XSLT-XPATH (2)

Claude Moulin

claude.moulin@utc.fr

Parcours

- Un parcours est une séquence de pas. Chaque pas est séparé par /.
- Exemple :
  - Le second élément `section` du cinquième élément `chapitre` du document.
    - `//chapitre[5]/section[2]`
  - L'élément `url` fils de l'élément `java`, fils du noeud courant, dont l'attribut `keyname` a comme valeur `ncsa`
    - `java/url[@keyname = "ncsa"]`

Test sur les noeuds

- Les tests sur les nœuds permettent de construire un ensemble de noeuds.
  - *
    - Sélectionne les nœuds du type principal de l'axe.
  - comment()
    - Sélectionne les nœuds commentaire.
  - node()
    - Sélectionne tous les nœuds sauf les attributs.
  - text()
    - Sélectionne les nœuds texte.
  - <nom>
    - Sélectionne les nœuds selon le <nom>.

Fonctions

- Les fonctions permettent d'écrire des expressions plus sophistiquées et sont utilisées dans les recommandations basée sur Xpath.
- Fonctions et opérateurs
  - Ensemble de noeuds
  - Chaîne
  - Opérateurs booléens
  - Numérique

Fonctions : ensemble de noeuds

- *number* **last**()
  - Retourne un nombre égal à la position du dernier nœud.
- *number* **position**()
  - Retourne la position d'un noeud.
- *number* **count**(*node-set*)
  - Retourne le nombre de nœuds sélectionnés par le paramètre.
- *node-set* **id**(*object*)
  - Retourne l'élément dont l'ID est égale à la valeur du paramètre.
  - Nécessite une référence à la DTD pour indiquer le type ID
- *string* **local-name**(*node-set*?)
  - Retourne le nom du premier élément sélectionné par le paramètre.

Quelques fonctions chaîne

- *string* **string**(*object*?)
  - Convertit un objet en chaîne.
- *string* **concat**(*string*, *string*, *string**)
  - Concatène des chaînes.
- *string* **substring**(*string*, *number*, *number*?)
  - Retourne une sous-chaîne selon les paramètres.
- *string* **substring-after**(*string*, *string*)
  - Retourne les caractères qui suivent certains caractères d'une chaîne.

Opérateurs booléens

- Les expressions booléennes se forment en utilisant :
  - les opérateurs de relation :
    - =, !, <, >, <=, >=
  - les opérateurs booléens :
    - not, and, or
- On doit utiliser les entités > et < dans certaines expressions au lieu de > et <.

Fonctions, opérateurs numériques

- Operateurs :

  +, -, *, mod, div

- Fonctions :
  - *number* **sum**(*node-set*)
  - *number* **floor**(*number*)
  - *number* **ceiling**(*number*)
  - *number* **round**(*number*)

Exemples

- Le dernier élément `url` fils du nœud courant
  - `child::url[position()=last()]`
- Le premier frère qui suit
  - `following-sibling::*[position() = 3 mod 2 ]`
  - Forme abrégée : `following-sibling::*[1]`
- Le premier fils `chapitre` ou `section`
  - `*[self::chapitre or self::section][1]`
    - `*` sélectionne tous les fils
    - `self::chapitre or self::section` sélectionne ceux qui se nomment `chapitre` ou `section`
    - `[1]` sélectionne le premier (il peut s'appeler `chapitre` ou `section`)

Variables

- C'est un identificateur auquel est associée une valeur.
- La valeur ne peut être assignée qu'une seule fois.
  - Les essais de donner une autre valeur n'ont pas d'effet.
  - Il s'agit davantage de constantes que de variables.
- Une variable est visible à l'intérieur de l'élément dans lequel elle est définie et dans ses sous-éléments.
- Pour obtenir la valeur de la variable on fait précéder son identificateur du symbole "$".

Exemple d'usage de xsl:variable

```
<xsl:template match = "...">
  <xsl:variable name="id_ensg">untel</xsl:variable>
  Le nom de l'enseignant identifié par :
    <xsl:value-of select="$id_ensg" /></div>
    <xsl:value-of
        select="//enseignant[@id=$id_ensg]/nom" />
</xsl:template>
```

Procédures sans paramètres

- Une procédure est une séquence d'instructions contenue dans un bloc défini par un xsl:template et identifié par un nom.
  - Utiliser l'attribut `name` de `xsl:template`
- Définition de la procédure.

```
<xsl:template name = "template_name">
    …
<xsl:template/>
```

- Appel de la procédure

```
<xsl:call-template name="template_name"/>
```

Procédures avec paramètre(s)

- Définition

```
<xsl:template name = "nom_procedure">
   <xsl:param name = "nom_du_parametre"/>
… … … <!-- avec l'utilisation du parametre -->
</xsl:template>
```

- Appel

```
<xsl:call-template name="nom_procedure">
   <xsl:with-param name="nom_du_parametre" select =
   "expression" />
</xsl:call-template>
```

Utilisation des procédures

- Les procédures sont écrites une fois et sont appelées n fois
- La mise au point est facilitée
- Il est possible de concevoir des bibliothèques de procédures

Exemple d'utilisation

```
<xsl:template name="affiche_enseignant" >
  <xsl:param name = "id_ensg" />
  <xsl:value-of
      select="//enseignant[@id = $id_ensg]/nom" />
</xsl:template>

<!- ref est une variable -->
<div>Le nom de l'enseignant identifié par
   <xsl:value-of select="$ref" />
</div>
<xsl:call-template name="affiche_enseignant" >
    <xsl:with-param name="id_ensg" select="$ref" />
</xsl:call-template>
```

