords = currentDocNode.getElementsByAttributeValue("name","keywords").get(0).attr("content");  }  **catch** (Exception e) {  // **TODO**: handle exception  keywords = " ";  }  String publishid = " ";  **try** {  publishid = currentDocNode.getElementsByAttributeValue("name","publishid").get(0).attr("content"); //publishid  }  **catch**(Exception e)  {  publishid = " ";  }  pageInfo.Set(  childElements.get(0).ownText(), //url  publishid,  currentDocNode.getElementsByTag("title").get(0).ownText(), //title  keywords,//keywords  currentDocNode.ownText() //content  );  **return** pageInfo; |

注意到给我们的XML文件里并没有包含时间信息，但是在URL里却包含了时间信息，所以专门写一个函数从url里提取出时间，并且用int类型存储，表示为20120101这种形式。

为了提取出url的时间，我考察了四种URL里出现日期的形式，写出了如下四种方法：

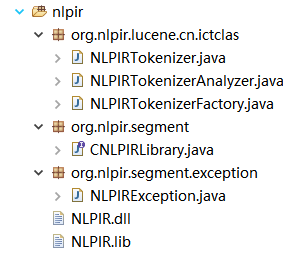
|  |
| --- |
| **boolean** success = **false**;  String[] dateStrRaw = \_url.split("/");  **try**  { //..../2012-1-1/xxxx.shtml  String[] dateStr = dateStrRaw[dateStrRaw.length-2].split("-");  date = Integer.*parseInt*(dateStr[0])\*10000 + Integer.*parseInt*(dateStr[1])\*100+Integer.*parseInt*(dateStr[2]);  success = **true**;  }  **catch** (Exception e)  {  // **TODO**: handle exception  success = **false**;  //System.out.println(url);  }  **if**(!success)  {  **try**  { // ...../20120101/xxxx.html  date = Integer.*parseInt*(dateStrRaw[dateStrRaw.length-2].substring(0,8));  success = **true**;  }  **catch**(Exception e)  {  success = **false**;  //System.out.println(url);  }  }  **if**(!success)  {  **try**  {  date = Integer.*parseInt*(dateStrRaw[dateStrRaw.length-1].substring(0, 8));  success = **true**;  }  **catch** (Exception e)  {  // **TODO**: handle exception  success = **false**;  //System.out.println(url);  }  }  **if**(!success)  {  **try**  { //..../2012-1-1/xxxxx/xxxx.shtml  String[] dateStr = dateStrRaw[dateStrRaw.length-3].split("-");  date = Integer.*parseInt*(dateStr[0])\*10000 + Integer.*parseInt*(dateStr[1])\*100+Integer.*parseInt*(dateStr[2]);  success = **true**;  }  **catch** (Exception e)  {  // **TODO**: handle exception  success = **false**;  System.***out***.println(url);  }  } |

## 分词器的使用

在这个项目里，我使用的是中科院分词工具 ICTCLAS。在ICTCLAS的官方的Github上有JAVA使用的示例：

<https://github.com/NLPIR-team/nlpir-analysis-cn-ictclas/tree/master/src>

将其下载下来，其目录结构如下：



其中NLPIR.dll和NLPIR.lib需要自行从官网上下载，放在其主目录下。然后CNLPIRLibrary中就会例化这个库，使其能够在NLPIRTokenizer类里面调用。

Tokenizer最重要的方法就是incrementToken(), 这个函数在到达词的尾部的时候返回true，否则返回false。

|  |
| --- |
| @Override  **public** **boolean** incrementToken() **throws** IOException {  **if** (buffer != **null** && buffer.length < current + 1) {  cbuffer = **null**;  buffer = **null**;  start = 0;  end = 0;  current = 0;  **return** **false**;  }  **while**(cbuffer == **null**||buffer.length==0) {  cbuffer = **new** StringBuffer();  **int** c = 0;  **while** ((c = input.read()) != -1)  {  cbuffer.append((**char**)c);  }  buffer = CNLPIRLibrary.***Instance***.NLPIR\_ParagraphProcess(cbuffer.toString(), 0).split("\\s");  }  clearAttributes();  **int** length = buffer[current].length();  end = start + length;  termAtt.copyBuffer(buffer[current].toCharArray(), 0, length);  offsetAtt.setOffset(correctOffset(start), correctOffset(end));  typeAtt.setType("word");  start = end;  current += 1;  **return** **true**;  } |

