北京郵電大學

Beijing University of Posts and Telecommunications

Information Retrieval & Extraction 课程作业
面向动漫御宅族的搜索引擎设计

院　　系	网研院&信通院	
小组成员	高楚阳	2016111441
	杨雪	2016111434
	封依萌	2016010259
	孙传昊	2016110324
时　　间	2016 年 7 月 23 日	
指导老师	李　　蕾	

目录

一、项目简介

在此次课程设计中，出于完成基本要求与实用性两方面考虑，我们决定制作一个面向御宅族的搜索引擎作为信息检索抽取系统的课程设计作品。

我们引用了知名二次元资讯网站—萌娘百科（https://zh.moegirl.org/Mainpage）的数据库进行了此次试验。

图 1 萌娘百科

本次实验中，我们针对萌娘百科网站上的内容进行了搜索，使用了 saber 等常用动漫角色作为关键词，进行了检索并得到了具有实用价值的返回结果。

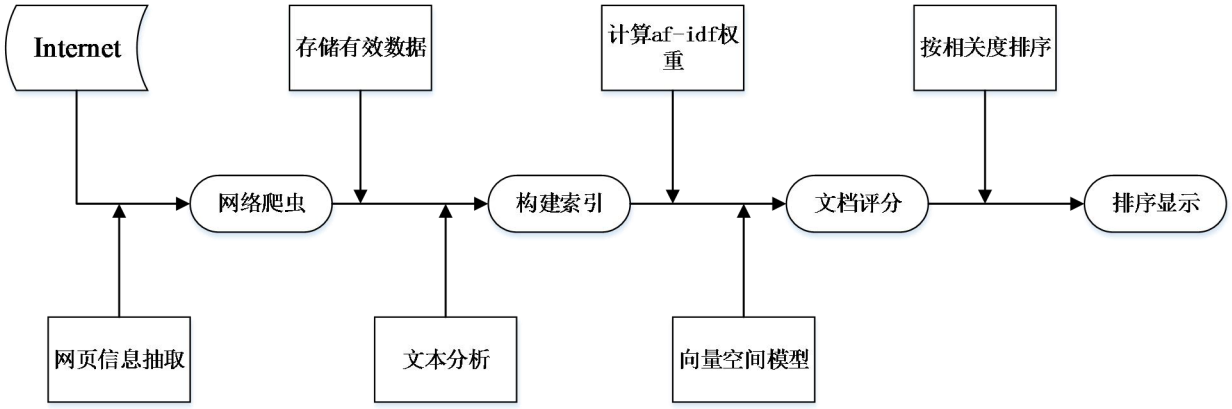
下面我们将会结合我们的代码详细讲解我们是如何制作前端界面，显示高亮，以及是如何对索引结果进行排序的。

我们还跟页面上的自带的搜索功能进行了一个简单的比对分析，得出了一些需要进一步进行改进的地方。

二、整体架构

结构分析：我们将任务分解为四个部分：萌娘百科词汇解释条目数据的爬取、倒排索引的构建、向量空间模型的实现和信息抽取前端界面。

主要分为四个模块：网络爬虫、构建索引、文档评分、排序显示。其中模块与模块之间又包含一些子模块，包括：网页信息抽取、数据存储、文本分析、tf-idf 权重计算、向量空间模型建模、相关度排序等。下面是整个搜索引起的设计结构图：

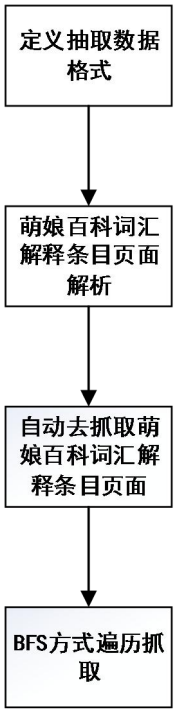


三、网络爬虫

在本次作业中，网络爬虫部分使用了 Beautiful Soup 开源框架。

Beautiful Soup 是用 Python 写的一个 HTML/XML 的解析器，它可以很好的处理不规范标记并生成剖析树。通常用来分析爬虫抓取的 web 文档。对于不规则的 Html 文档，也有很多的补全功能，节省了开发者的时间和精力。

本模块的流程图：



本模块的具体实现过程：

首先将源 url(https://zh.moegirl.org/Fate/stay_night)加载至 URL 管理器中，循环开始，判断条件是 URL 管理器中是否存在新的 url，当存在新的 url 时，开始爬虫；反之结束爬虫。那么爬虫开始后，就要借助 HTML 下载器下载源 url 中的所有页面内容，然后将下载得到的内容使用 HTML 解析器 HtmlParser 解析，得到正文出现的所有有效 url 以及正文内容和标题，接着把这些新获取的 urls 再次加载至 URL 管理器中，开始新一次的循环。最后将得到的数据使用 outputer 输出器输出并保存为 json 格式的文件。同时我们在爬虫过程中

还加入了异常处理程序，若出现异常，就会向控制台发送一条“craw failed”，继续新的下一次循环。

主要代码如下图：

```
from test import url_manager, html_downloader, html_parser,html_outputer

class SpiderMain(object):
    def __init__(self):
        self.urls=url_manager.UrlManager()
        self.downloader=html_downloader.HtmlDownloader()
        self.parser=html_parser.Htmlparser()
        self.outputer = html_outputer.HtmlOutputer()

    def craw(self,root_url):#调度程序
        count = 1 #用count记录当前爬取的是第几个Url
        self.urls.add_new_url(root_url)
        while self.urls.has_new_url():
            #百度百科有的网页已经无法访问，需要加入异常处理
            try:
                new_url = self.urls.get_new_url()
                print'craw %d : %s' %(count,new_url)
                html_cont = self.downloader.download(new_url)
                new_urls,new_data = self.parser.parse(new_url,html_cont)

                self.urls.add_new_urls(new_urls)
                self.outputer.collect_data(new_data)
                self.outputer.output_html(count)#每次抓到了就搞一次
                if count == 10000:
                    break
```

```
    def craw(self,root_url):#调度程序
        count = 1 #用count记录当前爬取的是第几个Url
        self.urls.add_new_url(root_url)
        while self.urls.has_new_url():
            #百度百科有的网页已经无法访问，需要加入异常处理
            try:
                new_url = self.urls.get_new_url()
                print'craw %d : %s' %(count,new_url)
                html_cont = self.downloader.download(new_url)
                new_urls,new_data = self.parser.parse(new_url,html_cont)

                self.urls.add_new_urls(new_urls)
                self.outputer.collect_data(new_data)
                self.outputer.output_html(count)#每次抓到了就搞一次
                if count == 10000:
                    break
                count =count +1

            except:
                print 'craw failed'
```

