Chapitre 4
Composants Web JSP

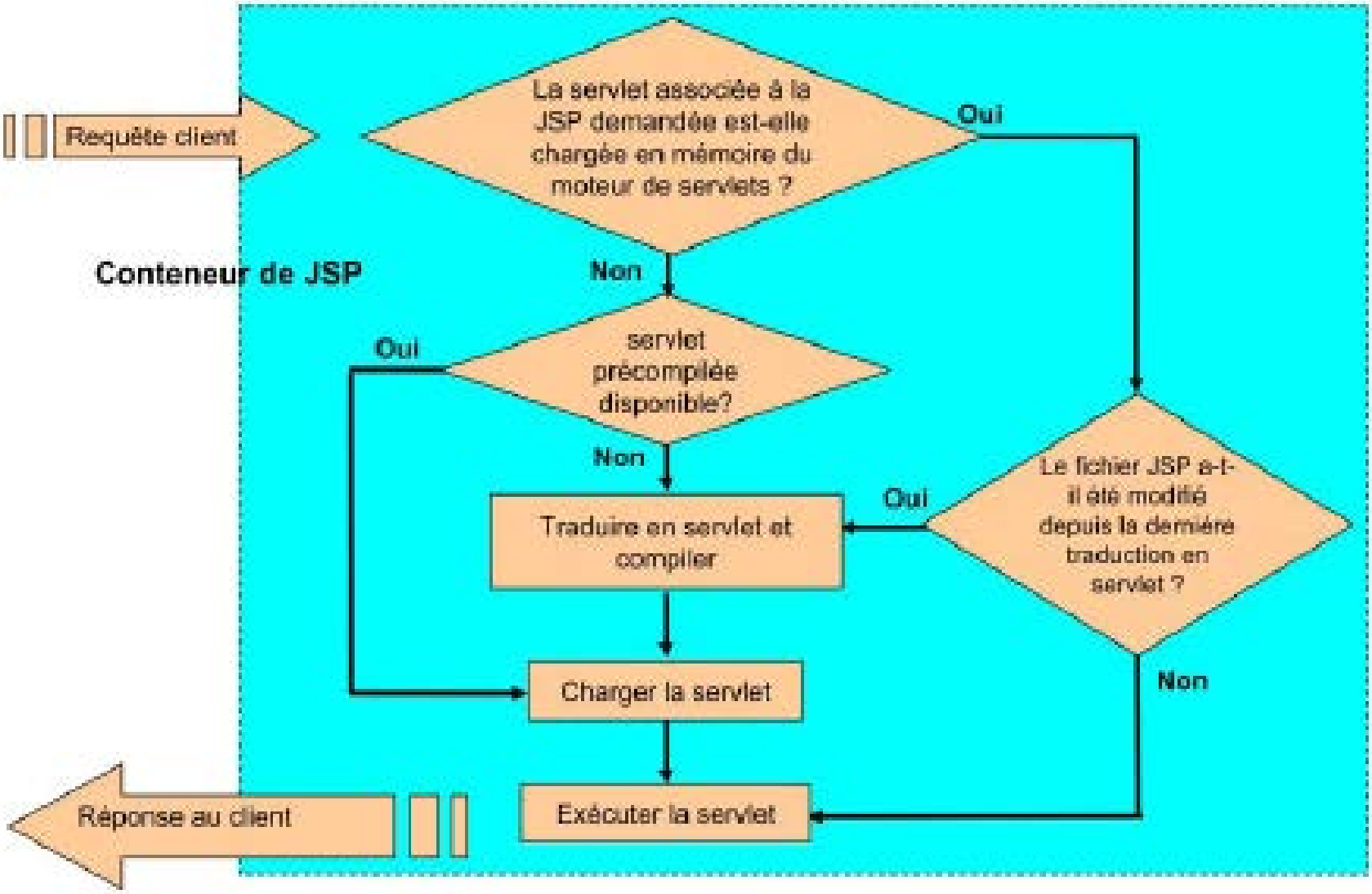
I. Definition

- Une **JSP** est un fichier contenant du code HTML et des fragments de code Java exécutés sur le moteur de Servlets et le contenu statique est servi comme du HTML normal. Elle est Comparable aux langages côtés serveur de type PHP, ASP, …
- **Servlet :** du code Java contenant du code HTML
- **JSP : ** une page HTML contenant du code Java
  - programme source Java embarqué dans une page .html
- 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

	côté client	côté serveur
.class autonome	applet	servlet
embarqué dans .html	JavaScript	JSP

II. Principe de Fonctionnement

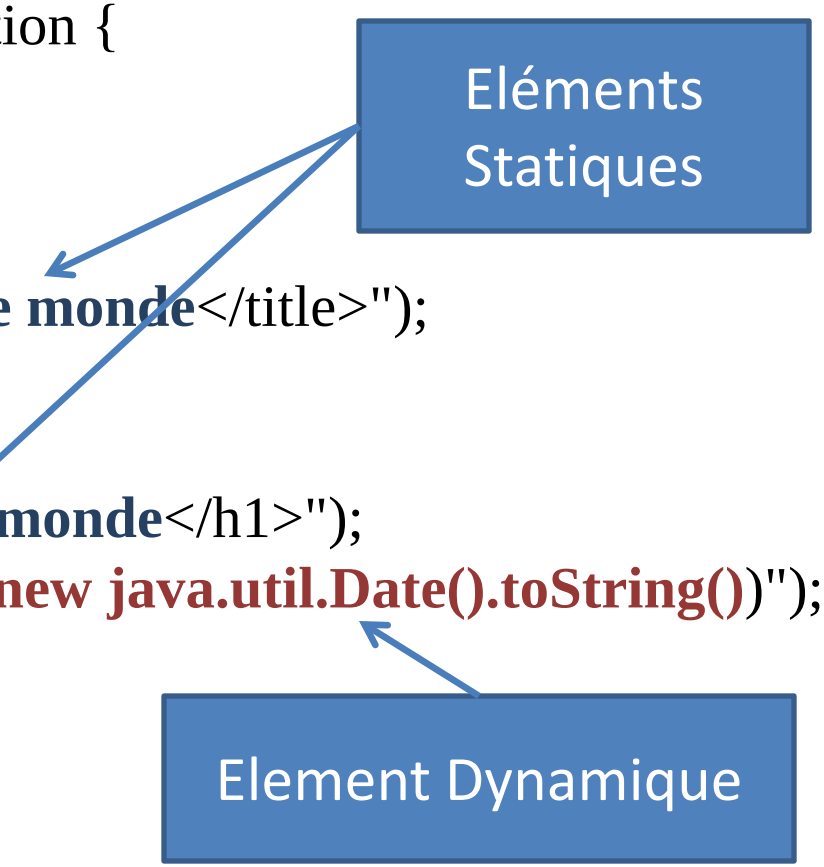
- Toute la page **JSP** est convertie en **une servlet**.
- Cette servlet est traitée par le moteur Java intégré au serveur Web (technologie des servlets) qui retourne la page HTML construite.


