



# Java pour le développement d'applications Web : Java EE

## Java Server Pages (JSP)

Mickaël BARON - 2007 (Rév. Août 2009)  
<mailto:baron.mickael@gmail.com> ou <mailto:baron@ensma.fr>

# Licence

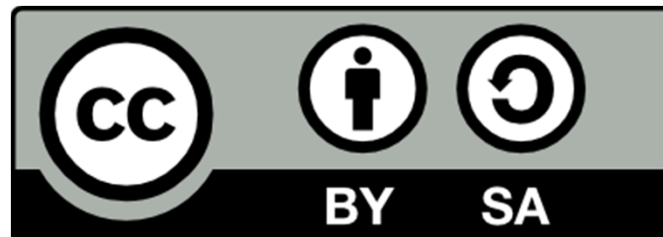
---

## Creative Commons

*Contrat Paternité*

*Partage des Conditions Initiales à l'Identique*

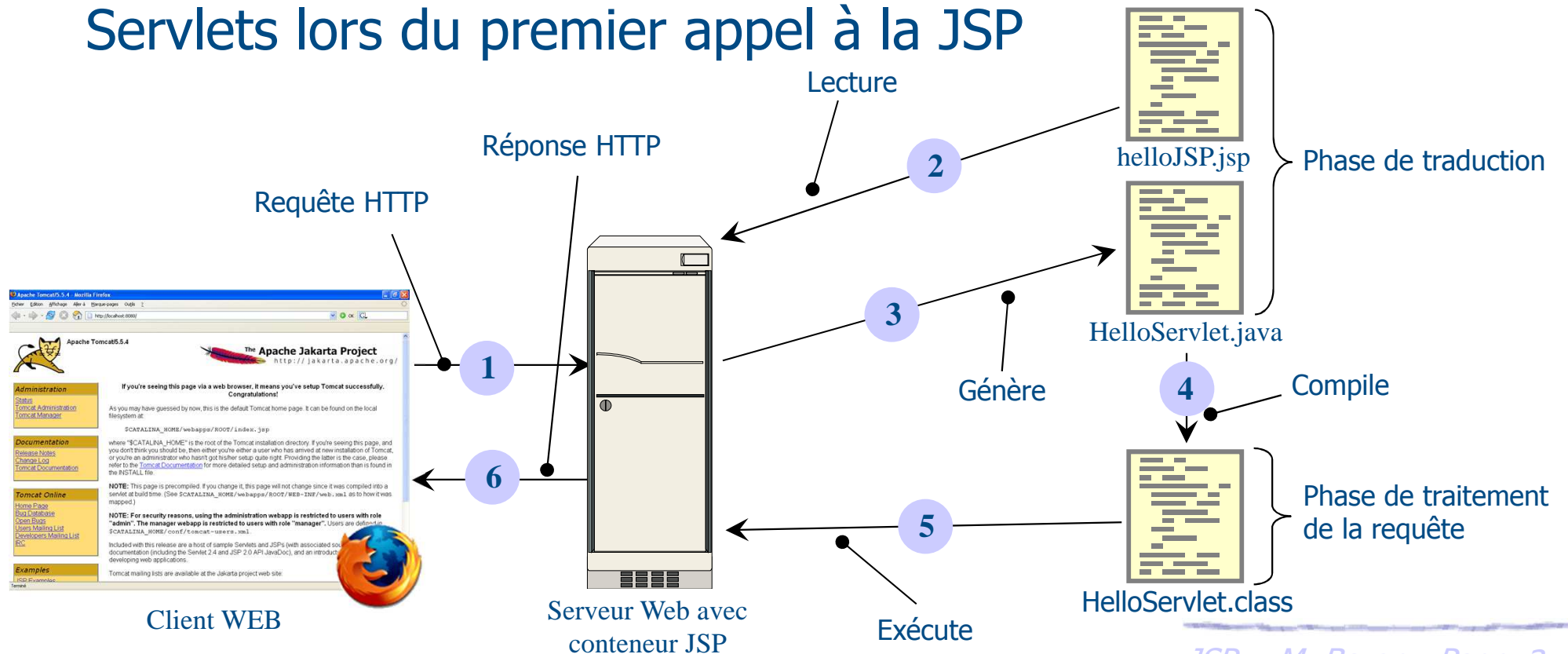
2.0 France



<http://creativecommons.org/licenses/by-sa/2.0/fr>

# C'est quoi JSP ?

- **JSP = Java Server Pages**
- Une JSP est un fichier contenant du code HTML et des fragments de code Java exécutés sur le moteur de Servlets
- Comparable aux langages côté serveur de type PHP, ASP, ...
- Les pages JSP sont converties en Servlet par le moteur de Servlets lors du premier appel à la JSP



# Ok mais ... HelloWorld avec une Servlet

## ➤ Exemple : *HelloWorld* version Servlet

### ➤ Besoin de modifier le fichier web.xml

```
public class HelloWorldServlet extends HttpServlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("    <head>");
        out.println("        <title>Bonjour tout le monde</title>");
        out.println("    </head>");
        out.println("    <body>");
        out.println("        <h1>Bonjour tout le monde</h1>");
        out.println("        Nous sommes le " + (new java.util.Date().toString()) +
            " et tout va bien.");
        out.println("    </body>");
        out.println("</html>");
    }
}
```

La partie structure du document HTML  
doit être précisée à l'aide de l'affichage  
de sortie : **devient vite contraignant**