3.1 定义抽取数据的格式

对于爬虫的设计，首先需要确定抽取数据的格式，我们通过需求分析，决定抽取以下 3 种数据。具体介绍如下：

Artical：新闻的正文，用于文本分析和索引构建；

Title：新闻的标题；

URL：新闻的链接，用于 web 显示；

3.2 HTML 解析器

在确定数据格式后，接下来要抓取某一个萌娘百科词汇解释条目页面， 然后解析出与上述数据格式相对应的数据。我们通过引入 Beautiful Soup 中的一个 HTML 解析包 HtmlParser 来对页面信息进行提取。

具体实现过程：1、利用 HTML 解析包 HtmlParser 将 html 倒模型解析出来；2、使用正则表达式“/%+”把词条的有效 url 过滤出来；3、利用 Beautiful Soup 中的 find 函数对特定 中的正文和标题进行提取。主要代码如下图：

```
def _get_new_data(self, page_url, soup):
    res_data = {}
    #把url也放到最终的数据中

    #<div class="lemma-summary" label-module="lemmaSummary">

    #summary_node = soup.find('div',class_="lemma-summary")
    summary_node = soup.find('div',id="mw-content-text",class_="mw-content-ltr")
    summary_node = soup.find_all('p')
    res_data['Artical']=""
    for summary in summary_node:
        ss=summary.get_text()
        res_data['Artical'] = res_data['Artical']+ss
```

3.3 HTML 下载器

HTML 下载器主要是使用 urllib2 模块对文档进行下载，具体实现代码如下图：

```
#-*- coding: UTF-8 -*-
#=============================================================================
# '''
# Created on 2016年7月8日
#
# @author: gaochuyang
# '''
#=============================================================================
import urllib2
class  HtmlDownloader(object):


    def download(self,url):
        if url is None:
            return None
        response = urllib2.urlopen(url)
        if response.getcode()!= 200:
            return None
        return response.read()
```

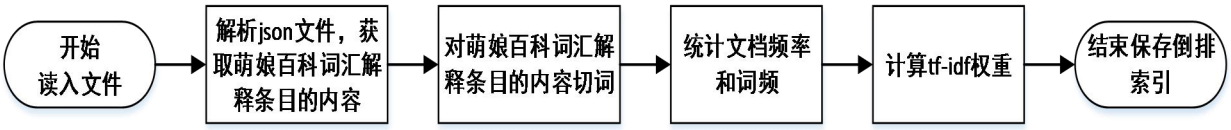
3.4 URL 管理器

URL 管理器可存储所有爬取到的 URL 地址，并且不会重复的爬取。具体实现代码如下图：

```python
class  UrlManager(object):
    def __init__(self):
        self.new_urls = set()#未爬取的url
        self.old_urls = set()#爬过的url
    #向管理器中添加一个url
    def add_new_url(self,url):
        if url is None:
            return
        if url not in self.new_urls and url not in self.old_urls:
            self.new_urls.add(url)



    def add_new_urls(self,urls):
        if urls is None or len(urls) == 0:
            return
        for url in urls:
            self.add_new_url(url)
    def has_new_url(self):
        return len(self.new_urls)!=0


    def get_new_url(self):
        new_urls = self.new_urls.pop()#pop方法会获取一个url并移除它
        self.old_urls.add(new_urls)
        return new_urls
```

四、倒排索引的构建

倒排索引是信息检索的重要一环。在这个模块中，主要包含四个关键步骤：从 json 文件中提取新闻内容、新闻内容切词、统计词频和文档频率、计算 tf-idf 权重。



4.1 文档切词

在常见的中文分词工具中，IKAnalyzer 分词器常用，效果好，方便拓展，受到了很多好评。因此，在我们的方案中，选用了 IKAnalyzer 分词器。IKAnalyzer 是一个开源的，基于 java 语言开发的轻量级的中文分词工具包。从 2006 年 12 月推出 1.0 版开始，IKAnalyzer 已经推出了 4 个大版本。

分词效果示例:IKAnalyzer 2012 版本支持细粒度切分和智能切分，以下是两种切分方式的演示样例。

文本原件 1:IKAnalyzer 是一个开源的，基于 java 语言开发的轻量级的中文分词工具包。从 2006 年 12 月推出 1.0 版开始，IKAnalyzer 已经推出了 3 个大版本。

智能分词结果:IKAnalyzer |是|一个|开源|的|基于|java|语言| 开发 | 的 | 轻量级 | 的 | 中文 | 分词 | 工具包 | 从 | 2006 年 | 12 月 | 推出 | 1.0 版 | 开始 | ikanalyzer | 已经 | 推| 出了 | 3 个 | 大 | 版本

最细粒度分词结果:IKAnalyzer | 是 | 一个 | 一 | 个 | 开源 | 的 | 基于 | java | 语言 | 开发 | 的 | 轻量级|的 | 中文 | 分词 | 工具包 | 工具 | 包 | 从 | 2006 | 年 | 12 | 月 | 推出 | 1.0 |版 | 开始 |ikanalyzer | 已经 | 推出 | 出了 | 3 | 个 | 大 | 版本。

