



estándares XML → DTD XSD XSL XPPM

Introducción

Lenguaje de Marcado → Sist. para anotar docs (importa el orden)

SGML → Standard Generalized Markup Language

- Extensor para definir lenguajes de marcado para doc.
- Info estructural y semántica
- No define la presentación del documento

Extensión

XML → Extensible Markup language

- Leng. de marcado con etiquetas
- Tienen contenido
- No describen la forma de mostrar el contenido
- No tienen hijos

Reglas XML

• Attributo: Atributo adicional asociada a una etiqueta

- Formato: nombre-atributo="valor"

- El valor debe ir siempre entre las comillas (" ") o apóstrofes ('')

- Cada elemento solo puede tener un atributo con el mismo nombre

- Siempre tiene un valor asociado

- `<name value="100">` </name>

• Comentario: no se procesa

- Entre <!-- -->

- No puede contener -->

• Se distinguen mayúsculas de minúsculas

• Todas las etiquetas abren y cierran

- <book> </book>

- <category> </category>

• Evitando caracteres especiales

- < -> less-than menor que

- > -> greater-than mayor que

- & -> & amperand el

- ' -> apostrophe apóstrofe

- " -> double-quote comillas

• Nombre de las etiquetas y atributos:

- Case sensitive

- Compuesto de caracteres Unicode

- No se permite: <?S&A;?> /> <?@!> />

- No puede contener espacios

- No puede empezar por -, o número

XML prolog

- Opcional

• Si existe, tiene que estar al principio, antes del elemento raíz

- No es parte del documento XML

- No es un error que no tenga etiqueta de cierre

<xml version="1.0" encoding="UTF-8">

- Define versión XML

- Define la codificación de caracteres

• Estructura en árbol

- Document Object Model (DOM)

- Elemento raíz (root)

- Tiene que ser único

- Todos los elementos, excepto el raíz, tienen padre (parent)

- Los elementos pueden tener hijos (child)

- Los elementos que no tienen hijos son las hojas (leaves)

- Dos elementos que comparten padre son hermanos (sibling)

Elementos vs ATR en XML

```
<notes>
  <note>
    <to>reindeer</to>
    <from>jon</from>
    <heading>Reindeer</heading>
    <body>Don't forget me this weekend!</body>
  </note>
</notes>
```

X no Pueden contener valores
X no se pueden expandir en Árbol
X no Pueden describir estructuras
✓ significado ✓ Rendimiento

Modelo Relacional vs XML

Books						
	id	Title	Author	Year	Price	cat_id
1	Everyday	Gladia De Laurentiis	2005	30.00	1	1
2	Serv	Peter Roowing	2005	29.99	2	1
3	Learning	Erik T. Ray	2003	39.95	3	1

Categories	
id	name
1	coding
2	children
3	web

Cada ATR → Una tabla

Modelo relacional	XML
Estructura	Tablas Jerárquica en árbol
Esquema	Fijado con antelación Flexible Auto-descriptivo
Consultas	Sencillas Intuitivas Algo más complicadas
Ordenación	Ninguna Implicita
Mejor para	Ordenador Humanos Almacenamiento Streaming

XML bien formado

- 1 solo nivel Raiz
- Etiquetas emparchadas
- Anidamiento apilado
- At. Unicas de cada elem
- Correct: adecuados

• Se utilizan lenguajes de reglas para transformar XML en HTML:
- CSS (Cascading Style Sheets): Hojas de estilo en cascada
- XSL (extensible Stylesheet Language): Lenguaje de hojas de estilo extensible

• Pasa previamente por el parser



XHTML valido

Documento bien formado → Estructuralmente ✓
Documento valido → u formato + Requisitos ?Contenido



Especificados por:

DTD : Document Type Definition
XSD : XML Schema Definition

DTD y XSD

DTD

Descriptor de tipo de documento

Extender para validar XML
Proporciona una gramática para especificar:

ERIM ATR Anidado Ordenación N° caracteres

```
<!DOCTYPE root-element >
<ELEMENT root-element (...)>
<ELEMENT element (...)>
<!-- Resto de elementos-->
<ATTLIST element attributename attribute-type
attribute-value ...>
<!-- Resto de atributos-->
]>
```

Elementos DTD

1) !DOCTYPE root-element []>

- root-element:
- Nombre del elemento raíz
- Tendrá que estar definido a continuación

<ELEMENT element-name category >

- element-name: Nombre del elemento
- category:
 - EMPTY: Vacío.
 - (PCDATA): Parsed Character Data. Texto que se va a parsear.
 - ANY: Cualquier combinación de datos parseables.
 - (child1 child2...):
 - Los hijos tienen que aparecer en el mismo orden
 - Si puedes especificar el número de ocurrencias:
 - * o more
 - * or more
 - ? or 1
 - Si no se especifica, implica uno y sólo uno
- Se puede usar | para permitir una elección

