

Programare cu XML

Prezentare

Introducere	1
SAX – Simple API for XML	1
DOM – Document Object Model.....	4
Concluzii.....	7
Referințe	8

Introducere

După cum am arătat în primul laborator, XML se folosește pentru definirea unor fișiere de configurare, pentru implementarea unor limbaje (XSL, SVG, RSS), pentru definirea unor protocoale (de exemplu SOAP). Pentru a implementa instrumente care să prelucreze aceste documente rapid și eficient au fost dezvoltate mai multe metode în funcție de particularitățile de prelucrare ale fiecărui caz. Principalele metode de prelucrare ale fișierelor XML sunt DOM (Document Object Model) și SAX (Simple API for XML). În cele ce urmează vom discuta aceste două metode de prelucrare, cazurile lor de utilizare și exemple de implementare în 2 limbaje de nivel înalt (Java și Python)

SAX – Simple API for XML

SAX este un parser care prelucrează fișierele XML în mod serial declanșând evenimente la întâlnirea elementelor fișierului XML. Cu un parser de tip SAX programatorul nu are acces la structura efectivă a documentului, fiind necesar să implementeze mecanisme pentru a se asigura că prelucrează doar acele elemente pe care le dorește analizate.

Programatorul trebuie să scrie handleri (funcții asociate evenimentelor declanșate) pentru fiecare element de interes – element pe care dorește să-l prelucreze.

Principalele avantaje ale utilizării SAX sunt legate de faptul că permite prelucrarea streamurilor XML pe măsură ce se primesc, consumă mult mai puțină memorie decât parserele DOM și din acest motiv permite prelucrarea unor documente XML care nu pot fi încărcate complet în memorie și care deci nu pot fi prelucrate cu DOM.

Ca dezavantaje putem menționa faptul că validarea unui fișier XML cu un parser SAX este mai dificilă și necesită câteodată parcurgerea întregului fișier (de exemplu în cazul în care se folosește ID și IDREF trebuie parsat tot fișierul XML pentru a se ști dacă un IDREF nu punctează către un ID ce nu există)

În continuare vom ilustra modul în care se folosește SAX în Java și în Python.

În Java principalele pachete ce vor fi utilizate sunt `org.xml.sax` (<http://java.sun.com/j2se/1.5.0/docs/api/org/xml/sax/package-summary.html>) și `javax.xml.parsers` (<http://java.sun.com/j2se/1.4.2/docs/api/javax/xml/parsers/package-summary.html>) [6].

În primul din aceste pachete se află clasa `org.xml.sax.helpers.DefaultHandler` ce va fi extinsă pentru implementarea parserului dorit. Această clasă conține 5 funcții ce vor efectua practic procesarea documentului:

- `startDocument` – funcție apelată automat la întâlnirea elementului rădăcină
- `endDocument` – funcție ce va fi apelată la întâlnirea marcajului de sfârșit al elementului rădăcină
- `startElement` – funcție ce primește ca parametrii URI-ul namespace-ului documentului, numele local al elementului (numele ce nu include prefixul spațiului de nume), numele elementului ce include și prefixul spațiului de nume și care se găsește în documentație sub numele de „qualified name” sau „qname” și un element de tip `org.xml.sax.Attributes` ce constă practic dintr-o listă de atribute. În funcție de starea curentă a parserului și de acești parametri se decide comportamentul pe care trebuie să-l aibă parserul când întâlnește un nou element.
- `endElement` – ce are aceiași parametri cu `startElement`, mai puțin lista de atribute.
- `characters` – primește ca parametri un șir de caractere și datele referitoare la ce caractere din acest șir sunt utile; această funcție se folosește pentru tratarea conținutului efectiv al fișierului XML

Două alte clase importante sunt `javax.xml.parsers.SAXParserFactory` și `javax.xml.parsers.SAXParser`. `SAXParserFactory` este folosită pentru a crea o instanță de factory și o instanță de parser (v. Șabloane de design în programare, respectiv șablonul fabrică de obiecte http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29, http://en.wikipedia.org/wiki/Factory_method_pattern) .

`SAXParser` creează parserul și asignează clasa ce extinde clasa `DefaultHandler` și care conține funcțiile descrise mai sus.