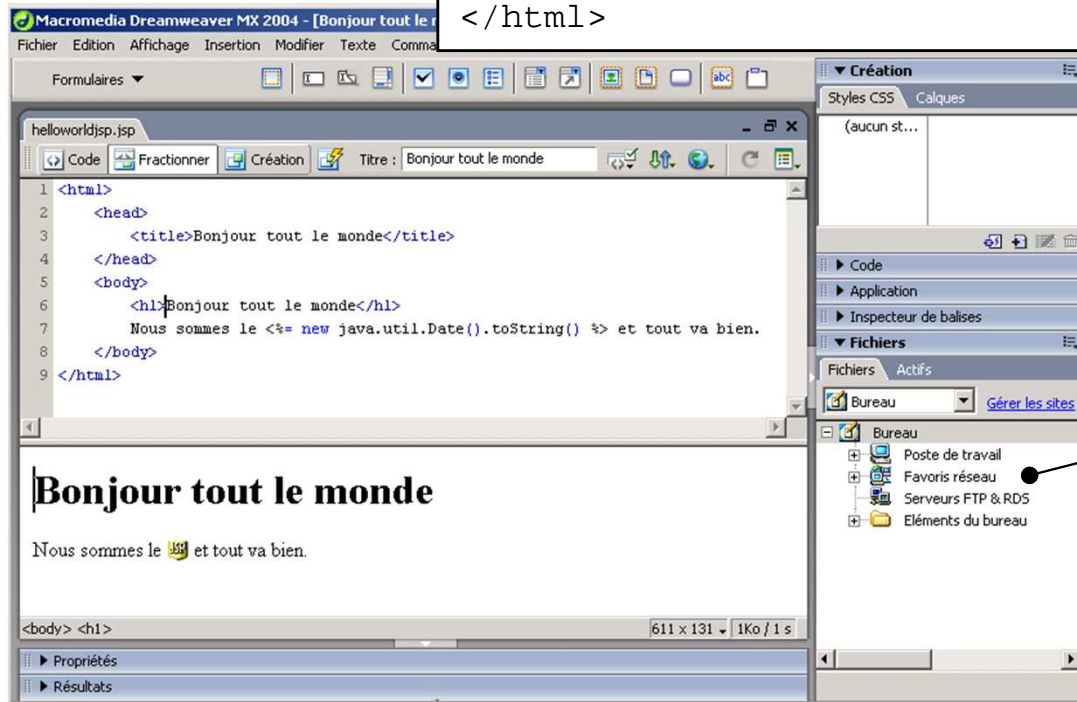
# HelloWorld avec une JSP

## ➤ Exemple : *HelloWorld* version JSP

- *helloworldjsp.jsp* doit être placé à la racine de l'application WEB
- Pas besoin de modifier le fichier web.xml

Ajout de fragment  
de code Java

```
<html>
<head>
<title>Bonjour tout le monde</title>
</head>
<body>
<h1>Bonjour tout le monde</h1>
Nous sommes le <%= new java.util.Date().toString() %> et tout va bien.
</body>
</html>
```



Utilisation d'un outil d'aide à la  
conception de page WEB avec  
prise en charge de code JSP

# HelloWorld avec une JSP après la génération

## ► Exemple : *HelloWorld* version Servlet

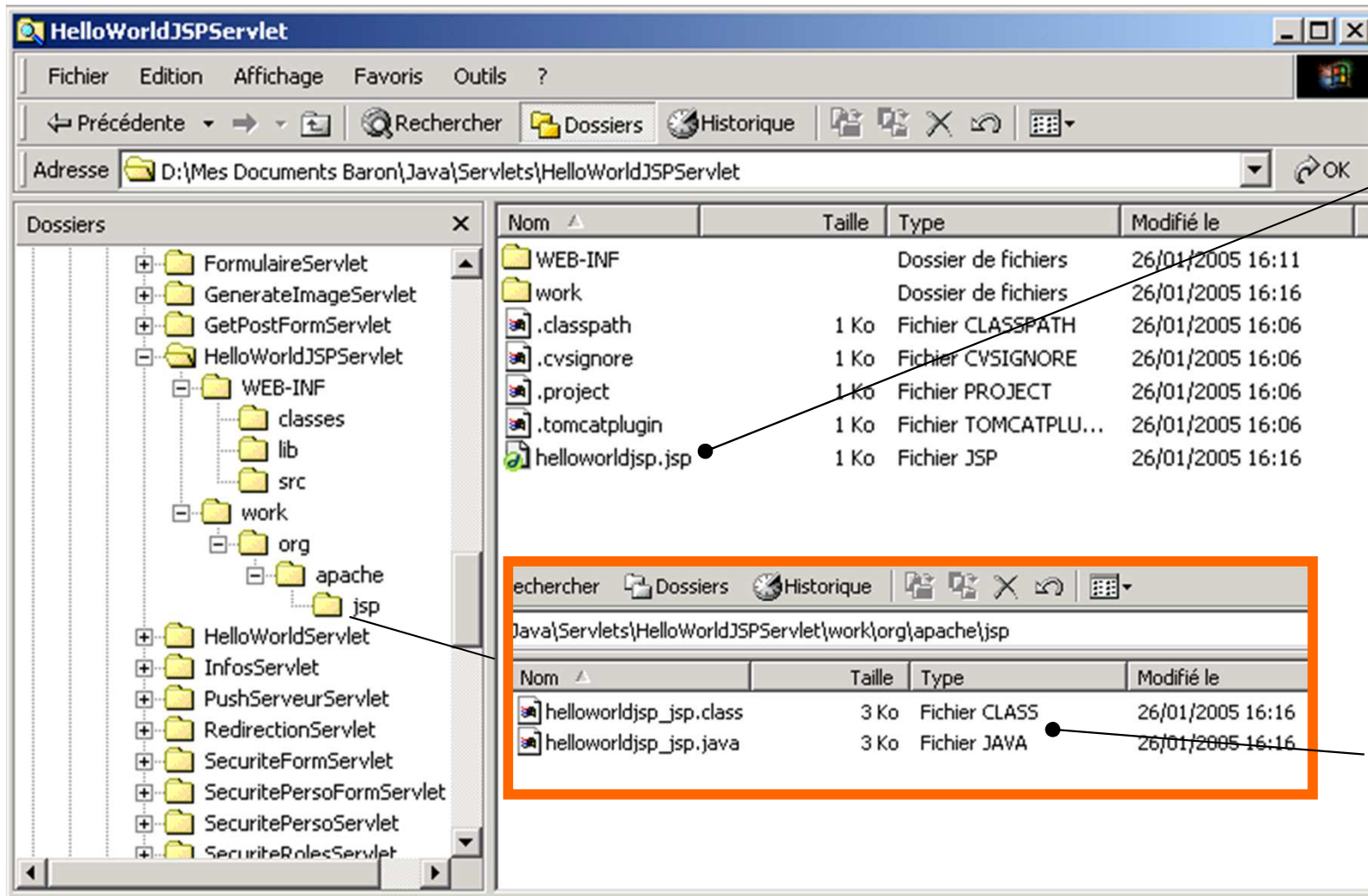
```
public final class helloworldjsp_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
        HttpSession session = null;
        ...
        try {
            ...
            _jspx_out = out;
            out.write("<html>\r\n");out.write("\t<head>\r\n");
            out.write("\t\t<title>Bonjour tout le monde</title>\r\n");
            out.write("\t</head>\r\n");out.write("\t<body>\r\n");
            out.write("\t\t<h1>Bonjour tout le monde</h1>\r\n");
            out.write("\t\tNous sommes le ");out.print( new java.util.Date().toString() );
            out.write(" et tout va bien.\r\n");out.write("\t</body>\r\n");out.write("</html>");
        } catch (Throwable t) {
            if (!(t instanceof SkipPageException)){
                out = _jspx_out;
                ...
                if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
            }
        } finally {
            if (_jspxFactory != null) _jspxFactory.releasePageContext(_jspx_page_context);
        }
    }
}
```

