Unidad : Acceso a Datos con Ficheros XML

Índice

Unidad 2: Acceso a Datos con Ficheros XML....1
1. ¿Qué es XML, dónde y cuándo utilizar XML en el diseño de un sistema?....2
2. Manejo de ficheros XML....3
3. JAXP: Java API for XML Processing....3
4. Acceso a datos con JAXP DOM....4
4.1. Operaciones sobre nodos DOM XML en Java....4
4.2. Operaciones sobre documentos DOM XML en Java....6
5. Acceso a datos con JAXP SAX....9
6. Acceso a datos con JAXB....10
6.1. Adaptadores....10
6.2. Anotaciones....10

1. ¿Qué es XML, dónde y cuándo utilizar XML en el diseño de un sistema?

XML (eXtensible Markup Language)

- Estándar industrial para representar datos, independiente del sistema.
- Es un lenguaje usado para el transporte de datos. También es un lenguaje llamado “de etiquetas”, es decir, cada paquete de información está delimitado por dos etiquetas como se hace también en el lenguaje HTML, pero XML separa el contenido de la presentación.

- XML ofrece una multiplataforma, capacidades multilenguaje para datos, así como Java ofrece soporte multiplataforma para lógica de aplicaciones.

XML podría tener un papel en múltiples áreas:

• **Navegador**
Es posible crear páginas web utilizando contenido XML y hojas de estilo XSL relacionadas. XSLT soporta esta posibilidad así como la conversión a muchos formatos distintos.

• **Pedido del cliente**
Un objeto XMLHttpRequest se encuentra en el corazón de Ajax.

• **Respuesta del servidor**
Cuando un objeto XMLHttpRequest regresa, los contenidos de la respuesta pueden venir en XML. Pero incluso si no lo están, el navegador utilizará el DOM para manipular la página web.

• **Servicios web**
SOAP es un protocolo basado en XML para intercambiar información a través de HTTP (en otras palabras, a través de la Web). Su función principal es solicitar servicios web de manera remota. Es el sucesor a la llamada a procedimiento remoto (RPC) XML.

• **Servicio de mensajería Java (JMS)**
JMS se utiliza para enviar mensajes entre los procesos de manera asincrónica. El contenido XML de los mensajes provee una vía de comunicación, para que todas las partes puedan comprender sin importar qué lenguaje utilicen ni la plataforma sobre la que ejecuten.

• **Presentación de informes**
Además de su utilidad para representar contenidos de páginas web, también es posible utilizar XSLT para producir informes en formatos múltiples.

• **Bases de datos**
Tanto IBM como Oracle (entre otras) han creado bases de datos XML nativas que almacenan estructuras de documentos XML y que soportan XQuery.

2. Manejo de ficheros XML

- Desde el punto de vista de “bajo nivel”, un documento XML no es otra cosa que un fichero de texto.
- Desde el punto de vista de “alto nivel”, no se considera un documento de texto y debe manipularse de la manera adecuada.

Las herramientas que leen y comprueban el contenido de los documentos XML se denominan Parsers o analizadores sintácticos.

Es posible elegir uno de cuatro modelos de procesamiento XML disponibles a través de las siguientes API:

1. SAX: Provee un modelo de programación basado en eventos.
2. DOM: Provee un modelo de programación de recorrido de árbol.

Estos dos modelos están disponibles a través de **Java API for XML Processing (JAXP).** Gracias al apoyo de W3C aparece la interfaz JAXP (da acceso a diferentes parsers).

3. Enlace de datos XML a través de la tecnología **JAXB**.
4. XSLT: Brinda un modelo de programación basado en plantillas.

3. JAXP: Java API for XML Processing

Provee la posibilidad de validar, parsear, transformar y hacer querys sobre documentos XML.
Las tres interfaces básicas de parseo son:

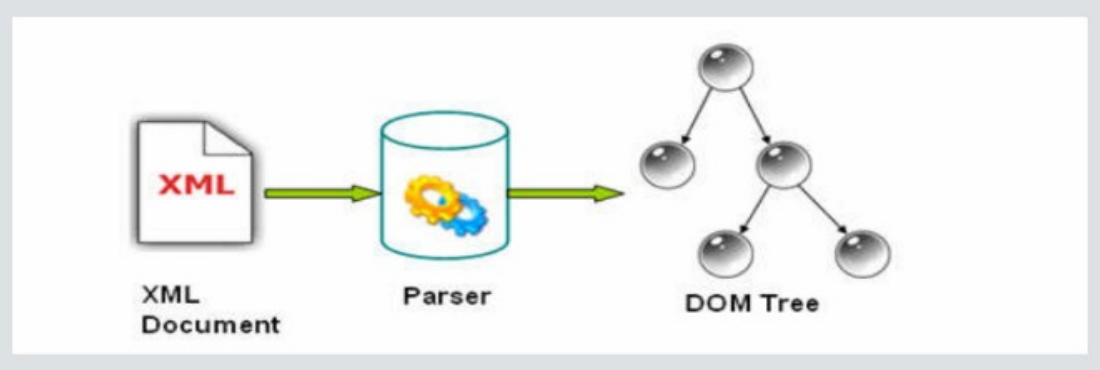
- DOM: Document Object Model parsing interface.
- SAX: Simple API for XML parsing interface
- StAX: Streaming API for XML (parte de JDK 6)

