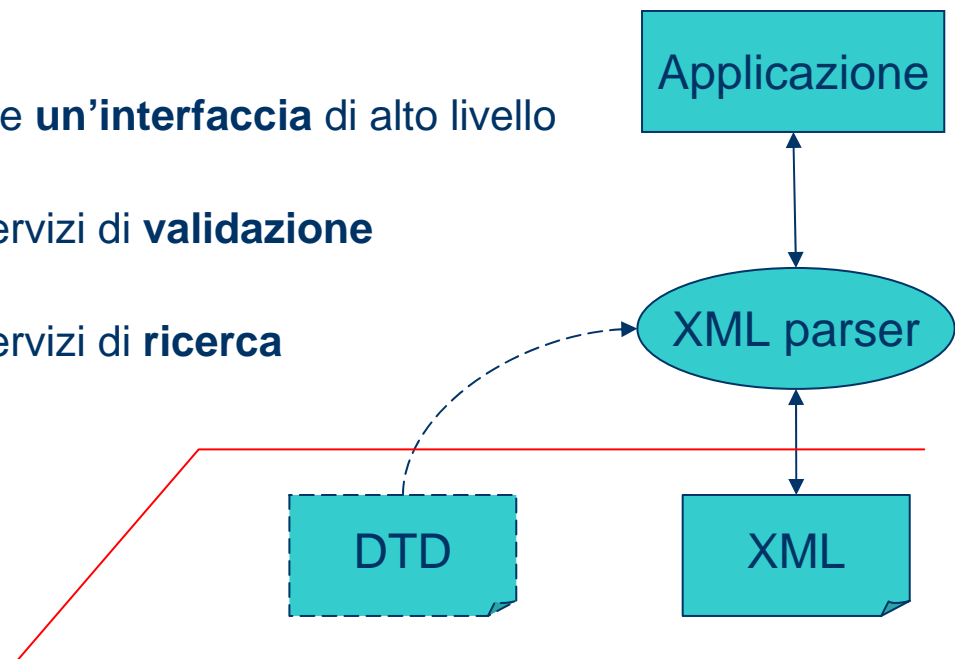


Simple API for XML Processing (SAX) Document Object Model (DOM)

Mario Arrigoni Neri

Il parser

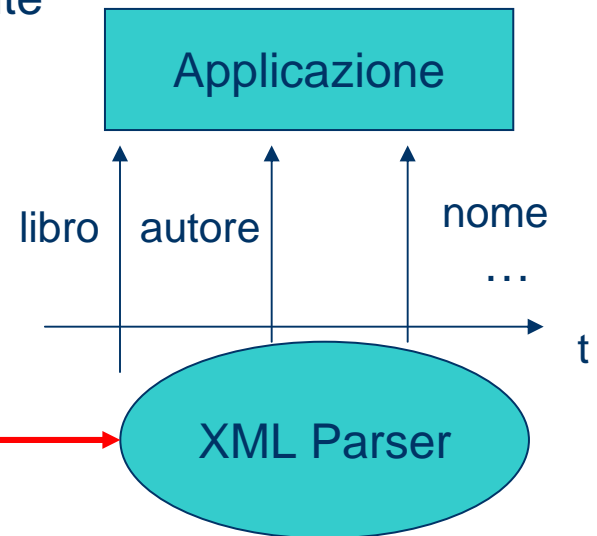
- Il parser si occupa di:
 - Recuperare il documento XML
 - Caricare i dati in memoria
 - Presentare all'applicazione **un'interfaccia** di alto livello
 - (opzionalmente) fornire servizi di **validazione**
 - (opzionalmente) fornire servizi di **ricerca**



