

Tecnologías y Desarrollo en Dispositivos Móviles

Apartado 13: **Acceso a la red**

Autores:

Víctor M. Rivas Santos / Miguel Á. García Cumbreras
(Antonio Rueda Ruiz)

Introducción

- El acceso a la red es uno de los permisos que debe conceder el usuario

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

- Es imprescindible comprobar si realmente hay o no acceso a la red:

```
/**
 * Comprueba si hay o no acceso a la red
 * @return true si hay acceso, false si no lo hay
 */
public boolean isOnline() {
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    return ( networkInfo != null && networkInfo.isConnected() );
}
```

<https://developer.android.com/training/basics/network-ops/index.html>

Tipo de conexión activa

- Puede ser interesante detectar el tipo de red activa de modo que ciertas descargas (o subidas) solo se realicen con wifi:

```
/**
 * Comprueba si la conexión a red activa es la del tipo que se indica
 * @param type Alguno de los tipos que aparecen como constantes en ConnectivityManager
 *      ConnectivityManager.TYPE_BLUETOOTH, ConnectivityManager.TYPE_DUMM,
 *      ConnectivityManager.TYPE_ETHERNET, ConnectivityManager.TYPE_MOBILE
 *      ConnectivityManager.TYPE_MOBILE_DUN, ConnectivityManager.TYPE_WIFI,
 *      ConnectivityManager.TYPE_WIMAX
 * @return true si es del tipo indicado, false en otro caso.
 */
public boolean isType_Net(int type) {
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getNetworkInfo(connMgr.getActiveNetwork());
    return (networkInfo != null && networkInfo.getType() == type);
}

/**
 * Comprueba si la conexión a red activa es de tipo WIFI
 * @return true si es WIFI, false en otro caso.
 */
public boolean isWifi() {
    return isType_Net(ConnectivityManager.TYPE_WIFI);
}
```

¿Estás usando red? ¿De qué tipo?

MTV: Practicar las funciones de comprobación de red

- Haz una pequeña aplicación que indique de qué tipo es la conexión a red que estás usando en el emulador.
- ¿Cómo podrías comprobar que realmente la función *isOnline()* está funcionando correctamente?
- Sube a la plataforma un documento PDF que incluya:
 - Una captura de pantalla de la aplicación donde se vea el mensaje que indica la red que se está usando.
 - La respuesta a segunda pregunta.

NetworkOnMainThreadException

- Desde la versión 11 de la API, NO es posible lanzar conexiones desde el hilo principal
 - Se lanza la excepción
NetworkOnMainThreadException
- Es imprescindible lanzar una nueva hebra.

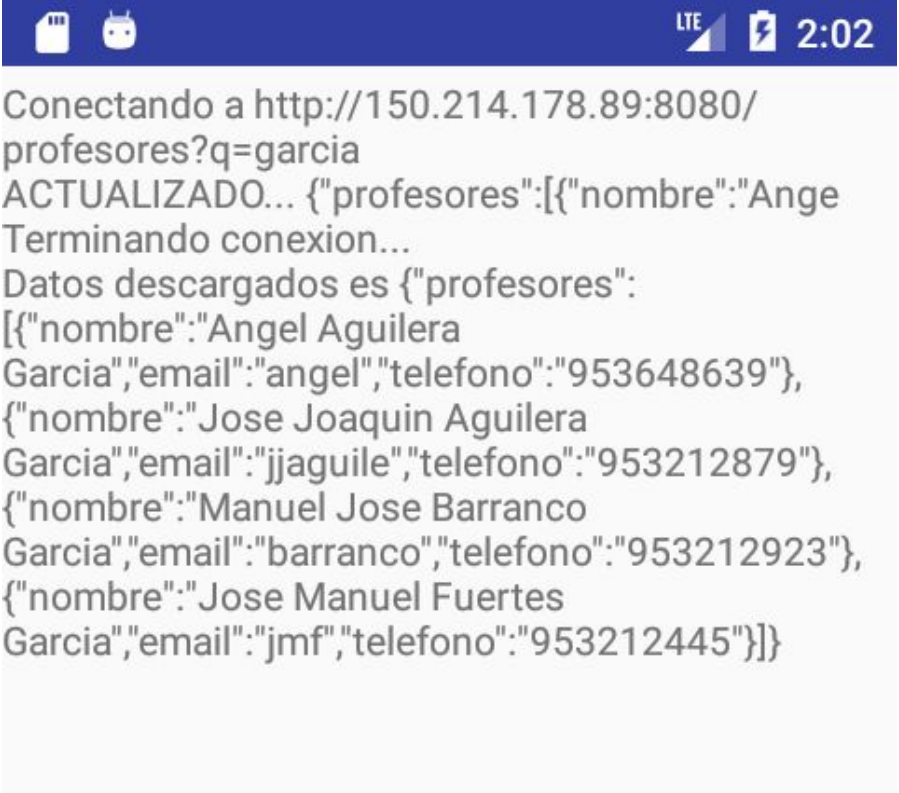
```
private class TareaAsincrona  
    extends AsyncTask<String  
        , Void  
        , TareaAsincrona.Resultado>
```

