Spring, fichiers structurés : JSON/XML/YAML

GLG 203/Architectures Logicielles Java

Serge Rosmorduc
serge.rosmorduc@lecnam.net
Conservatoire National des Arts et Métiers

2019–2020

Les démonstrations

SI POSSIBLE : REFONDRE CE COURS POUR LUI DONNER UN PLAN PLUS RAMASSÉ. ALLER PLUS LOIN SUR OBJECTMAPPER.

`https://gitlab.cnam.fr/gitlab/glg203_204_demos/08_spring_xml_json.git`

00_demo_simples mapping divers, JSON et XML ; démonstrations de diverses technologies XML (XPATH, XSLT…)

02_springJSonDemo Démonstration d'application Spring boot utilisant JSon ;

03_springJaxb Démonstration d'application Spring boot utilisant Jaxb

04_demoMessageConverters Démonstration des messages converters en Spring MVC.

Prélude : Http, Spring et formats de données

Content-type

- Pour décrire les formats produits et demandés, http utilise MIME :
- forme *famille/format*. Exemple `text/html`, `image/png`, `application/xml` ...
- éventuellement complété d'informations comme le codage : `text/html;charset=UTF-8`
- en-tête de requête ou de réponse : `Content-type` ;
- en-tête de requête : `Accept` ;
- négotiation possible de la forme du contenu demandé.

La classe MediaType

- définit les types MIME usuels : `MediaType.IMAGE_JPEG_VALUE`, etc.
- permet de créer des types plus précis :

```
type = new MediaType(MediaType.TEXT_HTML,
                     Charset.forName("UTF-8"))
```

- permet des comparaisons : `equals`, `equalsTypeAndSubtype`, `includes`, `isCompatibleWith`

MediaType et contrôleur

Les annotations de mapping admettent un attribut `produces` :

```
@GetMapping(value="/image",
            produces = {MediaType.IMAGE_JPEG_VALUE,
                        MediaType.IMAGE_PNG_VALUE})
```

- C'est généralement lié à l'usage de l'annotation `@ResponseBody`
- la valeur de retour du contrôleur est un objet, qui sera converti au bon format si c'est possible.

HttpMessageConverter

- utilisé par `ContentNegotiatingViewResolver`
- utilisés par Spring pour convertir un `ResponseBody` dans le format désiré ;
- certains sont déjà fournis ;
- d'autres doivent être activés si besoin ;
- on peut écrire les siens.

HttpMessageConverter

Permet potentiellement la conversion dans les deux sens (mais pas forcément).

```
public interface HttpMessageConverter<T> {
  boolean canRead(Class<?> clazz, @Nullable MediaType mediaType);
  boolean canWrite(Class<?> clazz, @Nullable MediaType mediaType);
  List<MediaType> getSupportedMediaTypes();
  T read(Class<? extends T> clazz, HttpInputMessage inputMessage)
      throws IOException, HttpMessageNotReadableException;
  void write(T t, @Nullable MediaType contentType,
                  HttpOutputMessage outputMessage)
      throws IOException, HttpMessageNotWritableException;
}
```

- souvent implémenté en étendant `AbstractHttpMessageConverter` ;
- mais souvent on se contente des Converters existant ;
- Exemple : projet `04_demoMessageConverter`.

Mise en place des converters

Dans une classe qui implémente `WebMvcConfigurer` :

```
@Configuration
public class MyConfig implements WebMvcConfigurer{
 @Override
 public void extendMessageConverters(
          List<HttpMessageConverter<?>> converters) {
   converters.add(new BufferedImageHttpMessageConverter());
   converters.add(new PersonneConverter());
 }
}
```

Deux méthodes possibles :

- `extendMessageConverters` : *ajoute* de nouveau convertisseurs à ceux qui existent ;
- `configureMessageConverters` : *remplace* les convertisseurs existant.

