Java Server Pages

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/∼duvallet/

Java Server Pages

Présentations des JSP (1/2)

- Les JSP permettent d'introduire du code Java dans des tags prédéfinis à l'intérieur d'une page HTML.
- La technologie JSP mélange la puissance de Java côté serveur et la facilité de mise en page d'HTML côté client.
- Sun fourni de nombreuses informations sur la technologie JSP à l'adresse suivante : http ://java.sun.com/products/jsp/index.html
- Une JSP est habituellement constituée :
  - de données et de tags HTML,
  - de tags JSP,
  - de scriptlets (code Java intégré à la JSP).
- Les fichiers JSP possèdent par convention l'extension .jsp.
- Concrètement, les JSP sont basées sur les servlets.

Présentations des JSP (2/2)

- Au premier appel de la page JSP, le moteur de JSP génère et compile automatiquement une servlet qui permet la génération de la page web.
- Le code HTML est repris intégralement dans la servlet. Le code Java est inséré dans la servlet.
- La servlet générée est compilée et sauvegardée puis elle est exécutée.
- Les appels suivants de la JSP sont beaucoup plus rapides car la servlet, conservée par le serveur, est directement exécutée.
- Il y a plusieurs manières de combiner les technologies JSP, les beans/EJB et les servlets en fonction des besoins pour développer des applications web.
- Comme le code de la servlet est généré dynamiquement, les JSP sont relativement difficiles à déboguer.

Avantages de l'utilisation des JSP

Il existe plusieurs avantages :

- L'utilisation de Java par les JSP permet une indépendance de la plate-forme d'exécution mais aussi du serveur web utilisé.
- La séparation des traitements et de la présentation : la page web peut être écrite par un designer et les tags Java peuvent être ajoutés ensuite par le développeur.
- Les traitements peuvent être réalisés par des composants réutilisables (des Java beans).
- Les JSP sont basées sur les servlets : tout ce qui est fait par une servlet pour la génération de pages dynamiques peut être fait avec une JSP.

JSP versus Servlet

- Les servlets et les JSP ont de nombreux points communs puisque qu'une JSP est finalement convertie en une servlet.
- Le choix d'utiliser l'une ou l'autre de ces technologies ou les deux doit être fait pour tirer le meilleur parti de leurs avantages.
- Dans une servlet, les traitements et la présentation sont regroupés.
- L'aspect présentation est dans ce cas pénible à développer et à maintenir à cause de l'utilisation répétitive de méthodes pour insérer le code HTML dans le flux de sortie.
- De plus, une simple petite modification dans le code HTML nécessite la recompilation de la servlet.
- Avec une JSP, la séparation des traitements et de la présentation rend ceci très facile et automatique.
- Il est préférable d'utiliser les JSP pour générer des pages web dynamiques.

Les Tags JSP

Il existe trois types de tags :

- tags de directives : ils permettent de contrôler la structure de la servlet générée
- tags de scripting : ils permettent d'insérer du code Java dans la servlet
- tags d'actions : ils facilitent l'utilisation de composants

Attention : Les noms des tags sont sensibles à la case.

Les tags de directives

<%@ ... %>

- Les directives permettent de préciser des informations globales sur la page JSP.
- Les spécifications des JSP définissent trois directives :
  - page : permet de définir des options de configuration.
  - include : permet d'inclure des fichiers statiques dans la JSP avant la génération de la servlet.
  - taglib : permet de définir des tags personnalisés.
- Leur syntaxe est la suivante :
  <%@ directive attribut="valeur" ... %>

La directive page

- Cette directive doit être utilisée dans toutes les pages JSP : elle permet de définir des options qui s'appliquent à toute la JSP.
- Elle peut être placée n'importe où dans le source mais il est préférable de la mettre en début de fichier, avant même le tag <HTML>.
- Elle peut être utilisée plusieurs fois dans une même page mais elle ne doit définir la valeur d'une option qu'une seule fois, sauf pour l'option import.
- Les options définies par cette directive sont de la forme option=valeur.
  - autoFlush, buffer, contentType, errorPage, extends, import, info, isErrorPage, isThreadSafe, langage, session.
- Exemple :

```
<%@ page import="java.util.*" %>
<%@ page import="java.util.Vector" %>
<%@ page info="Ma premiere JSP"%>
```


La directive include (1/2)

