



Università degli Studi di Bari



*Laboratorio di Acquisizione  
della Conoscenza e  
Apprendimento delle Macchine*

---

# **Parsing di documenti XML**

**Prof. Giovanni Semeraro**

**Dr. Oriana Licchelli**

**{semeraro, licchelli}@di.uniba.it**

**Corso di Gestione della Conoscenza d'Impresa**

**Anno Accademico 2005-2006**



# Parser XML

---

**Parser XML:** modulo software che si colloca tra l'applicazione e il documento XML. Permette all'applicazione di accedere al contenuto e alla struttura del documento XML.

Esistono due tipi di parser: validanti e non validanti.

**Parser Validanti:** controllano se un documento è ben-formato, cioè che ogni elemento sia racchiuso tra due tag (uno di apertura e uno di chiusura), e se è un documento XML valido, cioè se è fedele alle regole definite nel suo DTD. Un parser validante robusto, affidabile e gratuito è *Xerces*, scaricabile da <http://xml.apache.org/dist/xerces-j/>.

**Parser non validanti:** controllano se un documento è ben formato.



# DOM e SAX

---

Ci sono due modi per interfacciare un parser con un'applicazione: usando un'interfaccia *object-based* oppure un'interfaccia *event-based*.

**Approccio object-based:** il parser costruisce esplicitamente in memoria un albero che contiene tutti gli elementi del documento XML.

**Approccio event-based:** il parser genera un evento ogni qual volta incontra, durante la lettura del documento XML, un elemento, un attributo, o del testo. Esistono eventi per i tag di apertura e di chiusura, per gli attributi, per il testo, per le entità ecc...

Xerces implementa entrambi i parser mediante le classi *DOMParser* e *SAXParser* nel package *org.apache.xerces.parsers*.



# DOM e SAX: esempio...

Impiegati.xml:

```
<?xml version="1.0"?> <!DOCTYPE impiegati SYSTEM impiegati.dtd>
<impiegati>
  <impiegato id="M.R">
    <nome>Mario Rossi</nome>
    <email>mrossi@foo.com</email>
  </impiegato>
  <impiegato id="F.B">
    <nome>Filippo Bianchi</nome>
    <email>fbaldi@foo.com</email>
  </impiegato>
  <impiegato id="A.V">
    <nome>Alice Verdi</nome>
    <email>averdi@foo.com</email>
    <url href="http://www.foo.com/~averdi/" />
  </impiegato>
</impiegati>
```



## ... DOM e SAX: esempio...

Il relativo DTD:

```
<!ELEMENT impiegati (impiegato)*>  
  <!ELEMENT impiegato (nome, email, url?)>  
  <!ATTLIST impiegato id CDATA #REQUIRED>  
    <!ELEMENT nome (#PCDATA)>  
    <!ELEMENT email (#PCDATA)>  
    <!ELEMENT url EMPTY>  
    <!ATTLIST url href CDATA #REQUIRED>
```



# Creazione di un Parser DOM

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import java.io.IOException;

...
String xmlFile = "file:///impiegati.xml";
DOMParser parser = new DOMParser();
try {
    parser.parse(xmlFile);
} catch (SAXException se) { // Documento XML non ben formato
    se.printStackTrace();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
Document document = parser.getDocument();

/* istanza di Document - interfaccia che permette la
rappresentazione in memoria dell'intero documento XML */
```



# Creazione di un Parser SAX

```
import org.apache.xerces.parsers.SAXParser;
import org.xml.sax.Parser;
import org.xml.sax.ParserFactory;
import org.xml.sax.SAXException;
import java.io.IOException;

...
String xmlFile = "file:///impiegati.xml";
String parserClass = "org.apache.xerces.parsers.SAXParser";
Parser parser = ParserFactory.makeParser(parserClass);
try {
    parser.parse(xmlFile);
} catch (SAXException se) {
    se.printStackTrace();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```



# DOM o SAX?

---

La scelta tra la classe DOMParser e SAXParser dipende dal tipo di applicazione che lo utilizza.

**DOMParser**, parser *object-based*, è più appropriato in applicazioni dove è richiesta una manipolazione di documenti XML, come ad esempio browser, editor, processori XSL ecc.

**SAXParser**, parser *event-based*, è ideale nei casi in cui l'applicazione non richiede la memorizzazione dei dati in XML ma in un altro formato, come ad esempio un'applicazione per importare documenti XML in un database (il formato dell'applicazione è la struttura del database e non quella del documento XML).



