

JEE

Chapitre 4

Le composant web : **JSP**

Dr. Raoudha SOUABNI

Contact: souabni_raoudha@yahoo.fr

ISTIC

Plan

- 1 Introduction
- 2 Evolution des JSP : EL et JSTL
- 3 EL : Expression Language
- 4 JSTL - Jsp Standard Tag Library

Plan

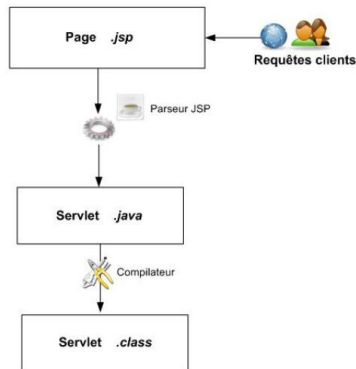
- 1 Introduction
- 2 Evolution des JSP : EL et JSTL
- 3 EL : Expression Language
- 4 JSTL - Jsp Standard Tag Library

Définition

- **JSP** : **J**ava **S**erver **P**age
- Les JSP sont composées à la fois de :
 - Balises HTML
 - Instructions Java
- **Rappel** : Les JSP font office dans le modèle MVC de Vue alors que les Servlets sont les Contrôleurs
- Le code source d'une JSP est transformé, compilé et utilisé comme une servlet.

Fonctionnement

- Lors du premier appel de la page par un utilisateur distant, la page JSP est traduite en une *Servlet Java* par l'intermédiaire du parseur JSP contenu dans le serveur JEE.
- Le compilateur traduit ensuite le fichier Java généré en un fichier *.class*. Le fichier *.class* est alors exécuté par la machine virtuelle.



Fonctionnement

Remarques :

- La transformation de la page *.jsp* en Servlet *.class* n'est effectuée qu'une seule fois, lors du premier appel de la page.
 - ➔ le premier chargement de la page est beaucoup plus long que les suivants
- En cas de modification de la page, celle-ci est à nouveau compilée et mise à jour.

Cycle de vie

Le cycle d'une page JSP comporte quatre phases :

- **Chargement et instanciation** : le serveur localise la classe Java correspondant à la page JSP et la charge.
- **Initialisation** : la page JSP est initialisée selon le principe d'une Servlet.
- **Traitement des requêtes clients** : la page répond aux requêtes des clients. Aucun code Java n'est envoyé aux clients.
- **Fin du cycle** : toutes les requêtes sont traitées et terminées, l'instance de la classe est détruite.

Plan

- 1 Introduction
- 2 Evolution des JSP : EL et JSTL
- 3 EL : Expression Language
- 4 JSTL - Jsp Standard Tag Library

- Première version JSP :
 - ❑ **HTML** + **code Java** entre des balises `< %` et `% >`
 - ❑ C'EST A EVITER : **Ne pas mélanger les langages de balises!!!**
- **JSP 2** : Utiliser les JSP autrement
 - ➔ On n'a plus besoin de Java dans les pages JSP
 - ➔ Utilisation de **Expression Language (EL)** + la librairie **JSP Standard Tag Library (JSTL)**
 - ➔ Développer des pages JSP sans connaissances du langage Java.

