

Les API pour XML

Fabrice Rossi

<http://apiacoa.org/contact.html>

Université Paris-IX Dauphine

Les API pour XML – p. 1/65

Analyse syntaxique

- **Principe** : lire le document et reconnaître les structures XML (balise ouvrante, attribut, etc.)
- Un grand standard : *Simple API for Xml* (SAX)
- Un concurrent (très spécifique) : *Xerces Native Interface* (XNI)
- Avantages (du travail au niveau *parsing*) :
 - efficacité maximale
 - occupation mémoire minimale
 - très souple
- Inconvénients :
 - délicat à programmer
 - il faut tout faire !
- En général, la validation est déjà possible au niveau de SAX (ou de XNI)

Les API pour XML – p. 3/65

Le problème

Comment manipuler un document XML dans un programme ?

- très bas niveau : analyse syntaxique (*parsing*)
- bas niveau : validation
- niveau intermédiaire :
 - manipulations du document chargé en mémoire
 - sauvegarde d'un document
- haut niveau :
 - sérialisation (*binding*)
 - transformations d'un document XML
 - interprétation d'un format XML particulier :
 - SVG
 - MathML
 - XSL:FO
 - etc.
 - etc.

Les API pour XML – p. 2/65

Chargement en mémoire

- **Principe** : lire le document, le placer en mémoire et fournir une API permettant sa manipulation
- Un grand standard : *Document Object Model* (DOM) du W3C
- Des concurrents :
 - JDOM (<http://www.jdom.org>) même esprit que DOM, mais spécifique à Java et plus simple
 - XOM (<http://cafeconleche.org/XOM/>) : tente de faire mieux que DOM et JDOM (pour Java)
 - Electric XML : intéressant mais non *open source*
- Principe informatique commun : un document XML est un arbre
- Avantages :
 - beaucoup de fonctions déjà définies
 - modifications, manipulations globales, etc.
- Inconvénient majeur : performances (mémoire et temps)

Les API pour XML – p. 4/65

Les API de *binding*

- **Principe** : transformation “automatique” et réversible d’un document XML en un objet
- une forme de persistance pour les langages OO
- le futur pour de nombreuses applications
- Des solutions :
 - travail officiel en cours pour Java : Java Architecture for XML Binding (JAXB), <http://java.sun.com/xml/jaxb/>
 - CASTOR (<http://castor.exolab.org>) : permet le *binding* d’objets Java vers XML mais aussi la persistance par d’autres techniques (bases de données, LDAP, etc.)
 - ZEUS (<http://zeus.enhydra.org/>), toujours pour Java
- Point négatif : vision objet pure et dure (pas vraiment adapté à des documents XML textuels)

API

SAX

Autres API (Java)

- Pour les opérations de bas niveau (i.e., chargement et validation), on trouve :
- Java API for XML Processing (JAXP)
(<http://java.sun.com/xml/jaxp/>) :
 - sorte de couche d’abstraction
 - comment charger et configurer un analyseur (SAX ou DOM)
 - depuis la version 1.1, comment lancer une transformation XSLT : TrAX (transformation API for XML)
 - datatype : ajout de nouveaux types de données aux validateurs RELAX NG
 - autres API disponibles, mais pour des tâches de plus haut niveau (par exemple RPC XML)

SAX

Simple API for Xml :

- développement communautaire
- <http://www.saxproject.org/>
- 1.0 : 11 Mai 98, 2.0 : 5 Mai 2000, 2.0.1 : 29 Janvier 2002
- nombreuses implantations (par exemple Xerces, open source)

Caractéristiques majeures :

- modèle évènementiel : évènement d’analyse syntaxique
- bas niveau
- efficace
- délicat à mettre en œuvre

Langage de référence : Java (officiellement intégré à j2se 1.4 avec JAXP 1.1). Actuellement :

- Xerces : Java, C++, COM, Perl
- Microsoft : binding COM et donc C, C++, VB, etc.
- Support en Python, Perl, etc.

Modèle évènementiel

Principes :

- un document est transformé en un flux d'**évènements syntaxiques** (balise ouvrante, balise fermante, etc.)
- **aucune mémoire**
- une application est un "écouteur" d'évènements
- modèle calqué sur celui des interfaces graphiques
- correspond à un parcours préfixe en profondeur de l'arbre du document : c'est l'ordre des informations dans le fichier
- **lecture seule**

Exemple

```
hello-sax.xml
1 <?xml version="1.0"?>
2 <doc>
3 <p>Un texte</p>
4 </doc>
```

