Transformations éditoriales avancées

LHQ1C1M1
Master 2 Humanités numériques

XSLT

eXstensible Stylesheet Language Transformations

Les modes

Parfois, on a besoin d'appliquer plusieurs traitements à un même ensemble de nœuds.

Par exemple pour une sortie HTML, on peut souhaiter appliquer deux traitements pour les titres de chapitre :

- les placer dans des balises `<h2/>` pour le corps du document ;
- en faire des éléments de liste `<li/>` afin de produire automatiquement une table des matières.

Les modes

Plusieurs solutions peuvent être envisagées avec XSLT, mais l'approche modale est certainement la plus commode.

On utilise l'attribut `@mode` des éléments `<xsl:template/>` et `<xsl:apply-templates />` afin de définir plusieurs règles modèles et les appeler de manière distincte.

Les modes - définir des templates

```
<xsl:template match="div[@type='chapter']/head">
    <h2><xsl:apply-templates/></h2>
</xsl:template>

<xsl:template match="div[@type='chapter']/head" mode="toc">
    <li><xsl:apply-templates/></li>
</xsl:template>
```

Les modes - appeler les templates

```
<xsl:template match="TEI">
    <html>
        <head><title>Le dernier jour d’un condamné</title></head>
        <body>
            <ul>
                <xsl:apply-templates
                    select="//div/head" mode="toc"/>
            </ul>
            <xsl:apply-templates/>
        </body>
    </html>
</xsl:template>
```

Les tests

Les tests correspondent à une instruction conditionnelle. C'est-à-dire que l'on va tester une ou plusieurs conditions et appliquer des instructions différentes suivant le résultat du test.

Le tests font partie des structures élémentaires en algorithmique.

Il peuvent prendre exactement 2 formes : la forme *simple* et la forme *complète*.

Les tests simples

```
si booléen alors
    instruction
finsi
```

La machine examine le booléen :

- si la condition est remplie (le booléen a pour valeur `VRAI`), elle exécute les instructions ;
- si la condition n'est pas remplie (le booléen a pour valeur `FAUX`), elle passe directement aux instructions qui suivent `finsi`.

Les tests simples

En XSLT, on dispose de l'élément `<xsl:if/>` :

```
<xsl:if test="expression_booléenne">
   <!-- Mon modèle a appliquer si la condition est vraie -->
</xsl:if>
```

Le modèle contenu dans `<xsl:if/>` est instancié **si et seulement si** l'expression XPath est vraie.

Les tests simples - exemple

```
<xsl:template match="p">
    <p>
        <xsl:if test="@rend">
            <xsl:attribute name="class" select="@rend"/>
        </xsl:if>
        <xsl:apply-templates/>
    </p>
</xsl:template>
```

Si la balise <p/> dispose d'un attribut @rend, alors on ajoute en sortie un attribut @class ayant pour valeur celle de @rend.

Les tests complets

```
si booléen alors
    instruction 1
sinon
    instruction 2
finsi
```

- si le booléen a pour valeur VRAI, l'instruction 1 est exécutée ;
- si la booléen a pour valeur FAUX, l'instruction 2 est exécutée.

En réalité, la première forme correspond au cas de figure où l'une des deux "branches" est vide. Plutôt que d'indiquer à la machine de ne rien faire, on n'écrit rien.

Les tests complets

Pour effectuer des tests complets en XSLT, on utilise l'élément `<xsl:choose/>`.

L'élément `<xsl:choose/>` contient :

- 1 ou n `<xsl:when/>`
- 0 ou 1 `<xsl:otherwise/>`

Les éléments `<xsl:when/>` disposent d'un attribut `@test`. Le premier qui voit son test vérifié est exécuté.

Si aucun test n'est vérifié mais qu'une instruction `<xsl:otherwise/>` est présente, alors c'est cette dernière qui sera exécutée.

Les tests complets - exemple

```
<xsl:template match="head">
    <xsl:choose>
        <xsl:when test="parent::div[@type = 'book']">
            <h1><xsl:apply-templates/></h1>
        </xsl:when>
        <xsl:when test="parent::div[@type = 'chapter']">
            <h2><xsl:apply-templates/></h2>
        </xsl:when>
        <xsl:otherwise>
            <h3><xsl:apply-templates/></h3>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Les boucles

Les boucles font également partie des structures élémentaires en algorithmique. Elles sont d'ailleurs propre à la programmation et n'existent pas ailleurs.

Elles permettent d'exécuter un ensemble d'instructions de manière répétée. On parle aussi de **structures itératives**.

Il en existe 2 types :

- les boucles `while` (« tant que » avec la variante do…`while` « faire…tant que »)
- les bouches `for` (« pour »)

Les boucles `while`

Les boucles `while` sont utilisées lorsqu'on ne peut pas déterminer à l'avance le nombre d'itérations nécessaires :

```
TANTQUE <expression booléenne> FAIRE
    instruction
    relance
FINTANTQUE
```

Les boucles `for`

À l'inverse, les boucles `for` sont utilisées pour appliquer des traitements aux éléments d'un ensemble dont on peut connaitre à l'avance la quantité.

```
POUR compteur <- valeur initiale à valeur finale (PAS valeur du pas)
    instruction
FINPOUR
```

Les boucles et XSLT

XSLT est un langage de programmation *fonctionnel*, et en tant que tel, il diffère des langages de programmation *procéduraux*.

Cet approche *fonctionnelle* permet bien souvent de se passer des boucles en utilisant simplement des règles modèles ; mais voyons tout de même ce que permet de faire l'élément `<xsl:for-each/>`.

Les boucles <xsl:for-each/>

<xsl:for-each/> permet de répéter une instruction sur une séquence d'items identifiés par l'attribut @select.

```
<xsl:template match="list">
   <ul>
      <xsl:for-each select="item">
         <li><xsl:apply-templates select="."/></li>
      </xsl:for-each>
   </ul>
</xsl:template>
```

Les boucles <xsl:for-each/> - l'approche fonctionnelle

```
<xsl:template match="list">
    <ul>
        <xsl:apply-templates/>
    </ul>
</xsl:template>

<xsl:template match="item">
    <li>
        <xsl:apply-templates/>
    </li>
</xsl:template>
```