XML

S. Rosmorduc, J.-M. Douin, J.-L. Dewez

GLG 203, 2016

Bibliographie

- E. R. Harold et W. S. Means, « XML in a Nutshell, 3rd Edition », O'Reilly, 2004

- Site du W3C

Pourquoi XML

Les documents structurés

```
#
# Real world MySQL Cluster configuration suited
# to be run on a developer machine
#
!include include/default_mysqld.cnf

[cluster_config.mysqld.1.1]
BatchSize=512
BatchByteSize=1024K

[mysqld]
# Make all mysqlds use cluster
ndbcluster

ndb-cluster-connection-pool=1
ndb-force-send=1
ndb-use-exact-count=0
ndb-extra-logging=1
ndb-autoincrement-prefetch-sz=256
engine-condition-pushdown=1
ndb-wait-connected=600
ndb-wait-setup=300
```

Exemple de fichier structuré configuration de mysql

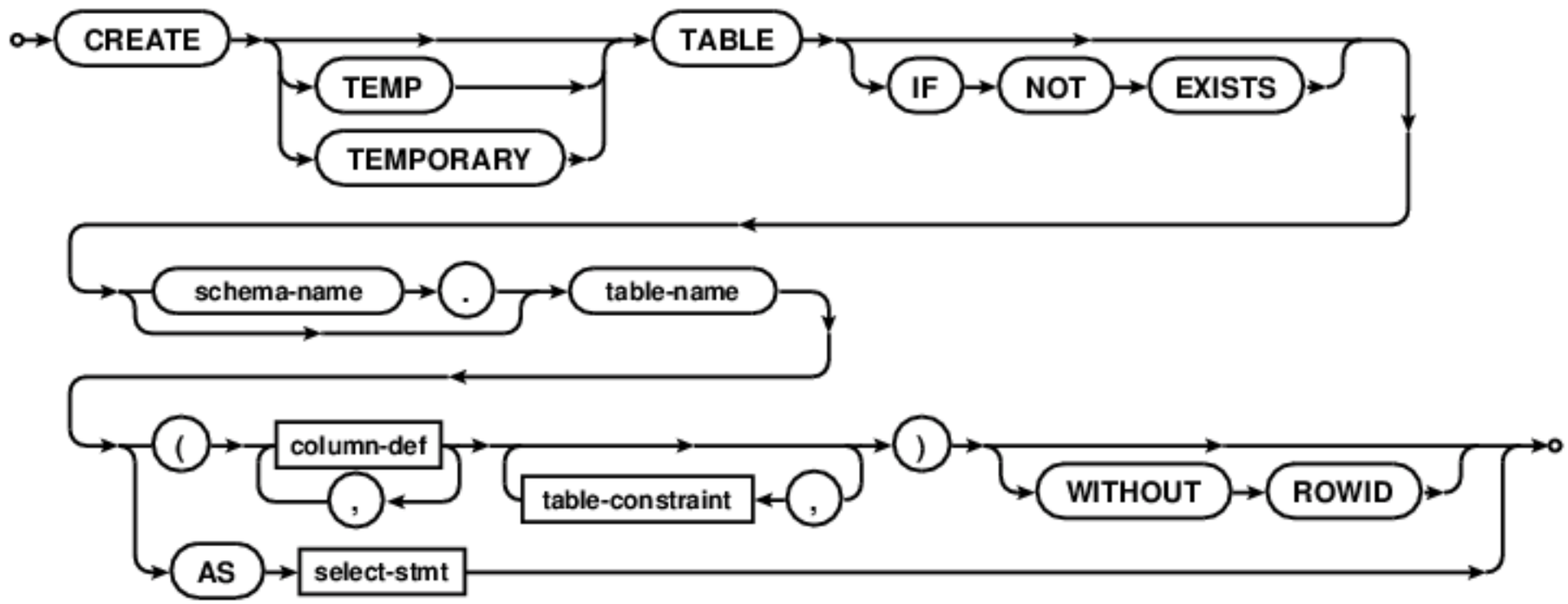
Exemple : fichier RTF

{\rtf1\ansi\ansicpg1252\cocoartf1265\cocoasubrtf210
{\fonttbl\f0\fswiss\fcharset0 Helvetica;}
{\colortbl;\red255\green255\blue255;}
\f0\b\fs24 \cf0 El deldischado\
\pard\tx566\tx1133\tx1700\tx2267\tx2834\tx3401\tx3968\tx4535\tx5102\tx5669\tx623
6\tx6803\pardirnatural
6803\li1540\fi340\pardirnatural
\cf0 Je suis le T\'e9n\'e9breux, - le Veuf, l\'92Inconsol\'e9,\
\pard\tx1133\tx1700\tx1906\tx2267\tx2834\tx3401\tx3968\tx4535\tx5102\tx5669\tx62
36\tx6803\li1540\fi360\pardirnatural
\cf0 Le prince d\'92Aquitainne \'e0 la Tour abolie :\
}

Grammaire formelle

- Exemple : diagramme de syntaxe

create-table-stmt:



Used by: sql-stmt

References: column-def select-stmt table-constraint

See also: lang_createtable.html

Grammaire formelle (ANTLR pour Java)

```
forControl
    :   enhancedForControl
    |   forInit? ';' expression? ';' forUpdate?
    ;

forInit
    :   localVariableDeclaration
    |   expressionList
    ;

enhancedForControl
    :   variableModifier* type variableDeclaratorId ':' expression
    ;

forUpdate
    :   expressionList
    ;
```

Grammaire
(de XML, site du W3C)

[39] element ::= EmptyElemTag
| STag content ETag [WFC: Element Type Match]
[VC: Element Valid]

[40] STag ::= '<' Name (S Attribute)* S? '>' [WFC: Unique Att Spec]
[41] Attribute ::= Name Eq AttValue [VC: Attribute Value Type]
[WFC: No External Entity References]
[WFC: No < in Attribute Values]

Problèmes

- Il existe des outils pour manipuler les grammaires formelles : YACC, CUP, ANTLR…
- … mais l'écriture d'une grammaire formelle est complexe
- il faut éviter les grammaires ambiguës
- l'écriture de la grammaire pour un outil donné suppose qu'elle soit conforme à un certain modèle (LR1, LALR…)

SGML

- « Standard Generalized Markup Language »
- 1986 (mais GML 1969)
- langage prévu pour l'écriture de *documentations* structurées
- permet de créer son propre système de balises à l'aide d'une DTD
- très souple ; un seul analyseur suffit pour tous les systèmes de balisage
- l'analyseur est complexe

XML

- Simplification de SGML (DTDs non compatibles)
- 1998
- Peut utiliser Unicode
- La structure du document peut être connue même si on ignore sa DTD
- Analyseur (assez) simple à écrire

