**Софийски университет „Св. Климент Охридски“**

****

Факултет по Математика и Информатика

Курсов проект

Извличане на Информация

На тема

Изграждане на система за търсене в корпус от статии от Techcrunch.com

Изготвил :.................. Проверил:.........................

/Щилиян Александров Узунов/ /доц. Иван Койчев/

**Същност на системата**

Системата представлява търсачка в корпус от документи, предварително извлечени от сайтът за технически новини Techcrunch.com. За целта са изградени няколко модула, всеки от които с различно предназначение. Налични са модул за извличане на информацияа – Crawer, модул за индексиране на информацията – Indexer, модул за търсене във индексираната информация – Searcher и модул за достъп до информацията от потребителя – User Interface.

**Мотивация**

Избрах да реализирам следната система, защото за нейното изграждане са нужни комбиниране на множество различни технологии, търсене на информацията в предварително зададен източник, индексиране на информацията, търсене и т.н. Всички тези неща са нови за мен, и чрез изпълнението на проекта получих нови знания.

**Преглед на използваните технологии**

* Java – Програмен език
* Crawler4j – Java библиотека служеща за обхождането на новинарският сайт.
* Apache Lucene – Java библотека за индексиране и търсене в индексираната информация.
* Spring Framework – Java Framework за изграждане на web приложения.
* jQuery и Bootstrap – Библиотеки за изграждане на потребителският интерфейс.

**Програмна реализация**

1. Crawler Модул

Кролерът служи за извличане на информация от новинарският сайт Techcrunch.com. За реализацията му е използвана Java библиотеката Crawler4j. Тъй като целта на търсенето на системата е да търси само във документи със смислено текстово съдържание, crawler-а трябва да се изгради така че да събира само статиите от сайта.

**Начин на обхождане**

За да се избегне събиране на ненужната информация, и да се събират само полезни статии, кролера не обхожда целият сайт, а само части от него, предварително дефинирани от програмиста. Обхождането започва от секцията със теми : <http://techcrunch.com/topic/> от там се слиза към статиите на всяка тема, например <http://techcrunch.com/topic/product/amazon-web-services/> след което се избира статия от темата например : <http://techcrunch.com/2014/12/02/aws-simplifies-ec2-reserved-instance-pricing/> след което се сваля съдържанието на статията. За целта предварително се прави анализ на HTML структурата на статиите, и се намират общи елементи за всичк истатии, от които се сваля избраното съдържание.

**Реализация**

След наследяване на Crawler класа, се предефинират ShouldVisit и Visit класа. Във ShouldVisit се описва коит страници трябва да бъдат посетени. За филтриране е използван регулярен израз който обхваща само Topic страници и Article страници (странци със статии), ако линкът за посещение не отговаря на регулярният израз – страницата се перскача.

Във Visit класа е логиката която се изпълнява след посещение на страницата. За целта се проверява дали страницата е Article страница (с регулярния израз) след това се обхожда катo xml документ, и от нея се свалят отделните части – Заглавие, съдържание и т.н. и се записват във отделен .xml документ, котйо после се използва от индекс модула за индексиране.

Важен момент в реализацяита е „маскирането“ на кролера като нормален browser, тъй като Techcrunch.com не позволява обръщения към него от кролери, подобни на crawler4j, за целта се поставя Header поле което представя кролера пред сървъра на techctunch като Browser – Google Chrome.

**Код на модула**

package com.project.informationRetrieval.crawler;

import java.io.File;

import java.util.regex.Pattern;

import edu.uci.ics.crawler4j.crawler.Page;

import edu.uci.ics.crawler4j.crawler.WebCrawler;

import edu.uci.ics.crawler4j.parser.HtmlParseData;

import edu.uci.ics.crawler4j.url.WebURL;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerConfigurationException;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.jsoup.\*;

import org.jsoup.nodes.\*;

import org.jsoup.select.Elements;

public class Crawler extends WebCrawler {

private final static String CORPUS\_LOCATION = "D:\\Programming\\Java\_64\_Workspace\\informationRetrieval\\Corpus\\";

private final static Pattern RESOURCE\_FILTER = Pattern

.compile(".\*(\\.(css|js|gif|jpe?g" + "|png|mp3|mp3|zip|gz))$");

private final static Pattern TOPIC\_PAGE = Pattern

.compile("http://techcrunch.com/topic/[a-zA-Z|-]+/[a-zA-Z|-]+/(\\d+/)\*");

private final static Pattern ARTICLE\_PAGE = Pattern

.compile("http://techcrunch.com/\\d+/\\d+/\\d+/.\*");

private DocumentBuilderFactory docFactory = DocumentBuilderFactory

.newInstance();

private DocumentBuilder docBuilder;

