Document Object Model

Fabrice Rossi

26 janvier 2003

Les exercices présentés dans ce document sont construits en partie grâce aux exercices de [4] et [5].

1 Instructions spécifiques au CRIO UNIX

Au début de chaque séance :

1. dans chaque terminal, utilisez tcsh (en tapant `tcsh`)
2. effectuez les réglages nécessaires au bon fonctionnement des différents programmes utilisés, en tapant `source /home/perm/ufrmd/rossi/xml/env.csh`

2 Traitement direct d'un document DOM

2.1 Chargement d'un document

Pour construire un arbre DOM, il faut créer un analyseur syntaxique, puis lui demander de réaliser le chargement du document. En `Java`, il existe deux solutions : utiliser l'API `JAXP` (Java API for XML Processing) ou DOM Level 3 (module *Load and Save*) qui n'est malheureusement pas encore finalisée. Pour sauvegarder un document, seule l'api DOM déjà citée est portable (mais encore sous forme de document de travail).

L'api JAXP utilise le *design pattern* (cf [1]) d'usine abstraite. On appelle une méthode de classe pour obtenir une usine. Le type de l'objet renvoyé peut être configuré à l'exécution du programme grâce à une propriété (entres autres). L'usine peut ensuite être configurée, puis utilisée pour obtenir un analyseur DOM. L'analyseur peut ensuite charger un document, être utilisé pour obtenir une `DOMImplementation` (qui permet quelques manipulations intéressantes comme la création d'un nouveau document vide), etc. Pour obtenir des messages d'erreurs personnalisés, on peut enregistrer un `ErrorHandler` (de SAX) auprès de l'analyseur. Voici un exemple de chargement de document :

LoadJAXP

```
1  import org.w3c.dom.Document;
2  import org.xml.sax.SAXException;
3  import java.io.IOException;
4  import javax.xml.parsers.DocumentBuilderFactory;
5  import javax.xml.parsers.FactoryConfigurationError;
6  import javax.xml.parsers.ParserConfigurationException;
7  import javax.xml.parsers.DocumentBuilder;
8  public class LoadJAXP {
9    public static void main(String[] args) {
10     if(args.length<1 || args.length>2) {
11       System.err.println("usage: LoadJAXP [-valid] xmlfile");
12       System.exit(1);
13     }
14     boolean validation=false;
15     String xmlFile=args[0];
16     if(args.length==2) {
17       if(!args[0].equals("-valid")) {
18         System.err.println("usage: LoadJAXP [-valid] xmlfile");
```

```
19          System.exit(1);
20        } else {
21          validation=true;
22          xmlFile=args[1];
23        }
24      }
25      // création de l'usine
26      DocumentBuilderFactory usine=null;
27      try {
28        usine=DocumentBuilderFactory.newInstance();
29      } catch(FactoryConfigurationError e) {
30        System.err.println("Impossible de créer l'usine : "+e);
31        System.exit(1);
32      }
33      // configuration
34      // prise en compte des espaces de noms
35      usine.setNamespaceAware(true);
36      // activation de la validation (suivant l'option -valid)
37      usine.setValidating(validation);
38      // suppresion des blancs ignorables
39      usine.setIgnoringElementContentWhitespace(true);
40      DocumentBuilder analyseur=null;
41      try {
42        analyseur=usine.newDocumentBuilder();
43      } catch(ParserConfigurationException e) {
44        System.err.println("Impossible de créer l'analyseur : "+e);
45        System.exit(1);
46      }
47      // gestionnaire d'erreur personnalisé
48      analyseur.setErrorHandler(new PrintError());
49      // on peut enfin demander une analyse
50      Document doc=null;
51      try {
52        doc=analyseur.parse(xmlFile);
53      } catch(SAXException e) {
54        System.err.println("Erreur d'analyse syntaxique : "+e);
55        System.exit(1);
56      } catch(IOException e) {
57        System.err.println("Erreur de chargement du document : "+e);
58        System.exit(1);
59      }
60      // on peut enfin travailler sur doc ...
61    }
62  }
63
```

Dans l'exemple proposé, la validation ne se fait que relativement à une DTD. Pour obtenir une validation par rapport à un schéma W3C ou RELAX NG, il faut utiliser des constructions plus spécifiques à chaque analyseur.

