UNIDAD 5 CONVERSIÓN Y ADAPTACIÓN DE DOCUMENTOS XML
PARTE II XSLT

1 XSLT

1.1 Introducción

XSLT (eXtensible Stylesheet Language Transformations) es un estándar de de W3C. Es un lenguaje funcional diseñado principalmente para transformar documentos XML en otras estructuras XML y no XML.

W3C empezó a desarrollar el XSL (eXtensible Stylesheet Language) expresión inglesa traducible como "lenguaje extensible de hojas de estilo") es una familia de lenguajes basados en el estándar XML que permite describir cómo la información contenida en un documento XML cualquiera debe ser transformada o formateada para su presentación en un medio.

Esta familia está formada por tres lenguajes:

- XSLT (siglas de Extensible Stylesheet Language Transformations, lenguaje de hojas extensibles de transformación), que permite convertir documentos XML de una sintaxis a otra (por ejemplo, de un XML a otro o a un documento HTML).
- XSL-FO (lenguaje de hojas extensibles de formateo de objetos), que permite especificar el formato visual con el cual se quiere presentar un documento XML, es usado principalmente para generar documentos PDF.
- XPath sintaxis (no basada en XML) para acceder o referirse a porciones de un documento XML.

1.2 ¿Qué es XSLT?

XSLT es la parte más importante de XSL. Se usa para transformar un documento XML en otro tipo de documento, que será reconocido en por un navegador, como HTML.

Con XSLT podemos añadir o eliminar elementos y atributos del fichero de salida XML. Podemos además ordenar elementos y tomar decisiones acerca de que elementos pueden ser o no visualizados.

XSLT se apoya en otras tecnologías XML para funcionar:

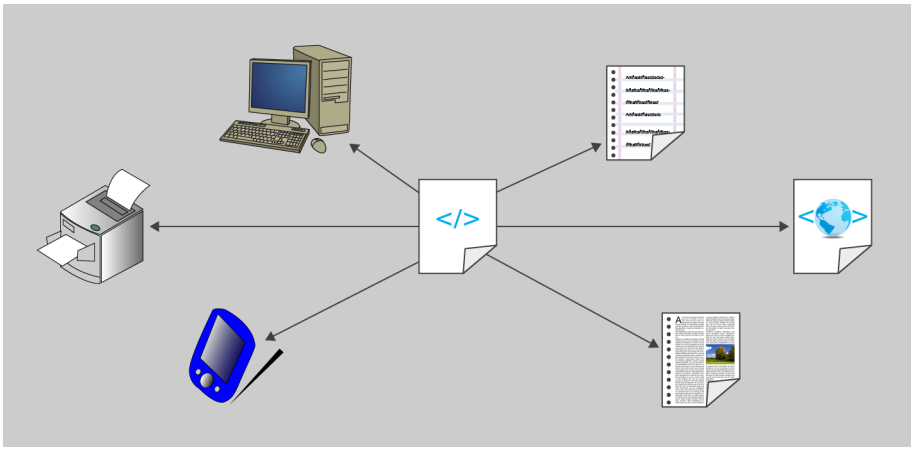
- XSLT usa XPath para buscar la información en un documento XML, es decir para navegar a lo largo del documento.
- Soporta XML Schemas para definir los tipos de datos.

Aunque existe la versión 2.0 y 3.0 de XSLT, la que más se utiliza es la 1.0.

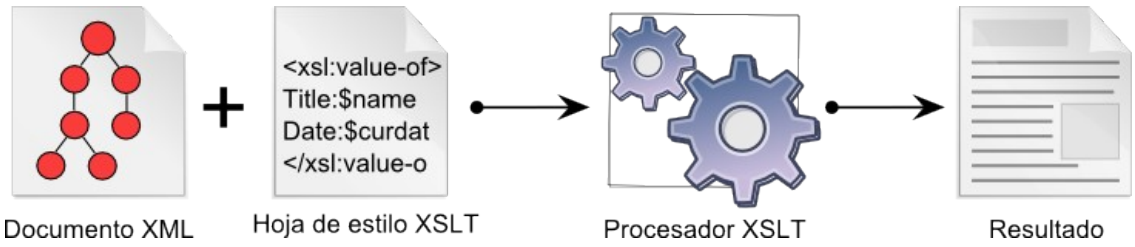
El resultado de la transformación puede ser:

- El mismo documento XML (mismas etiquetas), pero con filtros de salida.
- Otro documento XML (diferentes etiquetas)
- Otros documentos, que pueden ser XML o no
    - HTML
    - Text
    - WAP

Cada vez se utilizan más las plantillas XSLT para generar vistas personalizadas de un documento XML para ser visualizadas en diferentes destinos (impresoras, pantalla de ordenador, móvil, etc.) De este modo basta solo con tener el documento XML original, y el resto se crearán bajo demanda.



