XML et Programmation Internet

Cours 5

kn@lri.fr

Plan

`xsl:value-of`

On peut récupérer des nœuds texte dans le document source au moyen de la balise `<xsl:value-of select="xpath expr"/>`. L'expression `xpath expr` est évaluée et convertie en chaîne de caractères en suivant les conversions d'XPath (voir cours 3).

Itération avec `xsl:for-each`

La balise

```
<xsl:for-each select="xpath expr">
        corp de boucle
</xsl:for-each>
```

évalue le corp de la boucle pour chaque nœud renvoyé par l'expression `xpath expr`. À chaque itération le nœud considéré devient le nœud contexte (i.e. celui renvoyé par l'axe `self::`)

Itération ordonnée `xsl:sort`

On peut modifier l'ordre dans lequel les éléments sont parcourus par une boucle `xsl:for-each` en utilisant l'élément

```
<xsl:sort select="expression"
 lang="language-code"
 data-type="text|number|qname"
 order="ascending|descending"
 case-order="upper-first|lower-first"/>
```

qui n'est valide que comme fils d'un `xsl:for-each` ou `xsl:apply-templates`

- `select` : expression XPath selon laquelle trier
- `lang` : code de la locale selon laquelle trier (par exemple en français, `é` est avant `f`)
- `data-type` : types des données triées (text (defaut), nombre ou élément XML (ordre du document utilisé dans ce cas)
- `order` : s'il faut trier en ordre croissant ou décroissant
- `case-order` : lors d'un tri text si les majuscules sont avant les minuscules (defaut) ou inversement

Conditionnelle simple avec `xsl:if`

On peut évaluer conditionnellement une portion de code XSLT en la plaçant dans une balise:

```
<xsl:if test="expression">
...
</xsl:if>
```

Le résultat de l'expression est interprété comme un booléen (selon les règles de conversion d'XPath). S'il vaut vrai, le corp du `xsl:if` est exécuté.

Conditionnelle complexe avec `xsl:choose`

On peut écrire plusieurs portions de code, gardées par des conditions distinctes en les plaçant dans une balise:

```
<xsl:choose >
 <xsl:when test="cond 1"> code 1</xsl:when>
 <xsl:when test="cond 2"> code 2</xsl:when>
 <xsl:when test="cond 3"> code 3</xsl:when>
 ...
 <xsl:otherwise> code sinon </xsl:otherwise>
</xsl:choose>
```

Les conditions `xsl:when` sont évaluées dans l'ordre et le corp de la première valant vrai est exécuté. Si aucune condition n'est vérifiée, le corp de l'instruction `xsl:otherwise` est exécuté.

Exemple simple (génération d'une page Web)

On réutilise (encore) le fichier de recette:

```
<!DOCTYPE recette SYSTEM "recette.dtd">
<recette difficulte="facile">
<title>Tiramisú</title>
<ingredients>
  <ingredient>mascarpone</ingredient>
  <ingredient>oeufs</ingredient>
  <ingredient>sucre</ingredient>
  <ingredient>café</ingredient>
  <ingredient>biscuits</ingredient>
</ingredients>
<duree>2h</duree>
<etapes>
  <e num="1">Séparer les blancs des jaunes</e>
  <e num="2">…</e>
  …
</etapes>
</recette>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
         <title>Ingrédients du <xsl:value-of select="descendant::title" /> </title>
      </head>
      <body>
        <h1>Ingrédients du <xsl:value-of select="descendant::title" /> </h1>
        <ul>
          <xsl:for-each select="descendant::ingredient">
             <xsl:sort select="child::text()" />
            <li> <xsl:value-of select="child::text()" /> </li>
          </xsl:for-each>
        </ul>
        <p>Étape<xsl:if test="count(descendant::e) &gt;1">s</xsl:if></p>
        <ol>
          <xsl:apply-templates select="descendant::e">
             <xsl:sort select="@num" data-type="number"/>
          </xsl:apply-templates>
        </ol>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="e">
    <li><xsl:value-of select=" "/></li>
```


Plan

Nom d'attributs ou d'éléments dynamiques

Il est fréquent de vouloir choisir dynamiquement les noms des éléments. Par exemple :

```
<xsl:choose>
  <xsl:when test="condition"> <b> </xsl:when>
  <xsl:otherwise>  <i> </xsl:otherwise>
</xsl:choose>

  .... code complexe pour calculer le texte ....

 <xsl:choose>
  <xsl:when test="condition"> </b> </xsl:when>
  <xsl:otherwise>  </i> </xsl:otherwise>
</xsl:choose>
```

code incorrect : les balises `<b>` et `<i>` ne sont pas bien parenthésées

Solution naïve

On peut dupliquer le code de manière à respecter les balises ouvrantes/fermantes :

```
<xsl:choose>
  <xsl:when test="condition">
    <b>
          .... code complexe pour calculer le texte ....
    </b>
  </xsl:when>
  <xsl:otherwise>
    <i>
          .... code complexe pour calculer le texte ....
    </i>
    </xsl:otherwise>
</xsl:choose>
```

Inélégant et non-maintenable : si on a 10 cas, on copie cole 10 fois le code complexe.

