Unidad 6

XML

Acceso a datos

Índice

Introducción

En este tema estudiaremos el lenguaje XML y sus interacciones con otros formatos destinados, directa o indirectamente a modificarlo, validarlo o consultarlo.

En primer lugar, estudiaremos el lenguaje XML con el fin de crear una base, tras ello veremos los métodos de DOM y parser para modificar los documentos XML o serializarlos.

En segundo lugar, veremos los procesos de validación tanto empleando DTD como esquemas de XML.

En tercer lugar, podremos estudiar el proceso de binding de JAXB para la serialización y deserialización de documentos XML.

Finalmente, veremos XPath como un lenguaje de consulta y su funcionamiento.

Al finalizar esta unidad

- Conoceremos los conceptos de los documentos XML, desde su sintaxis a los elementos que contienen.
- Habremos estudiado los modelos DOM y su empleo como almacenamiento.
- Habremos descubierto los usos de parser para XML en DOM y SAX.
- Habremos analizados distintos métodos de validación para documentos XML, centrándonos de DTD y esquemas XML.
- Sabremos qué es la función binding que JAXB emplea y sus procesos de serialización y deserialización.
- Comprenderemos a realizar consultas con XPath.

6.1. El lenguaje XML

El lenguaje de marcas XML, *extensible markup language*, derivado de SGML, *standard generalized markup language*, considerado un lenguaje de etiquetado, es un metalenguaje semejante al HTML que surge para resolver los problemas de este a partir del W3C, World Wide Web Consortium. Usa marcas no predefinidas lo que da una gran libertad para definir nuestras propias etiquetas y poder trabajar elementos específicos. Este lenguaje se ha extendido creando una multitud basados en él como son XHTML, MathML, SVG, XUL, XBL, RSS o RDF.

La visualización del código XML puede hacerse desde el navegador o desde programas especializados, como son las hojas de estilo CSS o transformaciones XSLT, entre otras.

La particularidad del código XML se encuentra en el hecho de que es compatible con una multitud de programas, lo que facilita el cambio de información, la personalización y, sobre todo, la reutilización de código, lo que ahorrará mucho tiempo y esfuerzo.

6.2. Estructura de un documento de XML

La estructura de un documento XML es jerárquica arborescente, es decir surge de un único elemento padre o raíz, el XML tan solo permite poseer una única raíz, y los hijos o ramas que surgen de dicha raíz. A su vez estas ramas actúan como una pseudoraíz para nuevas ramas, como se verá en el ejemplo. La extensión de este tipo de documentos es .xml.

```
<?xml version="1.0" encoding="utf-8"?>
<raíz>
    <rama>
        <subrama> Hola </subrama>
    </rama>
</raíz>
```

Comúnmente al elemento superior se le denomina padre y al inferior hijo.

Podemos ver que las líneas de código cuentan con dos partes claramente separadas:

- En primer lugar, la primera línea de código, llamada Prólogo o encabezado, marca la información necesaria con respecto a la naturaleza del propio documento, como es la versión utilizada o el tipo de codificación usado, ya sea Unicode o siguiendo las normas ISO.
- En segundo lugar, podemos encontrar el resto del documento, con una raíz inicial junto al resto de ramas dependientes de él que forman el cuerpo del documento.

Es destacable dentro del documento la función de los elementos, estos son el conjunto de etiquetas de entrada y cierre más lo que contenga, ya sea texto, otros elementos, atributos, etc.

El nombre de un elemento, recogido en las etiquetas de entrada y cierre, puede contener caracteres alfanuméricos, pero los únicos signos de puntuación que puede soportar son el guion "-", el guion bajo "_" y el punto ".", nunca al inicio.

