Задание #7. Поисковый робот

Цель работы

В этой лабораторной работе, требуется написать простейший **поисковый робот**. Робот должен автоматически загружать веб страницы из сети Интернет, искать в них новые ссылки, и повторять эту операцию для каждой найденной ссылки. В этой лабораторной работе поисковый робот будет настолько простым, насколько можно себе это представить. Он будет просто просматривать новые URL (указывающее на расположение других веб страниц) на каждой странице, сохранять эти ссылки и печатать их в конце работы программы. Более сложные поисковые роботы используются, например, для индексирования содержимого сети Интернет или сбора адресов электронной почты для рассылки спама. Если вы когда либо пользовались поисковым сервисом, то вы искали в данных созданных поисковым роботом.

Терминология

URL: Uniform Resource Locator. Единый указатель ресурсов. Адрес веб страницы. В нашем случае, он содержит строку "http://", за которой следует имя веб сервера (например,"`www.cs.caltech.edu`"), за которым следует путь к веб странице на сервере (например, "`/courses/cs11`"). Если этот путь не заканчивается расширением ".html" сервер решает сам какую страницу он должен возвратить. Это может быть `index.html`, `index.shtml`, `index.php`, `default.htm`, или что то еще, что сервер считает документом “по умолчанию” для указанного каталога.
Замечание: Есть и другие типы URL, начинающиеся (например) с "`mailto://`" или "`ftp://`". Эти типы URL нас здесь не интересуют.
HTTP: HyperText Transfer Protocol. Протокол передачи гипертекста. Это текстовый протокол, используемый для передачи веб страниц через интернет. Последняя спецификация протокола HTTP имеет версию 1.1, которую мы и будем использовать. Запрос HTTP для загрузки страницы

```
http://www.cs.caltech.edu/courses/cs11
```

может выглядеть примерно так:

```
GET /courses/cs11 HTTP/1.1

Host: www.cs.caltech.edu

Connection: close

[дополнительная пустая строка]
```

Заметим, что запрос HTTP ДОЛЖЕН завешаться пустой строкой, иначе запрос не будет обработан сервером. Так же обратите внимание на использование заглавных букв. Если вы отправите в запросе Get или host сервер также не будет это обрабатывать.

В некоторых URL не указывается документ или ресурс, который нужно извлечь, например: "`http://www.caltech.edu`". В таких случаях, **необходимо** указать вместо ресурса обратную косую черту "/". Другими словами, путь к ресурсу *всегда* должен начинаться с символа /.
Socket: Сокет, это ресурс операционной системы использующийся для установки соединения с другим компьютером в сети. Сокет можно использовать для установки соединения с веб сервером, для этого нужно использовать TCP сокет, и передавать через него пакеты протокола HTTP.

Порт: Различные программы установленные на одном сервере могут ожидать соединений , прослушивая различные порты. Каждый порт имеет номер в диапазоне 1..65535; Номера от 1

до 1024 зарезервированы операционной системой. Большинство сервисов имеют порты по умолчанию; Для соединений HTTP обычно используется порт 80.

Программа, которую нужно написать

Ниже приведена спецификация программы, которую вы должны написать.

Программа должна иметь два аргумента в командной строке:

1. Строка, содержащая URL страницы с которой начинается поиск
2. Положительное целое число – максимальная глубина поиска (см. ниже)

Если программа получает некорректные аргументы, она должна немедленно прекратить работу и напечатать сообщение, содержащее правила вызова, например:

Правила вызова: java Crawler <URL> <глубина поиска>

(Дополнительную информацию об обработке аргументов командной строки можно посмотреть в этом документе.)

Программа должна сохранять URL в виде строки`String` вместе с глубиной (которая, в начале, равна 0). Для этого вы должны создать класс, содержащий пару значений [URL, глубина].
Программа должна подключаться к 80 порту заданного в URL сайта, используя класс `Socket`(см. ниже) и запрашивать указанную веб страницу,
Программа должна разбирать полученный в ответ текст (если он получен), строка за строкой отыскивая фрагменты строк такого формата:

```
<a href="[какой либо URL начинающийся с http://]">
```

Найденные URL должны сохраняться, вместе с новым значением глубины ссылки в списке `LinkedList` объектов (URL, глубина) (см. ниже подробности про `LinkedLists`). Новое значение глубины должно быть на единицу больше глубины URL обрабатываемой страницы.
Затем программа должна закрывать соединение с хостом.

Программа должна повторять шаги с 3 по 6 для каждого нового URL, до тех пор, пока глубина URL меньше максимальной глубины просмотра. Заметим, что при извлечении и поиске внутри URL глубина увеличивается на 1. Если глубина URL достигает максимума, не загружайте и не ищите новые ссылки в веб странице.

Перед завершением программа должна выводить все найденные URL в пределах заданной глубины поиска.

