XSLT

Ingénierie documentaire et management des contenus

Bruno Bachimont

Université de Technologie de Compiègne

Filière « Ingénierie des connaissances et des supports d'information »

Présentation

XSLT est le langage proposé transformer et publier des contenus XML :

Transformation : un document XML est transformé en un autre document XML selon les instructions d'une feuille de style, un document XSLT, appliqué par un processeur XSLT.

Production : un document XML est transformé en un nouveau document, rédigé dans un autre langage XML (XHTML, FO, SMIL), selon les instructions d'une feuille de style, un document XSLT, appliqué par un processeur XSLT.

Un document XML

```
<?xml version="1.0" encoding="UTF-8"?>

<Cours code="NF29">
    <Sujet> ingénierie documentaire </Sujet>
    <Enseignants>
        <!--Les enseignants (ir)responsables -->
        <Enseignant>
            <Nom>Bachimont</Nom>
            <Prénom>Bruno</Prénom>
        </Enseignant>
        <Enseignant>
            <Nom>Gebers</Nom>
            <Prénom>Erik</Prénom>
        </Enseignant>
    </Enseignants>
    <Programme>
        <Séance id="1">Introduction à XML</Séance>
        <Séance id="2">Introduction au Schémas</Séance>
        <Année>2004</Année>
        <Semestre>Automne</Semestre>
    </Programme>
</Cours>
```

Une feuille de style

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="/">
        <html><head><title>Fiche de cours </title></head>
            <body bgcolor="white"><hr></hr>
                <h1>Fiche du cours « <xsl:value-of select="Cours/Sujet"/> » </h1>
                <hr></hr><xsl:apply-templates/> </body> </html>
    </xsl:template>
    <xsl:template match="Sujet"/>
    <xsl:template match="Enseignants">
        <h2>Les Enseignants</h2>
        <ol><xsl:apply-templates select="Enseignant"/></ol>
    </xsl:template>
    <xsl:template match="Enseignant">
        <li><xsl:value-of select="Prénom"/><xsl:value-of select="Nom"/></li>
    </xsl:template>
    <xsl:template match="Programme">
        <h2>Programme du cours</h2>
        <h3>Année <xsl:value-of select="Année"/> </h3>
        <h4>Semestre <xsl:value-of select="Semestre"/> </h4>
        <ol><xsl:for-each select="Séance">
                <li> <xsl:value-of select="."/></li></xsl:for-each></ol>
    </xsl:template></xsl:stylesheet>
```

Le document produit en HTML

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Fiche de cours</title>
</head>
<body bgcolor="white">
<hr>
<h1>Fiche du cours &laquo;  ing&eacute;nierie documentaire  &raquo; </h1>
<hr>
<h2>Les Enseignants</h2>
<ol>
<li>BrunoBachimont</li>
<li>ErikGebers</li>
</ol>
<h2>Programme du cours</h2>
<h3>Ann&eacute;e 2004</h3>
<h4>Semestre Automne</h4>
<ol>
    <li>Introduction &agrave; XML</li>
    <li>Introduction au Sch&eacute;mas</li>
</ol>
</body>
</html>
```

Ce qui donne :

Fiche du cours « ingénierie documentaire »

Les Enseignants

1. BrunoBachimont
2. ErikGebers

Programme du cours

Année 2004

Semestre Automne

1. Introduction à XML
2. Introduction au Schémas

Les feuilles de style XSLT

Les feuilles de style sont des documents XML, qui doivent comme telles être **bien conformées**. Elles suivent par ailleurs les règles de validité de XSLT pour être déclarées **valides**. Une feuille de style commence par l'instruction :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

- l'attribut `version` est obligatoire. A ce jour, le W3C a publié deux version de XSLT : 1.0, et 2.0 qui date de novembre 2003. Elle est en cours de déploiement, la plupart des outils suivent la 1.0.
- la déclaration d'espace de nom permet de considérer tous les littéraux comme du contenu à insérer dans le document produit et de qualifier les éléments XSLT par `xsl :`.

Structure de XSLT

On distingue :

Eléments de premier niveau : sont les éléments fils de l'élément racine `xsl:stylesheet`

Instructions : ce sont les éléments que l'on trouve dans le corpus des règles.

Parmi les éléments, on peut également distinguer :

- les éléments qui affectent la production du document résultat ;
- les éléments qui s'appliquent à la transformation du document source.

