جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Documents structurés

Cours 7

Nassim ZELLAL

DOM - 1

Programmation avec DOM (Document Object Model)

- L'interface de programmation ou API (Application Programming Interface) DOM permet :
  - d'exploiter des documents structurés HTML ou XML;
  - de manipuler et transformer une page WEB dynamiquement en modifiant l'arbre DOM;
  - d'interagir dynamiquement avec l'information présentée;
  - De mettre à jour dynamiquement une page WEB;
- L'API DOM repose sur une représentation hiérarchique d'un document. Elle permet la représentation en mémoire d'un arbre et de ses nœuds.
- DOM est en général exploité dès lors qu'il s'agit d'opérer des transformations sur l'ensemble du document.

Programmation avec DOM (Document Object Model)

- Le DOM est donc une API qui s'utilise avec les documents XML et HTML et qui va nous permettre, via le langage JAVA, d'accéder au code source XML ou HTML d'un document (page WEB).
- C'est grâce au DOM que nous allons pouvoir modifier des éléments HTML et/ou XML, qui représentent le contenu d'une page WEB.
- **Nous allons utiliser le parseur JAXP (JAVA API for XML Processing)**. Il existe aussi le parseur Xerces de la fondation Apache.

Programmation avec DOM (Document Object Model)

- Le modèle DOM représente une spécification, qui puise ses origines dans le consortium **W3C.** Cette API bénéficie donc d'un niveau de **respectabilité** égal à la spécification XML elle-même.

- Le modèle DOM est non seulement une **spécification multi-plateformes**, mais aussi **multi-langages** : JAVA, JAVASCRIPT, PHP, PYTHON, C++, PERL, etc.

Lecture

- Nous allons lire le document XML « read.xml » se trouvant sur le bureau avec l'API DOM et le parseur JAXP (Java API for XML Processing ou Pasring dans certains documents - SUN) en JAVA sur Eclipse.

- Pour travailler sur Eclipse, placez votre fichier à lire « read.xml » (voir slide 8) à la racine du répertoire de votre projet DOM1 du Workspace (C:\workspace\DOM\read.xml), en utilisant le **tableau spécial args[]**. Sinon précisez le chemin de votre fichier XML dans votre code JAVA (voir le slide suivant).

Lecture - JAVA

Voir « Read.java »

```
package DOM;
import org.w3c.dom.Attr;
public class Read{
    public static void main( String [] args ) throws Exception{
        //Récupération de l'argument représentant le document à parser (analyser)
        //String xmlFile = "C:/XXX/XXX/Desktop/read.xml";
        String xmlFile= args[0];
```

On peut récupérer aussi l'argument « read.xml » grâce au tableau args[].

Document à lire - « read.xml »

```
<?xml version="1.0"?>
<liste>
    <eleve nom="E1" note="12">L2 ACAD</eleve>
    <eleve nom="E2" note="18">L3 ACAD</eleve>
    <eleve nom="E3" note="07">L2 ACAD</eleve>
</liste>
```

Voir « read.xml »

Il n'est pas nécessaire de préciser l'encodage dans la déclaration XML, car l'encodage par défaut en XML est UTF-8/UTF-16.

Lecture

- Le code s'appelle « **Read.java** » **(package DOM du projet DOM1)**. Ce code est utilisé pour lire le document XML « **read.xml** ».
- Pour ce faire, soit on utilise la méthode **getElementsByTagName(),** soit on emploie la méthode **getChildNodes()**.

- Si on utilise **getChildNodes()**, il faudra penser à filtrer les objets textes (**nœuds de texte ➔ #text**) **parasites** générés par la mise en forme (indentation) du fichier tels que des sauts de ligne, des espaces ou des tabulations, afin de les supprimer.
- **Mais avant cela, il faut créer un parseur (analyseur) XML.**

Création d'un parseur XML

- **import javax.xml.parsers.DocumentBuilderFactory;**
- **import javax.xml.parsers.DocumentBuilder;**
- //Récupération de l'argument représentant le document à **parser (analyser**).
- **String xmlFile= args[0];**
- //Création d'une instance de la classe « DocumentBuilderFactory » (ici « **factory** »). La classe « DocumentBuilderFactory » définit une API factory, permettant d'obtenir un parseur, qui produit un **arbre d'objets DOM**, à partir du document XML.
- **DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance();***
- //Création d'une instance de la classe « DocumentBuilder » (ici « **parseur** »), permettant d'obtenir, par la suite, une instance de Document DOM, à partir du document XML.
- **DocumentBuilder parseur = factory.newDocumentBuilder();**

