**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Информатики

****

**Отчет по лабораторной работе № 8**

по дисциплине «КТП»

на тему:

«**Модифицированный веб-сканер»**

Выполнила: студентка группы БВТ1802

Лаврухина Елена Павловна

Руководитель:

Полянцева Ксения Андреевна

Москва 2020

Цель работы

Расширить сканер для использования поточной обработки.

Выполнение

Код программы

1. Crawler

import java.io.\*;  
import java.net.\*;  
import java.util.\*;  
public class Crawler {  
 public static void main(String[] args) {  
 String[] arg = new String[] {"http://www.mtuci.ru/","2","3"};  
 *// переменные для текущей глубины и запрошенного количества потоков* int depth = 0;  
 int numThreads = 0;  
 *// проверяем, правильное ли количество параметров было введено* if (arg.length != 3) {  
 System.*out*.println("Введите URL, глубину и количество потоков сканера");  
 System.*exit*(1);  
 }  
 else {  
 try {  
 depth = Integer.*parseInt*(arg[1]);  
 numThreads = Integer.*parseInt*(arg[2]);  
 } catch (NumberFormatException nfe) {  
 *// проверяем являются ли глубина и количество потоков цифрами* System.*out*.println("Введите URL, глубину и количество потоков сканера");  
 System.*exit*(1);  
 }  
 }  
 *// создаем глубину URL-пары для представления веб-сайта* URLDepthPair currentDepthPair = new URLDepthPair(arg[0], 0);  
 *// создаем URL пул и добавляем введенный пользователем веб-сайт* URLPool pool = new URLPool(depth);  
 pool.put(currentDepthPair);  
 *// поле для начального количества потоков* int initialActiveThreads = Thread.*activeCount*();  
 */\* пока количество ожидающих потоков не равно их запрошенному числу, и  
 если количество всех потоков меньше их запрошенного количества, то  
 создаём больше потоков и запускаем их на CrawlerTask. Иначе ждём. \*/* while (pool.getWaitThreads() != numThreads) {  
 if (Thread.*activeCount*() - initialActiveThreads < numThreads) {  
 CrawlerTask crawler = new CrawlerTask(pool);  
 new Thread(crawler).start();  
 }  
 else {  
 try {  
 Thread.*sleep*(500); *// 0,5 секунды* }  
 *// ловим исключение для прерывания потока* catch (InterruptedException ie) {  
 System.*out*.println("Произошло прерывание потока" + " игнорируем...");  
 }  
 }  
 }  
 *// выводим все обработанные URL с глубиной* Iterator<URLDepthPair> iter = pool.processedURLs.iterator();  
 String temp = "";  
 while (iter.hasNext()) {  
 URLDepthPair s = iter.next();  
 temp+= temp.contains(s.toString()) ? "" : s.toString() + " ";  
 }  
 for (int i = 0; i < depth + 1; i++)  
 for (int j = 0; j < temp.length();)  
 {  
 if (temp.indexOf(String.*valueOf*(i) + "\t", j) != -1)  
 {  
 System.*out*.println(temp.substring(temp.indexOf(String.*valueOf*(i) + "\t", j),  
 temp.indexOf(" ", temp.indexOf(String.*valueOf*(i) + "\t", j))));  
 if (j != temp.indexOf(" ", temp.indexOf(String.*valueOf*(i) + "\t", j)))  
 j = temp.indexOf(" ", temp.indexOf(String.*valueOf*(i) + "\t", j));  
 else  
 break;  
 }  
 else  
 break;  
 }  
 *// выход* System.*exit*(0);  
 }  
 public static LinkedList<String> getAllLinks(URLDepthPair DepthPair) {  
 *// создаем связанный список LinkedList<String>, в котором будут храниться ссылки, что мы находим* LinkedList<String> URLs = new LinkedList<String>();  
 *// создаём новый сокет* Socket sock;  
 *// инициализируем новый сокет из строки String, содержащей имя хоста, и из номера порта, равного 80 (http)* try {  
