Introducción a XML | LMTW

Víctor Peinado

10 - 18 de mayo de 2016

Referencias

Hay numerosas referencias válidas en internet para aprender XML y tecnologías asociadas. En esta presentación, vamos a basarnos en las tres siguientes:

- A Gentle Introduction to XML[1]
- Tutorial de XML en W3Schools[2]
- Tutorial de DTDs en W3Schools[3]

[1] http://www.tei-c.org/release/doc/tei-p5-doc/en/html/SG.html

[2] http://www.w3schools.com/xml/

[3] http://en.wikipedia.org/wiki/Document_Type_Definition

Introducción a XML

XML[4] (*eXtensible Mark-up Language*) es un lenguaje de marcas orientado a transmitir y almacenar información de manera estructurada. Se ha convertido en uno de los principales formatos de intercambio de información utilizado en Internet y en bases de datos.

[4] http://en.wikipedia.org/wiki/XML

Fue diseñado por el W3C[5] como un subconjunto más restrictivo a partir de otro lenguaje de marcas anterior llamado SGML[6].

[5] http://www.w3c.es

[6] http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language

En términos generales, podemos considerar XML como un *metalenguaje*, es decir, un lenguaje (de marcas) que nos va a permitir definir otros lenguajes (de marcas).

Algunas características de XML

- XML se diseñó para transmitir y almacenar (y no para visualizar) información de manera estructurada.
- XML es extensible: las etiquetas XML no están predefinidas. Tenemos que/podemos definir nuestro propio conjunto de etiquetas.
- XML se diseñó para ser auto-descriptivo, fácilmente procesable por máquinas de manera automática y, aun así, ser lo suficientemente transparente como para ser leído por seres humanos.
- XML es independiente de la plataforma y del software.
- Los documentos XML están **bien formados** cuando cumplen un mínimo de reglas sintácticas.
- Los documentos XML pueden ser además **válidos**, si están estructurados siguiendo las reglas de una determinada gramática.

Diferencias entre XML y HTML

- XML no reemplaza al HTML. Cada lenguaje de marcas se diseñó con unos objetivos diferentes.
- XML se diseñó para transportar y almacenar datos, y se centra en **describir correctamente el significado** de los distintos tipos de datos que maneja.
- HTML se diseñó para visualizar documentos hipertextuales y multimedia, y se centra en **la presentación** de la información.
- Las primeras versiones de HTML (hasta HTML 4.1) estaban diseñadas sobre SGML.
- Pero, en paralelo, existen versiones más modernas de HTML que están basadas en XML y que se denominan XHTML.

Marcado descriptivo vs. procedimental

- Un lenguaje de marcas descriptivo etiqueta la estructura de un documento, indicando cosas como *aquí comienza un párrafo* o *aquí termina un epígrafe de nivel 2*.
- Por contra, un lenguaje de marcas procedimental además de marcar estructura define otro tipo de procedimientos que hay que realizar al procesar el documento, como que *determinada porción de texto es un párrafo y debe visualizar con justificación a la izquierda* o *los títulos se visualizan con tal fuente y en color azul*.
- XML es un lenguaje de marcas descriptivo. Las instrucciones para procesar el documento (para darle formato, p. ej.) se indican en un documento diferente.
- HTML, en sus orígenes, era un lenguaje de marcas claramente procedimental. Con el paso del tiempo y la inclusión de hojas de estilo, se ha ido convirtiendo en un lenguaje de marcas más descriptivo.
- La ventaja de usar un lenguaje descriptivo radica en que podemos procesarlo de distintas maneras. Por ejemplo, guión de teatro en HTML o XML.

Documentos XML

Tipo de documento

- Una de las características claves de XML es que todo documento puede clasificarse como una **instancia** de un determinado **tipo de documento**.
- Este **tipo de documento** consiste es una descripción formal de las partes constitutivas y la estructura del documento en cuestión,

p. ej.:

- Toda trama se divide en presentación, nudo y desenlace.
- Un artículo científico tiene varias secciones: título, nombre de autor, resumen, introducción, métodos, resultados, discusión, conclusiones y referencias.
- Una oración simple es una secuencia formada por un sintagma nominal y un sintagma verbal.

- El **analizador** (*parser*) es la herramienta que comprueba que efectivamente el documento XML en cuestión respeta la especificación formal del tipo de documento del que dice ser ejemplar. Cuando es así, decimos que el documento es **válido** o que **el documento está validado**.