#----------Ou----------

- **DocumentBuilder parseur = DocumentBuilderFactory.newInstance().newDocumentBuilder();**

Lecture

- Après avoir créé notre parseur, nous aurons besoin d'importer les **interfaces** suivantes:
- **import org.w3c.dom.Document;**
- **import org.w3c.dom.Element;**
- **----------**
- On utilise la méthode **parse()**, qui s'applique sur l'instance de la classe « DocumentBuilder » (voir slide précédent). Cette méthode prend en entrée un URI comme XML et retourne un objet **Document** (objet Document DOM), qui est un nœud de document (nœud racine).
- //Création d'un objet « Document » (ou objet Document DOM), en appliquant la méthode "parse()" sur notre "parseur". Cette méthode permet d'analyser le contenu de notre document XML.
- **Document document = parseur.parse(xmlFile);**
- //Récupération du **nœud d'élément racine ou l'élément racine** (en anglais : document element) « liste » du document XML « read.xml ».
- **Element liste = document.getDocumentElement();**
- **System.out.println("Élément racine => "+liste.getNodeName());**

Lecture des nœuds d'élément « eleve » - getElementsByTagName()

//interface « NodeList »

- **import org.w3c.dom.NodeList;**

----------Méthode 1 : getElementsByTagName()----------

- //Récupération des nœuds d'élément <eleve>. Ces nœuds sont placés dans une collection de nœuds de type « NodeList ». Ces nœuds peuvent être récupérés via leur indice à l'aide de la méthode item(indice).
- NodeList eleves =liste.**getElementsByTagName("eleve")**;
- **for (int i = 0;i<eleves.getLength();i++){**
  - **……..**
- **}**

Lecture des nœuds d'élément « eleve » - getElementsByTagName()

- //Récupération du nœud de la collection dont l'indice est passé en paramètre à la méthode "item()". Ensuite, récupération du nom de ce nœud.
- System.**out.println("Noeud XML => "+eleves.item(i).getNodeName());**
- //Retourne le nom du nœud du premier fils de l'indice de l'item de la collection.
- System.**out.println("Premier fils d'eleve => "+eleves.item(i).getFirstChild().getNodeName());**
- //Retourne la valeur du nœud du premier fils de l'indice de l'item de la collection.
- System.**out.println("Valeur du premier fils d'eleve => "+eleves.item(i).getFirstChild().getNodeValue());**
- //Il est possible aussi d'utiliser la méthode getTextContent(), qui retourne le texte du nœud et celui de ses descendants.

- //System.out.println("Valeur du premier fils d'eleve => "+eleves.item(i).getTextContent());

Lecture des nœuds d'élément « eleve » - getElementsByTagName()

```
//*************************Méthode 1 : getElementsByTagName()*************************************************//
//Récupération des nœuds d'élément <eleve>. Ces nœuds sont placés dans une collection de nœuds de type « NodeList ».
//Ces nœuds peuvent être récupérés via leur indice à l'aide de la méthode "item(indice)".
NodeList eleves = liste.getElementsByTagName("eleve");
for (int i = 0;i<eleves.getLength();i++){
    //Récupération du nœud de la collection dont l'indice est passé en paramètre à la méthode "item()".
    //Ensuite, récupération du nom de ce nœud XML.
    System.out.println("Nom du noeud XML => "+eleves.item(i).getNodeName());
    /* ou
    Node un_eleve = eleves.item(i);
    System.out.println("Noeud XML => "+un_eleve.getNodeName());
     */
    //Retourne le nom du nœud du premier fils de l'indice de l'item de la collection.
    System.out.println("Nom du nœud du premier fils d'eleve => "+eleves.item(i).getFirstChild().getNodeName());
    //Retourne la valeur du noeud du premier fils de l'indice de l'item de la collection.
    System.out.println("Valeur du premier fils d'eleve => "+eleves.item(i).getFirstChild().getNodeValue());
    //Il est possible aussi d'utiliser la méthode getTextContent(), qui retourne le texte du nœud et celui
    //de ses descendants.
    //System.out.println("Valeur du premier fils d'eleve => "+eleves.item(i).getTextContent());
```

Voir « Read.java »

Lecture des nœuds d'élément « eleve » - getElementsByTagName()

- 1) Nœud d'élément XML => eleve
- 2) Premier fils d'eleve => #text (objet texte - nœud de texte)
- 3) Valeur du premier fils d'eleve => L2 ACAD

- 1) Le nom du nœud d'élément est « eleve ».
- 2) Le nom du nœud du premier fils d'eleve est l'objet « #text ».
- 3) La valeur du nœud du premier fils d'eleve est la chaîne de caractères « L2 ACAD ».