- Cette directive permet d'inclure un fichier dans le code source JSP.
- Le fichier inclus peut être un fragment de code JSP, HTML ou Java.
- Le fichier est inclus dans la JSP avant que celle ci ne soit interprétée par le moteur de JSP.
- Ce tag est particulièrement utile pour insérer un élément commun à plusieurs pages tel qu'un en-tête ou un bas de page.
- Si le fichier inclus est un fichier HTML, celui ci ne doit pas contenir de tag <HTML>, </HTML>, <BODY> ou </BODY> qui ferait double emploi avec ceux présents dans le fichier JSP.
- Ceci impose d'écrire des fichiers HTML particuliers uniquement pour être inclus dans les JSP : ils ne pourront pas être utilisés seuls.

La directive include (2/2)

- La syntaxe est la suivante :
  <%@ include file="chemin relatif du fichier" %>
- Si le chemin commence par un '/', alors le chemin est relatif au contexte de l'application, sinon il est relatif au fichier JSP.
- Exemple :

```
<HTML>
  <HEAD>
    <TITLE>Essai de page JSP</TITLE>
  </HEAD>
  <BODY>
    <p align="center">Test d'inclusion d'un fichier dans la JSP</p>
    <%@ include file="monfichier.html"%>
  </BODY>
</HTML>
```


Les tags de scripting

- Ces tags permettent d'insérer du code Java qui sera inclus dans la servlet générée à partir de la JSP.
- Il existe trois tags pour insérer du code Java :
  - Le tag de déclaration : le code Java est inclus dans le corps de la servlet générée. Ce code peut être la déclaration de variables d'instances ou de classes ou la déclaration de méthodes.
  - Le tag d'expression : évalue une expression et insère le résultat sous forme de chaîne de caractères dans la page web générée.
  - Le tag de scriptlets : par défaut, le code Java est inclus dans la méthode service() de la servlet.
- Il est possible d'utiliser dans ces tags plusieurs objets définis par les JSP.

Le tag de déclaration (1/2)

`<%! ... %>`

- Ce tag permet de déclarer des variables ou des méthodes qui pourront être utilisées dans la JSP.
- Il ne génère aucun caractère dans le fichier HTML de sortie.
- La syntaxe est la suivante :
  <%! declarations %>
- Exemple :

```
<%! int i = 0; %>
<%! dateDuJour = new java.util.Date(); %>
```

- Les variables ainsi déclarées peuvent être utilisées dans les tags d'expressions et de scriptlets.
- Il est possible de déclarer plusieurs variables dans le même tag en les séparant avec des caractères ' ;'.

Le tag de déclaration (2/2)

- Ce tag permet aussi d'insérer des méthodes dans le corps de la servlet.
- Exemple :

```
<HTML>
  <HEAD>
    <TITLE>Test</TITLE>
  </HEAD>
  <BODY>
  <%!
    int minimum(int val1, int val2) {
    if (val1 < val2) return val1;
    else return val2;
    }
  %>
  <% int petit = minimum(5,3);%>
  <p>Le plus petit de 5 et 3 est <%= petit %></p>
  </BODY>
</HTML>
```


Le tag d'expressions

`<%= ... %>`

- Le moteur de JSP remplace ce tag par le résultat de l'évaluation de l'expression présente dans le tag.
- Ce résultat est toujours converti en une chaîne.
- Ce tag est un raccourci pour éviter de faire appel à la méthode `println()` lors de l'insertion de données dynamiques dans le fichier HTML.
- La syntaxe est la suivante : **<%= expression %>**
- Le signe '=' doit être collé au signe '%'.

Attention : il ne faut pas mettre de ' ;' à la fin de l'expression.

Le tag d'expressions

- Exemple : insertion de la date dans une page HTML.

```
<%@ page import="java.util.*" %>
<HTML>
  <HEAD>
    <TITLE>Essai de page JSP</TITLE>
  </HEAD>
  <BODY>
    <p align="center">Date du jour :
      <%= new Date() %>
    </p>
  </BODY>
</HTML>
```

- Résultat :

```
Date du jour : Tue May 11 17:02:30 CEST 2010
```

Les variables implicites

Les spécifications des JSP définissent plusieurs objets utilisables dans le code dont les plus utiles sont :

- L'objet `out` de la classe `javax.servlet.jsp.JspWriter` permet de générer un flux en sortie de la page HTML.
- L'objet `request` de la classe `javax.servlet.http.HttpServletRequest` contient les informations de la requête.
- L'objet `response` de la classe `javax.servlet.http.HttpServletResponse` contient les informations de la réponse.
- L'objet `session` de la classe `javax.servlet.http.HttpSession` gère la session.

