4.4. Utilización de herramientas de procesamiento: DOM y SAX

Índice

Objetivos

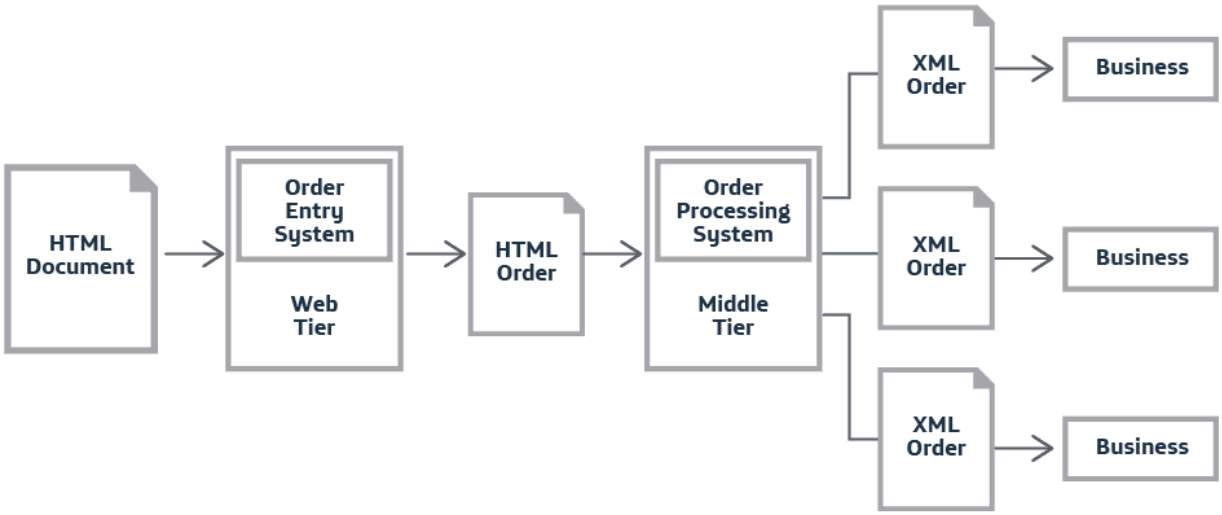
En esta lección perseguimos los siguientes objetivos:

- Conocer cómo se puede acceder y manejar los datos insertados en documentos de marcas.
- Conocer el estándar SAX, su definición, funcionamiento, características y ventajas.
- Conocer el estándar DOM, su definición, funcionamiento, características y ventajas.

Tratamiento de XML desde JAVA

Ya hemos visto que XML es una solución para el intercambio de datos entre distintas plataformas. Es por ello que podemos utilizar XML para almacenar y publicar datos de una forma estructurada.

En determinadas ocasiones puede que sea necesario procesar, validar y transformar documentos XML.



El lenguaje Java proporciona librerías para procesar documentos XML. El conjunto de estas librerías se conoce como el API JAXP.

Actualmente existen tres modelos para llevar a cabo dicho procesamiento:

- Simple API for XML (SAX).
- Document Object Model (DOM).
- Streaming API for XML (StAX).

