XPath

Wieslaw Zielonka

23 novembre 2013

XPath

Un document XML est arbre. Pour naviguer dans ce document on utilise les expression de XPath.
XPath permet de spécifier les chemins absolus ou relatifs dans l'arbre de document. XPath possède plus de 100 fonctions.
XPath est un langage des expressions et non un langage de programmation.

Deux versions de XPath 1.0 (1999) et 2.0 (2007).

On suppose que les références d'entité sont résolues avant d'entrer dans XPath.

XPath - types de sommets

- **noued élément** – qui peut contenir d'autres éléments,
- **noeud attribut** – XPath considère les attributs comme les noeuds séparés, différents des enfants, dans l'arbre de document. Le parent d'un noeud attribut est le noeud élément qui contient cet attribut.
- **noued texte** – c'est un noeud *textuel* qui contient le contenu texte d'un noeud élément (de son parent).
- **noeud commentaire** — contient un commentaire,
- namespace node – obsolète,
- **processing instruction node - instruction de traitement** — instructions de traitement dans <? ... ?>
- **document node** (root node) – c'est un élément à la racine. Il ne faut pas le confondre avec le noeud élément qui est à la racine, le *noued élément* à la racine est un enfant de document node.

exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- livre Potter -->
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- author wz -->
<cv non="Dupont" prenom="Pierre">
  <?print qualite="gras"?>
  CV de Pierre
  <!-- autre commentaire-->
  <A age="4">
    <?print qualite="italic"?>
    enfance
  </A>
  <A age="16">
    ecole
  </A>
  fin de CV
</cv>
```

exemple - suite

La racine / du document précédent a deux enfants, un commentaire et le noeud cv :

type de noeud	nom de noeud	valeur DOM
comment		author wz
element	cv	

exemple - suite

L'élément cv a 9 enfants :

type de noeud	nom de noeud	valeur DOM
text		
processing instruction	print	qualite="gras"
text		CV de Pierre
comment		autre commentaire
text		
element	A	
text		
element	A	
text		fin de CV

(noter l'insertion de texte vide entre différents enfants qui ne sont pas de type text)

Types de noeuds

