Introduction à XSLT

par Victor Stinner

Date de publication : 16/09/2003

Dernière mise à jour : 01/04/2005

Tutoriel d'introduction pour XSLT (eXtensible Stylesheet Language Transformation). Téléchargez la version PDF. Téléchargez la version HTML/ZIP.

I - Introduction

XSLT est l'abréviation de Extensible Stylesheet Language Transformation. C'est un langage de programmation qui sert à transformer des documents XML dans divers formats comme le HTML et ... le XML :-)

XSLT possède de nombreuses fonctions de traitement qui en font un langage de programmation complet. On peut créer des "fonctions", des boucles, calculer un maximum, faire des recherches dans un document XML, compter le nombre de résultats, etc. Mais XSLT est avant tout orienté vers le traitement d'un fichier XML. On va appliquer des modèles (templates) sur les balises XML, puis leur appliquer des traitements divers.

Cet article est une introduction à XSLT, pour les fonctions plus "avancées", voyez mon article Programmer en XSLT.

II - Exemple

II.A - Documents et codes

Documents XML source (liste.xml) :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<liste_nombres>
  <nombre valeur="10">dix</nombre>
  <nombre valeur="0">zéro</nombre>
  <nombre valeur="33">trente trois</nombre>
  <nombre valeur="6">le premier nombre parfait</nombre>
</liste_nombres>
```

Feuille de style XSLT (xslt.xsl) :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output
  method="html"
  encoding="ISO-8859-1"
  doctype-public="-//W3C//DTD HTML 4.01//EN"
  doctype-system="http://www.w3.org/TR/html4/strict.dtd"
  indent="yes" />

<xsl:template match="liste_nombres">
  <html><body>
    <p>Liste de nombres :</p>
    <ul>
      <xsl:apply-templates select="nombre" />
    </ul>
  </body></html>
</xsl:template>

<xsl:template match="nombre">
  <li>
    <xsl:value-of select="@valeur" />
    <xsl:text> : </xsl:text>
    <xsl:value-of select="." />
  </li>
</xsl:template>

</xsl:stylesheet>
```

En PHP, vous pouvez utiliser ce script :

```
<?
// Crée le processeur XSLT
$xh = xslt_create();
xslt_set_base ($xh, 'file://' . getcwd () . '/');

// Traite le document, puis affiche le résultat
$result = xslt_process($xh, 'liste.xml', 'xslt.xsl');
if (!$result)
  echo ("Erreur XSLT ...");
else
  echo ($result);

// Détruit le processeur XSLT
xslt_free($xh);
?>
```

II.B - Résultat attendu

Vous devriez obtenir le résultat suivant :

En code HTML, ça donne :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <body>
    <p>Liste de nombres :</p>
    <ul>
      <li>10 : dix</li>
      <li>0 : zéro</li>
      <li>33 : trente trois</li>
      <li>6 : le premier nombre parfait</li>
    </ul>
  </body>
</html>
```

II.C - Explication

Reprenons le code XSLT en détail. Déjà, on peut constater qu'une feuille XSLT est écrite en ... XML ;-)

Code	Explication
`<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`	Cette balise contient la version de XSLT utilisée (1.0 ici), ainsi qu'un lien vers l'espace de nom (namespace, abrégé en ns, xslns). Pour utiliser la version 1.0, vous devez ABSOLUMENT utiliser ce lien. En décodé, copiez/collez cette ligne sans vous poser de question ;-) On a ensuite la balise :
`<xsl:output method="html" encoding="ISO-8859-1" doctype-public="-//W3C//DTD HTML 4.01//EN" doctype-system="http://www.w3.org/TR/html4/strict.dtd" indent="yes" />`	Cette balise indique le format de sortie. Ici je désire du HTML, encodé en ISO-8859-1. "doctype-public" est le nom du standard respecté (j'ai choisi le HTML 4.01). "doctype-system" est un lien vers la DTD de ce standard. Référez-vous à W3.org pour choisir votre standard et indiquer le bon lien. Enfin, "indent=yes" indique que le fichier généré sera automatiquement indenté. Pas la peine dans de s'amuser à indenter le code HTML à l'intérieur de la feuille de style XSLT : le moteur XSLT s'en occupe ;-) En passant, désactiver l'indentation diminue la taille des fichiers générés ... A vous de voir. On a passé la partie bien barbante, entrons dans le vif du sujet !!!
`<xsl:template match="liste_nombres">`	Cette balise est une fonction XSLT qui va être appelée chaque fois que la balise liste_nombres est rencontrée (match="liste_nombres"). On appelle cela un modèle (template). Comme c'est notre balise racine, on va

Code	Explication
	devoir écrire l'entête HTML à cet endroit là.
`<xsl:apply-templates select="nombre" />`	Une autre fonction XSLT qui "appelle" les autres modèles pour les balises filles nommées nombre (select="nombre"). Plus généralement, on aurait pu appliquer les autres modèles à tous les fils avec select="*" (* = n'importe quelle balise fille).
`<xsl:value-of select="@valeur" />`	On va ici lire le contenu de l'attribut nommé valeur. De la même manière, on lit le contenu (en dehors des balises filles) par la requête select="." (. étant le chemin courant = le noeud courant de l'arbre XML).

II.D - Trions le résultat

Cet exemple n'est pas très utile. Alors rendons le plus attractif en triant le résultat. Il suffit de changer deux lignes de code ! Il suffit de remplacer :

```
<xsl:apply-templates select="nombre" />
```

par

```
<xsl:for-each select="nombre">
<xsl:sort select="@valeur" />
  <xsl:apply-templates select="." />
</xsl:for-each>
```

Résultat :

Ca marche ... Mais ce n'est pas ce qu'on veut. Les nombres sont triés caractères par caractères (0, 1, 3, 6 : ils sont dans l'ordre). Il faut alors corriger en spécifiant le type des données : on a des nombres !

