Язык программирования Java

Лекция 6

Перевод курса CS11 Java Track

Содержание

Задание

- На этой неделе пишем простой поисковый робот
  - Который подключается к веб серверу
  - Отправляет серверу HTTP запрос
  - Получает HTTP ответ от сервера
  - Разбирает этот ответ и отыскивает в нем ссылки на другие страницы
  - Повторяет весь процесс для каждой найденной ссылки!

Сетевые протоколы

- В сети Интернет главным образом используются два протокола:
- TCP/IP (Или просто TCP)
  - Протокол управления передачей
  - Потоко-ориентированный протокол обеспечивающий надежную и упорядоченную передачу данных
- UDP
  - Протокол пользовательских датаграмм
  - Протокол для передачи сообщений (датаграмм), не обеспечивающий надежную и упорядоченную передачу данных.
- Java поддерживает оба протокола с помощью пакета java.net:
  - TCP – java.net.Socket
  - UDP – java.net.DatagramSocket
  - Поддерживаются также другие протоколы, например SSL в пакете java.net.sll

Обращение к Web серверу

- Протокол передачи гипертекста HTTP (Hypertext Transfer Protocol)
  - Текстовый протокол
  - Основан на принципе обмена запросами и ответами
  - Для передачи данных использует протокол TCP/IP
- Для установки сетевого соединения указываются
  - IP адрес (или имя хоста которое преобразуется в IP адрес)
  - Номер порта в диапазоне от 1 до 65535 (номера портов от 1 до 1024 зарезервированы)
- Различные сетевые службы имеют различные номера портов для подключения к ним:
  - К серверу электронной почты обычно подключаются через порт 25
  - К серверу SSH обычно подключаются через порт 22
  - К веб серверу обычно подключаются через порт 80

Адреса web страниц

- Единый указатель ресурсов URL (Uniform Resource Locator) это текстовая строка, которая используется в протоколе HTTP для указания местонахождения ресурса. URL содержит:
  - Определение протокола
  - Имя хоста или его IP адрес
  - Номер порта (не всегда нужно указывать, так как каждый протокол имеет порт, использующийся по умолчанию)
  - Путь к документу или ресурсу (также указывается не всегда)
- Пример: http://www.cs.caltech.edu/people.html
  - Протокол: HTTP
  - Имя хоста: www.cs.caltech.edu
  - Порт по умолчанию для HTTP: 80
  - Документ на сервере: /people.html

Запрос web страницы

- Для того чтобы выполнить HTTP запрос надо:
  - Подключиться к заданному порту заданного хоста используя java.net.Socket, так как соединение происходит по протоколу TCP
- Отправить HTTP запрос нужной страницы
- Получить HTTP ответ содержащий требуемую страницу
  - …или сообщение об ошибке
- Закрыть сокет, использующийся для связи
  - Сетевые ресурсы надо освобождать после использования!
- Обработать полученный документ
  - В нашем случае будем искать в нем другие URL

Подключение к серверу

- Для того чтобы подключиться к серверу для каждого соединения создается сокет
  - Требуется указать имя хоста или его IP адрес в параметре типа String
  - И номер порта:

```
webServer = "www.cs.caltech.edu";
webPort = 80;
Socket sock = new Socket(webServer, webPort);
```

- Проблема:
  - Что если сервера с таким именем не существует?
  - Или он не “прослушивает” сообщения на указанном порту?
- Конструктор класса Socket сообщает об ошибках с помощью исключений

Взаимодействие с web сервером

- Если сокет не может подключиться к удаленному серверу, он вызывает исключение
- Соединение так же может прерваться во время работы
- Ваш интернет робот должен перехватывать эти исключения
  - Обработка может быть очень простой: напечатайте сообщение об ошибке и затем перейдите к следующему URL
- Посмотрите в документации Java, какие типы исключений следует обработать в вашей программе

