M1IF03
Conception d'applications Web

TEMPLATING DE PAGES WEB

LIONEL MÉDINI

OCTOBRE-DÉCEMBRE 2018

Plan du cours

- Introduction
- Exemples de templating
  - JSP
  - XSL
  - Web Components
  - Mustache
- Conclusion

Introduction

- Contexte
  - Application « correctement » structurée
    - Par exemple : patterns MV*
  - Framework
  - CMS
- Problème
  - Générer dynamiquement des vues à partir des données
- Solution 1 (programmatique)
  - Côté serveur

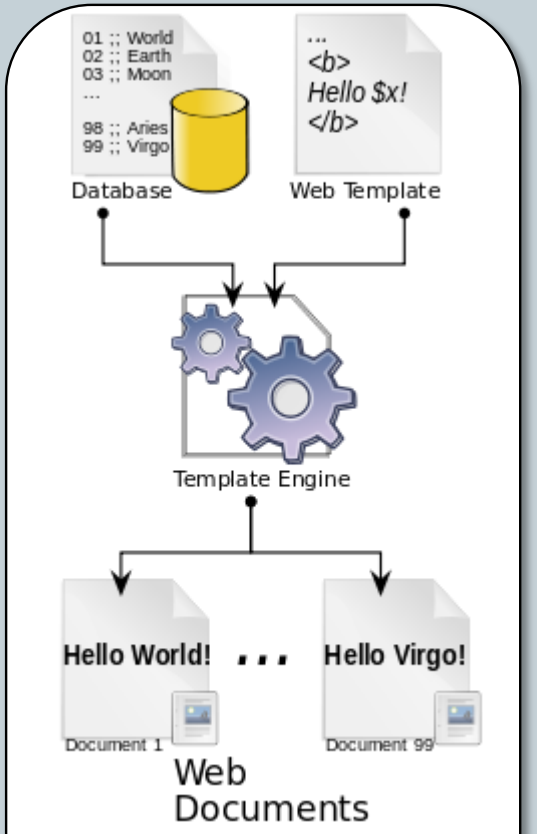
```
out.println("Le résultat est : " + resultat);
```

  - Côté client

```
Document.getElementById("monDiv").innerHTML = "Le résultat est : " + resultat;
```

Introduction

- Contexte
  - Application « correctement » structurée
    - Par exemple : patterns MV*
  - Framework
  - CMS
- Problème
  - Générer dynamiquement des vues à partir des données
- Solution 2 (déclarative)
  - Utiliser un outil générique capable de « mélanger » les deux



Source :
https://en.wikipedia.org/wiki/Web_template_system

Introduction

- Méthode
  - Définir des modèles à « trous »
    - Modèles (templates) : vues abstraites
  - Appeler un moteur de templates
    - Lui passer en paramètre
      - Les données à templater
      - Le modèle
- Fonctionnement
  - Évaluation des expressions dans les balises spécifiques
    - Remplacement
    - Structuration
    - Imbrication

Introduction

- Contraintes
  - Tenir compte du langage de programmation (pas de conflit entre les syntaxes)
  - Tenir compte de la structure des données
  - Pouvoir générer plusieurs types de contenus
- Remarques
  - Pattern pas spécifique au Web
  - Plusieurs types de tâches de templating
  - Les templates peuvent être compilés en code impératif pour gagner du temps

Exemples d'outils

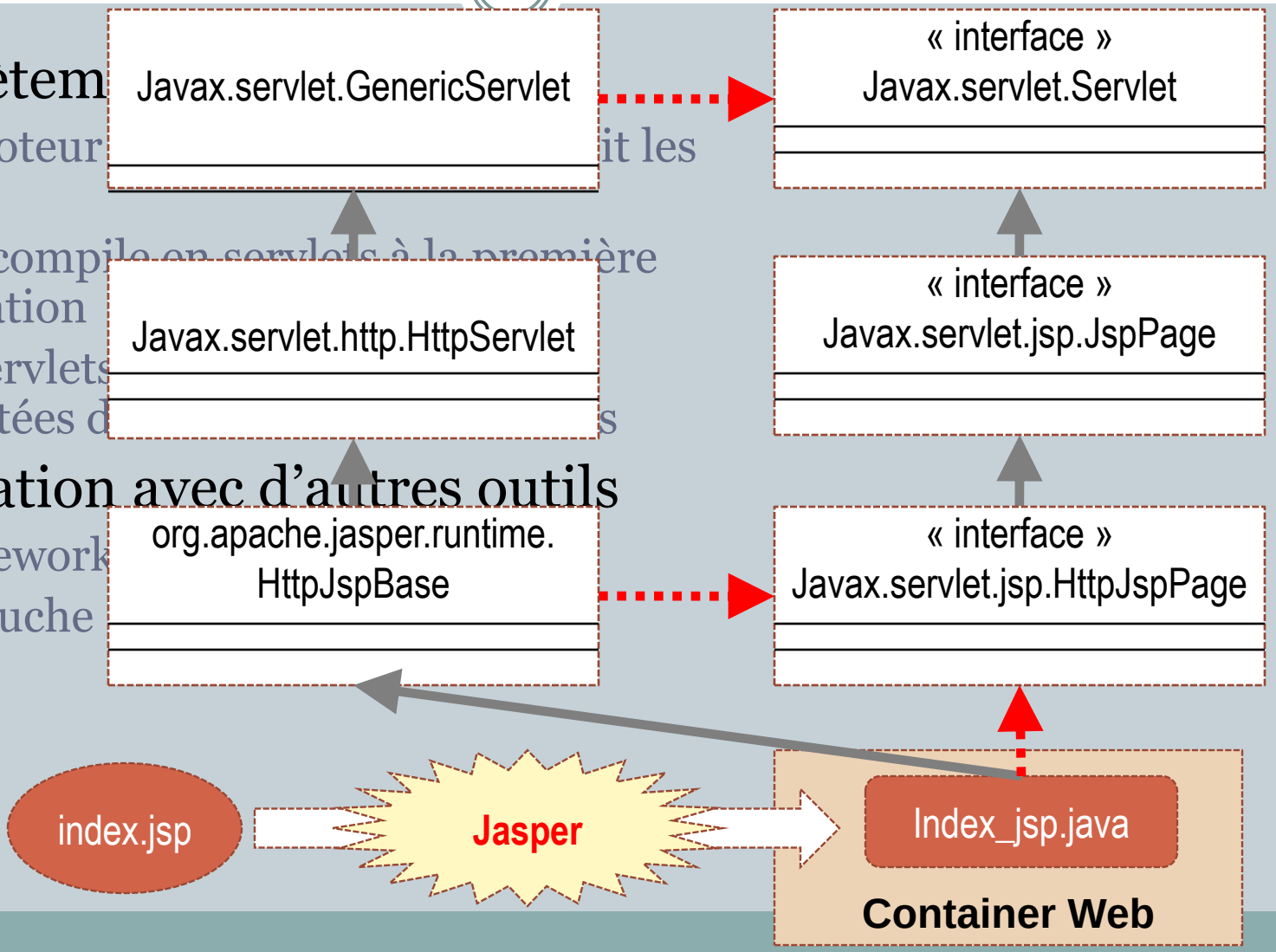
- Pouvez-vous en citer un ?