```
<xsl:sort select="@valeur" data-type="number" />
```

Et voilà le travail ;-) Trop facile ... Plus fort, on peut inverser la liste en ajoutant l'attribut order="descending" (ordre décroissant) à la balise sort.

Alors, vous commencez à percevoir l'intérêt de XML et XSLT ?

III - Recopier balises et textes

III.A - Code naïf

Si vous ne voulez pas écrire un modèle pour toutes les balises, il faudra tôt au tard écrire deux modèles génériques : un pour les attributs, un autre pour les balises. Ce qui nous donne :

```
<xsl:template match="@*">
  <xsl:copy />
</xsl:template>

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

III.B - Problème et solution

Si vous utilisez le code donné plus haut, vous aurez des problèmes de namespace (espace de nom) avec des balises qui ressemblent à ça :#

```
<div xmlns:xsl="http://www.w3.org/1999/XSL/Transform" class="baniere">
```

Ceci est du au fait que la balise xsl:copy copie également l'espace de nom du moteur XSLT ... Solution :

```
<xsl:template match="@*">
  <xsl:copy />
</xsl:template>

<xsl:template match="*">
  <xsl:element name="{name()}" >
    <xsl:apply-templates select="* | text() | @*"/>
  </xsl:element>
</xsl:template>1
```

IV - Problèmes courants

IV.A - Eviter un peu la lourdeur du XSLT grâce aux accolades

Quand j'ai écrit mes premières pages XSLT, j'écrivais très proprement mes balises :

```
<xsl:element name="a">
  <xsl:attribute name="href">
    <xsl:value-of select="lien" />
  </xsl:attribute>
  <xsl:value-of select="texte" />
</xsl:element>
```

Je sais pas ce que vous en dites, mais perso, je trouve ça très lourd juste pour écrire un lien ! La solution : les accolades. Ces dernières dans la valeur d'un attribut remplacent un xsl:value-of. Exemple équivalent à l'exemple ci-dessus :

```
<a href="{lien}">
  <xsl:value-of select="texte" />
</a>
```

C'est plus court, nan ? On peut faire beaucoup de chose dans des accolades, mais pas tout bien sûr (pas de xsl:if par exemple). On peut par exemple utiliser la concaténation : href="{concat($repertoire,'/',$nomfich)}". Je n'ai pas poussé les tests trop loin, mais je pense que toutes les fonctions XPATH y sont utilisables ;-)

IV.B - Supprimer les espaces dûs à l'indentation XML

Quand j'écris un document XML, j'aime bien indenter mon code pour avoir une bonne lisibilité. Mais le problème est que dans certains cas, on aimerait bien supprimer ces espaces ! Exemple tout bête :

```
<p>
  <a href="http://www.developpez.com">Developpez.com</a> :
  Le paradis des développeurs !
</p>
```

Sans aucune modification, le code HTML généré sera (indentation XSLT activée) :

```
<html>
  <body>
    <p>
  <a href="http://www.developpez.com">Developpez.com</a> :
  Le paradis des développeurs !
</p>
  </body>
</html>
```

Ce n'est pas très joli :-/ Il existe alors deux commandes XSLT pour remédier à ce problèmes :

```
<xsl:strip-space elements="*" />
<xsl:preserve-space elements="pre | code" />
```

"strip-space" va supprimer les espaces inutiles de toutes les balises. Mais les balises HTML pre et code doivent conserver leur indentation, on les écrit alors dans la commande XSLT "preserve-space" (on sépare les balises par

le caractère '|'). Par défaut, tous les espaces sont conservés.

IV.C - Supprimer les espaces dûs à l'indentation XSLT

De la même manière qu'en XML, j'aime bien indenter mon code XSLT. Exemple :

```
<!-- Version indigeste -->
<xsl:template match="lien">
  <a href="{href}"><xsl:value-of select="nom" /></a> : <xsl:value-of select="description" />
</xsl:template>

<!-- Version indentée -->
<xsl:template match="lien">
  <a href="{href}">
    <xsl:value-of select="nom" />
  </a>
  :
  <xsl:value-of select="description" />
</xsl:template>
```

Le problème est que la sortie donne quelque chose ressemblant à :

```
<a href="http://www.developpez.com">Developpez.com</a>
  :
  Le paradis des développeurs !
```

La solution : utiliser la balises XSLT "xsl:text". Cette balise permet d'écrire proprement du texte à l'intérieur d'une balise. L'idéal serait de n'utiliser que cette balise, mais en pratique on ne l'utilise que lorsqu'elle est vraiment nécessaire (pas folle l'abeille !). Ce qui donne le code XSLT final :

```
<xsl:template match="lien">
  <a href="{href}">
    <xsl:value-of select="nom" />
  </a>
  <xsl:text> : </xsl:text>
  <xsl:value-of select="description" />
</xsl:template>
```

V - Conclusion

Nous avons fait un tour rapide de XSLT, et de quelques problèmes courants rencontrés. Vous avez pu voir comment ça marche, et vous avez pu apercevoir la puissance de XSLT. Mais lisez vite mon article Programmer en XSLT pour voir ce que XSLT a vraiment dans le ventre ;-)