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 №7

«Лучший Web Crawler»

Выполнил:

Студент 2 курса

Группы БСТ-1602

Смышляев Владислав

Вариант №15

Москва, 2018

Содержание

[1. Цель и задачи 3](#_Toc512790659)

[2. Анализ предметной области, выбор инструментария 3](#_Toc512790660)

[3. Объяснение функций 3](#_Toc512790661)

[3.1. Класс Crawler 3](#_Toc512790662)

[3.2. Класс URLPool 4](#_Toc512790663)

[3.3. Класс CrawlerTask 5](#_Toc512790664)

[4. Выводы 7](#_Toc512790665)

1. Цель и задачи

Цель работы: Изучить способы написание программы, сделав её многопоточной, используя стандартные встроенные средства языка Java.

Задачи: Переписать Web Crawler, который, используя несколько потоков, будет выполнять поиск по одной заданной начальной ссылке.

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

Так как работа основана на предыдущей будут указаны только модифицированные части кода.

* 1. Класс Crawler

import java.util.LinkedList;  
  
public class Crawler {  
 public static int *max\_depth*;  
  
 public static void main(String[] args){  
 if (args.length < 3){  
 System.*out*.println("Not enough arguments provided. URL, maximum crawl depth and thread number is required!");  
 return;  
 }  
 //Set the max depth of crawling  
 *max\_depth* = Integer.*parseInt*(args[1]);  
 //Set the amount of threads to use  
 int threads = Integer.*parseInt*(args[2]);  
 //Set the site to crawl through  
 URLPool pool = new URLPool(args[0]);  
 //Init thread array  
 Thread[] thrds = new Thread[threads];  
 //Start all the threads  
 for(int i=0;i<threads;i++){  
 CrawlerTask task = new CrawlerTask(pool);  
 thrds[i] = new Thread(task,"Crawler "+i);  
 thrds[i].start();  
 }  
 //Check for threads for their work being done  
 while (pool.sleep\_threads < threads) {  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 break;  
 }  
 }  
 //Make threads quit their work since there is none left  
 pool.exitThreads();  
 for(int i=0;i<threads;i++){  
 thrds[i].interrupt();  
 }  
 //Get all the links that threads generated and output them  
 LinkedList<URLDepthPair> sites = pool.getSites();  
 while(!sites.isEmpty()) {  
 System.*out*.println(sites.removeFirst());  
 }  
 }  
}

Класс Crawler теперь выполняет только исполнительную функцию. Запускает потоки для выполнения задачи, а после выводит полученные результаты.

* 1. Класс URLPool

import java.util.LinkedList;  
  
public class URLPool {  
 private LinkedList<URLDepthPair> pending\_urls;  
 private LinkedList<URLDepthPair> scanned\_urls;  
 public int sleep\_threads;  
 private boolean bKillThreads = false;  
  
 URLPool(String url){  
 pending\_urls = new LinkedList<>();  
 pending\_urls.add(new URLDepthPair(url,0));  
 scanned\_urls = new LinkedList<>();  
 sleep\_threads = 0;  
 }  
  
 */\*\*  
 \** ***@return*** *pair - returns a URLDepthPair from the pending urls list  
 \*/* public synchronized URLDepthPair fetch(){  
 //If pending list is found empty go sleep until more links are found  
 while(pending\_urls.isEmpty()){  
 try {  
 sleep\_threads++;  
 wait();  
 sleep\_threads--;  
 }  
 catch (InterruptedException e) {  
 if (bKillThreads)  
 return null;  
 }  
 }  
 URLDepthPair pair = pending\_urls.getFirst();  
 scanned\_urls.add(pair);  
 pending\_urls.removeFirst();  
 return pair;  
 }  
  
