Introduction à XQuery

Code: xml-query

Originaux

url: **http://tecfa.unige.ch/guides/tie/html/xml-query/xml-query.html**

url: **http://tecfa.unige.ch/guides/tie/pdf/files/xml-query.pdf**

Auteurs et version

- Daniel K. Schneider - Vivian Synteta

- Version: 0.2 (modifié le 8/3/06)

Prérequis

Module technique précédent: xml-dom

Module technique précédent: xml-tech

Module d'exercices précédant: xml-xslt (surtout Xpath !)

Autres modules

Module technique suppl.: xml-ser

Module technique suppl.: xml-xslt

Abstract

- Ceci est une toute première version, il reste bcp à améliorer et faire ....
- Donc attention: lecture à vos risques et périls

Objectifs

- Introduction à XQuery et son utilisation
- Server-side XQuery (à faire !!!)

1. Table des matières détaillée

2. Introduction

2.1 Principe

- XQuery est un langage de "programmation" puissant pour extraire des données XML
- Type de données: un seul "document" ou encore des collections sous forme de:
  - fichiers
  - bases de données XML
  - XML "en mémoire" (abres DOM)
- Permet de faire des requêtes selon la structure ou encore les contenus en se basant sur des expressions Xpath (version 2.0)
- Peut générer des nouveaux documents
  - autrement dit: on peut manipuler un résultat obtenu et y ajouter
- Ne définit pas les mises à jour (équivalent de update/insert de SQL), à venir ....

Résumé:

XQuery permet d'extraire des fragments XML, d'y effectuer des recherches et de générer des fragments XML.

Standard:

url: http://www.w3.org/TR/xquery/ **(pour les interrogations, stable)**

url: http://www.w3.org/TR/xqupdate/ **(pour les mises à jour, instable)**

2.2 Exemple de sensibilisation

Chercher tous les noeds avec le chemin //topic/title

(tous les éléments **<topic><title>xxxx </title>**</topic>, topic peut se trouver n'import où dans l'arbre)

```
for $t in //topic/title
   return $t
```

Chercher tous les noeds avec le chemin //topic/title dans "catalog.xml"

```
for $t in document("catalog.xml")//topic/title
   return $t
```

Idem, mais on cherche le fichier via http

```
for $t in document("http://tecfa.unige.ch/proj/seed/catalog/net/
catalog.xml")//topic/title
   return $t
```

Résultat: une collection d'itèmes trouvés

```
<title>TECFA Seed Catalog</title>
<title>Introduction</title>
<title>Conceptual and technical framework</title>
<title>The socio-constructivist  .... etc.
```

2.3 Contextes d'utilisation:

- Il faut un processeur XQuery, par exemple
  - un outil "standalone", par exemple le processuer XSLT/XQuery "saxon"
  - un outil intégré dans un "service Web", une base de données XML, etc.
  - une librairie intégre dans un logiciel (par ex. un éditeur XML)

Ensuite, même principe que pour les bases de données SQL

- En ligne de commande
- Avec un outil d'administration (Web ou en local)
- Avec une application Web (donc il faut programmer une interface)

3. XQuery de base

3.1 La notion d'expression de recherche

XQuery est bâti sur des éléments XPath, par exemple

1. position dans un arbre
   - ex: retourne tous les noeuds <card-propvalue> sous <c3mssoft-desc> ...

```
//c3msbrick//c3mssoft-desc//card-propvalue
```

2. opérations de comparaison
   - ex: retourne seulement le fragment pour une id particulière

```
//c3msbrick//c3mssoft[@id="epbl"]
```

3. functions
   - ex: qui retourne juste le contenu d'un noeud

```
//c3msbrick/title/text()
```

3.2 Les expressions FLWOR

- FLOWR = "For-Let-Where-Order-Return"
- rappelle l'idée du select-from-where-..-order-by de SQL

Voici un survol:

1. for = iteration sur une liste de fragments xml et
2. let = association du résultat d'une expression à une variable
3. where = condition de sélection
4. order = tri des résultats
5. return = expression à retourner

