I.E.S. DOMINGO PÉREZ MINIK

CICLO FORMATIVO DE GRADO SUPERIOR:

"ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED (L.O.E.)"

MÓDULO:

Lenguajes de marcas y sistemas de gestión de información

(4 horas semanales)

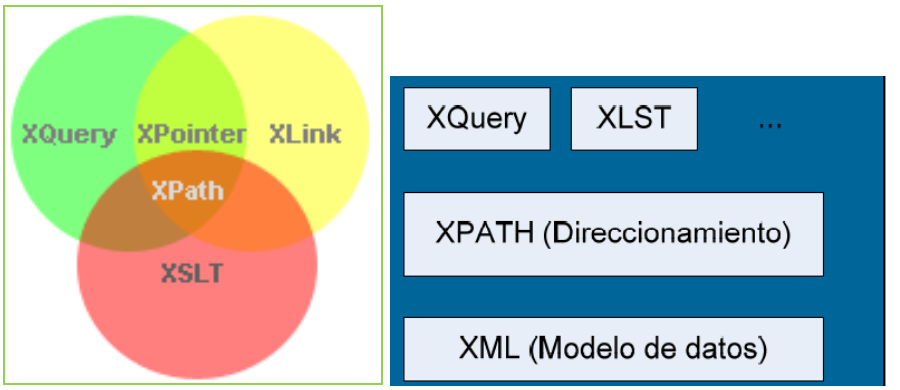
Índice

TEMA 7.- XPath.

1.- Introducción a XPath.

El término inglés *Path* podemos traducirlo como “ruta”, “camino” o “sendero”. Xpath es un lenguaje diseñado para acceder a partes de un documento XML y es la base para seleccionar mediante una ruta elementos concretos dentro de los datos. Con este lenguaje, podremos acceder tanto al texto, a los elementos, a los atributos y en general a cualquier información del fichero XML.

Por sí solo XPath no nos permitirá hacer nada más que acceder a partes de los datos, pero lo realmente interesante es que utilizando este lenguaje en combinación con otros como XSLT o XQuery podremos realizar transformaciones y operaciones complejas con los datos, transformar un documento en otro, aplicarle estilos a algunas partes, agregar marcas no existentes, ordenar contenido, etc. Por lo tanto para aprender estos lenguajes tendremos que conocer la sintaxis de XPath.



(Imagen tomada de w3schools.com) XPath es “la base” para otras tecnologías que veremos en temas posteriores.

2.- El árbol sintáctico.

Como ya sabemos los documentos XML se procesan con un programa llamado **parser o analizador sintáctico**. Este programa genera un árbol jerárquico de elementos. Este árbol comienza con un elemento raíz, que se diversifica a lo largo de los elementos que cuelgan de él y acaba en nodos hoja, que contienen solo texto, comentarios, intrucciones de procesamiento o incluso que están vacíos y sólo tienen atributos.

La forma en que XPath selecciona partes del documento XML se basa precisamente en la representación de este árbol. De hecho, los "operadores" de que consta este lenguaje nos recordarán la terminología que se utiliza a la hora de hablar de árboles en informática: raíz, hijo, ancestro, descendiente, etc.

