XSLT

XML, un langage d'arbres

Master Recherche SIA - Master Pro ILI

Année 2013-14

XSLT : eXtensible Stylesheet Language for Transformation

- sous-ensemble du langage de feuilles de style *XSL*
- un ensemble de règles avec des *sélecteurs* qui produisent en sortie du *XML*

XSLT

- Une feuille de transformation *XSLT* est un document *XML* (ce qui n'est pas le cas des *DTD*).
- Il est constitué d'une suite de règles de la forme

```
<xsl:template match=sélecteur>
instructions
</xsl:template>
```

- Un document *XSLT* complet est de la forme

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- regles de transformations -->
</xsl:stylesheet>
```

La balise racine du document `<xsl:stylesheet>` contient un attribut `xmlns:xsl`.
Il permet d'indiquer un espace de noms :

- ensemble d'éléments et d'attributs définis
- indique au processeur XML que les éléments et attributs, tels qu'ils sont définis dans cet espace, seront utilisés dans le document préfixé par le `nom:` indiqué après `xmlns`
  - permet au processeur *XML* également de traiter différemment des groupes d'éléments
- très important dans le cas particulier d'une feuille *XSLT*, car cela permet au processeur de différencier les instructions de transformations des données.

XSLT

- Une feuille *XSLT* transforme un arbre source en un arbre cible
- Elle permet de se déplacer dans l'arbre source grâce à des sélecteurs
- Les règles permettent de passer aux enfants et de traiter les sous-arbres par des appels récursifs
- Une règle (un *template*, ou *modèle*) décrit la façon dont un sous-arbre doit apparaître.

La DTD (partielle) d'une recette de cuisine

```
<!ELEMENT recette (titre,fiche-technique,preparation) >
<!ELEMENT fiche-technique
  (nb-personnes,ingredients,tps-preparation,
          tps-cuisson,difficulte)>
<!ELEMENT ingredients (ingredient+) >
<!ELEMENT ingredient (nom,qte?) >
<!ELEMENT qte (#PCDATA) >
<!ATTLIST qte unite NMTOKEN "piece">
<!ELEMENT difficulte EMPTY >
<!ATTLIST difficulte niveau (facile|moyen|difficile)
          #REQUIRED >
<!ELEMENT preparation (phase+) >
```

Une recette de cuisine

```
<recette>
  <titre>Spaghettis à la sauce Bolognaise</titre>
  <fiche-technique><nb-personnes>6</nb-personnes>
    <ingredients>
      <ingredient><nom>spaghettis</nom>
        <qte unite="gr">900</qte></ingredient>
      <ingredient><nom>tomates</nom><qte>6</qte>
      </ingredient> ...
    </ingredients>
    <tps-preparation>20min</tps-preparation>...
    <difficulte niveau="facile" />
  </fiche-technique>
  <preparation>
    <phase>Emincez les oignons...</phase>...
  </preparation>
</recette>
```

Une recette de cuisine au format odt

```
<xsl:template match="/">
  <office:document>
    <office:body>
      <text:h text:level="1"
              text:style-name="Heading">
        Recette de Cuisine
      </text:h>
      <xsl:apply-templates />
    </office:body>
  </office:document>
</xsl:template>
```

Où s'applique une transformation XSLT ?

- généralement, côté serveur :
  - appel d'un processeur XSLT : `xsltproc` commande en ligne, `xalan` en Java, ...
  - ou passage par un serveur d'applications XML : Cocoon
  - permet de choisir les feuilles de transformations à appliquer
- (très rarement) sur le client : Firefox possède un processeur XSLT !

On peut associer une feuille de transformation à un document XML, en ajoutant une déclaration dans le prologue du document :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="recette.xsl" type="text/xsl"?>
<!DOCTYPE recette SYSTEM "recette.dtd" >
```

Attention :

- généralement, on n'attache pas une feuille de transformations à un document XML
- on va au contraire écrire plusieurs feuilles de transformations pour un même document en fonction de la sortie souhaitée

