XML - Semaine 7

Pierre Nerzic

février-mars 2022

Le cours de cette semaine présente :

- l'analyse d'un document XML à l'aide de l'API SAX pour Java,
- l'écriture d'un document XML en langage PHP.

Pierre Nerzic

Simple API for XML

Pierre Nerzic

Présentation

Cette interface de programmation permet de lire et traiter un document XML sans le stocker entièrement en mémoire. C'est au contraire de DOM qui stocke la totalité du document sous forme d'un arbre de `Node`.

SAX est destiné à traiter des documents qui sont trop gros à stocker en mémoire ou dont on n'a pas besoin de parcourir le contenu de manière aléatoire. SAX ne permet qu'un seul parcours du document, dans l'ordre dans lequel il a été enregistré.

SAX signifie *Simple API for XML*, mais aurait pu être appelée *Sequential Access for XML*.

Pierre Nerzic

Principes de SAX

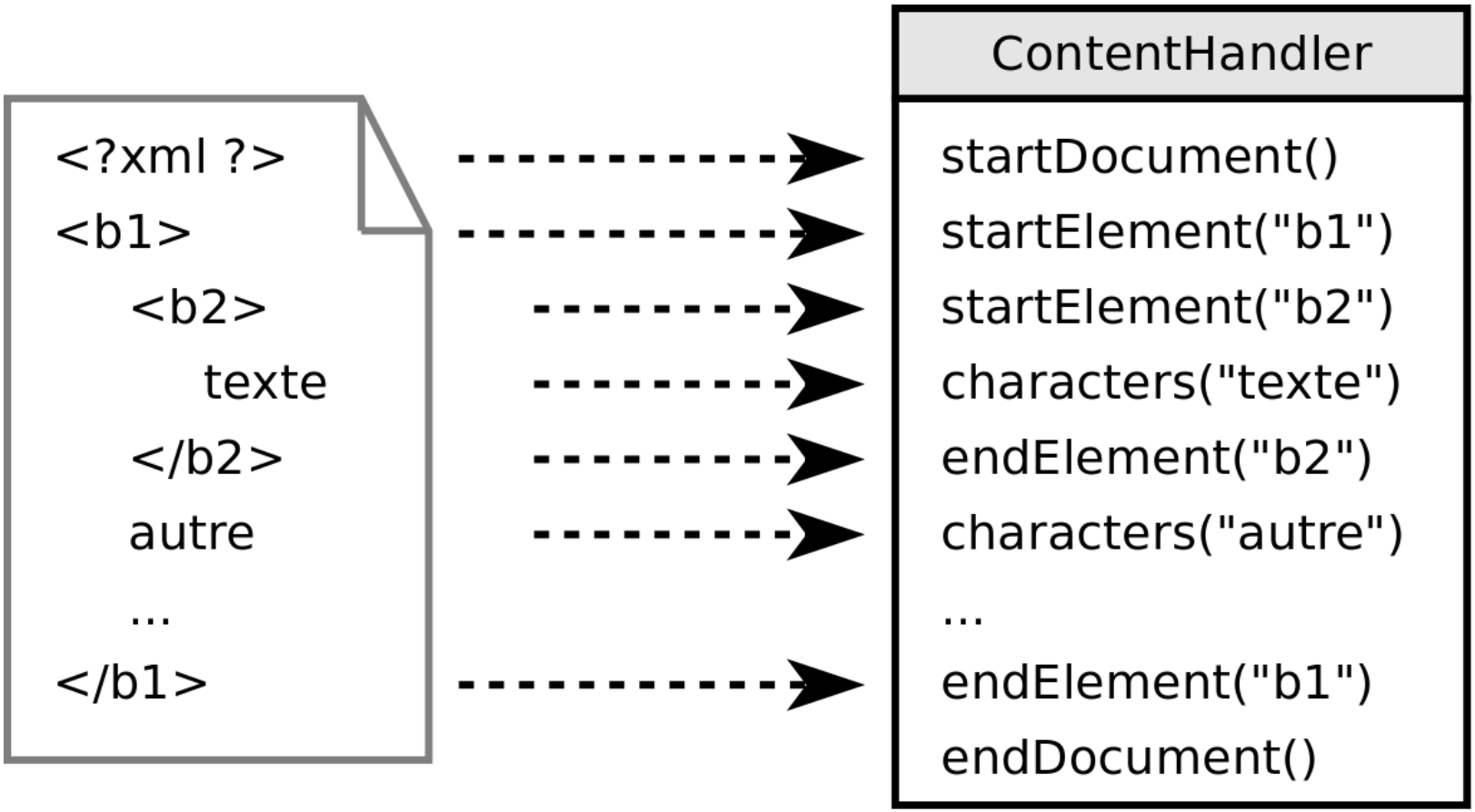
Avec SAX, vous devez construire un écouteur (*listener*), c'est à dire une classe possédant certaines méthodes publiques. Cet écouteur est fourni à SAX et ses méthodes sont appelées en fonction de ce qui se trouve dans le document XML. C'est de la programmation événementielle.

