XSL
Langage de transformation de XML

Approfondissements

http://www.zvon.org/xxl/XSLTreference/Output/index.html

Le langage XSL

Retour sur les modèles (template)

- Une feuille de style peut contenir plusieurs modèles
  - Chaque modèle contient des informations sur l'affichage d'une branche des éléments du document
  - S'il n'y a qu'un seul modèle alors :
    - Il s'applique sur la racine du document XML
    - Le modèle est appliqué à la branche spécifiée par l'attribut match de l'élément template

```
<xsl:template match="/">
        Action
</xsl:template>
```

racine

    - Ensuite, on utilise dans Action :
      - select pour sélectionner les nœuds

Exemple : value-of select

exemple1.xsl

XSL

```
<xsl:stylesheet xmlns:xsl =
"http://www.w3.org/1999/XSL/Transform" version
= "1.0" >
        <xsl:template match = "/" >
            <xsl:text >
            </xsl:text>
            <xsl:value-of select = "//BBB[1]" />
            <xsl:text > + </xsl:text>
            <xsl:value-of select = "//BBB[2]" />
            <xsl:text > + </xsl:text>
            <xsl:value-of select = "//BBB[3]" />
            <xsl:text > = </xsl:text>
            <xsl:value-of select = "sum(//BBB)" />
        </xsl:template>
    </xsl:stylesheet>
```

XML

```
<AAA >
    <BBB>10 </BBB>
    <BBB>5 </BBB>
    <BBB>7 </BBB>
</AAA>
```

Résultat

???

Autre exemple

Bibliotheque1.xml + bibliotheque1.xsl

```
<bibliotheque>
<livre>
        <titre>N ou M</titre>
        <auteur>Agatha Christie</auteur>
        <ref>Policier-C-15</ref>
</livre>
<livre>
        <titre>Le chien des Baskerville</titre>
        <auteur>Sir Arthur Conan Doyle</auteur>
        <ref>Policier-D-3</ref>
</livre>
<livre>
      <titre>Dune</titre>
      <auteur>Franck Heckbert</auteur>
<ref>Fiction-H-1</ref>
</livre>
</bibliotheque>
```

```
<xsl:template match="/">
        <H2>Bibliotheque</H2>
        <xsl:for-each select="bibliotheque/livre">
              <SPAN style="font-style:italic">
              <xsl:value-of select="titre"/>
              </SPAN>
              <SPAN style="color:red">
              <xsl:value-of select="auteur"/>
              </SPAN>
              <SPAN style="color:blue">
              <xsl:value-of select="ref"/>
              </SPAN>
            <br />
      </xsl:for-each>
</xsl:template>
```

Le langage XSL

bibliotheque5.xml + bibliotheque5.xsl

Utilisation de plusieurs modèles

- Un modèle général et un modèle pour chaque élément
  - Exemple : un modèle pour livre dans biblitheque5.xml
- Ce modèle remplace : xsl:for-each

```
<xsl:template match="/">
        <H2>Bibliotheque</H2>
        <xsl:apply-templates select="bibliotheque/livre" />
</xsl:template>

<xsl:template match="livre">
        <SPAN style="font-style:italic">
          <xsl:value-of select="titre"/>
        </SPAN>
        <SPAN style="color:red">
          <xsl:value-of select="auteur"/> …
```

On va appliquer ce modèle chaque fois que l'on va rencontrer l'élément "livre" après "bibliothèque"

Attention ! Un élément "livre" non fils de "bibliothèque" ne sera pas affiché

Le langage XSL

Plusieurs modèles

- Commentaires
  - Lorsqu'il y a plusieurs modèles, il faut toujours qu'il y en ait un pour l'affichage de la racine du document (le /)
    - ne serait ce que pour insérer les balises <HTML> </HTML>
  - Dans le premier modèle, le xsl:apply-templates indique que pour chaque élément livre, enfant de bibliotheque, il faut appliquer le deuxième modèle (celui pour lequel l'attribut match a pour valeur livre)
  - Exemple :

Bibliotheque

N ou M Agatha Christie Policier-C-15
Le chien des Baskerville Sir Arthur Conan Doyle Policier-D-3
Dune Franck Heckbert Fiction-H-1

Exemple1 : apply-templates

exemple2.xsl

XSL

```
<xsl:stylesheet xmlns:xsl =
"http://www.w3.org/1999/XSL/Transform"
version = "1.0" >
        <xsl:output method = "text" />
        <xsl:template match = "/" >
            <xsl:apply-templates select =
"//BBB" />
        </xsl:template>

      <xsl:template match = "BBB" >
            <xsl:text >
                BBB[</xsl:text>
            <xsl:value-of select = "position()"
/>
            <xsl:text >]: </xsl:text>
            <xsl:value-of select = "." />
        </xsl:template>
    </xsl:stylesheet>
```

XML

```
<AAA >
    <BBB>10 </BBB>
    <BBB>5 </BBB>
    <BBB>7 </BBB>
</AAA>
```

Résultat

???