安装部署方法：它的安装部署十分简单，将 IKAnalyzer2012.jar 部署于项目的 lib 目录中；IKAnalyzer.cfg.xml 与 stopword.dic 文件放置 class 根目录（对于 web 项目，通常是 WEB-INF/classes 目录，同 hibernate、log4j 等配置文件相同）下即可。

主要流程：将上一步提取的新闻主体的 txt 文档，全部遍历切词，将切词结果保存中本地中。

4.2 倒排索引构建流程

在本题目中，我们爬取得到了 4287 篇文档，数据量不是太大。因此，采用基于内存的索引构建方式，将索引构建的过程全部放入内存中进行统计，单次扫描，单次统计出结果，构建出索引。

主要流程：将上一步中分词的 4287 篇文档，全部遍历。在循环中，我们统计文档中每个词项出现的位置和次数，以及在 4287 篇文档中的总次数。在具体实现中，我们用两个 HashMap 来保存结果，HashMap 用词项做键，文档名、文档位置及其他属性为值（中间用特殊符号分割开），一个用于统计每篇文档的情况，一个统计所有文档的情况。

关键代码如下：

```
private static void CreateIndex() {

    Set<String> stopSet = new HashSet<String>();
    try {
        stopSet = GetStopWordSet("data/stopwords.txt");
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    HashMap<String, String> hashResult = new HashMap<String, String>();
    String fileName = "";
    // 读取文件
    try {
        for (int fileIndex = 1; fileIndex <= 100000; fileIndex++) {
            fileName = fileIndex + ".txt";
            File mFile = new File("data/words/" + fileName);
            if (!mFile.exists()) {
                System.out.println(fileName + "不存在");
                continue;
            }
            HashMap<String, Integer> hashMap = new HashMap<String, Integer>();
            String content = ReadAndWrite.readFileByChars("data/words/"
                    + fileName, "utf-8");
            String[] wordArray = content.split("\\|"); // “|”是转义字符 so need to addd \\
            /*split 方法:将一个字符串分割为子字符串，然后将结果作为字符串数组返回*/
            for (int i = 0; i < wordArray.length; i++) {
```

```
                String tmp = fileName + "#" + hashMap.get(str);// 最后一个为出现次数
                if (hashResult.keySet().contains(str)) {// 包含该词
                    String value = (String) hashResult.get(str);
                    value += ("@" + tmp);//如果包含该词说明不同的文档里都有它所以加上@来区分不同文档的value
                    hashResult.put(str, value);
                } else
                    hashResult.put(str, tmp);
            }
            System.out.println(fileName);
        }//至此计算出了所有文档中所有分词出现的次数,并弄出了每个词在不同文档中出现的次数和在同一文档中出现的次数
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } finally {//finally它只能用在try/catch语句中，并且附带着一个语句块，表示这段语句最终总是被执行。
        if (hashResult.size() > 0) {
            String value = "";

            for (String str : hashResult.keySet()) {
                String tmp = str + "&" + hashResult.get(str);// liang ge
                                                             // kong ge
                String[] timeWord = tmp.split("@"); // “|”是转义字符
                // 统计在文档中出现的总次数
                tmp += "&" + timeWord.length; //最后一个timeWord.length为这个词出现在了几个文档中
                System.out.println(tmp);//word&filename#times@filename1#times &length
                                        //用@分开以后就统计出了一个词出现在了几个文档中
```

4.3 计算 td-idf 权重

信息检索领域最出名的权重计算方法，tf-idf 权重计算公式：

$$w_{t,d} = (1 + \log tf_{t,d}) \times \log \frac{N}{df_t}$$

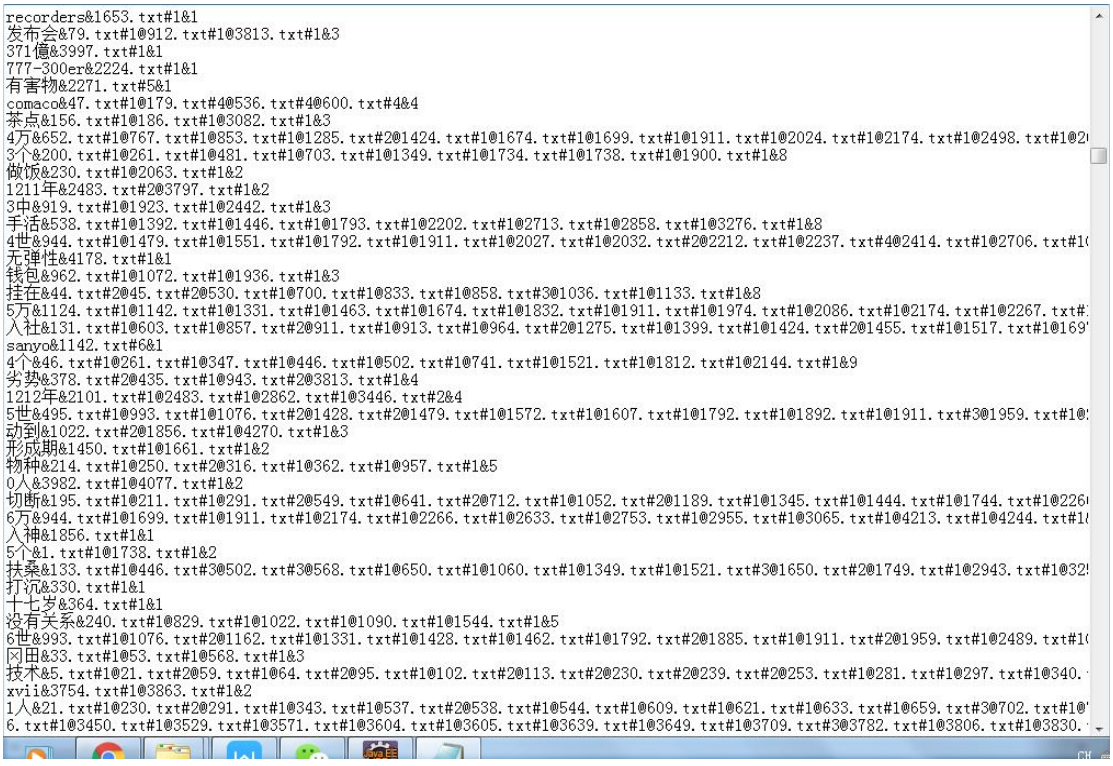
其中，df_t 是出现词项 t 的文档数目称为逆文档频率，$tf_{t,d}$ 是指 t 在 d 中出现的次数，是与文档相关的一个量，称为词项频率。随着词项频率的增大而增大，随着词项罕见度的增加而增大。

```
public static String getTfidfString(String content) {
    String wordKey; //要索引的单词
    int numbersOfDocs; //出现次数 这个值指的是一个词出现在了多少个文档里
    String tfidfStrs; //tfidf的值
    tfidfStrs="";
    String[] arrayStrings = content.split("&");
    wordKey = arrayStrings[0];
    numbersOfDocs = Integer.parseInt(arrayStrings[2]);//出现次数
                                                      //这个值指的是一个词出现在了多少个文档里

    String[] docsListArray=arrayStrings[1].split("@");//一个词在不同的文档里面出现的次数

    tfidfStrs+=wordKey;
    for (int i = 0; i < docsListArray.length; i++) {//对于这个词出现的每一个文档都要算权值
                                                    //即所谓的T-D
        String[] valuesStrings = docsListArray[i].split("#");
        String filenameString = valuesStrings[0];//valuesStrings0是文档的名字
        int timesOfwords = Integer.parseInt(valuesStrings[1]);//valuesStrings1是在文档中出现的次数
    //  int numOfDoc = Integer.parseInt(valuesStrings[1]);

        DecimalFormat dec = new DecimalFormat("0.0000");
        double tfidf = Tf_idf( timesOfwords, numbersOfDocs);

        tfidfStrs+="&";
        tfidfStrs+=valuesStrings[0];
        tfidfStrs+="@";
        tfidfStrs+=dec.format(tfidf);
```