 sock = new Socket(DepthPair.getWebHost(), 80);  
 }  
 *// ловим исключения неизвестного хоста и возвращаем пустой список* catch (UnknownHostException e) {  
 *//System.err.println("UnknownHostException: " + e.getMessage());* return URLs;  
 }  
 *// ловим исключения ввода-вывода и возвращаем пустой список* catch (IOException ex) {  
 *//System.err.println("IOException: " + ex.getMessage());* return URLs;  
 }  
 try {  
 *// устанавливаем таймаут сокета на 3 секунды* sock.setSoTimeout(3000);  
 }  
 *// ловим исключения сокета и возвращаем пустой список* catch (SocketException exc) {  
 System.*err*.println("SocketException: " + exc.getMessage());  
 return URLs;  
 }  
 *// поля для хранения пути к файлу и хоста* String docPath = DepthPair.getDocPath();  
 String webHost = DepthPair.getWebHost();  
 *// создаем поток вывода* OutputStream outStream;  
 *// возвращаем OutputStream связанный с сокетом используемый для передачи данных* try {  
 outStream = sock.getOutputStream();  
 }  
 *// ловим исключения ввода-вывода и возвращаем пустой список* catch (IOException exce) {  
 *//System.err.println("IOException: " + exce.getMessage());* return URLs;  
 }  
 *// создаем PrintWriter, true означает, что PrintWriter будет сбрасываться после каждого вывода* PrintWriter pw = new PrintWriter(outStream, true);  
 *// отправляем запрос на сервер* pw.println("GET " + docPath + " HTTP/1.1");  
 pw.println("Host: " + webHost);  
 pw.println("Connection: close");  
 pw.println();  
 *// создаем поток ввода* InputStream inStream;  
 *// возвращает InputStream, связанный с объектом Socket, используемого для приема данных* try {  
 inStream = sock.getInputStream();  
 }  
 *// ловим исключения ввода-вывода и возвращаем пустой список* catch (IOException excep) {  
 *//System.err.println("IOException: " + excep.getMessage());* return URLs;  
 }  
 *// создаем новый InputStreamReader и BufferedReader для чтения строк с сервера* InputStreamReader in = new InputStreamReader(inStream);  
 BufferedReader br = new BufferedReader(in);  
 while (true) {  
 String line;  
 try {  
 line = br.readLine();  
 }  
 *// ловим исключения ввода-вывода и возвращаем пустой список* catch (IOException except) {  
 System.*err*.println("IOException: " + except.getMessage());  
 return URLs;  
 }  
 *// прекращаем чтения документа* if (line == null)  
 break;  
 *// переменные для индексов* int beginIndex;  
 int endIndex;  
 int index = 0;  
 while (true) {  
 *// константа для строки, указывающей на ссылку* String URL\_PREFIX = "<a href=\"http";  
 int p = "<a href=\"".length();  
 *// константа для строки, указывающей конец веб-хоста* String END\_URL = "\"";  
 *// поиск нашего начала в текущей строке* index = line.indexOf(URL\_PREFIX, index);  
 if (index == -1)  
 break;  
 *// двигаем текущий индекс и устанавливаем в beginIndex* index += p;  
 beginIndex = index;  
 *// ищем наш конец в текущей строке и устанавливаем в endIndex* endIndex = line.indexOf(END\_URL, index);  
 index = endIndex;  
 *// устанавливаем ссылку на подстроку между начальным и конечным индексом, добавляем к нашему списку URL* try {  
 String newLink = line.substring(beginIndex, endIndex);  
 URLs.add(newLink);  
 }  
 *// ловим исключения несуществующего индекса в массиве* catch (IndexOutOfBoundsException ignored)  
 {  
 break;  
 }  
 }  
 }  
 *// возвращаем список URL* return URLs;  
 }  
}