@Override

public boolean shouldVisit(Page referringPage, WebURL url) {

String href = url.getURL().toLowerCase();

boolean isTopicPage = TOPIC\_PAGE.matcher(href).matches();

boolean isArticlePage = ARTICLE\_PAGE.matcher(href).matches();

boolean isResource = RESOURCE\_FILTER.matcher(href).matches();

return (isTopicPage || isArticlePage) && !isResource;

}

/\*\*

\* This function is called when a page is fetched and ready to be processed

\* by your program.

\*/

@Override

public void visit(Page page) {

String url = page.getWebURL().getURL();

if (ARTICLE\_PAGE.matcher(url).matches()) {

System.out.println("URL: " + url);

HtmlParseData data = (HtmlParseData) page.getParseData();

Document doc = Jsoup.parse(data.getHtml());

Elements articles = doc.select(".article-entry");

Element article = articles.first();

String title = doc.select(".tweet-title").first().text();

saveArticle(article, title, url);

}

}

private void saveArticle(Element article, String title, String url) {

String articleText = article.text();

title = title.replace('?', ' ');

title = title.replace('!', ' ');

title = title.replace('.', ' ');

title = title.replace(',', ' ');

try {

docBuilder = docFactory.newDocumentBuilder();

org.w3c.dom.Document doc = docBuilder.newDocument();

org.w3c.dom.Element rootElement = doc.createElement("Article");

doc.appendChild(rootElement);

org.w3c.dom.Element titleElement = doc.createElement("Title");

titleElement.setTextContent(title);

org.w3c.dom.Element contentElement = doc.createElement("Content");

contentElement.setTextContent(articleText);

org.w3c.dom.Element urlElement = doc.createElement("Url");

urlElement.setTextContent(url);

rootElement.appendChild(titleElement);

rootElement.appendChild(urlElement);

rootElement.appendChild(contentElement);

TransformerFactory transformerFactory = TransformerFactory

.newInstance();

Transformer transformer = transformerFactory.newTransformer();

DOMSource source = new DOMSource(doc);

String savePath = CORPUS\_LOCATION + title + ".xml";

StreamResult result = new StreamResult(new File(savePath));

transformer.transform(source, result);

} catch (ParserConfigurationException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (TransformerConfigurationException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (TransformerException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

package com.project.informationRetrieval.crawler;

import edu.uci.ics.crawler4j.crawler.CrawlConfig;

import edu.uci.ics.crawler4j.crawler.CrawlController;

import edu.uci.ics.crawler4j.fetcher.PageFetcher;

import edu.uci.ics.crawler4j.robotstxt.RobotstxtConfig;

import edu.uci.ics.crawler4j.robotstxt.RobotstxtServer;

public class CrawlerController {

private CrawlConfig config;

private CrawlController controller;

private static int NUMBER\_OF\_CRAWLERS = 1;

private void initConfig() {

config = new CrawlConfig();

config.setCrawlStorageFolder("D:\\Programming\\Java\_64\_Workspace\\informationRetrieval\\Corpus\\Meta");

config.setMaxPagesToFetch(-1);

config.setMaxDepthOfCrawling(4);

config.setPolitenessDelay(100);

config.setUserAgentString("Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.111 Safari/537.36");

config.setResumableCrawling(false);

}

private void initSeeds() {

controller.addSeed("http://techcrunch.com/topic/");

//controller.addSeed("http://techcrunch.com/topic/subject/internet-of-things/");

}

public CrawlerController() throws Exception {

initConfig();

PageFetcher pageFetcher = new PageFetcher(config);

RobotstxtConfig robotstxtConfig = new RobotstxtConfig();

robotstxtConfig.setEnabled(false);

robotstxtConfig.setUserAgentName("Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.111 Safari/537.36");

RobotstxtServer robotstxtServer = new RobotstxtServer(robotstxtConfig,

pageFetcher);

controller = new CrawlController(config, pageFetcher, robotstxtServer);

initSeeds();

}

public void start() {

controller.start(Crawler.class, NUMBER\_OF\_CRAWLERS);

}

public void stop() {

controller.shutdown();

}

1. Index Модул

Индекс модула служи за индексиране на вече изтеглените от Crawler-a документи. За да бъде търсенето по-ефективно, документа се разделя на 2 полета – заглавие и съдържание, всяко от което с различна тежест при търсенето. За целта се използват вградените методи на Apache Lucene.