Estructura de datos en XML: elementos

El término técnico usado en XML como unidad estructural mínima es el **elemento**.

Ya sabemos que, formalmente, estos elementos se especifican como los elementos HTML: escritos entre paréntesis angulares: `<elemento>` y `</elemento>`.

En XML estos elementos no tienen ningún significado (aunque lo ideal es darles nombres inteligibles que sean lo suficientemente expresivos para intuir su semántica). Lo que hacemos es simplemente definir la relación de un elemento con respecto a otros:

- El elemento `<presentación>` puede aparecer como elemento constitutivo del elemento `<trama>`.
- El elemento `<trompfundercillo>` puede aparecer como elemento constitutivo del elemento `<trompfunder>`.

Dentro de una instancia de documento XML, todo elemento tiene que estar explícitamente marcado, con etiquetas de inicio y fin.

```
<texto>Como dijo Federico Trillo frente a las
tropas de El Salvador: <cita>¡Viva Honduras!</cita></texto>
```

Ejemplo de documento XML

Una agenda de contactos es un conjunto de **fichas** con determinados *datos de contacto* asociados.

```
<agenda_de_contactos>
<contacto>
<nombre>Pepito</nombre>
```

```
<apellidos>
  <apellido1>Pérez</apellido1><apellido2>Ruz</apellido2>
</apellidos>
<email>pepito@mail.com</email>
<telefono>+34666888555</telefono>
</contacto>
<contacto>
<nombre>Ana</nombre>
<apellidos>
  <apellido1>Sánchez</apellido1>
</apellidos>
<email>asanchez@mail.com</email>
<telefono>+34911555888</telefono>
</contacto>
...
</agenda_de_contactos>
```

Documentos bien formados y válidos

Documentos bien formados

El ejemplo anterior es un documento XML **bien formado** ya que cumple las siguientes condiciones:

- Hay un único elemento raíz del que cuelgan el resto de elementos: en este caso `<agenda_de_contactos>`.
- Los elementos están correctamente anidados y no se solapan: antes de abrir un nuevo elemento cerramos el último que hemos abierto.
- Todos los elementos están correctamente cerrados, incluídos los elementos vacíos, si los hubiera.

Cualquier documento XML que esté bien formado puede ser procesado de muchas maneras útiles: p. ej. un programa que envía a los emails de mis amigos mi felicitación de navidad.

Documentos válidos

Un documento simplemente bien formado no aprovecha toda la potencia de las tecnologías XML.

Sería útil poder asegurarme de que los datos de mi agenda son consistentes y que todos mis contactos tienen, al menos, un teléfono o un email donde pueda *contactar* con ellos.

Para ello necesitamos previamente validar la estructura formal del documento y asegurarnos de que mi agenda es efectivamente una instancia del tipo de documento `agenda_de_contactos`.

La definición formal de lo que constituye un documento XML válido se especifica en un fichero aparte. Existen dos formatos:

- Document Type Definition (DTD)[7]
- XML Schema[8]

[7] http://en.wikipedia.org/wiki/Document_Type_Definition

[8] http://en.wikipedia.org/wiki/XML_Schema_(W3C)

Tanto las DTDs como los XML Schemas nos van a gustar mucho porque son **definiciones formales de gramáticas independientes del contexto** :-)

Document Type Definition (DTD)

Una DTD es un fichero de texto con un formato especial formado por un conjunto de instrucciones declarativas.

La DTD que define el tipo de documento `agenda_de_contactos` sería la siguiente:

```
<!ELEMENT agenda_de_contactos (contacto+) >
<!ELEMENT contacto            (nombre, apellidos, email*, telefono*) >
<!ELEMENT nombre              (#PCDATA) >
<!ELEMENT apellidos           (apellido1, apellido2?) >
<!ELEMENT apellido1           (#PCDATA) >
<!ELEMENT apellido2           (#PCDATA) >
<!ELEMENT email               (#PCDATA) >
<!ELEMENT telefono            (#PCDATA) >
```

Podemos entender estas DTDs como **gramáticas independientes del contexto** (Gramáticas Tipo 2, según la jerarquía de Chomski).

```
O -> SN SV
SN -> Det N (Adj)
SN -> NP
SV -> V
SV -> V SN
SV -> V SP
SP -> Prep SN
```

Como en cualquier definición formal de una gramática, en una DTD definimos una estructura en forma de árbol, con un elemento raíz y otros elementos hijos que cuelgan de él.