Approccio ad eventi

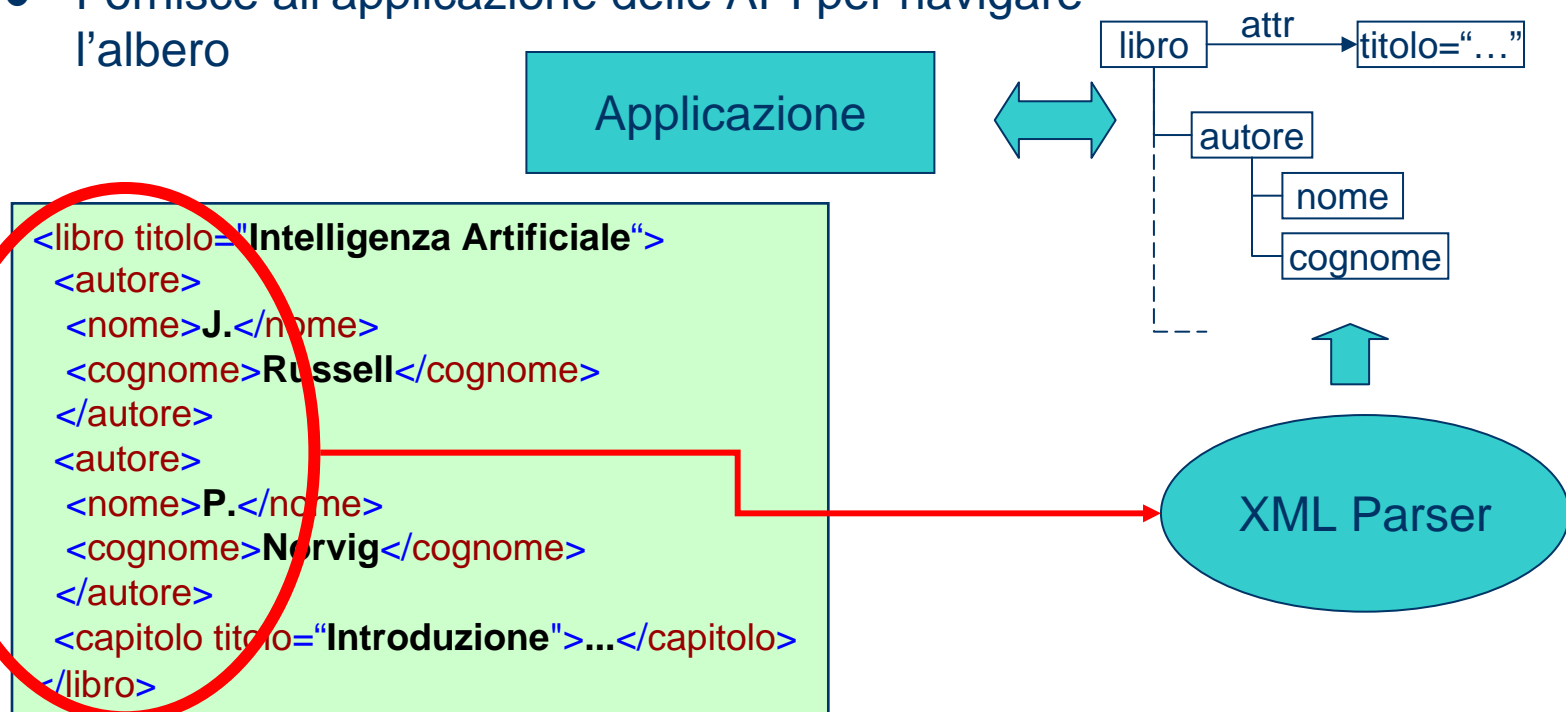
- Il parser scandisce l'intero file
- Per ogni elemento informa l'applicazione tramite la tecnica del callback

```
<libro titolo="Intelligenza Artificiale">  
  <autore>  
    <nome>J.</nome>  
    <cognome>Russell</cognome>  
  </autore>  
  <autore>  
    <nome>P.</nome>  
    <cognome>Norvig</cognome>  
  </autore>  
  <capitolo titolo="Introduzione">...</capitolo>  
</libro>
```

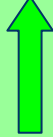



Approccio del modello

- Il parser costruisce una struttura ad albero che rappresenta il documento
- Fornisce all'applicazione delle API per navigare l'albero