Le tag des scriptlets (1/2)

`<% ... %>`

- Ce tag contient du code Java nommé un scriptlet.
- La syntaxe est la suivante : `<% code Java %>`.
- Exemple :

```
<%@ page import="java.util.Date"%>
<html>
  <BODY>
    <%! Date dateDuJour; %>
    <% dateDuJour = new Date();%>
    Date du jour : <%= dateDuJour %><BR>
  </BODY>
</html>
```

- Par défaut, le code inclus dans le tag est inséré dans la méthode `service()` de la servlet générée à partir de la JSP.
- Ce tag ne peut pas contenir autre chose que du code Java : il ne peut pas par exemple contenir de tags HTML ou JSP.

Le tag des scriptlets (2/2)

- Pour faire cela, il faut fermer le tag du scriptlet, mettre le tag HTML ou JSP puis de nouveau commencer un tag de scriptlet pour continuer le code.
- Exemple :

```
<HTML>
  <HEAD>
    <TITLE>Essai de page JSP</TITLE>
  </HEAD>
  <BODY>
    <% for (int i=0; i<10; i++) { %>
    <%= i %> <br>
    <% }%>
  </BODY>
</HTML>
```


Les tags de commentaires

- Il existe deux types de commentaires avec les JSP :
  - Les commentaires visibles dans le code HTML :
    - Ces commentaires sont ceux définis par format HTML.
    - Ils sont intégralement reconduits dans le fichier HTML généré.
    - Il est possible d'insérer, dans ce tag, un tag JSP de type expression qui sera exécuté.
    - La syntaxe est la suivante :
      `<!- - commentaires [ <%= expression %> ] - ->`
  - Les commentaires invisibles dans le code HTML :
    - Les commentaires cachés sont utilisés pour documenter la page JSP.
    - Leur contenu est ignoré par le moteur de JSP et ne sont donc pas reconduits dans la page HTML générée.
    - La syntaxe est la suivante : `<%- - commentaires - -%>`

Les tags d'actions

Les tags d'actions permettent de réaliser des traitements couramment utilisés. Nous allons voir :

- Le tag `<jsp:useBean>`
- Le tag `<jsp:setProperty>`
- Le tag `<jsp:getProperty>`
- Le tag de redirection `<jsp:forward>`
- Le tag `<jsp:include>`
- Le tag `<jsp:plugin>`

Le tag `<jsp:useBean>` (1/6)

- Le tag `<jsp:useBean>` permet de localiser une instance ou d'instancier un bean pour l'utiliser dans la JSP.
- L'utilisation d'un bean dans une JSP est très pratique car il peut encapsuler des traitements complexes et être réutilisable par d'autre JSP ou composants.
- Le bean peut par exemple assurer l'accès à une base de données.
- L'utilisation des beans permet de simplifier les traitements inclus dans la JSP.
- Lors de l'instanciation d'un bean, on précise la portée du bean.
- Si le bean demandé est déjà instancié pour la portée précisée alors il n'y pas de nouvelle instance du bean qui est créée mais sa référence est simplement renvoyée : le tag `<jsp:useBean>` n'instancie donc pas obligatoirement un objet.

Le tag `<jsp:useBean>` (2/6)

- Ce tag ne permet pas de traiter directement des EJB.
- La syntaxe est la suivante :

```
<jsp:useBean id="beanInstanceName"
             scope="page|request|session|application"
             { class="package.class" | type="package.class"
               | class="package.class" type="package.class"
               | beanName="{package.class | <%= expression %>}"
                 type="package.class"}
              { /> | > ...
</jsp:useBean>
```

- L'attribut id permet de donner un nom à la variable qui va contenir la référence sur le bean.
- L'attribut scope permet de définir la portée durant laquelle le bean est défini et utilisable.
- La valeur de cet attribut détermine la manière dont le tag localise ou instancie le bean.

