

Technologie JSP



Les bases du langage de la technologie JSP

1. Les balises
2. Les directives d'une page
 1. Directives d'importation, d'inclusion et de définition
3. La portée ou visibilité des objets dans une page JSP
4. Les actions dites standard de la technologie JSP
5. Les expressions EL

Technologie JSP



Les Balises de bases

Balises de scriptlet : <% %>

Venant des mots "script" et "servlet", à l'intérieur d'une scriptlet se cache simplement du code Java.

Cette balise que nous avons déjà utilisée sert en effet à inclure du code Java au sein de vos page JSP.

Boucle java pour remplir la liste de choix des jours

Jours :	Lundi	<input type="button" value="Valider"/>
	<input checked="" type="checkbox"/> Lundi <input type="checkbox"/> Mardi <input type="checkbox"/> Mercredi <input type="checkbox"/> Jeudi <input type="checkbox"/> Vendredi <input type="checkbox"/> Samedi <input type="checkbox"/> Dimanche	

Exemple

```

<%@ page pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head> <title>Test</title> </head>
  <body>

    <form action="#" method="post">
      <label> Jours : </label>
      <select>

        > <%
          String[] Jours = {"Lundi", "Mardi", "Mercredi",
                            "Jeudi", "Vendredi", "Samedi",
                            "Dimanche"};
          for(int i = 0; i< Jours.length; i++)
            out.println("<option>" + Jours[i] + "</option>");
        <%>
        </select>
      </form>
      <input type="submit" value="Valider"/>
    </body>
  </html>

```

Technologie JSP



Les Balises de bases

Balises de déclaration : <%! %>

Scriptlet permettant de déclarer une variable ou une méthode.

Il est possible d'effectuer plusieurs déclarations au sein d'un même bloc

Déclaration d'une chaîne et d'une méthode

Test d'appel à la méthode Inverser

" L'inverse de EMSI est ISM E "

JSP

Exemple

```
<%! String Message = "E M S I";  
public String Inverser(String S) {  
    String resultat = "";  
    for(int i = (S.length()-1) ; i >= 0 ; i--)  
        resultat += S.charAt(i);  
    return resultat;  
}  
  
<span><% out.print(" L'inverse de " + Message +  
" est " + Inverser(Message)); %>  
</span>
```

Technologie JSP



Les Balises de bases

Balises d'expression : <%= %>

C'est un raccourci de la scriptlet suivante :

Elle retourne le contenu d'une chaîne :

```
<% out.println( "Bonjour ^_~" ); %>  
  
<%= "Bonjour ^_~" %>
```

Balises de commentaire : <%-- --%>

```
<%-- Ceci est un commentaire JSP, non visible dans la page HTML finale. --%>  
<%= "Bonjour ^_~" %>
```

Technologie JSP



Les directives JSP :

Les directives contrôlent comment le conteneur de servlets va gérer votre JSP.

Il en existe trois : **taglib**, **page** et **include**.

À travers les quelles on va pouvoir :

- Importer un package ;
- Inclure d'autres pages JSP ;
- Inclure des bibliothèques de balises (nous y reviendrons dans un prochain chapitre) ;
- Définir des propriétés et informations relatives à une page JSP.

Les directives sont comprises entre les balises `<%@` et `%>`, et hormis la directive d'**inclusion** de page qui peut être placée n'importe où, **elles sont à placer en tête** de page JSP.

Technologie JSP



Les directives JSP :

Directive taglib : `<%@ taglib %>`

Utilisé pour inclure et utiliser une bibliothèque personnalisée dans vos page JSP

Exemple d'inclusion d'une bibliothèque personnalisée nommée **myLib** avec un préfix **ml**

```
<%@ taglib uri="myLib.tld" prefix="ml" %>
```

Directive page : `<%@ page %>`

Définit des informations relatives à la page JSP.

Exemple d'importation de deux classes Java :

```
<%@ page import="java.util.List, java.util.Date" %>
```

Cette fonctionnalité n'est utile que si vous mettez en place du code Java dans votre JSP puisque notre objectif est de faire disparaître le Java de nos vues, nous allons très vite apprendre à nous en passer !