Apache Lucene вътрешно имплементира InvertedIndex структурата, която служи за асоцииране на отделните терми, към различни документи. Използва се също така и StandardAnalyzer класа, който първо при приемане на нов документ – го разделя на терми, премахвайки всички бели пространства, след това премахва стоп думите. За премахването на стоп думите се използва вграденият в Lucene речник със стоп думи на английски език.

При индексирането се поставят различни тежести на заглавието и съдържанието. Идеята зад това решение е че ако даден терм се съдържа в заглавието, то той ще бъде по-важен от колкото ако се съдържа в тялото на документа. Теглата са съответно 0.6 за заглавие и 0.4 за съдържание на терм в документа.

При индексирането на документите освен тези 2 полета, са добавени и още 2 – които нямат значение при търсенето, и не се индексират, но се запазват с индекса (докато другите 2 не се запазват) – това са полетата URL и FileName. FileName се запазва за да може при търсене да се извлече информацията от файла на диска, а URL – за да може потребителя сам да прегледа страницата по-късно.

Резултатът от индексирането е файл, с формат – вътрешно дефиниран от Apache Lucene в който се съдържа InvertedIndex структурата. Този Файл се използва по-късно от Searcher-a за търсене по заявка в структурата.

**Код на модула**

package com.project.informationRetrieval.indexer;

import java.io.File;

import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import org.apache.lucene.index.IndexWriter;

import org.apache.lucene.index.IndexWriterConfig;

import org.apache.lucene.store.FSDirectory; //org.apache.lucene.analysis.standard

import org.apache.lucene.util.Version;

import org.apache.lucene.analysis.standard.\*;

import org.apache.lucene.document.Document;

import org.apache.lucene.document.Field;

import org.apache.lucene.document.FieldType;

import org.apache.lucene.document.Field.TermVector;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

public class Indexer {

// Directory where documents come from

public static final String DATA\_DIR = "D:\\Programming\\Java\_64\_Workspace\\informationRetrieval\\Corpus\\Docs\\";

// Directory where index is stored

public static final String INDEX\_DIR = "D:\\Programming\\Java\_64\_Workspace\\informationRetrieval\\Corpus\\Index\\";

private IndexWriter indexWriter;

public static StandardAnalyzer analyzer = new StandardAnalyzer(Version.LUCENE\_46);

public Indexer() throws IOException {

IndexWriterConfig config = new IndexWriterConfig(Version.LUCENE\_46,

analyzer);

indexWriter = new IndexWriter(FSDirectory.open(new File(INDEX\_DIR)),

config);

}

public void startIndexing() throws Exception {

File[] files = new File(DATA\_DIR).listFiles();

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

double segmentSize = files.length/100;

double startPrec = segmentSize;

for(int i = 0 ; i < files.length ; i++) {

File file = files[i];

org.w3c.dom.Document doc = dBuilder.parse(file);

addFileToIndex(doc);

if(i > startPrec) {

System.out.println((startPrec / segmentSize) + " % done.");

startPrec += segmentSize;

}

}

indexWriter.close();

}

@SuppressWarnings("deprecation")

private void addFileToIndex(org.w3c.dom.Document doc) throws IOException {

//Extracting info from XML document

String title = doc.getElementsByTagName("Title").item(0).getTextContent();

String content = doc.getElementsByTagName("Content").item(0).getTextContent();

String url = doc.getElementsByTagName("Url").item(0).getTextContent();

//Field types and properties

Field fileNameField = new Field("fileName", title, Field.Store.YES, Field.Index.NO, TermVector.NO);

Field titleField = new Field("title", title, Field.Store.NO, Field.Index.ANALYZED, TermVector.WITH\_POSITIONS\_OFFSETS);

Field contentField = new Field("content", content, Field.Store.NO, Field.Index.ANALYZED, TermVector.WITH\_POSITIONS\_OFFSETS);

Field urlField = new Field("url", url, Field.Store.YES, Field.Index.NOT\_ANALYZED, TermVector.NO);

