Services Web - Révision et compléments XML

E.Coquery

emmanuel.coquery@liris.cnrs.fr

http://www710.univ-lyon1.fr/~ecoquery/

XML

- eXtensible Markup Language
- Standard W3C issu du SGML
- Modèle de données
  - Structure arborescente
  - Informations textuelles
- Utilisé pour
  - documents
  - échanges de données (services Web)
  - bases de données semi-structurées

... personne n'a envie d'écrire du XML à la main

Plan

Un document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
  <personne nom="Toto">
    <adresse>8 bvd N.Bohr, Villeurbanne</adresse>
    <telephone numero='0456789012'/>
  </personne>
</carnet>
```

Un document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
  <personne nom="Toto">
    <adresse>8 bvd N.Bohr, Villeurbanne</adresse>
    <telephone numero='0456789012'/>
  </personne>
</carnet>
```

prologue

Un document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
  <personne nom="Toto">
    <adresse>8 bvd N.Bohr, Villeurbanne</adresse>
    <telephone numero='0456789012'/>
  </personne>
</carnet>
```

élément

Un document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
  <personne nom="Toto">
    <adresse>8 bvd N.Bohr, Villeurbanne</adresse>
    <telephone numero='0456789012'/>
  </personne>
</carnet>
```

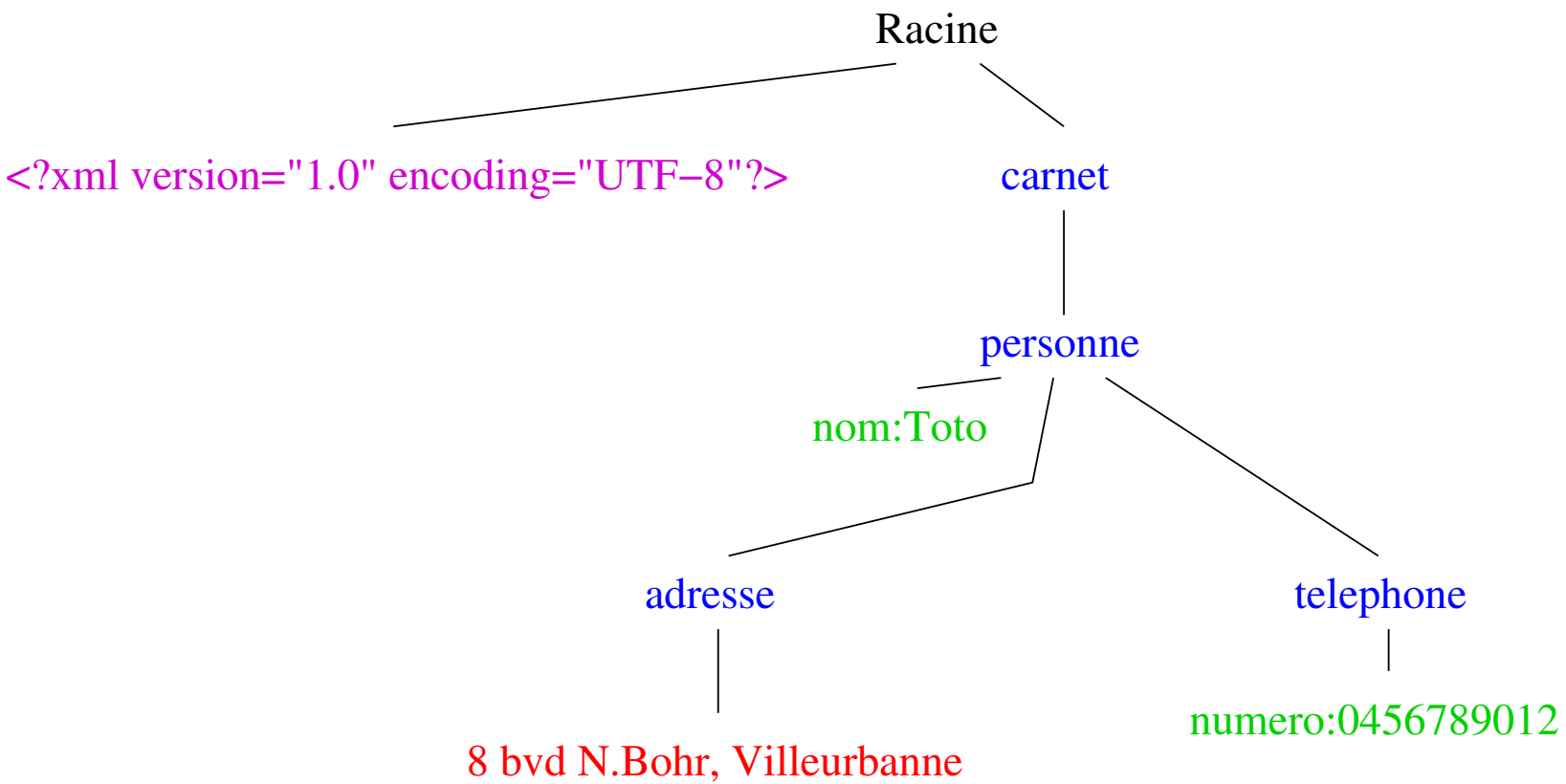
attribut

Un document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
  <personne nom="Toto">
    <adresse>8 bvd N.Bohr, Villeurbanne</adresse>
    <telephone numero='0456789012'/>
  </personne>
</carnet>
```

texte

Arbre correspondant (Infoset)



prologue élément attribut texte

Particularités syntaxiques

<![CDATA[*texte brut*]]>

<!-- Commentaire -->

<? Commande à interpréter par l'application ?>

<	<
