Rapport TP XML

MONDIAL

LOISON Laureen – GUERON Antoine
UNIVERSITÉ DE NANTES | MASTER 1 MIAGE 2016/2017

Introduction

Dans le cadre de notre formation en Master MIAGE à l'université de NANTES, et plus particulièrement dans le module d'ingénierie XML, nous avons eu à traiter un fichier XML en utilisant des méthodes différentes. Ce fichier répertorie des informations sur des pays, des zones d'eau et des villes sur les différents continents du globe. Ce fichier constitue une base de données au format XML nommée mondial.
A partir de cette base, nous devons reporter dans un fichier de sortie XML les pays avec leur nom et le nom de leur capitale qui sont des pays faisant partie du continent asiatique en partie mais pas à 100%. Voici la sortie attendue :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE liste-pays SYSTEM "liste-pays.dtd">
<liste-pays>
   <pays nom="Russia" capitale="Moscow" proportion-asie="80" proportion-autres="20"/>
   <pays nom="Turkey" capitale="Ankara" proportion-asie="95" proportion-autres="5"/>
   <pays nom="Egypt" capitale="Cairo" proportion-asie="10" proportion-autres="90"/>
   <pays nom="Indonesia" capitale="Jakarta" proportion-asie="80" proportion-autres="20"/>
</liste-pays>
```

Le fichier attendu devra répondre à la DTD suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT liste-pays (pays)*>
<!ELEMENT pays EMPTY>
<!ATTLIST pays
            nom CDATA #REQUIRED
            capitale CDATA #REQUIRED
            proportion-asie CDATA #REQUIRED
            proportion-autres CDATA #REQUIRED>
```

Le but de cet exercice est de nous faire manipuler les différentes méthodes que nous avons pu voir en cours. Nous pourrons ensuite nous intéresser plus en détail sur les différentes méthodes pour les comparer sur différents points tels que la performance, l'adéquation au traitement, la clarté du code et la maintenabilité.

Programme XSLT

A- Choix de d'implémentation

Après avoir vu la partie théorique de XSLT en cours, nous passons à l'application pratique.
XSLT est un langage qui permet de définir des règles à appliquer sur un ensemble de noeud contenu dans un document XML.
Dans notre programme XSLT, nous avons 2 templates qui s'appliquent sur le document mondial.xml. La premier template s'applique sur le noeud racine afin de contourner la règle par défaut. Et le second template s'applique sur les noeuds "country".
C'est dans le premier template que nous appliquons une règle qui permet de ne traiter que les noeud country faisant parti du continent Asiatique mais pas à 100%. Voici la règle :

```
<liste-pays>
    <xsl:apply-templates select="./mondial/country[encompassed[@continent eq 'asia' and number(@percentage) lt 100]]"/>
</liste-pays>
```

Cette règle est valable pour la version 2.0 de XSLT.
Cependant, pour pouvoir exécuter le programme XSLT avec php (pour l'exécuter de la même façon que les autres programmes), nous utilisions la librairies XSLTProcessor qui ne supporte pas la version 2.0 de XSLT. Nous avons donc dû revoir notre programme pour l'adapter à la version 1.0 de XSLT. Voici donc la règle en version 1.0 :

```
<liste-pays>
    <xsl:apply-templates select="./mondial/country[encompassed[@continent = 'asia' and number(@percentage) &lt; 100]]"/>
</liste-pays>
```

Une fois dans le template country, nous allons récupérer les informations dont nous avons besoins pour le résultat.

B- Analyse de l'implémentation

XSLT présente l'avantage de produire un code assez réduit 26 lignes pour la ligne 1.0 et 21 lignes pour la version 2.0. De plus, le code est clair et facilement maintenable car il se présente en plusieurs petit paragraphes pour chaque traitement que nous voulons réaliser.

DOM

A- Choix de l'implémentation

Ce programme est réalisé en DOM sans utiliser XPATH. Le programme DOM va stocker le fichier mondial.xml en mémoire et le structurer grâce à la DTD qui est renseignée. Dans un premier temps nous récupérons l'ensemble des pays grâce à l'instruction *getElementByTagName("country")*. Pour contourner l'utilisation de cette instruction, nous aurions pu utiliser un *getChildNodes()* et ensuite tester si c'était un *"country"* . Une fois dans les balises *country*, on va récupérer les informations dont nous avons besoin pour construire le fichier DOM final.

