Java - XML

Code: java-xml

Originaux

url: **http://tecfa.unige.ch/guides/tie/html/java-xml/java-xml.html**
url: **http://tecfa.unige.ch/guides/tie/pdf/files/java-xml.pdf**

Auteurs et version

- Daniel K. Schneider

- Version: 0.3 (modifié le 20/4/00)

Prérequis: Java de base, servlets, GUI et XML

Module technique précédent: java-intro
Module technique précédent: java-jhtml
Module technique précédent: java-servl
Module technique précédent: java-swing
Module technique précédent: xml-dom
Module technique précédent: xml-tech

Objectifs

1. JAVA - XML de base
   - DOM API
   - SAX API
2. simples servlets
   - DOM
   - SAX

- .[C'est pas fini, c'est fragile, ... A SUIVRE]

1. Java & XML: un mariage de raison

1.1 Possibilités

Atouts généraux:

- Processing load du "viewing" côté clients
- Internet Friendliness des deux:
  - Plateform et Vendor Independance de XML ET de Java
  - Fait spécialement pour le réseau
  - Unicode (ok pour toutes les langues)

A. Server-side

- traduction sur mesure de XML vers HTML
- en règle générale on utilise plutôt un publication framework basé sur XSL (comme Cocoon)

B. Visualisation client-side

- "Viewing" flexible selons besoins des utilisateurs
  - soit applications ou applets pour visualiser/regarder le contenu d'un ou plusieurs fichiers XML

C. Echange de données

- Middle-tier application (bridge entre bases de données et applications)
- Mediation entre différents types d'application

D. Edition

- Avec des applications ou applets Java on peut construire des éditeurs spécialisées à certains types de DTD's

2. APIs et "standards"

2.1 Parseurs XML

- Il existe plusieurs parseurs XML
  - On conseille Xerces de Apache basé sur xml4j de IBM: http://xml.apache.org/
  - A voir: l'intégration avec le JAXP de Sun (http://java.sun.com/xml/)
  - voir la page XML pour une liste d'autres packages
- Installation
  - Il vous faut soit télécharger le package entier (*.jar, documentation, exemples, etc.) de Apache
  - Sinon il faut juste xerces.jar (vérifier la version), par ex dans http://tecfa.unige.ch/guides/java/classes/
- Xerces sous Solaris à TECFA et sur le Web:
  - tout Xerces se trouve dans /local/java/xerces
  - Doc: http://tecfa2.unige.ch/guides/xml/local/xerces/docs/html/
  - Exemple: http://tecfa2.unige.ch/guides/xml/local/xerces/samples/
  - Initialisation de l'environnement: 'source /local/env/java12-xalan.csh'
- Applets et servlets
  - xerces.jar doit se trouver dans le classpath du servlet (ok pour Apache/jserv à Tecfa) et des applets.

2.2 La notion de parsing

Parsing is the process of splitting up a stream of information into its constituent pieces (often called tokens). In the context of XML, parsing refers to scanning an XML document (which need not be a physical file -- it can be a data stream) in order to split it into its various elements (tags) and their attributes. XML parsing reveals the structure of the information since the nesting of elements implies a hierarchy ....

- Validating parsers: vérifient que le document est conforme au DTD
- Non-validating parsers: vérifie juste la "well-formedness".

2.3 DOM Parsing

- Spécification pour traduire un fichier XML en un "arbre informatique" qui réside en mémoire.
- Java API Specification
  - Dom (level 1) Spec: **http://www.w3.org/TR/REC-DOM-Level-1/**
  - L'implémentation se trouve (normalement) dans le package org.w3c.dom
  - Dans Xerces:
- Définition officielle:

  The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.
- Avantage / utilité
  - permet de manipuler un document sous forme de structure de données
- Mécanisme du tree-based parsing (utilisé pour le DOM)
  - Tout le document parsé est traduit en un arbre qui réside dans la mémoire
  - on parle de "document-centered view")
  - Tous les éléments et attributs sont disponibles en même temps (une fois que le parsing a terminé)

2.4 SAX (The simple API for XML)

A. Nom et libraries

- SAX = "Simple API for XML"
  (interface standard pour event-based XML parsing)
  url: **API: http://www.megginson.com/SAX/index.html**
- Plusieurs implémentations en Java, Python, etc.

