

Manejo de ficheros XML en Java

Introducción

- ▶ XML de *eXtensible Markup Language* ('lenguaje de marcas extensible') es un metalenguaje, es decir, permite definir otros lenguajes de marcas. Se utiliza para representar información de forma jerarquizada.
- ▶ Se usa para dar soporte a bases de datos y permitir el intercambio de información entre distintas aplicaciones.
- ▶ Otra de sus aplicaciones es el intercambio de datos por Internet y en especial ha constituido la base de los servicios web.
- ▶ Son también útiles como ficheros de configuración de múltiples aplicaciones y como soporte para diversos protocolos de comunicación.
- ▶ El contenido de los ficheros XML es texto, pero organizado de forma jerarquizada mediante el uso de etiquetas o marcas delimitadas por los símbolos < y >.



Ejemplo de documento XML

```
▶ <?xml version="1.0" encoding="UTF-8"?>
▶ <project xmlns="http://www.netbeans.org/ns/project/1">
▶   <type>org.netbeans.modules.java.j2seproject</type>
▶   <configuration>
▶     <data xmlns="http://www.netbeans.org/ns/j2se-project/3">
▶       <name>EscribirFichAleatorio</name>
▶       <source-roots>
▶         <root id="src.dir"/>
▶       </source-roots>
▶       <test-roots>
▶         <root id="test.src.dir"/>
▶       </test-roots>
▶     </data>
▶   </configuration>
▶ </project>
```



Procesamiento de ficheros XML

- ▶ Estos ficheros se podrían manejar como si se trataran de ficheros de texto normales, pero la complejidad que se puede llegar a conseguir con la jerarquización ha hecho que se desarrollen una serie de bibliotecas para aumentar el rendimiento en su tratamiento.
- ▶ Para leer los documentos XML y comprobar si son válidos sintácticamente se utilizan los analizadores sintácticos o parsers. Son módulos, bibliotecas o programas encargado de transformar el fichero de texto en un modelo interno que optimiza su acceso.
- ▶ Existe un gran número de parsers pero dos de los modelos más conocidos son DOM y SAX, que tienen implementaciones para la mayoría de los lenguajes de programación.



Procesamiento de ficheros XML

- ▶ **DOM (*Document Object Model*)**: es una interfaz de programación que permite analizar y manipular dinámicamente y de manera global el contenido, el estilo y la estructura de un documento. Primero se almacena en memoria en forma de árbol con nodos padre, nodos hijo y nodos finales que son aquellos que no tienen descendientes.
- ▶ En este modelo todas las estructuras de datos del documento se transforman en algún tipo de nodo, y luego esos nodos se organizan jerárquicamente en forma de árbol.
- ▶ Este procesamiento necesita recursos de memoria y tiempo de procesamiento a medida que los ficheros aumentan en tamaño y jerarquización.



Procesamiento de ficheros XML

- ▶ **SAX (*Simple API for XML*)**: es una interfaz de programación que procesa los ficheros XML de forma secuencial, leyéndolos una única vez y procesándolos a medida que se van leyendo.
- ▶ No carga el documento en memoria, sino que lo lee directamente desde el fichero, por lo que es especialmente útil cuando el fichero XML es muy grande.
- ▶ Apenas consume recursos de memoria, pero impide tener una visión global del documento por el que navegar.



Acceso a ficheros XML con DOM

- ▶ Se necesitan las clases e interfaces de los paquetes `org.w3c.dom` y `javax.xml.parsers`
- ▶ Además para generar ficheros XML a partir de un árbol DOM, se utilizará el paquete `javax.xml.transform`
- ▶ Algunas de las interfaces que se utilizan son:
 - ▶ **Document**. Es un objeto que representa un documento XML.
 - ▶ **Element**. Cada uno de los objetos que representa cada elemento del documento XML. Tiene propiedades y métodos para manipular los elementos del documento.
 - ▶ **Node**. Representa a cualquier nodo del documento.
 - ▶ **NodeList**. Contiene una lista de los nodos hijos de un nodo.
 - ▶ **Attr**. Permite acceder a los atributos de un nodo.
 - ▶ **Text**. Son los datos carácter de un elemento.
 - ▶ **CharacterData**. Representa a los datos carácter de un documento, con atributos y métodos para manipularlos.
 - ▶ **DocumentType**. Proporciona información de la etiqueta `<!DOCTYPE>`



Acceso a ficheros XML con DOM: escritura

- ▶ Para empezar a crear un fichero XML se suele partir de los datos almacenados en algún fichero, o en alguna colección de datos.

