Programmazione Web - XML

3.0 - eXtensible Mark-up Language

XML (eXtensible Mark-up Language) è un linguaggio a marcatori estendibile che fornisce una struttura sintattica di base comune a tutti i linguaggi a marcatori. A differenza di HTML, XML non fornisce un insieme predefinito di marcatori ma offre gli strumenti linguistici per definire marcatori personalizzati e regole di correttezza sintattica, permettendo così di creare classi di documenti XML specifiche.

Vantaggi di XML

- È un formato aperto
- Non è legato a nessuna piattaforma hardware/software
- È ideale per l'interscambio di documenti tra applicazioni diverse

Svantaggi di XML

- Eccessiva "verbosità" del linguaggio
- Un documento XML richiede molti più byte rispetto a un formato piatto senza marcatori
- La rappresentazione in memoria centrale richiede una struttura ad albero, la cui costruzione e navigazione non sono banali e richiedono maggiori risorse di calcolo

Esempio di documento XML

```
<?xml version="1.0"?>
<!DOCTYPE PAPER SYSTEM "paper.dtd">
<PAPER name="sample doc">
  <TITLE>Using bibliography citations.</TITLE>
  <SECTION title="Introduction">
    This paper shows how to use a hypothetical
    <EMPH>XML language</EMPH> to describe structured
    documents which exploit the concept of citation
  </SECTION>
  <SECTION title="A Citation">
    Here we have an example of citation. We refer to
    Knuth's paper which introduced the concept of attribute
    grammar. This paper has the number <CITE label="Knuth68"/>
    in our bibliography.
  </SECTION>
  <BIBLIOGRAPHY>
    <BIBITEM label="ASU85">
      A.V. Aho, R. Sethi, J. D. Ullman, "Compilers: Principles,
      Techniques, Tools", Addison-Wesley, 1985.
    </BIBITEM>
    <BIBITEM label="Knuth68">
      D. E. Knuth, "Semantics of Context Free Languages",
      Mathematical System Theory, Vol. 2, pp. 127-145, 1968.
    </BIBITEM>
  </BIBLIOGRAPHY>
</PAPER>
```

3.1 - Elementi del Linguaggio XML

Marcatori

I marcatori in XML sono racchiusi tra i simboli "<" e ">" e delimitano gli elementi XML. Esistono due tipi di marcatori:

- **Marcatore di apertura**: `<Nome>` o `<Nome *attributi*>`
- **Marcatore di chiusura**: `</Nome>`

Gli elementi XML seguono una struttura a parentesi: ad ogni marcatore di apertura corrisponde un marcatore di chiusura con lo stesso nome.

Elemento (Element)

Una coppia di marcatori di apertura e di chiusura descrive un Elemento. Un elemento è un concetto semistrutturato che ha un contenuto e delle proprietà (attributi).

Esempio di elemento SECTION:

```
<SECTION title="A Citation">
  Here we have an example of citation. We refer to
  Knuth's paper which introduced the concept of attribute
  grammar. This paper has the number <CITE label="Knuth68"/>
  in our bibliography.
</SECTION>
```

Elemento Vuoto (Empty Element)

Un elemento senza contenuto può essere scritto nella forma compatta: `<Nome/>` o `<Nome attributi/>` .

Esempio:

```
<CITE label="Knuth68"/>
```

Attributo

Un attributo è una proprietà del concetto descritto dall'elemento. La struttura di un attributo è: `Nome = Valore` , dove il valore è una stringa racchiusa tra virgolette ("...") o apici ('...'). Nell'XML base non sono previsti tipi per gli attributi.

Esempio:

```
<SECTION title="A Citation">
<CITE label='Knuth68'/>
```

Contenuto di un elemento

Il contenuto di un elemento può essere un misto di testo e occorrenze di altri elementi (contenuto semi-strutturato). Nell'XML base non vi sono particolari vincoli sul contenuto dei singoli elementi. Questi vincoli possono essere specificati nel DTD (Document Type Definition).

Marcatore di preambolo

Tutti i documenti XML iniziano con il marcatore di preambolo:

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

È possibile specificare l'encoding del testo:

```
<?xml version="1.0" encoding="ASCII"?>
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="iso-8859-1"?>
```

Il rispetto delle regole viste finora rappresenta un primo livello di correttezza dei documenti XML. Un documento che rispetta queste regole viene definito **Ben Formato**.