En el elemento origen (`<origen>Dalmacia</origen>`) podemos ver sus partes principales:

- **Etiqueta de apertura:** `<origen>`
- **Contenido (en este caso texto):** `Dalmacia`
- **Etiqueta de cierre:** `</origen>`

En el caso del elemento raza podemos ver que este mismo contiene otros cinco elementos, también llamados subelementos:

```
<raza>
    <nombre>dálmata</nombre>
    <origen>Dalmacia</origen>
    <color>blanco y negro</color>
    <peso_medio>24 kg</peso_medio>
    <altura_media>58 cm</altura_media>
</raza>
```

Pueden existir elementos vacíos, sin contenido, sobre todo en casos dende se regula una estructura fija, aunque posea etiqueta sigue considerándose vacío. Estos elementos se pueden expresar de dos maneras:

- `<Texto></Texto>`
- `<Texto/>`
- `<Texto n="12"></Texto>`
- `<Texto n="12"/>`

Atributos

Si un elemento contiene un elemento con una propiedad específica, como un valor asignado, se debe adjuntar en la etiqueta de apertura. La forman de añadirlo sería la siguiente: Nombre del elemento, espacio, característica, signo de igual "=" y por último el valor o características entre comillas, si hubiese más atributos se añadirían con un espacio entre ellos. Como un ejemplo podemos ver los siguientes elementos.

```
<Texto Tipo="12">Hola</Texto>
<Plantas categoría="flores">Rosa</Plantas>
<Plantas categoría="arbustos"/>
```

Estos atributos añaden información al XML, pero su abuso puede ser contraproducente.

Comentarios

Pueden introducirse en cualquier parte del documento, en una línea independiente, con el fin de aclarar cosas, servir de recordatorio, dar información a otros lectores, etc. La sintaxis usada es la misma que la que HTML usa, iniciando con "`<!--`" y terminando con "`-->`".

```
<?xml version = "1.0" encoding = "utf-8"?>
<Libro>
    <!--Completar las entradas vacías-->
    <Título></Título>
    <Precio> 18€ </Precio>
    <Editorial></Editorial>
</Libro>
```

6.3. DOM

Podemos definir *DOM* como la interfaz de programación diseñada para acceder a los documentos XML y poder modificar sus contenidos. Dentro de DOM el documento de XML se representa en su estructura arbórea, mostrando como nodos los componentes de la estructura.

En estos nodos veremos todos los elementos del documento XML y junto con los nodos tipo `#text` y `#text` sin contenidos, los cuales incluyen saltos de línea y espacios en blanco. Podemos ignorar estos nodos innecesarios e ignorables, ya que es posible construir un documento XML en una sola línea. Podemos encontrar todos los componentes de DOM en org.w3c.dom.

6.4. Parsers o analizadores sintácticos, serialización y deserialización

Parsers es un programa que permite comprobar la sintaxis de un lenguaje formal, como XML, Java o C++, además, es posible acceder los contenidos del documento una vez revisados los contenidos.

El proceso que se lleva a cabo durante el *parsing* se denomina deserialización ya que transforma el documento, secuencial, en un conjunto de datos no secuenciales, el proceso inverso se denomina serialización.

El empleo de DOM nos permite consultar un documento, así como modificarlo, lo que nos da un acceso rápido a los datos contenidos. Como contrapunto DOM no es capaz de manejar ingentes cantidades de información, por lo que si se desea transformar uno de estos se recomienda emplear otro *parser* que se adapte más a esta situación, como es SAX.

6.5. DOM con Java

Podemos crear un DOM no solo a partir del *parsing* de un XML, sino que también es posible su creación directa como un documento en blanco que podemos rellenar nosotros mismos., aunque debemos tener en cuenta que DOM requiere de diversos elementos para funcionar.

- `Document`: documento en XML.
- `Node`: nodo de un documento en XML.
- `Element`: representa un elemento.
- `NodeList`: representa una lista de los distintos nodos para la búsqueda y recuperación de estos.
- `NamedNode;ap`: el equivalente de NodeList, pero para atributos.

Algunos de los métodos que emplean estos elementos para la consulta son los siguientes:

	Métodos de consulta para element y node	
	Método	**Funcionalidad**
Document	`String getXmlEncoding()`	Devuelve la codificación empleada.
	`String getXmlVersion()`	Devuelve la versión empleada.
Node	`Short getNodeType()`	Devuelve el tipo de nodo, con valores como `DOCUMENT_NODE`, `ELEMENT_NODE` o `TEXT_NODE`.
	`String getNodeName()`	Devuelve el nombre del nodo, ya sea elemento, con su nombre; texto, con `#text`; o atributo, con su nombre.
	`String getNodeValue()`	Devuelve el valor del nodo, ya sea texto o atributo.
	`NodeList getChildNodes()`	Devuelve una lista de los nodos hijos.
	`NamedNodeMao getAttributes()`	Devuelve los atributos de un nodo element como un `NamedNodeMap` o `null` si no lo es.
	`Node getParentNode()`	

6.5.1. Parsing DOM

Podemos encontrar tanto *parser* DOM como SAX dentro del paquete de javax.xml.parsers.

