

DOM

&

SAX

L'analyse XML

DOM

DOM levels, DOM level 1
Principes de l'API
Objets DOM
Traitement des blancs
Navigation, parcours, et mise à jour de l'arbre
Attributs et entités dans le DOM
Héritage des objets, hiérarchie des nœuds
Clonage des nœuds, échange des nœuds
Les espaces de nommage dans le DOM
Spécialisation des APIs DOM

SAX

SAX levels
Comment fonctionne SAX
Principaux *handlers*
L'interface *ContentHandler*
Enregistrement d'un handler
Exemple
Événements caractères
Filtres et pipelines SAX
Analyseurs SAX validants
Les espaces de nommage dans SAX

Comparaison de SAX et DOM

Quand utiliser SAX ou DOM ?

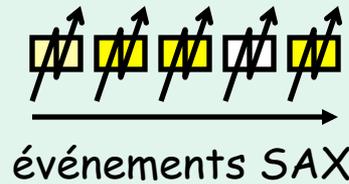
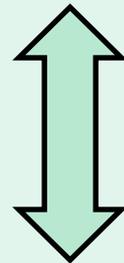
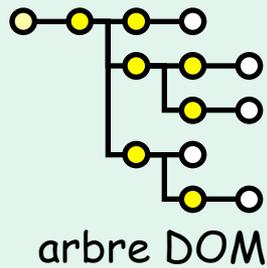
Autre APIs

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Cours SYSTEM "Cours.dtd">
<Cours>
  <Titre>Cours XML</Titre>
  <Auteur>
    <Nom>Poulard</Nom>
    <Prénom>Philippe</Prénom>
  </Auteur>
  <Description>Ce cours aborde les concepts
    de base mis en œuvre dans <b>XML</b>
  </Description>
</Cours>

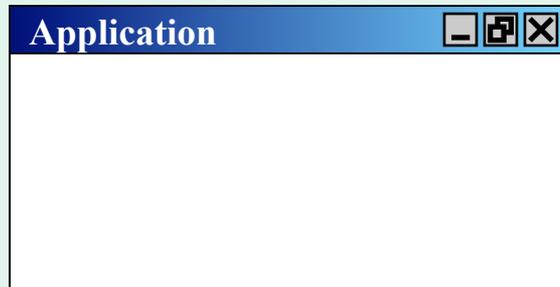
```

Structure physique



Modèle logique (Infoset)

Accède au modèle logique du document grâce aux APIs DOM ou SAX

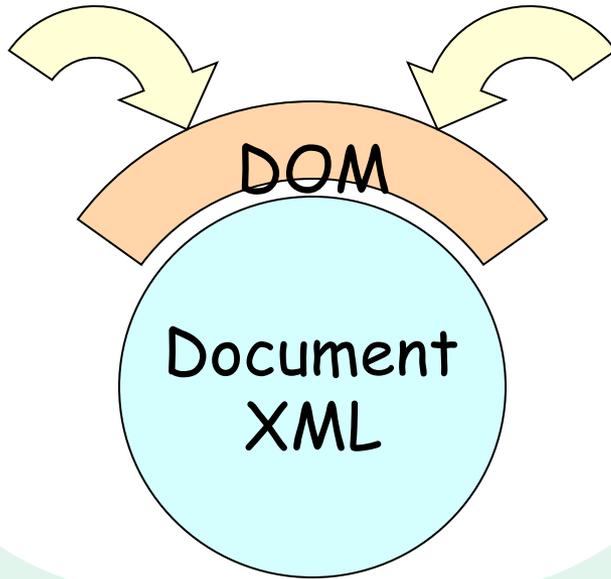


Application

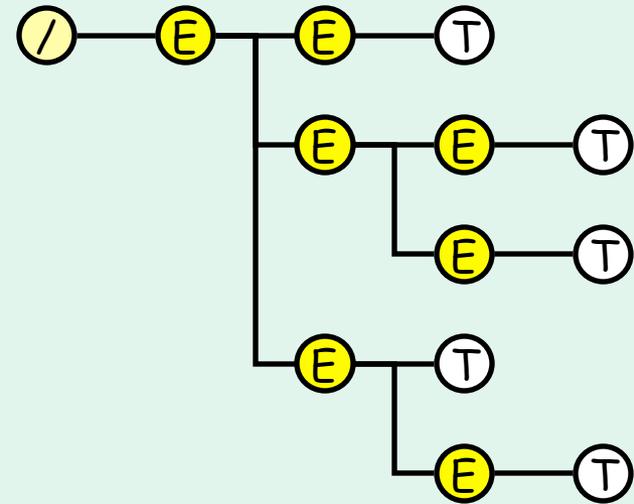
Document Object Model

DOM, norme du W3C

Modèle Object du Document :
une API pour **accéder** et **agir** sur
le contenu et la structure d'un document



Le document est vu comme un arbre
Sa représentation ne l'est pas nécessairement
(DOM spécifie une API, pas une implémentation)



DOM level 3

Schémas, XPath, entrées/sorties

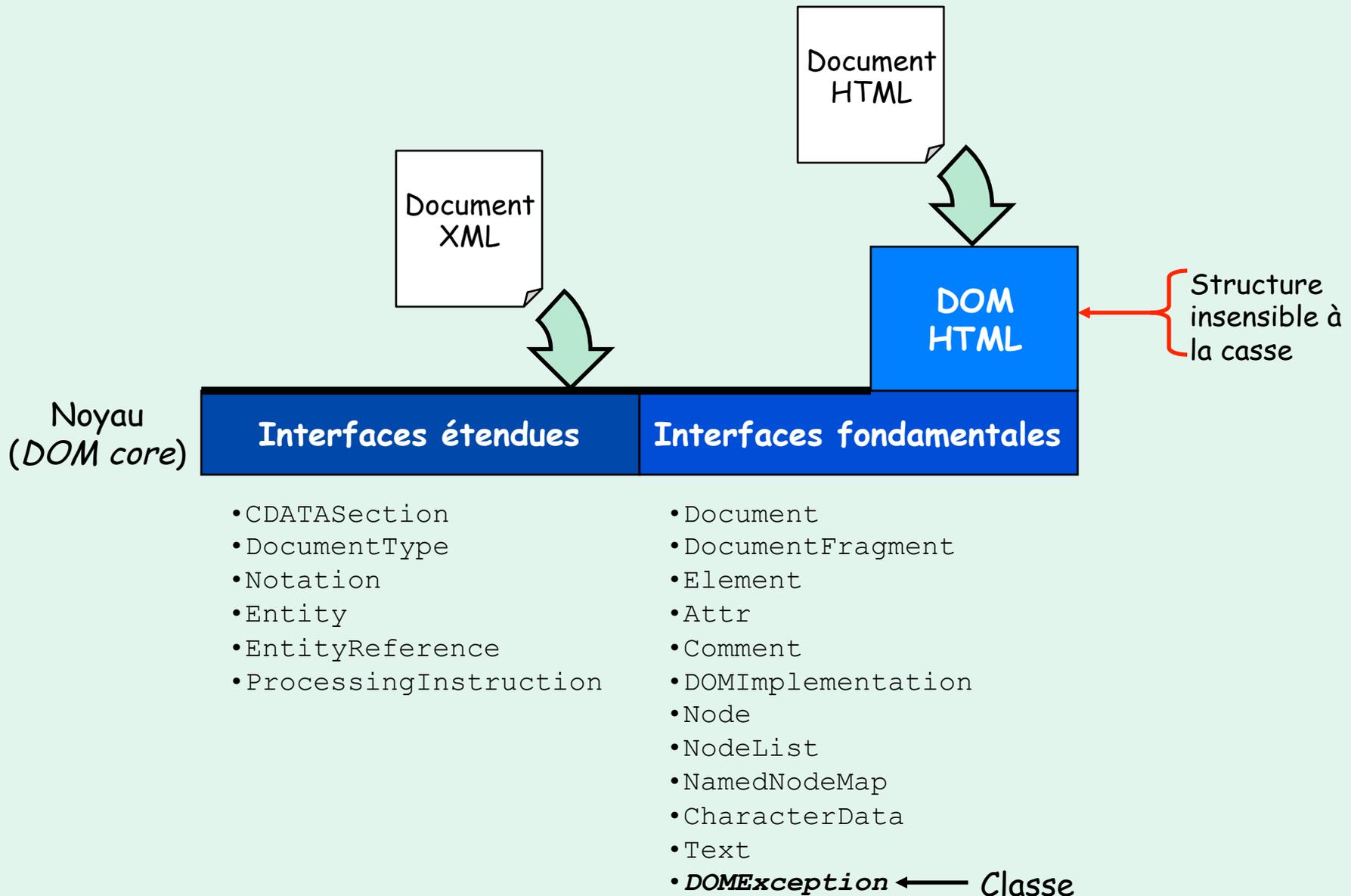
DOM level 2

Namespaces, vues, événements, styles, parcours

DOM level 1

Noyau, HTML

Pas de
compatibilité
binaire entre
différentes
implémentations



Le modèle d'objet spécifié par le W3C définit 12 types de nœuds différents.

Le modèle d'objet de document fournit toute une panoplie d'outils destinés à **construire et manipuler un document XML**. Pour cela, le DOM met à disposition des interfaces, des méthodes et des propriétés permettant de gérer l'ensemble des composants présents dans un document XML.

Le DOM spécifie diverses méthodes et propriétés permettant notamment, de **créer** (*createNode...*), **modifier** (*replaceChild...*), **supprimer** (*remove...*) ou d'**extraire des données** (*get...*) de n'importe quel élément ou contenu d'un document XML.

De même, le DOM définit les types de relation entre chaque nœud, et des directions de déplacement dans une arborescence XML. Les propriétés *parentNode*, *childNodes*, *firstChild*, *lastChild*, *previousSibling* et *nextSibling* permettent de retourner respectivement le père, les enfants, le premier enfant, le dernier enfant, le frère précédent et le frère suivant du nœud courant.

Le modèle d'objet de document offre donc au programmeur les moyens de traiter un document XML dans sa totalité

Exemple

- Vues héritées → Hiérarchie d'objet
- Vues aplaties → Tout est Node

```
Element.tagName
=
Node.nodeName
```

Si le nœud
est un
élément

Les implémentations sont libres d'utiliser ou non des accesseurs (`get`) et des mutateurs (`set`) pour protéger les propriétés

```
MSXML → theName = node.nodeName;
Java   → theName = node.getNodeName();
```

DOM spécifie une API à minima. Les implémentations sont libres d'étendre les fonctionnalités

```
MSXML → resultHTML = source.transformNode(stylesheet);
```

DOM level 2

Introduction des espaces de nommage : méthodes postfixées par `NS`

