

TP 5 : XML et Java

Ce TP est l'occasion de découvrir certains standards autour de XML : DTD, DOM, et XPath, et leur utilisation depuis Java.

Un squelette d'application est disponible depuis la page web pour faire les exercices – attention : il est nécessaire de faire pointer la variable d'environnement `JAVA_HOME` vers une version récente de Java. Il peut y avoir des problèmes pour compiler et exécuter le code fourni en exemple sous les versions anciennes de Java, et dans ce cas les problèmes devraient être résolus en demandant à `ivy` de télécharger `xerces` pour DOM et `xalan` pour XPath dans `ivy.xml`, et en ajoutant les chemins adéquats dans `build.xml`.

1 Documents XML

Le format de documents XML (pour *eXtensible Markup Language*) offre un standard de représentation pour des données structurées. Son importance actuelle dans le développement d'applications découle de l'intérêt de disposer d'un format unifié dans lequel échanger des données informatiques et développer de petits langages spécialisés (on a vu l'exemple des fichiers d'`ant` et d'`ivy`).

L'organisme responsable de l'écriture des standards XML que nous allons voir est le W3C, qui publie des recommandations dont la lecture est recommandée pour connaître les détails de ce que nous allons voir. Pour XML lui-même, cette recommandation est disponible à l'adresse <http://www.w3.org/TR/xml/>.

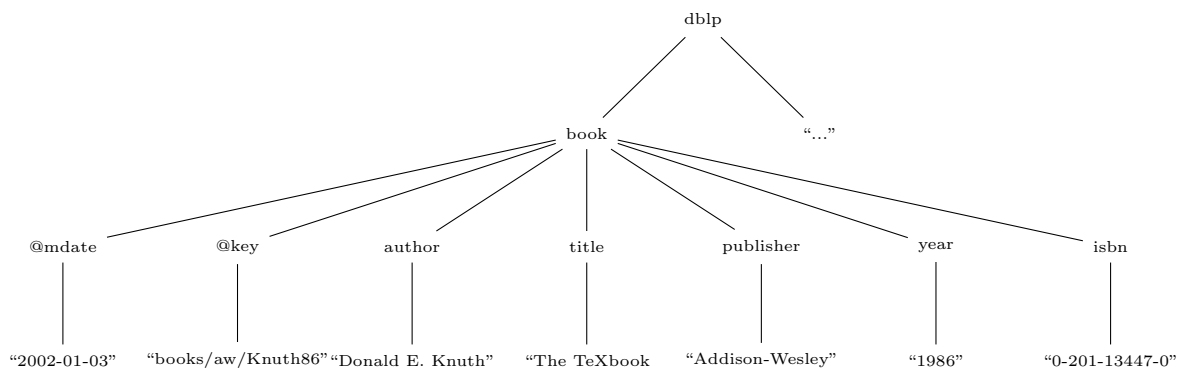
1.1 Structure d'un document XML

Un document XML typique aura la forme suivante :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>
  <book mdate="2002-01-03" key="books/aw/Knuth86">
    <author>Donald E. Knuth</author>
    <title>The TeXbook</title>
    <publisher>Addison-Wesley</publisher>
    <year>1986</year>
    <isbn>0-201-13447-0</isbn>
  </book>
  <!-- ... -->
</dblp>
```

- Un *en-tête* déclarant qu'il s'agit d'un document XML et précisant le jeu de codage de caractère si différent d'UTF-8.

- Une déclaration de type qui référence une “grammaire” décrivant à quel type de document XML on a affaire.
- La racine du document, ici `dblp`, qui va contenir tout le reste, et où on déclare aussi les espaces de nom si nécessaire.
- Le contenu du document, sous la forme d’un texte bien parenthésé entre des balises ouvrantes “`<balise>`” et fermantes “`</balise>`”. Chaque balise délimite ainsi un *élément* (ou un *nœud*) du document. Selon le type de document XML, un élément pourra avoir des sous-éléments, des attributs (comme `mdate` pour `book`), et parfois un contenu textuel (comme `author` dans l’exemple). Un élément sans contenu peut aussi s’écrire sous la forme `<balise />` plutôt que `<balise></balise>`.
- Les commentaires sont des nœuds spéciaux placés entre balises “`<!--`” et “`-->`”.



Un document XML peut être vu comme un arbre d’arité non bornée, où l’on peut de plus mettre les attributs des éléments et le contenu textuel dans des nœuds fils des éléments.

Les caractères “`<`”, “`>`” et “`&`” sont réservés, et peuvent être obtenus à l’aide d’*entités* “`< ;`”, “`> ;`” et “`& ;`”. D’autres entités peuvent être déclarées pour représenter par exemple des caractères spéciaux.

1.2 DTD

Il existe plusieurs formalismes pour décrire des dialectes XML, dont les DTD (*Document Style Definition*) – le plus ancien, le plus limité, mais aussi le plus simple... Une DTD définit un langage XML comme une grammaire algébrique définit un langage de mots.

Dans l’exemple précédent, le document XML était déclaré comme respectant la DTD `dblp.dtd` disponible localement. Voici un extrait de cette DTD