Code java Vs EL

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7 <title>Insert title here</title>
8 </head>
9 <body>
10
11 <!-- Code java -->
12 <%
13 String message = request.getAttribute("label").toString();
14 out.println(message);
15 %>
16
17 <!-- Equivalent en EL -->
18 ${label}
19
20 </body>
21 </html>
```

Plan

- 1 Introduction
- 2 Evolution des JSP : EL et JSTL
- 3 EL : Expression Language**
- 4 JSTL - Jsp Standard Tag Library

Pourquoi EL ?

- Limiter la présence du langage Java (non spécialiste)
- Meilleure lisibilité, le code se limite au nom des beans et des propriétés
- Les Expressions Language (EL) permettent de manipuler les données au sein d'une page JSP (essentiellement les beans)
- Une EL permet d'accéder aux beans des différents scopes de l'application (*page*, *request*, *session* et *application*)

Forme d'une expression EL

- Forme : **$\${expression}$**

Une expression correspond à l'expression à interpréter. Elle peut être composée de plusieurs termes séparés par des opérateurs


- **expression**


 **terme1** **opérateur** **terme2**


 **opérateur-unaire** **terme**


 **terme1** **opérateur** **terme2** **opérateur** **terme3** **[(opérateur terme)*]**

- **terme** : Les différents termes peuvent être :

 Un type primaire (Voir "Les types primaires").

 Un objet implicite (Voir "Les objets implicites").

 Un attribut d'un *scope* de l'application (Voir "Les attributs des scopes de l'application").

 Une fonction EL (Voir "Les fonctions").


Forme d'une expression EL

- **opérateur :**

-  **Opérateurs arithmétiques :** + , - , * , / , %


-  **Opérateurs relationnels :** == , != , < , > , <=

-  **Opérateurs logiques :** && , ||

-  **Autres :** ? : (condition ? valeur_si_true : valeur_si_false)

- **opérateur-unaire**

-  **Opérateurs logiques :** ! (not)

-  **Autres :** *empty* (retourne true si l'opérande est *null*, une chaîne vide, un tableau vide, une Map vide ou une List vide. false sinon)

EL - Les types primaires

Les principaux **types primaires** de Java peuvent être utilisés dans les expressions.

Termes	Description	Type
null	La valeur null .	-
123	Une valeur entière.	<code>java.lang.Long</code>
123.00	Une valeur réelle.	<code>java.lang.Double</code>
"chaîne" ou 'chaîne'	Une chaîne de caractère.	<code>java.lang.String</code>
true ou false	Un booléen.	<code>java.lang.Boolean</code>

Remarque : Contrairement au langage Java, les chaînes de caractères peuvent être représentées à la fois avec des quotes simples ' ' ou doubles "

EL - Les objets implicites

Les objets implicites permettent d'accéder aux composants d'une JSP :

- **pageContext** : accès à l'objet PageContext
- **pageScope*** : accès aux différents attributs du scope « page »
- **requestScope*** : accès aux différents attributs du scope « request »
- **sessionScope*** : accès aux différents attributs du scope « session »
- **applicationScope*** : accès aux différents attributs du scope « application »
- **param*** : accès aux paramètres de la requête HTTP
- **paramValues** : paramètres de la requête sous la forme d'un tableau String
- **cookie*** : accès aux différents cookies

* : Objet de type Map

EL - Les attributs des scopes de l'application

- Lors de l'évaluation d'un terme, si celui ci n'est ni un *type primaire*, ni un *objet implicite*, le conteneur web recherchera alors un *attribut* du même nom dans les différents *scopes de l'application* en utilisant la méthode `findAttribute()` du `PageContext` de la page JSP.

Ainsi, l'expression suivante :

```
EL  
${ name }
```

Correspond au code suivant :

```
scriptlet  
<%= pageContext.findAttribute ("name"); %>
```

- L'attribut sera recherché successivement dans les différents scopes (dans l'ordre : **page**, **request**, **session**, **application**).

EL - Accès aux propriétés des objets

- Les *méthodes accesseurs* sont utilisées pour accéder aux propriétés d'un bean.
- Pour accéder à la propriété *name*, la méthode *getName()* est recherchée puis appelée.
- Il y a deux manières d'accéder aux propriétés d'un bean :
 - en utilisant l'opérateur point (`.`)
 - en utilisant l'opérateur crochet (`[]`)

Exemple :

Opérateur 'point'

```
${ bean.name }
```

Opérateur 'crochet'

