

JavaServer Pages (JSP)

JavaServer Pages 101

Instructor: Rick Palmer, SCWCD

rick@online-ettraining.com

Topics Covered

- ❑ JavaServer Pages (JSP) Overview
- ❑ JSP with JavaBeans
- ❑ Model 1 and Model 2 Design Patterns

JSP Introduction

- ❑ Servlets embed HTML inside of Java.
- ❑ JSP embeds Java or processing tags inside HTML.

```
<HTML>
  <BODY>
    <H2>Welcome <%= request.getParameter("username") %>.</H2>
    <P>Current time: <%= new java.util.Date() %></P>
  </BODY>
</HTML>
```

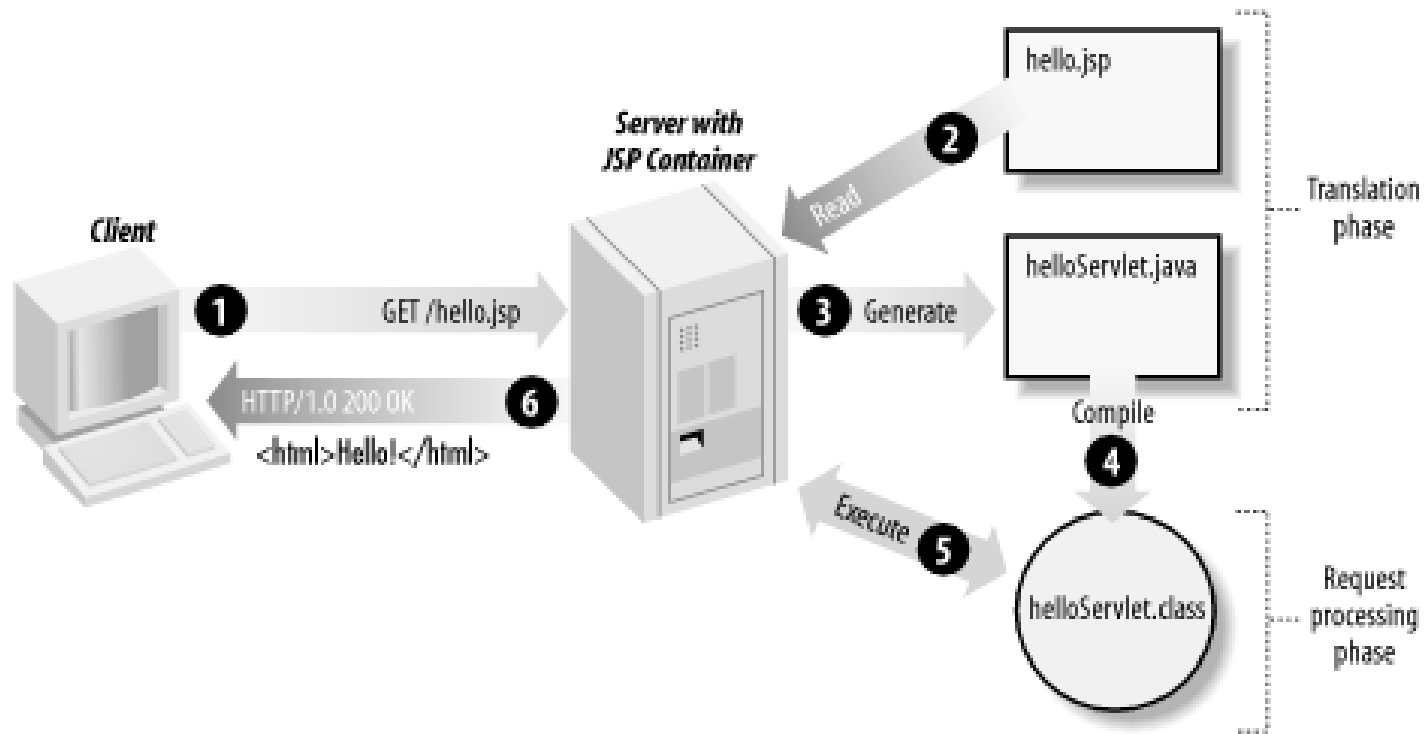
⇒ **No more** `out.println("<html>Hello</html>");`

⇒ Lets web designers focus on static HTML page design, and developers on dynamic Java content.

- ❑ Automatic compilation
- ❑ A compiled JSP is actually a servlet, with the same `service()` method and threaded behavior.