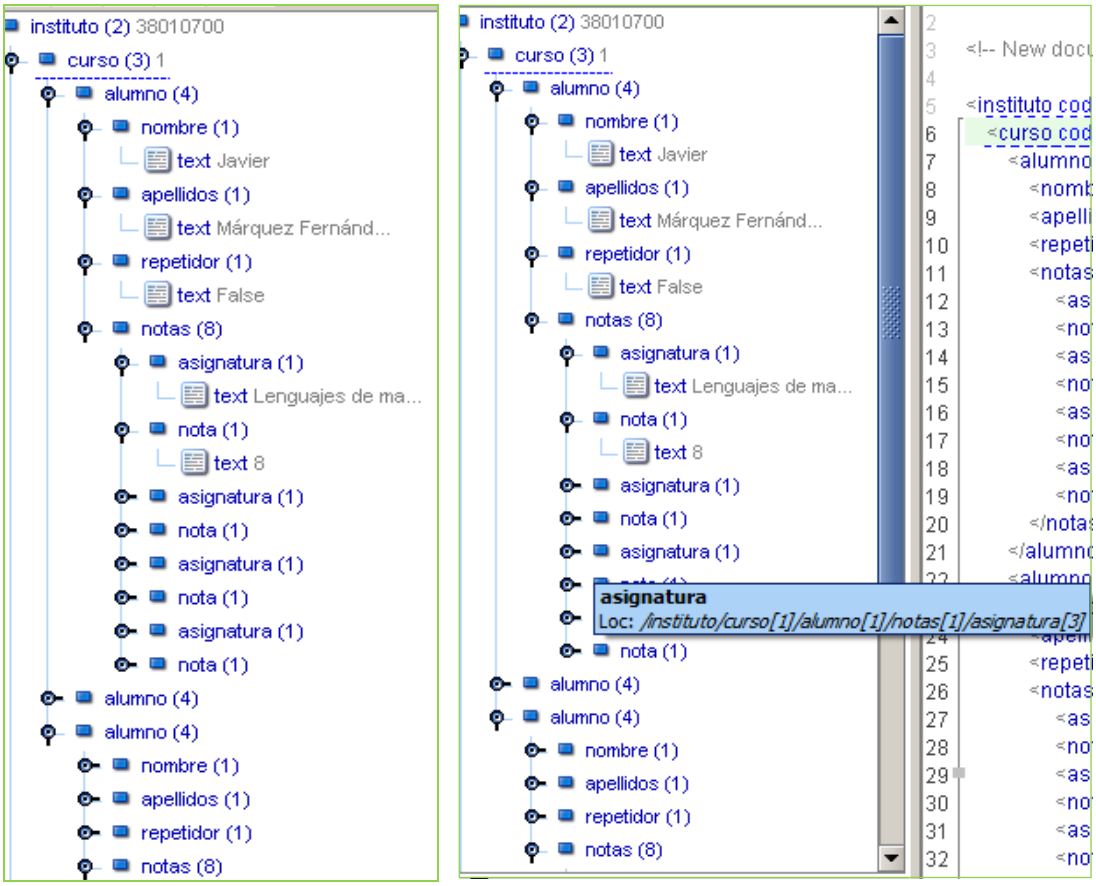
Un caso especial de nodo son los nodos atributo. Un nodo puede tener tantos atributos como desee, y para cada uno se le creará un nodo atributo. No obstante, dichos nodos atributo NO se consideran como hijos suyos, sino más bien como etiquetas añadidas al nodo elemento.

Para comprender bien los ejemplos en este tema trabajaremos con un documento XML llamado “alumnos.xml” con información de los alumnos del ciclo de Administración de Sistemas Informáticos en Red en nuestro instituto (mira un extracto):

```
<instituto codigo="38010700" nombre="IES Domingo Pérez Minik">
  <curso codigo="1" nombre="1º ASIR">
    <alumno cial="A18X111">
      <nombre>Javier</nombre>
      <apellidos>Márquez Fernández</apellidos>
      <repetidor>False</repetidor>
      <notas>
        <asignatura>Lenguajes de marcas</asignatura>
        <nota>8</nota>
        <asignatura>Redes</asignatura>
        <nota>7</nota>
        <asignatura>Fundamentos de hardware</asignatura>
        <nota>3</nota>
        <asignatura>Sistemas Operativos</asignatura>
        <nota>6</nota>
      </notas>
    </alumno>
    <alumno cial="A27M242">
      <nombre>María</nombre>
      <apellidos>Álvarez Pérez</apellidos>
      <repetidor>True</repetidor>
      <notas>
        <asignatura>Lenguajes de marcas</asignatura>
        <nota>5</nota>
        <asignatura>Redes</asignatura>
        <nota>5</nota>
        <asignatura>Fundamentos de hardware</asignatura>
        <nota>7</nota>
        <asignatura>Sistemas Operativos</asignatura>
        <nota>6</nota>
      </notas>
```

```
</alumno>
```

Con este fichero xml podemos observar directamente en el Editix el árbol sintáctico generado:



(Árbol generado) → Si dejamos el ratón sobre un elemento cualquiera del árbol aparece la "ruta" con sintaxis de Xpath → /instituto/curso[1]/alumno[1]/notas[1]/asignatura[3]

Como ves en las imágenes anteriores el programa Editix nos señala la ruta XPath de cualquier elemento del árbol. Lo que aprenderemos en este tema es la sintaxis para generar rutas de XPath para los distintos componentes de un fichero XML.

3.- Los nodos del árbol.