Converters par défaut

Pour Spring boot, les convertisseurs suivants sont déclarés :

ByteArrayHttpMessageConverter renvoie directement du binaire ;

StringHttpMessageConverter gère l'envoi direct de texte ;

ResourceHttpMessageConverter permet d'envoyer une ressource (un fichier inclus dans l'archive war) ;

ResourceRegionHttpMessageConverter partie d'une ressource ;

SourceHttpMessageConverter fichiers XML

AllEncompassingFormHttpMessageConverter traite les éléments de formulaires ;

MappingJackson2HttpMessageConverter JSON (et éventuellement XML) ;

Jaxb2RootElementHttpMessageConverter XML avec Jaxb, si **java 1.6–1.8** ou Jaxb inclus.

Les documents structurés

Exemple de fichier structuré

configuration de mysql

```
#
# 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
```

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\tx453
6803\li1540\fi340\pardirnatural
\cf0 Je suis le T\'e9n\'e9breux, - le Veuf, l\'92Inconsol\'
\pard\tx1133\tx1700\tx1906\tx2267\tx2834\tx3401\tx3968\tx45
\cf0 Le prince d\'92Aquitainne \'e0 la Tour abolie :\
```

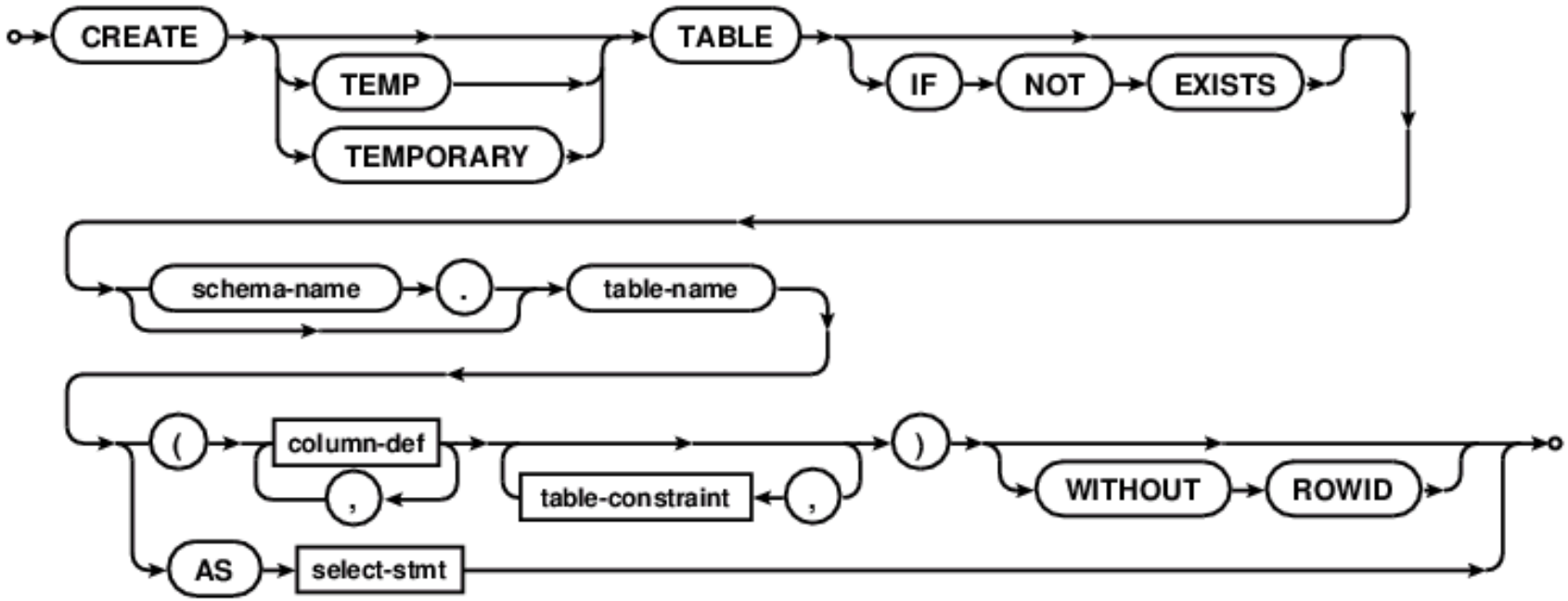
Analyse d'un fichiers structuré

- Approche classique : définition d'une *grammaire* ;
- écriture d'un analyseur syntaxique qui *reconnaît* cette grammaire ;
- produit une représentation en mémoire du contenu du fichier (typiquement un arbre)

Grammaires formelles

diagramme de syntaxe pour SQL

create-table-stmt:



Used by: sql-stmt

References: column-def select-stmt table-constraint

See also: lang_createtable.html

Grammaires formelles

ANTLR pour Java

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

forInit
    :   localVariableDeclaration
    |   expressionList
    ;

enhancedForControl
    :   variableModifier* type variableDeclaratorId ':' expression
    ;

forUpdate
    :   expressionList
    ;
```

