XSLT2

Philippe.Poulard@inria.fr

Sommaire

Différences - Spécificités

http://www.w3.org/TR/xslt20/

- Basé sur XPath 2.0 (sequences et types primitifs des W3C XML Schema)
  - Les séquences remplacent la notion de RTF ("result tree fragments")
  - Séquences: "first-class citizens"
    susceptibles d'être traités également
- Multiple documents de sortie (`<xsl:result-document>`)
- Meilleur support des groupes (`<xsl:for-each-group>`)
- Fonctions utilisateur (expressions XPath)
- Support de XML Base
- Méthodes de sortie XHTML
- Validation W3C XML Schema
- Annotations de types optionels
  sur les déclarations de variables et paramètres
- Validation dynamique
- Validation statique
- Serialisation
  - La sérialization est une spécification à part.
  - La sérialisation XSLT peut être influencée
    grâce aux attributs de `<xsl:output>`

http://www.w3.org/TR/xslt-xquery-serialization/

Modèle de données

- Dans XPath 2.0 il y a des noeuds et des valeurs typées.
- Dans XPath 2.0 il y a des séquences
  alors que XPath 1.0 avait des node sets:
  - Les séquences peuvent être dans un ordre arbitraire
  - Les séquences peuvent contenir des doublons
  - Les séquences peuvent être hétérogènes

- Les types simples de W3C XML Schema sont toujours disponibles.
- Pour utiliser des schémas additionnels, il faut les importer:

```
<xsl:import-schema
     namespace="http://www.acme.com/xmlns/acme"
     schema-location="acme.xsd"/>
```

Déclaration des types

```
<xsl:variable name="i" select="1"/>
```
est l'entier 1

```
<xsl:variable name="pi" select="3.14" as="xs:double"/>
```
est le double 3,14

```
<xsl:variable name="date" select="'2012-07-24'">
```
est la chaîne '2012-07-24'

```
<xsl:variable name="date" select="xs:date('2012-07-24')"/>
```
est la date du 24 juillet 2012

```
<xsl:variable name="date" as="xs:date" select="'2003-11-20'"/>
```
Erreur

```
<xsl:sequence select='(1,2,3,4)' as="xs:double"/>
```
est une séquence de doubles

Déclaration des types d'éléments

```
<xsl:variable name="rtf">
  <a/>
  <b/>
</xsl:variable>
```

est un RTF

```
<xsl:variable name="elemlist"
              as="element()*">
  <a/>
  <b/>
</xsl:variable>
```

est un élément

```
<xsl:variable name="elemlist"
              as="element(svg:circle)+">
  <svg:circle … >
            </svg:circle>
</xsl:variable>
```

est un élément `<svg:circle>`

Constructeurs / « as »

- Les constructeurs *`xs:type(string)`* construisent la valeur typée à partir de la valeur lexicale fournie.
- L'attribut "`as`" s'attend à une valuer du type requis.
  Peut faire des promotions de type mais pas de transtypage (*cast*).

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:html="http://www.w3.org/1999/xhtml"
                xmlns:ef="http://example.org/extension/functions"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                version="2.0">

  <xsl:output method="text"/>

  <xsl:variable name="links" select="/root/@a * /root/@b" as="xs:double"/>

  <xsl:template name="html-elements" as="element()*">
    <xsl:sequence select="//html:*"/>
  </xsl:template>

  <xsl:function name="ef:lunchp" as="xs:boolean">
    <xsl:param name="time"/>
    <xsl:call-template name="lunchtime">
      <xsl:with-param name="time" select="$time[1]" as="xs:integer"/>
    </xsl:call-template>
    <xsl:value-of select="true()"/>
  </xsl:function>

  <xsl:template name="lunchtime">
    <xsl:param name="time" select="0.0" as="xs:double"/>
    …
  </xsl:template>

  …
</xsl:stylesheet>
```

Transtypage implicite

- D'un sous-type à un type supérieur
  (`xs:NMTOKEN` quand `xs:string` est requis).
- Entre les types numériques
  (de `xs:decimal` à `xs:double`, etc.)
- Lors du calcul de la valeur booléenne effective
  (de "`NaN`" à `false()`)
- Depuis les valeurs sans type ("*untyped* *values*)

Erreurs

Si un type ne peut être transtypé dans un autre type, un *cast* provoquera une erreur.

Erreurs fréquentes:

- Opérations mathématiques sur des chaînes
  (`@attr + 1` si `attr` est validé en tant que string).
- Lexicale representation invalide
  ("`24/07/2012`" n'est pas une forme de date valide, utiliser "`2012-07-24`").
- Transtypage incompatible
  (`100 cast as r:foo`)