Technologie JSP



Les directives JSP :

Directive page : <%@ page %>

Il existe encore plus un ensemble des propriétés accessibles via cette directive page :

```
<%@ page
language      = ""
extends       = ""
import        = ""
session       = "    true || false"
isThreadSafe  = "    true || false"
isELIgnored   = "    true || false"
info          = ""
errorPage     = ""
contentType   = ""
pageEncoding  = ""
isErrorPage   = "    true || false"
%>
```

Exemple du pageEncoding. Qui s'ajoute automatiquement pour spécifier l'encodage « UTF-8 »

```
<%@page pageEncoding="UTF-8"%>
```

Technologie JSP



Les directives JSP :

Directive page : <%@ page isErrorPage="true" %> et <%@ page isErrorPage="false" %>

Page.jsp

```
<%@ page isErrorPage="error.jsp" %>
<!DOCTYPE html>
<html>
  <head>
    <title>Test Error Tag</title>
  </head>
  <body>
    <h1> Welcome </h1>
    <div>
      <%= 1/0 %>    <%-- L'erreur affichera la page error.jsp --%
    </div>
  </body>
</html>
```

error.jsp

```
<%@ page isErrorPage="true" %>
<div style="text-align: center; font-family: Optima;">
  <h1 style="color: white; background-color: red; font-size: 50px;">
    Error
  </h1>
  <h2 style="color: red; background-color: yellow;">
    <% exception.getMessage() %>
    <%-- te fait de faire true en haut nous permet
    d'utiliser un objet exception
    --%>
  </h2>
</div>
```

Error

/ by zero

Technologie JSP



Les directives JSP :

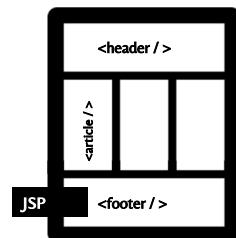
Directive include : <%@ include %>

Nos vues correspondent rarement à une **JSP** constituée d'un **seul bloc**. Cela permet notamment de pouvoir réutiliser certains blocs dans plusieurs vues différentes (ex. Les menus...)

Pour permettre un tel découpage, la technologie **JSP** met à votre disposition une balise qui inclut le contenu d'un autre fichier dans le fichier courant.

Via le code suivant par exemple, on peut inclure une feuille de style **css** interne dans notre page **JSP**.
(mais cela pourrait très bien être une page **HTML** ou **JSP** ou autre)

```
<style>
<%@ include file="css/bootstrap.min.css" %>
</style>
```



En pratique, il est très courant de découper littéralement une page web en plusieurs fragments, qui sont ensuite rassemblés dans la page finale à destination de l'utilisateur.