Une recette de cuisine au format html

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- regles de transformations -->
  <xsl:template match="/">
    <html>
      <head><title>Recette de cuisine</title></head>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>
```

Exemple

```
<xsl:template match="titre">
  <h1>
    <xsl:apply-templates />
  </h1>
</xsl:template>
<xsl:template match="nb-personnes">
  <p>Nbre de personnes :
    <xsl:apply-templates />
  </p>
</xsl:template>
<!-- idem pour tps-preparation, tps-cuisson -->
<xsl:template match="difficulte">
  <p>Difficulté :
    <xsl:value-of select="@niveau" />
  </p>
</xsl:template>
```

Exemple

```
<xsl:template match="preparation">
  <h3>Préparation</h3>
  <ol>
    <xsl:apply-templates />
  </ol>
</xsl:template>
<xsl:template match="phase">
  <li>
    <xsl:apply-templates />
  </li>
</xsl:template>
```

Exemple

```
  <xsl:template match="ingredients">
    <h3>Ingrédients</h3>
    <ul>
      <xsl:apply-templates />
    </ul>
  </xsl:template>
  <xsl:template match="ingredient">
    <li>
      <xsl:value-of select="nom" />,
      <xsl:value-of select="qte" />
      <xsl:value-of select="qte/@unite" />
    </li>
  </xsl:template>
</xsl:stylesheet>
```

Et le résultat.

Règles de modèles

Les règles de modèles :

- sont appliquées depuis le noeud racine
- déterminent récursivement les noeuds sur lesquels on peut appliquer une règle de modèle
- pour ignorer un sous-arbre : on omet l'appel récursif explicite sur la racine de ce sous-arbre
- règles implicites appliquées par défaut

Règles implicites

Noeuds de texte et attributs :

La règle modèle consiste à copier la valeur des noeuds de texte et d'attributs dans le document de sortie.

```
<xsl:template match="text()|@*">
    <xsl:value-of select="."/>
<xsl:template/>
```

Règles implicites

Éléments et noeud racine :

La règle modèle consiste à effectuer un appel récursif sur les fils (axe *child*).

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

Règles implicites

Commentaires, instructions de traitement et espaces de noms :

La règle modèle consiste à ignorer leur contenu.

```
<xsl:template match="comment()|
                     processing-instruction()|
                     spacename::node()"/>
```

Règles de modèles

Les sélecteurs : attribut `match`

`<xsl:template match="exprXPath"> ... </xsl:template>`

Sous-ensemble d'expressions XPath

- retournent uniquement des ensembles de noeuds ;
- uniquement axes descendants ou auto-référentiels (pas d'axe `parent`, ni `preceding`, ...)

- évaluation des expressions XPath de la droite vers la gauche.
- cohérent avec le comportement de XSLT : on veut savoir, pour un noeud donné, si une règle s'applique sur lui.

Exemple :

`corps/descendant::nomduchat`

Application des règles modèles

Pour chaque noeud sélectionné par un `xsl:apply-templates` :

- tester toutes les règles pour déterminer celles dont le sélecteur accepte le noeud ;
- si pas de règle : la règle implicite est appliquée
- si une seule règle : elle est appliquée
- si deux règles, une locale et une importée par `xsl:import` : priorité à la règle locale ;
- sinon algo de résolution des conflits basée sur une notion de priorité

Par défaut, `xsl:apply-templates` sélectionne les fils du noeud courant.

Règles de modèles

Résolution des conflits pour l'application des règles de modèles

- on peut spécifier des contextes d'appels (attribut `mode`)
- on peut définir la priorité de chaque règle (attribut `priority`)
- en l'absence de notation explicite, priorité calculée en fonction de la précision du sélecteur (similaire CSS)

Règles de transformation

Eléments de deuxième niveau

Eléments qui peuvent se trouver dans une règle de modèle.

- *structures de contrôle* : choix, conditionnelle, itération, boucle, etc ...
- définition de paramètres ou de variables locales (`xsl:param`, `xsl:variable`)
- ajouts d'éléments spécifiques dans le document cible

Règles de transformation

Eléments de deuxième niveau

- recopie de noeuds ou de sous-arbre (`xsl:copy` et `xsl:copy-of`)
- copier la valeur d'un noeud (`xsl:value-of`)
- applications de règles (`xsl:apply-templates`, `xsl:call-template`, `xsl:apply-imports`)
- numéroter des éléments (`xsl:number`)

xsl:value-of

Copier les valeurs des noeuds

Syntaxe :

