heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

IFC – LABORATOIRE N°3

XML PAR L'EXEMPLE

Professeur

M. Markus JATON

Assistant

M. Grégory RUCH

<?xml version="1.0"?>

Auteurs

Steve LIENHARD
Arnaud BURKHALTER

Temps à disposition : 2 semaines
Date de début: 10 mars 2010
Date de fin : 24 mars 2010

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Table des matières

Introduction

Les projets conséquents d'aujourd'hui utilisent de plus en plus la notion de « définition de donnée » qu'offre le langage XML. Ce dernier permet une plus grande liberté de nommage des balises html par exemple, ce qui est intéressant du fait que chaque domaine (Médecine, Bibliothèque, etc…) a ses demandes au niveau des noms de balises. Cette définition de donnée permet aussi une plus grande séparation entre le travail de programmation et le design de l'application WEB. Ce laboratoire va également nous faire découvrir les langages propres au Web, ainsi que ceux de mise en forme. Nous verrons comment écrire un simple fichier html et également comment le mettre en forme à l'aide d'une feuille de style. Nous découvrirons aussi comment appliquer un style à un document XML à l'aide d'un document XSL. En fin, se posera le problème de l'analyse syntaxique d'un document que nous résoudrons à l'aide des différents API offerts par Java pour implémenter différents parseurs.

Questions élémentaires

Insuffisances de HTML

Dès lors que l'on souhaite effectuer du commerce par internet, les différents problèmes suivants se posent :

- Sessions non présentes du fait que http est un protocole sans états. Chaque transaction est donc traitée indépendamment des autres.
- Dynamisme
- Requêtes / BDD

Pour répondre à la question posée, le fait d'apporter des modifications au langage lui-même implique la modification/adaptation des pages présentes sur le web. Il n'est donc pas envisageable d'effectuer des modifications importantes au langage sans qu'il n'y ait de répercussions sur tout le monde du web.

Présentation

Le moyen à disposition pour modifier le style des balises HTML est l'utilisation d'une feuille de style CSS, qui permet la personnalisation des balises.

Démarches à effectuer :

- Créer une feuille de style comprenant la ligne suivante :

```
H1 {font-style: Arial color :red}
```

- Dans le document HTML, insérer une référence à notre feuille de style.

```
<link rel= stylesheet type='text/css' href= 'style.css'>
```

Typage du contenu

Si cela n'était pas prévu d'emblée, il n'y a pas le moyen de changer le style de ces références puisqu'elles ne font partie d'aucune balise définie.

Par contre si des balises de type nom personne ont été prévues initialement, il est alors possible de modifier simplement le style à l'aide d'une feuille css.

XML

Le document que nous avons réalisé se présente de la manière suivante :

```
<?xml version="1.0"?>
<?xml-stylesheet href="style.css" type="text/css"?>
<!DOCTYPE liste SYSTEM 'personne.dtd'>

<liste>
	<personne>
		<nom>Joh</nom>
		<prenom>LuiTi</prenom>
		<email>joh.luiti@heig-vd.ch</email>
		<telephone>
			<fixe>
				<indicatif>024</indicatif>
				<numero>8224343</numero>
			</fixe>
			<mobile>
				<indicatif>078</indicatif>
				<numero>4453423</numero>
			</mobile>
		</telephone>
	</personne>
</liste>
```

Il est intéressant d'observer que dès l'instant qu'une erreur de syntaxe est présente dans le fichier xml, le browser nous indique que le document est incorrect, ainsi que l'erreur présente.

Par exemple, si l'on fait une erreur dans les balises <nom>, comme nous pouvons le voir ci-dessous, l'erreur nous sera indiquée de manière assez précise par le browser.

Message présent dans IE 9

Notion de DTD

DTD simple

Nous avons commencé par créer un fichier DTD qui va nous permettre par la suite de valider notre fichier xml.

