XPath

XPath se usa para navegar a través de los elementos y atributos de un documento XML. Es fundamental para otros muchos estándares como XSLT, XQuery, XPointer, XLink y otros.

XPath es un lenguaje declarativo. La programación declarativa es una forma de programación que implica la descripción de un problema dado, dejando la interpretación de los pasos específicos para llegar a dicha solución a un intérprete no especificado. La programación declarativa adopta, por lo tanto, un enfoque diferente al de la programación imperativa tradicional.

En otras palabras, la programación declarativa provee el "qué", pero deja el "cómo" liberado a la implementación particular del intérprete. Por lo tanto se puede ver que la programación declarativa tiene dos fases bien diferenciadas, la declaración y la interpretación. Otro ejemplo de lenguaje declarativo es SQL.

XPath es una sintaxis para definir partes de un documento XML. Sus expresiones pueden hacer referencia a un elemento ó conjunto de elementos de un documento.

La forma en que XPath selecciona partes del documento XML se basa precisamente en la **representación arbórea** que se genera del documento. 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 el siguiente documento XML:

```
<libro>

 <titulo>Dos por tres calles</titulo>

 <autor>Josefa Santos</autor>

 <capitulo num="1">
  La primera calle

  <parrafo>
        Era una sombría noche del mes de agosto...
   </parrafo>

  <parrafo destacar="si">
```

```
        Ella, inocente cual
      <enlace href="http://www.enlace.es">mariposa</enlace>
      que surca el cielo en busca de libaciones...
  </parrafo>

 </capitulo>

 <capitulo num="2" public="si">
  La segunda calle

  <parrafo>Era una obscura noche del mes de septiembre...</parrafo>

  <parrafo>
        Ella, inocente cual
        <enlace href="http://www.abejilla.es">abejilla</enlace>
        que surca el viento en busca del néctar de las flores...
  </parrafo>

 </capitulo>

 <apendice num="a" public="si">
  La tercera calle

  <parrafo>
        Era una densa noche del mes de diciembre...
  </parrafo>

  <parrafo>
        Ella, cándida cual
        <enlace href="http://www.pajarillo.es">abejilla</enlace>
        que surca el espacio en busca de bichejos para comer...
  </parrafo>
 </apendice>
</libro>
```

El árbol generado sería el siguiente:

```
/
|
+---libro
      |
      +---titulo
      |     |
      |     +---(texto)Dos por tres calles
      |
      +---autor
      |     |
      |     +---(texto)Josefa Santos
      |
      +---capitulo [num=1]
      |      |
      |      +---(texto)La primera calle
      |      |
      |      +---parrafo
      |      |     |
      |      |     +---(texto)Era una sombría noche ...
      |      +---parrafo
      |            |
      |            +---(texto)Ella, cual inocente mariposa...
      |
      +---capitulo [num=2]
             |
             +---(texto)La segunda calle
             |
             +---parrafo
             |     |
             |     +---(texto)Era una obscura noche ...
             +---parrafo
                   |
                   +---(texto)Ella, cual inocente abeja...
```

Tipos de nodos

Existen distintos tipos de nodos en un árbol generado a partir de un documento XML, a saber:

- raíz (root)
- elemento (elements)
- atributo (attribute)
- texto (text): caracteres del documento no marcados con alguna etiqueta
- comentario (comment) e instrucción de procesamiento (processing instruction): al contenido de estos nodos se puede acceder por medio de la propiedad *string-value*

A tener en cuenta:

Nodo Raíz
Se identifica por /. No se debe confundir el nodo raíz con el elemento raíz del documento. Así, si el documento XML de nuestro ejemplo tiene por elemento raíz a libro, éste será el primer nodo que cuelgue del nodo raíz del árbol, el cual es: /.

Insisto: / hace referencia al nodo raíz del árbol, pero no al elemento raíz del documento XML, por más que un documento XML solo pueda tener un elemento raíz. De hecho, **podemos afirmar que el nodo raíz del árbol contiene al elemento raíz del documento**.