Además, provee XSLT interface → transformaciones.

4. Acceso a datos con JAXP DOM

Realiza dos pasos para transformar un documento XML:

- Se almacena en memoria en forma de árbol.
- Los métodos DOM permiten recorrer y analizar los diferentes nodos del árbol.
- DOM ofrece acceso a XML (lectura y escritura de datos), aunque cuenta con el inconveniente de que el "árbol" se crea en la memoria principal.



Parsea un documento cargándolo en memoria por completo.

- Todo en memoria → es posible acceder o modificar cualquier elemento en acceso aleatorio.
- El parser DOM se llama DocumentBuilder.
- Se carga el XML en una estructura con forma de árbol.
- No recomendado para grandes documentos por el consumo de memoria.

4.1. Operaciones sobre nodos DOM XML en Java

Nodos en el modelo DOM de XML

En el modelo DOM para XML, se considera un nodo a cualquier cosa que se encuentra dentro del documento:

- Todo el documento en su conjunto es un nodo documento.
- Cada elemento XML es un nodo elemento.
- El texto de los elementos XML son nodos texto.
- Cada atributo es un nodo atributo.
- Los comentarios son nodos comentario.

Para manipular los nodos en Java se dispone de la clase ***Node***, que se encuentra en el paquete org.w3c.dom al igual que las clases de los distintos tipos de nodos que se acaban de comentar. Los nombres de estas clases coinciden con las de otras clases de otros paquetes de Java, por lo que puede ser recomendable importar todas las clases de ese paquete para evitar confusiones:

import org.w3c.dom.*

Métodos principales para navegación por los nodos

Node **getFirstChild**()
Retorna el primer nodo hijo de este nodo. Si no lo hay, retorna *null*.

Node **getNextSibling**()
Retorna el nodo inmediatamente siguiente (hermano) a este nodo. Si no lo hay, retorna *null*.

Node **getParentNode**()
Retorna el **padre** de este nodo. Todos los nodos, excepto los de tipo *Attr, Document, DocumentFragment, Entity,* y *Notation* pueden tener un padre. Sin embargo, será *null* si el nodo está recién creado y no ha sido añadido todavía al árbol, o si ha sido eliminado del árbol.
NodeList **getChildNodes**()
Retorna una lista de la clase *NodeList* que contiene todos los **nodos hijo** de este nodo. Si no hay nodos hijo, la lista estará vacía.
La **clase NodeList** dispone de los siguientes métodos que permiten obtener los nodos que componen una de esas listas:

int **getLength**()
Retorna el **número de nodos** de la lista.
Node **item**(int index)
Retorna el **nodo** de la lista que ocupa la posición *index*.

Métodos principales para gestionar los nodos

Node **appendChild**(Node hijoNuevo)

Añade el nodo *hijoNuevo* al final de la lista de hijos de este nodo. Si el *hijoNuevo* ya está en el árbol, se elimina previamente. Retorna el nodo que se ha añadido.

Node **insertBefore**(Node hijoNuevo, Node hijoReferencia)

Inserta el nodo *hijoNuevo* antes del nodo *hijoReferencia* que ya existe. Si el nodo *hijoReferencia* es *null*, se inserta el nodo *hijoNuevo* al final de la lista de nodos hijo. Retorna el nodo que se ha insertado.

Node **removeChild**(Node hijoViejo)

Elimina, de la lista de hijos, el hijo indicado por parámetro como *hijoViejo* y lo retorna.

Node **replaceChild**(Node hijoNuevo, Node hijoViejo)

Sustituye el nodo *hijoViejo* que se encuentre en la lista de nodos hijo por el nodo *hijoNuevo*.

Métodos principales para manipular la información del nodo

short **getNodeType**()

Retorna un código identificativo del **tipo de nodo** que se trata.

String **getNodeName**()

Retorna el **nombre** de nodos de determinados tipos, como los atributos (*Attr*) y elementos (*Element*).

String **getNodeValue**()

Retorna el **valor** de nodos de determinados tipos, como los atributos (*Attr*), comentarios (*Comment*) y textos (*Text*).

