Федеральное агентство связи

Ордена трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Отчет по лабораторной работе № 8 «Модифицированный веб-сканер»

по дисциплине «Объектно-ориентированное программирование»

Выполнил: студент группы БФИ1901

Кириллов Роман Сергеевич

Проверил: Мосева М.С

Москва, 2020

Задача: В данной лабораторной работе вы расширите и измените разработанную ранее программу:

1. Реализуйте класс с именем URLPool, который будет хранить список всех URL-адресов для поиска, а также относительный "уровень" каждого из этих URL-адресов (также известный как "глубина поиска"). Первый URL-адрес, который нужно будет найти, будет иметь глубину поиска равную 0, URLадреса, найденные на этой странице, будут иметь глубину поиска равную 1 и т.д. Необходимо сохранить URL-адреса и их глубину поиска вместе, как экземпляры класса с именем URLDepthPair, как это было сделано в прошлой лабораторной работе. LinkedList рекомендуется использовать для хранения элементов, так как это поможет эффективно выполнить необходимые операции. У пользователя класса URLPool должен быть способ получения пары URL-глубина из пула и удаления этой пары из списка одновременно. Должен также быть способ добавления пары URL-глубина к пулу. Обе эти операции должны быть поточно-ориентированы, так как несколько потоков будут взаимодействовать с URLPool одновременно.

У пула URL не должно быть максимального размера. Для этого нужен список необработанных URL-адресов, список уже отсканированных URLадресов и еще одно поле, о котором будет написано ниже.

2.Чтобы выполнить веб-сканирование в нескольких потоках, необходимо создать класс CrawlerTask, который реализует интерфейс Runnable. Каждый экземпляр CrawlerTask должна иметь ссылку на один экземпляр класса URLPool, который был описан выше. (Обратите внимание на то, что все экземпляры класса CrawlerTask используют единственный пул!) Принцип работы веб-сканера заключается в следующем: 1). Получение пары URL-Depth из пула, ожидая в случае, если пара не будет сразу доступна.2). Получение веб-страницы по URL-адресу. 3). Поиск на странице других URL-адресов. Для каждого найденного URL-адреса, необходимо добавить новую пару URL-Depth к пулу URL-адресов. Новая пара должна иметь глубину на единицу больше, чем глубина текущего URL-адреса, по которому происходит сканирование.4). Переход к шагу 1. Данный цикл должен продолжаться до тех пор, пока в пуле не останется пар URL-Depth.

3. Так как веб-сканер будет порождать некоторое количество потоков, измените программу так, чтобы она принимала третий параметр через командную строку, который будет определять количество порождаемых потоков веб-сканера. Функция main должна выполнять следующие задачи:

1). Обработать аргументы командной строки. Сообщить пользователю о любых ошибках ввода.

2). Создать экземпляр пула URL-адресов и поместить указанный пользователем URL-адрес в пул с глубиной 0.

3). Создать указанное пользователем количество задач (и потоков для их выполнения) для веб-сканера. Каждой задаче поискового робота нужно дать ссылку на созданный пул URL-адресов.

4). Ожидать завершения веб-сканирования.

5) Вывести получившийся список URL-адресов, которые были найдены.

4.Синхронизируйте объект пула URL-адресов во всех критических точках, так как теперь код должен быть ориентирован на многопоточность.

5. Веб-сканер не должен постоянно опрашивать пул URL-адресов в случае, если он пуст. Вместо этого пусть они ожидают в случае, когда нет доступных URL-адресов. Реализуйте это, используя метод wait() внутри «get URL» в случае, если ни один URL-адрес в настоящее время недоступен. Соответственно, метод "add URL" пула URL-адресов должен использовать функцию notify() в случае, когда новый URL-адрес добавлен к пулу. Обратите внимание на то, что потоки веб-сканера не сами будут выполнять какие-либо из этих операций синхронизации/ожидания/уведомления. По той же причине, что и пул URLадресов скрывает детали того, как URL-адреса хранятся и извлекаются: инкапсуляция! Точно так же, как и в вашей реализации пользователи пула URL-адресов не должны знать о деталях реализации, также они не должны знать о деталях организации потоков.

Ход работы:

Класс UrlDepthPair.java

import java.net.\*;  
public class URLDepthPair {  
 private int depth;  
 private String url = "";  
  
 URLDepthPair(String u, int d){  
 url = u;  
 depth = d;  
 }  
 URLDepthPair(String u){  
 url = u;  
 depth = 0;  
 }  
  
 public String getURL() {  
 return url;  
 }  
 public int getDepth() {  
 return depth;  
 }  
 public String toString() {  
 return url + " : " + depth;  
 }  
  
 public String getDocPath() {  
 try {  
 URL tempURL = new URL(url);  
 return tempURL.getPath();  
 }  
 catch (MalformedURLException malformedURLException) {  
 System.*err*.println("MalformedURLException in getDocPath(): " + malformedURLException.getMessage());  
 return null;  
 }  
 }  
  
 public String getWebHost() {  
 try {  
 URL tempURL = new URL(url);  
 return tempURL.getHost();  
 }  
 catch (MalformedURLException malformedURLException) {  
 System.*err*.println("MalformedURLException in getWebHost: " + malformedURLException.getMessage());  
 return null;  
 }  
 }  
}

Crawler.java

import java.util.\*;  
import java.io.\*;  
import java.net.\*;  
  
public class Crawler {  
 static int *numThreads* = 10;   
 static int *timeOut* = 1000;   
 public static void main(String[] args) {  
  
 URLDepthPair currentDepthPair = new URLDepthPair("http://ntsk.ru/");  
 ArrayList<String> seenURLs = new ArrayList<String>();  
 seenURLs.add(currentDepthPair.getURL());  
  