Exemple 1 : JSP

- Rappel
  - Côté serveur
  - Composants d'application dédiés à la vue
  - Peuvent "tirer" les objets du modèle
- Méthode de templating
  - Possèdent des balises spécifiques
    - Scriptlets
    - Taglib
    - Expression language
  - Accèdent à des données
    - Du modèle
    - De la requête
    - Du contexte applicatif
    - …

Exemple 1 : JSP

- Concrètem
  - Un moteur it les JSP
  - Il les compile en servlets à la première utilisation
  - Les servlets exécutées d s
- Intégration avec d'autres outils
  - Framework
  - Surcouche



Exemple 2 : XSL

- Caractéristiques
  - Côté serveur ou côté client
  - Composant de base de la “galaxie XML”
    - Lié aux recommandations XML, XML-Schema, DTD, XHTML…
- Origine
  - Première version : 2001
  - Officiellement : XML Stylesheet Language
  - En pratique, ça ne sert à rien d’appliquer des éléments de style à un document XML

  → On utilise les documents XML comme sources de données

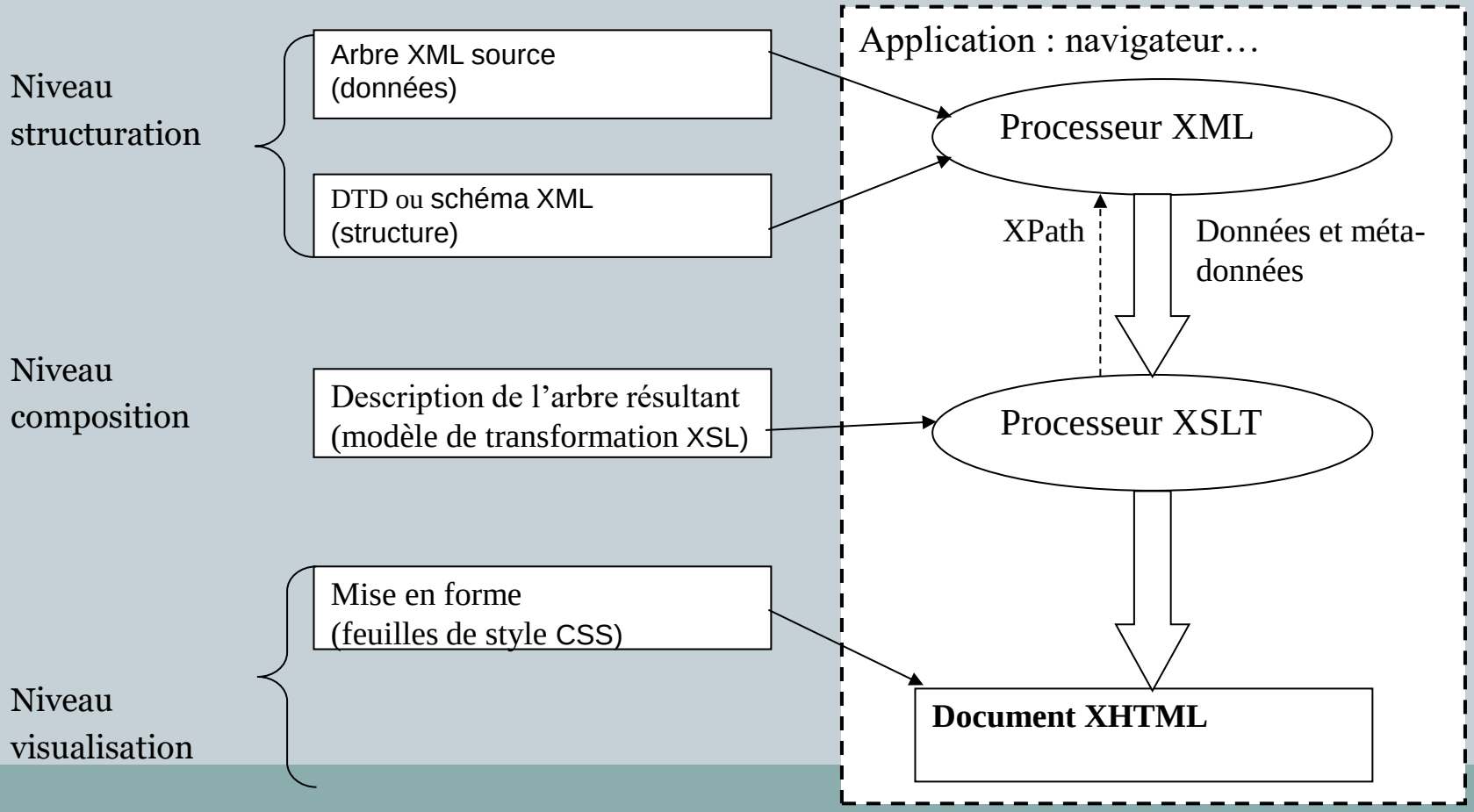
  → On réalise des transformation d’arbres

