IES DR LLUÍS SIMARRO
ENSENYAMENT PÚBLIC
XATIVA

Acceso a datos

UD02. Acceso a ficheros XML

Desarrollo de Aplicaciones Multiplataforma

Enric Giménez Ribes

e.gimenezribes@edu.gva.es

ÍNDICE

1. XML

El **lenguaje XML** se ha consolidado como uno de los estándares por excelencia en el intercambio de datos. Es ampliamente usado en el mundo de Internet, pero también lo es en otros tipos de aplicaciones como hojas de cálculo, bases de datos, etc. El objetivo de este documento no es comentar qué es XML y las ventajas que su uso conlleva. Una vez familiarizado con XML, el objetivo es explicar cómo procesar este metalenguaje desde aplicaciones Java por medio de distintos analizadores o parsers.

XML: Características

- Los documentos XML estructuran la información intercalando una serie de marcas denominadas etiquetas.
- Las etiquetas son contenedores de información.
- Una etiqueta puede contener otras etiquetas o información textual.
- Se puede subdividir la información estructurada de forma que pueda ser fácilmente interpretada.
- Como toda la información es textual, no existe el problema de representar los datos de diferente manera.

Definición del Tipo de Documento (DTD)

La validación del documento XML consiste en comprobar que el documento, además de estar bien formado de acuerdo a las reglas de XML, responde a una estructura definida en una **Definición del Tipo de Documento (DTD)**.

Analizador o parser XML

Un analizador o parser XML es una herramienta encargada de leer documentos XML, poder acceder a sus elementos y comprobar si el documento es sintácticamente válido. Estas herramientas son módulos, bibliotecas o programas que se ocupan de transformar un archivo XML en una representación interna.

Los analizadores usados para el tratamiento de XML son los siguientes:

1. **Jerárquicos → DOM (Document Object Model):**
    - Guardan todos los datos de XML en memoria dentro de una estructura jerárquica.
    - Permiten la creación de documentos XML.
    - Es un estándar independiente del lenguaje.
2. **Secuenciales → SAX (Simple API for XML):**
    - Únicamente procesaremos documentos XML, es decir, lo leeremos.
    - Permiten extraer el contenido a medida que se van descubriendo las etiquetas de apertura y cierre.
    - Son rápidos, pero tienen que leer todo el documento a cada consulta.

La diferencia estriba, básicamente, en que mientras **SAX implementa un procesado de documentos de acceso en serie y basado en eventos**, es decir, se encarga de recorrer la estructura del documento generando eventos que corresponden a los elementos que se va encontrando, mientras que **DOM mapea el documento XML en un árbol de objetos**, o lo que es lo mismo, representa el archivo en una estructura tipo árbol que usará para leer el documento.

1.1. Características DOM contra SAX

Característica	DOM	SAX (Simple API for XML)
Forma de trabajo	Carga todo el XML en memoria (árbol completo)	Procesa línea a línea (evento secuencial)
Acceso a datos	Aleatorio: puedes acceder a cualquier nodo	Secuencial: no puedes retroceder
Memoria usada	Alta (depende del tamaño del XML)	Baja
Facilidad de uso	Muy intuitivo (ideal para aprendizaje)	Más complejo (usa callbacks y eventos)
Ideal para...	Documentos pequeños o medianos	Documentos muy grandes

2. XML Y JAVA: DOM

La API **DOM (Document Object Model)**, definida en los paquetes javax.xml.parsers y org.w3c.dom, proporciona un modelo de objetos para representar documentos XML.

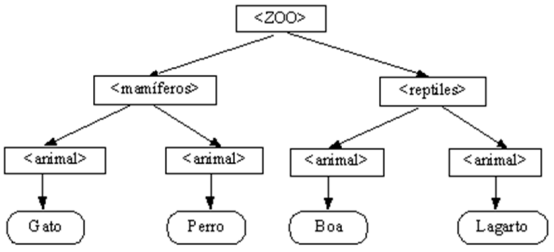
Su función principal es **convertir un documento XML en un árbol de objetos en memoria**, donde cada nodo del árbol representa un componente del documento (elementos, atributos, comentarios, texto, etc.). Los nodos están relacionados entre sí mediante vínculos padre-hijo, lo que permite navegar por el documento de forma estructurada.

Realmente DOM es una especificación del W3C, por lo que su objetivo principal es proporcionar una interfaz estándar que pueda ser utilizada en una amplia variedad de entornos y lenguajes de programación (Java, JavaScript, C++, PHP, Python, Ruby...).

