

XML

Una introducción práctica.

javierj@lsi.us.es

www.lsi.us.es/~javierj/

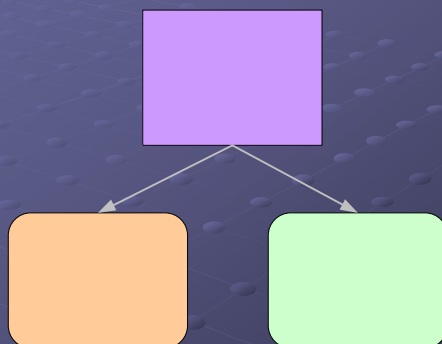
Índice.

- Introducción a XML.
- Estudiando XML.
- DTD's y Schemas.
- Manipulación de XML.

¿Qué es XML?.



¿De dónde viene XML?



XML vs HTML

```
<html>
<head>
  <title>Hello John</title>
</head>

<body bgcolor="#FFFFFF"
      text="#000000">
  <p> Hello John. </P>
</body>
</html>
```



XML vs HTML

```
<html>
<head>
  <title>Hello John</title>
</head>

<body bgcolor="#FFFFFF"
      text="#000000">
  <p> Hello John. </P>
</body>
</html>
```

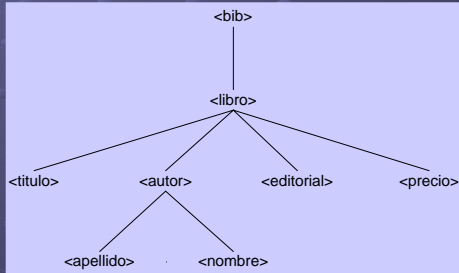
Es el título de la página

Es un párrafo

HTML no es XML !!

XML como un árbol

```
<bib>  
  <libro>  
    <titulo>TCP/IP Illustrated</titulo>  
    <autor>  
      <apellido>Stevens</apellido>  
      <nombre>W.</nombre>  
    </autor>  
    <editorial>Addison-Wesley</editorial>  
    <precio> 65.95</precio>  
  </libro>  
</bib>
```



Estudiando XML

XML es un metalenguaje

Ojo, XML no es un lenguaje de etiquetas, sino un conjunto de reglas para definir lenguajes de etiquetas (como XHTML).

XML me dice como crear mis propias etiquetas.

Aplicaciones de XML

¿Conoces alguna situación donde se aplique XML?.

¿Se te ocurre alguna situación donde se pueda aplicar XML?.

XML en el mundo real

- Frameworks de desarrollo (Struts, Spring, etc..)
- OpenOffice.
- Configuración de aplicaciones: Tomcat.
- Estándares de contenidos en e-Learning.
- Servicios web.
- Definición de interfaces gráficas (XUL).
- Sistemas de publicación de contenidos.
- Bases de datos.

Un ejemplo: RSS

- Really Simple Syndication: formato de sindicación de contenidos.
- Lenguaje de etiquetas construido a partir de XML.

```
<rss version="0.92">
<channel>
  <title>Dave Winer: Grateful Dead</title>
  <link>http://www.scripting.com/blog/categories/gratefulDead.html
  </link>
  <description>
    A high-fidelity Grateful Dead song every day.
  </description>
  <lastBuildDate>Fri, 13 Apr 2001 19:23:02 GMT</lastBuildDate>
  <docs>http://backend.userland.com/rss092</docs>
  <managingEditor>dave@userland.com (Dave Winer)</managingEditor>
  <webMaster>dave@userland.com (Dave Winer)</webMaster>
  <cloud domain="data.ourfavoritesongs.com" port="80" path="/RPC2"
    registerProcedure="ourFavoriteSongs.rssPleaseNotify" protocol="xml-rpc"/>
  <item>
    <description> It's been a few days since I added a song to the Grateful Dead channel.
    </description>
    <enclosure url="http://www.scripting.com/mp3s/weatherReportDicksPicsVol7.mp3"
      length="6182912" type="audio/mpeg"/>
  </item>
  ...
</channel>
</rss>
```


Términos en XML

```
<direccion>
  <nombre>
    <titulo>Mrs.</titulo>
    <nombre> Mary </nombre>
    <apellidos>McGoon</apellidos>
  </nombre>
  <calle> 1401 Main Street </calle>
  <ciudad estado="NC">Anytown</ciudad>
  <!-- Lo que ponga aquí es ignorado
        por el parser.
        Include <a> marcas </a>-->
</direccion>
```

Diagram illustrating XML terms:

- Etiqueta** (Tag): Points to the opening and closing tags of an element, such as `<titulo>` and `</titulo>`.
- Elemento** (Element): Points to the entire structure between the opening and closing tags, such as `<nombre>...`.
- Atributo** (Attribute): Points to the `estado="NC"` attribute within the `<ciudad>` tag.
- Comentario** (Comment): Points to the `<!-- ... -->` comment block.

Reglas sintácticas

- Una raíz.
- Anidación de las etiquetas.
- Sensible a mayúsculas.
- Atributos no vacíos y entrecomillados.
- Reglas para los nombres de etiquetas y atributos.

Caracteres no permitidos

&	&
<	<
>	>
'	'
"	"
O bien utilizar CDATA	<![CDATA[Texto sin limitaciones]]>

Ejercicio

```
<?xml version="1.0"?>
<X> Hola Mundo! </X>
<X> Hola Mundo! </X>
```

1

```
<?xml version="1.0"?>
<A>
  <B>
    <C> 1 </C>
  </B>
  <C> 2 </C>
</A>
```

2

```
<?xml version="1.0"?>
<A>
  <B>
    <C> 1 </B>
  </C>
  <C> 2 </C>
</A>
```

3

```
<?xml version="1.0"?>
<P>
  <X A="23" />
  <X B="24" />
</P>
```

4

```
<?xml version="1.0"?>
<A ID="1">
  <B X="xyz">
    <C C=45> 1 </C>
  </B>
</A>
```

5

```
<?xml version="1.0"?>
<X> Hola Mundo! </X>
```

6

Un ejercicio

¿Qué similitudes y diferencias existen entre almacenar información en una BBDD y en XML?.

