Servicios de red

En esta sesión se exponen varios casos típicos de uso de las conexiones de red. Antes de pasar a implementar una aplicación real es muy importante ver la última sección titulada "Operaciones lentas", pues todas las operaciones de red son lentas.

Para todas las conexiones por internet necesitaremos declarar los permisos en el `AndroidManifest.xml`, fuera del `application` tag:

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

Conexiones HTTP

Las conexiones por HTTP son las más comunes en las comunicaciones de red. En Android podemos utilizar la clase `HttpURLConnnection` en combinación con `Url`. Podemos ver información de las cabeceras de HTTP como se muestra a continuación (la información se añade a un TextView).

```
    TextView textView = (TextView)findViewById(R.id.TextView01);
    textView.setText("Conexión http.\n\n");
    try {
        textView.setText("Cabeceras www.escuelaartegranada.es:\n");
        URL url = new URL("http://www.escuelaartegranada.es");
        HttpURLConnection http = (HttpURLConnection)url.openConnection();
        textView.append(" longitud = "+http.getContentLength()+"\n");
        textView.append(" encoding = "+http.getContentEncoding()+"\n");
        textView.append(" tipo = "+http.getContentType()+"\n");
        textView.append(" response code = "+http.getResponseCode()+"\n");
        textView.append(" response message = "+http.getResponseMessage()+"\n");
        textView.append(" content = "+http.getContent()+"\n");
    } catch (MalformedURLException e) {
    } catch (IOException e) {
    }
```

Parsing de XML

En las comunicaciones por red es muy común transmitir información en formato XML, el ejemplo más conocido, depués del HTML, son las noticias RSS. En este último caso, al delimitar cada campo de la noticia por tags de XML se permite a los diferentes clientes lectores de RSS obtener sólo aquellos campos que les interese mostrar.

Android nos ofrece dos maneras de trocear o "parsear" XML. El `SAXParser` y el `XmlPullParser`. El parser SAX requiere la implementación de manejadores que reaccionan a eventos tales como encontrar la apertura o cierre de una etiqueta, o encontrar atributos. Menos implementación requiere el uso del parser Pull que consiste en iterar sobre el árbol de XML (sin tenerlo completo en memoria) conforme el código lo va requiriendo, indicándole al parser que tome la siguiente etiqueta (método `next()`) o texto (método nextText()).

A continuación mostramos un ejemplo sencillo de uso del `XmlPullParser`. Préstese atención a las sentencias y constantes resaltadas, para observar cómo se identifican los distintos tipos de etiqueta, y si son de apertura o cierre. También se puede ver cómo encontrar atributos y cómo obtener su valor.

```
try {
 URL text = new URL("http://www.escuelaartegranada.es");
```

```
    **XmlPullParserFactory parserCreator = XmlPullParserFactory.newInstance();
    XmlPullParser parser = parserCreator.newPullParser();
    parser.setInput(text.openStream(), null);**
    int parserEvent = **parser.getEventType()**;
    while (parserEvent != XmlPullParser.END_DOCUMENT) {

        switch (parserEvent) {
        case **XmlPullParser.START_DOCUMENT**:
            break;
        case **XmlPullParser.END_DOCUMENT**:
            break;
        case **XmlPullParser.START_TAG**:
            String tag = **parser.getName()**;
            if (tag.equalsIgnoreCase("title")) {
                Log.i("XML","El titulo es: "+ **parser.nextText()**);
            }else if(tag.equalsIgnoreCase("meta")){
                String name = **parser.getAttributeValue(
                    null, "name")**;
                if(name.equalsIgnoreCase("description")){
                    Log.i("XML","La descripción es:"+
                        parser.getAttributeValue(
                            null,"content"));
                }
            }
            break;
        case **XmlPullParser.END_TAG**:
            break;
        }

        parserEvent = **parser.next()**;
    }
} catch (Exception e) {
    Log.e("Net", "Error in network call", e);
}
```

