Curso de programación en Android

19/Junio/2012

Ramón Alcarria

Comunicación con el servidor

Parsing de mensajes:
SAX
DOM
JSON

Parsing de mensajes

Opciones:

- SAX
  - Analiza el documento XML por fragmentos a través de los llamados "event handlers", que tenemos que implementar.
- DOM
  - Almacena el documento XML en memoria y nos ofrece una serie de métodos para manipular y recorrer el arbol del documento
- XmlPull
  - Parecido a SAX pero tenemos control sobre la lectura del documento XML una vez iniciada: *parser.next()*

- Cual es mejor?
  - DOM puede ser más facil de implementar pero consume más memoria
  - SAX generalmente ofrece más rendimiento
  - XmlPull permite mayor flexibilidad pero es algo más complejo

Parsing de mensajes

SAX simplificado: En Android tenemos un SAX más facil de utilizar, que nos ahorra configurar el "event handler".

Permite asociar directamente las acciones a etiquetas XML concretas

USO:

Navegamos por la estructura del XML hasta llegar a las etiquetas que nos interesa tratar.

new RootElement("id") *root.getChild("id")*

Asignamos listeners disponibles: startEventListener / endEventListener y configuramos las acciones oportunas.

```
item.setStartElementListener (new StartElementListener(){….});
item.setEndElementListener (new EndElementListener(){….});
item.setEndTextElementListener (new EndTextElementListener(){….});
```

Parsing de mensajes

Clase android.sax.Element

void	setElementListener(ElementListener elementListener)Sets start and end element listeners at the same time.
void	setEndElementListener(EndElementListener endElementListener)Sets a listener for the end of this element.
void	setEndTextElementListener(EndTextElementListener endTextElementListener)Sets a listener for the end of this text element.
void	setStartElementListener(StartElementListener startElementListener)Sets a listener for the start of this element.
void	setTextElementListener(TextElementListener elementListener)Sets start and end text element listeners at the same time.

Ejercicio

Nuestro objetivo es analizar un documento RSS mediante SAX simplificado y extraer todos los campos interesantes.

El árbol del documento tiene la forma:

```
<rss>
  <channel>
     <title>……</title>
     <link> …… </link>
     <description>….</description>
      <lastBuildDate> … </lastBuildDate>
      <image> … </image>
      <item>
            <title> … </title>
            <link> … </link>
            <guid> … </guid>
            <description> … </description>
             <pubDate> … </pubDate>
      </item>
  </channel>
</rss>
```

Ejercicio

Nuestro objetivo es analizar un documento RSS mediante SAX simplificado y extraer todos los campos interesantes.

Vamos a crear una clase para almacenar el modelo de noticia:

- Titulo
- Link
- Guid (identifica univocamente al item)
- Descripción
- Fecha de publicación

Todas estas variables son strings

Crear getters y setters para todas las variables

Ejercicio

Nuestro objetivo es analizar un documento RSS mediante SAX simplificado y extraer todos los campos interesantes.

1. Ahora vamos a introducir el fichero logica.rss en res/raw

2. Para utilizar este recurso desde nuestra actividad lo recubrimos como un inputStream de la forma:

```
InputStream is = getResources().openRawResource(R.raw.logica);
```

3. Creamos un Vector de objetos de la clase Noticia para ir introduciendo las noticias que tengamos

4. Creamos un RootElement con id="rss"

5. Creamos sus hijos y configuramos los listener. En los métodos start y end de los listener vamos guardando los parámetros de la noticia

6. Finalmente, utilizamos el método parse de la clase android.util.xml

parse (InputStream in, Xml.Encoding encoding, ContentHandler contentHandler)

```
Xml.parse(is, Xml.Encoding.UTF_8, root.getContentHandler());
```

Parsing de mensajes

DOM: El documento XML se lee completamente antes de poder realizar ninguna acción. Se devuelve el contenido en forma de una estructura de tipo árbol

USO:

Instanciamos una fábrica DOM y creamos un nuevo parser a partir de ella:

```
//Instanciamos la fábrica para DOM
 DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

//Creamos el parser y le pasamos el inputstream
 DocumentBuilder builder = factory.newDocumentBuilder();
           Document dom = builder.parse(is);
```

Obtenemos el elemento raíz de nuestro árbol con: `dom.getDocumentElement()`

Métodos útiles:

element.getElementsByTagName("id") //Nos devuelve todos los sucesores de nombre "id"
node.getChildNodes() //Nos devuelve una lista de los nodos hijos
node.getNodeName() //Nos devuelve el nombre de la etiqueta
node.getTextContent() //Devuelve el texto del contenido del nodo

