



Tehnologii Java

Curs -

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică


Universitatea "Al. I. Cuza" Iași



Java Server Pages

Cuprins



- Ce sunt paginile JSP ?
 - JSP vs. alte tehnologii
 - Ciclul de viață al unei pagini
 - Sintaxa JSP
 - Obiecte implicite
 - Tratarea cererilor concurente
 - Tratarea excepțiilor
 - Acțiuni standard
 - Modele de acces
- 

Introducere

Contextul



Nivelul de prezentare din cadrul nivelului Web:

- interfața grafică cu utilizatorul
- suferă schimbări frecvente (legate de design)

Abordarea cu servleturi:

- dificil de dezvoltat și întreținut
- lipsa flexibilității și modularității
- lipsa separării rolurilor



Exemplu: HelloServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Hello extends HttpServlet {
    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws IOException {
        response.setContentType("text/html");
        PrintWriter out = new PrintWriter(response.getWriter());
        out.println("<html>" +
            "<head><title>Bine ati venit!</title></head>" +
            "<body>" +
            "<h1><font color=\"red\"> Bine ati venit! </font></h1>" +
            "<h2>Data si ora curente:" + new java.util.Date() + " </h2>" +
            "</body>" + "</html>");
        out.close();
    }
}
```

Exemplu: hello.jsp

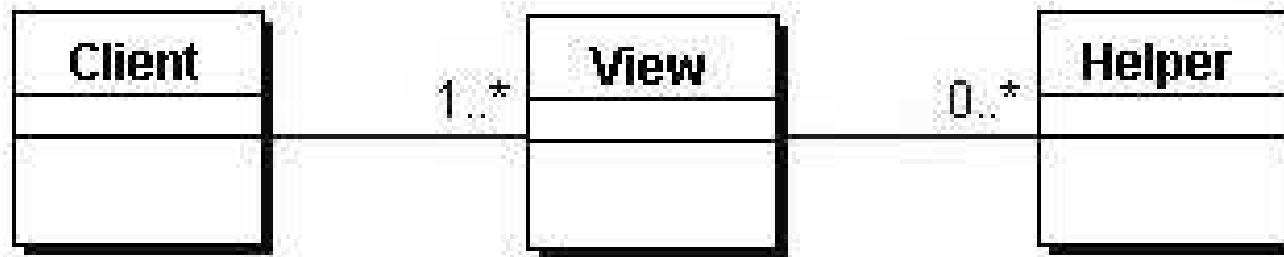
```
<html>

<head>
  <title>Bine ati venit!</title>
</head>

<body>
  <h1><font color="red"> Bine ati venit! </font></h1>
  <h2>Data si ora curente: <%= new java.util.Date() %> </h2>
</body>