XSLT es un lenguaje de programación declarativo que permite generar documentos a partir de documentos XML, como ilustra la siguiente imagen:



Veamos cada uno de los elementos de la anterior imagen:

- El documento XML es el documento inicial a partir del cual se va a generar el resultado.
- La hoja de estilo XSLT es el documento que contiene el código fuente del programa, es decir, las reglas de transformación que se van a aplicar al documento inicial.
- El procesador XSLT es el programa de ordenador que aplica al documento inicial las reglas de transformación incluidas en la hoja de estilo XSLT y genera el documento final.
- El resultado de la ejecución del programa es un nuevo documento (que puede ser un documento XML o no).

¿Por qué se transforman los documentos XML?

Los documentos XML se transforman principalmente por dos motivos:

- Para adaptarlos a otro modelo de datos. XML se define para intercambiar datos entre aplicaciones. Cada aplicación puede tener diferentes modelos de datos (es decir, diferentes tipos de etiquetas)
- Para hacerlos más legibles: XML es demasiado complicado para leer. Necesitamos una transformación previa.

Otra cosa que tenemos que tener en cuenta es que un documento XSLT es un documento XML, así que como tal debe cumplir la sintaxis de XML.

Un documento XSL mezcla justo dos familias de etiquetas; aquellas que son de XSL y aquellas del marcado del documento de salida.

Ejemplo: Veamos como transformar un documento XML en un documento HTML, es decir vamos a mezclar etiquetas de XSL y de HTML

El documento XML es:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="catalogo.xsl"?>
<catalogo>
 <cd>
   <titulo>Empire Burlesque</titulo>
   <artista>Bob Dylan</artista>
   <pais>USA</pais>
   <compañia>Columbia</compañia>
   <precio>10.90</precio>
   <año>1985</año>
</cd>
</catalogo>
```

Creamos el documento XSL con la regla de transformación:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
  <body>
    <h2>Mi Colección de CD</h2>
    <table border="1">
     <tr>
       <th>Titulo</th>
       <th>Artista</th>
     </tr>

     <xsl:for-each select="catalogo/cd">
     <tr>
       <td><xsl:value-of select="titulo"/></td>
       <td><xsl:value-of select="artista"/></td>
     </tr>
     </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XSLT es un lenguaje declarativo. Por ello, las hojas de estilo XSLT no se escriben como una secuencia de instrucciones, sino como una colección de plantillas (template rules). Cada plantilla establece cómo se transforma un determinado elemento (definido mediante expresiones XPath).

Veamos algo del documento XSL, en el mismo tenemos:

- La declaración xml, propia de cualquier documento XML
- Los documentos XSL son documentos XML, y por tanto, deben cumplir las reglas de los documentos

XML. Esto significa que deben tener un elemento raíz. Así, tienen solo un elemento `<xsl:stylesheet>`, sus atributos indican la versión y el espacio de nombres correspondientes.

- Las reglas de plantilla se definen como elementos `<xsl:template>`. El atributo `match` indica los elementos afectados por la plantilla y contiene una expresión XPath.
- Las plantillas de salida son contenidos de elementos `<xsl:template>` y definen la transformación.

Por último, para enlazar el documento XML al documento XSL debemos añadir:

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

1.3 Elementos simples

Para los ejemplos siguientes vamos a utilizar el siguiente documento XML, que pongo a continuación:

```
<personas>
<persona>
        <nombre>Albert Einstein</nombre>
        <descripcion>Albert Einstein fue uno de los científicos más famosos del siglo XX, a
    él, entre otras cosas,  le debemos la Teoría de la Relatividad
        </descripcion>
</persona>
<persona>
         <nombre>Marie Curie</nombre>
         <descripcion>Pionera en el campo de la radioactividad, fue la primera  persona en  recibir  dos
     premios Nobel y la primera mujer en ser profesora en la Universidad de París
         </descripcion>
</persona>
<persona>
        <nombre>Nikola Tesla</nombre>
        <descripcion>Fue un inventor, ingeniero mecánico e ingeniero eléctrico y uno de los
         promotores más importantes del nacimiento de la electricidad comercial.
        </descripcion>
</persona>
</personas>
```

El documento personas.xsl asociado es:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
   <html>
    <head>
       <title>Información sobre
       <xsl:value-of select="count(/personas/persona)" /> científicos</title>
    </head>
    <body>
       <h3>Información sobre <xsl:value-of select="count(/personas/persona)" />
   científicos </h3>
       <br/>
       <xsl:apply-templates select="/personas/persona" />
    </body>
   </html>
  </xsl:template>
  <xsl:template match="persona">
     <h3><xsl:value-of select="nombre" /></h3>
     <p><xsl:value-of select="descripcion" /></p>
     <br />
  </xsl:template>
</xsl:stylesheet>
```

1.3.1 El elemento <xsl:template>

Usamos el elemento <xsl:template match> para seleccionar diferentes partes del documento XML. Por ejemplo para seleccionar el elemento raíz:

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

En el ejemplo anterior tenemos un caso de una plantilla vacía. El procesador no genera ningún resultado.

1.3.2 El elemento <xsl:apply-templates>

La etiqueta **<xsl:apply-templates>** es usada para aplicar una plantilla (template) sobre el elemento actual o sobre alguno de sus nodos hijos.

Si añadimos un atributo **select** a la etiqueta **<xsl:apply-templates>** esta procesará solo el nodo o nodos hijos que coincidan con el valor del atributo. También podemos usar el atributo select para especificar el orden en el que los nodos hijos serán procesados.

Una vez que se encuentra la etiqueta **<xsl:apply-templates>** se buscará alguna de las templates definidas en el documento XSL que coincida con la expresión **XPath** contenida en su atributo **match**.

En el ejemplo anterior cada vez que el procesador XSLT encuentre un nodo persona que corresponda con la trayectoria /personas/persona, el contenido de la plantilla es procesada y el contenido es añadido al árbol de resultados.

1.3.3 El elemento <xsl:value-of>

Normalmente lo que queremos es extraer información del árbol del documento. XSLT posee varias formas de usar la información contenida en el árbol.

El elemento <xsl:value-of> proporciona el valor de una parte del árbol. Tiene un atributo select, cuyo valor es una expresión XPath.

Extrae el contenido del nodo seleccionado. En el ejemplo de personas, el documento final contiene el nombre de las personas y la descripción, ya que la plantilla los genera con esta instrucción.

También se pueden extraer los valores de los atributos, utilizando @.

1.3.4 El elemento <xsl:copy>

El elemento **<xsl:copy> copia un nodo al árbol resultado, pero no copia los nodos descendentes**. Esto es útil si queremos cambiar la estructura de un elemento o su contenido, o añadir o eliminar atributos.

Veamos un ejemplo:

```
<Personas>
   <Persona>
     <Nombre>Luis</Nombre>
     <Apellido>Perez</Apellido>
  </Persona>
  <Persona>
    <Nombre>Eva</Nombre>
    <Apellido>Rojo</Apellido>
  </Persona>
  <Persona>
    <Nombre>Alan</Nombre>
```

```
    <Apellido>Herrera</Apellido>
  </Persona>
</Personas>
```

Para el ejemplo anterior, al que llamaremos "Personas2.xml", le aplicaremos la siguiente hoja de estilos xslt, a la que llamaremos "Personas2.xsl":

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
<xsl:template match="/">
  <Personas>
    <xsl:apply-templates select="/Personas/Persona" />
  </Personas>
</xsl:template>
<xsl:template match="Persona">
  <xsl:copy>
   <xsl:attribute name="Nombre">
    <xsl:value-of select="Nombre"/>
   </xsl:attribute>
   <xsl:attribute name="Apellido">
    <xsl:value-of select="Apellido"/>
   </xsl:attribute>
  </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

Después de probar el ejemplo, vamos a analizarlo. Como anteriormente, existe un template que tiene el atributo `match="/"`. En vez de crear elementos literales HTML, se añade el elemento literal `<Persona>`. El elemento `<xsl:apply-templates>` es usado con la ruta XPath `/Personas/Persona`.