Le tag `<jsp:useBean>` (3/6)

- Les valeurs possibles sont :
  - **page :** Le bean est utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus. C'est la valeur par défaut.
  - **request :** Le bean est accessible durant la durée de vie de la requête.
    La méthode `getAttribute()` de l'objet request permet d'obtenir une référence sur le bean.
  - **session :** Le bean est utilisable par toutes les JSP qui appartiennent à la même session que la JSP qui a instanciée le bean.
    Le bean est utilisable tout au long de la session par toutes les pages qui y participent.
    La JSP qui créé le bean doit avoir l'attribut session = "true" dans sa directive page.
  - **application :** Le bean est utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instanciée le bean.

Le tag <jsp:useBean> (4/6)

- L'attribut class permet d'indiquer la classe du bean.
- L'attribut type permet de préciser le type de la variable qui va contenir la référence du bean.
- La valeur indiquée doit obligatoirement être une super classe du bean ou une interface implémentée par le bean (directement ou par héritage)
- L'attribut beanName permet d'instancier le bean grâce à la méthode `instanciate()` de la classe Beans.

Le tag <jsp:useBean> (5/6)

- Un exemple complet : Le fichier `TestBean.jsp`.

```
<HTML>
  <HEAD>
    <TITLE>Essai d'instanciation d'un bean dans une JSP</TITLE>
  </HEAD>
  <BODY>
    <p>Test d'utilisation d'un Bean dans une JSP </p>
    <jsp:useBean id="personne" scope="request" class="Personne" />
    <p>nom initial = <%=personne.getNom() %></p>
    <%
      personne.setNom("mon nom");
    %>
    <p>nom mise à jour = <%= personne.getNom() %></p>
  </BODY>
</HTML>
```


Le tag <jsp:useBean> (6/6)

- Le fichier `Personne.java`.

```
public class Personne {
  private String nom;
  private String prenom;

  public Personne() {
    this.nom = "nom par defaut";
    this.prenom = "prenom par defaut";
  }

  public void setNom (String nom) {
    this.nom = nom;
  }

  public String getNom() {
    return (this.nom);
  }

  public void setPrenom (String prenom) {
    this.prenom = prenom;
  }

  public String getPrenom () {
    return (this.prenom);
  }
}
```


Le tag <jsp:setProperty> (1/3)

- Le tag `<jsp:setProperty>` permet de mettre à jour la valeur d'un ou plusieurs attributs d'un Bean.
- Le tag utilise le setter (méthode setXXX() ou XXX est le nom de la propriété avec la première lettre en majuscule) pour mettre à jour la valeur.
- Le bean doit exister grâce à un appel au tag `<jsp:useBean>`.
- Il existe trois façons de mettre à jour les propriétés soit à partir des paramètres de la requête soit avec une valeur :
  - alimenter automatiquement toutes les propriétés avec les paramètres correspondants de la requête
  - alimenter automatiquement une propriété avec le paramètre de la requête correspondant
  - alimenter une propriété avec la valeur précisée

Le tag <jsp:setProperty> (2/3)

- La syntaxe est la suivante :

```
<jsp:setProperty name="beanInstanceName"
{ property="*" |
property="propertyName" [ param=" parameterName" ] |
property="propertyName" value="{string | <%= expression%>}"
}
/>
```

- L'attribut name doit contenir le nom de la variable qui contient la référence du bean.
- Cette valeur doit être identique à celle de l'attribut id du tag <jsp:useBean> utilisé pour instancier le bean.
- L'attribut property= "*" permet d'alimenter automatiquement les propriétés du bean avec les paramètres correspondants contenus dans la requête.
- Le nom des propriétés et le nom des paramètres doivent être identiques.

Le tag <jsp:setProperty> (3/3)

- Comme les paramètres de la requête sont toujours fournis sous forme de String, une conversion est réalisée en utilisant la méthode valueOf() du wrapper du type de la propriété.
- Exemple complet :

```
<HTML>
  <HEAD>
  <TITLE>Essai d'instanciation d'un bean dans une JSP</TITLE>
  </HEAD>
  <BODY>
    <p>Test d'utilisation d'un Bean dans une JSP </p>
    <jsp:useBean id="personne" scope="request" class="test.Personne" />
    <p>nom initial = <%= personne.getNom() %></p>
    <jsp:setProperty name="personne" property="nom" value="mon nom" />
    <p>nom mis à jour = <%= personne.getNom() %></p>
  </BODY>
</HTML>
```


Le tag <jsp:getProperty> (1/2)