3.2 - DTD (Document Type Definition)

Il DTD (Document Type Definition) definisce la struttura dei documenti XML, specificando:

- Gli elementi ammessi
- La struttura del contenuto degli elementi
- Gli attributi degli elementi

Meta-Tags per la definizione di elementi

```
<!ELEMENT Nome StrutturaCont.>
```

- `Nome` : il nome dell'elemento che si definisce
- `StrutturaCont.` : la struttura del contenuto dell'elemento, espressa come un'espressione regolare che specifica la sequenza delle occorrenze degli elementi nel contenuto

Operatori per le espressioni regolari

- `( ... )+` : ripetizione non vuota
- `( ... )*` : ripetizione anche vuota
- `( ... )?` : opzionalità
- `E1,E2` : sequenza di elementi
- `( E1 | E2 | ... )` : alternativa

Tipi di contenuto

- Contenuto Testuale: `(#PCDATA)` (solo testo) o `(#PCDATA | E1 | E2 | ...)*` (contenuto misto)
- Contenuto vuoto: `EMPTY`

Meta-Tags per la definizione di attributi

```
<!ATTLIST Elemento Nome Tipo Obblig.>
```

- `Elemento` : l'elemento per il quale si definiscono gli attributi
- `Nome` : il nome dell'attributo
- `Tipo` : tipologia dell'attributo (non il tipo di dato)
- `Obblig.` : opzione di obbligatorietà

Tipi di attributi

- `CDATA` : stringa generica
- `ID` : identifica l'elemento
- `IDREF` : riferimento all'ID di un altro elemento

Opzioni di obbligatorietà

- `#REQUIRED` : attributo obbligatorio
- `#IMPLIED` : attributo facoltativo
- `valore_default` : il valore da assumere quando non viene specificato

Esempio di DTD per PAPER

```
<!ELEMENT BIBITEM (#PCDATA)>
<!ATTLIST BIBITEM label ID #IMPLIED>
<!ELEMENT BIBLIOGRAPHY (BIBITEM)+ >
<!ELEMENT CITE EMPTY >
<!ATTLIST CITE label IDREF #REQUIRED>
<!ELEMENT EMPH (#PCDATA) >
<!ELEMENT SECTION (#PCDATA | CITE | EMPH)*>
<!ATTLIST SECTION title CDATA #REQUIRED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT PAPER (TITLE,(SECTION)*,(BIBLIOGRAPHY)?)>
<!ATTLIST PAPER name ID #REQUIRED>
```

Specificazione del DTD nel documento XML

Nel documento XML si può specificare il DTD che definisce il tipo del documento. Dopo il marcatore di preambolo:

```
<!DOCTYPE PAPER SYSTEM "paper.dtd">
```

Il file "paper.dtd" contiene il DTD.

Documento XML Valido

Il rispetto delle regole definite nel DTD porta ad un più alto livello di correttezza dei documenti. Un documento corretto rispetto al DTD viene detto **Valido**.

3.3 - Parsing di XML

Processore XML

Secondo la specifica, un "Processore XML" è un programma che ha il compito di processare un documento XML per utilizzare le informazioni riportate nel documento al fine di elaborarle e svolgere attività specifiche.

Parser XML

Un parser è uno strumento software che effettua l'analisi sintattica di testi basati su un linguaggio artificiale. XML è un linguaggio artificiale, quindi per processare un documento XML occorre un "XML Parser". Siccome il linguaggio ha una struttura sintattica comune, indipendente dalla particolare classe di documenti, esistono parser standard disponibili per i vari linguaggi di programmazione.