Exemple2 : apply-templates

exemple3.xsl

XSL

```
<xsl:stylesheet ... >
        <xsl:output method = "text" />

        <xsl:template match = "/" >
            <xsl:text >Text: </xsl:text>
            <xsl:apply-templates select = "//BBB" >
                <xsl:sort data-type = "text" />
            </xsl:apply-templates>

            <xsl:text >
                    Number: </xsl:text>
            <xsl:apply-templates select = "//BBB" >
                <xsl:sort data-type = "number" />
            </xsl:apply-templates>
        </xsl:template>

        <xsl:template match = "BBB" >
            <xsl:value-of select = "." />
            <xsl:text > </xsl:text>
        </xsl:template>
    </xsl:stylesheet>
```

XML

```
<AAA >
        <BBB>cc </BBB>
        <BBB>fff </BBB>
        <BBB>10 </BBB>
        <BBB>6 </BBB>
        <BBB>333 </BBB>
        <BBB>100000 </BBB>
<AAA>
```

Résultat

?

Exemple3 : apply-templates

XSL

```
<xsl:stylesheet ... >

        <xsl:output method = "text" />
        <xsl:template match = "/" >
            <xsl:apply-templates select = "//*" >
                <xsl:sort select = "name()" />
            </xsl:apply-templates>
        </xsl:template>

        <xsl:template match = "*" >
            <xsl:value-of select = "name()" />
            <xsl:text > </xsl:text>
        </xsl:template>

</xsl:stylesheet>
```

XML

```
<AAA >
    <BBB>cc </BBB>
    <BBB>fff </BBB>
    <BBB>10 </BBB>
    <BBB>6 </BBB>
    <BBB>333 </BBB>
    <BBB>100000 </BBB>
</AAA>
```

Résultat

?

Exemple4 : apply-templates

Que donne apply-templates sans paramètres ?

XSL

```
<xsl:stylesheet … >
  <xsl:template match="/">
        <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="AAA">
   <xsl:comment> …</xsl:comment>
   <xsl:value-of select="name()" />
 </xsl:template>

  <xsl:template match="BBB">
      <xsl:value-of select="name()" />
  </xsl:template>

</xsl:stylesheet>
```

XML

```
<AAA >
    <BBB>cc </BBB>
    <BBB>fff </BBB>
    <BBB>10 </BBB>
</AAA>
```

Résultat

?

Le langage XSL

Modèle nommé (explicite)

- Un modèle peut avoir un nom
  - On peut l'appeler explicitement, comme une fonction spécifique en quelque-sorte...

```
...
 <xsl:template match="dateRevision">
   <xsl:call-template name="formatDate"/>
 </xsl:template>

 <xsl:template name="formatDate">
  <!-- Code XSL ici pour formater la date -->
 </xsl:template>
...
```

  - Attention formatDate n'est pas un élément, mais un simple identifiant
  - On utilise "name" et nom "match" pour l'appel et la définition du template

Exemple1 : call-templates

XSL

```
<xsl:stylesheet ... >
        <xsl:output method = "xml"
         indent = "yes" />
        <xsl:template match = "/" >
            <xsl:call-template name = "print" />
        </xsl:template>

        <xsl:template name = "print" >
            <XXX >xxx</XXX>
        </xsl:template>
    </xsl:stylesheet>
```

XML

```
<AAA >
    <BBB>bbb </BBB>
    <CCC>ccc </CCC>
</AAA>
```

Résultat

???

Le langage XSL

- Passage de paramètres à un modèle explicite
  - Déclaration : paramètres formels à la définition (comme une fonction)

```
<xsl:template match="elementType">
  <xsl:param name="txt"/>
  <xsl:param name="lang" select="FR"/>
     …
</xsl:template>
```

  - Appel du template : paramètres effectifs

```
<xsl: template>
     …
     <xsl:apply-templates select="elementType">
     <xsl:with-param name="txt" select="text()"/>
     <xsl:with-param name="lang" select="@xml:lang"/>
     …
</xsl:template>
```

Le langage XSL

Commentaires

- Le template sera appelé avec :
  - **"txt" qui sera instancié avec le contenu textuel de *elementType : (text())***
  - **"lang" qui sera instancié avec la valeur de l'attribut "xml:lang" de *elementType***
- Exemple
  - with-param-doc.xml
  - with-param-doc.xsl

