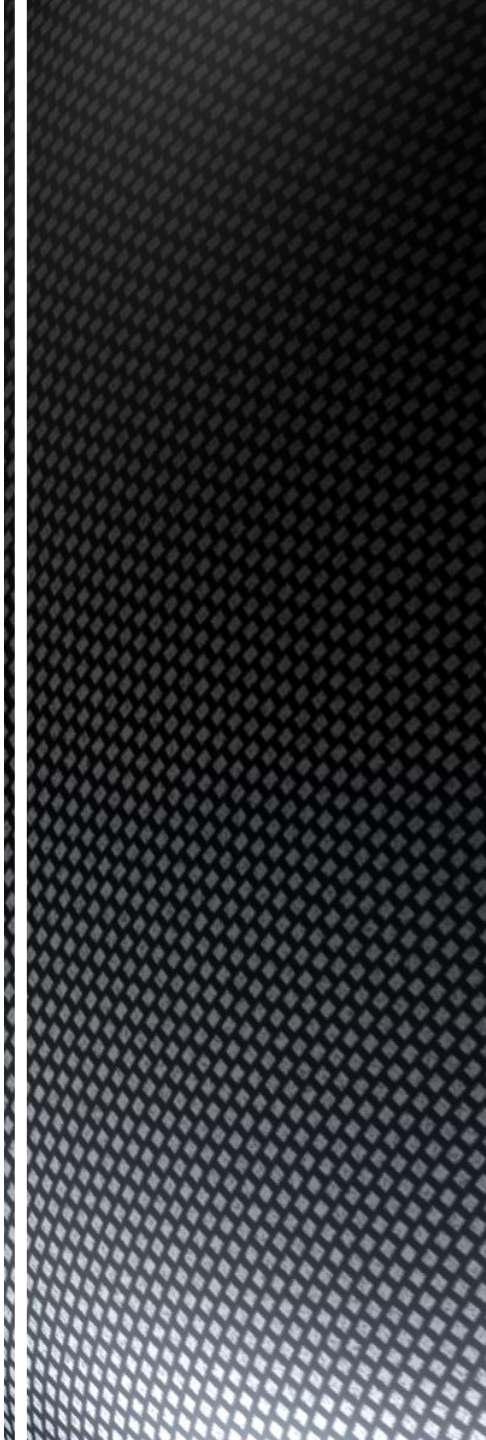
Android

Procesar XML



XML

Java dispone de gran cantidad de APIs para trabajar con XML, pero no todas están disponibles desde Android.

Librerías disponibles:

- Java's **S**imple **A**PI for **X**ML (SAX) (paquetes org.xml.sax.*)
- **D**ocument **O**bject **M**odel (DOM) (paquetes org.w3c.dom.*)

Leer y escribir ficheros XML es muy laborioso, pero potente y versátil.

Las dos alternativas más importantes son SAX y DOM.

El planteamiento es bastante diferente entre ambas alternativas.

Tras ver los ejemplos podrás decidir que herramienta se adapta mejor a tus gustos personales o al problema en concreto a resolver.

XML

Está compuesto por una serie de tags con apertura y cierre, también llamados elementos.

Dentro de los tags está el dato que va asociado al tag que lo engloba.

Los elementos (o tags) pueden tener atributos.

Los elementos pueden anidar a otros elementos.