Un exemple

```
<personne id="e34">
    <voir ref="e44"/>
    <nom>Toto</nom>
    <prenom>Hubert</prenom>
</personne>
```

XML vs HTML

- HTML :
  - à l'origine, suit une DTD **SGML**
  - certaines balises fermantes ne sont pas obligatoires (</p>)
- XHTML : DTD XML pour HTML
  - dans ce cadre, XML est un mécanisme général de description de documents
  - et XHTML est une utilisation particulière de ce formalisme avec une description particulière
- HTML5 : spécification qui abandonne la notion de DTD

Quelques exemples d'utilisation de XML

Textes structurés

- La TEI « Text Encoding Initiative»
- http://wiki.tei-c.org/index.php/Samples_of_TEI_texts
- Jeu modulaire de balises pour annoter des textes

```
<persName type="divine">
<name reg="Ἀφροδίτη">
<supplied reason="lost">Ἀφροδείτης</supplied>
</name>
</persName>
</ab>
</div>
<div type="apparatus">
<head>Apparatus</head>
<p>
For the supplements, compare the partner inscription
<xref type="inscription" n="1581" href="010038">1.38</xref>
.
</p>
</div>
<div type="translation">
<head>Translation</head>
<p>
[?This area is] the sacred asylum [?as defined by] the great [?Caesar, the] Dictator, and [?his son] Imperator [Caesar and
the ] Senate [and People] of Rome, [as is also contained in the] grants of privilege, the public documents [and decrees. C.
Iulius Zoilos priest of Aphrodite set up the boundary stones.]
</p>
</div>
```

Textes structurés

```
<persName type="divine">
<name reg="Ἀφροδίτη">
<supplied reason="lost">Ἀφροδείτης</supplied>
</name>
</persName>
</div>
<div type="apparatus">
<head>Apparatus</head>
<p>
For the supplements, compare the partner inscription
<xref type="inscription" n="1581" href="010038">1.38</xref>
</p>
</div>
<div type="translation">
<head>Translation</head>
<p>
[?This area is] the sacred asylum [?as defined by] the
great [?Caesar, the] Dictator, and [?his son] Imperator
[Caesar and the ] Senate [and People] of Rome, [as is also
contained in the] grants of privilege, the public documents
[and decrees. C. Iulius Zoilos priest of Aphrodite set up
the boundary stones.]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.qenherkhopeshef</groupId>
  <artifactId>JSesh-all</artifactId>
  <version>6.5.3</version>
  <packaging>pom</packaging>
  <name>JSesh complete distribution</name>
<build>
  <plugins><plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.0</version>
  <configuration>
    <source>1.6</source>
    <target>1.6</target>
     <encoding>UTF-8</encoding>
  </configuration>
      </plugin></plugins>
</build>
</project>
```

Fichiers de configuration

Format d'échange de données

- AJA**X**

- Protocole SOAP : les requêtes et les informations sont communiquées sous forme de données XML

- de plus en plus concurrencé par JSON, YAML

Récapitulation

- redondant, auto-descriptif
- simple à mettre en œuvre
- utilisation d'outils génériques
- ... très verbeux

Le format XML

- Formé à base de balises <balise>…</balise> parenthésées

- La *grammaire* du document peut être décrite par une **DTD (document type definition)**

- Avantage par rapport à des formats plus libres :

  - Pas d'ambiguïté possible dans le document (à cause du parenthésage)

  - documents analysables avec des analyseurs génériques

Vocabulaire

```
<personne id="e34">
    <voir ref="e44"/>
    <nom>Toto</nom>
    <prenom>Hubert</prenom>
</personne>
```

- **<personne>** : ***balise*** (tag) ouvrante
- **</personne>** : balise (tag) fermante
- **ref** : **attribut**
- **<personne>…</personne>** : **élément** personne
- **<voir…/>** : élément vide

Document XML ***bien formé***

- Document qui respecte les règles de XML, sans forcément conformer à une grammaire explicite
- en gros: correctement parenthésé
- plus précisément:
  - il y a une unique balise racine
  - les éléments sont toujours fermés et correctement parenthésés (<a>.. et non <a>..)
  - les noms des balises, des attributs vérifient les règles de bonne formation
  - les attributs ont toujours une valeur, donnée entre guillemets simples ou doubles

Noms XML

```
NameStartChar ::=
   ":" | [A-Z] | "_" | [a-z] | [#xC0-#xD6] | [#xD8-#xF6]
   | [#xF8-#x2FF] | [#x370-#x37D] | [#x37F-#x1FFF]
   | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF]
   | [#x3001-#xD7FF] | [#xF900-#xFDCF]
   | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]

NameChar::=
   NameStartChar | "-" | "." | [0-9] | #xB7
| [#x0300-#x036F] | [#x203F-#x2040]

Name ::= NameStartChar (NameChar)*
```

Un document XML typique

```
<?xml version="1.0" encoding="UTF-8"?>
<article>
    <titre>Le langage XML</titre>
    <auteur>Serge Rosmorduc</auteur>
    <chapitre>
        <titre>Introduction</titre>
        bla bla bla...
    </chapitre>
</article>
```

Document XML ***Valides***

- Documents qui sont conforme à une **grammaire** :
  - DTD
  - schéma
  - Relax NG
  - (il y en a d'autres)

Utilisation d'une DTD

- Déclaration DOCTYPE
- <!DOCTYPE html public '-//W3C//DTD HTML 4.0// EN' 'http://www.w3.org/TR/REC-html40/strict.dtd'>
- public : nom prédéfini, on cherche éventuellement dans un catalogue (à configurer)
- seconde partie, optionnelle, URI de la DTD, cherchable éventuellement sur le net

Déclaration de DTD

- <!DOCTYPE adresses SYSTEM 'adresse.dtd'>
- on va chercher la DTD sur le système de fichier.
- peut être une URL

Les DTD

- permettent de spécifier la syntaxe d'un fichier XML
- la syntaxe des DTD n'est pas XML
- définissent en particulier
  - les éléments (les balises et leur contenu)
  - les attributs possibles pour les balises
  - des entités (raccourcis)
- On peut étendre une DTD dans un document XML

!ELEMENT

- Déclare un élément, dont on donne le nom et le contenu

- <!ELEMENT *nom contenu*>

- *contenu* peut être:

  - ANY : tout contenu XML valide

  - EMPTY : contenu vide

  - (#PCDATA) : du texte

!ELEMENT

- élément complexe:
- (ELT) un autre élément
- (ELT1,ELT2…) séquence
- (ELT1|ELT2…) "ou" (disjonction)
- ELT? élément optionnel
- ELT+ : une ou plusieurs fois ELT
- ELT* : 0 ou plusieurs fois ELT
- (…) groupe des solutions