```
<xsl:variable name="products">
  <product cost="90" price="100"/>
  <product cost="100"/>
  <product cost="100" price="90"/>
</xsl:variable>
```

Si un produit à un prix, alors c'est le prix, sinon c'est 150% de son coût

```
<xsl:variable name="prices">
  <xsl:for-each select="$products/product">
    <xsl:choose>
      <xsl:when test="@price">
        <xsl:sequence select="xs:decimal(@price)"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:sequence select="xs:decimal(@cost) * 1.5"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:variable>
```

Alternative

```
<xsl:value-of
     select="for $p in products return
               if ($p/@price)
               then xs:decimal($p/@price)
               else xs:decimal($p/@cost) * 1.5"/>
```

Valeurs, copies, et séquences

Quelle est la différence entre
<xsl:value-of>, <xsl:copy-of>, et <xsl:sequence> ?

- <xsl:value-of> crée un noeud texte.
- <xsl:copy-of> crée une copie.
- <xsl:sequence> retourne les noeuds sélectionnés, avec une possible atomisation

Les séquences peuvent s'étendrent avec d'autres <xsl:sequence>

Voir `http://www.unicode.org/reports/tr10/`

- Les collations concernent XPath 2.0 et XSLT 2.0
- Les collations déterminent comment opérent les tris et la collation
- Les collations sont identifiées par URI
- Tous les processors supportent "Unicode code-point collation"

Fonction XPath2 de comparaison:

```
compare("Pêche", "Péché", "fr-FR")
```

→

Péché
Pêche

```
compare("Pêche", "Péché", "en-US")
```

→

Pêche
Péché

Grouper les données en XSLT 1.0 est difficile.

XSLT 2.0 offre une nouvelle instruction de groupage flexible pour grouper:

- Sur une clé spécifique
- Par transitions au début de chaque groupe
- Par transitions à la fin de chaque groupe
- Par des valeurs de clé adjacentes

Et les groupes peuvent aussi être triés

Grouper selon une clé

Grouper les données par pays:

```
<cities>
  <city name="Milano"  country="Italia"/>
  <city name="Paris"   country="France"/>
  <city name="München" country="Deutschland"/>
  <city name="Lyon"    country="France"/>
  <city name="Venezia" country="Italia"/>
</cities>
```

```
<xsl:for-each-group select="cities/city"
                    group-by="@country">
  <tr>
   <td><xsl:value-of select="position()"/></td>
   <td><xsl:value-of select="@country"/></td>
   <td>
    <xsl:value-of select="current-group()/@name"
                  separator=", "/>
   </td>
  </tr>
</xsl:for-each-group>
```

```
<tr>
   <td>1</td>
   <td>Italia</td>
   <td>Milano, Venezia</td>
</tr>
<tr>
   <td>2</td>
   <td>France</td>
   <td>Paris, Lyon</td>
</tr>
```

Grouper par valeur de départ

Grouper les données à chaque `h1` dans une division `div`

```
<body>
  <h1>Introduction</h1>
  <p>XSLT is used to write stylesheets.</p>
  <p>XQuery is used to query XML databases.</p>
  <h1>What is a stylesheet?</h1>
  <p>A stylesheet is an XML document used to define a transformation.</p>
  <p>Stylesheets may be written in XSLT.</p>
  <p>XSLT 2.0 introduces new grouping constructs.</p>
</body>
```

```
<xsl:for-each-group select="*"
                    group-starting-with="h1">
  <div>
    <xsl:apply-templates
         select="current-group()"/>
  </div>
</xsl:for-each-group>
```

```
<div>
   <h1>Introduction</h1>
   <p>XSLT is used to write stylesheets.</p>
   <p>XQuery is used to query XML databases.</p>
</div>
<div>
   <h1>What is a stylesheet?</h1>
   <p>A stylesheet is an XML document used to define a transformation.</p>
   <p>Stylesheets may be written in XSLT.</p>
   <p>XSLT 2.0 introduces new grouping constructs.</p>
</div>
```

Grouper par valeur de fin

Grouper les données de telle sorte que les pages continues soient dans un `pageset`

