西南财经大学

Southwestern University of Finance and Economics

课程论文

学年学期：2018-2019 学年第二学期

课程名称：网络文本信息挖掘

论文题目：期中项目报告

学生学号：41623061

学生姓名：夏渝薇

学　　院：经济信息工程学院

年级专业：2016 级计算机科学与技术专业

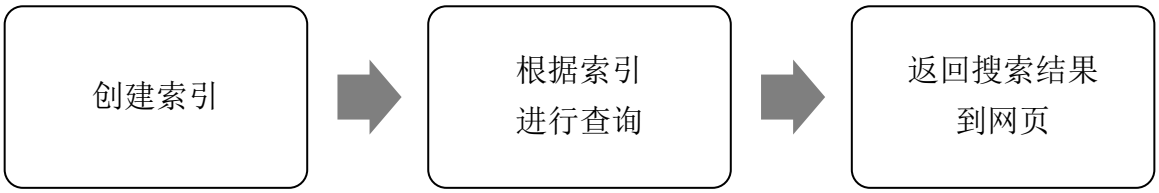
评语：
得　分： 评阅教师签字：　　年　月

《网络文本信息挖掘》期中项目报告

项目任务

做一个 JAVA WEB 项目，在网页上给出查询页面。根据查询检索 www 会议文档集和。搜索返回的结果条目应有高亮和 snippet。

程序流程图



项目中用到的 jar 包

- lecene-analyzers-common-5.5.0.jar
- lecene-analyzers-smartcn-5.5.0.jar
- lucene-core-5.5.0.jar
- lucene-highlighter-5.5.0.jar
- lucene-queryparser-5.5.0.jar
- pdfbox-app-2.0.15.jar

详细流程

1. **创建索引：**

通过 BuildIndex 类的 buildIndex 方法创建索引。创建索引过程中存在两个需要解决的问题：

1）文件的遍历和文件类型判断

www会议的文档集和呈树状结构 ,文件夹中包含文件夹和需要的pdf格式的会议文档，因此需要进行文件的遍历，判断文件是否存在以及文件是否是 pdf 格式， 依次找出所有会议论文 。

2）解析 pdf 格式文件

对于每一篇 pdf 格式文件，都需要进行解析，才能够获取 pdf 中的文本内容，因此利用 apache 提供的 pdfbox(pdfbox-app-2.0.15)处理 pdf。由于为了符合最终搜索结果的需要，需要保存三个域：分别是"contents"域，用于保存文本的主要内容（即从 ABSTRACT 开始的部分）；"title"域，用于保存文章的标题；"path"域，用于保存文本的路径。路径已知，因此我们需要获得的内容是两个方面：标题和文本主要内容 。我们通过找出最大字号的字来确定标题。步骤如下：

① 创建两个 PDDocument 对象，一个用于加载需要解析的 pdf 文件，另一个用于第⑥步中写入。

② 创建一个 PDFTextStripper 对象，利用该类的 getText()方法提取 pdf 中的全部文字内容，并保存在一个字符串中。

③ 找出字符串"ABSTRACT"或者"Abstract"在字符串中的位置，并从"ABSTRACT"或"Abstract"后的内容开始截取，作为用于检索的文本主要内容，放入名为"contents"的 field 中。

```
String str1="Abstract";
String str2="ABSTRACT";
int index=0;
int len=str1.length();
if(result[0].contains(str1)) {
    index=result[0].indexOf(str1);
    result[0]=result[0].substring(index+len);
}
else if(result[0].contains(str2)) {
    index=result[0].indexOf(str2);
    result[0]=result[0].substring(index+len);
}
```

④ 其次，由于 PDFTextStripper 并没有提供直接获得内容字号大小的方法，因此写了一个子类 GetCharFont，该类继承 PDFTextStripper，并重写其中的 writeString 方法，对于每一行字符串，该方法循环获取每个字的字号并将字号和该字符的

