

Mapeo de XML a Java (parte 1)

Fecha de creación: 15.05.2001

Revisión 1.0.1 (15.10.2002)

Alberto Molpeceres (al AT javahispano DOT org)



<http://www.javaHispano.org>

Copyright (c) 2002, Alberto Molpeceres. Este documento puede ser distribuido solo bajo los términos y condiciones de la licencia de Documentación de javaHispano v1.0 o posterior (la última versión se encuentra en <http://www.javahispano.org/licencias/>).

Mapeo de XML a Java (parte 1)

*En este artículo se plantea una introducción, totalmente funcional, a la lectura de ficheros XML y su mapeo a clases de Java con el **SAX (Simple Api for Xml)**. Así mismo, se explicara por que en muchas ocasiones este API es superior al **DOM (Document Object Model)**, aunque su manejo pueda resultar mas confuso.*

En este artículo dejaremos de lado (de momento) bastantes aspectos relacionados con la manipulacion de XML desde Java, como pueden ser los distintos tipos de parsers y funciones avanzadas, como la validacion.

Así mismo, se supone un cierto conocimiento de XML, su estructura y su funcion, ya que no se explicara en este artículo.

APIs: SAX vs DOM

Pasar un fichero XML a clases de Java es un proceso de **parsing**, hay que leer y procesar todo el fichero para hacerlo, por lo que si tenemos que leer gran cantidad de XML nos comeremos la memoria del sistema de forma rapida, recordando que como todos los procesos de parsing, no es un proceso rapido.

Aqui tengo que recordaros, que tanto SAX como DOM son dos APIs, no dos parsers, son dos ideas de como afrontar un problema, y son muchos los parsers existentes (ver recursos) que implementan estas APIs. Tambien existe una version de DOM optimizada para Java, llamada JDOM, pero su funcionamiento es muy similar al DOM.

Quizas el API mas usado sea el DOM, pero no por ello es el mejor. El DOM lee todo el contenido del fichero de una pasada, y crea en memoria una estructura jerarquica en forma de arbol que representa el fichero y que luego puede usarse para hacer el mapeo a clases de Java. La primera de las desventajas es clara, se ha de crear la imagen de los datos en memoria, por lo que si es muy grande puede ocupar mucho espacio, con una gran cantidad de pequeños elementos, cuyo control, recorrido y manipulacion requieren aun mas memoria y tiempo. Pero sus desventajas van mas alla, ya que se han de recorrer los datos en varias ocasiones, la primera para cargar la imagen en memoria y despues varias veces para sacar la infomacion necesaria para nuestra aplicacion.

SAX no trabaja asi. SAX extrae la informacion para nuestra aplicacion segun procesa el fichero, sin crear ninguna estructura adicional en memoria mas que la extrictamente necesaria para la lectura del fichero. SAX trabaja respondiendo a eventos, eventos que se producen al procesar el fichero XML, como pueden ser el fin o principio del documento, el fin o principio de un elemento, etc. Al hacer todo el proceso en una sola pasada obtenemos un mejor rendimiento.

Por supuesto no quiero decir que el API DOM no sirva para nada, nada mas lejos de la realidad, solo digo que en mi opinion no es el mas apropiado para realizar el mapeo entre documentos XML y Java. Si tu aplicación requiere de algun tipo de manipulacion del documento XML, como por ejemplo para combinar varios o aplicar una hoja de estilo, entonces DOM (o el mas moderno JDOM) es tu API, SAX no esta pensado para eso. Sax no vale para eso.

Un parser.

Bueno, lo primero de todo es conseguir un parser real, uno que implementa SAX (normalmente los parser de XML pueden trabajar con SAX y DOM). Yo os dire el parser con el que trabajo, libre, gratis y bueno, pero vosotros podeis trabajar con otro si quereis, cambiaran algunas clases, las especificas del parser, pero en general será el mismo esquema.

El parser que yo utilizo es el Xerces para Java de Apache, podeis bajarlo desde su pagina[1], alli podeis conseguir el parser (para varios lenguajes de programacion) y otras herramientas Java para trabajar con XML.