El ejemplo anterior serviría para imprimir en el LogCat el título del siguiente fragmento de página web, que en este caso sería "Universidad de Alicante", y para encontrar el `meta` cuyo atributo `name` sea "Description", para mostrar el valor de su atributo `content`:

```
<html lang="es" style="height: auto;">
▼<head>
    <meta charset="utf-8">
    <title>Escuela de Diseño en Granada</title>
    <meta name="description" content="CURSOS, CARRERAS y CICLOS: Diseño Gráfico, Interiores, Diseño Web, Desarrollo de aplicaciones Web,
    Audiovisual, Fotografía, Animación 3D">
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
    <meta name="keywords" content="Centro formación profesional en Granada, cursos y carreras profesionales">
```

Cargar imágenes de red

Otro caso típico en el trabajo con HTTP es el de cargar imágenes para almacenarlas o bien mostrarlas. En el siguiente código descargamos una imagen a partir de su url, y la mostramos en un `ImageView`:

```
ImageView imageView = (ImageView)convertView.findViewById(R.id.FilaImagen);
try{
    InputStream is= new URL("http://https://www.escuelaartegranada.com/images/
   logo.png").openStream();  Drawable imagen = new
   BitmapDrawable(BitmapFactory.decodeStream(is));
   imageView.setImageDrawable(imagen);
}catch(MalformedURLException e1){
}catch(IOException e2){
}
```

El `ImageView` se define, en el layout, así:

```
<ImageView
    android:id="@+id/ImageView01"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:layout_marginRight="10dip"
    android:src="@drawable/icon" />
```

Estado de la red

En algunas aplicaciones puede convenir comprobar el estado de red. El estado de red no es garantía de que la conexión vaya a funcionar, pero sí que puede prevenirnos de intentar establecer una conexión que no vaya a funcionar. Por ejemplo, hay aplicaciones que requieren el uso de la WIFI para garantizar mayor velocidad.

A continuación se muestra cómo usar el `ConnectivityManager` para comprobar el estado de red.

```
ConnectivityManager cm = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo wifi = cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
NetworkInfo mobile = cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean hayWifi = wifi.isAvailable();
boolean hayMobile = mobile.isAvailable();
boolean noHay = (!hayWifi && !hayMobile); //¡¡Iiiinnteerneeeeeeer!!
```

El `ConnectivityManager` también puede utilizarse para controlar el estado de red, o bien estableciendo una preferencia pero permitiéndole usar el tipo de conectividad que realmente esté disponible,

```
cm.setNetworkPreference(NetworkPreference.PREFER_WIFI);
```

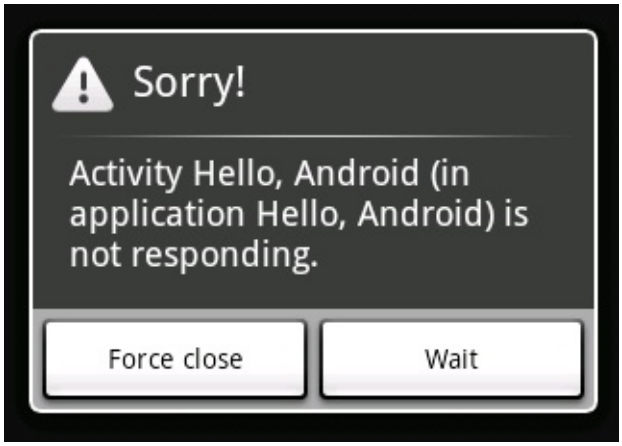
o bien pidiéndole explícitamente que se desconecte de la red móvil y se conecte a la red WiFi:

```
cm.setRadio(NetworkType.MOBILE,false);
cm.setRadio(NetworkType.WIFI,true);
```

Operaciones lentas

En Internet no se puede asumir que ninguna operación de red vaya a ser rápida o vaya a durar un tiempo limitado (el limite lo establece, en todo caso, el timeout de la conexión). En los dispositivos móviles, todavía menos, ya que continuamente pierden calidad de la señal o pueden cambiar de Wifi a 3G sin preguntarnos, y perder conexiones o demorarlas durante el proceso.

