eXtensible Markup Language - XML

Sommario

- Introduzione
- Strutturazione dei Dati
- XML Namespaces
- Document Type Definition (DTD) e Schema
- Altri linguaggi di markup
- Document Object Model (DOM) e Metodi DOM
- Simple API for XML (SAX)
- Extensible Stylesheet Language (XSL)
- Web Resources

Obiettivi

- Capire XML.
- Gestire dati mediante XML.
- Relazioni tra DTD, Schemi e XML.
- Parsing DOM-based e SAX-based.
- Concetti del namespace XML.
- Creare semplici documenti XSL.

Introduzione (1)

- XML (Extensible Markup Language):
  - È un'estensione di HTML che permette di definire nuovi elementi
- Derivato da Standard Generalized Markup Language (SGML)
- Tecnologia "Open" per scambiare e memorizzare dati
- Creato e standardizzato da W3C (1998)

Introduzione (2)

- Permette di creare altri linguaggi di markup per descrivere dati in modo strutturato
  - XML è stato progettato per descrivere dati con tag non predefiniti
- Il markup creato via XML può gestire vari tipi di informazione
  - XHTML
  - MathML – CML (Chemical Markup Language)
  - VoiceXML – SMIL (Synchronous Multimedia Integration Language)
  - …

Introduzione (3)

- Documenti XML
  - Sono collezioni gerarchiche di elementi
  - Contengono solo dati, senza informazioni relative alla loro formattazione
  - Altamente portabile
- I dati nel doc XML sono elaborati da parser, che
  - verificano la correttezza del doc XML
  - permettono ad altri programmi di elaborare i dati

Introduzione (4)

- I parser supportano principalmente due tecnologie
  - Document Object Model (DOM)
  - Simple API XML (SAX)
- La struttura dei dati XML è definita mediante
  - Document Type Definition (DTD)
  - Schema

Schema Generale

- Secondo un modello producer/consumer, un file XML
  - è prodotto da un'applicazione producer,
    - secondo quanto specificato dal dtd o dallo schema
  - è inviato a destinazione
  - dove un'applicazione consumer
    - valida la conformità al dtd/schema
    - estrae i dati mediante DOM o SAX

Strutturazione dei Dati (1)

- Ogni documento XML contiene la XML declaration, con il Value `version`, che indica la versione di XML a cui il documento è conforme
- È buona pratica di programmazione includere sempre la versione
- I dati in un doc XML sono strutturati ad albero

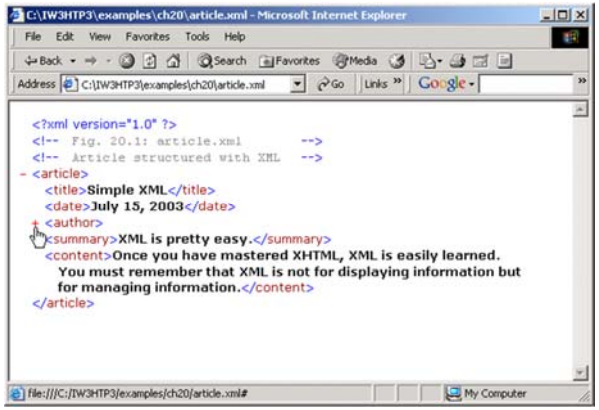
Strutturazione dei Dati (2)

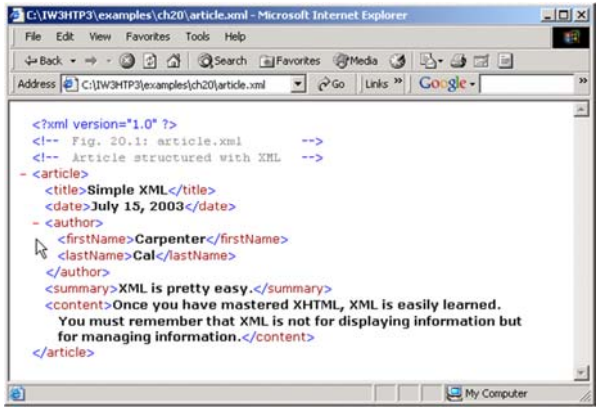
- In ogni doc XML c'è uno ed un solo Root element che racchiude tutti gli altri
- Si chiama
  - Container element ogni elemento che ne contiene altri
  - Child element ogni elemento all'interno di un elemento container

Strutturazione dei Dati (3)

- L'elemento `flag` è vuoto
  - Non contiene alcun testo
- Documenti DTD
  - Definiscono le regole strutturali di un documento XML
  - Sono file con estensione `.dtd`


```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.1: article.xml      -->
4  <!-- Article structured with XML -->
5
6  <article>
7
8     <title>Simple XML</title>
9
10    <date>July 15, 2003</date>
11
12    <author>
13       <firstName>Carpenter</firstName>
14       <lastName>Cal</lastName>
15    </author>
16
17    <summary>XML is pretty easy.</summary>
18
19    <content>Once you have mastered XHTML, XML is easily
20       learned. You must remember that XML is not for
21       displaying information but for managing information.
22    </content>
23
24 </article>
```