Tipologie di Parsing

Esistono due principali modalità di parsing XML:

1. **Parsing SAX (Simple API for XML)**
    - Modalità a "Eventi"
    - Quando viene attivato, al parser si deve fornire un oggetto "ContentHandler" (interfaccia)
    - L'handler fornisce un metodo specifico per ogni elemento sintattico del linguaggio
    - Per esempio, quando il parser incontra un marcatore di apertura, chiama un metodo specifico; quando trova un marcatore di chiusura, chiama un altro metodo

Esempio di handler in Java:

```
public void startElement(
    String namespaceURI,
    String localName,
    String qName,
    Attributes atts) { ... }
```

2. **Parsing DOM (Document Object Model)**
   - Il parser costruisce una rappresentazione del documento in memoria centrale
   - Esiste uno standard del W3C chiamato DOM
   - La rappresentazione è una struttura ad albero dove i nodi descrivono gli elementi, i blocchi di testo e gli attributi degli elementi

Tipologie di Parser

Esistono tre tipologie di parser XML:

1. **Parser Non validante**: verifica solo che il documento sia ben formato
2. **Parser Validante**: verifica la correttezza del documento rispetto al DTD
3. **Parser Validante rispetto a XML Schema**: valida il documento rispetto alla specifica XML Schema (un'alternativa più moderna e flessibile al DTD)

XML e HTML

A differenza di XML, HTML è un linguaggio a marcatori con un insieme ben definito di marcatori (elementi). Entrambi sono linguaggi a marcatori, ma qual è la relazione tra di essi?

Storia: SGML

Il capostipite di XML e HTML è SGML (Structured Generalized Mark-up Language), nato negli anni '80 nella comunità degli editori americani per descrivere libri e articoli. SGML era decisamente visionario e prevedeva costrutti che non sono mai stati implementati perché troppo complicati per le risorse computazionali dell'epoca. Tim Berners-Lee (l'inventore del World Wide Web) aveva lavorato al progetto SGML.

Nascita di HTML

Conoscendo le potenzialità di un linguaggio a marcatori, Tim Berners-Lee decise di adottare questo approccio per definire HTML, partendo dalla sua esperienza con SGML. Era il 1994, quando nacque il

World Wide Web.

Nascita di XML

Nel 1996, dopo il successo di HTML (che tuttavia può solo descrivere le pagine web), nacque l'idea di un formato indipendente dal contesto applicativo, aperto e indipendente dalla piattaforma hardware/software: nasceva XML. L'obiettivo era che XML diventasse "il formato" dei dati sul web. XML deriva direttamente da SGML, ma ripulito dai costrutti impossibili, rendendo il parsing fattibile.

Tuttavia, alcune scelte sintattiche di HTML non lo rendevano compatibile con XML. Ad esempio:

- `<br>` in HTML (prima della versione 5)
- `<br/>` in XML

Attualmente, HTML 5 è compatibile con la sintassi di XML, anche se i browser sono tolleranti a scritture non propriamente "corrette". Il passo intermedio è stato XHTML, che poi è confluito in HTML 5.

3.4 - JavaScript e XML

In JavaScript, è possibile elaborare documenti XML, passando sempre attraverso la rappresentazione DOM (Document Object Model). Lo standard W3C prevede una gerarchia specifica per i nodi, che viene mantenuta anche in JavaScript, sebbene il concetto di ereditarietà non esista in senso stretto.

Gerarchia dei Nodi

Node (Nodo)

Ogni elemento di un documento XML viene rappresentato da un oggetto `Node`. Le relazioni di annidamento degli elementi nel documento XML vengono rappresentate da una relazione padre-figlio tra nodi, inducendo una struttura ad albero.

Campi di Node

1. Attributi di sola lettura:
    - `attributes` : NamedNodeMap degli attributi
    - `childNodes` : Lista dei nodi figli
    - `firstChild` : Primo nodo figlio
    - `lastChild` : Ultimo nodo figlio
2. Ulteriori campi:
    - `nextSibling` : Nodo successivo

- `nodeName` : Nome del nodo
- `nodeType` : Tipo numerico del nodo
- `nodeValue` : Valore del nodo

3. Campi aggiuntivi:
    - `ownerDocument` : Documento proprietario
    - `parentElement` : Elemento padre
    - `parentNode` : Nodo padre
    - `previousSibling` : Nodo precedente

Metodi di Node

- `appendChild(newChild)` : Aggiunge un nuovo nodo figlio
- `cloneNode(deep)` : Clona il nodo
- `hasChildNodes()` : Verifica la presenza di nodi figli
- `insertBefore(newChild, refChild)` : Inserisce un nodo prima di un altro
- `removeChild(oldChild)` : Rimuove un nodo figlio
- `replaceChild(newChild, refChild)` : Sostituisce un nodo figlio