</html>
```

View Helper



- View: text formatat de tip *șablon*
- Helper: bean-uri, taguri proprii, servleturi, etc.

Ce sunt paginile JSP ?

- Modalitate pentru generarea de conținut dinamic pe Web.
- Se bazează pe tehnologia servleturilor
- Strategia Java EE pentru generarea prezentării (View)
- Sunt formate din
 - tipar (componentă statică - HTML, XML)
 - taguri JSP
 - secvențe de cod (scriptleturi)

JSP vs. servleturi

Orice pagină JSP poate fi rescrisă sub forma unui servlet

JSP oferă însă unele avantaje:

- Metodă declarativă, orientată pe prezentare de creare a servleturilor
- Separarea conținutului static (HTML) de cel dinamic
- Sunt mai comod de scris
- Nu trebuie compilate explicit
- Sunt mai ușor de instalat pe server

Mai folosim servleturi ?

DA

Servleturile vor implementa componente care extind functionalitatea serverului Web

- autentificare
- control
- conectare la baze de date, etc.

Paginile JSP vor fi folosite pentru generarea de conținut dinamic specific aplicației

Arhitectura JSP

Ciclul de viață al unei pagini

1. **Translația**

Pagina JSP este transformată într-un servlet.

`hello.jsp` \mapsto `hello_jsp.java`

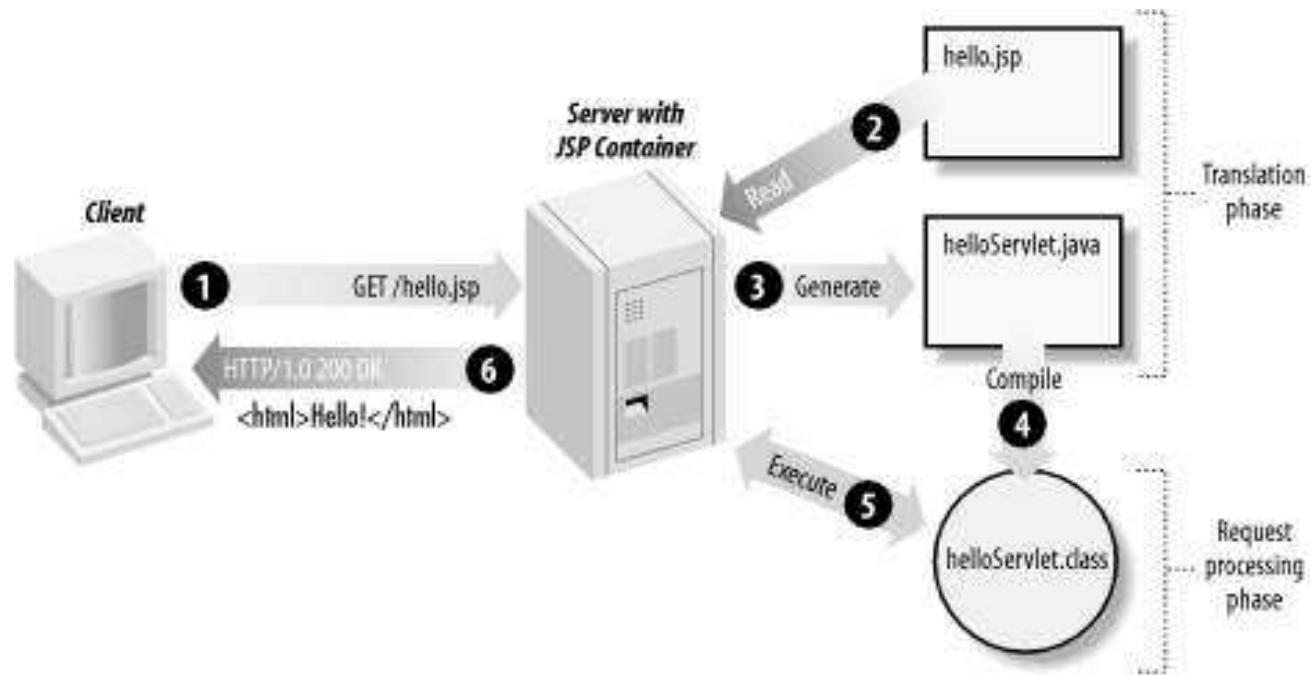
2. **Compilarea**

Servletul este compilat.

`hello_jsp.java` \mapsto `hello_jsp.class`

3. **Execuția**

Cererile către pagina JSP sunt direcționate către servlet, care va da răspunsul.



Translația

Orice pagină JSP este transformată într-un servlet printr-un procedeu numit **translație**. Translația se desfășoară automat:

- la prima solicitare venită către pagina JSP respectivă
- dacă pagina JSP a fost modificată

Servletul generat extinde clasa **HttpJspBase** și definește metodele `jspInit`, `_jspService`, `jspDestroy`.

Structura clasei generate

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class hello_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 {
        ...
        out.write("<HTML>");
        ...
        out.write("</HTML>");
    }
}
```

Sintaxa JSP

Elementele constitutive

- **Tiparul**
Reprezintă conținutul static al documentului, comentarii, etc.
- **Directive**
Instrucțiuni care specifică cum trebuie construită pagina, ce alte documente trebuie incluse, etc.
- **Elemente scriptice:** declarații, expresii, scriptleturi
Includ cod Java în pagina JSP.
- **Acțiuni**
Furnizează unei pagini JSP funcționalitate de nivel înalt.

Sintaxa elementelor

Element	Sintaxa %	Sintaxa XML
Comentarii vizibile	<code><!-- vizibil --></code>	<code><!-- vizibil --></code>
Comentarii ascunse	<code><%-- ascuns --%></code>	<code><%-- ascuns --%></code>
Declarații	<code><%! declaratii Java %></code>	<code><jsp:declaration></code>
Expresii	<code><%= expresie Java %></code>	<code><jsp:expression></code>
Scriptleturi	<code><% instructiuni Java %></code>	<code><jsp:scriptlet></code>
Directive	<code><%@ ... %></code>	<code><jsp:directive.tip ... /></code>
Acțiuni		<code><jsp:actiune></code>

Tiparul

- Reprezintă **tot ce nu este element JSP** dintr-o pagină JSP.
- Va apărea **nealterat** in orice document generat.
- Uzual, tiparul reprezintă **conținutul static** al documentului.

Pagina JSP de mai jos este formată doar din tipar.

```
<html>  
Hello world !  
</html>
```

Comentarii

- **Comentarii HTML**
- **Comentarii proprii JSP**
- **Comentarii Java**
incluse în elementele scriptice ale paginii

```
<html>
Hello world !
<!-- comentariu vizibil: Have a nice day ! -->
<%-- comentariu ascuns : Have a bad day... --%>
<jsp:scriptlet>
  // comentariu Java
  /* urmeaza un bloc de instructiuni Java
     sau nu... */
</jsp:scriptlet>
</html>
```

Directive

- Furnizează informații despre pagina respectivă.
- Controlează crearea servletului.
- Cele mai utilizate directive sunt:
 - **page**: importul unor pachete Java
 - **include**: includerea altor documente
 - **taglib**: accesarea unor librării de taguri proprii.