Le langage XSL

- Exemple 1 :
  - On peut imaginer un template permettant d'évaluer une expression polynomiale :

```
<xsl:template name="polynome">
<!– seul param formel -->
   <xsl:param name="variable_x"/>
   <xsl:value-of select="2*$variable_x*$variable_x+(-5)
   *$variable_x+2"/>
<!– corps du template -->
<xsl:template>
```

Le langage XSL

- Il suffit alors de l'appeler en lui passant diverses valeurs pour le paramètre variable_x pour qu'il évalue cette expression

```
<xsl:call-template name="polynome">
<xsl:with-param name="variable_x" select="3.4"/>
<!– appel avec la valeur 3.4 -->
</xsl:call-template>
```

- Permet d'afficher le résultat de 2*3.4^2-5*3.4+2

Exemple 2 : exemple6.xsl

XSL

```
<xsl:stylesheet ... >
        <xsl:template match = "/" >
            <xsl:call-template name = "print" >
                <xsl:with-param name = "A" > 11
                </xsl:with-param>
                <xsl:with-param name = "B" > 33
                </xsl:with-param>
            </xsl:call-template>
            <xsl:call-template name = "print" >
                <xsl:with-param name = "A" > 55
                </xsl:with-param>
            </xsl:call-template>
        </xsl:template>
        <xsl:template name = "print" >
            <xsl:param name = "A" />
            <xsl:param name = "B" >111</xsl:param>
            <xsl:text >
            </xsl:text>
            <xsl:value-of select = "$A" />
            <xsl:text > + </xsl:text>
            <xsl:value-of select = "$B" />
            <xsl:text > = </xsl:text>
            <xsl:value-of select = "$A+$B" />
        </xsl:template>
    </xsl:stylesheet>
```

XML

```
<AAA >
    <BBB>bbb </BBB>
    <CCC>ccc </CCC>
</AAA>
```

Résultat

?

Le langage XSL

Variables

- <xsl:variable>
  - L'élément <xsl:variable> permet d'ajouter une constante à une feuille de style
  - Exemple :

    < xsl:variable name="C-PI">3.14</xsl:variable>
  - Il devient alors possible d'accéder à cette constante partout en utilisant le nom de la variable précédé d'un signe $
  - Exemple :

    <math pi="{$C-PI}"/>

  ou

    <xsl:value-of select="$C-PI"/>
- Attention
  - On ne peut pas modifier la valeur d'une variable…
- Attention à la portée d'une variable :
  - Dans l'élément qui contient sa déclaration

Le langage XSL

Variable

- <xsl:variable> : plusieurs manières de lui affecter une valeur

1. Attribut select avec la valeur (une constante)

```
<xsl:variable name='var1' select='12'/>
```

Valeur = 12

2. Attribut select avec une expression XPath

```
<xsl:variable name='var2' select='/COURS/ENSEIGNANTS'/>
```

Valeur = contenu du fils de ENSEIGNANTS dans l'arbre

3. Valeur de l'élément <xsl:variable>

```
<xsl:variable name="nom" >
    bonbon
</xsl:variable>
```

Valeur = bonbon

Le langage XSL

Variable : plusieurs manières pour affecter une valeur

- Ou encore

4. Valeur de l'élément <xsl:variable>

```
<xsl:variable name="nom" >
    <xsl:value-of select="/personne/nom/prenom"/>
    <xsl:value-of select="$espace"/>
   <xsl:value-of select="/personne/nom/famille"/>
</xsl:variable>
```

Valeur = texte de prenom + " " + texte de famille

5. ou encore :

```
<xsl:variable name='var3'>
    Ceci est un
    <mot-cle>contenu</mot-cle> de variable
</xsl:variable>
```

Valeur : ceci est un contenu de variable

Le langage XSL

Variables

- Autre exemple : exemple7.xsl

```
<xsl:template match="/">

<xsl:variable name="espace">
<xsl:text> </xsl:text>
</xsl:variable>

<xsl:variable name="nom" >
    <xsl:value-of
     select="/personne/nom/prenom"/>
    <xsl:value-of select="$espace"/>
   <xsl:value-of
     select="/personne/nom/famille"/>
</xsl:variable>
<xsl:copy-of select="$nom"/>

</xsl:template>
</xsl:stylesheet>
```

- exemple7.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<personne>
    <nom>
      <prenom>John</prenom>
      <famille>Doe</famille>
    </nom>
</personne>
```

- Résultat

  ?

Le langage XSL

Variables

- Autre exemple : exemple8.xsl

```
<xsl:template match="/">
<xsl:variable name="nom"
    select="/personne/nom/prenom"/>
<xsl:copy-of select="$nom"/>
</xsl:template>
</xsl:stylesheet>
```

- exemple8.xsl