>	>
'	'
"	&quote;
&	&
caractere	&#code;

DTD

Spécification de la structure d'un document

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE carnet SYSTEM "carnet.dtd">
<carnet>
  <personne nom="Toto">
    <adresse>8 bvd N.Bohr, Villeurbanne</adresse>
    <telephone numero='0456789012'/>
  </personne>
</carnet>
```

Un document est valide s'il respecte la structure déclarée

DTD

carnet.dtd :

```
<!ELEMENT carnet (personne +)>
<!ELEMENT personne (adresse,telephone)>
<!ATTLIST personne nom CDATA #REQUIRED>
<!ELEMENT adresse (#PCDATA)>
<!ELEMENT telephone EMPTY>
<!ATTLIST telephone numero CDATA #REQUIRED>
```

DTD

carnet.dtd :

```
<!ELEMENT carnet (personne +)>
<!ELEMENT personne (adresse,telephone)>
<!ATTLIST personne nom CDATA #REQUIRED>
<!ELEMENT adresse (#PCDATA)>
<!ELEMENT telephone EMPTY>
<!ATTLIST telephone numero CDATA #REQUIRED>
```

expression régulière de nom d'éléments
#PCDATA $\leftrightarrow$ texte
EMPTY $\leftrightarrow$ pas d'enfants

DTD

carnet.dtd :

```
<!ELEMENT carnet (personne +)>
<!ELEMENT personne (adresse,telephone)>
<!ATTLIST personne nom CDATA #REQUIRED>
<!ELEMENT adresse (#PCDATA)>
<!ELEMENT telephone EMPTY>
<!ATTLIST telephone numero CDATA #REQUIRED>
```

CDATA	chaîne de caractères
$(val_1, val_2, \ldots)$	ensemble de valeurs possibles
ID	identifiant
IDREF	reférence à un identifiant
IDREFS	plusieurs références

DTD

carnet.dtd :

```
<!ELEMENT carnet (personne +)>
<!ELEMENT personne (adresse,telephone)>
<!ATTLIST personne nom CDATA #REQUIRED>
<!ELEMENT adresse (#PCDATA)>
<!ELEMENT telephone EMPTY>
<!ATTLIST telephone numero CDATA #REQUIRED>
```

"valeur"	valeur par défaut
#IMPLIED	optionnel
#REQUIRED	obligatoire
#FIXED "valeur"	valeur forcée (même si absent)

