UPMC/Licence/Info/2I013
Rushdroid – XML

Janvier 2016

1 Structure XML

Extensible Markup Language héritier de SGML (*Standard Generalize Markup Language*) est un language formel de description de *documents* normé par le W3C (*World Wide Web Consortium* – `http://www.w3.org/TR/REC-xml/`).

Les *balises* (*Markup*) du langage permettent d'annoter le contenu d'un document distinguant ainsi ses divers éléments, ce qui lui confére une *structure* (par exemple, on distingue dans un texte les titres et les paragraphes). Ces indications de structures sont destinées au traitement informatique du document. Le format XML est donc un *format de données* pour un traitement par programme informatique.

Un usage du format XML peut être plus simplement de décrire une *structure de données*. On retrouve d'ailleurs en XML les structures de données classiques en programmation : les listes et les arbres.

XML et les bons parenthésages Le principe générique de structuration d'un document XML est le *bon parenthèsage*.

Un bon parenthèsage est une suite judicieusement ordonnée de parenthèses ouvrantes et de parenthèses fermantes. On peut facilement en donner une définition *récursive* :

— la suite vide est un bon parenthèsage ;

— si p est un bon parenthèsage, alors `(`p`)` est un bon parenthèsage ;

— si p_1 et p_2 sont des bons parenthèsages, alors la suite p_1p_2 est un bon parenthèsage.

On peut utiliser n'importe quelle paire de symboles pour fabriquer des bons parenthèsages, par exemple `[` et `]` ou `begin` et `end`. Ce qui importe, c'est de respecter l'imbrication d'ouvrantes et de fermantes induit par la définition récursive ci-dessus.

XML utilise des *balises* qui sont des suites de caractères délimitées par des *chevrons* : les symboles `<` et `>`. Pour distinguer une balise fermante d'une balise ouvrante, on utilise le symbole `/` que l'on place directement après le `<`. Si n est un *nom* (suite de caractères répondant aux critères du *Name* du W3C) alors les suites `<`n`>` et `</`n`>` forment une paire de balises, respectivement ouvrante et fermante.

Balises XML Une basile ouvrante peut être plus complexe qu'un simple nom placé entre chevrons. Il est possible d'enrichir une balise ouvrante d'*attributs*. On les donne sous forme d'une liste de noms (d'attributs) auxquels on peut associer une valeur. Le symbole `=` est placé entre le nom d'un attribut et la valeur qu'on lui donne. La valeur est souvent mise entre guillements (symbole `"`). Une paire de balises ouvrante et fermante prend alors la forme `<`n n_1`="`v_1`" ...` n_k`="`v_k`">` et `</`n`>`, Où n est le nom de la balise, $n_1 \dots n_k$ sont les noms des attributs et $v_1 \dots v_k$ dénotent leur valeur.

Il est possible qu'une balise soit à la fois ouvrante et fermante. Dans ce cas, le chevron `>` est précédé de `/`. On aura, par exemple, `<`n n_1`="`v_1`" ...` n_k`="`v_k`"/>`.

XML et arbres La structure que donne à un document un balisage XML est une structure d'*arbre général*. L'imbrication des balises donne les niveaux des éléments correspondant dans l'arborescence.

2 Une API JAVA pour XML

Il y a deux étapes dans le traitement par programme d'une donnée XML

1. analyser le fichier texte XML pour construire sa représentation arborescente en mémoire ;
2. exploiter la structure arborescente pour en extraire les informations voulues.

Il existe en JAVA des bibliothèques permettant de traiter les documents textes XML et d'explorer leurs structures. Elles fournissent des fonctionnalités d'analyse du texte XML pour construire une représentation équivalente en mémoire. Cette représentation répond aux éléments de spécification du DOM (*Document Object Model*). Le DOM est également un standard du W3C (`http://www.w3.org/DOM/`). Ce standard définit une API qui offre des primitives de navigation et d'extraction de données pour la structutre arborescente du DOM.

Analyse du fichier XML On utilise les fonctionnalités des bibliothèques `javax.xml.parsers.DocumentBuilderFactory` et `javax.xml.parsers.DocumentBuilder` pour créer un analyseur qui construira une instance de `Document` (interface fournie par `org.w3c.dom.Document`).

Exemple de méthode de lecture d'un document XLM

```
public Document readXMLFile(String fname) {
  try {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    return builder.parse(fname);
  } catch (Exception ex) {
    return null;
  }
}
```

La méthode `parse` de `DocumentBuilder` peut également prendre en argument un `InputStream`.

Le DOM et son exploitation Le DOM est une structure arborescente de *nœuds* qui peuvent être des *éléments*, un simple bloc de *texte* ou d'autres choses encore. Les *éléments* peuvent à leur tour contenir d'autres éléments, blocs de texte ou autre. Les blocs de texte sont considérés comme des *feuilles* de la structure arborescente. Un *document* DOM contient un *élément racine* unique.

Lorsqu'un DOM provient de l'analyse d'un flux XML, les *éléments* correspondent aux balises et les blocs de textes, au texte simple (non balisé) que l'on trouve entre les balises.