Exemple

- <!ELEMENT article (titre,introduction?,section+)>

Élément mixte

- Quand un élément peut mêler directement des enfants de type élément et du texte, il a forcément la forme
- <!ELEMENT p (#PCDATA|em|….)*>
- →une disjonction commençant par PCDATA
- → avec une * pour la répétition

Définition des attributs

- On définit les attributs d'un élément avec ATTLIST :

```
<!ATTLIST monElement
      attribut1 type1 mode1
      attribut2 type2 mode2
....
>
```

- type
  - CDATA : texte
  - ID ou IDREF: clef ou renvoi (commence par une lettre)
  - NMTOKEN ou NMTOKENS un ou plusieurs identifiants
  - (VAL1|VAL2|VAL3…) : une valeur dans liste
- Mode:
  - "value" : valeur par défaut
  - #REQUIRED : obligatoire
  - #IMPLIED : facultatif
  - #FIXED "valeur" constant

Exemple

- DTD http://java.sun.com/dtd/properties.dtd

```
<!ELEMENT properties ( comment?, entry* ) >
<!ATTLIST properties version CDATA #FIXED "1.0">
<!ELEMENT comment (#PCDATA) >
<!ELEMENT entry (#PCDATA) >
<!ATTLIST entry key CDATA #REQUIRED>
```

Entités paramètres

- permettent la redéfinition d'un morceau de la DTD
- permettent aussi d'éviter des copiers/coller
- par exemple pour les attributs communs à plusieurs éléments

```
<!ENTITY %attrs "id ID #REQUIRED
    label CDATA #IMPLIED"/>

<!ELEMENT article (titre,contenu)>
<!ATTLIST article
    %attrs;
>
```

Entités génériques

- permettent de nommer une chaîne de caractères
- exemple : entités générales dans HTML (é)
- Définition <!ENTITY salut "Bien à vous">
- Utilisation **&salut;**

Modification d'une DTD avec Doctype

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

Espaces de noms

- problème : mêler plusieurs jeux de balises dans le même document sans confondre les balises (et les attributs)
    - exemple : textArea en SVG et HTML (zone d'affichage de texte vs. élément de formulaire)
- Solution : espace de noms
    - similaire à la notion de packages de java

Fonctionnement

- Un namespace est identifié par une **URI**
- mais ça n'est pas pratique ni possible à utiliser dans le XML
- on lie donc des préfixes aux URI pour les utiliser dans le document

```
<?xml version="1.0"?>
<html:html xmlns:html='http://www.w3.org/1999/xhtml'>
  <html:head><html:title>Frobnostication</html:title>
 </html:head>
  <html:body><html:p>Demonstration…</html:p>
  </html:body>
</html:html>
```

Namespace par défaut

- L'utilisation de l'attribut « xmlns », sans lui suffixer de nom local, déclare le namespace par défaut du contenu d'une balise.

- il se met typiquement sur la balise parente du document.

Namespace et attributs

- Un attribut peut être dans un namespace (par exemple **xmlns**, dans nos exemples, et fréquemment **xlink**
- mais : « A default namespace declaration applies to all unprefixed **element** names within its scope. Default namespace declarations **do not apply directly to attribute names;** the interpretation of unprefixed attributes is determined by the element on which they appear. »

Espaces de noms (exemple)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
    "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      xmlns:svg="http://www.w3.org/2000/svg"
      >
   <head><title>SVG et HTML</title></head>
    <body>
        <svg:svg baseProfile="tiny" width="300px" height="200px">
            <svg:title>A textarea</svg:title>
            <svg:desc>A textarea that illustrates the name
                collisions between svg and xhtml</svg:desc>
            <svg:textArea width="200" height="50" />
        </svg:svg>
        <div>
            <textarea rows='10' cols="80" name="text">
               Morceau de formulaire</textarea>
        </div>
    </body>
</html>
```

(d'après http://dev.w3.org/SVG/proposals/svg-html/svg-html-proposal.html)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->

<svg
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:cc="http://creativecommons.org/ns#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns="http://www.w3.org/2000/svg"
   xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
   xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
   width="1800"
   height="1800"
   id="svg3971"
   version="1.1"
   inkscape:version="0.91 r13725"
   sodipodi:docname="demo.svg"
   viewBox="0 0 1800 1800">
```

Exemple : svg d'inkscape

Exemple : svg d'inkscape

```
<metadata id="metadata3976">
   <rdf:RDF>
    <cc:Work rdf:about="">
     <dc:format>image/svg+xml</dc:format>
     <dc:type
       rdf:resource="http://purl.org/dc/dcmitype/StillImage"/>
     <dc:title></dc:title>
    </cc:Work>
   </rdf:RDF>
</metadata>
<g
   inkscape:label="Calque 1"
   inkscape:groupmode="layer"
   id="layer1"
   transform="translate(0,747.63782)">
  <ellipse
     style="color:#000000;" id="path7393"
     cx="919.61847" cy="147.45752"
    rx="812.33826" ry="454.30008" />
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->

<svg
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:cc="http://creativecommons.org/ns#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns="http://www.w3.org/2000/svg"
   xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
   xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
   width="1800"
   height="1800"
   id="svg3971"
   version="1.1"
   inkscape:version="0.91 r13725"
   sodipodi:docname="demo.svg"
   viewBox="0 0 1800 1800">
```

Exemple : svg d'inkscape

Espaces de noms et DTD

- Les namespaces n'existaient pas quand les DTD furent créés

- On peut les gérer avec des DTD, mais c'est un peu acrobatique, en particulier pour modifier le préfixe

- typiquement, soit on les utilise avec des documents bien formés, soit on les utilise avec des **schémas** (voir plus loin).

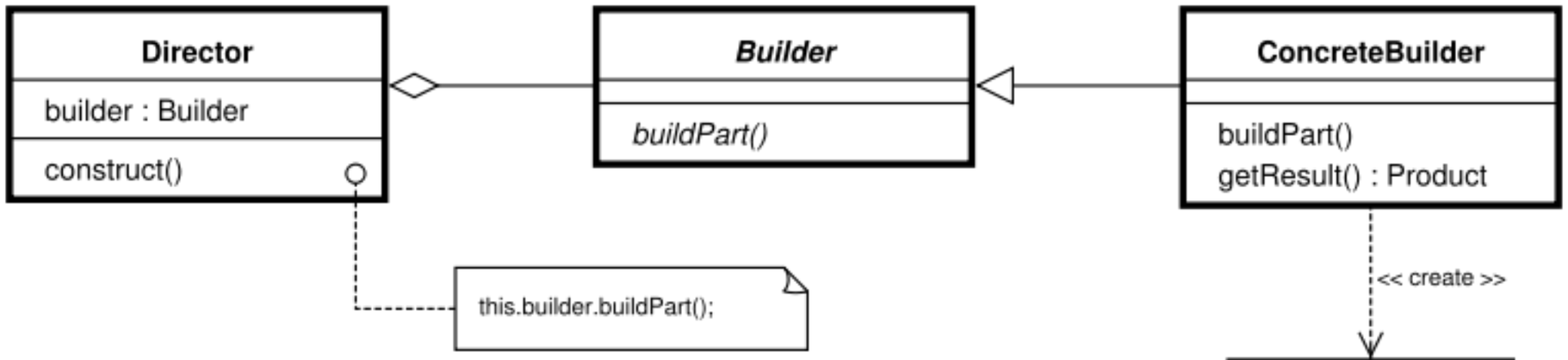
Outils logiciels

SAX
Simple API for XML

- Pourquoi : lire un fichier XML

- Comment : lecture « événementielle » (pattern « builder »