```
${ bean["name"] }  
ou  
${ bean['name'] }
```

EL par l'exemple

Exemple 1 :

- Créer un projet web dynamique contenant une servlet *Access.java* et 2 pages JSP *Authentification.jsp* et *Display.jsp*
- La page *Authentification.jsp* contient un formulaire ayant 2 champs : login et mot de passe
- La servlet permet de contrôler les données saisies par l'utilisateur :
 - ❑ Renvoyer l'utilisateur à la page d'authentification avec affichage d'un message d'erreur si un champ est manquant
 - ❑ Créer une session de l'utilisateur contenant son login, créer un cookie contenant la date de sa dernière connexion et renvoyer l'utilisateur à la page *Display.jsp* si les deux champs login et mot de passe sont saisis
 - ❑ Utiliser un *requestDispatcher* pour renvoyer la requête vers la page jsp adéquate

- La page *Display.jsp* permet d'afficher au sein d'un formulaire html les informations récupérées de la base concernant l'utilisateur (nom, prénom et email) ainsi que la date de sa dernière connexion

http://localhost:8080/Authentication/Access?login=raoudha&pwd=souabni

Date de la dernière connexion de raoudha: Wed Feb 07 20:58:45 CET 2018

Session ID : AGrTgewMrXedOCHZED2ccUjnBGaUlbDqaaMO7zhS.pc-de-raoudha

Nom :

Prenom :

E-mail :

Plan

- 1 Introduction
- 2 Evolution des JSP : EL et JSTL
- 3 EL : Expression Language
- 4 JSTL - Jsp Standard Tag Library

Pourquoi JSTL ?

JSTL : **J**ava server page **S**tandard **T**ag **L**ibrary.

- Permet de simplifier le développement des pages JSP (la couche *présentation* - les *vues*)
- Permet de concevoir des pages dynamiques complexes sans connaissances du langage Java.
- Permet de développer des pages JSP en utilisant des balises XML (syntaxe proche des langages utilisés par les web designers)

Présentation

- C'est un ensemble de tags personnalisés (*Custom Tags*)
- Utilisation conjointe avec les Expressions Language (EL)
- Propose des fonctionnalités souvent rencontrées dans les JSP :
 - Tag de structure (itération, conditionnement ...)
 - Internationalisation
 - Exécution de requêtes SQL
 - Utilisation de documents XML
- Les fonctionnalités de JSTL sont regroupées dans 5 bibliothèques de tags :

Librairie	Rôle
core	Fonctions de base
Format	Internationalisation
XML	Traitements XML
SQL	Traitements SQL
Fonctions	Fonctions EL

La bibliothèque Core

La bibliothèque Core comporte les actions de base pour la gestion des variables de scope d'une application web :

- Affichage de variable, création, modification et suppression de variables de scope et gestion des exceptions
- Actions conditionnelles et boucles
- Manipulation d'URL, redirection

La bibliothèque Core

La bibliothèque Core comporte les actions de base pour la gestion des variables de scope d'une application web :

- Affichage de variable, création, modification et suppression de variables de scope et gestion des exceptions
- Actions conditionnelles et boucles
- Manipulation d'URL, redirection

L'importation de la bibliothèque Core dans les JSP se fait par la directive suivante :

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
```

La bibliothèque Core - Manipulation des Variables

- **Balise** : `<c:out>`
- **Rôle** : Afficher une expression
- **Attributs** :
 - **Object value** : l'expression qui sera évaluée (obligatoire)
 - **Object default** : valeur à afficher si l'expression « *value* » est null
 - **boolean escapeXml** : détermine si les caractères `<`, `>`, `&`, `'` et `"` doivent être remplacés par leurs codes respectifs

La bibliothèque Core - Manipulation des Variables

- **Balise** : `<c :out>`
- **Rôle** : Afficher une expression
- **Attributs** :
 - **Object value** : l'expression qui sera évaluée (obligatoire)
 - **Object default** : valeur à afficher si l'expression « *value* » est null
 - **boolean escapeXml** : détermine si les caractères `<`, `>`, `&`, `'` et `"` doivent être remplacés par leurs codes respectifs

Exemple :

```
<c :out value="{sessionScope.compte}" default="New user" />
```

La bibliothèque Core - Manipulation des Variables

- **Balise** : `<c :out>`
- **Rôle** : Afficher une expression
- **Attributs** :
 - **Object value** : l'expression qui sera évaluée (obligatoire)
 - **Object default** : valeur à afficher si l'expression « *value* » est null
 - **boolean escapeXml** : détermine si les caractères `<`, `>`, `&`, `'` et `"` doivent être remplacés par leurs codes respectifs

Exemple :

```
<c :out value="{sessionScope.compte}" default="New user" />
```

Remarque :

Le **corps** du tag peut être utilisé à la place de l'attribut **default**.

