

Transformations éditoriales avancées

LHQ1C1M1

Master 2 Humanités numériques

XSLT

eXtensible 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 `` 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[@type='chapter']/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 sont l'une des 4 structures élémentaires en algorithmique.

Les tests simples

Les tests peuvent prendre exactement 2 formes :

1re forme simple

```
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, la première forme correspond à l'élément `<xsl:if/>` :

```
<xsl:if test="expression_booléenne">  
  <xsl:apply-templates/>  
</xsl:if>
```

L'instruction contenue dans `<xsl:if/>` est instanciée 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

2e forme complète

```
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="hi">
  <xsl:choose>
    <xsl:when test="@rend = 'bold'">
      <strong><xsl:apply-templates/></strong>
    </xsl:when>
    <xsl:when test="@rend = 'italic'">
      <em><xsl:apply-templates/></em>
    </xsl:when>
    <xsl:when test="@rend = 'superscript'">
      <sup><xsl:apply-templates/></sup>
    </xsl:when>
    <xsl:otherwise>
      <span class="{ @rend }"><xsl:apply-templates/></span>
    </xsl:otherwise>
  </xsl:choose>
```

Les boucles

Les boucles sont également une des 4 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 boucles `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 connaître 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 permettent de faire les éléments `<xsl:for-each/>` et `<xsl:for-each-group/>`.

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>
```

@todo ajouter une dia en faisant la même chose mais uniquement avec des règles modèles?

Les boucles `<xsl:for-each-group/>`

L'instruction `<xsl:for-each-group/>` permet d'itérer non pas sur un ensemble de nœuds, mais sur des **groupes** de nœuds.

- l'attribut `@select` identifie les nœuds à grouper;
- les attributs `@group-by`, `@group-adjacent`, `@group-starting-with` et `@group-ending-with` indiquent comment les grouper.
- la fonction `current-groupin-key()` retourne la clé de regroupement (si `@group-by`);
- la fonction `current-group()` retourne le groupe courant.

Trier avec `<xsl:sort/>`

L'instruction `<xsl:sort/>` permet de trier des nœuds afin, par exemple de les ordonner alphabétiquement.

Il s'utilise comme enfant des éléments `<xsl:apply-templates/>` et `<xsl:for-each/>`.

- l'attribut `@select` correspond à la clé de tri;
- l'attribut `@order` définit le sens du tri (ascendant ou descendant)
- l'attribut `@data-type` permet de préciser si l'on souhaite un tri alphabétique ou numérique

Trier avec <xsl:sort/>

```
<list>
```

```
  <item>c</item>
```

```
  <item>a</item>
```

```
  <item>b</item>
```

```
</list>
```

```
<xsl:template match="list">
```

```
  <xsl:apply-templates select="item">
```

```
    <xsl:sort select="."/>
```

```
  </xsl:apply-templates>
```

```
</xsl:template>
```

résultat : abc