La solution proposée par DOM Level 3 LS n'est pas encore implantée de façon complète et nous ne l'utiliserons pas pour l'instant. Son principe est très proche de celui de JAXP.

2.2 Manipulations de base

Une fois le document chargé, on peut travailler sur l'object `Document`. L'interface `Document` représente le document XML dans son ensemble et propose de nombreuses méthodes, décrites dans [3]. Les méthodes

sont essentiellement utiles pour créer des nouveaux éléments, attributs, etc. dans un document. Pour le traitement, les trois méthodes les plus utiles sont les suivantes :

`Element getDocumentElement()`
: Renvoie la racine de l'arbre XML.

`NodeList getElementsByTagNameNS(String URI,String localName)`
: Renvoie une liste d'éléments dont le nom local et l'URI de *namespace* sont ceux indiqués par les paramètres.

`NodeList getElementsByTagName(String name)`
: Même chose que la méthode précédente, mais en sélectionnant par le nom qualifié des éléments.

L'interface `Node` (racine de la hiérarchie de classes de DOM) est la plus importante de l'api DOM, car chaque partie (élément, texte, attributs, etc.) d'un document XML est représenté par une classe qui implante `Node`. Contentons nous pour l'instant de quelques méthodes de cette interface :

`String getNamespaceURI()`
: Renvoie l'URI du *namespace* associé au nœud (`null` pour autre chose qu'un attribut ou un élément).

`String getLocalName()`
: Renvoie la partie locale du nom du nœud (`null` pour autre chose qu'un attribut ou un élément).

`String getPrefix()`
: Renvoie le préfixe du nom du nœud (`null` pour autre chose qu'un attribut ou un élément).

`String getNodeName()`
: Renvoie le nom qualifié du nœud (`null` pour autre chose qu'un attribut ou un élément).

`String getNodeValue()`
: Renvoie `null` sauf pour un nœud attribut ou texte, pour lequel le résultat est la valeur de l'attribut ou du texte.

`NodeList getChildNodes()`
: Renvoie la liste des enfants du nœud appelant (attention, un attribut n'est pas un enfant de son élément correspondant).

`Node getParentNode()`
: Renvoie le père du nœud appelant dans l'arbre (`null` pour la racine de l'arbre).

Notons qu'il existe de nombreuses méthodes de navigation (comme par exemple `getPreviousSibling` qui permet d'obtenir le prédécesseur du nœud appelant dans la liste des fils du père de ce même nœud). Avant de donner des exemples, terminons par l'interface `NodeList` qui représente (comme son nom l'indique) une liste de nœuds :

`int getLength()`
: Renvoie le nombre de nœuds contenus dans la liste.

`Node item(int index)`
: Renvoie le nœud numéro `index` (le premier élément de la liste porte le numéro 0).

Considérons maintenant l'exemple de fichier XML suivant :

disques.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE disques SYSTEM "DisquesML.dtd">
3  <disques>
4  <groupe nom="muse">
5  <nom>MUSE</nom>
6  <membre>Matthew Bellamy</membre>
7  <membre>Dominic Howard</membre>
8  <membre>Chris Wolstenholme</membre>
9  </groupe>
10 <groupe nom="feeder">
11 <nom>FEEDER</nom>
12 <membre>Grant Nicholas</membre>
13 <membre>Taka Hirose</membre>
```

```
<membre>Jon Henry Lee</membre>
</groupe>
<disque>
<interprète nom="muse"/>
<titre>Showbiz</titre>
</disque>
<disque>
<interprète nom="feeder"/>
<titre>Echo Park</titre>
</disque>
<disque>
<interprète nom="muse"/>
<titre>Origin of symmetry</titre>
</disque>
<disque>
<interprète nom="feeder"/>
<titre>Comfort In Sound</titre>
</disque>
</disques>
```

On souhaite compter les disques contenus dans ce fichier. Pour ce faire, il suffit d'appliquer le traitement proposer dans la méthode suivante :

compteDisques