DOM se compone de un conjunto de interfaces que describen una estructura jerárquica estándar para cualquier documento XML. Gracias a ello, es posible **recorrer, consultar y modificar** el contenido de un XML directamente desde Java.

El hecho de que DOM genere un **árbol de nodos** a partir de un documento XML, se debe interpretar como que lee el documento XML, lo procesa y genera una serie de objetos que representan cada uno de los elementos del documento XML. Estos elementos están relacionados entre sí. Por ejemplo, las etiquetas están relacionadas con sus atributos y sus etiquetas hijas.

```
<zoo>
   <mamiferos>
      <animal> Gato </animal>
      <animal> Perro </animal>
   </mamiferos>
   <reptiles>
      <animal> Boa </animal>
      <animal> Lagarto </animal>
   </reptiles>
</zoo>
```



Como podrás observar, solamente se guardan como nodos las etiquetas de apertura y no las de cierre, y el contenido de la etiqueta se guarda en un nodo aparte.

2.1. Estructura del árbol DOM: la interfaz Node

La interfaz **Node**, ubicada en el paquete **org.w3c.dom**, es la base de todos los objetos del modelo DOM. Cada elemento del documento XML se representa mediante un nodo.

Vamos a ver los tipos de nodos más utilizados:

- **Document:**

  Representa al documento XML completo, es decir, la raíz lógica del árbol DOM. Es el nodo principal a partir del cual se organizan todos los demás elementos (Element, Attr, Text, etc.). Solo puede existir un objeto Document por cada archivo XML. Cualquier nodo que desees insertar en el árbol debe ser creado o importado previamente mediante el propio objeto Document; de lo contrario, no podrá añadirse como hijo de otro elemento.

  Almacena información como el encoding (getEncoding), la versión xml (getXmlVersion), si el documento es Standalone (getStandalone), y tiene como único hijo a la etiqueta raíz del documento (getDocumentElement).

- **Element:**

  Representa una etiqueta XML. Almacena entre otras la lista de atributos que posee (getAttributes), la lista de nodos hijos (getChildNodes), contenido de tipo texto (getTextContent). Tiene métodos de acceso directo a su primer hijo (getFirstChild) y acceso directo a su próximo nodo hermano (getNextSibling). Ambos devuelven un Node por lo que el primer hijo o el próximo hermano pueden ser de cualquier tipo (Element, Text, CDATASection...).

- **Attr:**

  Atributo dentro de un Element. Existen métodos para obtener información de los atributos de una etiqueta.

- **Text:**

  El texto entre la apertura y el cierre de una etiqueta.

- **Comment:**

  Un comentario entre <!-- -->.

Ejemplo XML sencillo (clientes.xml):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<clientes>
  <!-- Lista de clientes de la academia -->
  <cliente id="1">
    <nombre>Laura Gómez</nombre>
    <email>laura@example.com</email>
  </cliente>
</clientes>
```

Cómo lo ve DOM (esquema conceptual):

```
Document
└── Element: clientes
    ├── Comment: "Lista de clientes de la academia"
    ├── Element: cliente  (Attr id="1")
        ├── Element: nombre
        │   └── Text: "Laura Gómez"
        └── Element: email
            └── Text: "laura@example.com"
```

Puntos clave que debemos interiorizar:

- Siempre hay un Document. A partir de él obtenemos el elemento raíz mediante el mértodo getDocumentElement().
- Cada Element puede tener hijos (otros Element o Text) y atributos (Attr).
- El texto nunca está en el Element directamente: vive dentro de un nodo Text.

En resumen, las fases del procesamiento en XML con DOM son:

A. **Analizar un documento XML** generando la estructura de árbol asociado o crear un documento XML nuevo. Esto nos proporciona el objeto Document.
B. **Desplazarse por el árbol,** leer/recorrer nodos, buscar por etiquetas, crear/actualizar/eliminar nodos.
C. **Serializar el árbol**, es decir, guardar el resultado a disco o por la consola.

2.2. Pasos a seguir

El detalle del análisis y modificación, si se plantea, sigue los pasos siguientes:

1. **Crear un DocumentBuilder.**

   Para poder hacer uso del analizador se debe obtener una instancia de una factoría analizadora (DocumentBuiderFactory). Con esta factoría se crea el analizador (DocumentBuilder) que es capaz de producir un nodo Document que cumple la especificación DOM, es decir, crear el inicio del árbol.

   Esto es básicamente un contenedor de tipo Document sobre un analizador XML específico.

