XSLT

XML, un langage d'arbres

4 novembre 2019

Créer de nouveaux noeuds

Les éléments suivants permettent de créer de nouveaux noeuds dans le document cible :

- `xsl:element` : permet de créer un élément

```
<xsl:element name="le-nom" namespace="esp-nom"
use-attribute-sets="ens-attr">
autres attributs, fils
</xsl:element>
```

- `xsl:attribute` : permet de créer un attribut

```
<xsl:attribute name="le-nom" namespace="esp-nom">
contenu
</xsl:attribute>
```

Construction dynamique de noeuds

Modèle de valeur d'attribut

- `xsl:value-of` permet d'insérer dans un contenu d'élément le résultat d'une expression XPath
- on utilise { et } pour affecter à un attribut la valeur d'une expression XPath

Document initial :

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

Construction dynamique de noeuds

Feuille de transformation :

```
<xsl:template match="personne">
  <div class="personne"
       nom={nom} prenom={prenom} />
</xsl:template>
```

Résultat :

```
<div class="personne"
     nom="Martin" prenom="Jean" />
```

Créer de nouveaux noeuds

Pour passer de

```
<div class="MenuContextuel">
  <ul> ...  </ul>
</div>
<div class="principale">
  <h1>Les Cours</h1>
  <div class="section">
    <p>...</p>
  </div>
  <div class="section"> ... </div>
</div>
```

Créer de nouveaux noeuds

à

```
<MenuContextuel>
  <ul> ... </ul>
</MenuContextuel>
<principale>
  <h1>Les Cours</h1>
  <section>
    <p>...</p>
  </section>
  <section> ... </section>
</principale>
```

Créer de nouveaux noeuds

```
  <xsl:template match="div">
    <xsl:if test="@class">
      <xsl:element name="{@class}">
        <xsl:apply-templates />
      </xsl:element>
    </xsl:if>
  </xsl:template>

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

Définir des attributs récurrents

- l'élément `xsl:attribute-set` permet de définir un groupe d'attributs (élément de premier niveau)
- l'attribut `xsl:use-attribute-sets` permet d'utiliser un ensemble d'attributs déjà définis

Exemple :

```
<mon-document>
  <taille-max>
    <hauteur>3cm</hauteur>
    <largeur>3cm</largeur>
  </taille-max>
  <photo> ...</photo>
  ...
</mon-document>
```

Définir des attributs récurrents

```
<xsl:attribute-set name="taille">
  <xsl:attribute name="height">
    <xsl:value-of select="//taille-max/hauteur"/>
  </xsl:attribute>
  <xsl:attribute name="width">
    <xsl:value-of select="//taille-max/largeur"/>
  </xsl:attribute>
</xsl:attribute-set>
<xsl:template match="photo">
  <photo xsl:use-attribute-sets="taille"/>
</xsl:template>
<xsl:template match="mon-document">
  <xsl:apply-templates select="photo" />
</xsl:template>
```

Créer de nouveaux noeuds

Mais aussi. . .

- `xsl:comment` pour insérer un commentaire
- `xsl:processing-instruction` pour insérer une instruction de traitement (par exemple, une déclaration de feuille de style css dans un document html, . . .)
- `xsl:text` pour insérer un noeud texte

Trier des éléments

- Dans un élément `<xsl:apply-templates>` ou `<xsl:for-each>`, il est possible de choisir l'ordre dans lequel les éléments vont être traités (par défaut, c'est l'ordre dans l'arbre qui est pris) :

```
<xsl:sort [select="exprXPath" <!-- élt clé du tri -->]
          [data-type="text"|"number"]
          [order="ascending"|"descending"]
          [case-order="upper-first"|"lower-first"]
/>
```

- par défaut, la clé du tri est le noeud courant
- on peut déclarer plusieurs clés pour le tri (primaire, secondaire, etc...) en déclarant plusieurs éléments `<xsl:sort>`.

Trier des éléments – Exemple