Nombre	Apellidos	Calle	Ciudad
Mary	McGoon	1401 Main Street	Anytown
...

```
<direccion>
  <nombre>
    <titulo>Mrs.</titulo>
    <nombre> Mary </nombre>
    <apellidos>McGoon</apellidos>
  </nombre>
  <calle> 1401 Main Street </calle>
  <ciudad estado="NC">Anytown</ciudad>
</direccion>
```

Diferencias entre una BBDD y XML.

- Una BBDD es binaria, XML es texto plano.
- Una BBDD, por ejemplo una tabla, es muy dependiente de SGBD, un XML no.
- En una tabla no importa el orden, en XML sí.
- BBDD estructura relacional, XML estructura jerárquica.

¿Cuándo utilizar etiquetas y cuándo atributos?

- No hay un conjunto de normas claras.
- En general, manda la experiencia.
- Podemos vivir sin ellos.
- Los atributos no tienen orden.

Un ejemplo

Sin atributos

```
<mensajes>
<mensaje>
  <tipo> Informativo </tipo>
  <texto> Reunión a las 12 </texto>
</mensaje>
<mensaje>
  <tipo> Solicitud </tipo>
  <texto> Informe 9A </texto>
</mensaje>
</mensajes>
```

Con atributos

```
<mensajes>
  <mensaje tipo="Informativo">
    Reunión a las 12
  </mensaje>
  <mensaje tipo="Solicitud">
    Necesito el informe 9A
  </mensaje>
</mensajes>
```

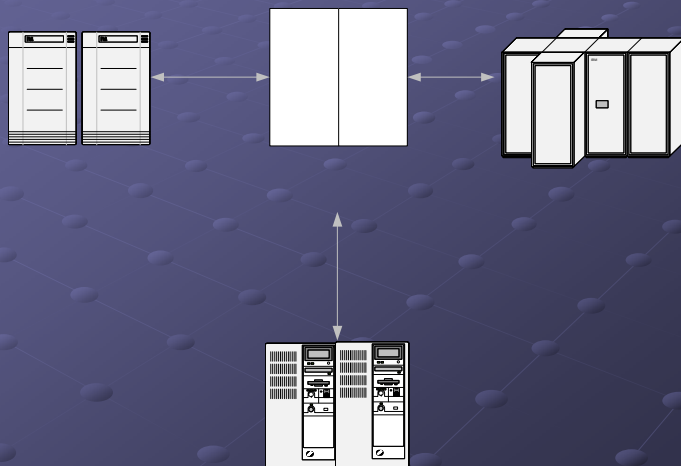
Aún hay mucho más.

- Instrucciones de procesado.
- Entidades.
- Namespaces.
- Caracteres de escape

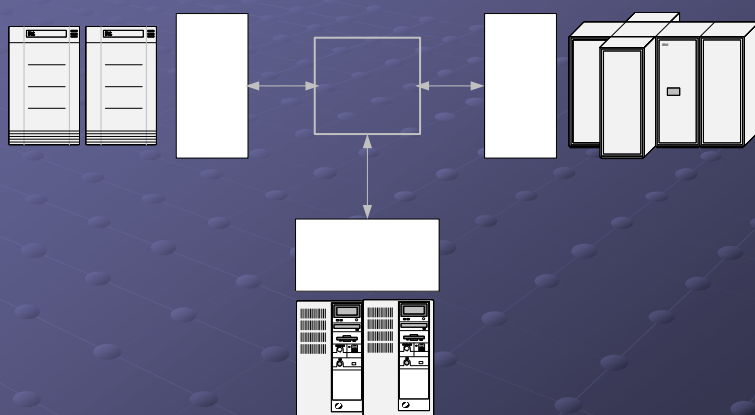
En resumen

- Lenguaje textual.
- XML no especifica ninguna etiqueta.
- Es estricto.

Un ejemplo



Un ejemplo



Un ejercicio

Queremos guardar información de un libro: su título, sus autores y el año de edición.

```
<libro>
<titulo> Libro 1 </titulo>
<autores>
<autor> Uno </autor>
<autor> Dos </autor>
</autores>
<año> 1999 </año>
</libro>
```

```
<libro año="1999">
<titulo> Libro 1 </titulo>
<autores>
<autor nombre="Uno">
<autor nombre="Dos">
</autores>
</libro>
```

¿Tengo que poner siempre, al menos un autor, o si no conozco los autores lo puedo dejar en blanco el año?

```
<autor>
<autores> Libro 1 </autores>
<titulo>
<libro> Uno </libro>
<libro> Dos </libro>
</titulo>
<año> 1999 </año>
</autor>
```

Otro ejemplo.

Un juego de texto donde cada habitación tiene un identificador, una descripción y sus salidas.

```
<ROOM ID="2.1">
<DESCRIPTION>
Te encuentras en un habitáculo exactamente igual al anterior. En el centro de la habitación ves un poste de color blanco de aproximadamente un metro de alto. También ves un pasillo que nace en la pared del fondo.
</DESCRIPTION>
<EXITS>
<EXIT ID="NORTH" TO="2.2"/>
</EXITS>
</ROOM>
```

```
<HABITACION ID="2.1">
<DESCRIPCION>
Te encuentras en un habitáculo exactamente igual al anterior. En el centro de la habitación ves un poste de color blanco de aproximadamente un metro de alto. También ves un pasillo que nace en la pared del fondo.
</DESCRIPCION>
<SALIDA>
<DIRECCION> norte </DIRECCION>
<HACIA>2.2</HACIA>
</SALIDA>
</HABITACION>
```

Unas notas sobre Namespaces

Tengo una lista de clientes y otras de empleados y quiero guardar en un documento XML los clientes y el empleado que los atiende.

¿Cómo distinguir <nombre> del cliente de <nombre> del empleado?.

Namespaces

- Un namespace es un ámbito de definición de etiquetas.
- Para cada etiqueta hemos de indicar, además su namespace.
- Los namespaces deben declararse antes de utilizarse.