El interface ContentHandler.

Este interface es el centro de todo proceso de XML con SAX. Es el que define todos los eventos que se producen a los largo del proceso del documentno, asi que como habreis adivinado, vuestras clases de proceso de XML deberan implementar este interface para que el parser haga lo que vosotros necesitais.

Antes de ver todos los metodos que tiene este interface, puede ocurrir que vosotros no los necesiteis todos, quizas solo necesitais unos pocos, asi que como siempre, para ahorrar esfuerzo, existe una clase que ya implementa este interface, como siempre, pero con los metodos vacios para que solo tengais que extenderla y sobreescribir los que os hagan falta. Esta clase se llama **DefaultHandler**. Los metodos del interface ContentHandler los veremos en el proximo articulo, asi como otras cosas mas avanzadas, para evitar que este se haga demasiado grande.

Como hemos dicho, el API SAX se basa en eventos, eventos que se producen al procesar el fichero, del estilo de inicio/fin del documento, inicio/fin de un elemento, datos entre tags, etc, asi pues los metodos de ContentHandler son del estilo `startDocument`, `endDocument`, `startElement`, `endElement`, `characters`, etc, que se ejecutaran cuando ocurre dicho suceso.

Veamos un ejemplo básico.

El ejemplo.

Supongamos que tenemos el siguiente sencillo documento XML para almacenar nuestras paginas web favoritas:

```
<?xml version="1.0"?>
<favoritos>
  <pagina valoracion="10" >
    <nombre> javaHispano lt;/nombre>
    <direccion> http://www.javahispano.org lt;/direccion>
  </pagina>
  <pagina valoracion="1" >
    <nombre> GFT Techonologies </nombre>
    <direccion> http://www.gft.com </direccion>
  </pagina>
</favoritos>
```

Obviamente necesitaremos una clase que albergue los datos de nuestros favoritos:

```
public class Pagina
{
    private String nombre;
    private String direccion;
    private int valoracion;

    public Pagina ()
    {
    }

    public void setNombre (String nombre)
    {
        this.nombre = nombre;
    }

    public String getNombre()
    {
        return nombre;
    }

    // y demas metodos similares
    . . .
}
```

Suponiendo que queramos guardar todos los favoritos leído en un "Vector" tendremos que sobrescribir los metodos que nos interesan de la clase **DefaultHandler**.

En este caso seran :

1. El constructor, para que acepte el Vector donde almacenar los elementos que leemos.
2. El metodo startElement, para que detecte cuando empieza un nuevo "favorito".
3. El metodo endElement, para que detecte el final de los elemetros "nombre" y "direccion" para asi poder coger su valor.
4. El metodo characters, que recoge los valores contenidos entre las distintas etiquetas de inicio y fin de un elemento.

1. Métodos más importantes de ContentHandler

Ahora solo explicare los parametros y metodos estrictamente necesarios para el ejemplo, una descripcion mas detallada de todos los metodos y sus parametros de el interface ContentHanlder la daré en el siguiente articulo.

Veamos nuestro Handler:

```
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.Attributes;
import java.util.Vector;

public class MiXMLHandler extends DefaultHandler
{
    //vector de instancias
    private Vector instancias;
    //"Pagina" que se esta procesando
```

```

private Pagina actual;
//valor contenido entre las etiquetas de un elemento
private String valor;

public MixMLHandler (Vector v)
{
    instancias = v;
}

/*
    localName: contiene el nombre de la etiqueta.
    att: de la clase "org.xml.sax.Attributes", es una tabla que
contiene
        los atributos contenidos en la etiqueta.
*/
public void startElement( String namespaceURI, String localName,
String qName,
                        Attributes attr ) throws
SAXException
{
    //comprobamos si empezamos un elemento "pagina"
    if (localName.equals("pagina")){
        //creamos una nueva pagina y la añadimos al vector
        actual = new Pagina ();
        instancias.addElement (actual);
        //y le asignamos la "valoracion" contenida como atributo
        actual.setValoracion
(Integer.parseInt(attrs.getValue("valoracion")));
    }
}

public void endElement (String namespaceURI, String localName, String
rawName)
                        throws
SAXException
{
    /*
        miramos de que elemento se trata y asignamos los atributos
correspondientes a la "Pagina" actual segun el contenido del
elemento
        recogido en "valor" por el metodo characters.
    */
    if (localName.equals("nombre")){
        actual.setNombre (valor);
    }
    else if (localName.equals("direccion")){
        actual.setDireccion (valor);
    }
}

/*
    Los parametros que recibe es la localizacion de los caracteres del
elemento.
*/
public void characters (char[] ch, int start, int end) throws
SAXException
{
    //creamos un String con los caracteres del elemento y le quitamos
//los espacios en blanco que pueda tener en los extremos.
    valor = new String (ch, start, end);
    valor = valor.trim();
}
}