- Root node,
- Element node,
- Attribute node (ne sont pas enfant de element node qui les contient),
- Namespace node (ne sont pas enfant de element node qui les contient),
- Procession instruction node,
- Comment node,
- Text node.

Modèle général

Évaluation de l'expression XPath donne un objet.
Types d'objets : `node-set`, booléen, nombre, string.
`node-set` un ensemble non ordonné de noeuds.

Contexte

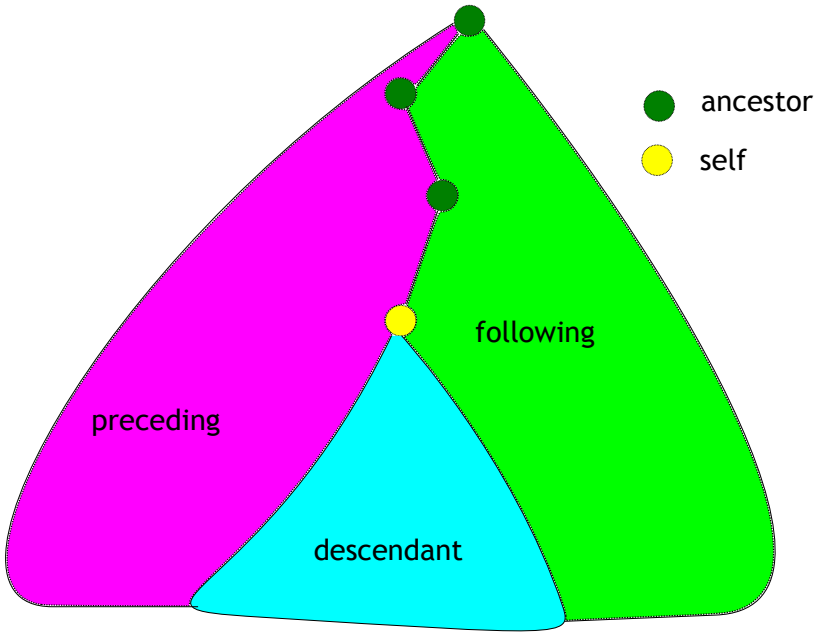
Chaque expression évaluée dans un contexte. Le contexte est définie par l'application qui utilise XPath. Le contexte consiste de :

- a noeud : noeud de contexte (context node),
- la position de contexte et taille de contexte (context position, context size). La position de contexte spécifie la position de noeud de contexte dans le contexte. Context position et context size - des entiers positifs.
- variable bindings (variable $\rightarrow$ valeur),
- librairie de fonctions (nom de fonction $\rightarrow$ fonction),
- ensemble de déclaration des espaces de noms.

XPath – les axes de navigation

syntaxe	
descendant	tous les descendant (enfants, petits-enfants etc.) du noeud courant
descendant-or-self	descendants plus le noeud courant
ancestor	les ascendant : parent, grand-parent etc. du noeud courant
ancestor-or-self	les ascendant plus le noeud courant
preceding	tous les noeuds avant le noeud courant dans le document mais qui ne sont pas les ascendants du noeud courant
following	les noeuds après le noeud courant dans l'ordre de document mais qui ne sont pas les descendants du noeud courant
attribute	attributs du noeud courant

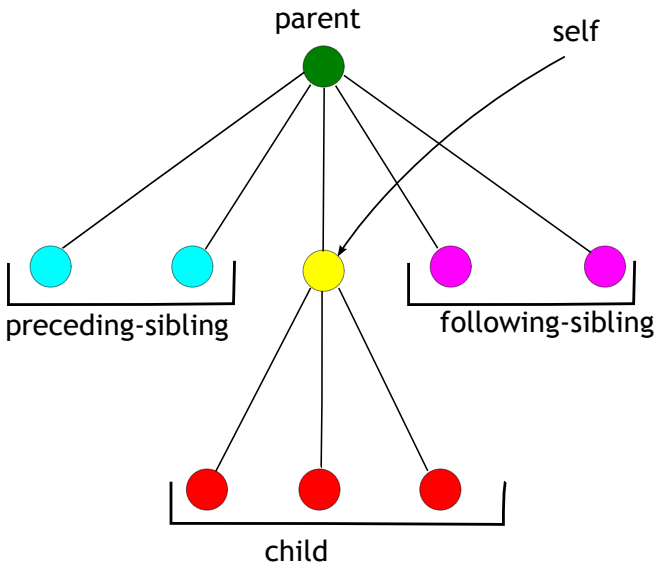
Les axes



Les axes — suite

syntaxe	
self	le noeud courant
child	les enfants du noeud courant
preceding-sibling	les frères à gauche du noeud courant
following-sibling	les frères à droite du noeud courant
parent	le parent
namespace	les sommets namespace du noeud courant

Les axes



XPath - emplacements et chemins

Le chemin (location path) est une suite d'emplacements séparés par / Les chemins relatifs et absolus (commencent par /).

Chaque emplacement a la forme suivante :

