UD7 Transformación de documentos XML

Cascading Style Sheets (CSS)

CSS, Cascading Style Sheets, hojas de estilo en cascada, es un lenguaje sencillo para la aplicación de estilos a un elemento XML. Es decir, podremos ver nuestros documentos XML en el navegador como si de una página HTML se tratase, aunque ésto sólo funcionará en browsers XML que soporten CSS (IE, Mozilla, Konqueror, Amaya, etc.).

Para un mismo documento XML se pueden tener varias CSS y así estar formateado de diferentes formas, según nuestros propósitos o del posible uso que se le vaya a dar al documento. Las CSS se expresa mediante una serie de reglas que guardaremos en un fichero de texto. Cada regla contiene el nombre del elemento al que se aplica y el estilo definido.

Ejemplo fichero XML:

```
<?xml version="1.0" encoding="UTF-7"?>
<?xml-stylesheet href="ejemplo0.css" type="text/css"?>
<ejemplo> Hola mundo ! </ejemplo>
```

Ejemplo fichero CSS:

```
ejemplo {display: block; color: blue}
```

CSS se utiliza con ficheros XML del mismo modo que se hace con ficheros XHTML. También se puede trabajar con clases (atributo *class*) e identificadores (atributo *id*).

XSLT: Transformaciones XSL.

www.w3schools.com/xsl/

XSLT es un lenguaje de programación declarativo que permite generar otros documentos a partir de documentos XML.

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. Normalmente se utilizará un navegador Web que lo soporte.

El resultado de la ejecución del programa es un nuevo documento (que puede ser un documento XML o no). En nuestro caso será un documento HTML5.

XSLT se utiliza para obtener a partir de un documento XML otros documentos (XML o no). A un documento XML se le pueden aplicar distintas hojas de estilo XSLT para obtener distintos resultados y una misma hoja de estilo XSLT se puede aplicar a distintos documentos XML.

Enlazar documentos XML con hojas de estilo XSLT

Se puede asociar de forma permanente una hoja de estilo XSLT a un documento XML mediante la instrucción de procesamiento <?xml-stylesheet ?>, la misma que permite asociar hojas de estilo CSS. La instrucción de procesamiento <?xml-stylesheet ... ?> va al principio del documento, después de la declaración XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="ejemplo.xsl"?>
```

Cuando se visualiza en un navegador web un documento XML enlazado con una hoja de estilo XSLT, los navegadores muestran el resultado de la transformación, aunque si se muestra el código fuente de la página, los navegadores muestran el documento XML original.

Hojas de estilo XSLT

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). La transformación del documento se realiza de la siguiente manera:

- El procesador analiza el documento y construye el árbol del documento.
- El procesador va recorriendo todos los nodos desde el nodo raíz,

aplicando a cada nodo una plantilla, sustituyendo el nodo por el resultado.

- Cuando el procesador ha recorrido todos los nodos, se ha terminado la transformación.

Plantillas (Template)

xsl:template

Define una plantilla de salida para los nodos de un tipo y contexto en particular.

<xsl:template match='/'> Todo el documento, nodo raíz.
<xsl:template match='tienda'> Nodo llamado tienda cae de la raíz
<xsl:template match='producto'> nodo llamado producto.

Extraer partes de un documento

xsl:value-of

Para extraer partes de un documento XML utilizamos el elemento `xsl:value-of`, dentro de un elemento xsl-template, y con la siguente sintaxis:

```
<xsl:value-of select="XPath" />
```

En el atributo select deberemos indicar el XPath del XML que queremos extraer. Para extraer un elemento lo indicamos tal cual, por ejemplo "comics". Si queremos extraer elementos dentro de otros, construimos un XPath usando "/" como separador: "comics/datos", "comics/datos/referencia". Los atributos se preceden de "@": "comics/comic/precio/@moneda".

Ten en cuenta que el codigo XML que se analiza es aquel que ha sido capturado por atributo match del `xsl-template`

```
Datos de mi comiteca:
Fecha: <xsl:value-of select="comics/datos/fecha_actualizacion" />
Referencia:<xsl:value-of select="comics/datos/referencia" />
```

El elemento xsl:value-of tiene una importante limitación: solo devuelve el primer trozo XML que cumple el selector. En nuestro ejemplo, xsl:value-of select="comics/comic", solo devuelve el XML del primer comic.

xsl:for-each

Para extraer más de un elemento tenemos el elemento xsl:for-each que es similar a un bucle. Los elementos contenidos en el, se repiten para cada trozo XML seleccionado.

Por ejemplo, para mostrar la lista de todos los comics:

```
<xsl:for-each select="comics/comic" >
      <xsl:value-of select="titulo" />
      <xsl:value-of select="autor" />
</xsl:for-each >
```

```
<xsl:for-each select="comics/comic[condicion]" >
```

xsl:sort

El bucle xsl:for-each permite ordenar los fragmentos xml con la etiqueta, xsl:sort cuya sintaxis es:

```
<xsl:sort select="XPATH" order="descending|ascending" />
```

Esta etiqueta va siempre después del xsl:for-each al que afecta, y se cierra en si misma.

```
<xsl:for-each select="comics/comic" >
      <xsl:sort select="titulo" />
      <xsl:value-of select="titulo" />
      <xsl:value-of select="autor" />
</xsl:for-each >
```

xsl:if

La etiqueta `xsl:if` permite generar contenido condicionalmente. Su sintaxis es:

```
<xsl:if test="expression"> ... ...contenido generado ... </xsl:if>
```

El atributo `test` es obligatorio. Indica la condición que se tiene que cumplir para que el contenido opcional sea generado.

