Introduction à la plate-forme Java Enterprise Edition

année 2012-2013

Agenda

Qu’est-ce la plateforme J2EE

Les Servlets et applications web

Les JSP

Le Modèle MVC (Architecture en couches)

L’Accès aux données via l’interface universelle « JDBC »

Que sont les JSP (Java Server Pages) ?

☛ Une page HTML dans laquelle sont introduits des morceaux de code Java nommés « éléments de script »

```
<html>
    <head>
        <title> JSP Bonjour </title>
    </head>
    <body>
        <% out.println ("BONJOUR");%>
    </body>
</html>
```

Les JSP

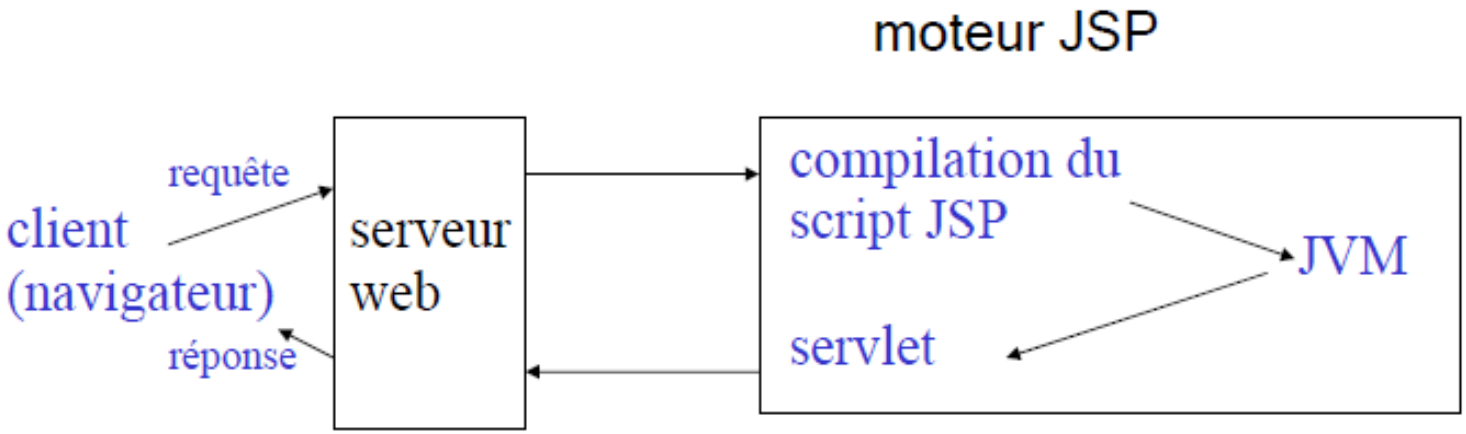
Chaque script est interprété par un serveur JSP qui différencie le code JSP du code HTML grâce à des balises spécifiques

Le serveur JSP créé, compile, exécute une servlet. C'est la servlet générée qui produit le code HTML,... de la page envoyée au client

L'utilisation des JSP n'est pas limitée à la génération de code HTML. Le contenu généré peut être du code XML ou XHTML

Les JSP

- le script JSP est compilé en une servlet
- la servlet est compilée puis exécutée



JSP vs. Servlets

- Servlet = du code Java contenant de l'HTML
- JSP = une page HTML contenant du code Java
- Concrètement avec les JSP :
  - les parties statiques de la page HTML sont écrites en HTML
  - les parties dynamiques de la page HTML sont écrites en Java

Exemple de JSP

- fichier `date.jsp`

```
<html><head><title>Obtenu par une JSP</title></head>
<body>

<h3>Bonjour de ma part </h3> <hr>
La date courante est :  <%= new java.util.Date() %>
</body>
</html>
```

- Traité quand le client demande l'URL de la JSP :
  `http://serveurWeb:<port>/.../date.jsp`

Mécanismes mis en oeuvre

Plusieurs zones <% ... %> peuvent cohabiter dans une même JSP

- Au premier chargement d'une JSP, le **moteur JSP**

  - rassemble **tous** les fragments <% ... %> de la JSP dans une classe
  - la **compile en une servlet**
  - la servlet est compilée et **instanciée**

JSP = objet Java présent dans le moteur

- Puis, ou lors des chargements suivants, le **moteur JSP**

  - exécute le code de la servlet dans un ***thread***

=> délai d'attente lors de la 1ère invocation dû à la compilation

Moteurs de JSP

- Pour exécuter des JSP , il faut un moteur de JSP dans le serveur Web.