Eléments de premier niveau

Type d'élément	Description
`xsl:attribute-set`	définit un groupe d'attributs
`xsl:decimal-format`	définit un format d'affichage pour les numériques
`xsl:import`	import d'un programme XSLT
`xsl:include`	inclusion d'un programme XSLT
`xsl:key`	définit une clé (d'indexation) pour un groupe de nœuds
`xsl:namespace-alias`	définit des alias pour certains espaces de noms
`xsl:output`	indique le format de sortie (HTML, XML, texte)
`xsl: param`	définit un paramètre
`xsl:preserve-space`	conserve les nœuds blancs
`xsl:strip-space`	supprime les nœuds blancs
`xsl:template`	définit une règle XSLT
`xsl:variable`	définit une variable XSLT

Les instructions XSLT

Type d'élément	Description
`xsl:apply-imports`	permet d'appliquer une règle importée, tout en la complétant par un nouveau corps ;
`xsl:apply-templates`	déclenche l'application des règles
`xsl:attribute`	insère un attribut dans un élément du document produit ;
`xsl:call-template`	appelle une règle par son nom ;
`xsl:choose`	structure de test (cd. `switch` ou `case`)
`xsl:comment`	insère un nœud commentaire ;
`xsl:copy`	copie un nœud du document source dans le document produit ;
`xsl:copy-of`	copie un nœud, ainsi que tous ses descendants :
`xsl:result-document`	permet de créer plusieurs documents résultats (XSLT 2.0)
`xsl:element`	insère un nœud `Element` dans le document produit ;

Les instructions XSLT - suite

Type d'élément	Description
`xsl:fallback`	règle déclenchée si le processeur ne reconnaît pas une instruction ;
`xsl:for-each`	pour effectuer des itérations ;
`xsl:if`	branchement conditionnel ;
`xsl:message`	produit un message pendant le traitement XSLT ;
`xsl:number`	permet de numéroter les nœuds du document produit ;
`xsl:variable`	définit un paramètre ;
`xsl:processing-instruction`	insère un nœud `PocessingInstruction` dans le document produit :
`xsl:text`	inère un nœud `Text` ;
`xsl:value-of`	évalue une expression Xpath et insère le résultat ;

Règles XSLT

- Les règles XSLT sont définies par l'élément `xsl:template`
- les règles XSLT sont déclenchées par les instructions `xsl:apply-templates` ou `xsl:call-template` :

  `xsl :apply-templates :` la règle est déclenchée ou instanciée sur une catégorie de nœuds spécifiée par une expression XPath, les *patterns* ;

  `xsl :call-template :` la règle est appelée par son nom.

structure d'une règle XSLT :

```
<xsl:template match="pattern">
   corps de la règle
   </xsl:template>
```

Appel d'une règle XSLT :

```
<xsl:apply-template select="Pattern" />
<xsl:call-template name="Qname"/>
```

Remarque

Les *patterns*, qui sont des expressions XPath, apparaissent à la fois comme valeur des attributs `select` des éléments `xsl :apply-templates` que des attributs `match` des éléments `xsl :template` :

Dans `xsl :apply-templates` , le pattern sert à désigner un ensemble de nœuds : usage habituel d'une expression XPath ;

Dans `xsl :template` , le pattern exprime une condition sur les nœuds qui vont permettre de déclencher la règle.

Moralité :

- le contrôle du déclenchement des règles ne s'effectue que par l'appel `xsl :apply-templates` ; la déclaration d'une règle n'implique pas qu'elle sera déclenchée ;
- le déclenchement est initialisé avec comme nœud courant le nœud racine : soit une règle s'applique spécifiquement à lui, soit on y applique les règles par défaut.

Attributs des règles

La règle peut prendre les attributs suivants :

`match` qui désigne un *pattern*, c'est-à-dire une expression Xpath sélectionnant les nœuds de l'arbre XML sur lequels la règle doit s'instancier. Cette évaluation s'effectue à partir d'un nœud courant à partir duquel le *pattern* est évalué pour déterminer si la règle est applicable ou non.

`name` qui définit une règle *nommée*, qui sera appelée par l'instruction `xsl :call-template`. Exclusif avec `match`.

`priority` donne une priorité explicite à la règle. Mais une priorité par défaut est toujours calculée.