Sintaxa este:

`<%@ directiva [...] %>`

sau

`<jsp:directive.directiva [...] />`

Directiva page

Furnizează informații despre crearea paginii:

```
<jsp:directive.page [atribut="valoare" ...] />
```

Atribut	Valoare	Implicit
import	pachete importate	java.lang.*, javax.servlet.http.*, javax.servlet.*, javax.servlet.jsp.*
session	true / false	true
autoFlush	true / false	true
isThreadSafe	true / false	true
info	text	nimic
errorPage	URL-ul paginii de eroare	nimic
isErrorPage	true / false	false
contentType	codificare MIME	text/html

Directiva include

Are ca efect includerea (inserarea) conținutului unui document specificat în rezultatul final ce va fi transmis solicitantului.

Includerea se va face la poziția la care se găsește directiva.

```
<jsp:directive.include file="nume-fisier" />
```

Exemple de directive

```
<%-- Directive de tip page --%>
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page import="java.sql.*, java.util.*" %>
<%@page import="app.dao.*, app.model.*, app.util.DBConnector" %>

<html>
<%-- Directive de tip include --%>
...
<%-- Includere continut static --%>
<%@page include="header.html" %>
...
<%-- Includere continut dinamic --%>
<%@page include="welcome.jsp" %>
...
</html>
```

Declarații

Permit **definirea de variabile/metode globale** în cadrul unei pagini JSP. Acestea vor fi **variabile/metode membre** ale clasei servletului generat.

Sintaxa este:

`<%! declaratii Java %>`

sau

`<jsp:declaration> declaratii Java </jsp:declaration>`

Exemple de declarații

```
<%! String s="abcd"; %>
```

```
<jsp:declaration>
```

```
  <%-- Declaratii de variabile --%>
```

```
  public static final int MAX = 100;
```

```
  int v[] = new int[MAX];
```

```
  java.util.ArrayList a;
```

```
  <%-- Declaratii de metode --%>
```

```
  public String salut(String nume) {
```

```
    return "Bine ai venit" + nume;
```

```
  }
```

```
  public void init() {
```

```
    for(int i=0; i<v.length; i++)
```

```
      v[i] = 0;
```

```
  }
```

```
</jsp:declaration>
```

Expresii

Se referă la **expresii Java** ce vor fi evaluate în momentul apelării paginii JSP, iar rezultatul inclus direct în rezultat. Sintaxa este:

`<%= expresie Java %>`

sau

`<jsp:expression> expresie Java </jsp:expression>`

```
Data de astazi: <%= new java.util.Date() %>
Vectorul are: <%= v.length %> elemente.
<!--
  Pagina generata dinamic la ora:
    <%= new java.util.Date() %>
-->
```

Scriptleturi

Sunt **secvențe de cod Java** care contribuie la realizarea unui flux logic. Sintaxa este:

```
<% bloc de instrucțiuni Java %>
```

sau

```
<jsp:scriptlet> bloc Java </jsp:scriptlet>
```

Scriptleturile pot fi combinate cu conținutul static al paginii JSP:

```
<% for (int i=1; i<=4; i++) { %>  
    <H<%=i%>> Hello </H<%=i%>>  
<% } %>
```

Exemplu de scriptlet

```
<%@page import="java.sql.*, java.util.*" %>
<html> <body>
  <%
    // Primum id-ul unui film ca parametru al cererii
    int movieId = Integer.parseInt(request.getParameter("movieId"));
    DBConnector.setDriver("org.postgresql.Driver");
    DBConnector.setUrl("jdbc:postgresql://localhost/movies");
    DBConnector.setUser("postgres");
    Connection conn = DBConnector.getConnection();
    // aflam numele filmului
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(
      "select name from movies where id=" + movieId);
    rs.next();
    String movieName = rs.getString(1);
    rs.close();
  %>
  <!-- Variabile movieName este vizibila in restul paginii -->
  Numele filmului este: <b> <%= movieName %> </b>
</body> </html>
```

Obiectele unei pagini JSP

Obiecte proprii

```
<html>
Obiectul este vizibil aici: <%= s%>
<jsp:declaration>
    String s = "Hello";
</jsp:declaration>
...dar si aici: <%= s%>
</html>
```

```
<html>
<!-- Obiectul sb nu este vizibil aici --%>
<jsp:scriptlet>
    StringBuffer sb = new StringBuffer("Hello");
    sb.append(" world!");
</jsp:scriptlet>
Obiectul este vizibil: <%= sb.toString() %>
</html>
```

Domenii de vizibilitate

- **Aplicație**

Obiecte accesibile paginilor din aceeași aplicație.

- **Sesiune**

Obiecte accesibile paginilor din aceeași sesiune cu cea în care au fost create.

