4. Acceso a servicios desde Java

Parte I. Acceso básico a servicios web

Programación de Servicios y Procesos

Nacho Iborra
Álvaro Pérez
Javier Carrasco

Tabla de contenido

1. Introducción

Hoy en día, la mayoría de las aplicaciones dependen de los servicios web para acceder a datos remotos y centralizados almacenados en un servidor remoto a través de la World Wide Web. Un servicio web es una tecnología que utiliza una serie de estándares para comunicar una aplicación que se ejecuta en el servidor (utilizando cualquier tecnología como PHP, Node, Ruby, Java, .NET, etc) con una aplicación cliente que se puede ejecutar en cualquier dispositivo y estar escrito también en cualquier lenguaje. En esta parte se centrará la atención principalmente en acceder a los servicios web REST utilizando Java como cliente, confiando en un lado del servidor implementado en Node.js de la unidad anterior.

1.1. Abrir y leer una conexión HTTP

Hay muchas formas de conectarse a una web a través del protocolo HTTP en Java. Por ejemplo, se pueden usar las clases nativas derivadas de URLConnection, o una biblioteca externa que simplifique el trabajo.

Debido a que uno de los usos principales de Java es para el desarrollo de Android, se verá lo que se recomienda. Existe una biblioteca llamada *Apache Http Client*, que se incluyó en las bibliotecas de Android, pero ya no es compatible (e incluso si aún puedes usarlo, no se recomienda). La opción recomendada es utilizar la clase *URLConnection* y sus derivados, como HttpURLConnection y HttpsURLConnection.

1.1.1. Uso de URLConnection

Este es el método de conexión de más bajo nivel, lo que significa que el programador controlará todos los aspectos de la conexión, pero en contraste, el código resultante será más grande y feo. Se recomienda en Android porque, si se usa correctamente, es el método más rápido y que menos memoria, procesador y batería consume.

La clase URL se utiliza para representar el recurso remoto en la World Wide Web al que se accederá.

```
URL google = new URL("http://www.google.es");
```

Este objeto devolverá un objeto *URLConnection* cuando se realice la conexión.

```
URLConnection conn = google.openConnection();
```

Para obtener el cuerpo de la respuesta (contenido), la conexión proporciona un *InputStream* para ese propósito. También se recomienda recuperar la codificación del juego de caracteres de los encabezados de respuesta para leer todo (como los acentos) correctamente.

```
// Get charset encoding (UTF-8, ISO,...)
public static String getCharset(String contentType) {
    for (String param : contentType.replace(" ", "").split(";")) {
```

```
        if (param.startsWith("charset=")) {
            return param.split("=", 2)[1];
        }
    }
    return null; // Probably binary content
}
```

El método *main()* quedaría de la siguiente forma.

```
public static void main(String[] args) {
    BufferedReader bufferedReader = null;
    try {
        URL google = new URL("http://www.google.es");
        URLConnection conn = google.openConnection();

        String charset = getCharset(conn.getHeaderField("Content-Type"));

        bufferedReader = new BufferedReader(new InputStreamReader(
                conn.getInputStream(),
                charset)
        );

        String line;
        while ((line = bufferedReader.readLine()) != null) {
            System.out.println(line);
        }
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (bufferedReader != null) {
            try {
                bufferedReader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

1.1.2. HttpURLConnection y redirecciones

La clase *HttpURLConnection* proporciona métodos adicionales como seguir redirecciones automáticamente u obtener el código de respuesta (como 404 para “No encontrado”).

Para obtener una *HttpURLConnection* y seguir las redirecciones automáticamente, debes llamar a este método:

```
URL url = new URL("http://wikipedia.com");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setInstanceFollowRedirects(true);
```

No siempre funciona. Por ejemplo, puede devolver el código 301 (Movido permanentemente) y, por tanto, no serás redirigido a la nueva ubicación automáticamente.

Puedes verificar cuál es la respuesta (y sus encabezados) utilizando los métodos disponibles:

```
System.out.println(conn.getResponseCode());
System.out.println(conn.getResponseMessage());
System.out.println(conn.getHeaderFields());
```

Con esta información se podría gestionar manualmente estos redireccionamientos (con el riesgo de caer en un bucle de redireccionamiento), aunque sean muchos, como este:

```
URL url = new URL("http://wikipedia.com");
HttpURLConnection conn;

do {
    conn = (HttpURLConnection) url.openConnection();
    if (conn.getResponseCode() == 301) {
        url = new URL(conn.getHeaderField("Location"));
    }
} while (conn.getResponseCode() == 301);
```

Ejercicio 1

Cree una aplicación Java de consola llamada ***LinkSearch*** que pedirá una dirección e imprimirá todos los enlaces (*<a>*) detectados en la respuesta. Si lo deseas, es una buena idea crear una clase auxiliar que extienda de *BufferedReader*, como se vio en la parte II de la unidad 1, para filtrar solo esos enlaces de la salida.

Este es el resultado obtenido para https://xataka.com:

```
Enter address: https://xataka.com
<a id="favicons-toggle" href="https://www.webedia.es/" data-target="#head-favicons"><abbr title="Webedia">Webedia</abbr></a>
<a href="/" class="masthead-brand">Xataka</a>
<a href="#sections" class="masthead-actions-menu m-v1 js-toggle" data-searchbox="#search-field-1">Menú</a>
<a href="#headlines" class="masthead-actions-nuevo m-v1 js-toggle">Nuevo</a>
<a class="masthead-nav-search js-toggle" data-searchbox="#search-field-2" href="#search"></a>
<a class="masthead-nav-topics-anchor" href="https://www.xataka.com/categoria/analisis">ANÁLISIS</a>
<a class="masthead-nav-topics-anchor" href="https://www.xataka.com/categoria/seleccion">XATAKA SELECCIÓN</a>
<a class="masthead-nav-topics-anchor" href="https://www.xataka.com/categoria/moviles">MÓVILES</a>
<a class="masthead-nav-topics-anchor" href="https://www.xataka.com/categoria/medicina-y-salud">CIENCIA</a>
<a class="masthead-nav-topics-anchor" href="https://www.xataka.com/categoria/pro">PRO</a>

...
```

2. Conceptos básicos del acceso a servicios web

Para acceder a un servicio web REST (*Representational State Transfer*) se necesita una URL (que representa el recurso al que se accede), y una operación (GET, POST, PUT, DELETE) para hacer con ese recurso, junto con los datos adicionales necesarios para la operación.

La operación más simple es GET, que generalmente se usa para buscar y obtener información sobre algo que ya existe. En una operación GET, los datos (si son necesarios) se envían en una URL como esta → *http://domain/resource?data1=value1&data2=value2* , o esta → *http://domain/resource/value1/value2* .

Este es un servicio *Express* realmente básico que leerá dos números pasados en la url (GET) e imprimirá (respuesta) el resultado de esa suma:

```
app.get('/services/sum/:n1/:n2', (req, res) => {
    let result = parseInt(req.params.n1) + parseInt(req.params.n2);
    res.send("" + result);
});
```

Y así es como podría llamarse desde **Java** y obtener el resultado:

```
private static String getSumFromService(int n1, int n2) {
    BufferedReader bufferedReader = null;
    String result;

    try {
        URL google = new URL("http://localhost/services/sum/" + n1 + "/" + n2);
        URLConnection conn = google.openConnection();
        bufferedReader = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
        result = bufferedReader.readLine();
    } catch (IOException e) {
        return "Error";
    } finally {
        if (bufferedReader != null) {
            try {
                bufferedReader.close();
            } catch (IOException e) {
                return "Error";
            }
        }
    }
    return result == null ? "Error" : result;
}

public static void main(String[] args) {
    System.out.println(getSumFromService(3, 5));
}
```

2.1. Clase ServiceUtils

Para envolver todo el código necesario para conectarse a un servicio web y enviar/recibir información hacia/desde él, se creará una clase propia. Se llamará ***ServiceUtils***, y su código será el siguiente:

```
public class ServiceUtils {
    private static String token = null;

    public static void setToken(String token) {
        ServiceUtils.token = token;
    }

    public static void removeToken() {
        ServiceUtils.token = null;
    }

    public static String getCharset(String contentType) {
        for (String param : contentType.replace(" ", "").split(";")) {
            if (param.startsWith("charset=")) {
                return param.split("=", 2)[1];
            }
        }
        return null; // Probably binary content
    }
```