Lecture des nœuds d'élément « eleve » - getElementsByTagName()

```
Nom du nœud XML => eleve
Nom du nœud du premier fils d'eleve => #text
Valeur du premier fils d'eleve => L2 ACAD
Nom du nœud XML => eleve
Nom du nœud du premier fils d'eleve => #text
Valeur du premier fils d'eleve => L3 ACAD
Nom du nœud XML => eleve
Nom du nœud du premier fils d'eleve => #text
Valeur du premier fils d'eleve => L2 ACAD
```

Lecture d'un attribut - getAttribute()

- **import org.w3c.dom.Element;**

- //Coercion (conversion forcée -cast-), car on ne peut pas récupérer la valeur de l'attribut à partir d'une « NodeList ».
- On convertir donc notre « NodeList » en « Element ».
- **Element** l_eleve = **(Element)** eleves.item(i);
- //On affiche l'attribut en utilisant la méthode « getAttribute() », en précisant l'attribut à afficher.
- System.out.println("Nom de l'étudiant => "+ l_eleve.**getAttribute("nom")**);

Lecture d’un attribut - getAttribute()

```
//*****************************Lecture d’un attribut - getAttribute()*********************************************
//Coercion (conversion forcée -cast-, car on ne peut pas récupérer la valeur de l'attribut à partir d'une "NodeList".
//On convertit donc notre "NodeList" en "Element".
Element l_eleve = (Element) eleves.item(i);
//On affiche l’attribut en utilisant la méthode getAttribute(), en précisant l'attribut à afficher.
System.out.println("Nom de l'étudiant => "+ l_eleve.getAttribute("nom"));
```

Voir « Read.java »

Lecture d'un attribut - getAttribute()

```
Noeud XML => eleve
Premier fils d'eleve => #text
Valeur du premier fils d'eleve => L2 ACAD
Nom de l'étudiant => E1
Noeud XML => eleve
Premier fils d'eleve => #text
Valeur du premier fils d'eleve => L3 ACAD
Nom de l'étudiant => E2
Noeud XML => eleve
Premier fils d'eleve => #text
Valeur du premier fils d'eleve => L2 ACAD
Nom de l'étudiant => E3
```

Lecture d'un attribut - getAttributes()

- Nous aurons besoin d'importer les **interfaces** suivantes:
- **import org.w3c.dom.Attr;**
- **import org.w3c.dom.NamedNodeMap;**
- **----------**
- //On place les attributs dans un objet de type « NamedNodeMap ».
- **NamedNodeMap** atts = ele.**getAttributes()**;
- //Ou NamedNodeMap atts = eleves.item(i).getAttributes();
- for(int e = 0; e < atts.getLength(); e++){
- //Cast (Node ==> Attr)
- Attr attr = **(Attr)** atts.item(e);
- //String nom = atts.item(e).getNodeName();
- //String val = atts.item(e).getNodeValue();
- String nom = attr.**getName()**;
- String val = attr.**getValue()**;
- System.*out.println(nom);*
- System.*out.println(val);*}

Lecture d'un attribut - getAttributes()

```
//***************Lecture d'un attribut - getAttributes()***********
    //On place les attributs dans un objet de type NamedNodeMap.
    NamedNodeMap atts = eleves.item(i).getAttributes();
    //Ou NamedNodeMap atts = eleves.item(i).getAttributes();
    for( int e = 0; e < atts.getLength(); e++ ){
        //Cast (NamedNodeMap ==> Attr)
        Attr attr =  (Attr) atts.item(e);
        //String nom = atts.item(e).getNodeName();
        //String val = atts.item(e).getNodeValue();
        String nom = attr.getName();
        String val = attr.getValue();
        System.out.println(nom);
        System.out.println(val);
    }
}
```