Exemple 2 : XSL

- Caractéristiques
  - XSL fournit un mécanisme très puissant pour transformer un arbre XML
    - En un autre arbre XML (échange de données)
    - En un arbre XHTML (visualisation des données XML)
    - En un texte simple (fichier non destiné à une application utilisant un parser XML)
    - En un document papier formatté (XSL-FO)
    - …

Transformation d'arbres XML : XSL

- Utilisation la plus courante de XSL



Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XPath
    - Permet de pointer vers les données de l'arbre XML
      - pour le parcours de documents XML
      - pour le test de valeurs associées aux contenus ou aux attributs d'éléments
    - Ne respecte pas la syntaxe XML
      - pour ne pas « perturber » l'analyse des feuilles de style XSLT par le parser XML

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XPath
    - Nœud
      - Tout type de données (élément, attribut, PI)
      - Racine du document : '/'
      - Les éléments sont identifiés par leurs noms
      - Les attributs sont identifiés par '@' suivi du nom de l'attribut
    - Chemin de localisation
      - Absolu : à partir de la racine de l'arbre XPath
      - Relatif : à partir du nœud contextuel
      - Récursif : à partir du nœud contextuel, mais seulement « vers le bas »

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XPath
    - Axes de navigation
      - Déplacements complexes dans l'arbre XPath
      - Syntaxe : Nom_D_Axe::Nom_De_Noeud
      - Recommandation : http://www.w3.org/TR/xpath20/#axes

Nom d'axe	Description	Exemple d'utilisation/ syntaxe abrégée
self	Nœud contextuel	self::node() ou ./node() ou .
child	Enfants du nœud contextuel	child::Etat_civil ou Etat_civil (défaut)
descendant	Tout enfant, petit enfant etc. du nœud contextuel	descendant::Etat_civil
descendant-or-self	Comme descendant + le nœud contextuel lui-même	descendant-or-self:: Etat_civil ou .//Etat_civil
parent	Parent du nœud contextuel	parent::Prenom ou ../Prenom
ancestor	Tout parent, grand parent etc. du nœud contextuel	ancestor::Prenom
ancestor-or-self	Comme parent + le nœud contextuel lui-même	ancestor-or-self::Prenom
following-sibling	Tous les frères suivants du nœud contextuel (vide si le nœud est un attribut)	following-sibling::Nom
preceding-sibling	Tous les frères précédents du nœud contextuel (vide si le nœud est un attribut)	preceding-sibling::Prenom
following	following–sibling + descendants de tous les nœuds frères suivants	following::Nom
preceding	preceding–sibling + descendants de tous les nœuds frères précédents	preceding::Prenom
attribute	Attributs du nœud contextuel	attribute::id ou./@id
namespace	Tous les nœuds appartenant au même espace de noms que le nœud indiqué	namespace::xhtml:div

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XPath
    - Opérateurs et fonctions
      - Expression de caractéristiques de sélection complexes
      - Communs avec XQuery
      - Recommandation à part entière : http://www.w3.org/TR/xquery-operators/

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XPath
    - Opérateurs et fonctions
      - Accesseurs
        - Pour récupérer un élément d'un nœud
        - Exemples : node-name(), string(), base-uri()
      - Génération d'erreurs
        - error()
      - Génération de traces
        - trace()
      - Constructeurs
        - Pour les types de données XML spécifiques
        - Exemple : MonType()
      - Casting entre types de données

Transformation d'arbres XML : XSL

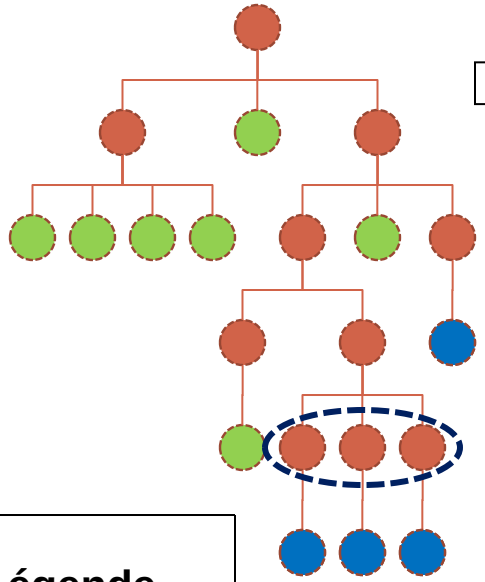
- Les deux composants de XSL
  - XSLT : principes de base
    - Description de l'arbre résultant (programmation déclarative)
    - Application XML définissant des « éléments de transformation »
      - ⇒ Référence à un espace de noms spécifique « `xsl:` »
    - Éléments et attributs spécifiques interprétés par un processeur XSLT
    - Structuration par modèles (« templates ») de contenus
      - Définissant le traitement à appliquer à un nœud repéré par une expression XPath
      - Imbriqués grâce à des mécanismes d'application de templates

      ➔parallèle avec les fonctions en programmation déclarative

Transformation d'arbres XML : XSL

- Principe général de la transformation d'arbres

Arbre XML source (données)