B- Analyse de l'implémentation

Ce code fait 68 lignes, il est donc plus long que le XSLT. Notre code possède deux boucles imbriquées ce qui le rend potentiellement plus difficile à comprendre. De plus, le rendu final est un document DOM, donc il faut pouvoir différencier le document DOM final et le document DOM qui représente le fichier mondial.xml. Le document final DOM est ensuite enregistré au format XML. Le code en reste cependant facilement maintenable.

DOM avec XPATH

A- Choix de l'implémentation

Ce programme est réalisé en DOM en utilisant XPATH. Le programme est un compromis entre le programme DOM sans XPATH et le XSLT. En effet, nous utilisons une requête qui est basée sur les mêmes critères que dans le programme XSLT. Voici la requête :

```
$query = "/mondial/country[./encompassed/@continent = 'asia' and ./encompassed/@percentage < 100]";
```

Cette requête permet de sélectionner les pays qui sont sur le continent asiatique mais pas totalement.

B- Analyse de l'implémentation

Dans cette version du code, il y a plus qu'une seule boucle. On peut donc penser que le code est plus simple à comprendre. Cependant, il utilise des requêtes pour aller chercher les informations, ce qui implique de connaître la syntaxe pour comprendre le but des requêtes. Le code reste lisible, et sa maintenabilité dépendra des connaissances qu'on dispose pour réaliser les requêtes XPATH.

SAX

A- Choix de l'implémentation

La programmation SAX est basée sur la lecture du document et des balises. Le programme comporte quatres fonctions principales : startDocument(), endDocument() qui permettent de réaliser des traitements au début et à la fin du document, et startElement() et endElement() qui vont permettre de réaliser des traitements à la lecture de balises ouvrantes et à la lecture des balises fermantes. De plus, pour réaliser les traitements, nous utilisons des flags (boolean) qu'on positionne à vrai lorsque les critères sont en accord avec ce que nous recherchons.

B- Analyse de l'implémentation

Le programme SAX tient sur 97 lignes. Il est plus long, donc pas forcément plus facile à lire. L'utilisation des flags, ainsi que de plusieurs attributs pour construire le document DOM final le rend plus difficile à faire évoluer. Cependant, le programme est structuré grâce aux 4 fonctions et à l'utilisation des structure conditionnelles if.

XMLReader / XMLWriter VS JDOM

Afin d'avoir de nouveaux éléments de comparaison, nous avons réalisé l'exercice avec deux autres méthodes : XMLReader/Writer (PHP) et JDOM (Java). Nous avons choisi de ne pas traiter le troisième exemple avec SimpleXML par manque de temps. Nous avons donc pu comparer les deux programmes l'un avec l'autre ainsi qu'avec les autres méthodes décrites dans la première partie du rapport. Notre analyse est la suivante :

XMLReader permet de lire le fichier de données XML puis de rechercher les objets qui nous sont utiles grâce à leur type et leur nom et leurs potentiels attributs et leurs valeurs. Une fois les objets que l'on cherche trouvés, on tag les objets nécessaires pour nous souvenir de notre emplacement dans l'architecture du fichier et on sauvegarde les valeurs souhaitées dans des variables qui nous servirons pour l'écriture avec XMLWriter par la suite. Une fois que nous avons stocké tous les objets nécessaires dans des variables et que nous sommes au bon emplacement, au fait appel à XMLWriter pour écrire le résultat dans le nouveau fichier XML. On créé les nouveaux éléments et attributs associés que l'on écrit dans le fichier XML. Enfin, on remet les tags à false pour rechercher les autres objets souhaités. On sauve et ferme ensuite le nouveau fichier XML.

XMLReader et XMLWriter sont utilisés dans un programme PHP. XMLReader va donc faire une lecture de tous les éléments du document et faire une comparaison avec nos expressions pour savoir si l'élément en cours satisfait telle ou telle expression. XMLReader n'utilise pas la mise en cache et réalise une lecture en avant du document c'est-à-dire que lorsqu'un élément a été lu, le lecteur ne peut pas revenir en arrière dessus. L'implémentation de cette solution se rapproche donc de la solution en PHP avec SAX.

