Web 信息处理与应用实验一程序说明——聂雷海（PB16080377）

简单的搜索引擎实现

【实验内容】

通过 Java，使用 lucene 构建一个简单的搜索引擎。其中，数据由爬虫从 dbworld 网站爬取，再做文本预处理以及信息提取特征，利用 Lucene 建立索引文件并实现搜索接口。前端用 jsp 简单实现搜索功能。

【实验环境】

编程语言: Java SE 18.3 , jsp

编程环境: Win 10 Pro 1803 , IntelliJ IDEA Ultimate 2018.3.2

运行环境:

JRE: 1.8.0_152-release-1343-b26 amd64
JVM: OpenJDK 64-Bit Server VM by JetBrains

使用工具:

Apache Lucene 7.5.0
Apache Tomcat 8.5.35
IntelliJ IDEA Ultimate 2018.3.2
NLP 3.9 (Stanford)
Chrome 71

【实验步骤及方法】

主要步骤:

1. 先期基础学习:

   NLP 工具使用:

   学习并参考 https://stanfordnlp.github.io/CoreNLP/ 上 demo，懂得了 NLP(Stanford)使用。

```
public static void main(String[] args) {
  // set up pipeline properties
  Properties props = new Properties();
  // set the list of annotators to run
  props.setProperty("annotators", "tokenize,ssplit,pos,le
  // set a property for an annotator, in this case the co
  props.setProperty("coref.algorithm", "neural");
  // build pipeline
  StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
  // create a document object
  CoreDocument document = new CoreDocument(text);
  // annnotate the document
  pipeline.annotate(document);
  // examples

  // 10th token of the document
  CoreLabel token = document.tokens().get(10);
  System.out.println("Example: token");
  System.out.println(token);
  System.out.println();
```

Lucene 使用:

参考 http://www.lucenetutorial.com/lucene-in-5-minutes.html 并结合一些 blog 资料，了解了 Lucene 索引构建以及查找函数的编写。

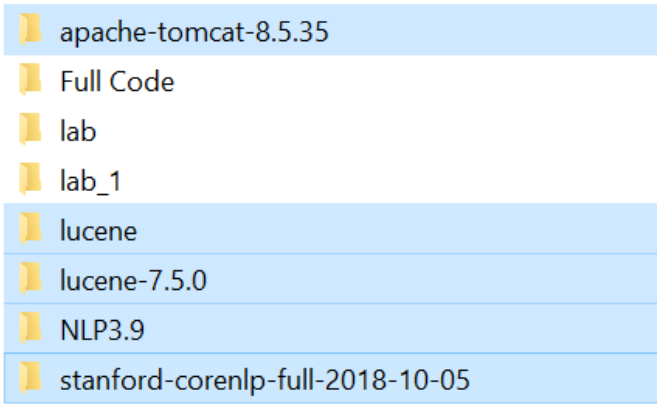
```
StandardAnalyzer analyzer = new StandardAnalyzer();

// 1. create the index
Directory index = new RAMDirectory();

IndexWriterConfig config = new IndexWriterConfig(analyzer);

IndexWriter w = new IndexWriter(index, config);
addDoc(w, "Lucene in Action", "193398817");
addDoc(w, "Lucene for Dummies", "55320055Z");
addDoc(w, "Managing Gigabytes", "55063554A");
addDoc(w, "The Art of Computer Science", "9900333X");
w.close();
```