```
<xsl:value-of select="expressionXPath" />
```

L'expression XPath est évaluée et son résultat est inséré à la place de la règle.

Exemple :

```
<personne>
   <nom>Dupont</nom>
   <prenom>Jean</prenom>
<personne>
```

xsl:value-of

On veut obtenir :

```
Etudiant : Jean Dupont
mel : Dupont.Jean@univ-artois.fr
```

xsl:value-of

On veut obtenir :

```
Etudiant : Jean Dupont
mel : Dupont.Jean@univ-artois.fr
```

On applique la règle de modèle :

```
<xsl:template match="personne">
Etudiant : <xsl:value-of select="prenom"/>
           <xsl:value-of select="nom"/>
mel : <xsl:value-of select="nom"/>
           .<xsl:value-of select="prenom"/>
           @univ-artois.fr
</xsl:template>
```

Attention ! Les retours à la ligne ont été ajoutés pour la lisibilité sur le transparent.

Un autre exemple

```
<liste>
  <personne>
    <prenom>Jean</prenom><nom>Dupont</nom>
  </personne>
  <personne>
    <prenom>Christine</prenom><nom>Dubois</nom>
  </personne>
  <personne>
    <prenom>Pierre</prenom><nom>Duchemin</nom>
  </personne>
</liste>
```

Un autre exemple

on veut comme résultat :

```
1/3 - Jean Dupont
2/3 - Christine Dubois
3/3 - Pierre Duchemin
```

Un autre exemple

on veut comme résultat :

```
1/3 - Jean Dupont
2/3 - Christine Dubois
3/3 - Pierre Duchemin
```

```
<xsl:template match="personne">
<xsl:value-of
     select="concat(position(),'/',last(),' - ',.)"/>
</xsl:template>
```

Les instructions conditionnelles

Le if

Syntaxe :

```
<xsl:if test="exprXPathBool">instructions</xsl:if>
```

Le test est une expression booléenne. Il n'y a pas de else.

```
<xsl:template match="*">
   <xsl:if test="child::*">
      J'ai des enfants.
   </xsl:if>
</xsl:template>
```

Les instructions conditionnelles

La recette de cuisine

Objectif : lorsqu'un ingrédient n'a pas de quantité spécifiée, on n'ajoute pas la virgule.

Règle à modifier :

```
<xsl:template match="ingredient">
 <li>
  <xsl:value-of select="nom" />,
  <xsl:value-of select="qte" />
  <xsl:value-of select="qte/@unite" />
 </li>
</xsl:template>
```

Proposition ?

Les instructions conditionnelles

La recette de cuisine

Objectif : lorsqu'un ingrédient n'a pas de quantité spécifiée, on n'ajoute pas la virgule.

```
<xsl:template match="ingredient">
 <li>
  <xsl:value-of select="nom" />
  <xsl:if test="qte">
   <xsl:text>, </xsl:text>
   <xsl:value-of select="qte" />
   <xsl:value-of select="qte/@unite" />
  </xsl:if>
 </li>
</xsl:template>
```

Et le résultat.

Les instructions conditionnelles

Le switch

Syntaxe :

```
<xsl:choose>
  <xsl:when  test="exprXPathBool">
    instructions
  </xsl:when>
  ...
  <xsl:otherwise>
    instructions
  </xsl:otherwise>
</xsl:choose>
```

La clause `<xsl:otherwise>` est optionnelle.

Conditionnelles : exemples

```
<xsl:template match="qte">
   <xsl:choose>
      <xsl:when test="@unite &gt; 100">
         je suis en grande quantité!
      </xsl:when>
      <xsl:when test="@unite &gt; 10">
         je suis en moyenne quantité
      </xsl:when>
      <xsl:otherwise>
         je suis en petite quantité
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

Structure de boucle

La boucle Pour

Syntaxe :

```
<xsl:for-each select="exprXPath">
  instructions
</xsl:for-each>
```

Attention ! Le contexte à l'intérieur de la boucle est le noeud traité !