```
<hotel>
  <chambre>
    <personne><nom>Durand</nom><age>52</age></personne>
    <animal>chien</animal>
  </chambre>
  <chambre>
    <personne><nom>Dupont</nom><age>22</age></personne>
    <personne><nom>Dupont</nom><age>18</age></personne>
  </chambre> ...
</hotel>

<xsl:apply-templates select="descendant::personne">
  <xsl:sort select="nom" />
  <xsl:sort select="age" data-type="number" />
</xsl:apply-templates>
```

Numéroter des éléments

- la fonction `position()` permet d'insérer facilement un numéro (par rapport à une position dans un ensemble)
- plus généralement pour insérer un nombre :

```
<xsl:number
      [value="exprXPath" <!-- position() par défaut -->]
      [count="exprXPath" <!-- ens. de référence -->]
      [from="exprXPath" <!-- restriction sur l'ensemble -->]
      [level="single"|"any"|"multiple"]
      [format="1"|"01"|"a"|"A"|"i"|"I"]
      ...
/>
```

Numéroter des éléments – Exemple (1)

```
<personne>
  <nom>Durand</nom><age>52</age><femme />
</personne>
<personne>
  <nom>Dupont</nom><age>22</age><femme />
</personne>
<personne>
  <nom>Dupont</nom><age>18</age><homme />
</personne>

<xsl:template match="personne">
  <xsl:number level="any" format="1" />
  Nom : <xsl:value-of select="nom" />
  Age : <xsl:value-of select="age" />
</xsl:template>
```

Numéroter des éléments – Exemple (2)

```
<xsl:template match="personne">
  <xsl:if test="femme">
    <xsl:number count="//personne[femme]"
                level="any" format="1" />
  </xsl:if>
  <xsl:if test="homme">
    <xsl:number count="//personne[homme]"
                format="a" level="any" />
  </xsl:if>
  Nom : <xsl:value-of select="nom" />
  Age : <xsl:value-of select="age" />
</xsl:template>
```

Numéroter des éléments – Exemple (3)

Instance XML :

```
<mon-document>
  <chapitre>
    <titre>le titre</titre>
    <chapitre>
      <titre>le titre</titre>
    </chapitre>
    <chapitre>
      <titre>le titre</titre>
      <chapitre>
        <titre>le titre</titre>
      </chapitre> ....
```

Transformations :

```
<xsl:template match="chapitre">
  <p><xsl:number level="multiple" format="1" />) Titre :
</xsl:template>
```

Numéroter des éléments – Exemple (4)

Résultat pour `level="multiple"` :

```
<p>1) Titre : le titre</p>
<p>1.1) Titre : le titre</p>
<p>1.2) Titre : le titre</p>
<p>1.2.1) Titre : le titre</p>
```

Résultat pour `level="single"` :

```
<p>1) Titre : le titre</p>
<p>1) Titre : le titre</p>
<p>2) Titre : le titre</p>
<p>1) Titre : le titre</p>
```

Résultat pour `level="any"` :

```
<p>1) Titre : le titre</p>
<p>2) Titre : le titre</p>
<p>3) Titre : le titre</p>
<p>4) Titre : le titre</p>
```

Appliquer des règles de modèle

- `xsl:apply-templates`
  par défaut, s'applique aux fils du noeud courant.
  L'attribut : `select="exprXPath"` permet de sélectionner quels descendants seront transformés (les autres étant ignorés)
- `xsl:call-template`
  une règle `xsl:template` peut être nommée par l'attribut `name` et appelée explicitement.

Appliquer des règles de modèle

On reprend l'exemple du document avec plusieurs chapitres...

