Conexiones HTTP

Existen dos bibliotecas que permiten la conexión HTTP contra un servidor web:

- **HttpURLConnection :** Biblioteca standard de java, se recomienda en versiones de Android mayores a Gingerbread por ser liviana.
- **Apache HTTP Components :** Las bibliotecas de Android incluyen las "Apache HTTP Components Libraries" las cuales son de código abierto y pueden usarse en java standard si se incluyen por separado, pero en las bibliotecas de Android fueron incluidas como parte de las standard por lo que no hay que incluirlas para usarlas.

Centraremos este material en el uso de la biblioteca de HttpURLConnection.

HttpURLConnection

Para realizar un request HTTP, deberemos crear un objeto del tipo HttpURLConnection el cual recibe como argumento un objeto del tipo URL, el cual es creado especificando la URL sobre la que se quiere realizar un request HTTP.
Una vez obtenido este objeto, podemos indicarle el protocolo HTTP a utilizar (GET, POST, PUT, PATCH, DELETE) y si vamos a obtener información o a leer información de la respuesta.
También podemos configurar los timeouts de conexión, y luego realizar el request, del cual obtendremos un objeto InputStream, que nos permitirá leer la información obtenida desde el servidor.

Creación de una clase HttpManager

Crearemos una clase HttpManager que nos simplificará la configuración y la obtención de datos permitiéndonos hacer requests http.

```
public class HttpManager {

	private String url;
	private HttpURLConnection conn;

	public HttpManager(String url)
	{
		conn = crearHttpUrlConn(url);
	}
```

En el constructor, recibimos la URL y creamos el objeto HttpURLConnection mediante el método *crearHttpUrlConn* el cual tendrá el siguiente contenido:

```
private HttpURLConnection crearHttpUrlConn(String strUrl)
{
	URL url = null;
	try {
		url = new URL(strUrl);
		HttpURLConnection urlConnection = (HttpURLConnection)
                                                        url.openConnection();
		urlConnection.setReadTimeout(10000 /* milliseconds */);
		urlConnection.setConnectTimeout(15000 /* milliseconds */);
		return  urlConnection;
	} catch (MalformedURLException e) {
		e.printStackTrace();
	} catch (IOException e) {
		e.printStackTrace();
	}
	return null;
}
```

Como se observa en el método, creamos un objeto URL mediante el cual podemos crear el objeto HttpURLConnection el cual devolvemos. En caso de error devolveremos null. Escribiremos ahora un método que nos indique si se pudo crear el objeto correctamente:

```
public boolean isReady(){
    return conn!=null;
}
```

Estamos en condiciones de escribir un método que realice un request del tipo GET y que nos devuelva un array de bytes con la respuesta. Si el request devolvía texto, luego convertiremos dicho array de bytes en un objeto String, si el request era sobre un archivo binario (un archivo de imagen por ejemplo) nos quedaremos con dicho array de bytes.

```
public byte[] getBytesDataByGET() throws IOException {
      conn.setRequestMethod("GET");
      conn.connect();
      int response = conn.getResponseCode();
      if(response==200) {
            InputStream is = conn.getInputStream();
            return readFully(is);
      }
      else
            throw new IOException();
}
```

En este método, configuramos al objeto HttpURLConnection para realizar un request del tipo GET y luego ejecutamos el método connect el cual es bloqueante y realiza la conexión.

NOTA: Es importante recordar que como el método es bloqueante y puede tardar varios segundos, deberemos ejecutar este método en un Thread.

Una vez que el método connect termina, podemos obtener el código de respuesta (200 OK, 404 página no encontrada, etc.) En el caso de obtener un 200, estamos en condiciones de leer la respuesta.

