Лабораторная работа №7. Веб-сканер

В этой лабораторной работе вам необходимо будет реализовать элементарный веб-сканер. Сканер будет автоматически загружать веб-страницы из Интернета, искать новые ссылки на этих страницах и повторять их. Он будет просто искать новые URL-адреса (местоположения веб-страниц) на каждой странице, собирать их и выводит в конце работы программы. Более сложные веб-сканеры используются для индексации содержимого Интернета или для очистки адресов электронной почты от спама. Если вы когда-нибудь использовали поисковую систему, то вы в ответ на запрос получали данные, генерируемые поисковым роботом.

Терминология

- **URL:** унифицированный указатель ресурса. Это адрес веб-страницы. Он имеет следующую структуру:

1) метод доступа к ресурсу;

2) доменное имя

3) путь к файлу

4) данные о файле

В данной лабораторной работе будет рассмотрен метод доступа «http://».

- **HTTP:** Hyper Text Transfer Protocol (Протокол передачи гипертекста). Это стандартный текстовый протокол, используемый для передачи данных веб-страницы через Интернет. Последней спецификацией HTTP является версия 1.1, которую будет использована в данной лабораторной работе.

- **Сокет:** Сокет(разъем) - это ресурс, предоставляемый операционной системой, который позволяет вам обмениваться данными с другими компьютерами по сети. Вы можете использовать сокет для установки соединения с веб-сервером, но вы должны использовать TCP-сокет и использовать протокол HTTP для того, чтобы сервер мог ответить.

- **Порт:** несколько разных программ на одном сервере могут слушать соединения через разные порты. Каждый порт обозначается номером в диапазоне 1..65535. Номера от 1 до 1024 зарезервированы для операционной системы. У большинства серверов есть порт по умолчанию. Для HTTP-соединений обычно используется порт 80.

Программа для записи

Ниже указаны требования к программе, которую вы должны написать.

1. Программа должна принимать в командной строке два параметра:

1) Строку, которая представляет собой URL-адрес, с которого можно начать просмотр страницы.

2) Положительное целое число, которое является максимальной глубиной поиска (см. ниже)

Если указаны некорректные аргументы, программа должна немедленно остановиться и выдать сообщение об используемых аргументах, например:

usage: java Crawler <URL><depth>

2. Программа должна хранить URL-адрес в виде строки вместе с его глубиной (которая для начала будет равна 0). Вам будет необходимо создать класс для представления пар [URL, depth].

3. Программа должна подключиться к указанному сайту в URL-адресе на порт 80 с использованием сокета (см. ниже) и запросить указанную веб-страницу.

4. Программа должна проанализировать возвращаемый текст, построчно для любых подстрок, имеющих формат:

<a href="[любой_URL-адрес_начинающийся_с_http://]">

Найденные URL-адреса должны быть сохранены в паре с новым значением глубины в LinkedList (URL, depth) (подробнее о LinkedLists см. ниже). Новое значение глубины должно быть больше, чем значение глубины URL-адреса, соответствующего анализируемой странице.

5. Далее программа должна закрыть соединение сокета с хостом.

6. Программа должна повторять шаги с 3 по 6 для каждого нового URL-адреса, если глубина, соответствующая URL-адресу, меньше максимальной. Обратите внимание, что при извлечении и поиске определенного URL-адреса глубина поиска увеличивается на 1. Если глубина URL-адреса достигает максимальной глубины (или больше), не извлекайте и не ищите эту веб-страницу.

7. Наконец, программа должна вывести все посещенные URL-страницы вместе с их глубиной поиска.

Предположения

• Сложно разобрать, а тем более подключиться ко всем правильно и неправильно сформированным гиперссылкам в Интернете. Предположим, что каждая ссылка сформирована правильно, с полным именем хоста, путем и ресурсу. Вы можете создать небольшой сайт для тестирования, или вы можете выбрать любой другой сайт в интернете с уровнем доступа http://. (Вы можете попробовать http://slashdot.org/ или http://www.nytimes.com.)

В случае, если вы найдете URL-адрес с отличным от http:// методом доступа, вы должны его игнорировать.

• Предположим, если ваш BufferedReader возвращает значение null, то сервер завершил отправку веб-страницы. На самом деле это не всегда верно для медленных веб-серверов, но для текущих целей это приемлемо.

Полезные классы и методы

Указанные ниже классы и методы должны помочь вам начать работу. Обратите внимание, что большинство этих методов выбрасывают различные виды исключений, с которыми вам придется работать. Более подробную информацию вы можете найти на сайте Java API.

