CSI 3540

Structures, techniques et normes du Web

Séparer la programmation de la présentation à l'aide de JSP

Objectifs :

1. Concevoir des documents JSP
2. Se familiariser avec le langage d'expression (EL) et les JavaBeans

Lectures :

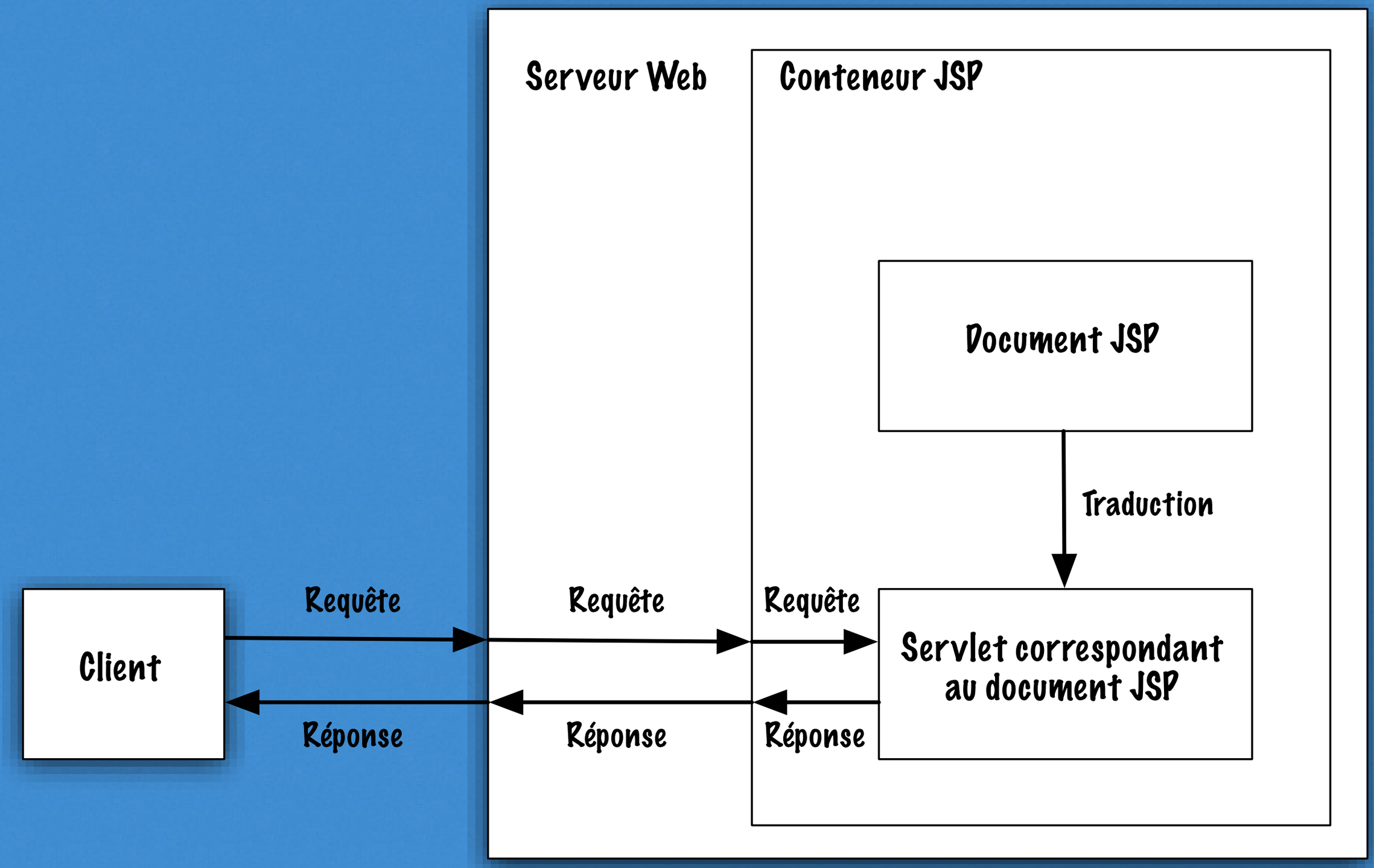
- Web Technologies (2007) § 8
  Pages 447–463

Plan

1. **Expression Language (EL)**
2. **JavaBeans et JSP**
3. **Bibliothèques de balises JSTL**
4. **MVC et JSP**

Introduction

- Les documents **JSP 2.0** (et +) sont des applications **XML**
- Lors de la première visite (ou avant), le document est traduit en Java (Servlet) puis compilé, pour les prochaines visites, la forme compilée est utilisée