Hérite de *javax.servlet.jsp.HttpJspPage*  
implémente la méthode *\_jspService(...)*  
équivalente à *service(...)*

# HelloWorld et les fichiers Servlet, JSP et Servlet générée

## ➤ Arborescence d'une application WEB : le retour



Les fichiers JSP sont à la racine de l'application WEB au même endroit que les fichiers HTML

Les Servlets générées sont placées dans le répertoire *work*

# Les Tags JSP

## ➤ Les **Tags** permettent de différencier le code HTML au code Java

### ➤ Tag de directive :

➤ `<%@ ... %>`

### ➤ Tag de commentaire :

➤ `<%-- blabla --%>`

### ➤ Tag de déclaration :

➤ `<%! ... %>`

### ➤ Tag de Scriptlet :

➤ `<% ... %>`

### ➤ Tag d'expression :

➤ `<%= ... %>`

**Attention ne pas se tromper dans la nomination des tags. Ils ont tous une signification différente**



Éléments de scripts



# Directives JSP

- Les directives contrôlent comment le serveur WEB doit générer la Servlet
- Elles sont placées entre les symboles `<%@` et `%>`
- Les directives suivantes sont disponibles
  - **include** : indique au compilateur d'inclure un autre fichier

```
<%@ include file="unAutreFichier" %>
```



Étudié en fin  
de partie

- **taglib** : indique une bibliothèque de balises a utiliser

```
<%@ taglib prefix="myprefix" uri="taglib/mytag.tld" %>
```

- **page** : définit les attributs spécifiques à une page (voir après)

## Directives JSP : include

---

- Cette inclusion se fait au *moment de la conversion*

```
<%@ include file="unAutreFichier" %>
```

- Tout le contenu du fichier externe est inclus comme s'il était saisi directement dans la page JSP
- Ne concerne que les ressources contenues dans le contexte
- La racine du chemin du fichier à inclure est la racine du contexte
- Pas de séparation de la portée des variables

**Il n'y a pas la possibilité  
de construire une chaîne  
d'interrogation**



# Directives JSP : include

## ➤ Exemple : inclusions par la directive JSP

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

Le fichier  
*entete.html*

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

Bonjour <%= mon_nom %>

<%@ include file = "/piedpage.html" %>
```

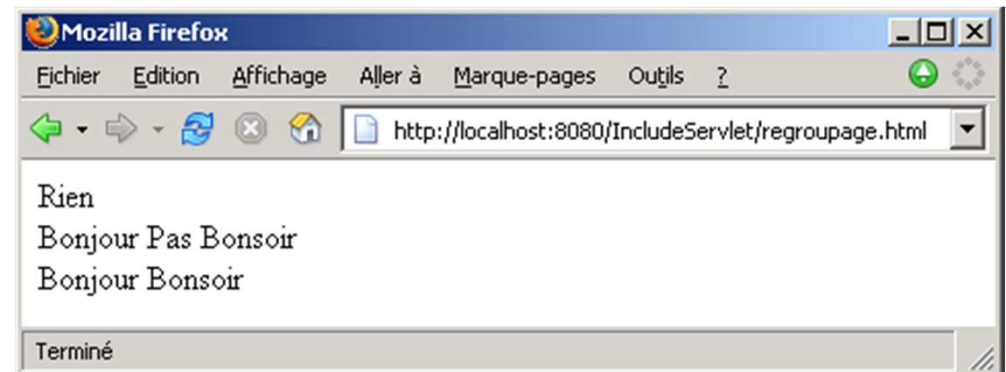
La variable *name*  
est définie dans  
*corps.jsp*

```
<%! String mon_nom; %>
<% mon_nom = "Baron Mickael"; %>
```

Le fichier *corps.jsp*

```
Je suis dans le pied de page.
</BODY>
</HTML>
```

Le fichier  
*piedpage.html*



# Directives JSP : page

- La directive **page** définit les attributs spécifiques à une page
  - **import** : importe un paquetage Java. Cette directive résulte en une instruction *import* dans la Servlet

```
<%@ page import="java.util.*, java.text.*" %>
```

- **langage** : définit le langage de script utilisé dans la page
- **contentType** : définit le type de contenu de la page générée

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

- **errorPage** : indique la page à afficher si une exception se produit pendant le traitement de la requête HTTP

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

- **isErrorPage** : vaut *true* si la page est une erreur et *false* pour une page normale

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

# Éléments de scripts JSP : commentaire

- Cet élément de script est utilisé pour faire un commentaire dans le code JSP
- Le texte dans un commentaire JSP ne sera pas envoyé au client ni compilé dans la Servlet
- Les commentaires sont placés entre les symboles **<%-- et --%>**

```
<html>
  <head>
    <title>Bonjour tout le monde</title>
  </head>
  <body>
    <!-- affichage d'un message classique -->
    <h1>Bonjour tout le monde</h1>
    Nous sommes le <%= new java.util.Date().toString() %> et tout va bien.
    <!-- Utilisation de la classe Date --%>
  </body>
</html>
```

Source de : http://localhost:8080/HelloWorldJSPServlet

```
<html>
  <head>
    <title>Bonjour tout le monde</title>
  </head>
  <body>
    <!-- affichage d'un message classique -->
    <h1>Bonjour tout le monde</h1>
    Nous sommes le Wed Jan 26 18:54:15 CET 2005 et tout va bien.
  </body>
</html>
```

Les commentaires JSP  
n'apparaissent pas dans  
le code HTML du client

# Éléments de scripts JSP : déclaration

- Une déclaration permet d'insérer du code dans la classe de la Servlet
- Les déclarations sont placées entre **<%!** et **%>**
- Elle peut être utilisée pour
  - Déclarer un attribut de classe
  - Spécifier et implémenter des méthodes

```
<%!
    private int count = 0;
    private int incrementCount() { return count++; }
%>
```

- Les attributs et les méthodes déclarées dans la page JSP sont utilisables **dans toute la page JSP**
- Possibilité de redéfinir des méthodes *jspInit()* et *jspDestroy()*



**A voir plus tard au moment  
du cycle de vie d'une JSP**

# Éléments de scripts JSP : scriptlet

- C'est un bloc de code Java qui est placé dans `_jspService(...)` de la Servlet générée (équivalent à `service(...)`)
- Les scriptlets sont placées entre les symboles `<%` et `%>`
- Tout code java a accès :
  - aux attributs et méthodes définis par le tag déclaration `<%! ... %>`
  - aux objets implicites



Code HTML

**En général les scriptlets ne sont jamais utilisées, préférez l'utilisation des balises personnalisées (Taglib)**