Socket

Для использования сокетов (Socket) вам необходимо включить эту строку в вашу программу:

import java.net. *;

Конструктор

Socket (String host, int port) создает новый сокет из полученной строки с именем хоста и из целого числа с номером порта, и устанавливает соединение.

Методы

- void setSoTimeout(int timeout) устанавливает время ожидания сокета (Socket) в миллисекундах. Данный метод необходимо вызвать после создания сокета, чтобы он знал, сколько нужно ждать передачи данных с другой стороны. В противном случае у него будет бесконечное время ожидания, что приведет к неэффективности разрабатываемого сканера.
- InputStream getInputStream() возвращает InputStream, связанный с Socket. Этот метод позволяет сокету получать данные с другой стороны соединения.
- OutputStream getOutputStream() возвращает *OutputStream*, связанный с Socket. Этот метод позволяет сокету отправлять данные на другую сторону соединения.
- void close() закрывает сокет (*Socket)*.

Потоки (Streams)

Для использования потоков в разрабатываемой программе необходимо включить в код следующую строку:

import java.io. *;

Для эффективного использования сокетов, вам необходимо преобразовать InputStream и OutputStream, связанные с Socket, во что-то более удобное в использовании. Объекты InputStream и OutputStream являются примитивными, так как они могут читать только байты или массивы байтов. Поскольку в данной работе необходимо читать и писать символы, вы должны использовать объекты, которые преобразуют байты в символы и печатают целые строки. Java API делает это несколькими разными способами для ввода и вывода.

Потоки ввода (Input Streams)

Для потоков ввода вы можете использовать классы InputStreamReader следующим образом:

InputStreamReader in = new InputStreamReader (my_socket.getInputStream());

Теперь in имеет тип InputStreamReader, который может читать символы из сокета (Socket). Но данный подход не очень удобен, потому что по-прежнему приходится работать с отдельными символами или массивами символов. Для чтения целых строк вы можете использовать класс BufferedReader. Вы можете создать BufferedReader с объектами типа InputStreamReader, а затем вызвать метод readLine предусмотренный в BufferedReader. Данный метод будетчитать целую строку с другого конца соединения.

Потоки вывода (Output Streams)

Потоки вывода организованы проще. Вы можете создать экземпляр PrintWriter непосредственно из объекта OutputStream, а затем вызвать его метод println для отправки строки текста на другой конец соединения. Для этого используйте следующий конструктор:

PrintWriter (OutputStream out, boolean autoFlush)

Параметр autoFlush установите в значение true. Это приведет к очищению буфера вывода после каждого вызова метода println.

Строковые методы

Приведенные ниже методы String будут полезны в работе. Смотрите документацию по API.

- boolean equals (Object anObject)
- String substring (int beginIndex)
- String substring (int beginIndex, int endIndex)
- boolean startsWith (String prefix)

ПРИМЕЧАНИЕ. Не используйте оператор == для сравнения строк! Оператор будет возвращать true, только если две строки являются одним и тем

же объектом. Если вы хотите сравнить содержимое двух строк, используйте метод equals.

Списки (Lists)

Списки похожи на массивы объектов, за исключением того, что они могут легко менять размерность при необходимости, и в них не используются скобки для поиска отдельных элементов. Чтобы использовать списки, вы должны включить следующую строку в код программы:

import java.util. *;

Для хранения пар (URL, depth) используйте LinkedList, который является реализацией List. Создайте его следующим образом:

LinkedList <URLDepthPair> myList = новый LinkedList <URLDepthPair> ();

Посмотрите в API используемые методы в списках и различные реализации списков. (В частности, вы можете заметить, что разные реализации List предоставляют разные функции. Именно поэтому рекомендуется LinkedList, некоторые из его функций больше подходят для этой работы.)

Специальный синтаксис для создания LinkedList продемонстрированный выше использует поддержку Java 1.5 generics. Этот синтаксис означает, что вам не нужно использовать приведение типов для объектов, которые вы храните или извлекаете из списка.

Исключения

В случае, если вы найдете URL-адрес, который не начинается с «http: //», вы должны выдать исключение MalformedURLException, которое является частью Java API.

Советы по проектированию

Ниже приведены рекомендации по разработке вашего поискового сканера.

Пары URL-Depth

Как упоминалось выше, вы должны создать специальный класс URLDepthPair, каждый экземпляр которого включает в себя поле типа String, представляющее URL-адрес, и поле типа int, представляющее глубину поиска. У вас также должен быть метод toString, который выводит содержимое пары. Этот метод упрощает вывод результатов веб-сканирования.