4.4 索引文件格式

索引的每一条记录都遵循如下规则：wordkey&文档名@tfidf 值#next#&文档名@tfidf

最终形成倒排索引文件如下图：



五、向量空间模型建模

5.1. 概念简介

向量空间模型（VSM：Vector Space Model）由 Salton 等人于 20 世纪 70 年代提出，并

成功地应用于著名的 SMART 文本检索系统。把对文本内容的处理简化为向量空间中的向量运算，并且它以空间上的相似度表达语义的相似度，直观易懂。

VSM 概念简单，把对文本内容的处理简化为向量空间中的向量运算，并且它以空间上的相似度表达语义的相似度，直观易懂。当文档被表示为文档空间的向量，就可以通过计算向量之间的相似性来度量文档间的相似性。文本处理中最常用的相似性度量方式是余弦距离。

M 个无序特征项 ti，词根/词/短语/其他每个文档 dj 可以用特征项向量 来表示（a1j,a2j，…，aMj）权重计算，N 个训练文档 AM*N=（aij）文档相似度比较 1）Cosine 计算，余弦计算的好处是，正好是一个介于 0 到 1 的数，如果向量一致就是 1，如果正交就是 0，符合相似度百分比的特性,余弦 的计算方法为，向量内积/各个向量的模的乘积.2）内积计算，直接计算内积，计算强度低，但是误差大。

向量空间模型是一个应用于信息过滤，信息撷取，索引以及评估相关性的代数模型。

文件（语料）被视为索引词（关键词）形成的多次元向量空间， 索引词的集合通常为文件中至少出现过一次的词组。搜寻时，输入的检索词也被转换成类似于文件的向量，这个模型假设，文件和搜寻词的相关程度，可以经由比较每个文件(向量）和检索词（向量）的夹角偏差程度而得知。余弦为零表示检索词向量垂直于文件向量，即没有符合，也就是说该文件不含此检索词。

通过上述的向量空间模型，文本数据就转换成了计算机可以处理的结构化数据，两个文档之间的相似性问题转变成了两个向量之间的相似性问题。

5.2 问题描述

5.2.1 系统任务

文本分类系统的任务是：在给定的分类 体系下，根据文本的内容自动地确定文本关联的类别。 从数学角度来看．文本分类是一个映射的过程，它将未 标明类别的文本映射到已有的类别中。该映射可以是一一映射，也可以是一对多的映射．因为通常一篇文本 可以同多个类别相关联，用数学公式表示如下： f：A->B 其中．A 为待分类的文本集台．B 为分类体系中的类别集合。

文本分类的映射规则是系统根据已经掌握的每类 若干样本的数据信息,总结出分类的规律性丽建立的 判别公式和判别规则；然后在遇到新文本时，确定文本相关的类别。

5.2 评估方法

因为文本分类从根本上说是一个映射过程,所以 评估文本分类系统的标志是映射的准确程度和映射的 速度。映射的速度取决于映射规则的复杂程度．而评 估映射准确程度的参照物

是通过专家思考判断后对文 本的分类结果(这里假设人工分类完全正确并且排除 个人思维差异的因素)。与人工分类结果越相近，分类 的准确程度就越高。

```
static String filepath = "C:\\test\\index.txt";//倒排索引文件的目录位置
static HashMap<String, LinkedList<String>> WDocs = new HashMap<String, LinkedList<String>>();// <Term,Documents>
static HashMap<String, Document> docs = new HashMap<String, Document>();//文档向量
static HashMap<String, Double> queryWeight = new HashMap<String, Double>();// <Term,Weight>
```