String **getTextContent**()

Retorna el **texto** del nodo y de sus descendientes.

void **setNodeValue**(String valorNodo)

Establece el valor de nodos de determinados tipos, como los atributos (*Attr*), comentarios (*Comment*) y textos (*Text*).

4.2. Operaciones sobre documentos DOM XML en Java

Abrir un fichero XML

Siendo la variable fichero la que hace referencia al fichero XML (como objeto File, String o InputStream), con estas instrucciones se obtiene en memoria el contenido completo del fichero XML:

```
DocumentBuilderFactory fabricaCreadorDocumento = DocumentBuilderFactory.newInstance();
DocumentBuilder creadorDocumento = fabricaCreadorDocumento.newDocumentBuilder();
Document documento = creadorDocumento.parse(fichero);
```

Si se desea obtener el contenido de un documento XML desde Internet, en lugar de la referencia al fichero que se hace al final, se debe usar algo como:

```
URL dir = new URL("http://direccionDelFicheroXML");
```

```
Document documento = creadorDocumento.parse(dir.openStream());
```

A partir de ese momento, todas las **acciones sobre el documento XML** se deberán realizar sobre la **variable** *documento* de la clase Document de Java, que permite la gestión de documentos XML basándose en el modelo DOM.
Las clases *DocumentBuilderFactory* y *DocumentBuilder* se encuentran en el paquete **javax.xml.parsers** de Java, y la clase *Document* es del paquete **org.w3c.dom**, por lo que se requiere importar dichos paquetes para usar estas clases.

Mostrar o guardar el contenido de un documento XML

Para realizar cualquiera de esas operaciones, es necesario crear previamente un transformador al que se le indique el documento y el destino que se le va a dar.

```
TransformerFactory fabricaTransformador = TransformerFactory.newInstance();
Transformer transformador = fabricaTransformador.newTransformer();
Source origen = new DOMSource(documento);
```

El objeto transformador que se ha creado dispone de un **método transform** que permite realizar el volcado del documento que se encuentra en memoria o otro formato. Se deben indicar dos parámetros: el primero es el **documento de origen**, y el segundo puede ser un **objeto** *Result*, en el que se obtendrá el resultado de la transformación. Para obtener un objeto *Result* se puede crear un **objeto de la clase StreamResult** indicando como parámetro un objeto *File***,** *OutpuStream***,** *String* **o** *Writer*.
Por ejemplo, para obtener el **resultado en un fichero** llamado *salida.xml*:

```
Result destino = new StreamResult("salida.xml");
transformador.transform(origen, destino);
```

Si por el contrario, se desea obtener el **resultado en la salida estándar**:

```
Result destino = new StreamResult(System.out);
transformador.transform(origen, destino);
```

Para obtener el contenido **como** *String***, y mostrarlo en la salida estándar**:

```
StringWriter writer = new StringWriter();
Result destino = new StreamResult(writer);
transformador.transform(origen, destino);
System.out.println(writer.toString());
```

Crear un documento XML en blanco

Las **clases** *DocumentBuilderFactory* **y DocumentBuilder** permiten la generación de un documento XML vacío almacenándolo en memoria. Para ello se debe utilizar el **método** *newDocument()* de esa última clase:

```
DocumentBuilderFactory fabricaCreadorDocumento = DocumentBuilderFactory.newInstance();
DocumentBuilder creadorDocumento = fábricaCreadorDocumento.newDocumentBuilder();
Document documento = creadorDocumento.newDocument();
```

Si se muestra el **contenido del objeto documento** que se ha creado, se puede observar algo como lo siguiente:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Métodos principales de la clase *Document* para obtener información

Element **getDocumentElement**()

Obtiene el elemento raíz del documento.

NodeList **getElementsByTagName**(String etiqueta)

Retorna una lista *NodeList* con todos los elementos del documento que tienen la *etiqueta* indicada.

Ya que la clase *Document* es una subclase de la clase *Node*, además de estos métodos propios, se pueden utilizar los **métodos de la clase Node**

Métodos principales de la clase *Document* para crear nodos

Los siguientes métodos de la clase Document permiten la creación de cada uno de los posibles tipos de nodos admitidos por XML:

Attr **createAttribute**(String *name*)

CDATASection **createCDATASection**(String *data*)

Comment **createComment**(String *data*)

DocumentFragment **createDocumentFragment**()

Element **createElement**(String *tagName*)

EntityReference **createEntityReference**(String *name*)

ProcessingInstruction **createProcessingInstruction**(String *target*, String *data*)

Text **createTextNode**(String *data*)

5. Acceso a datos con JAXP SAX

Ofrece una alternativa para leer documentos XML de manera secuencial.
A diferencia de DOM, no carga el documento en memoria, sino que lo lee directamente desde el fichero.