Podemos emplear DocumentBuilder para implementar DOM creando una instancia del proceso con la que poseerá la clase `DocumentBuilderFactory`, mediante el método `newInstance()`. Es posible crear una nueva instancia a partir de la instancia creada esta tomará como clase `newDocumentBuilder`. Existen diversos métodos por defecto para los *parsers* en la clase `DocumentBuilderFactory`, como son:

Métodos de DocumentBuilderFactory	
Método	**Funcionalidad**
`Static DocumentBuilderFactory newInstance()`	Crea una instancia de clase.
`DocumentBuilder newDocumentBuilder()`	Devuelve un *parser* DOM.
`Void setIgnoringComments(boolean ignoreComm)`	Ignora los comentarios, en su defecto estos se incluyen como nodos en DOM.
`Void setIgnoringElementContentWhitespace (boolean whitespace)`	Ignora los textos vacíos en el modo de validación.

DocumentBuilder posee diversos métodos para crear desde cero documentos DOM, los cuales estarán vacíos, como son:

Métodos de la clase DocumentBuilder	
Método	**Funcionalidad**
`Abstract Document newDocument()`	Devuelve un nuevo documento vacío.
`Document parse(File f)`	Construye un documento DOM a partir de los contenidos expresados.
`Document parse(InputStream is)`	
`Document parse(String uri)`	

6.5.2. Creación de documentos con DOM

Una vez creado el documento podemos rellenarlo con los siguientes métodos de Document y Element, con el fin de crear un fichero de XML a través de DOM.

Métodos de Document para la creación de nodos en DOM	
Método	Funcionalidad
`Element createElement(String Xx)`	Crea un nuevo elemento y se le asigna el nombre indicado.
`Text createTextNode(String Xx)`	Crea un nodo de tipo de texto.

Métodos de Element para la creación de atributos	
Método	Funcionalidad
`Void setAttribute(String Xx, String Xx)`	Añade un atributo con el nombre y el valor designado.

Es posible crear nuevos elementos para un documento DOM con métodos de Node, también es posible modificar elementos o valores. Una DOMException se lanzará si estas operaciones no funcionan.

Métodos de Node para añadir elementos en DOM	
Método	Funcionalidad
`Node appendChild(Node Xx)`	Añade un nodo hijo en último lugar y devuelve el nodo añadido.
`Node insertBefore(Node Xx, Node Xx)`	Añade un nodo hijo antes del nodo indicado.

6.5.3. Serialización de documentos DOM

Podemos serializar Dom en XML mediante la clase `Transformer`, `TransformerFactory`, empleando el método `newTransformer()`, también podemos obtener instancias de esta clase mediante el empleo del método `newInstance()`. Una vez iniciada la serialización podemos seleccionar sus propiedades con el método `setOutputProperty()`, las cuales se recogen en estándar XSL, y realizar la serialización con `transform()`.

Sintaxis:

```
DOMSource domsource = new DOMSource(doc);
Transformer transformer = TransformerFactory.newInstance(). newTransformer();
transformer.setOutputProperty(OutputKeys.METHOD, "xml");
…
Transformer.transform(domsource,sr);
…
```

Imagen 1. Ejemplo de sintaxis de documentación DOM

6.6. SAX

Parser DOM crea una copia en la memoria, pero si el documento es muy grande esto puede ser un problema, por lo que es preferible en estos casos emplear *parser* SAX. SAX no almacena una copia del documento en la memoria, sino que trabaja con él a través de eventos creados con una clase especial. Estas clases son manejadoras de eventos, y a su vez los eventos pueden ser manejadores de avisos o errores.