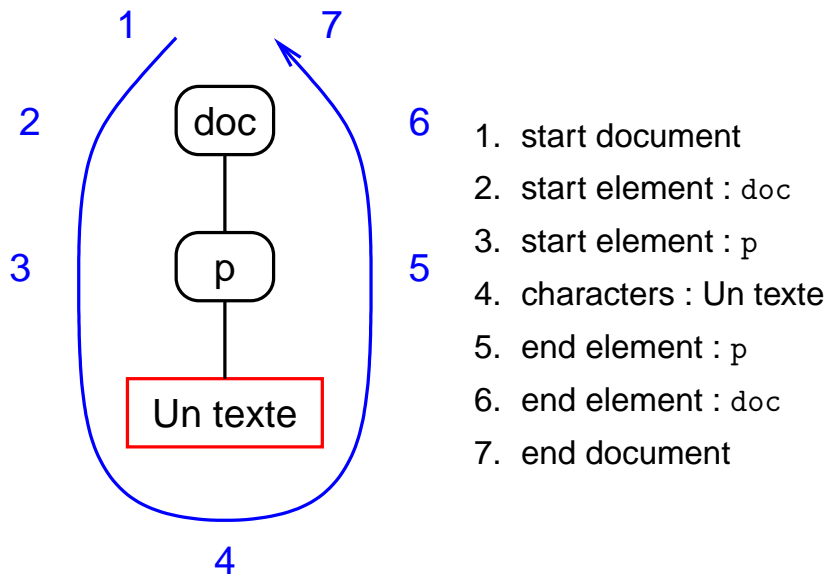
Evènements engendrés :

1. start document
2. start element : doc
3. start element : p
4. characters : Un texte
5. end element : p
6. end element : doc
7. end document

Les API pour XML - p. 9/65

Les API pour XML - p. 10/65

Exemple (suite)



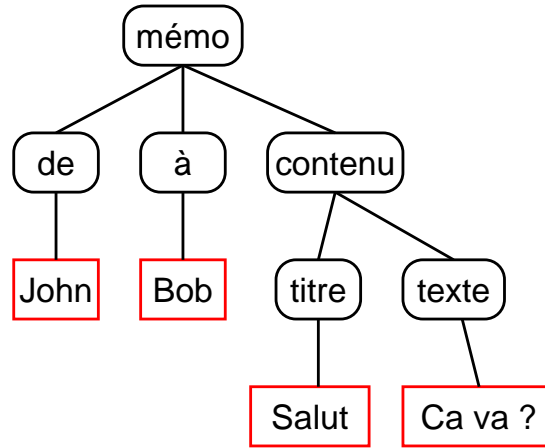
Les API pour XML - p. 11/65

Exemple 2 (1)

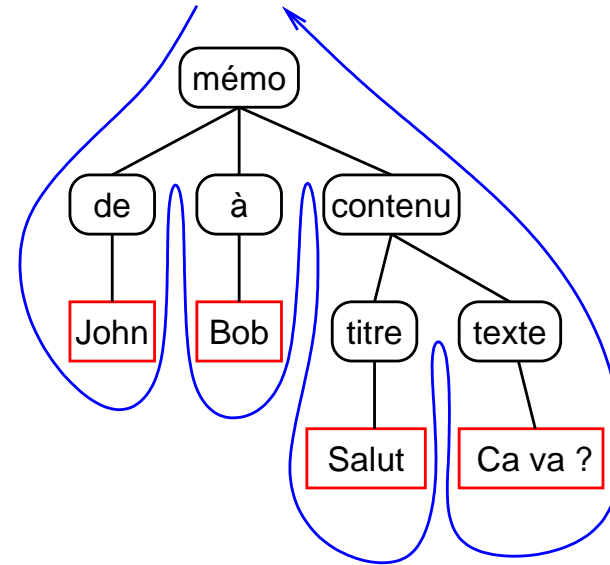
```
a-memo.xml
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <mémo>
3 <de>John</de>
4 <à>Bob</à>
5 <contenu>
6 <titre>Bonjour</titre>
7 <texte>Ca va ?</texte>
8 </contenu>
9 </mémo>
```

Les API pour XML - p. 12/65

Exemple 2 (2)



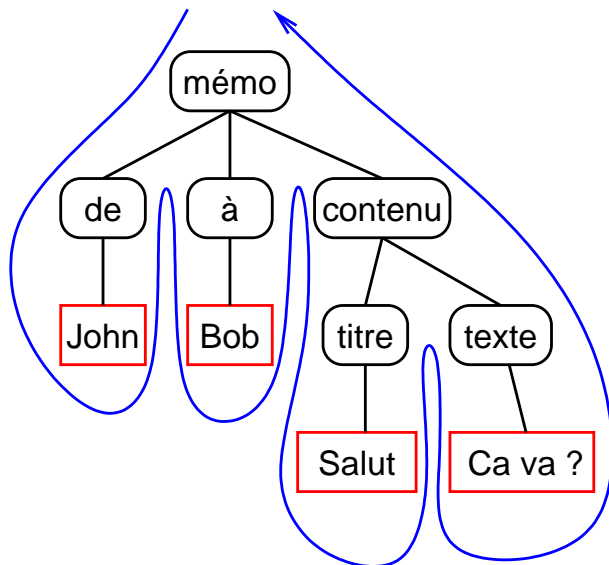
Exemple 2 (3)