Grammaires formelles

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
- très verbeux.

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 abandonne la notion de DTD.

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="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>
```

Fichiers de configuration

```
<?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.
  <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>
```

Utilisation pour l'échange de données

- le « X » de « AJAX » ;
- Protocole SOAP : les requêtes et les informations sont communiquées sous forme de données XML ;
- de plus en plus concurrencé par JSON, YAML.

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 spécifiques :
  - 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 se conformer à une grammaire explicite
- en gros : correctement parenthésé
- plus précisément :
  - une balise racine unique ;
  - 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.

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 explicitement conformes à une grammaire :
  - DTD
  - schéma
  - Relax NG
  - (il y en a d'autres)

Utilisation d'une DTD Publique

- 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

Utilisation d'une DTD System

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

Plus d'information sur les DTD dans les annexes

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)

Exemple : svg d'inkscape

```
<?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

```
<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" />
```

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** .

Les schémas

Palient les limites des `DTD` :

- au départ, XML = **textes** structurés → (presque) rien pour typer les attributs ou le contenu textuel des éléments ;
- 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>
```

```xml
<?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>
</xs:schema>
```

Donner la localisation d'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 `http://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.
- la valeur de `xsi:schemaLocation` est une suite d'URI de *namespaces* et d'url donnant l'adresse où le schéma peut être téléchargé.

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 }?
  }*
}
```

JSON (format)

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)

YAML

- Format reposant sur l'indentation et le passage à la ligne ;
- très lisible pour une structure arborescente ;
- pas adapté pour du *texte structuré* ;
- pas forcément utile pour l'échange de données ;
- très pratique pour les fichiers de configurations.

```
environments:
    dev:
        url: https://dev.example.com
        name: Developer Setup
    prod:
        url: https://another.example.com
        name: My Cool App
```

Outils logiciels

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}
- permet de créer des sérialiseurs spécifiques pour certains types (ex : date vers String au lieu de structure JSON) ;
- autres solutions : JSON Binding API ...

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.**

Classe ObjectMapper

- Permet de sérialiser des objets : writeValue, writeValueAsString ;
- permet de désérialiser des objets :

  ```
  Personne p= mapper.readValue(src, Personne.class);
  ```

  Comme JSon n'indique pas le type des objets, il faut le préciser.
- paramétrable et extensible grâce à des *modules.*

Origine des données

- Par défaut, Jackson utilise les getters et les setters ;
- en désérialisation, il a normalement besoin d'un constructeur par défaut ;
- ...mais on peut tout paramétrer ...