Unicode 编码分别保存在 list 中；同时，观察到标题中的每个词都是单独的一行（即一个 String），因此每处理一个 String，都在保存 Unicode 的 list 中加入一个空格，并在保存字号的 list 中加入一个-1 的值占位。

```
public void writeString(String string, List<TextPosition> textPositions) throws IOException {
    for(TextPosition text:textPositions) {
        list_fontsize.add(text.getFontSizeInPt());
        list_text.add(text.getUnicode());
    }
    list_text.add(" ");
    list_fontsize.add((float) -1.0);
}
```

⑤ 创建 GetCharFont 对象，先通过该对象调用 getText()方法获取需要解析的 pdf 的全部文本内容，并且由于标题一定会出现在最前面一页，因此设置 startPage 为 0，设置 endPage 为 2。

⑥ 创建 Writer 对象，并利用⑤中创建的 GetCharFont 对象调用 writeText 方法，通过 Writer 对象向①中的另一个 PDDocument 对象写入 startPage 到 endPage 的内容。writeText()方法每写入一行会调用 parse()方法，而 parse()方法会回调 writeString()方法对每一行的内容进行处理。由于已经重写了 writeString()方法，因此可以获得两个 list，分别是从 startPage 到 endPage 的全部文本内容的 Unicode 编码和每个字符的字号。

⑦ 从存储字号的 list 中找出第一个最大字号的位置和所有字号为最大字号的字符个数，再从存储 Unicode 编码的 list 中取出其实相应位置和对应长度，文章的标题就解析出来了。

```
for(int i=0;i<length_fontsize;i++) {
    if(list_fontsize.get(i)>max_fontsize) {
        max_fontsize=list_fontsize.get(i);
        length=1;
    }
    else if(list_fontsize.get(i)==max_fontsize|| list_fontsize.get(i)==(float)-1.0) {
        length++;
    }
    else if(list_fontsize.get(i)<max_fontsize) {
        break;
    }

}
```

创建索引过程中，有四篇文档无法解析，直接丢弃处理（四篇文档分为是：/www/www2018/ p1023-liu.pdf, /www /2012/proceedings/companion/p687.pdf,

/www/www2014/companion/p1037.pdf, /www /www2015/Global diffusion.pdf)。

至此，索引创建完毕。

2. 查询

首先，通过 Web 前端将用户输入的查询内容传给后端。利用 SearchIndex 类，根据用户提交的查询关键词，检索文档集和，并按照文档得分从高到低返回查询结果。返回的查询结果包含：1）文档标题；2）文档 highlights 和 snippets；3）文档路径。

搜索查询的过程中主要需要解决查询语句中包含多个词的问题。

搜索查询步骤如下：

① 先对输入的字符串进行切分，保存为一个字符串数组；同时创建一个对应检索位置的字符串数组。

```
String[] m = query.split( s: "\\s+");
int l=m.length;
for(int i=0;i<l;i++) {
    System.out.println(m[i]);
    m[i] = "contents";
}
```

② 创建 IndexReader 对象，根据指定路径读取索引；创建 Analyzer 对象，创建一个解析器。

③ 创建一个 Query 对象，使用 MultiFieldQueryParser. parse()实例化，产生一个组合查询 q。

```
Query q = MultiFieldQueryParser.parse(query.split( s: " "), m, textAnalyzer);
```

④ 创建 IndexSearcher 对象 is，调用 search()方法对上面得到的查询 q 进行查询，得到一个 TopDocs 对象 topDocs。

```
TopDocs topDocs=is.search(q, Integer.MAX_VALUE);
```