Technologie JSP



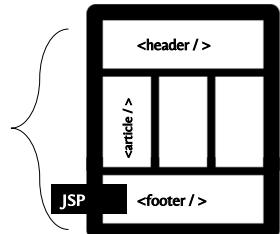
Directive include : <%@ include %>

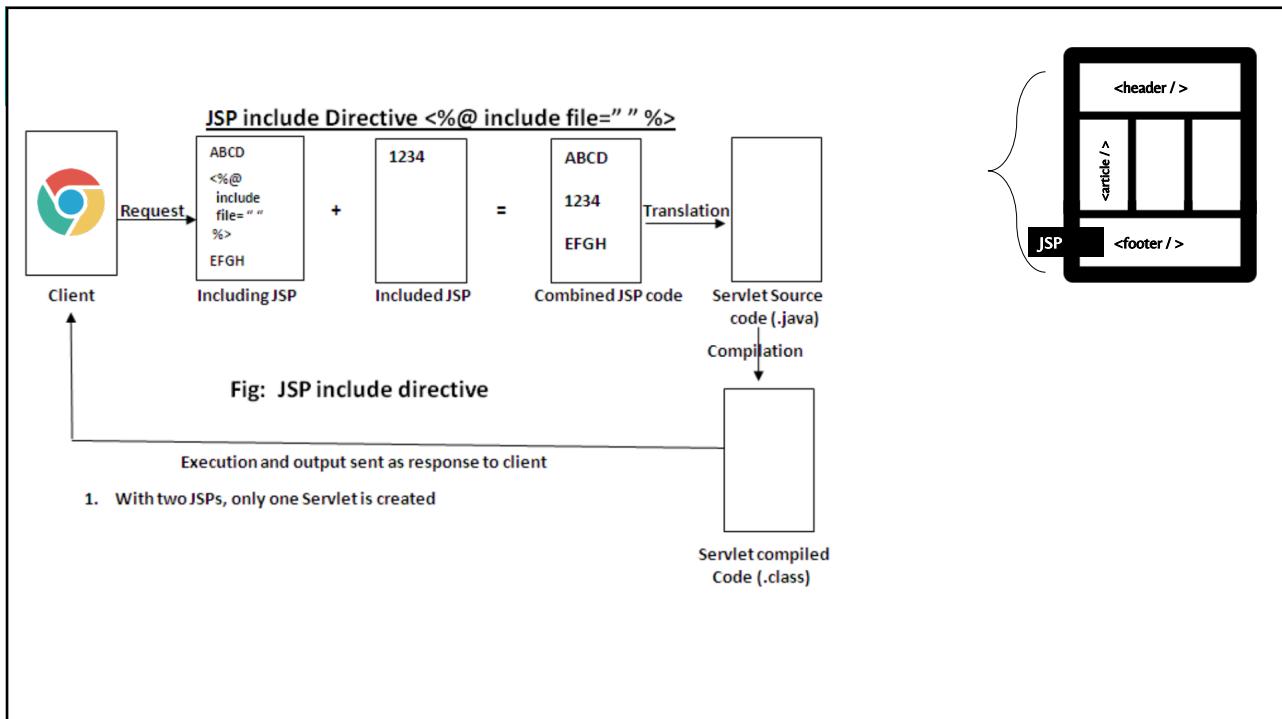
La subtilité à retenir, c'est que cette directive ne doit être utilisée que pour inclure du contenu "**statique**" dans votre page : i.e. **headers** ou le **footers** de la page, très souvent identiques sur l'intégralité des pages du site.

Statique : l'inclusion est réalisée au moment de la **compilation** ;

Alors si le code du fichier est changé par la suite, les changements sur la page l'incluant n'auront lieu qu'après **une nouvelle compilation** !

Cette directive peut être vue comme un simple **copier-coller** d'un fichier dans l'autre





Technologie JSP

Include dynamique avec Action standard :

Action standard include : `<jsp: include page="" %>`

Une autre balise d'inclusion dite "standard" existe, et permet d'inclure du contenu de manière "dynamique". (**après la compilation**)
Le contenu sera ici chargé à l'**exécution**, et non à la **compilation** comme c'est le cas avec la directive précédente :

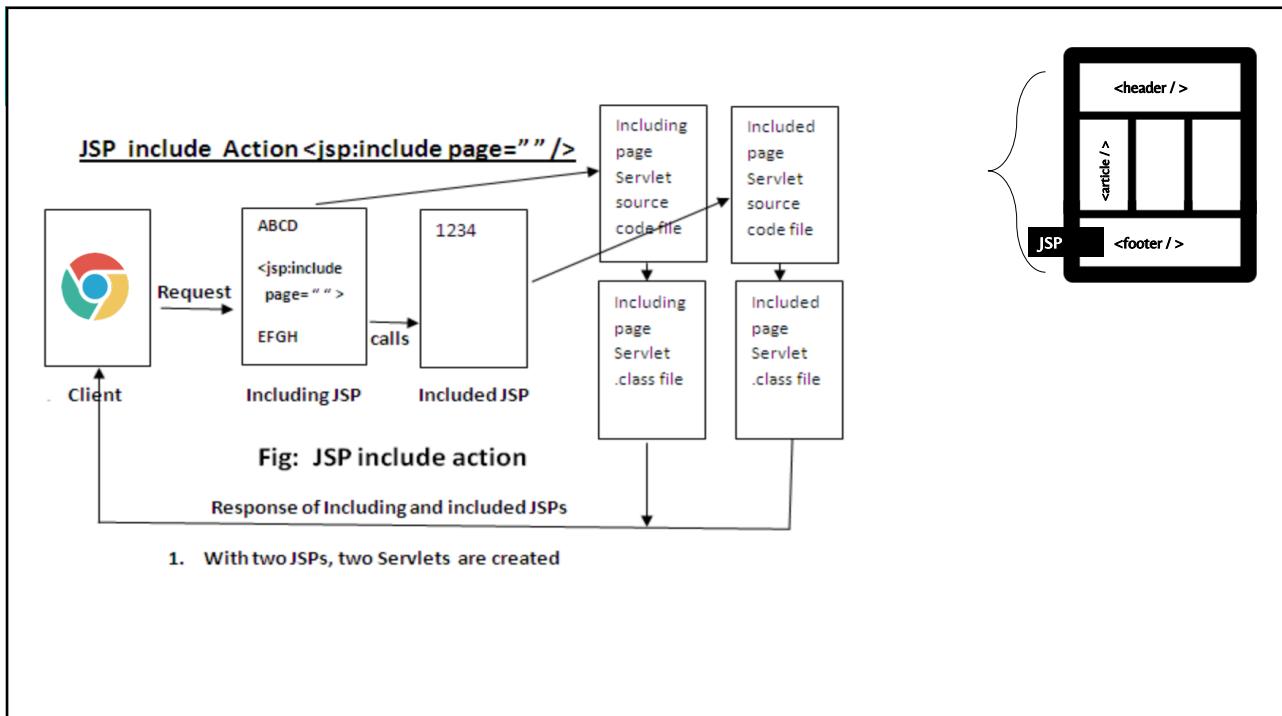
```

<%-- L'inclusion dynamique d'une page fonctionne par URL relative : --%>
<jsp:include page="page.jsp" />

<%-- Son équivalent en code Java est : --%>
<% request.getRequestDispatcher( "page.jsp" ).include( request, response ); %>

<%-- Et il est impossible d'inclure une page externe comme ci-dessous : --%>
<jsp:include page="http://www.emsi.ma" />