```
public static void compteDisques(Document doc) {
  NodeList list=doc.getElementsByTagName("disque");
  System.out.println("Nombre de disques : "+list.getLength());
}
```

Pour afficher les titres des disques, le code se complique très légèrement car il faut d'abord récupérer les éléments `titre`s, puis obtenir le (ou les) enfant(s) de chaque titre qui représente(nt) le texte contenu dans l'élément. On obtient la méthode suivante :

titreDisques

```
public static void titreDisques(Document doc) {
  NodeList list=doc.getElementsByTagName("titre");
  for(int i=0;i<list.getLength();i++) {
    NodeList texte=list.item(i).getChildNodes();
    System.out.print("Disque : ");
    for(int j=0;j<texte.getLength();j++) {
      System.out.print(texte.item(j).getNodeValue());
    }
    System.out.println();
  }
}
```

En utilisant la méthode `normalize` de l'interface `Node`, on peut simplifier le code. En effet, cette méthode a pour effet de concaténer les nœuds de texte adjacents dans l'arbre XML. Dans notre cas, cela implique qu'un élément `titre` ne peut avoir qu'un seul fils, un nœud de texte. On peut alors utiliser la méthode `getFirstChild` de l'interface `Node` pour obtenir ce fils. On obtient la nouvelle méthode suivante :

titreDisques2

```
public static void titreDisques2(Document doc) {
  NodeList list=doc.getElementsByTagName("titre");
  for(int i=0;i<list.getLength();i++) {
    list.item(i).normalize();
    System.out.println("Disque : "+list.item(i).getFirstChild().getNodeValue());
  }
}
```

Si on souhaite maintenant afficher la liste des disques avec le nom du groupe interprète, la situation se complique un peu. On doit en effet trouver un élément `interprète` et un `titre` associés. Comme la méthode `getElementsByTagName` respecte l'ordre du document, on peut traiter en parallèle une `NodeList` pour chaque catégorie d'éléments. L'autre difficulté est de retrouver le `groupe` associé à un identificateur. Pour ce faire, il faut d'abord lire l'attribut `nom` de l'élément `interprète`. Pour ce faire, on utilise le fait que l'objet qui représente un élément implante en fait l'interface `Element` (qui hérite de `Node`). Cette interface propose essentiellement des méthodes de manipulation des attributs, comme par exemple :

`String getAttribute(String name)`
: Renvoie la valeur de l'attribut de nom qualifié `name` et la chaîne vide si l'élément appelant ne possède pas d'attribut de ce nom.

`String getAttributeNS(String URI,String name)`
: Même chose que la méthode précédente, mais avec le nom local `name` et l'espace de nom associé à `URI`.

L'interface `Element` propose aussi deux méthodes `hasAttribute` et `hasAttributeNS` qui renvoient un `boolean` indiquant si l'élément appelant possède ou nom l'attribut recherché (les paramètres sont les mêmes que celles des méthodes `getAttribute` correspondantes). De plus, l'interface `Element` propose les mêmes méthodes `getElements...` que l'interface `Document`. Cette dernière propose de plus une méthode `getElementById` qui permet de retrouver l'élément référencé par un ID (il faut que l'analyseur soit validant pour pouvoir faire cette recherche). On peut alors proposer la méthode suivante :

afficheDisques