<https://developer.android.com/reference/android/os/AsyncTask.html>

Usamos un fragmento invisible

Ejemplo: Acceso_a_la_Red

- La filosofía es similar a la del uso de fragmentos:
 - Creamos un fragmento que contendrá la tarea asíncrona
 - Forzamos a la actividad principal a implementar una interfaz
- Además, definimos una nueva clase *Resultado* para devolver los datos leídos desde el servidor



The screenshot shows an Android application interface. At the top, there is a status bar with icons for signal, battery, and time (2:02). Below the status bar, the text "Conectando a http://150.214.178.89:8080/" is displayed. This is followed by "profesores?q=garcia" and "ACTUALIZADO...". Then, the text "Terminando conexion..." appears. Below this, the text "Datos descargados es {"profesores":..." is shown. The list of professors is displayed as follows:

```
{
  "nombre": "Angel Aguilera Garcia",
  "email": "angel",
  "telefono": "953648639",
  "nombre": "Jose Joaquin Aguilera Garcia",
  "email": "jjaguile",
  "telefono": "953212879",
  "nombre": "Manuel Jose Barranco Garcia",
  "email": "barranco",
  "telefono": "953212923",
  "nombre": "Jose Manuel Fuertes Garcia",
  "email": "jmf",
  "telefono": "953212445"
}
```

Código del fragmento (a)

```
public class FragmentoRed extends Fragment {

    private static ConexionCallback actividadContenedora = null;
    private TareaAsincrona tareaAsincrona = null;

    public static FragmentoRed nuevaInstancia(Context context
        , FragmentManager fragmentManager
        , String url) {
        FragmentoRed toRet = new FragmentoRed();
        if (context instanceof ConexionCallback) {
            actividadContenedora = (ConexionCallback) context;
        } else {
            throw new RuntimeException(context.toString()
                + " debe implementar la interfaz ConexionCallback");
        }
        Bundle args = new Bundle();
        args.putString(LA_URL, url);
        toRet.setArguments(args);
        fragmentManager.beginTransaction().add(toRet
            , ETIQUETA_IDENTIFICATIVA).commit();
        return toRet;
    }
}
```

Código del fragmento (b)

```
/**
 * Comienza una nueva TareaAsincrona que realiza la conexión con el servidor
 */
public void comenzarConexion() {
    cancelarConexion();
    tareaAsincrona = new TareaAsincrona(actividadContenedora);
    tareaAsincrona.execute(getUrl());
}

// Implementación de TareaAsincrona DENTRO del Fragmento.
// Permite descargar datos del servidor.
private class TareaAsincrona extends AsyncTask<String, Void, TareaAsincrona.Resultado> {
    private ConexionCallback<String> actividadContenedora;
    // Implementación de Resultado DENTRO de TareaAsíncrona.
    // UN Resultado puede ser una excepción o una cadena.
    class Resultado {
        public String datosDescargados;
        public Exception excepcion;

        public Resultado(String nuevosDatos) {
            datosDescargados = nuevosDatos;
        }

        public Resultado(Exception nuevaExcepcion) {
            excepcion = nuevaExcepcion;
        }
    }
}
```