可以看到，这里面调用了库函数 ParagraphProces(), 这样可以读出段文字并且进行分词。这样子就定制了一个Tokenizer类。

除此之外，还需要定制TokenizerAnalyzer类和TokenizerFactory类。

## Lucene索引制作

现在我已经获得了页面信息类PageInfo，含有了一个页面的基本信息，又有了汉语分词的Lucene Analyzer，接下来就要对信息进行索引(Index), 索引后的信息会保存在文件里面，然后就可以进行查询。

首先需要实例化一个Analyzer:

nta = new NLPIRTokenizerAnalyzer("", 1, "", "", false);

第一个参数是词典路径，第二个参数是编码。接下来的三个是授权码，用户词典，用户词典引入方式。

我只需要考虑第一项和第二项。这里默认路径是空，经过尝试我发现这样子需要把Data文件夹放在主目录的上一层目录，这样分词器就能正常工作。

Lucene的存储单元是Document，Document包含数个Indexable Field，Field用于储存信息，有两种属性，Stored和Indexed。Indexed用于索引查询，会生成倒排索引。而Stored Field还会被保存在Document里。比如可以把关键词作为Indexed但没有Stored的Field，这样文档可以根据关键词搜索，但是搜索出来后却不能查看关键词。这里我把正文，标题设为既是Indexed也是Stored的Field。为了能根据日期进行搜索，排序等，我把日期存进了一个NumericDocValuesField 里。

但是根据官方文档，NumericDocValuesField仅仅可以用来索引，并未被储存，所以还要新建一个StoredField用来保存日期。

使用Document.add(Field)方法可以把一个Field加入Document，然后需要一个IndexWriter把生成的Document写入文件系统，也就是硬盘中。

IndexWriter的初始化需要参数Directory和IndexWriterConfig。Directory就是需要保存的目录，可以这样获得：

indexSavingDir = FSDirectory.open(FileSystems.getDefault().getPath(".", savingPath));

这样就可以在当前路径下加上savingPath路径创建目录。

IndexWriterConfig只需要一个Analyzer作为参数就可以初始化，于是我选用了之前实现的NLPIRAnalyzer。很显然这个Analyzer就是用于对Document里面的文本进行分词处理的。

于是我写出了如下的索引实现代码：

|  |
| --- |
| **public** **boolean** Init(String savingPath)  {  nta = **new** NLPIRTokenizerAnalyzer("", 1, "", "", **false**);  inconf = **new** IndexWriterConfig(nta);  inconf.setOpenMode(OpenMode.***CREATE\_OR\_APPEND***);  **try** {  indexSavingDir = FSDirectory.*open*(FileSystems.*getDefault*().getPath(".", savingPath));  indexWriter = **new** IndexWriter(indexSavingDir, inconf);  }  **catch** (IOException e) {  // **TODO**: handle exception  //Cannot open the directory  **return** **false**;  }  commonFieldType = **new** FieldType();  commonFieldType.setStored(**true**);  commonFieldType.setIndexOptions(IndexOptions.***DOCS\_AND\_FREQS\_AND\_POSITIONS***);  notIndexedType = **new** FieldType();  notIndexedType.setStored(**true**);  notIndexedType.setIndexOptions(IndexOptions.***NONE***);  **return** **true**;  } |

## 实现查询

索引文件创建完毕后，就可以进行查询。Lucene提供了Query类来进行查询。Query需要用QueryParser进行生成，QueryParser是org.apache.lucene.queryparser.classic包下面的QueryParser，可以用QueryParser(String field, Analyzer ana)进行初始化，然后通过QueryParser.parse(String key) 的方法就会生成一个query，表示对某个关键字在某个Field通过某个分析器（分词器）的查询。

一个查询可以由很多个查询组成，每个查询可以由不同的权重，这时候就可以用Query的一个继承类BoostQuery。BoostQuery由初始化函数BoostQuery(Query query, float boost)生成，boost越大，表示权重越高。

注意到我一开始采用了NumericDocValueField这个Field用来存储日期，这个Field可以进行RangeQuery，即RangeQuery(String Field, int lowerBound, int higherBound), 这样可以筛选出指定的日期之间的文档。