```
    public static String getResponse(String url, String data, String method) {
        BufferedReader bufInput = null;
        StringJoiner result = new StringJoiner("\n");

        try {
            URL urlConn = new URL(url);
            HttpURLConnection conn = (HttpURLConnection) urlConn.openConnection();

            conn.setReadTimeout(20000 /*milliseconds*/);
            conn.setConnectTimeout(15000 /* milliseconds */);
            conn.setRequestMethod(method);
            conn.setRequestProperty("Host", "localhost");
            conn.setRequestProperty("Connection", "keep-alive");
            conn.setRequestProperty("Accept", "application/json");
            conn.setRequestProperty("Origin", "http://localhost");
            conn.setRequestProperty("Accept-Encoding", "gzip,deflate,sdch");
            conn.setRequestProperty("Accept-Language", "es-ES,es;q=0.8");
            conn.setRequestProperty("Accept-Charset", "UTF-8");
            conn.setRequestProperty("User-Agent", "Java");

            // If set, send the authentication token
            if (token != null) {
                conn.setRequestProperty("Authorization", "Bearer " + token);
            }

            if (data != null) {
                conn.setRequestProperty("Content-Type", "application/json; charset=UTF-8");
                conn.setRequestProperty("Content-Length", Integer.toString(data.length()));
                conn.setDoOutput(true);

                //Send request
                DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
                wr.write(data.getBytes());
                wr.flush();
                wr.close();
            }

            String charset = getCharset(conn.getHeaderField("Content-Type"));

            if (charset != null) {
                InputStream input = conn.getInputStream();
                if ("gzip".equals(conn.getContentEncoding())) {
                    input = new GZIPInputStream(input);
                }

                bufInput = new BufferedReader(new InputStreamReader(input));
                String line;
                while ((line = bufInput.readLine()) != null) {
                    result.add(line);
                }
            }
        } catch (IOException e) {
        } finally {
            if (bufInput != null) {
                try {
                    bufInput.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return result.toString();
    }
}
```

Como puedes ver, se utilizará el método *getCharset()* explicado antes para obtener la codificación del juego de caracteres para la comunicación. El método *getResponse()* se utilizará para enviar una solicitud a un servicio web. Tiene 3 parámetros: la *url* a conectar, los datos a enviar en el cuerpo de la solicitud (o nulo si no hay datos) y la operación o método (GET, POST, PUT, DELETE). Esta respuesta se almacenará en una cadena que se devolverá desde este método estático.

Además, se han establecido algunos encabezados HTTP para que la solicitud sea similar a lo que enviaría un navegador web, como el dominio de origen (en este caso *localhost*), el idioma preferido (español), la posibilidad de comprimir datos (gzip) para ahorrar ancho de banda, un tiempo de espera para la conexión o el tipo de datos utilizados para la comunicación (application/json).

También hay algunos otros métodos y atributos para tratar con los tokens para la autenticación del cliente, de modo que se pueda almacenar el token proporcionado por el servidor en una variable estática y enviarlo de vuelta al servidor en cada solicitud. Pero no se utilizará por ahora.

Se proporcionará esta clase para que puedas usarla en los ejercicios para ayudar en la conexión y obtención de datos de los servicios web más rápidamente.

3. Procesar JSON

Hoy en día, la mayoría de los servicios web envían y reciben información en formato JSON (XML está casi abandonado para este uso). Este formato es nativo de Javascript pero la mayoría de lenguajes como Java tienen las herramientas necesarias para procesarlo.

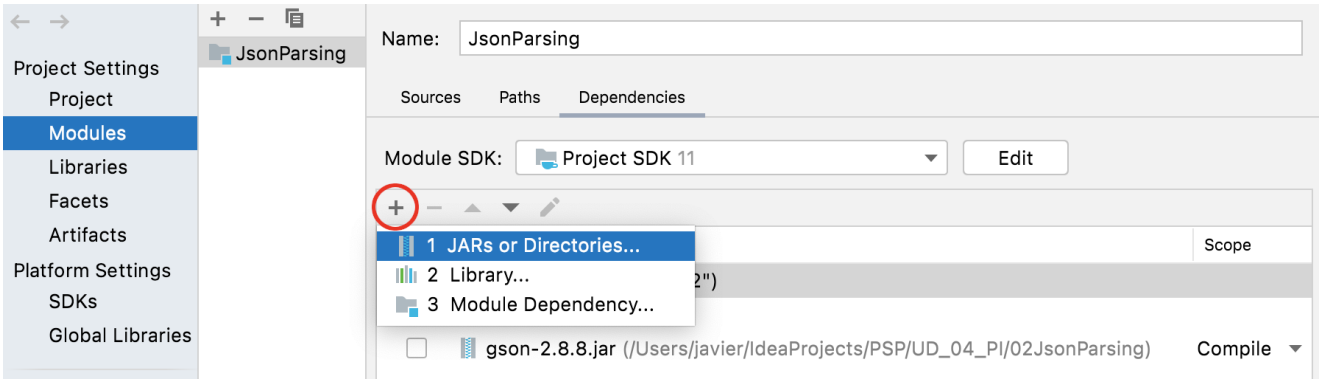
La información básica sobre JSON y las herramientas disponibles para cada idioma se pueden encontrar en http://www.json.org/. Para procesar esta información se puede utilizar la API org.json (también presente en Android) u otras opciones como GSON de Google, pero hay muchas opciones. Aquí se verá cómo usar la biblioteca GSON, y podrás ver cómo usar la API org.json en el Anexo I de esta unidad.

3.1. Uso de GSON

A continuación, se verá un ejemplo de cómo usar la biblioteca GSON. En primer lugar, deberá añadirse esa biblioteca al proyecto Java. Puedes descargar la última versión del archivo JAR en el repositorio de Maven, o la versión que encontrarás en el Aula Virtual.

Pasos para añadir librerías externas (*jar*) en IntelliJ IDEA:

1. Haz clic en ***File*** en la barra de herramientas.
2. ***Project Structure*** (CTRL+MAYÚS+ALT+S en Windows/Linux).
3. Selecciona ***Modules*** en el panel de la izquierda.
4. Pestaña ***Dependencies***.
5. ‘+’ → JARs o directorios



Imagina que se recibe esta información de un servicio web en formato JSON:

```
{
    "error":false,
    "person": {
        "name":"Peter",
        "age":30,
        "address":[
            {"city":"London","street":"Some street 24"},
            {"city":"New York","street":"Other street 12"}
        ]
    }
}
```

GSON intentará convertir automáticamente de JSON a un objeto nativo de Java. Por lo tanto, necesitarás una clase que contenga los mismos campos que la respuesta JSON (mismo nombre). No es necesario crear ningún constructor específico. Para el ejemplo anterior, se necesitará una clase llamada *Address* con dos atributos llamados *city* (*String*) y *street* (*String*), y otra clase llamada *Person* con los atributos *name* (*String*), *age* (*int*) y *address* (de tipo *Address*).

Ahora se necesitará una clase adicional que mapee el formato de respuesta JSON inicial:

```
public class GetPersonResponse {
   boolean error;
   Person person;

   public boolean getError() {
      return error;
   }

   public Person getPerson() {
      return person;
   }
}
```

Si los nombres de los campos están configurados correctamente, mapeará todo automáticamente:

```
public static void main(String[] args) {
    String json = ServiceUtils.getResponse("http://localhost/services/example", null, "GET");
    if(json != null) {
        Gson gson = new Gson();
        GetPersonResponse personResp = gson.fromJson(json, GetPersonResponse.class);

        if(!personResp.getError()) {
```

```
            System.out.println(personResp.getPerson().toString());
            System.out.println(personResp.getPerson().getClass());
        } else {
            System.out.println("There was an error in the request");
        }
    }
}
```

Para este ejemplo, es necesario sobrecargar el método *toString()* en ambas clases, *Person* y *Address*, para mostrar sus datos en un formato apropiado. Observa cómo se utiliza la clase *ServiceUtils* explicada anteriormente para obtener una respuesta, y luego se utiliza la biblioteca GSON para analizar la respuesta y almacenar los datos correspondientes en los objetos apropiados, de acuerdo con la clase *GetPersonResponse*.

Si el nombre de una propiedad de clase no coincide con el nombre del campo JSON, se puede decir al analizador GSON que tiene que asignar ese campo usando una anotación con esa propiedad:

```
@SerializedName("error")
boolean haserror; // In JSON it will be named "error"
```

Puedes obtener más información sobre la biblioteca GSON en este tutorial de GSON.

Ejercicio 2

Crea un proyecto Java llamado ***JsonParsing***. Añade la biblioteca GSON y luego implementa un programa (una aplicación de consola, no una JavaFX) que use el código anterior (clases *ServiceUtils*, *Person*, *Address* y *GetPersonResponse*, además de la aplicación principal) para conectarse a un servidor y recuperar la información de la persona.

Puedes usar el servidor de Node que se proporcionó en los recursos de esta sesión y acceder a la URL de *localhost/services/example* para recuperar los datos JSON. También puedes editar el código de este servidor para cambiar el URI o el puerto, si lo deseas.

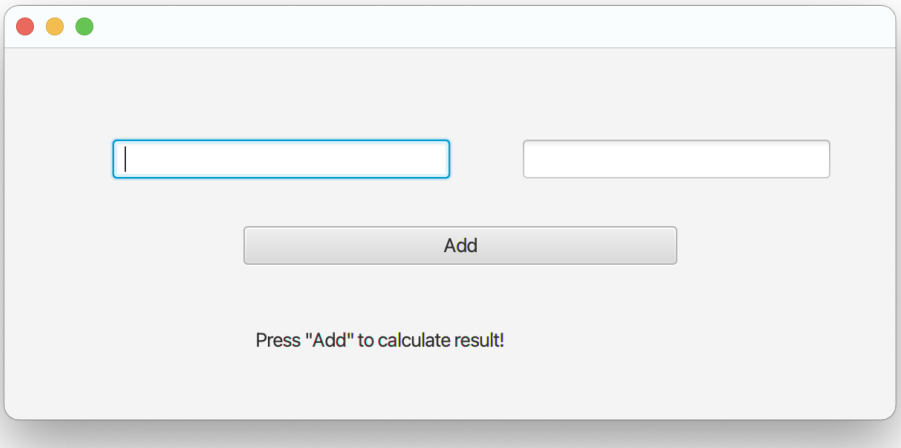
4. Acceder a servicios web desde un hilo diferente

Conectarse a un servicio web y obtener una respuesta puede ser una operación costosa, especialmente si la conexión a Internet no es la mejor y/o el servidor está sobrecargado. Si se accede a un servicio web en el hilo principal, la aplicación se bloqueará (no responderá) hasta que se obtenga el resultado.

La mejor manera de lidiar con los servicios web (o cualquier otra conexión remota) es usando un hilo separado para iniciar la conexión y luego procesar los resultados cuando se reciben. Si procesar esos resultados implica cambiar la vista en una aplicación JavaFX, se puede usar un servicio o el método *Platform.runLater()* (o *AsyncTask* en Android), como en este ejemplo que llama al ejemplo de servicio de suma mostrado en secciones anteriores.

```
public class GetSumService extends Service<Integer> {

    int n1, n2;

    public GetSumService(int n1, int n2) {
        super();
        this.n1 = n1;
        this.n2 = n2;
    }

    @Override
    protected Task<Integer> createTask() {
        return new Task<Integer>() {

            @Override
            protected Integer call() throws Exception {
                BufferedReader bufInput = null;
                Integer result = 0;

                try {
                    URL sumURL = new URL("http://localhost/services/sum/" + n1 + "/" + n2);
                    URLConnection conn = sumURL.openConnection();

                    bufInput = new BufferedReader(
                            new InputStreamReader(conn.getInputStream()));
                    result = Integer.parseInt(bufInput.readLine());

                } catch (IOException e) {
                } finally {
                    if (bufInput != null) {
                        try {
                            bufInput.close();
                        } catch (IOException e) {
                        }
                    }
                }

                Thread.sleep(5000); // simulate a 5 seconds delay!
                return result;
            }
        };
    }
}
```

La vista podría ser…



En el controlador de la aplicación, cuando se haga clic en el botón *"Add"*, se creará e iniciará el servicio, y se actualizará la etiqueta correspondiente cuando finalice:

```
private void sumNumbers(ActionEvent event) {
   gss = new GetSumService(
         Integer.parseInt(num1.getText()),
         Integer.parseInt(num2.getText()));

   gss.start();
   addButton.setDisable(true);
   imgLoad.setVisible(true);
   resultLabel.setVisible(false);

   gss.setOnSucceeded(e -> {
      resultLabel.setText("Result: " + gss.getValue());
      addButton.setDisable(false);
      imgLoad.setVisible(false);
      resultLabel.setVisible(true);
   });
}
```

Ejercicio 3

Crea un proyecto JavaFX llamado ***FXWebServiceExample*** y crea una aplicación JavaFX similar a la que se muestra arriba. Utiliza la clase *GetSumService* para acceder al servicio de suma y recuperar la suma de los dos dígitos enviados como parámetros.

Como en el ejercicio anterior, puedes utilizar los servicios de Node que se proporcionaron en esta sesión. En este caso, debes acceder a *localhost/services/sum* con los dos parámetros necesarios (por ejemplo, *localhost/services/sum/5/2* debería devolver 7 como resultado). También puedes cambiar el URI o el puerto en el proyecto Node, si lo deseas.