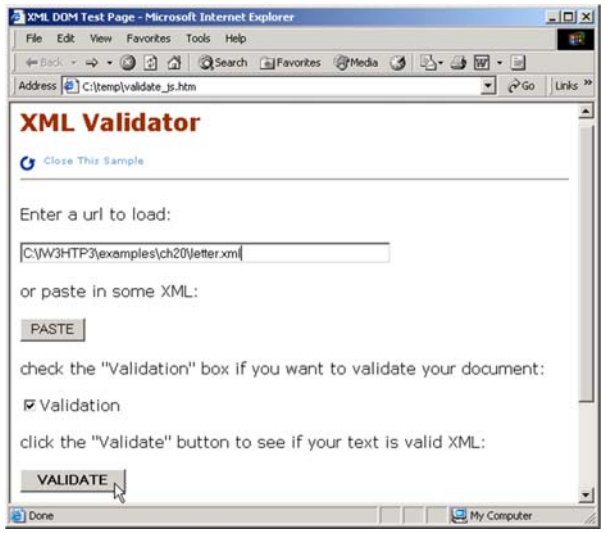


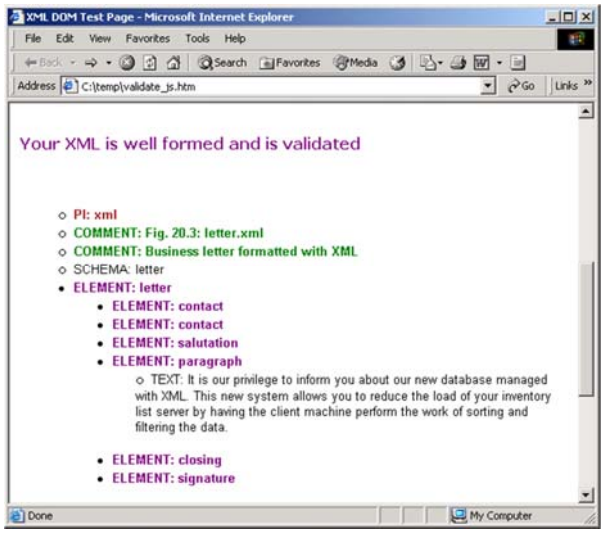



```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.3: letter.xml              -->
4  <!-- Business letter formatted with XML  -->
5
6  <!DOCTYPE letter SYSTEM "letter.dtd">
7
8  <letter>
9
10    <contact type = "from">
11       <name>John Doe</name>
12       <address1>123 Main St.</address1>
13       <address2></address2>
14       <city>Anytown</city>
15       <state>Anystate</state>
16       <zip>12345</zip>
17       <phone>555-1234</phone>
18       <flag gender = "M"/>
19    </contact>
20
21    <contact type = "to">
22       <name>Joe Schmoe</name>
23       <address1>Box 12345</address1>
24       <address2>15 Any Ave.</address2>
25       <city>Othertown</city>
```



```
26       <state>Otherstate</state>
27       <zip>67890</zip>
28       <phone>555-4321</phone>
29       <flag gender = "M"/>
30    </contact>
31
32    <salutation>Dear Sir:</salutation>
33
34    <paragraph>It is our privilege to inform you about our new
35       database managed with XML. This new system allows
36       you to reduce the load of your inventory list server by
37       having the client machine perform the work of sorting
38       and filtering the data.</paragraph>
39    <closing>Sincerely</closing>
40    <signature>Mr. Doe</signature>
41
42 </letter>
```





XML Namespaces (1)

- XML permette di creare elementi personalizzati
- Tale capacità potrebbe portare a problemi relativi al Naming collision
  - Lo stesso nome è usato per indicare elementi diversi prodotti da diversi autori; se un documento usa due di questi elementi omonimi si ha naming collision

XML Namespaces (2)

- XML namespace
  - Collezione di nomi di elementi e attributi per disambiguare elementi con nomi in collisione
- Strumento: Uniform resource identifier (URI)
  - Identifica unicamente il namespace
  - È una stringa di testo usata per differenziare i nomi
  - Due tipi principali di URI: URN (Uniform resource name) e URL (Uniform resource Locator)

XML Namespaces (3)

- Schema di URN

`urn:serie-di-nomi` (separati dal simbolo ‘:’)

- Alternativa: uso di una URL
  - Garantisce l’univocità grazie all’univocità della URL
- URI può essere qualsiasi stringa, ad eccezione del namespace riservato `xml`
- Nell’esempio seguente `Directory` è l’elemento radice che contiene tutti gli altri

XML Namespaces (4)

Esempio: namespace per differenziare l’elemento file riferito a file di testo e file immagini