```

<!ELEMENT dblp (article|inproceedings|proceedings|book|incollection|
                phdthesis|mastersthesis|www)*>
<!ENTITY % field "author|editor|title|booktitle|pages|year|address">

<!ELEMENT article      (%field;)*>

```

```
<!ATTLIST article      key CDATA #REQUIRED
                      reviewid CDATA #IMPLIED
                      rating CDATA #IMPLIED>
```

```
<!ELEMENT author      (#PCDATA)>
```

Cet extrait déclare trois types d'éléments (`dblp`, `article` et `author`) et un type union (`field`). Le contenu permis pour un élément `dblp` est ainsi n'importe quelle séquence d'éléments de type `article`, `inproceedings`, `proceedings`, ... tandis qu'un élément `author` ne peut contenir que du texte (`#PCDATA`). On pourrait forcer un `article` à contenir une séquence d'auteurs suivie d'un titre (et rien de plus) avec la déclaration

```
<!ELEMENT article      (author*, title)>
```

L'élément `dblp` ne permet pas d'attributs, mais `article` le fait, avec un attribut `key` obligatoire qui peut contenir n'importe quel type de valeur (`CDATA`), et deux attributs `reviewid` et `rating` optionnels. D'autres types de valeur pour les attributs sont `ID` pour un identifiant qui doit alors être unique dans le document, et `IDREF` pour une référence à un tel identifiant.

2 Lecture et écriture de fichiers XML

La manipulation de documents XML dans différents langages a été standardisée par le W3C au sein du DOM (*Document Object Model*). En particulier, le niveau 3 de cette recommandation fournit des moyens standards, valables dans tous les langages qui implémentent le DOM, pour lire et écrire des documents XML. La recommandation correspondante est à l'adresse <http://www.w3.org/TR/DOM-Level-3-LS/>.

Nous allons voir des exemples simples de lecture et d'écriture de fichiers XML en Java, sachant que la démarche serait la même dans d'autres langages.

2.1 Lecture

```
import org.w3c.dom.bootstrap.DOMImplementationRegistry;
import org.w3c.dom.Document;
import org.w3c.dom.ls.DOMImplementationLS;
import org.w3c.dom.ls.LSParser;

...

DOMImplementationRegistry registry =
    DOMImplementationRegistry.newInstance();

DOMImplementationLS impl =
    (DOMImplementationLS)registry.getDOMImplementation("LS");
```

```
LSParser builder =  
    impl.createLSParser(DOMImplementationLS.MODE_SYNCHRONOUS, null);  
  
Document document = builder.parseURI("data/personal.xml");
```

2.2 Écriture

```
import org.w3c.dom.bootstrap.DOMImplementationRegistry;  
import org.w3c.dom.Document;  
import org.w3c.dom.ls.DOMImplementationLS;  
import org.w3c.dom.ls.LSSerializer;  
  
...  
  
DOMImplementationRegistry registry = DOMImplementationRegistry.newInstance();  
  
DOMImplementationLS impl =  
    (DOMImplementationLS)registry.getDOMImplementation("LS");  
  
...  
  