⑤ 创建一个字符串数组 highlights，用来存储高亮显示的文本；创建一个 Document 对象数组 doc，用来存储 topDocs 中对应的文档。

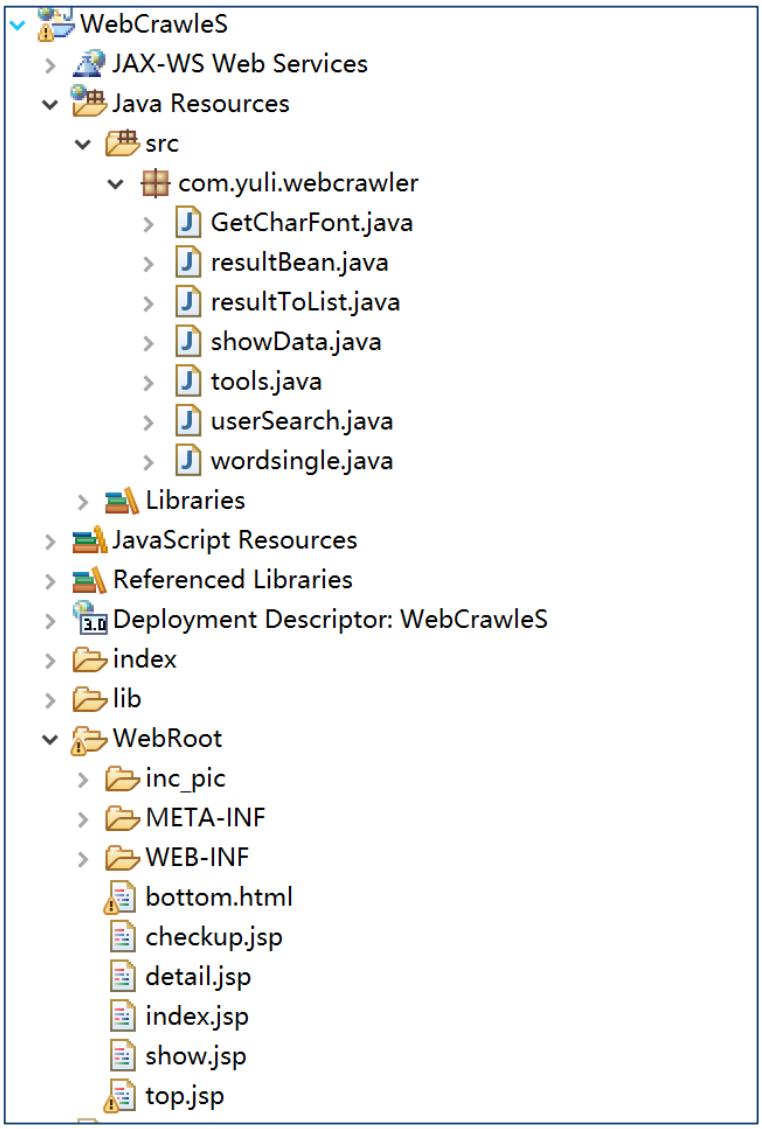
```
String highlights[] = highlighter.highlight( field: "contents", q, is, topDocs,  maxPassages: 3);
int length = highlights.length;
Document[] doc = new Document[length];
sdoc=topDocs.scoreDocs;
number=sdoc.length;
for(int i=0;i<number;i++) {
    System.out.println("sdoc:"+i+":"+sdoc[i]);
}
for(int i=0;i<number;i++){
    doc[i] = is.doc(sdoc[i].doc);
}
```

⑥ 根据 doc 和 highlights 将每篇文档依次保存在一个 ArrayList 中，返回数据给前端。

```
public static List<String[]> getSnippets(Document[] doc,String highlights[]) {
    List<String[]> snippets=new ArrayList<~>();
    for (int i=0;i<doc.length;i++) {
        String[] snippet=new String[3];
        snippet[0]=doc[i].get("title");
        snippet[1]=highlights[i];
        snippet[2]=doc[i].get("path");
        snippets.add(snippet);
    }
    return snippets;
}
```