Llamamos nodo a cada una de las partes de las que consta el árbol sintáctico. Típicamente los nodos coinciden con los atributos, pero no siempre. Hagamos pues una clasificación de los tipos de nodos:

Tipos de nodos:

- Nodo raíz.
- Nodo elemento.
- Nodo de atributo.
- Nodo de comentario.
- Nodo de instrucción de procesamiento.
- Nodo de espacio de nombres.

3.1.- Nodo Raíz:

Se identifica por /. Es el nodo del que parte todo el árbol del documento y está justo encima del primer elemento de nuestro fichero xml. En nuestro ejemplo está por encima del nodo “instituto”. Luego una cosa es el nodo raíz, y otra el elemento principal del fichero xml, no debemos confundirlos.

3.2.- Nodo Elemento:

Habrá un nodo elemento por cada etiqueta (o elemento) de nuestro fichero xml. Todos los nodos elemento tendrán un único nodo padre, que es el elemento que lo encierra (salvo el elemento principal, cuyo padre es el nodo raíz).

Los nodos elemento tienen a su vez hijos, que podrán ser otros nodos elemento, nodos de texto, nodos comentario y nodos de instrucciones de procesamiento.

Los nodos elemento también tienen propiedades tales como su nombre, sus atributos e información sobre los "espacios de nombre", si éstos están utilizándose.

Una propiedad interesante de los nodos elemento es que pueden tener identificadores únicos (para ello deben ir acompañados de un DTD que especifique al atributo como de tipo ID). El uso de IDs permite referenciar a dichos elementos de una forma mucho más directa.

3.3.- Nodos texto:

Aquí entendemos por texto a cualquier cadena que no esté dentro de los símbolos de etiqueta < y >. Típicamente la información suele estar en un nodo de texto. En nuestro fichero de ejemplo:

<asignatura>Lenguajes de marcas</asignatura> → tenemos un nodo elemento llamado asignatura que es **padre** de un nodo texto que contiene “Lenguajes de marcas”.

Un nodo texto no tiene hijos, es decir, los distintos caracteres que lo forman no se consideran hijos suyos.

3.4.- Nodos atributo:

Como ya hemos indicado, los nodo atributo no son tanto hijos del nodo elemento que los contiene como etiquetas añadidas a dicho nodo elemento. Cada nodo atributo consta de un nombre, un valor (que es siempre una cadena) y un posible "espacio de nombres".

Aquellos atributos que tienen valor por defecto asignado en el DTD se tratarán como si el valor se le hubiese escrito en el documento XML (aunque no fuera así). Al contrario, no se crea nodo para atributos no especificados en el documento XML, y con la propiedad #IMPLIED definida en su DTD. Tampoco se crean nodos atributo para las

definiciones de los espacios de nombre. Todo esto es normal si tenemos en cuenta que no es necesario tener un DTD para procesar un documento XML.

3.5.- Nodos comentario y de instrucciones de proceso:

Aparte de los nodos indicados, en el árbol también se generan nodos para cada comentarios (recuerda que en XML se escriben con <!—texto -->) y para cada intrucción de procesamiento (recuerda que en XML se escriben con <? Instrucción ?>). Al contenido de estos nodos se puede acceder con la propiedad string-value.

4.- Expresiones de localización.

En XPath a cada instrucción se le llama "expresión", e indica un camino dentro de la estructura del árbol. El resultado de la misma podrá ser un nodo concreto o un conjunto de nodos.

Las expresiones pueden incluir desde operaciones (veremos el conjunto de operadores en XPath) hasta llamadas a funciones más o menos complejas. Sin embargo el tipo más común de expresión en XPath se llama "**location Path**" (ruta de localización). La sintaxis de los "location Path" es similar a la utilizada en sistemas Unix y Linux para movernos por el árbol de directorios y subcarpetas en el sistema de archivos:

Ejemplo: **/instituto/curso/alumno/nombre** → es un location Path que devolverá **un conjunto de nodos**, en este caso los formados por TODOS los nodos elemento <nombre> de nuestro fichero. Esto tendremos que tenerlo en cuenta, muchas veces un location Path hace referencia no a un único elemento sino a todos los elementos del árbol que cumplen esa ruta de localización.

En el ejemplo que acabamos de ver tendríamos que leer la ruta como:

"Todos los nodos "nombre" que cuelguen de cualquier nodo "alumno" que a su vez cuelgue de cualquier nodo "curso" que a su vez cuelgue del nodo principal "instituto".

Si se diera el caso de que no hay en nuestro fichero ningún elemento que cumpla con este location Path se devolvería una lista vacía. En otro caso se devolverá una lista de referencias (o apuntadores) a cada nodo que cumplió con los requisitos de la localización.