- Le tag <jsp:getProperty> permet d'obtenir la valeur d'un attribut d'un Bean.
- Le tag utilise le getter (méthode getXXX() ou XXX est le nom de la propriété avec la première lettre en majuscule) pour obtenir la valeur et l'insérer dans la page HTML généré.
- Le bean doit exister grâce à un appel au tag <jsp:useBean>.
- La syntaxe est la suivante :

```
<jsp:getProperty name="beanInstanceName" property=" propertyName" />
```


Le tag <jsp:getProperty> (2/2)

- Exemple complet :

```
<HTML>
<HEAD>
  <TITLE>Essai d'instanciation d'un bean dans une JSP</TITLE>
</HEAD>
<BODY>
  <p>Test d'utilisation d'un Bean dans une JSP </p>
  <jsp:useBean id="personne" scope="request" class="test.Personne" />
  <p>nom initial = <jsp:getProperty name="personne" property="nom" /></p>
  <jsp:setProperty name="personne" property="nom" value="mon nom" />
  <p>nom mise à jour = <jsp:getProperty name="personne" property="nom" /></p>
</BODY>
</HTML>
```

Le tag de redirection <jsp:forward> (1/2)

- Le tag <jsp:forward> permet de rediriger la requête vers une autre URL pointant vers un fichier HTML, JSP ou un servlet.
- Dès que le moteur de JSP rencontre ce tag, il redirige le requête vers l'URL précisée et ignore le reste de la JSP courante.
- Tout ce qui a été généré par la JSP est perdu.
- La syntaxe est la suivante :

```
<jsp:forward page="{relativeURL | <%= expression %>}" />
```

ou

```
<jsp:forward page="{relativeURL | <%= expression %>}" >
<jsp:param name="parameterName"
           value="{ parameterValue | <%= expression %>}" /> +
</jsp:forward>
```

- L'option page doit contenir la valeur de l'URL de la ressource vers laquelle la requête va être redirigée.

Le tag de redirection <jsp:forward> (2/2)

- Cette URL est absolue si elle commence par un '/' sinon elle est relative à la JSP . Dans le cas d'une URL absolue, c'est le serveur web qui détermine la localisation de la ressource.
- Il est possible de passer un ou plusieurs paramètres vers la ressource appelée grâce au tag <jsp :param>.
- Exemple complet :

```
<html>
  <body>
    <p>Page initiale appelée</p>
    <jsp:forward page="forward.html"/>
  </body>
</html>
```

- Dans l'exemple, le fichier forward.html doit être dans le même répertoire que la JSP.
- Lors de l'appel à la JSP, c'est la page HTML qui est affichée.
- Le contenu généré par la page JSP n'est pas affiché.

Le tag <jsp:include> (1/2)

- Ce tag permet d'inclure le contenu généré par une JSP ou une servlet dynamiquement au moment où la JSP est exécutée.
- C'est la différence avec la directive include avec laquelle le fichier est inséré dans la JSP avant la génération de la servlet.
- La syntaxe est la suivante :

```
<jsp:include page="relativeURL" flush="true" />
```

- L'attribut page permet de préciser l'URL relative de l'élément à insérer.
- L'attribut flush permet d'indiquer si le tampon doit être envoyé au client et vidé.
- Si la valeur de ce paramètre est true :
  - il n'est pas possible d'utiliser certaines fonctionnalités dans la servlet ou la JSP appelée.
  - il n'est pas possible de modifier l'entête de la réponse (header, cookies) ou renvoyer ou faire suivre vers une autre page.

Le tag <jsp:include> (2/2)

- Exemple complet :

```
<html>
  <body>
    <jsp:include page="bandeau.jsp"/>
    <H1>Bonjour</H1>
    <jsp:include page="pied.jsp"/>
  </body>
</html>
```

- Il est possible de fournir des paramètres à la servlet ou à la JSP appelée en utilisant le tag <jsp:param>.

Le tag <jsp:plugin> (1/3)

- Ce tag permet la génération du code HTML nécessaire à l'exécution d'une applet en fonction du navigateur : un tag HTML <Object> ou <Embed> est généré en fonction de l'attribut User-Agent de la requête.
- Le tag <jsp:plugin> possède trois attributs obligatoires :
  - **code :** permet de préciser le nom de classe
  - **codebase :** contient une URL précisant le chemin absolu ou relatif du répertoire contenant la classe ou l'archive
  - **type :** les valeurs possibles sont applet ou bean

Le tag <jsp:plugin> (2/3)

