Comment filtrer les doublons avec XSLT 1.0. Méthodes "classique" et "muenchian"

Eric van der Vlist, Dyomedea (vdv@dyomedea.com).
jeudi 31 mars 2005

Table des matières

Question

Dans le document suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book>
        <isbn>0836217462</isbn>
        <title>Being a Dog Is a Full-Time Job</title>
        <author>
            <name>Charles M. Schulz</name>
            <nickName>SPARKY</nickName>
            <born>November 26, 1922</born>
            <dead>February 12, 2000</dead>
        </author>
        <character>
            <name>Peppermint Patty</name>
            <since>Aug. 22, 1966</since>
            <qualification>bold, brash and tomboyish</qualificati
on>
        </character>
        <character>
            <name>Snoopy</name>
            <since>October 4, 1950</since>
            <qualification>extroverted beagle</qualification>
        </character>
        <character>
            <name>Schroeder</name>
            <since>May 30, 1951</since>
            <qualification>brought classical music to the Peanuts
 strip</qualification>
        </character>
        <character>
            <name>Lucy</name>
            <since>March 3, 1952</since>
            <qualification>bossy, crabby and selfish</qualificati
on>
        </character>
    </book>
    <book>
        <isbn>0805033106</isbn>
        <title>Peanuts Every Sunday </title>
        <author>
```

```
            <name>Charles M. Schulz</name>
            <nickName>SPARKY</nickName>
            <born>November 26, 1922</born>
            <dead>February 12, 2000</dead>
        </author>
        <character>
            <name>Sally Brown</name>
            <since>Aug, 22, 1960</since>
            <qualification>always looks for the easy way out</qua
lification>
        </character>
        <character>
            <name>Snoopy</name>
            <since>October 4, 1950</since>
            <qualification>extroverted beagle</qualification>
        </character>
        <character>
            <name>Linus</name>
            <since>Sept. 19, 1952</since>
            <qualification>the intellectual of the gang</qualific
ation>
        </character>
        <character>
            <name>Lucy</name>
            <since>March 3, 1952</since>
            <qualification>bossy, crabby and selfish</qualificati
on>
        </character>
    </book>
</library>
```

Je souhaite afficher la liste de tous les personnages (éléments "character") en filtrant les doublons (chaque personnage ne doit apparaître qu'une seule fois même s'il figuer dans plusieurs livres.

Méthode "classique"

Pour sélectionner tous les personnages, nous écririons la requête **XPath** suivante :

```
/library/book/character
```

Cette méthode n'effectue aucun filtrage et elle sélectionne toutes les occurrences de l'élément "character" lorsqu'il est inclus dans un élément "author" lui-même inclus dans l'élément racine "library".

Les éléments "character" figurant dans plusieurs éléments "book", tels que l'élément :

```
        <character>
            <name>Snoopy</name>
            <since>October 4, 1950</since>
            <qualification>extroverted beagle</qualification>
        </character>
```

seront donc répétés dans l'ensemble de noeuds sélectionnés par cette première expression **XPath**.

Pour filtrer ces doublons, nous allons devoir ajouter une condition sur l'élément "character".

Cette condition peut s'exprimer en français en disant que nous souhaitons que l'on n'ait pas déjà rencontré un personnage portant le même nom.

Pour tester si nous avons déjà rencontré un personnage ayant le même nom que le personnage courant, nous écririons :

```
preceding::character/name = name
```

Ce test utilise l'opérateur **XPath** "=" pour comparer deux ensembles de noeuds (l'ensemble de tous les sous éléments "name" des éléments "character" précédents et l'ensemble composé de l'élément "name" de l'élément "character" courant. Pour **XPath**, ce test est vrai si les deux ensembles ont au moins un noeud ayant une valeur commune.

Pour tester la condition contraire, nous allons utiliser la fonction "not()" et notre sélection **XPath** peut s'écrire :

```
/library/book/character[not( preceding::character/name = name )]
```

Nous pouvons l'utiliser dans les attributs "select" ou "match" d'une instruction **XSLT**, par exemple :

```
<xsl:apply-templates select="/library/book/character[not( precedi
ng::character/name = name )]"/>
```

ou encore :