PRÁCTICA 01.- Abre en Editix el fichero alumnos.xml. Abre el panel de Xpath (menú view → Windows → XPath view). Prueba en la parte superior la ruta /instituto/curso/alumno/nombre desde el elemento raíz y observa en la parte inferior el resultado de la ruta.

Rutas de localización: (utilizan el símbolo /)

- Serán absolutas si empiezan por / (que denota el elemento raíz).
- Serán relativas si no empiezan por /, en cuyo caso se aplican desde el nodo contexto.

Habrás observado que puedes ejecutar la ruta desde el nodo raíz del árbol (“from root”) o bien desde el nodo actual (“from current”). El nodo actual en XPath se llama **nodo contexto** y es importante en determinadas aplicaciones de XPath.

Cuando utilicemos XPath dentro de otras aplicaciones como XSLT y XQuery las rutas de localización se aplicarán siempre desde el **nodo contexto**.

5.- Predicados XPath .

Los predicados se incluyen dentro de las expresiones XPath utilizando los corchetes. Sirven para refinar o acotar el conjunto de nodos resultante. Nos permitirán realizar muchas tareas como por ejemplo seleccionar los nodos que tengan un valor concreto en un atributo, seleccionar sólo el primer nodo que cumpla una expresión, etc.

Ejemplo: /instituto/curso[2]/alumno[1]/nombre

Aquí hemos usado dos predicados. Mostraremos el nodo nombre del **primer** alumno que esté dentro del **segundo** curso.

Ejemplo: /instituto/curso[@nombre="1º ASIR"]/alumno/nombre

Estamos utilizando un predicado que filtra los cursos y sólo utiliza aquellos que tengan un atributo llamado nombre cuyo valor sea “1º ASIR”.

Veamos primero más conceptos básicos antes de pasar a describir la sintaxis que se puede emplear en los predicados.

PRÁCTICA 02.- Prueba en Editix y sobre nuestro fichero alumnos.xml las siguientes rutas:
/instituto/curso[2]/alumno[2]/apellidos
/instituto/curso[@nombre=”2º ASIR”]/alumno[2]/apellidos
/instituto/curso[@nombre=”2º ASIR”]/alumno[2]/notas/asignatura
/instituto/curso[@nombre=”2º ASIR”]/alumno[2]/nota
¿Por qué no devuelve ningún resultado la última ruta? ¿Cómo podríamos ver las notas del segundo alumno de 2º de ASIR?

Predicados: (utilizan los símbolos de corchetes [ ])

- Sirven para acotar o filtrar nodos en los resultados.

PROBAR NUESTROS EJEMPLOS TRANSFORMANDO EL XML:

Hasta el momento sólo hemos probado nuestras expresiones utilizando el panel XPath view del Editix. No obstante vamos a chequear el uso de nuestras expresiones creando un fichero de XSLT que aplique la ruta XPath que deseemos evaluar aplicándosela a nuestro fichero XML. En el siguiente tema de la asignatura estudiaremos esta potente tecnología que es XSLT, por el momento sólo debes hacer lo siguiente:

1º - Agrega el fichero XSLT a alumnos.xml, colocando esta línea debajo de la versión de xml (2ª línea del fichero):

```
<?xml-stylesheet type="text/xsl" href="transformar.xsl"?>
```

2º - Ahora crea el fichero "transformar.xsl" copiando el siguiente código:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">
 <html>
 <head>
        <title>Prueba de XPATH</title>
 </head>
 <body>
        <H1>Resultado de la consulta XPATH:</H1>
        <pre>
        <xsl:apply-templates select="…nuestra ruta …/text()"/>
        </pre>
 </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Este fichero XSLT aplica una ruta XPath sobre un documento xml y lo transforma en un documento de HTML. Lo único que tendrás que hacer en cada práctica será cambiar lo escrito en verde y sustituirlo por la ruta que quieras probar. Bastará con que abras el fichero xml en un navegador web. En el próximo tema ya aprenderemos a manejar estas transformaciones para crear ficheros HTML con tablas, estilos y una apariencia más correcta. Ahora sólo aparecerán los valores seleccionados uno detrás de otro.

PRÁCTICA 03.- Prueba a modificar con un bloc de notas el fichero Transformar.xsl para ver los resultados de las siguientes rutas:
/instituto/curso[2]/alumno[2]/apellidos
/instituto/curso[@nombre="2º ASIR"]/alumno[2]/apellidos
/instituto/curso[@nombre="2º ASIR"]/alumno[2]/notas/asignatura
Nota: sustituye las comillas dobles por comillas simples (la que está en la tecla de la interrogación).