- Exemple des serveurs Web qui traitent les servlets et JSP :
  - Tomcat
  - IBM WebSphere

Tomcat

- Développé par la communauté qui implémente les spécifs servlets et JSP.
- Téléchargeable (en version d'utilisation élémentaire) gratuitement à
  http://tomcat.apache.org/download-60.cgi
- Existe pour plusieurs Unix et Win

Exécution de JSP

- Il faut mettre les pages JSP dans un endroit particulier du serveur Web
- Cet endroit dépend du serveur Web et de sa configuration
- Pour tomcat en configuration standard,
  `http://serveurWeb/examples/jsp/date.jsp`
  ~
  `REP_INSTAL_TOMCAT\webapps\examples\jsp\date.jsp`

Exécution de JSP

- Le résultat de `date.jsp` est :

Bonjour de ma part

La date courante est : Sun Dec 23 18:04:36 CET 2001

- Une autre exécution donne une autre date => dynamicité

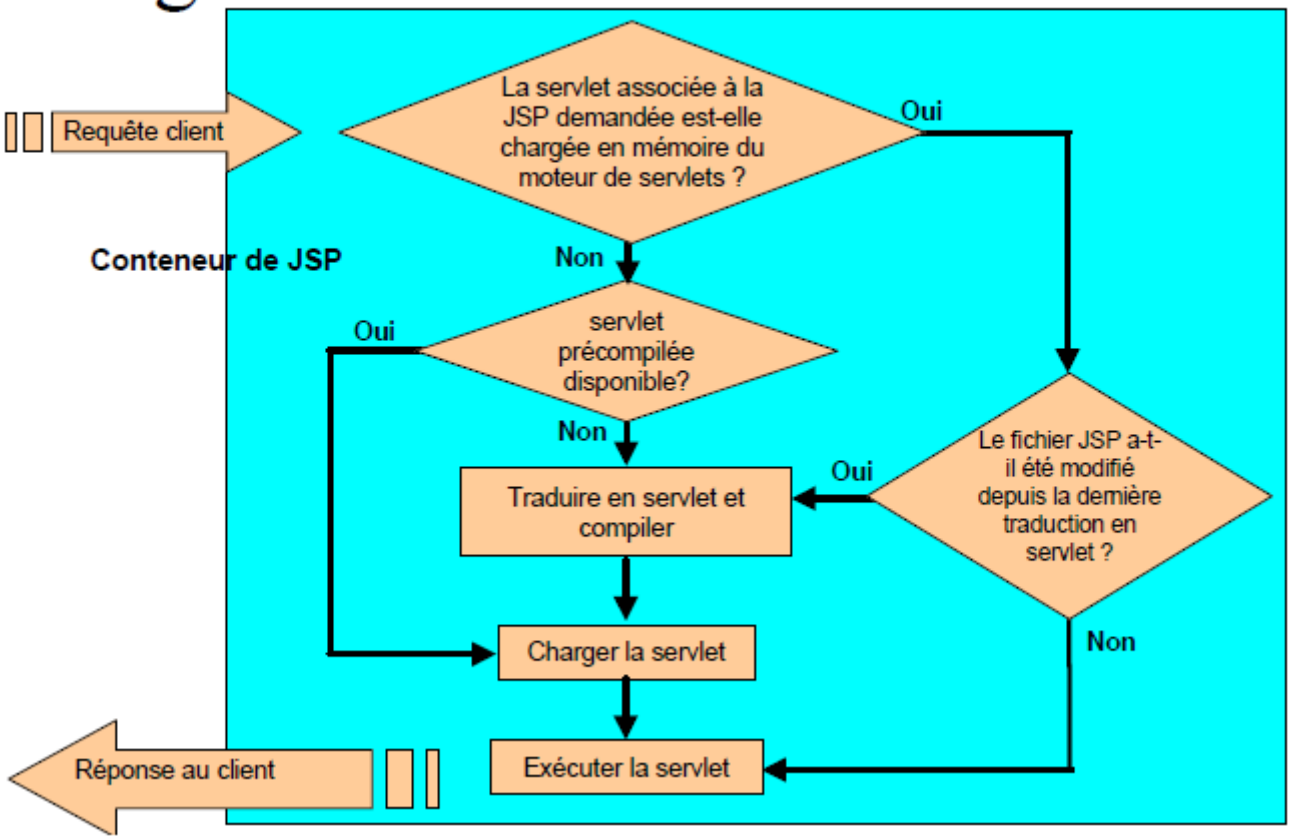
Que s'est il passé ?