```
<?xml version="1.0"
    encoding="UTF-8"?>
<personne>
    <nom>
      <prenom>John</prenom>
      <famille>Doe</famille>
    </nom>
</personne>
```

- Résultat

  ?

Le langage XSL

Variables

- Deux types de variables :
  - **Variables gobales** (éléments de premier niveau)
    - Elles sont visibles dans tout le programme XSLT
  - **Variables locales** (dans les corps des règles)
    - Visibles dans les following-sibling et leurs descendants

Le langage XSL

Variables

- Exemple de variable globale :

```
<xsl:variable name="annee" select="1970"/>
   <xsl:template match="FILM">
    <xsl:choose>
            <xsl:when test="ANNEE &lt; $annee">
                    "<xsl:value-of select="TITRE"/>" est ancien
            </xsl:when>
            <xsl:when test="ANNEE &gt;= $annee">
                    "<xsl:value-of select="TITRE"/>" est récent
            </xsl:when>
   </xsl:choose>
</xsl:template>
```

Le langage XSL

Variables

- Exemple de variable locale :

```
<xsl:template match="PROGRAMME">
   <SEANCES>
     <xsl:variable name="phrase">
                (valable pour l'an
                <xsl:value-of select="ANNEE"/>)
     </xsl:variable>
     <xsl:for-each select="SEANCE">
     <xsl:value-of select="concat(., $phrase)"/>
     </xsl:for-each>
   </SEANCES>
</xsl:template>
```

Le langage XSL

Variables

- Exemple de variable locale : la variable prend la valeur résultat de l'appel

```
<xsl:template match="/">
        <xsl:variable name="resultat">
            <xsl:call-template name="calcul"/>
        </xsl:variable>
    <xsl:value-of select="$resultat"/>
    </xsl:template>

    <xsl:template name="calcul">
        <xsl:value-of select="1"/>
    </xsl:template>
```

- Résultat :

  ?

Exemple1 : variable

XSL

```
<xsl:stylesheet xmlns:xsl =
"http://www.w3.org/1999/XSL/Transform"
version = "1.0" >
        <xsl:output method = "text" />
        <xsl:variable name = "a" > 12
        </xsl:variable>
        <xsl:template match = "/" >
            <xsl:variable name = "b" > 23
            </xsl:variable>
            <xsl:value-of select =
            "concat($a,' + ',$b,' = ', $a+$b)" />
        </xsl:template>
    </xsl:stylesheet>
```

XML

```
<AAA >
    <BBB>bbb </BBB>
    <CCC>ccc </CCC>
</AAA>
```

Résultat

???

Exemple2 : variable

exemple10.xml + exemple10.xsl

```
<xsl:stylesheet xmlns:xsl =
"http://www.w3.org/1999/XSL/Transform" version =
"1.0" >
        <xsl:output method = "text" />
        <xsl:variable name = "a" select = "//CCC" />
        <xsl:variable name = "b" >
            <xsl:value-of select = "//CCC" />
        </xsl:variable>
        <xsl:template match = "/" >
            <xsl:apply-templates select = "$a" />
            <xsl:text>
            ##########################
            </xsl:text>
            <xsl:value-of select = "$b" />
        </xsl:template>
        <xsl:template match = "CCC" >
            <xsl:text >
        CCC: </xsl:text>
            <xsl:value-of select = "." />
        </xsl:template>
    </xsl:stylesheet>
```

XML

```
<AAA >
    <BBB>
        <CCC>c11 </CCC>
        <CCC>c12 </CCC>
        <CCC>c13 </CCC>
    </BBB>
    <BBB>
        <CCC>c21 </CCC>
        <CCC>c22 </CCC>
        <CCC>c23 </CCC>
    </BBB>
    <BBB>
        <DDD>
            <EEE>e1 </EEE>
            <EEE>e2 </EEE>
            <EEE>e3 </EEE>
        </DDD>
        <DDD>
            <EEE>e1 </EEE>
            <EEE>e2 </EEE>
            <EEE>e3 </EEE>
        </DDD>
    </BBB>
</AAA>
```

Résultat ???

Exemple3 : variable

Format d'affichage : exemple11.xsl

```
<xsl:stylesheet .... >

      <xsl:output method = "text" />
      <xsl:variable name = "A" >-12.48 </xsl:variable>

  <xsl:template match = "/" >
     <xsl:text >
     </xsl:text>
          <xsl:value-of select = "format-number
                                  ($A,'#.##')" />
     <xsl:text >
     </xsl:text>
          <xsl:value-of select = "format-number
                                  ($A,'%#.##')" />
     <xsl:text >
     </xsl:text>
          <xsl:value-of select = "format-number
                                 ($A,'000.000')" />
     <xsl:text >
     </xsl:text>
          <xsl:value-of select = "format-number
                                  ($A,'0.0')" />
  </xsl:template>
</xsl:stylesheet>
```