Notre DTD se présente de la manière suivante :

```
<!ELEMENT liste (personne*)>
<!ELEMENT personne (nom,prenom,email,telephone)>
<!ELEMENT telephone (fixe,mobile)>
<!ELEMENT fixe (indicatif,numero)>
<!ELEMENT mobile (indicatif,numero)>

<!ELEMENT nom (#PCDATA) >
<!ELEMENT prenom (#PCDATA) >
<!ELEMENT email (#PCDATA) >
<!ELEMENT indicatif (#PCDATA) >
<!ELEMENT numero (#PCDATA) >
```

Une fois le fichier Echo10.java compilé, nous avons pu valider notre document XML à l'aide notre document DTD fraichement créé.

```
START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
PROCESS: <?xml-stylesheet href="style.css" type="text/css"?>
PROCESS: <?xml-stylesheet href="personne.xsl" type="text/xsl"?>
    ELEMENT: <liste>
        ELEMENT: <personne>
            ELEMENT: <nom>
            CHARS:   Joh
            END_ELM: </>
            ELEMENT: <prenom>
            CHARS:   LuiTi
            END_ELM: </>
            ELEMENT: <email>
            CHARS:   joh.luiti@heig-vd.ch
            END_ELM: </>
            ELEMENT: <telephone>
                ELEMENT: <fixe>
                    ELEMENT: <indicatif>
                    CHARS:   024
                    END_ELM: </>
                    ELEMENT: <numero>
                    CHARS:   8224343
                    END_ELM: </>
                END_ELM: </>
                ELEMENT: <mobile>
                    ELEMENT: <indicatif>
                    CHARS:   078
                    END_ELM: </>
                    ELEMENT: <numero>
                    CHARS:   4453423
                    END_ELM: </>
                END_ELM: </>
            END_ELM: </>
        END_ELM: </>
    END_ELM: </>
END DOCUMENT
```

Omission de la DTD

Si la validation ne passe pas, il est indiqué '*Document root element 'personne', must match DOCTYPE root 'null'*'. Chose logique étant donné qu'il parait normal que l'on indique au fichier XML qu'un document DTD lui est lié.

Sans cette ligne rien n'indique au fichier XML qu'il doit respecter la syntaxe d'une quelconque DTD.

Erreur sur des éléments du document XML

Si l'on oublie d'introduire un élément essentiel au document, tel que l'email dans notre exemple, la validation ne passe pas. Il nous est indiqué que le fichier devrait répondre à une certaine syntaxe, se qui n'est pas le cas actuellement. Cependant nous pouvons nous apercevoir qu'il est tout de même possible d'afficher le fichier XML dans un navigateur. L'affichage se fera de manière normale, jusqu'à que l'erreur soit détectée comme le montre la capture suivante :

Nous pouvons donc en déduire que le rôle d'une DTD est totalement structurelle et ne joue aucun rôle dans l'affichage d'un fichier XML. Cela permet simplement de donner un model de champs que vont devoir contenir les fichiers XML utilisant le fichier DTD.

Feuille de style CSS

La feuille de style que nous avons créée se présente de la manière suivante :

```
prenom
{
	font-family: Times New Roman
	font-size: 24px;
}

nom
{
	font-family: Times New Roman
	font-size: 24px;
	font-weight: bold;
}

email
{
	color: #FFD300;
	font-family: Courier;
	font-size: 18px;
	font-weight: bold;
}
```

Les feuilles de style CSS sont limitées dans le sens que l'on ne peut pas faire de retour à la ligne à l'aide de celles-ci, cependant elles permettent un enrichissement considérable de l'aspect d'un document HTML ou XML.

Avantages :

- Gestion centralisée de la présentation du site
- Mise à jour et changement éventuel du design simplifié
- Un même document peut posséder plusieurs styles différents

Inconvénients :

- Les feuilles de style sont étroitement liées à la structure du document HTML.
- La norme CSS étant relativement récente, les vieux navigateurs peuvent ne pas la supporter.

Transformation XSL

Mise en place du code XSL

Le fichier que nous avons réalisé se présente de la manière suivante :

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

	<xsl:output
		method="html"
		encoding="UTF-8"
	/>

	<xsl:template match="/">
		<xsl:for-each select="liste/personne">
			<xsl:choose>
				<xsl:when test="telephone/mobile/indicatif &#61; 076">
					<span style="color:red;">
						Nom: <xsl:value-of select="nom" /> <BR/>
						Prenom: <xsl:value-of select="prenom" />
					</span>
				</xsl:when>
				<xsl:when test="telephone/mobile/indicatif &#61; 078">
					<span style="color:orange">
						Nom: <xsl:value-of select="nom" /> <BR/>
						Prenom: <xsl:value-of select="prenom" />
					</span>
				</xsl:when>
				<xsl:when test="telephone/mobile/indicatif &#61; 079">
					<span style="color:blue">
						Nom: <xsl:value-of select="nom" /> <BR/>
						Prenom: <xsl:value-of select="prenom" />
					</span>
				</xsl:when>