https://en.wikipedia.org/wiki/Builder_pattern



https://en.wikipedia.org/wiki/Builder_pattern

SAX : fonctionnement

- On crée un SAXParser, qui va lire le fichier XML
- On fournit à ce parser un builder qui implémente l'interface ContentHandler
- Pour chaque balise ou texte lu, le parser va appeler la méthode correspondante de ContentHandler
- le parser peut vérifier que le document est conforme à une DTD ou à un schéma
- analyse lexicale améliorée : le travail sur **la structure** est à la charge du programmeur

Sax : configuration

- Le parser est créé par le SAXParserFactory, qui est un singleton
- on configure le futur parser en appelant des méthodes sur le SAXParserFactory
- on crée le parser avec newSAXParser()
- On lui fournit le handler lors de l'analyse de la source.

```
TestXML handler= new TestXML();
SAXParserFactory factory= SAXParserFactory.newInstance();
factory.setValidating(false);
SAXParser parser= factory.newSAXParser();
InputSource src= new InputSource(i);
src.setSystemId("file:///");
parser.parse(src, handler);
```

SAX : configuration

- sur l'objet SAXParserFactory :
  - setValidating(boolean): le parser doit-il valider par rapport à un catalogue de DTD ?
  - setFeature(String uri, boolean value)
    - voir http://www.saxproject.org/apidoc/org/xml/sax/package-summary.html#package_description pour la liste des propriétés et features

```
SAXParserFactory factory= SAXParserFactory.newInstance();
factory.setFeature("http://xml.org/sax/features/namespaces", true);
factory.setFeature("http://xml.org/sax/features/namespace-prefixes",
true);
```

ContentHandler (méthodes principales)

`void`	• **`characters`**`(char[] ch, int start, int length)` appelé quand du texte est lu
`void`	• **`endDocument`**`()` appelé à la fin du document.
`void`	• **`endElement`**`(`**`String`**` uri, `**`String`**` localName,` **`String`**` qName)` appelé à la fin d'un élément
`void`	• **`startDocument`**`()` appelé au début du document
`void`	• **`startElement`**`(`**`String`**` uri, `**`String`**` localName,` **`String`**` qName, `**`Attributes`**` atts)` appelé au début d'un élément

Un exemple : lecture et affichage d'un fichier sxw

- vieux format de fichier openoffice
- but: afficher le texte, en affichant les notes de bas de page à la fin du document
- les éléments sont dans un namespace de préfixe "text:"
- les notes de bas de page sont dans les éléments text:footnote
- algo
  - on a un marqueur booléen à vrai si on est dans une note
  - quand on lit du texte:
    - si on est dans une note, l'ajouter au texte de la note courante
    - sinon, afficher le texte directement
  - quand on lit un début d'élément
    - si c'est une note, incrémenter le numéro et afficher l'appel de note
  - quand on lit une fin d'élément
    - si c'est une note, ajouter la note à la liste des notes
  - en fin de document: afficher les notes

```
public class TestXML extends org.xml.sax.helpers.DefaultHandler {
  private String currentFootnote;
  private int footnoteNumber;
  private Map<Integer,String> footnotes;
  private boolean inFootnote;

  public TestXML() {
   inFootnote= false;
   footnoteNumber= 0;
   footnotes= new TreeMap<>();
   currentFootnote= null;
  }
private void beginFootnote() {
 currentFootnote= "";
 inFootnote= true;
}
private void ecrire(String string) {
 if (inFootnote)
    currentFootnote= currentFootnote+ string;
 else
    System.out.print(string);
}
```

```
public void startElement(
 String uri,
 String localName,
 String qname,
 Attributes attributes)
 throws SAXException {
            //ecrire("<" + qname + ">");
            switch (qname) {
                case "text:footnote":
                    beginFootnote();
                    break;
                case "text:tab-stop":
                    ecrire("\t");
                    break;
                case "text:line-break":
                    ecrire("\n");
                    break;
            }
   }


} // TestXML
```

```
@Override
public void characters(char[] charArray, int start, int length)
 throws SAXException {
 String s= new String(charArray, start, length);
 ecrire(s);
}

@Override
public void endElement(String uri, String localName, String qName)
 throws SAXException {
 if (qName.equals("text:footnote"))
    endFootNote();
 else if (qName.equals("office:document-content"))
    finText();
}
```

Exemple 2: construction d'un arbre

- Pour gérer la structure du document XML en cours de lecture, on maintient une ***pile***.
- Elle comporte les représentations des éléments ancêtres de l'élément courant
- quand on arrive dans un nouvel élément : on l'empile
- quand un élément se termine : on le dépile

```
public class ConstructeurArbre {

    private MyHandler handler = null;

    public Noeud lire(InputStream in) throws
ParserConfigurationException, SAXException, IOException {
        handler = new MyHandler();
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser parser = factory.newSAXParser();
        InputSource src = new InputSource(in);
        parser.parse(src, handler);
        return handler.getResult();
    }
```

```
private class MyHandler extends DefaultHandler {

     private Stack<NoeudElement> pile = new Stack<>();
     private NoeudElement result = null;

     private Noeud getResult() {
         return result;
     }

    @Override
     public void startElement(String uri, String localName, String
qName, Attributes attributes) throws SAXException {
             NoeudElement n = new NoeudElement(qName);
             if (pile.empty()) {
                 result = n;
             } else {
                 NoeudElement parent = pile.peek();
                 parent.addFils(n);
             }
             pile.push(n);
         }
```

```
    private class MyHandler extends DefaultHandler {

        private Stack<NoeudElement> pile = new Stack<>();
        private NoeudElement result = null;

        private Noeud getResult() {
            return result;
        }

        @Override
        public void characters(char[] ch, int start, int length) throws
SAXException {
            String s = new String(ch, start, length);
            NoeudElement parent = pile.peek();
            parent.addFils(new NoeudTexte(s));
        }

        @Override
        public void endElement(String uri, String localName, String
qName) throws SAXException {
            // On dépile, car on est sorti de l'élément...
            pile.pop();
        }
```

uri, localname et qname

- URI : l'URI associée au name space de la balise
- localname: le nom local (sans URI) de la balise
- qname : le nom de la balise, sous la forme prefix:nom ou nom
- Si URI est définie, alors on a un localname, et qname *peut* être null
- Sinon, uri est null, localname peut l'être, et qname est rempli.

uri, localname et qname

- tout dépend des valeurs des propriétés **http://xml.org/sax/features/namespaces** et **http://xml.org/sax/features/namespace-prefixes**

- URI et localName sont forcément fournis si **http://xml.org/sax/features/namespaces** est true et sont optionnels sinon ;

- URI et localName sont, soit tous les deux omis, soit tous les deux fournis ;

- qName est fourni si **http://xml.org/sax/features/namespace-prefixes** est vraie, et optionnel sinon ;

- par défaut, **namespaces** est à vrai, et **namespace-prefixes** est à faux

Désactivation complète de la validation

- On peut vouloir fournir une DTD fournie avec l'application, ou fournir une DTD vide en désactivant la validation.