```
axe::type[predicat]
```

- `axe` – indique dans quelle direction chercher
- `type` – permet de sélectionner de noeuds d'un certain type
- `predicat` – est une condition qui permet de raffiner la recherche.

Exemple

```
child::planet[position()=5]
```

`child` – l'axe qui indique qu'il faut sélectionner les enfants du noeud courant,
`planet` indique de choisir parmi ces enfants les éléments `<planet>`
le prédicat `position()=5` est vrai si la fonction `position()` retourne 5 c'est qui est vrai pour le 5ième enfant `planet` du noeud courant

Puisque l'axe `child` est un axe par défaut on peut aussi écrire :

```
planet[position()=5]
```

Examples

- `attribute::name`
  sélectionne l'attribut `name` de context node,
- `/descendant::numlist/child::item`
  sélectionne tous éléments textttitem qui ont parent `numlist`
- `child::para[position()=1]`
  le premier enfant `para` de context node,
- `/descendant::figure[position()=42]`
  le 42ème élément `figure` dans le document,
- `/child::doc/child::chap[position()=5]/child::sect[position()=2]`
  le 2ème enfant du 5ème enfant de l'élément `doc`,

Exemples - cont

- `child::para[attribute::type='warning'][position()=5]`
  5ème enfant `para` de context node qui possède l'attribut `type` dont la valeur est `'warning'`,
- `child::para[position()=5][attribute::type="warning"]`
  5ème enfant `para` de context node à condition que cet enfant possède l'attribut `type` de valeur ``'warning''``, sinon rien.

XPath – type

*	tous les éléments et attributs de l'axe utilisé (et seulement les éléments et attributs)
`name`	élément dont le nom est `name`
`node()`	tout type de noeuds
`text()`	tous les noeud textuels
`comment()`	tous les commentaires
`processing-instruction()`	les instructions de traitement
`processing-instruction('web')`	processing instruction avec le nom `web`

Exemples

basés sur `planets.xml`

`child::text()`
retourne tous les enfants qui sont des noeuds textuels du noeud courant

`child::node()`
retourne tous les enfants du noeud courant peu importe leur type

`child::planet[position() = last()]`
le dernier enfant du sommet courant de type `element` dont le nom est `planet`

`descendant::comment()`
tous les noeuds descendants du noeud courant de type commentaire `comment`

`child::*`
tous les enfants du noeud courant

`attribute::units` retourne l'attribut `units` du noeud courant

Noter que les attributs ne sont pas comptés parmi les enfants de noeud courant, en particulier `child::node()` ne donne aucun attribut du noeud courant.

`descendant::planet`
les éléments descendants de nom `<planet>` du noeud courant

Syntaxe abrégée

`child::` peut être omis

	peut être remplacé par
`attribute::`	`@`
`/descendant-or-self::node()`	`//`
`self::node()`	`.`
`parent::node()`	`..`
`[position()=i]`	`[i]`

XPath - exemples d'abréviations

`child::paragraphe[attribute::lang=''FR'']`

sélectionne les enfants du noeud courant et dont le nom est `paragraphe`, ensuite parmi ses enfants
`[attribute::lang=''FR'']` permet de choisir ces éléments qui possèdent l'attribut `lang` dont la valeur est `''FR''`
En utlisant la syntaxe abrégée cette expression est équivalente à

`paragraphe[@lang=''FR'']`

Exemples d'abréviations

`/child::doc/child::chap[position()=5]/child::sect[position()=2]`
`/doc/chap[5]/sect[2]`

`child::para[attribute::type='warning'][position()=5]`
`para[@type='warning'][5]`

`child::para[position()=5][attribute::type="warning"]`
`para[5][@type="warning"]`

Exemples de chemins

Un chemin c'est une suite de pas séparés par /.

Le chemin qui commence par / c'est un chemin absolu qui part du noeud racine, dans le cas contraire chemin relatif par rapport au noeud courant.

Exemples pour `planets.xml`

`child::*/child::planet`
tous les grands enfants <planet> du noeud courant, équivalent à `*/planet`

`/descendant::planet/child::name`
tous les éléments <name> qui ont <planet> comme parent, équivalent à `//planet/name`

Exemples de chemins

`/child::planets/child::planet[position() = 3 ]/child::name`
l'enfant <name> de troisième enfant <planet> du noeud <planets>, équivalent à
`/planets/planet[position() = 3 ]/name`
ou même encore plus court
`/planets/planet[3]/name`

Exemples - pets

L'exemple suivant utlise le document `pets.xml`
L'expression XPath :
`/pets/*/type`
donne :

```
<type>chat</type>
<type>chien</type>
<type>chat</type>
<type>chien</type>
<type>chat</type>
<type>cochon</type>
```

c'est-à-dire tous les noeuds <type> petits enfants de <pets>.

Exemples - pets (cont)

`/pets/*/type/preceding-sibling::node()`
donne

