ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ

Ордена Трудового Красного Знамени

федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский Технический Университет Связи и Информатики» (МТУСИ)

Кафедра МКиИТ

Лабораторная работа №6

«Сёрфинг в Интернете»

Выполнил:

Студент 2 курса

Группы БСТ-1602

Смышляев Владислав

Вариант №15

Москва, 2018

Содержание

[1. Цель и задачи 3](#_Toc512789635)

[2. Анализ предметной области, выбор инструментария 3](#_Toc512789636)

[3. Объяснение функций 3](#_Toc512789637)

[3.1. Класс URLDepthPair 3](#_Toc512789638)

[3.2. Класс Crawler 4](#_Toc512789639)

[4. Выводы 7](#_Toc512789640)

1. Цель и задачи

Цель работы: Изучить способы получения доступа к Интернет странице и получения от неё ссылок, которые также в свою очередь исследуются.

Задачи: Написать программу, которая по заданному через консоль адресу будет искать больше адресов, в которых будут искаться ещё больше адресов, пока программа не достигнет максимальной глубины поиска заданной, также через параметры консоли.

1. Анализ предметной области, выбор инструментария

Для написания данной программы использоваться IDE IntelliJ Idea, так как в данной и последующих задачах требуются средства для отладки программы. Здесь также представлены средства для работы с контролем версий, но для собственного удобства коммиты будет совершаться через Visual Studio Code.

1. Объяснение функций
   1. Класс URLDepthPair

import java.net.MalformedURLException;  
  
public class URLDepthPair {  
 private String URL;  
 private int depth;  
  
 */\*\*  
 \* Instancing the pair  
 \** ***@param*** *url - URL to store  
 \** ***@param*** *depth - Depth of the URL  
 \*/* URLDepthPair(String url, int depth){  
 this.URL = url;  
 this.depth = depth;  
 }  
  
 @Override  
 public String toString() {  
 return "URL: "+URL+" Depth: "+depth;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 URLDepthPair comp\_pair = (URLDepthPair) obj;  
 return comp\_pair.URL.equals(URL);  
 }  
  
 public int getDepth() {  
 return depth;  
 }  
  
 */\*\*  
 \* first component is the protocol  
 \* second component is the hostname  
 \* third component is the path to the resource  
 \** ***@return*** *components of the URL in string array  
 \*/* public String[] parse() {  
 String[] components = new String[3];  
 components[0] = "";  
 components[1] = "";  
 components[2] = "";  
 //Attempting to create the URL from the given string  
 try {  
 java.net.URL url = new java.net.URL(URL);  
 //Get the needed components  
 components[0] = url.getProtocol();  
 components[1] = url.getHost();  
 components[2] = url.getPath();  
 }catch(MalformedURLException e){  
 System.*out*.println("Error while parsing the URL, returned empty strings");  
 return components;  
 }  
 return components;  
 }  
}

Вспомогательный класс, где осуществляется парсинг и проверка на равенство пар. Описания всех функций, как всегда даны в комментариях к коду.

* 1. Класс Crawler

import java.io.\*;  
import java.net.Socket;  
import java.util.LinkedList;  
  
public class Crawler extends Thread {  
 private LinkedList<URLDepthPair> found\_urls; //List of processed URLS  
 private LinkedList<URLDepthPair> non\_processed\_urls; //List of pending to be processed URLs  
 private final int webPort = 80; //Port to connect to  
 private int max\_depth; //Max crawling depth, passed as parameter  
 private final String PROTO\_SEARCH = "http"; //Protocol to search for in links  
 //Indexing in url parts  
 private final int PROTOCOL = 0;  
 private final int HOSTNAME = 1;  
 private final int RESOURCE = 2;  
  
 private final String HREF = "href=\""; //HTML tag  
  
 */\*\*  
 \* Program's entry point  
 \** ***@param*** *args - Console arguments (At least 1 URL link and crawl depth is required)  
 \*/* public static void main(String[] args){  
 if (args.length < 2){  
 System.*out*.println("Not enough arguments provided. At least 1 URL link and maximum crawl depth is required!");  
 return;  
 }  
 String[] links = args; //Store reference to args in links  
 int depth = Integer.*parseInt*(args[args.length-1]); //Get the crawl depth  
 String[] url\_pool = new String[links.length-1]; //Generate the links to crawl through  
 for(int i=0;i<links.length-1;i++){  
 url\_pool[i] = links[i]; //Copy links to the array  
 }  
 Crawler[] crawler = new Crawler[url\_pool.length]; //Create the crawlers  
 //Start crawling  
 for(int i=0;i<crawler.length;i++){  
 crawler[i] = new Crawler(url\_pool[i],depth);  
 crawler[i].start();  
 }  
 }  
  