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.4 : namespace.xml -->
4  <!-- Demonstrating Namespaces  -->
5
6  <text:directory xmlns:text = "urn:deitel:textInfo"
7     xmlns:image = "urn:deitel:imageInfo">
8
9     <text:file filename = "book.xml">
10       <text:description>A book list</text:description>
11    </text:file>
12
13    <image:file filename = "funny.jpg">
14       <image:description>A funny picture</image:description>
15       <image:size width = "200" height = "100"/>
16    </image:file>
17
18 </text:directory>
```

Elemento root che contiene gli altri elementi

Attributo per creare prefissi di namespace e assegnare uri


```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.5 : defaultnamespace.xml -->
4  <!-- Using Default Namespaces         -->
5
6  <directory xmlns = "urn:deitel:textInfo"
7     xmlns:image = "urn:deitel:imageInfo">
8
9     <file filename = "book.xml">
10       <description>A book list</description>
11    </file>
12
13    <image:file filename = "funny.jpg">
14       <image:description>A funny picture</image:description>
15       <image:size width = "200" height = "100"/>
16    </image:file>
17
18 </directory>
```


Document Type Definition (DTDs) e Schema

- Due modalità per specificare la struttura dei documenti XML
  - Document Type Definition (DTD)
  - Schema

Document Type Definition (1)

- Permette
  - di verificare se il documento XML è valido
  - a gruppi indipendenti di utenti di controllare la struttura e lo scambio dei dati in forma standardizzata

Document Type Definition (2)

- Esprime un set di regole per la struttura usando la grammatica Extended Backus Naur Form (EBNF)
- `ELEMENT` type declaration
  - Definisce le regole
- `ATTLIST` attribute-list declaration
  - Definisce un attributo


```
1  <!-- Fig. 20.6: letter.dtd       -->
2  <!-- DTD document for letter.xml -->
3
4  <!ELEMENT letter ( contact+, salutation, paragraph+,
5     closing, signature )>
6
7  <!ELEMENT contact ( name, address1, address2, city, state,
8     zip, phone, flag )>
9  <!ATTLIST contact type CDATA #IMPLIED>
10
11 <!ELEMENT name ( #PCDATA )>
12 <!ELEMENT address1 ( #PCDATA )>
13 <!ELEMENT address2 ( #PCDATA )>
14 <!ELEMENT city ( #PCDATA )>
15 <!ELEMENT state ( #PCDATA )>
16 <!ELEMENT zip ( #PCDATA )>
17 <!ELEMENT phone ( #PCDATA )>
18 <!ELEMENT flag EMPTY>
19 <!ATTLIST flag gender (M | F) "M">
20
21 <!ELEMENT salutation ( #PCDATA )>
22 <!ELEMENT closing ( #PCDATA )>
23 <!ELEMENT paragraph ( #PCDATA )>
24 <!ELEMENT signature ( #PCDATA )>
```

•Il simbolo + indica che ci possono essere più occorrenze dell'elemento

•Se si vuole indicare l'opzionalità di un attributo si usa:

- **•? Per indicare un elemento che può presentarsi 0 o 1 volta**
- **•* Per indicare un elemento che può presentarsi 0, 1 o più volte**

Specifica che se il parser trova un elemento contact senza l'attributo type, allora può scegliere un valore arbitrario per l'attributo oppure può ignorarlo. In entrambi i casi il documento sarà valido

Valori dell'attributo type (1)

- CDATA: dati carattere
  - Il parser non elabora il dato ma lo passa all'applicazione senza modificarlo
- #PCDATA: Parser Character Data (cioè testo)
  - Non deve contenere caratteri di markup (quali <, >, &)
  - Devono essere sostituiti dall'autore con le corrispondenti entità (rispettivamente <, >, &)

Valori dell'attributo type (2)

- Empty: elemento vuoto
- Gli attributi type possono essere specificati come:
  - #IMPLIED
  - #REQUIRED
  - #FIXED

W3C XML Schema Documents (1)

- Introducono flessibilità non permessa dai DTD
  - Ad es., permettono di definire il tipo di un elemento
- Specificano la struttura del doc XML, mediante sintassi XML
  - Non usano la grammatica EBNF
- Possono essere manipolati come altri documenti XML
- Richiedono un parser per la validazione

W3C XML Schema Documents (2)

- XML schemas
  - Vocabolario dello schema definito dal W3C
- Schema valido
  - Documento XML conforme a uno schema
  - File con estensione `.xsd`

W3C XML Schema Documents (3)

- Elemento radice `schema`
  - Contiene elementi che definiscono la struttura del documento XML
  - `targetNamespace`
    - Namespace del vocabolario XML definito dallo schema
  - Tag `element`
    - Definisce l'elemento che deve essere incluso nella struttura del documento XML
  - Attributi `name` e `type`
    - Specificano il nome e il tipo di dato di un elemento

W3C XML Schema Documents (4)

- Tipi predefiniti
  - `date`
  - `int`
  - `double`
  - `time`
  - etc

W3C XML Schema Documents (5)

- Due categorie di tipi di dati
  - Semplici: non contengono attributi né elementi figlio
  - Complessi: possono contenere attributi e elementi figlio