LSSerializer writer = impl.createLSSerializer();  
String str = writer.toString(document);
```

Exercice 1. Compiler et exécuter le squelette d'application du TP. Quelles sont les erreurs de la DTD ? Corriger la DTD et valider le document `resources/bib.xml`.

2.3 Gestion des erreurs

La gestion des erreurs de fait via la classe `org.w3c.dom.DOMErrorHandler`, pour laquelle on peut trouver une implémentation dans `fr.ens_cachan.dptinfo.tp05.ErrorHandler`.

3 Utilisation du DOM

Le cœur du DOM est une interface de manipulation des documents XML en mémoire. En particulier la navigation dans les documents et leur édition sont standardisés dans la recommandation <http://www.w3.org/TR/DOM-Level-3-Core/>.

La classe de base de la représentation DOM d'un document XML est `org.w3c.dom.Node`. Tous les nœuds de l'arbre correspondant au document XML y sont représentés, y compris les nœuds de texte (qui peuvent ne contenir que des caractères blancs d'indentation !), et les nœuds des attributs. Le type d'un nœud est fourni par la méthode `getNodeType()`, qui retourne un entier qui pourra être par exemple `ELEMENT_NODE`, `ATTRIBUTE_NODE`, `TEXT_NODE` ou `COMMENT_NODE`. On a vu mieux comme modélisation objet, mais il faut garder à l'esprit que DOM doit pouvoir être implémenté dans de nombreux langages.

L'objet `Document` fourni par le parser de la section précédente peut être utilisé comme un `Node`, qui sera alors le nœud racine de l'arbre XML.

3.1 Navigation basique

La classe `org.w3c.dom.Node` fournit les méthodes de navigation dans l'arbre XML à partir du nœud courant `getParentNode()`, `getChildNodes()`, `getFirstChild()`, `getLastChild()`, `getPreviousSibling()`, `getNextSibling()`, `getAttributes()` et `getTextContent()`, avec les effets que l'on peut deviner.

Le nom d'un nœud est disponible par les méthodes `getNodeName()` et `getLocalName()`.

Exercice 2. Utiliser DOM pour compléter la classe `DOMVisitor` et afficher le type, le nom et le contenu de chaque nœud du document `resources/bib.xml`.

3.2 Navigation avancée

La recommandation <http://www.w3.org/TR/DOM-Level-2-Traversal-Range/> fournit des interfaces de plus haut niveau pour itérer une recherche dans un document :

- `org.w3c.dom.traversal.TreeWalker` pour parcourir un arbre XML dans plusieurs directions,
- `org.w3c.dom.traversal.NodeIterator` pour une interface d'itérateur simple,
- `import org.w3c.dom.traversal.NodeFilter` pour ignorer certains nœuds lors de la navigation.

On obtient un objet `TreeWalker` ou `NodeIterator` par un appel à `createTreeWalker()` ou `createNodeIterator()` sur un objet de la classe `DocumentTraversal` : cette interface est implémentée en particulier par tous les `Documents` DOM fournis par `xerces`, donc il suffit d'écrire un appel de la forme