Les API pour XML – p. 13/65

Les API pour XML – p. 14/65

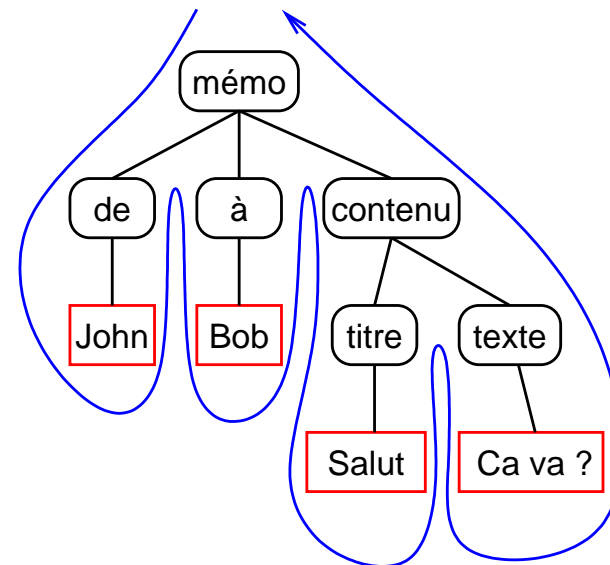
Exemple 2 (4)



1. SD
2. SE : mémo
3. SE : de
4. C : John
5. EE : de
6. SE : à
7. C : Bob
8. EE : à

Les API pour XML – p. 15/65

Exemple 2 (5)



9. SE : contenu
10. SE : titre
11. C : Salut
12. EE : titre
13. SE : texte
14. C : Ca va ?
15. EE : texte
16. EE : contenu
17. EE : mémo
18. ED

Les API pour XML – p. 16/65

Mécanismes

Tout est basé sur quatre interfaces :

- **ContentHandler** :
 - la plus importante
 - réagit aux évènements : une méthode par type d'évènement (exemple `startDocument`)
- **ErrorHandler** : gestion des erreurs, en particulier les problèmes de validation
- **DTDHandler** : gestion d'une petite partie des DTDs
- **EntityResolver** : gestion du remplacement des entités externes

Les quatre interfaces sont implantées par `DefaultHandler`, qui ne fait pas grand chose (même principe que les couples `Listener/Adapter` de Java).

Fonctionnement d'un analyseur

1. ouverture du fichier à analyser
2. lecture d'une partie significative du fichier :
3. si cette partie
 - (a) pose un problème (document mal formé ou non valide), appel d'une méthode du `ErrorHandler`
 - (b) est une référence à une entité (`&truc;`), appel d'une méthode du `EntityResolver`
 - (c) est un composant de la DTD, appel d'une méthode du `DTDHandler`
 - (d) est une balise, du texte, etc., appel d'une méthode du `ContentHandler`
4. si le document n'est pas terminé, retour en 2

Exemple

hello-sax.xml

```
1 <?xml version="1.0"?>
2 <doc>
3 <p>Un texte</p>
4 </doc>
```

Evènement	Appel
start document	<code>ContentHandler.startDocument</code>
start element : doc	<code>ContentHandler.startElement</code>
start element : p	<code>ContentHandler.startElement</code>
characters : Un texte	<code>ContentHandler.characters</code>
end element : p	<code>ContentHandler.endElement</code>
end element : doc	<code>ContentHandler.endElement</code>
end document	<code>ContentHandler.endDocument</code>

Modèle de mise en œuvre

Deux composants :

1. un programme principal basé sur JAXP
2. une implantation de `ContentHandler`

Comportement attendu : affichage les éléments contenus dans le document XML analysé.

Implantation du ContentHandler

```
1 import org.xml.sax.helpers.DefaultHandler;
2 import org.xml.sax.Attributes;
3 public class PrintElement extends DefaultHandler {
4     public void startElement(String namespaceURI, String localName,
5                             String qName, Attributes atts) {
6         System.out.println(qName);
7     }
8 }
```

- on dérive de DefaultHandler : permet de ne programmer que ce qui nous intéresse
- la méthode startElement est appelée à chaque balise ouvrante rencontrée
- Attributes : les attributs de la balise ouvrante
- les autres paramètres : nom de l'élément (prise en compte des espaces de noms)

Les API pour XML – p. 21/65

Modèle

```
1 import javax.xml.parsers.SAXParserFactory;
2 import javax.xml.parsers.SAXParser;
3 import org.xml.sax.SAXException;
4 import java.io.IOException;
5 import java.io.File;
6 public class AnalyseDeBase {
7     public static void main(String[] args) {
8         if(args.length!=1) {
9             System.err.println("usage: AnalyseDeBase xmlfile");
10            System.exit(1);
11        }
12        try {
13            // création de l'usine
14            SAXParserFactory factory=SAXParserFactory.newInstance();
15            factory.setNamespaceAware(true);
16            factory.setValidating(false);
17            // création de l'analyseur
18            SAXParser parser=factory.newSAXParser();
```

Les API pour XML – p. 23/65

Analyse syntaxique

Avec JAXP :

- permet de changer d'implantation de SAX sans modifier l'application
- algorithme :
 1. on crée une usine à parser (SAXParserFactory)
 2. on configure l'usine (c'est-à-dire les analyseurs qu'elle va produire)
 3. on crée l'analyseur (SAXParser)
 4. on lance l'analyse syntaxique (méthode parse)
- l'analyse syntaxique se fait à partir d'un DefaultHandler (i.e., toutes les interfaces implantées d'un seul coup)
- solution plus souple en obtenant un XMLReader (SAX)

