XML

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Marzo de 2016

Lenguajes de marcado

Un *lenguaje de marcado* es un sistema que permite incluir metainformación en un documento, esto es, información sobre la información

- La metainformación tiene que distinguirse sintácticamente del texto.
- Es la evolución del *lapiz azul* con el que tradicionalmente se editaban documentos cuando la tecnología era analógica
  - Ejemplo de lenguaje de marcado muy elemental: redacto un documento en un procesador de textos, lo imprimo y alguien lo revisa, incluyendo anotaciones a mano
    Las anotaciones (metainformación) se distingue fácilmente del texto original
    Para hacer algo semejante de forma digital, es necesaria una sintaxis que separe el texto de la metainformación
- Ejemplos de lenguajes de marcado: troff, LaTeX, JsonML, SGML, XML

- XML *Extensible Markup Language*
  Es una forma de describir datos jerárquicamente. Estándar para transferir información entre distintos sistemas sin tener que adaptarlos a cada plataforma concreta, y de forma que sea fácil de leer por un humano y fácil de procesar por un ordenador
- Creado en 1996 por el W3C (*World Wide Web Consortium*)
  Dos versiones: XML 1.0 y XML 1.1
- Algunos autores lo consideran un lenguaje de marcado, otros, un metalenguaje de marcado
- Proviene de SGML, *Standard Generalized Markup Language*, norma ISO 8879:1986
  SGML es un metalenguaje, un lenguaje para definir lenguajes de marcado,

- XML tiene una sintaxis similar a la de HTML porque ambos provienen de SGML
- XML y HTML no son lenguajes alternativos
  - XML está diseñado para describir y comunicar datos datos de máquina a máquina
  - HTML está diseñado para presentar en pantalla datos con formato. De máquina a persona

Estructura de un documento XML

- En la primera línea es recomendable incluir un *prólogo* aka *declaración*, indicando la versión de xml y, opcionalmente, la codificación empleada

```
<?xml version="1.1" encoding="UTF-8"?>
```

- A continuación puede aparecer un DTD, *Document Type Definition*. Aunque es casi obsoleto

```
CTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- A continuación debe aparecer el cuerpo, formado por un *elemento raiz*. Siembre habrá uno y solo uno, que podrá tener elementos anidados

- XML puede usar cualquier caracter unicode, por omisión codificado en UTF-8
- XML es *case sensitive*
- Se pueden poner comentarios en cualquier lugar del documento

```
<!-- Esto es un comentario -->
```

Elementos XML

- Un documento XML está formado por uno o varios elementos
- Cada elemento está delimitado por una etiqueta inicial y una etiqueta final (start tag, end tag)
- Una etiqueta inicial está formada por el signo de menor, el nombre y el signo de mayor
- Una etiqueta final está formada por el signo de menor, una barra (slash), el nombre y el signo de mayor
- No puede haber espacios ni a la izquierda ni dentro del nombre. Solo se admiten después del nombre
- El nombre puede usar cualquier carácter unicode
- El nombre puede incluir el guión, la barra baja y números, excepto en la primera posición

Ejemplo correcto:

```
<holamundo></holamundo>
```

Ejemplos incorrectos

```
< holamundo></ holamundo>
```

```
<hola mundo></hola mundo>
```

```
<1mundo></1mundo>
```

Ejemplos correctos

```
<holamundo ></holamundo >
```

```
<holamundo2></holamundo2>
```

```
<holamundo_2></holamundo_2>
```

Es frecuente que un elemento contenga una lista de elementos, con la misma etiqueta

```
<grupo>
    <alumno>
        Juan González
    </alumno>
    <alumno>
        María Fernández
    </alumno>
</alumno>
```

Numerar los elementos normalmente no tendrá sentido. Aunque sintácticamente puede ser válido, estamos forzando la creación de elementos distintos

```
<grupo>
    <!-- ¡¡MAL EJEMPLO!! -->
    <alumno1>
        Juan González
    </alumno1>
    <alumno2>
        María Fernández
    </alumno2>
</alumno>
```

Anidamiento de los elementos

- Los elementos se pueden anidar hasta cualquier nivel.
- El elemento raiz de un documento XML es el elemento de mayor nivel jerárquico. Tiene que haber exactamente uno
- El elemento raiz puede tener uno o más elementos anidados (nunca solapados), que se pueden anidar hasta cualquier nivel
- Los elementos están ordenados, se garantiza que el orden se mantiene
- Dentro de los elementos hay *character data*, normalmente llamado simplemente *text*, texto.