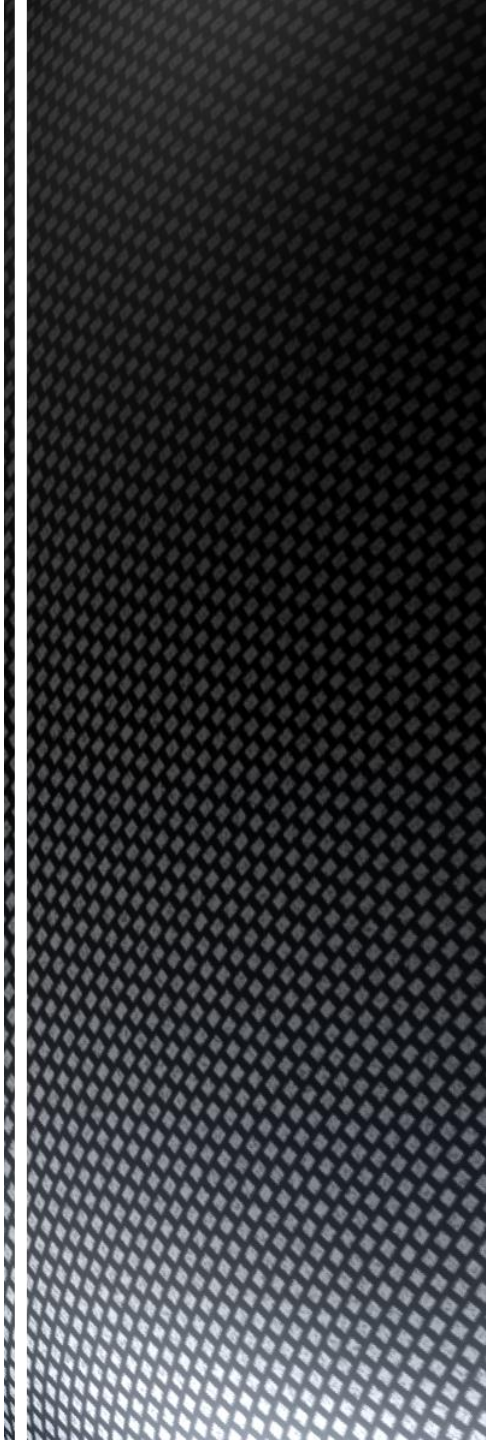
```
<?xml version="1.0" encoding="UTF-8"?>
<citas_celebres>
  <autor nombre="Arquímedes">
            <cita>Dadme un punto de apoyo y moveré el mundo</cita>
            <url>http://es.wikipedia.org/wiki/Arqu%C3%ADmedes</url>
            <drawable>arquimedes</drawable>
   </autor>
   <autor nombre ="Confucio">
            <cita>El silencio es un amigo que jamás traiciona</cita>
            <url>http://es.wikipedia.org/wiki/Confucio</url>
            <drawable>confucio</drawable>
   </autor>

    ...

</citas_celebres>
```

Android

Procesar XML con SAX



SAX - XML

Ventajas

SAX no utiliza muchos recursos, por lo que es muy indicado para dispositivos móviles con posos recursos.

SAX permite leer y procesar (parsear) un documento XML conforme se va leyendo, línea a línea, sin necesidad de almacenarlo en memoria completamente para su análisis.

Por tanto es ideal para procesar documentos muy grandes que no tengan que modificarse dinámicamente.

Inconvenientes

Pero es necesario definir un conjunto de callbacks para poder procesar los elementos del XML conforme se procesen.

Esto es costoso de construir sobre todo si el conjunto de etiquetas del fichero XML es muy variado.

Es necesario por tanto conocer la estructura del documento XML a parsear.

Fundamentos

SAX no almacena los datos, éstos son parseados y es nuestra responsabilidad almacenarlos en alguna **estructura de datos**.

Nosotros **tenemos que crear el parser**, que determinará que hacer con cada elmento que nos llegue de SAX.

SAX genera una serie de eventos conforme analiza el documento, para capturar estos eventos **tenemos que definir los callbacks** correspondientes.

SAX - XML

```
<?xml version="1.0" encoding="UTF-8"?>
<citas_celebres>
  <autor nombre="Arquímedes">
            <cita>Dadme un punto de apoyo y moveré el mundo</cita>
            <url>http://es.wikipedia.org/wiki/Arqu%C3%ADmedes</url>
            <drawable>arquimedes</drawable>
  </autor>
  <autor nombre ="Confucio">
            <cita>El silencio es un amigo que jamás traiciona</cita>
            <url>http://es.wikipedia.org/wiki/Confucio</url>
            <drawable>confucio</drawable>
  </autor>

   ...

</citas_celebres>
```

Eventos

Para el XML de ejemplo estos serían parte de los eventos que se producirían.

```
Start element : citas_celebres
Start element : autor
              : with attr. nombre="Arquimedes"
Start element : cita
Characters    : Dame un punto de apoyo y moveré
                el mundo
Ends element  : cita
Start element : url
Characters    : http://es.wikipedia.org/wiki/ -
                Arqu%C3%ADmedes
Ends element  : url

....

Ends element  : autor
```

SAX - XML

Callbacks

Los tipos de eventos fundamentales son:

startDocument()

Comienza el documento XML

endDocument()

