JDOM

Manipulation de XML avec JDOM et Java

JDOM

C’est quoi ?

- JDOM est une API du langage Java
- Permet de manipuler des donnés XML plus simplement qu'avec les API classiques
- Son utilisation est pratique pour tout développeur Java et repose sur les API XML de Sun
- Les binaires peuvent être téléchargées ici : http://www.jdom.org/dist/binary/, sinon regarder le répertoire jdom

JDOM

Origines

- SAX
  - SAX est l'acronyme de *Simple API for XML*
  - Ce type de parseur utilise des événements pour piloter le traitement d'un fichier XML
  - Un objet doit implémenter des méthodes particulières définies dans une interface de l'API pour fournir les traitements à réaliser : selon les événements, le parseur appelle ces méthodes
  - JDOM utilise des collections SAX pour parser les fichiers XML

JDOM

- Description de DOM
  - Acronyme de *Document Object Model*
  - Rôle d'après le W3C :
    - fournir une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et d'en permettre la manipulation (parcours, recherche et mise à jour)
  - DOM est défini pour être indépendant du langage dans lequel il sera implémenté (PHP, Java, etc.)
  - DOM n'est qu'une spécification qui, pour être utilisée, doit être implémentée par un éditeur tiers. **DOM n'est donc pas spécifique à Java**
  - Le parseur DOM pour JAVA le plus répandu est Xerces

JDOM

- Description de JDOM
  - JDOM utilise DOM pour manipuler les éléments d'un Document Object Model spécifique (créé grâce à un constructeur basé sur SAX)
  - JDOM permet donc de construire des documents, de naviguer dans leur structure, d'ajouter, de modifier, ou de supprimer leur contenu
- **Mais qu'est-ce que JDOM nous apporte de plus ?**
  - **La simplicité !**
  - Il est en vérité très laborieux de développer des applications complexes autour de XML avec DOM, qui rappelons le, n'a pas été développé spécifiquement pour Java

Créer un fichier XML avec JDOM

Téléchargement et installation de l'API JDOM

- Il vous faut dans un premier temps télécharger la dernière version de JDOM disponible à cette adresse : http://www.jdom.org/dist/binary/
- Il suffit ensuite de rendre accessible le fichier */build/jdom.jar*, en le plaçant dans votre classpath
- Sinon, si vous développez sous Eclipse
  - Copier les .jar : se mettre sur le projet, cliquer droit, properties, add jar, puis ajouter les .jar dans build et bin qui sont dans le répertoire jdom

Créer un fichier XML avec JDOM

Créer une arborescence simple

- Il suffit de construire chaque élément puis de les ajouter les uns aux autres de façon logique
- Un noeud est une instance de *org.jdom.Element*
- Nous commençons donc par créer une classe JDOM1 qui va se charger de créer l'arborescence suivante :
  - Fichier XML

```
<personnes>
  <etudiant classe="P2">
    <nom>CynO</nom>
  </etudiant>
<personnes>
```

```
import java.io.*;
import org.jdom.*;
import org.jdom.output.*;

public class JDOM1
{
  //Nous allons commencer notre arborescence en créant la racine XML
  //qui sera ici "personnes"
  static Element racine = new Element("personnes");

  //On crée un nouveau Document JDOM basé sur la racine que l'on vient de
    créer
  static org.jdom.Document document = new Document(racine);

  public static void main(String[] args)
  {
    //On crée un nouvel Element etudiant et on l'ajoute en tant qu'Element de
    racine
    Element etudiant = new Element("etudiant");
    racine.addContent(etudiant);
```

```
  //On crée un nouvel Attribut classe et on l'ajoute à etudiant
  //grâce à la méthode setAttribute
   Attribute classe = new Attribute("classe","P2");
   etudiant.setAttribute(classe);

  //On crée un nouvel Element nom, on lui assigne du texte
   //et on l'ajoute en tant qu'Element de etudiant
   Element nom = new Element("nom");
   nom.setText("CynO");
   etudiant.addContent(nom);

  affiche(); //définie plus loin
   enregistre("Exercice1.xml"); //définie plus loin
}
```