Jackson plus annotations

- @JsonProperty("name") : nomme une propriété ; permet aussi d'inclure une propriété privée sans accesseur.
- @JsonIgnore : ignore une propriété

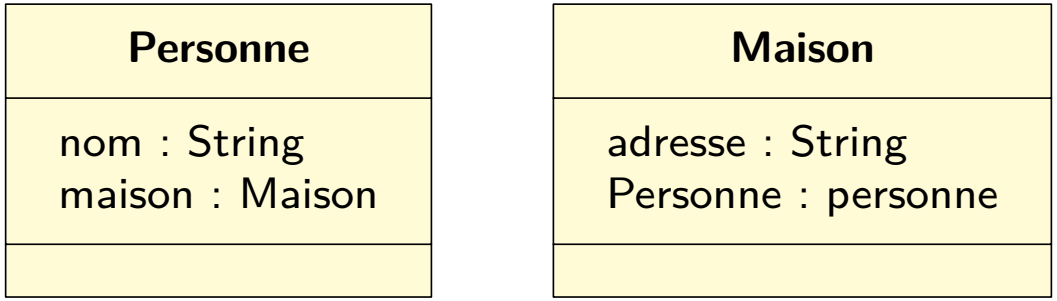
Annotation des constructeurs

Pour créer un objet à partir de code JSON, Jackson a besoin d'un constructeur par défaut (même `private`) !
Annoter le vrai constructeur 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;
}
```

Liaison bidirectionnelles

Le problème : si j'ai



Et que Ada a sa maison à Paris, la sérialisation JSON de l'objet ada :

```
Personne ada = new Personne("ada");
Maison m = new Maison("Paris");
ada.maison = m; m.personne = ada;
```

Donnerait :

```
{
 "nom" : "ada",
 "maison" : {
     "adresse" : "Paris",
     "personne" : {
            "nom" : "ada",
            "maison" : {
                "adresse" : "Paris",
                "personne": ...à l'infini...
```

Solution 1 : @JSonIgnore

Dit d'ignorer un champ :

```
class Maison {
  private String adresse;
  @JSonIgnore
  private Personne personne;
  ...
}
```

- Fonctionne bien en sérialisation :

```
{
 "nom" : "ada",
 "maison" : {"adresse" : "Paris"}}
```

- moins bien en désérialisation : le champ « personne » de maison n'est pas rempli à la lecture !

Solution 2 : @JsonManagedReference et @JsonBackReference

Solution robuste au problème :

On décide qu'un des deux côtés dirige :

```
class Personne {
  private String nom;
  @JsonManagedReference
  private Maison maison;
}
class Maison {
  private String adresse;
  @JsonBackReference
  private Personne personne;
  ...
}
```

- On obtient bien :

  ```
  {
   "nom" : "ada",
   "maison" : {"adresse" : "Paris"}}
  ```

  en sérialisation ;
- la sérialisation fonctionne.

@JsonIdentityInfo

- utilise un id, et l'utilise pour les références suivantes ;
- Très utile pour des structures de **graphes** ;
- les classes doivent avoir des champs « id » !

```
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class,
      property = "id")
public class Etudiant {
  private Long id;
  private String nom;
  private Maison maison;
}
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class,
      property = "id")
public class Maison {
  private Long id;
  private String adresse;
  private Personne personne;
}
```

La sérialisation donne :

```
{"id" : 1, "nom" : "Ada",
           maison : {"id" : 2,
                     "adresse" : "Paris", "personne": 1]]
```

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
/**
 * Classe 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() {}

}
```

Classes filles :

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

    public Cercle() {
    }
…
}
```

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

Utilisation :

```
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);
```

sorties :

```
[{"@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)]
```

Intégration de Jackson et de Spring

- Spring boot intègre par défaut `Jackson` ;
- on peut lui ajouter une extension XML de `Jackson` :

```
implementation
    'com.fasterxml.jackson.dataformat:jackson-dataformat-xml'
```

- Spring sait alors *lire* et *produire* du XML en plus du JSON.

Production de JSON et de XML

```
@Controller
public class ContactController {
  @Autowired
  private ContactService service;

 @GetMapping(value = "/contact/{id}")
  @ResponseBody
  public Contact getContact(@PathVariable Long id) {
    return service.getContactParId(id);
  }
```

@ResponseBody la valeur de retour de la méthode correspond à ce qui est renvoyé au client ;

cet objet est transformé selon le type de valeur acceptées ;

pour JSON, on va utiliser Jackson...

Consommation de JSON