Requête client
La servlet associée à la JSP demandée est-elle chargée en mémoire du moteur de servlets ?
Oui
Conteneur de JSP
Non
Oui
servlet précompilée disponible?
Non
Le fichier JSP a-t-il été modifié depuis la dernière traduction en servlet ?
Oui
Traduire en servlet et compiler
Non
Charger la servlet
Réponse au client
Exécuter la servlet

Exemple : Helloword

version Servlet

```
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> <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())");
    out.println(" </body>");
    out.println("</html>");
    }
}
```



Exemple : HelloWorld

Version JSP

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

Remarques :

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

Sevlet générée par le conteneur de 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("\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);
}
}
}
```

IV. Balises JSP

Les balises (**Tags**) permettent de différencier le code HTML du code Java.

Nous avons 3 types :

- **des directives :** instructions qui contrôlent le comportement du compilateur de pages JSP, et qui sont donc évaluées avant la compilation.

- **des éléments de scripts (scriptlets)** : blocs d'instructions Java inclus dans la page JSP entre des délimiteurs **<% et %>.**

- **des balises personnalisées (custom tags ou custom actions)**: elles sont définies par le programmeur et génèrent du contenu dynamique lorsque la page est demandée.

IV.1. Directives JSP

- Les directives JSP affectent la structure globale de la servlet générée à partir de la page JSP.
- Elles 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
- ✓ **taglib** : indique une bibliothèque de balises à utiliser
- ✓ **page** : définit les attributs spécifiques à une page

a. Directive include

Pour insérer un fichier dans un document JSP principal au moment où le document est traduit en servlet, on utilise la balise :

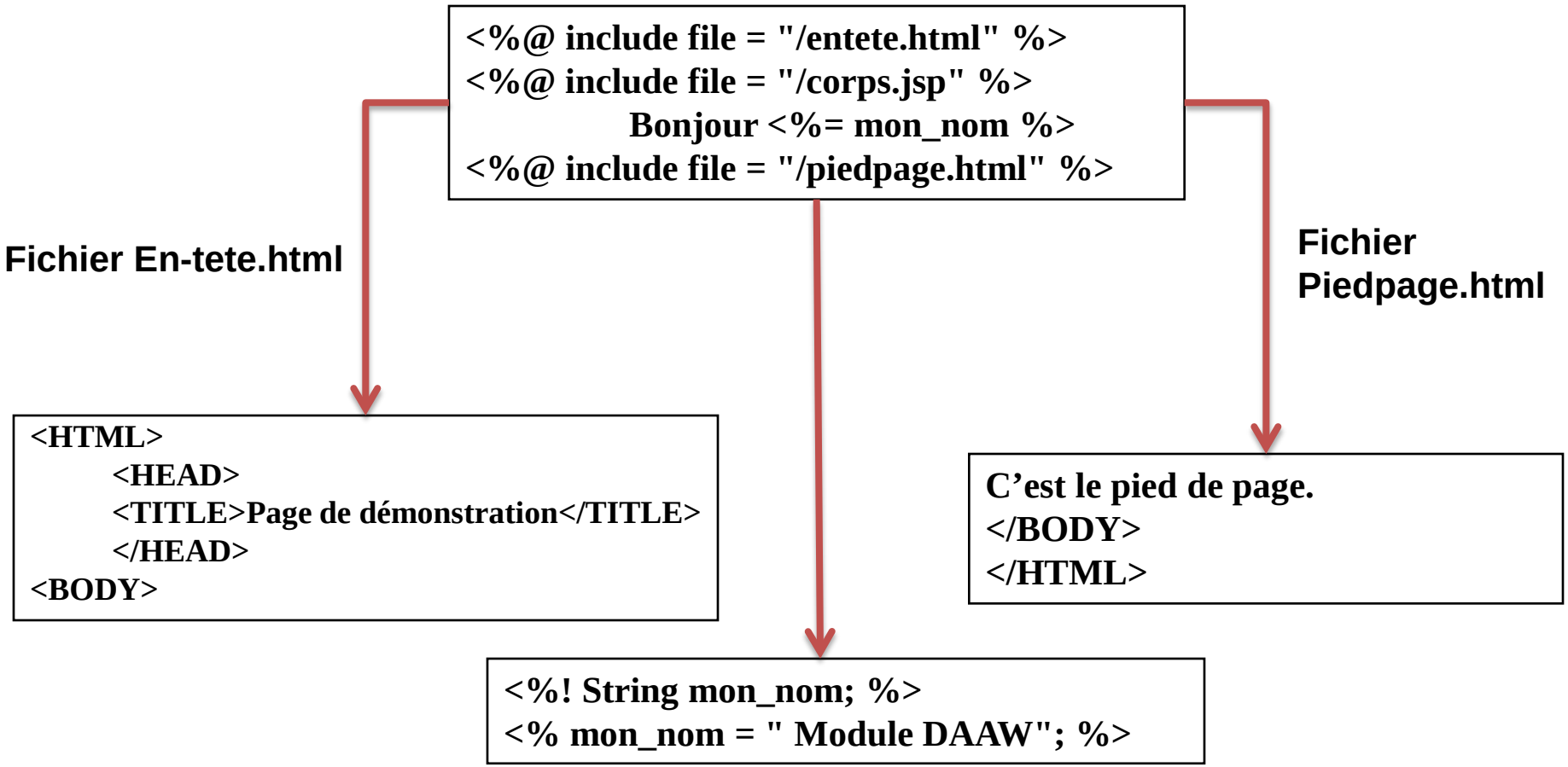
<%@ include file= "URL relative du fichier à inclure" %>

- La directive include correspond à l'inclusion statique d'un fichier JSP, avant la compilation.
- Une seule servlet est générée

Remarques :

- 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 de la page JSP principale.
- Le fichier inclus peut contenir du code JSP et non uniquement du code HTML statique.
- La page incluse peut accéder aux variables de la page principale est modifier cette page principale.

Exemple d'inclusion par la directive include



Jsp:include

- Des fichiers peuvent être inclus au moment où le client demande la page en utilisant **jsp:include**. Dans ce cas, le contenu du fichier inclus n'est analysé par la page JSP qu'au moment de l'exécution.
- Cette balise correspond à l'inclusion dynamique, au moment où la page principale est appelée.
- L'inclusion est sans impact sur la page principale
- La page principale n'est pas modifiée par l'inclusion
- Deux servlets sont générées.

b. Directive page

- La directive **page** permet de contrôler la structure de la servlet.
- Elle peut être placée n'importe où dans la page JSP.
- Elle définit les attributs spécifiques à une page :

- **import** : permet de spécifier des paquetages Java à importer par la servlet générée à partir de la page JSP. Cette directive résulte en une instruction *import* dans la Servlet

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

  - ❑ L'attribut import est le seul attribut qui peut apparaitre plusieurs fois dans une page JSP.
- **langage** : définit le langage de script utilisé dans la page. Pour l'instant, Java est à la fois le langage par défaut et le seul langage disponible.

b. Directive page(Suite)

- **contentType** : définit l'entête de réponse content-Type, c.à.d., 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="erreur.jsp" %>
```

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

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

IV.2.Éléments de Script JSP

a. Les Commentaires

- 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 --%>

Exemple :

```
<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>
```

Élément de Script JSP : Déclaration

- Une déclaration JSP permet de définir des méthodes ou des champs qui seront insérés dans le corps de la servlet générée (en dehors de **_jspService**).

