**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

отчет

**по курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Сбор новостей**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9391, 9392 |  | Кукар Н.М.  Макаренко Г.А.  Матыгин В.С. |
| Преподаватель |  | Спиридонов Р.Е. |

Санкт-Петербург

2020

**Задание:**

Задача сервиса - выполнять две основных операции: первичное наполнение и наблюдение за новостными или статейными сайтами.

**Первичное наполнение**

Пройти все страницы сайта и получить с них статьи с их заголовками, содержимым, датами публикации, тегами и/или категориями.

**Наблюдение**

Раз в заданный в параметрах период времени (от 1 минуты до 4 часов, можно задавать в минутах) заходить на сайт и добавлять в базу материалы, которых не было раньше на сайте.

При этом сама база может быть оформлена в виде SQL базы данных (ставится легко и подключается работа с одной-двумя таблицами), nosql базы данных, вроде sqlite, например, или же вообще без базы данных, а реализовать хранение в виде json файлов, где каждый отдельный файл - это одна статья. В названии файлов при этом или ID статьи с сайта (если его можно оттуда вытащить) или же часть URL, обозначающего статью.

**Выполнение задания:**

Курсовая работа написана на языке программирования JAVA. Для выполнения курсовой работы необходимо подключение библиотеки jsoup. [Jsoup](https://jsoup.org/)- это библиотека Java с открытым исходным кодом, используемая в основном для извлечения данных из HTML. Это также позволяет вам манипулировать и выводить HTML. Он имеет устойчивую линию разработки, отличную документацию, а также гибкий и гибкий API. Jsoup также может быть использован для анализа и сборки XML. Для выполнения задания, в качестве примера был предоставлен сайт [rb.ru](https://rb.ru/). У данного сайта присутствует Sitemap(карта сайта), с помощью которой можно легко получать ссылки, по которым можно переходить и выбирать нужную информацию с помощью кода элемента страницы. Для того, чтобы работать с jsoup, создадим проект maven. Maven — мощный инструмент сборки для Java-проектов. Это инструмент управления и автоматизации. Он в основном написан на языке Java и используется для создания и управления проектами, написанными на C #, Ruby, Scala и других языках. С помощью репозитория maven загрузим последнюю версию jsoup и SQLlite для создания базы данных.

<dependency>  
 <groupId>org.jsoup</groupId>  
 <artifactId>jsoup</artifactId>  
 <version>1.13.1</version>  
</dependency>  
<!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc -->  
<dependency>  
 <groupId>org.xerial</groupId>  
 <artifactId>sqlite-jdbc</artifactId>  
 <version>3.34.0</version>

После этого начнем писать код нашей программы:

Создадим класс Parser и в нём напишем функцию open, которая позволит подключиться нам к базе дынных SQLite и метод close для закрытия соединения:

public class Parser {  
 public static Connection *co*;  
  
  
 public static void open() throws ClassNotFoundException, SQLException {  
 Class.*forName*("org.sqlite.JDBC");  
 *co* = DriverManager.*getConnection*("jdbc:sqlite:parsering.db");  
 }

public static void close() throws SQLException {  
 *co*.close();  
}

Для удобства использования переменной co, класса Connection, который присутствует в пакете java.sql, сделаем ее публичной и статической.

Также в программе присутствует метод select, который получает информацию из базы данных и выводит её в консоль среды разработки, но это не необходимый метод, исключительно для проверки работоспособности программы на ранних этапах написания кода:

public static void select() throws SQLException {  
 Statement statement = *co*.createStatement();  
 String query = "SELECT id, url, name, header, date, article FROM parsering ";  
 ResultSet rs = statement.executeQuery(query);  
 while (rs.next()) {  
 int id = rs.getInt("id");  
 String url = rs.getString("url");  
 String name = rs.getString("name");  
 String header = rs.getString("header");  
 String date = rs.getString("date");  
 String article = rs.getString("article");  
 System.*out*.println(id + "\t|" + url + "\t|" + name + "\t|" + header + "\t|" + date + "\t|" + article);  
 }  
 rs.close();  
 statement.close();  
}

Напишем метод, который будет создавать таблицу, если её не существует:

public static void CreateTable() throws SQLException, ClassNotFoundException {  
 *open*();  
 Statement statement = *co*.createStatement();  
 String query = "CREATE TABLE if not exists 'parsering' ('id' INTEGER PRIMARY KEY AUTOINCREMENT, 'url' VARCHAR(50), 'name' VARCCHAR(50), 'header' VARCHAR(50), 'article')";  
 statement.executeQuery(query);  
}

Далее идёт основной метод нашей программы: getInfo, в котором мы заходим на основной sitemap сайта rb.ru, с помощью jsoup и класса document создаем соединение между сайтом и нашей программой, после чего считываем данные всех sitemap с ссылками, которые находятся внутри основной карты сайта. После этого мы помещаем sitemap с ссылками на все новости в цикл foreach и получаем ссылки на эти sitemap. После этого мы вновь помещаем полученные ссылки в цикл foreach для получения нужной нам информации. Далее с помощью класса Elements и его метода select, вытаскиваем нужную нам информацию через код элемента в браузере, копируя селектор. Нам нужна информация об авторе, заголовке страницы, времени публикации и содержании публикации. Также, для выполнения sql – запросов будем использовать класс Statement.