```



- With two JSPs, two Servlets are created

Technologie JSP

Include dynamique avec Action standard :

Action standard include : `<jsp: include page %>`

Inconvénient : Ce type d'inclusion ne prend pas en compte les imports et inclusions faits dans la page réceptrice.

List.jsp	Test.jsp	output
<pre><%@ page import="java.util.ArrayList"%> ArrayList<String> Days = new ArrayList<String>(); Days.add("Lundi"); Days.add("Mardi"); Days.add("Mercredi"); Days.add("Jeudi"); Days.add("Vendredi"); Days.add("Samedi"); Days.add("Dimanche"); <% for(int i = 0 ; i < Days.size() ; i++) out.println("" + Days.get(i) + ""); %> <h2> <% for(int i = 0 ; i < Days.size() ; i++) out.println("" + Days.get(i) + ""); %> </h2></pre>	<pre><!DOCTYPE html> <html> <head> <title>Test d'inclusion</title> </head> <body> <%@ include file="List.jsp" %> </body> </html></pre> <ul style="list-style-type: none"> Lundi Mardi Mercredi Jeudi Vendredi Samedi Dimanche <p>Avec la directive include ça marche très bien</p>	

Technologie JSP



Include dynamique avec Action standard :

Action standard include : `<jsp: include page %>`

Inconvénient : Ce type d'inclusion ne prend pas en compte les imports et inclusions faits dans la page réceptrice.

List.jsp

```
<%@page import="java.util.ArrayList"%>
<% ArrayList<String> Days = new ArrayList<String>();
Days.add( "Lundi" );
Days.add( "Mardi" );
Days.add( "Mercredi" );
Days.add( "Jeudi" );
Days.add( "Vendredi" );
Days.add( "Samedi" );
Days.add( "Dimanche" );
%>

<h2>
<% for( int i = 0 ; i < Days.size() ; i++ ) {
out.println("<li>" + Days.get( i ) + "</li>");
%>
</h2>
```

Test.jsp

```
<%@page import="java.util.ArrayList"%>
<!DOCTYPE html>
<html>
<head>
<title>Test d'inclusion</title>
</head>
<body>
<jsp:include page="List.jsp" />
</body>
</html>
```

output

```
java.lang.NullPointerException
```

Avec l'action standard **include** ça tient pas compte des l'import

Technologie JSP



Include dynamique avec Action standard :

- Les pages incluses via la balise `<jsp:include ... />` doivent en quelque sorte être "**indépendantes**" ; elles ne peuvent pas dépendre les unes des autres et doivent pouvoir être compilées séparément.
- Ce n'est pas le cas des pages incluses via la directive `<%@ include ... %>`
- Très bientôt, nous allons découvrir une meilleure technique d'inclusion de pages avec la JSTL !**

Technologie JSP



La gestion des objets par la technologie JSP

L'un des concepts les plus importants qui intervient dans la gestion des objets par la technologie JSP est **la portée des objets ou visibilité, ou scope** en anglais, qui définit tout simplement leur **durée de vie**.