3. 服务器构建及测试

1) 服务器框架

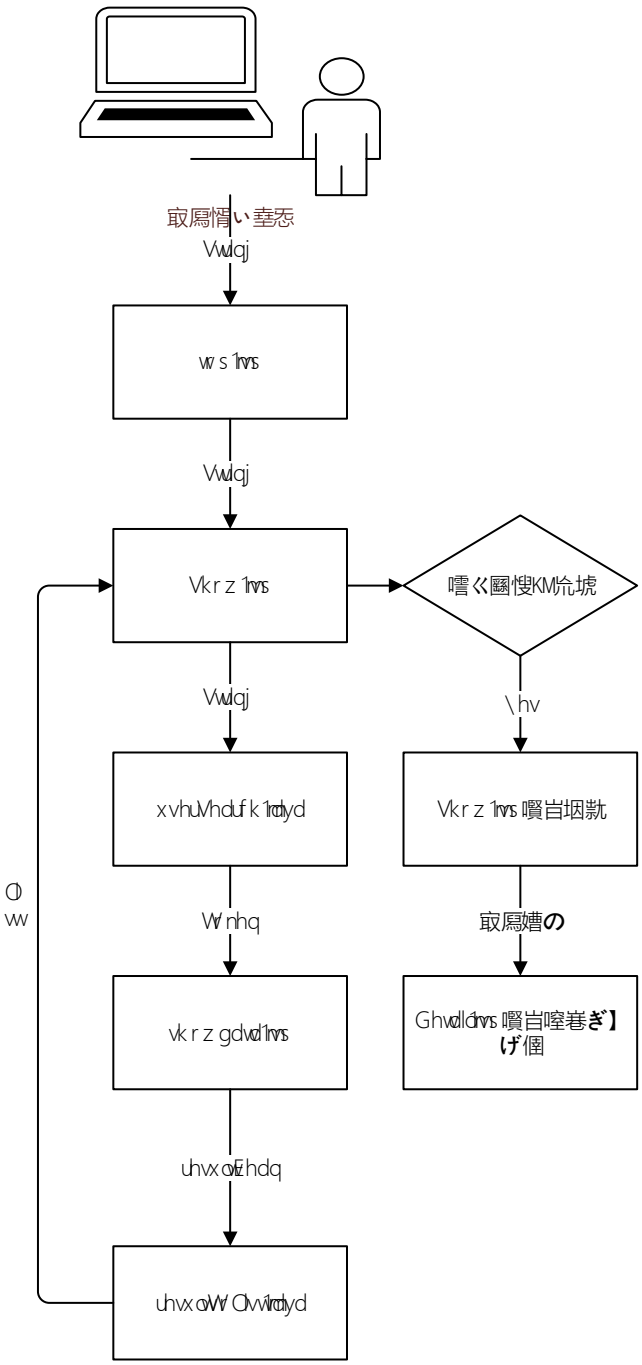


2) 各类接口及页面功能

GetCharFont.java	将 pdf 文档转换为字符流以供读取
resultBean.java	是服务器内部传输数据的基本单元，该类的对象包括文章的地址、文章的标题、文章的摘要以 highlight 字段
resultToList.java	将结果转换为 List，方便向客户端传递
showData.java	服务器的主要功能，即对用户的输入，根据已经生成的缩影查询并且返回结果

tools.java	提供编码的转换，主要用于处理非法输入，提高服务器的鲁棒性。
userSearch.java	用于对用户的输入进行处理，去掉非法字符以及停用词，生成可供 showData 类处理的对象
wordsingle.java	仅用于测试
文件夹 index	生成的所有索引存储于此
bottom.html	网站页面的底部，无功能
top.jsp、index.jsp	网站的初始查询页面及网站页面的顶部，用于处理用户的搜索内容
show.jsp	展示搜索结果，包括 url 链接、摘要、高光、结果总条目等
detail.jsp	以 pdf 的方式显示文章内容，支持 pdf 的下载

3) 后台运行流程



4) 运行结果显示

① 搜索关键词“data”