- Les déclarations sont initialisées lors de l'initialisation de la page JSP et possèdent une portée de classe.

- Une déclaration est placée entre les symboles **<%!** et **%>**
- Elle peut être utilisée pour :
  - Déclarer un attribut de classe
  - Spécifier et implémenter des méthodes

- La balise <%! ……%>
  - Permet de déclarer une variable.

Exemple 1:

```
<%! Int i; %>
<%! Int i = 0;%>
<%! Float f, g = 1.2, h; %>
```

Exemple 2 :

```
<%!
private int count = 0; // declaration de variable
private int incrementCount() { return count++;} // Déclaration de méthode
%>
```

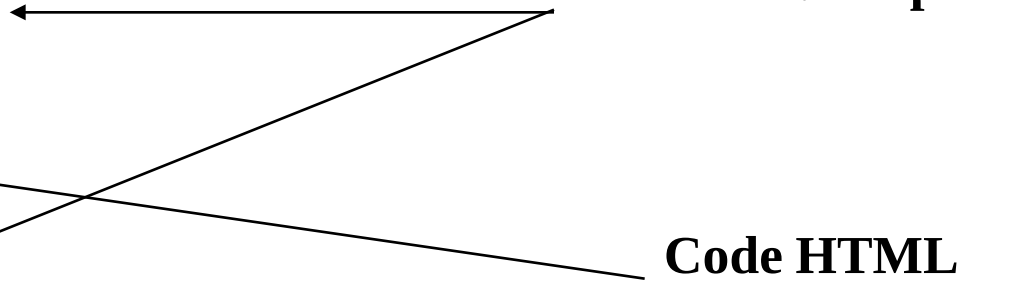
- **Remarque:** l'identificateur d'une variable doit respecter les règles des variables dans JAVA

Élément de Script JSP : Scriptlets

- Les scriptlets permettent d'insérer n'importe quel code java dans la méthode **_jspService** de la servlet générée.
- Les scriplets ont la forme **<% java code %>**
- Tout code java inséré dans un scriptlet a accès :
  - aux attributs et méthodes définis par la balise déclaration **<%! … %>**
  - aux objets implicites

Exemple

```
….
<%
    For (int i=0 ; i < 5 ; i++) {
%>
    HelloWord<br>
<%
    incrementCount() ;  }
%>
…
```



- **incrementCount()** est la Méthode déclarée par l'élément de script **déclaration** précédent

Élément de Script JSP : Expression

- Correspond à un scriptlet qui envoie une valeur dans le flux de réponse vers le client.
- L'expression est évaluée, convertie en une chaîne, puis insérée dans la page html générée par la servlet.
- Cette évaluation est effectuée au moment de l'exécution.
- Les expressions sont placées entre les symboles **<%=** et **%>**
- Remarques :
  - Retourne une valeur String de l'expression
  - Correspond à un scriplet de la forme

    *<% **out.println(…);** %>*
  - Elle se transforme en ***out.println("…");*** dans la méthode *_jspService(…)* après génération de la servlet.
  - Ne pas ajouter de « **;** » à la fin d'un élément expression.

Exemple de JSP

```
<%! int compteur =0; %>
<HTML>
<BODY>
    Bonjour. <BR>
    <%
    compteur ++;
    if (compteur == 1)
    %>
    Vous etes le premier visiteur.
    <% else %>
    Vous êtes le <%=compteur %> ème visiteur.
</BODY>
</HTML>
```

Déclaration

Scriptlet

Expression

Element de Script JSP : Scriptlet et Objets implicites

- Les objets implicites sont des objets qui peuvent être utilisés immédiatement dans une expression ou un scriptlet (**les objets employés dans la méthode *service(…)* dans la partie servlet**)
- Les objets suivants peuvent être utilisés :
  - **request** : l'objet **HttpServletRequest** associé à la requête courante. Il permet d'accéder aux paramètres de la requête.
  - **response** : est l'objet **HttpServletResponse** envoyé avec la réponse au client.
  - **session** : session courante
  - **out** : le canal de sortie. C'est le **PrintWriter** utilisé pour envoyer la sortie au client.
  - **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