- **Cerere**

Obiecte accesibile paginilor responsabile cu procesarea cererii.

- **Pagină**

Obiecte accesibile doar paginilor în care au fost create.

Obiecte implicite

Obiect	Tip	Vizibilitate
request	HttpServletRequest	Cerere
response	HttpServletResponse	Pagină
pageContext	PageContext	Pagină
application	ServletContext	Aplicație
out	JSPWriter	Pagină
config	ServletConfig	Pagină
page (this)	HttpJspPage	Pagină
session	HttpSession	Sesiune
exception	Throwable	Pagină

Comunicarea cu clienții

Parametrul nume din cerere:

```
<%= request.getParameter("nume") %>
```

Adresa IP client:

```
<%= request.getRemoteAddr() %>
```

Browser folosit de apelant:

```
<%= request.getHeader("user-agent") %>
```

```
<%
```

```
// Scriem date pe fluxul de iesire  
out.println("<h1> Hello </h1>");
```

```
// Putem redirecta cererea  
response.sendRedirect("http://www.infoiasi.ro/~acf");
```

```
// Putem trimite mesaje de eroare  
response.sendError(404);
```

```
%>
```

Tratarea cererilor concurente

Implicit, metoda `_jspService` este executată într-un **fir de execuție propriu**.

Putem însă impune ca servletul să implementeze interfață **SingleThreadModel**:

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

Sincronizarea accesului la attribute

```
<%  
synchronized (application) {  
    ObiectPartajat obj = (ObiectPartajat)  
        application.getAttribute("obiectPartajat");  
    application.setAttribute("obiectPartajat", obj);  
}  
%>
```

Tratarea excepțiilor

Redirectarea excepțiilor către o pagină JSP anume se face prin directiva:

```
<%@ errorPage="errorHandler.jsp" %>
```

Definirea unei pagini pentru tratarea excepțiilor:

```
<%@ isErrorPage="true" %>  
A aparut exceptia:  
<%= exception.toString() %> !
```

Folosirea sesiunilor

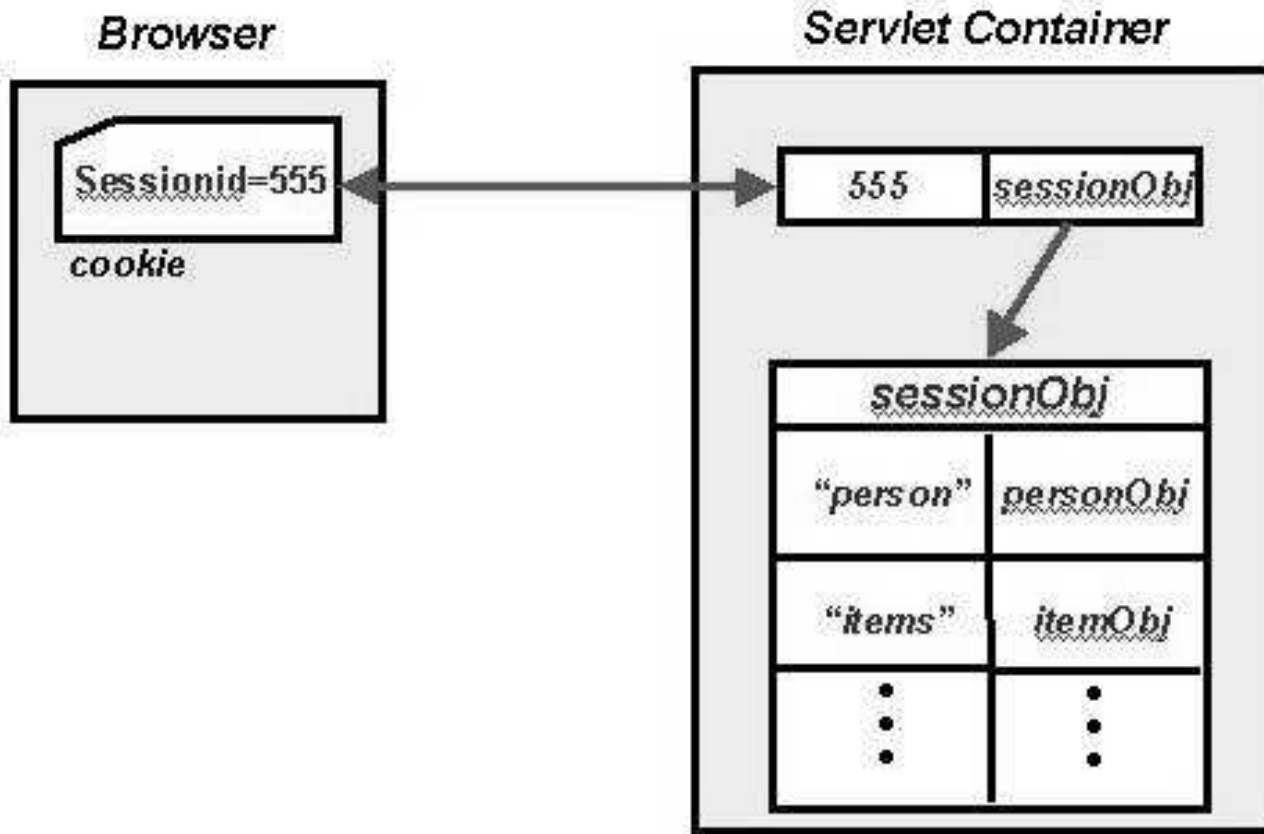
Implicit, toate paginile JSP participă la o sesiune HTTP. Obiectul corespunzător de tip `HttpSession` este **session**.