```
1  public static void afficheDisques(Document doc) {
2    NodeList titres=doc.getElementsByTagName("titre");
3    NodeList interprètes=doc.getElementsByTagName("interprète");
4    for(int i=0;i<titres.getLength();i++) {
5      Element titre=(Element)(titres.item(i));
6      titre.normalize();
7      System.out.print("Disque : "+titre.getFirstChild().getNodeValue());
8      Element interprète=(Element)(interprètes.item(i));
9      Element groupe=doc.getElementById(interprète.getAttribute("nom"));
10     groupe.normalize();
11     Node nom=groupe.getFirstChild();
12     System.out.println(" interprété par "+nom.getFirstChild().getNodeValue());
13   }
14 }
```

Cette méthode manque singulièrement de contrôle d'erreur, mais elle fonctionne sur un document valide. Notons que l'interface `Node` propose une méthode `getAttributes` qui permet d'obtenir la liste des attributs d'un élément, sous la forme d'une `NamedNodeMap`. Cette technique n'est pas très utile si on connaît la structure du document. `Node` définit aussi une méthode `getNodeType` qui renvoie un `short` précisant le type effectif du nœud. On peut tester cette valeur grâce aux constantes de l'interface `Node` (comme par exemple `Node.ELEMENT_NODE`).

2.3 Exercices

Exercice 2.1

Réaliser grâce à l'API DOM les questions de l'exercice 2.3 de [5], à savoir des manipulations élémentaires de pièces de théâtre au format `play`. Plus précisément, afficher :
- le plan de la pièce
- la liste des personnages
- le nombre total d'actes, de scènes et de vers (`LINE`)

Exercice 2.2

Réaliser grâce à DOM l'exercice 3.3 de [5], à savoir l'évaluation d'une expression ecrite grâce au dialecte inspiré de MathML. Il est vivement conseillé :
- d'écrire une DTD pour éviter les problèmes liés aux blancs ignorables ;
- d'écrire une méthode d'évaluation récursive (afin de traiter facilement le fait qu'un élément `apply` peut contenir des éléments `apply`).

3 Itérations

3.1 L'API DOM *Traversal*

L'API DOM *Traversal* (cf [2]) propose des techniques basées sur le *design pattern* itérateur pour parcourir un arbre DOM de façon plus simple que ce que nous avons vu dans la section précédente. L'idée est de proposer deux interfaces `NodeIterator` et `TreeWalker` qui permettent de parcourir certains nœuds d'un arbre DOM en masquant à l'utilisateur toute la complexité de ce parcours. L'interface `NodeIterator` est la plus simple car elle présente une vue aplatie de l'arbre DOM, au sens où les nœuds sélectionnés sont parcourus dans l'ordre préfixe en profondeur d'abord. On peut avancer ou reculer dans la liste ainsi produite. Pour fixer les idées, voici un exemple de méthode qui affiche le nom de tous les éléments d'un document en ordre préfixe en profondeur :

afficheElements

```
1  public static void afficheElements(Document doc) {
2    DocumentTraversal docT=(DocumentTraversal)doc;
3    NodeIterator iter=docT.createNodeIterator(doc.getDocumentElement(),
4                                              NodeFilter.SHOW_ELEMENT,
5                                              null,true);
6    Node node=iter.nextNode();
7    while (node!=null) {
8      System.out.println(node.getNodeName());
9      node=iter.nextNode();
10   }
11 }
```

L'interface `NodeIterator` est assez simple et propose les méthodes suivantes :

`Node nextNode()`
: Renvoie le nœud courant de l'itérateur et avance celui-ci vers le prochain nœud dans l'ordre préfixe en profondeur. Quand l'itérateur atteint la fin de la liste, il renvoie `null`.

`Node previousNode()`
: Renvoie le nœud précédent le nœud courant de l'itérateur et recule celui-ci vers le nœud précédent (toujours dans l'ordre préfixe en profondeur). Quand l'itérateur atteint le début de la liste, il renvoie `null`.

D'autres méthodes sont disponibles mais sont moins utiles en pratique. On crée un `NodeIterator` en utilisant une usine abstraite, à savoir un objet qui implante l'interface `DocumentTraversal` (c'est en fait le `Document` lui-même). La méthode de création (`createNodeIterator`) est relativement complexe. Son premier paramètre désigne la racine du sous-arbre qu'on souhaite parcourir. Le quatrième est dernier paramètre est assez technique et il est préférable d'utiliser `true` (au moins dans un premier temps). Toute la souplesse des `NodeIterator`s réside dans le deuxième et le troisième paramètre de la méthode de création. Ces paramètres précisent le mode de filtrage.

Le deuxième paramètre est un entier qui contient un masque (obtenu à partir des constantes définies dans l'interface `NodeFilter`). Ce masque permet de sélectionner des grandes catégories de nœuds. Dans l'exemple proposé, l'utilisation de `NodeFilter.SHOW_ELEMENT` assure que le `NodeFilter` ne va parcourir que les nœuds correspondant à des éléments. On peut de cette manière sélectionner les nœuds de texte, les commentaires, etc.

Le troisième paramètre, s'il ne vaut pas `null`, désigne un objet dont la classe implante l'interface `NodeFilter`. Cette interface ne définit qu'une seule méthode :

`short acceptNode(Node n)`
: Renvoie `NodeFilter.FILTER_ACCEPT` si et seulement si le filter accepte le nœud paramètre. La réponse négative est codée par `NodeFilter.FILTER_SKIP` ou `NodeFilter.FILTER_REJECT` (le sens n'est pas le même pour les `TreeWalker`s).

Grâce à l'objet `NodeFilter`, on peut sélectionner comme on le souhaite les nœuds à parcourir, en combinaison avec le masque déjà cité. Cela permet de réaliser une version très évoluée de la méthode `getElementsByTagName`. Si on considère par exemple un document XHTML, on peut définir des cibles internes pour des références croisées. Ces cibles sont précisées par un élément `a` contenant un attribut `name`, à ne pas confondre

avec les éléments `a` contenant un attribut `href` qui sont eux des liens (vers les cibles). On peut parcourir la liste des cibles en utilisant le code `NodeFilter` suivant :

Cibles