Confronto tra gli approcci

	Approccio ad eventi	Approccio del modello
Pro 	<ul style="list-style-type: none">• E' molto "leggero"• Il programmatore può implementare solo le funzionalità necessarie	<ul style="list-style-type: none">• fornisce all'applicazione un modello ricco del documento• mantiene una rappresentazione completa e durevole in memoria
Contro 	<ul style="list-style-type: none">• Interfaccia troppo semplice = si richiede più codice nell'applicazione• Nessun supporto per operare sul documento	<ul style="list-style-type: none">• richiede una occupazione di memoria per tutto il documento

JAXP

- Java API for XML Processing (JAXP)
- Supporto **standard di SUN** per l'elaborazione di XML
- Propone due standard per le due impostazioni:
 - **SAX** (Simple API for XML) ad eventi
 - **DOM** (Document Object Model) basato su una rappresentazione astratta in memoria

SAX

- Simple API for XML
- Nasce come API alternativa a DOM per l'ambiente Java
- Successivamente adottato anche al di fuori del mondo Java (Microsoft)
- Basa il suo sistema di parsing sugli eventi
 - Inizio / fine di un **documento**
 - Apertura e chiusura di un **elemento**
 - Apertura e chiusura di un blocco **CDATA**

L'Handler di SAX – 1

- L'applicazione deve implementare l'interfaccia **org.xml.sax.helpers.ContentHandler**

```
Interface ContentHandler {  
    void setDocumentLocator(Locator locator);  
    void startDocument();  
    void startElement(String namespaceURI, String localName, String qName,  
        Attributes atts);  
    void startPrefixMapping(String prefix, String uri);  
    void characters(char[] ch, int start, int length);  
    void endDocument();  
    void endElement(String namespaceURI, String localName, String qName);  
    void endPrefixMapping(String prefix);  
    void ignorableWhitespace(char[] ch, int start, int length);  
    void processingInstruction(String target, String data);  
    void skippedEntity(String name);  
}
```


L'Handler di SAX – 2

- La classe **DefaultHandler** implementa l'interfaccia permettendo all'applicazione di ridefinire (override) solo i metodi desiderati

```
public class MyHandler extends DefaultHandler {
    public void startElement(String namespaceURI, String localName, String
        qualifiedName, Attributes att) throws SAXException {
        System.out.println("startElement: " + qualifiedName);
    }
    public void characters(char ch[], int start, int length) throws SAXException {
        System.out.println("characters " + start + " to " +
            (start + length - 1) + ": " + new String(ch, start, length));
    }
    public void endElement(String namespaceURI, String localName,
        String qualifiedName) throws SAXException {
        System.out.println("endElement: /" + qualifiedName);
    }
}
```

Utilizzo dell'handler

- Il client utilizza la classe di **Factory** per recuperare il parser
- Al parser viene chiesto di “parsare” il file come un processo autonomo

```
public class Ex1 {  
    public static void main(String args[]) throws Exception {  
        // create a parser  
        SAXParserFactory spf = SAXParserFactory.newInstance();  
        SAXParser saxParser = spf.newSAXParser();  
        XMLReader parser = saxParser.getXMLReader();  
        // create a handler  
        ContentHandler handler = new MyHandler();  
        // assign the handler to the parser  
        parser.setContentHandler(handler);  
        // parse the document  
        parser.parse("test.xml");  
    }  
}
```

```
graph TD; Code[Code] --> XML[XML Document]; XML --> Events[Events];
```