 */\*\*  
 \* Create the crawler  
 \** ***@param*** *url - URL to crawl through  
 \** ***@param*** *depth - maximum crawling depth  
 \*/* Crawler(String url, int depth){  
 found\_urls = new LinkedList<>();  
 non\_processed\_urls = new LinkedList<>();  
 non\_processed\_urls.add(new URLDepthPair(url,0));  
 max\_depth = depth;  
 }  
  
 */\*\*  
 \* Thread's task  
 \* Get list of sites and print them out  
 \*/* @Override  
 public void run() {  
 LinkedList<URLDepthPair> list = getSites();  
 while(!list.isEmpty()) {  
 System.*out*.println(list.removeFirst());  
 }  
 }  
  
 */\*\*  
 \* Crawls through every site until no sites to search for or reaching too big of a depth  
 \** ***@return*** *list of crawled through sites  
 \*/* private LinkedList<URLDepthPair> getSites() {  
 //Check for list emptiness and current crawl depth  
 while(!non\_processed\_urls.isEmpty() && non\_processed\_urls.getFirst().getDepth() <= max\_depth) {  
 try {  
 //Get the components of the URL link  
 String[] url\_components = non\_processed\_urls.getFirst().parse();  
 //Check if the protocol the link has is the one we are searching for  
 if (!url\_components[PROTOCOL].equalsIgnoreCase(PROTO\_SEARCH)) {  
 non\_processed\_urls.removeLast();  
 continue;  
 }  
 //Get the current depth of the crawl  
 int cur\_depth = non\_processed\_urls.getFirst().getDepth();  
 //Create a socket connection  
 Socket sock = new Socket(url\_components[HOSTNAME], webPort);  
 //Set the timeout to 10 seconds  
 sock.setSoTimeout(10000);  
  
 //Create the outputstream  
 OutputStream os = sock.getOutputStream();  
 //Get the printwriter from the os  
 PrintWriter writer = new PrintWriter(os, true);  
 //Create a request  
 writer.println("GET "+url\_components[RESOURCE]+" HTTP/1.1");  
 writer.println("Host: "+url\_components[HOSTNAME]);  
 writer.println("Connection: close");  
 writer.println();  
  
 //Prepare the response reader  
 InputStream is = sock.getInputStream();  
 InputStreamReader isr = new InputStreamReader(is);  
 BufferedReader br = new BufferedReader(isr);  
  
 while(true){  
 //Read the line of the html document  
 String line = br.readLine();  
 //Exit the loop if document ended  
 if (line == null)  
 break;  
  
 //Check whether the read line contains a link tag or redirect link  
 if (line.contains(HREF)) {  
 //Keep reading this line in a loop in case there is more than one link tag  
 while(line.contains(HREF)) {  
 //Get rid of whitespaces and get the entirety of the link  
 int line\_start = line.indexOf(HREF);  
 line = line.substring(line\_start + HREF.length());  
 line = line.substring(0,line.indexOf("\""));  
 //Create the pair and check if is contained in processed or to be processed lists  
 URLDepthPair new\_pair = new URLDepthPair(line, cur\_depth+1);  
 if (!found\_urls.contains(new\_pair) && !non\_processed\_urls.contains(new\_pair)) {  
 non\_processed\_urls.add(new\_pair);  
 }  
 }  
 }  
 }  
 //Add the searched link to the processed ones  
 found\_urls.add(non\_processed\_urls.getFirst());  
 //Remove processed link from the unprocessed ones  
 non\_processed\_urls.removeFirst();  
 //Close the connection  
 sock.close();  
 }  
 catch (IOException e) {  
 //If we timed out or received an improper response move onto next URL  
 System.*out*.println("Couldn't connect to the specified URL!");  
 System.*out*.println("Connecting to next URL...");  
 found\_urls.add(non\_processed\_urls.getFirst());  
 non\_processed\_urls.removeFirst();  
 }  
 }  
 return found\_urls;  
 }  
}

С данного класса начинается работа программы. Класс сделан многопоточным, в том плане, что если в командной строке было передано больше одного адреса, под каждый адрес выделяется отдельный поток для поиска. Суть данного класса состоит в подключении к адресу через TCP пакет, получить текстовый документ HTML и построчно его обработать, при этом находя теги, указывающие ссылки. Более подробно о работе можно узнать из комментариев к коду.

1. Выводы

Таким образом, были изучены основы сёрфинга в Интернете с помощью языка программирования Java.