Namespaces. Un ejemplo

```
<?xml version="1.0"?>
```

```
<clientes-empleados
```

```
  xmlns:cliente="http://www.una-url.com/clientes/"
```

```
  xmlns:empleado="http://www.otra-url.com/empleados/">
```

```
    <cliente:nombre>XYZ</cliente:nombre>
```

```
    ...
```

```
    <empleado:nombre>XYZ</empleado:nombre>
```

```
    ...
```

```
</clientes-empleados>
```

Definimos un namespace. Debe ser única y se suele utilizar una URL.

Son dos etiquetas distintas.

DTDs y Schemas.

Un juego de etiquetas comunes

`<cliente>`, `<nombre>`,
`<apellido>`, `<dirección>`

1

```
<cliente>
  <nombre> Uno </nombre>
  <nombre> Dos </nombre>
</cliente>
```

2

```
<nombre>
  <cliente> Uno </cliente>
  <cliente> Dos </cliente>
</nombre>
```

Tipos de documentos XML

- Documento inválido.
- Documento bien formado.
- Documento válido.

Definición de la estructura de un documento

DTD's y Schemas.

- Para definir la estructura de un documento utilizamos otro documento.
- Ambas sirven para lo mismo.
- Estructura.
 - Que etiquetas pueden aparecer.
 - Como se combinan esas etiquetas.
- DTD en XML.
- Schema, su propia especificación

Un ejemplo

```
<!ELEMENT cliente (persona-contacto,  
                    direccion, ciudad)>  
<!ELEMENT persona-contacto (titulo?, nombre,  
                             apellidos)>  
  
<!ELEMENT titulo (#PCDATA)>  
<!ELEMENT nombre (#PCDATA)>  
<!ELEMENT apellidos (#PCDATA)>  
<!ELEMENT direccion (#PCDATA)>  
<!ELEMENT ciudad (#PCDATA)>
```

Documento válido

```
<cliente>  
  <persona-contacto>  
    <titulo> Don </titulo>  
    <nombre> Juan </nombre>  
    <apellidos> Nadie Nadie </apellidos>  
  </persona-contacto>  
  <direccion> X </direccion>  
  <ciudad> Y </ciudad>  
</cliente>
```

```
<cliente>  
  <persona-contacto>  
    <nombre> Juan </nombre>  
    <apellidos> Nadie Nadie </apellidos>  
  </persona-contacto>  
  <direccion> X </direccion>  
  <ciudad> Y </ciudad>  
</cliente>
```

D.T.D.s

- Declaración de tipo de documento.
- Declaración de elementos.
- Declaración de atributos.
- Declaración de notaciones.
- Declaración de entidades.

Nos centraremos en estas dos.

D.T.D.s

Estas líneas nos dan mucha información.

```
<!ELEMENT cliente (persona-contacto,  
                    direccion, ciudad)>  
<!ELEMENT persona-contacto (titulo?, nombre,  
                             apellidos)>  
<!ELEMENT titulo (#PCDATA)>  
<!ELEMENT nombre (#PCDATA)>  
<!ELEMENT apellidos (#PCDATA)>  
<!ELEMENT direccion (#PCDATA)>  
<!ELEMENT ciudad (#PCDATA)>
```

Un elemento <cliente> contiene una <persona-contacto>, una <dirección> y una <ciudad>. Todos estos elementos deben aparecer y deben hacerlo en ese mismo orden.

Un elemento <persona-contacto> contiene un elemento <titulo> opcional, seguido de un elemento <nombre> y de un elemento <apellidos>.

Todos estos elementos contienen texto. No puede incluirse ningún elemento en ellos.

Ejercicio

```
<cliente>
  <persona-contacto>
    <nombre> n </nombre>
    <apellidos> p >
  </persona-contacto>
  <direccion> d </direccion>
  <ciudad> ciudad </ciudad>
</cliente>
```

1

```
<cliente>
  <persona-contacto>
    <nombre> Juan </nombre>
    <nombre> Pablo </nombre>
    <apellidos> p </apellidos>
  </persona-contacto>
  <direccion> d </direccion>
  <ciudad> ciudad </ciudad>
</cliente>
```

3

```
<cliente>
  <persona-contacto>
  </persona-contacto>
  <ciudad> ciudad </ciudad>
  <direccion> d </direccion>
</cliente>
```

2

```
<cliente>
  <persona-contacto>
    <nombre> Juan </nombre>
    <apellidos>
      <primero> X </primero>
      <segundo> y </segundo>
    </apellidos>
  </persona-contacto>
  <direccion> d </direccion>
  <ciudad> ciudad </ciudad>
</cliente>
```

4

```
<cliente>
  <persona-contacto>
    <nombre> Juan </nombre>
    <apellidos> X </apellidos>
    <direccion> d </direccion>
  </persona-contacto>
  <ciudad> ciudad </ciudad>
</cliente>
```

5

D.T.D.s para elementos.

- La coma indica una lista de elementos.
- La interrogación, indica que un elemento puede aparecer o no.
- El signo mas indica que un elemento puede repetirse cualquier número de veces pero debe aparecer, al menos, una vez.
- El asterisco indica que un elemento puede repetirse cualquier número de veces o ninguna.
- La barra vertical indica una lista de opciones de las que solo puede aparecer una.

D.T.D.s para atributos.

```
<!ELEMENT a (#PCDATA)>  
<!ATTLIST a b CDATA #REQUIRED  
           c CDATA (X|Y)  
           d CDATA "Valor por defecto">
```

El primer atributo puede tomar cualquier valor pero es obligatorio.

El valor del segundo atributo sólo puede ser X o Y.

El elemento a puede tener hasta 3 atributos.

Si no se indica, el tercer atributo vale "Valor por defecto".