- Le moteur de JSP a construit une servlet (`_0002fjsp_0002fdate_0002ejspdate_jsp_0.java` dans `REP_INSTALL_TOMCAT\work\localhost_8080%2Fexamples`)
- Cette phase est parfois appelée la traduction de la JSP
- Puis a compilé et exécuté la servlet

La servlet construite

```
package jsp;
...
public class _0002fjsp_0002fjsp_0002fdate_0002ejspdate_jsp_1 extends
HttpJspBase {
            ...
    public void _jspService(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {
              ...
              _jspx_init();
              ...
              pageContext = _jspxFactory.getPageContext(...);
              session = pageContext.getSession();
              out = pageContext.getOut();
              // HTML
     // begin [file="C:\\...\\examples\\jsp\\date.jsp";from=(0,0);to=(4,24)]
           out.write("<html><head><title>Obtenu par une JSP</title></head>\r\n
<body>\r\n\r\n<h3>Bonjour de ma part</h3> <hr>\r\n
            La date courante est :  ");
     // end
//begin [file="C:\\...\\examples\\jsp\\date.jsp";from=(4,27)to=(4,49)]
                  out.print( new java.util.Date() );
              // end
              // HTML
// begin [file="C:\\...\\examples\\jsp\\date.jsp";from=(4,51);to=(6,7)]
                  out.write("\r\n</body>\r\n</html>"); // end
              ...
    }
}
```

Algorithme d'exécution de la JSP



Syntaxes des scripts : 5 types de balises

Les balises permettant de définir des variables globales à la page
balises déclarations <%! . . . %>

Les balises permettant d'évaluer une expression et l'affichage de sa valeur dans la page

<%= . . . %>

Les balises de scriptlets permettant d'inclure du code java

<% . . . %>

Les balises de commentaires

<%-- . . .--%>

Les balises de directives spécifient les paramètres applicables à la page

<%@ . . . %>

Déclarations <% ! %>

- Sont des déclarations Java.
- Seront insérées comme des membres de la servlet
- Permet de définir des méthodes ou des données membres
- Exemples :

```
<%!
      int random4() {
            return (int)(Math.random() * 4);
      }
%>
```

```
<%!
      int nombreFetiche = 2;
%>
```

Expressions <%= %>

- Expression Java qui renvoie un objet `String` ou un type primitif.
- Un raccourci pour `<% out.println(…); %>`
- `<%= XXX %>` ~ `<% out.println(XXX); %>`
- attention au `;`
- est donc converti en `out.println(…)` dans la méthode `_jspService(…)` de la servlet.

```
La somme est:  <%= (195 + 9 + 273) %>

Je vous réponds à l'adresse :  <%= request.getParameter("email_address") %>
```

Balises de scriptlets

Une scriptlet une suite d'instructions Java contenues dans la balise

<% %>

et exécutées dès qu'un client invoque la page

Les variables déclarées dans une scriptlet sont **locales** à une requête.