```
...
<%
    for (int i = 0; i < 5 ; i++) {
%>
HelloWorld<br>
<%
    incrementCount(); }
%>
...
```

Code JSP : scriptlet

Méthode déclarée par l'élément de scripts **déclaration** précédent

**Ne pas oublier le ;**



# Éléments de scripts JSP : expression

- Sert à évaluer une expression et à renvoyer sa valeur
- Les expressions sont placées entre les symboles `<%= %>`
- Retourne une valeur *String* de l'expression
- Correspond à une scriptlet comme `<% out.println(...); %>`
- Se transforme en `out.println("...");` dans la méthode `_jspService(...)` après génération

Éléments de scripts  
JSP : **scriptlet**

```
...
<%
    String[] noms={"mickey","donald"};
    for (int i = 0 ; i < noms.length ; i++) {
%>
    Le <%= i %> ème nom est <%= noms[i] %>
<% } %>
...
```

Éléments de scripts  
JSP : **expression**

**Ne pas ajouter de ; à la fin d'un élément script expression**





# Éléments de scripts JSP : bilan

## ➤ Éléments de scripts et génération de Servlet

Déclaration

Scriptlet

```
public final class example_jsp extends HttpJspBase {
    • String contenu[] = {"raoul","john","nicolas"};
    public void _jspService(HttpServletRequest req,
        HttpServletResponse res) throws IOException, ... {

        out.write("\t\t<title>Bonjour tout le monde</title>\r\n");
        out.write("\t</head>\r\n"); out.write("\t<body>\r\n");

        • for (int i = 0; i <contenu.length; i++) {
            out.write("\t\t\tLe ");
            out.print( i+1 );
            out.write(" Ã"me nom est ");
            out.print( contenu[i] );
            out.write(" <p>\r\n");
            out.write("\t\t");
        }
```

```
<html>
<head>
<title>Bonjour tout </title>
</head>
<body>
    • <%= String contenu[] = {"raoul","john","nicolas"}; %>
    <%
    for (int i = 0; i <contenu.length; i++) {
    %>
        Le <%= i+1 %> ème nom est <%= contenu[i] %> <p>
    <% } %>
</body>
</html>
```

Expression

# Éléments de scripts JSP : scriptlet et objets implicites

- Les objets implicites sont les objets présents dans la méthode *service(...)* qui ont été employés dans la partie Servlet
- Ils sont identifiés par des noms de variables uniques :
  - **request** : requête courante
  - **response** : réponse courante
  - **session** : session courante
  - **out** : flot de sortie permet l'écriture sur la réponse
  - **application** : contient des méthodes `log()` permettant d'écrire des messages dans le journal du contenu (*ServletContext*)
  - **pageContext** : utilisé pour partager directement des variables entre des pages JSP et supportant les beans et les balises
  - **exception** : disponible uniquement dans les pages erreurs donnant information sur les erreurs

**Les objets implicites ne sont utilisables que dans les éléments de scripts JSP de type scriptlet et expression (dans la méthode *service(...)*)**

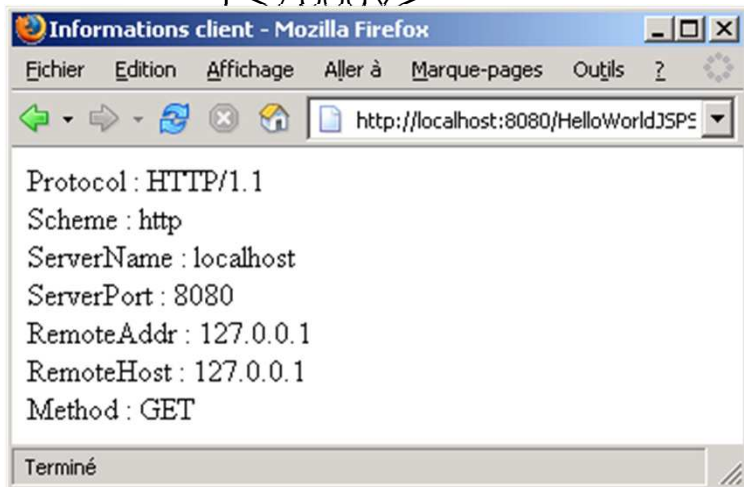


# Éléments de scripts JSP : scriptlet et objets implicites

## ➤ Exemple : JSP qui récupère des informations du client

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

<html>
<head>
    <title>Informations du client</title>
</head>
<body bgcolor="white">
    Protocol : <%= request.getProtocol() %><br>
    Scheme : <%= request.getScheme() %><br>
    ServerName : <%= request.getServerName() %><br>
    ServerPort : <% out.println(request.getServerPort()); %><br>
    RemoteAddr : <% out.println(request.getRemoteAddr()); %><br>
    RemoteHost : <% out.println(request.getRemoteHost()); %><br>
    Method : <%= request.getMethod() %><br>
</body>
```



Objets implicites uniquement  
visibles dans une scriptlet ou  
une expression JSP

# Cycle de vie d'une JSP

- Le cycle de vie d'une Java Server Page est identique à une Servlet
  - La méthode *jspInit()* est appelée après le chargement de la page
  - La méthode *\_jspService()* est appelée à chaque requête
  - La méthode *jspDestroy()* est appelé lors du déchargement (fermeture d'une base de données)
- Possibilité de redéfinir dans le code JSP les méthodes *jspInit()* et *jspDestroy()* en utilisant un élément de scripts **déclaration**

Redéfinition de la méthode *jspInit()*



```
<html>
<head>
<title>Bonjour tout </title>
</head>
<body>
<%! public void jspInit() {
    compteur = 23;
} %>
<%! String compteur; %>
La valeur du compteur est <%= compteur %>
</body>
</html>
```


# Cycle de vie d'une JSP

## ➤ Exemple : compteur avec une initialisation et une destruction

```
<%@ page language="java" contentType="text/html" %>
<%@ page import="java.util.Date" %>
<%!
    int global_counter = 0;
    Date start_date;

    public void jspInit() {
        start_date = new Date();
    }

    public void jspDestroy() {
        ServletContext context = getServletConfig().getServletContext();
        context.log("test.jsp a été visitée " + global_counter + "fois entre le
            " + start_date + " et le " + (new Date()));
    }
%>
<html>
<head><title>Page avec un compteur</title></head>
<body bgcolor="white">
    Cette page a été visitée : <%= ++global_counter %> fois depuis le <%=
        start_date %>.
</body></html>
```