Lors de la **transmission de données** de la servlet vers nos pages JSP, on a utilisé les attributs de requête. Tels objets sont accessibles via l'objet **HttpServletRequest**, et ne sont visibles que durant le traitement d'une même requête. Ils sont créés par le conteneur lors de la réception d'une **requête HTTP**, et disparaissent dès lors que le traitement de la requête est terminé.

Du coup, on a donc créé, sans le savoir, des objets ayant pour portée la requête !

Technologie JSP



La portée des objets

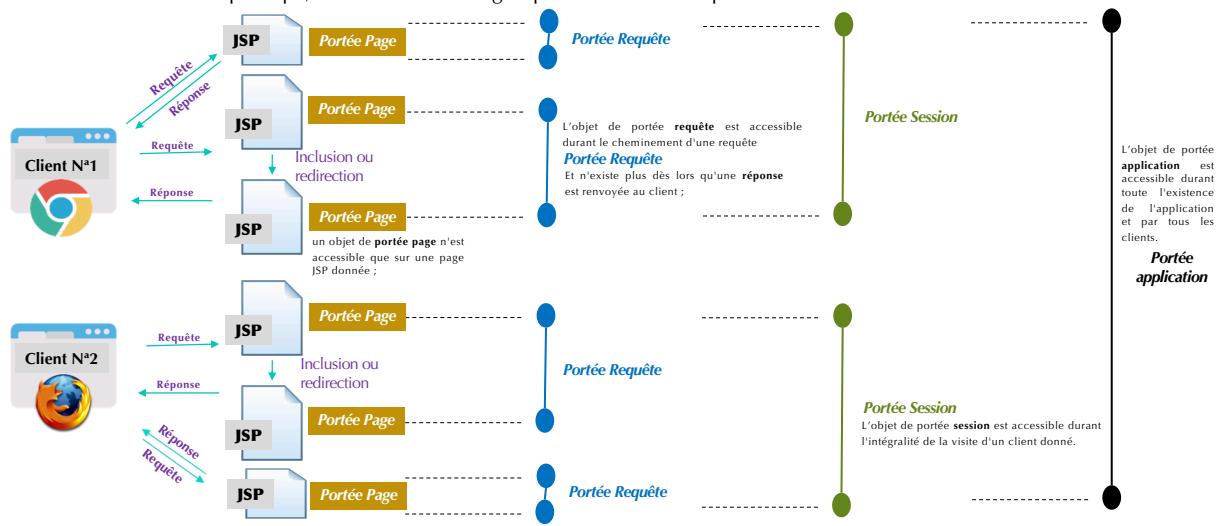
Il existe au total **quatre portées** différentes dans une application JEE:

- **page** (**JSP seulement**) : les objets dans cette portée sont uniquement accessibles dans la page JSP en question ;
 - **JSP seulement** veut dire qu'il n'est possible de créer et manipuler des objets de portée **page** que depuis une page JSP, ce n'est pas possible via une servlet.
 - Alors qu'il est possible de créer et manipuler des objets de portées **requête**, **session** ou **application** depuis une page **JSP** ou depuis une servlet.
- **requête** : les objets dans cette portée sont uniquement accessibles durant l'existence de la requête en cours ;
- **session** : les objets dans cette portée sont accessibles durant l'existence de la session en cours ;
 - Une session est l'objet associé à un utilisateur, elle existe tant qu'il utilise l'application, elle peut expirer lorsqu'il ferme son navigateur, reste inactif trop longtemps, ou encore lorsqu'il se déconnecte.
 - une session correspond en réalité à un navigateur particulier, plutôt qu'à un utilisateur.
- **application** : les objets dans cette portée sont accessibles durant toute l'existence de l'application.

