XPath

Introduction à XPath

Généralités[1]

Voir la recommandation du W3C sur XPath :

- l'officielle sur le site du W3C ; http://www.w3.org/TR/1999/REC-xpath-19991116.html
- une version française hébergée par XMLfr. http://xmlfr.org/w3c/TR/xpath/

Le langage XSLT se base sur le langage XPath pour faire réference aux différents noeuds composant un document XML. C'est notamment ce qui est utilisé dans les attributs "match" de la balise "template", "select" de la balise "apply-templates" ou encore "select" de la balise "value-of" (et qui ont été surlignés dans l'exemple suivant).

```
<?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"/>
   <xsl:template match="/">
      <html>
         <head><title>Annuaire</title></head>
         <body>
            <xsl:apply-templates/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="annuaire">
      <table border="1">
         <tr><th>Nom</th><th>Prenom</th><th>email</th></tr>
         <xsl:apply-templates/>
      </table>
   </xsl:template>

   <xsl:template match="personne">
      <tr><td><xsl:value-of select="@type"/>: <xsl:value-of select="nom"/></td>
         <td><xsl:value-of select="prenom"/></td>
         <td><xsl:value-of select="email"/></td></tr>
   </xsl:template>
</xsl:stylesheet>
```

XPath est un langage d'interrogation des documents XML. Il permet de sélectionner certaines parties d'un document XML : des sous-arbres, des noeuds, des attributs, etc.

XPath est central dans le monde XML, il intervient comme brique de base dans d'autres technologies XML :

- XML Schémas (expression des contraintes d'unicité et de clefs),
- les transformations XSLT,

1 Une partie de notes est reprise par toutestfacile.com.

- XQuery
- XLink
- XPointer, etc.

La syntaxe générale

Dans les exemples vus jusque là, l'utilisation de XPath c'est généralement limité à préciser "." (le noeud courant), un nom de balise (un sous-noeud) ou encore "@attribut" (l'attribut du noeud courant). En fait, il est possible de faire référence à n'importe qu'elle autre noeud en utilisant une notation toute simple (comme celle que l'on utilise pour faire référence à un fichier).

1. / indique le noeud racine
2. . indique le noeud courant
3. .. indique le noeud parent

Ainsi

1. /niveau1 indique le noeud appelé "niveau1" sous le noeud racine
2. ./ssniveau indique le noeud appelé "ssniveau" sous le noeud courant

Comme vous l'avez compris, c'est le caractère '/' qui sert de délimiteur des niveaux d'arborescence. Il faut toutefois noter, qu'il est possible d'indiquer un niveau quelconque d'arborescence en utilisant la notation "//".
Ainsi

.//ssniveau indique un noeud appelé "ssniveau" se trouvant à n'importe quelle profondeur sous le noeud courant. Le premier concept est celui de *noeud courant* : c'est l'endroit d'où l'on part. En première lecture on peut imaginer qu'il s'agit de la racine du document, mais n'importe quel noeud peut jour ce rôle. À partir de là, on considère trois éléments :

- un *axe* : la direction dans laquelle on se dirige à partir du noeud courant (vers le père, vers les fils, vers les frères de gauche, etc.) ;
- un *filtre* : le type de noeuds qui nous intéresse dans l'axe choisi (des noeuds quelconques, des éléments quelconques ou un élément précis, des commentaires, etc.) ;
- un *prédicat* optionnel : des conditions supplémentaires pour sélectionner des noeuds parmi ceux retenus par le filtre dans l'axe.

Ils constituent à eux trois une *étape* et on les note `axe::filtre[prédicat]`. L'enchaînement de plusieurs étapes constitue une *chemin XPath* :

axe1::filtre1[prédicat1]/axe2::filtre2[prédicat2]

Exemple concret : La syntaxe générale
sparent::*/child::node()[position()=2]

Si le chemin commence par un `/`, il s'agit d'un chemin absolu, c'est-à-dire prenant son origine à la racine du document et non pas au noeud courant.

Il est possible de faire une disjonction de requêtes XPath avec le signe `|` ; on obtient la syntaxe générale alors de l'union des deux ensembles de noeuds correspondants.
axe1::filtre1[prédicat1]/axe2::filtre2[prédicat2] | axe3::filtre3[prédicat3]

Chaque étape renvoie un ensemble de noeuds et pour chacun d'entre eux, on applique les étapes suivantes.

Les axes

- `self` : le noeud courant lui-même ;
- `child` : les enfants du noeud courant ;
- `descendant`, `descendant-or-self` : tous les descendants du noeud courant ;
- `parent` : le père du noeud courant ;
- `ancestor`, `ancestor-or-self` : les ancêtres du noeud courant ;
- `attribute` : les attributs du noeud courant ;
- `preceding`, `following` : les noeuds, précédants ou suivants, du noeud courant, dans l'ordre de lecture du document ;
- `preceding-sibling`, `following-sibling` : les frères, précédant ou suivant, le noeud courant ;
- `namespace` : les espaces de noms.

Les filtresLa syntaxe générale

- `node()` : tous les noeuds ;
- `text()` : les noeuds textuels ;
- * : tous les éléments ;
- *nom* : les éléments portant ce *nom* ;
- `comment()` : les noeuds commentaires ;
- `processing-instruction('cible')` : les noeuds instructions, seulement les instructions *cible* si cet argument est fourni.