Créer un fichier XML avec JDOM

Afficher et enregistrer son fichier XML

- Nous allons afficher puis enregistrer notre arborescence
- Nous allons utiliser une unique classe pour ces deux flux de sortie : *org.jdom.output.XMLOutputter*, qui prend en argument un *org.jdom.output.Format*
- En plus des trois formats par défaut (PrettyFormat, CompactFormat et RawFormat), la classe *Format* contient une panoplie de méthodes pour affiner votre sérialisation

```
  static void affiche()
  {
    try
    {
      //On utilise ici un affichage classique avec getPrettyFormat()
      XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
      sortie.output(document, System.out);
    }
    catch (java.io.IOException e){}
  }

static void enregistre(String fichier)
  {
    try
    {
      //On utilise ici un affichage classique avec getPrettyFormat()
      XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
      //Remarquez qu'il suffit simplement de créer une instance de FileOutputStream
      //avec en argument le nom du fichier pour effectuer la sérialisation.
      sortie.output(document, new FileOutputStream(fichier));
    }
    catch (java.io.IOException e){}
  }
}
```

Parcourir un fichier XML

- Parser un fichier XML
  - Cela revient à transformer un fichier XML en une arborescence JDOM
  - Nous utiliserons pour cela le constructeur SAXBuilder
  - Créez tout d'abord le fichier suivant dans le répertoire contenant votre future classe JDOM2 :

```
<?xml version="1.0" encoding="UTF-8"?>
<personnes>
  <etudiant classe="P2">
    <nom>CynO</nom>
    <prenoms>
      <prenom>Nicolas</prenom>
      <prenom>Laurent</prenom>
    </prenoms>
  </etudiant>
  <etudiant classe="P1">
    <nom>Superwoman</nom>
  </etudiant>
  <etudiant classe="P1">
    <nom>Don Corleone</nom>
  </etudiant>
</personnes>
```

Parcourir un fichier XML

- Notre objectif ici est d'afficher dans un premier temps le nom de tous les élèves
- Nous allons créer pour cela une nouvelle classe: JDOM2

```
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.filter.*;
import java.util.List;
import java.util.Iterator;
```

```
public class JDOM2
{
   static org.jdom.Document document;
   static Element racine;
   public static void main(String[] args)

   {
      //On crée une instance de SAXBuilder
      SAXBuilder sxb = new SAXBuilder();
      try
      {
         //On crée un nouveau document JDOM avec en argument le fichier XML
         //Le parsing est terminé ;)
         document = sxb.build(new File("Exercice2.xml"));
      }
      catch(Exception e){}

      //On initialise un nouvel élément racine avec l'élément racine du document.
      racine = document.getRootElement();

      //Méthode définie plus loin
      afficheALL();
   }
```

```
    static void afficheALL()
    {
      //On crée une List contenant tous les noeuds "etudiant" de l'Element racine
      List listEtudiants = racine.getChildren("etudiant");

      //On crée un Iterator sur notre liste
      Iterator i = listEtudiants.iterator();
      while(i.hasNext())
      {
        //On recrée l'Element courant à chaque tour de boucle afin de
        //pouvoir utiliser les méthodes propres aux Element comme :
        //selectionner un noeud fils, modifier du texte, etc.
        Element courant = (Element)i.next();
        //On affiche le nom de l'élément courant
        System.out.println(courant.getChild("nom").getText());
      }
    }
}
```

Parcourir un fichier XML

Filtrer les éléments

- Notre nouvel objectif est d'afficher la classe des étudiants dont le prénom est Laurent et le nom est CynO
- Les seuls filtres que nous ayons faits pour le moment étaient directement implémentés dans les méthodes que nous utilisions
  - List listEtudiants = racine.getChildren("etudiant") nous a permis de filtrer les sous éléments de racine selon leur nom
- Vous aurez remarqué que de toute façon nous n'avions que des étudiants, le problème ne se posait donc pas
- Les filtres permettent des sélections d'éléments selon plusieurs critères