D'un autre côté nous avons JDOM qui s'utilise dans un langage Java. JDOM propose aussi une intégration de SAX, DOM, XSLT et XPath. En effet, JDOM utilise SAX ou DOM principalement pour parser les documents XML. Ici nous avons choisi d'utiliser SAX. SAXBuilder va permettre de lire le fichier et de stocker le document entier dans une variable.

On va ensuite faire appel à la méthode permettant de récupérer les éléments souhaités avec les données du fichier (stockées dans une variable) comme base de travail pour la recherche en Java. Les méthodes de JDOM permettent notamment de beaucoup travailler sur les Collections. C'est pourquoi, nous récupérons tout d'abord la liste de tous les pays, sur lesquels nous allons ensuite boucler pour récupérer la liste des encompassed du pays en cours et sauvegarder les pays qui ont un pourcentage de population asiatique inférieure à 100. Ensuite, on récupère la liste des villes du pays et des villes des provinces du pays pour trouver la capitale du pays avec un parcours de ces deux listes. Une fois toutes les informations récupérées par imbrication d'itération successive, nous écrivons sur la console le nouvel élément XML en conformité avec les normes XML.

On peut voir que cette méthode est assez lourde car elle implique une imbrication de boucle importante qui coûte en terme de complexité et de traitement des algorithmes.

Si l'on compare JDOM et DOM vu précédemment, on observe que JDOM propose un programme plus long et plus lourd en terme d'imbrication de boucle. D'autant plus, si l'on compare avec la version DOM XPath qui est une version très light par rapport à la recherche de JDOM.

Concernant la comparaison entre JDOM et XMLReader, on peut dire que XMLReader possède un traitement puisque le fichier n'est lu qu'une seule fois du début à la fin sans retour en arrière contrairement à JDOM qui fonctionne par itération de liste et qui va potentiellement lire plusieurs fois les mêmes éléments pour récupérer ces listes. Cependant, JDOM possède la faculté de récupérer une liste d'éléments ou un élément quelconque à n'importe quel moment de l'exécution du programme puisque l'intégralité des données du document sont stockées dans une variable accessible n'importe où. Chacune des deux méthodes possèdent des avantages et des inconvénients. Concernant les performances ainsi que la maintenance de ces deux codes, il nous semble que l'utilisation de XMLReader et XMLWriter peut être plus intéressante même si JDOM possède peut-être une API plus large. De plus, en terme d'adéquation au traitement, XMLReader semble plus en corrélation avec l'architecture XML.

Problèmes rencontrés

Nous avons principalement rencontré deux problèmes d'ordre technique plus particulièrement. La première difficulté fut de s'adapter à chacune des syntaxes de chaque solution ainsi qu'à leur fonctionnement. Notamment pour les solutions qui nous était inconnue, il nous a fallu prendre le temps de lire la documentation de chacune d'elle en fonction des méthodes qui nous étaient utiles. Une autre difficulté était de bien organiser notre code et de bien gérer les tags, leur initialisation comme leur remise à zéro afin que le code soit bien exécuté.