Légende
- élément
- attribut
- texte

Requête XPath

Données et méta-données

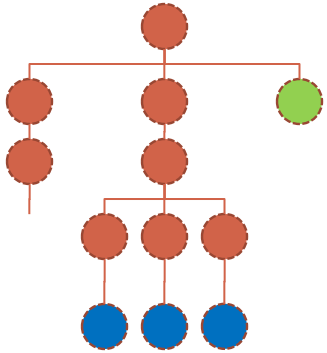
Processeur XSLT

Description de l'arbre résultant (feuille de style XSLT)

- Modèles de contenus
- Conditions à remplir pour ces modèles
- Spécification des données à récupérer

```
<CV xmlns="…">
<Etudiant>
<Nom>Toto</Nom>
<Rubrique>
<Item>
<Contenu>1</Contenu>
<Contenu>2</Contenu>
<Contenu>3</Contenu>
</Item>
</Rubrique>
</Etudiant>
</CV>
```

Arbre résultat (après transformation)



```
<html xmlns="…">
<head>
<title>Toto </title>
</head>
<body>
<ul>
<li> 1 </li>
<li> 2 </li>
<li> 3 </li>
</ul>
</body>
</html>
```

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : syntaxe
    - Élément racine

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

    - Éléments de premier niveau (cardinalité=0 ou 1)
      - `<xsl:output>` : définit le type d'arbre de sortie
        - Attribut `method` : 3 valeurs possibles (`text`, `html`, `xml`)
        - Autres attributs : `version`, `encoding`, `standalone`, `indent`…
      - `<xsl:include>` et `<xsl:import>` : permettent d'inclure d'autres feuilles de style
        - Attribut `href` : URI de la ressource à inclure
        - Différence entre les deux : règles de priorités

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : syntaxe
    - Éléments de premier niveau (cardinalité=0 ou 1)
      - `<xsl:strip-space>` et `<xsl:preserve-space>` : gestion des espaces dans l'arbre résultant (resp. suppression et conservation)
        - Attribut `elements` : noms des éléments concernés séparés par des espaces
      - `<xsl:template>` : modèle racine de l'arbre de sortie
        - Attribut `match` : désigne le nœud XPath concerné par le modèle (au premier niveau, toujours "/")
        - Contient la racine de la déclaration de l'arbre de sortie
      - Autres éléments (key, variable, attribute-set, param) : voir la recommandation

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les templates
    - Définition
      - Modèles simples : `<xsl:template match="noeud_XPath">`
        - L'expression XPath qui définit le nœud peut inclure un filtre
        - Ce nœud devient le nœud contextuel dans le template
      - Modèles nommés : `<xsl:template name="nom_tmplate">`
    - Appel
      - Modèles simples :

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

        - L'expression XPath est un chemin de localisation qui désigne le nœud
      - Modèles nommés :

```
<xsl:call-template name="nom_template" />
```

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les éléments
    - Génération de contenus XML
      - `<xsl:element name="p" namespace="xhtml">Contenu de l'élément (ici: un paragraphe XHTML)</xsl:element>`

        Remarque : `<xsl:element>` n'est nécessaire que lorsque le nom de l'élément à générer doit être calculé
      - `<xsl:attribute name="href" namespace="xhtml">Contenu de l'attribut (ici : référence XHTML)</xsl:attribute>`

        Remarque : `<xsl:attribute>` se place dans l'élément auquel il se rapporte
      - `<xsl:text>Contenu textuel quelconque.</xsl:text>`

        Remarque : `<xsl:text>` ne sert qu'au formatage du texte (gestion des espaces…)
      - Tout autre élément XML bien formé est accepté

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les éléments
    - Traitement de contenus de l'arbre XML source
      - `<xsl:value-of select="expr_XPath" />`
        - Permet d'obtenir la valeur d'un nœud (élément ou attribut)
        - L'expression XPath est un chemin de localisation
        - Elle désigne un nœud à partir du nœud contextuel
      - `<xsl:copy-of select="expr_XPath" />`
        - Permet de recopier dans l'arbre destination toute une partie de l'arbre source
        - L'expression XPath fonctionne comme précédemment
      - `<xsl:copy />`
        - Permet de copier uniquement un élément sans ses sous-éléments

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les éléments
    - Structures de contrôle
      - `<xsl:if test="expr_XPath">Contenu conditionnel</xsl:if>`
        - Le contenu conditionnel peut être composé d'autres éléments (`<xsl:value-of select="expr_XPath" />`)
      - `<xsl:for-each select="expr_XPath">Contenu répété</xsl:for-each>`
        - Cet élément est redondant avec `<xsl:apply-templates />` mais rend la feuille de style moins lisible

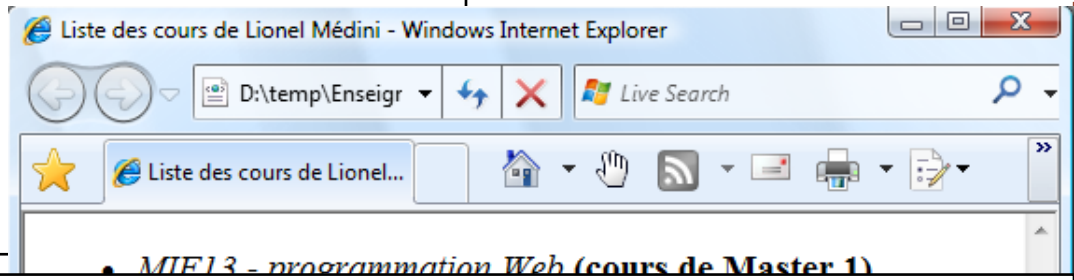
Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les éléments
    - Structures de contrôle
      - `<xsl:choose>`

```
<xsl:choose>
        <xsl:when test="expr_XPath1">
                Contenu conditionnel 1
        </xsl:when>
        <xsl:when test="expr_XPath2">
                Contenu conditionnel 2
        </xsl:when>
        ...
        <xsl:otherwise>
                Contenu conditionnel n
        </xsl:otherwise>
</xsl:choose>
```

```xml
<!-- enseignement.xml -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="enseignement.xsl"?>
<enseignement>
 <auteur>Lionel Médini</auteur>
 <cours>
  <titre>TI1 - systèmes d'information distribués</titre>
  <niveau>Master 2</niveau>
 </cours>
 <cours>
  <titre>MIF13 - programmation Web</titre>
  <niveau>Master 1</niveau>
 </cours>
</enseignement>
```