Передача данных через сокет

- После того как сокет открыт, из него можно извлечь ссылки на входной (InputStream) и выходной (OutputStream) потоки
  - OutputStream нужен для передачи данных удаленному хосту.
  - InputStream нужен для приема данных от удаленного хоста.
- Проблема
  - Классы InputStream и OutputStream не предназначены для передачи текстовых данных
  - Они созданы для работы с потоками байтов
  - Чтения/записи байта или массива байтов
  - Классы не могут работать с кодовыми таблицами символов
  - Преобразование массивов байтов в объекты String большая проблема

Классы чтения и записи

- Классы Reader и Writer сделаны для потоков символов
- Объект InputStream можно “обернуть” в Reader
  - Тогда объект Reader будет преобразовывать байты полученные из InputStream в символы или строки
- Объект OutputStream можно “обернуть” в Writer
  - Тогда объект Writer получая символы будет передавать объекту InputStream байты
    - Это именно то, что нужно для текстового протокола HTTP!
- Имеются несколько дочерних классов Reader и Writer
  - Которые так же работают с InputStream и OutputStream

Отправка HTTP запроса

- HTTP запрос должен имеет вид:

```
HTTP запрос должен имеет вид:
GET /people.html HTTP/1.1
Host: www.cs.caltech.edu
Connection: close
```

- Пустая строка обязательна! ☺
- В первой строке указано имя документа или ресурса, который требуется получить в ответ
  - Если документ находится в корне веб сервера, указывать “/” надо обязательно
- Во второй строке указано имя хоста
  - (Один физический сервер может обслуживать несколько виртуальных хостов)
- В третьей строке содержится указание серверу закрыть соединение после завершения передачи ответа

Пример кода запроса

```
Socket sock = new Socket(webHost, webPort);
sock.setSoTimeout(3000); // Таймаут после 3 секунд

OutputStream os = sock.getOutputStream();

// true заставляет PrintWriter каждый раз полностью
// передавать данные в поток
PrintWriter writer = new PrintWriter(os, true);

writer.println("GET " + docPath + " HTTP/1.1");
writer.println("Host: " + webHost);
writer.println("Connection: close");
writer.println();

// Запрос отправлен! Сервер теперь должен ответить.
```

Получение HTTP запроса

- Для чтения строк ответа из сокета используйте класс BufferedReader
  - Этому классу на вход данные должен передавать другой Reader
  - Для преобразования выходного потока сокета в Reader используйте класс InputStreamReader:

```
InputStream is = sock.getInputStream();
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);
```

- Метод br.readLine() надо вызывать до тех пор, пока он не вернет null
  - Для того чтобы это работало именно так, мы добавили к запросу строку “Connection: close”

Пример кода обработки ответа

```
InputStream is = sock.getInputStream();
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);

while (true) {
    String line = br.readLine();
    if (line == null)
        break; // Конец чтения документа!

    // Обрабатываем полученную строку.
    System.out.println(line);
}
```

Обработка исключений в поисковом роботе

- Продумайте глее и как добавить нужные обработчики исключений
- Каждая из операций поискового робота по обработке веб страницы:
  1. Подключение к удаленному серверу через сокет
  2. Отправка HTTP запроса
  3. Чтение HTTP ответа
  4. Извлечение URL из текста ответа
- Все эти операции могут приводить к вызову исключений
  - Извлечение URL может и не вызывать исключения. Это зависит от вашей реализации

Обработка исключений: простой подход

- Простой подход
  - Каждый шаг можно заключить в собственный блок try/catch
- Но имеет ли он смысл?
  - Если на каком то из этих шагов происходит ошибка, остальные теряют смысл!
- Исключение в шагах 1 – 3 приводит к завершению всей операции поиска в веб странице
  - На четвертом шаге, если разбор URL приводит к исключению, надо просто перейти к следующему URL на странице