```
...
DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = builderFactory.newDocumentBuilder();
...
```

2. **Invocar el analizador para crear un Document que representa el archivo XML.**

   En este punto se pueden hacer dos cosas:

   - Crear un nodo Document vacío con el método newDocument().

```
Document document = builder.newDocument();
```

   - Crear el árbol completo procedente del análisis de un documento XML pasándole éste al analizador con el método parser(documento_XML).

```
Document document = builder.parse(new File("/ruta_fichero/fichero.xml");
```

   La clase Document representa el resultado del análisis en una estructura de árbol.

3. **Normalizar el árbol.**

   Este paso, a veces opcional, se realiza para eliminar nodos de texto vacíos.

```
document.getDocumentElement().normalize();
```

4. **Obtener el nodo raíz del árbol.**

   Esta acción devuelve un objeto de tipo Element, el cual es una subclase de la clase Node que representa un elemento XML.

```
Element rootElement = document.getDocumentElement();
```

5. **Examinar las propiedades (información) procedente de un nodo.**

   Obtener las propiedades de un nodo incluyen:

   - El nombre del nodo (método getNodeName()),
   - El tipo de nodo (método getNodeType() comparándolo con las constantes definidas en la clase Node):

NODETYPE	NOMBRE DE LA CONSTANTE	NODETYPE	NOMBRE DE LA CONSTANTE
1	ELEMENT_NODE	7	PROCESSING_INSTRUCTION_NODE
2	ATTRIBUTE_NODE	8	COMMENT_NODE
3	TEXT_NODE	9	DOCUMENT_NODE
4	CDATA_SECTION_NODE	10	DOCUMENT_TYPE_NODE
5	ENTITY_REFERENCE_NODE	11	DOCUMENT_FRAGMENT_NODE
6	ENTITY_NODE	12	NOTATION_NODE

   - El valor del nodo (método getNodeValue()) para obtener el texto comprendido entre el inicio y fin de una etiqueta.
   - Los atributos (método getAttributes()) especificados en la etiqueta de inicio de un elemento.

- La lista de nodos hijos (método getChildNodes()) los elementos comprendidos en una etiqueta. Se puede consultar recursivamente los nodos hijos de un nodo hijo.

Recordar que NodeName y NodeValue son propiedades que difieren dependiendo del tipo de nodo. Por ejemplo, para un nodo de tipo elemento (Element), NodeName devolverá el nombre del elemento, pero para un nodo de texto (Text) devolverá #text.

En la siguiente tabla se pueden ver todas las propiedades y métodos de Node:

Propiedad/Método	Valor devuelto	Descripción
getNodeName()	String	El nombre del nodo (no está definido para algunos tipos de nodo).
getNodeValue()	String	El valor del nodo (no está definido para algunos tipos de nodo).
getNodeType()	Number	Una de las 12 constantes definidas anteriormente.
getOwnerDocument()	Document	Referencia del documento al que pertenece el nodo.
getFirstChild()	Node	Referencia del primer nodo de la lista childNodes.
getLastChild()	Node	Referencia del último nodo de la lista childNodes.
getChildNodes()	NodeList	Lista de todos los nodos hijo del nodo actual.
getPreviousSibling()	Node	Referencia del nodo hermano anterior o null si este nodo es el primer hermano.
getNextSibling()	Node	Referencia del nodo hermano siguiente o null si este nodo es el último hermano.
hasChildNodes()	Boolean	Devuelve true si el nodo actual tiene uno o más nodos hijo.
getAttributes()	NamedNodeMap	Se emplea con nodos de tipo Element. Contiene objetos de tipo Attr que definen todos los atributos del elemento.
appendChild(nodo)	Node	Añade un nuevo nodo al final de la lista childNodes.
removeChild(nodo)	Node	Elimina un nodo de la lista childNodes.
replaceChild(nuevoNodo, anteriorNodo)	Node	Reemplaza el nodo anteriorNodo por el nodo nuevoNodo.
insertBefore(nuevoNodo, anteriorNodo)	Node	Inserta el nodo nuevoNodo antes que la posición del nodo anteriorNodo dentro de la lista childNodes.

- https://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Node.html

6. **Modificar las propiedades de los nodos si se desea.**

   En vez de obtener los datos procedentes de un documento XML, se puede modificar el mismo añadiendo nuevos nodos hijos (método appendChild()), eliminar nodos hijo (método removeChild()) o cambiar el valor de un nodo (método setNodeValue).

En DOM, el árbol de nodos está en memoria y, por tanto, está ocupando un espacio que puede ser considerable dependiendo del tamaño del fichero XML. En principio no hay inconveniente, ya que hoy en día la RAM no es un problema para los ordenadores. Sin embargo, tampoco hay que olvidar de este posible problema, ya que si se descuida en el momento más inesperado pueden producirse errores del tipo OutOfMermoryException.

2.3. Archivo bookstore.xml

El fichero que utilizaremos para nuestros ejemplos que tenéis en el repositorio:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
   <book category="cooking">
     <title lang="en">Everyday Italian</title>
     <author>Giada De Laurentiis</author>
     <year>2005</year>
     <price>30.00</price>
   </book>
   <book category="children">
     <title lang="en">Harry Potter</title>
     <author>J K. Rowling</author>
     <year>2005</year>
   </book>
   <book category="web">
     <title lang="en">XQuery Kick Start</title>
     <author>James McGovern</author>
     <author>Per Bothner</author>
     <author>Kurt Cagle</author>
     <author>James Linn</author>
     <author>Vaidyanathan Nagarajan</author>
     <year>2003</year>
     <price>49.99</price>
   </book>
   ...
</bookstore>
```