- Il possède aussi plusieurs autres attributs optionnels dont les plus utilisés sont :
  - **align :** permet de préciser l'alignement de l'applet : les valeurs possibles sont bottom, middle ou top.
  - **archive :** permet de préciser un ensemble de ressources (bibliothèques jar, classes, ...) qui seront automatiquement chargées. Le chemin de ces ressources tient compte de l'attribut codebase.
  - **height :** précise la hauteur de l'applet en pixel ou en pourcentage.
  - **hspace :** précise le nombre de pixels insérés à gauche et à droite de l'applet.
  - **jrversion :** précise la version minimale du jre à utiliser pour faire fonctionner l'applet.
  - **name :** précise le nom de l'applet.
  - **vspace :** précise le nombre de pixels insérés en haut et en bas de l'applet.
  - **width :** précise la longueur de l'applet en pixel ou en pourcentage.

Le tag <jsp:plugin> (3/3)

- Pour fournir un ou plusieurs paramètres, il faut utiliser dans le corps du tag <jsp:plugin> le tag <jsp:params>.
- Chaque paramètre sera alors défini dans un tag <jsp:param> :
- Exemple complet :

```
<jsp:plugin type="applet" code="MonApplet.class" codebase="applets"
     jreversion="1.1" width="200" height="200" >
  <jsp:params>
     <jsp:param name="couleur" value="eeeeee" />
  </jsp:params>
</jsp:plugin>
```

- Le tag <jsp:fallback> dans le corps du tag <jsp:plugin> permet de préciser un message qui sera affiché dans les navigateurs ne supportant pas le tag HTML <Object> ou <Embed>.

Présentation de la librairie JSTL (1/3)

- JSTL est l'acronyme de Java server page Standard Tag Library.
- C'est un ensemble de tags personnalisés développé sous la JSR 052 qui propose des fonctionnalités souvent rencontrées dans les JSP :
  - Tag de structure (itération, conditionnement, ...).
  - Internationalisation
  - Exécution de requête SQL
  - Utilisation de document XML
- JSTL nécessite un conteneur d'application web qui implémente l'API servlet 2.3 et l'API JSP 1.2.
- L'implémentation de référence (JSTL-RI) de cette spécification est développée par le projet Taglibs du groupe Apache sous le nom "Standard".
- Il est possible de télécharger cette implémentation de référence à l'URL : http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html

Présentation de la librairie JSTL (2/3)

- JSTL est aussi inclus dans le JWSDP (Java Web Services Developer Pack), ce qui facilite son installation et son utilisation.
- JSTL possède quatre bibliothèques de tag :
  - Les **Fonctions de base** se trouvent dans le fichier `c.tld` à l'adresse `http://java.sun.com/jstl/core`
  - Les **Traitements XML** se trouvent dans le fichier `x.tld` à l'adresse `http://java.sun.com/jstl/xml`
  - L'**Internationalisation** se trouvent dans le fichier `fmt.tld` à l'adresse `http://java.sun.com/jstl/fmt`
  - Les **Traitements SQL** se trouvent dans le fichier `sql.tld` à l'adresse `http://java.sun.com/jstl/sql`

Présentation de la librairie JSTL (3/3)

- JSTL propose un langage nommé EL (expression langage) qui permet de faire facilement référence à des objets java accessibles dans les différents contextes de la JSP.
- La bibliothèque de tag JSTL est livrée en deux versions :
  - JSTL-RT : les expressions pour désigner des variables utilisant la syntaxe JSP classique
  - JSTL-EL : les expressions pour désigner des variables utilisant le langage EL
- Pour plus informations, il est possible de consulter les spécifications à l'url suivante : `http://jcp.org/aboutJava/communityprocess/final/jsr052/`.

JSTL : un exemple simple (1/4)

- L'application web d'exemple se nomme test.
- Il faut créer un répertoire pour l'application dans le répertoire webapps de tomcat.
- Pour utiliser JSTL, il faut copier les fichiers jstl.jar et standard.jar dans le répertoire WEB-INF/lib de l'application web.
- Il faut copier les fichiers .tld dans le répertoire WEB-INF ou un de ses sous répertoires.
- Dans la suite de l'exemple, ces fichiers ont été placés dans le répertoire /WEB-INF/tld.
- Il faut ensuite déclarer les bibliothèques à utiliser dans le fichier `web.xml` du répertoire WEB-INF comme pour toute bibliothèque de tags personnalisés.