```

Y eso es todo, esa es la clase que procesara nuestro fichero XML, pero por supuesto

en algun sitio tendremos que decirle al parser que fichero procesar y que tiene "DataHandler" utilizar, ¿no?. Ahora vamos con ello.

```

import java.util.Vector;

import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;

import org.apache.xerces.parsers.SAXParser;

public class Test
{
    Vector instancias = new Vector ();

    public Test()
    {
    }

    public void procesarFichero ()
    {
        try
        {
            /*
solo es puede paracer un lio mezclar "SAXParser" y "XMLReader", pero
un problema de nomenclatura entre versiones. Se usa un
SAXParser,
que implementa el interface XMLReader (todos los parsers de XML
lo hacen), porque en la version 1 de SAX ya se uso el termino
"parser",
y se mantiene por convencion.
*/
XMLReader parser = new SAXParser();
//añadimos al parser nuestro "ContentHandler", pasandole el
vector de instancias.
parser.setContentHandler(new MiXMLHandler(instancias));
//y le decimos que empieze a procesar nuestro fichero de
favoritos
parser.parse("misFavoritos.xml");
        }
        catch (Exception e)
        {
            System.out.println ("Error al procesar el fichero de favoritos: "
+ e.getMessage());
            e.printStackTrace();
        }
    }

    public static void main(String[] args)
    {
        Test test = new Test();

        test.procesarFichero();
    }
}

```

Y eso es todo, sencillo, ¿no?. Ahora podemos usar ese vector de instancias para lo que queramos, como por ejemplo, asegurarnos de que nuestro parser trabaja bien ;-).

El proximo capitulo.

En el proximo articulo explicare mas en detalle el interface ContentHandler, y si tengo

tiempo explicare que hacer cuando tenemos documentos XML "recursivos" en los que aparece la misma etiqueta en elementos anidados, como por ejemplo ocurre con **nombre** en el siguiente documento:

```
<?xml version="1.0"?>
<clientes>
  <empresa>
    <nombre> GFT Technologies </nombre>
    .
    .
    <persona-contacto>
      <nombre> Alberto Molpeceres </nombre>
      .
      .
    </persona-contacto>
  </empresa>
</clientes>
```

Conclusión

Como hemos visto, el uso de SAX para procesar archivos XML no es ni mucho menos difícil. El API SAX permite de forma sencilla y rápida obtener objetos Java a partir del XML

Recursos y referencias

[1], <http://xml.apache.org/xerces-j/>

Acerca del autor

Alberto es ahora mismo desarrollador de aplicaciones en ámbito cliente/servidor para la empresa T-Systems - debis Systemhaus en Munich (Alemania). Cuando no está trabajando o "metiendo caña" al resto de los integrantes de javaHispano intenta pasear con su novia, buscar la desaparecida lógica del idioma alemán o intentar olvidar la pesadilla que es buscar piso en Munich.

Copyright (c) 2002, Alberto Molpeceres. Este documento puede ser distribuido solo bajo los términos y condiciones de la licencia de Documentación de javaHispano v1.0 o posterior (la última versión se encuentra en <http://www.javahispano.org/licencias/>).