SAX trabaja mínimamente con el documento, limitándose a los eventos, generalmente a través de la clase `DefaultHandler`. Para poder implementar SAX es necesaria la clase `XMLReader`, que crea instancias con `XMLReaderFactory`, donde trabaja con el método `parse()`.

Métodos de XMLReader

Método	Funcionalidad

`void parse(inputSource input)`
 	
`void parse (String idSistema)`
	Realiza *parsing* en un documento XML. El identificador puede ser tanto un URI como un nombre de fichero.

`Void setContentHandler (ContentHandler handler)`	Asocia *parser* con un manejador de eventos, DefaultHandler por defecto.

Métodos de la clase DefaultHandler

Método	Funcionalidad

`void startDocument()`
 	
`void endDocument()`
	Inicio y fin del documento.

`void startElement (String uri, String NomLocal, String NomCualif, Attributes NomAtrib)`
 	
`void endElement(String uri,`
 	
`String NomLocal, String NomCualif)`
	Inicio y fin del elemento, con su nombre designado por NOMLocal.

`Void characters(char[] cars, int inicio, int longitud)`	Texto de Element. También se puede obtener mediante `String(cars, inicio, fin)`.

Ejemplo de un elemento en parser SAX

Parser SAX	XML

`startElement("usuario")`
 	
`characters("Antonio")`
 	
`startElement("registrado")`
 	
`endElement("registrado")`
 	
`endElement("usuario")`
	`<usuario>Antonio<registrado><usuario>`

Ejemplo de la sintaxis de SAX:

```
Class GEvent extends DefaultHandler
…
try{
XMLReader parserSAX = XMLReaderFactory.createXMLReader();
GEvent GEvent = new GEvent(system.out);
parserSAX.setContentHandler(GEvent);
parserSAX.parse(nombreFichero);
}
```

6.7. Validación de documentos de XML

Para poder trabajar adecuadamente con un documento XML este debe ser válido, para serlo tanto su estructura, su sintaxis y orden, como su contenido, los valores de sus elementos, deben ser correctos. Para poder comprobar la validez de un documento XML podemos emplear distintos mecanismos de validación como:

- **DTD, data type definition:** Anterior a XML, con el auge de este se adapto para poder validarlo, actualmente pierde popularidad.
- **Esquemas XML:** Más utilizado y potente se basa en esquemas XSD (XML Schema Definition).

6.7.1. Validación con DTD

Posee una funcionalidad inferior a los esquemas XML, centrándose en la estructura de los elementos, su orden y número de apariciones, y en su contenido, comprobando el tipo de contenido admitido en cada elemento y atributo.

6.7.2. Validación con esquemas de XML

Los esquemas XML permiten verificar documentos XML con mayor fiabilidad que DTD, mediante esquemas.

Podemos elaborar esquemas específicos para cada uno de nuestros proyectos XML:

```
<?xmlversion=“1.0”encoding=“UTF-8”?>
<xs:schema versión=“1.0”
Xmlns:XS=“http://www.w3.org/2001/XMLSchema”
elementFormDefault=“qualified”
<xs:element name=“usuarios”>
<xs:complexType>
     <xs:sequence>
<xs:elementname=“usuario”>
<xs:complexType>
     <xs:sequence>
         <xs:elementname=“Nombre”type=“xs:string”/>
         <xs:elementname=“PriApellido”type=“xs:string”/>
         <xs:elementname=“SegApellido”type=“xs:string”/>
     </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</schema>
```

Es posible especificar, dentro del propio documento XML el esquema deseado, para el anterior "`usuarios`" resultaría como:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<usuarios xmlns:xsi="http://www.w3.org/2001/XMLSche-
ma-instance"
xsi:noNamespaceSchemaLocation="usuarios.xsd"
</usuarios>
```

Podemos acceder al esquema, si se encuentra en la web, mediante su URI en lugar de su nombre.

Como alternativa al DTD, podemos encontrar el XSD, *XML schema definition*, el cambio se debe a que los archivos XSD permiten especificar tipos de elementos y añadir restricciones y atributos más complejos. Además de lo ya mencionado es destacable que su sintaxis es similar a XML y permite un mayor control sobre las ocurrencias de los elementos que los archivos DTD. Con él es más fácil realizar conversiones de datos.

Los XSD son archivos de texto plano con la extensión .xsd.

Creación y elementos de un XSD

La raíz del archivo XSD será "`<xs:schema>`" y el resto poseerá una estructura semejante a XML. Este archivo raíz puede contener uno de los siguientes atributos:

- `xmlns:alias`. Se especifica de dónde provienen los elementos y tipos usados, http://www.w3.org/ 2001 /XMLSchema". El alias puede ser xs o xsd, mientras se use el mismo en todo el documento no importa. **Ejemplo:**