JSTL : un exemple simple (2/4)

- Exemple : pour la bibliothèque Core

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
  <taglib-location>/WEB-INF/tld/c.tld</taglib-location>
</taglib>
```

- L'arborescence des fichiers est la suivante :

```
webapps
      MonApplication
            WEB-INF
                  lib
                        jstl.jar
                        standard.jar
                  tld
                        c.tld
                  web.xml
            test.jsp
```

JSTL : un exemple simple (3/4)

- Pour pouvoir utiliser une bibliothèque personnalisée, il faut utiliser la directive taglib :

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
```

- Voici les codes sources des différents fichiers de l'application web :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Exemple</title>
  </head>

  <body>
      <c:out value="Bonjour" /><br/>
  </body>
</html>
```


JSTL : un exemple simple (4/4)

- Exemple : le fichier WEB-INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app23.dtd">

<web-app>
  <taglib>
    <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
    <taglib-location>/WEB-INF/tld/c.tld</taglib-location>
  </taglib>
</web-app>
```

- Pour tester l'application, il suffit de lancer Tomcat et de saisir l'url `http://localhost:8080/MonApplication/test.jsp` dans un browser.

Le langage EL (Expression Language)

- JSTL propose un langage particulier constitué d'expressions qui permet d'utiliser et de faire référence à des objets java accessible dans les différents contextes de la page JSP.
- Le but est de fournir un moyen simple d'accéder aux données nécessaires à une JSP.
- La syntaxe de base est `${xxx}` où `xxx` est le nom d'une variable d'un objet java défini dans un contexte particulier.
- La définition dans un contexte permet de définir la portée de la variable (page, requête, session ou application).
- EL permet facilement de s'affranchir de la syntaxe de java pour obtenir une variable.
- Exemple : accéder à l'attribut nom d'un objet personne situé dans la session avec Java

```
<%= session.getAttribute("personne").getNom() %>
```


Le langage EL (Expression Language)

- Exemple : accéder à l'attribut nom d'un objet personne situé dans la session avec EL.

```
${sessionScope.personne.nom}
```

- EL possède par défaut les variables suivantes :
  - **PageScope :** variable contenue dans la portée de la page (PageContext).
  - **RequestScope :** variable contenue dans la portée de la requête (HttpServletRequest).
  - **SessionScope :** variable contenue dans la portée de la session (HttpSession).
  - **ApplicationScope :** variable contenue dans la portée de l'application (ServletContext).
  - **Param :** paramètre de la requête http.
  - **ParamValues :** paramètres de la requête sous la forme d'une collection.

Le langage EL (Expression Language)

- Suite variables :
  - **Header :** en tête de la requête.
  - **HeaderValues :** en têtes de la requête sous la forme d'une collection.
  - **InitParam :** paramètre d'initialisation.
  - **Cookie :** cookie.
  - **PageContext :** objet PageContext de la page.

Le langage EL (Expression Language)

- EL propose aussi différents opérateurs :
  - **.** Obtenir une propriété d'un objet.
  - `[]` Obtenir une propriété par son nom ou son indice.
  - **Empty** Teste si un objet est null ou vide si c'est une chaîne de caractère. Renvoie un booléen.
  - **== ou eq** test l'égalité de deux objets.
  - **!= ou ne** test l'inégalité de deux objets.
  - **< ou lt** test strictement inférieur.
  - **> ou gt** test strictement supérieur.
  - **<= ou le** test inférieur ou égal.
  - **>= ou ge** test supérieur ou égal.
  - **+** Addition ; **-** Soustraction.
  - ***** Multiplication ; / **ou div** Division.
  - **% ou mod** Modulo.
  - **&& ou and** Et logique.
  - `||` **ou or** Ou logique.
  - **! ou not** Négation d'une valeur.

Le langage EL (Expression Language)

- EL ne permet pas l'accès aux variables locales.
- Pour pouvoir accéder à de telles variables, il faut obligatoirement en créer une copie dans une des portées particulières : page, request, session ou application
- Exemple :

```
<%
  int valeur = 101;
%>
  valeur = <c:out value="${valeur}" /><BR/>
```

- Résultat :

```
valeur =
```

- Exemple : avec la variable copiée dans le contexte de la page

```
<%
  int valeur = 101;
  pageContext.setAttribute("valeur", new Integer(valeur));
%>
  valeur = <c:out value="${valeur}" /><BR/>
```

- Résultat :

```
1.valeur = 101
```