

# XML et Programmation Internet

## Cours 4

kn@lri.fr

- 1 Introduction, UTF-8 et XML ✓
- 2 XPath ✓
- 3 XPath (suite) ✓
- 4 XSLT
  - 4.1 Namespaces
  - 4.2 XSLT, principes
  - 4.3 Les templates XSLT



## Utilisation des namespaces

**Problème** : comment mélanger au sein d'un même document des balises venant de deux DTDs (ou schémas) **différentes** ?

Exemple : mettre du SVG (images vectorielles) dans du XHTML (page web)

```

<svg width="400" height="180">
  <rect x="50"
    y="20"
    rx="20"
    ry="20"
    width="150" height="150"
    style="fill:red;stroke:black;
      stroke-width:5;opacity:0.5" />
</svg>
    
```

```

<html>
<head><title>Mon image</title></head>
<body>
  <h1>Une image</h1>
  <!-- contenu de l'image -->
</body>
</html>
    
```

Comment spécifier que certains éléments doivent être **interprétés** ? (par exemple par le navigateur web) et que d'autres font parties des données ?



## L'attribut xmlns

On peut utiliser un **attribut** spécial, `xmlns` qui est autorisé dans **tous les documents XML sur n'importe quel élément**. La valeur de ce dernier est un lien identifiant la DTD et indique tous les descendants de l'élément en question sont valides par rapport à la DTD.

Exemple:

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Mon image</title></head>
<body>
  <h1>Mon image</h1>
  <svg width="400" height="180" xmlns="http://www.w3.org/2000/svg">
    <rect x="50"
      y="20"
      rx="20"
      ry="20"
      width="150" height="150"
      style="fill:red;stroke:black;
        stroke-width:5;opacity:0.5" />
  </svg>
</body>
</html>
    
```

## L'attribut xmlns

On peut aussi donner un nom à un namespace particulier et réutiliser ce nom dans les balises:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head><title>Mon image</title></head>
  <body>
    <h1>Mon image</h1>
    <svg:svg width="400" height="180"
      <svg:rect x="50"
        y="20"
        rx="20"
        ry="20"
        width="150" height="150"
        style="fill:red;stroke:black;
          stroke-width:5;opacity:0.5" />
    </svg:svg>
  </body>
</html>
```

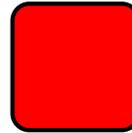
en utilisant l'attribut xmlns:toto="..." alors toutes la balises de la forme <toto:nombalise> appartiennent au namespace toto. Il ne faut pas oublier le namespace sur la balise fermante !

5/19

## Digression

L'exemple précédent ressemble à ça :

### Mon image



6/19

## Plan

- 1 Introduction, UTF-8 et XML ✓
- 2 XPath ✓
- 3 XPath (suite) ✓
- 4 XSLT
  - 4.1 Namespaces ✓
  - 4.2 XSLT, principes
  - 4.3 Les templates XSLT

7/19

## XSLT

**eXtensible Stylesheet Language Transformations** : est un langage de transformation de documents XML. Il permet d'extraire (au moyen d'XPath) des nœuds d'un document d'entrée et de les ré-organiser et de les copier dans un document de sortie. Le document de sortie est souvent du XML lui-même mais peut être aussi du texte, du PDF, ...

- XSLT est standardisé par le W3C
- Les programmes XSLT sont écrit eux-même en XML
- Il existe plusieurs versions du standard (1.0, 2.0, 3.0). Ce cours fait juste une introduction aux fonctionnalités les plus simples (1.0)

8/19

## Un exemple simple

On réutilise un document de recette de cuisine du **cours 01**. La DTD est:

```
<!ELEMENT recette (title,ingredients,duree,etapes)>
<!ATTLIST recette difficulte (facile|normal|difficile) #REQUIRED>
<!ELEMENT title #PCDATA >
<!ELEMENT ingredients (ingredient+) >
<!ELEMENT duree #PCDATA >
<!ELEMENT etapes (e*) >
<!ELEMENT ingredient #PCDATA>
<!ELEMENT e #PCDATA>
<!ATTLIST e num CDATA>
```

## Un exemple simple (suite)

Un document valide par rapport à la DTD est le suivant

```
<!DOCTYPE recette SYSTEM "recette.dtd">
<recette difficulte="facile">
<title>Tiramisú</title>
<ingredients>
  <ingredient>mascarpone</ingredient>
  <ingredient>oeufs</ingredient>
  <ingredient>sucre</ingredient>
  <ingredient>café</ingredient>
  <ingredient>biscuits</ingredient>
</ingredients>
<duree>2h</duree>
<etapes>
  <e num="1">Séparer les blancs des jaunes</e>
  <e num="2">...</e>
  ...
</etapes>
</recette>
```

## Un exemple simple (suite)

On va extraire des informations du document et en faire une page web :