Intr-o sesiune se pot memora date sub forma **cheie-valoare**.

```
<%  
    MyClass obj = new MyClass();  
    session.setAttribute("cheie", obj);  
%>  
  
<%  
    MyClass obj = session.getAttribute("cheie");  
%>
```

Dacă nu dorim ca o pagină JSP să participe la o sesiune:

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



Acțiuni standard

Ce sunt acțiunile ?

Acțiunile permit definirea de taguri corespunzătoare unei anumite funcționalități. Sunt de două tipuri:

- **Standard** - predefinite
- **Proprii** - create de utilizator

Scop

- Minimizarea utilizării scriptlet-urilor
- Crearea de componente reutilizabile
- Dezvoltarea mai eficientă a aplicațiilor

Acțiunile standard

- Instanțierea unor componente JavaBean
<**jsp:useBean**>
- Comunicarea cu resursele serverului Web
 - <**jsp:forward**>
 - <**jsp:include**>

Componente JavaBeans

Componentele **JavaBeans** sunt:

- componente de sine stătătoare, reutilizabile, independente de platformă
- pot fi folosite pe orice platformă de programare
- sunt conforme cu o serie de specificații
- definesc un model pentru crearea de componente software.

Un **container** reprezintă un context în care componentele pot fi grupate și cu care acestea pot interacționa.

Componente JavaBeans

- Sunt descrise de **clasa publice**
- Constructorul fără argumente trebuie să existe și să fie public
- Fiecare proprietate gestionată trebuie să conțină **metode de accesare** de tipul
 - **getProprietate**
 - **setProprietate**

In arhitectura JSP, sunt folosite pentru **stocarea de informații** la un anumit nivel de vizibilitate.

Caracteristici JavaBeans

- **Proprietăți** - pentru individualizare si programare
- **Introspecție** - pentru analizarea componentei de catre container
- **Personalizare** - pentru modificarea aspectului si functionalitatii
- **Evenimente** - pentru comunicare
- **Persistență** - salvare / restaurare proprietăți

Crearea componentelor JavaBean

O componentă JavaBean va fi descrisă prin:

- **ID-ul său**
- **Clasa din care face parte**
- **Domeniul de vizibilitate**

```
<jsp:useBean id="user" class="Utilizator" scope="session" />
```

```
<jsp:useBean id="user" class="Utilizator" scope="session" >  
  //Cod Java de initializare  
  user.setNume("Ion");  
  ...  
</jsp:useBean>
```

Folosirea componentelor JavaBean

Un Bean va fi folosit prin setarea / obtinerea proprietatilor sale.

- **<jsp:setProperty>**

- **<jsp:getProperty>**

```
<jsp:useBean id="user" class="Utilizator" scope="session" />  
<jsp:setProperty name="user" property="nume" value="Vasile" />  
<jsp:getProperty name="user" property="nume" />
```

Bean-uri pentru formulare

Correspondență între numele proprietăților și numele câmpurilor din formular.

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

Procesul de redirectare formular-bean este realizat prin mecanismul **instrospecției**.

Maparea poate fi și explicită:

```
<jsp:setProperty name="user" property="nume" param="numePersoana" />
```

Exemplu: login.jsp

```
<%@ page import="test.*" %>

<jsp:useBean id="user" scope="session" class="test.Utilizator" />
<jsp:setProperty name="user" property="*" />

<% if ( request.getParameter("nume") == null ) { %>
    <%@ include file="formular.html" %>
<% } else {
    if (request.getParameter("parola").equals("jsp")) { %>
        <%@ include file="confirmare.jsp" %>
    <% } else { %>
        <%@ include file="refuz.jsp" %>
    <% }
} %>
```

Exemplu: formular.html

```
<html>
<head>
  <title>Autentificare</title>
</head>

<form method="get">
  Nume: <input type="text" name="nume" size="25" />
  <br/>
  Parola: <input type="password" name="parola" size="10" />
  <br/>
  <input type="submit" value="Confirmare" />
</form>
</html>
```