Il y a donc adaptation du contexte pour :

- toutes les intructions XPath
- en particulier : `last()` et `position()`

Boucle : exemple

Exemple : obtenir une table des matières

```
<xsl:template match="livre">
   <xsl:for-each select="chapitre">
      <xsl:value-of select="position()"/>
      <xsl:value-of select="titre"/>
   </xsl:for-each>
   <xsl:apply-templates />
</xsl:template>
```

Recopie de noeuds et leur structure

Copier un fragment d'arbre

Syntaxe :

```
<xsl:copy-of select="exprXPath" />
```

Recopie les noeuds sélectionnés par l'expression XPath :

- les éléments
- leurs attributs
- leurs espaces de noms
- leurs fils (axe `child`)

Si l'expression ne retourne pas un noeud ou un ensemble de noeuds
$\Rightarrow$ conversion du résultat en une chaîne de caractères.

Recopie de noeuds et de leur structure

Pour copier le noeud courant

```
<xsl:template match="liste" mode="regle2">
  <xsl:copy-of select="personne" />
</xsl:template>
```

donne comme résultat :

```
<personne>
  <prenom>Jean</prenom><nom>Dupont</nom>
</personne>
<personne>
  <prenom>Christine</prenom><nom>Dubois</nom>
</personne>
...
```

Recopie de noeuds sans la structure

Pour copier le noeud courant

Syntaxe :

```
<xsl:copy>contenu</xsl:copy>
```

- les espaces de noms sont automatiquement copiés
- les attributs et les fils ne sont pas automatiquement copiés

```
<xsl:template match="personne">
  <xsl:copy>
    <!-- ce qu'on veut comme contenu -->
  </xsl:copy>
</xsl:template>
```

Recopie de noeuds

Pour copier le noeud courant

```
<xsl:template match="personne">
  <xsl:copy>
    Son nom est <xsl:value-of select="nom" />
  </xsl:copy>
</xsl:template>
```

ce qui donne :

```
<personne>son nom est Dupont</personne>
<personne>son nom est Dubois</personne>
<personne>son nom est Duchemin</personne>
```

Format du document résultat

- L'élément de premier niveau `xsl:output` permet de déterminer le formatage du document résultat :

```
<xsl:output
    [method="xml"|"html"|"text"]
    [encoding="nomEncodage"]
    [omit-xml-declaration="no"|"yes"]
    [doctype-public="public-ID"]
    [doctype-system="system-ID"]
    [cdata-section-elements="nomElt nomElt ..."]
    [indent="no"|"yes"]
    [media-type="typeMime" <!--text/xml,text/html-->]
/>
```

Gestion des espaces

- par défaut, les blancs (espaces, tabulations, ...) sont conservés par le processeur XSLT ;
- on peut demander la suppression des blancs :

```
<xsl:strip-space
    elements="*"|"prefixe:* nomElt nomElt"
/>
```

- les éléments listés correspondent aux éléments du document source
- les blancs seront enlevés avant la transformation
- `*` indique tous les éléments
- `prefixe:*` indique tous les élts de l'espace de nom `prefixe`

Modularité

- on peut incorporer des règles provenant d'une autre feuille de style

  `<xsl:import href="uri" />` `<xsl:include href="uri" />`
- les règles d'import doivent être placées en tête ;
- différence `xsl:import` et `xsl:include` : les règles ajoutées via `import` ont une priorité plus faible que les règles locales ou ajoutées via `include`.