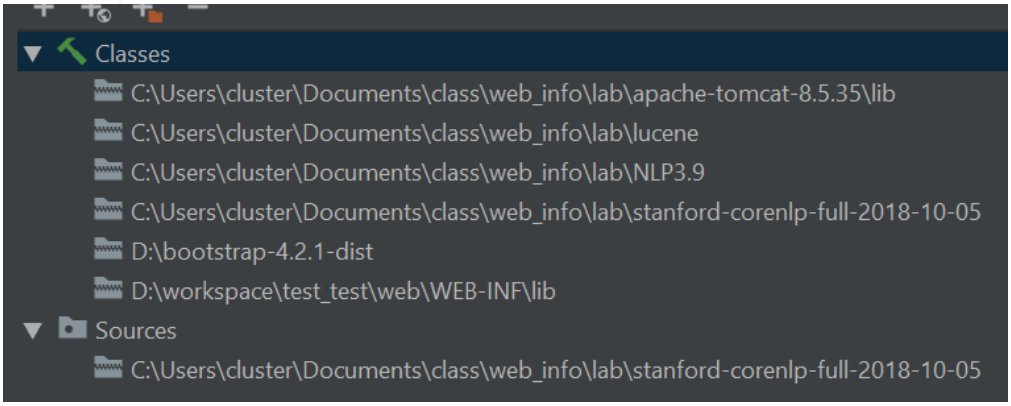
```
int hitsPerPage = 10;
IndexReader reader = DirectoryReader.open(index);
IndexSearcher searcher = new IndexSearcher(reader);
TopDocs docs = searcher.search(q, hitsPerPage);
ScoreDoc[] hits = docs.scoreDocs;
```

之后，参考了 FTP 上 jsp 资料，以及 tomcat 环境配置。

2. 配置环境:

下载 Lucene,CoreNLP 以及 tomcat 到本地，并添加到 lib(以及 bootstrap) lucene 是从 lucene-7.5.0(source code)中提取出来的我认为会使用的 jar 包， NLP3.9 为 CoreNLP 配套的 English 语言包。



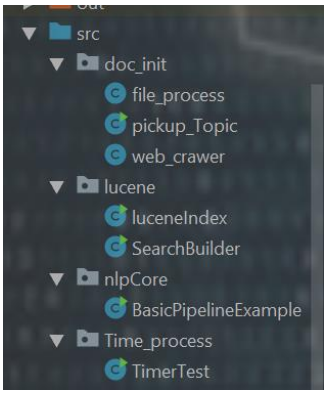


Ps:

为了项目配置便利，我一开始尝试了几天用 maven 管理，然而 maven+tomcat 始终都没有配置好。最后采用的是 Java EE (Web Application) 构建 Project.

3. code and debug

Code structure:



1. 爬虫的编写

对 dbworld 主页利用正则抓取对应的链接，即相应事件 url。

```
// function begins from begin(int) , num is file id with help that means the array size.
public static void doCycle(String[] sites, int begin, int num) {
    String content;
    int len = num;
    index = begin;
    for (int j = len - 1 - begin; j >= 0; j--) {
        System.out.print("id: " + index + " ");
        System.out.println(sites[j]);
        content = sendGet(sites[j]);
        content = test_true(content);
        WriteStringToFile(content, file_path, flag: 1, i: 0);
        //      WriteStringToFile(content, md_file_path, 0, index);
    }
}
```

doCycle 函数用于访问 Url,将静态文本写入本地文件。

```
static String sendGet(String url_base) {
    String result = "";
    BufferedReader in = null;

    try {
        URL realurl = new URL(url_base);
        URLConnection connection = realurl.openConnection();
        connection.setRequestProperty("accept", "*/*");
        connection.setRequestProperty("Accept-Language", "zh-CN,zh;q=0.8");
        connection.setRequestProperty("Cache-Control", "max-age=0");
```

sendGet 用于模拟 web 请求。

Test_true 通过下面的循环解决重定向问题。

```
    while (index_wrong != -1) {
        matcher = pattern_1.matcher(doc);
        if (matcher.find()) {
            site = matcher.group(1);
        } else return "error";
        doc = sendGet(site);
        index_wrong = doc.indexOf(wrong_str)
    }
```

2. 文件预处理:

通过多个正则匹配规则，去掉文本相应的 html 标签，特殊字符以及多余的无意义字符。

```
private static final String text_0 = "font-face";
private static final String text_1 = "<PRE.*?>(.*)</PRE>";
//    private static final String text_2 = "<p>(.*)</p>";
private static final String text_3 = "<HEAD>(?s).*</HEAD>";
private static final String text_4 = "<ul>(?s).*<ul>";
//  private static final String text_5 = "\\t|\\r|\\n";
private static final String text_5 = "\\t|\\r";
private static final String text_6 = "<a(?s).*?/a>";
private static final String text_7 = "\\+|\\||={2,}|-{5,}|\\*{3,}";
private static final String text_8 = "https:.*?\\s";
private static final String text_9 = "DBWorld Message";
private static final String text_10 = "http:.*?\\s";
private static final String text_11 = ",|\\.|#|$|@|!";
// maybe use text_16 rather than text_11 about processing comma.
private static final String text_12 = "www.*?(org|com|cn)";
private static final String text_13 = "\\{(.|\\s)*?\\}";
private static final String text_14 = "&.*?;";
private static final String text_15 = "/\\*(.|\\s)*?\\*/";
private static final String text_16 = "\\n[ ]\\n";
private static final String text_17 = "-{2,}";
private static final String text_18 = "\\*{2,}";
```

经过文本预处理后，就可以进行提取信息了。

```
String index[] = {text_9, text_0, text_5, text_4, text_3, text_6, text_7, text_8};
String index_2[] = {text_15, text_10, text_12, text_11, text_13, text_14};
Matcher matcher;
for (int i = 0; i < index.length; i++) {
    line = duplicate_replace(line, index[i]);
}

Pattern pattern = Pattern.compile(text_1);
matcher = pattern.matcher(line);
if (matcher.find()) {
    line = matcher.group(1);
}
line = duplicate_replace(line, html_tag);
line = duplicate_replace(line, html_tag_1);
for (int j = 0; j < index_2.length; j++) {
    line = duplicate_replace(line, index_2[j]);
}

line = duplicate_replace(line, html_tag);
line = line.replaceAll(text_17,  replacement: "-");
line = line.replaceAll(text_18,  replacement: "*");
line = line.replaceAll( regex: ":{1,}",  replacement: ":");
```

3. 特征提取:

对于时间的匹配，会先用正则匹配，在没有匹配的情况下，再考虑用 nlp 工具。

```
public static String year_val = "", month_val = "", date_from = "", date_to = "";
private static final String year = "(2018|2019)";
private static final String month = "(Jan(?:uary)?|Feb(?:ruary)?|Mar(?:ch)?|Apr(?:il)?|May|June|July|" +
        "Aug(?:ust)?|Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:ember)?)";
private static final String date = "(\\d{1,2})(?:th)?[ ]*-[ ]*(\\d{1,2})(?:th)?";
private static final String time_1 = month + " " + date + " " + year;
private static final String time_2 = date + " " + month + " " + year;
// don't consider over two months.
```

正则规则(匹配时间)

对于地点，直接使用 nlp 工具中 ner。鉴于实际文本中，地点与时间常常绑定在一起，因此在 nlp 发现地点后，会找[country]附近是否存在时间（当然这个时间，是在正则未完成时间匹配的情况下。）并且为了给出结果(即 0 与 1 的差别)，我在上述均无结果的情况下，使用极弱的条件来匹配以求得到结果。

```
String matcher_right = "ORGANIZATION";
int right_right_sum = 0, left_right_sum = 0;
String matcher_left = "CITY|STATE_OR_PROVINCE";
//      String matcher_special = "Setubal|Bengalaru";
int skip_blank_right = 0, skip_blank_left = 0;
if ((index = args.indexOf("COUNTRY")) != -1) {
    for (int j = index + 1; j < args.size(); j++) {
        if (skip_blank_right >= 2) {
            break;
        }
        String elem = args.get(j);
        if (right_right_sum > 2) {
            break;
        } else if (elem.equals(matcher_right)) {
            location.add(j);
            right_right_sum++;
        } else if (elem.equals("O")) {
            skip_blank_right++;
            continue;
        } else if (elem.matches(matcher_left)) {
            location.add(j);
        } else
            break;
    }
```

对于一些特别情况尚未做处理。比如某些地名会被识别为人名等。

```
List<String> sub_datelist_1 = Arrays.asList("DATE", "DATE", "DATE", "DATE");
List<String> sub_datelist_2 = Arrays.asList("DATE", "DATE");
```