2.4. Lectura de archivos XML: métodos relacionados

Métodos de la interfaz Document:

- **Element getDocumentElement()** → Devuelve el elemento raíz del documento.
- **NodeList getElementsByTagName(String name)** → Devuelve una lista de nodos hijos que coinciden con el nombre de la etiqueta dada. Esta interfaz tiene unos métodos bastante interesantes:
    - El método getLength(), devuelve la cantidad de elementos de tipo NodeList.
    - El método item(int index), devuelve un elemento particular de la lista.

Métodos de Node:

- **String getNodeName()** → Obtiene el nombre del nodo actual.
- **short getNodeType()** → Obtiene el tipo de nodo (*ELEMENT_NODE, ATTRIBUTE_NODE, TEXT_NODE...*).
- **String getNodeValue()** → Obtiene el valor del nodo.
- **NodeList getChildNodes()** → Obtiene una lista con los nodos hijos.
- **Node getFirstChild()** → Devuelve el primer hijo.
- **Node getLastChild()** → Devuelve el último hijo.
- **NamedNodeMap getAttributes()** → Devuelve una lista con los atributos del nodo.
- **Node getParentNode()** → Devuelve el nodo padre.
- **String getTextContent()** → Devuelve el texto contenido en el elemento y sus descendientes.
- **boolean hasChildNodes()** → Devuelve true si el nodo tiene algún hijo.
- **boolean hasAttributes()** → Devuelve true si el nodo tiene algún atributo.

Métodos de la interfaz Element:

- **String getAttribute(String name)** → Devuelve el valor del atributo indicado por su nombre.
- **NodeList getElementsByTagName(String name)** → Devuelve una lista de nodos hijos que coinciden con el nombre dado.
- **boolean hasAttribute(String name)** → Devuelve true si el elemento tiene el atributo indicado.

2.5. Lectura de archivos XML: XPath

XPath (XML Path Language) es un lenguaje que permite seleccionar nodos de un documento XML y calcular valores a partir de su contenido. La idea es parecida a las expresiones regulares pero aplicado a XML. Básicamente, se trata de describir una ruta a través del árbol de nodos como si cada etiqueta fuera un directorio.

Por ejemplo, **/etiquetaPrincipal/etiquetaHija**:

- Esta ruta XPath seleccionaría la etiquetaHija de etiquetaPrincipal de nuestro XML.

Por supuesto, es mucho más complejo y potente: hay funciones, caracteres comodín, condicionales...

Sobre el lenguaje XPath, la mejor forma es consultar teoría y ejemplos, para ello una posibilidad consiste en acceder a un tutorial, a continuación se cita una web de referencia: Tutorial de XPath.

Conceptos XPath

Al igual que DOM, XPath considera un documento XML como un árbol de nodos.

Una expresión XPath es una cadena de texto que representa un recorrido en el árbol de nodos.

XPath utiliza como expresión patrón la barra (/) al inicio de la expresión, seguida de una lista con los nombres de los elementos hijos que describen un recorrido a través del documento.

Las expresiones más simples se parecen a las rutas de los archivos en el explorador de Windows o en la shell de GNU/Linux.