Technologie JSP

La portée des objets

Pour visualiser bien le principe, voici un schéma regroupant les différentes portées existantes :



Exemple pratique

Technologie

On envoie le string **login** comme paramètre de **requête** vers la servlet TestServlet

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <a href="Home.jsp">Home Page</a>
    <h5>Connected as : <%=session.getAttribute("login")%></h5>
    <form action="ts" method="post">
        <span> UserName : </span>
        <input type="text" name="login"/>
        <button> Connect </button>
    </form>
</body>
</html>
```

Home Page

Connected as :

UserName : Connect

Exemple pratique

Technologie 

TestServlet.java

On crée une session et on stocke le paramètre envoyé dans la session

```
@WebServlet("/ts")
public class TestServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private static final String VUE = "Test.jsp";

    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
        throws ServletException, IOException {

        String lg = request.getParameter("login");
        HttpSession session = request.getSession();
        session.setAttribute("login", lg);
        request.getRequestDispatcher(VUE).forward(request, response);
    }
}
```

Exemple pratique

Technologie 

Login.jsp

On affiche la donnée du login stocké dans la session

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <a href="Home.jsp">Home Page</a>
    <h5>Connected as : <%=session.getAttribute("login")%></h5>
    <form action="ts" method="post">
        <span> UserName : </span>
        <input type="text" name="login"/>
        <button> Connect </button>
    </form>
</body>
</html>
```

La valeur sauvegardé dans la session reste affiché durant toute la session de l'utilisateur.
Vous pouvez tester en faisant un rafraîchissement de la page

Home Page

Connected as : Admin

UserName :

Admin

Connect

Exemple pratique

```
Home.jsp
```

On peut même tester avec une autre page accessible par cet utilisateur

```
<!DOCTYPE html>
<html>
  <head>
    <title>Home</title>
  </head>
  <body>
    <a href="Login.jsp">Login Page</a>
    <h5>Connected as : <%=session.getAttribute("login")%></h5>
    <h1> Welcome to Home Page </h1>
  </body>
</html>
```



Login Page	Connected as : Admin
Welcome to Home Page	

Technologie JSP



Les actions standard

On a déjà fait connaissance dans le chapitre précédent de l'action standard `<jsp:include>`

On va présenter dans ce chapitre d'autres **actions standard** de la technologie **JSP**, parmi lesquelles ceux utilisés pour accéder aux objets **Java Bean** depuis nos pages JSP.

Mais Notez bien que on va présenter par la suite d'autres manière plus simples pour accéder au Java Bean.

On présente quatre action standard : `<jsp:useBean>`, `<jsp:getProperty>`, `<jsp:setProperty>` et enfin `<jsp:forward>`.

Les actions standard



- L'action standard useBean (<jsp:useBean>):**

Cette action standard permet d'utiliser un **Bean**, ou de le créer s'il n'existe pas, depuis une page **JSP**

L'action suivante récupère un Bean de type Etudiant nommé "Mehdi" dans la portée requête s'il existe, ou en crée un sinon.

```
<jsp:useBean id="Mehdi" class="web.model.Etudiant" scope="request" />
```

- id** : Est le nom du Bean à récupérer, ou le nom que vous allez donner au Bean à créer.
- class** : Class du Bean, obligatoire pour créer un Bean, mais pas si vous souhaitez simplement récupérer un Bean existant.
- scope** : Correspond à la portée de l'objet, si cet attribut n'est pas renseigné, alors le scope par défaut sera limité à la **page**

Elle a le même effet que le code Java suivant :

```
<% web.model.Etudiant Mehdi = (web.model.Etudiant) request.getAttribute("Mehdi");
if (Mehdi == null) {
    Mehdi = new web.model.Etudiant();
    request.setAttribute("Mehdi", Mehdi);
}%>
```

- Soit la class définissant un Bean **Etudiant**

```
public class Etudiant implements Serializable{
    private static final long serialVersionUID = 1L;
    private static int increment_PK = 0;

    private int id;
    private String nom, prénom;

    public int getId() { return id; }
    public String getNom() { return nom; }
    public String getPrenom() { return prénom; }

    public void setId(int id) { this.id = id; }
    public void setNom(String nom) { this.nom = nom; }
    public void setPrenom(String prénom) { this.prénom = prénom; }

    public Etudiant() {
        this.id = ++increment_PK;
        this.nom = "nom_inconnu";
        this.prénom = "prénom_inconnu";
    }

    @Override
    public String toString() { return "Etudiant nº" + id +
        " [ nom : " + nom +
        ", prénom : " + prénom +
        "]"; }
}
```



Les actions standard



- L'action standard getProperty (<jsp:getProperty>):**

Cette action standard permet d'obtenir la valeur d'une des propriétés d'un **Bean**, depuis une page **JSP**