  ```
  <xs:schema xmlns:xs="http://www.w3.org/2001/
  XMLSchema">
  ```

  Por lo tanto, las siguientes líneas siempre comenzarán con el prefijo "`<xs:`"

- `elementFormDefault`. Aclara si se debe añadir los valores "`qualified`" y "`unqualified`" a los elementos. Por defecto "`qualified`":

  ```
  elementFormDefault="qualified"
  ```

  Lo que indica que los elementos utilizados por el documento XML que aparezcan en este esquema deben estar calificados para el espacio de nombres.

- `attributeFormDefault`. Aclara si se debe añadir los valores "qualified" y "unqualified" a los atributos.

- `targetNamespace`. Se especifica de donde se extraen los nombres de los elementos definidos, "https://www.w3schools.com".

- `version`. Se especifica la versión del documento de esquema.

Los atributos siempre se separarán de su valor con un igual (=) y estos valores irán entre comas.

Un ejemplo, sin elementos, es el siguiente:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">
...
</xs:schema>
```

Elementos

En los archivos XSD el elemento schema siempre será la raíz, y por tanto se abrirá y cerrará con él. Este elemento contendrá los atributos ya mencionados.

Podemos encontrar elementos simples, que solo pueden contener texto, sin atributos u otros elementos subordinados. Que aparecerá como:

```
<xs:elemento nombre="X" tipo="Y">
</xs:elemento>
```

En este elemento la "X" sería el nombre del elemento e "Y" el tipo de datos que el elemento contiene.

Los tipos más comunes de elementos son los siguientes:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

```
<xs:element name="Inscripción" type="xs:date"/>
```

Se les puede añadir un valor predeterminado o fijo a los elementos simples. Este valor puede ser por defecto (default) que permite modificaciones o un valor fijo obligatorio (fixed) que no las permite.

```
<xs:element name="flores" type="xs:string" default="Rosa"/>
```

Los atributos son opcionales de forma predeterminada, salvo que se use el atributo "use" seguido por su valor, los atributos obligatorios:

Se pueden añadir restricciones a los atributos con la entrada `<xs:restriction base"atributo">`.

Puedes consultar más información sobre este asunto en: https://www.w3schools.com/xml/schema_facets.asp.

- **Indicadores de ocurrencias.** La cantidad de veces que puede o debe aparecer un elemento en el XML.
- **maxOccurs.** Siendo por defecto 1, especifica el número máximo de veces que un elemento puede aparecer, en caso de no poseer limitación su valor sería "*unbounded*".
- **minOccurs.** Siendo por defecto 1, especifica el número mínimo de veces que un elemento debe aparecer, en caso de no poseer limitación su valor sería "*unbounded*".
- **Elementos complejos.** Pueden ser elementos vacíos o contener elementos hijos, con contenido y contenido vacío, atributos, etc.

Los elementos vacíos aparecen de la siguiente forma:

```
<xs:element name="Texto"/>
```

La aparición de estos elementos se marca con "`<xs:complexType>`" el indicador de orden y los elementos que contenga. El indicador Orden especifica el orden de aparición de los distintos elementos. Puede ser de tres tipos:

- `<xs:sequence>`: todos los elementos hijos según la secuencia indicada.
- `<xs:choice>`: solo uno de los indicados.
- `<xs:all>`: todos sin importar el orden.

```
<xs:element name="flora">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Flores" type="xs:string" default="Rosa"/>
            <xs:element name="Arbustos" type="xs:string" default="Cardo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

6.8. Parsing con validaciones con Java

Es posible que el *parser* valide los documentos XML, con esquemas o DTD, mediante métodos de las clases DocumentBuilderFactory y SAXParserFactory, para DOM y SAX respectivamente.

Métodos de validación de las clases DocumentBuilderFactory y SAXParserFactory

Método	Función
`setValidating(boolean validating)`	Realiza una validación con DTD, el valor de `validating` puede ser `true`.
`Void setSchema(Schema schema)`	Especifica el esquema para la validación.
`Void setIgnoringElement-ContentWhitespace (boolean whitespace)`	Si se realiza la validación se ignorarán los textos vacíos y los elementos ignorables.

La validación provoca sus propios errores que no deben confundirse con errores de sintaxis y, por lo tanto, deben gestionarse apropiadamente:

Gestión de errores y avisos en la validación mediante métodos de la clase DefaultHandler

Método	Función
`void error(SAXParseException e) throws SAXException`	Error recuperable.
`void fatalError(SAXParse-Exception e) throws SAXException`	Error no recuperable que detiene el *parsing*.
`void warning(SAXParseEx-ception e) throws SAXException`	Aviso en el *parsing*.

DTD inicia la validación mediante setValidating(true), si se usan esquemas es necesario emplear setSchema(Schema schema). Con el gestor de eventos, hijo de DefaultHandler, como en *parsing* SAX, a este método se le añade SAXParseException con throws.

SAX es muy parecido, pero sustituyendo DocumentBuilderFactory y DocumentBuilder por SaxParserFactory y SAXParser. Otra particularidad es que necesitan emplear dos clases manejadoras distintas, una para los contenidos y otro para errores y eventos.

6.9. Binding con JAXB

JAXB permite traducir un documento XML en una colección de objetos o viceversa mediante un API, con un proceso de deserialización o serialización.

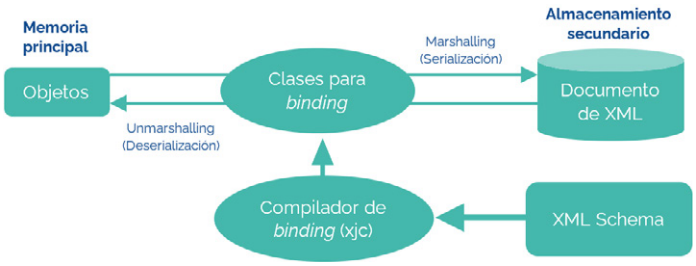

Imagen 2. Binding con JAXB

Para poder emplear JAXB es necesaria la creación de una colección de clases Java de los elementos y sus relaciones, interfaces y clases, creadas a partir de un esquema XML y un compilador *binding* xjc, con el que realizar los procesos de serialización y deserialización.

6.9.1. Esquemas de XML para Binding

Con el fin de que se registre correctamente para la posterior navegación es importante emplear una definición única con cualquier elemento, `<xs:element>` que pueda repetirse o poseer en su interior otros elementos y, por tanto, utilice `<xs:complexType>`, esto nos permitirá referenciarlo fácilmente y sin peligro de equivocaciones mediante la siguiente construcción `<xs:elementref="…">`.

```
…
<xs:elementname="usuario">
<xs:complexType>
      <xs:sequence>
…
<xs:elementref="usuario">
<xs:complexType>
      <xs:sequence>
…
```

6.9.2. Compilador de binding

Un compilador *binding* es un programa conocido como xjc, empleado desde los comandos o desde un IDE como NetBeans o Eclipse.

Para poder emplearlo desde el código se debe incluir la línea `xjc` con el nombre del proyecto con extensión, por ejemplo, `xjc usuario.xjc`.

Podemos emplear el compilador desde NetBeans, en un proyecto con XML y un fichero .xsd, abriéndolo desde File, New File, XML y JAXB Binding.

6.9.3. Clases generadas por el compilador de binding

Las clases que genera *binding* siempre poseen métodos `get` y `set` para poder asignar valores a los elementos `<xs:simpletype>`, en caso de que el elemento contenga otros elementos tan solo empleará `get`, para obtener una lista de clases de los otros elementos, pero `set` no se empleará.

Si se compilan se obtendrán diversas clases, destacando la del elemento Usuarios con tan solo un elemento get, el resto de los elementos con `<xs:complextype>` en parejas de `get` y `set` y un tercer tipo para los elementos `<xs:simpletype>`.

6.10. El lenguaje de consulta XPath

XPath, un lenguaje *path language*, con una sintaxis *path like* que permite buscar y seleccionar nodos en archivos con lenguajes XML. XPath posee más de 200 comandos predefinidos que podemos manejar para desarrollar la función deseada. Además de ser un elemento importante con XSLT XPath es el recomendado por W3C. Podéis encontrar toda la información necesaria en: https://www.w3schools.com/xml/xpath_intro.asp.

XPath puede trabajar con los lenguajes JavaScript, Java, XML Schema, PHP, Python, C y C ++, entre otros.

XPath analiza el documento XML con el fin de crear árboles de nodos de siete tipos:

- Nodo raíz.
- Elemento.
- Texto.
- Atributo.
- Namespace (espacio de nombres).
- Instrucción procesable.
- Comentario.

Así, con archivo XML como el siguiente, podemos realizar diversas acciones y ver sus diferentes partes. Usaremos este XML como ejemplo para el resto del tema.