- On redéfinit resolveEntity

```
@Override
public InputSource resolveEntity(String publicId, String systemId)
                    throws SAXException {
  return new InputSource(
  new StringReader("<?xml version=\"1.0\" encoding=\"UTF-8\"?>"));
}
```

Conclusion sur SAX

- API de très bas niveau
- le document n'est pas intégralement chargé en mémoire
- utile pour extraire des informations simples
- si on veut travailler sur la structure de l'arbre, pas pratique : DOM et JDOM fournissent une bibliothèque plus riche

DOM et JDOM

- DOM : Document Object Model, spécification *indépendante des langages* pour manipuler des arbres de nœuds XML

- trop générale : n'utilise pas les collections → JDOM plus idiomatique en Java

- Utilisable pour ***lire*** et ***écrire*** des documents XML

JDOM

- Document XML vu comme un arbre :
- nœuds Element: associés aux éléments XML
- nœuds Text : associés au texte XML

```
public class DomDump {
public static void main(String[] args) throws Exception {
        SAXBuilder builder = new SAXBuilder();
        builder.setFeature(
    "http://apache.org/xml/features/nonvalidating/load-external-dtd",
                    false);
        InputSource xmlSource = new InputSource(…);
        Document jdomDocument = builder.build(xmlSource);
        affiche(jdomDocument.getRootElement(), 0);
    }
    private static void affiche(Content c, int marge) {
        switch (c.getCType()) {
            case Element:
                Element e = (Element) c;
                decale(marge);
                System.out.println("élément " + e.getQualifiedName());
                decale(marge);
                System.out.println("Contenu ");
                for (Content fils : e.getContent()) {
                    affiche(fils, marge + 1);
                }
                break;
           case Text:
                Text t = (Text) c;
                decale(marge);
                System.out.println(t.getTextTrim());
                break;
    }
}
```

Production de XML à partir de JDOM

- créer un Document (org.jdom2.Document)
- lui ajouter du contenu
- Ensuite:

```
XMLOutputter out = new XMLOutputter(Format.getPrettyFormat());
out.output(root, writer);
writer.close();
```

XMLEncoder et XMLDecoder

- Classes de java.bean
- fournissent un dump des beans java
- peu structuré (on ne choisit pas comment il est écrit)
- plus "stable" que la sérialisation normale.

XML "moderne"

Les schémas

- problème des **DTD**
  - au départ, ***textes*** structurés → (presque) rien de prévu pour limiter les valeurs des attributs ou du contenu textuel des éléments
    - par exemple, impossible de dire que dans
      - <prix unite="euro">245</prix>, le texte doit être un nombre
      - idem pour les attributs (les DTD permettent de proposer des listes de valeurs prédéfinies, ou de contraindre la valeur à être un identifiant, mais rien d'autre)
  - pas adapté aux ***données*** structurées
- Les schémas **xsd** résolvent le problème

Définition des schémas

- Dans un fichier XML de suffixe .xsd

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  … contenu…
</xsd:schema>
```

Schéma

- on a des types prédéfinis (exemple : xsd:integer ou xsd:string)

- on peut déclarer ses propres types; qui peuvent faire référence à des éléments

- on déclare aussi des élements, dont on donne le type

XSD

```
<xs:element name="telephone">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="numero" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

XSD

```
<?xml version="1.0"?>
<xs:schema version="1.0"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
   <xs:element name="personne">
      <xs:complexType>
         <xs:sequence>
            <xs:element  name="id" type="xs:integer"/>
            <xs:element  name="nom" >
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:minLength value="1"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="telephones">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="telephone" maxOccurs="unbounded"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
</xs:element>
```

Lier un document XML à un schéma

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
          "http://maven.apache.org/POM/4.0.0
                http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

- Ce document se trouve dans le namespace des fichiers POM 4.0 (xmlns)

- le schéma, dont le namespace est maven.apache.org/POM/4.0.0, se trouve à l'adresse maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)

- la ligne xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" sert juste à déclarer le namespace xsi

Lier un document à un schéma

- xsi:schemaLocation: si le schéma est lié à un namespace
- xsi:noNamespaceSchemaLocation : si le schéma n’est pas lié à un namespace (souvent, fichier local).

Relax NG

- XML est illisible (et une plaie à écrire à la main). Les schémas sont en XML. Donc les schémas sont illisibles.

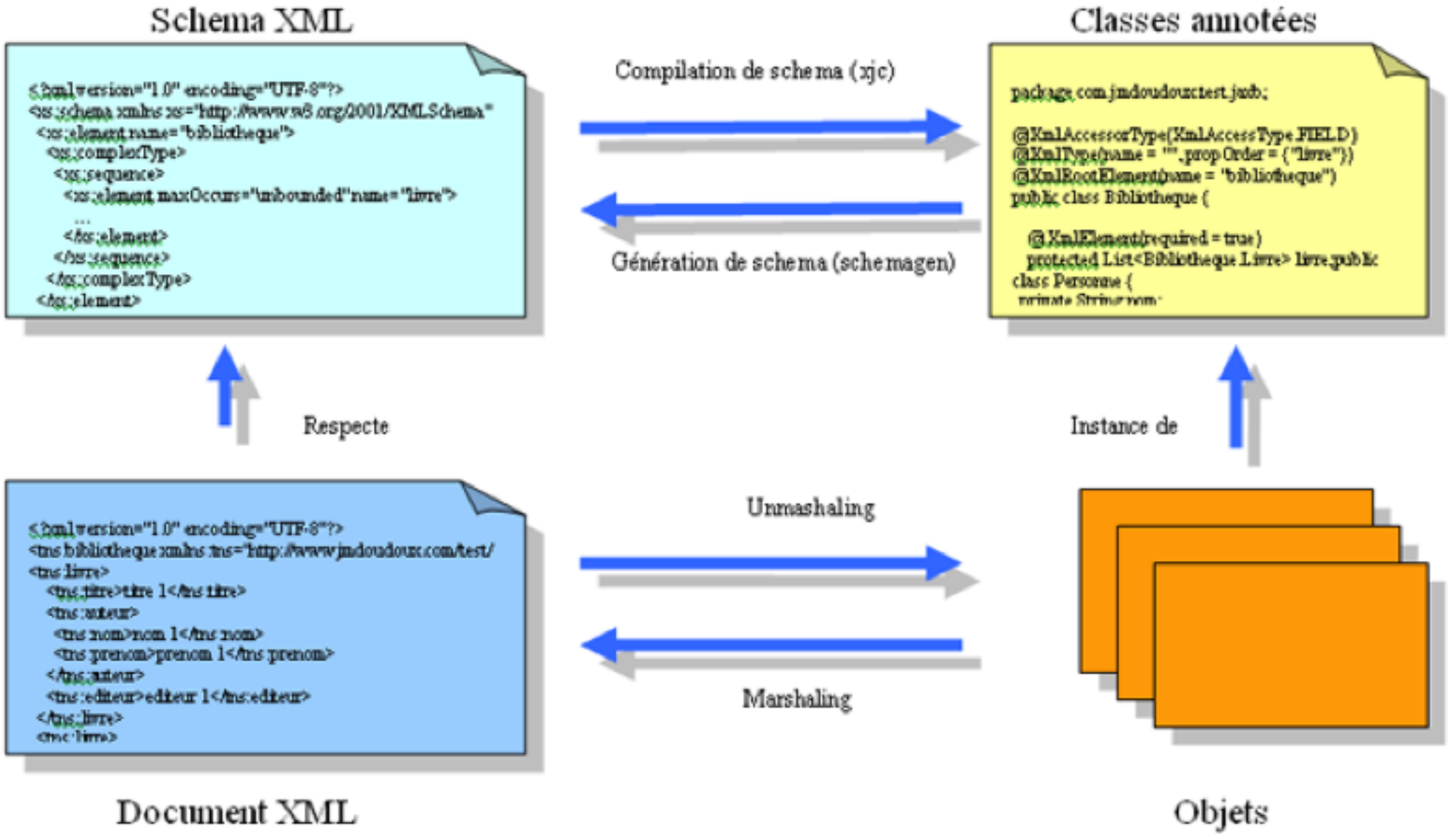
- Relax NG: mécanisme permettant (à l'origine) d'écrire un schéma dans un langage dédié, et de le transformer ensuite au format .xsd. Deux syntaxes pour les schémas relax NG: XML et non-XML ("compacte")

```
element addressBook {
  element card {
    element name { text },
    element email { text },
    element prefersHTML { empty }?
  }*
}
```

JAXB

- Système permettant de mapper des classes java et du XML
- Standard à partir de Java 1.6
  - sérialisation/désérialisation= marshal/unmarshal

Adéquation



Extrait de http://www.jmdoudoux.fr/java/dej/chap037.htm

```
public class Cours {
   private String titre = "";
    private List<String> labels = new ArrayList<String>();
    public Cours() {}
    public Cours(String titre) {this.titre= titre;}
    public void addLabel(String l) {labels.add(l);}
    public String getTitre() {return titre;}
    public void setTitre(String titre) {this.titre = titre;}
    public List<String> getLabels() {return labels;}
    public void setLabels(List<String> labels) {this.labels = labels;}
    public static void main(String[] args) throws JAXBException {
      Cours c= new Cours("glg203");
       c.addLabel("java");
       c.addLabel("pattern");
       StringWriter w= new StringWriter();
       JAXB.marshal(c, w);
       String s= w.toString();
       System.out.println(s)
       Cours c1 = JAXB.unmarshal(new StringReader(s), Cours.class);
       System.out.println(c1);
    }
}
```

Exemple (sans schéma)