Pasos:
- Se le dice al parser SAX el fichero que quiere que sea leído.
- El documento XML es traducido a eventos.
- Los eventos generados pueden controlarse con métodos Callbacks.
- Para implementar los callbacks basta con implementar la interfaz ContentHandler

- Procesa la información por eventos.
- Mejor que DOM para manipular archivos grandes, ya que no genera un árbol en memoria.
- Se recorre XML de manera secuencial.
- Permite acceso a datos pero no modificarlos.
- El parseo de SAX se puede detener en cualquier momento.

SAX es un parseador de acceso serie y basado en eventos. Es decir, va recorriendo el xml poco a poco (en serie) y va generando eventos conforme va encontrando determinadas partes del xml, como el principio de etiquieta, el fin de etiqueta, texto.

Una vez visto esto, como resumen podemos enumerar las ventajas e inconvenientes de usar SAX:

Ventajas
- No consume mucha memoria ya que va procesando el documento en serie

Inconvenientes
- Es poco intuitivo para el desarrollador.
- Requiere una serie de controles de estado adicionales que complican el procesado.
- No permite construir XML. Sólo permite parsearlos.
- No permite el acceso aleatorio al documento xml.

Como conclusión final, decir que sólo se puede justificar su uso desde el punto de vista del rendimiento al tratar XMLs grandes, ya que al acceder en serie al xml no necesita mantener

en memoria todo el XML o representación equivalente, como hace DOM. En el resto de los casos, es mejor usar alguna API basada en DOM.

6. Acceso a datos con JAXB

JAXB (no confundir con la interfaz de acceso JAXP) es una librería de (Un)-Marshalling.

- El concepto de **Serialización o Marshalling** que ya ha sido introducido, es el proceso de almacenar un conjunto de objetos en un fichero.
- **Unmarshaling** es justo el proceso contrario: convertir en objetos el contenido de un fichero.

JAXB es capaz de obtener de un esquema XML una estructura de clases que le da soporte en Java.
Básicamente, la parte más importante aquí es el uso de la clase `javax.xml.bind.JAXBContext`. Esta clase proporciona un marco para la validación, serialización de objetos Java a XML y es el punto de entrada a la API JAXB.

6.1. Adaptadores

Al manipular los tipos complejos que pueden no estar disponibles directamente en JAXB tenemos que escribir un adaptador para indicar JAXB cómo manejar el tipo específico.

6.2. Anotaciones

Anotaciones utilizadas en JAXB para XML Vamos a enumerar aquí las más importantes:

- `XmlAccessorOrder` : Esta anotación controla el orden de los campos y las propiedades de una clase en la que aparecerá en el XML.
- `XmlAccessorType` : indica si un elemento se debe serializar o no. Se utiliza en combinación con `javax.xml.bind.annotation .XMLAccessType` .
- `XmlAnyAttribute` : Asigna un elemento a un mapa de atributos de comodín.
- `XmlAttribute` : Esta anotación es uno de los básicos y más utilizados. Se asigna un elemento de Java (propiedad, un atributo de campo) a un atributo del nodo XML.
- `XmlElement` : Asigna un elemento de Java para un nodo XML usando el nombre.
- `XmlElementRef` : Asigna un elemento de Java para un nodo XML usando su tipo (diferente a la anterior, donde se utiliza el nombre de mapeo).
- `XmlElementRefs` : las marcas de una propiedad que se refiere a las clases con `XmlElement` o `JAXBElement` .
- `XmlElements` : Este es un contenedor de múltiples `XMLElement` anotaciones.
- `XmlElementWrapper` : Genera una envoltura alrededor de una representación XML, destinado a ser utilizado con las colecciones.

- `XmlEnum` : Proporciona mapeo para una enumeración de una representación XML. Funciona en combinación con `XmlEnumValue` .
- `XmlEnumValue` : asigna una constante para un elemento XML enumeración.
- `XmlID` : Mapea una propiedad con un identificador XML.

- `XmlMixed` : El elemento anotado puede contener contenido mixto. l
- `XmlRootElement` : Este es probablemente el más utilizado dentro de anotación JAXB. Se utiliza para asignar una clase a un elemento XML.
- `XmlSchema` : asigna un paquete a un espacio de nombres XML.
- `XmlSchemaType` : Asigna un tipo Java a un simple esquema de tipo incorporado.
- `XmlType` : Se utiliza para asignar una clase o una enumeración a un tipo en un esquema XML.
- `XmlValue` : Permite asignar una clase a un tipo complejo esquema XML con un simpleContent o un tipo simple esquema XML.