МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Знамени федеральное государственное

Бюджетное образовательное учреждение высшего образования

МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И

ИНФОРМАТИКИ

Кафедра математической кибернетики и информационных технологий

Лабораторная работа № 8 по теме:

«Модифицированный веб-сканер»

Выполнил: студент группы БИБ1902 \_\_\_\_\_\_\_Арустамян А.Б.

Проверил: \_\_\_\_\_\_\_\_Херсонский А. В.

Москва 2021

**Цель работы:**

В данной лабораторной работе расширен сканер для использования поточной обработки Java так, чтобы несколько веб-страниц можно было сканировать параллельно. Это приведет к значительному повышению производительности, так как время, которое каждый поток сканера тратит на ожидание завершения сетевых операций, может прерываться другими операциями обработки в других потоках.

**Код программы:**

**Main.java**

package com.company;  
  
import java.io.IOException;  
import java.util.Iterator;  
  
public class Main {  
 public static void main(String[] args) throws IOException {  
  
 String[] arg = new String[] {"https://vk.com/","1","1"};  
 int depth = 0;  
 int numThreads = 0;  
 if (arg.length != 3) {  
  
 System.*out*.println("usage: java Crawler <URL> <depth> "  
 + "<number of crawler threads>");  
 System.*exit*(1);  
 }  
 try {  
 depth = Integer.*parseInt*(arg[1]);  
 numThreads = Integer.*parseInt*(arg[2]);  
 }  
 catch (NumberFormatException nfe) {  
 System.*out*.println("usage: java Crawler <URL> <depth> "  
 + "<number of crawler threads>");  
 System.*exit*(1);  
 }  
 URLDepthPair currentDepthPair = new URLDepthPair(arg[0], 0);  
 URLPool pool = new URLPool(depth);  
 pool.put(currentDepthPair);  
 int initialActiveThreads = Thread.*activeCount*();  
 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  
 }  
 catch (InterruptedException ie) {  
 System.*out*.println("Caught unexpected InterruptedException,"  
 + " ignoring...");  
 }  
 }  
 }  
 Iterator<URLDepthPair> iter = pool.processedURLs.iterator();  
 while (iter.hasNext()) {  
 System.*out*.println(iter.next());  
 }  
 System.*exit*(0);  
 }  
}

***Crawler.java***

package com.company;  
  
import java.io.\*;  
import java.net.\*;  
import java.util.\*;  
import java.util.regex.\*;  
public class Crawler {  
  
 public static LinkedList<String> getAllLinks(URLDepthPair myDepthPair) throws IOException {  
 LinkedList<String> URLs = new LinkedList<String>();  
 Socket sock;  
 try {  
 sock = new Socket(myDepthPair.getWebHost(), 80);  
 }  
 catch (UnknownHostException e) {  
 System.*err*.println("UnknownHostException: " + e.getMessage());  
 return URLs;  
 }  
 catch (IOException ex) {  
 return URLs;  
 }  
 try {  
 sock.setSoTimeout(3000);  
 }  
 catch (SocketException exc) {  
 System.*err*.println("SocketException: " + exc.getMessage());  
 return URLs;  
 }  
 String docPath = myDepthPair.getDocPath();  
 String webHost = myDepthPair.getWebHost();  
 OutputStream outStream;  
 try {  
 outStream = sock.getOutputStream();  
 }  
 catch (IOException exce) {  
 return URLs;  
 }  
 PrintWriter myWriter = new PrintWriter(outStream, true);  
 if (docPath.length() == 0) {  
 myWriter.println("GET / HTTP/1.1");  
 myWriter.println("Host: " + webHost);  
 myWriter.println("Connection: close");  
 myWriter.println();  
 }  
 else {  
 myWriter.println("GET " + docPath + " HTTP/1.1");  
 myWriter.println("Host: " + webHost);  
 myWriter.println("Connection: close");  
 myWriter.println();  
 }  
 InputStream inStream;  
 try {  
 inStream = sock.getInputStream();  
 }  
 catch (IOException excep) {  
 return URLs;  
 }  
 InputStreamReader inStreamReader = new InputStreamReader(inStream);  
 BufferedReader BuffReader = new BufferedReader(inStreamReader);  
 int serverCode = 0;  
 String lineCode;  
 try {  
 lineCode = BuffReader.readLine();  
 }  
 catch (IOException except) {  
 return URLs;  
 }  
 Pattern patternCode = Pattern.*compile*("(2|3|4)[0-9]{2}");  
 Matcher matcherCode = patternCode.matcher(lineCode);  
 while (matcherCode.find()) {  
 serverCode = Integer.*parseInt*(lineCode.substring(matcherCode.start(), matcherCode.end() - 2));  
 }  
 if (serverCode == 2) {  
 while (true) {  
 String line;  
 try {  
 line = BuffReader.readLine();  
 }  
 catch (IOException except) {  
 return URLs;  
 }  
 if (line == null) {  
 break;  
 }  
 Pattern patternURL = Pattern.*compile*(  
 "[\"]"  
 + "[https?://]{7,8}"  
 + "([w]{3})?"  
 + "[\\w\\.\\-]+"  
 + "\\."  
 + "[A-Za-z]{2,6}"  
 + "[\\w\\.-/]\*"  
 + "[\"]");  
 Matcher matcherURL = patternURL.matcher(line);  
 while (matcherURL.find()) {  
 String newLink = line.substring(matcherURL.start() + 1,  
 matcherURL.end() - 1);  
 URLs.add(newLink);  
 }  
 }  
 sock.close();  
 return URLs;  
 }  
 if (serverCode == 3) {  
 String newURL = "";  
 String tempLine;  
 while (true) {  
 try {  
 tempLine = BuffReader.readLine();  
 }  
 catch (IOException except) {  
 return URLs;  
 }  
 if (tempLine == null) {  
 break;  
 }  
 Pattern patternNewURL =Pattern.*compile*("(Location:){1}[\\S]+");  
 Matcher matcherNewURL = patternNewURL.matcher(tempLine);  
 while (matcherNewURL.find()) {  
 newURL = tempLine.substring(matcherNewURL.start() + 10,  
 matcherNewURL.end());  
 }  
 }  
 if (newURL.equals(myDepthPair.getURL())) {  
 sock.close();  
 return URLs;  
 }  
 URLDepthPair newDepthPair;  
 newDepthPair = new URLDepthPair(newURL, myDepthPair.getDepth());  
 return *getAllLinks*(newDepthPair);  
 }  
 return URLs;  
 }  
}