Según lo dicho anteriormente, XPath usa expresiones de ruta para navegar a través de los documentos XML (al estilo de los sistemas de ficheros en los SSOO). Las expresiones más utilizadas al describir una ruta son:

Expresión	Descripción
Nombre	Selecciona todos los nodos hijo del nodo nombrado
/	Selecciona el nodo raíz
//	Selecciona los nodos del documento desde el nodo actual que cumplen la expresión, **independientemente de donde estén**
.	Selecciona el nodo actual
..	Selecciona el padre del nodo actual
@	Selecciona atributos

Los *Location Paths*

Expresiones

Una "instrucción" en lenguaje XPath se denomina **expresión**.

Dichas expresiones pueden incluir cierta variedad de operaciones sobre distintos tipos de operando. En nuestro caso, nos vamos a ceñir a dos tipos de operando: llamadas a funciones y *location paths* ("caminos de localización").

Un *location path* es la más importante de los tipos de expresiones que se pueden especificar en notación XPath. La sintaxis de un *location path* es similar a la usada a la hora de describir los directorios que forman una unidad de disco en Unix o Linux (y similar a la de los sistemas basados en MS-DOS y Windows, si exceptuamos la unidad de disco -C:, A:- y que las barras usadas son / en vez de las típicas \ de estos últimos sistemas operativos).
La siguiente expresión en XPath:

```
/libro/capitulo/parrafo
```

hace referencia a TODOS los elementos parrafo que cuelguen directamente de CUALQUIER elemento capitulo que cuelgue de CUALQUIER elemento libro que, finalmente, cuelguen del nodo raíz, /.

Hay que tener en cuenta que una expresión en XPath no devuelve los elementos que cumplen con el patrón que representa dicha expresión, sino que devuelve

una referencia a dichos elementos; es decir, una expresión XPath nos devuelve una lista de apuntadores a los elementos que encajan en el patrón. Dicha lista puede estar vacía o contener uno o más nodos.

Nodo contexto

Un *location path* siempre tiene un punto de partida llamado **nodo contexto**. Para entendernos, es como el directorio actual si nos referimos a un sistema de ficheros. Así, si estando en Unix, damos una orden *ls*, obtendremos los ficheros que existen en el directorio actual, mientras que si decimos ls /usr/bin, obtendremos el listado de los ficheros existentes en el directorio /usr/bin con independencia del directorio en que estemos colocados al dar la orden.
En los *location path* ocurre lo mismo. A menos que se indique un camino explícito, se entenderá que el *location path* parte del nodo que en cada momento se esté procesando.
El concepto de "nodo contexto" es imprescindible para comprender cómo se lleva a cabo la elección de los nodos que ajustan con el patrón indicado en el *location path*. Para explicar esto, veamos cómo actuaría un motor de evaluación en la expresión XPath siguiente:

```
/libro/capitulo/parrafo
```

- En primer lugar comienza por leer **/**, lo cual le dice que debe seleccionar el nodo raíz, independientemente del nodo contexto que en ese momento exista. En el momento en que el evaluador de XPath localiza el nodo raíz, éste pasa a ser el nodo contexto de dicha expresión.
- A continuación, el analizador leería **libro**, lo cual le dice que seleccione TODOS los elementos que cuelgan del nodo contexto (que atendiendo al párrafo anterior es el nodo raíz) que se llamen libro. En este caso solo hay uno, porque solo puede haber un elemento raíz.
- A continuación el analizador leería **capitulo**, lo cual le dice que seleccione TODOS los elementos que cuelgan del nodo contexto (que atendiendo al párrafo anterior es el nodo libro). En nuestro documento XML podemos ver cómo hay **dos elementos capitulo** colgando del elemento raíz libro. Por tanto, en estos momentos hay dos elementos que encajan con el patrón /libro/capitulo.
- El analizador continúa leyendo la expresión XPath que le hemos dado y llega a parrafo. Con ello le estamos diciendo que seleccione TODOS los elementos parrafo que cuelgan del nodo contexto...¡¡pero NO hay un nodo contexto, sino DOS!! El evaluador de expresiones lo va a recorrer uno por uno haciendo que, mientras evalúa un determinado nodo, ése sea el nodo contexto de ese momento.

En resumen, para localizar todos los elementos parrafo tal y como deseamos, se procesa el primer elemento capitulo y de él se extraen todos los parrafo que contenga. A continuación se pasa al próximo elemento capitulo del cual se

vuelven a extraer todos los de tipo parrafo que tenga... y así sucesivamente. El resultado final es un nuevo conjunto de nodos (o para ser más precisos, conjunto de punteros a nodo) que encajan con el patrón buscado.

Predicados

Los predicados se incluyen dentro de un *location path* utilizando los corchetes, como por ejemplo:

```
/libro/capitulo[@num="1"]/parrafo
```

Mediante el anterior *location path* estamos indicando que se escojan todos los elementos **parrafo** de todos los elementos **capitulo** que tengan un atributo llamado **num** al cual se le haya asignado el valor "1" (recordemos que en XML todos los atributos tienen valores de tipo cadena).
Atendiendo a nuestro ejemplo, solo hay un capitulo que cumpla dichas condiciones, por lo que solo los elementos **parrafo** que él contiene serán seleccionados.

Ejes

- Child: es el hacha utilizada por defecto. Se corresponde con la barra, / (aunque tiene una forma más larga que es: /child::).

Usando /	Usando /child::
Seleccionar todos los títulos de un libro	
/libro/titulo	/child::libro/child::titulo

- Attribute: se corresponde con el signo arroba, @ (o en su forma larga que es attribute::).

Usando @	Usando attribute::
Seleccionar el atributo 'num' que posean los elementos capítulo	
/libro/capitulo/@num	/libro/capitulo/attribute::num
Seleccionar todos los elementos hijo de los capitulos que posean el atributo 'public'	
/libro/capitulo[@public]/*	/libro/capitulo[attribute::public]/*
Seleccionar todos los elementos hijo de parrafo cuyo atributo 'destacar' sea igual a 'si'.	
/libro/capitulo/parrafo[@destacar="si"]	/libro/capitulo/parrafo[attribute::destacar="si"]

- Descendant: se corresponde con la doble barra, // (o en su forma larga que es /descendant::). Sirve para seleccionar TODOS los nodos que desciendan 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.

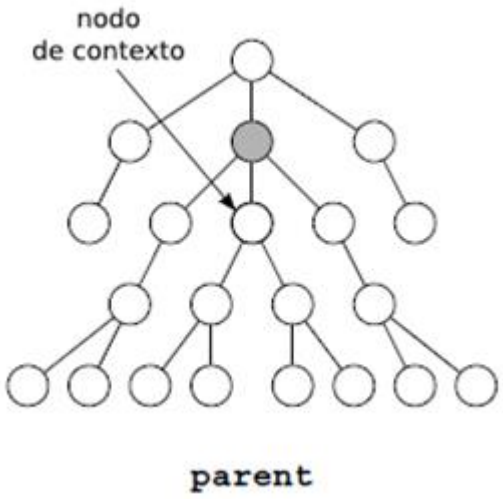
Usando //	Usando /descendant::
Seleccionar todos los párrafos de un libro	
/libro//parrafo	/libro/descendant::parrafo
Seleccionar todos los descendientes de parrafo que tienen un atributo 'href'	
//parrafo//*[@href]	/descendant::parrafo/ descendant::*[@href]
Ver el valor del atributo 'href' del caso anterior	
//parrafo//*[@href]/@href	/descendant::parrafo/descendant::*[@href]/@href
Ver todos los descendientes de capitulo	
/libro/capitulo//*	/libro/capitulo/descendant::*

- Self: Se especifica mediante el punto, .

Es muy útil pues sirve para seleccionar el nodo contexto. Por ejemplo, supongamos que deseamos seleccionar todos los parrafo descendientes del nodo contexto. No podemos escribir //parrafo, dado que seleccionaría todos los descendientes del nodo raíz. Por ello, la forma correcta es: .//parrafo

- Parent: se utilizan los dos puntos, ..

Usando ..	Usando parent::
Seleccionar todos los nodos que tienen algún hijo de tipo parrafo	
//parrafo/..	//parrafo/parent::*
Seleccionar todos los nodos capitulo que tiene algún hijo de tipo parrafo	
//parrafo/../../capitulo	//parrafo/parent::*/parent::*/capitulo
//capitulo/párrafo/..	



Ver: http://mauriciomatamala.net/LMSGI/xpath2.php

- Ancestor: no tiene abreviatura; hay que usarla tal cual ancestor::. Devuelve todos los elementos del cual, el nodo contexto, es descenciente

Usando ancestor::
Seleccionar todos los elementos que tienen entre sus descendientes algún parrafo
//parrafo/ancestor::*
Seleccionar todos los capitulos que tienen entre sus descendientes alguno con el atributo href
//*[@href]/ancestor::capitulo

Nodos TEST

Son funciones que ayudan a restringir la salida de una expresión XPath.

- Nodos TEST aplicables a todos los ejes:
  - * (asterisco): devuelve todos los nodos de tipo principal (elementos, atributos y espacios de nombres), pero no nodos de texto, de comentarios y de instrucciones de proceso.

Usando *
Seleccionar todos los nodos principales descendientes de los parrafo:
//parrafo/*

  - node(): devuelve todos los nodos de todos los tipos:

Usando node()
Seleccionar todos los nodos descendientes de los parrafos
//parrafo/node()

- Nodos TEST aplicables sólo a los nodos de contenido:
  - text().

Usando text()
Seleccionar el texto de todos los nodos parrafo:
//parrafo/text()
Seleccionar TODO el texto que cuelga de todos los nodos parrafo:
//parrafo//text()

  - comment (): cualquier nodo de tipo comentario.
  - processing-instruction (): cualquier nodo de tipo de instrucción de proceso, independientemente de su destino.
  - processing-instruction (destino): cualquier nodo de tipo de instrucción de proceso relativo al destino especificado.
  - processing-instruction (cursor): cualquier nodo de tipo de instrucción de proceso con el destino cursor.

Predicados

En los predicados se pueden usar funciones producir un valor boolean que indica si el nodo pasa el filtro o no.

Hay que tener en cuenta que la secuencia vacía y la cadena vacía se consideran false mientras que las secuencias con uno o más elementos o las cadenas no vacías se consideran true.

Los predicados se construyen mediante los operadores y las funciones de XPath.

- Predicados:

Predicados
Seleccionar todos los elementos que tengan el atributo 'num'
//*[@num]
Seleccionar todos los capitulos que tengan un parrafo que tenga algún elemento con atributo href
//capitulo[parrafo/*[@href]]
*Seleccionar todos los capitulo que tengan un parrafo que tenga algún elemento con atributo href **y** que ellos mismos (los capitulo tengan el atributo public a valor si:*
//capitulo[parrafo/*[@href]][@public='si']
//capitulo[(parrafo/*[@href]) **and** (@public='si')]
*Seleccionar todos los capitulo que tengan un parrafo que tenga algún elemento con atributo 'href' **o** todos los apendice:*
//capitulo[parrafo/*[@href]] | //apendice
Seleccionar todos los capitulo que no tengan el atributo public
//capitulo[not(@public)]
Seleccionar todos los párrafos que tengan algún atributo
//parrafo[@*]
Seleccionar todos los párrafos que no tengan ningún atributo
//parrafo[not(@*)]

- Predicados con funciones de cardinalidad:
  - position()

position()
Seleccionar el segundo capítulo
//capitulo[position()=2]

  - last()

last()
Seleccionar el último capítulo
//capitulo[last()]
Seleccionar todos los capítulos menos el último
//capitulo[not(position()=last())]
Seleccionar el penúltimo capítulo

```
//capitulo[last()-1]
```

- id(): sólo se podrá usar en aquellos ficheros XML que sean validados por un DTD en el que se especifique que el atributo id es único. Selecciona elementos por su identidad.

id()
Seleccionar los parrafo hijos del elemento con id="capitulo_1"
id("capitulo_1")/parrafo

Funciones

- función "normalize-space ()": elimina los espacios al principio y al final, así como también remplaza las secuencias de blancos por un solo espacio.

//BBB[@id='b1']
Devuelve la entrada marcada en azul
<AAA> <BBB id = "b1"/> <BBB name = " bbb "/> <BBB name = "bbb"/> </AAA>
//BBB[@name='bbb']
<AAA> <BBB id = "b1"/> <BBB name = " bbb "/> <BBB name = "bbb"/> </AAA>
//BBB[normalize-space(@name)='bbb']
<AAA> <BBB id = "b1"/> <BBB name = " bbb "/> <BBB name = "bbb"/> </AAA>

- función "count ()": cuenta el número de elementos seleccionados

//*[count(BBB)=2]
<AAA> <CCC> <BBB/> <BBB/> <BBB/> </CCC> <DDD> <BBB/> <BBB/> </DDD>

```
      <EEE>
         <CCC/>
         <DDD/>
      </EEE>
   </AAA>
```

```
//*[count(*)=2]
```

```
   <AAA>
      <CCC>
         <BBB/>
         <BBB/>
         <BBB/>
      </CCC>
      <DDD>
         <BBB/>
         <BBB/>
      </DDD>
      <EEE>
         <CCC/>
         <DDD/>
      </EEE>
   </AAA>
```

- función "name ()": devuelve los elementos con el texto indicado

```
//*[name()='BBB']
```

```
   <AAA>
      <BCC>
         <BBB/>
         <BBB/>
         <BBB/>
      </BCC>
      <DDB>
         <BBB/>
         <BBB/>
      </DDB>
      <BEC>
         <CCC/>
         <DBD/>
      </BEC>
   </AAA>
```

- función "starts-with ()": es verdadera cuando la cadena de caracteres pasada en primer argumento tiene como prefijo al segundo argumento:

```
//*[starts-with(name(),'B')]
```

```
   <AAA>
      <BCC>
         <BBB/>
         <BBB/>
         <BBB/>
      </BCC>
      <DDB>
         <BBB/>
```

```
      <BBB/>
    </DDB>
    <BEC>
      <CCC/>
      <DBD/>
    </BEC>
  </AAA>
```

- función "contains ()":es verdadera cuando la cadena del primer argumento contiene al segundo argumento:

```
//*[contains(name(),'C')]
  <AAA>
    <BCC>
      <BBB/>
      <BBB/>
      <BBB/>
    </BCC>
    <DDB>
      <BBB/>
      <BBB/>
    </DDB>
    <BEC>
      <CCC/>
      <DBD/>
    </BEC>
  </AAA>
```

FUNCIONES DE CADENA

- **starts-with (cadena-completa, subcadena)**

Ej. starts-with("Pepelui", "Pe") → true()
contains(cadena-completa, subcadena)
Ej: contains("aeIou", "eI") → true()

- **substring-before (cadena, subcadena)**

Ej: substring-before("DD/MM/AAAA", "/") → "DD"

- **substring-after (cadena, subcadena)**

Ej: substring-after("DD/MM/AAAA", "/") → "MM/AAAA"

- **substring (cadena, inicio [,longitud])**

Ej: substring("DD/MM/AAAA", 3) → "/MM/AAAA"

- **string-length (cadena)**

Ej: substring("DD/MM/AAAA", 3) → "/MM/AAAA"

- **normalize-space (cadena) → Elimina espacios extra**

FUNCIONES NUMÉRICAS

- **div**: realiza divisiones en punto flotante.

```
//BBB[position() mod 2 = 0 ]
```

```
<AAA>
   <BCC>
      <BBB/>
      <BBB/>
      <BBB/>
   </BCC>
   <DDB>
      <BBB/>
      <BBB/>
   </DDB>
   <BEC>
      <CCC/>
      <DBD/>
   </BEC>
</AAA>
```

- **mod**: calcula el resto entero de la división.

```
//BBB[position() mod 2 = 0 ]
```

```
<AAA>
   <BCC>
      <BBB/>
      <BBB/>
      <BBB/>
   </BCC>
   <DDB>
      <BBB/>
      <BBB/>
   </DDB>
   <BEC>
      <CCC/>
      <DBD/>
   </BEC>
</AAA>
```

- **floor**: calcula el más grande entero que no es mayor que el argumento
- **ceiling**: produce el mas pequeño entero que no es menor que el argumento

```
//CCC[ position() = floor(last() div 2 + 0.5) or position() = ceiling(last() div 2 + 0.5) ]
```

```
<AAA>
   <BBB/>
   <BBB/>
   <BBB/>
   <BBB/>
   <BBB/>
   <BBB/>
   <BBB/>
   <BBB/>
   <CCC/>
```

```
      <CCC/>
      <CCC/>
   </AAA>
//BBB[ position() = floor(last() div 2 + 0.5) or position() = ceiling(last() div 2 + 0.5) ]
<AAA>
  <BBB id="1"/>
  <BBB id="2"/>
  <BBB id="3"/>
  <BBB id="4"/>
  <BBB id="5"/>
  <BBB id="6"/>
  <BBB id="7"/>
  <BBB id="8"/>
  <CCC/>
  <CCC/>
  <CCC/>
</AAA>
```

ANEXOS

Eje	Descripción
	child::
.	self::
..	parent::
@	attribute::
.//	descendant-or-self/node()/
//	/descendant-or-self/node()/
*	all child elements of the context node
@*	all attributes of the context node
[n]	[position() = n]

XPath Operators

Operator	Description
and	Boolean AND
or	Boolean OR
=	Equals
!=	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
+	Addition
-	Subtraction
*	Multiplication
div	Division

Node Set Functions

Function	Description
last()	Returns the number of the number of items in the selected node set.
position()	Returns the position of the context node in the selected node set.
count()	Takes a location path as an argument and returns the number of nodes in that location path.
id()	Takes an id as an argument and returns the node that has that id.

String Functions

Function	Description
starts-with()	Takes a string and substring as arguments. Returns true if the string begins with the substring. Otherwise, returns false.
contains()	Takes a string and substring as arguments. Returns true if the string contains the substring. Otherwise, returns false.
substring-before()	Takes a string and substring as arguments. Returns the portion of the string to the left of the substring.
substring-after()	Takes a string and substring as arguments. Returns the portion of the string to the right of the substring.
substring()	Takes a string, start position and length as arguments. Returns the substring of length characters beginning with the character at start position.
string-length()	Takes a string as an argument and returns its length.
name()	Returns the name of an element.
text() *	Returns the text child nodes of an element.

Boolean Functions

Function	Description
boolean()	Takes an object as an argument. Returns true if: the object is a number greater than zero the object is a non-empty node-set the object is a string with at least one character.

Number Functions

Function	Description
sum()	Takes a node-set as an argument and returns the sum of of the string values of the node-set.
ceiling()	Takes a number as an argument and returns the rounded-up value.
floor()	Takes a number as an argument and returns the rounded-down value.
round()	Takes a number as an argument and returns the rounded value.