Les fonctionnalités d'exploitation de la structure DOM sont fournies par les interfaces données par `org.w3c.dom`. On pourra insérer

```
import org.w3c.dom.*;
```

dans le code chargé de l'exploitation du DOM : on se servira à peu près de tout. On y trouve, en particulier, les interfaces `Document` et `Element` qui implémentent toute deux l'interface `Node`.

Si `theDoc` est un `Document`, alors `theDoc.getDocumentElement()` permet de récupérer l'élément racine. Le résulat est une instance de `Element`.

Si `node` est un `Node` alors `node.getChildNodes()` donne la liste des instances de `Node` qu'il contient. Cette liste est une instance de `NodeList`.

Ceci vaut également si `node` est une instance de `Element`. Dans ce cas, on a également que `node.getNodeName()` donne le nom de l'élément (i.e. du point de vue XML : le nom de la balise) et que `node.getAttributes()` donne la liste des couples d'attributs et valeurs associés à cet élément. Le résultat obtenu est une instance de `NamedNodeMap`.

Si `nodes` est un `NodeList`, alors `nodes.getLength()` donne le nombre d'objets que contient `nodes` ; si `i` est un entier (compris entre 0 et `nodes.getLength()`, strictement) alors `nodes.item(i)` donne le `i`ème élément de la liste – instance de `Node`.

Si `attrs` est un `NamedNodeMap`, alors `attrs.getLength()` donne le nombre d'objets contenus dans `attrs` ; si `i` est un entier (compris entre 0 et `attrs.getLength()`, strictement) et si `name` est une chaîne de caractères, alors `attrs.getNamedItem(name)` donne l'association nom-valeur de l'attribut `name`. C'est une instance de `Node`. Pour obtenir la valeur de l'attribut, on peut forcer le type du résultat vers `Attr`. Si `attr` est un `Attr` alors `attr.getValue()` donne la valeur de l'attribut comme une chaîne de caractères (instance de `String`).

ATTENTION : `getChildNodes` donne TOUS les nœuds contenus dans un élément. En particulier, les tabulations, les espaces ou les passages à la ligne peuvent constituer de tels nœuds. Il faut avoir cela en tête lorsque nous explorerons les DOM construits à partir de nos fichiers XML.

3 Format XML pour Rushdroid

Le fichier contient un ensemble de descritpions de configurations initiales pour une partie de *rush hour*.

La balise racine de la donnée XML est `rushpuzzles`.

Une configuration est est délimitée par la balise `puzzle`. Elle possède un attribut `name` dont la valeur est l'identifiant de la configutation.

Une configuration est décrite par une liste de pièces. La balise `piece` contient la description d'une pièce. C'est une balise vide. Elle possède les attributs suivants :

`id` l'identifiant de la pièce. C'est une valeur numérique entière.

`len` sa valeur donne la longueur de la pièce.

`dir` dont la valuer donne l'orientation de la pièce. Ce doit être `"H"` ou `"V"`, pour, respectivement, horizontale et verticale.

`col` dont la valeur donne le numéro de colonne de la position de la pièce.

`lig` dont la valeur donne le numéro de ligne de la position de la pièce.

DTD Un document XML est formellement définit par une `DTD` (pour *Document Type Définition*. Le format de description d'une document XML adopte la forme d'une *grammaire*.

La grammaire des *DTD* définition un type de document : `!DOCTYPE`. Chaque élément est définit comme un `!ELEMENT` et les attributs sont définis pour un élément comme des `!ATTLIST`.

Chaque définition est incluse entre chevrons.

La définition `!DOCTYPE` donne le nom de la balise racine, puis la liste des éléments. La liste est incluse entre un crochet ouvrant (caractère `[`) et un crochet fermant (caractère `]`).

La définition `!ELEMENT` d'un élément donne le nom de la balise XML correspondante. Si l'élément est vide, on l'indique avec `EMPTY`, sinon, on indique, entre parenthèses, le nom des éléments qu'il contient.

La définition `!ATTLIST` d'un attribut indique le nom de l'élément auquel il est associé et le nom de l'attribut lui-même. Le type de valuer que l'on peut donner à l'attribut peut être spécifié. Par exemple, `ID` indique un identifiant qui doit être unique dans le document XML ; `CDATA` indique une simple chaîne de caractères ; on peut limité les valeurs prises par un attributs à un nombre fini en les énumérant, entre parenthèses et en les séparant par la barre verticale (caractère |). Lorsqu'un attribut doit être obligatoirement définit, on l'indique avec `#REQUIRED`.

La DTD pour Rushdroid est la suivante

```
<!DOCTYPE rushpuzzles [
<!ELEMENT rushpuzzles (puzzle)>
<!ELEMENT puzzle (piece)>
<!ATTLIST puzzle num CDATA #REQUIRED>
<!ELEMENT piece EMPTY>
<!ATTLIST piece id ID #REQUIRED>
<!ATTLIST piece len (2|3) #REQUIRED>
<!ATTLIST piece dir (H|V) #REQUIRED>
<!ATTLIST piece col CDATA #REQUIRED>
```

```
<!ATTLIST piece lig CDATA #REQUIRED>
]>
```