```
<?xml version="1.0" encoding="UTF-8"?>

<Almacén>
     <películas>
        <título lang="es">Harry Potter</título>
        <director>J K. Rowling</director>
        <año>2005</año>
     </películas>
     <películas>
     …
     </películas>
     …
</Almacén>
```

XPath permite trabajar un documento con estructura arbórea, como se ve en la siguiente imagen:

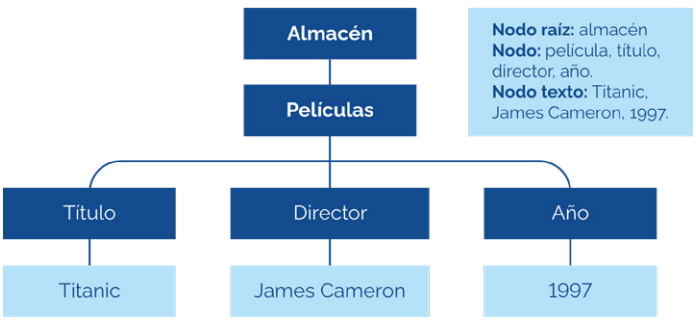

Imagen 3. Forma arbórea

Los nodos en XPath

Dentro de la estructura los nodos se relacionan entre sí:

- **Nodo.** Los valores intermedios.
- **Nodo raíz.** El nodo principal es aquel que inicia el archivo.
- **Valor atómico.** Elementos sin padres ni hijos.
- **Ítem o elemento.** Puede ser tanto un nodo como un valor atómico.

Las relaciones entre los nodos son las siguientes:

- **Padre (parent).** Es el nodo del que desciende otro, "películas" es padre de "título", "director" y "año".
- **Hijo (children).** Es el nodo que desciende de otro, "título" es hijo de "película".
- **Hermanos (siblings).** Nodos que comparten un mismo padre.
- **Antecesores (ancestors).** Es el nodo padre de otro nodo padre, el abuelo; así el elemento "Almacén" es el antecesor de "título".
- **Descendientes (descendant).** Es el nodo hijo de otro nodo hijo, "título" es descendiente de "Almacén".

Cómo seleccionar los nodos de un documento XML

El lenguaje XPath usa expresiones de ruta para navegar por el documento:

Expresiones de ruta	
Expresiones de ruta	**Descripción**
Nombre_del_nodo	Selecciona los nodos con el nombre especificado.
/	Selecciona el nodo raíz.
//	Selecciona los nodos que coinciden con el nodo actual.
.	Selecciona el nodo actual.
..	Selecciona el nodo padre del actual.
@	Selecciona atributos.

Ejemplos:

Ejemplos de expresiones de ruta	
Expresiones de ruta	**Descripción**
director	Selecciona los nodos "director"
/Almacén	Selecciona el nodo raíz
//título lang="es"	Selecciona los nodos «título lang="es"»
//director | //año	Selecciona los nodos "director" y "año"
//@lang	Selecciona el atributo "lang"

Así la siguiente ruta:

path = "/Almacén/películas/director"

Selecciona los nodos director hijos de películas, mostrándose solo estos de la siguiente manera:

- director: James Cameron
- director: …2
- director: …3

Utilizando predicados para mejorar las búsquedas

Los predicados permiten extraer un nodo concreto de un conjunto, se escriben entre corchetes.

Predicados XPath	
Expresiones	**Significado**
`/Almacén/películas[1]/director`	Se selecciona el director del nodo de películas 0, en este caso James Cameron. En ocasiones, dependiendo del navegador, los nodos se empiezan a contar desde el 0.
`/Almacén/películas[last()]/director`	El director de la última película.
`/Almacén/películas[last()-1]/director`	El director de la penúltima película.