El método doInBackground

@Override

```
protected TareaAsincrona.Resultado doInBackground(String... urls) {  
    Resultado toRet = null;  
    if (!isCancelled() && urls != null && urls.length > 0) {  
        String urlString = urls[0];  
        try {  
            URL url = new URL(urlString);  
            String resultString = conectarURL(url);  
            if (resultString != null) {  
                toRet = new Resultado(resultString);  
            } else {  
                throw new IOException("No se recibieron datos.");  
            }  
        } catch (Exception e) {  
            toRet = new Resultado(e);  
        }  
    }  
    return toRet;  
}
```

El método conectarURL

```
private String conectarURL(URL url) throws IOException {
    InputStream stream = null;
    HttpURLConnection conexion = null;
    String result = null;
    try {
        conexion = (HttpURLConnection) url.openConnection();
        conexion.setReadTimeout(READ_TIMEOUT);
        conexion.setConnectTimeout(CONNECT_TIMEOUT);
        conexion.setRequestMethod("GET");
        conexion.setDoInput(true);
        conexion.connect();
        int responseCode = conexion.getResponseCode();
        if (responseCode != HttpURLConnection.HTTP_OK) {
            throw new IOException("HTTP error code: " + responseCode);
        }
        stream = conexion.getInputStream();
        if (stream != null) {
            result = stream2string(stream, BUFFER_SIZE);
        }
    } catch (Exception e) {
        result="Excepción en conectarURL: " + e.getMessage();
    } finally {
        if (stream != null) stream.close();
        if (conexion != null) conexion.disconnect();
    }
    return result;
}
```

El método onPostExecute

@Override

```
protected void onPostExecute(Resultado resultado) {  
    if (resultado != null && actividadContenedora != null) {  
        if (resultado.excepcion != null) {  
            actividadContenedora.actualizarActividad(  
                resultado.excepcion.getMessage());  
        } else if (resultado.datosDescargados != null) {  
            actividadContenedora.actualizarActividad(  
                resultado.datosDescargados);  
        }  
        actividadContenedora.terminarConexion();  
    }  
}
```

Descargar datos desde la actividad principal

```
public void descargarDatos(String url) {
    fragmentoRed = FragmentoRed.nuevaInstancia(this, getFragmentManager(), url);
    if( fragmentoRed!=null && !estaDescargando ) {
        fragmentoRed.comenzarConexion();
        estaDescargando = true;
    } else {
        nuevoMensaje("No pudo lanzarse una nueva descarga para "+url );
    }
}

@Override
public void actualizarActividad(Object result) {
    datosDescargados=result.toString();
}

@Override
public void terminarConexion() {
    nuevoMensaje("Datos descargados es "+datosDescargados, false);
    estaDescargando = false;
    if (fragmentoRed != null) {
        fragmentoRed.cancelarConexion();
    }
}
```

La interfaz ConexionCallback

- Debe ser implementada por la actividad principal

```
public interface ConexionCallback<T> {  
    interface Progress {  
        int ERROR = -1;  
        int CONNECT_SUCCESS = 0;  
        int GET_INPUT_STREAM_SUCCESS = 1;  
        int PROCESS_INPUT_STREAM_IN_PROGRESS = 2;  
        int PROCESS_INPUT_STREAM_SUCCESS = 3;  
    }  
    public boolean isOnline();  
    public boolean isType_Net(int type);  
    public boolean isWifi();  
    void actualizarActividad(T result);  
    NetworkInfo getActiveNetworkInfo();  
    void actualizarProgreso(int codigo, int porcentaje);  
    void terminarConexion();  
}
```

Conclusión

- Sistema complejo para tratar la sincronización, sobre todo en un modelo FrontEnd-BackEnd-App
- La implementación vista solo permite una conexión activa:
 - Puede ser interesante deshabilitar la interfaz mientras se descargan los datos.