Informatics Large Practical: The Java HTTP Client

Stephen Gilmore and Paul Jackson

School of Informatics

Monday, 5th October 2020
Revised Monday, 2nd November 2020

Networking in Java

- Until recently, accessing web servers in Java was done with the java.net.HttpUrlConnection class but this class has become dated and has been superseded by a native HTTP Client for Java (added in Java 11).
- The Java HTTP Client makes accessing web servers and web server content much easier so it is the recommended way to access the web server which we use.

Java 11 HTTP Client

Java 11 has new classes allowing us to make an HTTP request, send it to an HTTP client, and receive an HTTP response.

- java.net.http.HttpClient
- java.net.http.HttpRequest
- java.net.http.HttpResponse
- java.net.http.HttpResponse.BodyHandlers

These classes are built into the Java 11 Standard Edition; we need to **import** any of these which we use into our Java classes but we do not have to add a new dependency to our Maven pom.xml file.

Synchronous and asynchronous requests

- HTTP requests can be sent either synchronously or asynchronously.
  - The send method blocks the calling thread until the HTTP response is available. It is suitable for transferring small files.
  - The sendAsync method immediately returns with a future object that eventually completes with an HTTP response. It is suitable for transferring large files.
- In this practical we work with very small files (max. 8Kb) so we have no need to use sendAsync.

Creating an HttpClient

JAVA

```java
// Just have one HttpClient, shared between all HttpRequests
private static final HttpClient client = HttpClient.newHttpClient();
```

- The HTTP Client is a heavyweight object which uses a native system thread, so we want to declare just one client and use it multiple times.
- We only need one HTTP Client so we declare it as **static**.
- The HTTP Client is not updated so we declare it as **final**.

Creating an HttpRequest and getting an HttpResponse

JAVA

```
// HttpClient assumes that it is a GET request by default.
var request = HttpRequest.newBuilder()
            .uri(URI.create(urlString))
            .build();
// The response object is of class HttpResponse<String>
var response = client.send(request, BodyHandlers.ofString());
```

- We build an HTTP GET request and then send it to the HTTP client.
- After this we can check response.statusCode() for the HTTP status code and get the content as a string using response.body().

Possible errors: Correct and incorrect URL strings

htpp://localhost:80/buildings/no-fly-zones.geojson

- Syntactically incorrect, URI.create(urlString) will throw IllegalArgumentException. This is a bug in our Java code.

---

http://localhost:80/buildings/no-fly-zones.jeogson

- Not an illegal argument, but semantically incorrect.
- We will get a response.statusCode() of 404 [Not Found].

---

http://localhost:80/buildings/no-fly-zones.geojson

- Syntactically and semantically correct.
- We will get a response.statusCode() of 200 [OK!].
- In this case we can use response.body().

Running the web server on localhost (outside of Eclipse)

- We run a lightweight web server* which is written in Java.
- HTTP requests are confirmed to the console so you can see that your client.send(...) requests are happening.

```
$ java -jar WebServerLite.jar
[Fri Oct 02 12:34:48 BST 2020] 0:0:0:0:0:0:0:1 "GET / HTTP/1.1" 200
[Fri Oct 02 12:35:02 BST 2020] 0:0:0:0:0:0:0:1 "GET /buildings/
    HTTP/1.1" 200
[Fri Oct 02 12:35:07 BST 2020] 0:0:0:0:0:0:0:1 "GET
    /buildings/no-fly-zones.geojson HTTP/1.1" 200
...
```

*Download this from the ILP Learn page at http://learn.ed.ac.uk.

Possible errors: The web server is not running

- If the web server is not running, or not running on the port that we think it is, then an attempt to invoke client.send() will end with a java.net.ConnectException.
- A java.net.ConnectException is a fatal error which we cannot recover from and should just exit the application gracefully.

JAVA

```
System.out.println("Fatal error: Unable to connect to " +
    server + " at port " + port + ".");
System.exit(1); // Exit the application
```

Summary

- The Java 11 HTTP Client is an easy way to interact with the web server which we use.
- Remember that your web server must be running if your Java code is to interact with it.

Thank you for listening.