Ejemplo :

```
<ELEMENT name (#PCDATA) --> <name>Anas</name>
<ELEMENT email (<from>,<to>,<body>)>
  <email>
    <from></from>
    <to></to>
    <body></body>
  </email>
```

Atributos DTD

2) !ATTLIST element-name attribute-name attribute-type attribute-value [attribute-name attribute-type attribute-value]>

- element-name: Nombre del elemento al que pertenece
- attribute-name: Nombre del atributo
- attribute-type:
 - CDATA: Character Data. Todo lo que se va a parsear
 - (entity): Solo se permiten esos valores específicos
 - ID
 - IDREF
 - attribute-value:
 - REQUIRED: El atributo tiene que estar
 - IMPLIED: El atributo es optional
 - [attribute-type attribute-type attribute-value]
- attribute-type: Especifica el tipo de dato
- Se pueden definir más atributos en la misma sintaxis
- También se pueden definir en otra etiqueta ATTLIST

Ejemplo :

```
Archivo dtd_ejemplo1.xml:
<!DOCTYPE >
<!ELEMENT note SYSTEM "dtd_ejemplo1.dtd" >
<note>
  <to>John</to>
  <from>Diana</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend.</body>
</note>
```

```
Archivo dtd_ejemplo2.xml:
<!DOCTYPE note SYSTEM "dtd_ejemplo2.dtd" >
<note>
  <to>John</to>
  <from>Diana</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend.</body>
</note>
```

```
Archivo note.dtd:
<ELEMENT note (to,from,heading,body)>
<ELEMENT to (#PCDATA)>
<ELEMENT from (#PCDATA)>
<ELEMENT heading (#PCDATA)>
<ELEMENT body (#PCDATA)>
```

```
Archivo dtd_ejemplo4.xml:
<!DOCTYPE >
<!ELEMENT Bookstore [
  <ELEMENT Bookstore (Book, Author)*>
  <ELEMENT Book (Title, Author)*>
  <ELEMENT BookRef (BookID, BookRef)*>
]>
<ELEMENT BookRef (BookID REQUIRED)
  Price #CDATA REQUIRED
  Authors #PCDATA REQUIRED>
```

```
Archivo dtd_ejemplo4.xml (cont.):
<ELEMENT Book (Title, Author)*>
<ELEMENT BookRef (BookID REQUIRED)
  Price #CDATA REQUIRED
  Authors #PCDATA REQUIRED>
<ELEMENT Title (#PCDATA)>
<ELEMENT Author (#PCDATA)>
<ELEMENT BookRef (#PCDATA)>
<ELEMENT BookRef (BookID REQUIRED)
  Price #CDATA REQUIRED
  Authors #PCDATA REQUIRED>
<ELEMENT BookRef (BookID REQUIRED)
  Price #CDATA REQUIRED
  Authors #PCDATA REQUIRED>
<ELEMENT BookRef (BookID REQUIRED)
  Price #CDATA REQUIRED
  Authors #PCDATA REQUIRED>
<ELEMENT BookRef (BookID REQUIRED)
  Price #CDATA REQUIRED
  Authors #PCDATA REQUIRED>
]>
```

XQuery no lo vemos con encabezado
*SLT

Xpath

DOM (Document Object Model) → Recorriendo como en árbol

Xpath ↗ Recorrido

- sintaxis = a la de los directorios
- Usado en otros entornos (como XSLT)
- Recomendación W3C



Construcciones básicas para rutas

- /: Separador que implica por / referencia el nodo raíz
- //: Yo, y cualquier elemento descendiente
- /Book: Etiquetas
- @ISBN: Atributos
 - Se obtiene su valor con /@ISBN
- ::: OR lógico
- *: Comodín
- ../../: Ir un nivel hacia atrás

Ejemplo:

```
<xml version="1.0" encoding="UTF-8"?>
<class>
  <student>1.1.</name>
  <student id="035">
    <nameMarks></name>
    <year>1999</year>
  </student>
  <student id="007">
    <name>Ana</name>
    <year>1998</year>
  </student>
</class>
```

```
student
  • Todos los nodos con el nombre student
  /class
  • Elemento raíz
  //name
  • Todos los elementos name
  independientemente de donde están
  /class/student/name
  • Todos los nombres de los alumnos
  //@id
  • Todos los atributos con nombre id
```