Pentru a valida automat documentul XML trebuie ca fabrica de parsere să permită acest lucru. În acest scop `SAXParserFactory` oferă 2 metode:

- `setValidating(boolean)` – true dacă vrem ca parserele construite cu această fabrică să permită validare și false altfel
- `setNamespaceAware(boolean)` – true dacă vrem ca parserul să facă validarea folosind spațiile de nume descrise în document sau false dacă vrem să le ignorăm

Un exemplu destul de ușor de înțeles și totodată destul de complex de parser XML de tip SAX ce folosește toate aceste funcții poate fi găsit la <http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/sax/work/Echo10.java> [1]

În Python, un astfel de parser se scrie în mod similar. Vom exemplifica folosind un parser ce folosește biblioteca `sgmlib` – și deci merge atât pe fișiere XML cât și pe fișiere HTML ce nu sunt bine formate (adică nu sunt XHTML) [2]

Codul unui astfel de parser în Python este prezentat comentat în continuare și este comentat pe larg pentru a explica de ce este necesară fiecare funcție

```
import urllib, sgmlib
class SimpleHTMLParser(sgmlib.SGMLParser):
```

```

#parser pentru o pagina html oarecare ce extinde
#clasa sgmlib.SGMLParser

def parse(self, s):
    #metoda a clasei care parseaza stringul primit
    #toate metodele unei clase in Python trebuie sa primeasca
parametrul self
    self.feed(s)
    self.close()

def __init__(self, verbose=0):
    #un fel de constructor - prima functie care se apeleaza la
constructia unui obiect
    #in aceasta functie se initializeaza toate variabilele pe care
#doram sa le folosim
# pe prima linie se apelează functia de initializare a clasei pe care o
#extindem - similar cu super() în Java
    sgmlib.SGMLParser.__init__(self, verbose)
    self.startTitle=0
#variabilă ce ne va permite să știm când prelucrăm datele din cadrul
#unui element de tip title
    self.title=""
#variabilă în care vom reține titlul paginii

def start_title(self, attributes):
    #functie care se apeleaza la inceputul tag-ului title
#pentru a scrie o funcție care sa fie apelata la întâlnirea unui
#element trebuie sa ii dam nume de forma start_numeElement
#variabila attributes contine lista de atribute in perechi (nume,
#valoare)
    self.startTitle=1

def end_title(self):
    #incheiem tag-ul title deci resetam variabila startTitle
    self.startTitle=0

def handle_data(self, data):
    #functie care se apeleaza cand se intalnesc date
    if self.startTitle==1:
#daca suntem în interiorul tag-ului title atunci variabila startTitle
#este 1 si atunci stim ca variabila data contine textul continut in
#interiorul elementului title
        self.title=data

def printTitle(self):
    if self.title!="":
        print self.title

#main-ul - in acelasi fisier .py
fisierHtml = urllib.urlopen("http://google.com")
s = fisierHtml.read()
parser =SimpleHTMLParser()
parser.parse(s)
parser.printTitle()

```

După cum se poate observa principiile sunt absolut identice cu cele din API-ul Java, diferențele fiind în principal următoarele:

- în Java detectarea elementului în care suntem se face într-o singură funcție „startElement” pe baza atributelor acestuia iar în Python se construiește o funcție cu un nume standard pentru fiecare element pe care dorim să-l prelucrăm
- prelucrarea conținutului în Java se face folosind funcția characters, iar în Python funcția handle_data. Ambele ignoră datele dacă nu este prevăzut un comportament explicit în funcție de elementul în interiorul căruia ne aflăm (asta însă este mai mult asemănare decât deosebire☺)

DOM – Document Object Model

Spre deosebire de SAX, DOM este un standard W3C independent de platformă [3]. Tot spre deosebire de SAX, parsarea unui document XML folosind DOM presupune încărcarea întregului document în memorie. După ce documentul este încărcat navigarea printre nodurile documentului se poate face în orice ordine folosind relațiile de ordine din interiorul arborelui XML. Astfel se pot accesa rapid părinții, copiii și frații oricărui element. Aceste avantaje date de accesul rapid la conținut sunt compensate de consumul mare de memorie necesar de încărcarea documentelor XML complet în memorie. Totodată, în cazul unor documente XML foarte mari sau în cazul unor dispozitive cu resurse de memorie foarte limitate este posibil ca o prelucrare bazată pe DOM să nu poată fi făcută pentru că nu documentul nu încapă în memorie.