Si una aplicación realiza una operación de red en el mismo hilo de la interfaz gráfica, el lapso de tiempo que dure la conexión, la interfaz gráfica dejará de responder. Este efecto es indeseable ya que el usuario no lo va a comprender, ni aunque la operación dure sólo un segundo. Es más, si la congelación dura más de dos segundos, es muy probable que el sistema operativo muestre el diálogo ANR, "Application not responding", invitando al usuario a matar la aplicaicón:



Para evitar esto hay que crear otro hilo (`Thread`) de ejecución. Crearlo puede ser tan sencillo como:

```
ImageView imageView = (ImageView)findViewById(R.id.ImageView01);
new Thread(new Runnable() {
    public void run() {
      Drawable imagen = cargarLaImagen("http://...");
      //Desde aquí NO debo acceder a imageView
    }
}).start();
```

Pero hay un problema: tras cargar la imagen no puedo acceder a la interfaz gráfica porque la GUI de Android sigue un modelo de hilo único: sólo un hilo puede acceder a ella. Se puede solventar de varias maneras. Una es utilizar el método `View.post(Runnable)` .

```
ImageView imageView = (ImageView)findViewById(R.id.ImageView01);
new Thread(new Runnable() {
    public void run() {
      Drawable imagen = cargarLaImagen("http://...");
      imageView.post(new Runnable() {
        public void run() {
          imageView.setDrawable(imagen);
        }
      });
    }
}).start();
```

Otra manera es utilizar una `AsyncTask` . Es una clase creada para facilitar el trabajo con hilos y con interfaz gráfica, y es muy útil para ir mostrando el progreso de una tarea larga, durante el desarrollo de ésta. Nos facilita la separación entre tarea secundaria e interfaz gráfica permitiéndonos solicitar un refresco del progreso desde la tarea secundaria, pero realizarlo en el hilo principal.

```
TextView textView;
ImageView[] imageView;

public void bajarImagenes()
{
    textView = (TextView)findViewById(R.id.TextView01);
    imageView[0] = (ImageView)findViewById(R.id.ImageView01);
    imageView[1] = (ImageView)findViewById(R.id.ImageView02);
    imageView[2] = (ImageView)findViewById(R.id.ImageView03);
    imageView[3] = (ImageView)findViewById(R.id.ImageView04);

    new BajarImagenesTask().execute(
        "http://a.com/1.png",
        "http://a.com/2.png",
        "http://a.com/3.png",
        "http://a.com/4.png");
}
```

```
private class BajarImagenesTask
    extends AsyncTask<String, Integer, List<Drawable>>
{
     @Override
     protected List<Drawable> doInBackground(String... urls) {
          ArrayList<Drawable> imagenes = new ArrayList<Drawable>();
          for(int i=1;i<urls.length; i++){
            cargarLaImagen(urls[0]);
            **publishProgress(i);**
          return imagenes;
     }

     @Override
     protected void onPreExecute() {
       super.onPreExecute();
       textView.setText("Cargando imagenes...");
     }

     @Override
     protected void **onProgressUpdate(String... values)** {
         textView.setText(values[0] + " imagenes cargadas...");
     }

     @Override
     protected void onPostExecute(List<Drawable> result) {
       for(int i=0; i<result.length; i++){
         imageView[i].setDrawable(result.getItemAt(i));
       }
       textView.setText("Descarga finalizada");
     }

     @Override
     protected void onCancelled() {
         textView.setText("Cancelada la descarga");
     }
}
```

> La notación `(String ... values)` indica que hay un número indeterminado de parámetros, y se accede a ellos con `values[0], values[1],` ..., etcétera. Forma parte de la sintaxis estándar de Java.

Lo único que se ejecuta en el segundo hilo de ejecución es el bucle del método `doInBackground(String...)`. El resto de métodos se ejecutan en el mismo hilo que la interfaz gráfica. La petición de publicación de progreso, `publishProgress(...)` está resaltada, así como la implementación de la publicación del progreso, `onProgressUpdate(...)`. Es importante entender que la ejecución de `onProgressUpdate(...)` no tiene por qué ocurrir inmediatamente después de la petición `publishProgress(...)`, o puede incluso no llegar a ocurrir.