RPC en Android (android-json-rpc)

Gallardo Morales, Juan Carlos

Izquierdo Vera, Javier

Abstract. RPC permite conectar dispositivos, pudiendo realizar una conexión cliente servidor. En este texto se pretende usar esta tecnología para desarrollar un cliente en Android que permita a una aplicación móvil utilizar servicios RPC. Para este propósito se ha seleccionado y utilizado una biblioteca que sigue el formato JSON para la serialización de los datos. Se habla acerca de la biblioteca, de sus funcionalidades, de su instalación, y concluímos llevándolo a la práctica mediante el desarrollo de un pequeño ejemplo de cliente.

1 Introducción

RPC (Remote Procedure Call), en resumidas palabras, es una técnica de programación distribuida cuya finalidad principal es abstraer la comunicación entre dos equipos informáticos por medio de un middleware con la intención de ejecutar procedimientos remotos de otra máquina sin preocuparnos en cómo se establece la comunicación. Esta idea puede aplicarse sobre cualquier dispositivo, y para ello hay varios ejemplos de entornos RPC tales como Java-RMI, SOAP, CORBA, etc. [1,2]

Actualmente no tiene mucho sentido utilizar esta tecnología en dispositivos Android ya que se está desarrollando muy rápidamente la arquitectura REST (concretamente RESTful) donde se desarrollan servicios web listos para su uso desde cualquier otro dispositivo, algo que resulta muy cómo para programar aplicaciones cliente de forma muy rápida. Para ello el dispositivo cliente tan solo tiene que conectarse a un servidor web que ofrezca dicho servicio para invocarlo y poder usar el resultado devuelto. Dicho resultado podrá estar en diferentes formatos, de los cuales destacamos XML y JSON [3].

Si nuestro objetivo es ejecutar procedimientos de otra máquina en nuestro dispositivo Android, sin tener que depender de ningún servicio ofrecido en la web, tenemos disponibles varias bibliotecas que nos ofrecen una forma de implementar RPC en Android de forma muy sencilla. Concretamente vamos a centrarnos en una de las bibliotecas open Source más conocidas en la actualidad en cuanto a llamadas a procedimientos remotos se refiere, denominada **Android-rpc-json.** Como su nombre indica trata de facilitarnos el uso de RPC en Android por medio del formateo de datos en JSON [10].

adfa, p. 1, 2011.

2 Trabajos Relacionados

Actualmente existen muchas bibliotecas relacionadas con Android-rpc-json, entre las que destacamos las siguientes:

a. XML-RPC: es un protocolo de llamadas a procedimientos remotos (RPC) que trabaja sobre internet devolviendo los resultados en formato XML. Fue el comienzo de lo que hoy en día es conocido como SOAP. Actualmente parece estar bastante abandonado debido a su gran simplicidad, y por esta razón muchos desarrolladores han optado por crear nuevas bibliotecas a partir de esta para intentar solucionar los problemas que presenta, como aXMLRPC [4,5].
b. Apache XML-RPC: adaptación XML-RPC al lenguaje Java realizado por Apache Software Foundation. La web oficial fue editada por última vez en el año 2005, por lo que parece no estar en desarrollo actualmente. En dicha web existe una API muy extensa para su uso y podemos acceder directamente a su descarga de forma gratuita [6].
c. Android-xmlrpc: se trata de otra adaptación de XML-RPC a dispositivos Android, caracterizada por ser muy ligera. Hay muy poca información sobre ella y muy pocos ejemplos prácticos sobre su utilización. Desde code.google.com podemos ver que su última actualización fue subida en el año 2010, y su último commit fue en 2012 por lo que deja muchas dudas sobre su actual desarrollo [9].

3 Análisis

3.1 Descripción

Centraremos nuestra atención en la biblioteca "android-json-rpc" [10] creada y mantenida por una comunidad de usuarios bajo licencia MIT [11]. Como ya hemos mencionado en apartados anteriores, hay varias bibliotecas y proyectos que han pretendido dar soporte a este concepto, el motivo por el que hemos seleccionado esta biblioteca en concreto es porque ha sido mantenida hasta el año 2013. No es una fecha muy reciente pero sí de las más recientes, además de que es una biblioteca que tiene claro su objetivo y se centra en él, por lo que no es demasiado grande y no tiene mucho más de lo que necesitamos.
Esta biblioteca se centra exclusivamente en dar el soporte necesario para crear un cliente RPC en plataformas Android, proporcionando los métodos adecuados para crear el cliente con el host concreto, y para enviar y recibir datos del servidor, usando para ello el formato JSON, el cual se explicará más adelante. Todo ello se basa en JSON-RPC [7].