La bibliothèque Core - Manipulation des Variables

- **Balise** : `<c :out>`
- **Rôle** : Afficher une expression
- **Attributs** :
 - **Object value** : l'expression qui sera évaluée (obligatoire)
 - **Object default** : valeur à afficher si l'expression « *value* » est null
 - **boolean escapeXml** : détermine si les caractères `<`, `>`, `&`, `'` et `"` doivent être remplacés par leurs codes respectifs

Exemple :

```
<c :out value="$ {sessionScope.compte}" default="New user" />
```

Remarque :

Le **corps** du tag peut être utilisé à la place de l'attribut **default**.

Exemple :

```
<c :out value="$ {sessionScope.compte}" >
```

New user

```
</c :out>
```

La bibliothèque Core - Manipulation des Variables

- **Balise :** `<c:set>`
- **Rôle :** Définir une variable de scope ou une propriété
- **Attributs :**
 - **Object value** : l'expression à évaluer
 - **String var** : nom de l'attribut qui contiendra l'expression dans le scope
 - **String scope** : nom du scope qui contiendra l'attribut **var** (page, request, session ou application)
 - **String property** : nom de la propriété qui sera modifiée
 - **Object target** : l'objet dont la propriété définie par « property » sera modifiée

La bibliothèque Core - Manipulation des Variables

Exemple : Définition et ajout d'un attribut *message* dans l'objet *request* dont la valeur est "Bonjour"

```
<c :set value="Bonjour" var="message" scope="request" />
```

La bibliothèque Core - Manipulation des Variables

Exemple : Définition et ajout d'un attribut *message* dans l'objet *request* dont la valeur est "Bonjour"

```
<c :set value="Bonjour" var="message" scope="request" />
```

Exemple : Modification du nom (propriété *nom*) de l'objet *request.user*

```
<c :set value="Mohamed" property="nom"  
target="${requestScope.user}" />
```


La bibliothèque Core - Manipulation des Variables

- **Balise** : `<c:remove>`
- **Rôle** : Supprimer une variable donnée du scope indiqué
- **Attributs** :
 - **String var** : nom de la variable à supprimer (obligatoire)
 - **String scope** : nom du scope qui contient l'attribut *var* (page, request, session ou application)

La bibliothèque Core - Manipulation des Variables

- **Balise** : `<c:remove>`
- **Rôle** : Supprimer une variable donnée du scope indiqué
- **Attributs** :
 - **String var** : nom de la variable à supprimer (obligatoire)
 - **String scope** : nom du scope qui contient l'attribut `var` (page, request, session ou application)

Remarque :

Le corps de la balise **remove** ne contient aucune information

La bibliothèque Core - Manipulation des Variables

- **Balise** : `<c:remove>`
- **Rôle** : Supprimer une variable donnée du scope indiqué
- **Attributs** :
 - **String** `var` : nom de la variable à supprimer (obligatoire)
 - **String** `scope` : nom du scope qui contient l'attribut `var` (page, request, session ou application)

Remarque :

Le corps de la balise **remove** ne contient aucune information

Exemple :

```
<c:remove var="message" scope="request" />
```

La bibliothèque Core - Les actions conditionnelles

- **Balise** : `<c :if>`
- **Rôle** : Effectuer un traitement de la même manière que le mot-clef **if** du langage Java
- **Attributs** :
 - **boolean test** : condition qui détermine si le corps est évalué ou pas (obligatoire)
 - **String var** : nom d'une variable de type boolean contenant le résultat du test
 - **String scope** : valeur du scope pour l'attribut var

Exemple :

```
<c :if test="$ { !empty(param['page']) }" var="valeurTest"
scope="page" >
```