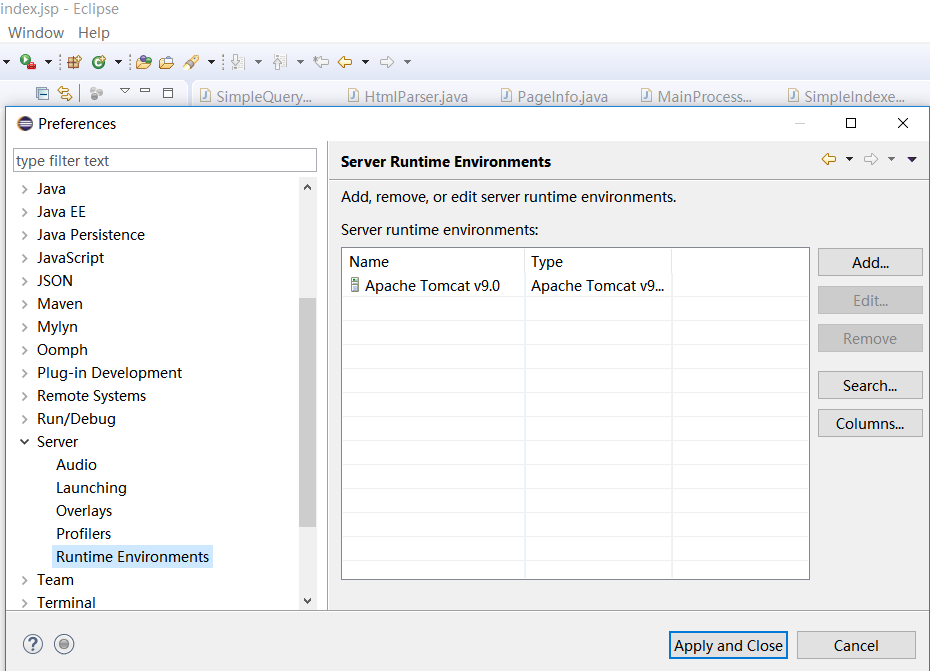
但是查询并不只有一个Query，可能是多个Query的叠加，所以可以使用BooleanQuery类。这个类可以把多个查询结合起来，并且指定哪些是“需要出现”，哪些是“必须不出现”。使用方法就是：

|  |
| --- |
| BooleanQuery.Builder combinedQueryBuilder = **new** BooleanQuery.Builder();  **for**(**int** i = 0;i<Fields.length;i++)  {  QueryParser newQueryParser = **new** QueryParser(Fields[i], nta);  Query newQuery = newQueryParser.parse(Keys[i]);  BoostQuery newBoostQuery = **new** BoostQuery(newQuery, Boosts[i]);  **if**(IsOccur[i])  {  combinedQueryBuilder.add(newBoostQuery, Occur.***MUST***);  }  **else**  {  combinedQueryBuilder.add(newBoostQuery,Occur.***MUST\_NOT***);  }  } |
|  |

这样就形成了一个完整的查询。

## Tomcat服务器加入项目

在window-preference-Server-Runtime Environments窗口里可以添加服务器，直接add…然后选择tomcat的安装路径就可以配置服务器。



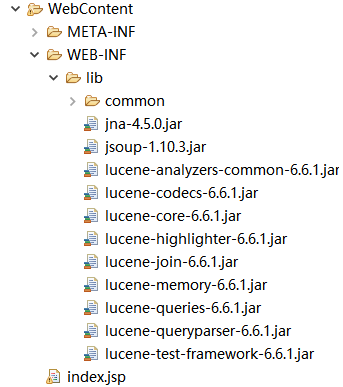
把服务器加入后，再在项目的Property-Server页面里就可以配置该服务器。

## JSP页面设计

**JSP页面加入项目**

在项目文件夹下面建立WebContent文件夹，用来保存jsp页面。默认访问的是index.jsp。为了使JSP页面能够调用原先写好的java文件，需要新建文件夹WEB-INF，以及子文件夹src。把源文件移动到该文件夹后，还需要右键把该文件夹“BuildPath-Use As Source Folder”,这样这个文件夹才能成为源文件夹，然后jsp文件才可以import里面的package中的类。

但是加入JSP页面后，发现尽管在Eclipse窗口中可以看到JSP页面首部import的库已经加入了，但是一旦“Run on server”后却总会报错。经过研究发现JSP页面里引用的包似乎需要放在WebContent/Web-INF/lib才可以读取，所以把jar包直接拷贝到文件夹下。再refresh整个项目，就不会报错了。



**中文字符作为get参数传递**

在某些浏览器中，无法把中文字符作为uri的一部分传递，所以我在jsp里使用了encodeURI()和decodeURI()方法来对中文字符编码和解码。

**利用javascript进行页面的初始化**

进行了一次搜索之后，我希望搜索结果的页面里的输入框依然保留着搜索的语句，所以在页面初始化的时候利用javascript的window.onload = function(){…}方法，把url的参数加入对应的html元件中。