```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.7 : book.xml       -->
4  <!-- Book list marked up as XML -->
5
6  <deitel:books xmlns:deitel = "http://www.deitel.com/booklist">
7     <book>
8        <title>XML How to Program</title>
9     </book>
10    <book>
11       <title>C How to Program</title>
12    </book>
13    <book>
14       <title>Java How to Program</title>
15    </book>
16    <book>
17       <title>C++ How to Program</title>
18    </book>
19    <book>
20       <title>Perl How to Program</title>
21    </book>
22 </deitel:books>
```



```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.8 : book.xsd           -->
4  <!-- Simple W3C XML Schema document -->
5
6  <schema xmlns = "http://www.w3.org/2001/XMLSchema"
7          xmlns:deitel = "http://www.deitel.com/booklist"
8          targetNamespace = "http://www.deitel.com/booklist">
9
10    <element name = "books" type = "deitel:BooksType"/>
11
12    <complexType name = "BooksType">
13       <sequence>
14          <element name = "book" type = "deitel:SingleBookType"
15                   minOccurs = "1" maxOccurs = "unbounded"/>
16       </sequence>
17    </complexType>
18
19    <complexType name = "SingleBookType">
20       <sequence>
21          <element name = "title" type = "string"/>
22       </sequence>
23    </complexType>
24
25 </schema>
```

```
Target: file:///usr/local/XSV/xsvlog/@11038.1uploaded
   (Real name: C:\IW3HTP3\examples\ch 20\book.xsd)
docElt: {http://www.w3.org/2001/XMLSchema}schema
Validation was strict, starting with type [Anonymous]
The schema(s) used for schema-validation had no errors
No schema-validity problems were found in the target
```



```
1  <?xml version = "1.0"?>
2
3  <!-- Fig 20.9 : computer.xsd -->
4  <!-- W3C XML Schema document -->
5
6  <schema xmlns = "http://www.w3.org/2001/XMLSchema"
7          xmlns:computer = "http://www.deitel.com/computer"
8          targetNamespace = "http://www.deitel.com/computer">
9
10    <simpleType name = "gigahertz">
11       <restriction base = "decimal">
12          <minInclusive value = "2.1"/>
13       </restriction>
14    </simpleType>
15
16    <complexType name = "CPU">
17       <simpleContent>
18          <extension base = "string">
19             <attribute name = "model" type = "string"/>
20          </extension>
21       </simpleContent>
22    </complexType>
23
```



```
24    <complexType name = "portable">
25       <all>
26          <element name = "processor" type = "computer:CPU"/>
27          <element name = "monitor" type = "int"/>
28          <element name = "CPUSpeed" type = "computer:gigahertz"/>
29          <element name = "RAM" type = "int"/>
30       </all>
31       <attribute name = "manufacturer" type = "string"/>
32    </complexType>
33
34    <element name = "laptop" type = "computer:portable"/>
35
36 </schema>
```



