

## BDAV - APIs XML

E.Coquery

`emmanuel.coquery@liris.cnrs.fr`

`http://liris.cnrs.fr/~ecoquery/`

# Document Object Model

- Modèle objet pour représenter les arbres XML
- et aussi plein d'autres choses ...
  - événements, styles, validation, interrogation, etc

→ on peut tirer une API de ce modèle

- Pourquoi faire ?
  - Parcourir un document
  - Modifier un document
  - Créer un document
- En général, une implémentation de DOM sont fournies avec des parseurs et des fonction pour écrire les documents.

# Le DOM en Java

Dans la bibliothèque standard Java :

- Paquet `org.w3c.dom`
- Un ensemble d'interfaces représentant les différents types de noeuds :

Node

- Attr
- Element
- CharacterData
  - CDATASection
  - Comment
  - Text
- ...

## Digression : usines (factories)

- Mécanisme très utilisé dans J2EE
- Classe ou objet avec des méthodes servant à créer des objets
- Finalité différente d'un constructeur :
  - Le but n'est pas d'initialiser l'objet
  - Objectif fournir un objet qui est implémentation d'une interface
  - La classe réelle de l'objet créé est "cachée"
- Permet d'avoir des mécanismes d'implémentation par défaut et de choix d'implémentation

## Parsing et création

```
import javax.xml.parsers.*;  
import org.w3c.dom.*;
```

```
DocumentBuilder db =  
    DocumentBuilderFactory  
        .newInstance()  
        .newDocumentBuilder();
```

```
Document doc = db.newDocument();
```

```
Document doc2 = db.parse("toto.xml");  
// On peut utiliser n'importe quelle url ici
```

## Création et ajout de noeuds

```
Element principal = doc.createElement("carnet");
Element personne1 = doc.createElement("personne");
Comment commentaire =
    doc.createComment("un commentaire");
Text texte =
    doc.createTextNode("8_bvd_N.Bohr, Villeurbanne");

doc.appendChild(principal);
principal.appendChild(personne1);
principal.appendChild(commentaire);
personne1.appendChild(texte);
personne1.setAttribute("nom", "Toto");
```

## Parcours d'arbre XML

```
public void printXMLElements(Element el) {
    System.out.print(el.getTagName());
    NamedNodeMap nnm = el.getAttributes();
    for (int i = 0; i < nnm.getLength(); i++)
        System.out.print("[" +
            nnm.item(i).getNodeName() + ":"
            + nnm.item(i).getNodeValue() + "]" )

    System.out.println();
    NodeList nl = el.getChildNodes();
    for(int i = 0; i < nl.getLength(); i++)
        if (nl.item(i).getNodeType() == Node.ELEMENT_NODE)
            printXMLElements((Element) nl.item(i));
}
```

## Écriture dans un fichier

- Java : pas de méthode “write”

```
import org.w3c.dom.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;

TransformerFactory tf =
    TransformerFactory.newInstance();
Transformer copy = tf.newTransformer();
copy.transform(
    new DOMSource(doc),
    new StreamResult("monFichier.xml"));
```

# SAX

- Simple API for XML
- API pour lire des fichiers XML
- Gère l'aspect lexical (découpage en tag, attributs, ...)
- Ne gère pas la structure arborescente
  - Pas besoin de stocker tout le document en mémoire
- Principe du “push parsing” :
  - Le programmeur fournit les actions à effectuer sur chaque objet lexical (élément, commentaire, texte, etc)
  - Le parseur appelle les actions correspondantes lorsqu'il rencontre un élément, du texte, etc ...
- Style de programmation événementielle.

## interface org.xml.sax.ContentHandler

Méthodes (à implémenter) utilisées par un parseur sax pour gérer les objets lexicaux :

- void startDocument()
- void endDocument()
- void startElement(String uri, String localName, String qName, Attributes atts)
- void endElement(String uri, String localName, String qName)
- void characters(char[] ch, int start, int length)
- ...