定义 String 类型的 filepath 指向倒排索引文件的目录位置
定义 HashMap 类型的 WDocs 为文件中一些属性
定义 HashMap 类型的 docs 为文档向量
定义 HashMap 类型的 docs 为权重值
1. load( )

```
public static void Load() {
    new HashMap<String, LinkedList<Document>>();
    BufferedReader br = null;
    int n = 0;
     File f=new File(filepath);//changed by jingtiangao
    try {
         FileInputStream in=new FileInputStream(f);
         InputStreamReader r=new InputStreamReader(in,"utf-8");
           br=new BufferedReader(r);
        //br = new BufferedReader(new FileReader(filepath));

        String line;
        LinkedList<String> list;

        int z = 0;
        while ((line = br.readLine()) != null) {
            n++;
            list = new LinkedList<String>();
            // System.out.println(line);
            // System.out.println(line);
            String[] msg = line.split("&");
            String term = msg[0].trim();// TERM
            for (int i = 1; i < msg.length; i++) {
                msg[i] = msg[i].replace("#next#", "");
                String[] info = msg[i].split("@");
                String doc = info[0].trim();// DoucmentTitle  trim 为去掉字符左右的空格

                double weight = Double.valueOf(info[1]);//weight

                Document dos = docs.get(doc);//doc是这个文档的名字str
                if (dos != null) {           // dos是这个文档的向量
```

Load() 的主要功能是切词。首先读入文件。

```
for (int i = 1; i < msg.length; i++) {
    msg[i] = msg[i].replace("#next#", "");
    String[] info = msg[i].split("@");
    String doc = info[0].trim();// DoucmentTitle  trim 为去掉字符左右的空格

    double weight = Double.valueOf(info[1]);//weight

    Document dos = docs.get(doc);//doc是这个文档的名字str
    if (dos != null) {           // dos是这个文档的向量
        dos.setWeight(term, weight);
        docs.put(doc, dos);
    } else {
        dos = new Document();
        dos.setWeight(term, weight);
        docs.put(doc, dos);
    }
    list.add(doc);//把文档的名字加入list
}
```

在这个循环体中，去掉字符左右的空格。

定义 dos 是文档的向量，dos 是文档的名字 str，只要有新的文档向量时，设置向量权重，并把文档的名字加入 list。

```
WDocs.put(term, list);//wdoc为一个单词对应的所有的文档的表
```

wdoc 为一个单词对应的所有文档的表。

```
N = n;
System.out.println("载入内存中文档的数目为: " + n);
System.out.println("我在执行中");
```

2. query( )

```
public static LinkedList<String> Query(String query) throws IOException {
    LinkedList<String> list1 = null;
    LinkedList Quer = new Segment(query).run();//对查询语句进行分词
    queryWeight.clear();
    LinkedList lis = new LinkedList();
    for (int i = 0; i < Quer.size(); i++) {
        System.out.println(Quer.get(i) + "|");
        if (WDocs.containsKey(Quer.get(i))) {

            System.out.println("倒排索引中有: " + Quer.get(i));
            if (queryWeight.containsKey(Quer.get(i))) {
                String key = (String) Quer.get(i);
                double count = queryWeight.get(key);
                count++;
                queryWeight.put(key, count);
            } else {
                String key = (String) Quer.get(i);
                queryWeight.put(key, 1.0);
            }

            lis.add((String) Quer.get(i));//lis为索引中的所有关键词
        } else {
            System.out.println("倒排索引中没有: " + Quer.get(i));
        }
```

首先先进行分词， 建立新的在哈希表中查询索引。

```
        System.out.println();
        KeyWords = lis;
        System.out.println("索引中的关键词有: " + lis);
        // Start Compute the weight of terms
        java.util.Iterator<String> it = queryWeight.keySet().iterator();
        while (it.hasNext()) {
            String key = it.next();
            if (WDocs.containsKey(key)) {
                int n = WDocs.get(key).size();
                double idf = Math.log10(N / n);
                double tf = queryWeight.get(key);
                double w = (1 + Math.log10(tf)) * idf;
                queryWeight.put(key, w);
                // System.out.println("Term:"+key+"    weight:"+w);
            }//算出了真正的query中的权重
        }
```

计算机并不具有人类的智能，人在阅读文章后，根 据自身的理解能力可以产生对文章内容的模糊认识； 而计算机并不能轻易地“读懂”文章，从根本上说，它只 认识 0 和 1，所以必须将文本转换为计算机可以识别 的格式。根据“贝叶斯假设”，假定组成文本的字或词 在确定文本类别的作用上相互独立，这样，可以就使用 文本中出现的字或词的集合来代替文本。不言而喻， 这将丢失大量关于文章内容的信息，但是这种假设可 以使文本的表示和处理形式化，并且可以在文本分类 中取得较好的效果。 目前，在信息处理方向上，文本的表示主要采用向 量空间模型(VSM)。向量空间模型的基本思想是以向 量来表示文衣：(w1，w2,…,wn)，其中粥为第 i 个特征项的权重。那么选取什么作为特征项呢?一般 可以选择字、词或词组。根据实验结果，普遍认为选取 词作为特征项要优于字和词组，因此．要将文本表示为 向量空间中的一个向量，就首先要将文本分词，由这些 词作为向量的维数来

表示文本。最初的向量表示完全 是 0,1 形式，即：如果文本中出现了该词，那么文本向量的为 1．否则为 0。

文档向量化方法：利用 TF-IDF 权重进行表示。

(3)Similarity

```
public static double Similarity(LinkedList query, String doc) {
    java.util.Iterator<String> it = docs.get(doc).terms.keySet().iterator();
    LinkedList list = new LinkedList();
    double d = 0.0;
    while (it.hasNext()) {
        String term = it.next();
        d = Weight(doc, term) * Weight(doc, term) + d;
        list.add(term);
    }
    d = Math.sqrt(d);//文档向量的模
    double q = 0.0;
    for (int i = 0; i < query.size(); i++) {
        String term = (String) query.get(i);
        double weight = queryWeight.get(term);
        q = q + weight * weight;//query的模
    }
    q = Math.sqrt(q);
    //找出query和文档共同的部分，做内积
    list = ANDList(query, list);
    double sim = 0.0;
    for (int i = 0; i < list.size(); i++) {
        String term = (String) list.get(i);
        sim = sim + Weight(doc, term) * queryWeight.get(term);
    }
    double score = sim / d / q;
    return score;
}
```