**L'objet implicite application n'est pas disponible ici, ne pas l'utiliser !!!!**

# Technique de gestion des erreurs

- Permet de contrôler la gestion des erreurs pendant l'exécution de l'application WEB
- Les erreurs sont déclenchées par l'intermédiaire des exceptions et transmises à une page d'erreur
- La définition de la page d'erreur se fait par la directive *page* et l'attribut *errorPage*

```
<%@ page errorPage="erreur.jsp" %>
... code JSP lançant l'exception
```

Lancement d'une exception possible dans le reste de la page JSP

Possibilité de transmettre des informations à la page d'erreur par la méthode GET  
 ... "erreur.jsp?debug=log" %>

**Impossibilité dans une page JSP de déclarer plus d'une page d'erreur**



# Technique de gestion des erreurs

- Une page JSP est définie comme une page erreur par la directive *page* et l'attribut *isErrorPage*

```
<%@ page isErrorPage=true %>
... code JSP traitant l'erreur
```

Quand *false* il s'agit d'une page standard, *true* une page erreur

- Possibilité de manipuler l'exception qui a été lancée par l'objet implicite *exception* (*Exception*)
  - *exception.getMessage()* : renvoie le message d'erreur décrivant l'exception
  - *exception.printStackTrace()* : retourne la liste des appels de méthodes ayant conduit à l'exception

**Une page erreur doit être obligatoirement une page JSP**



**La référence *exception* n'est utilisable que dans les éléments de scripts Scriptlet et Expression**



# Technique de gestion des erreurs

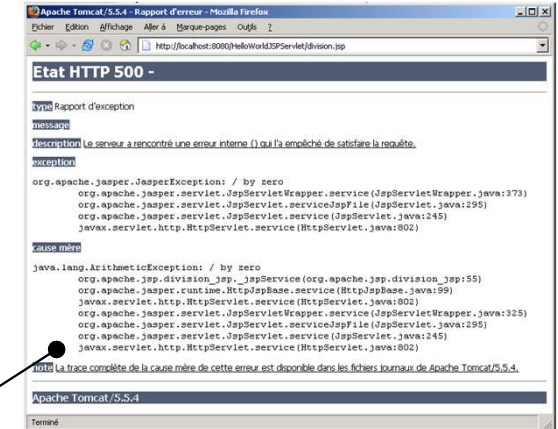
## ➤ Exemple : une division par zéro avec gestion de l'erreur

```
<%@ page language="java" contentType="text/html" %>
<%@ page errorPage="errorpage.jsp" %>

<html>
<head>
<title>Page avec une erreur</title>
</head>
<body bgcolor="white">

<% int var = 90; %>
Division par <% var = var / 0; %> <%= var %>
</body>
</html>
```

Division par zéro :  
exception lancée

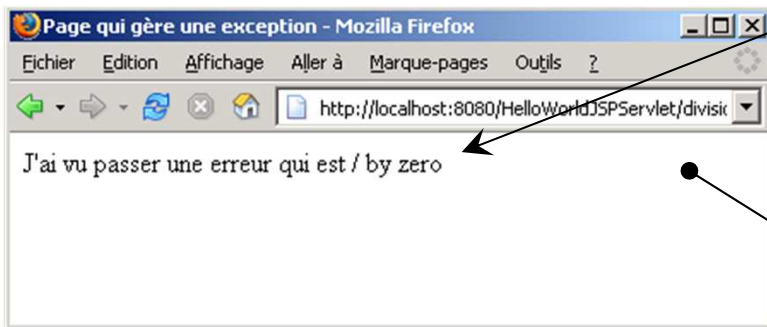


Résultat sans la gestion  
de l'exception

```
<%@ page language="java" contentType="text/html" %>
<%@ page isErrorPage="true" %>

<html>
<head>
<title>Page gérant une erreur</title>
</head>
<body bgcolor="white">
    J'ai vu passer une erreur qui est <%=
    exception.getMessage() %>
</body>
</html>
```

Résultat avec la gestion  
de l'exception





# JSP et Actions

- Les actions permettent de faire des traitements au moment où la page est demandée par le client
  - utiliser des Java Beans
  - inclure dynamiquement un fichier
  - rediriger vers une autre page
- Les actions sont ajoutées à la page JSP à l'aide d'une syntaxe d'éléments XML (définis par des balises personnalisées)



**L'inclusion et la redirection fonctionnent sur un principe commun évoqué dans la partie Servlet**

Balise ouvrante

`<ma:balise ... />`

Balise fermante

ou

**Deux écritures différentes, l'une quand il n'y a pas de corps et l'autre quand il y en a un**



`<ma:balise ... >`  
`...`  
`</ma:balise>`

corps

# Java Beans

---

- Permet de coder la logique métier de l'application WEB
- L'état d'un Bean est décrit par des attributs appelés propriétés
- La spécification des Java Beans définit les Beans comme des classes qui supportent principalement
  - Introspection : permet l'analyse d'un Bean (nombre de propriétés)
  - Événements : métaphore de communication
  - Persistance : pour sauvegarder l'état d'un Bean

# Java Beans

- Les Java Beans sont des classes Java respectant un ensemble de directives
  - 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
- Lecture et écriture des propriétés
  - *type getNomDeLaPropriété()* : pas de paramètre et son type est celui de la propriété
  - *void setNomDeLaPropriété(type)* : un seul argument du type de la propriété et son type de retour est void
- En option, un Java Beans implémente l'interface *java.io.Serializable* permettant la sauvegarde de l'état du Bean

**Respecter absolument  
la convention  
d'écriture**



# Java Beans

## ► Exemple : le retour de la classe *Voiture*

```
public class Voiture {  
    private int puissance;  
    private boolean est_demarree;  
    private double vitesse;  
  
    public void setDemarree(boolean p) {  
        est_demarree = p;  
    }  
    public boolean getDemarree() {  
        return est_demarree;  
    }  
    public void setVitesse(double p) {  
        vitesse = p;  
    }  
    public double getVitesse() {  
        return vitesse;  
    }  
    public int getPuissance() {  
        return puissance;  
    }  
}
```

Utilise le constructeur par défaut ne possédant aucun paramètre

Propriété **Demarree** visible en lecture et en écriture

Propriété **Vitesse** visible en lecture et en écriture

Propriété **Puissance** visible en lecture uniquement

# Java Beans et JSP

- Pour déclarer et allouer un Java Beans dans une page JSP il faut employer l'action `<jsp:useBean>`

```
<jsp:useBean id="Nom" class="Package.class" scope="attribut" />
```