Les API pour XML – p. 22/65

Modèle (2)

```
19 // ContentHandler
20 PrintElement handler=new PrintElement();
21 try {
22     // analyse du document
23     parser.parse(new File(args[0]),handler);
24 } catch (SAXException se) {
25     se.printStackTrace();
26 } catch (IOException ioe) {
27     ioe.printStackTrace();
28 }
29 } catch (Exception e) {
30     System.err.println(e);
31 }
32 }
33 }
```

Les API pour XML – p. 24/65

Résultat

Appliqué à

a-memo.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <mémo>
3 <de>John</de>
4 <à>Bob</à>
5 <contenu>
6 <titre>Bonjour</titre>
7 <texte>Ca va ?</texte>
8 </contenu>
9 </mémo>
```

le programme affiche

```
mémo
de
à
contenu
titre
texte
```

Les API pour XML – p. 25/65

Affichage des données

On propose un nouveau DocumentHandler :

PrintText

```
1 import org.xml.sax.helpers.DefaultHandler;
2 import org.xml.sax.SAXException;
3 public class PrintText extends DefaultHandler {
4     public void characters(char[] ch, int start, int length) {
5         System.out.println("'" + new String(ch, start, length) + "'");
6     }
7 }
```

- la méthode `characters` est appelée à chaque évènement de texte
- paramètres :
 - `char[] ch` : tableau de caractères contenant le texte de l'évènement
 - `int start` : début du texte dans le tableau
 - `int length` : longueur du texte
- on modifie `AnalyseBase` pour utiliser `PrintText`

Les API pour XML – p. 26/65

Résultat

Appliqué au mémo, le programme affiche :

```
"
"
"John"
"
"
"Bob"
"
"
"
"Bonjour"
"
"
"Ca va ?"
"
"
"
"
```

Les API pour XML – p. 27/65

Interprétation

- un évènement `characters` à chaque fois qu'il y a du texte en dehors d'une balise
- chaque couple de guillemets correspond à un texte
- caractère invisible : **passage à la ligne !**
- un passage à la ligne après chaque élément (ici 7)
- deux solutions :
 - ajouter une DTD et activer la validation
 - modifier la ligne 16 de `AnalyseDeBase`
 - ajouter éventuellement un `ErrorHandler` (pour avoir les messages d'erreur)
 - déterminer (à la main) si la chaîne de caractères correspond à quelque chose

Les API pour XML – p. 28/65

Version avec validation

On remplace le mémo par :

```
a-memo-dtd.xml
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE mémo [
3 <!ELEMENT mémo (de,à,contenu)>
4 <!ELEMENT de (#PCDATA)>
5 <!ELEMENT à (#PCDATA)>
6 <!ELEMENT contenu (titre,texte)>
7 <!ELEMENT titre (#PCDATA)>
8 <!ELEMENT texte (#PCDATA)>
9 ]>
10 <mémo>
11 <de>John</de>
12 <à>Bob</à>
13 <contenu>
14 <titre>Bonjour</titre>
15 <texte>Ca va ?</texte>
16 </contenu>
17 </mémo>
```

Les API pour XML – p. 29/65

Version avec validation (2)

Quand on active la validation, on obtient l'affichage suivant :

```
"John"
"Bob"
"Bonjour"
"Ca va ?"
```

Interprétation :

- l'analyseur sait que les espaces (au sens large) peuvent être ignorés
- pour une suite d'espaces ignorables, il appelle ignorableWhitespace à la place de characters
- certains analyseurs (comme Xerces) n'ont pas besoin d'être en mode validant pour lire la DTD et l'exploiter (en partie)

Solution intéressante, mais qui nécessite un modèle de contenu et une validation (partielle) de celui-ci.

Les API pour XML – p. 30/65

Version avec validation (3)

Il est resté des subtilités. Par exemple :

```
a-memo-dtd-2.xml
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE mémo SYSTEM "memo.dtd">
3 <mémo><de>John</de><à>Bob</à>
4 <contenu><titre>Bonjour</titre><texte>Ca
5 va
6 ?</texte>
7 </contenu>
8 </mémo>
```

donne :

```
"John"
"Bob"
"Bonjour"
"Ca "
"
va "
"
?"
```

Les API pour XML – p. 31/65

Emplacement dans l'arbre

- une feuille de données peut être traduite en *plusieurs* événements characters
- la méthode characters (comme les autres méthodes) n'a aucun moyen (automatique) de connaître son contexte d'appel :
 - l'appel correspondant à "John" est "identique" à celui correspondant à "Bob"
 - il n'y a pas de lien entre les appels correspondant au contenu de l'élément texte
- question générale : **comment connaître le contexte d'un événement ?**
- réponse : pas de solution automatique ⇒ programmation au cas par cas

Les API pour XML – p. 32/65

Contexte d'un évènement

Cas général :

- on utilise une pile (variable d'instance du ContentHandler)
- chaque balise ouvrante empile un élément
- chaque balise fermante dépile un élément
- à tout moment, la pile contient la branche de l'arbre contexte d'un évènement
- la localisation reste limitée : pas de prise en compte de l'ordre des fils d'un élément (et plus généralement du contexte d'un nœud, excepté les parents)

Les API pour XML - p. 33/65

Une solution par pile