Termina el documento XML

startElement(Srting tag_uri, String tag_name, String tag_qualified, Attributes tag_attributes)

tag_uri: La URI del espacio de nombres o vacío si no se ha definido.

tag_name: nombre de la etiqueta sin prefijo

tag_qualified: nombre cualificado de la etiqueta con prefijo o vacio si nombres con prefijo no están disponibles

tag_attributes: Una lista de los atributos vinculados en la etiqueta

endElement(String tat_uri, String tag_name, String tag_qualified)

Termina un elemento o tag

charcters(char ch[], int start, int length)

Devuelve el contenido de una etiqueta en el array ch[]

Para obtener un string de estos caracteres usaremos:

```
String s = new String(ch,start,length);
```

SAX - XML

Handler

Para implementar estos callbacks tenemos que crear un handler (manejador), extendiendo de la clase DefaultHandler

En nuestro Handler tendremos que sobercargar @Override los callbacks necesarios.

```
class CitasXMLHandler extends DefaultHandler{

   ... variables y estructuras de datos

   public CitasXMLHandler( ...  ){
            super();
   }

   @Override
   public void startDocument() throws SAXException{
   ...
   }

   @Override
   public void startElement(String tagUri, String tagName, String tagCualif, Attributes tagAtr)
           throws SAXException{
   ...
   }

   @Override
   public void characters(char cChars[], int cBegin, int cLength){
   ...
   }

   @Override
   public void endElement(String tagUri, String tagName, String tagCualif) throws SAXException{
   ...
   }

}
```

SAX - XML

Lectura de los datos.

Podemos definir un método en la clase que queramos para leer el fichero XML.

Por ejemplo:

```
public void leerXML (InputStream ficheroXML) thorws Exception {
    ...
}
```

Este método instanciará un SAXParserFactory llamando a newInstance de la clase estatica.

Del objeto SAXParseFactory, llamando al método newSAXParser instanciará un SAXParser

Del objeto SAXParser obtendremos un XMLReader llamando al método getXMLReader.

Por otro lado creamos un objeto de nuestra clase Handler, siguiendo con el ejemplo anterior: CitasXMLHandler

Al objeto XMLReader le asignamos el handler y finalmente llamamos a su método parse() pasándole como parámetro un objeto InputSource (fuente XML) a cuyo constructor le pasamos el nombre fichero xml (un FileInputStream).

Estos pasos son casi siempre los mismos, lo que tenemos que parametrizar nosotros es el nombre del fichero XML y crear el objeto XMLHandler para que el XMLReader llame a sus callbacks, ver el ejemplo:

```
public void loadXMLFile() throws Exception{
FileInputStream instream = new FileInputStream(XML_FILE);

SAXParserFactory fabrica = SAXParserFactory.newInstance();
SAXParser parser = fabrica.newSAXParser();
XMLReader lector = parser.getXMLReader();
CitasXMLHandler citas_handler = new
CitasXMLHandler(autores);
lector.setContentHandler(citas_handler);
lector.parse(new InputSource(instream));
}
```

SAX - XML

Consideraciones

Los callbacks de nuestro manejador CitasXMLHandler van a ser llamados por Android cada vez que suceda su evento relacionado, y ahí termina la responsabilidad de Android.

Conforme vayamos recibiendo los callbacks, puesto que sabemos la estructura del documento, podremos ir almacenando la información en estructuras de datos simples o complejas, como queramos.

Como hemos visto en el ejemplo de función loadXMLFile(), nuestro objeto handler será pasado a un XMLReader, que procesará el fichero XML pero no recibirá ni hará nada con los datos del XML ni devolverá nada, ninguna estructura de datos.

El XMLReader simplemente establece el handler y llama al método parse()

Es nuestra responsabilidad de gestionar los datos en el Handler.

Por tanto, respecto a las estructuras de datos para almacenar los leidos del XML tenemos dos opciones:

- Definirlas, instanciarlas y gestionarlas dentro del Handler.
- Definirlas e instanciarlas fuera del handler y recibir en el handler una referencia a esta instancia.

En el ejemplo se ha optado por pasarle al constructor de Handler una referencia a la estructura de datos donde se debe almacenar la información leida del fichero XML.

En este caso se trata de un ArrayList<Autor>, es decir una lista de objetos autor, que está accesible desde el método loadXMLFile() puesto que pertenece a la clase CitasCelebres donde instancio ese objeto.