 URLPool pool = new URLPool(2); pool.put(currentDepthPair);  
  
 int activeAmount = Thread.*activeCount*();  
  
 while (pool.getWaitThreads() != *numThreads*) {  
  
 if (Thread.*activeCount*() - activeAmount < *numThreads*) {  
 CrawlerTask crawler = new CrawlerTask(pool);  
 new Thread(crawler).start();  
 }  
 else {  
 try {  
 Thread.*sleep*(100);  
 }  
 catch (InterruptedException ie) {  
 System.*out*.println("Caught unexpected: InterruptedException, ignoring...");  
 }  
  
 }  
 }  
 for(String s : pool.seenURLs) {  
 System.*out*.println(s);  
 }  
  
 System.*out*.println(pool.seenURLs.size());  
 System.*exit*(0);  
  
 }  
  
 public static LinkedList<String> getAllLinks(URLDepthPair myDepthPair) {  
  
 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(*timeOut*);  
 }  
 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 e) {  
 return URLs;  
 }  
  
 PrintWriter printWriter = new PrintWriter(outStream, true);  
 printWriter.println("GET " + docPath + " HTTP/1.1");  
 printWriter.println("Host: " + webHost);  
 printWriter.println("Connection: close");  
 printWriter.println();  
  
 InputStream inStream;  
 try {  
 inStream = sock.getInputStream();  
 }  
 catch (IOException ioException){  
 System.*err*.println("IOException: " + ioException.getMessage());  
 return URLs;  
 }  
 InputStreamReader inStreamReader = new InputStreamReader(inStream);  
 BufferedReader BuffReader = new BufferedReader(inStreamReader);  
  
 while (true) {  
 String line;  
 try {  
 line = BuffReader.readLine();  
 }  
 catch (IOException e) {  
 return URLs;  
 }  
 if (line == null)  
 break;  
 int beginIndex = 0;  
 int endIndex = 0;  
 int index = 0;  
  
 while (true) {  
 String START\_URL = "a href=\"";  
 String END\_URL = "\"";  
  
 index = line.indexOf(START\_URL, index);  
 if (index == -1)  
 break;  
 index += START\_URL.length();  
 beginIndex = index;  
  
 endIndex = line.indexOf(END\_URL, index);  
 index = endIndex;  
  
 try {  
 String newLink = line.substring(beginIndex, endIndex);  
 if(URLs.contains(newLink))  
 continue;  
  
 if(newLink.startsWith("http")) {  
 URLs.add(newLink);  
 }else if(!newLink.startsWith("tel")) {  
 if(newLink.startsWith("/"))  
 URLs.add("http://"+webHost+""+newLink);  
 else  
 URLs.add("http://"+webHost+"/"+newLink);  
 }  
  
 }catch(Exception exception) {  
 break;  
 }  
  
 }  
  
 }  
 return URLs;  
 }  
}

CrawlerTask.java

import java.util.\*;  
  
public class CrawlerTask implements Runnable {  
 public URLDepthPair depthPair;  
  
 public URLPool myPool;  
  
 public CrawlerTask(URLPool pool) {  
 myPool = pool;  
 }  
   
 public void run() {   
 depthPair = myPool.get();  
  
 int myDepth = depthPair.getDepth();  
  
 LinkedList<String> linksList = new LinkedList<String>();  
 linksList = Crawler.*getAllLinks*(depthPair);   
 for (String newURL : linksList) {  
 URLDepthPair newDepthPair = new URLDepthPair(newURL, myDepth + 1);  
 myPool.put(newDepthPair);  
 }  
 }  
}

Класс UrlPool.java

import java.util.\*;  
public class URLPool {  
 public static int *max\_depth*;  
 private LinkedList<URLDepthPair> pendingURLs;  
 public LinkedList<URLDepthPair> processedURLs;  
 public ArrayList<String> seenURLs = new ArrayList<String>();  
  
 public int waitingThreads;  
  
 public URLPool(int d) {  
 waitingThreads = 0;  
 pendingURLs = new LinkedList<URLDepthPair>();  
 processedURLs = new LinkedList<URLDepthPair>();  
 *max\_depth* = d;  
 }  
   
 public synchronized int getWaitThreads() {  
 return waitingThreads;  
 }  
  
 public synchronized int size() {  
 return pendingURLs.size();  
 }  
  
 public synchronized void decrimentWaitingThreads() {  
 waitingThreads--;  
 }  
  
 public synchronized boolean put(URLDepthPair depthPair) {  
 boolean isAdded = false;  
  
 if (depthPair.getDepth() < *max\_depth* && !processedURLs.contains(depthPair.getURL())) {  
 pendingURLs.addLast(depthPair);  
 isAdded = true;  
 this.notify();  
 } else {  
 addSeenURL(depthPair);  
 }  
  
 return isAdded;  
 }  
  
 public synchronized URLDepthPair get() {  
  
 URLDepthPair depthPair = null;  
 if (pendingURLs.size() == 0) {  
 waitingThreads++;  
 try {  
 this.wait();  
 }  
 catch (InterruptedException interruptedException) {  
 System.*err*.println("InterruptedException: " + interruptedException.getMessage());  
 return null;  
 }  
 }  
 if(waitingThreads>0)waitingThreads--;  
 depthPair = pendingURLs.pop();  
 addSeenURL(depthPair);  
 processedURLs.add(depthPair);  
 return depthPair;  
 }  
  
 private synchronized void addSeenURL(URLDepthPair dp) {  
 if(!seenURLs.contains(dp.toString()))  
 seenURLs.add(dp.toString());  
 }  
  
 public synchronized ArrayList<String> getSeenList() {  
 return seenURLs;  
 }  
}

Результат работы программы:

