**XSLT**

En muchas ocasiones es necesario cambiar el formato de un documento XML. Por ejemplo, aunque dos organizaciones utilicen XML para almacenar los mismos datos, pueden emplear etiquetas con distinto nombre o en distinto orden. O en otros casos, puede ser necesario generar XML que fusione información procedente de distintos documentos (como un pedido en el que aparecen datos de un artículo y de un cliente). **XSLT** es un lenguaje específicamente diseñado para transformar documentos XML, cambiando su estructura, fusionando documentos o filtrando datos. En realidad, aunque el documento origen debe ser XML, el destino puede tener cualquier formato: XML, HTML, texto plano...

**1. Estructura de un documento XSLT**

Podemos considerar un documento XSLT como un programa, escrito con sintaxis XML, que especifica cómo procesar un documento XML para transformarlo en otro. Evidentemente, para ejecutar dicho programa necesitaremos un intérprete o un compilador de XSLT (lo más común es lo primero). J2SE (Java 2 Standard Edition o Java Standard Edition), incluye un intérprete XSLT denominado Xalan, cuyo uso veremos posteriormente.

El XSLT toma como entrada el documento a transformar en forma de árbol, y empieza a procesarlo por la raíz, descendiendo sucesivamente por los nodos. Un documento XSLT no es más que un conjunto de plantillas (**templates**) que especifican qué salida generar cuando el procesamiento alcanza un nodo determinado.

Por ejemplo, supongamos que se desea obtener un documento XML únicamente con los libros del documento "novedades.xml" que teníamos en el tema anterior. Además, de cada libro solo nos interesa el primer autor, que debe venir expresado con la etiqueta <autorPrinc> en lugar de <autor>. El siguiente es un documento XSLT que realiza la tarea:

<!-- las instrucciones de XSLT vienen asociadas con el espacio de nombres

adecuado -->

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

**<!--seguir procesando desde la raíz del árbol-->**

<xsl:template match="/">  
 <xsl:apply-templates/>  
 </xsl:template>

**<!-- ignorar la fecha -->**

<xsl:template match="fecha">  
 </xsl:template>

**<!--ignorar los discos -->**

<xsl:template match="disco">  
 </xsl:template>

**<!-- de los libros, procesar únicamente el título y el primer autor -->**

<xsl:template match="libro">  
 <libro>  
 <xsl:apply-templates select="titulo"/>  
 <xsl:apply-templates select="autor[position()=1]"/>  
 </libro>  
 </xsl:template>

**<!-- en los títulos, generar la misma etiqueta -->**

<xsl:template match="titulo">  
 <titulo>  
 <xsl:value-of select="."/>  
 </titulo>  
 </xsl:template>

**<!-- en los autores, generar la etiqueta autorPrinc -->**

<xsl:template match="autor">  
 <autorPrinc>  
 <xsl:value-of select="."/>  
 </autorPrinc>  
 </xsl:template>

</xsl:stylesheet>

Como puede verse, el documento sigue la sintaxis XML. Las etiquetas que son instrucciones propias del lenguaje XSLT vienen precedidas del prefijo de espacio de nombres xsl. Por supuesto este prefijo es arbitrario, podría ser cualquier otro mientras se asocie al espacio de nombres propio de XSLT (http://www.w3.org/1999/XSL/Transform).

El XSLT anterior puede ejecutarse mediante la clase org.apache.xalan.xslt.Process, incluida en J2SE y que es un intérprete XSLT en línea de comandos (el API XSLT también es accesible mediante programa, como se verá). Si se ejecuta dicha clase sin argumentos se verá una lista de los que admite. Tres de los más básicos son -IN (XML de entrada), -XSL (nombre del XLST) y -OUT (dónde se guardará el resultado, por defecto en la salida estándar). Por ejemplo

java org.apache.xalan.xslt.Process -IN novedades.xml -XSL trans.xsl

**2. Procesamiento de un documento XSLT**

El procesamiento se realiza de la siguiente manera: el "intérprete" XSLT tiene en cada momento un **nodo actual**en el árbol del documento XML a transformar. Además hay una **salida** a donde se irá enviando el resultado del procesamiento. Este comienza por la raíz, buscando algún **template** cuyo **match** coincida con el nodo actual. En ese caso, se ejecutan las instrucciones de dentro del *template*, que básicamente pueden ser dos (aunque hay muchas más en el lenguaje):

* **<xsl:apply-templates>**: significa que el intérprete XSLT debe descender un nivel más en el árbol, procesando por defecto todos los nodos hijos del actual. Mediante el atributo **select** se puede especificar si deseamos procesar un conjunto distinto de nodos.
* **<xsl:value-of>**: significa que el intérprete XSLT debe enviar a la salida el valor del nodo especificado mediante el atributo **select**.

En los atributos match y select se colocan expresiones con sintaxis similar a la utilizada en UNIX o MS-DOS para moverse por los directorios. Por ejemplo, "." representa el nodo actual, y "autor" un nodo de tipo autor hijo del nodo actual. En realidad estas expresiones son parte de un estándar XML llamado XPath, que veremos a continuación.

Cuando dentro de un *template* se pone texto o etiquetas que no pertenecen al espacio de nombres XSLT (en el ejemplo, no llevan el prefijo *xsl*), el intérprete xslt las copia en la salida.

**3.*Templates* por defecto**

En XSLT no es necesario escribir *templates* para procesar todos los nodos del documento XML original, ya que el estándar exige que el intérprete XSLT aplique una serie de *templates* por defecto para algunos casos (si el usuario no define *templates* para ellos). Estos casos son:

* **Nodos no-hoja del árbol**: el comportamiento por defecto del intérprete es bajar a procesar todos los hijos del nodo actual. Por ello, se puede considerar que el *template* del ejemplo que encaja con la raíz del árbol es superfluo. Esta es la razón, además, de que haya que definir *templates* vacíos para aquellos nodos que no deseamos seguir procesando.
* **Nodos hoja:** el comportamiento por defecto es imprimir el texto que contienen, como si se hubiera incluido una sentencia <xsl:value-of select="."/>
* **Comentarios e instrucciones de procesamiento:** el comportamiento por defecto es ignorarlos.

**4. XPath**

**XPath** es un lenguaje que permite referenciar una parte concreta de un documento XML, por ello se utiliza en XSLT para especificar en qué nodos se debe disparar una regla y a qué nodos se debe aplicar el procesamiento. Su sintaxis es similar a la que se usa en sistemas de ficheros estilo UNIX para expresar rutas de directorios (de ahí el **Path** del nombre). Como un documento XML puede verse como un árbol cuyos nodos son las etiquetas, el problema de especificar el acceso a determinadas partes del documento se puede ver como una versión más compleja de las rutas de directorios.

**4.1. Trayectorias**

La parte más importante del estándar XPath es la definición de **trayectorias** (*location paths*). En principio, las trayectorias parten por defecto del denominado **nodo de contexto,**que sería algo así como el equivalente al "directorio actual" en UNIX. En XSLT el nodo de contexto será el nodo que se está procesando actualmente. Si el primer símbolo de la misma es una barra, entonces se considera que la trayectoria parte del nodo raíz del documento. Por ejemplo, la expresión XPath

/novedades/libro

Se refiere a las etiquetas

<libro>

<titulo>Programación de aplicaciones web</titulo>

<autor>John McCarra</autor>

<autor>Frank Webini</autor>

<direccion>http://www.paw.org</direccion>

</libro>

<libro>

<titulo>Manolito Gafotas</titulo>

<autor>Elvira Lindo</autor>

<direccion>http://www.manolitogafotas.com</direccion>

</libro>

Como se ve, una expresión XPath puede referirse a un **conjunto** de nodos (a diferencia de una ruta de directorios, que siempre se refiere solo a uno).

En general, una trayectoria está compuesta por un conjunto de pasos separados por un símbolo /. El conjunto de nodos seleccionados en un paso es la base para el paso siguiente. Además, en cada paso se puede aplicar opcionalmente un **predicado** (función *booleana* entre corchetes) que restringe el conjunto de nodos seleccionados. Así, el formato general de una trayectoria es

paso1[predicado1]/paso2[predicado2]...

Por ejemplo, para seleccionar los autores del primer libro del documento se haría

/novedades/libro[position()=1]/autor

Donde la función position() devuelve el orden que ocupa un nodo en un conjunto de nodos seleccionados.

En la siguiente tabla se resumen algunos de los operadores que permiten expresar caminos.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operador** | **Significado** | **Ejemplo** | **Resultado** |
| // | nodo hijo o descendiente | /novedades//titulo (todos los títulos dentro de la etiqueta novedades - a cualquier nivel del árbol) | <titulo>O sea</titulo>  <titulo>Programación de  aplicaciones web</titulo>  <titulo>Manolito Gafotas</titulo> |
| \* | cualquier nodo | /novedades/\*/titulo | mismo que el anterior |
| @ | atributo | /novedades/disco/@genero | pop |
| . | el propio nodo |  |  |
| .. | nodo padre | /novedades/disco/titulo/.. | <disco genero="pop">  <titulo>O sea</titulo>  <interprete>  Enrique Iglesias  </interprete>  <direccion>  http://www.enriqueiglesias.com  </direccion>  </disco> |
| text() | nodo de texto | /novedades/disco/titulo/text() | O sea |
| processing-instruction() | instrucción de procesamiento | /processing-instruction() | <?xml-stylesheet  type="text/css"  href="novedades.css" ?> |
| comment() | comentario |  |  |

**4.2. Predicados**

Se utilizan para restringir el conjunto de nodos seleccionados. Se seleccionarán finalmente aquellos que cumplan el predicado, que no es más que una expresión entre corchetes que devuelve un resultado booleano. XPath dispone de un conjunto de operadores y funciones bastante extenso que podemos emplear a la hora de definir predicados.

**Operadores**

En XPath hay operadores de distintos tipos:

* **Relacionales:**=, !=, <, >, >=, <=
* **Aritméticos:**+, -, \*, div (en lugar de /)
* **Lógicos:** not(), and, or

**Funciones**

XPath tiene un gran número de funciones, algunas de las cuales se resumen en la tabla siguiente. En las siguientes funciones, el nodo de contexto es el nodo que se está evaluando en este momento, y la lista de nodos de contexto es el conjunto de nodos seleccionados.

|  |  |  |
| --- | --- | --- |
| **Función** | **Tipo de retorno** | **Descripción** |
| position() | número | devuelve la posición del nodo de contexto en la lista de nodos de contexto. Las posiciones empiezan por el 1. |
| last() | número | devuelve el número de nodos en la lista de contexto |
| count(conj-nodos) | número | devuelve el número de nodos en conj-nodos |
| local-name(conj-nodos) | cadena | devuelve la parte local (lo que viene detrás del prefijo de espacio de nombres) del primer nodo del conj-nodos. Si se llama sin argumentos opera sobre el nodo de contexto |
| generate-id(conj-nodos) | cadena | devuelve un identificador único para el primer nodo de conj-nodos. Si se usa sin argumento, lo devuelve para el nodo de contexto. |

Podemos utilizar operadores y funciones para definir predicados. Por ejemplo

|  |  |
| --- | --- |
| **Expresión** | **Significado** |
| /novedades/disco[@genero='pop'] | todos los discos de género pop |
| /novedades/disco[titulo='O sea']/interprete | el intérprete del disco titulado 'O sea' |
| /novedades/libro[position()=1]/autor[position()=last()] | el último autor del primer libro |

**4.3. Ejes**

Las expresiones XPath vistas hasta el momento utilizan lo que se conoce como **sintaxis abreviada**. XPath tiene operadores más complejos, que permiten seleccionar cualquier conjunto de nodos relativo a una posición en el árbol. Estos operadores son los **ejes**, que se resumen en la tabla siguiente.

|  |  |
| --- | --- |
| **Eje** | **Descripción** |
| ancestor | antepasado del nodo actual (padre, "abuelo, etc) |
| ancestor-or-self | los anteriores contando también al propio nodo |
| attribute | los atributos del nodo actual |
| child | los hijos del nodo actual |
| descendant | hijos, "nietos", etc, del nodo actual |
| descendant-or-self | los anteriores contando también al propio nodo |
| following | nodos que comienzan después que el nodo actual en el orden del documento (no se cuentan atributos ni declaraciones de espacios de nombres) |
| following-sibling | nodos que comienzan después que el actual en el orden del documento y además tienen el mismo padre ("hermanos siguientes") |
| namespace | espacio de nombres del nodo actual |
| parent | nodo padre del nodo actual |
| preceding | nodos que comienzan antes que el nodo actual en el orden del documento (no se cuentan atributos ni declaraciones de espacios de nombres) |
| preceding-sibling | nodos que comienzan antes que el actual en el orden del documento y además tienen el mismo padre |
| self | el nodo actual |

La forma de utilizar los ejes es poner su nombre seguido de los caracteres :: y una expresión que restrinja los nodos seleccionados. Por ejemplo:

/novedades/disco/titulo/following-sibling::\*

Seleccionaría todos (símbolo \*) los "siguientes hermanos" (following-sibling) del título de los discos. En este caso, al mismo nivel que título vienen interprete y dirección, con lo que el resultado sería

<interprete> Enrique Iglesias </interprete>

<direccion>http://www.enriqueiglesias.com</direccion>

En la notación abreviada que hemos visto hasta ahora hay formas de expresar más brevemente algunos de los ejes. No obstante, otros ejes no tienen "traducción abreviada"

|  |  |
| --- | --- |
| **Notación abreviada** | **Notación con ejes** |
| /novedades/disco | /child::novedades/child::disco |
| /novedades//titulo | /novedades/descendant::titulo |

En XSLT, el valor que puede tener un atributo match es más o menos el de una expresión XPath con notación abreviada (por ejemplo, no se pueden utilizar ejes). En los atributos select se puede colocar cualquier expresión XPath válida.

**5. Alterar la estructura del documento original**

Las instrucciones XSLT vistas hasta el momento permiten hacer filtrados, reordenaciones, o cambiar unas etiquetas por otras, pero en algunos casos puede ser necesario efectuar manipulaciones más complejas

**5.1 Generar atributos**

Supongamos que deseamos generar una etiqueta con un atributo que en el documento original era otra etiqueta. Por ejemplo, el siguiente *template* (que es **incorrecto**) intenta transformar la etiqueta "disco" en una etiqueta que contiene como atributo el título.

<xsl:template match="disco">

**<disco titulo="<xsl:value-of select="titulo">/>**

</xsl:template>

El problema con el *template* anterior es que no es XML bien formado, ya que una etiqueta no puede ir "incrustada" dentro de otra. Por ello, hay una forma alternativa de expresar el *select* para estos casos que consiste en ponerlo entre llaves.

<xsl:template match="disco">

**<disco titulo="{titulo}"/>**

</xsl:template>

**5.2 Generar etiquetas a partir de atributos**

También puede darse el caso inverso al anterior. Por ejemplo, supongamos que se desea transformar el atributo "genero" de los discos en una etiqueta. Para ello podemos utilizar la instrucción xsl:element, cuyo atributo name utilizamos para especificar el nombre de la nueva etiqueta.

<xsl:template match="disco">

<xsl:element name="{@genero}">

...

</xsl:element>

</xsl:template>

**6 XSLT y Java**

Como ya se ha comentado, J2SE incluye Xalan, un procesador XSLT desarrollado por el proyecto Apache. Xalan puede ejecutarse a través de línea de comandos y es accesible también mediante un API. Dicho API es bastante sencillo, como puede verse en el siguiente fragmento de código.

//El TransformerFactory sirve para crear un objeto Transformer  
TransformerFactory tf = TransformerFactory.newInstance();

//El Transformer sirve para aplicar el XSLT

Transformer t = tf.newTransformer(new StreamSource(

new FileInputStream("ejemplo.xsl")));

//Aplicar el XSLT del Transformer sobre un XML

t.transform(new StreamSource(new FileInputStream("ejemplo.xml")),

new StreamResult(System.out));

Como se ve, el objeto responsable de la transformación XSLT es el Transformer, que aplica una hoja XSLT sobre el documento XML que se desee. El mismo Transformer puede aplicar varias transformaciones sobre distintos documentos. La fuente de la transformación es un objeto de la clase StreamSource y el destino uno de la clase StreamResult.