Exemplu: clasa Utilizator

```
package test;
public class Utilizator {
    private String nume;
    private String parola;

    public String getNume() {
        return nume;
    }
    public void setNume(String nume) {
        this.nume = nume;
    }
    public String getParola() {
        return parola;
    }
    public void setParola(String parola) {
        this.parola = parola;
    }
}
```

Exemplu: confirmare.jsp

```
<html>
<head>
  <title>Confirmare</title>
</head>

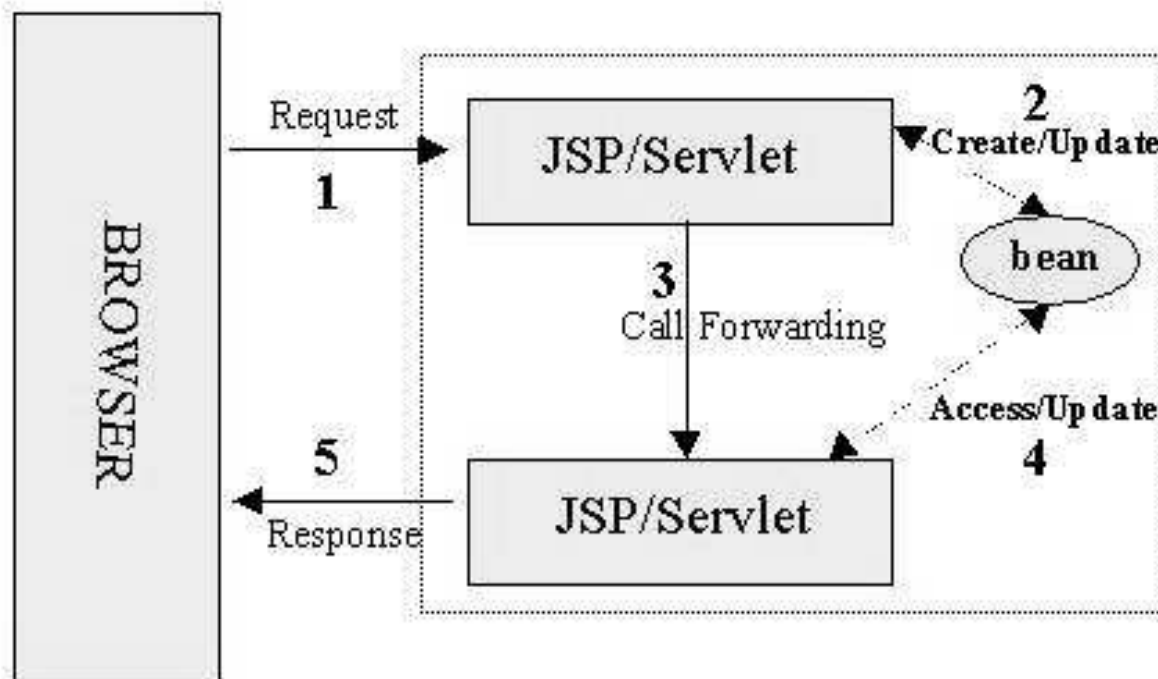
<h1>
Salut,
<jsp:getProperty name="user" property="nume" /> !
</h1>
</html>
```