Traitement à faire si la condition du test est valide


```
</c :if>
```

La bibliothèque Core - Les actions conditionnelles

- **Balise** : `<c :choose>`
- **Rôle** : Permet d'effectuer un traitement conditionnel exclusif de la même manière que le mot-clef **switch** du langage Java
- **Attributs** : Elle ne dispose pas d'attribut et le corps peut comporter :
 - Une ou plusieurs balises de type `<c :when>`
 - Zéro ou une balise `<c :otherwise>`

La bibliothèque Core - Les actions conditionnelles

- **Balise** : `<c :choose>`
 - **Rôle** : Permet d'effectuer un traitement conditionnel exclusif de la même manière que le mot-clef **switch** du langage Java
 - **Attributs** : Elle ne dispose pas d'attribut et le corps peut comporter :
 - Une ou plusieurs balises de type `<c :when>`
 - Zéro ou une balise `<c :otherwise>`
- `<c :choose>` : exécute le corps du premier tag `<when>` dont la condition de test est *vraie*.
- `<c :otherwise>` : est exécutée si aucune des conditions n'est vérifiée

La bibliothèque Core - Les actions conditionnelles

- **Balise** : `<c :choose>`
 - **Rôle** : Permet d'effectuer un traitement conditionnel exclusif de la même manière que le mot-clef **switch** du langage Java
 - **Attributs** : Elle ne dispose pas d'attribut et le corps peut comporter :
 - Une ou plusieurs balises de type `<c :when>`
 - Zéro ou une balise `<c :otherwise>`
- `<c :choose>` : exécute le corps du premier tag `<when>` dont la condition de test est *vraie*.
- `<c :otherwise>` : est exécutée si aucune des conditions n'est vérifiée
- La balise `<c :when>` dispose d'un attribut **boolean test** : condition qui détermine si le corps doit être évalué ou non (obligatoire)
- La balise `<c :otherwise>` ne dispose pas d'attribut

La bibliothèque Core - Les actions conditionnelles

Exemple :

```
<c :set var="level" value="2" scope="request" />
<c :choose>
  <c :when test="$ {level==1}">Niveau débutant </c :when>
  <c :when test="$ {level==2}">Niveau intermédiaire </c :when>
  <c :when test="$ {level==3}">Niveau confirmé </c :when>
  <c :when test="$ {level==4}">Niveau expert </c :when>
  <c :otherwise>Niveau non reconnu : $ {level}</c :otherwise>
</c :choose>
```


La bibliothèque Core - Les boucles

- **Balises** : `<c:forEach>` et `<c:forEachTokens>`
- **Rôle** : Effectuer un traitement itératif de la même manière que le mot-clef **for** et **while** du langage Java
- **Attributs en commun** :
 - **String var** : variable qui comporte l'élément courant de l'itération
 - **int begin** : spécifie l'index de départ de l'itération
 - **int end** : spécifie l'index de fin de l'itération
 - **int step** : l'itération s'effectue sur les step éléments de la collection

La bibliothèque Core - Les boucles

- La balise **<c :forEach>** permet d'effectuer des itérations sur plusieurs types de collection de données
- Elle dispose d'un attribut :
 - **Object items** : collection d'éléments sur lesquels s'effectue l'itération

La bibliothèque Core - Les boucles

- La balise **<c :forEach>** permet d'effectuer des itérations sur plusieurs types de collection de données
- Elle dispose d'un attribut :
 - **Object items** : collection d'éléments sur lesquels s'effectue l'itération

L'attribut **items** accepte les éléments suivant :

- les **tableaux** d'objets ou les tableaux de types primaires
- les objets de type **Collection**
- les objets de type **Iterator**
- les objets de type **Enumeration**
- les objets de type **Map**
- *null* est considérée comme une collection vide

La bibliothèque Core - Les boucles

- La balise **<c :forEach>** permet d'effectuer des itérations sur plusieurs types de collection de données
- Elle dispose d'un attribut :
 - **Object items** : collection d'éléments sur lesquels s'effectue l'itération

L'attribut **items** accepte les éléments suivant :

- les **tableaux** d'objets ou les tableaux de types primaires
 - les objets de type **Collection**
 - les objets de type **Iterator**
 - les objets de type **Enumeration**
 - les objets de type **Map**
- *null* est considérée comme une collection vide
- Si l'attribut *items* est absent, *begin* et *end* permettent d'effectuer une itération entre deux nombres entiers