```
<xsl:for-each select="comics/comic" >
      <xsl:if test="año_publicación = 2006" >
            <span class=”rojo”>NOVEDAD</span>
      </xsl:if>
</xsl:for-each >
```

xsl:choose

Esta etiqueta permite construir una estructura condicional. Su sintaxis es:

```
<xsl:choose>
      <xsl:when test="expression"> ... some output ... </xsl:when>
      <xsl:otherwise> ... some output .... </xsl:otherwise>
</xsl:choose>
```

En cada etiqueta `xsl:when` realizamos un test. Se genera el primer test que resulta ser verdadero, o la parte `xsl-otherwise` en caso de que no se cumpla ningún test, y `xsl:otherwise`, que es opcional, se encuentre presente.

Ejemplo para la comiteca:

```
<xsl:choose>
      <xsl:when test="formato= 'BN' " >Blanco y Negro</xsl:when>
      <xsl:when test="formato= 'COL'" >Color</xsl:when>
      <xsl:when test="formato= 'LX'" >edición de lujo</xsl:when>
      <xsl:when test="formato= 'TD'" >Tapa dura, color</xsl:when>
      <xsl:otherwise>
            <xsl:value-of select="formato" />
      </xsl:otherwise>
</xsl:choose>
```

Test

La sintaxis completa del atributo `test` es:

```
test = "selector_XPATH [= | != | &lt; | &gt; ] expresion"
```

Para comprobar la existencia de un elemento test = "selector_XPATH"

El operador debería ser `=,<, >`, pero en XML tienes que escapar los dos últimos operadores. Si la expresión es un número, se pone tal cual, y si es cadena, hay que delimitarla con comilla simple.

```
test ="año_publicación &gt; 2005"
      test ="autor = 'Breccia'"
test ="formato != 'BN'"
test = "precio"
```

xsl:element

El elemento `xsl:element` sirve para generar en la salida, un nuevo elemento cuyo nombre será el atributo `name`. Por ejemplo:

```
<xsl:element name="ul" >
      <xsl:for-each select="comics/comic" >
            <xsl:element name="li">
                  <xsl:value-of select="titulo" />
            </xsl:element >
      </xsl:for-each >
</xsl:element >
```

xsl:copy-of

Con la instrucción `xsl:copy-of` podemos copiar un elemento completo (incluyendo etiquetas, atributos y nodos hijos) en la salida.

Solo tenemos que definir en el atributo `select` que elemento queremos copiar.

```
<xsl:copy-of select="comics/datos">
```

xsl:copy

Este elemento efectua una copia "hueca" de un elemento: solo copia la etiqueta del elemento, ignorando atributos y nodos hijos. No tiene atributo `select` ya que siempre copia el elemento actual dentro de un "template" o un bucle for-each.

```
<xsl:for-each select="comics/comic" >
      <xsl:copy />
</xsl:for-each >
```

Dentro de un template: <xsl:copy > <xsl:apply-templates /> </xsl:copy >

Observa este ejemplo para ver las diferencias entre ambos métodos.

Este elemento se puede utilizar para hacer una copia de un documento XML eliminando todos los atributos:

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

xsl:text

El procesador XSLT omite los espacios, tabuladores y saltos de línea en la salida. Con esta técnica además de poder incluir texto en la salida, el procesador respeta todos los blancos. Si quieres añadir una salto de linea tienes tres posibiliades:

```
Usando Unix LF <xsl:text>&#10;</xsl:text>
Usando DOS CR+LF <xsl:text>&#13;&#10;</xsl:text>
Poniendo un salto de linea <xsl:text> </xsl:text>
```

xsl:attribute

Para añadir atributos a un elemento, XSLT dispone de `xsl:attribute`, que añade un atributo al ultimo elemento generado en la salida. Su sintaxis es:

```
<xsl:attribute name="NOMBRE">VALOR</xsl:attribute>
```

Como VALOR podemos poner desde un texto como por ejemplo "disabled", hasta una etiqueta `xsl:value-of` para capturar un valor. Aqui tienes un ejemplo:

```
<xsl:for-each select="comics/comic">
      <li>
            <xsl:attribute name="title">
                  <xsl:value-of select="autor"/>
            </xsl:attribute>
            <xsl:attribute name="id">
                  <xsl:value-of select="position()"/>
            </xsl:attribute>
            <xsl:value-of select="titulo"/>
      </li>
</xsl:for-each>
```

El elemento xsl:apply-templates

Cuando se aplica una plantilla a un nodo, en principio la plantilla se aplica únicamente al nodo, pero se sustituye el nodo y todos sus descendientes por el resultado de la aplicación de la plantilla, lo que nos haría perder a los descendientes. Si se quiere que antes de sustituir el nodo y todos sus descendientes se apliquen también a los descendientes las plantillas que les correspondan, hay que utilizar la instrucción <xsl:apply-templates />, como en el ejemplo 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='/'>
            <xsl:apply-templates />
      </xsl:template>
      <xsl:template match="elemento">
      </xsl:template>
</xsl:stylesheet>
```

Esta instrucción permite aplicar un template al nodo actual o a los nodos hijo del elemento actual. Si la instrucción no contiene atributos, es decir se declara como *<xsl:apply-template />* el procesador buscará templates que

coincidan con los nombres de nodos hijo del nodo contexto. Se aplicará tantas veces como nodos hijo haya.

El elemento xsl:apply-templates contiene un atributo select que permite especificar una expresión XPath para seleccionar los nodos hijos a los cuales se les aplicará un template. Este atributo puede ser usado para especificar el orden en el cual los hijos serán evaluados.