计算相似度，将每个文件的权重加入到哈希表中。

排序方法：根据相似度计算结果，按照得分高低来进行排序。 同时，可以根据时间和热度进行简单排序。

(4) LinkedList

```
public static LinkedList<String> Top10(LinkedList query, LinkedList lists) {
    Map map = new TreeMap();
    LinkedList<String> list1 = new LinkedList<String>();
    for (int i = 0; i < lists.size(); i++) {
        String docname = (String) lists.get(i);
        double score = Similarity(query, docname);
        // System.out.println("Score=" + score);
        map.put(docname, score);
    }

    List<Map.Entry<String, Double>> list = new ArrayList<Map.Entry<String, Double>>(
            map.entrySet());

    Collections.sort(list, new Comparator<Map.Entry<String, Double>>() {
        public int compare(Entry<String, Double> o1,
                Entry<String, Double> o2) {
            return o2.getValue().compareTo(o1.getValue());
        }
    });
    int i = 0;
    for (Map.Entry<String, Double> mapping : list) {
        if ((i++) < 30) {
            // System.out.println(mapping.getKey()+":"+mapping.getValue());
            list1.add(mapping.getKey());
        } else
            break;
    }

    return list1;
}
```

最后的排序结果 LinkedList 以哈希表的形式返回。

六、搜索引擎前端

6.1 效果展示

本次实验对知名动漫咨询网站萌娘百科进行了检索，我们之所以对这家网站进行了实验，主要是想设计出一款能够帮助广大御宅族快速获取一手新鲜咨询的工具。

以下是我们返回的搜索结果：

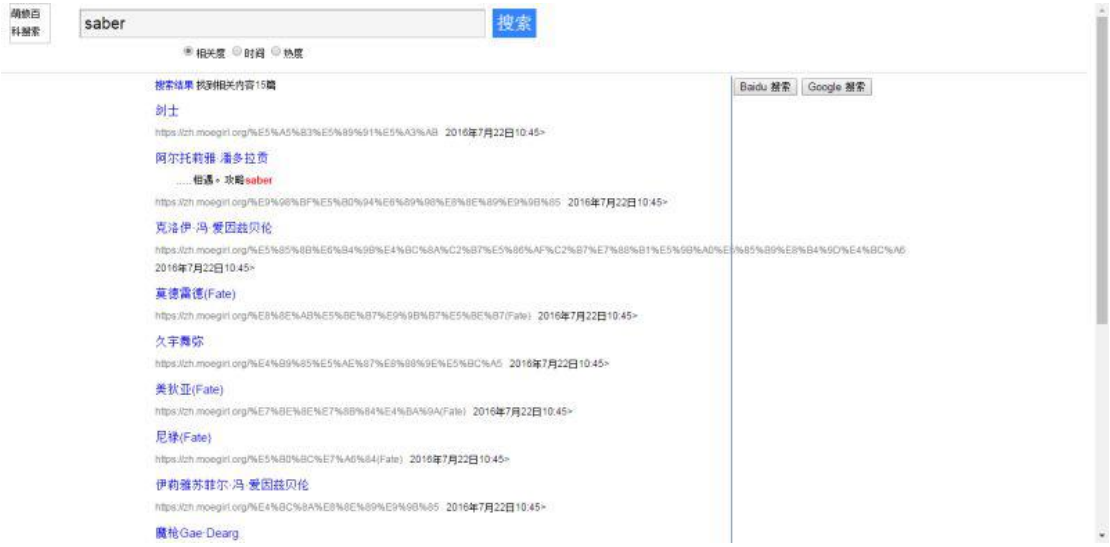


图 1-1 英文检索结果

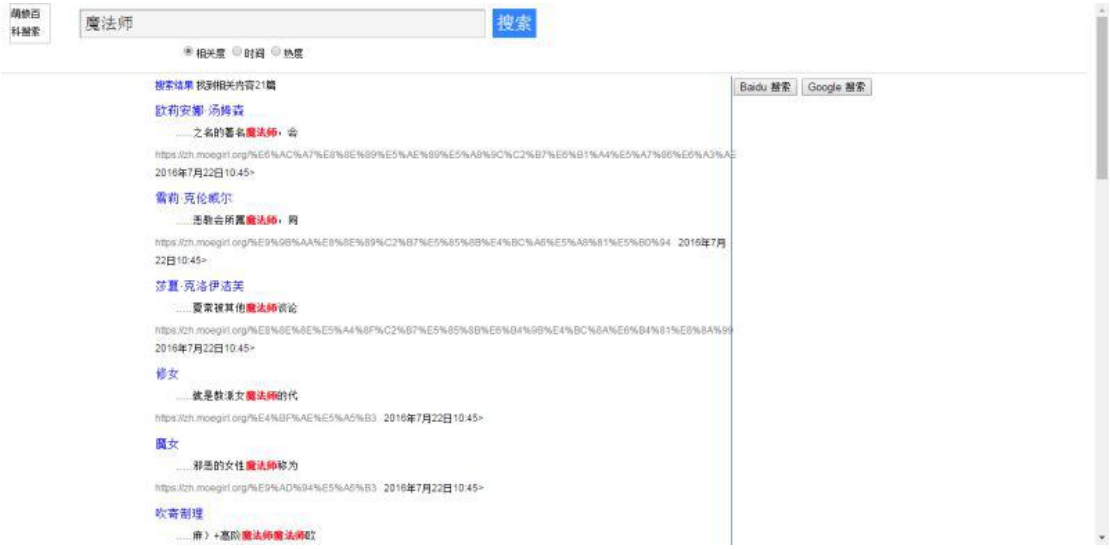


图 1-2 中文检索结果

图 1-3 多字段检索结果

我们还可以看一下相同的关键词使用网页自带的搜索功能所返回的结果，如下图所示，当搜索内容为词条时，页面会自动进入该词条的解释：

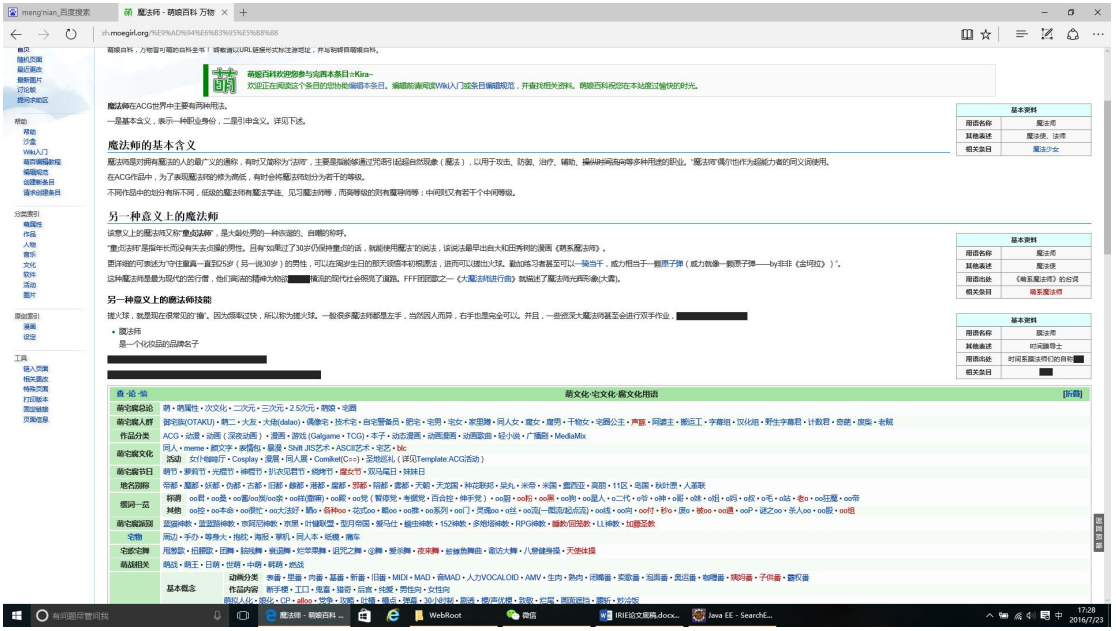


图 1-4 进入词条

在输入内容并非已知词条时，返回结果如下：

图 1-5 搜索“1984 年” “提案”

显然其返回的结果跟我们的引擎有所不同。我们的返回结果多一些。官方在搜索引擎中使用了什么算法我们不得而知。比如他们可以根据个页面的访问次数对搜索结果进行排序和筛选，这些功能我们还实现不了。

此外，官方引擎的界面也要比我们的好看一些，这也是我们需要改进之处，应该根据二次元爱好者进行一些对应的美化工作。

6.2 前端设计代码段讲解

以下是我们前端的相关代码：

```
TreeMap<String,String> map1 = new TreeMap<String,String>(new
Comparator(){

        public int compare(Object o1, Object o2) {
            // TODO Auto-generated method stub
            return o2.toString().compareTo(o1.toString());
        }

    });

     //按热度的TreeMap，新添加
    TreeMap<String,String> map2 = new TreeMap<String,String>( new
Comparator(){