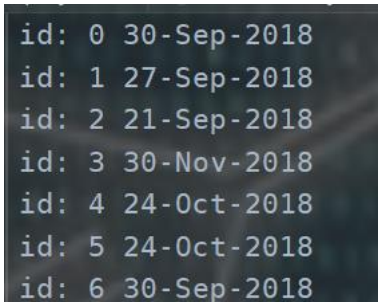
此为利用 nlp 工具 ner 判别时间的正则规则。规则 2 极弱，匹配广泛

对于主题，由于时间有限，我未用 nlp 工具 pattern 对 topics 分析。而是采取正则匹配，这样匹配出来，结果的查准率极高(我观测 30 组提取的，基本均正确提取 80%).但由于实际文本复杂的情况，查全率不高，

```
private static final String text_0 = "topics";
private static final String text_1 = "\\n.*?\\n";
private static final String text_2 = "\\n";
private static final String text_3 = "topics.*?\\n.*?topics";
private static final String text_star_dash = "\\n[ ]?(?:[\\*-])[ ]?(.*?)\\n";
public static ArrayList<String> topics_list = new ArrayList<>();
```

当然这也本身是正则的一个问题(很难用较少的正则规则来得到普适的结果，但过于繁琐的正则规则集又不易理解与再开发)
基于正则，主要是找到 “topics”后，从下一行开始匹配 topics.

4. Lucene 创建索引



由于我之前写入函数的处理，提取出来的本地文件均呈现如上图.

```
private static void init() throws IOException {
    //    ram_dir = new RAMDirectory();
    //   deleteAllFile(index_path);
    analyzer = new StandardAnalyzer();
    iwc = new IndexWriterConfig(analyzer);
    iwc.setOpenMode(OpenMode.CREATE_OR_APPEND);
    files = new File(origin_path).listFiles();
    fs_dir = FSDirectory.open(Paths.get(index_path));
    indexWriter = new IndexWriter(fs_dir, iwc);
    //ref link: https://www.jianshu.com/p/1f3ba892fc64

}
```

```
public static void main(String[] args) throws IOException {
    init();
    // deleteAllFile(index_path);
    luceneIndex.getField( rows: 1000);

    for (File file : files) {
        System.out.println("create index for " + file.getName());
        create_index(file);
        System.out.println(file.getName() + " ok! ");
    }
}
```

此为初始化

此为 main 函数

至于 create_index 函数，基于本文开头所参考的 Lucene demo，就不赘述了。而我所做的改变是将文件按格式读入数据(这个不是重点，仅是存储的 trick)

5. 查询函数

这里是基于 lucene 搜索 demo 编写。 使用 map 数据结构，通过 doc.get()得到相应字符串，再通过 map.put()将其存入 HashMap 中，最后存入 ArrayList 中。

```
String indexDir = "D:\\workspace\\test_test\\lucene\\index";
Directory directory = FSDirectory.open(Paths.get(indexDir));
IndexReader indexReader = DirectoryReader.open(directory);
IndexSearcher searcher = new IndexSearcher(indexReader);
QueryParser queryParser = new QueryParser(searchType, new StandardAnalyzer());
Query query = null;
ArrayList<Map<String, String>> results = new ArrayList<~>();
try {
    query = queryParser.parse(queryStr);
    long startTime = System.currentTimeMillis();
    TopDocs docs = null;
    if(sortType.equals("INDEXORDER")) docs = searcher.search(query, n: 1000, Sort.INDEXORDER);
    else  docs = searcher.search(query, n: 1000, Sort.RELEVANCE);
    long endTime = System.currentTimeMillis();
    System.out.println("查找" + queryStr + "所用时间: " + (endTime - startTime));
    System.out.println("查询到" + docs.totalHits + "条记录");
    //遍历查询结果
    for (ScoreDoc scoreDoc : docs.scoreDocs) {
        Document doc = searcher.doc(scoreDoc.doc);
        Map<String, String> map = new HashMap();
        map.put("from", doc.get("from"));
        map.put("to", doc.get("to"));
```