```
node.setAttributeNS(null, attr, value);
```

→ Default namespace

Node Représente un nœud de l'arborescence d'un document XML.

Propriétés :

`attributes`, `childNodes`, `firstChild`, `lastChild`, `localName`, `namespaceURI`,
`nextSibling`, `nodeName`, `nodeType`, `nodeValue`, `ownerDocument`, `parentNode`, `prefix`,
`previousSibling`

Méthodes :

`appendChild`, `cloneNode`, `hasAttributes`, `hasChildNodes`, `insertBefore`, `isSupported`,
`normalize`, `removeChild`, `replaceChild`

L'interface `Node` définit des constantes qui permettent de connaître le type du nœud

`nodeType`

```

ELEMENT_NODE = 1
ATTRIBUTE_NODE = 2
TEXT_NODE = 3
CDATA_SECTION_NODE = 4
ENTITY_REFERENCE_NODE = 5
ENTITY_NODE = 6
PROCESSING_INSTRUCTION_NODE = 7
COMMENT_NODE = 8
DOCUMENT_NODE = 9
DOCUMENT_TYPE_NODE = 10
DOCUMENT_FRAGMENT_NODE = 11
NOTATION_NODE = 12

```

`nodeName`

```

Nom d'élément
Nom de l'attribut
"#text"
"#cdata-section"
Nom de l'entité référencée
Nom de l'entité
Cible
"#comment"
"#document"
Nom du type de document
"#document-fragment"
Nom de notation

```

`nodeValue`

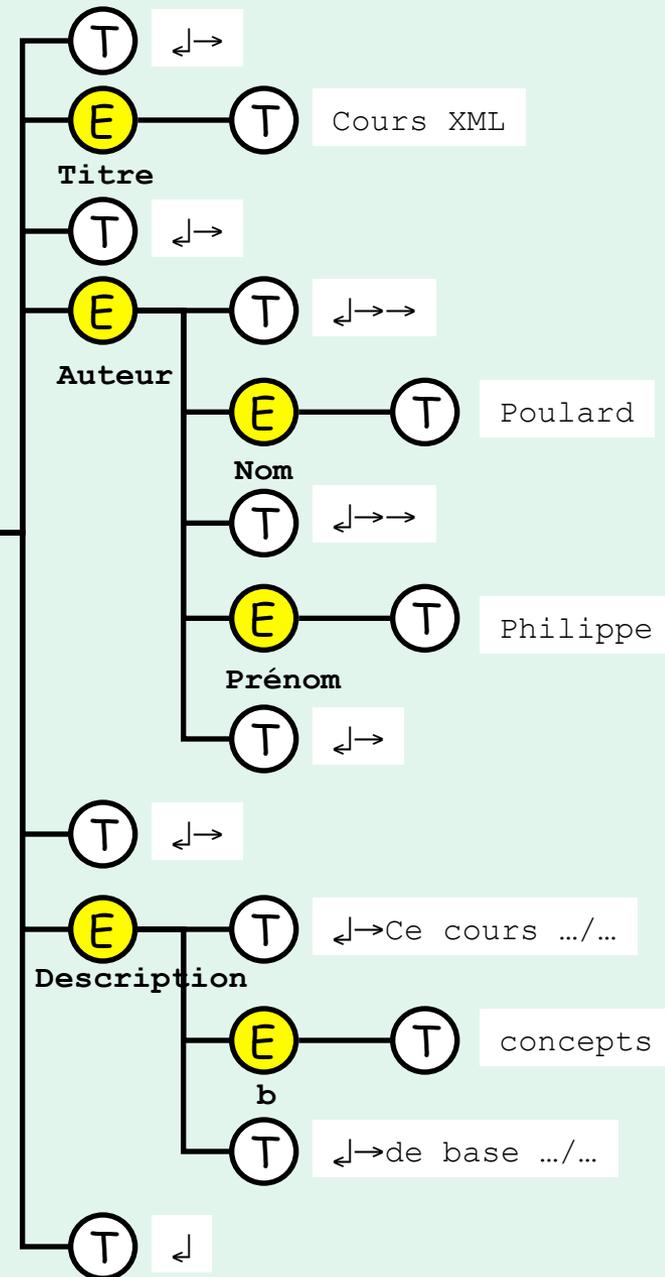
```

null
Valeur de l'attribut
Contenu du nœud du texte
Contenu de la section CDATA
null
null
Contenu complet, sans la cible
Contenu du commentaire
null
null
null
null

```

`nodeType` permet d'obtenir le type de nœud

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
  → <Titre>Cours XML</Titre>
  → <Auteur>
    → <Nom>Poulard</Nom>
    → <Prénom>Philippe</Prénom>
  → </Auteur>
  → <Description>
    → Ce cours aborde les <b>concepts</b>
    → de base mis en œuvre dans XML.
  → </Description>
</Cours>
```

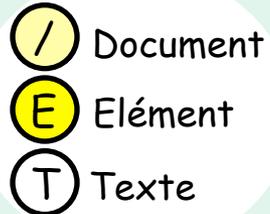


! Les séquences de caractère qui ne contiennent que des blancs (espaces, tabulations, interlignes) génèrent des nœuds de texte, même si la DTD spécifie le contraire.

```
<!ELEMENT Auteur (Nom, Prénom)>
```

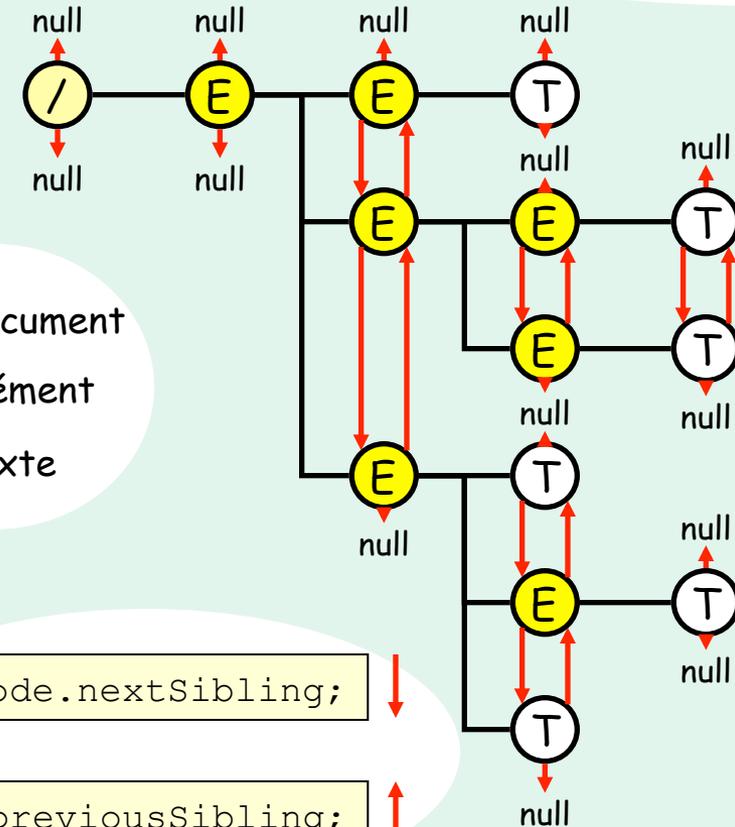
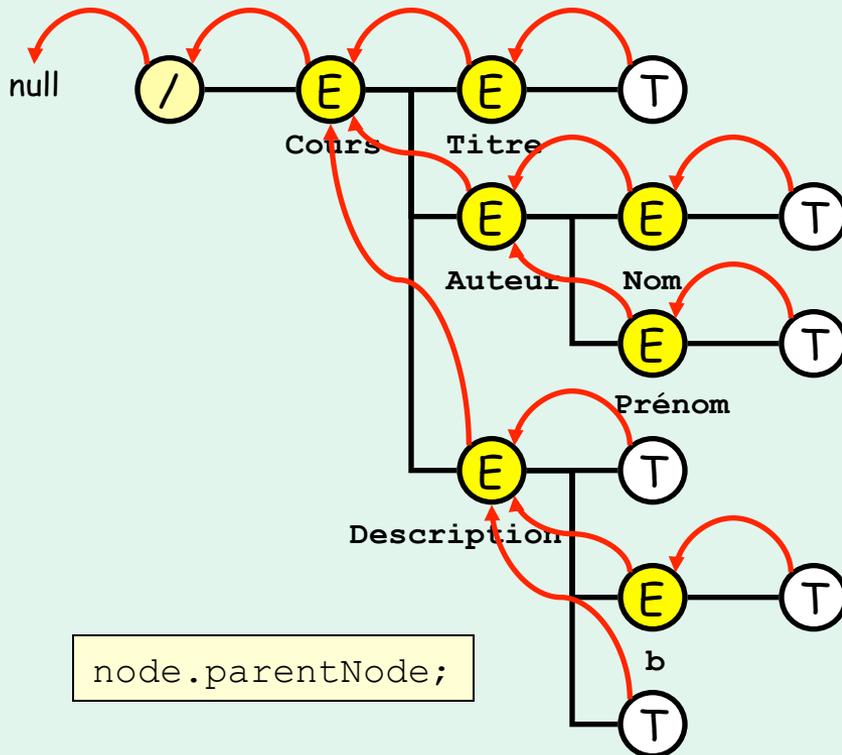
i Il est possible d'infléchir ce comportement grâce aux options du parseur.

! Le comportement par défaut de certains parseurs est d'éliminer les nœuds blancs.



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
  <Titre>Cours XML</Titre>
  <Auteur>
    <Nom>Poulard</Nom>
    <Prénom>Philippe</Prénom>
  </Auteur>
  <Description>
    Ce cours aborde les <b>concepts</b> de
    base mis en œuvre dans XML.
  </Description>
</Cours>
```

```
Node#parentNode;
Node#hasChildNodes();
Node#firstChild;
Node#lastChild;
Node#previousSibling;
Node#nextSibling;
```



NodeList

Représente une collection de nœuds ordonnés.

Propriétés :

`length`

Méthodes :

`item`

En Java : pas d'utilisation du « collection framework » (`java.util.*`)
listes (`List`), tableaux associatifs (`Map`) ☹

Parcours des éléments de niveau inférieur immédiat

```
for (Node child = node.getFirstChild();
     child != null;
     child = child.getNextSibling()) {
    // do something
}
```

ou

```
NodeList nodeList = node.getChildNodes();
for (int i = 0;
     i < nodeList.getLength();
     i++) {
    Node child = nodeList.item(i);
    // do something
}
```

Parcours récursif

```
public void processNodeRecursively(Node node) {
    // do something
    for (Node child = node.getFirstChild();
         child != null;
         child = child.getNextSibling()) {
        processNodeRecursively(child);
    }
}
```



DOM level 2

traversal