`mode` définit des catégories de règles, qui sont appelées dans des circonstances particulières.

Patterns

Les *patterns* sont :

- des expressions Xpath ;
- elles déterminent un ensemble de nœuds de l'arbre du document source de la façon suivante : un nœud N satisfait un pattern s'il existe quelque part dans l'arbre du document source un nœud C tel qu'en évaluant le patterne avec C comme nœud contexte, on obtienne un ensemble qui contienne N.
- les seuls axes (au sens de Xpath) autorisés sont `child` et `attribute` ;
- on peut utiliser `//` mais pas explicitement `descendant-or-self : :node()`.

No Good	Good
`B/preceding : :node()`	Nom[position()=2] *[position()=2][name()="NOM"]

Evaluation d'un *pattern*

L'évaluation d'un pattern P à partir d'un nœud courant N s'effectue de la manière suivante :

1. Si P est absolu, prendre la racine comme nœud contexte, et évaluer P. Si N fait partie du résultat, alors N satisfait P ;
2. Sinon
   (a) prendre N comme nœud contexte et évaluer P ; si N fait partie du résultat, alors N satisfait P ;
   (b) sinon prendre le père de N, et recommencer l'évaluation ; si N fait partie du résultat, alors ok ;
   (c) sinon recommencer en parcourant les ancêtres de N jusqu'à la racine du document.

Justification des *patterns*

- on maîtrise ainsi la complexité de l'évaluation des patterns, que l'on ramène au parcours des branches du nœud courant jusqu'à la racine du document.
- par conséquent, les patterns sont donc des sous-ensembles des expressions XPath.

Règles par défaut

XSLT comporte trois règles par défaut qui sont appliquées quand aucune règle du programme n'est sélectionnée.

1. la première règle s'applique à la racine et à tous les éléments ; elle déclenche un appel de règle sur tous les fils du nœud courant :

```
<xsl :template match=*|/"> <xsl :apply-templates/> </xsl :template>
```

2. la deuxième règle s'applique aux nœuds texte et aux attributs et insère le résultat textuel de ces nœuds dans le document résultat :

```
<xsl :template match="text() |@*"><xsl :value-of select="."
/></xsl :template>
```

3. la troisième règle s'applique aux commentaires et aux instructions de traitement. Par défaut, ces éléments sont ignorés :

```
<xsl :template match="processing-instruction() | comment()" />
```

Le programme XSLT minimal

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
</xsl:stylesheet>
```

Appliqué sur le document XML NF29.xml, on obtient :

```
<?xml version="1.0" encoding="UTF-8"?>
     ingénierie documentaire

            Bachimont
            Bruno
            Gebers
            Erik

        Introduction à XML
        Introduction au Schémas
        2004
        Automne
```

Note : les attributs sont ignorés.

Déclenchement de règles

Le déclenchement s'effectue avec l'instruction
`xsl:apply-templates` :

- elle désigne un ensemble de nœuds avec une expression XPath et demande l'application d'une règle pour chacun de ces nœuds.
- L'expression XPath est toujours évaluée en prenant comme nœud contexte le nœud courant pour lequel la règle contenant `xsl:apply templates` a été instanciée :
- cette instruction a deux attributs :

  `select` qui contient l'expression XPath désignant les nœuds à traiter ;

  `mode` qui désigne la catégorie des règles à considérer.

L'expression de l'attribut `select` doit toujours ramener un ensemble de nœuds. Sa valeur par défaut est : `child : :node()`, c'est-à-dire tous les fils du nœud courant, à l'exception des attributs.

Instanciation des règles

Soit la feuille de style

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="/">
    <xsl:apply-templates select="//Enseignants"/></xsl:template>
    <xsl:template match="Enseignants">
    <xsl:comment>Application de la règle Enseignants</xsl:comment>
    <xsl:apply-templates/></xsl:template>
    <xsl:template match="Nom">
    <xsl:value-of select="position()"/> : Nœud Nom = <xsl:value-of select="."/></xsl:template>
        <xsl:template match="Prénom">
    <xsl:value-of select="position()"/> : Nœud Prenom = <xsl:value-of select="."/></xsl:template>
    <xsl:template match="text()">
        <xsl:value-of select="position()"/> : Nœud de texte = [<xsl:value-of select="."/>] </xsl:template>
    <xsl:template match="comment()">
    <xsl:value-of select="comment()"/>
       <xsl:value-of select="position()"/> : Nœud de commentaire = <xsl:value-of select="."/>
    </xsl:template>
</xsl:stylesheet>
```