Les prédicats

Ils prennent la forme de tests que les noeuds sélectionnés devront vérifier. Ces tests peuvent impliquer des fonctions ou de nouveaux chemins XPath.

```
child::toto[position()=2]
```

```
child::toto[@ref='id125']
```

Il est possible de combiner ces tests à l'aide des opérateurs logiques classiques (`and`, `or` et `not`) ou de les enchaîner :

```
child::toto[@ref='id125' or @ref='id47']
```

```
child::toto[contains(text(),'coucou') and position()=2]
```

```
child::toto[contains(text(),'coucou')][position()=2]
```

Les deux dernières requêtes ne sont pas équivalentes :

- la première renvoie le deuxième fils `toto` si celui-ci contient le texte *coucou* ;
- la seconde sélectionne tous les fils `toto` qui contiennent le texte *coucou* et parmi ceux-ci renvoie le deuxième.

Les fonctions

Ces fonctions peuvent apparaître dans des prédicats ou être utilisées directement dans un évaluateur d'expressions Xpath.

- position()

- last()
- contains(elem, elem)
- concat(elem, elem)
- starts-with(elem, elem)
- substring(elem, elem)
- count(elem)
- id(elem)
- name(elem)
- local-name(elem)
- namespace-uri(elem)

Exemples

```
/talk/slide[contains(title, 'XSLT')]

/talk/slide[count(*)>5]
```

Il y a des fonctions sur les chaînes de caractères et qui vont donc porter sur les contenus textuels de noeuds :

- `concat` : colle ensemble les chaînes de caractères passées en paramètres ;
- `string-length` : la longueur de la chaîne fournie ;
- `contains`, `starts-with`, `ends-with` : tests d'appartenance d'une chaîne dans une autre.

Des fonctions qui prennent en argument une requête XPath et vont donc porter sur des ensembles de noeuds :

- `count` : le nombre de noeuds dans l'ensemble sélectionné par la requête ;
- `name` : le nom de l'élément courant.

Et enfin, nous disposons de fonctions sans paramètre mais liées au noeud courant :

- `position` : le numéro du noeud courant dans la liste des noeuds considérés ;
- `last` : le nombre de noeuds sélectionnés à l'étape courante.

Extensions de XPath Pour XQuery

- Sequence : collection ordonnée de valeurs ou de noeuds
- union, intersection, except

```
if (Exp) then Exp else Exp

some | every Variable in Exp satisfies Exp

every $person in //team satisfies $person/@fulltime
```

Toutes ces références peuvent être affinées selon le modèle chemin[condition] pour n'obtenir que les noeuds pointé par "chemin" qui répondent à la condition donnée.

Les conditions

Les conditions peuvent s'appuyer sur des fonctions dont

1. position() qui retourne l'indice du noeud
2. last() qui retourne l'indice du dernier noeud

sur les valeurs des attributs via

1. attribute::nomattribut

et les opérateurs

1. or, and, =, !=, <=, <, >=, >

On peut donc avoir des expressions du genre

1. ./ssniveau[position()<3] pour avoir les 2 premiers sous niveaux nommé "ssniveau"
2. /ssracine[position()=last()] pour avoir le dernier élément "ssracine"
3. ./ssniveau[attribute::nomattribut="nomcherche"] pour avoir ceux ayant nomattribut="nomcherche"
4. ./[ssniveau or ssniveau2] pour avoir les élément nommés "ssniveau" ou "ssniveau2"

Ce qui peut s'écrire de façon abrégée

1. /ssracine[last()] pour avoir le dernier élément "ssracine"
2. ./ssniveau[@nomattribut="nomcherche"] pour avoir ceux ayant nomattribut="nomcherche"

Les conditions sur la nature de l'élément

Il est également possible (notamment pour les attributs match) de selectionner selon le type de l'élément.
Pour cela, nous disposons des fonctions *text()* s'il s'agit du contenu d'une balise, *processing-instruction()* s'il s'agit d'une instruction de traitement ou encore *comment()* s'il s'agit d'un commentaire.

Combiner les conditions

Il est possible d'appliquer le même template pour différents types ou sélections de noeud en précisant les différents conditions séparées par un |.

La syntaxe abrégée

Cette notation est plus simple mais pas aussi expressive que la notation étendue. De plus, l'équivalent étendu de certaines notations abrégées n'est pas toujours celui que l'on pense.

Syntaxe abrégée	Syntaxe étendue
.	self::node()
toto	child::toto
../toto	parent::toto

@titi	attribute::titi
//toto	/descendant-or-self::node()/child::toto
.//toto	descendant-or-self::node()/child::toto
toto[2]	child::toto[position() = 2]

Notez bien que la notation `//` implique de repartir depuis la racine ; si l'on veut un `descendant-or-self` depuis le noeud courant, on écrira `.//`.

Il faut prendre garde également au fait que `//toto[2]` n'est pas équivalent à

```
/descendant-or-self::toto[position()=2]
```

mais à

```
/descendant-or-self::node()/child::toto[position()=2],
```

autrement dit :

- `//toto[2]` fournit tous les noeuds `toto` qui sont deuxième fils de leur père ;
- `/descendant-or-self::toto[position()=2]` désigne un unique noeud, le deuxième noeud `toto` du document.