Por ejemplo, la expresión Xpath /libro/capitulo/parrafo, hace referencia a TODOS los elementos párrafo que cuelguen directamente de CUALQUIER elemento capitulo que cuelgue de CUALQUIER elemento libro, que a su vez cuelgan del nodo raíz.

Evaluar una expresión XPath consiste en buscar si hay nodos en el documento que se ajustan al recorrido definido en la expresión. El resultado de la evaluación son todos los nodos que se ajustan a la expresión.

Sintaxis XPath

Las expresiones XPath se pueden dividir en pasos de búsqueda. Cada paso de búsqueda se puede a su vez dividir en tres partes:

1. **Eje:** nos permite seleccionar un subconjunto de nodos del documento. Se corresponde con recorridos en el árbol del documento.
   - Los nodos "Elemento" se indican mediante el nombre del elemento.
   - Los nodos "Atributo" se indican mediante @ y el nombre del atributo.
   - / si está al principio de la expresión, indica el nodo raíz,
   - / entre medio del eje, indica nodo "hijo" y debe ir seguida del nombre de un elemento.

   Veamos algún ejemplo:

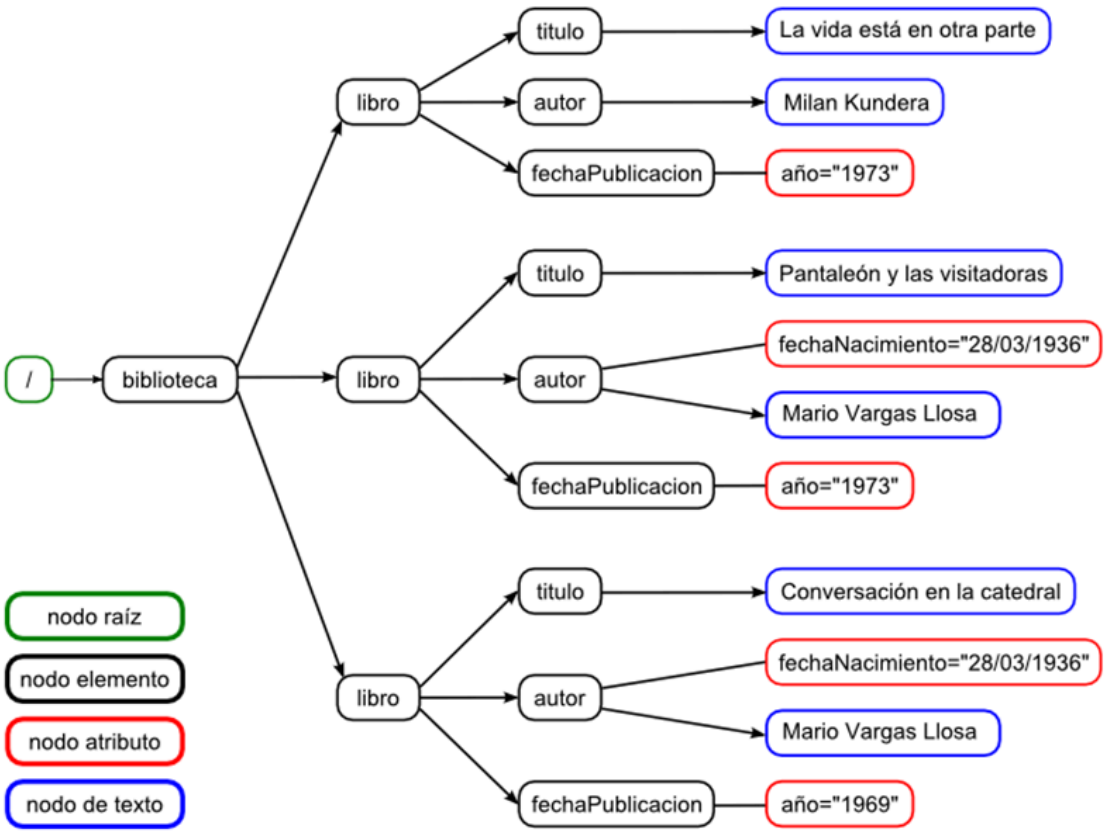
/AAA : Selecciona el elemento raíz <AAA>	**/AAA/CCC :** Selecciona todos los elementos <CCC> que son hijos del elemento raíz <AAA>	**/AAA/DDD/BBB :** Selecciona todos los elementos <BBB> que son hijos de <DDD>, y hijos del elemento raíz <AAA>
<AAA> <BBB/> <CCC/> <BBB/> <BBB/> <DDD> <BBB/> </DDD> <CCC/> **</AAA>**	<AAA> <BBB/> **<CCC/>** <BBB/> <BBB/> <DDD> <BBB/> </DDD> **<CCC/>** </AAA>	<AAA> <BBB/> <CCC/> <BBB/> <BBB/> <DDD> **<BBB/>** </DDD> <CCC/> </AAA>