Contrairement aux variables déclarées dans la balise <%! %> qui persistent d'une requête à l'autre

Exemple :

```
<% float[] pttc = {23.4,78.9,32.0};
   for (int i=0;i<pttc.length;i++){
       out.println("<h2>Prix TTC</h2>");
       out.println(pttc[i]);
   }
%>
```

Balises de commentaires

- 2 manières de placer des commentaires dans une page JSP

```
<!--  mon commentaire -->
```

dans ce cas les commentaires sont transmis à la réponse. Ils sont donc visibles lorsqu'on visualise le code source de la page HTML.

ou

```
<%-- mon commentaire --%>
```

Le serveur JSP ne les prend pas en compte. Leur intérêt est uniquement de commenter le code JSP

Balises de directives

balises qui spécifient des directives de génération de page au compilateur :

- **`page`** permet de définir un certain nombre d'attributs qui s'appliqueront à la page
- **`include`** permet d'inclure un fichier texte ou autre ressource à la création de la servlet

La directive page

Attributs qui s'appliqueront à la page

- **import** définit les paquetages à importer
  l'attribut **extends** définit la classe parente de la servlet

```
<%@page import="java.util.Date"%>
   date du jour : <%= new Date()%>
```

- **langage** définit le langage de script utilisé dans la page

```
<%@page langage="java"%>
```

- **contentType** définit le type de contenu de la page générée

```
<%@page contentType="text/plain" %>
<%@page contentType="text/html" %>
```

- **errorPage** spécifie la page à afficher en cas d'erreur

```
 <%@page errorpage ="500.jsp" %>
```

- **isErrorPage** vaut `true` si la page est une page d'erreur, `false` sinon

```
<%@page isContentType ="false" %>
```

Directive `page`, attribut **`errorPage`**

Dans la JSP provoquant l'erreur :

- on peut récupérer une exception déclenchée dans une scriptlet JSP.
- son traitement pourra être assurée par une page d'erreur
- la page d'erreur est spécifiée par la balise `page` et son attribut `errorPage`

```
<%@ page errorPage="erreur.jsp" %>
```

ou

```
<%@ page errorPage="erreur.jsp?code=xxx" %>
```

NOTE :

Une seule page d'erreur par JSP

transmission d'information à la page d'erreur

Directive `page`, attribut **`isErrorPage`**

`false` indique une page normale

Dans la page de traitement de l'erreur :

- on définit la page JSP comme une page d'erreur par

```
<%@ page isErrorPage=true %>
```

- l'objet accompagnant l'exception est référencé par la variable implicite `exception`
  On peut donc recueillir des informations sur l'erreur par les méthodes classiques de la classe `Exception`:

```
exception.getMessage()
exception.printStackTrace()
```

Directive page, attribut errorPage: exemple

```
<%@ page language="java" contentType="text/html" %>
<%@ page errorPage="/erreur.jsp" %>
<html><head><title>Page avec une erreur</title></head>
<body>
<% int var=90; var = var/0; %>
Division par 0 = <%= var %>
</body></html>
```

division.jsp

```
<%@ page language="java" contentType="text/html" %>
<%@ page isErrorPage="true" %>
<html><head><title>Page de gestion de l'erreur</title></head>
<body><h2><%=exception.getClass().getName()%><br>
l'exception déclenchée est : <%= exception.getMessage()%>
</body></html>
```

objet implicite

erreur.jsp

Directive `page`, attribut `errorPage`: exemple

La directive `include`

- **`include`** permet d'inclure dans la page un fichier texte ou autre ressource à la création de la servlet

```
<%@include
        file="/chemin/relatif/auContexte"%>
<%@include
        file="dansLe/repertoire/deLaPage"%>
```

recherche relativement à `ServletContext`

recherche dans le même répertoire que la JSP courante

- En général, le contenu du fichier est un fragment de page. Les variables créées par le fichier inclus sont dans la portée de la page

La directive `include`

- Cette inclusion se fait au moment de la conversion, pendant la création de la servlet correspondante.
- Le contenu du fichier externe est inclus comme s'il était saisi directement dans la page JSP
- Les ressources à inclure doivent être contenues dans le contexte de l'application web