JSON: Acceso Web

Además de XML, muchos de los mensajes de servicios Web están codificados en JSON

Características:

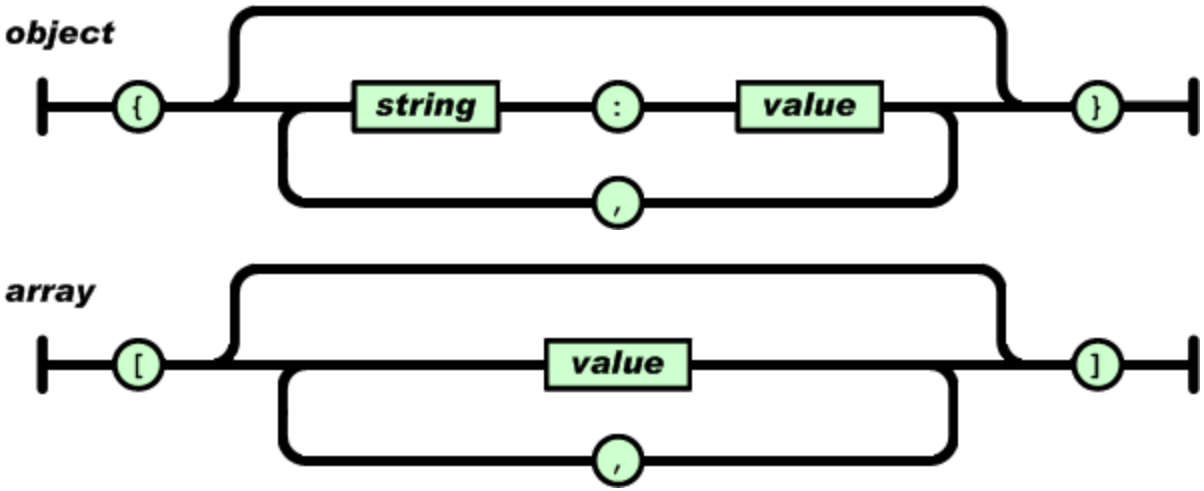
- Formato ligero
- Fácil de parsear
- No tiene características complejas como NameSpaces o extensiones
- Soportado por la mayoría de lenguajes

JSON:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

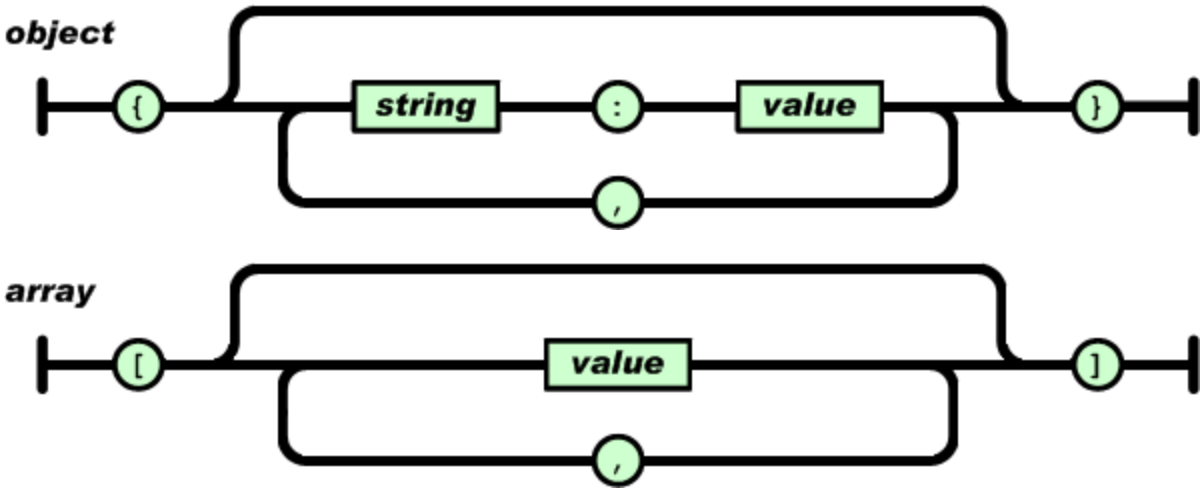


JSON:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```



JSON USO:

Una vez obtenido el texto JSON queremos transformarlo a un array de JSON Objects, llamado JSONArray.

Obtener un JSONArray es tan facil como crear el objeto pasándole el texto JSON como String

```
JSONArray jsonArray = new JSONArray(StringJSON);
```

Para obtener un JSONObject utilizamos el siguiente método de la clase JSONArray:

```
JSONObject jsonObject = jsonArray.getJSONObject(index);
```

Para trabajar con JSONObjects podemos utilizar los métodos *put* y *get*

Ejercicio:

Crear una clase para acceder a un Feed del Twitter en JSON y representar los titulares en un TextView.