Sintaxis de las DTDs

Todas las instrucciones de las DTD estructuras de varios elementos separadas por espacios. En general, tienen la forma:

```
<!TIPO_DE_OBJETO identificador_genérico (modelo_de_contenido)>
```

Los **identificadores genéricos** son los nombres que reciben los elementos: debe comenzar por una letra, y está formado por una secuencia de caracteres alfanuméricos, guiones, puntos, diferenciando mayúsculas y minúsculas.

El **modelo de contenido** se declara entre paréntesis y define el contenido del elemento en cuestión. Dicho contenido pueden ser otros elementos, con cuantificadores, o determinadas *palabras reservadas* que permiten especificar otros detalles, como veremos a continuación.

En el caso de las definiciones de elementos, las instrucciones son tripletas del siguiente tipo:

```
<!ELEMENT identificador_genérico (modelo_de_contenido)>
```

#PCDATA

Hay varias *palabras reservadas* para indicar los detalles del modelo de contenido de un elemento.

La más común es `#PCDATA` (*parsed character data*) que se utiliza para indicar que el elemento puede etiquetar cualquier secuencia de caracteres (pero no elementos XML).

```
<!ELEMENT nombre              (#PCDATA) >
<!ELEMENT apellido1           (#PCDATA) >
<!ELEMENT apellido2           (#PCDATA) >
<!ELEMENT email               (#PCDATA) >
<!ELEMENT telefono            (#PCDATA) >
```

Para más detalles sobre estas palabras reservadas, consulta el tutorial de DTD en W3Schools[9].

[9] `http://www.w3schools.com/dtd/dtd_building.asp`

Cuantificadores

Existen tres cuantificadores, que permiten indicar dentro del modelo de contenido, el número de veces que puede aparecer un elemento:

- El símbolo de la suma `+` indica que el elemento puede aparecer *1 o más veces*.
- La interrogación `?` indica que el elemento es opcional, pero si aparece solo lo hace una vez: *o ó 1 vez*.
- El asterisco `*` indica que el elemento es opcional y, si aparece, puede hacerlo más de una vez: *o, 1 o más veces*.

Cuando no hay cuantificador, la aparición del elemento es obligatoria y lo hace una sola vez.

```
<!ELEMENT agenda_de_contactos (contacto+) >
<!ELEMENT contacto            (nombre, apellidos, email*, telefono*) >
<!ELEMENT apellidos           (apellido1, apellido2?) >
```

Conectores

Cuando el modelo de contenido define más de un componente podemos utilizar distintos contectores para especificar el orden de aparición.

- La coma , permite representar secuencias, en las que el orden está definido.
- La barra vertical | indica disyunción: o bien un componente o bien otro, pero no los dos.

```
<!ELEMENT agenda_de_contactos (contacto+) >
<!ELEMENT contacto            (nombre|apellidos|email*|telefono*) >
<!ELEMENT apellidos           (apellido1, apellido2?) >
```

Podemos agrupar los elementos usando cuantificadores y conectores de maneras más sofisticadas.

```
<!--
el elemento apellidos puede contener un único apellido1
o bien una secuencia de apellido1 y apellido2
-->
<!ELEMENT apellidos ( apellido1 | (apellido1, apellido2) ) >

<!--
un poema está formado por un título opcional seguido de:
- una secuencia de estrofas, o bien,
- una secuencia de pareados; o bien,
- una secuencia de líneas,
-->
<!ELEMENT poem (title?, (stanza+ | couplet+ | line+) ) >

<!--
un poema está formado por un título opcional seguido de una
mezcla indeterminada de estrofas, pareados y líneas
-->
<!ELEMENT poem (title?, (stanza | couplet | line)+ ) >
```

Añadiendo complejidad

```
<agenda_de_contactos>
<contacto>
```

```
<amigo/>
<nombre>Pepito</nombre>
<apellidos>
  <apellido1>Pérez</apellido1><apellido2>Ruz</apellido2>
</apellidos>
<email>pepito@mail.com</email>
<telefono>+34666888555</telefono>
</contacto>

<contacto>
<trabajo/>
<nombre>Ana</nombre>
<apellidos>
  <apellido1>Sánchez</apellido1>
</apellidos>
<email>asanchez@mail.com</email>
<telefono>+34911555888</telefono>
</contacto>
...
</agenda_de_contactos>
```