Possibilité d'étendre la classe org.xml.sax.helpers.DefaultHandler

- Nécessaire pour les méthodes parse de la classe SAXParser

## Exemple d'extension de DefaultHandler

```
public class ElementPrinter extends DefaultHandler
  public void startElement(String uri ,
    String localName , String name ,
    Attributes atts) throws SAXException {
    System.out.print(localName);
    for (int i = 0; i < atts.getLength(); i++)
      System.out.print(" [" + atts.getLocalName(i)
        + ":" + atts.getValue(i) + "]");
    }
}
```

# Utilisation

```
import org.xml.sax.helpers.*  
import javax.xml.parsers.*;  
  
SAXParserFactory spf =  
    SAXParserFactory.newInstance();  
SAXParser sp = spf.newSAXParser();  
DefaultHandler myHandler =  
    new ElementPrinter();  
  
sp.parse(fichier, myHandler);
```

# StAX

- Streaming API for XML (Java SE 6)
  - Permet de lire et de créer des documents XML
  - Peut être vu comme un compromis entre DOM et SAX
- 
- Principe du “pull parsing”
    - Le programmeur contrôle la boucle de lecture
  - Style de programmation plus classique que SAX

# javax.xml.stream.XMLStreamReader

Méthode `int next()`

- passe à l'objet syntaxique suivant
- renvoie un entier spécifiant le type d'objet lu :
  - `XMLStreamReader.START_ELEMENT`
  - `XMLStreamConstants.END_ELEMENT`
  - `XMLStreamConstants.START_DOCUMENT`
  - `XMLStreamConstants.END_DOCUMENT`
  - `XMLStreamConstants.CHARACTERS`
  - `XMLStreamConstants.CDATA`
  - `XMLStreamConstants.COMMENT`
  - ...

Méthode `boolean hasNext()`

# javax.xml.stream.XMLStreamReader

Méthodes d'accès :

- String getLocalName()
- int getAttributeCount()
- String getAttributeLocalName(int index)
- String getAttributeValue(int index)
- String getText()
- ...

# Utilisation de XMLStreamReader

```
public void printStAX(String fichier) throws XMLStreamExcept
XMLInputFactory xif = XMLInputFactory.newInstance();
XMLStreamReader sr =
    xif.createXMLStreamReader(new StreamSource(fichier));
while (sr.hasNext()) {
    int code = sr.next();
    switch (code) {
    case XMLStreamReader.START_ELEMENT:
        System.out.println();
        System.out.print(sr.getLocalName());
        for (int i = 0; i < sr.getAttributeCount(); i++) {
            String att = "[" + sr.getAttributeLocalName(i) + ":"
                + sr.getAttributeValue(i) + "]";
            System.out.print(att);
        }
    }
}
xif.close();
}
```

# StAX : Création de document

## Interface javax.xml.stream.XMLStreamWriter

- void writeStartElement(String localName)
- void writeAttribute(String localName, String value)
- void writeEndElement()
- void writeCharacters(String text)
- void writeComment(String data)
- void writeStartDocument()
- void writeEndDocument()

## Utilisation de XMLStreamWriter

```
XMLOutputFactory xof =
    XMLOutputFactory.newInstance();
XMLStreamWriter xsw =
    xof.createXMLStreamWriter(System.out);
xsw.writeStartDocument();
xsw.writeStartElement("carnet");
xsw.writeStartElement("personne");
xsw.writeAttribute("nom", "Toto");
xsw.writeStartElement("adresse");
xsw.writeCharacters("8_Bvd_Niels_Bohr");
xsw.writeEndElement();
xsw.writeEndElement();
xsw.writeEndElement();
xsw.writeEndDocument();
xsw.close();
```

# DOM, SAX, StAX ?

	DOM	SAX	StAX
Lecture	Oui	Oui (push)	Oui (Pull)
Ecriture	Indirecte	Non	Oui
Modification	Oui	Non	Non
Parcours non ordonné	Oui	Non	Non
Conso. Mémoire	Élevée	Basse	Basse

# Validation

## Classes

- `javax.xml.validation.Schema`  
Représente un schema, une XML, etc
- `javax.xml.validation.Validator`  
Objet effectuant la validation

```
SchemaFactory sf =  
    SchemaFactory.newInstance  
        (XMLConstants.W3C_XML_SCHEMA_NS_URI);  
Schema schema =  
    sf.newSchema("monSchemaXML.xsd");  
Validator validator =  
    schema.newValidator();  
validator.validate(new StreamSource("fichier.xml"))
```

# Validation à la lecture

Dans DocumentBuilderFactory et SAXParserFactory :