<?xml version="1.0"?>
<display>
 Hello World!
</display>

startElement: display
characters 32 to 31:
characters 0 to 0:

characters 34 to 49:
Hello World!
characters 0 to 0:

endElement: /display

DOM

- E' un'API (Application Programming Interface) per documenti XML
- Definisce una **modalità di rappresentazione** di documenti XML
- Utilizza strutture dati accessibili dall'applicazione
- Definisce le operazioni necessarie per operare sulla rappresentazione intermedia
 - **Parsing** di un file = caricamento della struttura
 - Costruzione di un **nuovo documento**
 - Browsing e **navigazione**
 - Aggiungere, eliminare, **modificare** e spostare le componenti
- Specifica le operazioni indipendentemente dal linguaggio, esistono implementazioni di DOM in linguaggi differenti:
 - Linguaggi **server-side** : java, C++, C#
 - Linguaggi di **script**: VB-Script, JavaScript

I livelli del DOM

- LIVELLO 0 :
 - Funzionalità originali di Netscape / IE
 - NON è una raccomandazione W3C
- LIVELLO 1 – Settembre 2000:
 - Oggetti fondamentali per il modello DOM
 - Non completamente compatibile con il livello 0
- LIVELLO 2 – Novembre 2000:
 - Fogli di stile
 - **validazione**
 - Incorpora un modello ad eventi per il controllo dei processi
- LIVELLO 3 - Febbraio 2004
 - Validazione **in-line** delle modifiche
 - ...

La struttura di DOM

- DOM vede i documenti come una struttura gerarchica ad albero, composta da oggetti di tipo *Node*
 - Ogni nodo può avere uno o più figli
 - In realtà in generale si lavora con una foresta di alberi
- DOM definisce solo la vista logica e non dice nulla sull'effettiva struttura in memoria
- Come modello ad oggetti si specificano:
 - Gli **oggetti** utilizzati per rappresentare il documento ed i suoi costituenti
 - Le **interfacce** “pubblicate” dagli oggetti per poter essere utilizzati dal livello applicativo
 - Le **relazioni** tra gli oggetti e tra le interfacce
 - La **semantica** associata ad ogni classe

Esempio di DOM

```
<sentence>
  The &projectName; <![CDATA[<i>project</i>]]> is
  <?editor: red?><bold>important</bold><?editor: normal?>.
</sentence>
```

```
+ ELEMENT: sentence
+ TEXT: The
+ ENTITY REF: projectName
+ COMMENT: The latest name we're using
+ TEXT: Eagle
+ CDATA: <i>project</i>
+ TEXT: is
+ PI: editor: red
+ ELEMENT: bold
+ TEXT: important
+ PI: editor: normal
```

Caricamento di un DOM

```
import org.w3c.dom.*;  
import javax.xml.parsers.*;  
import org.xml.sax.*;
```

DOM usa SAX

Implementazione
di JAXP

Definizione
DOM standard

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
DocumentBuilder builder = factory.newDocumentBuilder();  
Document doc = builder.parse("libro.xml");
```

- Il DOM di JAXP è definito nel package **org.w3c.dom**
 - Fornisce un'interfaccia standard che si appoggia su implementazioni differenti
- Sono disponibili implementazioni differenti (SUN, Apache, Oracle)
- L'implementazione di SUN è in javax.xml.parsers

Navigazione

- Estrazione del **nodo radice** e stampa del nome del nodo

```
Element root = doc.getDocumentElement();  
System.out.println(root.getNodeName());
```

- Nodi **figli**

```
NodeList children = root.getChildNodes();  
for (int i = 0; i < children.getLength(); i++)  
    System.out.println(children.item(i).getNodeName());
```

- **Attributi**

```
NamedNodeMap map = root.getAttributes();  
for (int i = 0; i < map.getLength(); i++)  
    System.out.println(map.item(i).getNodeName());
```


Tipizzazione – 1

- Tipizzazione tramite **attributi di descrizione**