B. Packages JAVA

- Les packages Java sont à des endroits variés, mais souvent dans org.xml.sax.

C. Définition du mécanisme SAX

- il s'agit d'une "event-based parsing API", on parle de "data centric view"

1. Une source d'input envoie le document dans un parseur
2. Le parseur fait appel a des "callback" (que l'utilisateur doit implémenter) et qui indique ce qu'il voit dans le document.
3. Pour chaque type d'élément le callback est différent

- Note: ce principe ressemble à la programmation GUI

D. Exemple abstrait:

XML:

```
<fiche>
   <auteur> DKS </auteur>
   <titre>Ma fiche </titre>
</fiche>
```

Parsing (événements)

```
début document
début élément: fiche
début élément: auteur
charactères: DKS
fin élément: auteur
........
fin élément: fiche
fin document
```


E. Avantages et désavantages du modèle event-based:

Avantages = efficacité (cachée)

- moins gourmand en mémoire (surtout pour des documents larges)
- plus rapide
- vous pouvez représenter les documents comme vous voulez (si c'est utile)

Désavantages = difficultés à programmer

- un élément recraché par le parseur n'a parfois pas assez de sens pour le traiter tout-de-suite.
- Souvent, il faut stocker de l'information dans des structures de données (piles ou vecteurs) pour un usage ultérieur
- la méthode "characters()" ne retourne pas forcément tous les charactères d'un élément en une seule fois (il paraît)

3. Simple DOM

3.1 Principe de base

Exemple 3-1: Simple application qui affiche les éléments d'un arbre

url: **http://tecfa.unige.ch/guides/java/staf2x/ex/xml/formcont/simple-dom/**

Syntaxe: java SpitElements <nom du fichier>

```
java SpitElements ../formcont.xml
```

Les classes de base

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.*;
```

Initialiser un Parseur, parser le document et prendre l'élément root

```
DOMParser parser = new DOMParser();
parser.parse(filename);
// The document is the root of the DOM tree.
Document doc = parser.getDocument();
// Get the Document Element (Top Node)
Element root = (Element)doc.getDocumentElement();
// Do something with this Node
walkDOMBranch(root);
```


Simple traversé d'un arbre

```
    public static void walkDOMBranch(Node node) {

  // print the name and value (characters) of the Node
  System.out.println(node.getNodeName()+":"+node.getNodeValue());

  // if the node has children do the same thing for each child
  if (node.hasChildNodes()) {
     NodeList nodeList = node.getChildNodes();
     int size = nodeList.getLength();
     for (int i = 0; i < size; i++) {
      walkDOMBranch(nodeList.item(i));
      }
  }
    }
```

A retenir:

- la méthode de NodeList `getLength` donne la longeur de la liste
- la méthode de NodeList `item` permet d'obtenir l'élément suivant

La classe `Element: Elements represent most of the "markup" and structure of the document. They contain both the data for the element itself (element name and attributes), and any contained nodes, including document text (as children). Elements may have Attributes associated with them; the API for this is defined in Node, but the function is implemented here.`

3.2 Gestion d'erreurs

Le minimum: attraper une exception autour du parse et imprimer un StackTrace

```
try { ......
    parser.parse(filename);
   ......
 } catch (Exception e) {
      System.out.println("Something is wrong ...");
      e.printStackTrace(System.err)     }
```

- A chaque fois où on lit, parse où traverse un arbre il faut un erreur handling !

Exemple 3-2: Simple application qui affiche les éléments d'un arbre II

url: voir SpitElements2.java

SAXParseException est levée quand le parseur tombe sur une erreur XML

- notez les méthodes getLineNumber et getColumnNumber, qui sont bien utiles pour l'utilisateur !