Serveur Web
Conteneur JSP
Document JSP
Traduction
Client
Requête
Requête
Requête
Servlet correspondant au document JSP
Réponse
Réponse
Réponse

Introduction

```
<html xmlns:core="http://java.sun.com/jsp/jstl/core"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns="http://www.w3.org/1999/xhtml">

 <jsp:useBean id="date" class="java.util.Date" />

 <head>
  <title>La date du jour</title>
 </head>
 <body style="font-size:x-large">
  <h2>Voici la date du jour :</h2>
  <p>
   <core:out value="${date}" />
  </p>
 </body>
</html>
```

Éléments JSP (3 types)

- **scriptage** : permet l'insertion de code Java dans les documents JSP, par exemple **<jsp:declaration>** (variables d'instance ou de classe), **<jsp:scriptlet>**, **<jsp:expression>**

- **consignes** :
  - **<jsp:directive.page contentType="..."/>**
  - **<jsp:directive.include file="footer.jspf"/>**

```
<html ...>
  ...
  <body>
    <h1>Scriptlet</h1>
    <p>
      <jsp:declaration>
      String date = ( new java.util.Date() ).toString();
      </jsp:declaration>
      <jsp:scriptlet>
      out.write( date );
      </jsp:scriptlet>
    </p>
    <h1>Expression</h1>
    <p>
      <jsp:expression>
      date
      </jsp:expression>
    </p>
  </body>
</html>
```

Éléments JSP (3 types)

- **actions** :
  - Prédéfinies (standard actions)
    - **useBean, setProperty, getProperty, forward...**
  - Définies par l'usager, à l'aide des mécanismes de **JSTL** (JSP Standard Tag Library)

Éléments JSP et les espaces de nommage

```
<html xmlns:core="http://java.sun.com/jsp/jstl/core"
      xmlns:jsp="http://java.sun.com/JSP/Page"
       xmlns="http://www.w3.org/1999/xhtml">

 <jsp:useBean id="date" class="java.util.Date" />
   ....
</html>
```

```
<html xmlns:core="http://java.sun.com/jsp/jstl/core"
      xmlns:page="http://java.sun.com/JSP/Page"
       xmlns="http://www.w3.org/1999/xhtml">

 <page:useBean id="date" class="java.util.Date" />
   ....
</html>
```

Resources

- Une ressource étonnamment **brève** et très utile
  - JavaServer Pages (JSP) v2.0 Syntax Reference [http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html] 2008-03-05
- Référence **complète** (1126 pages)
  - http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf

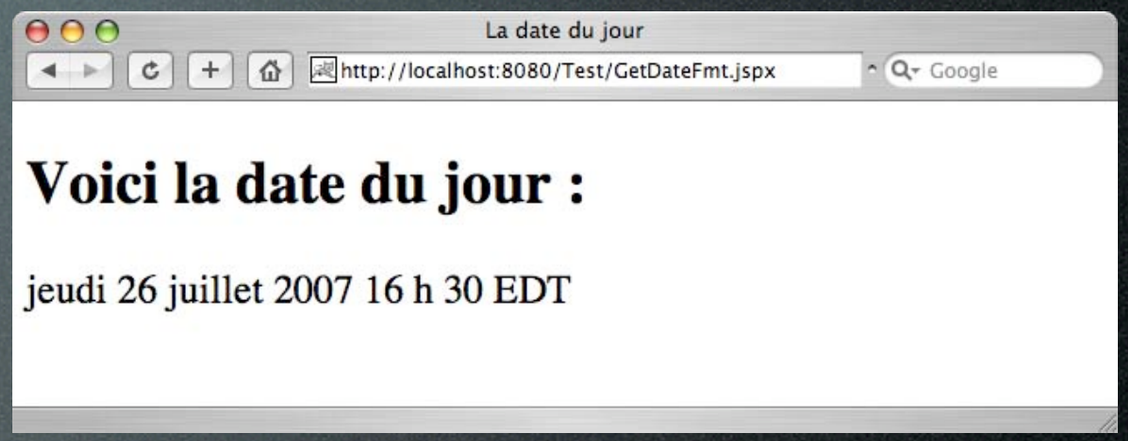
Présentation + programmation

```
<html xmlns:core="http://java.sun.com/jsp/jstl/core"
    xmlns:jsp="http://java.sun.com/JSP/Page"
    xmlns="http://www.w3.org/1999/xhtml">

 <head><title>La date du jour</title></head>
 <body>
  <h2>Voici la date du jour :</h2>
  <p>
   <jsp:scriptlet>

    int fmt = java.text.DateFormat.FULL;
    java.text.DateFormat df;
    df = java.text.DateFormat.getDateTimeInstance( fmt, fmt, java.util.Locale.CANADA_FRENCH );
    out.write( df.format( new java.util.Date() ) );

   </jsp:scriptlet>
  </p>
 </body>
</html>
```