Son résultat

```
<!--Application de la règle Enseignants-->1 : Nœud de texte = [
        ] 2 : Nœud de commentaire = Les enseignants (ir)responsables 3 : Nœud de texte = [
        ] 1 : Nœud de texte = [
            ] 2 : Nœud Nom = Bachimont3 : Nœud de texte = [
            ] 4 : Nœud Prenom = Bruno5 : Nœud de texte = [
        ] 5 : Nœud de texte = [
        ] 1 : Nœud de texte = [
            ] 2 : Nœud Nom = Gebers3 : Nœud de texte = [
            ] 4 : Nœud Prenom = Erik5 : Nœud de texte = [
        ] 7 : Nœud de texte = [
    ]
```

Une instruction importante : `xsl :value-of`

Syntaxe : `<xsl:value-of select="Expression" />`

Description : cet élément évalue l'expression donnée en attribut de `select` sur le contexte et nœud courants, convertit le résultat en chaîne de caractères et insère la chaîne résultante dans le document produit.

Exemple :

```
<xsl:value-of select="." />
<xsl:value-of select="concat(position(),'/',last(),':',) "/>
```

Priorité d'une règle

Le système calcule pour chaque règle une priorité en se fondant sur le *pattern* de l'attribut `match` :

1. Tous les *patterns* constitués d'une seule étape XPath, avec un nom d'élément ou d'attribut, et sans prédicat, ont une priorité de 0 ;
2. le *pattern* `processing-instruction('nom')` a une priorité de 0 ;
3. Tous les *patterns* constitués d'une seule étape XPath, avec un filtre sur tous les éléments ou attributs d'un espace de nom ont une priorité de 0,25 ;
4. les filtres sans espaces de nom et autres qu'un nom d'élément ou d'attribut ont une priorité égale à -0,5.
5. tous les *patterns* différents des précédents ont une priorité de 0,5 : cas des *patterns* avec prédicats, ou constitués de plusieurs

étapes.

6. si le *pattern* est une union, on scinde l'union en autant de règles que de membres.

```
<xsl :template match="Sujet | node()"> corps de règle
</xsl :template>
```

donne :

```
<xsl :template match="Sujet"> corps</xsl :template>
<xsl :template math="node()">corps</xsl :template>
```

La première règle a une priorité de 0, la seconde de 0,5.

Moralité : plus c'est spécifique, plus c'est prioritaire

Mode d'une règle

L'attribut `mode` permet de sélectionner une règle en particulier via un appel `apply-templates` spécifiant son attribut `mode` avec la même valeur.

Feuille de style avec des attributs « mode »

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="Sujet"/><xsl:template match="Programme"/>
    <xsl:template match="Enseignants">
     <html>
            <head>
                <title>Fiche de cours </title>
            </head>
            <body bgcolor="white">
                <center><table><a name="table"/>
                <xsl:apply-templates select="Enseignant" mode="Ancres"/>
                    </table></center>
                <xsl:apply-templates select="Enseignant" mode="standard"/>
            </body>
     </html>
     </xsl:template>
```

Les deux templates selon le mode

```
    <xsl:template match="Enseignant" mode="Ancres">
    <td><a href="#{Nom}"><xsl:value-of select="Nom"/></a></td>
    </xsl:template>

    <xsl:template match="Enseignant" mode="standard">
        <hr></hr>
    <a name="{Nom}"/><h3><xsl:value-of select="Nom"/></h3>
    <p><b>Titre :</b><xsl:value-of select="Titre"/></p>
    <p><b>Fonction :</b><xsl:value-of select="Fonction"/></p>
        <p><a href="#table"> Autres Enseignants</a></p>
    <img src="toto.jpg" border="1"/>
    </xsl:template>
</xsl:stylesheet>
```

Résultats

```
<html>
<head><META http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>Fiche de cours </title> </head>
<body bgcolor="white">
<center><table>
<a name="table"></a><td>
   <a href="#Bachimont">Bachimont</a></td><td>
   <a href="#Gebers">Gebers</a></td></table></center>
<hr> <a name="Bachimont"></a>
<h3>Bachimont</h3>
<p><b>Titre :</b>Enseignant-Chercheur Contractuel, HdR</p>
<p><b>Fonction :</b>Responsable Mineur TCN</p>
<p><a href="#table"> Autres Enseignants</a></p>
<img border="1" src="toto.jpg"><hr>