Annotations

- On contrôle le fonctionnement de JAXB à l'aide d'annotations

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "numero"
})
@XmlRootElement(name = "telephone")
public class Telephone {
    @XmlElement(required = true)
    protected String numero;
    public String getNumero() {
        return numero;
    }
    public void setNumero(String value) {
        this.numero = value;
    }
}
```

Autre exemple

- package glg203.xml.jaxb des exemples de cours :
  - génération de fichiers java à partir d'un schéma
  - sauvegarde et chargement des objets correspondant

En pratique

- On utilisera éventuellement des tâches ant ou maven

- netbeans permet de créer des classes java à partir d'un schéma

Recherche et manipulation des fichiers XML

XPath

- Pour manipuler, chercher… un document XML, il est utile de disposer d'un système qui permet de définir des **parties** d'un document

- Utilisé par XSLT (et XLink, XQuery…)

Principes de XPATH

- On veut décrire une structure dans un arbre
- XPath permet de décrire un ensemble de nœuds ou de valeurs à partir *d'une racine* (celle du document, ou un nœud donné)
- Les éléments de base de Xpath sont
  - les axes (qui expliquent dans quel sens on cherche)
  - les nœuds : restreignent la recherche sur un type d'élément
  - les prédicats (optionnels) qui permettent d'exprimer des restrictions sur les nœuds

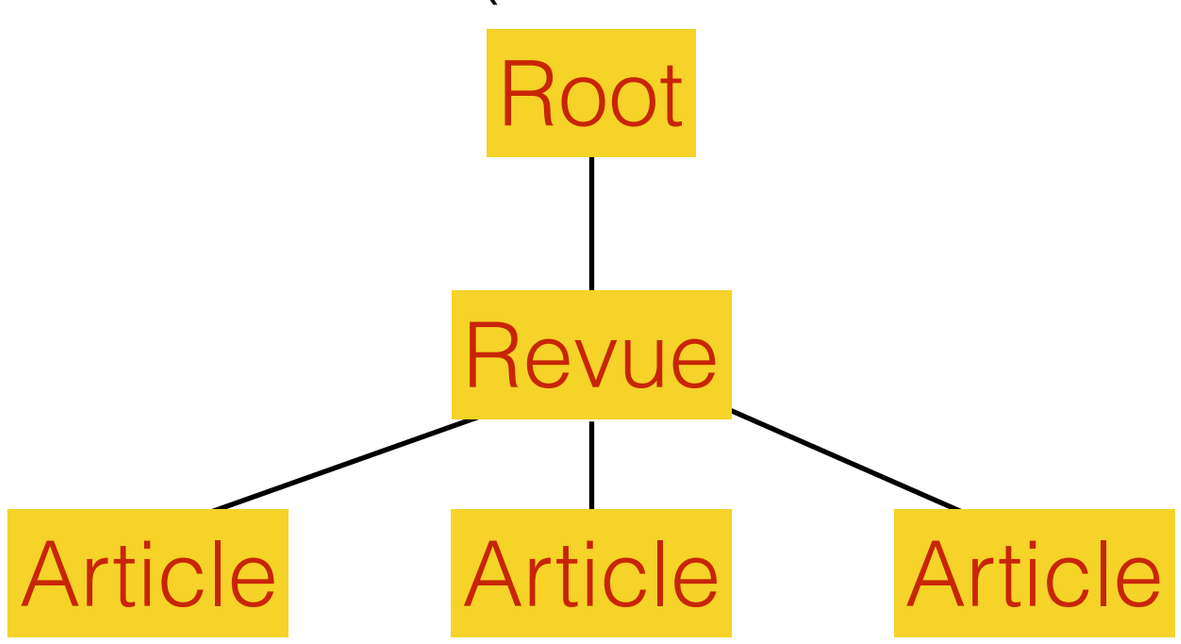
```
<?xml version="1.0" encoding="UTF-8"?>
<revue>
    <preface></preface>
    <article pages="1-30">
        <titre>Le calcul de la suite de Fibonnacci </titre>
        <auteur>Ada Lovelace</auteur>
        <chapitre>
            <titre>Introduction</titre>bla bla bla...
        </chapitre>
    </article>
    <article pages="31-60">
        <titre>Enigma decrypted </titre>
        <auteur>Alan Turing</auteur>
        <chapitre>
            <titre>Introduction</titre> bla bla bla...
        </chapitre>
    </article>
    <article pages="61-80">
        <titre>Notes de lecture</titre>
        <chapitre>
            <titre>Introduction</titre>bla bla bla...
        </chapitre>
    </article>
</revue>
```

Document utilisé dans les exemples

Forme de l'arbre

- Pour garder la cohérence avec le dom, les bibliothèques XML ajoutent une racine abstraite, qui ne correspond pas à l'élément racine du document

- dans notre document (élément racine: revue), on aura:



Expression xpath

- dans une expression xpath, on a au moins un axe, et un sélecteur de nœud, séparé par « :: »

- Exemple :

- **descendant::auteur** : sélectionne tous les nœuds auteurs descendant de la racine

  - descendant : axe

  - auteur : sélecteur de nœud

- descendant::auteur/ancestor::article

Exemples simples (donnés à partir de la racine du document)

- Note: le résultat est a priori un ***ensemble***.
- **descendant::*** : sélectionne tous les nœuds, sauf root (mais pour le coup, sélectionne aussi "revue").
- **descendant::auteur** : sélectionne tous les nœuds "auteur"
- **descendant::text()** : sélectionne tous les nœuds-textes (par opposition aux éléments)
- **descendant::auteur/text()** : le texte contenu dans les nœuds "auteurs" (pas dans leurs descendants)
  - text() est un sélecteur de nœud

Plus complexe

- On peut avoir plusieurs séries de couples axe::sélecteur de nœud, séparés par des "/"
- auquel cas, on commence la recherche pour le couple n+1 à partir de ce que le couple n a sélectionné
- exemple : **descendant::auteur/ancestor::article**
  - on sélectionne d'abord tous les nœuds auteur du document
  - ensuite, on remonte, et on sélectionne les ancêtre de ces nœuds qui sont des articles
  - au final: sélectionne tous les articles qui ont un auteur (soit les deux premiers articles dans notre exemple)

Prédicats

- Suite d'expressions entre crochets […], placés après un sélecteur de nœuds. Conditions supplémentaires

- [3] (un nombre) : sélectionne le nième (on commence à 1) élément dans l'ensemble des résultats
    - **descendant::article[3]** : sélectionne le troisième article (et non pas un article qui serait le troisième fils; on a l'élément <preface> avant)

- [contains(.,'une chaîne')] : teste si le texte inclus (pas forcément fils) dans le nœud contient la chaîne.
    - **descendant::auteur[contains(.,'Lovelace')] :** vrai si l'auteur contient "Lovelace"

Raccourcis pour les axes

- On a la syntaxe abrégée suivante (on n'utilise alors pas « :: »

  - // : pour "descendant"

  - @attr : l'attribut de nom attr. **//article/@pages :** *valeur des attributs* pages *des éléments* article

  - . : nœud courant

  - .. : parent du nœud courant

Les axes

- child
- descendant ; descendant-or-self
- parent, ancestor, ancestor-or-self
- following-slibing, preceding-slibing : frères
- preceding, following : nœuds précédents (suivants) sauf les ancêtres (les descendants)
- attribute : axe des attributs
- namespace : axe des namespaces d'un élément
- self : le nœud courant

les sélecteurs de nœuds

- nom d'élément (ou d'attribut) : reconnaissent les éléments ou attributs correspondants. Sensibles aux namespaces :
  - **//pom:modelVersion** : reconnaît modelVersion dans l'espace de nom dont le préfixe est pom
- * : reconnaît tout élément
  - ***//**** : tous les éléments du document.
- text() : reconnaît les nœuds-texte
- node() : reconnaît tout nœud (texte, élément, attribut, namespace, racine, commentaire, instruction)

Les prédicats (sélection)

- ***N*:** où N est un nombre; ième élément (en fait, parmi les nœuds frères sélectionnés, renvoie le ième)
- last() : indice du dernier élément. On peut utiliser last() -1 pour l'avant dernier, etc.
- comparaisons utilisant diverses fonctions
  - count(ensemble de nœuds)
    - **//article[count(*) > 2]**: renvoie les articles qui ont plus de 2 éléments fils (premier et second article dans l'exemple)
  - contains(s1,s2) : vérifie si s1 contient s2. s1 est souvent '.'.
  - @ + attribut : teste si un attribut existe, ou, avec une comparaison, s'il a une certaine valeur.
    - **//*[@pages]** ou **//*[@pages='61-80']**
  - nom d'élément : permet de tester la présence d'un élément, ou sa valeur

Opérateurs

- | : sépare deux ensembles de nœuds (dont on fait l'union)

- +, -, *, div, mod : opérateurs arithmétiques usuels

- =, !=, <, >, <=, >= : comparaisons

- or, and : opérateurs booléens.

- Remarque : not(…) est une fonction, pas un opérateur (parenthèses obligatoires).

XSLT

- langage pour créer des documents (essentiellement XML et texte) à partir de documents XML
- système de réécriture ; on applique des règles
- écrit en XML:
  - racine xsl:stylesheet
  - contient un élément xsl:output, dont l'attribut method permet de choisir ce qui est produit (xml, html, text)
  - contient essentiellement des templates, règles de réécriture qui
    - reconnaissent un nœud
    - expliquent comment le réécrire
- En l'absence de toute règle, le système recopie le *texte* des éléments.

XSLT : template

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
    <xsl:output method="text"/>
    <xsl:strip-space elements="*"/>
    <xsl:template match="auteur">
        **<xsl:value-of select="text()"/>**
    </xsl:template>
</xsl:stylesheet>
```

sortie texte (non xml)

supprime les espaces

règle qui entoure les noms des auteurs par des ** **

Template

- la template précédente reconnaît "auteur"
- elle ne transforme que ces éléments
- les autres sont transformés par la transformation par défaut: elle copie leur texte
- donc: notre feuille recopie le *texte* des documents, en entourant les noms d'auteurs en ** ..**

template "effacer"