```
<name>Puce</name>
<age>14</age>
<name>Amanda</name>
<age>10</age>
<name>Jack</name>
<age>3</age>
<name>Bernard</name>
<age>12</age>
<name>Naia</name>
<age>1</age>
<name>Stop</name>
<age>5</age>
```

c'est-à-dire les frères aînés de noeuds qui ont été sélectionnés précédemment.

Exemples - pets (cont)

Et finalement
/pets/*/type/preceding-sibling::node()/text()
donne

Puce14Amanda10Jack3Bernard12Naia1Stop5

c'est-à-dire les enfants de type texte de noeuds sélectionnés sur le transparents précédent (notez qu'il n'y a pas d'espaces pour séparer les éléments texte sélectionnés).

Quelques exemples

Exemples pour le document po.xml (ordres d'achats) :
L'expression
//node()[@PartNumber] donne tous les éléments qui possèdent l'attribut PartNumber

et
//@PartNumber/..
la même chose obtenue autrement.

Noter que dans un noeud :

<Item PartNumber="872-AA">

PartNumber n'est pas un enfant de Item mais son attribut (il faut suivre l'axe attribute depuis Item pour arriver à PartNumber). Par contre le parent de l'attribut PartNumber est bien le noeud Item

Quelques exemples

//@PartNumber/parent::node()/child::node()[2]
le deuxième noeud enfant du parent de chaque attribut PartNumber

//@PartNumber/parent::node()/child::node()[position()>2]
d'abord on va vers l'attribut PartNumber, ensuite vers son parent (i.e. vers les noeuds qui possèdent cet attribut), ensuite vers tous les enfants des noeuds choisis à l'étape précédent, et finalement on sélectionne tous les enfants à partir du troisième.

//Item/parent::node()
//Item/ancestor::node()/child::node()[2]
/*/*/Address

/*/*/Address[attribute::Type="Billing"]
les éléments <Address> qui sont des petits enfants de la racine et qui possèdent l'attribut Type dont la valeur est ''Billing''

Prédicats multiples

Il est possible de spécifier plusieurs prédicats

axe::type[predicat1][predicat2]...[predicats n]

Exemple

/descendant::planet[position()=3]
[attribute::language="English"] // le troisième noeud planet s'il possède l'attribut language de valeur ''English''.

Rôle de prédicats

Les prédicats permettent de raffiner la recherche. Le prédicat est appliqué à chaque noeud dans node-set. S'il est évalué true alors le noeud est sélectionné, sinon il est supprimé de node-set.
Le prédicat peut être lui-même une expression complète de XPath.

`/descendant::chapter[attribute::author][attribute::date]`
tous les éléments `chapter` qui possèdent les attributs `author` et `date`.

`/descendant::chapter[descendant::figure][descendant::table]`
tous les éléments `chapter` qui possèdent des éléments descendants `figure` et `table`.

Prédicats imbriqués

Nous pouvons imbriquer les prédicats comme :
`//project[descendant : :name[preceding-sibling : :active]]`
Comparer avec :
`//project[descendant : :name][preceding-sibling : :active]`

Opérateurs booléens

Dans les prédicats on utilise toutes les conditions et opérateurs habituels
`< > <= >= = != false() true() not()`

Exemple
`//*[not(@units)]` — donne tous les éléments qui n'ont pas d'attribut `units`

`boolean()` appliqué à `number` retourne true si le nombre différent de 0

`boolean()` appliqué à `string` retourne true si le string non vide
`boolean()` appliqué à `node-set` retourne true si le le node-set est non vide

Opérateurs logiques :
`and or`
Exemple `//*[not(@units) and @nom='Mercury']` — donne tous les éléments qui ne possèdent pas d'attributs `units` mais possèdent l'attribut `nom` avec la valeur `Mercury`.

Opérateurs et fonctions numériques

`+ - * div mod ceiling() floor()`
`round() sum() number()`

`div` – la division (et non pas division entière)
`number` – transforme son argument en nombre
`sum` – la somme de valeurs numériques des éléments de node-set

Exemple
`sum(//USPrice)` ou `sum(//USPrice/text())`
`USPrice` sont des éléments qui contiennent un texte

Fonctions sur les strings

`concat(string s1, string s2,...)`

`contains(string s1,string s2)`
true si s1 contient s2

`string-length(string s1)`

`substring(string s, number offset, number length)`

`string-join(string* s1, string s2)`
cette fonction prend comme premier argument une suite de string et les concatène en utilisant s2 comme séparateur.
Par exemple
`string-join(/pets/*/type/preceding-sibling::node()/text(), ' ')` retourne
`Puce 14 Amanda 10 Jack 3 Bernard 12 Naia 1 Stop 5`
(voir exemple pets dans un transparent précédent).

`normalize-space(string string1)` – supprime les espaces au début et à la fin et remplace chaque suite d'espaces par une espace

et beaucoup beaucoup d'autres fonctions pour traiter les strings.

Opération sur les ensembles de noeuds

`count(node-set)`
retourne le nombre de noeuds dans `node-set`

`position()`
retourne la position du noeud courant dans le node-set courant, les positions commencent à 1 et la dernière position est `last()`

`last()`
retourne le nombre de noeuds dans le node-set composé des noeuds courants est de ces frères et soeurs.

`name(node-set)` - retourne le nom du premier noeud de le node-set

`local-name(node-set)` – retourne (comme chaîne de caractères) le nom du premier noeud dans `node-set`

Exemples

Les deux expression suivantes sont évaluées sur `planets.xml`
`sum(//radius) div count(//radius)`
rayon moyen de toutes les planètes.

`//*[local-name()="radius"]` — les noeuds dont le nom est `radius`.

L'expression suivante est évaluée sur le fichier `po.xml`
`//PurchaseOrder[count(descendant::Item)>=2]` — tous les noeuds `PurchaseOrder` qui possèdent au moins deux descendants `Item`

Union des ensembles de noeuds

Nous pouvons combiner plusieurs expressions de chemin avec `|` pour obtenir l'union.

`//text()|//comment()`
donne la liste de tous les noeuds texte et tous les noeuds commentaires.

`//a[ancestor::ul | ancestor::ol]` — hyperliens dans `ol` ou `ul`.

XPath 2

Les nouveaux types :

- xs :string
- xs :boolean
- xs :decimal
- xs :float
- xs :double
- xs :duration
- xs :dateTime
- xs :time
- xs :date
- xs :gYearMonth
- xs :gYear

XPath 2 - nouveaux types

- xs :gMonthDay
- xs :gDay
- xs :gMonth
- xs :hexBinary
- xs :base64Binary
- xs :anyURI
- xs :QName
- xs :NOTATION

Séquences versus node-sets