La directive include

```
<HTML><HEAD>
<TITLE>Page de titre</TITLE>
</HEAD><BODY>
```

entete.html

corps.jsp

```
<%! String nom; %>
<% nom=" test "; %>
```

```
<%@ include file="/entete.html" %>
<%@ include file="/corps.jsp" %>
Bonjour <%=nom %>
<%@ include file="/piedpage.html" %>
```

nom : variable déclarée dans le fichier inclus `corps.jsp`

```
Ceci est le pied de page.
</BODY>
</HTML>
```

piedpage.html

La directive `taglib`

l'API 1.1 de Java Server Pages permet aux développeurs de créer une bibliothèque de balises qui peuvent être mélangées aux balises jsp standards ainsi qu'aux balises html.

Le moteur jsp traite ces balises comme des balises standards pour créer du code java compilé ensuite en servlets.

```
<%@taglib uri="taglib/maBalise.tld"
          prefix="unPrefixe"%>
```

fichier xml spécifiant l'ensemble des attributs de chaque balise

Objets prédéfinis dans une JSP

- 3 objets peuvent être immédiatement utilisés dans une expression ou une scriptlet d'une JSP :
  - `out` : le canal de sortie
  - `request` (`HttpServletRequest`) : l'objet requête
  - `response` (`HttpServletResponse`) : l'objet réponse
- Il y en a d'autres
- Cf. ces mêmes objets dans une servlet

Un exemple

déclaration, expression, scriplets, objet out

```
<html><head><title>JSP complet</title></head>
<body>
<%! String[] langages = {"Java", "C++", "Smalltalk", "Simula 67"};
    int random4() {
            return (int) (Math.random() * 4);
    }
%>
<p>Parmi tous les langages orientés objets :</p>
<ol>
<%
    for (int i=0; i < langages.length; i++) {
            out.println("<li>" + langages[i] + "</li>");
    }
%>
</ol>
<p>Le prochain que j'apprends est <b><%= langages[random4()] %> </b></p>
</body>
</html>
```

Les JSP


JSP complet - Netscape
Fichier Edition Affjcher Aller Communicator Aide
Précédent Suivant Recharger Accueil Rechercher
Signets Adresse : p/complet.jsp Infos connexes
Parmi tous les langages orientés objets :
1. Java
2. C++
3. Smalltalk
4. Simula 67
Le prochain que j'apprends est C++

Enchaîner les pages

- Un page JSP peut en appeler une autre par la directive : `<jsp:forward>`
- Syntaxe :

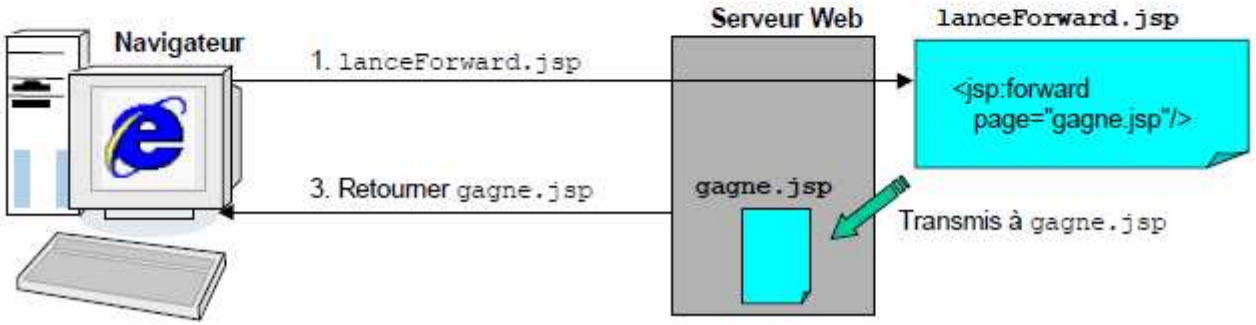
```
<jsp:forward page="pageDeRedirection" />
```

lanceForward.jsp

```
<% String repUtilisateur = request.getParameter("repTextField");
   int rep = Integer.parseInt(repUtilisateur);
        if ((rep % 2) == 0) {
%>
        <jsp:forward page="gagne.jsp"/>
<% } else { %>
        <jsp:forward page="perdu.jsp"/>
<% } %>
On n'affiche jamais cela
```