```
DocumentTraversal t;
NodeIterator ni;
Node child;
t = (DocumentTraversal) node.getOwnerDocument();
ni = t.createNodeIterator(node, NodeFilter.SHOW_ALL, null, false);
while ((child = iterator.nextNode()) != null) {
    // do something
}
ni.detach();
```



Document

Représente le document XML ou HTML.

Propriétés :

`doctype`, `documentElement`, `implementation`
`attributes`, `childNodes`, `firstChild`, `lastChild`, `localName`, `namespaceURI`,
`nextSibling`, `nodeName`, `nodeType`, `nodeValue`, `ownerDocument`, `parentNode`, `prefix`,
`previousSibling`

Méthodes :

`createAttribute`, `createAttributeNS`, `createCDATASection`, `createComment`,
`createDocumentFragment`, `createElement`, `createElementNS`, `createEntityReference`,
`createProcessingInstruction`, `createTextNode`, `getElementById`,
`getElementsByTagName`, `getElementsByTagNameNS`, `importNode`
`appendChild`, `cloneNode`, `hasAttributes`, `hasChildNodes`, `insertBefore`, `isSupported`,
`normalize`, `removeChild`, `replaceChild`



Element

Représente un nœud d'élément dans un document XML.

Propriétés :

tagName

attributes, childNodes, firstChild, lastChild, localName, namespaceURI, nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix, previousSibling

Méthodes :

getAttribute, getAttributeNS, getAttributeNode, getAttributeNodeNS, getElementsByTagName, getElementsByTagNameNS, hasAttribute, hasAttributeNS, removeAttribute, removeAttributeNS, removeAttributeNode, setAttribute, setAttributeNS, setAttributeNode, setAttributeNodeNS, appendChild, cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported, normalize, removeChild, replaceChild

Spécialiser le nœud quand on a besoin d'un type particulier :

```
if ( node.getNodeType() == Node.ELEMENT_NODE ) {
    Element el = (Element) node;
    Attr fooAttr = el.getAttributeNode("foo");
}
```

Un élément n'a pas de valeur : `element.getNodeValue()` retourne toujours `null`.
Un élément a des nœuds enfants.

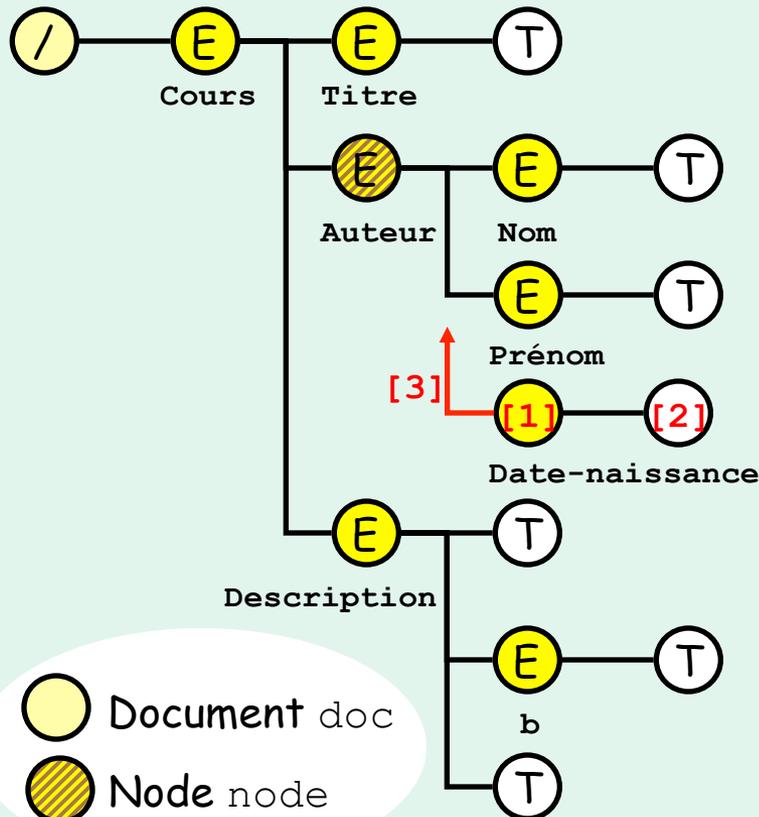
Un nœud créé fait partie du document mais n'est pas attaché à l'arbre.

Les ajouts se font donc en 2 étapes :

- création
- ajout

```
Document#createXXX()
Node#insertBefore(newChild, refChild)
Node#appendChild(newChild)
Node#replaceChild(newChild, refChild)
Node#removeChild(oldChild)
```

```
[1] Element newChild = doc.createElement("Date-naissance");
[2] newChild.appendChild(doc.createTextNode("10-06-1969"));
[3] node.appendChild(newChild);
```



 Les objets sont réactifs aux modifications de l'arbre qui peuvent avoir des effets de bord inattendus.

```
for (Node child = node.getFirstChild();
     child != null;
     child = child.getNextSibling()) {
    node.removeChild(child);
}
```

Après suppression du premier nœud de `node`, `child.getNextSibling()` retourne `null`, et les autres nœuds de `node` ne seront pas supprimés.

Pour supprimer tous les fils d'un nœud :

```
while (node.hasChildNodes()) {
    node.removeChild(node.getFirstChild());
}
```



DocumentType

Représente la déclaration de type de document indiqué par la balise `<!DOCTYPE>`

Propriétés :

`entities`, `internalSubset`, `name`, `notations`, `publicId`, `systemId`
`attributes`, `childNodes`, `firstChild`, `lastChild`, `localName`, `namespaceURI`,
`nextSibling`, `nodeName`, `nodeType`, `nodeValue`, `ownerDocument`, `parentNode`, `prefix`,
`previousSibling`

Méthodes :

`appendChild`, `cloneNode`, `hasAttributes`, `hasChildNodes`, `insertBefore`, `isSupported`,
`normalize`, `removeChild`, `replaceChild`

DocumentFragment

Représente une partie de l'arborescence d'un document. Ce fragment pouvant ne pas être bien formé, est utilisé généralement pour des opérations d'insertion.

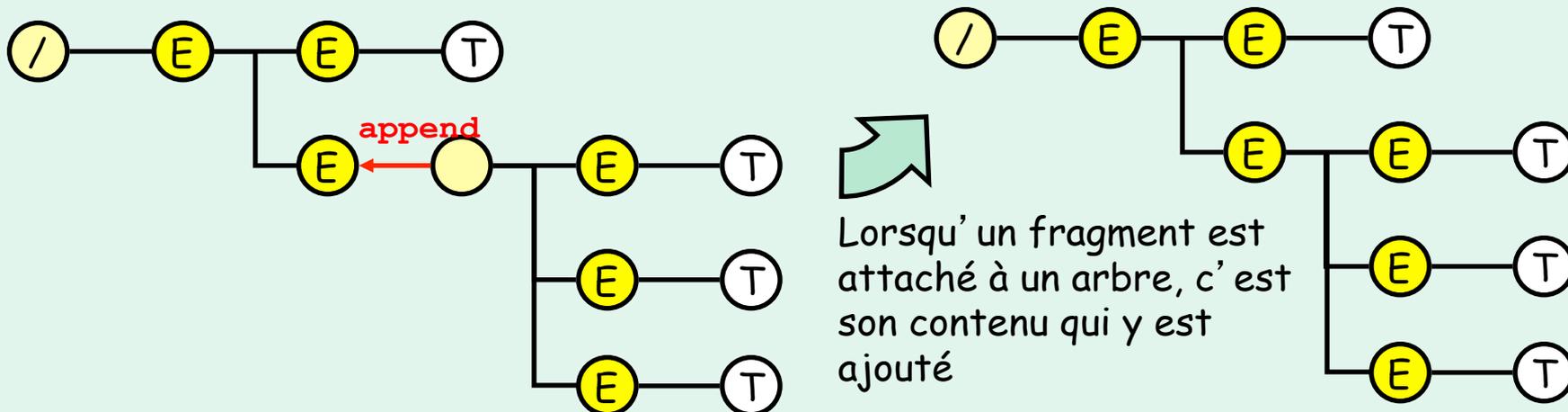
Propriétés :

`attributes`, `childNodes`, `firstChild`, `lastChild`, `localName`, `namespaceURI`,
`nextSibling`, `nodeName`, `nodeType`, `nodeValue`, `ownerDocument`, `parentNode`, `prefix`,
`previousSibling`

Méthodes :

`appendChild`, `cloneNode`, `hasAttributes`, `hasChildNodes`, `insertBefore`, `isSupported`,
`normalize`, `removeChild`, `replaceChild`

C'est un "espace de travail" pratique...



Lorsqu'un fragment est attaché à un arbre, c'est son contenu qui y est ajouté

**Attr**Représente un attribut d'un objet *Element*.

Propriétés :

```
name, ownerElement, specified, value  
attributes, childNodes, firstChild, lastChild, localName, namespaceURI,  
nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix,  
previousSibling
```

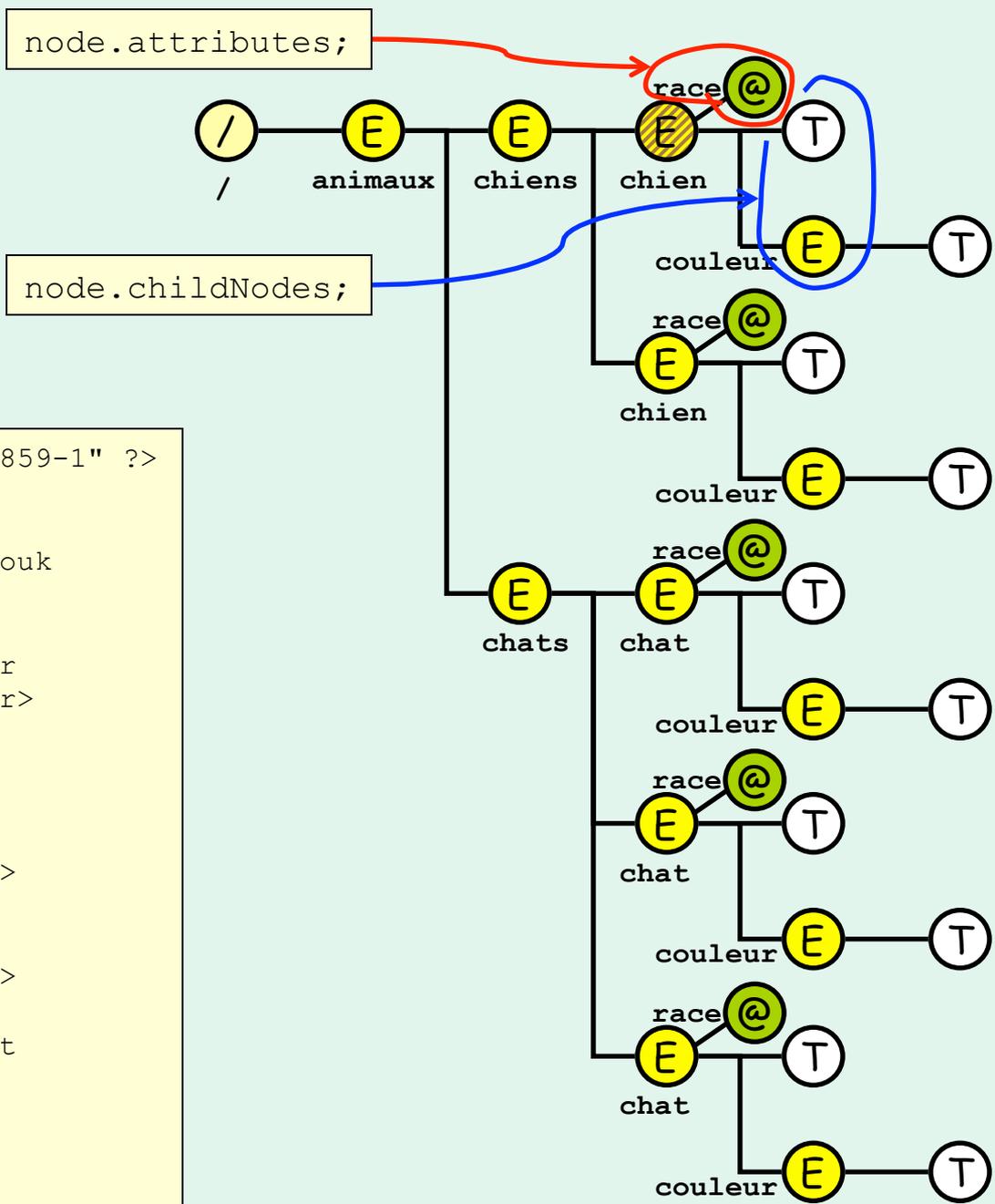
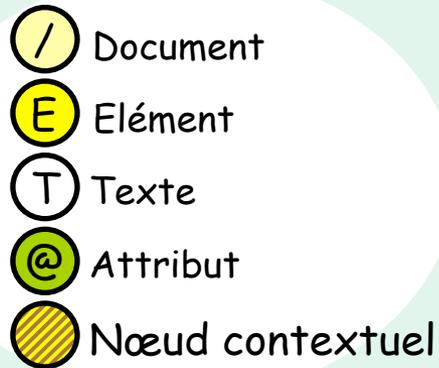
Méthodes :