- *id* = "nom de l'instance" : nom de l'instance pour identification
- *class* = "Nom de la classe" : package du Bean
- *scope* = "attribut" : champ d'existence de l'objet Bean
  - *request* : Bean valide pour la requête et peut être transmise (forward)
  - *page* : idem request sans transmission (le cas de l'objet *pageContext*)
  - *session* : Bean ayant la durée de vie de la session
    - Contenu dans le ServletContext de l'application web
  - *application* : Bean créée pour l'application WEB courante

# Java Beans et JSP : lecture propriétés

- Pour lire une propriété du Bean deux éléments sont utilisés
  - La référence du Bean définie par l'attribut *id*
  - Le nom de la propriété
- Deux manières existent pour interroger la valeur d'une propriété et la convertir en String
  - En utilisant un tag action `<jsp:getProperty>`

```
<jsp:getProperty name="référence Bean" property="nom propriété" />
```

Référence du Bean  
désignée dans l'attribut *id*

Nom de la propriété en  
minuscule

- En utilisant l'éléments de scripts JSP : expression

```
<%= nom_instance.getNomPropriete() %>
```

Référence du Bean  
désignée dans l'attribut *id*

Méthode liée à la propriété

# Java Beans et JSP : écriture propriétés

- Modification de la valeur d'une propriété en utilisant le tag action `<jsp:setProperty>`

```
<jsp:setProperty name="référence" property="nom propriété" param="param" value="valeur" />
```

Référence du Bean désignée dans l'attribut *id*

Nom de la propriété à modifier écrit en minuscule

Nom d'un paramètre de requête contenant la valeur pour la propriété indiquée

Valeur explicite à donner à la propriété. Ne peut pas être combiné à l'attribut *param*

- Modification de l'ensemble de propriétés suivant les paramètres fournis par la requête

```
<jsp:setProperty name="référence" property="*" />
```

**Les noms des paramètres de requête doivent être identiques aux noms des propriétés**



Efficacité : modification des propriétés

# Java Beans et JSP : lecture et écriture propriétés

## ➤ Exemple : caractéristique d'une voiture (simplifiée)

```
<%@ page language="java" contentType="text/html" %>
<jsp:useBean id="ma_voiture" class="Voiture"></jsp:useBean>

<%
    ma_voiture.setDemarree(true);
    ma_voiture.setVitesse(21.2);
%>

<html>
<head>
<title>Page pour lecture d'information</title>
</head>
<body bgcolor="white">
    La voiture a-t-elle démarré: <%= ma_voiture.getDemarree() %><br>
    La vitesse de la voiture est de : <jsp:getProperty
        name="ma_voiture" property="vitesse" /> km/h<br>
    La puissance de la voiture est de : <jsp:getProperty
        name="ma_voiture" property="puissance" /> CV
</body>
</html>
```

La classe Voiture  
doit être placée  
dans le répertoire  
WEB-INF\classes

Modification des  
propriétés du Bean  
*Voiture* par Scriptlet

Interrogation par  
tag Expression

Interrogation par tag Action.  
Le nom de la propriété en  
minuscule



# Java Beans et JSP : scope

## ➤ Exemple : affectation et récupération des valeurs d'un Bean

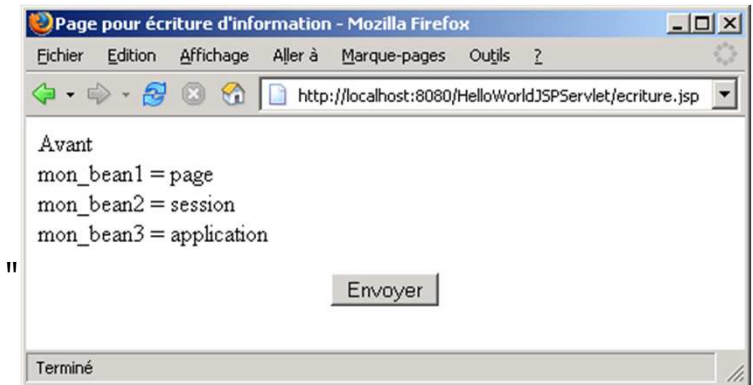
```
<%@ page language="java" contentType="text/html" %>

<jsp:useBean id="mon_bean1" scope="page" class="SimpleName"></jsp:useBean>
<jsp:useBean id="mon_bean2" scope="session" class="SimpleName"></jsp:useBean>
<jsp:useBean id="mon_bean3" scope="application" class="SimpleName"></jsp:useBean>

<jsp:setProperty name="mon_bean1" property="name" value="page"/>
<jsp:setProperty name="mon_bean2" property="name" value="session"/>
<jsp:setProperty name="mon_bean3" property="name" value="application"/>

<html>
<head><title>Page pour écriture d'information</title></head>
<body bgcolor="white">
Avant<br>
    mon_bean1 = <%= mon_bean1.getName() %><br>
    mon_bean2 = <%= mon_bean2.getName() %><br>
    mon_bean3 = <%= mon_bean3.getName() %><br>
    <FORM METHOD=GET ACTION="lecture.jsp">
        <p align="center"><input type="submit" name="
    </FORM>
</body>
</html>
```

Champs d'utilisation de l'objet Java Bean



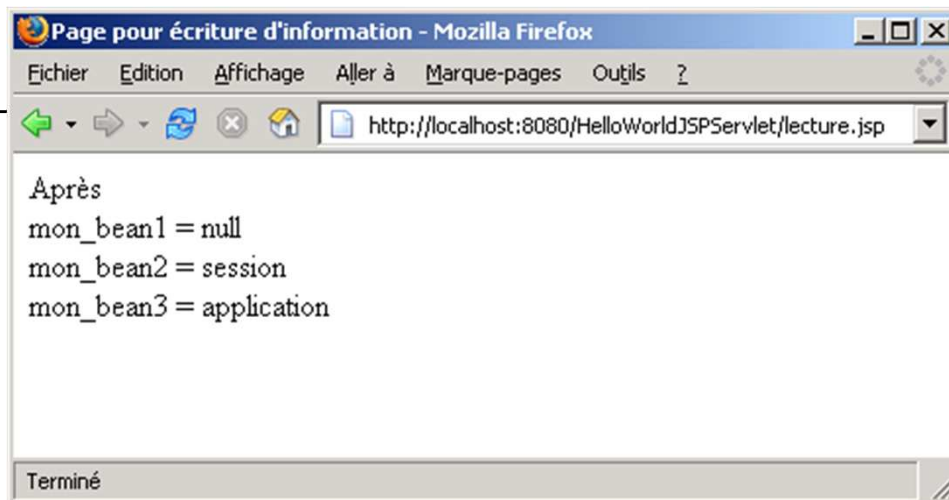
# Java Beans et JSP : scope

## ➤ Exemple : affectation et récupération des valeurs d'un Bean

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

<jsp:useBean id="mon_bean1" scope="page" class="SimpleName"></jsp:useBean>
<jsp:useBean id="mon_bean2" scope="session" class="SimpleName"></jsp:useBean>
<jsp:useBean id="mon_bean3" scope="application" class="SimpleName"></jsp:useBean>