```
ContextStack
1 import org.xml.sax.helpers.DefaultHandler;
2 import org.xml.sax.Attributes;
3 import java.util.*;
4 public class ContextStack extends DefaultHandler {
5     private List stack;
6     public void startDocument() {
7         stack=new ArrayList();
8     }
9     public void startElement(String namespaceURI, String localName,
10                             String qName, Attributes atts) {
11         stack.add(qName);
12         Iterator i=stack.iterator();
13         StringBuffer context=new StringBuffer(i.next().toString());
14         while(i.hasNext()) {
15             context.append('/');
16             context.append(i.next());
17         }
18         System.out.println(context);
19     }
}
```

Les API pour XML - p. 34/65

Une solution par pile (2)

```
ContextStack
20 public void endElement(String namespaceURI, String localName,
21                       String qName) {
22     stack.remove(stack.size()-1);
23 }
24 }
```

Affichage sur le mémo :

```
mémo
mémo/de
mémo/à
mémo/contenu
mémo/contenu/titre
mémo/contenu/texte
```

Remarque : startDocument, méthode appelée une fois au début de l'analyse.

Les API pour XML - p. 35/65

Application

- Extraire le texte du mémo sous forme d'une (unique) String :
- Algorithme naïf : concaténer les chaînes obtenues lors des appels de characters correspondant à l'élément texte
 - Problème en SAX :
 - obtenir le contexte
 - garder en mémoire le résultat des évènements antérieurs
 - communiquer le résultat à la méthode appelante
 - Solution possible :
 - déterminer le contexte
 - conserver les résultats intermédiaire dans des variables d'instance du ContentHandler
 - ajouter des méthodes au ContentHandler pour obtenir les résultats

Les API pour XML - p. 36/65

Une solution par pile

```
1 import org.xml.sax.helpers.DefaultHandler;
2 import org.xml.sax.Attributes;
3 import java.util.*;
4 public class MemoGetText extends DefaultHandler {
5     private List stack;
6     private StringBuffer result;
7     public void startDocument() {
8         stack=new ArrayList();
9         result=new StringBuffer();
10    }
11    public void startElement(String namespaceURI, String localName,
12        String qName, Attributes atts) {
13        stack.add(qName);
14    }
15    public void endElement(String namespaceURI, String localName,
16        String qName) {
17        stack.remove(stack.size()-1);
18    }
19 }
```

Les API pour XML – p. 37/65

Une solution par pile (2)

```
19 public void characters(char[] ch, int start, int length) {
20     if(stack.size()>0 && stack.get(stack.size()-1).equals("texte")) {
21         result.append(new String(ch,start,length));
22     }
23 }
24 // méthode spécifique
25 public String getText() {
26     return result.toString();
27 }
28 }
```

Modification dans AnalyseBase :

```
20 MemoGetText handler=new MemoGetText();
21 try {
22     // analyse du document
23     parser.parse(new File(args[0]),handler);
24     // récupération du résultat
25     String result=handler.getText();
26     System.out.println(result);
27 }
```

Les API pour XML – p. 38/65

Une solution par pile (3)

L'application à

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE mémo SYSTEM "memo.dtd">
3 <mémo><de>John</de><à>Bob</à>
4 <contenu><titre>Bonjour</titre><texte>Ca
5 va
6 ?</texte>
7 </contenu>
8 </mémo>
```

donne :

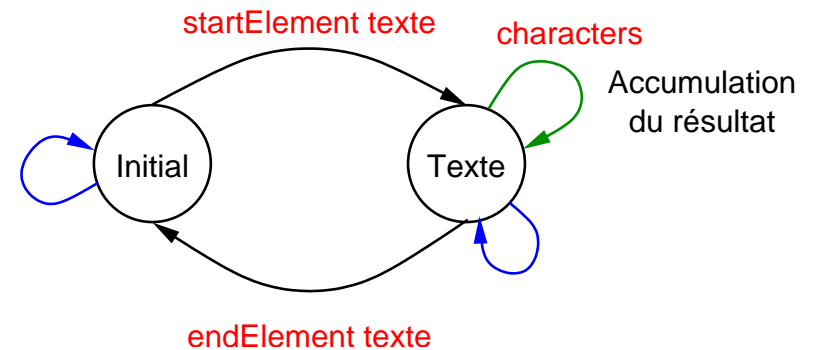
Ca
va
?

sous forme d'une seule String.