```
catch (SAXParseException e) {
    System.out.println("The File is not well formed.");
    System.out.println(e.getMessage()
           + " at line " + e.getLineNumber()
           + ", column " + e.getColumnNumber());
    }
```


Erreurs SAX non identifiés

```
catch (SAXException e) {
    System.out.println("SAX exception found");
    System.out.println(e.getMessage());
    e.printStackTrace(System.out);
    }
```

Si la lecture/écriture pose problème

```
catch (IOException e) {
    System.out.println("Beurk: IOException " + e);
    }
```

Tout ce qui peut encore arriver

```
catch (Exception e) {
    e.printStackTrace(System.out);
```

Note:

- Si ne me trompes pas, il n'existe pas d'erreurs spécifiques liés au DOM parsing
- Vous utilisez donc la même stratégie pour le SAX parsing.

3.3 Filtrage avec un simple Servlet

Exemple 3-3: Simple servlet qui affiche les éléments d'un arbre

url: **http://tecfa.unige.ch/guides/java/staf2x/ex/xml/formcont/servl-dom/FormContDOMTree.java**
url: **http://tecfa2.unige.ch/servlets/FormContDOMTree**

A. getElementsbyName

- La méthode "getElementsbyName" permet de collectionner tous les éléments d'un même nom dans une liste de type `NodeList`

```
Document doc = parser.getDocument();
printElements(doc.getElementsByTagName("title"));
```

B. Traitement d'une NodeList

```
......
public void printElements(NodeList listOfNodes) {
 for (int i=0; i<listOfNodes.getLength();i++) {
   Element node = (Element) listOfNodes.item(i);
   out.println("Node Name = " + node.getNodeName());
   // out.println("Node Value = " + node.getNodeValue());
   out.println("String Value= " + node.getFirstChild().getNodeValue());
  }
}
```


A retenir:

- Rappel: La classe `Element: They contain both the data for the element itself (element name and attributes), and any contained nodes, including document text (as children).`
- Donc le contenu d'un node est un sous-node !!
- la méthode Element `getNodeValue()` permet d'obtenir la valeur (contenu) d'un noeud. Pour les nodes de type Text, CDATASection, ProcessingInstruction, and Comment la valeur est un string et c'est cela qui nous intéresse ici.
- la méthode Element `getFirstChild()` retourne le premier enfant.

Exemple 3-4: Simple servlet qui affiche en HTML les éléments d'un arbre

url: **http://tecfa.unige.ch/guides/java/staf2x/ex/xml/formcont/servl-dom/**
url: **http://tecfa2.unige.ch/servlets/FormContDOMEl2**

Exemple 3-5: Simple servlet qui demande (en GET) un URL et un tag

url: **http://tecfa2.unige.ch/servlets/FormContDOMEl3**

On cherche les paramètres

```
String  xmlURL = req.getParameter ( "url" );
String  tag = req.getParameter ( "tag" );
```

... et on les utilise

```
parser.parse(xmlURL);
getElements(doc.getElementsByTagName(tag));
```

4. Simple SAX

4.1 Sensibilisation

Exemple 4-1: Simple Listing d'éléments avec SAX

url: **http://tecfa.unige.ch/guides/java/staf2x/ex/xml/formcont/simple-sax/**

A. Utilisation d'une fabrique et définition d'un handler

Syntaxe: Parser parser = ParserFactory.makeParser(....);

```
Parser parser = ParserFactory.makeParser
                ("org.apache.xerces.parsers.SAXParser");
// create a handler
DocumentHandler handler = new SpitElements ();
parser.setDocumentHandler(handler);
....
parser.parse(...)
```

B. Implémentation du callback

Soit:

- extension de `HandlerBase`
- implementation de `DocumentHandler`

Exemple 4-2: SAX parsing par extension de HandlerBase

url: **/guides/java/staf2x/ex/xml/formcont/simple-sax/SpitElements.java**