Présentation + programmation

```
<html xmlns:core="http://java.sun.com/jsp/jstl/core"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns="http://www.w3.org/1999/xhtml">

 <jsp:useBean id="date" class="java.util.Date" />

  <head>
   <title>La date du jour</title>
  </head>
  <body style="font-size:x-large">
   <h2>Voici la date du jour :</h2>
   <p>
    <core:out value="${date}" />
   </p>
  </body>
 </html>
```

Présentation + programmation

- Bien utilisés, les documents JSP favorisent la séparation entre la **présentation** et la **programmation** (**business logic**)
- On peut même interdire l'utilisation d'éléments **scriptlet** (et ainsi bannir tous les énoncés Java) à l'aide d'une directive (**isScriptingEnabled**) ou dans le descripteur de déploiement (**web.xml**)

Java Server Page (JSP)

- Afin de séparer présentation et programmation :

  1. **JSP Expression Language (EL)**
  2. **JSP Standard Tag Library (JSTL)**
  3. **JavaBeans**

- **«Java-free (scriptlet-free) documents!»**

Expression Language

(EL)

Unified Expression Language

- **EL** (Expression Language) a été introduit lors du développement de **JSP 2.0**
- Avec **JSP 2.1**, les langages d'expression de **JSF** et **JSP** ont été unifiés

Expression Language (EL)

- **Accès dynamique** (lecture) aux données sauvegardées dans les **objets implicites**, **JavaBeans**, et autres structures de données
- **Accès dynamique** (écriture) aux **JavaBeans**, expressions différées seulement
- Appels de méthodes
- Opérations arithmétiques

Expression Language (EL)

- **EL** est un langage d'**expression** ; donc volontairement un langage assez simple
- Calculs élémentaires **+** accès aux objets côté serveur
- Exemples :
  - **${ 4 < (3/2) }** (**${ 4 lt (3/2) }**)
  - **${ empty param.temperature }**
  - **${ header["host"] }**

Comme le nom le suggère, le langage est très simple, c'est un langage d'expressions!

Expression Language (EL)

- Deux modes d'évaluation :
  - **Évaluation immédiate**
    (associées à **JSP**, lecture seulement)

    ${ ... }
  - Évaluation différée
    (associées à JSF)

    #{ ... }

Évaluation des expressions **EL**

```
..
out.write("<body>");
out.write("<p>");
out.write("Le nombre de visites est ");
out.write( (java.lang.String)
                org.apache.jasper.runtime.PageContextImpl.evaluateExpression("${ visits }",
                                java.lang.String.class, (PageContext)_jspx_page_context, null) );
out.write(".");
out.write("</p>");
out.write("</body>");
...
```

- Une expression **EL** est introduite à l'aide de la construction **${ ... }**
- On retrouve des expressions EL :
  - Comme données textuelles (**contexte statique**)
    - **"Le nombre d'accès est ${ visits }"**
  - Valeurs des **attributs** des balises JSP
    - **page="${ param.nextPage }"**

Expression Language (EL)

- Une expression **EL** valide comprend des littéraux, des opérateurs, des parenthèses, des références d'objets (variables) et des appels de fonctions

- Que manque-t-il?

Aucun mécanisme pour définir des fonctions

Expression Language (EL)

- Littéral (literal):
  - booléens : **true**, **false**
  - entiers et nombres en point flottant
  - chaînes de caractères (protégées par les guillemets simples ou doubles)
  - mot réservé **null**
- Idientifiants : comme Java

Expression Language (EL)

- 16 mots réservés seulement :
  - **and**, **div**, **empty**, **eq**, **false**, **ge**, **gt** **instanceof**, **le**, **lt**, **mod**, **ne**, **not**, **null**, **or**, **true**
- Opérateurs usuels, tels que +, -, /, *, %, etc. ainsi que «(» et «)»

Plusieurs sont des synonymes pour les opérateurs unaires/ binaires <, &&, ...

Évidemment lié au fait que & et < ont une sémantique toute particulière en XML

Expression Language (EL)

Type	**Opérateur**
Arithmétique	+ - * / div % mod
Relationnel	!= ne == eq < lt > gt <= le >= ge

Expression Language (EL)

Type	**Opérateur**
Logique	&& and || or ! not
Autre	() empty instanceof [] .

