UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Introduction XML

Mathieu RAYNAL

mathieu.raynal@irit.fr

http://www.irit.fr/~Mathieu.Raynal

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Comment mettre en page un document XML ?

- Les Cascading StyleSheets (CSS)
- XSL Transformations (XSLT)
- XSL Formating Object (XSL-FO)

Liaison avec un document XML

```
<?xml-stylesheet type=" " href=" "?>
```

- Instruction à rajouter
  - Après la déclaration XML
  - Avant la balise racine

- L'attribut *type* peut avoir comme valeur :
  - text/css pour les feuilles de style CSS
  - text/xml ou application/xml pour les feuilles XSL

Le langage CSS

- Permet de décrire l'apparence d'éléments du document XML
- Dans un fichier avec l'extension .css
- Syntaxe non XML

La syntaxe d'un fichier CSS

```
<body>
<h1>Titre 1</h1>
<h2>Sous titre 1.1</h2>
<h2>Sous titre 1.2</h2>
<h2>Sous titre 1.3</h2>

<h1>Titre 2</h1>
<h2>Sous titre 2.1</h2>
<h2>Sous titre 2.2</h2>
<h2>Sous titre 2.3</h2>

<h1>Titre 3</h1>
<h2>Sous titre 3.1</h2>
<h2>Sous titre 3.2</h2>
            <h2>Sous titre 3.3</h2>
</body>
```


```
h1{
            color:red;
            font-family: Verdana;
            font-size: 14pt;
}

h2{
            color:green;
            font-family: Verdana;
            font-size: 11pt;
}
```

Les classes

- Il est possible de spécifier différents styles pour un même type d'éléments

```
<body>
<h1 class="cas1">Titre 1</h1>
<h2>Sous titre 1.1</h2>
<h2>Sous titre 1.2</h2>
<h2>Sous titre 1.3</h2>

<h1 class="cas2">Titre 2</h1>
<h2>Sous titre 2.1</h2>
<h2>Sous titre 2.2</h2>
<h2>Sous titre 2.3</h2>

<h1 class="cas1">Titre 3</h1>
<h2>Sous titre 3.1</h2>
<h2>Sous titre 3.2</h2>
        <h2>Sous titre 3.3</h2>
</body>
```


```
h1.cas1{
        color:red;
        font-family: Verdana;
        font-size: 14pt;
}

h1.cas2{
        color:blue;
        font-family: Verdana;
        font-size: 14pt;
}

h2{
        color:green;
        font-family: Verdana;
        font-size: 11pt;
}
```

Utilisation de plusieurs classes

```
<html>
<head>
        <link rel="stylesheet" type="text/css" href="test.css" media="screen" />
</head>
<body>
        <div id="haut" class="gauche rouge">
                <h1>Titre du haut</h1>
                Un peu de texte
                <h1 class="noir">Deuxieme Titre du haut</h1>
        </div>
        <div id="bas" class="centre bleu">
                <h1>Titre du bas</h1>
                Un autre texte
        </div>
</body>
</html>
```

```
.rouge{color: #f00;}
.bleu{color: #00f;}
.gauche{text-align: left;}
.centre{text-align: center;}
#haut h1{color: #0f0;}
#haut h1.noir{color: #000;}
```

Titre du haut

Un peu de texte

Deuxieme Titre du haut

Titre du bas

Un autre texte

Les principaux styles

- Les couleurs
  - color
  - background-color
- Les fontes
  - font-family
  - font-size
  - font-style
  - font-weight
- Le texte
  - text-indent
  - text-align
- Les bordures
  - border: width style color
  - border-color
  - border-width
  - border-style
- Les marges
  - Intérieur : padding
  - Extérieur : margin
- Les marges et les bordures peuvent être appliquées à l'ensemble des côtés ou à un côté en particulier (left, top, right, bottom)

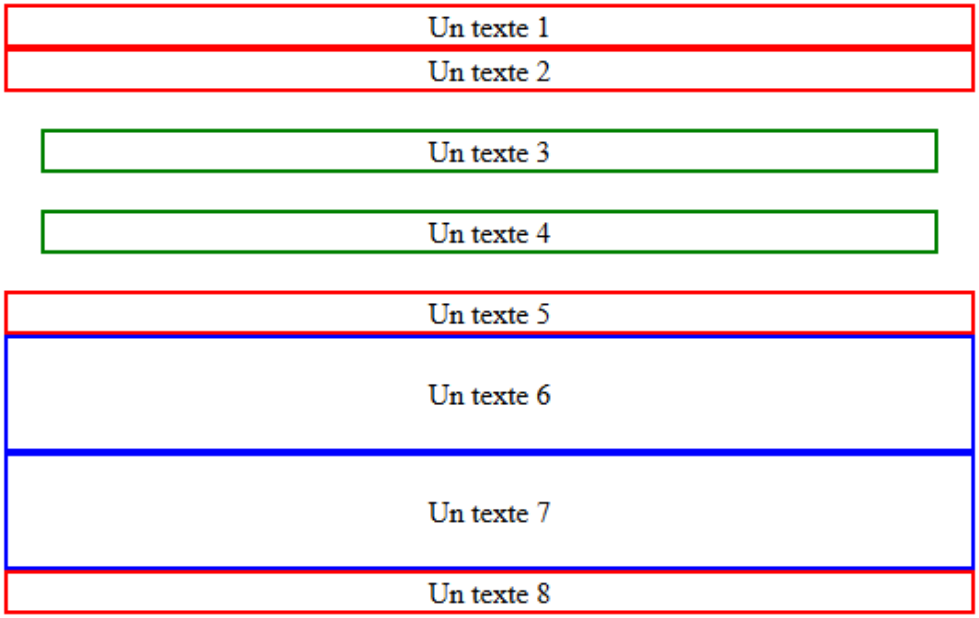
Différence entre margin et padding

```
<body>
<center>
        <div class="d1">Un texte 1</div>
        <div class="d1">Un texte 2</div>
        <div class="d2">Un texte 3</div>
        <div class="d2">Un texte 4</div>
        <div class="d1">Un texte 5</div>
        <div class="d3">Un texte 6</div>
        <div class="d3">Un texte 7</div>
        <div class="d1">Un texte 8</div>
</center>
</body>
```

```
.d1{
        border: 2px solid red;
}

.d2{
        border: 2px solid green;
        margin: 20px;
}

.d3{
        border: 2px solid blue;
        padding: 20px;
}
```