Enchaîner les pages (suite)

- Après un `<jsp:forward>`, le traitement est entièrement pris en charge par nouvelle page



JSP et Java beans

- But : avoir le moins de code Java possible dans une page JSP (HTML)
- Sous-traiter le code à un Java bean
- balise XML : `<jsp:useBean>`

Définition

- Un bean est une classe Java à laquelle on associe des propriétés
- Chaque propriété a une ou plusieurs méthodes d'accès
- Le codage des méthodes suit certaines conventions de nommage

Les JSP et Java beans

Les Java Beans sont des classes Java qui respectent les directives suivantes :

- un constructeur public sans argument
- les propriétés d'un Bean sont accessibles au travers de méthodes `getXXX` (lecture) et `setXXX` (écriture) portant le nom de la propriété

Méthodes de lecture des propriétés :

```
type getNomDeLaPropriété()
```

pas de paramètre et son type est celui de la propriété

```
boolean isNomPropriété()
```

Méthodes d'écriture des propriétés :

`void setNomDeLaPropriété(type)` : un seul argument du type de la propriété et son type de retour est `void`

Un Java Beans implémente l'interface `java.io.Serializable` permettant la sauvegarde de l'état d'1 bean

Java bean : exemple

```
public class Personne implements Serializable{
  private String nom;
  private String prenom;
  private int age;
  public Personne(){}
  public Personne(String nom, String prenom, int age) {
    this.nom = nom;this.prenom = prenom;this.age = age;}
  public int getAge() {return age;}
  public String getNom() {return nom;}
  public void setAge(int age) {this.age = age;}
  public void setNom(String nom) { this.nom = nom;}
  public void setPrenom(String prenom){this.prenom = prenom;}
  public String getPrenom() {return prenom;}
}
```

variables d'instance privées

Beans et JSP

- Un bean est inclus dans une page JSP par la balise :

  `<jsp:useBean>`

- Après exécution, un bean rend son résultat à la page JSP ou à la servlet
- L'utilisation de beans évite l'emploi de la méthode `getRequestParameter(...)`
- La valeur du paramètre d'une requête peut être utilisée directement pour renseigner la propriété d'un bean
- Un serveur peut gérer la portée d'un bean en le liant soit à une page, une requête, une session ou une application

Portée d'un bean

```
<jsp:useBean id = "nomDuBean"
        scope = "page|request|session|application"
        class = "nomPackage.nomClasse"
        type = "typeDuBean">
</jsp:useBean>
```

- `class = "nomClasse"` on suppose ici que la classe `nomClasse` se trouve dans le package `nomPackage` de l'application
- `class = "fr.eclipse.nomPackage.nomClasse"` On indique ici le nom absolu de la classe
- `type` est utilisé pour le transtypage. Il indique soit une classe mère soit une interface de la classe

l'attribut `scope`

- Il indique la portée du bean.

valeur	Description
`request`	Le bean est valide pour cette requête. Il est utilisable dans les pages de redirection de la requête (`<jsp:forward>`). Il est détruit à la fin de la requête.
`page`	Similaire à `request`, mais le bean n'est pas transmis aux pages de redirection `<jsp:forward>`. C'est la portée par défaut
`session`	Le bean est valide pour la session courante. S'il n'existe pas encore dans la session courante, il est créé et placé dans la session du client. Il est réutilisé jusqu'à ce que la session soit invalidée
`application`	Le bean est valide pour l'application courante. Il est créé une fois et partagé par tous les clients des JSP.