C'est comme avec les interfaces Swing, vous définissez un écouteur pour les clics souris. Lorsque l'utilisateur clique, cela appelle la méthode que vous avez définie.

Avec SAX, votre écouteur doit implémenter les méthodes de l'interface `org.xml.sax.ContentHandler` ou sous-classer `org.xml.sax.DefaultHandler` qui en est une implémentation par défaut.

Fonctionnement de SAX

SAX parcourt le document et le découpe en fragments : balises ouvrantes, balises fermantes, textes... À chaque fragment rencontré, il appelle une méthode spécifique de l'écouteur.



Pierre Nerzic

Interface ContentHandler

Cette interface Java définit ce que doit implémenter un écouteur SAX. Ce sont 11 méthodes, dont :

- `void startDocument()` : appelée quand on est au début du document
- `void endDocument()` : appelée à la fin du document
- `void startElement(String uri, String localName, String qName, Attributes attrs)` : on arrive sur une balise ouvrante, le paramètre qName est son nom qualifié (préfixe:nom local), `attrs` contient la liste des attributs, voir le transparent suivant.
- `void endElement(String uri, String localName, String qName)` : appelée quand on arrive sur une balise fermante dont le nom qualifié est qName.

Type Attributes

Le type `Attributes` mentionné dans `startElement` représente un tableau d'attributs :

- `String getValue(String nomqual)` retourne la valeur de l'attribut ayant ce nom qualifié (préfixe:nom)
- `String getValue(String uri, String nom)` retourne la valeur de l'attribut ayant cet URI pour identifiant de préfixe et ce nom local.

Il y a d'autres méthodes pour parcourir les attributs un par un :

- `String getLength()` retourne le nombre d'attributs
- `String getQName(int i)` retourne le nom qualifié du i^e attribut
- `String getValue(int i)` retourne la valeur du i^e attribut

Interface `ContentHandler` (suite)

Suite des méthodes d'un `ContentHandler` :

- `void characters(char[] ch, int start, int length)` : signale une zone de texte qui est à extraire du tableau `ch` par : `String texte = new String(ch, start, length);`
  - Attention : les zones de texte sont tout ce qu'il y a entre deux balises, y compris les retours à la ligne et espaces d'indentation. Donc il faudra nettoyer les chaînes lues des espaces avant et après : utiliser la méthode `trim()` des chaînes.
  - Attention aussi car les entités peuvent être (ou pas) concaténées avec les textes qui les entourent. Donc il faudra concaténer tous les textes successifs.

Texte, CDATA et entités

Soit le document suivant : 

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE info [ <!ENTITY ent "texte4"> ]>
<info>texte1<![CDATA[texte2]]>texte3&ent;texte5</info>
```

Son analyse par SAX produit les événements suivants :

1. `startElement("", "info", "info", [])`
2. `characters("texte1")`
3. `characters("texte2")`
4. `characters("texte3")`
5. `characters("texte4texte5")`
6. `endElement("", "info", "info")`

On voit que l'entité `&ent;` a été remplacée par sa valeur et concaténée avec `texte5` mais pas avec `texte3`.

Programmation d'un analyseur

Pierre Nerzic

Implémentation d'un ContentHandler

Pour traiter la plupart des documents, on peut se contenter de définir `startElement`, `endElement` et `characters` et dériver la classe `DefaultHandler` qui implémente `ContentHandler` :

```
class MonHandler extends DefaultHandler {
    public void startElement(...) throws SAXException {
        ...
    }
    public void endElement(...) throws SAXException {
        ...
    }
    public void characters(char[] text, int debut, int lng) {
        String texte = new String(text, debut, lng);
        ...
    }
}
```


Pierre Nerzic

Lancement de l'analyse

Ensuite, voici comment on lance le travail sur un URL :