Existe un template que tiene `match` a "Persona", lo que coincide con el valor del atributo "`select`" del elemento "`xsl:apply-templates`". Por tanto, para cada nodo "`Persona`" en el documento fuente, el template especifica como procesarlo.

El elemento `xsl:copy` es usado cuando el nodo de contexto es un nodo elemento Persona (en el segundo template). Por tanto, cada nodo elemento "Persona" es añadido al árbol resultado, pero sin copiar los nodos hijo.

En este punto, lo que tendríamos si el documento acabase en el primer elemento `xsl:copy` tendríamos el siguiente resultado:

```
<Personas>
 <Persona/>
 <Persona/>
 <Persona/>
</Persona>
```

El template emplea el elemento `xsl:attribute` para añadir un nuevo nodo atributo al elemento persona en el árbol resultado. En la línea `<xsl:attribute name="Nombre">` estamos diciendo que el atributo se llamará "Nombre". Y con la línea `<xsl:value-of select="Nombre"/>` indicamos que el valor del atributo será el contenido del elemento "Nombre".

Lo mismo ocurre con el atributo "Apellido".

Añadir elementos hijo

Si necesitamos seguir el camino inverso al del ejemplo anterior, podemos hacerlo con en el siguiente ejemplo. Partimos del resultado generado en el ejemplo anterior.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
<xsl:template match="/">
<Personas>
  <xsl:apply-templates select="/Personas/Persona" />
</Personas>
</xsl:template>
<xsl:template match="Persona">
  <xsl:copy>
    <xsl:element name="Nombre">
      <xsl:value-of select="@Nombre"/>
    </xsl:element>
    <xsl:element name="Apellido">
      <xsl:value-of select="@Apellido"/>
    </xsl:element>
  </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

Si observamos el código, se vuelve a repetir más o menos lo mismo que en el anterior. Tenemos un template con match a "/".

También tenemos un elemento "xsl:apply-templates" con select a "/Personas/Persona". Para este "xsl:apply-templates" hay un template (cuyo atributo match vale "Persona") que indica cómo procesar cada nodo elemento "Persona".

Con el primer <xsl:copy>, copiamos al árbol resultado el nodo elemento "Persona". Dejamos abierto el elemento <xsl:copy>, porque dentro colocaremos otros elementos.

Dentro del elemento <xsl:copy> creamos nuevos elementos, con el elemento <xsl:element>. A cada elemento le asignamos un nombre. Por ejemplo con "<xsl:element name="Nombre">" indicamos que se debe crear un un nuevo nodo elemento llamado Nombre.

A su vez, dentro de estos nuevos elementos, colocamos elementos de tipo xsl:value-of que recuperan el contenido de cada atributo. Por ejemplo, la línea <xsl:value-of select="@Apellido"/> recupera el valor del atributo "Apellido".

1.4 El elemento *<xsl:copy-of>*

Este elemento realiza una copia en profundidad. Es decir, copia un nodo, junto con todos sus nodos atributo y nodos descendientes.

```
<Envio>
   <Origen>Palencia</Origen>
   <Destino>Bilbao</Destino>
```

```
 <Direccion>
   <Calle>Lamentos,5</Calle>
      <Ciudad>Gotham</Ciudad>
      <Pais>ES</Pais>
      <CP>54321</CP>
 </Direccion>
</Envio>
```

La hoja de estilos asociada es:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
<xsl:template match="/">
  <Notificacion>
    <xsl:apply-templates select="/Envio/Destino" />
    <xsl:apply-templates select = "/Envio/Origen" />
    <xsl:apply-templates select="/Envio/Direccion" />
  </Notificacion>
</xsl:template>
<xsl:template match="Destino">
  <xsl:element name="Origen">
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>
<xsl:template match="Origen">
  <xsl:element name="Destino">
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>
<xsl:template match="Direccion">
  <xsl:copy-of select="." />
</xsl:template>
</xsl:stylesheet>
```

Como podemos ver, hemos utilizado el elemento "copy-of" dentro de un "template" aplicado al nodo "Dirección". Es decir, hemos copiado el nodo Dirección, así como a todos sus descendientes. De este modo podemos integrar toda esta zona del árbol en el árbol resultado.

1.5 Estructuras de control

Hasta ahora hemos visto que las transformaciones se realizan en un solo sentido, conforme se va instanciando cada template. Sin embargo, podemos crear diferentes templates y aplicar uno u otros en función de alguna condición.

1.5.1 El elemento *<xsl:if>*

El elemento if comprueba si una cierta condición booleana es verdadera o falsa. Así, una instrucción condicional puede determinar si una acción se lleva a cabo o no. Su sintaxis es:

```
<xsl:if test=condicion>
```

```
    …
</xsl:if>
```

Veamos un ejemplo, tenemos un documento con información sobre la memoria RAM de ciertos equipos de nuestra empresa y queremos generar a partir de él otro documento donde se informe de ciertos ordenadores demasiado antiguos:

```
<ordenadores>
   <equipo memoria=”1024”>B302PC01</equipo>
   <equipo memoria=”512”>B302PC02</equipo>
   <equipo memoria=”2048”>B302PC03</equipo>
   <equipo memoria=”512”>B302PC04</equipo>
</ordenadores>
```

El documento XSLT que deberíamos utilizar es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
  <html>
    <head>
       <title>Equipos obsoletos</title>
    </head>
    <body>
        <h2>La memoria es demasiado baja</h2>
        <xsl:apply-templates select="/ordenadores/equipo" />
    </body>
  </html>
  </xsl:template>
  <xsl:template match="equipo">
     <xsl:if test="@memoria&lt;1024">
     <p><strong><xsl:value-of select="." /></strong> tiene poca memoria</p>
     </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

1.5.2 El elemento *<xsl:choose>*

Con este elemento podemos elegir entre varias opciones excluyentes entre sí. Podemos indicar más posibilidades de elección. La sintaxis de este elemento es la siguiente:

```
<xsl:choose>
   <xsl:when test=condicion1>
   …
   </xsl:when>
   <xsl:when test=condicion2>
   …
   </xsl:when>
```

```
      …
      <xsl:otherwise>
      …
      </xsl:otherwise>
   </xsl:choose>
```

Utilicemos choose en el ejemplo anterior:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
       <html>
         <head>
            <title>Equipos </title>
         </head>
         <body>
            <h2>RESUMEN DE LOS EQUIPOS</h2>
             <xsl:apply-templates select="/ordenadores/equipo" />
          </body>
        </html>
  </xsl:template>
  <xsl:template match="equipo">
       <xsl:choose>
           <xsl:when test="@memoria&lt;1024">
             <p><strong><xsl:value-of select="." /></strong> tiene poca memoria</p>
           </xsl:when>
           <xsl:when test="@memoria=1024">
             <p><strong><xsl:value-of select="." /></strong> tiene memoria normal</p>
           </xsl:when>
           <xsl:otherwise>

             <p><strong><xsl:value-of select="." /></strong>tiene mucha memoria</p>
           </xsl:otherwise>
        </xsl:choose>
      </xsl:template>
   </xsl:stylesheet>
```

1.5.3 El elemento *<xsl:for-each>*

Para poder mostrar todos los nodos en un “nodo-set” utilizamos el elemento for-each. Este elemento permite que todos los nodos puedan ser procesados de acuerdo a las instrucciones que hay dentro de un elemento “for-each”.

Veamos un ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<ordenadores>
   <equipo memoria=”1024”>
```