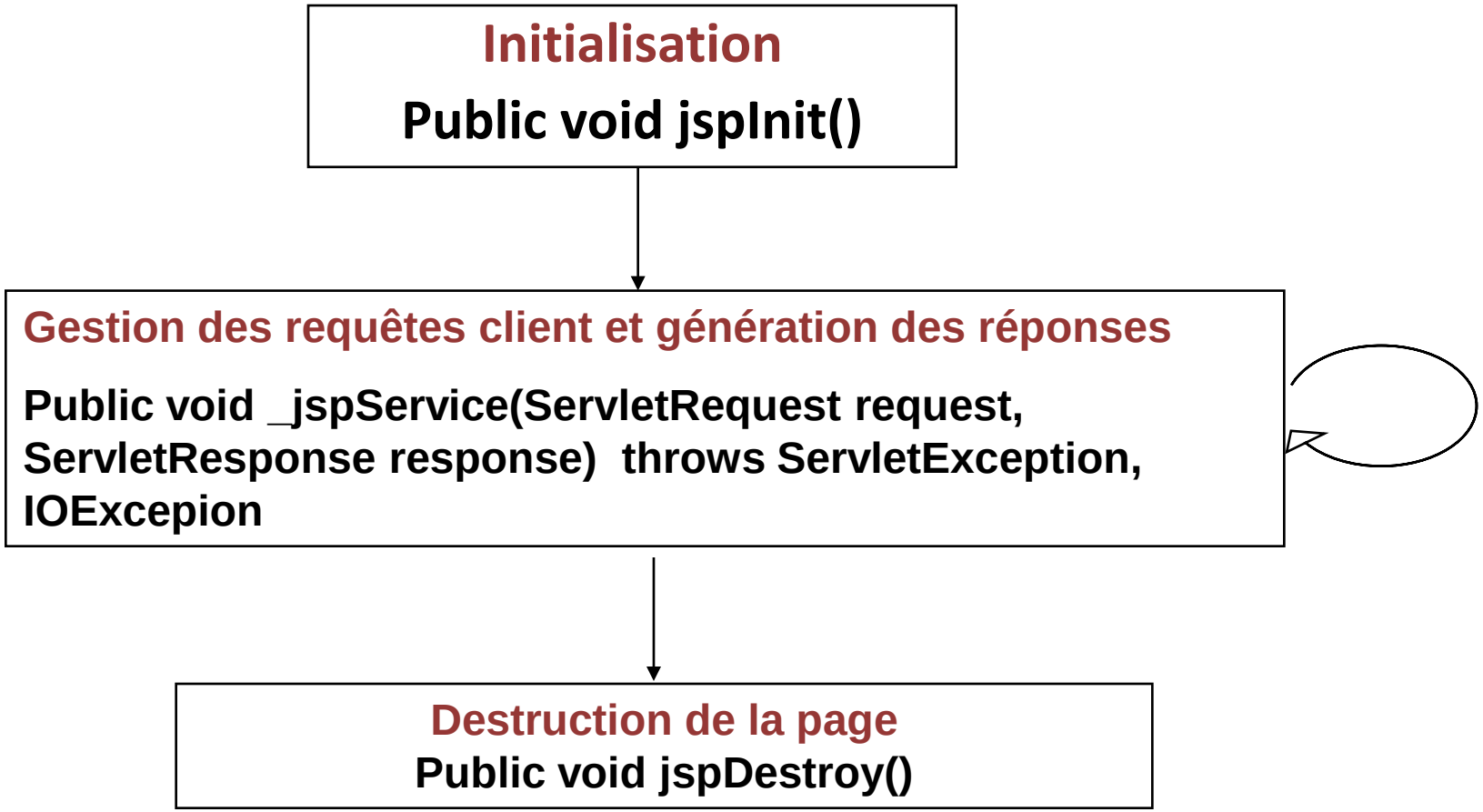
Exemple : JSP récupèrant des informations du client

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

V. Cycle de vie d'une JSP

- Le cycle de vie d'une JSP est identique à une Servlet :
  - La méthode ***jspInit()*** est appelée après le chargement de la page. Elle est similaire à la méthode init() des servlets. Cette méthode peut être implémentée par le concepteur de la page JSP.
  - La méthode **_jspService()** est appelée à chaque requête
  - La méthode ***jspDestroy()*** est appelé lorsque la page JSP va être détruite par le conteneur (fermeture d'une base de données ou arrêt du serveur)
- Ainsi le cycle de vie d'une page JSP est :

Cycle de vie d'une JSP

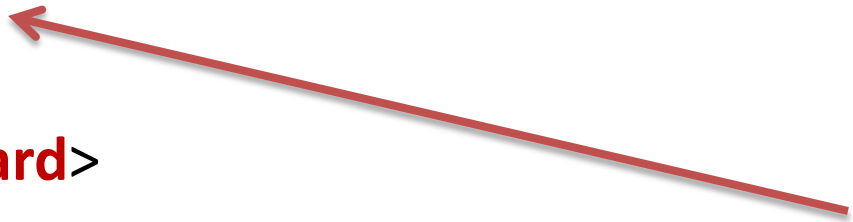


Enchaîner les pages

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

```
<jsp:forward    page="pageDeRedirection" >
    <jsp:param  name= " NomParam"  value = "valeur " />

</jsp:forward>
```



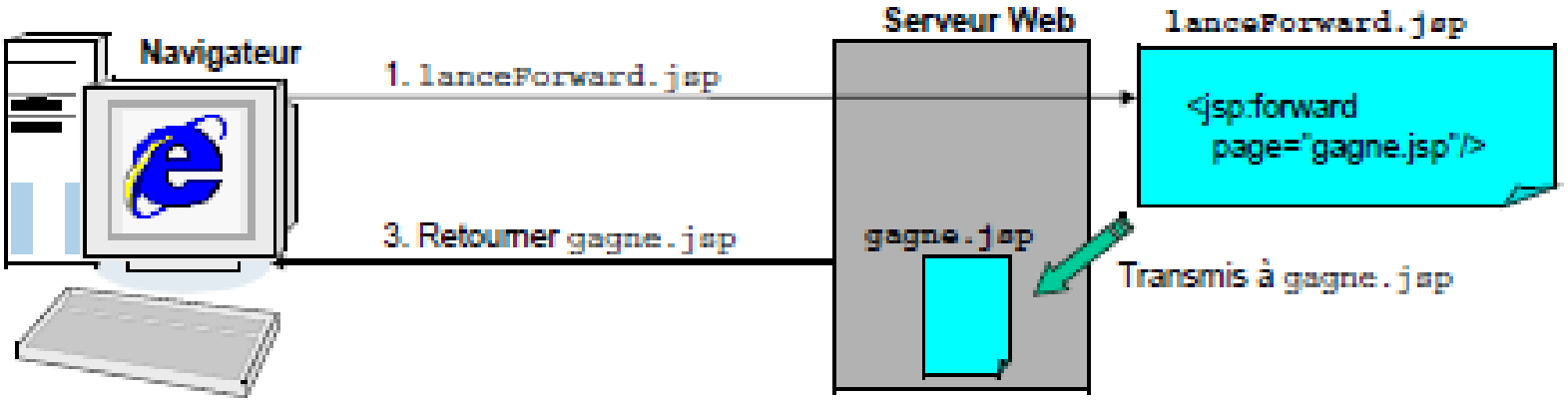
Passer des paramètres à la page appelée

Exemple

```
<% 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"/>
<% } %>
```

Remarque : On n'affiche jamais la partie qui suit **jsp:forward**

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



Jsp et bean

- Un bean est un simple objet java respectant certaines contraintes.
- Il représente généralement des données du monde réel.