Les API pour XML – p. 39/65

Critique

- l'utilisation de la pile n'est pas justifiée
- solution possible par un automate à pile très simple
- Exemple dans le cas du mémo :



Les API pour XML – p. 40/65

Implantation simple

```
1 import org.xml.sax.helpers.DefaultHandler;
2 import org.xml.sax.Attributes;
3 import java.util.*;
4 public class MemoGetText2 extends DefaultHandler {
5     private boolean inText=false;
6     private StringBuffer result=new StringBuffer();
7     public void startElement(String namespaceURI, String localName,
8         String qName, Attributes atts) {
9         if(qName.equals("texte")) {
10             inText=true;
11         }
12     }
13     public void endElement(String namespaceURI, String localName,
14         String qName) {
15         if(qName.equals("texte")) {
16             inText=false;
17         }
18     }
19 }
```

Les API pour XML – p. 41/65

Implantation simple (2)

```
19 public void characters(char[] ch, int start, int length) {
20     if(inText) {
21         result.append(new String(ch,start,length));
22     }
23 }
24 // méthode spécifique
25 public String getText() {
26     return result.toString();
27 }
28 }
```

- ici l'automate est basique \Rightarrow un boolean
- il faut être très attentif aux transitions : ici on reste dans l'état Texte tant qu'on n'a pas vu une balise fermante texte

Les API pour XML – p. 42/65

Implantation simple (3)

L'application à

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <mémo>
3 <de>John</de>
4 <à>Bob</à>
5 <contenu>
6 <titre>Bonjour</titre>
7 <texte><strong>Ca va ?</strong>
8 Je teste le mémo.
9 </texte>
10 </contenu>
11 </mémo>
```

donne :

```
1 Ca va ?
2 Je teste le mémo.
3
```

sous forme d'une seule String.

Les API pour XML – p. 43/65

Gestion des erreurs

- pour gérer les erreurs, il faut implanter ErrorHandler
- très utile quand le *parser* est validant
- Exemple :

```
1 import org.xml.sax.ErrorHandler;
2 import org.xml.sax.SAXParseException;
3 import org.xml.sax.SAXException;
4 public class PrintError implements ErrorHandler {
5     public void error(SAXParseException exception) throws SAXException {
6         System.err.println("error: "+exception);
7     }
8     public void fatalError(SAXParseException exception)
9         throws SAXException {
10         System.err.println("fatal error: "+exception);
11     }
12     public void warning(SAXParseException exception)
13         throws SAXException {
14         System.err.println("warning: "+exception);
15     }
16 }
```

Les API pour XML – p. 44/65

Activation de la validation

```

1 import javax.xml.parsers.SAXParserFactory;
2 import javax.xml.parsers.SAXParser;
3 import org.xml.sax.SAXException;
4 import org.xml.sax.ContentHandler;
5 import java.io.IOException;
6 import java.io.File;
7 import org.xml.sax.XMLReader;
8 public class AnalyseValid {
9     public static void main(String[] args) {
10         if(args.length!=1) {
11             System.err.println("usage: AnalyseValid xmlfile");
12             System.exit(1);
13         }
14         try {
15             SAXParserFactory factory=SAXParserFactory.newInstance();
16             factory.setNamespaceAware(true);
17             // activation de la validation (DTD !)
18             factory.setValidating(true);
19             SAXParser parser=factory.newSAXParser();
20             MemoGetText2 handler=new MemoGetText2();

```

Les API pour XML – p. 45/65

Activation de la validation (2)

```

21 // on passe par SAX
22 XMLReader reader=parser.getXMLReader();
23 reader.setContentHandler(handler);
24 // mise en place du gestionnaire d'erreur
25 reader.setErrorHandler(new PrintError());
26 try {
27     // analyse (SAX)
28     reader.parse(args[0]);
29     System.out.println(handler.getText());
30 } catch (SAXException se) {
31     se.printStackTrace();
32 } catch (IOException ioe) {
33     ioe.printStackTrace();
34 }
35 } catch (Exception e) {
36     System.err.println(e);
37 }
38 }
39 }

```

Les API pour XML – p. 46/65

Activation de la validation (3)

L'analyse de another-memo.xml donne :

```

1 error: org.xml.sax.SAXParseException: Document is invalid: no grammar found.
2 error: org.xml.sax.SAXParseException: Document root element "mémo", must
3 match DOCTYPE root "null".
4 Ca va ?
5 Je teste le mémo.
6

```

En jouant avec le ErrorHandler et les exceptions, on peut éviter de traiter un document non valide. Limitations :

- pas de support direct des schémas dans SAX et JAXP (1.1)
- pour les schémas W3C, on doit passer par des *features* SAX spécifiques à Xerces (par exemple) ou par JAXP 1.2
- pour RELAX NG, on utilise un validateur externe

Les API pour XML – p. 47/65

Gestion des attributs

- très facile grâce à Attributes
- permet de parcourir la liste des attributs :
 - grâce à la position
 - directement grâce au nom de l'attribut
- exemple :