```
1  <?xml version = "1.0"?>
2
3  <!-- Fig 20.10 : laptop.xml             -->
4  <!-- Laptop components marked up as XML -->
5
6  <computer:laptop xmlns:computer = "http://www.deitel.com/computer"
7                   manufacturer = "IBM">
8
9     <processor model = "Centrino">Intel</processor>
10    <monitor>17</monitor>
11    <CPUSpeed>2.4</CPUSpeed>
12    <RAM>256</RAM>
13
14 </computer:laptop>
```

Altri linguaggi di markup (1)

- A partire da XML vari linguaggi di markup sono stati definiti, per specifici domini applicativi
  - W3C XML Schema
  - XSL (Extensible Stylesheet Language)
  - MathML (Mathematical Markup Language)
    - Descrive notazioni ed espressioni matematiche

Altri linguaggi di markup (2)

- CML (Chemical Markup Language)
  - Vocabolario XML per la rappresentazione di informazioni chimiche e molecolari
- MusicML
  - Distribuzione di musica
- …

Altri linguaggi di markup (3)

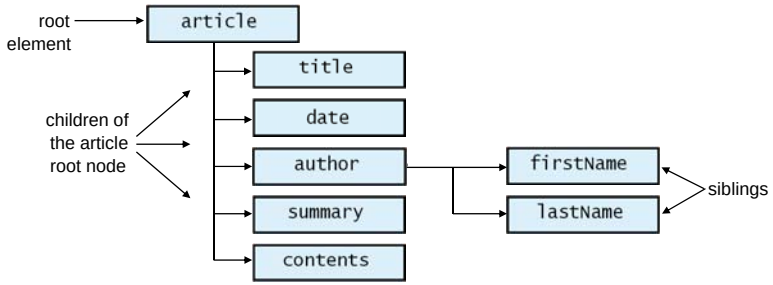
Markup language	Description
VoiceXML	The VoiceXML Forum founded by AT&T, IBM, Lucent and Motorola developed VoiceXML. It provides interactive voice communication between humans and computers through a telephone, PDA (personal digital assistant) or desktop computer. IBM's VoiceXML SDK can process VoiceXML documents. Visit `www.voicexml.org` for more information on VoiceXML. We introduce VoiceXML in Chapter 29, Accessibility.
Synchronous Multimedia Integration Language (SMIL)	SMIL is an XML vocabulary for multimedia presentations. The W3C was the primary developer of SMIL, with contributions from some companies. Visit `www.w3.org/AudioVideo` for more on SMIL. We introduce SMIL in Chapter 28, Multimedia.
Research Information Exchange Markup Language (RIXML)	RIXML, developed by a consortium of brokerage firms, marks up investment data. Visit `www.rixml.org` for more information on RIXML.
ComicsML	A language developed by Jason MacIntosh for marking up comics. Visit `www.jmac.org/projects/comics_ml` for more information on ComicsML.
Geography Markup Language (GML)	OpenGIS developed the Geography Markup Language to describe geographic information. Visit `www.opengis.org` for more information on GML.
Extensible User Interface Language (XUL)	The Mozilla Project created the Extensible User Interface Language for describing graphical user interfaces in a platform-independent way.

Fig. 20.17 Various markup languages derived from XML.

Document Object Model – DOM (1)

- È lo strumento usato per recuperare dati da un file xml
- DOM è un insieme di metodi per gestire un albero
  - Nodi
  - Nodo padre
  - Nodo figlio
  - Un unico nodo radice
    - Contiene tutti gli altri nodi del documento

Document Object Model – DOM (2)



Metodi DOM (1)

- `nodeName`
  - Nome di un elemento, attributo, etc
- `NodeList`
  - Elenco di nodi
  - A cui si può accedere come fosse un array usando il method `item`
- Proprietà `length`
  - Restituisce il numero di children di un elemento root

Metodi DOM (2)

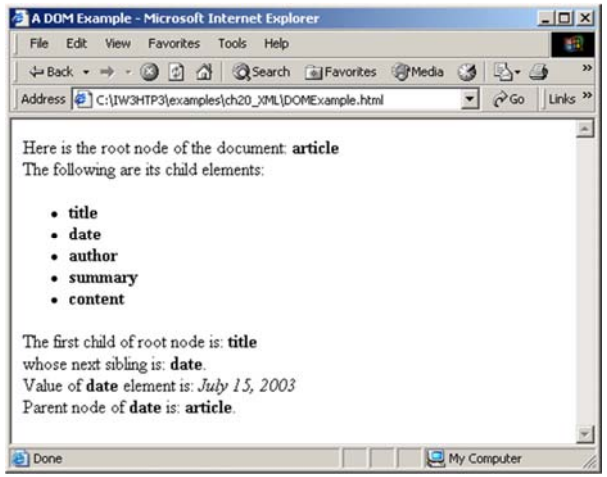
- `nextSibling`
  - Restituisce il sibling successivo del nodo
- `nodeValue`
  - Recupera il valore di un nodo testuale
- `parentNode`
  - Restituisce il nodo parent di un nodo


```
1  <?xml version="1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml">
5
6  <!-- Fig. 20.19: DOMExample.html -->
7  <!-- DOM with JavaScript         -->
8
9     <head>
10       <title>A DOM Example</title>
11    </head>
12
13    <body>
14
15    <script type = "text/javascript" language = "JavaScript">
16       <!--
17       var xmlDocument = new ActiveXObject( "Microsoft.XMLDOM" );
18
19       xmlDocument.load( "article.xml" );
20
21       // get the root element
22       var element = xmlDocument.documentElement;
23
24       document.writeln(
25          "<p>Here is the root node of the document: " +
```

```
26          "<strong>" + element.nodeName + "</strong>" +
27          "<br />The following are its child elements:" +
28          "</p><ul>" );
29
30       // traverse all child nodes of root element
31       for ( var i = 0; i < element.childNodes.length; i++ ) {
32          var curNode = element.childNodes.item( i );
33
34          // print node name of each child element
35          document.writeln( "<li><strong>" + curNode.nodeName
36             + "</strong></li>" );
37       }
38
39       document.writeln( "</ul>" );
40
41       // get the first child node of root element
42       var currentNode = element.firstChild;
43
44       document.writeln( "<p>The first child of root node is: " +
45          "<strong>" + currentNode.nodeName + "</strong>" +
46          "<br />whose next sibling is:" );
47
48       // get the next sibling of first child
49       var nextSib = currentNode.nextSibling;
50
```