SAX

Definición

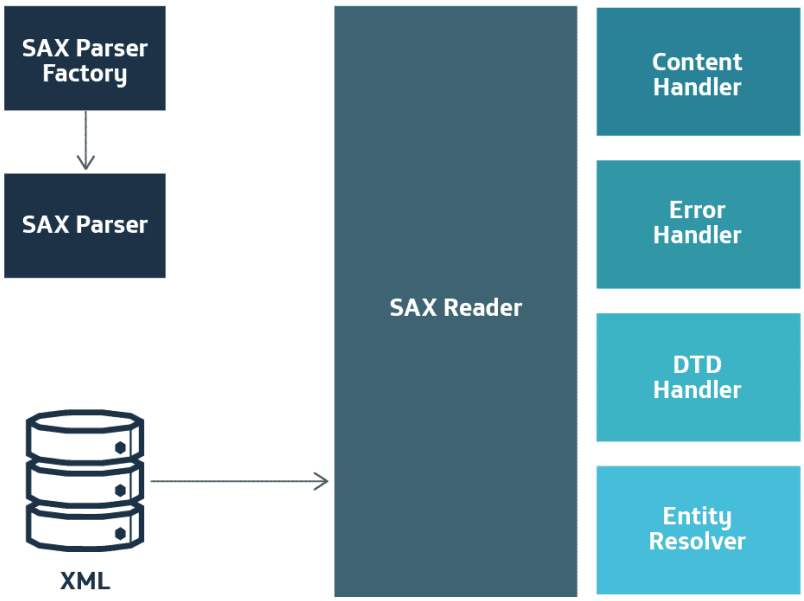
SAX, API para XML, es un estándar público desarrollado para el análisis de documentos XML basado en eventos.

SAX define una interfaz abstracta mediante programación que modela el conjunto de información XML (*infoset*) a través de una secuencia lineal de llamadas a métodos. Por lo tanto podemos indicar que:

- La API SAX es una API orientada a eventos.
- A diferencia del DOM, no conlleva la generación de estructuras internas.
- La mayoría de los procesadores soportan esta API.
- La programación de SAX se realiza en Java.

SAX está definido en los siguientes paquetes:

- *org.xml.sax.*
- *org.xml.sax.ext.*
- *org.xml.sax.helpers.*
- *javax.xml.parsers.*



Características

Las características revisadas de este estándar indican que SAX es el más apropiado para la lectura y validación de documentos en los que solo sea necesario procesarlo una sola vez, ya que SAX no crea un modelo y lo mantiene en memoria para su posterior uso.

El analizador que se use **debe leer el fichero XML secuencialmente**, cada vez que reconoce una etiqueta o nodo lo notifica a la aplicación que lo está ejecutando a través de la llamada a un método de la interfaz.

- El procesamiento se realiza como un *stream* de eventos.
- Los eventos fuerzan la ejecución de métodos de la interfaz, que el programador debe implementar en sus clases.
- Los eventos se producen de forma anidada, como los métodos que se invocan.
- SAX no genera una estructura en memoria del documento, por lo que es más eficiente que DOM.
- Permite la validación de un documento XML.

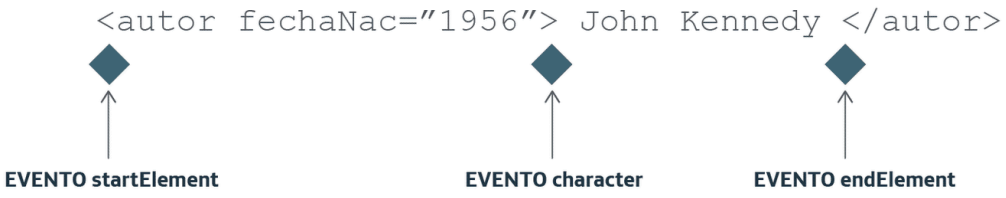
En el caso de tener archivos de gran tamaño es bastante eficiente, pues lee el documento sin ocupar gran cantidad de memoria.

SAX sirve por lo tanto para labores de validación de documentos XML.

Funcionamiento

SAX lee el documento secuencialmente de principio a fin, sin cargarlo en memoria, de forma que cuando encuentra un elemento se encarga de lanzar su evento asociado, que producirá la llamada a un método de la interfaz.

Cuando el evento es lanzado, puede ser capturado para realizar una función determinada. Esta API está definida en el paquete: *javax.xml.parsers.*



Para poder capturar estos eventos y programar las acciones correspondientes de análisis y otras operaciones que se deseen, se debe implementar un manejador de eventos en el lenguaje de programación del sistema de proceso que use el XML.

Un **manejador es una clase con una serie de métodos** y cada método se ejecutará cuando el analizador capture su evento asociado. **Estos eventos se producen al leer un documento** (al

comienzo del documento, apertura o cierre de un elemento, al encontrar una instrucción de proceso o un comentario, etc.).