Directive d'application de la feuille de style au document XML




```xml
<!-- enseignement.xsl - autre possibilité, avec templates imbriqués -->
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
  <html>
   <head><title>Liste des cours de <xsl:value-of select="/enseignement/auteur"/></title></head>
   <body>
    <ul>
     <xsl:apply-templates select="enseignement/cours">
    </ul>
   </body>
  </html>
 </xsl:template>
 <xsl:template match="cours">
  <xsl:sort select="niveau"/>
  <li>
   <span style="font-style: italic;"><xsl:value-of select="titre"/></span>
   <span style="font-weight: bold;"> (cours de <xsl:value-of select="niveau"/>)</span>
  </li>
 </xsl:template>
</xsl:stylesheet>
```

Transformation d'arbres XML : XSL

- Autre exemple (date de 2002)
  - À ouvrir uniquement avec IE ou une application *ad hoc*
  - http://liris.cnrs.fr/~lmedini/LIA/LSI/Exam/Septembre/
- Quelques outils
  - Dans un navigateur
    - IE : processeur XSLT incorporé à MSXML (contrôle ActiveX)
    - Gecko, WebKit : objet window.XSLTProcessor
  - En JavaScript : Google AJAXSLT
  - Dans HTML-Kit (éditeur HTML WYSIWYG) : plugin ErgXSLT
  - En Java, intégré à JAXP 2 (J2EE puis JEE5) :
    TrAX Transformation API for XML (javax.xml.transform, xsltc)
  - En C / en PHP : libxslt (PHP 5)

Exemple 3 : Web Components

- Caractéristiques
  - https://www.webcomponents.org/
  - Côté client
  - Poussé par Google (Polymer, AngularJS)
  - Futur standard du W3C
- Objectifs
  - Permettre la création de widgets réutilisables en HTML natif
  - Destiné à être nativement traité dans le navigateur

Exemple 3 : Web Components

- Composants
  - Custom Elements (Working Draft oct. 2016)
    - Permettent de définir ses propres widgets
    - contenus : éléments HTML
    - comportements : API (scripts JS)
  - Shadow DOM (Working Draft sept. 2017)
    - Permet d'associer un sous-arbre DOM à un élément sans qu'il soit pris en compte / rendu par le moteur HTML
    - Mêmes outils que pour le DOM "normal"
  - HTML Imports (Working Draft fev. 2016)
    - Packaging et chargement de templates et custom elements
    - Syntaxe HTML simple (balise <link>)
  - HTML Templates (Group note, mars 2014)
    - Sous-arbres HTML inertes qui peuvent être activés en JS

Exemple 3 : Web Components

- Exemple de code

```
var data = [
  { name: 'Pillar', color: 'Ticked Tabby', sex: 'Female (neutered)', legs: 3 },
  { name: 'Hedral', color: 'Tuxedo', sex: 'Male (neutered)', legs: 4 },
 ];
```

```
<table>
 <thead>
  <tr>
   <th>Name <th>Color <th>Sex <th>Legs
 <tbody>
  <template id="row">
   <tr><td><td><td><td>
  </template>
</table>
```

```
var template = document.querySelector('#row');
 for (var i = 0; i < data.length; i += 1) {
   var cat = data[i];
   var clone = template.content.cloneNode(true);
   var cells = clone.querySelectorAll('td');
   cells[0].textContent = cat.name;
   cells[1].textContent = cat.color;
   cells[2].textContent = cat.sex;
   cells[3].textContent = cat.legs;
   template.parentNode.appendChild(clone);
 }
```

Source : https://html.spec.whatwg.org/multipage/scripting.html#the-template-element

Exemple 3 : Web Components

- Avantages
  - Futur standard du W3C
    Permettra du templating "bas niveau" directement en HTML et CSS (identique sur tous les navigateurs)
  - Pensé en séparant les différentes étapes du templating
  - Techno prometteuse, poussée par Google et Mozilla
  - L'élément template sera intégré à la spec HTML5
- Inconvénients
  - Standard pas encore mûr
  - Nécessite une surcouche (Polymer, X-tag) ?
  - Un peu surdimensionné pour faire uniquement du templating
    - Nécessite d'utiliser toute la spec (tout faire en Web Components)
    - Compatibilité avec les outils choisis

Exemple 4 : Mustache / Handlebars

- Mustache
  - https://mustache.github.io/
  - Logic-less templates
    - Remplacement de données
    - Gestion des tableaux et des dictionnaires
    - Pas de tests ni de boucles
  - Moteur implémenté dans de nombreux langages
    - Java, JS, PHP, Python…
  - Surcouche
    - Handlebars

Exemple 4 : Mustache / Handlebars

- Mustache
  - Exemples (basique) de code

Template

```
Hello {{name}}
You have just won {{value}} dollars!
{{#in_ca}}
Well, {{taxed_value}} dollars, after taxes.
{{/in_ca}}
```