Parcourir un fichier XML

Filtrer les éléments

- Nous allons donc créer un filtre qui permettra de ne prendre en compte que les Elements qui possèdent :
  1. Un sous élément *nom* qui doit avoir pour valeur "CynO"
  2. Un sous élément *prenoms* qui doit posséder au moins un sous élément *prenom* dont la valeur est "Laurent"
- Une fois le filtre créé nous pourrons récupérer une liste contenant les éléments répondant à ces critères

```
//Ajouter cette méthode à la classe JDOM2
//Remplacer la ligne afficheALL(); par afficheFiltre();
static void afficheFiltre()
{
  //On crée un nouveau filtre
  Filter filtre = new Filter()
  {
    //On définit les propriétés du filtre à l'aide de la méthode matches
    public boolean matches(Object ob)
    {
      //1 ère vérification : on vérifie que les objets
      //qui seront filtrés sont bien des Elements
      if(!(ob instanceof Element)){return false;}

      //On crée alors un Element sur lequel on va faire les
      //vérifications suivantes
      Element element = (Element)ob;

      //On crée deux variables qui vont nous permettre de vérifier
      //les conditions de nom et de prenom
      int verifNom = 0;
      int verifPrenom = 0;
```

```
//2 ème vérification: on vérifie que le nom est bien "CynO"
if(element.getChild("nom").getTextTrim().equals("CynO"))
{
   verifNom = 1;
}
//3 ème vérification: on vérifie que CynO possède un prenom "Laurent"
//On commence par vérifier que la personne possède un prenom,
//en effet notre fichier XML possède des étudiants sans prénom !
Element prenoms = element.getChild("prenoms");
if(prenoms == null){return false;}

//On constitue une list avec tous les prenom
List listprenom = prenoms.getChildren("prenom");
```

```
    //On effectue la vérification en parcourant notre liste de prenom
        Iterator i = listprenom.iterator();
        while(i.hasNext())
        {
            Element courant = (Element)i.next();
            if(courant.getText().equals("Laurent"))
            {
                verifPrenom = 1;
            }
        }

        //Si nos conditions sont remplies on retourne true, false sinon
        if(verifNom == 1 && verifPrenom == 1)
        {
            return true;
        }
        return false;
    }
}; //Fin du filtre
```

```
    //getContent va utiliser notre filtre pour créer une liste d'étudiants
     //répondant à nos critères
    List resultat = racine.getContent(filtre);
    //On affiche enfin l'attribut classe de tous les éléments de notre
     //list
    Iterator i = resultat.iterator();
    while(i.hasNext())
    {
       Element courant = (Element)i.next();
       System.out.println(courant.getAttributeValue("classe"));
    }
 }
```

A l'exécution vous devriez voir s'afficher P2 à votre écran

Parcourir un fichier XML

Conclusion sur les filtres

- La puissance de cet outil réside dans sa capacité à être utilisé à tout moment par n'importe quel Element de votre arborescence
- Dans notre exemple, nous nous sommes servi de notre filtre JDOM comme d'un moteur de recherche
- Et il est tout à fait envisageable de créer des filtres dynamiques selon vos besoins
- Pour en savoir plus sur la classe Filter je vous invite à vous rendre ici

  http://www.jdom.org/docs/apidocs/org/jdom/filter/package-summary.html

Modifier une arborescence JDOM

- Modifier des Éléments

Nom	Arguments des surcharges	Description
addContent	Collection, String ou Content, c'est à dire un Element ou quoi que se soit qui peut être contenu par un noeud.	Ajoute le contenu de l'argument à la fin du contenu d'un Element. On peut spécifier un index pour l'inserer à la position voulu.
clone		Retourne un clone parfait de l'Element.
cloneContent		Comme son nom l'indique on ne copie que le contenu.
removeAttribute	Attribut ou nom de l'attribut (String)	Supprime un attribut d'un Element
removeChild	nom du noeud enfant (String)	Supprime le premier enfant portant ce nom.
removeChildren	nom des noeuds enfants (String)	Supprime tous les enfants ayant ce nom.
removeContent	Content, Index ou Filtre	Supprime l'intégralité d'un noeud donné en argument ou par sa position. removeContent accept aussi les filtres, tout comme getContent vu précédement.
setAttribute	Attribut ou nom de l'attribut et sa valeur (String, String)	Cette méthode permet à la fois de créer un attribut et d'en modifier sa valeur.
setContent	Content	Remplace le contenu d'un Element. On peut spécifier un index si l'on ne veut pas tout remplacer.
setName	Nouveau nom de l'Element (String)	Change le nom de l'Element.
setText	Nouveau Text à inserer (String)	Change le text contenu par l'Element. <element>TEXT</element>
toString		Retourne une représentation de l'Element sous forme de chaine.

Modifier une arborescence JDOM

- Modifier des Éléments : Exemple
  - Nous allons modifier le contenu de notre fichier Exemple2.xml en supprimant tous les Element prenoms de notre arborescence

```
//Créer une nouvelle class JDOM3
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
import java.util.List;
import java.util.Iterator;

public class JDOM3
{
   static org.jdom.Document document;
   static Element racine;

   public static void main(String[] args)
   {
      try
      {
         lireFichier("Exercice2.xml");
         supprElement("prenoms");
         enregistreFichier("Exercice2.xml");
       }
       catch(Exception e){}
   }
```

```
//On parse le fichier et on initialise la racine de notre arborescence
static void lireFichier(String fichier) throws Exception
{
   SAXBuilder sxb = new SAXBuilder();
   document = sxb.build(new File(fichier));
   racine = document.getRootElement();
}

//On fait des modifications sur un Element
static void supprElement(String element)
{
   //Dans un premier temps on liste tous les étudiants
   List listEtudiant = racine.getChildren("etudiant");
   Iterator i = listEtudiant.iterator();
   //On parcourt la liste grâce à un iterator
   while(i.hasNext())
   {
      Element courant = (Element)i.next();
      //Si l'etudiant possède l'Element en question on applique
      //les modifications.
      if(courant.getChild(element)!=null)
      {
```

```
//On supprime l'Element en question
            courant.removeChild(element);
            //On renomme l'Element père sachant qu'une balise XML n'accepte
            //ni les espaces ni les caractères spéciaux
            //"etudiant modifié" devient "etudiant_modifie"
            courant.setName("etudiant_modifie");
        }
      }
   }

   //On enregistre notre nouvelle arborescence dans le fichier
   //d'origine dans un format classique.
   static void enregistreFichier(String fichier) throws Exception
   {
        XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
        sortie.output(document, new FileOutputStream(fichier));
   }
}
```