```
appendChild, cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported,  
normalize, removeChild, replaceChild
```

Les attributs sont des `Nodes`, mais pas connectés à l'arbre par une dépendance hiérarchique.



(Pour un `Node` élément, `Node#childNodes` ne retourne pas d'attributs, il faut utiliser `Node#attributes`)



```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<animaux>
  <chiens>
    <chien race="Labrador">Mabrouk
      <couleur>noir</couleur>
    </chien>
    <chien race="Labrador">Médor
      <couleur>marron</couleur>
    </chien>
  </chiens>
  <chats>
    <chat race="Siamois">Félix
      <couleur>crème</couleur>
    </chat>
    <chat race="Birman">Tom
      <couleur>crème</couleur>
    </chat>
    <chat race="Abyssin">Rominet
      <couleur>gris</couleur>
    </chat>
  </chats>
</animaux>
  
```

```
NamedNodeMap Element#attributes
String Element#getAttribute(String name)
void Element#setAttribute(String name, String value)
void Element#removeAttribute(String name)
Attr Element#getAttributeNode(String name)
void Element#setAttributeNode(Attr attr)
void Element#removeAttributeNode(Attr attr)
```

```
NamedNodeMap map = elem.getAttributes();
for (int i = 0; i < map.getLength(); i++) {
    Attr attr = (Attr) map.item(i);
    // do something
}
```



Si l'attribut n'existe pas :

```
String value = elem.getAttribute("style");
```

Retourne ""

```
Attr attr = elem.getAttributeNode("style");
```

Retourne null

CDATASection

Représente une section de données textuelles (Character Data Section).

Propriétés :

`data`, `length`, `attributes`, `childNodes`, `firstChild`, `lastChild`, `localName`, `namespaceURI`, `nextSibling`, `nodeName`, `nodeType`, `nodeValue`, `ownerDocument`, `parentNode`, `prefix`, `previousSibling`

Méthodes :

`appendData`, `deleteData`, `insertData`, `replaceData`, `substringData`, `appendChild`, `cloneNode`, `hasAttributes`, `hasChildNodes`, `insertBefore`, `isSupported`, `normalize`, `removeChild`, `replaceChild`, `splitText`

CharacterData

Est une extension de l'interface *Node* qui permet d'accéder aux données textuelles dans le modèle d'objet.

Propriétés :

data, length

attributes, childNodes, firstChild, lastChild, localName, namespaceURI, nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix, previousSibling

Méthodes :

appendData, deleteData, insertData, replaceData, substringData

appendChild, cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported, normalize, removeChild, replaceChild



Comment

Représente un commentaire dans un document XML ou HTML

```
<!-- Commentaire -->
```

Propriétés :

```
data, length, attributes, childNodes, firstChild, lastChild, localName, namespaceURI,  
nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix,  
previousSibling
```

Méthodes :

```
appendData, deleteData, insertData, replaceData, substringData, appendChild,  
cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported, normalize,  
removeChild, replaceChild
```

DOMImplementation

Fournit des méthodes qui sont indépendantes de n'importe quelles instances particulières du Modèle d'Objet du Document

Propriétés :

Aucune

Méthodes :

`createDocument, createDocumentType, hasFeature`



Entity

Représente une entité analysée ou non-analysée dans un document XML.

Propriétés :

```
<!ENTITY entité SYSTEM "blah-blah">
```

notationName, publicId, systemId

attributes, childNodes, firstChild, lastChild, localName, namespaceURI,
nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix,
previousSibling

Méthodes :

appendChild, cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported,
normalize, removeChild, replaceChild



EntityReference

Contient le nom de l'entité

```
&entité;
```

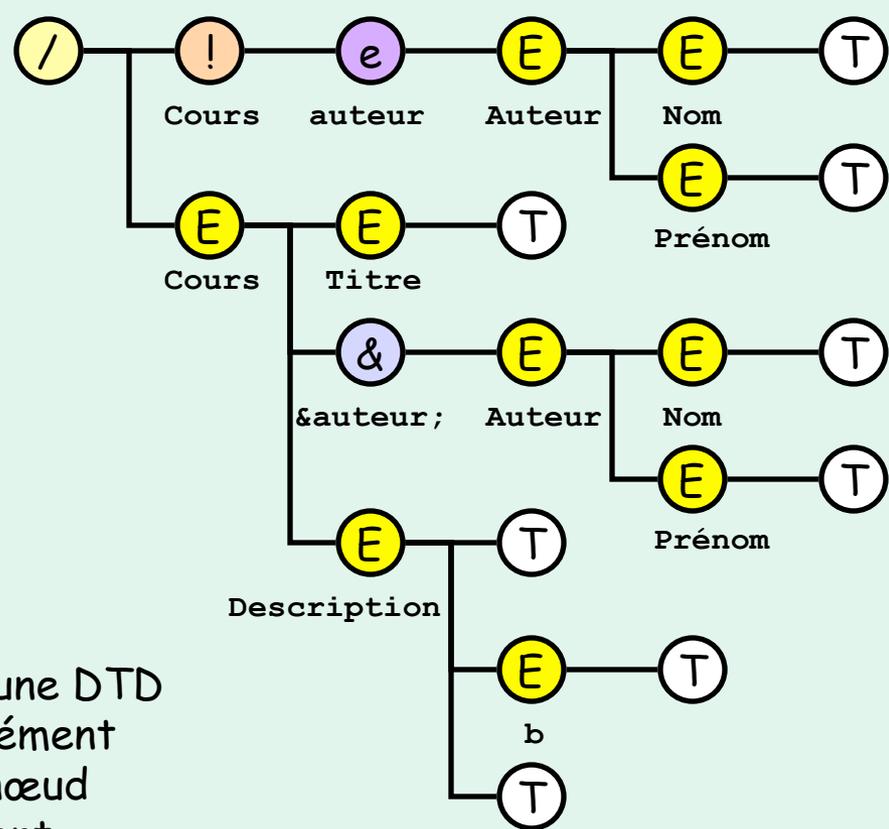
Propriétés :

```
attributes, childNodes, firstChild, lastChild, localName, namespaceURI,  
nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix,  
previousSibling
```

Méthodes :

```
appendChild, cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported,  
normalize, removeChild, replaceChild
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Cours [
  <!ENTITY auteur
    "<Auteur>
      <Nom>Poulard</Nom>
      <Prénom>Philippe</Prénom>
    </Auteur>">
]>
<Cours>
  <Titre>Cours XML</Titre>
  &auteur;
  <Description>
    Ce cours aborde les <b>concepts</b>
    de base mis en œuvre dans XML.
  </Description>
</Cours>
```



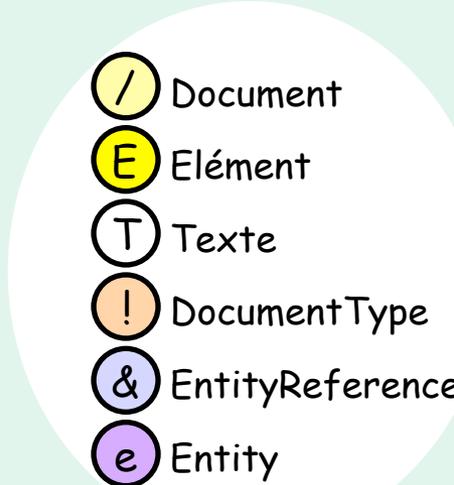
! Ce document pourrait être conforme à une DTD dans laquelle l'élément `Auteur` suivrait l'élément `Titre`. Pourtant, dans le DOM, le premier nœud suivant (de type `ELEMENT_NODE`) de l'élément `Titre` n'est pas l'élément `Auteur`.

i Certains parseurs proposent des options pour intégrer directement les contenus d'entités sans les références

DOM level 2

Les vues (*views*) permettent d'obtenir une représentation du document sans les références d'entités.

! Le premier objet d'un document n'est pas nécessairement un élément



NamedNodeMap

Représente des collections de nœuds qui peuvent être accédées par un nom.

Propriétés :

`length`

Méthodes :

`getNamedItem`, `getNamedItemNS`, `item`, `removeNamedItem`, `removeNamedItemNS`,
`setNamedItem`, `setNamedItemNS`

Notation

Représente une notation `<!NOTATION notation>` déclarée dans la DTD.

Propriétés :

publicId, systemId

`attributes, childNodes, firstChild, lastChild, localName, namespaceURI, nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix, previousSibling`

Méthodes :

`appendChild, cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported, normalize, removeChild, replaceChild`



ProcessingInstruction

Représente une instruction de traitement