6.- Los ejes XPath .

Antes de ver la sintaxis y el potencial de los ejemplos XPath debemos hablar de los ejes, que son parte también de las rutas de localización. Hasta ahora hemos hablado de las rutas de localización estableciendo la siguiente sintaxis:

/elemento1/elemento2[predicado1][predicado2]/elemento3

En esta ruta se produce una evaluación de izquierda a derecha. Es decir, se analiza si existe “elemento 1” y se construye un **conjunto de nodos** que lo cumple. Por cada nodo en este conjunto se analiza si existe un hijo que sea “elemento2” y se construye otro **conjunto de nodos**. Después encontramos “predicado1” que lo que hace es descartar del conjunto los que no lo cumplen. Después aplicamos el mismo razonamiento descartando los que no cumplen el “predicado 2”. Sobre el conjunto de nodos resultante se mira si existe un hijo llamado “elemento 3” y se genera otro **conjunto de nodos.** Así es como funcionan las expresiones de XPath.

El eje es otra parte más de una expresión XPath, y se escribe seguido de dos veces el carácter “dos puntos”:

eje :: elemento1/elemento2[predicado1][predicado2]/elemento3

Los ejes existentes son los siguientes:

- **child::** Hijos directos (pero no atributos)
- **parent::** Padre directo (si existe)
- **descendant::** Descendientes
- **ancestor::** Antecesores
- **following-sibling::** Hermanos posteriores
- **preceding-sibling::** Hermanos precedentes
- **following::** Posteriores (excluyendo descendientes)
- **preceding::** Precedentes (excluyendo antecesores)

- **self::** El nodo de contexto
- **ancestor-or-self::**
- **descendant-or-self::**

Ejes de tipo especial (no elementos):

- **attribute::** Atributos del nodo de contexto
- **namespace::** Espacios de nombres del nodo de contexto

El eje sirve para crear el primer **conjunto de nodos** que cumple la ruta de localización, y permite que seleccionemos partes del árbol sintáctico concretas. Cuando nosotros hemos empleado la ruta /instituto/curso/alumno/nombre en realidad estábamos aplicando el eje /**child::**, que es el eje que se aplica por defecto (es decir, en aquellas expresiones que se escriben sin eje).

/instituto/curso/alumno/nombre equivale a child::instituto/curso/alumno/nombre.

Como ya sabes este eje sirve para seleccionar los hijos en el árbol sintáctico.

Prueba en la ventana XPath view del Editix la siguiente expresión:

/child::instituto/child::curso/child::alumno → /child:: equivale a escribir sólo la barra /

Observa el resultado. Observa también que se ha seleccionado la versión 2.0 de XPath. Este lenguaje tiene dos versiones:

XPath 1.0 → se desarrolló junto con la tecnología XSLT. Los resultados de las expresiones pueden ser nodos, valores booleanos (true/false), números o cadenas.

XPath 2.0 → se desarrolló junto con la tecnología XQuery. Soporta todo lo que soportaba la versión 1.0 pero incluye el uso de ejes y además los resultados de las expresiones pueden ser de cualquier tipo definido en XML Schema.

En realidad algunos ejes muy utilizados tienes una forma abreviada de escritura. Por ejemplo /child:: puede escribirse directamente usando **la barra /**. Para utilizar el eje de atributos podemos hacer /attribute:: o bien usar el símbolo @. Aquí mostramos las abreviaturas más utilizadas en XPath 1.0:

- child:: es el eje por omisión
- attribute:: se abrevia '@'
- '.' equivale a "self::node()"
- '..' equivale a "parent::node()"
- '//' equivale a "/descendant-or-self::node()/"

PRÁCTICA 04.- Usando transformar.xsl genera el fichero practica04.html que contenga todos los nombres de los alumnos.
Realiza esto cargando transformar.xsl en Editix → en la ventana "XML Data Source" asigna el fichero alumnos.xml. Para producir el fichero XML utiliza el menú XSLT/XQuery → opción "Transform a document with this XSLT".

Abre el fichero html para comprobar el resultado.

TRABAJO CON LOS EJES MÁS UTILIZADOS:

EJE Attribute:

Podemos utilizar la forma attribute:: o bien basta con utilizar una **@**. Si lo que queremos es filtrar elementos por el valor de un atributo tendremos que incluirlo dentro de un predicado (práctica 6):

PRÁCTICA 05.- Ahora genera el fichero practica05.html que contenga todos ciales de los alumnos.
¿Qué expresión Xpath se necesita? Quita /text() de la expresión en el .xsl

PRÁCTICA 06.- Ahora genera el fichero practica06.html que contenga todas las notas del alumno con cial A95M600.
¿Qué expresión Xpath se necesita?

EJE Descendant:

- Se especifica poniendo una doble barra: // (o con su forma larga: descendant::).

Sirve para seleccionar TODOS los nodos que descendiendan del conjunto de nodos contexto. Es decir, no solo los hijos de los nodos contexto, sino también los hijos de los hijos, y los hijos de estos, etc.

Ejemplo: seleccionar todos los alumnos del instituto:

/instituto/curso/alumno → usando el eje child

/instituto//alumno → usando el eje descendant

PRÁCTICA 07.- Ahora genera el fichero practica07.html que contenga todos los apellidos de todos los alumnos. Utiliza el eje descendant.

PRÁCTICA 08.- Genera el fichero practica08.html que contenga todos los nombres de los alumnos de 2º de ASIR. Utiliza el eje descendant.

PRÁCTICA 09.- Genera el fichero practica09.html que contenga todos los ciales de los alumnos de 2º de ASIR.

PRÁCTICA 10.- Genera el fichero practica10.html que contenga toda la información de texto del alumno con cial A95M600.

EJE Self:

- Sirve para seleccionar el nodo contexto actual. Podríamos escribir self:: o bien utilizar la forma abreviada que es el **carácter punto**. Lo utilizaremos cuando estemos trabajando con XSLT porque por el momento sólo estamos aplicando rutas desde la raíz.
- Como ejemplo imagina que el nodo contexto actual es el curso de 1º de DAI y queremos listar todos los nombres de alumnos. Si hacemos: //alumnos/nombre estaríamos partiendo de la raíz, con lo que se mostrarían todos los alumnos del fichero. En este caso tendríamos que hacer ./alumnos/nombre → o bien self::/alumnos/nombre.

EJE Parent:

- Sirve para seleccionar el nodo padre del nodo contexto actual. Podríamos escribir parent:: o bien **dos puntos seguidos**, que es su forma abreviada.
- Imagina que el nodo contexto es el primer alumno del fichero. Si hacemos /alumno/.. estaríamos seleccionando el nodo padre, que es curso. Si hacemos /alumno/../.. estaríamos seleccionado el abuelo, que es instituto. Es decir, permite subir de nivel por el árbol sintáctico.

PRÁCTICA 11.- Genera el fichero practica11.html que contenga el nombre del curso donde está el alumno con cial A95M600.

EJES Ancestor:

Ancestor permite seleccionar todos los nodos que están por encima del nodo actual, es decir, el padre, el padre del padre, etc. No tiene forma abreviada, lo usaremos como ancestor::

Es lo contrario a descendant.

PRÁCTICA 12.- Utilizando el eje ancestor genera el fichero practica12.html que contenga el nombre del curso donde está el alumno con cial A90R112.

7.- Otros elementos en XPath .

Dentro de las expresiones XPath podemos tener los siguientes símbolos:

- **Un nombre**: devuelve todos los nodos de elemento que tienen ese nombre.

  Ejemplo: /instituto/curso → devolverá tres nodos, cada uno de ellos correspondiente a un curso.

- **Un asterisco (*)**: devuelve todos los nodos que sean elementos, atributos o espacios de nombre que estén en el eje seleccionado. No selecciona comentarios ni instrucciones de proceso ni el texto de los elementos.

  Ejemplo: /instituto/curso/alumno/* → devolverá todos los nodos que son hijos de todos los alumnos, esto es, todos los nombres, todos los apellidos y todos los nodos "notas", así como los hijos de éstos.

PRÁCTICA 13.- Genera el fichero practica13.html comprobando el resultado de la expresión /instituto/curso/alumno/*.

- **node()**: devuelve cualquier tipo de nodo que esté en el eje seleccionado. Se diferencia del asterisco en que devolverá también nodos de comentarios y de instrucciones de proceso. También incluirá los textos de los elementos.
- **text():** devuelve sólo el texto de los elementos en el eje seleccionado. Lo hemos utilizado casi en todos los ejemplos en el fichero XSLT, de esta forma obteníamos las cadenas de texto en el fichero .html resultante.

