JAXP

(Java API for XML Processing)

L’API JAXP

- JAXP est un ensemble d'APIs Java permettant d'effectuer des traitements XML qui inclut SAX (SAX 2.0), DOM (DOM Level 2) et la manipulation de transformations XSLT.
- Le but est de :
  - Fournir une couche d’abstraction au dessus de SAX ou DOM afin de garantir un effort de standardisation des aspects non couverts par SAX et DOM
    - L’analyse
- L'API est le fruit de travaux de développement communs à Ariba, AOL/Netscape, BEA, Bluestone, Fujitsu, IBM, et Oracle.

L'API JAXP

Raison d'utiliser JAXP:

- DOM ne spécifie pas comment créer un objet parseur → différent API selon implémentation → portabilité du code est affecté
  - En utilisant DOM (ou SAX), il faut explicitement importer et référencer la classe de l'analyseur

  (exp : org.apache.xerces.parsers.DOMParser)
  - Si on change de parseur → problèmes

L’API JAXP

Raison d’utiliser JAXP:

- Avant chaque opération DOM, le document XML a besoin d’être parser et un objet doit être créer. Cette opération n’est pas standardiser par le W3C
- indépendance des processeurs

JAXP offre la possibilité de changer le parseur sans avoir à recompiler l’application.

JAXP : document DOM

- Chargement d'un document
  - L'api JAXP utilise le design pattern d'usine abstraite.
    javax.xml.parsers.DocumentBuilderFactory

    DocumentBuilderFactory usine =

    DocumentBuilderFactory.newInstance();
  - L'usine peut ensuite être configurée,

    .setNamespaceAware .setValidating
    .setIgnoringElementContentWhitespace
  - puis utilisée pour obtenir un analyseur DOM.

    javax.xml.parsers.DocumentBuilder;

    DocumentBuilder analyseur = usine.newDocumentBuilder();

JAXP : document DOM

- Chargement d'un document
  - L'analyseur peut ensuite charger un document, être utilisé pour obtenir une **DOMImplementation** (qui permet quelques manipulations intéressantes

    org.w3c.dom.Document

    Document doc=analyseur.parse(xmlFile);
  - Pour obtenir des messages d'erreurs personnalises, on peut enregistrer un **ErrorHandler** (de SAX) auprès de l'analyseur.

    org.xml.sax.SAXException

    analyseur.setErrorHandler(new ErrorClass())

JAXP : document DOM

- Manipulations de base
  - L'interface **Document** :
    - 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 indiques par les paramètres.
    - NodeList getElementsByTagName(String name) : même chose que la méthode précédente, mais en sélectionnant par le nom qualifié des éléments.
    - getElementById qui permet de retrouver l'élément référencé par un ID (il faut que l'analyseur soit validant pour pouvoir faire cette recherche).

JAXP : document DOM

- Manipulations de base
  - L'interface Node
    - racine de la hiérarchie de classes de DOM
    - Chaque partie (élément, texte, attributs, etc.) d'un document XML est représenté par une classe qui implante Node.
  - quelques méthodes de cette interface :
    - String getNamespaceURI(), String getLocalName(), String getPrefix(), String getNodeName() (Renvoie null pour autre chose qu'un attribut ou un élément).
    - String getNodeValue() (Renvoie null sauf pour un nœud attribut ou texte)

JAXP : document DOM

- Manipulations de base
  - quelques méthodes de cette interface :
    - 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 null pour la racine de l'arbre)
    - méthodes de navigation (exp getPreviousSibling)
  - l'interface NodeList :
    - int getLength() : nombre de nœuds contenus dans la liste.
    - Node item(int index) : Renvoie le nœud numéro index (le premier porte le numéro 0).

JAXP : document DOM

- Manipulations de base
  - l'interface Element (qui hérite de Node):
    - String getAttribute(String name)
    - String getAttributeNS(String URI,String name)
    - hasAttribute et hasAttributeNS : retourne un booléen
    - propose les mêmes méthodes getElements... que l'interface Document ()

JAXP : Itérations

- L'API DOM **Traversal** propose des techniques basées sur le design pattern itérateur pour parcourir un arbre DOM de façon plus simple que le traitement précédent.
- 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.