```
<AAA>
        <BBB>bbb</BBB>
        <CCC>ccc</CCC>
</AAA>
```

Résultat ???

TD

- Énoncé
  - Exercices 1-3

Le langage XSL

Les modes

- L'instruction apply-templates peut être agrémentée de plusieurs attributs. Nous avons vu dans la section précédente que plusieurs templates peuvent être applicables aux mêmes éléments
- Dans ce cas, pour différencier les traitements sur un même élément sélectionné par apply-templates, on utilise l'attribut : mode
  - Template avec un mode particulier :

```
<xsl:template match='book' mode='index'>
</xsl:template>
<xsl:template match='book' mode='full'>
</xsl:template>
```

  - Appel du template avec un mode :

```
<xsl:apply-templates select='book' mode='index' />
```

- L'attribut mode permet de sélectionner lequel des patrons éligibles doit être sélectionné. En l'absence de cet attribut, aucun des patrons n'étant plus spécifique que l'autre, il serait impossible de savoir lequel des deux le processeur XSLT choisirait

Le langage XSL

Les modes

Exemple 1

- Soit le document XML suivant :

```
<doc>
    <taggersent>
        <taggertoken wordform="Il"       lemma="il"          pos="PRO"/>
        <taggertoken wordform="était"    lemma="être"        pos="VER:sg"/>
        <taggertoken wordform="une"      lemma="un"          pos="DET:femi:sg"/>
        <taggertoken wordform="fois"     lemma="fois"        pos="NOM:femi:sg"/>
        <taggertoken wordform=","        lemma=","           pos="PUN"/>
        <taggertoken wordform="les"      lemma="le"          pos="DET:masc:pl"/>
        <taggertoken wordform="voleurs" lemma="voleur" pos="NOM:masc:pl"/>
        <taggertoken wordform="étaient" lemma="étayer|être" pos="VER:pl"/>
        <taggertoken wordform="dans"     lemma="dans"        pos="PRP"/>
        <taggertoken wordform="la"       lemma="le"          pos="DET:femi:sg"/>
        <taggertoken wordform="forêt"    lemma="forêt"       pos="NOM:femi:sg"/>
        <taggertoken wordform="."        lemma="."           pos="PUN"/>
    </taggersent>
</doc>
```

Le langage XSL

Les modes

```
<xsl:template match="/">
   Liste des tokens :
  <xsl:apply-templates mode="lemme"/>
  </xsl:template>
  <xsl:template match="taggertoken" mode="lemme">
   Lemme : <xsl:value-of select="@lemma"/>
  </xsl:template>
  <xsl:template match="taggertoken" mode="forme">
   Forme : <xsl:value-of select="@wordform"/>
  </xsl:template>
```

- Liste des tokens :
  Lemme : il
  Lemme : être
  Lemme : un
  Lemme : fois
  Lemme : le
  Lemme : voleur
  Lemme : étayer|être
  Lemme : dans
  Lemme : le
  Lemme : forêt
  Lemme : .