Predicados

- []: Separador del predicado
 - Se pueden anidar para agrupar condiciones
- <, >, =, !=: Comparadores
- AND: AND lógico, enlaza condiciones
- OR: OR lógico, enlaza condiciones
- [número]: Contador, se empieza a contar en 1
- [@attr]: Selecciona los atributos de nombre attr
- [@attr='value']: Atributo attr con valor value

```
<xml version="1.0" encoding="UTF-8"?>
<class>
  <student id="035">
    <nameMarks></name>
    <year>1999</year>
  </student>
  <student id="007">
    <name>Ana</name>
    <year>1998</year>
  </student>
</class>
```

```
• El segundo estudiante?
  /class/student[2]
  • ¿Estudiante con id 035?
  /class/student[@id="035"]
  • Estudiantes del año >= 1999
  /class/student[year>=1999]
  • student con algún atributo
  /class/student[@*]
```

TS: XML

Fixes incluidos en Xpath

- se pueden usar en los predicados
- Ejemplos
 - Contiene: contains(elemento, "texto")
 - Contador: count()
 - Contenido del nodo: text()
 - Último elemento: last()

```
<xml version="1.0" encoding="UTF-8"?>
<class>
  <student id="035">
    <name>Ana</name>
    <year>1999</year>
  </student>
  <student id="007">
    <name>Ana</name>
    <year>1998</year>
  </student>
</class>
```

Jugando con Xpath

- incluye 13 distintos
- Ejemplos
 - Predicado parent::
 - Hermano que le precede preceding-sibling::
 - Hermano que le sigue following-sibling::
 - Descendentes descendant::
 - Propio etiqueta self::
 - ... hasta 12 diferentes...

Ejemplos

Partiendo del archivo xml_ejemplo1.xml

- Todos los títulos de libro:
/bookstore/book/title
- Todos los nombres de los alumnos
//@id
- Todos los atributos con nombre id:
//@*

Partiendo del archivo xml_ejemplo1.xml

- Título de los libros con un precio mayor que 35:
/bookstore/book[price>35]/title
- Todos los títulos con un atributo "lang" con valor "en":
/title[@lang="en"]
- Todos los elementos título con al menos un atributo:
/title[@*]

Nota: Para trabajar con archivos grandes
vamos a usar SAX

→ Tener java instalado
→ Cargar el XML: doc ('hamlet.xml')

XQuery

- lenguaje para realizar consultas XML
- es el sucesor de SLT
- cada ejecución crea sobre y devuelve una secuencia de elementos:
 - Documento XML o Stream XML
- XQuery es uno de los tipos de expresiones q. secuencia.

XSLT

- Extensible StyleSheet language w/ Transformation
- se ejecuta utilizando XML
- estructura el documento en nodos: elem, attrs, texto, comentarios,...
- Sirve para encontrar partes de un documento XML (usando Xpath) y reemplazarlos por otras
- Sirve conforme una plantilla y reemplaza el resultado entero.
- Pueden aplicarse recursivamente
- Usa construcciones típicas de los lenguajes de programación:
 - condicionales (if-else)
 - iteraciones (for-each)
- Al usarlo se debe tener cuidado en:
 - comportamientos extraños con los espacios en blanco
 - prioridad estricta de las plantillas

Open API

JSON puede utilizar para definir APIs

```
components:
  schemas:
    book:
      type: object
      properties:
        id:
          type: integer
        title:
          type: string
        author:
          type: string
```

```
<book>
  <id>0</id>
  <title>string</title>
  <author>string</author>
</book>
```

```
<book id="0">
  <type>object
  <properties>
    <id:
      type: integer
      title:
        type: string
        attribute: true
      author:
        type: string
    </id>
  </properties>
</book>
```

```
books:
  type: array
  items:
    type: string
    xml: name="item"
    wrapped: true
    name: books-array
  example:
    - "one"
    - "two"
    - "three"
```

```
<books-array>
  <item>one</item>
  <item>two</item>
  <item>three</item>
</books-array>
```

Node.js

Node.js → No pensado para trabajar con XML → tiene librerías
carga cosas y las convierte a JSON

en general no validan XML builder → no permite manipular con Xpath XSLT XQuery SODA

Nota: RSS

Un RSS (Really Simple Syndication) es un formato que permite distribuir y recibir actualizaciones de contenido web de forma automática. Se utiliza principalmente en blogs, noticias o sitios que publican contenido con frecuencia.

Funciona a través de un archivo XML que contiene los títulos, resúmenes y enlaces de las últimas publicaciones. Los usuarios pueden suscribirse a este archivo mediante un lector RSS para ver las novedades sin tener que visitar cada sitio web.