```
<doc>
  <page continued="yes">Some text</page>
  <page continued="yes">More text</page>
  <page>Yet more text</page>
  <page continued="yes">Some words</page>
  <page continued="yes">More words</page>
  <page>Yet more words</page>
</doc>
```

+

```
<xsl:for-each-group select="*"
     group-ending-with="page[not(@continued
                                 ='yes')]">
  <pageset>
    <xsl:for-each select="current-group()">
      <page><xsl:value-of select="."/></page>
    </xsl:for-each>
  </pageset>
</xsl:for-each-group>
```

→

```
<doc>
   <pageset>
      <page>Some text</page>
      <page>More text</page>
      <page>Yet more text</page>
   </pageset>
   <pageset>
      <page>Some words</page>
      <page>More words</page>
      <page>Yet more words</page>
   </pageset>
</doc>
```

Grouper par valeur de clé adjacentes

Grouper les données de telle sorte que les listes ne soient pas dans des paragraphes

```
<p>Do <em>not</em>:
    <ul>
    <li>talk,</li>
    <li>eat, or</li>
    <li>use your mobile telephone</li>
    </ul>
while you are in the cinema.</p>
```

+

```
<xsl:for-each-group select="node()"
     group-adjacent="self::ul or self::ol">
  <xsl:choose>
    <xsl:when test="current-grouping-key()">
      <xsl:copy-of select="current-group()"/>
    </xsl:when>
    <xsl:otherwise>
      <p>
        <xsl:copy-of
             select="current-group()"/>
      </p>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each-group>
```

→

```
<p>Do <em>not</em>:
</p>
<ul>
   <li>talk,</li>
   <li>eat, or</li>
   <li>use your mobile telephone</li>
</ul>
<p>
   while you are in the cinema.
</p>
```

Il y a 3 fonctions d'expressions régulières:

- `matches()` teste si une expression régulière correspond à une chaîne.
- `replace()` utilise une expression régulière pour remplacer des portions de chaîne.
- `tokenize()` retourne une séquence de chaînes formée par la coupe d'une chaîne selon un séparateur fourni.

```
$Id: matches.xsl,v 1.3 2006-08-10 21:12:03 ndw Exp $
```

```
<xsl:choose>
  <xsl:when test="matches(., '[1-9][0-9]{3}-[0-1][0-9]-[0-3][0-9]')">
    <xsl:value-of
        select="replace(., '([1-9][0-9]{3})-([0-1][0-9])-([0-3][0-9])', '$2/$3/$1')"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:value-of select="."/>
  </xsl:otherwise>
</xsl:choose>
```

```
$Id: matches.xsl,v 1.3 08/10/2006 21:12:03 ndw Exp $
```

Instructions

- `<xcsl:analyze-string>` utilise une expression régulière pour définir du marquage sur une chaîne

```
<xsl:analyze-string select="…" regex="…">
  <xsl:matching-substring>…</xsl:matching-substring>…
  <xsl:non-matching-substring>…</xsl:non-matching-substring>…
</xsl:analyze-string>
```

- La fonction `regex-group()` permet de référencer les groupes capturés

```
12/8/2003
```

```
<xsl:analyze-string select="$date" regex="([0-9]+)/([0-9]+)/([0-9]{{4}})">
  <xsl:matching-substring>
    <xsl:number value="regex-group(3)" format="0001"/>
    <xsl:text>-</xsl:text>
    …
```

```
2003-12-08
```

Définition

```
<xsl:function name="str:reverse" as="xs:string">
  <xsl:param name="sentence" as="xs:string"/>
  <xsl:value-of separator=" ">
    <xsl:choose>
      <xsl:when test="contains($sentence, ' ')">
          <xsl:sequence select="str:reverse(substring-after($sentence, ' '))"/>
          <xsl:sequence select="substring-before($sentence, ' ')"/>
      </xsl:when>
      <xsl:otherwise>
          <xsl:sequence select="$sentence"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:value-of>
</xsl:function>
```

Utilisation