```
<?Nom_Instruction Data?>
```

Propriétés :

data, target

`attributes, childNodes, firstChild, lastChild, localName, namespaceURI, nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix, previousSibling`

Méthodes :

`appendChild, cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported, normalize, removeChild, replaceChild`

Ⓣ **Text** Représente le contenu textuel d'un attribut ou d'un élément.

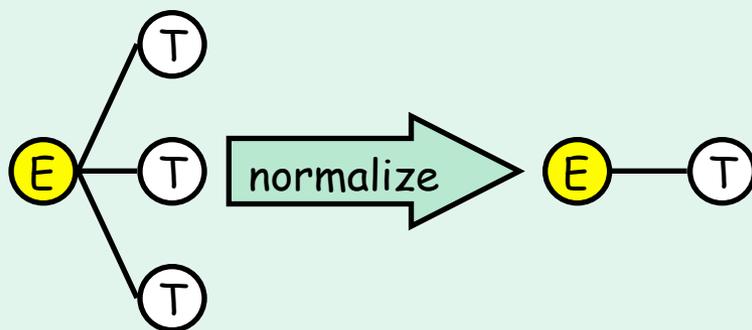
Propriétés :

`data`, `length`, `attributes`, `childNodes`, `firstChild`, `lastChild`, `localName`, `namespaceURI`, `nextSibling`, `nodeName`, `nodeType`, `nodeValue`, `ownerDocument`, `parentNode`, `prefix`, `previousSibling`

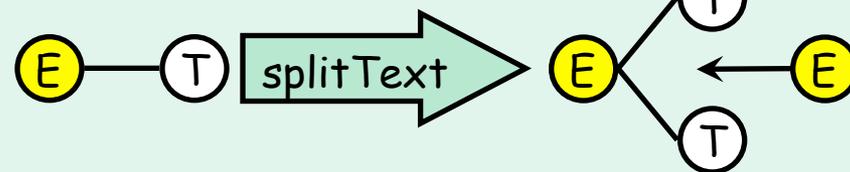
Méthodes :

splitText

`appendData`, `deleteData`, `insertData`, `replaceData`, `substringData`, `appendChild`, `cloneNode`, `hasAttributes`, `hasChildNodes`, `insertBefore`, `isSupported`, `normalize`, `removeChild`, `replaceChild`

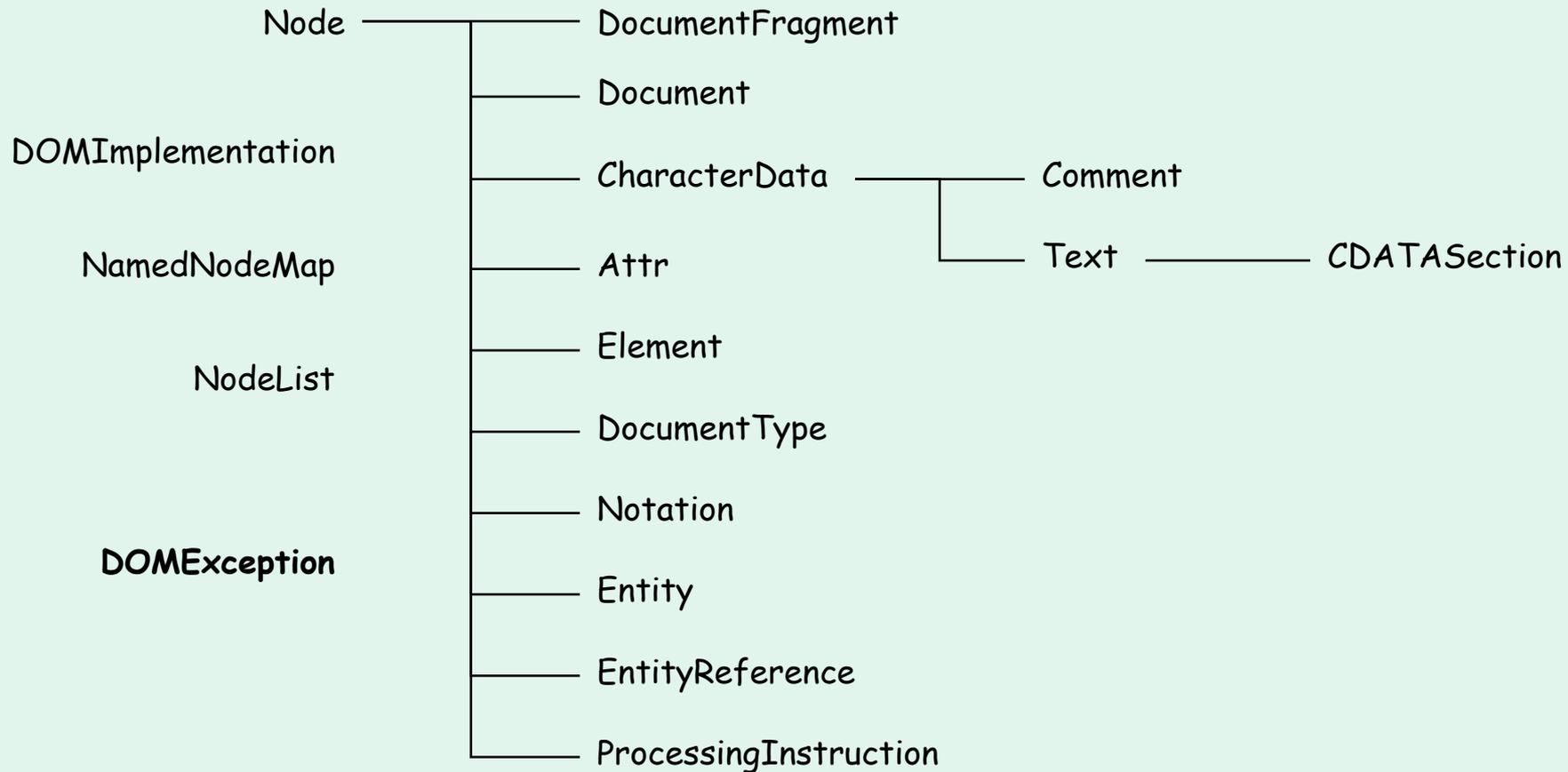


```
Text.splitText(offset);
```



Possibilité
d'inclusion d'un
élément au
milieu du texte

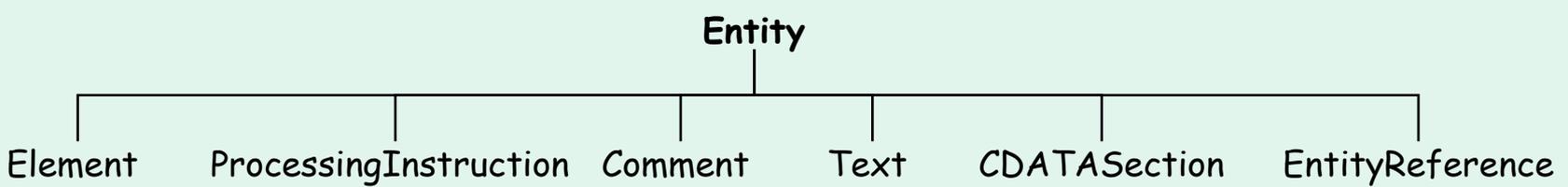
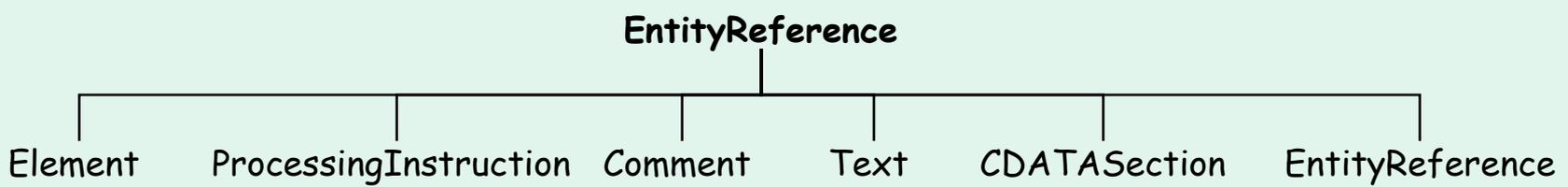
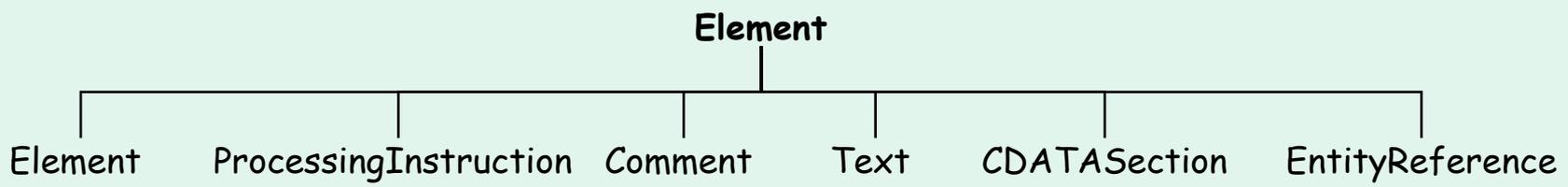
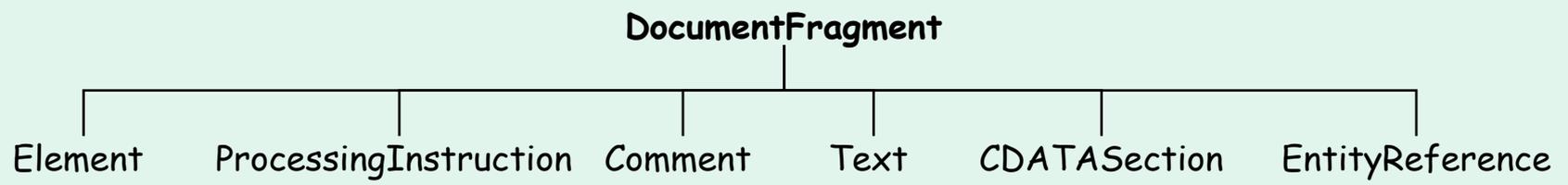
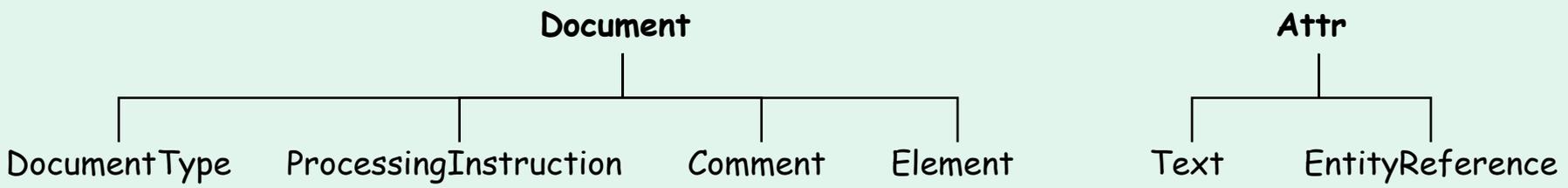
```
try {
    ...
    root.appendChild(document.createTextNode("Un peu"));
    root.appendChild(document.createTextNode(" "));
    root.appendChild(document.createTextNode("de texte"));
    document.getDocumentElement().normalize();
} catch (ParserConfigurationException pce) {
    ...
}
```



Pour obtenir les nœuds textuels, il faut penser à prendre en compte les sections CDATA

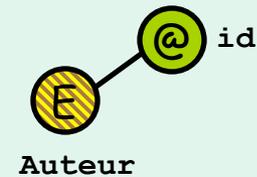
```