A l’exécution, on obtient ceci :

```
<?xml version="1.0" encoding="UTF-8"?>
<personnes>
  <etudiant_modifie classe="P2">
    <nom>CynO</nom>
  </etudiant_modifie>
  <etudiant classe="P1">
    <nom>Superwoman</nom>
  </etudiant>
  <etudiant classe="P1">
    <nom>Don Corleone</nom>
  </etudiant>
</personnes>
```

Modifier une arborescence JDOM

- Passer de DOM à JDOM et l'inverse
  - Il vous arrivera parfois de devoir travailler sur un document DOM
  - Nous allons voir comment transformer un document DOM en un document JDOM et vis versa
  - Voici une petite méthode qui reçoit en argument un document DOM et retourne un document JDOM

```
//Pour être compilé, cette fonction a besoin de l'importation suivante
//qui contient la classe DOMBuilder
import org.jdom.input.*;

org.jdom.Document DOMtoJDOM(org.w3c.dom.Document
   documentDOM) throws Exception
{
 //On utilise la classe DOMBuilder pour cette transformation
 DOMBuilder builder = new DOMBuilder();
 org.jdom.Document documentJDOM = builder.build(documentDOM);
 return documentJDOM;
}
```

Modifier une arborescence JDOM

- Et maintenant, voici la fonction inverse qui reçoit en argument un document JDOM et qui retourne un document DOM
- Vous remarquerez la similitude avec la fonction précédente

```
//Pour être compilée, cette fonction a besoin de l'importation suivante
//qui contient la classe DOMOutputter
import org.jdom.output.*;


org.w3c.dom.Document DOMtoJDOM(org.jdom.Document
   documentJDOM) throws Exception
{
 //On utilise la classe DOMOutputter pour cette transformation
 DOMOutputter domOutputter = new DOMOutputter();
 org.w3c.dom.Document documentDOM =
   domOutputter.output(documentJDOM);
 return documentDOM;
}
```

JDOM et XSLT

Comment ça marche ?

- Grâce à l'API JAXP et TraX il est très facile de faire des transformation XSLT sur un document JDOM
- Dans l'exemple suivant nous allons créer une méthode qui prend en entrée un document JDOM et le nom d'un fichier XSL et qui crée en sortie un fichier XML transformé

```
//Pour être compilé cette fonction à besoin des importations suivantes
import java.io.*;
//JDOM
import org.jdom.transform.*;
import org.jdom.output.*;
//TrAX
import javax.xml.transform.*;
import javax.xml.transform.stream.StreamSource;

void outputXSLT(org.jdom.Document documentJDOMEntree,String fichierXSL)
{
  //Document JDOMResult, résultat de la transformation TraX
  JDOMResult documentJDOMSortie = new JDOMResult();
  //Document JDOM après transformation
  org.jdom.Document resultat = null;

  try
  {
    //On définit un transformer avec la source XSL
    //qui va permettre la transformation
    TransformerFactory factory = TransformerFactory.newInstance();
    Transformer transformer = factory.newTransformer(new
   StreamSource(fichierXSL));
```

```
        //On transforme le document JDOMEntree grâce à notre transformer.
        //La méthode transform() prend en argument le document d'entrée
        //associé au transformer et un document JDOMResult, résultat de
        //la transformation TraX
        transformer.transform(new
        org.jdom.transform.JDOMSource(documentJDOMEntree),
        documentJDOMSortie);

        //Pour récupérer le document JDOM issu de cette transformation
        //il faut utiliser la méthode getDocument()
        resultat = documentJDOMSortie.getDocument();

        //On crée un fichier xml corespondant au résultat
        XMLOutputter outputter = new
        XMLOutputter(Format.getPrettyFormat());
        outputter.output(resultat, new FileOutputStream("resultat.xml"));
    }
    catch(Exception e){}
}
```