3.2 Requerimientos y clases ofrecidas

No hay ningún requerimiento concreto para utilizar esta biblioteca, simplemente hay que incluirla en el proyecto Android e importarla para poder usar sus funciones. Lo

único que tenemos que tener en cuenta es importar algunas utilidades de la biblioteca HTTP de apache que se requieren, lo cual no debería suponer un problema porque suele encontrarse en el entorno y es fácil de conseguir. Más tarde en el apartado de instalación se explicará cómo importar esta biblioteca.
Android-json-rpc ofrece documentación en formato html de todas sus clases y métodos, algo de lo que carecían algunas de las bibliotecas encontradas en Internet. Ésta biblioteca se compone de 7 clases principales: JSONEntity, JSONRPCClient, JSONRPCException, JSONRPCHttpClient, JSONRPCParams, JSONRPCThreadedClient, y JSONRPCThreadedHttpClient.
Vamos a ceñirnos a lo necesario para crear y hacer funcionar un cliente RPC por medio de esta biblioteca sin tener en cuenta todas las funciones incluidas, debido a que toda la documentación ya se encuentra detallada y sería redundante centrarnos en ello. La clases clases que utilizaremos son: JSONRPCClient, JSONRPCException y JSONRPCParams.

JSONRPCClient es la clase más importante, es la que compone la estructura del cliente (aunque también podríamos haber utilizado la clase JSONRPCHttpClient en caso de realizarlo usando http). Contiene los métodos necesarios para registrar el host con el que se establece conexión (create), para establecer el timeout (setConnectionTimeout), consultarlo (getConnectionTimeout), establecer el timeout del socket (setSoTimeout), obtenerlo (getSoTimeout), etc.
Además, también contiene los métodos que necesitaremos para realizar las peticiones al servidor, estos son los llamados métodos "call". Hay un call distinto para diversos tipos de variables, callBoolean, callDouble, callInt, callString, callLong, callJSONArray, etc. Cada call devuelve el tipo de variable que se especifica en su sintaxis, y recibe el nombre del método (String) que se va a llamar en el servidor, y los parámetros que se requieren para ello, de modo que estos métodos realizan la petición al servidor y devuelven la respuesta en el tipo indicado. Con la excepción del propio método "call", que devuelve un Object.

JSONRPCException se ha utilizado para controlar la posible excepción en caso de que la llamada al servidor falle, y JSONRPCParams, que no es más que un enum, se ha utilizado para indicar la versión de JSON-RPC que utiliza el servidor, la cual es necesaria indicar al enlazar el host. En este caso se utilizará la versión 2 de JSON-RPC, que es la más reciente [7].
Toda esta información puede encontrarse más detallada en la documentación de la biblioteca [10].

3.3 Formato de salida

Los datos de salida están formateados en JSON, formato de texto ligero para intercambio de datos. Se trata de una alternativa mucho más eficiente que XML y es soportada y apoyada directa o indirectamente por casi todos los lenguajes de programación modernos. Su estructura básica es la siguiente:

- Colección de pares nombre / valor, que puede estar representada por objetos, tablas hash, listas con claves, etc.
- Lista ordenada de valores, representada por matrices, vectores, arrays, etc.

Su implementación, de forma resumida, es la siguiente:

1. Objetos:

```
{string1: value1, string2: value2,…}
```

2. Arrays:

```
[value1, value2,…]
```

Donde value puede ser un objeto, array, booleano, número, string, null, etc.

