Compléments de cours sur XSLT

Objectif

Ce document a pour but de présenter quelques notions supplémentaires sur XSLT, nécessaires en particulier pour les parties 15 et 16 des TP et/ou les examens d'XML.

Plan du document

- variables globales et locales
- paramètres
- modèles «fonctions»
- inclusion d'une feuille xslt dans une autre feuille xslt
- pseudo-tableaux.

Introduction

Toutes les notions présentées dans ce document ont un point commun entre elles : elles présentent toutes des façons de factoriser ou réutiliser des modèles xslt facilement et efficacement. La dernière fait exception : elle montre comment «tricher» en utilisant des «faux tableaux».

Nous verrons tout d'abord les variables qui sont en fait des constantes et qui fonctionnent à peu près comme les constantes de n'importe quel langage.

Puis, nous introduirons les paramètres qui permettront notamment de modifier le comportement d'un template appelé.

Ensuite, nous introduirons les notions de «fonctions» en xslt. Celles-ci peuvent être «appelées» à l'aide de variables et d'éléments call-template.

Nous verrons comme inclure une feuille xslt dans d'autres feuilles.

Enfin, nous montrerons comment utiliser des espèces de tableaux pour faire correspondre une valeur à une autre.

xsl:variable / variables globales et locales

Syntaxe pour la déclaration

```
<xsl:variable name='mon_nom' select='ma_valeur'/>
```

ou

```
<xsl:variable name='nom_nom'>
      <xsl:value-of select='...'/>
</xsl:variable>
```

attention aux pièges :

```
<xsl:variable name='mon_nom' select='ma_valeur'/>
```

est différent de :

```
<xsl:variable name='mon_nom' select='«ma_valeur»'/>
```

Dans le premier cas, `mon_nom` vaut `ma_valeur` (qui peut être un ensemble de noeuds). Dans le deuxième cas, `mon_nom` vaut la chaîne de caractères `«ma_valeur»`.

Syntaxe pour l'utilisation

`$mon_nom` : lorsque la variable est remplacée par son contenu dans le traitement xslt

`{$mon_nom}` : lorsque la variable doit être remplacée par le texte de son contenu dans le résultat (en particulier html)

Variables globales et locales

Une variable est globale si l'élément `xsl:variable` est directement sous-élément de `xsl:stylesheet`. La variable globale peut être utilisée dans tout le document. On les utilise souvent pour stocker des informations d'apparence (couleurs, formats...) ou des informations qui vont conditionner l'ensemble du résultat.

Une variable est locale si l'élément `xsl:variable` est sous-élément d'un élément `xsl:template`. Elle ne peut être utilisée que dans ce modèle. On les utilise surtout pour factoriser.

Exemple et démonstration

Les fichiers `variable.xsl` et `disques1.xml` montrent l'utilisation de deux variables globales et d'une variable locale.

`color` et `color2` sont deux variables globales d'apparence. Pour modifier les couleurs de la page résultat, il n'est donc plus nécessaire de modifier toutes les occurrences, d'en oublier, etc. Il suffit simplement de modifier le contenu de la variable. Notez dans l'utilisation, que ce qui est stocké c'est la chaîne de caractères `«red»` ou `«blue»`. Notez également dans l'utilisation, qu'on utilise `{$color}` pour préciser qu'on veut recopier le contenu de la chaîne de caractères dans le résultat html.

Le modèle `«/»` contient une variable locale `disques`. On l'utilise ici pour factoriser, et ne pas avoir besoin de réécrire cette sélection avec le prédicat. Ici, la variable ne représente pas une chaîne de caractères, mais un ensemble de noeuds `d:disque`. Pour l'utiliser dans la transformation, on appelle simplement `$disques`, qui est ici remplacé par `//d:disque[starts-with(d:titre, "B")]` par exemple.

un peu plus sur xsl:variable...