JAXP : Itérations

- 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.

JAXP : Itérations

- Comment ?
  - On crée un NodeIterator en utilisant une usine abstraite, à savoir un objet qui implante l'interface DocumentTraversal
  - 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).

JAXP : Itérations

- Comment ?
  - Toute la souplesse des NodeIterators réside dans le deuxième et le troisième paramètre : Ils précisent le mode de filtrage.
    - Le deuxième paramètre est un entier qui contient un masque (défini dans l'interface NodeFilter) qui permet de sélectionner des grandes catégories de nœuds
    - Exp : NodeFilter.SHOW_ELEMENT assure que le NodeFilter ne va parcourir que les nœuds correspondant à des éléments. ➔ sélectionner les nœuds de texte, les commentaires, …etc.

JAXP : Itérations

- Comment ?
  - 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 filtre accepte le nœud paramètre.
    - La réponse négative est codée par NodeFilter.FILTER_SKIP ou NodeFilter.FILTER_REJECT

JAXP : Itérations

- Grâce a 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 methode getElementsByTagName.
- Exp 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, a ne pas confondre avec les éléments **a** contenant un attribut href qui sont eux des liens (vers les cibles).

JAXP : Itérations

- 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;
    }
}
```

JAXP : Itérations

- Utilisation :

```
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();
    }
}
```

JAXP : XSLT

- JAXP supporte XSLT avec le package **javax.xml.transform**, qui permet de brancher un transformateur XSLT pour effectuer des transformations.
- Ses sous-paquetages permettent d'effectuer des transformations directement à partir d'arbres DOM et d’événements SAX .

JAXP : XSLT

Exp1 : transformer arbre DOM en document XML

- Pour transformer un arbre DOM en un document XML, on crée d'abord un objet **Transformer** qui va effectuer la transformation.

TransformerFactory **transFactory** = TransformerFactory.newInstance();

Transformer **transformer** = transFactory.newTransformer();

JAXP : XSLT

Exp1 : transformer arbre DOM en document XML

- À partir du nœud racine de l'arbre DOM, on construit un objet **DOMSource** source de la transformation.

DOMSource **source** = new DOMSource(document);

JAXP : XSLT

- En suite on crée un objet **StreamResult** afin de prendre les résultats de la transformation et transcrire l'arbre dans un fichier XML.

```
File newXML = new File("newXML.xml");
FileOutputStream os = new
FileOutputStream(newXML);
StreamResult result = new StreamResult(os);
transformer.transform(source, result);
```

JAXP : XSLT

Exp 2 : transformer un doc XML en page HTML

- Pour effectuer la transformation, on doit se procurer un transformateur XSLT et l'utiliser pour appliquer la feuille de style sur les données XML.
- Le fragment de code suivant
  - obtient un transformateur par l'instanciation d'un objet **TransformerFactory**,
  - lit la feuille de style et le fichier XML,
  - Crée un fichier HTML pour la sortie,
  - et finit par obtenir l'objet transformateur provenant de l'objet tFactory de TransformerFactory.

JAXP : XSLT

Exp 2 : transformer un doc XML en page HTML

```
TransformerFactory tFactory
    =          TransformerFactory.newInstance();
String stylesheet = "myStyle.xsl";
String sourceId = "myXML.xml";
File myHTML = new File("myHTML.html");
FileOutputStream os = new FileOutputStream(myHTML);
Transformer transformer
    =   tFactory.newTransformer(new
    StreamSource(stylesheet));
```

JAXP : XSLT

Exp 2 : transformer un doc XML en page HTML

- La transformation est effectuée en invoquant la méthode **transform** :

```
transformer.transform(
    new StreamSource(sourceId),
    new StreamResult(os)
    );
```