```
((DocumentTraversal)doc).createNodeIterator(...)
```

pour utiliser ce mode de navigation.

Ces interfaces ne sont pas implémentées par défaut dans l'environnement Java, il faut ajouter par exemple `xerces` dans les dépendances du projet et dans les chemins de compilation et d'exécution pour pouvoir en profiter. La documentation pour ces interfaces peut être trouvée à l'adresse <http://xerces.apache.org/xerces2-j/javadocs/api/org/w3c/dom/traversal/>.

Exercice 3. Implémenter une recherche de tous les articles d'un auteur dont le nom est fourni en ligne de commande.

4 XPath

Lors de la manipulation de documents XML, on souhaite souvent travailler sur un ensemble d'éléments précis qu'il peut être laborieux d'énumérer à travers un parcours linéaire du document qui n'utilise que des primitives simples comme *child*, *parent*, *next*.

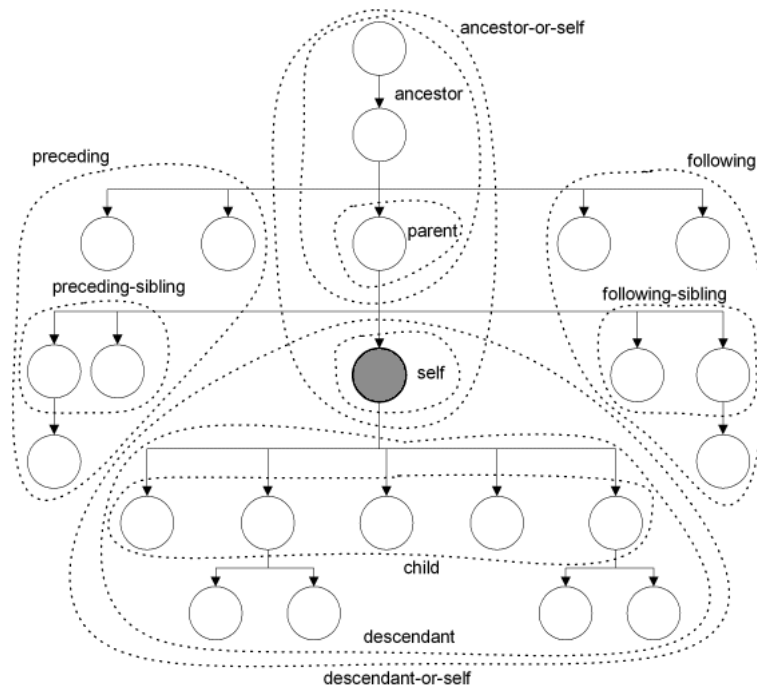
Pour éviter de devoir réécrire un algorithme de parcours différent à chaque fois que l'on souhaite extraire certains éléments particuliers, un langage de sélection d'éléments XML a été conçu : *XPath*. La recommandation W3C correspondante est <http://www.w3.org/TR/xpath>.

4.1 Chemins et axes

XPath est un langage qui permet d'écrire des sortes d'expressions régulières sur les documents XML. Comme son nom l'indique, ce langage raisonne surtout sur les chemins. Les chemins XPath fonctionnent un peu comme l'adressage dans l'arborescence des fichiers UNIX, avec "." pour désigner l'élément courant, "/" pour désigner la racine ou pour trouver les éléments fils, et ".." pour remonter au père; le chemin "a//b" trouve tous les éléments b descendants de l'élément a. Enfin, on peut utiliser le wildcard * : a/* désigne tous les fils de a, et b//* désigne tous les descendants de b.

Par exemple, si on travaille sur un fichier XHTML, l'expression XPath suivante
`/body/h1/*/b`

va sélectionner tous les fils b de n'importe quel nœud dont le père est un nœud h1, lequel est lui-même le fils d'un nœud body à la racine du document. Ce chemin est absolu (il commence par la racine "/"), mais en général une requête XPath va sélectionner des nœuds en suivant un chemin relativement à un *nœud courant*.



Les chemins que nous venons de voir sont en fait des notations abrégées pour différents *axes* de recherche : "." correspond à l'axe `parent::`, "/" à l'axe `descendant::`.

Des généralisations de ces axes existent, comme `descendant-or-self::`, `ancestor::`, `ancestor-or-self::`, et ainsi de suite.

Enfin, on peut tester l'existence d'un frère avant ou après l'élément atteint par `preceding-sibling::` et `following-sibling::` respectivement. Ainsi, le chemin `preceding-sibling::a/b` trouve un élément `b` fils d'un élément `a` qui précède immédiatement le nœud courant.

4.2 Prédicats

Pour chaque nœud désigné dans un chemin XPath, on peut ajouter un prédicat à vérifier entre crochets. Un tel prédicat peut lui-même être une requête XPath, qui est interprétée comme vraie si l'ensemble de nœuds sélectionné est non vide. Ainsi l'expression

```
/a/b[../e]/c[../d]
```

sélectionnera les nœuds `c` qui ont un fils `d` et un parent `b` qui a lui-même un frère `e` et un père `a`. Le prédicat peut-être un nombre qui désigne le numéro d'occurrence de l'élément courant. Par exemple

```
//p[1]//a[2]
```

sélectionne le deuxième lien du premier paragraphe d'un document XHTML. On peut utiliser dans un prédicat la fonction `last()` qui renvoie le nombre d'occurrences d'un élément :

```
//p//a[last()]
```

Cette expression renvoie le dernier lien de chaque paragraphe du document. D'autre part il est possible de faire des tests arithmétiques dans les prédicats XPath. Ainsi, pour obtenir les 3 premiers titres de première importance d'un document XHTML, on utiliserait la requête XPath suivante :

```
/body/h1[position() <= 3]
```

`position()` est en effet une fonction qui renvoie la position du nœud courant, et la notation `[1]` est un raccourci pour `[position()=1]`. `count(path)` est une autre fonction utile qui prend en argument un chemin XPath relatif au nœud courant et compte le nombre de nœuds qui vérifient ce chemin.