## Errori in un documento XML...

Gli errori, che si possono incontrare durante il parsing di un documento XML, si suddividono in tre categorie:

- **Fatal Errors:** errori non recuperabili; si hanno quando il documento non è ben formato (errori in well-formedness).
- **Errors:** errori recuperabili; si hanno quando il documento non è valido, cioè non rispetta le regole definite nella sua DTD.
- **Warnings:** errori di altro tipo.



# ...Errori in un documento XML...

**Xerces:** opzione di validazione non attiva (errori e warnings ignorati).

Un errore durante il parser genera un'eccezione di tipo SAXException.

Per attivare la proprietà (feature) di validazione si usa:

```
parser.setFeature(http://xml.org/sax/features/validation, true);
```

Per catturare le eccezioni: implementare un **error handler** e registrarlo usando il metodo *setErrorHandler()*. Se si fa tutto in una sola classe si usa: `parser.setErrorHandler(this);` altrimenti si passa al metodo un'istanza della classe che implementa l'error handler.



# Parser DOM validante...

```
import org.apache.xerces.parsers.DOMParser;
import org.xml.sax.*; import org.w3c.dom.*;
import java.io.IOException;

public class ValidatingDOM implements ErrorHandler {
    public ValidatingDOM (String xmlFile) {
        DOMParser parser = new DOMParser();

        try { // Attiviamo la validazione

            parser.setFeature("http://xml.org/sax/features/validation", true);
        } catch (SAXNotRecognizedException e) { System.err.println (e);
        } catch (SAXNotSupportedException e) { System.err.println (e); }

        parser.setErrorHandler (this); // Registriamo un Error Handler

        try { // Facciamo il parsing del documento e attraversiamo il DOM
            parser.parse(xmlFile);
            Document document = parser.getDocument();
            traverse (document);
        } catch (SAXException) { System.err.println (e);
        } catch (IOException e) { System.err.println (e); } }
```



# ... Parser DOM validante...

```
// Warning Event Handler
public void warning (SAXParseException e)
    throws SAXException {
    System.err.println ("Warning:  " + e);
}

// Error Event Handler
public void error (SAXParseException e)
    throws SAXException {
    System.err.println ("Error:  " + e);
}

// Fatal Error Event Handler
public void fatalError (SAXParseException e)
    throws SAXException {
    System.err.println ("Fatal Error:  " + e);
}
```



# ... Parser DOM validante

```
/* metodo ricorsivo per attraversare il DOM Tree che
   stampa i nomi degli elementi */

private void traverse (Node node) {
    int type = node.getNodeType();
    if (type == Node.ELEMENT_NODE)
        System.out.println (node.getNodeName());
    NodeList children = node.getChildNodes();
    if (children != null) {
        for (int i=0; i < children.getLength(); i++)
            traverse (children.item(i));
    }
}

// Main: prende come argomento un documento XML

public static void main (String[] args) {
    ValidatingDOM validatingDOM = new ValidatingDOM (args[0]);
}
}
```



# Test

```
<?xml version"1.0"> <!DOCTYPE cliente SYSTEM "cliente.dtd">
<cliente>
  <nome>Marco</nome>
  <cognome>Bianchi</cognome>
</cliente>
```

cliente.dtd:

```
<!ELEMENT cliente (nome,cognome)>
  <!ELEMENT nome (#PCDATA)>
  <!ELEMENT cognome (#PCDATA)>
  <!ATTLIST cliente codice CDATA #REQUIRED>
```

Il doc XML passato come argomento alla classe ValidatingDOM:

**Error: org.xml.sax.SAXParseException: Attribute "codice" is required and must be specified for element type "cliente"**

Nell'elemento <cliente> manca l'attributo codice, dichiarato obbligatorio nel suo DTD (errore di validità).



# Test

```
<?xml version="1.0"?>
<!DOCTYPE cliente SYSTEM "cliente.dtd">
<cliente codice="101010">
  <nome>Luigi
  <cognome>Rossi</cognome>
</cliente>
```

Il doc XML passato come argomento alla classe ValidatingDOM:

**Fatal Error: org.xml.sax.SAXParseException: The element type "nome" must be terminated by the matching end-tag "</nome>"**

Il documento non è ben formato perché il tag “nome” è stato aperto ma non chiuso (**errore di ben formato**).