				<xsl:otherwise>
					<span style="color:black">
						Nom:<xsl:apply-templates select="nom" /><BR/>
						Prenom:<xsl:apply-templates select="prenom" />
					</span>
				</xsl:otherwise>
			</xsl:choose>
			<BR/>
		</xsl:for-each>
	</xsl:template>

</xsl:stylesheet>
```

Le slash dans la balise <xsl:template match="/"> indique qu'il faut se placer à la racine de l'arborescence, puis la balise imbriquée <xsl:for-each select="liste/personne"> signifie que pour chaque personne, il faudra appliquer se qui suit. Donc pour chaque personne, on va choisir l'un des critères qui suivent la balise. Cette architecture pourrait être comparée à une instruction *case* en Java ou *Switch* en C.

Les balises <xsl:when test="telephone/mobile/indicatif = …"> permettent de comparer l'indicatif des numéros (= permet de tester l'égalité).

Définition d'une feuille de style

Une fois notre feuille de style (XSL) créée, nous pouvons l'appliquéE à notre fichier XML à l'aide de la commande suivante :

```
Java –classpath xalan.jar org.apache.xalan.xslt.Process –IN personne.xml –XSL personne.xsl –OUT personne.html
```

Toutefois avant d'exécuter cette commande, il faut bien s'assurer que tous les documents utiles, soit les fichiers XML, XSL ainsi que xalan.jar, se situent dans le répertoire courant.

Pour vérifier l'exécution correcte de la commande, nous pouvons ouvrir le fichier HTML ainsi créé et en vérifier le contenu.

Nom: Joh
Prenom: LuiTi

Nom: Roger
Prenom: Federer

Association fixe du fichier XML et du fichier XSL

Afin d'associer de manière permanente notre fichier XML avec la feuille de style XSL, nous avons introduit la balise suivante à l'intérieur du fichier XML :

```
<?xml-stylesheet href="personne.xsl" type="text/xsl"?>
```

Cela nous offre le même résultat que celui obtenu dans le point précédent à l'aide de xalan.jar, et indique donc au document XML qu'il doit effectuer sa mise en page selon la feuille de style précisée en paramètre.

Parser

API SAX

Voic une brève explication sur les différentes méthodes et leurs fonctions au sein du parser écrit à l'aide de l'API SAX.

Méthodes XML

Public void startDocument()

Cette méthode est appelée au début du parsage du document. Dans notre cas elle s'occupe de l'en-tête xml du document analysé et effectue un newLine.

Public void endDocument()

Cette méthode est appelée à la toute fin de l'analyse syntaxique. Celle-ci permet de flusher le flux de sortie (dans notre cas System.out).

Public void startElement()

La méthode startElement permet de vérifier la syntaxe de chaque balise XML, qui sont considérées comme des éléments par SAX. Nous pouvons donc vérifier la syntaxe des balises ouvrantes ainsi que leurs noms.

Public void endElement()

Cette méthode gère la fermeture des balises XML. Elle peut être utile dans le cas où on aurait besoin d'avertir d'autre méthode ou classe de la fin du parsage du document.

Méthodes Utilitaires

Private void emit()

Cette méthode a été définie afin d'écrire dans le flux de sortie choisi. Dans notre code, c'est donc sur System.out que le flux de sortie a été placé. La méthode va donc écrire le string passé en paramètre sur ce flux pour ensuite flusher le buffer.

Private void nl()

Cette méthode écrit tout simplement une nouvelle ligne sur le flux de sortie.

Résultat obtenu

Nous avons lancé le parseur afin qu'il analyse notre fichier personne.xml. Voici le résultat obtenu :