```
<livre>
  <chapitre>
    <titre>un titre de chapitre</titre>
    <para>et un paragraphe</para>
    <para>un autre paragraphe</para>
    <para>et encore ...</para>
  </chapitre>
  <chapitre>...</chapitre>
</livre>
```

Appliquer des règles de modèle

```
<xsl:template match="chapitre">
  <!-- mise en forme du titre -->
  <h1><xsl:value-of select="titre" /></h1>
  <!-- et les paragraphes-->
  <xsl:apply-templates select="para"/>
</xsl:template>

<xsl:template match="para">
  <p><xsl:value-of select="." /></p>
</xsl:template>
```

Appliquer des règles de modèle

```
<xsl:template match="livre">
  <!-- une table des matières -->
  <xsl:call-template name="table-des-matieres" />
  <!-- et la transformation du livre complet-->
  <xsl:apply-templates />
</xsl:template>
```

Appliquer des règles de modèle

```
<xsl:template name="table-des-matieres">
  <ul>
    <xsl:for-each select="chapitre">
      <li><xsl:value-of select="titre" /></li>
    </xsl:for-each>
  </ul>
</xsl:template>
```

Le noeud-contexte lors de l'exécution d'une template nommée est le noeud à partir duquel on a fait l'appel de la template.
Ici, le noeud contexte est donc un noeud `livre`, ce qui permet d'accéder à ses fils `chapitre`.

Appliquer des règles de modèle

ou bien . . .

```
<xsl:template match="livre">
  <!-- une table des matières -->
  <ul>
    <xsl:apply-templates select="chapitre" mode="tdm"/>
  </ul>
  <!-- et la transformation du livre complet-->
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="chapitre" mode="tdm">
  <li><xsl:value-of select="titre" /></li>
</xsl:template>

<!-- et les deux templates pour chapitre et para -->
```

Création de constantes

- on peut nommer des constantes globales par l'élément :

```
<xsl:variable name="nomConstante"
              [select="exprXPath"]>
  contenu
</xsl:variable>
```

- le contenu de la constante est déterminé soit par l'expression select soit par le contenu (ou exclusif !)
- on peut accéder ensuite à la constante par `$nomConstante`
- la constante est globale si la déclaration s'est faite au premier niveau
- la constante est locale si la déclaration s'est faite à un deuxième niveau (dans une template).

Règles paramétrées

- Les règles `xsl:template` peuvent être paramétrées. Elles acceptent (comme élément de deuxième niveau) :

```
<xsl:param name="nomParamètre"
           [select="exprXPath"]>
  contenu
</xsl:param>
```

- le contenu du paramètre est déterminé soit par l'expression select soit par le contenu (ou exclusif !)
- on peut accéder ensuite dans la règle au paramètre par `$nomParamètre`

Règles paramétrées

Différences avec les constantes :

- la valeur spécifiée dans un paramètre est une valeur par défaut, qui sera utilisée si aucun paramètre effectif n'est transmis à la règle
- `<xsl:param>` peut se trouver également au premier niveau : il permet alors de définir des paramètres globaux à la feuille de style, qui peuvent être transmis directement à l'appel de la transformation.

```
<xsl:param name="monParametre">tata</xsl:param>
<xsl:template match="/">
  Bonjour <xsl:value-of select="$monParametre" />
</xsl:template>

xsltproc --stringparam monParametre toto
         essai.xsl essai.xml
```

Règles paramétrées

- Le passage de paramètre se fait dans un appel de règle : soit `xsl:call-template` soit `xsl:apply-template`
- La syntaxe est

```
<xsl:with-param name="nomParamètre"
          [select="exprXPath"]>
  contenu
</xsl:param>
```

- les règles syntaxiques sont les mêmes que pour `xsl:param` et `xsl:variable`.

Règles paramétrées – Exemple

```
<xsl:template match="/">
  <xsl:call-template name="fact">
    <xsl:with-param name="n" select="$pourFact" />
  </xsl:call-template>
</xsl:template>

<xsl:param name="pourFact">3</xsl:param>
```

Règles paramétrées – Exemple

```
<xsl:template name="fact">
  <xsl:param name="n">1</xsl:param>
  <xsl:param name="res">1</xsl:param>
  <xsl:choose>
    <xsl:when test="number($n)>1">
      <xsl:call-template name="fact">
        <xsl:with-param name="n" select="number($n) - 1"/>
        <xsl:with-param name="res"
             select="number($res) * number($n)"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>Résultat : <xsl:value-of select="$res" /
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Quelques critères d'utilisation

Règles nommées, paramétrées, par match, for-each...Que choisir ?