Structure

- Doit être une **classe publique**;
- Doit avoir au moins **un constructeur par défaut**, public et sans paramètres;
- Peut implémenter l'interface **Serializable**, il devient ainsi persistant et son état peut être sauvegarder;
- Ne doit pas avoir de champs publiques;
- Des propriétés accessibles au travers de **setters et des getters** : getProp (lecture) et setProp (écriture) portant le nom de la propriété.

jsp:useBean

- Permet de séparer la partie traitement de la partie présentation
- Permet d'instancier un composant JavaBean (classe java) qui pourra être appelé dans la page JSP

<**jsp:useBean** **id**= " Nombean " **class** = " NomClassebean" **scope**="request" />

Ou bien

<**jsp:useBean** **id**= " Nombean " **type** = " NomClassebean" **scope**="request" />

Remarque: on utilise l'attribut class si on a le chemin du fichier (.class) et on utilise type si on a le chemin du fichier (.java)

Jsp et bean

Règles générales pour les noms d'attributs :

- Les setters et les getters des propriétés d'un bean sont construits par la concaténation du mot **get** / **set** suivi du nom de la propriété.
- Le nom de la propriété commence par minuscule ➔ le setter/getter commence par majuscule:

```
String lastName;
String getLastName();
```

Jsp et bean

Exceptions:

- Si le getter retourne un boolean alors on utilise **is** au lieu de **get**
- Nom de la méthode: getPrime ou isPrime
- Nom de la propriété : prime

Création d'instance d'un Bean

Format

<Jsp:useBean id = " idbean " type = "Nomclassbean"
"scope= "porté" >

Objectif

- Permet l'instantiation d'une classe java dans une page jsp sans utiliser le code java si le bean n'a pas encore été déjà crée.
- Permet de récupérer le bean s'il a déjà été déposé dans un objet implicite(Request, session ou Context)

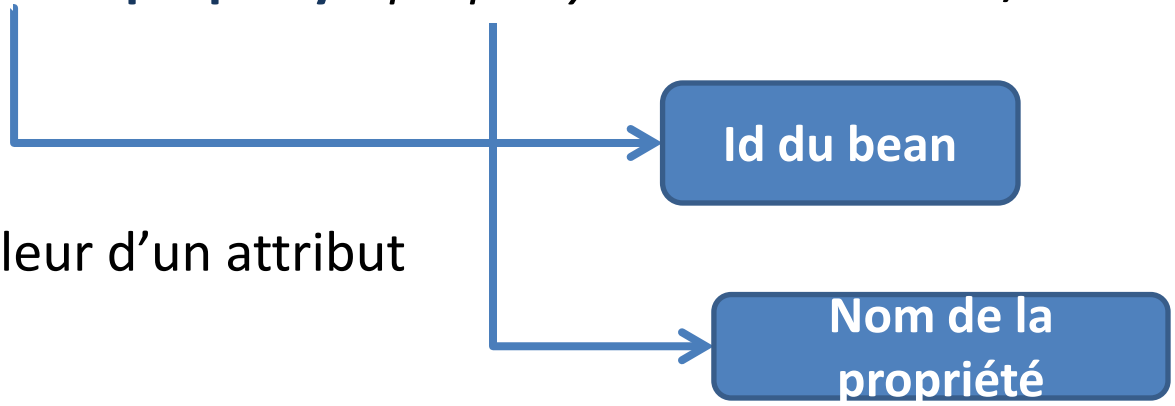
Avantages de jsp:useBean

- La balise **jsp:useBean** a deux autres avantages:
  - La récupération des valeurs de l'objet à partir des paramètres de requête est plus facile.
  - Le partage d'objets entre pages jsp et servlet est plus facile.

Modification de la valeur d'une propriété

Format :

<jsp:setProperty **name**="*name*" **property**="*property*" **value**="*value*" />



Role

Permet de modifier la valeur d'un attribut

Remarque

<jsp:setProperty **name**="livre1" **property**="title" **value**= "Informatique"/>

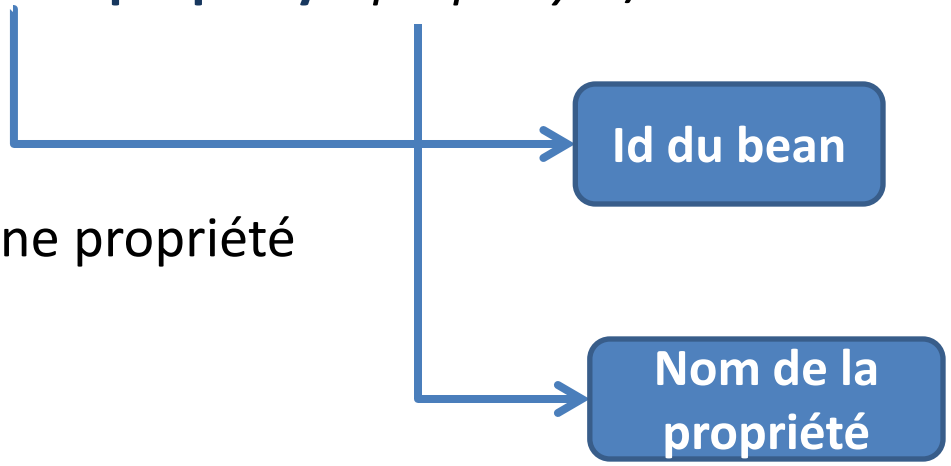
Equivalente au scriptlet

<% livre1.**setTitle** ("Informatique"); %>

Récupération de la valeur d'une propriété

Format

<jsp:getProperty **name**="*name*" **property**="*property*" />



Role

Permet de lire la valeur d'une propriété

Remarque

<jsp:getProperty **name**="livre1" **property**="title" />

Equivalente au scriptlet

<% =**livre1.getTitle()**; %>

Remarques

- Le bean est créé dans le répertoire **src** du projet eclipse
- Le bean est déployé dans **WEB-INF/classes**

(.class du bean)

- Les beans doivent toujours être définis dans des packages.

Exemple : Définition du Bean

```
package coreservlets;
public class StringBean {
private String message = “Aucun message";
public String getMessage() {
        return(message);
}
public void setMessage(String message) {
    this.message = message;
}
}
```

Exemple : Utilisation du bean

```
<jsp:useBean id="stringBean"  class="coreservlets.StringBean" />
<OL>
    <LI>Initial value (from jsp:getProperty):
    <I><jsp:getProperty name="stringBean"  property="message" /></I>
    <LI>Valeur Initiale (à partir d'une expression):
    <I><%= stringBean.getMessage() %></I>
    <LI><jsp:setProperty name="stringBean" property="message"
    value="Meilleur Bean: Fortex" />
    La valeur après jsp:setProperty
    <I><jsp:getProperty name="stringBean" property="message" /></I>
</OL>
```

Récupération à partir de la requête

Deux façons de faire

1. En utilisant la méthode getParameter de l'objet **Request**

```
<jsp:setProperty name="entry"
    property="itemID"
    value='<% Request.getParameter("itemID") %>' />
```

2. En utilisatant l'attribut **param** de la balise **jsp: setProperty**

```
<jsp:setProperty name="entry"
    property="itemID"
    param = "itemID "  />
```

Exemple Récupération à partir de la requête

```
<jsp:useBean id="entry"
             class="coreservlets.SaleEntry" />
<jsp:setProperty
    name="entry"
    property="itemID"
    param="itemID" />
<jsp:setProperty
    name="entry"
    property="numItems"
    param="numItems" />
<jsp:setProperty
    name="entry"
    property="discountCode"
    param="discountCode" />
```

Récupération de toutes les propriétés d'un bean aux paramètres de requête

- On utilise *

```
<jsp:useBean id="entry"
             class="coreservlets.SaleEntry" />
<jsp:setProperty name="entry" property="*" />
```

Chaque propriété du bean prend la valeur du paramètre de la requête qui a le même nom.