1. En nuestra actividad principal, crear un método que acceda a una dirección Web y devuelva la respuesta como un String.

```
HttpClient client = new DefaultHttpClient();
HttpGet httpGet = new HttpGet("https://twitter.com/statuses/user_timeline/logicaes.json");

HttpResponse response = client.execute(httpGet);
HttpEntity entity = response.getEntity();
String result = EntityUtils.toString(entity);
```

Recuerda generar try/catch

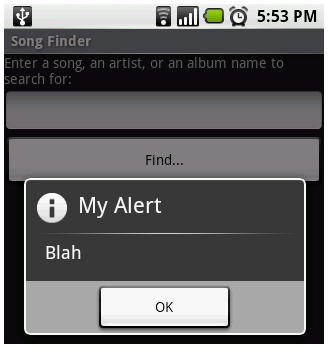
2. En el onCreate() procesamos el String del método anterior creando un JSONArray y recorriendolo para sacar el valor del campo "text" de cada JSONObject

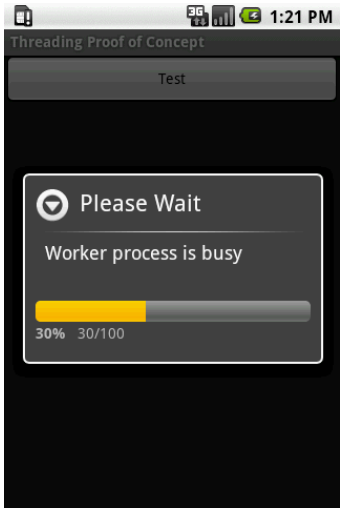
3. Representamos estos valores en un TextView
3.1 Recuerda introducir el TextView dentro de un ScrollView para que la información por pantalla pueda deslizarse.

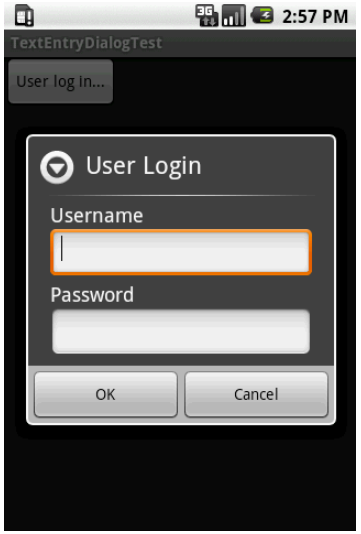
Diálogos:

Ventana emergente que capta el foco hasta que el usuario lo cierra o el proceso termina.

- Alert Dialog: Gestiona hasta 3 botones
- Progress Dialog: Muestra una barra de progreso
- DatePickerDialog: Permite al usuario seleccionar una fecha
- TimePickerDialog: Permite al usuario seleccionar una hora







Diálogos:

Alert dialog:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("¿Quieres terminar la actividad?");
builder.setCancelable(true);
builder.setPositiveButton("Si", new OkOnClickListener());
builder.setNegativeButton("No", new CancelOnClickListener());
AlertDialog dialog = builder.create();
dialog.show();

private final class CancelOnClickListener implements
DialogInterface.OnClickListener {
public void onClick(DialogInterface dialog, int which) {
Toast.makeText(getApplicationContext(), “La Actividad continúa",
Toast.LENGTH_LONG).show();}}

private final class OkOnClickListener implements
DialogInterface.OnClickListener {
public void onClick(DialogInterface dialog, int which) {
AndroidDialogsActivity.this.finish();}}
```

Progress Dialog:

```
ProgressDialog progressDialog;
progressDialog = new ProgressDialog(this);
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressDialog.setMessage("Loading...");
progressDialog.setCancelable(false);
progressDialog.show();
```

```
//Para avanzar el progreso de la barra
progressDialog.setProgress(int progress);

//Para cerrar el diálogo
progressDialog.dismiss();
```

Sabríamos crear una hebra para avanzar en el progreso y cuando llegue a 100% cerrar el diálogo?

```
Thread.sleep(1000);
```

MIT App Inventor

Creación de aplicaciones de manera sencilla para android

Setup:
http://beta.appinventor.mit.edu/learn/setup/#setupComputer

Tener cuenta de google
Tener el Java instalado
Descargar AppInventor

MIT App Inventor

Nuestra primera aplicación:

Gatito

http://beta.appinventor.mit.edu/learn/setup/hellopurr/hellopurrphonepart1.html
http://beta.appinventor.mit.edu/learn/setup/hellopurr/hellopurrpart2.html

FIN

Muchas gracias!