DTD intégrée

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE carnet [
<!ELEMENT carnet (personne +)>
<!ELEMENT personne (adresse,telephone)>
<!ATTLIST personne nom CDATA #REQUIRED>
<!ELEMENT adresse (#PCDATA)>
<!ELEMENT telephone EMPTY>
<!ATTLIST telephone numero CDATA #REQUIRED>
]>

<carnet>
  <personne nom="Toto">
    <adresse>8 bvd N.Bohr, Villeurbanne</adresse>
    <telephone numero='0456789012'/>
  </personne>
</carnet>
```

Plan

Noms de modules pour XML

En informatique on a souvent des conflits de noms

- ex : une procédure `init()`

En programmation : utilisation de “modules”

- ex : les packages Java, les modules Python, ...

Idée générale

- le nom utilisé est un nom courant,
- le vrai nom est le nom du module + le nom courant
  $\Rightarrow$ le nom du module sert à lever l'ambiguïté

Les espaces de nommage (namespaces) sont les noms de modules de XML.

URIs

Les namespaces sont des URIs (Uniform Resource Identifier) :

- soit des URLs (Uniform Resource Location) :
  - ex : `http//example.com/toto`
- soit des URNs (Uniform Resource Name) :
  - ex : `urnpersonnagestoto`

Pas de résolution des URLs (on ne cherche pas forcément à y accéder)

Utilisation des namespaces

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet xmlns="http://www.carnets.com"
 xmlns:infos="urn:informations:personnes">
  <personne infos:nom="Toto">
    <adresse xmlns="http://www.mailing.com">8 bvd
                          N.Bohr, Villeurbanne</adresse>
    <infos:telephone numero='0456789012'/>
  </personne>
</carnet>
```

http://www.carnets.com
urn:informations:personnes
http://www.mailing.com

Plan

Spécifications, encore

Défauts des DTDs :

- Pas de gestion des espaces de nommage
- Un élément ne peut avoir qu'un seul modèle de contenu
  - ex : personne dans un carnet vs personne à l'université
- Les types de données de base sont limités
  - CDATA, ID, IDREF ...
  - Pas d'entiers, de flottants, de dates...

$\Rightarrow$ XML-Schema

- Syntaxe XML lourde

Entête XML-Schema

```
<schema xmlns="http://www.w3.org/2001/XML-Schema"
 xmlns:toto="http://example.com/toto"
 xmlns:xs="http://www.w3.org/2001/XML-Schema"
 targetNameSpace="http://example.com/toto"
 elementFormDefault="qualified">


...


</schema>
```

Simple types

Types des données.

Types prédéfinis :

- string
- boolean, integer, positiveInteger, float, ...
- time, date
- ...

voir `http://www.w3.org/TR/xmlschema11-2/`

Possibilité de définir ses propres types simples

Simple types : Restriction

```
<simpleType name="codeBarre">
  <restriction base="string">
    <pattern value="[0-9]{6} [0-9]{6}"/>
  </restriction>
</simpleType>
```

- expressions régulières
- taille
- énumération
- valeur min/max

Simple types : Union

```
<simpleType name="codeOuRef">
 <union>
  <simpleType>
   <restriction base="positiveInteger">
    <maxInclusive=’65535’/>
   </restriction>
  </simpleType>
  <simpleType name="codeBarre"/>
 </union>
</simpleType>
```

Simple types : Listes

```
<simpleType name="listOfInt">
  <list itemType='integer'/> </simpleType>
```

`listOfInt` $\leftrightarrow$ liste d'entiers séparés par des espaces

Éléments

```
<element name="un-nom" type="un-type"/>
```

- un-nom : nom de l'élément
- un-type : type de l'élément.
  - Peut également être spécifié par un enfant de `element`
  - Peut être un type simple ($\leftrightarrow$ `#PCDATA`)
  - Peut être un type complexe (attributs + enfants)

```
<element name="adresse" type="xs:string"/>
```

Types complexes

Spécification d'enfants et d'attributs

```
<complexType name="nom-du-type">
...
</complexType>
```

`<sequence>`	suite d'élements	,
`<choice>`	choix	|
`<element name="nom" ...>`	élément	nom
`<attribute name="nom" type="type">`	attribut	