```
<xsl:template match="/">
   Liste des tokens :
    <xsl:apply-templates mode="forme"/>
</xsl:template>
<xsl:template match="taggertoken" mode="lemme">
   Lemme : <xsl:value-of select="@lemma"/>
</xsl:template>
<xsl:template match="taggertoken" mode="forme">
   Forme : <xsl:value-of select="@wordform"/>
</xsl:template>
```

- Liste des tokens :
  Forme : Il
  Forme : était
  Forme : une
  Forme : fois
  Forme : les
  Forme : voleurs
  Forme : étaient
  Forme : dans
  Forme : la
  Forme : forêt
  Forme : .

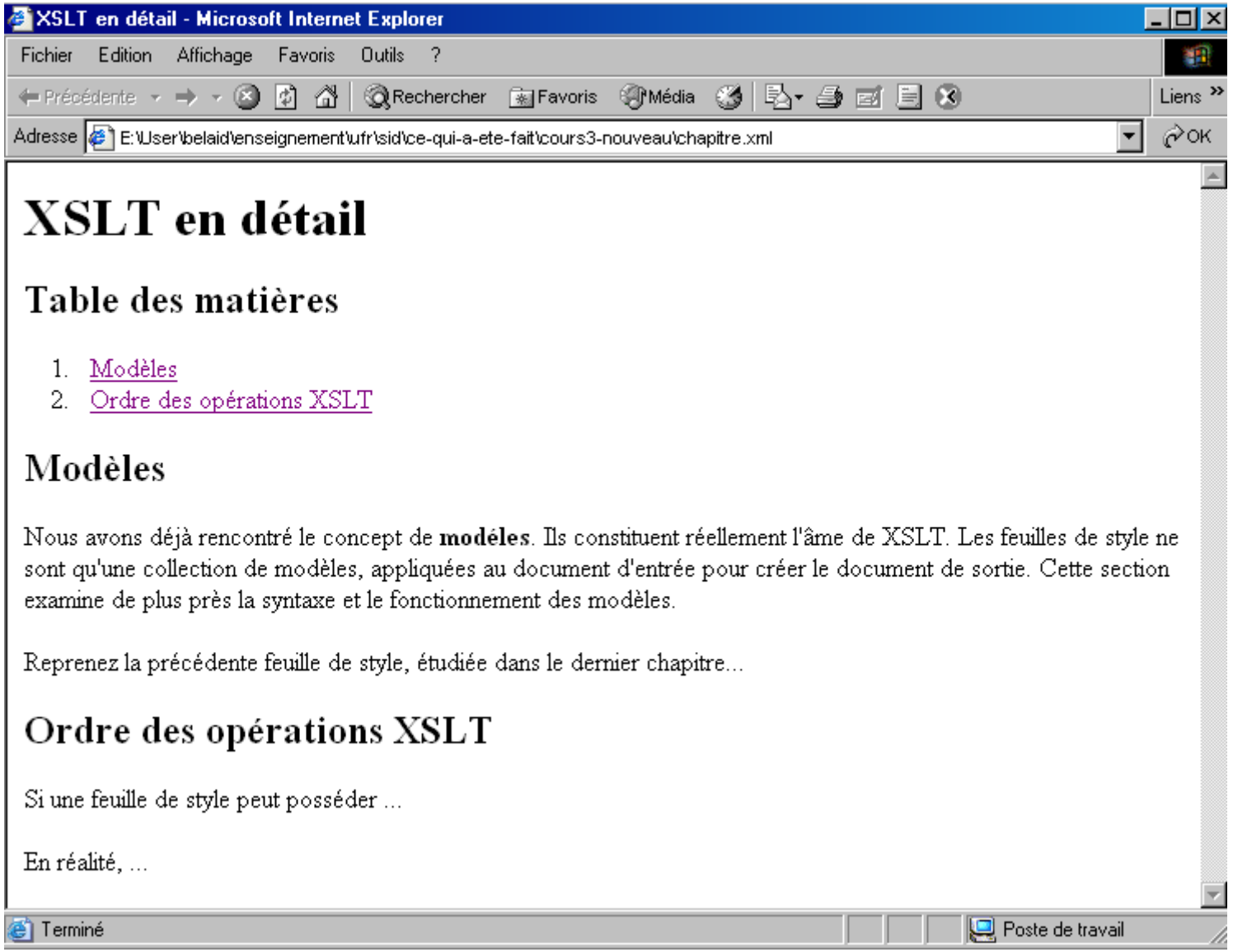
Le langage XSL

Les modes

- Exemple 2 :
  - Utiliser cet attribut sur les titres de sections d'un document pour différencier deux modes traitements :
    - En mode 1 : affichage d'une table de matières
      - le titre est :
        - une référence dans une table des matières
    - En mode 2 : affichage du corps du texte
      - le titre :
        - un titre de section dans le corps du document : <h1> en HTML dans le fil du texte

Le langage XSL

- Résultat



Le langage XSL

Illustration : chapitre.xml+chapitre.xsl

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<doc>
  <titre> XSLT en détail</titre>
  <section>Modèles
    <paragraphe>Nous avons déjà rencontré le concept de
    <important>modèles</important>. Ils constituent réellement l'âme de XSLT. Les
    feuilles de style ne sont qu'une collection de modèles, appliquées au document
    d'entrée pour créer le document de sortie. Cette section examine de plus près la
    syntaxe et le fonctionnement des modèles. </paragraphe>
    <paragraphe>Reprenez la précédente feuille de style, étudiée dans le dernier
    chapitre...</paragraphe>
  </section>
  <section>Ordre des opérations XSLT
    <paragraphe>Si une feuille de style peut posséder ...</paragraphe>
<paragraphe>En réalité, ...</paragraphe>
  </section>
</doc>
```

Le langage XSL

Les étapes de construction

- Étape 1 : Initialisation
  - Écriture de la racine de l'arbre HTML +
    - Définition de deux modes pour le template "section" (élément qui servira dans la table des matières et dans le texte)
    - La même portion de l'arbre (section) va être traitée deux fois par deux modèles distincts :

```
<xsl:template match="/">
 <HTML>
 <HEAD><TITLE><xsl:value-of select="/doc/titre"/></TITLE></HEAD>
 <BODY>
    <H1><xsl:value-of select="/doc/titre"/></H1>
    <H2>Table des matières</H2>
     <OL> <xsl:apply-templates select="/doc/section" mode="TDM"/> </OL>
            <xsl:apply-templates select="/doc/section" mode="corps"/>
 </BODY>
```