Допущения

Довольно трудно разобрать и еще труднее подключиться ко всем без исключения ссылкам, имеющим правильный или ошибочный формат. Предположим, что каждая ссылка имеет правильный формат, включающий полное имя хоста, путь к ресурсу, и заключена в двойные кавычки. Как это не удивительно, есть несколько больших веб сайтов, в который имеется много ссылок такого вида; можете попробовать этот `http://slashdot.org/`, или этот `http://www.nytimes.com`. (Кстати, обратная косая черта после slashdot.org, важна ...)
Заметим, что большинство ссылок которые мы не обрабатываем это те которые начинаются не с "http://"; это могут быть, например, ссылки "mailto://" или "ftp://". Не обрабатывайте такие ссылки.

Допустим, что если `BufferedReader` возвращает `null`, сервер завершил отправку страницы. Это в действительности может быть не так, в случае, если сервер работает очень медленно, но для наших целей это вполне приемлемо.

Полезные классы и методы

Как всегда детали выясняйте в документации Java API. Классы и методы, о которых будет сказано ниже, помогут вам начать работу. Большинство этих методов вызывает различные виды исключений, которые вы должны обрабатывать. Подробности также посмотрите в документации Java API.

Socket

Для того чтобы использовать класс `Socket` к программе надо добавить строку:

```
import java.net.*;
```

Конструктор

`Socket(String host, int port)` создает новый объект `Socket` из строки `String` содержащей имя хоста и номер порта, и устанавливает соединение.

Методы

`void setSoTimeout( int timeout)` устанавливает таймаут сокета в миллисекундах. Метод надо вызвать после создания объекта `Socket` чтобы задать время ожидания передачи данных с другой стороны соединения. Иначе вы будете ждать бесконечно долго, и это вероятно плохая идея для поискового робота. `InputStream getInputStream()` возвращает `InputStream` связанный с объектом `Socket` используемый для приема данных. OutputStream getOutputStream() возвращает `OutputStream` связанный с сокетом используемый для передачи данных. `void close()` закрывает `Socket`.

Потоки

Для того чтобы использовать потоки в программу надо добавить строку:

```
import java.io.*;
```

Для эффективного использования сокетов, объекты `InputStream` и `OutputStream` надо конвертировать во что то более удобное для работы. Экземпляры классов `InputStream` и `OutputStream` это очень простые объекты; они могут только считывать байты или массивы байтов (даже не символы). Так как нам требуется читать и записывать символы, нужны объекты способные конвертировать байты в символы и наоборот. К сожалению Java API делает это разными способами для входных и выходных потоков.

Входные потоки

Для входных потоков можно использовать класс `InputStreamReader`:

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

Теперь `in` это объект `InputStreamReader` который может считывать символы из объекта `Socket`. Однако, это и сейчас не очень удобно потому что приходится работать с отдельными символами (`char`) или массивами символов. Было бы хорошо считывать целиком строки. Для этого используется класс `BufferedReader`.
Объект `BufferedReader` можно создать из экземпляра класса `InputStreamReader` и затем вызывать метод `readLine` объекта `BufferedReader`. Метод считывает целую строку из другой стороны соединения сокета.

Выходные потоки

Работа с выходными потоками организована проще. Создаем экземпляр класса `PrintWriter` прямо из объекта сокета `OutputStream` и затем вызываем его метод `println` для передачи строки текста на другую сторону соединения сокета. Надо использовать такой конструктор:

```
PrintWriter(OutputStream out, boolean autoFlush)
```

с `autoFlush` = `true`. Тогда данные не будут попадать в буфер передатчика, но будут передаваться сразу после каждого вызова `println`.

методы класса `String`

Вам могут пригодиться нижеследующие методы класса `String`. См. также документацию API.

```
boolean equals(Object anObject)

String substring(int beginIndex)

String substring(int beginIndex, int endIndex)

boolean startsWith(String prefix)
```

ЗАМЕЧАНИЕ: Не используйте оператор == для проверки равенства объектов `String`! Оператор возвращает `true` только, если обе переменные `Strings` ссылки на один и тот же объект. Для сравнения содержимого двух строк, используйте метод `equals`.

Списки `List`

Списки (`List`) очень похожи на массивы объектов (`Object`) но их можно расширять или уменьшать, если это необходимо, и они не используют нотацию с квадратными скобками для доступа к отдельным элементам. Для использования `List` в программу надо добавить строку:

```
import java.util.*;
```

Пары значений (URL, глубина) можно хранить в `LinkedList`. Это одна из реализаций класса `List`. Создается так:

```
LinkedList<URLDepthPair> myList = new LinkedList<URLDepthPair>();
```

Посмотрите сами документацию на большой набор методов `Lists` и различных вариантов реализации `Lists`. (Обратите внимание на то, разные реализации `List` имеют разные свойства. Рекомендуем использовать `LinkedList`; некоторые свойства этого класса очень удобно использовать для решения нашей задачи.)
Особенный синтаксис, использованный выше для инициализации `LinkedList`, использует принципы обобщенного программирования Java 1.5 . Использование такого синтаксиса позволяет убрать явное преобразование типов элементов при удалении и добавлении их к списку, и избавляет вас от связанной с этим головной боли.