<a name="Gebers"></a>
<h3>Gebers</h3>
<p><b>Titre :</b>Doctorant</p>
<p><b>Fonction :</b>Ing&eacute;nierie p&eacute;dagogique&gt;</p>
<p><a href="#table"> Autres Enseignants</a></p>
<img border="1" src="toto.jpg"></body>
</html>
```

Préséance entre règles

Distinguer :

Priorité entre règles : les règles possèdent une *priorité* calculée sur la base de leur *pattern* ;

Préséance entre règles : les règles possèdent une *préséance* en fonction de l'ordre de l'importation des fichiers les contenant.

La préséance repose sur deux éléments XSLT particuliers permettant d'importer des règles définies dans d'autres fichiers :

`xsl :import :` le processeur affecte aux règles importées une préséance moindre qu'aux règles du programme importateur ; les éléments `import` doivent apparaître avant tous les autres.

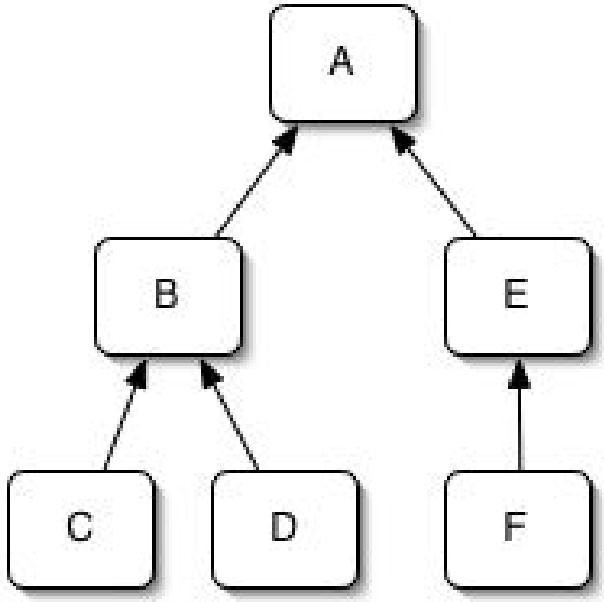
`xsl :include :` le processeur attribue la même préséances aux règles importées qu'aux autres.

Importations en cascade

- si un document D_1 est importé avant un document D_2, alors les règles de D_2 ont une préséance supérieure à celles de D_1 ;
- si une règle r_1 a une préséance supérieure à r_2, elle-même supérieure à r_3, alors r_1 a une préséance supérieure à r_3 (transitivité de la préséance).

1. A importe B, qui importe C et D ;
2. A importe E qui importe F.

L'ordre de préséance est : A, E, F, B, D, C.



Algorithme de sélection d'une règle

Le processeur XSLT choisit une règle à instancier pour un nœud sélectionné par `xsl :apply-templates` :

1. Au départ, la liste des règles candidates est constituée de toutes celles qui ont un attribut `match`.
2. On restreint la liste à toutes les règles qui ont le même mode que l'élément `xsl :apply-templates` appelant, ou pas d'attribut mode si `xsl :apply-templates` lui-même n'en a pas ;
3. on teste le *pattern* de l'attribut `match` pour déterminer si le nœud satisfait la règle ;
4. si on trouve plusieurs règles, on ne garde que celle(s) qui a(ont) la plus grande préséance d'importation ;
5. s'il reste encore plusieurs règles, choisir la plus grande priorité.

S'il reste encore des choix à faire, le processeur choisit en général la dernière dans l'ordre du programme. Si aucune règle n'est trouvée, les règles par défaut sont appliquées.

Appel de règles

Une règle peut déclarée avec un attribut `name` qui permet de l'appeler directement via l'instruction `xsl :call-template`.

Remarques :

- un appel avec `xsl :call-template` ne change pas le contexte : le nœud courant, sa position et la taille du contexte reste connus et identique dans le corps de la règle nommée.
- l'instanciation d'une règle nommée ne renvoie pas de valeur à la règle appelante ;
- l'instanciation d'une règle nommée ne prend pas d'argument autre que le nœud auquel elle s'applique.

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <xsl:apply-templates select="//Enseignants"/></xsl:template>
    <xsl:template match="Enseignants">
        <xsl:comment>Application de la règle Enseignants</xsl:comment>
        <xsl:apply-templates/></xsl:template>
    <xsl:template name="Afficher">
        <xsl:value-of select="position()"/> : <xsl:value-of select="."/>
        </xsl:template>
    <xsl:template match="Nom">
        <xsl:call-template name="Afficher"/> </xsl:template>
    <xsl:template match="Prénom">
        <xsl:call-template name="Afficher"/></xsl:template>
    <xsl:template match="text()">
        <xsl:call-template name="Afficher"/> </xsl:template>
    <xsl:template match="comment()">
        <xsl:call-template name="Afficher"/></xsl:template>
</xsl:stylesheet>
```