Ejercicio

Escribir una DTD para el siguiente documento.

```
<!DOCTYPE LIBROS SYSTEM "libros.dtd">  
<LIBROS>  
  <LIBRO>  
    <TITULO>AutoSketch</TITULO>  
    <AUTOR>Ramón Montero</AUTOR>  
    <PRECIO>2.500</PRECIO>
```

```
<!ELEMENT LIBROS (LIBRO)+>  
<!ELEMENT LIBRO (TITULO,AUTOR,PRECIO)>  
<!ELEMENT TITULO (#PCDATA)>  
<!ELEMENT AUTOR (#PCDATA)>  
<!ELEMENT PRECIO (#PCDATA)>
```

```
</LIBRO>  
</LIBROS>
```

Conclusiones sobre las DTDs

- La sintaxis DTD es distinta de XML.
- Pobre soporte de namespaces.
- No permite crear nuevos tipos.
- No permiten establecer restricciones complejas.
- Son parte de XML.

Schemas

- Son documentos XML.
- Soportan tipos de datos.
- Soportan namespaces.
- Permiten definir nuevos tipos de datos y restricciones sobre los valores de un elemento.
- Ofrecen mayor precisión que los DTDs.

Schemas

```
<!ELEMENT cliente (persona-contacto direccion)
<!ELEMENT persona-contacto (titulo nombre apellidos direccion ciudad)
<!ELEMENT titulo (#PCDATA)
<!ELEMENT nombre (#PCDATA)
<!ELEMENT apellidos (#PCDATA)
<!ELEMENT direccion (#PCDATA)
<!ELEMENT ciudad (#PCDATA)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cliente">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="persona-contacto"/>
        <xsd:element ref="direccion"/>
        <xsd:element ref="ciudad"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="persona-contacto">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="titulo" minOccurs="0"/>
        <xsd:element ref="nombre"/>
        <xsd:element ref="apellidos"/>
        <xsd:element ref="direccion"/>
        <xsd:element ref="ciudad"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="titulo" type="xsd:string"/>
  <xsd:element name="nombre" type="xsd:string"/>
  <xsd:element name="apellidos" type="xsd:string"/>
  <xsd:element name="direccion" type="xsd:string"/>
  <xsd:element name="ciudad" type="xsd:string"/>
</xsd:schema>
```

Schemas

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <element name="nombre-y-apellidos">
    <complexType>
      <sequence>
        <element name="nombre" type="string"/>
        <element name="apellido1" type="string"/>
        <element name="apellido2" type="string"/>
      </sequence>
      <attribute name="sexo" type="string"/>
    </complexType>
  </element>
</schema>
```

Namespace del Schema.
Este es el namespace
recomendado en la
especificación.

Declaración de un
elemento.

Compuesto por una
secuencia de otros
elementos.

Con un atributo. Sólo los
elementos complejos
tienen atributos.

Schemas

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <element name="nombre-y-apellidos">
    <complexType>
      <sequence>
        <element name="nombre" type="string"/>
        <element name="apellido1" type="string"/>
        <element name="apellido2" type="string"/>
      </sequence>
      <attribute name="sexo" type="string"/>
    </complexType>
  </element>
</schema>
```

Elementos simples que sólo pueden contener texto.

Schemas

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <element name="nombre-y-apellidos">
    <complexType>
      <sequence>
        <element name="nombre" type="string"/>
        <element name="apellido1" type="string"/>
        <element name="apellido2" type="string"/>
      </sequence>
      <attribute name="sexo" type="string"/>
    </complexType>
  </element>
</schema>
```



```
<nombre-y-apellidos sexo="varon">
  <nombre> Juan </nombre>
  <apellido1> Perez </apellido1>
  <apellido2> Perez </apellido2>
</nombre-y-apellidos>
```

Tipos de datos.

- string
- decimal
- integer
- positiveInteger
- boolean

```
<producto id="1345" />
```



```
<xs:element name="producto">  
  <xs:complexType>  
    <xs:attribute name="id" type="xs:positiveInteger"/>  
  </xs:complexType>  
</xs:element>
```

Indicadores.

● Indicadores de orden.

- All
- Choice
- Sequence

```
<xs:element name="libro">  
  <xs:complexType>  
    <xs:choice>  
      <xs:element name="autor" type="xs:string"/>  
      <xs:element name="editor" type="xs:string"/>  
    </xs:choice>  
  </xs:complexType>  
</xs:element>
```

● Indicadores de cantidad.

- maxOccurs
- minOccurs
- Ilimitada: maxOccurs="unbounded"

Tipos propios

```
<xs:complexType name="persona">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:element name="empleado" type="persona">
<xs:element name="cliente" type="persona">
```

```
...
<empleado>
  <nombre> Yo </nombre>
  <apellidos> mismo </apellidos>
</empleado>
<cliente>
  <nombre> Yo </nombre>
  <apellidos> mismo </apellidos>
</cliente>
...
```



Extensiones de tipos

```
<persona edad="36">Juan Pérez</ persona >
```



```
<xs:element name="persona">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="edad" type="xs:integer" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Extensiones de tipos

```
<!-- Elementos comunes -->
<xs:complexType name="Tipo-Ficha">
  <xs:sequence>
    <xs:element name="titulo" type="xs:string" />
    <xs:element name="resumen" type="xs:integer"
      minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<!-- Elementos propios del libro -->
<xs:complexType name="Tipo-Libro">
  <xs:complexContent>
    <xs:extension base="Tipo-Ficha">
      <xs:sequence>
        <xs:element name="ISBN" type="xs:string" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Elemento libro -->
<xs:element name="libro" type="Tipo-Libro" />
```

Restricciones

El elemento edad puede valer entre 0 y 100

```
<xs:element name="edad">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

El elemento coche sólo puede ser Audi, Golf o BMW

```
<xs:element name="coche">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restricciones

Tres letras

```
<xs:element name="iniciales">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Un código postal de 5 dígitos

```
<xsd:element name="codigo-postal">
  <xsd:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xsd:simpleType>
</xsd:element>
```

Restricciones

Una cadena entre 5 y 8
Caracteres.