```
n.getNodeType() == Node.ELEMENT_NODE
```

<i>Tipo di nodo</i>	<i>nodeName</i>	<i>nodeValue</i>	<i>attributes</i>
Element	Nome del tag	null	<i>NamedNodeMap</i>
Attr	Nome dell'attributo	Valore dell'attributo	null
Text	"#text"	Testo associato	null
CDATASection	"#cdata-section"	Testo associato	null
EntityReference	Nome dell'entità	null	null
Entity	Nome dell'entità	null	null
ProcessingInstruction	Valore dell'attributo target	Contenuto escluso l'attributo target	null
Comment	"#comment"	Testo associato	null
Document	"#document"	null	null
DocumentType	Nome del tipo di documento	null	null
DocumentFragment	"#document-fragment"	null	null
Notation	Nome della NOTATION	null	null

Tipizzazione – 2

- DOM in JAXP definisce una **gerarchia di classi** che corrisponde alla tassonomia dei nodi XML (Element, Attribute, ecc..)
- E' possibile combinare:
 - **Polimorfismo**: le librerie ritornano un'istanza della sottoclasse corretta
 - **Reflection**: il client può recuperare il tipo dinamico dell'istanza

```
If (n instanceof Element)
{
    ...
}
```

- Le due soluzioni si combinano. Grazie alla gerarchia ogni classe assegna l'attributo nel costruttore

Validazione

- Per default DOM è **non validante**

```
factory.setValidating(true);
```

- Una volta abilitata la validazione, questa avviene tramite **il DTD**

Error handling

- Eredita la gestione degli errori di **SAX**
- ma l'applicazione si “registra” come **listener** per gli eventi che segnalano un errore
- Tre tipi di errori: **fatali**, errori **semplici**, **warning**

```
class Handler implements ErrorHandler {  
    public void fatalError(SAXParseException ex) throws SAXException {  
        ex.printStackTrace();  
    }  
    public void error(SAXParseException ex) throws SAXParseException {  
        ex.printStackTrace();  
        throw ex;  
    }  
    public void warning(SAXParseException err) throws SAXParseException {  
        System.out.println("** Warning" + err.getLineNumber() + err.getMessage());  
    }  
}
```

```
builder.setErrorHandler(new Handler());
```

Browsing e stampa

```
public static void print(Node n, String spaces) {  
    if (n == null) return;  
    if (n instanceof Element) {  
        String s = spaces + n.getNodeName() + " (";  
        NamedNodeMap map = n.getAttributes();  
        if (map != null)  
            for (int i = 0; i < map.getLength(); i++)  
                s += map.item(i).getNodeName() + "=" + map.item(i).getNodeValue();  
        s += ")";  
        System.out.println(s);  
    } else if (n instanceof Text)  
        System.out.println(spaces + n.getNodeValue());  
    NodeList children = n.getChildNodes();  
    for (int i = 0; i < children.getLength(); i++)  
        print(children.item(i), spaces + "  - ");  
}
```

```
Element root = doc.getDocumentElement();  
print(root, "");
```

Esempio – 1

- XML in ingresso

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<libro      titolo="Introduzione ad XML">
  <autore>
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
  </autore>
  <autore>
    <nome>Marco</nome>
    <cognome>Bianchi</cognome>
  </autore>
  <capitolo titolo="CAP 1. XML e DTD">...</capitolo>
</libro>
```

Esempio – 2

- Output

```
libro ( titolo=Introduzione ad XML)
- autore ()
- - nome ()
- - - Mario
- - cognome ()
- - - Rossi
- autore ()
- - nome ()
- - - Marco
- - cognome ()
- - - Bianchi
- capitolo ( titolo=CAP 1. XML e DTD)
- - ...
```

Modifica dell'albero

- DOM permette di inserire **nuovi elementi nell'albero**

```
Element root = doc.getDocumentElement();  
Element cap = doc.createElement("capitolo");  
cap.setAttribute("titolo", "Introduzione");  
cap.appendChild(doc.createTextNode("... testo ..."));  
Node capitolo = doc.getElementsByTagName  
("capitolo").item(0);  
root.insertBefore(cap, capitolo);  
doc.normalize();
```

Creazione
del nodo

Creazione
dell'attributo

Ricerca del primo
capitolo presente

Inserimento

normalizzazione. Es: unisce
nodi di testo consecutivi