```
<holamundo>
    <hola_europa>
        <hola_españa>
            Texto de ejemplo del elemento hola_españa
        </hola_españa>
        <hola_portugal>
        </hola_portugal>
    </hola_europa>
    <hola_asia>
        Texto de ejemplo del elemento hola_asia
    </hola_asia>
</holamundo>
```

Etiquetas autocerradas

Cuando un elemento no tiene texto, hay dos alternativas posibles

- Usar una etiqueta de cierre y otra de apertura

  ```
  <holamundo></holamundo>
  ```

- Usar una etiqueta *auto cerrada*

  ```
  <holamundo/>
  ```

  Signo de menor, nombre, barra, signo de mayor

Atributos

Un elemento puede tener *atributos*

```
<log date="2016-02-07 17:01:05+00:00" lang="es">
No hay actividad
</log>
```

- Los atributos están dentro de la etiqueta inicial, separados por espacios
- Un atributo es un par formado por un nombre y un valor
- El nombre del atributo no se puede repetir dentro del mismo elemento. Sí puede aparece el mismo nombre de atributo en un elemento distinto
- A continuación del nombre va el signo igual y el valor, entre comillas simples o dobles
- El valor es texto
- Los atributos no están ordenados, no hay garantía de que se mantenga el orden

Una misma información puede presentarse o bien como texto o bien como atributo, las siguientes cuatro formas son correctas y equivalentes

- ```
  <log>
      <date>
          2015-07-10 21:01:05+00:00
      </date>
      <lang>
          es
      </lang>
      <text>
          Pass
      </text>
  </log>
  ```

- ```
  <log date="2015-07-10 21:01:05+00:00" lang="es" text="Pass">
  </log>
  ```

- ```
  <log date="2015-07-10 21:01:05+00:00" lang="es">
      Pass
  </log>
  ```

- ```
  <log date="2015-07-10 21:01:05+00:00" lang="es" text="Pass"/>
  ```

Usar un enfoque y otro es decisión del diseñador, no hay reglas fijas pero sí algunas recomendacions

- Es preferible usar atributos para información breve, de una sola pieza
- Es preferible usar texto dentre de un nuevo elemento para valores de cierta complejidad. Por ejemplo un mensaje de cierta longitud, una dirección, una descripción, etc
- En caso de duda, se recomienda el atributo

Cuando una información conste de varias unidades que pueden repetirse, es obligado usar elementos, porque los atributos no pueden repetirse

- Ejemplo correcto:

```
<cliente>
    <telefono>
        91 00 0000
    </telefono>
    <telefono>
        91 11 1111
    </telefono>
</cliente>
```

- ¡¡INCORRECTO!!:

```
<cliente
    telefono="91 00 0000"
    telefono="91 11 1111"
/>
```

Esto sí sería correcto

```
<cliente
    telefono1="91 00 0000"
    telefono2="91 11 1111"
/>
```

Pero tiene inconvenientes

- Es más complicado
- No sirve para 3 teléfonos

Naturalmente, también es correcto que un nombre de atributo se repita en un elemento distinto

```
<cliente
    telefono="91 00 0000"
/>

<cliente
    telefono="91 11 1111"
/>

<proveedor
    telefono="91 22 2222"
/>
```

Gramáticas

Un documento XML puede incluir una referencia a una gramática. La gramática indica qué etiquetas, qué atributos y qué texto se permiten en un documento XML.

- Originalmente la gramática se indicaba mediante DTD, *Document Type Definition*
  La referencia al DTD se coloca entre el prólogo y el elemento raiz
- En la actualidad es más habitual emplear XSD, *(XML Schema Definition)*
  La referencia al XSD se indica como atributo del elemento raíz

Referencia a un DTD

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- the XHTML document body starts here-->
<html xmlns="http://www.w3.org/1999/xhtml">
 ...
</html>
```

Fuente: wikipedia

Ejemplo de gramática XSD

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Address">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Recipient" type="xs:string" />
        <xs:element name="House" type="xs:string" />
        <xs:element name="Street" type="xs:string" />
        <xs:element name="Town" type="xs:string" />
        <xs:element name="County" type="xs:string" minOccurs="0" />
        <xs:element name="PostCode" type="xs:string" />
        <xs:element name="Country" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="DE" />
              <xs:enumeration value="ES" />
              <xs:enumeration value="UK" />
              <xs:enumeration value="US" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Documento XML referenciando al XSD anterior

```
<?xml version="1.0" encoding="utf-8"?>
<Address xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="SimpleAddress.xsd">
  <Recipient>Mr. Walter C. Brown</Recipient>
  <House>49</House>
  <Street>Featherstone Street</Street>
  <Town>LONDON</Town>
  <PostCode>EC1Y 8SY</PostCode>
  <Country>UK</Country>
</Address>
```