```
void Analyser(String documentURL) throws Exception {
    // créer un générateur d'analyseur
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setNamespaceAware(true);
    factory.setValidating(true);
    // créer un analyseur
    SAXParser parser = factory.newSAXParser();
    // créer un écouteur qui sera activé par l'analyseur
    MonHandler handler = new MonHandler();
    // lancer l'analyse sur l'URI : fichier ou http://...
    parser.parse(documentURL, handler);
}
```

documentURL est le nom d'un fichier ou un URL sur le réseau.

Gestion des erreurs

La classe `DefaultHandler` implémente l'interface `ErrorHandler` qui récupère les exceptions provoquées par les erreurs. Le problème est qu'elle n'affiche rien. Alors en général, on surcharge au moins la méthode `fatalError` :

```
class MonHandler extends DefaultHandler {
    ...
    public void fatalError(SAXParseException e) {
        System.err.println(
            "Erreur fatale ligne "+e.getLineNumber()
            +" colonne "+e.getColumnNumber());
        System.err.println(e.getMessage());
    }
}
```

Après une erreur fatale, l'analyse s'arrête définitivement.

Traitement d'un document XML

Pierre Nerzic

Aucune visibilité globale

La conséquence du fonctionnement événementiel de SAX, c'est qu'on n'a aucune vision globale du document. Par exemple dans :

```
<voiture id="871"><prix monnaie="yen">331212</prix>...
```

Voici les événements déclenchés en séquence mais indépendamment les uns des autres :

1. `startElement("","voiture","voiture",[id="871"])`
2. `startElement("","prix","prix",[monnaie="yen"])`
3. `characters("331212")`
4. `endElement("","prix","prix")`

Comment faire pour convertir le prix en euros et l'associer à la voiture ? Quand on est dans la méthode `characters`, on ne dispose plus des attributs de la balise ouvrante `prix`.

Mémoriser les informations au passage

Le principe est de mémoriser certaines informations pendant le parcours des données :

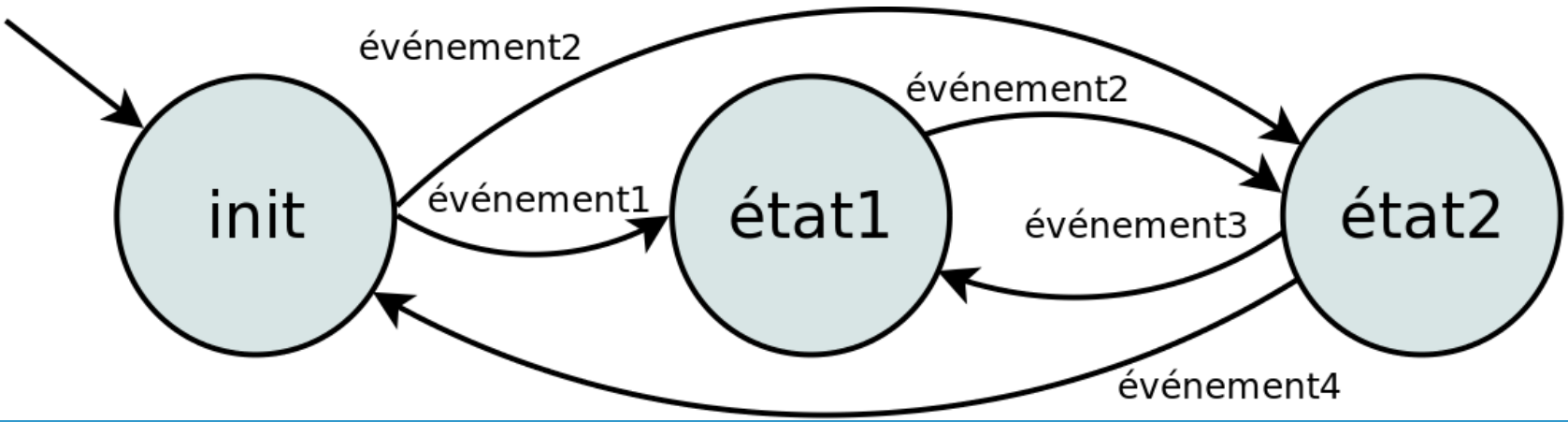
- Il faut mémoriser les informations dont on a besoin. Par exemple, quand on rencontre l'élément `prix`, il faut mémoriser la valeur de l'attribut monnaie ou le taux de change, afin de pouvoir faire la conversion au moment où on rencontrera le texte de la valeur.
- Il faut aussi gérer un *état* indiquant où on se trouve dans l'arbre XML sous-jacent, par exemple, pour savoir quand on est dans le texte de l'élément `<prix>` parce que tous les textes sont gérés par la même méthode `characters`.

Pour résoudre élégamment ces problèmes, il est recommandé de faire appel à un **automate à états**, et plus particulièrement une **Machine de Mealy**.

Automate à états

Un automate à états finis est un mécanisme abstrait possédant différents états possibles, et l'un d'entre eux est l'*état courant*. La machine peut passer d'un état à l'autre, mais c'est défini par une liste de *transitions* possibles entre ses états, déclenchées par des événements. Au début la machine est dans l'un des états désigné comme étant l'état initial.

On représente une telle machine par un graphe. Les nœuds sont les états et les arcs sont les transitions possibles. Exemple :



Pierre Nerzic

Programmation d'un automate à états

C'est assez simple. Chaque état est représenté par un code. Il y a un écouteur par événement et un aiguillage selon l'état courant :