Template avec paramètres - 1

- Appel du template

```
<xsl:apply-templates select="//responsable">
  <xsl:with-param name="res">moulin</xsl:with-
 param>
</xsl:apply-templates>
```

Template avec paramètres - 2

- Règle XSLT

```
<xsl:template match="responsable">
  <xsl:param name="res“ />
  <xsl:if test="not ($res = ‘’) "> --- <xsl:value-of
    select="$res"/> --- </xsl:if>
    … …
</xsl:template>
```

Evaluer une expression

- Les expressions sont évaluées dans les attributs *select*, *test*, etc. des instructions xslt, mais ne le sont pas dans les éléments insérés dans l'arbre construit.
  - Dans ce cas les accolades s'utilisent pour indiquer que l'expression doit être évaluée.
- Exemple.
  - `<a href="{.}"> Lien </a>` : indique que l'attribut `href` assumera la valeur du contenu du noeud courant.
- L'évaluation s'utilise lorsque l'expression à insérer est simple
  - Dans certains cas cette façon de faire n'est pas possible

Exemples

- `<a href="{.}">site</a>`
  - le noeud courant est une URL
- `<a href="{/uv/site}">site</a>`
  - Le nœud courant est quelconque puisque le chemin est absolu.
- `<a href="{./url}" id="{./url/@keyname}">site</a>`
  - Le nœud courant est parent d'un élément `url` et on ajoute la valeur de son attribut `keyname` comme identificateur.
- `<xsl:variable name="my_url">http://… …</xsl:variable>`

  `<a href="{$my_url}">site</a>`
  - Utilisation de la valeur de la variable `my_url`

Ajouter un attribut à un élément

- Modèle :

```
<xsl:attribute name="attribute_name">
    value_attribute
</xsl:attribute>
```

- Sert à ajouter un attribut à un élément de l'arbre résultant.
- Exemple :

```
<a>
   <xsl:attribute name="href">
            <xsl:value-of select="."/>
   </xsl:attribute>
   Site
</a>
```

Tri

- Il est possible de trier les noeuds sélectionnés par apply-templates ou for-each, par ordre croissant (par défaut) ou décroissant

```
<xsl:for-each select = "expr">
   <xsl:sort  select = "expr" order="descending"/>
   <!-- corps de la boucle -->
</xsl:template>
```

Clé

- Elles permettent de définir un index pour un groupe d'éléments à partir par exemple de la valeur de certains attributs.
- Exemple : utiliser l'attribut *id* des éléments *enseignant* comme index.
- Définition de la clé (avant la première règle) :

```
<xsl:key name="idkey" match="enseignant"   use="@id" />
```

- Utilisation

```
<xsl:value-of select="key('idkey',$refenseignant)/nom" />
```

Paramètre d'une feuille

- Il est possible de passer un (des) paramètre(s) à une feuille de style au moment de l'appel de la transformation
- Déclaration du paramètre (avant la première règle)
  - <xsl:param name="nom_param" />
- Utilisation
  - $nom_param donne la valeur du paramètre

Préséance des règles

- Plusieurs règles peuvent correspondre à la sélection d'un même élément.
- Des éléments de même nom peuvent apparaître à des endroits différents du document
- Règle générale :
  - La règle désignant le noeud avec plus de précision a la préséance sur les autres.
  - enseignants/enseignant est plus précis que enseignant

Créer textes et commentaires

- **`xsl:text`**
  - Crée un nœud de texte dans l'arbre résultant.
  - Les caractères particuliers (<,>, &, etc.) seront remplacés par leur entités (< , > , &emb; , etc.) si l'attribut `disable-output-escaping="yes".`
  - Exemple : `<xsl:text> … </xsl:text>`

- **`xsl:comment`**
  - Crée un noeud de type commentaire.
  - Exemple : `<xsl:comment> … </xsl:comment>`

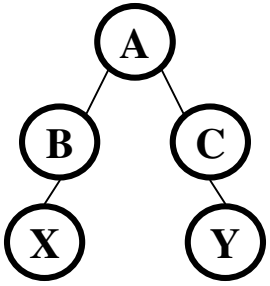
Inclusion de feuilles de style

- **`xsl:include`**
  - A l'intérieur de la feuille de style incluant sont insérés les éléments des feuilles incluses.
  - Les éléments de la feuille de style incluant ont la précédence sur ceux qui sont inclus.
  - L'attribut `href`: identifie le fichier de la feuille qui doit être inclus.
- Exemple :

```
<xsl:include href="filename.xsl"/>
```

Importation de feuilles de style

- **`xsl:import`**
  - Similaire à l'inclusion mais porte à la création d'un arbre qui permet de déterminer les règles de préséance pour l'élaboration des éléments.
  - Si A importe B avant C les règles de C ont préséance sur celle de B
  - Si A importe B puis C, si B importe X et C importe Y. L'ordre décroissant sera : A, C, Y, B, X.



- Exemple :

```
<xsl:import href = "filename.xsl"/>
```

Combinaison de feuilles de style

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3c.org/1999/XSL/Transform">
     <xsl:import href="fileA.xsl"/>
     <xsl:import href="fileB.xsl"/>
     <xsl:include href="fileC.xsl"/>
     …
</xsl:stylesheet>
```

- Les « import » s'insèrent avant les règles XSLT.
- xsl:import doit précéder xsl:include.
- Sont exécutées d'abord les inclusions et ensuite les importations.
- Ne pas inclure directement ou indirectement la feuille qui inclut les autres.