Analyse des performances

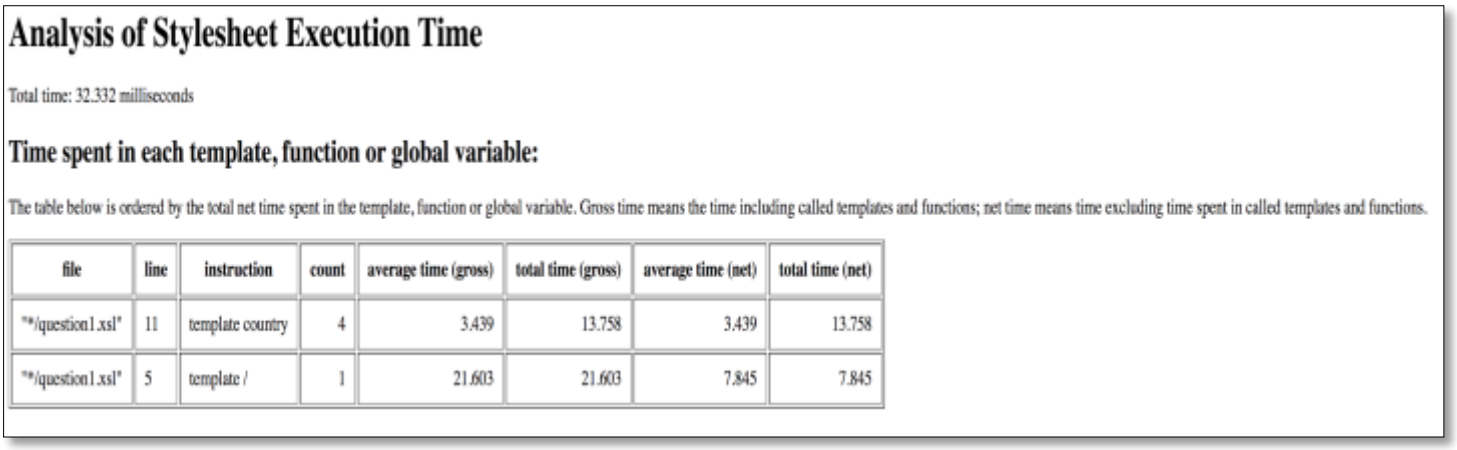
Afin d'analyser les performances de chacune de nos solutions, nous avons utilisé la fonction time(). Cette fonction nous permet d'obtenir un tableau de tous les temps d'exécution de chaque solution. Nous pouvons donc réaliser une comparaison de toutes les solutions suivant leur temps d'exécution mais également suivant le nombre de lignes de codes ou le nombre d'instruction réalisée.

Comparaisons multicritères

Après avoir présenté les différentes versions des programmes que nous avons réalisés, nous pouvons exercer une comparaison sur plusieurs critères tel que la performance, l'adéquation au traitement, de clarté du code et de maintenance.

Pour comparer les différents temps d'exécution, nous allons utiliser la fonction time. Nous utiliserons le résultat real qui est le temps total de l'exécution de la commande.

Cependant, avec la fonction time, l'environnement d'exécution à une répercussion sur le temps total d'exécution. En effet, l'exécution du XSLT en php met 0,230s alors que l'exécution avec java met 1,979s. C'est probablement dû au lancement de la JVM java pour l'exécution de la librairie Saxon. Cependant, avec l'option -TP, nous avons pu retrouver les informations répertoriées sur cette page :



Analysis of Stylesheet Execution Time

Total time: 32.332 milliseconds

Time spent in each template, function or global variable:

The table below is ordered by the total net time spent in the template, function or global variable. Gross time means the time including called templates and functions; net time means time excluding time spent in called templates and functions.

file	line	instruction	count	average time (gross)	total time (gross)	average time (net)	total time (net)
"*/question1.xsl"	11	template country	4	3.439	13.758	3.439	13.758
"*/question1.xsl"	5	template /	1	21.603	21.603	7.845	7.845

	Temps (en secondes)	Adéquation au traitement	Clarté du code	Maintenance
XSLT (php)	0.213s	75%	80%	80%
XSLT (java)	0.323s	75%	80%	80%
DOM sans XPATH (php)	0.348s	75%	70%	65%
DOM avec XPATH (php)	0.179s	75%	70%	75%
SAX (php)	0.204s	60%	60%	60%
XMLReader / XMLWriter	0.177s	75%	70%	60%
JDOM (java)	0.738s	75%	70%	75%

Conclusion

Nous avons ainsi pu expérimenter sur un fichier comportant une quantité non négligeable les différentes techniques pour traiter un fichier XML. Nous avons pu conclure sur différents points tel que la performance des traitements est liée à l'environnement dans lequel le programme s'exécute ou encore l'efficacité des requêtes XPATH. En effet, nous avons pu comparer PHP et java et nous constatons que PHP est plus rapide. Il serait intéressant de réaliser ces tests dans d'autre langues pour pouvoir voire les différences.
De plus, nous pensions que certaines méthodes de traitements seraient plus efficaces tel que le SAX comparé au DOM.
Mais ce qui nous a le plus surpris c'est qu'un programme est plus rapide à l'exécution lorsqu'il est couplé à des requêtes XPATH.