```
private enum Etat {INIT, ETAT1, ETAT2};
private Etat m_EtatCourant = INIT;
public void onEvenement1() {
    switch (m_EtatCourant) {
        case INIT:  m_EtatCourant = Etat.ETAT1; break;
        default: break;
    }
}
public void onEvenement2() {
    switch (m_EtatCourant) {
        case INIT:  m_EtatCourant = Etat.ETAT2; break;
        case ETAT1: m_EtatCourant = Etat.ETAT2; break;
        default: break;
    }
}
```


Pierre Nerzic

Machine de Mealy

Tel quel, un automate à état ne peut pas faire grand chose. Une Machine de Mealy définit en plus des traitements à faire sur les transitions. L'arrivée d'un événement déclenche non seulement le passage d'un état à l'autre, mais aussi un traitement.

Par exemple quand on passe de l'état INIT à l'état ETAT2 à cause de l'événement 2, on peut mémoriser une information, incrémenter un compteur, afficher un message, etc.

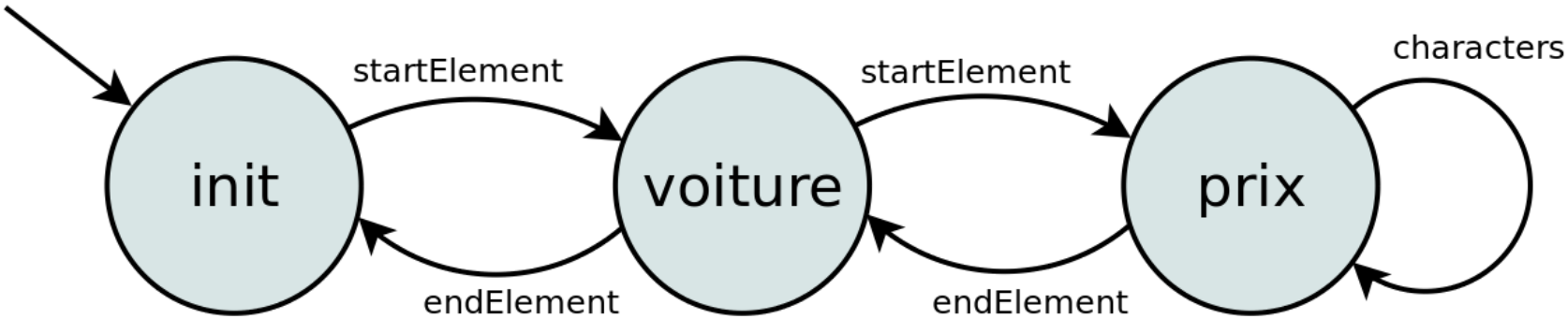
On va utiliser ce dispositif dans le *Handler* SAX pour coller à la structure du fichier XML et mémoriser les informations nécessaires pour les traitements. Les états représentent les éléments du document XML et les transitions seront provoquées par les événements SAX.

Pierre Nerzic

Application à l'analyse SAX

Voici la machine qu'on pourrait définir pour gérer le document XML des voitures.