<html>
<head>
<title>Page pour écriture d'information</title>
</head>
<body bgcolor="white">
Après<br>
mon_bean1 = <jsp:getProperty name="mon_bean1" property="name" /><br>
mon_bean2 = <jsp:getProperty name="mon_bean2" property="name" /><br>
mon_bean3 = <jsp:getProperty name="mon_bean3" property="name" /><br>
</body>
</html>
```



# Collaboration de JSP

---

- Rappel sur la collaboration (voir partie Servlet)
  - *partage d'information* : un état ou une ressource
  - *partage du contrôle* : une requête
- Processus identique à la collaboration de Servlet pour le partage d'information et de contrôle
- Partage d'information
  - Utilisation du contexte pour transmettre des attributs
  - Méthode *getContext(...)*, *setAttribute(...)* et *getAttribute(...)*
- Partage du contrôle
  - Utilisation des tags action JSP *include* et *forward*

# Partage d'information

## ➤ Exemple : transmettre un simple attribut à tout un contexte

```
Enregistrement dans le contexte d'un attribut  
<% application.setAttribute("attribut1", "Bonjour tout le monde"); %>
```

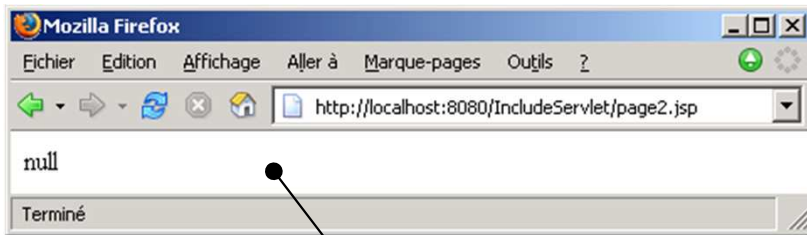
*page1.jsp*

Objet implicite  
*application* est de type *ServletContext*

Enregistrement dans le contexte  
courant d'une valeur pour l'attribut  
« attribut1 »

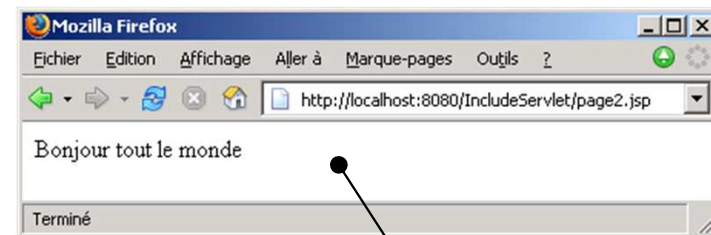
```
<% out.println(application.getAttribute("attribut1")); %>
```

*page2.jsp*



Résultat d'une requête à la *page2.jsp*  
sans envoi de requête à *page1.jsp*

Lecture dans le contexte courant de  
la valeur de l'attribut « attribut1 »



Résultat d'une requête à la *page2.jsp*  
après envoi de requête à *page1.jsp*

**Peut-être vue comme une variable  
globale à l'application WEB**



# Partage du contrôle : distribuer une inclusion

- Cette inclusion se fait au *moment de la requête*

La page incluse peut-être un fichier HTML ou JSP

```
<jsp:include page="/page.html" />
```

- La racine du chemin de la page est celle du contexte
- Le serveur exécute la ressource dynamique indiquée et inclut sa sortie au contenu envoyé au client
- Ne peut définir ni le code d'état ni la valeur des en-têtes
- Possibilité de transmettre des informations lors de l'inclusion

```
<jsp:include page="/page.jsp" />  
    <jsp:param name="defaultparam" value="nouvelle" />  
</jsp:include>
```



**La transmission d'informations autre que des simples chaînes de caractères ne peut être réalisée directement**

Définition d'un paramètre appelé *defaultparam*

# Partage du contrôle : distribuer un renvoi

- Cette redirection se fait également au *moment de la requête*

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

Le racine du fichier doit être ici à celle du contexte (équivalent à /page.html)

- Les choses à ne pas faire ...

- effectuer des modifications sur la réponse avant un renvoi (ignorées)
- placer du code en séquence à la suite d'un renvoi

- Possibilité de transmettre des informations lors d'un renvoi