L'action suivante L'action suivante affiche le contenu de la propriété 'prénom' du Bean 'Mehdi' :

```
<jsp:getProperty name="Mehdi" property="prénom" />
```

- name** : Correspond au nom réel du **Bean**, en l'occurrence **l'id** que l'on a saisi auparavant dans la balise de récupération du **Bean**.
- property** : Contient le nom de la propriété dont on souhaite afficher le contenu

Elle a le même effet que le code Java suivant :

```
<%= Mehdi.getPrénom() %>
```

- Soit la class définissant un Bean **Etudiant**

```
public class Etudiant implements Serializable{
    private static final long serialVersionUID = 1L;
    private static int increment_PK = 0;

    private int id;
    private String nom, prénom;

    public int getId() { return id; }
    public String getNom() { return nom; }
    public String getPrenom() { return prénom; }

    public void setId(int id) { this.id = id; }
    public void setNom(String nom) { this.nom = nom; }
    public void setPrenom(String prénom) { this.prénom = prénom; }

    public Etudiant() {
        this.id = ++increment_PK;
        this.nom = "nom_inconnu";
        this.prénom = "prénom_inconnu";
    }

    @Override
    public String toString() { return "Etudiant nº" + id +
        " [ nom : " + nom +
        ", prénom : " + prénom +
        "]"; }
}
```



Les actions standard



- **L'action standard setProperty (<jsp:setProperty>)**

Cette action standard permet de modifier la valeur d'une des propriétés d'un **Bean**, depuis une page **JSP**

L'action suivante L'action suivante modifie le contenu des propriétés 'nom' et 'prénom' du Bean 'Mehdi' :

```
<jsp:setProperty property="nom" name="Mehdi" value="Assali"/>
<jsp:setProperty property="prénom" name="Mehdi" value="Mehdi"/>
```

- </> **name** : Correspond au nom réel du **Bean**, en l'occurrence **'Mehdi'** que l'on a saisi auparavant dans la balise de récupération du **Bean**.
- property** : Contient le nom de la propriété dont on souhaite afficher le contenu
- value** : Contient la valeur à attribuer au propriété

Elle a le même effet que le code Java suivant :

```
<% Mehdi.setNom("Assali"); Mehdi.setPrénom("Mehdi"); %>
```

</>

```
<!-- cette action associe automatiquement la valeur récupérée
depuis chaque paramètre de la requête à la propriété de même nom :
-->
<jsp:setProperty name="Mehdi" property="*" />

<!-- Équivalente à :
-->
<%
    Mehdi.setNom(request.getParameter("nom"));
    Mehdi.setPrénom(request.getParameter("prénom"));
%>
```

</>

- Soit la class définissant un Bean **Etudiant**

```
public class Etudiant implements Serializable{
    private static final long serialVersionUID = 1L;
    private static int increment_PK = 0;

    private int id;
    private String nom, prénom;

    public int getId() { return id; }
    public String getNom() { return nom; }
    public String getPrénom() { return prénom; }

    public void setId(int id) { this.id = id; }
    public void setNom(String nom) { this.nom = nom; }
    public void setPrénom(String prénom) { this.prénom = prénom; }

    public Etudiant() {
        this.id = ++increment_PK;
        this.nom = "nom_inconnu";
        this.prénom = "prénom_inconnu";
    }

    @Override
    public String toString() { return "Etudiant n°" + id +
        " [ nom : " + nom +
        ", prénom : " + prénom +
        "]"; }
}
```



Les actions standard



- **L'action standard forward (<jsp:forward>.):**

Cette action standard permet d'effectuer une navigation ou redirection vers une autre page.

C'est une action qui s'effectue côté serveur du coup il est impossible de faire une redirection vers une page extérieure à l'application.

L'action de **Forwarding** est ainsi limitée aux pages présentes dans le contexte de la servlet ou de la JSP utilisée :

Le **Forwarding** vers une page de l'application fonctionne par URL relative :

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

Elle a le même effet que le code Java suivant :

```
<% request.getRequestDispatcher( "/page.jsp" ).forward( request, response ); %>
```



Sachez enfin que lorsque vous utilisez le **Forwarding**, **le code présent après cette balise dans la page n'est pas exécuté**.