If (node.getNodeType() == Node.TEXT_NODE ||
    node.getNodeType() == Node.CDATA_SECTION_NODE) {
    // traitement des nœuds texte
}
  
```



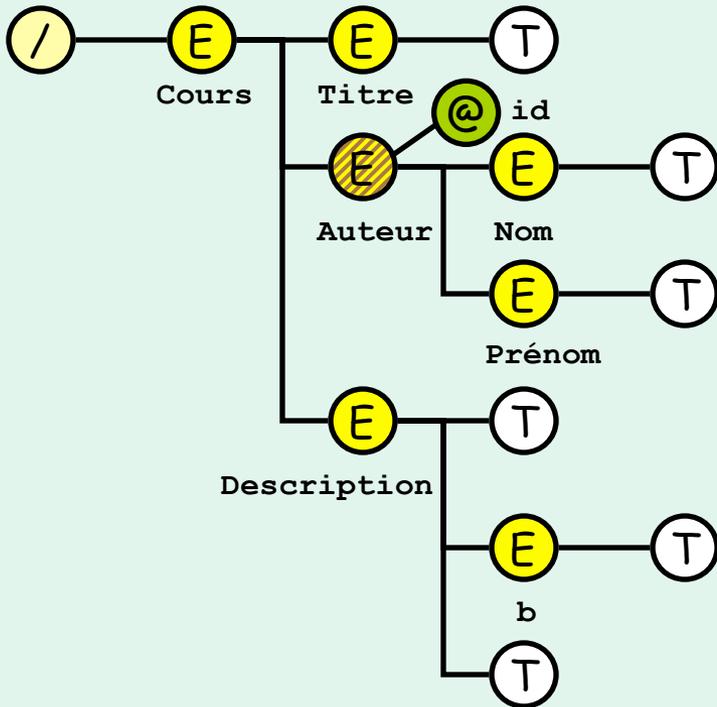
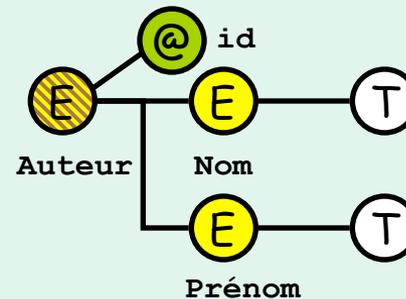
Clonage en surface

```
Element newElement = elem.cloneNode(false);
```



Clonage en profondeur

```
Element newElement = elem.cloneNode(true);
```



- Élément `elem`
- Élément `newElement`

```
Document Document#ownerDocument  
Node Document#importNode()
```

Les nœuds, même non attachés à l'arbre, appartiennent au document.

```
DocumentBuilder parser = .../...;  
Document doc1 = parser.parse(.../...);  
Document doc2 = parser.parse(.../...);  
.../...  
Element e11 = doc1.getDocumentElement();  
Element e12 = doc2.getDocumentElement();  
e11.appendChild(e12);
```



Non

Provoque une exception
DOMException

```
DocumentBuilder parser = .../...;  
Document doc1 = parser.parse(.../...);  
Document doc2 = parser.parse(.../...);  
.../...  
Element e11 = doc1.getDocumentElement();  
Element e12 = doc2.getDocumentElement();  
Node node = doc1.importNode(e12, true);  
e11.appendChild(node);
```



Oui

Importation correcte d'un
nœud

DOM level 2

Propriétés des `Element` et `Attr` : `namespaceURI`
`localName`
`prefix`

Vérifier que votre *parser* prend en compte les espaces de nommage.
 Le cas échéant, activer la fonctionnalité.



DOM ne réalise aucune déclaration d'espace de nommage automatiquement.

```
Element elem = doc.createElementNS(
    "http://www.foo.com",
    "foo:bar");
```

← URI de l'espace de nom

← Nom qualifié de l'élément

Crée un élément `bar` dont l'URI de l'espace de nom est `http://www.foo.com` et dont le préfixe est `foo`.

La déclaration de l'espace de nommage `xmlns:foo="http://www.foo.com"` n'est pas créée automatiquement.

Habituellement, on n'associe pas d'espaces de nommage aux attributs, puisqu'ils dépendent de leur élément parent. Lorsque ce n'est pas le cas, on peut les déclarer ainsi :

```
elem.setAttributeNS(
    "http://www.w3.org/1999/xlink",
    "xlink:type",
    "simple");
elem.setAttributeNS(
    "http://www.w3.org/1999/xlink",
    "xlink:href",
    "myDocument.html");
```

← URI de l'espace de nom

← Attribut

← Valeur

← URI de l'espace de nom

← Attribut

← Valeur

```
elem.setAttributeNS(
    "http://www.w3.org/2000/xmlns/",
    "xmlns:foo",
    "http://www.foo.com");
```

← URI de l'espace de nom xmlns
 ← Attribut
 ← Valeur

```
elem.setAttributeNS(
    "http://www.w3.org/2000/xmlns/",
    "xmlns:xlink",
    "http://www.w3.org/1999/xlink");
```

← URI de l'espace de nom xmlns
 ← Attribut
 ← Valeur

D'une manière générale :

```
ancestorElem.setAttributeNS(
    "http://www.w3.org/2000/xmlns/",
    "xmlns:" + elem.getPrefix(),
    elem.getNamespaceURI());
```

← URI de l'espace de nom xmlns
 ← Attribut
 ← Valeur

Les déclarations doivent se faire sur l'élément ou l'un de ses ancêtres.



Les déclarations d'espace de nommage (ou l'absence de déclaration) peuvent rendre le DOM inconsistant. Il appartient au programmeur de réaliser les déclarations idoines.

```
Element elem = doc.createElementNS(
    "http://www.foo.com", "foo:bar");
elem.setAttributeNS(
    "http://www.w3.org/2000/xmlns/", "xmlns:foo", "http://www.blah.com");
```



Le DOM level 3 permet de lire et écrire des documents XML

- Bootstrapping

Support des espaces de nommage

- La méthode *write* s'assure de l'intégrité des déclarations d'espace de nommage avec le contenu

Support de `xml:base`

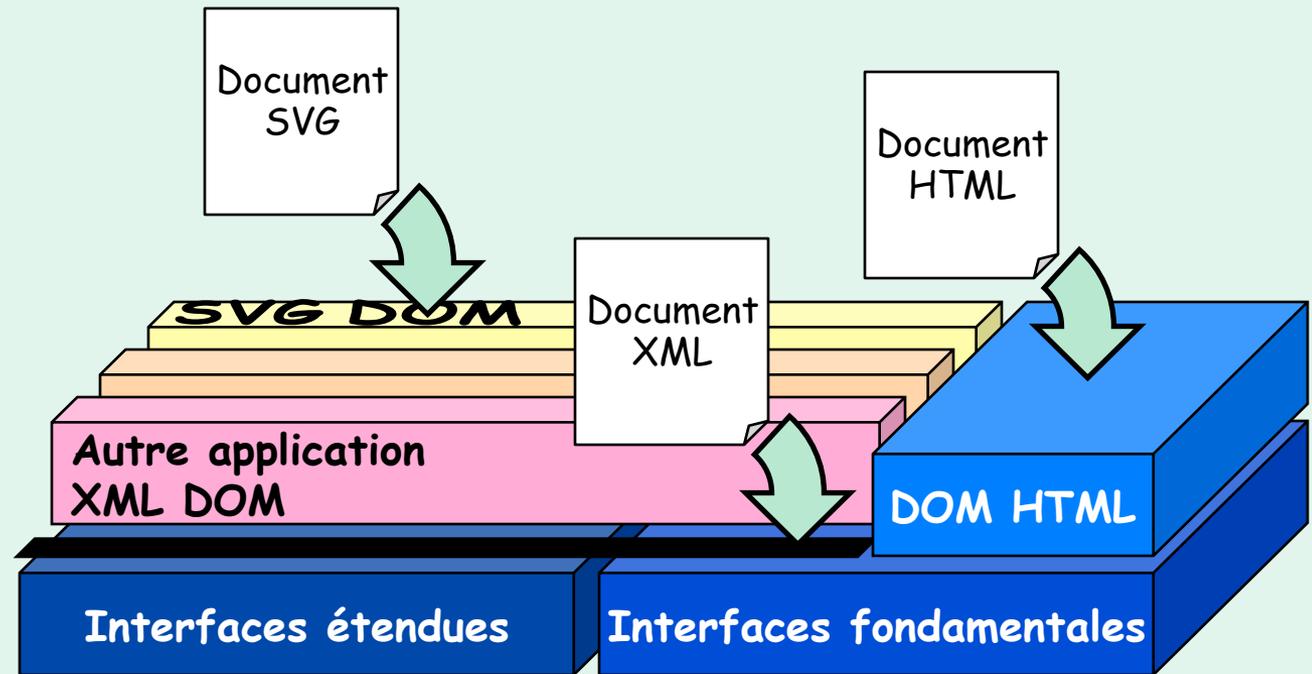
```
http://www.w3.org/TR/xmlbase/
```

Support des types de données de W3C XML Schema

Alignement avec XML Infoset

```
http://www.w3.org/TR/2004/REC-xml-infoset-20040204/
```

DOM Core
+
spécialisation
par domaine



Exemple : SVG DOM

SVG DOM	→	<code>Point.setX(12);</code>
DOM Core	→	<code>Element.setAttribute("x", "12");</code>

Généralisation par le design pattern « abstract factory »

```
package org.inria.ns.snippet.dom;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.xml.sax.SAXException;

public class DOMParserUtil {

    public static Document parse(InputStream input)
        throws ParserConfigurationException, SAXException, IOException {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setNamespaceAware(true);
        factory.setValidating(true);
        DocumentBuilder parser = factory.newDocumentBuilder();
        return parser.parse(input);
    }

    public static Document parse(File file)
        throws ParserConfigurationException, SAXException, IOException {
        return parse(new FileInputStream(file));
    }
}
```

Objet `document` disponible pour accéder au DOM HTML de la page.

Méthodes additionnelles pour accéder à des collections:

```
document.forms
document.links
documents.images
```

pour accéder à

```
<form>
<a>
<img>
```

Nombreuses librairies et framework pour accéder au DOM:

- **jQuery**
- Prototype
- MooTools
- dojo
- The Yahoo! User Interface Library
- **GWT** (Google Web Toolkit)

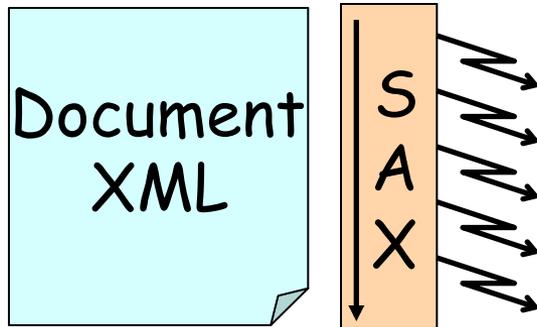
← Java compilé en Javascript

Contexte spécifique:

- Objets du navigateur
 - Gestion des événements
 - Sélecteurs CSS
 - AJAX : XMLHttpRequest
 - ✧ Widgets HTML
- Optimisation du code
 - Suppression code mort
 - Minification / obfuscation
 - Variantes par navigateur
 - Librairies Java
 - Outils Java (Eclipse)
 - *Impedance mismatch*

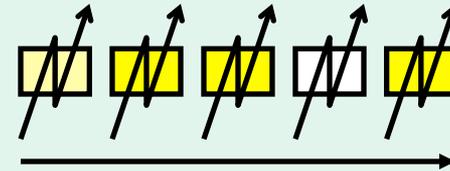
Simple API for XML

API simple pour XML :
une API pour **réagir** sur
le contenu et la structure d'un document



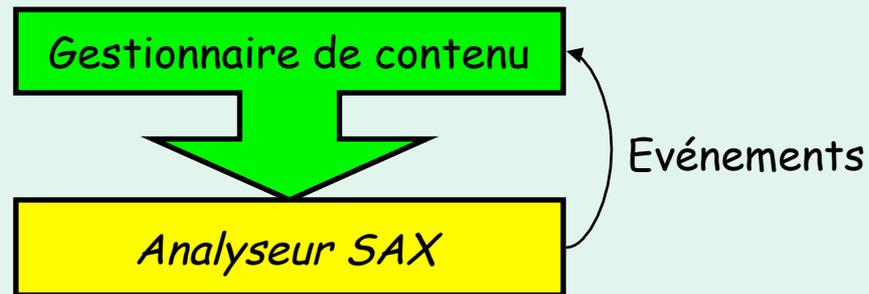
SAX, standard issu de xmldev mailing list
implémenté par David Megginson

Le document est vu comme une suite
d'événements qui surviennent lors de la
lecture séquentielle du document



Support des espaces de nommage, support des filtres SAX, nouvelles interfaces, changement du nom de certaines classes et interfaces, mécanisme générique pour lire ou modifier les propriétés et fonctionnalités du parser





A chaque événement, un *callback* est appelé dans l'ordre d'apparition dans le document

- Début de document
- Début d'élément
- Caractères
- ...
- Fin de document

Le gestionnaire de contenu est une classe qui effectue les traitements appropriés à chaque appel d'un *callback*

Cette classe doit être enregistrée auprès de l'analyseur SAX

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
  <Titre>Cours XML</Titre>
  <Auteur>
    <Nom>Poulard</Nom>
    <Prénom>Philippe</Prénom>
  </Auteur>
  <Description>
    Ce cours aborde les <b>concepts</b> de
    base mis en œuvre dans XML.
  </Description>
</Cours>

```

```

startDocument
startElement: Cours
startElement: Titre
characters: "Cours XML"
endElement: Titre
startElement: Auteur
startElement: Nom
characters: "Poulard"
endElement: Nom
startElement: Prénom
characters: "Philippe"
endElement: Prénom
endElement: Auteur
startElement: Description
characters: "Ce cours aborde les "
startElement: b
characters: "concepts"
endElement: b
characters: " de base mis en œuvre dans XML."
endElement: Description
endElement: Cours
endDocument

```

Les applications doivent enregistrer le *handler* d'événements auprès d'une instance d'un parser qui implémente l'interface `XMLReader`.

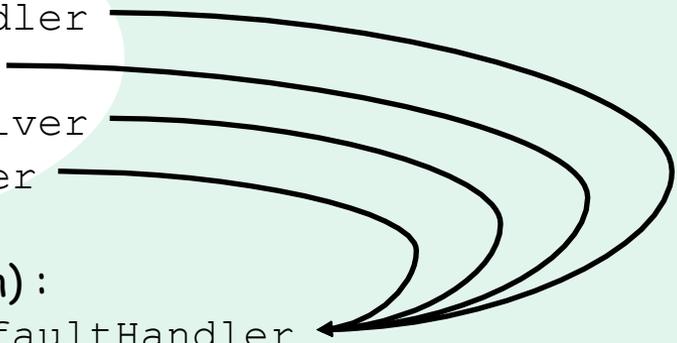
En Java :

Interfaces

Parser : `org.xml.sax.XMLReader`

Handlers : `org.xml.sax.ContentHandler`
`org.xml.sax.DTDHandler`
`org.xml.sax.EntityResolver`
`org.xml.sax.ErrorHandler`

Implémentation par défaut (qui ne fait rien) :
`org.xml.sax.helpers.DefaultHandler`

The diagram shows four curved arrows pointing from the right side of the 'Handlers' list to the 'DefaultHandler' implementation. Each arrow originates from one of the handler interface names and points to the 'DefaultHandler' text.

Avec MSXML6 :

Interfaces

Parser : `ISAXXMLReader`

Handlers : `ISAXContentHandler`
`ISAXDTDHandler`
`ISAXEntityResolver`
`ISAXErrorHandler`

`startDocument()` Notification du début du document

`endDocument()` Notification de la fin du document

`startElement(String namespaceURI, String localName, String qName, Attributes atts)` Notification du début d'un élément

`endElement(String namespaceURI, String localName, String qName)`
Notification de la fin d'un élément

`characters(char[] ch, int start, int length)`
Notification de données caractères

`ignorableWhitespace(char[] ch, int start, int length)`
Notification d'espaces blancs ignorables dans le contenu d'un élément

`startPrefixMapping(String prefix, String uri)`
Notification du début de la portée d'un espace de nommage

`endPrefixMapping(String prefix)`
Notification de la fin de la portée d'un espace de nommage

`processingInstruction(String target, String data)`
Notification d'une instruction de traitement

`skippedEntity(String name)`
Notification d'une entité non résolue

`setDocumentLocator(Locator locator)`
Permet d'enregistrer un `Locator` qui délivre des informations sur la localisation d'un événement SAX (numéro de ligne, colonne...)

Généralisation par le design pattern « abstract factory »

```
package org.inria.ns.snippet.sax;

import java.io.IOException;
import java.io.InputStream;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class SAXParserUtil {

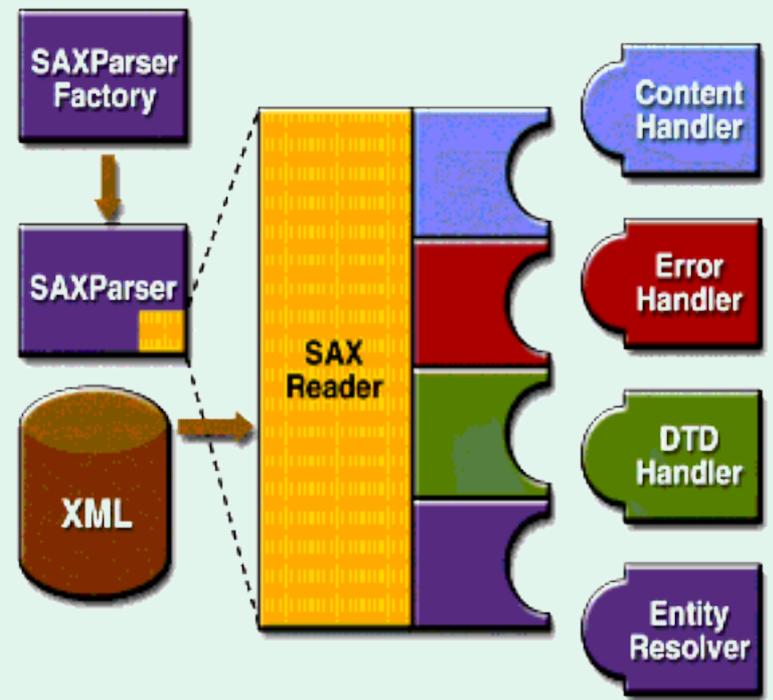
    static void parse(InputStream input, DefaultHandler handler)
        throws ParserConfigurationException, SAXException, IOException {
        SAXParserFactory factory = SAXParserFactory.newInstance();
        factory.setNamespaceAware(true);
        factory.setValidating(true);
        SAXParser parser = factory.newSAXParser();
        parser.parse(input, handler);
    }
}
```

```
String fileName = "...";
ContentHandler myContentHandler = new ...;
.../...
XMLReader parser = ...; ←
parser.setContentHandler(myContentHandler);
parser.parse(fileName);
```

```
SAXParserFactory factory =
    SAXParserFactory.newInstance();
SAXParser saxParser =
    factory.newSAXParser();
XMLReader parser =
    saxParser.getXMLReader();
```

```
String fileName = "...";
ContentHandler myContentHandler = new ...;
DTDHandler myDTDHandler = new ...;
ErrorHandler myErrorHandler = new ...;
EntityResolver myEntityResolver = new ...;
.../...
XMLReader parser = ...;
parser.setContentHandler(myContentHandler);
parser.setDTDHandler(myDTDHandler);
parser.setErrorHandler(myErrorHandler);
parser.setEntityResolver(myEntityResolver);
parser.parse(fileName);
```

```
String fileName = "...";
DefaultHandler handler = new ...;
.../...
XMLReader parser = ...;
parser.setContentHandler(handler);
parser.setDTDHandler(handler);
parser.setErrorHandler(handler);
parser.setEntityResolver(handler);
parser.parse(fileName);
```



Liste des éléments rencontrés :

```
import java.io.File;
import java.io.IOException;
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

public class MySAXParser extends DefaultHandler {
    public MySAXParser() {}
    static public void main(String[] args) {
        try {
            DefaultHandler handler = new MySAXParser();
            XMLReader parser = .../...;
            parser.setContentHandler(handler);
            parser.parse(args[0]);
        } catch (SAXException e) {e.printStackTrace();}
        } catch (IOException e) {e.printStackTrace();}
        }
    }
    public void startElement(String uri, String localName, String qName, Attributes atts) {
        System.out.println("Élément rencontré : " + qName);
    }
}
```

Invite de commandes

```
C:\>java MySAXParser cours.xml
Élément rencontré : Cours
Élément rencontré : Titre
Élément rencontré : Auteur
Élément rencontré : Nom
Élément rencontré : Prénom
Élément rencontré : Description
Élément rencontré : b
C:\>
```

La spécification SAX permet aux *parsers* de décomposer un segment de texte en plusieurs événements caractères de l'interface `ContentHandler`

```
<foo>
  Les APIs
  DOM &#x26; SAX
</foo>
```

```
startDocument
startElement : foo
characters : "\n  Les APIs\n  DOM & SAX\n"
endElement : foo
endDocument
```