SAX - XML

Consideraciones

Por tanto el constructor de nuestro Handler recibirá una referencia a un objeto de tipo ArrayList<Autores>

Esta referencia la puedo guardar en una variable miembro del handler del mismo tipo para que esté disponible en todos los callbacks.

Recordar que hay que llamar al constructor por defecto del handler cuando definimos nuestro constructor.

```
class CitasXMLHandler extends DefaultHandler{
   private Autor l_autor;
   private ArrayList<Autor> l_autores;

public CitasXMLHandler(ArrayList<Autor> l_autores){
   super();
   this.l_autores = l_autores;
}
```

SAX - XML

Handler Callbacks

Como sabemos los callbacks serán llamados conforme se procese el documento XML

Las variables miembro del handler declaradas se inicializarán en el constructor o en el callback startDocument().

El callback characters() será llamado tantas veces como el parser necesite para procesar toda la información contenida entre los tags de un elemento, por tanto, tendremos que concatenar, en cada llamada, lo que había con lo que llega en la nueva llamada.

Para ello utilizaremos un StringBuidler, declarándolo como miembro del handler e instanciandolo en startDocument().

En el callback startElement() colocaremos una estructura switch o varios ifs para determinar cual es el tag que empieza, porque así sabremos de que tag son los posibles atributos que llegan. Además como empieza un nuevo elemento, establecemos a 0 la longitud del StringBuilder que usamos para almacenar su contenido.

Si el tag que comienza tiene atributos podremos acceder a ellos mediante el método getValue("nombre_atributo") del parámetro Attributes que recibimos.

```
public void startElement(String tagUri, String tagName, String tagCualif, Attributes tagAtr)
                         throws SAXException{
  tag_content.setLength(0); //Inicializamos el StringBuilder
  if (tagName.equals("autor")){
      //ya sabemos que el tag es autor y como tiene un atributo lo capturamos.
      str_nombre = tagAtr.getValue("nombre").toString();
  }
}
```

SAX - XML

Handler Callbacks

En el callback characters() concatenamos con el método append() del StringBuilder el contendio a lo que había. Cuando termine de cargar el contenido, y por tanto de llamar a este callback, se llamará a endElement()

El Callback endElement() es donde normalmente haremos las acciones pertinentes con los datos que hemos ido recibiendo, atributos y contendio, en función del tag que termina.

Así tendremos también una estructura de ifs o switch para determinar que hacer.

```
public void endElement(String tagUri, String tagName, String tagCualif) throws SAXException{
if (tagName.equals("autor")){
  ... Creamos un nuevo objeto autor y a su constructor le pasamos los datos leidos con otros tags ...
  ... al autor que hemos creado le añadimos la cita que se leyó con el tag "cita" ...
  ... añadimos el autor al array de autores que recibimos en el constructor.
  }
  //Cuando terminan los otros tags, simplemente guardamos temporalmente los datos en strings miembro
  else if (tagName.equals("cita")) str_cita = cadena.toString();
  else if (tagName.equals("url"))  str_url = cadena.toString();
  else if (tagName.equals("drawable")) str_drawable = cadena.toString();
}
```

Finalmente en el método endDocument() podemos hacer cualquier acción de finalización necesaria. En el ejemplo no es necesario.

SAX - XML

Generar un fichero XML

Análogamente al método leerXML() podríamos crear un método escribirXML() en la clase oportuna, donde tengamos acceso a los datos a guardar. En el ejemplo, sería CitasCebebres.

Suponer que la aplicación permite crear autores y citas, entonces deberíamos poder guardar en un fichero XML el ArrayList<Autor> completo de forma que no perdamos los autores que hemos creado y sus citas asociadas.

El método escribirXML(OutputStream osXML) recibe como parámetro un OutpuStream donde volcar el XML.

El volcado lo realiza con un objeto XMLSerializer

Se instancia un XMLSerializer.

Se establece la codificación del fichero con setOutput()

Se llama a continuación a startDocument(osXML, "UTF-8") para generar la línea inicial con la codificación del fichero. `<?xml version="1.0" encoding="UTF-8"?>`

A continuación con startTag() creamos los tags.

Podemos realizar un bucle si hay repeticiones en el patrón de tags.

Con attribute() a continuación de startTag() añadimos atributos al tag.

Con text() a continuación de startTag() colocamos el contenido del tag.

Ejercicio E0604 CitasCelebres