- ▶ Lo primero es construir el parser:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
try {  
    DocumentBuilder builder = factory.newDocumentBuilder();
```

- ▶ Lo siguiente es crear un documento XML vacío:

```
DOMImplementation implementation = builder.getDOMImplementation();  
Document document = implementation.createDocument(null, "Empleados",  
null);  
document.setXmlVersion("1.0");
```



Acceso a ficheros XML con DOM: escritura

- ▶ Ahora hay que ir creando los elementos anidados. Por cada registro, u objeto, o elemento similar, se crea un nodo, por ejemplo un empleado, y para cada empleado sus atributos:

```
Element raiz = document.createElement("empleado");  
document.getDocumentElement().appendChild(raiz);
```

//se crea cada atributo del nodo

```
Element id = document.createElement("id");  
Text textid = document.createTextNode(Integer.toString(emp.getId()));  
raiz.appendChild(id); //se une el elemento con su padre  
id.appendChild(textid); // se añade el valor del elemento
```

```
Element nombre = document.createElement("nombre");  
Text textnombre = document.createTextNode(emp.getNombre());  
raiz.appendChild(nombre);  
nombre.appendChild(textnombre);
```

□ ...



Acceso a ficheros XML con DOM: escritura

- Por último se genera el fichero XML, a partir del árbol DOM creado, utilizando transform, que permite especificar una fuente y un destino:

//se crea la fuente XML partir del documento

```
Source source = new DOMSource(document);
```

//se crea el fichero de texto Empleados.xml

```
Result result = new StreamResult(new java.io.File("Empleados.xml"));
```

// se crea un TransformerFactory

```
Transformer transformer = TransformerFactory.newInstance().newTransformer();
```

//se transforma el árbol al fichero

```
transformer.transform(source, result);
```

//se visualiza el documento por pantalla

```
Result console= new StreamResult(System.out);
```

```
transformer.transform(source, console);
```



Acceso a ficheros XML con DOM: lectura

- ▶ Para leer un fichero XML se crea una instancia de *DocumentBuilderFactory* para construir el parser
- ▶ Se crea una instancia del documento, a partir de los datos del fichero XML, mediante el parser

```
public void leerDOM() throws ParseException {  
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
    try {  
        //Pasamos un fichero XML a un árbol DOM usando el DocumentBuilder  
        DocumentBuilder builder = factory.newDocumentBuilder();  
  
        //Se crea un Java DOM XML parser(analizador sintáctico)  
        Document document = builder.parse("empleados.xml");  
        document.getDocumentElement().normalize();  
  
        //Se visualiza el nombre del elemento raíz  
        System.out.println("Elemento raíz:" +  
            document.getDocumentElement().getNodeName());  
    }  
}
```



Acceso a ficheros XML con DOM: lectura

- ▶ A partir de este momento se pueden cargar todos los elementos de un tipo determinado en una lista de nodos para después procesarlos. Por ejemplo:

```
NodeList listaempleados =  
document.getElementsByTagName("empleado");
```

- ▶ Podemos ir sacando los elementos de esa lista y hacer el tipo de tratamiento que necesitemos, por ejemplo visualizarlos por pantalla:



Acceso a ficheros XML con DOM: lectura

```
for (int i = 0; i < listaempleados.getLength(); i++) {  
    Node empleado = listaempleados.item(i); //obtiene un nodo  
    if (empleado.getNodeType() == Node.ELEMENT_NODE) {  
        Element elemento = (Element) empleado;  
        System.out.println("Id: "+getNodo("id", elemento));  
        System.out.println("Nombre: "+ getNodo("nombre", elemento));  
        System.out.println("Apellido: "+ getNodo("apellido", elemento);  
        System.out.println("Apellido2: "+ getNodo("apellido2", elemento);  
        System.out.println("Salario: "+ getNodo("salario", elemento);  
    }  
}
```

- Podemos ir sacando los elementos de esa lista y hacer el tipo de tratamiento que necesitemos, por ejemplo visualizarlos por pantalla:



Acceso a ficheros XML con DOM: lectura

- Para obtener la información de un nodo hemos creado el método:

```
private static String getNodo(String tag, Element elem) {  
    NodeList nodo =  
    elem.getElementsByTagName(tag).item(0).getChildNodes();  
    Node valornodo = (Node) nodo.item(0);  
    return valornodo.getNodeValue();  
}
```



Acceso a ficheros XML con SAX: lectura

- ▶ La lectura de un fichero XML produce una serie de eventos relacionados con el inicio y la finalización de cada una de las etiquetas, o con la aparición de caracteres entre las etiquetas, por ejemplo `startDocument()`, `endDocument()`, `startElement()`, `endElement()`, `characters()`.

- ▶ Estos eventos son tratados mediante la llamada a métodos.

- ▶ Se comienza creando una factoría para el procesador de XML.