Attributs `minOccurs` et `maxOccurs` :

- permettent de coder ? * + {n} et {m,n}
- valeur par défaut : 1
- unbounded $\leftrightarrow$ infini

Exemple

```
<complexType name="contenuCarnet">
 <sequence>
  <element name="personne" maxOccurs="unbounded">
   <complexType>
    <sequence>
     <element name="adresse" type="xs:string"/>
     <element name="telephone">
      <complexType>
       <attribute name="numero" type="toto:numTel"/>
      </complexType>
     </element>
    </sequence>
    <attribute name="nom" type="xs:string"/>
   </complexType>
  </element>
 </sequence>
</complexType>
```

Plan

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 mechanismes 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 lorqu'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"));
```

Plan

XPath

- Langage pour sélectionner des noeuds dans un arbre XML
- Basé sur une notion de chemin
  - Une expression (de chemin) XPath $\leftrightarrow$
    forme d'un chemin à suivre pour arriver au noeud voulu
  - Composée d'étapes séparée par des /
  - Une étape contient :
    - Un axe (direction générale)
    - Un test (type, éventuellement nom, de noeud)
    - Un prédicat (expression booléenne pour préciser encore la sélection)
    - .../axe : :test[predicat]/...
  - Si l'expression commence par un /, on évalue depuis la racine, sinon depuis le noeud courant

Exemple

```
child::personne[attribute::nom = "Toto"] /
                                    following-sibling::element()
```

- parmi les enfants,
- prendre les éléments "personne"
- ayant un attribut "nom" dont la valeur est "Toto"
- puis aller vers les noeuds "frères" suivants
- prendre n'importe quel élément

Axes (vers l'avant)

`child::` Enfant de l'élément sélectionné

`descendant::` Descendant
(i.e. on peut descendre autant que l'on veut)

`attribute::` Attribut de l'élément actuellement sélectionné

`self::` Sélection du noeud courant : pas de déplacement.

`descendant-or-self::` Auto explicatif (i.e. descendant ou self)

`following-sibling::` Noeuds au même niveau que le noeud courant
et après le noeud courant.

`following::` Noeuds après le noeud courant, mais pas forcément au même niveau.

Rien équivalent à `child::`

`@` équivalent à `attribute::`

Axes (vers l'arrière)

`parent::` Noeud parent du noeud courant

`ancestor::` Noeuds ancêtres du noeud courant

`preceding-sibling::` Noeuds avant le noeud courant, et au même niveau

`preceding::` Noeuds avant le noeud courant, pas forcément au même niveau

`ancestor-or-self::` Auto explicatif

Tests

nom	Sélectionne les élément dont le nom est nom
`*`	N'importe quel élément, mais pas le texte
`element()`	Un élément. Le nom peut être donné entre parenthèses.
`node()`	N'importe quel noeud, y compris le texte et les attributs.
`text()`	Du texte
`attribute()`	Un attribut. Le nom peut être donné entre parenthèses
`document-node()`	La racine du document
`processing-instruction()`	Une commande
`comment()`	Un commentaire
`//`	équivalent à `descendant-or-self::node()`

Prédicats

- `=`, `<`, `>`, `or`, `and` ...
- entier $n \rightarrow$ vrai si le noeud est le n^{ieme} dans les noeuds sélectionnés par le test
- *chemin* $\rightarrow$ vrai l'ensemble des noeuds correspondant à *chemin* n'est pas vide
- `chemin=`*expr* $\rightarrow$ vrai si parmi les noeuds sélectionnés par *chemin*, il y en a 1 égal à *expr*

Exemples

```
<bib>
  <book title="a" />
  <book title="b">
   <critique>
     C'est un livre tres interessant
    </critique>
  </book>
</bib>
```

```
child::bib
```

Exemples

```
<bib>
  <book title="a" />
  <book title="b">
   <critique>
     C'est un livre tres interessant
    </critique>
  </book>
</bib>
```

```
child::bib/child::book
```

Exemples

```
<bib>
  <book title="a" />
  <book title="b">
   <critique>
     C'est un livre tres interessant
   </critique>
  </book>
</bib>
```