En nuestro caso nos interesa utilizarlo en Java, lo que cual se realizaría de la siguiente forma:

- Convertir objetos Java a JSON (serialización): podemos utilizar la librería Gson para facilitarnos el trabajo. Por ejemplo podríamos serializar de forma estándar de la siguiente forma:

```
Empleado emp =  new Empleado();
Gson gson = new Gson();
String representacion = gson.toJson(emp);
```

- Convertir JSON a objetos Java (deserialización): usando la misma librería (Gson) podemos convertir una cadena Json a objetos Java de forma muy sencilla, veamos un ejemplo de uso estándar:

```
Gson gson = new Gson();
Type tipoEmp = new TypeToken<Empleado>().getType();
Emp = gson.fromJson(representación,tipoEmp);
```

Podemos ver más información de Json en [7] y más información de Gson en [8].

3.4 Instalación

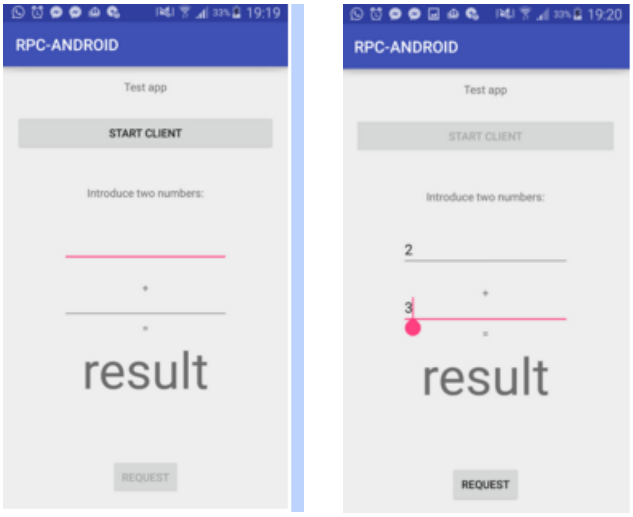
Como se ha mencionado anteriormente, la instalación es relativamente sencilla y se limita a descargar las bibliotecas e importarlas en el proyecto Android. En la web de la biblioteca android-json-rpc se habla de cómo importarla en eclipse [12], esto es debido a que no está actualizado, nosotros desaconsejamos el uso de eclipse para esta labor y a continuación explicaremos cómo importarla en Android Studio.

Una vez tengamos Android Studio y el proyecto Android cargado (la explicación de cómo se realiza la instalación y la creación del proyecto escapa del objetivo de este texto), y hayamos descargado la biblioteca desde su web, nos situamos en la estructura del proyecto (File → Project Structure...) y añadimos las dependencias requeridas mediante el botón “Add”. Otra alternativa es arrastrar las clases de java a la carpeta de

nuestro proyecto. Una vez realizado esto tendremos que especificar los “import” necesarios en el código, indicando el paquete y la clase. Por ejemplo, en caso de haberlos importado en el paquete “RPC”, habría que indicar “import RPC.clase_concreta;”. El proceso para importar las clases requeridas de Apache HTTP sería análogo, la biblioteca la podemos encontrar en la web de Apache [13].

3.5 Ejemplo de uso

Siguiendo el proceso de instalación mencionado, y utilizando las clases y métodos que se han explicado, se ha realizado un ejemplo sencillo de aplicación Android que suma dos números en un servidor remoto. Para ello la aplicación solicita al usuario iniciar el cliente, el cual se ha abstraído de la aplicación tratándose como un objeto, intentando reducir el acoplamiento entre ambos. Una vez iniciado el cliente, el usuario puede introducir dos números cualesquiera y pulsar el botón de sumar, este botón llama a un método del objeto cliente el cual llama al método “call” indicando los dos números a sumar, y el método que se encarga de sumar en el servidor, recibiendo así una respuesta y devolviéndoselo a la clase java que se encarga de controlar los eventos del layout, mostrándolo así como un resultado visible en la pantalla del dispositivo.
Esta es la interfaz de la aplicación:



Ejemplo de aplicación Android, suma de números [14]