```
      <caracteristica tipo=”id”>B302PC01</caracteristica>
      <caracteristica tipo=”marca”>Genérico</caracteristica>
  </equipo>
   <equipo memoria=”512”>
    <caracteristica tipo=”id”>B302PC02</caracteristica>
    <caracteristica tipo=”marca”>Genérico</caracteristica>
   </equipo>
   <equipo memoria=”2048”>
    <caracteristica tipo=”id”>B302PC03</caracteristica>
    <caracteristica tipo=”marca”>HP</caracteristica>
  </equipo>
   <equipo memoria=”512”>
    <caracteristica tipo=”id”>B302PC04</caracteristica>
    <caracteristica tipo=”marca”>HP</caracteristica>
   </equipo>
</ordenadores>
```

Podríamos aplicar la siguiente hoja de estilos:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
  <html>
    <head>
       <title>Equipos</title>
    </head>
    <body>
       <h2>RESUMEN DE LOS EQUIPOS</h2>
       <xs:apply-templates select=”/ordenadores/equipo” />
    </body>
  </html>
  </xsl:template>
  <xsl:template match=”equipo”>
    <ul>
    <xsl:for-each select=”caracteristica”>
     <li><xsl:value-of select=”.”/></li>
    </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

Podemos filtrar la salida del documento XML añadiendo un criterio en el atributo select del elemento <xsl:for.each>.

1.5.4 El elemento *<xsl:sort>*

Este elemento se emplea para mostrar la salida de los elementos ordenada. Se puede utilizar en combinación con “apply-templates” y “for-each”. Para el ejemplo anterior, podríamos aplicar la siguiente hoja de estilos:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
  <html>
    <head>
       <title>Equipos</title>
    </head>
    <body>
       <h2>RESUMEN DE LOS EQUIPOS</h2>
       <xsl:apply-templates select=”/ordenadores/equipo” >
         <xsl:sort select=”@memoria”/>
        </xsl:apply-templates>
    </body>
  </html>
  </xsl:template>
  <xsl:template match=”equipo”>
    <h3><xsl:value-of select=”@memoria”/></h3>
    <ul>
    <xsl:for-each select=”caracteristica”>
    <xsl:sort select=”.” order=”descending” />
     <li><xsl:value-of select=”.”/></li>
    </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

Si probamos este ejemplo, podremos observar que el orden de la memoria se ha establecido por orden alfabético, cuando lo normal hubiera sido por orden numérico, Para eso, debemos usar el atributo “data-type”, del siguiente modo:

```
<xsl:sort select=”@memoria” data-type=”number” />
```

Este atributo puede tener los siguientes valores:

- text, es el valor por defecto
- number
- qname, se refiere a tipos definidos por el usuario.

1.6 Modos XSLT

Hasta ahora, cuando hemos utilizado un elemento apply-templates, lo hemos relacionado con un template, para lo cual, hacíamos corresponder el atributo select de apply-templates con el atributo match de template. No hemos tenido ambigüedades, porque para cada apply-templates sólo había un template. Pero, si tenemos que utilizar varios template con el mismo valor para match, ¿cómo los distinguiremos?

Veamos un ejemplo donde tendríamos que utilizar lo anterior, cuando tengamos que procesar un mismo nodo varias veces en el documento.

```
<?xml version="1.0" encoding="UTF-8"?>
<Articulo>
  <Autores>
   <Autor>Mauricio Aznar</Autor>
   <Autor>Esperanza Rojo</Autor>
   <Autor>José María Soler</Autor>
  </Autores>
  <Titulo> Fundamentos de XML </Titulo>
  <Capitulos>
       <Capitulo número="1" título="Introducción"> Introducción al XML</Capitulo>
       <Capitulo número="2" título="XPath"> Lenguaje XPath</Capitulo>
       <Capitulo número="3" título="XSLT"> El empleo de XSLT.</Capitulo>
       <Capitulo número="4" título="XQuery"> Conexión con las bases de datos.</Capitulo>
  </Capitulos>
</Articulo>
```

Supongamos que queremos generar a partir del documento xml un documento html que tenga el siguiente aspecto:

```
 <html>
  <head>
    <title> Fundamentos de XML </title>
  </head>
  <body>
   <h3> Fundamentos de XML </h3>
   <p>Autores: Mauricio Aznar,Esperanza Rojo y José María Soler.</p>
   <h2>Índice</h2>
    <p><strong>1:</strong>Introducción</p>
   <p><strong>2:</strong>XPath</p>
   <p><strong>3:</strong>XSLT</p>
   <p><strong>4:</strong>XQuery</p>
   <h3>1. Introducción</h3>
   <p> Introducción al XML </p>
   <h3>2. XPath</h3>
    <p> Lenguaje Xpath.</p>
   <h3>3. XLST</h3>
    <p> El empleo de XSLT.</p>
   <h3>4. XQuery</h3>
   <p> Conexión con las bases de datos. </p>
  </body>
</html>
```

Como se puede observar, el título de los capítulos se emplea dos veces: la primera vez para la tabla de contenidos, y la segunda durante el desarrollo del texto. Esto quiere decir, que el documento XSLT que necesitamos utiliza dos template con el atributo match con valor Capitulo.

¿Cómo los distinguiremos entre sí, para cuando los referenciemos desde el elemento "apply-templates". La solución es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
  <html>
  <head>
     <title><xsl:value-of select="/Articulo/Titulo"/></title>
  </head>
  <body>
     <h3><xsl:value-of select="/Articulo/Titulo"/></h3>
    <p>Autores: <xsl:apply-templates select="/Articulo/Autores/Autor"/></p>
    <h2>Índice</h2>
    <xsl:apply-templates select="/Articulo/Capitulos/Capitulo" mode="Índice"/>
    <xsl:apply-templates select="/Articulo/Capitulos/Capitulo" mode="Desarrollo"/>
  </body>
  </html>
</xsl:template>
<xsl:template match="Autor">
   <xsl:choose>
     <xsl:when test="position() < last()-1"> <xsl:value-of select="."/>,</xsl:when>
      <xsl:when test="position() = last()-1"> <xsl:value-of select="."/> y </xsl:when>
     <xsl:otherwise> <xsl:value-of select="."/>.</xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:template match="Capitulo" mode="Índice">
   <p>
     <strong><xsl:value-of select="@número"/>:</strong>
     <xsl:value-of select="@título"/>
   </p>
</xsl:template>
<xsl:template match="Capítulo" mode="Desarrollo">
    <h3><xsl:value-of select="@número" />. <xsl:value-of select="@título"/></h3>
    <p><xsl:value-of select="."/></p>
</xsl:template>
</xsl:stylesheet>
```

Observa que se utiliza el atributo mode, que distingue un template de otro.

1.7 Parámetros y variables XSLT

XSLT permite especificar variables y parámetros mediante los elementos xsl:variable y xsl:parameter. Ambos

pueden ser referenciados usando $NombreVariable o $NombreParametro.

Nota: No hay que confundir las variables en XSLT con las variables en otros lenguajes, ya que una vez que su valor es fijado no se puede modificar.

Veamos un ejemplo de su utilización: Suponer que tenemos un documento XML con los nombres y edades de una serie de personas y queremos que al introducir el nombre nos aparezca la edad.

```
<?xml version="1.0" encoding="UTF-8"?>
<edades>
  <persona nombre=”Ana” edad=”21” />
  <persona nombre=”Eva” edad=”19” />
  <persona nombre=”Antonio” edad=”41” />
  <persona nombre=”Luis” edad=”35” />
</edades>
```

La hoja de estilo XSLT que debemos aplicar es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:parameter name=”persona” />
  <xsl:template match="/">
     <html>
       <head>
          <title>Encontrar la edad usando un parámetro XSLT</title>
       </head>
       <body>
          <xsl:apply-templates select=”/edades/persona[@nombre=$persona]” />
       </body>
     </html>
</xsl:template>
<xsl:template match=”persona”>
   <p>La edad de <xsl:value-of select=”$persona” /> is
      <xsl:value-of select=”@edad” /> </p>
</xsl:template>
</xsl:stylesheet>
```

Para pasar el parámetro desde la línea de comando usamos la siguiente sintaxis:

```
java -jar saxon.jar -o edades.html edades.xml edades.xslt persona=”Ana”
```

1.8 El elemento *<xsl:output>*

Con XSLT se pueden generar documentos XML, HTML o de texto. Podemos elegir de qué tipo será el resultado mediante el elemento <xsl:output method=”tipo de documento a generar” />, donde tipo de documento a generar puede ser xml, html o text.

1.9 El elemento *<xsl:text>*

En algunas ocasiones necesitaremos texto literal en la salida, por ejemplo espacios en blanco para separar valores. En este caso utilizaremos la etiqueta <xsl:text> con su código correspondiente.

1.10 El elemento *<xsl:element>* y el elemento *<xsl:attribute>*

Cuando tengamos que crear nodos nuevos en el documento de salida, con su etiquetas y atributos necesitaremos utilizar las etiquetas <xsl:element> y <xsl:attribute> para poder utilizar los valores del XML origen en dichos nodos.

La instrucción <xsl:attribute> permite generar un atributo y su valor. Se utiliza cuando el valor del atributo se obtiene a su vez de algún nodo.

Bibliografía

- https://www.w3schools.com/xml/xsl_intro.asp