Résultats

```
<!--Application de la règle Enseignants-->1 :
        2 : Les enseignants (ir)responsables 3 :
        1 :
            2 : Bachimont3 :
            4 : Bruno5 :
        5 :
        1 :
            2 : Gebers3 :
            4 : Erik5 :
        7 :
```

Paramètres

L'utilisation de paramètre en XSLT s'effectue en 2 étapes :

1. dans la définition de la règle, on indique les paramètres associés et leur valeur par défaut :

   ```
   <xsl :param name="nom" select="expression" />
   ```

   l'attribut `name` donne le nom du paramètre, `select` donne une expression XPath dont l'évaluation donne la valeur par défaut.

   le paramètre est utilisé dans le corps de la règle sous la forme `$param`.

2. la règle est appelée en passant une valeur au paramètre :

   ```
   <xsl :call-template name="Afficher">
   <xsl :with-param name="texte" select="'texte vide'"/></xsl :call-template>
   ```

Paramètres : usage et exemple

L'utilisation de `xsl :with-param` obéit à des règles précises :

- il peut apparaître dans `<xsl :call-template>` dont c'est le seul contenu possible ;
- dans `<xsl :apply-templates>`, il faut le placer immédiatement après la balise ouvrante de l'élément.

Variables : `xsl :variable`

Syntaxe : `<xsl:variable>` `name=`*Qname* `select=`*Expression* `/>`

Corps de règle `</xsl :variable>`

Description : une variable peut référencer une information variée : un nœud, un ensemble de nœuds, une valeur numérique ou alphanumérique.

La valeur est donnée soit par l'évaluation de l'expression donnée en attribut `select`, soit par le contenu de la règle : l'un, ou l'autre !

Si la variable est un élément de premier niveau, elle est globale, sinon elle est locale au `xsl :template` qui la définit : visible dans les frères droits et les descendants. Une variable locale peut redéfinir une globale, mais pas de conflit autorisé entre variables locales.

Exemple

```
<xsl:variable name="var1" select="/Cours/Enseignants" />
<xsl:variable name="var3">contenu de la variable</xsl:variable>
```

Instructions de contrôle : `xsl :choose`

Syntaxe : `<xsl:choose>`

```
<xsl:when test:="@longeur &gt; 60">  ce texte est long </xsl:when>
<xsl:when test:="@longeur &lt; 20">  ce texte est court </xsl:when>
<xs:otherwise> ce texte est de longueur moyenne </xsl:otherwise>
```

Explication : le contenu de `choose` comporte une succession de tests (au moins un), compris dans des instructions `xsl :when`. Le processeur s'arrête au premier test fructueux, dont il exécute le corps. Sinon, il exécute le corps de `xsl :otherwise` (optionnel).

Exemple : `<xsl:choose>`

```
<xsl:when test:="@longeur &gt; 60">  ce texte est long </xsl:when>
<xsl:when test:="@longeur &lt; 20">  ce texte est court </xsl:when>
<xs:otherwise> ce texte est de longueur moyenne </xsl:otherwise>
```

Instructions de contrôle : `xsl :for-each`

Syntaxe :
```
<xsl:for-each select="Expression">
    corps
    </xsl:for-each>
```

Explication : on prend successivement les nœuds correspondant à l'évaluation du l'expression donnée en valeur de l'attribut `select`. Le nœud courant, après l'exécution du `for-each` redevient celui pour lequel la règle a été instanciée.