        public int compare(Object o1, Object o2) {
            // TODO Auto-generated method stub
            return o2.toString().compareTo(o1.toString());
        }

        });
```

我们先将所检索出的结果根据 VSM 算法得出的排序进行排序，按热度由高到低。

当输入关键字非空时则查找包含关键字的内容的绝对位置。

List 变量用于记录返回的结果条数，list 为 null 时提示“没有找到任何东西”

```
}else
        {
        %>
              <div class="bobyTitle">

                 <div class="bobyContent">
                      没有查到任何东西！！！
                 </div>
             </div>
                <%} %>
```

页面的设计是用 html 完成的，我们的显示项包括热度、结果数等。实现方式如下：

```
<div id="head">
         <a herf="index.jsp" id="input_a"><img alt="萌娘百科搜索"
class="inputimg"></a>
          <form id="fmSearch" method="post" action="index.jsp">
              <input type="text" name="keyWord" class="inputText"
value="<%=keyword %>"/>
            
           <input type="submit" value="搜索" class="inputsubmit"/>
           <form method="post" action="index.jsp" class="formRadio">
               <div style="clear:both;margin-left:230px;">
              <input type="radio" name="radio" value="score"
onclick="save()"/>相关度
              <input type="radio" name="radio" value="time"
onclick="save()"/>时间
              <input type="radio" name="radio" value="hottop"
onclick="save()"/>热度
              <!--input type="submit" value="提交"/-->
               </form>
              </div>
               </form>
```

按照之前的排序，我们将地址链接和简介信息逐条显示在页面上，对应的代码段如下图所示：

```
}
           for(String o:list)
           {
               //Engine.ResultModel mod = (Engine.ResultModel)o;
               //Engine.Result mod=new Result(o+".txt");
               //ucas.JsonObject myObject =
ucas.JsonUtil.readFile2JsonObject("/data/"+o+".json");
```

```
            String path= parse.Path(strPath,o);
               ucas.JsonObject myObject =
ucas.JsonUtil.readFile2JsonObject(path);
               String str = myObject.articalString;
               //LinkedList query=getSplit2Words(keyword);
               LinkedList<String> query=new
parse.Segment(keyword).run();
               String partContent=AutoAbstract.getAbstract(str,query);
        %>
        。。。。。。。。。。。。。。。。。。。。。。。。。
```