Dans la page précédente, nous avons deux syntaxes différentes pour `xsl:variable`. En fait, l'élément `xsl:variable` peut contenir d'autres sous-éléments que `xsl:value-of` (à condition qu'il ne possède pas l'attribut `select`). Il peut par exemple avoir un sous-élément `xsl:choose`, ce qui permettra d'avoir une «vraie» variable :-)

Exemple et démonstration

Les fichiers `variable_b.xsl` et `disques1b.xml` montrent l'utilisation d'une variable dont le contenu est déterminé par un élément `xsl:choose`. Ceci permet d'afficher en bleu les disques sortis avant 1960, et en noir les autres. Le `xsl:choose` teste la date de sortie du disque et remplit le `xsl:variable` avec le contenu adapté.

xsl:param / xsl:with-param

`xsl:param` fonctionne comme `xsl:variable` et s'utilise de la même manière et pour les mêmes raisons. Il est juste possible de faire des choses en plus. Ce sont ces points là que nous allons présenter ici.

`xsl:with-param` possède également la même syntaxe. `xsl:with-param` doit simplement être un sous-élément de `xsl:apply-templates`. Si ensuite un élément `xsl:param` est sous-élément de l'élément `xsl:template` correspondant au `xsl:apply-templates` précédent (attributs `select` et `match` égaux et attributs `mode` égaux), et si l'attribut `name` de `xsl:param` est le même que celui du `xsl:with-param`, alors le `xsl:param` aura la même valeur que le `xsl:with-param`. En résumé, cela revient à passer des paramètres lors de l'appel du modèle.

Syntaxe

```
<xsl:apply-templates select='mon_elt'>
      <xsl:with-param name='mon_param'>coucou</xsl:with-param>
      <xsl:with-param name='mon_param2' select='"coucou"'/>
</xsl:apply-templates>
...
<xsl:template match='mon_elt'>
      <xsl:param name='mon_param'/>
      <xsl:value-of select='$mon_param'/> <---- affiche 'coucou'
      <xsl:value-of select='$mon_param2'/> <---- affiche 'coucou'
</xsl:template>
```

Exemple et démonstration

Les fichiers param_with-param.xsl et disques2.xml montrent l'utilisation de paramètres et des éléments `xsl:param` et `xsl:with-param`.

L'exemple est divisé en deux parties.

Dans la première partie, nous montrons comment passer un paramètre de type chaîne de caractères. Cela peut permettre de passer une valeur qui sera directement utilisée ou qui sera testée via un `xsl:if` ou un `xsl:choose`. En particulier, deux méthodes sont présentées, celle qui utilise l'attribut `select` directement dans l'élément `xsl:with-param` (`A` et `C`), et celle qui utilise un sous-élément de `xsl:with-param` de type texte (`B` et `S`). Notez que pour récupérer le paramètre dans le modèle appelé, il faut juste que l'attribut `name` de `xsl:param` ait la même valeur que l'attribut `name` du `xsl:with-param` correspondant. On utilise le symbole `$` pour l'utiliser comme pour un `xsl:variable`. Notez aussi que si vous voulez utiliser la chaîne de caractères elle-même, il faudra aussi utiliser `{$mon_param}` comme pour `xsl:variable` (voir la page sur `xsl:variable` pour un exemple).

Dans la deuxième partie, nous montrons comment passer un paramètre de type ensemble de noeuds. En donnant un motif XPath dans l'attribut `select` de l'élément `xsl:with-param`, nous passons ici un ensemble de noeuds qui sera ensuite utilisé dans le modèle appelé.

Les deux méthodes sont particulièrement intéressantes pour factoriser votre feuille XSLT et éviter d'écrire plusieurs fois les mêmes modèles à quelques valeurs près. N'hésitez donc pas à en user et à en abuser. Vous devrez d'ailleurs le faire dans les parties 15 et 16 des TP.

modèles «fonction» 1/2

XSLT n'est pas un langage fonctionnel. On ne peut a priori pas écrire de fonctions qu'on pourrait appeler et ré-appeler. Pour combler ce défaut, Les concepteurs d'XSLT ont permis une utilisation des modèles comme des fonctions si on utilise pas l'attribut `match`. Ce modèle «fonction» généralement affiche quelque chose (en html par exemple dans le cas ou la sortie est de l'html) ou rend une sortie à l'aide d'un élément `xsl:value-of`. On peut ensuite appeler ces modèles comme des fonctions en utilisant l'élément `xsl:call-template`.