Voir « Read.java »

Lecture d'un attribut - getAttributes()

```
Noeud XML => eleve
Premier fils d'eleve => #text
Valeur du premier fils d'eleve => L2 ACAD
Nom de l'étudiant => E1
nom
E1
note
12
Noeud XML => eleve
Premier fils d'eleve => #text
Valeur du premier fils d'eleve => L3 ACAD
Nom de l'étudiant => E2
nom
E2
note
18
Noeud XML => eleve
Premier fils d'eleve => #text
Valeur du premier fils d'eleve => L2 ACAD
Nom de l'étudiant => E3
nom
E3
note
07
```

Lecture d’un attribut - getAttributes()

On peut aussi faire plus court, en modifiant « Read.java ».

```
System.out.println(((Attr) eleves.item(i).getAttributes().item(e)).getName());
System.out.println(((Attr) eleves.item(i).getAttributes().item(e)).getValue());
System.out.println(eleves.item(i).getAttributes().item(e).getNodeName());
System.out.println(eleves.item(i).getAttributes().item(e).getNodeValue());
```

Lecture d’un attribut - getAttributes()

```
nom
E1
nom
E1
note
12
note
12
nom
E2
nom
E2
note
18
note
18
nom
E3
nom
E3
note
07
note
07
```

Lecture des nœuds « eleve » - getChildNodes()

- **import org.w3c.dom.NodeList;**
- **import org.w3c.dom.Node;**

- Méthode 2 : getChildNodes()

- **NodeList eleves_bis = liste.getChildNodes();**

Document à lire - « read.xml »

```
<?xml version="1.0"?>
<liste>
    <eleve nom="E1" note="12">L2 ACAD</eleve>
    <eleve nom="E2" note="18">L3 ACAD</eleve>
    <eleve nom="E3" note="07">L2 ACAD</eleve>
</liste>
```

Voir « read.xml »

Il n'est pas nécessaire de préciser l'encodage dans la déclaration XML, car l'encodage par défaut en XML est UTF-8/UTF-16.

Lecture des nœuds « eleve » - getChildNodes()

- **NodeList** eleves_bis = liste.**getChildNodes()**;
- for (int o = 0;o<eleves_bis.getLength();o++){
- //On affiche le nom de tous les nœuds fils de l'élément racine <liste>
- System.out.println("Nom du nœud "+eleves_bis.item(o).getNodeName());
- }

Lecture des nœuds « eleve » - getChildNodes()

Voir « Read.java »

```
**********Méthode 2 : getChildNodes()**********
NodeList eleves_bis =  liste.getChildNodes();
for (int o = 0;o<eleves_bis.getLength();o++){
    //On affiche le nom de tous les nœuds fils de l'élément racine <liste>
    System.out.println("Nom du nœud "+eleves_bis.item(o).getNodeName());
}
```

Lecture des nœuds « eleve » - getChildNodes()

```
Nom du nœud #text
Nom du nœud eleve
Nom du nœud #text
Nom du nœud eleve
Nom du nœud #text
Nom du nœud eleve
Nom du nœud #text
```

Lecture des nœuds « eleve » - getChildNodes()

```
**********Méthode 2 : getChildNodes()**********
NodeList eleves_bis =  liste.getChildNodes();
for (int o = 0;o<eleves_bis.getLength();o++){
    //On affiche le nom de tous les nœuds fils de l'élément racine <liste>
    System.out.println("Nom du nœud "+eleves_bis.item(o).getNodeName());
    //Problème d'objets textes (#text) parasites si on active la ligne ci-dessous :
    System.out.println("Année-Section "+eleves_bis.item(o).getFirstChild().getNodeValue());
}
```

Voir « Read.java »

Lecture des nœuds « eleve » - getChildNodes()

- //Problème d'objets textes (#text) parasites si on active la ligne ci-dessous :
- **System.out.println("Année-Section"+eleves_bis.item(o).getFirstChild().getNodeValue());**
- On aura un problème d'objets textes (#text) parasites **(sauts de ligne, tabulations, espaces).**
- Exception in thread "main" java.lang.NullPointerException
- at DOM.Read.main(Read.java:xx)