- `setSchema(schema)` : schéma à utiliser pour la validation
- `setValidating(true)` : active la validation

```
SchemaFactory sf =  
    SchemaFactory.newInstance  
        (XMLConstants.W3C_XML_SCHEMA_NS_URI);  
Schema schema =  
    sf.newSchema("monSchemaXML.xsd");  
DocumentBuilderFactory dbf =  
    DocumentBuilderFactory.newInstance();  
dbf.setSchema(schema);  
dbf.setValidating(true);  
DocumentBuilder bd = dbf.newDocumentBuilder();  
Document doc = db.parse("fichier.xml");
```

# TrAX

- Transformation API for XML (Java)
- Exécution d'un programme XSLT en Java
- Source
  - Représente un document à traiter ou du code xsl
  - DOMSource, SAXSource, StAXSource, StreamSource
- Result
  - Représente "l'endroit" où mettre le résultat
  - DOMResult, SAXResult, StAXResult, StreamResult

## TrAX : exemple

```
public void exempleTrax(String entree ,  
    String sortie , String xsl)  
    throws TransformerException {  
  
    TransformerFactory tf =  
        TransformerFactory.newInstance();  
  
    Transformer trans =  
        tf.newTransformer(new StreamSource(xsl));  
  
    trans.transform(new StreamSource(entree),  
        new StreamResult(sortie));  
}
```

# XML et BD relationnelles

## Plusieurs angles d'attaque

- Traitement XML purement au niveau applicatif (pas de lien direct)
- Génération de XML via la BD
  - fonctions et types XML dans la norme SQL 2003 (Oracle, DB2, SQL Server, PostgreSQL)
- Stockage de (fragments de) documents XML dans la BD
  - Brut : CLOB et assimilés
  - Optimisé : utilisation des infos de schéma pour avoir une représentation relationnelle efficace
  - Utilisation de XQuery (et/ou XSLT) pour les requêtes
  - Accès à certaines relations sous forme XML.

# Fonctions SQL de génération XML

- Le type XMLType représente un fragment de document
  - Représentation interne au SGBD, même si souvent sérialisé textuellement au final
- Fonctions de génération :
  - XMLElement(name "nom-element", valeur)
  - XMLAttributes(valeur, valeur AS nom\_attribut, ...)
    - utilisé dans XMLElement de manière optionnelle
  - XMLForest(...)
  - XMLText(...), XMLComment(...), ...
  - XMLAgg(...), XMLAgg(..., order by ...)

## Exemple

```
SELECT XMLElement(name "carnet",
                 XMLAttributes(CARNET_ID as "id"),
                 XMLAgg(XMLElement(name "personne",
                                   XMLForest(NOM as "nom",
                                             TELEPHONE as "tel")))
                 order by NOM))
AS CARNET_XML
FROM Personne
GROUP BY CARNET_ID
```

lignes produites avec un attribut relationnel CARNET\_XML de la forme :

```
<carnet id="2">
  <personne><nom>Titi</nom><tel>01234567</tel> </personne>
  <personne><nom>Toto</nom><tel>02345678</tel> </personne>
</carnet>
```

# Manipulation de données XMLType

- Fonctions de mise à jour :
  - UpdateXML(XMLType\_instance, XPath\_string, value\_expr)
  - DeleteXML(XMLType\_instance, XPath\_string)
  - ...
- XQuery :
  - Fonctions SQL :  
XMLQuery(XQuery\_string RETURNING CONTENT)  
XMLQuery(XQuery\_string PASSING val1 as "nom\_var1", ...  
RETURNING CONTENT)
  - Oracle : Commande XQUERY  
XQUERY  
... code XQuery ...  
/
- Dans Oracle, la fonction XPath/XQuery  
ora:view(table\_ou\_vue) renvoie une représentation XML  
d'une relation

## Exemple de mise à jour

Insérer un attribut `ident` dans l'élément principal de chaque document, avec pour valeur l'identifiant du document, dans une table `DOCS(id INTEGER, doc XMLType)`

```
UPDATE DOCS  
SET doc := UpdateXML(doc,'/*/@ident',id)
```

## Exemple XQuery

```
SELECT XMLQuery('<personne>$n$t</personne>'  
             PASSING XMLElement(name "nom", nom) AS "n",  
                    XMLElement(name "tel", telephone) AS "t"  
             RETURNING CONTENT)  
FROM carnet WHERE id = 2
```