Los métodos utilizados por SAX para leer un documento son:

- *startDocument.*
- *endDocument.*
- *startElement.*
- *endElement.*
- *characters.*

```
package xmlSax;

import java.util.Arrays;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class EjemploSax  extends DefaultHandler{

	public static void main(String[] args) throws Exception{
		SAXParserFactory factory = SAXParserFactory.newInstance();
		SAXParser saxParser = factory.newSAXParser();
		saxParser.parse("alumno.xml", new EjemploSax());
	}

	@Override
	public void startDocument() throws SAXException {
		System.out.println("Empezamos a leer empleados.xml");
	}

	@Override
	public void endDocument() throws SAXException {
		System.out.println("Terminamos de leer empleados.xml");
	}

	@Override
	public void startElement(String uri, String localName, String qName,
			Attributes at) throws SAXException {
		System.out.println("Procesando " + qName);
	}

	@Override
	public void endElement(String uri, String localName, String qName)
			throws SAXException {
		System.out.println("Fin " + qName);
	}

	@Override
	public void characters(char[] ch, int start, int length)
			throws SAXException {
		String contenido = new String(ch, start, length);
		System.out.println(contenido);
	}
}
```

Ejemplo de lectura de un documento XML desde SAX en Java.

DOM

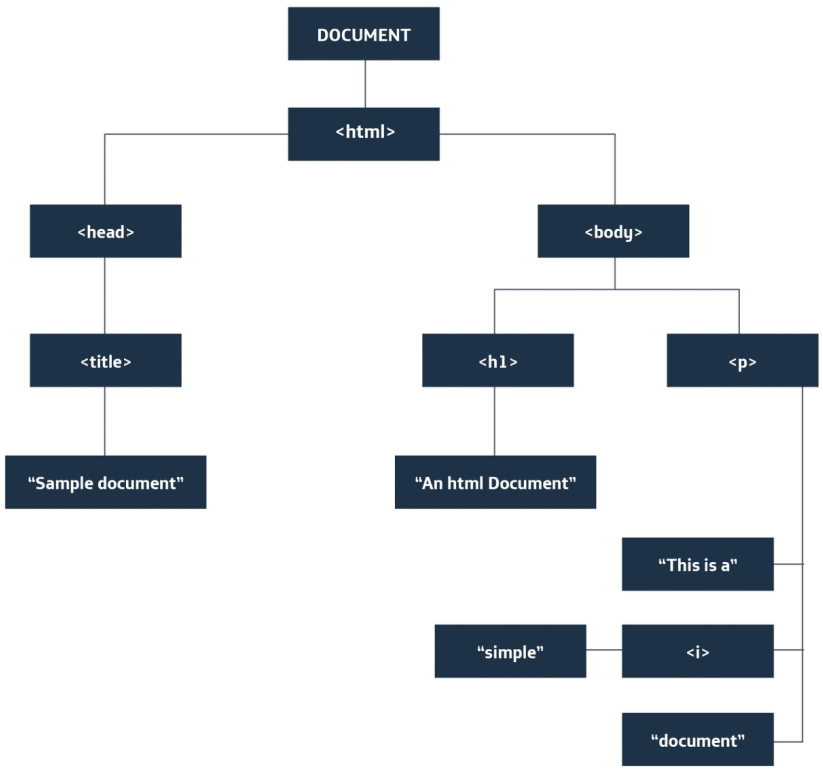
Definición

Al procesar un documento XML a través de DOM se genera lo que es denominado árbol jerárquico en memoria. Este árbol contiene toda la información del fragmento/documento XML en cuestión. Cada elemento del documento pasa a ser un nodo del árbol, por lo que en DOM todo es un nodo.

La API de **"Document Object Model" (DOM)** es un estándar que se define a través de un conjunto de interfaces que describen una estructura abstracta para documentos XML y HTML. **DOM crea el documento XML o HTML entero** en memoria con una estructura tipo árbol. Cada elemento del documento XML o HTML se representa con un nodo (*DOMNode*).