```
public class SpitElements extends HandlerBase {

    public void startElement (String name, AttributeList atts)
            throws SAXException
    {
        System.out.println("Start element: " + name);
    }

    public void endElement (String name) throws SAXException
    {
        System.out.println("End element: " + name);
    }

    public void characters(char[] charArray, int start, int length)
   throws SAXException {
        String content = new String(charArray, start, length);
        System.out.println(" Contents: " + content);
    }
```

Comme il s'agit d'un extension, on n'implémente que:

- startElement (String nom, AttributeList atts)
- endElement (String nom)
- characters (char[] ch, int start, int length)

Exemple 4-3: SAX parsing par implémentation de DocumentHandler

url: **/guides/java/staf2x/ex/xml/formcont/simple-sax/SpitElements2.java**

```
public class SpitElements2 implements DocumentHandler {
   public void setDocumentLocator(Locator locator) { }
   public void startDocument() throws SAXException {
        System.out.println("--------- START");  }
   public void endDocument() throws SAXException {
        System.out.println("--------- END");
    }
   public void startElement (String name, AttributeList atts)
        throws SAXException {
        System.out.println("Start element: " + name);  }
   public void endElement (String name) throws SAXException {
       System.out.println("End element: " + name); }
   public void characters(char[] charArray, int start, int length)
       throws SAXException {
       String content = new String(charArray, start, length);
       System.out.println(" Contents: " + content);
    }
   public void ignorableWhitespace(char[] text, int start, int length)
       throws SAXException { // do nothing here
   }
   public void processingInstruction(String target, String data) {}
}
```

- On est obligé d'implémenter toutes les méthodes call-back, même si on ne fait rien

4.2 Simple servlet

Exemple 4-4: Simple SaX Servlet qui liste les éléments d'un fichier (fixe)

url: **/guides/java/staf2x/ex/xml/formcont/servl-sax/FilterFormContServlet2.java**
url: **http://tecfa2.unige.ch/servlets/FilterFormContServlet2**

Ce servlet est basé sur le code des exemples précédents

Seules différences notables:

- On construit une class interne `MyDocumentHandler` qui implémente DocumentHandler (qu'on aurait pu mettre dans un autre fichier)
  - on n'aurait pas pu implémenter ca avec la classe du servlet (j'ai essayé)

```
public class FilterFormContServlet2 extends HttpServlet {
  private class MyDocumentHandler implements DocumentHandler {
  public void setDocumentLocator(Locator locator) {}
  .... }
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
     throws ServletException, IOException {
  try {
     // create a handler
     Parser parser = ParserFactory.makeParser (......) ;
     DocumentHandler handler = new MyDocumentHandler ();
     parser.setDocumentHandler(handler);
     parser.parse(xmlURL);
    }
```

- On doit gérér le PrintWriter (il doit être accessible par les méthodes de cette classe)
  - On a une variable globale: `PrintWriter out;`
- On fait du HTML
  - Notez le rôle de la méthode `printTail ()`

```
public void printTail () {
  out.println("</pre>");
  out.println("</body>");
  out.println("</html>");
  out.flush();
  out.close();
}
```

- Attention, ce fichier crée 2 fichiers.classe qu'il faudra copier (un pour la classe principale et un autre pour la classe interne)