Expression Language (EL)

- **${ empty nom }** :
  - vrai si nom est une référence dont la valeur est null
  - vrai si nom est une référence vers un objet **String**, **List**, **Map** ou **tableau** qui est vide
  - faux sinon

Expression Language (EL)

- **EL** fournit une vision **élémentaire** et **unifiée** des objets implicites
- Élémentaire parce que **EL** ne permet pas (explicitement) les appels de méthodes
- Unifiée parce les composants logiciels (objets implicites) sont vus comme un ensemble de propriétés peut importe leur type : **Map**, **List**, **tableau** ou **JavaBeans**

Expression Language (EL)

- Les expressions **EL** donnent accès à
  - **JavaBeans**
  - **Collections**
  - **Types énumérés**
  - **Objets implicites**

Évaluation d'une variable

- Si l'identifiant correspond au nom de l'un des objets implicites, alors l'objet implicite correspondant est retourné :
- **pageContext**, **pageScope**, **requestScope**, **sessionScope**, **applicationScope**, **param**, **paramValues**, **header**, **headerValues**, **cookie**, **initParam**

Variables

- Sinon, la recherche se poursuit en considérant les 4 espaces lexicaux suivants : **page**, **request**, **session**, et **application**
- Sinon, si l'identifiant est absent, la valeur **null** est retournée

4 espaces lexicaux

- Recherche l'identifiant dans ces 4 espaces de lexicaux (dans l'ordre) :
  - **page** (local à la page/accès)
  - **request** (spécifiquement pour cette requête)
  - **session** (la session de l'usager)
  - **application** (pour tous les usagers)

Variables

- Depuis **JSP 2.1**, on peut définir ses propres mécanismes de résolution de noms de variables, sauf pour les objets implicites

Expression Language (EL)

- Les composants logiciels ont des **propriétés** (tout comme ECMAScript)
- Deux constructions syntaxiques équivalentes pour l'**accès aux propriétés** (tout comme ECMAScript) :

```
expr.identifiant

expr[ "identifiant" ]
```

- Évaluer **expr-a[expr-b]** :
  - Évaluer **expr-a** afin de produire **value-a**
  - Évaluer **expr-b** afin de produire **value-b**
  - Si **value-a** est de type **Map** alors retourner **value-a.get(value-b)**
  - Si **value-a** est de type **List** ou tableau alors prendre la valeur entière de **value-b et** retourner **Array.get(value-a, value-b)**
  - Si **value-a** est un objet JavaBeans, utiliser la méthode d'accès correspondant à la propriété **value-b** du composant JavaBeans **value-a**

Objets implicites

Traduction de JSP à Servlet

```
public void _jspService( HttpServletRequest request, HttpServletResponse response )
    throws java.io.IOException, ServletException {

  PageContext pageContext = null;
  HttpSession session = null;
  ServletContext application = null;
  ServletConfig config = null;
  JspWriter out = null;
  Object page = this;
    …
  }
}
```

- **_jspService** définit et initialise plusieurs objets que nous utiliserons

- **glassfish/domains/domain1/generated/jsp/OO_HelloWorld_Glassfish_Eclipse/index_jspx.javaorg/apache/jsp/index_jspx.java**

Traduction de JSP à Servlets

```
try {
    response.setContentType("text/html; charset=ISO-8859-1");
    response.setHeader("X-Powered-By", "JSP/2.1");
    pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    ...
 } catch (Throwable t) {
    ...
}
```

index.jspx

```
<html xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns="http://www.w3.org/1999/xhtml">
 <jsp:directive.page contentType="text/html" />
 <jsp:output
   omit-xml-declaration="yes"
   doctype-root-element="html"
   doctype-public="-//W3C//DTD XHTML Basic 1.0//EN"
   doctype-system="http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd" />
  <head><title>La date du jour</title></head>
  <body style="font-size:x-large">
   <h2>Voici la date du jour :</h2>
   <p>
    <jsp:scriptlet>
     out.write( request.getParameter( "nom" ) );
    </jsp:scriptlet>
   </p>
  </body>
</html>
```

Java EE - http://localhost:8080/00_HelloWorld_Glassfish_Eclipse/index.jspx - Eclipse - /Users/turcotte/Documents/workspace

Java EE

Project Explorer

00_HelloWorld_Glassfish_Eclipse

- Deployment Descriptor: 00_HelloWorld_
- JavaScript Resources
- src
  - csi3540
    - HelloWorld.java
- JRE System Library [JVM 1.6.0 (MacOS X
- GlassFish v3 Java EE 6 [GlassFish v3 Java
- Web App Libraries
- EAR Libraries
- build
- WebContent
  - META-INF
  - WEB-INF
  - index.jspx

index.jspx | CSI3540

http://localhost:8080/00_HelloWorld_Glassfish_Eclipse/index.jspx?nom=Marcel

Les objets implicites

Votre nom : Marcel

Outline | Task List

An outline is not available.