Exemple :
```
<xsl:for-each select="Séance">
                <li> <xsl:value-of select="."/></li>
</xsl:for-each>
```

Instructions de contrôle : `xsl :if`

Syntaxe :
```
<xsl:if test="Expression">
    corps
</xsl:if>
```

Explication : Le contenu est instancié si l'expression de l'attribut `test` s'évalue à `true`. Il n'y a pas d'équivalent au `else` des langages de programmation.

Exemple :
```
<xsl:if test="Année = 2004">
            <xsl:copy-of select="."/>
</xsl:if>
```

Instructions de contrôle : `xsl :sort`

Syntaxe : `<xsl:sort`

```
select="Expression"
lang="CodeLangue"
order=("ascending" | "descending")
case=("lower-first" | "upper-first"
data-type=("test" | "number")  />
```

Explication : élément permettant de trier des nœuds sélectionnés par `xsl :apply-templates` ou `xsl :for-each`. `xsl :sort` doit apparaître juste après leurs balises ouvrantes. La clef de tri est indiquée par l'attribut `select`, évalué pour chaque nœud du contexte. Par défaut, le tri se fait selon l'ordre lexicographique sur les chaînes de caractères. Si `data-type` vaut `number`, le tri est numérique. Le tri est croissant par défaut, sauf si `order` vaut `descending`. L'attribut `lang` permet d'indiquer l'ordre des caractères spéciaux (« é » inférieur ou supérieur à « è »).

Quand plusieurs `sort` sont successifs, les tris postérieurs portent sur les groupes non départagés des premiers tris.

Exemple :

```
<xsl:apply-templates select="//Chapitre">
<xsl:sort select="@longeur" data-type="number"/>
</xsl:apply-templates>
```

XSLT, HTML, XHTML

Soit le document suivant :

```
<html>
  <head> <title>Exemple de page HTML </title></head>
  <BODY>
      <P>Paragraphe 1
      <br><img SRC="image.gif" width=30pt ismap>
      <p>paragraphe 2  </body>
</html>
```

Ce document, conforme à HTML 4.0, n'est pas conforme selon XML :

- les balises vides de HTML4.0 ne doivent pas être fermées ; `<br>`, alors qu'en XML on devrait avoir `<br></br>` ;
- les noms de balises en HTML4.0 ne sont pas sensibles à la casse : `<P>` et `<p>` sont équivalentes ; XML est sensible à la casse.
- certaines balises non vides de HTML4.0 peuvent ne pas être fermées ;
- certains attributs peuvent avoir une valeur non encadrée d'apostrophes : `width=30pt` ;
- les attributs booléens de HTML4.0 peuvent être utilisés sans valeur : `ismap`.

Le problème entre HTML et XSLT

Problème : une feuille de style ne peut avoir que des contenus conformes à XML, car un programme XSLT est un document XML. Comment donc insérer du contenu HTML dans les sections littérales d'une feuille de style ?

Solution : elle tient en deux temps :

- d'une part, on produit dans le corps de la feuille de style des contenus XHTML, c'est-à-dire des contenus « à la HTML » conformes aux règles de bonne formation de XML ; la feuille de style est donc validable ;
- d'autre part, on spécifie *via* l'élément `xsl :output` le format de sortie du résultat de la transformation. Il peut prendre 3 valeurs : `html`, `xml` et `text`. `html` est la valeur par défaut. En fonction de la valeur donnée, le processeur sérialise en HTML le contenu XHTML produit par la transformation XSLT.

xsl :output="html"

La feuille de style :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html" indent="yes"  encoding="UTF-8" />
    <xsl:template match="/">
    <html><head><!-- un titre -->
        <title>Alien</title></head>
        <body>texte avec des
        caractères à accents et accentués
        </body></html></xsl:template>
</xsl:stylesheet>
```

Le résultat :

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Alien</title>
</head>
<body>texte avec des
        caract&egrave;res &agrave; accents et accentu&eacute;s
        </body>
</html>
```

xsl :output="xml"

La feuille de style :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="xml" indent="yes"  encoding="UTF-8" />
    <xsl:template match="/">
    <html><head><!-- un titre -->
        <title>Alien</title></head>
        <body>texte avec des
        caractères à accents et accentués
        </body></html></xsl:template>
</xsl:stylesheet>
```

Le résultat :

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<head>
<title>Alien</title>
</head>
<body>texte avec des
        caractères à accents et accentués
        </body>
</html>
```