最后，作为功能的拓展，我们还在页面上加入了谷歌与百度的链接供大家使用，毕竟我们的数据库大小还是很有限的，相关代码如下所示：

```
    <form action="http://www.baidu.com/baidu">
                        <input type=hidden value='<%=keyword %>'
name=word>
                        <input type="submit" value="Baidu 搜索">
                        <input name=tn type=hidden value="bds">
                        <input name=cl type=hidden value="3">
                        <input name=ct type=hidden value="2097152">
                        <input name=si type=hidden
value="www.williamlong.info">
                    </form>
                    </td>
```

七、信息抽取显示部分代码说明

核心部分代码展示：

```
            //LinkedList query=getSplit2Words(keyword);
            LinkedList<String> query=new parse.Segment(keyword).run();
            String partContent=AutoAbstract.getAbstract(str,query);


    %>

        <div class="bobyTitle">
            <a href="<%=myObject.URL%>" target="_blank">
                <%= parse.HighLightKey(myObject.titleString) %></a></div>
            <div class="bobyContent">
                <%= parse.HighLightKey(partContent) %>
            </div>
            <div class="bobyUrl">
                <span style="color: Gray;"><%=myObject.URL %></span>  
                <span><%=myObject.timeString %></span>>
            </div>
    <%
        }
    }else
    {
    %>
        <div class="bobyTitle">

            <div class="bobyContent">
```

图 2-1 代码示意

```
        return list1;
    }



    public String HighLightKey(String content) {
        content = content.replaceAll(" ", "");
        for (int i = 0; i < KeyWords.size(); i++) {
            String word = KeyWords.get(i).toString();
            content = content.replaceAll(word,
                    "<font style='color:#ff0000;font-weight:bold;'>" + word
                            + "</font>");
        }

        return content.replaceAll(
                "</font>[\\W]*<font style='color:#ff0000;font-weight:bold;'>",
                "");
    }

    public String Path(String path, String o) {

        o = o.replace(".txt", ".json");
        String data = "data\\";
```

图 2-2 检索代码示意

信息检索的实现方式如上图所示。整体思路为，我们在整个网站上遍历所有的索引文件，对包含输入关键字的页面进行记录，然后将上下文本显示出来，高亮其中的关键字，并记录其 url，予以显示，供用户进行访问。

我们先生成了一个指向目标 url 的链接，页面上显示内容为关键字的包含句，其中关键字高亮标出。如图 2-1 所示。

高亮的实现比较简单，前面已经记录的关键字的起始位置和长度，我们只需逐字修改其颜色字体即可。如图 2-2 所示。

关于下面介绍文本的显示，我们以关键字本身为基准，显示其前后固定长度的字符（具体长度根据关键字的个数会进行调整）。

这样我们就完成了相关信息的初步提取，实际上就是返回关键词的上下文内容。

显然，我们会返回许多的相关页面。其显示顺序我们按照 VSM 排序算法来进行。

八、开发中遇到的问题

8.1 写爬虫程序时遇到的问题

一开始写爬虫程序时只爬取了 100 个网页作为测试，所以在代码里是把爬取到的数据统一存到一个 list 中，爬完所有的网页再统一保存，但是当数据非常大时，就有可能产生堆耗竭问题，导致内存溢出，所以后来改为，爬取一个网页后立刻进行持久化操作，然后，将内存中的 data 及时清除，防止内存空间不够用。

如果爬虫下载的某个 url 已经失效，会导致程序崩溃，所以在循环程序中加入了异常处理，防止因为 404 页面导致的爬虫停止。

修改后的代码如下：

```
    def craw(self,root_url):#调度程序
        count = 1 #用count记录当前爬取的是第几个Url
        self.urls.add_new_url(root_url)
        while self.urls.has_new_url():
            #百度百科有的网页已经无法访问，需要加入异常处理
            try:
                new_url = self.urls.get_new_url()
                print'craw %d : %s' %(count,new_url)
                html_cont = self.downloader.download(new_url)
                new_urls,new_data = self.parser.parse(new_url,html_cont)

                self.urls.add_new_urls(new_urls)
                self.outputer.collect_data(new_data)
                self.outputer.output_html(count)#每次抓到了就搞一次
                if count == 10000:
                    break
                count =count +1

            except:
                print 'craw failed'
        # self.outputer.output_html()
if __name__=="__main__":
    #root_url = "http://baike.baidu.com/view/21087.htm"
```

8.2 写搜索引擎程序时遇到的问题

我们的搜索引擎后台是用 Java 语言依托现有的框架进行二次开发写成的，但是爬虫获得的萌娘百科一个页面的数据结构是经过我们自己重新定义的 Json 格式的文件，所以要对编码、数据读写接口进行重写，否则后台就会出现 NULL Pointer Exception 此类的异常，还会出现读入索引文件乱码问题。

为了解决上述问题，我们将框架中原有的 Unicode 编码格式修改为 UTF-8，解决了乱码问题，重新写了 Json 格式文件读入部分的代码，解决了数据读写空指针的异常。修改后的代码如下。

```
 */
public static JsonObject readFile2JsonObject(String path) {
    String JsonContext = JsonUtil.ReadFile(path);
    JSONArray jsonArray = JSONArray.fromObject("["+JsonContext+"]");
    JSONObject jsonObject = jsonArray.getJSONObject(0);
    JsonObject MyObject = new JsonObject(jsonObject.get("Artical").toString(), jsonObject.get("Time").toString(),
                jsonObject.get("Total").toString(),
                jsonObject.get("title").toString(), jsonObject.get("URL").toString());

    return MyObject;
```

九、小组分工

高楚阳：信息索引、相关度排序代码开发，相关文档编写
杨雪：爬虫系统开发测试，相关文档部分编写
封依萌：信息抽取系统开发测试，相关文档编写
孙传昊：WEB 页面实现，相关文档编写
四人工作量相当，老师酌情给分即可。