TP
Requête HTTP et appel de web services

Romain Raveaux

Objectif :

Création d'un client HTTP simple : Requête HTTP GET
Requête sur un web service SOAP.
Parser et afficher la réponse d'un web service SAOP.

1°) Requête HTTP

Créer une application simple avec un bouton et un TextView.
L'appuie sur le bouton déclenchera une requête http au serveur de google.

Le code ci-dessous permet de faire une requête simple au site de google :

```
URL url;
            HttpURLConnection urlConnection =null;
            try {
                  url = new URL("http://www.google.com/");
                  urlConnection = (HttpURLConnection) url.openConnection();

                  InputStream in = new
BufferedInputStream(urlConnection.getInputStream());
                  readStream(in);
                  urlConnection.disconnect();
            } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
                  urlConnection.disconnect();
            }
```

1°) Créer la fonction readStream qui affiche la page web (texte) dans un textview
La fonction readStram lit le flux d'entrée et le transforme en String pour l'afficher dans un TextView.
2°) Tester votre application. Que se passe-t-il ?

3°) Une première correction possible
Il est fortement déconseillé de faire des requêtes HTTP dans le même thread que celui de l'interface graphique.
Il est quand même possible d'outre passer cette limitation en mettant avant votre code de requêtage les lignes suivantes :

```
StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
```

Essayer ? Est ce que cela fonctionne ?

4°) Deuxième correction possible

La solution précédente fonctionne mais en faisant un peu le forcing en plus si la requête met du temps à s'exécuter alors vous aurez le droit à « L'application ne répond pas » de la part de l'OS. La bonne pratique est de créer une tache asynchrone s'exécutant en tache de fond puis une fois la requête effectuée le résultat est transmis au thread de la vue graphique pour affichage.

Implémenter la classe CallWebApi comme suit :

```
class CallAPI extends AsyncTask<String, String, String> {
	private TextView mTextView;
	public CallWebAPI(TextView mTextView){
		this.mTextView=mTextView;
	}
	@Override
	protected String doInBackground(String... params) {
		String inputLine = "";
		StringBuilder result=null;
		URL url;
		try {
			url = new URL("http://www.google.com");
			HttpURLConnection urlConnection = (HttpURLConnection)
url.openConnection();

			InputStream in = new
BufferedInputStream(urlConnection.getInputStream());

			BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
			result = new StringBuilder();
			String line;
			while((line = reader.readLine()) != null) {
				result.append(line);
			}


			in.close();
			return result.toString();

		}catch (Exception e){

		}
		return "error";
	}
	protected void onPostExecute(String result) {
		mTextView.setText(result);
	}
} // end CallAPI
```

Dan votre vue graphique, vous pouvez instancier la classe CallAPI de la manière suivante :

```
CallWebAPI c = new CallWebAPI(mTextView);
c.execute();
```

Les fonctions doInBackground et onPostExecute s'exécutent dans 2 threads différents.

PS : On aurait pu faire un service aussi ...

Il existe aussi une API de haut niveau appelée Volley mais non supportée par l'API 15 d'Android.
https://developer.android.com/training/volley/simple.html

5°) URL en paramètre

Dans l'exemple précédent l'URL est codée en dur dans la classe CallWebAPI.
Modifier cette classe pour pouvoir passer une URL en paramètre comme dans l'exemple ci-dessous.

```
URL newurl  = new URL("http://www.google.com");
CallWebAPI c = new CallWebAPI(mTextView);
c.execute(newurl.toString()) ;
```

Dans la classe CallWebAPi, les changements sont mineurs, il est possible de récupérer l'URL en paramétre de la fonction « doInBackground »

```
url =new URL(params[0])
```

6°) Application à un web service : Requête + Parsing

Aller sur le site « http://www.webservicex.net »
Vous y trouverez des webservices libres d'accès.
Regarder le webservice « geoipservice ».
Ce web service expose une méthode appelée GetGeoIP prenant un String en entrée (l'adresse IP) et retournant le nom du pays de l'adresse IP.

Exemple de requête :
http://www.webservicex.net/geoipservice.asmx/GetGeoIP?IPAddress=80.0.3.4
Données retournées :

```
<GeoIP>
<ReturnCode>1</ReturnCode>
<IP>80.0.3.4</IP>
<ReturnCodeDetails>Success</ReturnCodeDetails>
<CountryName>United Kingdom</CountryName>
<CountryCode>GBR</CountryCode>
</GeoIP>
```

Le but de cet exercice est de créer une application laissant l'utilisateur saisir une adresse IP. Après l'appuie sur un bouton l'application appelle le webservice GeopIP afin d'obtenir des informations de localisation de l'adresse IP. Finalement, votre application affiche le résultat dans une nouvelle fenêtre.
Voici les étapes :

1°) Saisie d'une adresse IP
2°) Construction requête HTTP GET
3°) Exécution de la requête
4°) Parser la requête pour créer un objet GeoIP
5°) Afficher le résultat dans une autre Activity

La classe GeopIP peut reprendre à minima les informations retournées par le service web :

```
public class GeoIP {
	Integer  ReturnCode;
	String IP;
	String ReturnCodeDetails;
	String CountryName;
	String CountryCode;
	//rajouter une méthode toString()
}
```

Une des difficultés est de parser les données XML renvoyées par le web service. Je vous propose une manière de réaliser ce parsing mais libre à vous de compléter.
Dans la classe CallWebAPi, ajouter les lignes suivantes dans la fonction « doInBackground »

```
	// Parse XML
			XmlPullParserFactory pullParserFactory;
			try {
				pullParserFactory = XmlPullParserFactory.newInstance();
				XmlPullParser parser =
pullParserFactory.newPullParser();

	parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
				parser.setInput(in, null);
				result = parseXML(parser);
			} catch (XmlPullParserException e) {
				e.printStackTrace();
			} catch (IOException e) {
				e.printStackTrace();
			}
```

La méthode parseXML peut ressembler à cela (à compléter) :

```
private GeoIP parseXML( XmlPullParser parser ) throws XmlPullParserException,
IOException {
			int eventType = parser.getEventType();
			GeoIP result = new GeoIP();
			while( eventType!= XmlPullParser.END_DOCUMENT) {
				String name = null;
				switch(eventType)
				{
				case XmlPullParser.START_TAG:
					name = parser.getName();
					if( name.equals("Error")) {
						System.out.println("Web API Error!");
					}
					else if ( name.equals("ReturnCode")) {
						result.ReturnCode =
Integer.parseInt(parser.nextText());
					}
					else if (name.equals("IP")) {
						result.IP = parser.nextText();
					}
					else if (name.equals("CountryName")) {
						result.CountryName = parser.nextText();
					}
					//ajouter d'autres champs
					break;
				case XmlPullParser.END_TAG:
					break;
				} // end switch
				eventType = parser.next();
			} // end while
			return result;
		}
```

Ce TP est inspiré du blog suivant :
http://blog.strikeiron.com/bid/73189/Integrate-a-REST-API-into-Android-Application-in-less-than-15-minutes