Node RecordType

Il campo `nodeType` indica il tipo di nodo. Ecco le costanti più comuni:

- 1: `ELEMENT_NODE`
- 2: `ATTRIBUTE_NODE`
- 3: `TEXT_NODE`
- 4: `CDATA_SECTION_NODE`
- 8: `COMMENT_NODE`
- 9: `DOCUMENT_NODE`
- 10: `DOCUMENT_TYPE_NODE`
- 11: `DOCUMENT_FRAGMENT_NODE`

Element (sottoclasse di Node)

Estende `Node` con caratteristiche specifiche per descrivere gli elementi.

Campi di Element

- `tagName` : Nome del tag (coincide con `nodeName`)

Metodi di Element

- `getAttribute(name)` : Ottiene il valore di un attributo
- `removeAttribute(name)` : Rimuove un attributo

- `setAttribute(name, value)` : Imposta un attributo
- `getElementsByTagName(name)` : Restituisce la lista di nodi con un determinato tag nel sottoalbero

CharacterData

Rappresenta il contenuto testuale dei documenti XML. È una sottoclasse di `Node` che gestisce qualsiasi pezzo di testo tra due marcatori, compresi gli a capo.

Campi di CharacterData

- `data` : Contenuto testuale
- `length` : Lunghezza del testo

Metodi di CharacterData

- `appendData(arg)` : Aggiunge testo
- `deleteData(offset, count)` : Elimina parte del testo
- `insertData(offset, arg)` : Inserisce testo
- `replaceData(offset, count, arg)` : Sostituisce parte del testo
- `substringData(offset, count)` : Estrae una sottostringa

Text

Sottoclasse di `CharacterData` che rappresenta i nodi testuali. Aggiunge il metodo `splitText(offset)` per dividere un nodo di testo.

Document

Rappresenta l'intero documento XML.

Campi di Document

- `documentElement` : Elemento radice del documento

Metodi di Document

- `createElement(tagName)` : Crea un nuovo elemento
- `createTextNode(data)` : Crea un nuovo nodo di testo
- `getElementById(elementId)` : Trova un elemento per ID
- `getElementsByTagName(tagname)` : Trova elementi per nome tag

NodeList e NamedNodeMap

Classi ausiliarie per gestire collezioni di nodi e attributi.

NodeList

- `length` : Numero di elementi
- `item(index)` : Restituisce il nodo all'indice specificato

NamedNodeMap

- `length` : Numero di attributi
- `getNamedItem(name)` : Ottiene un attributo per nome
- `item(index)` : Restituisce l'attributo all'indice specificato
- `removeNamedItem(name)` : Rimuove un attributo
- `setNamedItem(arg)` : Imposta un attributo

Creazione e Parsing di Documenti XML in JavaScript

Serializzazione di Documenti XML

Per inviare un documento XML, è necessario serializzarlo, ovvero generare il testo corrispondente. Gli interpreti JavaScript forniscono un oggetto `XMLSerializer` :

```
ser = new XMLSerializer();
doc = ser.serializeToString(xml_doc);
// 'doc' contiene la stringa XML generata
```

Parsing di Documenti XML

Per convertire una stringa XML in un oggetto DOM, si usa `DOMParser` :

```
var parser = new DOMParser();
var xmlobject = parser.parseFromString(xmlstring, "text/xml");
```

Esempio di Scansione DOM

Dato un documento XML:

```xml
<?xml version="1.0"?>
<root>
  <name>A</name>
  <name>B</name>
</root>
```

Funzione di scansione in JavaScript:

```javascript
function scandoc(doc) {
  var nl = doc.getElementsByTagName("name");
  for(var i = 0; i < nl.length; i++) {
    var n2 = nl.item(i).firstChild; // recupero il testo contentuo nel campo
    myfunction(n2.data); // passo il testo ad una funzione
  }
}
```

Questa funzione recupera tutti gli elementi `<name>` , accede al loro primo nodo figlio (il testo) e passa il valore a una funzione `myfunction()` .