```
<xs:element name="clave">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ejercicios

```
<persona>
  <nombre> Yo </nombre>
  <apodo> uno </apodo>
  <apodo> dos </apodo>
  <apodo> tres </apodo>
</ persona >
```

Una persona que pueda tener tantos apodos como se desee, o ninguno.

Soluciones

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="empleado">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="apellidos" type="xs:string"/>
        <xs:element name="clave">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:length value="8"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<xs:element name="foto">
  <xs:complexType>
    <xs:sequence>
      <xs:attribute name="archivo" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Soluciones

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apodo" type="xs:string"
        maxOccurs="maxOccurs="unbounded"
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Ejercicio

```
<CHESSBOARD>
  <WHITEPIECES>
    <KING><POSITION COLUMN="G" ROW="1"/></KING>
    <BISHOP><POSITION COLUMN="D" ROW="6"/></BISHOP>
    <ROOK><POSITION COLUMN="E" ROW="1"/></ROOK>
    <PAWN><POSITION COLUMN="A" ROW="4"/></PAWN>
  </WHITEPIECES>
  <BLACKPIECES>
    <KING><POSITION COLUMN="B" ROW="6"/></KING>
    <PAWN><POSITION COLUMN="D" ROW="4"/></PAWN>
  </BLACKPIECES>
</CHESSBOARD>
```

- La única pieza obligatoria es el rey.
- El rey y la reina solo aparecen una vez, el resto de las piezas dos veces salvo el peón que aparece 8.

Solución

```
<?xml version="1.0" encoding="UTF-8"?>
xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="CHESSBOARD">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="WHITEPIECES"
          type="pieces" />
        <xsd:element name="BLACKPIECES"
          type="pieces" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="pieces">
    <xsd:sequence>
      <xsd:element name="KING" type="piece"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="QUEEN" type="piece"
        minOccurs="0" maxOccurs="1"/>
      <xsd:element name="BISHOP" type="piece"
        minOccurs="0" maxOccurs="2"/>
      <xsd:element name="ROOK" type="piece"
        minOccurs="0" maxOccurs="2"/>
      <xsd:element name="KNIGHT" type="piece"
        minOccurs="0" maxOccurs="2"/>
      <xsd:element name="PAWN" type="piece"
        minOccurs="0" maxOccurs="8"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="piece">
    <xsd:sequence>
      <xsd:element name="POSITION"
        minOccurs="1" maxOccurs="1">
        <xsd:complexType>
          <xsd:attribute name="COLUMN" use="required">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:pattern value="[A-H]"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
          <xsd:attribute name="ROW" use="required">
            <xsd:simpleType>
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:minInclusive value="1"/>
                <xsd:maxInclusive value="8"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

¿Existe ya un vocabulario de etiquetas para mi problema?

Probablemente sí:

- <http://www.oasis-open.org/cover/xml.html>
- <http://www.dublincore.org>
- <http://www.rosettanet.org>
- Google.
- Aplicaciones similares.



Manipulación de XML.

Tecnologías para trabajar con XML.

- API SAX.
- API DOM.
- Otros APIs
- Bindings.
- Serializadores.
- XSL y XSLT.
- XPath
- XQuery.
- Otras.

Simple API for XML

```
<persona>  
  <nombre> Yo </nombre>  
  <apodo> uno </apodo>  
  <apodo> dos </apodo>  
  <apodo> tres </apodo>  
</persona>
```

Comienza el
elemento
<nombre>

Programa

Termina el
elemento
<nombre>

Simple API for XML

- Orientado a eventos.
- Muy detallado.
- No construye nada en memoria.
- En tiempo real.
- Sin estado.

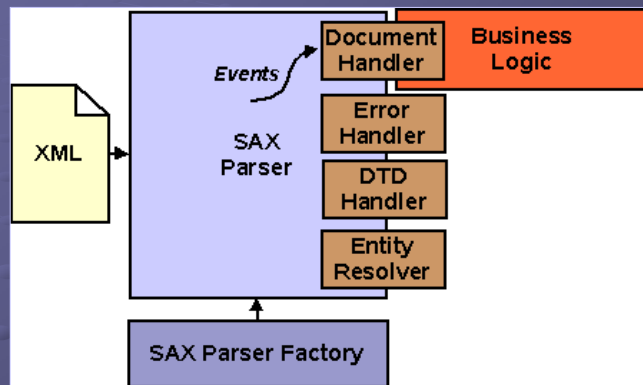
Simple API for XML. Ventajas

- Sencillo.
- Consume poca memoria.

Simple API for XML. Inconvenientes

- Complejo para documentos complejos.
- Más lento.
- No hay vuelta atrás.
- Poco reutilizable.

Un ejemplo



Un ejemplo

```

import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;

public class ChessboardSAXPrinter {
    private SAXParser parser;

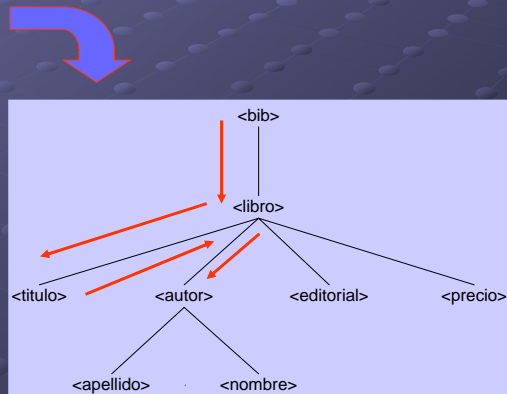
    public class ChessboardHandler extends HandlerBase {
        private boolean whitePiece = false;

        public void startElement(String name,
            AttributeList attrs) {
            if (name.equals("WHITEPIECES")) {
                whitePiece = true;
            } else if (name.equals("BLACKPIECES")) {
                whitePiece = false;
            } else if (name.equals("KING") || name.equals("QUEEN") || name.equals("BISHOP") ||
                name.equals("ROOK") || name.equals("KNIGHT") || name.equals("PAWN")) {
                System.out.print((whitePiece ? "White" : "Black") + " " + name.toLowerCase() + ": ");
            } else if (name.equals("POSITION")) {
                if (attrs != null) {
                    System.out.print(attrs.getValue("COLUMN"));
                    System.out.println(attrs.getValue("ROW"));
                }
            }
        }
        return;
    }
}

```

Document Object Model