```
FilterFormContServlet2$MyDocumentHandler.class
FilterFormContServlet2.class
FilterFormContServlet2.java
```


4.3 Un servlet qui traduit XML vers HTML

Exemple 4-5: Simple SaX Servlet qui liste les éléments d'un fichier (fixe)

url: **http://tecfa2.unige.ch/servlets/FicheServletSAX**

url: **http://tecfa2.unige.ch/guides/java/staf2x/ex/xml/fiches/SAX/**

- Cet exemple montre qu'il est plus difficile de travailler avec SAX que DOM
  - lorsqu'on traite les characters (contenus dans des elements/tags) il faut savoir où on est (dans quel tag) pour décider s'il faut faire un markup spécial (<dd> dans notre cas).

.... mais que c'est rapide :)

- la logique du servlet correspond exactement à l'exemple précédent, la différence ici c'est utilisation de HandlerBase est un traitement amélioré

```
public class MySaxHandler extends HandlerBase {

 // on ne se fatigue pas trop ici :=)
 String current;

 public void startElement (String name, AttributeList atts)
    throws SAXException
 {

     if (name.equals("Title"))
     out.println("<hr><h2>");
```



```
     else if (name.equals("Entry")||name.equals("List")) { }
            else out.println ("<dl><dt>" + name + "<dt>");
     current = name;
 }

 public void characters(char[] charArray, int start, int length)
     throws SAXException {
     String content = new String(charArray, start, length);
     if (current.equals("Title"))
     out.println(content);
     else out.println ("<dd>" + content + "<dd>");
 }

 public void endElement (String name)
 {
     if (name.equals("Title"))
     out.println("</h2>");
     else if (name.equals("Entry")||name.equals("List")) { }
     else out.println ("</dl>");
     out.println("<!-- end" + name + "-->" );
 }
   }
```

- A faire: TRAITER les attributs (particulièrement horrible, car normalement il faut les imprimer après avoir vu le end-tag)
- Voir nos exemples PhP, où il se pose exactement le même problème.
- .....

5. Affichage de simples structures XML

5.1 Simples fiches (sans attributs)

Exemple 5-1: Fiches.xml: list sous forme de table

```
 <!ELEMENT List (Entry+)>
 <!ELEMENT Entry (Title, Creator, Description, Keywords, Location,
MetaDocumentAuthor, MainType, SubType, Topic, SubTopic)>
 <!ELEMENT Title (#PCDATA)>
 <!ELEMENT Creator (#PCDATA)>
 <!ELEMENT Description (#PCDATA)>
 <!ELEMENT Keywords (#PCDATA)>
 <!ELEMENT Location (#PCDATA)>
 <!ELEMENT MetaDocumentAuthor (#PCDATA)>
 <!ELEMENT MainType (#PCDATA)>
 <!ELEMENT SubType (#PCDATA)>
 <!ELEMENT Topic (#PCDATA)>
 <!ELEMENT SubTopic (#PCDATA)>
```

Exemple 5-2: Fiches2.xml : liste à plat avec qq éléments répétitifs

```
<!ELEMENT Entry (Title, Creator?, Description, Keywords, Location?,
MetaDocumentAuthor, MainType, SubType*, Topic, SubTopic*)>
.... sinon les mêmes éléments !
```

5.2 Simples Servlets XML->DOM->HTML

url: Sources: **http://tecfa2.unige.ch/guides/java/staf2x/ex/xml/fiches/v2/**

Exemple 5-3: Affichage d'une structure de table sous forme d'arbre

url: **http://tecfa2.unige.ch/guides/java/staf2x/ex/xml/fiches/v2/fiches-arbre-fixe.jsp**

Exemple 5-4: Affichage d'une structure de quasi-table sous forme d'arbre

url: **http://tecfa2.unige.ch/guides/java/staf2x/ex/xml/fiches/v2/fiches-arbre.jsp**

Exemple 5-5: Quasi-table sous forme d'abre avec URL et tags variables

url: **http://tecfa2.unige.ch/guides/java/staf2x/ex/xml/fiches/v2/table-arbre.jsp**

Exemple 5-6: Affichage d'une structure de quasi-table sous forme de table

url: **http://tecfa2.unige.ch/guides/java/staf2x/ex/xml/fiches/v2/fiches.jsp**

A FAIRE: Une liste d'idiômes DOM