Syntaxe

```
<xsl:template name='ma_fonction'>
      ...
      <xsl:value-of select='...'/>
</xsl:template>
...
<xsl:template match='...'>
      ...
      <xsl:call-template name='ma_fonction'/>
      ...
</xsl:template>
```

Exemple et démonstration

Les fichiers `call-template.xsl` et `disques3.xml` montrent l'utilisation de modèles «fonctions» et des éléments `xsl:template` sans l'attribut `match` et `xsl:call-template`.

Vous y trouverez notamment deux modèles `getNbDisques` et `getNbDisquesDeMachin`, qui respectivement, calculent le nombre d'éléments `disque` dans le document et le nombre d'éléments `disque` d'un `artiste` dans le document. Certes il n'y avait pas besoin de faire des modèles pour des choses aussi simples, mais c'est pour que l'exemple reste lisible. Quoique même dans ce cas, l'utilisation des modèles améliore la lisibilité de la transformation.

Les deux modèles utilisent un élément `xsl:value-of` pour «rendre» une valeur. Dans ce cas, la valeur est directement affichée dans le résultat. Le deuxième modèle (`getNbDisquesDeMachin`) utilise un paramètre (cf. page précédente), comme les autres modèles pour choisir un `artiste`.

Dans les parties 15 et 16, vous devrez créer et utiliser de tels modèles dans le fichier `common.xsl` pour calculer le nombre de matchs, de victoires et de défaites d'un joueur (en passant les infos sur le joueur en paramètres...) entre autres.

modèles «fonction» 2/2

Nous venons de voir dans la page précédente, l'intérêt des modèles et de l'élément `xsl:call-template`. Voici maintenant un truc encore plus fort : on peut récupérer la valeur de retour (via le xsl:value-of) d'un xsl:call-template dans une xsl:variable. Ceci permet d'en faire un autre traitement par la suite.

Syntaxe

```
<xsl:template name='ma_fonction'>
      ...
      <xsl:value-of select='...'/>
</xsl:template>
...
<xsl:template match='...'>
      ...
      <xsl:variable name='blabla'>
            <xsl:call-template name='ma_fonction'/>
      </xsl:variable> <-- blabla contient ce qu'a retourné le xsl:value-of au-dessus
      ...
</xsl:template>
```

Exemple et démonstration

Les fichiers `call-template_variable.xsl` et `disques4.xml` montrent l'utilisation de modèles «fonctions» et des éléments `xsl:template` sans l'attribut `match` et `xsl:call-template`, plus l'utilisation de `xsl:variable`.

Dans cet exemple, on affiche le nom, la date de naissance et la date de décès de chaque artiste du document. Pour cela, nous utilisons un modèle `getDateDeMachin`, qui rend la date de naissance ou de décès (en fonction d'un paramètre) d'un artiste (dont on donne le nom en paramètre). Ce modèle est un `template` sans attribut `match`, comme nous l'avons vu dans la page précédente. Il rend une date via un élément `xsl:value-of` qui a priori devrait l'afficher.

Le modèle `formatDate` (lui aussi un `xsl:template` sans attribut `match`) transforme bourrinement une date sous le format `2009-12-31` en `31 décembre 2009`. Il prend en paramètre la date à transformer.

Enfin, le modèle sur les éléments `artiste` utilise ces modèles. Il récupère les dates de naissance et de mort de chaque artiste dans des variables à l'aide du modèle `getDateDeMachin`, puis utilise ces variables dans le modèle `formatDate`, pour les afficher. Pour récupérer les résultats de `getDateDeMachin` dans une variable, il utilise la syntaxe présentée au-dessus.Vous noterez que lorsqu'un `call-template` est utilisé de cette manière, le `xsl:value-of` qu'il contient n'affiche rien dans le résultat html, mais que le résultat est enregistré dans la variable. Magique ! Super pratique ! et demandé dans les parties 15 et 16 de TP.