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

GlassFish v3 Java EE 6 at localhost [Started, Synchronized]

00_HelloWorld_Glassfish_Eclipse [Synchronized]

Applications Web

- **getServletContext()** retourne le même objet pour tous les documents JSPs et Servlet de l'application (voir objet **application**)
  - Paramètres initialisés dans web.xml (le descripteur de déploiement)
  - **setAttribute()** et **getAttribute()**

index.jspx

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:core="http://java.sun.com/jsp/jstl/core"
      lang="fr-CA">

  <core:if test="${ empty visits }">
    <core:set var="visits" scope="application" value="${ initParam.visits }"/>
  </core:if>
  <core:set var="visits" scope="application" value="${ visits+1 }"/>

  <head>
    <title>CSI3540. Structures, techniques et normes du Web</title>
  </head>

  <body>
    <p>
      Le nombre de visites est ${ visits }.
    </p>
  </body>
</html>
```

Objet	**Classe**
pageContext	PageContext
page	Object
response	HttpServletResponse
request	HttpServletRequest
out	JspWriter
session	HttpSession
application	ServletContext
config	ServletConfig

Objets implicites

- **out** : flux de sortie, on l'utilise pour construire le contenu dynamiques des documents

```
<jsp:scriptlet>
  out.write( <b> + ( new java.util.Date() ).toString() + </b> );
</jsp:scriptlet>
```

Objets implicites

- **session** : tout comme les Servlets, les documents JSP ont un objet **session** afin d'associer des requêtes liées les unes aux autres (et palier au fait que HTTP est un protocole sans état)

```
<jsp:scriptlet>
   if ( session.isNew() ) {
      ...
   }
</jsp:scriptlet>
```

Objets implicites

- **request** : modélise le message requête reçu

```
<jsp:scriptlet>
  if ( request.getParameter( "codePostal") ) {
    ...
  }
</jsp:scriptlet>
```

Objets implicites

- **response** : modélise le message réponse à retourner

```
<jsp:scriptlet>
  response.addCookie( new Cookie( "username", ... ) );
  ...
</jsp:scriptlet>
```

Objets implicites

- **application** : donne accès aux paramètres définis par le descripteur de déploiement (web.xml)

```
<webapp>
  <context-param>
    <param-name>serveur</param-name>
    <param-value>sql.site.uottawa.ca</param-value>
  </context-param>
</webapp>
```

```
application.getInitParameter( "serveur" );
```

Objets implicites

- **config** : donne accès aux paramètres définis par le descripteur de déploiement (web.xml), mais ces paramètres sont spécifiques aux documents JSP

```
<servlet>
  <servlet-name>Elgoog<servlet-name>
  <servlet-class>csi3540.SearchEngine</servlet-class>
  <init-param>
    <param-name>theme</param-name>
    <param-value>snow</param-value>
  </init-param>
</servlet>
```

```
config.getInitParameter( "theme" );
```

Objets implicites

- **exception** : accessible à partir d'une page erreur, donne accès à l'objet **java.lang.Throwable** qui modélise la situation d'erreur

- Objets implicites donnant accès aux **quatre (4) environnements lexicaux**

durée de vie

- **pageScope** : accès aux variables associées à la page
- **requestScope** : accès aux variables associées à la requête (voir **include** et **forward**)
- **sessionScope** : accès aux variables associées à la session
- **applicationScope** : accès aux variables associées à l'application

Remarques

- Les environnements lexicaux **page** et **request** sont associés à un seul filin d'exécution et sont dit **thread-safe**
- Les environnements lexicaux **session** et **application** sont partagés par plusieurs filins d'exécutions et peuvent donc nécessiter l'utilisation de méthodes "**synchronized**"

Objets implicites

- **pageContext** : donne accès à **servletContext**, **session**, **request** et **response**
- **servletContext** : contexte de l'application défini par le fichier web.xml

- Raccourcis pour les objets utilisés fréquemment (accès redondant, alias)
  - **param** : donne accès aux valeurs des paramètres (valeur unique) ; **request.getParameter(...)**
  - **paramValues** : donne accès aux valeurs des paramètres (valeurs multiples - tableau) ; **request.getParameters(...)**

Autrement accessibles à partir de l'objet pageContext et servletContext

- **header** et **headerValues** donne accès aux entêtes (valeur unique, multiples) ; **request.getHeader()**, **getHeaders()**

- **cookie** : accès au témoins **request.getCookies()**

- **initParam** : paramètre dont la valeur tiré du context (context-param lu lors de l'initialisation de l'application) ; **application.getInitParameter()**

Évaluation d'une variable

- Finalement, il y a les objets qui désignent les environnements d'évaluation :
- **pageScope**, **requestScope**, **sessionScope**, **applicationScope**

Exemple d'utilisation d'un objet implicite

- **Problématique** : l'application doit compter le nombre de visites à cette page Web. Lorsqu'on redémarre l'application, on souhaite donner une valeur initiale au compteur de visites