```
<?xml version='1.0' encoding='UTF-8'?>
<liste><personne><nom>Joh</><prenom>LuiTi</><email>joh.luiti@heig-vd.ch</><telephone
type="fixe"><indicatif>024</><numero>8224343</></><telephone
type="mobile"><indicatif>078</><numero>4453423</></></><personne><nom>Roger</><pr
enom>Federer</><email>roger.federer@heig-vd.ch</><telephone
type="fixe"><indicatif>024</><numero>1458985</></><telephone
type="mobile"><indicatif>078</><numero>1254789</></></></></>
```

Le parseur n'indique donc aucune erreur. Nous pouvons donc en conclure que la syntaxe du document est correcte. Si nous testons maintenant le parseur en y ajoutant une erreur voici le résultat obtenu :

The element type "personne" must be terminated by the matching end-tag "</personne>".

L'erreur que nous avons ajouté est une balise fermante </telephone> en trop. Nous remarquons donc que le parseur a effectivement détecté cette erreur.

Listing des emails

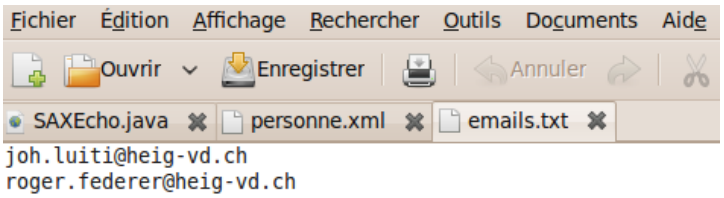
Le listing des emails de notre fichier « personne.xml » s'est fait à l'aide d'un FileWriter qui permet donc l'écriture dans un fichier emails.txt. Nous testons dans la méthode startElement() si la balise en cours est égale à « email » et mettons le boolean estEmail à true si c'est le cas :

```
{
    String eName = lName; // element name
    if ("".equals(eName)) eName = qName; // namespaceAware = false
    emit("<"+eName);
    if (attrs != null) {
        for (int i = 0; i < attrs.getLength(); i++) {
            String aName = attrs.getLocalName(i); // Attr name
            if ("".equals(aName)) aName = attrs.getQName(i);
            emit(" ");
            emit(aName+"=\""+attrs.getValue(i)+"\"");
        }
    }
    emit(">");

    if(qName.equals("email"))
        estEmail = true;
    else
        estEmail = false;
}
```

C'est ensuite dans la méthode characters() que nous testons le boolean (pour savoir si c'est un email). Si c'est le cas nous écrivons la valeur du string s dans le file. Enfin, nous fermons le FileWriter à l'aide d'un file.close().

```
public void characters(char buf[], int offset, int len)
throws SAXException
{
    String s = new String(buf, offset, len);

 // Si c'est un email on le met ds le fichier
    if(estEmail)
    {
        try
        {
            file.write(s+"\n");
        } catch (IOException e) {System.out.println(e);}
        estEmail = false;
    }
    emit(s);
}
```

Voici le résultat obtenu dans le fichier emails.txt :



Questions sur l'API

La raison pour laquelle que nous utilisions un SAXParserFactory pour ensuite créer un SAXParseur vient du fait que la classe SAXParseur est une classe abstraite et n'est donc pas instanciable directement. La classe SAXParserFactory permet de fabriquer des analyseurs SAX sur le document XML. Nous pouvons donc ensuite créer une instance SAXParseur (sur la base d'un SAXFactory) pour récupérer un de ces analyseurs. A noter qu'un objet SAXParserFactory prépare les caractéristiques et certaines propriétés des instances de la class SAXParseur(validation par DTD activée, traitement des espaces de nommage, etc…). Cette manière de faire est appelée « le concept de singleton » et consiste à limiter le nombre d'instanciation de la classe généralement à une et une seule.

Dans le cas d'un parser spécialisé pour un fichier XML donné, les deux méthodes principales qui changeraient seraient les méthodes startElement() et endElement() puisque c'est ces méthodes qui analyse la syntaxe des éléments XML et qui se basent sur les balises ouvrantes et fermantes de ces éléments. Nous pourrions donc analyser comme bon nous semble la syntaxe de chaque élément un par un et ainsi garantir l'intégrité du document xml. Les autres méthodes n'auraient pas besoin d'être modifiées.

API JDOM

Avant de pouvoir utiliser l'API JDOM, il nous a fallu effectuer les démarches suivantes :

- À l'aide de la console, exécuter le fichier build.bat (sous Window , build.sh sous Linux)se trouvant dans le répertoire jdom-1.0 (qui a comme résultat la construction d'un fichier jdom.jar)
- Ajouter le fichier *jar* dans la librairie du projet (pour se faire aller dans les propriétés du projet *Project/ Proprietes/ Java Build Path*, puis dans l'onglet *Libraries* et ajouter un fichier jar externe)

Code

Voici la première partie du code qui permet d'afficher les différents nœuds et leurs attributs.