Исключения

Если вы встречаете нечто выглядящее как URL, но не начинается с "`http://`", вы должны вызвать исключение типа `MalformedURLException`, которое имеется в Java API.

Рекомендации по проектированию

Вот некоторые рекомендации по реализации вашего поискового робота. (Не следование этим рекомендациям может привести к необходимости переделки работы ...*!*)

Пара URL-Глубина просмотра

Как уже было сказано, вы должны создать класс `URLDepthPair`, каждый экземпляр которого в поле типа `String` будет хранить URL и в поле типа `int` текущую глубину поиска. К классу

надо добавить метод `toString`, который печатает пару значений на экране. Это облегчит вам вывод результатов работы поискового робота.
URL возможно потребуется разбивать на фрагменты. Разбор этих URL следует реализовать в созданном вами классе для хранения пар URL-глубина. Хороший объектно-ориентированный подход к программированию подразумевает, что если класс хранит какие либо данные, реализация процедур обработки этих данных должна быть реализована в этом же классе. Следовательно, если вам нужны функции, разделяющие URL на фрагменты, или функции проверки правильности формата URL добавьте их к этому классу!

Класс Crawler

Как уже было сказано, вы должны спроектировать класс `Crawler`, который реализует основной функционал приложения. Этот класс должен иметь метод `getSites` который возвращает весь список пар URL-глубина которые были просмотрены роботом. Его можно вызвать из метода `main` после завершения поиска; получить список, затем перебрать его элементы и вывести на экран все имеющиеся в нем ссылки.
Самый простой способ контролировать просматриваемые ссылки, это завести два списка. Один должен содержать все известные на данный момент ссылки, а другой, включать еще не просмотренные ссылки. Надо перебирать все ссылки которые еще не просмотрены, удаляя ссылку перед загрузкой ее содержимого, и каждый раз когда найден новый URL, он должен быть добавлен в список необработанных ссылок. Когда список необработанных ссылок пуст, работа закончена – вы нашли все ссылки.

Хотя можно подумать, что: "Открытие сокета URL это операция связанная с URL, и следовательно ее тоже нужно сделать в классе хранящем пару значений URL-глубина просмотра," это действие все таки выходит за рамки функций этого класса. Это только место для хранения значений URL и глубины, с некоторыми дополнительными функциями их обработки. Crawler, это класс который перемещается по веб страницам и ищет в них ссылки, поэтому класс Crawler должен содержать код открывающий и закрывающий сокеты.

Вам нужно создать новый экземпляр класса `Socket` для каждого URL извлеченного из загруженного документа. Не забудьте закрыть сокет после того как вы закончите обработку страницы, иначе у операционной системы закончатся свободные номера портов! (Компьютер одновременно может держать открытыми ограниченное количество сокетов.) Также, не используйте рекурсию для поиска страниц на большей глубине; сделайте это с помощью цикла. Учитывая объем используемых ресурсов этот способ лучше.

Константы!

В вашей программе несомненно будут использоваться строки вида `"http://"` и `"a href=\""`, и возможно вы собираетесь просто повторять их везде где требуется. Кроме этого в коде понадобятся длины этих строк используемые в различных строковых операциях, и возможно вы также собираетесь прямо указывать их в кодеso. ***Не делайте так!!!*** Это очень сильно затрудняет дальнейшую работу с кодом! Если вы сделаете опечатку, или в дальнейшем измените процедуру поиска, придется исправлять много разных строк кода. Вместо этого, создайте в вашем классе строковые константы. Например, такую:

```
    public static final String URL_PREFIX = "http://";
```

Теперь если нужна строка, вместо того чтобы прямо вставлять ее, используйте константу `URL_PREFIX`. Если вам нужна длина этой строки, вам повезло - `URL_PREFIX` это объект типа `String`, а значит, вы можете вызвать метод `URL_PREFIX.length()` и получить длину строки.
Подумайте о том, куда поместить эти константы. Каждая константа должна быть объявлена в коде проекта только один раз, и она должна располагаться в наиболее подходящем месте. Например, так как префикс URL нужен для того чтобы определить имеет ли URL правильный формат, поместите его в класс для хранения пары значений URL-глубина. Если у вас есть еще одна константа для HTML ссылок, поместите ее в ваш класс Crawler. Если классу Crawler

понадобится, зачем либо, префикс URL, он может использовать константу из класса пары значений URL-глубина, вместо того чтобы дублировать эту константу у себя.

Дополнительные баллы

Напишите код, который добавляет к списку ссылок для обработки только те ссылки, которые не обрабатывались раньше. Улучшите возможности робота по распознаванию ссылок в тексте, используя регулярные выражения для поиска в собранных данных. Следует также добавить дополнительную логику для определения к какой машине следует подключиться следующий раз. Поисковый робот должен переходить по ссылкамразличных популярных сайтов. Использование регулярных выражений требует дополнительного изучения, но в действительности реализуется гораздо проще, чем поиск подстрок в строках (и это гораздо **более** мощное средство).