Si añado un elemento vacío que indique si la persona de contacto es amigo o compañero del trabajo, necesito modificar mi DTD:

```
<!ELEMENT agenda_de_contactos (contacto+) >
<!ELEMENT contacto            ((amigo?|trabajo?), nombre, apellidos,
                                  email*, telefono*) >
<!ELEMENT amigo               EMPTY >
<!ELEMENT trabajo             EMPTY >
<!ELEMENT nombre              (#PCDATA) >
<!ELEMENT apellidos           (apellido1, apellido2?) >
<!ELEMENT apellido1           (#PCDATA) >
<!ELEMENT apellido2           (#PCDATA) >
<!ELEMENT email               (#PCDATA) >
<!ELEMENT telefono            (#PCDATA) >
```

Pero también podemos añadir metadatos e información en forma de **atributos**.

```
<agenda_de_contactos>
<contacto id="1" tipo="amigo">
<nombre>Pepito</nombre>
<apellidos>
  <apellido1>Pérez</apellido1><apellido2>Ruz</apellido2>
</apellidos>
<email tipo="personal">pepito@mail.com</email>
```

```
<telefono tipo="movil">+34600888111</telefono>
</contacto>
  <contacto id="2" tipo="trabajo">
<nombre>Ana</nombre>
<apellidos>
  <apellido1>Sánchez</apellido1>
</apellidos>
<email tipo="personal">asanchez@mail.com</email>
<telefono tipo="trabajo">+34911555888</telefono>
<telefono tipo="casa">+349172222111</telefono>
</contacto>
...
</agenda_de_contactos>
```

Estructura en XML: Atributos

En el contexto de las tecnologías XML, el término **atributo** tiene un uso y un significado claro.

Se utilizan habitualmente para codificar información que describe un tipo de elemento concreto y que no se considera parte de su contenido.

Formalmente, ya lo sabemos de HTML, los atributos son pares *clave:valor* que se especifican solo en la etiqueta de inicio.

```
<telefono tipo="trabajo">+34911555888</telefono>
<telefono tipo="casa">+349172222111</telefono>
```

Las claves y los valores se relacionan con el símbolo de la igualdad = y los valores aparecen siempre entre comillas.

En el caso de que un elemento contenga más de un atributo, el orden en el que aparecen éstos es irrelevante.

Definiendo atributos en mi DTD

Al igual que los elementos, los atributos se definen en la DTD utilizando una sintaxis similar. Además de especificar su nombre y el nombre del elemento con el que aparecen, podemos especificar qué tipo de valores puede aceptar, incluídos valores por defecto.

La sintaxis general es:

```
<!ATTLIST elemento
          atributo  tipo_de_valor   valor_por_defecto >
```

La DTD de nuestro ejemplo anterior sería algo como:

```
<!ELEMENT contacto ((amigo?|trabajo?), nombre, apellidos,
                    email*, telefono*) >
```

```
<!ATTLIST contacto
          id        ID                            #REQUIRED
          tipo      (amigo|trabajo|desconocido)   "trabajo" >


<!ELEMENT email               (#PCDATA) >
<!ATTLIST email
          tipo      (personal|casa|trabajo)       #IMPLIED >


<!ELEMENT telefono            (#PCDATA) >
<!ATTLIST telefono
          tipo      (movil|casa|trabajo)          #IMPLIED >
```

Atributos: tipos de valores

El segundo elemento de la definición de un atributo es el tipo de valor que acepta, y puede tomar dos formas diferentes:

```
<!ATTLIST contacto
          id        ID                            #REQUIRED
          tipo      (amigo|trabajo|desconocido)   "trabajo" >
```

- El primer caso consiste en usar un número especial de palabras clave que declaran el tipo de valor que el atrobuto puede tomar.
  - `CDATA` para cualquier secuencia de caracteres alfanuméricos, incluídos etiquetas de elementos.
  - `ID`: identificadores únicos.
  - Otras palabras clave como `NMTOKEN`, `NMTOKENS`, `IDEREF`, `IDREFS`, `ENTITY`, `ENTITIES`.
- En el segundo caso, también se puede proporcionar de manera explícita una lista de posibles valores que el atributo puede tomar.