Exemplu: refuz.jsp

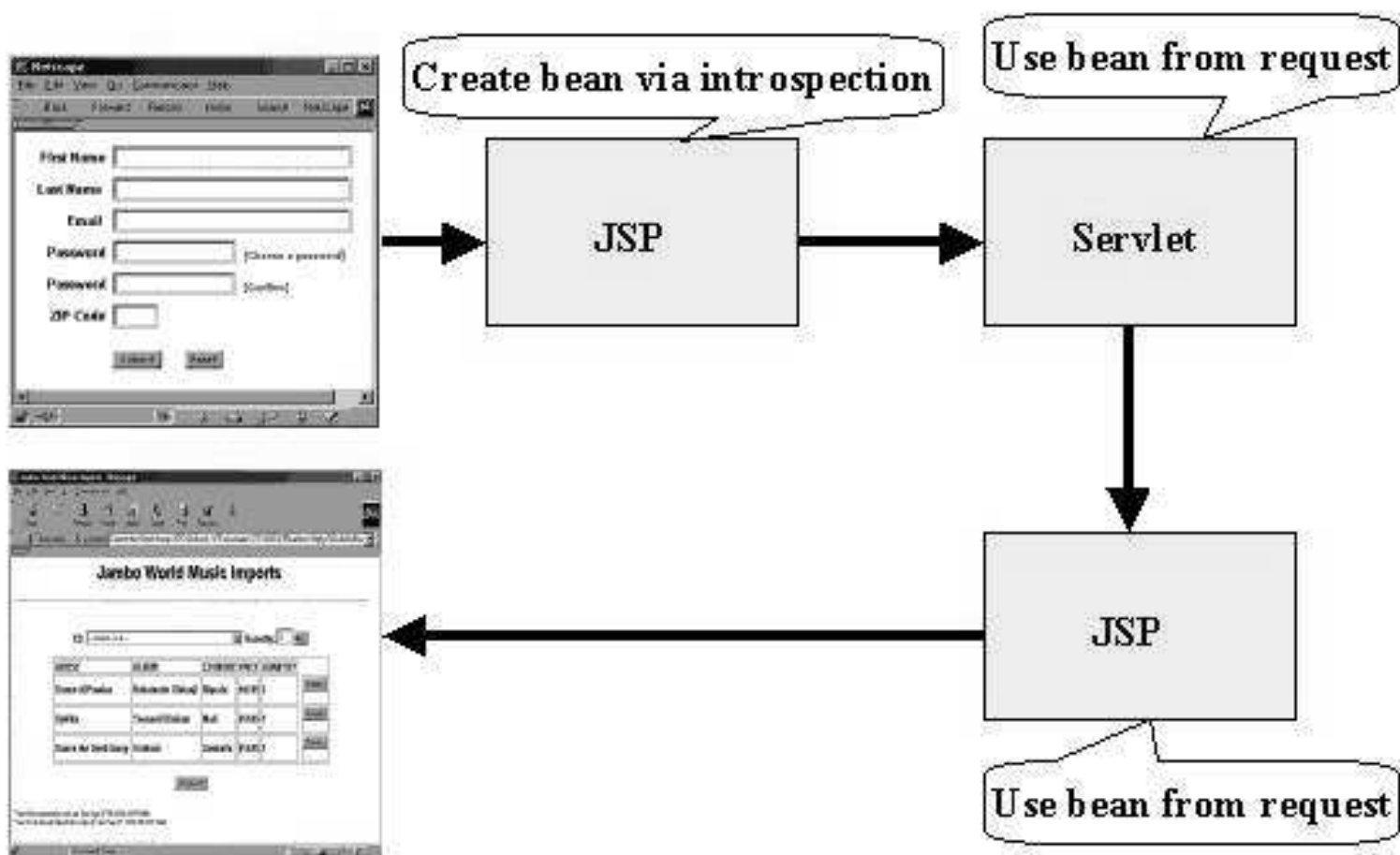
```
<html>
<head>
  <title>Eroare</title>
</head>
<h1>
Parola <jsp:getProperty name="user" property="parola" /> este invalida!
</h1>
</html>
```

Redirectarea cererilor: *forward*

```
<jsp:forward page="altaPagina.jsp" >  
  <jsp:param name="nume1" value="valoare1" />  
  <jsp:param name="nume2" value="valoare2" />  
</jsp:forward >
```



Inlantuirea cererilor



Inlantuirea cererilor (cont.)

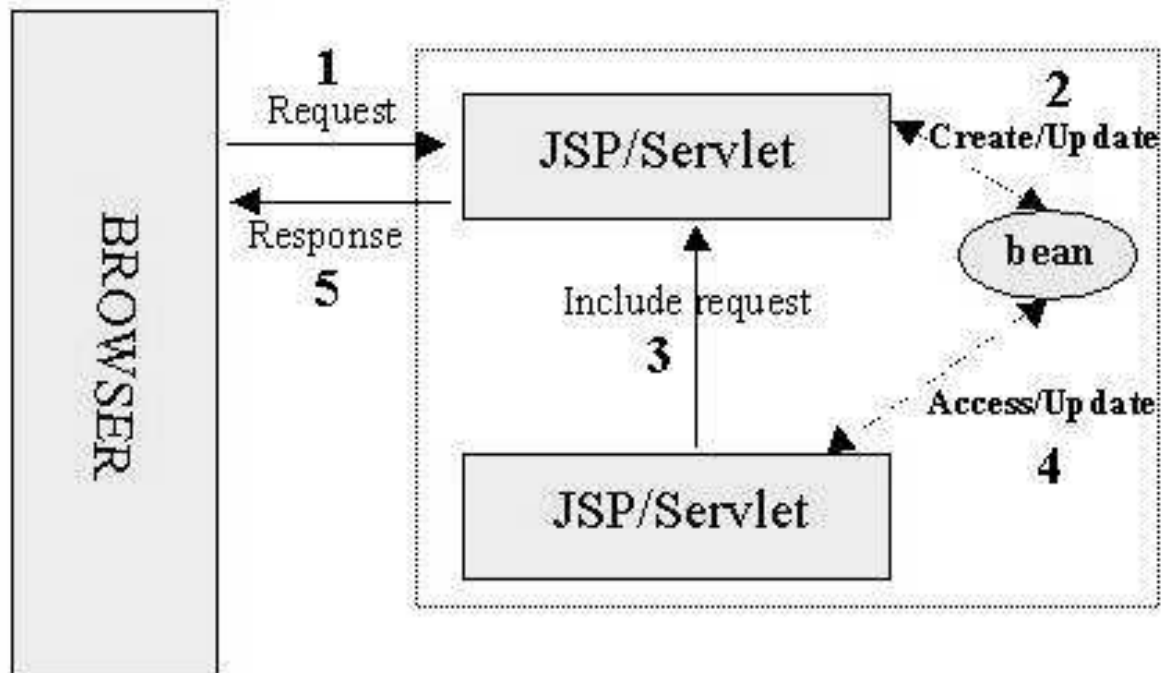
```
<jsp:useBean id="fBean" class="demo.FormBean" scope="request" />
<jsp:setProperty name="fBean" property="*" />
<jsp:forward page="/servlet/JSP2Servlet" />
```

```
public void doPost (HttpServletRequest req, HttpServletResponse res) {
    FormBean f = (FormBean) req.getAttribute("fBean");
    f.setName("..."); // actualizam starea componentei Bean
    getServletContext().getRequestDispatcher("Result.jsp")
        .forward(req, res);
}
```

```
<html>
<jsp:useBean id="fBean" class="demo.FormBean" scope="request" />
<jsp:getProperty name="fBean" property="name" />
</html>
```