```
(//planet/mass, //planet/name)
```

Les expressions arithmétiques

`idiv` division entière
- soustraction (mettre les espaces autour de -)

Type

Dans `axe::type` :
Les même types que dans XPath 1 :
`* comment() node() processing-instruction() text() attribute()`

et les nouveaux :
`element() document-node()`

Opérations sur le séquences

L'union existait déjà dans XPath 1 : |

Maintenant : `union intersect except`

Les comparaisons - comparaisons de valeurs

`eq ne lt le qt ge`

à utiliser uniquement sur les valeurs atomiques
`$teperature lt 72`

Les comparaisons générales

Possible sur les séquences :
= < > <= >= !=
Compare les éléments de la première séquence aux éléments de la deuxième et retourne true s'il existe un couple qui satisfait la condition (par exemple lt pour $<$).

Comparaisons de noeuds

`is`

Exemple :
`//planet[name="Venus"] is //planet[days=116.75]`
donne `true` si c'est le même noeud.

Ordre de noeuds

`<<` et `>>`

`//planet[name="Venus"] << //planet[days=116.75]`
true si le noeud à gauche avant le noeud à droite.

Expression for

```
for variable in sequence return expression
```

```
for $variable in //planet return $variable/name
```

```
for $x in (3, 4) return $x + 1
```

Variables multiples :

```
for $x in (3, 4) ,
       $y in (7,8)
return $x + $y
```

Les expressions conditionnelles

```
if expression then then-expression else
else-expression
```

```
if (//planet[1]/mass > //planet[2]/mass)
    then //planet[1]
    else //planet[2]
```

Les expression quantifiées

```
some $planet in //planet satisfies $planet/name
```
expression vrai si au moins une planète possède un enfant `name`

```
some $planet in //planet satisfies $planet/@language
```
vrai si au moins une planète possède l'attribut `language`

```
some $planet in //planet satisfies ($planet/mass >
1.5)
```

```
every $planet in //planet satisfies $planet/name
```
Vrai si chaque planète possède un enfant `name`

```
every $planet in //planet satisfies ($planet/day ge
1)
```

Evaluer les expressions XPath

Quelques logiciels permettant d'évaluer les expression de XPath : **BaseX** - SGBD pour XML, `basex.org`. Les version gratuite pour Linux et Windows. Permet d'evaluer les expression XPath et XQuery. Possède interface graphique.

saxon `saxon.sourceforge.net` permet d'evaluer XPath et XQuery en ligne de commande. Il est possible d'installer l'interface graphique `kernow` :

`http://sourceforge.net/projects/kernowforsaxon/files/latest/download`

éditeur texte jEdit `www.jedit.org` écrit en java. Il faut installer les plugins XML, XQuery, XSLT (depuis le menu Plugins - installation automatique une fois les plugins sélectionnés), jEdit permet aussi faire les requêtes XQuery.

plugin XPath Checker pour firefox – `https://addons.mozilla.org/fr/firefox/addon/xpath-checker/`