```
<voiture id="871"><prix monnaie="yen">331212</prix></voiture>
```



NB: toutes les transitions possibles ne sont pas dessinées, ex: `characters` dans les états INIT et VOITURE.

Traitements des transitions

Alors par exemple, voici une partie du traitement de l'événement `startElement` :

```
private float m_TauxChange;
public void startElement(..., String qName, Attributes attrs) {
    switch (m_EtatCourant) {
    ...
    case VOITURE:
        if (qName.equals("prix")) {
            // calculer le taux de change s'il y a une monnaie
            String monnaie = attrs.getValue("monnaie");
            if ("yen".equals(monnaie)) m_TauxChange = 0.0082f;
            else m_TauxChange = 1.0f;
            m_EtatCourant = Etat.PRIX;
        } else throw new SAXException("balise inattendue ici");
        break;
```


Pierre Nerzic

Traitements des transitions (suite)

Et voici le traitement des événements `characters`. On se contente de mémoriser le texte dans une variable globale. Le traitement de ce texte sera fait dans l'événement `endElement` (c'est un choix personnel, on pourrait faire autrement).

```
private String m_Texte;

public void characters(char[] text, int debut, int lng)
{
    m_Texte = new String(text, debut, lng);
}
```

- Notez que `m_EtatCourant` ne change pas : on reste dans le même état comme c'est défini dans le schéma.
- On pourrait/devrait concaténer tous les textes qui arrivent successivement, voir le transparent suivant.

Concaténation de tous les textes

Lorsqu'un élément contient plusieurs types de textes, comme :

```
<info>texte1<![CDATA[texte2]]>texte3&ent;texte5</info>
```

Ça va générer plusieurs événements `characters` distincts. Alors on doit concaténer les morceaux ainsi :

```
public void characters(char[] text, int debut, int lng)
{
    m_Texte.concat(new String(text, debut, lng));
}
```

Et il faut penser à réinitialiser `m_Texte` à chaque élément, dans `startElement` et dans `endElement`.

Voir en TP, l'utilisation de la classe `StringBuilder`.

Traitements des transitions (fin)

Pour finir, voici le traitement de l'événement `endElement`. C'est lui qui affiche le prix en € :

```
public void endElement(..., String qName) {
    switch (m_EtatCourant) {
        case INIT:      break;
        case VOITURE:   m_EtatCourant = Etat.INIT; break;
        case PRIX:
            float prix = Float.valueOf(m_Texte) * m_TauxChange;
            System.out.println("prix = "+prix+" €");
            m_EtatCourant = Etat.VOITURE;
            break;
        default:
    }
}
```

NB: cet exemple se limite à de l'affichage, mais voir mieux en TP.

API XMLWriter de PHP

Pierre Nerzic

Présentation

L'API XMLWriter pour PHP ressemble énormément à ce qu'on vient de voir, sauf qu'elle sert à créer un document XML. Voici un court extrait pour vous convaincre :

```php
$writer = new XMLWriter();
$writer->openURI('php://output');
$writer->startDocument('1.0');
    $writer->startElement('voiture');
        $writer->startElement('prix');
            $writer->writeAttribute('monnaie', 'yen');
            $writer->text('331212');
        $writer->endElement();
    $writer->endElement();
$writer->endDocument();
$writer->flush();
```

L'indentation permet de vérifier visuellement la structure.

Ouverture du flux de sortie

Le script PHP doit produire un document XML en sortie. Voici le début typique d'un tel script :

```php
<?php
header("Content-Type: text/xml");
$writer = new XMLWriter();
$writer->openURI('php://output');
```

L'entête définit la nature des données émises par le script PHP. Ensuite, on crée un écrivain redirigé vers la sortie du script (c'est le même flux que echo et print).

Le script PHP se termine par :

```php
$writer->flush();
?>
```


Pierre Nerzic

Écriture d'éléments

L'API est très riche (documentation). Quelques fonctions utiles :

- `startDocument(version, encodage, standalone)` : écrit le prologue XML du document avec les paramètres optionnels fournis
- `endDocument()` : clôture le document
- `writeElement(nom, contenu)` : écrit un petit élément `<nom>contenu</nom>`
- `startElement(nom)` : écrit le début d'une balise ouvrante `<nom>`. On peut ensuite rajouter des attributs et un contenu
- `writeAttribute(nom, valeur)` : rajoute l'attribut `nom="valeur"` à l'élément actuellement ouvert
- `text(texte)` : écrit le texte, il est rajouté à l'élément courant
- `endElement()` : écrit la balise fermante `</nom>`.

Pierre Nerzic