- La feuille suivante produit un document vide:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="text"/>
   <xsl:template match="/"><!-- rien !!! --></xsl:template>
</xsl:stylesheet>
```

- la template reconnaît la racine du document
- et la remplace par rien…
- ***si un élément est reconnu, ses enfants ne sont plus analysés (sauf si on le demande avec xsl:apply-templates)***

Remplacement d'un élément

- ici, deux règles:
  - a priori, on recopie tel quels tous les éléments
  - si la balise est « auteur », on remplace par « nomAuteur »
    - xls:copy : recopie la balise
      - dedans: appel récursif des templates
    - règles de priorités pour départager les règles applicables (pour auteur, les deux règles conviendraient)

```
<xsl:template match="*">
        <xsl:copy>
            <xsl:apply-templates>
            </xsl:apply-templates>
        </xsl:copy>
</xsl:template>

<xsl:template match="auteur">
    <nomAuteur>
        <xsl:value-of select="text()"/>
    </nomAuteur>
</xsl:template>
```

Exemple

- première règle: efface tout
- seconde règle: recopie le texte de dc:description

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dc='http://purl.org/dc/elements/1.1/'>
<xsl:output method='text'/>
<xsl:template match='text()|@*'>
</xsl:template>
<xsl:template match='//dc:description'>
    <xsl:value-of select='.'/>
</xsl:template>
</xsl:stylesheet>
```

Exemple production d'une table des matières

```
<xsl:template match="mondocument">
    <html>
      <body>
          <!-- la table des matières -->
          <ul>
             <xsl:for-each select="chapitre/titre">
                <li>
                   <a href="#{generate-id(..)}">
                      <xsl:number count="chapitre"/><xsl:text>. </xsl:text>
                      <xsl:value-of select="."/>
                   </a>
                </li>
             </xsl:for-each>
          </ul>
          <!-- le corps du document -->
          <xsl:apply-templates/>
      </body>
    </html>
</xsl:template>
```

(à suivre …)

```
<!-- On transforme les titres en titres html,
   en créant un identificateur pour la table des matières
-->
<xsl:template match="chapitre/titre">
    <h2 id="{generate-id(..)}">
        <xsl:number count="chapitre"/>
        <xsl:text>. </xsl:text>
        <xsl:apply-templates/>
    </h2>
</xsl:template>
```

Exemple production d'une table des matières

```
<xsl:template match="mondocument/titre">
    <h1>
        <xsl:apply-templates/>
    </h1>
</xsl:template>
<!-- Les commentaires ne sont pas affichés : -->
<!-- Sans cette règle, leur texte serait recopié tel quel. -->
<xsl:template match="commentaire">
</xsl:template>
```

JSON
Javascript Object Notation

- Format très très simple
- représentation des objets javascript
- types de base : string, nombres, booléens, null
- objets sous la forme :
  - {"nom" : "Turing", "prenom" : "Alan"}
- listes entre [….]

JSON

```
[
    {"nom": "Lovelace", "prenom": "Ada", "telephones": [
            "073544333", "06876434"
        ]
    },
    {"nom": "Turing", "prenom": "Alan", "telephones": [
        "05683494", "06744624"
        ]
    }
]
```

JSON

- Des systèmes de spécifications (mais aucun ne s’est imposé) : JSON Schema
- Très simple, moins verbeux que XML
- plus adapté pour des données que pour du texte
- Langage « naturel » des applications web riches (Javascript)

De java à JSON: Jackson

- Fonctionne avec des annotations (voire sans)
- En écriture :

```
ObjectMapper mapper= new ObjectMapper();
Personne p= new Personne("Alan", "Turing", 30);
StringWriter w= new StringWriter();
mapper.writeValue(w, p);
System.out.println(w.toString());
```

- (Personne est un POJO)
- Produit: {« nom":"Alan","prenom":"Turing","age":30}

Jackson

- En lecture

```
String data= "{\"nom\":\"Alan\",\"prenom\":\"Turing\",\"age\":30}";
ObjectMapper mapper= new ObjectMapper();
StringReader src= new StringReader(data);
Personne p= mapper.readValue(src, Personne.class);
System.out.println(p.getNom()+ " age: " + p.getAge());
```

- Personne est un POJO avec un constructeur par défaut

Jackson plus annotations

- @JsonProperty("name") : nomme une propriété

- @JsonIgnore : ignore une propriété

Annotation des constructeurs

- Permet de ne pas avoir de constructeur par défaut ;

```
@JsonCreator
  public Personne2(
        @JsonProperty("nom")String nom,
        @JsonProperty("prenom") String prenom,
        @JsonProperty("age") int age) {
        this.nom= nom;
        this.prenom= prenom;
        this.age= age;
}
```

Mapping de l’héritage

- problème : faire comprendre à JSON la classe des objets ;
- plusieurs manières de préciser les classes ;
- généralement dans la classe de base

```java
/**
 * Forme
 */
@JsonTypeInfo(use = JsonTypeInfo.Id.NAME,
              include = JsonTypeInfo.As.PROPERTY)
@JsonSubTypes({
    @JsonSubTypes.Type(value = Cercle.class, name = "cercle"),
    @JsonSubTypes.Type(value = Rectangle.class, name = "rectangle")})
public abstract class Forme {
    protected Forme() {}

}
```

```java
public class Cercle extends Forme{
    private int x,y,r=1;

    public Cercle() {
    }
…
}
```

```
class FormeList extends ArrayList<Forme>{}
```

```
ObjectMapper mapper= new ObjectMapper();
Cercle c= new Cercle(2,4,6);
Rectangle r= new Rectangle(4,7,10,11);
FormeList formes= new FormeList();
formes.add(c);
formes.add(r);
StringWriter w= new StringWriter();
String s= mapper.writeValueAsString(formes);
System.out.println(s);
FormeList f= mapper.readValue(new StringReader(s), FormeList.class);
System.out.println(f);
```

[{"@type":"cercle","x":2,"y":4,"r":6},{"@type":"rectangle","x1":4,"x2":7,"y1":10,"y2":11}]

[cercle(2,4,6), rectangle(4,7,10,11)]

Les concurrents d'XML

- JSON
  - JavaScript Object Notation
  - bibliothèque "**jackson**"
  - draft: json-schema
  - remplace XML dans AJAX
- YAML