  Tanto node() como text() pueden utilizarse dentro de los predicados.

PRÁCTICA 14.- Práctica avanzada: genera el fichero practica14.html que imprima el nombre de aquellos alumnos que son repetidores.
Nota: utiliza los ejes necesarios.

- **comment():** devuelve sólo los nodos del árbol que son comentarios, en el eje seleccionado.
- **processing-instruction():** sólo devuelve los nodos correspondientes a instrucciones de procesamiento en el eje seleccionado.

Cuando trabajemos con XSLT utilizaremos esta sintaxis de forma más completa.

8.- Operadores y funciones en XPath .

XPath también permite utilizar una serie de funciones para, por ejemplo, seleccionar el primer nodo de un conjunto de nodos, o bien el último. Además de las funciones también se incluye un conjunto de operadores para realizar tareas sencillas como la comparación el and lógico, etc. Tanto las funciones como los operadores se aplican **dentro de los predicados**, es decir, dentro de corchetes. Veamos algunas formas de trabajar con predicados:

Uso de un predicado:

PRÁCTICA 15.- genera el fichero practica15.html que imprima el nombre de todos los alumnos que tengan atributo cial (da igual el valor del cial).

8.1.- Operadores booleanos:

También se llaman operadores lógicos. Siempre devuelven un valor booleano (cierto o falso). Son el and (significa "y"), el or (significa "ó") y la función not().

Se aplican a expresiones que se escriben entre paréntesis:

(expresión1) **and** (expresión2) → será cierto sólo en los casos en los que se cumple expresión1 y también se cumple expresión2. Es decir, deben cumplirse las dos expresiones a la vez.

(expresión1) **or** (expresión2) → será cierto en los casos en los que se cumple expresión1 o bien si se cumple expresión2. Es decir, sólo es necesario que se cumpla alguna de las dos expresiones.

not(expresión1) → será cierto cuando NO se cumple expresión1.

Veamos un ejemplo con el uso de varios predicados juntos. Los predicados pueden sucederse uno tras otro y en este caso actúan como si estuvieran conectados por un AND.

PRÁCTICA 16.- genera el fichero practica16.html que imprima el nombre de todos los alumnos de 1º de ASIR que tengan atributo cial (da igual el valor del cial). Hazlo como sabes sin utilizar operadores.

Nota: prueba a hacer el mismo ejercicio conectando dos predicados de las siguientes formas:

1.- usando dos predicados juntos:

```
//curso[@nombre='1º ASIR'] [alumno[@cial]] //nombre/text()
```

2.- usando el operador and:
//curso[(@nombre='1º ASIR') **and** (alumno[@cial])] //nombre/text()

Nota: observa que al usar el operador **and** sólo usamos un único predicado y las distintas operaciones las encerramos entre paréntesis.

PRÁCTICA 17.- genera el fichero practica17.html que imprima todos los nombre de los alumnos salvo del que tiene cial=A18X111.

8.2.- Funciones de cardinalidad.

Las funciones de cardinalidad permiten filtrar nodos de un conjunto dependiendo de su posición en este conjunto. Las posiciones comienzan en 1. Así, si queremos mostrar toda la información del segundo alumno de 1º de DAI haríamos:

/instituto/curso[@nombre='1º DAI']/alumno[2]//text()

O lo que es lo mismo, dado que 1º de DAI es el tercer curso de nuestro fichero podríamos hacer:

/instituto/curso[3]/alumno[2]//text()

También sería equivalente utilizar la función **position**:

/instituto/curso[position()=3]/alumno[position()=2]//text()

También existe la función **last()** para devolver el último nodo de un conjunto de nodos, así como la función **id()** que permite filtrar y quedarnos con un nodo que tenga atributo un atributo id concreto. Esta última función, ejemplo de uso id("382008"), funcionaría siempre que nuestro fichero XML haya sido validado por un DTD y hayamos marcado el atributo de tipo ID.

PRÁCTICA 18.- genera el fichero practica18.html que muestre toda la información del último alumno de 2º de ASIR.

Para la práctica 19 necesitarás usar algún otro operador, así que hagamos un repaso completo de los operadores soportados en XPATH:

- **Operadores aritméticos**:
  - +, -, *, div, mod [un espacio debe preceder a '-']
- **Operadores de comparación**:
  - =, !=, <, >, <=, >= [igualdad con =, no ==]

Muchos de estos operadores los utilizaremos cuando estemos trabajando con XSLT en el siguiente tema.
*→ multiplicación DIV → división MOD → resto de una división.
!= → distinto

PRÁCTICA 19.- genera el fichero practica19.html que muestre nombres de los alumnos del fichero, salvo aquellos que sean los últimos de su curso.