```
<bib>
<libro>
<titulo>TCP/IP Illustrated</titulo>
<autor>
<apellido>Stevens</apellido>
<nombre>W.</nombre>
</autor>
<editorial>Addison-Wesley</editorial>
<precio> 65.95</precio>
</libro>
</bib>
```



Document Object Model

- Construye un árbol en memoria.
- Ofrece una interfaz para recorrer el árbol

Document Object Model. Ventajas

- Más rápido.
- Ofrece una interfaz para recorrer el árbol

Document Object Model. Inconvenientes

- Todo el documento en memoria.
- Puede consumir mucha memoria.
- Puede tardar más en arrancar.

Un ejemplo

```
public void print(String fileName)
    throws SAXException, IOException {

    Document document = builder.parse(fileName);

    NodeList positions
    = document.getElementsByTagName("POSITION");
    for (int i = 0; i < positions.getLength(); i++) {
        Element position = (Element) positions.item(i);
        Element piece = (Element) position.getParentNode();
        Element pieces = (Element) piece.getParentNode();
        out.println(
            (pieces.getTagName().equals("WHITEPIECES")
             ? "White " : "Black ")
            + piece.getTagName().toLowerCase() + ": "
            + position.getAttribute("COLUMN")
            + position.getAttribute("ROW"));
    }
    return;
}
```

Conclusiones

- SAX es útil cuando tenemos limitaciones de memoria o para desarrollar rápidamente con documentos sencillos.
- DOM ofrece el máximo control y flexibilidad.
- Son herramientas básicas.
- En Java: SUN o Xerces (xml.apache.org).

Otras APIs

- Implementaciones propias de cada lenguaje.
- En Java:
 - JDom (<http://www.jdom.org>).
 - Dom4J (<http://www.dom4j.org>).
 - XOM (<http://www.cafeconleche.org/XOM/>).
- Una comparativa:
<http://www.dom4j.org/compare.html>

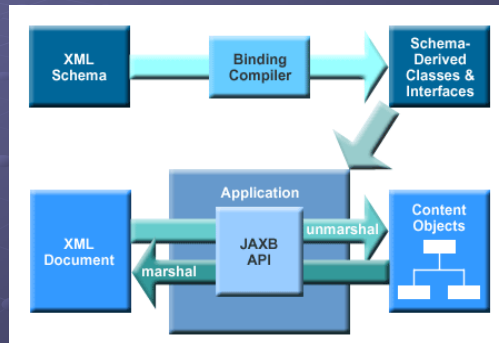
XML Binding

- ¿Qué es?
 - Vincular un conjunto de clases con un conjunto de estructuras XML.
 - Crear objetos a partir del contenido de las estructuras.
 - Y viceversa.

XML Binding

● Tres elementos característicos.

- Generación de clases.
- Marshalling / unmarshalling.
- Vinculación de Schemas.



XML Binding

● Herramientas (en java).


- JAXB <http://java.sun.com/xml/jaxb/>
- Castor <http://castor.exolab.org>
- XMLBeans <http://xmlbeans.apache.org/>
- Una buena comparativa:
<https://bindmark.dev.java.net/current-results.html>
- La mejor: <http://www.manageability.org/polls/tool-for-xml-binding>

XMLBinding. Herramientas.

Castor	http://www.castor.org/
Xgen	http://www.commerceone.com/developers/docsoapxdk
Breeze	http://www.breezefactor.com/
Zeus	http://zeus.objectweb.org/
XMLBeans	http://xml.apache.org/xmlbeans/

Un ejemplo.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns="Faenor.Global"
  targetNamespace="Faenor.Global"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="GLOBAL">
    <xs:complexType>
      <xs:all>
        <xs:element name="ROOM" type="xs:string"/>
        <xs:element name="WHOAMI" type="xs:string"/>
        <xs:element name="INTRO" type="xs:string"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GLOBAL xmlns="Faenor.Global"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.imsproject.org/xsd/ims_cp_rootv1lp2
    http://www.imsproject.org/xsd/ims_cp_rootv1lp2.xsd
    http://www.imsproject.org/xsd/ims_md_rootv1p1
    http://www.imsproject.org/xsd/ims_md_rootv1p1.xsd">

  <ROOM>1</ROOM>
  <WHOAMI> Eres Korbe, el joven hijo de unos granjeros.
</WHOAMI>
  <INTRO> Has elegido un magnifico día soleado de primavera para
    emprender tu camino. </INTRO>

</GLOBAL>
```

Un ejemplo.

```
<xs:all>
  <xs:element name="ROOM" type="xs:string"/>
  <xs:element name="WHOAMI" type="xs:string"/>
  <xs:element name="INTRO" type="xs:string"/>
</xs:all>
</xs:complexType>
</xs:element>
</xs:schema>
```

```

//// This file was generated by the Java™ Architecture for XML
Binding(JAXB) Reference Implementation, vl.0 package
Adventure.XMLData.Global.impl;

```

```
public class GLALImpl
    extends Adventure.XMLData.Global.impl.GLOBALTypeImpml
        implements Adventure.XMLData.Global.GLOBAL,
            com.sun.xml.bind.RIElement,
            com.sun.xml.bind.unmarshaller.UnmarshallableObject,
            com.sun.xml.bind.serializer.XMLSerializable,
            com.sun.xml.bind.validator.ValidatableObject
{
    private final static com.sun.msv.grammar.Grammar
schemaFragment =
com.sun.xml.bind.validator.SchemaDeserializer.deserialize("  0  ac  0ed  0000  0005sr  0000'com.sun.msv.grammar.trex.ElementP  ttern  0000  0000  0000  0000  0000  0000  0000  0001  0002  000  0000011  0000  nameClass  0000  001fcom/sun/msv/grammar/Na  meClass;xr  0000  00lecom.com.sun.msv.grammar.ElementExp  0000  0  0  0000  0000  0000  0000  0001  0002  0000  0002Z  000  0001gnoreUndeclaredAttributesL  0000  contentModelL  0000 ");

    public java.lang.String ____jaxb_r1____getNamespaceURI() {
        return "Faenor.Global";
    }

    public java.lang.String ____jaxb_r1____getLocalName() {
        return "GLOBAL";
    }

    private final static java.lang.Class
PRIMARY_INTERFACE_CLASS() {
        return Adventure.XMLData.Global.GLOBAL.class;
    }
}

//...

}
```

Un ejemplo

XML Binding. Consejos.