A. "for"

```
Syntaxe: for $variable in expression_recherche
   RETURN
```

"for" associe à chaque $variable une valeur (fragment XML) trouvé pour l'expression XQuery (voir page 7)
Exemple:

```
for $t in //topic/title
```

B. let

- "let" permet de d'assigner une valeur à une variable
- Voici une expression avec un for simple:

```
for $t in document("catalog09.xml")//c3msbrick//c3mssoft-desc//card-propvalue
return $t
```

- Voici une expressions équivalente au niveau du résultat retourné
  - pour chaque fragment $t trouvé on définit une variable $desc qui contient un sous-fragment.

```
for $t in document("catalog09.xml")//c3msbrick
let $desc := $t//c3mssoft-desc//card-propvalue
return $desc
```

- L'exemple suivant est plus utile, pour chaque élément $t trouvé dans la boucle "for" on compte le nombre de certains sous-éléments et on affiche les titres de t$ avec le compte:

```
for $t in document("catalog09.xml")//c3msbrick
let $n := count($t//c3mssoft)
return <result> {$t/title/text()} possède {$n} briques </result>
```

C. where

- permet de définir une condition de sélection
- une seule par requête (je pense ? - à vérifier)
- Dans l'exemple suivant on reprent l'exemple ci-dessus, mais on affiche seulement les cas où on est en présence d'au moins 2 <c3mssoft>

```
for $t in document("catalog09.xml")//c3msbrick
let $n := count($t//c3mssoft)
where ($n > 1)
return <result> {$t/title/text()} possède {$n} briques </result>
```

D. order

- permet de trier les résultats
- Voici un simple tri alphabétique des résultats:

```
for $t in //topic/title order by $t return $t
```

- Voici le même exemple avec un tri dans l'ordre croissant

```
for $t in document("catalog09.xml")//c3msbrick
let $n := count($t//c3mssoft)
where ($n > 1)
order by $n
return <result> {$t/title/text()} possède {$n} briques </result>
```

- Voici un exemple un peu plus informatif, on donne aussi les <titles> des <c3msbricks>:

```
for $t in document("catalog09.xml")//c3msbrick
let $brick_softs := $t//c3mssoft
let $n := count($brick_softs)
where ($n > 0)
order by $n
return <result>
      For {$t/title/text()} we found {$n} softwares:
         {$brick_softs//title}
      </result>
```

E. return

- construit l'expression à retourner à chaque iteration
- Attention: Chaque iteration doit retourner un seul fragment XML (pas une collection)

Juste:

```
for $t in document("catalog09.xml")//c3msbrick
let $n := count($t//c3mssoft)
return <result>
       {$t/title/text()} possède {$n} briques
       </result>
```

Faux:

```
for $t in document("catalog09.xml")//c3msbrick
let $n := count($t//c3mssoft)
return $t/title/text() possède $n briques
```

3.3 Construction de fragments XML

- Le résultat d'une requête devrait normalement se présenter sous une form appropriée pour être réutilisée ...

A. Version multi-fragment (collection)

- il faudrait encore la traiter pour laffichage

  L'expression suivante de base

```
for $t in //c3msbrick/title
   return $t
```

  retourne une liste (une collection)

```
1    <title class ="- topic/title " > TECFA Seed Catalog </ title >
2    <title class ="- topic/title " > Introduction </ title >
3    <title class ="- topic/title " > Conceptual and technical framework </
title >
4    <title class ="- topic/title " > The socio-constructivist approach </
title >
....
```

- Autrement dit, on reçoit une collection de fragments xml
- Il va falloir la traiter pour pouvoir l'afficher (normalement avec un programme)

B. Version un seul fragment

L'expression suivante:

```
<result>
 { for $t in //topic/title/text()
 return <titre>{$t}</titre> }
</result>
```

donne:

```
<result  >
<titre > TECFA Seed Catalog </ titre >
<titre > Introduction </ titre >
<titre > Conceptual and technical framework </ titre >
<titre > The socio-constructivist approach </ titre >
....
</result>
```