Propriétés d'un bean

- Les valeurs de tous les paramètres de la requête dont les noms correspondent aux propriétés du bean leur sont affectés

```
<jsp:setproperty  name="nomDuBean" property="*" />
```

- Seul le paramètre `unePropriété` est utilisé pour renseigner la propriété `unePropriete`

```
<jsp:setproperty name="nomDuBean"
                 property="unePropriete"/>
```

- On utilise ici un paramètre de requête dont le nom est différent de celui de la propriété du bean

```
<jsp:setproperty name="nomDuBean"
  property="unePropriete" param=nomduParametre/>
```

- La propriété prend directement la valeur `constante`

```
<jsp:setproperty name="nomDuBean"
  property="unePropriete"  value="constante" />
```

Exemple

- La page JSP qui suit affiche le texte : `Bonjour` suivi de la valeur du paramètre `nom` de la requête.
- La classe `HelloBean` est stockée dans le répertoire `entites`.
- La propriété `nom` du bean `HelloBean` est alimentée par le paramètre de la requête passé dans son en-tête si celui-ci est renseigné
- Sinon c'est sa valeur par défaut qui est utilisée
- L'exécution de la page JSP provoque l'affichage du texte `bonjour` suivi du contenu de la propriété nom du bean `HelloBean`
- Intérêt :
  - pas de code Java dans la page JSP
  - modularité dans la construction de l'application

Exemple (suite)

```
package entites;
import java.io.Serializable;
public class HelloBean implements Serializable{
   private String nom = " à tous";
   private String prenom = "  ";
   public void setNom( String nom ){
        this.nom=nom;
   }
   public String getNom(){
        return nom;
   }
   public void setPrenom( String prenom ){
        this.prenom=prenom;
   }
   public String getPrenom(){
        return prenom;
   }
}
```

Exemple (suite)

```
<%-- hello.jsp --%>

<jsp:useBean id="hello" class="entites.HelloBean">
<jsp:setProperty name="hello" property="prenom"
                              value=" test  " />
</jsp:useBean>

<HTML>
<HEAD><TITLE>Bonjour</TITLE></HEAD>
<BODY>
<H2>
Bonjour <jsp:getProperty name="hello" property="nom" />
   <br>
        <jsp:getProperty name="hello" property="prenom" />
</H2>
</BODY>
</HTML>
```

valeur à donner à la propriété

référence du bean

nom de la propriété

Les JSP et Java beans

Autre Exemple

un prénom est saisi
à travers un formulaire

page affichée
en réponse

Le bean HelloBean

```
package entites;
import java.io.Serializable;
public class HelloBean implements Serializable{
	private String nom;
	private String prenom;
	public void setNom( String nom ){
		this.nom = nom;
	}
	public String getNom(){ return nom; }
	public void setPrenom( String prenom ){
		this.prenom = prenom;
	}
	public String getPrenom(){ return prenom; }
}
```

hello.jsp

initialisation

```
<jsp:useBean id="hello" class="entites.HelloBean">
<jsp:setProperty name="hello" property="prenom" value="  "/>
<jsp:setProperty name="hello" property="nom" value=" à tous!"/>
</jsp:useBean>
<jsp:setProperty name="hello" property="prenom" />
<HTML>
 <HEAD><TITLE>Bonjour</TITLE></HEAD>
 <BODY>
  <h2>Tapez votre prénom</h2>
  <form action="hello.jsp" method="post"><hr>
   <table><tr>
    <td>Prénom</td>
    <td><input name="prenom" value="" type="text" size="15">
    <td><input type="submit" value="Envoyer"></td>
   </tr></table></form><hr>
<b>Bonjour</b>
  <jsp:getProperty name="hello" property="nom" /><br>
  <jsp:getProperty name="hello" property="prenom" />
</BODY></HTML>
```

Fonctionnement

Initialisation

après le premier appel à `hello.jsp`, `nom` et `prenom` sont initialisés

L'objet `hello` est créé et initialisé avec les valeurs par défaut comprises entre les balises

```
<jsp:useBean id="hello" ..... </jsp:useBean>
```

Fonctionnement

Après validation du formulaire, la nouvelle valeur de la propriété prenom est transmise à l'objet `hello` par la balise jsp :

```
<jsp:setProperty name="hello" property="prenom" />
```

la page `hello.jsp` est de nouveau invoquée et la valeur des propriétés `nom` et `prenom` est affichée


```
<b>Bonjour</b>
  <jsp:getProperty name="hello" property="nom" /><br>
  <jsp:getProperty name="hello" property="prenom" />
```