Le langage XSL

- Étape 2 : Création des deux modes
  - Le premier aboutit à la sortie de la table des matières

```
<xsl:template match="section" mode="TDM">
   <LI><A href="{concat('#section', position())}"><xsl:value-of
   select="text()"/></A>
   </LI>
</xsl:template>
```

➔Chaque fois que l'on rencontre "section", on crée un item avec : le numéro de la section, suivi de la référence dans le texte au contenu de "section " :

➔position() : donne le numéro de la section : puce numérique 1., 2., etc.

➔text() donne le contenu du titre de la section

Le langage XSL

- Le second produit le corps principal du document

```
<xsl:template match="section" mode="corps">
 <A name="{concat('section', position())}"> <H2> <xsl:value-of
  select="text()"/> </H2>
 </A>
 <xsl:apply-templates/>
</xsl:template>
```

  - Crée une ancre avec la section + son numéro, suivi d'un titre de niveau H2, comportant le titre de la section, l'enveloppe dans un A pour l'identifier, de sorte à garantir le fonctionnement des liens de la table des matières
  - Puis <xsl:apply-templates/> est appelé pour se charger du reste du travail en créant les sections

Le langage XSL

- Étape 3
  - Les deux autres sont plus simples, ils transforment les éléments <paragraphe> en éléments HTML <p>, et les éléments <important> en éléments HTML

```
<xsl:template match="paragraphe">
  <P><xsl:apply-templates/></P>
</xsl:template>
<xsl:template match="important">
<STRONG><xsl:apply-templates/></STRONG>
</xsl:template>
```

Le langage XSL

- Étape 4
  - Le dernier modèle :

    <xsl:template match="/doc/section/text()"/>

    - Permet d'identifier tout PCDATA enfant direct d'un élément <section> : autrement, son titre
    - Cela permet que ces titres n'aillent pas dans le corps de la section

Le langage XSL

Création dynamique d'éléments

<xsl:element>

- Permet d'ajouter un élément (une balise) dans l'arbre XML :
- Création de l'élément :

```
<xsl:element name="table">Mon texte</xsl:element>
```

➔ produit l'élément table avec text() dans l'arbre résultat

```
<table>Mon texte</table>
```

- Utilisation pour une création dynamique de noms d'éléments

```
<xsl:template match="nom">
  <xsl:element name="."> Mon texte</xsl:element>
</xsl:template>
```

  - va créer tout élément ayant une valeur provenant du nœud contextuel
  - en l'exécutant sur : <nom>Andrea</nom>, il produira :

```
<Andrea>Mon texte</Andrea>
```

  - en l'exécutant sur : <nom>Ify</nom>, il produira :

```
<Ify>Mon texte</Ify>
```

Le langage XSL

Création dynamique d'éléments

- Exemple12.xsl

```
<xsl:template match = "/" >
    <xsl:element name = "QQQ" >
        <xsl:element name = "{//BBB}" >
            <xsl:element name = "{//CCC}" >
                <XXX />
            </xsl:element>
        </xsl:element>
    </xsl:element>
</xsl:template>
```

- Exemple12.xml

```
<AAA >
    <BBB>bbb </BBB>
    <CCC>ccc </CCC>
</AAA>
```

- Résultat

???

Le langage XSL

Création dynamique d'éléments et d'attributs

- XSL

```
<xsl:template match = "/" >
    <xsl:element name = "QQQ" >
        <xsl:attribute name = "xxx" >111
        </xsl:attribute>
        <xsl:attribute name = "{name(//*[1])}" >
            <xsl:value-of select = "//BBB" />
        </xsl:attribute>
    </xsl:element>
</xsl:template>
```

XML

```
<AAA >
    <BBB>bbb </BBB>
    <CCC>ccc </CCC>
</AAA>
```

Résultat

???

Le langage XSL

Création dynamique d'éléments

- Exemple : soit le document carnet.xml

```
<?xml version="1.0"?>
<carnet>
  <nom prenom ="John" prenom2="Fitzgerald Johansen" famille="Doe"</nom>
  <nom prenom ="Franklin" prenom2="D." famille="Roosevelt"</nom>
  <nom prenom ="Alfred" prenom2="E." famille="Neuman"</nom>
  <nom prenom ="John" prenom2="Q." famille="Public"</nom>
  <nom prenom ="Jane" prenom2="" famille="Doe"</nom>
</carnet>
```

- On veut produire le document xml suivant :