Leyendo la respuesta del request mediante InputStream

Para leer la respuesta del servidor, deberemos obtener un objeto InputStream del objeto HttpURLConnection, mediante el método getInputStream

```
InputStream is = conn.getInputStream();
```

Este stream tiene métodos que nos permitirá obtener los bytes del mensaje que respondió el servidor. Para ello creamos un método privado en nuestra clase HttpManager, que se encargue de ir leyendo los bytes desde el stream y devolver un array de bytes, llamamos a este método readFully.

```
private byte[] readFully(InputStream inputStream) throws IOException {
      ByteArrayOutputStream baos = new ByteArrayOutputStream();
      byte[] buffer = new byte[1024];
      int length = 0;
      while ((length = inputStream.read(buffer)) != -1) {
            baos.write(buffer, 0, length);
      }
      inputStream.close();
      return baos.toByteArray();
}
```

En el método readFully, creamos un objeto ByteArrayOutputStream, en el cual podemos ir concatenando bytes de forma eficiente, en este caso, creamos un buffer auxiliar de 1Kbyte en donde vamos leyendo los datos del InputStream, luego escribimos dicho buffer en el objeto ByteArrayOutputStream concatenándolos con los existentes.
Al terminar de leer el stream, simplemente retornamos un objeto array de bytes obtenido desde el ByteArrayOutputStream.

Por último agregamos un método que haga un request por GET y nos devuelva un String, para ello utilizaremos el método escrito previamente, y luego convertiremos el array de bytes a String:

```
	public String getStrDataByGET() throws IOException {
		byte[] bytes = getBytesDataByGET();
		return new String(bytes,"UTF-8");
	}
```

Subiendo datos por POST

Escribiremos un pequeño script en nuestro servidor, para que nos devuelva lo mismo que enviamos por POST, con la clave "data".

```
<?php
    echo($_POST["data"]);
?>
```

http://www.lslutnfra.com/alumnos/practicas/postEcho.php

De esta manera, si realizamos un request a dicha URL, pasándo un texto con la clave "data" por POST, obtendremos como respuesta del request, el mismo texto.

Agregaremos a nuestra clase HttpManager, un método para hacer un request POST

```
public byte[] getBytesDataByPOST(Uri.Builder postParams) throws IOException
{
	conn.setRequestMethod("POST");
	conn.setDoOutput(true);
	String query = postParams.build().getEncodedQuery();
	OutputStream os = conn.getOutputStream();
	BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os,
                                                                      "UTF-8"));
	writer.write(query);
	writer.flush();
	writer.close();
	os.close();

	int response = conn.getResponseCode();

	if(response==200) {
		InputStream is = conn.getInputStream();
		return readFully(is);
	}
	else
		throw new IOException();
}
```

La primera diferencia que encontramos es que pasamos un String con la palabra "POST" y ejecutamos el método setDoOutput, esto nos permitirá obtener un OutputStream para escribir los datos que queremos enviar al servidor.

Los datos clave-valor que queremos enviar por POST, estará dentro del objeto Uri.Builder, el cual recibiremos como argumento. Para crear este objeto y cargarle los pares clave-valor podemos utilizar el siguiente código:

```
Uri.Builder params = new Uri.Builder();
params.appendQueryParameter("data","Hola mundo");
params.appendQueryParameter("data2", "LSL UTN-FRA");
```

Dentro de nuestro método, transformamos este objeto Uri.Builder en un String con el formato correcto para el protocolo HTTP, mediante:

```
String query = postParams.build().getEncodedQuery();
```

Luego escribimos este String mediante el objeto OutputStream:

```
BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os,
                                                                  "UTF-8"));
writer.write(query);
writer.flush();
writer.close();
os.close();
```

Es importante destacar que en este caso no ejecutamos el método connect del objeto "conn" ya que la conexión se establecerá mediante el uso del OutputStream.

Luego obtenemos la respuesta de forma idéntica que en el método getBytesDataByGET.

NOTA: Es importante recordar que para que la aplicación pueda conectarse a Internet, es necesario aclarar el permiso de conexión en el archivo manifest.xml.

```
<uses-permission android:name="android.permission.INTERNET" />
```