"data"的搜索结果有：1184个！

The Information Workbench as a Self-Service Platform for Developing Linked Data Applications

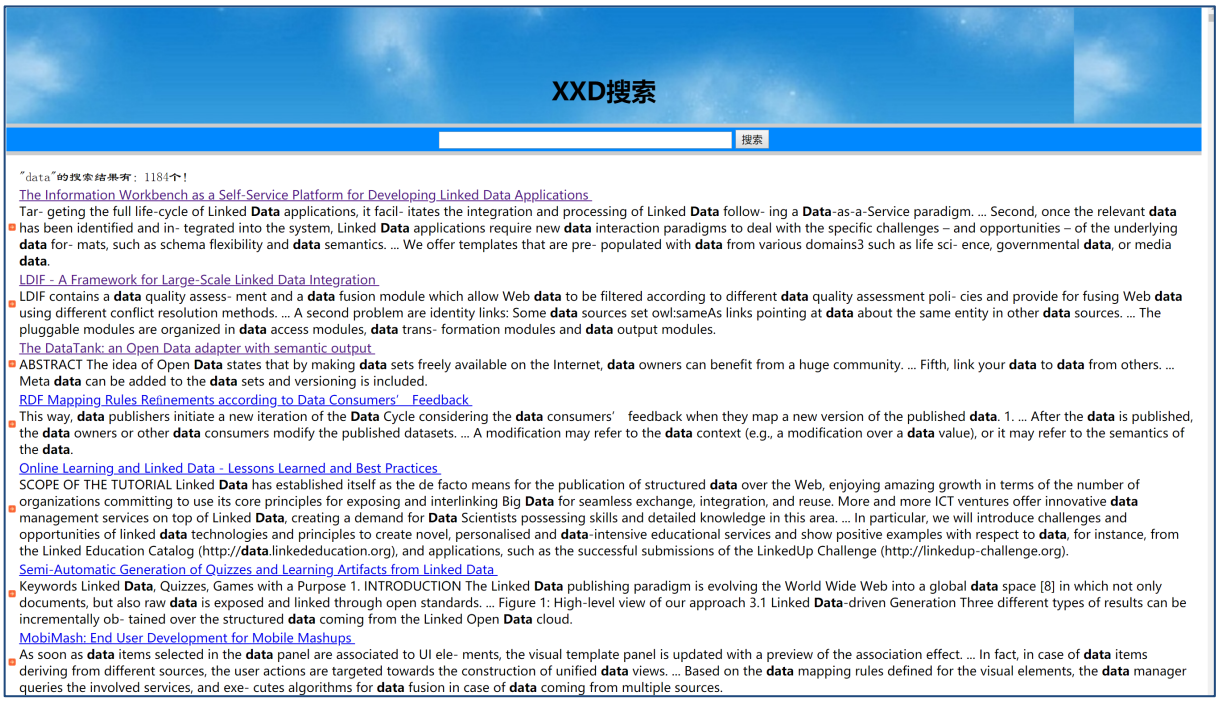
Tar- geting the full life-cycle of Linked **Data** applications, it facil- itates the integration and proces
- has been identified and in- tegrated into the system, Linked **Data** applications require new **data** in **data** for- mats, such as schema flexibility and **data** semantics. ... We offer templates that are pre- p **data**.

LDIF - A Framework for Large-Scale Linked Data Integration

- LDIF contains a **data** quality assess- ment and a **data** fusion module which allow Web **data** to be f using different conflict resolution methods. ... A second problem are identity links: Some **data** sou pluggable modules are organized in **data** access modules, **data** trans- formation modules and **dat**

The DataTank: an Open Data adapter with semantic output

- ABSTRACT The idea of Open **Data** states that by making **data** sets freely available on the Internet, Meta **data** can be added to the **data** sets and versioning is included.