El código de la aplicación será el siguiente:

```
package com.red.lifka.rpc_android;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class Main extends AppCompatActivity {

   private Button start_client;
   private EditText val1;
   private EditText val2;
   private TextView result_value;
   private Button launch;

   private Client client = null;
   private double x;
   private double y;
   private double result = 0.0;
   private int operation = 0;

   private String host = "localhost";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       // Enlazar objetos del layout
       start_client = (Button)findViewById(R.id.startclient);
       val1 = (EditText)findViewById(R.id.value1);
       val2 = (EditText)findViewById(R.id.value2);
       result_value = (TextView)findViewById(R.id.resultvalue);
       launch = (Button)findViewById(R.id.launch);
       launch.setEnabled(false);

       // Iniciar cliente
       start_client.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               // Clase cliente, abstrae el uso del cliente que ofrece
la biblioteca
               client = new Client(host);

               // Ajustar layout
               start_client.setEnabled(false);
               launch.setEnabled(true);
           }
       });

       // Request
       launch.setOnClickListener(new View.OnClickListener() {
           @Override
```

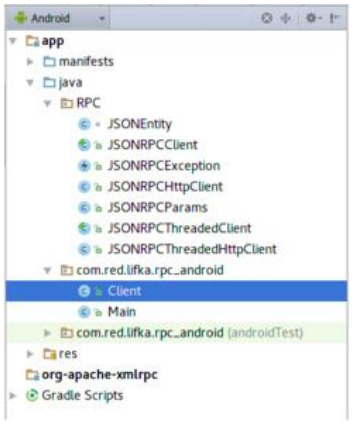
```
            public void onClick(View v) {
                // Obtener los números a sumar
                x =  new Double(val1.getText().toString());
                y =  new Double(val2.getText().toString());

                // Pedir a la clase cliente que lance la operación de su-
mar los dos números
                result = client.launch(x, y, operation);

                // Mostrar resultado en el layout
                result_value.setText(String.valueOf(result));
            }
        });

    }
}
```

El código cliente podemos verlo a continuación:

```
package com.red.lifka.rpc_android;
import android.util.Log;
import RPC.JSONRPCClient;
import RPC.JSONRPCException;
import RPC.JSONRPCParams;

public class Client {
   private JSONRPCClient client;
   private String host;

   Client(String host){
       Log.d("Info:", "Va a enlazar"); //Debug

       // Inicializar cliente
       client = JSONRPCClient.create(host, JSONRPCParams.Ver-
sions.VERSION_2);

       Log.d("Info:", "Enlazado"); //Debug
   }

   public double launch(double v1, double v2, int operation){

       // Set the connection timeout
       // setConnectionTimeout(int connectionTimeout)
       client.setConnectionTimeout(4000);
       client.setSoTimeout(4000);
       double result;

       // operation => SIEMPRE VA A SUMAR EN ESTE EJEMPLO
       // Intentamos lanzar la petición
       try {
           // Petición de Double
```

```
            result = client.callDouble("suma", v1, v2);
        } catch (JSONRPCException e) {
            e.printStackTrace();
            result = -1;
        }
        return result;
    }
}
```

Podemos ver la estructura de nuestro proyecto Android en la siguiente figura:



4 Referencias

1. http://ccia.ei.uvigo.es/docencia/SCS/1011/transparencias/Tema2-2.pdf
2. https://books.google.es/books?id=K9hnCJ_NGq4C&pg=PT500&lpg=PT500&dq=servicios+web+desde+android+arquitectura&source=bl&ots=KGDgREu7gJ&sig=unwp9Gmf-FOnubFQbhiCN4Z2HOig&hl=es&sa=X&ved=0ahUKEwiwuv2a2P7LAhVMXBoKH-YHxCd0Q6AEIRDAG#v=onepage&q=servicios%20web%20desde%20android%20arquitectura&f=false
3. http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb
4. XML-RPC: http://xmlrpc.scripting.com/spec.html
5. aXML-RPC: https://github.com/timroes/aXMLRPC
6. Apache XML-RPC: http://ws.apache.org/xmlrpc/
7. http://www.json.org/
8. https://github.com/google/gson
9. https://code.google.com/archive/p/android-xmlrpc/
10. https://code.google.com/archive/p/android-json-rpc/
11. https://opensource.org/licenses/mit-license.php
12. https://code.google.com/archive/p/android-json-rpc/wikis/GettingStarted.wiki
13. https://ws.apache.org/xmlrpc/client.html
14. https://drive.google.com/file/d/0B_4Dc8dm9rLxVkgzR2JCVENGVkU/view?usp=sharing