//Boosting

titleField.setBoost(0.6f);

contentField.setBoost(0.4f);

//Adding the fields to the lucene Document

Document luceneDoc = new Document();

luceneDoc.add(fileNameField);

luceneDoc.add(titleField);

luceneDoc.add(contentField);

luceneDoc.add(urlField);

indexWriter.addDocument(luceneDoc);

}

}

1. Search Модул

Search модула служи за търсене във вече изграденият index от Index модула. За целта му е нужен достъп до файлът, изграден от indexer-a. Търсенето е вътрешно реализирано във Apache Lucene. Изпълнява се като подадената заявка за търсене се разцепва на отделни терми (използвайки същият анализатор като index modula), след което се търси кой от индексираните документи има най-голямо съвпадение със посочените от заявката терми.

Параметрите които се подават при търсене са 2 – параметър заявка, и параметър брой резултаити на страница. Заявката – се подава на Lucene за изпълнение на търсенето, брой резултати на страница - също се подава на Lucene. Целта на този параметър е да не се показват на потребителя всички документи, а само N на брой най-добри, със възможност той после да си избере някоя от другите страници при търсенето, и да посети нея.

Друг метод на Search модула е извличането на документ по подадено ID. Тъй като Lucene има своя вътрешна индексация, и намерените резултати от търсенето са на документи, които се идентифицират единствено по ID, REST модула(описан по-долу) има жунда да знае точно кой документ да поиска от файловата система. За целта той трябва да достъпи Lucene документа където е описано името на файла, а за да достъпи Lucene документа – му е нужно неговото LuceneID.

Като крайни наблюдения от работата на Index Модула и Search Модула са следните цифри – при корпус от 6000 документа, и заявка “Windows Youtube” търсачката връща 539 резултата за 0.29 секунди. Резултатът от 0.29 секунди е повече от задоволителен, така че можем да заключим че двата модула са имплементирани правилно, и работата на системата ще бъде удовлетворяваща за крайният потребител, предоставяйки му достатъчно количество информация, за кратко време.

**Код на модула**

package com.project.informationRetrieval.searcher;

import com.project.informationRetrieval.indexer.Indexer;

import java.io.File;

import java.io.IOException;

import org.apache.lucene.document.Document;

import org.apache.lucene.index.IndexReader;

import org.apache.lucene.queryparser.classic.ParseException;

import org.apache.lucene.queryparser.classic.QueryParser;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.TopDocs;

import org.apache.lucene.store.FSDirectory;

import org.apache.lucene.util.Version;

import org.apache.lucene.index.DirectoryReader;

public class Searcher {

private IndexSearcher searcher;

public Searcher() throws IOException {

IndexReader reader = DirectoryReader.open(FSDirectory.open(new File(Indexer.INDEX\_DIR)));

searcher = new IndexSearcher(reader);

}

public TopDocs executeQuery(String stringQuery, int resultsPerPage) throws ParseException, IOException {

QueryParser qParser = new QueryParser(Version.LUCENE\_46, "content", Indexer.analyzer);

Query q = qParser.parse(stringQuery);

TopDocs docs = searcher.search(q, resultsPerPage);

return docs;

}

public Document getDoc(int docID) throws IOException {

return searcher.doc(docID);

}

}

1. WebApi Модул и UserInterface Модул

Тъй като тези 2 модула нямат пряка връзка към предмета, а са били предмет на изучаване в други дисциплини, не смятам за нужно да разяснявам в детайли тяхната реализация.

На кратко – Използваният Server е Apache Tomcat. В него са деплойнати .war файлове съдържащи имплементация на RESTFull Service-и които са реализирани със Java Spring Framework-а Задачата на тези Service-и е при обръщение от страна на клиента, да предадат заявката към Search модула, и след това да върнат резултат.

Клиентската страна е реализирана с помоща на програмната библиотека с отворен код – jQuery, и библиотеката за стилизиране също с отворен Bootstrap.

**Предложения за бъдещо развитие**

* Да се добави Soundex Алгоритъм към анализатора за коригиране на направените потребителски грешки при въвеждане на заявката.
* Да се добави Autocomplete на всяка дума за по-голямо удобство.
* Да се добави Machine Learning модул който да научава кои са харесваните от потребителя статии, и след това добавя неговите желания при правенето на оценка на резултатите.