A veces es imposible saber qué valores diferentes puede tomar un atributo, pero en general lo mejor es definirlos claramente *a priori*.

Atributos: valores por defecto

El tercer elemento de la definición de un atributo especifica cómo se interpreta la ausencia del atributo.

Esto puede hacer proporcionando un valor concreto (entre comillas) o indicando una de las dos palabras clave siguiente:

- `#REQUIRED`: el atributo es obligatorio.
- `#IMPLIED`: el atributo es opcional.

Entidades

- XML proporciona un método fácil y sencillo para definir nombres que hagan referencia de partes arbitrarias del contenido de un documento.
- De hecho, estas **entidades** en XML funcionan como una especie de *alias* de otras porciones de texto.
- Una entidad puede hacer referencia a una cadena de texto e incluso un documento completo.
- Las entidades se definen en las DTDs de una manera similar a como hemos hecho con **elementos** o **atributos**.
- Las entidades se pueden incluir en un documento XML utilizando lo que se denomina **referencia a una entidad**.

Añadiendo entidades a mi DTDs

Como las declaraciones anteriores, la definición de una entidad es una instrucción que comienza con una palabra reservada `ENTITY`, sguida del nombre de la entidad y del valor que adopta cuando se le hace referencia.

Por un lado, la instrucción siguiente define una **entidad interna** llamada `agenda` cuyo valor es *Agenda de contactos de Víctor Peinado*:

```
<!ENTITY agenda "Agenda de contactos de Víctor Peinado">
```

Por otro lado, la siguiente instrucción declara distintas **entidades externas** que hacen referencia ficheros externos diferentes:

```
<!ENTITY agenda_antigua SYSTEM "../old/agenda-2012.xml">

<!ENTITY recetas SYSTEM "http://bit.ly/recetas">
```

Existen incluso entidades públicas, que utilizan **identificadores públicos**.

```
<!ENTITY p3.sg PUBLIC "-//TEI//TEXT Guidelines Chapter on XML//EN" "p4chap2.xml">
```

Referencias a entidades

Una vez hemos definidio una entidad, podemos utilizarla en cualquier sitio, siempre que le hagamos referencia.

La sintaxis de las **referencias a entidades** es la siguiente: `&nombre_de_entidad;`:

```
<agenda_de_contactos descripcion="&agenda;">

&agenda_antigua;
```

```
<contacto id="1" tipo="amigo">
<nombre>Pepito</nombre>
<apellidos>
  <apellido1>Pérez</apellido1><apellido2>Ruz</apellido2>
...
</agenda_de_contactos>
```

Cuando en analizar XML encuentra una referencia a una entidad, inmediatamente la sustituye o expande por el valor declarado de la entidad.

Entre las ventajas de poder usar este tipo de entidades están:

- Ahorra tecleo en aquellas porciones de texto que se repiten a menudo.
- Ayuda a facilidar el mantenimiento y la consistencia de ficheros grandes.

Por ejemplo, en HTML ya están definidas algunas entidades[10] de uso común:

[10] http://ascii.cl/es/codigos-html.htm

```
<p>Las vocales con tilde son &aacute;, &eacute;, &iacute;,
&oacute; y &uacute;.</p>
```

Declaración de documentos XML

La estructura final de un documento XML bien formado y válido contiene dos partes: - un prólogo. - una instancia de documento.

El prólogo contiene una **declaración de XML** y, de manera opcional puede incluir una **declaración del tipo de documento (DTD)**.

La declaración de XML contiene la línea que especifica la versión de XML y el esquema de codificación y el tipo de documento.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ejemplo [
<!ELEMENT ejemplo (p+) >
<!ATTLIST ejemplo n CDATA #IMPLIED>
<!ELEMENT p (#PCDATA)>
]>
<ejemplo n="1">
<p>Esto es una instancia de un ejemplo de documento XML.</p>
</ejemplo>
```

Lo más habitual será situar la DTD en un fichero aparte y enlazar dicho fichero de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE agenda_de_contactos SYSTEM "agenda.dtd" []>
<agenda_de_contactos>
<contacto id="1" tipo="amigo">
<nombre>Pepito</nombre>
<apellidos>
  <apellido1>Pérez</apellido1><apellido2>Ruz</apellido2>
</apellidos>
<email tipo="personal">pepito@mail.com</email>
<telefono tipo="movil">+34600888111</telefono>
</contacto>

...
</agenda_de_contactos>
```