```
//L'élément racine est initialisé avec la valeur de la racine du document
Element racine = doc.getRootElement();

//On crée une liste de tous les noeuds de la racine du document
List listpersonnes = racine.getChildren();

//On crée un Iterator sur notre liste
Iterator i = listpersonnes.iterator();
```

Nous créons tout d'abord un élément racine en appliquant getRootElement() sur le document xml. Cette racine nous permet ensuite de créer une liste de personnes grâce à la méthode getChildren, qui retourne une liste des enfants de la racine.

L'étape suivante consiste à parcourir cette liste et pour chaque personne à récupérer les différentes informations de celle-ci, soit le nom, le prénom, l'email ainsi que les numéros de téléphone.

Pour se faire nous avons procédé de la manière suivante et avons fait une boucle *while* afin de parcourir la liste.

```
Element e;
while(i.hasNext())
{
	e = (Element) i.next();

	System.out.println("Nom : " + e.getChildText("nom"));
	System.out.println("Prenom : " + e.getChildText("prenom"));
	System.out.println("Email : " + e.getChildText("email"));
```

Composition page HTML

Code

Le code ci-dessous permet de créer un document html sur la base d'une association entre un document xml et un document xsl.

```
File xmlFile = new File("./src/personne2.xml");
org.jdom.input.SAXBuilder saxb = new org.jdom.input.SAXBuilder();
org.jdom.Document doc = saxb.build(xmlFile.getPath()); // le document XML en mémoire


FileWriter w = new FileWriter("./src/personne2.html"); // à modifier pour sortie !

TransformerFactory transFact = TransformerFactory.newInstance();

if(transFact.getFeature(SAXTransformerFactory.FEATURE)){
    SAXTransformerFactory stf = (SAXTransformerFactory) transFact;
    TransformerHandler transHand = null;

    transHand = stf.newTransformerHandler(new StreamSource(new File("./src/personne2.xsl")));

    transHand.setResult(new StreamResult(w));
    SAXOutputter saxOut = new SAXOutputter(transHand);
    saxOut.output(doc);
    w.close();
```

Nous commençons par créer un File qui contient le xml de base. Les deux instructions qui suivent permettent de placer le document XML en mémoire. Vient ensuite la déclaration et la création d'un FileWriter sur le fichier de sortie html.

La classe TransformerFactory permet de « fabriquer » un transformer. Il est indispensable pour pouvoir ensuite créer un SAXTransformerFactory. Nous pouvons ensuite créer et initialiser l'objet transHand qui va permettre d'effectuer la transformation des documents xml et xsl. Nous appliquons ensuite la méthode setResult sur cet objet qui va écrire le flux sur le document w (.html).

Résultat obtenus

Voici le résultat du fichier html affiché par le navigateur Mozilla Firefox :

Nom: Joh
Prenom: LuiTi

Nom: Roger
Prenom: Federer

Nous constatons que cette manière est équivalente du point de vue du résultat à la méthode décrite précédemment (*voir « Transformation XSL »*).

Modification du document en mémoire

Pour montrer que la modification du document en mémoire est possible, nous avons tenté avec succès de supprimer une des deux personnes à l'aide du code suivant :

```
//L'élément racine est initialisé avec la valeur de la racine du document
Element racine = doc.getRootElement();
racine.removeChild("personne");
```

Nous avons ensuite constaté que la première personne a effectivement été supprimée.

Une application envisageable à cette manière de faire est la création de fichier HTML à la volée. Nous pouvons par exemple créer à la volée, sur la base de plusieurs fichiers XML qui auront tous le même fichier XSL comme fichier de mise en forme, différents document HTML très facilement.

Il peut être aussi intéressant de pouvoir tout simplement stocker un objet dans un fichier XML afin de pouvoir le récupérer plus tard.

Conclusion

Contrairement à se que beaucoup de gens croient, beaucoup de possibilités, autres que HTML, sont à notre disposition pour la création et la mise en forme de documents web. Ce laboratoire a été particulièrement intéressant dans le sens qu'il nous a permit d'apercevoir et prendre en main quelque unes de ces possibilités et de pouvoir nous apercevoir par nous même des possibilités et limitations de chacune d'entre elles.

De plus ces mécanismes étant présents dans univers du web, il est très important de connaître leur existence.

La seconde partie du laboratoire, ciblée sur les Parseur nous a fait découvrir un sujet totalement nouveau, mélangeant programmation en java et document XML, se que nous avons particulièrement apprécié.