Parser SAX de Xerces

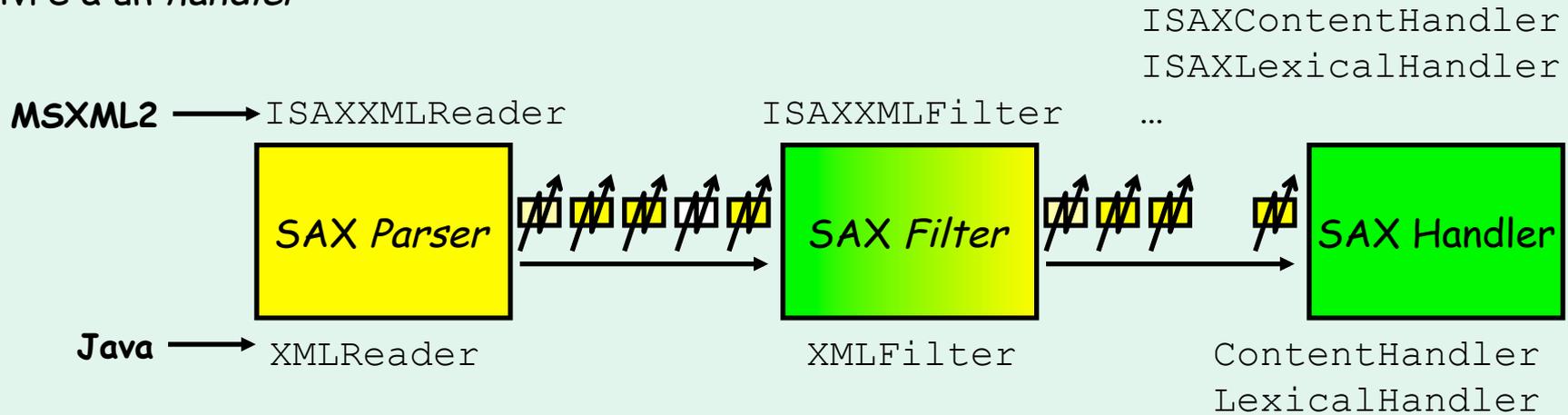
```
startDocument
startElement : foo
characters : "\n  Les APIs\n  DOM "
characters : "&"
characters : " SAX\n"
endElement : foo
endDocument
```

Parser SAX de Crimson

```
startDocument
startElement : foo
characters : ""
characters : "\n"
characters : "  Les APIs"
characters : "\n"
characters : "  DOM "
characters : "&"
characters : " SAX"
characters : "\n"
endElement : foo
endDocument
```

⇒ Les programmes qui traitent les caractères doivent utiliser des tampons (*buffers*)

Un filtre SAX reçoit des événements SAX d'un *parser* SAX, peut les modifier, et les fait suivre à un *handler*

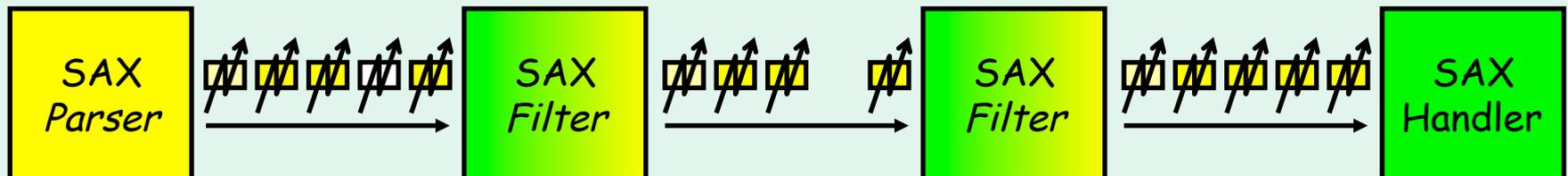


Le filtre SAX est vu par le *parser* SAX comme un *handler* SAX

Le filtre SAX est vu par le *handler* SAX comme un *parser* SAX

XMLFilter est dérivé de XMLReader

Pipeline SAX



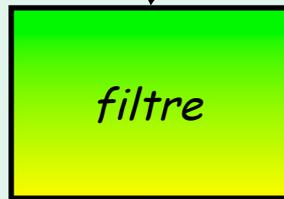
Permet à un *handler* d'accepter différentes variantes de structures

```
<livre>
  <titre>Les misérables</titre>
  .../...
</livre>
```

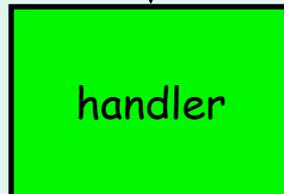
```
startElement : livre
startElement : titre
characters : "Les misérables"
endElement : titre
...
endElement : livre
```

```
<livre titre="Les misérables">
  .../...
</livre>
```

```
startElement : livre
...
endElement : livre
```



```
startElement : livre
startElement : titre
characters : "Les misérables"
endElement : titre
...
endElement : livre
```



à la réception de l'élément `livre`, le filtre vérifie s'il existe l'attribut `titre`.
Si c'est le cas, le filtre génère les événements :

```
startElement : titre
characters : "Les misérables"
endElement : titre
```

```
import java.io.File;
import java.io.IOException;
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

public class MySAXFilter extends XMLFilterImpl {
    public void startElement(String uri, String localName, String qName, Attributes atts) {
        if ( "livre".equals( localName ) && uri == null ) {
            int index = atts.getIndex( "titre" );
            if ( index != -1 ) {
                char[] titre = atts.getValue( index ).toCharArray();
                AttributesImpl myAtts = new AttributesImpl( atts );
                myAtts.removeAttribute( myAtts.getIndex( "titre" ) );
                super.startElement(uri, localName, qName, myAtts);
                super.startElement(uri, "titre", "titre", new AttributesImpl() );
                super.characters( titre, 0, titre.length );
                super.endElement(uri, "titre", "titre" );
                return;
            }
        }
        super.startElement(uri, localName, qName, atts);
    }
}
```

```
XMLReader parser = ...
XMLFilter filter = new MyFilter();
filter.setParent(parser);
filter.setContentHandler(handler);
filter.parse(.../...);
```

ou

```
XMLReader parser = ...
XMLFilter filter = new MyFilter(parser);
filter.setContentHandler(handler);
filter.parse(.../...);
```

Pipeline SAX

```
XMLReader parser = ...
XMLFilter myFilter = new MyFilter();
XMLFilter yourFilter = new YourFilter();
myFilter.setParent(parser);
yourFilter.setParent(myFilter);
yourFilter.setContentHandler(handler);
yourFilter.parse(.../...);
```

ou

```
XMLReader parser = ...
XMLFilter yourFilter = new YourFilter(new MyFilter(parser));
yourFilter.setContentHandler(handler);
yourFilter.parse(.../...);
```

Un analyseur validant qui ne valide pas
n'est pas
 un analyseur non validant

Traitement des espaces blancs

Certains analyseurs disposent de 2 implémentations :

- analyseur validant → `ignorableWhitespace()`
- analyseur non validant → `ignorableWhitespace()` **ou** `characters()`

Espaces blancs ignorables

```
<!ELEMENT dauphin (nom, sexe)>
<dauphin>
  <nom>Flipper</nom>
  <sexe>M</sexe>
</dauphin>
```

Truc : éviter d'avoir des espaces blancs significatifs

Si on ne peut pas faire autrement : les inclure dans une section CDATA

Pour les entités non résolues, il existe un *callback* spécifique :

```
public void skippedEntity(String name) throws SAXException {}
```

Mais les analyseurs non validants peuvent résoudre les appels d'entités

La spécification SAX2 définit les 2 fonctionnalités standards suivantes :

Introduits dans



Fonction	Nom	Valeur par défaut
Namespaces	<code>http://xml.org/sax/features/namespaces</code>	true
Namespace-prefix	<code>http://xml.org/sax/features/namespace-prefixes</code>	false

Le parser peut fournir des informations sur les URI d'espace de nommage et les *local names* via `startElement()` et `endElement()` de `ContentHandler` et `getURI()` et `getLocalName()` de `Attributes`.

Disponibilité des noms qualifiés, selon l'implémentation

`startPrefixMapping()` et `endPrefixMapping()` de `ContentHandler` sont appelés quand les éléments qui déclarent des espaces de nommage sont rencontrés et quittés.

Disponibilité des déclarations d'espace de nommage (`xmlns...`) dans les instances d'`Attributes`.

Namespace-prefix
Namespaces

false	true	x	-	x	-	Par défaut
true	true	x	x	x	x	
false	false	-	-	-	x	
true	false	-	x	-	x	

DOM

Un processus qui utilise le DOM ne peut traiter l'arbre qu'après la lecture entière du document

Accès aléatoire

Utilise beaucoup de mémoire

Construction de l'arbre du document
Les objets sont réutilisables
Richesse des fonctionnalités

Programmation aisée

Les implémentations de DOM reposent souvent sur un analyseur SAX

SAX

Un analyseur SAX délivre les données à un processus au fur et à mesure de la lecture du document

Accès séquentiel

Utilise peu de mémoire pour son propre fonctionnement, et le strict nécessaire pour le processus traitant

Événementiel
Les objets ne sont pas stockés
Fonctionnalités rudimentaires

Beaucoup de code à produire

DOM

Toutes les données doivent être utilisées

Les performances peuvent être limitées

Abondance de mémoire

SAX

Seule une partie des données doivent être utilisées

Les performances sont critiques

Peu de mémoire



Les *parsers* DOM récents ont des temps de réponse à peine plus faibles que ceux de SAX : ils diffèrent la création des objets qui ne sont pas sollicités.

DOM pour Java :

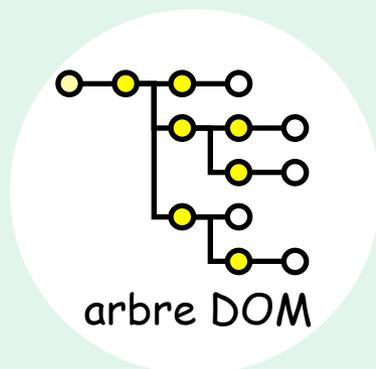
- JDOM
- DOM4J
- XOM

```
http://dom4j.org/  
http://www.jdom.org/  
http://xom.nu/
```

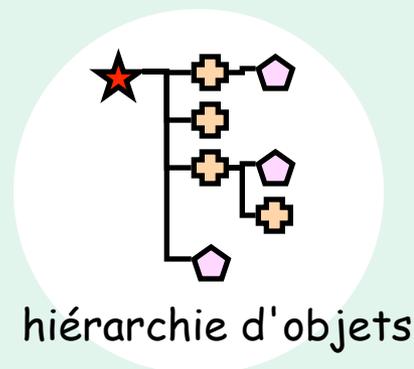
XML Data Binding

Au lieu d'obtenir un modèle de données XML, on cherche à instancier des classes qui correspondent aux éléments rencontrés

Modèle généraliste



Modèle spécifique



Des techniques permettent de réaliser la désérialisation (*unmarshal*) automatiquement

Avec Java :

- JAXB
- Castor

Autre API Java

```
javax.xml.stream.XMLOutputFactory  
javax.xml.stream.XMLStreamWriter  
  
javax.xml.stream.XMLInputFactory  
javax.xml.stream.XMLStreamReader
```

Plus facile à utiliser que SAX