***URLDepthPair.java***

package com.company;  
  
import java.net.\*;  
public class URLDepthPair {  
  
 private final String currentURL;  
 private final int currentDepth;  
 public URLDepthPair(String URL, int depth) {  
 currentURL = URL;  
 currentDepth = depth;  
 }  
 public String getURL() {  
 return currentURL;  
 }  
 public int getDepth() {  
 return currentDepth;  
 }  
 public String toString() {  
 String stringDepth = Integer.*toString*(currentDepth);  
 return stringDepth + '\t' + currentURL;  
 }  
  
 public String getDocPath() {  
 try {  
 URL url = new URL(currentURL);  
 return url.getPath();  
 }  
 catch (MalformedURLException e) {  
 return null;  
 }  
 }  
 public String getWebHost() {  
 try {  
 URL url = new URL(currentURL);  
 return url.getHost();  
 }  
 catch (MalformedURLException e) {  
 return null;  
 }  
 }  
}

***CrawlerTask.java***

package com.company;  
  
import java.io.IOException;  
import java.util.LinkedList;  
import java.util.logging.\*;  
public class CrawlerTask implements Runnable {  
 public URLDepthPair depthPair;  
 public URLPool pool;  
 public CrawlerTask (URLPool newPool) {  
 pool = newPool;  
 }  
 public void run() {  
 depthPair = pool.get();  
 int depth = depthPair.getDepth();  
 LinkedList<String> linksList = null;  
 try {  
 linksList = Crawler.*getAllLinks*(depthPair);  
 }  
 catch (IOException ex) {  
 Logger.*getLogger*(CrawlerTask.class.getName()).log(Level.*SEVERE*,null, ex);  
 }  
 for (int i = 0; i < linksList.size(); i++) {  
 String newURL = linksList.get(i);  
 URLDepthPair newDepthPair = new URLDepthPair(newURL, depth + 1);  
 pool.put(newDepthPair);  
 }  
 }  
}

***URLPool.java***

package com.company;  
  
import java.util.\*;  
public class URLPool {  
 private final LinkedList<URLDepthPair> pendingURLs;  
 public LinkedList<URLDepthPair> processedURLs;  
 private final ArrayList<String> seenURLs = new ArrayList<>();  
 public int waitingThreads;  
 int maxDepth;  
 public URLPool(int maxDepthPair) {  
 maxDepth = maxDepthPair;  
 waitingThreads = 0;  
 pendingURLs = new LinkedList<>();  
 processedURLs = new LinkedList<>();  
 }  
 public synchronized int getWaitThreads() {  
 return waitingThreads;  
 }  
 public synchronized int size() {  
 return pendingURLs.size();  
 }  
 public synchronized void put(URLDepthPair depthPair) {  
 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());  
 }  
 }  
 }  
 public synchronized URLDepthPair get() {  
 URLDepthPair myDepthPair;  
 while (pendingURLs.isEmpty()) {  
 waitingThreads++;  
 try {  
 this.wait();  
 }  
 catch (InterruptedException e) {  
 System.*err*.println("MalformedURLException: " + e.getMessage());  
 return null;  
 }  
 }  
 myDepthPair = pendingURLs.pop();  
  
 while (seenURLs.contains(myDepthPair.getURL())) {  
  
 myDepthPair = pendingURLs.pop();  
 }  
  
 processedURLs.add(myDepthPair);  
 seenURLs.add(myDepthPair.getURL());  
  
 return myDepthPair;  
 }  
}

**Вывод:** В данной ЛР — я доработал свой веб-сканер, сделав его многопоточным, благодаря чему он стал более производительным и увеличил эффективность своей работы.