```
@PostMapping(path = "/contact",
             consumes = {"application/json",
                         "application/xml"})
public void sendContact(@RequestBody Contact contact,
                        Writer out,
                        HttpServletResponse resp)
                        throws IOException {
    resp.setContentType("text/plain");
    out.write(contact.toString());
}
```

consumes précise quels formats sont acceptés d'entrée ;

@RequestBody : le *corps* de la méthode POST est du texte JSON.

tests manuels

Peuvent se réaliser avec l'utilitaire `curl` :

- ```
  curl -v -H "Accept: application/json" \
              http://localhost:8080/contact/3
  ```

- ```
  curl -d '{"id": 10, "nom": "lovelace", "adresse": "Cambridge", \
           "numeros": [\
           {"typeInfo" : "BUREAU", "numero": "01402654678"}]}' \
       -H "Content-Type: application/json" \
       -X POST http://localhost:8080/contact
  ```

Configuration de l'ObjectMapper

- par défaut, rien à faire...
- On peut ajouter un bean `Jackson2ObjectMapperBuilder` à une configuration ;
- il sera automatiquement utilisé pour configurer le ObjectMapper :

```
@Configuration
public class JSonEtXmlConfiguration {
  @Bean
  @Order(1) // utilisé après les valeurs par défaut.
  public Jackson2ObjectMapperBuilder monMapper() {
    Jackson2ObjectMapperBuilder builder =
                                  new Jackson2ObjectMapperBuilder();
    builder.indentOutput(true); // jolie présentation
    // on conserve les tableau vides, pas null :
    builder.featuresToDisable(
      DeserializationFeature.ACCEPT_EMPTY_ARRAY_AS_NULL_OBJECT
    );
    return builder;
  }
}
```

Configuration de l'ObjectMapper

- On peut aussi directement renvoyer un `ObjectMapper`.
- config de jackson par `application.properties` (annexe A de la doc de Spring Boot) :
  `spring.jackson.serialization.indent_output=true` ;

JAXB

- Système permettant de mapper des classes java et du XML ;
- Standard à partir de Java 1.6 ;
- Plus inclus dans le jdk à partir de Java 1.9...
- sérialisation/désérialisation= marshal/unmarshal.

Exemple (sans schéma)

```
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);
  }
}
```

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;
    }
}
```

En pratique

- On utilisera éventuellement des tâches ant, maven ou gradle ;
- netbeans permet de créer des classes java à partir d'un schéma.

JAXB et Spring boot

- Si java 8 : Utilisé automatiquement si des annotations Jaxb sont trouvées.
- Si java 9 et plus : ajouter en plus la dépendance `org.glassfish.jaxb:jaxb-runtime`

Autres bibliothèques

SAX Simple API for XML

- Pourquoi : lire un fichier XML
- Comment : lecture « événementielle » (pattern **builder**)

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)

- `characters(char[] ch, int start, int length)` : appelé quand du texte est lu ;
- `endDocument()` appelé à la fin du document ;
- `endElement(String uri, String localName, String qName)` appelé à la fin d'un élément ;
- `startDocument()` appelé au début du document ;
- `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 {
            switch (qname) {
                case "text:footnote":
                    beginFootnote();
                    break;
                case "text:tab-stop":
                    ecrire("\t");
                    break;
                case "text:line-break":
                    ecrire("\n");
                    break;
            }
  }
}
```

```
@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();
}
```

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

- pourquoi : garantie de produire du XML bien formé ;
- 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)
- format plus « stable » que la sérialisation normale.

Compléments

@JSonView

Permet de filtrer le code JSon produit et d'ignorer à volonté certains champs.
Exemple :

```
public class Utilisateur {
  @JsonView({FullView.class, NomEtIdView.class})
  private Long id;

  @JsonView({FullView.class, NomEtIdView.class})
  private String nom;

  @JsonView({FullView.class, MotDePasseView.class})
  private String motDePasse;
  ...
}
```

- id et nom seront montrés par les vues `NomEtIdView` et `FullView` ;
- le mot de passe sera montré par les vues `MotDePasseView` et `FullView`.

@JSonView : création des vues

Les vues sont de simples classes, généralement vides !