```
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.traversal.NodeFilter;
public class Cibles implements NodeFilter {
  public short acceptNode(Node n) {
    if (n.getNodeType()==Node.ELEMENT_NODE) {
      Element e = (Element)n;
      if (! e.getNodeName().equals("a")) {
        return FILTER_SKIP;
      }
      if (e.hasAttribute("name")) {
        return FILTER_ACCEPT;
      }
    }
    return FILTER_SKIP;
  }
}
```

On peut l'utiliser de la façon suivante :

afficheCibles

```
  public static void afficheCibles(Document doc) {
    DocumentTraversal docT=(DocumentTraversal)doc;
    NodeIterator iter=docT.createNodeIterator(doc.getDocumentElement(),
                                              NodeFilter.SHOW_ELEMENT,
                                              new Cibles(),
                                              true);
    Node node=iter.nextNode();
    while (node!=null) {
      Element cible=(Element)node;
      System.out.println(cible.getAttribute("name"));
      node=iter.nextNode();
    }
  }
```

3.2 Exercices

Exercice 3.1

On travaille toujours sur les pièces de théatre au format de la DTD `play` (cf exercice 3.1 de [5]). On souhaire réaliser une recherche ciblée dans un tel document :

1. Ecrire un `NodeFilter` paramétrable qui n'accepte que les éléments `SPEECH` dont le `SPEAKER` est celui donné en paramètre au moment de la construction du filtre.
2. En déduire une méthode qui recherche un mot dans le texte dit par un personnage d'une pièce en utilisant un `NodeIterator`.
3. Proposer une nouvelle version de cette solution qui place le maximum de traitement dans le `NodeFilter`.

Exercice 3.2

On propose une version simplifiée de la DTD XBEL :

xbel-simple.dtd

```
<!ELEMENT xbel (title?, (bookmark|folder|alias)*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT bookmark (title?)>
```

```
<!ATTLIST bookmark href CDATA #REQUIRED
                   id ID #IMPLIED>
<!ELEMENT folder (title?, (bookmark|folder|alias)*)>
<!ATTLIST folder id ID #IMPLIED>
<!ELEMENT alias EMPTY>
<!ATTLIST alias ref IDREF #REQUIRED>
```

Cette dtd permet de représenter des listes de signets (*bookmark*s) correspondant à des sites internets. Ces listes sont organisées grâce à des dossiers (*folder*s). De plus, pour permettre une vision thématique par dossier sans recopier les signets (ou les dossiers), on peut utiliser un système de références croisées par l'intermédiaire des `alias`. Voici un exemple de fichier au format XBEL :

bookmarks-short.xml