Partage de bean

- Nous pouvons définir la porté d'un bean à travers **scope**
- Les valeurs de scope sont :

page (<jsp:useBean ... scope="page"/> or <jsp:useBean...>)

- Page est la valeur par défaut de scope

application (<jsp:useBean ... scope="application"/>)

- Le bean est accessible par toutes jsp de l'application
- Il est sauvegarder dans servletContext
- Accessible dans les servlet à travers getServletContext()

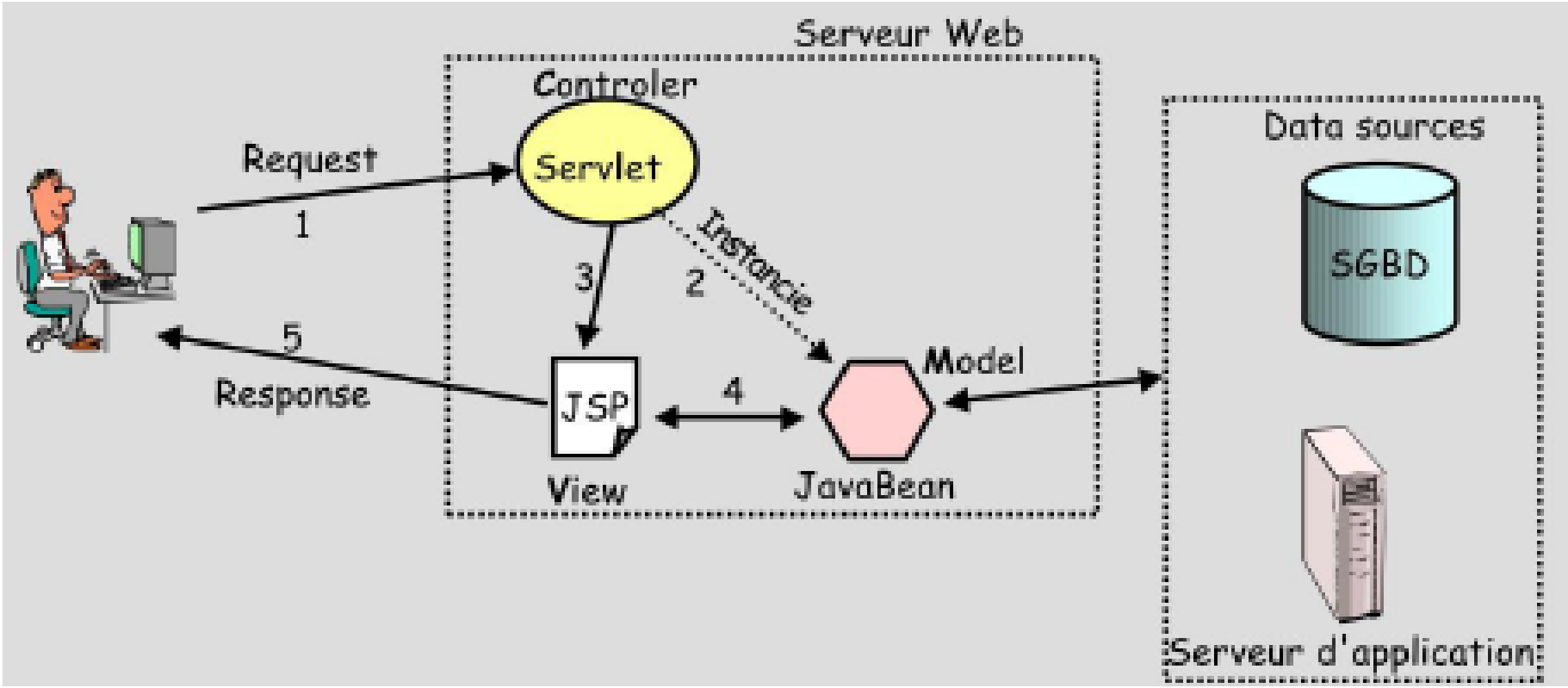
session
(<jsp:useBean … scope="session"/>)

- Le bean est enregistré dans l'objet **HttpSession** associé à la requête en cours
- L'accés au bean dans les servlets est réalisé à travers **getAttribute** et **setAttribute**

request
(<jsp:useBean … scope="request"/>)

- Le bean est accessible dans la servlet via l'objet **ServletRequest**

Architecture MVC



Architecture MVC : Principe de fonctionnement

1. Par le biais d'une page web, l'utilisateur émet une requête HTTP au serveur web en cliquant sur un lien ou sur un bouton. Cette requête est prise en charge par le contrôleur (servlet).
2. Le contrôleur exécute les traitements nécessaires (appelle un EJB Stateless par exemple) et récupère le modèle, c'est-à-dire les entity beans.
3. Le contrôleur sélectionne alors la JSP qui sera en charge de la construction de la réponse et lui transmet les entity beans contenant les données à afficher.
4. La JSP construit la réponse en faisant appel aux accesseurs des entity beans.
5. La réponse HTTP est transmise au navigateur qui l'affiche sous forme de page web.

- Syntaxe dans la servlet pour lancer la JSP

```
public void doPost(HttpServletRequest request, HttpServletResponse response){

ServletContext context = this.getServletContext();
RequestDispatcher dispatcher =
request.getRequestDispatcher("/maPageMiseEnForme.jsp");
dispatcher.forward(request, response);
}
```

Architecture MVC

- La servlet peut passer des valeurs à la JSP appelée grâce à **setAttribute()**

```
public void doPost(HttpServletRequest request, HttpServletResponse response) {
// appelle les méthodes sur les objets métiers
Livre livre = new Livre()// un objet à passer
// ajouté à la requête
request.setAttribute("livre",  livre);
RequestDispatcher dispatcher  =  request.getRequestDispatcher("/jsptest.jsp");
dispatcher.forward(request, response);
}
```