Données

```
{
  "name": "Chris",
  "value": 10000,
  "taxed_value": 10000 - (10000 * 0.4),
  "in_ca": true
}
```

Résultat

```
Hello Chris
You have just won 10000 dollars!
Well, 6000.0 dollars, after taxes.
```

Exemple 4 : Mustache / Handlebars

- Mustache
  - Principaux types de balises
    - Variables : {{name}}
      - Évaluée en fonction de name dans les données JSON
    - Sections : {{#name}} … {{/name}}
      - Blocs dépendant de la valeur de la donnée
      - Si la donnée est un tableau, le moteur itère dessus
    - “Partials” : {{> user}}
      - Permet l’inclusion de templates
  - Remarques
    - Pour l’évaluation des données, le template n’est pas affiché si la donnée n’existe pas, ou est false, vide, nulle, NaN…
    - Les lambdas ou fonctions peuvent être envoyées en tant que données

Exemple 4 : Mustache / Handlebars

- Mustache
  - Utilisation en Java
    - Doc : https://github.com/spullara/mustache.java
    - Maven :

```
<dependency>
  <groupId>com.github.spullara.mustache.java</groupId>
  <artifactId>compiler</artifactId>
  <version>0.9.5</version>
</dependency>
```

    - Exécution :

```
public static void main(String[] args) throws IOException {
    MustacheFactory mf = new DefaultMustacheFactory();
    Mustache mustache = mf.compile("template.mustache");
    mustache.execute(new PrintWriter(System.out), new Example()).flush();
 }
```

Exemple 4 : Mustache / Handlebars

```
public class Example {
  List<Item> items() {
    return Arrays.asList(
      new Item("Item 1", "$19.99", Arrays.asList(new Feature("New!"), new Feature("Awesome!"))),
      new Item("Item 2", "$29.99", Arrays.asList(new Feature("Old."), new Feature("Ugly.")))
    );
  }

  static class Item {
    Item(String name, String price, List<Feature> features) {
      this.name = name;
      this.price = price;
      this.features = features;
    }
    String name, price;
    List<Feature> features;
  }

  static class Feature {
    Feature(String description) {
       this.description = description;
    }
    String description;
  }
}
```

```
{{#items}}
Name: {{name}}
Price: {{price}}
  {{#features}}
  Feature: {{description}}
  {{/features}}
{{/items}}
```

```
Name: Item 1
Price: $19.99
  Feature: New!
  Feature: Awesome!
Name: Item 2
Price: $29.99
  Feature: Old.
  Feature: Ugly.
```

Exemple 4 : Mustache / Handlebars

- Mustache
  - Utilisation en Java
    - Doc : https://github.com/spullara/mustache.java
    - Exemple

Exemple 4 : Mustache / Handlebars

- Mustache
  - Utilisation en Java
    - Doc : https://github.com/spullara/mustache.java
    - Remarques
      - Nécessite Java 8
      - Les données doivent être dans des champs publics d'objets
      - Les listes doivent être des Iterable
      - Les templates doivent être **compilés** pour pouvoir être utilisés

Exemple 4 : Mustache / Handlebars

- Mustache
  - Utilisation avec Spring Web MVC
    - Doc : https://spring.io/blog/2016/11/21/the-joy-of-mustache-server-side-templates-for-the-jvm
    - Implémentation utilisée : JMustache
      - https://github.com/samskivert/jmustache
      - Configuration

```
// Add the following to an @Configuration class
 @Bean
 public ScriptTemplateConfigurer mustacheConfigurer() {
    ScriptTemplateConfigurer configurer = new ScriptTemplateConfigurer();
    configurer.setEngineName("nashorn");
    configurer.setScripts("mustache.js");
    configurer.setRenderObject("Mustache");
    configurer.setRenderFunction("render");
    return configurer;
 }
```

Source : https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/view/script/ScriptTemplateConfigurer.html

Exemple 4 : Mustache / Handlebars

- Mustache
  - Utilisation en JavaScript
    - Côté client ou côté serveur (NodeJS)
    - MustacheJS https://github.com/janl/mustache.js
    - URL d'import (CDN, nov. 2017)

```
https://cdnjs.cloudflare.com/ajax/libs/mustache.js/2.3.0/mustache.min.js
```

    - Voir doc pour outils de gestion des paquets (npm, bower)
    - Exemple d'utilisation (basique)

```
var view = {
  title: "Joe",
  calc: function () { return 2 + 4; }
};

var output = Mustache.render("{{title}} spends {{calc}}", view);
```

Exemple 4 : Mustache / Handlebars

- Mustache
  - Utilisation en JavaScript
    - MustacheJS
      - https://github.com/janl/mustache.js
    - Exemple d'utilisation (avec compilation)

```
//Ahead of time
Mustache.parse(template);

// Then, sometime later.
Mustache.render(template, view);
```

Exemple 4 : Mustache / Handlebars

- Mustache
  - Utilisation en JavaScript
    - MustacheJS
      - https://github.com/janl/mustache.js
    - Remarque :
      - Il est possible de placer un template dans un élément <script> de la page

```
<script id="template" type="x-tmpl-mustache">
Hello {{ name }}!
</script>
```

      - On y accède ensuite avec l'API DOM classique

```
var template = document.getElementById(“template“).innerHTML;
```

Exemple 4 : Mustache / Handlebars

- Mustache
  - Utilisation en JavaScript
    - MustacheJS
      - https://github.com/janl/mustache.js
    - Utilisation avec jQuery
      - Template dans le DOM

```
var template = $('#template').html();
Mustache.parse(template);   // optional, speeds up future uses
var rendered = Mustache.render(template, {name: "Luke"});
$('#target').html(rendered);
```

      - Template téléchargé

```
$.get('template.mst', function(template) {
    var rendered = Mustache.render(template, {name: "Luke"});
    $('#target').html(rendered);
});
```