1. CrawlerTask

import java.util.LinkedList;  
public class CrawlerTask implements Runnable {  
 *// поле для заданной пары глубин* public URLDepthPair depthPair;  
 *// поле для URL пула* public URLPool pool;  
 *// конструктор для инициализации пула* public CrawlerTask (URLPool newPool) {  
 pool = newPool;  
 }  
 *// метод для запуска задач в CrawlerTask* public void run() {  
 *// достаём из пула следующую пару* depthPair = pool.get();  
 if (depthPair != null)  
 {  
 int depth = depthPair.getDepth();  
 *// поиск всех ссылок на рассматриваемом сайте и сохранение их в linksList* LinkedList<String> linksList;  
 linksList = Crawler.*getAllLinks*(depthPair);  
 *// перебираем ссылки с сайта* for (int i = 0; i < linksList.size(); i++) {  
 String newURL = linksList.get(i);  
 *// создаем новую пару для каждой ссылки и добавляем её в пул* URLDepthPair newDepthPair = new URLDepthPair(newURL, depth + 1);  
 pool.put(newDepthPair);  
 }  
 }  
 }  
}

1. URLDepthPair

import java.net.\*;  
public class URLDepthPair {  
 *// поля для хранения текущего URL и текущей глубины* private String currentURL;  
 private int currentDepth;  
 *// конструктор, который устанавливает ввод для текущих URL и глубины* public URLDepthPair(String URL, int depth) {  
 currentURL = URL;  
 currentDepth = depth;  
 }  
 *// метод, который возвращает текущий URL* public String getURL() {  
 return currentURL;  
 }  
 *// метод, который возвращает текущую глубину* public int getDepth() {  
 return currentDepth;  
 }  
 *// метод, который возвращает текущий URL и текущую глубину в строковом формате* public String toString() {  
 String stringDepth = Integer.*toString*(currentDepth);  
 return stringDepth + '\t' + currentURL;  
 }  
 *// метод, который возвращает путь документа текущего URL* public String getDocPath() {  
 try {  
 URL url = new URL(currentURL);  
 return url.getPath();  
 }  
 catch (MalformedURLException e) {  
 *//System.err.println("MalformedURLException: " + e.getMessage());* return null;  
 }  
 }  
 *// метод, который возвращает хост текущего URL* public String getWebHost() {  
 try {  
 URL url = new URL(currentURL);  
 return url.getHost();  
 }  
 catch (MalformedURLException e) {  
 *//System.err.println("MalformedURLException: " + e.getMessage());* return null;  
 }  
 }  
}

1. URLPool

import java.util.\*;  
public class URLPool {  
 *// список для не просмотренных пар <URL, глубина>* private LinkedList<URLDepthPair> pendingURLs;  
 *// список для просмотренных пар <URL, глубина>* public LinkedList<URLDepthPair> processedURLs;  
 *// список для просмотренных URL* private ArrayList<String> seenURLs = new ArrayList<>();  
 *// количество потоков, которые ожидают обработки* public int waitingThreads;  
 int maxDepth;  
 *// конструктор для инициализации waitingThreads, processedURLs и pendingURLs* public URLPool(int depthPair) {  
 maxDepth = depthPair;  
 waitingThreads = 0;  
 pendingURLs = new LinkedList<>();  
 processedURLs = new LinkedList<>();  
 }  
 *// метод для доступа к waitingThreads* public synchronized int getWaitThreads() {  
 return waitingThreads;  
 }  
 *// метод для получения размера pendingURLs* public synchronized int size() {  
 return pendingURLs.size();  
 }  
 *// метод для добавления новой пары <URL, глубина>* public synchronized void put(URLDepthPair depthPair) {  
 *// если был вызван put и есть потоки, находящиеся в ожидании, то надо вызвать эти потоки и уменьшить их счётчик* if (waitingThreads != 0) {  
 --waitingThreads;  
 this.notify();  
 }  
 if (!seenURLs.contains(depthPair.getURL()) &  
 !pendingURLs.contains(depthPair)) {  
 if (depthPair.getDepth() < maxDepth) {  
 pendingURLs.add(depthPair);  
 }  
 else {  
 processedURLs.add(depthPair);  
 seenURLs.add(depthPair.getURL());  
 }  
 }  
 }  
 *// метод для получения следующей пары из пула* public synchronized URLDepthPair get() {  
 URLDepthPair DepthPair = null;  
 *// пока пул пуст, ждем* if (pendingURLs.size() == 0) {  
 waitingThreads++;  
 try {  
 this.wait();  
 }  
 *// ловим исключение для прерывания потока* catch (InterruptedException e) {  
 System.*err*.println("MalformedURLException: " + e.getMessage());  
 return null;  
 }  
 }  
 *// удаляем первую пару, добавляем к просмотренным и обработанным URL, и возвращаем* if (!pendingURLs.isEmpty()) {  
 DepthPair = pendingURLs.getFirst();  
 pendingURLs.removeFirst();  
 seenURLs.add(DepthPair.getURL());  
 processedURLs.add(DepthPair);  
 }  
 return DepthPair;  
 }  
}

Скриншоты работы программы



Рисунок 1 – Работа программы с сайтом <http://www.mtuci.ru>



Рисунок 2 – Работа программы с сайтом <http://www.abitura.com/>

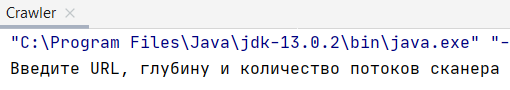


Рисунок 3 – Работа программы при неправильно введенных данных

Вывод

В ходе работы мы расширили сканер для использования поточной обработки Java так, чтобы несколько веб-страниц можно было сканировать параллельно.