Nom d'attributs ou d'éléments dynamiques

On a besoin :

- de pouvoir stocker dans des « variables » le résultat d'une expression XSLT
- de pouvoir créer des éléments ou des attributs dont le nom est une expression

Problème : XPath ne permet pas de définir des variables

Variables en XSLT (`xsl:variable`)

Comme dans tout langage, on veut pouvoir évaluer une expression et donner un nom au résultat pour pouvoir le réutiliser plusieurs fois. En XSLT, on utilise la balise `xsl:variable` :

```
<xsl:variable name="result" select="count(descendant::foo)" />

Le document contient <xsl:value-of select="$result" /> éléments foo.
```

- L'attribut `name` est obligatoire et permet de définir le nom de la variable
- L'attribut `select` s'il est présent permet de définir le contenu
- Au sein d'une expression XPath, on peut utiliser la notation `$x` pour référencer le contenu de la variable `x`

Définitions complexes de variables

On peut aussi donner du contenu à l'élément `xsl:variable`, pour définir la valeur de la variable :

```
<xsl:variable name="result">
  <xsl:choose>
    <xsl:when test="…condition…">b</xsl:when>
    <xsl:otherwise>i</xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```


Portée des variables

Les variables sont non-modifiables une fois définies. Elles sont visibles par tous les éléments suivant la balise fermante `</xsl:variable>` se trouvant avant la fermeture de l'élément contenant la variable. Exemple :

```
<xsl:template match="x">
  <xsl:variable name="total" select="count(descendant::text())" />
  <!-- la variable total est visible à partir d'ici et
       jusqu'au </xsl:template> -->


  <xsl:for-each select="descendant::text()">
    <p>texte <xsl:value-of select="position()" /> sur
      <xsl:value-of select="$total" /> :
      <xsl:value-of select="." />
    </p>
  </xsl:for-each>

</xsl:template>
```

Création dynamique d'éléments ou d'attributs

On peut utiliser les instructions `xsl:element` et `xsl:attribute` pour créer des éléments et des attributs dont le nom est calculé dynamiquement :

```
<xsl:element name="foo">
  <xsl:attribute name="bar">baz</xsl:attribute>
  ...
</xsl:element>
```

produira dans le document de sortie :

```
<foo bar='baz'>
 ...
</foo>
```


Échappement d'expressions XPath dans les chaînes

À part les attributs `select` et `test` de certaines balises XSLT, les autres attributs sont des chaines de caractères. On peut cependant intégrer des expressions XPath au moyen d'accolades `{ }`

```
<a href="descendant::url/child::text()">Le site!</a>
```

Dans le code ci-dessus, `href` étant un attribut non interprété (ce n'est pas une balise XSLT), le chemin XPath apparaîtra dans le résultat (ce qu'on ne veut pas). On peut écrire :

```
<a href="{descendant::url/child::text()}">Le site!</a>
```


Retour sur notre exemple

```
<xsl:variable name="nombalise">
  <xsl:choose>
   <xsl:when test="condition">b</xsl:when>
   <xsl:otherwise>i</xsl:otherwise>
  </xsl:choose>
</xsl:variable>

<xsl:element name="{$nombalise}">
    .... code complexe pour calculer le texte ....
</xsl:element>
```

Attention : il faut écrire `{$nombalise}` sinon le processeur XSLT essayera de créer un élément `<$nombalise>` ce qui est illégal (et provoquera une erreur)

Plan

Spécifier le type de sortie

On peut placer une balise `xsl:output` dans la balise racine de la feuille de style (`xsl:stylesheet`)

```
<xsl:output
   method="xml|html|text"
   cdata-section-elements="namelist"
   indent="yes|no"
/>
```

- `method` : décrit le type de fichier généré (XML (defaut), HTML (plus permissif sur les balises non fermées), texte)
- `cdata-section-elements` : liste de nom de balises séparées par des espaces. Dans ces balises, le contenu du texte n'est pas interprété (>, < peuvent êtres écrits directement)
- `indent` : si la sortie est du XML, indenter les balises de manière lisible

Copie de texte et de nœuds

- La balise `<xsl:copy-of select='expression'>` copie le résultat de l'expression XPath (qui peut être un nœud) dans la sortie, avec ses attributs et son contenu
- La balise `<xsl:text> ... </xsl:text>` copie le texte tel quel (sans toucher aux blancs) dans la sortie