4. Acceso a servicios desde Java

Anexo I. Otras librerías

Programación de Servicios y Procesos

Arturo Bernal
Nacho Iborra
Javier Carrasco

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

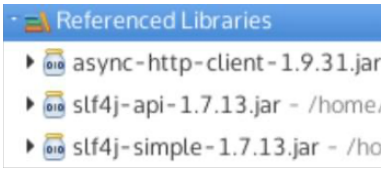
Tabla de contenido

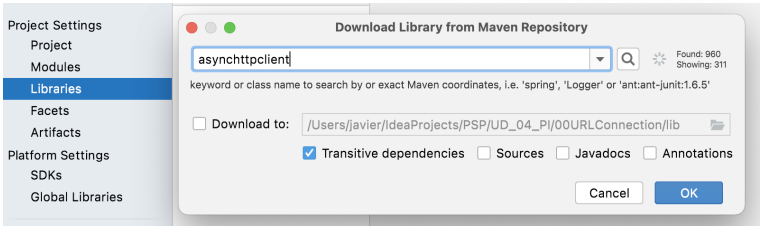
1. Acceso a servicios desde Java

A lo largo de esta unidad, aprenderás cómo conectarte a una URL y obtener su información, o cómo recibir y analizar datos JSON. Para este propósito, se presentarán las bibliotecas y clases más útiles o comunes, pero hay algunas otras bibliotecas que puedes utilizar para obtener los mismos (o similares) resultados.

1.1. *AsyncHttpClient*

Hay bibliotecas como AsyncHttpClient que simplifican mucho las conexiones HTTP. Si el proyecto no utiliza herramientas como *Maven* o *Gradle*, tendrás que descargar y añadir las bibliotecas necesarias (JAR) manualmente.






```java
public static void main(String[] args) {
    AsyncHttpClient asyncHttpClient = asyncHttpClient();

    Future<Response> f = asyncHttpClient.prepareGet("http://www.google.es").execute();

    try {
        System.out.println(f.get().getResponseBody());
        asyncHttpClient.close();
    } catch (IOException | InterruptedException | ExecutionException e) {
        e.printStackTrace();
    }
}
```

1.1.1. ¿Qué biblioteca es mejor para trabajar con conexiones HTTP?

Como siempre, utilizar una biblioteca que simplifique el trabajo tiene muchas ventajas. Sin embargo, esta biblioteca debe estar en desarrollo activo y debe ofrecer buena documentación para ser útil; de lo contrario, las desventajas podrían ser peores que las ventajas. Otras desventajas incluyen que si la biblioteca no admite una funcionalidad que necesitas, puede ser más difícil hacer que funcione para esa tarea (o tal vez lo admita, pero no está bien documentado y es difícil encontrar cómo hacer lo que deseas…).

Teniendo esto en cuenta, se utilizará la nativa ***HttpURLConnection*** (como se recomienda en Android) por muchas razones. Es fácil encontrar ayuda cuando hay algún problema (muy utilizado), se puede hacer cualquier cosa con ella (acceso de bajo nivel) y no es necesario incluir una biblioteca que tal vez no sea compatible con la plataforma o el desarrollador deje de mantenerla. Por contra, el código será más grande y más difícil de mantener, por lo que debe tenerse cuidado y mantener un código limpio y modular.

1.2. Utilizar la API org.json para obtener datos JSON

También has visto cómo utilizar la biblioteca GSON de Google para obtener y analizar datos JSON de un servicio web. Ahora se verá cómo usar la API org.json para hacer lo mismo. Esta biblioteca se puede utilizar, por ejemplo, en Android Studio sin tener que añadir ningún JAR externo.

Se utilizará el mismo ejemplo:

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

En primer lugar se deberá cargar el objeto JSON raíz y luego navegar desde ahí analizando su información o creando objetos Java a partir de los JSON correspondientes.

Es realmente útil crear un constructor en cada clase que necesite recibir un *JSONObject* que contenga los campos necesarios y crear el objeto desde allí:

```
public class Address {
    String city;
    String street;

    public Address(JSONObject addressJson) {
        city = addressJson.getString("city");
        street = addressJson.getString("street");
    }

    @Override
    public String toString() {
        return street + " ( " + city + " )";
    }
}

public class Person {
    String name;
    int age;
    List<Address> address;

    public Person(JSONObject personJson) {
        name = personJson.getString("name");
        age = personJson.getInt("age");

        JSONArray addressArray = personJson.getJSONArray("address");
        address = new ArrayList<>();

        addressArray.forEach(a -> {
            address.add(new Address((JSONObject) a));
        });
    }
```

```
    @Override
    public String toString() {
        String str = "Name -> " + name + "\nage -> " + age + "\nAddress →";

        for (Address a : address)
            str += "\n\t" + a.toString();

        return str;
    }
}
```

Finalmente, se necesita conectar al servicio web, obtener su respuesta y procesar la cadena que contiene el código JSON resultante. Se utilizará la clase auxiliar *GetServiceResponse*, explicada en el contenido principal de esta unidad.

```
public static void main(String[] args) {
    GetServiceResponse resp = new GetServiceResponse("http://localhost/services/example");
    String json = resp.getResponse();

    if (json != null) {
        // We load the main (root) object
        JSONObject object = (JSONObject) new JSONTokener(resp.getResponse()).nextValue();
        if (!object.getBoolean("error")) {
            Person p = new Person(object.getJSONObject("person"));
            System.out.println(p.toString());
        } else {
            System.out.println("There was an error in the request");
        }
    }
}
```

El resultado que debería imprimir este programa sería:

```
Name -> Peter
age -> 30
Address ->
     Some street 24 ( London )
     Other street 12 ( New York )
```