Отдельные URL-адреса необходимо разбить на части. Этот анализ URL-адреса и манипуляция им должны быть частью созданного вами класса URLDepthPair. Правила хорошего объектно-ориентированного программирования гласят, что если какой-либо класс хранит в себе определенный тип данных, тогда любые виды манипуляций с этими данными также должны быть реализованы в данном классе. Итак, если вы пишете какие-либо функции для того, чтобы разбить URL-адрес, или для проверки на то, является ли URL-адрес допустимым, поместите их в этот класс.

Сканеры (Crawlers)

Как уже упоминалось выше, вы должны спроектировать класс Crawler, который будет реализовывать основные функциональные возможности приложения. Этот класс должен иметь метод getSites, который будет возвращать список всех пар URL-глубины, которые были посещены. Вы можете вызвать его в основном методе после завершения сканирования; получить список, затем выполнить итерацию по нему и распечатать все URL-адреса.

Самый простой способ отслеживания посещенных сайтов состоит в том, чтобы хранить два списка, один для всех сайтов, рассмотренных до текущего момента, и один, который включает только необработанные сайты. Вам следует перебирать все сайты, которые не были обработаны, удаляя каждый сайт из списка перед загрузкой его содержимого, и каждый раз, когда вы находите новый URL-адрес, необходимо поместить его в необработанный список. Когда необработанный список пуст, сканер завершает работу.

Несмотря на то, что возникает предположение, что открытие сокета по URL-адресу - операция, связанная с URL-адресом, и, следовательно, должна быть реализована в классе пар URL-Depth, это было бы слишком специализированным для целей класса. Класс пар URL-Depth является только местом для хранения URL-адресов и значений глубины, а также включает в себя несколько дополнительных утилит. Сканер (Crawler) - это класс, который перемещается по веб-страницам и ищет URL-адреса, поэтому класс сканера должен включать в себя код, который фактически открывает и закрывает сокеты.

Вам нужно создать новый экземпляр Socket для каждого URL-адреса, с которого вы загружаете текст. Обязательно закройте сокет, когда вы закончите сканирование этой веб-страницы, чтобы операционная система не исчерпала сетевые ресурсы. (Компьютер может держать открытыми очень много сокетов одновременно). Кроме того, не используйте рекурсию для поиска глубоко вложенных веб-страниц; реализуйте эту функцию через цикл. Это также сделает ваш веб-сканер более эффективным с точки зрения использования ресурсов.

Константы

Разрабатываемая программа будет содержать строки типа «http: //» и «a href = \», и у вас может возникнуть желание использовать эти строки везде, где они вам могут понадобиться. Кроме того, вам понадобятся длины для разных строковых операций, поэтому у вас также может возникнуть желание жестко закодировать длины этих строк в коде. Не стоит это делать. Если вы сделаете опечатку или позже поменяете способ поиска, вам придется соответственно поменять достаточное количество строк кода.

Вместо этого создайте строковые константы в классах. Например:

public static final string URL_PREFIX = "http: //";

Теперь, если вам будет нужна эта строка, используйте константу URL_PREFIX. Если вам нужна ее длина, используйте строковый метод для приведенной выше константы: URL_PREFIX.length().

Продумайте расположение констант. Вам необходимо, чтобы каждая константа появлялась один раз во всем проекте, и вы должны разместить константу там, где это наиболее целесообразно. Например, поскольку префикс URL-адреса нужен для того, чтобы определить, действителен ли URL-адрес, вы должны поместить эту константу в класс URL-Depth. Если у вас есть еще одна константа для ссылок HTML, поместите ее в класс сканера. Если у сканера появится необходимость в префиксе URL, он может ссылаться на константу пары URL-depth, вместо дублирования этой константы.

Дополнительное задание

- Добавьте код для добавления сайтов в необработанный список, если они не были просмотрены ранее.
- Расширьте возможности по поиску гиперссылок сканера, используя поиск по регулярным выражениям в собранных данных. Вам также понадобится больше логики, чтобы решить, к какой машине подключаться в следующей итерации. Сканер должен иметь возможность перемещаться по ссылкам на различных популярных сайтах.
- Создайте пул из пяти (или более) сканеров. Каждый должен работать в своем потоке, каждый может получить URL-адрес для просмотра и каждый из них должен вернуть список ссылок по завершению работы. Посылайте новые URL-адреса в этот пул, как только они станут доступными.
- Расширьте многопоточный сканер так, чтобы можно было выполнить поиск на глубину до 1 000 000. Сохраняйте результаты каждого обхода в базе данных через JDBC и запишите, сколько раз на каждую конкретную уникальную страницу ссылались другие. Включите интеллектуальный алгоритм для «поиска смысла» на каждой странице путем взвешивания слов и фраз на основе повторяемости, близости к началу абзацев и

разделов, размера шрифта или стиля заголовка и, по крайней мере, мета-ключевых слов.