```
./
  ./war/
      WEB-INF/
          classes/
          lib/
          web.xml
      css/
      images/
      index.jspx
```

```
./web/
    WEB-INF/
        lib/
        web.xml
    css/
    images/
    index.jspx
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                             http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">

  <display-name>Compteur de visites</display-name>

  <description>
    Compteur de visites.  Utilise une valeur initiale sauvegardée
    dans le contexte de déploiement.
  </description>

  <context-param>
    <param-name>visits</param-name>
    <param-value>54321</param-value>
  </context-param>

  <welcome-file-list>
    <welcome-file>/index.jspx</welcome-file>
  </welcome-file-list>

</web-app>
```

index.jspx

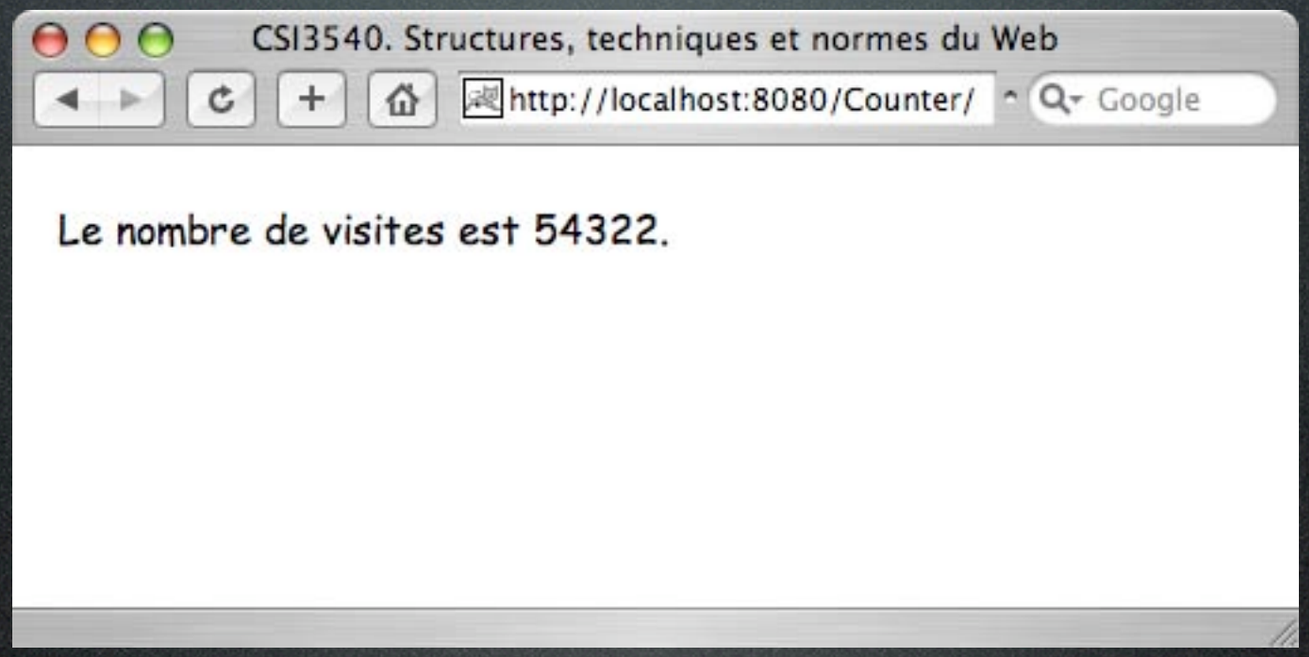
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:core="http://java.sun.com/jsp/jstl/core"
      lang="fr-CA">

  <core:if test="${ empty visits }">
    <core:set var="visits" scope="application" value="${ initParam.visits }"/>
  </core:if>
  <core:set var="visits" scope="application" value="${ visits+1 }"/>

  <head>
    <title>CSI3540. Structures, techniques et normes du Web</title>
  </head>

  <body>
    <p>
      Le nombre de visites est ${ visits }.
    </p>
  </body>
</html>
```

Donner une valeur initiale au compteur



EL et fonctions

- Finalement, **EL** n'a aucun mécanisme pour définir des fonctions
- Les fonctions sont définis soit par des composants **JavaBeans** ou **JSTL**, ou encore des classes Java
- Un appel de fonction : **${ ns:name(x, y) }**

JavaBeans

Une interface simple entre JSP et Java

JavaBeans

- “A Java Bean is a reusable software component that can be manipulated visually in a builder tool.”
- Mais encore!
- C’est un **ensemble de recommendations** qui permettent la **manipulation** des composants par des **outils logiciels**
- Je ne vois toujours pas!