Ca și în cazul SAX există mai multe API-uri pentru prelucrarea bazată pe DOM a documentelor XML și tot ca în cazul SAX vom prezenta atât un exemplu în Java cât și unul în Python.

Pachetele Java pentru prelucrarea documentelor sunt org.w3c.dom (<http://java.sun.com/j2se/1.4.2/docs/api/org/w3c/dom/package-summary.html>) și același javax.xml.parsers (<http://java.sun.com/j2se/1.4.2/docs/api/javax/xml/parsers/package-summary.html>).

Java oferă, ca și pentru SAX, 2 clase ce vor contribui la parsarea documentului XML – o fabrică de parsere javax.xml.parsers.DocumentBuilderFactory și o clasă parser – javax.xml.parsers.DocumentBuilder. La fel ca și în cazul SAX clasa fabrică poate fi configurată pentru a valida documentele XML parsate folosind funcțiile setValidating și setNamespaceAware.

Diferența constă ca parserul DOM întoarce un element de tip org.w3c.dom.Document care oferă o serie de metode prin care i se pot accesa elementele, textul și atributele.

Cele mai importante metode oferite de această clasă sunt următoarele:

- getElementElement – ce oferă acces la elementul rădăcină al documentului
- getElementByID – întoarce elementul ce are ID-ul egal cu șirul de caractere primit ca parametru de funcție

- `getElementsByTagName` – întoarce elementele ce au numele egal cu șirul de caractere primit ca parametru de funcție, în ordinea în care sunt întâlnite în document

Aceste funcții întorc un obiect sau o listă de obiecte de tip `org.w3c.dom.Element`. `Element` este interfața ce reprezintă un element oarecare din XML și care permite în principal următoarele operații:

- `getAttribute` – primește numele unui atribut și întoarce un șir de caractere ce reprezintă valoarea atributului
- `getChildNodes` – întoarce o lista de `org.w3c.dom.Node` ce sunt copii ai elementului curent. Interfața `Node` reprezintă interfața extinsă de `Element` și care conține toate tipurile de noduri dintr-un fișier XML. Pentru a identifica tipul unui anumit nod în scopul prelucrării sale putem folosi funcția `getNodeType` care poate întoarce valorile cuprinse în tabelul următor (preluat de la [4])

Node	nodeName()	nodeValue()	attributes	nodeType()
Attr	name of attribute	value of attribute	null	2
CDATASection	#cdata-section	content of the CDATA Section	null	4
Comment	#comment	content of the comment	null	8
Document	#document	null	null	9
DocumentFragment	#document-fragment	null	null	11
DocumentType	document type name	null	null	10
Element	tag name	null	NamedNodeMap	1
Entity	entity name	null	null	6
EntityReference	name of entity referenced	null	null	5
Notation	notation name	null	null	12
ProcessingInstruction	target	entire content excluding the target	null	7
Text	#text	content of the text node	null	3

Următoarele 2 funcții exemplifică o parcurgere foarte simplă a unui arbore DOM – fără a ține cont însă de tipul nodurilor:

```

public static void checkSiblings(Node firstBrother)
{
    Node nextSibling=firstBrother.getNextSibling();
    //parcurgerea nodurilor "frati"
    while(nextSibling!=null)
        { //afisarea nodurilor frati

```

```

        System.out.println(nextSibling.getNodeName()+
"+nextSibling.getNodeValue()+" "+nextSibling.getNodeType());
//parcurgerea recursiva a copiilor nodurilor frati
        checkChildren(nextSibling);
        nextSibling=nextSibling.getNextSibling();
    }
}

public static void checkChildren(Node parent)
{
    Node son=parent.getFirstChild();
//parcurgerea nodurilor "copii"
    if(son!=null)
    {
//parcurgerea nodurilor frati
        System.out.println(son.getNodeName()+
"+son.getNodeValue()+" "+son.getNodeType());
//parcurgerea copiilor copilului curent
        checkChildren(son);
//parcurgerea nodurilor frati
        checkSiblings(son);
    }
}
}

```

Un exemplu mai complex de prelucrare a unui fișier XML folosind DOM poate fi găsit aici:

<http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/dom/work/DomEcho02.java>