```
public class NomEtIdView {}
```

On peut éventuellement utiliser l'héritage.

Utilisation dans le code :

```
ObjectMapper mapper= new ObjectMapper();
Utilisateur u = new Utilisateur(31, "unnom",
                                "un mot de passe");
System.out.println("Complet : "+
                   mapper.writeValueAsString(u));
System.out.print("Vue sans MdP : ");
System.out.println(
  mapper.writerWithView(NomEtIdView.class)
      .writeValueAsString(u)
  );
```

Définition d'une DTD

- permet de spécifier la syntaxe d'un fichier XML
- DTD elle-même pas écrite en 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

Attributs des éléments

Définis 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

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 (`&eacute;`)
- Définition `<!ENTITY salut "Bien à vous">`
- Utilisation : `&salut;`
- le « ; » fait partie de l'entité !

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)>
]>
```

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

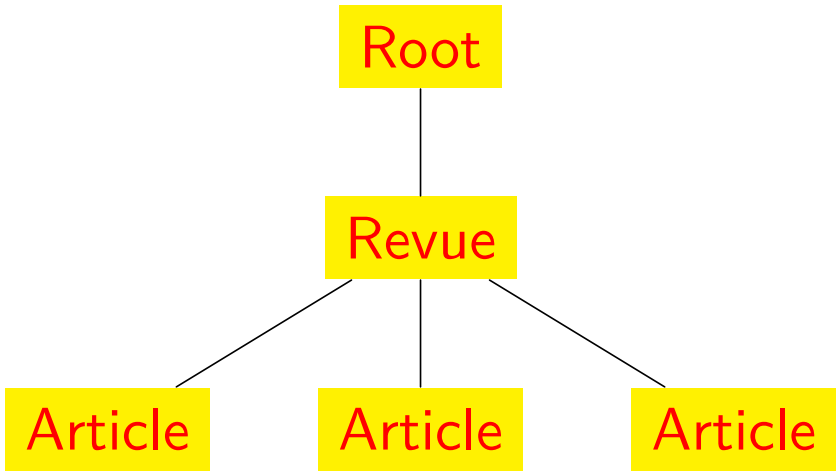
- On veut décrire une structure dans un arbre ;
- XPath permet de décrire un ensemble de nœuds ou de valeurs à partir d'une **racine** (celle du document, ou un nœud donné) ;
- Les éléments de base de Xpath sont :
  - les axes (qui expliquent dans quel sens on cherche) ;
  - les nœuds : restreignent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

Document utilisé dans les exemples

```
<?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>
```

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, séparés par « : »

Exemple

`descendant::auteur` : sélectionne tous les nœuds auteurs descendant de la racine

- descendant : axe ;
- auteur : sélecteur de nœud

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 ;
- définissent des 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 « :: »)

- `//` « 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]` 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 d'autres 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.

Exemple

```
  <?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>
```

- la template précédente reconnaît « auteur »
- elle ne transforme que ces éléments
- les autres sont transformés par la transformation par défaut : elle copie leur texte
- $\rightarrow$ 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>
```

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>
```

production d'une table des matières

```
<!-- 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>

<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>
```

Bibliographie

- E. R. Harold et W. S. Means, « XML in a Nutshell, 3rd Edition », O'Reilly, 2004 ;
- documentations de Jackson (très très lacunaires) : `https://github.com/FasterXML/jackson-docs`
- Site du W3C ;
- sur les grammaires formelles : Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, *Compilers : Principles, Techniques, and Tools,* 2006 (alias « the dragon book »).