```
SAXParserFactory parserFactory = SAXParserFactory.newInstance();
```

```
SAXParser parser = parserFactory.newSAXParser();
```

```
XMLReader reader = parser.getXMLReader();
```

- ▶ A este *reader* habrá que pasarle el manejador que se encargue de responder a los eventos

```
reader.setContentHandler(new GestionContenido());
```

Acceso a ficheros XML con SAX: lectura

- ▶ Para poder utilizar los métodos que responderán a los eventos, es necesario implementar alguna de las siguientes interfaces:
 - ▶ **ContentHandler**: recibe las notificaciones de los eventos que ocurren en el documento
 - ▶ **DTDHandler**: recoge eventos relacionados con el DTD (*Document Type Definition*)
 - ▶ **ErrorHandler**: define métodos de tratamiento de errores
 - ▶ **EntityResolver**: sus métodos se llaman cuando se encuentra una referencia a una entidad
 - ▶ **DefaultHandler**: proporciona una implementación por defecto para todos los métodos y será necesario desarrollar los que se vayan a utilizar



Acceso a ficheros XML con SAX: lectura

- ▶ Vamos a utilizar **DefaultHandler** e implementar los siguientes métodos:
 - ▶ **startDocument**: se produce al comenzar el procesamiento del documento XML
 - ▶ **endDocument**: se produce al finalizar el procesamiento
 - ▶ **startElement**: se produce al comenzar una etiqueta XML (aquí se leen los atributos de las etiquetas)
 - ▶ **endElement**: al finalizar la etiqueta
 - ▶ **characters**: se produce al encontrar una cadena de texto
- ▶ Por último es necesario pasarle al parser el documento para que se haga el procesamiento:

```
reader.parse(new InputSource(new  
FileInputStream("empleados.xml")));
```

Acceso a ficheros XML con SAX: lectura

- El código de la clase que extiende la interfaz DefaultHandler es:

```
class GestionContenido extends DefaultHandler{

    @Override
    public void startDocument() throws SAXException {
        System.out.println("\nPrincipio del documento XML");  }

    @Override
    public void endDocument() throws SAXException {
        System.out.println("\nFin del documento XML");  }

    @Override
    public void startElement(String uri, String localName, String name,
        Attributes attributes) throws SAXException {
        System.out.print("\nEtiqueta -> "+name);
        //Recorremos los atributos
        System.out.println("\t"+attributes.getLength()+" atributos: ");
        for(int i=0;i<attributes.getLength();i++){
            System.out.print("\t"+attributes.getQName(i)+": "+attributes.getValue(i));
        }
    }

    @Override
    public void characters(char[] ch, int start, int length)
        throws SAXException {
        System.out.println("\tTexto: "+String.valueOf(ch, start, length));  }

    @Override
    public void endElement(String uri, String localName, String name)
        throws SAXException {
        System.out.println("\nfin "+name);  }
```

Acceso a ficheros XML con SAX: escritura

- ▶ Se necesita una clase que derive de **XMLReader** y que implemente una versión del método parser adaptado a nuestros datos.
- ▶ También se necesita un objeto de la clase **InputSource** a la que se le pasará la colección con nuestros datos.
- ▶ Se creará un objeto de tipo Source, al que se le pasarán los dos objetos anteriores, y será el punto de partida de la transformación:

```
XMLReader datosReader = new DatosReader();
```

```
InputSource datosSource = new Alm_datos(50);
```

```
// La clase que contenga los datos deberá extends InputSource
```

```
Source source = new SAXSource(datosReader, datosSource);
```



Acceso a ficheros XML con SAX: escritura

- ▶ Lo siguiente será preparar la salida, con los datos del fichero de salida.