```
//child::book/attribute::title
```

Exemples

```
<bib>
  <book title="a" />
  <book title="b">
   <critique>
     C'est un livre tres interessant
   </critique>
  </book>
</bib>
```

```
//self::critique/child::text()
```

Exemples

```
<bib>
   <book  title="a"  year="1999" />
   <book  title="b" />
</bib>
```

```
bib/book[@year]
```

Exemples

```
<bib>
   <book  title="a"  year="1999" />
   <book  title="b" />
</bib>
```

```
bib/book[@title = 'b']
```

Exemples

```
<bib>
   <book  title="a"  year="1999" />
   <book  title="b" />
</bib>
```

```
bib/book[2]
```

Exemples

```
<bib>
   <book  title="a"  year="1999" />
   <book  title="b" />
</bib>
```

```
bib/book[2]/attribute::*
```

Plan

XSLT

- Langage de transformation de documents XML
- Basé sur une notion de “template”
  - Une expression XPath détermine sur quoi un template peut s’appliquer
  - Le moteur XSLT choisit le template qui correspond le plus précisement à un élément
- Syntaxe XML :
  - Le programme est un document XML
  - Les “tags-clés” (instructions, contrôles, etc) XSLT sont dans le namespace `http://www.w3.org/1999/XSL/Transform`
  - Les tags “non clés” sont écrits dans le document résultat

Programme XSLT

Début de programme pour générer du XHTML :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xhtml="http://www.w3.org/1999/xhtml"
      xmlns="http://www.w3.org/1999/xhtml">
   <xsl:output
         method="xml"
         encoding = "utf-8" />

 ...

</xsl:stylesheet>
```

Template

```
<xsl:template match="carnet">

...

  <xsl:apply-templates/>
...

</xsl:template>
```

- attribut `match` : expression XPath utilisée pour déterminer les noeuds sur lesquels le template peut s'appliquer
- `xsl:apply-templates` : indique qu'il faut essayer d'appliquer un template sur chaque enfant du noeud courant
  - attribut `select` : expression XPath indiquant sur quoi appliquer les templates

Quelques instructions

- `<xsl:variable name="`*nom*`">`
  valeur
  `</xsl:variable>`
  Les variables sont alors accessibles dans les expressions XPath par $nom
- `<xsl:value-of select="`*xpath*`"/>`
  Insère la valeur de l'expression XPath
- `<xsl:copy-of select="`*xpath*`"/>`
  Insère une copie des noeuds sélectionnés par l'expression XPath

Quelques instructions (2)

- <xsl:for-each select="*xpath*">
  instructions
  </xsl:for-each>
- <xsl:if test="*xpath*">
  instructions
  </xsl:if>
- <xsl:choose>
  <xsl:when test="*xpath*">...</xsl:when>
  ...
  <xsl:otherwise>...</xsl:otherwise>
  </xsl:choose>

Exemple

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
    <html><head><title>Carnet</title></head>
      <body><h1>Carnet</h1>
        <xsl:apply-templates select="carnet/personne" />
      </body>
    </html>
  </xsl:template>

  <xsl:template match="personne">
    <p><b><xsl:value-of select="string(@nom)" /></b>
      (<xsl:value-of select="string(telephone/@numero)" />):
      <xsl:copy-of select="adresse/text()" />
    </p>
  </xsl:template>
</xsl:stylesheet>
```

Source

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
  <personne nom="Toto">
    <adresse>8 bvd N.Bohr, Villeurbanne</adresse>
    <telephone numero='0456789012'/>
  </personne>
</carnet>
```

Résultat

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Carnet</title></head>
  <body><h1>Carnet</h1>
    <p><b>Toto</b>
      (0456789012):
      8 bvd N.Bohr, Villeurbanne
    </p>
  </body>
</html>
```

TrAX

- Transformation API for XML (Java)
- Exécution d'un programme XSLT en Java
- Source
  - Représente un document à traite ou du code xsl
  - DOMSource, SAXSource, StAXSource, StreamSource
- Result
  - Représente "l'endroit" ou 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));
}
```