«Simple properties»

- L'une des recommendations est l'adoption du modèle de conception («design pattern») «simple properties»

Les méthodes sont concrètes. Dans le cas de get, la valeur de retour n'est pas void. La lettre qui suit get/set est majuscule.

- La définition d'une méthode **publique**, **sans paramètre**, dont le nom débute par **get** donne lieu à la création automatique d'une propriété, accessible en lecture,

Java: `public PropertyType getPropertyName()`

JSP : `${ bean.propertyName }` ou `${ bean[ "propertyName" ] }`

- De même, la définition d'une méthode **publique**, **n'ayant qu'un paramètre**, dont le nom débute par **set** donne lieu à la création automatique d'une propriété, accessible en écriture,

Dans le cas du set, il n'y a pas de valeur de retour

Java : `public void setPropertyName( PropertyType value )`

JSP : `<core:set target="${ bean }" property="propertyName" value="..." />`

- S'il y a un getter et un setter pour une propriété de même nom, la valeur de retour et le paramètre doivent avoir le même type

JavaBeans : constructeur

- La classe doit posséder un constructeur publique sans paramètres

Java: `public Bean() { ... }`

JSP :< `jsp:useBean scope="session" id="bean" class="pkg.Bean"/>`

JavaBeans : exemple

- **Problématique** : on souhaite paramétrer l'aspect visuel d'un ensemble de pages Web (dynamiques) afin que **1)** ces pages aient le même aspect visuel et que **2)** l'usager puisse choisir les valeurs des différents paramètres

- Nous utiliserons un objet **JavaBean** pour cet usage

JavaBeans : exemple

```
package config;

public class Theme {

    private String backgroundColour;

    public Theme() {
      backgroundColour = "white";
    }

    public String getBackgroundColour() {
      return backgroundColour;
    }

    public void setBackgroundColour( String colour ) {
      backgroundColour = colour;
    }
}
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:core="http://java.sun.com/jsp/jstl/core"
      lang="fr-CA">

  <jsp:useBean scope="session" id="theme" class="config.Theme"/>

  <head>
   <title>CSI3540. Structures, techniques et normes du Web</title
  </head>

  <body bgcolor="${ theme.backgroundColour }">
   <p>
    <b>backgroundColour</b> = ${ theme.backgroundColour }
   </p>
   <p>
    <b>set backgroundColour to</b> green
    <core:set target="${ theme }" property="backgroundColour" value="green" />
   </p>
   <p>
    <a href="index.jspx">retour à la page principale</a>
   </p>
  </body>
</html>
```

theme est le nom de la variable EL

jsp:useBean crée un objet de la classe config.Theme

Dans les éléments textuels on peut lire la valeur d'une propriété, de même dans la valeur d'un attribut

MAJ de la valeur à l'aide de core:set

CSI3540. Structures, techniques et normes du Web

http://localhost:8080/BeanTest/index.jspx

Google

backgroundColour = white

allez à green.jspx
allez à orange.jspx
allez à white.jspx

Open "http://localhost:8080/Be

CSI3540. Structures, techniques et normes du Web

http://localhost:8080/BeanTest/orange.js

Google

backgroundColour = white

set backgroundColour to orange

retour à la page principale

Open "http://localhost:8080/BeanTest/inde

CSI3540. Structures, techniques et normes du Web

http://localhost:8080/BeanTest/index.jsp

Google

backgroundColour = orange

allez à green.jspx
allez à orange.jspx
allez à white.jspx

Open "http://localhost:8080/BeanTest/white.jspx" in a new tab

«Simple properties» (bis)

- Si la valeur de routour est un booléen, le nom de la méthode d'accès peut aussi être **isPropertyName**()

Java : `public boolean isPropertyName()`

JSP : `${ bean.propertyName }`

- Une application Web se compose, généralement, d'un ensemble de Servlets et de documents JSP

- Les composants logiciels «Java-Beans» peuvent tout aussi bien être **créés à partir de Servlets** :

```
import config.Theme;
...
HttpSession session = requete.getSession();
Theme bean = new Theme();
session.setAttribute( "theme", bean );
```