```
File f = new File("empleadosSAX.xml");  
Result resultado = new StreamResult(f);
```

- ▶ Se crea la instancia y se configuran los parámetros del tipo de fichero que se quiere obtener:

```
Transformer transformer =  
TransformerFactory.newInstance().newTransformer();  
transformer.setOutputProperty(OutputKeys.INDENT, "yes");  
transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
```

- ▶ Por último se produce la transformación:

```
transformer.transform(source, resultado);
```



Acceso a ficheros XML con SAX: escritura

```
public class DatosReader implements XMLReader {  
    ..... private ContentHandler handler;  
    ..... private final AttributesImpl atts = new AttributesImpl();  
    ..... //métodos que serán usados por el transformer  
    @Override  
    public void parse(InputSource input) throws IOException, SAXException {  
        if (!(input instanceof Alm_datos)) {  
            String m = "El único parámetro para el parser es un Alm_datos";  
            throw new SAXException(m);  
        }  
        if (handler == null) {  
            throw new SAXException("Sin ContentHandler");  
        }  
        Alm_datos source = (Alm_datos) input;  
        List<Empleado> empleados = source.getDatos();  
        handler.startDocument();  
        handler.startElement("", "empleados", "empleados", atts);  
        for (Empleado empl : empleados) { //El id, se añade como atributo para ver el uso de atributos, podría ser un elemento más  
            atts.addAttribute("", "id", "id", "", String.valueOf(empl.getId()));  
            handler.startElement("", "empleado", "empleado", atts);  
            atts.clear();  
            handler.startElement("", "nombre", "nombre", atts);  
            char[] nombre = empl.getNombre().toCharArray();  
            handler.characters(nombre, 0, nombre.length);  
            .....  
            handler.endElement("", "nombre", "nombre");  
        }  
    }  
}
```

Acceso a ficheros XML con SAX: escritura

//Otros métodos que hay que implementar en DatosReader

.....
@Override

```
public ContentHandler getContentHandler() {  
    return handler; }  
.....
```

@Override

```
public void setContentHandler(ContentHandler handler) {  
    this.handler = handler; }  
.....
```

@Override

```
public void parse(String systemId) { }  
.....
```

@Override

```
public boolean getFeature(String name) {  
    return false; }  
.....
```

@Override

```
public void setFeature(String name, boolean value) { }  
.....
```

@Override

```
public Object getProperty(String name) {  
    return null;  
}  
.....
```

@Override

```
public void setProperty(String name, Object value) { }  
.....
```



Acceso a ficheros XML con SAX: escritura

//Otros métodos que hay que implementar en DatosReader

@Override

```
public void setEntityResolver(EntityResolver resolver) { }
```

@Override

```
public EntityResolver getEntityResolver() {  
    return null; }
```

@Override

```
public void setDTDHandler(DTDHandler handler) { }
```

@Override

```
public DTDHandler getDTDHandler() {  
    return null; }
```

@Override

```
public void setErrorHandler(ErrorHandler handler) { }
```

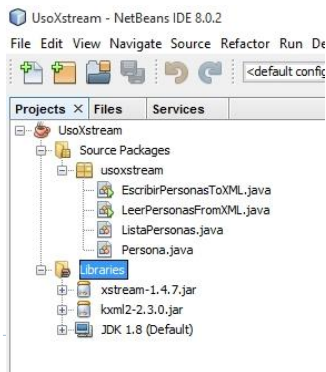
@Override

```
public ErrorHandler getErrorHandler() {  
    return null;  
}
```



Serialización de objetos a XML con XStream

- ▶ Se utilizará la biblioteca XStream, que es necesario descargar de: <http://x-stream.github.io/>
- ▶ En el apartado de descargas se irá a la última versión estable y se elegirá *Binary Distribution*, se trata de un fichero zip comprimido con varios ficheros .jar, se necesita *xstream-1.4.x.jar*
- ▶ También se necesitará el parser *kxml2*, del apartado *Optional Dependencies*
- ▶ Estos ficheros se añaden al apartado *Libraries* del proyecto en el que queramos usarlos.



Serialización de objetos a XML con XStream

- ▶ Vamos a serializar una colección de elementos. Tendremos por una parte una clase con el elemento base, por ejemplo persona, o empleado o factura. Por otra parte una segunda clase que contiene una colección de elementos del tipo base, por ejemplo ListaPersonas.
- ▶ Para utilizar XStream lo primero es crear una instancia de la clase:
`XStream xstream= new XStream();`
- ▶ En principio las etiquetas xml se corresponden con los atributos de las clases a serializar, tendríamos una etiqueta ListaPersonas y dentro de ella tendríamos un conjunto de etiquetas persona. Si quisiéramos cambiarlas en el fichero de salida, lo haríamos así:
`xstream.alias("ListaPersonasXS", ListaPersonas.class);`
`xstream.alias("PersonasXS", Persona.class);`



Serialización de objetos a XML con XStream

- ▶ Si quisiéramos que no apareciera el atributo lista de la clase ListaPersonas utilizaríamos:

```
xstream.addImplicitCollection(ListaPersonas.class, "lista");
```

- ▶ Para generar el fichero a partir de la lista de objetos se utilizaría:

```
xstream.toXML(listaper, new  
FileOutputStream("PersonasXS.xml");
```

siendo listaper un objeto con datos, del tipo ListaPersonas.



Deserialización de un fichero XML con XStream

- ▶ Se declara el objeto XStream:
`XStream xstream= new XStream();`
- ▶ Si se han declarado alias al escribir habría que volverlos a declarar:
`xstream.alias("ListaPersonasXS", ListaPersonas.class);`
`xstream.alias("PersonasXS", Persona.class);`
- ▶ Lo mismo, si hubiéramos omitido algún atributo:
`xstream.addImplicitCollection(ListaPersonas.class, "lista");`
- ▶ En las últimas versiones, se ha añadido un marco de seguridad, y en ocasiones puede aparecer una excepción del tipo *com.thoughtworks.xstream.security.ForbiddenClassException*
- ▶ haciendo referencia a alguna de las clases básicas que intervienen en la deserialización, por ejemplo Persona



Deserialización de un fichero XML con XStream

- ▶ En estos casos deberíamos permitir de forma explícita deserializar ese tipo, se puede hacer de dos formas:

```
xstream.allowTypes(new String[]{"entidades.Persona"});
```

- ▶ o bien:

```
xstream.allowTypes(new Class[]{"entidades.Persona.class"});
```

- ▶ Por último recuperamos el fichero en una colección:

```
ListaPersonas listaper= (ListaPersonas) xstream.fromXML(new  
FileInputStream("PersonasXS.xml"));
```



Utilización de hojas de estilo con XML

- ▶ Una hoja de estilos XSL(*eXtensible Stylesheet Language*) describe el proceso de presentación mediante elementos XML.
- ▶ Partiendo de un fichero con datos XML y de una hoja de estilos XSL, se puede generar un fichero HTML, que visualice los datos del fichero XML con formato.



Utilización de hojas de estilo con XML

► Fichero alumnos.xml

```
<?xml version="1.0"?>
<listadealumnos>
  <alumno>
    <nombre>Juan</nombre>
    <edad>19</edad>
  </alumno>
  <alumno>
    <nombre>Maria</nombre>
    <edad>20</edad>
  </alumno>
</listadealumnos>
```



Utilización de hojas de estilo con XML

► Fichero alumnosplantilla.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
```

```
<html><xsl:apply-templates /></html>
```

```
</xsl:template>
```

```
<xsl:template match="listadealumnos">
```

```
<head><title>LISTADO DE ALUMNOS</title></head>
```

```
<body>
```

```
<h1>LISTA DE ALUMNOS</h1>
```

```
<table border="1">
```

```
<tr><th>Nombre</th><th>Edad</th></tr>
```

```
<xsl:apply-templates select="alumno" />
```

```
</table>
```

```
</body>
```

```
</xsl:template>
```

```
<xsl:template match="alumno">
```

```
<tr><xsl:apply-templates /></tr>
```

```
</xsl:template>
```

```
<xsl:template match="nombre|edad">
```

```
<td><xsl:apply-templates /></td>
```

```
</xsl:template>
```

```
</xsl:stylesheet>
```

Utilización de hojas de estilo con XML

- Instrucciones Java necesarias:

```
File pagHTML = new File("pagina.html");  
FileOutputStream fos = new FileOutputStream(pagHTML);  
Source estilos = new StreamSource("alumnosPlantilla.xsl");  
Source datos = new StreamSource("alumnos.xml");  
Result result = new StreamResult(fos);  
Transformer transformer=  
TransformerFactory.newInstance().newTransformer(estilos);  
transformer.transform(datos, result);  
fos.close();
```

- Habría que añadir el tratamiento de excepciones



Utilización de hojas de estilo con XML

```
<head><META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>LISTADO DE ALUMNOS</title>
</head>
<body>
<h1>LISTA DE ALUMNOS</h1>
<table border="1">
<tr>
<th>Nombre</th><th>Edad</th>
</tr>
<tr>
<td>Juan</td>
<td>19</td>
</tr>
<tr>
<td>Maria</td>
<td>20</td>
</tr>
</table>
</body>
</html>
```



LISTA DE ALUMNOS

| Nombre | Edad |
|--------|------|
| Juan | 19 |
| Maria | 20 |