```
51       document.writeln( "<strong>" + nextSib.nodeName +
52          "</strong>.<br />Value of <strong>" +
53          nextSib.nodeName + "</strong> element is: " );
54
55       var value = nextSib.firstChild;
56
57       // print the text value of the sibling
58       document.writeln( "<em>" + value.nodeValue + "</em>" +
59          "<br />Parent node of <strong>" + nextSib.nodeName +
60          "</strong> is: <strong>" +
61          nextSib.parentNode.nodeName + "</strong>.</p>" );
62          -->
63    </script>
64
65    </body>
66 </html>
```

Altri Metodi DOM (1)

Method	Description
`getNodeType`	Returns an integer representing the node type.
`getNodeName`	Returns the name of the node. If the node does not have a name, a string consisting of # followed by the type of the node is returned.
`getNodeValue`	Returns a string or null depending on the node type.
`getParentNode`	Returns the parent node.
`getChildNodes`	Returns a `NodeList` (Fig. 20.21) with all the children of the node.
`getFirstChild`	Returns the first child in the `NodeList`.
`getLastChild`	Returns the last child in the `NodeList`.
`getPreviousSibling`	Returns the node preceding this node, or null.
`getNextSibling`	Returns the node following this node, or null.
`getAttributes`	Returns a `NamedNodeMap` (Fig. 20.22) containing the attributes for this node.
`insertBefore`	Inserts the node (passed as the first argument) before the existing node (passed as the second argument). If the new node is already in the tree, it is removed before insertion. The same behavior is true for other methods that add nodes.

Altri Metodi DOM (2)

`replaceChild`	Replaces the second argument node with the first argument node.
`removeChild`	Removes the child node passed to it.
`appendChild`	Appends the node passed to it to the list of child nodes.
`getElementsByTagName`	Returns a `NodeList` of all the nodes in the subtree with the name specified as the first argument ordered as they would be encountered in a preorder traversal. An optional second argument specifies either the direct child nodes (0) or any descendant (1).
`getChildAtIndex`	Returns the child node at the specified index in the child list.
`addText`	Appends the string passed to it to the last `Node` if it is a `Text` node, otherwise creates a new `Text` node for the string and adds it to the end of the child list.
`isAncestor`	Returns `true` if the node passed is a parent of the node or is the node itself.

Fig. 20.20 Some DOM `Node` object methods.

Altri Metodi DOM (3)

Method	Description
`item`	Passed an index number, returns the element node at that index. Indices range from 0 to *length* – 1.
`getLength`	Returns the total number of nodes in the list.

Fig. 20.21 Some DOM `NodeList` methods.

Method	Description
`getNamedItem`	Returns either a node in the `NamedNodeMap` with the specified name or null.
`setNamedItem`	Stores a node passed to it in the `NamedNodeMap`. Two nodes with the same name cannot be stored in the same `NamedNodeMap`.
`removeNamedItem`	Removes a specified node from the `NamedNodeMap`.
`getLength`	Returns the total number of nodes in the `NamedNodeMap`.
`getValues`	Returns a `NodeList` containing all the nodes in the `NamedNodeMap`.

Fig. 20.22 Some DOM `NamedNodeMap` methods.

Altri Metodi DOM (4)

Method	Description
`getDocumentElement`	Returns the root node of the document.
`createElement`	Creates and returns an element node with the specified tag name.
`createAttribute`	Creates and returns an attribute node with the specified name and value.
`createTextNode`	Creates and returns a text node that contains the specified text.
`createComment`	Creates a comment to hold the specified text.

Fig. 20.23 Some DOM `Document` methods.

Altri Metodi DOM (5)

Method	Description
`getTagName`	Returns the name of the element.
`setTagName`	Changes the name of the element to the specified name.
`getAttribute`	Returns the value of the specified attribute.
`setAttribute`	Changes the value of the attribute passed as the first argument to the value passed as the second argument.
`removeAttribute`	Removes the specified attribute.
`getAttributeNode`	Returns the specified attribute node.
`setAttributeNode`	Adds a new attribute node with the specified name.

Fig. 20.24 Some DOM `Element` methods.

Method	Description
`getValue`	Returns the specified attribute's value.
`setValue`	Changes the value of the attribute to the specified value.
`getName`	Returns the name of the attribute.

Fig. 20.25 Some DOM `Attr` methods.

Method	Description
`getData`	Returns the data contained in the node (text or comment).
`setData`	Sets the node's data.
`getLength`	Returns the number of characters contained in the node.

Fig. 20.26 Some DOM `Text` and `Comment` methods.

Simple API for XML - SAX (1)

- Sviluppato da parte dei membri della mailing list XML-DEV
- Permette di svolgere il parsing di documenti XML utilizzando un modello basato su eventi
- Fornisce varie API per accedere alle informazioni del documento XML

Simple API for XML - SAX (2)

- È basato sull'invocazione dei metodi di listener
- Trasmette i dati del documento XML alle applicazioni che ne fanno richiesta
- Rispetto ai parser basati su DOM i parser SAX offrono migliori prestazioni e minor overhead di memoria

Extensible Stylesheet Language - XSL (1)

- È un linguaggio dichiarativo
  - creato dal W3C
  - che permette di specificare come presentare un documento XML
  - che permette di specificare come trasformare un doc XML
    - in un altro documento XML
    - HTML, XHTML, text, o altro linguaggio interpretabile da un dispositivo di presentazione

Extensible Stylesheet Language - XSL (2)

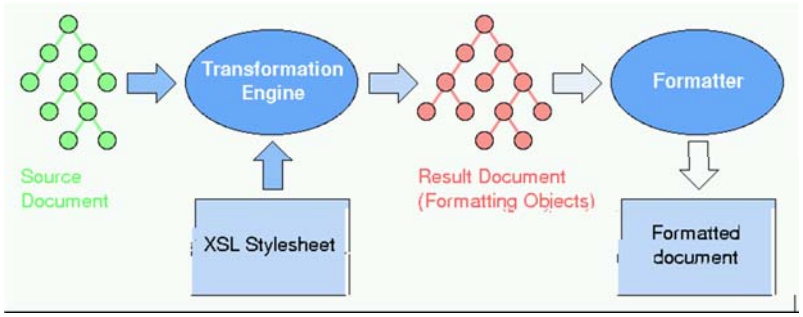
- Può essere utilizzato sia sul lato client che sul lato server
- Comprende tre tecnologie:
  - XSL-FO (XSL Formatted Objects)
    - Vocabolario per specificare la formattazione
  - XSLT (XSL Transformation)
    - Alberi sorgente e risultato dell'elaborazione
  - Xpath
    - Localizza le parti del documento con albero sorgente che corrispondono ai modelli definiti nel foglio di stile XSL

Extensible Stylesheet Language - XSL (3)

Element	Description
`<xsl:apply-templates>`	Applies the templates of the XSL document to the children of the current node.
`<xsl:apply-templates match = "`*expression*`">`	Applies the templates of the XSL document to the children of *expression*. The value of the attribute `match` (i.e., *expression*) must be some XPath expression that specifies elements.
`<xsl:template>`	Contains rules to apply when a specified node is matched.
`<xsl:value-of select = "`*expression*`">`	Selects the value of an XML element and adds it to the output tree of the transformation. The required `select` attribute contains an XPath expression.
`<xsl:for-each select = "`*expression*`">`	Implicitly contains a template that is applied to every node selected by the XPath specified by the `select` attribute.
`<xsl:sort select = "`*expression*`">`	Used as a child element of an `<xsl:apply-templates>` or `<xsl:for-each>` element. Sorts the nodes selected by the `<apply-template>` or `<for-each>` element so that the nodes are processed in sorted order.
`<xsl:output>`	Has various attributes to define the format (e.g., xml, html), version (e.g., 1.2, 2.0), document type and media type of the output document. This tag is a top-level element, which means that it can be used only as a child element of a `stylesheet`.
`<xsl:copy>`	Adds the current node to the output tree.

Fig. 20.27 Commonly used XSL stylesheet elements.

Architettura XSL



XSLT (1)

- Una trasformazione XSL
  - accetta per input albero sorgente, rappresentato da un documento XML
  - produce in output un albero risultato, rappresentato generalmente da un altro documento XML (può anche essere (X)HTML o testo)
- Un file XSL è composto da una lista di regole di trasformazione (template rules)
  - una template rule specifica come ogni nodo dell'albero sorgente deve apparire nell'albero risuLtato
  - Una template rule: pattern + template
  - il pattern (espresso in Xpath) specifica il nodo (radice) dell'albero sorgente a cui applicare la regola di trasformazione
  - il template specifica il nodo (radice) dell'albero risultato da instanziare

XSLT (2)

- XSL Processor
  - Il processore XSLT scandisce l'albero sorgente, partendo dal nodo rAdice per cercare un matching template rule nel file XLS
  - quando un pattern è individuato nell'albero sorgente, il template corrispondente è generato nell'albero risultato
  - l'istruzione <xsl:apply-templates/> permette di applicare le regole di trasformazione in modo ricorsivo ai nodi figli