2. **Predicados:** restringe la selección del eje a que los nodos cumplan ciertas condiciones.
   - //autor[@fechaNacimiento] muestra sólo los autores que tiene fecha de nacimiento
3. **Selección de nodos:** de los nodos seleccionados por el eje y predicado, selecciona los elementos, el texto que contienen o ambos.

Partimos del siguiente archivo XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<biblioteca>
   <libro>
      <titulo>La vida está en otra parte</titulo>
      <autor>Milan Kundera</autor>
      <fechaPublicación año="1973" />
   </libro>
   <libro>
      <titulo>Pantaleón y las visitadoras</titulo>
      <autor fechaNacimiento="28/03/1936">Mario Vargas Llosa</autor>
      <fechaPublicación año="1973" />
   </libro>
   <libro>
      <titulo>Conversación en la catedral</titulo>
      <autor fechaNacimiento="28/03/1936">Mario Vargas Llosa</autor>
      <fechaPublicación año="1969" />
   </libro>
</biblioteca>
```

A continuación, mostramos el árbol de nodos correspondientes al archivo anterior:



Vamos a ver diferentes ejemplos de expresiones XPath:

- **/biblioteca/libro/autor** → muestra todos los autores de todos los libros.
- **/autor** → no muestra nada porque "autor" no es hijo del nodo raíz.
- **/biblioteca/autor** → no muestra nada porque autor no es hijo de "biblioteca".

// Indica descendientes (hijos, hijos de hijos…):

- **/biblioteca//autor** → muestra todos los autores.
- **//autor** → muestra todos los autores.
- **//autor//libro** → no muestra nada porque "libro" no es descendiente de "autor".

@ Selecciona el atributo:

- **/biblioteca/libro/autor/@fechaNacimiento** → muestra fechas de nacimiento de los autores.

.. Selecciona el elemento padre:

- **/biblioteca/libro/autor/@fechaNacimiento/..** → muestra los autores que tienen fecha de nacimiento.

| Permite indicar varios recorridos:

- **//autor | //titulo**
- **//autor | //titulo | //@año**

***** Permite seleccionar todo:

- **/*** → selecciona todos los hijos (sólo elementos) del nodo.
- **//*** → selecciona todos los descendientes (sólo elementos) del nodo.
- **//@*** → Permite seleccionar todos los atributos.

[n] Permite seleccionar un número determinado de nodos:

- **//libro[1]** → selecciona el primer libro.
- **//libro[last()]** → selecciona el último libro.

[condición] Selecciona los nodos que cumplen la condición:

- **//fechaPublicacion[@año>1970]**
- **//libro[autor="Mario Vargas Llosa"]**

Implementación en Java

Una vez conocido el lenguaje de consulta XPath se pueden realizar las consultas sobre el XML. Para ello, hay que generar un objeto de la clase XPath, que a su vez, requiere una instancia de XPathFactory.

```
XPathFactory xpathFactory = XPathFactory.newInstance();
XPath xpath = xpathFactory.newXPath();
```

Sobre el objeto XPath se puede efectuar la consulta con el método evaluate(), cuya sintaxis es:

```
Object xpath.evaluate(expresionXPath, nodoDom, tipo)
```

donde:

- **expresionXPath** es un String con la consulta que se quiere realizar.
- **nodoDom** es un objeto de tipo org.w3c.dom.Node, usualmente corresponde al documento sobre el que realizar la consulta.
- **tipo** puede ser booleano, lista de nodos, un nodo, un número o una cadena.

Concretamente, el tipo es uno de los siguientes:

- XPathConstants.NUMBER, y en este caso la llamada a evaluate() nos retornará un objeto Double.
- XPathConstants.STRING, retornando un String.
- XPathConstants.BOOLEAN, retornando un Boolean.
- XPathConstants.NODE, retornando un objeto org.w3c.dom.Node.
- XPathConstants.NODESET, retornando un objeto org.w3c.dom.NodeList.

Como el método retorna un objeto de tipo Object, se debe convertir al tipo apropiado en cada llamada. Observar también que org.w3c.dom.Document deriva de org.w3c.dom.Node, por lo que podremos usar un documento DOM completo para realizar las consultas XPath.

Ejemplo de uso XPATH

Un ejemplo de uso de este método es el siguiente, en el que se aprecia la conversión explícita y como indicamos el tipo de retorno que se quiere en el último argumento de la llamada:

```
String s = (String) xpath.evaluate("/raiz/str", nodoDom, XPathConstants.STRING);
```

La llamada a este método lanzará una excepción de tipo XPathExpressionException si la expresión XPath no se puede evaluar, por ejemplo, debido a un error de sintaxis. Sin embargo, no se lanzará la excepción si la expresión es correcta, aunque haga referencia a elementos que no existen.

Veamos ahora una tabla con los datos de retorno de la función en diferentes casos. En la primera columna se indica aquello que se intenta recuperar en la expresión XPath, en las siguientes columnas aquello que se obtiene según el tipo indicado en el tercer argumento de evaluate():

	NUMBER	STRING	BOOLEAN	NODE	NODESET
nodo existente y con contenido	Double	String	true	Node	NodeList
nodo vacío	NaN (Not a Number)	""	true	Node con el nodo vacío	NodeList de un elemento con el nodo vacío
nodo inexistente	NaN	""	false	null	NodeList sin contenido
lista de nodos				Node con el primer nodo	NodeList con la lista completa

⚠️ **Importante: Mirar los ejemplos App1.java y App2.java del proyecto adt2_ejemplos_dom.**

2.6. Lectura de archivos XML: recorrer el árbol de nodos DOM

DOM permite el acceso a los nodos de un árbol mediante la exposición de propiedades que funcionan con estos conceptos.

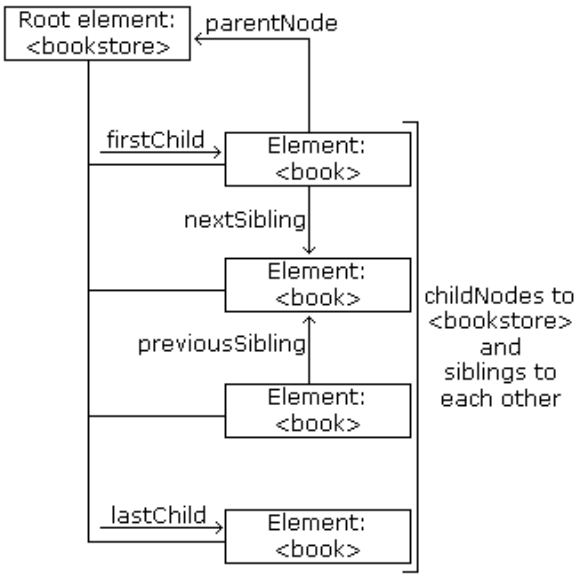
Estas propiedades son:

- **parentNode** retorna el nodo al que pertenece ese nodo en el árbol.
- **firstChild** retorna el primer nodo hijo del nodo contexto.
- **lastChild** retorna el último nodo hijo del nodo contexto.
- **previuosSibling** retorna el nodo hermano anterior al nodo contexto del árbol.
- **nextSibling** retorna el nodo hermano posterior al nodo contexto del árbol.
- **childNodes** retorna una lista (NodeList) con todos los nodos hijos del nodo contexto.

Excepto childNodes, todas las demás propiedades devuelven un nodo.

No todos los nodos pueden tener hijos como por ejemplo los atributos. O incluso aunque pueda tenerlos, es posible que no los tenga. En ambos casos, cuando se intenta acceder a ellos se obtiene como retorno null.

En el siguiente diagrama se muestra un nodo y el nodo que retornaría utilizando cada una de las propiedades mencionadas anteriormente.



DOM también expone una propiedad hasChildNodes que es booleana y que indica si un nodo tiene hijos (true) o no (false). Hay que tener en cuenta que esto incluye nodos de texto, por lo que si un elemento solamente tiene texto como hijo, esta propiedad retorna true.

Una vez tenemos el objeto Document, se accede al árbol. Es importante destacar que para obtener la etiqueta raíz del árbol DOM, uno de los mecanismos más habituales de acceso es por medio del método getDocumentElement(), el cual devuelve un objeto Element correspondiente al elemento raíz.

Descendiendo por el árbol de nodos

A partir de ese momento, podemos ir descendiendo por el árbol de nodos a partir de la etiqueta principal y así sucesivamente por los hijos de los nodos.