```

1 import org.xml.sax.helpers.DefaultHandler;
2 import org.xml.sax.Attributes;
3 public class PrintAttributes extends DefaultHandler {
4     public void startElement(String namespaceURI, String localName,
5                             String qName, Attributes atts) {
6         System.out.println(qName);
7         for(int i=0;i<atts.getLength();i++) {
8             System.out.println("\t"+atts.getQName(i)+"="+atts.getValue(i));
9         }
10        String bozo=atts.getValue("bozo");
11        if(bozo!=null) System.out.println("\tBozo : "+bozo);
12    }
13 }

```

Les API pour XML – p. 48/65

Exemple

L'application à

atts.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <a val="12">
3   <b/><c truc="bidude"/>
4   <c bozo="hop"/><d x="1" y="2"/>
5 </a>
```

donne :

```
1 a
2   val=12
3 b
4 c
5   truc=bidude
6 c
7   bozo=hop
8   Bozo : hop
9 d
10  x=1
11  y=2
```

Les API pour XML - p. 49/65

Résumé

SAX n'est pas toujours facile d'emploi :

- l'accès est séquentiel
- l'API n'a aucune mémoire : impossible de connaître le contexte d'un évènement

Il faut :

- avoir un modèle de l'analyse (par exemple sous forme d'un automate)
- gérer à la main la mémorisation des informations importantes
- en particulier le contexte (la position dans l'arbre)

Les API pour XML - p. 51/65

Autres possibilités de SAX

- EntityResolver : gestion des entités XML
- DTDHandler : support basique des DTDs
- DeclHandler : support additionnel pour les DTDs (extension de SAX)
- LexicalHandler : support des commentaires et autres détails (extension)
- XMLFilter :
 - comme un XMLReader, mais "lit" un XMLReader au lieu d'un fichier
 - permet de fabriquer une chaîne de traitements

Les API pour XML - p. 50/65

API

DOM

Les API pour XML - p. 52/65

DOM

Document Object Model :

- recommandation du W3C
- <http://www.w3.org/DOM/>
- s'applique à XML et à HTML (utilisé en particulier dans les navigateurs avec Javascript)
- Level 1 : 1 octobre 98, Level 2 (XML) : 13 novembre 2000, Level 3 : *working draft*, devrait sortir dans le courant de l'année 2003
- nombreuses implantations (par exemple Xerces, open source)

Langage de référence : IDL (Interface Definition Language, CORBA). Il existe des *bindings* dans de très nombreux langages (Java, Javascript, C, C++, Perl, Python, VB, etc.).

Caractéristiques majeures

- modèle objet (arbre)
 - haut niveau
 - très riche (nombreux outils intégrés, surtout depuis le Level 2)
 - coûteux : document entièrement en mémoire
 - parfois un peu lourd à programmer
 - en général construit au dessus de SAX : un ContentHandler spécifique transforme le flux d'évènements en un arbre (exemple : Xerces)
- Attention** : pour de gros documents XML, DOM est pratiquement impossible à utiliser (la définition de gros évolue avec les capacités des ordinateurs !)

Les API pour XML - p. 53/65

Modèle par arbre

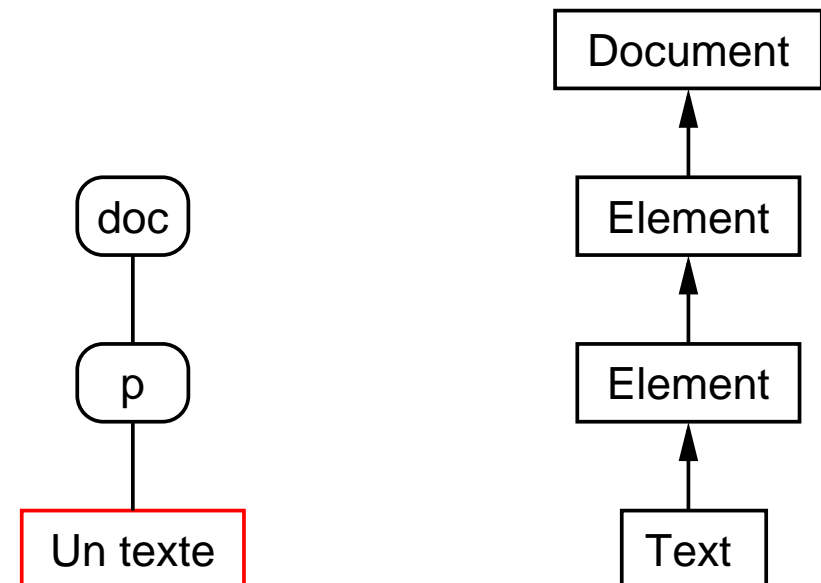
Principes :

- un document est transformé en un objet Document
- un Document possède quelques fils :
 - un Element : la racine du document
 - divers éléments *top level* : commentaires, *processing instructions* et DTD (pour l'accès aux entités)
- la structure d'arbre est obtenue grâce aux Elements :
 - un Element possède éventuellement des fils Elements ou Text
 - un Element possède éventuellement des attributs (Attr)
- DOM définit toutes les interfaces de manipulations de l'arbre et de son contenu