```
1  <?xml version = "1.0"?>
2  <?xml:stylesheet type = "text/xsl" href = "games.xsl"?>
3
4  <!-- Fig. 20.28 : games.xml -->
5  <!-- Sports Database        -->
6
7  <sports>
8
9     <game id = "783">
10       <name>Cricket</name>
11
12       <paragraph>
13          Popular in Commonwealth nations.
14       </paragraph>
15    </game>
16
17    <game id = "239">
18       <name>Baseball</name>
19
20       <paragraph>
21          Popular in America.
22       </paragraph>
23    </game>
24
```

I simboli <? e ?> racchiudono le processing instructions


```
25    <game id = "418">
26       <name>Soccer (Football)</name>
27
28       <paragraph>
29          Popular sport in the world.
30       </paragraph>
31    </game>
32
33 </sports>
```


```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.29 : games.xsl      -->
4  <!-- A simple XSLT transformation   -->
5
6  <!-- reference XSL stylesheet URI   -->
7  <xsl:stylesheet version = "1.0"
8     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
9
10    <xsl:output method = "html" omit-xml-declaration = "no"
11       doctype-system =
12          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
13       doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>
14
15 <xsl:template match = "/">
16
17   <html xmlns="http://www.w3.org/1999/xhtml">
18
19     <head>
20        <title>Sports</title>
21     </head>
22
23     <body>
24
25        <table border = "1" bgcolor = "cyan">
```