```
<jsp:forward page="/page.jsp" />
  <jsp:param name="defaultparam" value="nouvelle" />
</jsp:forward>
```

Définition d'un paramètre appelé *defaultparam*

## Partage du contrôle : remarques ...

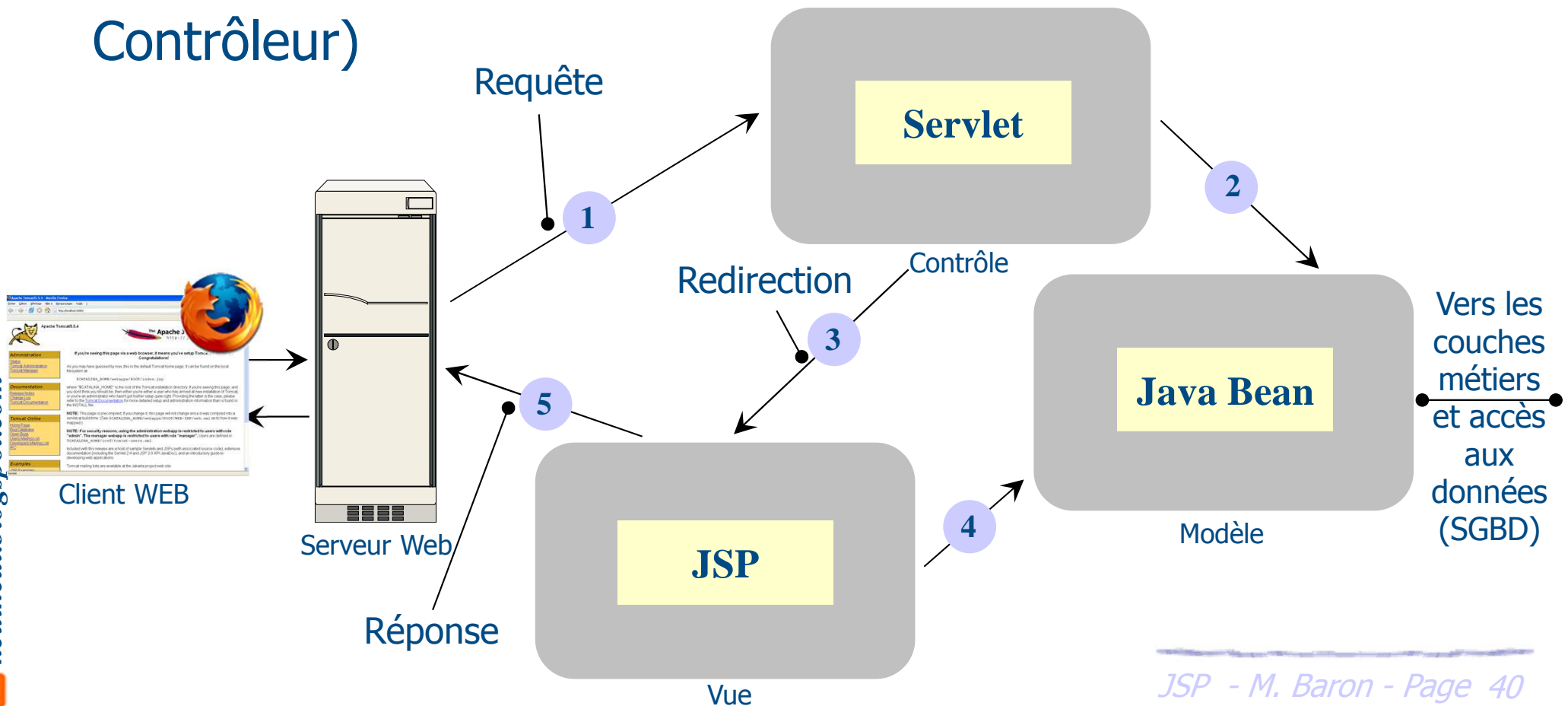
- La partage du contrôle et plus précisément l'inclusion et le renvoi par des balises actions ne permettent pas le transfert d'attributs au sens objet (uniquement des chaînes de caractères)
- Obligation de passer par un objet *RequestDispatcher* et l'objet implicite *request*
- Exemple pour une inclusion (même chose pour un renvoi)

```
<% RequestDispatcher dispatch = request.getRequestDispatcher("/fichier.jsp");%>
<% request.setAttribute("attribut1", "bonjour"); %>
<% dispatch.include(request, response); %>
```

Objet implicite

# Collaboration de Servlets et JSP : architecture MVC

- Un constat : les pages JSP doivent contenir le moins de Java
- Besoin de structurer le code plus finement selon les compétences de chaque technologies (JSP, Servlet, ...)
- Utilisation du modèle d'architecture : MVC (Modèle Vue Contrôleur)





# Collaboration de Servlets et JSP : architecture MVC

---

- Architecture MVC (en règle générale)
  - Le contrôle est géré par les Servlets (pas de sortie dans la réponse)
  - Le modèle est géré par les Beans (ne connaît pas le contrôle ni la vue)
  - La vue est gérée par les JSP (pas de contrôle)
- Besoin de savoir communiquer entre des Servlets et des JSP
  - Requête envoyée par le client (1)
  - Communication entre le contrôle et le modèle (2)
  - Transmission à la page JSP des informations du Java Bean (3)
  - La page JSP utilise les informations du Java Bean traduit (4)
  - La vue est envoyée en réponse au client (5)

# Collaboration de Servlets et JSP : architecture MVC

## ➤ Dans la Servlet

Instance du Bean  
envoyée comme attribut

Ressource à rediriger

```
bean = new MonBean(...);
req.setAttribute("idbean", bean);
RequestDispatcher dispatch = this.getServletContext().getRequestDispatcher("/page.jsp");
dispatcher.forward(req, res);
```

## ➤ Dans la JSP

Même nom que l'attribut  
donné dans la Servlet

```
<jsp:useBean id="idbean" class="monpackage.MonBean" scope="request"/>
<%= idbean... %>
```

ou

```
<% monpackage.MonBean idbean = (monpackage.MonBean)request.getAttribute("idbean"); %>
<%= idbean... %>
```

```
<%@ page import="monpackage.MonBean" %>
<% MonBean idbean = (MonBean)request.getAttribute("idbean"); %>
```

Pour éviter de mettre les  
noms de package  
utilisation de la directive  
*page import*

# Collaboration de Servlets et JSP : architecture MVC

## ➤ Exemple : application MVC qui dit « Bonjour » ...

```
...
import monpackage.MonBean;

public class Controle extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        MonBean bean = new MonBean();
        req.setAttribute("idbean", bean);
        RequestDispatcher disp = this.getServletContext().getRequestDispatcher("/page.jsp");
        disp.forward(req, res);
    }
}
```

Cette Servlet gère  
le contrôle. Il  
dispatche à la vue

```
package monpackage;

public class MonBean {
    public String getMessage() {
        return "Message issu du Bean";
    }
}
```

**En générale pas de  
sortie sur la réponse  
de la Servlet**



Ce Java Bean gère  
le modèle de  
l'application WEB

# Collaboration de Servlets et JSP : architecture MVC

## ➤ Exemple (suite) : application MVC qui dit « Bonjour » ...

```
<HTML>
<jsp:useBean id="idbean" class="monpackage.MonBean" scope="request"/>
<HEAD><TITLE>Architecture MVC : Java EE</TITLE></HEAD>
<BODY>
<%= idbean.getMessage() %>
</BODY>
</HTML>
```

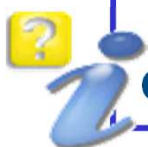
1<sup>ère</sup> solution en  
utilisant la tag  
action *useBean*

**Peu de contrôle dans  
les pages JSP ...**



```
<%@ page import="monpackage.MonBean" %>
<HTML>
<HEAD><TITLE>Architecture MVC : Java EE</TITLE></HEAD>
<BODY>
<% MonBean idbean = (MonBean)request.getAttribute("idbean");%>
<%= idbean.getMessage() %>
</BODY>
</HTML>
```

2<sup>ème</sup> solution en  
utilisant l'objet  
implicite *request*



**Vos Beans doivent être définis  
obligatoirement dans un package**