Statement используется для выполнения SQL-запросов. Существует три типа класса Statement, которые являются как бы контейнерами для выполнения SQL-выражений через установленное соединение:

* Statement, базовый;
* PreparedStatement, наследующий от Statement;
* CallableStatement, наследующий от PreparedStatement.
* public static void getInfo() throws IOException, SQLException {  
   String urls = "https://rb.ru/sitemap.xml";  
   Document page = Jsoup.*connect*(urls)  
   .userAgent("Chrome/4.0.249.0 Safari/532.5")  
   .referrer("http://www.google.com")  
   .get();  
    
   Elements pages = page.select("loc");  
   String str = pages.text();  
   String[] words = str.split(" ");  
   for (String word : words) {  
   Document pagein = Jsoup.*connect*(word)  
   .userAgent("Chrome/4.0.249.0 Safari/532.5")  
   .referrer("http://www.google.com")  
   .get();  
    
   Elements pagesin = pagein.select("loc");  
   String strin = pagesin.text();  
   String[] wordsin = strin.split(" ");  
    
   for (String wordin : wordsin) {  
   Document pagein1 = Jsoup.*connect*(wordin)  
   .userAgent("Chrome/4.0.249.0 Safari/532.5")  
   .referrer("http://www.google.com")  
   .get();  
    
   Elements pagesName = pagein1.select("#article-feed > div > section.article-header > div > div.article-header\_\_wrap.article-header\_\_wrap--l2 > div.article-header\_\_wrap.article-header\_\_wrap--author > div > span > a");  
   Elements pagesHeader = pagein1.select("#article-feed > div > section.article-header > div > div.article-header\_\_wrap.article-header\_\_wrap--rubric > h1");  
   Elements pagesDate = pagein1.select("#article-feed > div > section.article-header > div > div.article-header\_\_wrap.article-header\_\_wrap--l2 > div.article-header\_\_wrap.article-header\_\_wrap--header.article-header\_\_wrap--header-2 > time");  
   Elements pagesArticle = pagein1.select("#article-feed > div > section.article > div.article\_\_container.article\_\_container--main > div > div:nth-child(2) > div.article\_\_content-block.abv > p");  
   String query1 = "INSERT INTO parsering(url, name, header, date, article) SELECT DISTINCT '" + wordin + "', '"+pagesName.text()+"', '"+pagesHeader.text()+"', '"+pagesDate.text()+"', '"+pagesArticle.text()+"' FROM parsering WHERE NOT EXISTS (SELECT url FROM parsering WHERE url = '" + wordin + "')";  
    
   Statement statement = *co*.createStatement();  
   statement.executeUpdate(query1);  
   }  
   }  
   }

После напишем два метода, которые будут реализованы для того, чтобы пользователь мог выбирать, какой режим программы ему использовать: режим первичного заполнения, или режим наблюдения:

public static void filling() throws ClassNotFoundException, SQLException, IOException {  
 *open*();  
 Statement statement = *co*.createStatement();  
 String query = "INSERT INTO parsering (url, name, header, date, article) VALUES ('url','name', 'header', 'date', 'article')";  
 statement.executeUpdate(query);  
 *open*();  
 *getInfo*();  
 *select*();  
 *close*();  
}  
public static void observation() throws SQLException, IOException, ClassNotFoundException, InterruptedException {  
 boolean b = true;  
 Scanner scanner = new Scanner(System.*in*);  
 int id;  
 while(b){  
 System.*out*.println("Введите время (в минутах), через которое нужно будет вновь пропарсить страницу: ");  
 id = scanner.nextInt();  
 if(id>0){  
 Thread.*sleep*(60000 \* id);  
 *open*();  
 *getInfo*();  
 *close*();  
  
  
 }  
 else{  
 System.*out*.println("Program terminated");  
 b = false;  
 }  
 }  
  
}

Далее реализуем метод main. Создадим таблицу в базе данных с помощью метода CreateTable. Для того, чтобы пользователь мог осуществлять выбор, в каком режиме должна быть запущена программа, создадим конструкцию switch, и поместим её в бесконечный цикл для того, чтобы можно было перемещаться по меню:

public static void main(String[] args) throws IOException, ClassNotFoundException, SQLException, InterruptedException {  
 *CreateTable*();  
 System.*out*.println("Выберите один из режимов:" + "\n" + "1 - Режим первичного заполнения базы данных" + "\n" + "2 - Режим наблюдения" + "\n" + "0 - Выход их программы");  
 Scanner scanner = new Scanner(System.*in*);  
 int n = 1;  
 while(n==1 && n!=0) {  
 int id = scanner.nextInt();  
 switch (id) {  
 case (1):  
 *filling*();  
 break;  
 case (2):  
 *observation*();  
 break;  
 case (0):  
 break;  
 }  
 System.*out*.println("Для продолжения работы программы введите 1, если необходимо выйти - введите 0: ");  
 n = scanner.nextInt();  
 System.*out*.println("Выберите один из режимов:" + "\n" + "1 - Режим первичного заполнения базы данных" + "\n" + "2 - Режим наблюдения" + "\n" + "0 - Выход их программы");  
 }  
  
  
 }  
}

**Примечания к курсовой работе:**

Для выполнения условия, при котором было необходимо реализовать первичное заполнение так, чтобы после остановки программы она продолжала заполнять таблицу с того же места, был применён sql – запрос:

INSERT INTO parsering(url, name, header, date, article) SELECT DISTINCT '" + wordin + "', '"+pagesName.text()+"', '"+pagesHeader.text()+"', '"+pagesDate.text()+"', '"+pagesArticle.text()+"' FROM parsering WHERE NOT EXISTS (SELECT url FROM parsering WHERE url = '" + wordin + "'

Который позволяет нам добавлять только те значения в таблицу, которых ещё не существует в ней (проверяется ссылка на статью). С помощью этого запроса также можно легко реализовать режим наблюдения, потому что после функции Thread.sleep() программа заново подключается к sitemap и вписывает новости, которых еще нет в таблице, далее проверяет, и при совпадении значения ссылки на страницу, пропускает её.