En la imagen siguiente se puede visualizar la estructura generada por un documento HTML. DOM está definido en los paquetes *Java org.w3c.dom*, *javax.xml.transform.dom y javax.xml.parsers.*

El árbol jerárquico de información en memoria permite que a través del manejador pueda manipularse la información: crear o eliminar información de un nodo en cualquier punto del árbol, acceder o cambiar su contenido y mover la herencia de los nodos.



Árbol DOM.

DOM de W3C utiliza una referencia doble para referirse a los objetos del documento. Todo lo que hay en el modelo de objeto es un:

- Nodo.
- Tipo de objeto con nombre.

La implementación del DOM tiene varios niveles. El W3C ha establecido tres niveles de soporte DOM:

Nivel 0

Se utiliza para manipular HTML.

Nivel 1

Permite acceder a todas las partes del cuerpo de un documento XML. No permite acceder a las DTD. Es la recomendación actual.

Nivel 2

Además de las capacidades del nivel 1, permite acceder a DTD, hojas de estilo y espacios de nombres. Aún está en desarrollo.

La principal ventaja de este mecanismo es que toda la información XML se encuentra en memoria. Esta residencia en memoria hace que los datos puedan ser navegables por el programa DOM, permitiendo regresar y manipular información ya procesada, lo que no es posible con SAX.

Por extensión, **DOM también se usa para la manipulación de documentos XHTML y HTML**. La estructura de información de un DOM es a través de nodos, de forma que:

- El documento en sí es un nodo.
- Todos los elementos XML o HTML son elementos nodos.
- Todos los atributos XML o HTML son atributos nodos.
- El texto dentro de los elementos XML o HTML son textos nodos.
- Los comentarios son comentarios nodos.

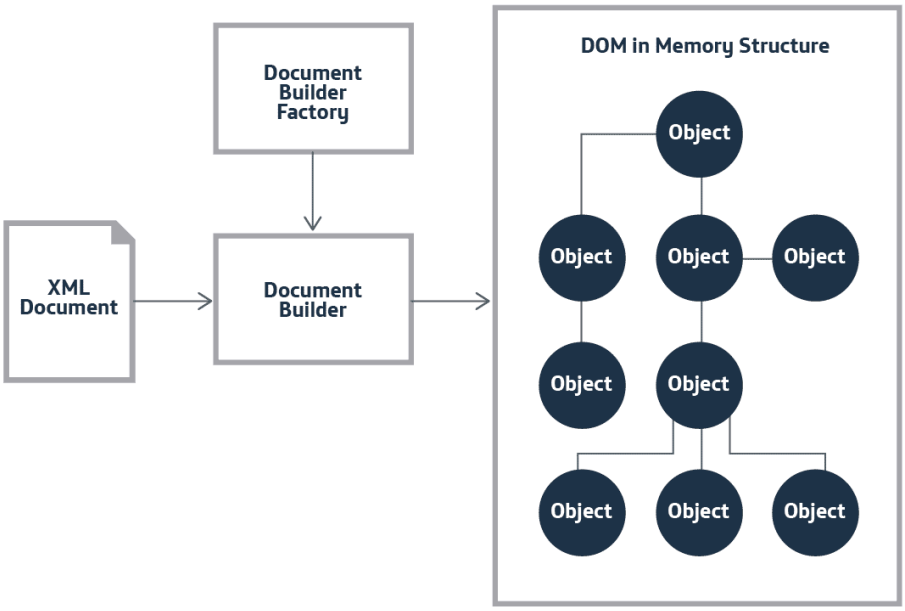
Características

DOM posee las siguientes características:

- Genera una estructura de árbol en memoria al procesar el documento.
- Se puede usar para generar documentos XML.
- Es recurso intensivo.
- Requiere más tiempo y recursos para procesar documentos.
- No es recomendable para grandes documentos.

DOM está definido en los siguientes paquetes:

- org.w3c.dom.
- javax.xml.parsers.