② 点击返回的第一篇文章

RDF Mapping Rules Refinements according to Data Consumers' Feedback

Anastasia Dimou
anastasia.dimou@ugent.be

Miel Vander Sande
miel.vandersande@ugent.be

Tom De Nies
tom.denies@ugent.be

Erik Mannens
erik.mannens@ugent.be

Rik Van de Walle
rik.vandewalle@ugent.be

Ghent University – iMinds – Multimedia Lab
Gaston Crommenlaan 8/201
9050 Ghent, Belgium

ABSTRACT

The missing feedback loop is considered the reason for broken Data Cycles on current Linked Open Data ecosystems. Read-Write platforms are proposed, but they are restricted to capture modifications after the data is released as Linked Data. Triggering though a new iteration results in loosing the data consumers' modifications, as a new version of the source data is mapped, overwriting the currently published. We propose a prime solution that interprets the data consumers' feedback to update the mapping rules. This way, data publishers initiate a new iteration of the Data Cycle considering the data consumers' feedback when they map a new version of the published data.

1. INTRODUCTION

Nowadays, more and more data owners map their data to a corresponding RDF representation before they publish and interlink them with other Linked Data. After the data is published, the data owners or other data consumers modify the published datasets. While this is in compliance with the envisaged Read-Write Web and Government Linked Open Data life cycle[1] , Semantic Web compliant systems neither intuitively process data consumers' feedback, nor properly propagate the modifications over different serialisations of the original data. Most current solutions that publish Linked Data from other source formats focus on providing the RDF data model, dissociating the resulting representation from its original source. As a result, modifications over the triplified data are not propagated to the mapping rules and to the source data, as their RDF representation is handled independently of their source. Thus they are overwritten every time a new version of the source data is mapped to RDF.

We propose a solution that considers the data consumers feedback to refine the mapping rules. Provenance information is used to identify the mapping rules that generated the triples, and thus the data in their original serialisation. The modifications over the triplified data are captured as they are translated to refinements over the corresponding mappings definitions. This way, the next time a new version of the same data is mapped, namely a new Data Cycle is initiated, the modifications applied to the RDF representation of the data remain unaffected.

2. INCORPORATING USER FEEDBACK

In our approach, the mapping rules from different source files to their RDF representation are defined using the RDF Mapping Language (RML, [1]). RML is a generic language for mapping heterogeneous structured data formats to RDF[2]. An RML mapping consists of several Triples Maps. A *Triples Map* is a rule that defines a number of RDF triples to be generated from a source file and is identified by a URI. We output all the triples generated by a Triples Map as *quads* [3], extending the triple with an extra context element. This allows us to uniquely identify each triple, by adding a URI to it. With PROV[4], we describe the relation between each generated triple and its generating Triples Map, and thus the original file defined in its *Logical Source*. In the following example, the *GeneratedTriple* (3) is an entity generated from a non-RDF source file (7), mapped according to the *MappingProcedure* (4) and executed by the *dataOwner* (5). The *MappingProcedure* refers to the corresponding *TriplesMap*. The source file (12) is specified at the *Logical source* of the *Triples Map*.

```
1  @prefix prov:  <http://www.w3.org/ns/prov#>.
2  @prefix :  <http://example.com/>.
3  :GeneratedTriple a prov:Entity;
4     prov:wasGeneratedBy :MappingProcedure;
5     prov:wasAttributedTo :dataOwner;
6     prov:hadPrimarySource
7        <http://example.org/sourceData.xml>.
8  :MappingProcedure a prov:Activity;
9     prov:wasAssociatedWith :dataOwner;
10    prov:used   <http://example.org/sourceData.xml>;
11    prov:generated :GeneratedTriple.
12 <http://example.org/sourceData.xml> a prov:Entity.
13 :dataOwner a prov:Agent.
```

Next, R&Wbase [2] can track all the changes made to the generated triples. R&Wbase is an interpretation layer built on top of SPARQL endpoints and follows the principles of distributed version control. It stores different deltas in a single graph, allowing simple resolving of different versions