Architecture MVC

- La JSP extrait les objets de **Request** grâce à **getAttribute()**

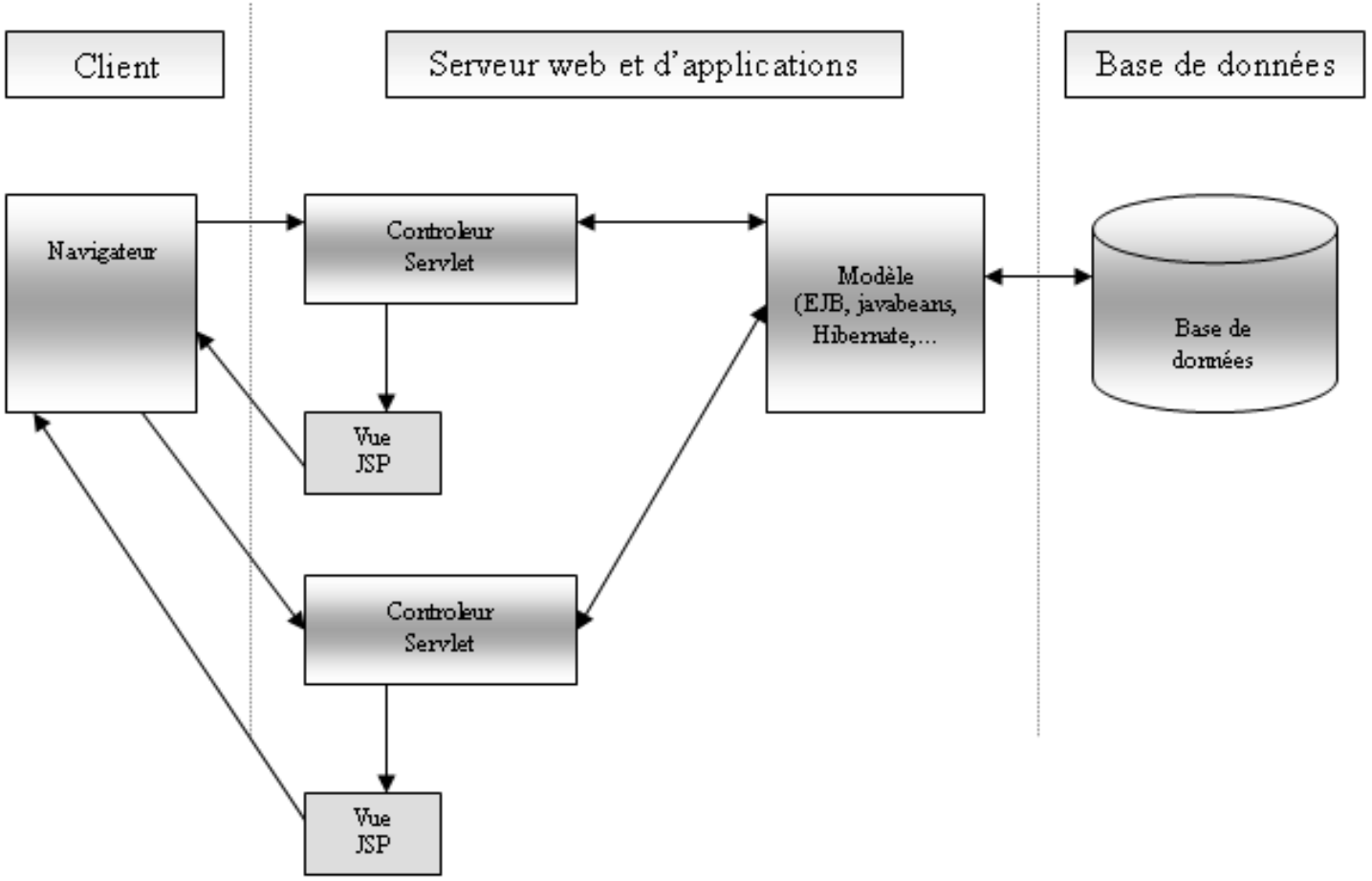
```
<% Livre   li = (Livre) request.getAttribute("livre ");
   // maintenant, utiliser l'objet
%>
```

Exemple

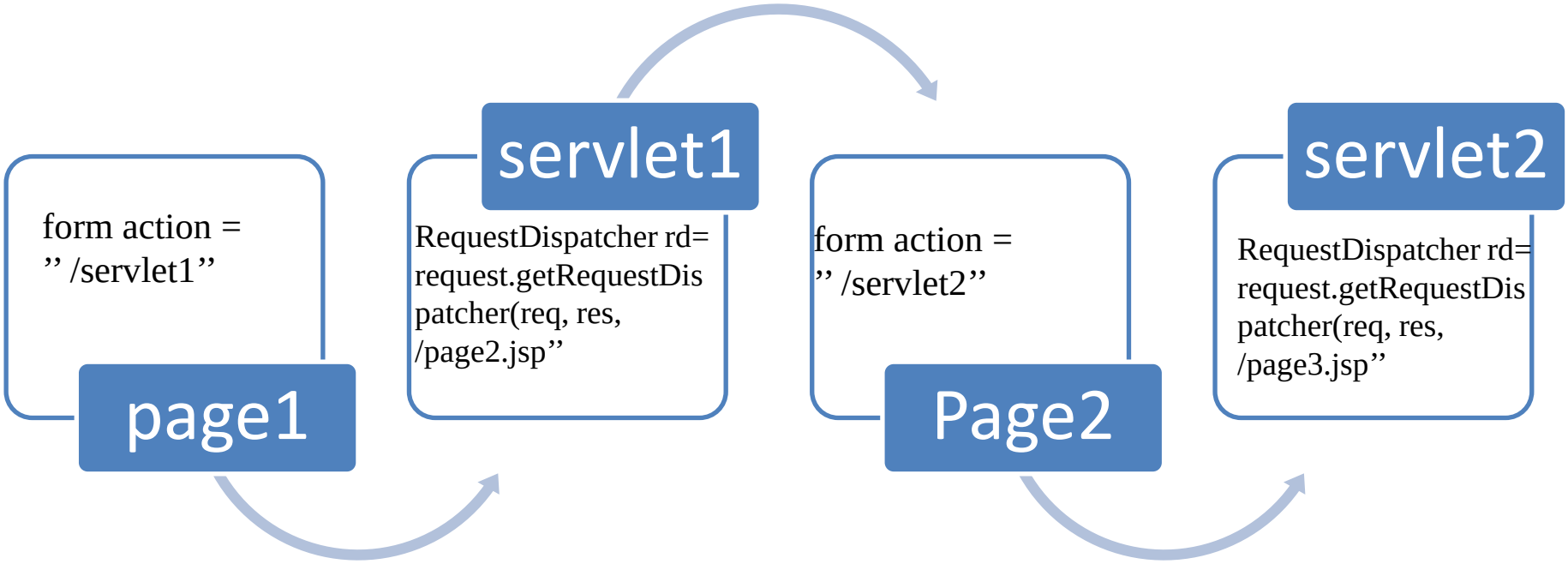
```
protected void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {
    String message = "Transmission de variables : OK" ;
    req.setAttribute("test", message);
    RequestDispatcher dispatcher =
    req.getRequestDispatcher("/WEB-INF/test.jsp");
    dispatcher.forward(req, res);
}
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<p> Ceci est une page générée depuis une JSP. />
<%
	String attrib=(String) request.getAttribute(‘’test ‘’);
	out.println(attribut);
%>
</body>
</html>
```