La bibliothèque Core - Les boucles

Exemple 1 : Affichage des paramètres de l'objet *request* sous forme de
nomParam = valeurParam

```
<c :forEach var="elt" items="{param}" >  
<c :out value="{elt.key} = {elt.value}" ></c :out>  
</c :forEach>
```

La bibliothèque Core - Les boucles

Exemple 1 : Affichage des paramètres de l'objet *request* sous forme de
nomParam = valeurParam

```
<c :forEach var="elt" items="{param}" >  
<c :out value="{elt.key} = {elt.value}" ></c :out>  
</c :forEach>
```

Exemple 2 : Affichage des deux premiers paramètres de l'objet *request*
sous forme de **nomParam = valeurParam**

```
<c :forEach var="elt" items="{param}" begin="0" end="1" >  
<c :out value="{elt.key} = {elt.value}" ></c :out>  
</c :forEach>
```

La bibliothèque Core - Les boucles

Exemple 1 : Affichage des paramètres de l'objet *request* sous forme de
nomParam = valeurParam

```
<c :forEach var="elt" items="${param}" >  
<c :out value="${elt.key} = ${elt.value}" ></c :out>  
</c :forEach>
```

Exemple 2 : Affichage des deux premiers paramètres de l'objet *request*
sous forme de **nomParam = valeurParam**

```
<c :forEach var="elt" items="${param}" begin="0" end="1" >  
<c :out value="${elt.key} = ${elt.value}" ></c :out>  
</c :forEach>
```

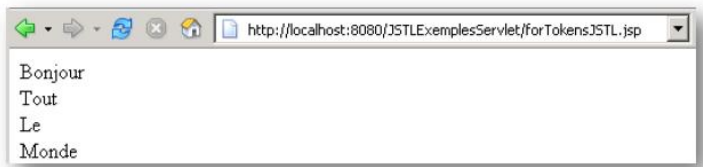
Exemple 3 : Affichage des entiers de 0 à 9

```
<c :forEach var="elt" begin="0" end="9" >  
<c :out value="${elt}"/>  
</c :forEach>
```

La bibliothèque Core - Les boucles

- La balise **<c :forTokens>** permet de découper des chaînes de caractères selon un ou plusieurs *délimiteurs*
- Elle dispose des attributs :
 - **String items** : collection d'éléments contenant les éléments de l'itération
 - **String delims** : la liste des caractères qui serviront de *délimiteurs* (obligatoire)

```
<c:forTokens var="content" items="Bonjour Tout Le Monde" delims=" ">  
  ${content}<br/>  
</c:forTokens>
```



La bibliothèque Core - Les URL

- **Balise** : `<c:url>`
- **Rôle** : Créer des URL absolues
- **Attributs** :
 - **String value** : l'URL à traiter (obligatoire)
 - **Object context** : spécifie le chemin du contexte de l'application locale
 - **Object var** : le nom de la variable scope qui contiendra l'URL
 - **Object scope** : nom du scope

La bibliothèque Core - Les URL

- **Balise** : `<c:url>`
- **Rôle** : Créer des URL absolues
- **Attributs** :
 - **String value** : l'URL à traiter (obligatoire)
 - **Object context** : spécifie le chemin du contexte de l'application locale
 - **Object var** : le nom de la variable scope qui contiendra l'URL
 - **Object scope** : nom du scope
- La balise `<c:param>` permet d'ajouter un paramètre à une URL représentée par la balise parente.
- Cette balise contient différents attributs
 - **String name** : nom du paramètre de l'URL (obligatoire)
 - **String value** : valeur du paramètre de l'URL

La bibliothèque Core - Les URL

- **Balise** : `<c:redirect>`
- **Rôle** : redirection HTTP au client
- **Attributs** :
 - **String url** : l'url de redirection (obligatoire)
 - **String context** : spécifie le chemin du contexte de l'application locale à utiliser (début obligatoirement par `< / >`)