Le langage XSL

Création dynamique d'éléments

```
<?xml version="1.0" ?>
<carnet>
<nom><prenom>John</prenom> <prenom2>Fitzgerald
    Johansen</prenom2> <famille>Doe</famille>
</nom>
<nom><prenom>Franklin</prenom><prenom2>D.</prenom2><famille>Ro
    osevelt </famille>
</nom>
<nom><prenom>Alfred</prenom><prenom2>E.</prenom2><famille>Neum
    an </famille>
</nom>
<nom><prenom>John</prenom><prenom2>Q.</prenom2><famille>Public
</famille>
</nom>
<nom><prenom>Jane</prenom><prenom2 />
    <famille>Doe</famille></nom>
</carnet>
```

Le langage XSL

Création dynamique d'éléments

- La feuille de style est : carnet.xsl
  - 1ère étape : créer le modèle à comparer à la racine du document. Ce modèle peut générer l'élément racine <carnet>, puis appeler <xsl:apply-templates> afin d'identifier tous les autres éléments

```
<xsl:template match="/">
    <carnet>
        <xsl:apply-templates select="/carnet/nom"/>
    </carnet>
</xsl:template>
```

  - 2ème étape : on doit maintenant gérer ces éléments <nom>. Pour y parvenir, on peut créer un modèle pour générer le nouvel élément <nom>, puis appeler <xsl:apply-template> pour prendre en compte tous les attributs :

```
<xsl:template match="nom">
<nom>
    <xsl:apply-templates select="@*"/>
</nom>
</xsl:template>
```

Le langage XSL

Création dynamique d'éléments

- 3ème étape : il faut un modèle pour traiter ces attributs
- Pour chaque attribut rencontré, un élément portant le même nom et la même valeur doit être généré :

```
<xsl:template match="@*">
  <xsl:element name="{name()}"><xsl:value-of select="."/>
  </xsl:element>
</xsl:template>
```

- 4ème étape : combiner ces trois modèles pour obtenir la feuille de style

Le langage XSL

Création dynamique d'éléments

- L'élément <xsl:copy>
  - Les éléments XSL Copy et Copy-of servent à copier des nœuds. L'élément Copy ne copie que le nœud courant sans ses fils, ni les attributs, tandis que l'élément Copy-of copie tout
  - Exemple :

```
<xsl:template match="ATOM">
   <xsl:copy>
         <B><xsl:value-of select="."/></B>
   </xsl:copy>
</xsl:template>
```

  - Produit entre <B> et </B> les valeurs de tous les éléments

Le langage XSL

Création dynamique d'éléments

- Autre exemple :

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

- Produit le document XML tel qu'il est

Exemple

```
<xsl:stylesheet ...>
<xsl:template match="p">
    <DIV>
        <B>
            <xsl:text>copy-of : </xsl:text>
        </B>
        <xsl:copy-of select="."/>
    </DIV>
    <DIV>
        <B>
            <xsl:text>copy : </xsl:text>
        </B>
        <xsl:copy/>
    </DIV>
    <DIV>
        <B>
            <xsl:text>value-of : </xsl:text>
        </B>
        <xsl:value-of select="."/>
    </DIV>
</xsl:template>
</xsl:stylesheet>
```

- XML

```
<source>
  <p id="a12"> Compare
    <B>these constructs</B>.
  </p>
</source>
```

- Résultat

???

Le langage XSL

- Que se passe-t-il si on applique une feuille de style vide au document ?
  - Si aucune règle ne s'applique, ce sont les règles par défaut qui s'appliquent (affichage des textes et recopie des PCDATA contenus dans les balises)
- Exemple :
  - Si on appliquait à Bibliotheque.xml la feuille de style suivante :

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

  - Le résultat sera :

N ou MAgatha ChristiePolicier-C-15Le chien des BaskervilleSir Arthur Conan DoylePolicier-D-3DuneFranck HeckbertFiction-H-1

Le langage XSL

- <xsl:output>
  - Indique à XSL ce qu'il est en train de générer
    - html, xml ou text
    - Codage des caractères : UTF-8, UTF-16 ou ISO-8859-1
      - Attention : certains processeurs ne reconnaissent pas l'UTF-16
  - Si <xsl:output> est absent et que l'élément racine de l'arbre produit est <HTML>, la méthode de sortie est "html"

```
<xsl:stysheet …>
  <xsl:output method="html" encoding="UTF-8"/>

  <xsl:template …

</xsl:stylesheet>
```

Le langage XSL

- Chargement depuis JavaScript

```
<script type="text/javascript">
// chargement du fichier XM
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("test.xml“)
// chargement du fichier XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load("test.xsl")
// transformation en Html
document.write(xml.transformNode(xsl))
</script>
```

TD

- Énoncé
  - TD-XSL-Plus : exercice 4