```
<?xml version="1.0"?>
<!DOCTYPE xbel SYSTEM "xbel-simple.dtd">
<xbel>
<folder><title>XML</title>
<folder><title>Specs</title>
<bookmark id="xml-rec" href="http://www.w3.org/TR/REC-xml">
<title>Extensible Markup Language (XML)</title>
</bookmark>
<bookmark id="xpath" href="http://www.w3.org/TR/xpath">
<title>XML Path Language (XPath)</title>
</bookmark>
<bookmark id="xslt" href="http://www.w3.org/TR/xslt">
<title>XSL Transformations (XSLT)</title>
</bookmark>
</folder>
<folder><title>Implementations</title>
<bookmark id="xerces-j" href="http://xml.apache.org/xerces-j/index.html">
<title>Xerces Java Parser</title>
</bookmark>
<alias ref="xalan"/>
<alias ref="cocoon"/>
</folder>
</folder>
<folder>
<title>HTML</title>
<folder><title>Separating content and style</title>
<alias ref="xslt"/>
<bookmark id="xalan" href="http://xml.apache.org/xalan/index.html">
<title>Xalan</title>
</bookmark>
<bookmark id="cocoon" href="http://xml.apache.org/cocoon/index.html">
<title>Cocoon</title>
</bookmark>
</folder>
</folder>
</xbel>
```

Questions :

1. Ecrire un programme DOM qui affiche les titres des `bookmark`s en utilisant un `NodeIterator`, avec le plus de travail possible réalisé dans le `NodeFilter`.
2. Améliorer le programme de la question précédente afin d'afficher aussi les titres des `bookmark`s induit par les `alias`. Attention, un `alias` peut désigner un `folder`, ce qui complique la tâche.

3. Il n'est pas possible dans la DTD de se prémunir contre les cycles. En effet, le document suivant est valide :

bad.xml

```
<?xml version="1.0"?>
<!DOCTYPE xbel SYSTEM "xbel-simple.dtd">
<xbel>
<folder id="a">
<alias ref="b"/>
</folder>
<folder id="b">
<alias ref="a"/>
</folder>
</xbel>
```

   Ecrire un programme DOM qui indique si un fichier XBEL est cohérent, c'est-à-dire sans cycle.

4. Lire la documentation de DOM Level 2 ([3]) pour comprendre comment modifier un arbre DOM, puis écrire un programme qui lit un fichier XBEL, remplace tous les `alias` par les `bookmark`s et `folder`s correspondant, et sauvegarde le résultat dans un nouveau fichier XML. Pour la sauvegarde on pourra utiliser la méthode suivanté, spécifique à Xerces :

save

```
public static void save(Document doc,String file) {
  OutputFormat format=new OutputFormat(doc,"ISO-8859-1",true);
  try {
    OutputStream out=new FileOutputStream(file);
    XMLSerializer printer=new XMLSerializer(out,format);
    printer.serialize(doc);
  } catch(IOException ioe) {
    ioe.printStackTrace();
  }
}
```

Références

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[2] Joe Kesselman, Jonathan Robie, Mike Champion, Peter Sharpe, Vidur Apparao, and Lauren Wood, editors. *Document Object Model (DOM) Level 2 Traversal and Range Specification (Version 1.0)*. W3C Recommendation. W3C, 13 November 2000. `http://www.w3.org/TR/DOM-Level-2-Traversal-Range`.

[3] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne, editors. *Document Object Model (DOM) Level 2 Core Specification (Version 1.0)*. W3C Recommendation. W3C, 13 November 2000. `http://www.w3.org/TR/DOM-Level-2-Core`.

[4] Fabrice Rossi. Dtd et schémas. Recueil d'exercices, Université Paris-IX Dauphine, 2002. `http://apiacoa.org/teaching/xml/exercices/dtd-et-schemas.pdf`.

[5] Fabrice Rossi. Simple api for xml. Recueil d'exercices, Université Paris-IX Dauphine, 2002. `http://apiacoa.org/teaching/xml/exercices/sax.pdf`.