Les API pour XML - p. 55/65

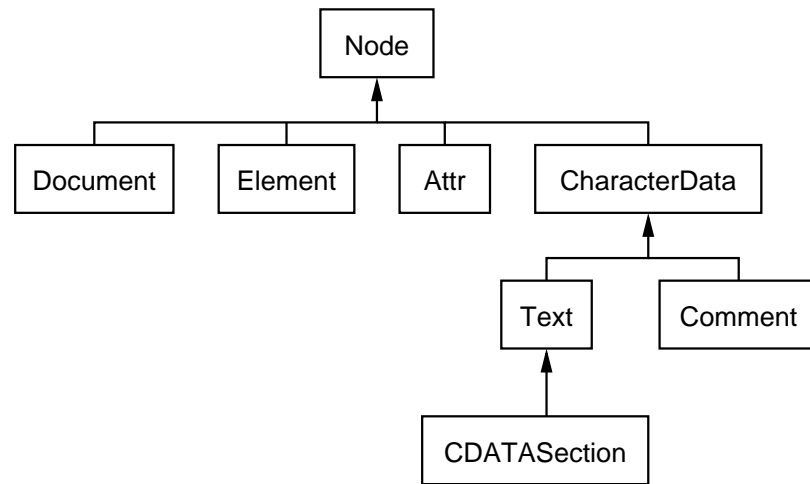
Les API pour XML - p. 54/65

Modèle en mémoire



Les API pour XML - p. 56/65

Diagramme des interfaces (partiel)



Les API pour XML – p. 57/65

Utilisation

Modèle d'une application DOM :

1. création d'un parseur DOM
2. (in)activation des options du parseur
3. obtention de l'objet Document
4. manipulations du document

Amorce de traitement récursif :

dom/java/Amorce

```
1 Document doc=...;
2 Node root=doc.getDocumentElement();
3 root.normalize();
4 traitement(root);
```

Les API pour XML – p. 59/65

Remarques

- il y a d'autres classes (DocumentType, EntityReference, etc.)
- l'héritage est "déroutant" :
 - certains méthodes de Node n'ont pas vraiment de sens à ce niveau :
 - getNodeName : nom de l'élément (pas de sens pour du texte)
 - getNodeValue : contenu du nœud (étrange pour un élément)
 - le typage se fait par une constante de type short :
getNodeType
- implantations astucieuses :
 - développement paresseux de l'arbre
 - modèle interne léger (*Document Table Model* de Xalan)

Les API pour XML – p. 58/65

Problème de portabilité

DOM n'est pas assez complet, ce qui oblige à utiliser des extensions spécifiques à chaque implantation :

- DOM niveau 1 : pas de création du *parseur*, pas de gestion des options, pas d'obtention d'un Document
- DOM niveau 2 : création d'un Document vide (pas de lecture, pas de sauvegarde, etc.)
- DOM niveau 3 (enfin !) : API spécifique *Load and Save* qui règle les problèmes de portabilité (*draft* supporté par Xerces)
- JAXP 1.1 : règle les problèmes de portabilité en Java

Les API pour XML – p. 60/65

Exemple récursif

```
dom/java/PrintTextRécursif
1 public static void print(Node node) {
2     switch(node.getNodeType()) {
3     case Node.ELEMENT_NODE:
4         NodeList children=node.getChildNodes();
5         for(int i=0;i<children.getLength();i++) {
6             print(children.item(i));
7         }
8         break;
9     case Node.TEXT_NODE:
10        String text=node.getNodeValue().trim();
11        if(text.length()>0) {
12            System.out.println(text);
13        }
14        break;
15    default:
16        // rien à faire d'autre
17    }
18 }
```

Les API pour XML – p. 61/65

Exemple itératif

```
dom/java/PrintTextItératif
1 public static void printItératif(Node node) {
2     final int DOWN=1;
3     final int UP=2;
4     int status=DOWN;
5     while(node!=null) {
6         if(node.getNodeType()==Node.TEXT_NODE) {
7             String text=node.getNodeValue().trim();
8             if(text.length()>0)
9                 System.out.println(text);
10        }
11        Node next;
12        if(status==DOWN) {
13            next=node.getFirstChild();
14            if(next!=null) {
15                node=next;
16                continue;
17            }
18        }
```

Les API pour XML – p. 62/65

Exemple itératif (suite)

```
dom/java/PrintTextItératif2
1     next=node.getNextSibling();
2     if(next!=null) {
3         node=next;
4         status=DOWN;
5     } else {
6         node=node.getParentNode();
7         status=UP;
8     }
9 }
10 }
```

Les API pour XML – p. 63/65

Outils de haut niveau

DOM Level 2 apporte des outils de haut niveau :

- *Traversal* :
 - API de parcours de l'arbre
 - filtre sur les noeuds NodeFilter
 - itération NodeIterator
 - parcours complet TreeWalker
- *Range* : modèle de sélection dans un arbre DOM
- *Events* : évènements de modification d'un arbre DOM
- etc.

Les API pour XML – p. 64/65

Exemple

dom/java/PrintTraversal

```
1 public static void printTraversal(Node node, Document doc) {
2     NodeIterator iter=
3         ((DocumentTraversal)doc).createNodeIterator(
4             node, NodeFilter.SHOW_TEXT, null, true);
5     node=iter.nextNode();
6     while (node!=null) {
7         String text=node.getNodeValue().trim();
8         if(text.length()>0) {
9             System.out.println(text);
10        }
11        node=iter.nextNode();
12    }
13 }
```