- Aplicarlo siempre que tengamos un Schema bien definido que no vaya a cambiar.
- Cuando solo necesitemos los datos del XML.
- Cuando el código generado nos sirve.
- Puede presentar los mismos problemas que DOM.

Serializadores

- Hay quien los incluye dentro de las herramientas de Binding.
- Su objetivo es el mismo pero sin vínculos con Schemas.
- No generan el código.
- Son más rápidos de aplicar pero menos potentes.

Serializadores

- XStream <http://xstream.codehaus.org/>
- Betwixt <http://jakarta.apache.org/commons/betwixt/>
- Probablemente haya alguno más.

Un ejemplo

```
<person>
  <firstname>Joe</firstname>
  <lastname>Walnes</lastname>
  <phone>
    <code>123</code>
    <number>1234-456</number>
  </phone>
  <fax>
    <code>123</code>
    <number>9999-999</number>
  </fax>
</person>
```



```
public class Person {
  private String firstname;
  private String lastname;
  private PhoneNumber phone;
  private PhoneNumber fax;
  // ...
}

public class PhoneNumber {
  private int code;
  private String number;
  // ...
}
```

```
XStream xstream = new XStream();

// Necesita un parser DOM
XStream xstream = new XStream(new DomDriver());

// Es opcional. XStream podría adivinarlo él solo (?)
xstream.alias("person", Person.class);
xstream.alias("phoneNumber", PhoneNumber.class);

Person newJoe = (Person)xstream.fromXML(xml);
```


eXtensible Stylesheet Language

- Familia de recomendaciones para la definición de transformaciones y presentaciones de documentos XML.

- XSLT. Transformaciones.
- XPath. Language de expresiones.
- XSL-FO. Presentaciones.

eXtensible Stylesheet Language Transformations

- Basada en una especificación:

<http://www.w3.org/TR/xslt>

- Permite especificar transformaciones de XML.
- Podemos transformar una estructura XML en otras estructura XML, HTML o crear un PDF.
- 1.0 oficial y 2.0 borrador

Un ejemplo

```
<source>
<title>XSL</title>
<author>Alan</author>
</source>
```

```
<xsl:stylesheet version = '1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
  <h1>
    <xsl:value-of select="//title"/>
  </h1>
  <h2>
    <xsl:value-of select="//author"/>
  </h2>
</xsl:template>
</xsl:stylesheet>
```

```
<h1>XSL</h1>
<h2>John Smith</h2>
```

Aplicaciones

- Generar páginas web.
- Convertir unas estructuras XML en otras (compatibilidad).
- Crear PDFs.
- Crear código.

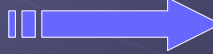
Plantillas

```
<xsl:stylesheet version = '1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:template match="bold">
    <p> <b>
      <xsl:value-of select="."/>
    </b> </p>
  </xsl:template>
</xsl:stylesheet>
```

Se aplica sobre el elemento <bold>

Escribe el valor de este elemento.

<bold> Frase </bold>



<p>
 Frase
</p>

Plantillas

```
<xsl:stylesheet version = '1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:template match="bold">
    <p> <b>
      <xsl:value-of select="."/>
    </b> </p>
  </xsl:template>
</xsl:stylesheet>
```

Expresiones XPath

XPath

- Lenguaje de expresión utilizado para referenciar nodos de información en un conjunto de datos XML.
- También es un estándar del W3C.
- Similar a expresiones regulares para nodos XML.

XPath. Ejemplos.

<code>/html/body/h1</code>	Selecciona todos los nodos <code><h1></code> que son hijos de un nodo <code><body></code> que, a su vez, es hijo de un nodo <code><html></code> que es el nodo raíz
<code>//h1</code>	Selecciona todos los nodos <code><h1></code> que aparezcan en cualquier posición del árbol XML. La doble barra indica cualquier profundidad.
<code>count(//libro)</code>	Devuelve el número de nodos <code><libro></code> que aparecen en el documento en cualquier posición.
<code>//libro[autor = "Hunter"]</code>	Devuelve todos los nodos <code><libro></code> que aparezcan en el documento en cualquier posición y que tengan un nodo hijo <code><autor></code> con el valor "Hunter". Los corchetes indican un filtro para seleccionar los resultados que cumplan una determinada condición.
<code>//libro[@año > 1999]</code>	Devuelve todos los nodos <code><libro></code> que tengan un atributo "año" con un valor superior a 1999. La arroba indica que "año" no es un hijo (una etiqueta) sino un atributo de la etiqueta libro.
<code>(i b)</code>	Devuelve todos los nodos <code><i></code> o todos los nodos <code></code> que encuentre en el nodo contexto. Por defecto el nodo contexto es el nodo raíz del documento.

XPath. Ejemplos.

<code>doc("libros.xml")/bib/libro/autor[1]</code>	La expresión anterior devuelve solo el primero nodo autor que encuentre para cada nodo libro.
<code>//key[. = "Tiempo total"]</code>	Devuelve todos los nodos <key> que tengan un valor de "Tiempo total.". El carácter "." representa el nodo contexto, lo cual tiene una función similar al operador "this" en lenguajes como C++ o Java.
<code>(//key)[1]/texto</code>	Devuelve los nodos <texto> del primer nodo <key> del documento.

