

Programmation SAX

Concepts et Technologies XML

Master pro ILI

Année 2007-08

- ▶ **DOM** *Document Object Model*
- ▶ **SAX** *Simple API for XML*

DOM :

- ▶ charge en mémoire un document XML
- ▶ permet *de se promener dans l'arbre*

Avantages :

- ▶ possibilité d'expression, transformation, ...

Inconvénients :

- ▶ très coûteux

SAX :

- ▶ fonctionne sur le mode **Producteur-Consommateur**
- ▶ le producteur lit le document XML et envoie des événements dans un flux
- ▶ le consommateur récupère les évènements

SAX

Avantages :

- ▶ lecture linéaire,
- ▶ peu coûteux
- ▶ simple

Inconvénients :

- ▶ au programmeur de construire les structures de données *ad hoc* (si nécessaire)

- ▶ API proposées dans un grand nombre de langages
- ▶ API DOM pour JAVA : package org.w3c.dom
- ▶ API SAX pour JAVA : package org.xml.sax

- ▶ `org.xml.sax` contient les éléments principaux : une dizaine d'interfaces et deux classes concrètes.
- ▶ `org.xml.sax.helpers` fournissent des classes concrètes d'application des interfaces de `org.xml.sax`.
- ▶ `org.xml.sax.ext` extensions permises depuis SAX2.

Un exemple simple

```
public class Exemple{
    // args[0] contient une URI absolue d'un document XML
    public static void main(String[] args){
        // le producteur : un parseur SAX
        XMLReader prod ;
        // le consommateur : l'appli proprement dite !
        DefaultHandler cons ;

        try {
            //obtenir un parser
            prod = SAXParserFactory.getXMLReader() ;
        } catch (SAXException e){
            System.err.println("pbe de configuration") ;
            return ;
        }
    }
}
```



```
try {  
    //et un consommateur qui ne fait rien  
    cons = new DefaultHandler();  
    prod.setContentHandler(cons);  
    prod.setErrorHandler(cons);  
} catch (Exception e){  
    Sytem.out.println("il y a eu un souci");  
    return;  
}
```

Un exemple simple

```
// lancement du traitement
// (le parse)
try {
    prod.parse(args[0]);
} catch(IOException e){
    System.out.println("pbe d'entrée-sortie");
    return;
} catch(SAXException e){
    System.out.println("pbe du parseur");
    return;
}
```

```
// les imports nécessaires
import java.io.* ;
import org.xml.sax.* ;
import org.xml.sax.helpers.DefaultHandler ;
import javax.xml.parsers.SAXParserFactory ;
import javax.xml.parsers.ParserConfigurationException ;
import javax.xml.parsers.SAXParser ;
```

`DefaultHandler` est une classe concrète qui implémente

- ▶ l'interface `ContentHandler` : contient toutes les méthodes relatives au contenu du document
- ▶ l'interface `ErrorHandler` : pour recevoir les messages d'erreur (avertissements, erreurs, erreurs fatales. . .)

L'interface ContentHandler

```
interface ContentHandler {  
    void startDocument();  
    void endDocument();  
    void startElement(String uri, String localName,  
        String qName, Attributes atts);  
    void endElement(String uri, String localName,  
        String qName);  
    void characters(char[] ch, int start, int length);  
    void ignorableWhitespace(char[] ch,  
        int start, int length);  
}
```

```
interface Attributes {  
    int getLength();  
    String getValue(String qName);  
    String getValue(int index);  
    String getQName(int index);  
    int getIndex(String qName);  
}
```

Un exemple simple

```
public class MonContentHandler extends DefaultHandler {
    private boolean ok = false;
    public void startElement(String namespaceURI,
        String localName, String qName, Attributes atts){
        if (qName.equals("nom")) {
            System.out.print("Salut ");
            ok = true;
        }
    }
    public void endElement(String namespaceURI,
        String localName, String qName){
        if (qName.equals("nom")) {
            ok = false;
            System.out.println();
        }
    }
}
```

Un exemple simple

```
public void characters(char[] ch,
    int start, int length){
    if (ok)
        System.out.print(new String(ch,start,length)) ;
}

public void endDocument(){
    System.out.println("Au revoir!");
}
}
```

Une proposition :

- ▶ créer une hiérarchie de classes avec un mapping classe/élément
- ▶ le **consommateur** principal retransmet les informations à l'instance de la classe concernée

Un exemple

```
abstract class Element{
    private final String tagName ;
    public Element(String tagName){
        this.tagName = tagName ;
    }
    public void startElement(Attributes attributes){
    }
    public void characters(String s){
    }
    public void endElement(){
    }
    public String tagName(){
        return tagName ;
    }
}
```

On ajoute une hiérarchie de classes :

- ▶ Nom hérite de Element
- ▶ Prenom hérite de Element
- ▶ Age hérite de Element
- ▶ ...

```
class MonDoc extends DefaultHandler {
    /** pile des éléments rencontrés */
    private Vector<Element> parents = new Vector<Element>();
    private Map<String, Element> lesElts;

    public MonDoc() {
        lesElts = new HashMap<String, Element>();
        lesElts.put("nom", new Nom("nom"));
        lesElts.put("prenom", new Prenom("prenom"));
        lesElts.put("age", new Age("age"));
    }
}
```

```
public void startElement(String uri, String localName,  
                        String qName, Attributes atts) {  
    Element current = theElts.get(qName);  
    parents.add(current);  
    current.startElement(atts);  
}  
}
```

```
public void characters(char[] ch, int start,
                    int length) {
    parents.lastElement().characters
        (new String(ch, start, length));
}
```

```
public void endElement(String uri, String localName,
                    String qName) {
    Element current = theElts.get(qName);
    current.endElement();
    parents.remove(parents.size() - 1);
}
```