Le langage XSL

- XSL : eXtensible Stylesheet Language
- XSL est défini avec le formalisme XML
- Il est composé de :
  - XSLT : eXtensible StyleSheet Transformation
    - Transforme un document XML à partir de règles
  - XSL/FO : eXtensible StyleSheet Formatting
    - Définit la mise en page du document

XSL Transformation (XSLT)

- Permet de transformer un document XML en un autre document XML
- Contient des règles de modèle
- Compare le document en entrée avec les différentes règles
- Si correspondance →Modèle appliqué pour le document de sortie

Structure d'un document XSLT

- Fichier avec extension .xsl

- Une feuille de style minimale

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

        Opérations de transformations …

</xsl:stylesheet>
```

  - xsl:stylesheet est l'élément racine

Les opérations de transformations

- Chaque modèle est représenté par un élément xsl:template

- Définit un comportement particulier pour tous les types d'éléments indiqués par l'attribut match

```
<xsl:template match="nomElt"><b>Contact :</b></xsl:template>
```

- Pour appliquer un template

```
<xsl:apply-templates select="nomElt" />
```

L'attribut *match* de *xsl:template*

- / est l'élément racine

- Il est possible de définir un nœud en particulier :
  - elt1|elt2 : l'élément 1 ou l'élément 2
  - elt1/elt2 : tout élément 2 qui est un fils d'élément 1
  - * : tout élément
  - elt1//elt2 : tout élément 2 qui est dans élément 1 (quelque soit la profondeur)

Les opérations de transformations

- Insère la valeur d'un élément

```
<xsl:value-of select="">
```

- Insère un nouvel élément

```
<xsl:element name="">valeur</xsl:element>
```

- Insère un attribut dans l'élément qui le précède

```
<elt><xsl:attribute name="">valeur</xsl:attribute></elt>
```

Utilisation de boucles

- Appliquer un comportement à plusieurs éléments

```
<xsl:for-each select="">traitement …</xsl:for-each>
```

  - Possibilité d'effectuer un tri

```
<xsl:sort select="“ data-type="“ order="“/>
```

    - data-type : text, number ou elt
    - Order : ascending ou descending

Exemple

```
<xsl:template match="/">
<HTML> <HEAD> <TITLE>Titre de la page</TITLE> </HEAD>
 <BODY BGCOLOR="#FFFFFF"> <xsl:apply-templates/> </BODY>
</HTML>
 <xsl:template >
            <xsl:template match="personne" >
             <ul>
            <li> <xsl:value-of select="nom"/> - <xsl:value-of select="prenom"/> </li>
            </ul>
</xsl:template >
```

Traitement des données

Document Object Model (DOM)

- Spécification du W3C
- Approche hiérarchique
  - Construit un arbre en mémoire contenant l'intégralité des éléments du document
- On peut agir dynamiquement sur la structure
- C'est une API qui permet :
  - Créer / modifier / supprimer des nœuds
  - Consulter le contenu (attributs/données) d'un nœud

Les différents nœud DOM

- Node
  - DocumentFragment
    - Document
- DocumentType
- Notation
- Element
- Entity
- Attribute
- Data
  - Text
    - CDATASection
  - Comment

- En PHP
  - Lecture
    - getElementByTagName
    - getElementById
    - getAttribute
  - Ecriture
    - createElement
    - createTextNode
    - setAttribute
    - removeChild

DOM : Avantages / Inconvénients

- Avantages
  - Adapté aux applications dynamiques
  - Utilisé dans les navigateurs Web
    - Pour faire du web dynamique

- Inconvénients
  - Gourmand en ressources
    - Stockage de l'arbre en mémoire

Simple API for XML (SAX)

- API événementielle incrémentale
  - Dès qu'une balise, un attribut, des données sont rencontrés → appel d'une méthode
- Le tout n'est pas conservé en mémoire
- Défini au départ en JAVA

- L'interface XMLReader permet de lire le contenu d'un fichier XML
- Création d'une instance de XMLReader grâce à la classe XMLReaderFactory

```
XMLReader saxReader = XMLReaderFactory.createXMLReader();
saxReader.setContentHandler(new Parseur(listTasks,fileDir));
saxReader.parse(XMLFileName);
```

  - Où **Parseur** est une instance de ContentHandler

Interface ContentHandler

- Sert d'écouteur
- Permet de parser un fichier XML sous forme événementielle
  - Pour chaque balise ou contenu du fichier, une méthode associée

```
public void startDocument()
public void endDocument()
public void startElement(String uri, String localName, String qName, Attributes atts)
public void endElement(String uri, String localName, String qName)
public void characters(char[] ch, int start, int length)
public void startPrefixMapping(String prefix, String uri)
public void endPrefixMapping(String prefix)
public void ignorableWhitespace(char[] ch, int start, int length)
public void processingInstruction(String target, String data)
public void setDocumentLocator(Locator locator)
public void skippedEntity(String name)
```