Remarque

```
<xsl:with-param name="naissance_ou_mort" select="string('naissance')"/>
```

et `<xsl:with-param name="naissance_ou_mort" select="'naissance'"/>`

signifient exactement la même chose.

inclusion de feuilles XSLT

Dernier point intéressant pour les parties 19 et 20 des TP : l'inclusion de fichiers.

L'idée de l'inclusion est toujours de permettre de factoriser mais cette fois-ci sur plusieurs fichiers. On peut ainsi placer des modèles dans une première feuille xslt, et utiliser ces modèles dans plusieurs autres feuilles xslt.

Syntaxe

```
<xsl:include href='feuille.xsl'/>
```
[1]

Exemple et démonstration

Le fichier `common.xsl` contient le modèle permettant de formater une date `formatDate` présenté dans la partie précédente. Les fichiers `call-template_variable2.xsl` et `disques5.xml` sont les mêmes que ceux de la partie précédente sauf que le modèle `formatDate` a été retiré de la feuille `call-template_variable2.xsl` qui inclut à la place la feuille `common.xsl`. Le fichier `disques6.xml` utilise la feuille de style `call-template_variable3.xsl`, qui inclut elle-aussi le fichier `common.xsl` et le modèle `formatDate`, mais cette fois-ci pour formater la date de sortie des albums.

Dans les parties 15 et 16, vous créez des modèles que vous placez dans un fichier `common.xsl` afin de les réutiliser à la fois dans `BestPlayersStyleSheet.xsl` et dans `PlayerProfile.xsl`. C'est un vrai gain de temps, une meilleure maintenance, une meilleure lisibilité. L'inclusion, du travail de pro.

[1] il existe aussi `xsl:import` avec un comportement légèrement différent : si un modèle avec le même nom existe déjà dans la feuille qui importe la deuxième feuille, c'est celui de la feuille qui importe qui gagne. Avec `include`, les deux sont à égalité.

utilisation de pseudo-tableaux

Pour bien comprendre l'intérêt de cette section, considérez le document XML `disques7.xml`. Vous pouvez voir que le concepteur de ce document a voulu ajouter des notes d'appréciation aux disques. Il souhaite maintenant créer une page html permettant de les visualiser, en présentant d'abord le disques ayant la note «`excellent`», puis ceux ayant la note «`très bien`» et enfin ceux ayant la note «`bien`». Malheureusement, il se rend compte maintenant qu'il aura du mal à faire un `for-each` + `sort` permettant de le faire, puisqu'il n'y a aucun moyen de trier dans cet ordre des disques ayant ces notes. Il aurait fallu choisir un autre système de notation. Comment faire ? En ajoutant un pseudo-tableau à la feuille XSL.

Syntaxe

Dans la feuille XSL :

```
<t:tabNotes>
      <t:note t:num="1">excellent</t:note>
      <t:note t:num="2">très bien</t:note>
      <t:note t:num="3">bien</t:note>
</t:tabNotes>
```

où `t` est un préfixe associé à un espace de noms choisi.

Le tri peut alors se faire sous la forme suivante :

```
<xsl:for-each select="d:disques/d:disque">
<xsl:sort
  select="document('')//t:tabNotes/t:note[. = current()/@d:note]/@t:num"
  data-type="number"
  order="ascending"/>
      ...
</xsl:for-each>
```

`document('')` représente le document courant (c'est-à-dire la feuille XSL). On recherche donc la `note`, ayant la même valeur que le disque courant (récupéré avec `current()`), puis on trie avec l'attribut `num` attribué à cette `note`. Magique !

Exemple et démonstration

Le fichier `pseudo-tableau.xsl` contient le modèle permettant d'utiliser ce tableau pour trier les disques de `disques7.xml`.