PRÁCTICA 20.- genera el fichero practica20.html que muestre el nombre del penúltimo alumno de 1º de DAI.

8.3.- Otras funciones.

Existen muchas otras funciones en XPath (algunas de ellas sólo aplicables en la versión 2.0), pero el uso de las mismas se comprenderá mucho mejor cuando estemos utilizando XSLT. Por el momento veamos algunas y hagamos alguna práctica más o menos sencilla:

- **Funciones de conversión de valores:**

string(): convierte a cadena de texto
- Si es un número, se devuelve su representación como cadena de texto (es mejor utilizar la función format-number)
- Si es un booleano, se devuelve "true" o "false"
- Si es un node-set, se aplica al primer nodo
- Si es un nodo, se devuelve el valor-cadena

number(): convierte a un número flotante
- Si es una cadena, trata de parsearla
- Si es un booleano, true() es 1 y false() es 0
- Si es un node-set, se convierte primero a cadena mediante string()

boolean(): convierte a un booleano
- Si es un número, es true() si distinto de cero
- Si es un node-set, es true() si contiene algún elemento (muy útil en predicados)
- Si es una cadena, es true() si no es la cadena vacía

- **Funciones para trabajar con números:**

Matemáticas
- sum()
- floor()
- ceiling()
- round()

sum → realiza sumas.
floor → obtiene el entero inferior de un número decimal. Ej: floor(8,3) = 8
ceiling → obtiene el entero superior de un número decimal. Ej: ceilling(8,3) = 9

round → redondea un decimal a su entero más cercano. round(3,2) → 3, round(3,6) → 4 y por último round(3,5) → 6

PRÁCTICA 21.- genera el fichero practica20.html que muestre el nombre de aquellos alumnos cuya suma de notas sea mayor que 23.

- **Funciones para trabajar con cadenas de texto:**

- Manipulación de cadenas:
  - concat()
  - substring()
  - substring-before(), substring-after()
  - translate()
  - normalize-space()
  - string-length() [resultado: entero]
- Predicados sobre cadenas:
  - contains()
  - starts-with() [pero no hay ends-with()]

Cada una de estas funciones puede necesitar de un número diferente de argumentos. Los argumentos son los datos que se introducirán dentro de los paréntesis de la función. En el Editix presta atención a los mensajes de error.

PRÁCTICA 22.- genera el fichero practica22.html que muestre toda la información de los alumnos cuyo nombre comience por M.

PRÁCTICA 23.- genera el fichero practica23.html que muestre toda la información de los alumnos cuyo nombre comience por M y que sean repetidores.

PRÁCTICA 24.- genera el fichero practica24.html que muestre toda la información de los alumnos que tengan Pérez como alguno de sus apellidos.

- Relativas al node-set o el contexto:
  - position()
  - last()
  - count(node-set)
  - id() [devuelve un node-set]
- Propiedades de los nodos:
  - name()
  - local-name()
  - namespace-uri()
  - lang()

PRÁCTICA 25.- genera el fichero practica25.html que muestre toda la información de los alumnos que tienen menos de 4 materias.

- **document()** permite cargar un documento XML en tiempo de ejecución
- Esto permite extraer y combinar información de múltiples ficheros
- Advertencia: suele presentar problemas de rendimiento

9.- Prácticas finales.

En este apartado utilizaremos el fichero msn_data.xml, obtenido de un lector de noticias en RSS. Sobre este documento realiza las siguientes prácticas, utilizando rutas XPath y generando cada vez un fichero XHTML:

PRÁCTICA 26.- genera el fichero practica26.html que muestre los títulos de las noticias publicadas por el autor “chupidos”.

PRÁCTICA 27.- genera el fichero practica27.html que muestre el título de la primera noticia del autor “DUNALUNA”.

PRÁCTICA 28.- genera el fichero practica28.html que muestre el autor de las noticias que dentro del título tengan la palabra “Fukushima”.

I.E.S. DOMINGO PÉREZ MINIK

CICLO FORMATIVO DE GRADO SUPERIOR:

"ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED (L.O.E.)"

MÓDULO:

Lenguajes de marcas y sistemas de gestión de información

(4 horas semanales)