Exemple 4 : Mustache / Handlebars

- Handlebars
  - Surcouche de MustacheJS
    - “build **semantic templates** effectively with no frustration”
  - Côté client ou côté serveur
  - Docs
    - Getting started : http://handlebarsjs.com/
    - API docs : http://handlebarsjs.com/reference.html
    - Différences avec Mustache : https://github.com/wycats/handlebars.js#differences-between-handlebarsjs-and-mustache

Exemple 4 : Mustache / Handlebars

- Handlebars
  - Getting started : http://handlebarsjs.com/
  - URL d'import (CDN, nov. 2017)

```
https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.11/handlebars.min.js
```

  - Voir doc pour outils de gestion des paquets (npm, bower)
  - Exemple d'utilisation (basique)

```
var context = {title: "My New Post", body: "This is my first post!"}
var html    = template(context);
```

  - Exemple de compilation

```
var source   = document.getElementById("entry-template").innerHTML;
var template = Handlebars.compile(source);
```

    - Remarque : une précompilation peut s'effectuer côté serveur http://handlebarsjs.com/precompilation.html

Exemple 4 : Mustache / Handlebars

- Handlebars
  - http://handlebarsjs.com/
  - Principales différences avec Mustache : https://github.com/wycats/handlebars.js#differences-between-handlebarsjs-and-mustache
    - Nested paths
      - Permettent la “navigation” dans les structures de données JSON

```
<div class="entry">
  <h1>{{title}}</h1>
  <h2>By {{author.name}}</h2>

  <div class="body">
    {{body}}
  </div>
</div>
```

```
var context = {
  title: "My First Blog Post!",
  author: {
    id: 47,
    name: "Yehuda Katz"
  },
  body: "My first post. Wheeeee!"
};
```

Exemple 4 : Mustache / Handlebars

- Handlebars
  - http://handlebarsjs.com/
  - Principales différences avec Mustache : https://github.com/wycats/handlebars.js#differences-between-handlebarsjs-and-mustache
    - Helpers
      - Pour rajouter de la logique applicative (programmatiquement)

```
<div class="post">
  <h1>By {{fullName author}}</h1>
  <div class="body">{{body}}</div>
</div>
```

```
var context = {
  author: {firstName: "Alan", lastName: "Johnson"},
  body: "I Love Handlebars“
};

Handlebars.registerHelper('fullName', function(person) {
  return person.firstName + " " + person.lastName;
});
```

  - Remarque

“Helpers receive the current context as the **this** context of the function.”

Exemple 4 : Mustache / Handlebars

- Handlebars
  - http://handlebarsjs.com/
  - Principales différences avec Mustache : https://github.com/wycats/handlebars.js#differences-between-handlebarsjs-and-mustache
    - Block helpers
      - Pour rajouter de la logique applicative (dans les templates)
      - Doc : http://handlebarsjs.com/block_helpers.html
      - Exemples

```
{{#with story}}
    <div class="intro">{{{intro}}}</div>
    <div class="body">{{{body}}}</div>
{{/with}}
```

```
{{#each comments}}
  <h2>By {{fullName author}}</h2>
  <div class="body">{{body}}</div>
{{/each}}
```

```
{{#if isActive}}
  <img src="star.gif" alt="Active">
{{else}}
  <img src="cry.gif" alt="Inactive">
{{/if}}
```

Conclusion

- Principes du templating (côté client)
  1. Description des templates de manière déclarative
     - À l'intérieur d'une variable ou d'un élément HTML prédéfini (souvent : <script>)
     - À l'aide d'une syntaxe spécifique
  2. Mettre en place un mécanisme d'association template / données
     - S'appuie en général sur des structures de données en JSON
     - Requêtes sur les données en fonction des propriétés des objets
  3. Ajouter le résultat au DOM
     - À l'aide des sous-couches disponibles (jQuery)

Conclusion

- De nombreux outils disponibles
  - Avec chacun leurs spécificités
- Outils de templating eux-même imbriqués dans une “stack” d’autres outils éventuellement (in)compatibles entre eux
  - Langage de programmation
  - Environnement d’exécution
  - Framework
  - Moteur de templates
  - Mise en forme…

➔Grands principes identiques côté client et côté serveur

➔Sachez choisir l’outil adapté à votre problème

Quelques références

- Voir les URLs citées dans les descriptions des différents outils
- Une page Wikipedia (un peu pauvre en novembre 2017) sur le templating pour le Web
- Guide du templating côté client : http://sylvainpv.developpez.com/tutoriels/javascript/guide-templating-client/
- Web components
  - Présentation très complète sur HTML5-demos (à voir avec Chrome)
  - Présentation de Google sur les Web components (à voir avec Chrome)

Suite du CM précédent

- Fetch API…

Fetch API

- Principe
  - Fournir des primitives de plus haut niveau que `XmlHttpRequest`
  - Accepter les réponses streamées (« chunked »)
  - Récupérer du texte ou du JSON (pas de XML)
  - ➔ Encapsuler les requêtes asynchrones dans des promesses
- Exemple (simple)

```
fetch('./monUrl/quiRenvoie/du.json')
.then(res => res.json())
.then(json => console.log(json));
```

Fetch API

- La méthode `fetch()`
  - Seul paramètre obligatoire : URL
  - Le reste est sous forme d'options dans un objet JSON

```
fetch(url, {
  method: "POST", // *GET, POST, PUT, DELETE, etc.
  mode: "cors", // no-cors, cors, *same-origin
  cache: "no-cache", // *default, no-cache, reload, force-cache, only-if-cached
  credentials: "same-origin", // include, *same-origin, omit
  headers: {
    "Content-Type": "application/json; charset=utf-8",
  },
  redirect: "follow", // manual, *follow, error
  referrer: "no-referrer", // no-referrer, *client
  body: JSON.stringify(data), // body data type must match "Content-Type" header
})
```