- JSP/EL :

```
${ sessionScope.theme.backgroundColor }
```

useBean

- Considère un seul espace lexical : celui qui est désigné par l'attribut **scope** (portée), sinon l'espace **page** est l'espace lexical par défaut
- **L'objet est créé seulement s'il n'existait pas !**

useBean

- L'élément **useBean** peut avoir un corps non vide
- Les actions du corps seront exécutées, si et seulement si l'action force la création d'un nouveau composant JavaBeans

```
<jsp:useBean scope="session" id="theme" class="config.Theme">
  <core:set target="${theme}" property="backgroundColour" value="red"/>
</jsp:useBean>
```

- Ainsi, lorsqu'on revisite cette page, les actions du corps ne sont pas exécutées à nouveau

useBean

- Conversions de type automatiques

```
<jsp:setProperty name="beanName" property="propName" value="expr"/>
```

- java.lang.Boolean.valueOf(String),
java.lang.Integer.valueOf(String)...

JavaBeans

- L'utilisation des composants JavaBeans ne se limite pas la création d'**associations nom-valeur**
- Les méthodes **get** et **set** peuvent calculer des résultats arbitrairement complexes
- Un composant JavaBeans peut servir d'interface vers une banque de données, par exemple

```
package unit;

public class Temperature {
    private double celsius;
    public Temperature() {
      celsius = 0.0;
    }
    public double getTemperature() {
      return celsius;
    }
    public void setTemperature( double temperature ) {
      celsius = temperature;
    }
    public double getFahrenheit() {
      return celsius * 1.8 + 32.0;
    }
    public void setFahrenheit( double fahrenheit ) {
      celsius = ( fahrenheit - 32.0 ) / 1.8;
    }
}
```

```
<html xmlns="http://www.w3.org/1999/xhtml"  lang="fr-CA">

  <head>
   <title>CSI3540. Structures, techniques et normes du Web</title>
  </head>

  <body>
   <p>
     <form action="http://localhost:8080/Temperature/toFahrenheit.jspx" method="post">
       <div>
        <label>Température en degré Celsius : <input type="text" size="20" name="celsius" /></label><br />
        <input type="submit" value="Convertir en Fahrenheit" />
       </div>
     </form>
    </p>
  </body>
 </html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:jsp="http://java.sun.com/JSP/Page"
    xmlns:core="http://java.sun.com/jsp/jstl/core"
    lang="fr-CA">

 <jsp:useBean scope="session" id="temp" class="unitTemperature"/>
 <core:set target="${temp}" property="temperature" value="${ param.celsius }"/>

 <head>
  <title>CSI3540. Structures, techniques et normes du Web</title>
 </head>

 <body>
  <p>
   ${ temp.temperature } <b>Celsius</b> équivaut à ${ temp.fahrenheit } <b>Fahrenheit</b>
  </p>
  <p>
   <a href="index.jspx">Retour à la page principale</a>
  </p>
 </body>
</html>
```

CSI3540. Structures, techniques et normes du Web

http://localhost:8080/Temperature/

Google

Température en degré Celsius : 22

Convertir en Fahrenheit

CSI3540. Structures, techniques et normes du Web

http://localhost:8080/Temperature/toFahrenheit.jspx

Google

22.0 **Celsius** équivaut à 71.6 **Fahrenheit**

Retour à la page principale

Open "http://localhost:8080/Temperature/index.jspx" in a new tab

Setters/Getters pour tous

- Les mécanismes d'accès (définis de façon implicite) s'appliquent aussi aux objets qui ne sont pas des composants JavaBeans
- Par exemple, l'objet implicite **request** possède une méthode **getPathInfo()**
- On peut faire un appel à cette méthode à partir de JSP comme ceci :
  ${ request.pathInfo }

Résumé

- **EL** est un langage simple donnant une vision **élémentaire** et **unifiée** des objets hôtes côté serveur
- Couplé aux composants **JavaBeans**, ils favorisent la **séparation** entre la **programmation** et **présentation**

Ressources

J2EE 1.4 Tutorial est une excellente source d'information, mais le PDF fait 1542 pages ...

- The J2EE 1.4 Tutorial [http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html] 2007
- The Jakarta Taglibs Project [http://jakarta.apache.org/taglibs/index.html] 2007
- JavaBeans(TM) Specification 1.01 [http://java.sun.com/products/javabeans/docs/spec.html] 2007

Resources

- Une ressource étonnamment brève et très utile
  - JavaServer Pages (JSP) v2.0 Syntax Reference [http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html] 2008-03-05