 */\*\*  
 \*  
 \** ***@param*** *pair - new url to add to crawling  
 \*/* public synchronized void add(URLDepthPair pair) {  
 //Check whether this link was already checked or is in the pending list already  
 if (scanned\_urls.contains(pair) || pending\_urls.contains(pair)) {  
 return;  
 }  
 //Check for crawl depth  
 if (pair.getDepth() >= Crawler.*max\_depth*) {  
 scanned\_urls.add(pair);  
 return;  
 }  
 //Add it to pending list and wake up a sleeping thread to start working again  
 pending\_urls.add(pair);  
 notify();  
 }  
  
 */\*\*  
 \*  
 \** ***@return*** *- list of sites that were crawled through  
 \*/* public LinkedList<URLDepthPair> getSites(){  
 return scanned\_urls;  
 }  
  
 */\*\*  
 \* Set the flag for threads to exit  
 \*/* public void exitThreads() {  
 bKillThreads = true;  
 }  
  
 */\*\*  
 \* Check for thread exit flag  
 \** ***@return*** *- state of flag  
 \*/* public boolean isbKillThreads() {  
 return bKillThreads;  
 }  
}

Данный класс отвечает за синхронность всех потоков, чтобы никто из них случайно вместе не пытались получить доступ к одним и тем же данным. Также сообщает потокам когда нужно завершать работу.

* 1. Класс CrawlerTask

import java.io.\*;  
import java.net.Socket;  
  
public class CrawlerTask implements Runnable {  
 private final String HREF = "<a href=\"";  
 private final int webPort = 80;  
 private URLPool pool;  
 private final int PROTOCOL = 0;  
 private final int HOSTNAME = 1;  
 private final int RESOURCE = 2;  
  
 CrawlerTask(URLPool pool){  
 this.pool = pool;  
 }  
  
 */\*\*  
 \* Main method where all the threads are performing the crawling task.  
 \* Links are being fetched until pool says to stop or until they get  
 \* nothing (null) from the fetch method  
 \*/* @Override  
 public void run() {  
 while(!pool.isbKillThreads()) {  
 try {  
 //Get the pending link  
 URLDepthPair pair = pool.fetch();  
 //If no actual link was given that means we can stop working  
 if (pair == null) break;  
 //Get the url\_components  
 String[] url\_components = pair.parse();  
 //Get the current depth  
 int cur\_depth = pair.getDepth();  
 //Connect to the site and start getting data  
 Socket sock = new Socket(url\_components[HOSTNAME], webPort);  
 System.*out*.println("Attempting to connect to "+url\_components[HOSTNAME]);  
 //Set the timeout (10 seconds)  
 sock.setSoTimeout(10000);  
  
 OutputStream os = sock.getOutputStream();  
  
 PrintWriter writer = new PrintWriter(os, true);  
 //Send the request  
 writer.println("GET "+url\_components[RESOURCE]+" HTTP/1.1");  
 writer.println("Host: "+url\_components[HOSTNAME]);  
 writer.println("Connection: close");  
 writer.println();  
  
 InputStream is = sock.getInputStream();  
 InputStreamReader isr = new InputStreamReader(is);  
 BufferedReader br = new BufferedReader(isr);  
  
 while(true){  
 String line = br.readLine();  
 if (line == null)  
 break;  
 if (line.contains(HREF)) {  
 //Read the line for every link  
 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("\""));  
 //Add the link to the pool  
 pool.add(new URLDepthPair(line, cur\_depth+1));  
 }  
 }  
 }  
 sock.close();  
 }  
 catch (IOException e) {  
 System.*out*.println("Couldn't connect to the specified URL!");  
 System.*out*.println("Connecting to next URL...");  
 }  
 }  
 }  
}

Данный класс является реализацией потока, который и выполняет весь поиск в Интернете, он производит попытку получения нового адреса, к которому требуется подключиться. В случае его отсутствия поток временно останавливает выполнения, до получения нового адреса, если в ответ было получено ничего поток завершает работу.

1. Выводы

Таким образом, были изучены основы многопоточности и его использования в языке программирования Java.