  - Remarque : il faut explicitement autoriser `fetch()` à envoyer des credentials (cookies…)

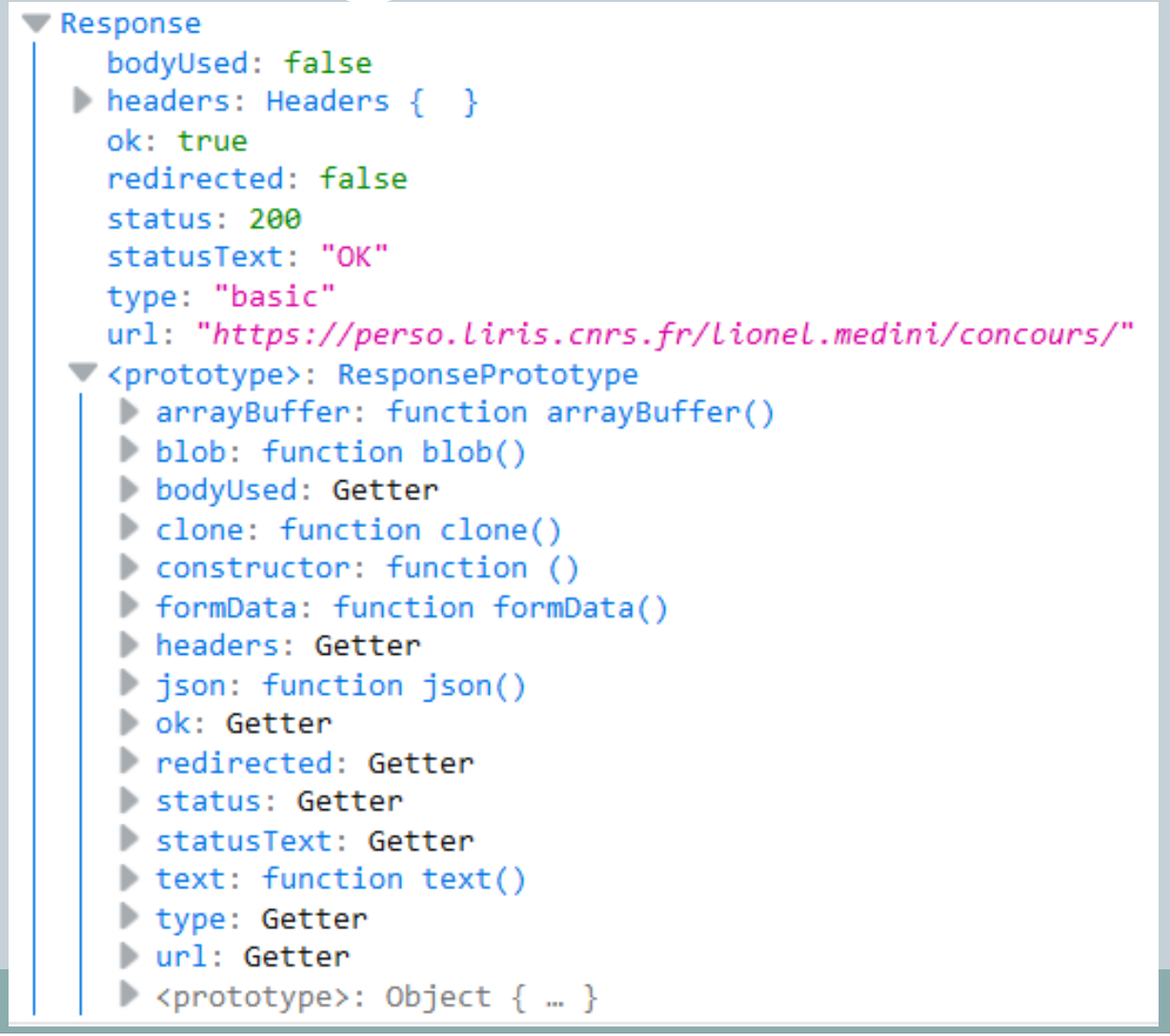
Fetch API

- La méthode `fetch()`
  - Renvoie une réponse
    - Qui peut être wrappée dans différents formats (en fonction du Content-Type)
    - Pas d'erreur (sauf erreur réseau)
      - ➔ Il faut tester le code de retour HTTP
  - La valeur de retour est une promesse
    - Pour permettre l'émission en plusieurs fois (objet `Observable`)

```
>> fetch('https://perso.liris.cnrs.fr/lionel.medini/concours/')
   .then(res => console.log(res.json));
<- ▶ Promise { <state>: "pending" }
   ▶ function json()
```

Fetch API

- Exemple de réponse



Fetch API

- Gestion des erreurs
  - `fetch()` lève une erreur en cas de :
    - Problème réseau
    - Réponse non conforme au résultat attendu (parsing JSON)
  - ...mais pas en cas d'erreur HTTP (404, 500...)

    ➔ C'est à vous de tester si la réponse correspond à vos attentes

```
fetch(https://maRessourceCORS/', {mode: 'cors'})
  .then(response => {
    if(response.ok) { return response.text(); }
    throw new Error('Erreur HTTP : ' + response.status);
  })
  .then(text => { console.log('Corps de la réponse :', text); })
  .catch(error => {
    console.log('Erreur dans la réception de la requête : ', error);
  });
```

Fetch API

- Références
  - Spec
    - https://fetch.spec.whatwg.org/
  - Tutos
    - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
    - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
    - https://developers.google.com/web/updates/2015/03/introduction-to-fetch
    - https://www.sitepoint.com/introduction-to-the-fetch-api/