```
<xsl:variable name="distinct-characters" select="/library/book/ch
aracter[not( preceding::character/name = name )]"/>
```

Méthode "muenchian"

Contexte

La méthode que nous venons d'exposer demande au processeur **XSLT** de rechercher de manière répétitive dans tous les éléments "character" précédant s'il y en a un de même nom que l'élément courant.

Cette recherche peut devenir longue lorsque la taille du document est importante.

La méthode connue sous le nom de méthode "**muenchian**" a été proposée pour optimiser le filtrage des doublons par Steve Muench, "Developer, Product Mgr, Java/XML Evangelist" chez **Oracle**.

Clés XSLT

Elle repose sur l'utilisation des clés **XSLT** (xsl:key).

Les clés **XSLT** permettent l'accès rapide à des ensembles de noeuds dont on défini les identifiants.

Dans notre cas, nous pouvons définir une clé sur les éléments "character" en utilisant leur sous élément "name" :

```
<xsl:key name="characters" match="character" use="name"/>
```

Cette déclaration définit une clé dont le nom est "characters", qui porte sur tous les éléments "character" du document et qui utilise comme identifiant la valeur de leur sous élément "name".

Pour sélectionner la liste des éléments "character" ayant pour nom "Snoopy", nous pouvons maintenant écrire l'expression **XPath**/**XSLT** :

```
key('characters', 'Snoopy')
```

Cet accès par clé est optimisé par les processeurs **XSLT** qui construisent une table de correspondance à la première utilisation de la clé. Cette première utilisation est donc plus lente qu'un de nos balayages répétitifs mais les utilisations suivantes utilisent la table de correspondance et elles sont beaucoup plus rapides.

Filtrage des doublons à l'aide d'une clé XSLT

Intéressant me direz-vous, mais quel est le rapport avec le filtrage des doublons?

On va pouvoir utiliser cette clé en spécifiant que l'on veut sélectionner les éléments "character" qui sont les premiers de la liste des éléments qui ont le même nom.

En **XPath**, l'élément courant s'exprime par "." et la premier élément de la liste des éléments qui ont le même nom est "key('characters', name)[1]".

Nous avons deux possibilités pour tester que ces deux expressions correspondent au même noeud et ces deux possibilités vont nous fournir deux variantes de la méthode "**muenchian**".

La première possibilité est de tester si les identifiants obtenus par la fonction **XSLT** "generate-id()" sont égaux. Nous écrirons alors :

```
/library/book/character[generate-id() = generate-id(key('characte
rs', name)[1])]
```

La deuxième possibilité est de tester que l'ensemble des noeuds constitué par le noeud courant et le premier élément de la liste des éléments de même nom ne comporte qu'un seul élément :

```
/library/book/character[ count( . | key('characters', name)[1]) =
 1]
```

Les deux variantes donnent strictement le même résultat et sont équivalentes en terme de performance.

Dans les deux cas, ces expressions XPath/XSLT peuvent être utilisées dans une instruction XSLT telle que :

```
<xsl:apply-templates select="/library/book/character[generate-id(
) = generate-id(key('characters', name)[1])]"/>
```

ou encore :

```
<xsl:variable name="distinct-characters" select="/library/book/ch
aracter[ count( . | key('characters', name)[1]) = 1]"/>
```

Avantages et inconvénients de la méthode "**muenchian**"

- Si la méthode **muenchian** est plus rapide que le méthode classique lorsque le nombre d'éléments manipulés est important, elle est également plus gourmande

en mémoire puisque le processeur **XSLT** construit une table de correspondance dont la taille sera fonction du nombre d'éléments sélectionnés dans la clé.
- Elle nécessite également un processeur **XSLT** supportant les clés, ce qui n'est pas le cas des processeurs **XSLT** anciens tels que le processeur **XT** et les version **MSXML** antérieures à 2000.
- Enfin, les clés ne portant que sur le document courant, elle ne peut pas être appliquée telle quelle lorsque l'on consolide des informations provenant de plusieurs documents (en utilisant la fonction **XSLT** "document()").

Et en XPath/XSLT 2.0?

Les versions de travail actuelles (publiées le 11 février 2005) de **XPath** 2.0 et **XSLT** 2.0 intègrent toutes deux des fonctionnalités permettant de faire ce type de traitement :

- La fonction **XPath** 2.0 "distinct-values" permet de faire un filtrage analogue à celui que nous venons de voir.
- Les fonctionnalités de "groupage" (grouping) de **XSLT** 2.0 permettent d'implémenter des regroupements plus complexes.

La méthode "**muenchian**" risque donc fort de tomber en désuétude.