```
//h1[count(h2) >= 2]
```

sélectionne les nœuds `h1` qui ont au moins deux fils `h2`.

```
/*[count(*) = 3]
```

sélectionne tous les nœuds à la racine qui ont exactement 3 fils. Les prédicats peuvent être combinés par les opérations booléennes classiques : **and**, **or** et **not**.

XPath permet de faire des appels de fonctions. La bibliothèque de base propose en particulier des manipulations de chaînes de caractères, comme les noms des éléments : **local-name()** renvoie le nom du nœud courant sans son préfixe d'espace de nom, **name()** renvoie le nom complet du nœud courant, **starts-with(*chaîne1*, *chaîne2*)** vérifie si *chaîne2* est un préfixe de *chaîne1*, **contains(*chaîne1*, *chaîne2*)** vérifie si *chaîne2* est un facteur de *chaîne1* et **substring(*chaîne1*, *i*, *n*)** renvoie la sous-chaîne de longueur *n* à partir du *i*ème caractère (l'argument *n* est optionnel). On peut bien sûr tester l'égalité des chaînes de caractères avec "=". Par exemple, on peut extraire l'ensemble des titres d'un document XHTML avec l'expression XPath suivante

```
/body/*[starts-with(local-name(),"h")]
```

4.3 Attributs

Les éléments des documents XML peuvent (et doivent pour certains) avoir des attributs. En XPath comme en DOM, un attribut est un nœud fils du nœud de l'élément XML dont il fait partie. On le désigne avec **@*attribute*** où *attribute* est le nom de l'attribut en question. Ainsi

```
//p/@name
```

va sélectionner les attributs **name** de tous les éléments **p**.

Naturellement, les attributs peuvent être utilisés dans les tests de prédicats :

```
/body/h1//*[ @id="main"]
```

cette expression va sélectionner les éléments descendants des éléments **h1** qui ont un attribut **id** dont la valeur est **main**.

On peut également utiliser des wildcards pour désigner l'ensemble des attributs d'un élément. Par exemple, pour obtenir l'ensemble des attributs d'une image dans un document XHTML, on peut utiliser l'expression XPath suivante :

```
//img/@*
```

Enfin, pour demander la présence d'un attribut dans un élément, il suffit de mettre le nom de cet attribut seul dans un prédicat.

```
//*[ @id]  
/body//p[ @*]
```

La première expression sélectionne tous les éléments qui ont un nœud **id**, et la seconde sélectionne tous les paragraphes qui ont au moins un attribut.

4.4 XPath en Java

Les interfaces pour XPath sont situées dans le paquet `javax.xml.xpath`. Une expression XPath peut être évaluée en Java en suivant les étapes suivantes :

```
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
XPathExpression expression = xpath.compile("/doc/name");
String result = expression.evaluate(theDocument);
```

Les fichiers fournis pour le TP donnent un exemple (commenté) d'utilisation de XPath qui répond en partie à l'exercice précédent.

Utiliser jaxen Une implémentation tierce de XPath est fournie par <http://www.jaxen.org>, et permet d'évaluer des expressions XPath sur différentes interfaces de navigation dans des arbres XML (dont DOM, mais aussi jdom, dom4j et XOM qui offrent des interfaces mieux intégrées dans Java).

Pour utiliser **jaxen**, il suffit de l'ajouter aux dépendances dans `ivy.xml` et aux chemins de `build.xml`. En ajoutant les paquets `org.jaxen` et `org.jaxen.dom`, le code précédent s'écrirait alors :

```
XPath xpath = new DOMXPath("/doc/name");
List result = xpath.selectNodes(theDocument);
```

À noter que certains ont rencontré des problèmes avec **xalan** sous Java 1.5, et que dans ce cas **jaxen** peut être une bonne solution de secours...

Exercice 4. Écrire une méthode d'affichage d'une entrée bibliographique qui suit le format de `dblp.dtd`. En particulier, il est nécessaire d'afficher les informations des conférences référencées par une entrée `inproceedings` : faites la recherche en utilisant XPath.