În Python prelucrarea se desfășoară în mod similar[5]. Pentru a arăta ceva nou vom exemplifica crearea unui fișier XML folosind biblioteca xml.dom din Python. Programul pe care îl vom realiza va construi următorul document XML și-l va scrie într-un fișier.

```

<?xml version='1.0' encoding='UTF-8'?>
<xmlTest xmlns='http://cs.pub.ro/ie' atribut1='valoare'>
  <test atribut2='alta valoare'>text continut in interiorul unui
element</test>
</xmlTest>

```

Principalele funcții folosite de program din biblioteca xml.dom sunt următoarele:

- createElementNS – primește ca atribute un spațiu de nume și numele unui element și întoarce un element (metodă a clasei Document și a clasei Element – moștenite de la clasa Node – după cum se vede aproape identic cu API-ul din Java)
- appendChild – se adaugă un copil nodului (Document, Element) current
- createAttributeNS – adaugă un atribut elementului current
- createTextNode – construiește un nod de tip text [7]

```

#importam bibliotecile de functii necesare pentru utilizarea DOM in
#Python
import xml.dom.ext
import xml.dom.minidom

```

```

#construim un nou document
doc = xml.dom.minidom.Document()
#definim un spatiu de nume - un string ce va fi dat ca parametru spatiu
#de nume la crearea elementelor sau atributelor
namespace="http://cs.pub.ro/ie"
#construim elementul radacina
root_element=doc.createElementNS(namespace,"xmlTest")
#construim un atribut asignat elementului radacina
root_element.setAttributeNS(namespace, "atribut1", "valoare")
#adaugam elementul radacina la documentul xml
doc.appendChild(root_element)

#mai construim un element pe care il adaugam la elementul radacina
test_element=doc.createElementNS(namespace,"test")
test_element.setAttributeNS(namespace, "atribut2", "alta valoare")
root_element.appendChild(test_element)

#construim un nod de tip text pe care il adaugam in interiorul
#elementului creat anterior
pcdata=doc.createTextNode("text continut in interiorul unui element")
test_element.appendChild(pcdata)

#deschidem un fisier pentru scriere (se creeaza unul nou daca nu
#exista)
fisier = open("d:\\test.xml", "w")
#scriem documentul in fisierul deschis
xml.dom.ext.PrettyPrint(doc, fisier)
#inchidem fisierul
fisier.close()

```

După cum se poate observa DOM oferă mecanisme relativ simple pentru crearea și prelucrarea fișierelor XML. Principiile sunt aproape identice în limbaje diferite de programare deoarece se bazează pe standarde comune.

Concluzii

Prelucrarea fișierelor XML este deosebit de facilă existând un număr mare de API-uri în majoritatea limbajelor de programare de nivel înalt. Există modalități de prelucrare diferite și alegerea unei metode se bazează pe mai multe aspecte dintre care cele mai importante ar fi următoarele:

- dimensiunea fișierului vs. cantitatea de resurse de memorie disponibile
- tipul de prelucrare efectuat (1 singură parcurgere suficientă sau necesare prelucrări succesive)

În documentul de față am prezentat exemple de utilizare atât pentru SAX cât și pentru DOM în două limbaje de programare destul de diferite, constatând similarități majore între API-uri. Astfel, putem concludiona că dacă putem folosi aceste tehnologii într-un limbaj de programare de nivel înalt ne putem adapta rapid la API-urile din alt limbaj.

O ultimă concluzie ar fi că în funcție de aplicația pe care o avem de dezvoltat putem alege (dat fiind că API-urile din limbaje diferite sunt foarte similare) limbajul în care putem dezvolta cel mai comod și mai facil. Rapiditatea și facilitatea dezvoltării sunt de altfel și motivele pentru care am ales să prezentăm API-urile Python în acest document.

Referințe

1. Tutorialul pentru SAX oferit de Sun
<http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/sax/index.html>
2. Prelucrarea HTML în Python – <http://www.boddie.org.uk/python/HTML.html>
3. Standardul DOM pe site-ul W3C - <http://www.w3.org/DOM/>
4. Tutorialul pentru DOM oferit de Sun –
<http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/dom/index.html>
5. Prelucrarea XML în Python – http://www.boddie.org.uk/python/XML_intro.html
6. Sun API Javadocs – <http://java.sun.com/j2se/1.4.2/docs/api/>
7. Python Library Reference – Structured Markup Processing Tools
<http://docs.python.org/lib/markup.html>