- notez qu'on a remplacé les balises "title" par "titre"
- toutes les les expression entre { ...} sont évaluées
- mais à l'intérieur des ces expression toutes les <balises> sont copiés "tels quels"

3.4 For dans for

- on peut imbriquer des boucles dans des boucles ...

```
<result>
<title>List of C3MSBricks and associated Software</title>
{ for $t in document("catalog09.xml")//c3msbrick
  let $brick_softs := $t//c3mssoft
  let $n := count($brick_softs)
  where ($n > 0)
  order by $n descending
  return
    <brick> Pour {$t/title/text()} on a les {$n} modules suivants:
       { for $soft in $brick_softs
         return <soft> {$soft//title/text()}</soft>
       }
    </brick>
}
</result>
```

- Chaque expression FLWOR ou variable se trouve dans des { ... }
- Le "return" de la grande boucle contient une boucle qui décortique encore
  - $brick_softs contient une collection de $softs donc on sort les titres
- On trie dans l'ordre descendant

Voici le résultat:

```
<result  >
<title > List of C3MSBricks and associated Software </ title >
<brick > Gallery possède les 5 briques suivants:
<soft > PhotoShare </ soft >
<soft > Photoshare combined with PageSetter </ soft >
<soft > My_eGallery </ soft >
<soft > Coppermine </ soft >
<soft > Gallery </ soft >
</ brick >
<brick > User statistics possède les 5 briques suivants:
<soft > commArt </ soft >
<soft > pncUserPoints </ soft >
<soft > pncSimpleStats </ soft >
<soft > Statistics module </ soft >
<soft > NS-User_Points </ soft >
</ brick >
.....
</result>
```

4. Fonctions définies par l'utilisateur

- à suivre :)

5. XQuery avec Saxon

- Permet d'apprendre XQuery sans avec accès à un serveur ou éditeur XML

On conseille fabriquer un fichier *.bat ou équivalent pour Unix:

- exemple: Un fichier "xquery.bat" contiendra le contenu suivant:

```
rem  set the variable JAVA to the place where the java command is located rem
not the case if you just have one that is correctly installed), for example
rem    set JAVA="c:\Program Files\java\jdk1.5.0\bin\java"

set JAVA=java
rem    modify the following line to set the classpath (saxon.jar)
set CP=-cp c:\soft\saxon861b\saxon8.jar

%JAVA% %CP% net.sf.saxon.Query %1 %2 %3 %4 %5 %6 %7 %7
```

- Voir aussi: **http://tecfa.unige.ch/guides/tie/html/xml-xslt/xml-xslt.html**

Utilisation:

(1) Afficher toutes les options:

```
xquery
```

(2) Exécuter un fichier xquery (donc le xquery sait dans quel fichier chercher):

```
xquery fichier.xquery
```

(3) Exécuter un fichier xquery avec un fichier xml:

```
xquery -s fichier.xml fichier.xquery
```

(4) Exécuter un fichier xquery avec un fichier xml et mettre le résultat dans un fichier:

```
xquery -s fichier.xml -o output.xml fichier.xquery
```

6. XQuery avec eXist

- eXist est une base de donnée XML (et qui vient aussi à option avec un portail)
- Pour l'utiliser, il faut avoir un login et les droits d'écrire ou de lire des collections
- Une collection contient des fichiers XML (et qui normalement représentent une application, un domaine, etc.)
- Il existe plusieurs API pour accéder à eXist

eXist à TECFA

url: **http://tecfax.unige.ch:8088/exist/index.xml**

Le portail

- permet de faire certaines opérations de gestion
- contient des exemples et de la documentation

6.1 Le client Java

Installation et lancement

- soit il faut l'installer chez vous (c.f. le site de eXist)
- soit vous le lancez par "Java WebStart" depuis le portail (recommandé !)

Fonctionalités

- Gestion des utilisateurs et des collections
- Exécution de XQuery

6.2 L' API PHP/xml-rpc

url: **http://tecfa.unige.ch/guides/xml/examples/xquery/exist_phpapi/**

(cf. les exemples)