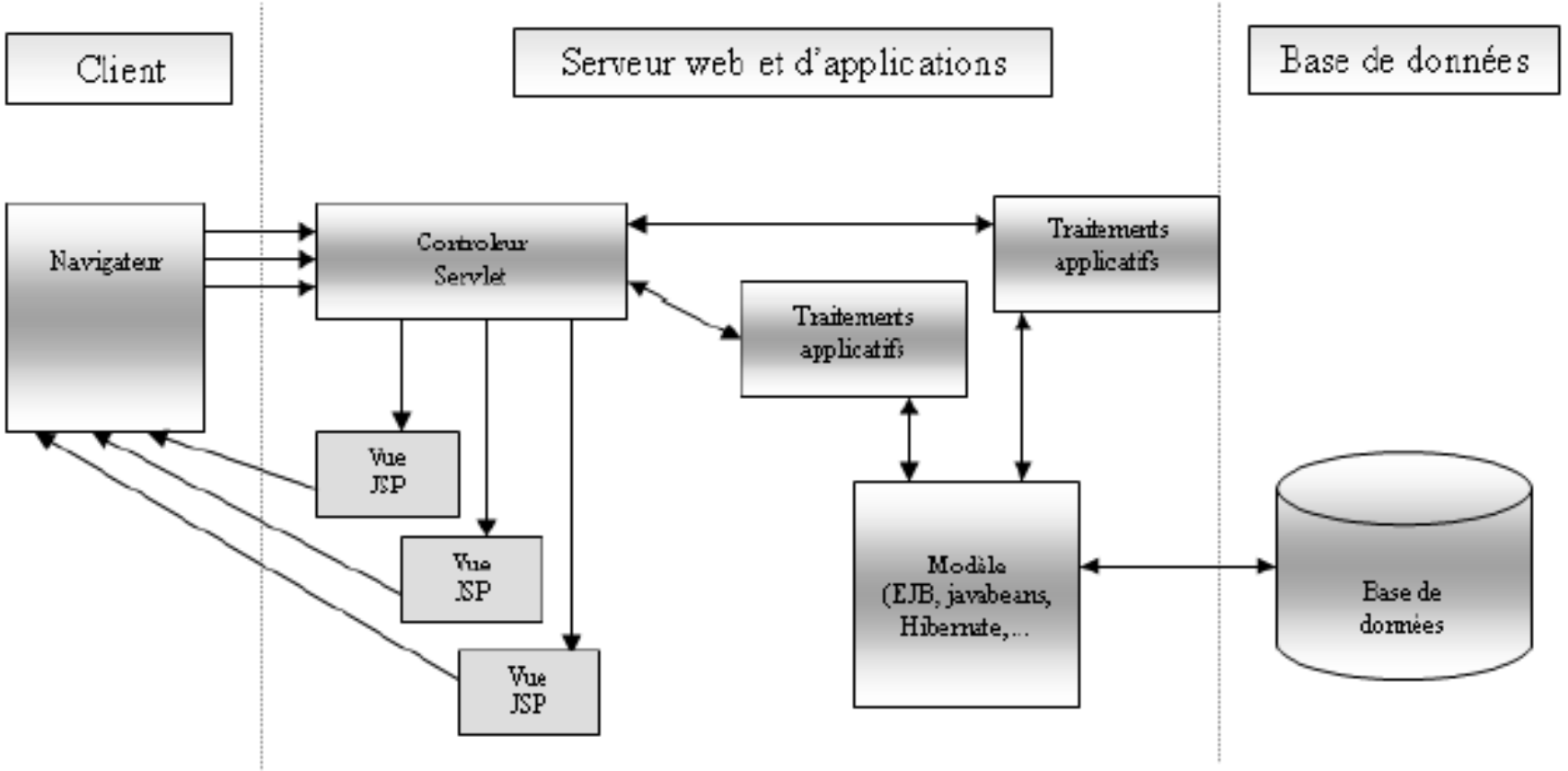
Implémentaion MVC1



- Dans ce cas, chaque page jsp lui est associée une servlet.
- Pour appeler la servlet à partir de la jsp, on spécifie l'URL de la servlet dans l'attribut action du formulaire de la jsp.
- La servlet utilise le dispatcher de la requête pour appeler une autre page jsp.
- Pour partager des données entre les différentes servlets, on utilise l'objet Context (**this.getServletContext()**).
- Ajouter un attribut à l'aide de **context.setAttribute()**
- Récupérer un attribut global à travers **context.getAttribute()**


form action =
'' /servlet1''
page1
servlet1
RequestDispatcher rd=
request.getRequestDis
patcher(req, res,
/page2.jsp''
form action =
'' /servlet2''
Page2
servlet2
RequestDispatcher rd=
request.getRequestDis
patcher(req, res,
/page3.jsp''

Implémentation MVC2



- Dans ce, une seule servlet est définie.
- Pour appeler la servlet à partir de la jsp, on spécifie l'URL de la servlet dans l'attribut action du formulaire de la jsp.
- La servlet utilise le dispatcher du context pour appeler une autre page jsp.
- Pour partager des données entre les différentes servlet, on utilise l'objet Context (**this.getServletContext()**).
- Ajouter un attribut à l'aide de **context.setAttribute()**
- Récupérer un attribut global à travers **context.getAttribute()**

Une jsp : MVC2

```
1  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
2  <title>Une jsp pour MVC2</title>
3  </head>
4  <body>
5  <h1><center>Opérations sur Compte Bancaire</center></h1>
6  <form action="controlleur" method="POST">
7      <input type="hidden" name="FormName" value="Authentification"/>
8      <p><label> Numero de Compte :</label> <input type="Number" name="Num" /> </p>
9      <p><label> Code  : </label> <input type="Number" name="Code" />  </p>
10     <center><input type="submit" value="Valider"/></center>
11 </form>
12 </body>
13 </html>
```

Le Controlleur MVC2

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    ServletContext context = this.getServletContext();
    String action = request.getParameter("FormName");
    String address="";
if(action.equals("action1")){
        // faire un travail
        address = "/WEB-INF/Page1.jsp";
}

else
        if(action.equals("action2")){
            // faire un traitement
            address= "/WEB-INF/Page2.jsp";
        }
        else
            if(action.equals("action3")){
                //faire un tratement
                address = "/WEB-INF/page3.jsp";

            }
    RequestDispatcher dispatcher = context.getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
}
```