Fuente:wikipedia

ElementTree: procesamiento de XML desde python

En python hay varias librerías para procesar xml

- Tal vez la más habitual y casi *estándar* es ElementTree
- También se puede usar cElementTree, cuya API es idéntica, pero es más eficiente por estar implementada en C
- Otra librería interesante, más avanzada, es lxml. Tiene un API similar a ElementTree, con más funcionalidad

ejemplo:biblioteca.xml

```
<?xml version="1.1"?>
<biblioteca>
    <libro isdn="978-1-118-16213-2"  edicion="5" fecha="july 2012" editorial="Wiley" >
        <autor>
            Joe Fawcett
        </autor>
        <autor>
            Danny Ayers
        </autor>
        <autor>
            Liam R. E. Quin
        </autor>
        <titulo>
            Beginning XML
        </titulo>
    </libro>
    <libro isdn="978-1-484202-03-6"   edicion="1" fecha="march 2015" editorial="Apress" >
        <autor>
            Ben Smith
        </autor>
        <titulo>
            Beginning JSON
        </titulo>
    </libro>
</biblioteca>
```

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
nombre_fichero="biblioteca.xml"

def imprime_elemento(elemento):
    print "etiqueta:",elemento.tag
    print "atributos:",elemento.attrib
    print "texto:", elemento.text
    print "elementos incluidos en ",elemento.tag
    for subelemento in elemento:
        imprime_elemento(subelemento)

def main():
    arbol=ET.ElementTree(file=nombre_fichero)
    root=arbol.getroot()
    imprime_elemento(root)

if __name__ == "__main__":
    main()
```

```
etiqueta: biblioteca
atributos: {}
texto:

elementos incluidos en  biblioteca
etiqueta: libro
atributos: {'fecha': 'july 2012', 'edicion': '5', 'editorial': 'Wiley', 'isdn': '978-1-118-16213-2'}
texto:

elementos incluidos en  libro
etiqueta: autor
atributos: {}
texto:
            Joe Fawcett

elementos incluidos en  autor
etiqueta: autor
atributos: {}
texto:
            Danny Ayers

elementos incluidos en  autor
etiqueta: autor
atributos: {}
texto:
            Liam R. E. Quin

elementos incluidos en  autor
etiqueta: titulo
atributos: {}
texto:
            Beginning XML

[...]
```

Este programa produce un resultado idéntico

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET

nombre_fichero="biblioteca.xml"
def main():
    root=ET.ElementTree(file=nombre_fichero).getroot()
    for elemento in root.iter():
       print "etiqueta:",elemento.tag
       print "atributos:",elemento.attrib
       print "texto:", elemento.text

if __name__ == "__main__":
    main()
```

Para leer desde la entrada estándar

```
import sys
    [...]
    root=ET.ElementTree(file=sys.stdin).getroot()
    [...]
```

A la función iter() se le puede añadir un parámetro como filtro, para que solo devuelva los elementos con cierta etiqueta

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
nombre_fichero="biblioteca.xml"

def main():
    root=ET.ElementTree(file=nombre_fichero).getroot()
    for elemento in root.iter("libro"):
       print "etiqueta:",elemento.tag
       print "atributos:",elemento.attrib
       print "texto:", elemento.text

if __name__ == "__main__":
    main()
```

Modificación del documento

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
nombre_fichero="biblioteca.xml"

def main():
    root=ET.ElementTree(file=nombre_fichero).getroot()
    for elemento in root.iter("titulo"):
       print "etiqueta:",elemento.tag
       print "atributos:",elemento.attrib
       elemento.text=elemento.text.upper()
       print "texto:", elemento.text
    nuevo_arbol=ET.ElementTree(root)
    x=ET.tostring(root, encoding="utf-8", method="xml")
    print x

if __name__ == "__main__":
    main()
```

Creación de nuevos documentos

- Para crear un elemento raiz, invocamos el método `Element`, pasando como argumento su etiqueta

```
root = ET.Element(u"holamundo")
```

- Para crear un elemento, invocamos el método `SubElement`, pasando como primer argumento el elemento raiz, y como segundo elemento la etiqueta

```
elemento=ET.SubElement(root, u"hola_europa")
```

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
def main():
    root = ET.Element(u"holamundo")
    root.attrib={u"fecha":u"febrero 2016"}
    root.text=u"¡Hola, mundo!"

    elemento=ET.SubElement(root, u"hola_europa")
    atributos={u"fecha":u"marzo 2016"}
    elemento.attrib=atributos
    elemento.text=u"¡Hola, Europa!"

    elemento=ET.SubElement(root, u"hola_asia")
    elemento.text=u"¡Hola, asia!"
    atributos={u"fecha":u"marzo 2016"}
    elemento.attrib=atributos

    print ET.tostring(root, encoding="utf-8",  method="xml")

if __name__ == "__main__":
    main()
```

```
<holamundo fecha="febrero 2016">
    ¡Hola, mundo!
    <hola_europa fecha="marzo 2016">
        ¡Hola, Europa!
    </hola_europa>
    <hola_asia fecha="marzo 2016">
        ¡Hola, asia!
    </hola_asia>
</holamundo>
```