Arquitectura DOM.

Funcionamiento

Para los programadores, trabajar con DOM les permite disponer de un control muy preciso sobre toda la estructura del documento XML.

La API de DOM proporciona funciones que nos permiten añadir, eliminar, modificar y reemplazar cualquier nodo de cualquier documento de forma sencilla. Los objetos del DOM tienen propiedades y la API nos ofrece métodos para poder interactuar con esos elementos.

Los tipos de nodos en el árbol son:

- ***Document***: el nodo raíz de todos los documentos HTML y XML. Todos los demás nodos cuelgan del árbol que genera al cargarse en el navegador.
- ***DocumentType***: contiene la representación DTD que se ha utilizado en la página (mediante el DOCTYPE).
- ***Element***: representa a un elemento del lenguaje de guion, en el caso de HTML serían los elementos propios que se crean con una etiqueta de apertura y otra de cierre. Es el único nodo que puede tener tanto nodos hijos como atributos.
- ***Attr***: representa el atributo y su valor, propios de los elementos.
- ***Text***: almacena el contenido del texto incluido en un elemento.
- ***CDataSection***: es el nodo que representa una sección de tipo *<![CDATA[ ]]>*.
- ***Comment***: representa un comentario.

Necesitamos conocer los métodos que nos permiten navegar entre nodos, conocer las propiedades del nodo y demás. Para ello, dependiendo del **nivel de DOM** al que tengamos acceso a través de su implementación en el navegador, podremos utilizarlos.

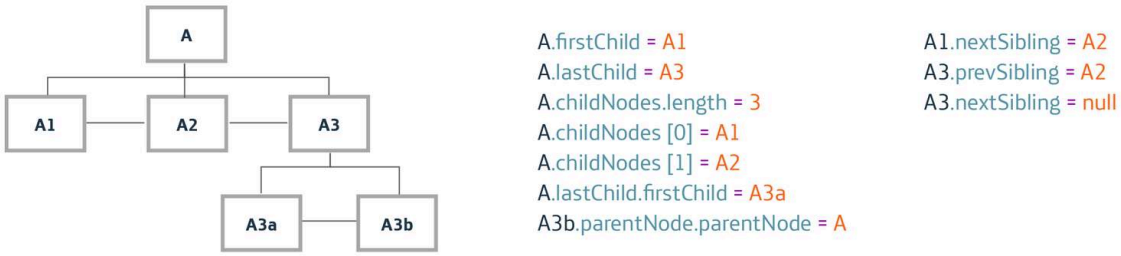
Incluimos un listado de algunos métodos y atributos que nos proporciona el estándar:

Nivel de documento

- *createElement*: crea un elemento nuevo.
- *createTextNode*: crea un nodo de texto.
- *createComment*: crea un nodo comentario.
- *createAttribute*: crea un nodo atributo.
- *getElementsByTagName*: devuelve un listado de nodos con un *tagname* en orden.

Nivel de nodo

- *get* y *set* de *NodeName*: nombre del nodo.
- *get* y *set* de *NodeValue*: valor del nodo.
- *get* y *set* de *NodeType*: tipo del nodo.
- *get* y *set* de *ParentNode*: padre del nodo actual.
- *get* y *set* de *ChildNodes*: una lista de nodos hijo.
- *get* y *set* de *FirstChild*: primer hijo del nodo.
- *get* y *set* de *FastChild*: último hijo del nodo.
- *get* y *set* de *PreviousSibling*: nodo precedente.
- *get* y *set* de *NextSibling*: nodo siguiente.
- *get* y *set* de *Attributes*: lista de atributos del nodo.



Métodos de jerarquía DOM.