```
<xsl:value-of select="str:reverse('DOG BITES MAN')>
```

```
MAN BITES DOG
```

Extension de fonctions:

•Etend la librairie de fonctions

•La plupart des processeurs XSLT 1.0 supportaient déjà l'extension de fonctions.

Extensions d'éléments:

•Etend la librairie d'instructions (éléments)

S'utilisent en combinaison avec l'attribut "`use-when`"

```
<xsl:function name="f:gcd" use-when="function-available('gcd:distance')">
  <xsl:param name="lat1"/>
  <xsl:param name="lon1"/>
  <xsl:param name="lat2"/>
  <xsl:param name="lon2"/>
  <xsl:value-of select="gcd:distance($lat1,$lon1,$lat2,$lon2)"/>
</xsl:function>

<xsl:function name="f:gcd" use-when="not(function-available('gcd:distance'))">
  <xsl:param name="lat1"/>
  <xsl:param name="lon1"/>
  <xsl:param name="lat2"/>
  <xsl:param name="lon2"/>
  <xsl:message>Warning: cannot calculate great circle distance!</xsl:message>
  <xsl:value-of select="1000"/>
</xsl:function>
```

- Les feuilles de style peuvent être incluses ou importées
- `<xsl:include>` permet la modularité du code
- `<xsl:import>` apporte la modularité logique
- `<xsl:apply-import>` permet à une feuille de style d'appliquer une règle d'une feuille de style importée
- Nouveauté: `<xsl:next-match>`

`<xsl:next-match>`

- Si plusieurs *templates* correspondent, ils sont triés par priorité
- Le *template* de plus haute priorité est exécuté, les autres ne le sont pas
- `<xsl:next-match>` permet d'exécuter le template suivant de plus haute priorité indépendamment de la précédence de feuille de style

```
<products>
  <product id="p1" name="Delta" price="3250" stock="4"/>
  <product id="p2" name="Golf" price="1000" stock="5"/>
  <product id="p3" name="Alpha" price="1200" stock="19"/>
</products>
```

+

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
  <xsl:output indent="yes"/>
  <xsl:template match="/">
    <PRODUCTS>
      <xsl:apply-templates/>
    </PRODUCTS>
  </xsl:template>
  <xsl:template match="product">
    <PRODUCT id="{@id}" price="{@price}" stock="{@stock}"/>
  </xsl:template>
  <xsl:template match="product[@id = 'p2']">
    <PRODUCT id="{@id}" price="{@price * 1.25}" stock="{@stock}"/>
  </xsl:template>
  <xsl:template match="product|product[@id = 'p2']">
    <xsl:comment>
      <xsl:value-of select="concat(' ', @name, ' ')"/>
    </xsl:comment>
    <xsl:text>&#xA;</xsl:text>
    <xsl:next-match/>
    </xsl:template>
</xsl:stylesheet>
```

→

```
<PRODUCTS>
  <!-- Delta -->
  <PRODUCT id="p1" price="3250" stock="4"/>
  <!-- Golf -->
  <PRODUCT id="p2" price="1250" stock="5"/>
  <!-- Alpha -->
  <PRODUCT id="p3" price="1200" stock="19"/>
</PRODUCTS>
```

Production multiple de documents


```
<products>
  <product id="p1" name="Delta" price="3250" stock="4" country="Denmark"/>
  <product id="p2" name="Golf" price="1000" stock="5" country="Germany"/>
  <product id="p3" name="Alpha" price="1200" stock="19" country="Germany"/>
  <product id="p4" name="Foxtrot" price="1500" stock="5" country="Australia"/>
  <product id="p5" name="Tango" price="1225" stock="3" country="Japan"/>
</products>
```

+

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:for-each select="products/product">
      <xsl:result-document method="xml" href="product_{@id}-output.xml">
        <xsl:copy-of select="."/>
      </xsl:result-document>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

product p1-output.xml

```
<product id="p1" name="Delta" price="3250" stock="4" country="Denmark"/>
```

product_p2-output.xml

```
<product id="p2" name="Golf" price="1000" stock="5" country="Germany"/>
```

product p3-output.xml

```
<product id="p3" name="Alpha" price="1200" stock="19" country="Germany"/>
```

product p4-output.xml

```
<product id="p4" name="Foxtrot" price="1500" stock="5" country="Australia"/>
```

product p5-output.xml

```
<product id="p5" name="Tango" price="1225" stock="3" country="Japan"/>
```