Ejercicio.

```
<bib>
  <libro año="1994">
    <titulo>TCP/IP Illustrated</titulo>
    <autor>
      <apellido>Stevens</apellido>
      <nombre>W.</nombre>
    </autor>
    <editorial>Addison-Wesley</editorial>
    <precio> 65.95</precio>
  </libro>

  <libro año="1992">
    <titulo>Advan Programming for Unix environment</titulo>
    <autor>
      <apellido>Stevens</apellido>
      <nombre>W.</nombre>
    </autor>
    <editorial>Addison-Wesley</editorial>
    <precio>65.95</precio>
  </libro>

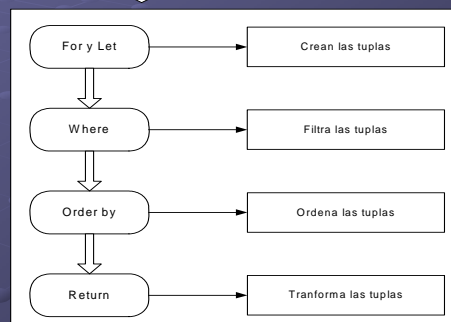
  ....
</bib>
```

XQuery

- Una evolución de XPath.
- Podemos considerar a XQuery como el SQL para XML.

XQuery.

Datos XML



Resultado XML



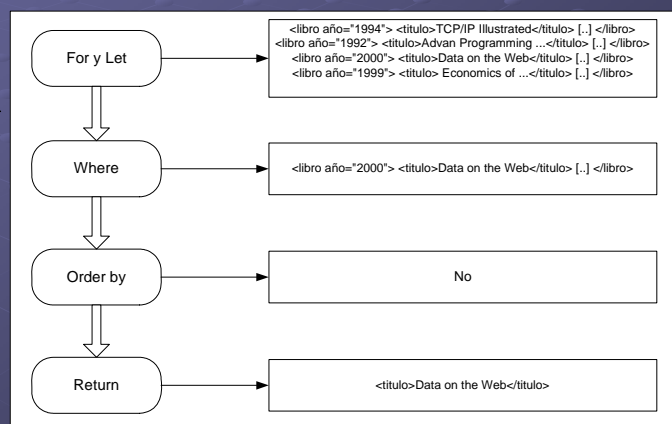
XQuery

● Tres aplicaciones principales.

- Recuperar datos de documentos XML.
- Transformar unas estructuras en otras.
- Transformaciones XML->XHTML.

Ejemplo

```
for $b in doc("libros.xml")//libro
where $b/año = "2000"
return $b/titulo
```



Ejemplo

```
for $b in doc("libros.xml")//libro
let $c := $b/autor
return
<libro>{ $b/titulo,
  <autores>{ count($c) }</autores>}
</libro>
```

Esta consulta devuelve el título de cada uno de los libros de archivo "libros.xml" junto con el número de autores de cada libro



```
<libro>
  <titulo>TCP/IP Illustrated</titulo>
  <autores>1</autores>
</libro>
<libro>
  <titulo>Advanced Programming in the UNIX Environment</titulo>
  <autores>1</autores>
</libro>
```

Ejemplo

```
<libro año="1994">
  <titulo>TCP/IP Illustrated</titulo>
  <autor>
    <apellido>Stevens</apellido>
    <nombre>W.</nombre>
  </autor>
  <editorial>Addison-Wesley</editorial>
  <precio>65.95</precio>
</libro>
```

```
<libro año="1992">
  <titulo>Programming for Unix env.</titulo>
  <autor>
    <apellido>Stevens</apellido>
    <nombre>W.</nombre>
  </autor>
  <editorial>Addison-Wesley</editorial>
  <precio>65.95</precio>
</libro>
```



```
<html><head><title></title>
<body><table>
{
  for $b in doc("libros.xml")//libro
  return
  <tr><td><!-- { string( $b/titulo ) } -->
    </td></tr>
}
</table></body>
</html>
```

```
<html><head><title></title>
<body>
<table>
<tr><td><!-- TCP/IP Illustrated --></td></tr>
<tr><td><!-- Advan Programming for Unix environment --></td></tr>
<tr><td><!-- Data on the Web --></td></tr>
<tr><td><!-- Economics of Technology for Digital TV --></td></tr>
</table>
</body>
</html>
```



XQuery. Enlaces.

- XML Query Use Cases.

<http://www.w3.org/TR/xmlquery-use-cases>

- XQuery 1.0: An XML Query Language.

<http://www.w3.org/TR/XQuery/>

- XML Path Language (XPath) 2.0.

<http://www.w3.org/TR/xpath20/>

XQuery. Herramientas.

- XQEngine.

<http://xqengine.sourceforge.net/>

- BumbleBee.

<http://www.XQuery.com/bumblebee/>

- Qexo.

<http://www.gnu.org/software/qexo/>

- Qizx/open.

<http://www.xfra.net/qizxopen/>

- Saxon.

<http://saxon.sourceforge.net/>

Un ejemplo de código

```
import com.fatdog.xmlEngine.ResultList;
import com.fatdog.xmlEngine.XQEngine;
import com.fatdog.xmlEngine.exceptions.*;
import javax.xml.parsers.*;
import org.xml.sax.XMLReader;

public class EjemploXQEngine
{
    public static void main(String[] args)
    {
        String query = "<bib> { "
            + "   for $b in doc(\"libros.xml\")/bib/libro"
            + "   where $b/editorial = \"Addison-Wesley\" and $b/@año > 1991 "
            + "   return "
            + "   <libro> { $b/titulo } </libro>"
            + " } </bib> ";

        XQEngine engine = new XQEngine();
        SAXParserFactory spf = SAXParserFactory.newInstance();
        try {
            SAXParser parser = spf.newSAXParser();
            XMLReader reader = parser.getXMLReader();
            engine.setXMLReader( reader );
            engine.setDocument( "libros.xml" );
            ResultList results = engine.setQuery( query );
            System.out.println( results.emitXml() );
        } catch( Exception e ) {
            e.printStackTrace();
        }
    }
}
```

Otros

- Existen muchas más herramientas para trabajar con XML.
 - Jakarta commons XML-IO.
 - Jakarta commons Digester.
 - Etc...