```
// Obtenemos la etiqueta raíz
Element elementRaiz = doc.getDocumentElement();

// Iteramos sobre sus hijos
NodeList hijos = elementRaiz.getChildNodes();
for(int i=0; i<hijos.getLength(); i++){
   Node nodo = hijos.item(i);
   if (nodo instanceof Element){
       System.out.println(nodo.getNodeName());
   }
}
```

Como se puede ver en el ejemplo, es interesante asegurarse de que los nodos devueltos son Element. No todos los hijos de un Element lo serán. Puede evitar muchos errores el acostumbrarse a ello, ya que son errores en tiempo de ejecución y el compilador no avisa de ellos.

Este ejemplo de acceso permite la visualización por pantalla de los nodos hijos del nodo raíz, pero ello no permite la visualización completa de todos y cada uno de los nodos del árbol DOM. Para conseguir tal fin, sería interesante la realización de una llamada recursiva que muestre todos y cada uno de los nodos del árbol. Esta invocación recursiva se deja propuesta para su realización.

2.7. Escritura de archivos XML: métodos relacionados

Métodos de la interfaz Document:

- **Element createElement(String name)** → Crea un nuevo elemento con el nombre indicado.
- **Text createTextNode(String text)** → Crea un nuevo nodo de texto.
- **Node appendChild(Node node)** → Añade un nuevo nodo hijo.

Métodos de Node:

- **Node appendChild(Node node)** → Añade un nuevo nodo como último hijo.
- **void removeChild(Node node)** → Elimina el nodo especificado de la lista de nodos hijos.

Métodos de la interfaz Element:

- **void setAttribute(String name, String value)** → Añade un atributo al elemento con el nombre y el valor indicados.
- **void setTextContent(String texto)** → Crea un hijo de nodo de tipo texto.
- **void removeAttribute(String name)** → Elimina el atributo indicado por su nombre.

Tener en cuenta que podemos modificar el árbol DOM a partir de un fichero XML o crear el árbol desde cero. La única diferencia clave es cómo obtenemos el Document inicial:

- Si partimos de un XML existente, el Document lo devuelve el parser al hacer parse(...) y ya trae nodo raíz y contenido.
- Si lo creamos desde cero, el Document se obtiene con newDocument() y viene vacío: debemos crear explícitamente el elemento raíz y luego ir añadiendo nodos (elementos, atributos y textos).

En ambos casos, las operaciones posteriores (crear nodos, añadir hijos, establecer atributos, etc.) son exactamente las mismas; cambia solo el punto de partida.

2.8. Guardar o serializar el árbol DOM a fichero XML

Una vez conocido el pasar un fichero XML a un árbol DOM, modificarlo e incluso crear nuevos documentos XML, ahora corresponde ver cómo volver a convertir dicho árbol en un fichero XML, el cual puede ser almacenado en un archivo o visualizado por pantalla, entre otras opciones. En general, cualquier cosa que pueda ser serializable mediante un Writer o un OutputStream como un String o un array de bytes.

Para la conversión del árbol DOM, hay diferentes opciones de implementación, pero vamos a hacerlo por medio de JAXP.

Por un lado, TransformerFactory obtiene un objeto de la factoría de transformadores. Una vez obtenida una instancia de transformador, se pueden definir propiedades para su posterior serialización. Se debe indicar el Document de donde se obtiene el XML (clase Source) y también donde se va a escribir por medio del objeto de tipo Result.

Ejemplo de utilización del API JAXP para la serialización del árbol DOM:

```
TransformerFactory transFactory = TransformerFactory.newInstance();

Transformer idTransform = transFactory.newTransformer();
idTransform.setOutputProperty(OutputKeys.METHOD, "xml");
idTransform.setOutputProperty(OutputKeys.INDENT,"yes");
Source input = new DOMSource(doc);

// Guardar XML a fichero
Result output = new StreamResult(new FileOutputStream("ficheroSalida.xml"));
idTransform.transform(input, output);

// Mostrar XML por pantalla
Result pantalla = new StreamResult(System.out);
idTransform.transform(input, pantalla);
```

Con esto se da por concluido lo más importante de la gestión de un árbol DOM.

⚠️ **Importante: Mirar los ejemplos App3.java y App4.java del proyecto adt2_ejemplos_dom.**