```
26
27        <thead>
28
29          <tr>
30             <th>ID</th>
31             <th>Sport</th>
32             <th>Information</th>
33          </tr>
34
35        </thead>
36
37        <!-- insert each name and paragraph element value  -->
38        <!-- into a table row.                              -->
39        <xsl:for-each select = "/sports/game">
40
41           <tr>
42              <td><xsl:value-of select = "@id"/></td>
43              <td><xsl:value-of select = "name"/></td>
44              <td><xsl:value-of select = "paragraph"/></td>
45           </tr>
46
47        </xsl:for-each>
48
49        </table>
50
```

```
51      </body>
52
53   </html>
54
55 </xsl:template>
56
57 </xsl:stylesheet>
```



```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.30 : sorting.xml         -->
4  <!-- Usage of elements and attributes -->
5
6  <?xml:stylesheet type = "text/xsl" href = "sorting.xsl"?>
7
8  <book isbn = "999-99999-9-X">
9     <title>Deitel&amp;apos;s XML Primer</title>
10
11    <author>
12       <firstName>Paul</firstName>
13       <lastName>Deitel</lastName>
14    </author>
15
16    <chapters>
17       <frontMatter>
18          <preface pages = "2"/>
19          <contents pages = "5"/>
20          <illustrations pages = "4"/>
21       </frontMatter>
22
23       <chapter number = "3" pages = "44">
24          Advanced XML</chapter>
```



```
25       <chapter number = "2" pages = "35">
26          Intermediate XML</chapter>
27       <appendix number = "B" pages = "26">
28          Parsers and Tools</appendix>
29       <appendix number = "A" pages = "7">
30          Entities</appendix>
31       <chapter number = "1" pages = "28">
32          XML Fundamentals</chapter>
33    </chapters>
34
35    <media type = "CD"/>
36 </book>
```



```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 20.31 : sorting.xsl                      -->
4  <!-- Transformation of Book information into XHTML -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9     <xsl:output method = "html" omit-xml-declaration = "no"
10
11       doctype-system =
12          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
13       doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>
14
15    <xsl:template match = "/">
16       <html xmlns = "http://www.w3.org/1999/xhtml">
17          <xsl:apply-templates/>
18       </html>
19    </xsl:template>
20
21    <xsl:template match = "book">
22       <head>
23          <title>ISBN <xsl:value-of select = "@isbn"/> -
24             <xsl:value-of select = "title"/></title>
25       </head>
```

```
26
27      <body>
28         <h1><xsl:value-of select = "title"/></h1>
29
30         <h2>by <xsl:value-of select = "author/lastName"/>,
31            <xsl:value-of select = "author/firstName"/></h2>
32
33         <table border = "1">
34            <xsl:for-each select = "chapters/frontMatter/*">
35               <tr>
36                  <td align = "right">
37                     <xsl:value-of select = "name()"/>
38                  </td>
39
40                  <td>
41                     ( <xsl:value-of select = "@pages"/> pages )
42                  </td>
43               </tr>
44            </xsl:for-each>
45
46            <xsl:for-each select = "chapters/chapter">
47               <xsl:sort select = "@number" data-type = "number"
48                  order = "ascending"/>
49               <tr>
50                  <td align = "right">
```



```
51                     Chapter <xsl:value-of select = "@number"/>
52                  </td>
53
54                  <td>
55                     <xsl:value-of select = "text()"/>
56                     ( <xsl:value-of select = "@pages"/> pages )
57                  </td>
58               </tr>
59            </xsl:for-each>
60            <xsl:for-each select = "chapters/appendix">
61               <xsl:sort select = "@number" data-type = "text"
62                  order = "ascending"/>
63               <tr>
64                  <td align = "right">
65                     Appendix <xsl:value-of select = "@number"/>
66                  </td>
67
68                  <td>
69                     <xsl:value-of select = "text()"/>
70                     ( <xsl:value-of select = "@pages"/> pages )
71                  </td>
72               </tr>
73            </xsl:for-each>
74         </table>
75
```



```
76         <br />Pages:
77            <xsl:variable name = "pagecount"
78               select = "sum(chapters//*/@pages)"/>
79            <xsl:value-of select = "$pagecount"/>
80         <br />Media Type: <xsl:value-of select = "media/@type"/>
81      </body>
82   </xsl:template>
83
84 </xsl:stylesheet>
```

Web Resources (1)

- www.w3.org/xml
- www.xml.org
- www.w3.org/style/XSL
- www.w3.org/TR
- xml.apache.org
- www.xmlbooks.com
- www.xmlsoftware.com
- www.xml-zone.com
- wdvl.internet.com/Authoring/Languages/XML

Web Resources (2)

- www.xml.com
- msdn.microsoft.com/xml/default.asp
- www.oasis-opEn.org/cover/xml.html
- www.gca.org/whats_xml/default.htm
- www.xmlinfo.com
- www.ibm.com/developer/xml
- developer.netscape.com/tech/xml/index.html

Web Resources (3)

- www.projectcool.com/developer/xmlz
- www.ucc.ie/xml
- www.xml-cml.org
- backend.userland.com/rss
- www.w3.org/2002/ws
- www.oasis-open.org
- www.clearmethods.com
- www.textuality.com/xml
- www.zvon.org