Улучшенный вариант обработки исключений

- Исключения должны обрабатываться по принципу один обработчик на всю операцию
- Пример:
  - Такой операцией может быть обработа одной веб страницы в поисковом роботе.
- Улучшить структуру обработки ошибок в нашем случае можно так:
  - Поместить код обработки одного URL в отдельную функцию.
  - В этой функции операции могут вызывать исключения
    - Исключения приводят к прерыванию работы и выходу из функции
    - Каждое исключение приводит к завершению всей работы
  - Код, вызывающий функцию сам обрабатывает возникающие в ней исключения с помощью блока try/catch

Поиск строк

- Класс String имеет много полезных функций
- Для поиска символа или подстроки в строке можно использовать одну из этих:
  - int indexOf(int ch)
  - int indexOf(int ch, int fromIndex)
  - int indexOf(String str)
  - int indexOf(String str, int fromIndex)
  - или, lastIndexOf(...) для поиска с конца
- Все эти функции возвращают -1 если значение не найдено или позицию в строке
  - Допустимый диапазон индекса от 0 до length() -1

Обработка строк

- Получить подстроку из строки можно так:
  - String substring(int beginIndex)
  - String substring(int beginIndex, int endIndex)
- Изменить регистр символов строки:
  - String toLowerCase()
  - String toUpperCase()
- Удалить пробелы в конце и начале строки:
  - String trim()
- Заметим, что строки в Java неизменяемы
  - Это означает, что все указанные операции возвращают новые объекты String

Пример поиска слов

```
//TODO: получить откуда то строку и слово ...
String word = "after";
String line = …;

//Поиск слова в текущей строке.
int idx = 0;
while (true) {
    idx = line.indexOf(word, idx);
    if (idx == -1) // в строке больше нет таких слов
        break;

    //записать, что найдена еще одна копия слова.
    count++;

    //пропускаем эту копию слова, так чтобы
    //не найти ее снова в следующей итерации цикла!
    idx += word.length();
}
```

Поиск ссылок

- Найти ссылку в тексте немного труднее:

  Caltech

  1)Ищем строку a href="

  2)Если нашли, ищем закрывающие кавычки "

  3)Текст между кавычками и есть URL

- Надо учесть, что в одной строке может быть несколько ссылок
  - После извлечения URL переместите индекс в позицию за URL и ищите следующий
  - Но для простоты не нужно обрабатывать ссылки, имеющие перенос на следующую строку.

Обработка деталей

- Создайте простой класс URLDepthPair чтобы следить за глубиной просмотра найденных ссылок
- Первая ссылка находится на уровне 0
- Всем найденным на странице ссылкам присваивается уровень страницы + 1
  - Заносите в список новые объекты URLDepthPair созданные для всех ссылок на странице
  - После обработки страницы возьмите из этого списка следующая ссылка для обработки
- Второй аргумент командной строки программы должен указывать глубину просмотра поискового робота
- Эта стратегия не предусматривает обработку циклов …

Списки пар URL-глубина

- Для решения задачи со списками ссылок подходит класс LinkedList:

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

- Найдя новую ссылку, добавьте его к списку:

```
pendingURLs.add(new URLDepthPair(linkText, childDepth));
```

- Когда вам понадобится следующая ссылка для обработки:

```
while (!pendingURLs.isEmpty()) {
    nextURLPair = pendingURLs.removeFirst();
    ... // обработка этой пары ссылка-уровень
}
```

- Когда URL обработан
  - Создайте новый LinkedList для хранения ссылок
  - В конце программы распечатайте все ссылки

План повторного использования кода

- Сделайте так, чтобы код обработки ссылок можно было повторно использовать
  - Поместите его в отдельный метод или несколько методов
  - Это пригодится в 7 и 8 лабораторных работах
- На следующей неделе к программе добавятся новые свойства:
  - Многозадачное исполнение поиска
  - Ссылки будут обрабатываться параллельно
  - Доступ к общим ресурсам будет минимизирован