Il faut essayer de limiter les risques d'un débogage long et fastidieux, et privilégier la maintenabilité du code

- Si la transformation implique des notions de position : privilégier une structure `for-each` permet de s'assurer que le contexte est bien celui attendu ;
- Les templates nommées doivent être utilisées à bon escient : le contexte peut être difficile à déterminer. L'idéal est de les réserver à une utilisation qui ne dépend pas du contexte ;
- Les templates utilisées par `select/match` et le cas échéant `mode` sont un moyen simple et classique de travailler ;
- Les templates paramétrées permettent de revenir à un mode de programmation plus classique, et permettent également une clarification des éléments utilisés pour travailler. On peut toujours l'élément lui-même en paramètre d'un appel `select="."`

Identifier des éléments

L'élément `xsl:key` permet d'associer à des éléments la clé qui permet de les identifier.

- l'attribut `name` permet de nommer la clé, ce nom sera utilisé par la fonction `key()`
- l'attribut `match` permet de sélectionner les éléments qu'on veut identifier
- l'attribut `use` permet d'indiquer ce qui doit être utilisé comme clé (généralement un attribut identifiant !)

```
<xsl:key name="idfich"
         match="fiche-observations"
 use="@id-obs" />
```

Identifier des éléments

La fonction `key(nomClé,valeur)` permet de récupérer les éléments ainsi identifiés :

```
<xsl:template match="espece">
  <xsl:copy>
  espèce <xsl:value-of select="@nom" /> observée par
  <xsl:for-each select="aussi-dans/@ref-obs">
    <xsl:text> - </xsl:text><xsl:value-of
                    select="key('idfich',.)/@auteur" />
  </xsl:for-each>
  </xsl:copy>
</xsl:template>
```

Identifier des éléments

Pour le document :

```
<oiseaux>
  <fiche-observations auteur="L.Svensson" id-obs="obs1"> ..
  <fiche-observations auteur="O.Peterson" id-obs="obs2">
    <espece nom="grive mauvis" nbre="20">
      <famille>Turdidés</famille>
      <description>en migration probablement.</description>
      <aussi-dans ref-obs="obs1" />
      <aussi-dans ref-obs="obs2" />
      <aussi-dans ref-obs="obs4" />
    </espece>
  <fiche-observations auteur="M.Yoop" id-obs="obs4"> ...</f
```

donnerait le résultat suivant :

```
<espece>espèce grive mauvis observée par
- L.Svensson - O.Peterson - M. Yoop</espece>
```

Accéder à d'autres documents

La fonction `document(nomFichier,exprXPath)` permet d'accéder aux éléments du document XML qui se trouve dans `nomFichier`.
Exemple : on dispose d'un fichier `monFichier.xml` contenant des éléments qui s'appellent `monElt`
On peut y accéder par

```
  <xsl:template match="/">
    <xsl:for-each select="document('monFichier.xml',//monEl
<xsl:value-of select="." />
    </xsl:for-each>
  </xsl:template>
```

Produire d'autres documents

L'élément `xsl:document` permet de créer un document XML dont le nomde fichier sera celui de l'attribut `href`
Exemple :

```
  <xsl:template match="/">
    <xsl:for-each select="//@id">
      <xsl:variable name="moi" select="." />
      <xsl:document href="{concat('res_',.,'.xml')}">
<unElt>
  <xsl:value-of select="//o[@id = $moi]" />
</unElt>
      </xsl:document>
    </xsl:for-each>
  </xsl:template>
```