Transformations éditoriales avancées

LHQ1C1M1
Master 2 Humanités numériques

XSLT

eXstensible Stylesheet Language Transformations

Espace de nom XPath par défaut

Pour déclarer un espace de nom XPath par défaut, on utilise l'attribut `@xpath-default-namespace`. Il n'est alors pas nécessaire de préfixer les expressions XPath avec `*:` (ou `tei:` si utilisation de `xmlns:tei`)

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xpath-default-namespace="http://www.tei-c.org/ns/1.0"
   exclude-result-prefixes="xs"
   version="2.0">
</xsl:stylesheet>
```

Paramètres de sortie - <xsl:output/>

L'élément <xsl:output/> permet de configurer les paramètres de sortie :

- @method : format de sortie ("xml" par défaut | "html" | "xhtml" | "text" | "json")
- @encoding : encodage des caractères ("UFT-8")
- @indent : règle d'indentation ("true" | "false" | "1" | "0")

Liste des autres paramètres

<xsl:result-document/>

L'élément `<xsl:result-document/>` est utilisé pour diriger la sortie vers une destination secondaire, par exemple un fichier, un mail, une URI, etc.

Il est très intéressant pour produire plusieurs fichiers automatiquement à partir d'une même source.

<xsl:result-document/>

```
<xsl:template match="div[@type='chapter']">
    <xsl:result-document href="{./@xml:id}.html" method="xhtml">
        <html>
            <head><xsl:value-of select="head"/></head>
            <body>
                <xsl:apply-templates/>
            </body>
        </html>
        <xsl:apply-templates/>
    </xsl:result-document>
</xsl:template>
```

<xsl:call-template/>

L'instruction `call-template` permet d'appeler un template par son nom :

```
<xsl:template name='monTemplate'>
    <!-- modèle -->
</xsl:template>

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

<xsl:call-template/>

Cette méthode est très utile, car elle permet de hiérarchiser les feuilles XSLT en créant des templates avec un rôle bien défini.

Par exemple, nous avons vu qu'avec l'instruction `<xsl:result-document/>` il était possible de générer un site composé de plusieurs pages à partir d'une unique source XML.

<xsl:call-template/>

```
<xsl:template match="/">
    <xsl:result-document href="index.html">
        <html><!-- … --></html>
    </xsl:result-document>
    <xsl:for-each select="./descendant::TEI">
        <xsl:result-document href="{./@xml:id}.html">
            <html><!-- … --></html>
        </xsl:result-document>
    </xsl:for-each>
    <xsl:result-document href="about.html">
        <html><!-- … --></html>
    </xsl:result-document>
</xsl:template>
```

<xsl:call-template/>

Toutefois, cette méthode est très verbeuse : p. ex. on répète les éléments de structure HTML alors qu’ils pourraient être partagés.

<xsl:call-template/> - exemple simple

`call-template` agit comme `apply-templates` mais pour un template *nommé* :

```
<xsl:template match="/">
    <xsl:call-template name="content"/>
</xsl:template>
<xsl:template name="content">
    <p>Hello World!</p>
</xsl:template>
```

<xsl:call-template/> - <xsl:with-param/>

L'instruction `with-param` permet de passer des paramètres au modèle, qui les reçoit par l'intermédiaire de `param` :

```
<xsl:template match="/">
    <xsl:call-template name="content">
        <xsl:with-param name="name" select="'John'"/>
    </xsl:call-template>
</xsl:template>
<xsl:template name="content">
    <xsl:param name="name"/>
    Bonjour <xsl:value-of select="$name"/> ! 
</xsl:template>
```

Copier <xsl:copy-of/> et <xsl:copy-of/>

L'élément <xsl:copy-of/> permet d'ajouter dans l'arbre de sortie un élément de l'arbre d'entrée, sans le modifier.

```
<xsl:template match="ref">
    <a href="{ ./@target }"><xsl:apply-templates/></a>
    <xsl:copy-of select="."/>
</xsl:template>

<xsl:template match="emph">
    <em><xsl:apply-templates/></em>
</xsl:template>
```

Résultat :

```
<a href="www.w3.org/TR/xslt-30/">The <em>XSLT</em> Standard</a>
<ref target="www.w3.org/TR/xslt-30/">The <emph>XSLT</emph> Standa
```

Copier <xsl:copy-of/> et <xsl:copy-of/>

L'élément <xsl:copy/> crée une paire de balises (ouvrante et fermante) dont le nom est celui de l'élément courant, mais ni les nœuds descendants, ni les attributs ne sont copiés.

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

<xsl:template match="emph">
    <em><xsl:apply-templates/></em>
</xsl:template>
```

Résultat :

```
<ref>The <em>XSLT</em> Standard</ref>
```

Copier <xsl:copy-of/> et <xsl:copy-of/> - Copie intégrale

```
<xsl:template match="node() | @*">
    <xsl:copy>
        <xsl:apply-templates match="node() | @*"/>
    </xsl:copy>
</xsl:template>
```

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` définie 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

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.

Chapitres absents de cette introduction à XSLT :

- les fonctions
- transformations XML vers XML
- transformations XML vers texte (et LaTeX)