`//título[@lang="es"]`	Solo los títulos de películas en español (atributo "es").
`/Almacén/películas[año>1990]/director`	Los directores de las películas del año 1991 en adelante.
`/Almacén/películas[position()<3]/director`	Los directores de las dos primeras películas.

Algunos de los operadores empleados por XPath son los siguientes:

Operadores usados por XPath		
Operador	**Descripción**	**Ejemplo**
|	Computa dos nodos simultáneos	//director | //año
+	Adición	1 + 1
-	Sustracción	3 - 1
*	Multiplicación	1 * 3
div	División	4 div 2
=	Igual	año=1997
!=	No igual	año!=1997
<	Menos que	año<1997
<=	Menos o igual que	año<=1997
>	Más que	año>1997
>=	Más o igual que	año>=1997
or	O	año=1997 o año=1996
and	Y	año>1990 y año<2000

Combinar varias rutas

Con el fin de combinar varios días podemos usar el operador "|" que nos permitirá unir varias rutas, por ejemplo:

path = "Almacén/películas/director | Almacén/películas/año"

Mostrando así todos los directores y después todos los años.

Comodines

Para seleccionar nodos desconocidos XPath cuenta con expresiones específicas llamadas comodines:

Comodines y sus ejemplos	
Comodín	**Descripción**
*	Selecciona cualquier nodo del elemento
@*	Selecciona cualquier atributo del nodo
node()	Selecciona cualquier nodo de cualquier clase
Ejemplos	
/películas/*	Selecciona todos los hijos de películas
//*	Todos los elementos del documento
//título[@*]	Selecciona el elemento título sin importar el atributo

6.11. El lenguaje XSL

XSL permite crear documentos diferentes a partir de un solo documento XML a través de diferentes plantillas.

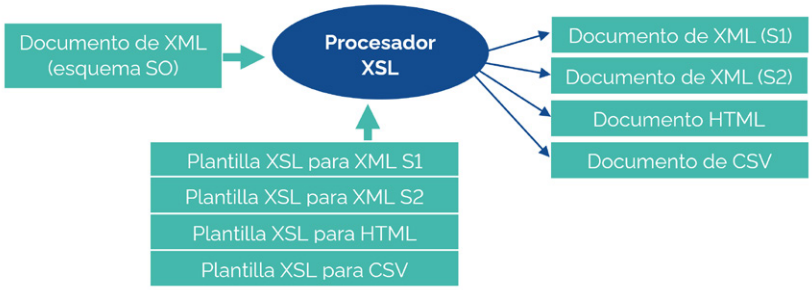

Imagen 4. Creaciones de documentos a partir de plantillas y un único documento XML

Los archivos XSL deben comenzar con la misma línea que los archivos XML.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Pero a diferencia de ellos deberán contener una segunda línea que los marcará como archivos XSL:

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

Mediante el documento XSL podemos controlar diversos aspectos del documento original, como ocurre con `<xsl:output>` con el atributo `method` puede poseer los valores html y text o `<xsl:strip-space elements="*"/>` que permite introducir texto sin separadores de relleno.

XSL puede emplear bytes, creando directamente ficheros binarios.

Podemos emplear `<xsl:template match="...">` con un XPath, con nodo raíz iniciado por "/", o `<xsl:apply-templates select="...">` que emplea XPath para el uso de otras plantillas. Existen muchas más opciones que no se verán en este tema.

Por último, debemos conocer que es posible emplear `Transformer` para aplicar una plantilla a un documento XML, con el fin de generar un nuevo fichero con las especificaciones deseadas.

www.universae.com