Ejemplo

Veamos un ejemplo de cómo crear un archivo XML y convertirlo en una clase de Java.

```
package xmlDom;

import java.io.File;
import java.io.FileWriter;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
```

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;

public class Principal {

      public static void main(String[] args) throws Exception{

            String nombreArchivo = "probando.xml";
            try{

            File archivo = new File(nombreArchivo);
            FileWriter escribir = new FileWriter(archivo);
            String cabecera = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<empleados></empleados>";

            escribir.write(cabecera);
            escribir.close();
                  }catch (Exception e) {
                        System.out.println("Error al crear archivo");

                  }

            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document document = db.parse(nombreArchivo);

            Element raiz = document.getDocumentElement();

            Element newEmp = document.createElement("empleado");

            Element newId = document.createElement("id");
            Text IDText = document.createTextNode("00133");
            newId.appendChild(IDText);

            Element newNombre = document.createElement("nombre");
            Text nombreText = document.createTextNode("Luis Sanchez");
            newNombre.appendChild(nombreText);

            Element newDir = document.createElement("direccion");

            Element newCalle = document.createElement("calle");
            Text calleText = document.createTextNode("Castellana");
            newCalle.appendChild(calleText);

            Element newCiudad = document.createElement("ciudad");
            Text ciudadText = document.createTextNode("Madrid");
            newCiudad.appendChild(ciudadText);

            Element newCp = document.createElement("codigo_postal");
            Text cpText = document.createTextNode("28027");
            newCp.appendChild(cpText);

            newDir.appendChild(newCalle);
            newDir.appendChild(newCiudad);
            newDir.appendChild(newCp);
```

```
            Element newEmail = document.createElement("email");
            Text emailText = document.createTextNode("luis@yahoo.es");
            newEmail.appendChild(emailText);

            Element newDpto = document.createElement("dpto");
            Text dptoText = document.createTextNode("Compras");
            newDpto.appendChild(dptoText);

            Element newSueldo = document.createElement("sueldo");
            Text sueldoText = document.createTextNode("60000");
            newSueldo.appendChild(sueldoText);

            newEmp.appendChild(newId);
            newEmp.appendChild(newNombre);
            newEmp.appendChild(newDir);
            newEmp.appendChild(newEmail);
            newEmp.appendChild(newDpto);
            newEmp.appendChild(newSueldo);

            raiz.appendChild(newEmp);

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

            DOMSource source = new DOMSource(document);
            StreamResult resultado = new StreamResult(nombreArchivo);

            transformer.transform(source, resultado);

      }
}
```

Código de la clase para la creación de un XML mediante DOM en Java.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<empleados>
      <empleado>
            <id>00133</id>
            <nombre>Luis Sanchez</nombre>
            <direccion>
                  <calle>Castellana</calle>
                  <ciudad>Madrid</ciudad>
                  <codigo_postal>28027</codigo_postal>
            </direccion>
            <email>luis@yahoo.es</email>
            <dpto>Compras</dpto>
            <sueldo>60000</sueldo>
      </empleado>
</empleados>
```

El XML resultante.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.
El análisis y modificación de documentos de marcas como XML y HTML es necesario para que los sistemas de procesamiento que los usan puedan manejarlos y entender sus datos.

Muchos *frameworks* usan documentos XML para emitir directivas de proceso o definir su propia configuración. Poder entender y analizar estos documentos es vital para el funcionamiento de dichos *frameworks*.

En este tema hemos visto los conceptos más básicos de cómo **trabajar con documentos XML utilizando las API de SAX y DOM**.

Las **ventajas** de procesar documentos XML **utilizando el modelo SAX son:**

- Permite procesar documentos de cualquier tamaño.
- Es útil cuando solo queremos procesar parte del documento.
- Es muy simple y rápido.
- No consume mucha memoria ni necesita mucho procesador.

Las **ventajas** del uso de **DOM** son:

- Puede crear elementos e insertarlos en el árbol.
- Puede modificar los valores de los elementos.
- Si se necesita, se pueden realizar múltiples procesados, ya que el árbol está en memoria.
- Una vez analizado y validado un documento se evita tener que volver a analizar el documento.
- Dependiendo del procesador utilizado, se puede acceder al DOM utilizando C++, Java, JavaScript y VBScript.