6. 搜索主页

web 主页设计我不大擅长，对前端也不大感兴趣。

DBWorld Search

搜索

搜索内容: 标题相关 ▾ 排序方式： 相关度优先 ▾

此为前端界面。

```
<p align="center">
    <font color="black" size="4">搜索内容: </font>
    <select name="searchType">
        <option value="subject">标题相关</option>
        <option value="topics">主题相关</option>
        <option value="location">地点相关</option>
    </select>
    <font color="black" size="4"> 排序方式:  </font>
    <select name="sortType">
        <option value="RELEVANCE">相关度优先</option>
        <option value="INDEXORDER">索引序优先</option>
    </select>
</p>
```

相关的代码主要参考 lab01.pdf 中给出的参考界面的某些设定，并且参考了 FTP 中 ppt 某些选中传值的操作。通过 post 方法，将搜索字符串传到 result.jsp

7. 搜索结果界面

我用了 bootstrap 包，简单地改了显示字符串的颜色。

这里代码的思路：返回搜索结果，同时还要处理分页，跳转，查询历史等信息。

这里用于获取查询需要准备的信息。

```
queryText = request.getParameter("query");
queryPage = request.getParameter("page");
```

将 index.jsp 输入字符串读取，并实现跳转。

这里体现搜索函数调用。

```
<input type = "text" name = "query" style = "..." value="<%= queryText %>"
<input type = "submit" style = "..." value = "搜索并获取第" style = "...">
```

```
ArrayList<Map<String,String>> docs = new ArrayList<>(),results=new ArrayList<>()
int Page = 1, size = 0;
SearchBuilder search = new SearchBuilder();
if( queryPage != null && queryPage.length()!=0)
    Page = Integer.parseInt(queryPage);
if(queryText != null && queryText != "") {
    try {
        docs = search.doSearch(queryText, searchType, sortType);
```

这里将搜索返回的结果写回到 html 内容中。

```
out.println("<font class=\"text-primary\"  size = \"3\">");
out.print("<a href=\"" + strUrl + "\">" + strSubject + "</a>");
out.println("</font>" + "<br>");
out.println("<font class=\"text-warning\"  size = \"1\">" + " <strong>从</strong>: " + s
out.println("</font>" + "<br>");
out.println("<font class=\"text-danger\"  size = \"1\">" + "<strong>截止时间</strong>: "
out.println("</font>" + "<br>");
out.println("<font class=\"text-success\"  size = \"1\">" + " <strong>地点</strong>: " +
out.println("</font>" + "<br>");
out.println("<font class=\"text-info\"  size = \"1\">" + " <strong>主题</strong>: " + st
```

【实验结果说明及演示】

给出实验过程中需要说明的问题，以及实验过程及结果中需要给出的示例截图。

DBWorld Search

搜索

搜索内容: 标题相关 ▾ 排序方式： 相关度优先 ▾

输入 china,点击搜索

末端记录搜索记录

实现功能： 搜索历史，分页展示

【实验总结】

说明自己实验中的亮点，并给出实验中的不足和需要改进的地方。

亮点:

对于时间，地点提取的考虑比较精细。

对于原始文档，文件预处理考虑正则处理较多，对文件理解较深。

不足与需要改进的部分:

1. 对于主题提取，做的比较粗糙，希望用 nlp pattern 再改进一下，提高查全率。
2. 把 Web 文件夹中注册登录的功能应当整合到搜索主界面来，我由于排版以及注册登录未实现本地存储等原因，导致没有做出来。
3. 前端有一些 bug.比如,选取某搜索选项后,搜索之后仍会跳到 第一个选项。虽然结果不影响，但会影响操作。这跟 jsp 不熟有关。

注意：实验所需提交的内容包括本文档，以及所有的代码文件，把它们打包后上传到 FTP 服务器，其中实验过程中建立的索引文件不需要提交，请同学们在打包的时候注意一下。