JSP Life Cycle

- ❑ JSPs are compiled initially by the web container (Translation Phase)
- ❑ JSPs are then used by the container to service requests (Request Processing Phase)



Predefined JSP Objects

- ❑ request – `HttpServletRequest`
 - ⇒ Determine request parameters from either GET/POST
- ❑ response – `HttpServletResponse`
 - ⇒ Set response content type and cookies
- ❑ out – `PrintWriter`
 - ⇒ Send buffered HTML output back to the browser
- ❑ session – `HttpSession` associated with the request
- ❑ application – `ServletContext` available to all servlets and JSP pages

JSP Page Directives

- Importing classes (HttpServletRequest and I/O classes are automatically imported):

```
<%@ page import="javax.xml.*, com.mypackage.*" %>
```

- Setting content type:

```
<%@ page contentType="text/html" %> (default)
```

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

- Turning sessions on/off for this page:

```
<%@ page session="true" %> (default)
```

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

JSP Scripting Elements

❑ Declarations `<%! variables or methods %>`

```
<%! private int m_iAccessCount = 0; %>
```

❑ Scriptlets `<% Java code %>`

```
<%
```

```
String strUserName = request.getParameter("username");
```

```
String strPassword = request.getParameter("password");
```

```
%>
```

❑ Expressions `<%= Java expressions %>`

```
<p>Welcome <%= strUserName %> </p>
```

```
<p>The current time is <%= new java.util.Date() %> </p>
```

JSP Include Directive/Tag

- Directive: includes the file at page compilation time

```
<%@ include file="Toolbar.jsp" %>
```

- ⇒ Includes the actual file itself, *before* the JSP has been converted to a servlet.
- ⇒ Changes to an included file are not picked up by its parent page, until the parent page changes.

- Tag: includes the file at request time

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

- ⇒ Server runs the included page and inserts its *output* at request time, and does this *for each request*.
- ⇒ Requires runtime processing – slower for dynamic include files.
- ⇒ Changes to included files are automatically picked up at request time, even if the parent file did not change.

JavaBeans Introduction

□ A JavaBean is a public Java class that meets the following requirements:

⇒ Constructor with no parameters.

⇒ Public get/set methods for privately stored properties:

```
private String m_strUserName = null;
```

```
public String getUsername();
```

```
public void setUsername(String);
```

⇒ Must reside in a *package* under WEB-INF/classes.

□ Why use a JavaBean?

⇒ Modularizes business logic, and separates it from display logic.

⇒ Allows transparent scaling of back-end architecture (e.g EJB).

⇒ Automatic property population reduces Java code.

Using JavaBeans with Servlets

- ❑ Create the JavaBean.

- ❑ Import the JavaBean so it's visible to the servlet.

```
import cis.AccountBean;
```

- ❑ Load the JavaBean and set its properties.

```
AccountBean objAccount = new AccountBean();
```

```
objAccount.setUsername(strUserName);
```

```
objAccount.setPassword(strPassword);
```

- ❑ Store the JavaBean in the request (or session) so it can be used by other web components.

```
request.setAttribute("Account", objAccount);
```

```
RequestDispatcher reqDispatcher =
```

```
    getServletContext().getRequestDispatcher("/Login.jsp");
```

```
reqDispatcher.forward(request, response);
```

Using JavaBeans with JSP

- ❑ Identify the JavaBean and its scope.

```
<jsp:useBean id="Account" class="cis.AccountBean" scope="request" />
```

- ❑ Read JavaBean properties, or method return values

```
<jsp:getProperty name="Account" property="userName" />
```

- ❑ Set JavaBean properties directly

```
<jsp:setProperty name="Account"
    property="userName" value="Greg" />
```

- ❑ Set JavaBean properties using request input parameters

```
<jsp:setProperty name="Account" property="userName" />
```

- ❑ Populate all JavaBean properties with input parameters

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

Data Type Conversions

- The J2EE container converts request parameter values (Strings) to commonly used Java data types:

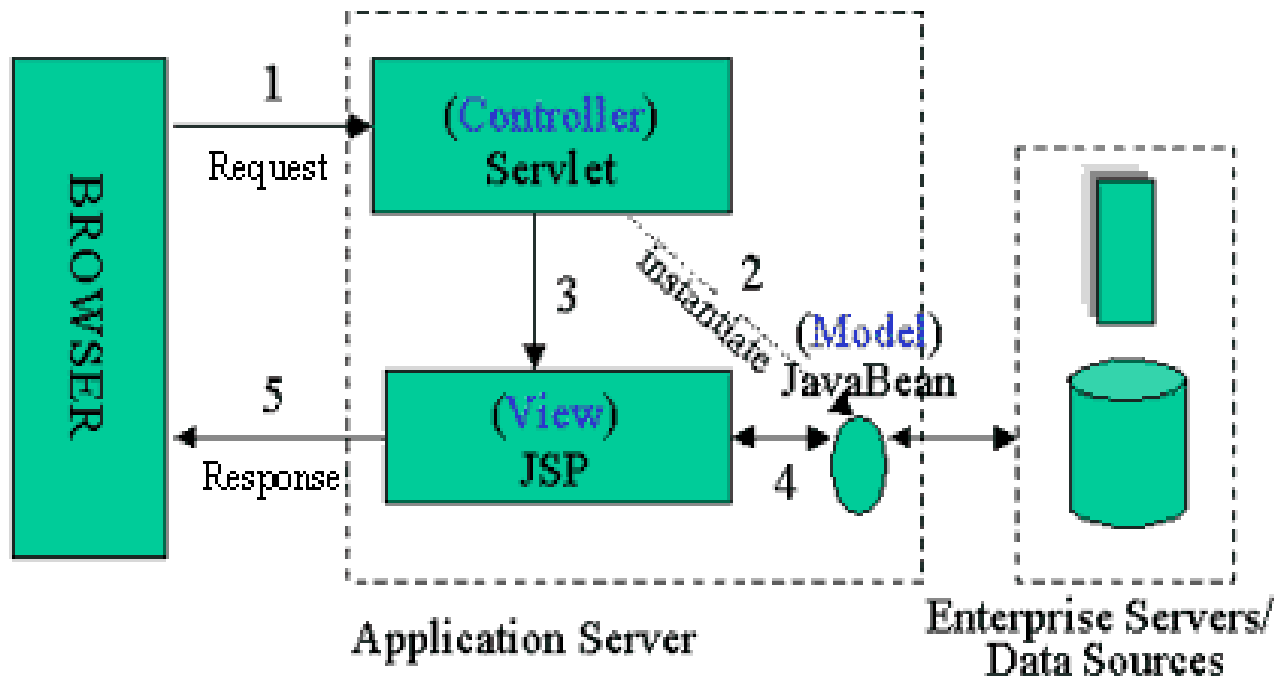
Property Type	Conversion Method
<code>boolean</code> or Boolean	<code>Boolean.valueOf(String)</code>
<code>byte</code> or Byte	<code>Byte.valueOf(String)</code>
<code>char</code> or Character	<code>String.charAt(int)</code>
<code>double</code> or Double	<code>Double.valueOf(String)</code>
<code>int</code> or Integer	<code>Integer.valueOf(String)</code>
<code>float</code> or Float	<code>Float.valueOf(String)</code>
<code>long</code> or Long	<code>Long.valueOf(String)</code>

Design Patterns: Model 1

- ❑ Single Servlet or JSP page is responsible for
 - ⇒ processing the request
 - ⇒ performing business logic and/or database access
 - ⇒ building HTML output
- ❑ Fine for simple applications
- ❑ Doesn't cut it for enterprise applications
 - ⇒ Hard to scale without significant rewrite.
 - ⇒ Hard to maintain or enhance, especially if the display or business logic changes frequently.
 - ⇒ Hard to separate development tasks.

Design Patterns: Model 2 (MVC)

- ❑ JavaBeans handle business logic (model).
- ❑ JSPs get data from JavaBeans and build HTML (view).
- ❑ Servlets process the request (controller).



Model 2 Architecture (M)

□ JavaBean (model):

- ⇒ Move business logic from servlet or JSP page into a JavaBean.
- ⇒ Create public get and set methods that correspond to each expected request parameter (e.g. `getUserName()` and `setUserName(String UserName)`).
- ⇒ Create methods that handle business logic, such as querying the database to get an account balance.

Model 2 Architecture (C)

□ Servlet (controller):

- ⇒ Get each request parameter.
- ⇒ Load the JavaBean and set its properties by calling the appropriate methods – `setUserName()`, `set...()`
- ⇒ Store the JavaBean as an attribute of the request.
`request.setAttribute("Account", objAccount);`
- ⇒ Forward the request to a JSP page for display.

```
RequestDispatcher reqDispatcher =  
    getServletContext().getRequestDispatcher("/Login.jsp");  
reqDispatcher.forward(request, response);
```

Model 2 Architecture (V)

□ JSP (view):

⇒ Get the JavaBean from the request.

```
<jsp:useBean id="Account" scope="request"  
            class="cis.AccountBean" />
```

⇒ Insert the JavaBean's properties as dynamic values within static HTML template text.

```
<H2>Welcome <jsp:getProperty  
            name="Account" property="userName" /> </H2>
```