Includerea altor rezultate: *include*

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



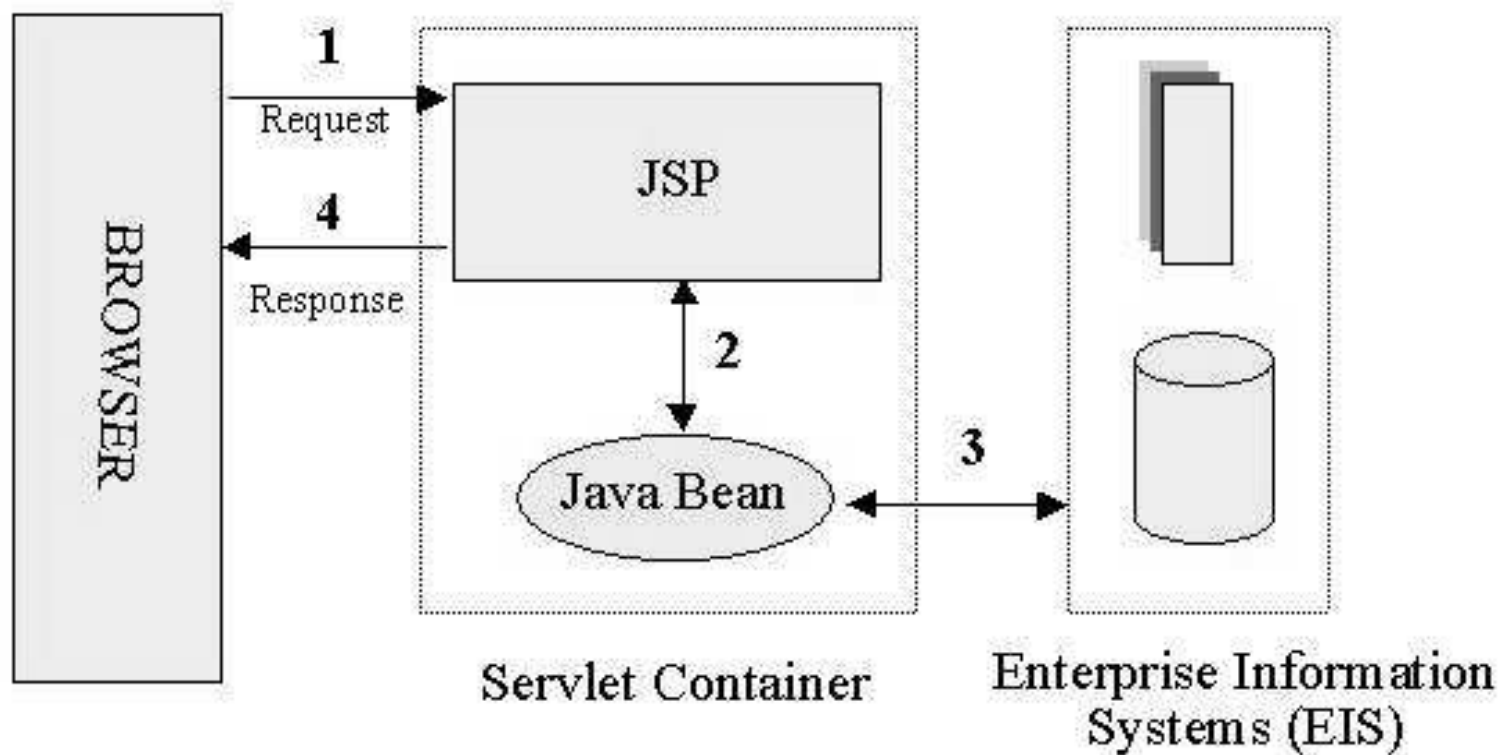
directiva include ≠ *acțiunea include*



Modele de acces



Modelul simplu



Model - View - Controller