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>Ingrédients du <xsl:value-of select="descendant::title" /> </title>
</head>
<body>
<h1>Ingrédients du <xsl:value-of select="descendant::title" /> </h1>
<ul>
<xsl:for-each select="descendant::ingredient">
<li> <xsl:value-of select="child::text()" /> </li>
</xsl:for-each>
</ul>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

## Comment exécuter un programme XSLT

- En utilisant un moteur XSLT comment le programme `xsltproc`
- En référençant la feuille de style et en ouvrant le document XML avec Firefox :

```
<!-- fichier recette.xml -->
<?xml-stylesheet type="text/xsl" href="prog.xsl"?>
<recette difficulte="facile">
  <title>Tiramisú</title>
  <ingredients>
    ...
```

- En Java grâce à la classe `Transformer` du package `javax`

## Structure d'un programme XSLT

## Plan

Un programme XSLT commence par la balise :

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

et fini par la balise :

```
</xsl:stylesheet>
```

Le texte ainsi que les éléments qui ne sont pas dans le *namespace* « `xsl:` » sont copiés à l'identique dans la sortie.

Dans la balise principale `<xsl:stylesheet>` on trouve un certain nombre de balises `<xsl:template match="...">`.

Les *templates* vont jouer le rôle de fonctions et vont être appelées pour transformer des nœuds du document en nœuds de sortie.

1 Introduction, UTF-8 et XML ✓

2 XPath ✓

3 XPath (suite) ✓

4 XSLT

4.1 Namespaces ✓

4.2 XSLT, principes ✓

4.3 Les templates XSLT

## Templates

Un template est délimité par une balise `<xsl:template match="...">` où le « ... » est une expression XPath. Il peut y avoir plusieurs templates dans un fichier XSLT. Un template est appliqué à un ensemble de nœuds. Parmi cet ensemble, tous les nœuds qui vérifient l'expression XPath de l'attribut `match` sont passés en argument au template.

Initialement, tous les templates existants sont appliqués au nœud fictif document.

## Retour sur l'exemple

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <head>
    <title>Ingrédients du <xsl:value-of select="descendant::title" /> </title>
  </head>
  <body>
    <h1>Ingrédients du <xsl:value-of select="descendant::title" /> </h1>
    <ul>
      <xsl:for-each select="descendant::ingredient">
        <li><xsl:value-of select="child::text()" /> </li>
      </xsl:for-each>
    </ul>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

1. Initialement tous les templates (il n'y en a qu'un ici) sont appliqués à la racine du document

2. Pour chacun d'entre eux, l'expression XPath dans le `match` est contrôlée. Si elle réussit, alors le template est appliqué au nœud

## Deuxième exemple, 2 templates

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>Ingrédients du <xsl:value-of select="descendant::title" /> </title>
</head>
<body>
<h1>Ingrédients du <xsl:value-of select="descendant::title" /> </h1>
<ul>
<xsl:apply-templates select="descendant::ingredient" />
</ul>
</body>
</html>
</xsl:template>
<xsl:template match="ingredient">
<li> <xsl:value-of select="child::text()" /> </li>
</xsl:template>
</xsl:stylesheet>
```

1. On applique tous les templates à la racine. Le template `match="/"` réussit, le template `match="ingredient"` échoue.
2. L'expression `<xsl:apply-templates select="..." />` réapplique tous les templates à tous les nœuds sélectionnés par l'expression XPath.

17/19



## Priorité des templates

Que se passe-t-il si deux templates peuvent s'appliquer à un même élément ? On applique les règles de priorité suivantes (simplifiées) :

1. Si le match d'un template contient un `|` entre plusieurs chemins, la priorité la plus élevée de ces chemins est choisie
2. si le match est de la forme : `child::foo` ou `attribute:foo`, sa priorité est 0
3. si le match est de la forme : `child::*` ou `attribute:*`, sa priorité est -0.25
4. si le match est de la forme : `child::text()` ou `child::node()` (idem avec `attribute`), sa priorité est -0.5
5. sinon (test complexe, par exemple `foo[ count(child::bar) < 4 ]`), la priorité est 0.5

Le template ayant la priorité la plus élevée est choisi. On peut donner explicitement à un template une priorité avec l'attribut `priority="..."`

Le moteur XSLT lève une erreur d'exécution si deux templates ont exactement la même priorité.

18/19

## Templates par défaut

Il existe 3 templates par défaut qui sont toujours définis :

```
<xsl:template match="*|/">
<xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="text()|@*">
<xsl:value-of select="."/>
</xsl:template>
```

```
<xsl:template match="processing-instruction()|comment()"/>
```

Ils ont des priorités inférieures aux templates présents dans le fichier donc ne sont appliqués que si aucun templates ne fonctionne. Ils permettent de « descendre dans l'arbre » (jusqu'aux textes), si aucun template ne s'applique à la racine.