- **Remarque : Les espaces, les sauts de ligne et les tabulations sont considérés par DOM comme des contenus textuels/textes/nœuds de texte (ce sont des objets Textes/#text), au même titre qu'un objet texte/#text de type chaîne de caractères d'un nœud d'élément. Dans le premier cas, ce sont des contenus textuels produits par la mise en forme d'un fichier et sont donc des objets textes parasites.**

Lecture des nœuds « eleve » - getChildNodes()

- **Il faudra alors filtrer ces objets textes (#text) parasites.**
- **Comment ?**
- **La réponse est dans les slides suivants (voir codes : Read.java+Filtrer_parasites.java).**

Lecture des nœuds « eleve » - getChildNodes() - filtrage des objets textes (#text) parasites

- On crée la classe « Filtrer_parasites » contenant la méthode « Filtrer() », qui renvoie un booléen (voir « Filtrer_parasites.java ») :

- public class Filtrer_parasites {
  - public boolean Filtrer(Node noeud) {
    - return (noeud.getNodeName() != "#text");
  - }
- }

Voir « Filtrer_parasites.java »

Filtrage des objets textes (#text) parasites

- //On exploite la méthode Filtrer() de notre classe « Filtrer_parasites » à partir de « Read.java » (classe « Read »).

- Filtrer_parasites a = new Filtrer_parasites();
- if(a.Filtrer(eleves_bis.item(0)))
  - System.out.println(eleves_bis.item(0).getFirstChild().getNode Value());

Filtrage des objets textes (#text) parasites

Voir « Read.java » et « Filtrer_parasites.java »

```
//On exploite la méthode Filtrer() de notre classe « Filtrer_parasites »
//à partir de « Read.java » (classe « Read »).
Filtrer_parasites a = new Filtrer_parasites();
if(a.Filtrer(eleves_bis.item(o)))
    System.out.println("Année-Section "+eleves_bis.item(o).getFirstChild().getNodeValue());
```

```
package DOM;
import org.w3c.dom.Node;
/**
Cette classe permet de filtrer les noeuds parasites
de type "#text" rencontrés dans un document XML.
*/
public class Filtrer_parasites{
    public boolean Filtrer (Node noeud){
        return (noeud.getNodeName() != "#text");
    }
}
```

Lecture des nœuds « eleve » - getChildNodes() - filtrage des objets textes (#text) parasites

- On peut aussi créer dans la classe « Read » une méthode non-statique « **Filtrer** », qui renvoie un booléen :
- //On ne peut pas faire, à partir de la méthode statique « main », une référence statique (static) à la méthode non-statique (non-static) « **Filtrer** ». Donc, on instancie au préalable un objet de la classe « Read ».
- **Read a = new Read();**
- if(**a.Filtrer(**eleves_bis.item(o)**)**)
  - System.out.println(eleves_bis.item(o).getFirstChild().getNode Value());

  //**********//
- public boolean Filtrer **(**Node noeud**)**{
  - return (noeud.getNodeName() != "#text");
  - }

Lecture des nœuds « eleve » - getChildNodes() - filtrage des objets textes (#text) parasites

- Sinon on peut créer une méthode statique « Filtrer() », qui renvoie un booléen :
- //Inutile de mettre le nom de la classe en mode accès static "Read.Filtrer(eleves_bis.item(o))" puisque les méthodes "Filtrer()" et "main()" appartiennent à la même classe « Read ».
- if(Filtrer(eleves_bis.item(o)))
  - System.out.println(eleves_bis.item(o).getFirstChild().getNode Value());
- //**********//
- public static boolean Filtrer(Node noeud){
  - return (noeud.getNodeName() != "#text");
- }

Résultat du filtrage des objets textes (#text) parasites

```
Nom du nœud #text
Nom du nœud eleve
Année-Section L2 ACAD
Nom du nœud #text
Nom du nœud eleve
Année-Section L3 ACAD
Nom du nœud #text
Nom du nœud eleve
Année-Section L2 ACAD
Nom du nœud #text
```

Exercice

- Lire tout le contenu (pas les objets textes parasites) des deux documents XML (contacts.xml et AAA.xml), qui se trouvent également dans le fichier « exercice-cours-7-support.pdf ».
- Chaque document XML du fichier « exercice-cours-7-support.pdf » doit être placé dans un fichier **.xml** à part.

- Le programme JAVA doit afficher le résultat de cette lecture sur la console de votre IDE (Eclipse, NetBeans).